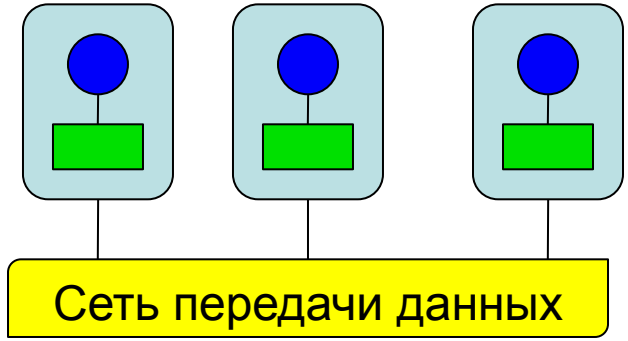
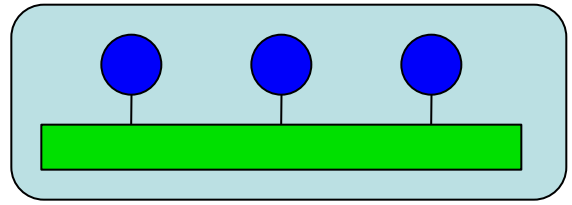
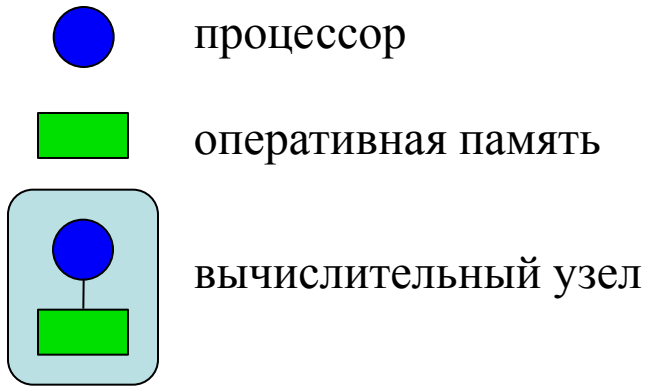
Уточнение круга рассматриваемых систем

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



- Системы с доступом всех процессоров к общей оперативной памяти

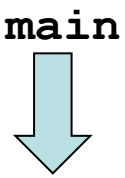




Модель выполнения программы на общей памяти

- 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

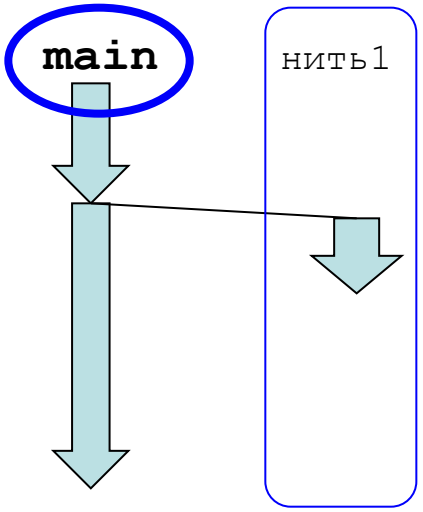
```
int a_global;
main()
{
int b1_local;
Запуск нити(fun())
}
fun()
{
int b2_local;
Запуск нити(…)
}
```



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

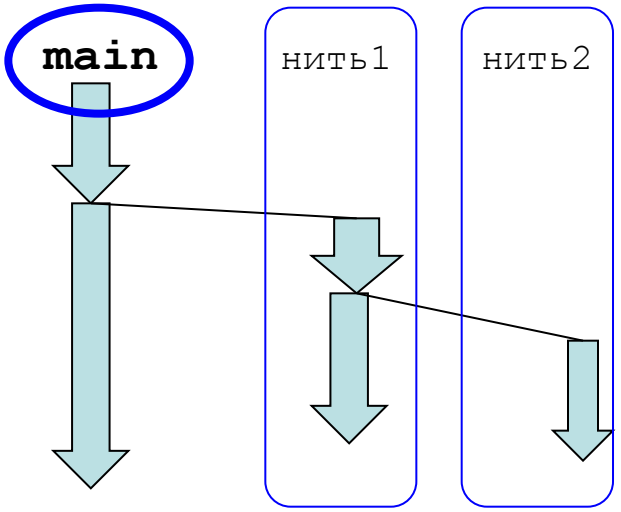
```
int a_global;
main()
{
int b1_local;
Запуск нити(fun())
}
fun()
{
int b2_local;
Запуск нити(…)
}
```



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

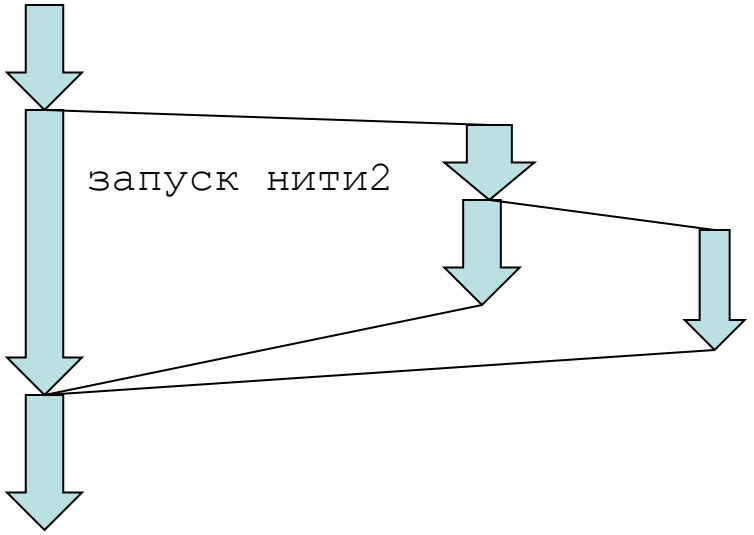
```
int a_global;
main()
{
int b1_local;
Запуск нити(fun())
}
fun()
{
int b2_local;
Запуск нити(…)
}
```



Модель программы на общей памяти

- Работа начинается с запуска одной программы
- При необходимости программа порождает новые процессы, эти процессы:
  - Обладают собственными локальными переменными
  - Имеют доступ к общим глобальным переменным

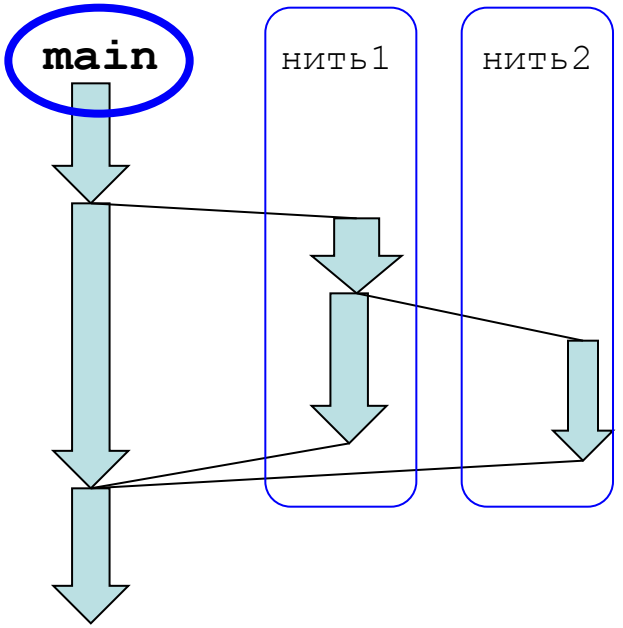
```
int a_global;
main()
{
Start_process(fun);
int b1_local;
}

fun()
{
int b2_local;
}
```



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;
```

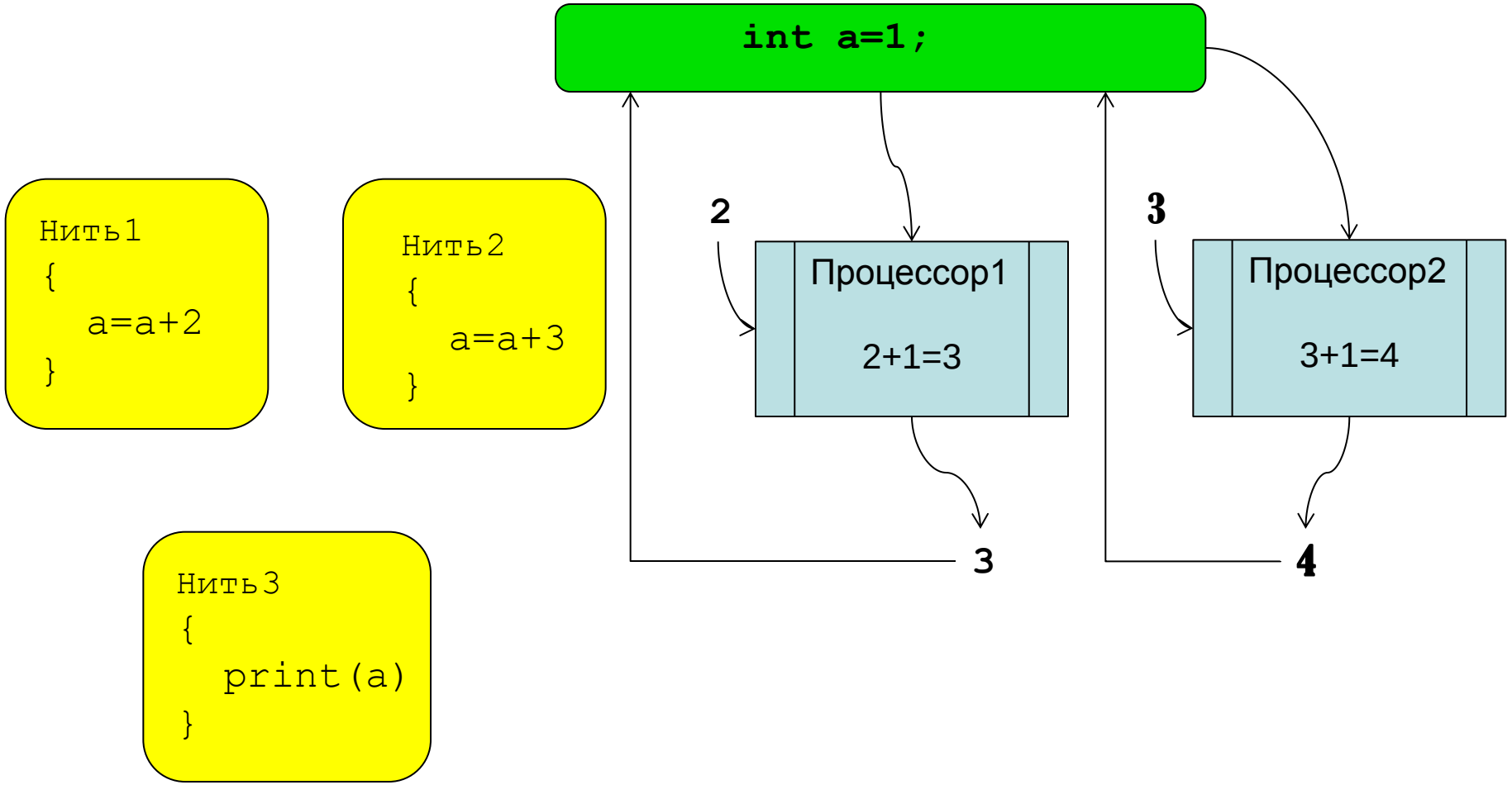
```
Нить1
{
   a=a+2
}
```

```
Нить3
{
   print(a)
}
```

```
Нить2
{
   a=a+3
}
```

3

Что будет напечатано?



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) – атомарная !
  - Атомарно увеличивает значение S на 1

- Операция P(S) – атомарная !
  - 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

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

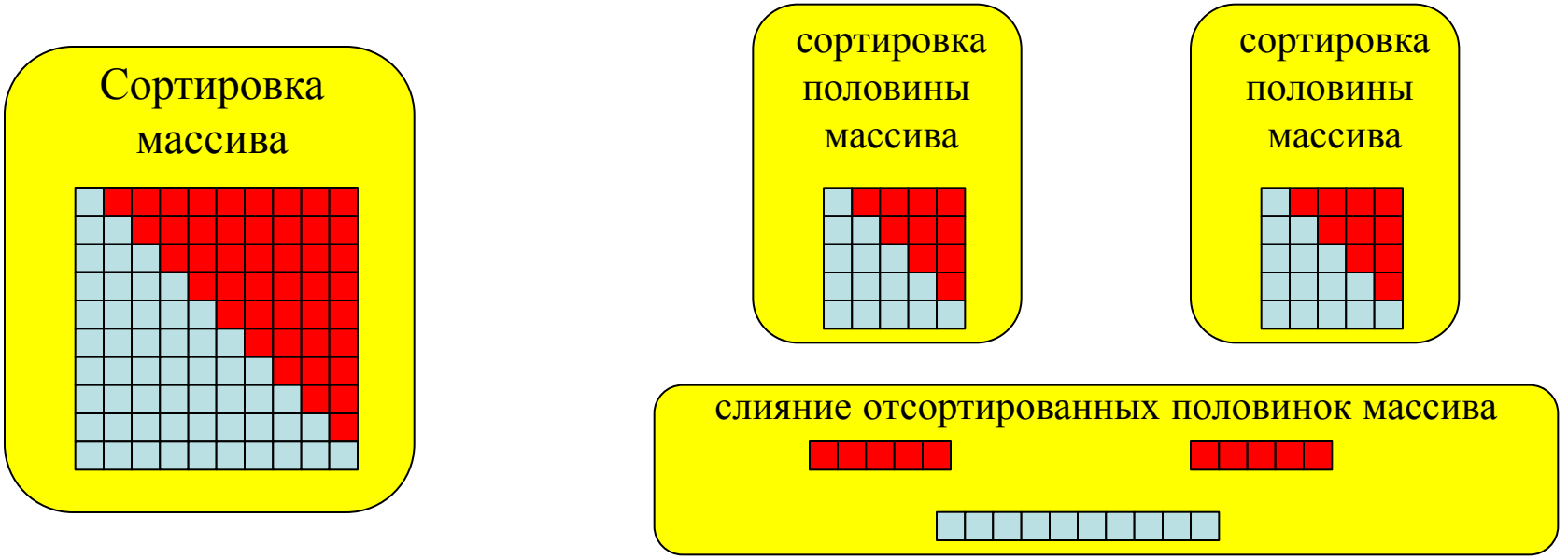
Может ли ускорение превышать число процессоров?

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

Да, если выбран неудачный последовательный алгоритм

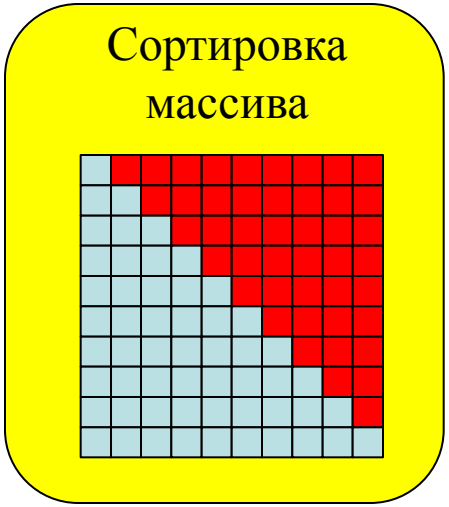
Может ли быть S(p)>p ?

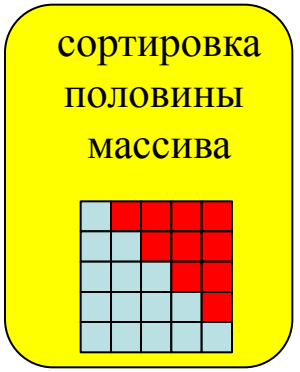
– Пример неудачного последовательного алгоритма

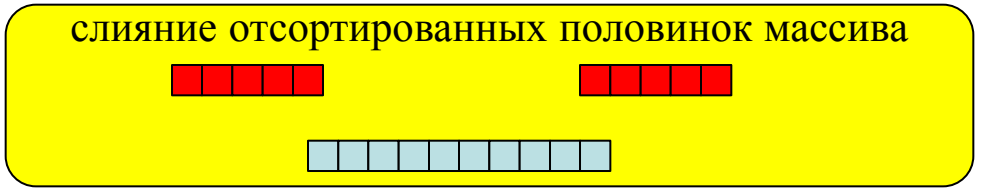


Может ли быть S(p)>p ?

– Пример неудачного последовательного алгоритма





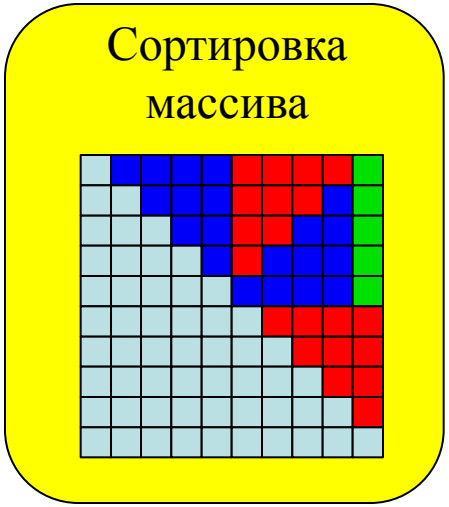


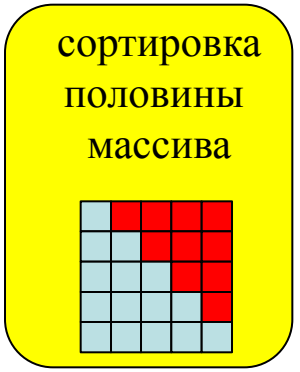
$$\frac{n(n-1)}{2} \sim \frac{n^2}{2}$$

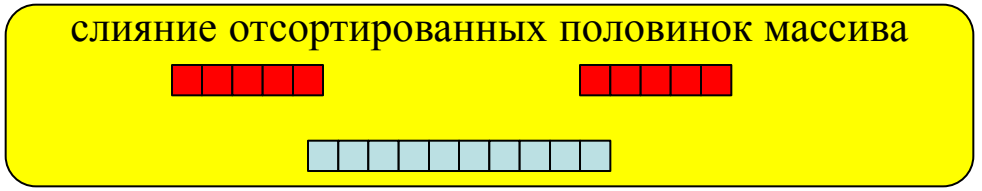
$$\frac{\left(\frac{n}{2}\right)^2}{2} + n \sim \frac{1}{4}\frac{n^2}{2}$$

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

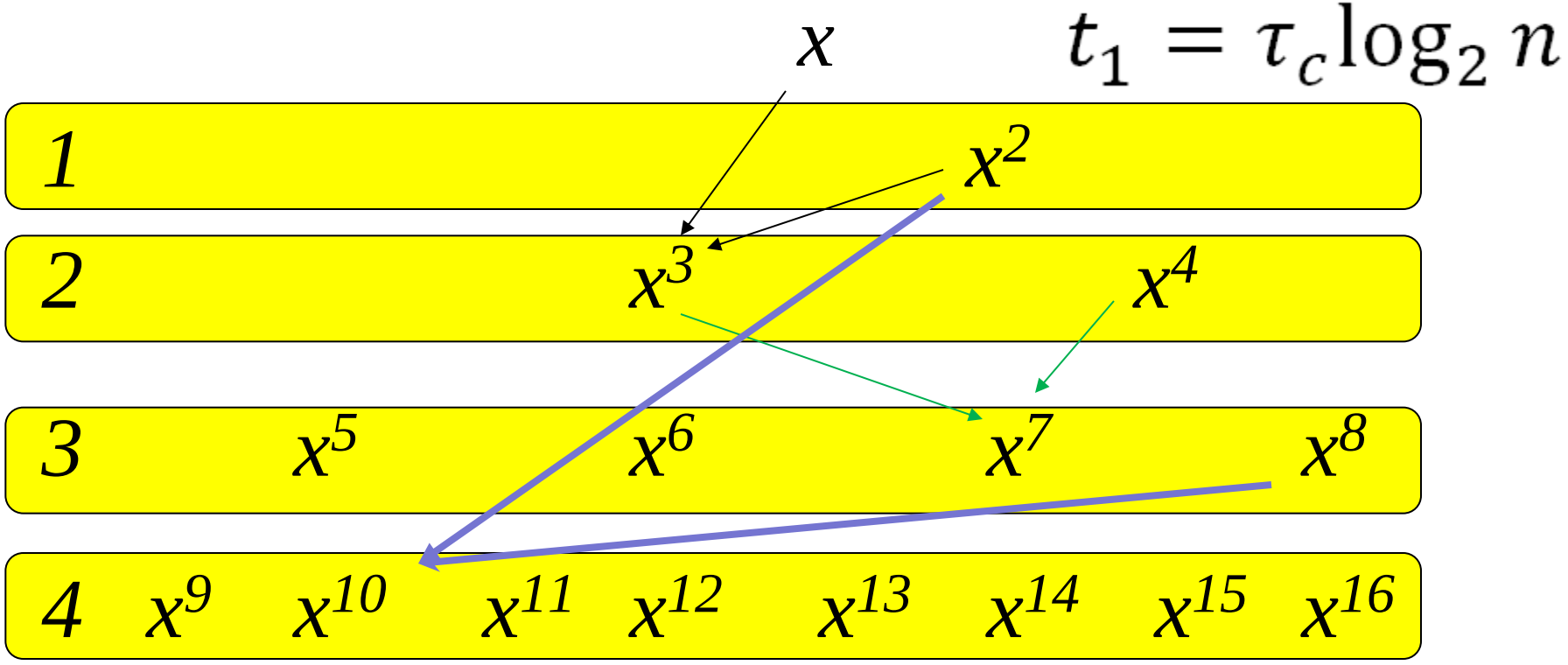
```
S=1;
a=1;
for(i=1;i<= n;i++)
  {
     a=a*x/i;
     S=S+a;
  }
```

$$T_1 = 3n\tau_c$$

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

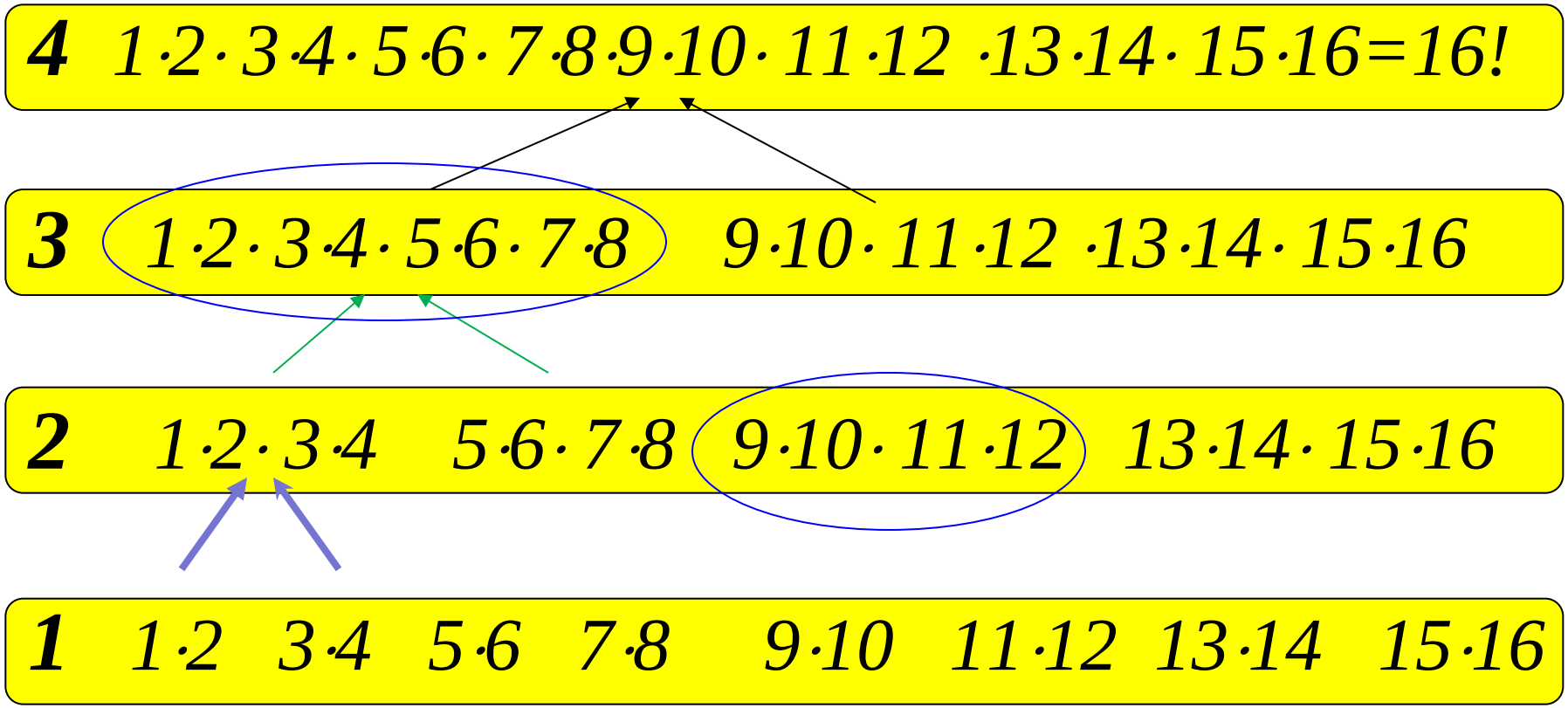
Для вычисления x^i воспользуемся методом бинарного умножения



Параллельное вычисление всех требуемых $i!$ *?*

?

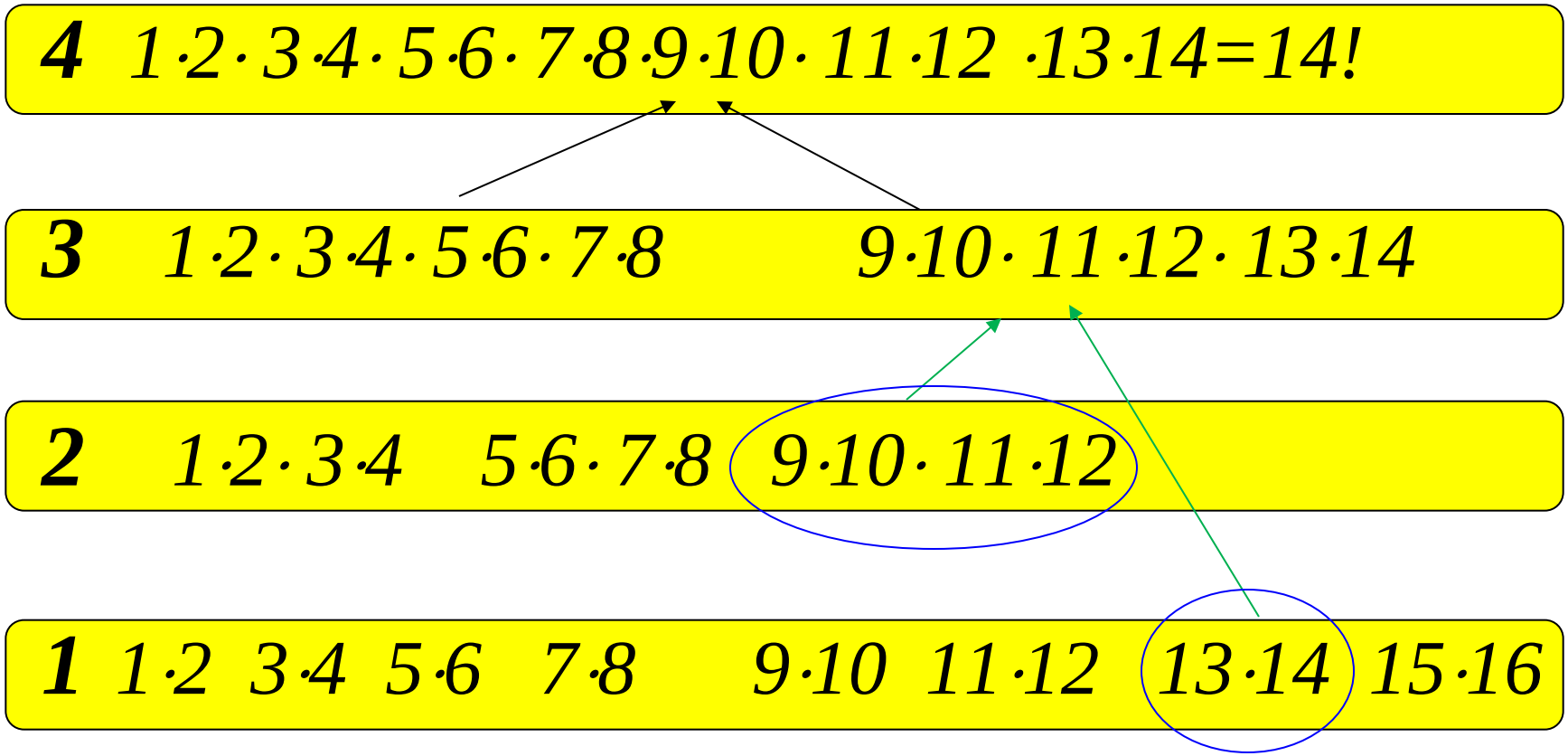
$i!$


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

14!

Для вычисления $i!$ воспользуемся аналогичным методом

$$t_2 = \tau_c \log_2 n$$



Параллельный алгоритм

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

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

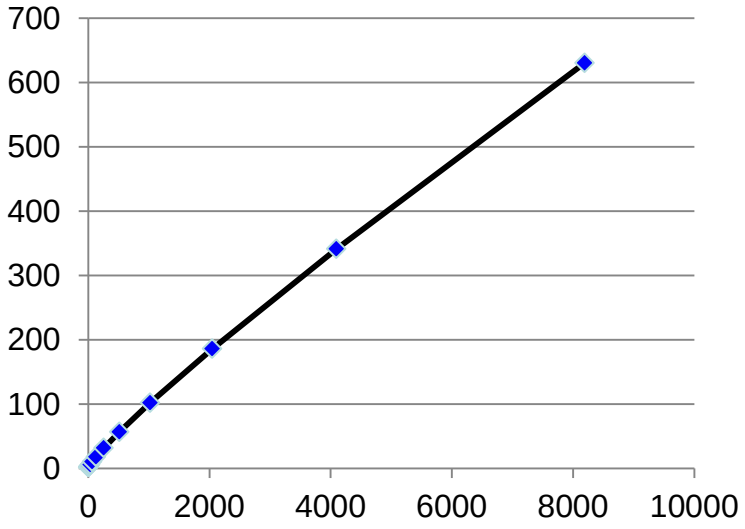
Вычислить для всех $i = 1, \ldots, n : a_i = \frac{x^i}{i!}$ $T_{p=n}(n) = \tau_c$

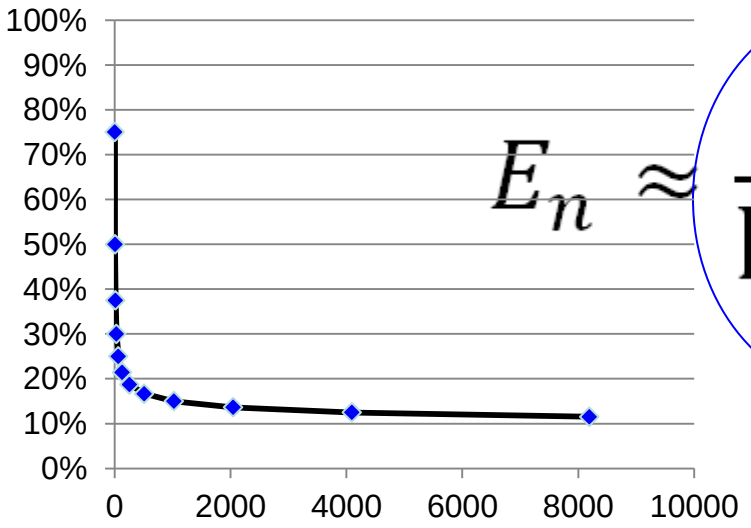
Вычислить сумму всех членов a_i $T_{p=n/2}(n) = \tau_c \log_2 n$

Ускорение и эффективность при $p=n$

$p=n$

$$S_n = \frac{3n\tau_c}{2\tau_c \log_2 n + \tau_c} \approx \frac{n}{\log_2 n} \mathbf{1.5}$$





$$E_n \approx \frac{\mathbf{1.5}}{\log_2 n}$$

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., *чл.-корр. РАН, проф., д.ф.-м.н.,*

Заместитель директора по научной работе
Института прикладной математики
им. М.В.Келдыша Российской академии наук

mail: lira@imamod.ru

web: http://lira.imamod.ru