

# Интернет Университет Суперкомпьютерных технологий

## Лекция 3

### *Методы построения параллельных программ (продолжение)*

Учебный курс

### *Введение в параллельные алгоритмы*

Якобовский Михаил Владимирович  
проф., д.ф.-м.н.

Институт прикладной математики им. М.В.  
Келдыша РАН, Москва

# Предварительные замечания

*... если для нас представляют интерес реально работающие системы, то требуется убедиться, (и убедить всех сомневающихся) в корректности наших построений*

*... системе часто придется работать в невоспроизводимых обстоятельствах, и мы едва ли можем ожидать сколько-нибудь серьезной помощи от тестов*

*Dijkstra E.W.  
1966*

# Содержание лекции

---

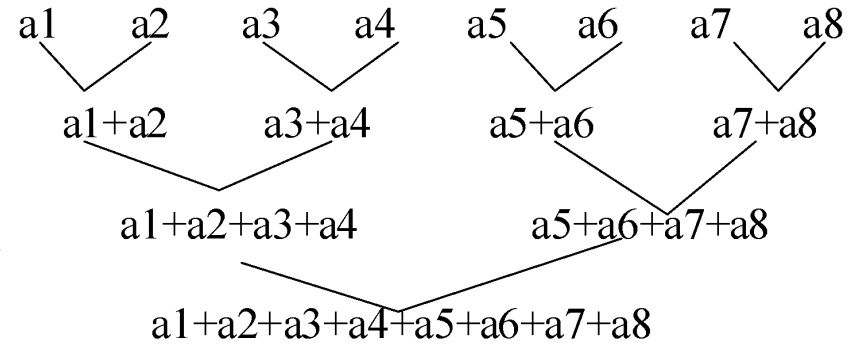
- Методы построения параллельных алгоритмов и их свойства:
  - Статическая балансировка
    - метод сдвигания
    - геометрический параллелизм
    - конвейерный параллелизм
  - Динамическая балансировка
    - коллективное решение
    - диффузная балансировка загрузки

# Метод сдвигивания

## Каскадная схема

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

$$S_{p=n/2}(n) = \frac{(n-1)}{\log_2 n} \quad E_{p=n/2}(n) \approx \frac{1}{\log_2 n}$$

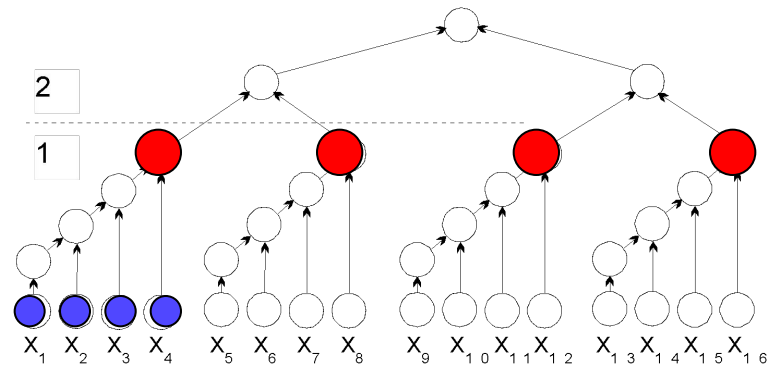


## Модифицированная каскадная схема

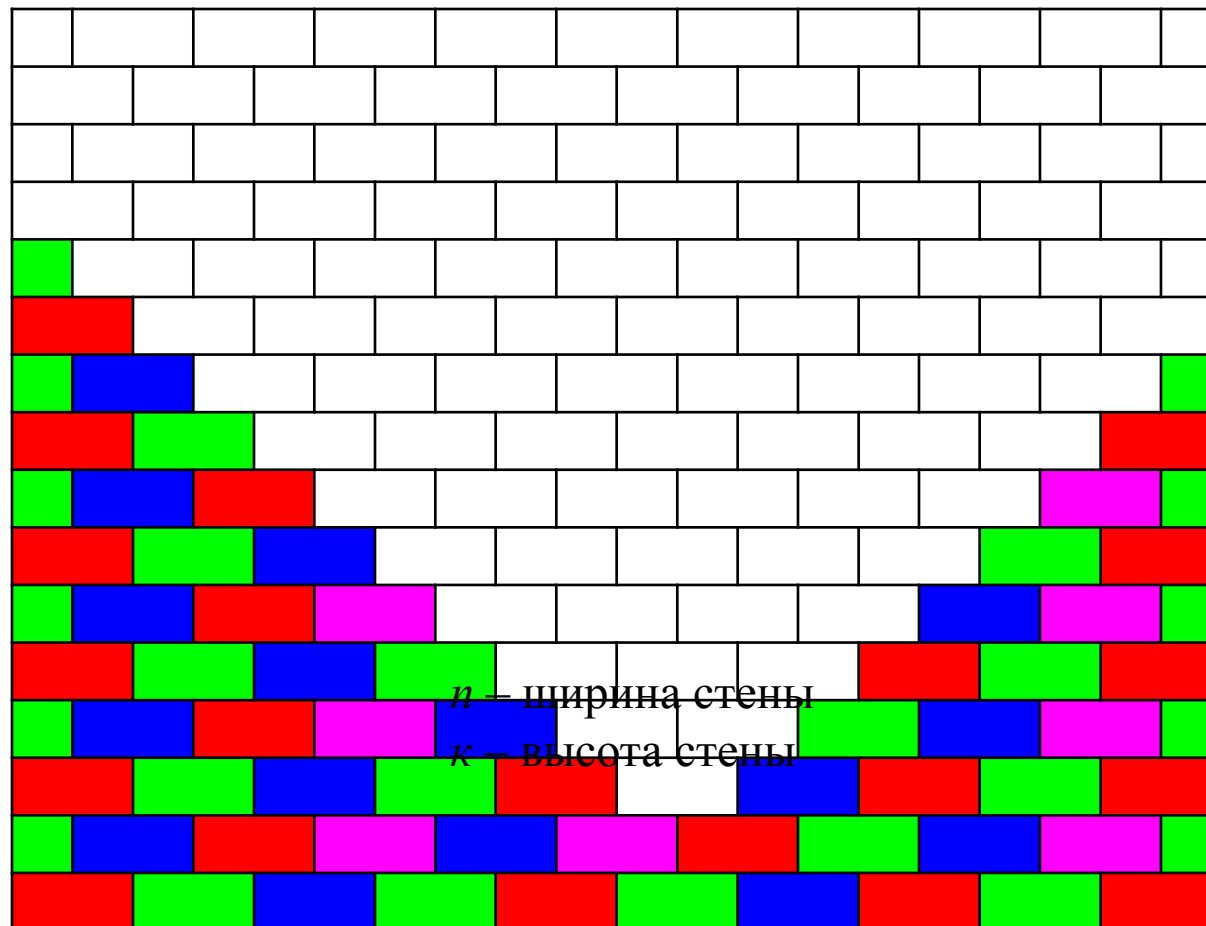
[В.П.Гергель Основы параллельных вычислений, лекция 4, слайд 23](#)

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

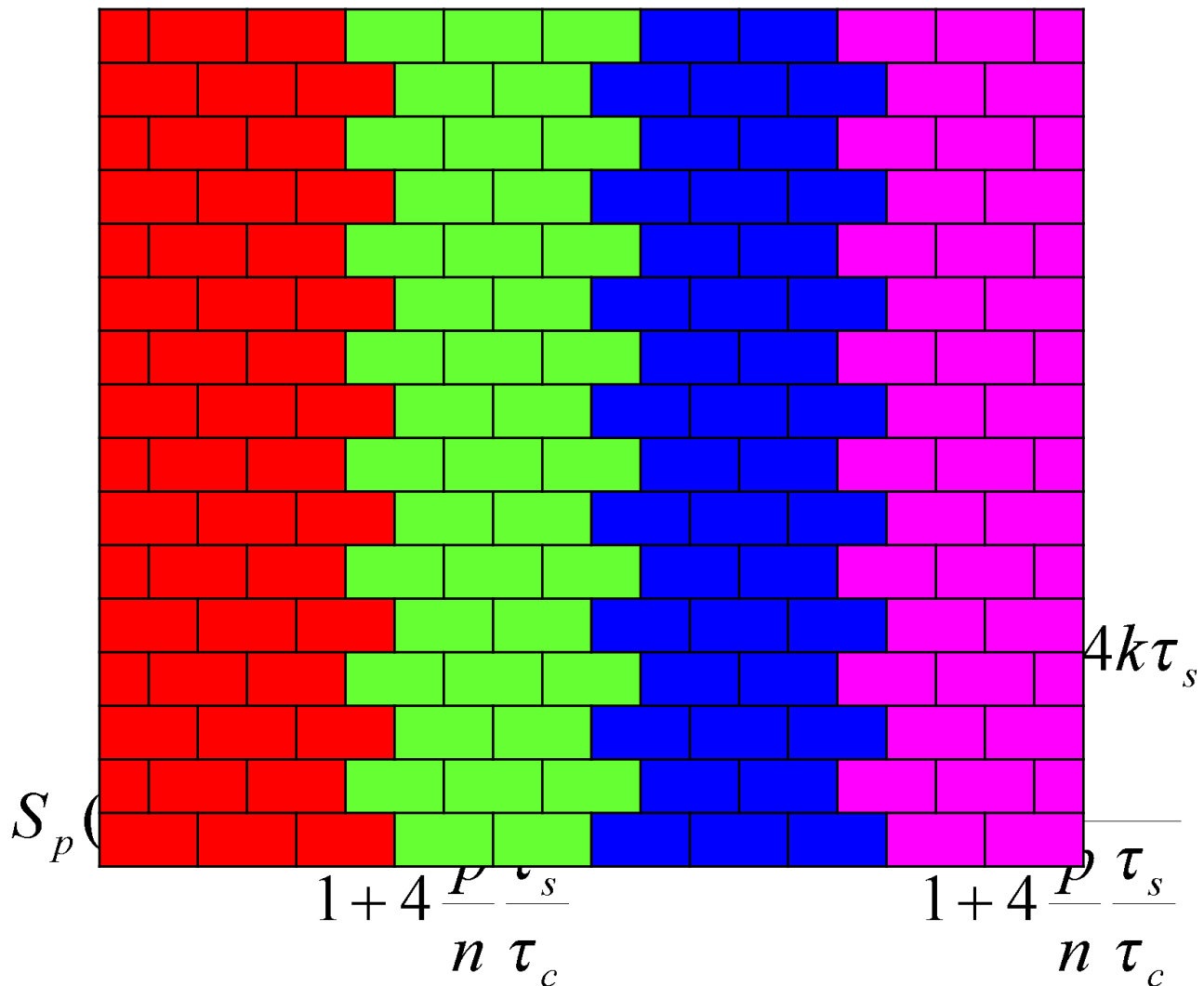
$$S_{p=\frac{n}{\log_2 n}}(n) \approx \frac{(n-1)}{2\log_2 n} \quad E_{p=\frac{n}{\log_2 n}}(n) \approx \frac{1}{2}$$



# Стена Фокса

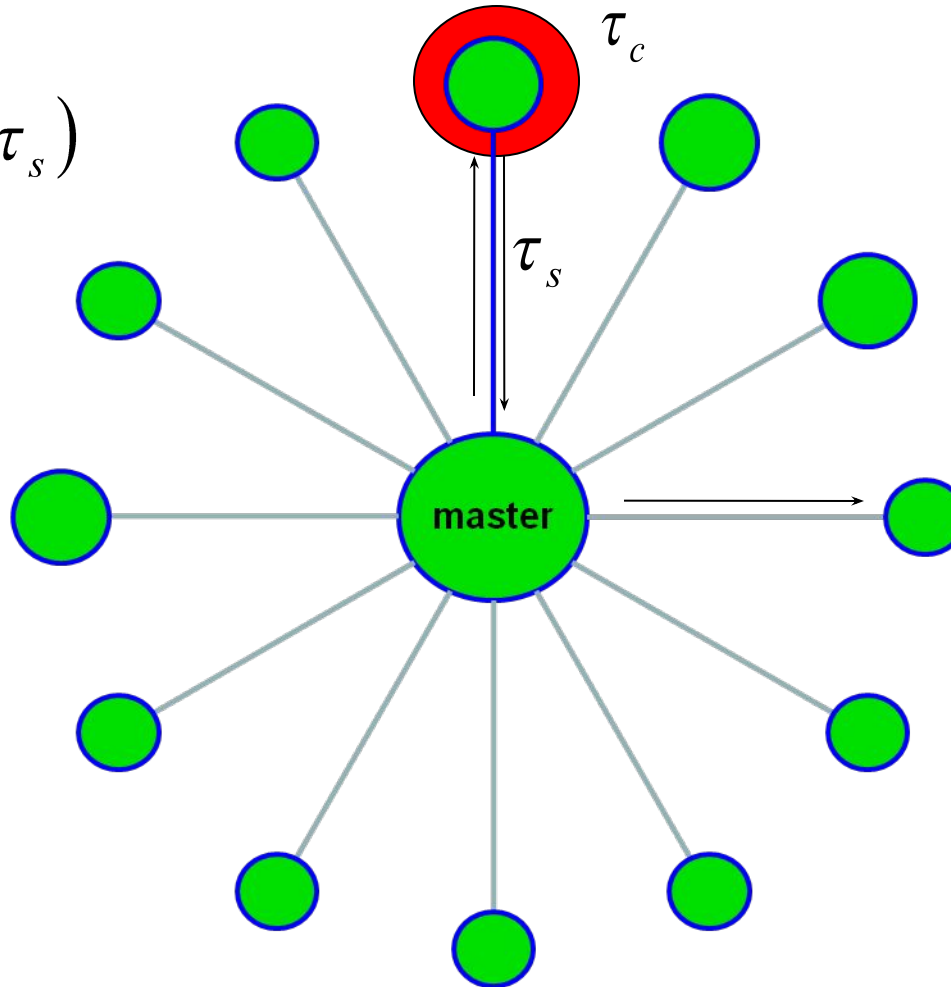


# Метод геометрического параллелизма



# Метод коллективного решения (укладка паркета)

$$T_p = \frac{N}{p} (\tau_c + \tau_s)$$



$$p_{\max} = \frac{\tau_c}{\tau_s}$$

# Метод коллективного решения (укладка паркета)

$$T_1(kn) = \tau_c kn$$

Число порций

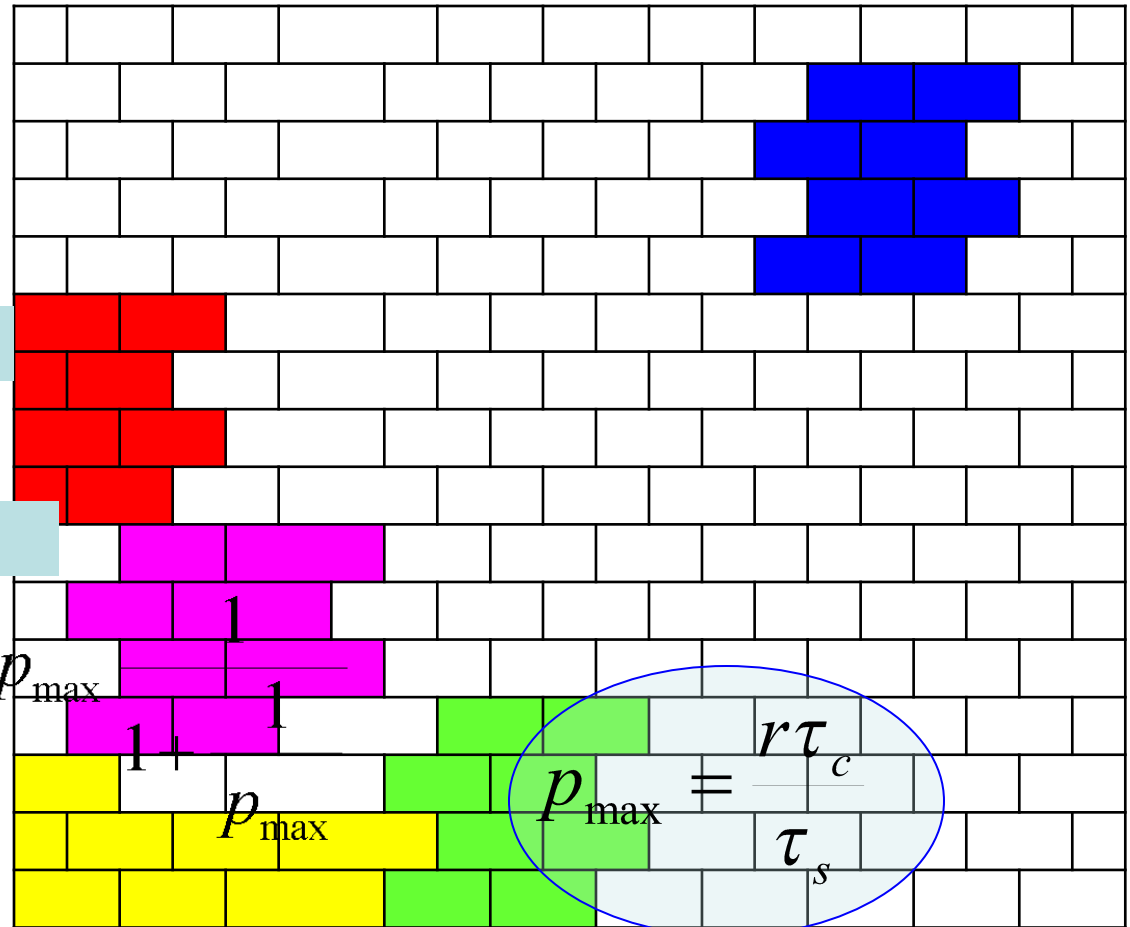
$$T_p(kn) = \frac{kn}{rp} (r\tau_c + \tau_s)$$

Обработка порции

$$S_{p=\frac{r\tau_c}{\tau_s}}(kn) = \left( \frac{r\tau_c}{\tau_s} \right)^2 \frac{1}{1 + \frac{r\tau_c}{\tau_s}} = p_{\max}$$

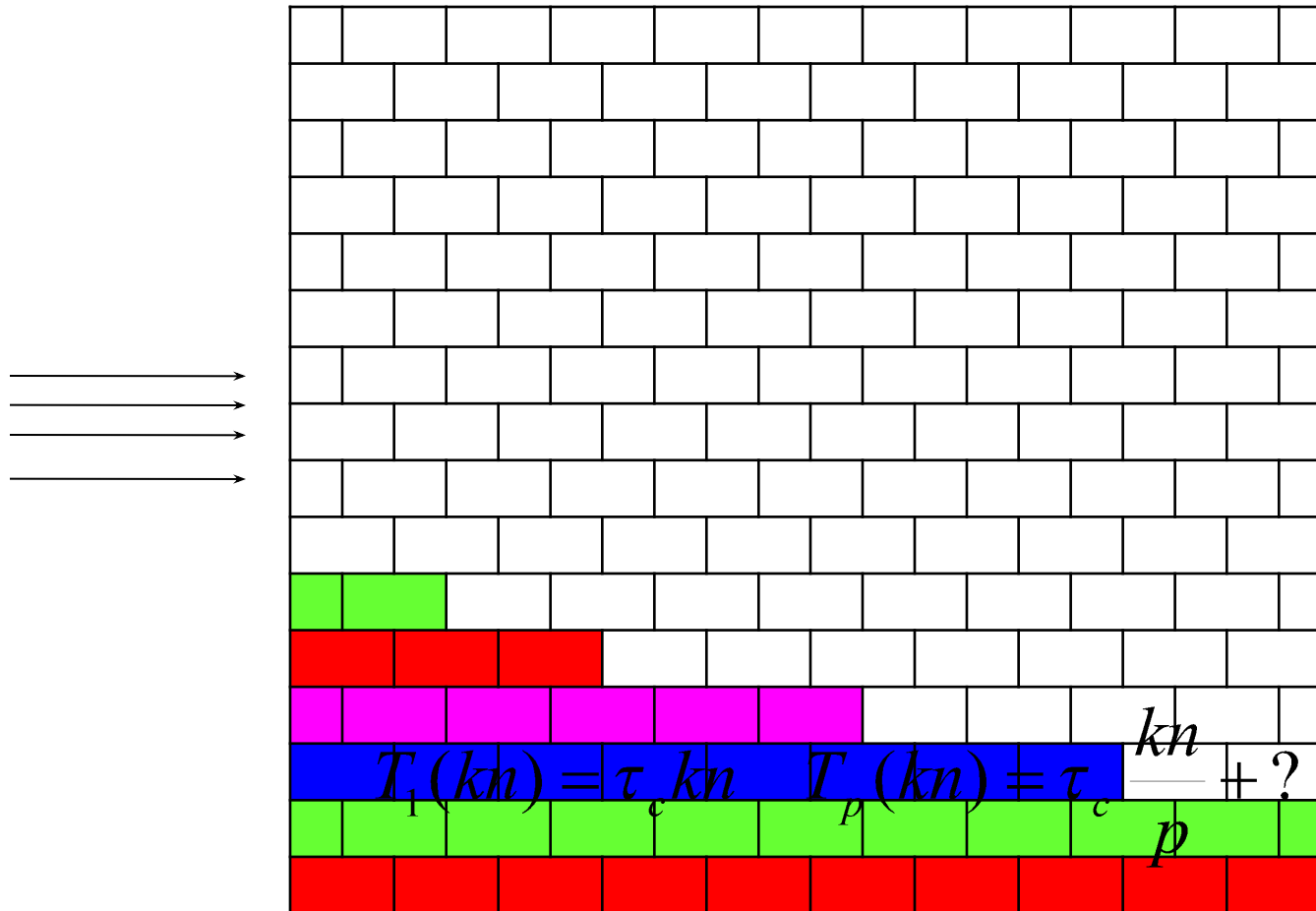
Обмен данными

$$E_{p=\frac{r\tau_c}{\tau_s}}(kn) = \frac{1}{1 + \frac{\tau_s}{r\tau_c}}$$





# Метод конвейерного параллелизма



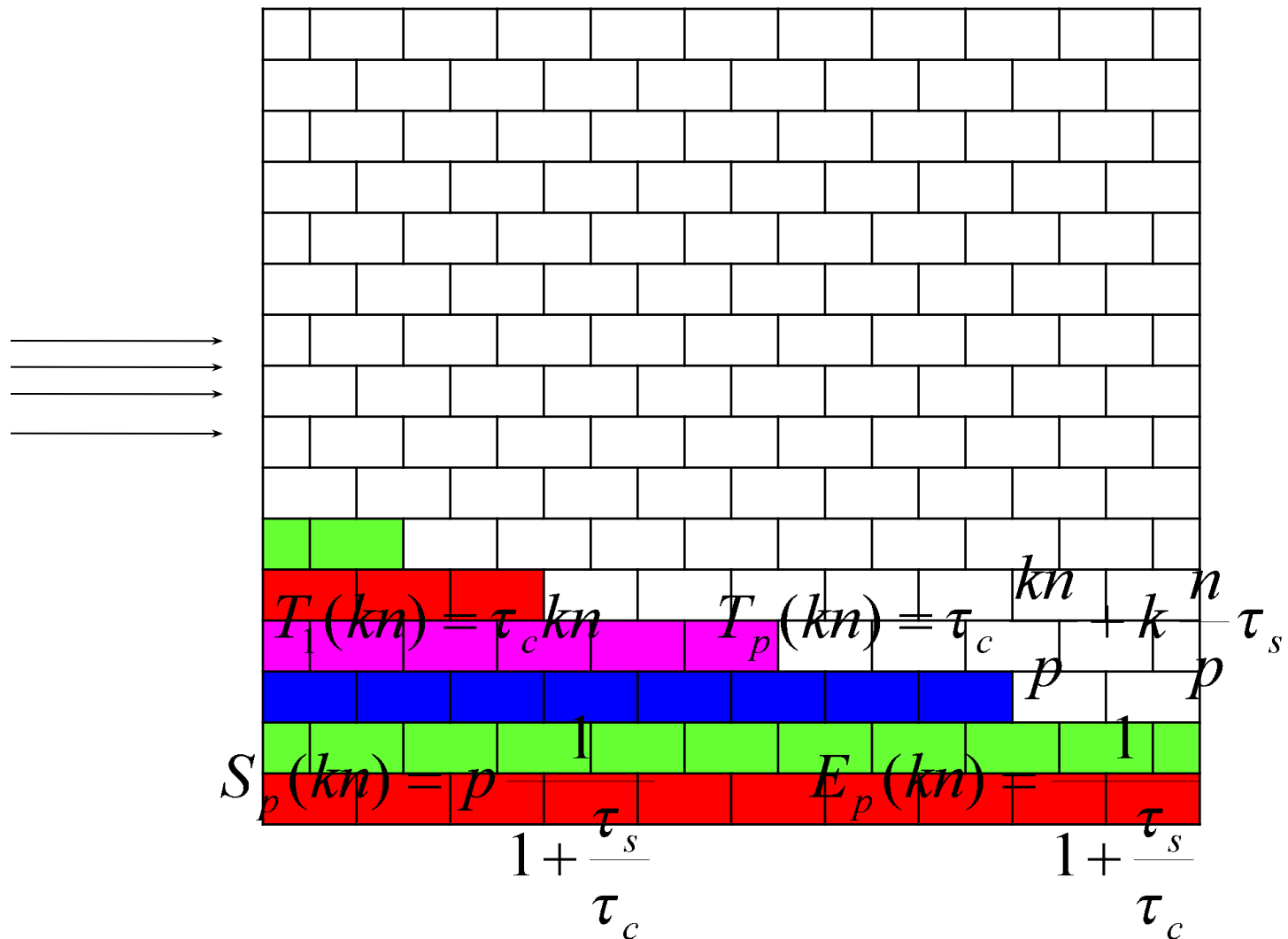
# Метод конвейерного параллелизма

Время выполнения на  $p$  процессорах

?

$$T_1(kn) = \tau_c kn \quad T_p(kn) = \tau_c \frac{kn}{p} + ?$$

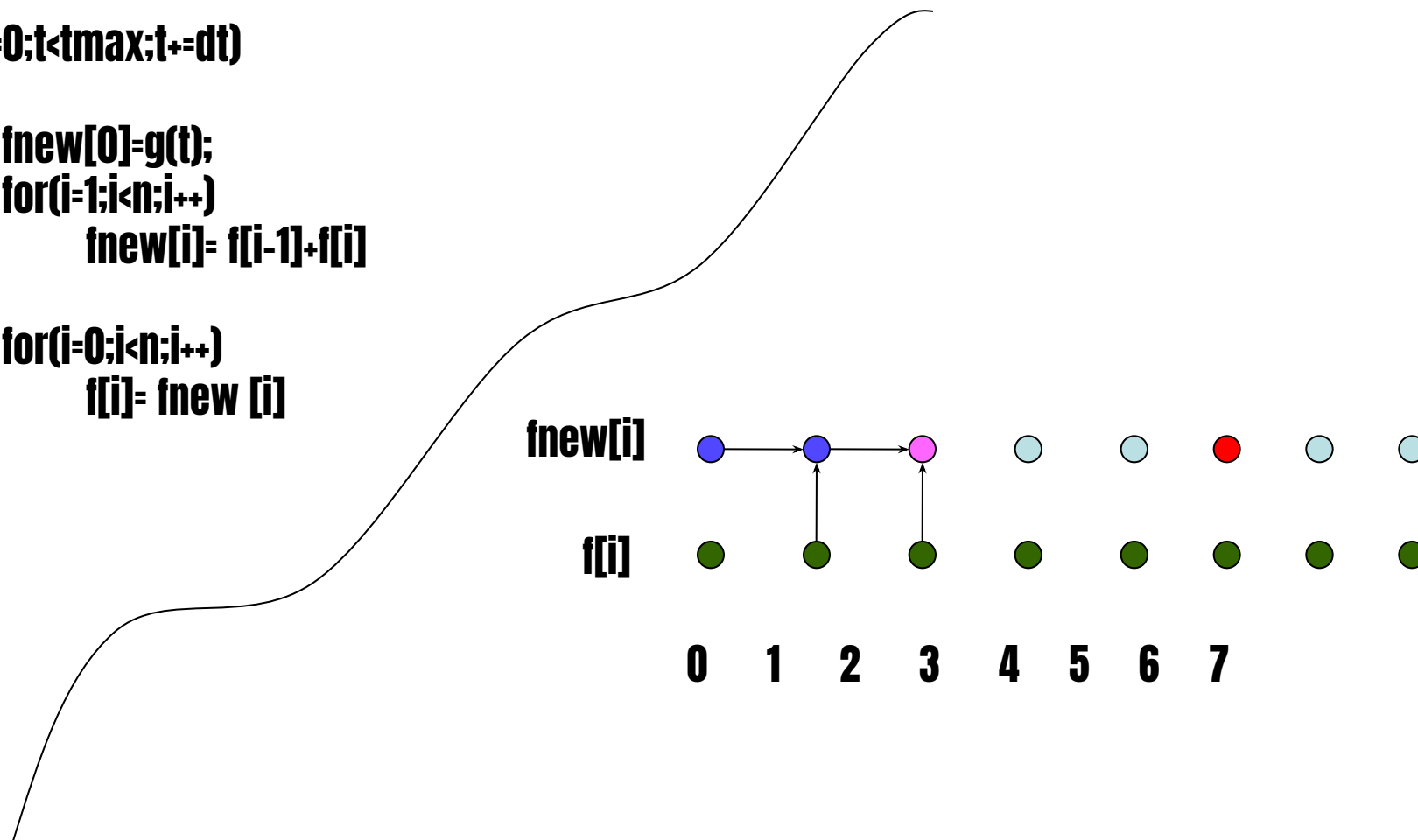
# Метод конвейерного параллелизма



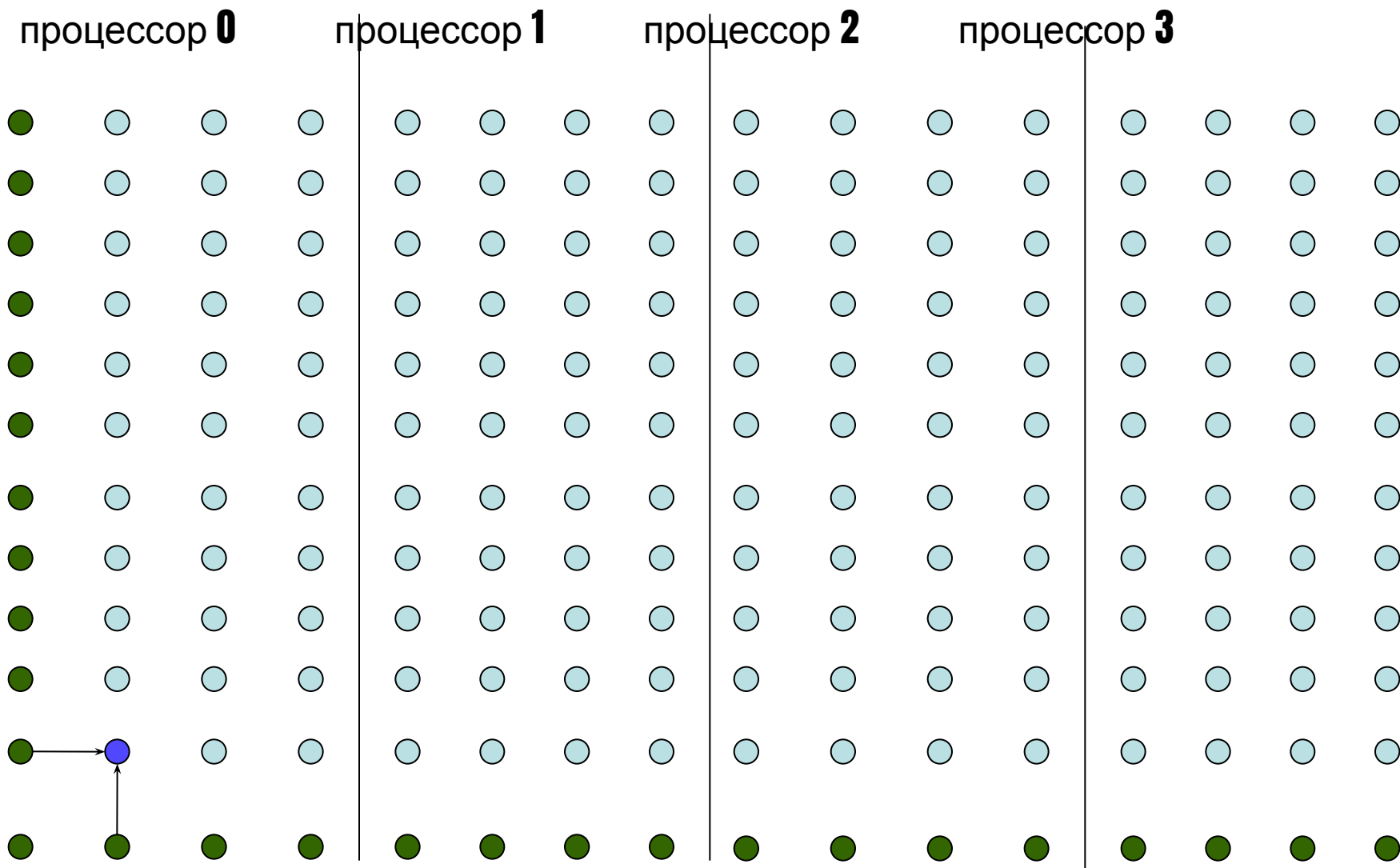
# Метод конвейерного параллелизма

```
for(t=0;t<tmax;t+=dt)
{
  fnew[0]=g(t);
  for(i=1;i<n;i++)
    fnew[i]= f[i-1]+f[i]

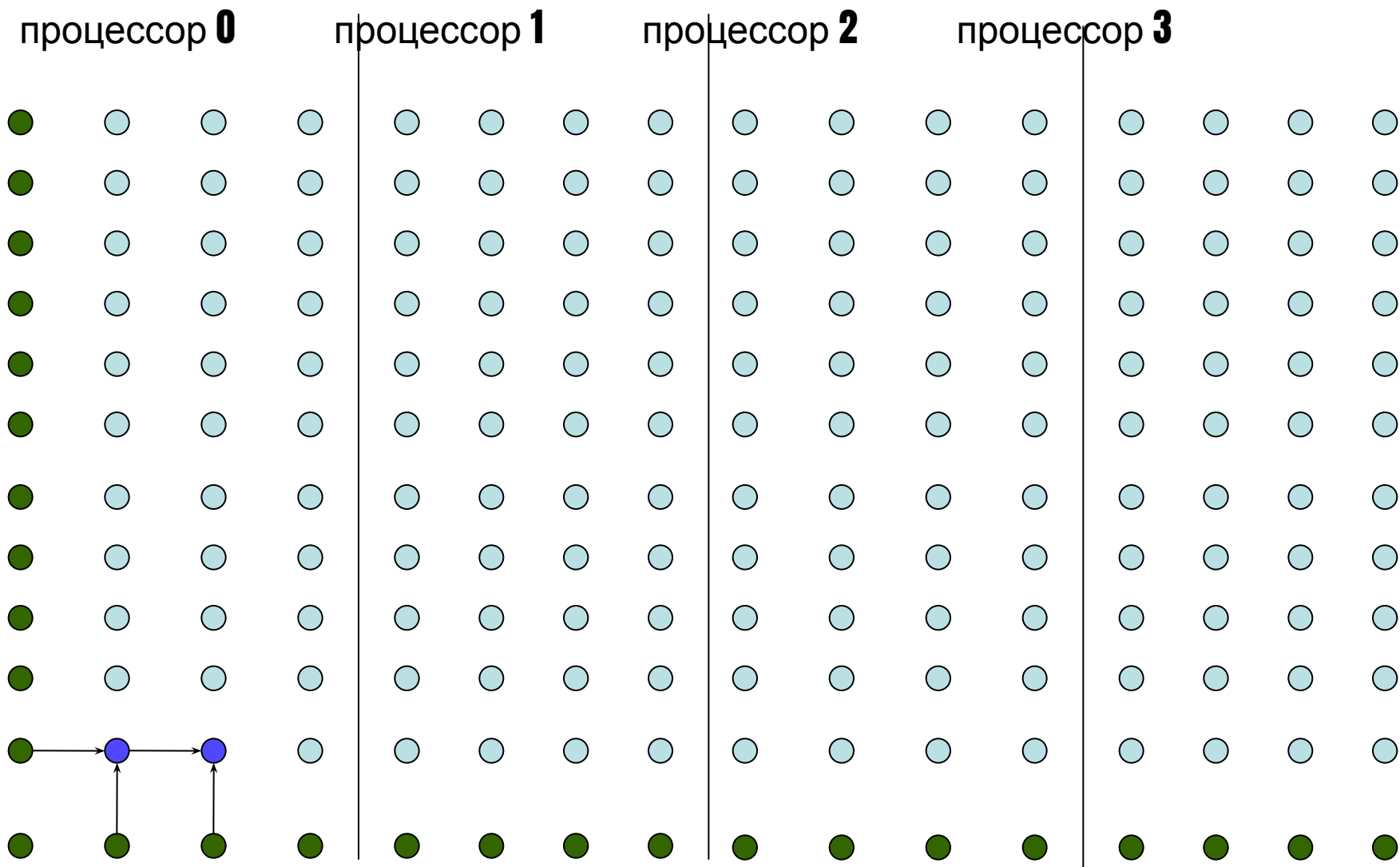
  for(i=0;i<n;i++)
    f[i]= fnew [i]
}
```



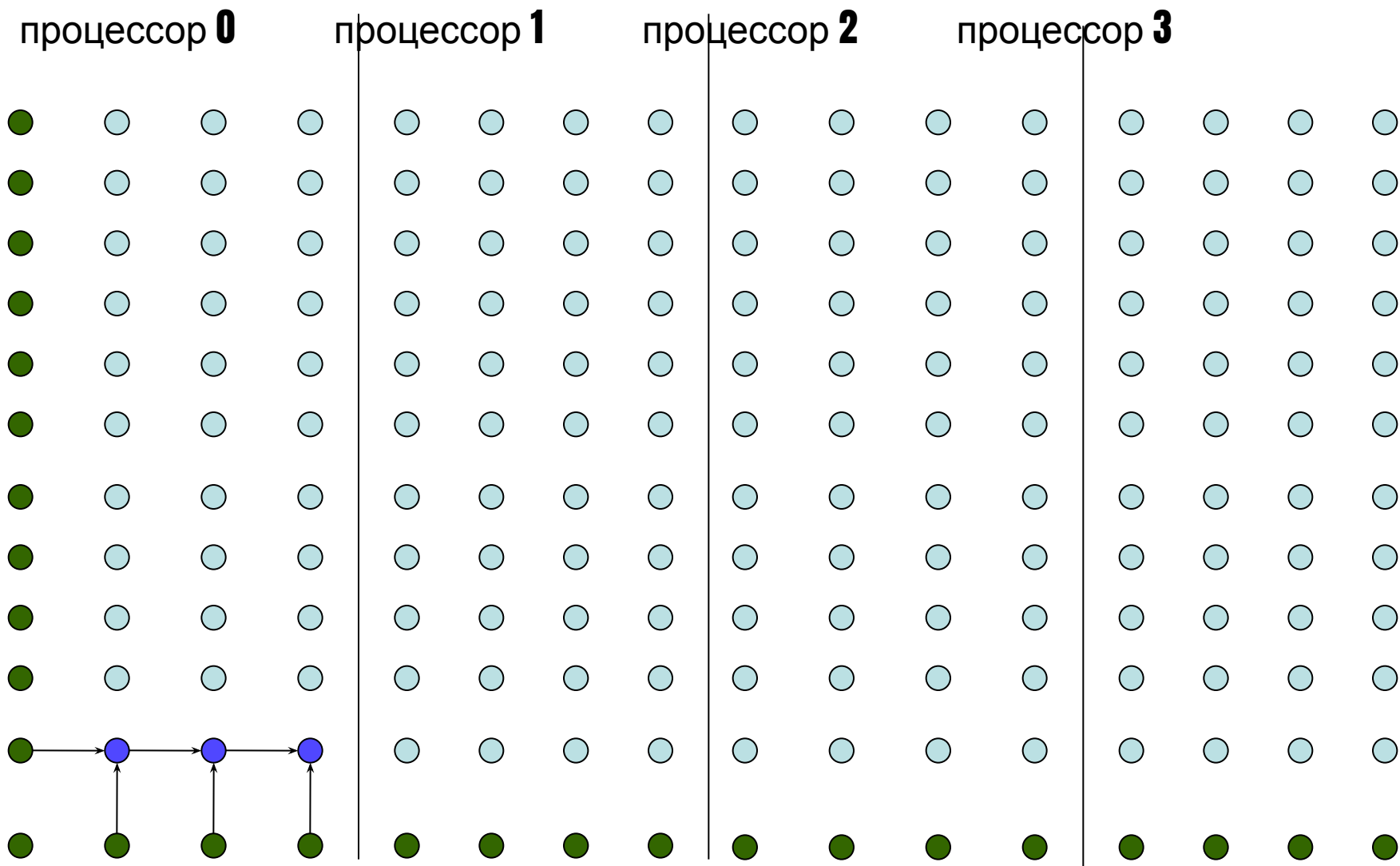
# Метод конвейерного параллелизма



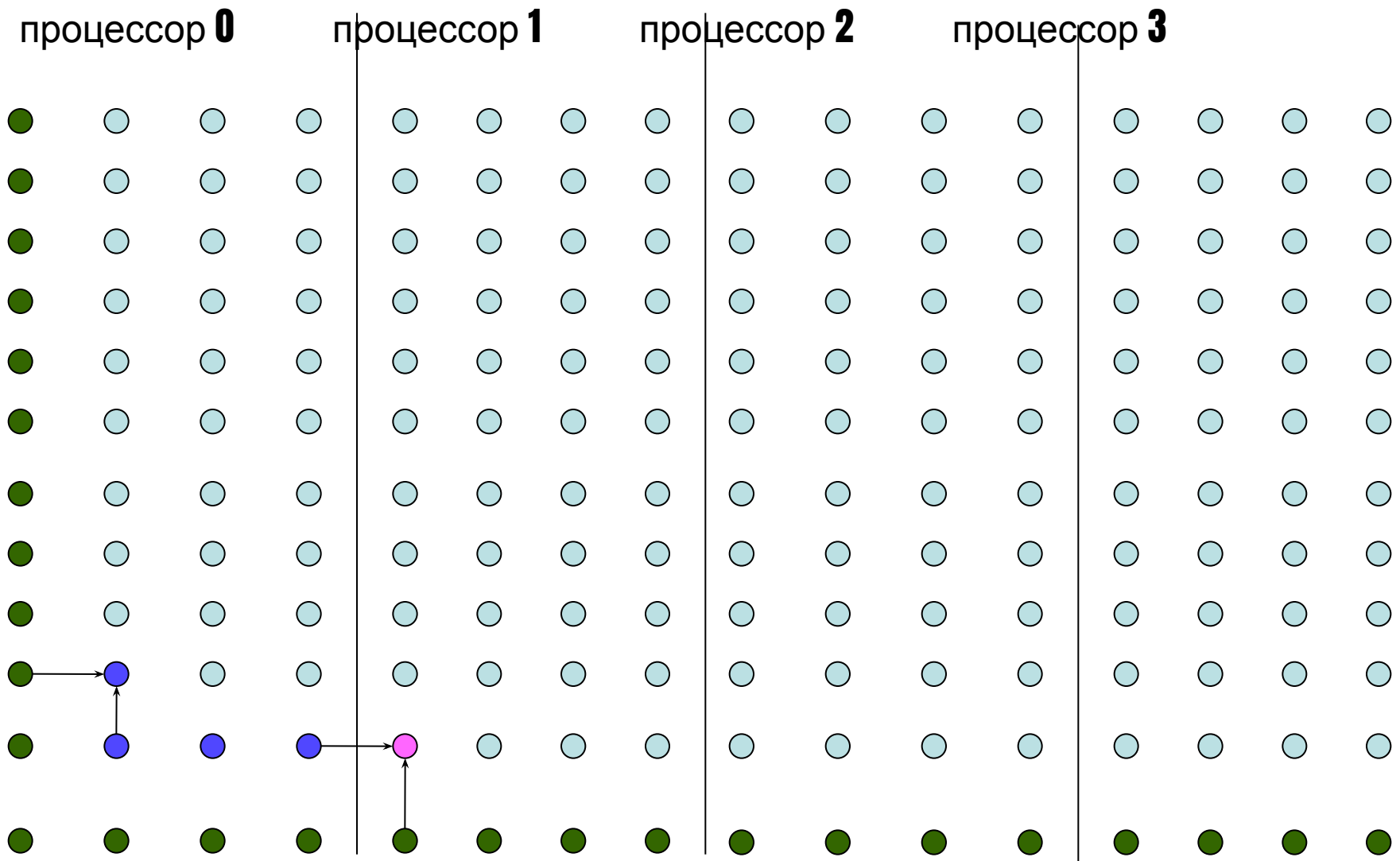
# Метод конвейерного параллелизма



# Метод конвейерного параллелизма

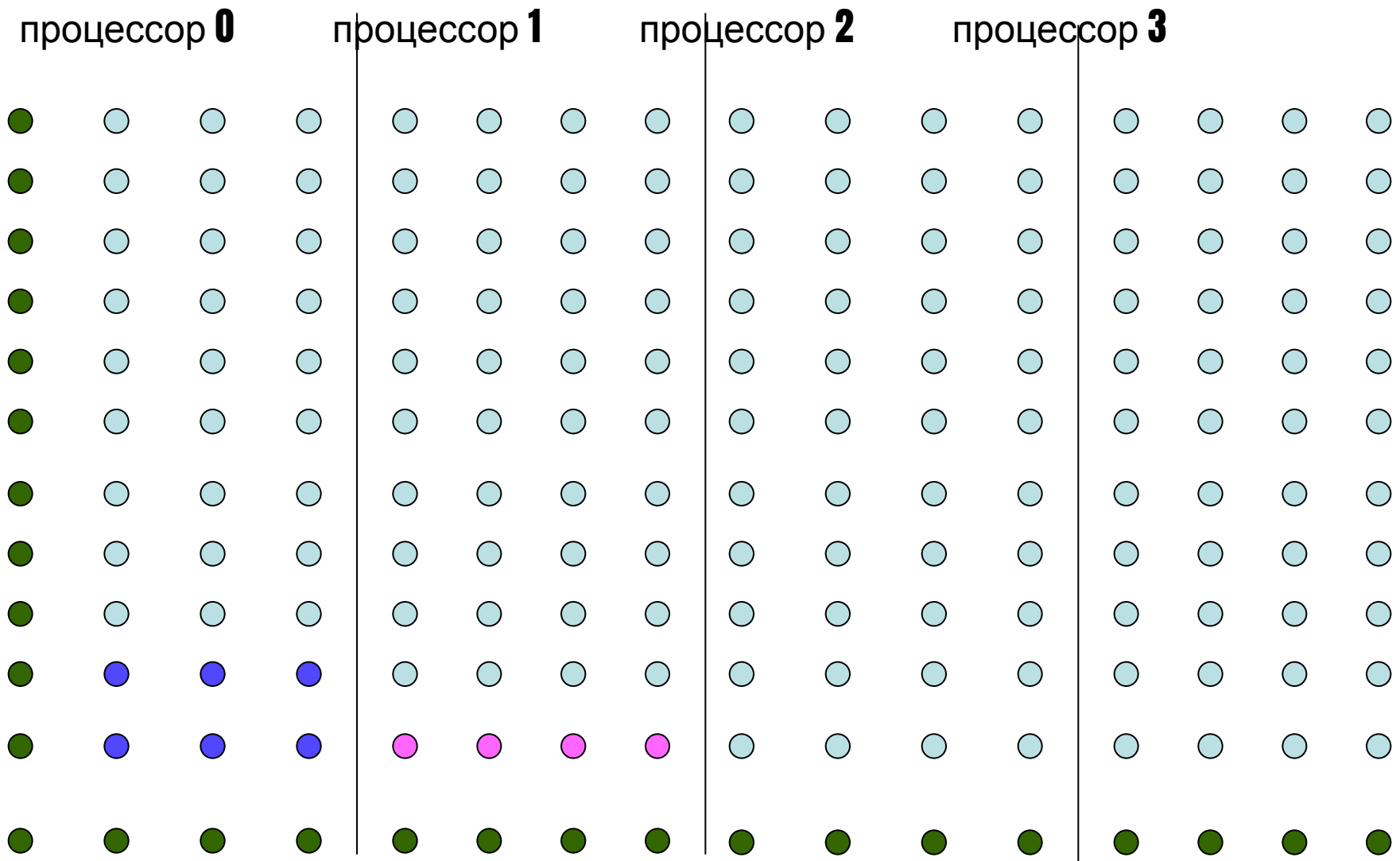


# Метод конвейерного параллелизма

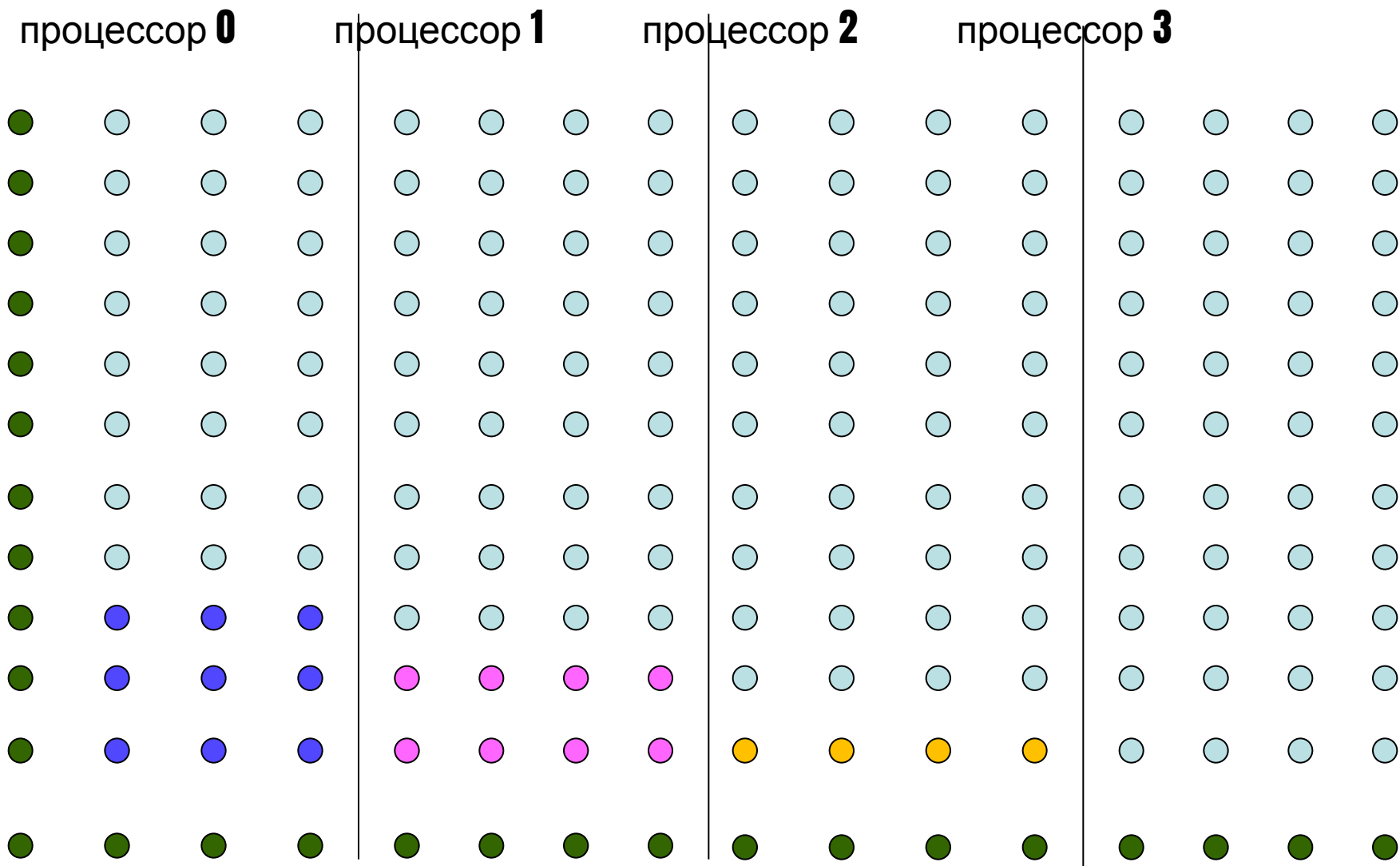




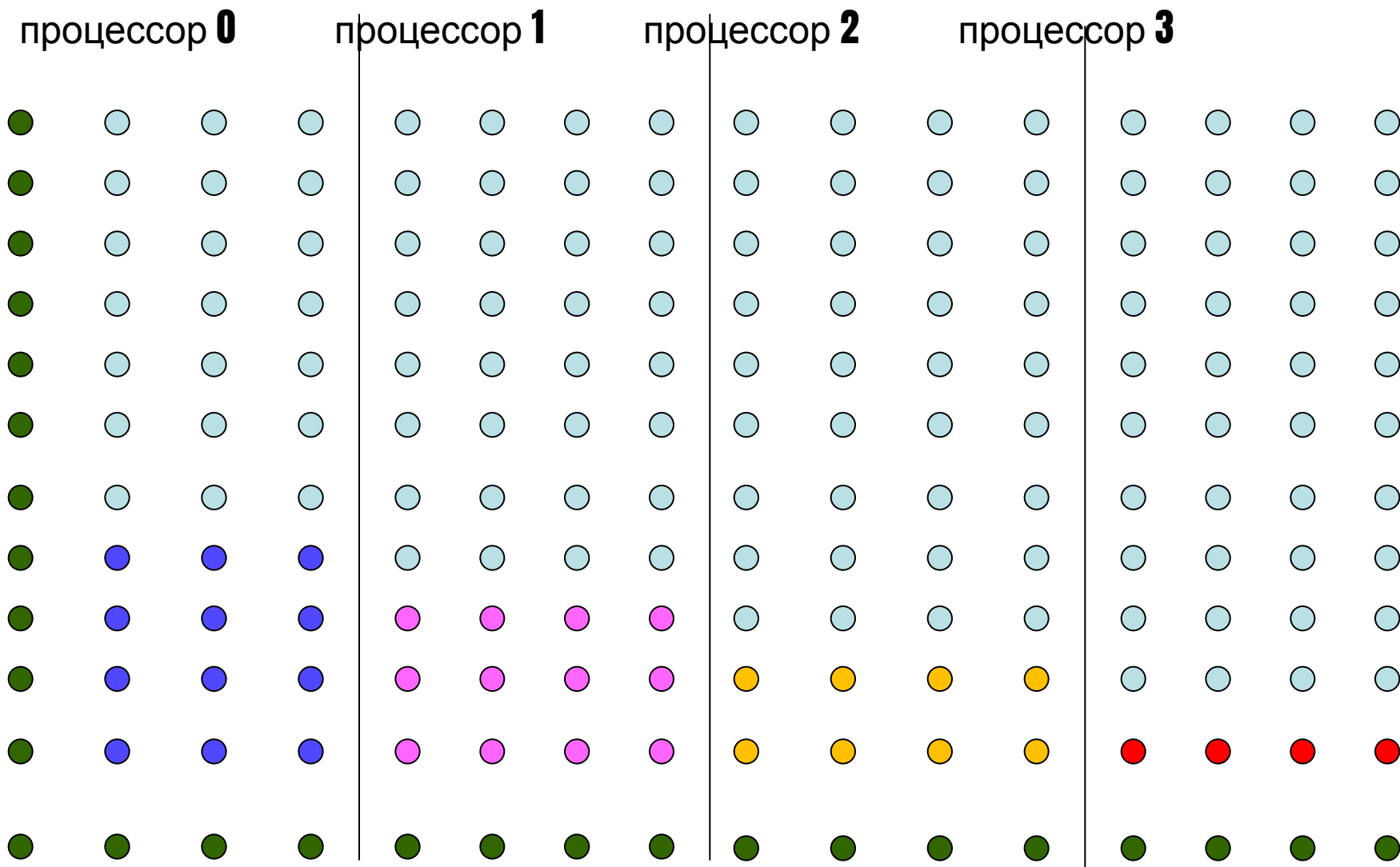
# Метод конвейерного параллелизма



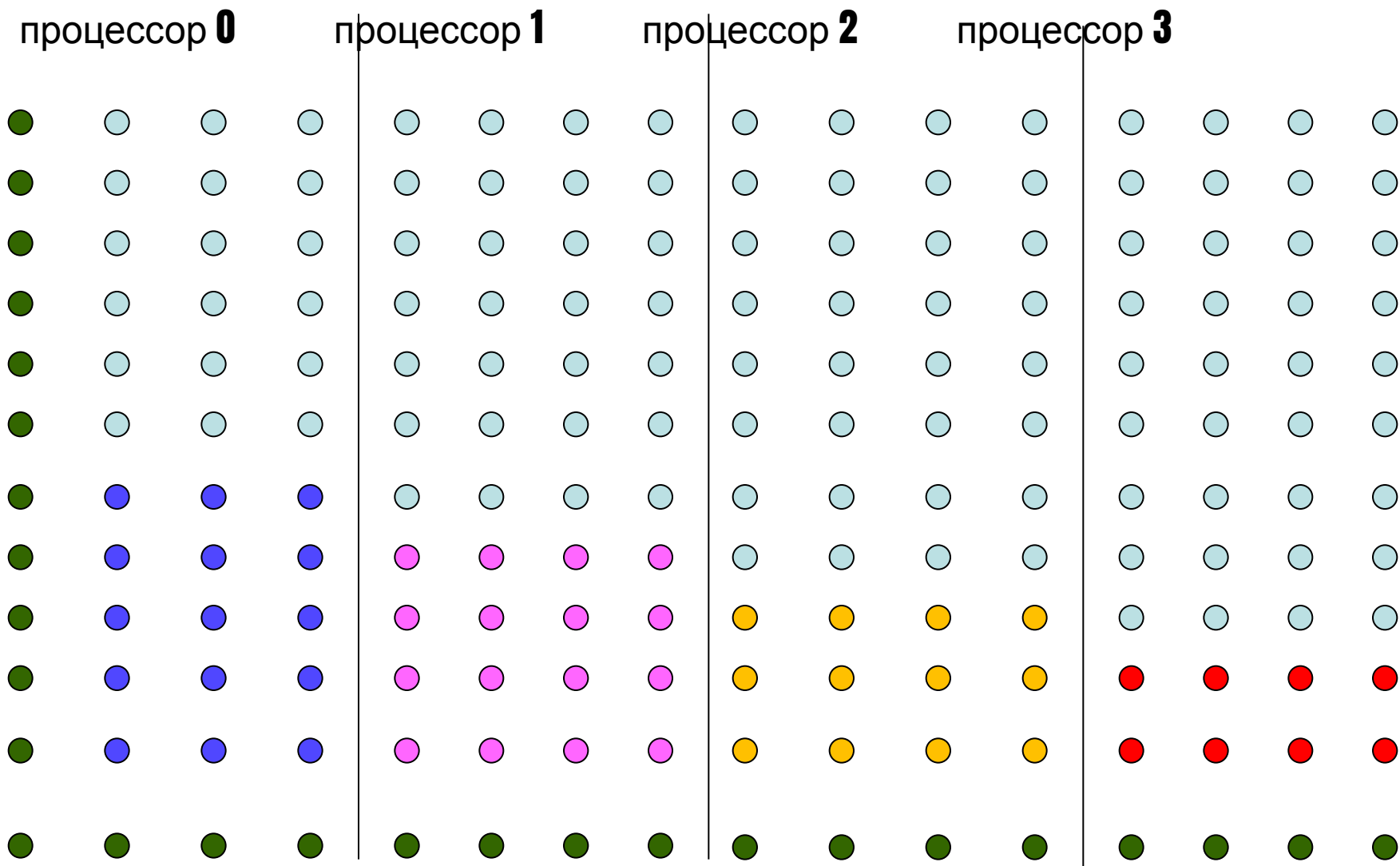
# Метод конвейерного параллелизма



# Метод конвейерного параллелизма

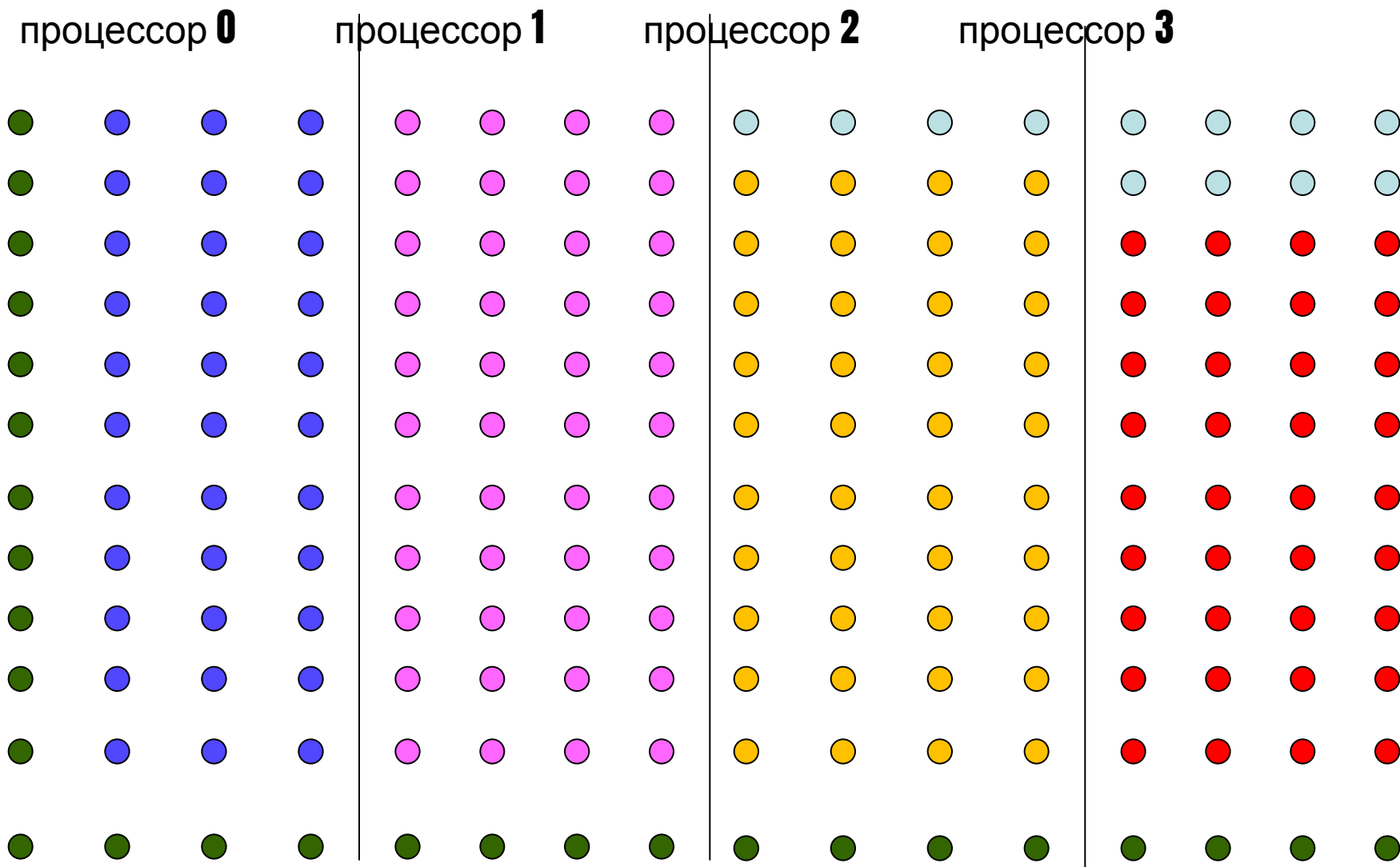


# Метод конвейерного параллелизма

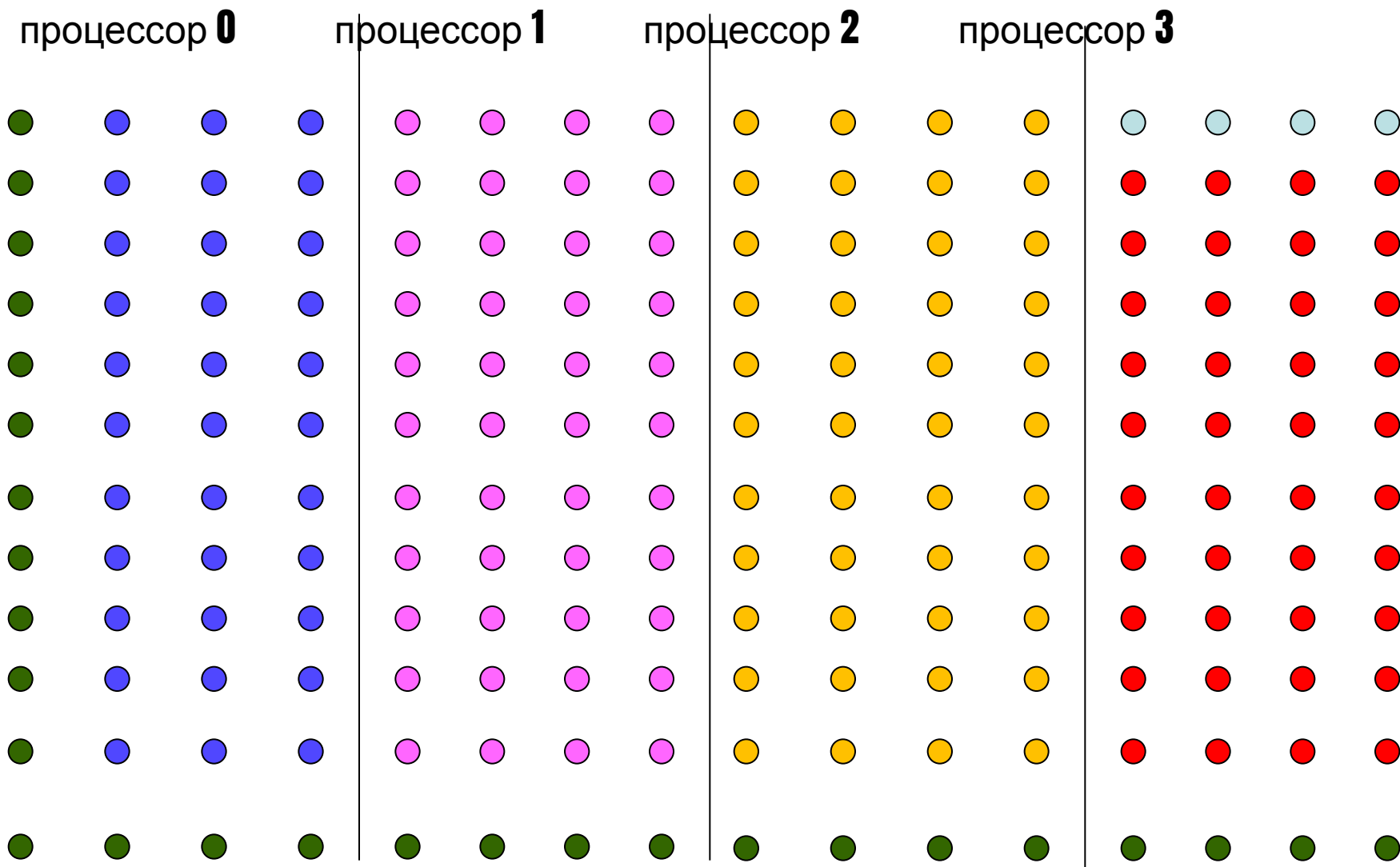




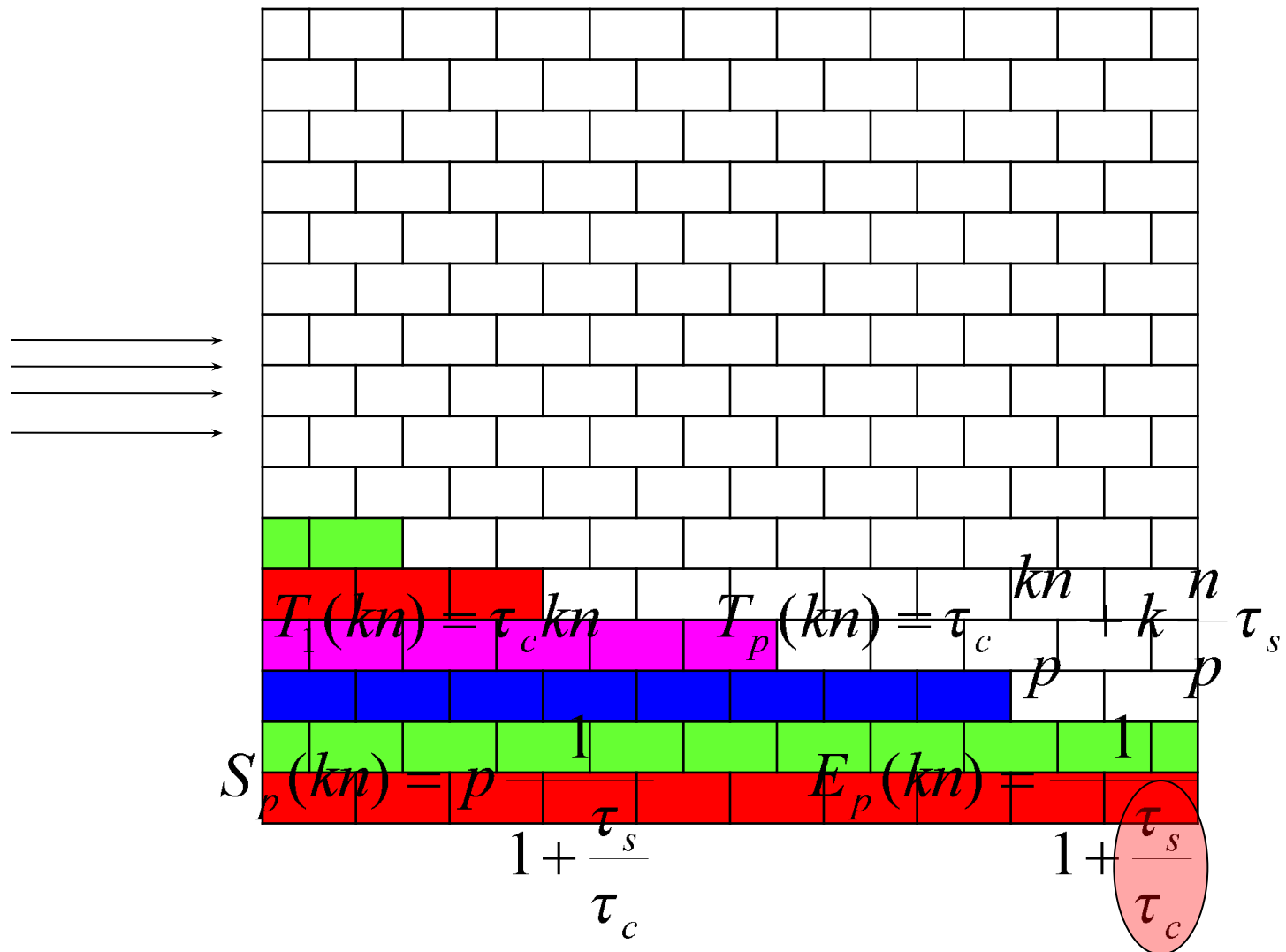
# Метод конвейерного параллелизма



# Метод конвейерного параллелизма

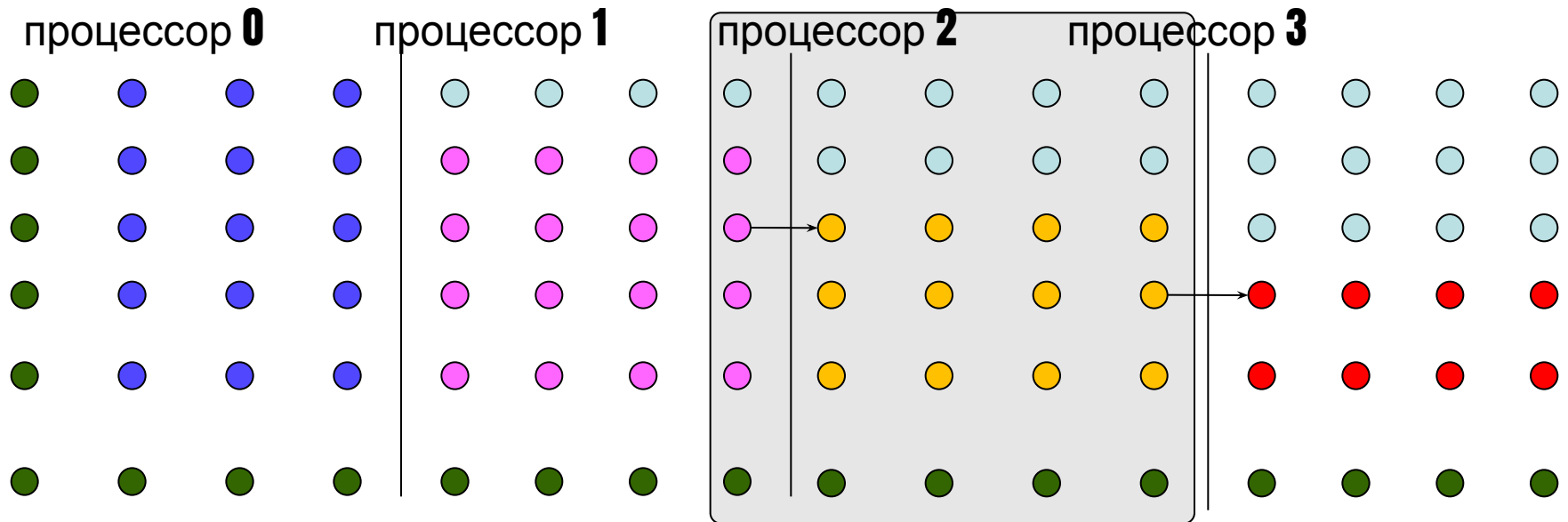


# Метод конвейерного параллелизма





# Метод конвейейрного параллелизма



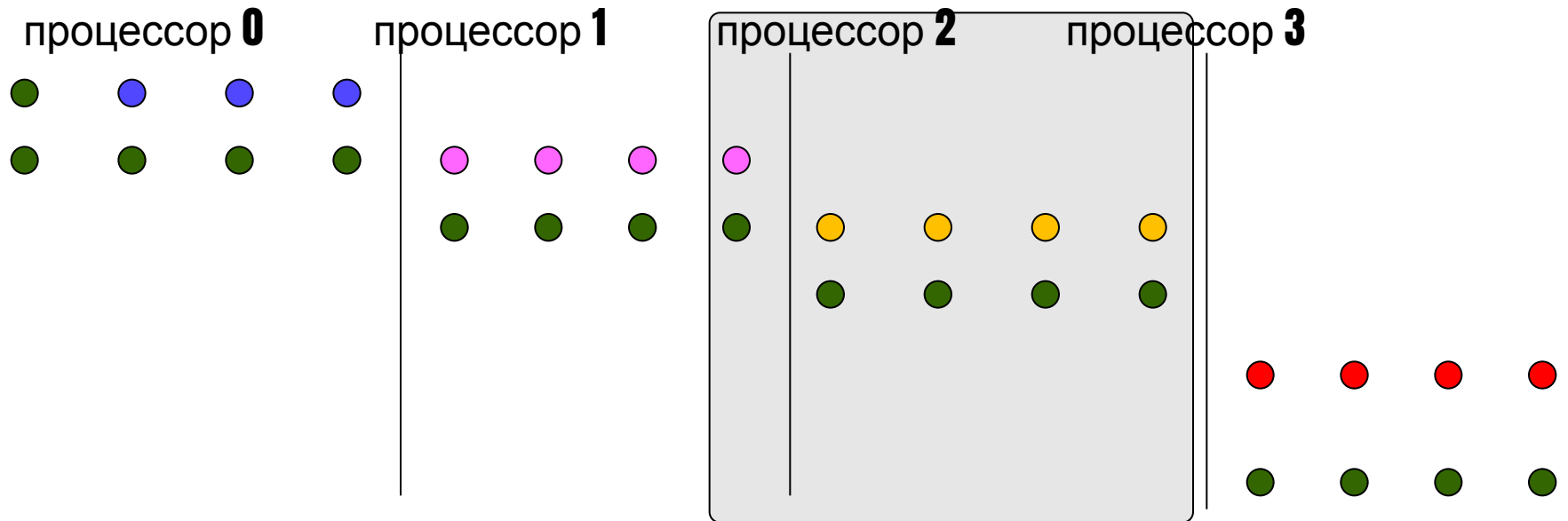
$$T_1(kn) = \tau_c kn$$

$$T_p(kn) = \tau_c \frac{kn}{p} + 2k\tau_s$$

$$S_p(kn) = p \frac{1}{1 + 2 \frac{p \tau_s}{n \tau_c}}$$

$$E_p(kn) = \frac{1}{1 + 2 \frac{p \tau_s}{n \tau_c}}$$

# Объём хранимых данных



$$T_1(kn) = \tau_c kn$$

$$T_p(kn) = \tau_c \frac{kn}{p} + 2k\tau_s$$

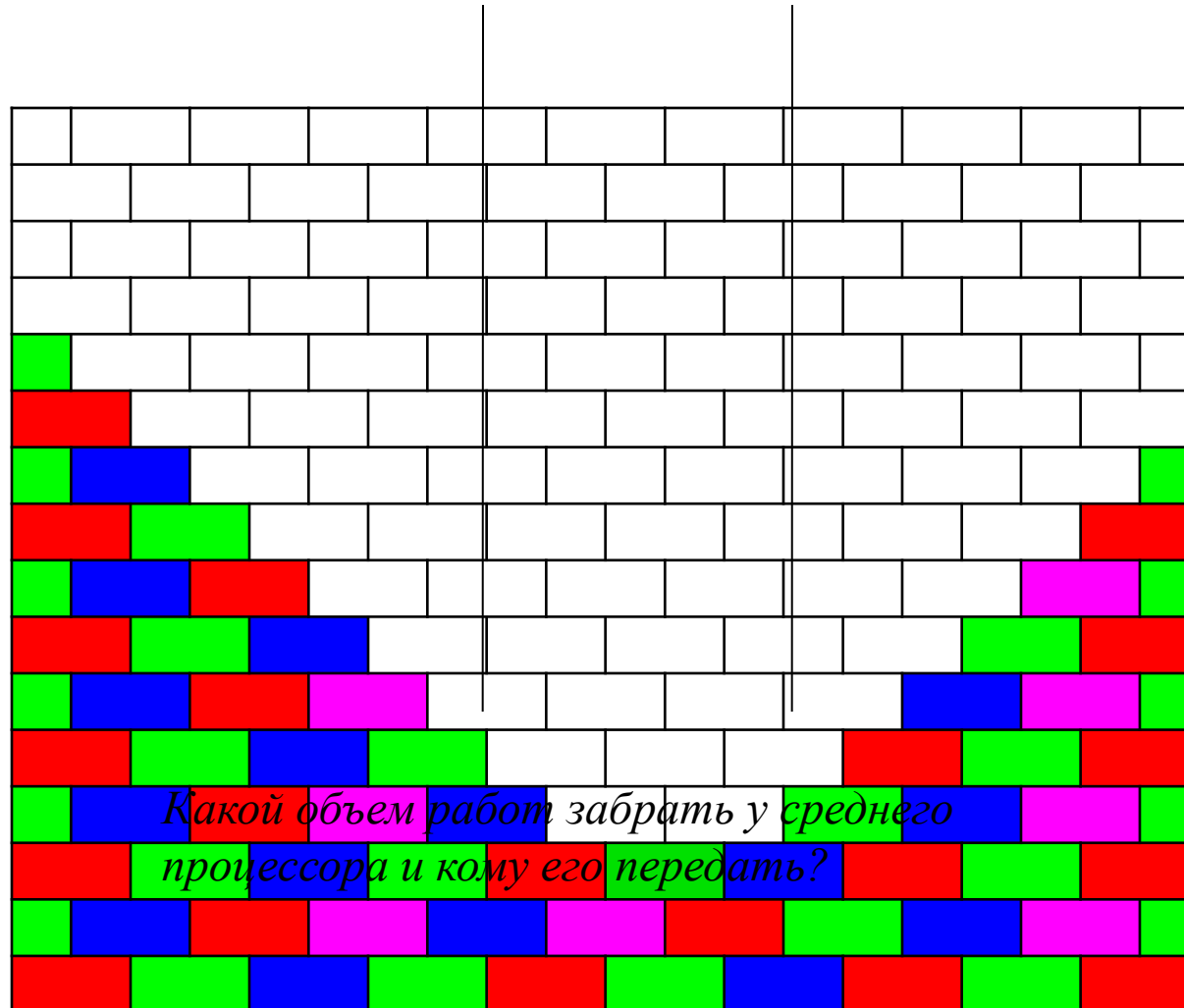
$$S_p(kn) = p \frac{1}{1 + 2 \frac{p \tau_s}{n \tau_c}}$$

$$E_p(kn) = \frac{1}{1 + 2 \frac{p \tau_s}{n \tau_c}}$$

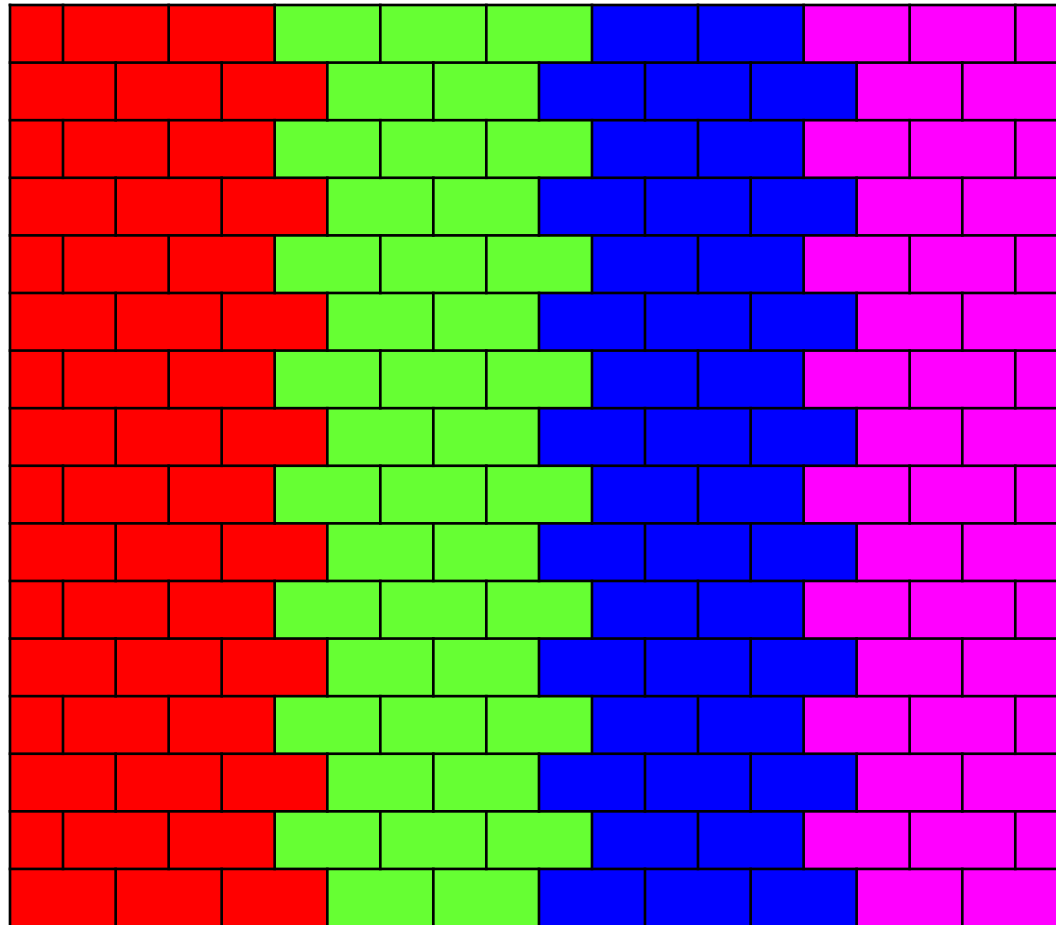
# Диффузная балансировка

- Причины дисбаланса вычислительной нагрузки
  - Разные процессоры
  - Внешнее воздействие
  - Разная вычислительная сложность заданий
  
- Результат дисбаланса
  - Эффективная производительность определяется самым медленным процессором

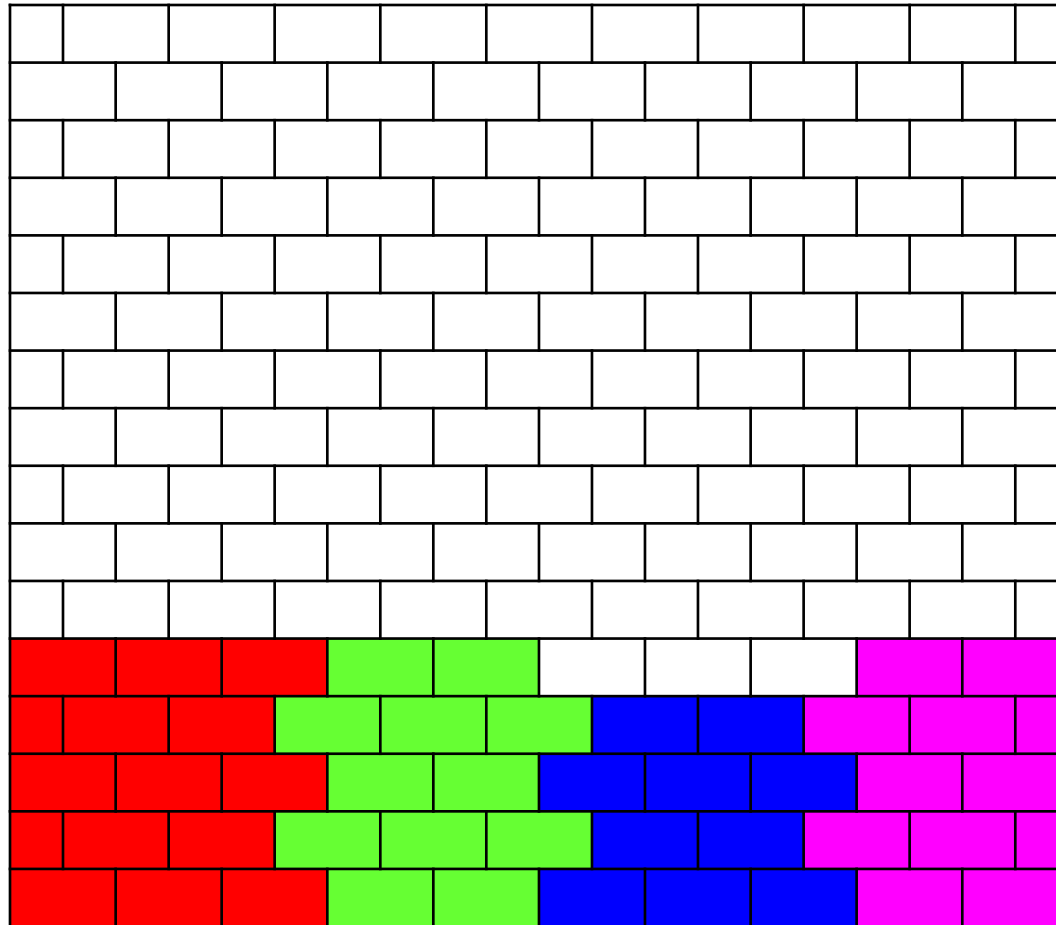
# Медленный процессор



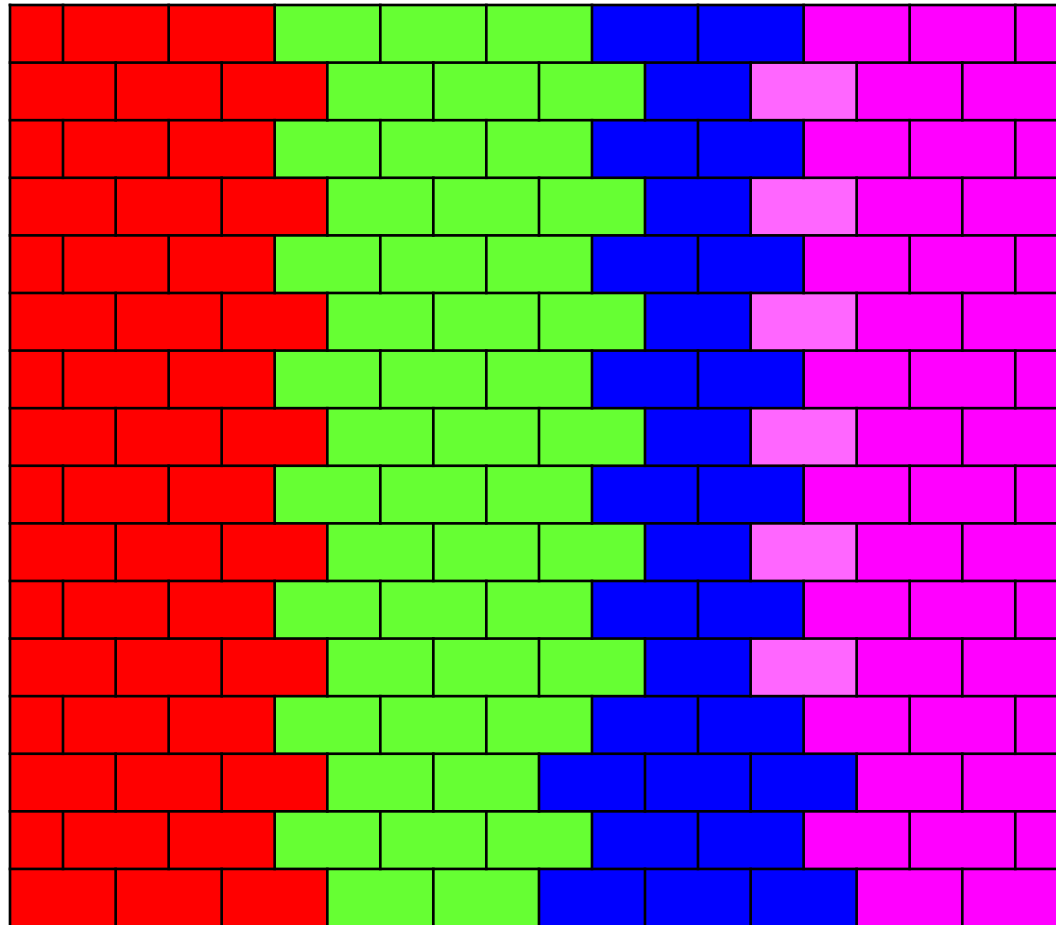
# Метод геометрического параллелизма



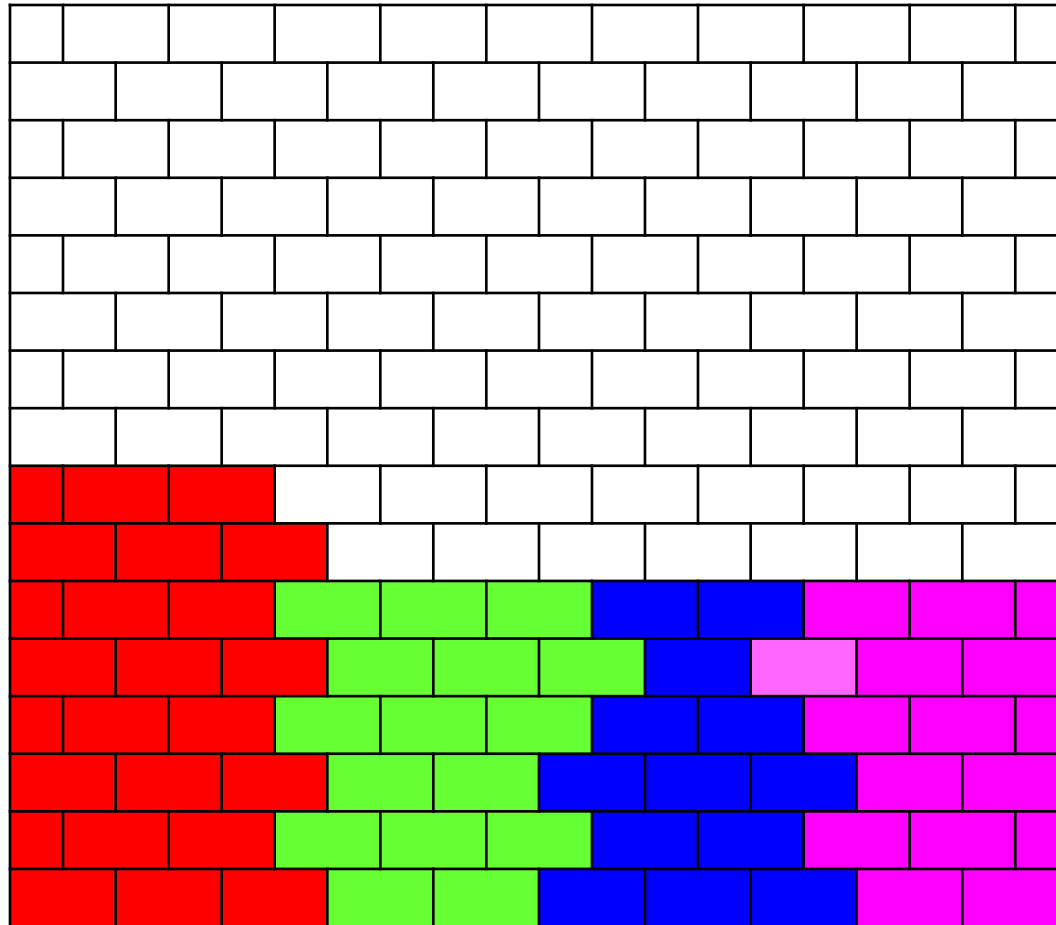
# Метод геометрического параллелизма



# Диффузная балансировка загрузки

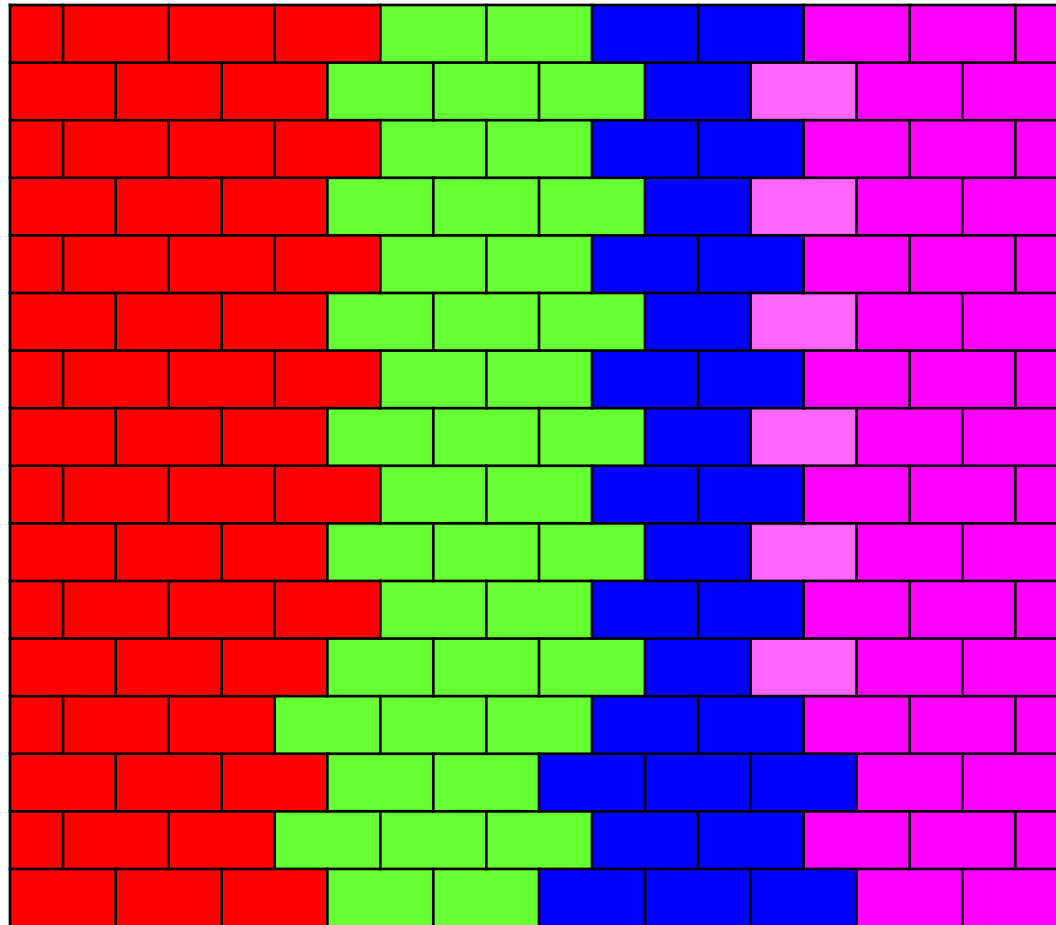


# Диффузная балансировка загрузки

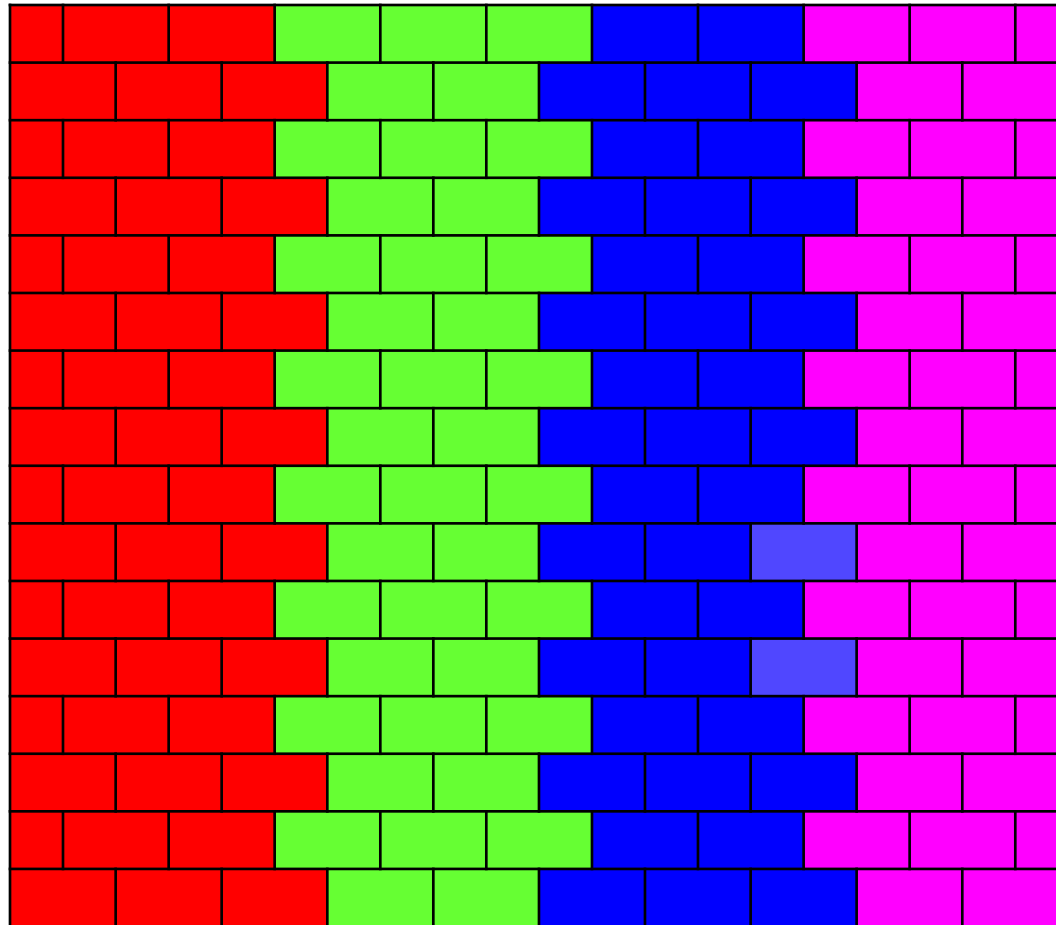




# Диффузная балансировка загрузки



# Статическое распределение



# Постановка задачи диффузной балансировки

Дано:

- Количество точек –  $N$
- Количество процессоров –  $p$
- Процессор  $i$  обработал  $n_i$  точек за время  $t_i$

Требуется:

- Найти количества точек  $n'_i$ , которое следует обработать процессорам на следующем шаге
- Определить сколько точек каждый из процессоров должен передать соседним процессорам

# Диффузная балансировка

$$n'_i = N \frac{\frac{n_i}{t_i}}{\sum_{j=0}^{p-1} \frac{n_j}{t_j}}$$

# Резюме

---

- Статическая и динамическая балансировка загрузки процессоров
  - Статическая балансировка
    - метод сдваивания
    - геометрический параллелизм
    - конвейерный параллелизм
  - Динамическая балансировка
    - коллективное решение
    - диффузная балансировка

# Общая схема вычислений

```
K = 1 000 000;
```

```
шаг_вывода = 10 000;
```

```
for (шаг=0; шаг<k; шаг++)
```

```
{
```

```
    for (кирпич=rank*n/p; кирпич<(rank+1)*n/p; кирпич++)
```

```
        Уложить (кирпич)
```

```
    Обменяться данными о точках, прилегающих к внутренним  
    границам()
```

```
    if( шаг % шаг_вывода == 0 )
```

```
    {
```

```
        Вывести промежуточные результаты()
```

```
    }
```

```
}
```

# Определение суммы двух многоразрядных чисел

```
r=0;  
for (i=0; i<=n; i++)  
  {  
    d=a[i]+b[i]+r;  
    c[i]=d%10;  
    r=d/10;  
  }  
c[i]=r;
```

```
+ 6934317835  
  3221643577  
-----  
10155961412
```

$$T_1 = 4n\tau_c$$

# «Параллельный» алгоритм

Последовательное распространение разряда переноса на четырёх процессорах

$$\begin{array}{r} + 999999999 \\ \phantom{+} 1 \\ \hline 1000000000 \end{array}$$

99	99	99	99
			1
			100
		100	
	100		
100			
100	00	00	00



# « Параллельный » алгоритм

---



# Спекулятивный алгоритм

## ❑ Спекулятивное вычисление двух сумм

$$\begin{array}{r} + 999999999 \\ \phantom{+} \phantom{999999999} 1 \\ \hline 1000000000 \end{array}$$

99	99	99	99	
			1	
99	99	99	100	+0
100	100	100		+1
100	00	00	00	

# Спекулятивный алгоритм

```
r1=0;
r2=1;
for (i=0; i<=n1; i++)
{
  d1=a[i]+b[i]+r1;
  c1[i]=d1%10;
  r1=d1/10;
  d2=a[i]+b[i]+r2;
  c2[i]=d2%10;
  r2=d2/10;
}
Recv(&r)
if (r) c=c1;
else c=c2;
```

$$T' = \delta n_1 \tau_c$$

# Спекулятивный алгоритм

## □ Спекулятивное вычисление двух сумм

$$T_1 = 4n\tau_c$$

$$T_p = 8 \frac{n}{p} \tau_c$$

$$S_p = \frac{p}{2}$$

$$E_p = 50\%$$

99	99	99	99	
			1	
99	99	99	100	+0
100	100	100		+1
100	00	00	00	

# Заключение

---

- ❑ Рассмотрены некоторые методы построения параллельных алгоритмов
- ❑ Рассмотрен алгоритм диффузной балансировки загрузки процессоров
- ❑ Представлен масштабируемый параллельный алгоритм, основанный на неэффективном последовательном алгоритме

# Контакты

**Якобовский М.В.**

проф., д.ф.-м.н.,

зав. сектором

«Программного обеспечения многопроцессорных систем и вычислительных сетей»

Института прикладной математики им. М.В.

Келдыша Российской академии наук

mail: [mail: lira@imamod.ru](mailto:lira@imamod.ru)

web: <http://lira.imamod.ru>