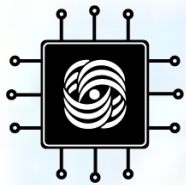


ИНФОРМАЦИОННО-УПРАВЛЯЮЩИЕ СИСТЕМЫ РЕАЛЬНОГО ВРЕМЕНИ

Лекция 1: *Введение в ИУС РВ*

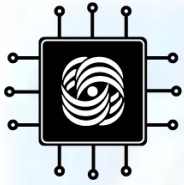
Кафедра АСВК,
Лаборатория Вычислительных Комплексов
Балашов В.В.



ИУС РВ

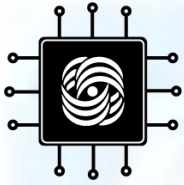
Сложные технические объекты
управляются распределёнными ВС





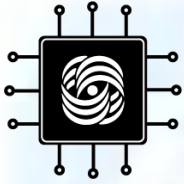
ИУС РВ

- Информационно-управляющая система (ИУС) – вычислительная система верхнего уровня, обеспечивающая:
 - функциональную и информационную интеграцию составных частей управляемого объекта
 - взаимодействие между объектом и оператором
- Функционирует в реальном времени
 - рассчитать результат *правильно* и *вовремя*



Содержание курса

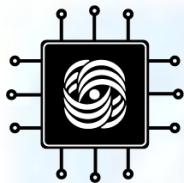
- Что есть:
 - принципы функционирования
 - архитектура аппаратной и программной части
 - основные технические решения
 - подходы к разработке и отладке
 - математические задачи и методы их решения
- Чего нет:
 - конкретные API
 - подробности протоколов
 - наборы команд процессоров
 - ...RTFM



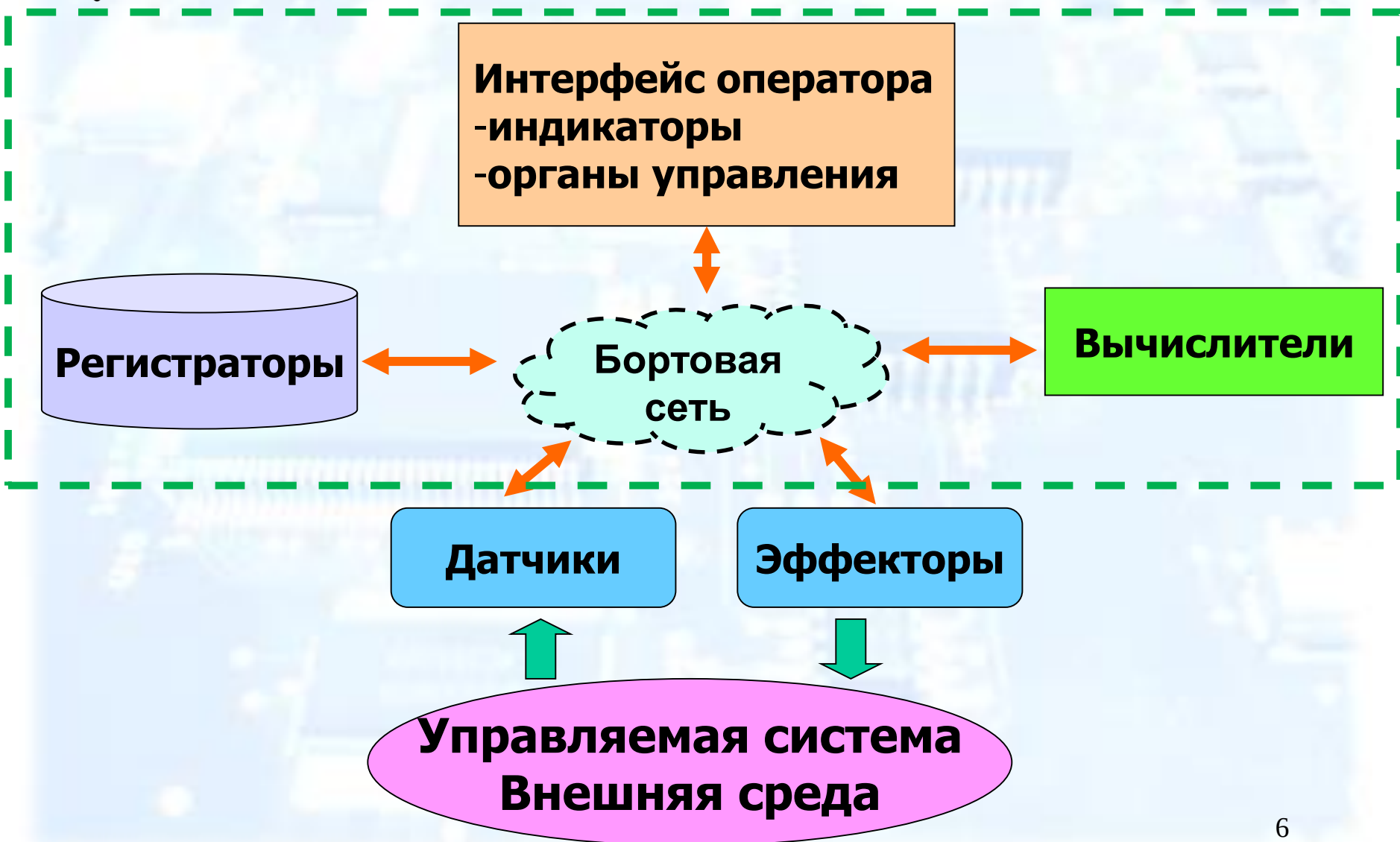
Где это пригодится

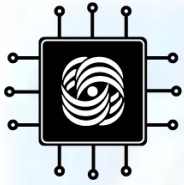
- Области:
 - автоматизация производства, энергетика, наземный транспорт, авиация/космос, «умный» дом
- Задачи:
 - проектирование, разработка, тестирование, сопровождение, модернизация
- Вклад:
 - создание средств решения задач
 - решение задач

Особенность: передний край



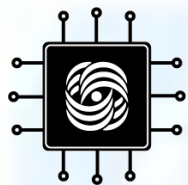
Состав ИУС



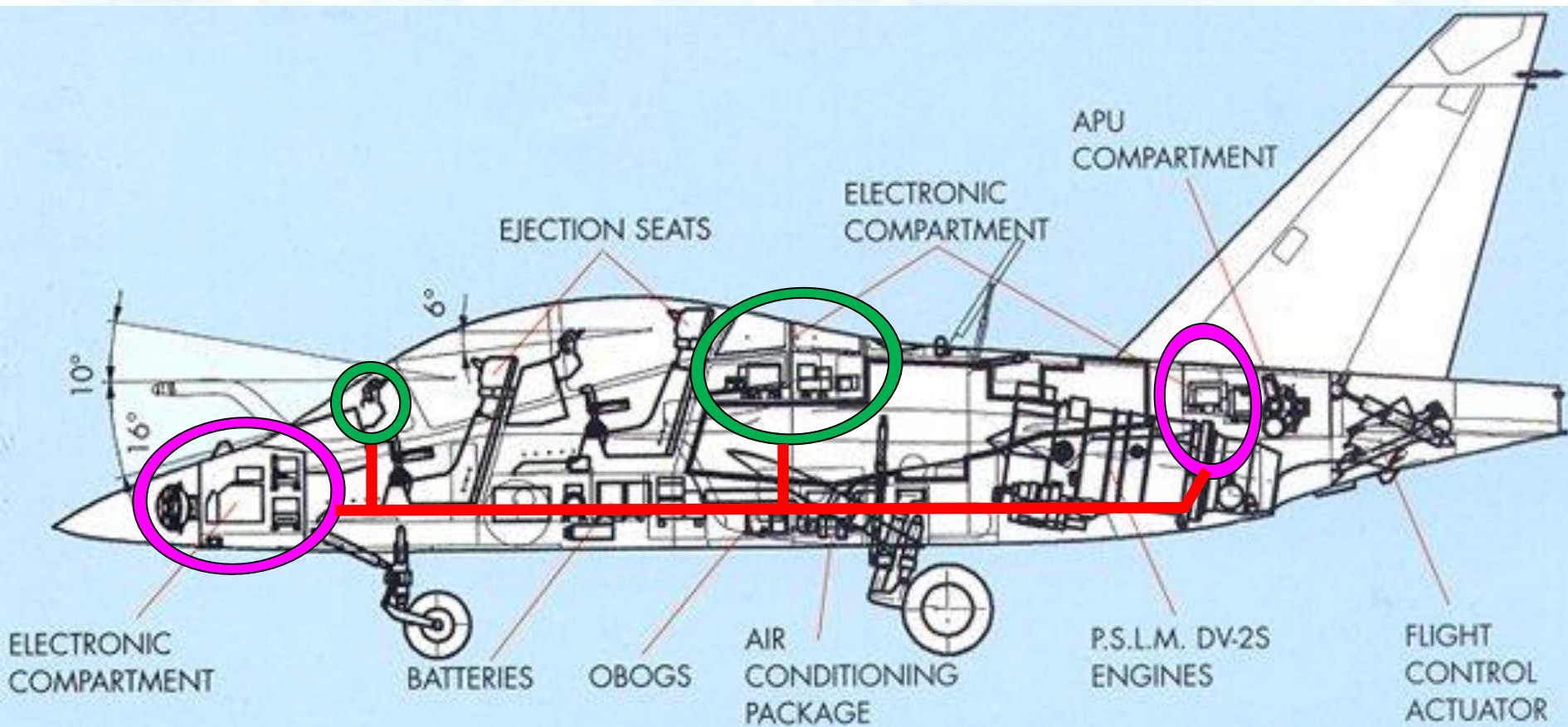


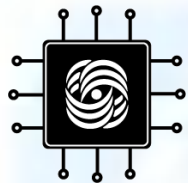
Функции ИУС

- Контроль состояния управляемого объекта
- Управление движением объекта или его частей
- Отслеживание положения объекта или его частей в пространстве
- Обмен данными с внешними системами
- Управление специализированными приборами (прикладной нагрузкой)
- Обмен данными с оператором
 - отображение данных
 - ввод данных



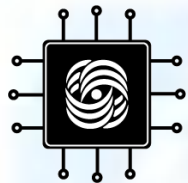
ИУС в управляемой системе



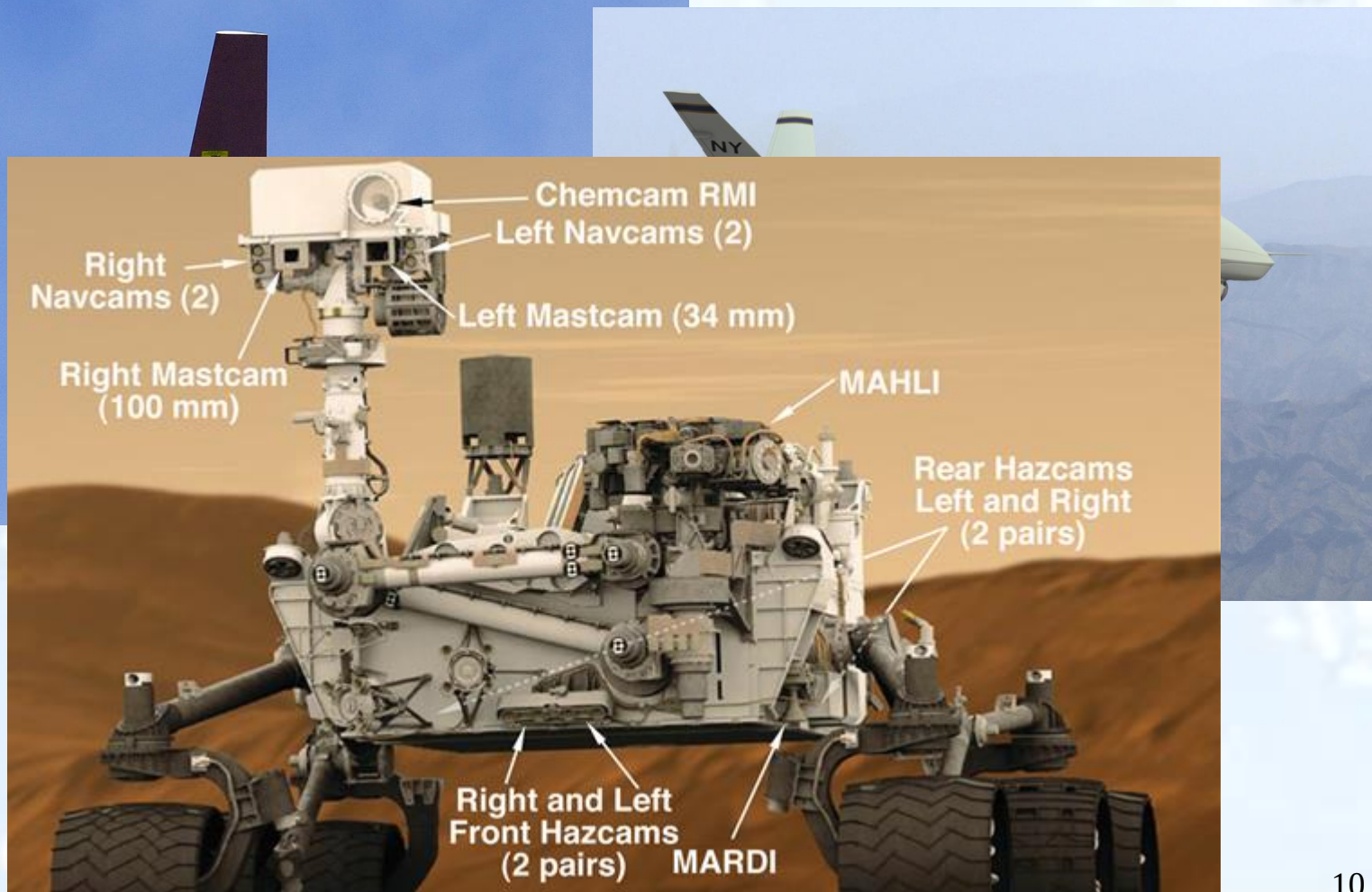


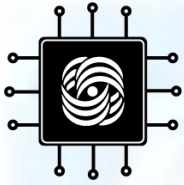
Устройства в составе ИУС





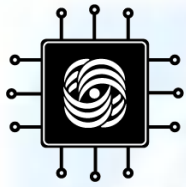
Прикладная нагрузка (вне ИУС)





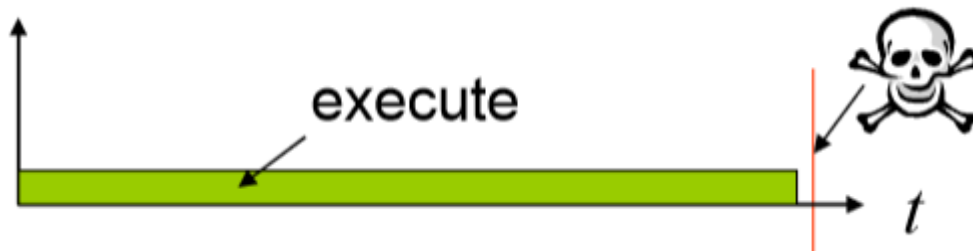
Специфика ИУС

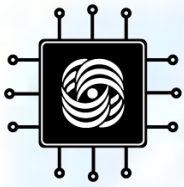
- Работа в реальном времени
 - ориентация на «наихудшие» случаи
- Непрерывное функционирование
- Параллелизм
 - управлять одновременно многим
- Интеграция с управляемой системой
- Критичность для управляемой системы
 - высокая цена ошибки
- Устойчивость к сбоям
- Ограниченное участие оператора
- Предсказуемое поведение
- «Экстремальные» условия работы
- Ограничения по ресурсам
- Координация между ИУС взаимодействующих объектов



Работа в реальном времени

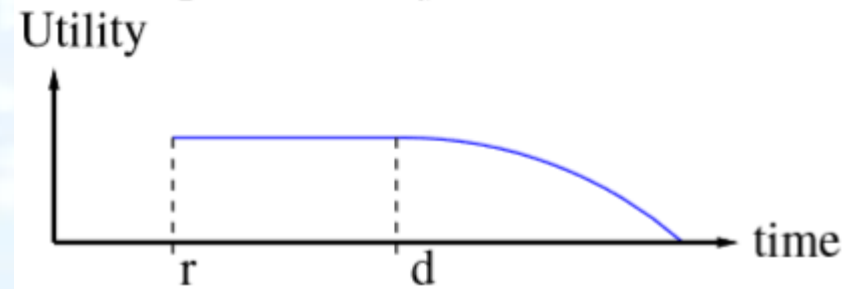
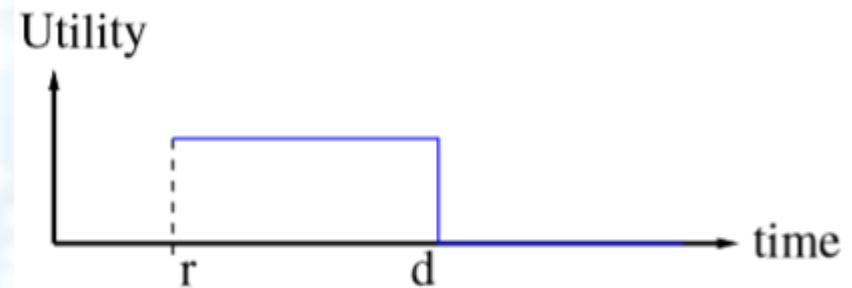
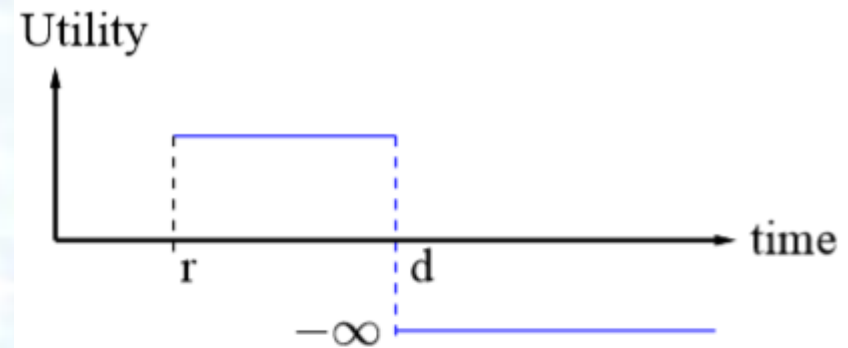
- Реагирующая система (reactive system) – ВС, функционирующая в постоянном взаимодействии с внешней средой и отвечающая на внешние воздействия в темпе, определяемом внешней средой
- Реакция на каждое воздействие должна укладываться в *директивный срок*

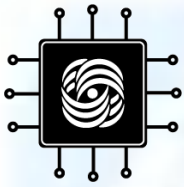




Градации требований реального времени

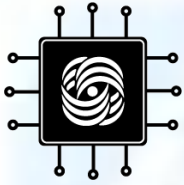
- Жёсткое (hard) РВ: нарушение ДС приводит к фатальным последствиям для управляемой системы (управление полётом)
- Промежуточный вариант (firm): нарушение ДС приводит к бесполезности результата, без фатальных последствий
- Мягкое (soft) РВ: нарушение ДС приводит к постепенному снижению ценности результата (автомобильный навигатор)





Распространённые заблуждения

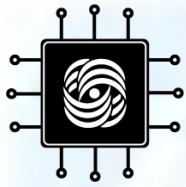
- Работа в реальном времени = быстрая работа
 - своевременность важнее быстродействия
 - предсказуемость и надёжность важнее быстродействия
- Рост производительности процессоров решит все проблемы с реальным временем
 - современные высокопроизводительные процессоры быстры в «среднем» случае, а для РВ критичен наихудший случай
 - тонкая технология производства => ненадёжность в экстремальных условиях
 - источники быстродействия современных процессоров слишком непредсказуемы
- Бессмысленно говорить о работе в реальном времени, если аппаратура может дать сбой
 - постепенная деградация функциональности
 - реконфигурируемость, «сбойные» режимы
- Разработка систем реального времени – чистая инженерия, здесь нет науки
 - вот и посмотрим...



Примеры ИУС РВ

Functions by embedded processing:

- ABS: Anti-lock braking systems
 - ESP: Electronic stability control
 - Airbags
 - Efficient automatic gearboxes
 - Theft prevention with smart keys
 - Blind-angle alert systems
 - ... etc ...
- 
- © P. Marwedel, 2011
- Multiple networks
 - Multiple networked processors

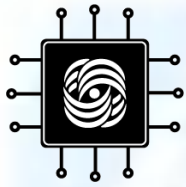


Примеры ИУС РВ

- Flight control systems,
- anti-collision systems,
- pilot information systems,
- power supply system,
- flap control system,
- entertainment system,
- ...



© P. Marvedel, 2011



Примеры ИУС РВ



Networked computer system

- Controlling arms & tools
- Navigating the forest
- Recording the trees harvested
- Crucial to efficient work

“Tough enough to be out in the woods”



産業用 ICS

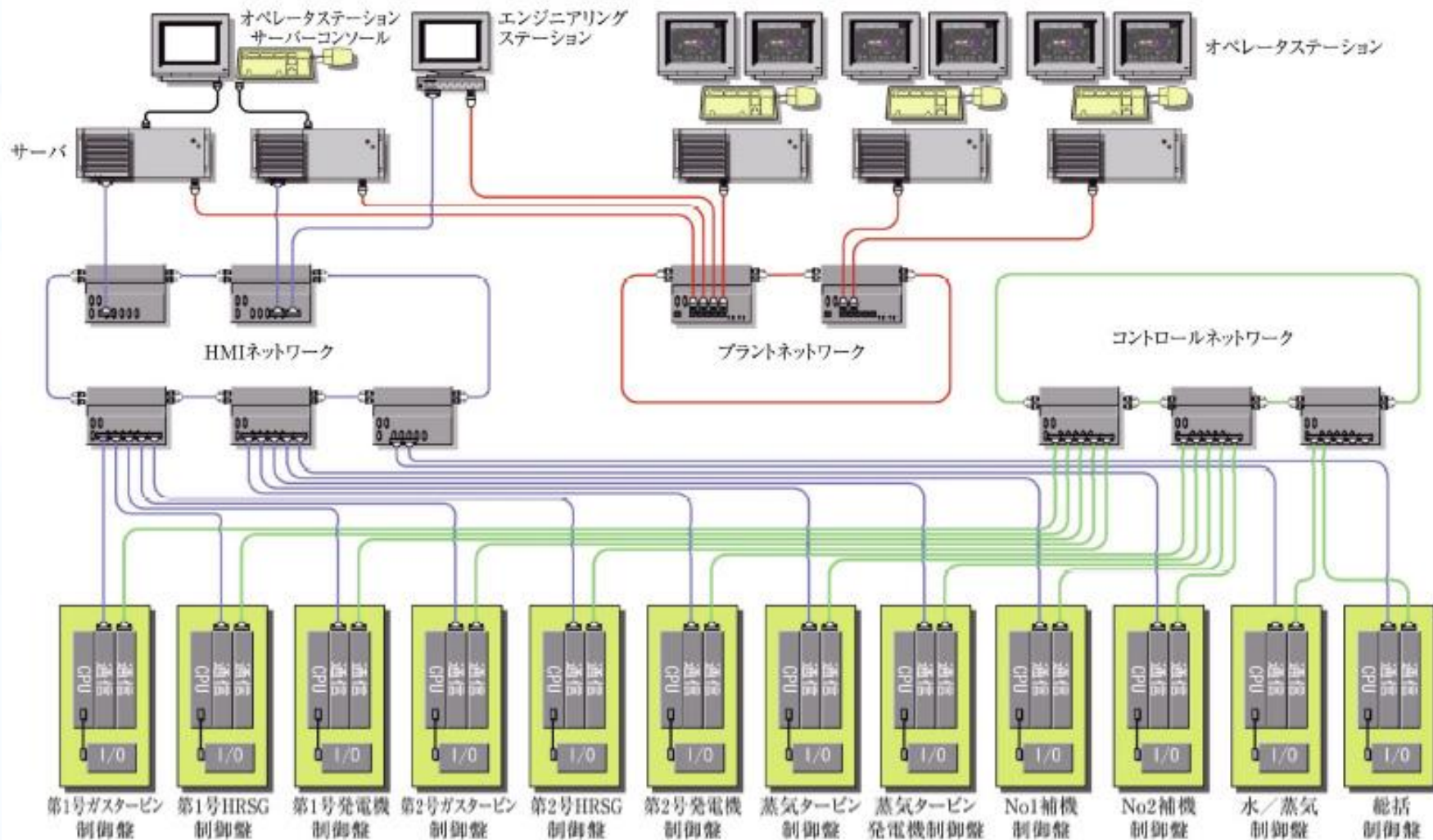
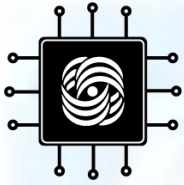


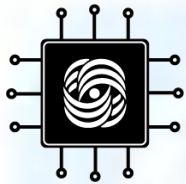
図2 中型発電制御装置のシステム構成例

Fig.2 System configuration of medium class gas turbine power plant



Эволюция ИУС

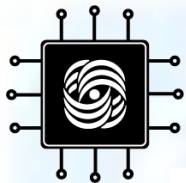
- Полностью аналоговая система
- Центральный вычислитель + аналоговые устройства
- Федеративная архитектура
 - медленные каналы связи (< 1 Мбит/с)
 - специализированные вычислители
 - локальная обработка данных
- Интегрированная модульная архитектура
 - «облако» вычислительных модулей
 - быстрые каналы связи
 - виртуализация сетевых и вычислительных ресурсов



Неоднородность ИУС

- Каналы: точка-точка, шина, коммутатор; 12 kbps, 1 Mbps, 1 Gbps
- Устройства: датчики, индикаторы, вычислители, органы управления, исполнительные устройства
- Данные: аналоговые, цифровые; числовые массивы, видеопотоки

Проблема унаследованных устройств



ИСТРЕБИТЕЛЬ 5 ПОКОЛЕНИЯ

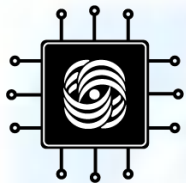
T-50 PAK FA



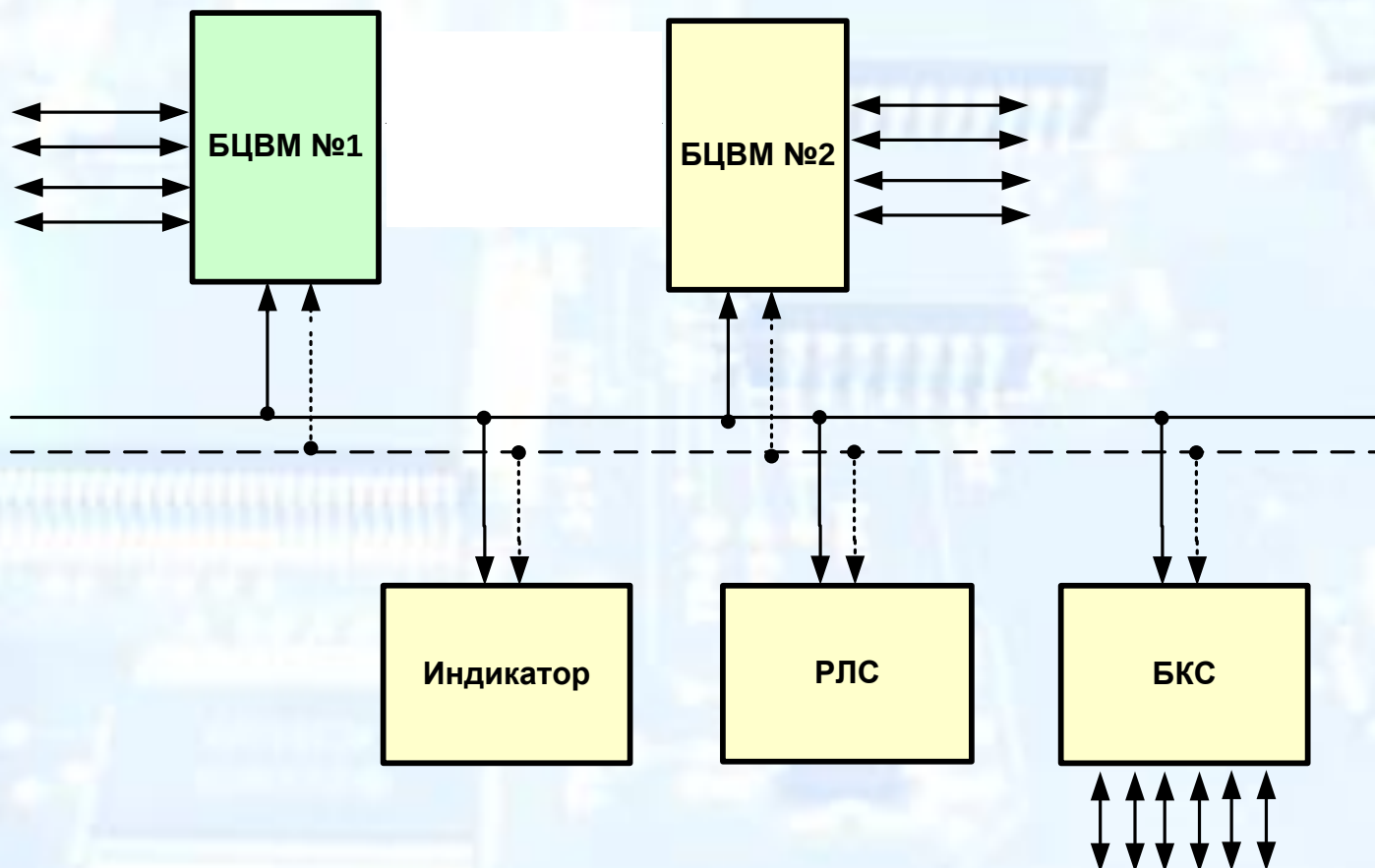
F22A

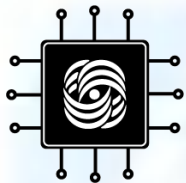


0 1 2 3 4 5m
25 pixel = 1 m

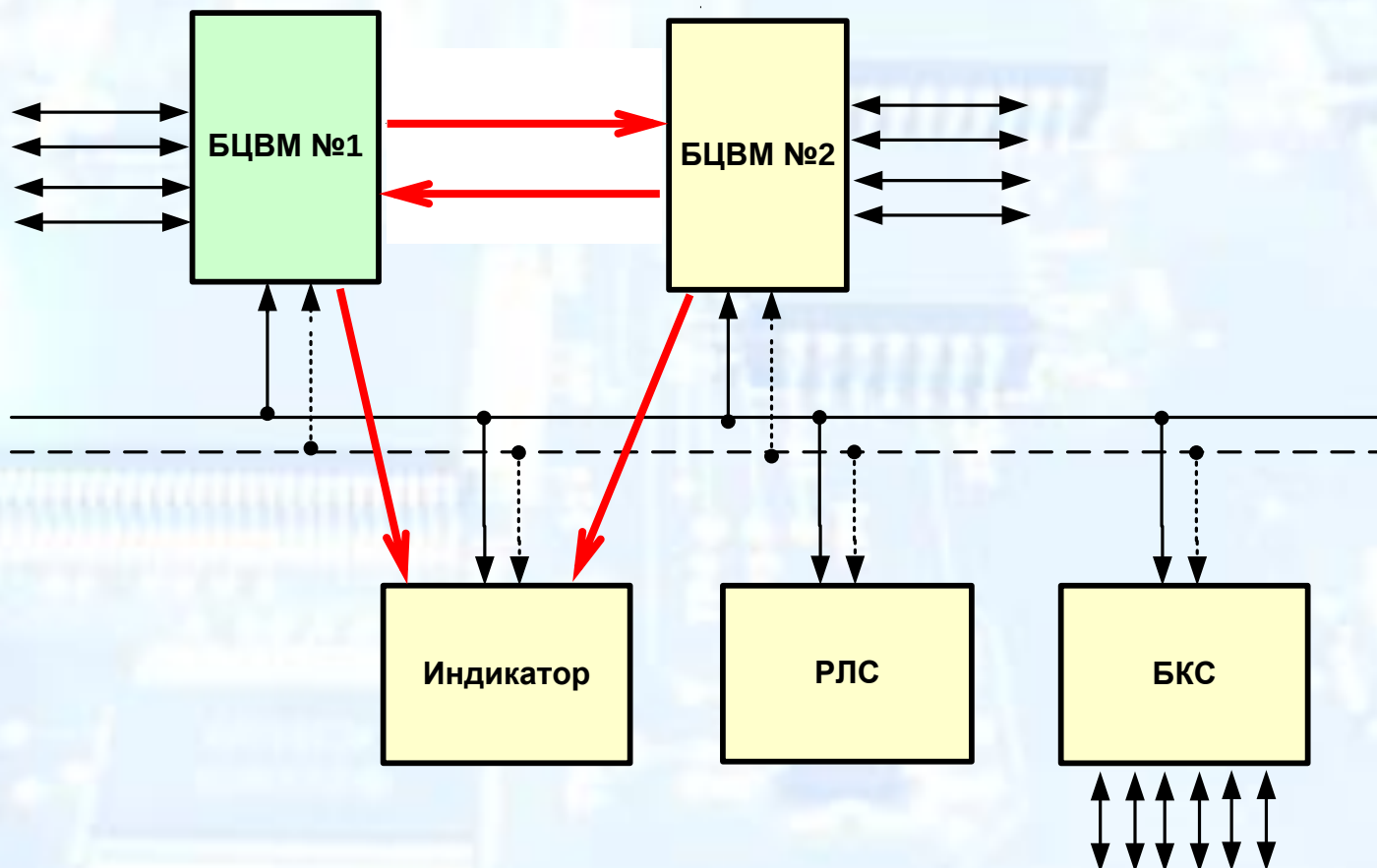


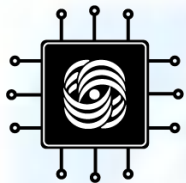
Унаследованная архитектура (4 поколение)



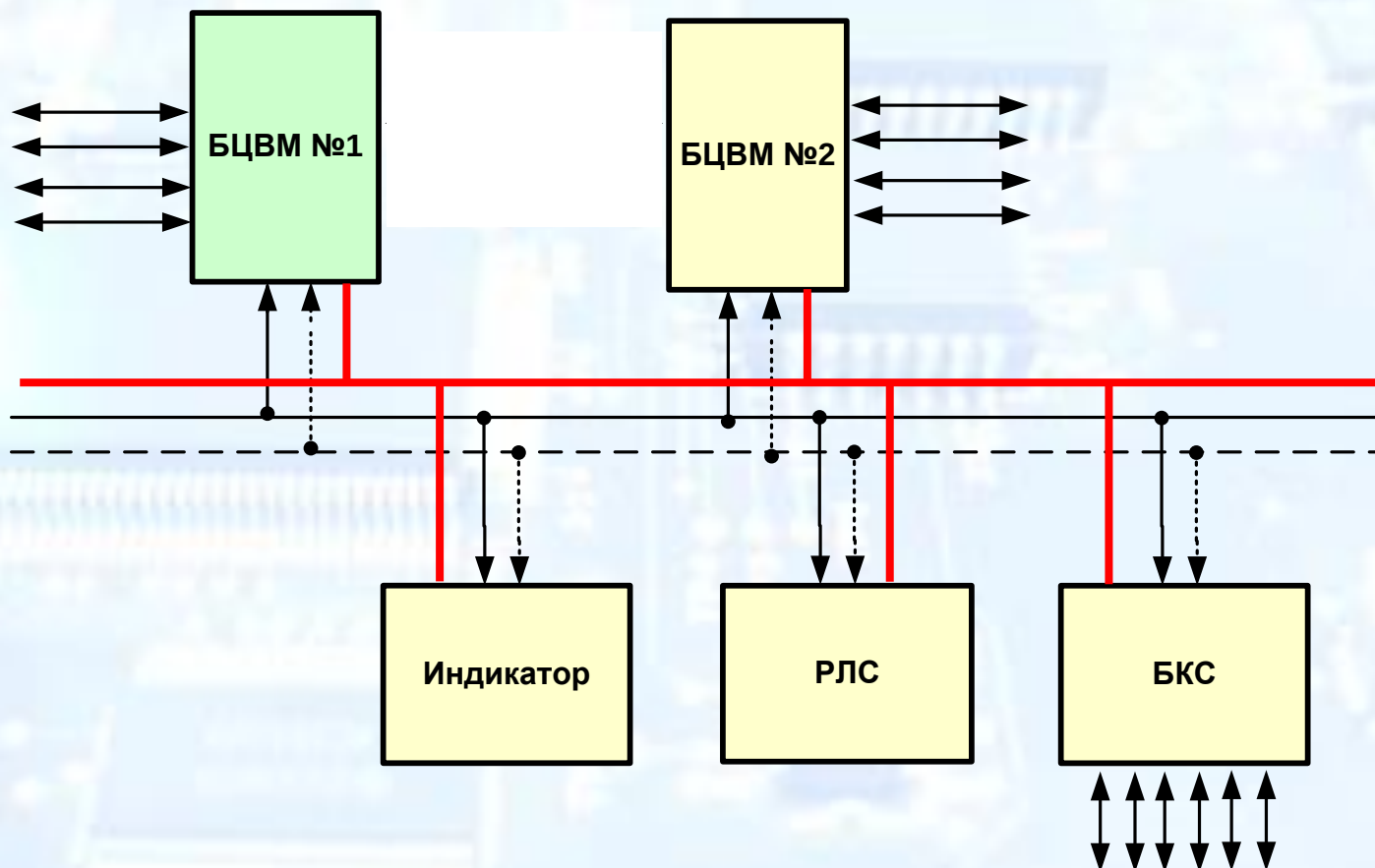


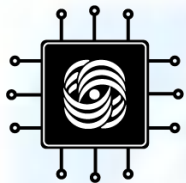
Оптические каналы точка-точка (4+)





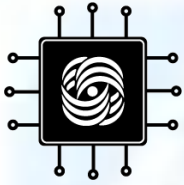
Оптическая магистраль данных (4+)





Интегрированная модульная авионика (5 поколение)

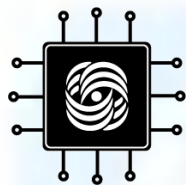




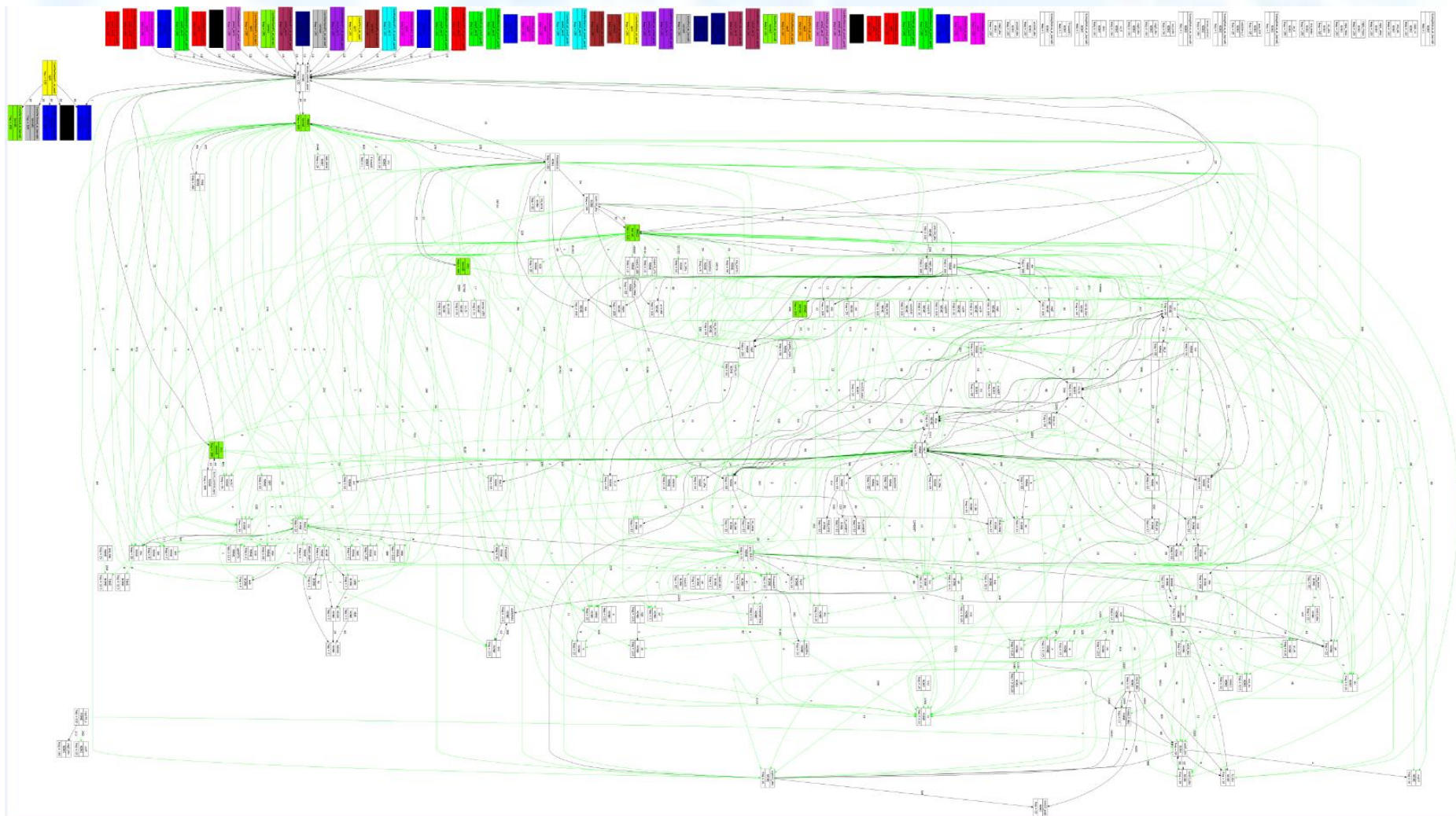
Рост сложности ПО

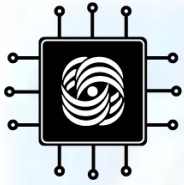
Year	Size
1986	10 KB
1992	100 kB
1998	1 MB
2008	15 MB

- Управляющее ПО телевизора
- Экспоненциальный рост
- ПО ИУС РВ – те же темпы, критичность выше



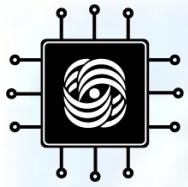
ПО БЦВМ Т-50



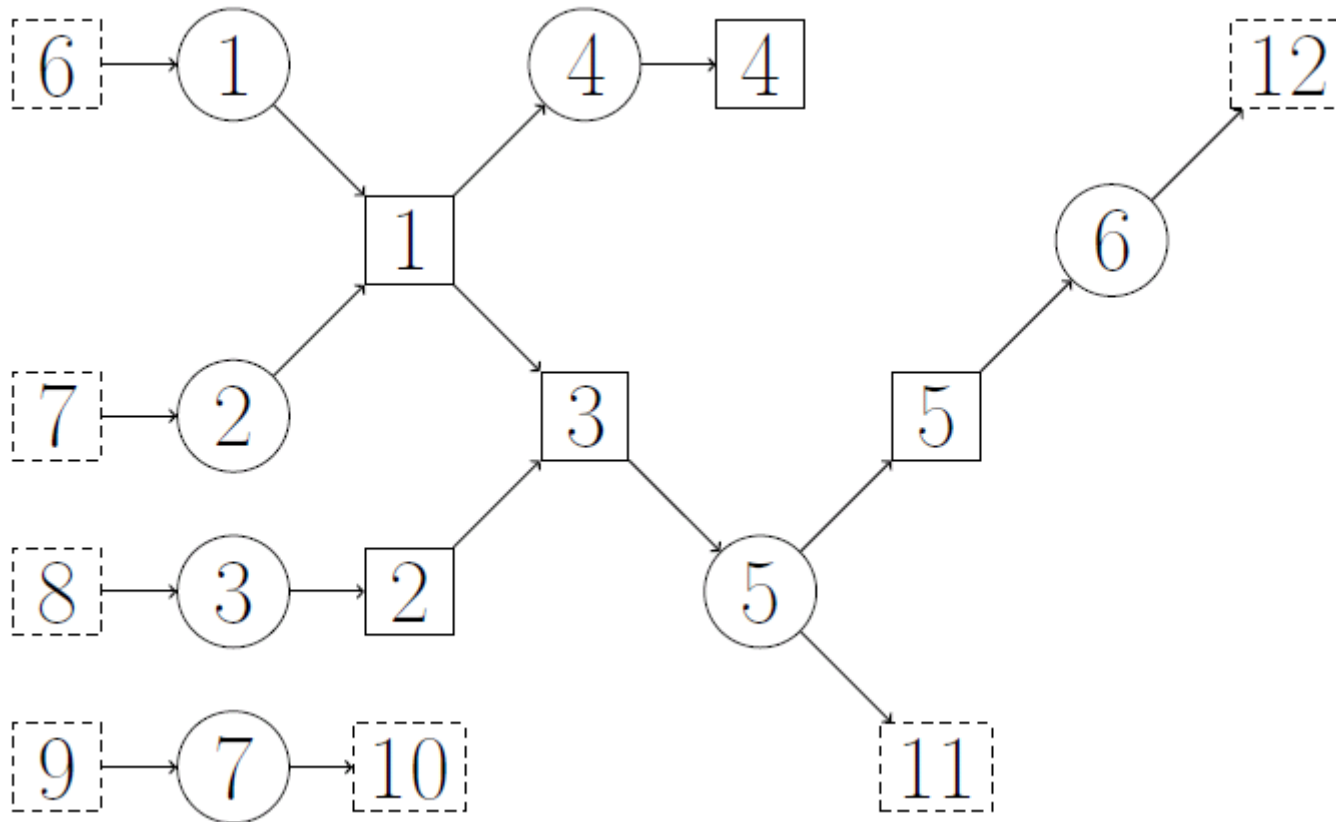


Информационное сопряжение вычислительных задач

- Интерфейс задачи:
 - входные и выходные параметры
- Обмен между задачами в одном блоке
 - синхронные зависимости по данным
- Обмен по каналам передачи данных
 - сообщения
 - расписание обмена



Функционирование ИУС в реальном времени





Жизненный цикл ПО ИУС

Входные данные на следующий инкремент (дополнительный функционал)

Базовый, повторяющийся элемент инкрементальной модели ЖЦ

Замечания на версию

Фаза планирования

Ввод системы в эксплуатацию

Разработка системных требований

Системное тестирование

Разработка архитектуры системы

Системная интеграция

Разработка требований на ПО высокого уровня

Функциональное тестирование

Разработка требований на ПО низкого уровня

Тестирование компонентов ПО

Кодирование, отладка и интеграция ПО

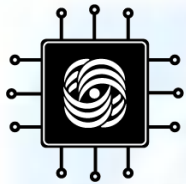
Тестирование, интеграция, отработка

Информационное сопряжение

Специфика инкрементальной модели

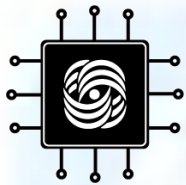
Системные активности

Активности по разработке ПО



Инструментальные средства

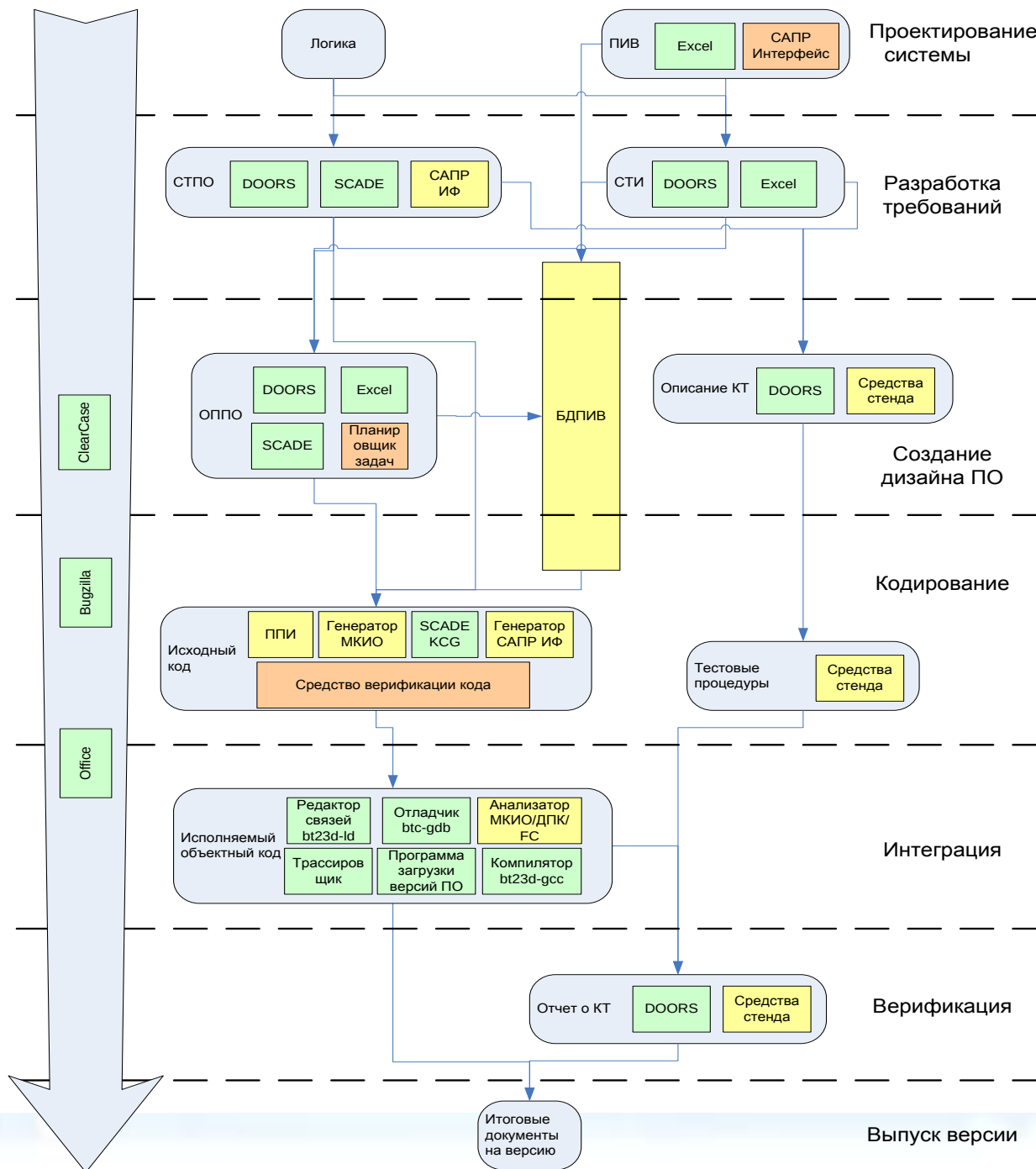
- разработка требований
- управление версиями
- отслеживание проблем и изменений
- поддержка сопряжения подсистем ПО
- проектирование индикационных форматов
- проектирование алгоритмов
- построение расписаний
- конфигурирование сред обмена данными
- верификация и тестирование ПО ИУС

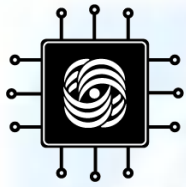


Цепочка средств разработки бортового ПО

- Сквозная поддержка ЖЦ, включая активности на всех фазах
- Сопряжение «вход-выход» с обеспечением совместимости форматов данных
- Особое внимание на переходы между фазами
 - требуется фиксация выходных артефактов

Цепочка средств разработки бортового ПО





Тестирование ИУС

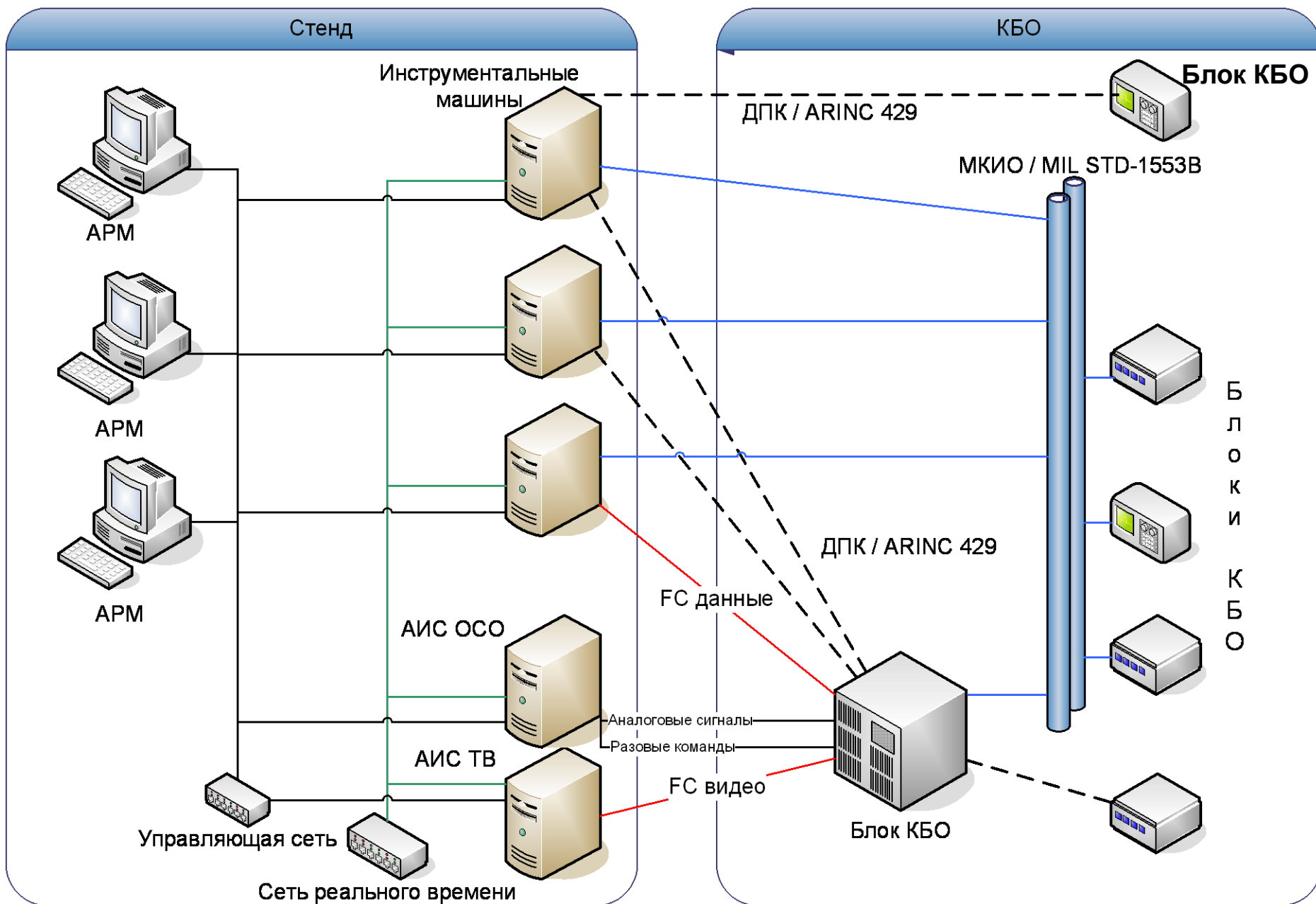
Цена ошибки: Ariane-5

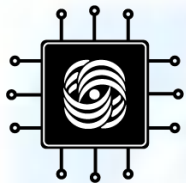
- Июнь 1996 года, взрыв ракеты спустя 40 сек. после старта,
- Ущерб – \$500млн (разработка – \$7 млрд.),
- Причина – 64bit float -> 16bit int.



*Кажется,
что-то
пошло не
так...*

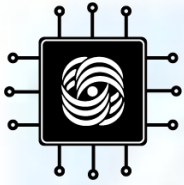
Тестирование ИУС





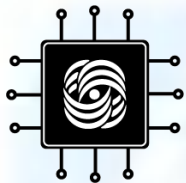
Математические задачи

- Выбор оптимальной конфигурации ИУС РВ
 - требования реального времени
 - требования надёжности
 - ограничения по ресурсам
- Построение расписания вычислений
- Построение расписания обмена данными
- Конфигурирование коммутируемой среды обмена данными
- Верификация работы ИУС РВ (доказательная)
 - функциональная
 - временная
- Генерация тестовых покрытий

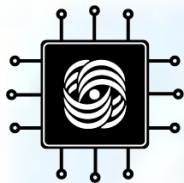


Далее...

- Планирование выполнения задач в ИУС РВ
- Доказательство выполненности требований РВ к выполнению задач
- Оценка наихудшего времени выполнения программ
- Конфигурирование сред информационного обмена, доказательство выполненности требований РВ к обмену



Спасибо за внимание!



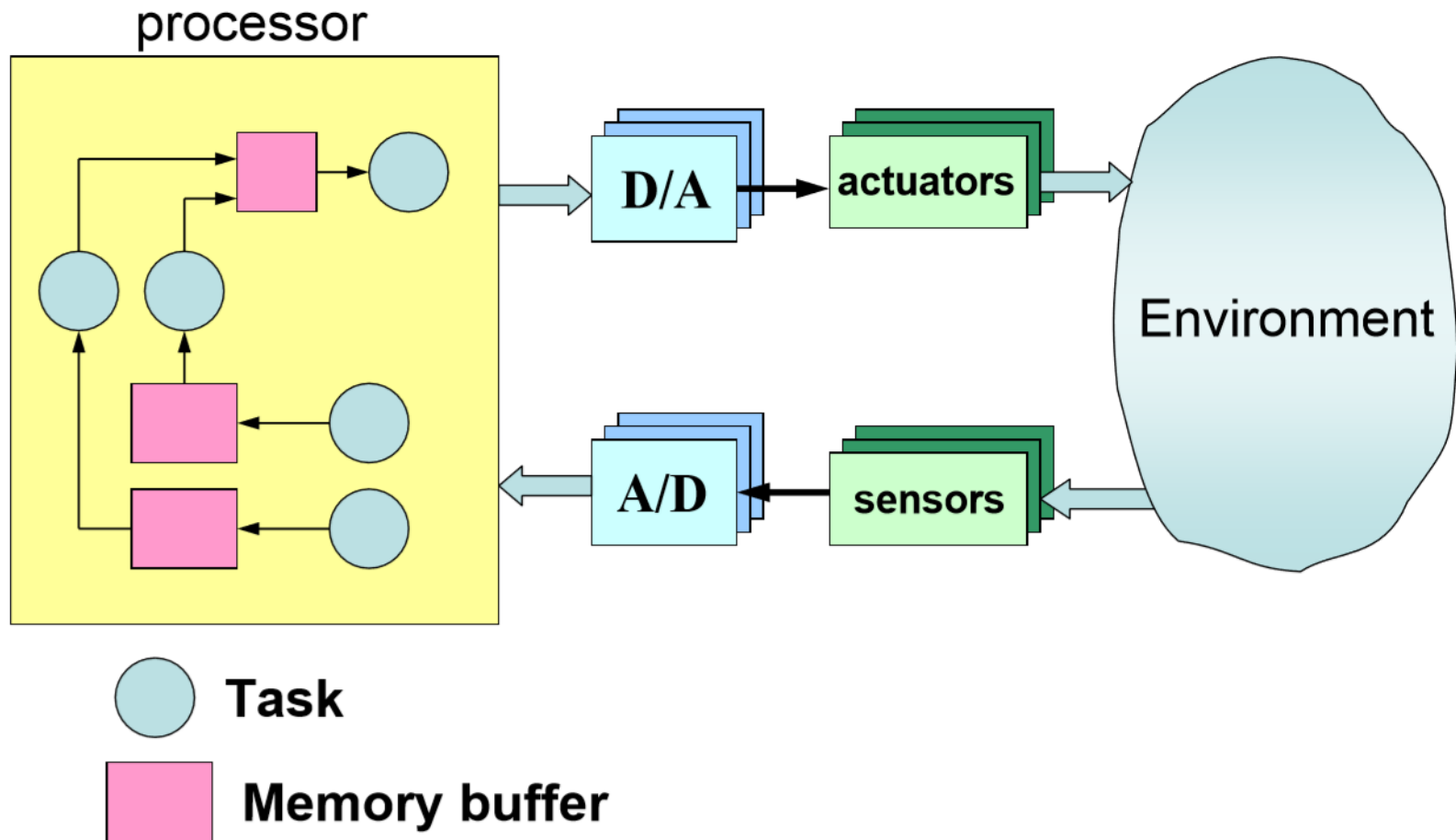
ИНФОРМАЦИОННО-УПРАВЛЯЮЩИЕ СИСТЕМЫ РЕАЛЬНОГО ВРЕМЕНИ

Лекция 2:

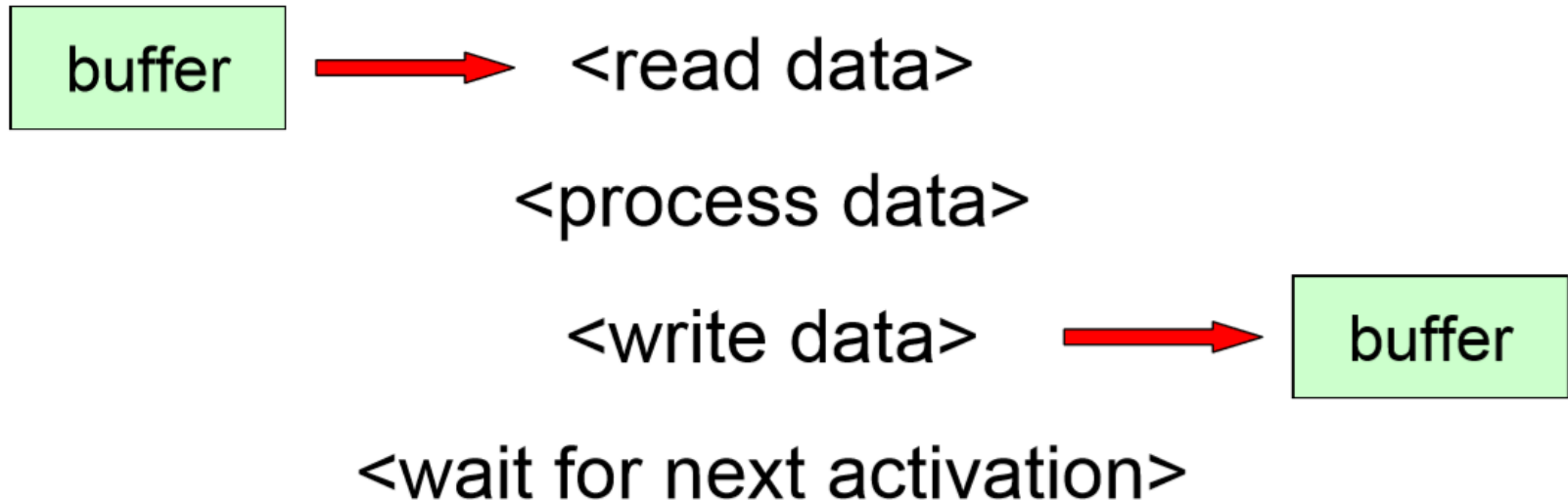
Динамическое планирование вычислений и оценка планируемости - 1

Кафедра АСВК,
Лаборатория Вычислительных Комплексов
Балашов В.В.

Software Control Systems



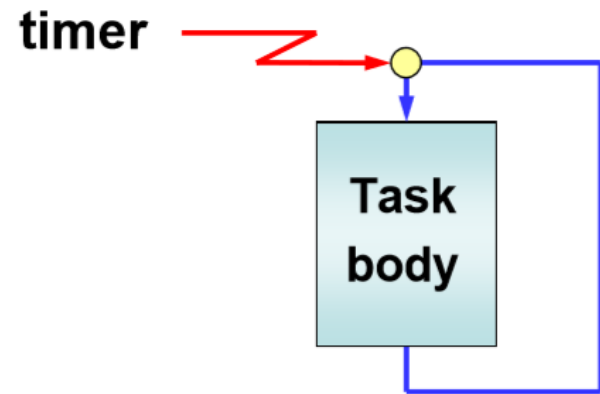
Typical task structure



Activation modes

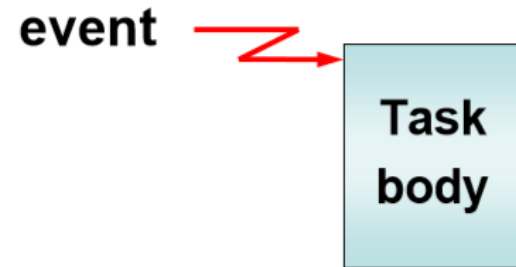
Periodic task (time driven)

A task is automatically activated by the kernel at regular time intervals



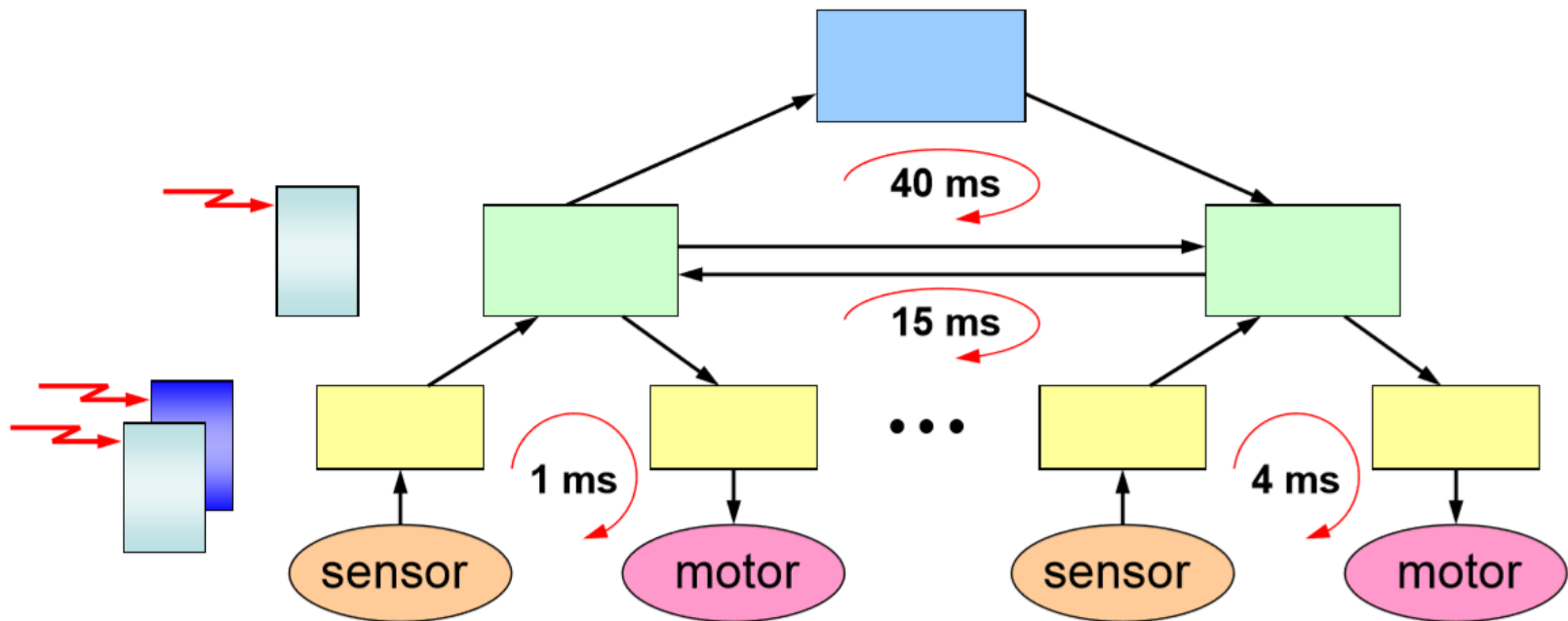
Aperiodic task (event driven)

A task is activated upon the arrival of an event (interrupt or explicit activation)



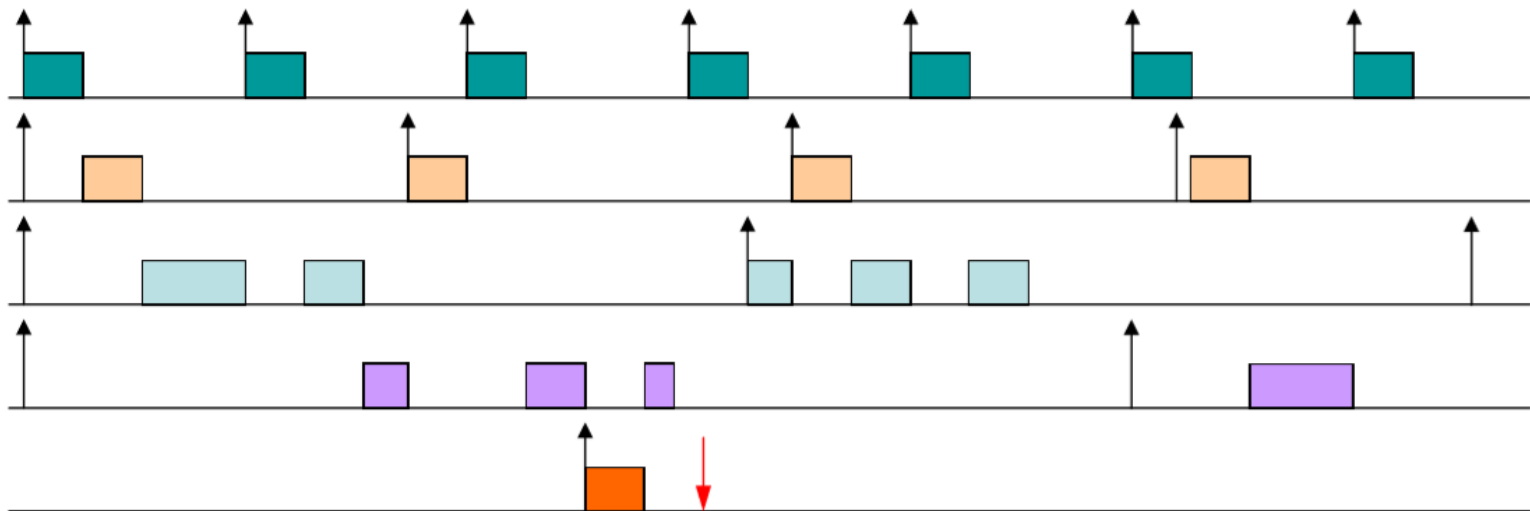
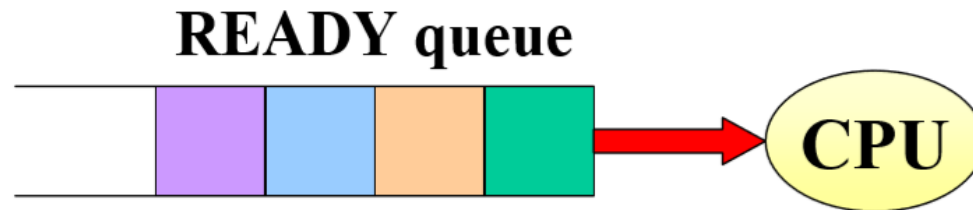
Complex control applications

- Hierarchical design
- Many periodic activities running at different rates
- Many event-driven routines



Task scheduling

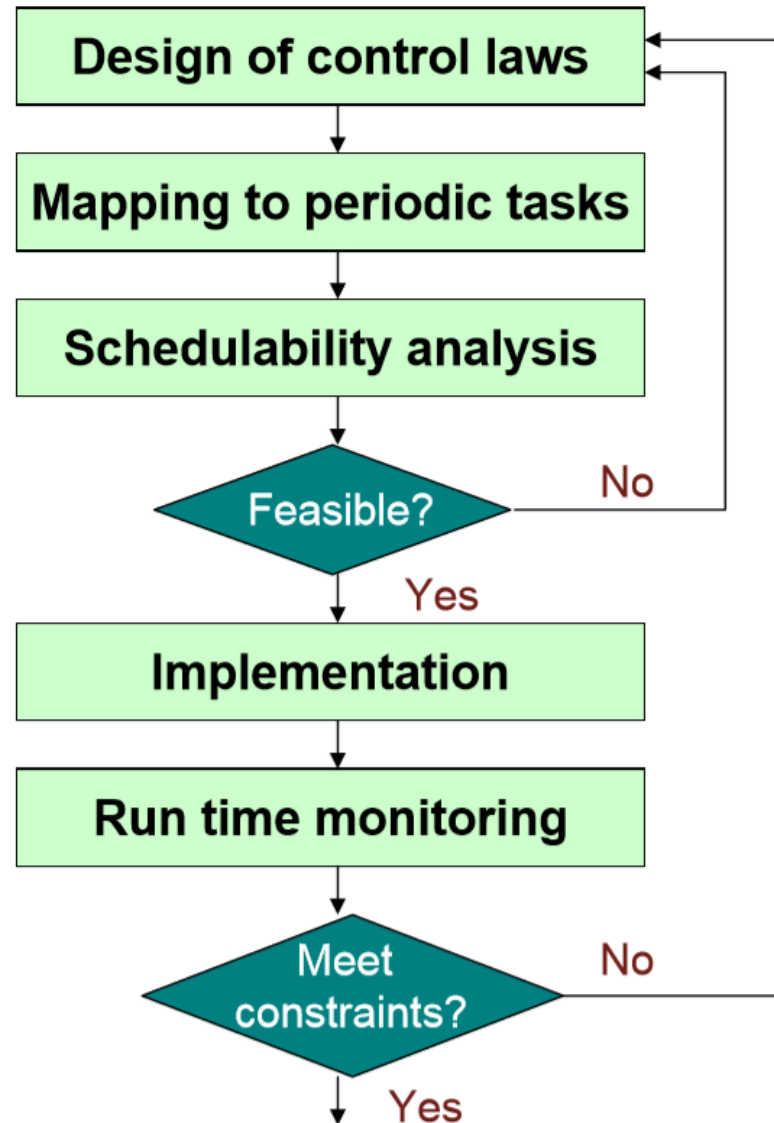
When more tasks are ready to execute, the order of execution is decided by the scheduler:



Importance of scheduling

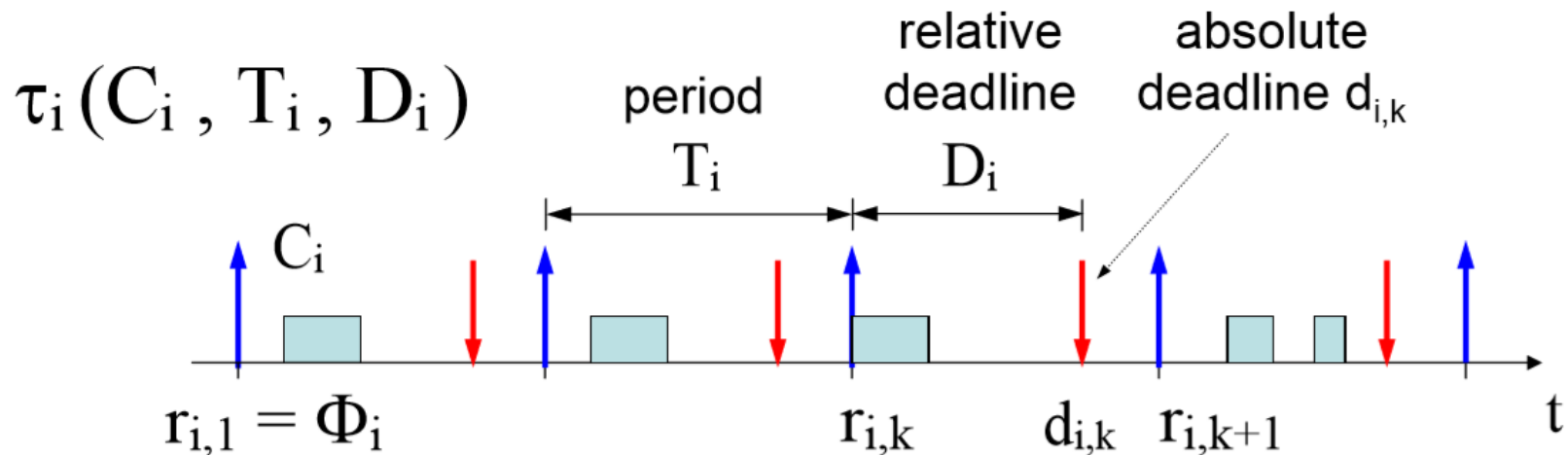
- It affects task response times
- It affects delay and jitter in control loops
- It affects execution times (preemptions destroy cache data and prefetch queues)
- It can be used to cope with overload conditions
- It can be used to optimize resource usage
- It can be used to save energy in processors with voltage scaling (energy-aware scheduling)

Control design



Periodic Task Scheduling

We have n periodic tasks: $\{\tau_1, \tau_2 \dots \tau_n\}$



Goal

- Execute all tasks within their deadlines
- Verify feasibility before runtime

$$r_{i,k} = \Phi_i + (k-1) T_i$$

$$d_{i,k} = r_{i,k} + D_i$$

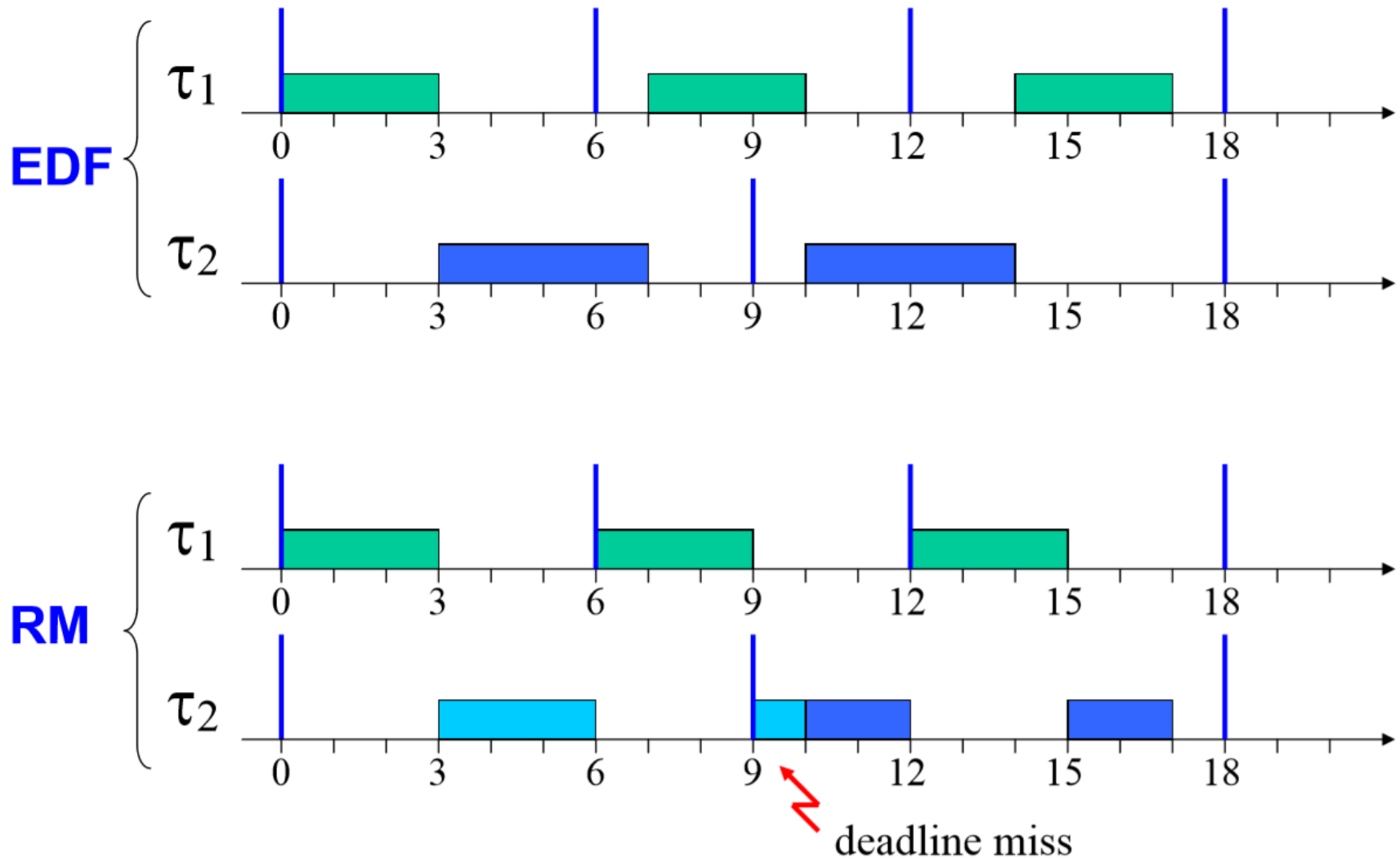
Fixed-Priority Scheduling (FPS)

- This is the most widely used approach
- Each task has a fixed, **static**, priority which is computer pre-run-time
- The runnable tasks are executed in the order determined by their priority
- In real-time systems, the “priority” of a task is derived from its temporal requirements, not its importance to the correct functioning of the system or its integrity

Earliest Deadline First (EDF)

- The runnable tasks are executed in the order determined by the absolute deadlines of the tasks
- The next task to run being the one with the shortest (nearest) deadline
- Although it is usual to know the relative deadlines of each task (e.g. 25ms after release), the absolute deadlines are computed at run time and hence the scheme is described as **dynamic**

EDF vs. RM Schedule



FPS v EDF

- FPS is easier to implement as priorities are static
- EDF is dynamic and requires a more complex run-time system which will have higher overhead
- It is easier to incorporate tasks without deadlines into FPS; giving a task an arbitrary deadline is more artificial
- It is easier to incorporate other factors into the notion of priority than it is into the notion of deadline

FPS v EDF

- During overload situations
 - FPS is more predictable; Low priority process miss their deadlines first
 - EDF is unpredictable; a domino effect can occur in which a large number of processes miss deadlines
- But EDF gets more out of the processor!

Preemption

- With priority-based scheduling, a high-priority task may be released during the execution of a lower priority one
- In a **preemptive** scheme, there will be an immediate switch to the higher-priority task
- With **non-preemption**, the lower-priority task will be allowed to complete before the other executes
- Preemptive schemes enable higher-priority tasks to be more reactive, and hence they are preferred

Scheduling Characteristics

- Sufficient – pass the test will meet deadlines
- Necessary – fail the test will miss deadlines
- Exact – necessary and sufficient
- Sustainable – system stays schedulable if conditions ‘improve’

Simple Task Model

- The application is assumed to consist of a fixed set of tasks
- All tasks are periodic, with known periods
- The tasks are completely independent of each other
- All system's overheads, context-switching times and so on are ignored (i.e, assumed to have zero cost)
- All tasks have a deadline equal to their period (that is, each task must complete before it is next released)
- All tasks have a fixed worst-case execution time

Standard Notation

B	Worst-case blocking time for the task (if applicable)
C	Worst-case computation time (WCET) of the task
D	Deadline of the task
I	The interference time of the task
N	Number of tasks in the system
P	Priority assigned to the task (if applicable)
R	Worst-case response time of the task
T	Minimum time between task releases, jobs, (task period)
U	The utilization of each task (equal to C/T)

Rate Monotonic Priority Assignment

- Each task is assigned a (unique) priority based on its period; the shorter the period, the higher the priority
- i.e, for two tasks i and j ,

$$T_i < T_j \Rightarrow P_i > P_j$$

- This assignment is optimal in the sense that if any task set can be scheduled (using pre-emptive priority-based scheduling) with a fixed-priority assignment scheme, then the given task set can also be scheduled with a rate monotonic assignment scheme
- Note, priority 1 is the lowest (least) priority

Example Priority Assignment

Process	Period, T	Priority, P
a	25	5
b	60	3
c	42	4
d	105	1
e	75	2

Basic results

Assumptions: $\left\{ \begin{array}{l} \text{Independent tasks} \\ \Phi_i = 0 \quad D_i = T_i \end{array} \right.$

In 1973, Liu & Layland proved that a set of n periodic tasks can be feasibly scheduled

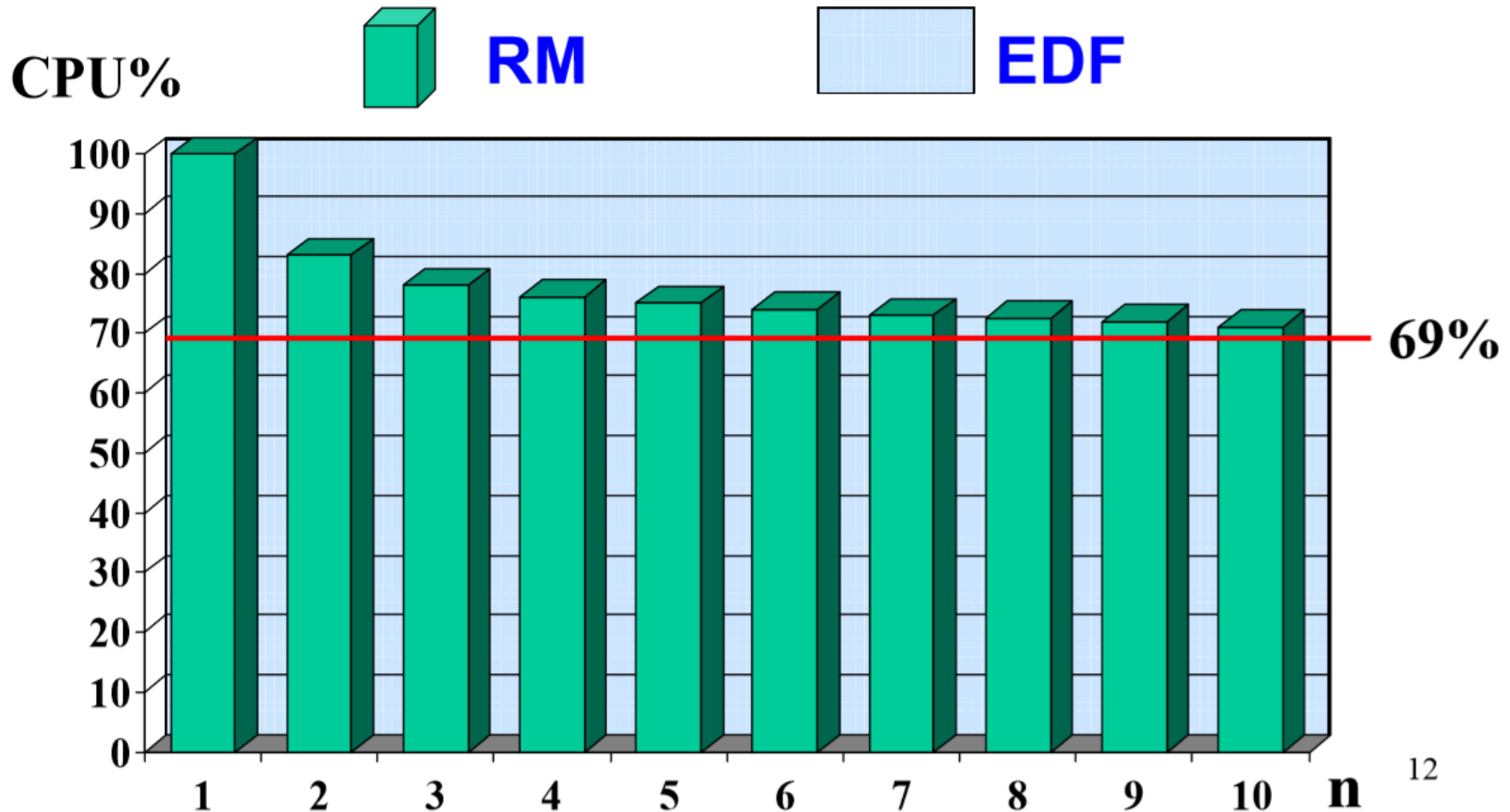
$\left\{ \begin{array}{ll} \text{under RM} & \text{if} \\ \text{under EDF} & \text{if and only if} \end{array} \right.$

$$\sum_{i=1}^n \frac{C_i}{T_i} \leq n(2^{1/n} - 1)$$

$$\sum_{i=1}^n \frac{C_i}{T_i} \leq 1$$

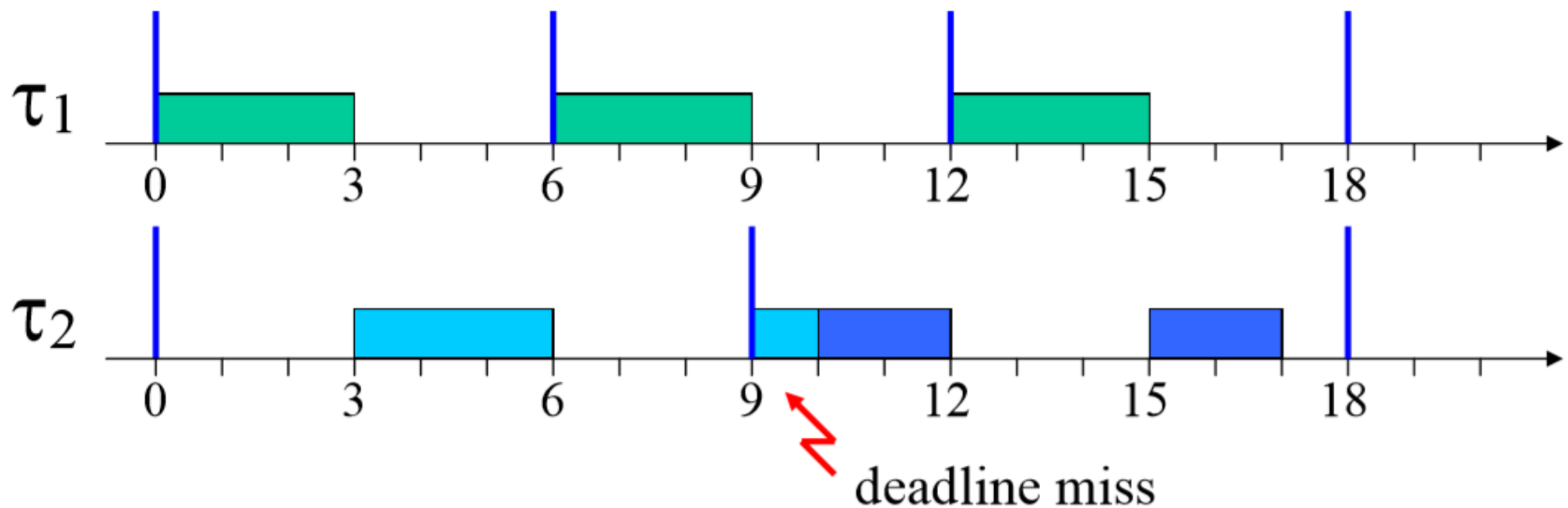
Schedulability bound

for $n \rightarrow \infty$ $U_{\text{lub}} \rightarrow \ln 2 \cong 0.69$

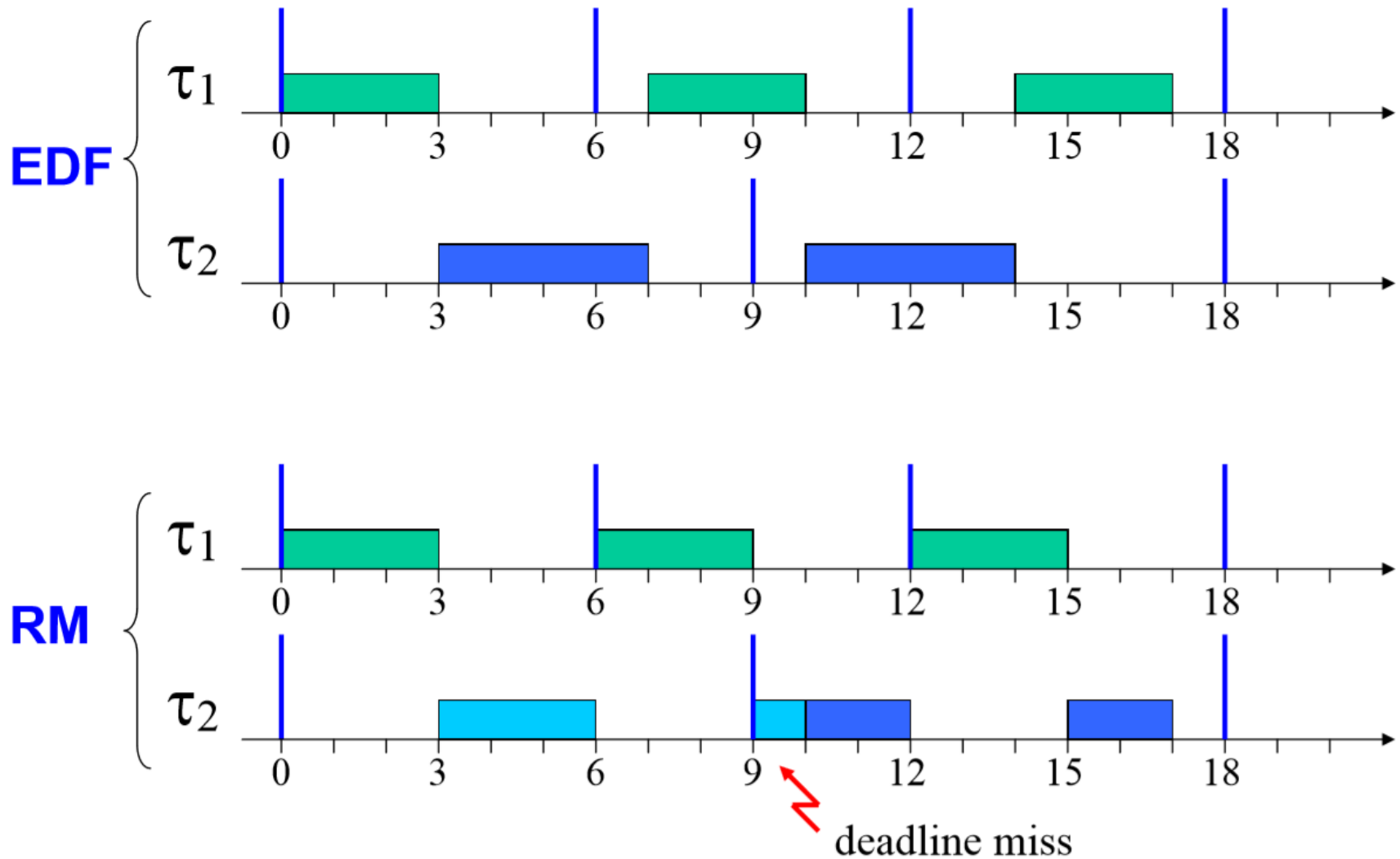


An unfeasible RM schedule

$$U_p = \frac{3}{6} + \frac{4}{9} = 0.944$$

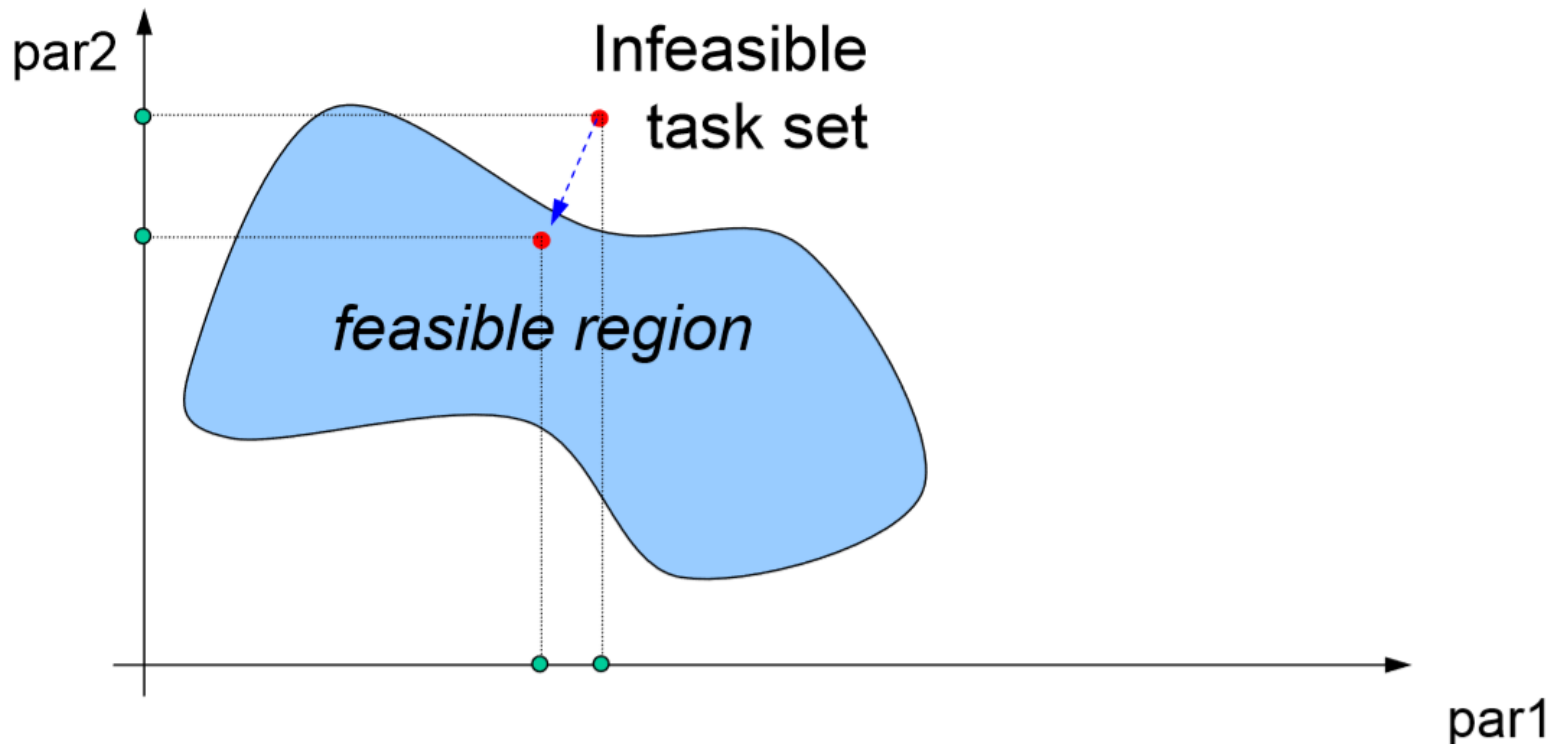


EDF vs. RM Schedule



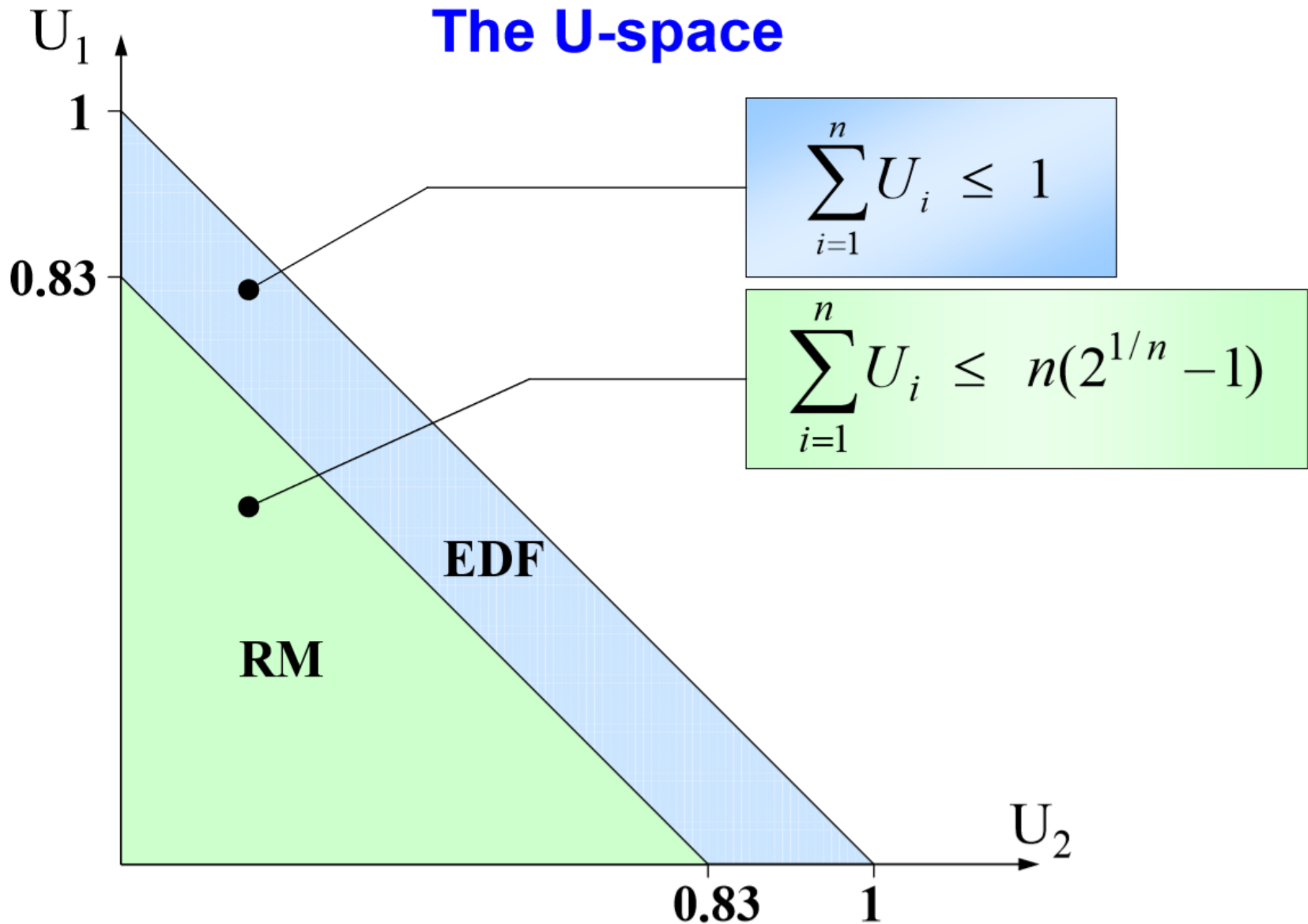
Schedulability region

A more useful approach is to identify a region in the space of task parameters where the system is schedulable by an algorithm.



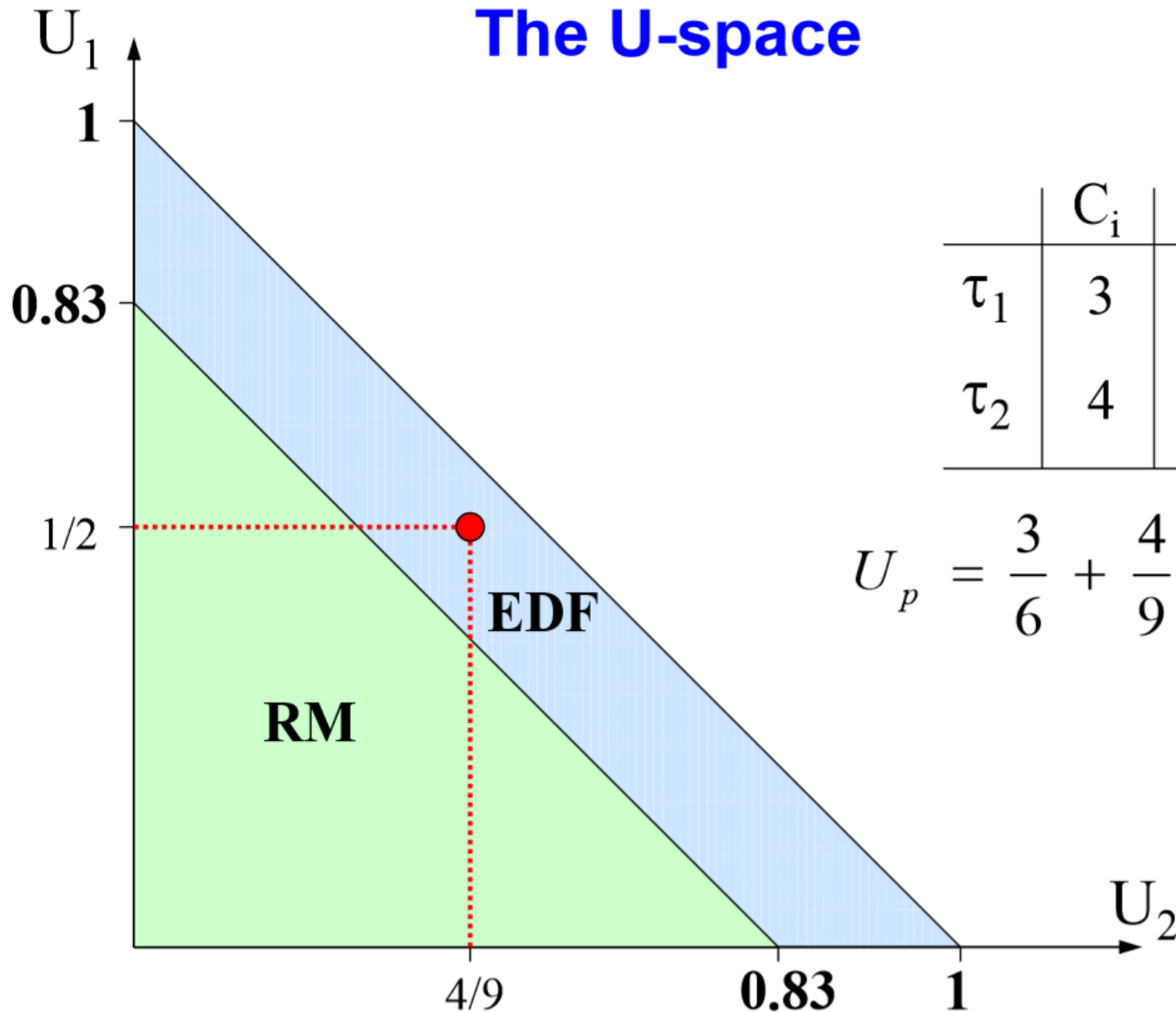
Schedulability region

The U-space



Schedulability region

The U-space



	C_i	T_i
τ_1	3	6
τ_2	4	9

$$U_p = \frac{3}{6} + \frac{4}{9} = 0.94$$

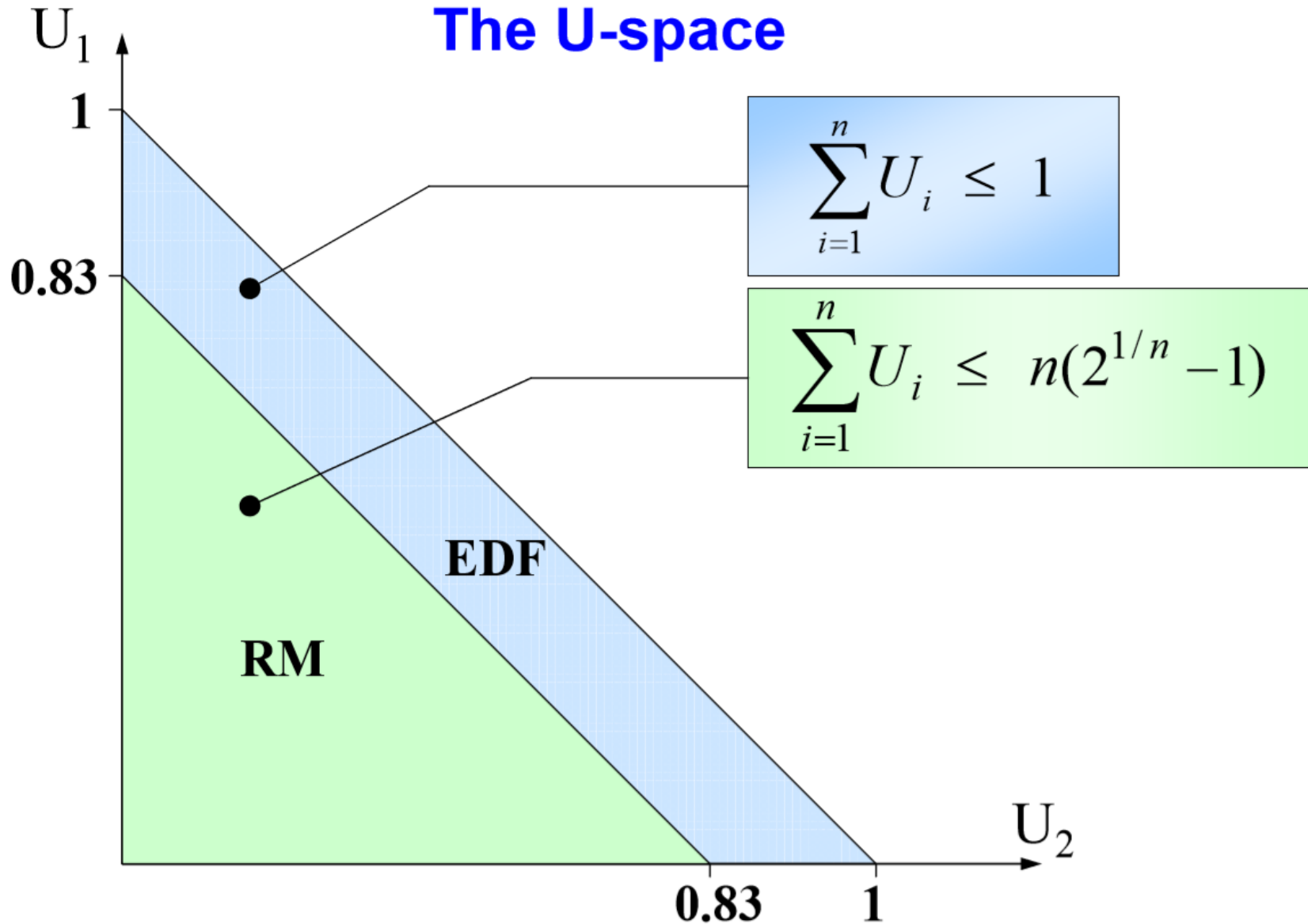
The Hyperbolic Bound

- In 2000, **Bini et al.** proved that a set of n periodic tasks is schedulable with RM if:

$$\prod_{i=1}^n (U_i + 1) \leq 2$$

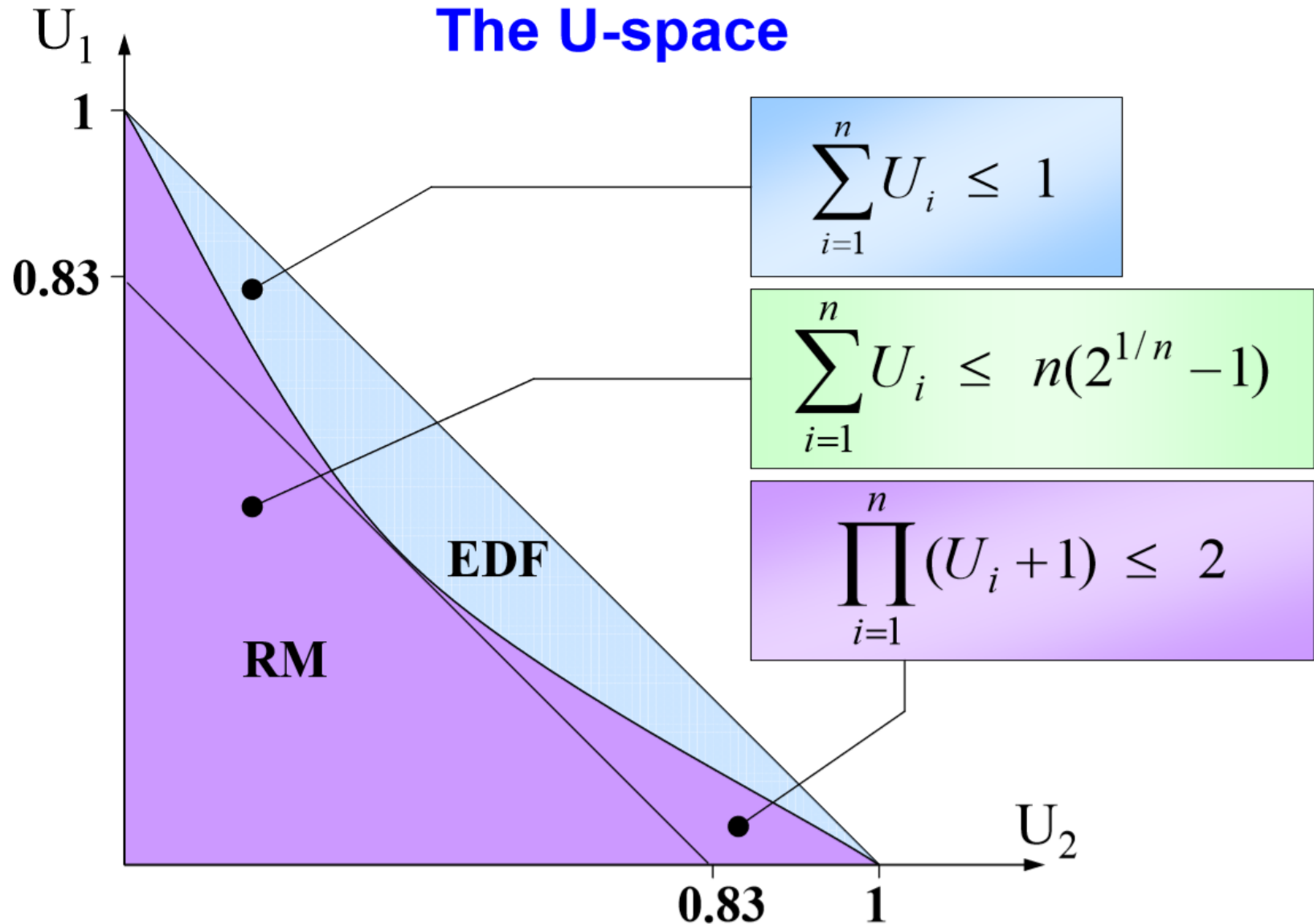
Schedulability region

The U-space

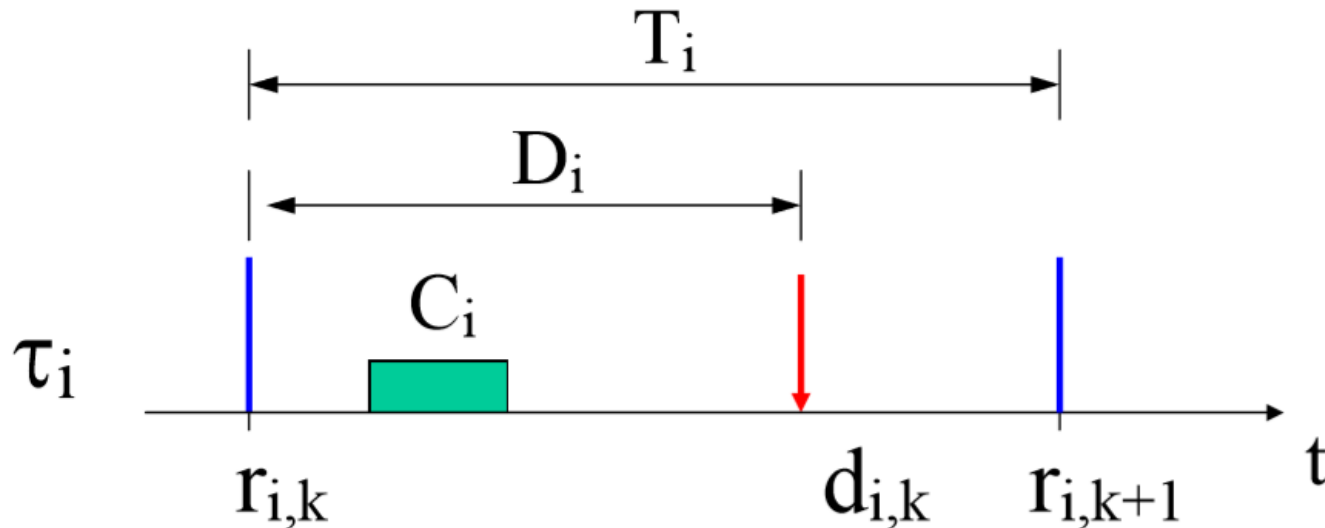


Schedulability region

The U-space



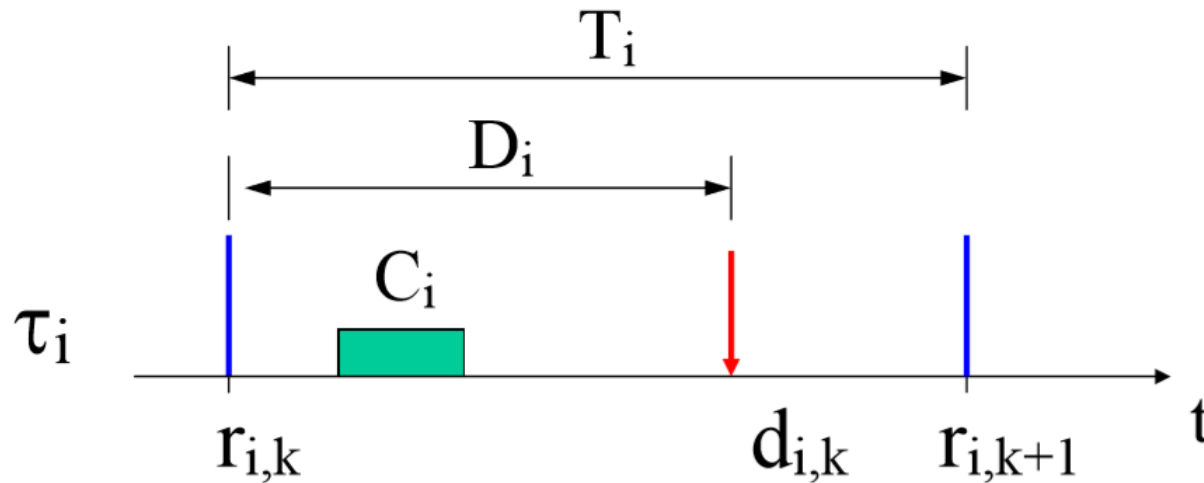
Handling tasks with $D < T$



Scheduling algorithms

- Deadline Monotonic: $p_i \propto 1/D_i$ (static)
- Earliest Deadline First: $p_i \propto 1/d_i$ (dynamic)

How to guarantee feasibility?



- **Fixed priority:** Response Time Analysis (RTA)
- **EDF:** Processor Demand Criterion (PDC)

Response Time Analysis

[Audsley, 1990]

- For each task τ_i compute the interference due to higher priority tasks:

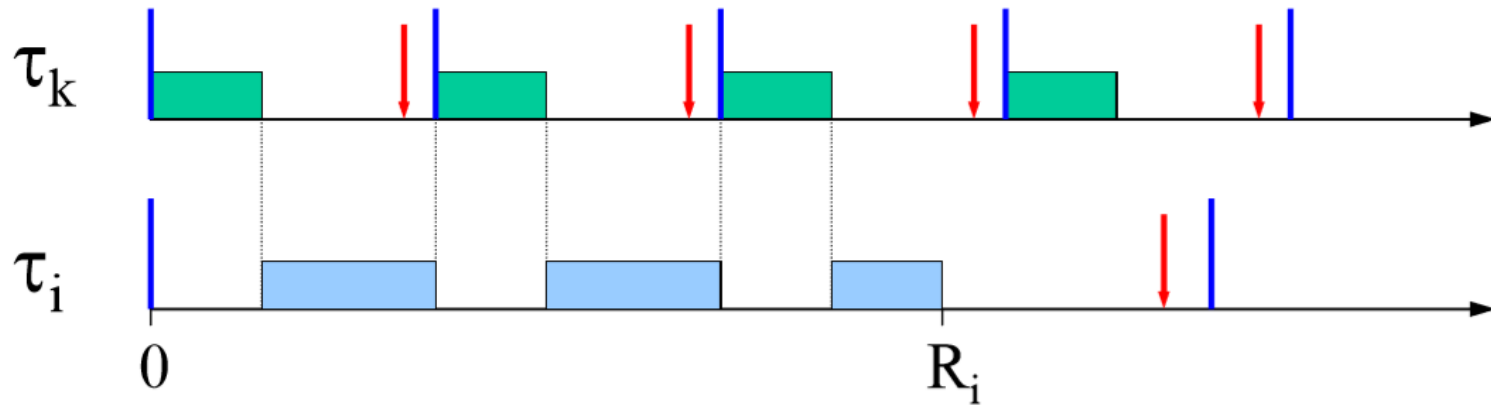
$$I_i = \sum_{D_k < D_i} C_k$$

- Compute its response time as

$$R_i = C_i + I_i$$

- Verify if $R_i \leq D_i$

Computing the interference



Interference of τ_k on τ_i
in the interval $[0, R_i]$:

$$I_{ik} = \left\lceil \frac{R_i}{T_k} \right\rceil C_k$$

Interference of high
priority tasks on τ_i :

$$I_i = \sum_{k=1}^{i-1} \left\lceil \frac{R_i}{T_k} \right\rceil C_k$$

Response Time Equation

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

Where $hp(i)$ is the set of tasks with priority higher than task i

Solve by forming a recurrence relationship:

$$w_i^{n+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^n}{T_j} \right\rceil C_j$$

The set of values $w_i^0, w_i^1, w_i^2, \dots, w_i^n, \dots$ is monotonically non decreasing.
When $w_i^n = w_i^{n+1}$ the solution to the equation has been found; w_i^0 must not be greater than R_i (e.g. 0 or C_i)

Response Time Calculation Algorithm

```
for i in 1..N loop -- for each process in turn
    n := 0
     $w_i^n := C_i$ 
loop
    calculate new  $w_i^{n+1}$ 
    if  $w_i^{n+1} = w_i^n$  then
         $R_i = w_i^n$ 
        exit value found
    end if
    if  $w_i^{n+1} > T_i$  then
        exit value not found
    end if
    n := n + 1
end loop
end loop
```

Task Set A

Task	Period T	ComputationTime C	Priority P
a	7	3	3
b	12	3	2
c	20	5	1

$$w_b^0 = 3$$

$$R_a = 3$$

$$w_b^1 = 3 + \left\lceil \frac{3}{7} \right\rceil 3 = 6$$

$$w_b^2 = 3 + \left\lceil \frac{6}{7} \right\rceil 3 = 6$$

$$R_b = 6$$

$$w_c^0 = 5$$

$$w_c^1 = 5 + \left\lceil \frac{5}{7} \right\rceil 3 + \left\lceil \frac{5}{12} \right\rceil 3 = 11$$

$$w_c^2 = 5 + \left\lceil \frac{11}{7} \right\rceil 3 + \left\lceil \frac{11}{12} \right\rceil 3 = 14$$

$$w_c^3 = 5 + \left\lceil \frac{14}{7} \right\rceil 3 + \left\lceil \frac{14}{12} \right\rceil 3 = 17$$

$$w_c^4 = 5 + \left\lceil \frac{17}{7} \right\rceil 3 + \left\lceil \frac{17}{12} \right\rceil 3 = 20$$

$$w_c^5 = 5 + \left\lceil \frac{20}{7} \right\rceil 3 + \left\lceil \frac{20}{12} \right\rceil 3 = 20$$

$$R_c = 20$$

Task Set B

Process	Period T	ComputationTime C	Priority P	Response time R
a	80	40	1	80
b	40	10	2	15
c	20	5	3	5

- The combined utilization is 1.0
- This was above the utilization threshold for three tasks (0.78), therefore it failed the test
- The response time analysis shows that the task set will meet all its deadlines

Response Time Analysis

- Is **sufficient and necessary** (exact)
- If the task set passes the test they will meet all their deadlines; if they fail the test then, at run-time, a task will miss its deadline (unless the computation time estimations themselves turn out to be pessimistic)

Sporadic Tasks

- Sporadic tasks have a minimum inter-arrival time
- They also require $D < T$
- The response time algorithm for fixed priority scheduling works perfectly for values of D less than T as long as the stopping criteria becomes

$$W_i^{n+1} > D_i$$

- It also works perfectly well with any priority ordering — $hp(i)$ always gives the set of higher-priority tasks

Aperiodic Tasks

- These do not have minimum inter-arrival times
- Can run aperiodic tasks at a priority below the priorities assigned to hard processes, therefore, they cannot steal, in a pre-emptive system, resources from the hard processes
- This does not provide adequate support to soft tasks which will often miss their deadlines
- To improve the situation for soft tasks, a **server** can be employed

Execution-time Servers

■ A server:

- Has a capacity/budget of C that is available to its client tasks (typically aperiodic tasks)
- When a client runs it uses up the budget
- The server has a replenishment policy
- If there is currently no budget then clients do not run
- Hence it protects other tasks from excessive aperiodic activity

Periodic Server (PS)

- Budget C
- Replenishment Period T , starting at say 0
- Client ready to run at time 0 (or T , $2T$ etc) runs while budget available, is then suspended
- Budget 'idles away' if no clients
- Analyzed as a periodic task

Deferrable Server (DS)

- Budget C
- Period T – replenished every T time units (back to C)
 - For example 10ms every 50ms
- Anytime budget available clients can execute
- Client suspended when budget exhausted
- DS is referred to as *bandwidth preserving*
 - Retain capacity as long as possible
- PS is not bandwidth preserving

Task Sets with $D < T$

- For $D = T$, Rate Monotonic priority ordering is optimal
- For $D < T$, Deadline Monotonic priority ordering is optimal

$$D_i < D_j \Rightarrow P_i > P_j$$

- Response time analysis is applicable “as is” to task sets with $D \leq T$

D < T Example Task Set

Task	Period T	Deadline D	ComputationTime C	Priority P	Response time R
a	20	5	3	4	3
b	15	7	3	3	6
c	10	10	4	2	10
d	20	20	3	1	20

Proof that DMPO is Optimal

- Deadline monotonic priority ordering (DMPO) is optimal if any task set, $\mathcal{Q}$, that is schedulable by priority scheme, $\mathcal{W}$, is also schedulable by DMPO
- The proof of optimality of DMPO involves transforming the priorities of $\mathcal{Q}$ (as assigned by $\mathcal{W}$) until the ordering is DMPO
- Each step of the transformation will preserve schedulability

DMPO Proof Continued

- Let i and j be two tasks (with adjacent priorities) in $\mathcal{Q}$ such that under $\mathcal{W}$: $P_i > P_j \wedge D_i > D_j$
- Define scheme $\mathcal{W}'$ to be identical to $\mathcal{W}$ except that tasks i and j are swapped

Consider the schedulability of $\mathcal{Q}$ under $\mathcal{W}'$

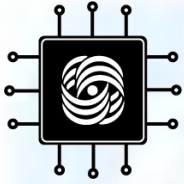
- All tasks with priorities greater than P_i will be unaffected by this change to lower-priority tasks
- All tasks with priorities lower than P_j will be unaffected; they will all experience the same interference from i and j
- Task j , which was schedulable under $\mathcal{W}$, now has a higher priority, suffers less interference, and hence must be schedulable under $\mathcal{W}'$

DMPO Proof Continued

- All that is left is the need to show that task i , which has had its priority lowered, is still schedulable
- Under $\bar{W}$
$$R_j < D_j, D_j < D_i \text{ and } D_i \leq T_i$$
- Hence task i only interferes once during the execution of j
- It follows that:
$$R'_i = R_j \leq D_j < D_i$$
- It can be concluded that task i is schedulable after the switch
- Priority scheme $\bar{W}'$ can now be transformed to $\bar{W}''$ by choosing two more tasks that are in the wrong order for DMP and switching them

СПАСИБО ЗА ВНИМАНИЕ

hbd@cs.msu.su



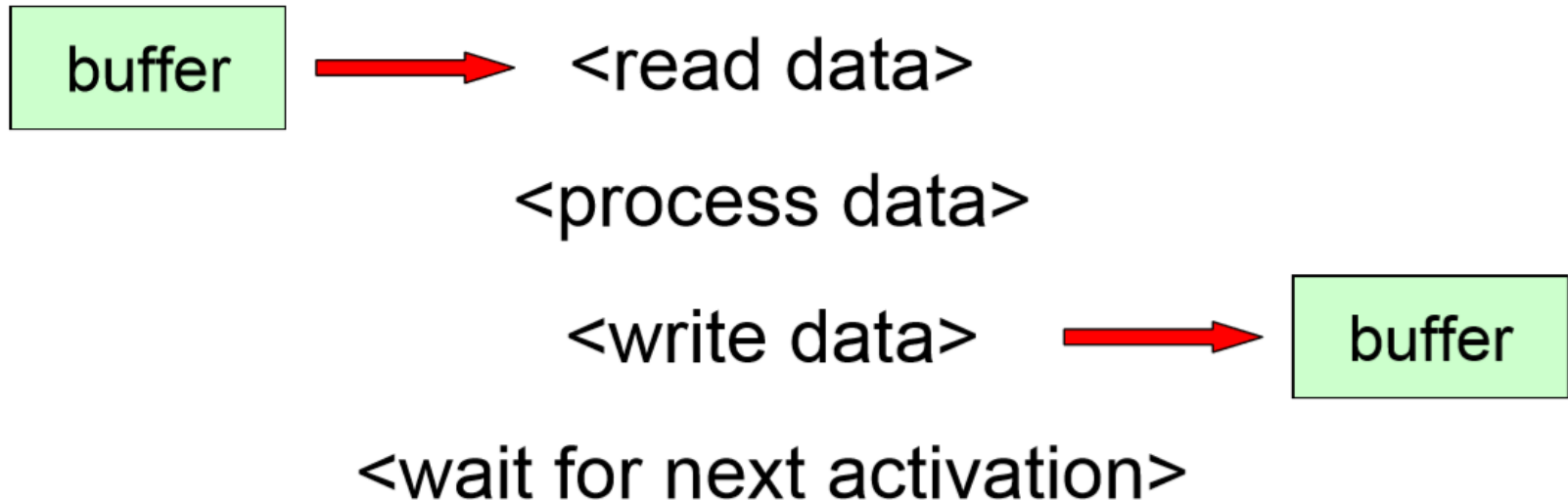
ИНФОРМАЦИОННО-УПРАВЛЯЮЩИЕ СИСТЕМЫ РЕАЛЬНОГО ВРЕМЕНИ

Лекция 3:

Динамическое планирование вычислений и оценка планируемости – 2

Кафедра АСВК,
Лаборатория Вычислительных Комплексов
Балашов В.В.

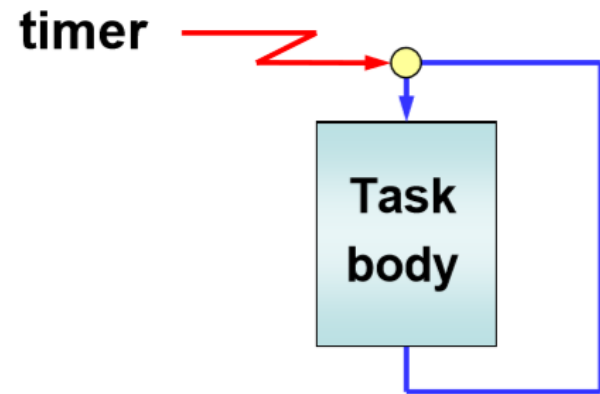
Typical task structure



Activation modes

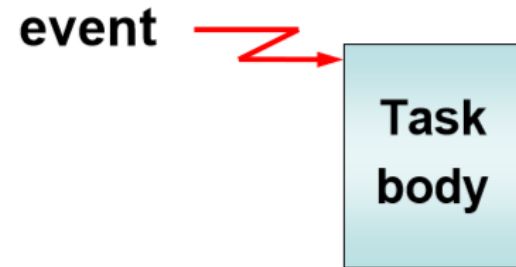
Periodic task (time driven)

A task is automatically activated by the kernel at regular time intervals



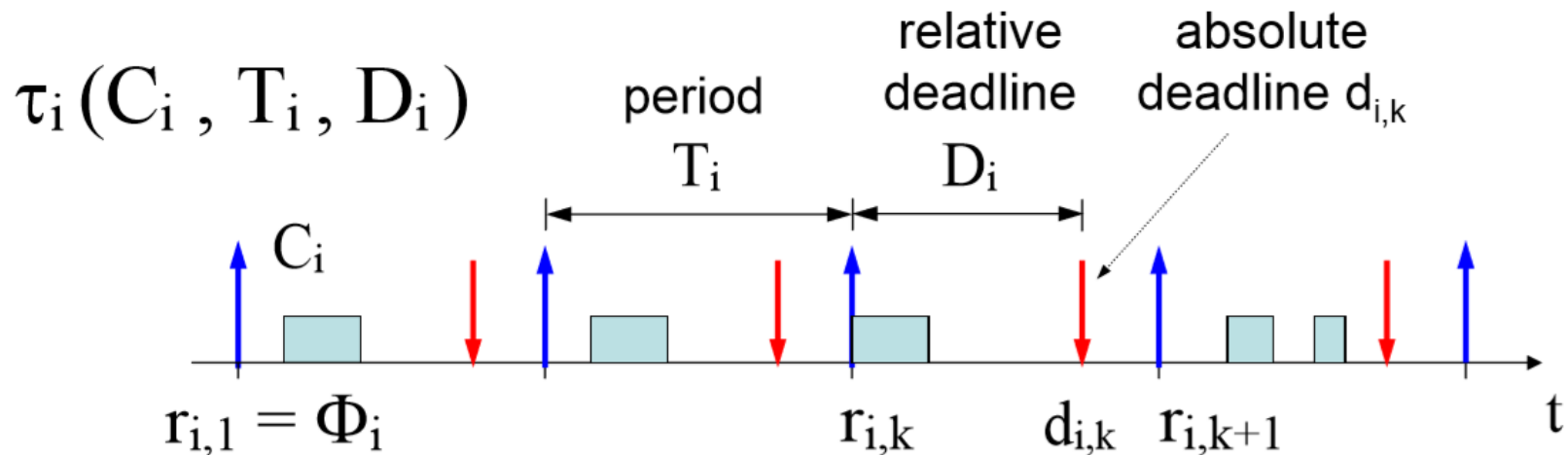
Aperiodic task (event driven)

A task is activated upon the arrival of an event (interrupt or explicit activation)



Periodic Task Scheduling

We have n periodic tasks: $\{\tau_1, \tau_2 \dots \tau_n\}$



Goal

- Execute all tasks within their deadlines
- Verify feasibility before runtime

$$r_{i,k} = \Phi_i + (k-1) T_i$$

$$d_{i,k} = r_{i,k} + D_i$$

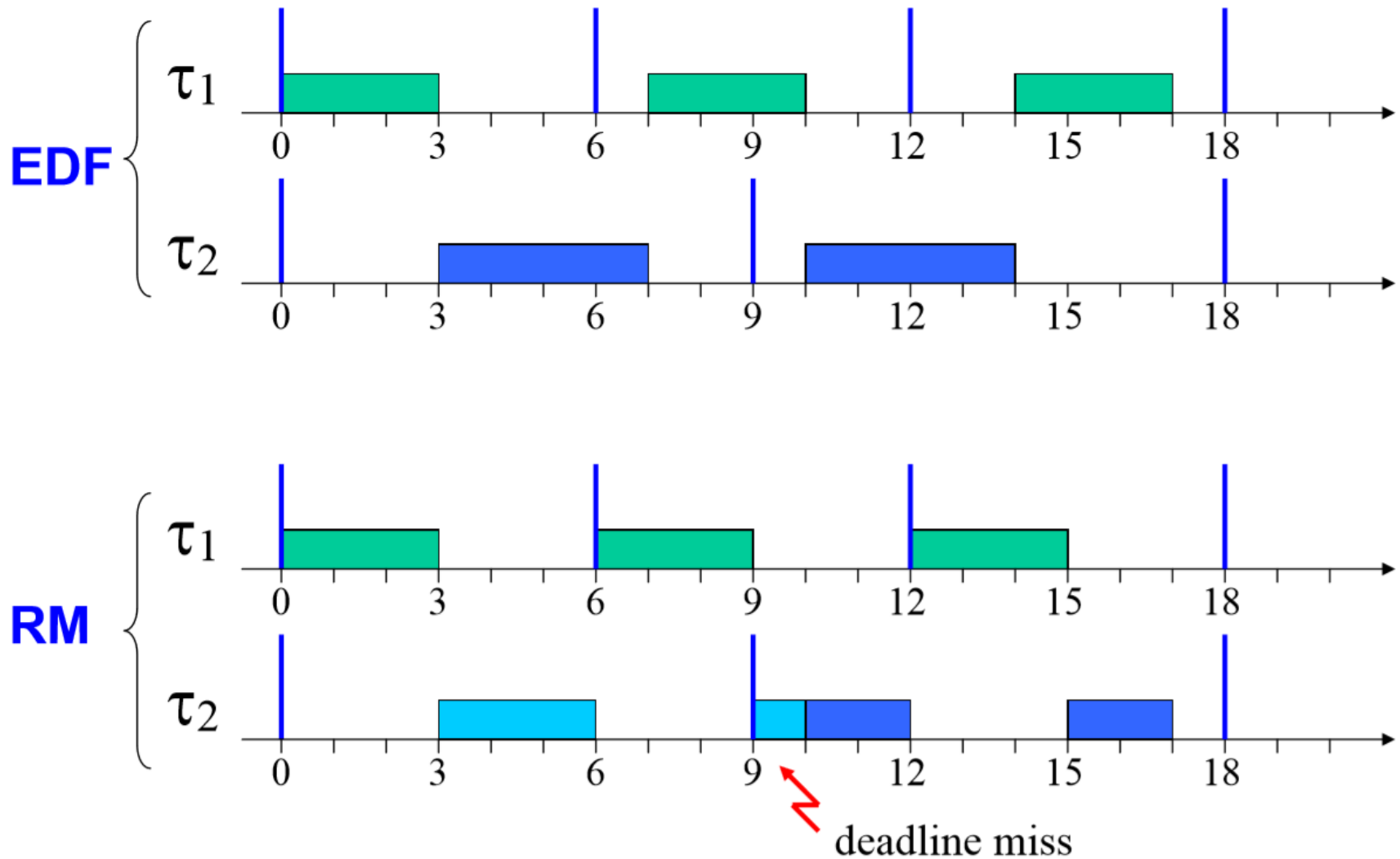
Fixed-Priority Scheduling (FPS)

- This is the most widely used approach
- Each task has a fixed, **static**, priority which is computer pre-run-time
- The runnable tasks are executed in the order determined by their priority
- In real-time systems, the “priority” of a task is derived from its temporal requirements, not its importance to the correct functioning of the system or its integrity

Earliest Deadline First (EDF)

- The runnable tasks are executed in the order determined by the absolute deadlines of the tasks
- The next task to run being the one with the shortest (nearest) deadline
- Although it is usual to know the relative deadlines of each task (e.g. 25ms after release), the absolute deadlines are computed at run time and hence the scheme is described as **dynamic**

EDF vs. RM Schedule



Response Time Analysis

[Audsley, 1990]

- For each task τ_i compute the interference due to higher priority tasks:

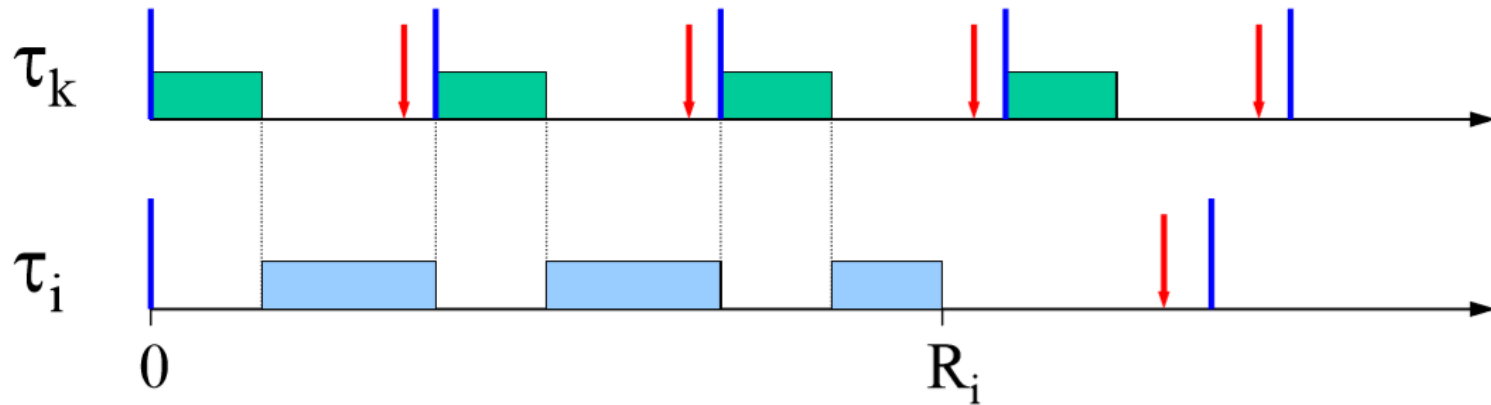
$$I_i = \sum_{D_k < D_i} C_k$$

- Compute its response time as

$$R_i = C_i + I_i$$

- Verify if $R_i \leq D_i$

Computing the interference



Interference of τ_k on τ_i
in the interval $[0, R_i]$:

$$I_{ik} = \left\lceil \frac{R_i}{T_k} \right\rceil C_k$$

Interference of high
priority tasks on τ_i :

$$I_i = \sum_{k=1}^{i-1} \left\lceil \frac{R_i}{T_k} \right\rceil C_k$$

Response Time Equation

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

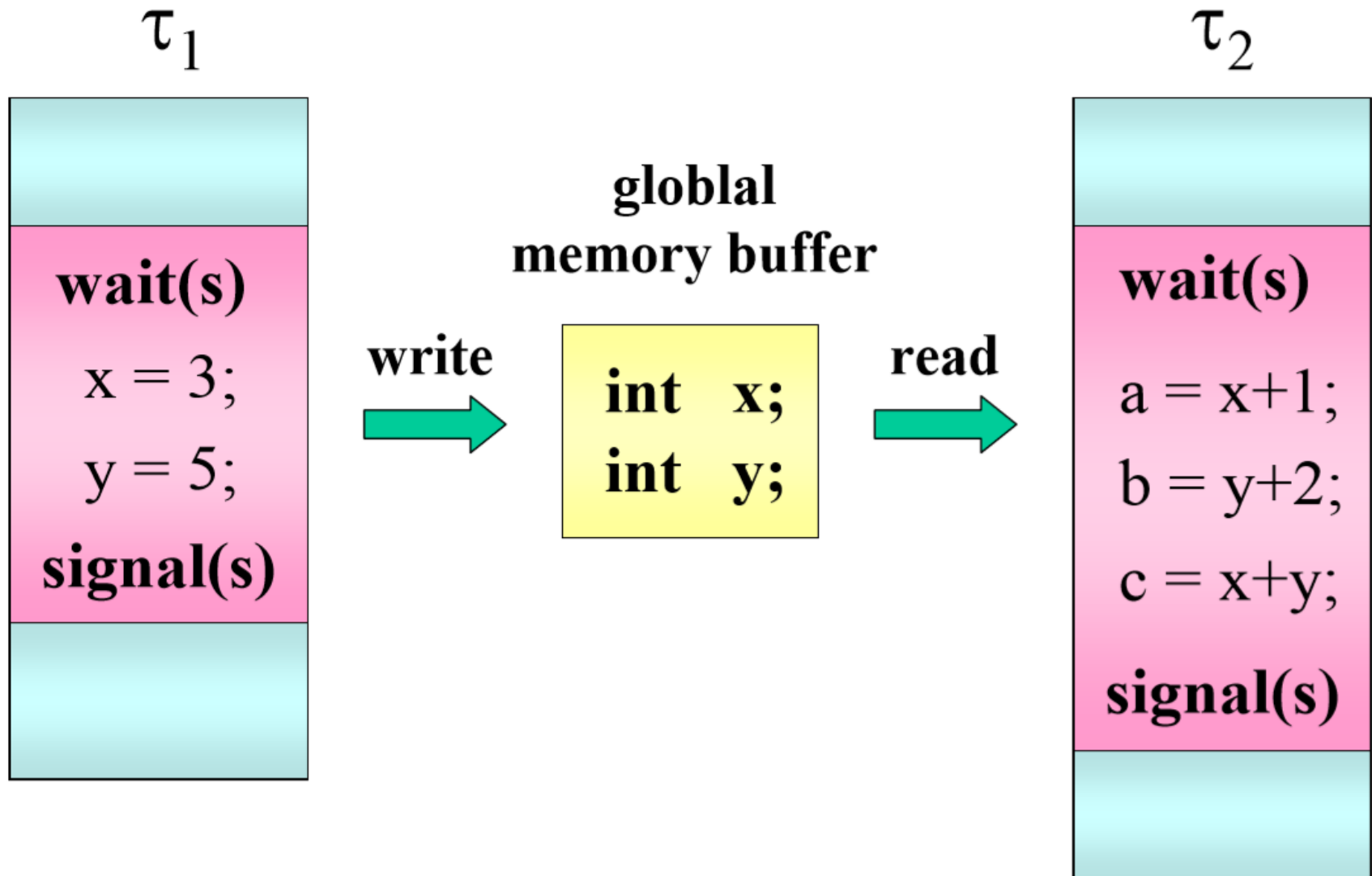
Where $hp(i)$ is the set of tasks with priority higher than task i

Solve by forming a recurrence relationship:

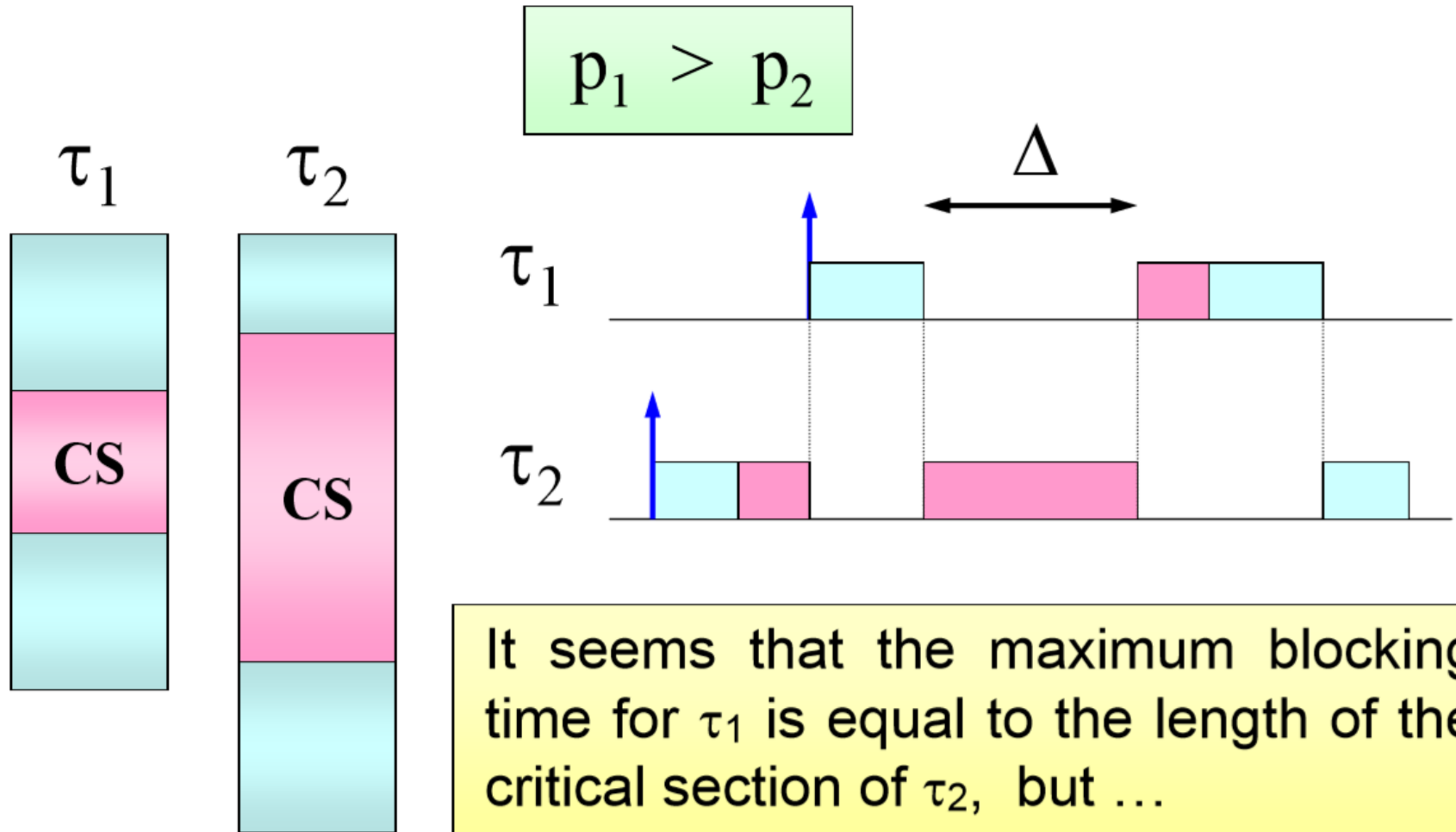
$$w_i^{n+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^n}{T_j} \right\rceil C_j$$

The set of values $w_i^0, w_i^1, w_i^2, \dots, w_i^n, \dots$ is monotonically non decreasing.
When $w_i^n = w_i^{n+1}$ the solution to the equation has been found; w_i^0 must not be greater than R_i (e.g. 0 or C_i)

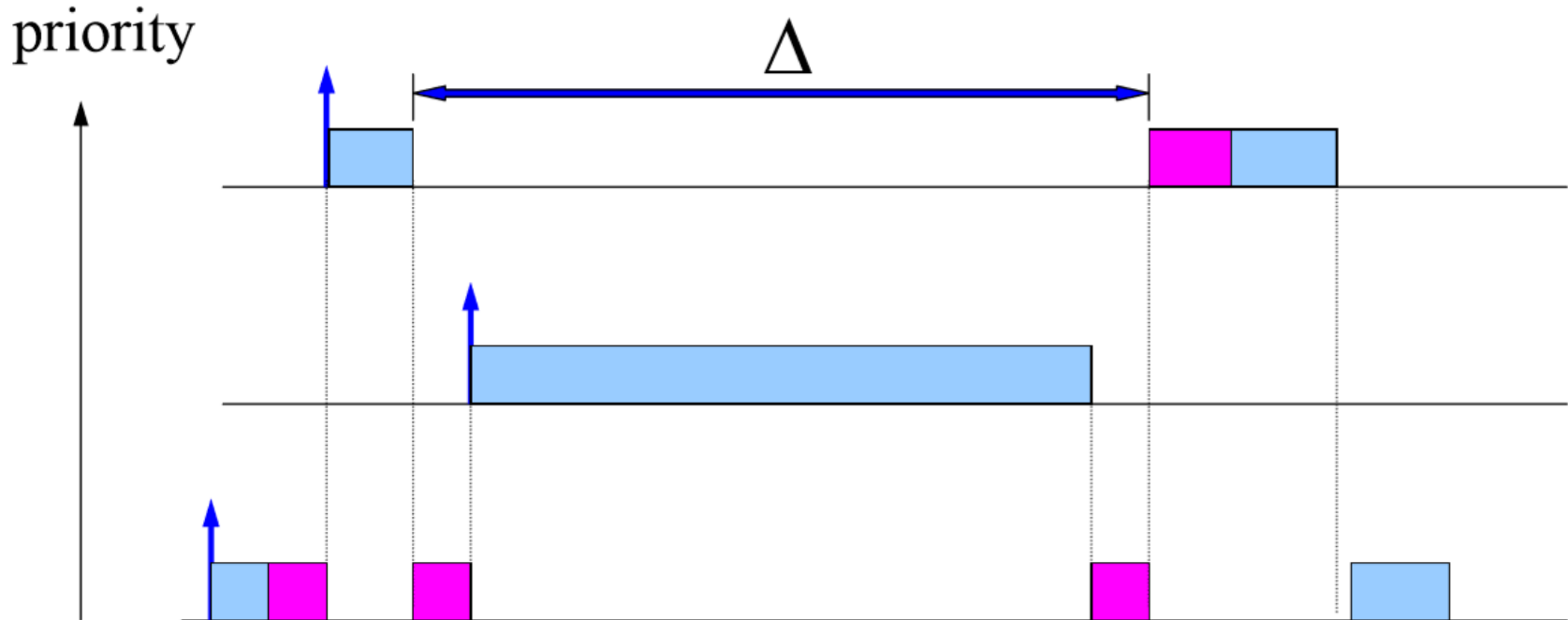
Critical sections



Blocking on a semaphore

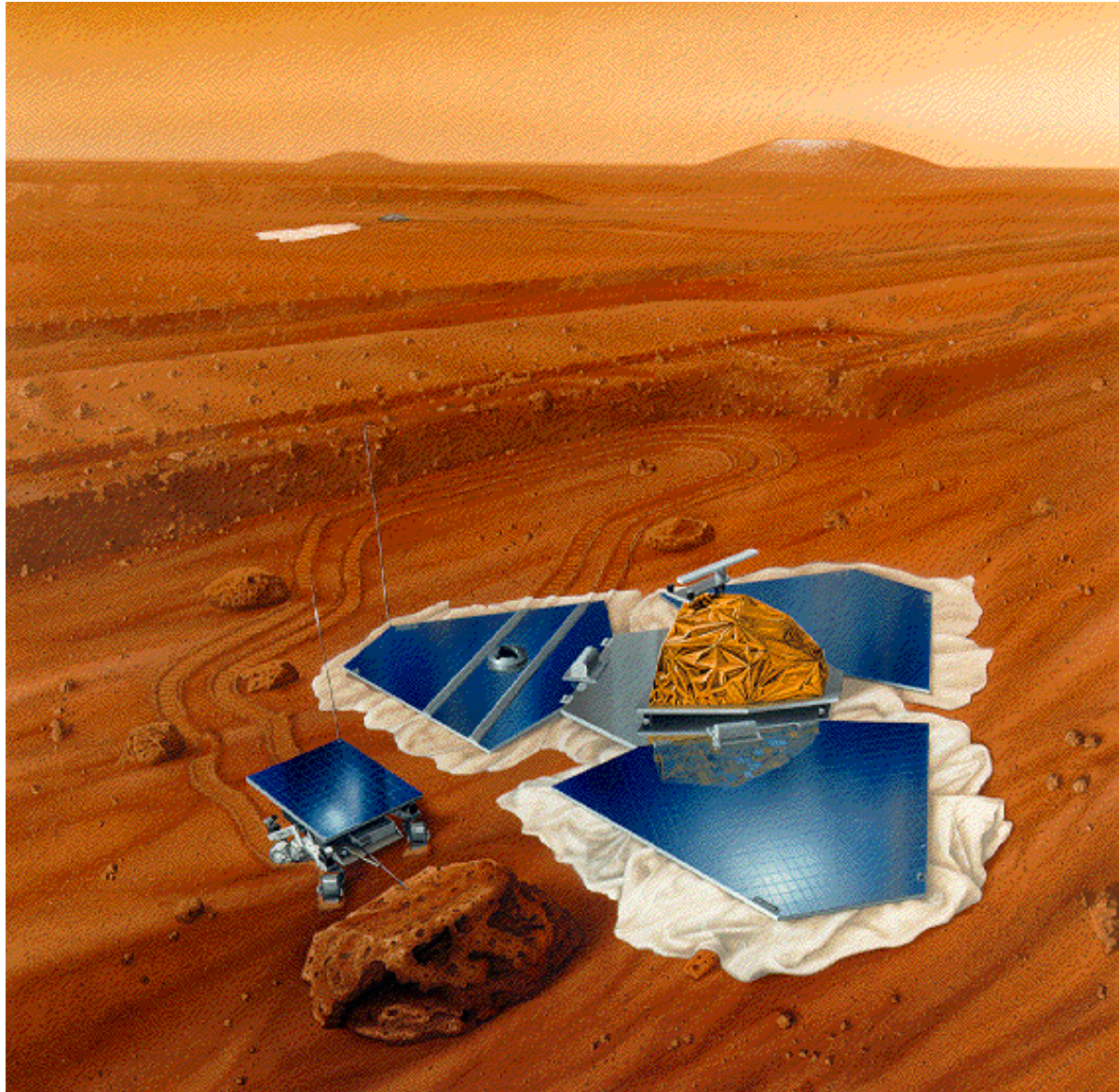


Priority Inversion



Occurs when a high priority task is blocked by a lower-priority task for an unbounded interval of time.

Mars Pathfinder



The MARS Pathfinder problem

“VxWorks provides preemptive priority scheduling of threads. Tasks on the Pathfinder spacecraft were executed as threads with priorities that were assigned in the usual manner reflecting the relative urgency of these tasks.”

“Pathfinder contained an “information bus”, which you can think of as a shared memory area used for passing information between different components of the spacecraft.”

- A bus management task ran frequently with high priority to move certain kinds of data in and out of the information bus. Access to the bus was synchronized with mutual exclusion locks (mutexes).”

The MARS Pathfinder problem

- The meteorological data gathering task ran as an infrequent, low priority thread, ... When publishing its data, it would acquire a mutex, do writes to the bus, and release the mutex. ..
- The spacecraft also contained a communications task that ran with medium priority.”



High priority: retrieval of data from shared memory

Medium priority: communications task

Low priority: thread collecting meteorological data

The MARS Pathfinder problem

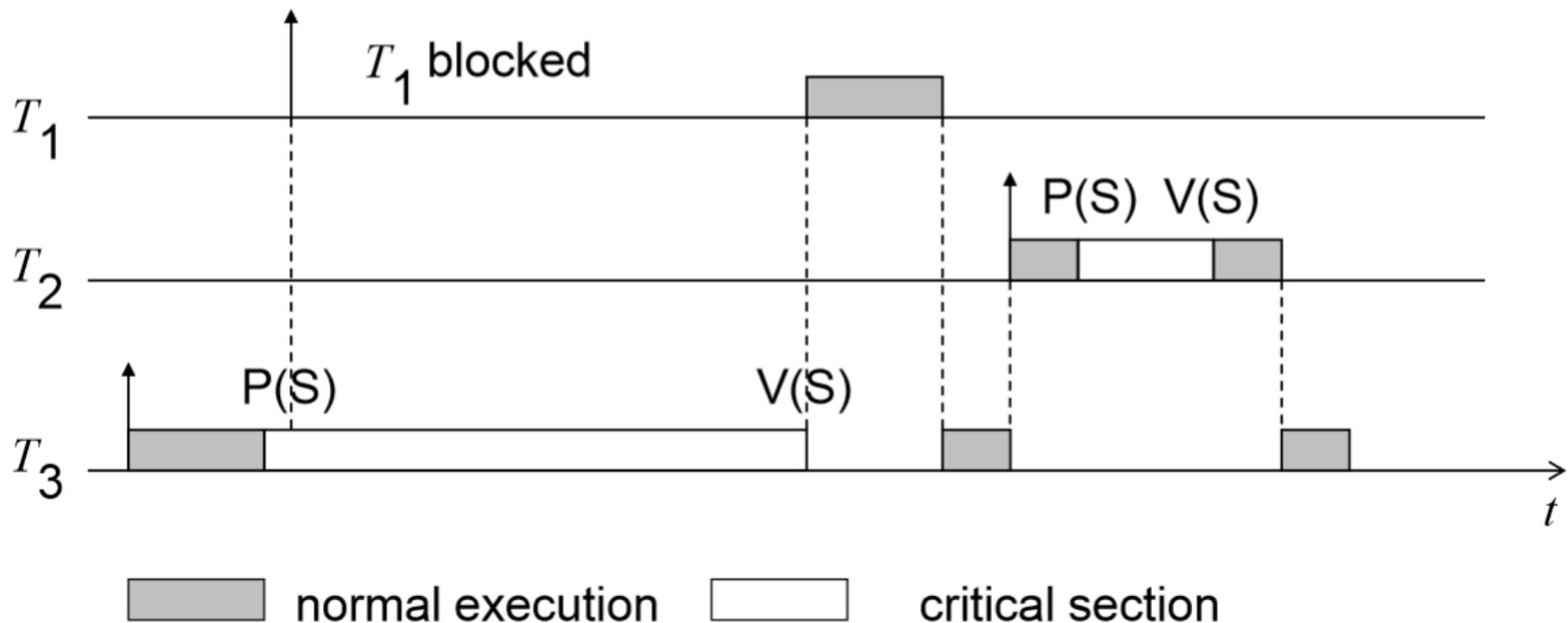
“... However, very infrequently it was possible for an interrupt to occur that caused the (medium priority) communications task to be scheduled during the short interval while the (high priority) information bus thread was blocked waiting for the (low priority) meteorological data thread.

In this case, the long-running communications task, having higher priority than the meteorological task, would prevent it from running, consequently preventing the blocked information bus task from running.

After some time had passed, a watchdog timer would go off, notice that the data bus task had not been executed for some time, conclude that something had gone drastically wrong, and initiate a total system reset.”

Solutions

Disallow preemption during the execution of all critical sections. Simple, but creates unnecessary blocking as unrelated tasks may be blocked.

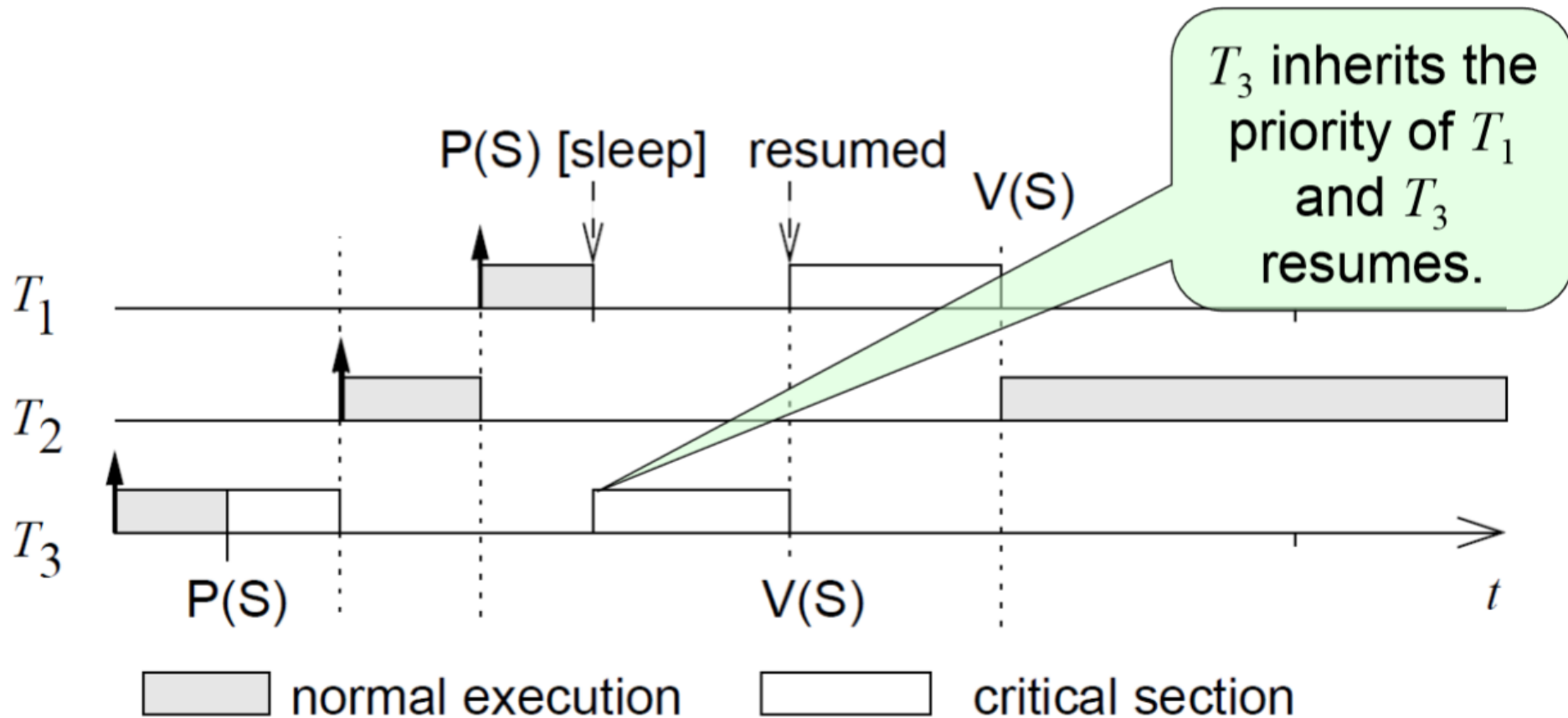


Coping with priority inversion: the priority inheritance protocol

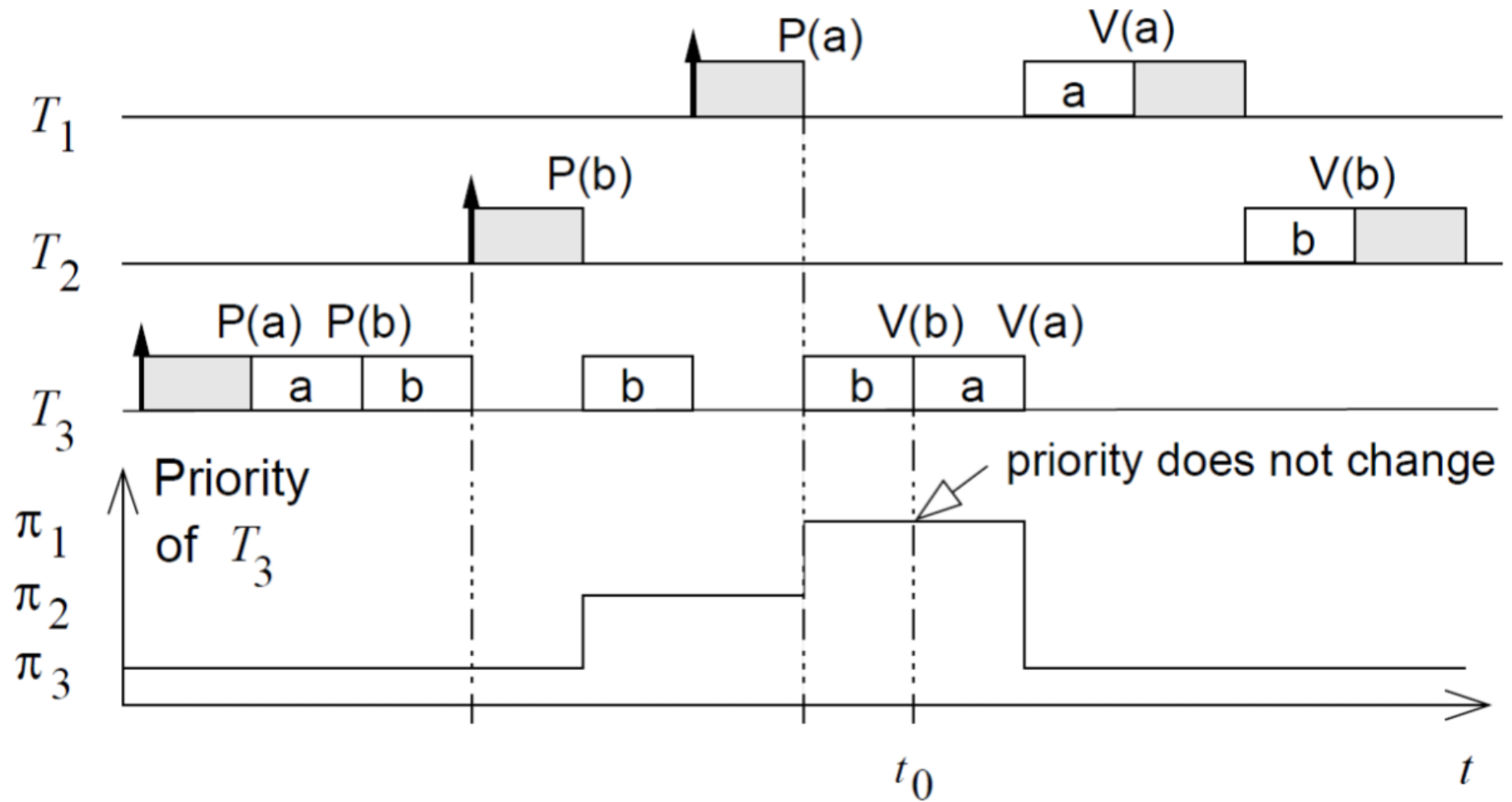
- Tasks are scheduled according to their active priorities. Tasks with the same priorities are scheduled FCFS.
- If task T_1 executes **P(S)** & exclusive access granted to T_2 : T_1 will become blocked.
If $\text{priority}(T_2) < \text{priority}(T_1)$: T_2 inherits the priority of T_1 .
☞ T_2 resumes.
Rule: tasks inherit the **highest** priority of tasks blocked by it.
- When T_2 executes **V(S)**, its priority is decreased to the **highest** priority of the tasks blocked by it.
If no other task blocked by T_2 : $\text{priority}(T_2) := \text{original value}$.
Highest priority task so far blocked on S is resumed.
- Transitive: if T_2 blocks T_1 and T_1 blocks T_0 , then T_2 inherits the priority of T_0 .

Example

How would priority inheritance affect our example with 3 tasks?



Nested critical sections



Priorities and shared resources

Priority Ceiling Protocol:

- Basic idea: Each resource is assigned a priority ceiling equal to the priority of the highest-priority task that can lock it. Then, a task τ_i is allowed to enter a critical region only if its priority is higher than all priority ceilings of the resources currently locked by tasks other than τ_i .
When the task τ_i blocks one or more higher-priority tasks, it temporarily inherits the highest priority of the blocked tasks.
- Advantage:
 - **No deadlock:** priority ceilings prevent deadlocks
 - **No chained blocking:** a task can be blocked at most the duration of one critical region.

Priority Ceiling Protocol

- A high-priority task can be blocked at most once during its execution by lower-priority tasks
- Deadlocks are prevented
- Transitive blocking is prevented
- Mutual exclusive access to resources is ensured (by the protocol itself)

Immediate Ceiling Priority Protocol



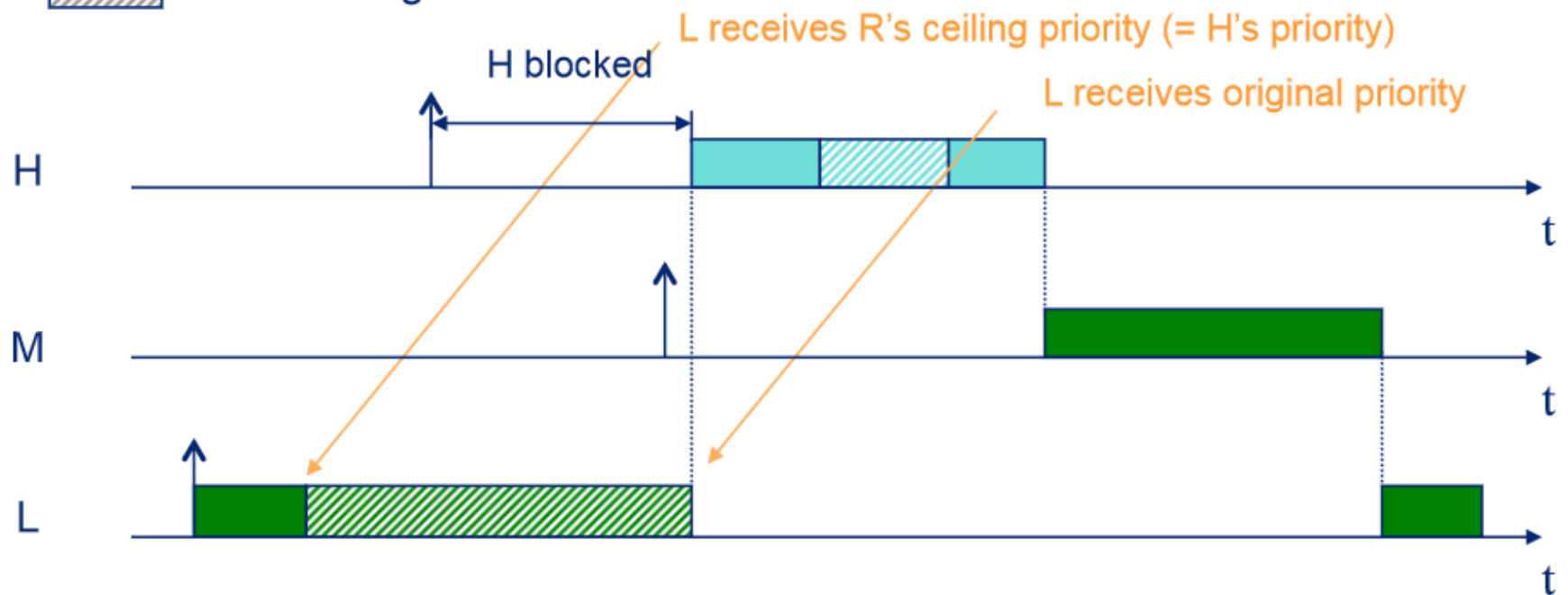
normal execution



critical region

priority (H) > priority (M) > priority (L)

H and L share resource R



Response Time and Blocking

$$R_i = C_i + B_i + I_i$$

$$R_i = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

$$w_i^{n+1} = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^n}{T_j} \right\rceil C_j$$

An Extendible Task Model

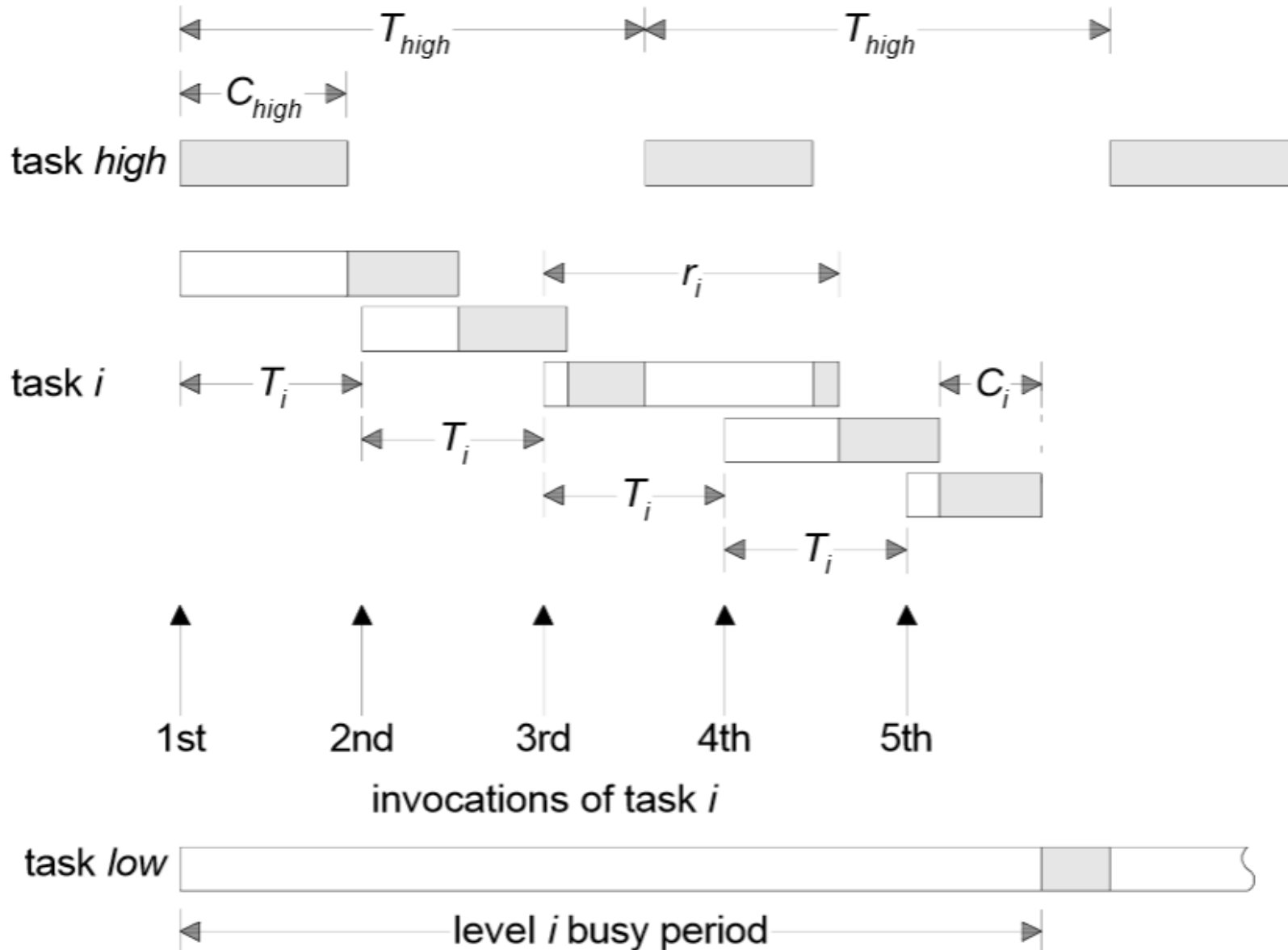
So far:

- Deadlines can be less than period ($D < T$)
- Sporadic and aperiodic tasks, as well as periodic tasks, can be supported
- Task interactions are possible, with the resulting blocking being factored into the response time equations

More:

- Arbitrary Deadlines
- Offsets

Arbitrary Deadlines



Arbitrary Deadlines

- To cater for situations where D (and potentially R) $> T$

$$w_i^{n+1}(q) = B_i + (q + 1)C_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^n(q)}{T_j} \right\rceil C_j$$

$$R_i(q) = w_i^n(q) - qT_i$$

- The number of releases is bounded by the lowest value of q for which the following relation is true: $R_i(q) \leq T_i$
- The worst-case response time is then the maximum value found for each q :

$$R_i = \max_{q=0,1,2,\dots} R_i(q)$$

[A. Burns, K. Tindell and A.J. Wellings.
Fixed Priority Scheduling with Deadlines Prior to Completion]

Offsets

- So far assumed all tasks share a common release time (critical instant)

Task	T	D	C	R
a	8	5	4	4
b	20	10	4	8
c	20	12	4	16

- With offsets

Task	T	D	C	O	R
a	8	5	4	0	4
b	20	10	4	0	8
c	20	12	4	10	8

Arbitrary offsets are not amenable to analysis

Non-Optimal Analysis

- In most realistic systems, task periods are not arbitrary but are likely to be related to one another
- As in the example just illustrated, two tasks have a common period. In these situations it is easy to give one an offset (of $T/2$) and to analyse the resulting system using a transformation technique that removes the offset — and, hence, critical instant analysis applies
- In the example, tasks b and c (having the offset of 10) are replaced by a single notional process with period 10, computation time 4, deadline 10 but no offset

Notional Task Parameters

$$T_n = \frac{T_a}{2} = \frac{T_b}{2}$$

$$C_n = \text{Max}(C_a, C_b)$$

$$D_n = \text{Min}(D_a, D_b)$$

$$P_n = \text{Max}(P_a, P_b)$$

Can be extended to more than two processes

Non-Optimal Analysis

Process	T	D	C	O	R
a	8	5	4	0	4
n	10	10	4	0	8

Non-Optimal Analysis

- This notional task has two important properties:
 - If it is schedulable (when sharing a critical instant with all other tasks) then the two real tasks will meet their deadlines when one is given the half period offset
 - If all lower priority tasks are schedulable when suffering interference from the notional task (and all other high-priority tasks) then they will remain schedulable when the notional task is replaced by the two real tasks (one with the offset)
- These properties follow from the observation that the notional task always uses more (or equal) CPU time than the two real tasks

Insufficient Priorities

- If insufficient priorities then tasks must share priority levels
- If task a shares priority with task b , then each must assume the other interferes
- Priority assignment algorithm can be used to pack tasks together
- Ada requires 31, RT-POSIX 32 and RT-Java 28

Processor Demand Criterion

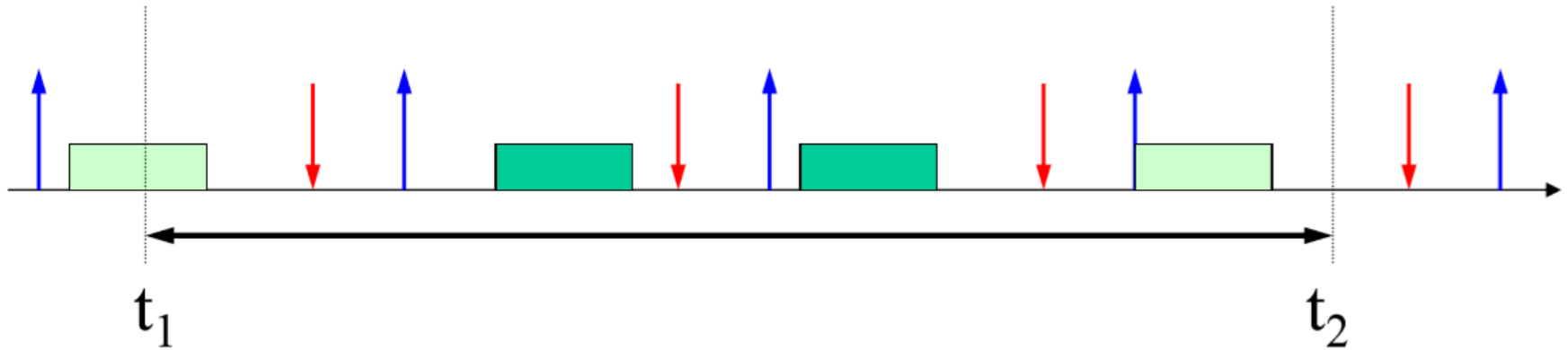
[Baruah, Howell, Rosier 1990]

For checking the existence of a feasible schedule
under **EDF**

In any interval of time, the computation demanded by the task set must be no greater than the available time.

$$\forall t_1, t_2 > 0, \quad g(t_1, t_2) \leq (t_2 - t_1)$$

Processor Demand

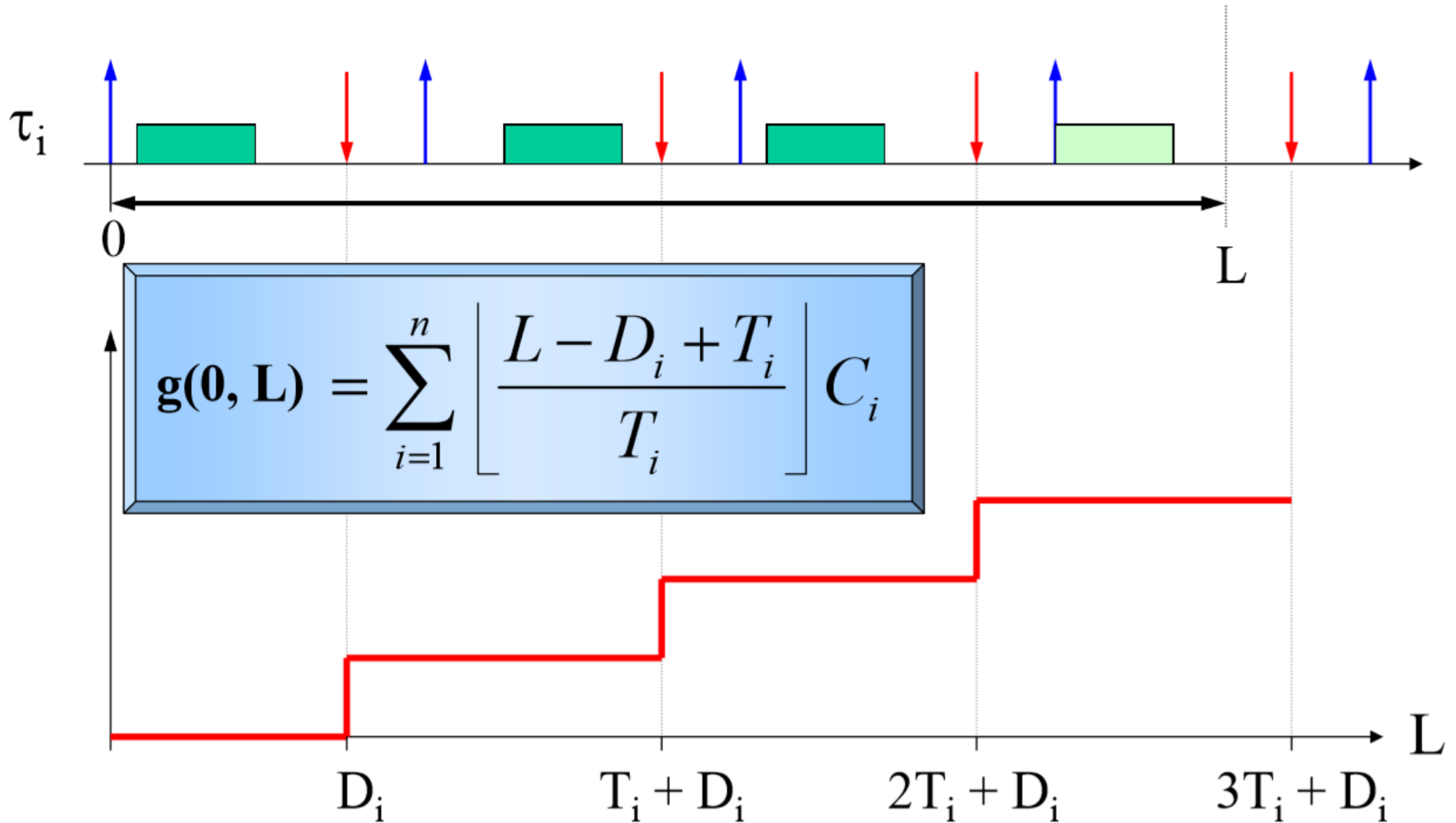


The demand in $[t_1, t_2]$ is the computation time of those jobs started at or after t_1 with deadline less than or equal to t_2 :

$$g(t_1, t_2) = \sum_{\substack{d_i \leq t_2 \\ r_i \geq t_1}} C_i$$

Processor Demand

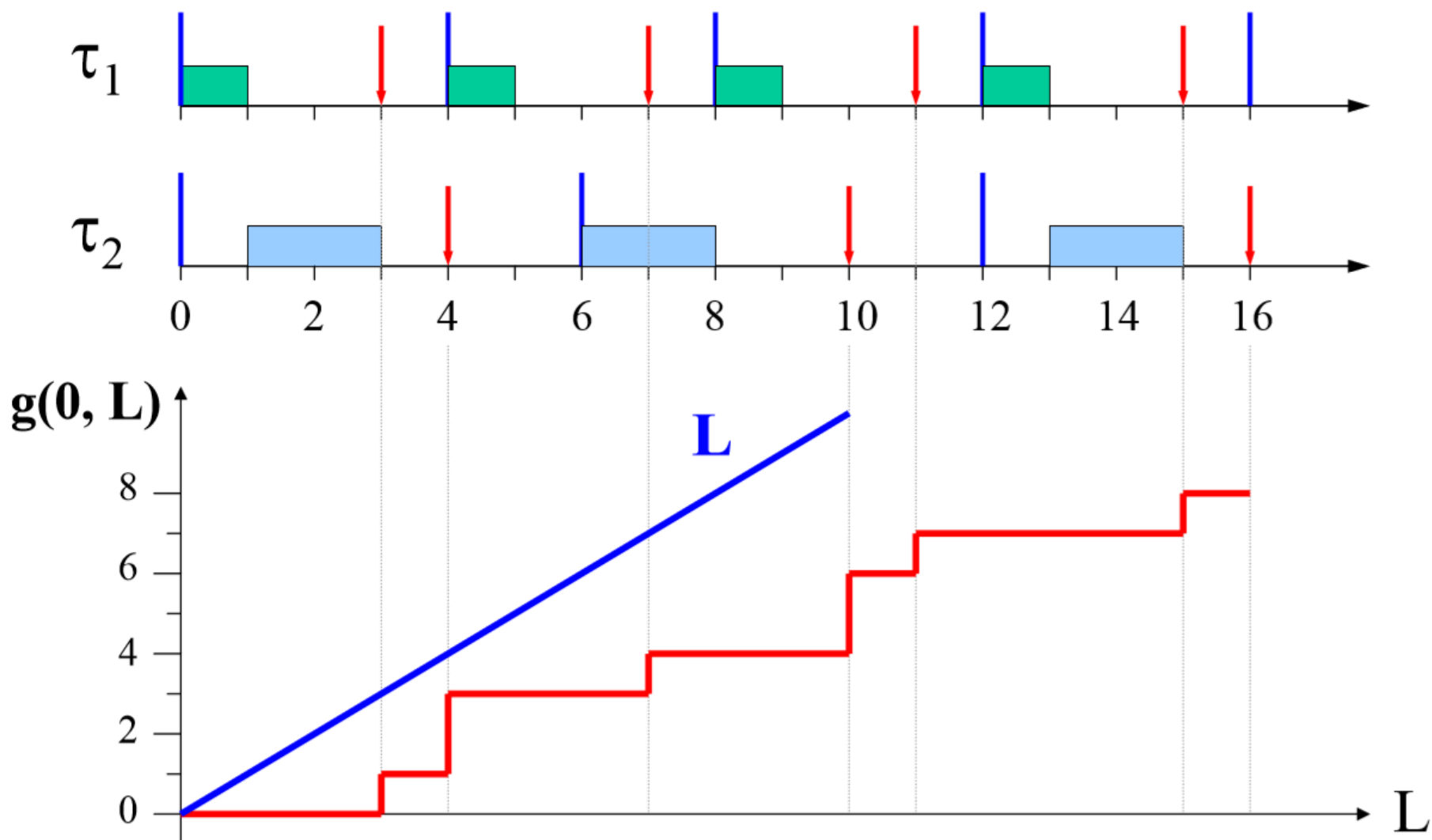
For synchronous task sets we can only analyze intervals $[0, L]$



Processor Demand Test

$$\forall L > 0 \quad \sum_{i=1}^n \left\lfloor \frac{L - D_i + T_i}{T_i} \right\rfloor C_i \leq L$$

Example



Upper Bound for PD Test

$$L_a = \max \left\{ D_1, \dots, D_N, \frac{\sum_{i=1}^N (T_i - D_i) C_i / T_i}{1 - U} \right\}$$

U is the utilisation of the task set, note upper bound not defined for U=1

[U.C. Devi. An Improved Schedulability Test for Uniprocessor Periodic Task Systems]

PD Test with Blocking

- Compute the maximum blocking time for each task
- Inflate C_i by B_i

EDF **$D = T$**

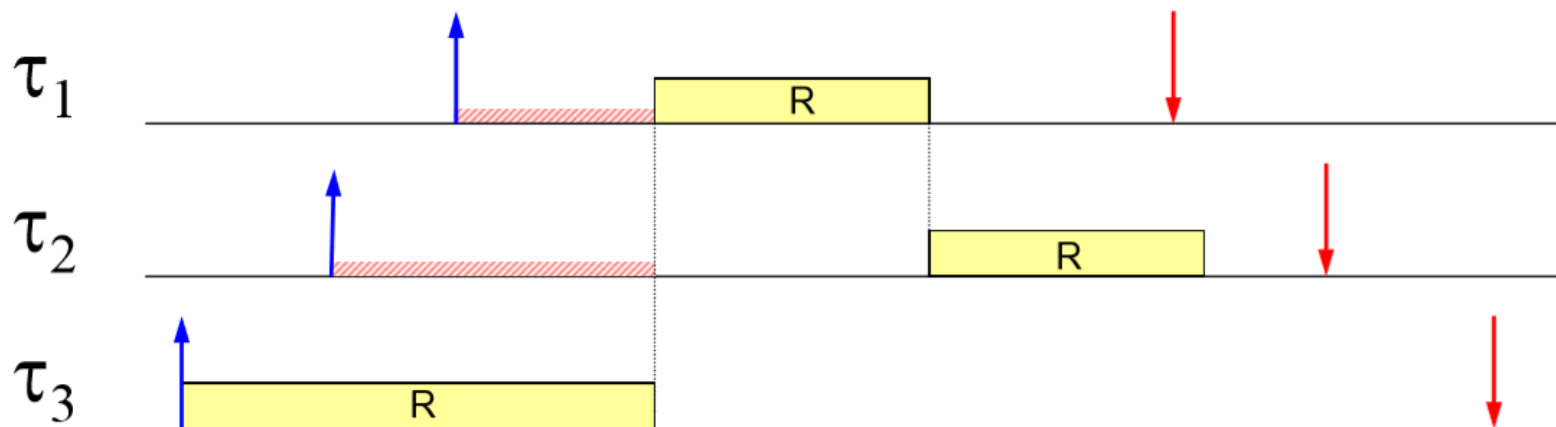
$$\forall i \quad \sum_{k=1}^{i-1} \frac{C_k}{T_k} + \frac{C_i + B_i}{T_i} \leq 1$$

EDF **$D \leq T$** task set is schedulable if $U < 1$ and

$$\forall i \quad \forall L \quad B_i + \sum_{k=1}^n \left\lfloor \frac{L + T_k - D_k}{T_k} \right\rfloor C_k \leq L$$

Non-preemptive scheduling

It is a special case of preemptive scheduling where all tasks share a single resource for their entire duration.



The max blocking time for task τ_i is given by the largest C_k among the lowest priority tasks:

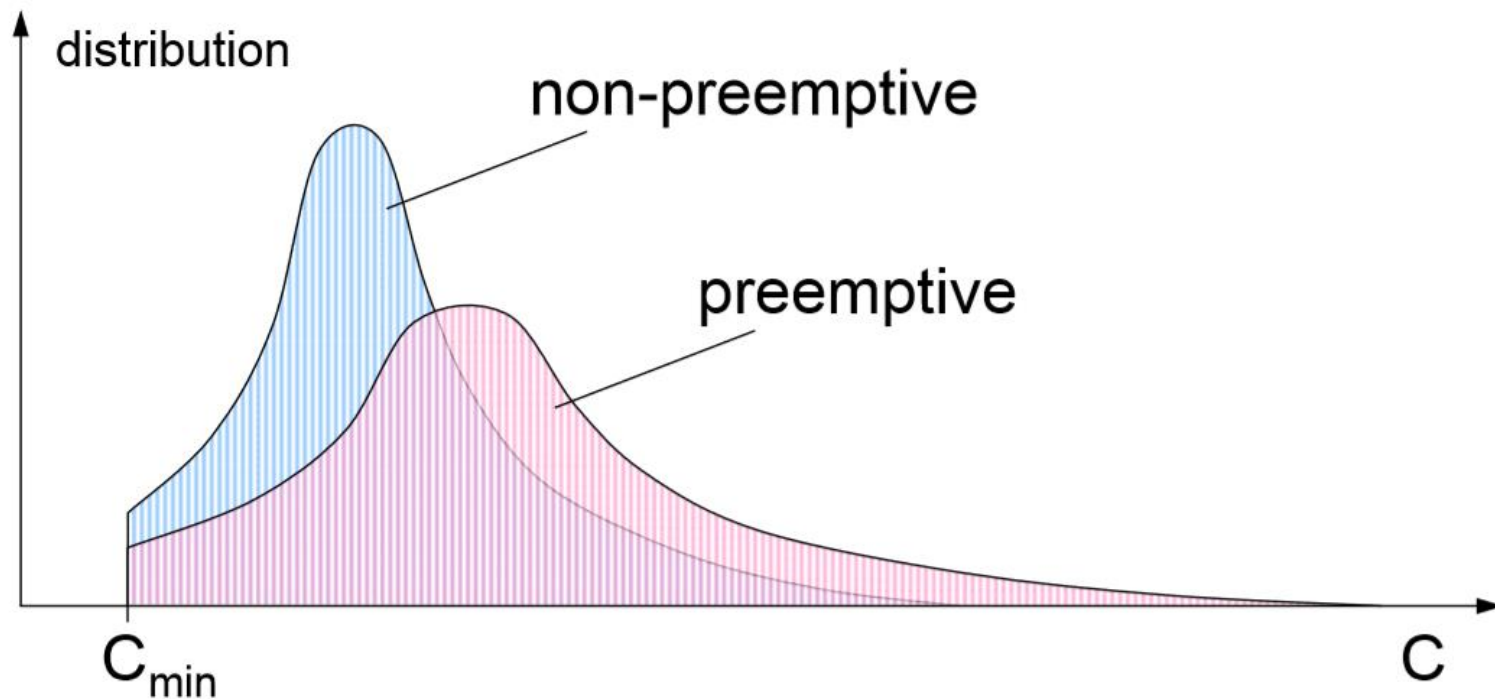
$$B_i = \max\{C_k : P_k < P_i\}$$

Advantages of NP scheduling

- It reduces runtime overhead
 - Less context switches
 - No semaphores are needed for critical sections
- It reduces stack size, since no more than one task can be in execution.
- It preserves program locality, improving the effectiveness of
 - Cache memory
 - Pipeline mechanisms
 - Prefetch queues

Advantages of NP scheduling

- As a consequence, task execution times are
 - Smaller
 - More predictable

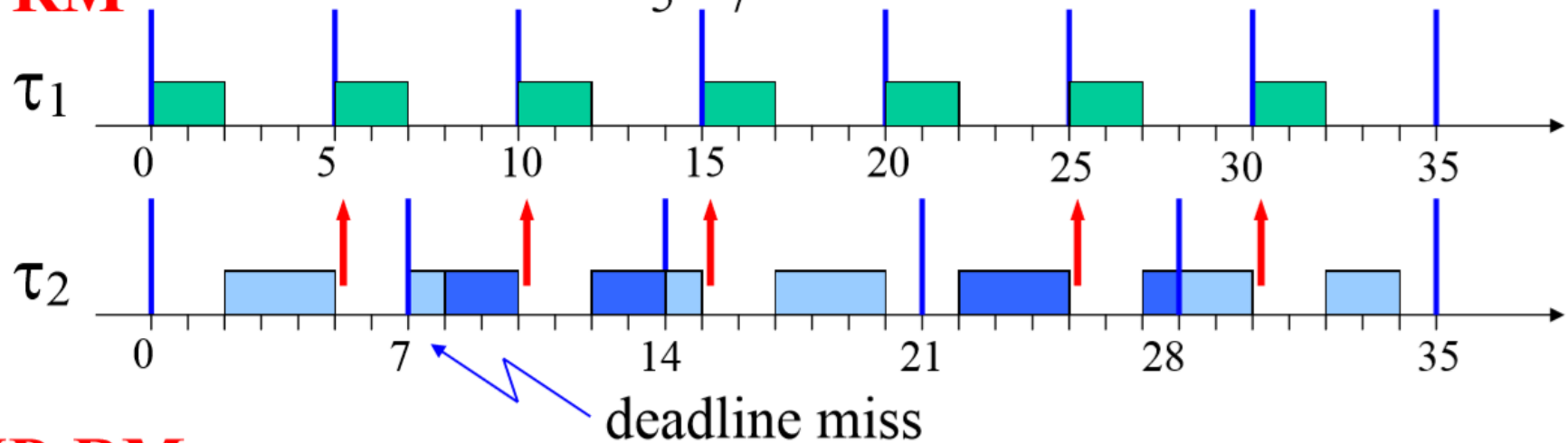


Advantages of NP scheduling

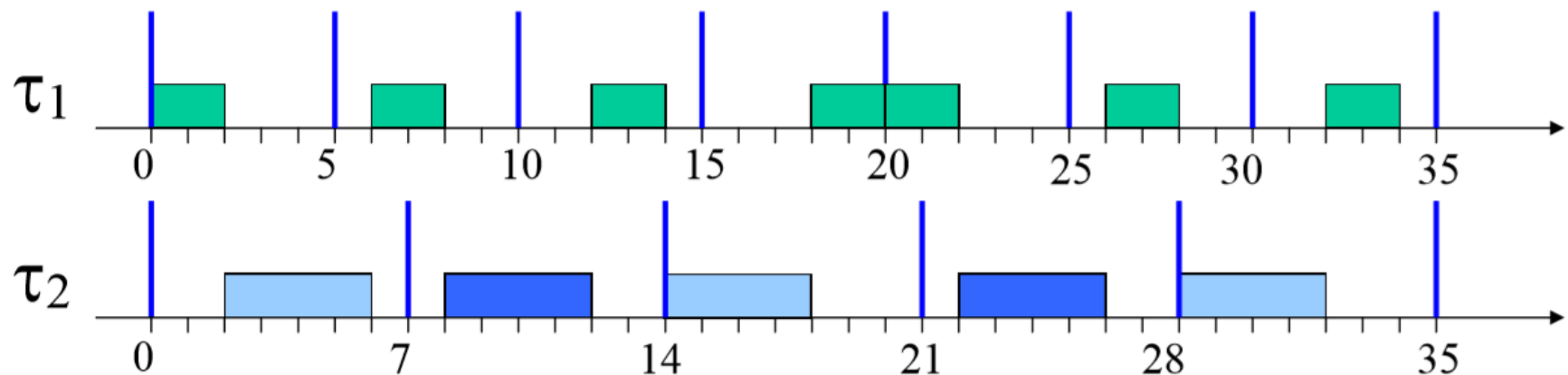
In fixed priority systems can improve schedulability:

$$U = \frac{2}{5} + \frac{4}{7} \cong 0.97$$

RM

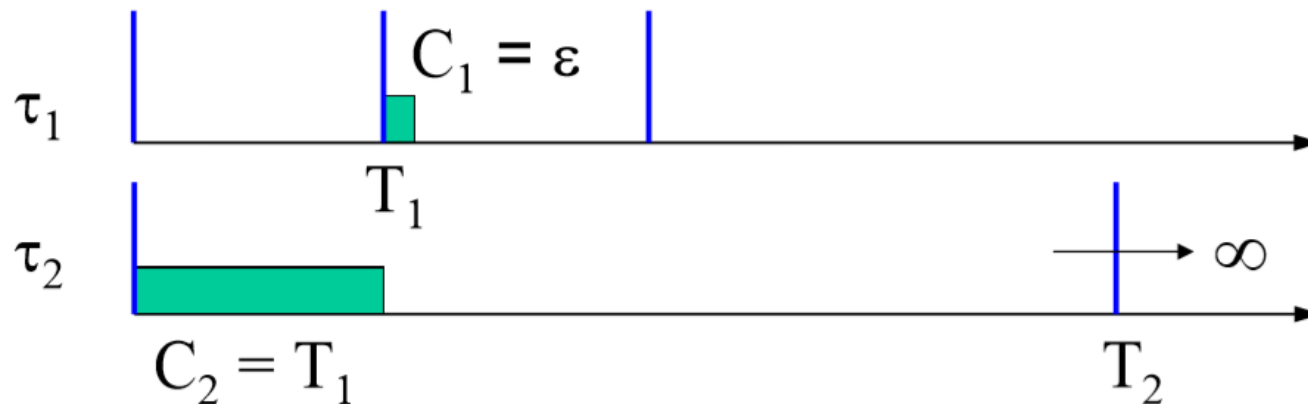


NP-RM



Disadvantages of NP scheduling

- In general, NP scheduling reduces schedulability.
- The utilization bound under non preemptive scheduling drops to zero:

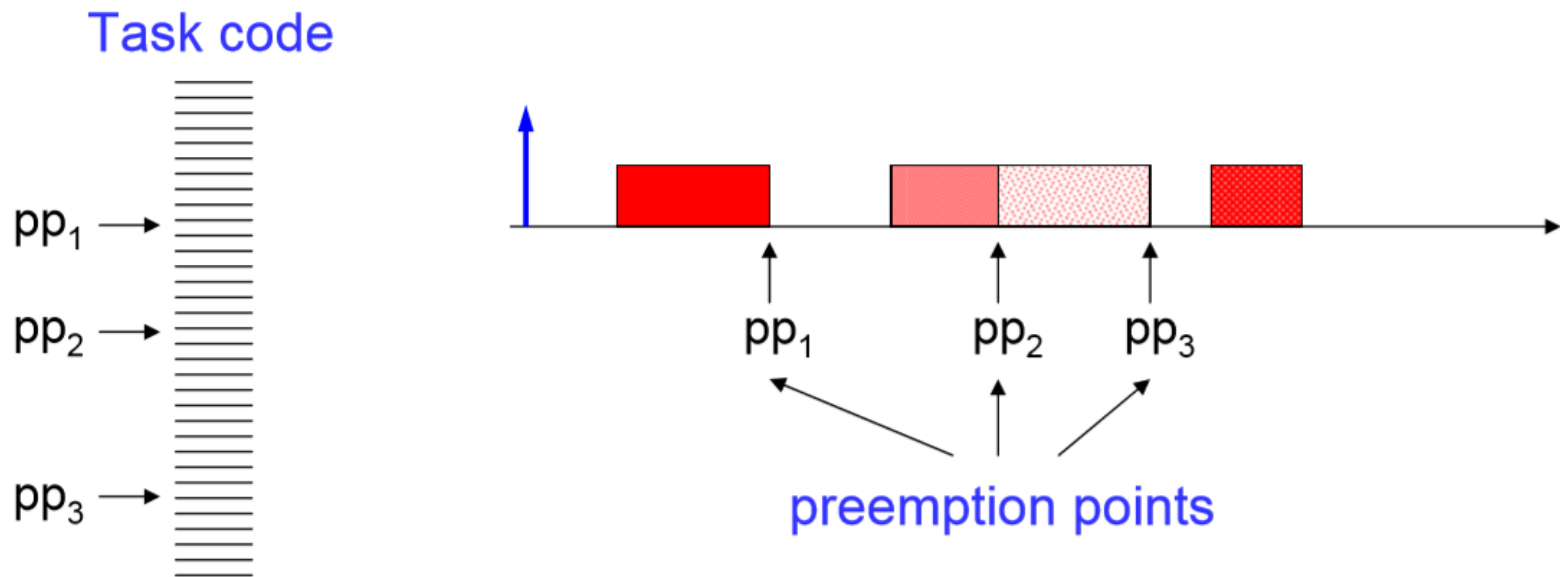


$$U = \frac{\varepsilon}{T_1} + \frac{C_2}{\infty} \rightarrow 0$$

Trade-off solutions

Tunable Preemptive Systems

- Compute the longest non-preemptive section that allows a feasible schedule
- Allow preemption only in certain points in the code.



Handling Jitter & Delay

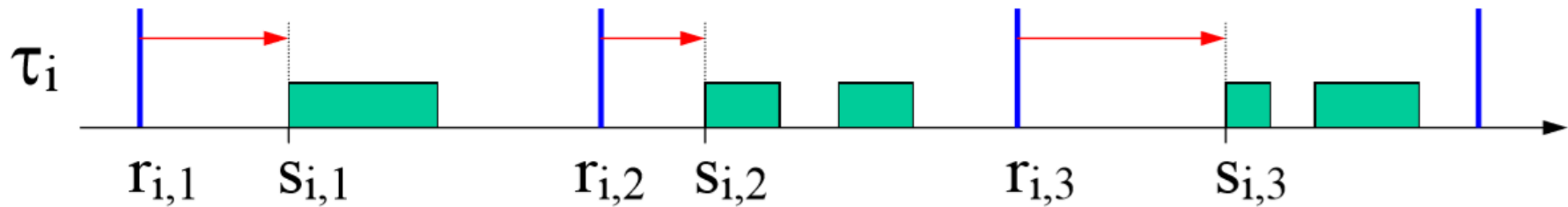
Jitter for an event

The maximum time variation in the occurrence of a particular event in two consecutive jobs.

In many control applications, delay and jitter can cause instability or jerky behavior

Definitions

Start time delay (Input Latency): $INL_{i,k} = s_{i,k} - r_{i,k}$



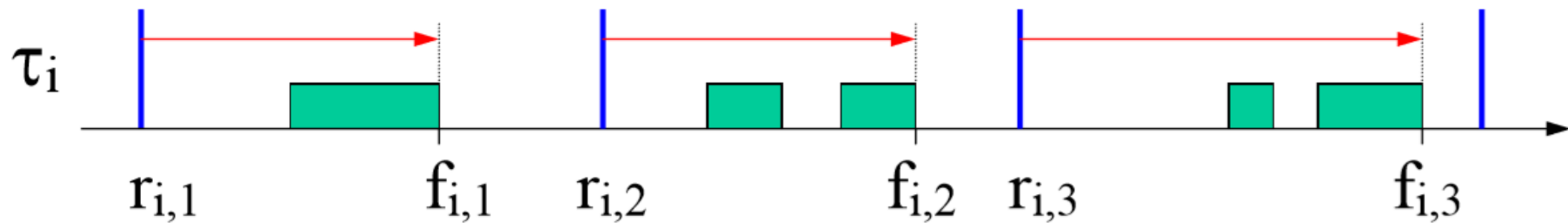
Start time Jitter (Input Jitter):

Absolute: $INJ_i^{abs} = \max_k (s_{i,k} - r_{i,k}) - \min_k (s_{i,k} - r_{i,k})$

Relative: $INJ_i^{rel} = \max_k \left| (s_{i,k} - r_{i,k}) - (s_{i,k-1} - r_{i,k-1}) \right|$

Definitions

Response Time (Output Latency): $R_{i,k} = f_{i,k} - r_{i,k}$



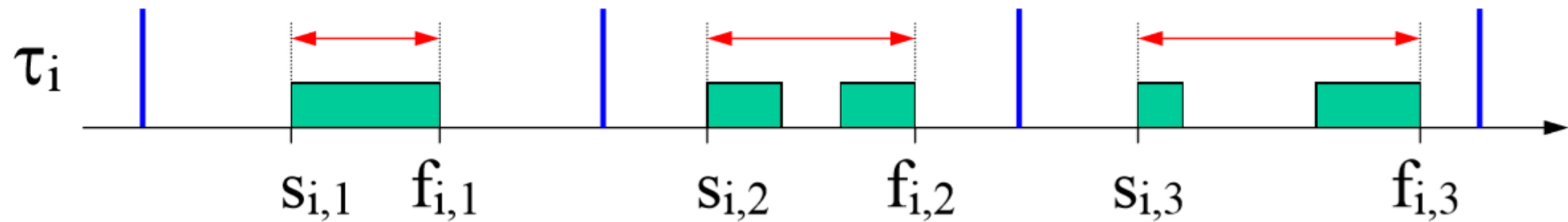
Response Time Jitter (Output Jitter):

Absolute: $RTJ_i^{abs} = \max_k (f_{i,k} - r_{i,k}) - \min_k (f_{i,k} - r_{i,k})$

Relative: $RTJ_i^{rel} = \max_k \left| (f_{i,k} - r_{i,k}) - (f_{i,k-1} - r_{i,k-1}) \right|$

Definitions

Input-Output Latency: $\text{IOL}_{i,k} = f_{i,k} - s_{i,k}$

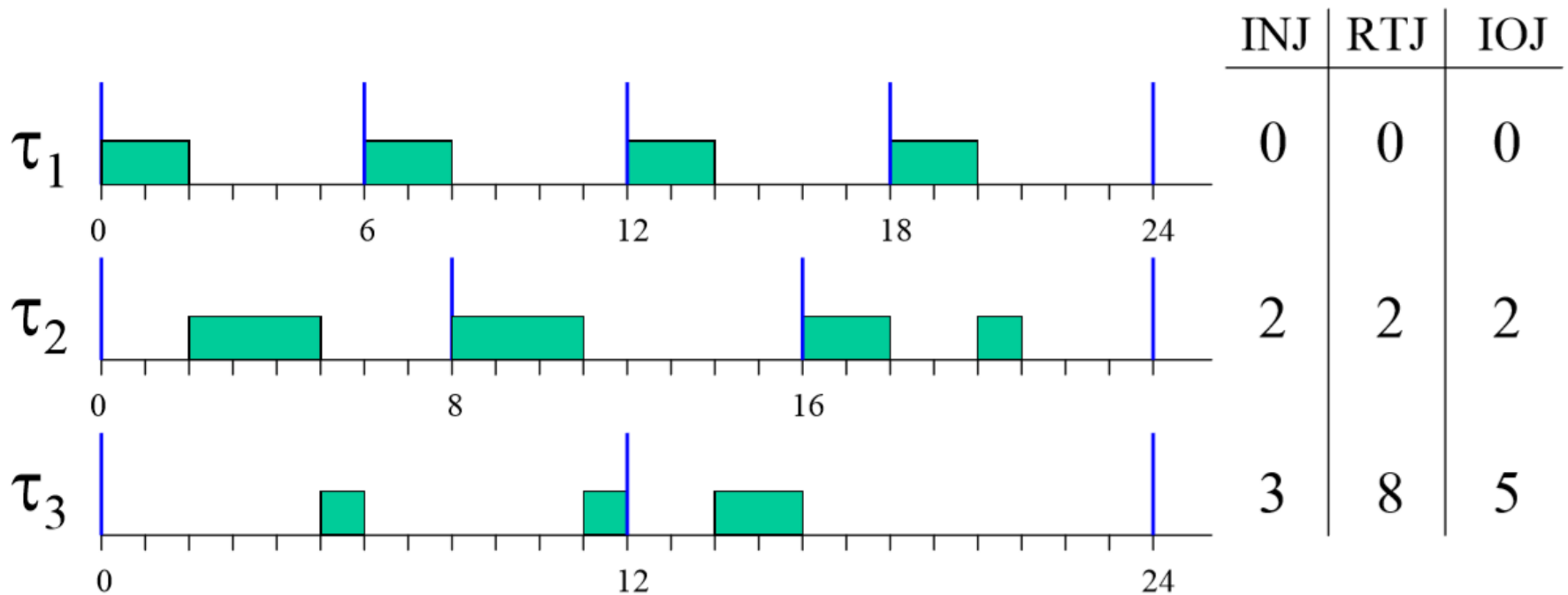


Input-Output Jitter:

Absolute: $\text{IOJ}_i^{\text{abs}} = \max_k (f_{i,k} - s_{i,k}) - \min_k (f_{i,k} - s_{i,k})$

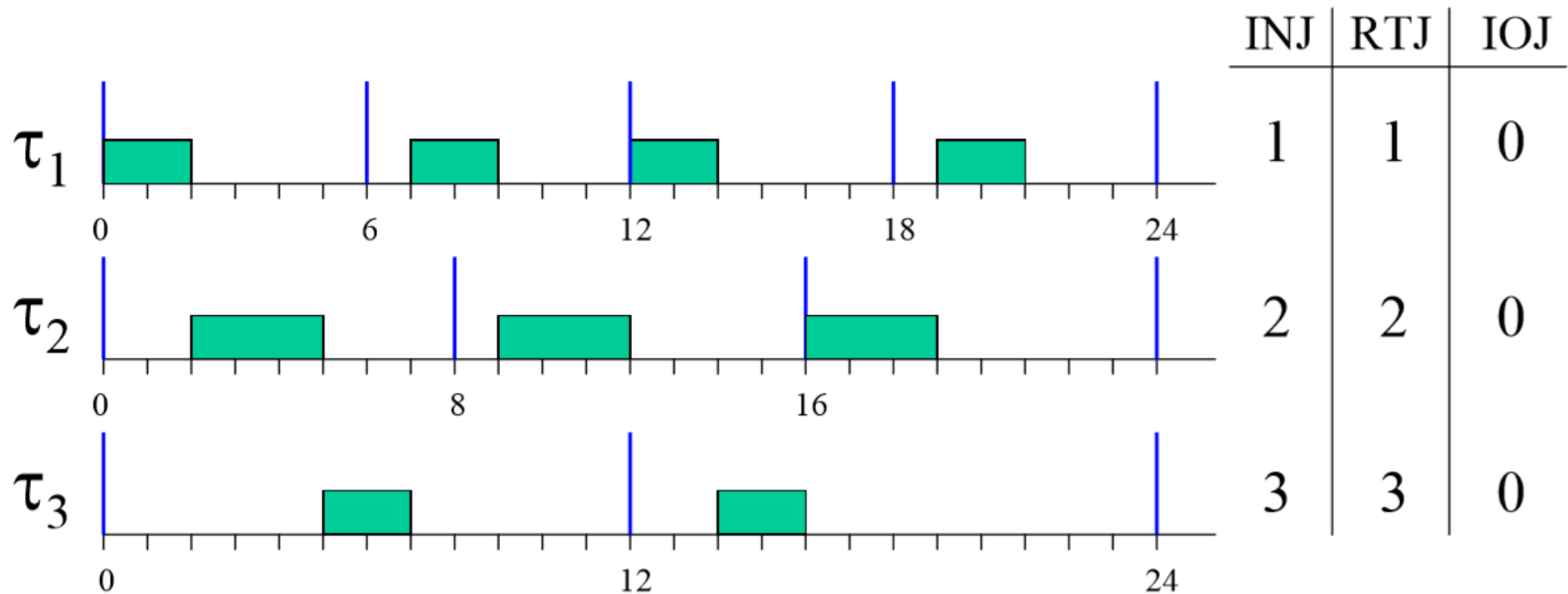
Relative: $\text{IOJ}_i^{\text{rel}} = \max_k \left| (f_{i,k} - s_{i,k}) - (f_{i,k-1} - s_{i,k-1}) \right|$

Jitter under RM



Low priority tasks experience very high delay and jitter

Jitter under EDF



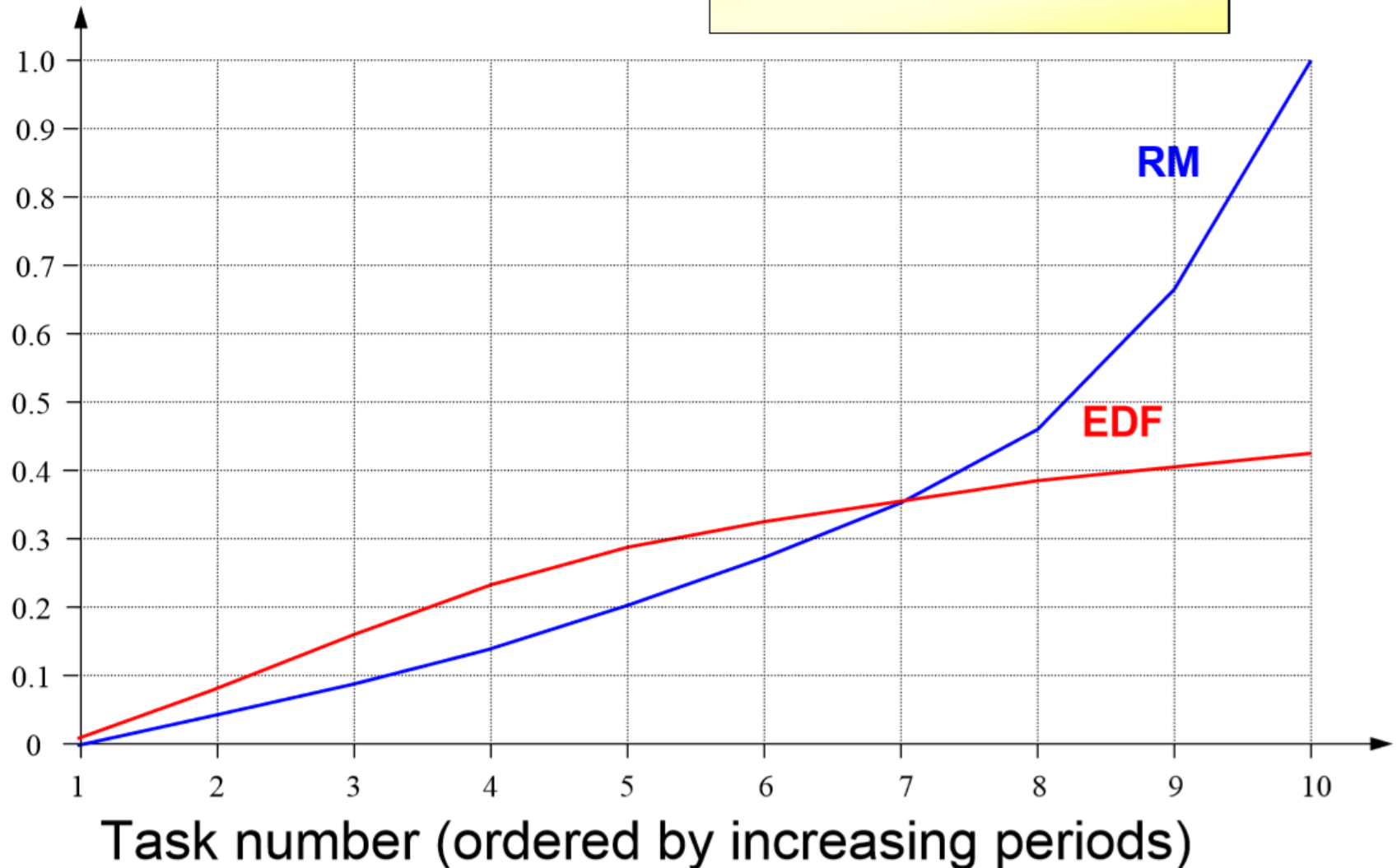
For a little increase of RTJ_1 , RTJ_3 decreases a lot

$IOJ = 0$ for all the tasks

Jitter under RM and EDF

Normalized Avg. RTJ

$U = 0.9$ $N = 10$



How to handle delay and jitter

Two main methods can be used to reduce the effect of delay and jitter:

1. **compensate** them by proper control actions;
2. **reduce** them as much as possible.

Even when compensation is used, reducing delay and jitter improves system performance



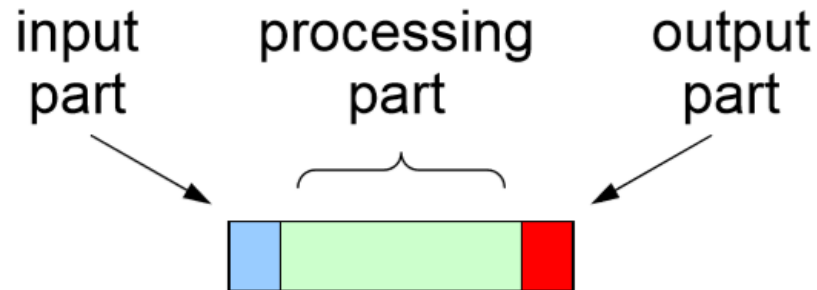
Hence we concentrate on **reduction methods**

Jitter Reduction methods

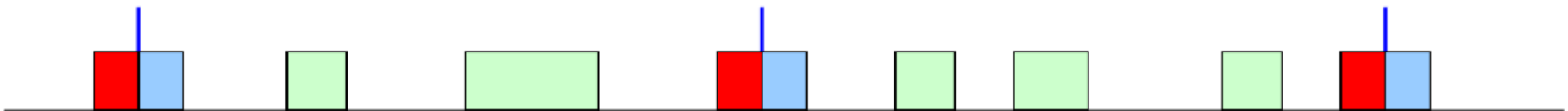
Three methods can be used to reduce the jitter caused by task interference:

1. Task Splitting
2. Advancing Deadlines
3. Non Preemptive Scheduling

Reducing Jitter by Task Splitting



The idea is to force **input** and **output** parts to execute in a time-triggered fashion, using timers:



Reducing Jitter by Task Splitting

Advantages

1. Jitter is reduced at the minimum possible value;
2. If input and output parts are small, this method is effective for any task, independently of the scheduler and task parameters.

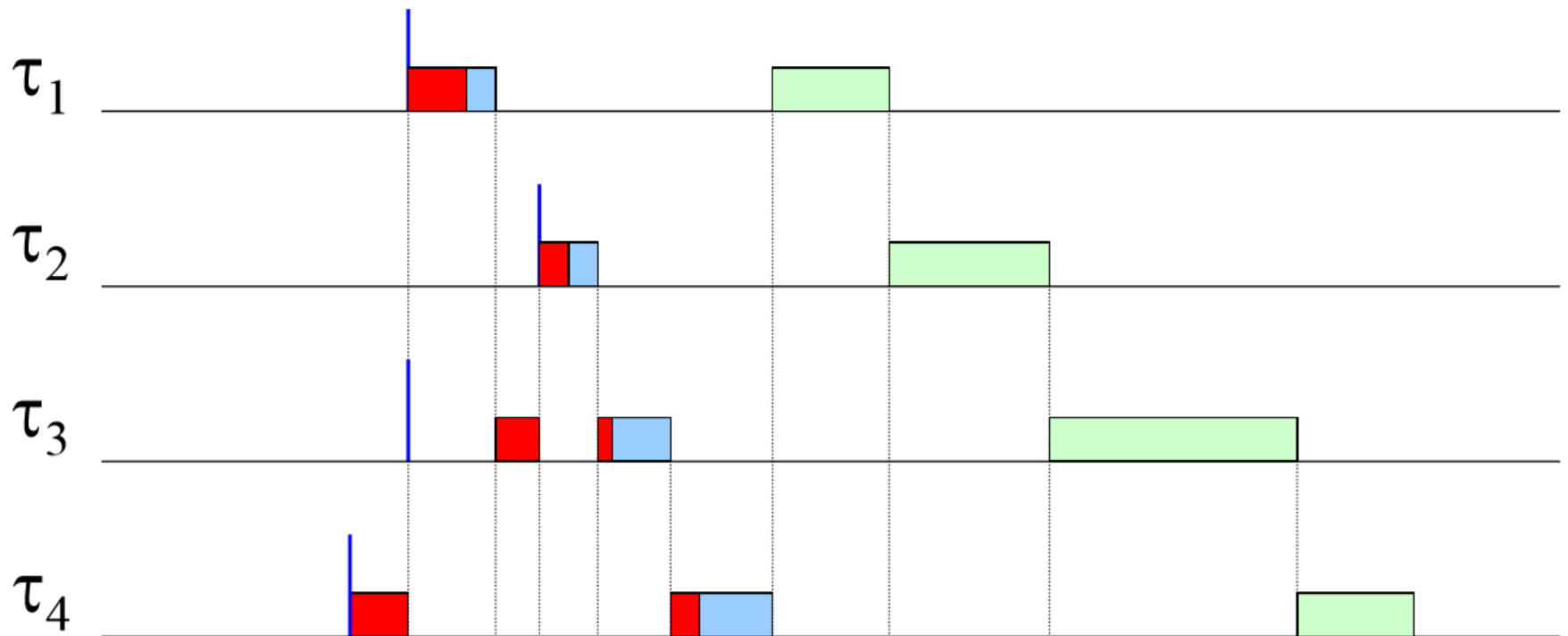
Reducing Jitter by Task Splitting

Disadvantages

1. Extra effort to be implemented;
2. Jitter is reduced at the expense of delay;
3. Input and output parts create extra interference which complicates the analysis and reduces schedulability;
4. Input and output parts may compete and need to be scheduled with some policy.

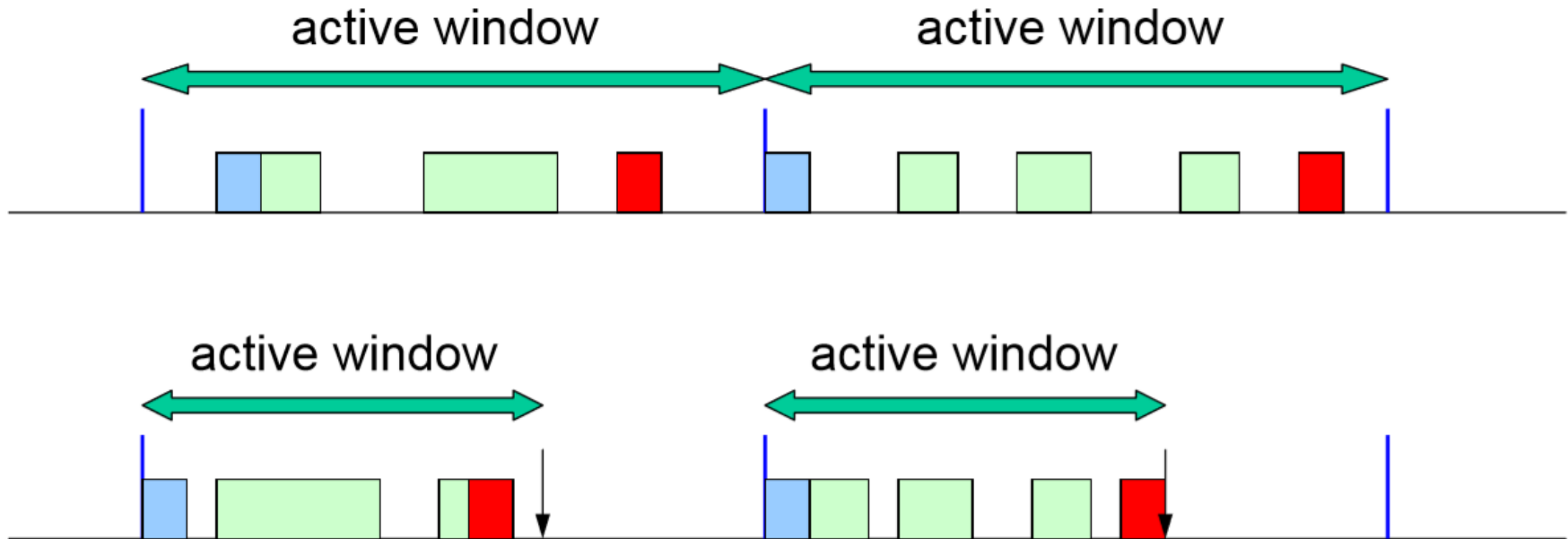
Reducing Jitter by Task Splitting

Interfering I/O parts



Reducing Jitter by Advancing Deadlines

The idea is to advance task deadlines to reduce the active window in which jobs can be executed:



Reducing Jitter by Advancing Deadlines

Advantages

1. Easy to implement (no special support is required from the OS);
2. No extra interference caused by additional timer interrupts;
3. Both delay and jitter are reduced!!

Reducing Jitter by Advancing Deadlines

Disadvantages

1. Not all tasks can reduce jitter to zero. A further reduction can be achieved by proper offsets, but the analysis requires exponential complexity.
2. Advancing deadlines reduces system schedulability.

Reducing Jitter by Non Preemption

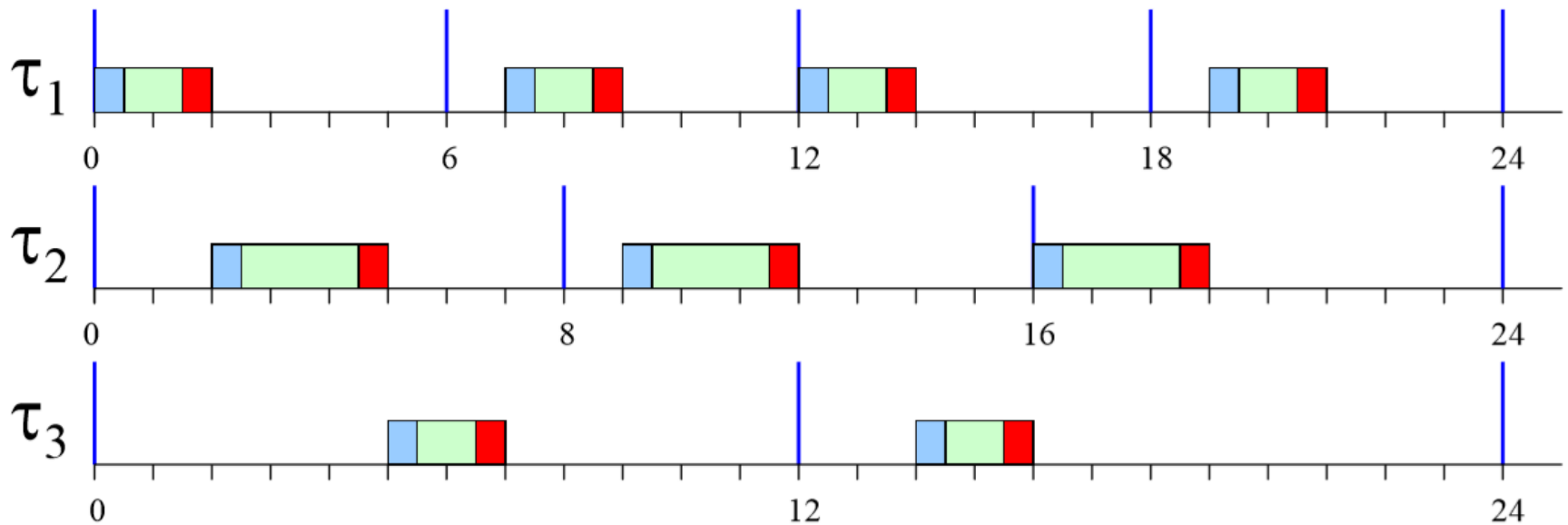
Disabling preemptions a task can be delayed, but once started cannot be interrupted:



$$\forall_k \begin{cases} IOL_{i,k} = C_i \\ IOJ_{i,k} = 0 \end{cases}$$

Reducing Jitter by Non Preemption

Example with 3 tasks



Reducing Jitter by Non Preemption

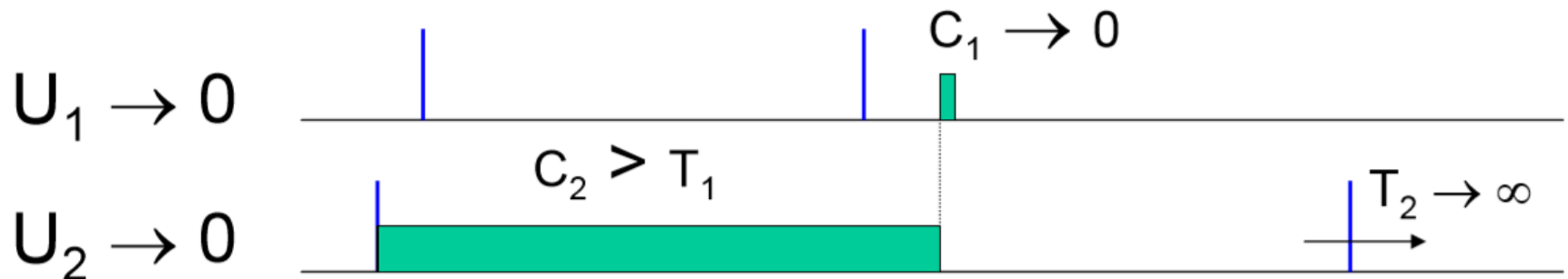
Advantages

1. $IOJ_i = 0$ for all tasks;
2. $IOL_i = C_i$ for all tasks, simplifying the use of delay compensation techniques;
3. Non preemptive execution also simplifies resource management (there is no need to protect critical sections).
4. Non preemptive execution allows stack sharing.

Reducing Jitter by Non Preemption

Disadvantages

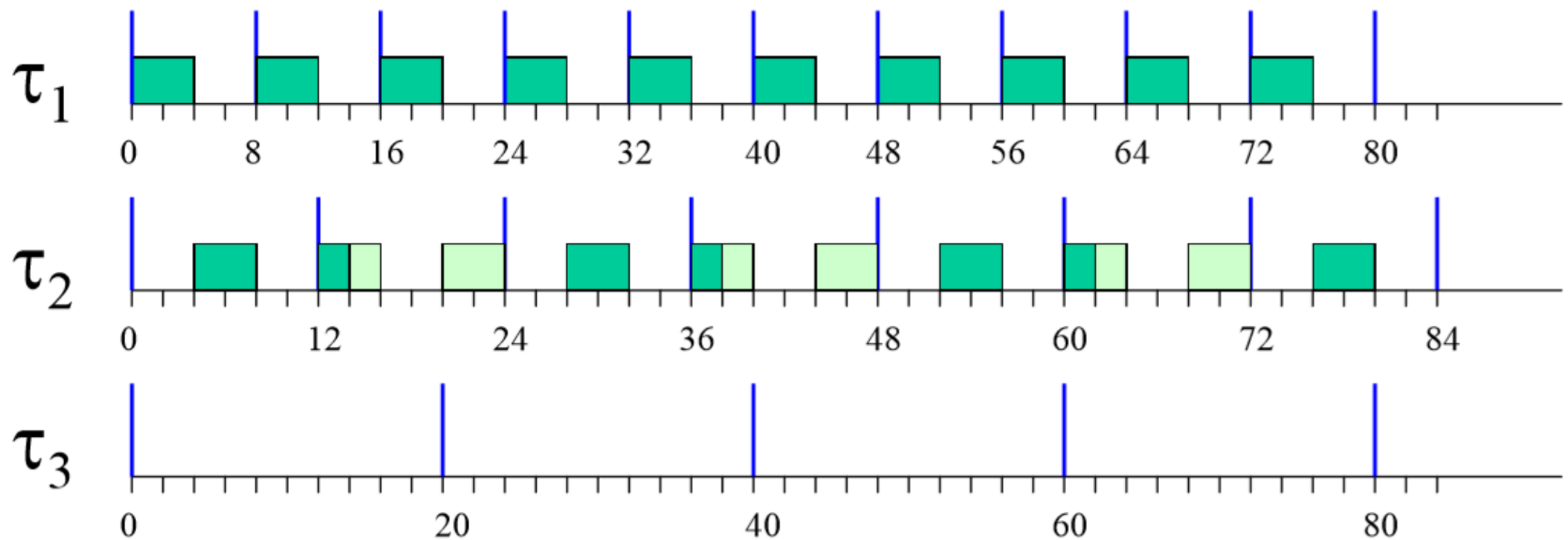
1. Non preemption reduces schedulability (analysis must take blocking times into account);
2. The utilization upper bound drops to zero:



Scheduling under overload conditions

RM under overloads

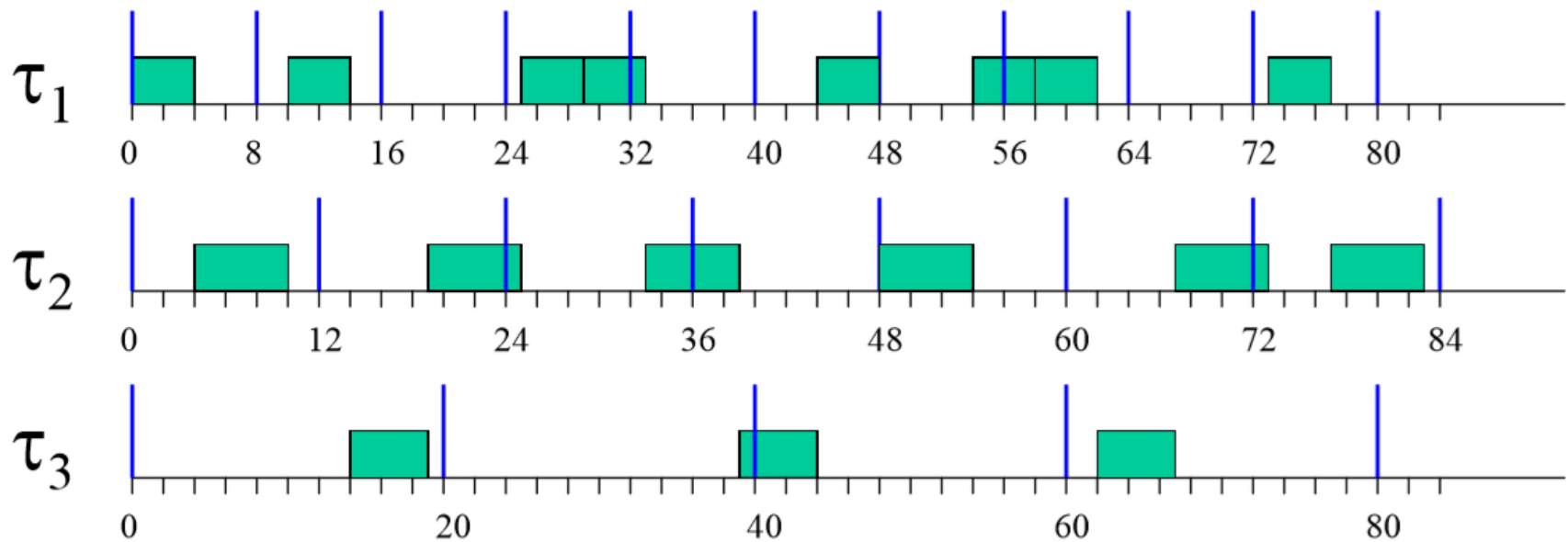
$$U = \frac{4}{8} + \frac{6}{12} + \frac{5}{20} = 1.25$$



- High priority tasks execute at the proper rate
- Low priority tasks are completely blocked

EDF under overloads

$$U = \frac{4}{8} + \frac{6}{12} + \frac{5}{20} = 1.25$$



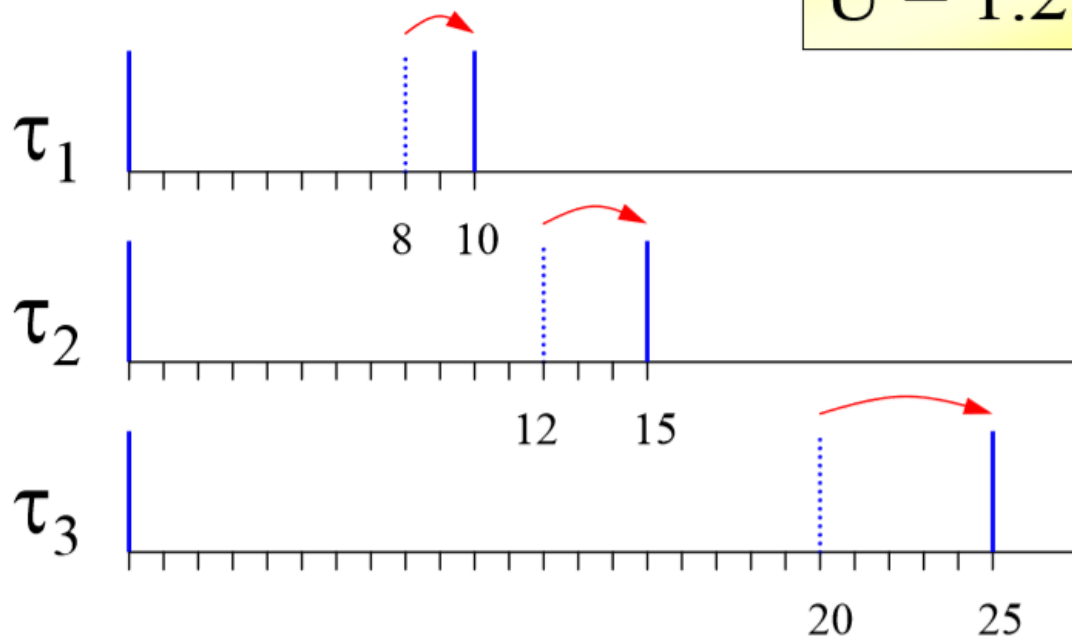
- All tasks execute at a slower rate
- No task is blocked

EDF under overloads

Theorem (Cervin '03)

If $U > 1$, EDF executes tasks with an average period $T'_i = T_i U$.

$$U = 1.25$$



	T_i	T'_i
τ_1	8	10
τ_2	12	15
τ_3	20	25

Exploiting control flexibility

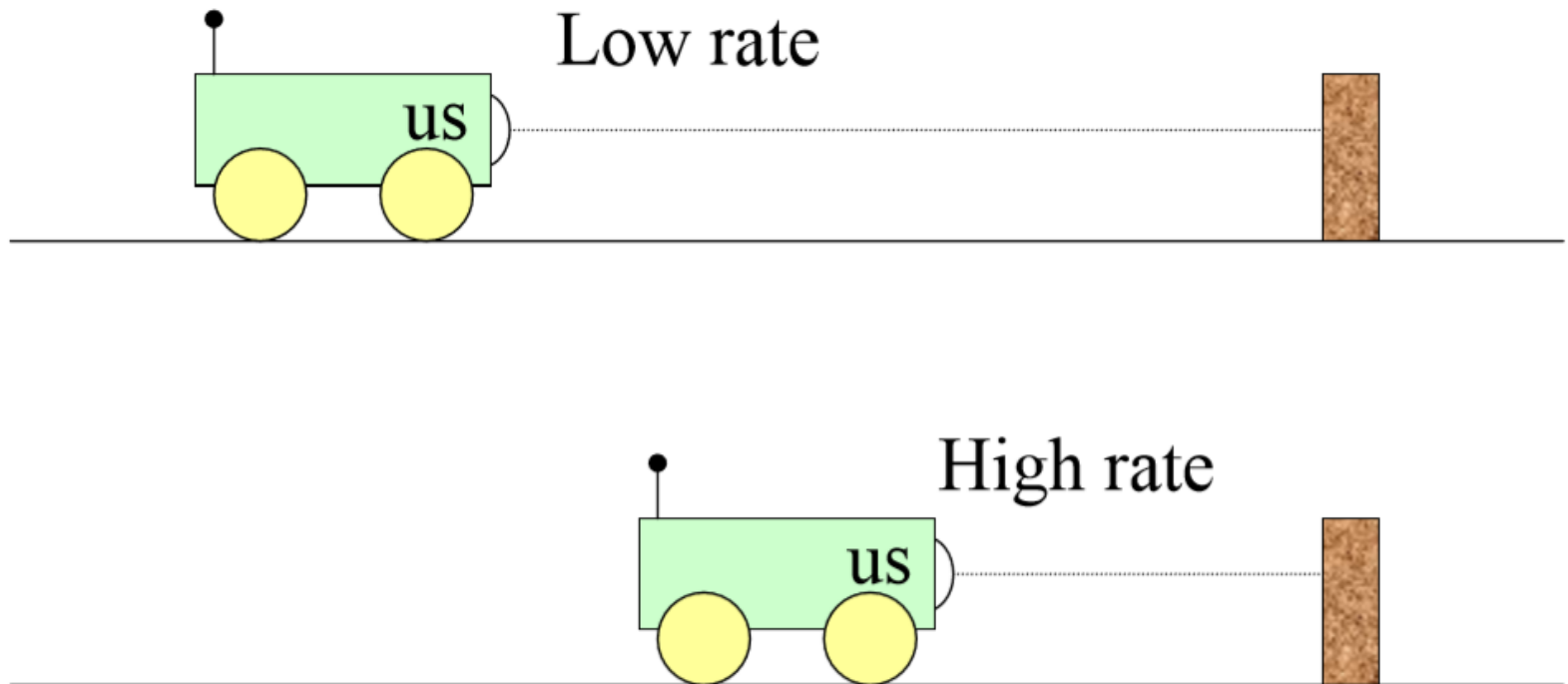
Relaxing timing constraints

- The idea is to reduce the load by increasing deadlines and/or periods.
- Each task must specify a range of values in which its period must be included.
- Periods are increased during overloads, and reduced when the overload is over.

Many control applications allow timing flexibility

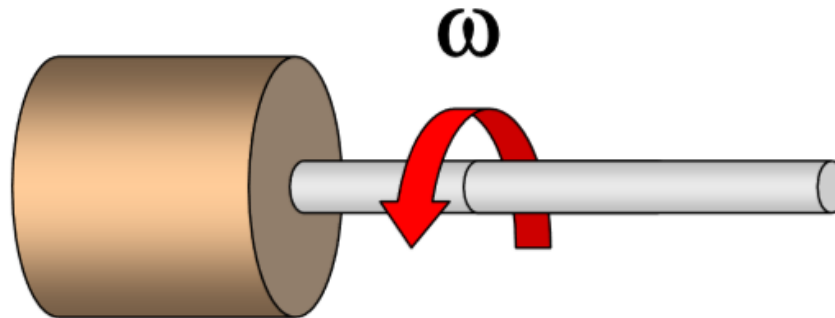
Obstacle avoidance

- The closer the obstacle, the higher the acquisition rate:



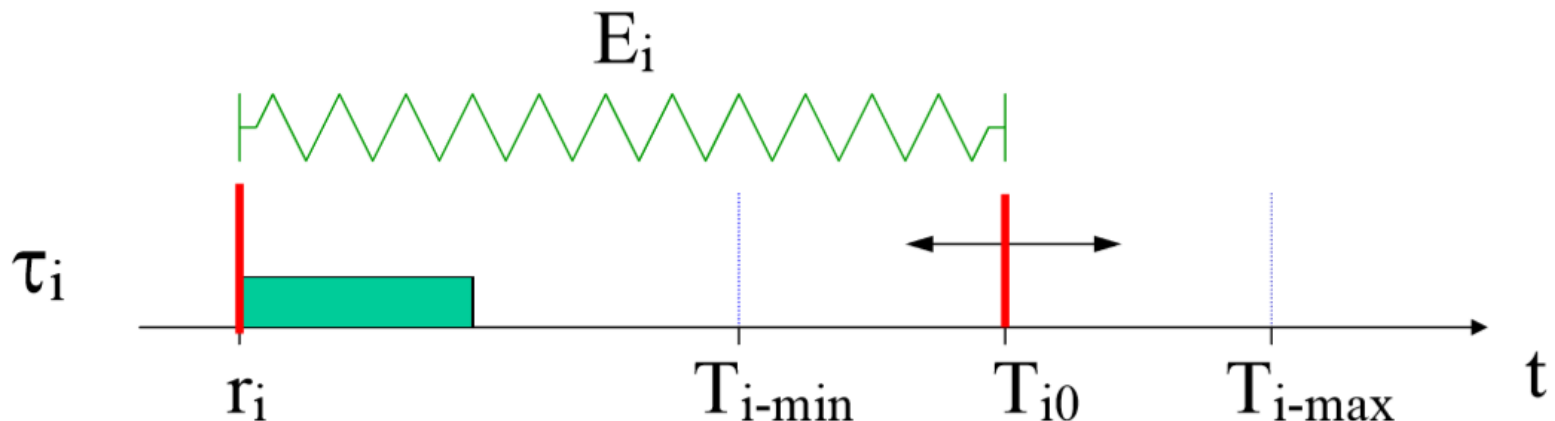
Engine control

- Some tasks need to be activated at specific angles of the motor axis:
⇒ the higher the speed, the higher the rate.



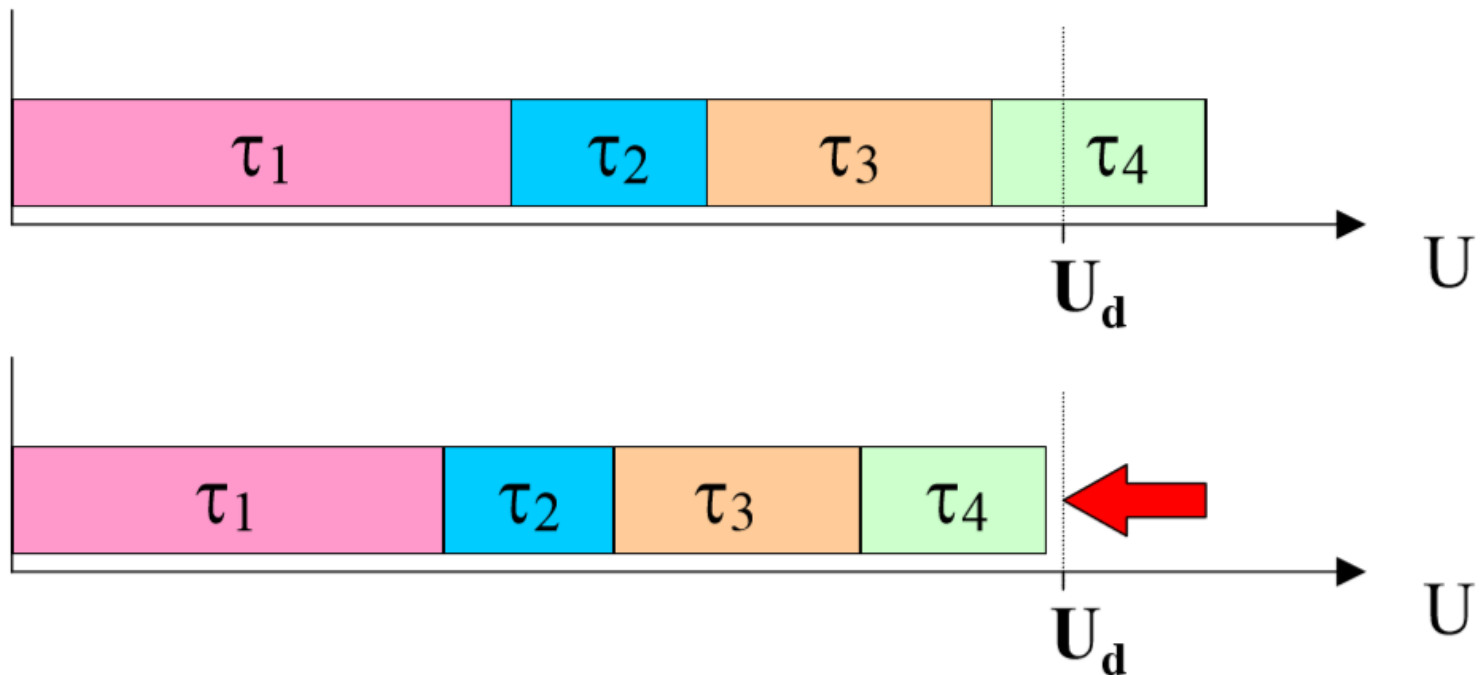
Elastic task model

- A periodic task τ_i is characterized by:
 $(C_i, T_{i-\min}, T_{i-\max}, E_i)$
- Tasks' utilizations are treated as elastic springs
- The resistance of a task to a period variation is controlled by an **elastic coefficient E_i**



Compression algorithm

During overloads, utilizations must be compressed to bring the load below a desired value U_d .



Solution for tasks

$$U_i = U_{io} - (U_0 - U_d) \frac{E_i}{E_s}$$

then:

$$T_i = \frac{C_i}{U_i}$$

Conclusions

Estimate **worst-case computation times** of tasks, using specific tools and testing.

Select an appropriate **scheduling algorithm** and a suitable **resource access protocol**.

Estimate **maximum blocking times** due non preemptive sections or mutually exclusive resources.

Apply **schedulability analysis** to verify feasibility.

Exploit system flexibility defining admissible ranges of parameters to cope with overloads.

СПАСИБО ЗА ВНИМАНИЕ

hbd@cs.msu.su

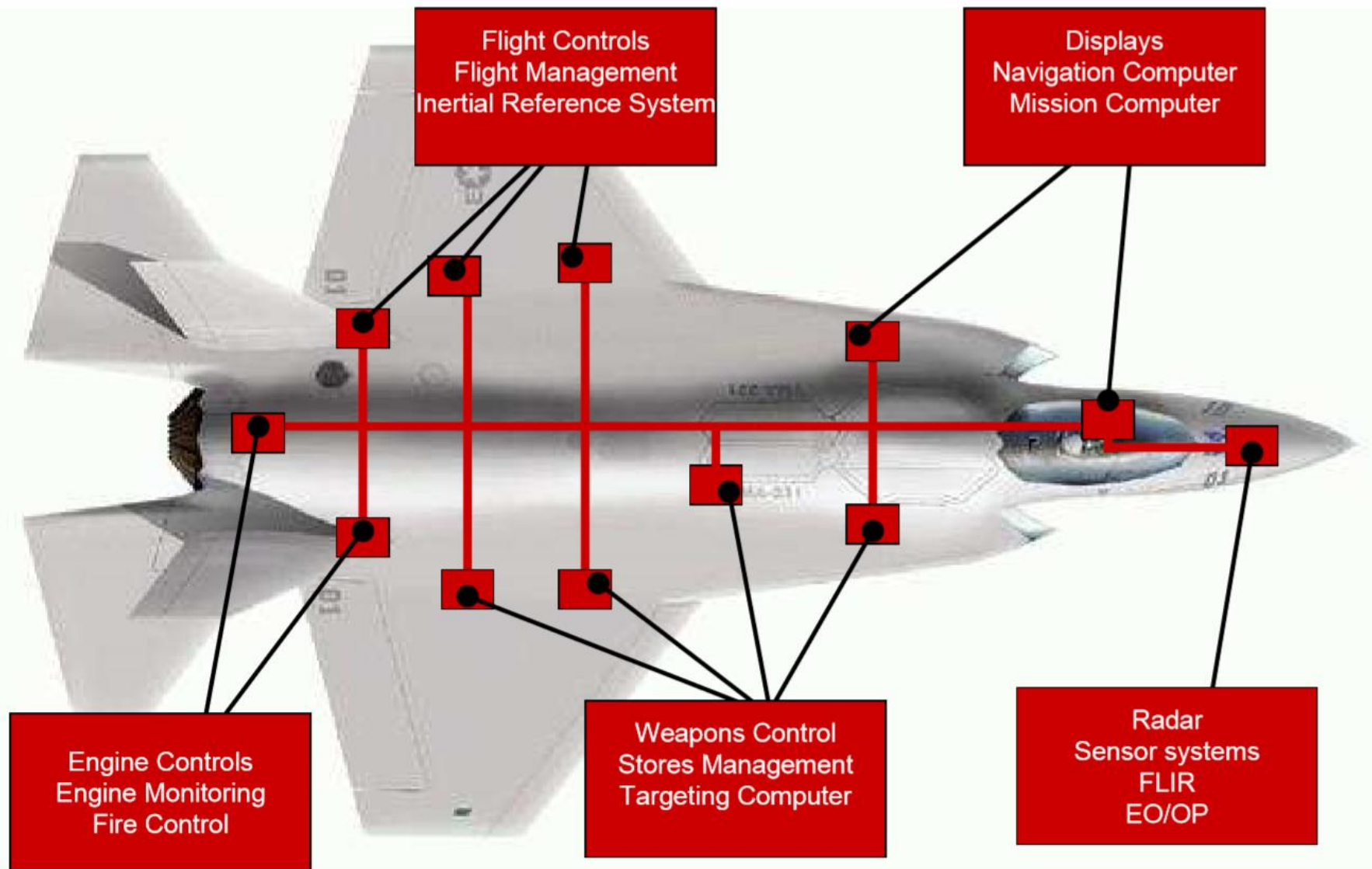
ИНФОРМАЦИОННО-УПРАВЛЯЮЩИЕ СИСТЕМЫ РЕАЛЬНОГО ВРЕМЕНИ

Лекция 4:

***Статико-динамическое планирование вычислений
в системах интегрированной модульной авионики***

Кафедра АСВК,
Лаборатория Вычислительных Комплексов
Балашов В.В.

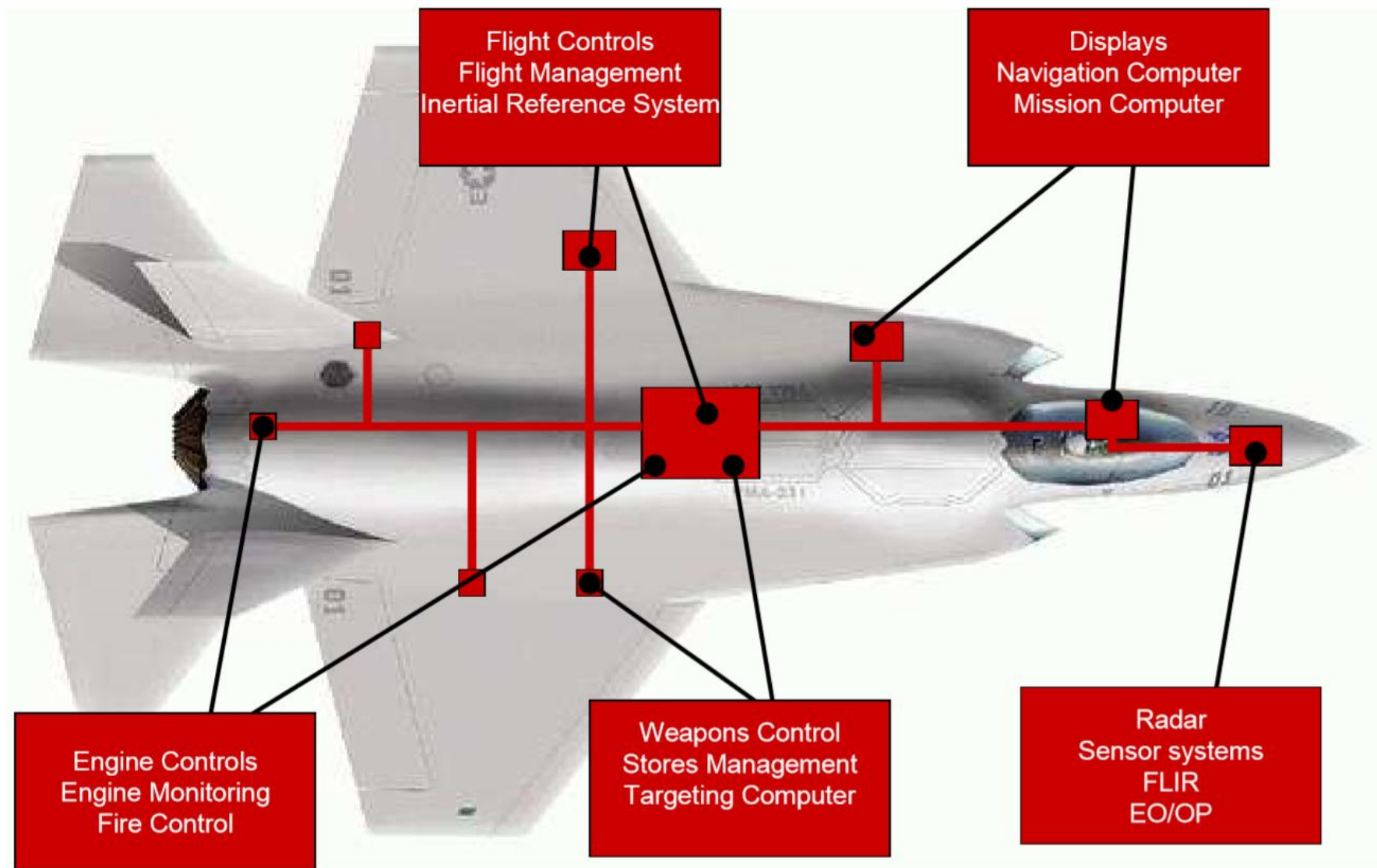
Федеративная ИУС РВ



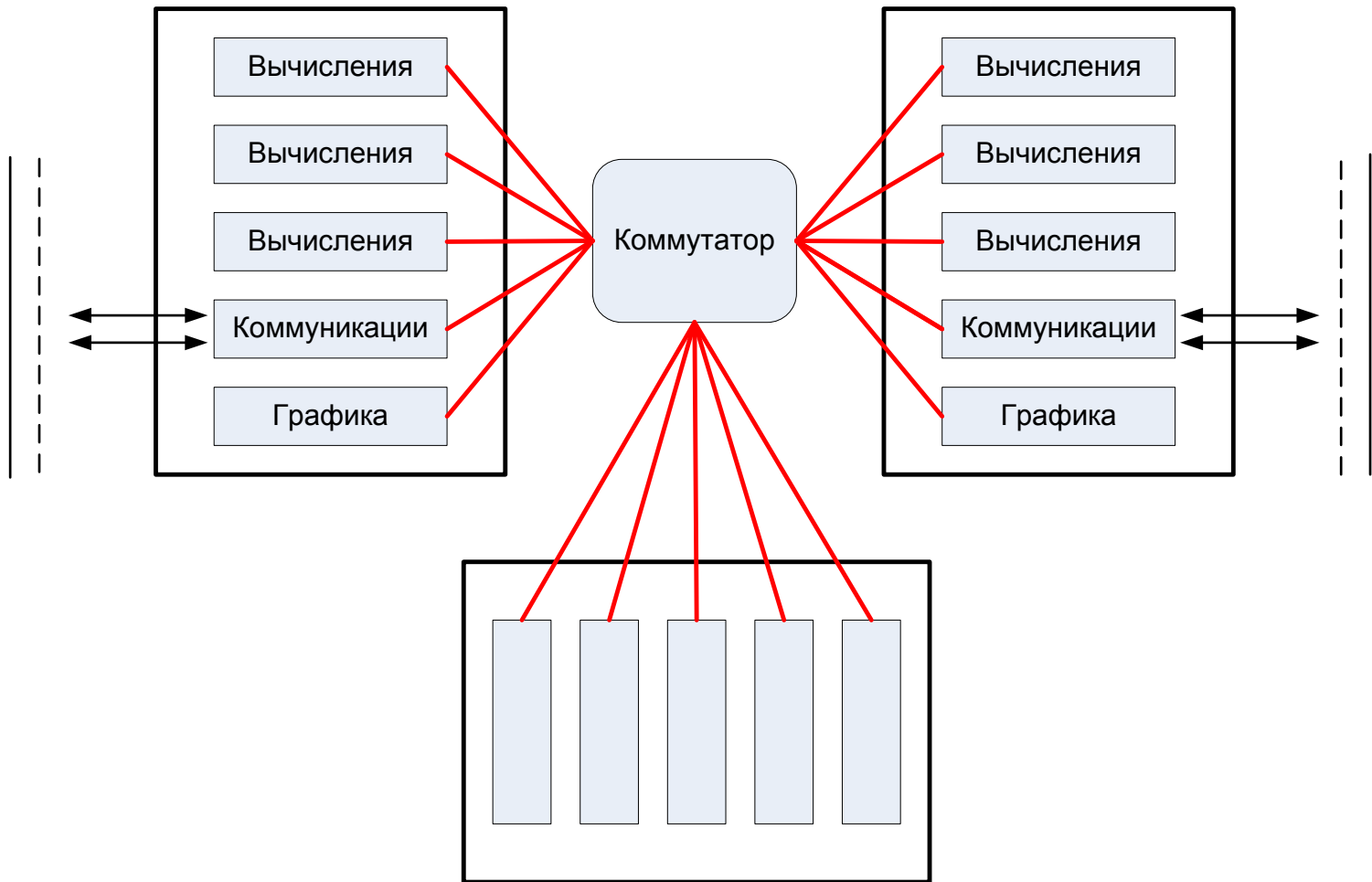
Федеративная ИУС РВ

- Специализированные блоки
 - по назначению
 - по архитектуре
- ПО различных подсистем – на различных блоках
 - изоляция по памяти
 - нет конкуренции подсистем за процессорное время
- Недостатки:
 - низкая переносимость и повторная используемость ПО
 - «зоопарк» процессорных архитектур и программных интерфейсов

ИУС РВ с архитектурой ИМА



ИУС РВ с архитектурой ИМА с сетью на базе коммутатора



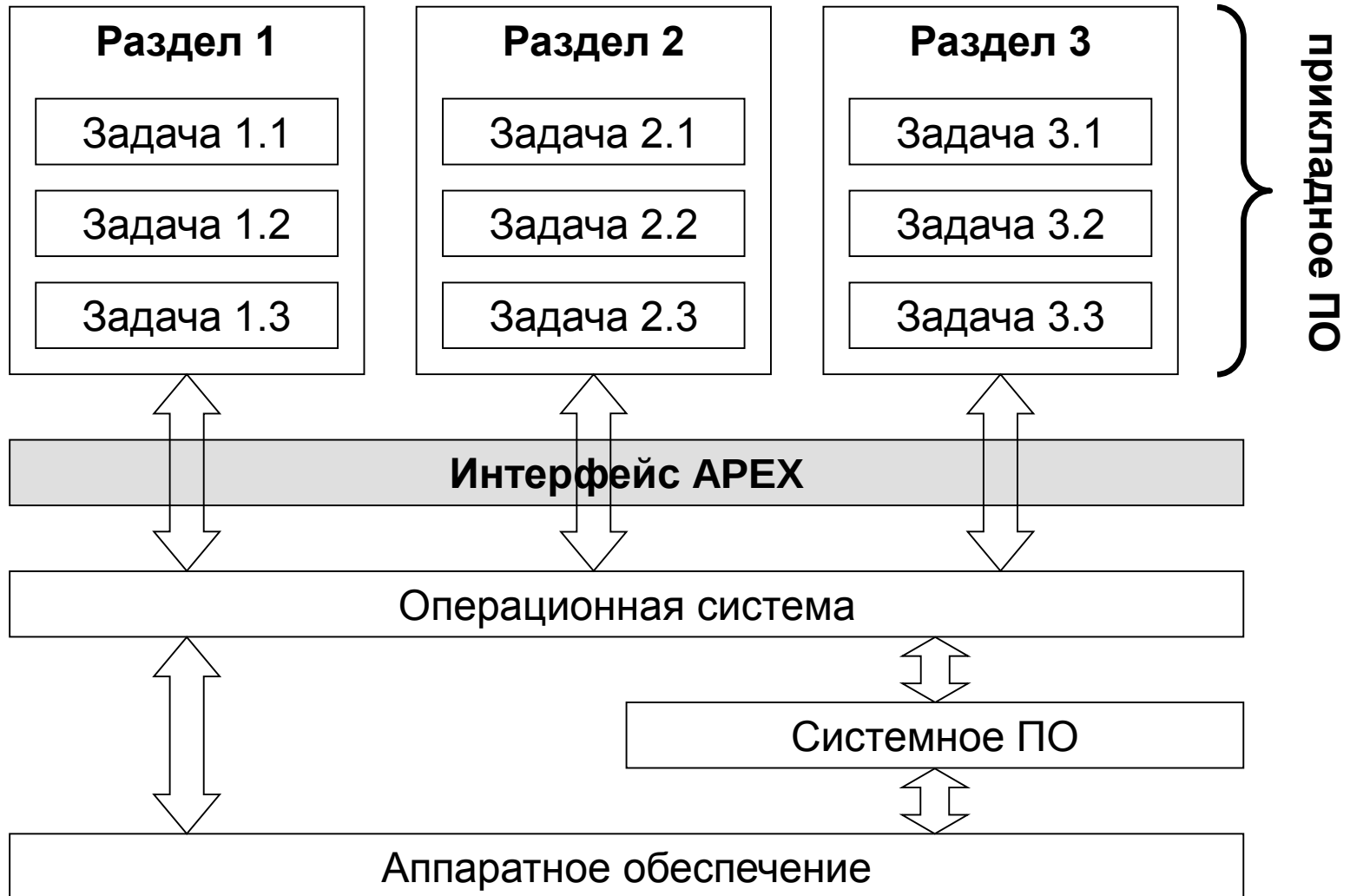
ИУС РВ с архитектурой ИМА

- Логически единый распределенный вычислитель
 - унифицированные модули нескольких типов
 - единая архитектура процессоров
 - унифицированный программный интерфейс
- Разделение вычислительных ресурсов между ПО различных подсистем
- Проблемы:
 - изоляция по памяти
 - разделение процессорного времени
- Решение: каждой подсистеме → раздел

ИМА и разработка ПО

- Особенности:
 - Стандартное API со стороны ОС
 - Статическое разделение времени, памяти и ресурсов
- Преимущества:
 - Надежность
 - Переносимость
 - Возможность повторного использования
 - Модульность
 - Упрощение верификации и сертификации

Структура ПО. Интерфейс АРЕХ



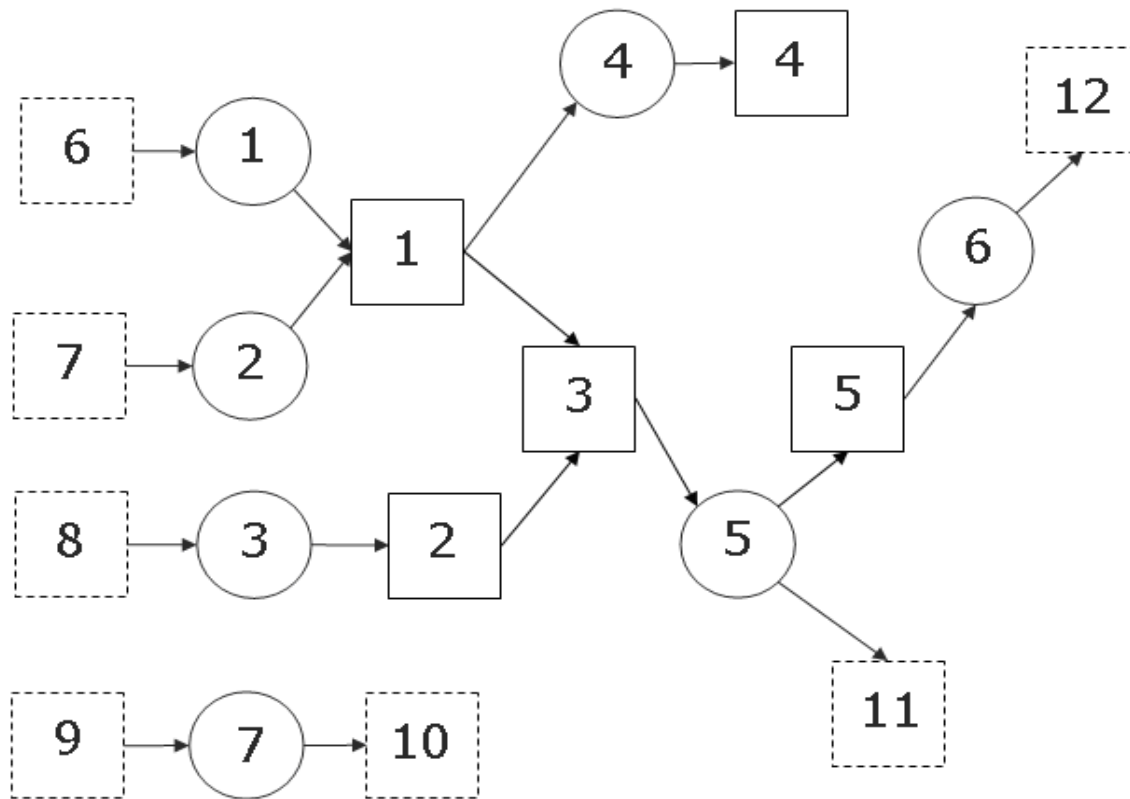
Взаимодействие между разделами

- Порты с очередью сообщений
- Порты с перезаписью сообщений
 - Буфер фиксированной длины
 - Сообщение в буфере перезаписывается
 - Отправка с заданным периодом

Взаимодействие внутри разделов

- Обмен данными
 - Передача сообщений
 - Общая память
- Механизмы синхронизации
 - Семафоры
 - События

Рабочая нагрузка

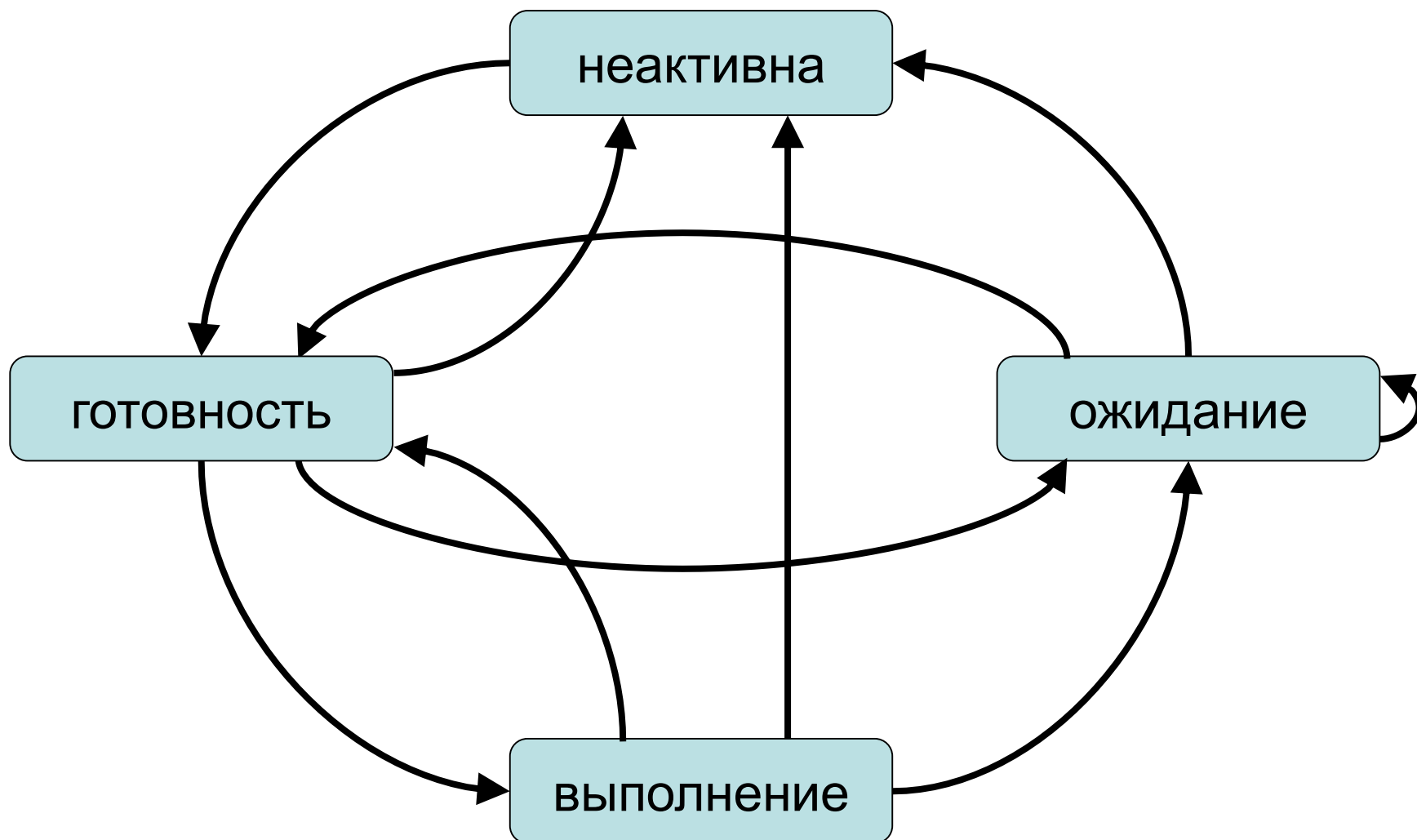


- периодические задачи
- сообщения
- зависимости по данным

Синхронизация на практике

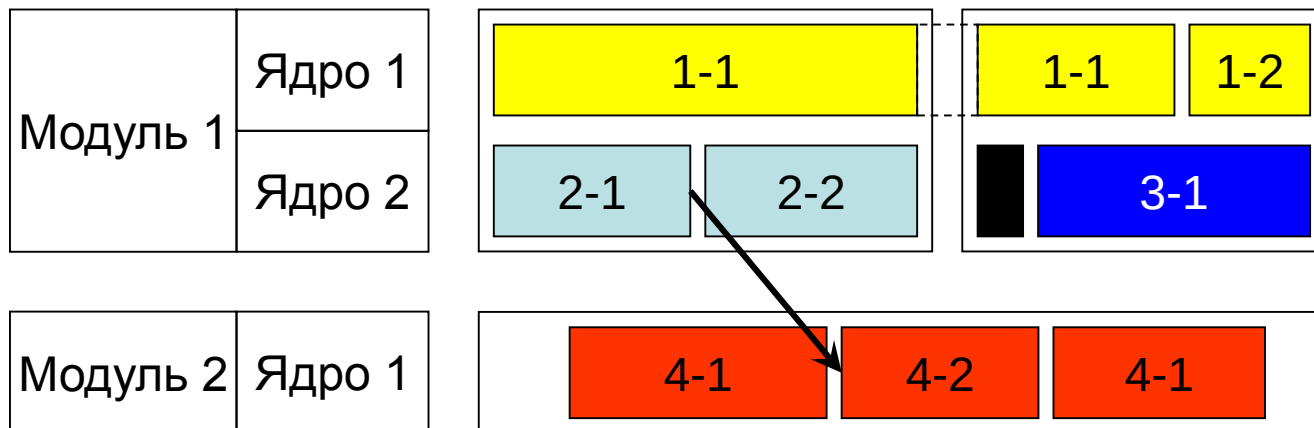
- Ожидание данных только от задач с той же частотой
 - «синхронная» зависимость по данным
- Отправитель выполняется чаще получателя:
 - на входе получателя почти всегда актуальные данные
 - ожидать нет смысла
- Отправитель выполняется реже получателя
 - ожидание нарушает требования по частоте для задачи-получателя

Состояния задач в ARINC653

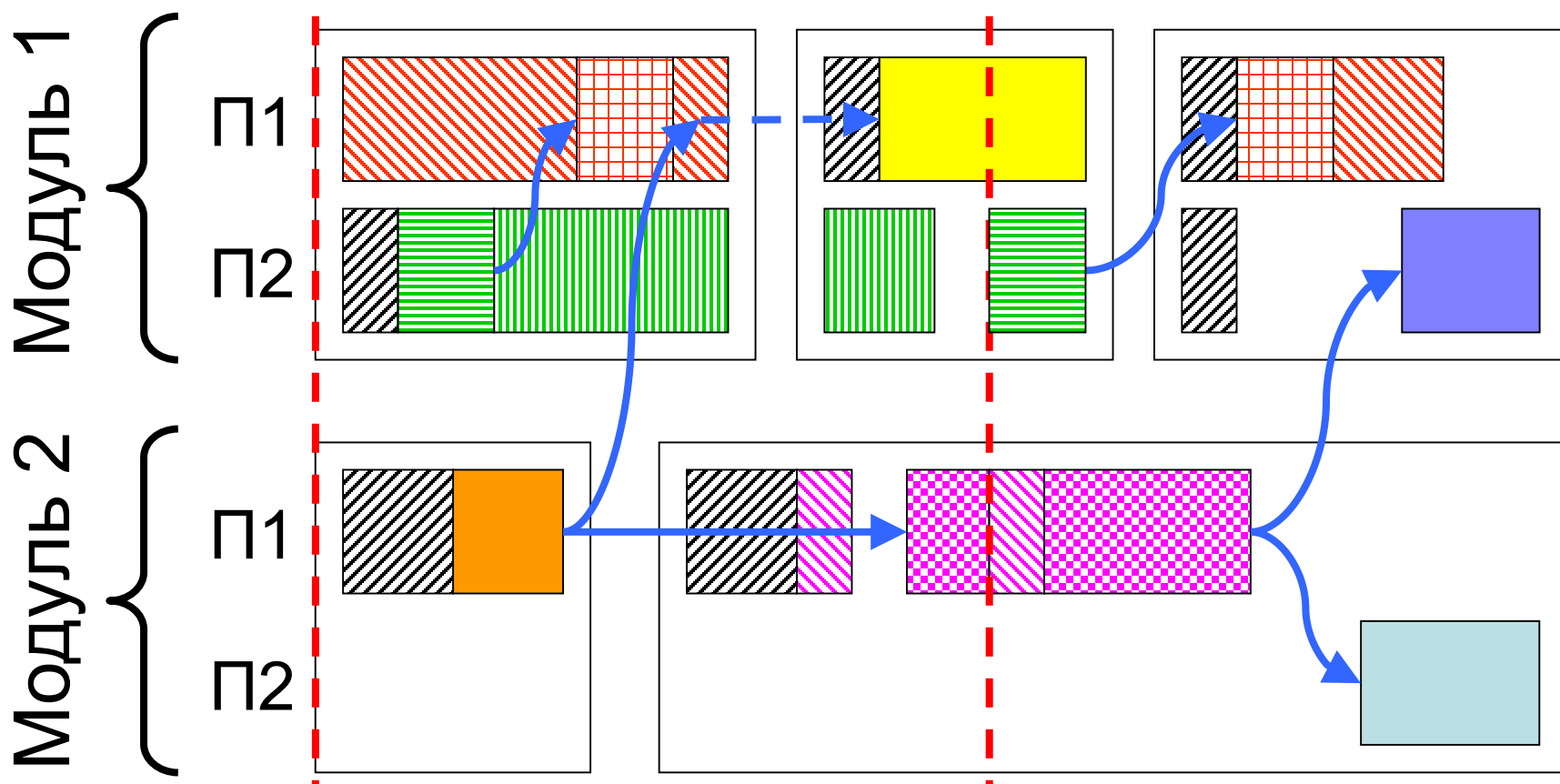


Выполнение задач в системе

- Задачи раздела выполняются в рамках окон
 - статическое расписание окон
 - границы окон одинаковы для всех ядер одного модуля
- Между разделами происходит переключение контекста
- Работы в окне: динамическое планирование
 - очередь выполнения
 - приоритеты
 - вытеснение
 - ожидание входных данных
- Незавершенная работа может быть возобновлена в следующем окне



Пример расписания



Задача планирования:

ВХОДНЫЕ ДАННЫЕ

- Описание системы ИМА
 - набор модулей
 - модуль → набор и типы процессорных ядер
 - ядро → верхняя граница загрузки
- Описание рабочей нагрузки
 - наборы задач, сообщений, разделов
 - задача → период, приоритет, WCET (для типа ядра)
 - раздел → задачи, допустимые ядра
 - сообщение → отправитель, получатель, размер, длительность передачи (через память, через сеть)
 - *свободные задачи*, допустимые ядра

Задача планирования: цели

- Составить разделы из свободных задач
 - Трафик между разделами $\rightarrow \min$
 - Загрузка ядра разделом $\leq U_{\max}$
(при выполнении раздела целиком на одном ядре)
- Привязать разделы к ядрам
 - Трафик между модулями $\rightarrow \min$
(минимизация загрузки сети)
 - Загрузка ядра $\leq U_{\max}$ (ядро)
 - Привязка к допустимым ядрам
 - Выполнение условий динамической планируемости
 - Инкрементальная привязка (расширение прежней)
- Построить расписание окон для каждого ядра
 - Корректность расписание проверяется моделированием работы динамического планировщика
 - Расписание считается корректным, если все задачи выполняются в пределах директивных сроков (Д.С. = период) при длительностях выполнения, равных WCET

$$\sum_{i=0}^n T_i F_i \leq n \left(2^{\frac{1}{n}} - 1 \right)$$

Ограничения корректности

- Раздел привязан ровно к одному ядру
- Не более одного раздела в окне
- Окна не пересекаются по времени
- Работы выполняются в рамках окон раздела
- В каждый момент времени выполняется не более одной работы
- Все работы выполняются полностью
- Выполняются зависимости по данным
- Соблюдаются приоритеты

Жадный алгоритм привязки разделов к процессорным ядрам

1. Выбрать непривязанный раздел P с максимальным трафиком между P и привязанными разделами.
2. Для каждого модуля рассчитать трафик между P и разделами, привязанными к ядрам других модулей.
3. Упорядочить модули по убыванию этого трафика.
4. В цикле по модулям:
 - если раздел P может быть привязан к какому-либо ядру данного модуля без нарушения ограничений на загрузку ядра, то привязать P к этому ядру и выйти из цикла;
 - иначе, если разделы на данном модуле могут быть перераспределены между ядрами этого модуля так, чтобы в достаточной мере «разгрузить» одно из ядер, то выполнить перераспределение, привязать P к этому ядру и выйти из цикла;
5. Если на шаге 4 не выбрано никакое ядро, то **стоп** (неуспех).
6. Если остались непривязанные разделы, то перейти к шагу 1, иначе **стоп** (успех).

Альтернативные алгоритмы привязки разделов к процессорным ядрам

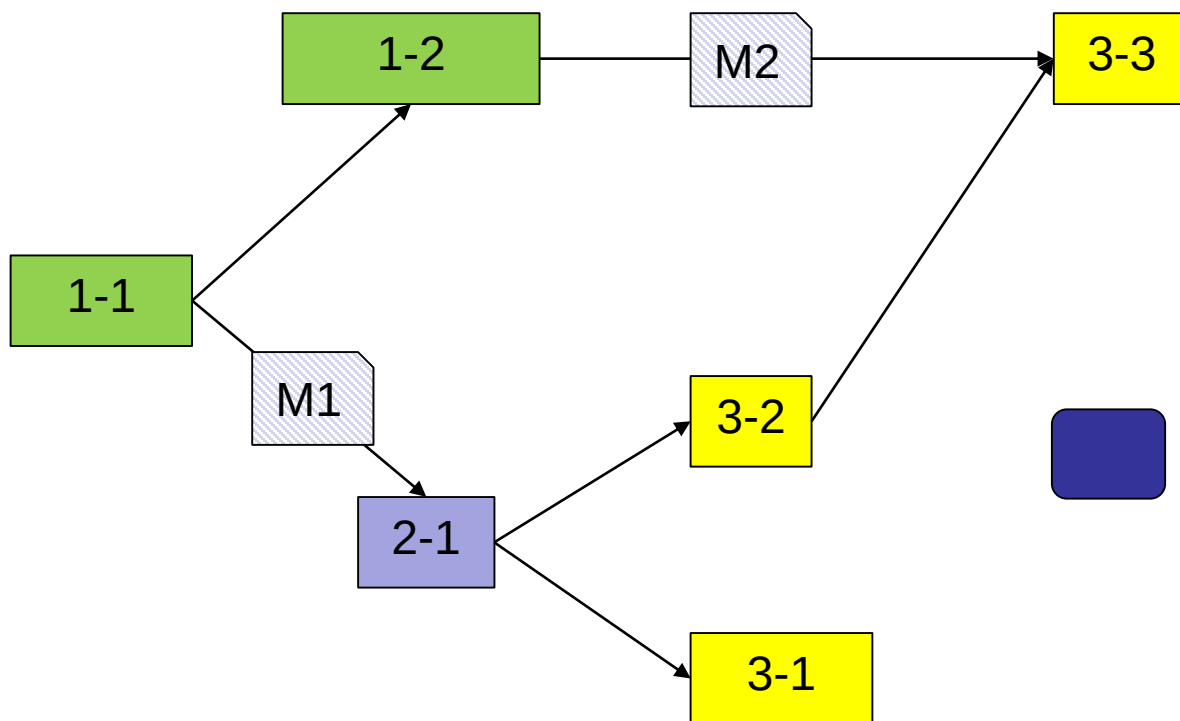
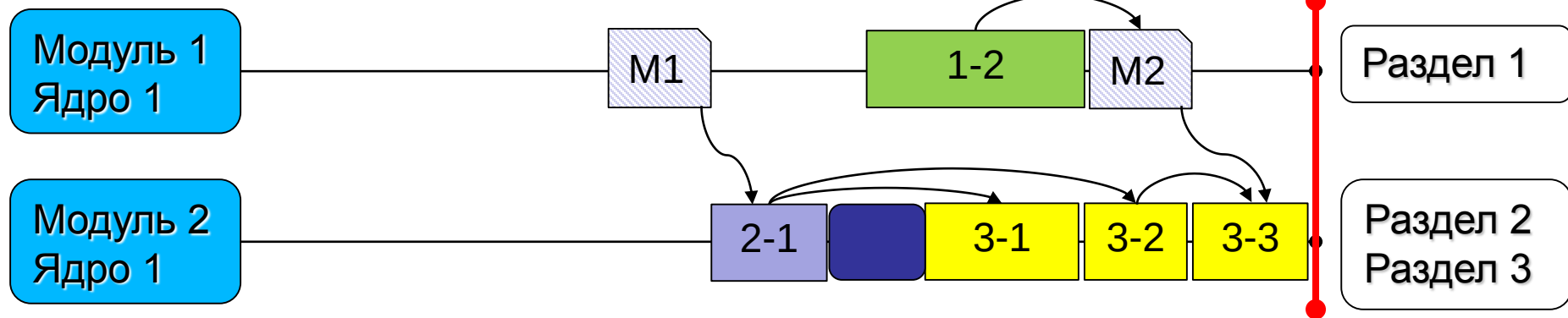
- Метод ветвей и границ
 - Критерий отсечения: загрузка ядра превышает допустимую, или ранее найдено лучшее решение
 - Недостаток: время выполнения на реальных данных
- Упаковка в контейнеры
 - Ядро = контейнер
 - емкость = $U_{\max}(\text{ядро})$
 - Раздел = объект
 - объем = вклад раздела в загрузку ядра
 - стоимость = трафик, становящийся «внутренним» для модуля в случае привязки раздела к ядру этого модуля
 - Проблемы:
 - объем объекта зависит от выбора контейнера (длительность выполнения задачи зависит от типа ядра)
 - стоимость объекта зависит от расположения других объектов

Построение расписания: схема алгоритма

1. Построение графа зависимостей работ
2. Уточнение директивных интервалов работ на основе зависимостей
3. Построение последовательностей выполнения работ «слева направо», параллельно для всех ядер
4. Построение расписания окон на основе построенной последовательности работ
5. Назначение приоритетов задач на основе построенной последовательности работ

Уточнение директивного срока

Дедлайн задачи 1-1



Переключение
контекста и
инициализация окна

Планирование «слева направо»

1. Работа помещается в список готовых для выполнения, если:
 - Начался ее директивный интервал
 - Получены все синхронные входящие сообщения для данной работы
2. Очередная работа для размещения выбирается из списка:
 - Если отсутствуют готовые работы с большим приоритетом
 - По критерию EDF (если приоритет еще не определен)

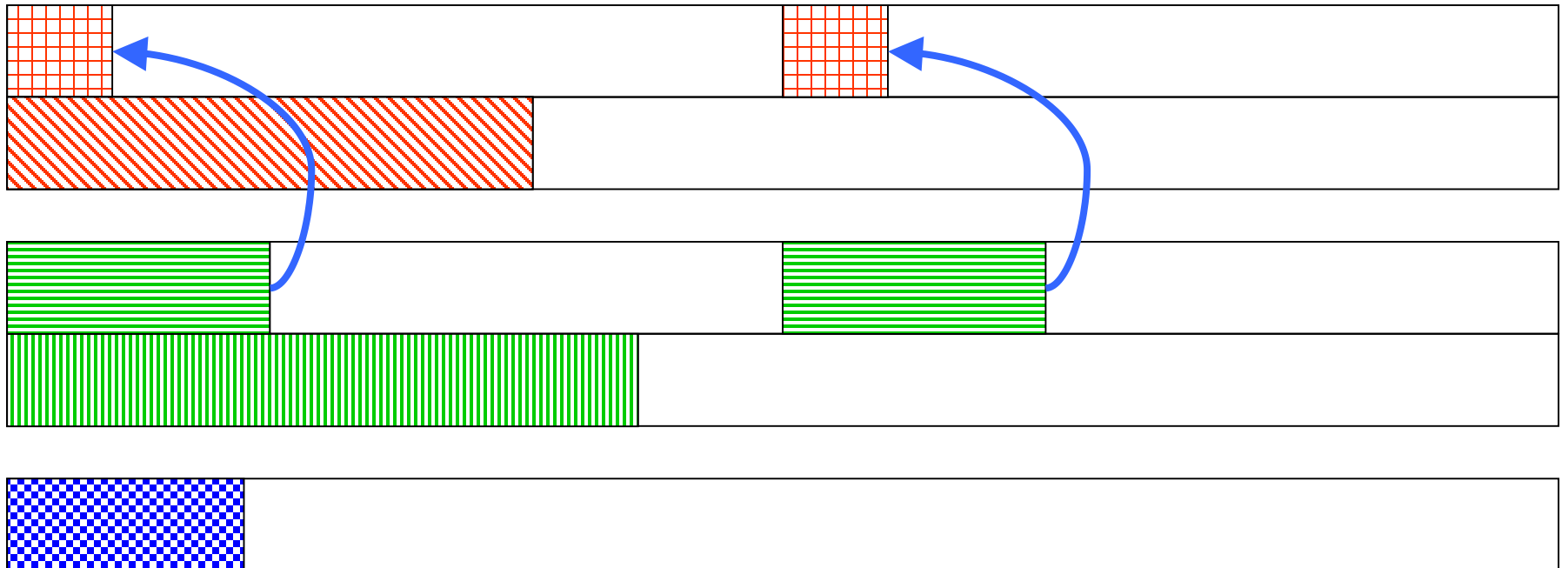
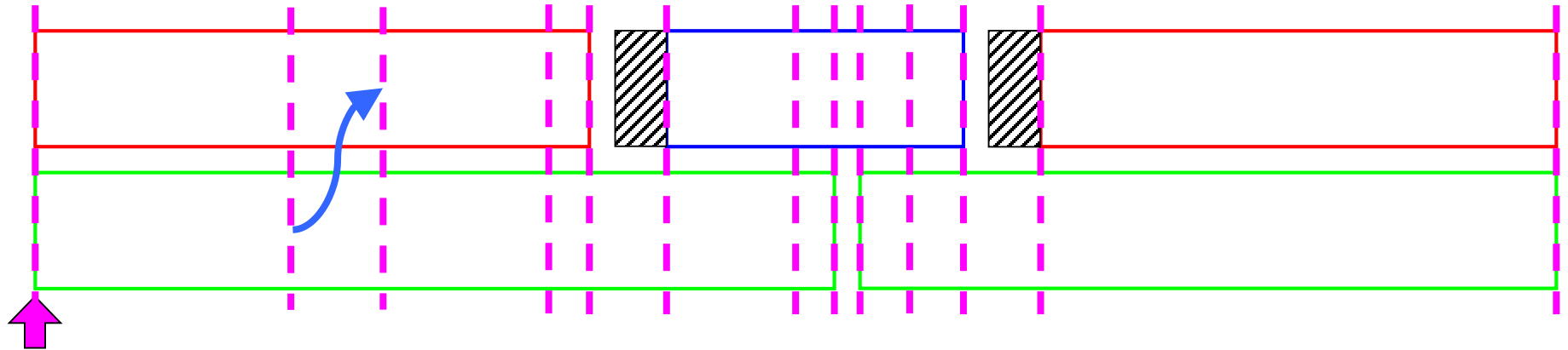
Модель вычислительного процесса

- Служит для проверки корректности расписания окон
- Модель основана на событиях
- Схема работы:
 1. Создание начальных событий
 2. Выбор событий с минимальным временем
 3. Обновление списка готовых к выполнению работ
 4. Обработка выбранных событий
 5. Если список событий не пуст, переход к п.2

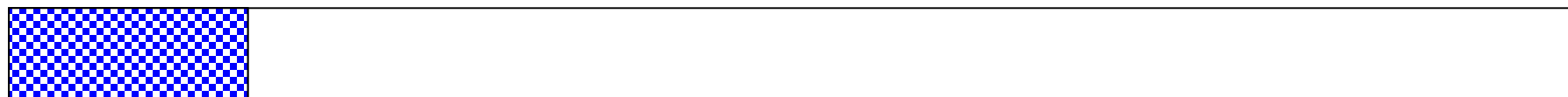
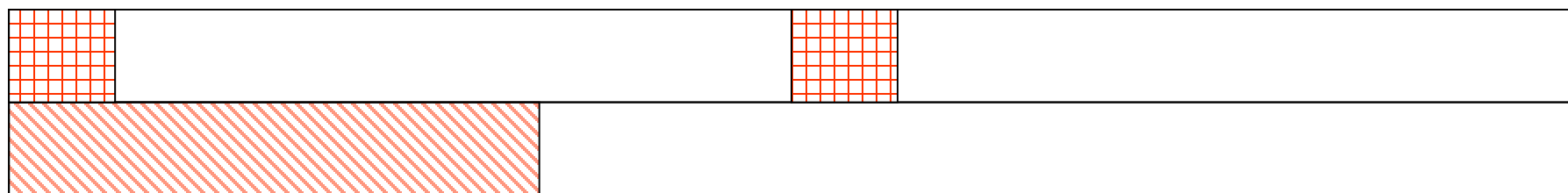
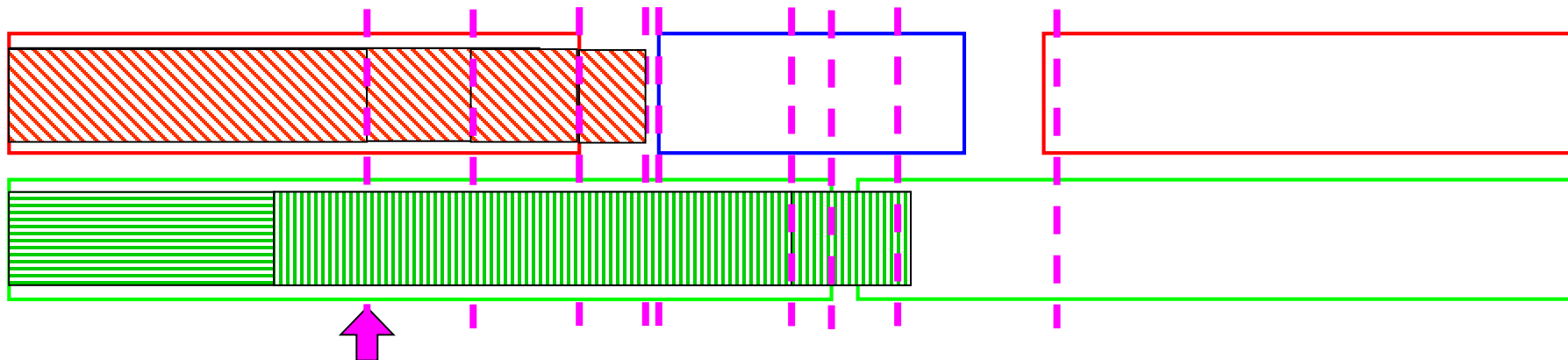
Типы событий

- Открытие окна
- Закрытие окна
- Начало директивного интервала
- Завершение работы
- Доставка сообщения

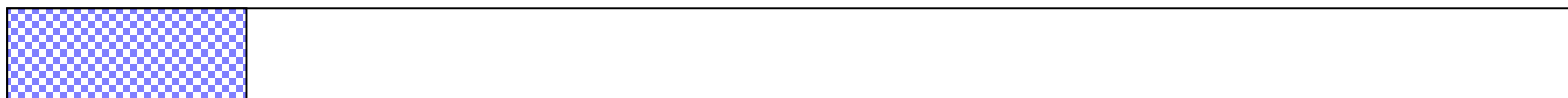
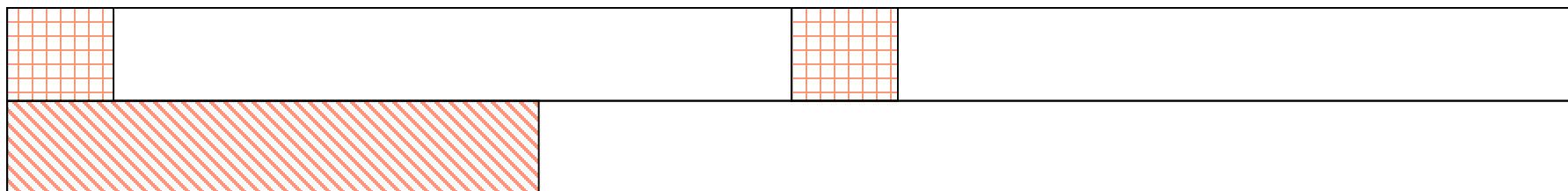
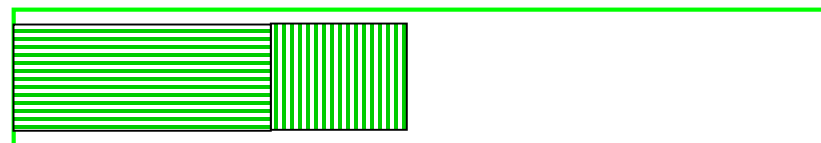
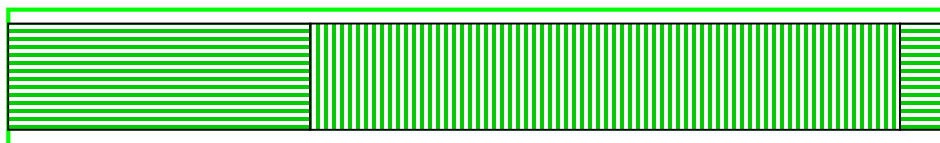
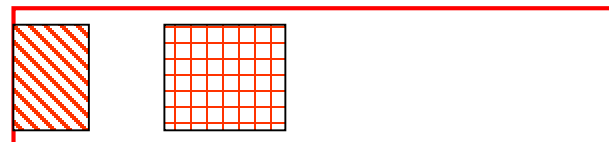
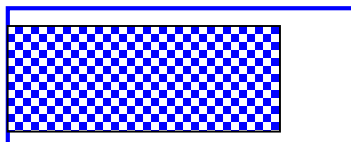
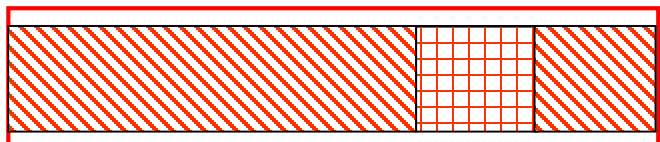
Работа модели



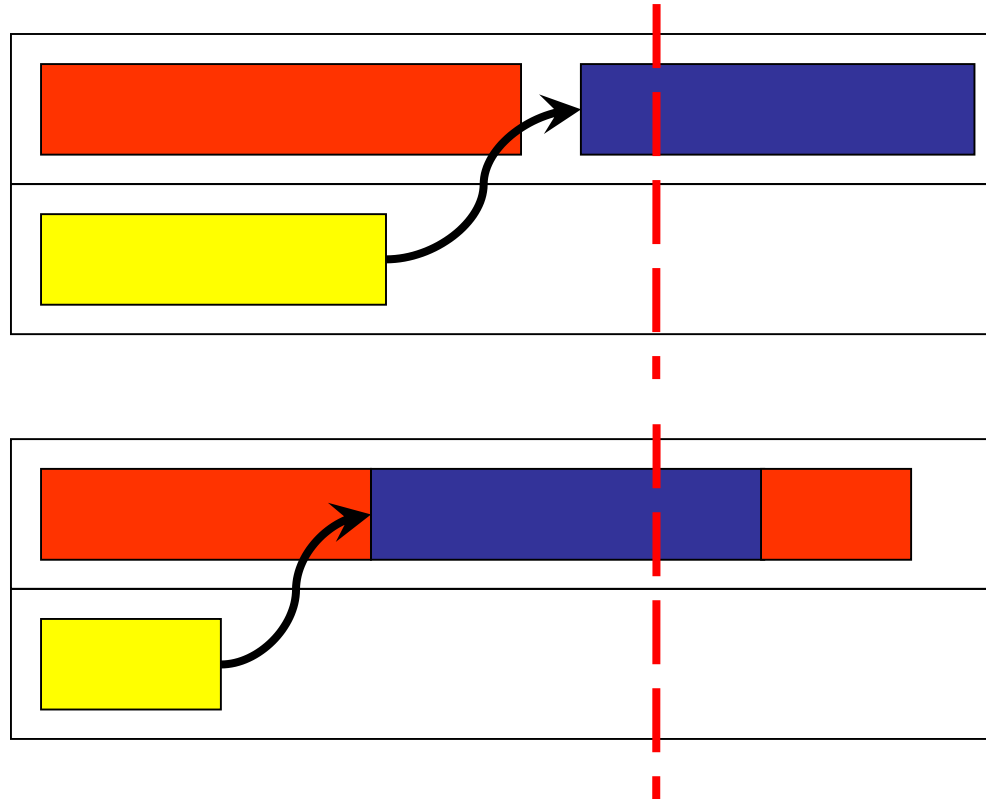
Работа модели



Работа модели



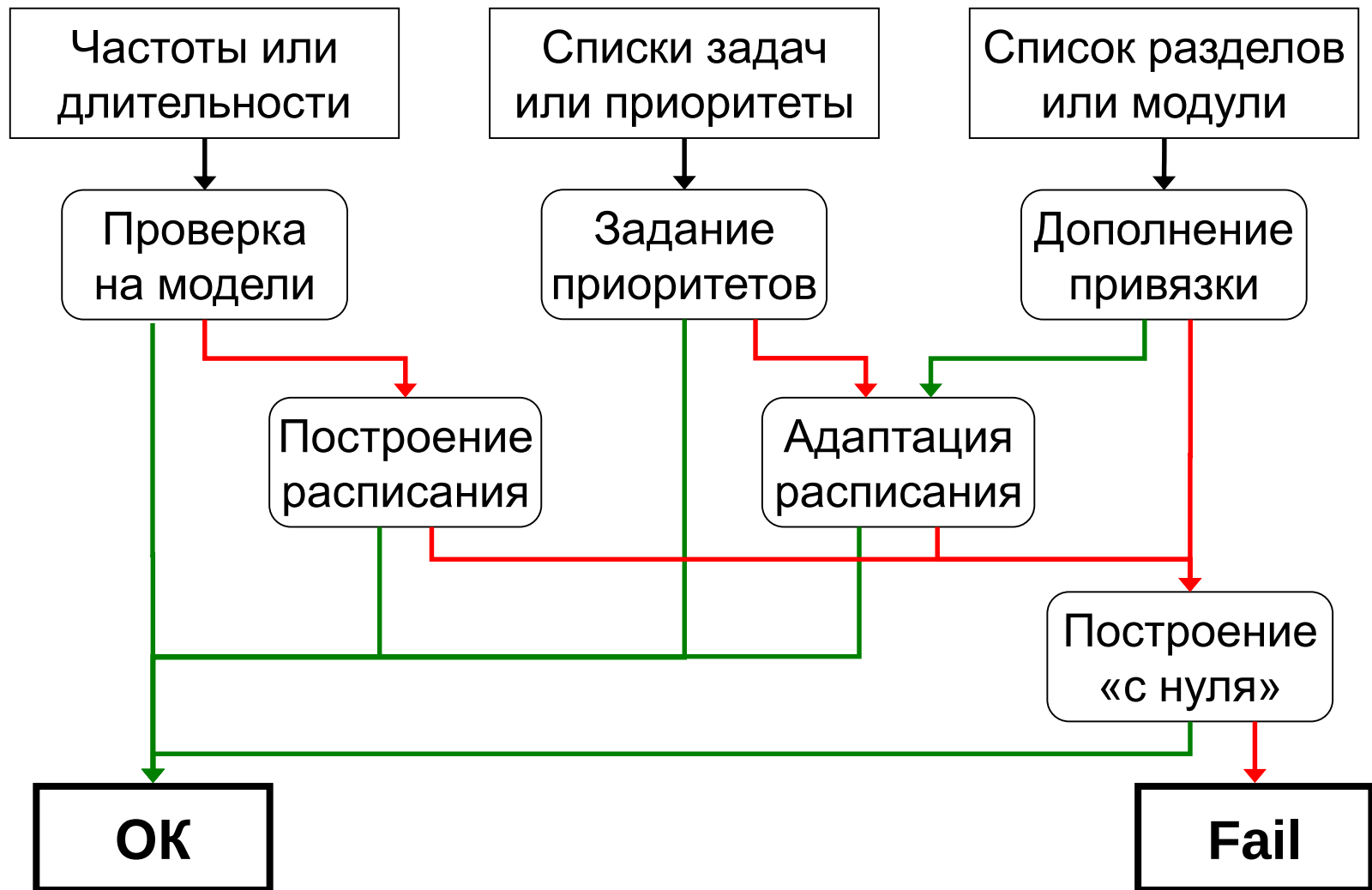
Недостатки модели



Решение:

не выполнять низкоприоритетные задачи, пока есть высокоприоритетные, ожидающие сообщений

Адаптация к изменениям



Перспективные задачи

- Построение расписания с минимизацией затрат на переключение контекста
- Корректировка распределения разделов по ядрам при неуспешном построении расписания
- Построение расписания, гарантированно корректного при временах выполнения задач, меньших чем WCET
- Минимизация сетевой загрузки с учётом синхронности зависимостей по данным
 - сообщения, которые ожидает задача-получатель, следует передавать через память (=> привязка отправителя и получателя к одному модулю)
- Совместное планирование задач и конфигурирование сети

Спасибо за внимание

ИНФОРМАЦИОННО-УПРАВЛЯЮЩИЕ СИСТЕМЫ РЕАЛЬНОГО ВРЕМЕНИ

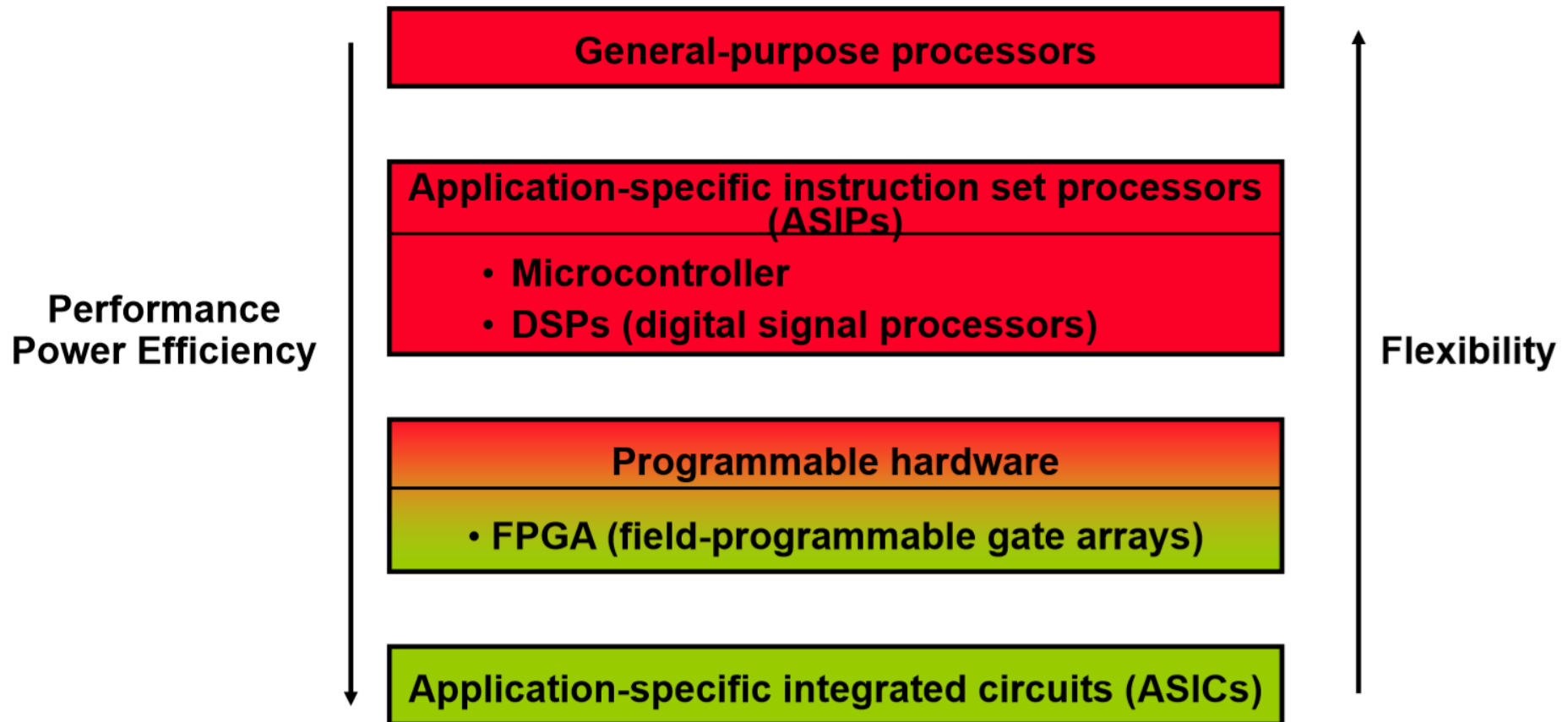
Лекция 5: *Процессоры ИУС РВ*

Кафедра АСВК,
Лаборатория Вычислительных Комплексов
Балашов В.В.

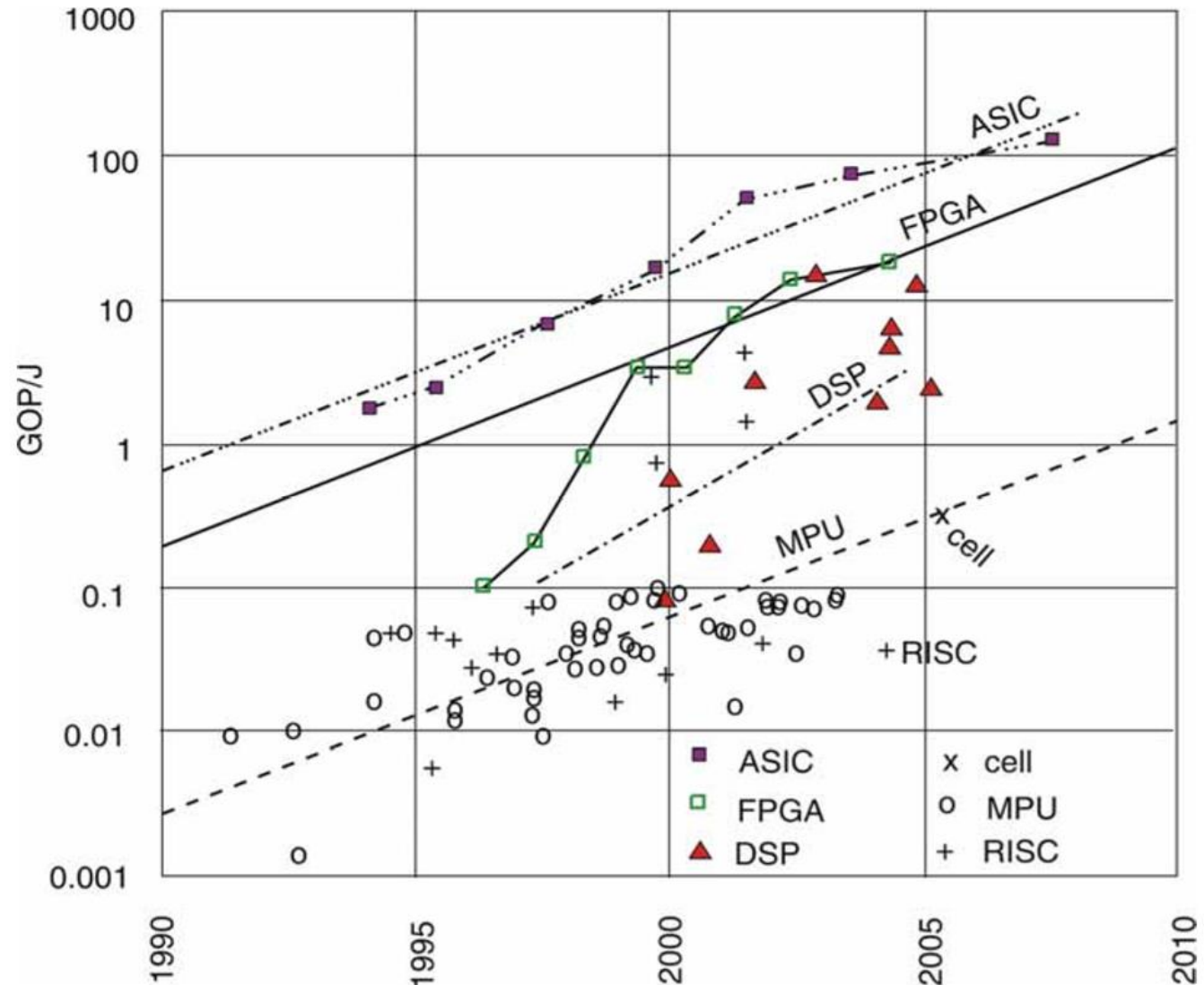
Ограничения на процессоры ИУС РВ

- Технологические ограничения:
 - вынужденное применение «грубого» технологического процесса
 - жёсткие ограничения по энергопотреблению и тепловыделению
- Откуда берутся ограничения
 - требование к устойчивости к внешним воздействиям (излучение и т.п.)
 - неразвитость технологических процессов производства микросхем
 - общий лимит на энергопотребление ИУС РВ
 - ограниченные возможности теплоотвода

Implementation Alternatives



Energy Efficiency



General purpose processors
+
specialization

General-purpose Processors

► *High performance*

- Highly optimized circuits and technology
- Use of parallelism
 - superscalar: dynamic scheduling of instructions
 - super-pipelining: instruction pipelining, branch prediction, speculation
- complex memory hierarchy

► *Not suited for real-time applications*

- Execution times are highly unpredictable because of intensive resource sharing and dynamic decisions

► *Properties*

- Good average performance for large application mix
- High power consumption

Basic Pentium® III Processor Misprediction Pipeline

1	2	3	4	5	6	7	8	9	10
Fetch	Fetch	Decode	Decode	Decode	Rename	ROB Rd	Rdy/Sch	Dispatch	Exec

Basic Pentium® 4 Processor Misprediction Pipeline

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
TC Nxt IP	TC Fetch	Drive	Alloc	Rename	Que	Sch	Sch	Sch	Disp	Disp	RF	RF	Ex	Flgs	Br Ck	Drive			

Intel Pentium 4 Northwood

Buffer Allocation & Register Rename

Execution Pipeline Start

Instruction Trace Cache

Trace Cache Access, next Address Predict

Instruction Queue (for less critical fields of the uOps)

General Instruction Address Queue & Memory Instruction Address Queue (queues register entries and latency fields of the uOps for scheduling)

Floating Point, MMX, SSE2 Renamed Register File 128 entries of 128 bit.

uOp Schedulers

FP Move Scheduler: (8x8 dependency matrix)

Parallel (Matrix) Scheduler for the two double pumped ALU's

General Floating Point and Slow Integer Scheduler: (8x8 dependency matrix)

Load / Store uOp Scheduler: (8x8 dependency matrix)

Load / Store Linear Address Collision History Table

Integer Execution Core

- (1) uOp Dispatch unit & Replay Buffer Dispatches up to 6 uOps / cycle
- (2) Integer Renamed Register File 128 entries of 32 bit + 6 status flags 12 read ports and six write ports
- (3) Databus switch & Bypasses to and from the Integer Register File.
- (4) Flags, Write Back
- (5) Double Pumped ALU 0
- (6) Double Pumped ALU 1
- (7) Load Address Generator Unit
- (8) Store Address Generator Unit
- (9) Load Buffer (48 entries)
- (10) Store Buffer (24 entries)

Register Alias History Tables (2x126)
Register Alias Tables uOp Queue

Micro code Sequencer
Micro code ROM & Flash

Trace Cache Fill Buffers
Distributed Tag comparators 24 bit virtual Tags

Trace Cache Branch Prediction Table (BTB), 512 entries.

Return Stacks (2x16 entries)

Trace Cache next IP's (2x)

Miscellaneous Tag Data

Instruction Decoder

Up to 4 decoded uOps/cycle out. (from max. one x86 instr/cycle) Instructions with more than four are handled by Micro Sequencer

Trace Cache LRU bits

Raw Instruction Bytes in Data TLB, 64 entry fully associative, between threads dual ported (for loads and stores)

Instruction Fetch from L2 cache and Branch Prediction

Front End Branch Prediction Tables (BTB), shared, 4096 entries in total

Instruction TLB's 2x64 entry, fully associative for 4k and 4M pages. In: Virtual address [31:12] Out: Physical address [35:12] + 2 page level bits

Front Side Bus Interface, 400..800 MHz

- (11) ROB Reorder Buffer 3x42 entries
- (12) 8 kByte Level 1 Data cache four way set associative, 1R/1W

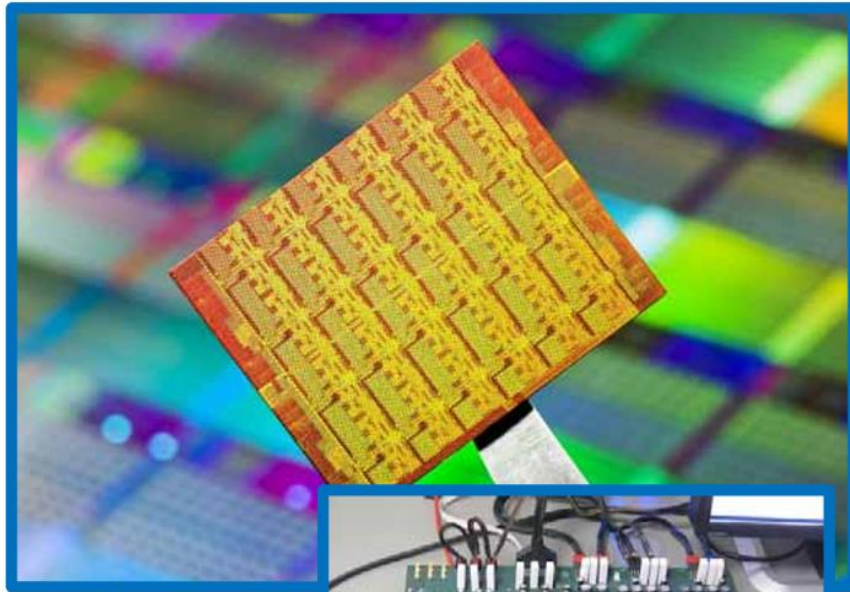
- (13) Summed Address Index decode and Way Predict
- (14) Cache Line Read / Write Transferbuffers and 256 bit wide bus to and from L2 cache

General-purpose Processors

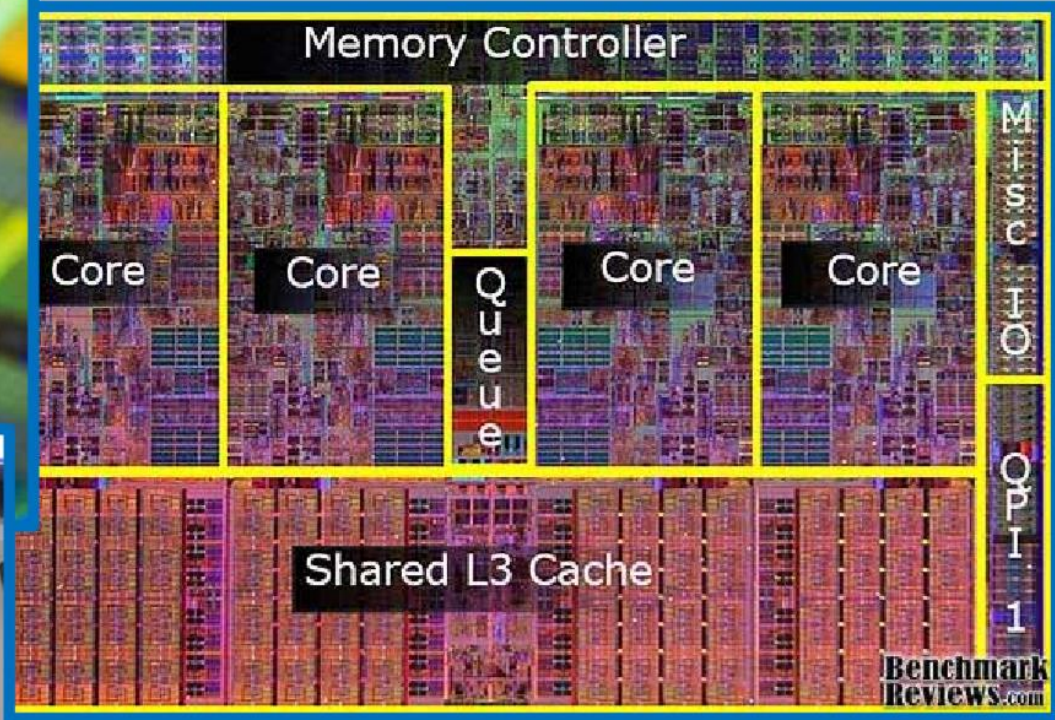
► *Multicore Processors*

- Potential of providing higher execution performance by exploiting parallelism
- Especially useful in high-performance embedded systems, e.g. autonomous driving
- Disadvantages and problems for embedded systems:
 - Increased interference on shared resources such as buses and shared caches
 - Increased timing uncertainty
 - Often, there is limited parallelism in embedded applications

Multicore Examples

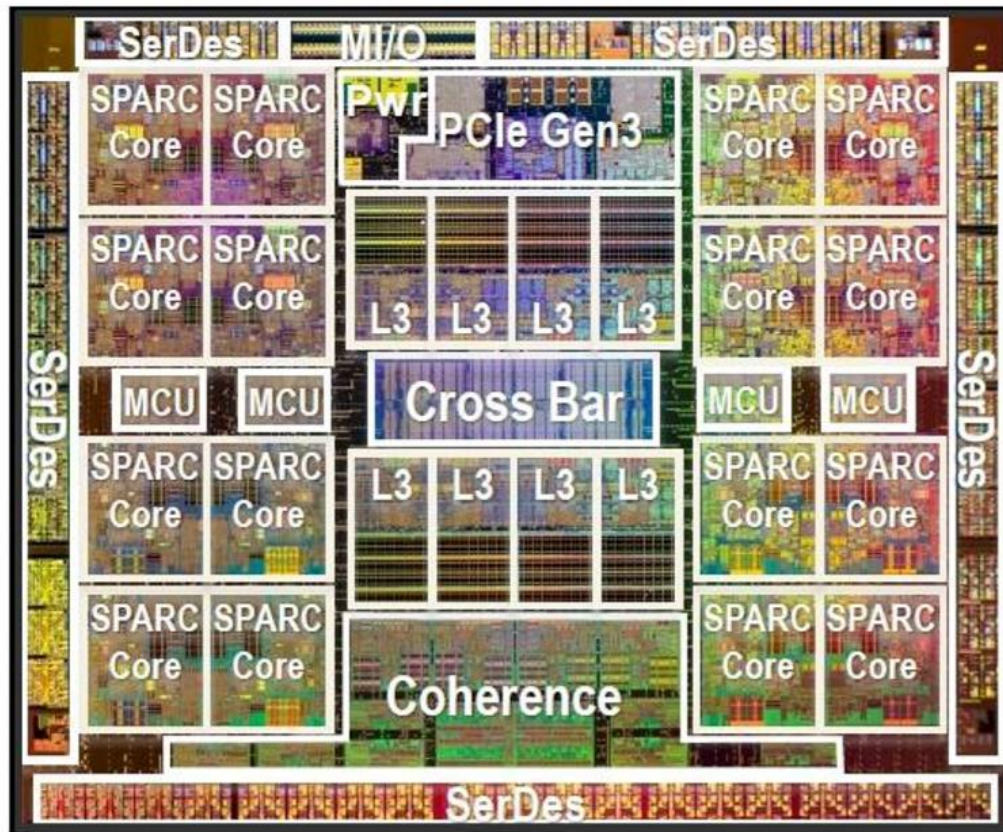


48 cores



4 cores

Multicore Examples



Oracle Sparc T5

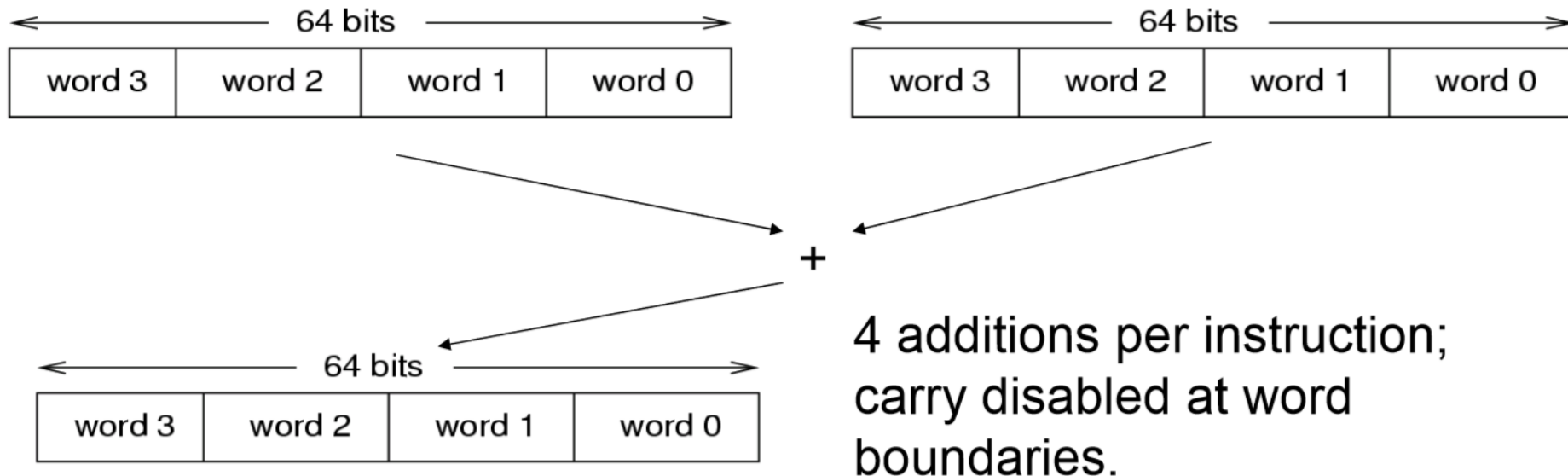
System Specialization

- ▶ The main difference between general purpose highest volume microprocessors and embedded systems is *specialization*.
- ▶ *Specialization should respect flexibility*
 - application domain specific systems shall cover a class of applications
 - some flexibility is required to account for late changes, debugging
- ▶ *System analysis required*
 - identification of application properties which can be used for specialization
 - quantification of individual specialization effects

Example: Multimedia-Instructions

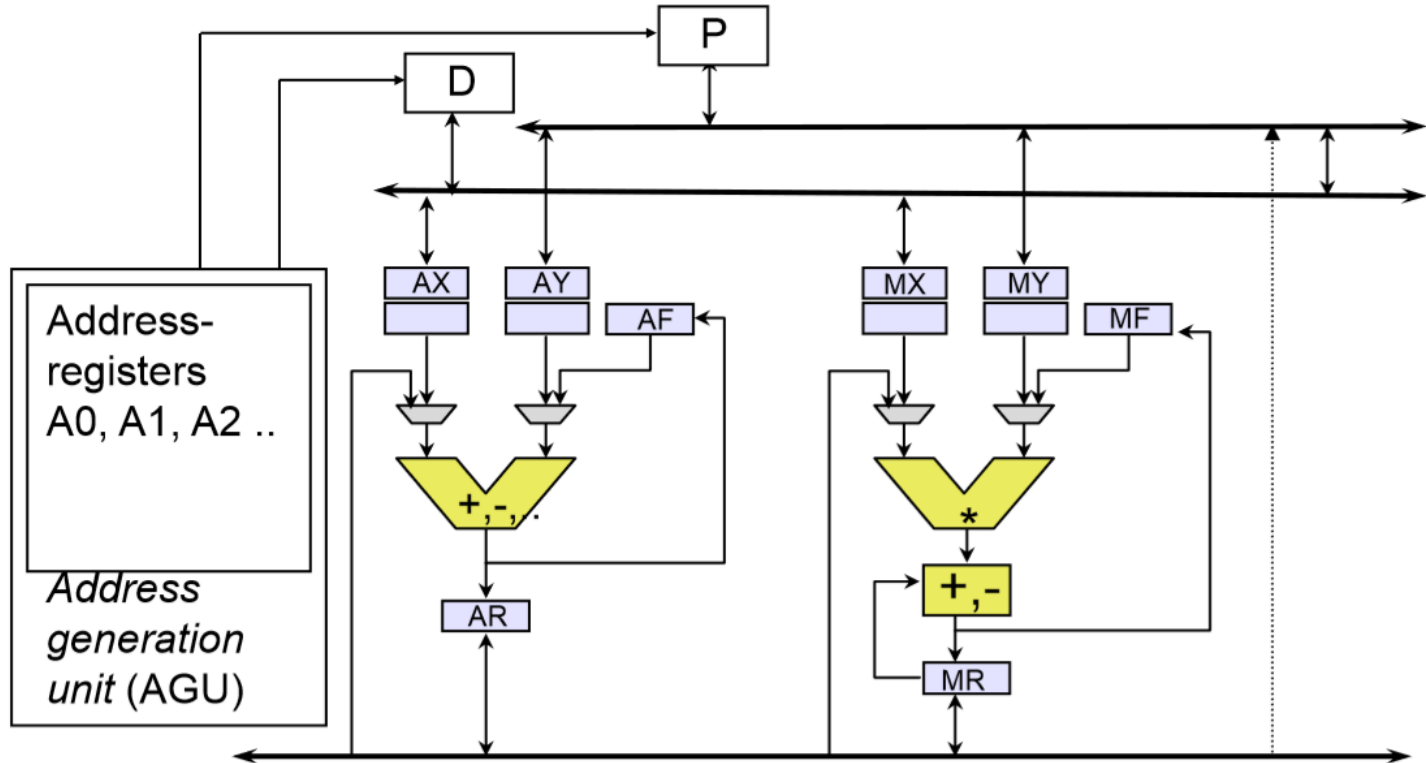
Multimedia instructions exploit that many registers, adders etc are quite wide (32/64 bit), whereas most multimedia data types are narrow (e.g. 8 bit per color, 16 bit per audio sample per channel)

☞ 2-8 values can be stored per register and added.



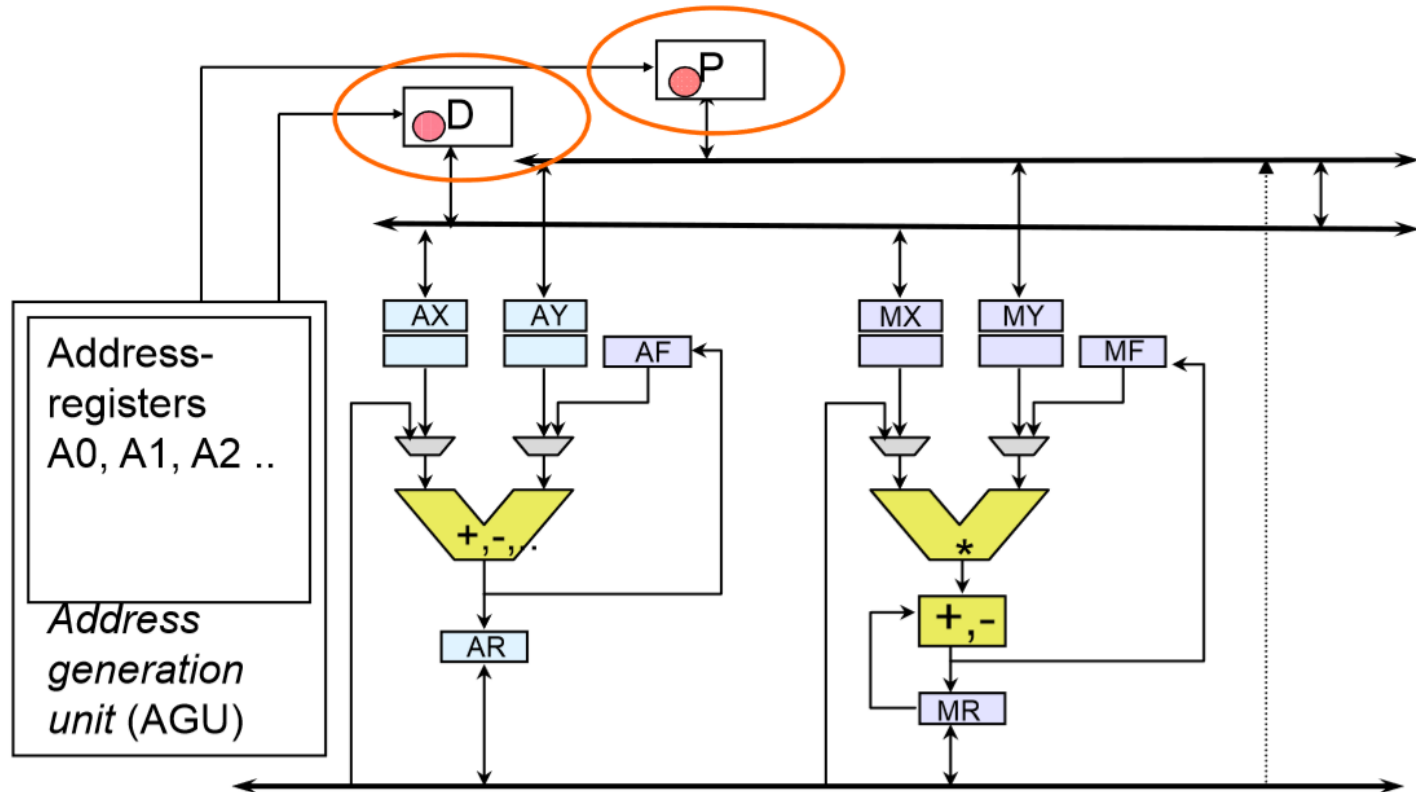
Example: Heterogeneous registers

Example (ADSP 210x):



Different functionality of registers AR, AX, AY, AF, MX, MY, MF, MR

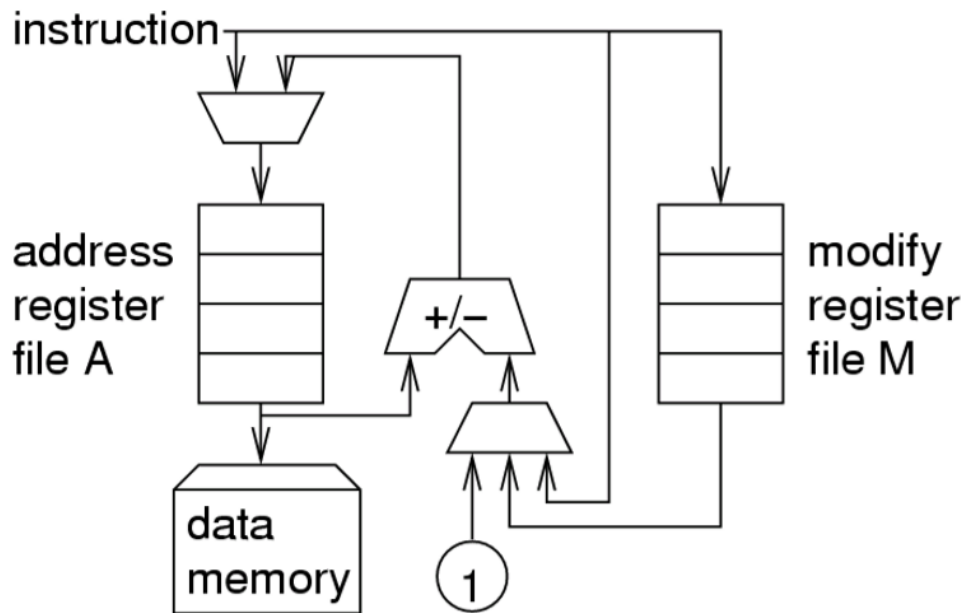
Example: Multiple memory banks or memories



Simplifies parallel fetches

Example: Address generation units

Example (ADSP 210x):



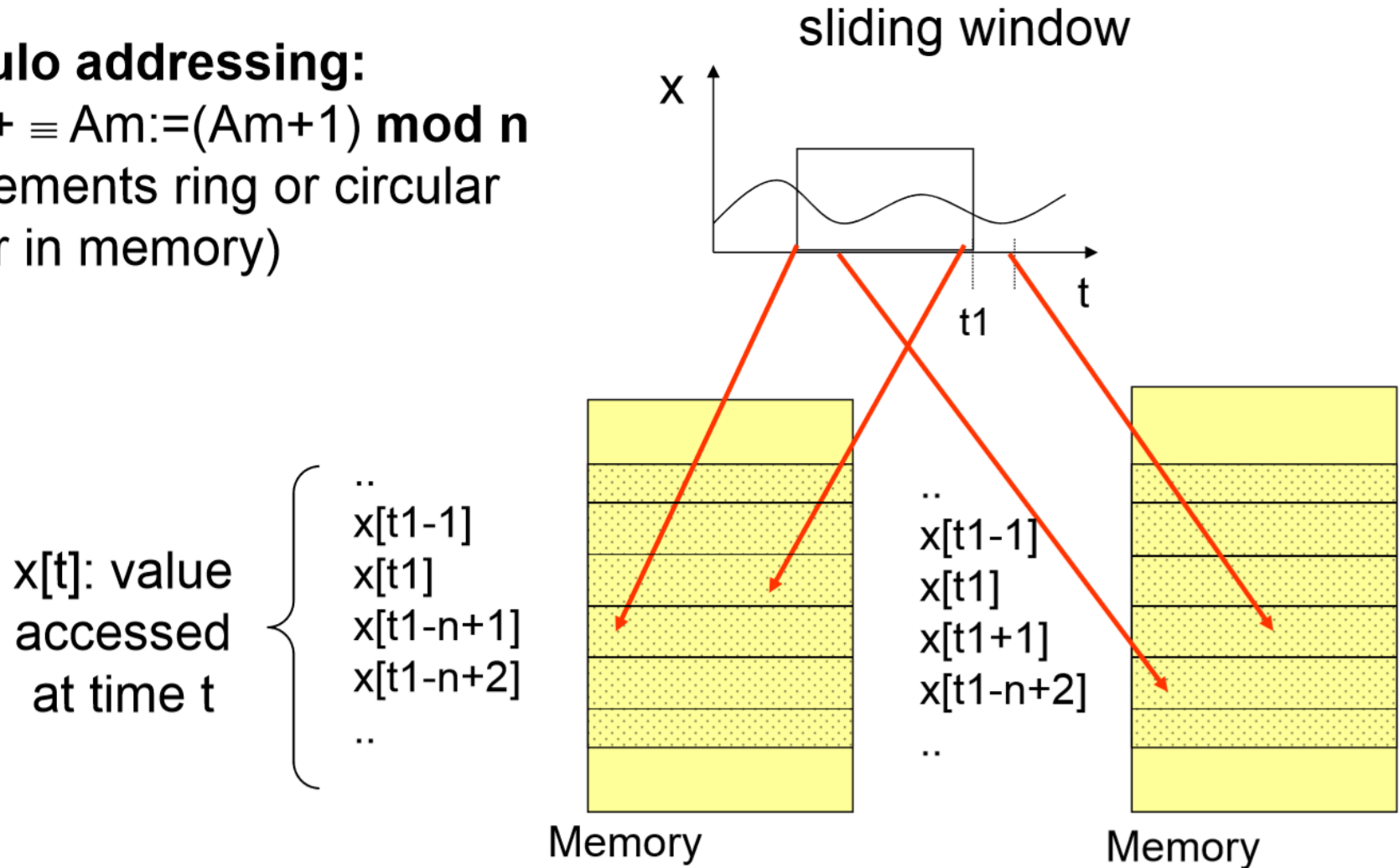
- Data memory can only be fetched with address contained in register file A, but its update can be done in parallel with operation in main data path (takes effectively 0 time).
- Register file A contains several precomputed addresses $A[i]$.
- There is another register file M that contains modification values $M[j]$.
- Possible updates:
 - $M[j] := \text{'immediate'}$
 - $A[i] := A[i] \pm M[j]$
 - $A[i] := A[i] \pm 1$
 - $A[i] := A[i] \pm \text{'immediate'}$
 - $A[i] := \text{'immediate'}$

Example: Modulo addressing

Modulo addressing:

$A_{m++} \equiv A_m := (A_m + 1) \bmod n$

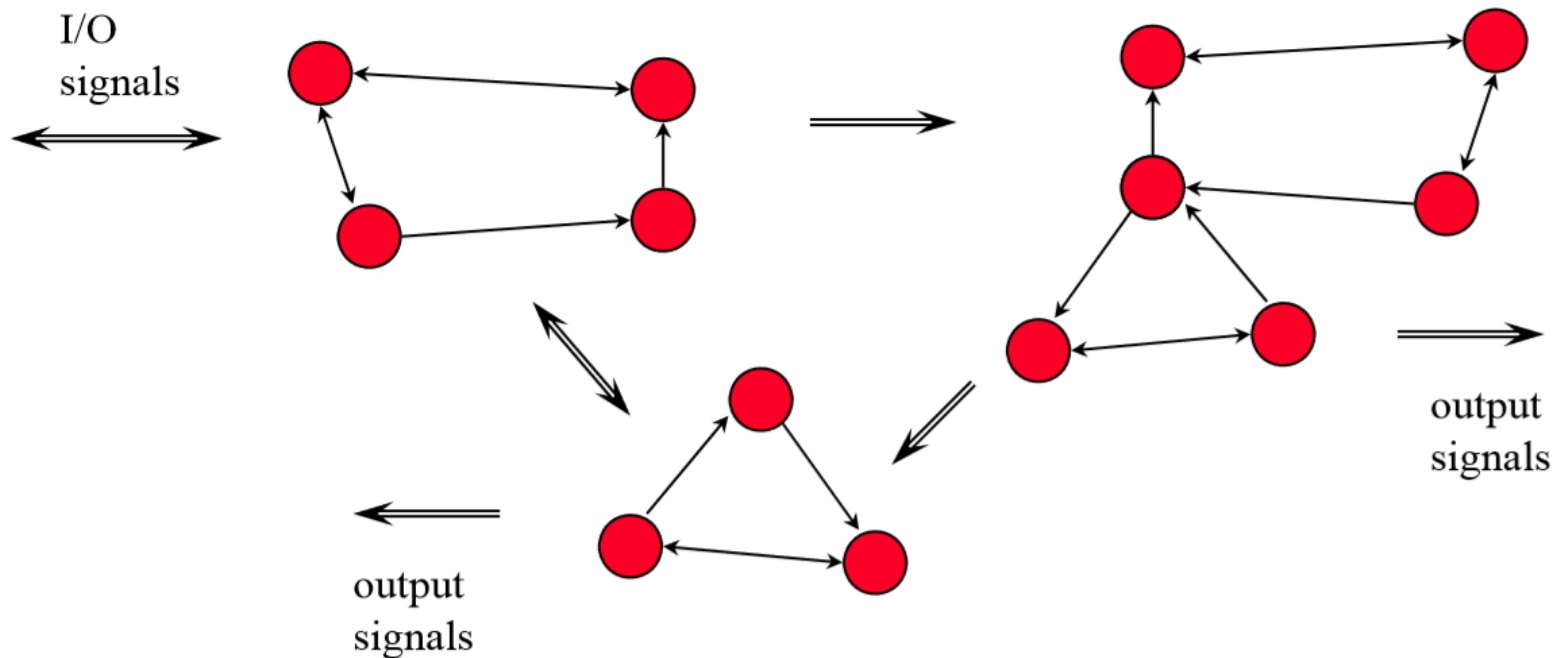
(implements ring or circular buffer in memory)



Application specific processors: *microcontrollers*

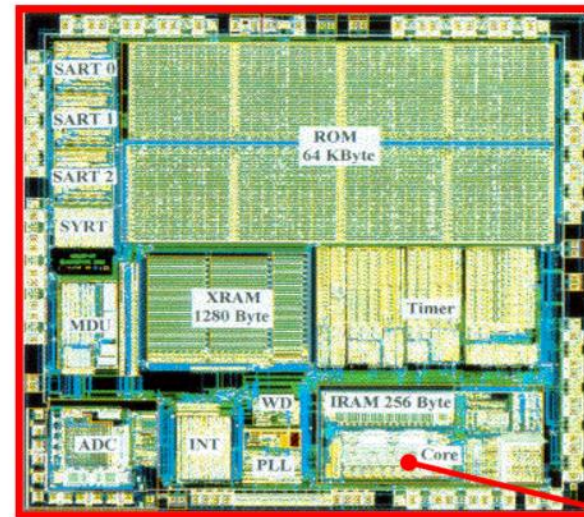
Control Dominated Systems

- ▶ Reactive systems with *event driven behavior*
- ▶ Underlying semantics of system description (“input model of computation”) typically (coupled) Finite State Machines or Petri Nets



Microcontroller

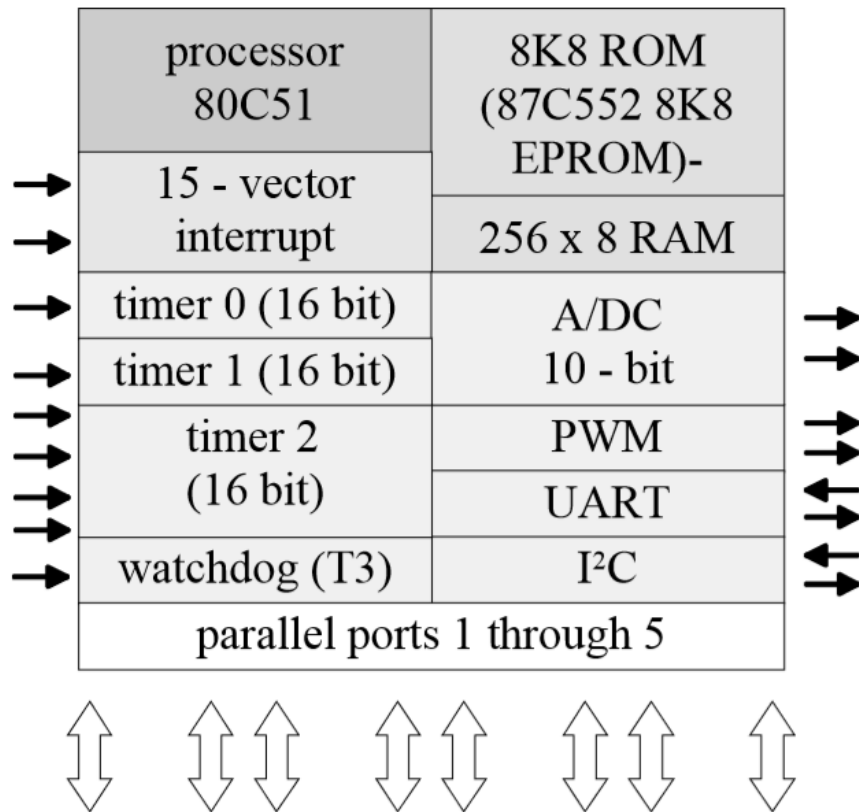
- ▶ control-dominant applications
 - supports process scheduling and synchronization
 - preemption (interrupt), context switch
 - short latency times
- ▶ low power consumption
- ▶ peripheral units often integrated
- ▶ suited for real-time applications



SIECO51 (Siemens)

8051 core

Microcontroller as a System-on-Chip



- complete system
- timers
- I²C-bus and par./ser. interfaces for communication
- A/D converter
- watchdog (SW activity timeout): safety
- on-chip memory
- interrupt controller

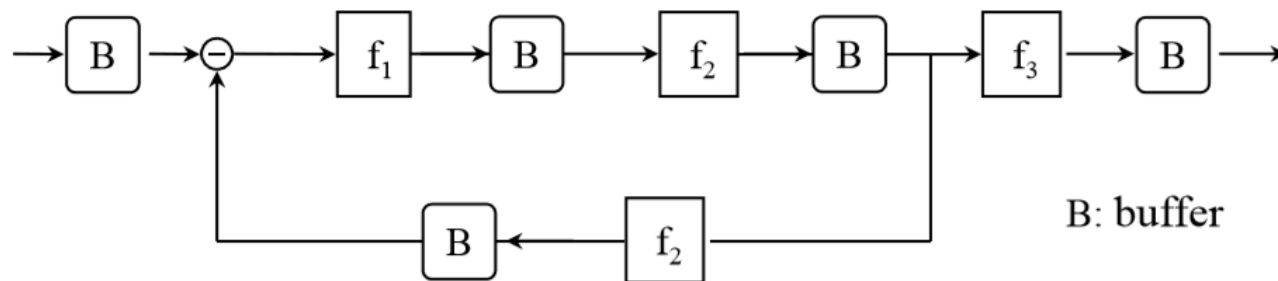
Philips 83 C552: 8 bit-8051 based microcontroller

Application specific processors:

DSP & VLIW

Data Dominated Systems

- ▶ **Streaming oriented systems** with mostly periodic behavior
- ▶ Underlying semantics of input description e.g. **flow graphs** (“input model of computation”)

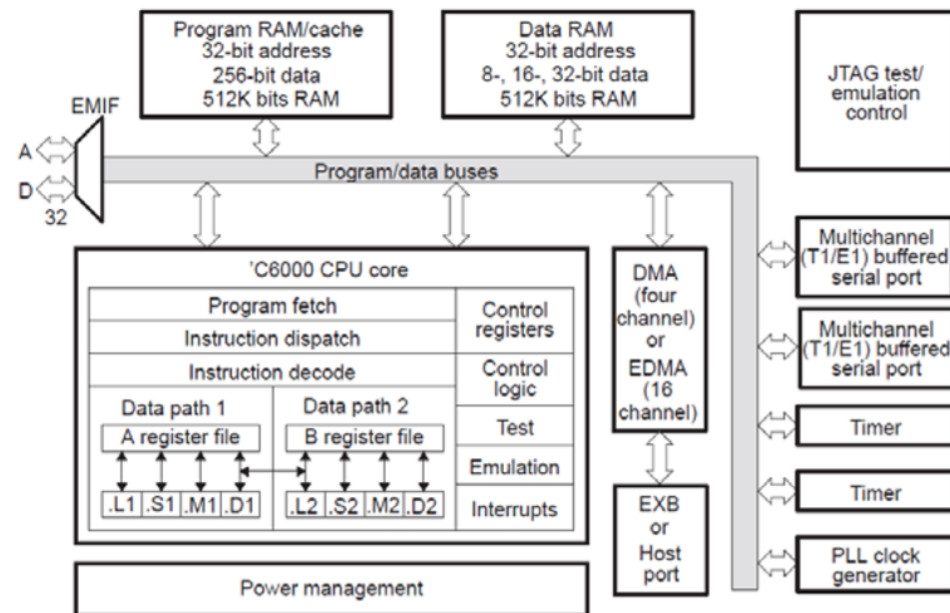


- ▶ **Application examples**: signal processing, control engineering

Digital Signal Processor

- ▶ optimized for data-flow applications
- ▶ suited for simple control flow
- ▶ parallel hardware units (VLIW)
- ▶ specialized instruction set
- ▶ high data throughput
- ▶ zero-overhead loops
- ▶ specialized memory
- ▶ suited for real-time applications

Figure 2-1. TMS320C62x/C67x Block Diagram

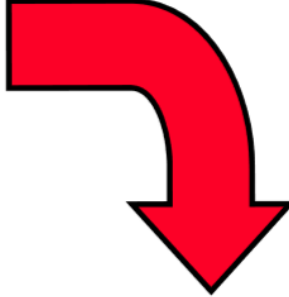


MAC (multiply & accumulate)

```
sum = 0.0;  
for (i=0; i<N; i++)  
    sum = sum + a[i]*b[i];
```

zero-overhead loop
(repeat next instruction N times)

MAC - Instruktion



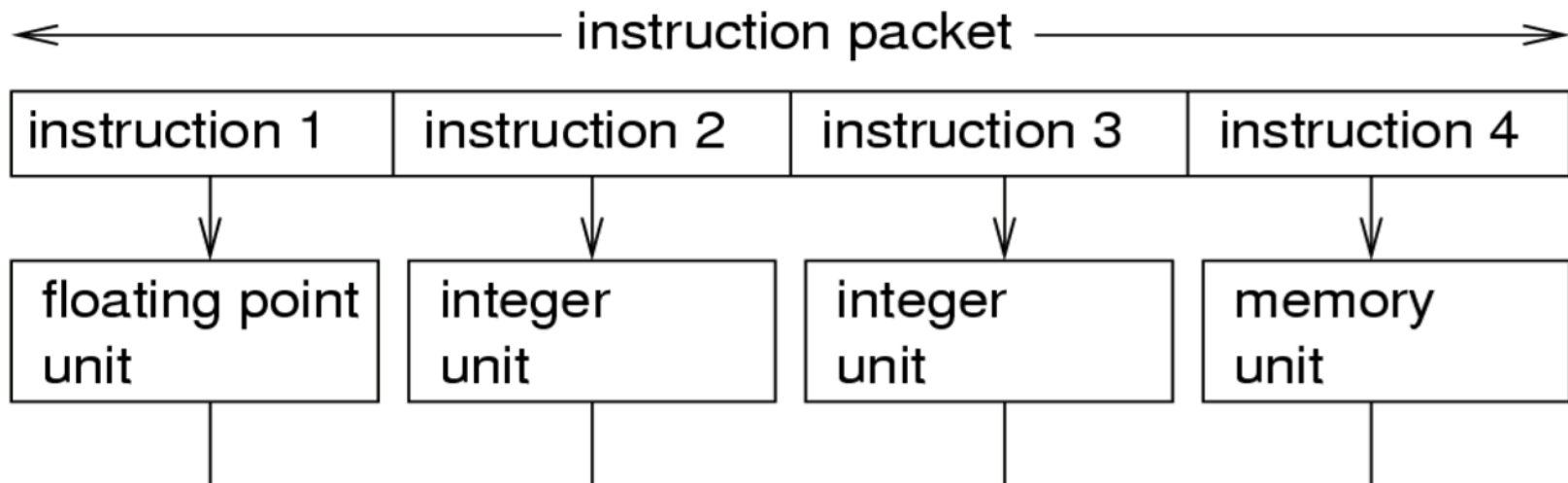
```
LDF    0, R0  
LDF    0, R1  
RPTS   N  
MPYF3  *(AR0)++, *(AR1)++, R0  
|| ADDF3 R0, R1, R1
```

TMS320C3x Assembler
(Texas Instruments)

Very Long Instruction Word (VLIW)

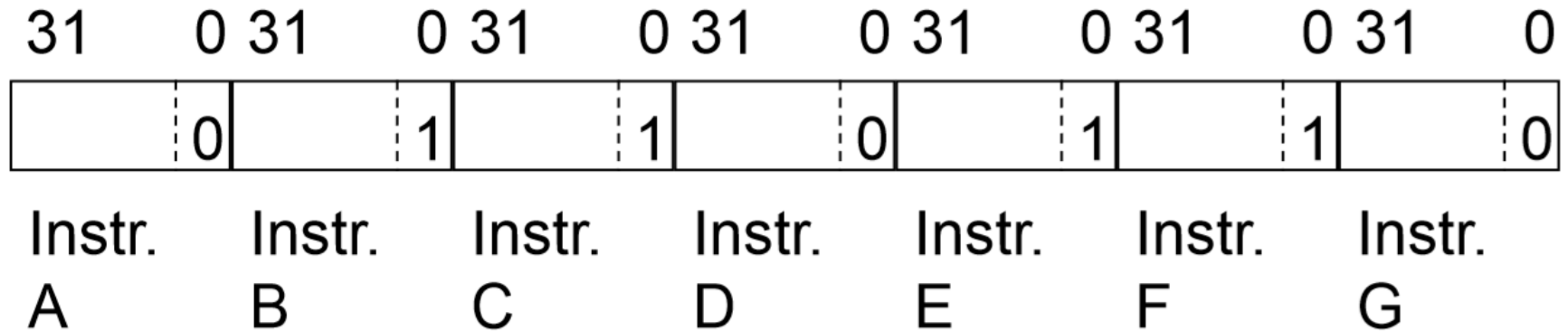
Key idea: detection of possible parallelism to be done by compiler, not by hardware at run-time (inefficient).

VLIW: parallel operations (instructions) encoded in one long word (instruction packet), each instruction controlling one functional unit. E.g.:



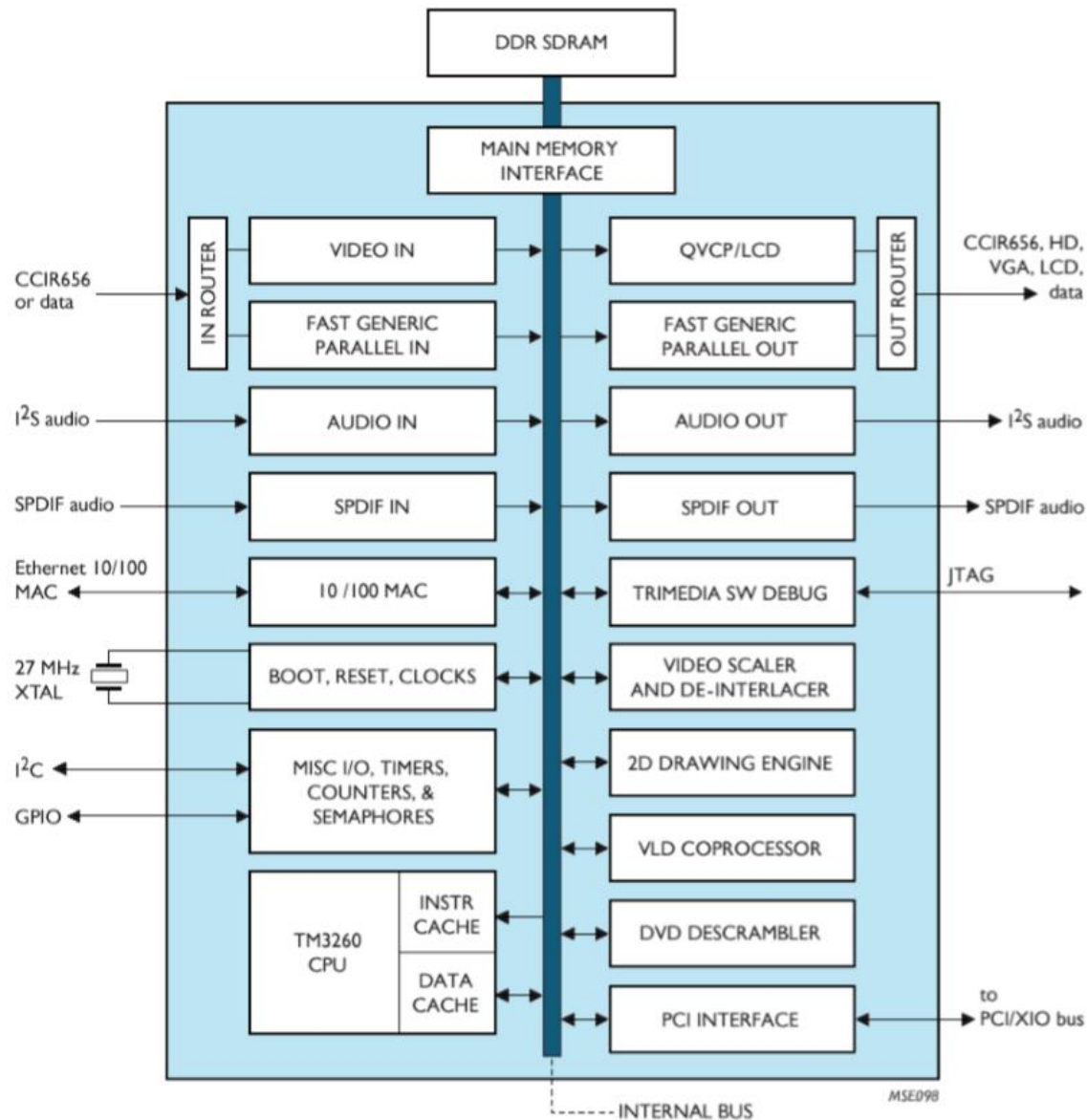
Explicit Parallelism Instruction Computers

The TMS320C62xx VLIW Processor as an example of EPIC:



Cycle	Instruction		
1	A		
2	B	C	D
3	E	F	G

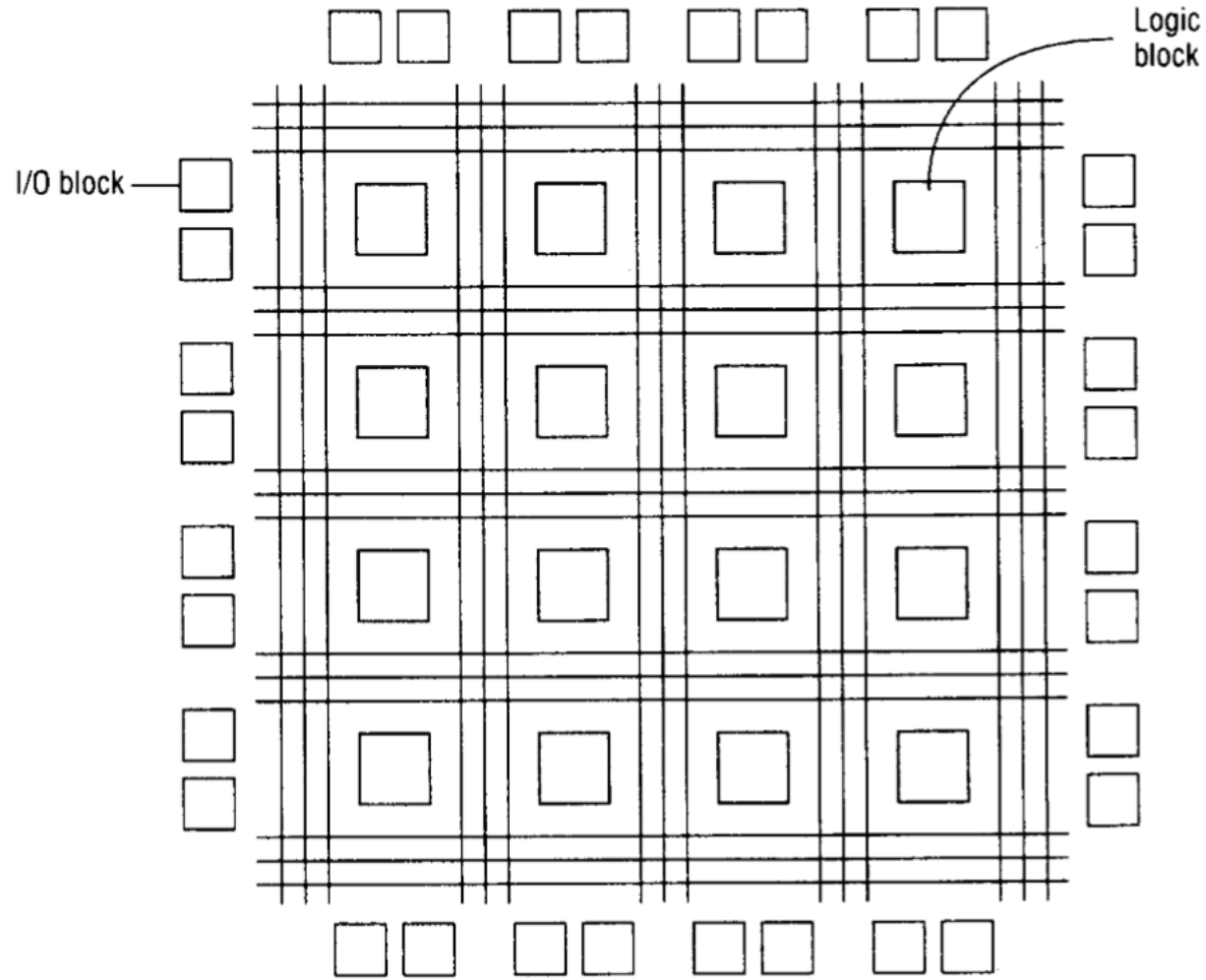
Example: NXP TriMedia TM1000



Programmable hardware: FPGA

FPGA – Basic Structure

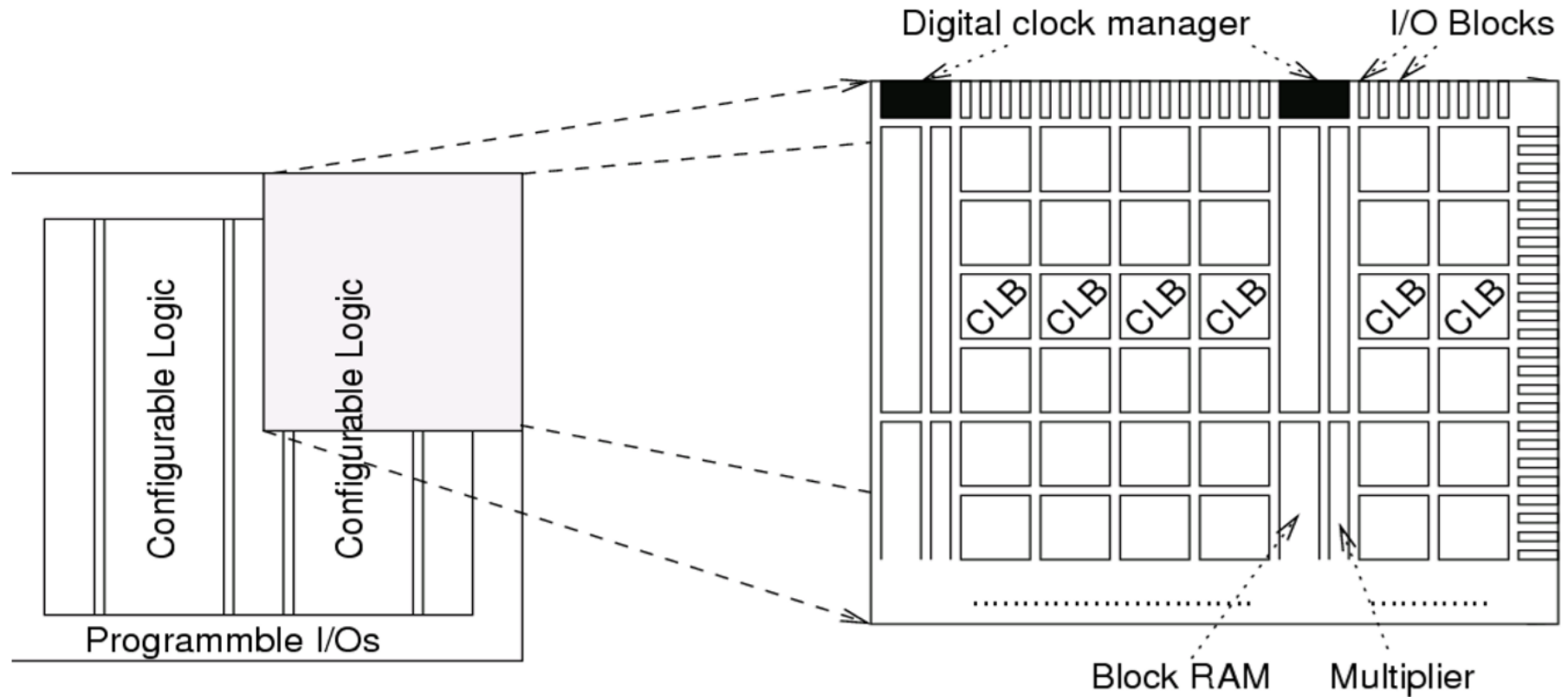
- ▶ Logic Units
- ▶ I/O Units
- ▶ Connections



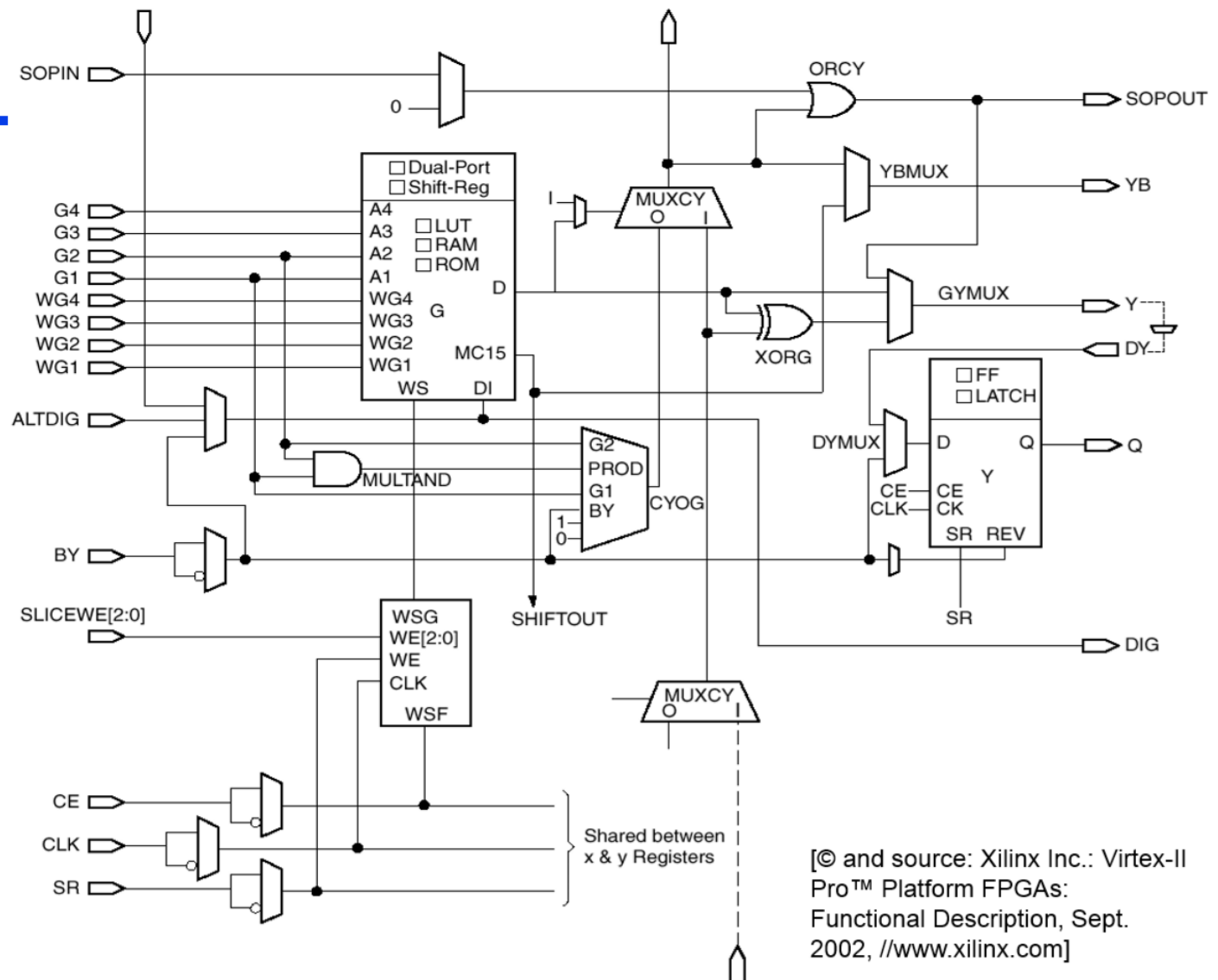
FPGA - Classification

- ▶ ***Granularity of logic units:***
 - Gate, tables, memory, functional blocks (ALU, control, data path, processor)
- ▶ ***Communication network:***
 - Crossbar, hierarchical mesh, tree
- ▶ ***Reconfiguration:***
 - fixed at production time, once at design time, dynamic during run-time

Floor-plan of VIRTEX II FPGAs



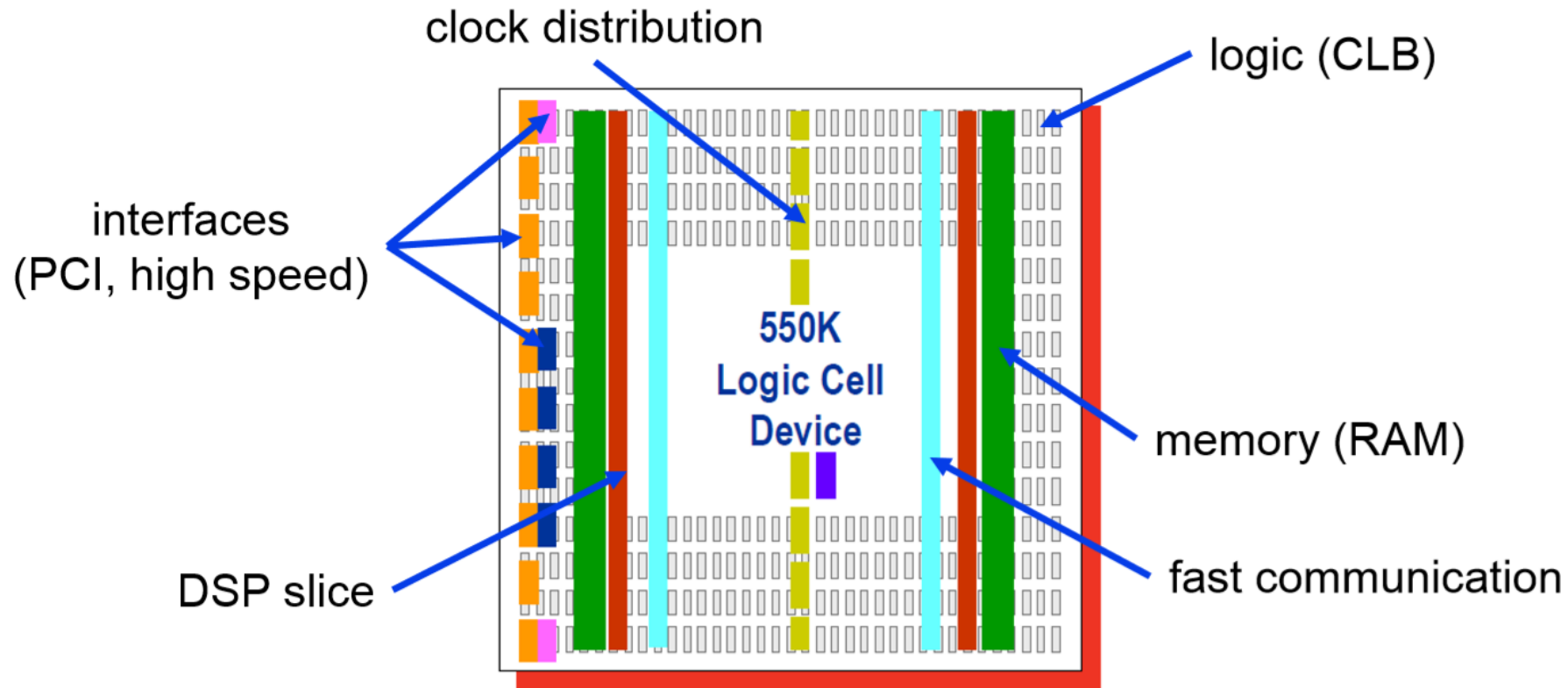
Virtex Logic Cell



[© and source: Xilinx Inc.: Virtex-II Pro™ Platform FPGAs: Functional Description, Sept. 2002, //www.xilinx.com]

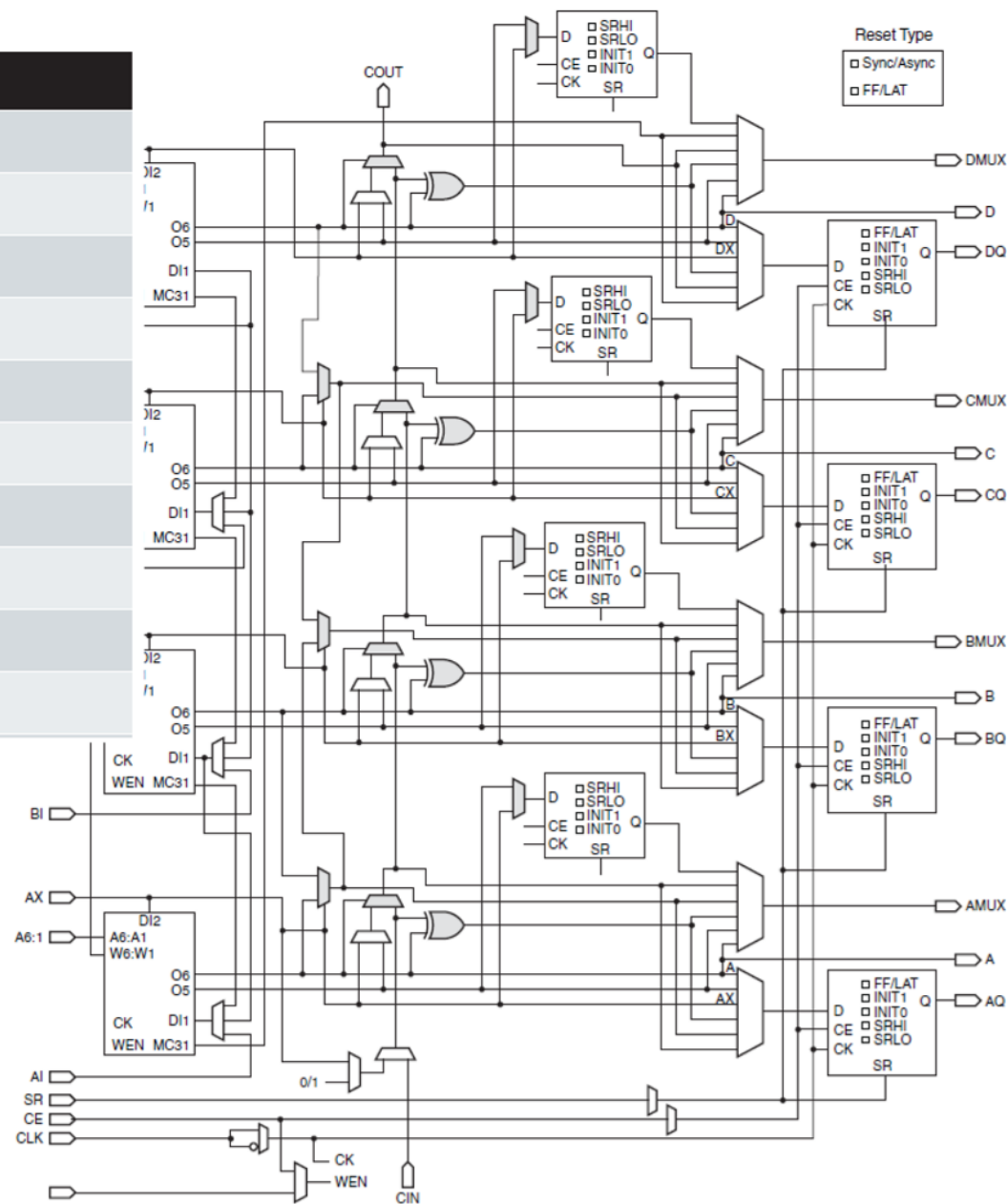
Example Virtex-6

- Combination of flexibility (CLB's), Integration and performance (heterogeneity of hard-IP Blocks)



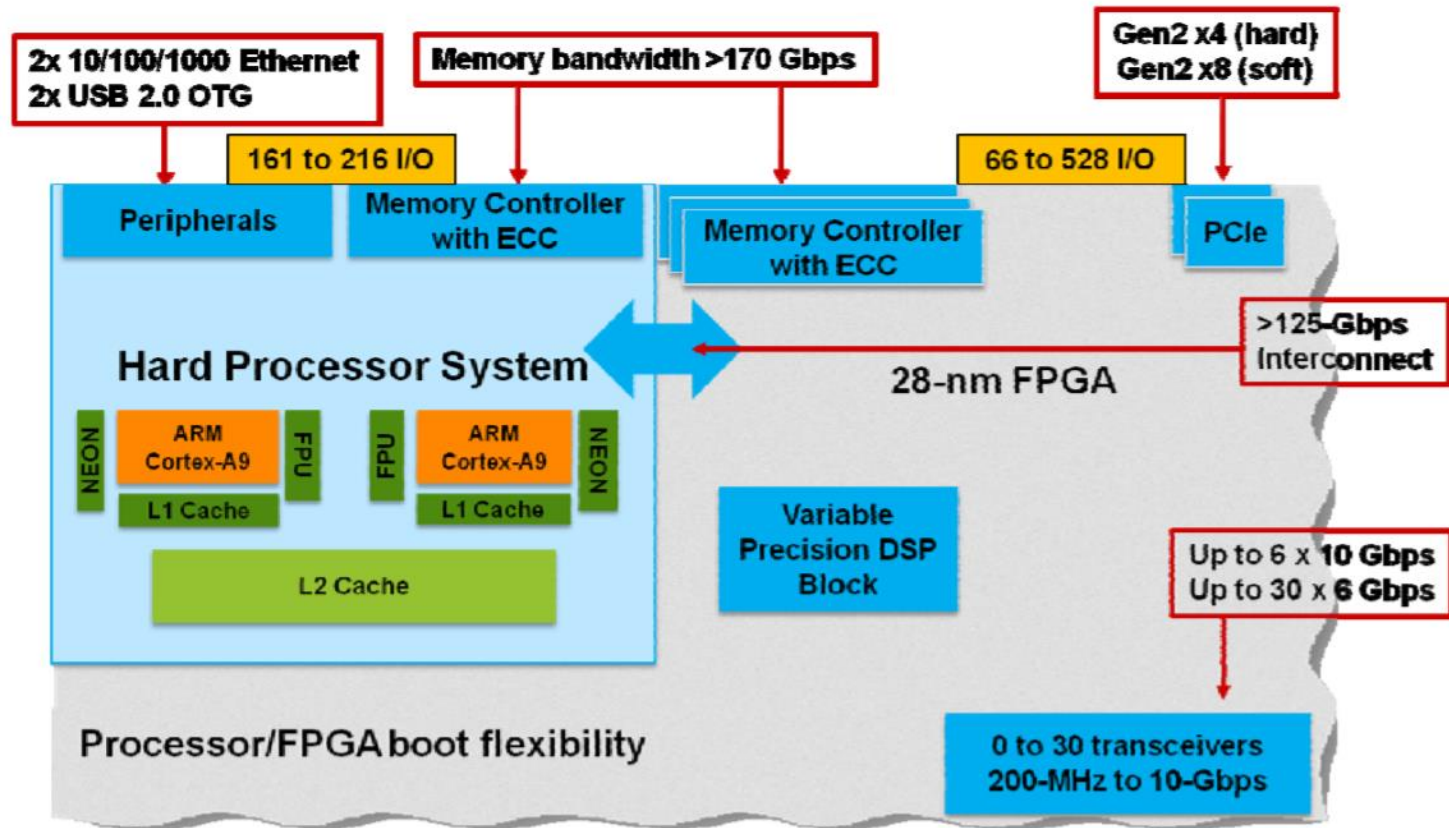
MAXIMUM CAPABILITY	VIRTEX-7 FPGAS
Logic Cells	1,955K
Block RAM	85Mb
DSP Slices	5,280
Peak DSP Performance (symmetric FIR)	6,737 GMACS
Transceiver Count	96
Peak Transceiver Speed	28.05Gbps
Peak Serial Bandwidth (full duplex)	2,784Gbps
PCI Express® Interface	Gen3 x8*
Memory Interface	1,866Mbps
I/O Pins	1,200

Virtex-6 CLB Slice



Configurable System-On-Chip

Example:
Altera's SoC
FPGA integrates a
dual-core ARM
Cortex-A9
processor system
with a low power
FPGA fabrics



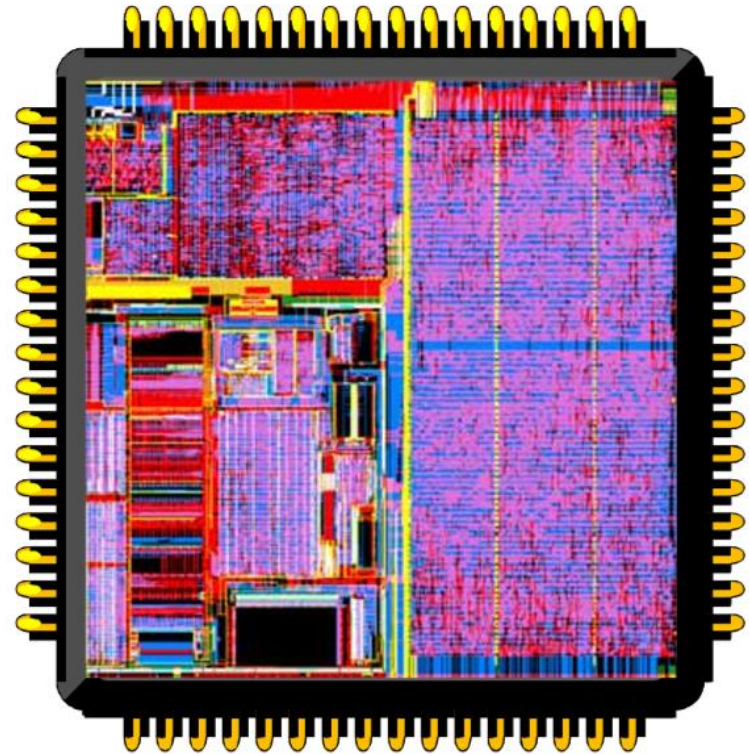
Application Specific Circuits (ASICs)

Custom-designed circuits necessary

- if ultimate speed or
- energy efficiency is the goal and
- large numbers can be sold.

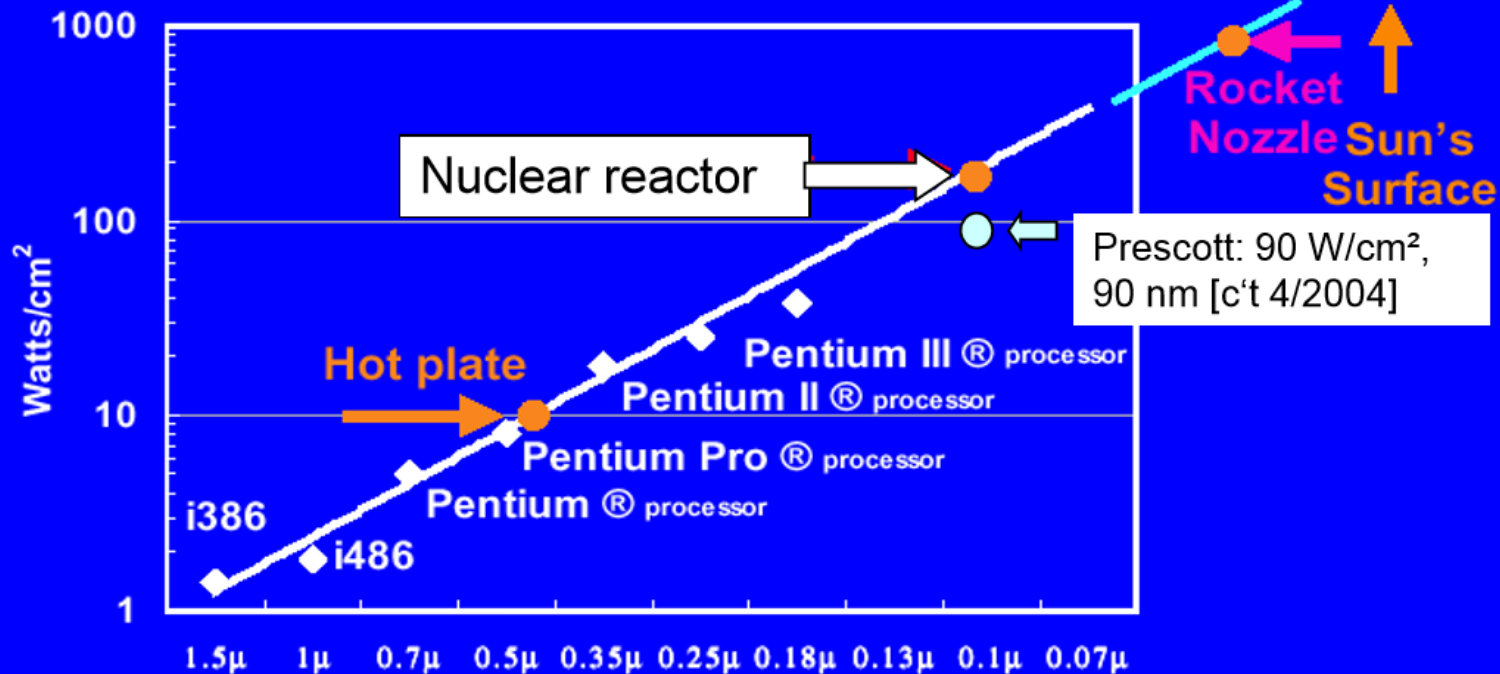
Approach suffers from

- long design times,
- lack of flexibility
(changing standards) and
- high costs
(e.g. Mill. \$ mask costs).



Power aware design & scheduling

PCs: Problem: Power density increasing



Surpassed hot-plate power density in 0.5μ
Not too long to reach nuclear reactor

PCs: Surpassed hot (kitchen) plate ...?

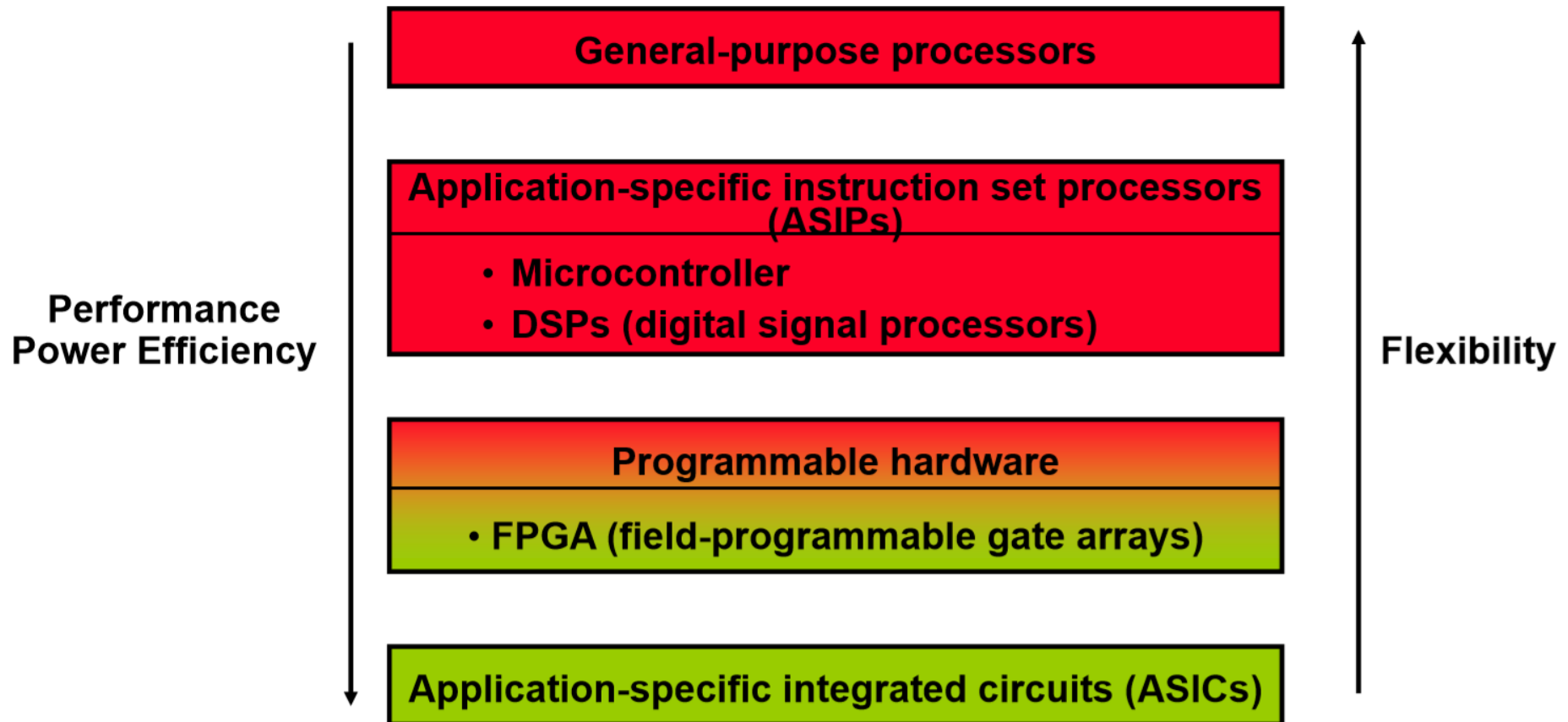
Why not use it?



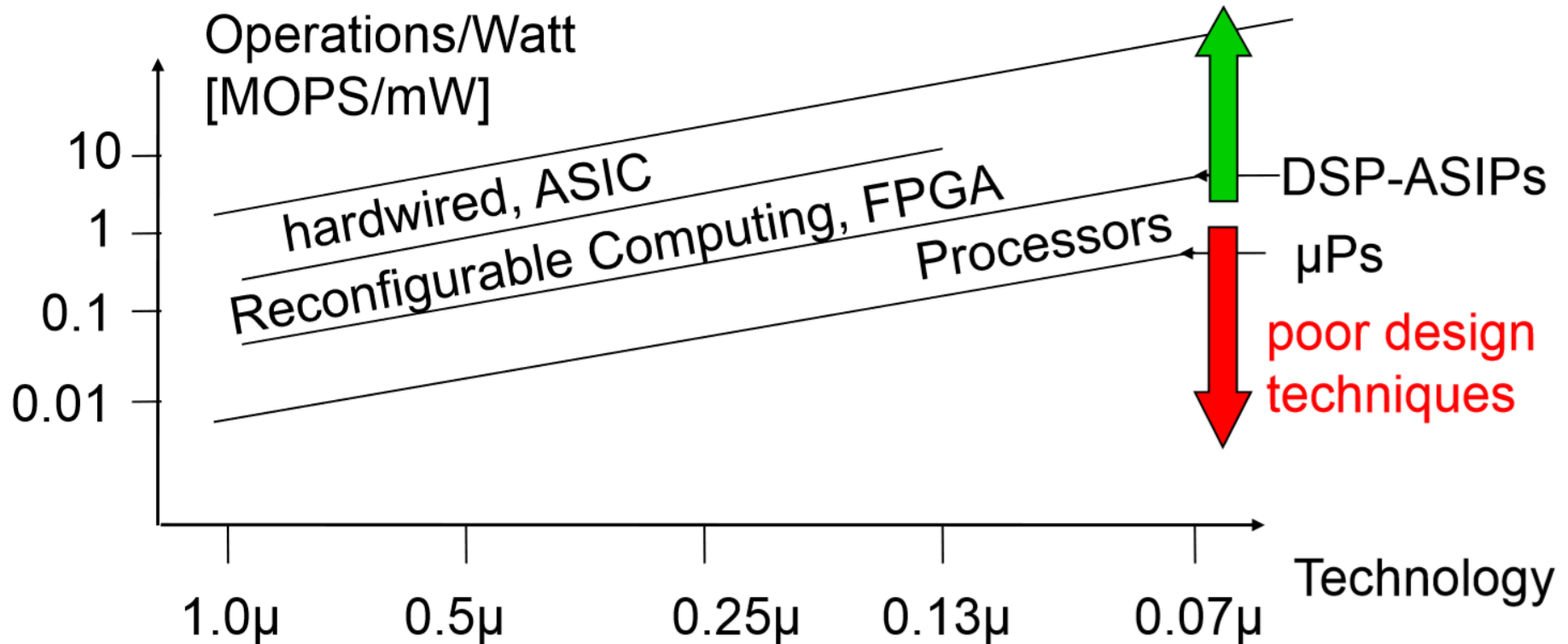
Strictly speaking, energy is not “consumed”, but converted from electrical energy into heat energy

http://www.phys.ncku.edu.tw/~htsu/humor/fry_egg.html

Implementation Alternatives



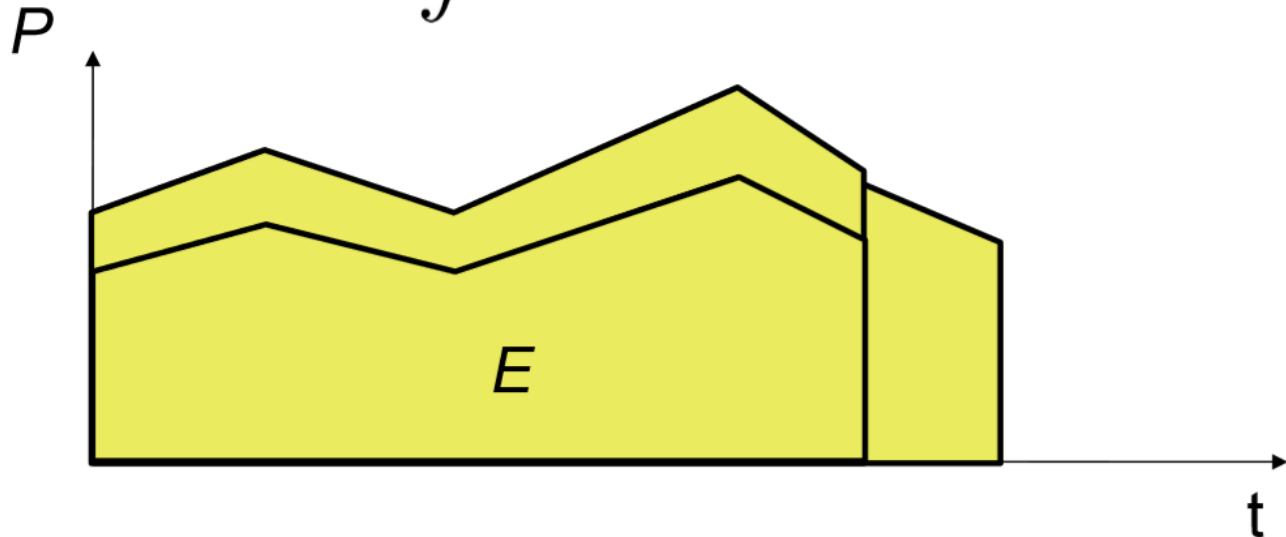
The Power/Flexibility Conflict



Necessary to **optimize HW and SW**.
Use **heterogeneous architectures**.
Apply **specialization techniques**.

Power and Energy are Related

$$E = \int P(t) dt$$



In many cases, faster execution also means less energy, but the opposite may be true if power has to be increased to allow faster execution.

Low Power vs. Low Energy

- ▶ Minimizing the *power consumption* is important for
 - the design of the power supply
 - the design of voltage regulators
 - the dimensioning of interconnect
 - cooling (short term cooling)
 - high cost (estimated to be rising at \$1 to \$3 per Watt for heat dissipation [Skadron et al. ISCA 2003])
 - limited space
- ▶ Minimizing the *energy consumption* is important due to
 - restricted availability of energy (mobile systems)
 - limited battery capacities (only slowly improving)
 - very high costs of energy (solar panels, in space)
 - long lifetimes, low temperatures

Dynamic Voltage Scaling (DVS)

Power consumption of CMOS circuits (ignoring leakage):

$$P \sim \alpha C_L V_{dd}^2 f$$

V_{dd} : supply voltage
 α : switching activity
 C_L : load capacity
 f : clock frequency

Delay for CMOS circuits:

$$\tau \sim C_L \frac{V_{dd}}{(V_{dd} - V_T)^2}$$

V_{dd} : supply voltage
 V_T : threshold voltage
 $V_T \ll V_{dd}$

Decreasing V_{dd} reduces P quadratically (f constant).

The gate delay increases reciprocally with decreasing V_{dd} .

Maximal frequency $f_{\max}$ decreases linearly with decreasing V_{dd} .

Potential for Energy Optimization: DVS

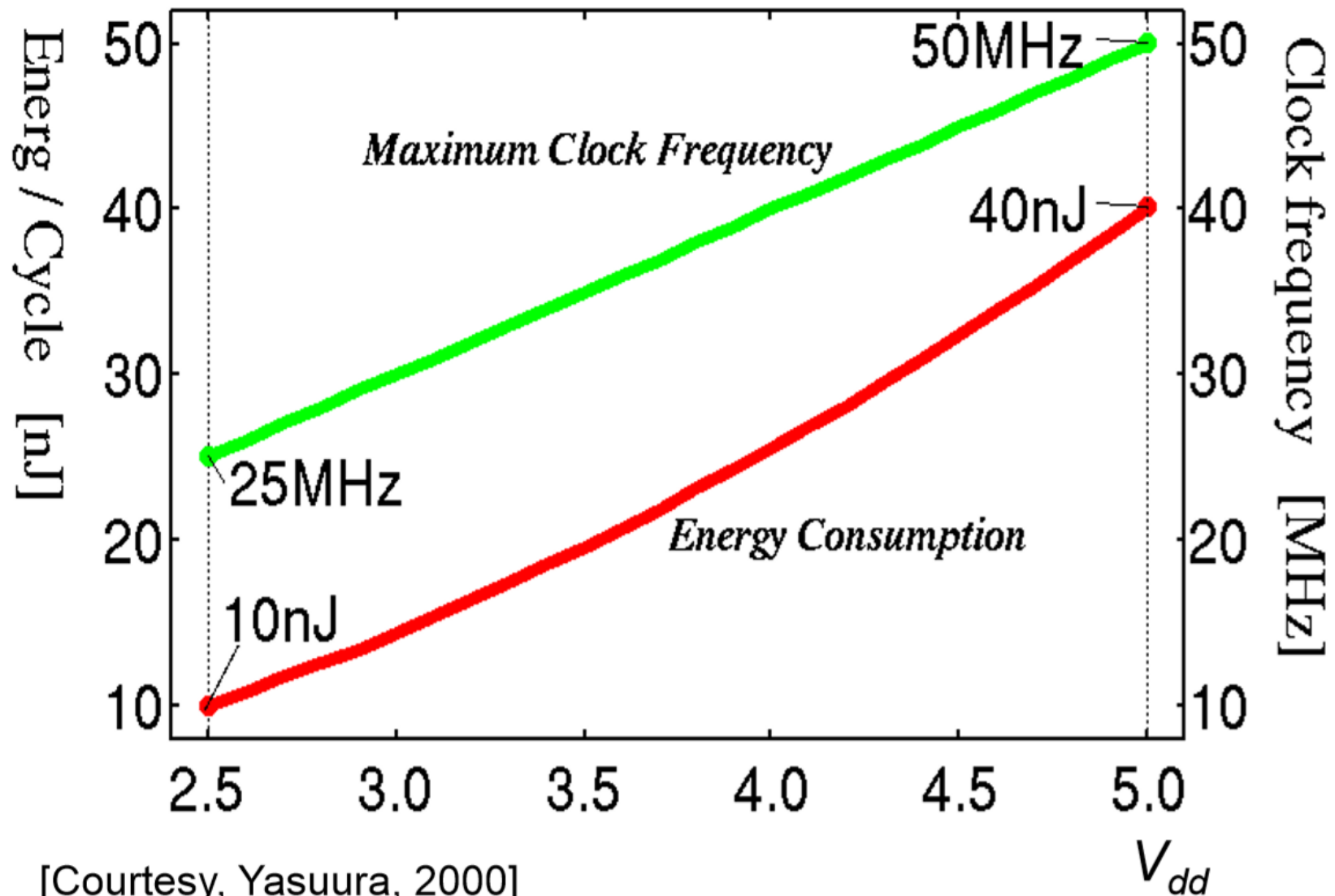
$$P \sim \alpha C_L V_{dd}^2 f$$

$$E \sim \alpha C_L V_{dd}^2 f t = \alpha C_L V_{dd}^2 (\#cycles)$$

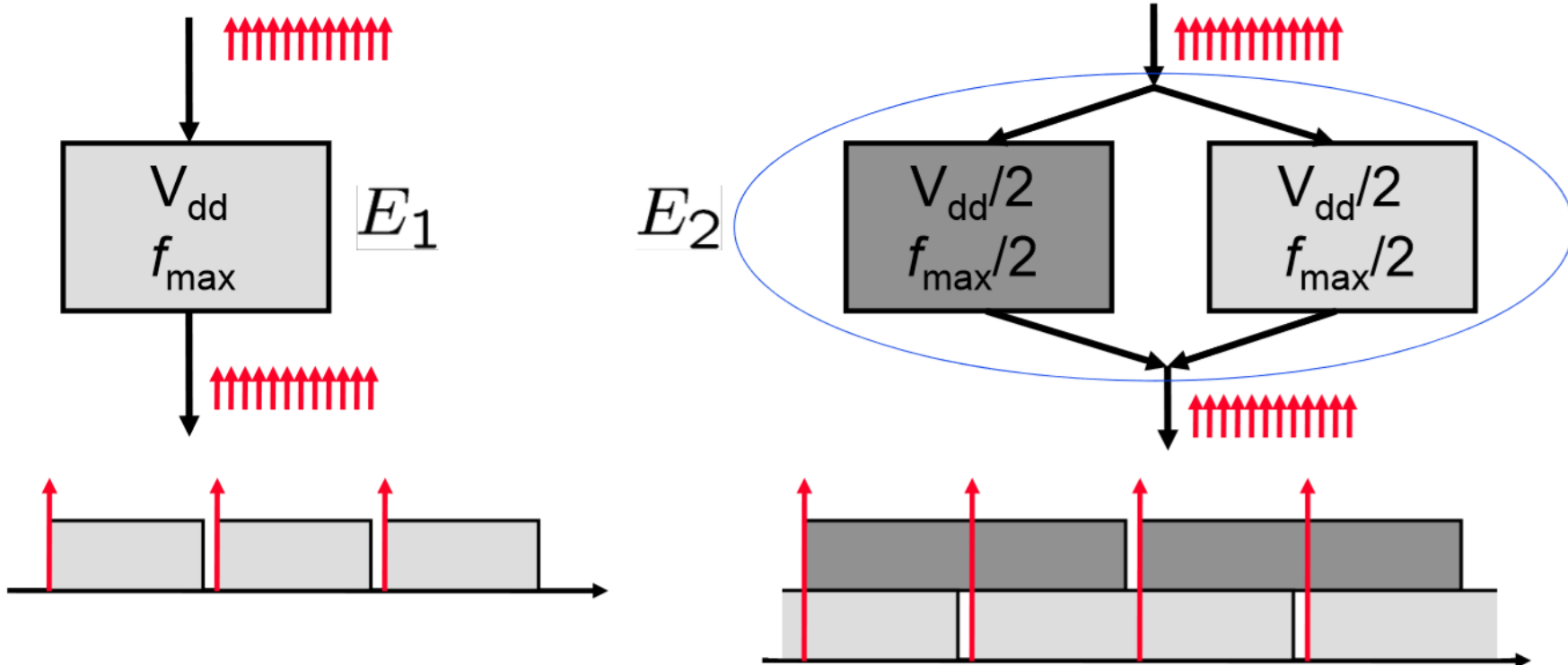
Saving energy for a given task:

- Reduce the supply voltage V_{dd}
- Reduce the number of cycles $\#cycles$

Example: Voltage Scaling



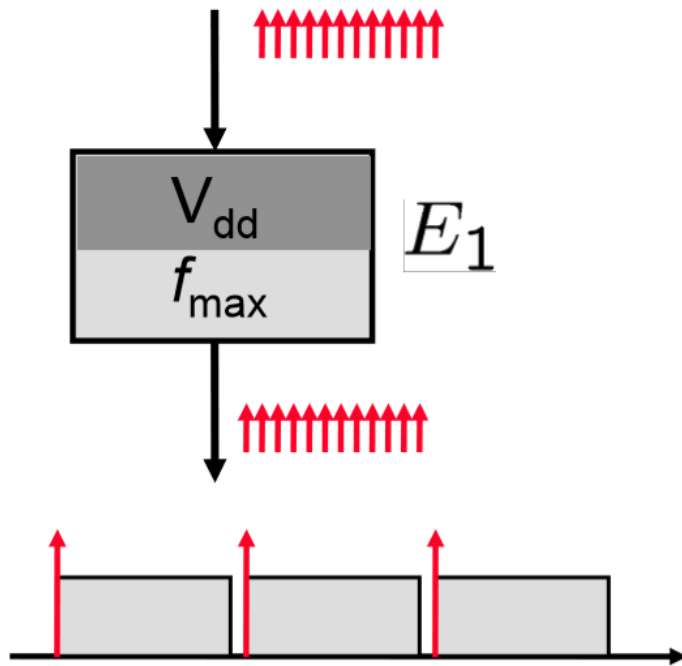
Use of Parallelism



$$E \sim V_{dd}^2 (\text{\#cycles})$$

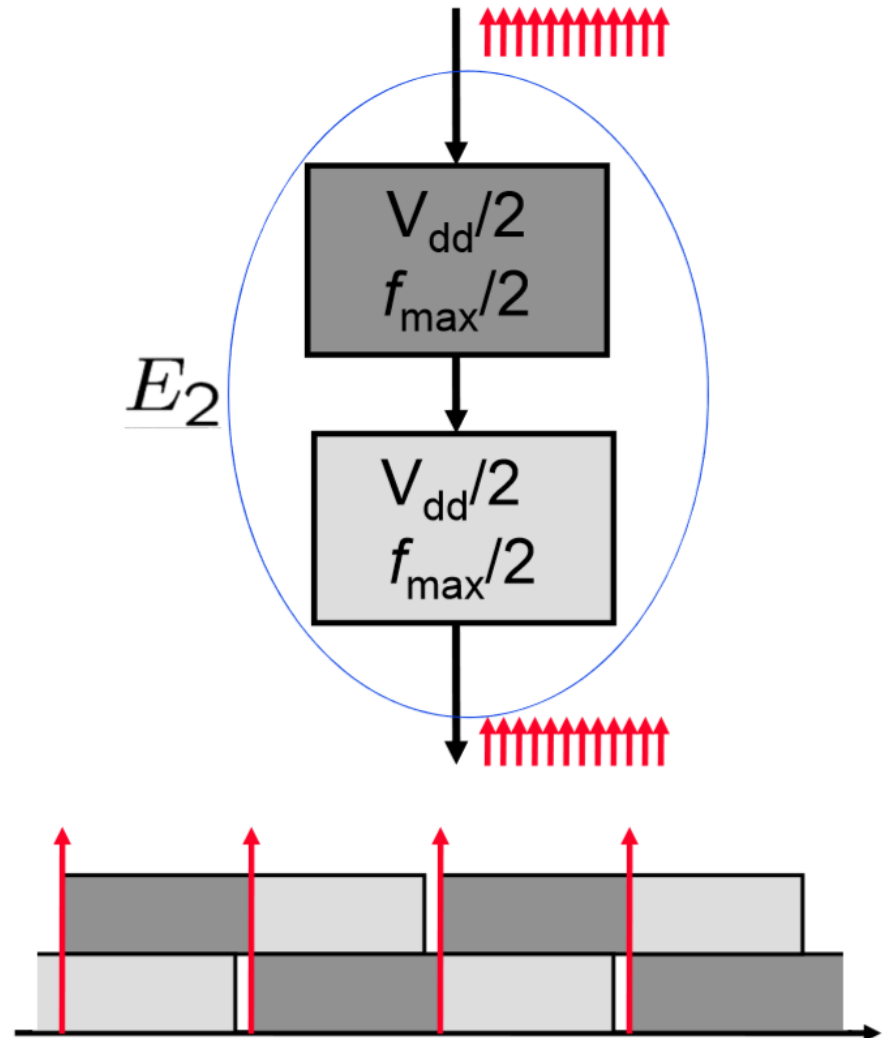
$$E_2 = \frac{1}{4} E_1$$

Use of Pipelining



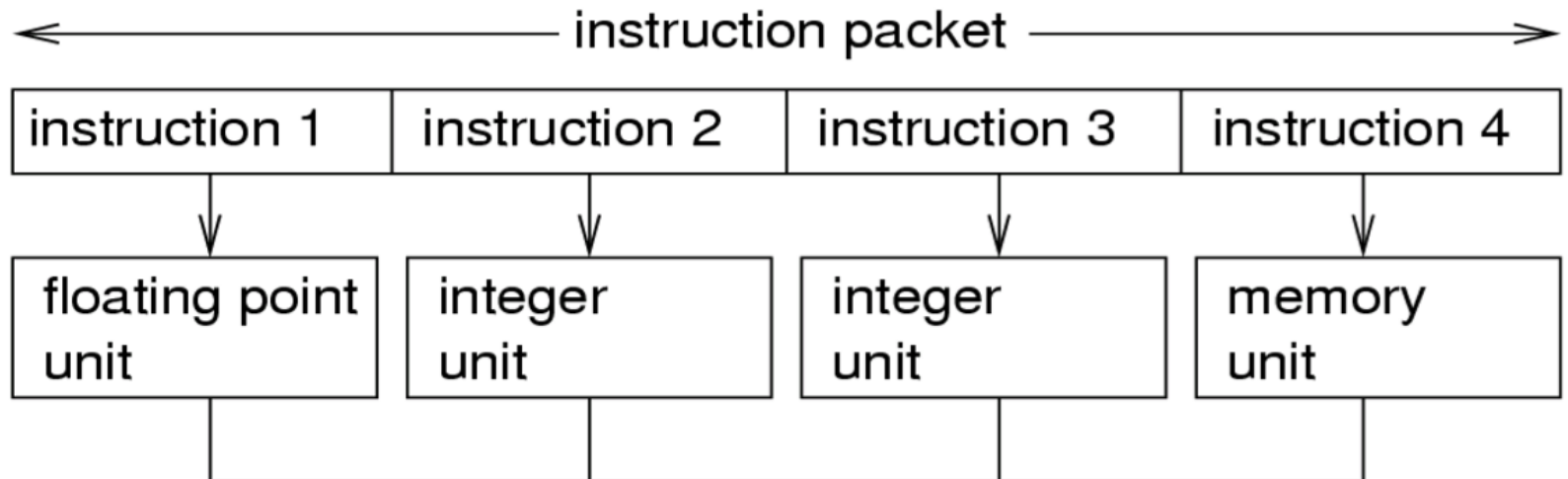
$$E \sim V_{dd}^2 (\text{\#cycles})$$

$$E_2 = \frac{1}{4} E_1$$

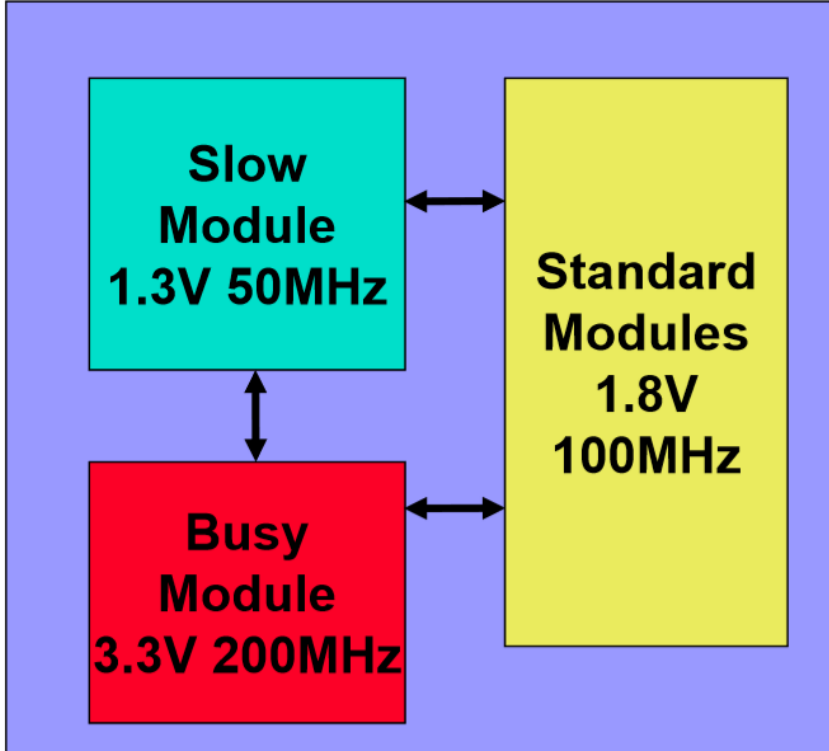


VLIW Architectures

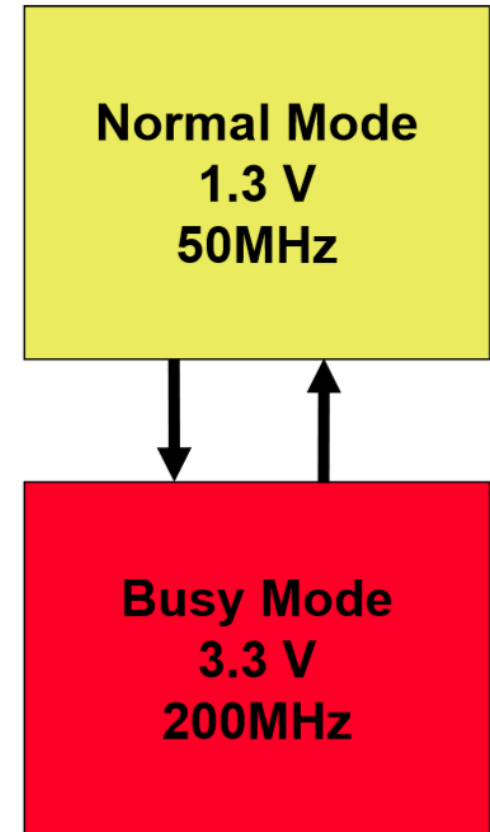
- ▶ *Large degree of parallelism*
 - many computational units, (deeply) pipelined
- ▶ *Simple hardware architecture*
 - explicit parallelism (parallel instruction set)
 - parallelization is done offline (compiler)



Spatial vs. Dynamic Voltage Management



Not all components require same performance.

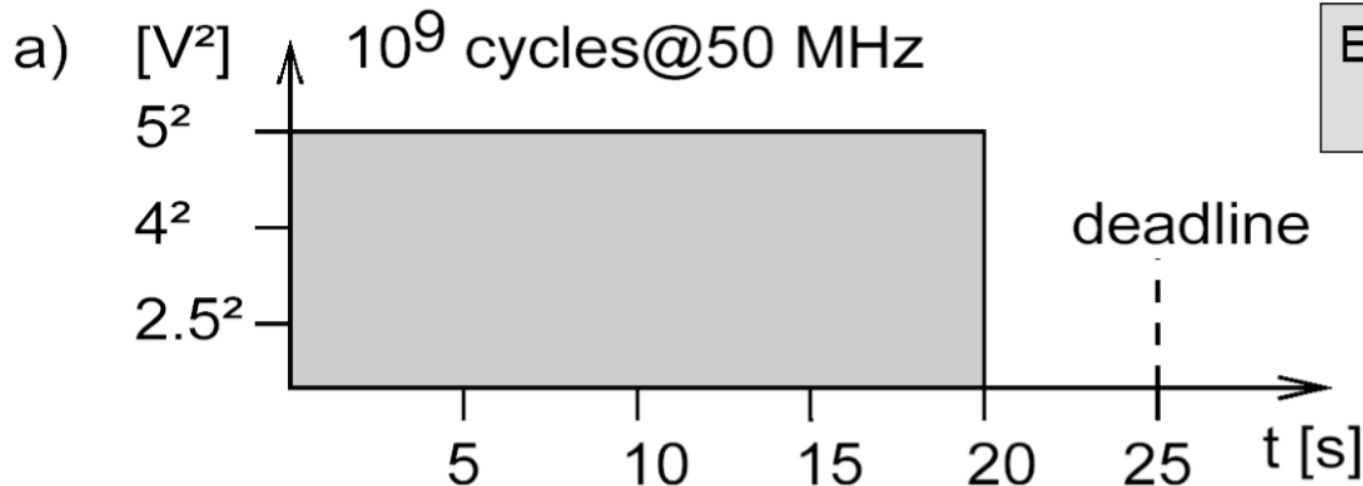


Required performance may change over time

DVS Example: a) Complete task ASAP

V_{dd} [V]	5.0	4.0	2.5
Energy per cycle [nJ]	40	25	10
f_{max} [MHz]	50	40	25
cycle time [ns]	20	25	40

Task that needs to execute 10^9 cycles within 25 seconds.

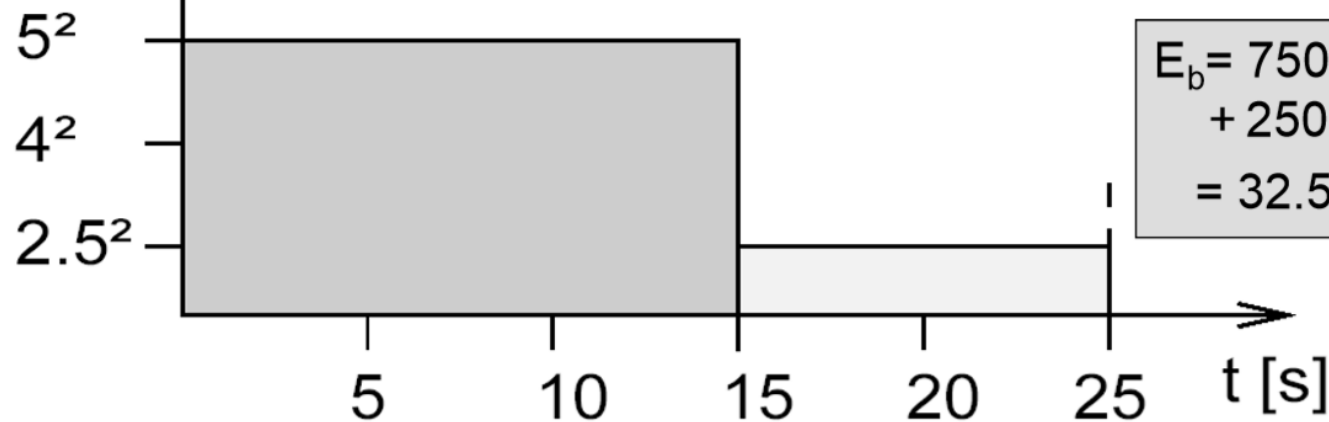


$$E_a = 10^9 \times 40 \times 10^{-9} = 40 \text{ [J]}$$

DVS Example: b) Two voltages

V_{dd} [V]	5.0	4.0	2.5
Energy per cycle [nJ]	40	25	10
f_{max} [MHz]	50	40	25
cycle time [ns]	20	25	40

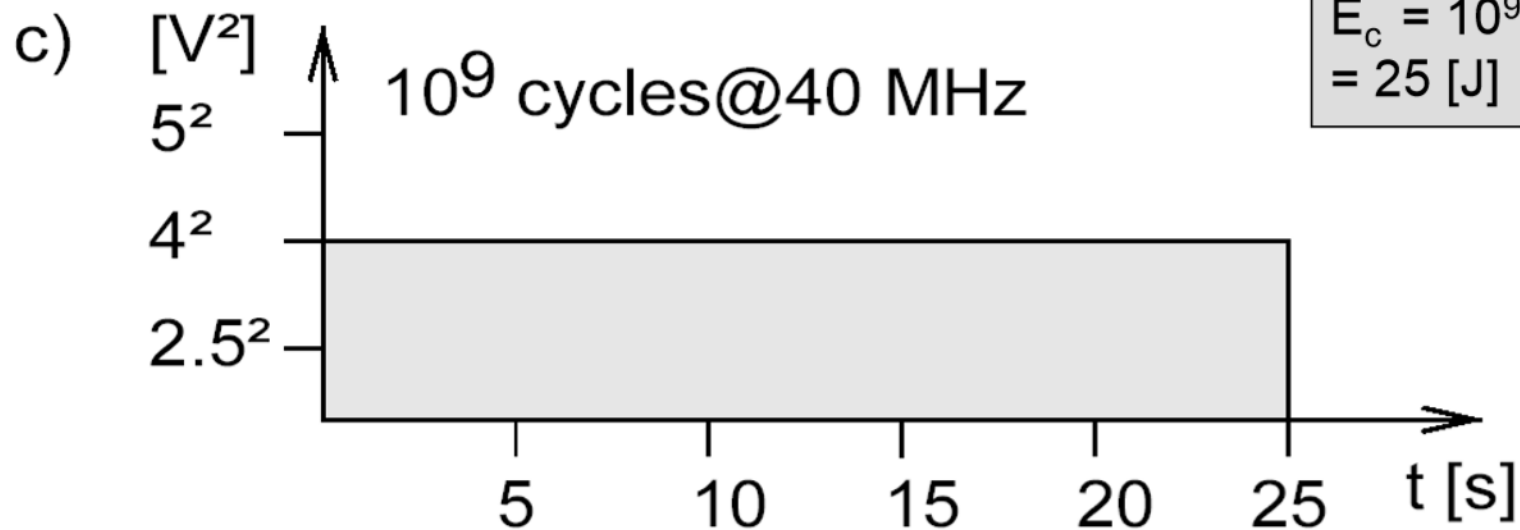
b) [V²] 750M cycles @ 50 MHz + 250M cycles @ 25



$$\begin{aligned} E_b &= 750 \cdot 10^6 \times 40 \times 10^{-9} \\ &\quad + 250 \cdot 10^6 \times 10 \times 10^{-9} \\ &= 32.5 \text{ [J]} \end{aligned}$$

DVS Example: c) Optimal Voltage

V_{dd} [V]	5.0	4.0	2.5
Energy per cycle [nJ]	40	25	10
f_{max} [MHz]	50	40	25
cycle time [ns]	20	25	40

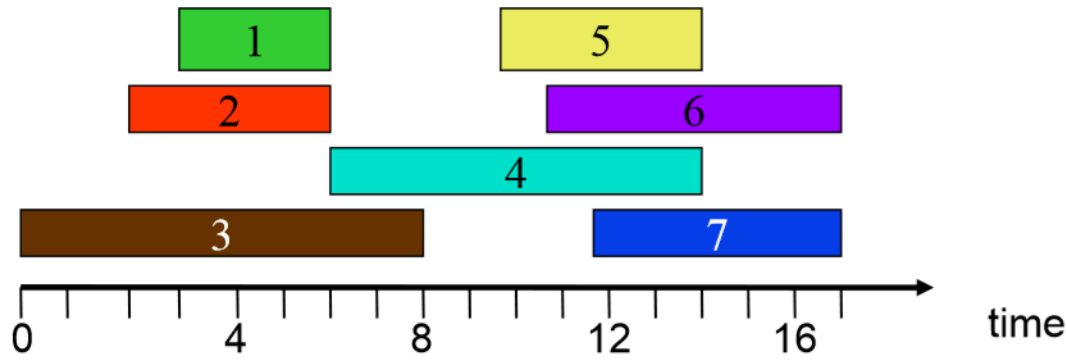


DVS: Offline Scheduling on One Processor

- ▶ Let us model a set of independent tasks as follows:
 - We suppose that a task $v_i \in V$
 - requires c_i computation time at normalized processor frequency 1
 - arrives at time a_i
 - has (absolute) deadline constraint d_i
- ▶ How do we schedule these tasks such that all these tasks can be finished **no later than their deadlines** and the energy consumption is **minimized**?
 - YDS Algorithm from “A Scheduling Model for Reduce CPU Energy”, Frances Yao, Alan Demers, and Scott Shenker, FOCS 1995.”

If possible, running at a constant frequency (voltage) minimizes the energy consumption for dynamic voltage scaling.

YDS Algorithm for Offline Scheduling



3,6,5
2,6,3
0,8,2
6,14,6
10,14,6
11,17,2
12,17,2
a_i, d_i, c_i

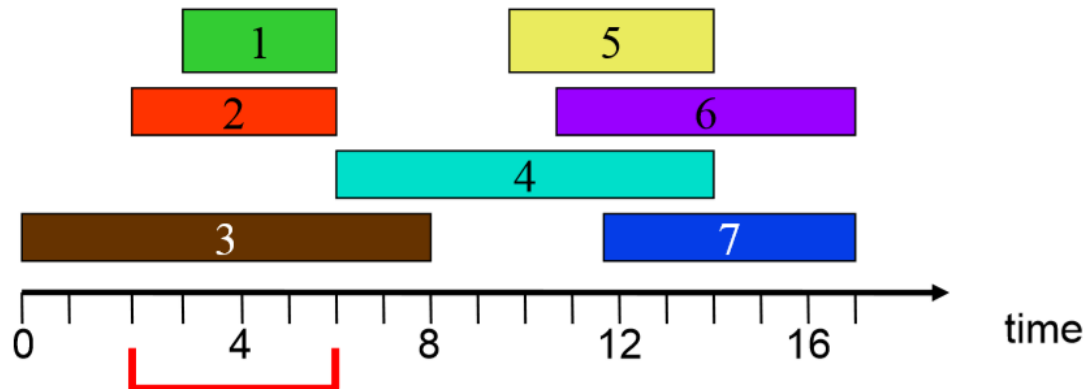
- Define **intensity** $G([z, z'])$ in some time interval $[z, z']$:
 - average accumulated execution time of all tasks that have arrival and deadline in $[z, z']$ relative to the length of the interval $z'-z$

$$V'([z, z']) = \{v_i \in V : z \leq a_i < d_i \leq z'\}$$

$$G([z, z']) = \sum_{v_i \in V'([z, z'])} c_i / (z' - z)$$

YDS Algorithm for Offline Scheduling

- **Step 1:** Execute jobs in the interval with the highest intensity by using the earliest-deadline first schedule and running at the intensity as the frequency.



3,6,5

2,6,3

0,8,2

6,14,6

10,14,6

11,17,2

12,17,2

a_i, d_i, c_i

$$G([0,6]) = (5+3)/6=8/6, G([0,8]) = (5+3+2)/(8-0) = 10/8,$$

$$G([0,14]) = (5+3+2+6+6)/14=11/7, G([0,17]) = (5+3+2+6+6+2+2)/17=26/17$$

$$G([2, 6]) = (5+3)/(6-2)=2, G([2,14]) = (5+3+6+6) / (14-2) = 5/3,$$

$$G([2,17]) = (5+3+6+6+2+2)/15=24/15$$

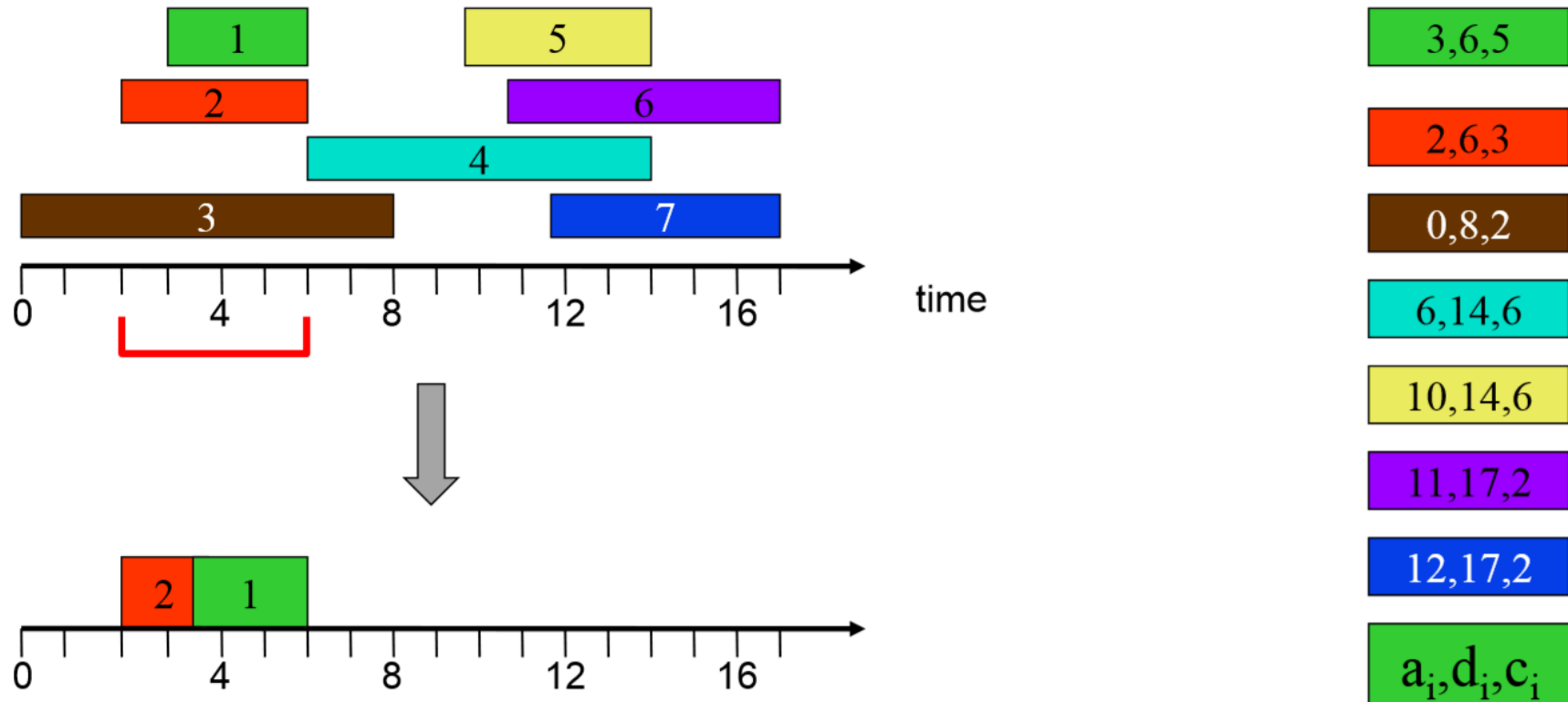
$$G([3,6]) = 5/3, G([3,14]) = (5+6+6)/(14-3) = 17/11, G([3,17])=(5+6+6+2+2)/14=21/14$$

$$G([6,14]) = 12/(14-6)=12/8, G([6,17]) = (6+6+2+2)/(17-6)=16/11$$

$$G([10,14]) = 6/4, G([10,17]) = 10/7, G([11,17]) = 4/6, G([12,17]) = 2/5$$

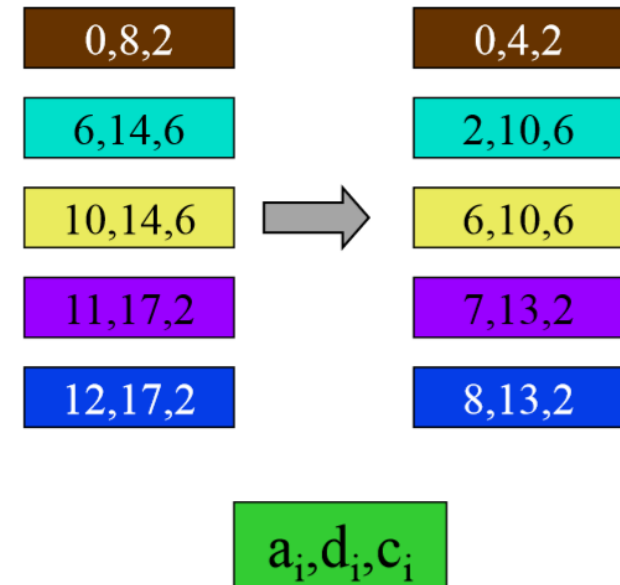
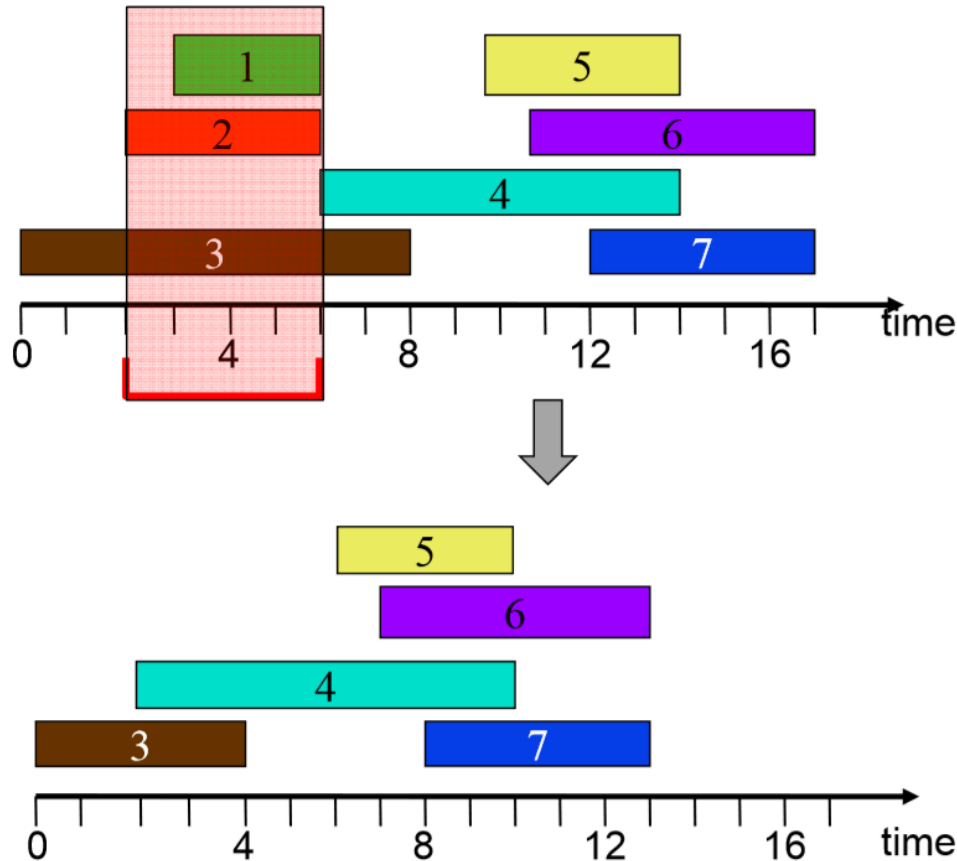
YDS Algorithm for Offline Scheduling

- **Step 1:** Execute jobs in the interval with the highest intensity by using the earliest-deadline first schedule and running at the intensity as the frequency.



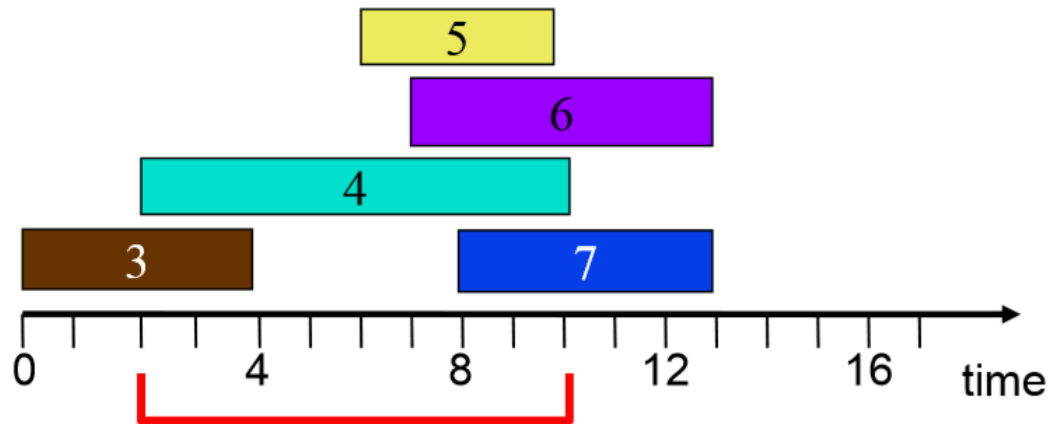
YDS Algorithm for Offline Scheduling

- **Step 2:** Adjust the arrival times and deadlines by excluding the possibility to execute at the previous critical intervals.



YDS Algorithm for Offline Scheduling

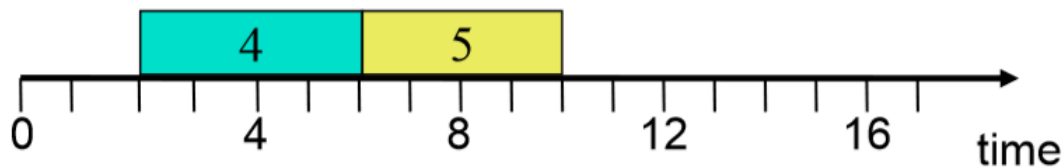
- **Step 3:** Run the algorithm for the revised input again



$$G([0,4])=2/4, G([0,10]) = 14/10, G([0,13])=18/13$$

$$G([2,10])=12/8, G([2,13]) = 16/11, G([6,10])=6/4$$

$$G([6,13])=10/7, G([7,13])=4/6, G([8,13])=4/5$$



0,4,2

2,10,6

6,10,6

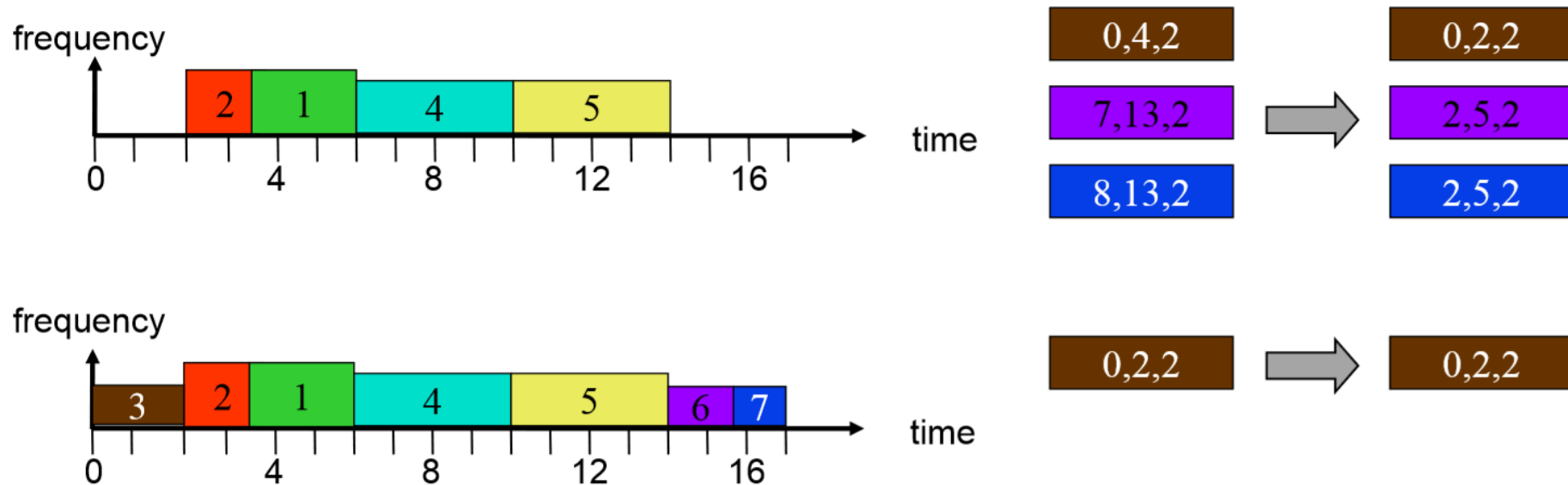
7,13,2

8,13,2

a_i, d_i, c_i

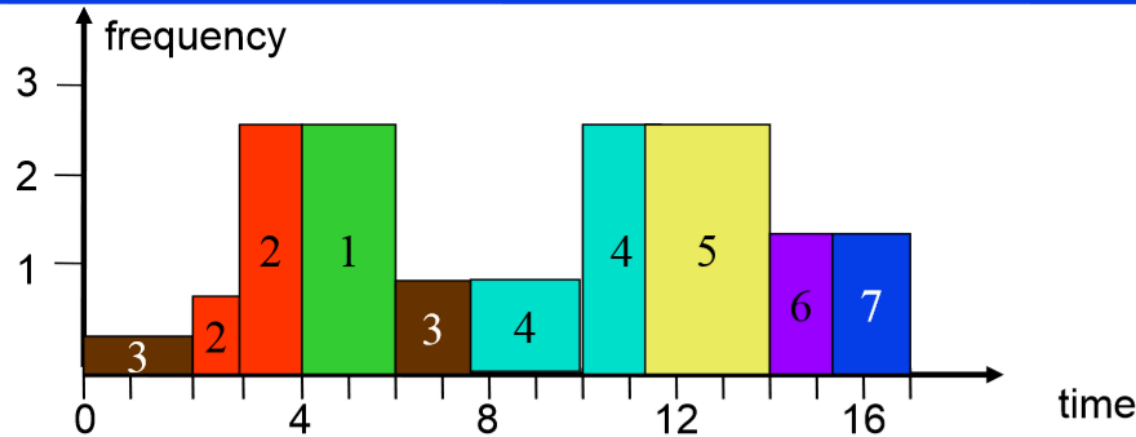
YDS Algorithm for Offline Scheduling

- **Step 3:** Run the algorithm for the revised input again
- **Step 4:** Put pieces together



	v_1	v_2	v_3	v_4	v_5	v_6	v_7
frequency	2	2	1	1.5	1.5	4/3	4/3

DVS: Online Scheduling on One Processor



3,6,5

2,6,3

0,8,2

6,14,6

10,14,6

11,17,2

12,17,2

a_i, d_i, c_i

► Continuously update to the best schedule for all arrived tasks

- Time 0: task v_3 is executed at $2/8$
- Time 2: task v_2 arrives
 - $G([2,6]) = 3/4$, $G([2,8]) = 4.5/6 = 3/4 \Rightarrow$ execute v_2 at $3/4$
- Time 3: task v_1 arrives
 - $G([3,6]) = (5+3-3/4)/3 = 29/12$, $G([3,8]) < G([3,6]) \Rightarrow$ execute v_2 and v_1 at $29/12$
- Time 6: task v_4 arrives
 - $G([6,8]) = 1.5/2$, $G([6,14]) = 7.5/8 \Rightarrow$ execute v_3 and v_4 at $15/16$
- Time 10: task v_5 arrives
 - $G([10,14]) = 39/16 \Rightarrow$ execute v_4 and v_5 at $39/16$
- Time 11 and Time 12
 - The arrival of v_6 and v_7 does not change the critical interval
- Time 14:
 - $G([14,17]) = 4/3 \Rightarrow$ execute v_6 and v_7 at $4/3$

Remarks on YDS Algorithm

► *Offline*

- The algorithm guarantees the minimal energy consumption while satisfying the timing constraints
- The time complexity is $O(N^3)$, where N is the number of tasks in V
 - Finding the critical interval can be done in $O(N^2)$
 - The number of iterations is at most N
- Exercise:
 - For periodic real-time tasks with deadline=period, running at ***constant speed with 100% utilization*** under EDF has minimum energy consumption while satisfying the timing constraints.

► *Online*

- Compared to the optimal offline solution, the on-line schedule uses at most 27 times of the minimal energy consumption.

Dynamic Power Management (DPM)

Dynamic Power management tries to assign optimal **power saving states**

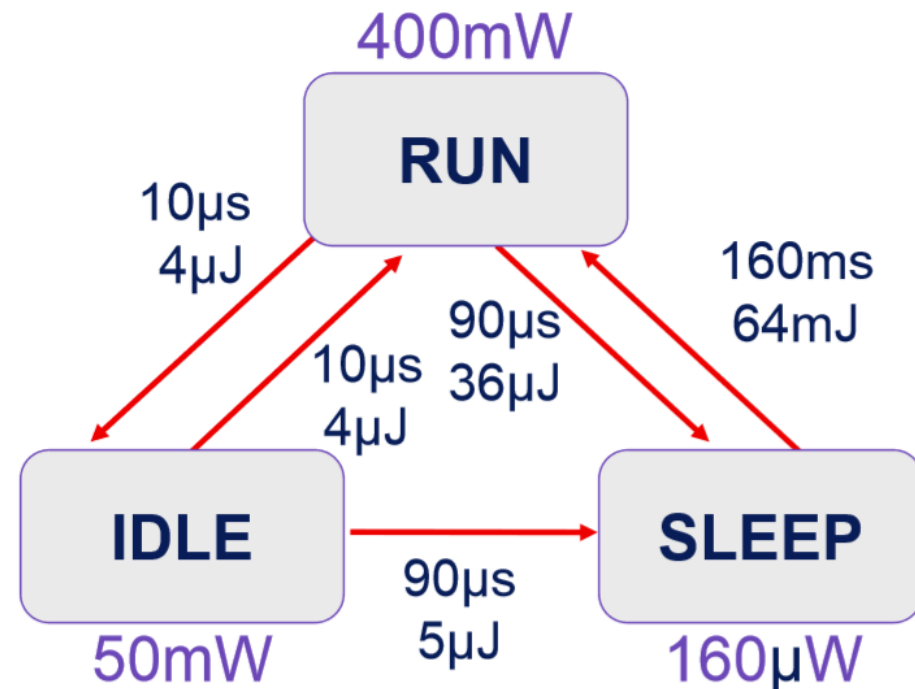
Requires Hardware Support

Example: StrongARM SA1100

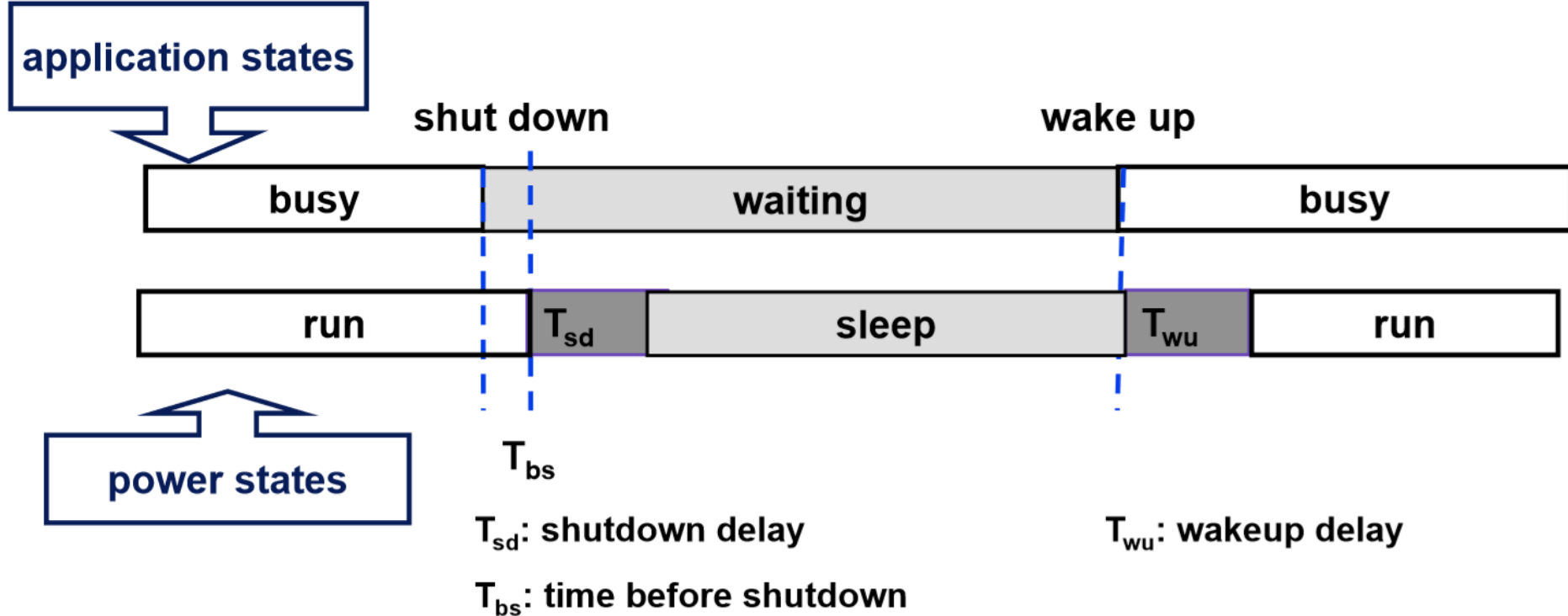
RUN: operational

IDLE: a SW routine may stop the CPU when not in use, while monitoring interrupts

SLEEP: Shutdown of on-chip activity

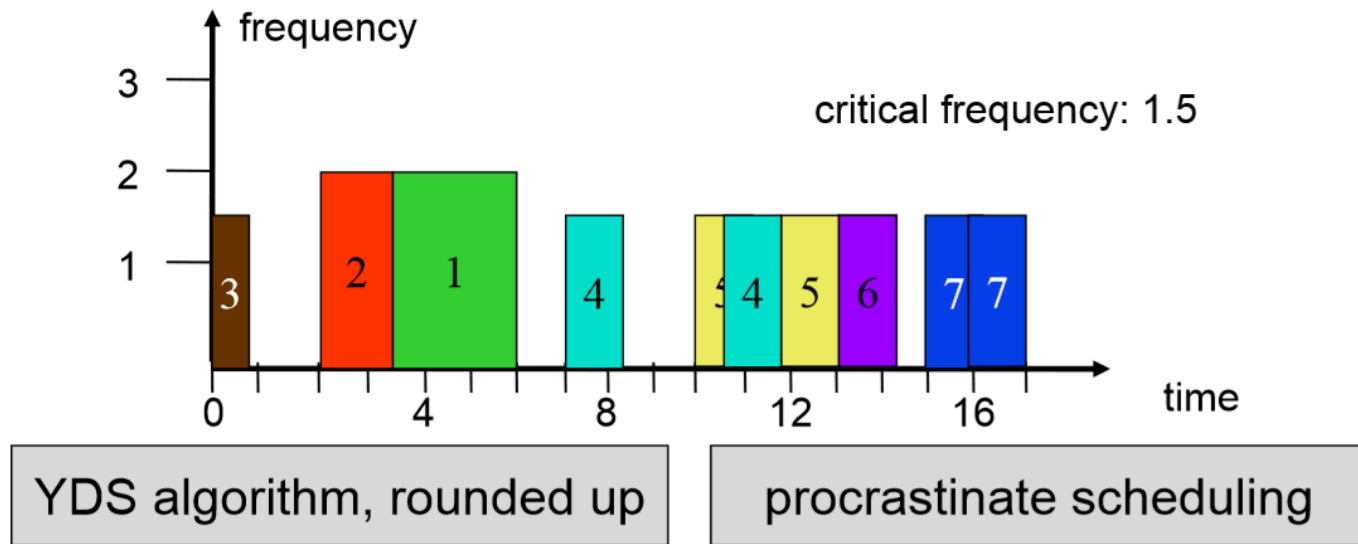


Reduce Power According to Workload



Desired: Shutdown only during long idle times
→ Tradeoff between savings and overhead

Procrastination Schedule



- ▶ Execute by using voltages higher or equal to the critical voltage only
 - apply YDS algorithm
 - round up voltages lower than the critical voltage
- ▶ Procrastinate the execution of tasks to aggregate enough time for sleeping
 - Try to reduce the number of times to turn on/off
 - Sleep as long as possible

A separate problem: how to choose the critical voltage ($V_c \rightarrow F_c$)

Спасибо за внимание!

ИНФОРМАЦИОННО-УПРАВЛЯЮЩИЕ СИСТЕМЫ РЕАЛЬНОГО ВРЕМЕНИ

Лекция 6:

Оценка наихудшего времени выполнения программ (WCET)

Кафедра АСВК,
Лаборатория Вычислительных Комплексов
Балашов В.В.

Почему важен WCET

- *Время выполнения* программ играет ключевую роль при анализе систем реального времени
- Это время, например, встречается в формуле:

Наихудшее время отклика

Период

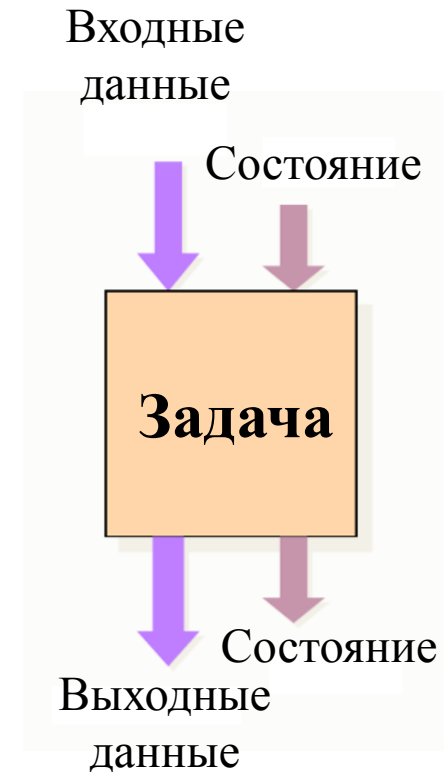
Наихудшее время выполнения

$$R_i = C_i + \sum_{j \in hp(i)} \lceil R_i / T_j \rceil C_j$$

Откуда берутся значения C_i, C_j ?

Простейшая вычислительная задача

- Входные данные доступны в момент старта
- Выходные данные готовы в момент завершения
- Нет блокировок в процессе выполнения
- Нет синхронизации или обмена данными в процессе выполнения
- Время выполнения зависит только от:
 - входных данных
 - состояния задачи в момент старта (внешние воздействия отсутствуют)

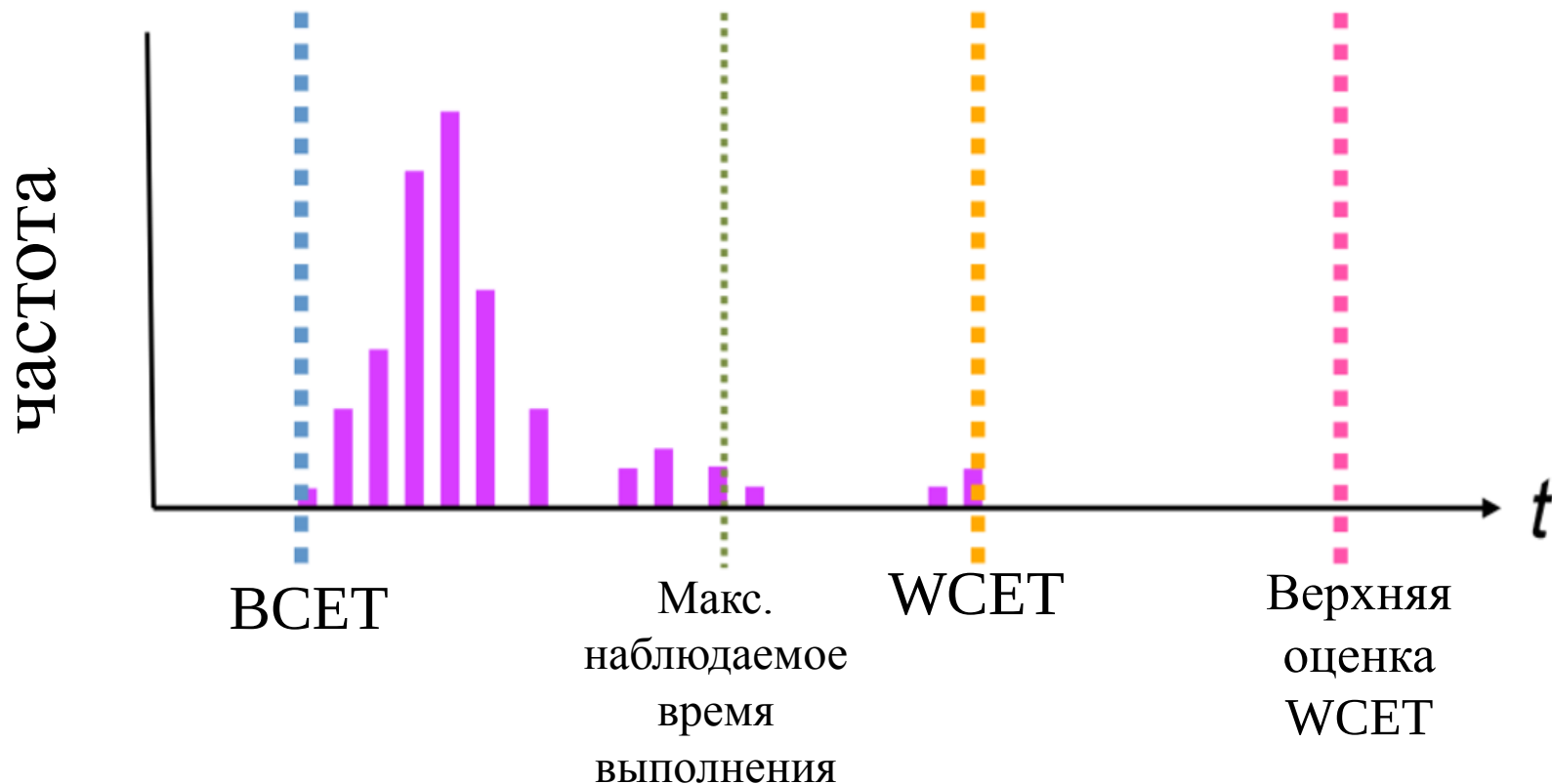


Наихудшее (максимальное) время выполнения

Наихудшее время выполнения программного кода (worst case execution time, WCET) – это максимальное время, которое требуется для выполнения

- данного фрагмента кода
- в данном контексте (входные данные, состояние)
- на заданном аппаратном вычислителе

Время выполнения: терминология



BCET – best-case execution time (минимальное время выполнения)

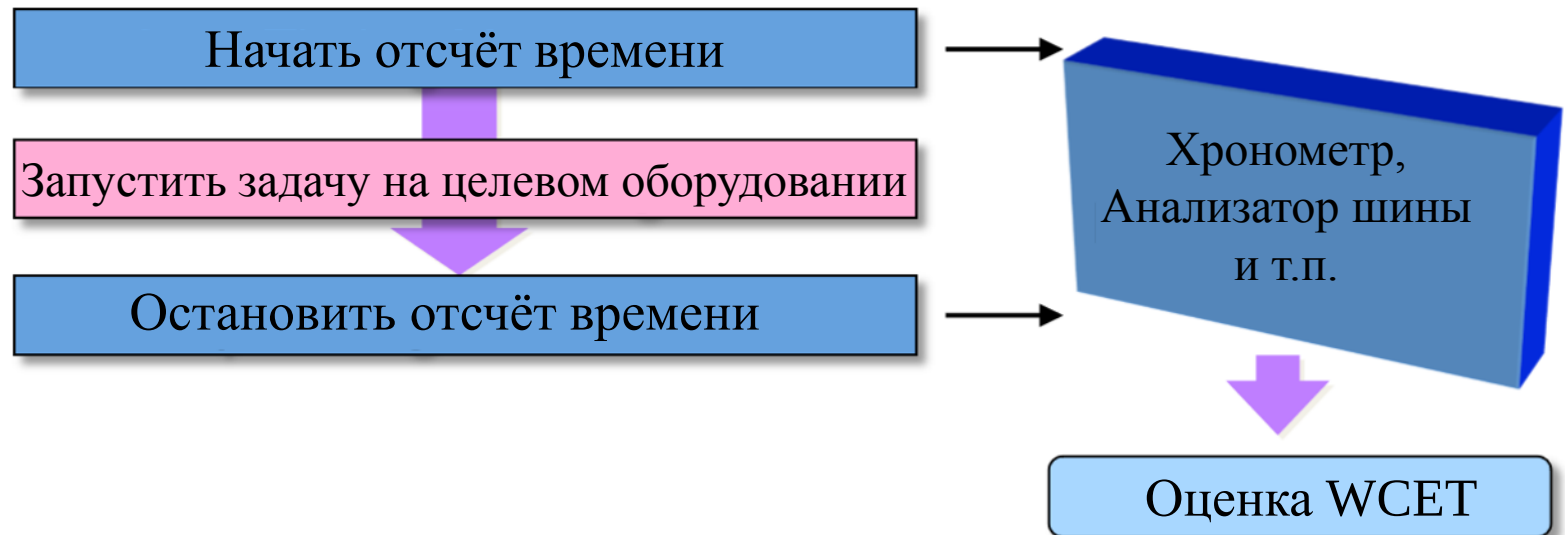
WCET – worst-case execution time (максимальное время выполнения)

Анализ WCET

Цель анализа WCET: **оценить сверху** время выполнения фрагмента кода

- оценка WCET должна быть **безопасной** (недопустимо ошибаться в меньшую сторону)
- оценка WCET должна быть достаточно **точной** (её завышенность приведет к излишнему резервированию ресурсов системы)
- затраты вычислительных ресурсов на анализ должны быть **разумными**

Измерение WCET



Почему нельзя просто измерить WCET?

- ❑ Замер времени выполнения на **всех** путях выполнения реалистичной программы – на практике **невозможен**
- ❑ При определении тестовой выборки могут быть **упущены редкие сценарии** выполнения (обработка ошибочных ситуаций и т.п.)
- ❑ Выбранные тестовые данные **могут не породить самую длинную** (по времени) **трассу** выполнения
- ❑ Внутреннее **состояние процессора** на момент старта измерений может не быть наихудшим

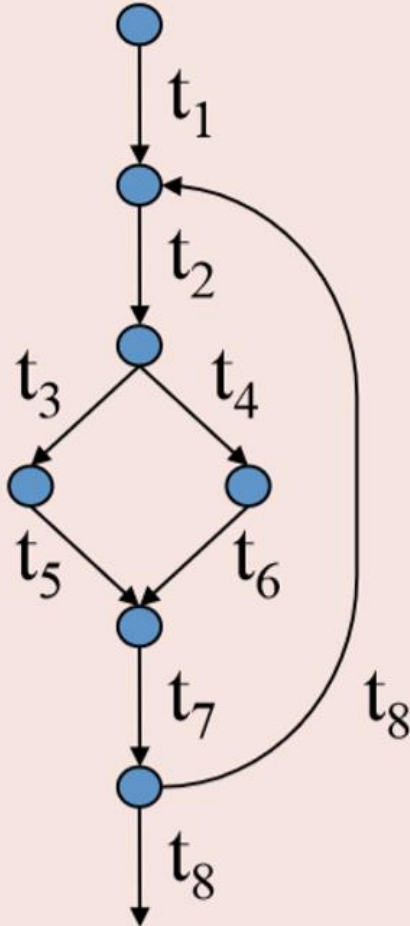
Простые замеры могут послужить источником **первоначальной (грубой)** *нижней* оценки WCET

Статический анализ WCET

- При статическом анализе WCET вычисляется **верхняя оценка** времени выполнения фрагментов кода
- Моделируются **аппаратные** и **программные средства**, а также **контекст** выполнения
 - Программные средства: исходный код, двоичный код (с привязкой к физическим адресам)
 - Аппаратные средства: процессор (в т.ч. конвейер), память (в т.ч. кэш-память)
 - Контекст: начальное состояние аппаратных и программных средств

Чем определяется WCET

Задача

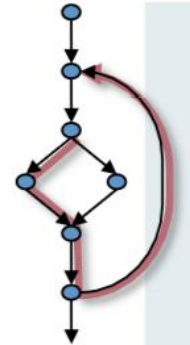


- Возможные **последовательности действий** задачи (пути выполнения)
- **Длительность** выполнения каждого действия на каждом допустимом (т.е. практически возможном) пути выполнения

Чем определяется WCET

Последовательности действия определяются:

- Семантикой программного кода (спецификой реализации, в т.ч. аппаратно-зависимой семантикой)
- Входными данными, возможными в данном контексте вызова программы



Длительность действий определяется:

- Аппаратной реализацией команд процессора
- Состоянием аппаратных средств, влияющих на тайминги (кэш-память, конвейер и т.п.)
 - Факторы, внутренние для задачи
 - Внешние факторы – состояние на момент старта; состояние после вытеснения задачи



Длительность выполнения путей – простой и сложный случаи

Длительность выполнения пути k : $xt(p_k)$

Простейшая архитектура:

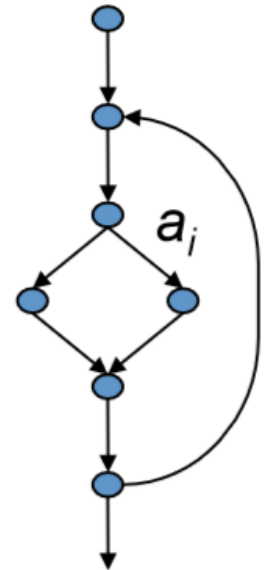
длительность каждого действия a_i – константа:

$$xt(p_k) = \sum n_{k,i} t(a_i)$$

Реалистичная архитектура:

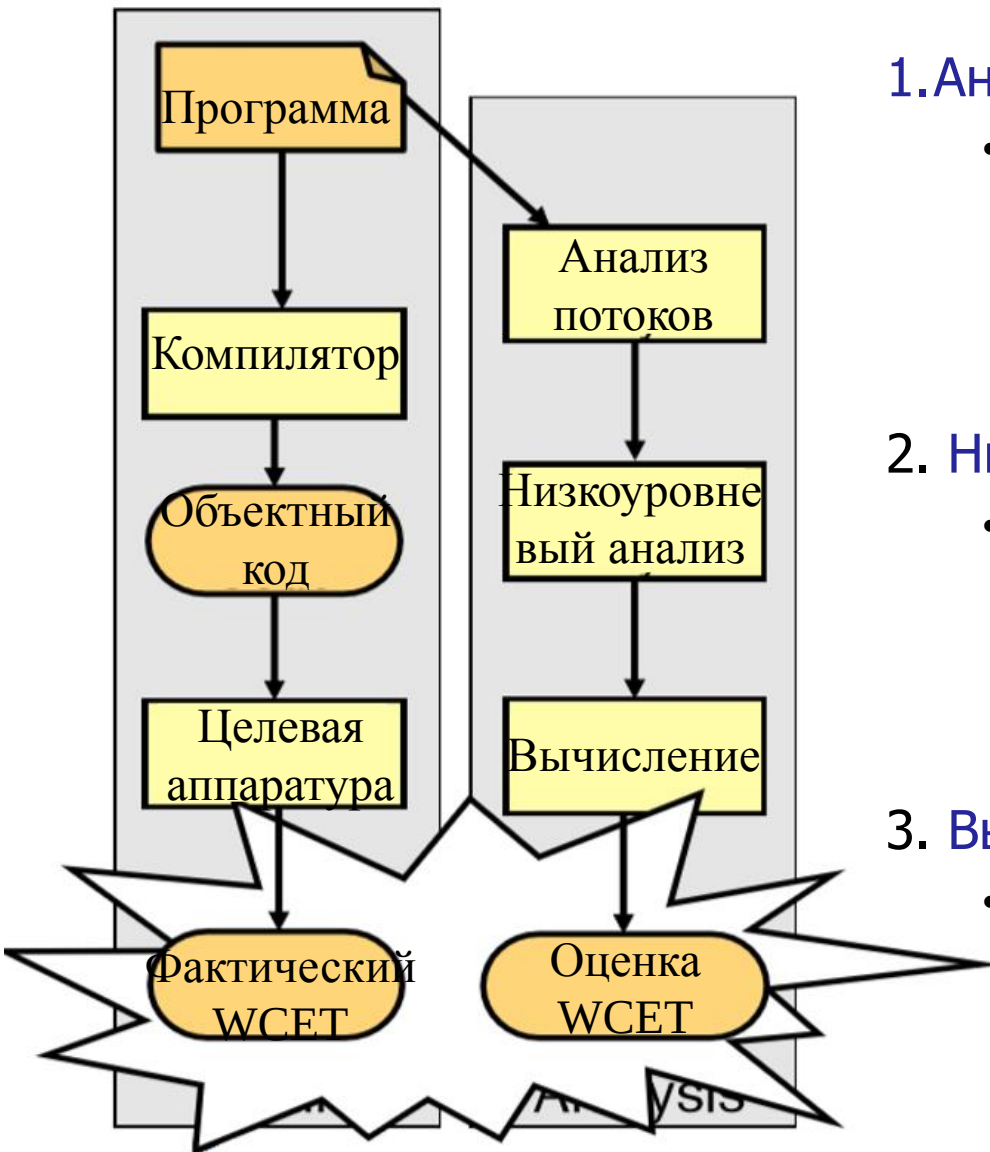
длительности действия варьируются:

$$xt(p_k) = \sum t(a_{i,j(k)})$$



Причины: конвейер, кэш-память, параллелизм в процессоре, ...

Фазы анализа WCET



1. Анализ потоков.

- Ограничить (сверху) число выполнений различных фрагментов программы (в основном, анализ программной составляющей)

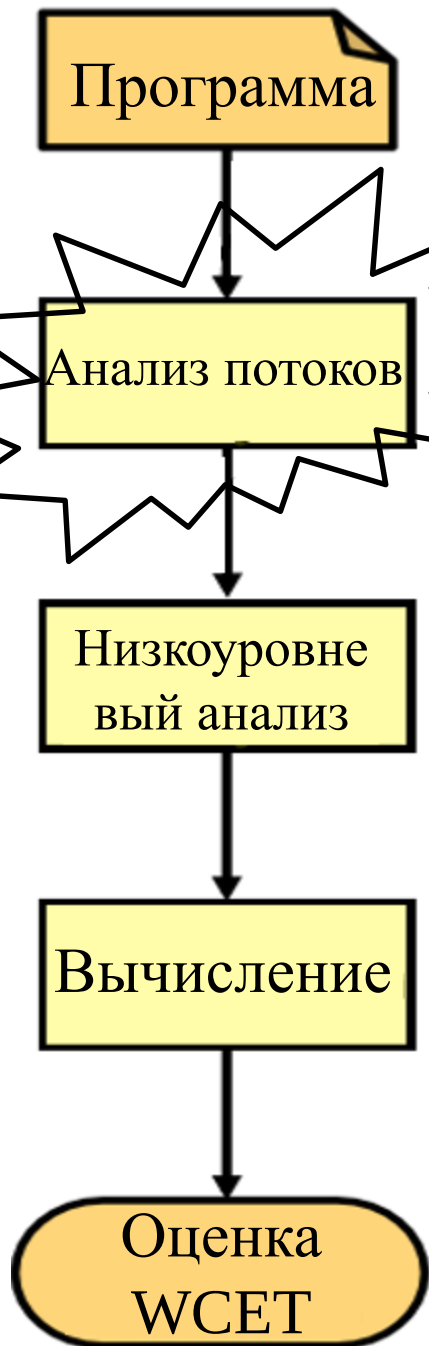
2. Низкоуровневый анализ

- Ограничить (сверху) время выполнения различных фрагментов программы (сочетание анализа программной и аппаратной составляющих)

3. Вычисление

- Объединить результаты анализа потоков и низкоуровневого анализ, чтобы получить верхнюю оценку WCET

Анализ потоков

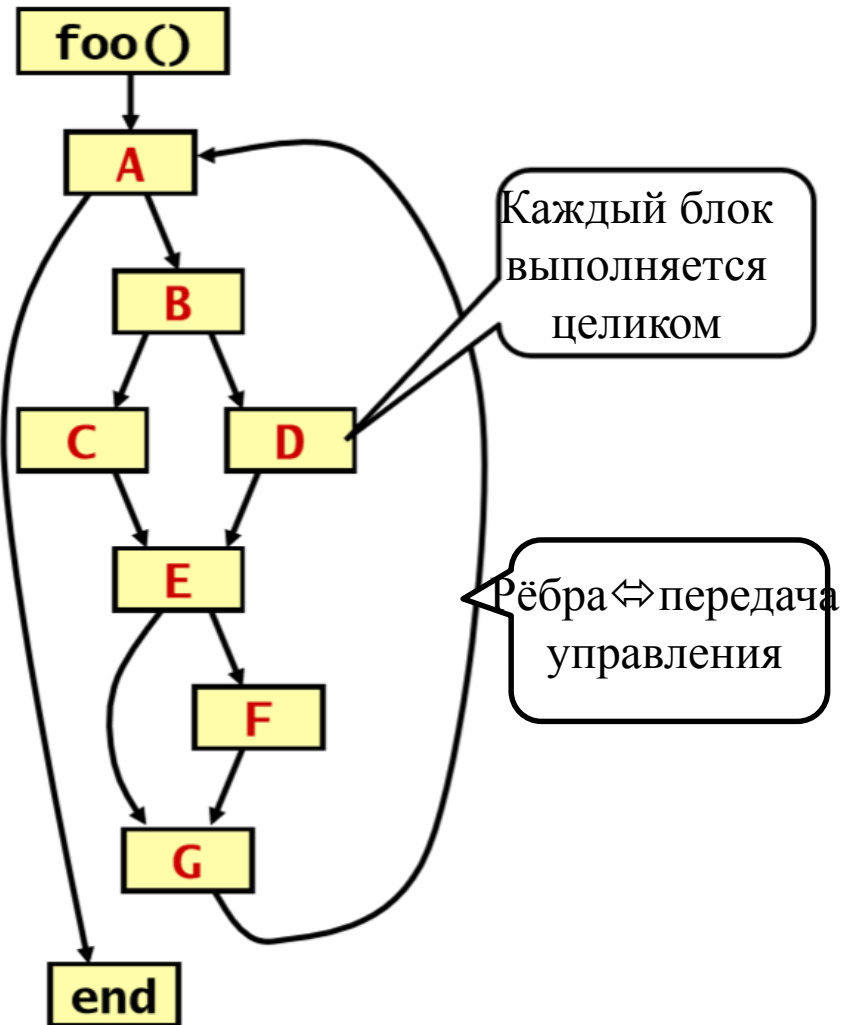
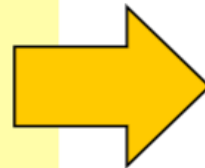


- Ограничивает число выполнений различных фрагментов программы (вычисляет верхнюю оценку)
 - оценка корректна для всех возможных трасс выполнения
- Примеры предоставляемой информации:
 - Ограничение на число итераций цикла
 - Ограничение на глубину рекурсии
 - Недопустимые пути выполнения
- Источники информации:
 - Статический анализ программы
 - Ручные аннотации кода

Граф потока управления

```
foo(x, i):
```

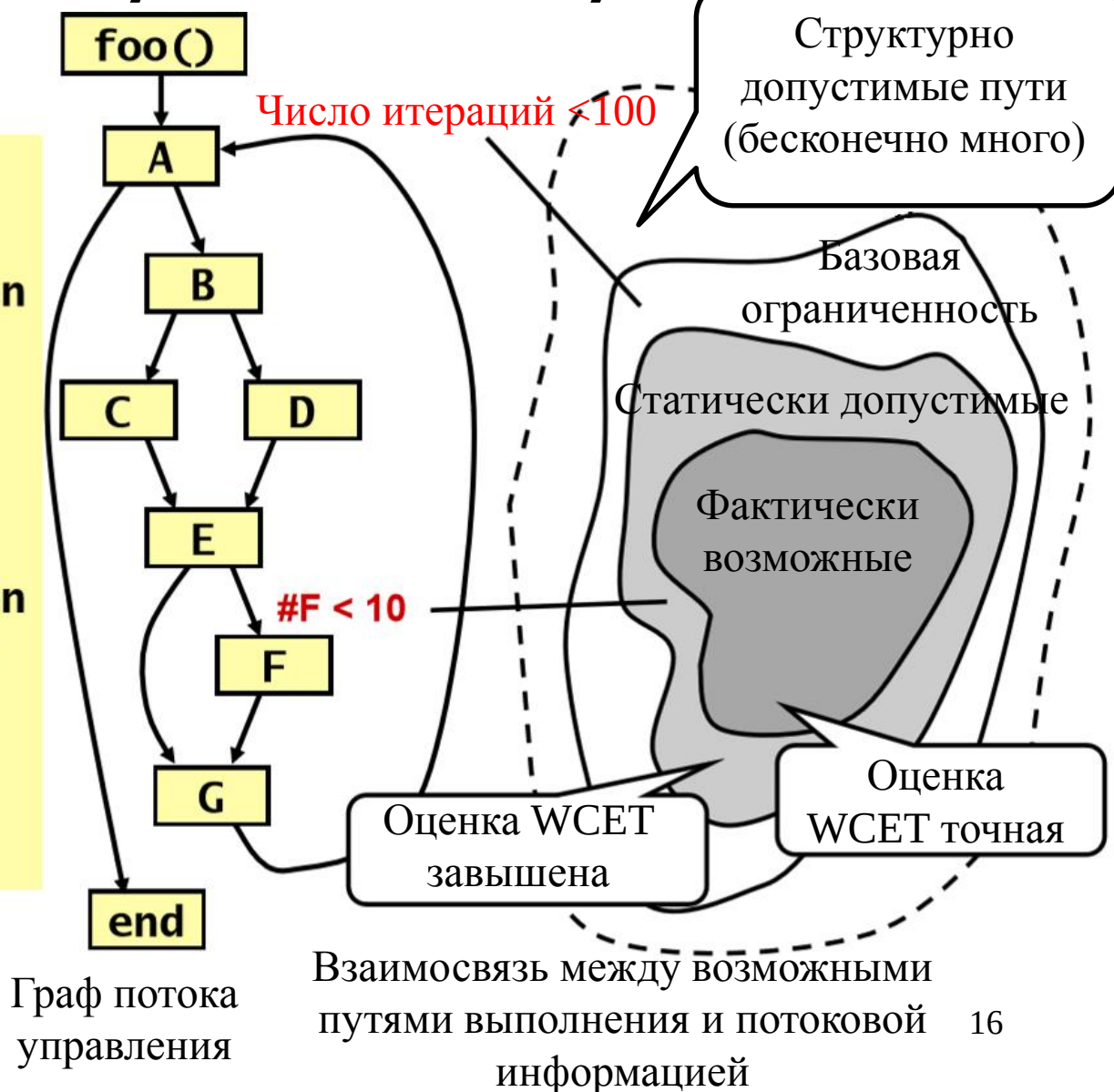
```
A:   while(i < 100)
B:     if (x > 5) then
C:       x = x*2;
           else
D:       x = x+2;
           end
E:     if (x < 0) then
F:       b[i] = a[i];
           end
G:     i = i+1;
           end
end
```



Потоковая информация и недопустимые пути

```
1 ≤ i ≤ 100  
-22 ≤ x ≤ 20  
foo(x, i):  
A: while(i < 100)  
B:   if (x > 5) then  
C:     x = x*2;  
   else  
D:     x = x+2;  
   end  
E:   if (x < 0) then  
F:     b[i] = a[i];  
   end  
G:   i = i+1;  
end
```

Пример программы



Пример: ограничение числа итераций

```
foo(x, i):
```

```
A:   while(i < 100)
B:       if (x > 5) then
C:           x = x*2;
           else
D:           x = x+2;
           end
E:       if (x < 0) then
F:           b[i] = a[i];
           end
G:       i = i+1;
           end
```

Ограничение числа итераций:

- Зависит от возможных значений входной переменной I
 - например, если $1 \leq i \leq 100$, то число итераций не превышает 100
- В общем случае – очень сложная задача
- Имеет решение для многих частных случаев (типов циклов)

Требование базовой ограниченности:

- Для каждого цикла должно быть известно (вычислено или задано) ограничение на число итераций

Пример: недопустимый путь

```
foo(x, i):
```

```
A:   while(i < 100)
B:     if (x > 5) then
C:       x = x*2;
      else
D:       x = x+2;
      end
E:     if (x < 0) then
F:       b[i] = a[i];
      end
G:     i=i+1;
      end
```

Недопустимый путь:

- Путь A-B-C-E-F-G не может быть выполнен, т.к. выполнение C исключает выполнение F
- Если $(x > 5)$, то невозможно, чтобы $(x * 2) < 0$

Недопустимые пути исключаются из множества статически допустимых путей

- Это может уточнить оценку WCET

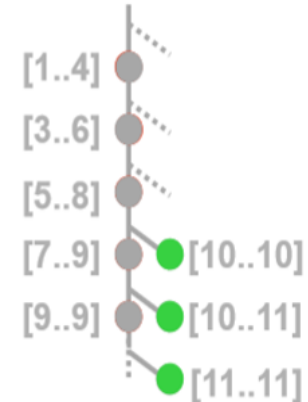
Абстрактная интерпретация

- Ограничивает число итераций циклов и выявляет недопустимые пути
 - Вычисляет безопасную (расширенную) оценку множества значений каждой переменной для различных точек выполнения программы
 - В ходе АИ, переменной сопоставляется не конкретное значение, а множество значений («абстрактное» значение)
- Программа «выполняется» с использованием абстрактных значений переменных
- Результат выполнения: безопасная (расширенная) оценка множества допустимых путей выполнения
 - Все фактически допустимые пути входят в полученное множество
 - Также в него могут входить некоторые фактически НЕдопустимые пути
 - Пути, не вошедшие в полученное множество, гарантированно не допустимы

Ограничение числа итераций цикла с помощью АИ

```
i = INPUT;  
// i = [1..4]  
while (i < 10) {  
    // точка p  
    ...  
    i = i + 2;  
}  
// точка q
```

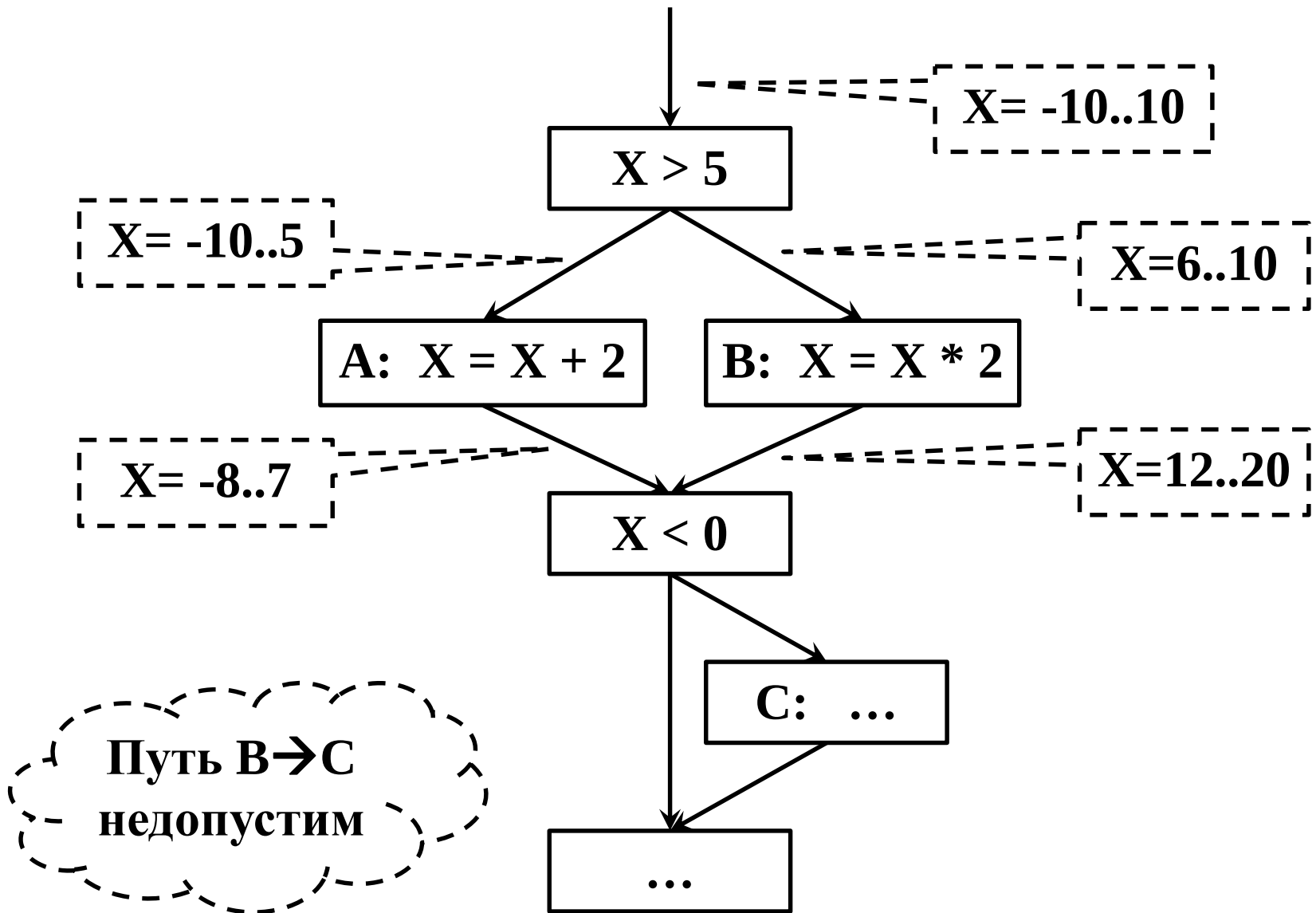
Итерация цикла	Абстрактное состояние в p	Абстрактное состояние в q
1	i = [1..4]	$\perp$
2	i = [3..6]	$\perp$
3	i = [5..8]	$\perp$
4	i = [7..9]	i = [10..10]
5	i = [9..9]	i = [10..11]
6	$\perp$	i = [11..11]



Результат:
Мин. число итераций: 3
Макс. число итераций: 5

- Анализируются все возможные пути выполнения цикла
- В точке q система может находиться в одном из трёх абстрактных состояний
 - Они могут быть объединены в одно абстрактное состояние

Выявление недопустимого пути с помощью АИ



Треугольный цикл

```
triangle(a,b):  
A:  loop(i=1..100)  
B:    loop(j=i..100)  
C:      a[i,j]=...  
      end loop  
    end loop
```

- Два вложенных цикла
 - Итераций в цикле A: не более 100
 - Итераций в цикле B: не более 100
- Число выполнений блока C:
 - Исходя из ограничений на число итераций циклов:
 $100 * 100 = 10\ 000$
 - Фактически: $100 + \dots + 1 = 5\ 050$

=> Аннотации кода

Виды аннотаций кода

Простейшая архитектура

- Сведения о **частоте** выполнения действий
 - Границы и соотношения для **частот** выполнения
 - Нотация: **метки** (marker), **соотношения** (relation), **области** (scope)

Сложная (реалистичная) архитектура

- Сведения о частоте выполнения **последовательностей** действий
 - Информация о (не)допустимых **путях**
 - Нотация: на основе **регулярных выражений**, например **IDL** (path Information Description Language)

Пример аннотации путей

```
for (i=0; i<N; i++)  
{  
  if (i % 3 == 0)  
  {  
    M1  
  }  
  if (i % 3 != 0)  
  {  
    M2  
  }  
}
```

Выражение для пути
Сложная архитектура:

$N = 3 \cdot k$

$(M1.M2.M2)^{\lceil N/3 \rceil} +$

$N = 3 \cdot k + 1$

$(M1.M2.M2)^{\lceil N/3 \rceil} . M1 +$

$N = 3 \cdot k + 2$

$(M1.M2.M2)^{\lceil N/3 \rceil} . M1.M2$

Простая архитектура:

$f(M1) = \lceil N/3 \rceil,$

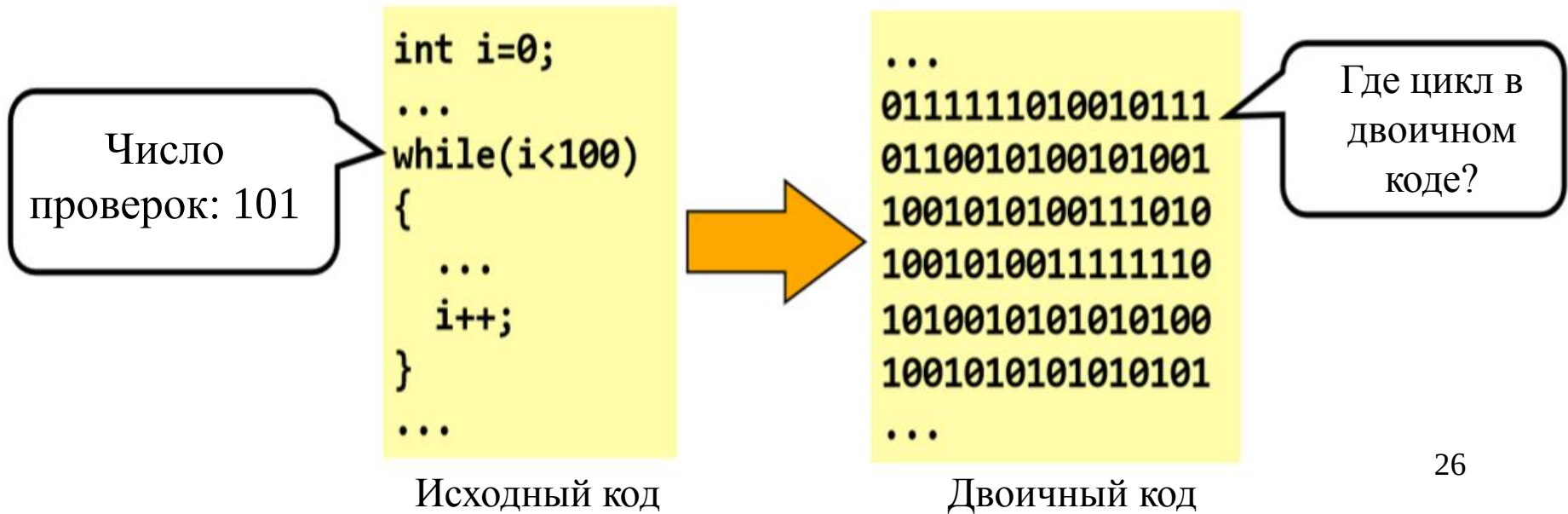
$f(M1) + f(M2) = N$

Маркеры, соотношения и области

```
SCOPE
{
  for (i=0; i<N; i++)
  {
    MAX_ITERATIONS(N);
    for (j=0; j<i; j++)
    {
      MAX_ITERATIONS(N);
      MARKER(M1);
      ...
    }
  }
  REL(FREQ(M1) == N * (N+1) / 2);
}
```

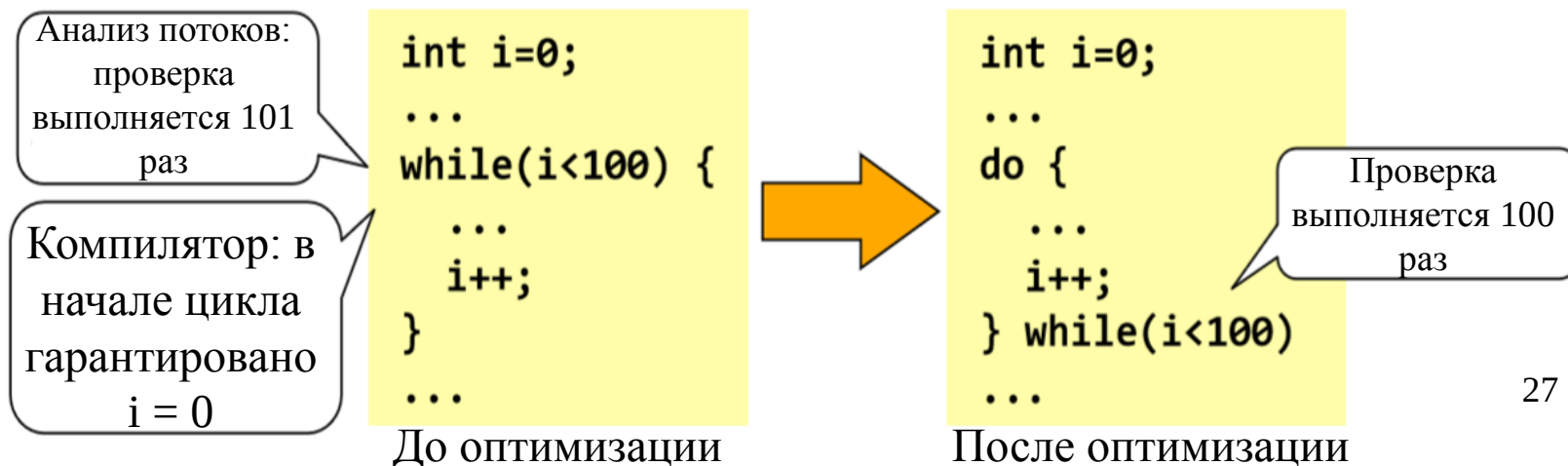

Проблема соответствия между ИСХОДНЫМ И ДВОИЧНЫМ КОДОМ

- Анализ потоков проще проводить на уровне исходного кода
 - Более ясная семантика кода
 - Проще получить потоковую информацию (и для программиста, и для автоматических инструментов)
- Низкоуровневый анализ работает с двоичным кодом, фактически выполняемым на процессоре
- Как отобразить потоковую информацию с уровня исходного кода на двоичный код?



Проблема соответствия между исходным и ДВОИЧНЫМ КОДОМ

- Компиляторы для встроенных систем реального времени выполняют глубокую оптимизацию кода
 - Нужно уложиться в ограничения по времени и объему памяти
- Оптимизации могут существенно изменить размещение кода и данных
 - В результате оптимизаций потоковая информация с уровня исходного кода становится неприменимой
- Решения:
 - Реализовать в компиляторе средства отображения потоковой информации (недостижимый идеал)
 - Использовать компиляцию с отладочной информацией (работает только при отсутствии или с минимумом оптимизаций)
 - Для систем с большим объемом памяти – не производить оптимизации кода с целью сокращения объема
 - Проводить потоковый анализ на двоичном коде (так чаще всего и делают)



Низкоуровневый анализ



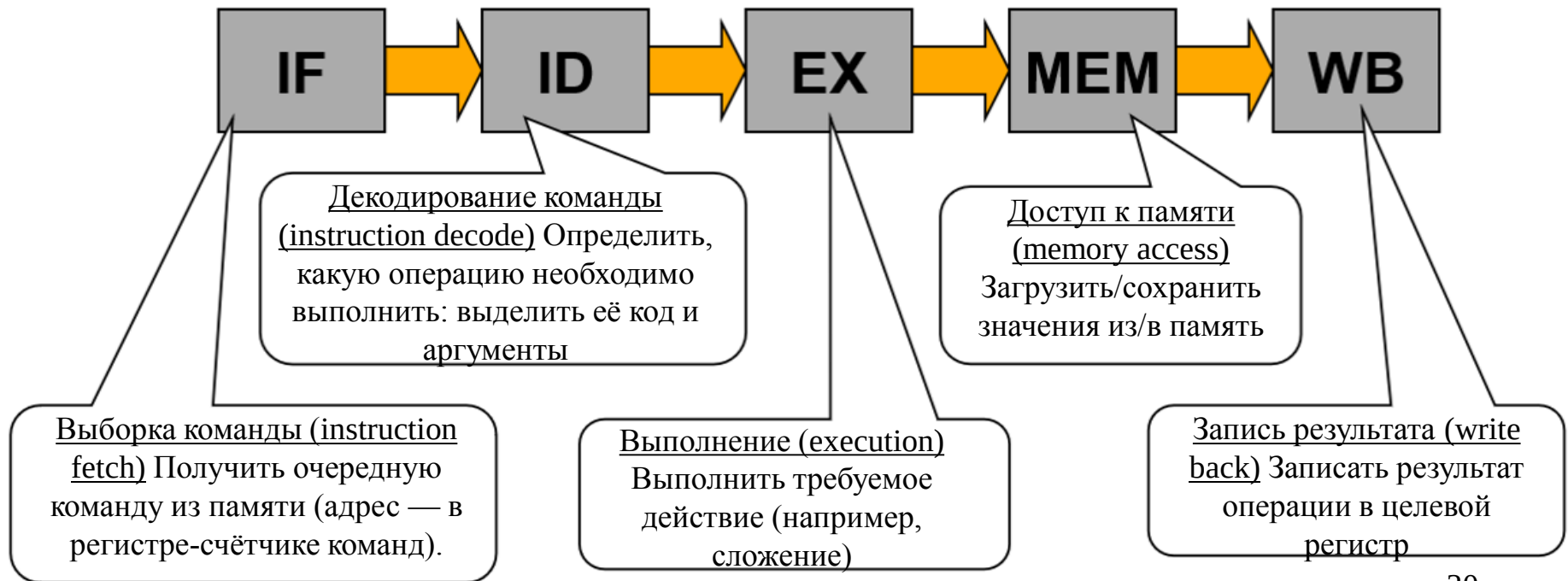
- ❖ Ограничить время выполнения различных фрагментов программы
 - ❑ Основная задача большей части исследований по WCET
- ❖ Использовать модель целевой аппаратуры
 - ❑ Не требуется моделировать все подробности работы аппаратуры
 - ❑ При этом необходимо безопасно (сверху) оценить все задержки при работе аппаратуры
- ❖ Применяется к скомпонованному двоичному коду (исполняемой программе)

Проблемы моделирования аппаратуры

- ❖ Высокая сложность моделирования внутренней работы процессора
 - Конвейер, предсказание ветвлений, суперскалярность, внеочередное выполнение...
 - ❖ Высокая сложность моделирования иерархической памяти
 - Необходимо детальное моделирование кэш-памяти
 - Другие виды памяти также являются источником задержек
 - ❖ Многие аспекты функционирования сложных процессоров должны моделироваться совместно
 - Тайминги выполнения команд процессора зависят от предыстории
 - ❖ Разработка безопасной временной модели функционирования процессора – сложная задача
 - Занимает месяцы и даже годы работы
 - Необходимо учесть все факторы, влияющие на время выполнения (как минимум, оценить сверху их влияние)
- => Аппаратура с предсказуемым временем работы важнее, чем очень быстрая аппаратура

Простой конвейер

- Большинство команд проходят в процессоре через одни и те же этапы выполнения
- Пример: классический 5-ступенчатый конвейер RISC-процессора



Работа конвейера

- Параллельная работа различных стадий конвейера для различных команд (=> ускорение)
- Нет конвейера:
 - следующая команда не может начать выполнение, пока текущая команда не завершит последнюю стадию выполнения
- Конвейер:
 - В идеале: коэффициент ускорения равен числу ступеней конвейера
 - Фактически: между командами есть зависимости по данным; «слом» конвейера при неверном предсказании ветвления; ожидание данных из памяти

	1	2	3	4	5	6	7	8	9	10
IF										
ID										
EX										
MEM										
WB										

	1	2	3	4	5	6
IF						
ID						
EX						
MEM						
WB						

Непосредственная
зависимость по
данным

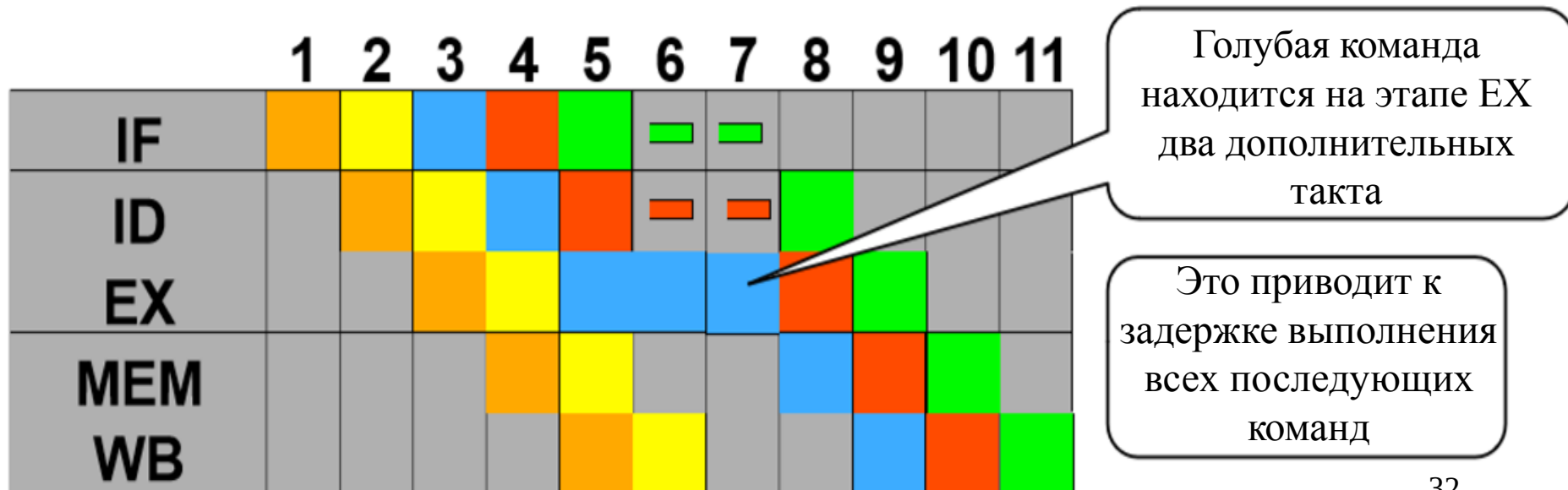
I1. add \$r0, \$r1, \$r2
I2. sub \$r3, \$r0, \$r4

I2 ждёт
вычисления
результата I1

Может возникнуть
задержка в
конвейере

Виды конвейеров

- **Отсутствует:** простейшие процессоры (68HC11, 8051, ...)
- **Скалярный:** единственный конвейер (ARM7, ARM9, V850, ...)
- **VLIW:** несколько конвейеров, статическое планирование их загрузки на уровне компилятора (DSPs, Itanium, Crusoe, ...)
- **Суперскалярный:** несколько конвейеров, внеочередное выполнение команд (PowerPC 7xx, Pentium, UltraSPARC, ...)



Задержки: нет конвейера

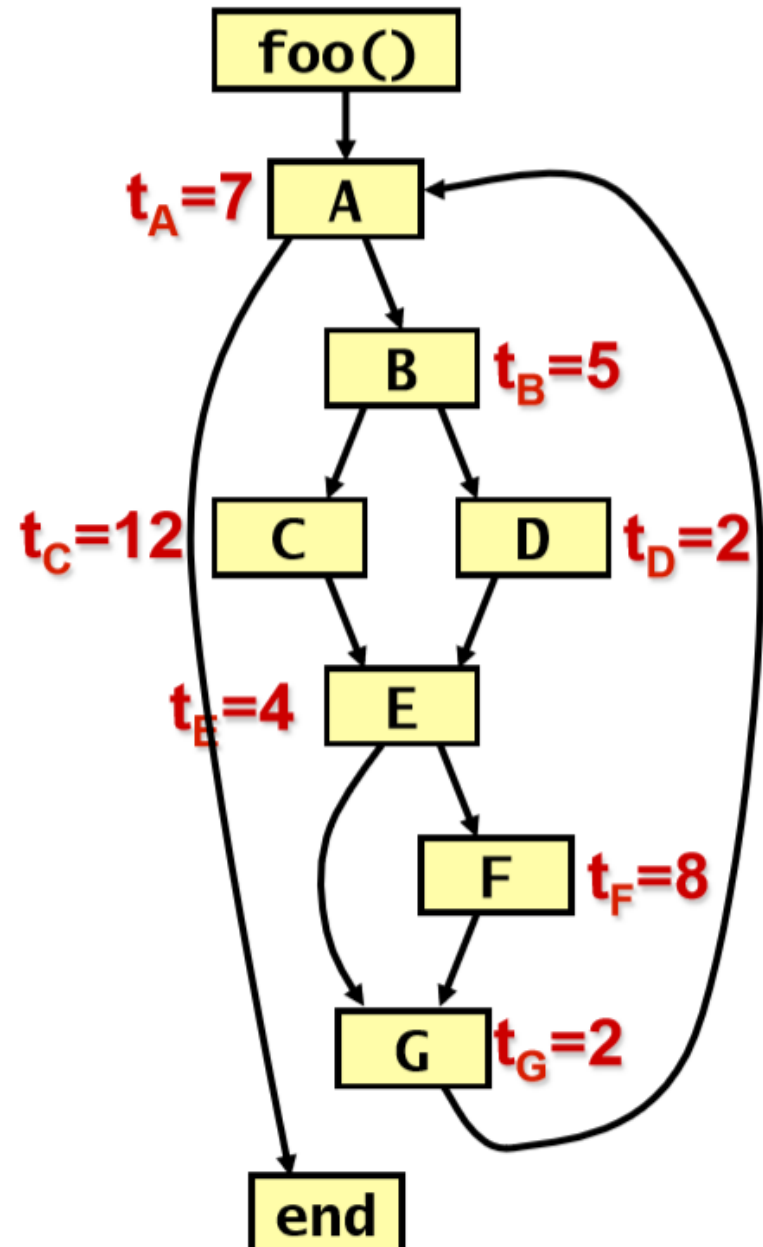
```
foo(x, i):  
A:   while(i < 100)           (7 cycles)  
B:     if (x > 5) then        (5 c)  
C:       x = x*2;             (12 c)  
       else  
D:       x = x+2;             (2 c)  
       end  
E:     if (x < 0) then        (4 c)  
F:       b[i] = a[i];         (8 c)  
       end  
G:     i = i+1;               (2 c)  
      end
```

- Фиксированное время для каждого фрагмента кода
- Двоичный код не показан

Задержки: нет конвейера

```
foo(x, i):
```

```
A:   while(i < 100)
B:     if (x > 5) then
C:       x = x*2;
      else
D:       x = x+2;
      end
E:     if (x < 0) then
F:       b[i] = a[i];
      end
G:     i = i+1;
      end
```



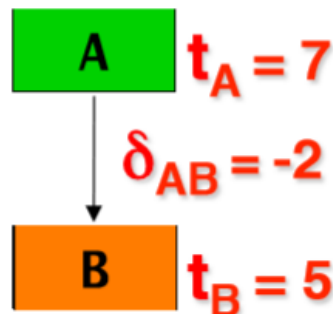
Задержки: простой конвейер

```
foo(x, i):
```

```

A:   while(i < 100)
B:     if (x > 5) then
C:       x = x*2;
           else
D:       x = x+2;
           end
E:     if (x < 0) then
F:       b[i] = a[i];
           end
G:     i = i+1;
           end

```



	1	2	3	4	5	6	7
IF							
EX							
M							
F							

	1	2	3	4	5
IF					
EX					
M					
F					

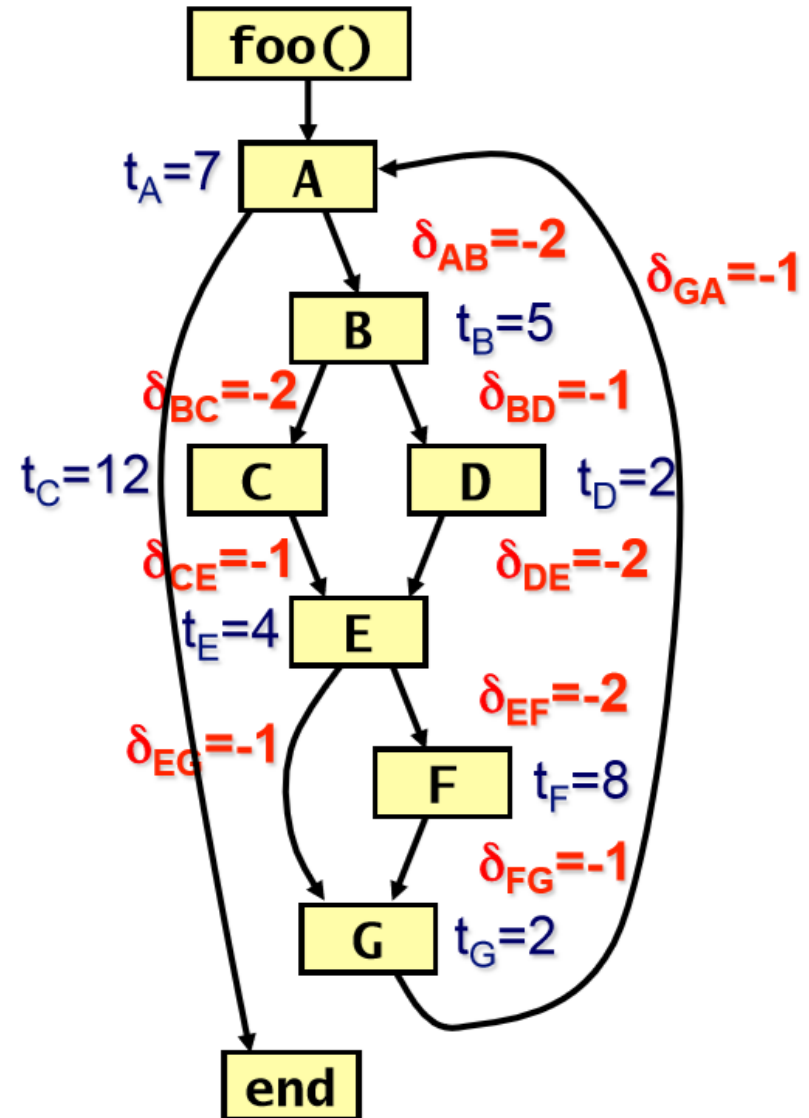
	1	2	3	4	5	6	7	8	9	10
IF										
EX										
M										
F										

$t_{AB} = 10$

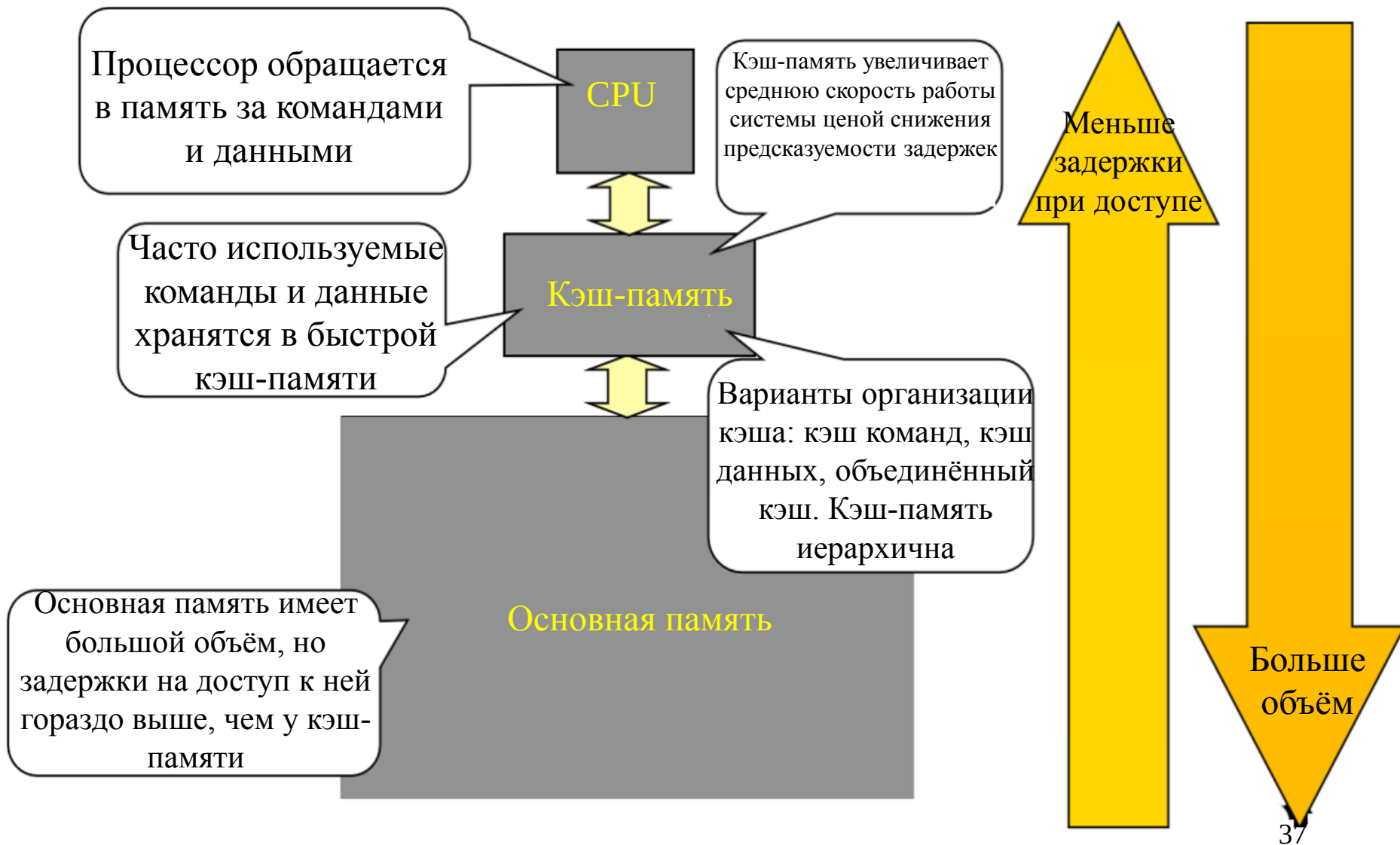
$$\delta_{AB} = 10 - (7 + 5) = -2$$

Задержки: простой конвейер

```
foo(x, i):  
A:   while(i < 100)  
B:     if (x > 5) then  
C:       x = x*2;  
       else  
D:       x = x+2;  
       end  
E:     if (x < 0) then  
F:       b[i] = a[i];  
       end  
G:     i = i+1;  
       end
```



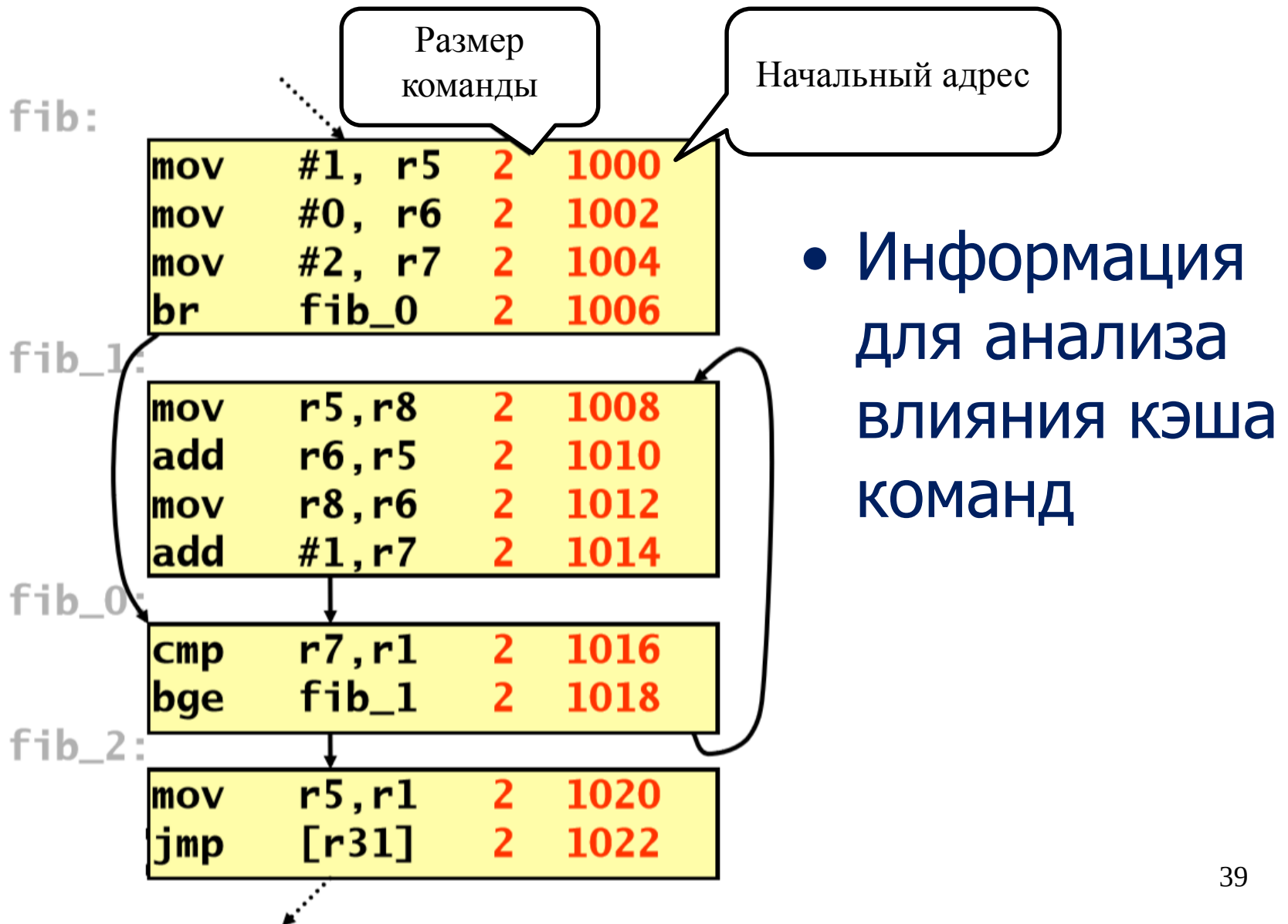
Иерархическая память



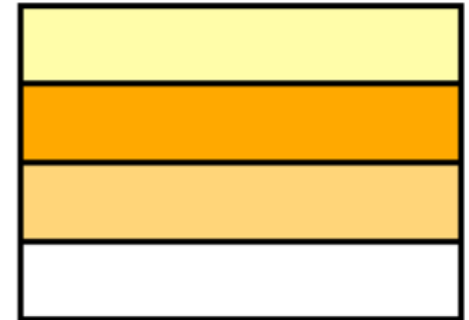
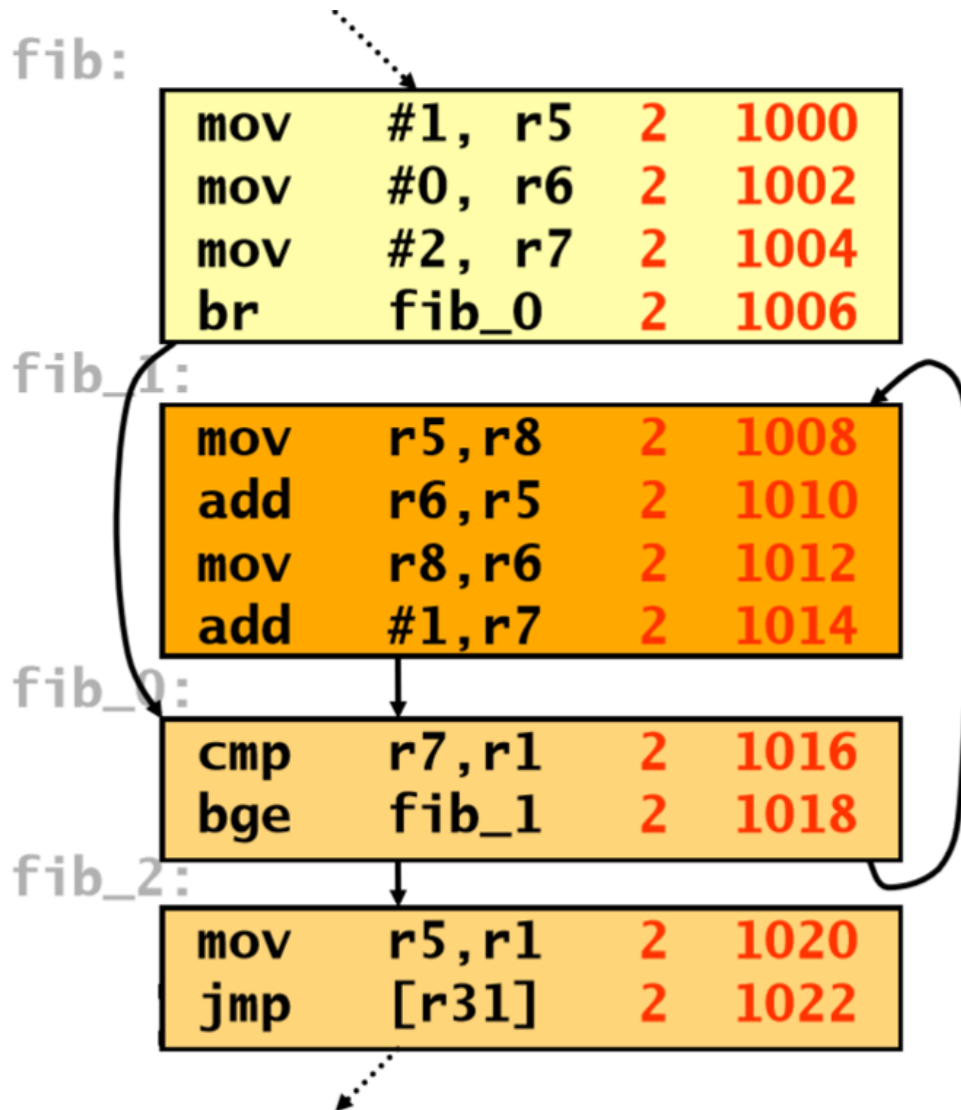
Анализ влияния кэш-памяти



Анализ влияния кэш-памяти



Анализ влияния кэш-памяти



- Отображение в кэш команд

Анализ влияния кэш-памяти

fib:

mov	#1, r5	miss
mov	#0, r6	hit
mov	#2, r7	hit
br	fib_0	hit

fib_1:

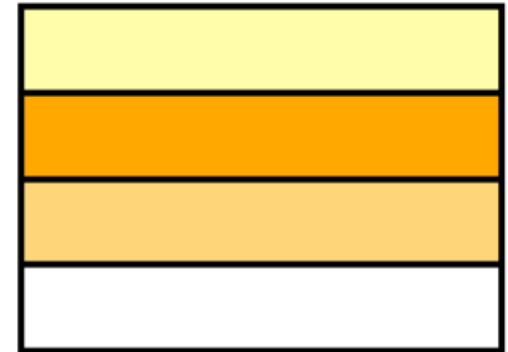
mov	r5, r8	miss
add	r6, r5	hit
mov	r8, r6	hit
add	#1, r7	hit

fib_0:

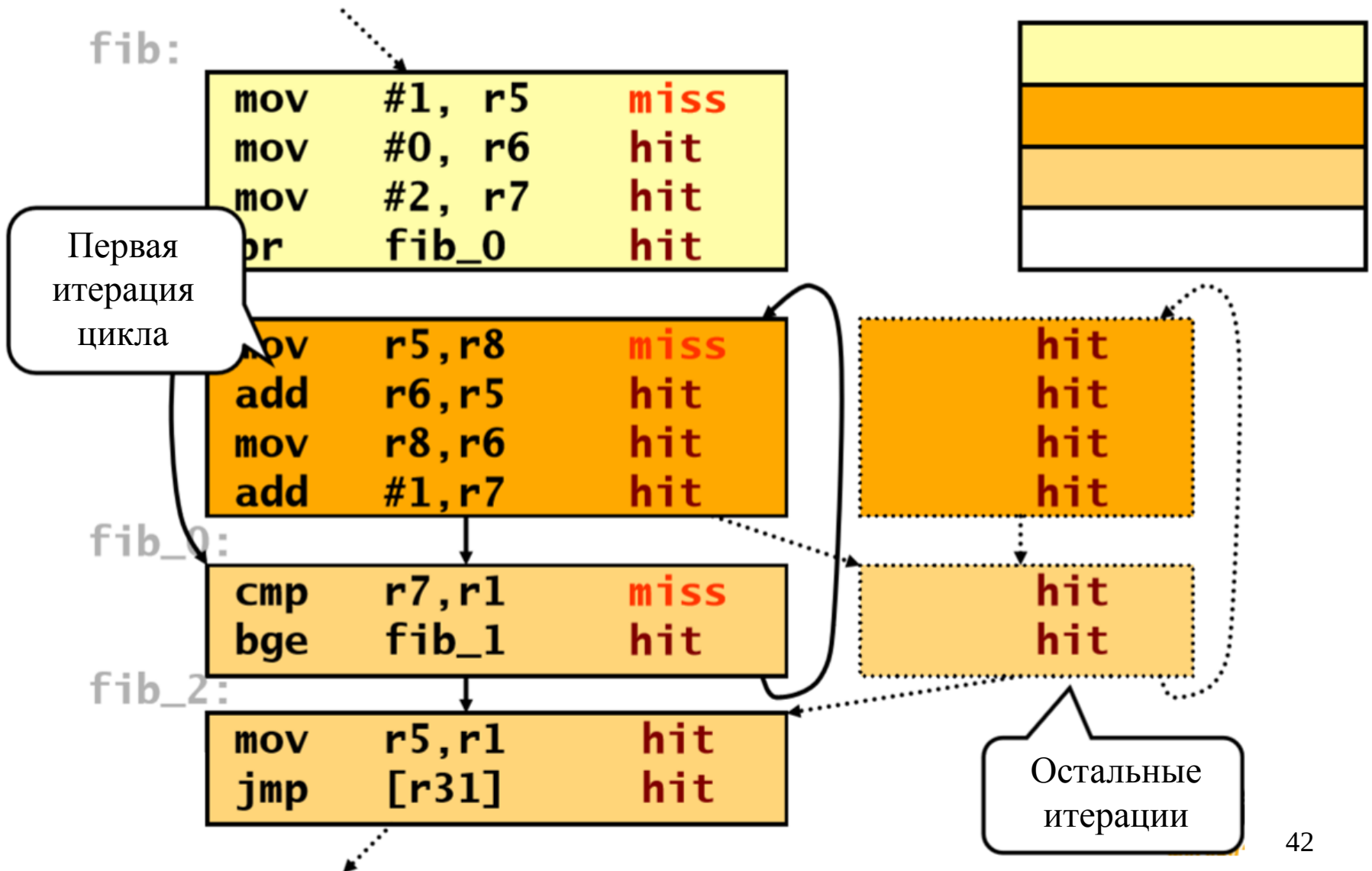
cmp	r7, r1	miss
bge	fib_1	hit

fib_2:

mov	r5, r1	
jmp	[r31]	

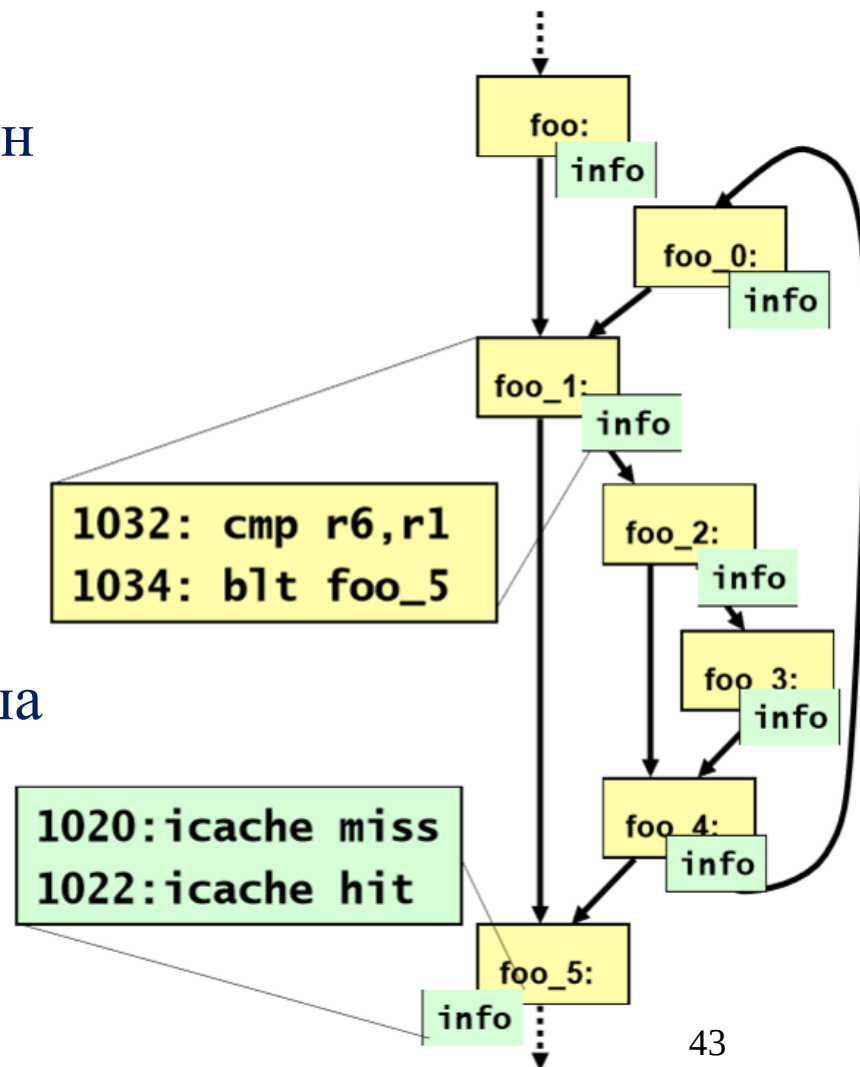


Анализ влияния кэш-памяти



Учет совместного влияния кэша и конвейера

- Анализ влияния конвейера должен брать на вход результаты анализа влияния кэш-памяти
 - Команды помечаются попаданием/промахом в кэш
 - Попадания/промахи влияют на задержки в конвейере
- Сложная аппаратура требует совместного анализа влияния кэша и конвейера



Программа

Вычисление WCET

Анализ потоков

- Вычислить верхнюю оценку WCET программы

- Исходные данные: информация о задержках и потоковая информация

Низкоуровневый анализ

Вычисление

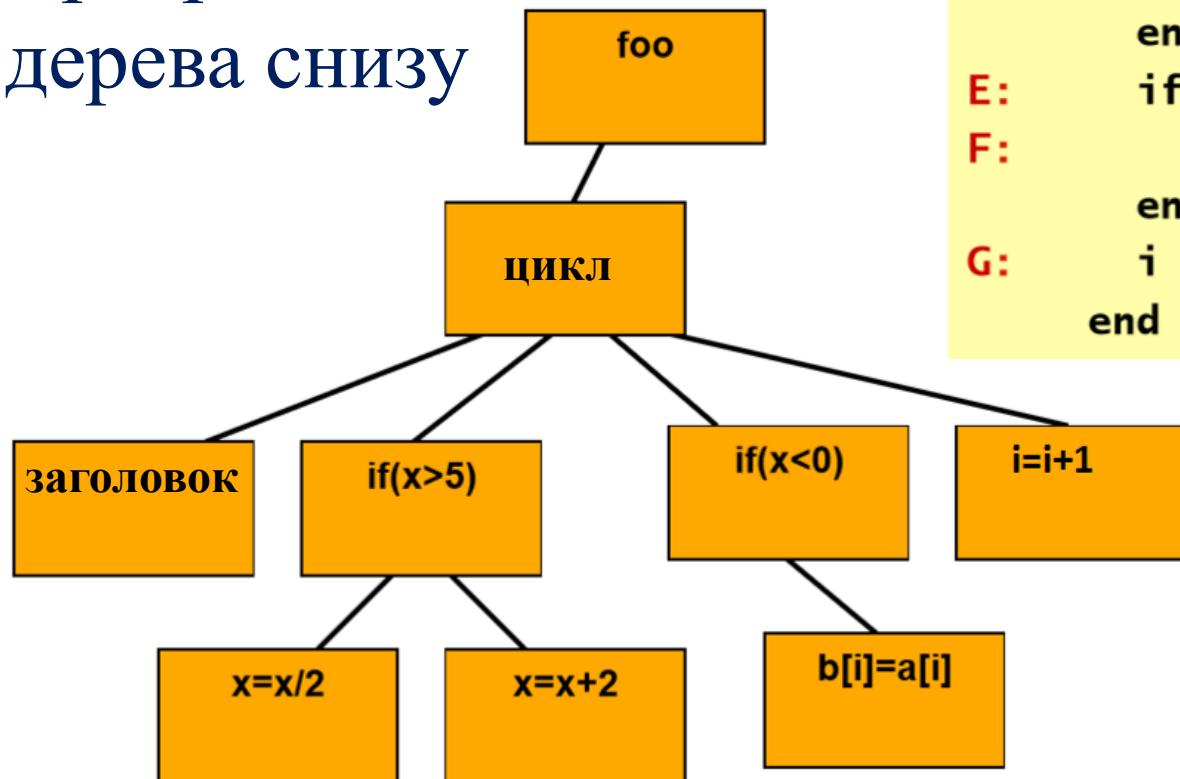
- Примеры подходов:

- Расчёт по синтаксическому дереву
- Расчёт по путям выполнения
- Неявный перебор путей (IPET)

Оценка WCET

Расчёт WCET по дереву

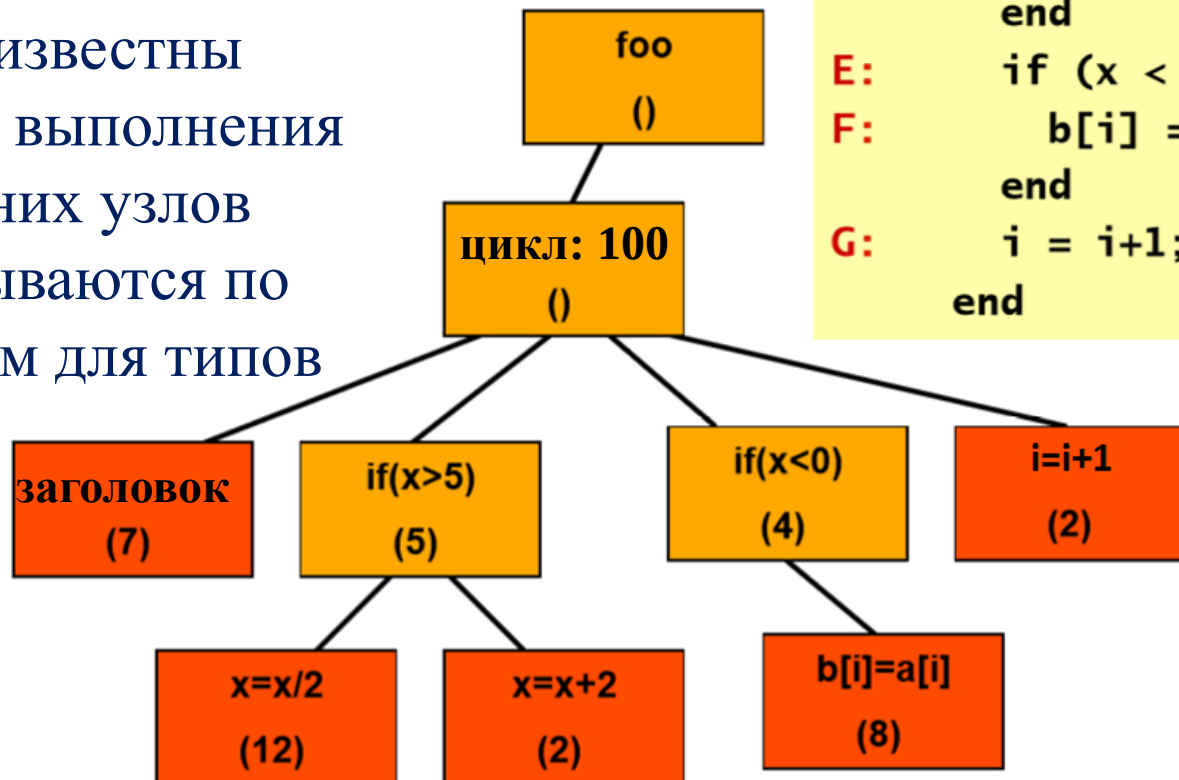
- Анализируется синтаксическое дерево программы
- Обход дерева снизу вверх



```
foo(x,i):  
A:  while(i < 100)  
B:    if (x > 5) then  
C:      x = x/2;  
      else  
D:      x = x+2;  
      end  
E:    if (x < 0) then  
F:      b[i] = a[i];  
      end  
G:    i = i+1;  
      end
```

Расчёт WCET по дереву

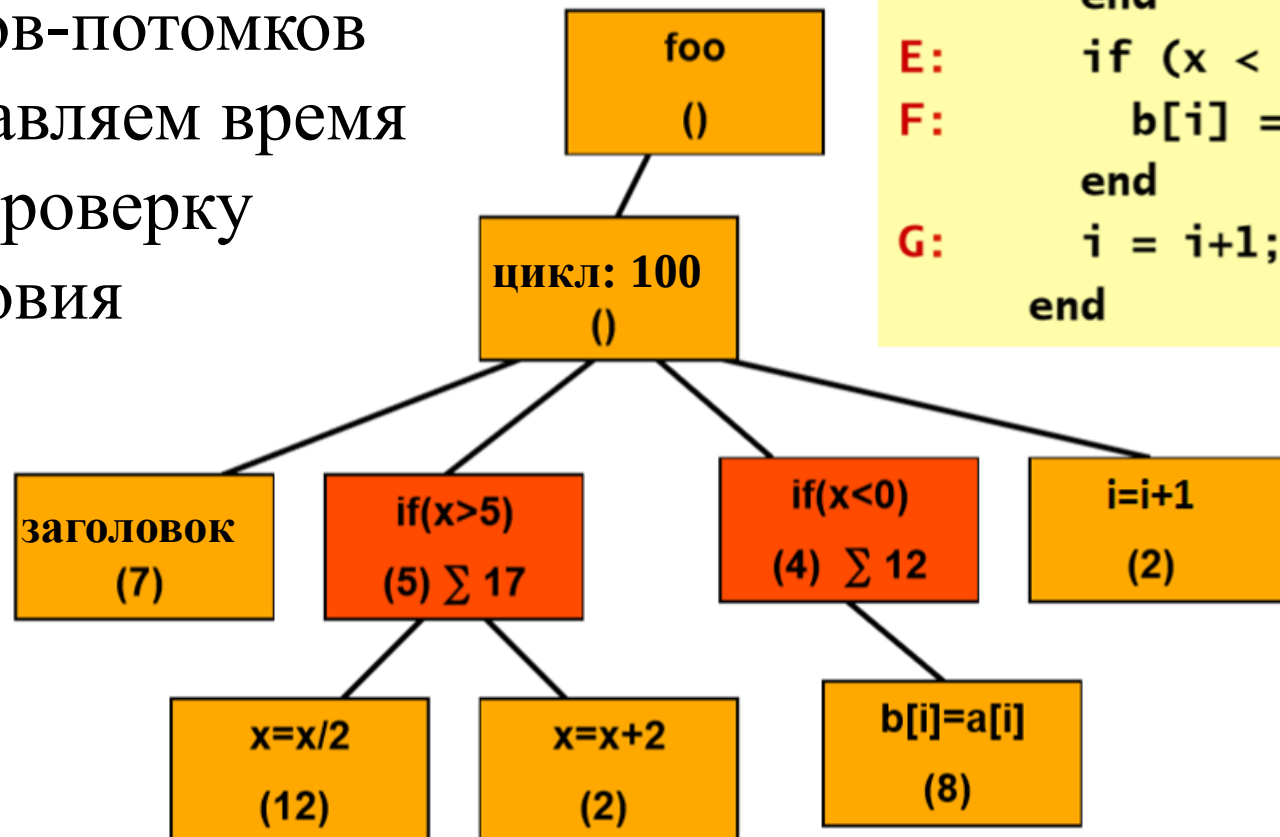
- ❑ Фиксированные времена выполнения узлов
- ❑ Времена выполнения листьев известны
- ❑ Времена выполнения внутренних узлов рассчитываются по формулам для типов узлов



```
foo(x, i):  
A:   while(i < 100) (7 c)  
B:     if (x > 5) then (5 c)  
C:       x = x/2; (12 c)  
       else  
D:       x = x+2; (2 c)  
       end  
E:     if (x < 0) then (4 c)  
F:       b[i] = a[i]; (8 c)  
       end  
G:     i = i+1; (2 c)  
end
```

Правило для оператора ветвления

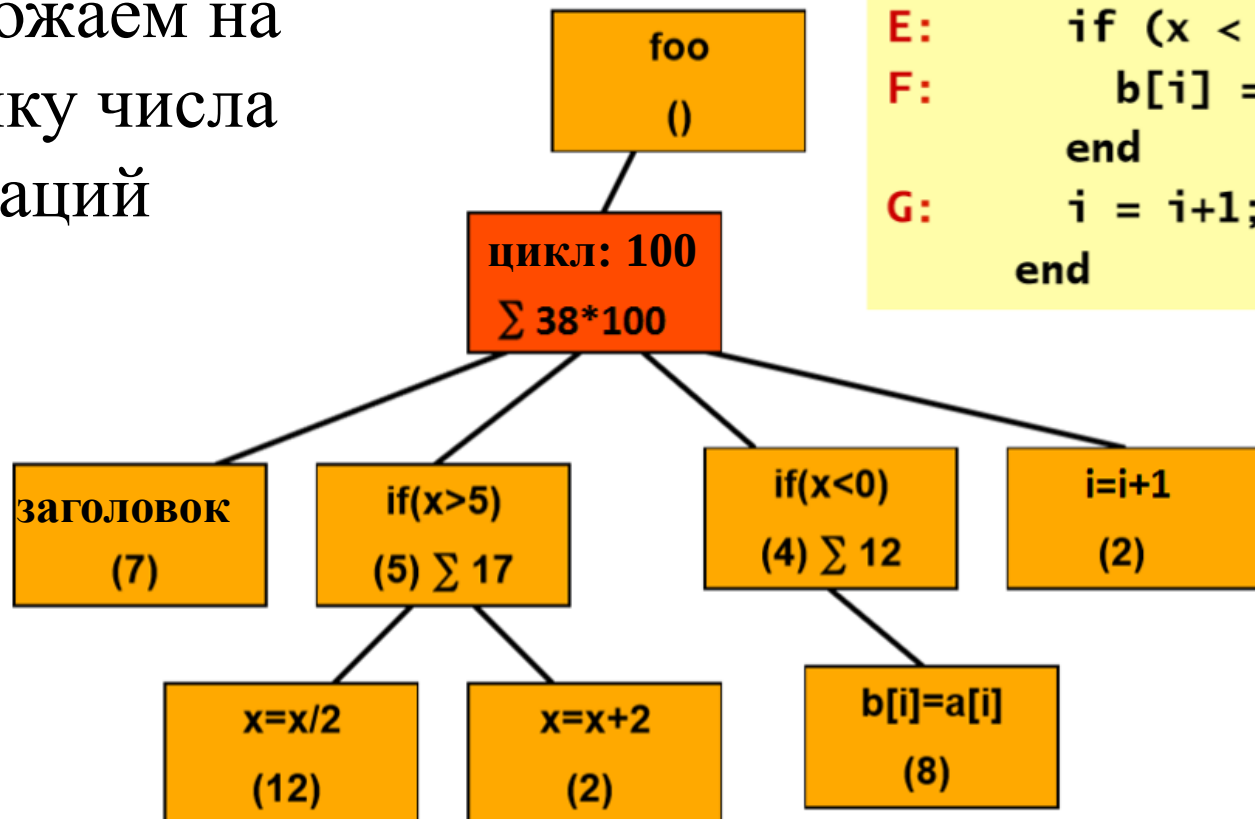
- Ветвление:
 - берем максимум из значений для узлов-потомков
 - добавляем время на проверку условия



```
foo(x,i):  
A:   while(i < 100)  
B:     if (x > 5) then  
C:       x = x*2;  
       else  
D:       x = x+2;  
       end  
E:     if (x < 0) then  
F:       b[i] = a[i];  
       end  
G:     i = i+1;  
       end
```

Правило для оператора цикла

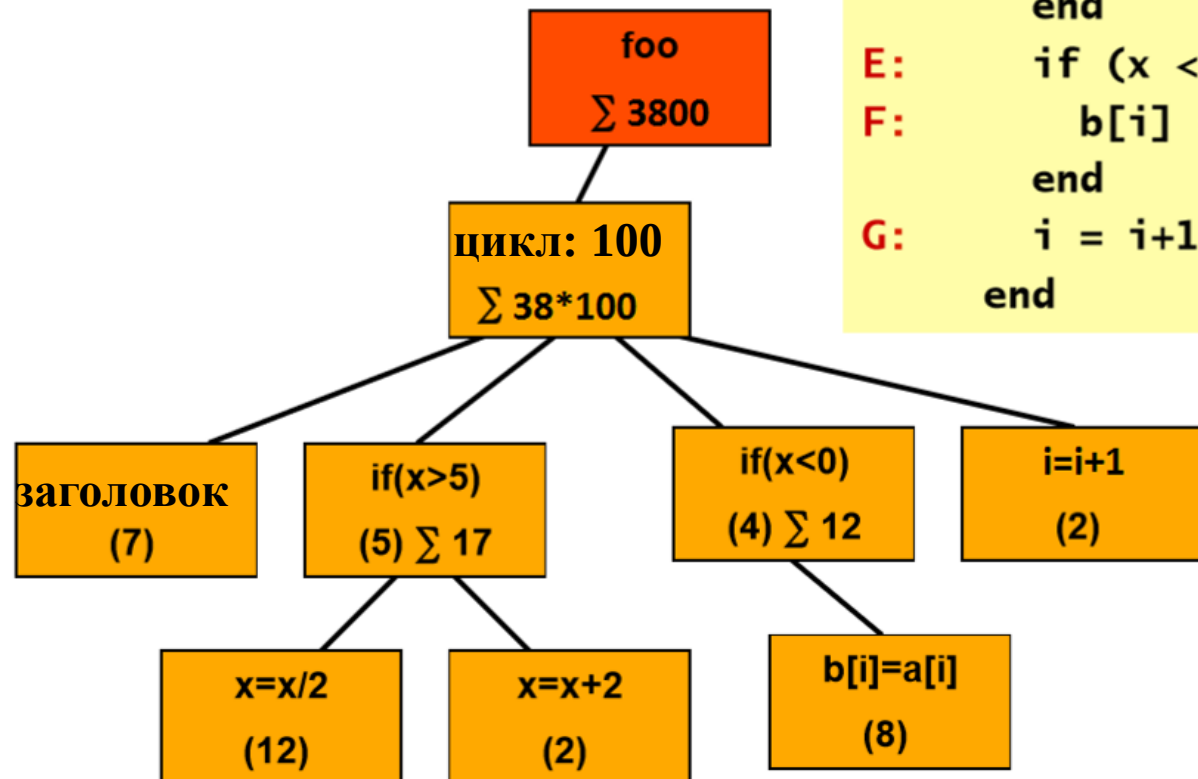
- Цикл:
 - Суммируем оценки для узлов-потомков
 - Умножаем на оценку числа итераций



```
foo(x,i):  
A:  while(i < 100)  
B:    if (x > 5) then  
C:      x = x*2;  
      else  
D:      x = x+2;  
      end  
E:    if (x < 0) then  
F:      b[i] = a[i];  
      end  
G:    i = i+1;  
      end
```

Результат расчёта по дереву

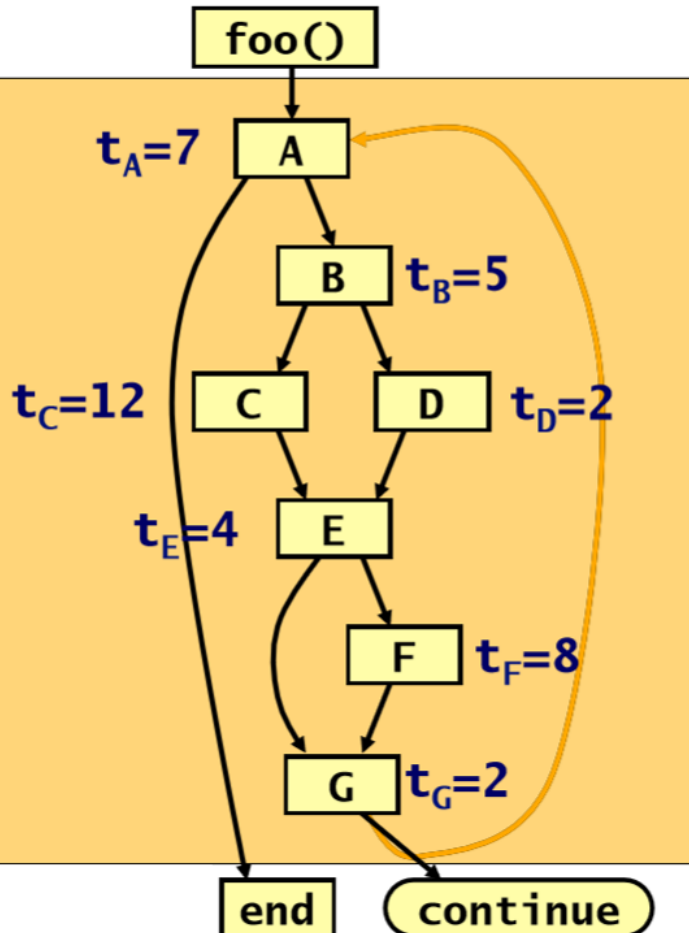
- WCET функции `foo()` равен 3800 тактам



```
foo(x,i):  
A:  while(i < 100)  
B:    if (x > 5) then  
C:      x = x*2;  
      else  
D:      x = x+2;  
      end  
E:    if (x < 0) then  
F:      b[i] = a[i];  
      end  
G:    i = i+1;  
      end
```

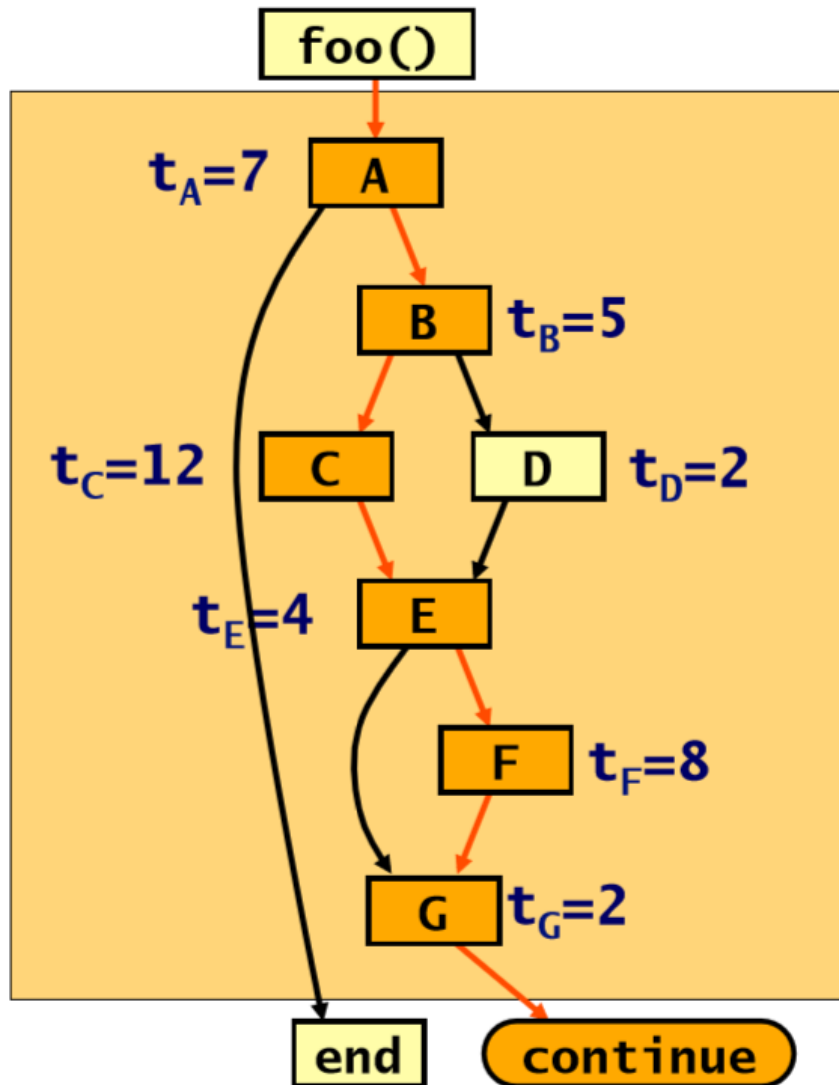

Расчёт WCET по путям

```
foo(x,i):  
A:  while(i < 100)  
B:    if (x > 5) then  
C:      x = x*2;  
      else  
D:        x = x+2;  
      end  
E:    if (x < 0) then  
F:      b[i] = a[i];  
      end  
G:    i = i+1;  
      end
```



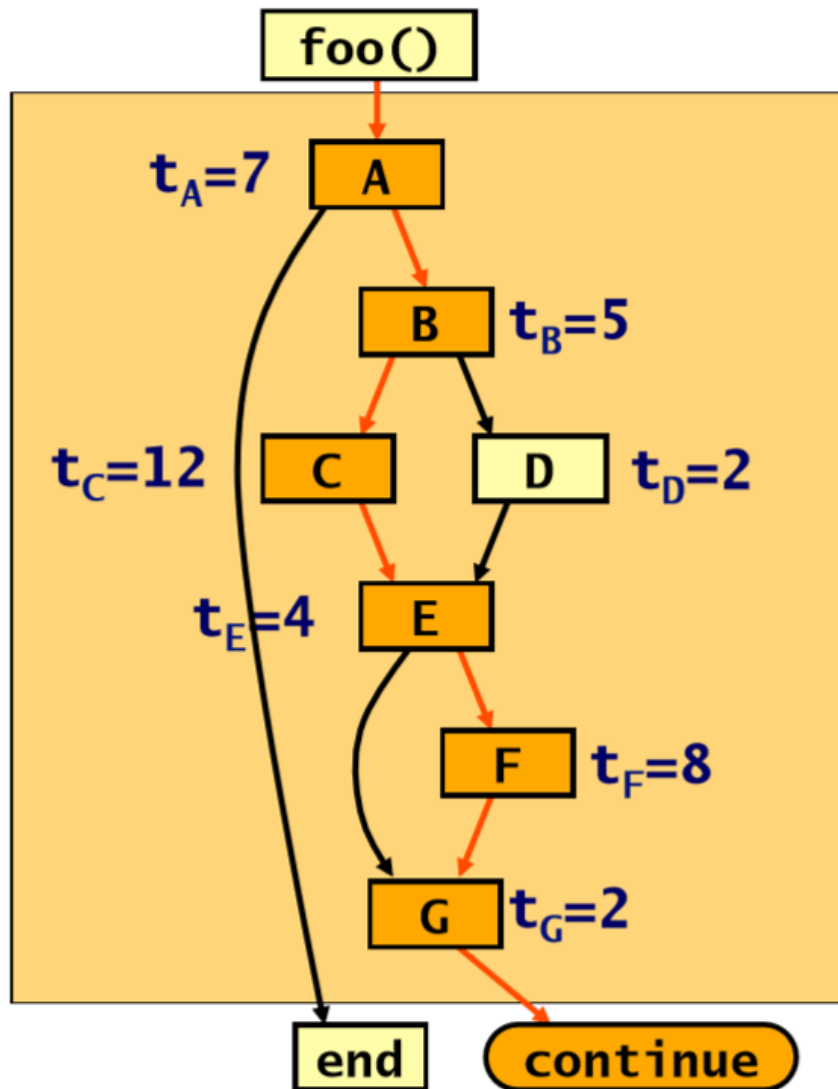
- Найти самый длинный путь
 - Рассматриваем итерации цикла по одной
- Подготовить цикл
 - Убрать обратные дуги
 - Перенаправить их на специальные узлы «continue»

Расчёт WCET по путям



- Самый длинный путь:
 - A-B-C-E-F-G
 - $-7 + 5 + 12 + 4 + 8 + 2 = 38$ тактов
- Суммарное время:
 - 100 итераций
 - 38 тактов на итерацию
 - Итого: 3800 тактов

Расчёт WCET по путям



С и F никогда не
выполняются
совместно

```
foo(x,i):  
A:  while(i < 100)  
B:    if (x > 5) then  
C:      x = x*2;  
      else  
D:        x = x+2;  
      end  
E:    if (x < 0) then  
F:      b[i] = a[i];  
      end  
G:    i = i+1;  
      end
```

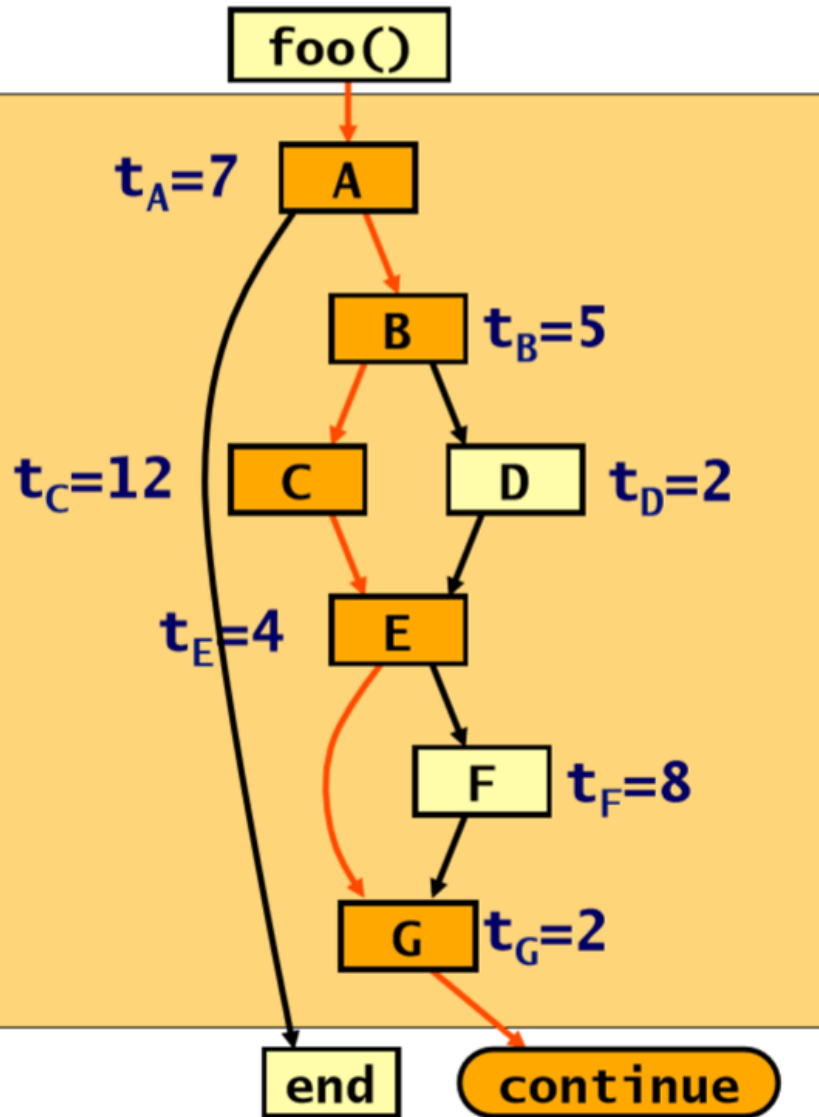
- Недопустимый путь:
 - A-B-C-E-F-G
 - Отбрасываем, ищем следующий по длине

Расчёт WCET по путям

foo(x,i):

```
A: while(i < 100)
B:   if (x > 5) then
C:     x = x*2;
      else
D:       x = x+2;
      end
E:   if (x < 0) then
F:     b[i] = a[i];
      end
G:   i = i+1;
      end
```

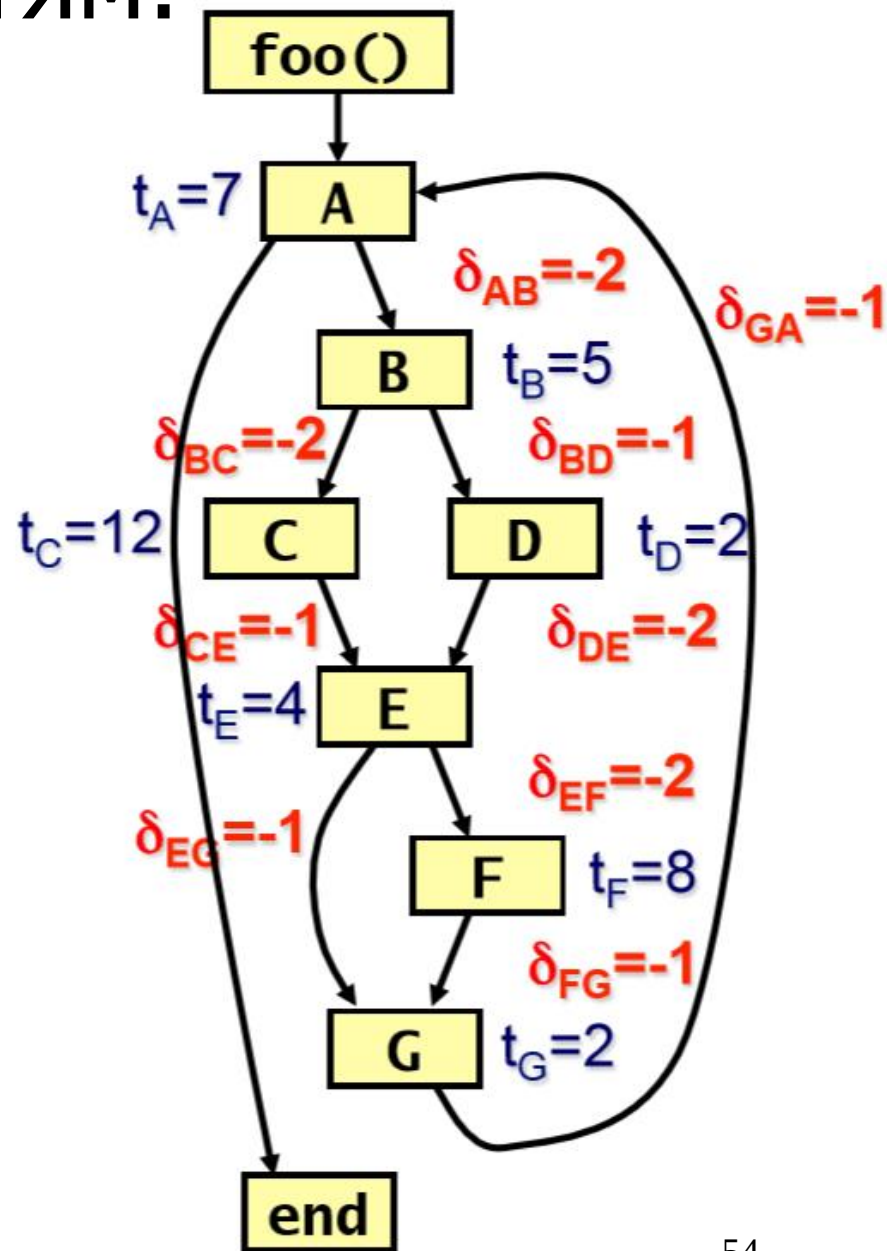
С и F никогда не
выполняются
совместно



- Недопустимый путь:
 - A-B-C-E-F-G
 - Отбрасываем, ищем следующий по длине
- Новый самый длинный путь:
 - A-B-C-E-G
 - 30 тактов
- Итого: 3000 тактов

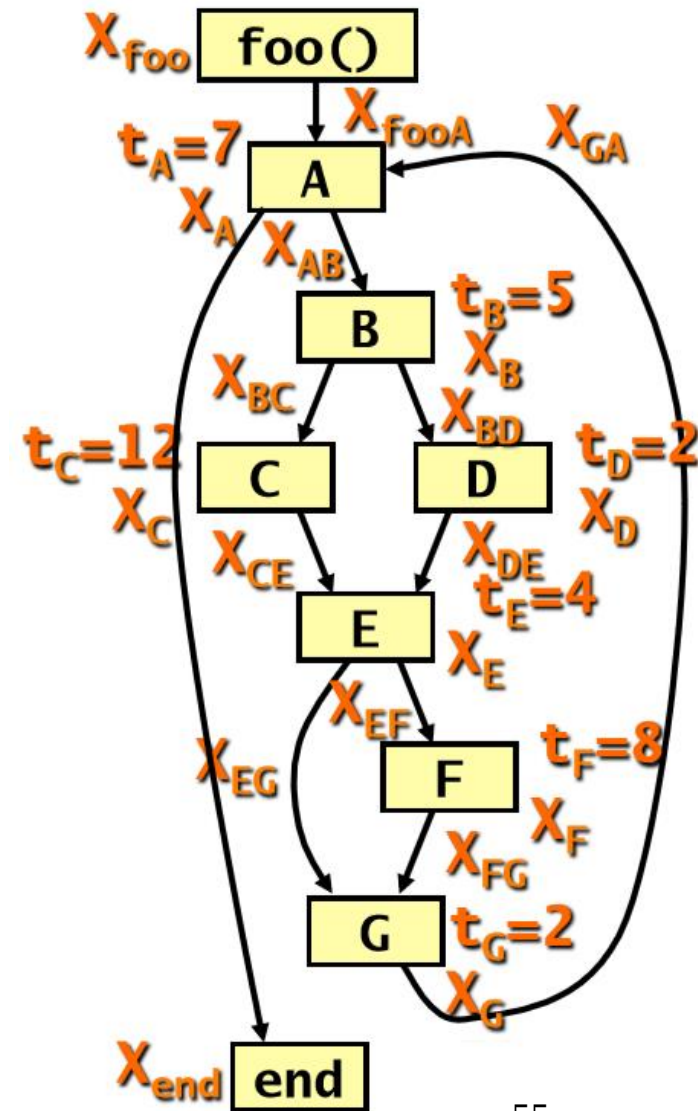
Расчёт WCET по путям: учёт конвейера

- Упорядочиваем допустимые пути по убыванию грубой оценки WCET (сумма оценок WCET для участков путей)
- для x из {допустимые_пути}
 - Вычислить $WCET_{PL}(x)$ с учётом «экономии» δ_{xy} от конвейерного выполнения последовательных участков
 - Если $WCET_{PL}(x)$ больше, чем наибольшая из грубых оценок WCET для оставшихся путей, или если других путей не осталось, то x – самый длинный (наихудший) путь; стоп
иначе продолжить цикл



Неявный перебор путей

- Implicit path enumeration technique (IPET)
 - Пути выполнения не обрабатываются в явном виде
- Представление программы
 - Информация о задержках (t_{entity})
 - Значения в узлах: выполнение участков
 - Значение на дугах: экономия за счёт конвейера
 - Число выполнений (X_{entity})



Неявный перебор путей

***WCET=**

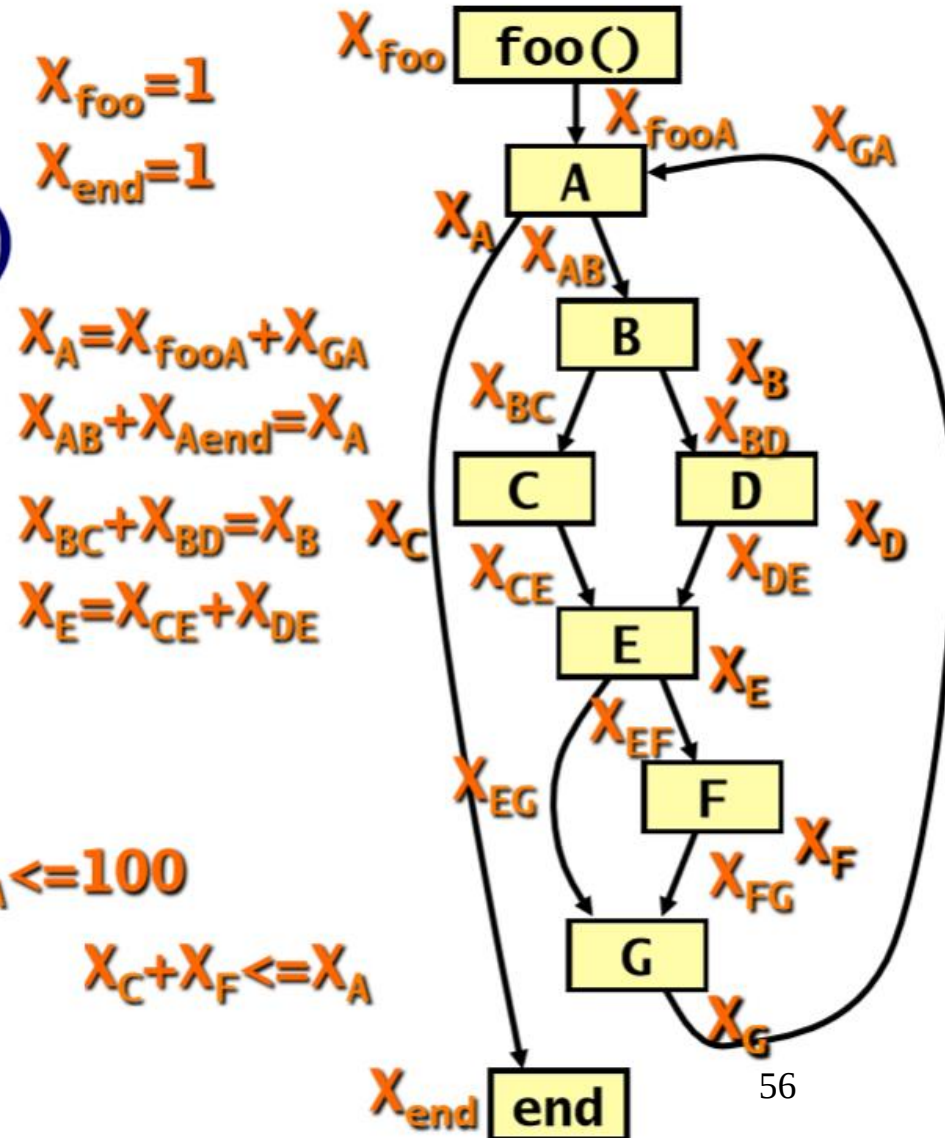
$$\max \sum (X_{entity} * t_{entity})$$

где совокупность X_{entity} удовлетворяет ограничениям:

- начальные и конечные условия
- структура программы
- ограничения на число итераций
- прочая потоковая информация

$$X_A \leq 100$$

$$X_C + X_F \leq X_A$$



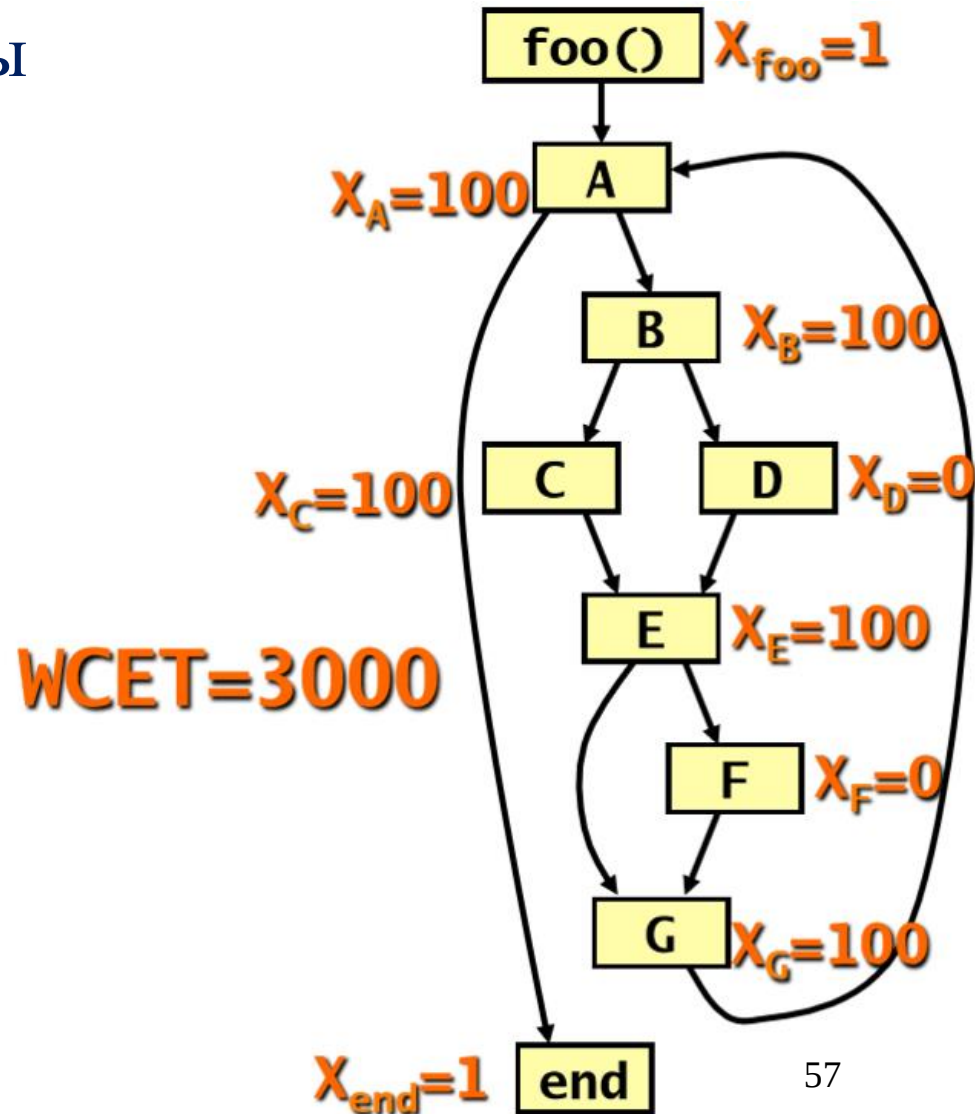
Неявный перебор путей

- Методы решения системы ограничений:

- Целочисленное линейное программирование
- Разрешение ограничений (constraint satisfaction)

- Результат

- Число выполнений для узлов и дуг
- Оценка WCET



Спасибо за внимание!

ИНФОРМАЦИОННО-УПРАВЛЯЮЩИЕ СИСТЕМЫ РЕАЛЬНОГО ВРЕМЕНИ

Лекция 6:

Оценка наихудшего времени выполнения программ (WCET) - 2

Кафедра АСВК,
Лаборатория Вычислительных Комплексов
Балашов В.В.

Почему важен WCET

- *Время выполнения* программ играет ключевую роль при анализе систем реального времени
- Это время, например, встречается в формуле:

Наихудшее время отклика

Период

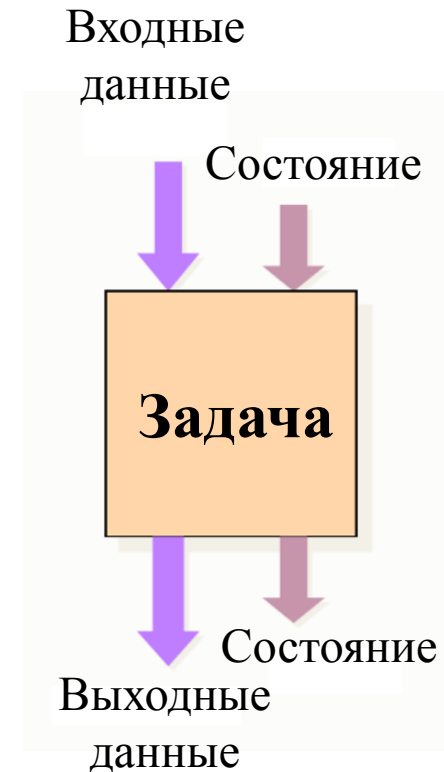
Наихудшее время выполнения

$$R_i = C_i + \sum_{j \in hp(i)} \lceil R_i / T_j \rceil C_j$$

Откуда берутся значения C_i, C_j ?

Простейшая вычислительная задача

- Входные данные доступны в момент старта
- Выходные данные готовы в момент завершения
- Нет блокировок в процессе выполнения
- Нет синхронизации или обмена данными в процессе выполнения
- Время выполнения зависит только от:
 - входных данных
 - состояния задачи в момент старта (внешние воздействия отсутствуют)



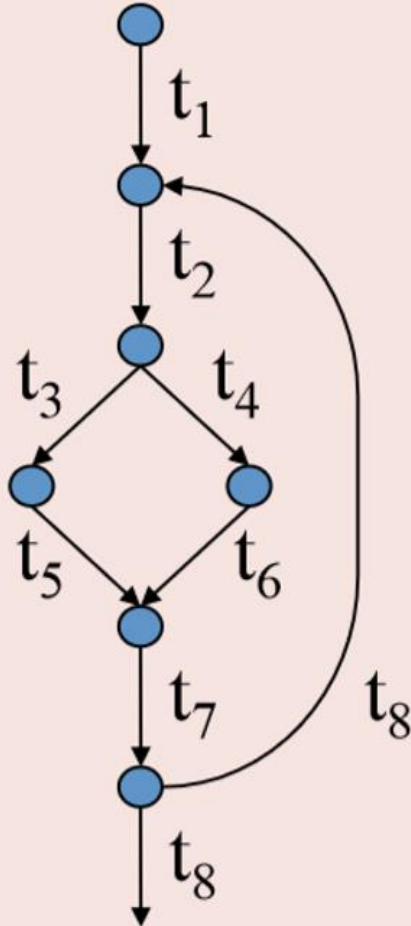
Наихудшее (максимальное) время выполнения

Наихудшее время выполнения программного кода (worst case execution time, WCET) – это максимальное время, которое требуется для выполнения

- данного фрагмента кода
- в данном контексте (входные данные, состояние)
- на заданном аппаратном вычислителе

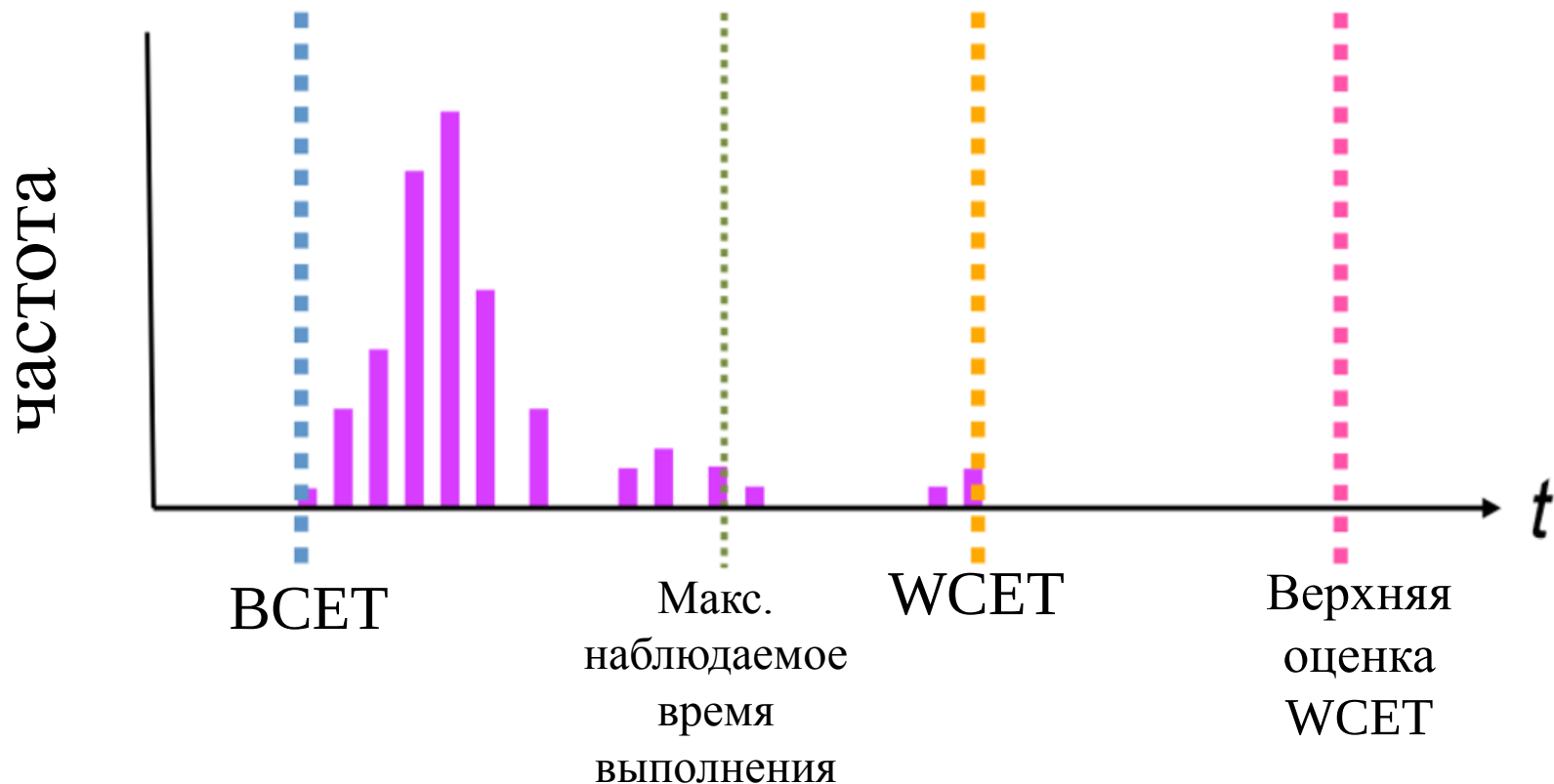
Чем определяется WCET

Задача



- Возможные **последовательности действий** задачи (пути выполнения)
- **Длительность** выполнения каждого действия на каждом допустимом (т.е. практически возможном) пути выполнения

Распределение времён выполнения



Множество различных времён выполнения

Нетривиальный анализ путей выполнения

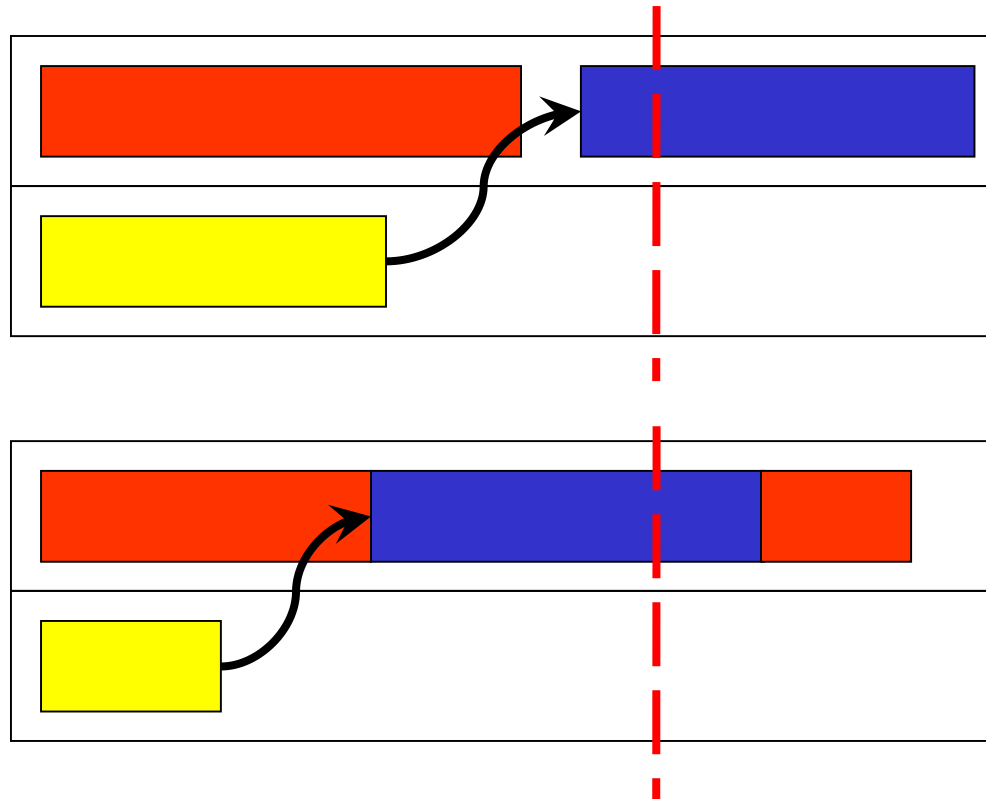
Сложное моделирование задержек от аппаратуры

Оптимизация программного кода для временной предсказуемости

Что важнее всего в таймингах работы RV-систем?

- В первую очередь – предсказуемость
 - Предсказуемый WCET
 - Также – предсказуемый BCET
- Производительность – во вторую очередь
- => потребность во временной предсказуемости:
 - Аппаратура
 - Программное обеспечение

Почему важен VCET



Желтая задача: $VCET \ll WCET$

➔ красная задача нарушает директивный срок

Performance Criteria

Write program π with best performance

MEAN

traditional: minimize $\sum_i p_i \cdot c(\pi, \delta_i, \mu)$

probability of input δ_i

machine

input data

WCET

HRT: minimize $\max_i c(\pi, \delta_i, \mu)$

Традиционное программирование

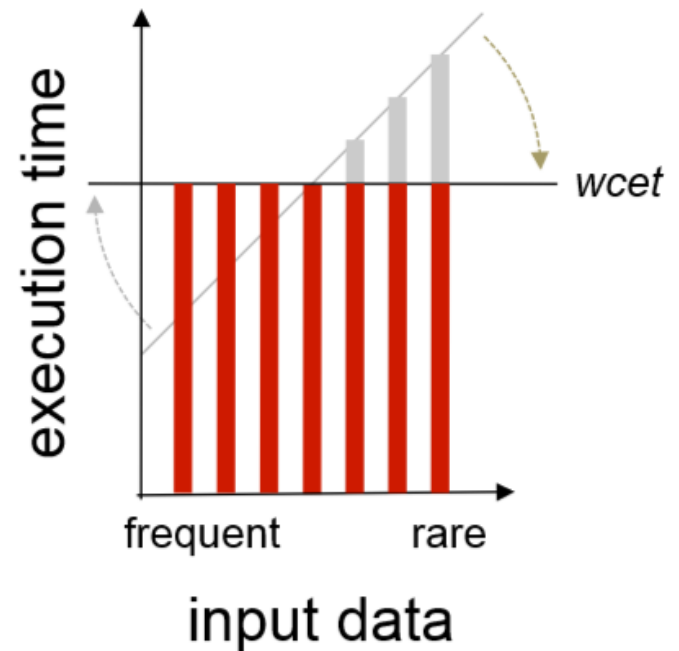
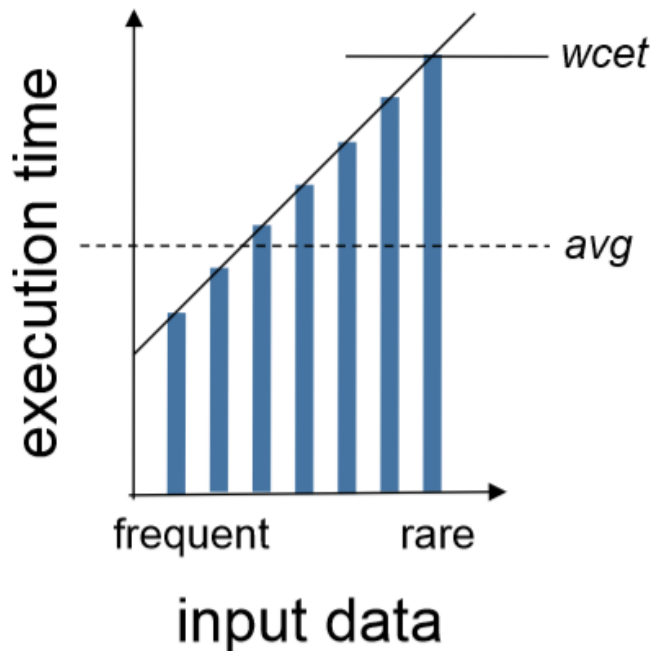
- Цель: хорошая производительность в среднем
- Стратегия: профилировать и оптимизировать производительность на наиболее часто встречающихся наборах входных данных => возможное ухудшение WCET

Причина: сложность неравномерно распределена между наборами входных данных; оптимизация частых сценариев приводит к ухудшению производительности на редких сценариях

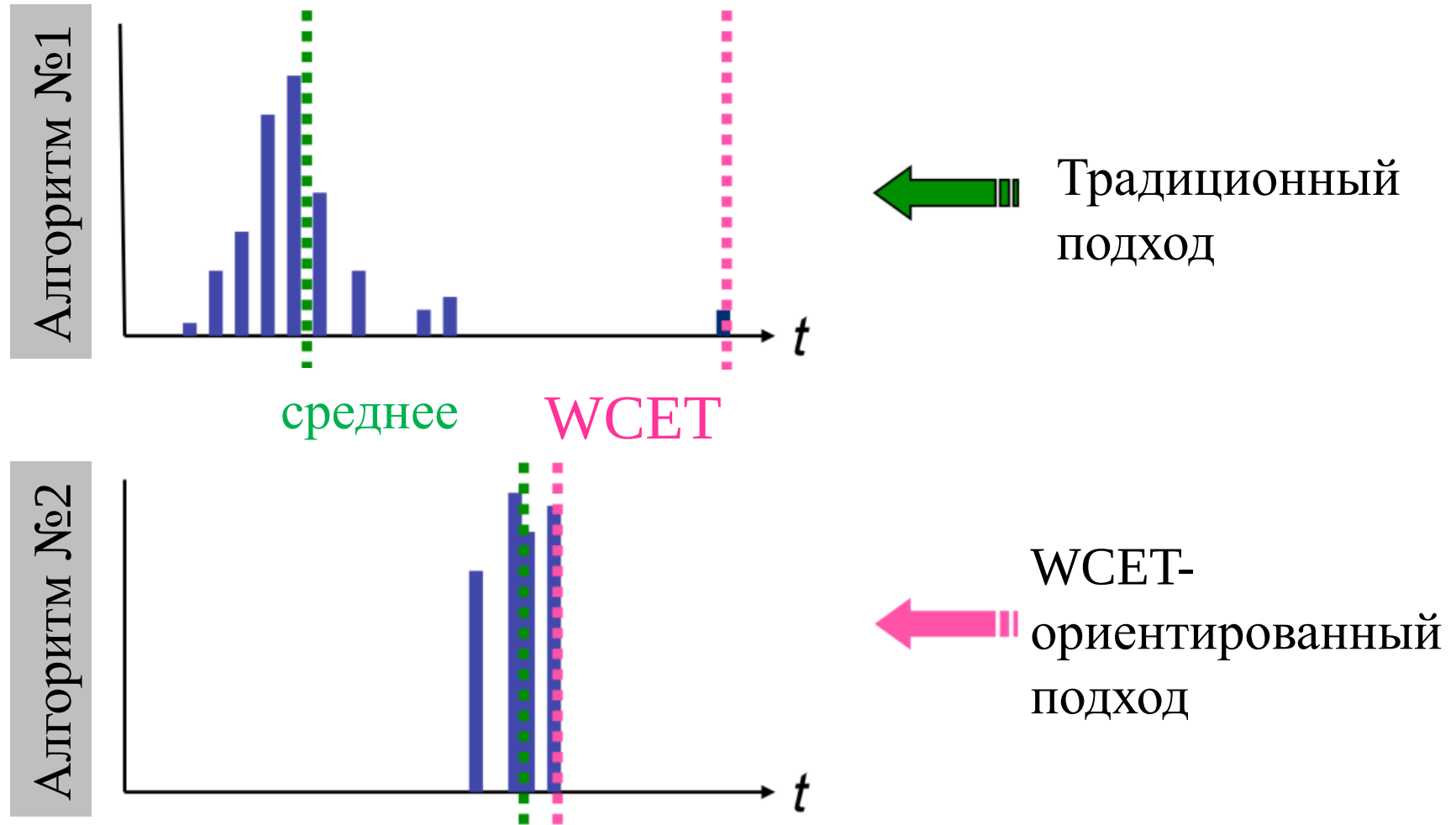
WCET-ориентированное программирование

- ❑ Стремимся освободить код от ветвлений в зависимости от входных данных
- ❑ Минимизируем число действий, выполняемых только для отдельных наборов входных данных

Традиционное vs. WCET-ориентированное программирование



Критерий производительности



Эксперименты

- Традиционный vs. WCET-ориентированный код
- Ветвящийся vs. линейный код

Реализованные алгоритмы:

- Пузырьковая сортировка
- Поиск первого вхождения
- Двоичный поиск
 - одно/два сравнения на каждой итерации


```

static int binSearch_avg(int key, int a[])
{
    int left = 0, right = SIZE-1, idx, inc;
    int found = 0;

    do
    {
        idx = (right + left) >> 1;
        if (a[idx] == key)
        {
            found = 1;
        }
        else if (a[idx] < key)
        {
            left = idx+1;
        }
        else
        {
            right = idx-1;
        }
    } while (!found && (right >= left));

    if (found)
    {
        return idx;
    }
    else
    {
        return -1;
    }
}

```

```

static int binSearch_wcet(int key, int a[])
{
    int left = 0, right = SIZE-1, idx, inc;

    idx = (right + left) >> 1;

    for(inc = SIZE; inc > 0; inc = inc >> 1)
    {
        right = (key < a[idx] ? idx - 1 : right);
        left = (key > a[idx] ? idx + 1 : left);
        idx = (right + left) >> 1;
    }

    return idx;
}

```

Эксперименты

- Традиционное vs. WCET-ориентированное программирование

Алгоритм	Традиционное		WCET-ориентированное	
	Среднее	WCET	Среднее	WCET
Bubble*	599	724	609	663
Find-first*	68	122	103	103
Bin-search	94	124	105	106

* В традиционном варианте: без goto; используется признак раннего завершения

Свойства WCET-ориентированных программ

- **WCET меньше**, чем у традиционных программ
- **Анализ путей проще** в связи с устранением/сокращением зависимостей от входных данных
- **Оценка WCET проще** из-за меньшего числа путей выполнения (или единственного пути)
- **Меньше разброс** времен выполнения => более стабильна работа системы в целом
- **Низкая сложность** потока управления => хорошо подходит для управляющих программ, критичных для безопасности
- Нетрадиционные алгоритмы

Стремимся к временной предсказуемости

Сокращаем зависимость порядка действий от
внешних факторов

⇒ Линейный (single-path) код

- Нет ветвлений по входным данным
- Предикатное выполнение
- Акцент на потоке данных (а не на потоке управления)

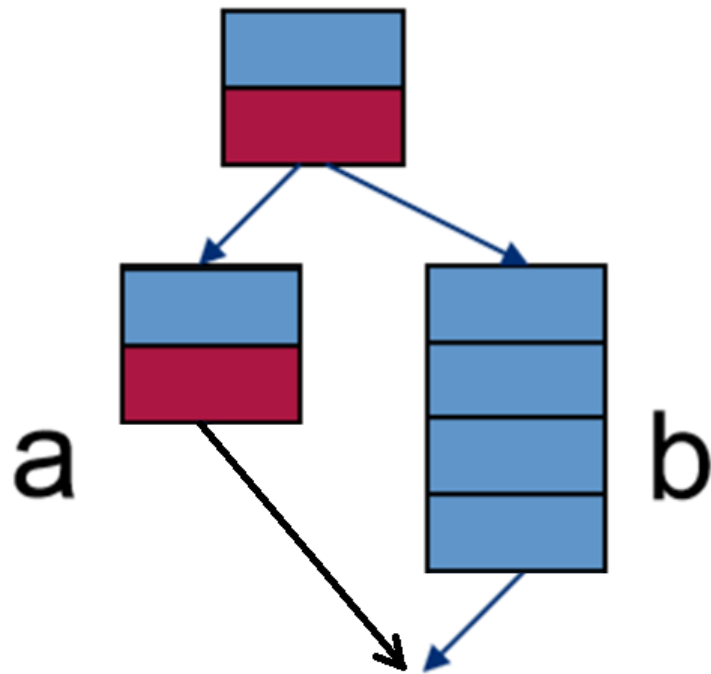
WCET-ориентированное программирование + линеаризация кода

- Константное время выполнения
- Низкий WCET
- Тривиальный анализ путей
- Простой анализ WCET

⇒ Полная временная предсказуемость

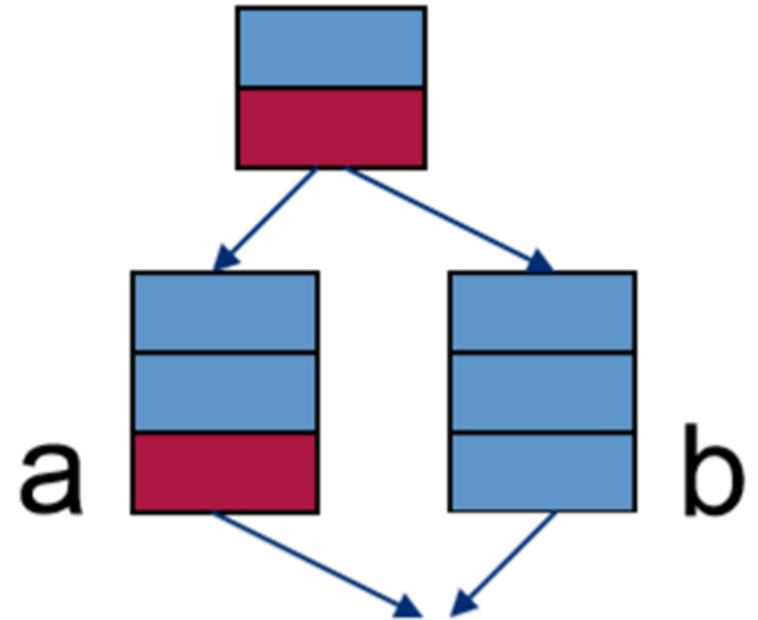
⇒ Приемлемая производительность

Линеаризация: код, оптимизированный под средний случай, замедляется сильнее



$$T_{\text{avg}}: 4,1 \rightarrow 8$$

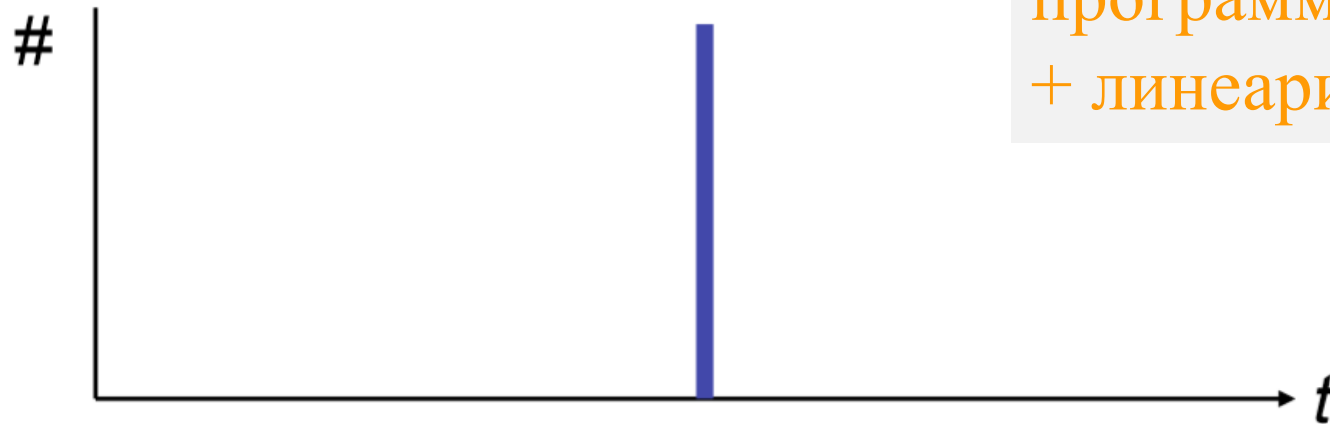
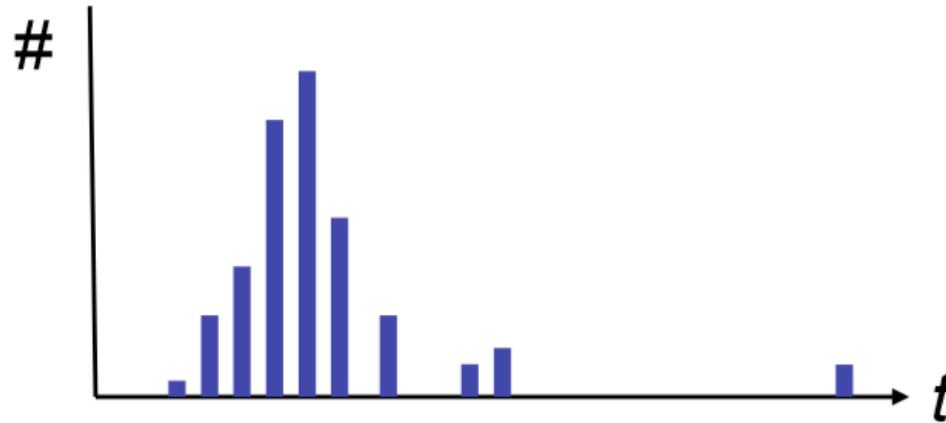
(x1,95)



$$T_{\text{avg}}: 5 \rightarrow 8$$

(x1,6)

Времена выполнения

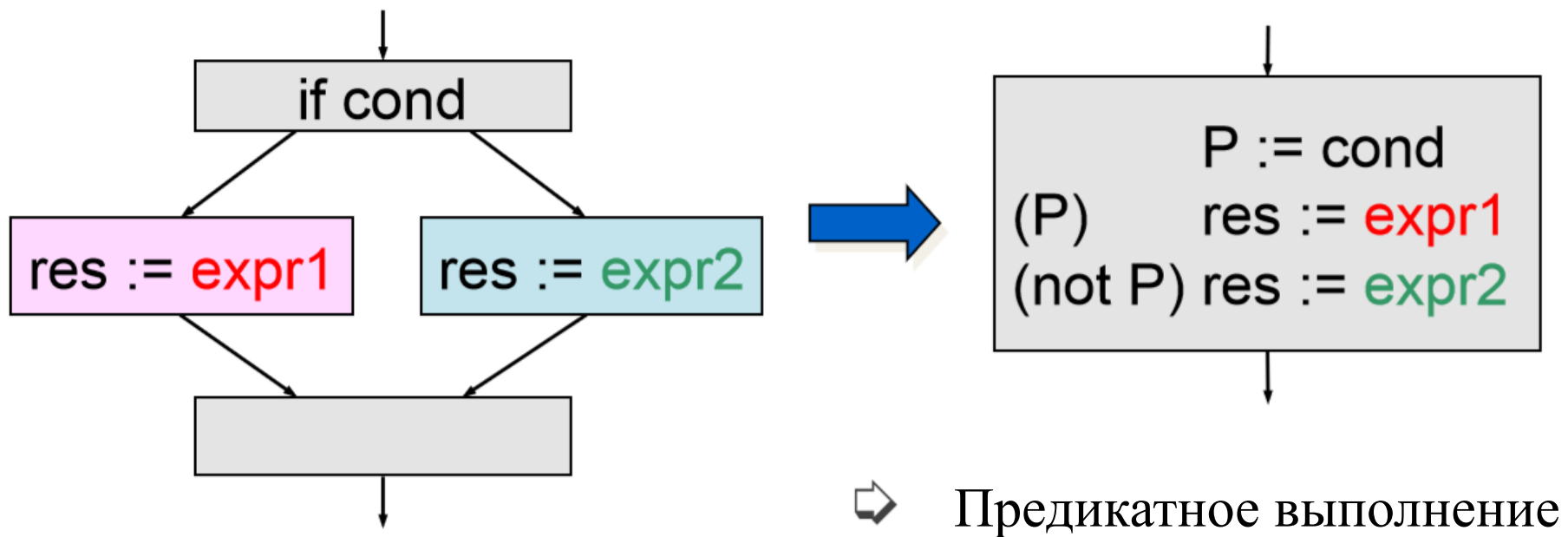


Времена
выполнения кода до
и после перехода на
WCET-
ориентированное
программирование
+ линеаризации

Как получить линейный код?

Устранить зависимости по данным из потока управления

- Аппаратура с константными задержками
- Линеаризованный код



Предикатное выполнение

*...это условное выполнение или невыполнение команды в зависимости от значения булевой переменной, называемой предикатом
[Hsu et al. 1986]*

Выполнение предикатной команды:

- Безусловная выборка команды из памяти
- Предикат истинен $\Rightarrow$ нормальное выполнение команды
- Предикат ложен $\Rightarrow$ команда не изменяет состояние процессора

Ветвящийся vs. предикатный код

Пример кода:

if rA < rB **then** swap(rA, rB);



Ветвящийся код

```
cmplt rA, rB
bf     skip
swp    rA, rB
skip:
```



Предикатный код

```
predlt Pi, rA, rB
(Pi) swp    rA, rB
```

Аппаратная поддержка предикатного выполнения

Предикатные регистры

Команды для работы с предикатами (определить, установить, сбросить, загрузить, сохранить)

- Предикатные команды
 - Поддержка **полной предикатности**: выполнение любых команд управляется предикатами
 - Поддержка **частичной предикатности**: лишь некоторое подмножество команд могут быть предикатными (например, копирование данных, присваивание значений)

Специфика частичной предикатности

Опережающее выполнение вычислений

- Команды, не являющиеся предикатными, выполняются безусловно
- Их результаты сохраняются во временных переменных
- В дальнейшем предикат определяют, содержание какой из временных переменных следует использовать

Пример:

(pred)	cmov dest, src1
(not pred)	cmov dest, src2

Внимание: команды, вычисляющие значения временных переменных, не должны генерировать исключения (пример: деление на ноль, обращение к недопустимому адресу памяти).

Полностью vs. частично предикатный код

Оригинальный код

```
if src2 ≠ 0 then dest := src1 / src2;
```



Полностью
предикатный код:

```
Pred := (src2 ≠ 0)
```

```
(Pred) div dest, src1, src2
```

Fully vs. Partially Predicated Code (2)

Original code:

```
if src2 ≠ 0 then dest := src1 / src2;
```



Partially predicated code, first attempt:

may raise
an exception
on division
by zero

```
Pred := (src2 ≠ 0)
```

(Pred)

```
div    tmp_dst, src1, src2  
cmov   dest, tmp_dst
```

Fully vs. Partially Predicated Code (3)

Original code:

```
if src2  $\neq$  0 then dest := src1 / src2;
```



Partially predicated code:

if src2 equals 0,
then replace it
by a safe value
(e.g., 1) to avoid
division by zero

	Pred := (src2 $\neq$ 0)
(not Pred)	mov tmp_src, src2 cmov tmp_src, \$safe_val
(Pred)	div tmp_dst, src1, tmp_src cmov dest, tmp_dst

Минимальная поддержка предикатности

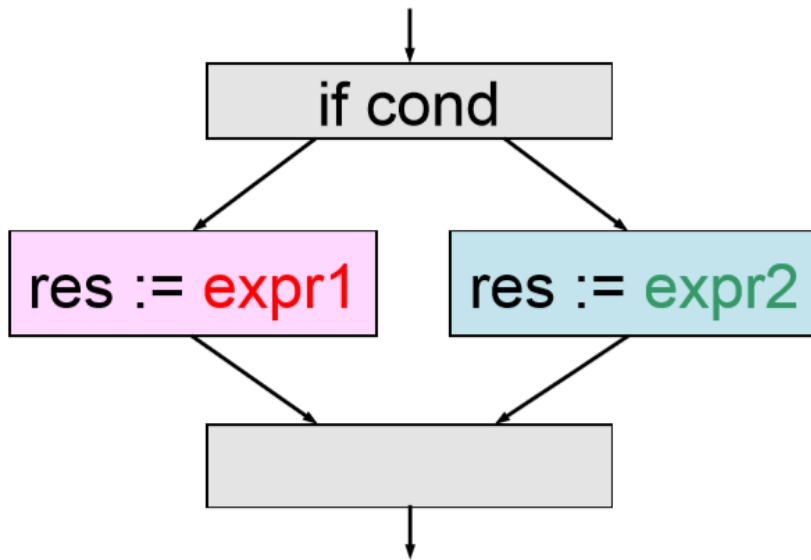
- Команда условного копирования:

```
movCC destination, source
```

Семантика:

```
if CC  
then destination := source  
else no operation
```

If-преобразование с условным копированием



```
t1 := expr1'  
t2 := expr2'  
test cond  
movT res, t1  
movF res, t2
```

} Избегать
побочных
эффектов

Эмуляция условного копирования

- В архитектурах без поддержки предикатности, условное копирование можно эмулировать при помощи операций с битовыми масками.

Пример: **if (cond) x=y; else x=z;**



```
t0 = 0 – cond;      // fat bool: 0..false, -1..true
t1 = ~t0;           // bitwise negation (fat bool)
t2 = t0 & y;
t3 = t1 & z;
x = t2 | t3;
```

Допущение: переменные всех используемых типов имеют одинаковый размер

Пример:

```
for(i=SIZE-1; i>0; i--)  
{  
  for(j=1; j<=i; j++)  
  {  
    if (a[j-1] > a[j])  
    {  
      t = a[j];  
      a[j] = a[j-1];  
      a[j-1] = t;  
    }  
  }  
}
```



Пузырьковая сортировка: на входе массив $a[SIZE]$

```
for(i=SIZE-1; i>0; i--)  
{  
  for(j=1; j<=i; j++)  
  {  
    t1 = a[j-1];  
    t2 = a[j];  
  
    (t1>t2): t = a[j];  
    (t1>t2): a[j] = a[j-1];  
    (t1>t2): a[j-1] = t;  
  }  
}
```

Пример:

Пузырьковая сортировка: на входе массив $a[SIZE]$

```
for(i=SIZE-1; i>0; i--)  
{  
  for(j=1; j<=i; j++)  
  {  
    if (a[j-1] > a[j])  
    {  
      t = a[j];  
      a[j] = a[j-1];  
      a[j-1] = t;  
    }  
  }  
}
```



```
for(i=SIZE-1; i>0; i--)  
{  
  for(j=1; j<=i; j++)  
  {  
    cond = (a[j-1] > a[j]);  
  
    (cond): t = a[j];  
    (cond): a[j] = a[j-1];  
    (cond): a[j-1] = t;  
  }  
}
```

Пример:

Пузырьковая сортировка: на входе массив $a[SIZE]$

```
for(i=SIZE-1; i>0; i--)  
{  
  for(j=1; j<=i; j++)  
  {  
    if (a[j-1] > a[j])  
    {  
      t = a[j];  
      a[j] = a[j-1];  
      a[j-1] = t;  
    }  
  }  
}
```



```
for(i=SIZE-1; i>0; i--)  
{  
  for(j=1; j<=i; j++)  
  {  
    t1 = a[j-1];  
    t2 = a[j];  
  
    test (t1>t2);  
    movt: a[j-1] = t2;  
    movt: a[j] = t1;  
  }  
}
```

Свойства линейного кода

- Каждое выполнение порождает одну и ту же трассу команд, т.е. одну и ту же последовательность обращений к памяти команд.
- Анализ путей тривиален – существует единственный путь.
- Два выполнения, начинающиеся с одинакового состояния кэша команд, порождают одинаковые последовательности промахов/попаданий в кэш команд.

Линейный код и задержки

Каждое выполнение порождает одну и ту же последовательность (а также количество) выполненных инструкций -> хорошая основа для получения фиксированного времени выполнения.

Команды с неконстантным, зависящим от данных, временем выполнения вызывают флуктуации времени выполнения кода.

Разные начальные состояния памяти (в т.ч. кэша) могут привести к различным задержкам при доступе к памяти команд и данных, а значит к различным временам выполнения кода.

Обеспечение константного времени выполнения

Не позволяем внешним факторам влиять на:

- Последовательность выполняемых команд
 - Длительность команд
-
- Всегда начинать с одного и того же состояния кэша команд, конвейера, логики предсказания ветвлений и т.п.
 - Обеспечивать константные задержки при доступе к данным
 - Обеспечивать константные длительности всех операций процессора
 - Все внешние воздействия (в т.ч. вытеснение задач) должны быть предсказуемыми

Что делать:

- Сбрасывать кэши при старте задач
- Размещать данные по фиксированным адресам
- Избегать сложного вычисления адресов

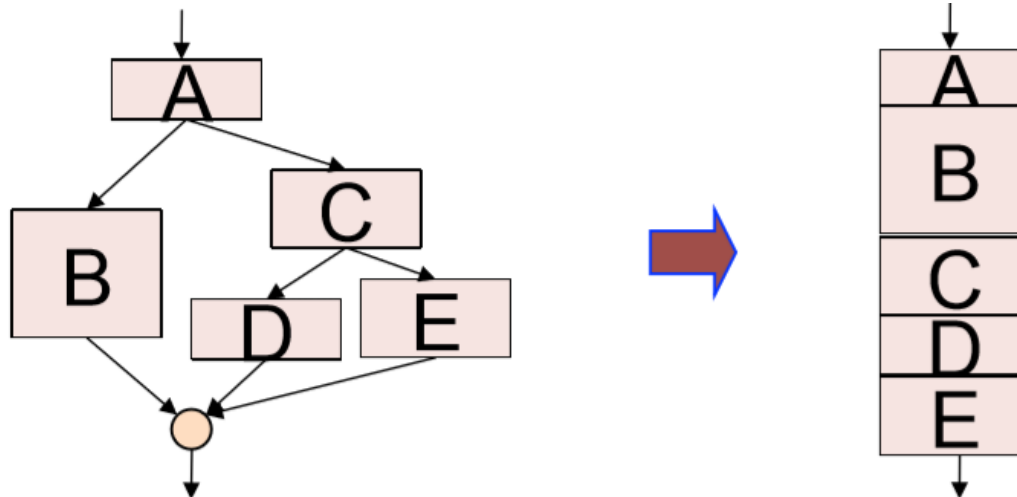
Константные длительности операций процессора

- Все операции процессора должны быть реализованы так, чтобы выполняться за константное время, независимо от значений операндов (пример: целочисленный сумматор всегда отрабатывает за 1 такт)
- В частности, предикатные команды должны выполняться за константное время. Если предикат ложен, команда выполняется, но её результаты отбрасываются и не изменяют состояние процессора.
- У циклически выполняемых команд в коде должны быть константные значения числа итераций.

Производительность линейного кода

Времена выполнения ветвей, выбираемых в зависимости от входных данных, суммируются в связи с линеаризацией.

- Время выполнения линейного кода довольно велико (по сравнению с оригинальным), если путь выполнения оригинального кода существенно зависит от входных данных



Производительность линейного кода (2)

Процессоры с длинными конвейерами тратят много тактов на перезаполнение конвейера при ошибочном предсказании ветвления.

- Предикатное выполнение может быть «дешевле» ветвления
- Современные компиляторы и процессоры могут использовать предикатное выполнение для улучшения производительности

Пример: ускорение за счет if-преобразования

```
if rA < rB then swap(rA, rB);
```

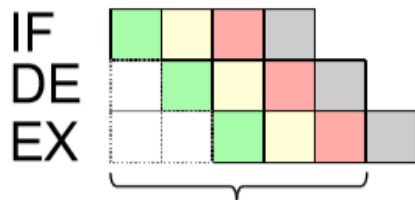
Ветвящийся код

■ `cmplt rA, rB`
■ `bf skip`
■ `swp rA, rB`
■ `skip:`

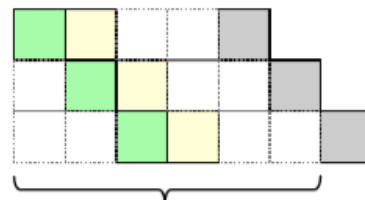
Предикатный код

■ `predlt Pi, rA, rB`
■ `(Pi) swp rA, rB`
■

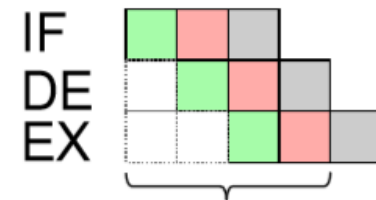
Выполнение в трёхступенчатом конвейере:



5 тактов



6 тактов

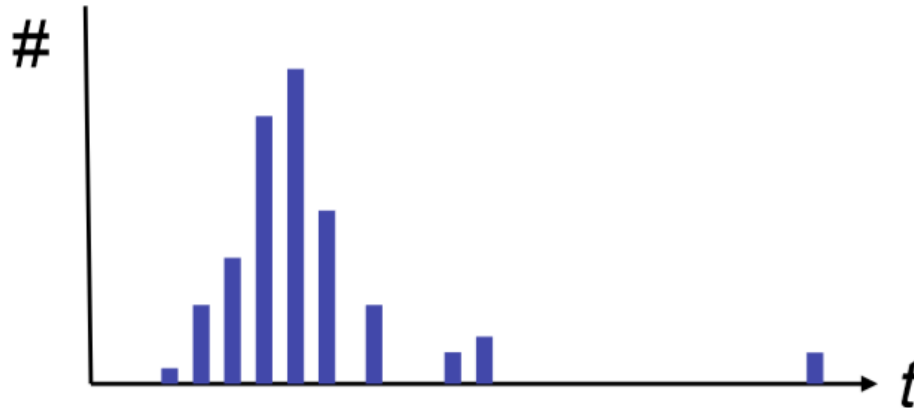


4 такта

Свойства процедуры линеаризации кода

- **Полнота**: любой участок кода с ограниченным WCET может быть линеаризован
- В линеаризованном коде путь выполнения – **единственный**
- Анализ WCET **тривиален**: запустить код и измерить время (при «наихудшем» начальном состоянии системы)
- Код выполняется заметно дольше ветвящегося кода

Времена выполнения

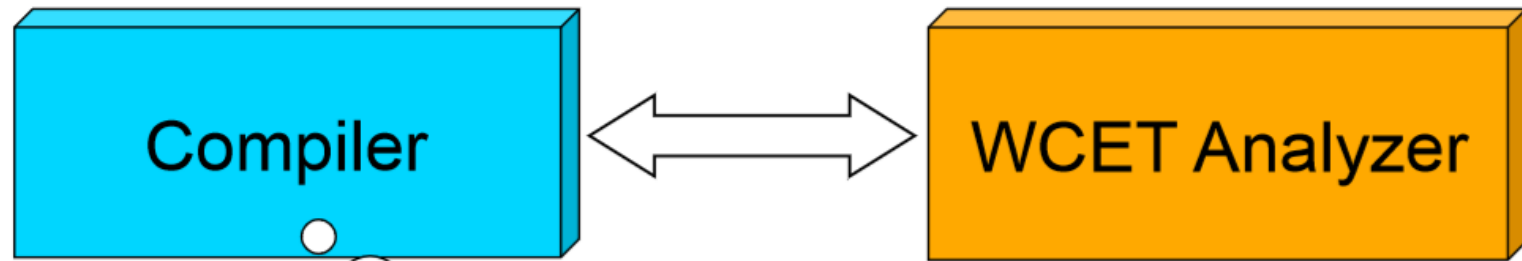


Времена выполнения
до и после
линеаризации



Оптимизация WCET на этапе компиляции

WCET-aware Code Optimizations

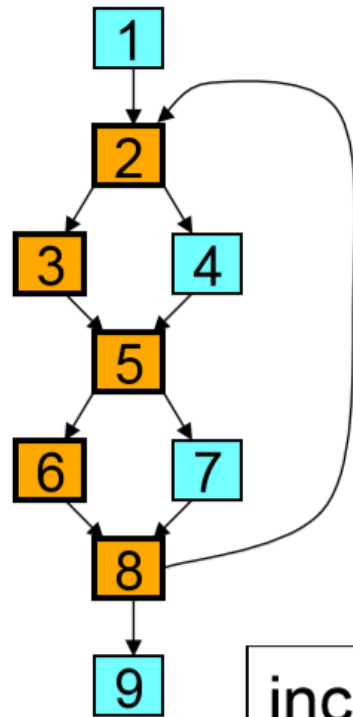


- localize worst-case path,
- select appropriate optimizations,
- identify code locations to apply optimizations

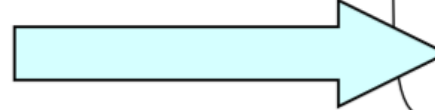
WCET-aware Code Optimizations

Superblock Formation:

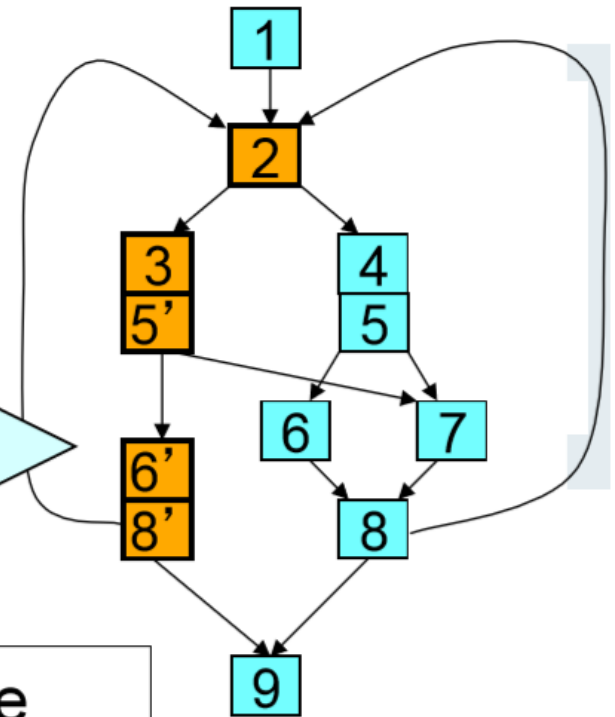
superblock:
...block with
single entry
point



superblock
formation



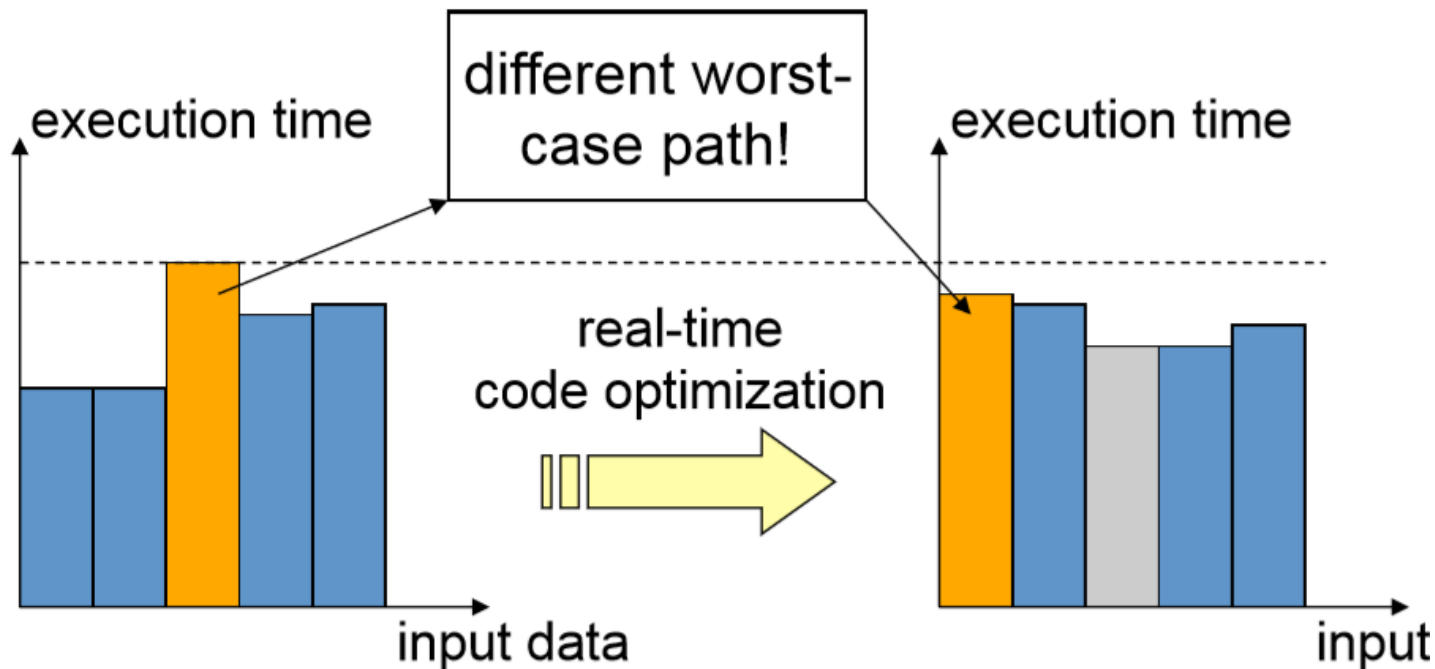
■ ...worst case path



increase of code size
but reduction of WCET

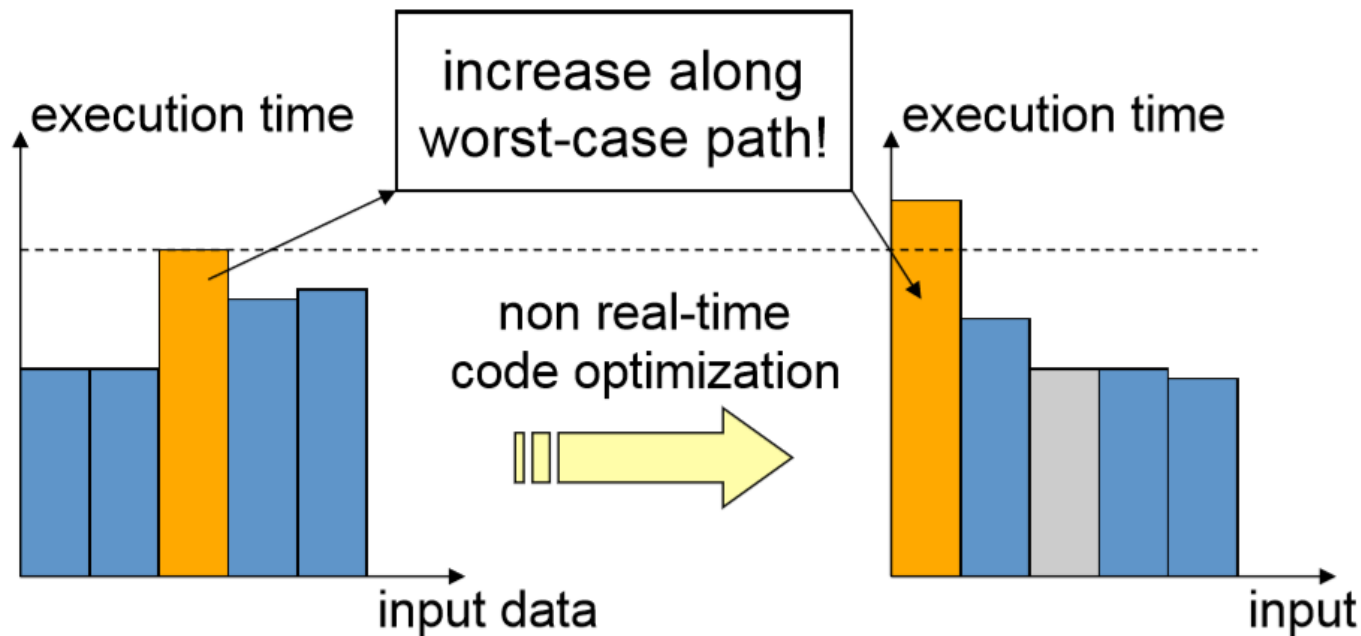
Наихудший путь

- *Внимание:* в результате оптимизации другой путь может стать наихудшим



Наихудший путь

- *Внимание:* новый наихудший путь может выполняться дольше, чем первоначальный



Или всё-таки замерять?

Почему нельзя просто измерить WCET?

- ❑ Замер времени выполнения на **всех** путях выполнения реалистичной программы – на практике **невозможен**
- ❑ При определении тестовой выборки могут быть **упущены редкие сценарии** выполнения (обработка ошибочных ситуаций и т.п.)
- ❑ Выбранные тестовые данные **могут не породить самую длинную** (по времени) **трассу** выполнения
- ❑ Внутреннее **состояние процессора** на момент старта измерений может не быть наихудшим

Простые замеры могут послужить источником **первоначальной (грубой)** *нижней* оценки WCET

С другой стороны...

Не во всех случаях строго необходима безопасная (не заниженная) оценка WCET

- Системы мягкого реального времени (например, мультимедиа)
- Системы, устойчивые к редким превышениям директивных сроков

Для новой аппаратной платформы быстрее всего можно начать именно замеры времени выполнения (а статические методы анализа аппаратных задержек адаптировать долго)

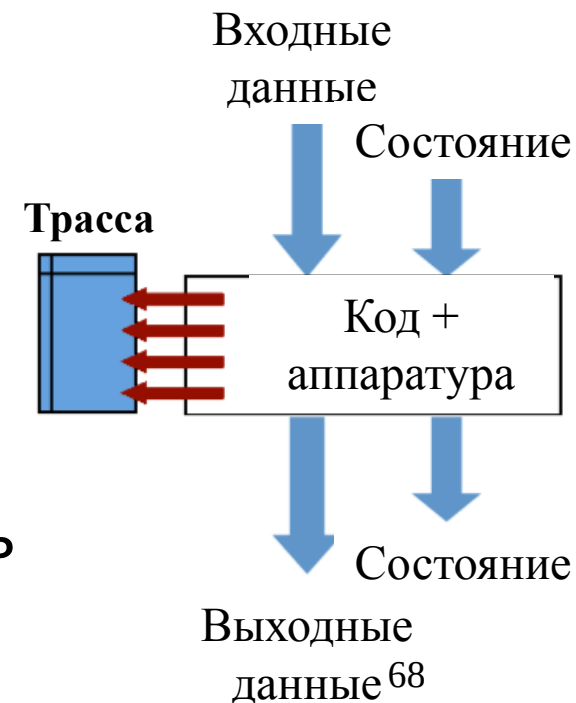
Низкие затраты на аннотирование кода => быстрое получение грубой оценки времени выполнения

Замеры дополняют статический анализ, предоставляя реальные значения задержек
Замеры могут использоваться для уточнения результатов статического анализа WCET

- Моделировать некоторые современные процессоры *действительно* сложно
- Для некоторых задач зависимость времени выполнения от входных данных *по-настоящему* сложна (вещественные числа, массивы и т.п.)
- Люди из промышленности требуют подтверждения теоретических оценок WCET данными «из жизни»

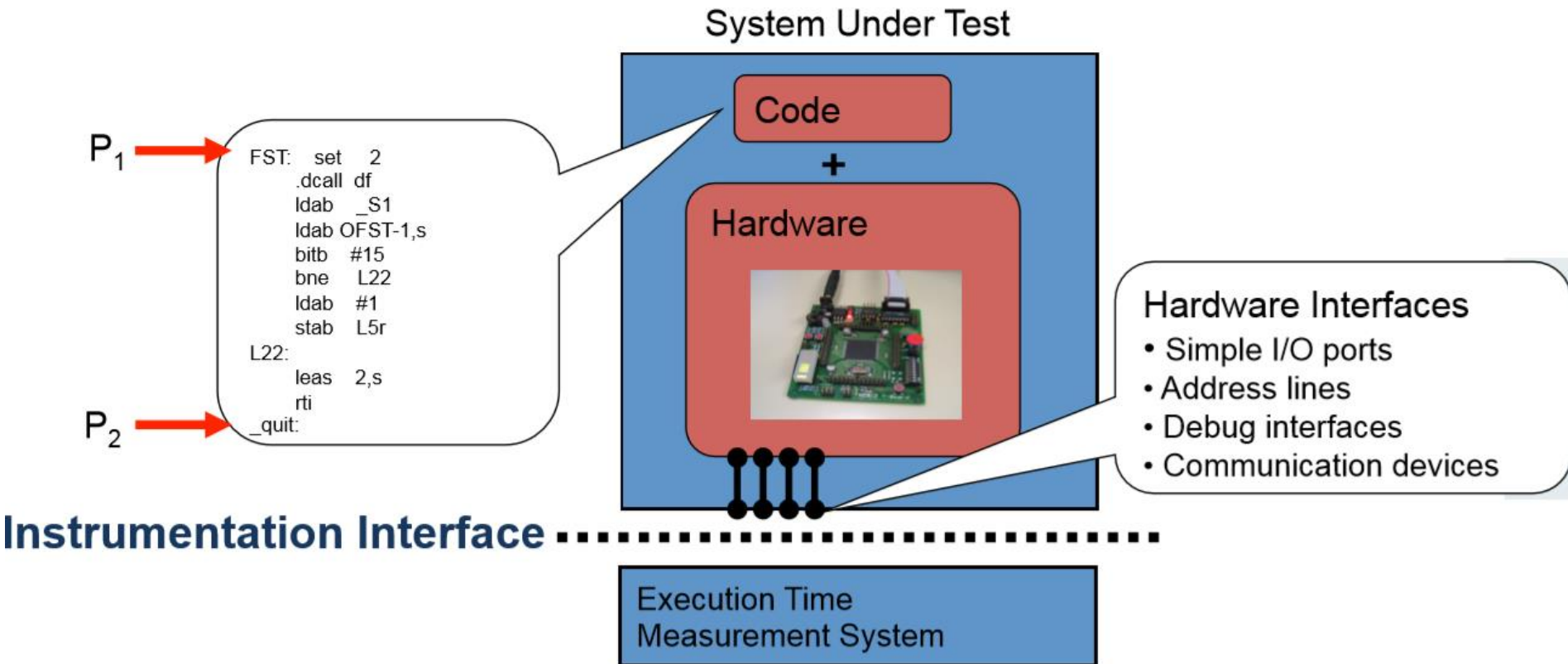
Оценка WCET с помощью измерений

- Информация о таймингах получается путём **измерения** времени **выполнения** кода на реальной **целевой аппаратуре**
- Точки инструментирования кода формируют **наблюдаемые извне события**, используемые для старта и завершения измерений
- Трасса выполнения содержит собираемую *совместно* информацию о **путях** в программе и **временах** их **выполнения** (путь = последовательность линейных участков)



Execution Time Measurements

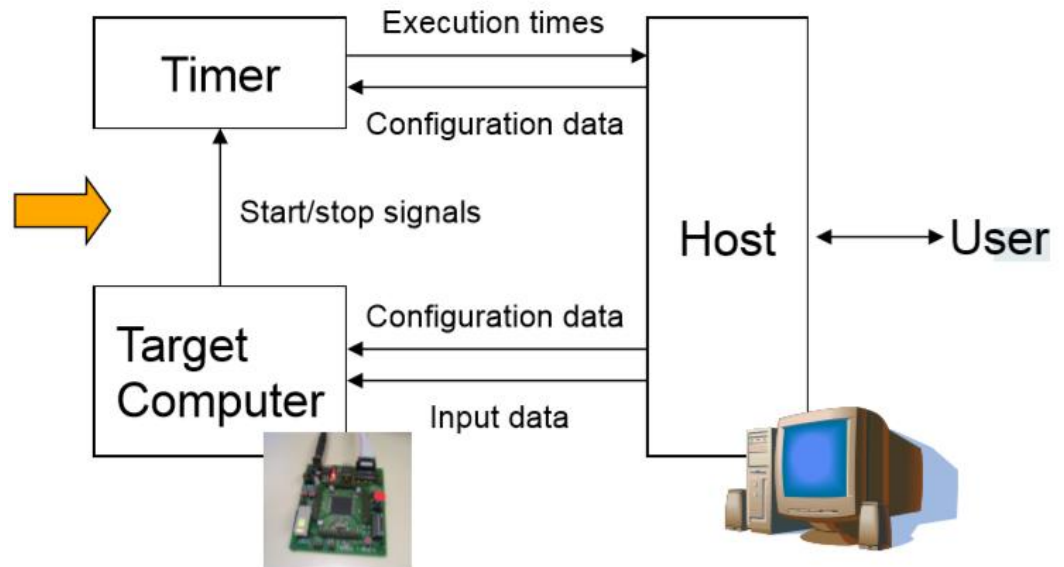
Goal: Obtain execution time for a **path**



- Необходимо СБРОСИТЬ состояние процессора (кэш, конвейер и т.п.) перед каждым прогоном
- НО: для измерения WCET конкретного пути необходимо специфическое начальное состояние кэша (например, если путь входит в состав второй или более поздней итерации цикла, команды должны *находиться в кэше*)

Методы инструментирования

- Чисто аппаратное инструментирование
- Внешние замеры времени выполнения при помощи программно формируемых сигналов вовне
- Чисто программное (внутреннее) инструментирование



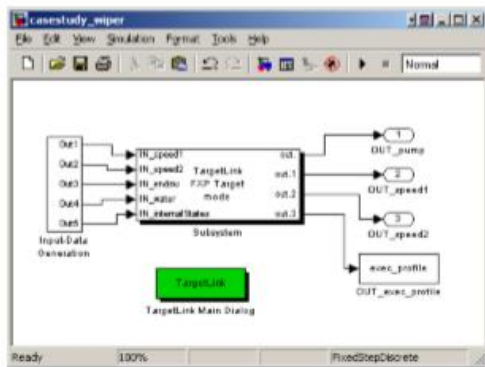
Важные соображения по замерам

Как измерить именно то, что нужно:

- Инструментирование кода не должно изменять путь выполнения или время выполнения программы *непредсказуемым* или *неизвестным* способом. Точки инструментирования должны располагаться в фиксированных местах кода.
- Нужна уверенность в том, что запуски для замеров всегда стартуют с известного требуемого состояния процессора (кэш, конвейер, предсказание ветвлений и т.п.)

Промышленный подход

- Пример промышленного процесса разработки:

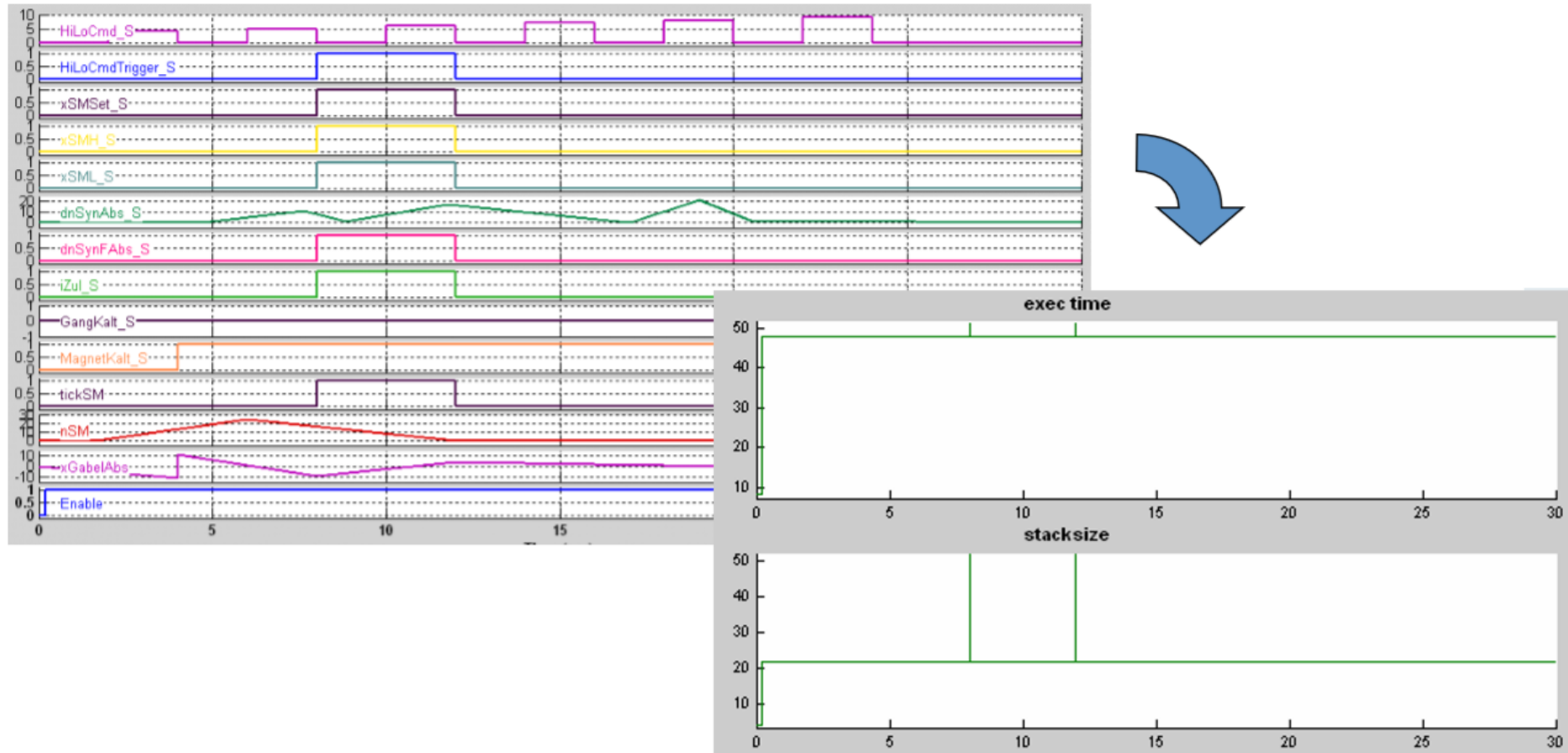


Проектирование
Matlab + Simulink
Matlab + RTW

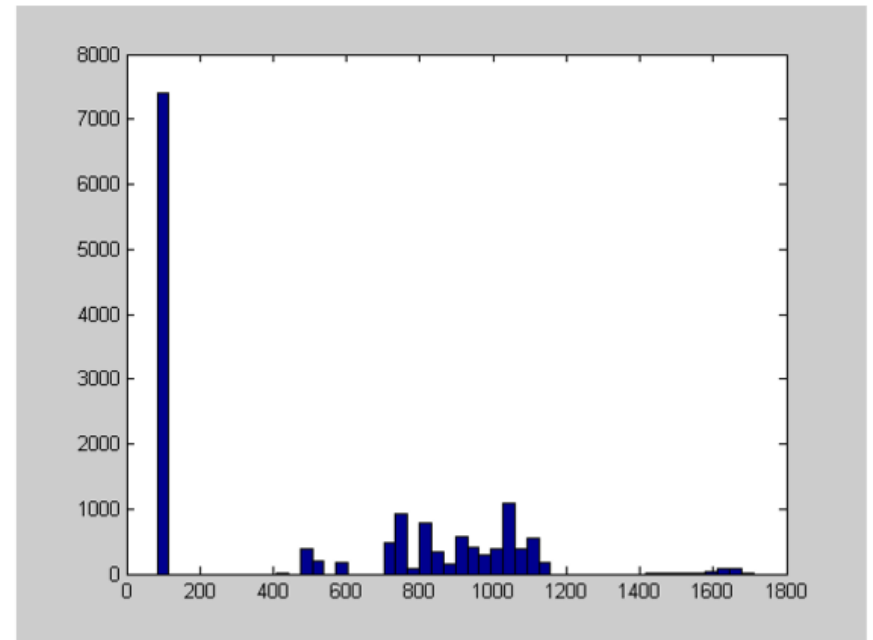
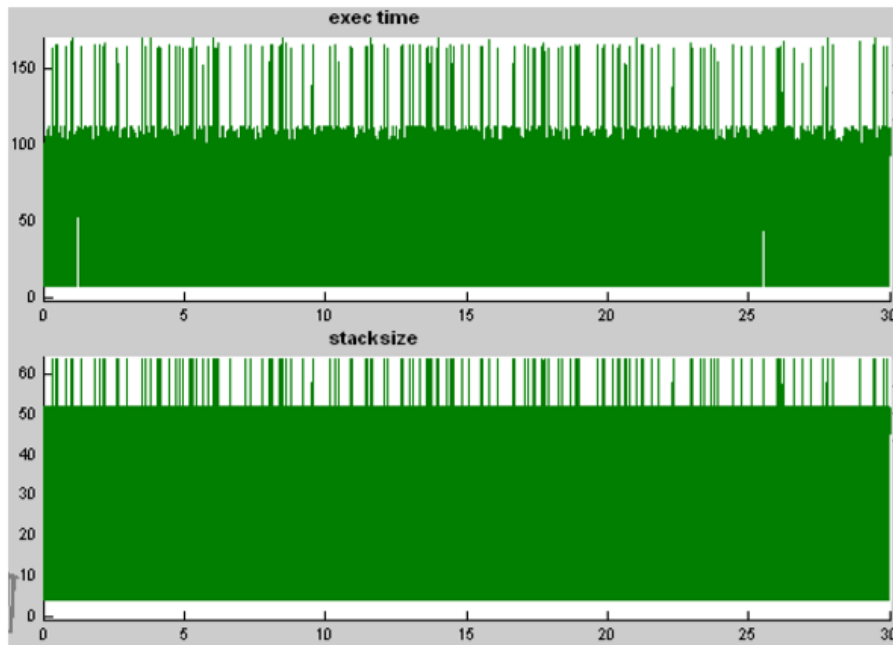
Тестирование
Платы-прототипы
Аппаратура целевой системы

Внедрение
Аппаратура целевой
системы

Промышленный подход – массивы входных данных



Промышленный подход – случайные данные

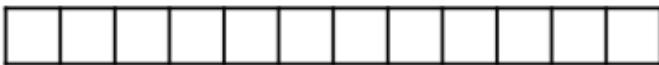


Поиск «наихудших» входных данных – оптимизационная задача

- Массивы входных данных → ограниченный перебор
- Случайные входные данные → случайный поиск
- Как насчёт «умной» оптимизации?

Evolutionary Algorithms (EA)

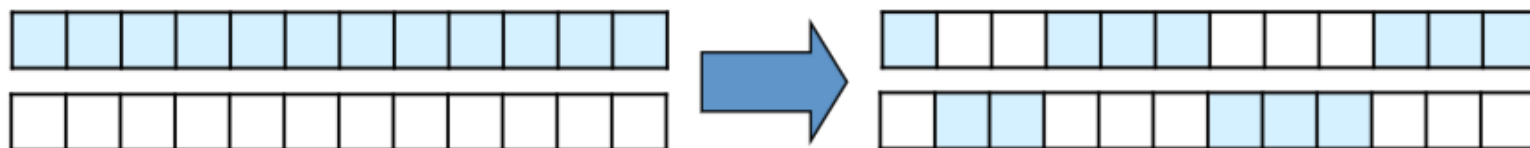
Gene = an independent property of an individual

Individual = vector of genes 

Population = set of a number individuals

Fitness value = chance of survival of an individual

Recombination = mating of two individuals,
exchange of genes



Mutation = random change of a gene

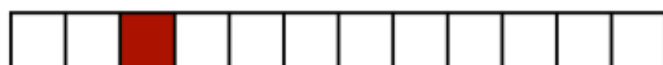
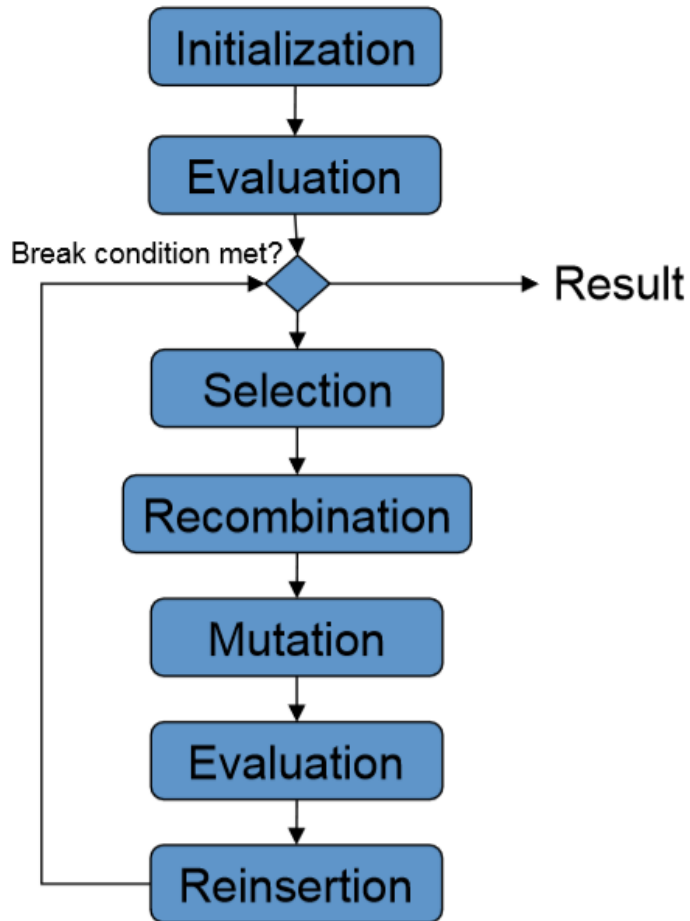


Схема эволюционного алгоритма



- Отбор: выживание наиболее приспособленных. Отбор рандомизирован и выполняется с учётом значения целевой функции.
- Скрещивание: обмен генами, например одно- или n-точечное скрещивание
- Мутация: рандомизированное изменение генов

Оценка WCET эволюционными алгоритмами

- Ген = значение входной или внутренней переменной
- Целевая функция = измеренное время выполнения (больше значение => особь лучше)
- Результат = «наилучшая особь» = особь (набор данных) с наибольшим временем выполнения программы
- Даёт хорошую, но *небезопасную* оценку WCET

Оценка WCET эволюционными алгоритмами

```
if(x) {  
    fast();  
} else {  
    slow();  
}  
if(y) {  
    slow();  
} else {  
    fast();  
}
```

- Старт

- [0] $x = 0, y = 0 \rightarrow$ ET: 40

- [1] $x = 1, y = 1 \rightarrow$ ET: 40

- Скрещивание:

- [2] $x = 0, y = 1 \rightarrow$ ET: 50

- [3] $x = 1, y = 0 \rightarrow$ ET: 30

Алгоритм завершается, если значение целевой функции не улучшается на протяжении заданного числа итераций

Результат применения эволюционного алгоритма

Программа	WCET	WCET SA	WCET EA	
Matrix	13,190,619	15,357,471	13,007,019	Завышенная Заниженная
Sort	11,872,718	24,469,014	11,826,117	
Graphics	N/A	2,602	2,176	
Railroad	N/A	23,466	22,626	
Defense	N/A	72,350	35,226	

SA Static analysis
EA Evolutionary algorithms

Нет результата

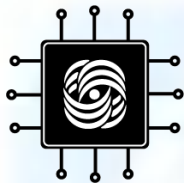
Точность - ?

[Mueller, Wegener, RTSS1998]

Выводы

- Оценка WCET – сложная задача, разрешимая (безопасно и с приемлемой точностью) только для некоторых типов *процессоров* и *программ*
- На практике для оценки WCET применяется сочетание формальных методов и измерений
- Система реального времени должна быть устойчива к отдельным/локальным превышениям имеющихся оценок WCET
 - сторожевые таймеры для обнаружения превышения WCET
 - устойчивость к сбросу задачи-нарушителя (и пропуску итерации её выполнения)
 - «растяжимые» периоды выполнения задач для динамической регулировки загрузки процессора

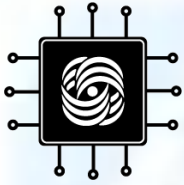
Спасибо за внимание!



ИНФОРМАЦИОННО-УПРАВЛЯЮЩИЕ СИСТЕМЫ РЕАЛЬНОГО ВРЕМЕНИ

Лекция 7: *Архитектура вычислительных блоков ИУС РВ*

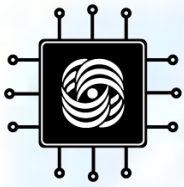
Кафедра АСВК,
Лаборатория Вычислительных Комплексов
Балашов В.В.



План

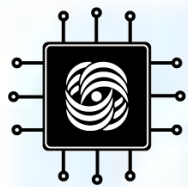
- Однопроцессорные вычислительные блоки
- Распределённые системы из однопроцессорных блоков
- Многомодульные вычислительные блоки
- Архитектура интегрированной модульной авионики

ИНТЕГРАЦИЯ



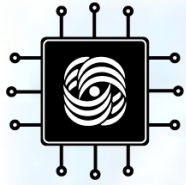
Необходимый минимализм: однопроцессорные блоки

- Технологические ограничения:
 - вынужденное применение «грубого» технологического процесса
 - жёсткие ограничения по размерам и энергопотреблению
- Откуда берутся ограничения
 - требование к устойчивости к внешним воздействиям (излучение и т.п.)
 - неразвитость технологических процессов производства микросхем
 - общий лимит на размеры и энергопотребление ИУС РВ



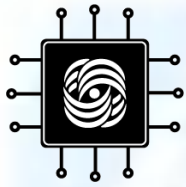
Аппараты с жёсткими ограничениями на ИУС РВ



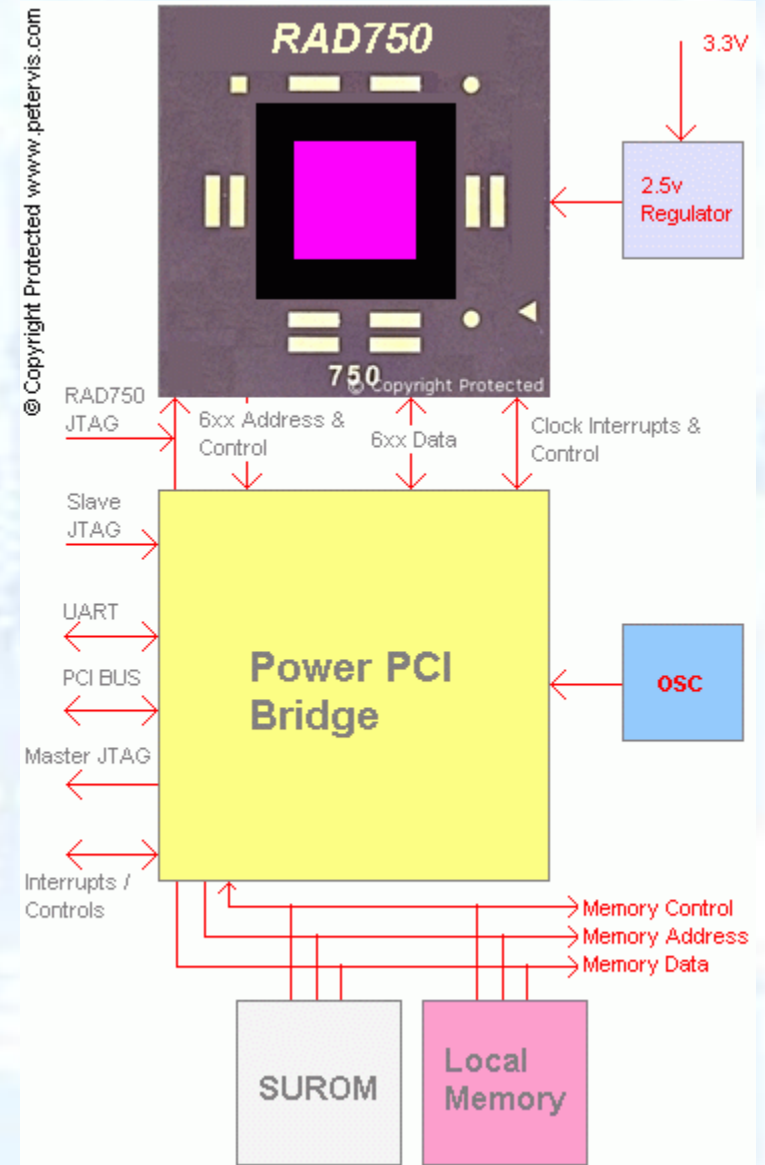


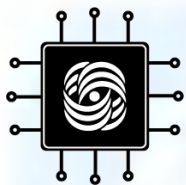
Центральные ЭВМ марсоходов

Аппарат	CPU	RAM	Flash	EEPROM	ОС
Sojourner (1997)	Intel 80C85, 2 МГц, 8-разрядный, 6000 транзисторов (аналог Intel 8080 1974 г. разработки)	512 Кбайт	176 Кбайт	Нет	Однозадачная, жёсткий порядок выполнения задач
Mars Exploration Rover (2004)	IBM/BAE RAD6000, 20 МГц, 32-разрядный, 1.1 млн транзисторов	128 Мбайт	256 Мбайт	3 Мбайт	VxWorks, многозадачная
Mars Scientific Laboratory (2011)	BAE RAD750, 132 МГц, 10.4 млн транзисторов	256 Мбайт	2 Гбайт	256 Кбайт	VxWorks, многозадачная

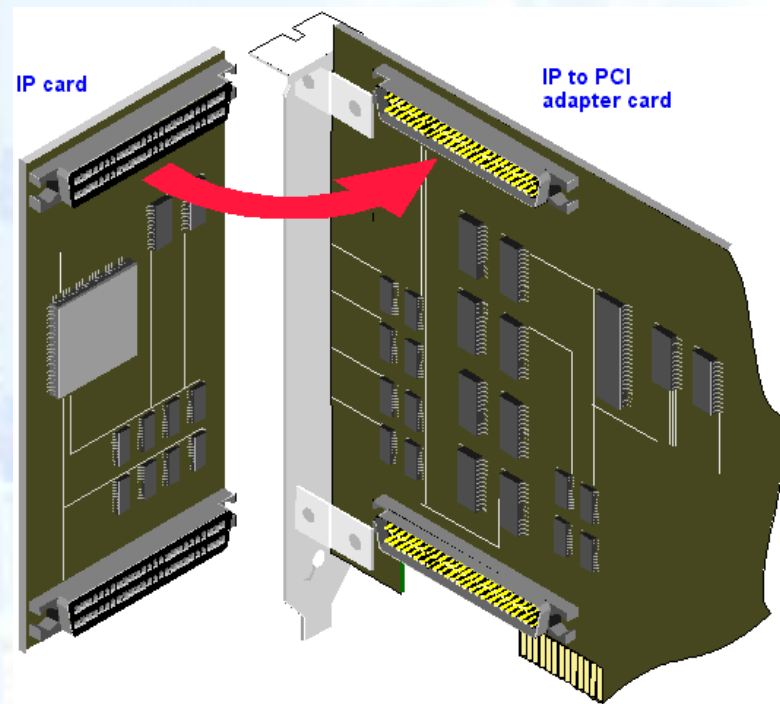
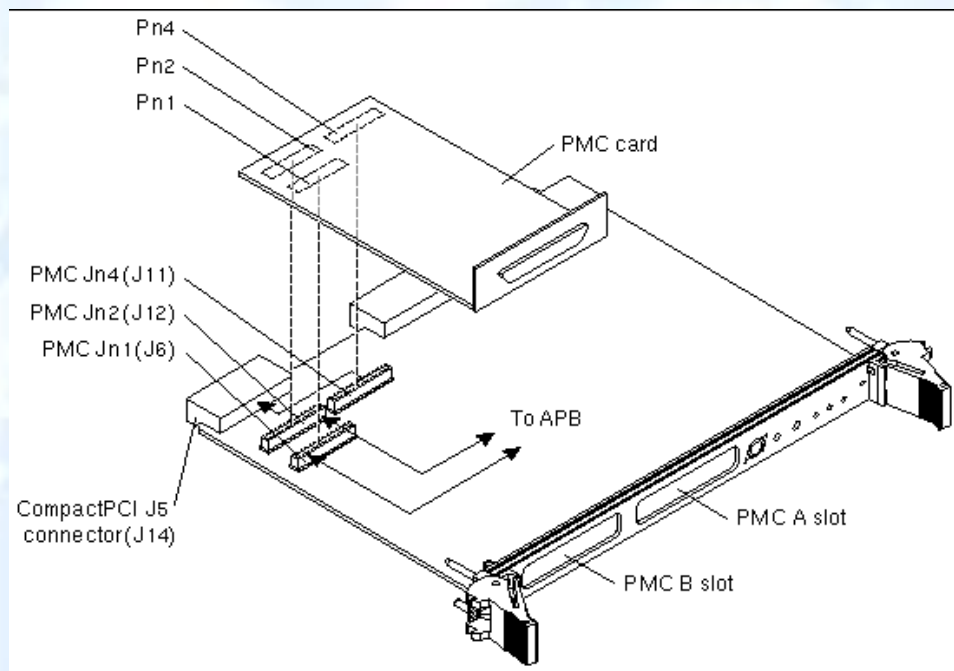


RAD750

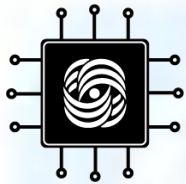




Расширение одноплатного компьютера: карта-мезонин



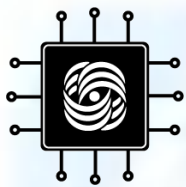
- На одноплатном компьютере – локальная шина (например, PCI)
- К локальной шине присоединяется карта расширения (например, адаптер канала)
- Карта расширения не зависит от «основного» разъёма, через который компьютер подключается к управляемой системе



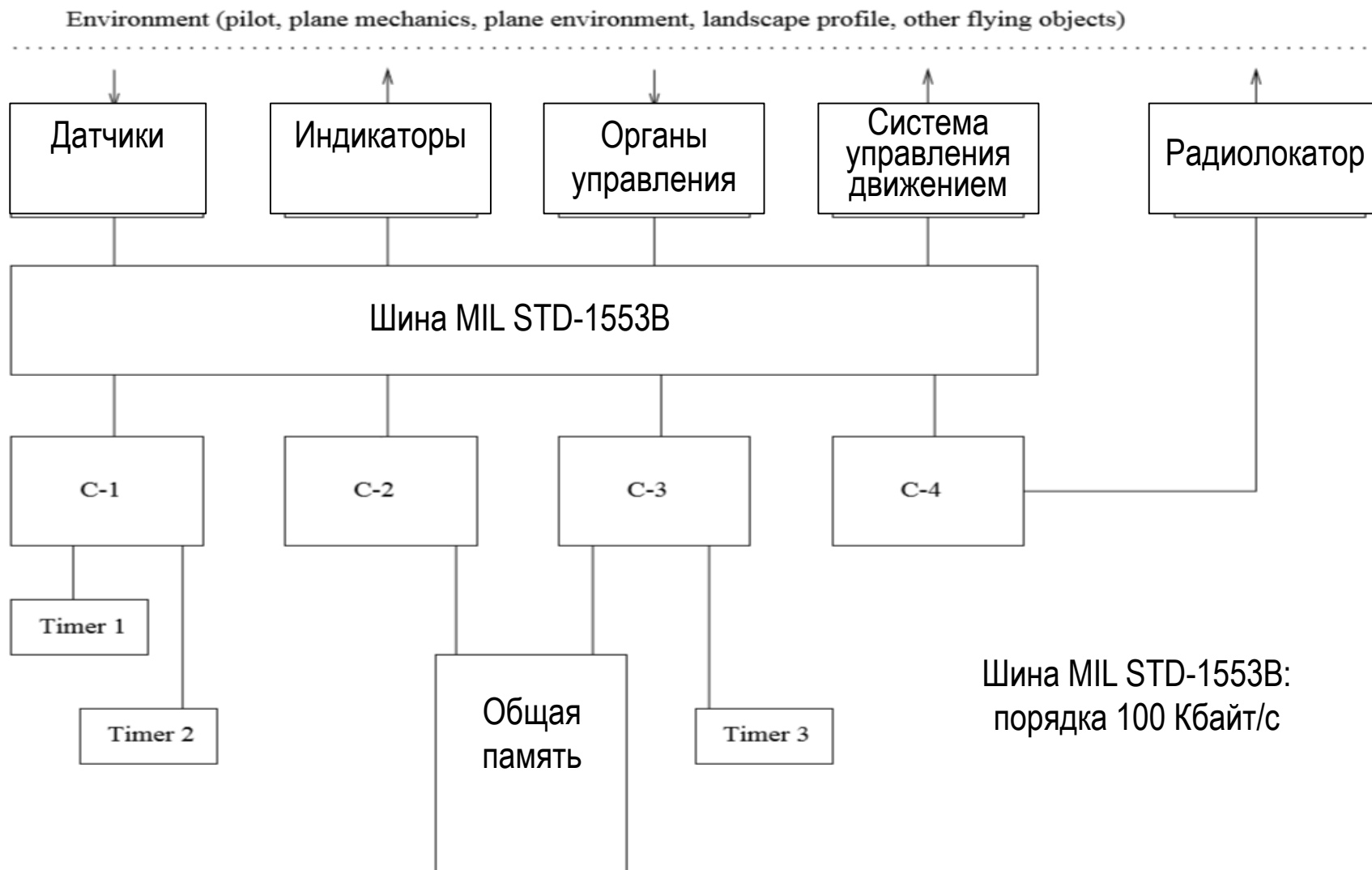
RAD750: внедрения

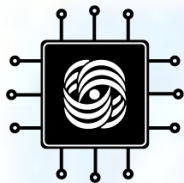
- Deep Impact (полёт к комете)
- Mars Reconnaissance Orbiter (спутник Марса)
- Lunar Reconnaissance Orbiter (спутник Луны)
- WorldView-1 (спутник оптической съёмки)
- Fermi Gamma-ray Space Telescope (орбитальный телескоп)
- Kepler space telescope (орбитальный телескоп)
- Wide-field Infrared Survey Explorer (орбитальный телескоп)
- Mars Scientific Laboratory (марсоход)





Слабая интеграция: однопроцессорные блоки на медленной шине





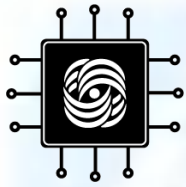
Назначение блоков

- C-1:
 - контроль состояния бортовых систем
 - выбор режима работы ИУС РВ
 - управление обменом по шине (контроллер канала)
- C-2:
 - вычисление управляющих параметров полёта для передачи в систему управления движением
 - подготовка данных для индикаторов
- C-3:
 - вычисление параметров движения самолёта на основе показаний датчиков
 - обеспечение движения самолёта по маршруту
 - управление датчиками
- C-4:
 - управление полётом на малой высоте по показаниям радара

Общая память

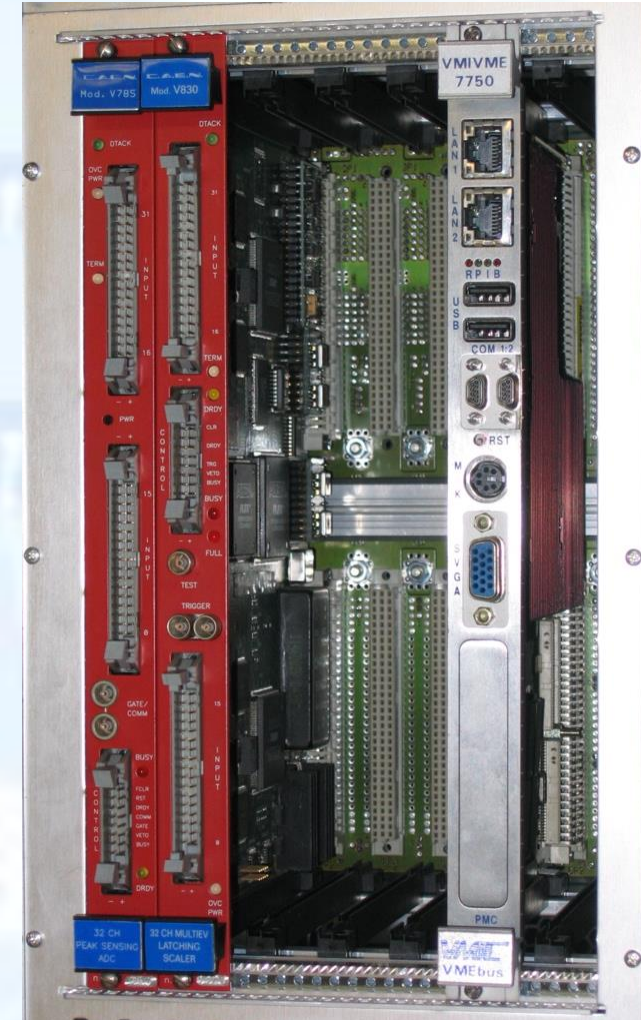


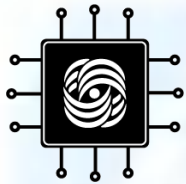
Прямой канал
к радару



Тесная интеграция: шина VME

- Параллельная шина с арбитражем
- Реализует прямой доступ к памяти модулей
- Объединяет модули в блок (крейт)
- Разрядность шины данных: 32 или 64 бита
- Пропускная способность:
 - 40 Мбайт/с (VME32)
 - 80 Мбайт/с (VME64)
 - до 320 Мбайт/с (VME64 в блочном режиме, на одну передачу адреса – несколько передач данных)



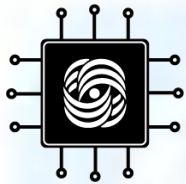


Структура шины VME

- Адресная шина
- Шина данных
- Шина арбитража (для управления доступом)
- Шина прерываний

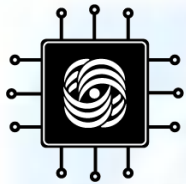
**Это похоже на
процессорную шину!**

**Это она и есть
(процессор Motorola
68000)**



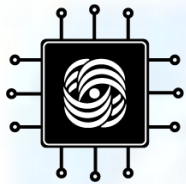
Роли модулей на шине VME

- Ведущий (Master)
 - Может инициировать передачу данных
- Подчинённый (Slave)
 - Отвечает на запросы от ведущего
- Источник прерывания (Interrupter)
 - Модуль, способный формировать прерывание (обычно – подчинённый)
- Обработчик прерывания (Interrupt handler)
 - Модуль, способный обрабатывать прерывания (как правило, одноплатный компьютер)
- Арбитр (Arbiter)
 - Модуль, управляющий доступом к шине и осуществляющий мониторинг обмена по шине. Устанавливается в слот №1



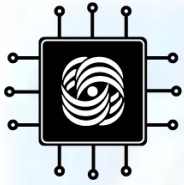
Процедура передачи данных

1. Ведущий устанавливает запрос на передачу данных на шине арбитража
 - при этом ведущий устанавливает на Ш.А. «свою» линию запроса в активное состояние (логический 0)
2. Шина освобождается от текущей передачи данных =>
 - арбитр определяет, какие линии запроса активны на Ш.А.
 - арбитр выбирает ведущего, которому отдать шину, и устанавливает в активное состояние линию Ш.А. «доступ дан» для этого ведущего



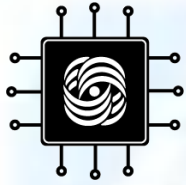
Процедура передачи данных

3. Получив доступ к шине, ведущий устанавливает:
 - на шине данных: значения передаваемых данных (в случае отправки); разрядность данных – не больше разрядности шины данных
 - на шине адреса:
 - номер подчинённого устройства
 - адрес в памяти подчинённого устройства
 - разрядность передаваемых данных (8, 16, 32 бита; также 64 бита для VME64)
 - признак «чтение» или «запись»
4. Подчинённое устройство:
 - на Ш.А. признак «чтение» => устанавливает на шине данных значения данных заданной разрядности с заданного адреса своей памяти
 - на Ш.А. признак «запись» => записывает по заданному адресу своей памяти данные заданной разрядности с шины данных



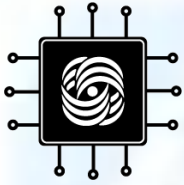
Прерывания

- Прерывание – способ для подчинённого устройства оповестить какое-либо из ведущих устройств о необходимости обмена данными
- Запрос прерывания выставляется на одной из 7 линий шины прерывания
- Ведущие устройства сами разбираются, кому адресован запрос



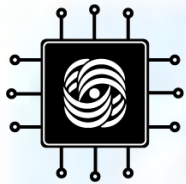
Блочная передача данных

- Поддерживается в VME64
- В начале обмена задаётся адрес и число передаваемых блоков данных
- Выполняется передача заданного числа блоков данных (каждый блок – не шире шины данных)
- => значительная экономия времени на задании адреса



Недостатки VME

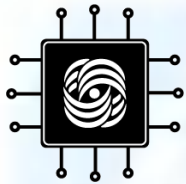
- Медленная шина (по современным меркам)
- Параллельная шина (много линий на материнской плате, сложность повышения частоты работы)
- Невозможно одновременное выполнение нескольких обменов данными



Борьба с недостатками VME: VPX

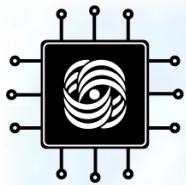
- Программа «VME Renaissance»: дать VME будущее в мире скоростных процессоров
- Обратная совместимость с VME: возможность установки в VPX-систему существующих VME-модулей
 - механическая совместимость
 - поддержка протокола и «идеологии» VME
- Уход от общей шины с арбитражем: поддержка коммутируемых сетей
 - Gigabit Ethernet, 10-Gigabit Ethernet
 - PCI Express
 - InfiniBand (высокоскоростной прямой доступ в память)

Высокоскоростные каналы из мира x86



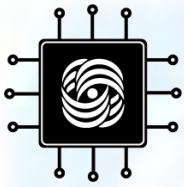
VPX: плавный уход от VME

- Современная VPX-система:
 - нет VME (т.к. нет унаследованных модулей)
 - x86 процессор
 - технологии из настольных систем (PCI Express, Gigabit Ethernet, USB, HDMI/DisplayPort)
 - межмодульное взаимодействие – по каналам Ethernet (дорожки на материнской плате крейта)
- Фактически – вычислительный кластер на технологиях «мира x86»
 - для создания системы нужно меньше экзотических знаний
- Модули в закрытых кожухах – можно вставлять и вынимать из работающей «в поле» системы



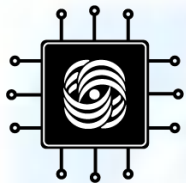
Интегрированная модульная авионика

- Недостаток традиционных многомодульных блоков: чрезмерная внутренняя интеграция
- VME: шина ограничена пределами блока
 - Единственный арбитр
 - Параллельная шина (сложно провести между блоками, в т.ч. защитить от помех)
 - Выход арбитра из строя – фатален (слот №1 – единственный)
 - Идеология прямого доступа к памяти: у модуля «слишком много» знания о внутреннем устройстве других модулей того же блока
- В итоге:
 - система неоднородна (блоки сильно различаются)
 - модули одного блока тесно интегрированы друг с другом
 - модули разных блоков слишком изолированы
 - низкая отказоустойчивость и реконфигурируемость (блок выходит из строя целиком)

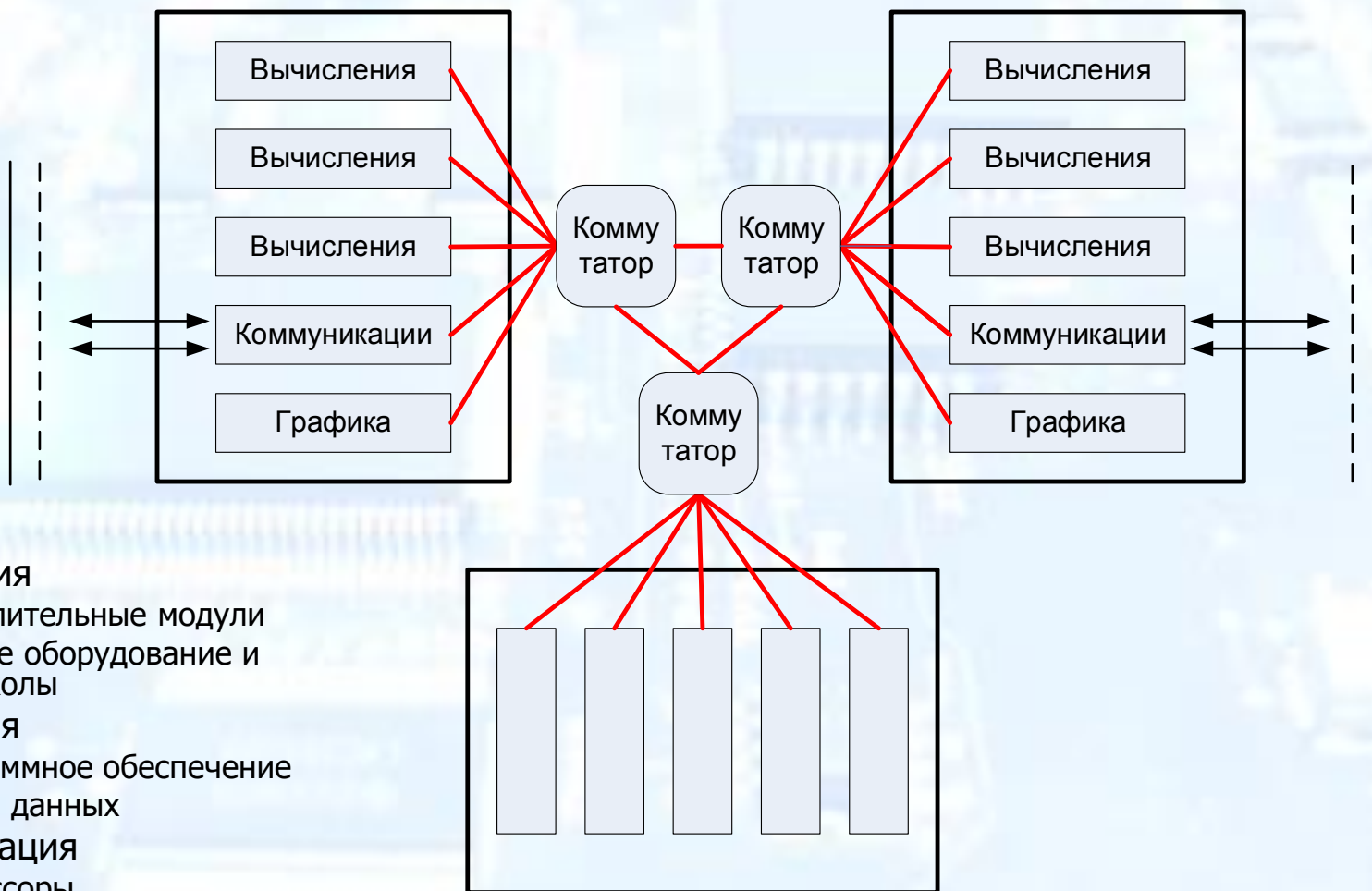


Интегрированная модульная авионика

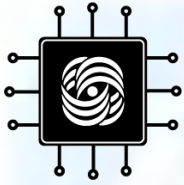
- Среда обмена данными на основе коммутатора
 - использование виртуальных каналов для разделения пропускной способности
 - ограниченные задержки при передаче сообщений
- Модули всех блоков равноправно подключены к среде обмена данными
=> система – «облако» модулей
- Сервисная шина – последовательная, относительно медленная (помехоустойчивость), может объединять все модули системы ИМА
 - CAN bus
- Трафик по сервисной шине минимален (низкоуровневые данные о состоянии, простейшие команды вроде вкл/выкл модуля)
- Высокая отказоустойчивость и реконфигурируемость
 - модуль вышел из строя => заменить его может модуль из другого блока
 - поддержка виртуальных каналов => возможность миграции вычислительных задач



Интегрированная модульная авионика (5 поколение)

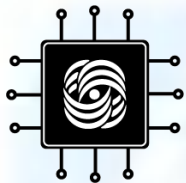


- Унификация
 - вычислительные модули
 - сетевое оборудование и протоколы
- Интеграция
 - программное обеспечение
 - потоки данных
- Виртуализация
 - процессоры
 - память
 - сеть



Сервисная шина CAN

- Абонент CAN на модуле ИМА – отдельная подсистема
 - включается до активизации основной логики модуля
 - продолжает функционировать при сбое основной логики модуля
- Короткие сообщения (до 8 байт данных)
 - периодическая выдача модулями статусных сообщений
 - запрос статуса модуля + ответ на запрос
 - команды вкл/выкл основной логики модуля
- Арбитраж
 - каждый абонент «слышит» передачу данных по каналу
 - нет передачи данных => можно начинать отправку своих данных; иначе ждать освобождения канала
 - при одновременном начале выдачи:
 - доминантные и рецессивные биты
 - сообщение начинается с адреса абонента-отправителя
 - абонент с «меньшим» адресом слышит в сети не то, что выдал, и прекращает выдачу
- Как ограничить задержку доставки данных?
 - ИМА: низкая загрузка канала, нет проблемы
 - в следующей лекции: CAN с централизованным управлением



Примеры модулей ИМА

Модуль процессора данных

Функциональный модуль общего назначения

Задачи:

- обработка данных,
- выполнение вычислительных задач,
- принятие управленческих решений.

Модуль ввода-вывода

Функциональный модуль специального назначения

Задачи:

- прием/выдача сигналов по «унаследованным» бортовым интерфейсам,
- преобразование унаследованных форматов сообщений в стандартный формат ARINC 653.

Модуль графического контроллера

Функциональный модуль специального назначения

Задачи:

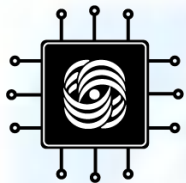
- построение изображений на основе данных, полученных от вычислительных задач,
- обработка входных видеоизображений,
- прием/выдача изображений по оптическим видеоканалам.

Модуль коммутатора FC

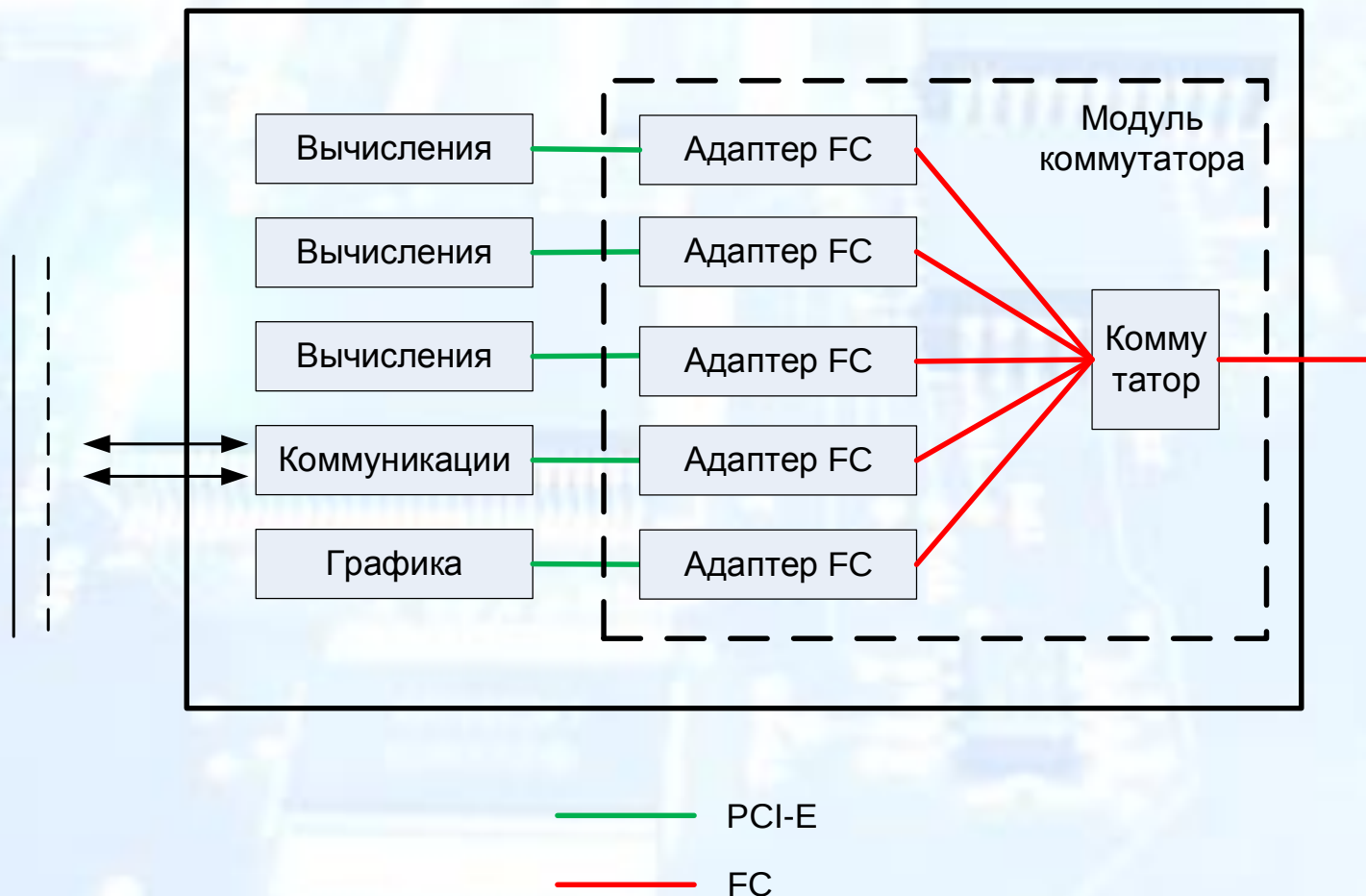
Предназначен для обеспечения взаимодействия в сети Fibre Channel

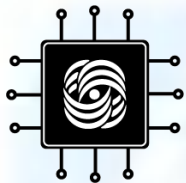
Модуль источника питания

Вспомогательный модуль, обеспечивающий вторичное питание модулей с требуемыми характеристиками



Пример применения шины PCI Express





Спасибо за внимание

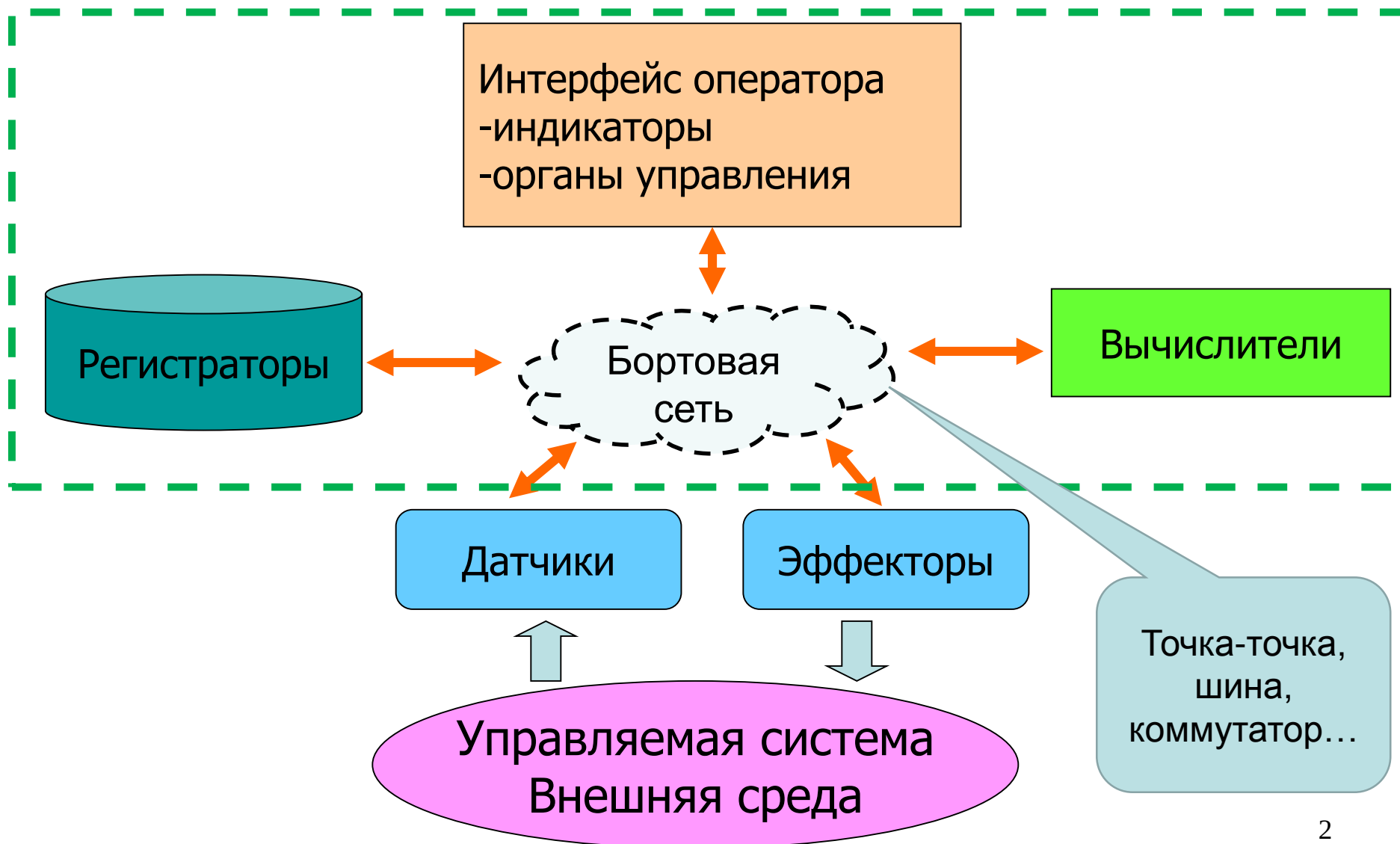
ИНФОРМАЦИОННО-УПРАВЛЯЮЩИЕ СИСТЕМЫ РЕАЛЬНОГО ВРЕМЕНИ

Лекция 8:

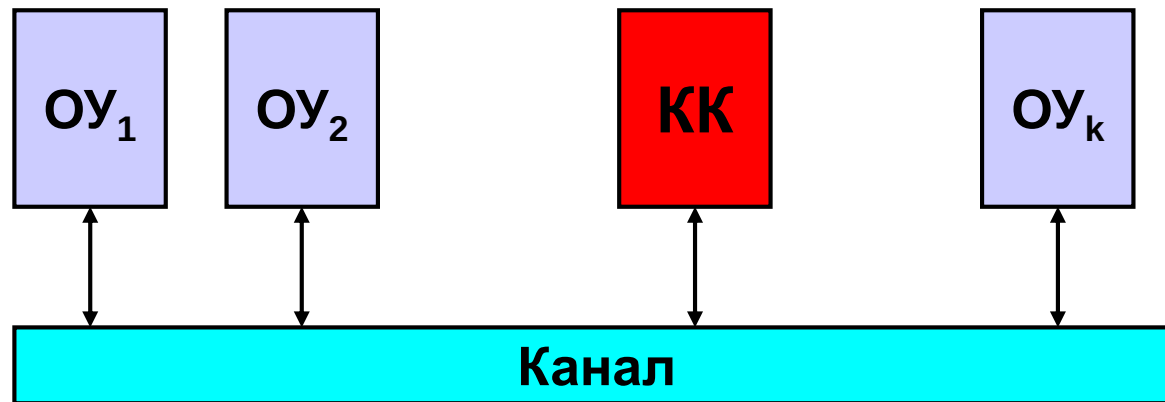
Каналы с централизованным управлением

Кафедра АСВК,
Лаборатория Вычислительных Комплексов
Балашов В.В.

Состав ИУС



Канал с централизованным управлением



- Контроллер – управляет обменами в соответствии с предварительно построенным расписанием обменов.
- Информация передается в виде сообщений.
- Обмен информацией осуществляется путем поочередной передачи данных по принципу "команда-ответ".

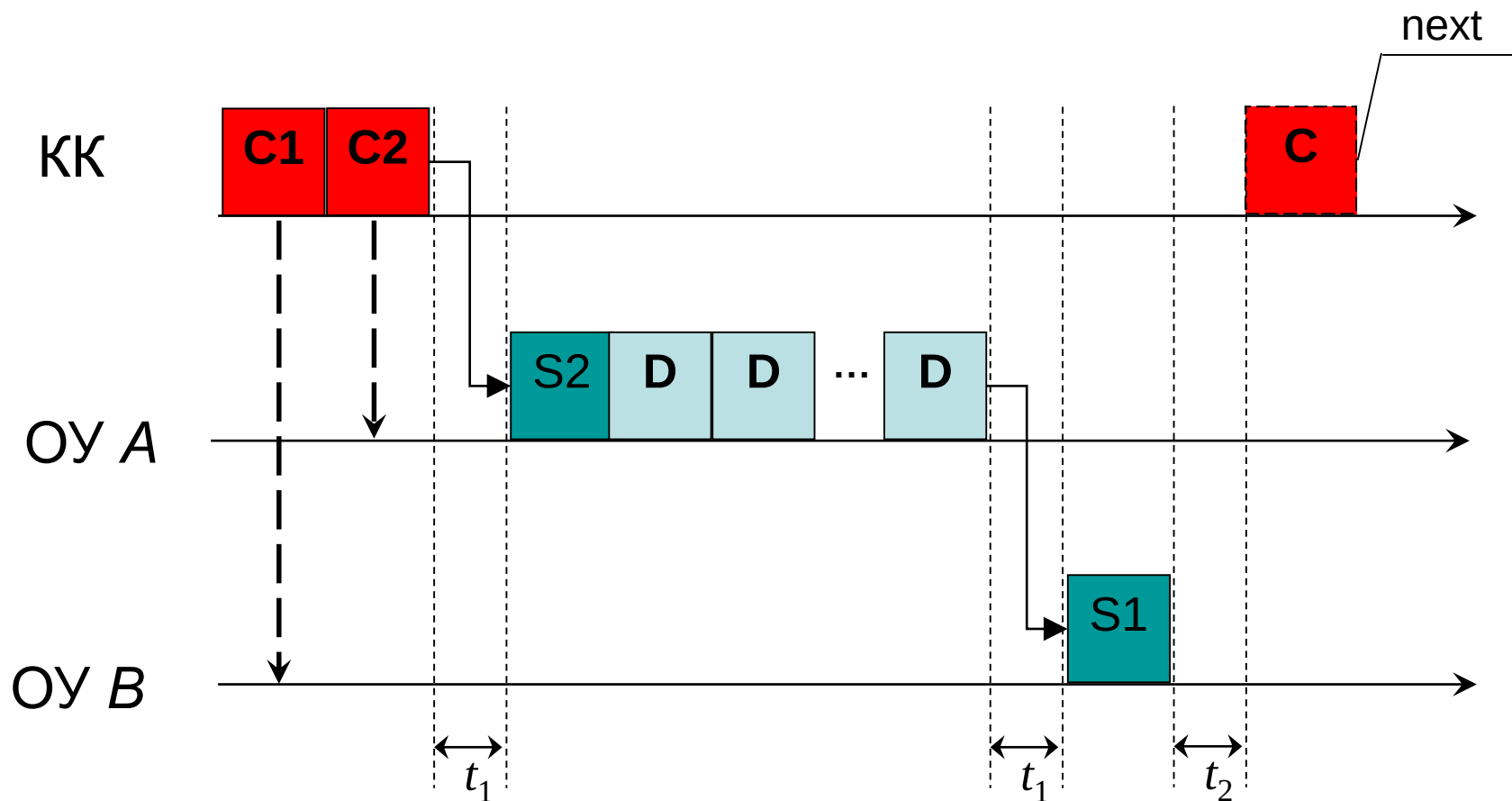
Преимущества каналов с централизованным управлением

- Отсутствие конфликтов.
- Гарантированная передача сообщений в режиме реального времени.
- Минимальное количество «проводов» в сети обмена.

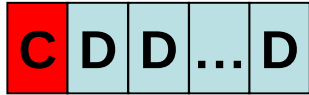
Промышленные каналы с централизованным управлением

- MIL STD-1553B
- Модификации MIL STD-1553B:
 - MIL STD-1773 (оптика)
 - Space Shuttle MIA bus (более длинное слово)
 - EBR-1553 (топология «звезда»)
- MIL STD-1760 (иерархия)
- STANAG 3910
- FC-AE-1553
- CAN bus с «искусственной» централизацией

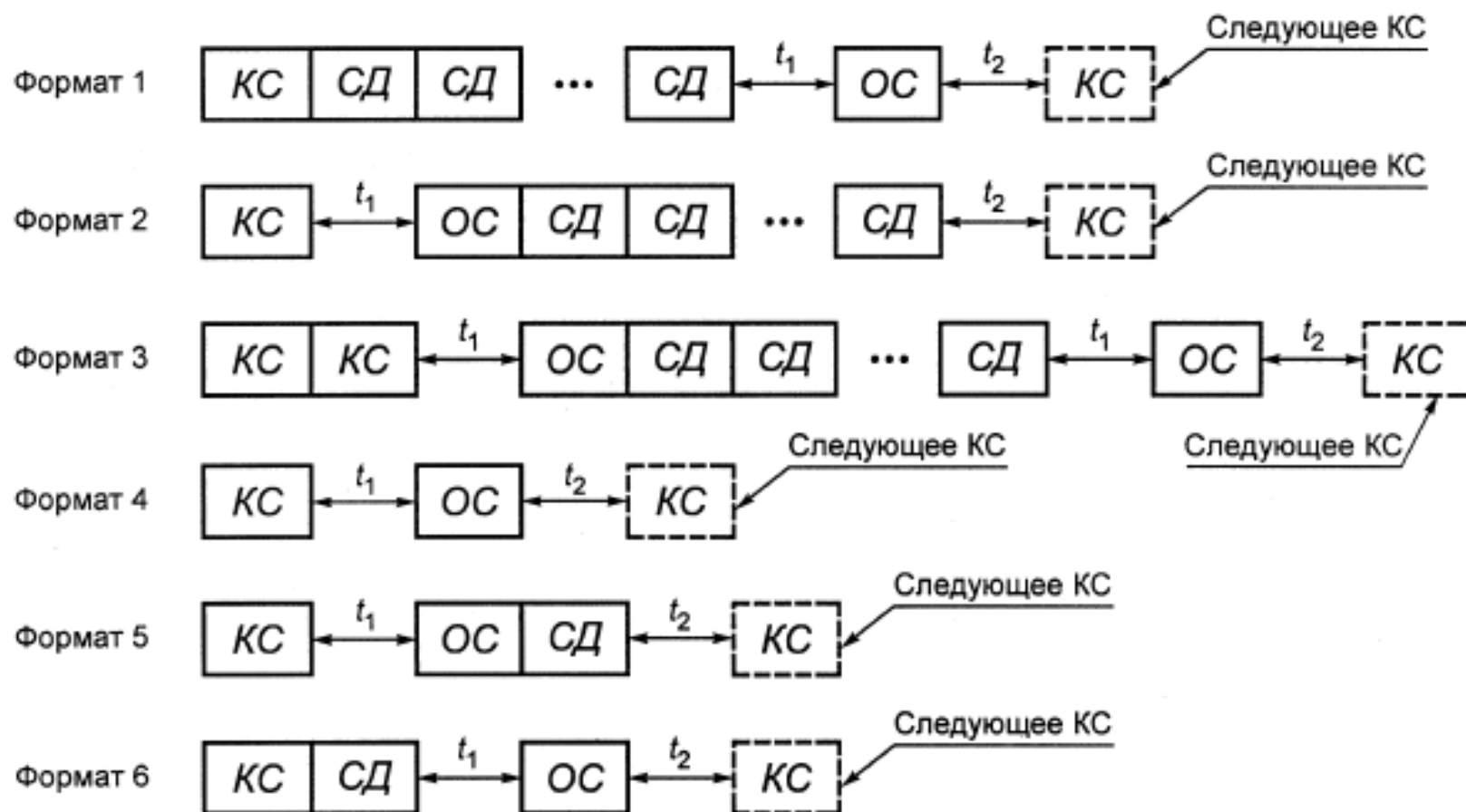
MIL STD-1553B: передача сообщения



MIL STD-1553B: форматы сообщений

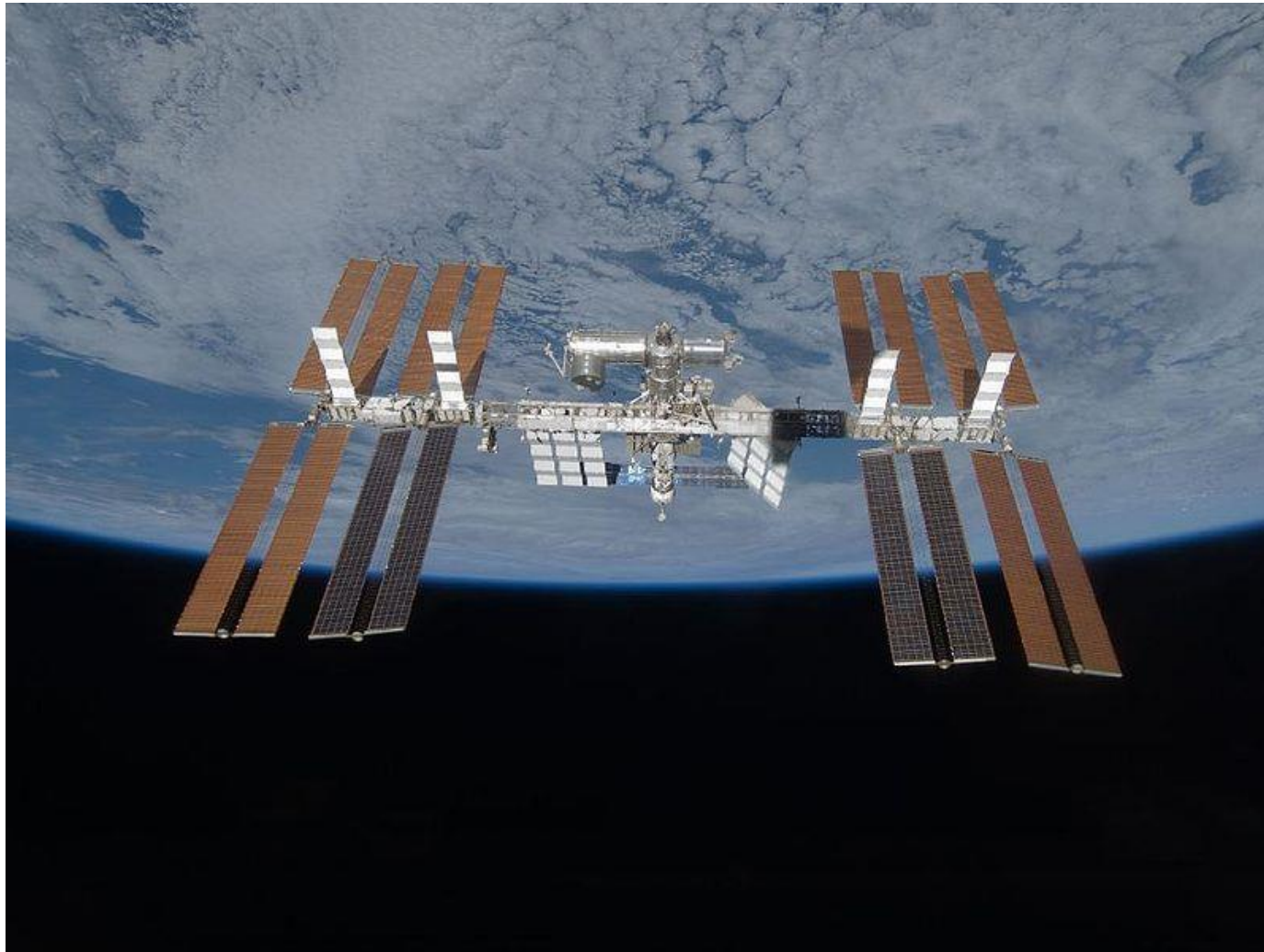
	передача данных	
точка-точка	КК-ОУ	
	ОУ-КК	
	ОУ-ОУ	
широко- вещательный	КК-ОУ	
	ОУ-ОУ	

То же, из ГОСТа

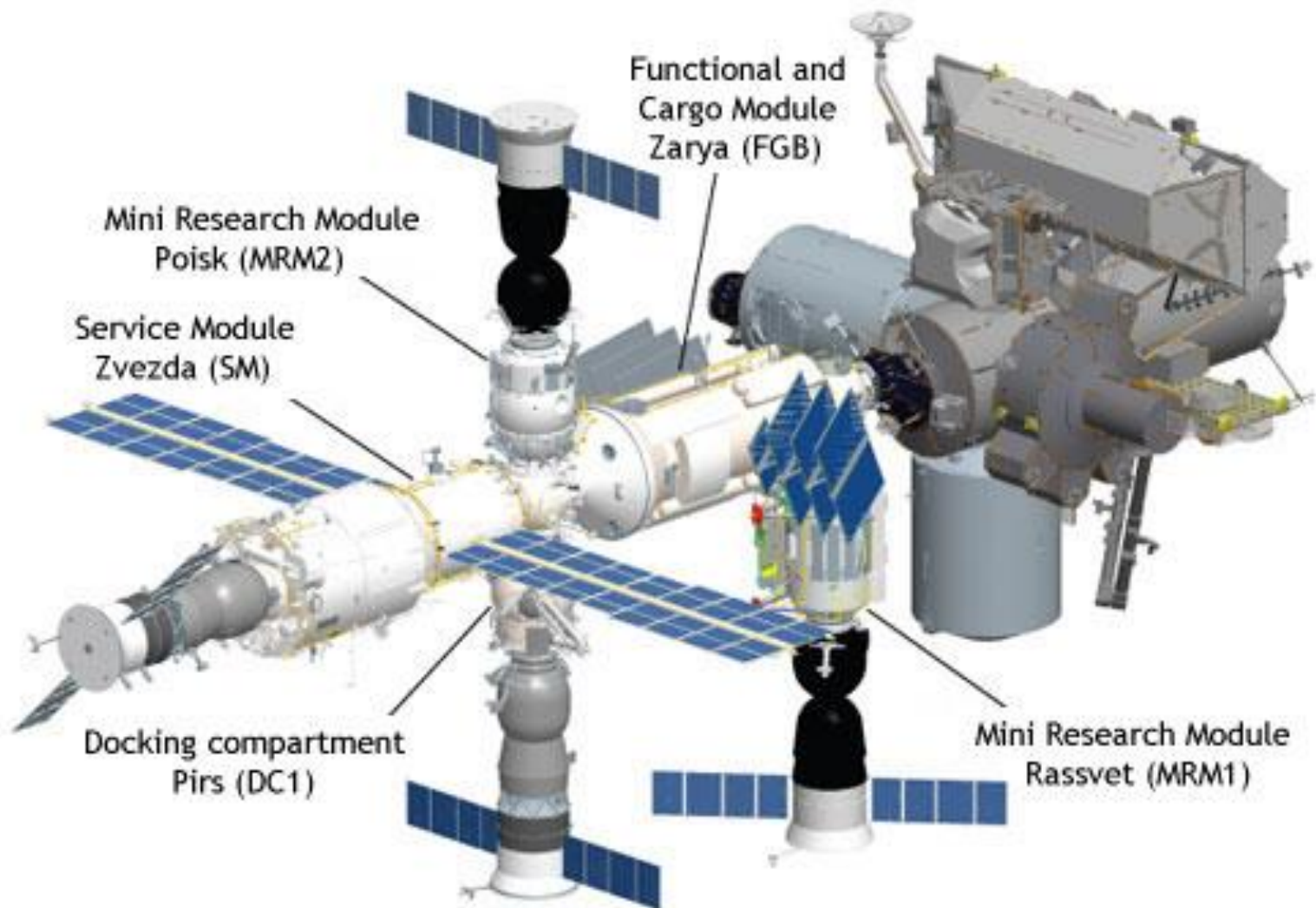


КС — командное слово; *СД* — слово данных; *ОС* — ответное слово; t_1 — пауза перед выдачей ответного слова;
 t_2 — пауза между сообщениями

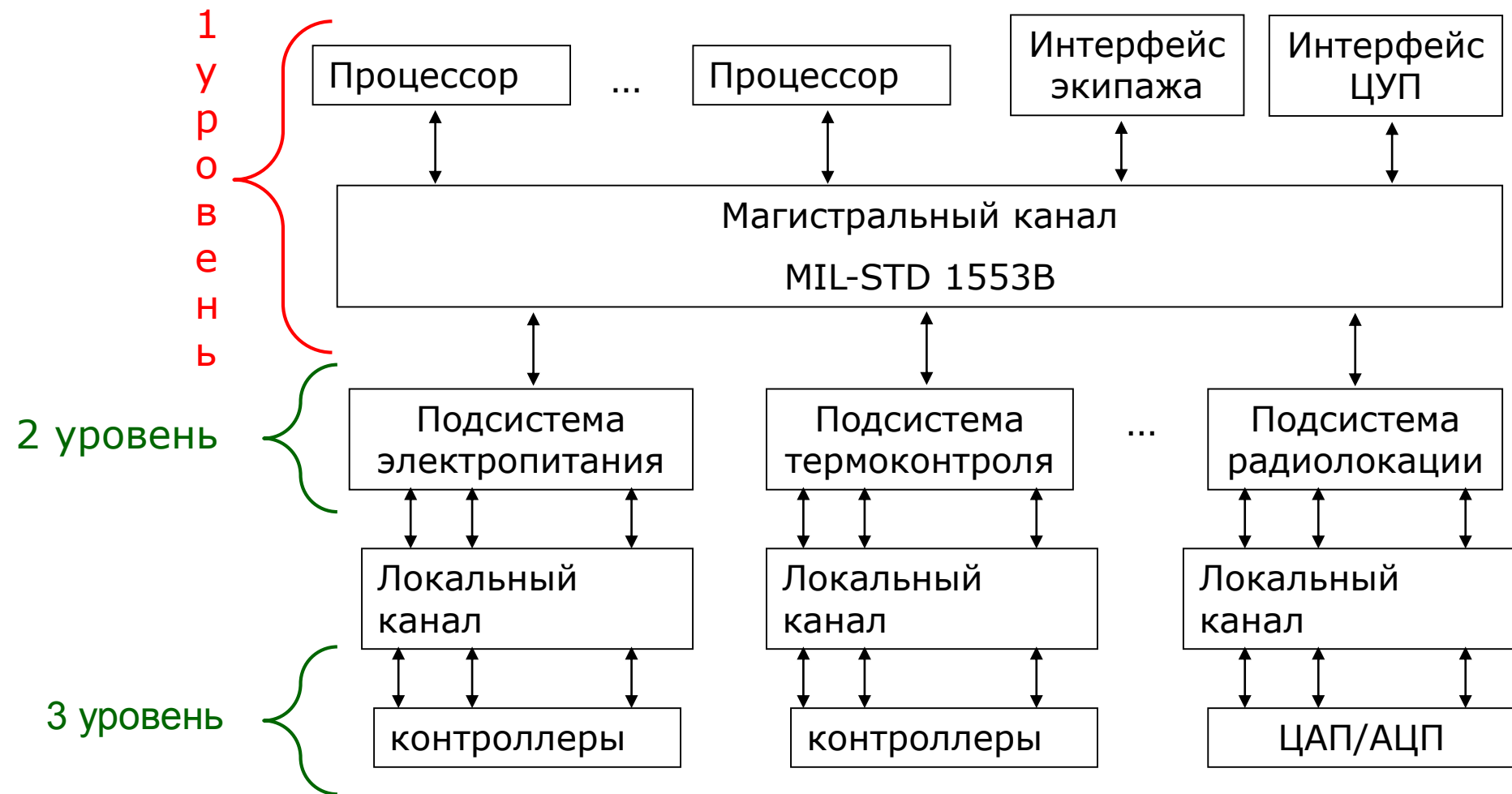
Международная космическая станция



Российский сегмент МКС



Иерархическая организация ИУС РВ МКС



Типичные режимы работы ИУС на МКС

- стандартный режим;
- режим микрогравитации для выполнения научных экспериментов;
- режим сближения и стыковки с транспортными кораблями;
- режим для выхода экипажа в открытый космос;
- режим выживания с отключением наименее важных экспериментов и систем;
- режим аварийного покидания экипажем МКС.

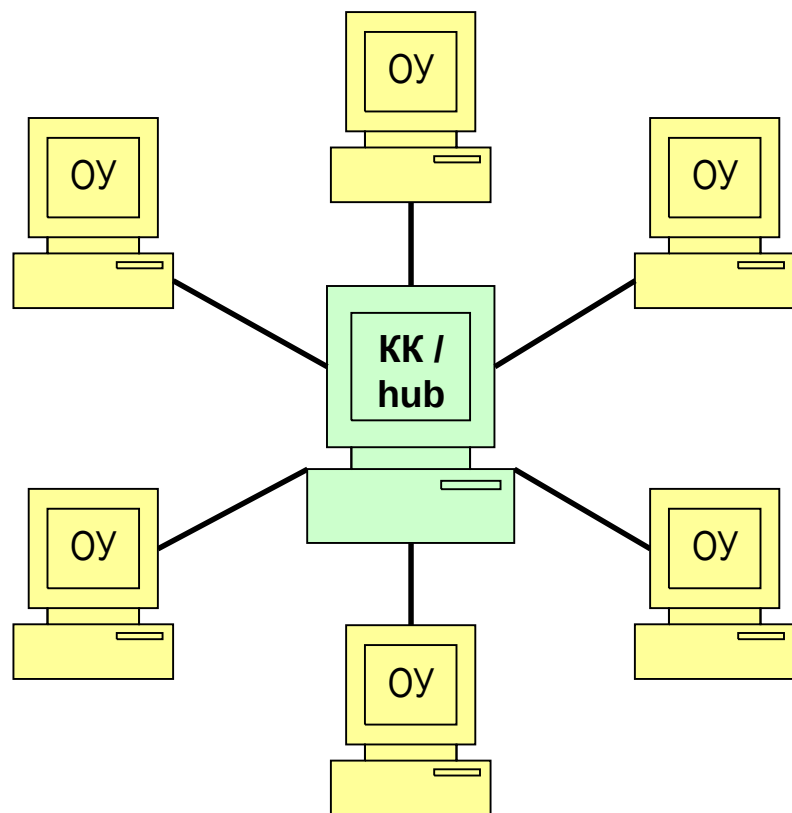
Режим → набор сообщений, расписание передачи

«Фокусы» с каналом MIL STD-1553B

- Ракета-носитель ARIANE 5:
обнаружение разделения
ступеней
(разрыв магистрали)
- Транспортный корабль ATV:
индикация успешной стыковки
(соединение периферийного
канала)

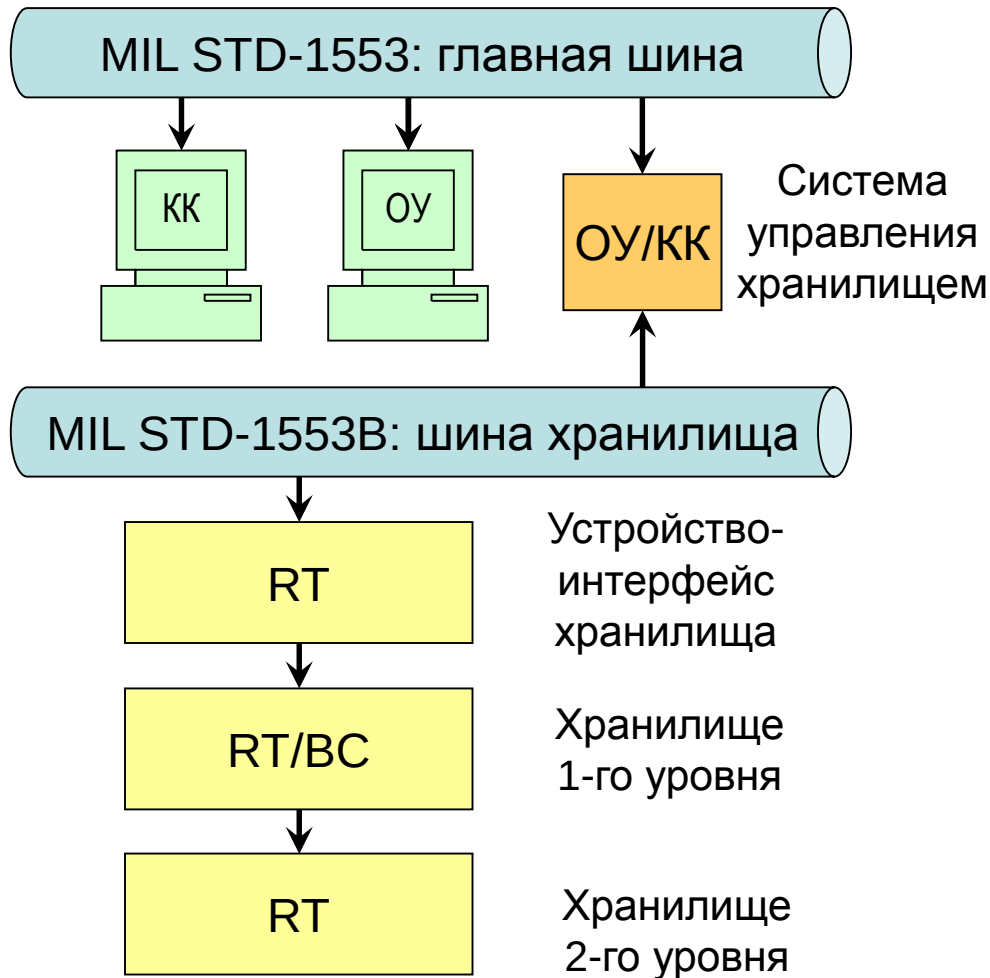


EBR-1553 (Enhanced Bit Rate)



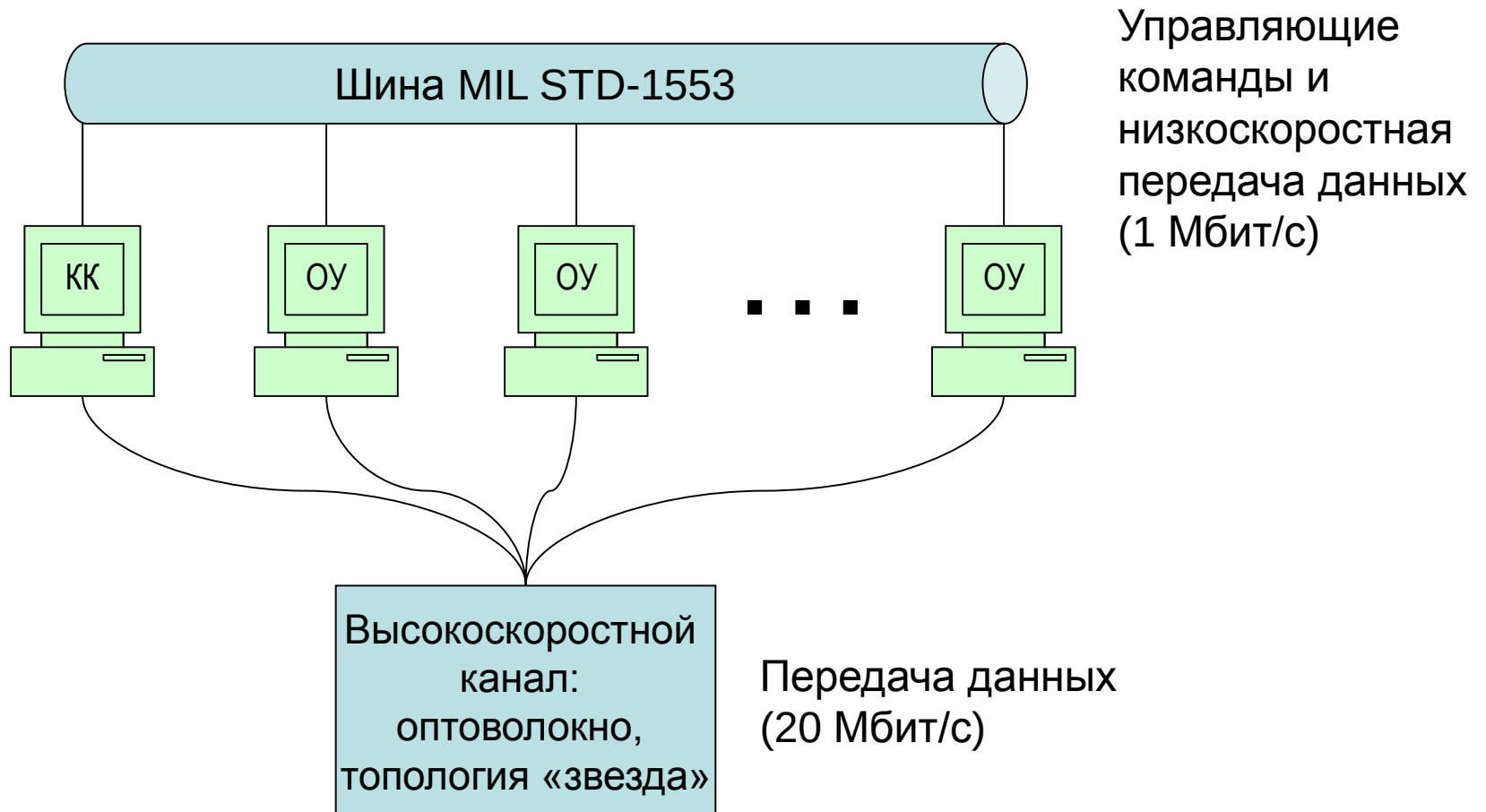
- Развитие MIL STD-1553B
- Новые возможности:
 - скорость 10 Мбит/с
 - топология «звезда», контроллер объединен с сетевым концентратором (hub)
- Нет обменов ОУ-ОУ

MIL STD-1760: иерархия

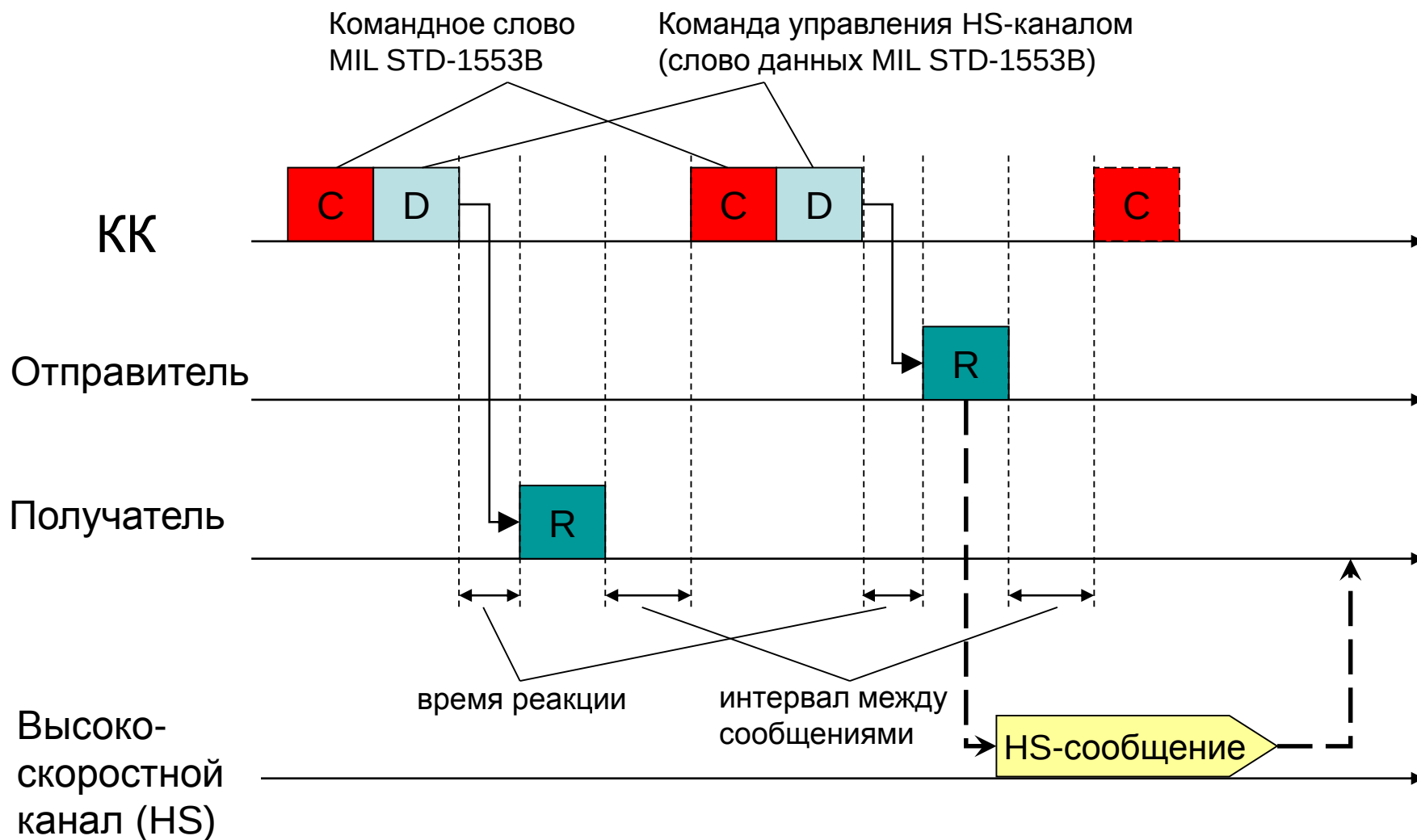


- Расширяет MIL STD-1553B для поддержки бортовых хранилищ данных
- Определяет стандартные форматы сообщений для функций управления хранилищами
- Акцент на надежности:
 - резервированная шина
 - контрольные суммы (CRC) в заголовках сообщений
 - специальные форматы сообщений для контроля целостности данных
 - подтверждение получения критических управляющих сообщений

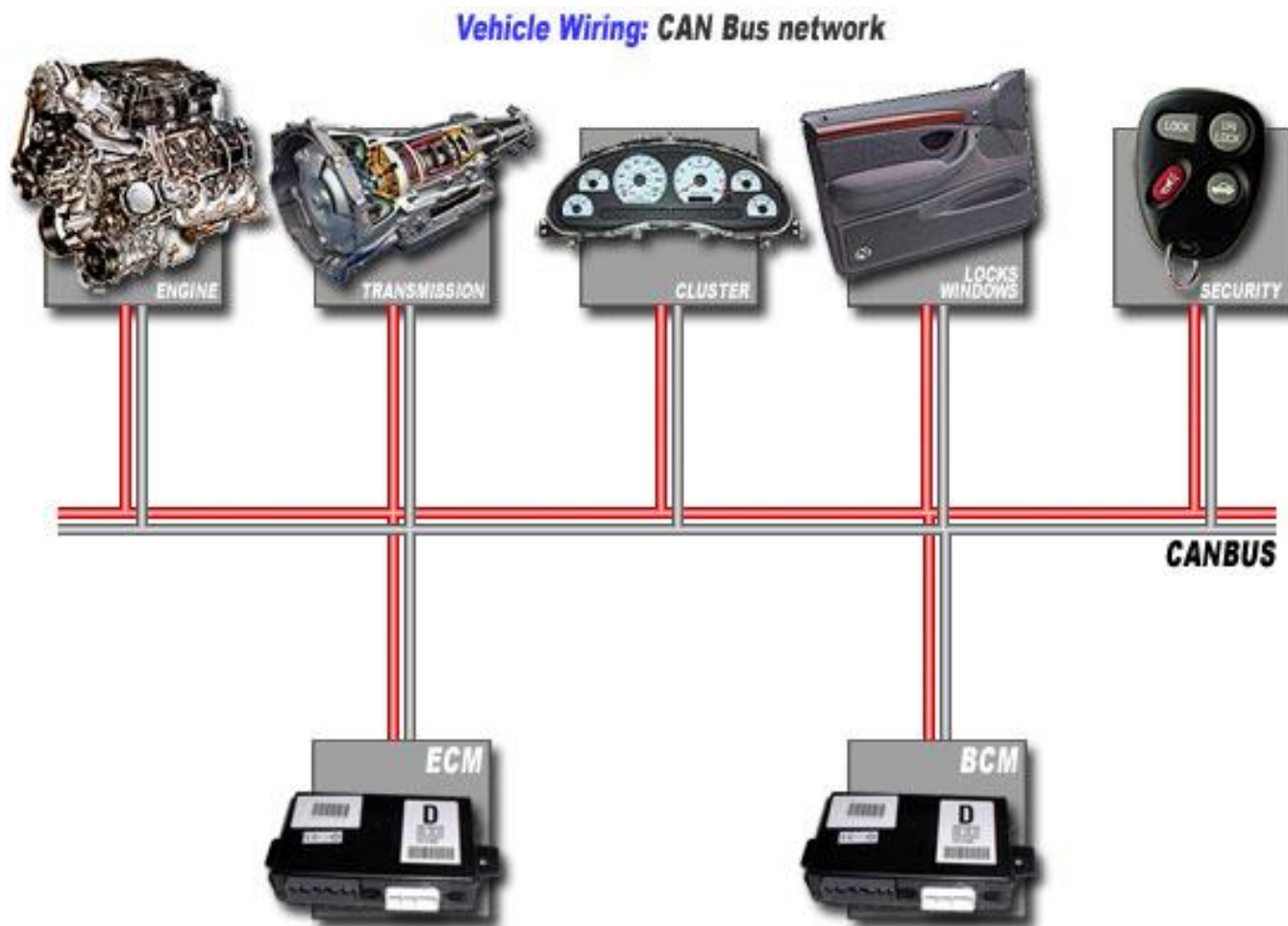
STANAG 3910: топология



STANAG 3910: обмен данными



Шина CAN

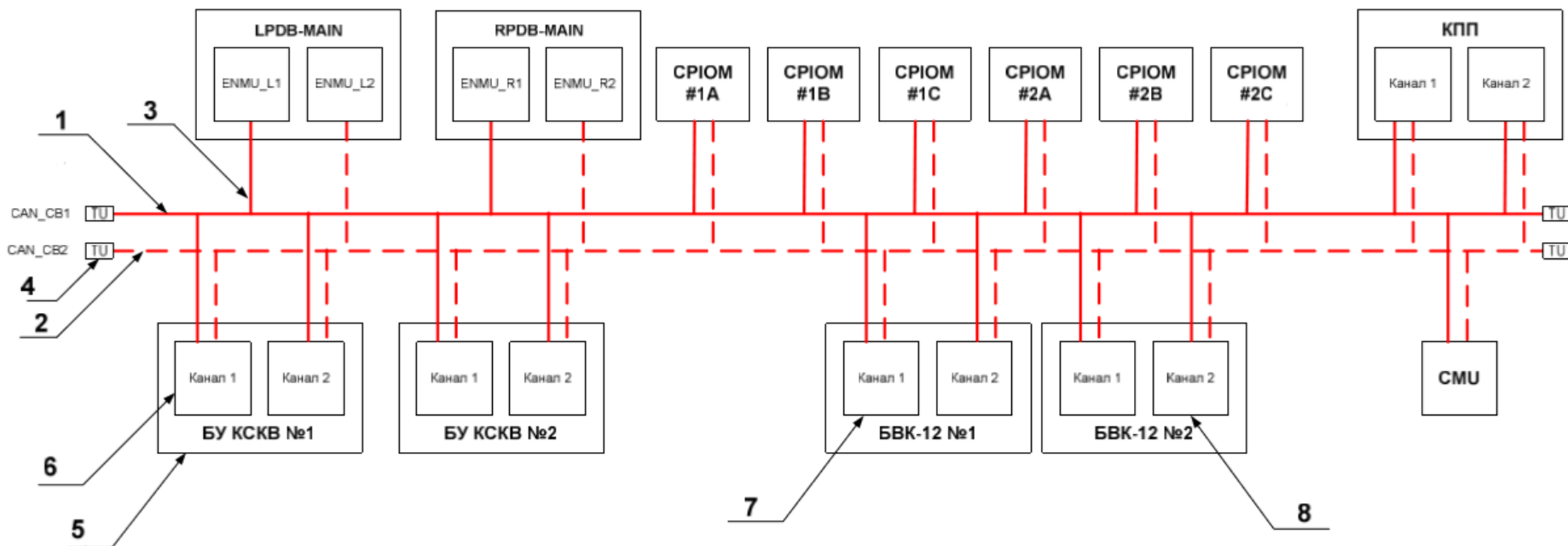


- служебная шина
- короткие сообщения

Шина CAN: арбитраж

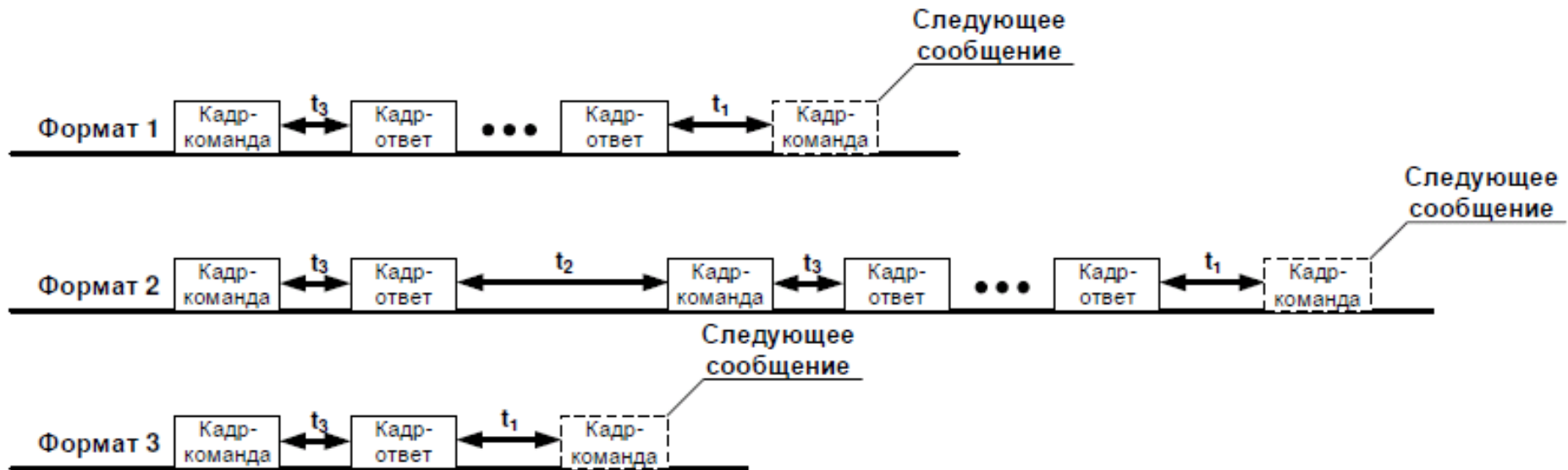
- Каждый абонент «слышит» передачу данных
- Нет передачи данных => можно начинать отправку своих данных
- Конфликт => абонент с «меньшим» адресом отступает
- Проблема: как ограничить задержку доставки данных?

CAN с централизованным управлением



- 1 – Основная магистральная шина
- 2 – Резервная магистральная шина
- 3 – Ответвитель основной магистральной шины
- 4 – Согласующий резистор
- 5 – Абонент интерфейса (БУ КСКВ №1)
- 6 – Устройство интерфейса
- 7 – Контроллер основной шины/Резервный контроллер резервной шины
- 8 – Контроллер резервной шины/Резервный контроллер основной шины

CAN с централизованным управлением



Математика...

Задача построения расписания выполнения работ в одноприборном устройстве

- Шина (канал) с централизованным управлением может рассматриваться как одноприборное устройство, обслуживающее исходно заданный набор работ без прерываний.
- Расписание выполнения работ, представляет собой упорядоченный набор

$$H = \left\{ s_j^* \right\}_{k=1}^{N_H}, j \in J$$

- $J = \{(t_j, s_j, f_j)\}$ - исходно заданный набор работ (длительности, директивные сроки)

Задача построения расписания выполнения работ в одноприборном устройстве

- Множество корректных расписаний H^* определим набором ограничений:

$$g_1 : (\forall j \in H) \Rightarrow ((s_j^* \geq s_j) \wedge (f_j^* \leq f_j))$$

$$g_2 : (\forall j \in H) \Rightarrow (f_j^* - s_j^* = t_j)$$

$$g_3 : (\forall (j, l) \in H, j \neq l) \Rightarrow (((s_j^* < s_l^*) \vee (s_j^* \geq f_l^*)) \wedge ((f_j^* \leq s_l^*) \vee (f_j^* > f_l^*)))$$

Задача построения расписания выполнения работ в одноприборном устройстве

- Задача:

$$\max_{H \in H^*} |H|$$

- известна в теории расписаний как задача о выборе максимального числа совместимых заявок и является NP –трудной.

Задача построения расписания выполнения работ в одноприборном устройстве

- Для частной задачи:

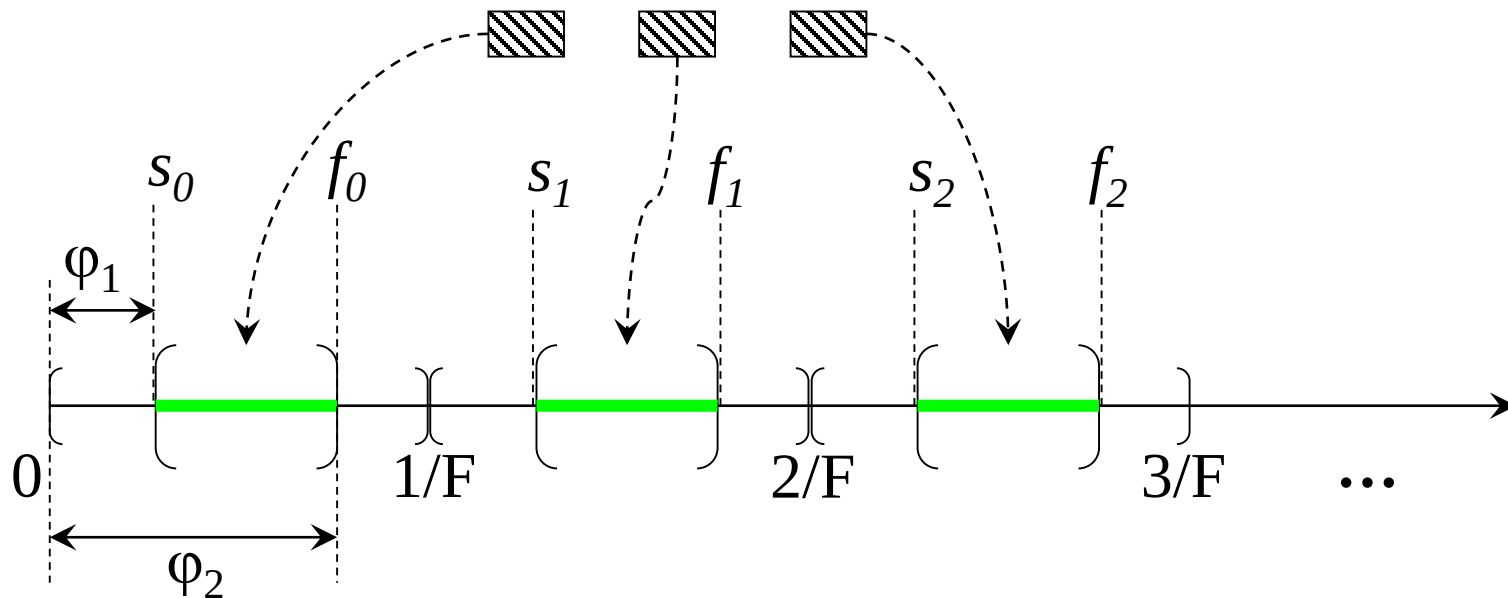
$$\max_{H \in H^*} |H|$$

$$\forall j: t_j = f_j - s_j$$

- известен оптимальный жадный алгоритм сложности $O(n \cdot \log n)$.

Задача построения расписания передачи сообщений по шине

- Исходный набор периодических сообщений преобразуется во множество работ J .
- Сообщение m : $(F, \varphi_1, \varphi_2, t)$.

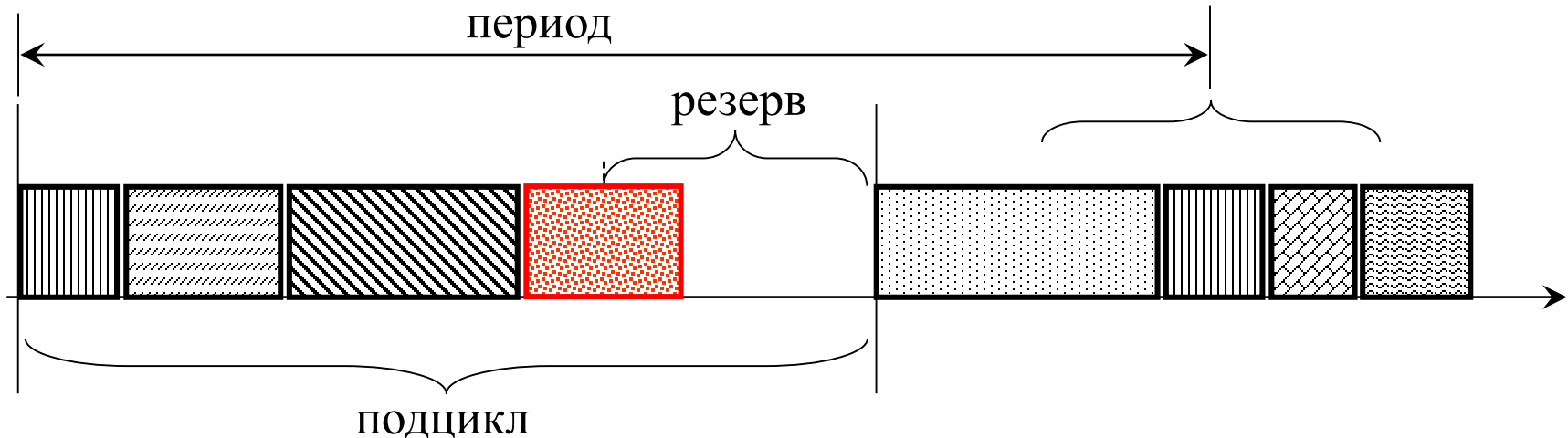


Задача построения расписания передачи сообщений по шине

Дано:

- Множество работ, которые должны выполняться на системе $J = \{j\}_{j=1}^{N_j}$
- Технологические ограничения на корректность расписания $g_i(H, \bar{x}) \leq 0, i = 4 \dots N_g$
- Вектор параметров технологических ограничений $\bar{x} = (x_1 \dots x_{N_x})$

Пример: технологические ограничения на расписание передачи сообщений по шине



- Одна цепочка работ в подцикле
- Резерв времени в конце подцикла
- Максимальная длительность цепочки работ
- Максимальное отклонение расстояния между последовательными работами одного сообщения от периода сообщения

Ограничения для схемы с подциклами

- g4* - в каждом подцикле может находиться не более одной цепочки работ.
- g5* - интервалы выполнения работ не должны пересекать границы подцикла.
- g6* - время начала цепочки работ относительно начала подцикла не должно быть меньше заданного значения.
- g7* - в конце подцикла должен быть зарезервирован интервал времени не меньше заданной длительности.
- g8* - число работ в цепочке не должно превышать заданного значения.

Ограничения для схемы без подциклов

- g_4 - число работ в цепочке не должно превышать заданного значения.
- g_5 - суммарная длительность выполнения работ цепочки не должна превышать заданного значения.
- g_6 - интервал времени между последовательными цепочками должен быть не меньше заданного значения.

Жадный алгоритм построения расписания

t : самое раннее время, на которое допустимо планировать работы;
 t монотонно возрастает.

Алгоритм:

Например, EDF

- 1) $t := 0$
- 2) выбрать, в соответствии с детерминированным критерием CR , работу, которая может быть начата в момент t без нарушения ограничений;
нет такой работы $\rightarrow t := \text{минимальное значение, для которого такая работа существует}$; перейти к шагу 5
- 3) поместить выбранную работу в расписание, с временем начала t
- 4) $t := \text{время завершения выбранной работы}$
- 5) учесть технологические ограничения
 - обновить значение t
 - обновить директивные интервалы оставшихся работ (начало $\geq t$)
- 6) директивные интервалы некоторых работ короче самих работ
 $\rightarrow$ внести эти работы в список *исключенных*
- 7) есть работы, не исключенные и не вошедшие в расписание
 $\rightarrow$ перейти к шагу 2

Недостатки жадного алгоритма

- Критерий выбора работ должен быть определен заранее
- Для каждого критерия (RM, EDF, ...) существуют наборы работ, на которых он дает плохой результат
→ ручной выбор критерия или перебор критериев
- Нет «обучения» по результатам неудачных запусков
→ если алгоритм неуспешен на конкретных данных, он всегда будет неуспешен на них при том же критерии CR

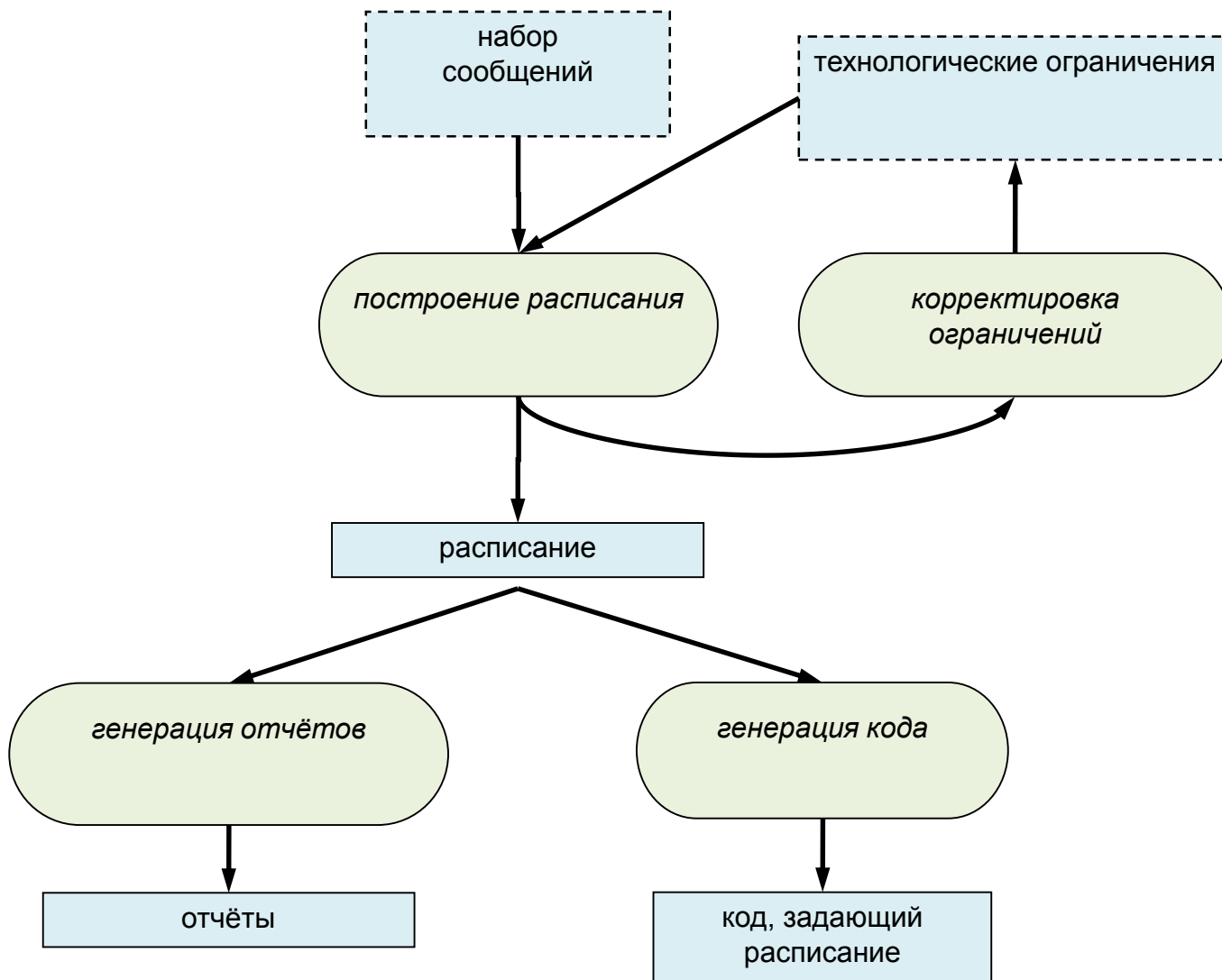
Возможное решение: адаптивный алгоритм с элементами недетерминизма (например, муравьиный)

САПР циклограмм

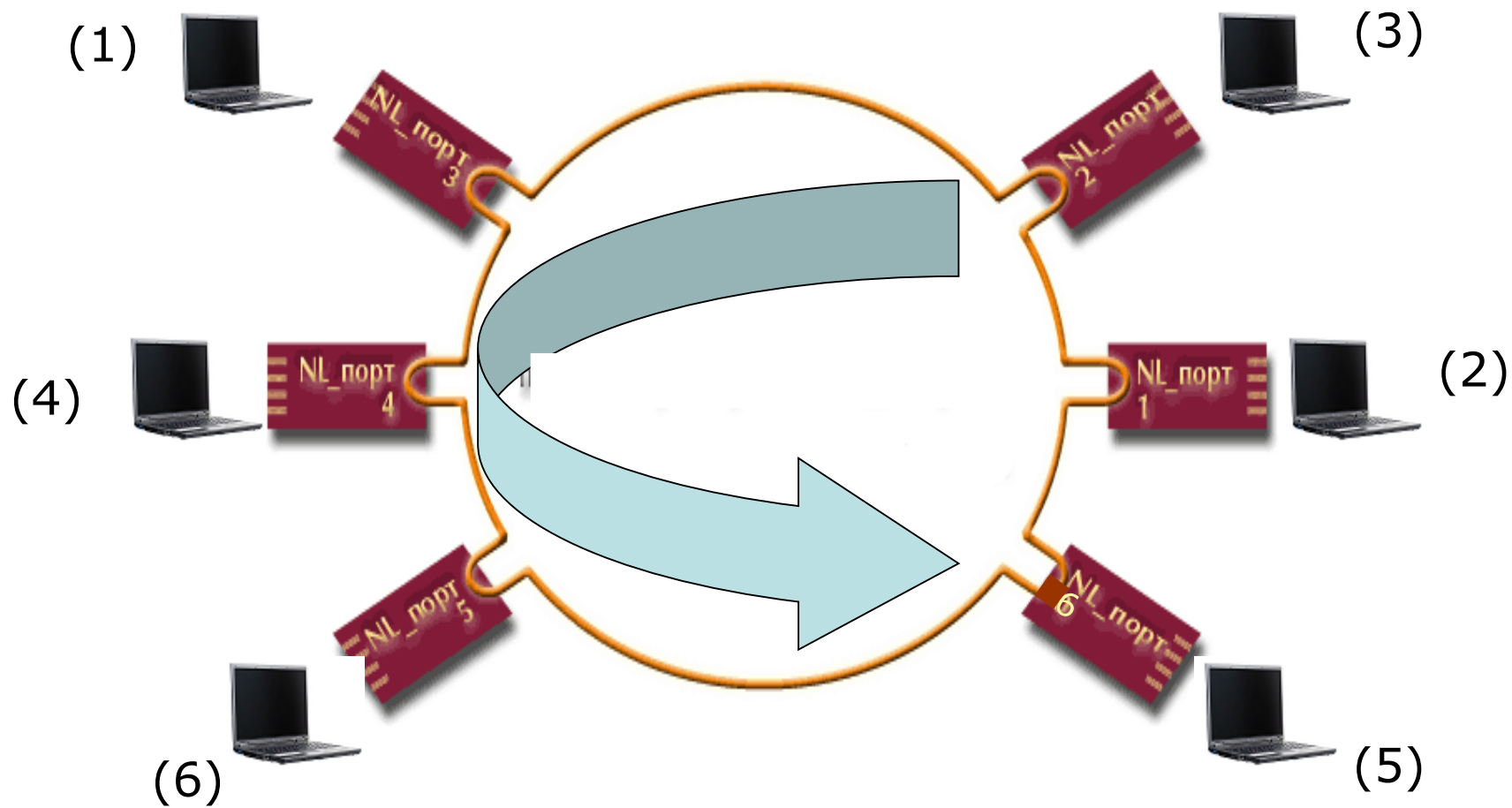
1. Создание проекта: наполнение базы данных информацией о структуре бортовой сети и протоколах информационного взаимодействия.
2. Автоматическое построение расписания.
3. Возможность ручной корректировки расписания.
4. В случае, если нельзя построить полное и корректное расписание: автоматическая корректировка значений ограничений на расписание.
5. Генерация программного кода, задающего расписание.
6. Генерация отчётов о входных данных и построенных расписаниях.

САПР циклограмм

(диаграмма технологического процесса)



Кольцо с арбитражем



Топологии и классы обслуживания стандарта Fibre Channel

- точка-точка (Point-to-Point),
- коммутируемая сеть (Switched Fabric),
- кольцо с арбитражем (Arbitrated Loop,).

Кольцо с арбитражем Fibre Channel (*классы обслуживания*)

- класс 1 – выделенное соединение с подтверждениями;
- класс 2 – передача без установки соединения с подтверждениями;
- класс 3 – передача без установки соединения без подтверждений.

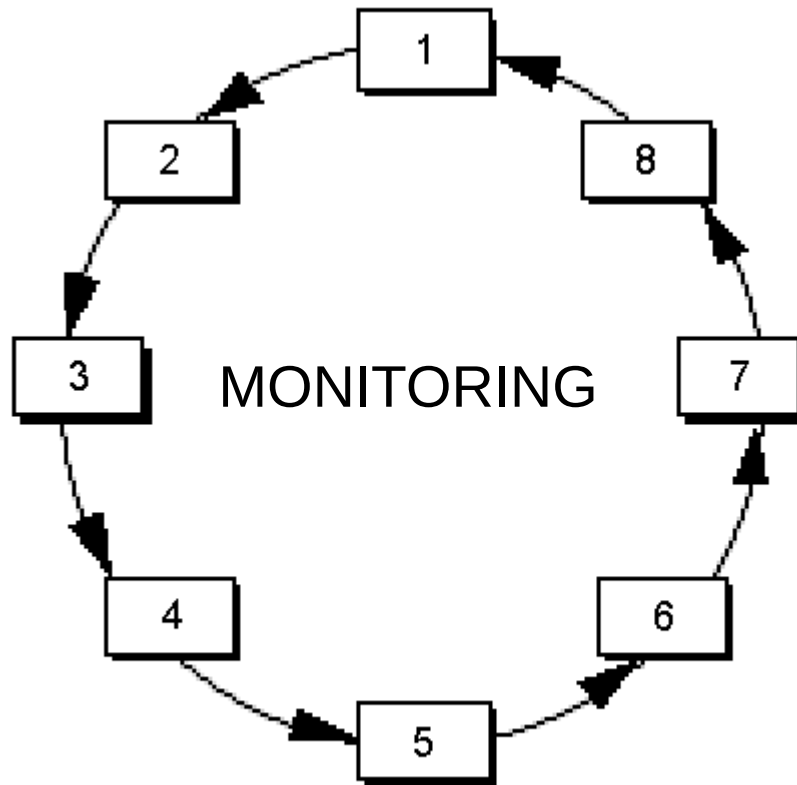
Описание работы кольца с арбитражем

- *MONITORING* - все принятые портом слова ретранслируются далее, т.е. порт передает в выходной канал принятый набор из 40 бит.
- *ARBITRATING* - порт переходит в это состояние, когда ему необходимо получить доступ к кольцу для передачи информации.
- *ARBITRATION WON* - состояние, в котором порт считается выигравшим арбитраж.
- *OPENED* - порт-приемник переходит в это состояние, когда он получает слово OPN с указанием своего адреса.

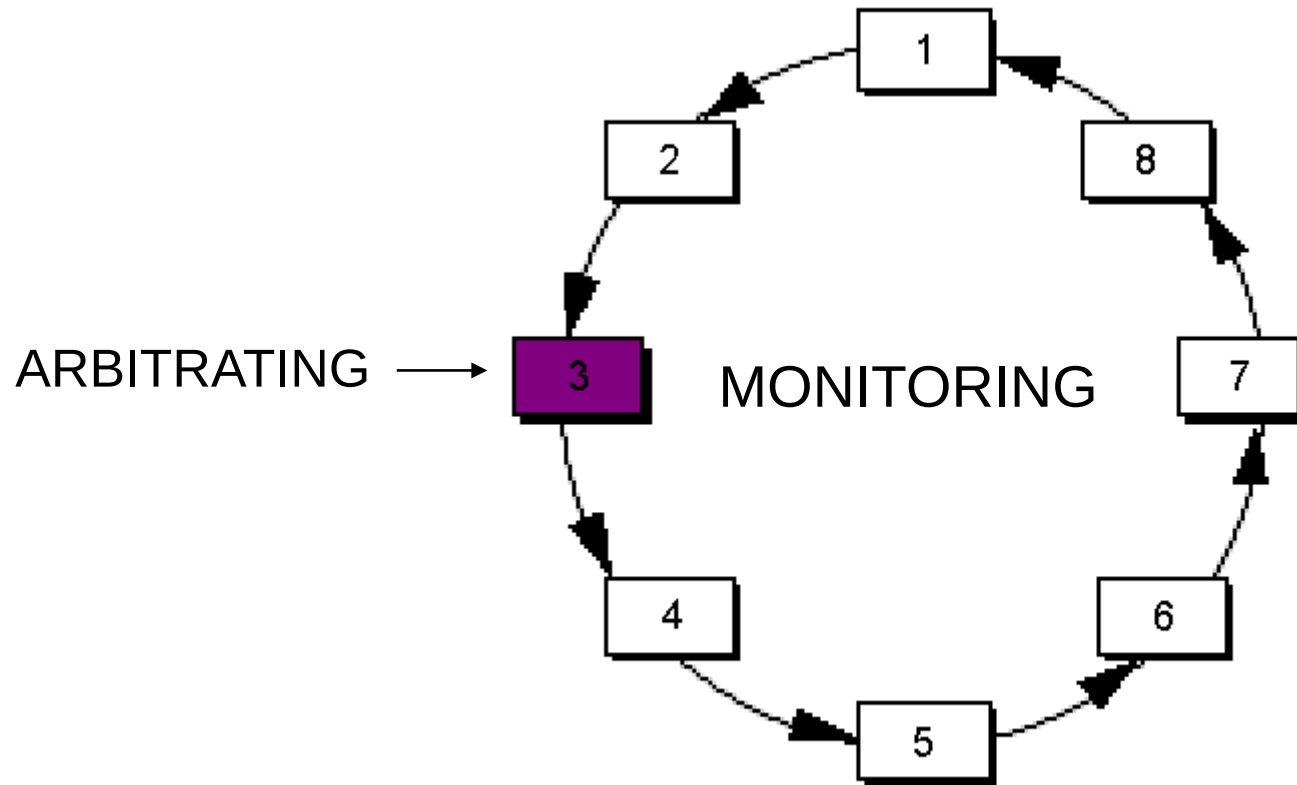
Описание работы кольца с арбитражем

- *OPEN* - порт начинает передавать кадр с данными.
- *XMITTED CLOSE* - порт переходит в это состояние, когда у него больше нет данных для передачи, и для закрытия портов-приемников он передал служебное слово CLS.
- *RECEIVED CLOSE* - порт-приемник переходит в это состояние, когда он получает служебное слово CLS. В этом состоянии порт ретранслирует слово CLS и переходит в состояние *MONITORING*.

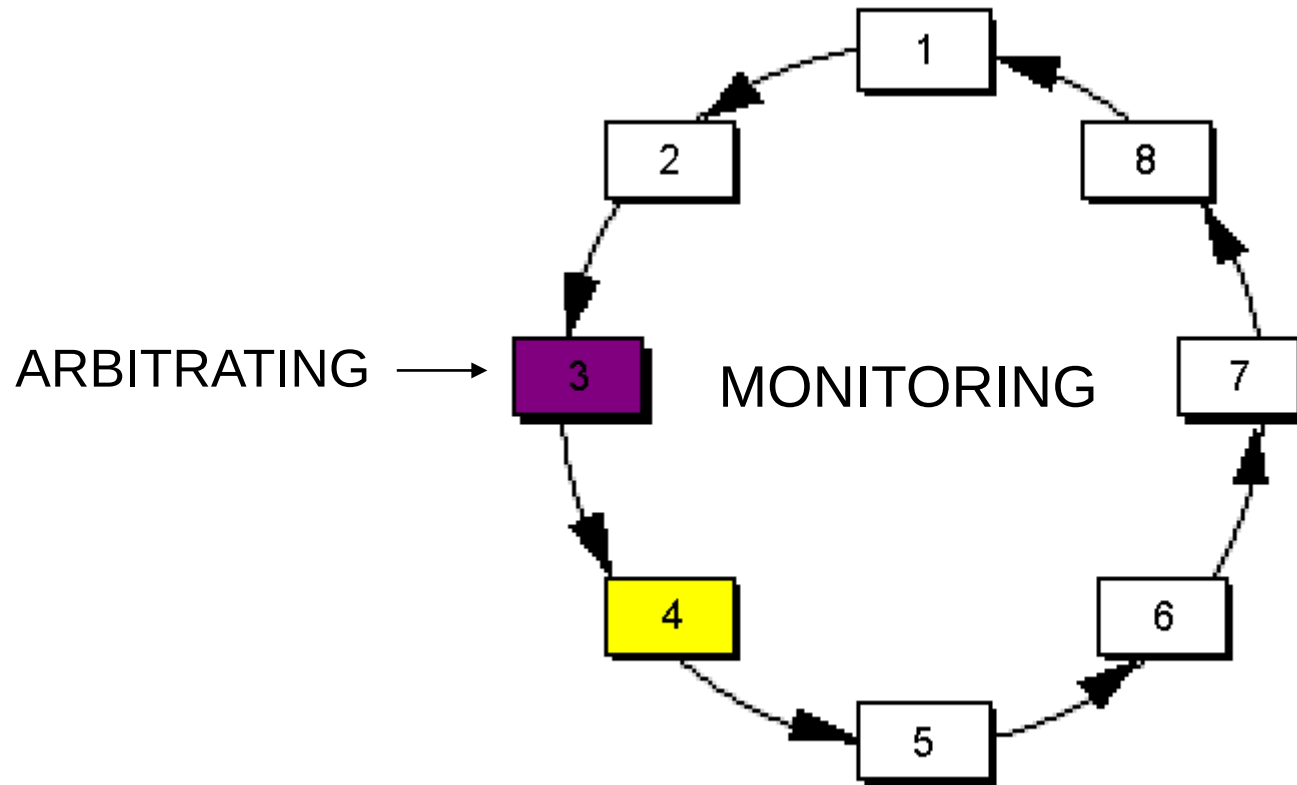
Передача сообщения



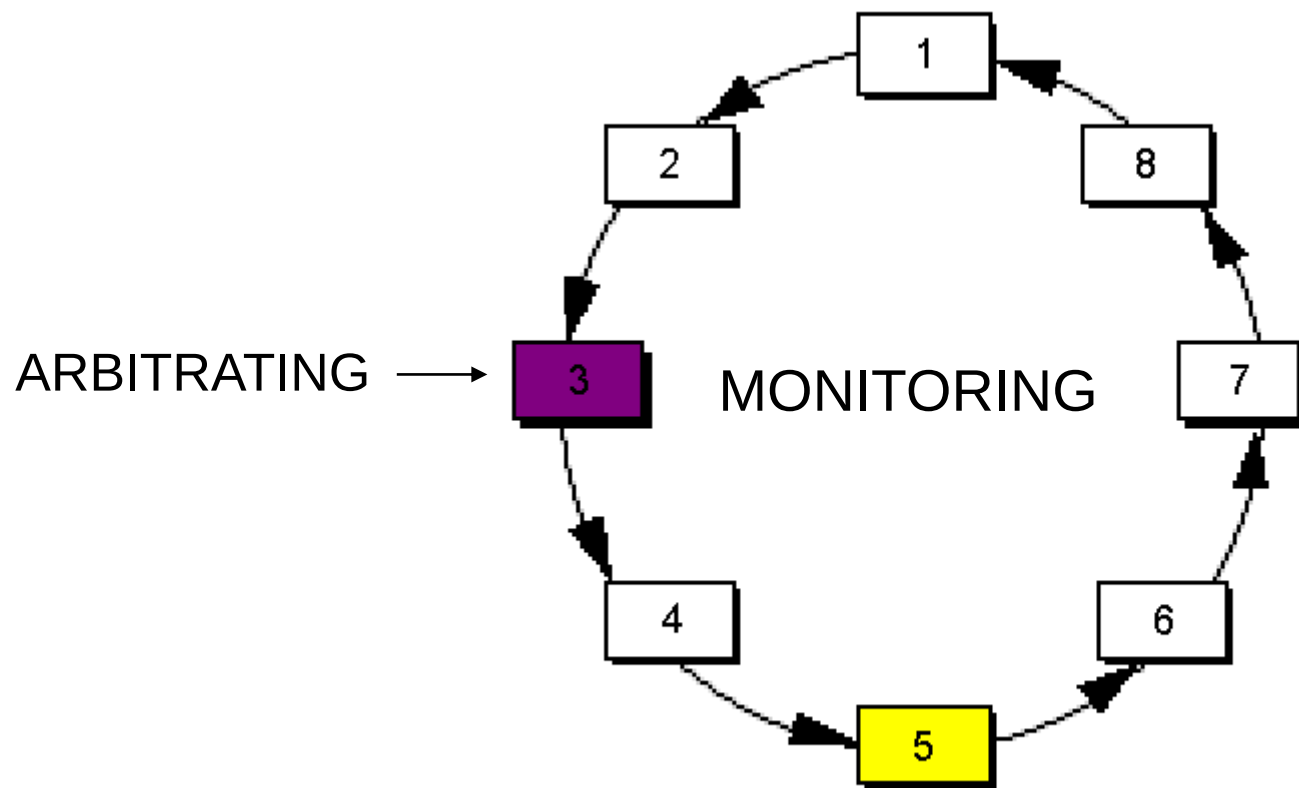
Передача сообщения



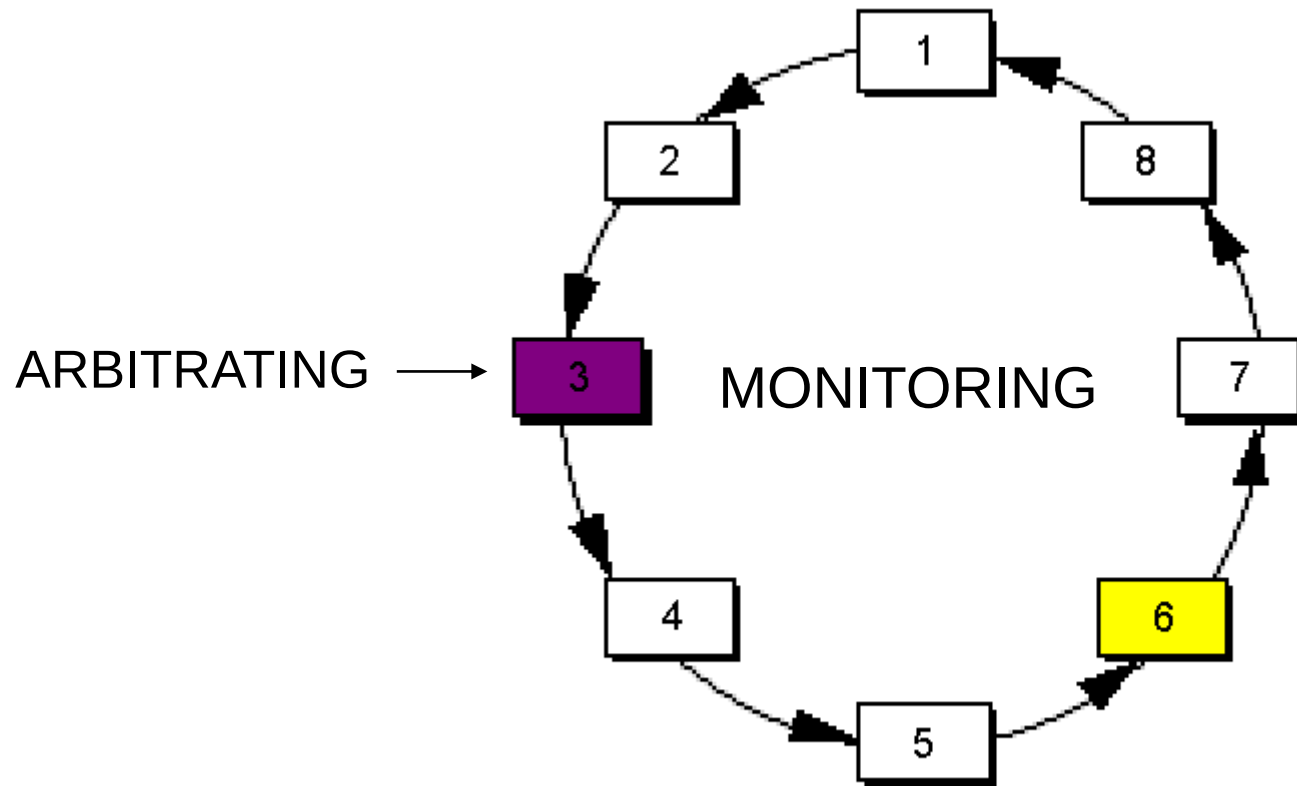
Передача сообщения



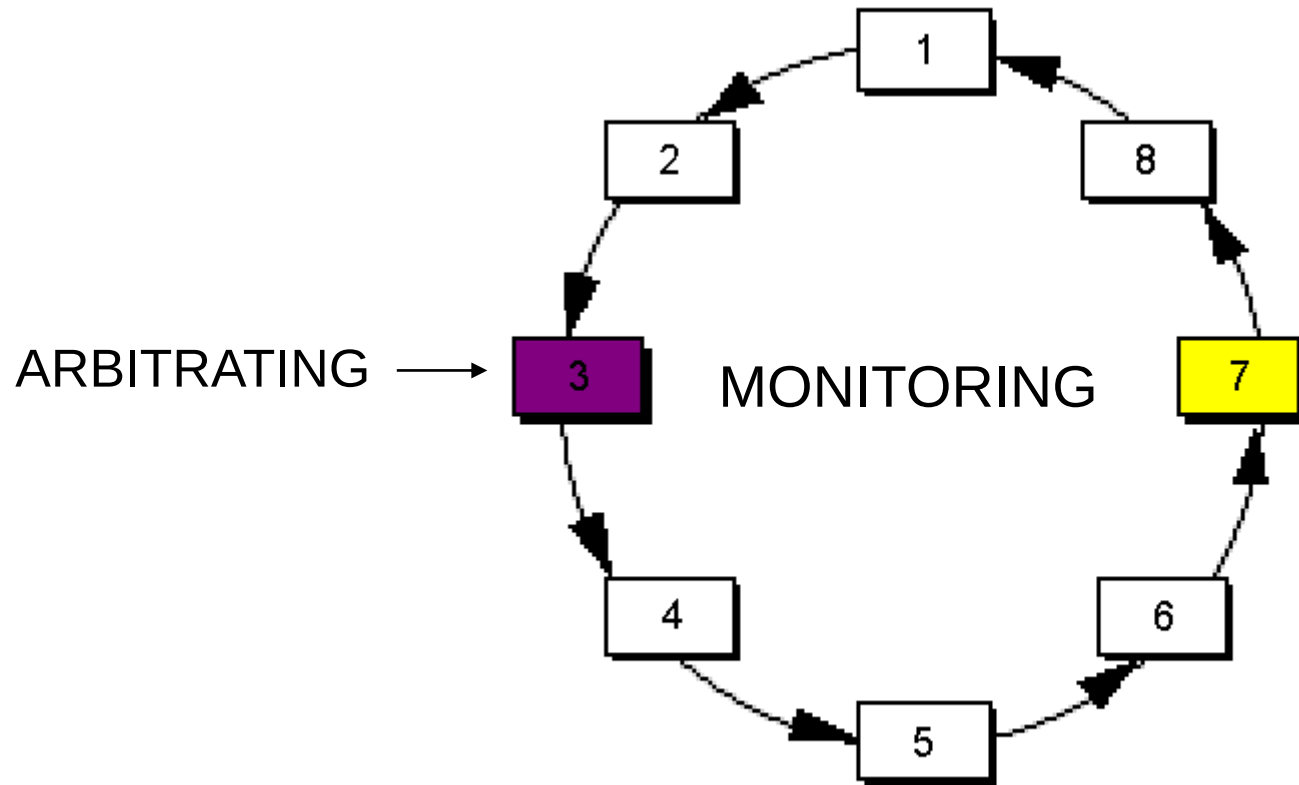
Передача сообщения



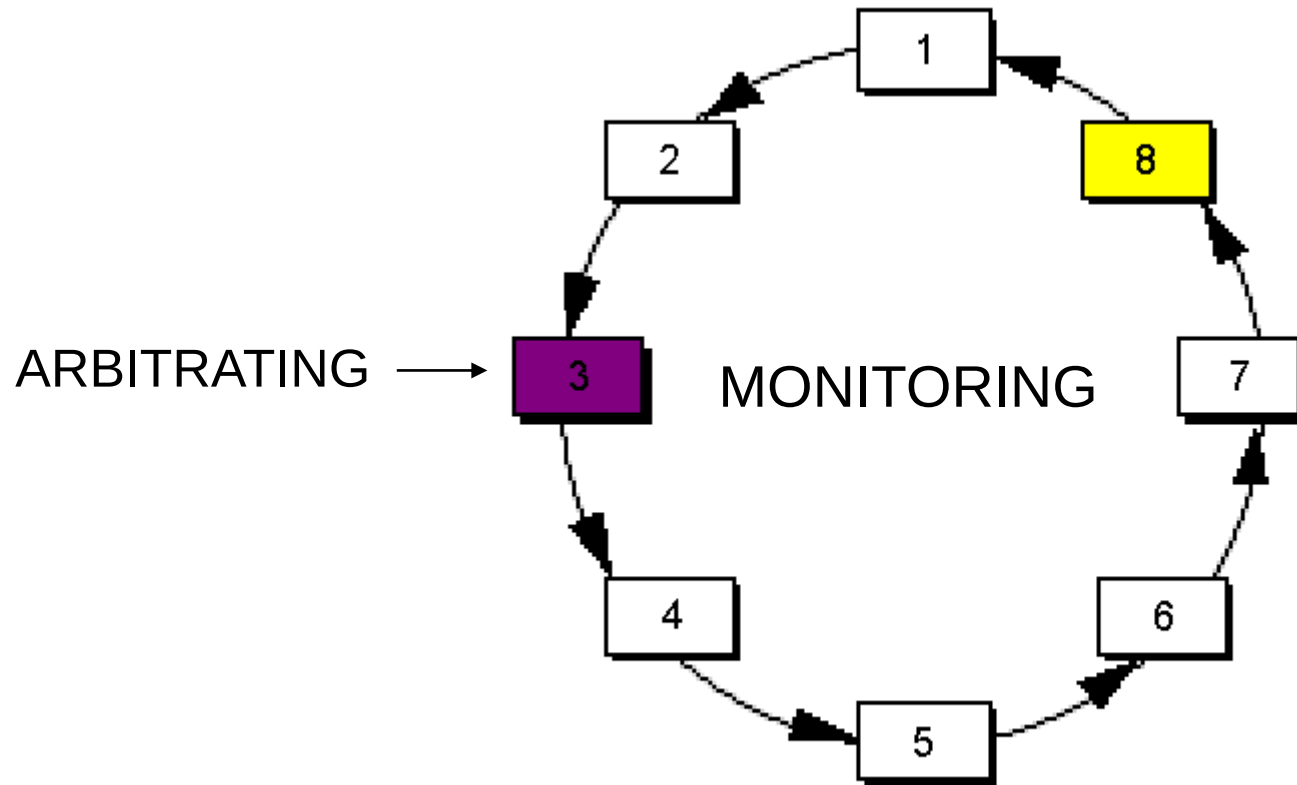
Передача сообщения



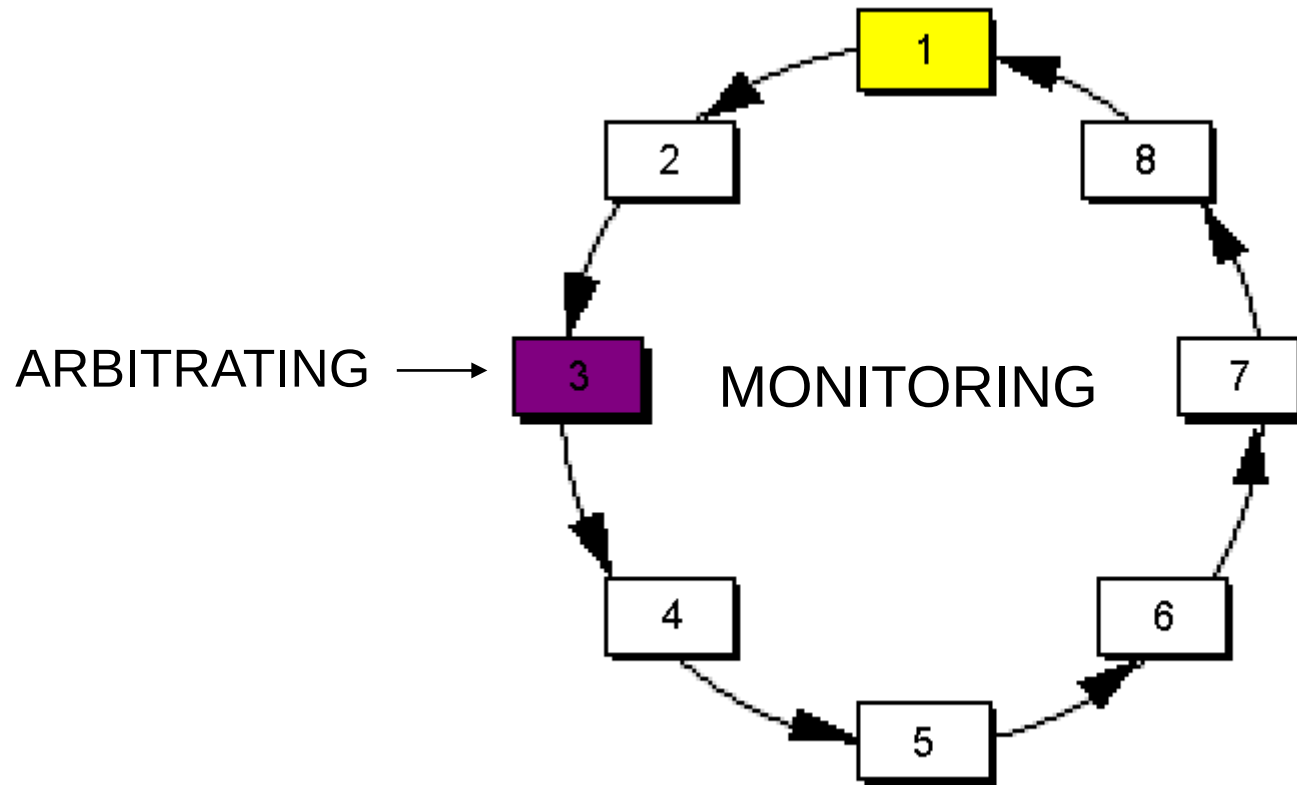
Передача сообщения



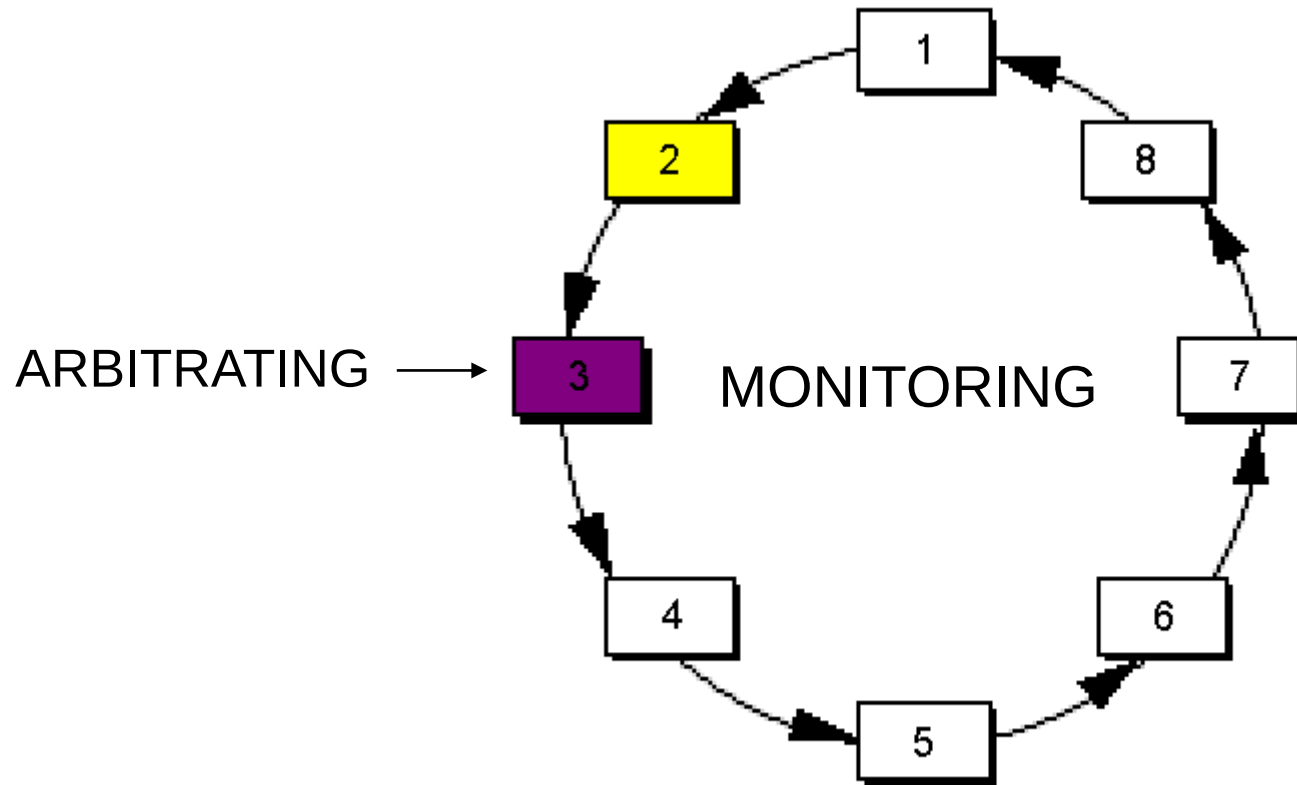
Передача сообщения



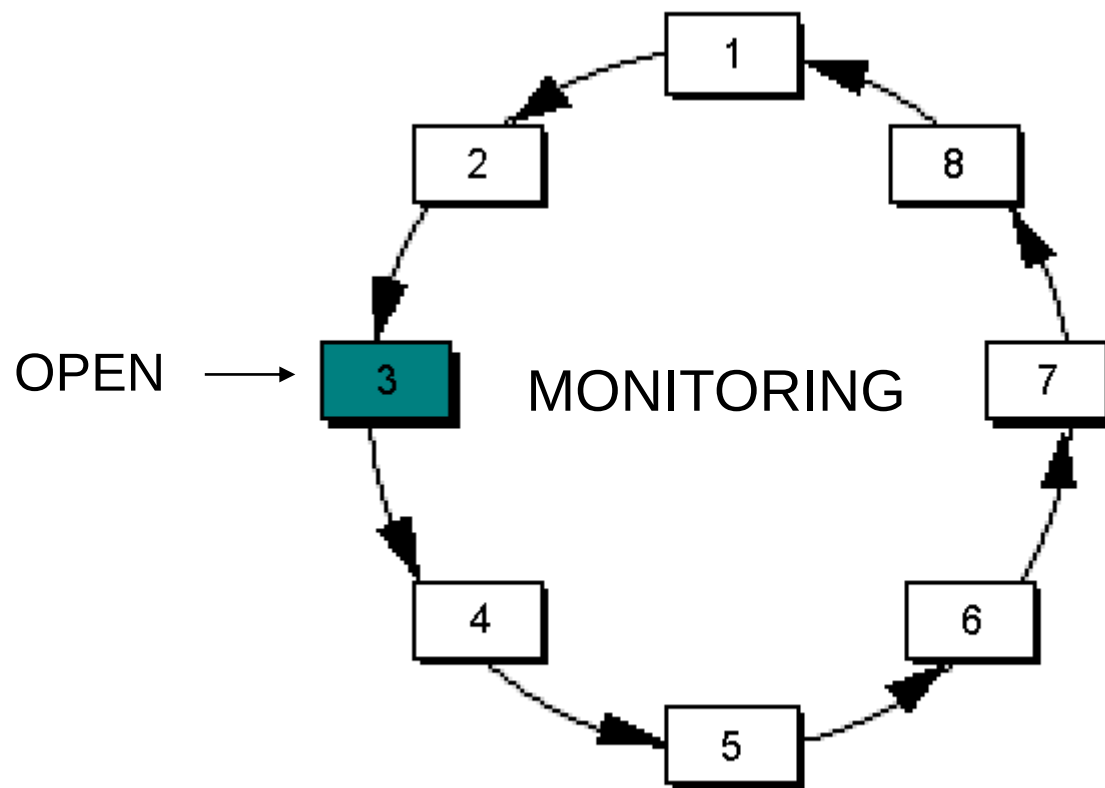
Передача сообщения



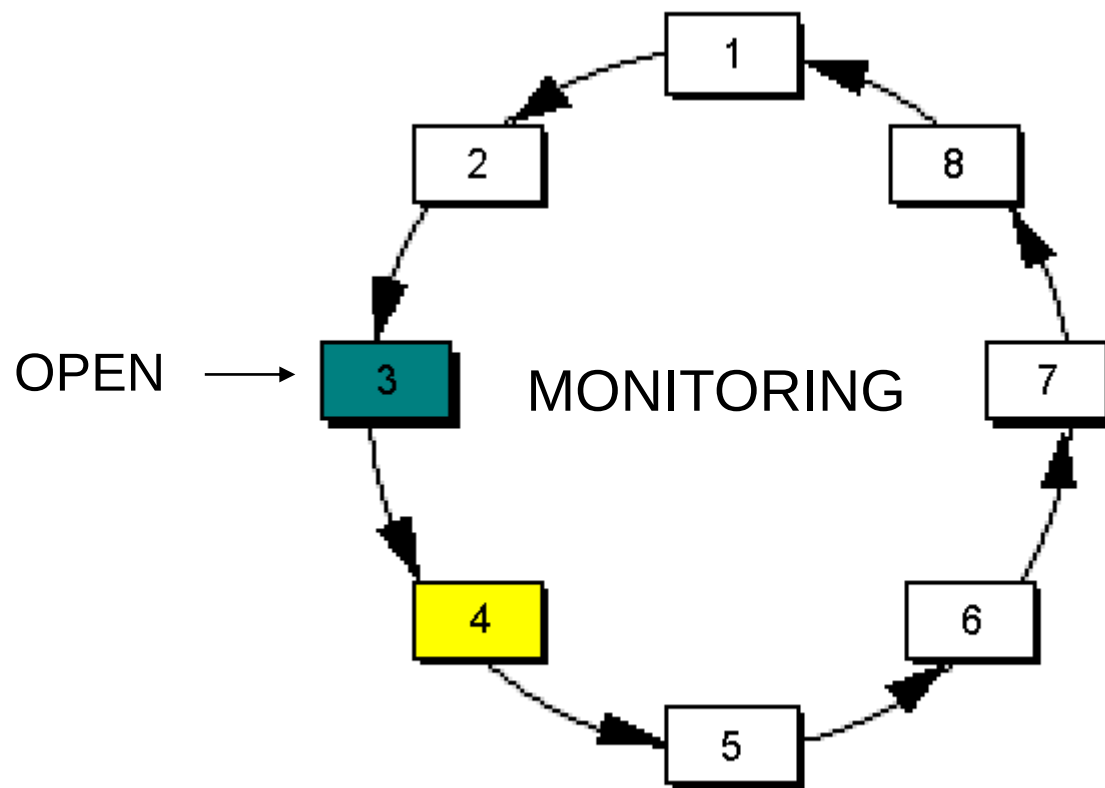
Передача сообщения



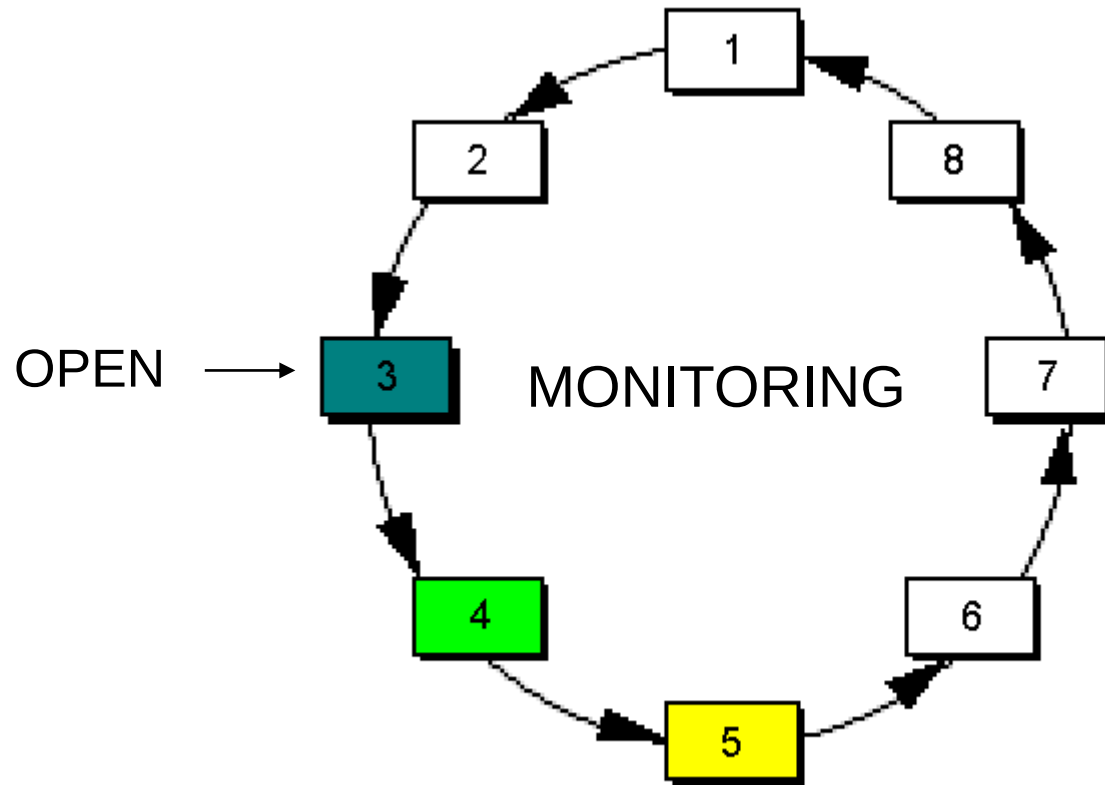
Передача сообщения



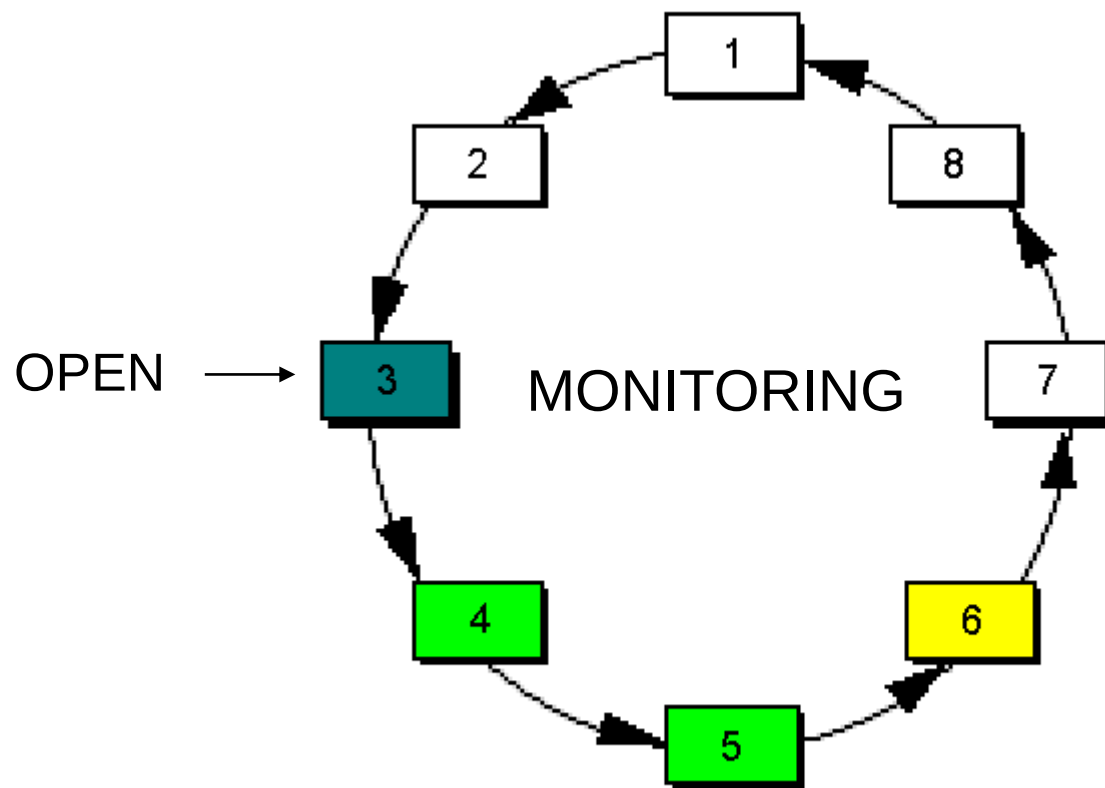
Передача сообщения



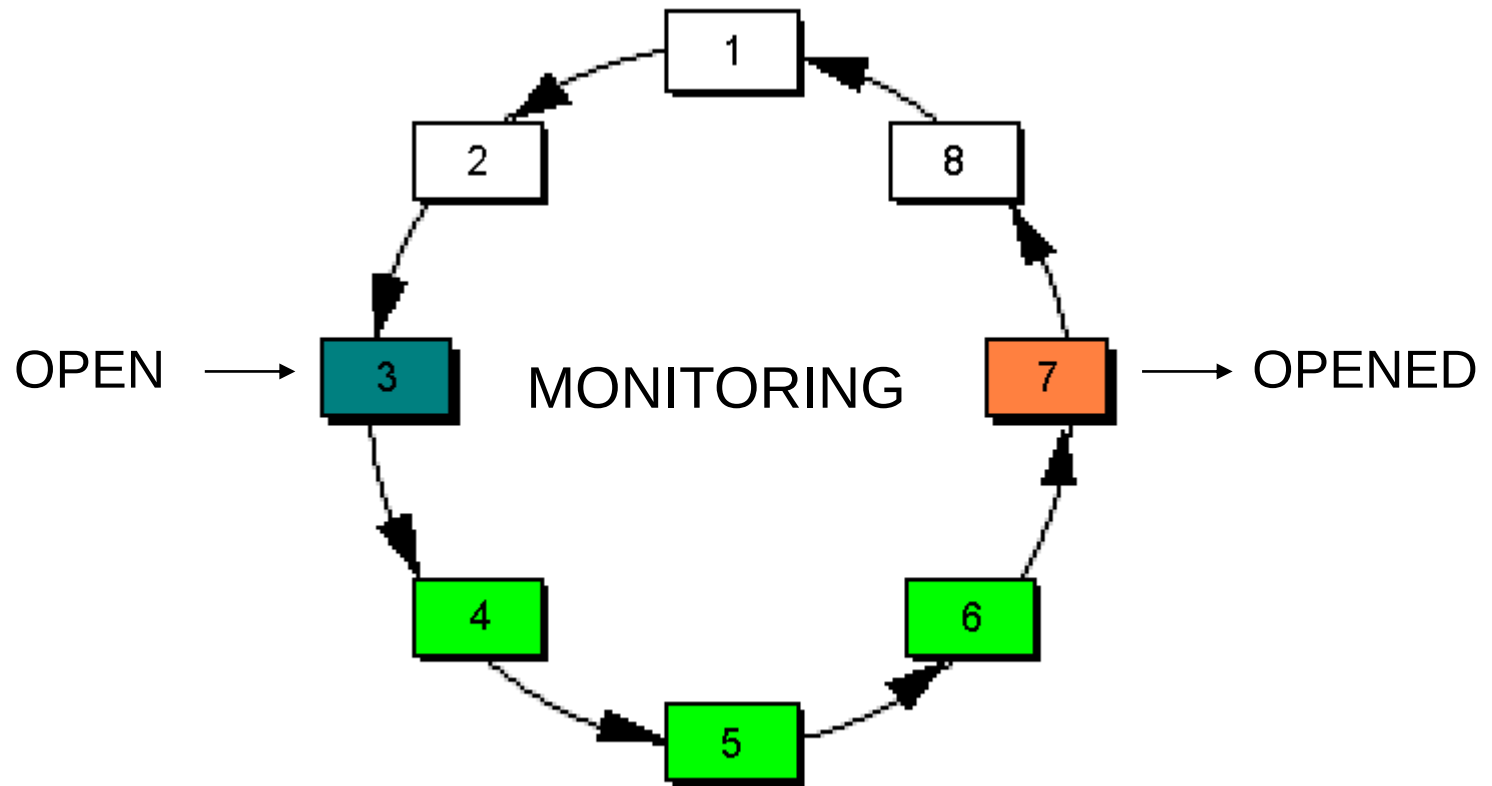
Передача сообщения



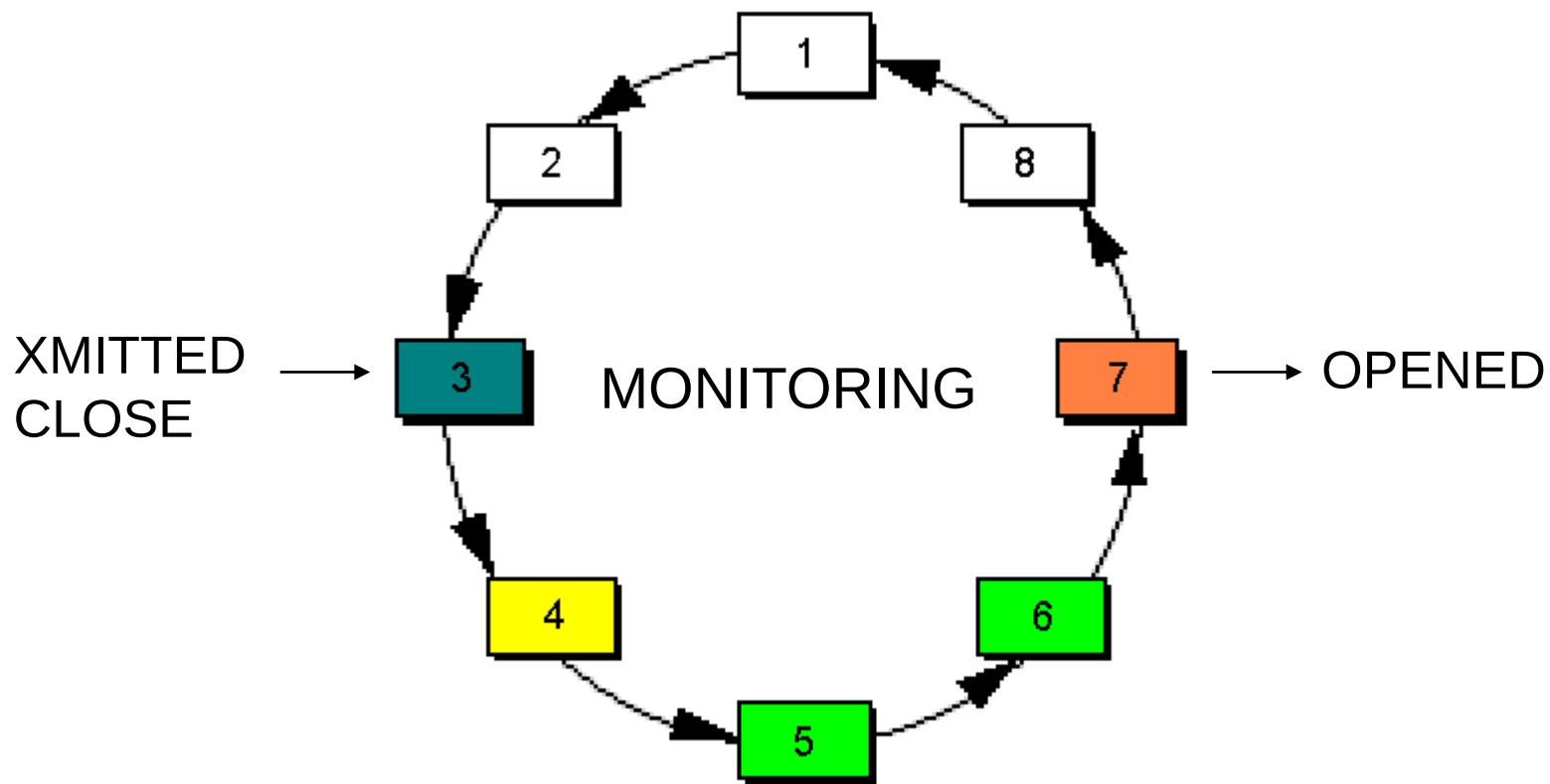
Передача сообщения



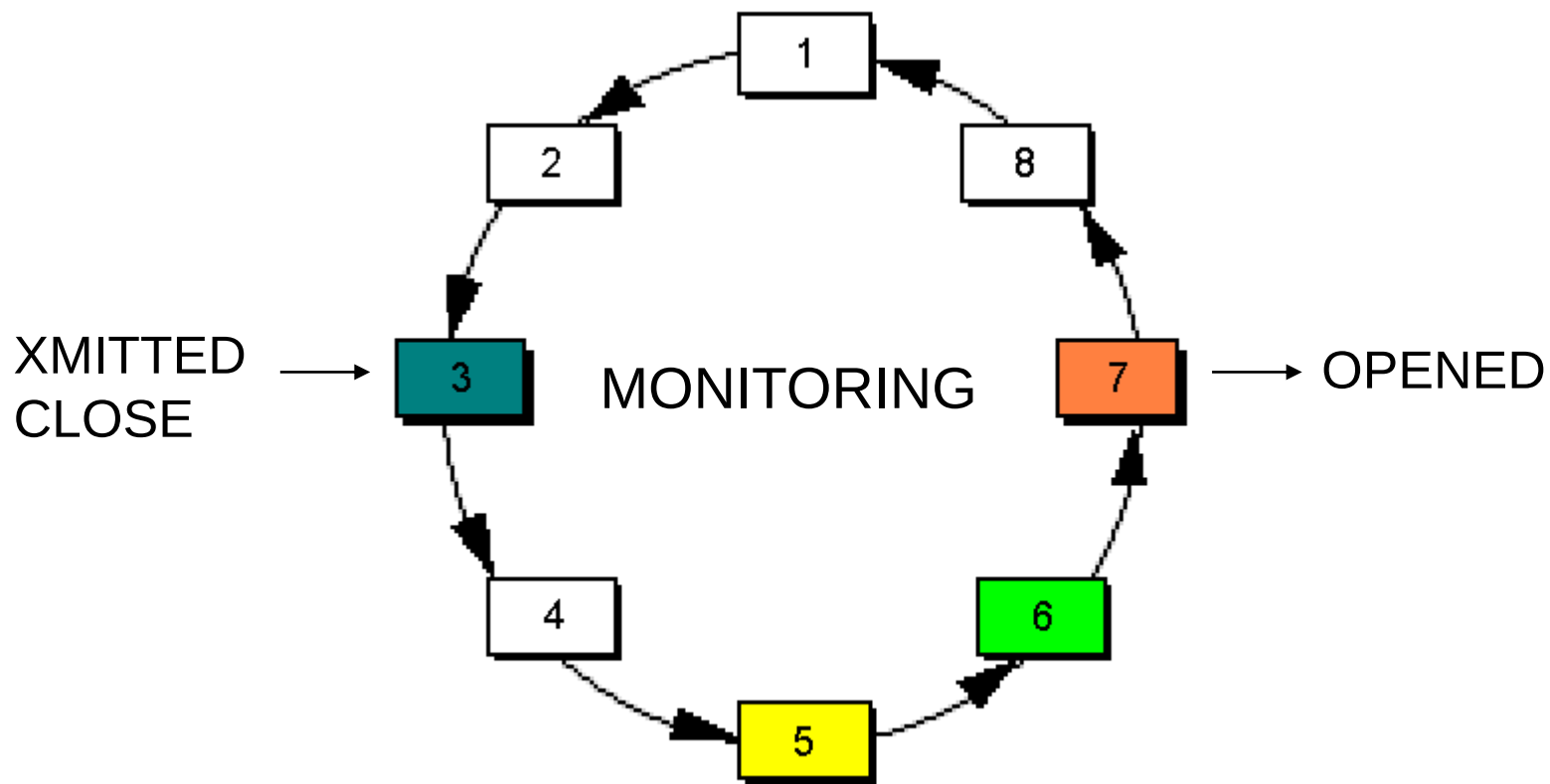
Передача сообщения



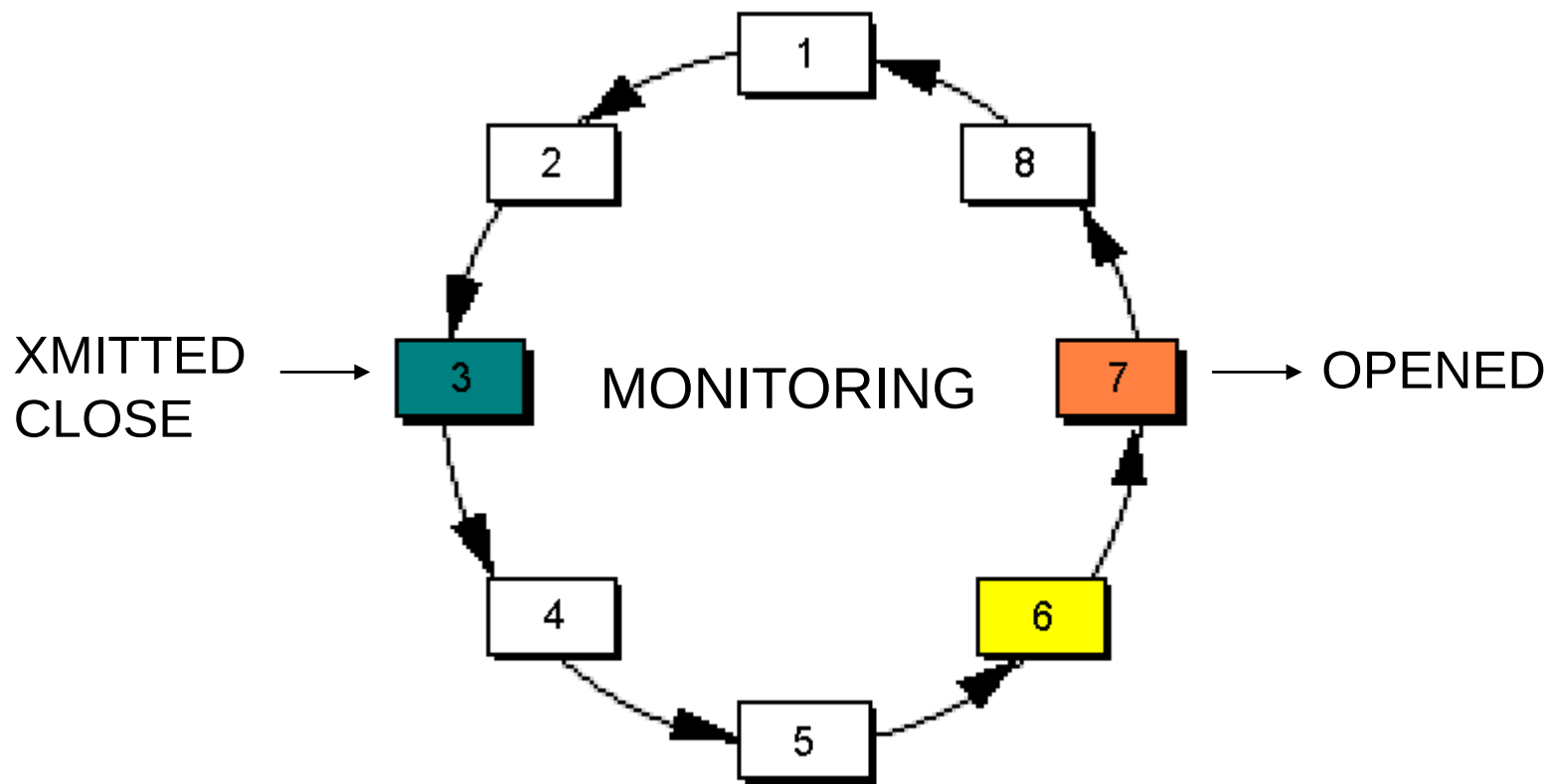
Передача сообщения



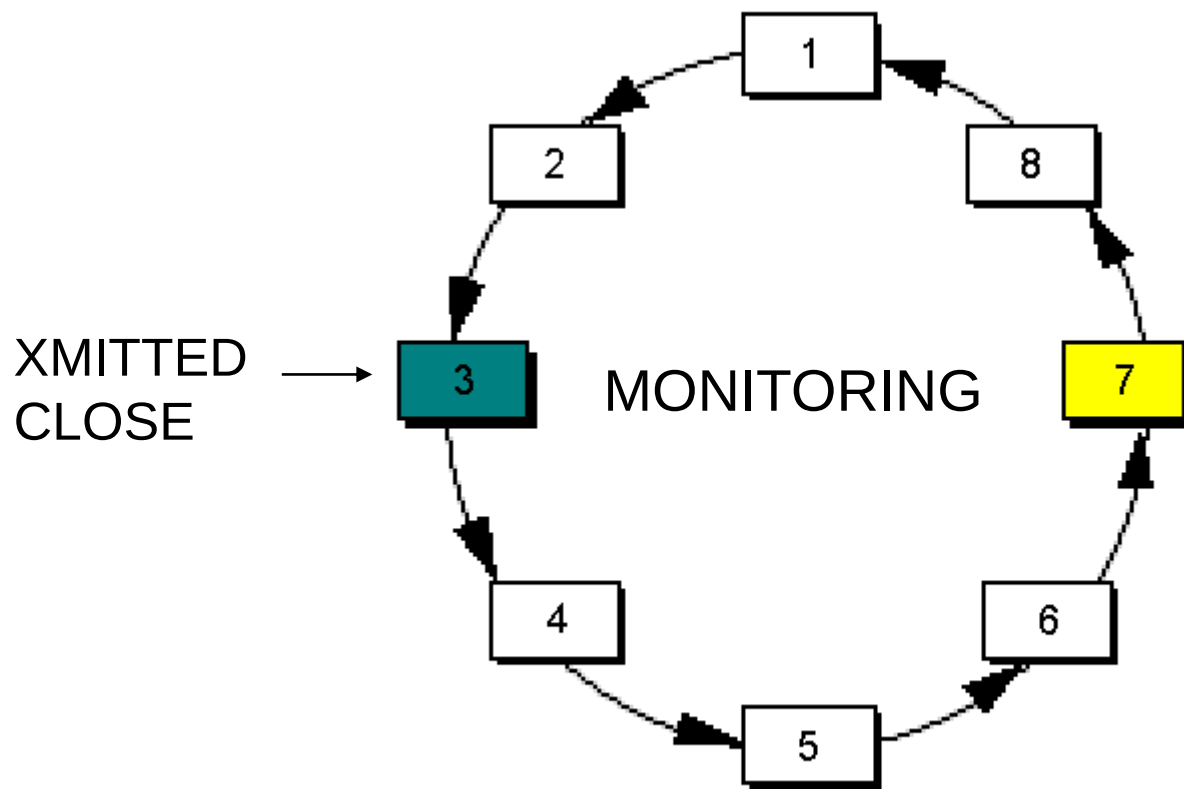
Передача сообщения



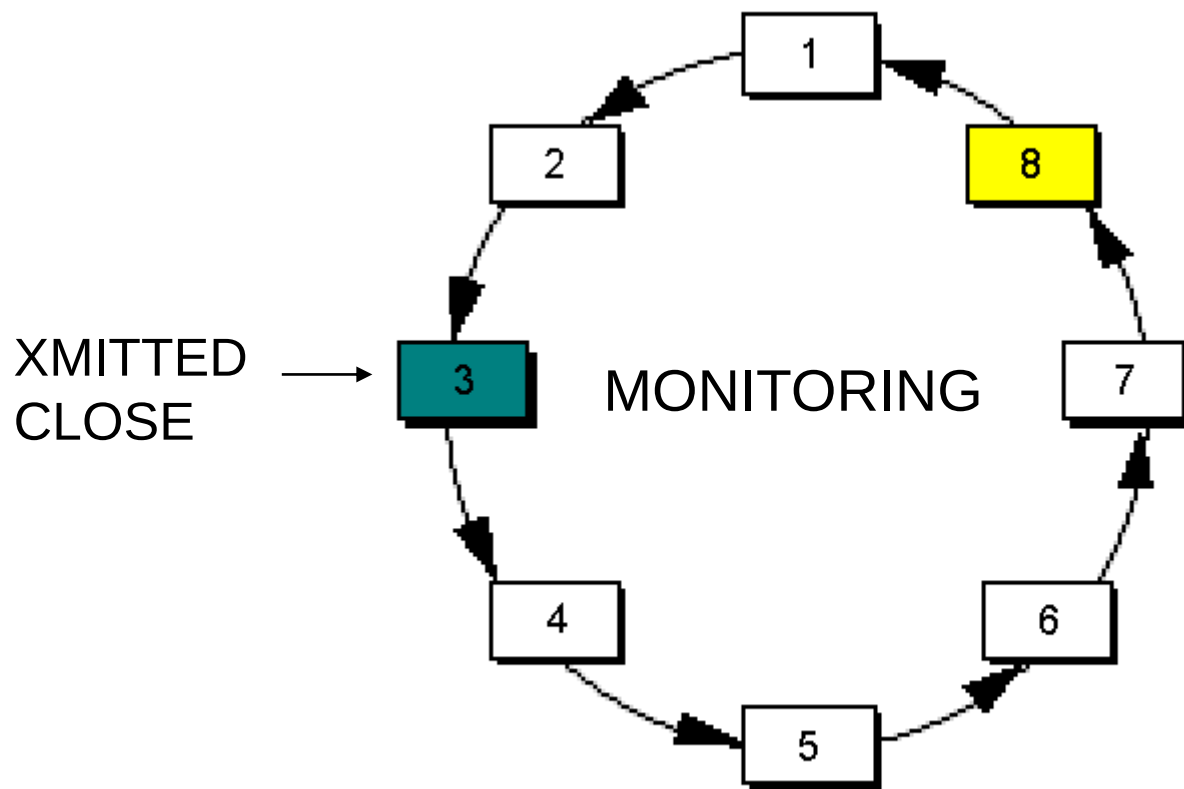
Передача сообщения



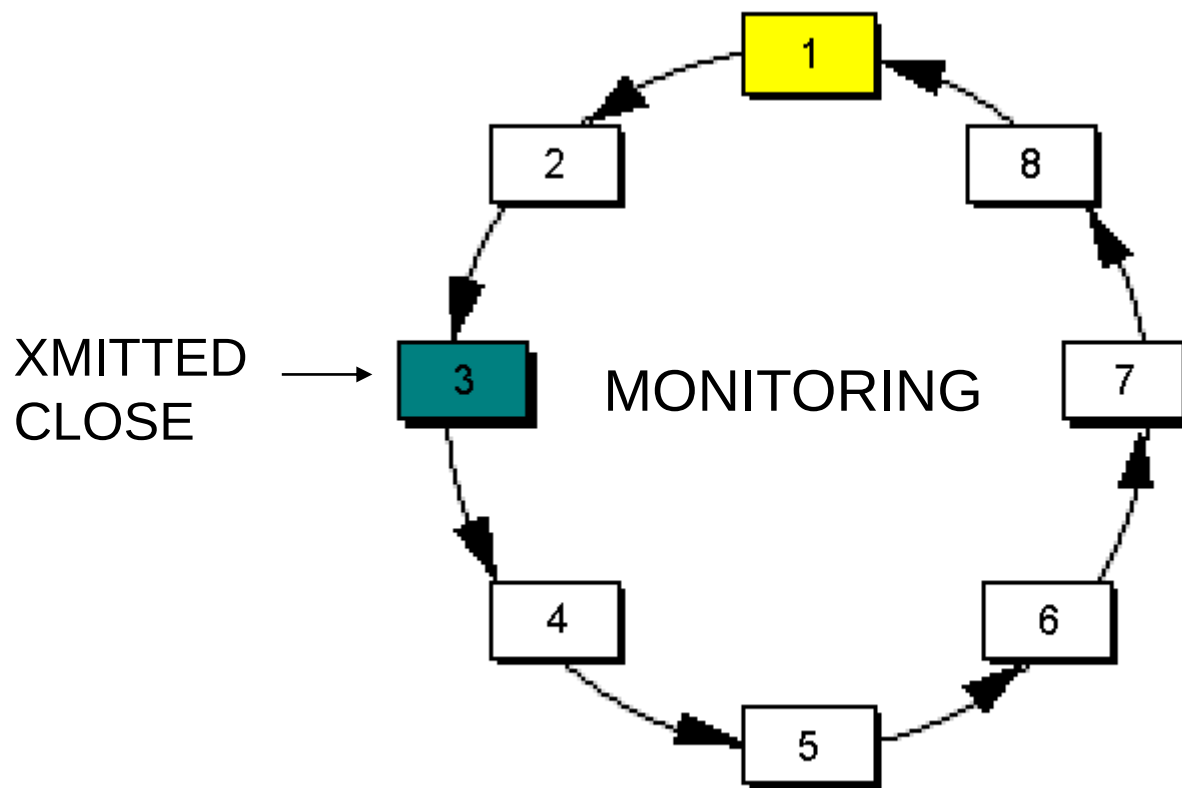
Передача сообщения



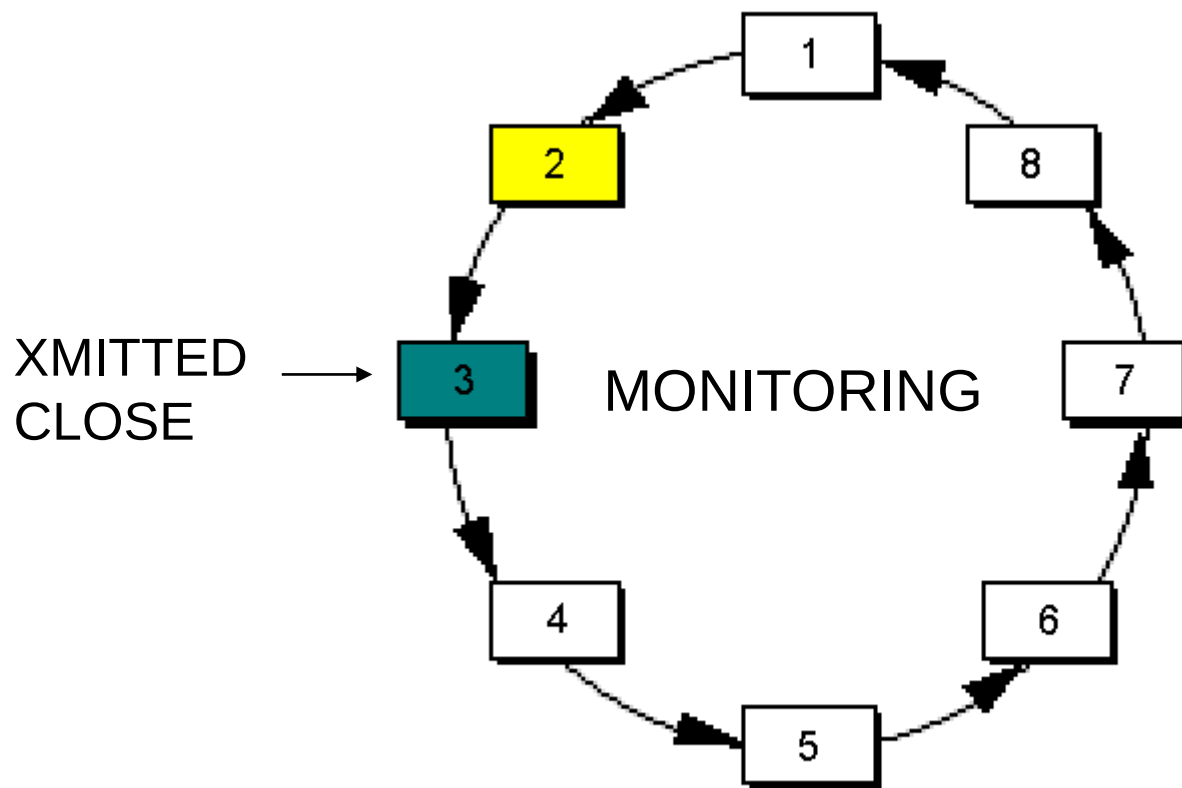
Передача сообщения



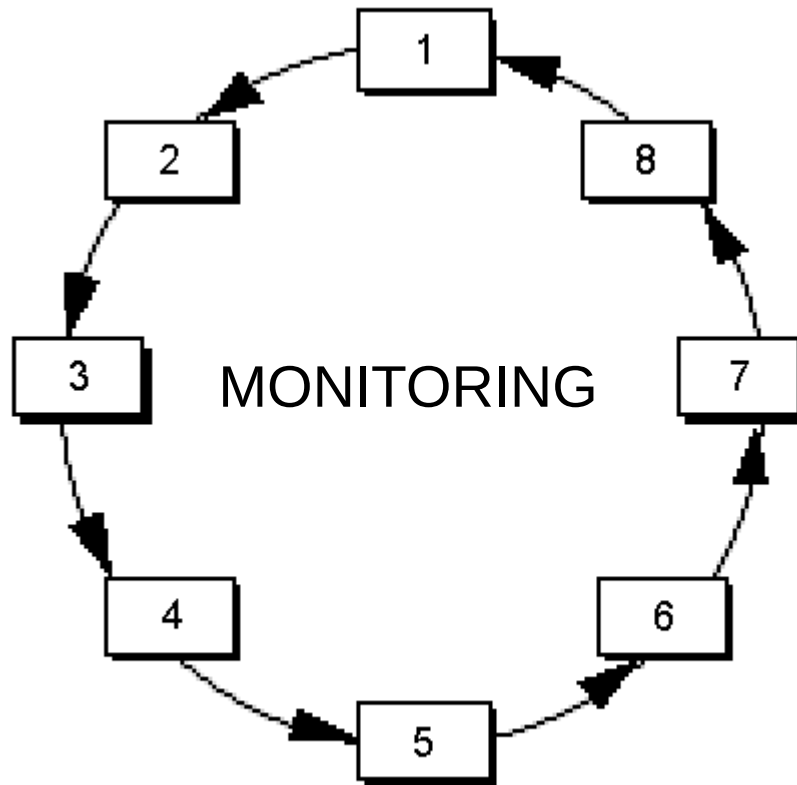
Передача сообщения



Передача сообщения



Передача сообщения

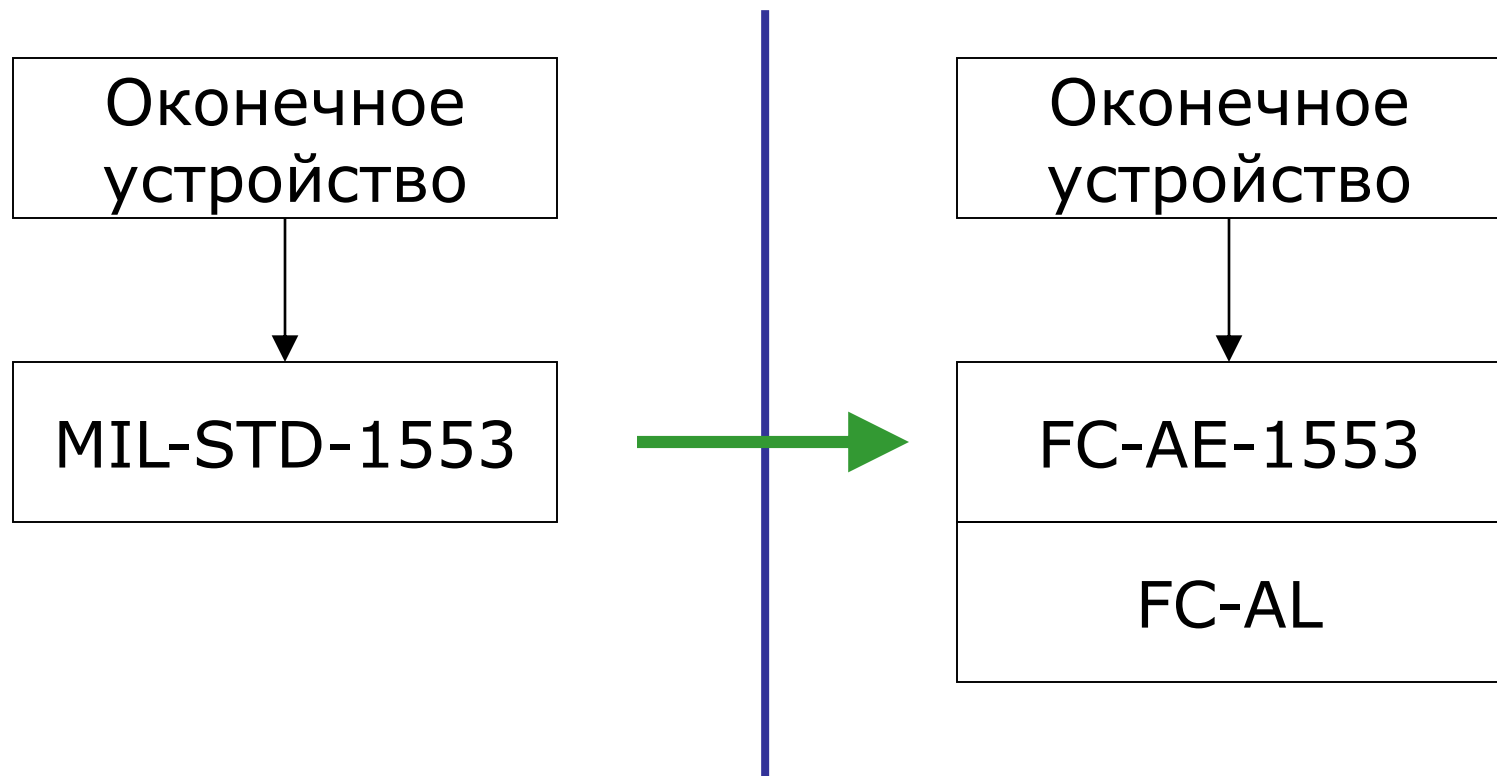


Протокол FC-AE-1553

- Протокол FC-AE-1553 является протоколом верхнего уровня (в стеке Fibre Channel) и эмулирует работу канала MIL STD-1553B в кольце с арбитражем.
- Оконечные устройства кольца с арбитражем сами не инициируют информационные обмены. За это отвечает специально выделенное оконечное устройство кольца с арбитражем, называемое контроллером кольца.

Протокол FC-AE-1553

предназначен для «имитации» протокола
MIL STD-1553 на кольце с арбитражем



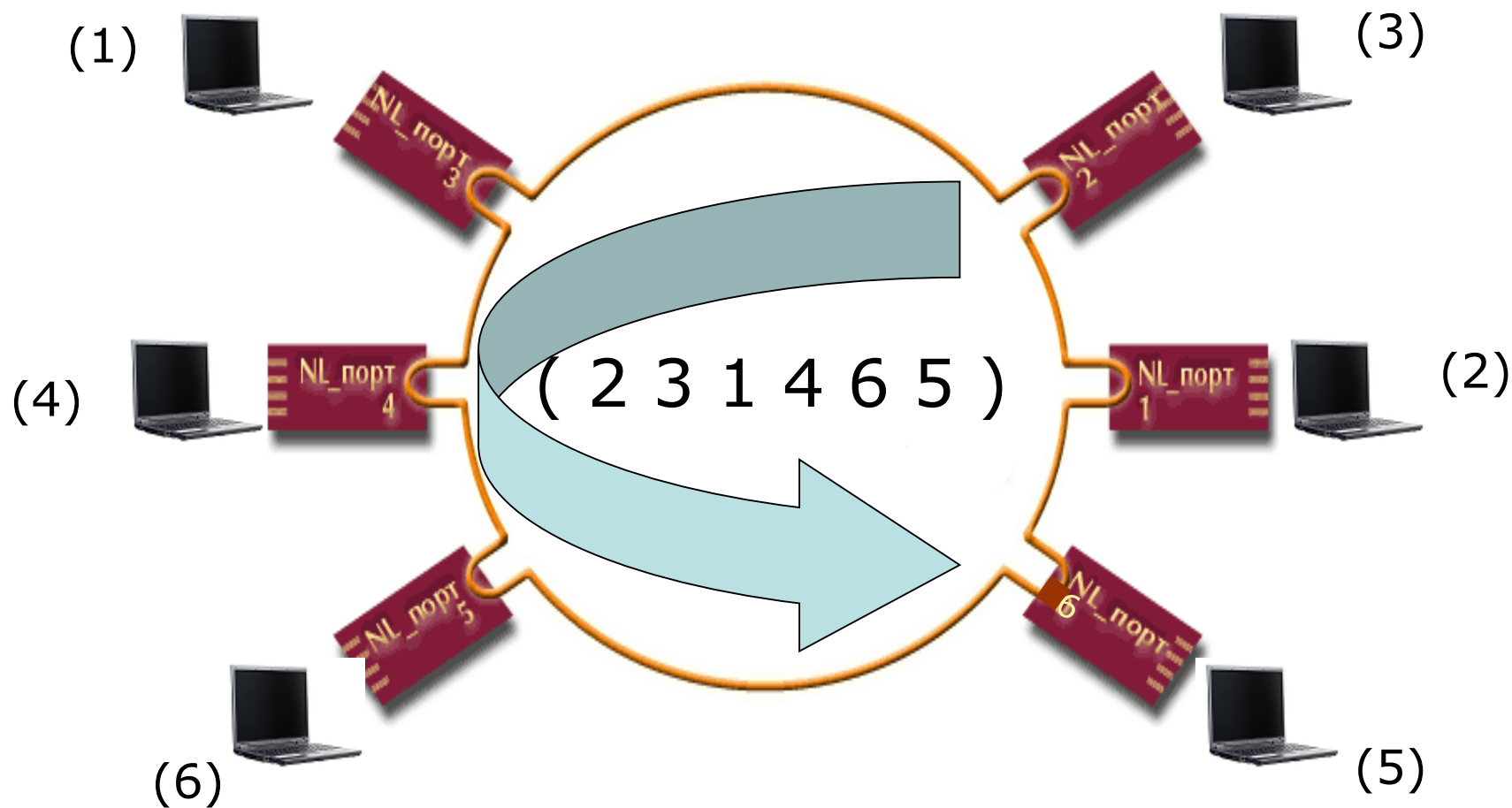
Протокол FC-AE-1553

- Передача данных от абонента *A* к абоненту *B*:
 - 1) передача сообщения от контроллера кольца оконечному устройству с адресом *A* с командой передать сообщение оконечному устройству с адресом *B*;
 - 2) передача сообщения с данными от оконечного устройства с адресом *A* оконечному устройству с адресом *B*.

Задача построения магистральных каналов информационного обмена с использованием кольца с арбитражем FC

- Выбор порядка расположения оконечных устройств в кольце с арбитражем
 - от порядка следования устройств зависит длительность передачи сообщений
- Назначение адресов оконечным устройствам
 - влияет на арбитраж, если *нет* централизованного управления
- Построение расписания обменов

Топология кольца с арбитражем

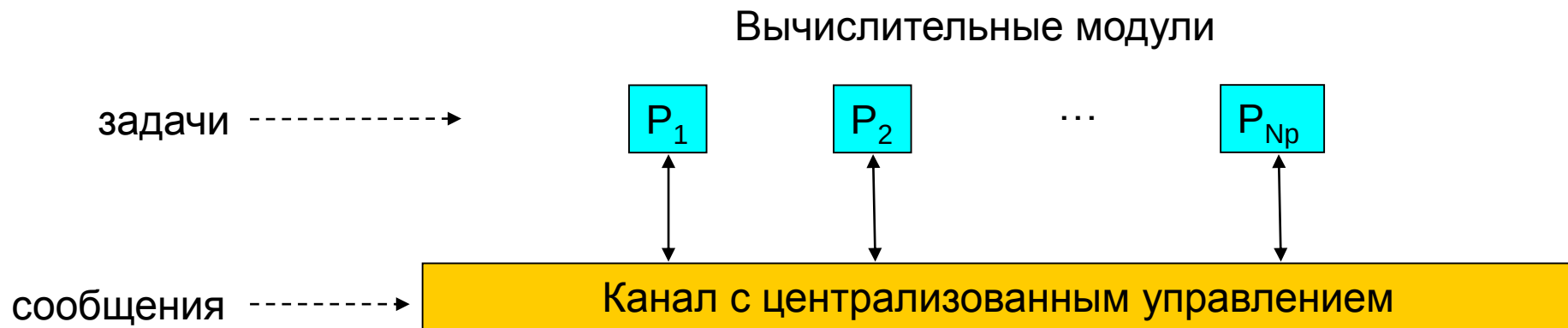


Построение расписания обменов

- Для схемы с централизованным управлением. Эта задача возникает, если используется протокол FC-AE-1553 и формулируется аналогично задаче для канала MIL STD-1553B.
- Для схемы с децентрализованным управлением. В этом случае должно составляться расписание выставления заявок на арбитраж для каждого оконечного устройства.

Совместное планирование вычислений и обменов

Система реального времени

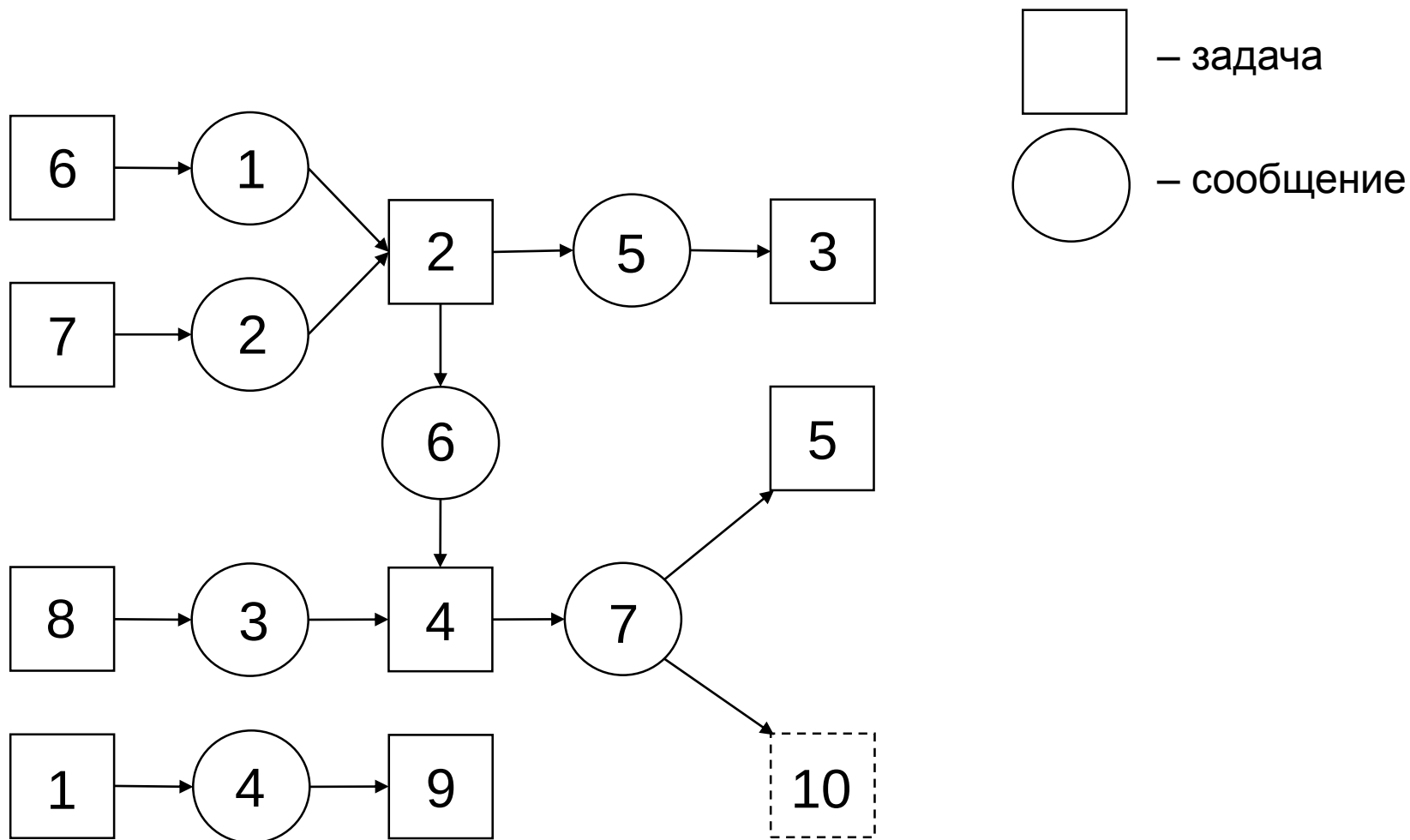


Длительность передачи сообщения
зависит от того, на какие вычислительные
модули размещены задачи

Цель

- Построение расписания выполнения задач на вычислительных модулях в составе ИУС РВ и расписания передачи сообщений между ними по каналу с централизованным управлением с соблюдением ограничений:
 - реального времени
 - совместимости расписаний
 - ограничений, связанных со спецификой аппаратных и программных средств ИУС РВ

Задачи и сообщения

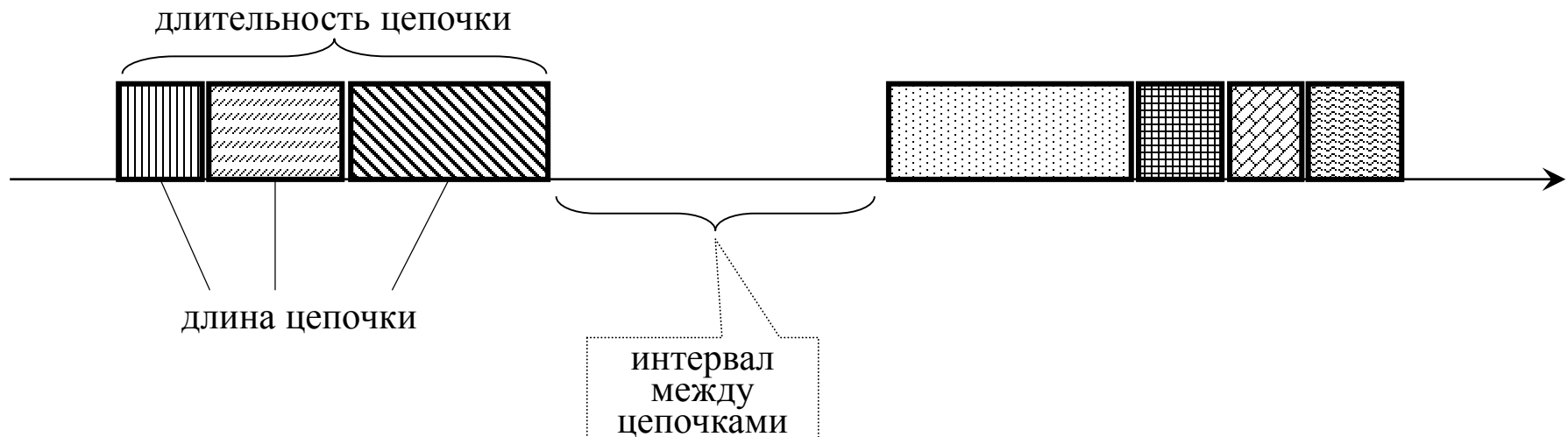


Расписание выполнения задач

- Требования реального времени
- На одном и том же вычислительном модуле в каждый момент времени может выполняться только одна задача
- Каждая задача запланирована на допустимый вычислительный модуль

Расписание передачи сообщений

- Требования реального времени
- В каждый момент времени может передаваться только одно сообщение
- Технологические ограничения на обмен данными



Задача

- По заданным наборам задач и сообщений построить корректные совместимые расписания выполнения задач и передачи сообщений, содержащие максимальное количество задач и сообщений

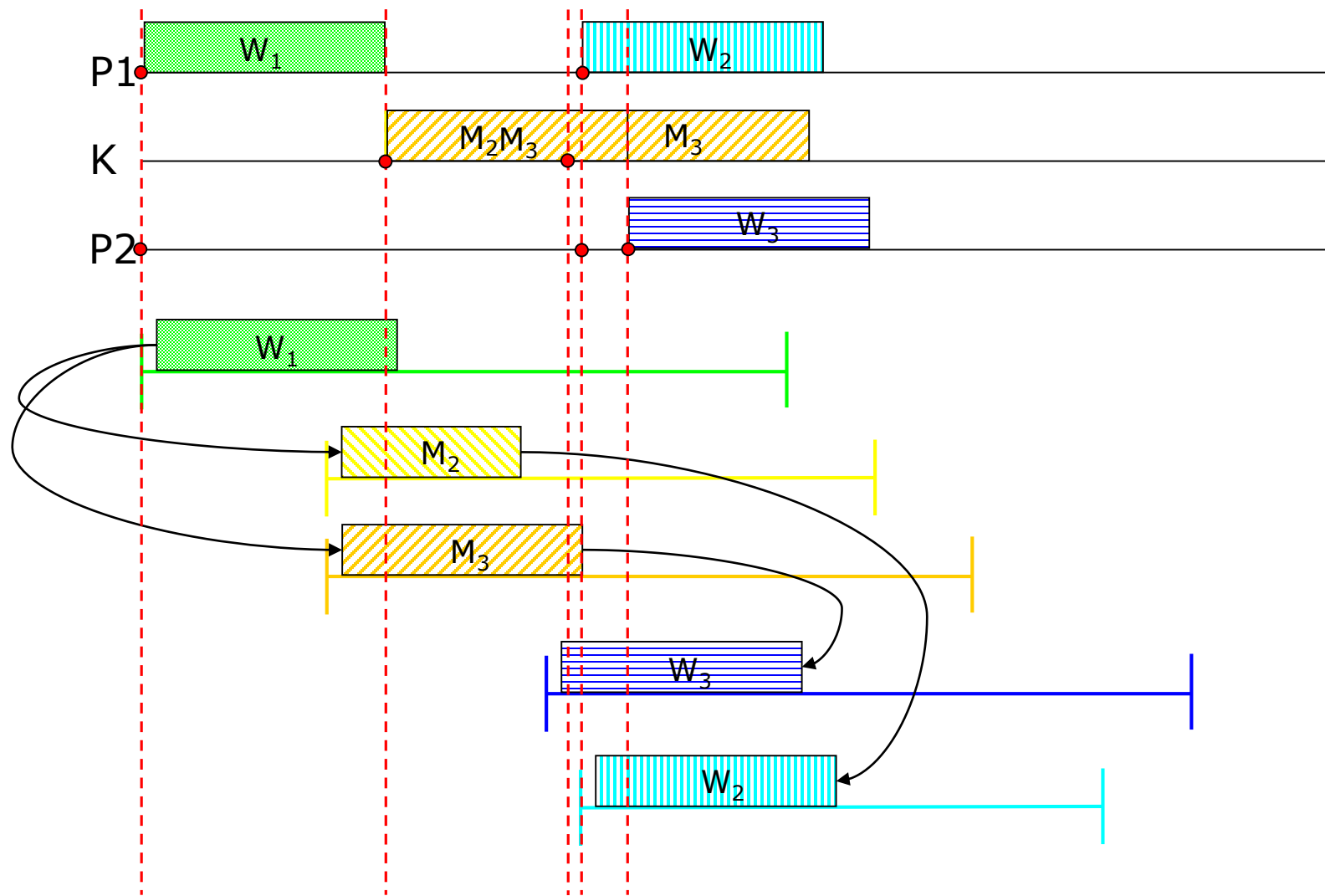
Подходы

- 3 подхода:
 - Сначала построить расписание сообщений, затем расписание задач
 - Сначала построить расписание задач, затем расписание сообщений
 - Одновременное построение обоих расписаний
- Для первых двух подходов существуют частные задачи, имеющие полное расписание, которое нельзя построить в рамках этих подходов

Жадный алгоритм

- Проблема:
 - Изменяющаяся длительность передачи сообщений
- Решение:
 - Считать длительность передачи сообщения максимально возможной и корректировать её в процессе работы алгоритма

Жадный алгоритм: пример



Далее...

- Каналы информационного обмена на основе коммутаторов
 - топология
 - виртуальные каналы
 - планирование обмена

Спасибо за внимание!

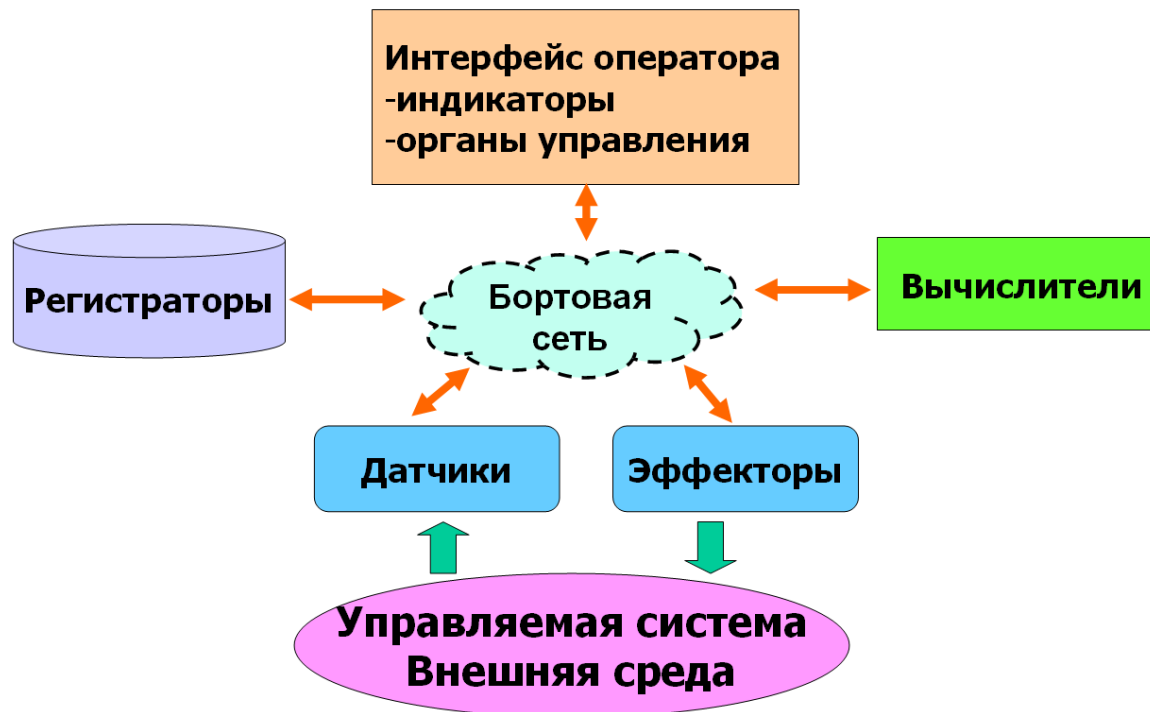
ИНФОРМАЦИОННО-УПРАВЛЯЮЩИЕ СИСТЕМЫ РЕАЛЬНОГО ВРЕМЕНИ

Лекция 9: *Коммутируемые сети*

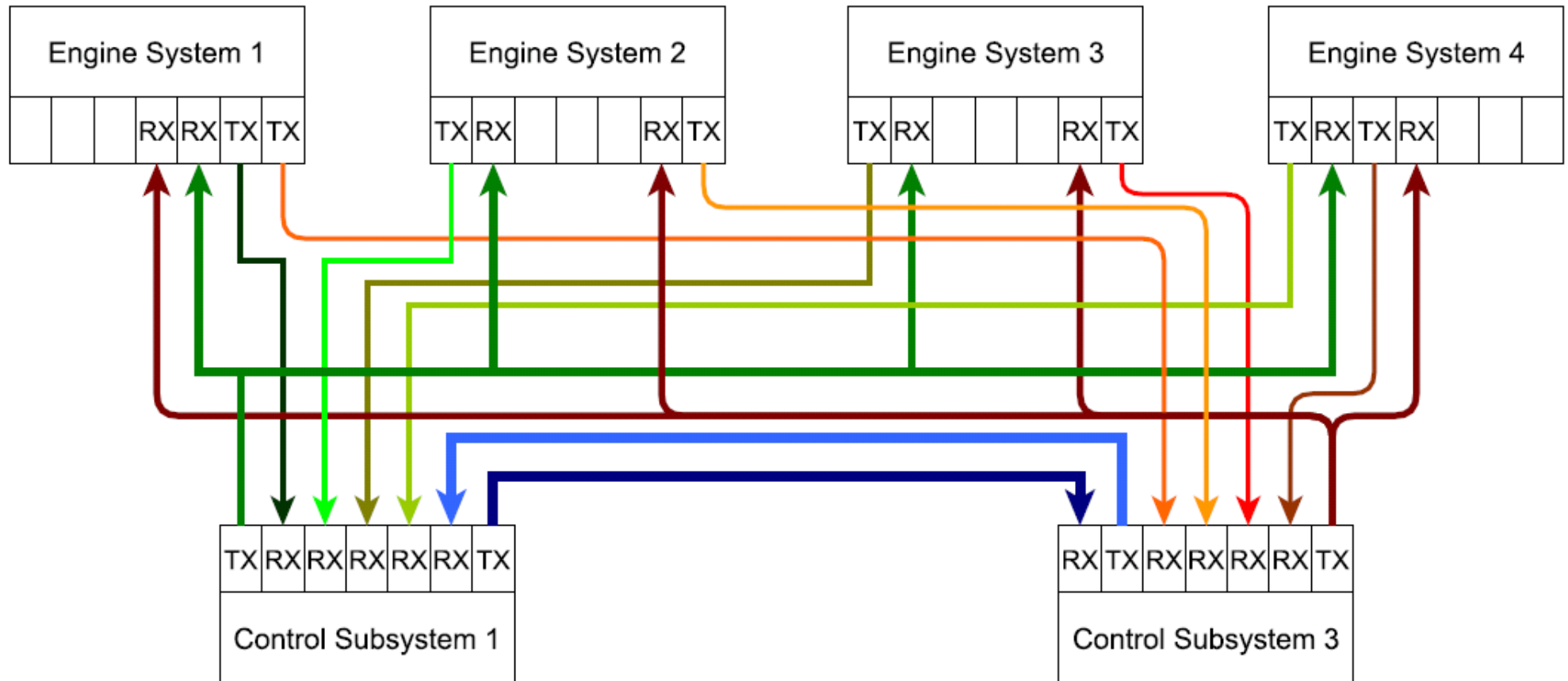
Кафедра АСВК,
Лаборатория Вычислительных Комплексов
Балашов В.В.

Бортовые сети

- Бортовые сети – обеспечение связи между бортовыми подсистемами
 - Надежная доставка
 - Соблюдение требований реального времени

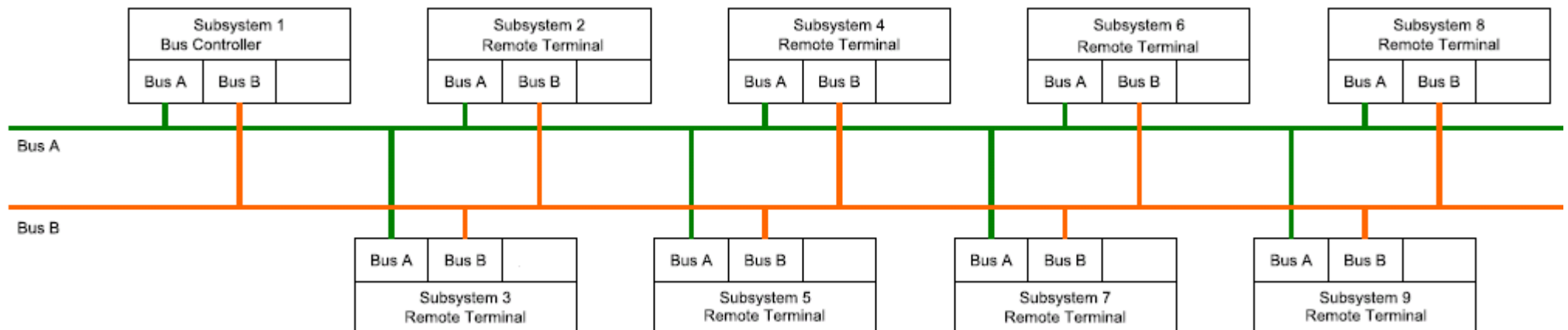


Каналы точка-точка



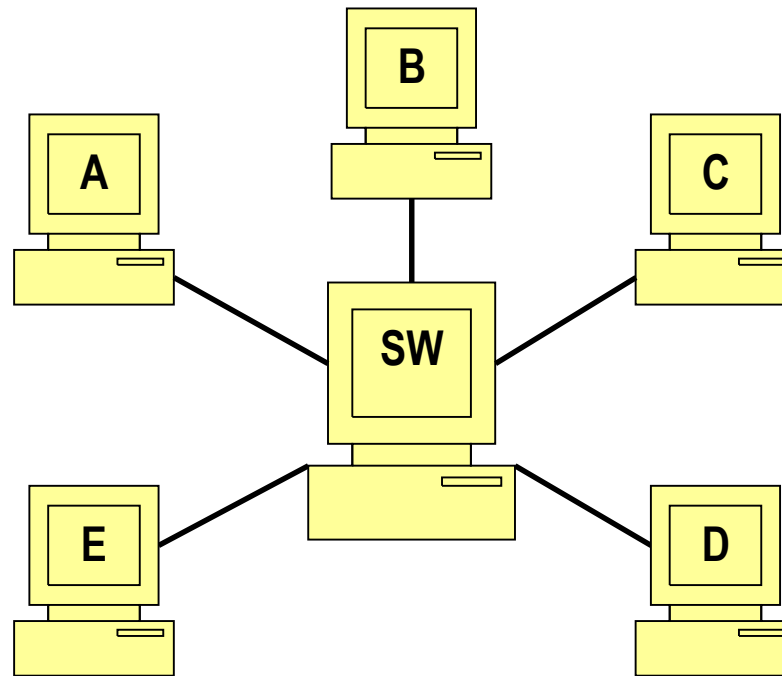
- Много кабелей
- Пропускная способность: недоиспользование, нехватка, сложность наращивания
- Сложно закладывать резерв
- Проблемы с передачей данных по сложному маршруту
- Низкая адаптивность (невозможна реконфигурация)

Интеграция каналов, мультиплексирование трафика



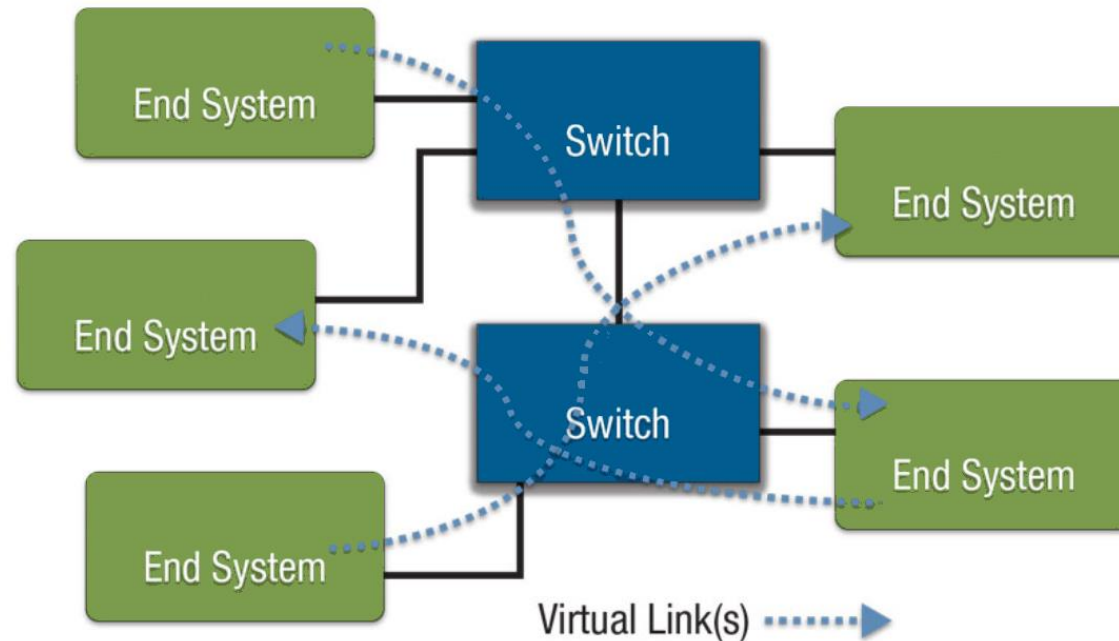
- Много каналов точка-точка → общая шина для многих потоков данных
- Проблема коллизий при доступе к шине
 - синхронизация доступа (нужно единое время)
 - *либо* централизованное управление (накладные расходы...)
- Последовательная обработка запросов => задержки
- Нет устойчивости к «генерации» в канале при выходе абонента из строя

Коммутатор: параллелизм обмена



- Дуплексные (двунаправленные) каналы
- Частичный параллелизм обмена
 - $A \rightarrow B \parallel B \rightarrow C \parallel C \rightarrow D$ – нет конфликта
 - $A \rightarrow B, C \rightarrow B$ – конфликт на линии $SW \rightarrow B$; как делить линию?
- Неустойчивость к «генерации»
- Проблема мультиплексирования потоков данных при отправке
- Оставшиеся конфликты – как лечить?
 - синхронизация доступа (нужно единое время)
 - централизованное управление (накладные расходы...)
 - верхние оценки задержек (а если между A и B большой поток?)

Виртуальные каналы



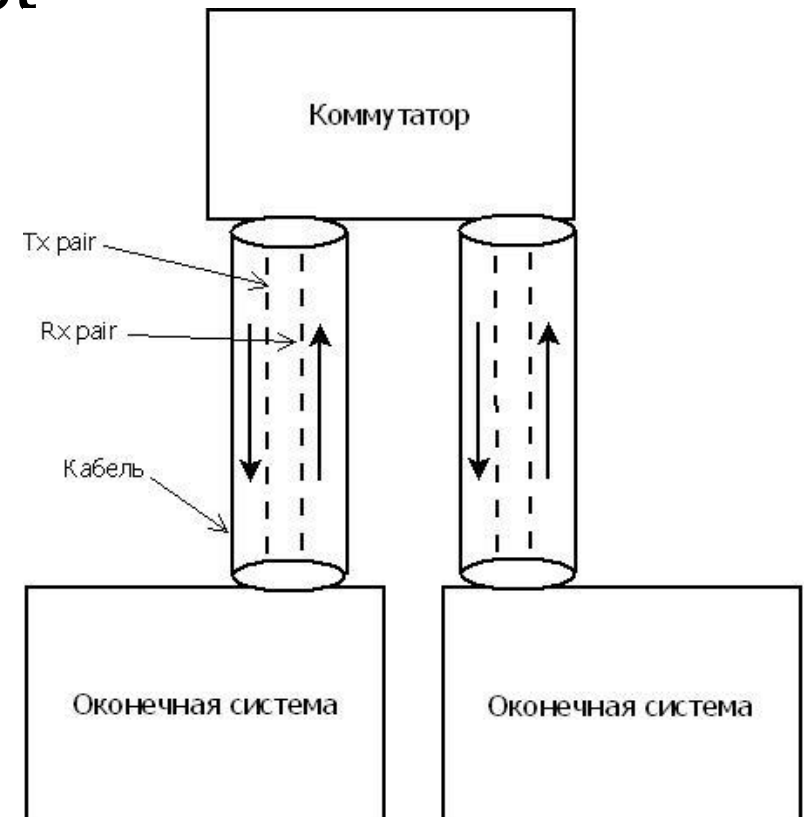
- Разделение пропускной способности, разграничение потоков данных
 - => пригодность для сложного трафика из множества потоков данных
- Гарантированные верхние границы задержек
- Резервирование, гибкость реконфигурации
- Реализация: согласованные действия отправителя и коммутаторов
- Устойчивость к «генерации»
 - коммутатор сбрасывает слишком частые кадры

Протоколы коммутируемых сетей с поддержкой ВК

- AFDX (на базе 100 Мбит Ethernet)
- FC-RT (на базе Fibre Channel)
- Программно-конфигурируемые сети

Стандарт AFDX

- Avionics Full-Duplex Ethernet (AFDX) – стандарт построения бортовых сетей на основе протокола Ethernet
 - Основан на протоколе Ethernet
 - Полнодуплексная передача данных
 - Теоретическая пропускная способность – до 100 Мбит/с на одном физическом соединении



Архитектура сети AFDX

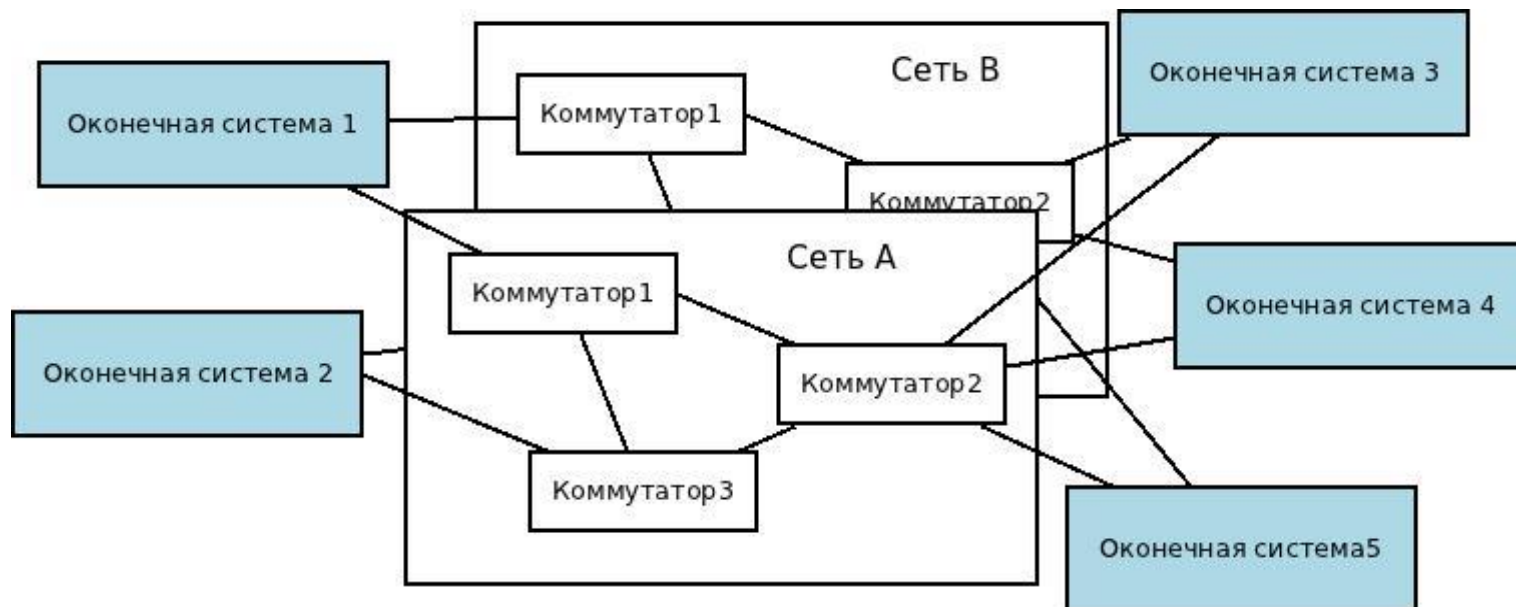
- Компоненты:

- Абоненты (бортовые подсистемы, отправители и получатели данных)
- Оконечные системы – интерфейс между абонентами и сетью
- Коммутаторы и физические соединения



Архитектура сети AFDX

- Дублирование сети для увеличения надежности передачи
 - Кадры передаются одновременно в обе сети
 - При диагностировании ошибки (например, несовпадение контрольной суммы) в одной сети данные берутся из другой сети
 - На конечной системе производится сброс кадра в случае, если кадр уже пришел из другой сети

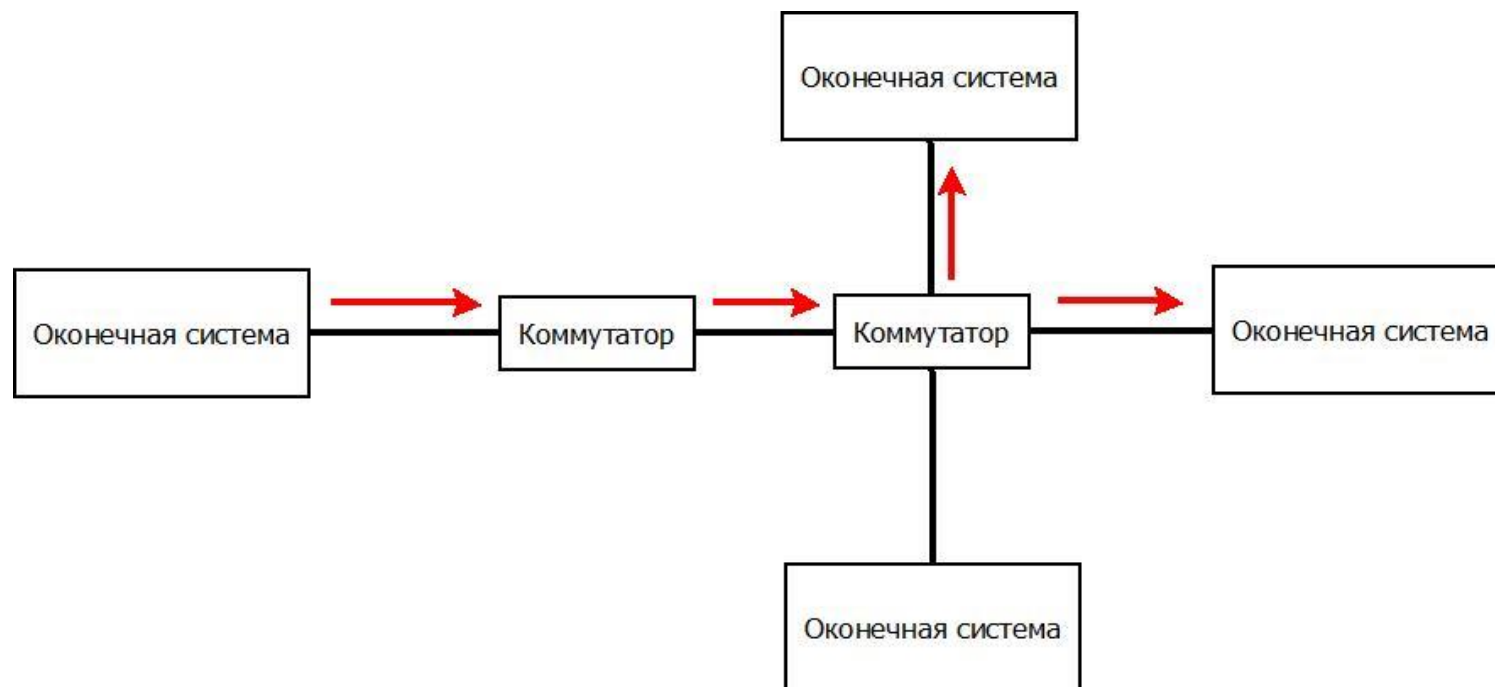


Стек протоколов

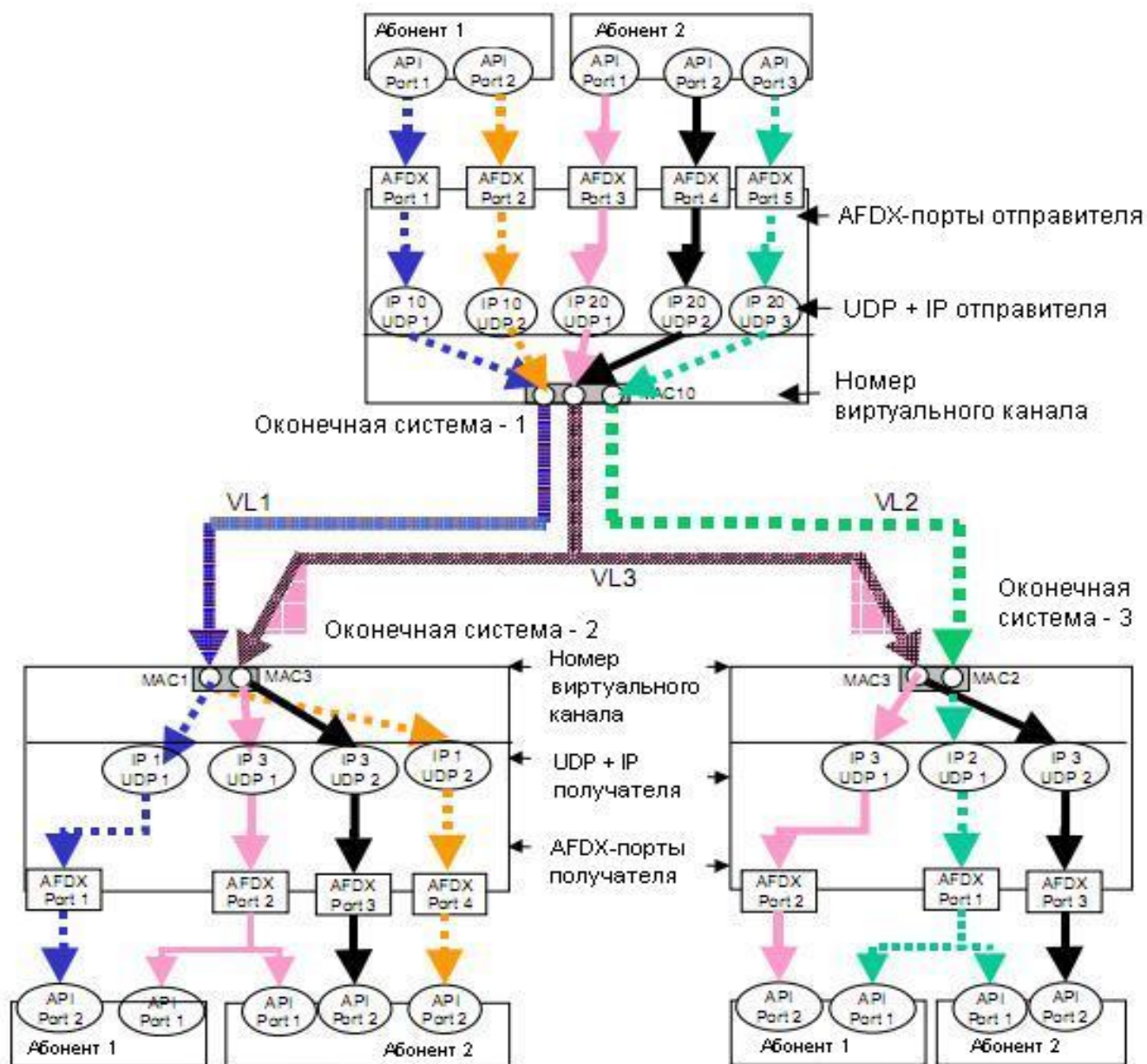
- Канальный уровень
 - Ethernet
 - Виртуальные каналы
 - Маршрутизация
- Сетевой уровень
 - IP (без маршрутизации)
- Транспортный уровень
 - UDP

Стек протоколов

- Виртуальные каналы
 - Одна оконечная система – отправитель; одна или более оконечная система – получатель
 - Маршрут следования кадров виртуального канала прописан статически в коммутаторах



Передача данных

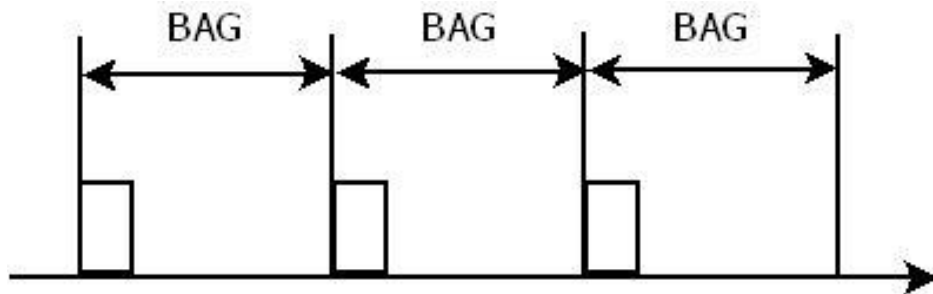


Параметры виртуальных каналов

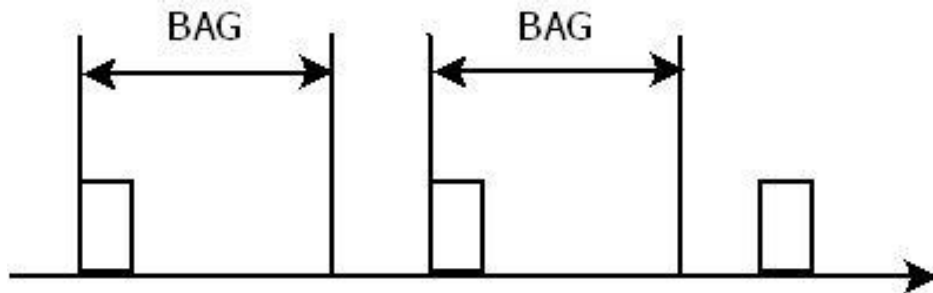
- Для каждого виртуального канала вводятся следующие параметры:
 - *BAG* – Bandwidth Allocation Gap – минимальный интервал времени между началами выдачи последовательных кадров на одном виртуальном канале (1-128 мс, является степенью двойки)
 - *L_{max}* – максимальный размер кадра (≤ 1518 байт)
 - *J_{max}* – максимально допустимое отклонение между кадрами от *BAG*

Параметры виртуальных каналов: *BAG*

- Использование *BAG* для достижения максимальной выделенной пропускной способности:



- Альтернативный вариант:



- В дальнейшем рассматривается только первый вариант передачи, без промежутков между соседними *BAG*-интервалами

Пропускная способность виртуальных каналов

- Вычисление:

- $Bandwidth = L_{max} / BAG$

- $BAG = 32 \text{ мс}$

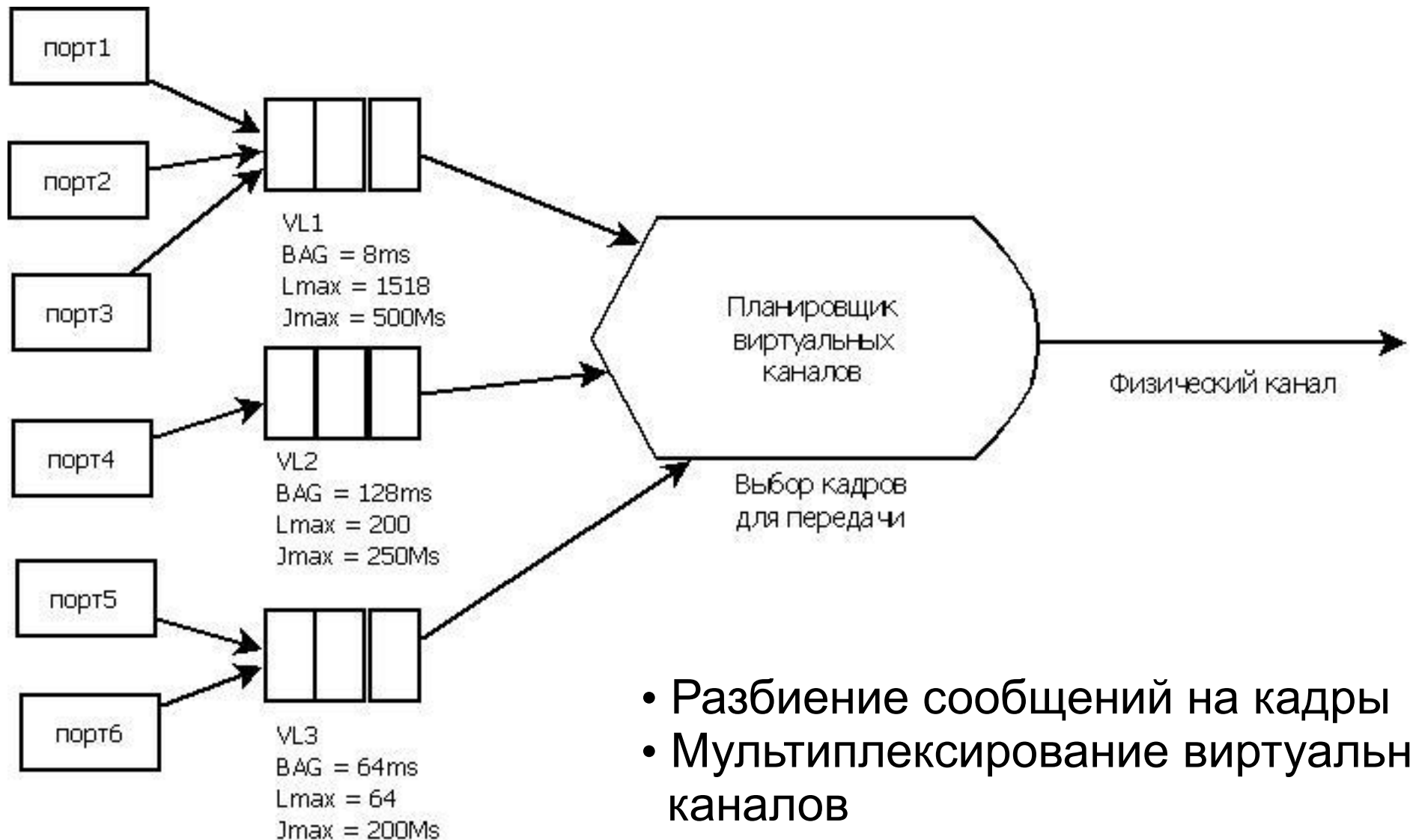
- $L_{max} = 200 \text{ байт}$

- $Bandwidth = 200 \text{ байт} / (32 / 1000) \text{ сек} = 6250 \text{ байт/сек}$

- Ограничение на зарезервированную пропускную способность на физическом канале (проводе):

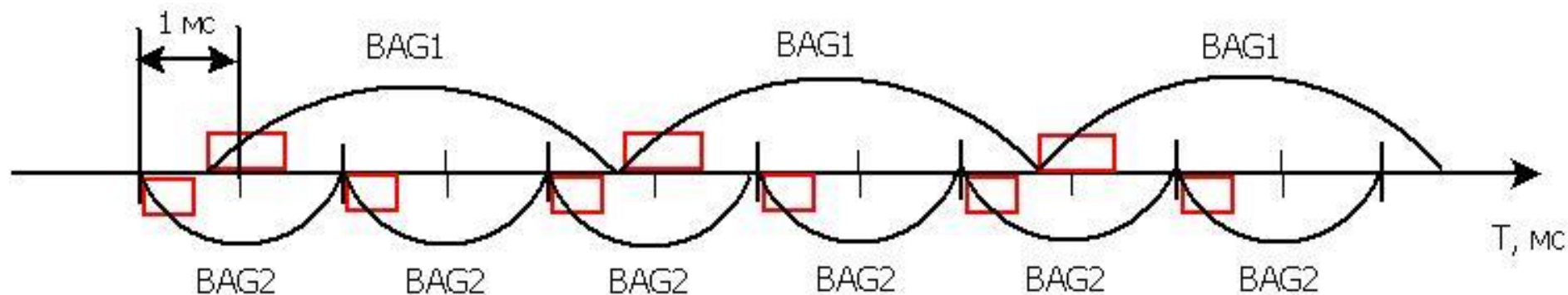
- $\sum_{VL=1..n} L_{VL,max} / BAG_{VL} \leq 100 \text{ Мбит} / \text{сек}$

Управление виртуальными каналами



Формирование трафика

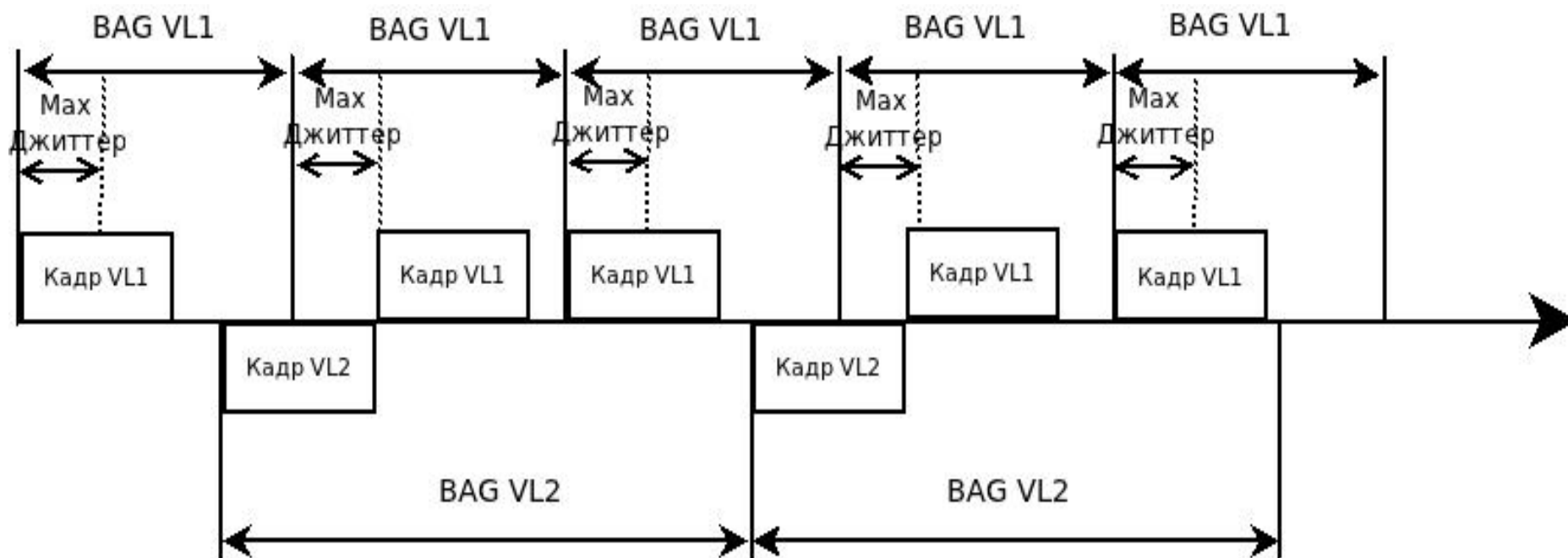
- При формировании трафика на отправителе – мультиплексирование
- При мультиплексировании определяется значение джиттера
- С нулевым джиттером:



Кратность BAG-ов, известность всех L_{max} , отсутствие джиттера *готовности кадров* в каждом ВК
=> исключено взаимное влияние ВК,
отсутствует джиттер *выдачи кадров* в каждом ВК

Формирование трафика

- Мультиплексирование с ненулевым джиттером



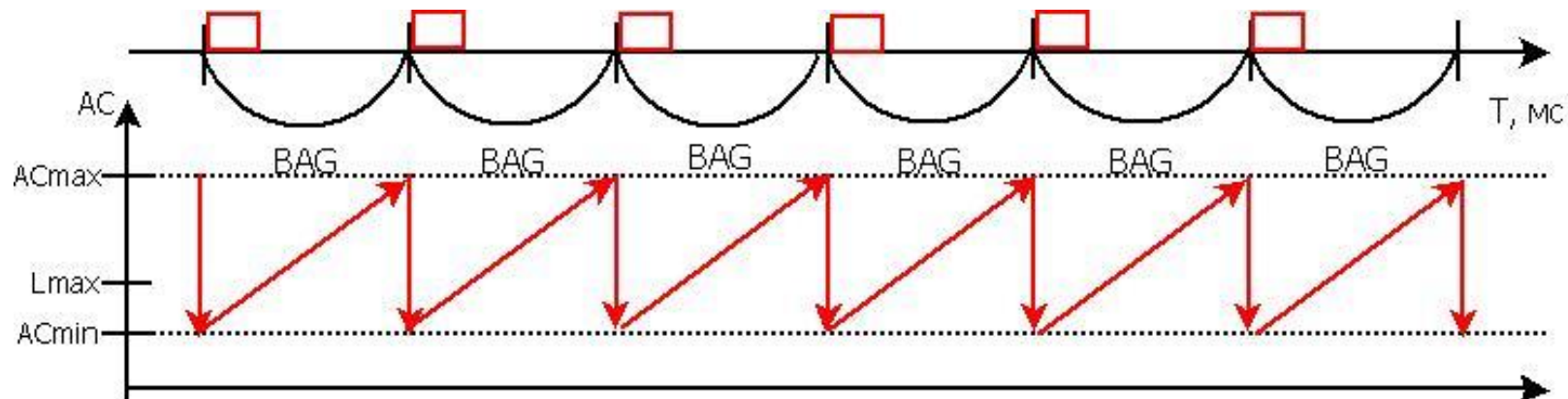
Такого варианта стремятся избегать

Коммутатор

- Функции коммутатора:
 - Маршрутизация кадров по пути следования виртуальных каналов (пути виртуальных каналов конфигурируются статически)
 - Фильтрация трафика (контроль целостности кадра, контроль следования кадра по виртуальному каналу)
 - Контроль трафика
 - размер кадра (не должен превышать L_{max})
 - BAG, J_{max}
 - нарушение => сброс кадра

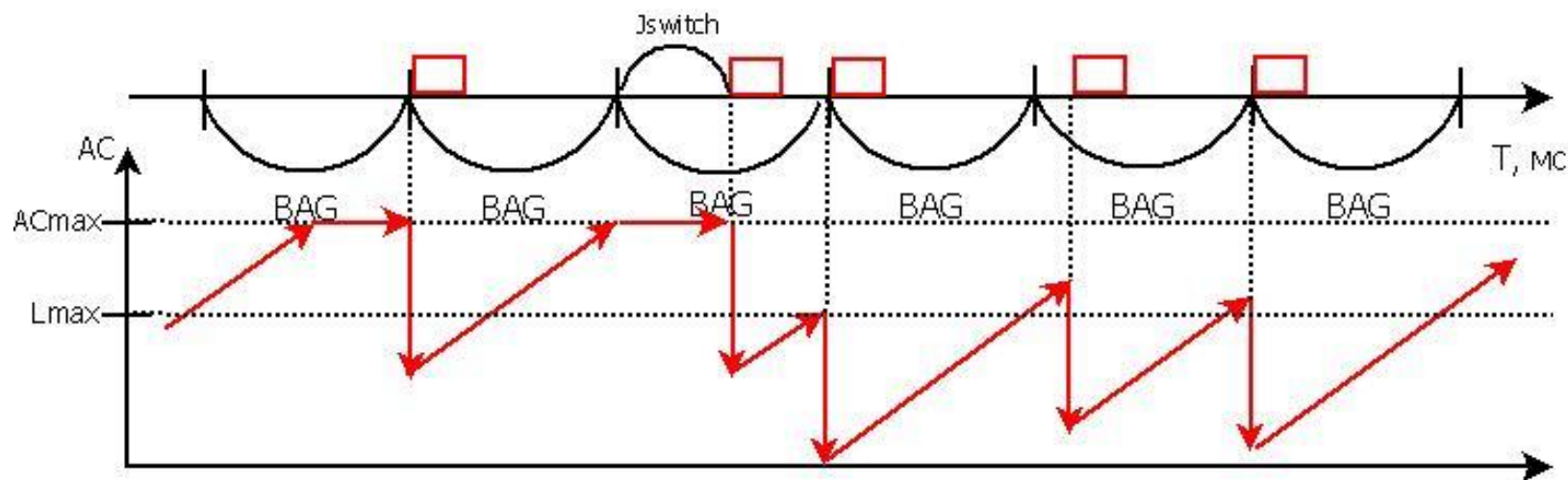
Контроль трафика на коммутаторе

- Контроль времен прихода кадров на соответствие BAG и J_{max} :
 - Производится на входном порту коммутатора
 - Используется алгоритм, основанный на вычислении кредита
 - AC – кредит, растет с течением времени до значения AC_{max}
 - При приходе кадра AC уменьшается на размер кадра; если кредита не хватает – кадр сбрасывается



Контроль трафика на коммутаторе

- Кредит соответствует количеству байт, которые пропускает канал
 - За время BAG кредит увеличивается на L_{max}
 - AC_{max} – соответствует количеству байт, которое позволяет пропустить 2 кадра за $(BAG - J_{switch})$
 - Случай с неравномерной передачей кадров:



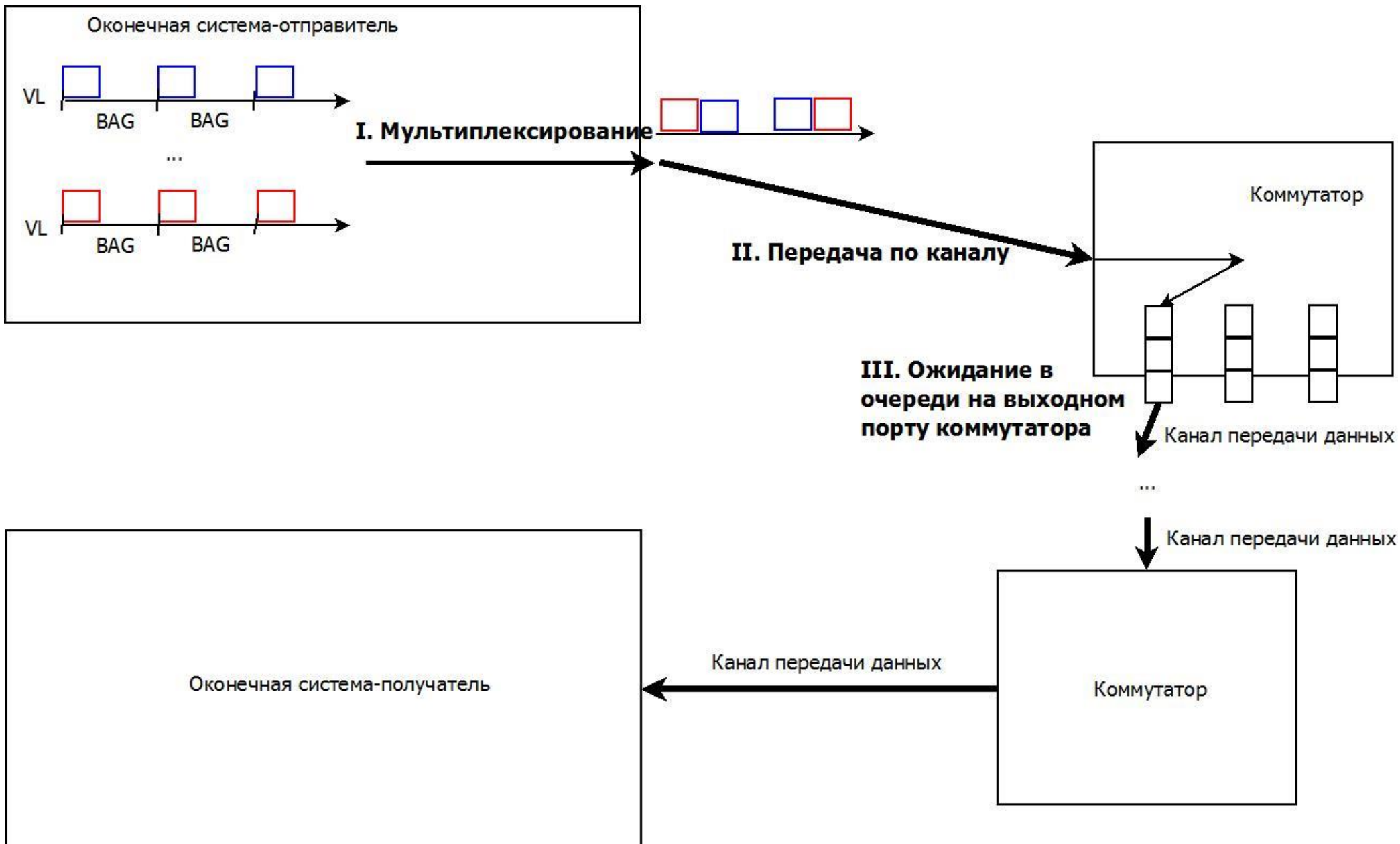
Задачи проектирования сети AFDX

- Дано: потоки данных, требования к их передаче в реальном времени
 - размер сообщения
 - частота передачи
 - макс. допустимый джиттер (end-to-end)
 - макс. допустимая задержка (end-to-end)
- Требуется:
 - построить систему виртуальных каналов:
 - маршруты
 - параметры (BAG, Lmax)
 - рассчитать конфигурационные параметры сетевых устройств – коммутаторов, абонентов (в т.ч. Jmax)

Оценка длительности передачи кадра

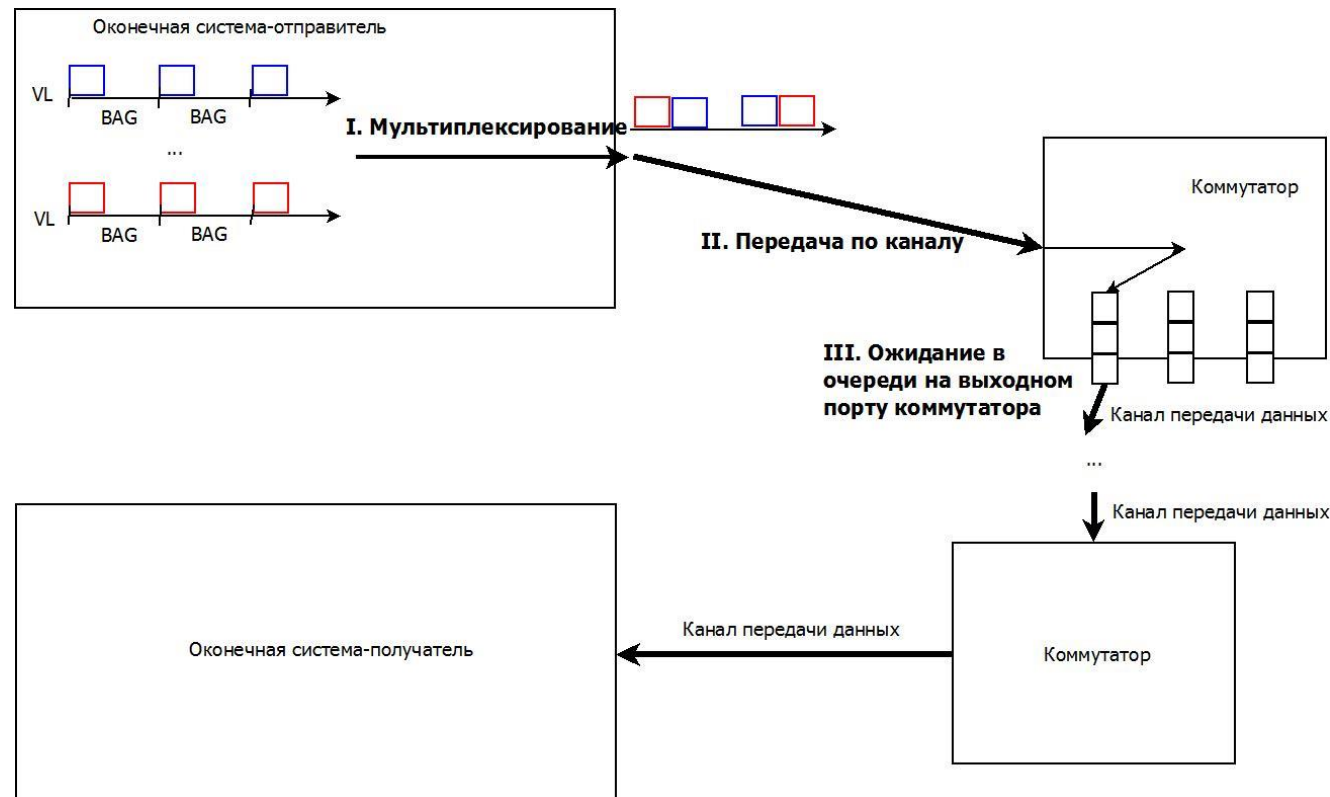
- Необходима для оценки длительности передачи сообщения
 - Актуальность: требования реального времени – длительность не должна превышать заданных значений
 - Длительность вычисляется с момента поступления кадра для выдачи в канал до момента поступления кадра на окончательную систему-получатель

Оценка длительности передачи кадра



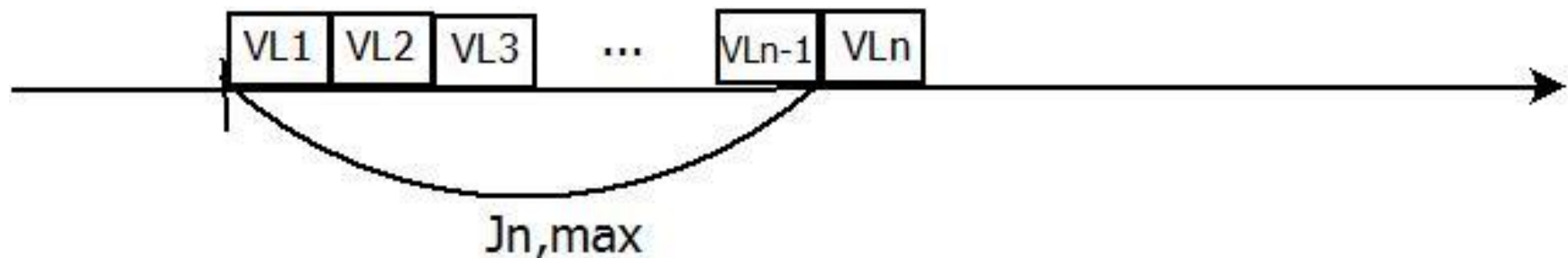
Оценка длительности передачи кадра

- Длительность передачи кадра:
 - максимальный джиттер на отправителе
 - длительность передачи по каналам
 - задержки на выходных портах коммутаторов



Оценка длительности передачи кадра

- Мультиплексирование
 - При мультиплексировании может возникать джиттер
 - Максимальная задержка – при максимальном джиттере
 - Максимальное значение джиттера – при ожидании кадров всех других виртуальных каналов



Оценка длительности передачи кадра

- Мультиплексирование

- Вычисление максимального джиттера на отправителе

$$J_{\max} \leq J_T + \frac{\sum_{i \in VLs} L_{i, \max}}{R}$$

- VLs – множество виртуальных каналов, формируемых на оконечной системе-отправителе
 - R – скорость выдачи данных на канал (100 Мбит/сек)
 - J_T – технический джиттер (время обработки кадра), в AFDX равен 40 мкс

Оценка длительности передачи кадра

- Длительность передачи кадров по каналам
 - R – скорость выдачи данных на канал (100 Мбит/сек)
 - n – количество каналов передачи данных на пути следования кадра
 - формула расчета длительности передачи кадра по каналам:

$$t_{links} = n \cdot \frac{L_{max}}{R}$$

Оценка задержки кадра на коммутаторе

- В очереди перед кадром: кадры с других ВК, идущие на тот же выходной порт
- Сколько этих кадров?
 - интервал накопления M: макс. интервал после предыдущего кадра данного ВК
 - $M = BAG + \text{накопленный макс. джиттер}$
 - число кадров с другого ВК, набежавших за интервал M: см. формулы анализа времени отклика
 - какие-то из набежавших кадров уже выданы в выходной порт к моменту прихода «нашего» кадра ($\Rightarrow$ их выдачи не нужно ожидать)

Оценка длительности и джиттера передачи сообщения

$$Dur_{\max}(msg) = \mu + \delta + \Delta$$

$\mu = \text{const}$ (разбиение и сборка сообщения)

δ - время выдачи сообщения в канал (выдача последнего кадра)

Δ - длительность передачи кадра

$$J(msg) = Dur_{\max}(msg) - Dur_{\min}(msg)$$

Fibre Channel, профиль реального времени

- Базовая схема обмена данными позаимствована из AFDX
 - виртуальные каналы
 - 100% резервирование
- В каждом коммутаторе – несколько таблиц ВК
 - переключение в реальном времени между конфигурациями сети
 - нет избыточного резервирования пропускной способности для поддержки нескольких наборов ВК (втч. в разных режимах ИУС РВ)
 - поддержка статического набора возможных миграций задач
- Управление джиттером отправки целых сообщений
- Система приоритетов → поддержка нерегулярных (апериодических) сообщений
 - низкий приоритет → нет помех для сообщений из ВК
 - высокий приоритет → доставка без задержек (ценой задержки сообщений из ВК)
- Встроенные средства синхронизации времени

Программно-конфигурируемые сети на борту

- Бортовая сеть управляется приложением, функционирующем на контроллере
- Синхронизация времени: например, протокол RTP
- Пассивный режим:
 - правила формируются заранее и загружаются на коммутаторы до старта системы
 - большую часть времени работы ИУС РВ коммутаторы функционируют автономно (без обращения к контроллеру)
- Активный режим:
 - контроллер постоянно осуществляет мониторинг обмена данными
 - обнаружены новые потоки данных → новые правила формируются и загружаются в коммутаторы
- Первый шаг внедрения ПКС на борт: схема выделения пропускной способности
 - воспроизведение схемы на основе алгоритма текущего ведра (по образцу AFDX)?

Активный или пассивный режим?

- Отказоустойчивость / резервирование
 - активный режим: контроллер и управляющая сеть обязательно должны быть продублированы
 - пассивный режим: более высокая устойчивость к сбоям контроллера и управляющей сети
- Реконфигурируемость
 - потоки данных в ИУС РВ являются предсказуемыми
 - правила для заранее определенных режимов могут быть сформированы заранее
 - правила для динамических режимов (в т.ч. при отказах оборудования) могут быть сгенерированы контроллером и в пассивном режиме
- ➔ предпочтителен пассивный режим

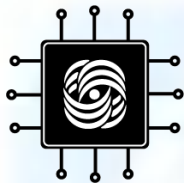
Функциональность приложения для контроллера ПКС в ИУС РВ

- Построение маршрутов передачи сообщений между абонентами сети с обеспечением требуемого качества обслуживания, в т.ч. ограниченных задержек передачи данных и ограниченного джиттера
- Динамическая адаптация маршрутов в случае сбоев сети или миграции задач
- Формирование правил для коммутаторов, в т.ч.:
 - правил проверки трафика на соответствие требованиям качества обслуживания
 - правил маршрутизации
 - правил разделения пропускной способности сети между потоками данных

Востребованность ПКС в ИУС РВ

- Отказоустойчивые системы с миграцией задач
 - сценарии миграции задач при множественных отказах не могут быть просчитаны заранее (комбинаторный взрыв числа сценариев)
 - AFDX: резервирование виртуальных каналов под сбойные режимы непродуктивно расходует пропускную способность сети
 - FC-RT: ограниченный объем памяти коммутаторов под таблицы ВК для сбойных режимов
- Динамическое формирование и смена режимов ИУС РВ
 - реконфигурируемость – «изюминка» ПКС
- Динамическое подключение оборудования и ПО
 - подключаемые в режиме plug-and-play устройства (датчики, устройства связи и т.п.)
 - ПО загружается с подключаемого устройства и выполняется на бортовом вычислителе с архитектурой ИМА в отдельном разделе
 - ВК для обмена бортового вычислителя с подключенным устройством автоматически настраиваются контроллером ПКС
 - пример: многофункциональный беспилотник со сменными датчиками

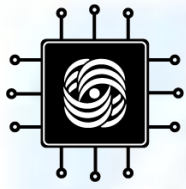
Спасибо за внимание!



ИНФОРМАЦИОННО-УПРАВЛЯЮЩИЕ СИСТЕМЫ РЕАЛЬНОГО ВРЕМЕНИ

Лекция 10: *Обеспечение отказоустойчивости ИУС РВ*

Кафедра АСВК,
Лаборатория Вычислительных Комплексов
Балашов В.В.



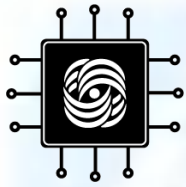
Цена отказа: Ariane-5

(причина: неисправность ПО)

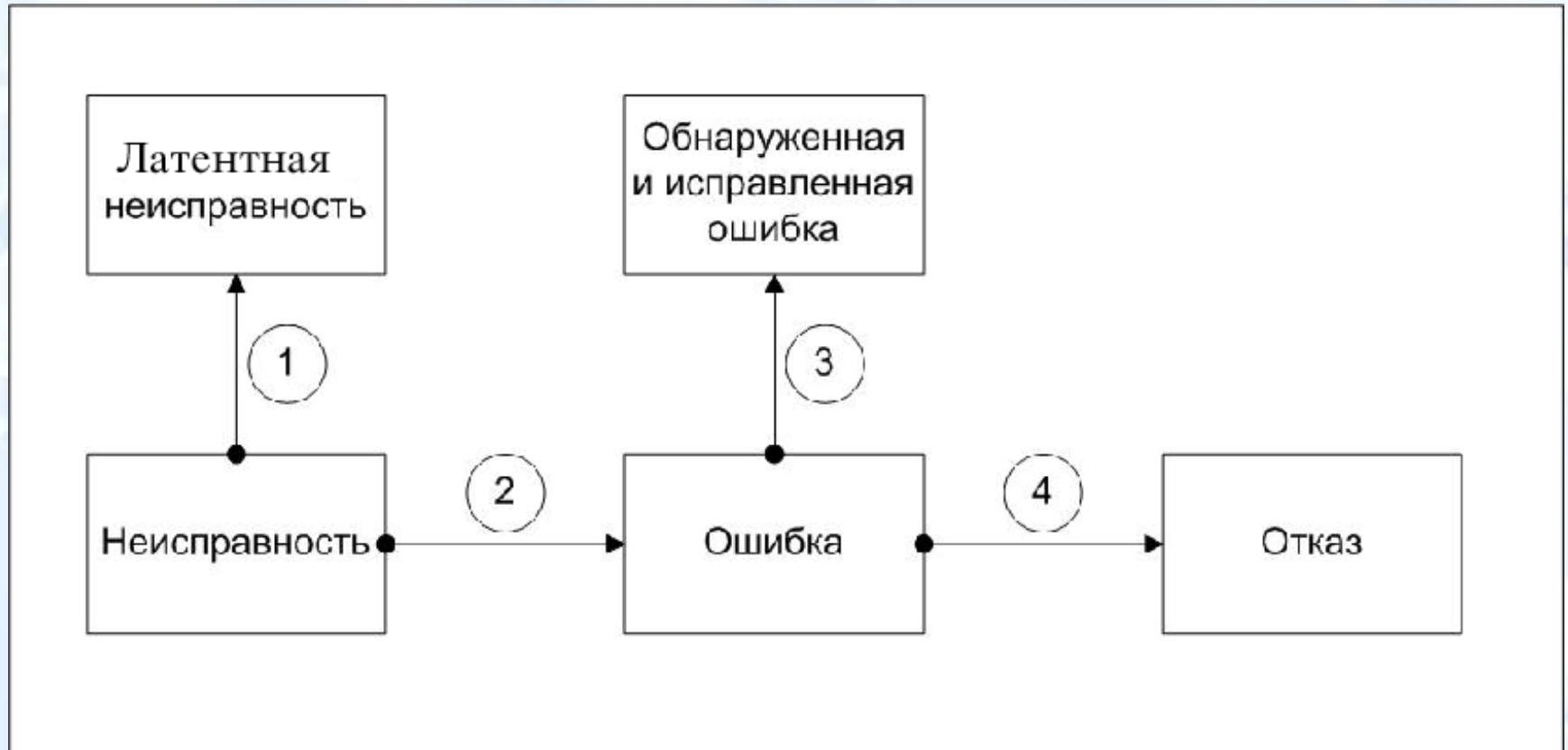
- Июнь 1996 года, взрыв ракеты спустя 40 сек. после старта,
- Ущерб – \$500млн (разработка – \$7 млрд.),
- Причина – 64bit float -> 16bit int.



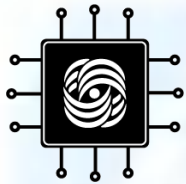
*Кажется,
что-то
пошло не
так...*



Неисправность, ошибка, отказ



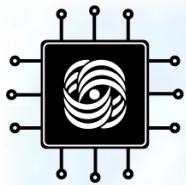
- Полностью избежать неисправностей невозможно
- Цель: минимизировать вероятность отказа



Классификация неисправностей

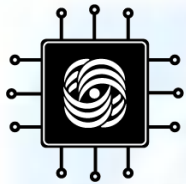
- Активность: латентные / активные
- Постоянство: проходящие / постоянные
- Источник: внешнее воздействие / ошибка разработки
- Распространение последствий: обнаруживаются и локализуются / проникают в другие подсистемы
- Одиночность: одиночные / групповые
- Взаимосвязанность: независимые / связанные

(источник: Software Fault Tolerance: A Tutorial, NASA, 2014, pp.28-29, <http://www.iet.unipi.it/c.bernardeschi/didattica/ANNO2014-15/DEP/SoftwFT.pdf>)



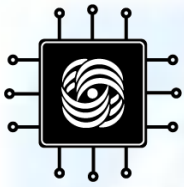
Шаги противодействия неисправности

- Обнаружение
- Ограничение распространения
- Маскировка
- Диагностика
- Восстановление
- Возобновление штатного функционирования



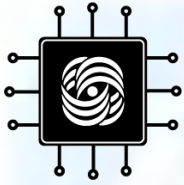
Классификация неисправностей (2)

- По локализации
 - Программные
 - Аппаратные
- По этапу возникновения
 - Проектирование/разработка
 - => все изделия, созданные по проекту
 - Производство
 - Дефект серии => все изделия серии
 - Дефект при производстве конкретного изделия
 - Эксплуатация
 - => одиночные изделия, с учётом особенностей эксплуатации



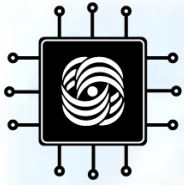
Борьба с серийными неисправностями

- Возникают на этапе проектирования, разработки или серийного производства
 - На ранних этапах их и следует обнаруживать...
- Затрагивают всю серию компонентов
 - Отзыв серии и всех использующих её систем...
- Борьба: использование проектного или реализационного разнообразия
 - Аппаратура: различные архитектуры, производители и элементная база
 - ПО: различные языки, алгоритмы/подходы, команды разработчиков



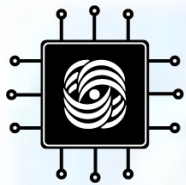
Специфика ИУС РВ

- Жесткие условия эксплуатации
 - Внешние воздействия вызывают неисправности оборудования
- Недопустимость прекращения функционирования при возникновении ошибки
 - Ошибка = реализация неисправности
- Невозможность оперативного ремонта (или ремонта вообще)
 - Самолет, спутник
- Реакция на ошибки должна укладываться в директивные сроки



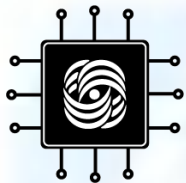
Принципы построения отказоустойчивых систем

- Недопустимость единственной точки отказа
- Поддержка локализации отказа (обнаружения отказавшего компонента)
- Нераспространение последствий отказа далее по системе
 - Защита аппаратуры
 - Блокировка распространения некорректных результатов вычислений
- Постепенная деградация
 - По мере отказа подсистем, вначале отключаются второстепенные функции
 - Функции, критические для выживания/восстановления системы поддерживаются «до последнего»
 - Защитный режим: поддержка существования + обеспечение удалённого доступа для диагностики и обслуживания

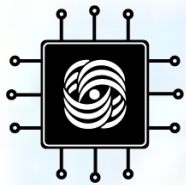


Механизмы обеспечения отказоустойчивости

- Аппаратные
- Программные
- Аппаратно-программные

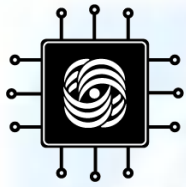


Аппаратные МОО



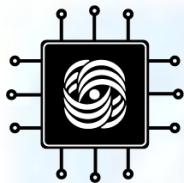
Аппаратное резервирование

- По разнообразию компонентов
 - Использование идентичных компонентов
 - Борьба с дефектами изделия, в т.ч. возникающими в ходе эксплуатации
 - Использование различных компонентов
 - Функционально идентичны
 - Различная элементная база, производитель, проект и т.п.
 - Борьба с дефектами проекта, серии (не только изделия)
- По уровню
 - Система/подсистема
 - Отдельные компоненты

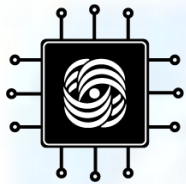


Аппаратное резервирование

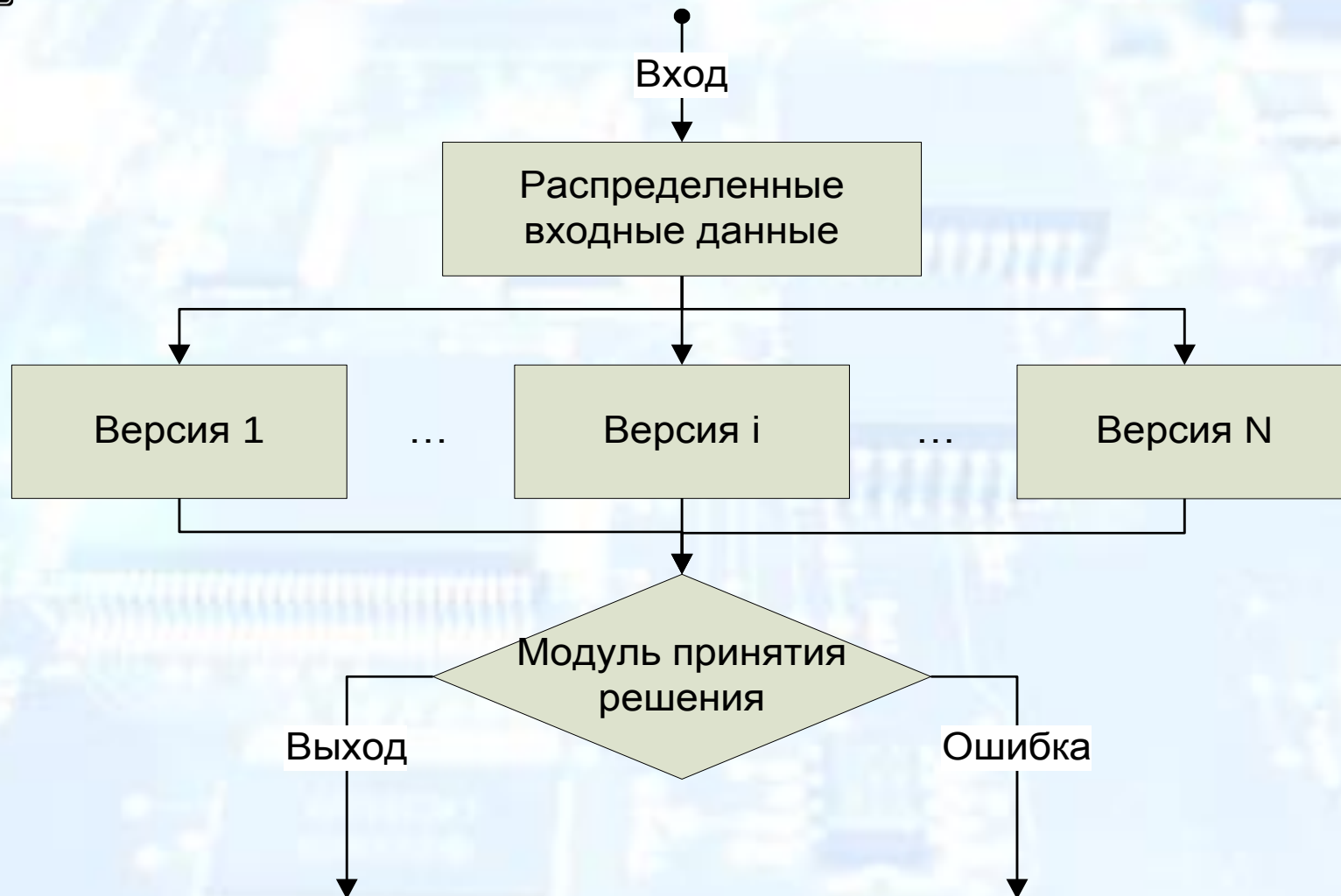
- Активное: основные и резервные компоненты функционируют одновременно
 - Синхронизация данных
 - Минимальное время переключения на резервный компонент
 - Использование результатов:
 - Игнорирование результатов резервных компонентов
 - Голосование (нет выделенного основного компонента)
 - Повышенное энергопотребление
- Пассивное: резервный компонент включается при выходе основного из строя
 - Затраты времени на инициализацию
 - Проблема записи состояния основного компонента (нужно внешнее запоминающее устройство)
 - Экономия энергии
- Жаргон: «горячий» и «холодный» резерв



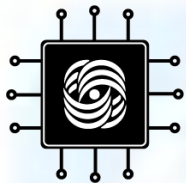
Программные МОО



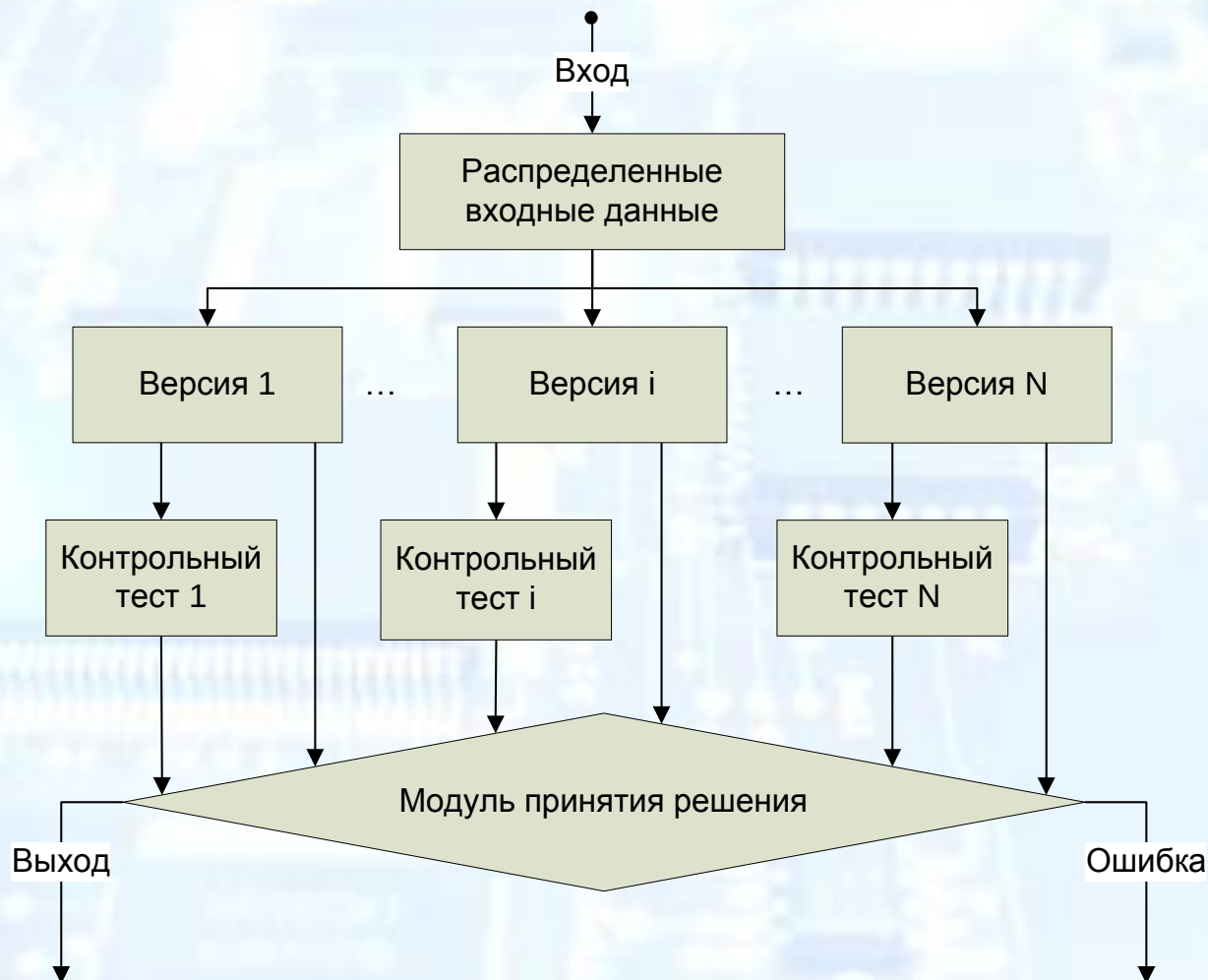
N-версионное программирование



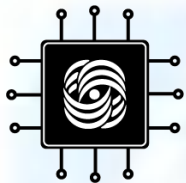
- Версии ПО функционально эквивалентны
- Различаются: алгоритмы, методы разработки, языки программирования, группы разработчиков



N-самотестируемое программирование

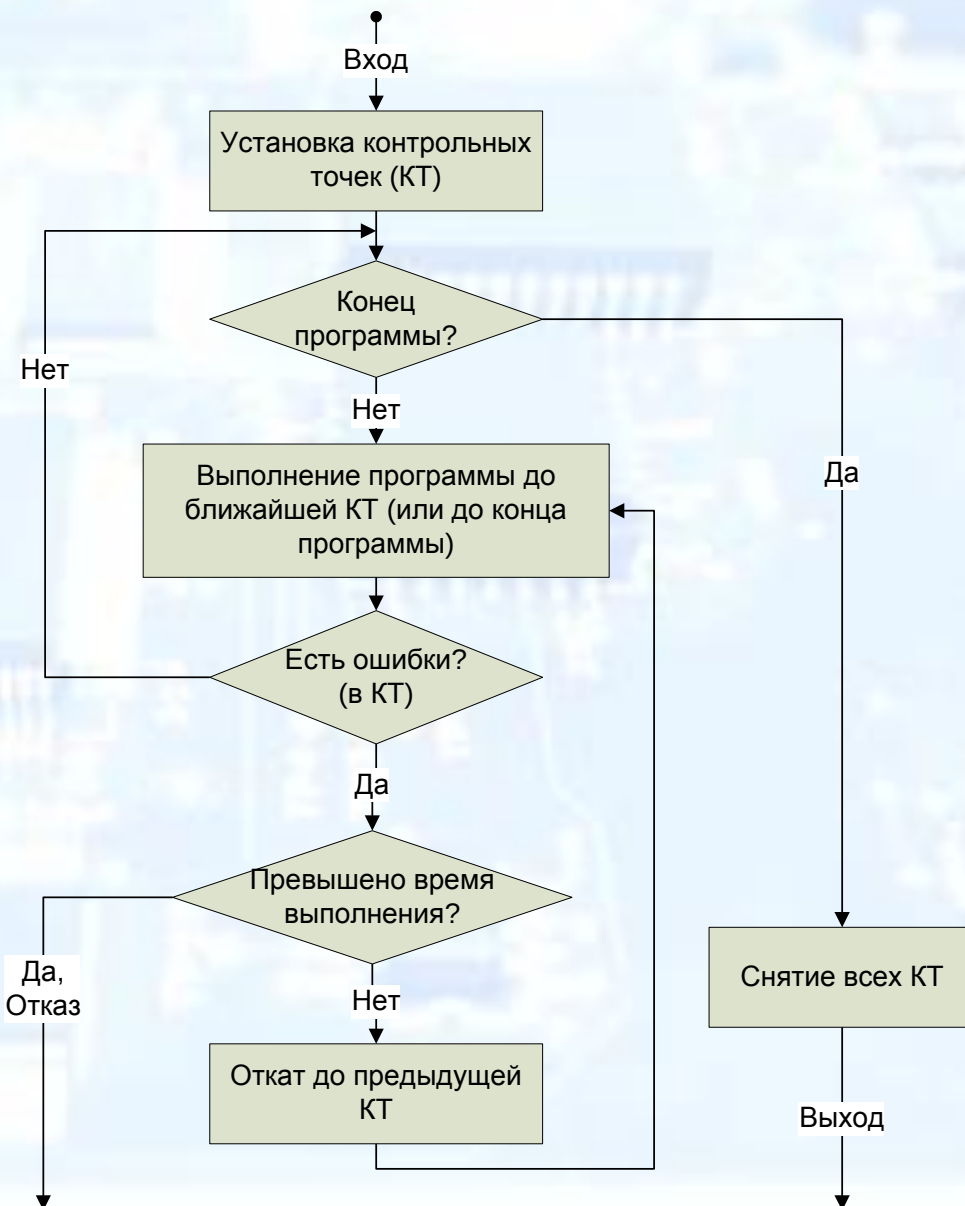


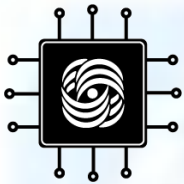
- Версии ПО запускаются последовательно (до первого результата, прошедшего контрольный тест) или параллельно (голосование среди успешных результатов)



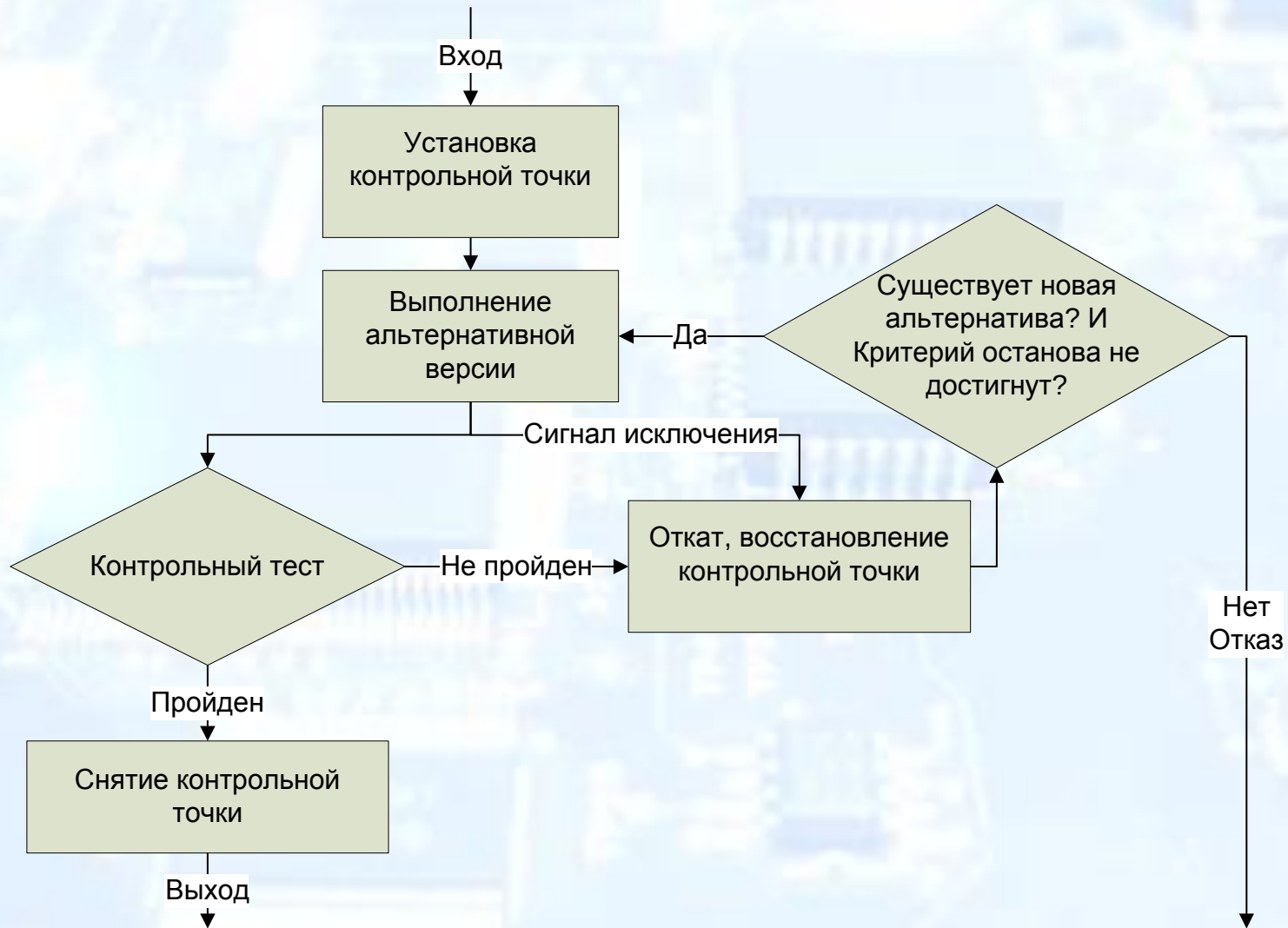
Контрольные точки

- Борьба только со случайными «однократными» неисправностями (инверсия бита памяти и т.п.)
- Не помогает от «постоянных» неисправностей, в т.ч. в ПО
- Затраты ресурсов на сохранение состояния в КТ, сложность механизмов поддержки сохранения

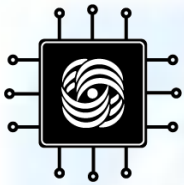




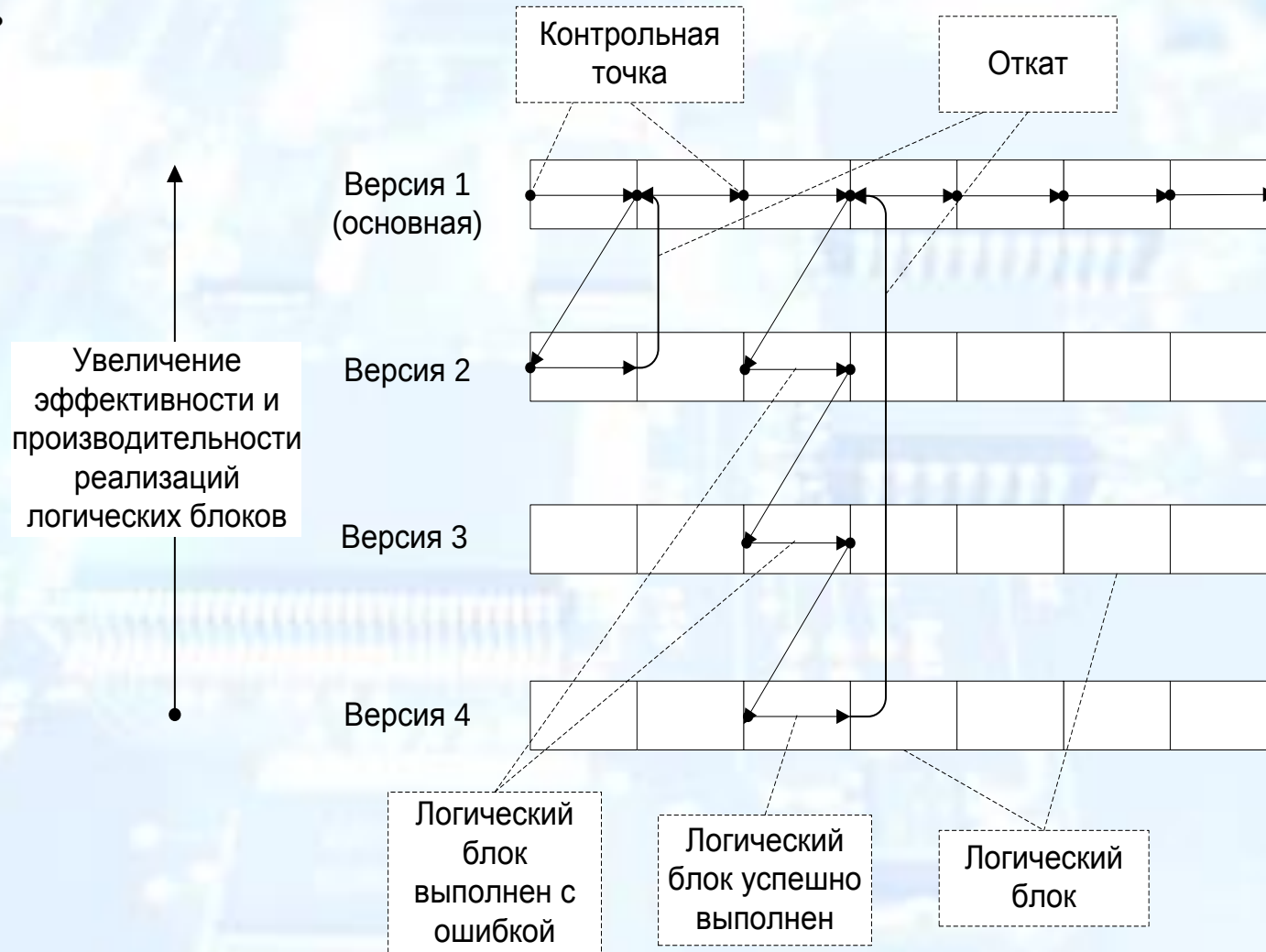
Восстановление блоками



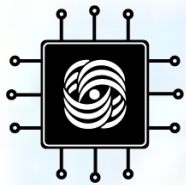
- Сочетает N-версионное программирование и контрольные точки



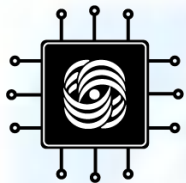
Восстановление блоками



- Перебор блоков по убыванию «скорости»
- WCET оценивается по наихудшему пути...

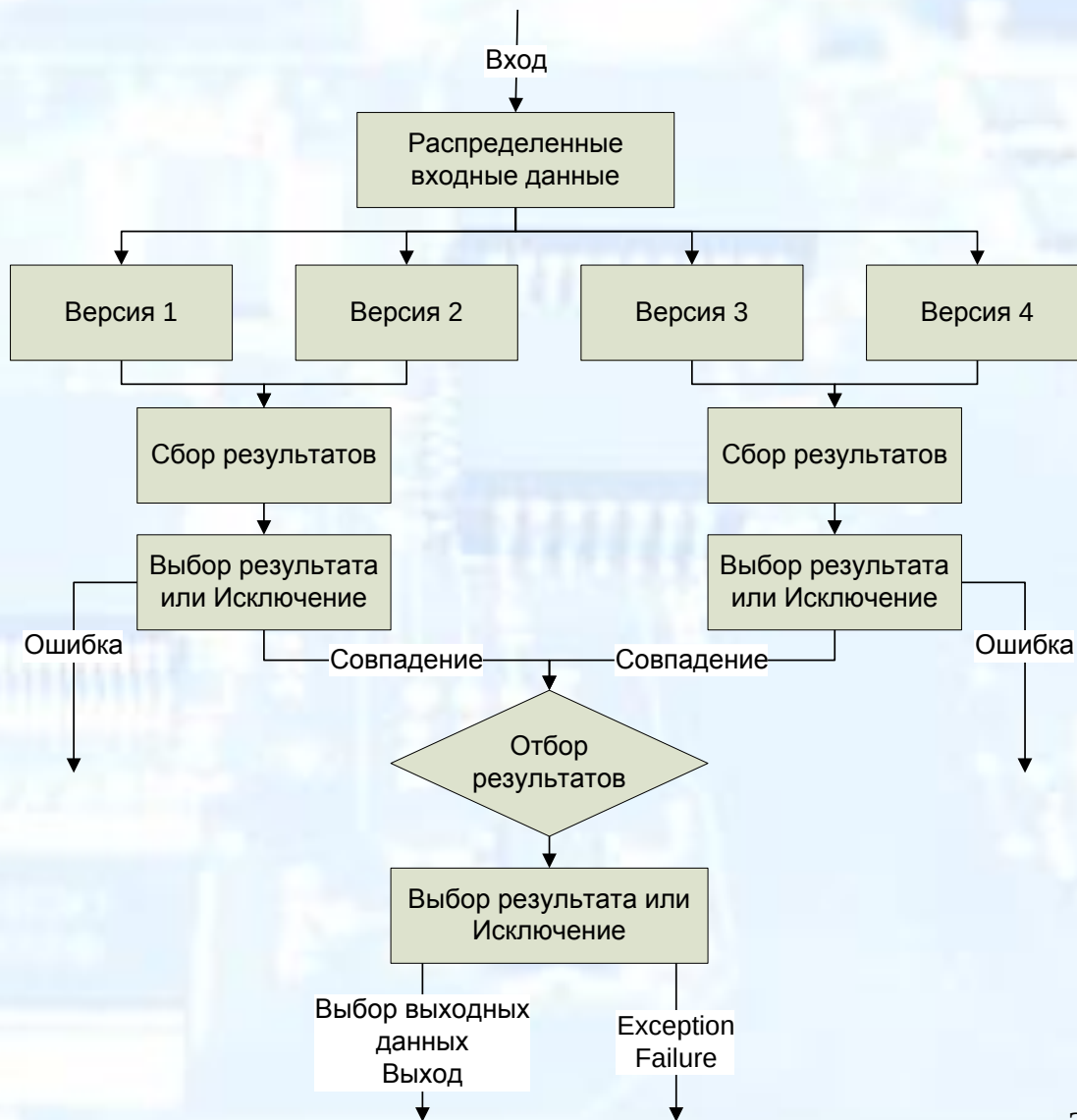


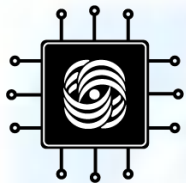
Программно-аппаратные МОО



N-самоконтролируемое программирование

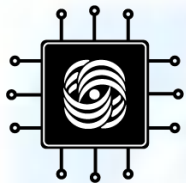
- Борьба с аппаратными и программными ошибками
- ПО в каждой паре: одна версия + контрольный тест, или более одной версии + алгоритм выбора



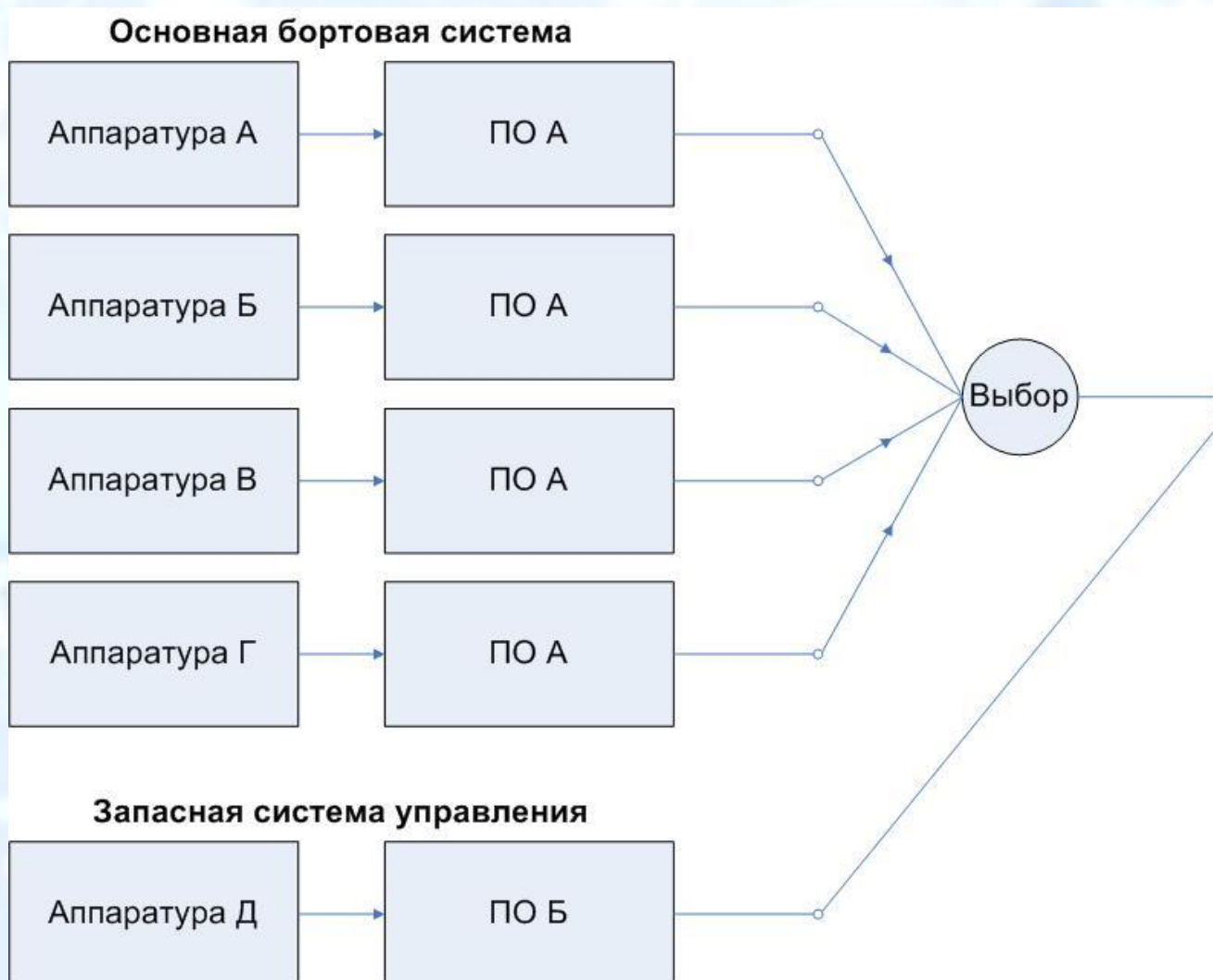


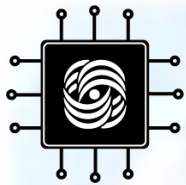
Активное резервирование + N-версионное программирование

- На каждом аппаратном компоненте выполняется своя версия программы
- Все компоненты активны
- Борьба с программными и аппаратными неисправностями

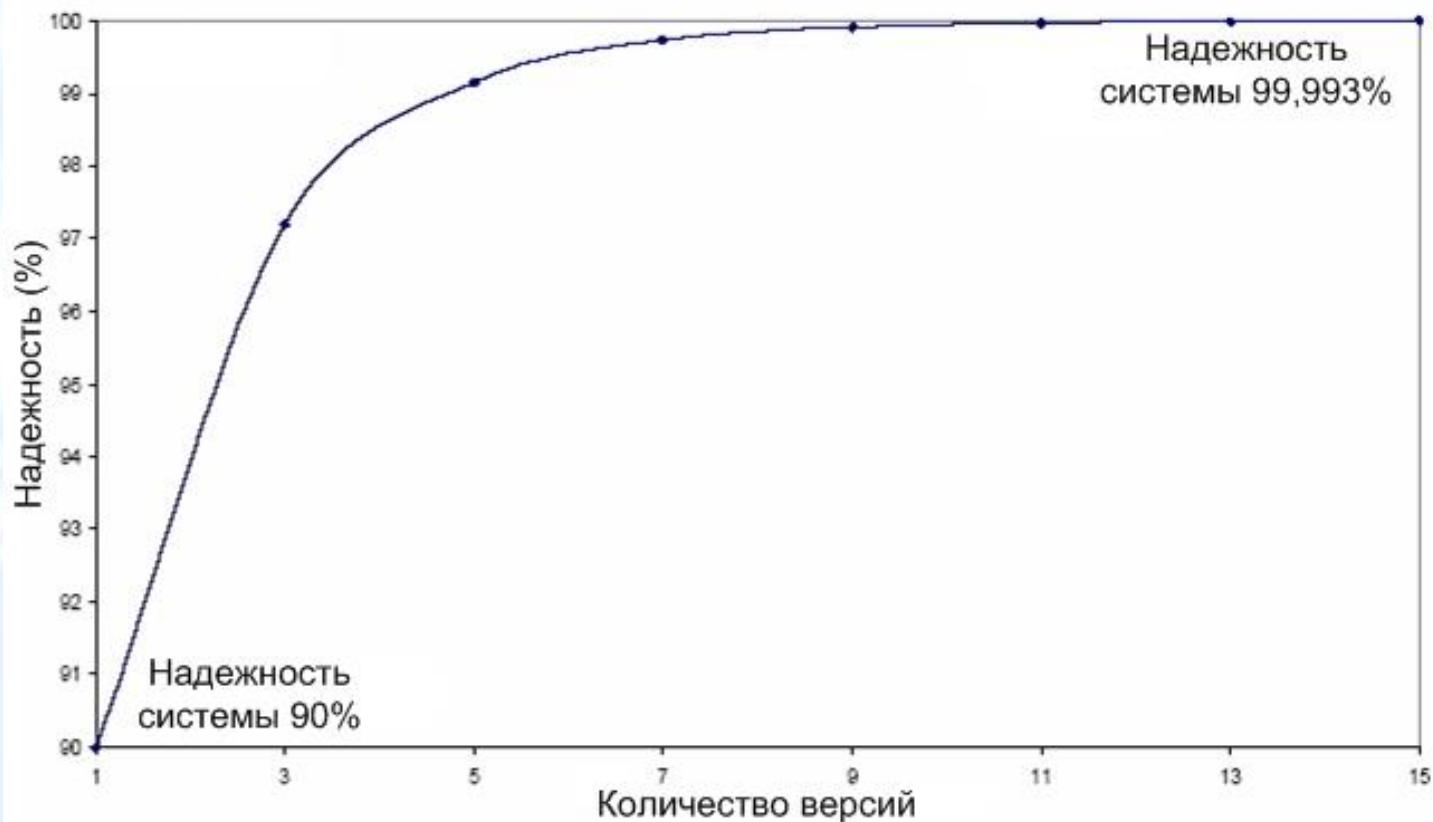


Космический челнок





Зависимость надежности от количества версий



Принципы проектирования безопасных и отказоустойчивых систем

1. Требования отказоустойчивости и безопасности должны быть неотъемлемой частью спецификации системы, а для некоторых систем – основой для процесса проектирования.
2. Ожидаемые виды отказов и частоты их возникновения должны быть определены в начале процесса проектирования.
3. Должны быть определены области локализации неисправностей в системе (fault containment regions). Последствия неисправности в одной из таких областей не должны распространяться на другие области.

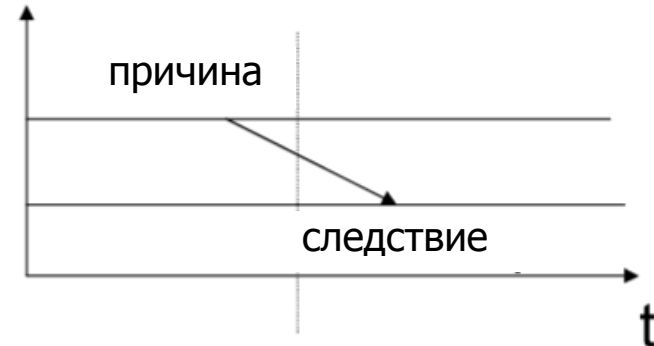


Салон
цел

Двигатель и
электронику
не жалко

Принципы проектирования безопасных и отказоустойчивых систем

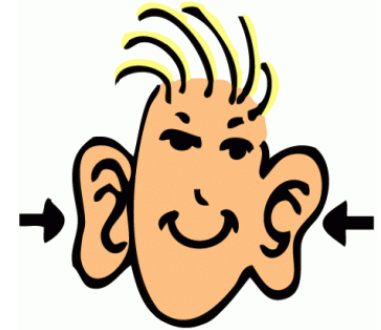
4. Понятие времени и состояния системы должны быть строго определены. В противном случае затруднительно отличить первичную неисправность от ее последствий.
5. Необходимо четко определить интерфейсы, чтобы скрыть внутреннее устройство компонентов системы.
6. Необходимо обеспечить независимость возникновения неисправностей в различных компонентах.



Две независимые
тормозные
системы

Принципы проектирования безопасных и отказоустойчивых систем

7. Компонент системы должен считать себя корректно функционирующим, пока два или более других компонента не примут противоположную точку зрения.
8. Механизмы обеспечения отказоустойчивости должны быть устроены так, чтобы не усложнять анализ поведения системы. МОО должны быть отделены от основной функциональности системы.
9. Возможность диагностирования должна быть заложена в систему на этапе проектирования. В частности, должна обеспечиваться возможность выявления скрытых неисправностей, не проявляющихся в поведении системы.



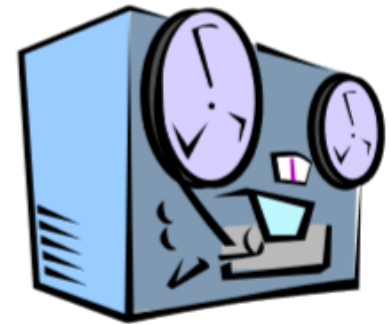
Принципы проектирования безопасных и отказоустойчивых систем

10. Интерфейс с оператором должен быть интуитивно понятным и устойчивым к ошибкам. Безопасность должна обеспечиваться несмотря на ошибки человека-оператора.



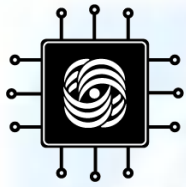
Подушка безопасности

11. Любая аномалия функционирования системы должна быть зарегистрирована. Некоторые аномалии не обнаруживаются на уровне интерфейсов между компонентами. Регистрация должна фиксировать в т.ч. внутренние аномалии, в противном случае они могут быть замаскированы в результате работы МОО.



12. Система должна реализовывать стратегию «никогда не сдавайся», гарантируя заданный минимальный уровень обслуживания. Полный выход из строя крайне нежелателен.



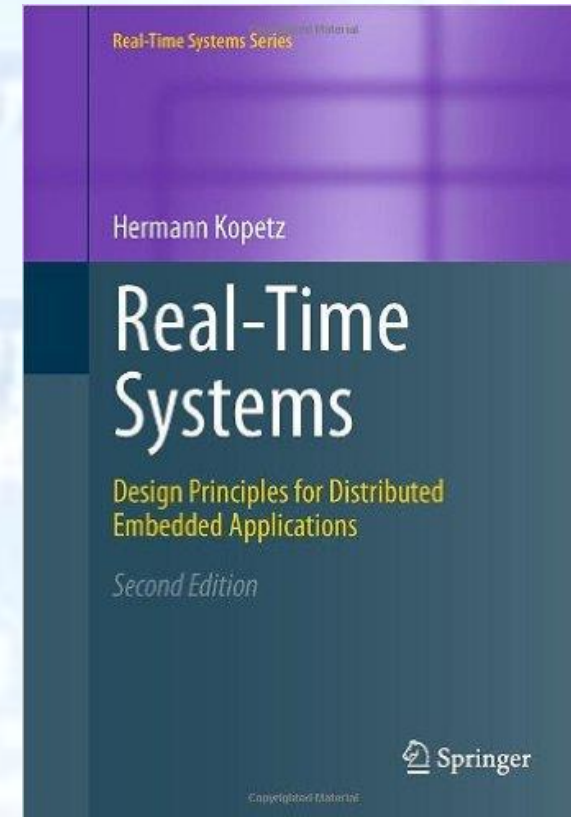


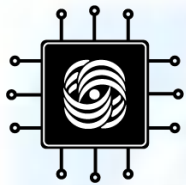
Принципы проектирования безопасных и отказоустойчивых систем

Источник:

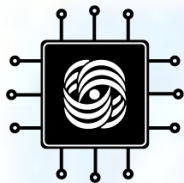
“Real-Time Systems: Design Principles for Distributed Embedded Applications”
(Real-Time Systems Series),
H. Kopetz, Springer, 2011

https://vowi.fsinf.at/images/temp/2/2c/20110606133809!TU_Wien-Echtzeitsysteme_VO_%28Kopetz%29_-_TU_Wien-Echtzeitsysteme_VO_%28Kopetz%29_-_TU_Wien-Echtzeitsysteme_VO_%28Kopetz%29_-_Real_Time_Systems_-_Design_Principles_for_Distributed_Embedded_Applications_--_Hermann_Kopetz_--_2._Edition.pdf





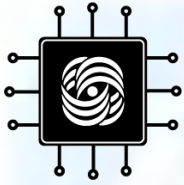
Спасибо за внимание!



ИНФОРМАЦИОННО-УПРАВЛЯЮЩИЕ СИСТЕМЫ РЕАЛЬНОГО ВРЕМЕНИ

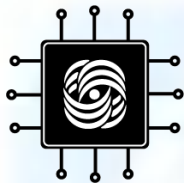
Лекция 11: *Тестирование и интеграция ИУС РВ*

Кафедра АСВК,
Лаборатория Вычислительных Комплексов
Балашов В.В.

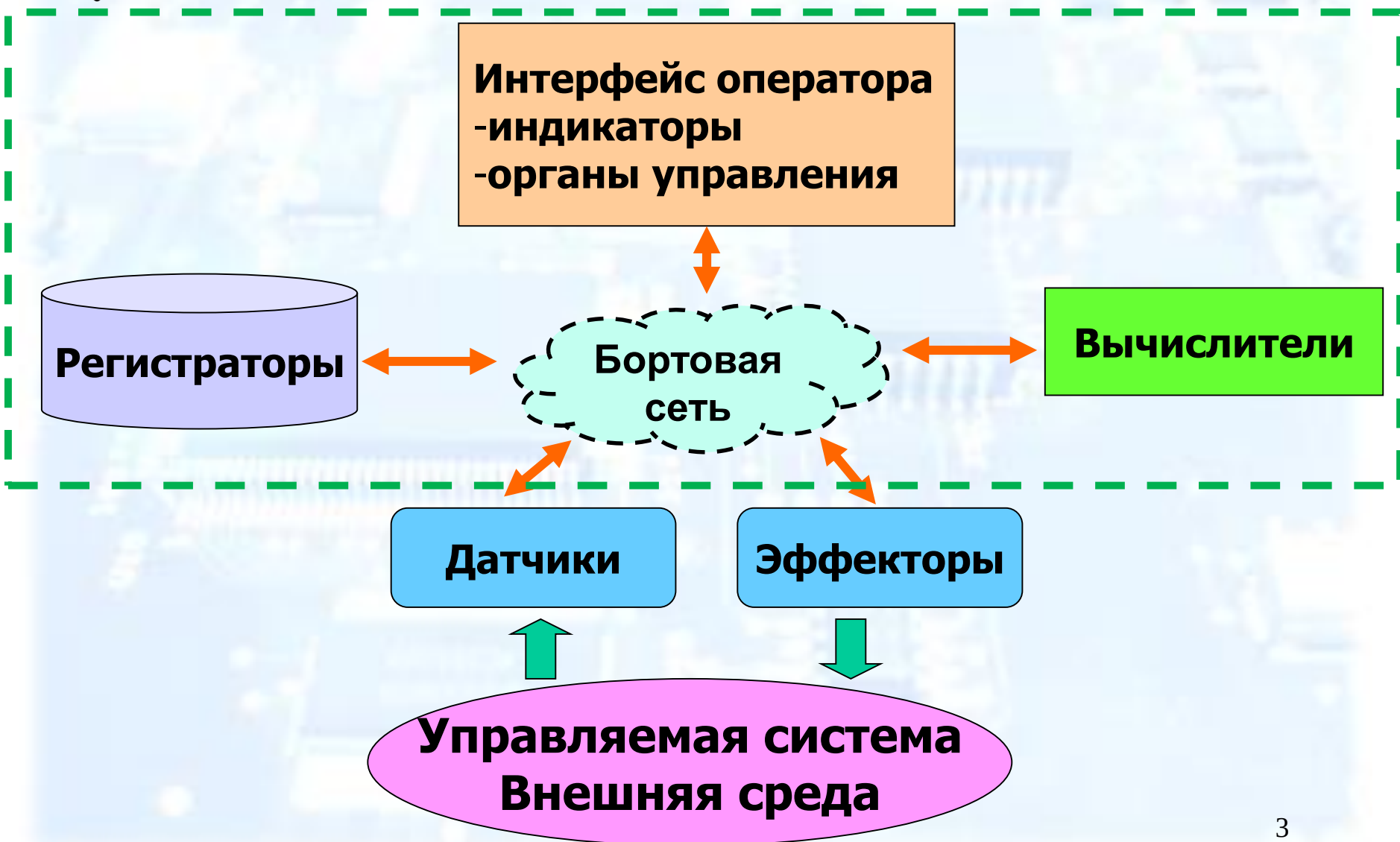


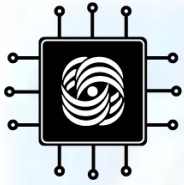
ИУС РВ

- Информационно-управляющая система (ИУС) – вычислительная система верхнего уровня, обеспечивающая:
 - функциональную и информационную интеграцию составных частей управляемого объекта
 - взаимодействие между объектом и оператором



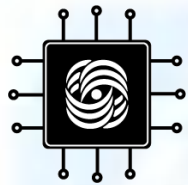
Состав ИУС



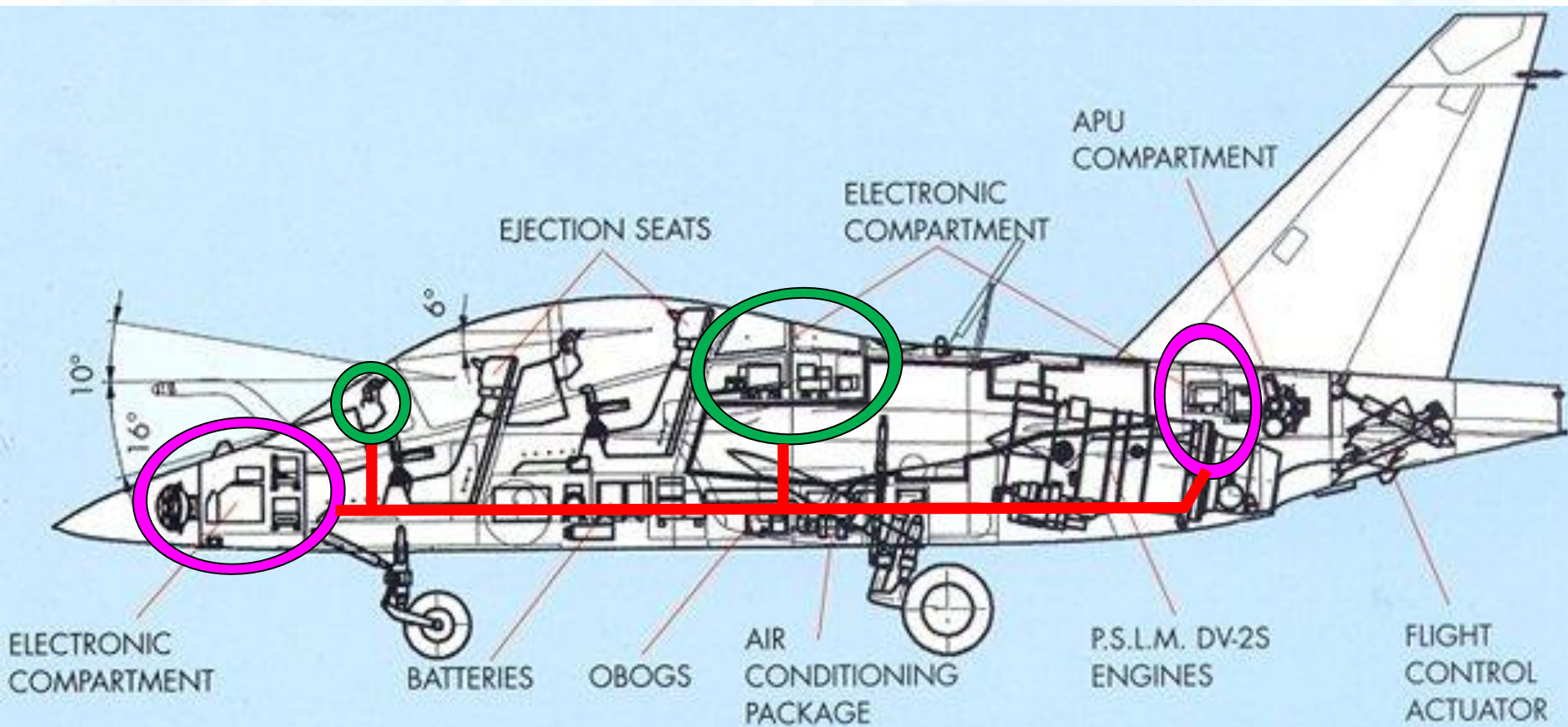


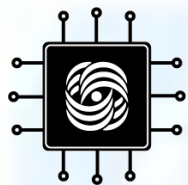
Функции ИУС

- Контроль состояния управляемого объекта
- Управление движением объекта или его частей
- Отслеживание положения объекта или его частей в пространстве
- Обмен данными с внешними системами
- Управление специализированными приборами (прикладной нагрузкой)
- Обмен данными с оператором
 - отображение данных
 - ввод данных



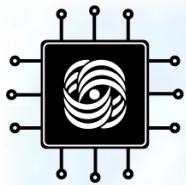
ИУС в управляемой системе





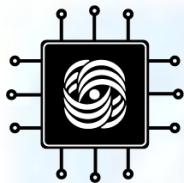
Устройства в составе ИУС





Устройства вне ИУС (на примере самолета)

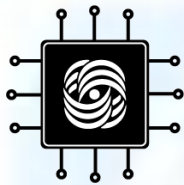
- Датчики:
 - Радиолокатор
 - Радиовысотомер
 - Топливный датчик
 - Тахометр (измеритель частоты вращения турбины)
 - Датчики спутниковой навигации
- Эффекторы
 - управление двигателем
 - управление рулями высоты
 - управление выпуском шасси
 - радиопередатчик



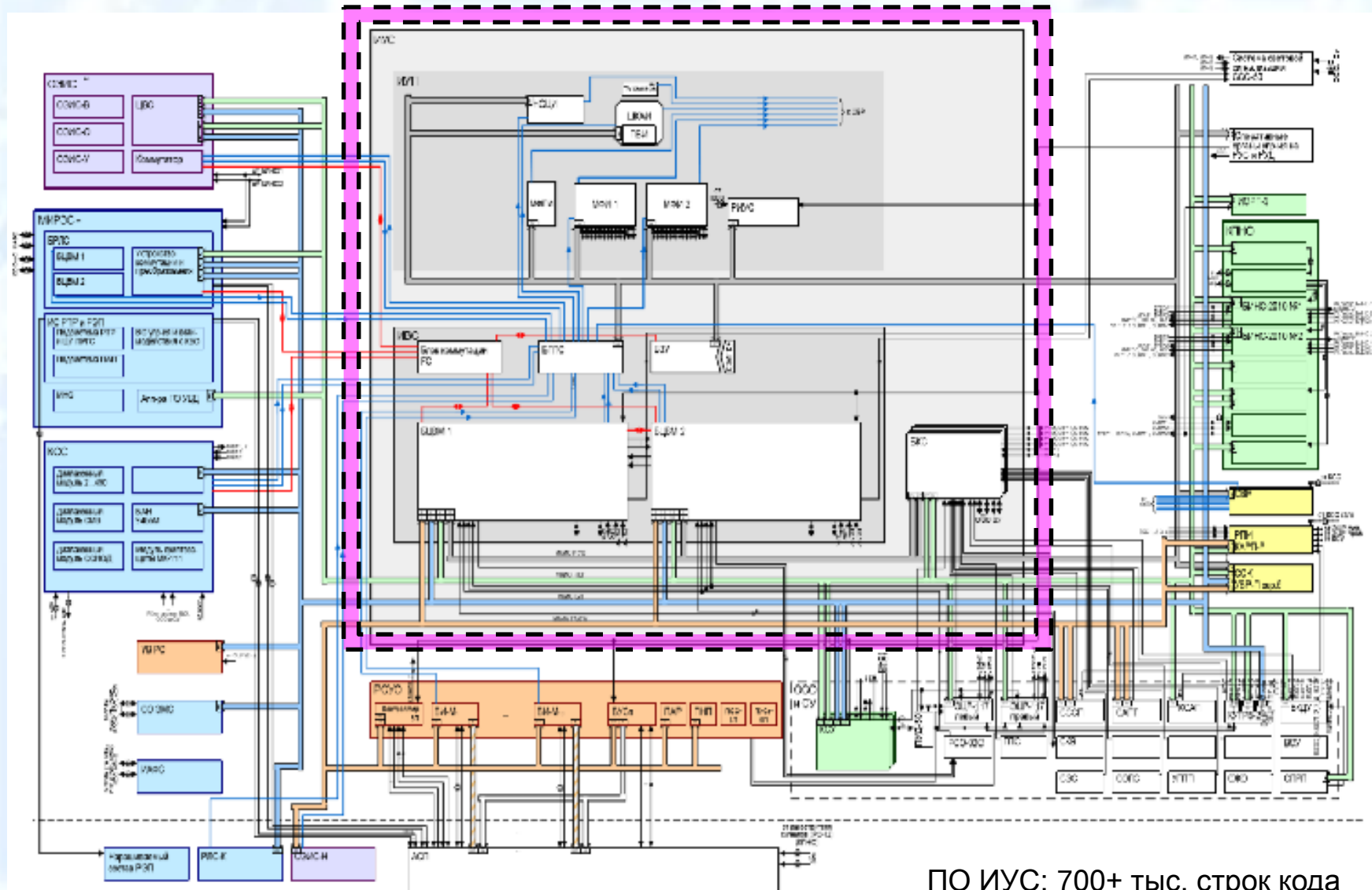
Неоднородность ИУС

- Каналы: точка-точка, шина, коммутатор; 12 kbps, 1 Mbps, 1 Gbps
- Устройства: датчики, индикаторы, вычислители, органы управления, исполнительные устройства
- Данные: аналоговые, цифровые; числовые массивы, видеопотоки

Проблема унаследованных устройств



Уровень сложности комплекса бортового оборудования



ПО ИУС: 700+ тыс. строк кода



Жизненный цикл ПО ИУС

Входные данные на следующий инкремент (дополнительный функционал)

Базовый, повторяющийся элемент инкрементальной модели ЖЦ

Замечания на версию

Фаза планирования

Ввод системы в эксплуатацию

Разработка системных требований

Системное тестирование

Разработка архитектуры системы

Системная интеграция

Разработка требований на ПО высокого уровня

Функциональное тестирование

Разработка требований на ПО низкого уровня

Тестирование компонентов ПО

Кодирование, отладка и интеграция ПО

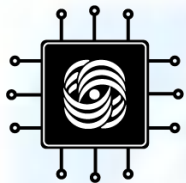
Тестирование, интеграция, отработка

Информационное сопряжение

Специфика инкрементальной модели

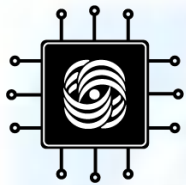
Системные активности

Активности по разработке ПО



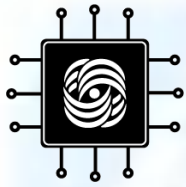
Требования к средствам тестирования ИУС

- Поддержка тестирования ПО на целевом вычислителе без инструментирования аппаратуры или ПО вычислителя
- Поддержка обмена данными через все типы каналов, используемых в ИУС
 - выдача в каналы тестовых данных и прием ответных данных для последующего анализа
 - мониторинг обмена по каналам бортовых интерфейсов и обеспечение доступа тестовых сценариев к результатам мониторинга
 - поддержка формирования сбойного трафика
- Поддержка тестирования временных характеристик функционирования целевой системы
 - формирование и выдача тестовых данных в режиме реального времени
 - измерение задержек поступления ответных данных
- Поддержка многомашинных конфигураций
 - одновременный обмен по десяткам (сотням) каналов



Требования к средствам тестирования ИУС

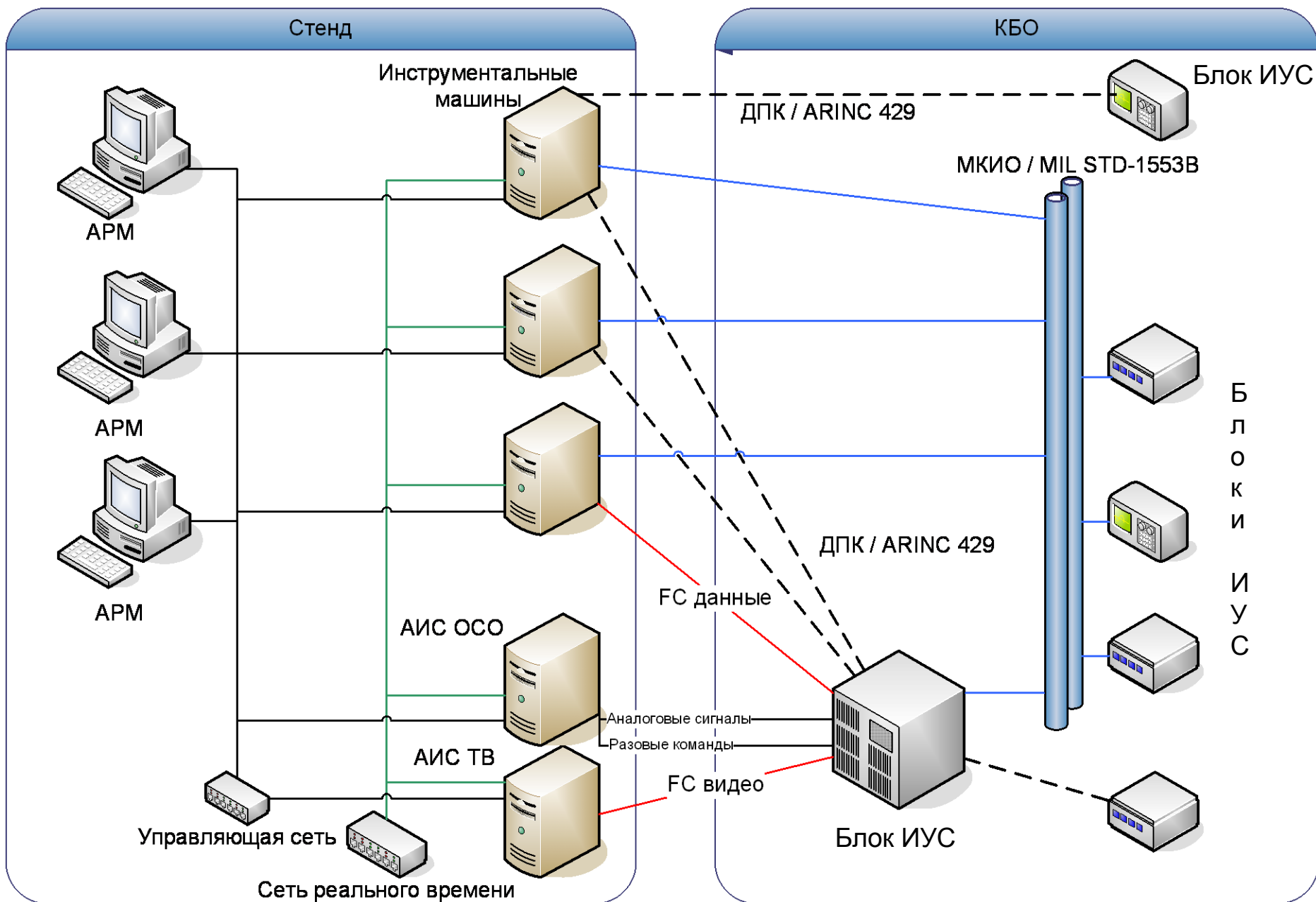
- Поддержка автоматического и интерактивного тестирования
 - интерактивное: для проверки индикаторов и пультов управления
- Поддержка пакетного режима выполнения тестов
- Поддержка оперативного управления тестированием
 - выбор порядка выполнения тестов
 - задание значений тестовых данных
- Поддержка оперативного отображения хода тестирования
 - значения тестовых данных и ответных данных от тестируемой системы
 - протоколы тестирования
- Поддержка прослеживаемости требований и формирования отчётов по результатам тестирования
 - задание соответствия требований тестовым сценариям
 - формирование матрицы прослеживаемости требований
 - формирование отчёта о прохождении тестов и выполненности требований по результатам тестирования
- Интеграция со средствами поддержки разработки ПО ИУС
 - средства управления версиями (хранение ПО и тестов в едином репозитории)
 - средства управления требованиями
 - база данных бортовых интерфейсов (форматы информационных сообщений)
- Единый подход к тестированию для различных фаз жизненного цикла ИУС

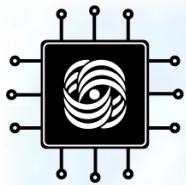


Комплекс средств тестирования ИУС

- Разработан в Лаборатории вычислительных комплексов ВМК МГУ
- Предназначен для тестирования устройств ИУС через каналы бортовых интерфейсов (КБИ)
- Функционирует на ПК под управлением ОС Linux, в состав которых входят адаптеры КБИ
- Поддерживает распределенное выполнение тестовых сценариев
- Удовлетворяет перечисленным выше требованиям
- Положен в основу семейства стендов тестирования, отработки и интеграции ИУС

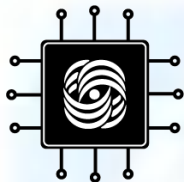
Архитектура стенда тестирования ИУС





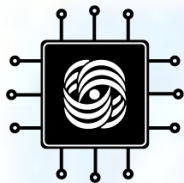
Задачи, требующие работы с натурными устройствами ИУС

- интеграция аппаратуры и ПО, отладка ПО ИУС на целевой платформе
- интеграция компонентов ПО ИУС, в т.ч. компонентов, поступающих от предприятий-созработчиков
- интеграция подсистем ИУС, а также ИУС в целом как многокомпонентной аппаратно-программной системы
- функциональное и квалификационное тестирование ПО ИУС
- приемосдаточные испытания серийно выпускаемых комплектов ИУС
- диагностика блоков ИУС, по которым поступили рекламации
- диагностика блоков ИУС в составе объекта

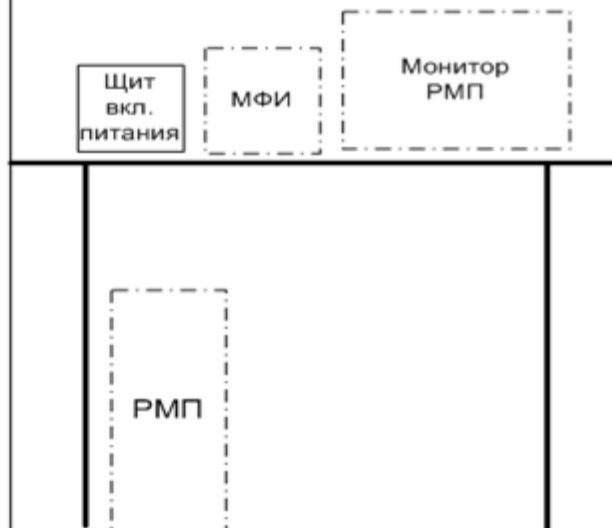


Стенд отработки ПО БЦВМ

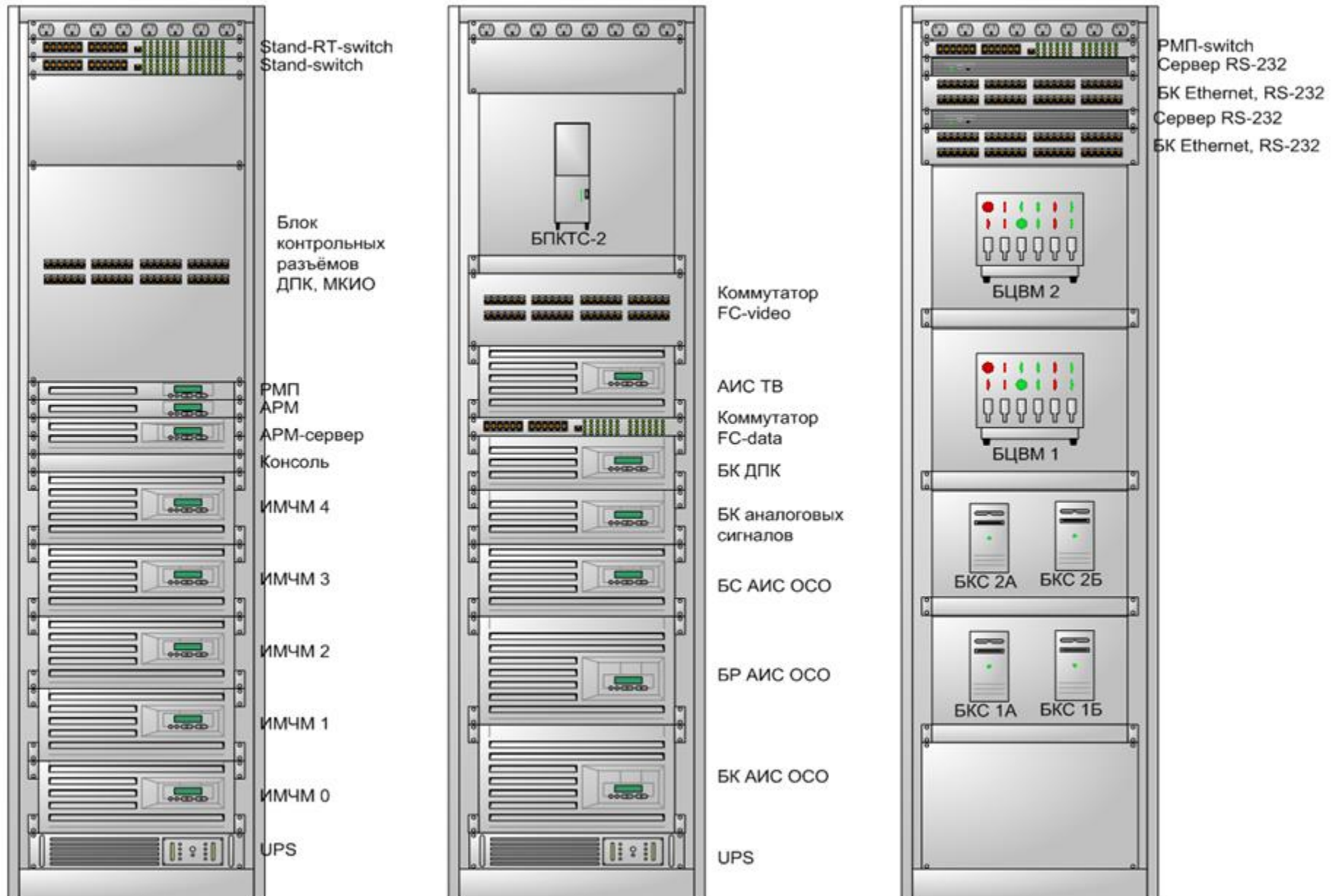


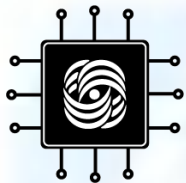


Стенд отработки БЦВМ+МФИ



Стенд тестирования серийных комплектов ИУС





Мобильная рабочая станция мониторинга и анализа бортовых интерфейсов



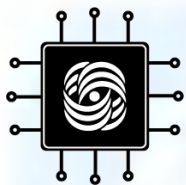
Промышленный ноутбук



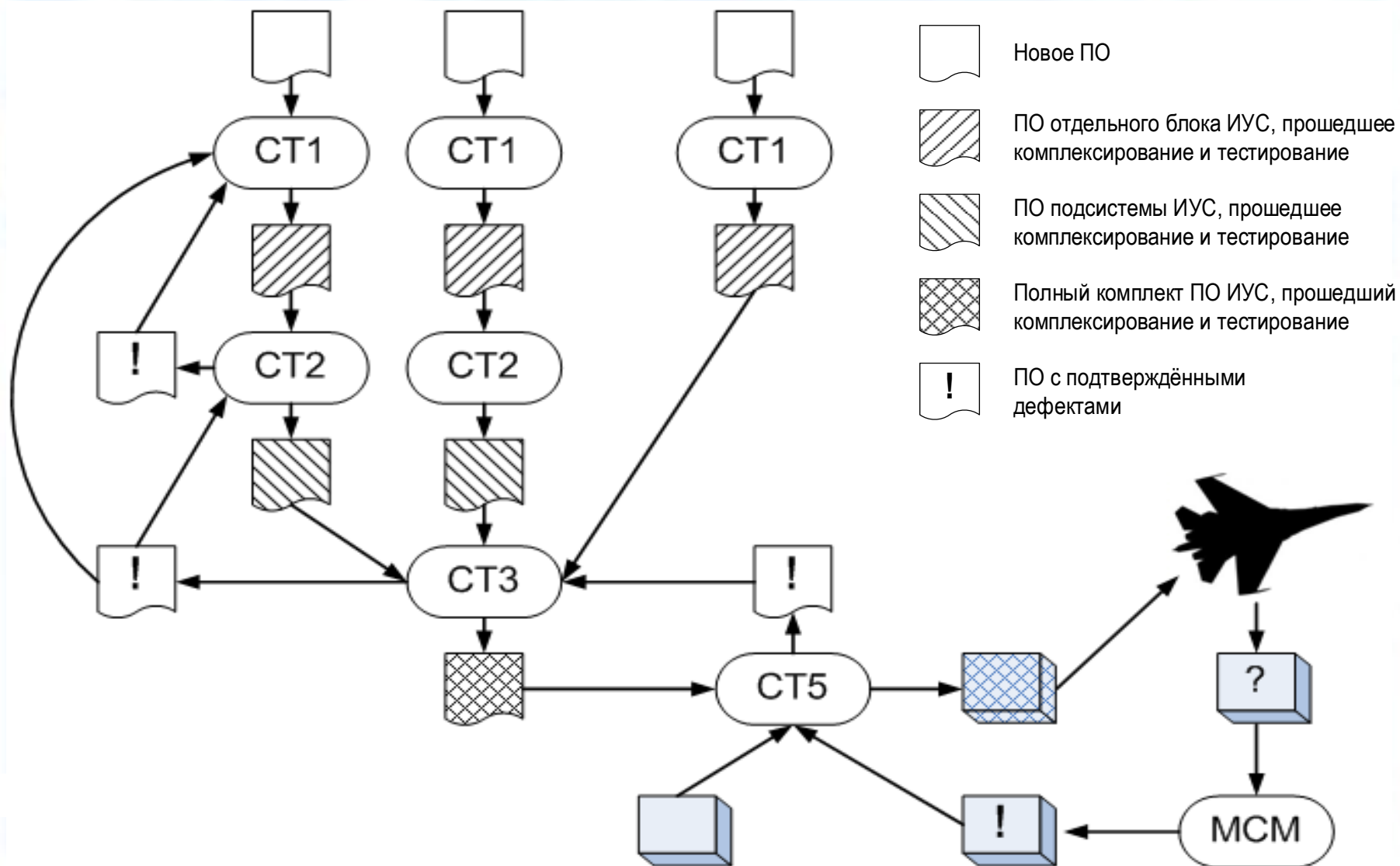
Адаптеры бортовых
интерфейсов
(MIL STD-1553B,
Fibre Channel и др.)



Мобильный коммутатор
Fibre Channel



Технологический цикл применения семейства стенов отработки ИУС





Технологический цикл применения семейства стандов отработки ИУС

Обозначения

Программное обеспечение (ПО)



Новое ПО



ПО отдельного блока ИУС, прошедшее комплексирование и тестирование



ПО подсистемы ИУС, прошедшее комплексирование и тестирование



Полный комплект ПО ИУС, прошедший комплексирование и тестирование



ПО с подтверждёнными дефектами

Стенды

СТ1

Стенд тестирования и отладки ПО отдельного блока ИУС

СТ2

Стенд тестирования и отладки ПО нескольких сопряжённых блоков ИУС

СТ3

Стенд комплексирования и приёмосдаточных испытаний ИУС

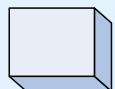
СТ5

Стенд приёмосдаточных испытаний серийных комплектов ИУС

МСМ

Мобильная рабочая станция мониторинга и тестирования блоков ИУС

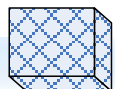
Аппаратура



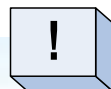
Новые серийные образцы блоков ИУС



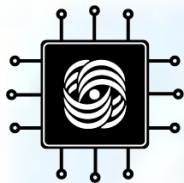
Комплект ИУС, предположительно имеющий дефекты



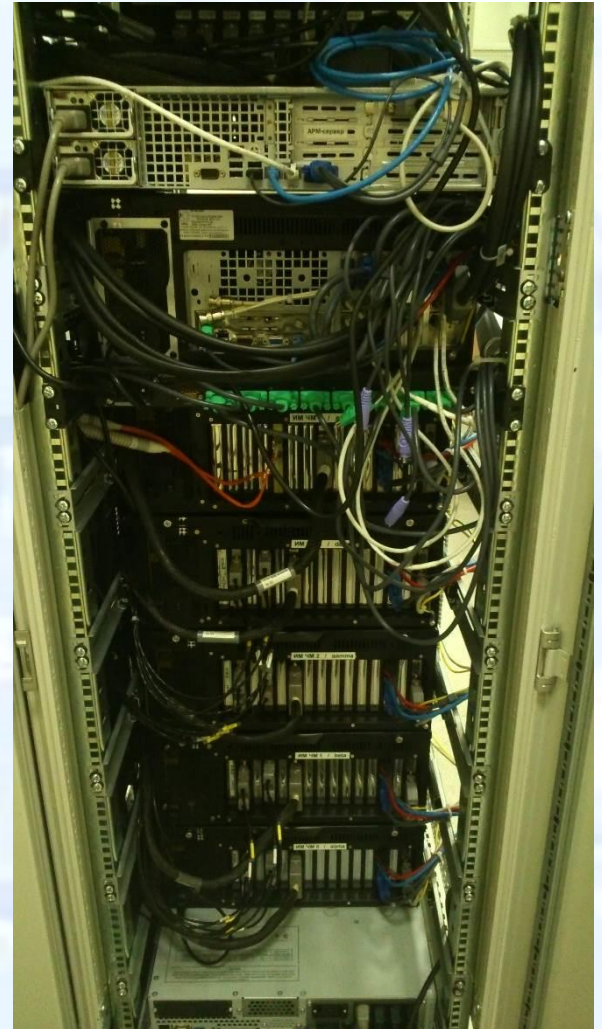
Комплект аппаратуры и ПО ИУС, прошедший комплексирование и приёмосдаточные испытания

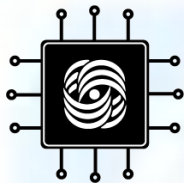


Блок ИУС с выявленными дефектами



Стенд тестирования серийных комплектов ИУС Су-35



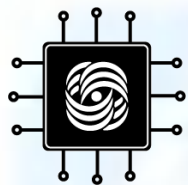


Стенд интеграции ИУС МС-21





- ACIB – Avionics Core Integration Bench



Стенд «Электронная птица»

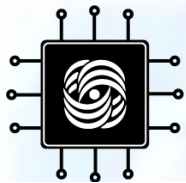


- SIB – System Integration Bench
- Компьютерное моделирование бортового электромеханического оборудования

Стенд «Железная птица»



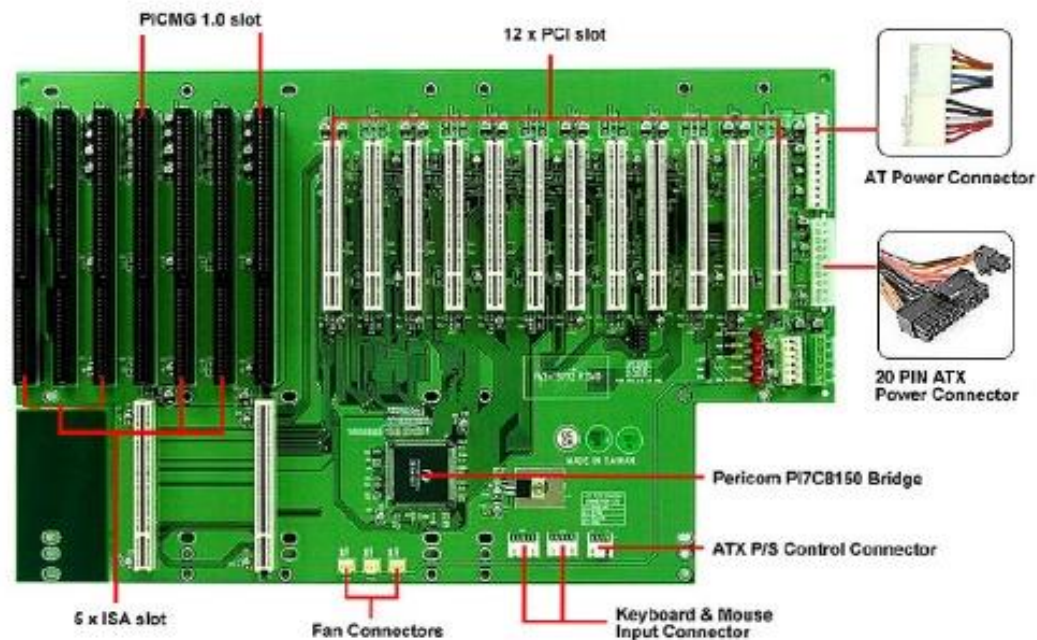
- Установлена большая часть электромеханического оборудования
- «Самолёт на полках», дальше – прототип самолёта в сборе

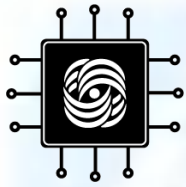


Аппаратная база стенда

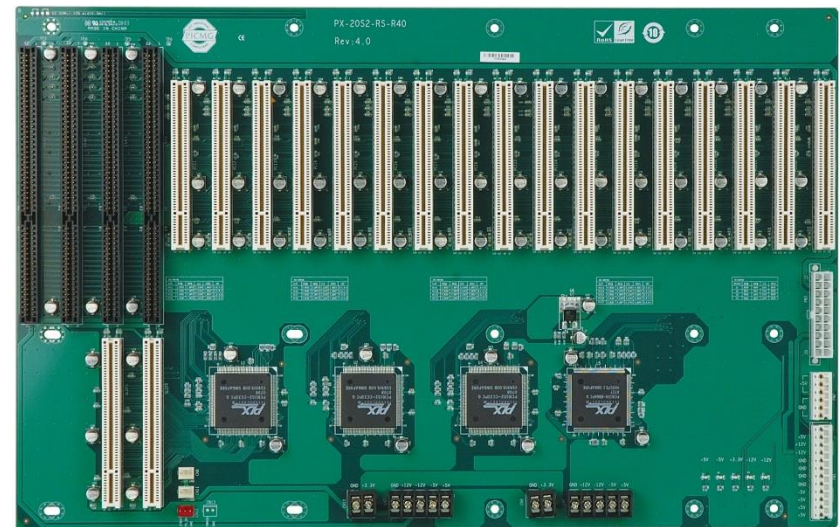
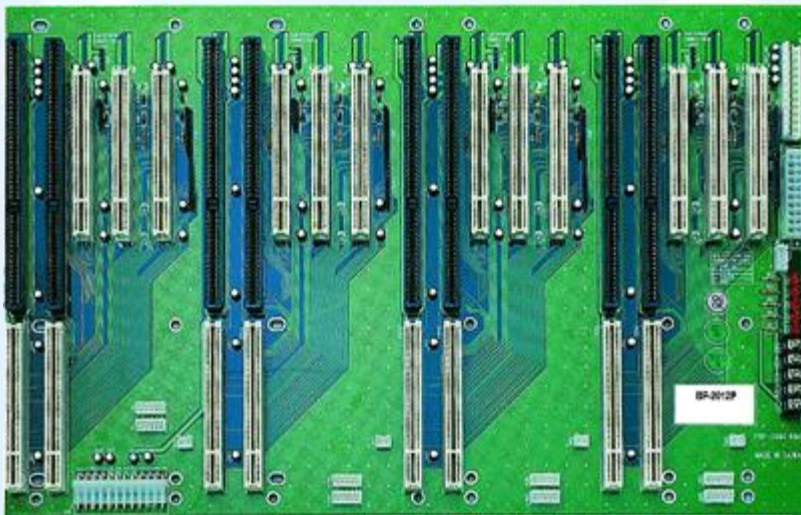
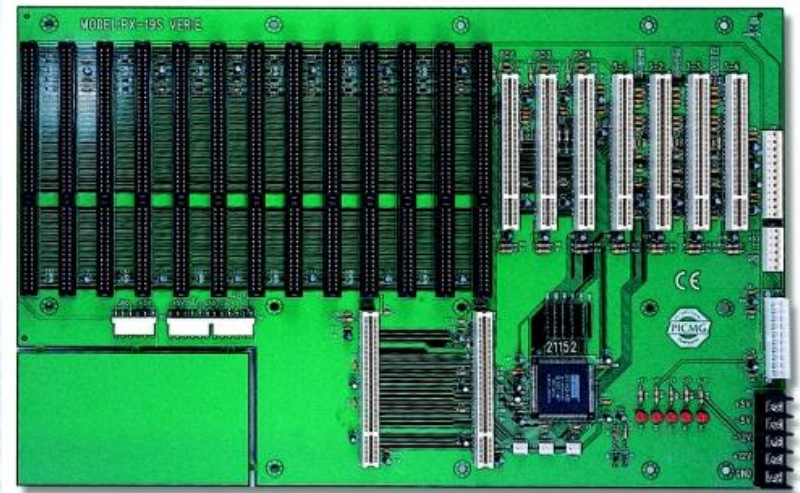


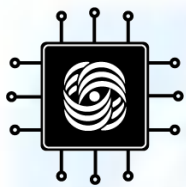
- Промышленные компьютеры (PICMG)
- Платы-расширители шин
 - ISA
 - PCI
 - PCI Express
- Процессорные платы
- Адаптеры бортовых интерфейсов





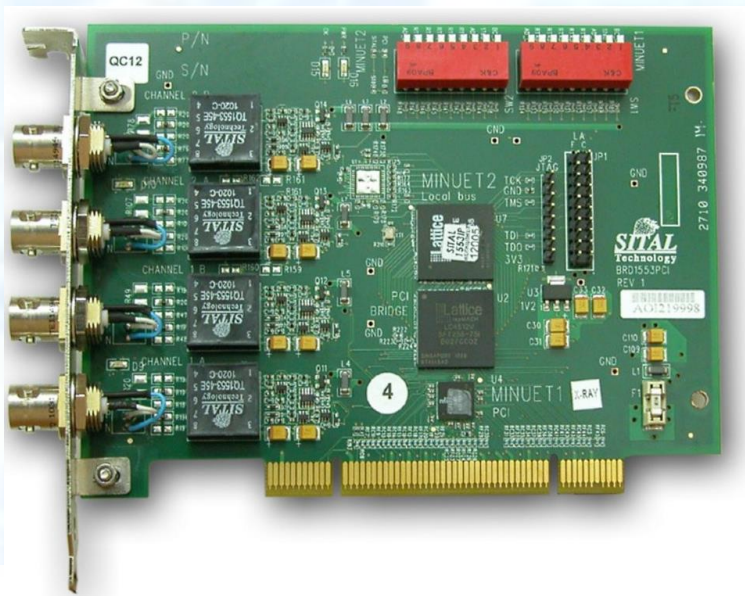
Аппаратная база стенда





Аппаратная база стенда

2x MIL STD-1553B / PCI

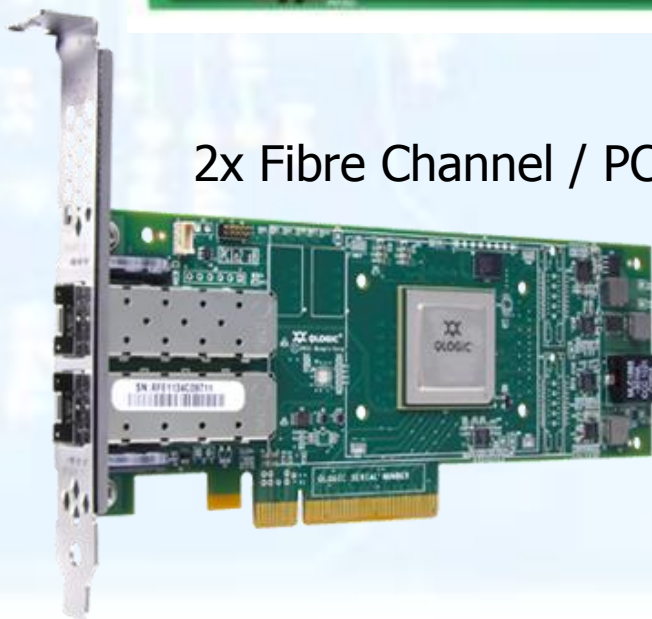


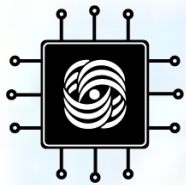
1x MIL STD-1553B / PCI Express



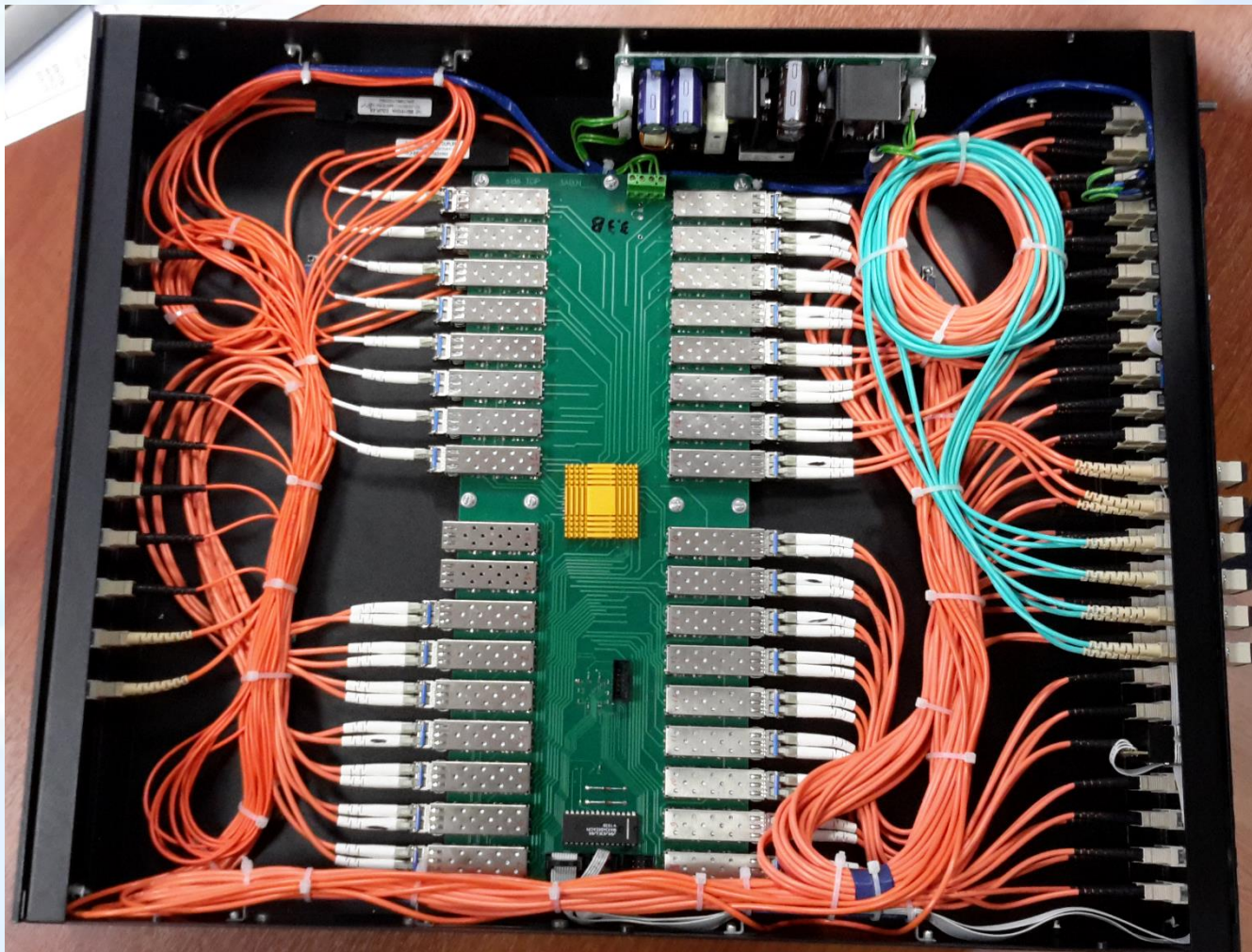
16T16R
ARINC 429 / PCI

2x Fibre Channel / PCI Express

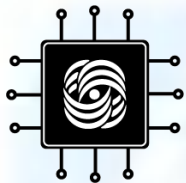




Аппаратная база стенда

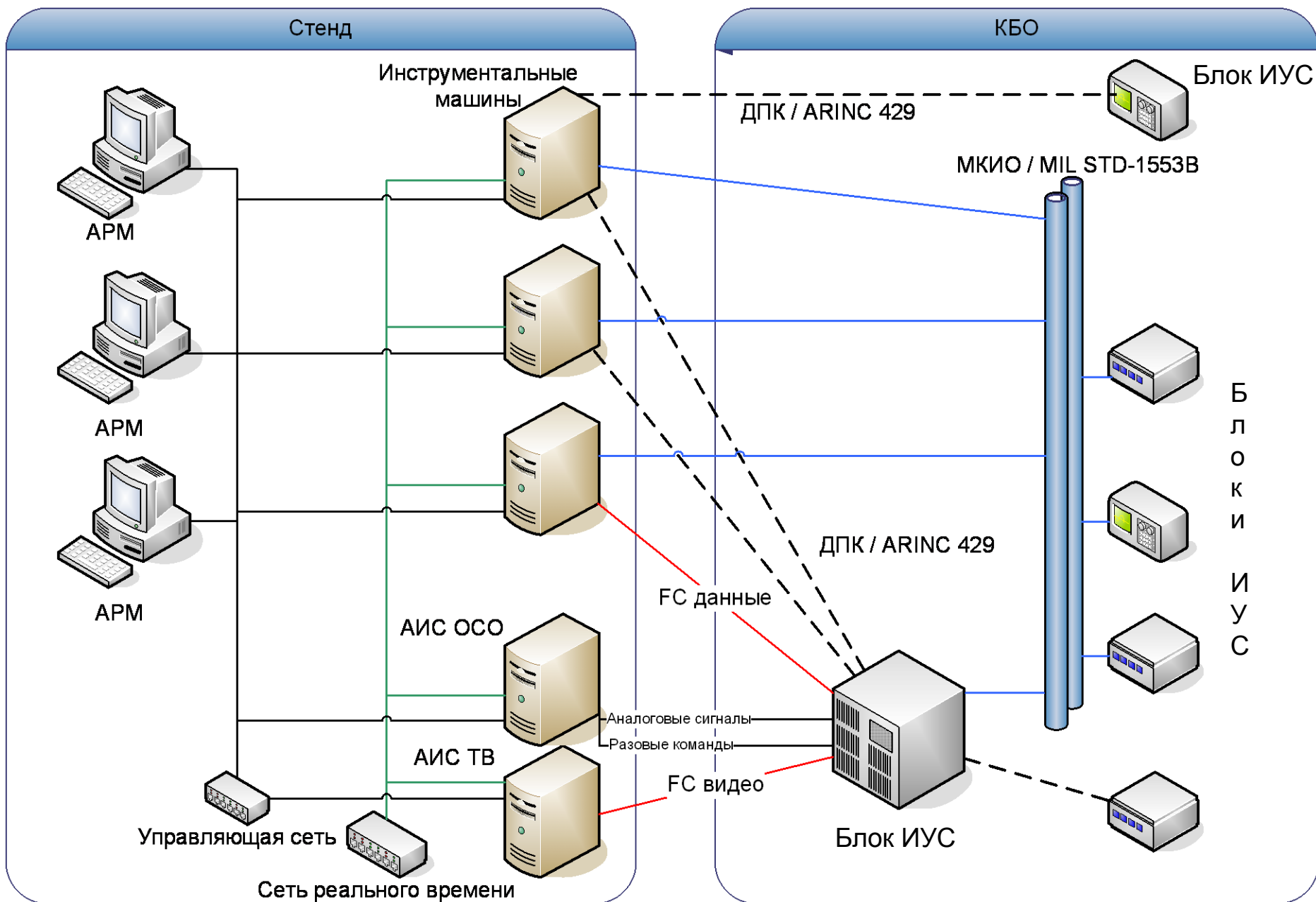


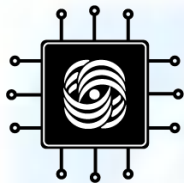
- Программируемый коммутатор Fibre Channel
- Оптический сигнал преобразуется в электрический
- Коммутация электрического сигнала при помощи ПЛИС



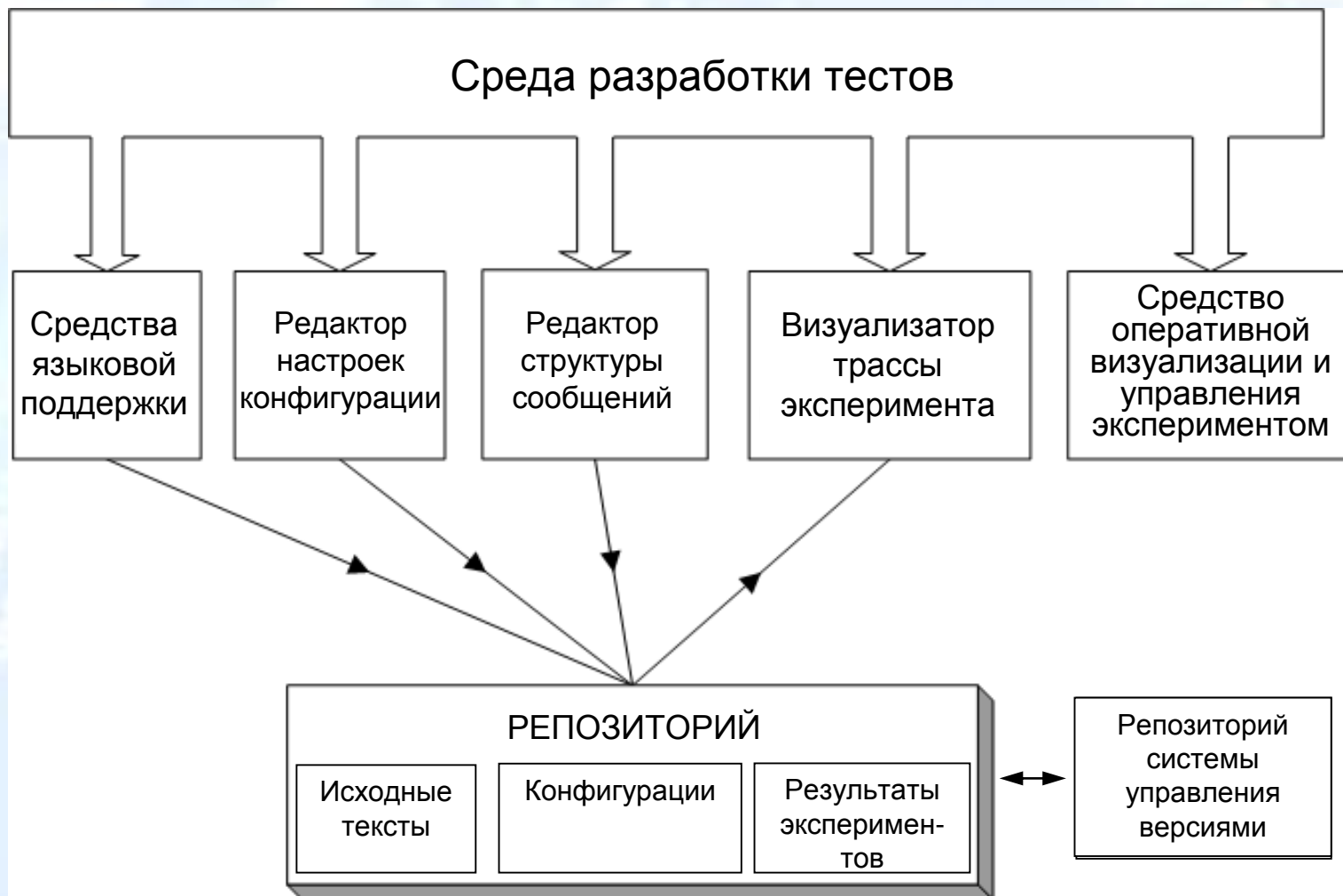
Программные средства функционального тестирования в составе стенда

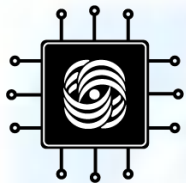
Архитектура стенда тестирования ИУС



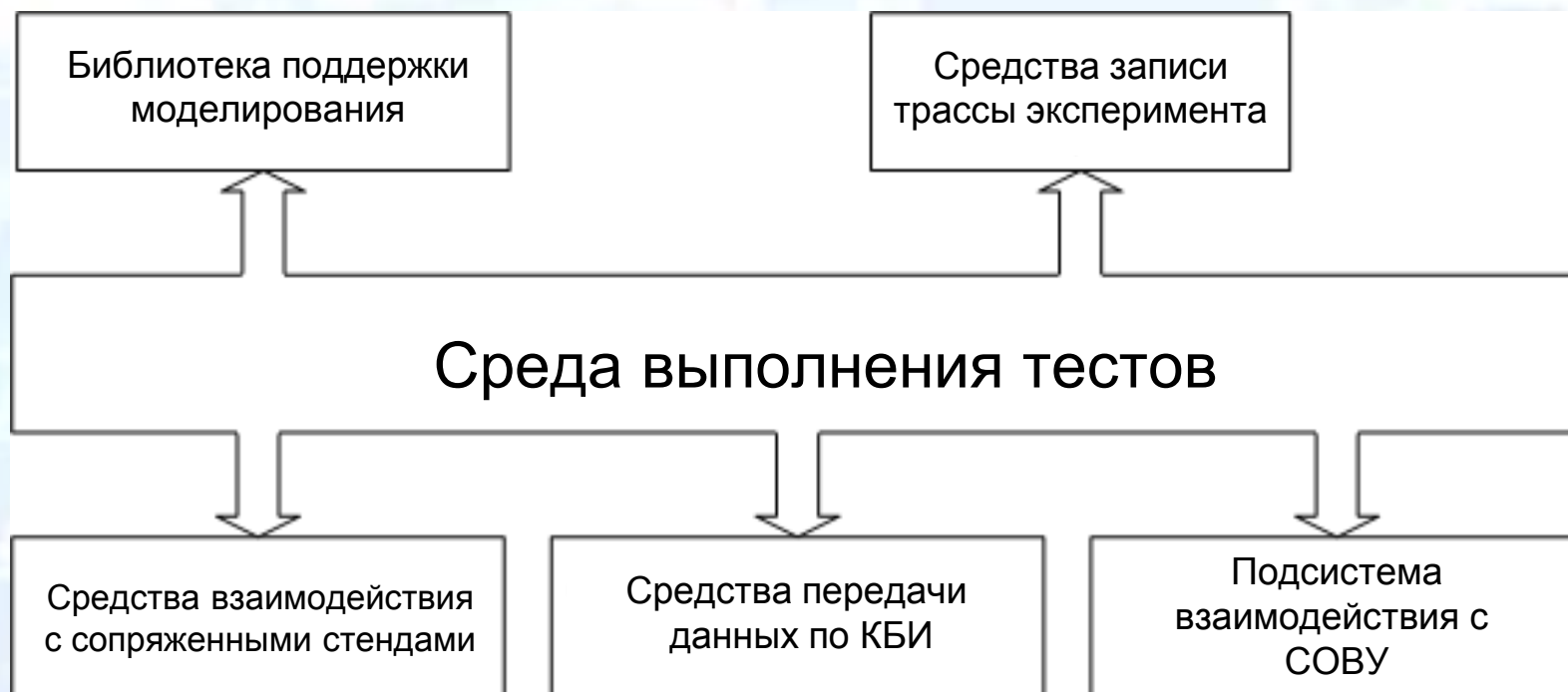


Средства АРМ





Средства ИМ



Язык описания тестов (ЯОТ)

- Поддерживает тестирование ИУС через каналы бортовых интерфейсов (MIL-STD-1553B, ARINC 429, FC-AE-ASM, CAN, AFDX и т.п.) с использованием следующих основных элементов:
 - Сообщения и слова КБИ
 - Битовые поля сообщений/слов
- Предоставляет операторы для:
 - Определения структуры слов/сообщений (состав битовых полей, их имена и расположение)
 - Задания и обработки значений битовых полей
 - Автоматической проверки тестируемых условий
 - Взаимодействия с пользователем
 - Задания структуры тестового сценария

Основные понятия ЯОТ

- Тестовые компоненты (ТСК)
- Интерфейсы
 - Обеспечивают подключение к каналам БИ
- Сообщения бортовых интерфейсов
- Битовые поля сообщений
- Тестовые случаи (test case)
- Тестовые шаги (test step)

Тестовые компоненты

Тестовый компонент (ТСК) – это средство компоновки объектов ФТ в составе стенда. ТСК определяет состав и последовательность выполнения тестов.

С каждым ТСК связаны:

- *имя (определяет тип ТСК)*
- *описание сообщений КБИ с указанием их структуры*
- *описание тестов*

Структура ТСК

Заголовок ТСК определяет состав интерфейсов, состав и структуру сообщений КБИ, состав тестов и их привязку к требованиям

Тело ТСК определяет состав и логику выполнения тестовых случаев и тестовых шагов

Интерфейсы ТСК

Интерфейс – это объект средств ФТ, предоставляющий программный интерфейс для взаимодействия ТСК с натурными блоками ИУС, а также друг с другом

- Интерфейсы обеспечивают доступ ТСК к адаптерам КБИ
- Через интерфейсы обеспечивается передача и прием сообщений

Сообщение MIL STD-1553B

Сообщение MIL STD-1553B – это объект средств ФТ, соответствующий сообщению бортового интерфейса MIL STD-1553B.

С каждым сообщением связаны:

- *имя*
- *интерфейс MILS*
- *адрес*
- *подадрес*

С отдельными словами в составе сообщения можно работать как с элементами массива.

(сообщения для других типов интерфейсов устроены аналогично)

Сообщения MIL STD-1553B

RTS_OUTPUT_MESSAGE (<имя>,
 <интерфейс MILS>,
 <адрес>,
 <подадрес>,
 <кол-во слов в сообщении>,
 <период в мс>)

RTS_INPUT_MESSAGE (<имя>,
 <интерфейс MILS>,
 <адрес>,
 <подадрес>,
 <кол-во слов в сообщении>,
 <период в мс>)

RTS_OUTPUT_ / RTS_INPUT_TERM_MESSAGE (<имя>,
 <интерфейс MILS>,
 <подадрес>,
 <кол-во слов в сообщении>)

Битовые поля и Биты

Битовое поле – это объект средств ФТ, соответствующий группе последовательных битов в составе слова сообщения.

Бит – это битовое поле, соответствующее 1 биту.

С каждым битовым полем связаны:

- *имя*
- *позиция в сообщении (номер первого слова, номер начального бита, кол-во битов)*
- *признак использования знакового или беззнакового поля*
- *коэффициент масштабирования (ЦМР)*

С битовым полем можно работать как с числовой переменной.

Битовые поля и биты

RTS_BITFIELD (<имя поля>,
 <имя сообщения>[<номер слова>],
 <начальный бит>,
 <кол-во битов>
)

RTS_BOOLEAN (<имя поля>,
 <имя сообщения>[<номер слова>],
 <позиция бита>)

Задание иерархического идентификатора теста

Идентификатор теста задается в формате: _____

<имя 1>.<имя 2>....<имя n>

Например: **«Су35.СОЛС.Наземный_контроль.17»**

RTS_TESTNAME (“<имя теста>”)

RTS_TESTNUM (<номер теста>)

RTS_TEST_SHOW ()

RTS_TEST_PASSED () _____

RTS_TEST_FAILED ()

Задание соответствия требований идентификаторам тестов

```
RTS_DOCUMENT("<requirement specification document name>",  
    RTS_REQUIREMENT(<requirement identifier>, "<requirement  
    name>"),  
    RTS_REQUIREMENT(<requirement identifier>, "<requirement  
    name>"),  
    ...  
)  
  
RTS_TESTSPEC ("<test name (or name template)>",  
    <requirement identifier>, <requirement identifier>, ...)
```

Задание соответствия требований идентификаторам тестов

Пример:

Заголовок

```
RTS_DOCUMENT( "Board computer system initialization",  
    RTS_REQUIREMENT(BCS.INIT.START, "Starting up MIL STD-1553B  
    subscribers"),  
    RTS_REQUIREMENT(BCS.INIT.WORK, "Starting BCS operation main  
    loop"),  
)  
  
RTS_TESTSPEC ("3.2.{1-4}", BCS.INIT.START)  
  
RTS_TESTSPEC ("3.2.{1-4}", BCS.INIT.WORK, BCS.SKV.WORK)
```

Тело

```
RTS_TESTNAME(3.2.2)
```

```
// логика тестового шага
```

Временная задержка, отправка данных

// Задержать выполнение теста на заданное время

RTS_DELAY (<duration (ms)>)

// Выдать сообщение в канал (для активных абонентов)

RTS_EXECUTE (<Message/word name>)

Взаимодействие с пользователем

// Отобразить значение объекта
// и занести это значение в протокол
RTS_DISPLAY (<object name>)

// Отобразить и записать в протокол текстовое сообщение
RTS_COMMENT (<message text>)

// Приостановить выполнение теста
// и ждать команды пользователя на продолжение
RTS_PAUSE ()

// Запросить у пользователя заключение (тест пройден / не пройден)
// и завершить тест в соответствии с полученным заключением
RTS_CONFIRM ()

// Запросить у пользователя заключение и занести его в протокол
RTS_REQUEST ()

// Запросить у пользователя текстовый комментарий и занести его в протокол
RTS_EXPLAIN ()

Пользователь
выполняет все
действия через
интерфейс СОВУ

Тестовый шаг

Тестовый шаг – фрагмент теста, проверяющий элементарную функцию тестируемого устройства.

Тестовый шаг может проверять правильность приема/передачи отдельного сообщения или слова, или отдельного битового поля в составе сообщения.

Типичная структура тестового шага:

- *Установка тестовых условий (входных значений)*
- *Ожидание результата*
- *Проверка ожидаемого результата (+ нумерация)*
 - Операторы **if** и **RTS_WHEN**

Тело:

```
RTS_TESTNAME ("<name>"),  
RTS_TESTNUM (<multi-level number>)           // автоинкремент
```

Оператор RTS_WHEN

```
RTS_WHEN ( "W46A185_13 == 0" , 15) { // ожидание 15 мс
    RTS_THEN():
        RTS_COMMENT (
            "ERROR Circuit- Command Port (W46A185_13 = 0)"
        );
        RTS_TEST_FAILED();
        break;
    RTS_OTHERWISE():
        RTS_TEST_PASSED();
        break;
}
```


Пример тестового шага

```
// StateValue – битовое поле в RTS_OUTPUT_TERM_MESSAGE()
// ControlWord – битовое поле в RTS_INPUT_TERM_MESSAGE ()
// -----
RTS_TESTNUM( 2.1 );

// Установка значения отправляемых данных
StateValue = 0.0;
// Ожидание выдачи сообщения
RTS_DELAY( 0.5 );
// Проверка ответа от тестируемого устройства
RTS_WHEN ( "ControlWord == 5" ) {
    //.....
}
```

Тестовый случай

Тестовый случай – последовательность связанных по смыслу тестовых шагов, соответствующих отдельным разделам или отдельным требованиям *системной спецификации* на тестируемое устройство.

“Минималистический” тестовый случай может состоять из одного тестового шага.

При выполнении тестового случая увеличивается нумерация тестов на соответствующем уровне.

Протокол тестирования

- Информация о смене (назначении) номеров тестов
- Информация о смене выполняемых тестов
- Результаты тестирования: «Тест прошел» / «Тест не прошел» с указанием номера теста
- Результаты всех команд взаимодействия с пользователем
- Комментарии с описаниями ожидаемого поведения теста и другая дополнительная информация, выдаваемая командой RTS_COMMENT
- Информация о текущих значениях объектов ФТ (в т.ч. битовых полей и битов), выдаваемая командой RTS_DISPLAY

Работа с протоколами тестирования

Заголовок

```
RTS_LOG_FILE(LOG1, "log1.log")  
// Файл «log1.log» будет помещен в подкаталог  
// «ПротоколыТестирования» каталога  
// результатов эксперимента
```

Тело

```
RTS_LOG(LOG1)  
RTS_CLEAR_LOG(LOG1)
```

Протоколы тестирования служат основой для формирования отчетов по тестированию.

Пример протокола тестирования

- * Протокол функционального тестирования LOG1
- * Собран компонентом tsk (Stand.tsk)
- * Тип компонента: TCK_Stand::TCK::TCK
- * Имя эксперимента: T1
- * Дата эксперимента: Втр Май 30 09:46:25 MSD 2006
- * Продолжительность эксперимента: 30с 000мс
- * Эксперимент запущен пользователем: mike

>>>> [0мс] от начала эксперимента

| Начало функционального тестирования БЦВМ из проекта DemoLang

Тест номер 3.2.1

>>>> [9с 000мс] от начала эксперимента

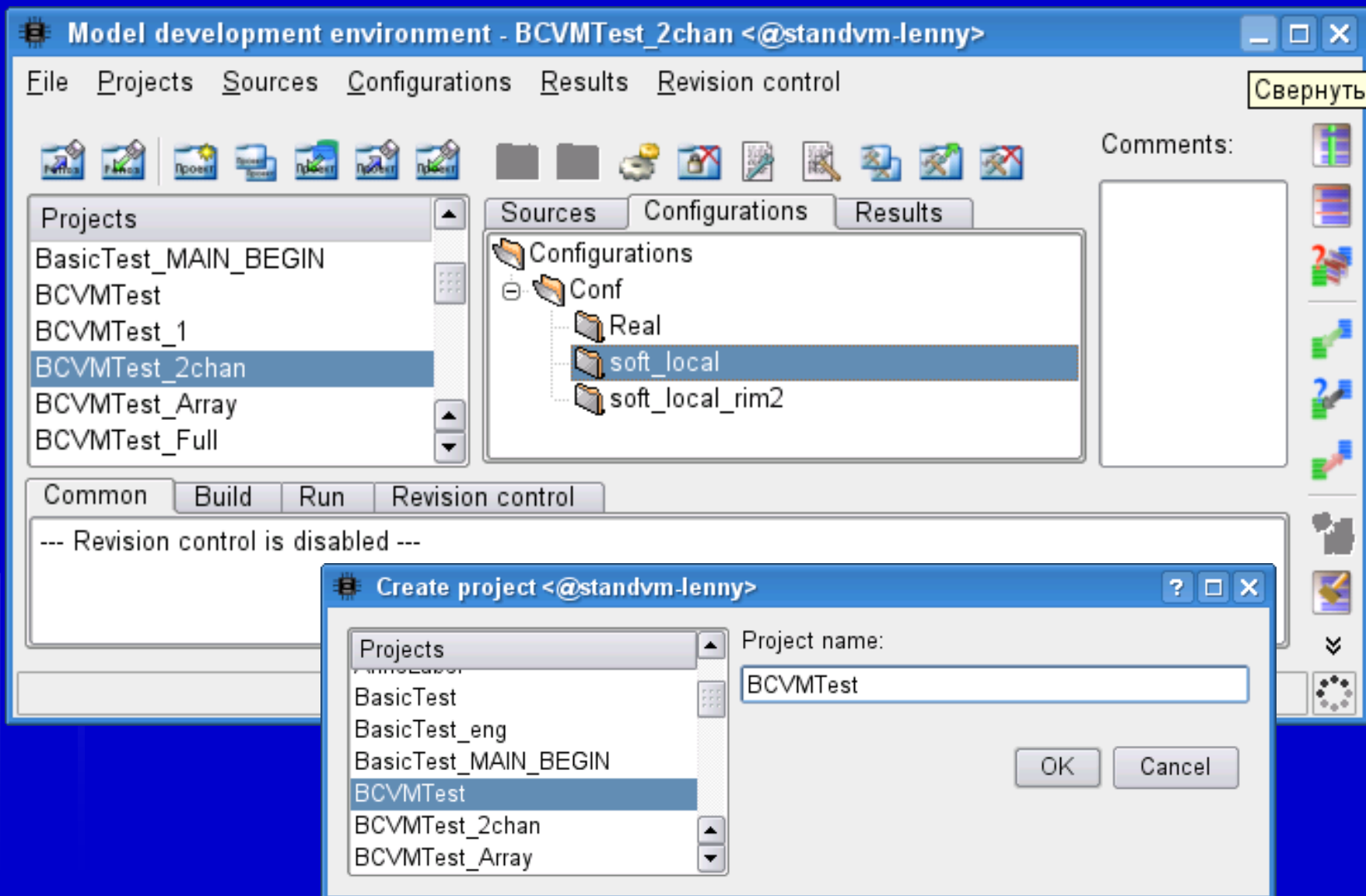
Тест номер 3.2.1 прошёл

Тест номер 3.2.2

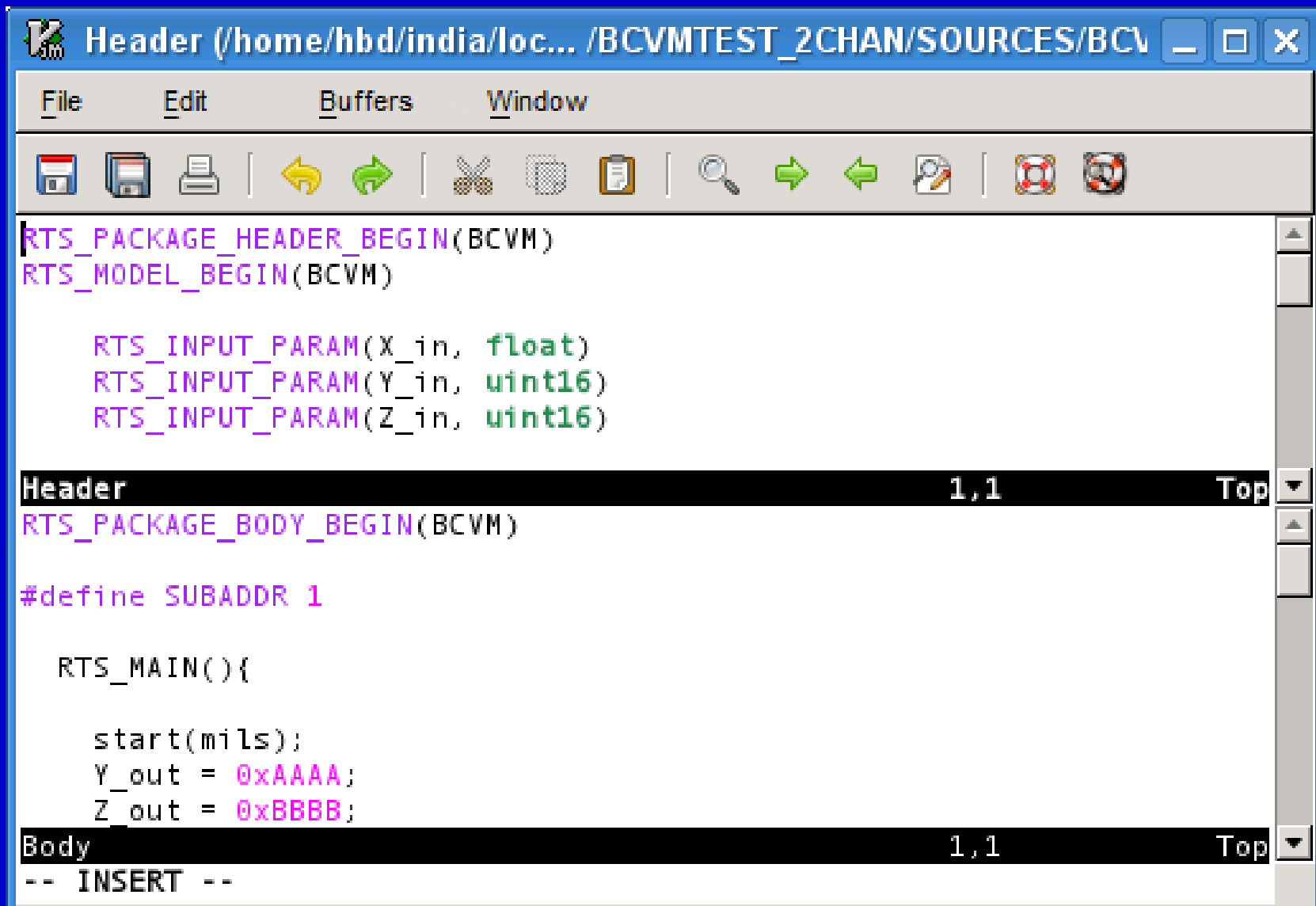
...

Подготовка и выполнение тестирования

Создание проекта



Редактирование кода ТСК



```
Header (/home/hbd/india/loc... /BCVMTEST_2CHAN/SOURCES/BCV)
File Edit Buffers Window

RTS_PACKAGE_HEADER_BEGIN(BCVM)
RTS_MODEL_BEGIN(BCVM)

    RTS_INPUT_PARAM(X_in, float)
    RTS_INPUT_PARAM(Y_in, uint16)
    RTS_INPUT_PARAM(Z_in, uint16)

Header 1,1 Top
RTS_PACKAGE_BODY_BEGIN(BCVM)

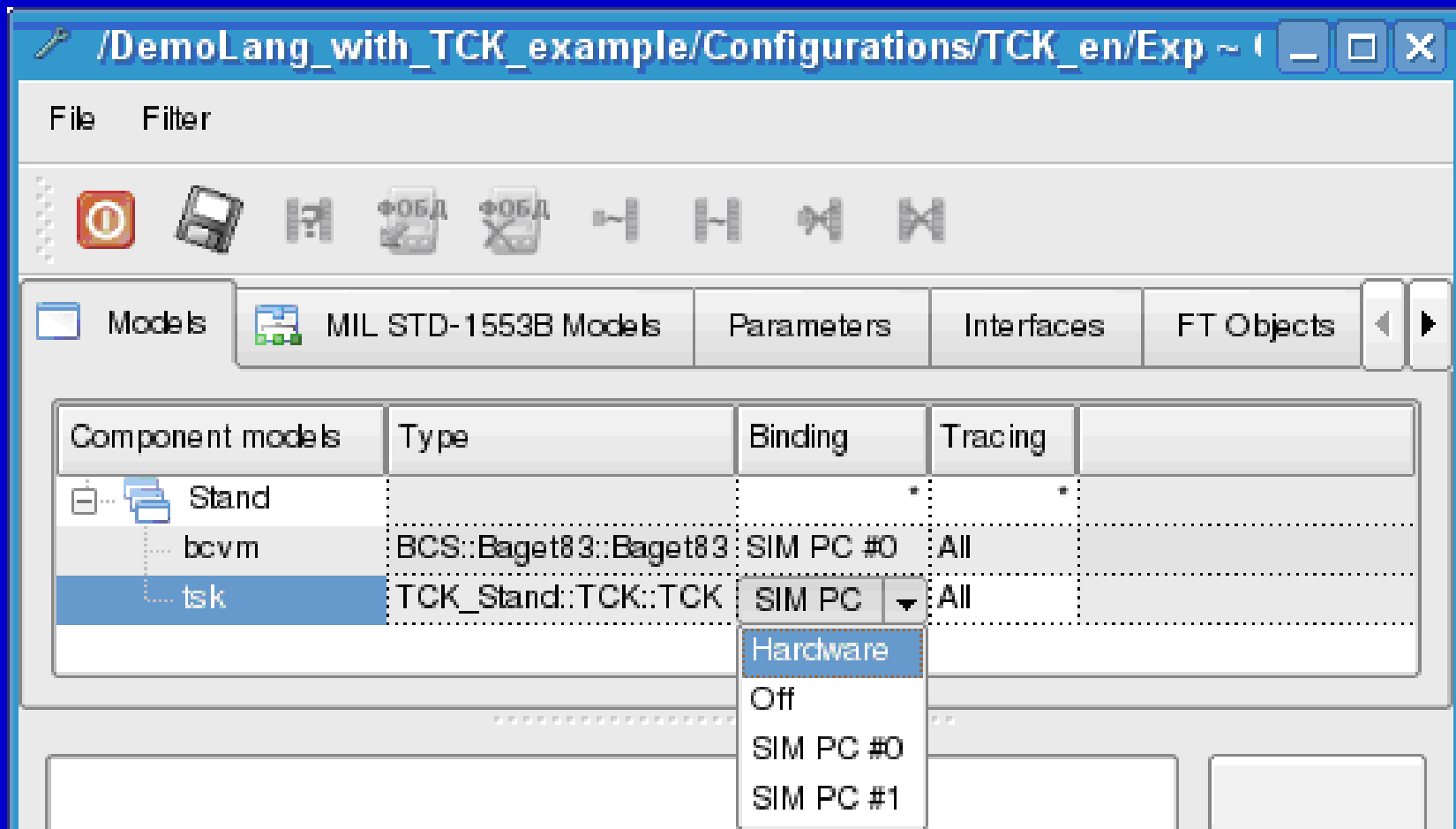
#define SUBADDR 1

    RTS_MAIN(){

        start(mils);
        Y_out = 0xAAAA;
        Z_out = 0xB BBB;

Body 1,1 Top
-- INSERT --
```


Привязка ТСК к инструментальным машинам



Привязка интерфейсов к адаптерам КБИ

/DemoLang/Configurations/Conf_en/Local ~ Configuration settings editor. Version 1.0

File Filter

Models MIL STD-1553B Models ARINC 429 Models Parameters Interfaces DBDF

Inter-model MIL STD-1553B ARINC 429 interfaces

Models/Interfaces	Tracing	RT address	Controller	Binding	Adapter	Connects To	Alias	De
Stand	*			*				
bcs	*			*				
bcvm	*			*				
mils	All	0	True	With model	-- not set --	Stand.bcs.mils.io		
rv	*			*				
mils	All	18	False	With model	-- not set --	Stand.bcs.mils.io		
skv	*			*				
mils	All	17	False	With model	-- not set --	Stand.bcs.mils.io		

Stand::bcs::skv::milsAll

Параметры эксперимента

Start experiment <@standvm-lenny>

Configuration: Conf/Local

Results
Exp1

Define experiment parameters

Experiment name: Exp1

Experiment duration, [hh:]mm:ss[.ms]: 0:01:00.000

Soft real time parameters

Model time scale: 1

Execution mode:

- ☒ Soft real time
- ☐ Real time
- ☐ Debug
- ☐ Non real time

OK Cancel

Оперативное управление экспериментом

Online visualization and control tool <@standvm-lenny>

Window Commands Help

State tree

- Stand
 - env
 - all
 - Fictive
 - On
 - bcs
 - bcvm
 - On
 - Waiting
 - Waiting
 - rv
 - On
 - On
 - RT is listening to the bus
 - skv
 - Tect
 - On
 - RT is listening to the bus
 - arinc
 - Free
 - Waiting

Value tree

- Stand
 - env
 - X: 49925.4
 - Y: -2730.24
 - V_x: 54.6048
 - V_y: 998.508
 - Psi: 3.13018
 - H_suf: 10000
 - w_z: -1.14592
 - a_x: -19.9702
 - a_y: 1.0921
 - bcs
 - bcvm
 - SKVControlWord: 2
 - w_z
 - a_x

Messages

Experiment status: Waiting for user command to start experiment
Experiment status: Experiment is running
[SIM PC 0, 0:00:00.080000]
SIM PC 0: Experiment duration: 60000 milliseconds
Experiment status: Experiment is stopped by user

Experiment status: Experiment is stopped by user

Model time: 0:00:19 Deviation: 0 us

Инструменты оперативного управления тестированием

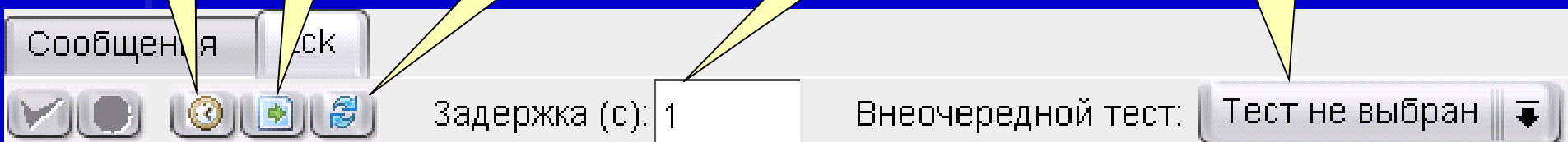
Ожидание
ввода
данных

Ожидание
выбора
теста

Циклическое
выполнение
теста

Задержка
циклического
выполнения

Выбор
внеочередного
теста



Тестовый компонент: Stand.tck Состояние: Тестирование остановлено

>>>> [0мс] от начала эксперимента

Тест "1"

Включён файл протокола тестирования LOG ("Log.log")

>>>> [3с 000мс] от начала эксперимента

Тест "ИУС.А.1"

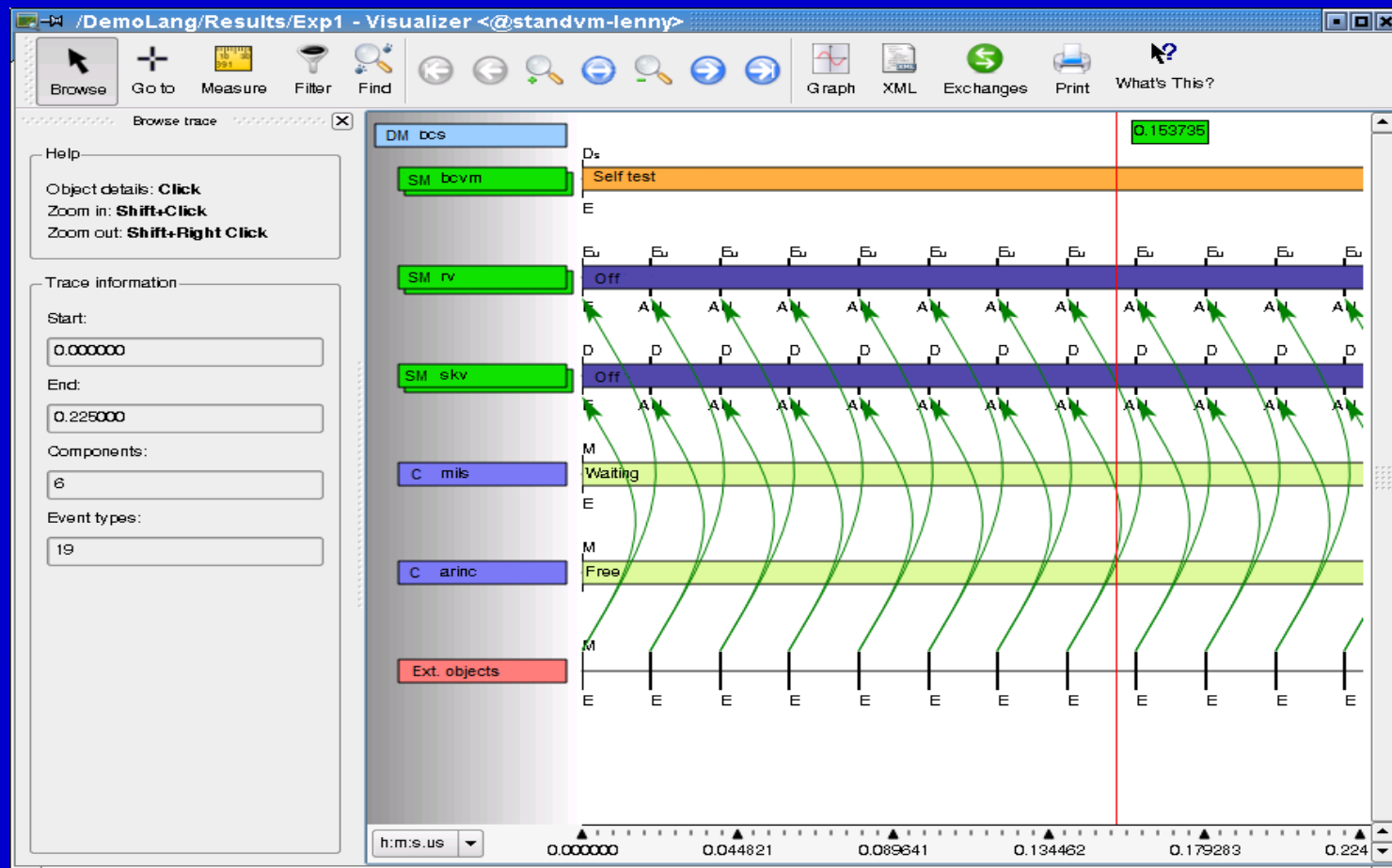
>>>> [5с 000мс] от начала эксперимента

Тест "ИУС.А.1" прошёл

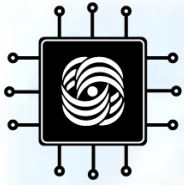
>>>> [6с 000мс] от начала эксперимента

Тест "ИУС.В.2"

Обработка результатов эксперимента

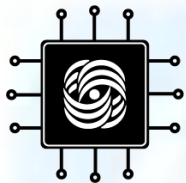


- Формирование отчётов по результатам тестирования
- Анализ временной диаграммы событий эксперимента

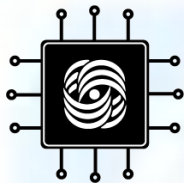


Перспективы

- Автоматизация проверки выполнения расписаний обмена
- Автоматизация тестирования индикаторов
 - интеграция с «блоком нажатия кнопок»
 - перехват изображения
 - сравнение с эталоном
- Автоматизация генерации тестов по описанию требований
- Интеграция тестов с отладочными средствами БЦВМ
- Новые внедрения
 - существующие: Су-35, Т-50/ПМИ, МС-21



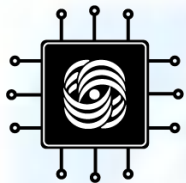
Спасибо за внимание!



ИНФОРМАЦИОННО-УПРАВЛЯЮЩИЕ СИСТЕМЫ РЕАЛЬНОГО ВРЕМЕНИ

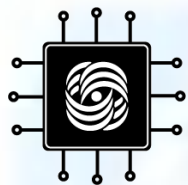
Лекция 11-Б: *Мониторинг обмена данными в ИУС РВ*

Кафедра АСВК,
Лаборатория Вычислительных Комплексов
Балашов В.В.



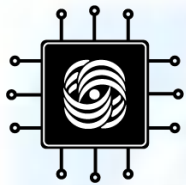
Бортовые устройства





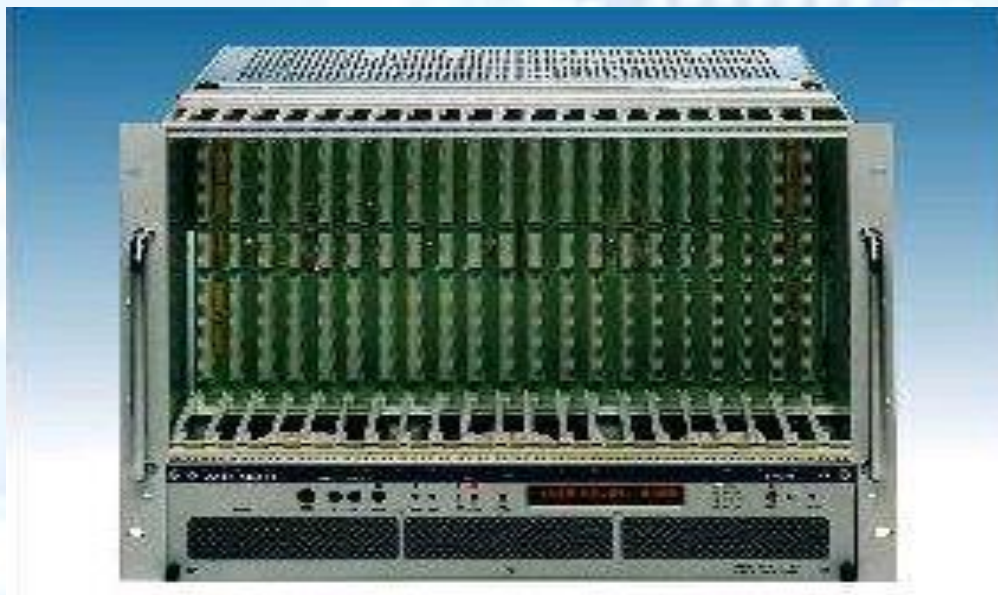
Пример: Интерфейсы БЦВМ

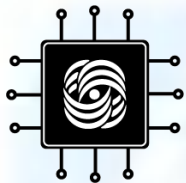




Внутриблочные интерфейсы

- VME32 60 МБ/с
- PCI32 33 МГц 60-80 МБ/с
- PCI Express x4 4x250 МБ/с





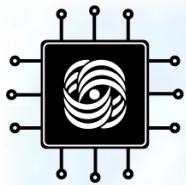
Внешние интерфейсы блоков

- МКИО (MIL-STD-1553B) 80-90 КБ/с
- ДПК (ARINC 429) 7.12 КБ/с
- FC-AE-ASM 100 МБ/с
- AFDX 10-12 МБ/с
- ARINC 818 (видео) 70.1 МБ/с
- CAN (500 КГц) 24.4 КБ/с
- Разовые команды (РК)

Необходимы специализированные инструментальные средства мониторинга и анализа информационных обменов.

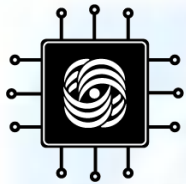


- Технологические интерфейсы
- Внутренние (локальные) интерфейсы
- Бортовые интерфейсы (внешние)



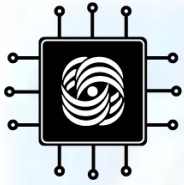
Мониторинг информационного обмена

- Уровни информационного обмена:
 - Между блоками (канал)
 - Между модулями в составе блока (внутренняя шина)
 - Между функциональными задачами в рамках модуля (разделяемая память, очереди сообщений)



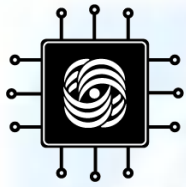
Мониторинг каналов/шин

- По уровням протокола:
 - физический: проверка наличия и характеристик сигнала
 - канальный:
 - проверка соответствия информационных слов и кадров стандарту канала
 - проверка ограничений реального времени на передачу слов/кадров
 - анализ активности абонентов канала
 - логический:
 - проверка соответствия параметров, передаваемых в полезной нагрузке, протоколу информационного взаимодействия

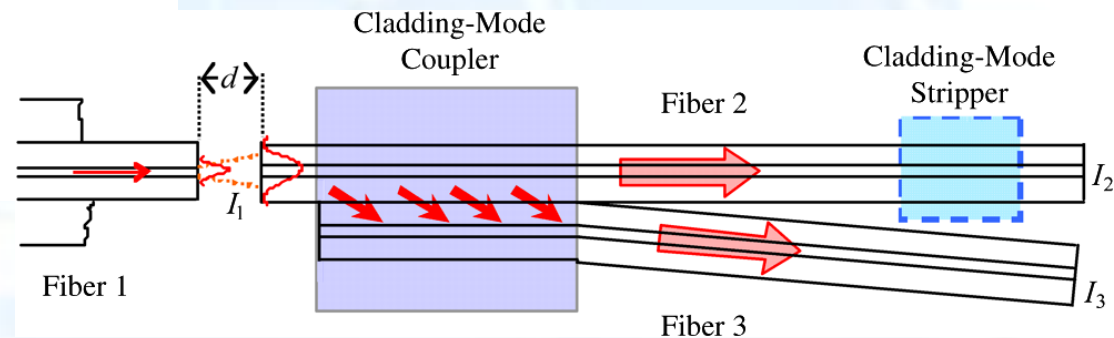
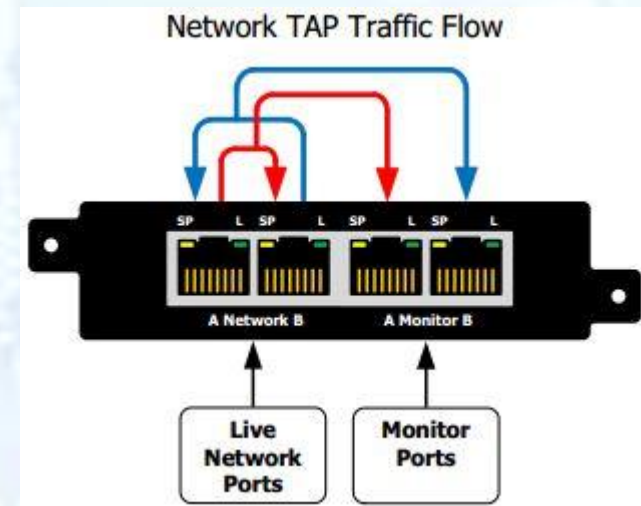
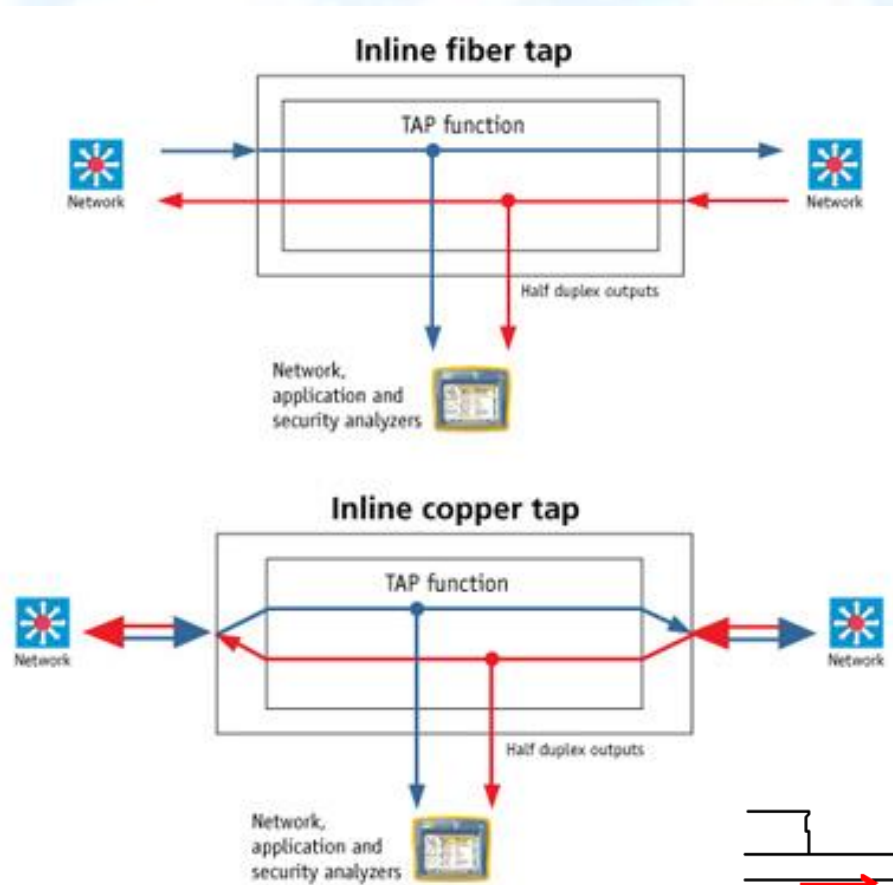


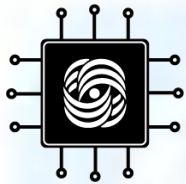
Варианты анализа

- Оперативный: выявление и анализ проблем «на лету»
 - Автоматически проверяемые условия корректности
 - Визуальный анализ
- Анализ результатов регистрации
 - Большой объём
 - Начало регистрации «по событию»

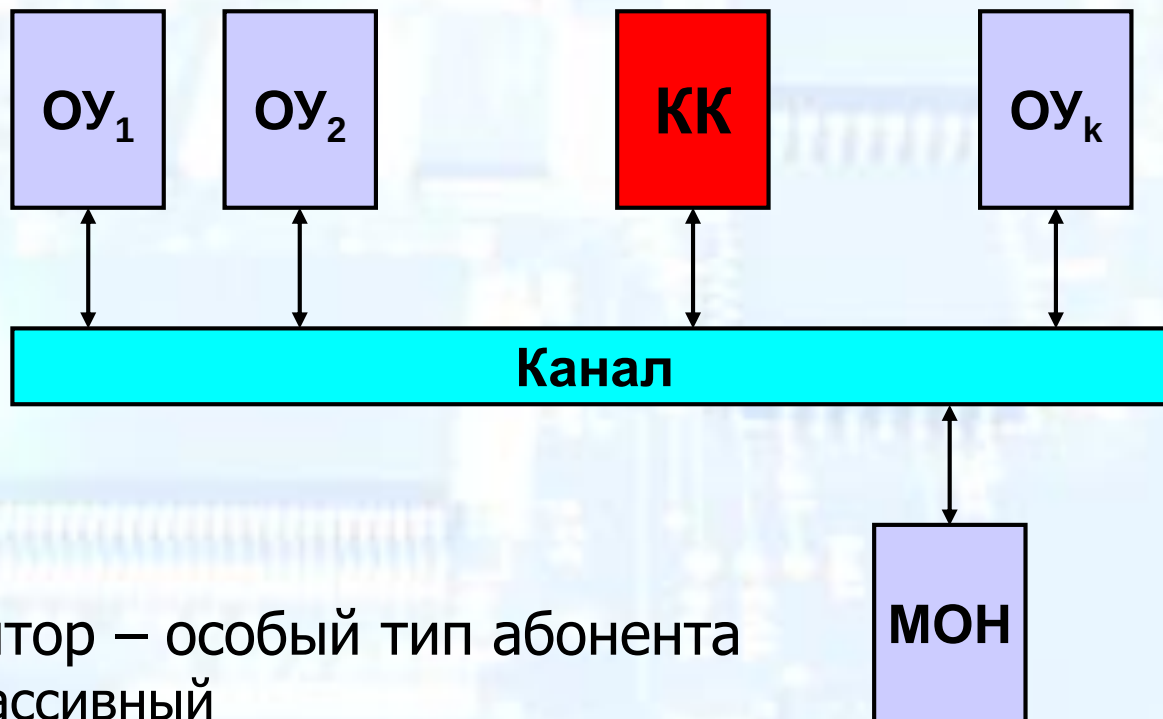


Доступа к каналу точка-точка

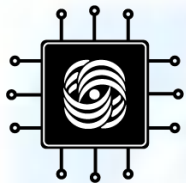




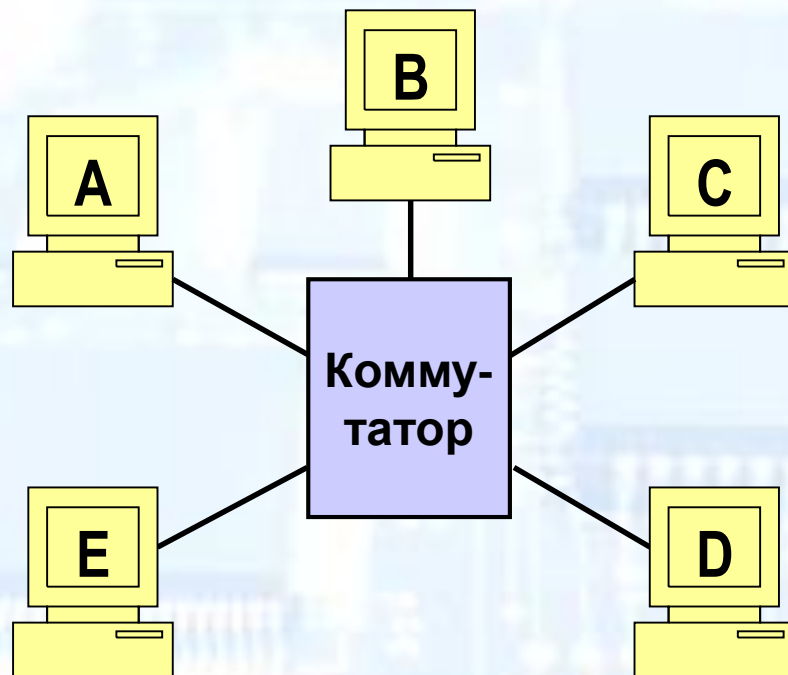
Доступ к общей шине



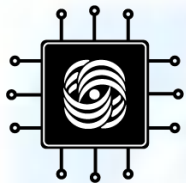
- Монитор – особый тип абонента
 - Пассивный
 - Без собственного адреса



Доступ к коммутируемой сети



- Слушаем выбранные линии как каналы «точка-точка»
 - Коммутационная панель
 - Каждый канал выведен с разрывом
 - Варианты замыкания: напрямую или через разветвитель
 - Громоздко...
- Порт мониторинга на коммутаторе
 - «сливаются» только корректные кадры, воспринятые коммутатором
 - Нарушение временных характеристик потока
 - Возможное превышение пропускной способности порта
 - Компактное решение

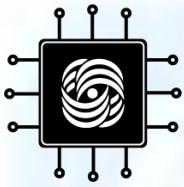


Низкоуровневый анализ



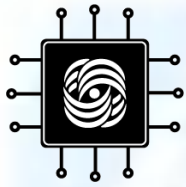
- Осциллограф:
 - Физические характеристики сигнала
 - Правильность формирования слова/кадра, повторяемого по каналу (можно «непосредственно» увидеть)
 - ...и даже значения данных (в повторяемых кадрах)



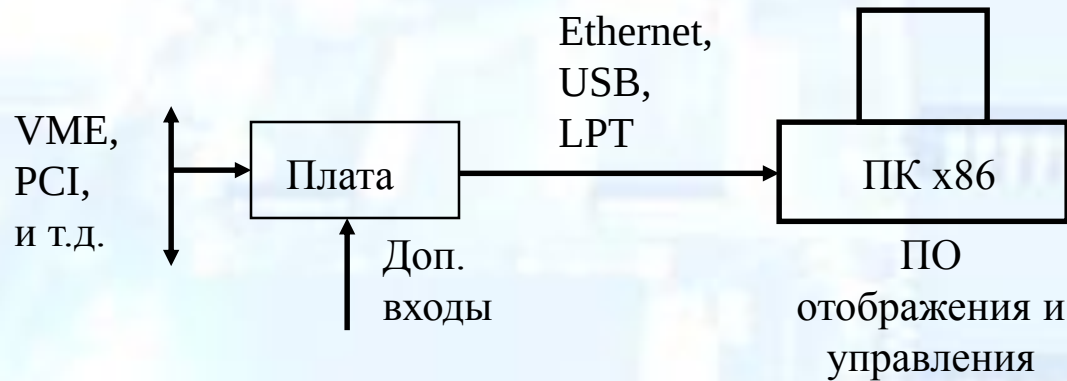


Анализ канальных протоколов

- Виды представления данных
 - последовательность обменов
 - быстро обновляется, большой объём
 - фильтрация, поиск
 - постоянный мониторинг + запуск регистрации по событию
 - статистика обменов
 - по абонентам
 - по потокам данных (виртуальные каналы, метки слов, пары отправитель-получатель)
 - статистика ошибок



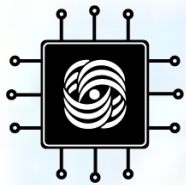
Анализаторы шин VME/PCI



Инструментальные средства анализа:

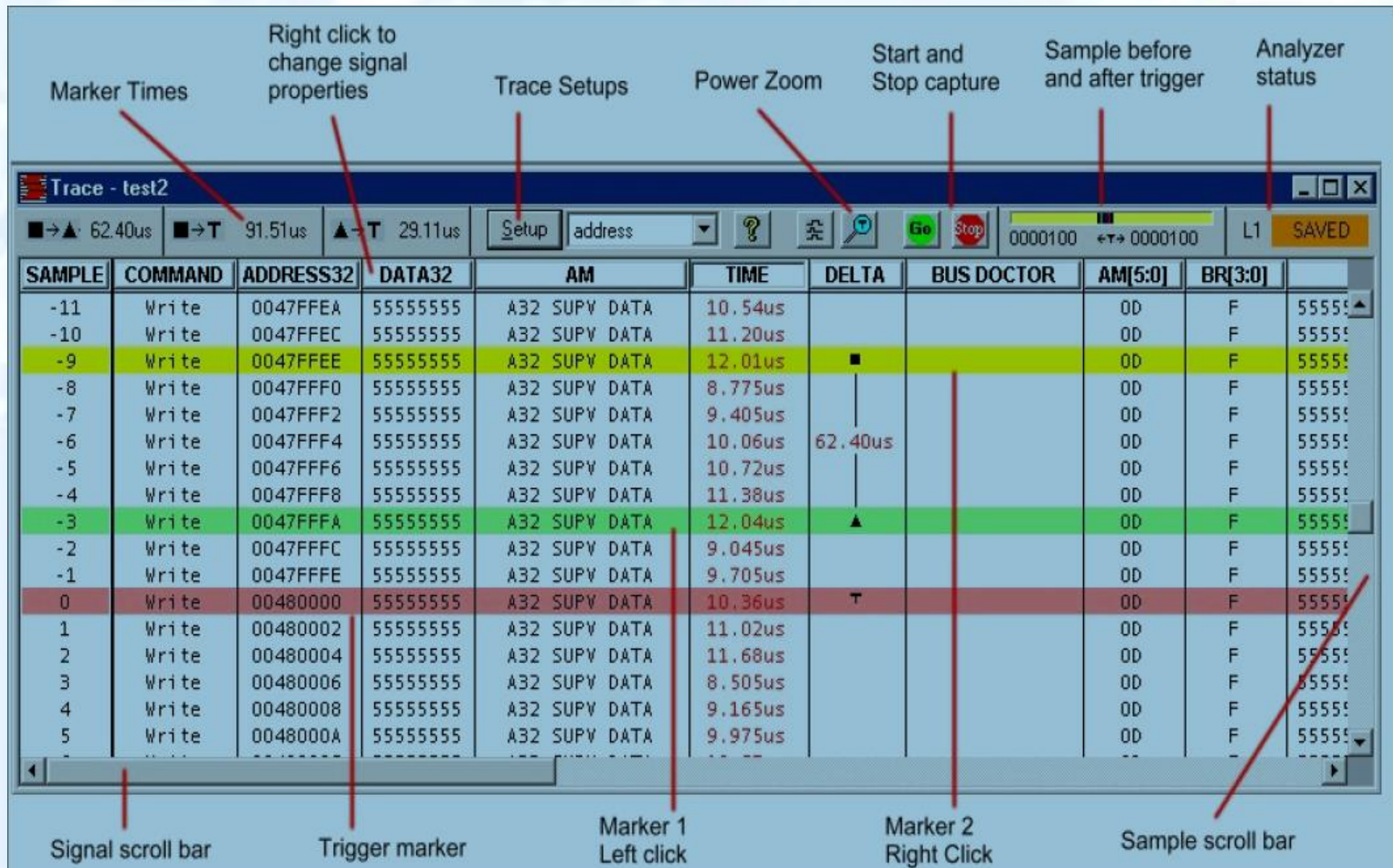
- Silicon Control Inc
- Curtiss-Wright Electronics systems / VMETRO
- LeCroy Inc
- Tektronix Inc
- Гранит-BT

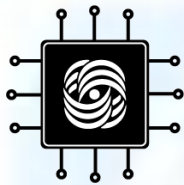




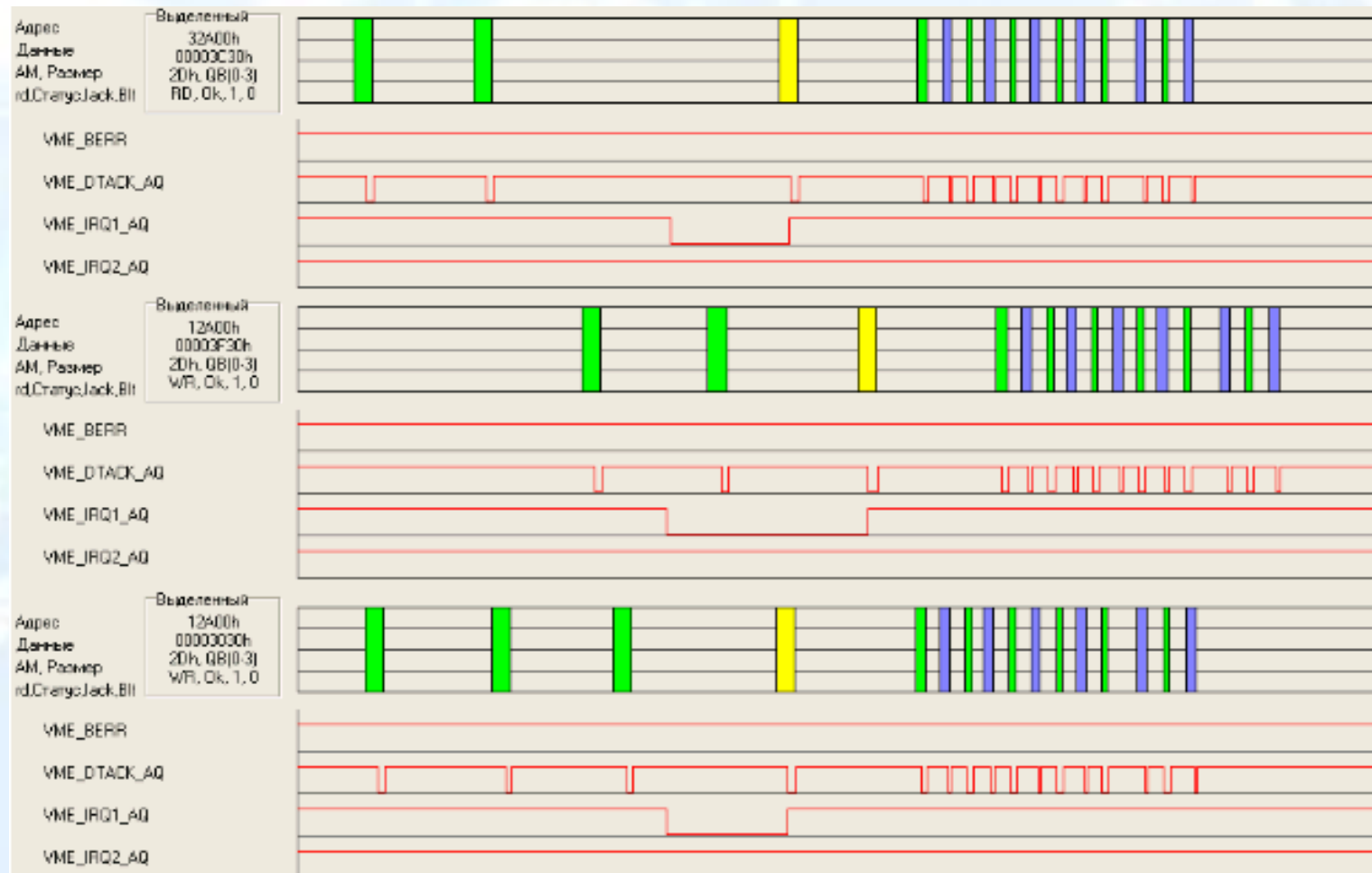
Пример анализаторов шин:

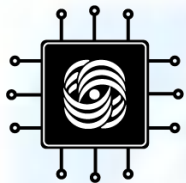
Таблица обменов





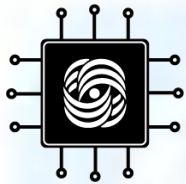
Пример анализаторов шин: Временная диаграмма





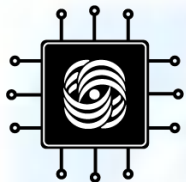
Архитектура средств мониторинга





Поддерживаемые протоколы

- MIL STD-1553B
- ARINC 429
- CAN
- Fibre Channel (оптика/данные)
- ARINC 818 (оптика/видео)



Анализатор MIL STD-1553B

Панель инструментов Боковая панель Область отображения результатов

Текущий сеанс: "2012-07-02 14:07" <@standvm-lenny>

2 июля 2012 Время: 15:00 - 16:00 Длительность: 1:01

Каналы

Имя	Тип	ОН	P	B	OY
mils	MKIO				
mils1	MKIO				

Свойство Протокол

Имя	значение
Имя	mils
Протокол	По умолчанию
Тип	M1553
MK	☒
Размещение lizard:0	
KK	☒
Размещение lizard:0	

Добавить МК Добавить КК

Добавить ОУ Удалить

[MKIO] mils (7439) 0.89% Интервал: 15:05:07.779 - 16:06:07.780 Обмен Слова данных Статистика

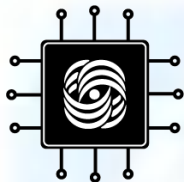
№	Время	Дата	Δt	Формат	Отправитель	Получатель
5242	15:56:09.778	2 Июл 2012	0:00:00.000	2 (OY->KK)	0x01:0x01	KK
5243	15:56:09.779	2 Июл 2012	0:00:00.000	2 (OY->KK)	0x01:0x01	KK
5244	15:56:09.780	2 Июл 2012	0:00:00.000	2 (OY->KK)	0x01:0x01	KK
	15:56:09.780	2 Июл 2012	Пауза: 0:00:00.991			
5245	15:56:10.772	2 Июл 2012	0:00:00.991	2 (OY->KK)	0x01:0x01	KK
5246	15:56:10.773	2 Июл 2012	0:00:00.000	2 (OY->KK)	0x01:0x01	KK
5247	15:56:10.773	2 Июл 2012	0:00:00.000	2 (OY->KK)	0x01:0x01	KK
5248	15:56:10.774	2 Июл 2012	0:00:00.000	2 (OY->KK)	0x01:0x01	KK

[MKIO] mils (4920) 0.76% Интервал: 14:58:57.784 - 15:59:57.785 Обмен Слова данных Статистика

№	Время	Дата	Δt	Формат	Отправитель
3036	15:57:19.780	2 Июл 2012	0:00:00.000	2 (OY->KK)	0x01:0x01
	15:57:19.780	2 Июл 2012	Пауза: 0:00:00.991		
3037	15:57:20.772	2 Июл 2012	0:00:00.991	2 (OY->KK)	0x01:0x01
3038	15:57:20.773	2 Июл 2012	0:00:00.000	2 (OY->KK)	0x01:0x01
3039	15:57:20.773	2 Июл 2012	0:00:00.000	2 (OY->KK)	0x01:0x01
3040	15:57:20.774	2 Июл 2012	0:00:00.000	2 (OY->KK)	0x01:0x01
3041	15:57:20.775	2 Июл 2012	0:00:00.000	2 (OY->KK)	0x01:0x01
3042	15:57:20.775	2 Июл 2012	0:00:00.000	2 (OY->KK)	0x01:0x01

14:07:32: Сеанс "2012-05-05 15:22" успешно загружен.
14:08:02: Сеанс "2012-07-02 14:07" успешно создан.
14:08:58: Error occurred while processing bind request. Requested device busy.
14:09:04: Error occurred while processing bind request. Requested device busy.
14:09:04: Error occurred while processing bind request. Requested device busy.

Область информационных сообщений Статусная строка



Троированный канал

Текущий сеанс: "Default" <@standvm-jenny>

2 февр. 2012 Время: 00:04.406850 - 01:04.406850 Длительность: 01:00.000000 Ч.М:

Каналы

Имя	Тип	ОН	P
mils	МКИО		
mils1	МКИО		
mils2	МКИО		

Имя: mils Тип: М1553 Размещение: lizard:3

Добавить МК Удалить

18:14:10: Сеанс "Default" успешно загружен.

[МКИО] mils (158) 0.00% Интервал: 16:00:04.406850 - 16:01:04.406850 Обмен Слова данных Статистика

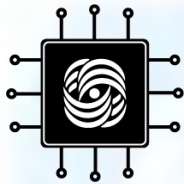
№	Время	Дата	Δt	Формат	Отправитель	Получатель	Размер
1240	16:01:04.006490	2 Фев 2012	0:00:00.000018	3 (ОУ->ОУ)	0x17:0x02	0x0b:0x16	16
	16:01:04.006900	2 Фев 2012	Пауза: 0:00:00.000127				
1241	16:01:04.007027	2 Фев 2012	0:00:00.000127	4 (КК(КС)->ОУ)	КК	0x1d	0
1242	16:01:04.007082	2 Фев 2012	0:00:00.000010	10 (КК(КС+СД)...	КК	0x1f	1
1243	16:01:04.007135	2 Фев 2012	0:00:00.000013	1 (КК->ОУ)	КК	0x15:0x0e	32
1244	16:01:04.007833	2 Фев 2012	0:00:00.000013	1 (КК->ОУ)	КК	0x18:0x17	15
1245	16:01:04.008191	2 Фев 2012	0:00:00.000013	2 (ОУ->КК)	0x07:0x19	КК	17

[МКИО] mils (0) Интервал: 18:13:10.000000 - 18:14:10.000000 Обмен Слова данных Статистика

№	Время	Дата	Δt	Формат	Отправитель	Получатель	Размер
1196	16:01:04.308816	2 Фев 2012	0:00:00.000019	3 (ОУ->ОУ)	0x03:0x07	0x15:0x05	10
1197	16:01:04.309116	2 Фев 2012	0:00:00.000010	10 (КК(КС+СД)...	КК	0x1f	1
	16:01:04.309156	2 Фев 2012	Пауза: 0:00:00.089000				
1198	16:01:04.398156	2 Фев 2012	0:00:00.089000	1 (КК->ОУ)	КК	0x06:0x1a	12
1199	16:01:04.398455	2 Фев 2012	0:00:00.000014	2 (ОУ->КК)	0x04:0x16	КК	32
1200	16:01:04.399160	2 Фев 2012	0:00:00.000020	3 (ОУ->ОУ)	0x00:0x02	0x07:0x0b	22
1201	16:01:04.399700	2 Фев 2012	0:00:00.000010	10 (КК(КС+СД)...	КК	0x1f	1

[МКИО] mils (0) Интервал: 18:13:10.000000 - 18:14:10.000000 Обмен Слова данных Статистика

№	Время	Дата	Δt	Формат	Отправитель	Получатель	Размер
1056	16:01:04.404460	2 Фев 2012	0:00:00.000015	1 (КК->ОУ)	КК	0x03:0x0b	30
1057	16:01:04.405118	2 Фев 2012	0:00:00.000013	1 (КК->ОУ)	КК	0x10:0x18	25
1058	16:01:04.405676	2 Фев 2012	0:00:00.000013	2 (ОУ->КК)	0x05:0x1c	КК	29
1059	16:01:04.406320	2 Фев 2012	0:00:00.000019	3 (ОУ->ОУ)	0x17:0x02	0x0b:0x16	16
1060	16:01:04.406742	2 Фев 2012	0:00:00.000012	4 (КК(КС)->ОУ)	КК	0x1d	0
1061	16:01:04.406798	2 Фев 2012	0:00:00.000011	10 (КК(КС+СД)...	КК	0x1f	1
1062	16:01:04.406850	2 Фев 2012	0:00:00.000012	1 (КК->ОУ)	КК	0x15:0x0e	32



Информация о сообщении

[МКИО] mls (5309) Интервал: 15:14:25.000 - 15:44:33.190 Обмен Слова данных Статистика КК Индикация параметров

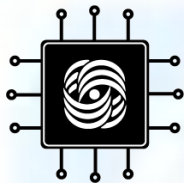
Формат	Отправитель	Получатель
2 (0У->КК)	0x01:0x01	КК
3 (0У->0У)	0x01:0x02	0x02:0x02
8 (0У->0У...)	0x01:0x03	0x1F:0x01

Время 15:44:33.190 **Получатель** 0x1F:0x01 **КС** 0xF824, 0x0C64
Дата 4 Июл 2012 **Отправитель** 0x01:0x03 **ОС** 0x0800
Формат 8 (0У->0У...) **Размер** 4

Сообщение передаётся

СД	0-7	0xABCD	0xEFEF	0xDCBA	0xFEFE				
8-15									
16-23									
24-31									

Количество	1769
Всего ошибок	0
Нет ответного слова	0
Ошибка в сообщении	0
Абонент занят	0
Ошибка абонента	0
Ошибка 0У	0
Ошибка четности/кодировки	0
Нарушение непрерывности	0
Слишком много слов данных	0
Ошибка адреса 0У	0
Ошибка синхроимпульса	0
Вытеснение	0
Нарушение КΣ	0
Прочие ошибки	0



Статистика обмена

[МКИО] mls (158)

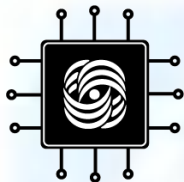
Интервал: 16:00:04.406850 - 16:01:04.406850

Обмен

Слова данных

Статистика

№	Адрес	Количество	Нет ОС	Ошибка сообщения	Абонент занят	Абонент неисправен	ОУ неисправно
1	0x00	8	0	0	0	0	0
2	0x01	5	0	0	0	0	0
3	0x02	4	0	0	0	0	0
4	0x03	9	0	0	0	0	0
5	0x04	9	0	0	0	0	0
6	0x05	10	0	0	0	0	0
7	0x06	5	0	0	0	0	0
8	0x07	13	0	0	0	0	0
9	0x0b	9	0	0	0	0	0
10	0x10	5	0	0	0	0	0
11	0x11	4	0	0	0	0	0
12	0x12	14	0	0	0	0	0
13	0x15	9	0	0	0	0	0
14	0x16	4	0	0	0	0	0
15	0x17	10	0	0	0	0	0



Фильтрация обмена

Формат Адрес Подадрес

Формат обменов:

- ☒ 01 KK->OY
- ☐ 02 OY->KK
- ☒ 03 OY->OY
- ☒ 04 KK(KY)->OY
- ☒ 05 OY(OC+CD)->KK
- ☒ 06 KK(KY+CD)->OY
- ☒ 07 KK->OY...
- ☒ 08 OY->OY...
- ☒ 09 KK(KY)->OY...
- ☒ 10 KK(KY+CD)->OY..

Все Данные

Групповые Очистить

Формат Адрес Подадрес

Адрес:

A00	A01	A02	A03
Rx	Rx	Rx	Rx
A04	A05	A06	A07
Rx	Rx	Rx	Rx
A08	A09	A0a	A0b
Rx	Rx	Rx	Rx
A0c	A0d	A0e	A0f
Rx	Rx	Rx	Rx
A10	A11	A12	A13
Rx	Rx	Rx	Rx
A14	A15	A16	A17
Rx	Rx	Rx	Rx
A18	A19	A1a	A1b
Rx	Rx	Rx	Rx
A1c	A1d	A1e	A1f
Rx	Rx	Rx	Rx

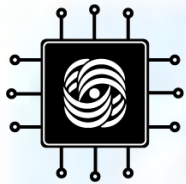
T/R Tx

Rx Off

Формат Адрес Подадрес Ошибки

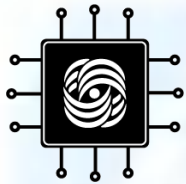
☒ Учитывать ошибки

- ☒ Нет ответного слова
- ☒ Ошибка в сообщении
- ☒ Абонент занят
- ☒ Ошибка абонента
- ☒ Ошибка OY
- ☒ Ошибка четности/кодировки
- ☒ Нарушение непрерывности
- ☒ Слишком много слов данных
- ☒ Ошибка адреса OY
- ☒ Ошибка синхроимпульса
- ☒ Вытеснение
- ☒ Нарушение КΣ
- ☒ Прочие ошибки



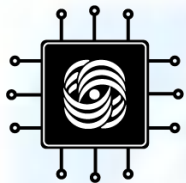
Анализ передаваемых данных

- Проверяемые условия:
 - Обновление данных
 - Гладкость: $|A_i - A_{i-1}| < d$
 - Пороговое сравнение с эталоном
 - Сравнение по маске с эталоном или другим параметром
- Анализ графиков изменения параметров
- Интеграция с БД протоколов информационного взаимодействия



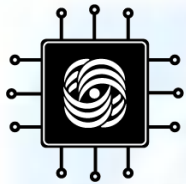
Мониторинг межзадачного обмена

- Агент отладки
- Запись трассы + сброс в технологический порт
- «Снимки» интерфейсных переменных по внешнему запросу
- Мониторинг системной информации
- Показатели реального времени искажаются!



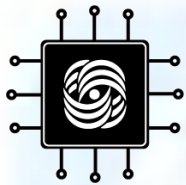
Мониторинг переменных

- Инструмент мониторинга переменных позволяет получить доступ к значениям переменных и содержимому ячеек памяти в защищённых разделах памяти целевого вычислителя.
- Поддерживаются глобальные и статические переменные как пользовательских процессов, так и процесса ядра.
- Компонент позволяет получать и отображать значения, а также сохранять их в файле трассы.

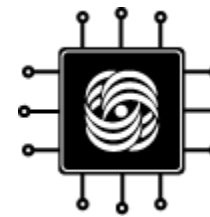


Мониторинг системной информации

- сведений о распределении памяти;
- информации о каналах;
- информации об условной переменной;
- содержимого памяти;
- информации о канале межмодульной передачи данных;
- информации об очереди сообщений;
- информации о мьютексе;
- информации о порте с очередью сообщений;
- информации о расписаниях и окнах;
- информации о сегментах;
- информации о семафоре;
- сведений о состоянии потока управления;
- информации о таймере;
- статистики по использованию окон.



Спасибо за внимание!



Лаборатория
Вычислительных
Комплексов

ИНФОРМАЦИОННО-УПРАВЛЯЮЩИЕ СИСТЕМЫ РЕАЛЬНОГО ВРЕМЕНИ

Лекция 12: Жизненный цикл ПО ИУС РВ и инструментальные средства его поддержки

Кафедра АСВК,
Лаборатория Вычислительных Комплексов
Балашов В.В.

РАЗРАБОТКА ПО ИУС РВ – ТЕХНОЛОГИЧЕСКИ И ОРГАНИЗАЦИОННО СЛОЖНЫЙ ПРОЦЕСС

На примере разработки бортовых ИУС РВ

Комплекс бортового оборудования

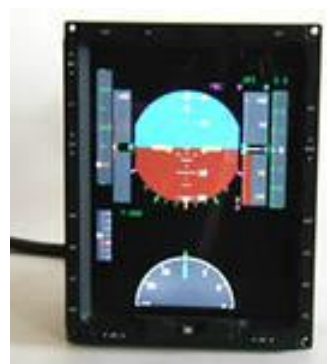


Специфика бортовых ИУС РВ

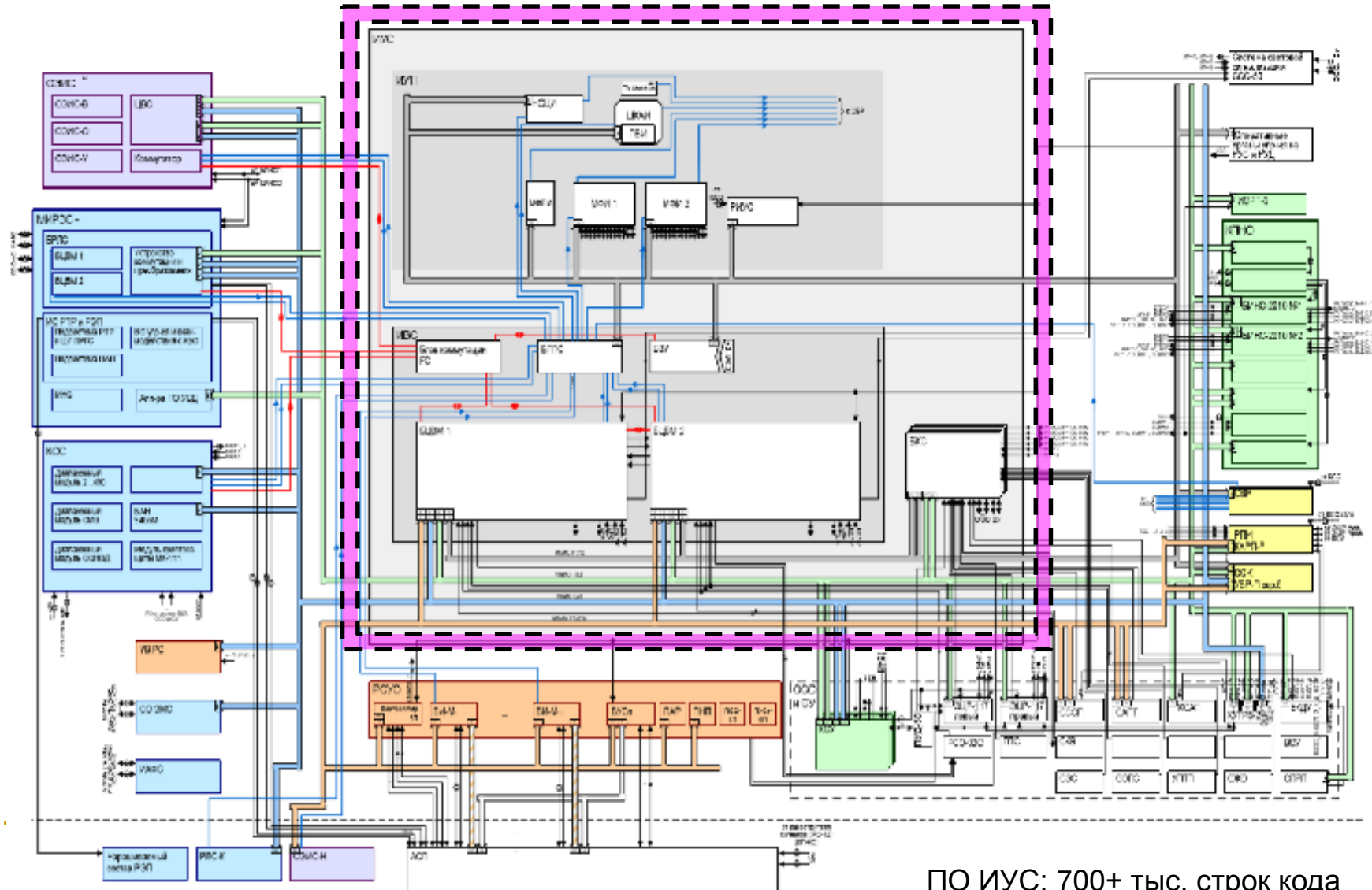
- Высокая сложность
- Функционирование в реальном времени
 - Вычисления
 - Информационный обмен
- Требования
 - Функциональность ← разнообразие
 - Надёжность
 - Реальное время
- Критичность
- Неоднородность
 - Каналы: точка-точка, шина, коммутатор; 12 kbps, 1 Mbps, 1 Gbps
 - Устройства: датчики, индикаторы, вычислители, органы управления, исполнительные устройства
 - Данные: аналоговые, цифровые; числовые массивы, видеопотоки

Бортовые ИУС РВ и их ПО требуют систематического подхода к проектированию, реализации и тестированию

Бортовые устройства



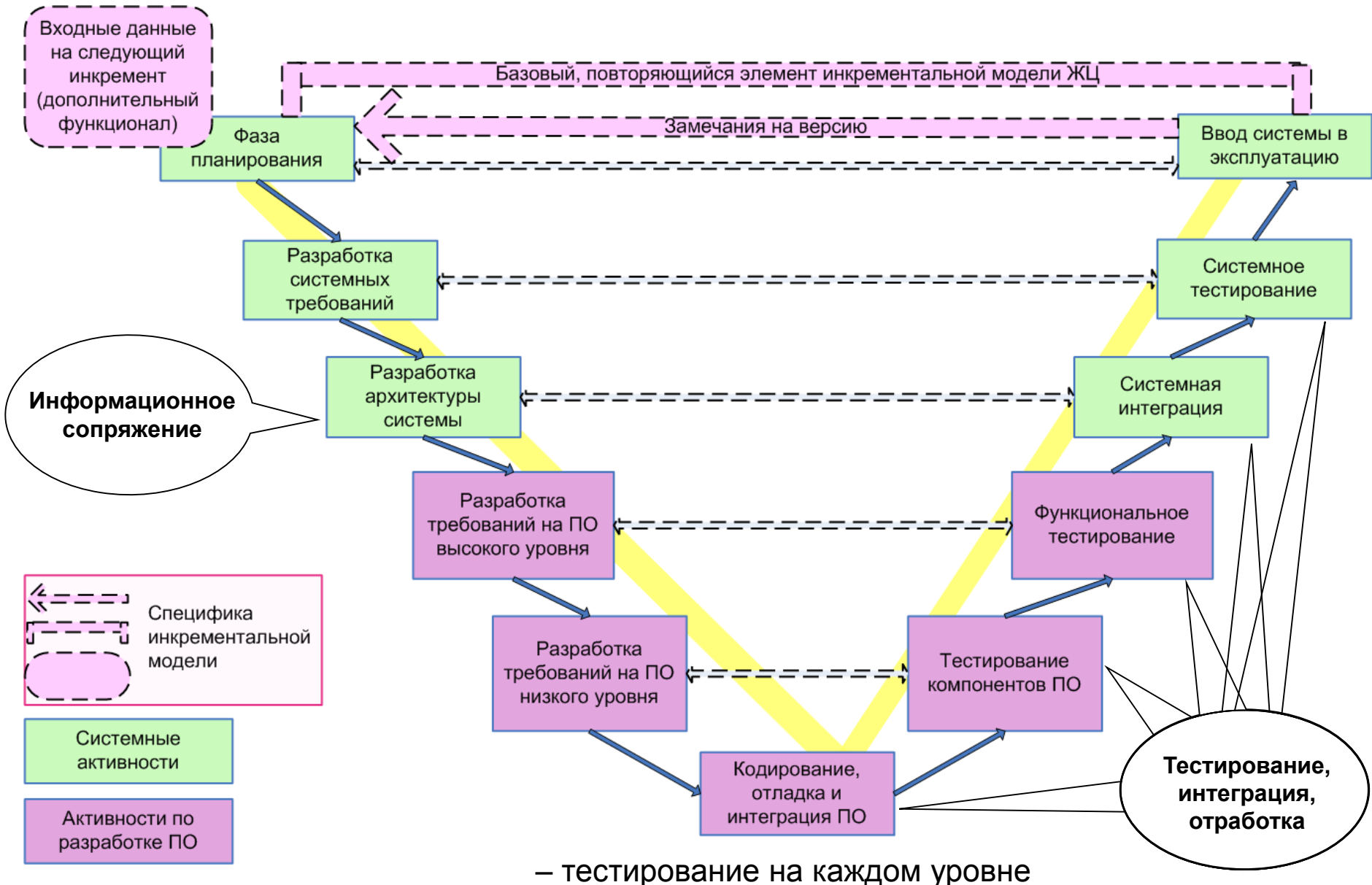
Уровень сложности комплекса бортового оборудования



ПО ИУС: 700+ тыс. строк кода

Жизненный цикл бортового ПО

Жизненный цикл бортового ПО



Процессы ЖЦ по стандарту DO-178B

Software Considerations in Airborne Systems and Equipment

- Процесс планирования и управления проектом
- Процесс разработки ПО
 - Разработка требований
 - Проектирование
 - Кодирование, отладка и интеграция
- Интегральные процессы
 - Управление конфигурацией
 - Верификация
 - Управление качеством
 - Сертификационное взаимодействие
(в данном курсе не рассматривается)

Влияние на состав фаз ЖЦ

Процессы

Процесс	Подпроцессы	Основные цели	Основные артефакты
Планирование и управление проектом		Определение основной методологии разработки, написание планов разработки, определение инструментов разработки, определение и проведение необходимых тренингов, определение методов мониторинга проекта. Выпуск необходимых шаблонов.	Планы (разработки, тестирования ПО, конфигурационного управления, управления качеством и т.д.); отчеты о рецензировании планов; графики; стандарты; шаблоны артефактов (документов) проекта; протоколы совещаний; официальные письма; рабочая переписка (e-mail); метрики и индикаторы, отражающие состояние процессов и прогресс, презентации; списки мероприятий.

Процессы

Процесс	Подпроцессы	Основные цели	Основные артефакты
Разработка программного обеспечения	Процесс разработки требований к ПО и процесс проектирования ПО	Создание и контроль требований верхнего и нижнего уровня на разработку программного обеспечения. Контроль полноты и непротиворечивости требований, прослеживаемости и тестируемости.	Спецификации требований в текстовом формате. Если требования разрабатываются с использованием инструментальных средств, то модули в формате инструментального средства. Формальные спецификации (данные, приготовленные для автоматической генерации кода) в формате их редакторов; формальные спецификации в формате HTML, или текстовом формате; отчеты о рецензировании формальных спецификаций; описание графической информации в необходимом объеме; Результаты автоматического контроля формальных спецификаций; матрица соответствия элементов формальных спецификаций и требований к ним с отчетом о рецензировании; сгенерированный код.

Процессы

Процесс	Подпроцессы	Основные цели	Основные артефакты
Разработка программного обеспечения	Процесс кодирования ПО и процесс интеграции	Разработка программного кода в соответствии со спецификациями, получение загрузочных файлов. В процессе кодирования исходный код составляется на основе архитектуры ПО и требований нижнего уровня. Формирование исполняемого кода, загрузка исполняемого объектного кода на целевой вычислитель для интеграции ПО и аппаратных средств.	Код, разработанный вручную; служебные файлы, необходимые для компиляции и сборки кода; матрица покрытия требований кодом; код, сгенерированный автоматическими средствами генерации. Объектные файлы. Загрузочный файл

Процессы

Процесс	Основные цели	Основные артефакты
Верификация программного обеспечения	Интегральный процесс, целью которого является обнаружение и регистрация ошибок, которые могли появиться в процессе разработки ПО. Эти цели достигаются через ряд мероприятий: написание тестовых процедур в соответствии с требованиями, проведение анализа кода и тестовых процедур, проведение формального тестирования, анализ полученных результатов, написание сообщений о проблемах, найденных в процессе различных верификационных активностях, создание и выпуск отчетов о проведенном тестировании.	Отчет об автоматическом анализе кода, отчеты о рецензировании кода. Тестовые процедуры и скрипты в формате редакторов тестов и текстовом формате, в случае использования инструментальных средств разработки требований и тестовых процедур – модули в формате инструментального средства; отчеты о рецензировании тестов; отчеты о рецензировании результатов тестирования; матрица покрытия требований тестами с отчетом о её рецензировании; матрица результатов тестирования с отчетом о её рецензировании; реестр со списком всех верификационных документов для версии ПО.

Процессы

Процесс	Основные цели	Основные артефакты
Управление качеством	Интегральный процесс, целью которого является контроль качества продукта и контроль соблюдения процессов.	Отчеты (протоколы) проверки качества продукции; отчеты (протоколы) проверки процессов; планы проверки.

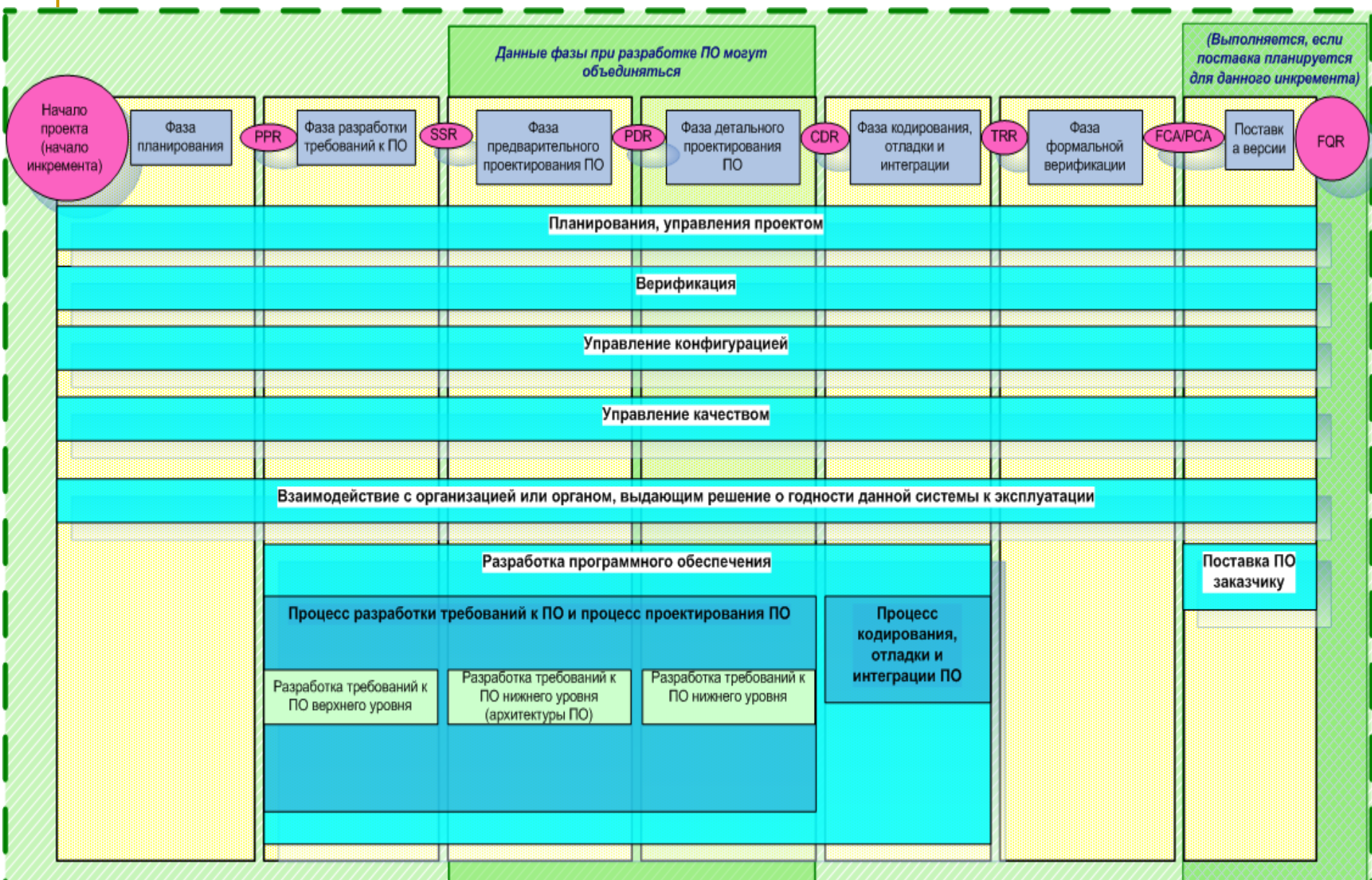
Процессы

Процесс	Основные цели	Основные артефакты
Конфигурационное управление, включая управление изменениями	Интегральный процесс, целью которого является: - определение и идентификация продукта, - контроль выпуска и изменения версий в течение жизненного цикла, - контроль корректности и полноты версии, - версионный контроль всех артефактов проекта. Определение конфигурационной структуры, обеспечение однозначной конфигурационной идентификации артефактов проекта. Установление и проведение процесса управления изменениями.	Поставки (файлы-архивы) и сопровождающие файлы-отчеты по каждому из процессов; описание версии ПО; сопровождающая документация; носитель версии (оптический диск), если это оговорено – с необходимыми атрибутами (наклейками или другой идентифицирующей информацией); конфигурационный перечень артефактов проекта. Базы данных проблем - описания проблем (в формате, пригодном для импорта в средство управления изменениями; в текстовом, или ином удобном для чтения формате), протоколы совещаний по управлению конфигурацией. В случае наличия независимых баз данных у различных участников кооперации, предусмотреть наличие процесса и артефактов для поддержки синхронизации информации в этих базах данных.

Процессы

Процесс	Основные цели	Основные артефакты
Поставка ПО заказчику	Передача разработанного продукта заказчику.	Описание версии ПО; финальная версия поставляемого ПО; заключение о годности версии ПО к натурным испытаниям на целевой системе.

Соотношение фаз и процессов ЖЦ ПО



Фазы

Название фазы	Основные активности внутри фазы	Применяемые инструментальные средства
Фаза планирования	Определение методов и инструментов для производства программного продукта в процессе написания планов разработки ПО, написание стандартов разработки и создание необходимых шаблонов.	Инструменты, позволяющие: - осуществлять планирование (в т.ч. создавать графики разработки), - создавать планы разработки и тестирования, - создавать шаблоны разработки, - отслеживать ход выполнения проекта, - осуществлять коммуникацию, взаимодействие различных участников проекта.

Фазы

Название фазы	Основные активности внутри фазы	Применяемые инструментальные средства
Фаза разработки требований к ПО	Анализ входных требований. Написание требований верхнего и нижнего уровня на разработку ПО. Определение бортовых интерфейсов, структуры и протоколов среды обмена. Построение матрицы прослеживаемости между требованиями различного уровня. Написание формальных спецификаций. Начало написания тестовых процедур, соответствующих требованиям верхнего уровня.	Инструменты, позволяющие: - создавать требования, - создавать связи между требованиями, - разрабатывать и поддерживать интерфейсы между различными компонентами ПО, - проводить необходимые обзоры артефактов, - создавать тестовые процедуры, - разрабатывать формальные спецификации.

Фазы

Название фазы	Основные активности внутри фазы	Применяемые инструментальные средства
Фаза проектирования (дизайна) ПО	Определение архитектуры ПО, декомпозиция ПО, декомпозиция распределения памяти. Определение структуры ПО с учетом требований реального времени. Написание (детализация) тестовых процедур в соответствии с уточнениями и детализацией требований. Построение предварительной матрицы покрытия требований тестовыми процедурами.	Инструменты, позволяющие: - создавать и поддерживать описание архитектуры ПО, - прослеживать связи между требованиями различного уровня, - задавать и прослеживать соответствие тестовых процедур и требований, - осуществлять необходимые обзоры артефактов, - создавать матрицу покрытия требований тестовыми процедурами.

Фазы

Название фазы	Основные активности внутри фазы	Применяемые инструментальные средства
Фаза кодирования, отладки и интеграции	Генерация кода из формальных спецификаций. Разработка ручного кода. Интеграция всех разработанных программных компонент в единый модуль. Загрузка программного модуля на целевой вычислитель. Отладка разработанного кода и, параллельно, тестовых процедур. В конце фазы происходит «заморозка» программного кода и тестовых процедур.	Инструменты, позволяющие: - генерировать код на языке высокого уровня из формальных спецификаций; - разрабатывать ручной код; - проводить компиляцию исходного кода; - проводить компоновку двоичного кода; - проводить загрузку на целевой вычислитель и интеграцию; - проводить отладку на целевых вычислителях; - выполнять интеграционные тесты с имитацией данных на входах целевых вычислителей; - проводить коррекцию программного кода и тестовых процедур в случае обнаружения ошибок; - разрабатывать и выпускать необходимую документацию.

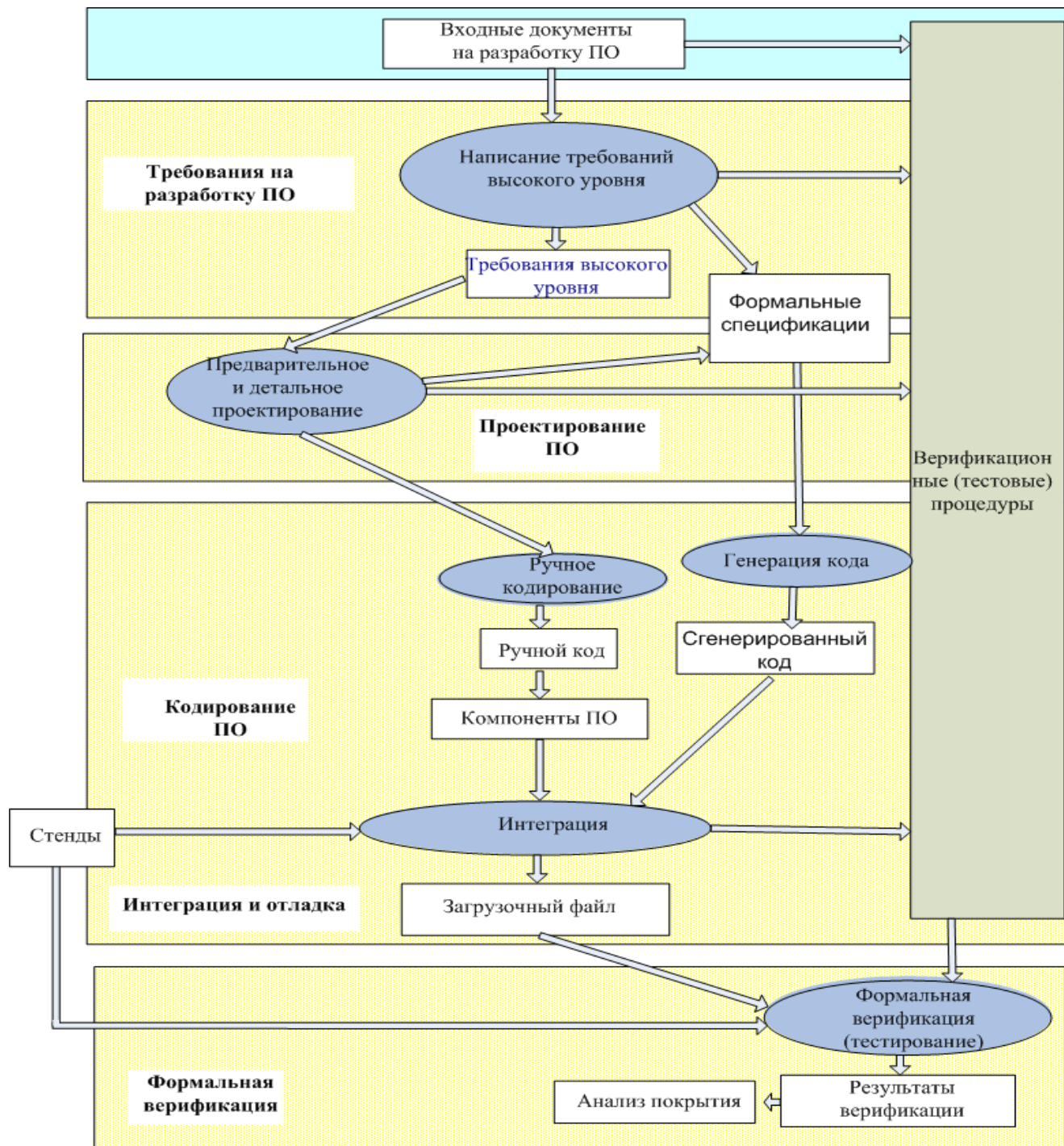
Фазы

Название фазы	Основные активности внутри фазы	Применяемые инструментальные средства
Формальная верификация	Проведение прогона разработанных тестовых процедур на «замороженной» версии ПО (тестирование). Анализ результатов тестирования. Создание сообщений о проблемах по результатам тестирования. Финализация матрицы покрытия.	Инструменты и средства, позволяющие: - осуществлять прогон тестируемого ПО на целевом вычислителе, - выполнять тесты с имитацией данных на входах целевых вычислителей, - собирать результаты тестирования, в т.ч. формировать протоколы тестирования, - проводить необходимые обзоры артефактов, - оформлять сообщения о проблемах, - модифицировать матрицу покрытия.

Фазы

Название фазы	Основные активности внутри фазы	Применяемые инструментальные средства
Подготовка финальной поставки	Выпуск итоговых документов по составу версии.	Инструменты, позволяющие создавать, модифицировать и выпускать итоговые документы проекта

Активности по фазам ЖЦ ПО



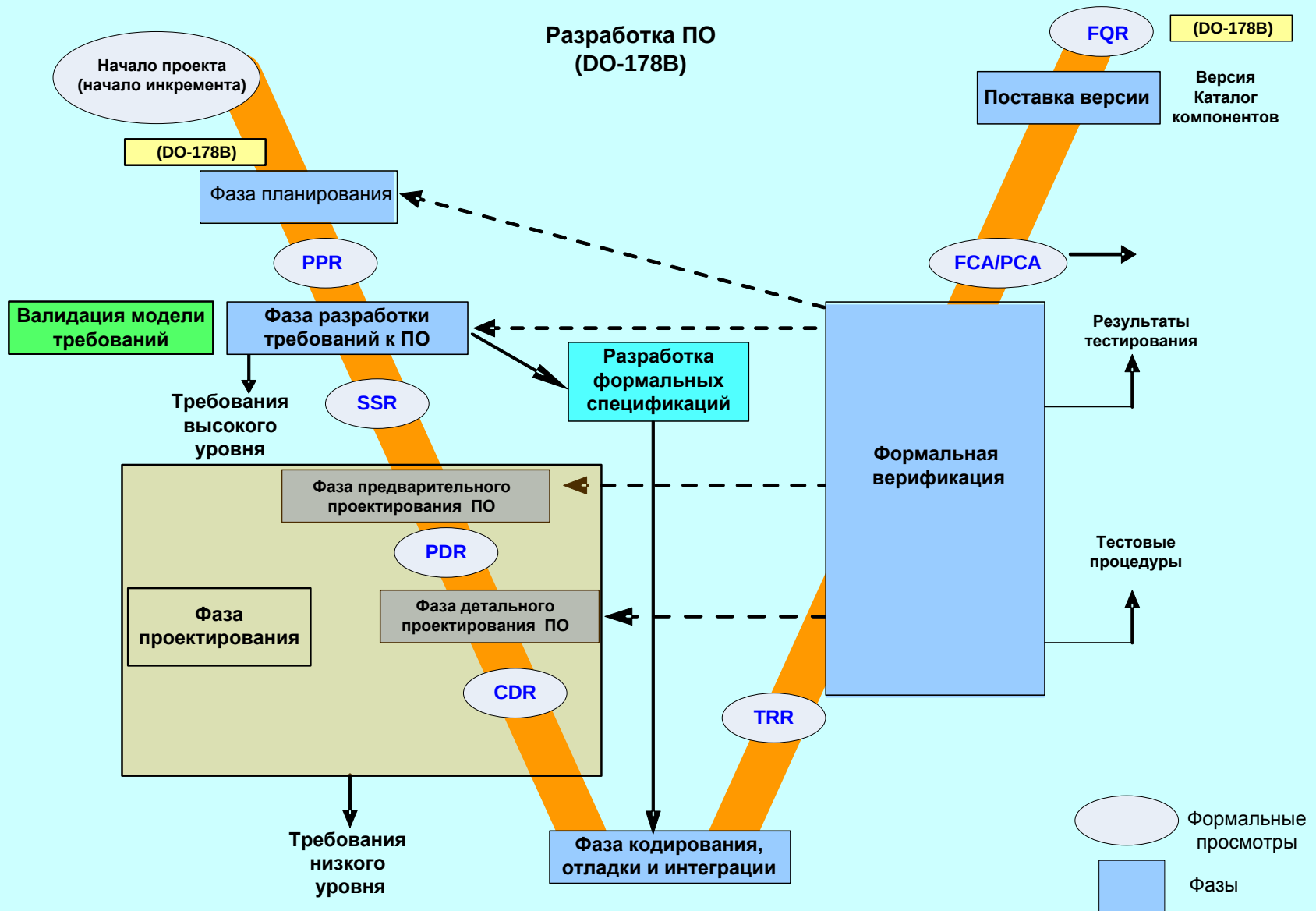
Вехи

Вехи		Аббревиа- тура	Описание
Русское название	Английское название		
Обзор планов	Plan Process Review	PPR	Одобрение, официальный выпуск планов разработки
Обзор спецификаций	Software Specification Review	SSR	Одобрение, официальный выпуск требований на разработку ПО и начало фазы предварительного дизайна
Предварительный обзор архитектуры	Preliminary Design Review	PDR	Одобрение, официальный выпуск предварительного дизайна (архитектуры ПО) и начало активностей по определению детального дизайна
Критический обзор архитектуры	Critical Design Review	CDR	Одобрение, официальный выпуск детального дизайна ПО
Обзор готовности тестов	Test Readiness Review	TRR	Начало выполнения формальной верификации (формального тестирования)

Вехи

Вехи		Аббревиа- тура	Описание
Русское название	Английское название		
Функциональный конфигурационный аудит (Функциональный аудит программных конфигураций)	Functional Configuration Audit	FCA	Проверка того, что элементы конфигурации отвечают всем функциональным требованиям, в том числе требованиям по производительности. FCA представляет обзор функционирования элементов, не только для обеспечения соответствия спецификации, но и для выявления не заданных требованиями функциональных характеристик.
Физический конфигурационный аудит (Физический аудит программных конфигураций)	Physical Configuration Audit	PCA	PCA обеспечивает гарантию того, что поставляемые результаты физически соответствуют перечисленным в документации пунктам и представлены в поставляемой версии.
Формальный квалификационный обзор	Formal Qualification Review	FQR	Успешное завершение этих аудитов может быть обязательным требованием для финального фиксирования версии.

Фазы и вехи на V-цикле



Инструментальные средства поддержки ЖЦ бортового ПО

Средства поддержки разработки требований

- Функциональность бортового ПО описывается десятками тысяч требований (системных; собственно к ПО)
- Необходимая функциональность:
 - создание и хранение требований, отслеживание истории
 - связывание требований с версиями документов и ПО
 - прослеживаемость требований на:
 - Низкоуровневые требования
 - Формальные спецификации
 - Код
 - Тесты
- Примеры средств: IBM DOORS, Borland CaliberRM, SyBase PowerDesigner

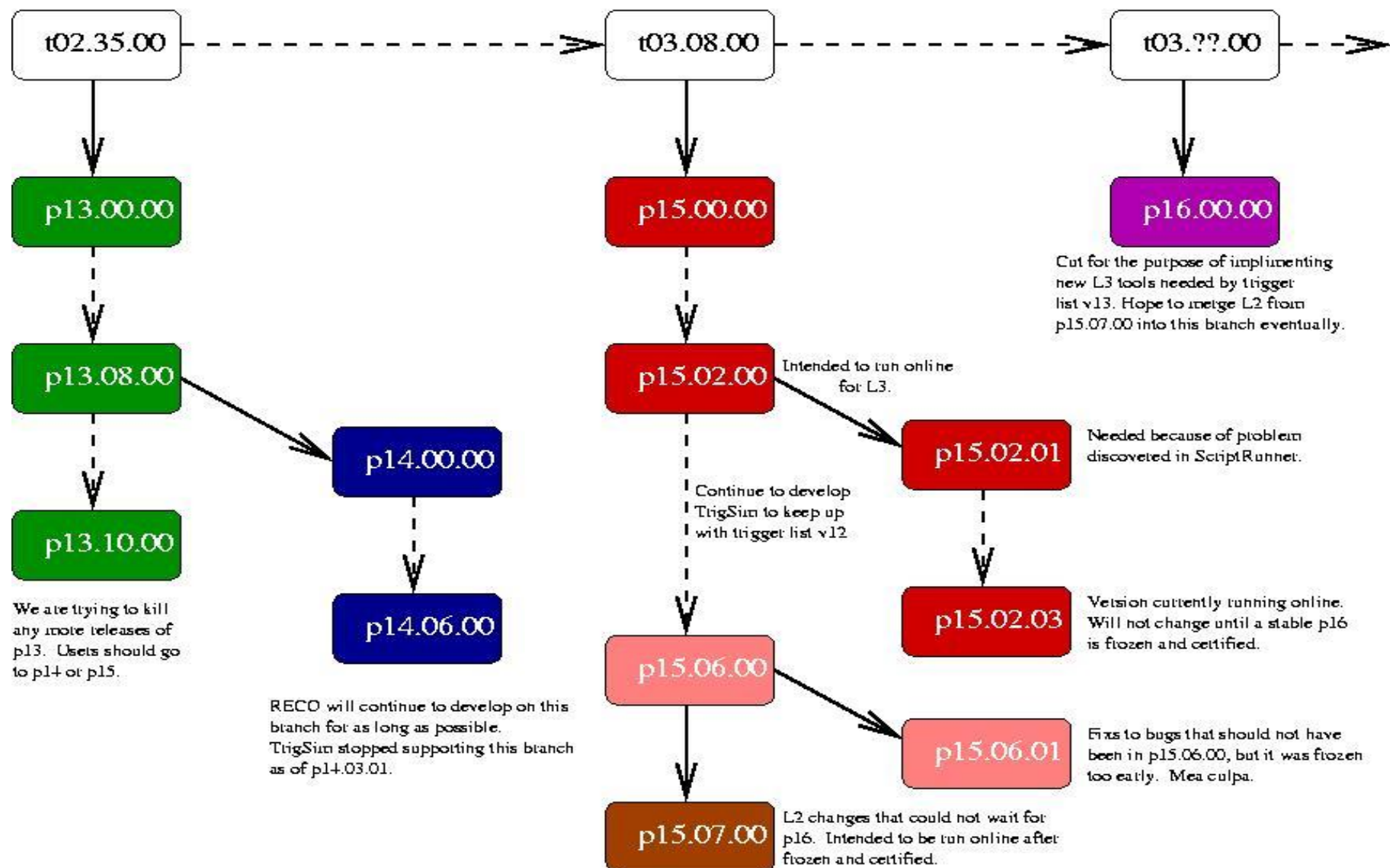
Примеры требований к подсистеме ИУС РВ

BCS.INIT.SELF	Встроенная самопроверка БЦВМ	
	После включения питания БЦВМ должна выполнять начальную встроенную самопроверку. Взаимодействие по каналам КБИ при этом не выполняется. Продолжительность встроенной самопроверки не менее 2 сек. и не более 8 сек. По результатам встроенной проверки загорается светодиод исправности БЦВМ.	
BCS.INIT.MILS	Контроль МКИО	
	После встроенной самопроверки БЦВМ должна выполнять контроль обменов по МКИО: - передачу групповой команды "Установить ОУ в исходное состояние"; - передачу групповой команды "Начать самоконтроль ОУ"; - опрос векторного слова самоконтроля у абонентов МКИО; Продолжительность встроенной самопроверки не более 3 сек.	
BCS.SKV.TEST.START	Включение режима 'Тест'	
	Не позднее, чем через 10 сек. после включения питания БЦВМ должна перевести СКВ в режим 'Тест'.	
BCS.SKV.TEST.DELAY	Продолжительность режима 'Тест'	
	Продолжительность режима 'Тест' СКВ должна быть не менее 2 сек.	

Средства версионного/ конфигурационного контроля

- При разработке ПО формируется множество документов, группируемых в версии
 - Промежуточные («инженерные») версии
 - Официальные версии
- Версия = вся совокупность документов
 - Требования, спецификации, код, тесты,...
 - Часть конфигурации борта
- Необходимая функциональность:
 - Версионирование совокупности документов (в т.ч. атомарность изменений)
 - Поддержка ветвей истории
 - Поддержка групповой разработки, в т.ч. разграничения доступа
 - Обновление документов в реальном времени
- Примеры: CVS, Subversion, git, IBM ClearCase

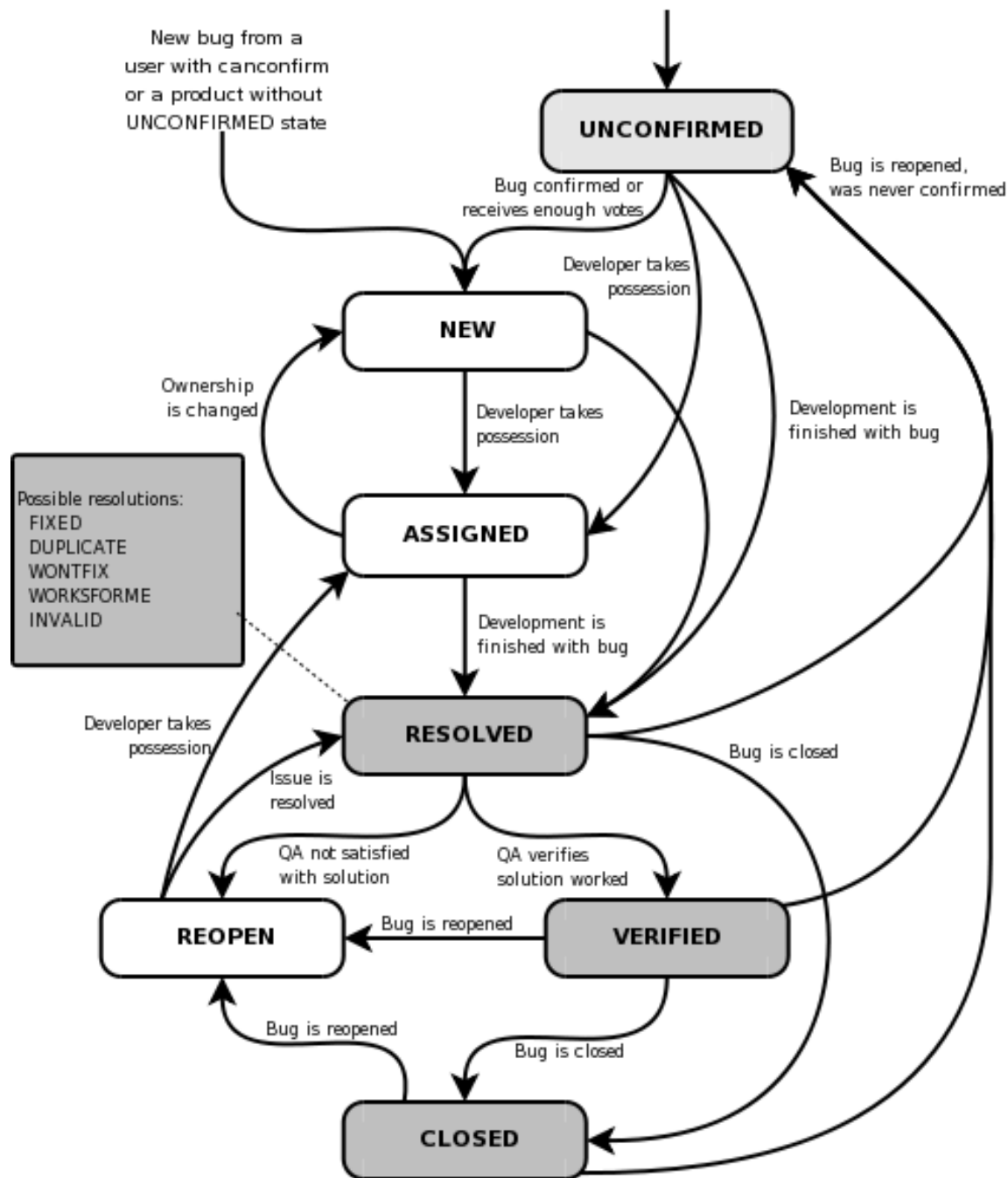
Дерево версий



Средства отслеживания проблем и изменений

- В процессе разработки ПО:
 - Дефекты и ошибки: зарегистрированы, проанализированы и устранены
 - Изменения: зарегистрированы, утверждены и реализованы
- Необходимая функциональность:
 - Поддержка структуры продукта и процесса
 - Настраиваемый формат сообщения о проблеме
 - Настраиваемый ЖЦ сообщения, поддержка согласования
 - Поддержка групповой разработки, втч разграничения доступа
 - Интеграция со средствами управления версиями
- Примеры: Bugzilla, Trac, IBM ClearQuest

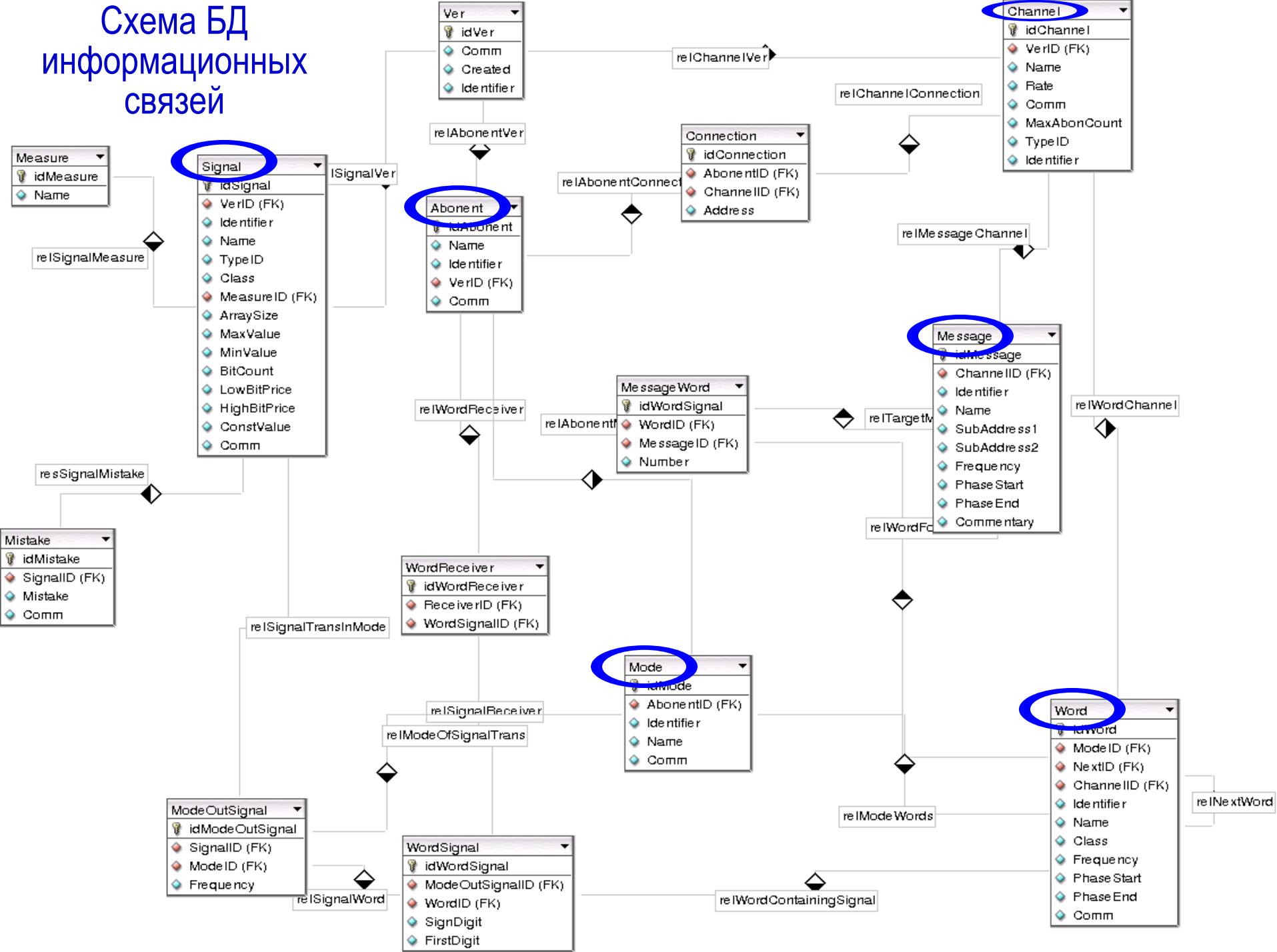
Жизненный цикл сообщения о проблеме



Средства поддержки сопряжения подсистем ПО

- Средства автоматизации проектирования бортовых интерфейсов
 - Балансировка загрузки каналов
 - Формирование набора сообщений
 - Построение расписаний обмена (канал с централизованным управлением)
 - Построение системы виртуальных каналов (сеть на основе коммутаторов)
- Средства автоматизации интеграции ПО
 - Использование унифицированных структурных компонентов ПО
 - По управлению: расписание выполнения СКПО
 - По данным:
 - БД информационных связей СКПО
 - Автоматическое формирование описания интерфейсов ПО (буфера, каналы, сообщения) для конфигурирования ОСРВ

Схема БД информационных связей



Средства автоматизации проектирования индикационных форматов

- Индикационный формат = набор графических элементов + правила поведения
- Необходимая функциональность:
 - Редактирование в графической форме, WYSIWYG
 - Поддержка библиотеки элементов
 - Поддержка автономного тестирования
 - Генерация кода в формате для целевого устройства
- Примеры: SCADE Display, VAPS, САПР ИФ

П/Т

НВГ

ОВО

БКС

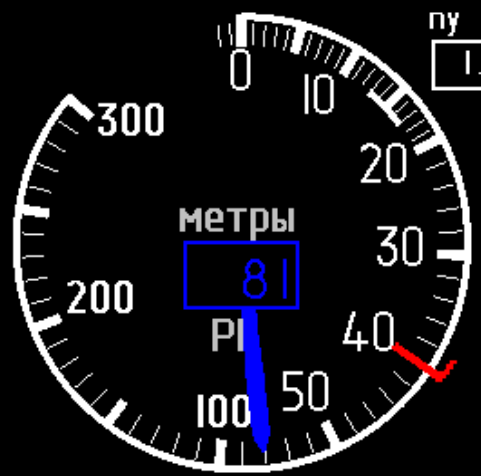
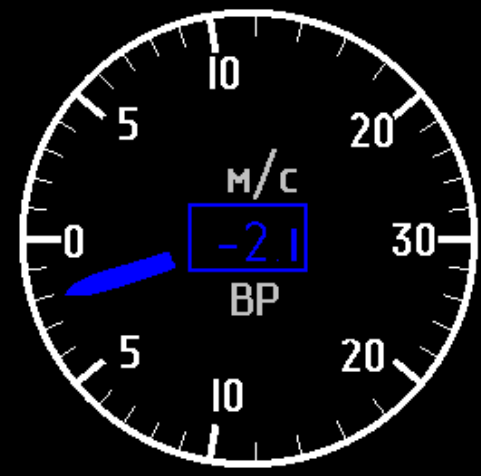
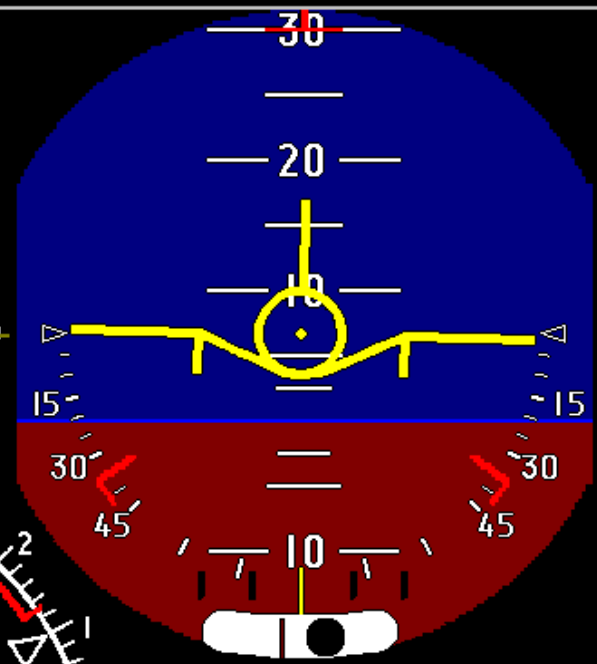
БКО

Р/С

ОПС

★

Н
7
6
0



ПЗ 9 ППМІ Д 91 КМ
Аз 313° ↑

И
М
С
И
Н
П
У
И
Н
С
У
Б
В

А
П
Р

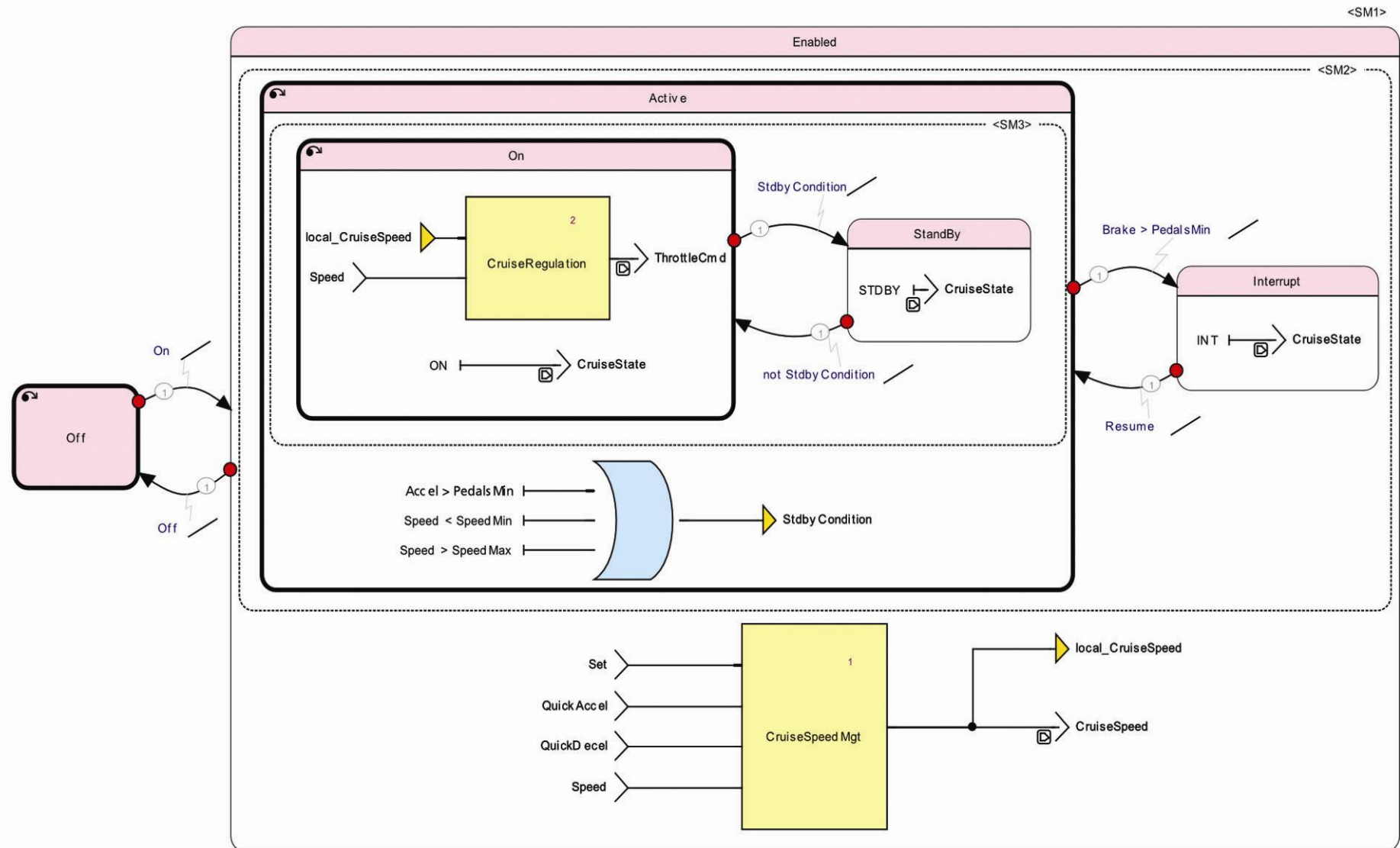
Т
А
Н
Г

ПУ

Средства проектирования алгоритмов

- Алгоритмы бортового ПО хорошо формализуются
 - Поточковая обработка данных
 - Конечный автомат
- Проектирование/описание на формальном уровне позволяет формализовать проверку алгоритма
- Необходимая функциональность:
 - Поддержка обоих видов формального представления
 - Графическое описание
 - Тестирование и пошаговая отладка на уровне модели
 - Верификация на основе формальных методов
 - Сертифицированный кодогенератор
- Примеры: Telelogic Rhapsody, SCADE Suite, Simulink

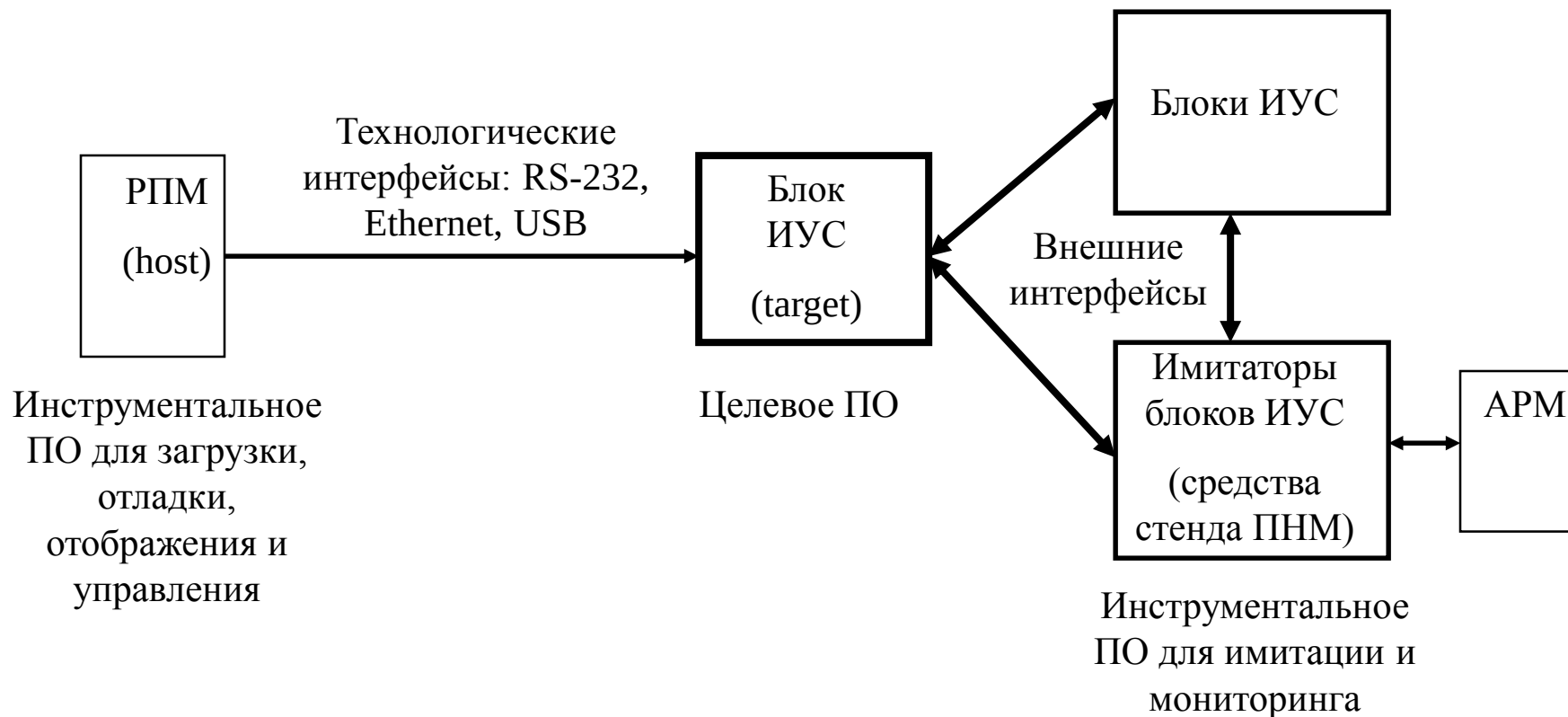
Диаграмма SCADE



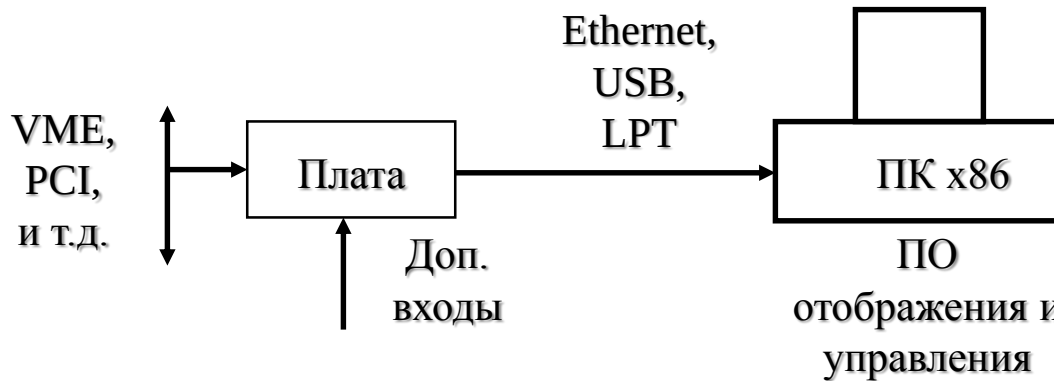
Технологический комплекс разработки программ

- Поддержка целевой ОС и аппаратной архитектуры
- Поддержка редактирования/компиляции/компоновки программ
- Поддержка отладки
 - В среде инструментального ПК
 - В среде эмулятора целевой системы
 - На целевой системе
- Поддержка отладки в реальном времени
 - Мониторинг внешних каналов связи
 - Мониторинг внутренних данных программы
 - Мониторинг системных шин БЦВМ

Отладка ПО ИУС РВ на реальном блоке

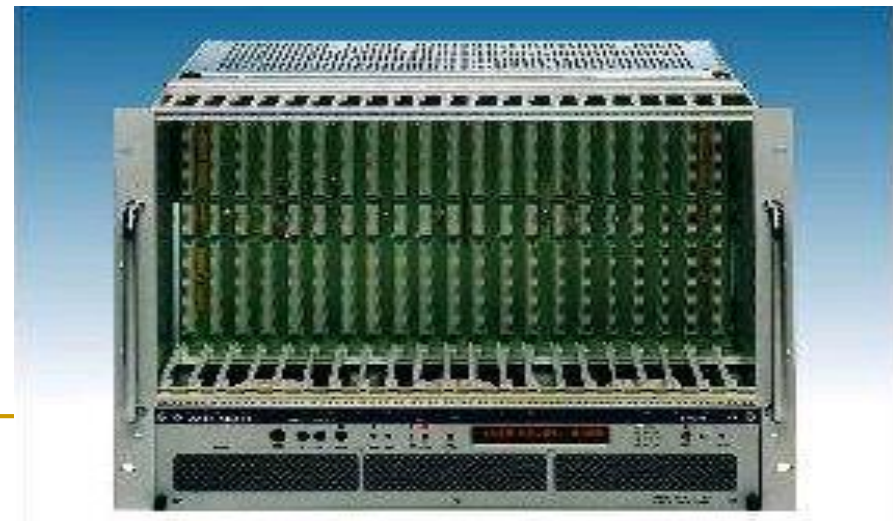


Анализаторы шин VME/PCI

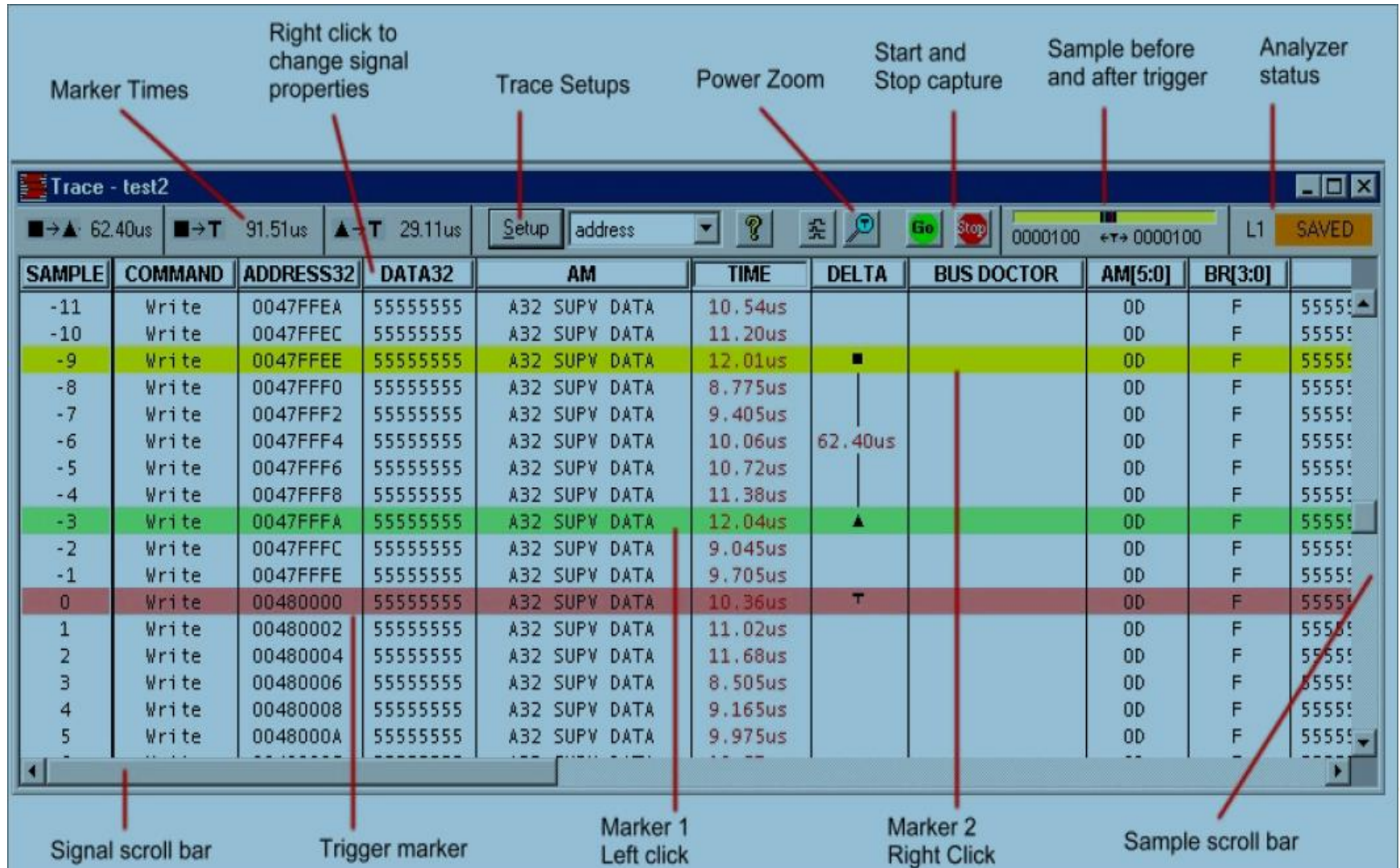


Инструментальные средства анализа:

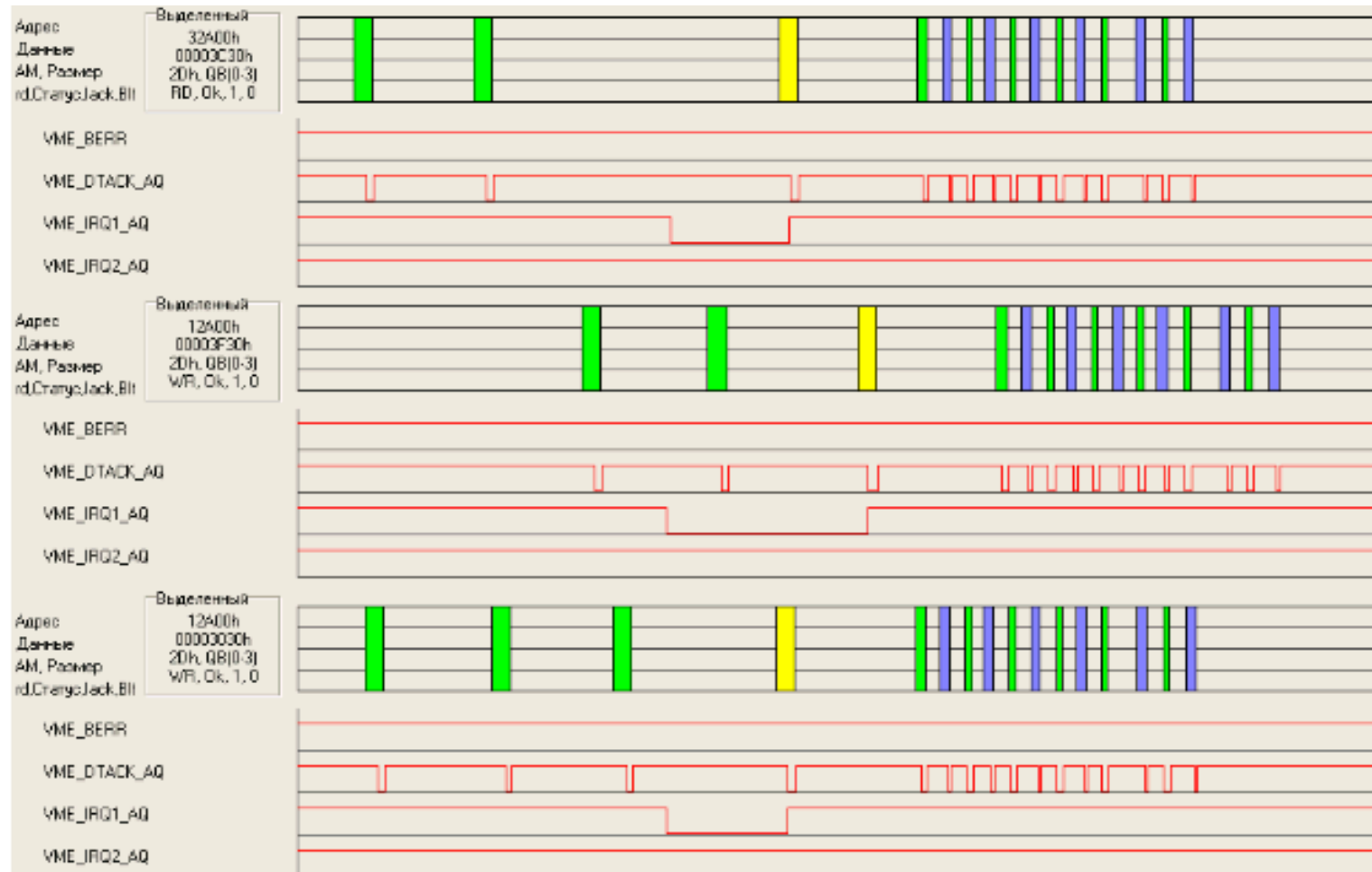
- Silicon Control Inc
- Curtiss-Wright Electronics systems / VMETRO
- LeCroy Inc
- Tektronix Inc
- Гранит-BT



Анализатор шины VME: таблица обменов



Анализатор шины VME: временная диаграмма



Анализатор канала MIL STD-1553B

Текущий сеанс: "2012-05-05 14:31" <@standvm-lenny>

5 мая 2012 Время: 11:01.817125 - 12:01.817141 Длительность: 01:00.000016 Ч:М

Каналы

Имя	Тип	ОН	P	B	ОУ
mil	МКИО				
mil1	МКИО				
mil2	МКИО				

Свойство Филтр

Имя	значение
Циклограмма	По умолчанию
Имя	mil
Тип	M1553
МК	☒
Размещение	hamster:2
Филтр	По умолчанию
КК	☒
Размещение	hamster:2
ОУ	☒
Размещение	hamster:2
Филтр	По умолчанию

Добавить МК Добавить КК

Добавить ОУ Удалить

[МКИО] mils (180) Интервал: 15:07:20.817125 - 15:08:20.817141 Обмен Адрес ОУ Per Статистика КК

№	Время	Дата	Δt	Формат	Отправитель	Получатель	Размер
4600	15:07:53.815...	5 Май 2012	0:00:00.9991...	1 (КК->ОУ)	КК	0x01:0x01	32
4601	15:07:53.816...	5 Май 2012	0:00:00.0000...	2 (ОУ->КК)	0x01:0x01	КК	0
4602	15:07:53.816...	5 Май 2012	0:00:00.0000...	3 (ОУ->ОУ)	0x00:0x01	0x01:0x01	0
Пауза: 0:00:00.999144							
4603	15:07:54.815...	5 Май 2012	0:00:00.9991...	1 (КК->ОУ)	КК	0x01:0x01	32
4604	15:07:54.816...	5 Май 2012	0:00:00.0000...	2 (ОУ->КК)	0x01:0x01	КК	0
4605	15:07:54.816...	5 Май 2012	0:00:00.0000...	3 (ОУ->ОУ)	0x00:0x01	0x01:0x01	0
Пауза: 0:00:00.999119							
4606	15:07:55.815...	5 Май 2012	0:00:00.9991...	1 (КК->ОУ)	КК	0x01:0x01	32
4607	15:07:55.816...	5 Май 2012	0:00:00.0000...	2 (ОУ->КК)	0x01:0x01	КК	32
4608	15:07:55.817...	5 Май 2012	0:00:00.0000...	3 (ОУ->ОУ)	0x00:0x01	0x01:0x01	32
Пауза: 0:00:00.997816							
4609	15:07:56.815...	5 Май 2012	0:00:00.9978...	1 (КК->ОУ)	КК	0x01:0x01	32
4610	15:07:56.816...	5 Май 2012	0:00:00.0000...	2 (ОУ->КК)	0x01:0x01	КК	32
4611	15:07:56.817...	5 Май 2012	0:00:00.0000...	3 (ОУ->ОУ)	0x00:0x01	0x01:0x01	32

[МКИО] mils (181) 0.28% Интервал: 15:11:01.817125 - 15:12:01.817141 Обмен Адрес ОУ Per Статистика КК

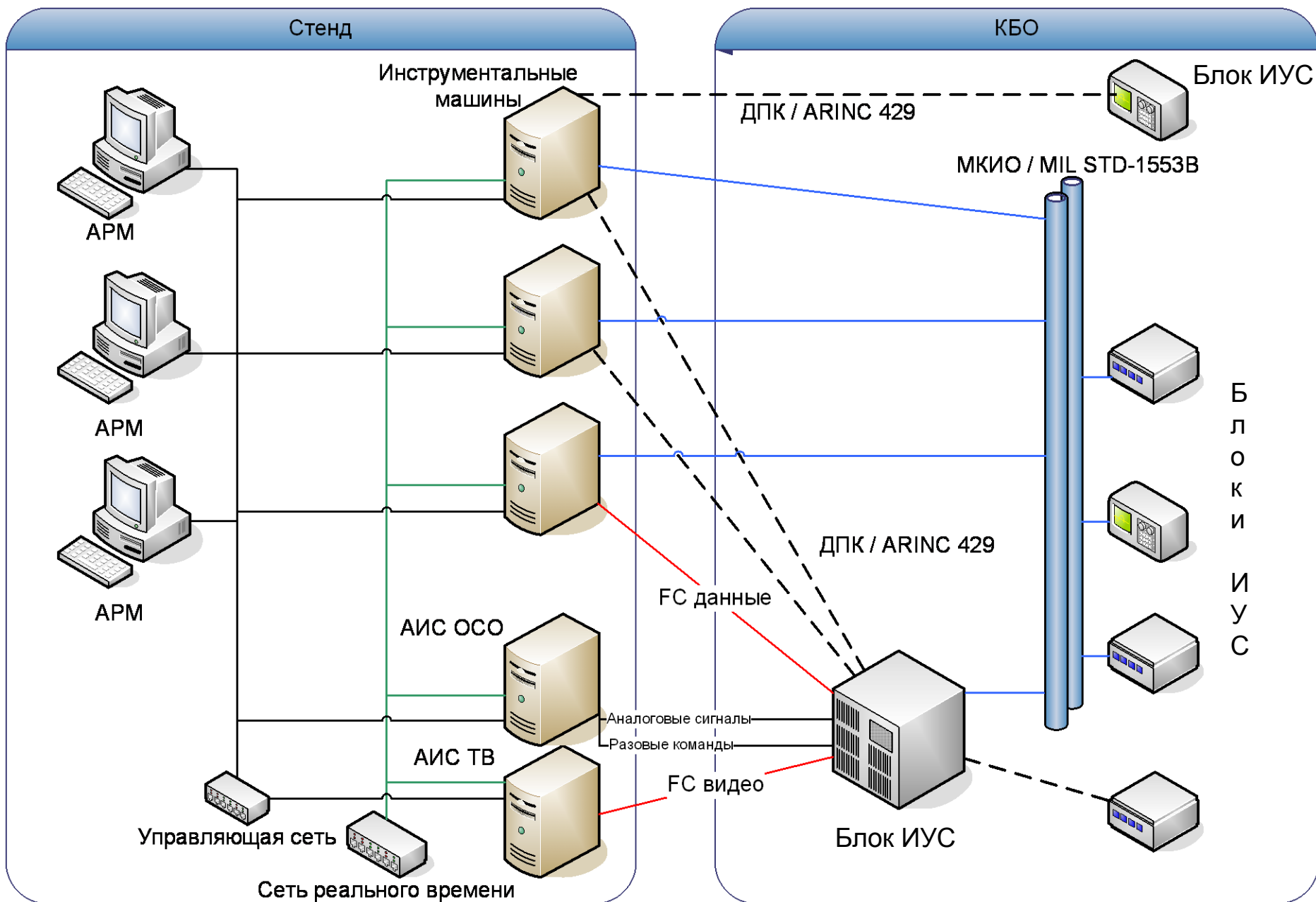
№	Время	Дата	Δt	Формат	Отправитель	Получатель
5166	15:11:01.817161	5 Май 2012	0:00:00.000000	3 (ОУ->ОУ)	0x00:0x01	0x01:0x01
Пауза: 0:00:00.997801						
5167	15:11:02.815692	5 Май 2012	0:00:00.997801	1 (КК->ОУ)	КК	0x01:0x01
5168	15:11:02.816413	5 Май 2012	0:00:00.000036	2 (ОУ->КК)	0x01:0x01	КК
5169	15:11:02.817140	5 Май 2012	0:00:00.000042	3 (ОУ->ОУ)	0x00:0x01	0x01:0x01
Пауза: 0:00:00.997821						
5170	15:11:03.815691	5 Май 2012	0:00:00.997821	1 (КК->ОУ)	КК	0x01:0x01
5171	15:11:03.816413	5 Май 2012	0:00:00.000037	2 (ОУ->КК)	0x01:0x01	КК
5172	15:11:03.817140	5 Май 2012	0:00:00.000042	3 (ОУ->ОУ)	0x00:0x01	0x01:0x01
Пауза: 0:00:00.997822						
5173	15:11:04.815692	5 Май 2012	0:00:00.997822	1 (КК->ОУ)	КК	0x01:0x01
5174	15:11:04.816413	5 Май 2012	0:00:00.000036	2 (ОУ->КК)	0x01:0x01	КК
5175	15:11:04.817151	5 Май 2012	0:00:00.000053	3 (ОУ->ОУ)	0x00:0x01	0x01:0x01
Пауза: 0:00:00.997811						
5176	15:11:05.815692	5 Май 2012	0:00:00.997811	1 (КК->ОУ)	КК	0x01:0x01

14:27:26: Сеанс "Default" успешно загружен.

Средства поддержки верификации и тестирования бортового ПО

- Тестирование на целевой платформе
 - Недопустимость инструментирования
 - Тестирование через каналы бортовых интерфейсов
 - Тестирование требований реального времени
 - Интерактивное тестирование индикационных форматов
- Многоэтапное тестирование
 - Сопровождение интеграции подсистем КБО
- Необходимая функциональность:
 - Поддержка стандартов бортовых интерфейсов
 - Многомашинные конфигурации
 - Выполнение тестов в реальном времени
 - Автоматическое и интерактивное тестирование
 - Пакетный режим
 - Формирование отчётов, прослеживаемость требований
- Примеры средств: Rational Test RealTime, VectorCast, средства разработки ЛВК

Архитектура стенда тестирования ИУС

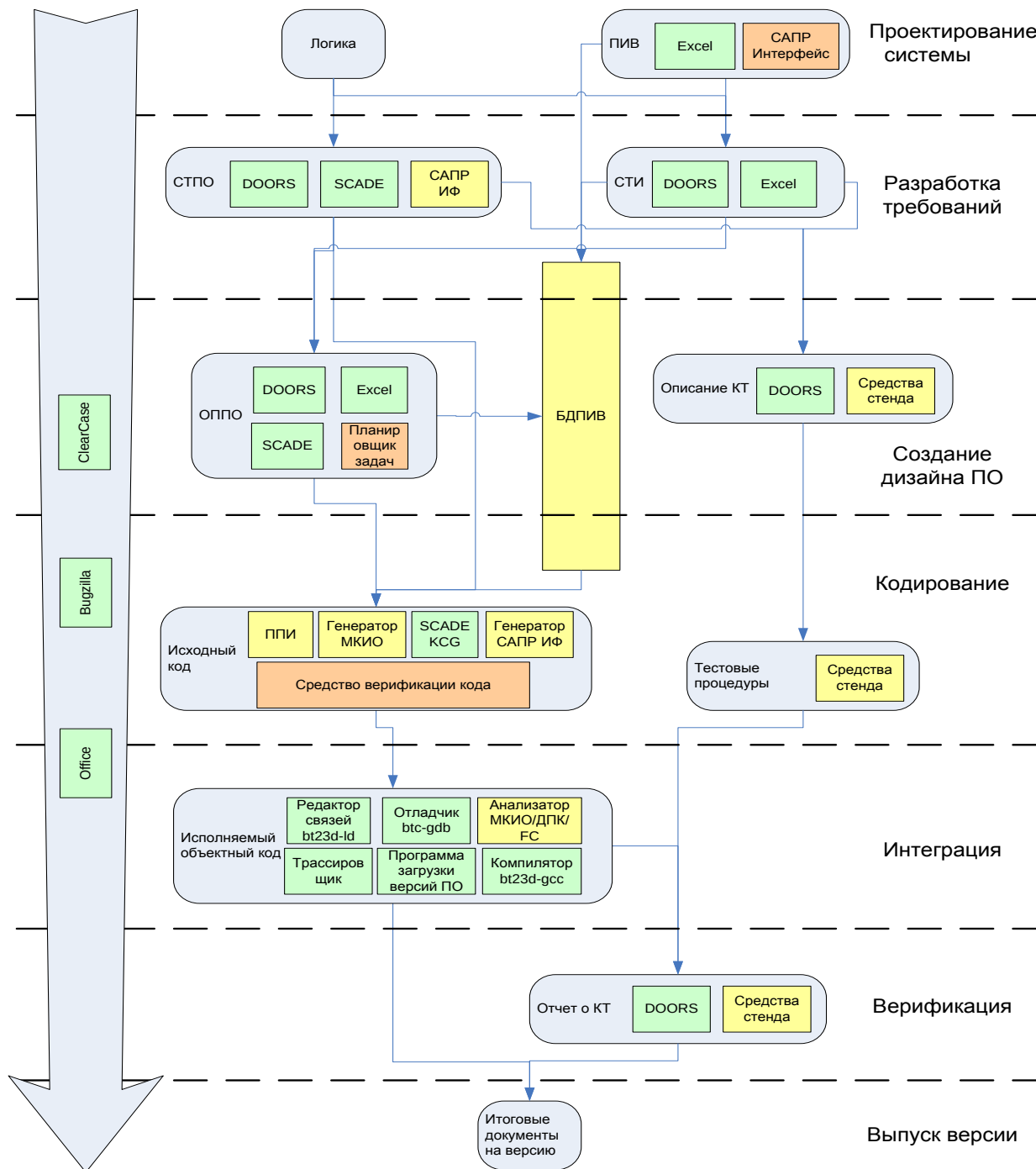


Технологическая цепочка средств разработки бортового ПО

Принцип построения технологической цепочки

- Сквозная поддержка ЖЦ, включая активности на всех фазах
- Сопряжение «вход-выход» с обеспечением совместимости форматов данных
- Особое внимание на переходы между фазами
 - Требуется фиксация выходных артефактов

Цепочка средств разработки бортового ПО



Хорошие новости

- ПО ИУС РВ требует систематического подхода к проектированию, разработке и тестированию
- Сложность и критичность ПО ИУС РВ определяет организационную и технологическую сложность процесса его разработки
- Выполнено организационное структурирование жизненного цикла ПО ИУС РВ по фазам, процессам, активностям
- Построена и внедрена технологическая цепочка средств разработки ПО ИУС РВ, применяемая в соответствии со структурой ЖЦ
- Разработана и внедрена линейка стендов отработки ПО ИУС РВ

Спасибо за внимание

Балашов В.В.
к.ф.-м.н., н.с.
hbd@cs.msu.su