

академия
больших
данных



mail.ru
group

Высокопроизводительные вычисления. Лекция 5. MPI

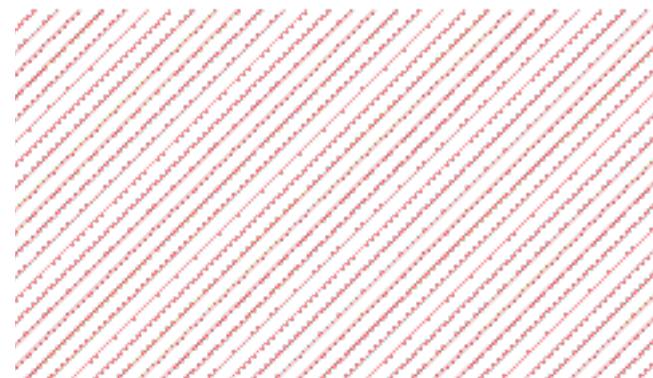
Рыкованов Сергей

доцент Сколковского института науки и технологий



Матвеев Сергей

научный сотрудник Сколковского института науки и технологий



0. Разница между разными библиотеками

Системы с распределенной памятью



+ компьютерная сеть

Системы с распределенной памятью



+ компьютерная сеть

пересылка сообщений между узлами:
БИБЛИОТЕКА MPI

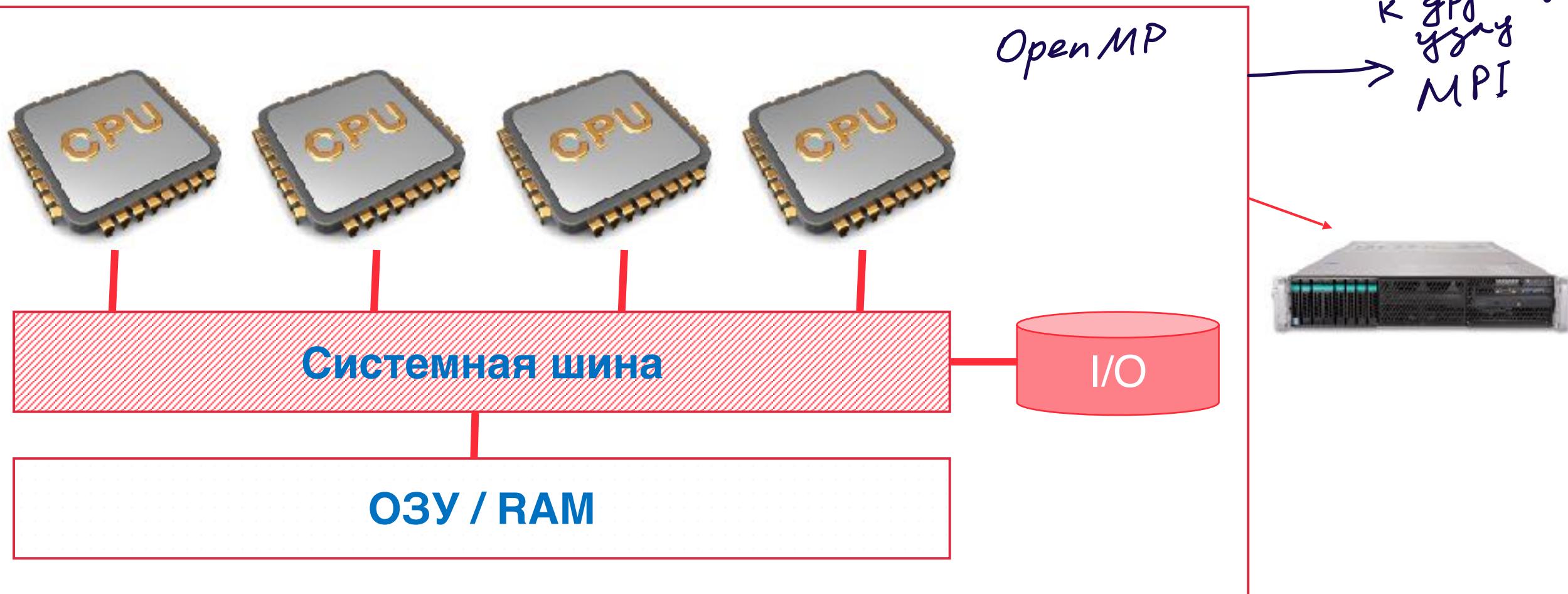
GPU узел:

- CUDA
- OpenCL
- HIP
- Open Acc

СРУ узел:

- Open MP
- Open Acc
- pthreads

Системы с общей памятью

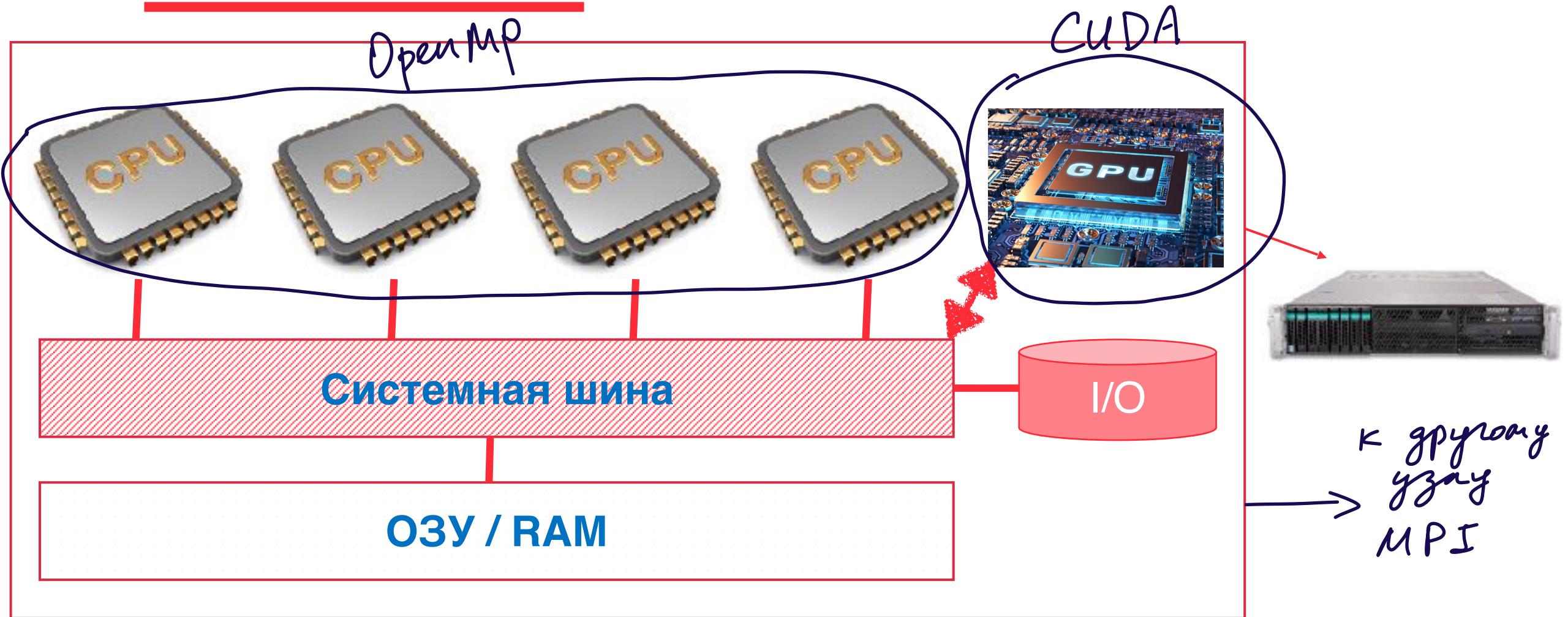


* теоретически можно и внутри узла MPI, но

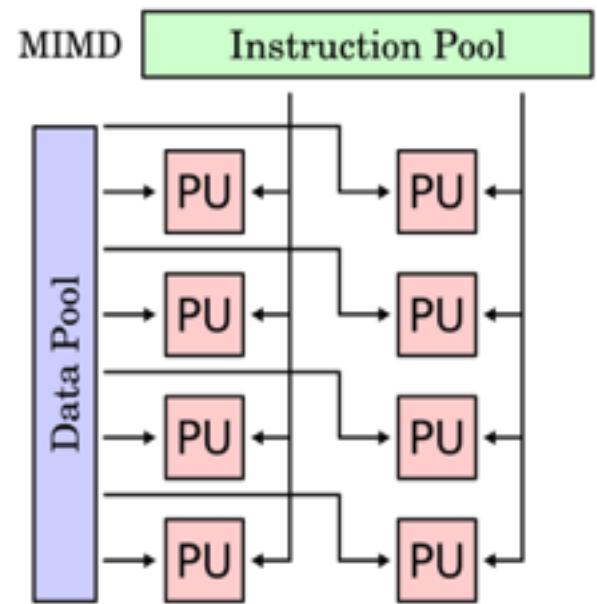
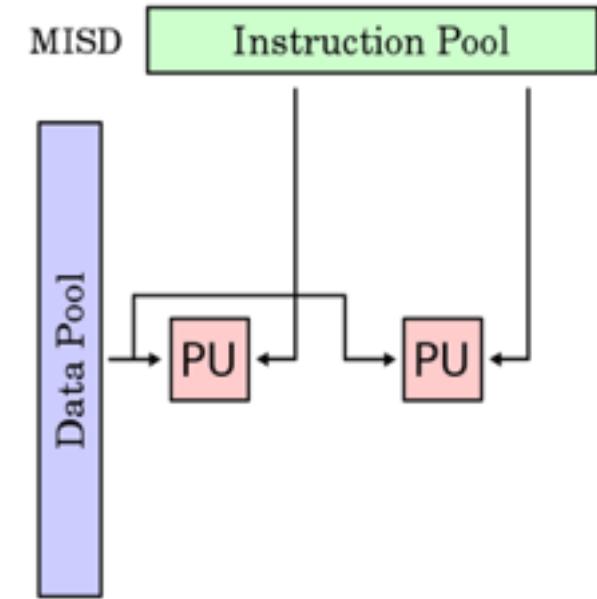
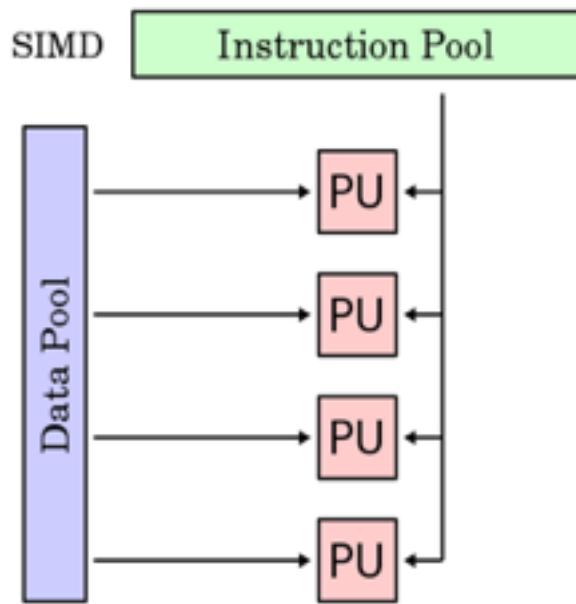
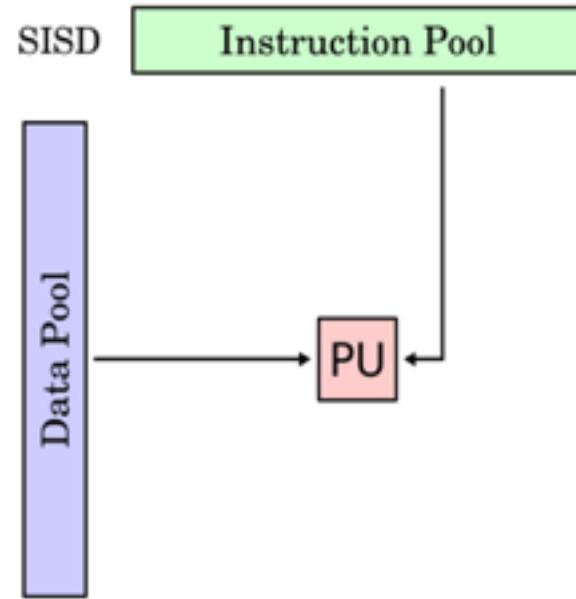
Системы с общей памятью + GPU



Системы с общей памятью + GPU



Классификация Флинна

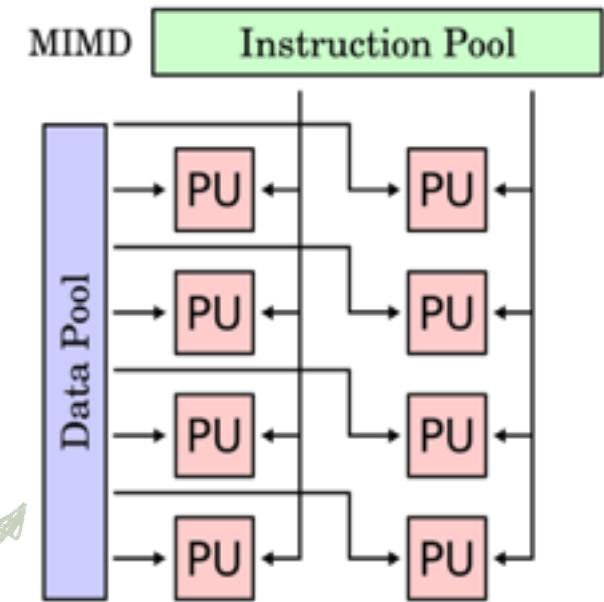
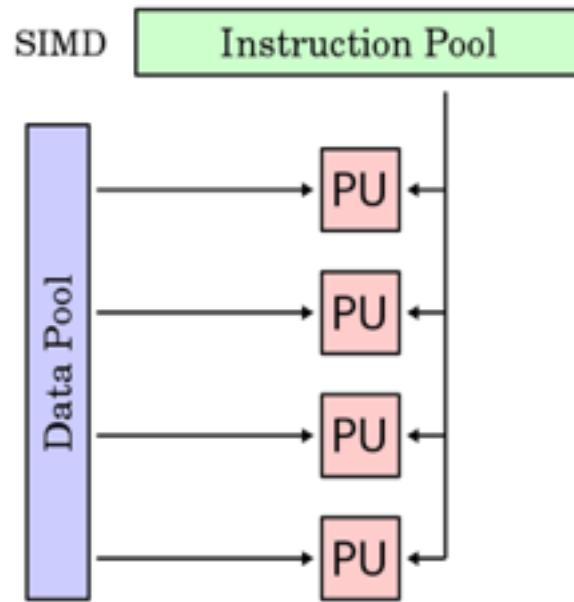
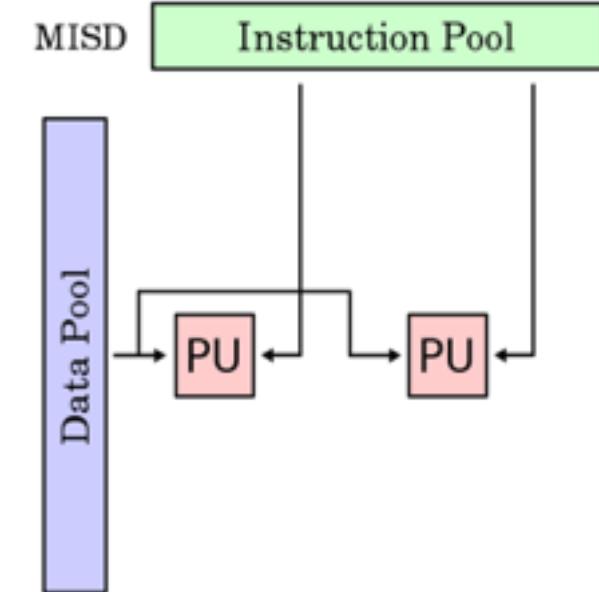
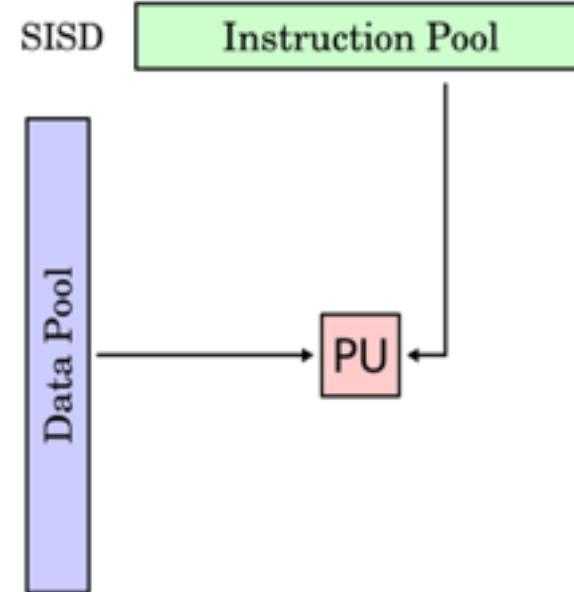


Классификация Флинна

"OLD"

CPU

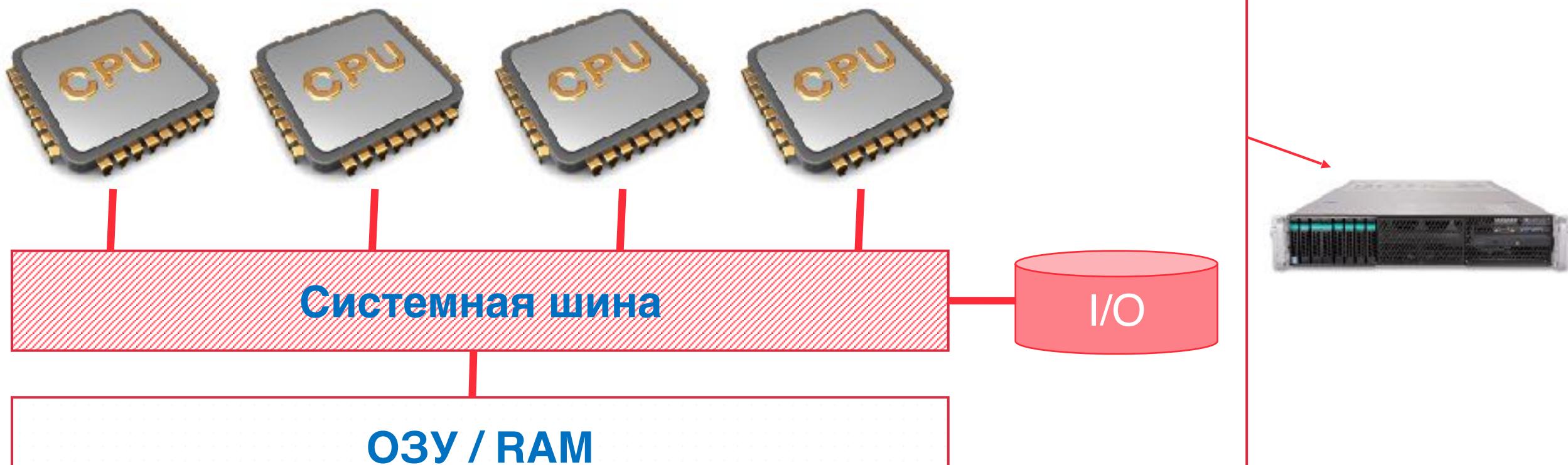
GPU



MODERN CPU

1. Краткое напоминание о системах с общей памятью. Fork-join

Системы с общей памятью



Processes and threads



Thread



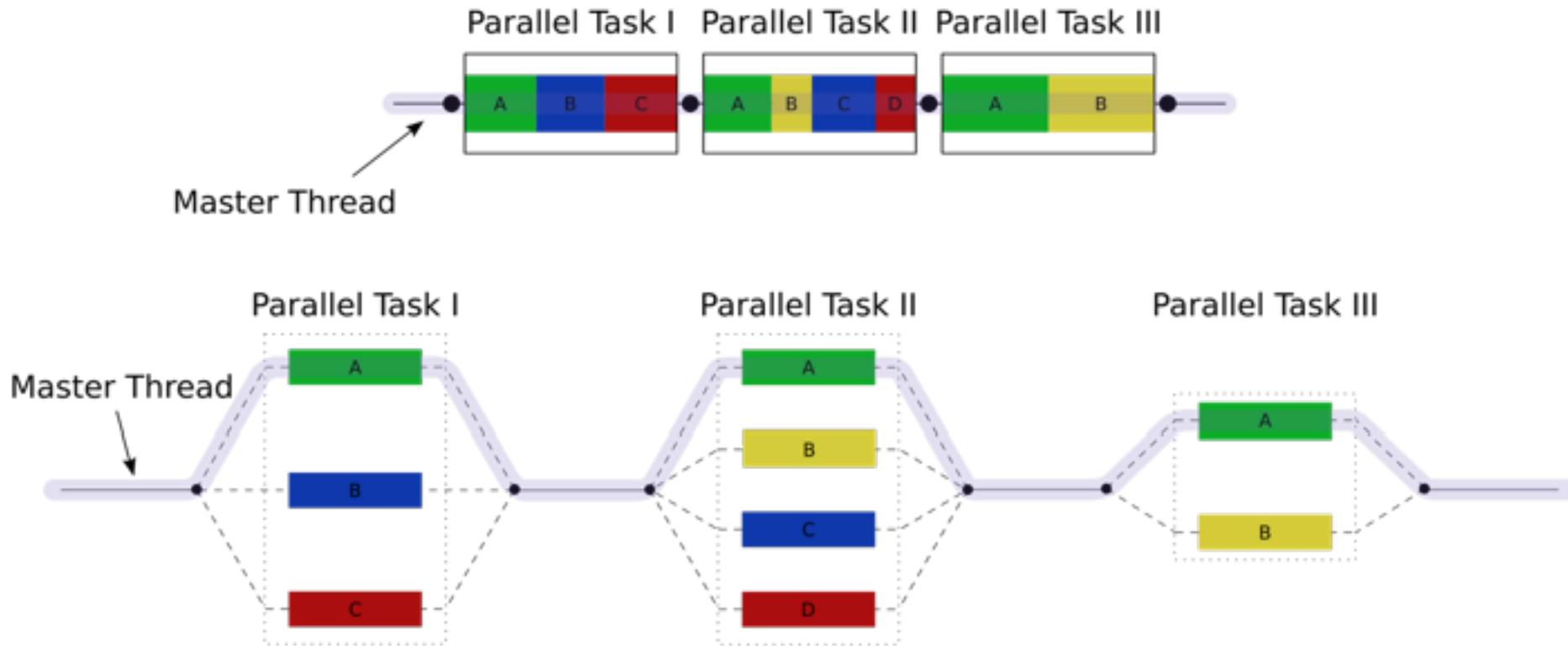
Thread



Address space

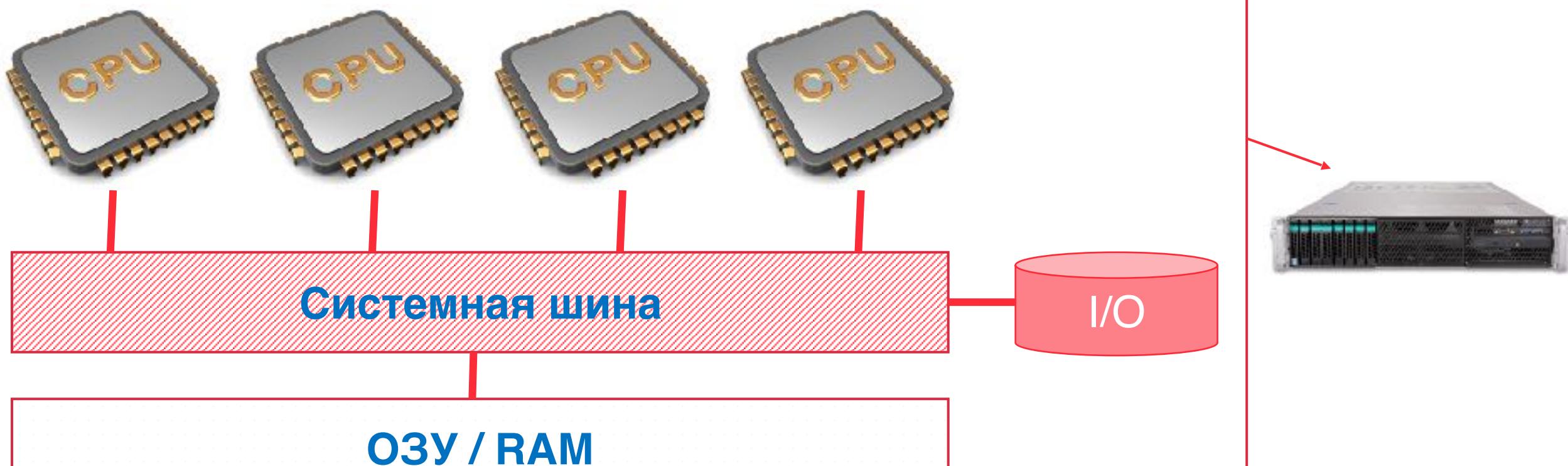


Fork-join concept



2. Системы с распределенной памятью. Message Passing

Системы с общей памятью

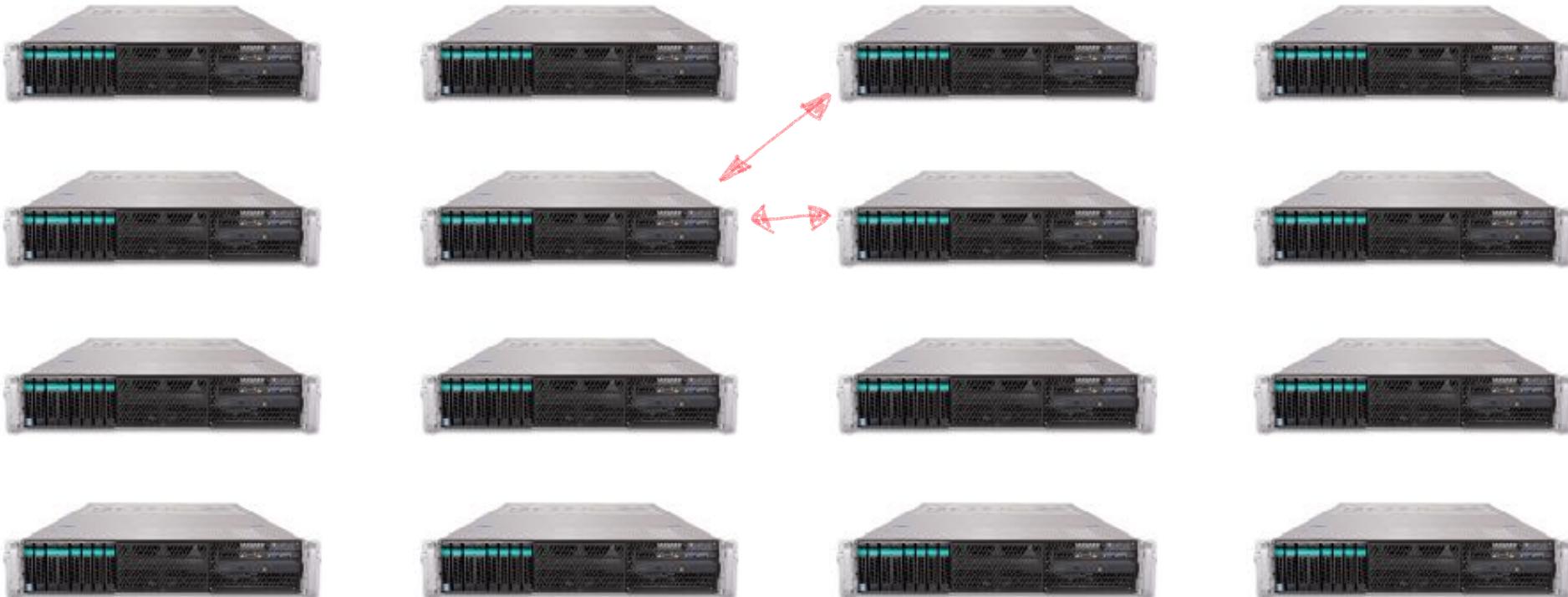


Системы с распределенной памятью



+ компьютерная сеть

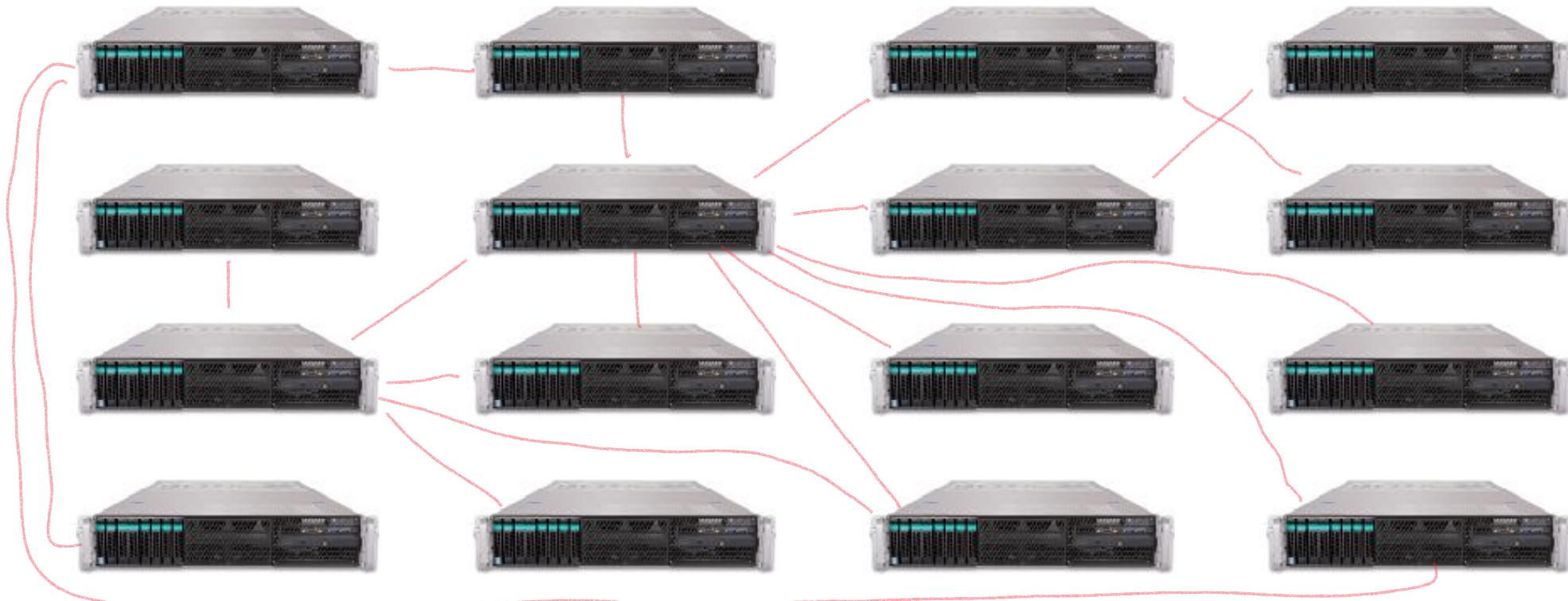
Системы с распределенной памятью



+ компьютерная сеть

Системы с распределенной памятью

Не хватает мощности: покупай больше серверов



+ компьютерная сеть

Системы с распределенной памятью

Не хватает мощности: покупай больше серверов



+ компьютерная сеть

3. Библиотека MPI (Message Passing Interface)

Message Passing Interface (MPI)

- Используется с 1992 года, когда было несколько нестандартизированных библиотек для передачи сообщений между узлами кластера.
- Более 40 компаний участвуют в разработке MPI
- Задача: сделать так, чтобы программы работали на различных архитектурах без потери производительности
- Разные версии (более или менее совместимые):
MPICH, OpenMPI, IntelMPI (коммерческая)
- Более 120 команд, но используются в основном около 10
- Написан для C/C++ и Fortran
- Обертки для Python, Julia, OCaml и других языков

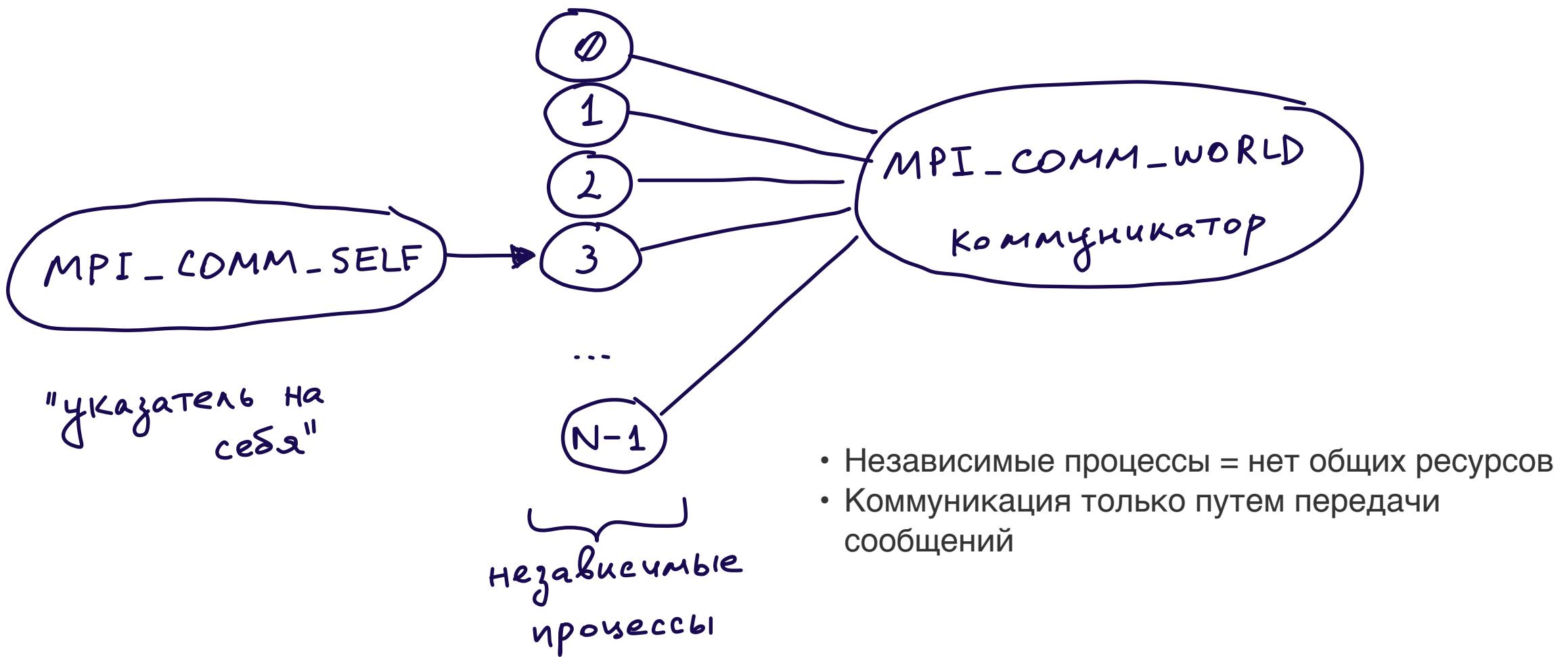


www.open-mpi.org



mpitutorial.com

Message Passing Interface (MPI)





Message Passing Interface (MPI)

Все идентификаторы

Имена всех:

функций
типов данных
констант

MPI_

```
#include "mpi.h"  
....  
MPI_Init(&argc, &argv);  
....  
MPI_Comm_rank(MPI_COMM_WORLD, &rank);  
MPI_Comm_size(MPI_COMM_WORLD, &size);  
....  
MPI_Finalize();
```

```
# include <mpi.h>
# include <stdio.h>
# include <stdlib.h>

int main(int argc, char ** argv)
{
    int psize;
    int prank;
    MPI_Status status;

    int ierr;

    ierr = MPI_Init(&argc, &argv);
    ierr = MPI_Comm_rank(MPI_COMM_WORLD, &prank);
    ierr = MPI_Comm_size(MPI_COMM_WORLD, &psize);

    if (prank == 0)
    {
        printf("The number of processes available is %d\n", psize);
    }

    printf("Hello world from process[%d]\n", prank);

    ierr = MPI_Finalize();
    return 0;
}
```

4. MPI коммуникация точка-точка

MPI. Коммуникация точка-точка (point-to-point)



Блокирующая пересылка

```
MPI_Send(  
    void* data,  
    int count,  
    MPI_Datatype datatype,  
    int destination,  
    int tag,  
    MPI_Comm communicator)
```

```
MPI_Recv(  
    void* data,  
    int count,  
    MPI_Datatype datatype,  
    int source,  
    int tag,  
    MPI_Comm communicator,  
    MPI_Status* status)
```

Типы данных MPI

| MPI datatype | C equivalent |
|------------------------|------------------------|
| MPI_SHORT | short int |
| MPI_INT | int |
| MPI_LONG | long int |
| MPI_LONG_LONG | long long int |
| MPI_UNSIGNED_CHAR | unsigned char |
| MPI_UNSIGNED_SHORT | unsigned short int |
| MPI_UNSIGNED | unsigned int |
| MPI_UNSIGNED_LONG | unsigned long int |
| MPI_UNSIGNED_LONG_LONG | unsigned long long int |
| MPI_FLOAT | float |
| MPI_DOUBLE | double |
| MPI_LONG_DOUBLE | long double |
| MPI_BYTE | char |

Пример MPI_Send / MPI_Recv

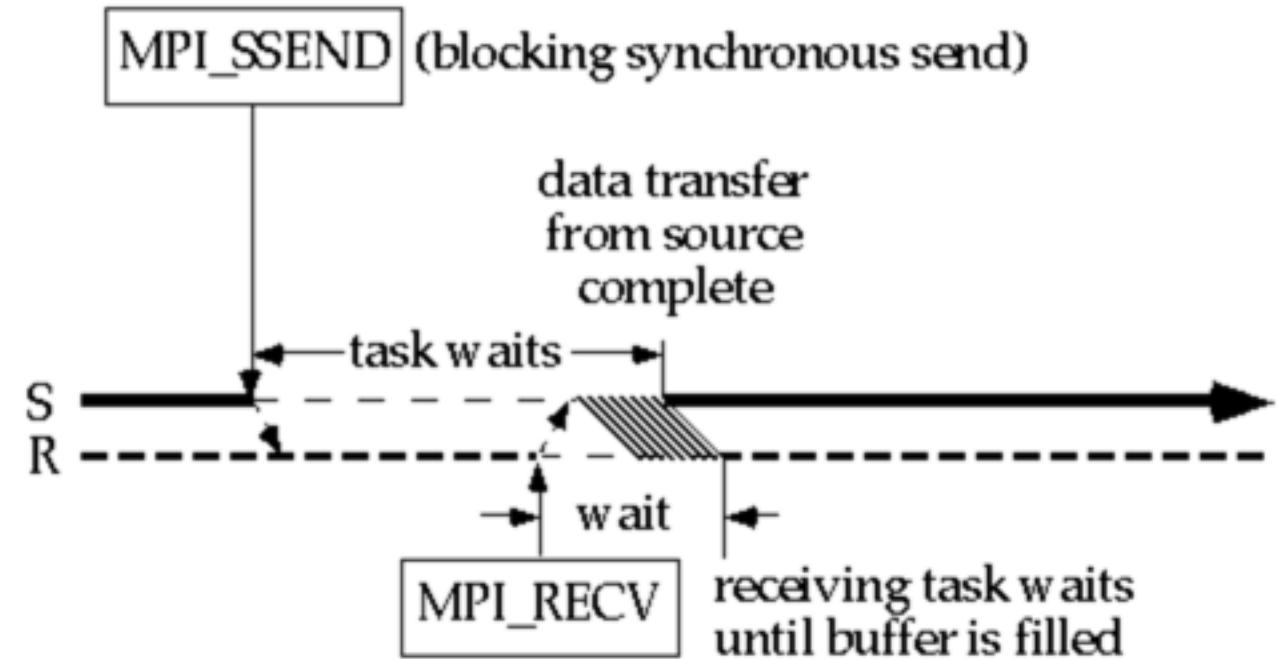
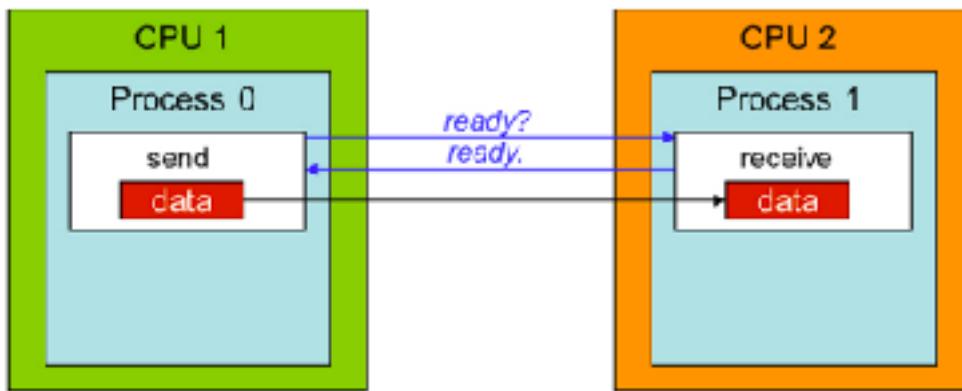
```
// Find out rank, size
int world_rank;
MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
int world_size;
MPI_Comm_size(MPI_COMM_WORLD, &world_size);

int number;
if (world_rank == 0) {
    number = -1;
    MPI_Send(&number, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
} else if (world_rank == 1) {
    MPI_Recv(&number, 1, MPI_INT, 0, 0, MPI_COMM_WORLD,
             MPI_STATUS_IGNORE);
    printf("Process 1 received number %d from process 0\n",
           number);
}
```

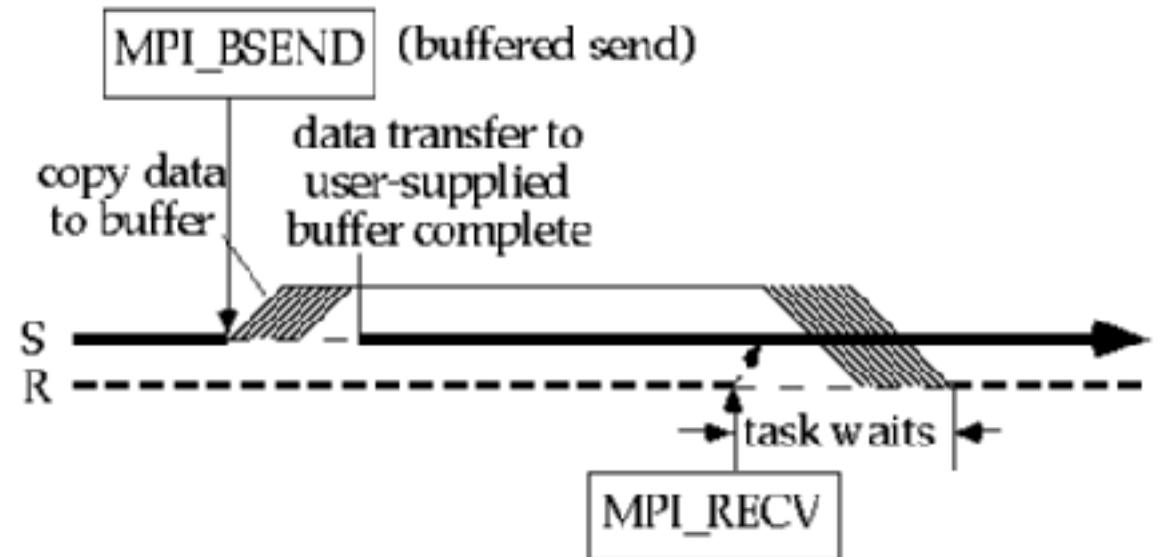
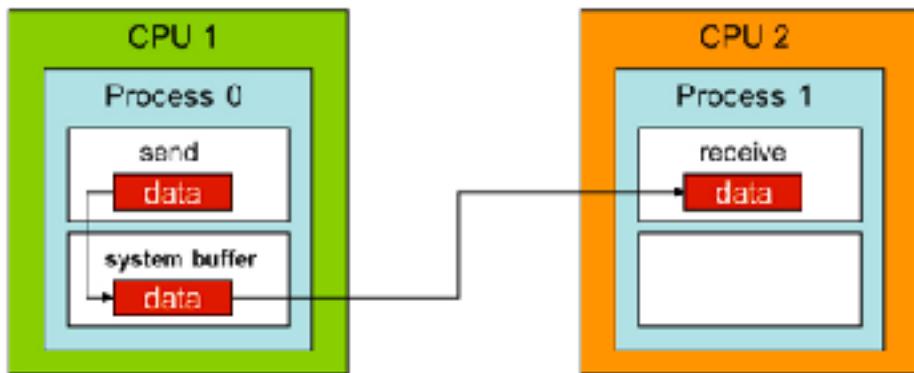
Пример MPI_Send / MPI_Recv

```
if(rank==0)
{
    MPI_Send(x to process 1)
    MPI_Recv(y from process 1)
}
if(rank==1)
{
    MPI_Send(y to process 0);
    MPI_Recv(x from process 0);
}
```

MPI Synchronous Send / Recv (SSend / Recv)



MPI Buffered Send / Recv (BSend / Recv)



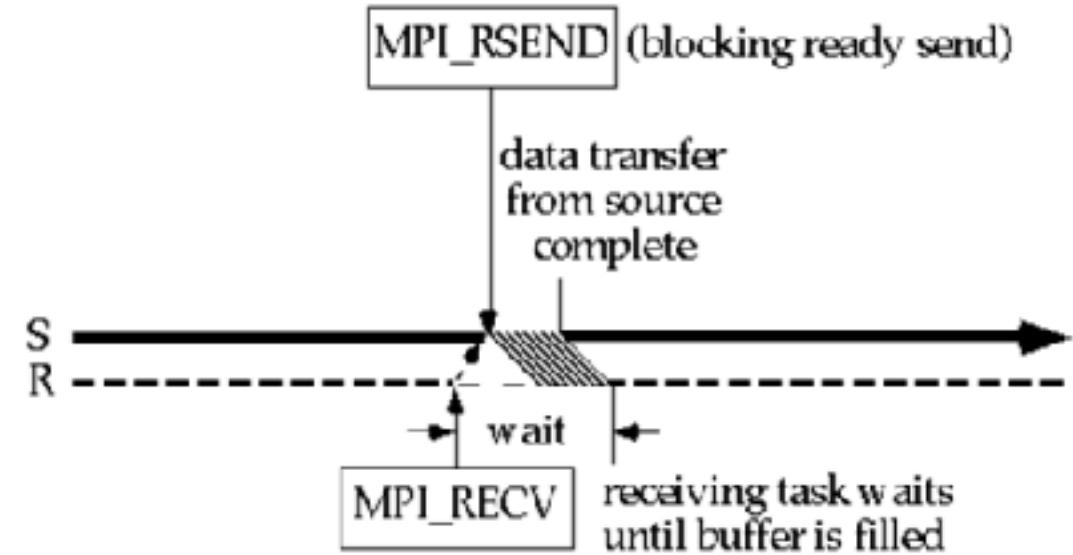
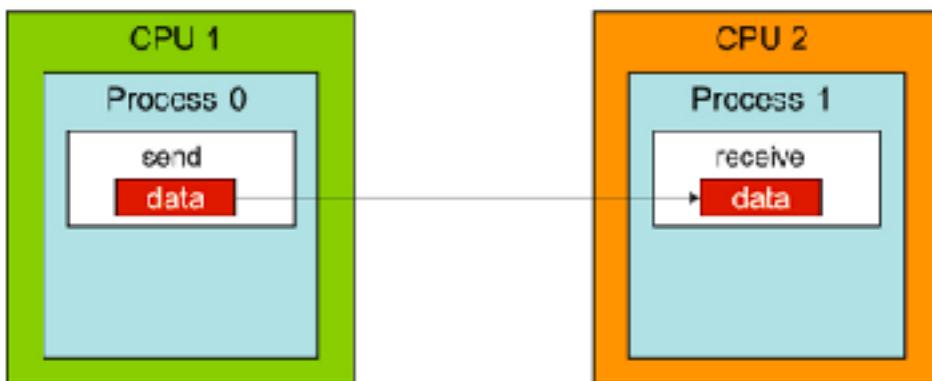
Плюсы:

- Нет рукопожатия - не надо ждать синхронизации
- Изначальные данные можно менять
- Время пересылки и получения может быть позже

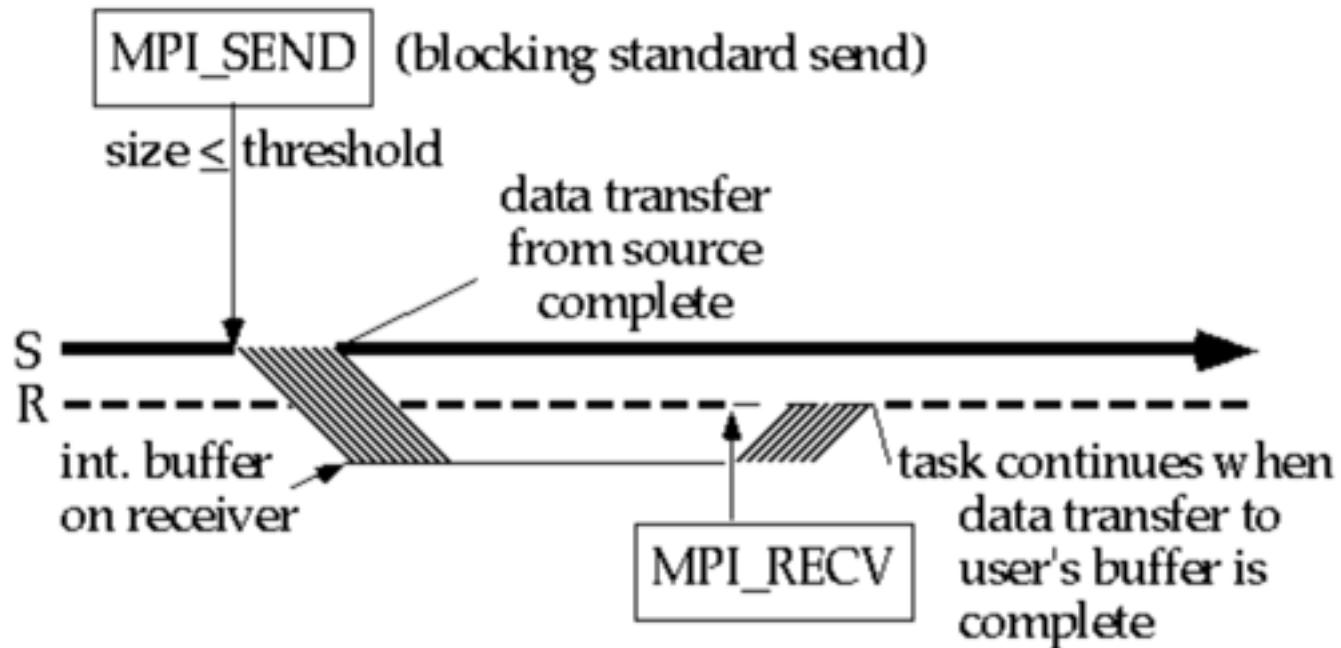
Минус: нужен доп. буфер

<https://cvw.cac.cornell.edu/MPIP2P/>

MPI Ready Send / Recv (RSend / Recv)



MPI Standard Send / Recv (Send / Recv)



MPI Standard Send / Recv (Send / Recv)

| Mode | Advantages | Disadvantages |
|-------------|--|---|
| Synchronous | <ul style="list-style-type: none">- Safest, therefore most portable- No need for extra buffer space- SEND/RECV order not critical | <ul style="list-style-type: none">- Can incur substantial synchronization overhead |
| Ready | <ul style="list-style-type: none">- Lowest total overhead- No need for extra buffer space- SEND/RECV handshake not required | <ul style="list-style-type: none">- RECV <i>must</i> precede SEND |
| Buffered | <ul style="list-style-type: none">- Decouples SEND from RECV- no sync overhead on SEND- Programmer can control size of buffer space- SEND/RECV order irrelevant | <ul style="list-style-type: none">- Copying to buffer incurs additional system overhead |
| Standard | <ul style="list-style-type: none">- Good for many cases- Compromise position | <ul style="list-style-type: none">- Protocol is determined by MPI implementation |

MPI non-blocking P2P communication

```
MPI_Status status; —  
MPI_Request request;  
  
MPI_Isend(  
    &count, 1, MPI_INT, dest, prank, MPI_COMM_WORLD, &request  
);  
  
MPI_Irecv(  
    &count, 1, MPI_INT, source, source, MPI_COMM_WORLD, &request  
);
```



Testing whether the message has arrived:

```
int MPI_Wait(MPI_Request *request, MPI_Status *status)  
  
int MPI_Test(MPI_Request *request, int *flag, MPI_Status *status)
```

5. MPI коллективные коммуникации

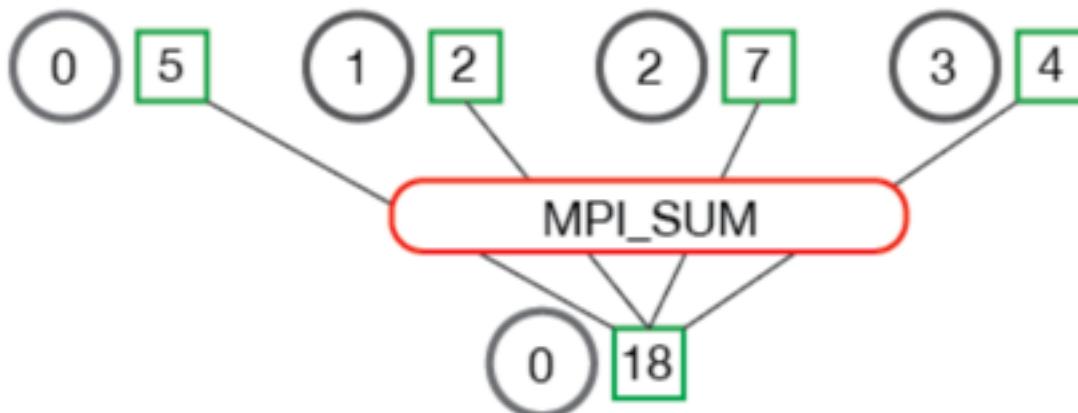
MPI collective communication. Reduce

```
MPI_Reduce(  
    void* send_data,  
    void* recv_data,  
    int count,  
    MPI_Datatype datatype,  
    MPI_Op op,  
    int root,  
    MPI_Comm communicator)
```

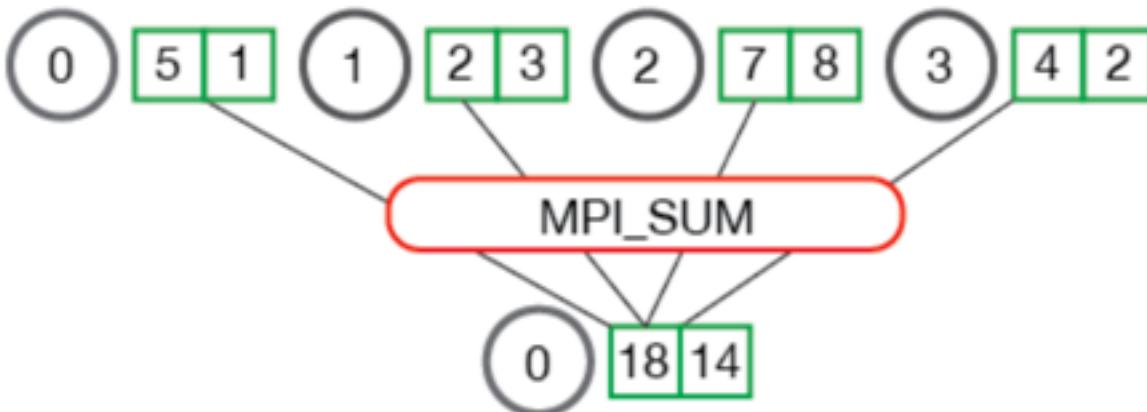
- **MPI_MAX** - Returns the maximum element.
- **MPI_MIN** - Returns the minimum element.
- **MPI_SUM** - Sums the elements.
- **MPI_PROD** - Multiplies all elements.
- **MPI_LAND** - Performs a logical *and* across the elements.
- **MPI_LOR** - Performs a logical *or* across the elements.
- **MPI_BAND** - Performs a bitwise *and* across the bits of the elements.
- **MPI_BOR** - Performs a bitwise *or* across the bits of the elements.
- **MPI_MAXLOC** - Returns the maximum value and the rank of the process that owns it.
- **MPI_MINLOC** - Returns the minimum value and the rank of the process that owns it.

MPI collective communication. Reduce

MPI_Reduce



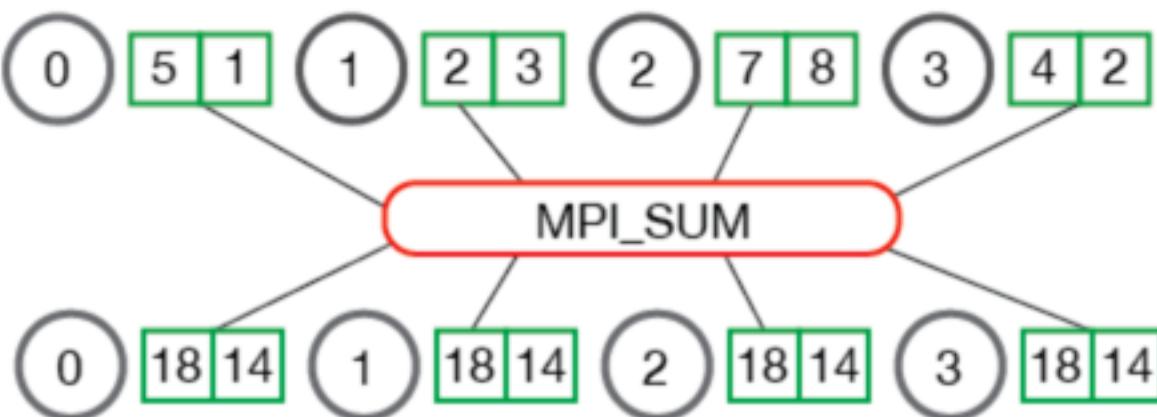
MPI_Reduce



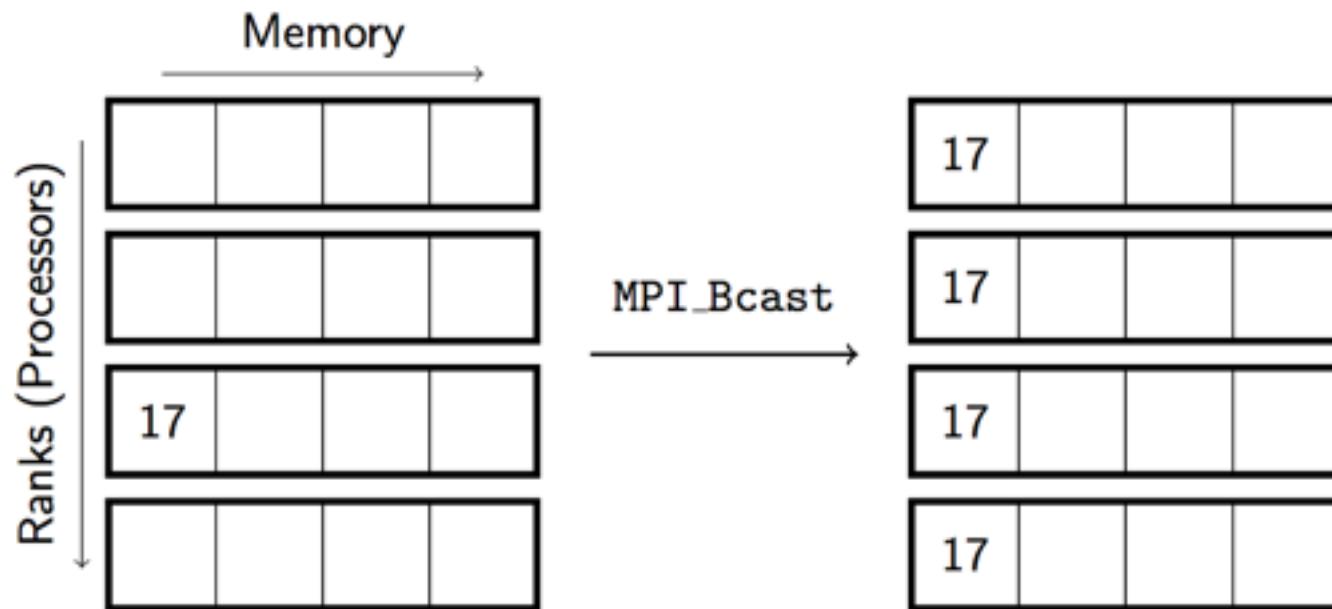
MPI collective communication. Allreduce

```
MPI_Allreduce(  
    void* send_data,  
    void* recv_data,  
    int count,  
    MPI_Datatype datatype,  
    MPI_Op op,  
    MPI_Comm communicator)
```

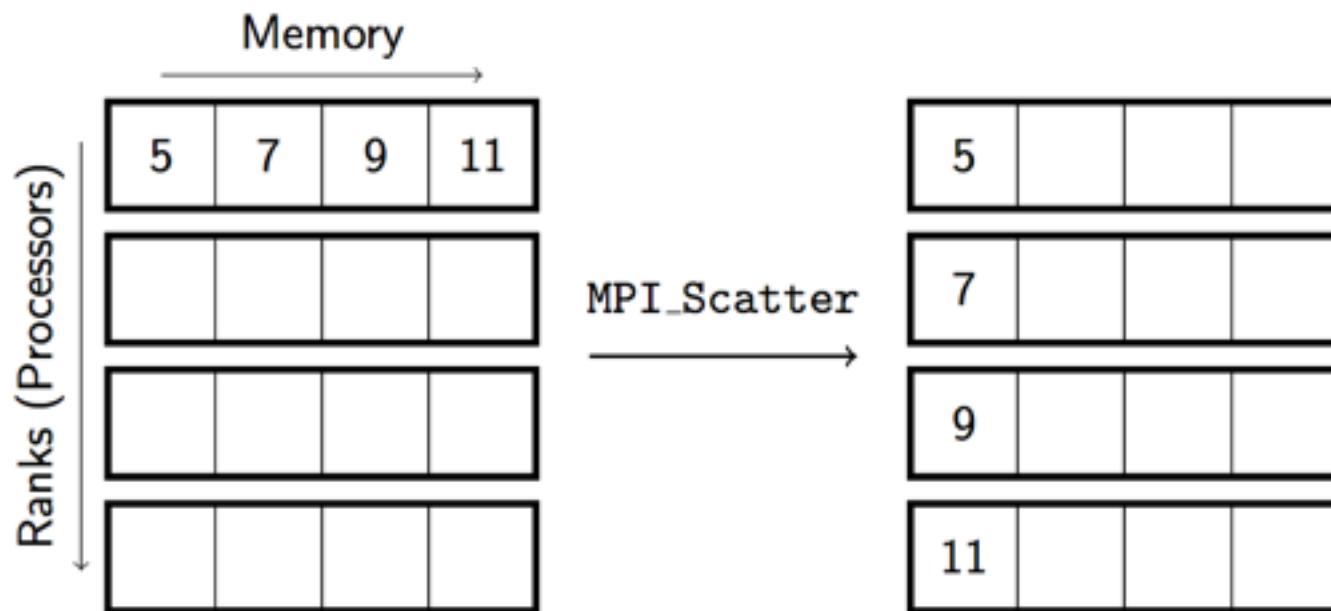
MPI_Allreduce



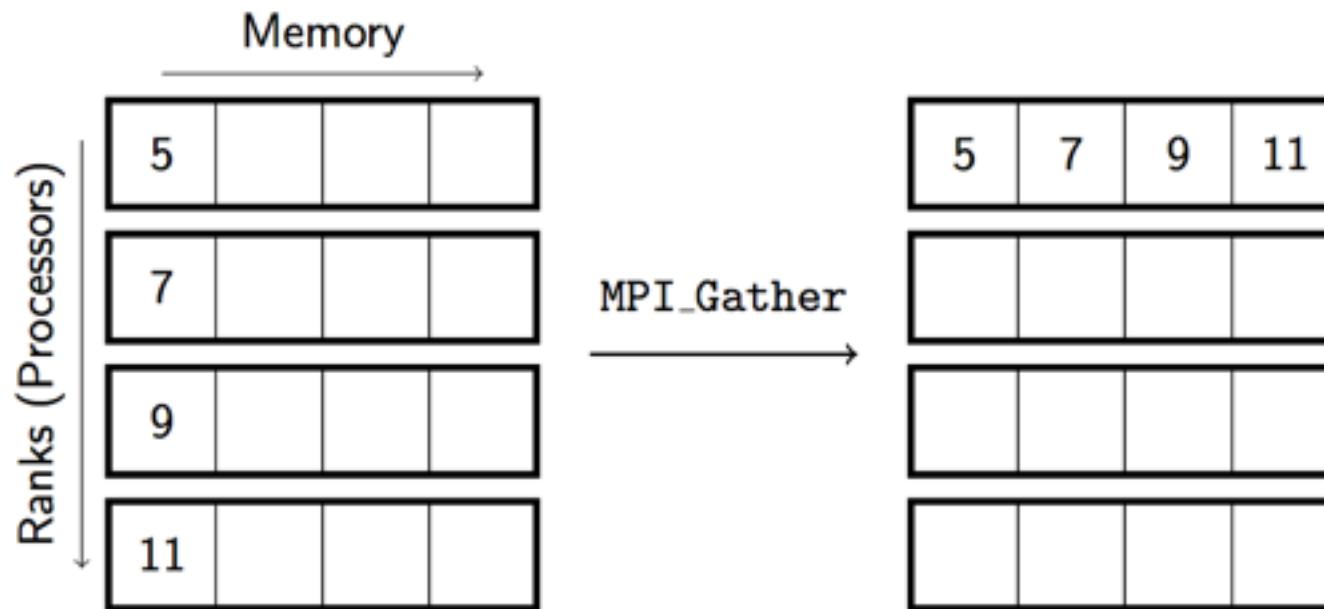
MPI collective communication. Broadcast



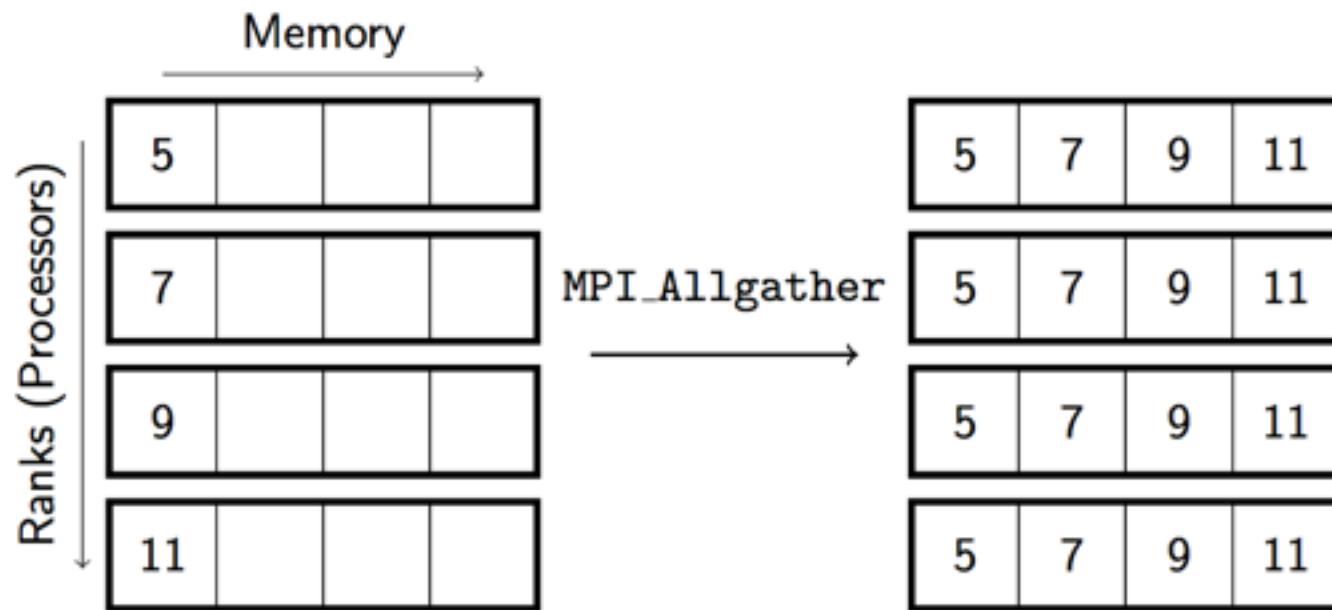
MPI collective communication. Scatter



MPI collective communication. Gather

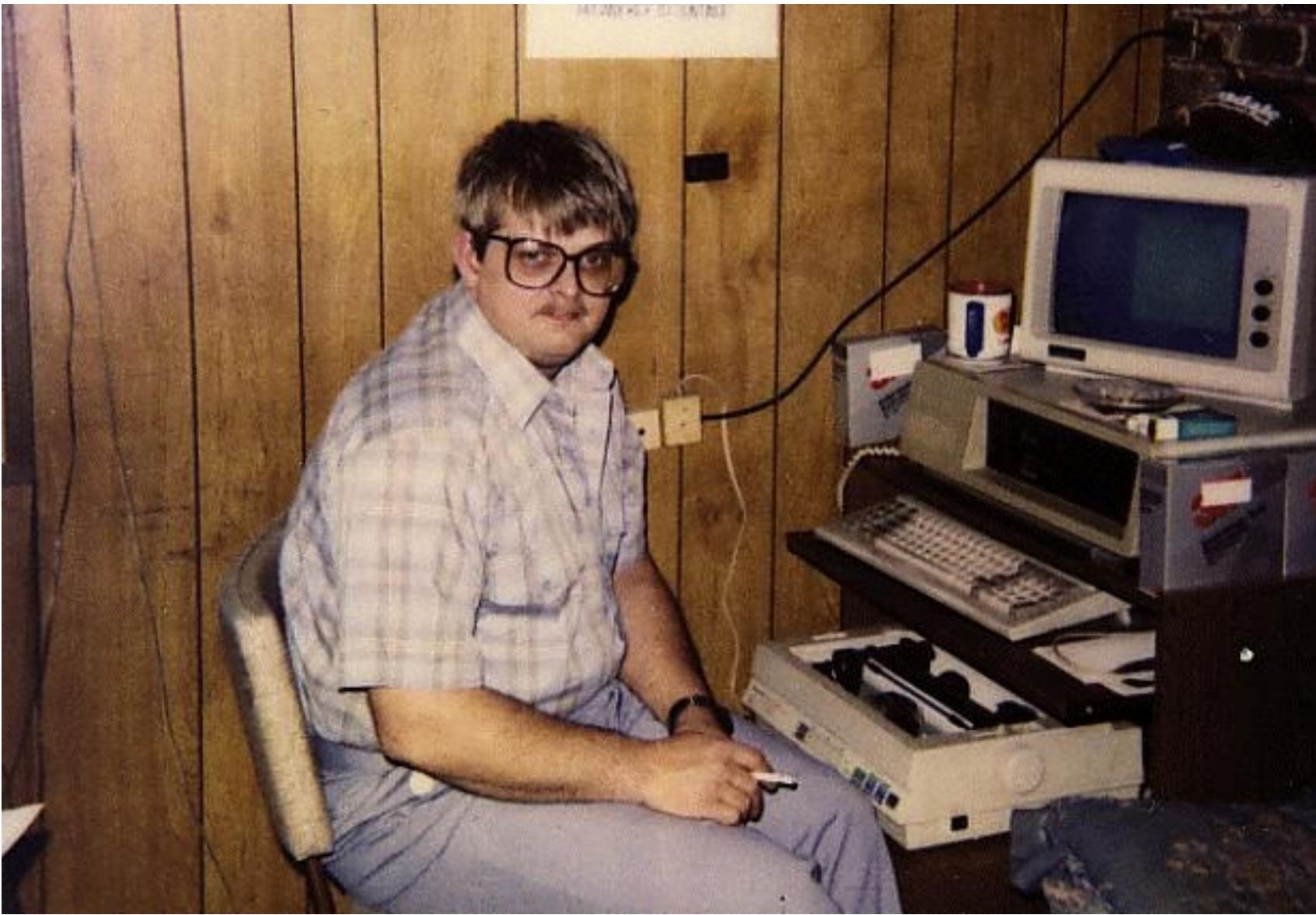


MPI collective communication. Allgather



MPI измерение времени исполнения

```
#include "mpi.h"
.....
MPI_Init(&argc, &argv);
.....
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);
.....
double time_elapsed = MPI_Wtime();
// Computations
// more computations
time_elapsed = MPI_Wtime() - time_elapsed;
MPI_Finalize();
```



6. Вычисления на распределенных сетках

Вычисления на распределенной сетке

Например, уравнение Лапласа:

$$\nabla^2 f = 0$$

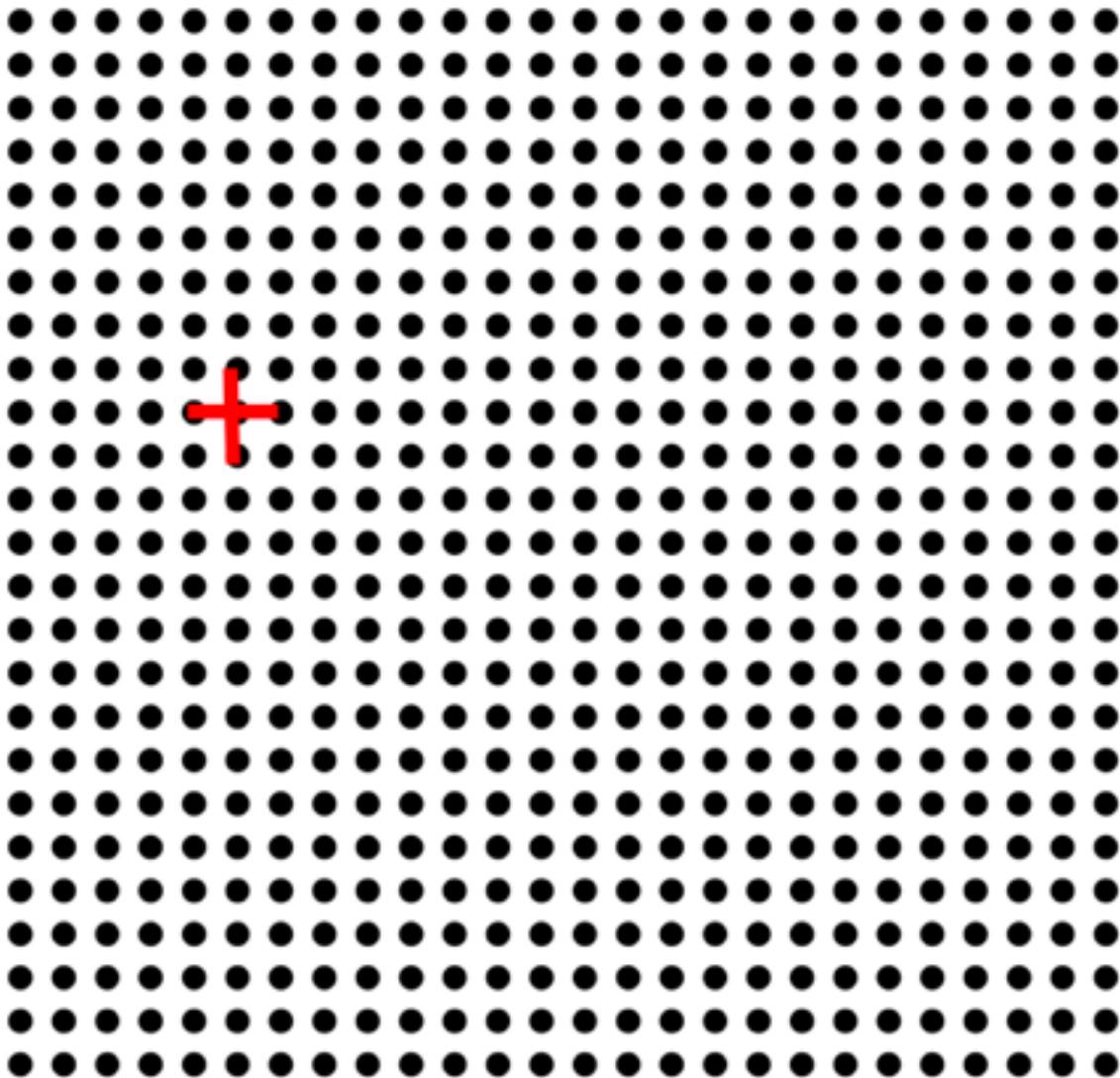
Или размытие картинки

Или клеточный автомат

Черные круги - узлы вычислительной сетки

Красный крестик - численный шаблон

Нужна коммуникация с соседями



Вычисления на распределенной сетке

Например, уравнение Лапласа:

$$\nabla^2 f = 0$$

Или размытие картинки

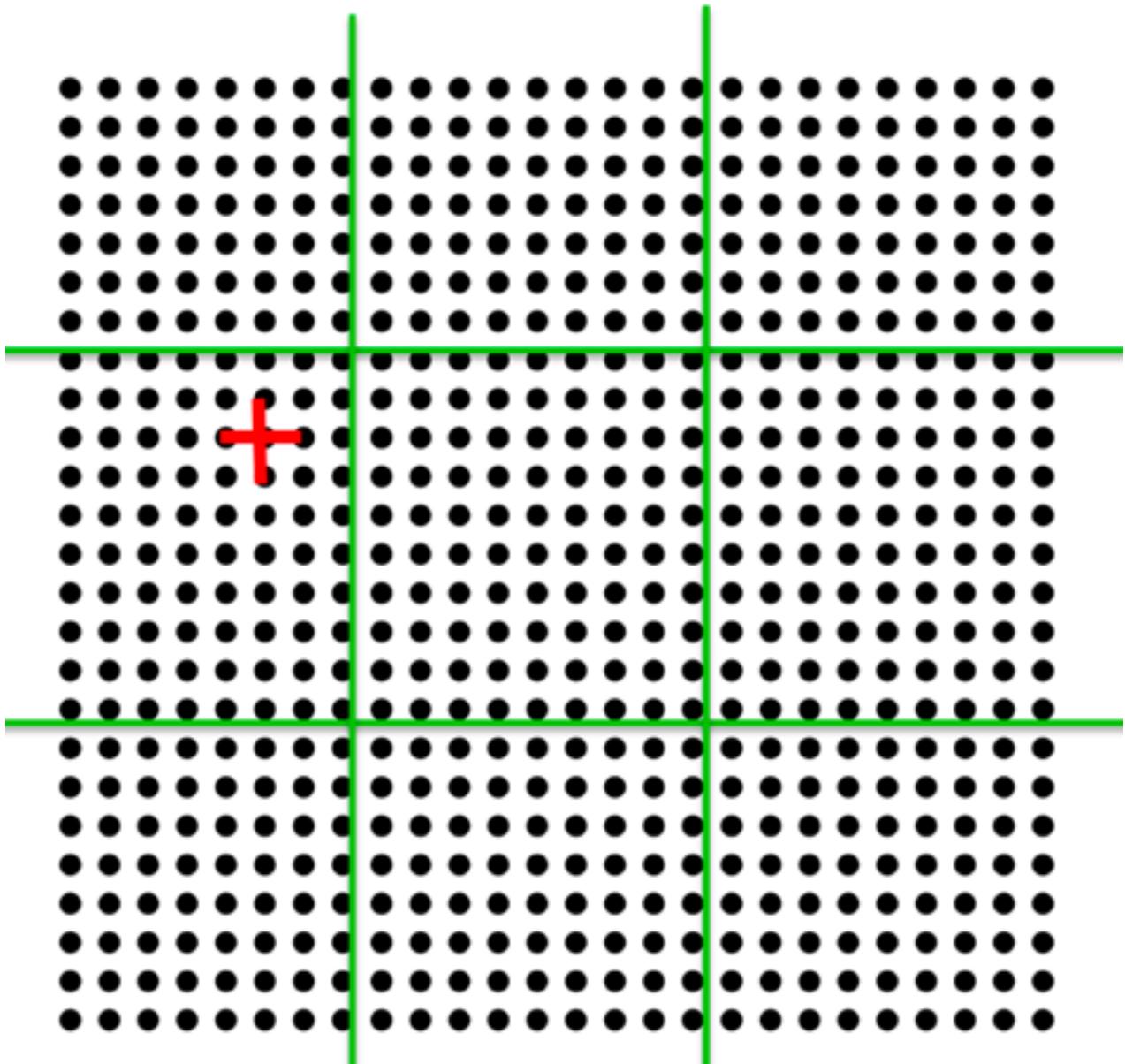
Или клеточный автомат

Черные круги - узлы вычислительной сетки

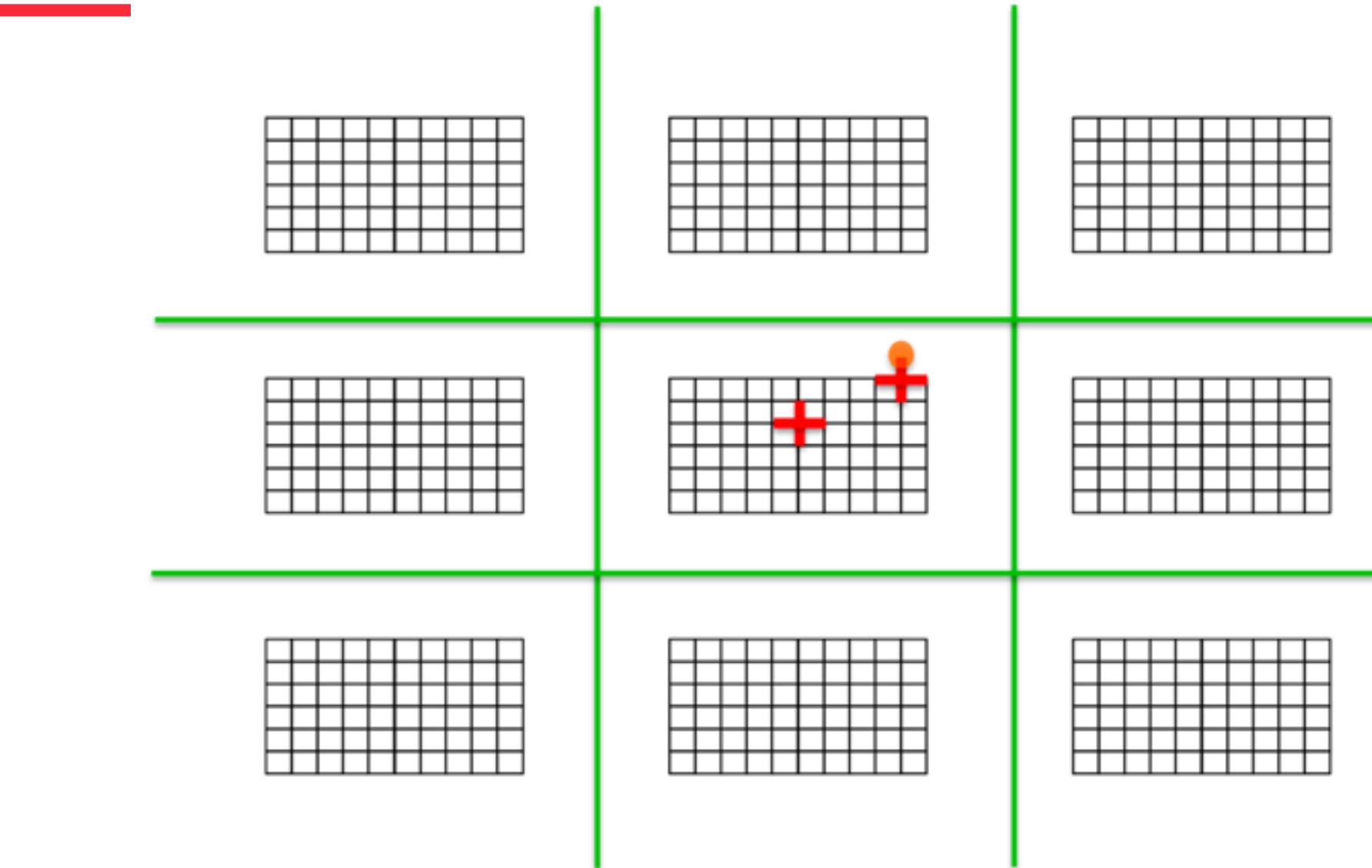
Красный крестик - численный шаблон

Нужна коммуникация с соседями

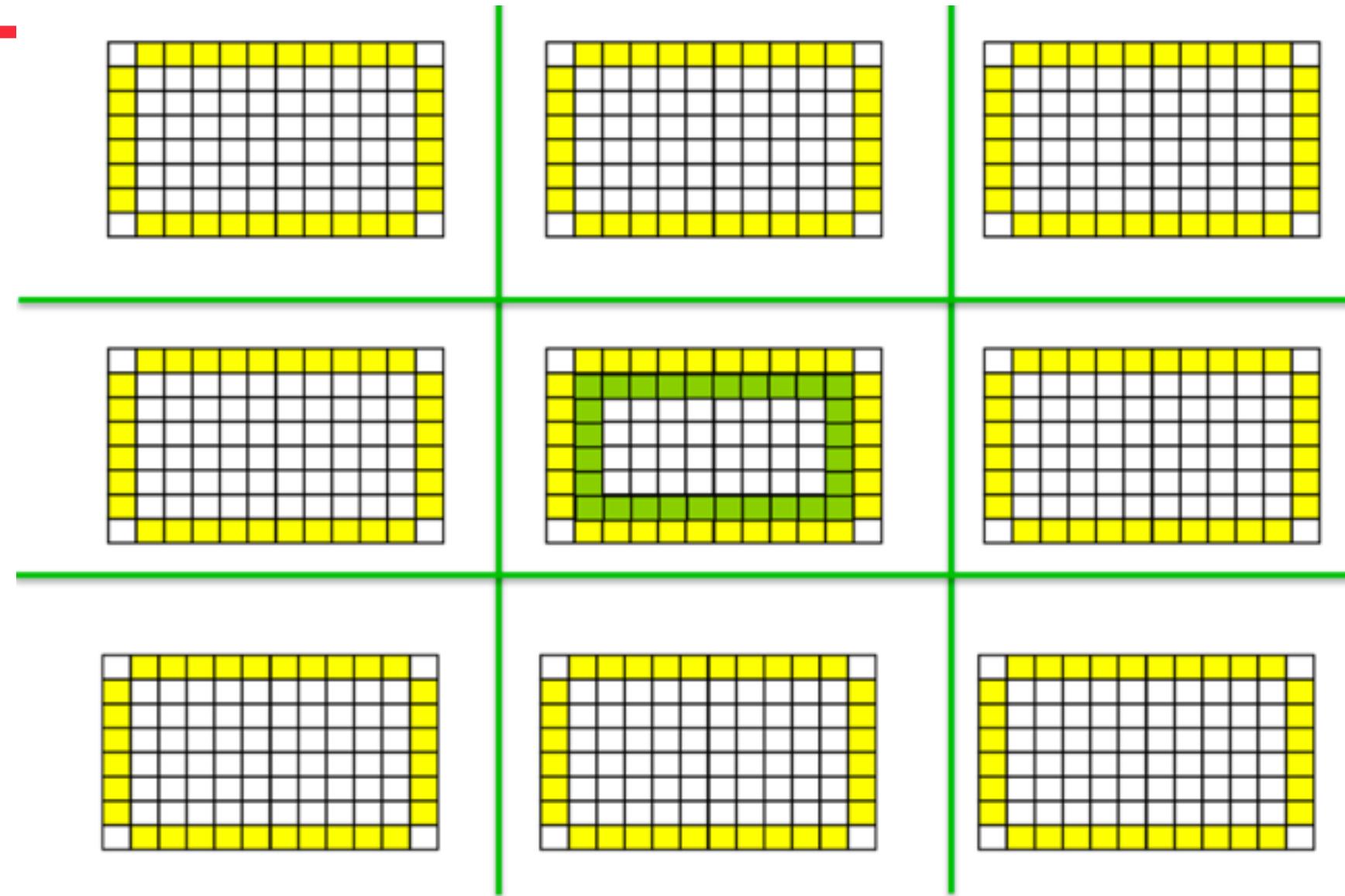
Разбиваем сетку на (равные) куски и
раздаем процессам



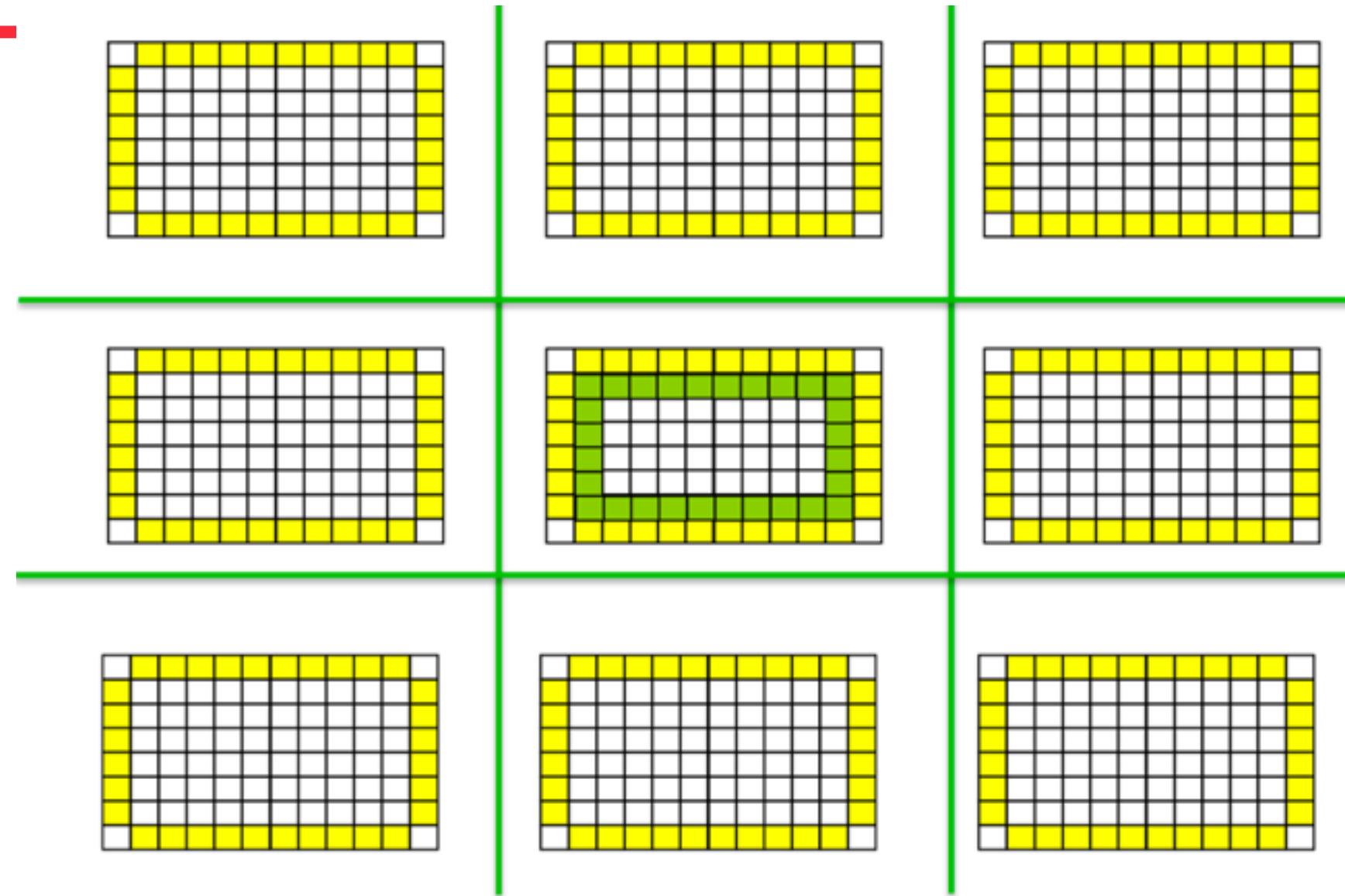
Необходимая коммуникация



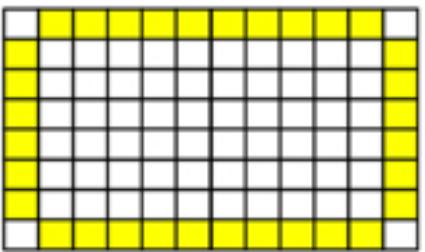
Необходимая коммуникация. Ghost cells



Необходимая коммуникация. Ghost cells



Алгоритм вычисления на распределенных сетках



Цикл по времени или цикл сходимости:

1. Отсылаем “клетки призраки” соседним процессорам
2. Получаем “клетки призраки” от соседних процессоров
3. Проводим вычисления (можем использовать OpenMP, Cuda)

Пример: Конвеевская игра “Жизнь”



John Horton Conway
(1937 – 2020)

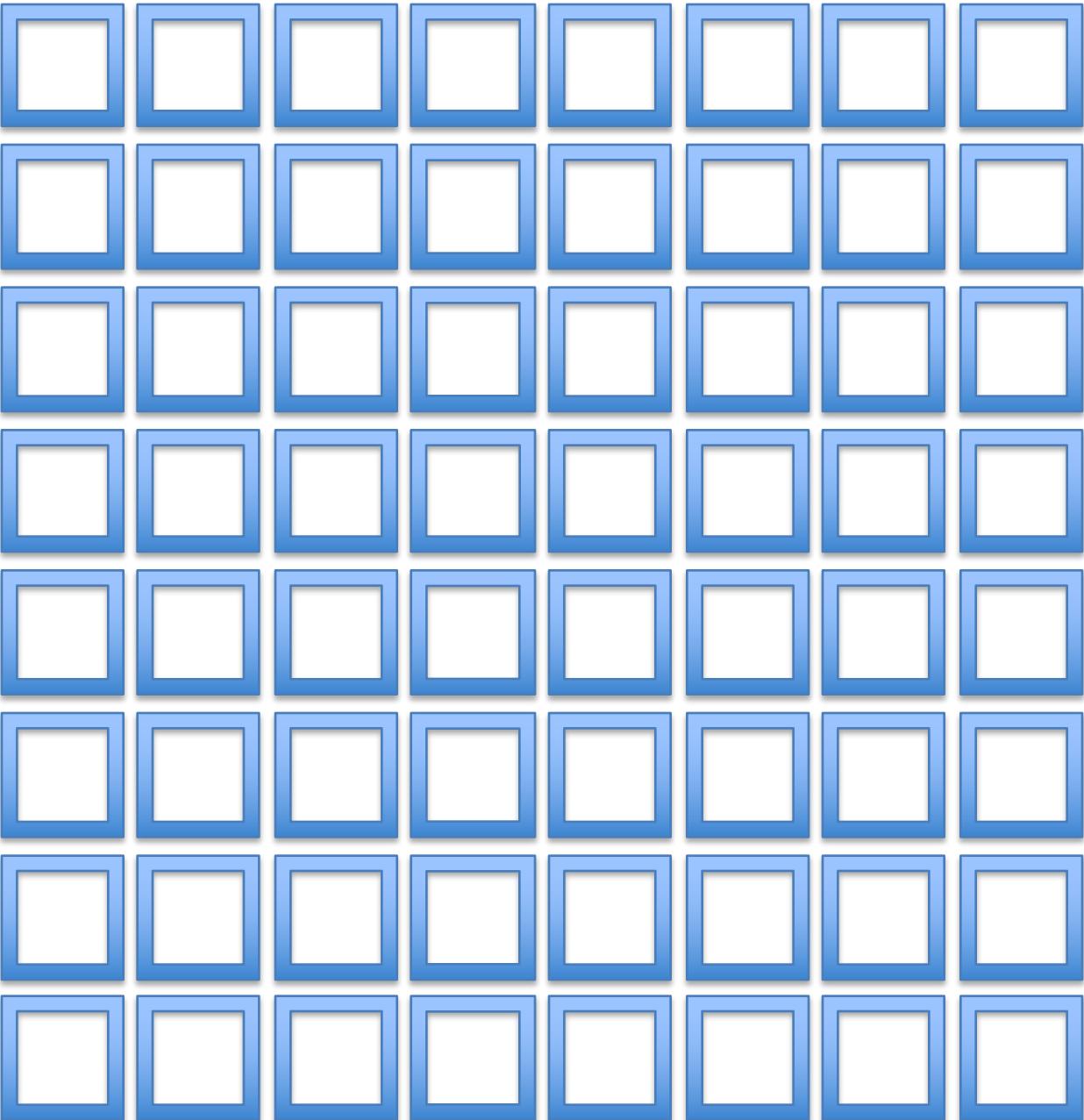


Клеточный автомат с простыми правилами:

1. Живая клетка (черная) с 2 или 3 живыми соседями выживает
2. Мертвая клетка (белая) с 3 живыми соседями становится живой
3. В других случаях все живые умирают, а мертвые остаются мертвыми

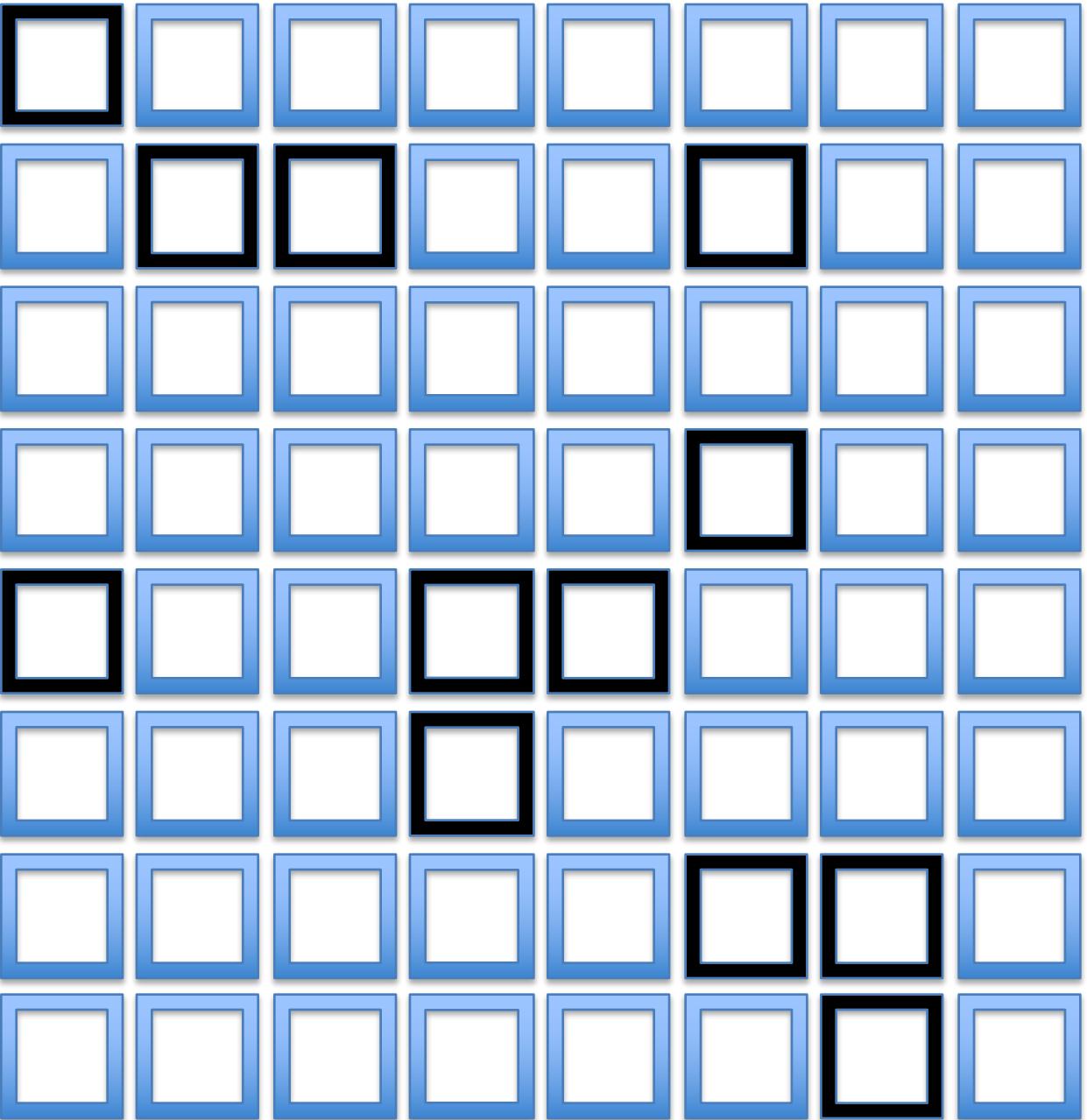


Инициализация (random)





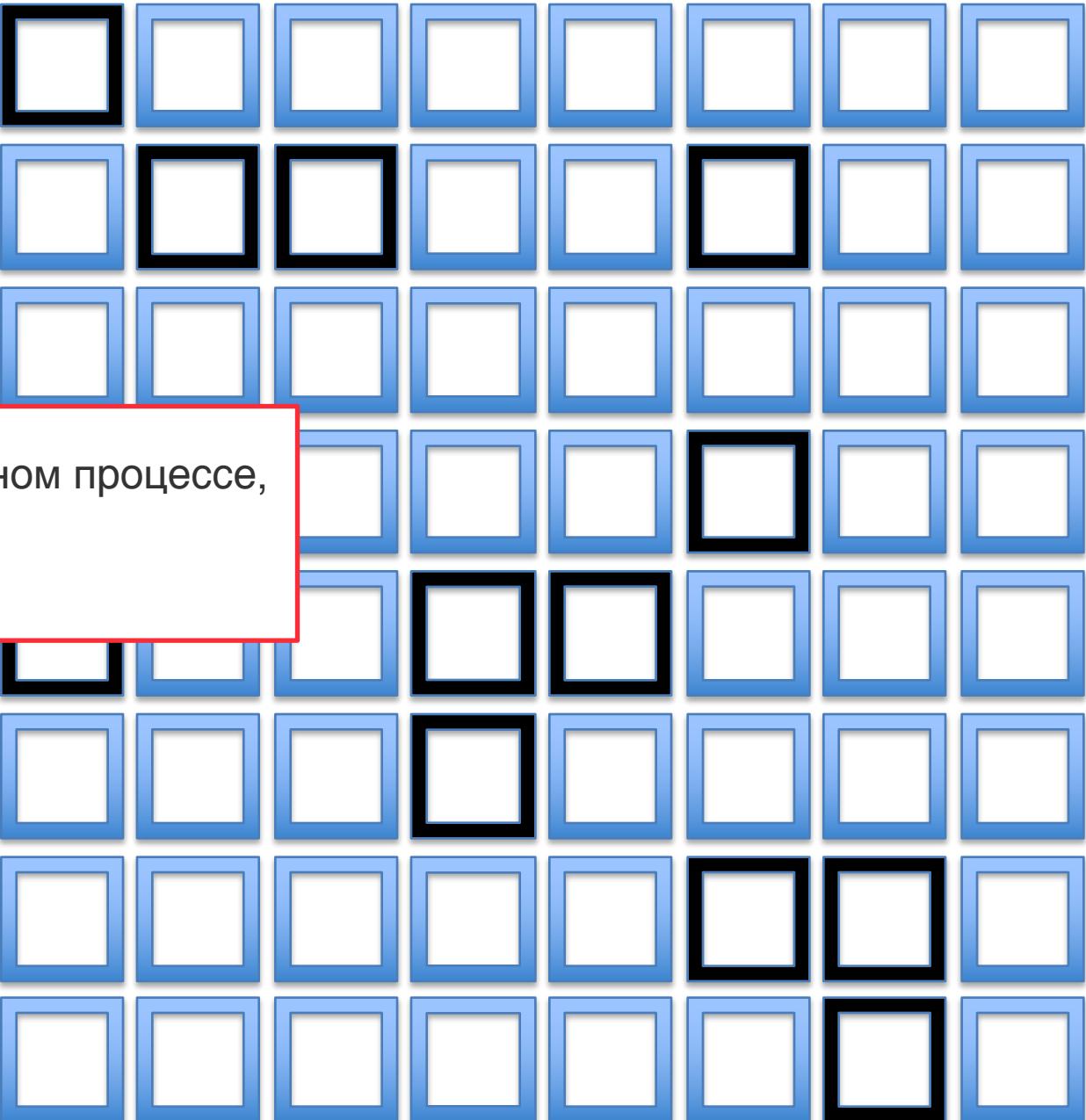
Инициализация (random)

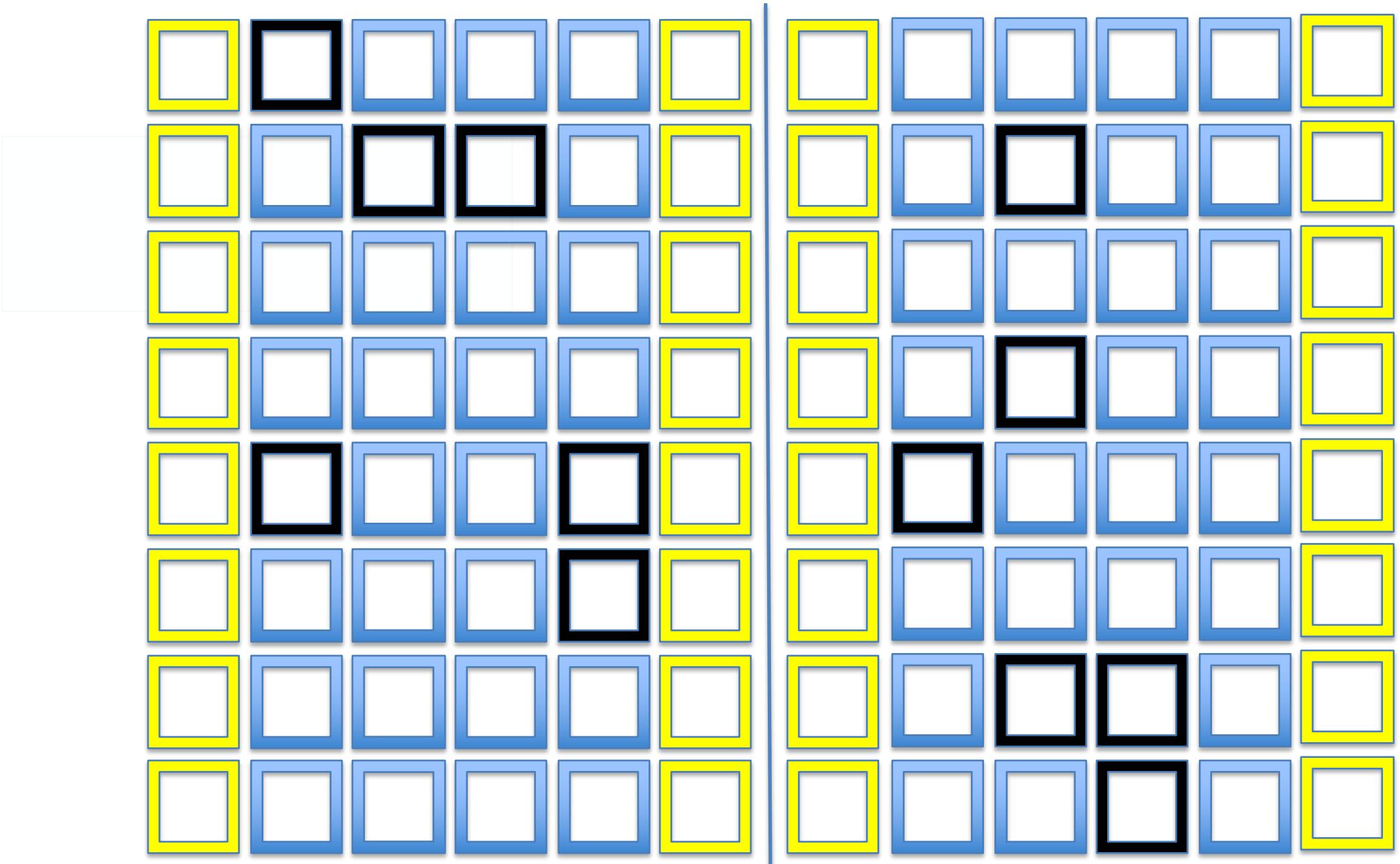


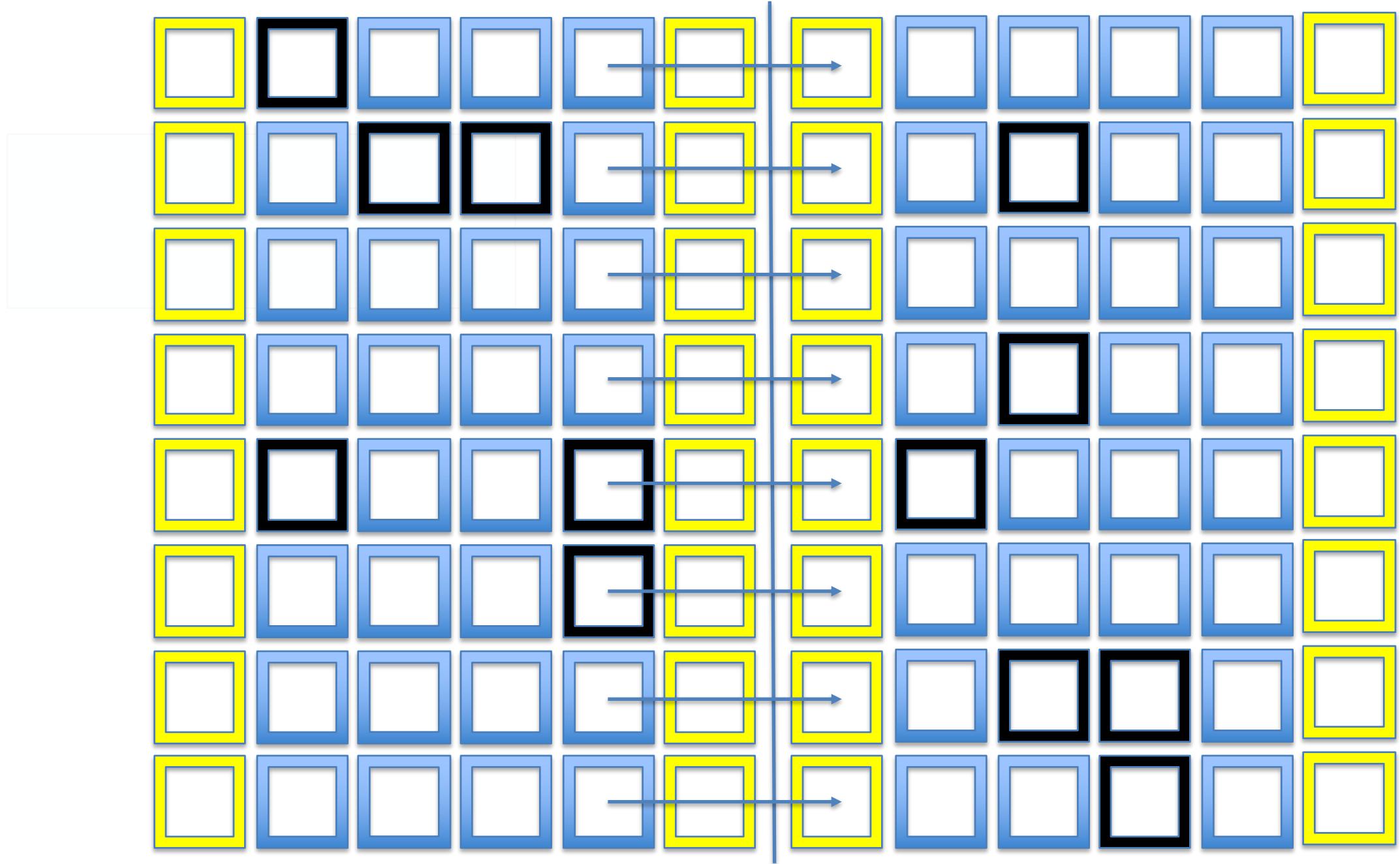


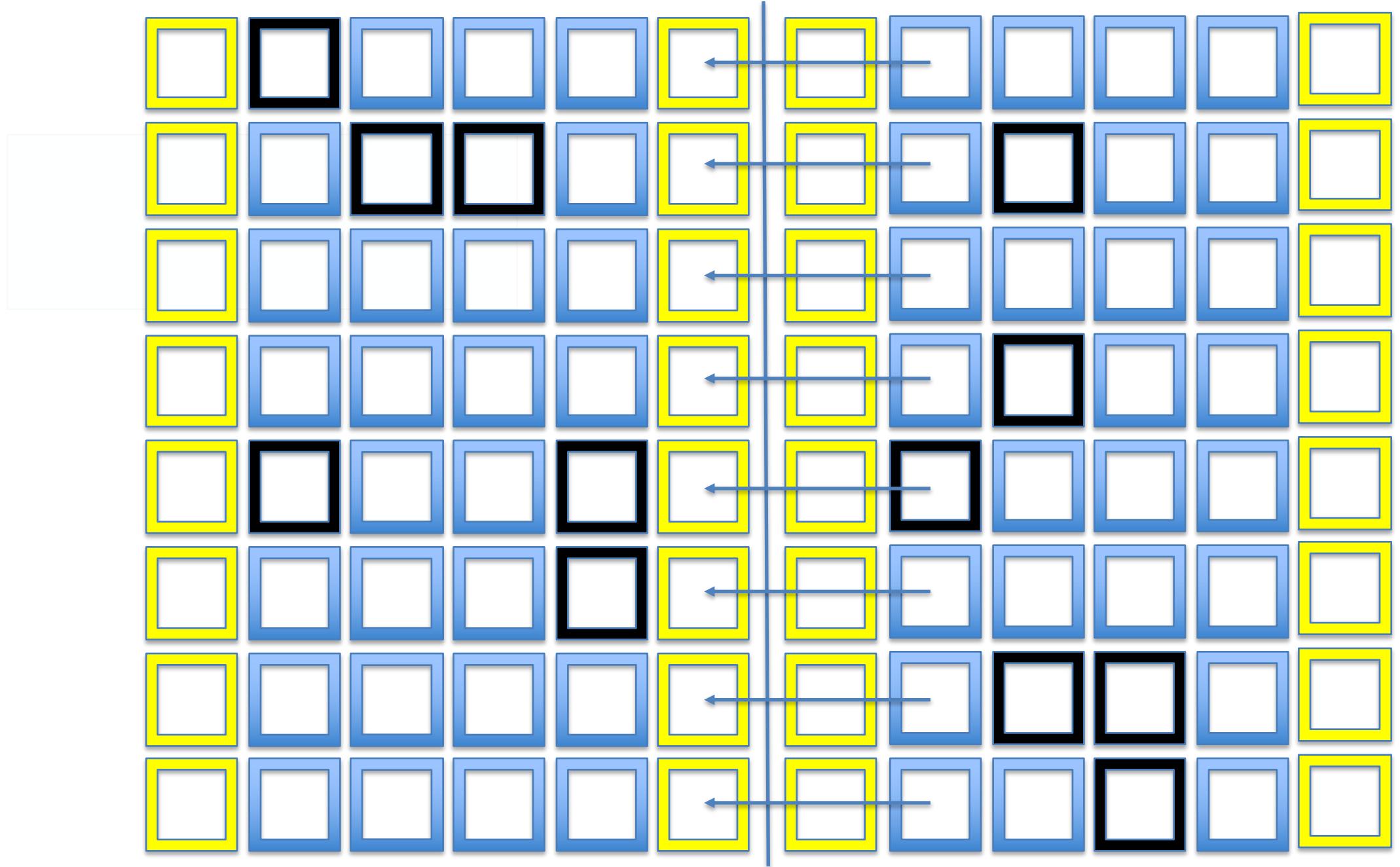
Инициализация (random)

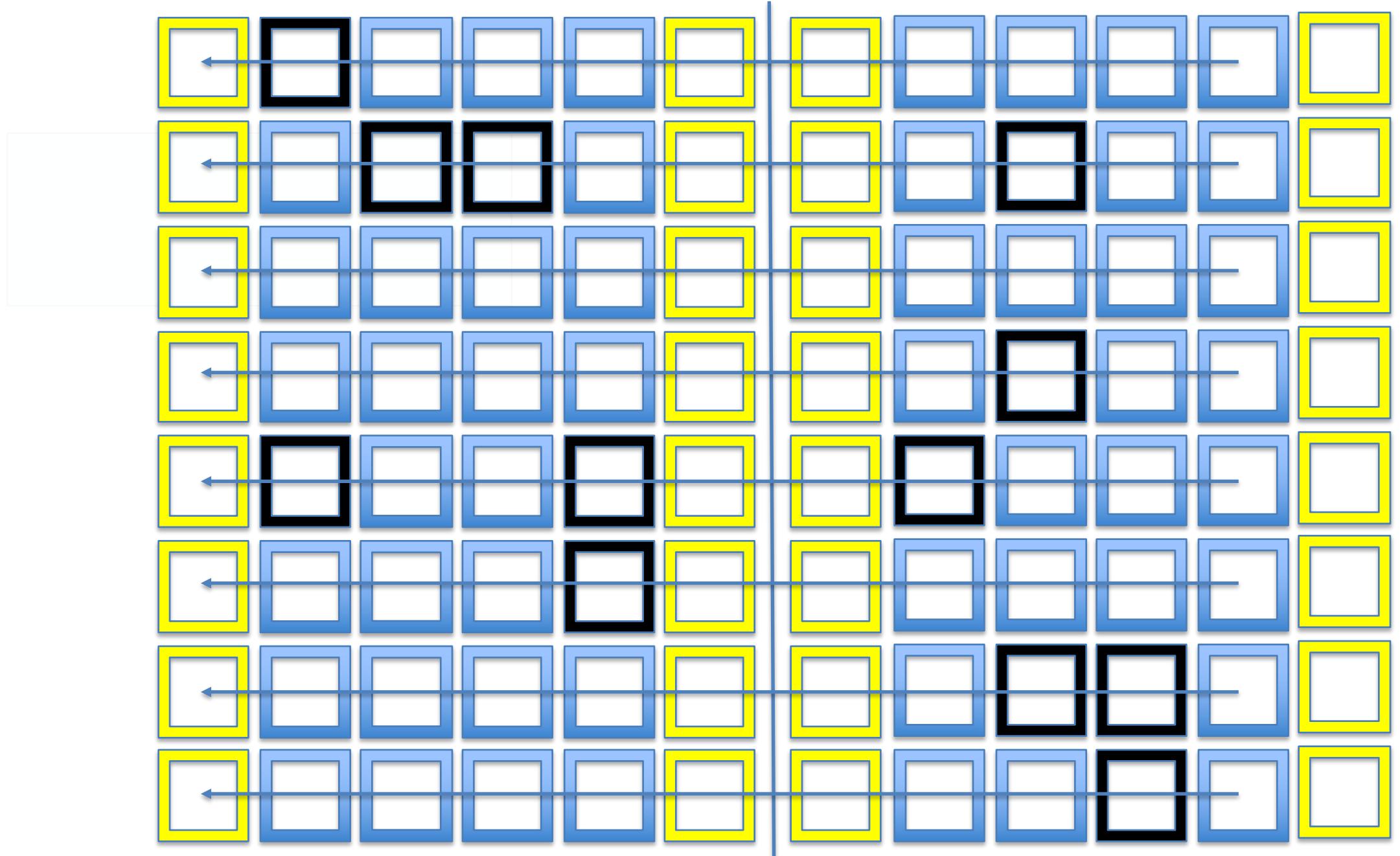
Не стоит инициализировать всю карту на одном процессе,
а потом делать broadcast! Каждый процесс
инициализирует свою часть карты

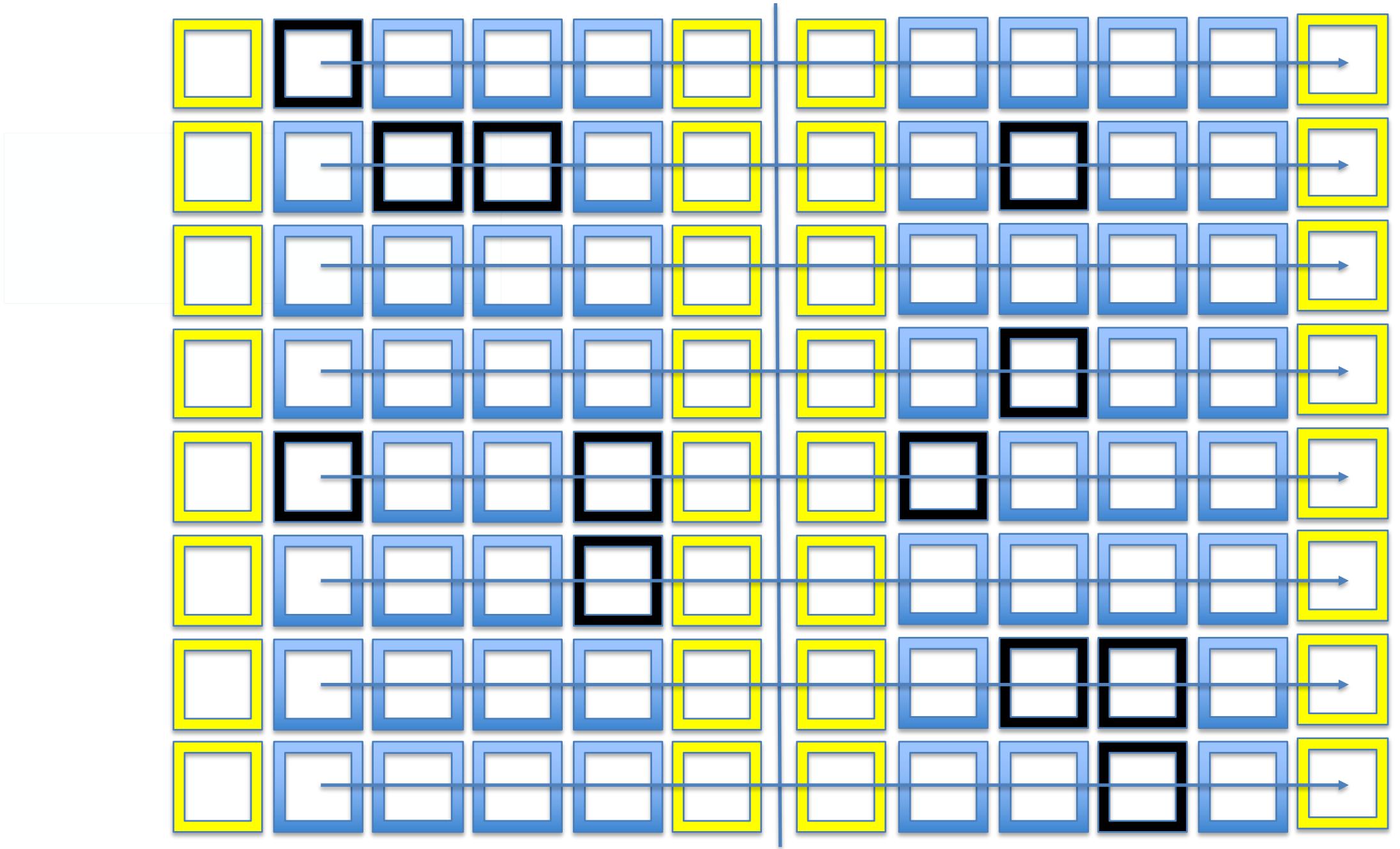














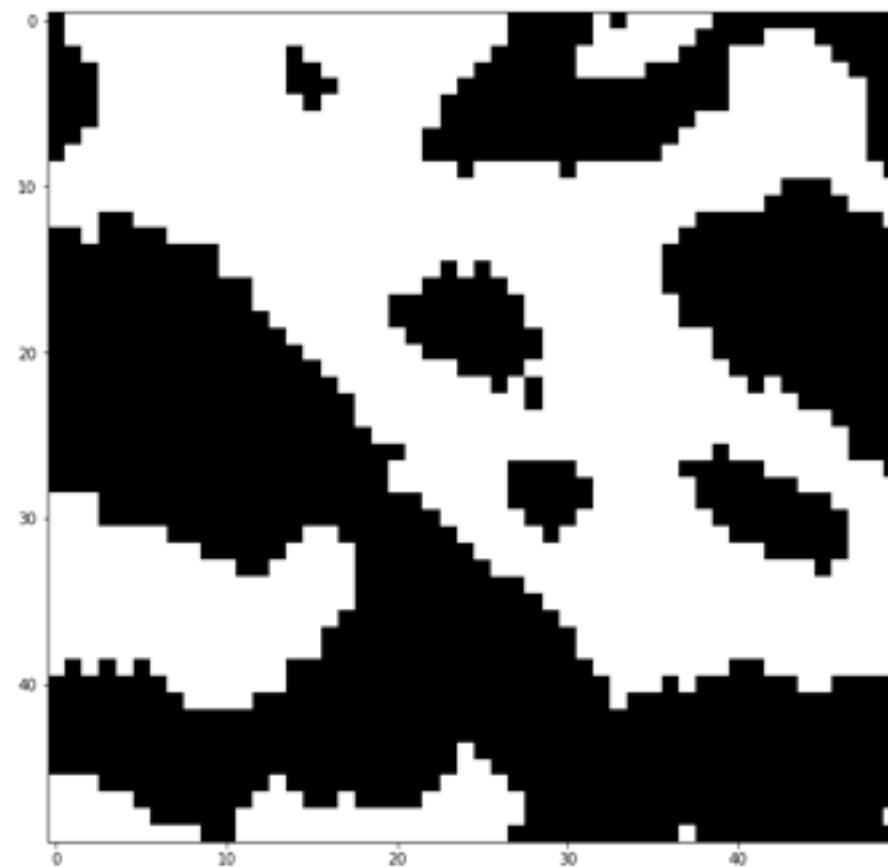
MPI как сохранить данные на диск

1. Каждый процессор записывает свою часть вычислений в отдельный файл: “grid-000.dat, grid-001.dat...”
2. Parallel I/O (mpi, HDF5 library, ADIOS)
3. Процессинг данных “на лету” (например, OpenGL)

DYNAMIC MODELS OF SEGREGATION†

THOMAS C. SCHELLING

Harvard University



дз. Одномерный клеточный автомат (Вольфрам)

current automaton contents



rule 110 (01101110)

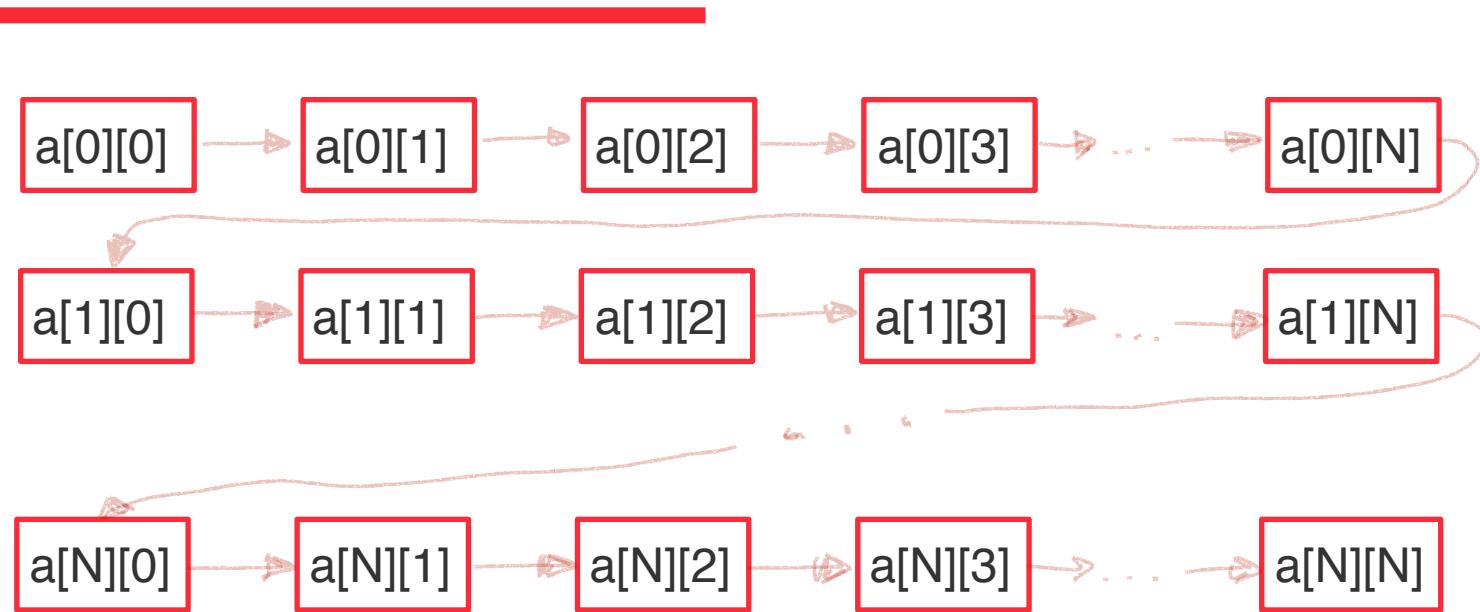


the next generation of the automaton



7. MPI создание нового типа данных

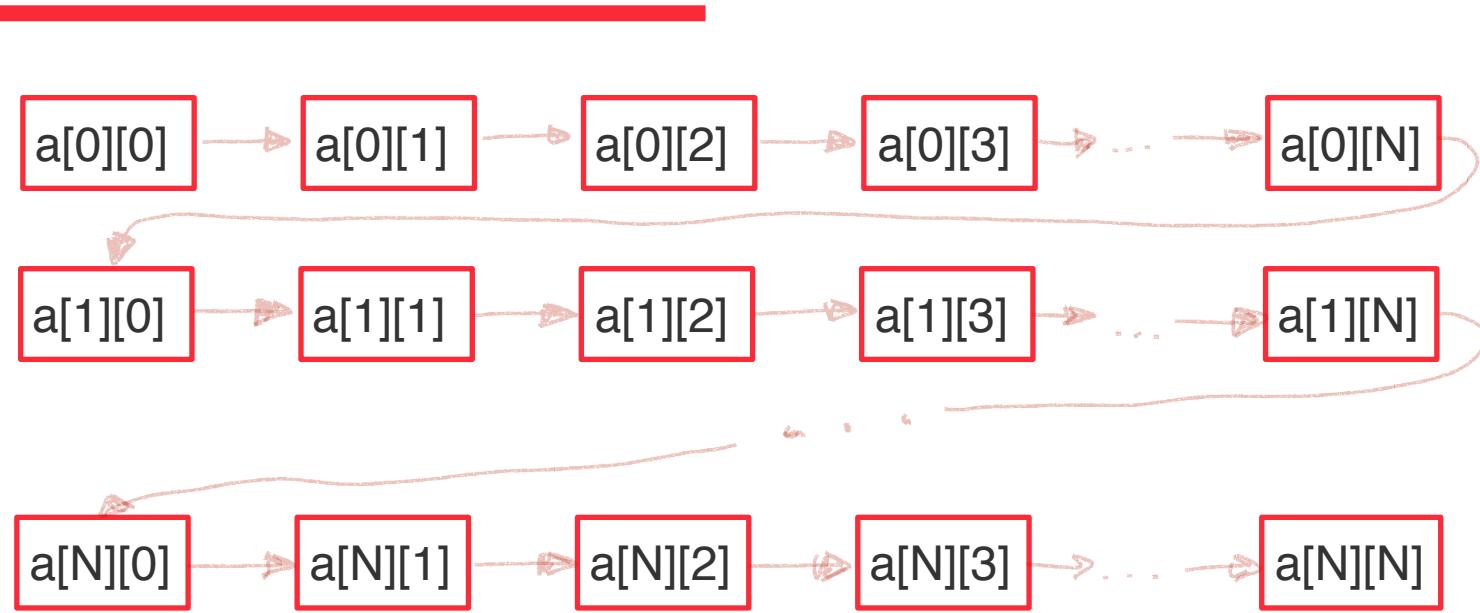
MPI derived data-types



Пересылка строчки элементарна:

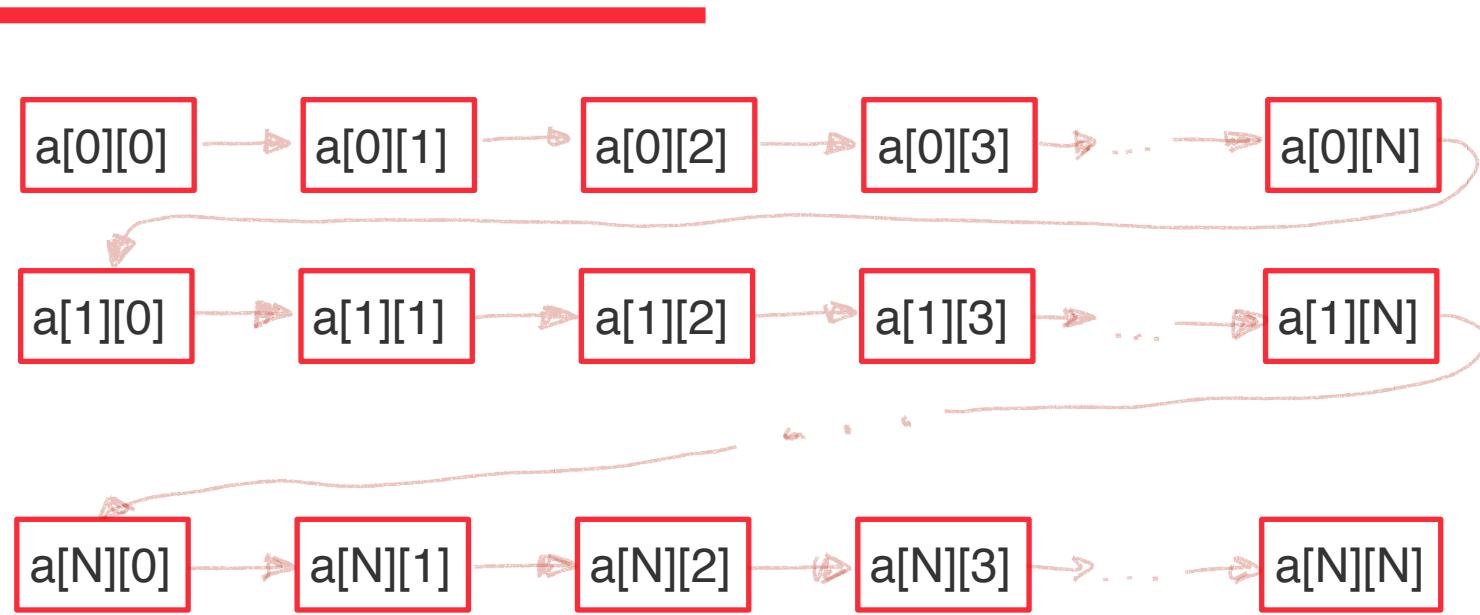
```
MPI_Send(&a[0][0], N+1, MPI_INT, 1, 1, MPI_COMM_WORLD);
```

MPI derived data-types



Что насчет пересылки столбца?

MPI derived data-types

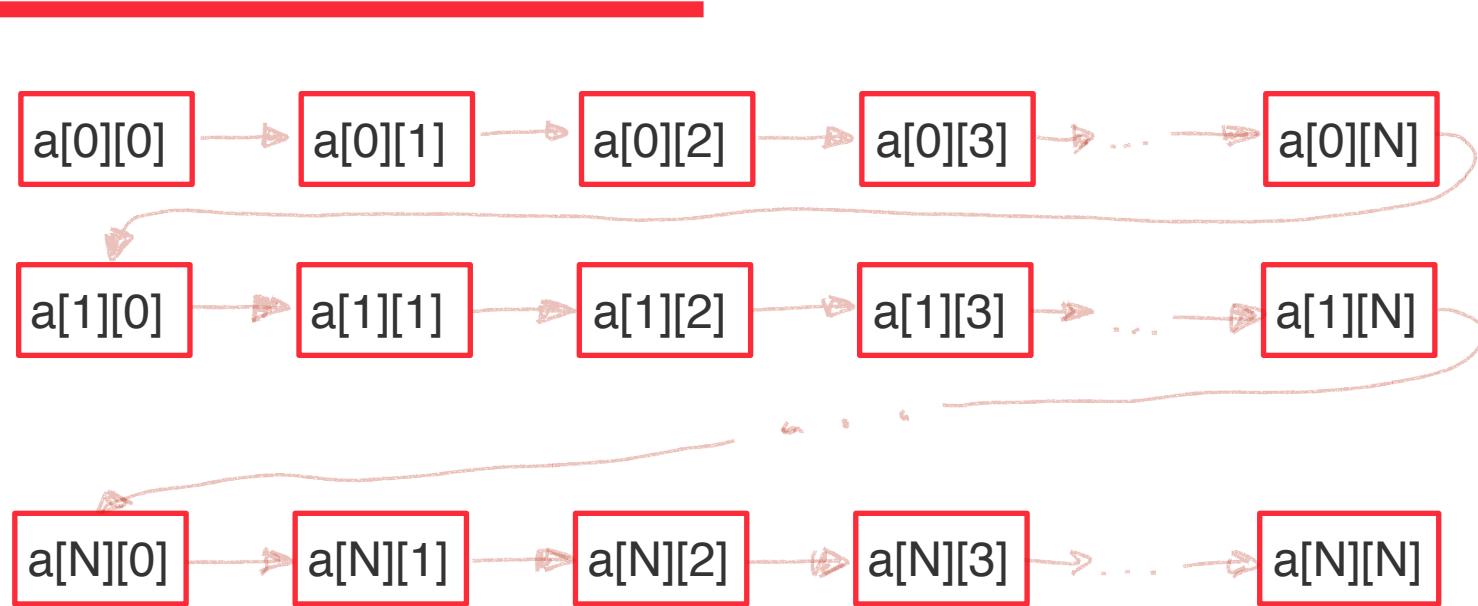


Что насчет пересылки столбца?

Пересылка данных = Установка соединения (latency) + Непосредственно почта (по пакетам)

Решение 1. Создаем временный буфер, куда помещаем нужный столбец

MPI derived data-types



Что насчет пересылки столбца?

Пересылка данных = Установка соединения (latency) + Непосредственно почта (по пакетам)

Решение 2. Используем шаблоны обращения к памяти, предоставленные MPI

Например: “Возьми p блоков, каждый по 2 элемента с расстоянием между началом каждого блока равным 5”



MPI derived data-types

MPI предоставляет возможность создать свой тип данных и потом его использовать в MPI_Send.

1. Создаем новый тип используя конструктор типов:

`MPI_Type_vector`

`MPI_Type_indexed`

`MPI_Type_struct`

2. Регистрируем новый тип используя команду “`MPI_Type_commit`”

3. Удаляем тип используя команду “`MPI_type_free`”

Пример MPI_Type_vector

```
int MPI_Type_vector(  
    int count,  
    int blocklength,  
    int stride,  
    MPI_Datatype oldtype,  
    MPI_Datatype *newtype);
```

кол-во блоков

кол-во элементов
в каждом блоке

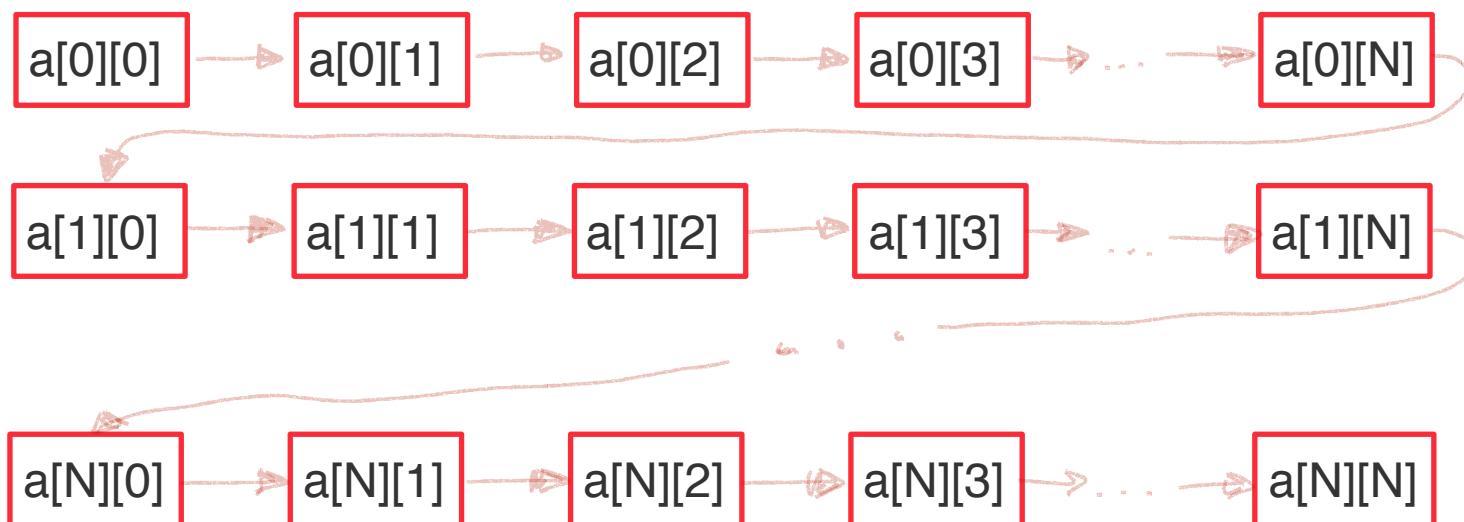
расстояние между
блоками

MPI_Type_indexed
MPI_Type_struct

Концептуально также, как MPI_Type_vector

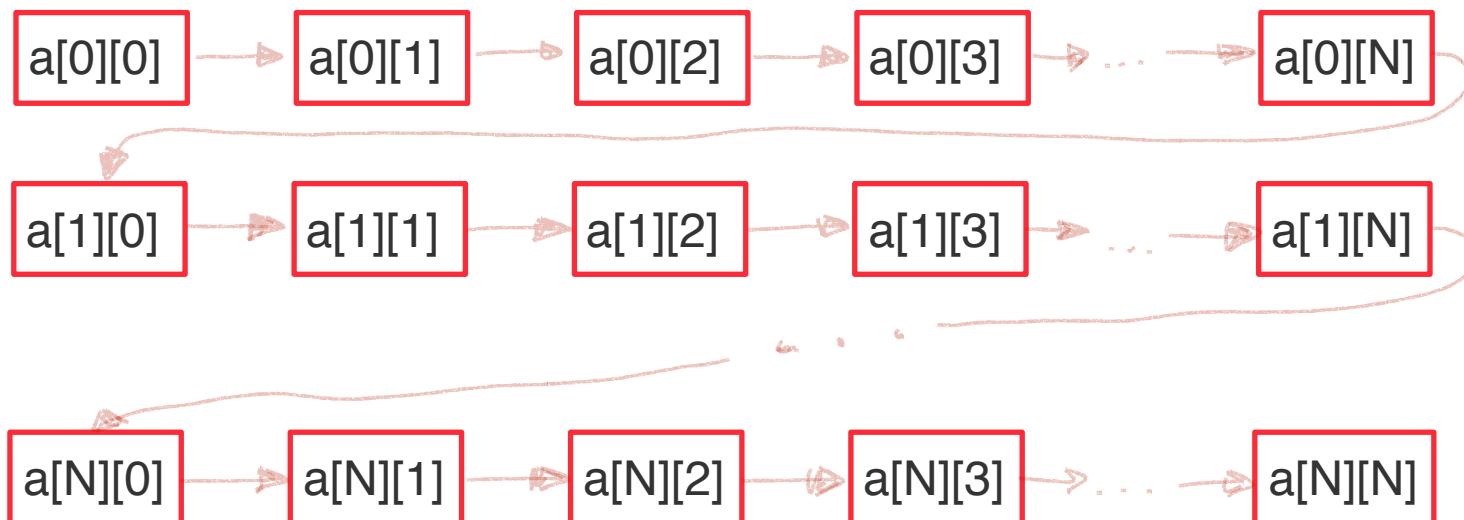
Пример MPI_Type_vector

```
MPI_Datatype ColumnType;  
MPI_Type_vector(n, 1, n, &ColumnType, MPI_INT);  
MPI_Type_commit(&ColumnType);  
  
MPI_Send(&a[0][0], 1, ColumnType, 1, 1, MPI_COMM_WORLD);
```



Пример MPI_Type_vector

```
MPI_Datatype ColumnType;  
MPI_Type_vector(n, 1, n, &ColumnType, MPI_INT);  
MPI_Type_commit(&ColumnType);  
  
MPI_Send(&a[0][0], 1, ColumnType, 1, 1, MPI_COMM_WORLD);
```



1. Пересылка четных строчек?
2. Пересылка четных столбцов?
3. Дома: сравните несколько вариантов пересылки столбца (поэлементно, буфер, шаблон)

8. MPI группы и коммуникаторы

MPI группы и коммуникаторы

```
MPI_Comm_split(
```

```
    MPI_Comm comm,
```

```
    int color,
```

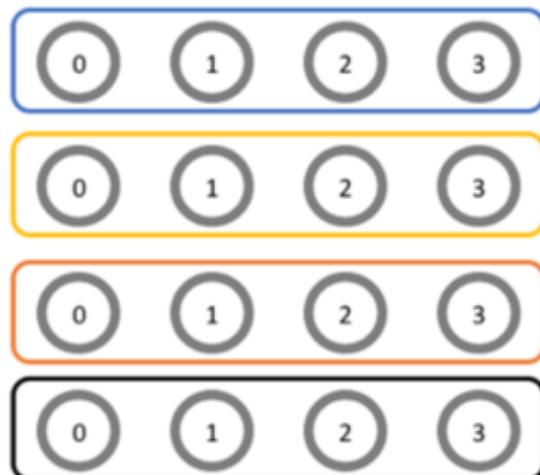
```
    int key,
```

```
    MPI_Comm* newcomm)
```

напри мер, *MPI_COMM_WORLD*

к какому комм-рч будет относиться
процесс (один цвет = один комм-р)
определитель ранга ($\min \rightarrow \text{root}$)

новой коммуникатор



MPI группы и коммуникаторы

```
// Get the rank and size in the original communicator
int world_rank, world_size;
MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
MPI_Comm_size(MPI_COMM_WORLD, &world_size);

int color = world_rank / 4; // Determine color based on row

// Split the communicator based on the color and use the
// original rank for ordering
MPI_Comm row_comm;
MPI_Comm_split(MPI_COMM_WORLD, color, world_rank, &row_comm);

int row_rank, row_size;
MPI_Comm_rank(row_comm, &row_rank);
MPI_Comm_size(row_comm, &row_size);

printf("WORLD RANK/SIZE: %d/%d \t ROW RANK/SIZE: %d/%d\n",
      world_rank, world_size, row_rank, row_size);

MPI_Comm_free(&row_comm);
```



MPI группы и коммуникаторы

Другие функции для коммуникаторов:

`MPI_Comm_dup` – создает дубликат коммуникатора

`MPI_Comm_create` – создает новый коммуникатор из группы процессов

Группа процессов – упорядоченное множество идентификаторов процессов. Порядок устанавливается присваиванием в группе ранга от 0 до `group.size - 1`

Коммуникатор – группа процессов + обертка для коммуникации между ними

```
int MPI_Comm_group(MPI_Comm comm, MPI_Group *group);
```

```
int MPI_Group_incl(MPI_Group group, int n, int *ranks, MPI_Group *newgroup);
```

```
int MPI_Group_excl(MPI_Group group, int n, int *ranks, MPI_Group *newgroup);
```

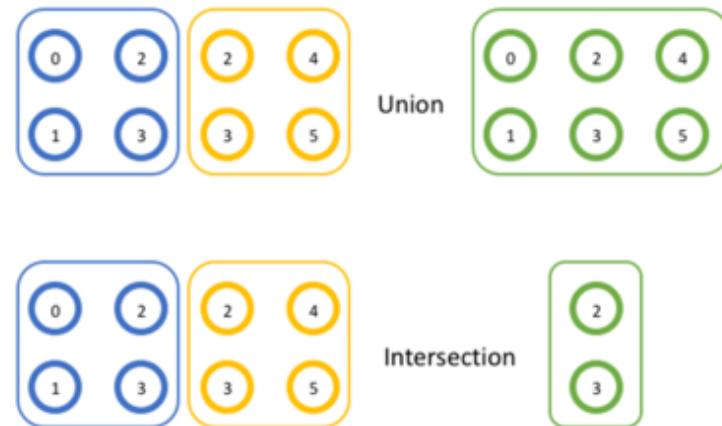
MPI группы и коммуникаторы

```
MPI_Group ngr;
int ranks[2];
ranks[0]=0;
ranks[1]=2;
ranks[2]=4;
MPI_Group_incl(gr, 3, &ranks, &ngr);
```

MPI группы и коммуникаторы

```
MPI_Group ngr;  
int ranks[2];  
ranks[0]=0;  
ranks[1]=2;  
ranks[2]=4;  
MPI_Group_incl(gr, 3, &ranks, &ngr);
```

```
int MPI_Group_union(MPI_Group group1, MPI_Group group2, MPI_Group *newgroup);  
  
int MPI_Group_intersection(MPI_Group group1, MPI_Group group2,  
                           MPI_Group *newgroup);  
  
int MPI_Group_difference(MPI_Group group1, MPI_Group group2,  
                         MPI_Group *newgroup);  
  
int MPI_Group_free(MPI_Group *group);  
  
MPI_GROUP_EMPTY
```





MPI группы и коммуникаторы

```
MPI_Comm newcomm;  
  
int MPI_Comm_dub(MPI_Comm oldcomm, MPI_Comm *newcomm);  
  
int MPI_Comm_create(MPI_Comm comm, MPI_Group group, MPI_Comm *newcomm);  
  
int MPI_Comm_free(MPI_Comm comm);
```

Один процесс может входить в несколько коммуникаторов, но коммуникации только внутри одного коммуникатора!

```
background-color: #f5f5f5;
text-shadow: 0px 1px 0px #777;
filter: dropshadow(color=#777);
color:#777;

}
header #main-navigation ul li span:hover{
    border: 1px solid #777;
    background-color: #f5f5f5;
    color: #777;
    box-shadow: 0px 0px 1px #777;
    -webkit-box-shadow: 0px 0px 2px #777;
    moz-box-shadow: 0px 0px 1px #777;
    padding: 5px;
    margin: 5px;
}

header #main-navigation ul li span.active{
    border: 1px solid #777;
    background-color: #f5f5f5;
    color: #777;
    box-shadow: 0px 0px 1px #777;
    -webkit-box-shadow: 0px 0px 2px #777;
    moz-box-shadow: 0px 0px 1px #777;
    padding: 5px;
    margin: 5px;
}

header #main-navigation ul li span.dashboard{
    border: 1px solid #777;
    background-color: #f5f5f5;
    color: #777;
    box-shadow: 0px 0px 1px #777;
    -webkit-box-shadow: 0px 0px 2px #777;
    moz-box-shadow: 0px 0px 1px #777;
    padding: 5px;
    margin: 5px;
}

header #main-navigation ul li span.dashboard.active{
    border: 1px solid #777;
    background-color: #f5f5f5;
    color: #777;
    box-shadow: 0px 0px 1px #777;
    -webkit-box-shadow: 0px 0px 2px #777;
    moz-box-shadow: 0px 0px 1px #777;
    padding: 5px;
    margin: 5px;
}

header #main-navigation ul li span.dashboard img{
    width: 100px;
    height: 100px;
    border: 1px solid #777;
    border-radius: 50%;
    margin-bottom: 5px;
}
```

9. MPI виртуальная топология

MPI виртуальная топология

Как мы адресуемся к процессам?

Пока что одномерный массив процессов (ранги от 0 до size-1)

Иногда это очень не удобно (16 процессов должны разбить на куски многомерный тензор)

Декартова топология

| | | |
|-------------------------------|-----------------------|-----------------------|
| 0 (\emptyset, \emptyset) | 1 $(\emptyset, 1)$ | 2 $(\emptyset, 2)$ |
| 3 $(1, \emptyset)$ | 4 $(1, 1)$ | 5 $(1, 2)$ |



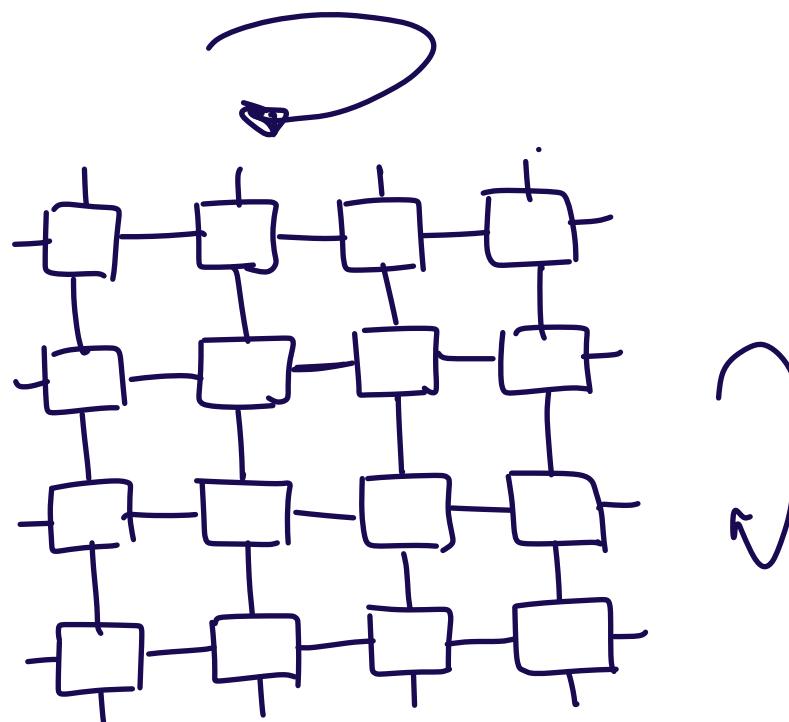
MPI виртуальная топология

```
MPI_Cart_create(MPI_Comm comm_old,  
                 int ndims,  
                 int *dims,  
                 int *periods,  
                 int reorder,  
                 MPI_Comm *comm_2D)
```

MPI виртуальная топология

```
MPI_Cart_create(MPI_Comm comm_old,  
                int ndims,  
                int *dims,  
                int *periods,  
                int reorder,  
                MPI_Comm *comm_2D)
```

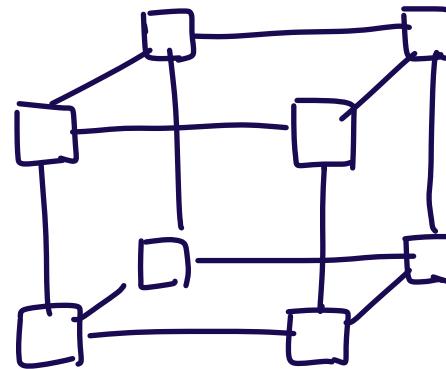
```
MPI_Comm comm_2D;  
int ndim = 2;  
int dims[2], periods[2];  
dims[0] = 4;  
dims[1] = 4;  
periods[0] = 1; → TOP  
periods[1] = 1;  
MPI_Cart_create(MPI_COMM_WORLD,  
                ndim, &dims, &periods,  
                0, &comm_2D);
```



MPI виртуальная топология

```
MPI_Cart_create(MPI_Comm comm_old,  
                int ndims,  
                int *dims,  
                int *periods,  
                int reorder,  
                MPI_Comm *comm_2D)
```

```
MPI_Comm comm_3D;  
int ndim=3;  
int dims[3], periods[3];  
dims[0]=2; dims[1]=2; dims[2]=0;  
periods[0]=0; periods[1]=0; periods[2]=0;  
MPI_Cart_create(MPI_COMM_WORLD, ndim, &dims, &periods, 0, &comm_3D);
```



куб

```
background-color: #f5f5f5;
text-shadow: 0px 1px 1px #ccc;
filter: dropshadow(color="#777");
color:#777;

}

header #main-navigation ul li span:hover,
header #main-navigation ul li span:active {
    border: 1px solid #ccc;
    background-color: #fff;
    color: #777;
    box-shadow: 0px 0px 1px #ccc;
    -webkit-box-shadow: 0px 0px 2px #ccc;
    moz-box-shadow: 0px 0px 1px #ccc;
    text-decoration: none;
    font-weight: bold;
    padding: 5px 10px;
    margin: 5px 0px;
}
```

Домашнее задание



Пояснения к шаблону

В вашем распоряжении есть следующие слайды:

- Титульный слайд
- Слайд №1.1 Стандартный
- Слайд №1.2 Стандартный с подзаголовком
- Слайд №2.1 Разворот справа
- Слайд №2.2 Разворот слева
- Слайд №3.1 Ноутбук справа
- Слайд №3.2 Ноутбук слева
- Слайд №4.1 2 столбца
- Слайд №4.2 2 столбца
- Слайд №5 Контакты
- Слайд №6.1 Отбивка вертикальная
- Слайд №6.2 Пустой слайд

Для акцентов в коде и тексте на слайдах в настройках цвета у вас есть готовая палитра:



HEX #fc2c38
RGB 252, 43, 56
Pantone 032C
CMYK 0, 97, 75, 0



HEX #e5e8e8
RGB 230, 231, 232
Pantone Cool Gray 1C
CMYK 0, 0, 0, 9



HEX #75787b
RGB 117, 120, 123
Pantone Cool Gray 9C
CMYK 0, 0, 0, 73



HEX #030303
RGB 3, 3, 3
Pantone Black
CMYK 75, 67, 67, 89

Слайды всех типов находятся в шаблоне, их можно выбрать из выпадающего меню в окне «Создать слайд»

Используйте для
отбивки между
блоками фотографии
из папки

```
background-color: #fff;
text-shadow: 0px 1px 1px #000;
filter: dropshadow(color="#000");
color:#777;
}
header #main-navigation ul li span:hover,
box-shadow: 0px 0px 1px #000;
-webkit-box-shadow: 0px 0px 2px #000;
moz-box-shadow: 0px 0px 1px #000;
background-color:#F0F9F9;
active span,
li span.dashboard,
.../img/dashboard.jpg);
ul li span.dashboard,
finaldashboard.jpg);
```

Текст

Иконки и элементы

Для создания ориентиров на слайде
используйте иконки из готового набора
или подходящие по цветовой гамме:

