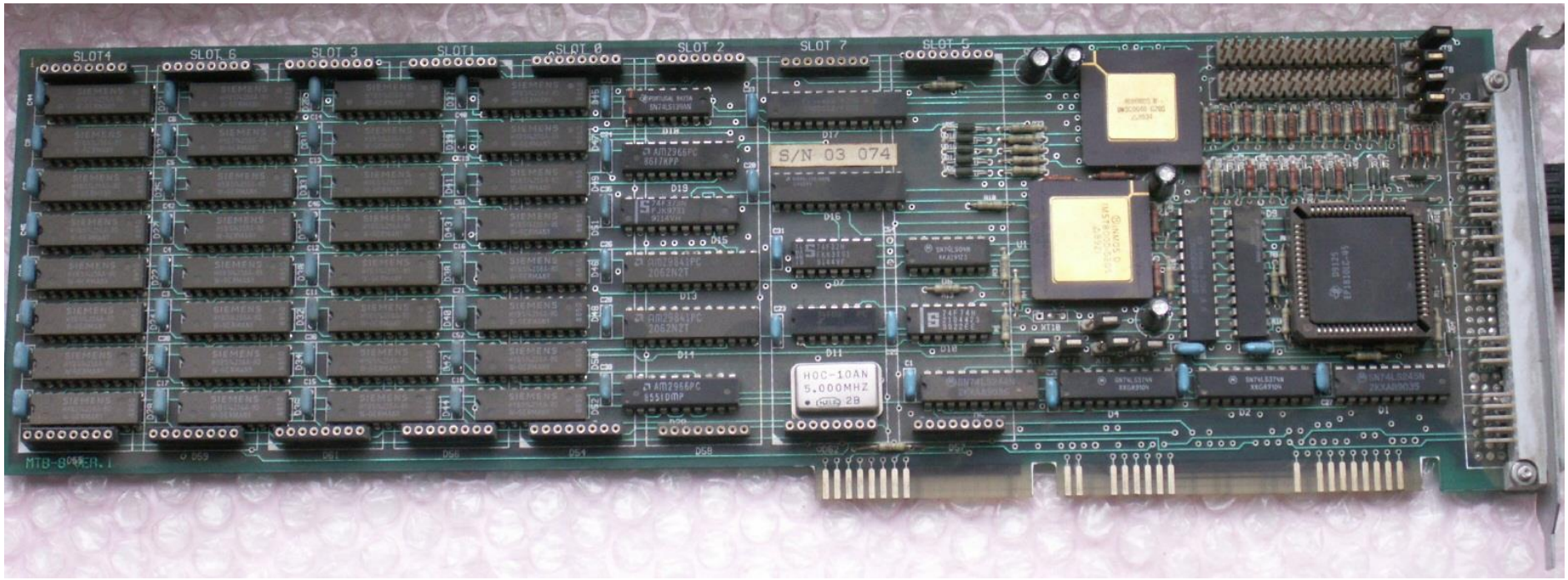
Параллельные высокопроизводительные вычисления

Якобовский М.В., проф., д.ф.-м.н.
СКИ МГУ,
Институт прикладной математики
им. М.В.Келдыша РАН, Москва

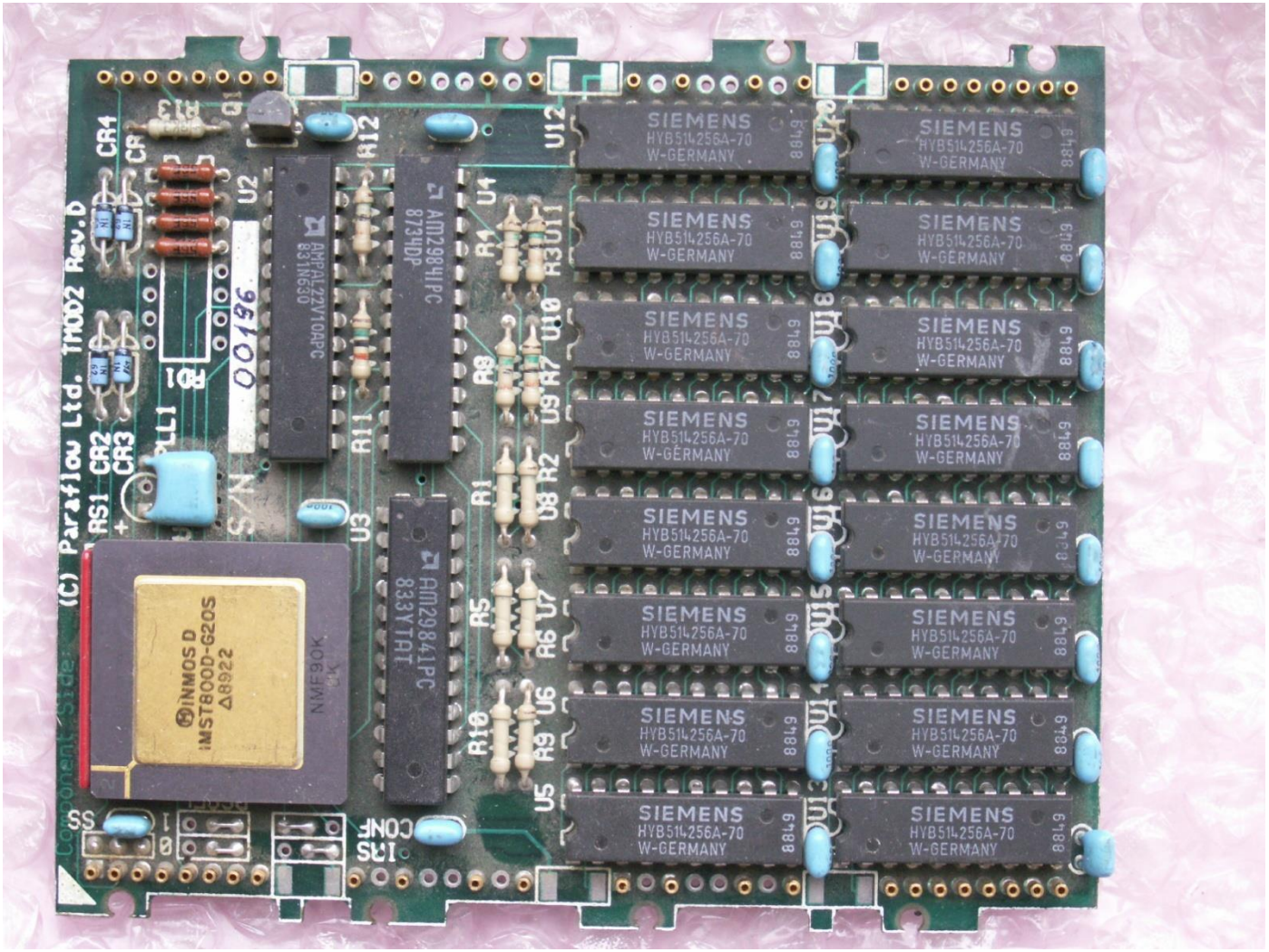
Содержание лекции

- Многопроцессорные системы
  - с распределенной памятью
  - с общей памятью
  - Гибридные
- Модели выполнения программ
- Свойства канала передачи данных
- Методы взаимодействия процессов
  - Методы передачи данных
  - Семафоры
- Ускорение и эффективность параллельных алгоритмов
- Пример алгоритма с низкой эффективностью, но высоким быстродействием

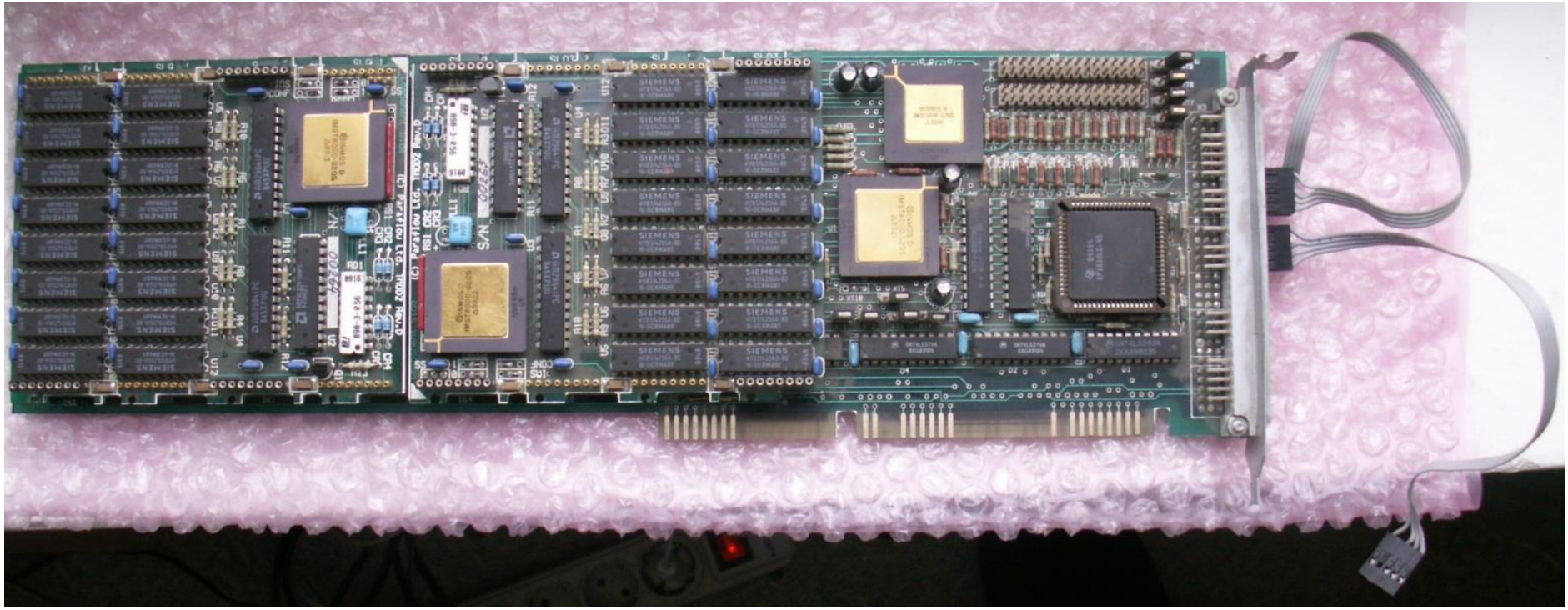
Транспьютерная материнская плата МТБ-8



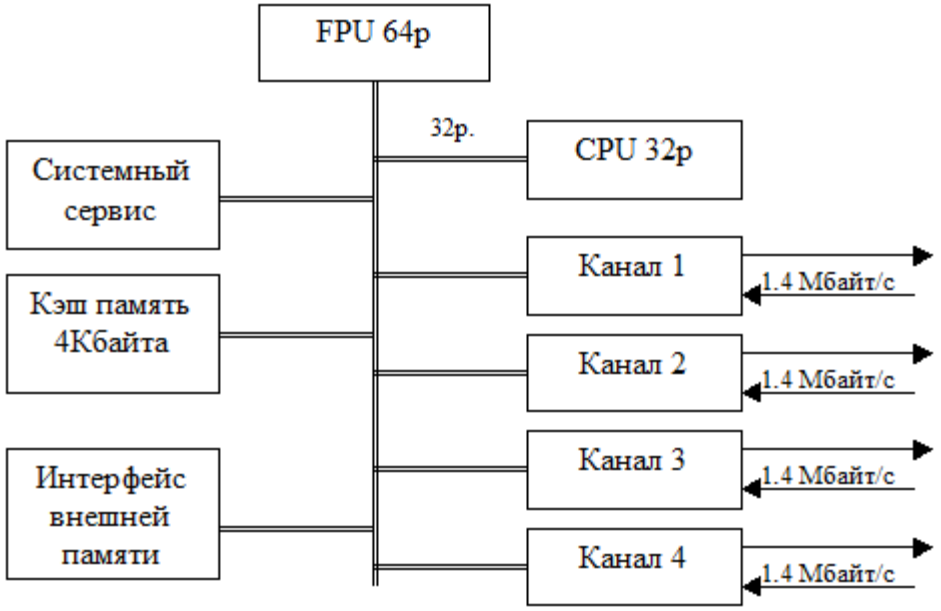
Транспьютер и оперативная память



Три транспьютера на плате МТБ-8

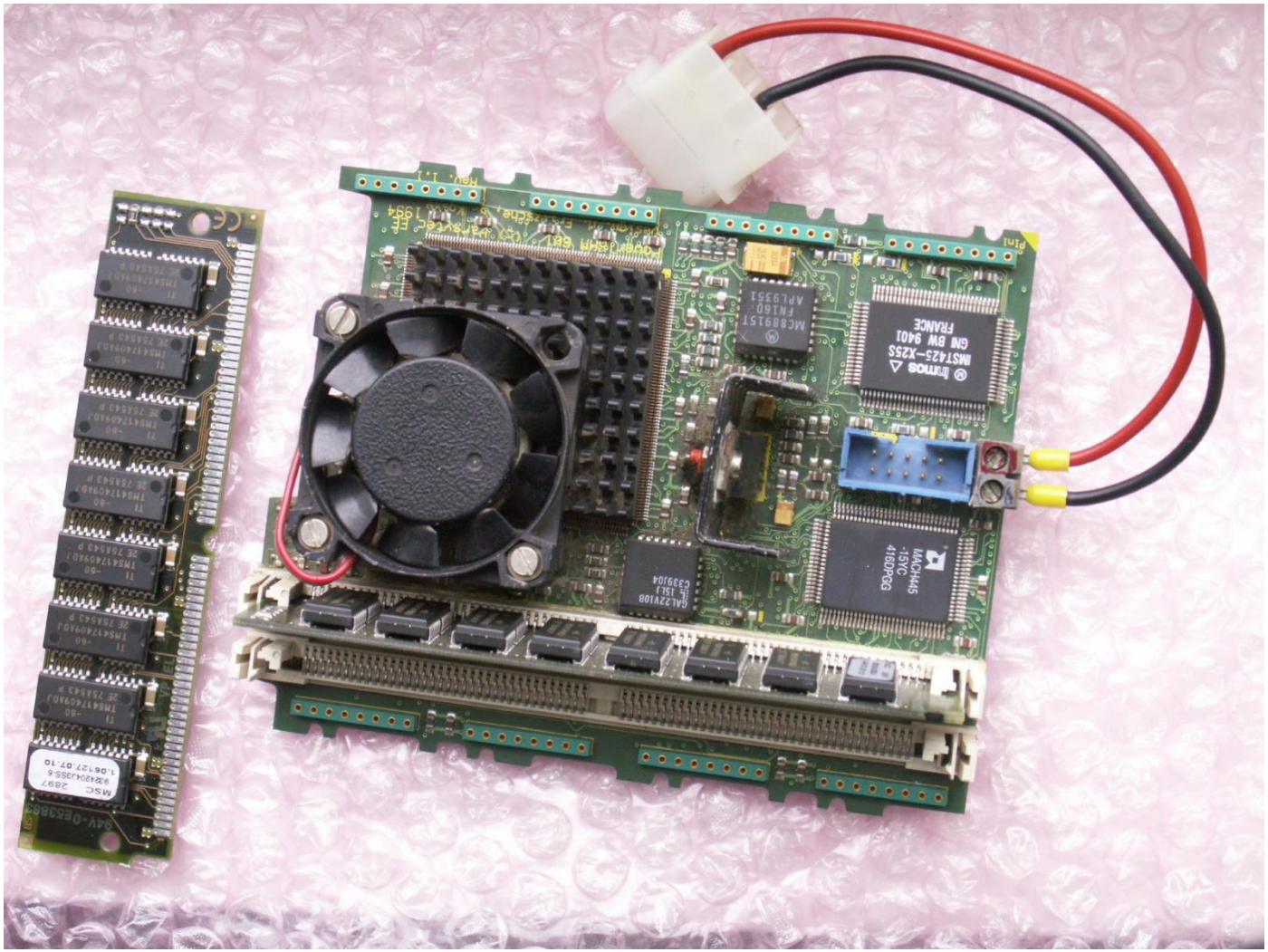


Структура транспьютера Т-800

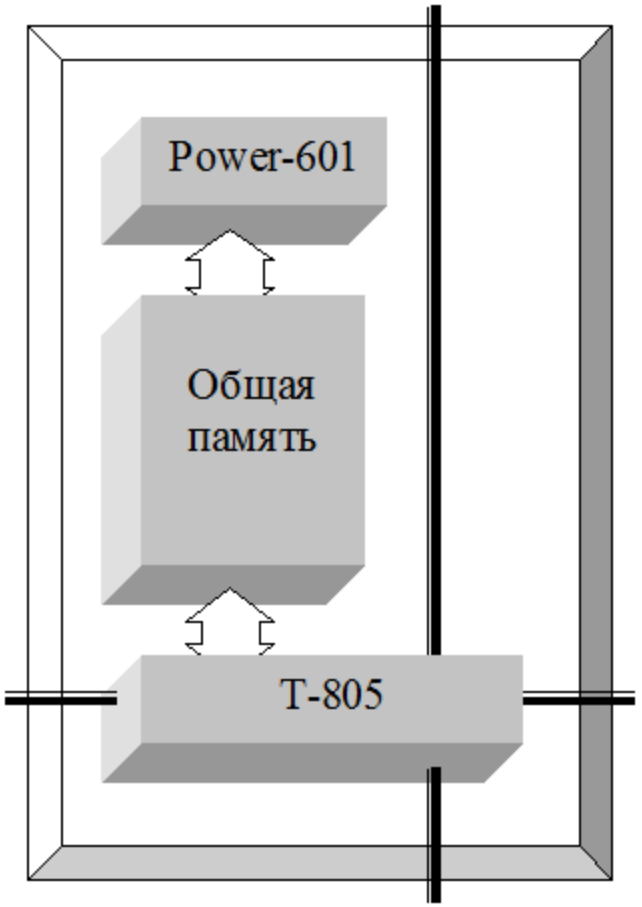


Структура транспьютера Т-800

Узел с общей памятью – транспьютер и процессор общего назначения

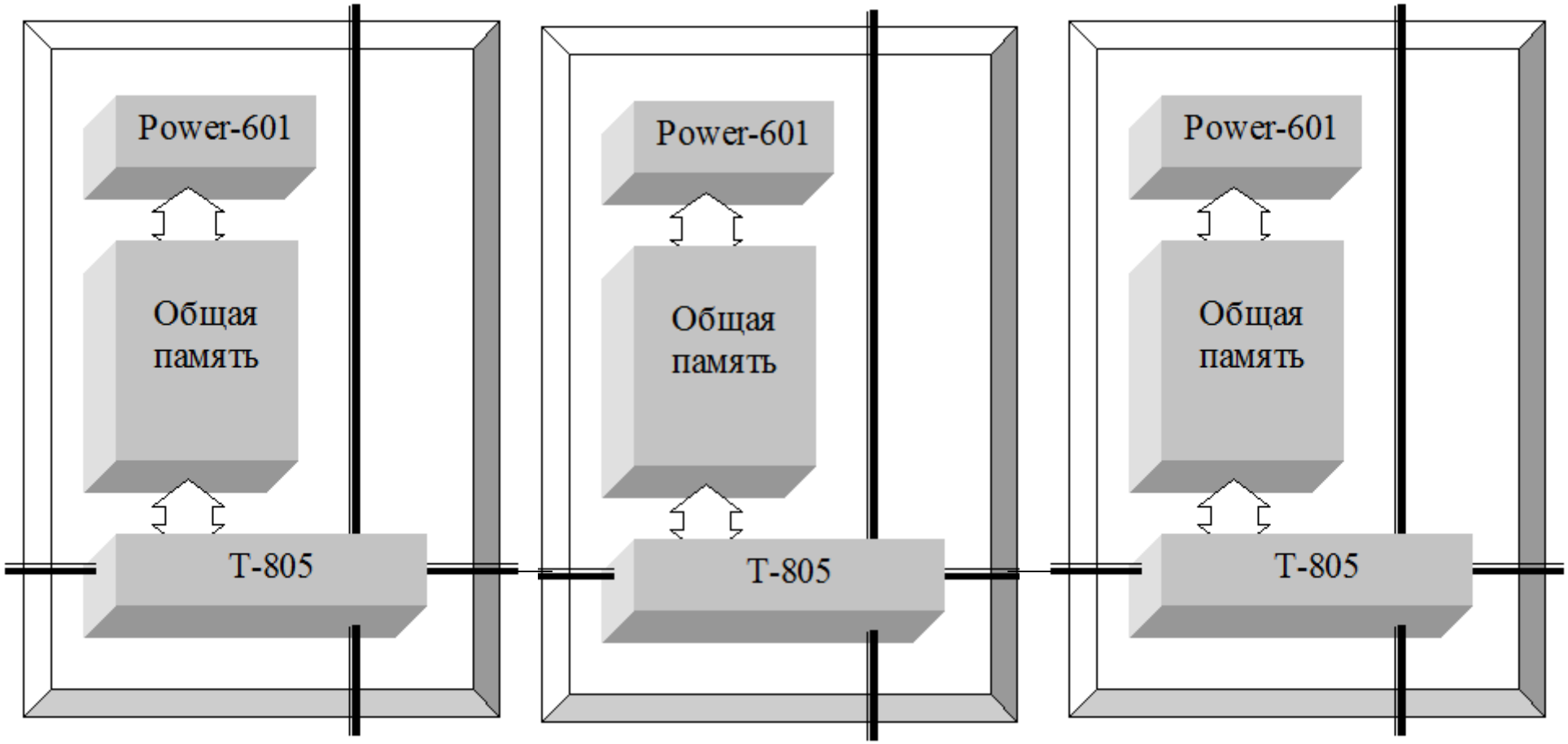


Узел PowerXplorer



Структура узла PowerXplorer

Гибридная система



Вопрос

Зачем нужны транспьютерные линки?

a. Для ввода-вывода данных
b. Для подвода питания к транспьютерам
c. Для передачи данных между транспьютерами

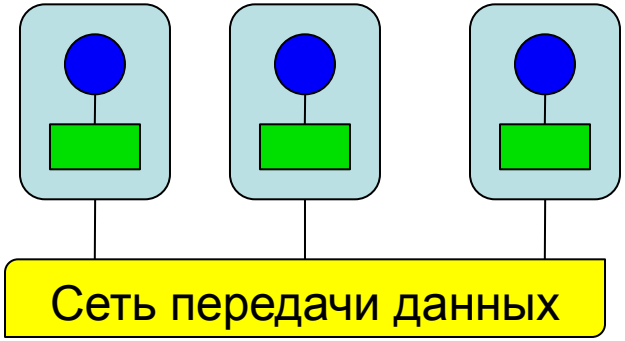
Вопрос

Можно ли на основе транспьютеров делать системы с общей памятью?

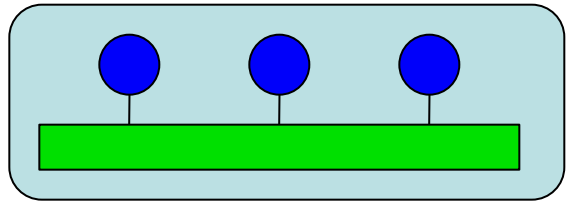
a. Можно делать
b. Нельзя делать
c. Только такие системы и можно делать

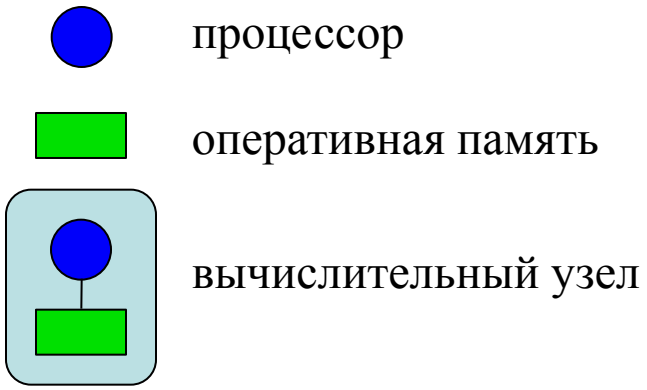
Уточнение круга рассматриваемых систем

- Системы на основе объединенных сетью типовых вычислительных узлов – системы с распределенной оперативной памятью



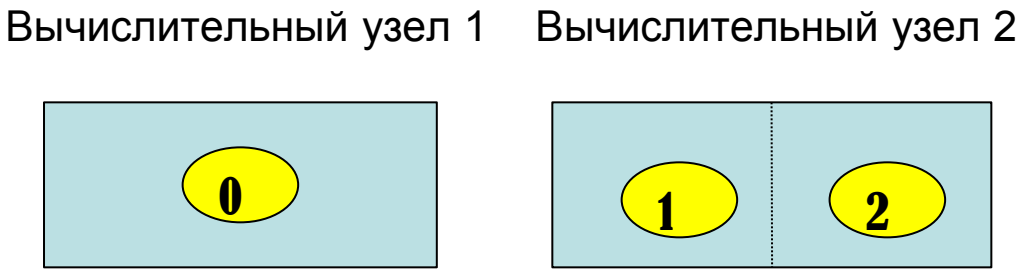
- Системы с доступом всех процессоров к общей оперативной памяти





Модель выполнения программы на распределенной памяти

- При запуске указывается число требуемых процессоров Np и название программы
- На выделенных для расчета узлах запускается Np копий программы
  - Например, на двух узлах запущены три копии программы. Копия программы с номером 1 не имеет непосредственного доступа к оперативной памяти копий 0 и 2:



- В каждой копии программы известны значения двух переменных
  - Np – одинаковое во всех копиях – число копий
  - rank из диапазона [0 … Np-1] – уникальный номер копии
- Любые две копии программы могут непосредственно обмениваться данными с помощью функций передачи сообщений Send/Recv

Синхронные обмены данными

A=3

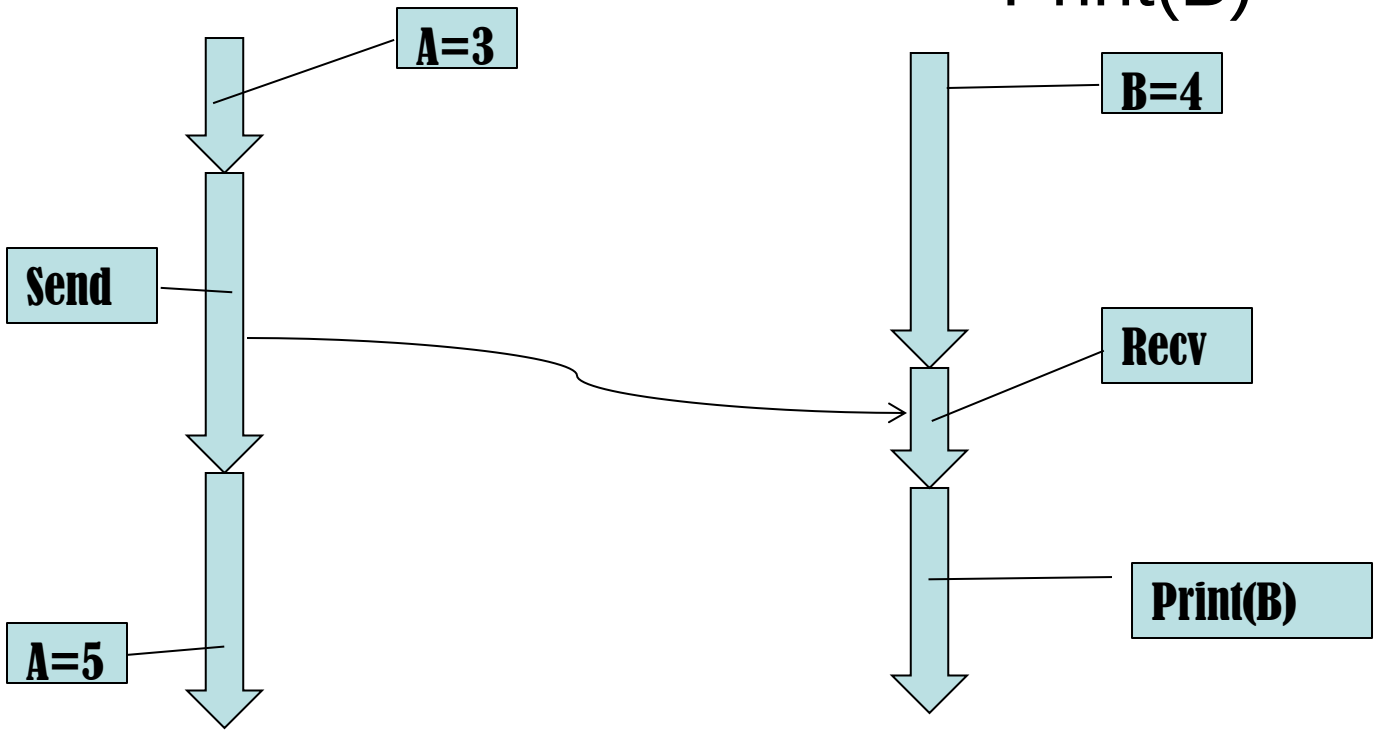
Send(&A)

A=5

B=4

Recv(&B)

Print(B)

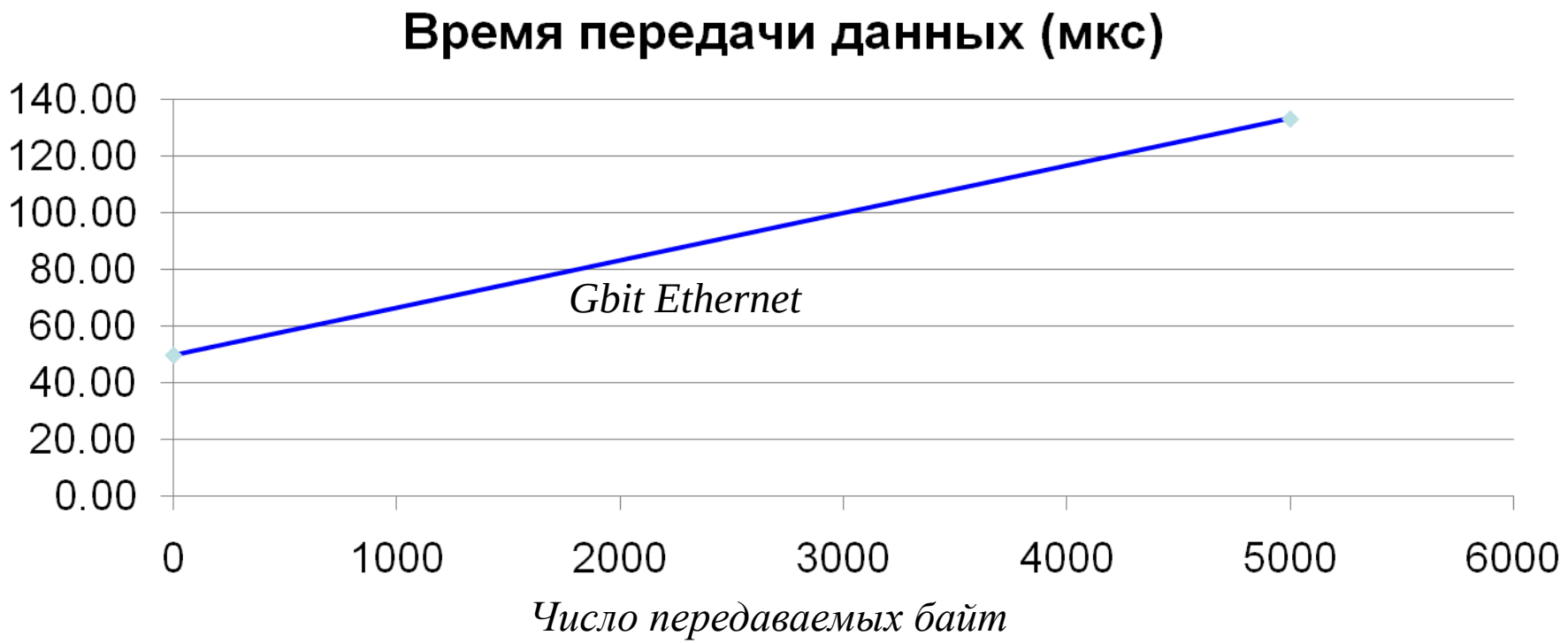


Результат

3

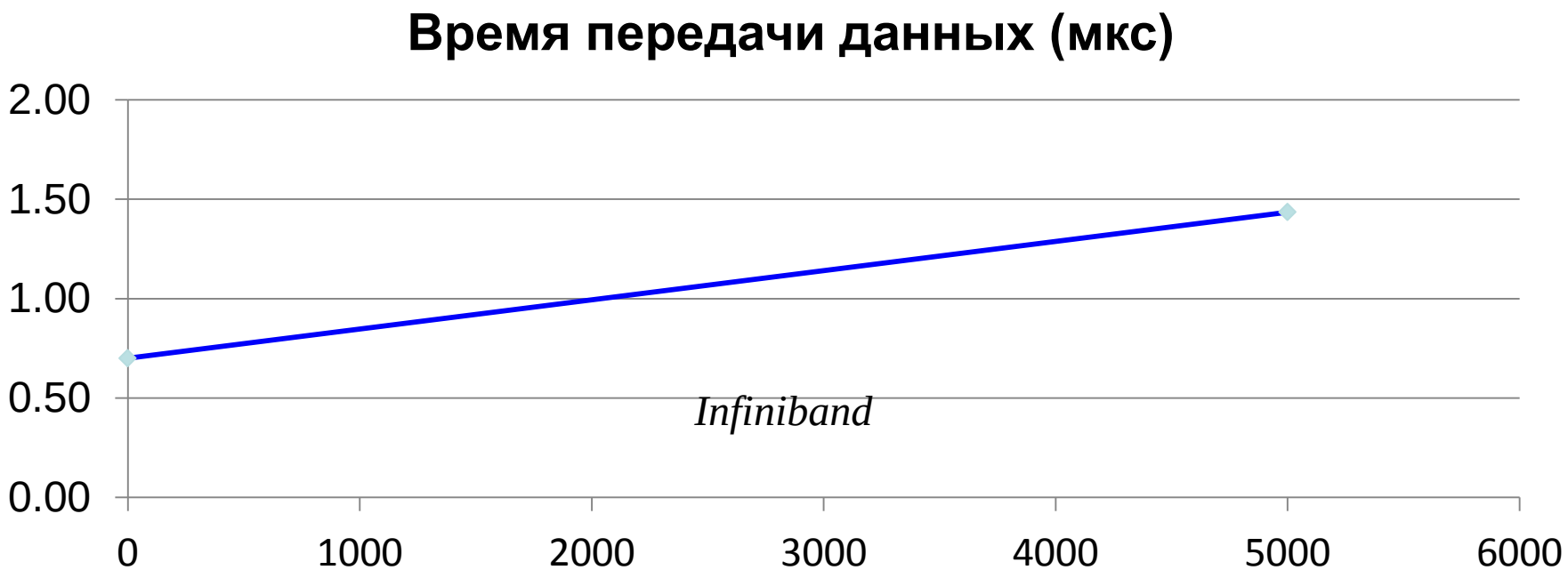
Свойства канала передачи данных

$$T(n) = n * T_{\text{передачи байта}} + T_{\text{латентности}}$$



Свойства канала передачи данных

$$T(n) = n * T_{\text{передачи байта}} + T_{\text{латентности}}$$



Число передаваемых байт

6.8 Гбайт/с

За 0.7 мкс передаётся 5 Кбайт

Вопрос

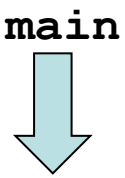
При каком способе передачи массива чисел будет затрачено меньше времени?

a. Передача каждого числа отдельным сообщением
b. Объединение всех чисел в единый массив и передача массива одним сообщением
c. Не имеет значения

Модель выполнения программы на общей памяти

- Работа начинается с запуска одного экземпляра программы
- При необходимости программа порождает новые процессы, каждый из которых:
  - Обладает собственными локальными переменными
  - Имеет доступ к глобальным переменным

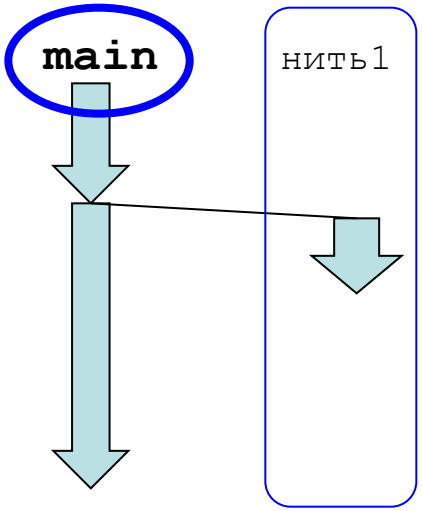
```
int a_global;
main()
{
int b1_local;
Запуск нити(fun())
}
fun()
{
int b2_local;
Запуск нити(…)
}
```



Модель выполнения программы на общей памяти

- Работа начинается с запуска одного экземпляра программы
- При необходимости программа порождает новые процессы, каждый из которых:
  - Обладает собственными локальными переменными
  - Имеет доступ к глобальным переменным

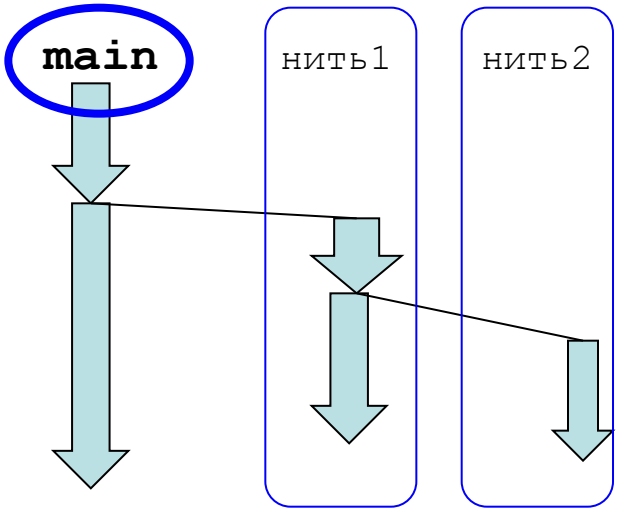
```
int a_global;
main()
{
int b1_local;
Запуск нити(fun())
}
fun()
{
int b2_local;
Запуск нити(…)
}
```



Модель выполнения программы на общей памяти

- Работа начинается с запуска одного экземпляра программы
- При необходимости программа порождает новые процессы, каждый из которых:
  - Обладает собственными локальными переменными
  - Имеет доступ к глобальным переменным

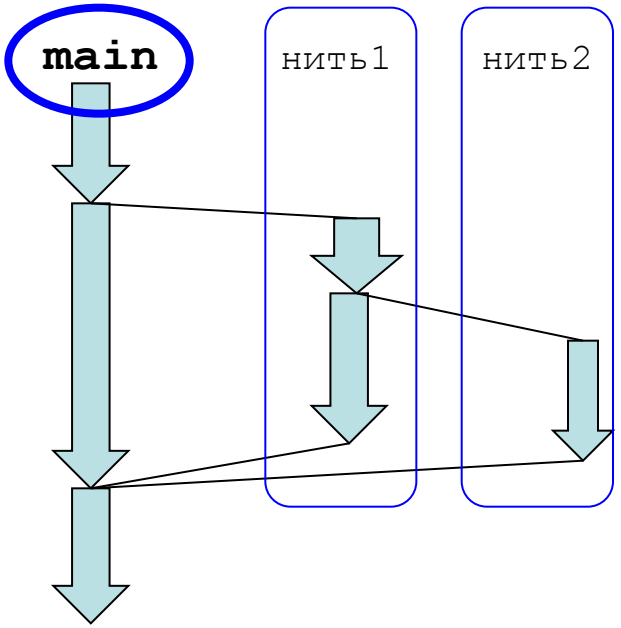
```
int a_global;
main()
{
int b1_local;
Запуск нити(fun())
}
fun()
{
int b2_local;
Запуск нити(…)
}
```



Модель выполнения программы на общей памяти

- Работа начинается с запуска одного экземпляра программы
- При необходимости программа порождает новые процессы, каждый из которых:
  - Обладает собственными локальными переменными
  - Имеет доступ к глобальным переменным

```
int a_global;
main()
{
int b1_local;
Запуск нити(fun())
}
fun()
{
int b2_local;
Запуск нити(…)
}
```



Что будет напечатано?

```
int a=1;
```

```
Нить1
{
   a=a+2
}
```

```
Нить2
{
   a=a+3
}
```

```
Нить3
{
   print(a)
}
```

a=?

Что будет напечатано?

```
int a=1;
```

```
Нить1
{
   a=a+2
}
```

```
Нить2
{
   a=a+3
}
```

```
Нить3
{
   print(a)
}
```

6

Что будет напечатано?

```
int a=1;
```

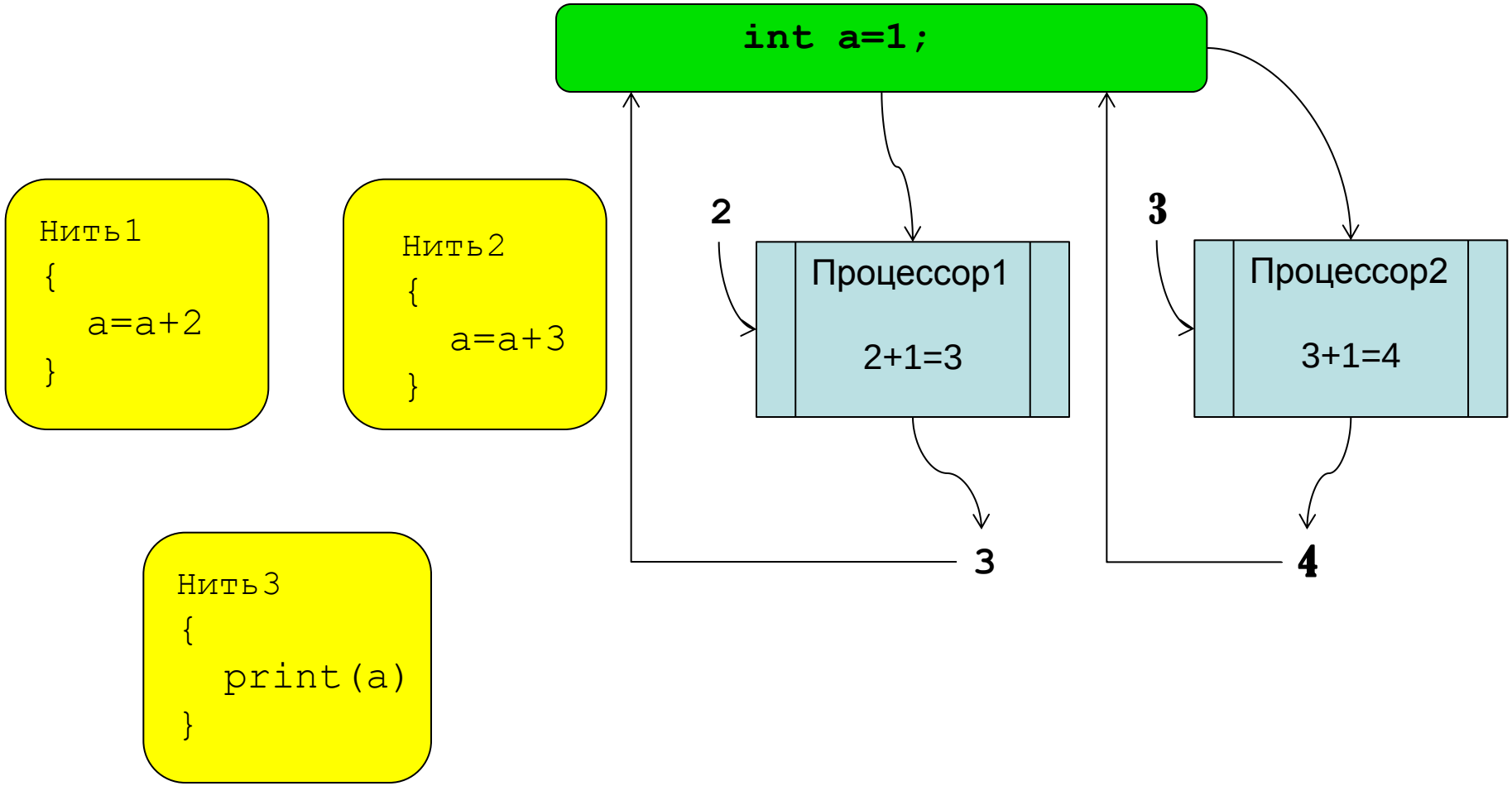
```
Нить1
{
   a=a+2
}
```

```
Нить3
{
   print(a)
}
```

```
Нить2
{
   a=a+3
}
```

3

Что будет напечатано?



6 ? 4 ? 3

Что будет напечатано?

```
int a=1;
```

```
Нить1
{
   a=a+2
}
```

```
Нить2
{
   a=a+3
}
```

```
Нить3
{
   print(a)
}
```

6 ? 4 ? 3 ? ___

Семафор

- Целочисленная неотрицательная переменная
- Инициализация и две атомарные операции

- Операция V(S)
  - Увеличивает значение S на 1

- Операция P(S)
  - Если S положительно, то уменьшает S на 1
  - Иначе ждет, пока S не станет больше 0

Языки программирования. Редактор Ф.Женюи. Перевод с англ. В.П.Кузнецова. Под ред. В.М.Курочкина. М:."Мир", 1972 Э. Дейкстра. Взаимодействие последовательных процессов.
http://khpi-iip.mipk.kharkiv.edu/library/extent/dijkstra/ewd123/index.html

Что будет напечатано?

```
int a=0;
Sem S=1, S1=0, S2=0;
```

```
Нить1
{
   P(S)
   a=a+2
   V(S)
   V(S1)
}
```

```
Нить2
{
   P(S)
   a=a+3
   V(S)
   V(S2)
}
```

```
Нить3
{
 P(S1)
 P(S2)
 print(a)
}
```

6

Ускорение и эффективность параллельных алгоритмов

ускорение параллельного алгоритма

$$S(p)=T(1)/T(p)$$

эффективность использования вычислительной мощности

$$E(p)=S(p)/p$$

Может ли ускорение превышать число процессоров?

$$S(p)> p$$
$$E(p)>100\%$$

Ускорение и эффективность параллельных алгоритмов

ускорение параллельного алгоритма

$$S(p)=T(1)/T(p)$$

эффективность использования вычислительной мощности

$$E(p)=S(p)/p$$

Может ли ускорение превышать число процессоров?

$$S(p)> p$$
$$E(p)>100\%$$

Да, если учитывать влияние аппаратных особенностей вычислительной системы

Ускорение и эффективность параллельных алгоритмов

ускорение параллельного алгоритма

$$S(p)=T(1)/T(p)$$

эффективность использования вычислительной мощности

$$E(p)=S(p)/p$$

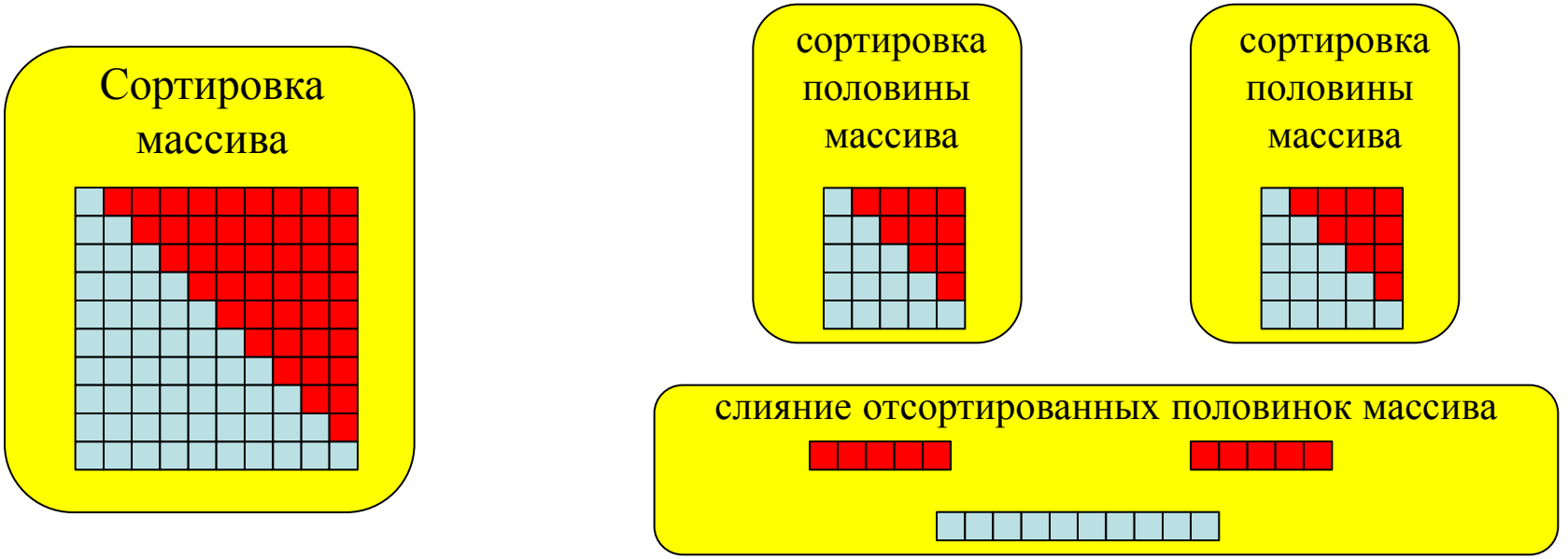
Может ли ускорение превышать число процессоров?

$$S(p)> p$$
$$E(p)>100\%$$

Да, если выбран неудачный последовательный алгоритм

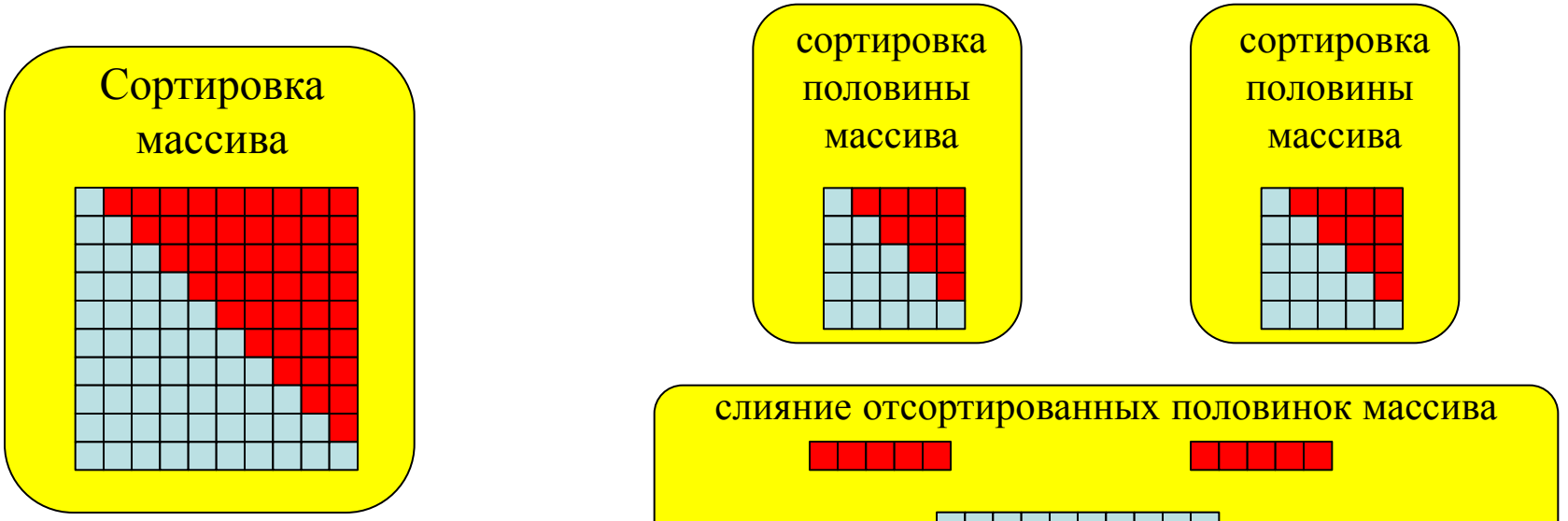
Может ли быть S(p)>p ?

– Пример неудачного последовательного алгоритма



Может ли быть S(p)>p ?

– Пример неудачного последовательного алгоритма

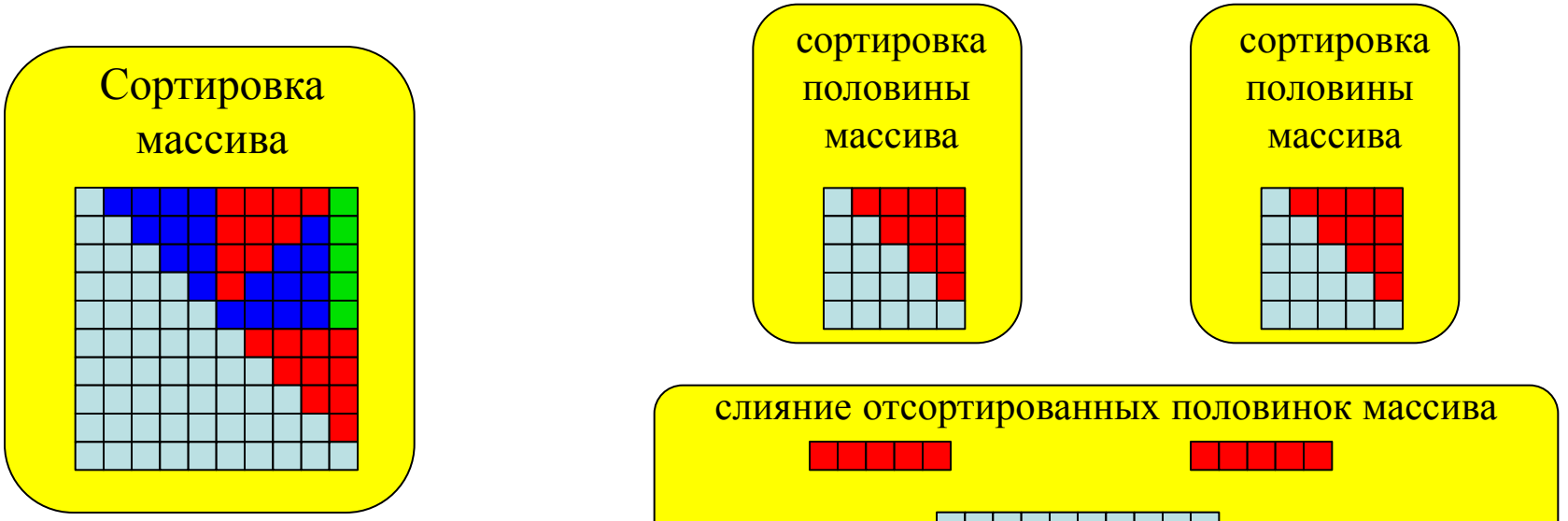


$$\frac{n(n-1)}{2} \sim \frac{n^2}{2}$$

$$\frac{\left(\frac{n}{2}\right)^2}{2} + n \sim \frac{1}{4}\frac{n^2}{2}$$

Может ли быть S(p)>p ?

– В последовательном алгоритме выполняется больше операций, чем в параллельном



$$\frac{n(n-1)}{2} \sim \frac{n^2}{2}$$

$$\frac{\left(\frac{n}{2}\right)^2}{2} + n \sim \frac{1}{4}\frac{n^2}{2}$$

Ускорение и эффективность параллельных алгоритмов

ускорение параллельного алгоритма

$$S(p)=T(1)/T(p)$$

эффективность использования вычислительной мощности

$$E(p)=S(p)/p$$

Ускорение параллельного алгоритма относительно наилучшего последовательного

$$S^*(p)= T^*(1)/T(p)$$

$$E^*(p)= S^*(p)/p$$

Может ли неэффективный алгоритм работать быстрее эффективного?

- Да
  - Если первый алгоритм позволяет использовать больше процессоров, чем второй.

- Самый эффективный алгоритм – использующий один (1) процессор.
  - Его эффективность равна 100%, но он не всегда самый быстрый.

Определить сумму конечного ряда

$$S = \sum_{i=0}^{n} \frac{x^i}{i!}$$

- *n* известно заренее, меняется только *x*
- Все элементарные операции (+ - * /) выполняются за время τ_c
- Все операции выполняются точно, результат не зависит от порядка их выполнения
- Число процессоров не ограничено

Последовательный алгоритм

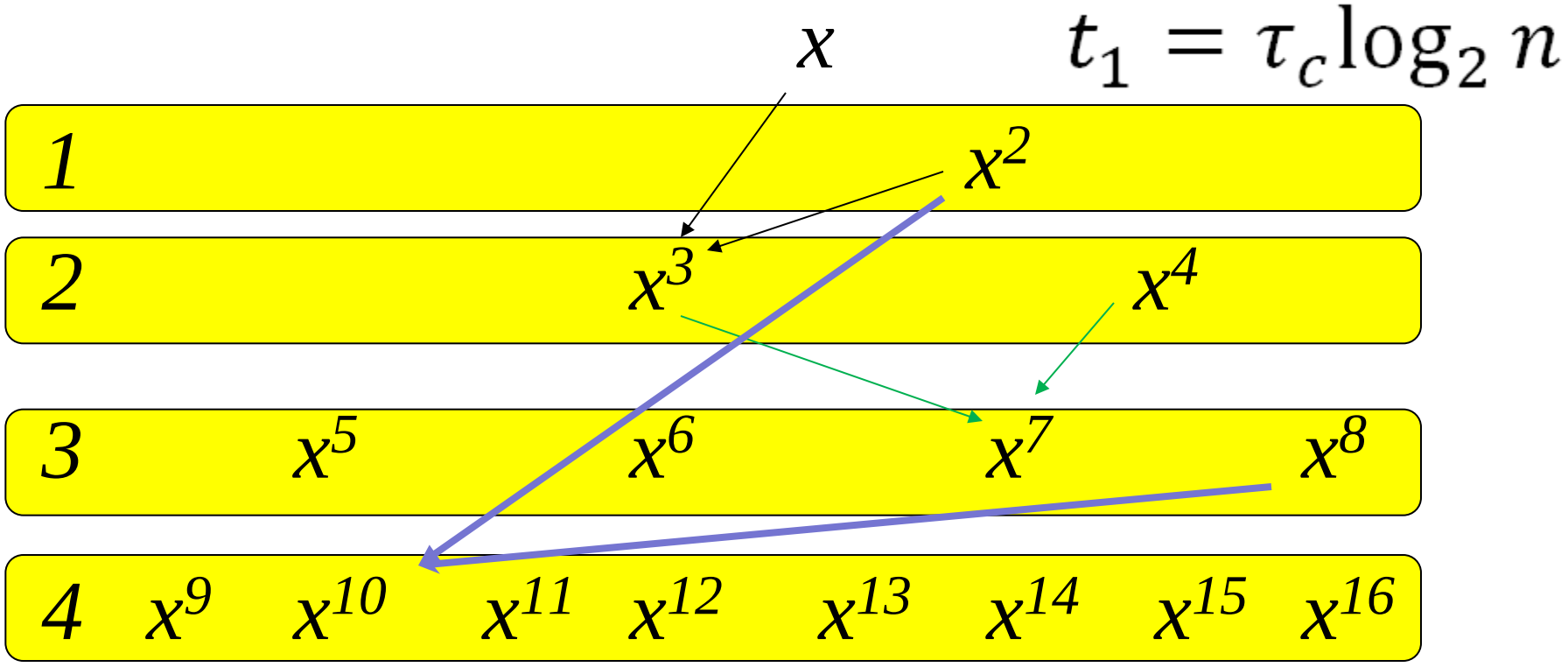
```
S=1;
a=1;
for(i=1;i<= n;i++)
  {
     a=a*x/i;
     S=S+a;
  }
```

$$T_1 = 3n\tau_c$$

Параллельный алгоритм

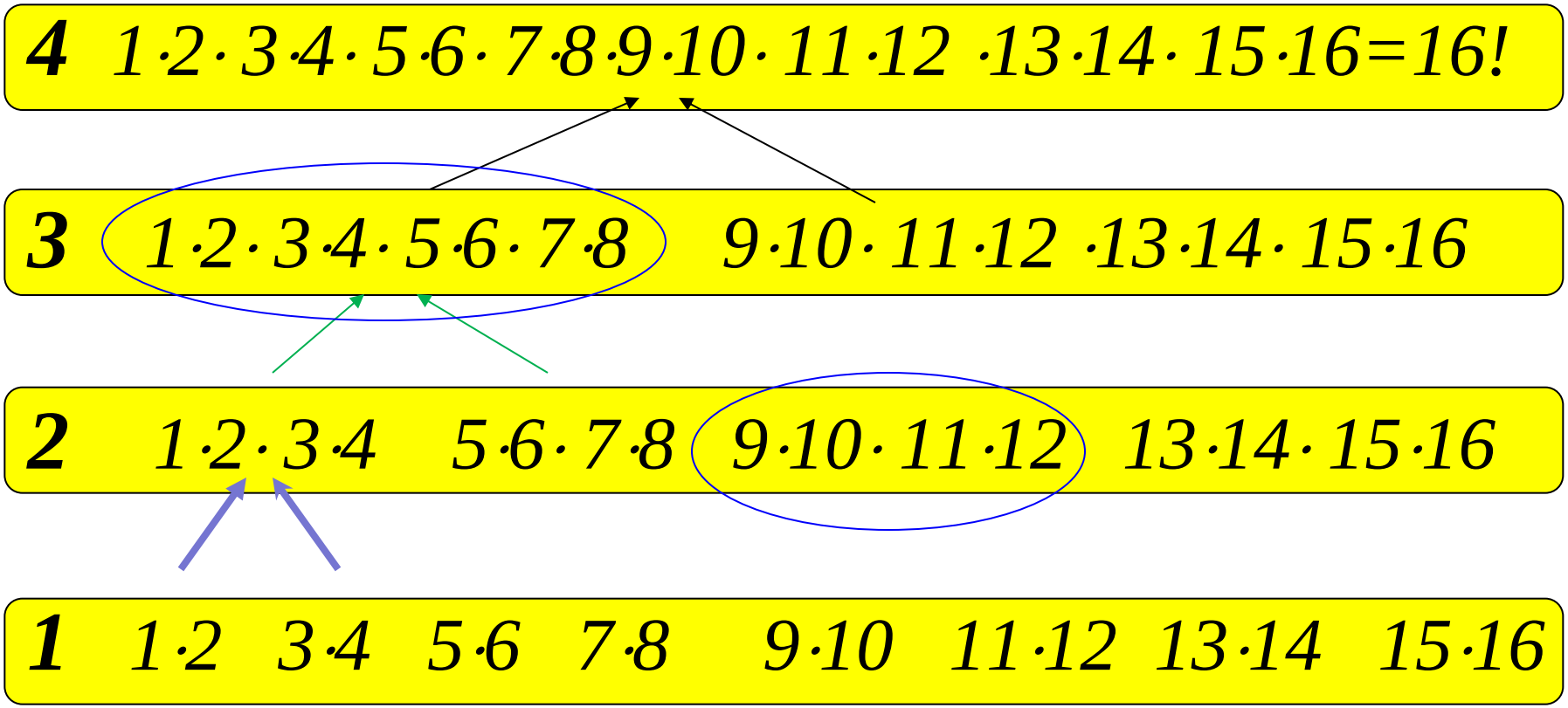
- Вычислить для всех $i = 1, \ldots, n$: x^i
- Вычислить для всех $i = 1, \ldots, n$: $i!$
- Вычислить для всех $i = 1, \ldots, n$: $a_i = \frac{x^i}{i!}$
- Вычислить сумму всех членов a_i

Для вычисления x^i воспользуемся методом бинарного умножения



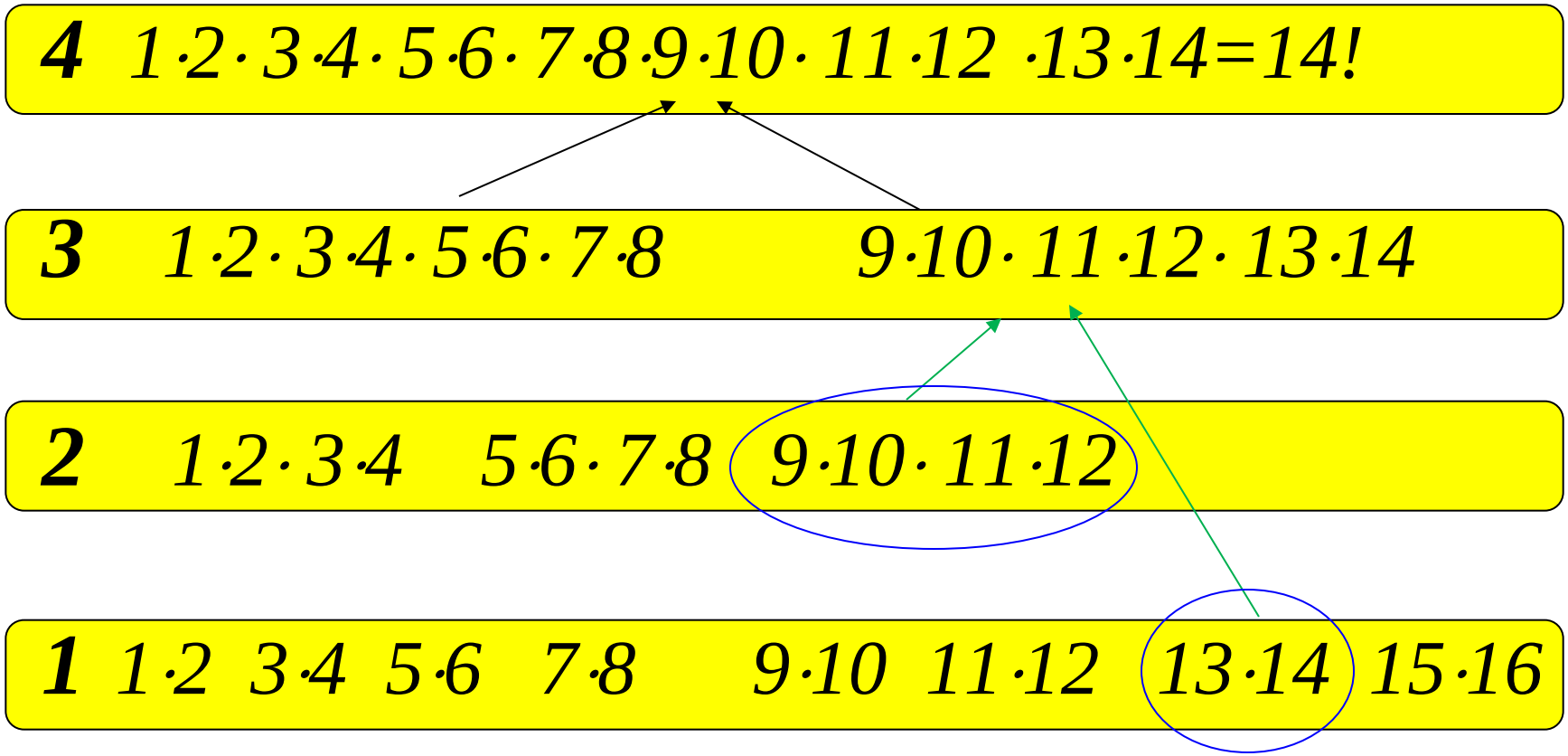
Параллельное вычисление всех требуемых $i!$ *?*

i!


4 1·2·3·4·5·6·7·8·9·10·11·12·13·14·15·16=16!
3 1·2·3·4·5·6·7·8 9·10·11·12·13·14·15·16
2 1·2·3·4 5·6·7·8 9·10·11·12 13·14·15·16
1 1·2 3·4 5·6 7·8 9·10 11·12 13·14 15·16

Для вычисления *i!* воспользуемся аналогичным методом

$$t_2 = \tau_c \log_2 n$$



Новые операции

Вычисление всех факториалов до 4! включительно

Шаг	*Процессор 1*	*Процессор 2*	*Процессор 3*	*Процессор 4*
1	$2!=1\cdot2$			
2	$3!=1\cdot2\cdot3$	$4!=1\cdot2\cdot3\cdot4$		
3				

Новые операции

Вычисление всех факториалов до 4! включительно

Шаг	*Процессор 1*	*Процессор 2*	*Процессор 3*	*Процессор 4*
1	$2!=1\cdot 2$			
2	$3!=(2!)\cdot(3)$	$4!=(1\cdot 2)\cdot(3\cdot 4)$		
3				

Шаг	*Процессор 1*	*Процессор 2*	*Процессор 3*	*Процессор 4*
1	$2!=1\cdot 2$			
2	$3!=12\cdot 3$	$4!=12\cdot 34$		
3				

Новые операции

Вычисление всех факториалов до 4! включительно

Шаг	*Процессор 1*	*Процессор 2*	*Процессор 3*	*Процессор 4*
1	*1·2*	*3·4*		
2	*12·3*	*12·34*		
3				

Новые операции

Вычисление всех факториалов до 8! включительно

Шаг	*Процессор 1*	*Процессор 2*	*Процессор 3*	*Процессор 4*
1	*1·2*	*3·4*	*5·6*	*7·8*
2	*12·3*	*12·34*	*56·7*	*56·78*
3	*1234·5*	*1234·56*	*1234·567*	*1234·5678*

Новые операции

Вычисление всех факториалов до 8! включительно

Шаг	*Процессор 1*	*Процессор 2*	*Процессор 3*	*Процессор 4*
1	*1·2*	*3·4*	*5·6*	*7·8*
2	*12·3*	*12·34*	*56·7*	*56·78*
3	*1234·5*	*1234·56*	*1234·567*	*1234·5678*

```
F=1;
for(i=2;i <= n;i++)
    F*=i;
```

$$T_1(n) = \tau_c (n-1)$$

$$T_{p=n/2}(n) = \tau_c \log_2 n$$

$$S = \frac{n-1}{\log_2 n}\Bigg|_{\substack{n=8\\p=4}} = \frac{7}{3} = 2.33(3) < 4 = p$$

$$E_{p=4}(n=8) = \frac{7}{12}$$

Новые операции

Шаг	Процессор 1	Процессор 2	Процессор 3	Процессор 4
1	1·2	3·4	5·6	7·8
2	12·3	12·34	56·7	56·78
3	1234·5	1234·56	1234·567	1234·5678

$$T_{p=n/2}(n) = \tau_c \log_2 n \quad S = \frac{n-1}{\log_2 n}\Big|_{\substack{n=8\\p=4}} = \frac{7}{3} < 4 = p \quad E_{p=4}(n=8) = \frac{7}{12}$$

Шаг	Процессор 1	Процессор 2	Процессор 3	Процессор 4
1	1 2!	8 3·4	9 5·6	10 7·8
2	2 3!	3 4!	11 56·7	12 56·78
3	4 5!	5 6!	6 7!	7 8!

Параллельный алгоритм

Вычислить для всех $i = 1, \ldots, n : x^i$ $T_{p=n/2}(n) = \tau_c \log_2 n$

Вычислить для всех $i = 1, \ldots, n : i!$ $T_{p=n/2}(n) = \tau_c \log_2 n$

Вычислить для всех $i = 1, \ldots, n : a_i = \frac{x^i}{i!}$ $T_{p=n}(n) = \tau_c$

Вычислить сумму всех членов a_i $T_{p=n/2}(n) = \tau_c \log_2 n$

Заключение

- Определены классы рассматриваемых вычислительных систем
- Представлены модели параллельных программ
- Представлен ряд способов организации взаимодействия последовательных процессов
- Представлен алгоритм, обладающий низкой эффективностью, но высоким быстродействием

Вопросы для обсуждения

- Сколько нужно процессоров для вычисления суммы ряда $S = \sum_{i=0}^{n} \frac{x^i}{i!}$ за время

$$2\tau_c \log_2 n + q\tau_c \ ,$$

где q – константа

- Приведите примеры алгоритмов, обладающих эффективностью больше 100%.

- Есть ли процессорные инструкции P(S) и V(S)?

Контакты

Якобовский М.В.

д.ф.-м.н., проф.,

зав. сектором

«Программного обеспечения многопроцессорных систем и вычислительных сетей»

Института прикладной математики им. М.В.Келдыша Российской академии наук

mail: lira@imamod.ru

web: http://lira.imamod.ru