

# ***Параллельное программирование для высокопроизводительных вычислительных систем***

сентябрь – декабрь 2018 г.

Лектор доцент Н.Н.Попова

---

Лекция 6  
15 октября 2018 г.

# Тема

---

- Виртуальные топологии
- Группы процессов

# Понятие коммуникатора MPI

---

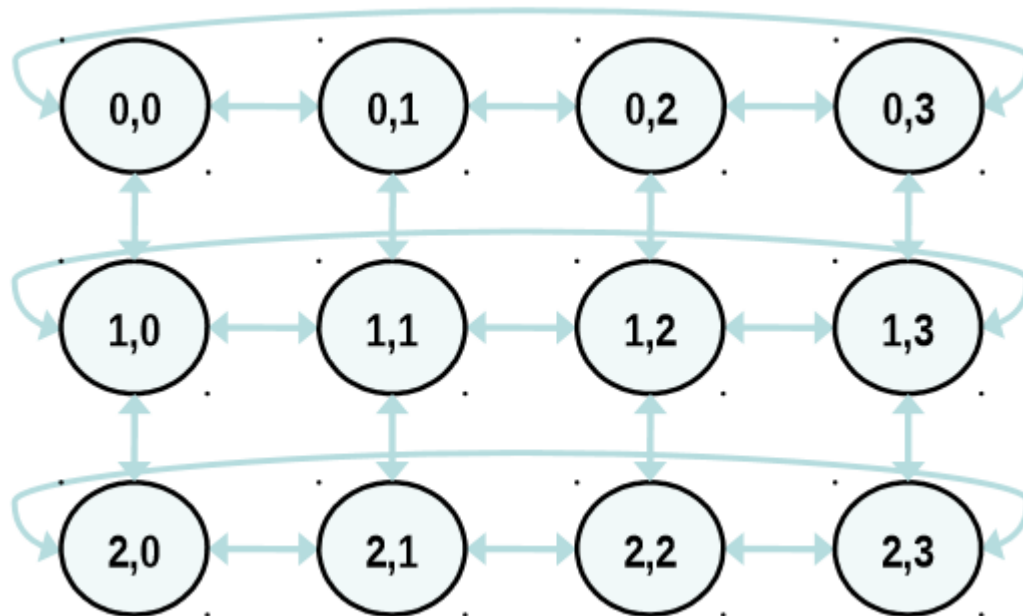
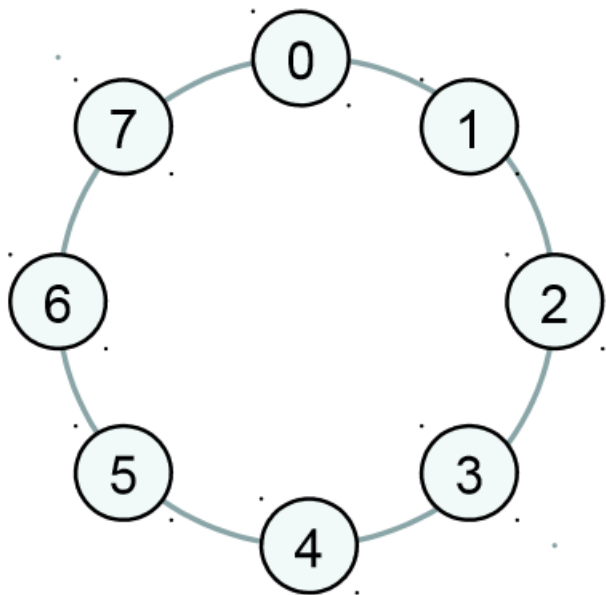
- Коммуникатор - управляющий объект, представляющий группу процессов, которые могут взаимодействовать друг с другом

# Виртуальные топологии

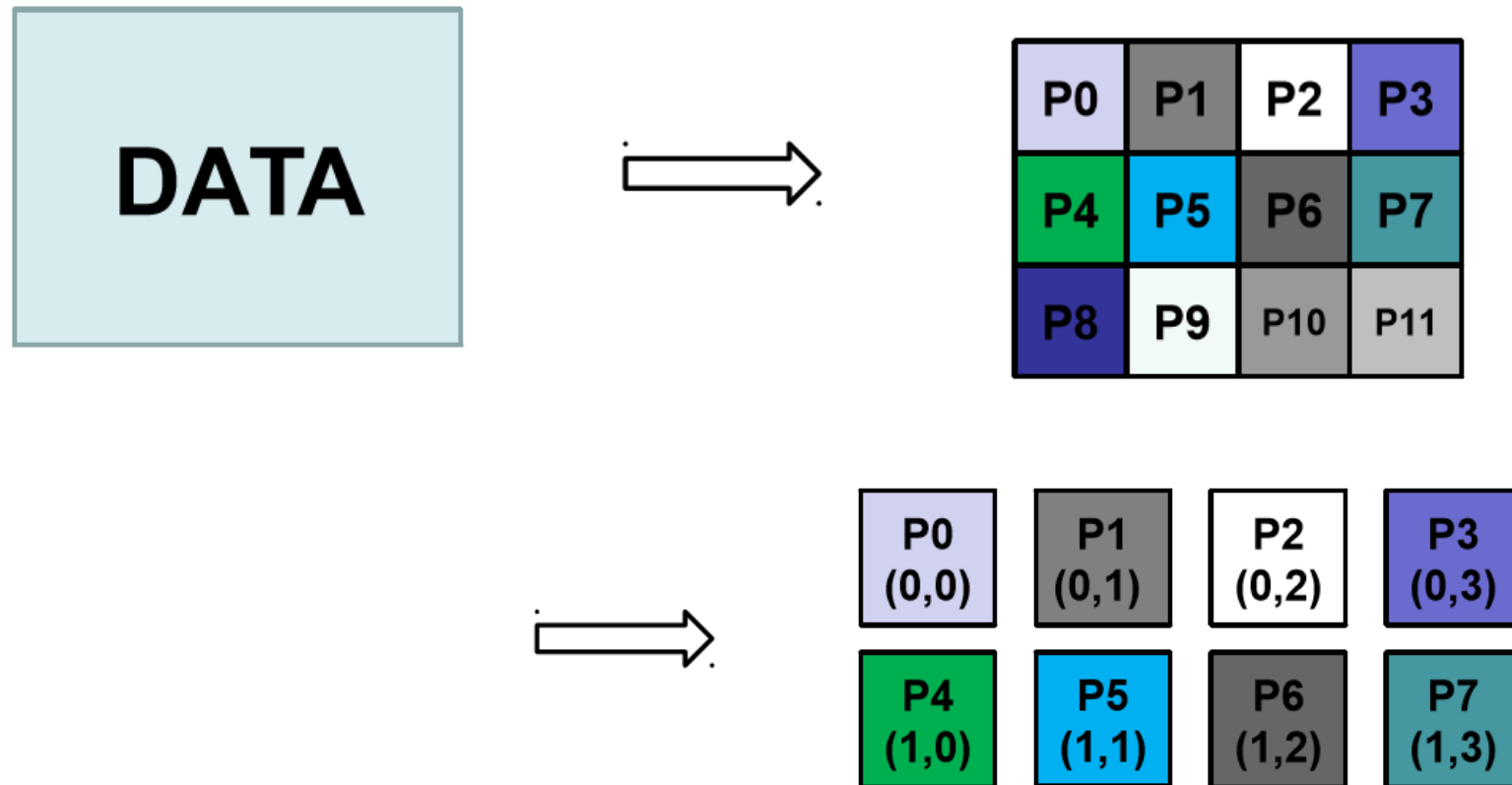
---

- Удобный способ именования процессов
- Упрощение написания параллельных программ
- Оптимизация передач
- Возможность выбора топологии, соответствующей логической структуре задачи

# Пример виртуальных топологий



# Отображение данных на виртуальную топологию



# Виртуальные топологии

---

- Основные функции:
  - MPI\_CART\_CREATE
  - MPI\_DIMS\_CREATE
  - MPI\_CART\_COORDS
  - MPI\_CART\_RANK
  - MPI\_CART\_SUB
  - MPI\_CARTDIM\_GET
  - MPI\_CART\_GET
  - MPI\_CART\_SHIFT

# Как использовать виртуальные ТОПОЛОГИИ

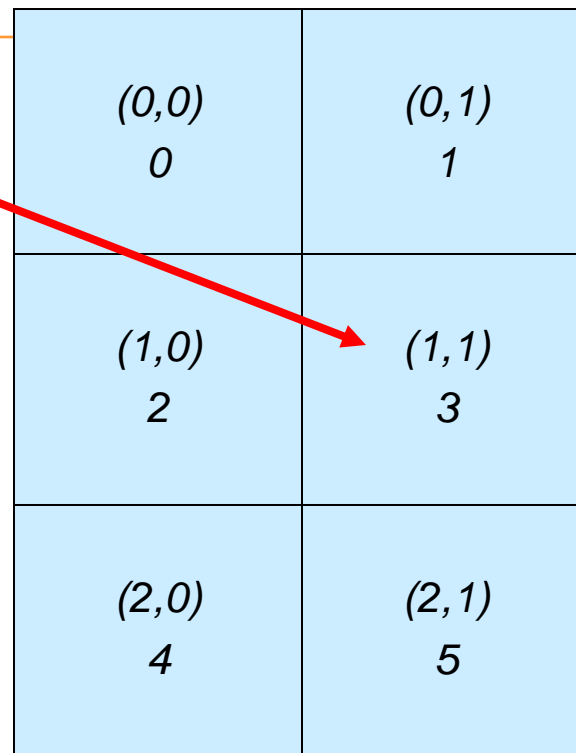
---

- Создание топологии – новый коммуникатор
- MPI обеспечивает “mapping functions”
- Mapping функции вычисляют ранг процессов, базируясь на топологии



# 2D решетка

- Отображает линейно упорядоченный массив в 2-мерную решетку ( 2D Cartesian topology),
- Пример: номер 3 адресуется координатами  $(1, 1)$ .
- Каждая клетка представляет элемент  $3 \times 2$  матрицы.
- Нумерация начинается с 0.
- Нумерация построчная.



$(0,0)$ 0	$(0,1)$ 1
$(1,0)$ 2	$(1,1)$ 3
$(2,0)$ 4	$(2,1)$ 5

# Создание виртуальной топологии решетка

---

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

# Параметры

---

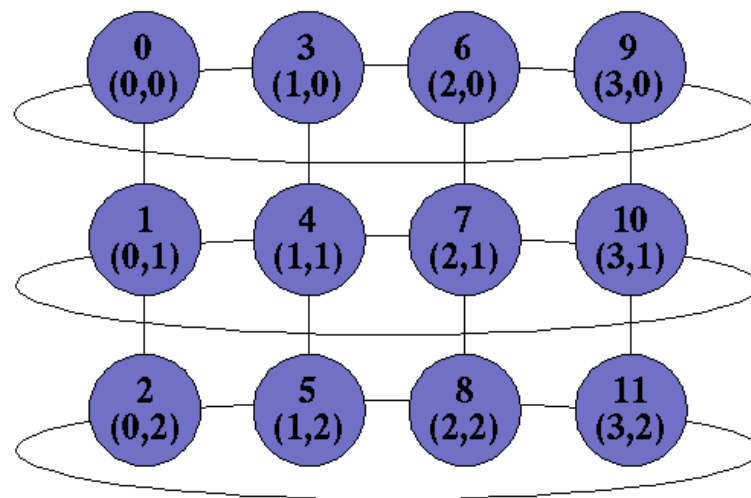
<i>comm_old</i>	старый коммуникатор
<i>ndims</i>	размерность
<i>Periods</i>	логический массив, указывающий на циклическое замыкание: TRUE/FALSE => циклическое замыкание на границе
<i>reorder</i>	возможная перенумерация
<i>comm_cart</i>	новый коммуникатор

# Пример виртуальной топологии решетки

```
MPI_Comm vu;  
int dim[2], period[2], reorder;
```

```
dim[0]=4; dim[1]=3;  
period[0]=TRUE; period[1]=FALSE;  
reorder=TRUE;
```

```
MPI_Cart_create(MPI_COMM_WORLD,2,  
               dim,period,reorder,&vu);
```



# Пример (решетка)

```
#include<mpi.h>
/* Run with 12 processes */
int main(int argc, char *argv[]) {
    int rank;
    MPI_Comm vu;
    int dim[2],period[2],reorder;
    int coord[2];
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD,&rank);
    dim[0]=4; dim[1]=3;
    period[0]=TRUE; period[1]=FALSE;
    reorder=TRUE;
    MPI_Cart_create(MPI_COMM_WORLD,2,dim,period,reorder,&vu);
```

# Координаты процесса в виртуальной решетке

---

```
int MPI_Cart_coords (  
    MPI_Comm comm,           /* Коммуникатор */  
    int rank,                /* Ранг процесса */  
    int numb_of_dims,        /* Размер решетки */  
    int  coords[]            /* координаты процесса в решетке */  
)
```

# MPI\_CART\_RANK

---

*int MPI\_Cart\_rank( MPI\_Comm comm, int \*coords, int \*rank )*

Перевод логических координат процесса в решетке в ранг процесса.

- Если *i*-ое направление размерности периодическое и *i*-ая координата выходит за пределы, значение автоматически сдвигается  $0 < \text{coords}(i) < \text{dims}(i)$ .
- В противном случае - ошибка

# Определение сбалансированного распределения процессов по решетке

int **MPI\_Dims\_create** (int **nnodes**, int **ndims**, int \***dims**)

nnodes - число процессов

ndims - размер решетки

dims - число элементов по измерениям решетки

- Помогает определить сбалансированное распределение процессов по измерениям решетки.
- Если `dims[i]` – положительное целое, это измерение не будет модифицироваться

dims before call	Function call	dims on return
(0, 0)	MPI_DIMS_CREATE(6, 2, dims)	(3, 2)
(0, 0)	MPI_DIMS_CREATE(7, 2, dims)	(7, 1)
(0, 3, 0)	MPI_DIMS_CREATE(6, 3, dims)	(2, 3, 1)
(0, 3, 0)	MPI_DIMS_CREATE(7, 2, dims)	erroneous call



# Пример использования MPI\_Dims\_create

```
— MPI_Comm_size(MPI_COMM_WORLD, &nprocs); —
```

```
int dim[3];  
dim[0] = 0; // let MPI arrange  
dim[1] = 0; // let MPI arrange  
dim[2] = 3; // I want exactly 3 planes
```

```
MPI_Dims_create(nprocs, 3, dim);
```

```
if (dim[0]*dim[1]*dim[2] < nprocs) {  
    fprintf(stderr, "WARNING: some processes are not in use!\n"  
}
```

```
int period[] = {1, 1, 0};  
int reorder = 0;
```

```
MPI_Cart_create(MPI_COMM_WORLD, 3, dim, period, reorder,  
&cube_comm);
```

# MPI\_CART\_RANK

---

*int MPI\_Cart\_rank( MPI\_Comm comm, int \*coords, int \*rank )*

Перевод логических координат процесса в решетке в ранг процесса.

- Если *i*-ое направление размерности периодическое и *i*-ая координата выходит за пределы, значение автоматически сдвигается  $0 < \text{coords}(i) < \text{dims}(i)$ .
- В противном случае - ошибка

# MPI\_CART\_SHIFT

---

- Получение номеров посылающего (**source**) и принимающего (**dest**) процессов в декартовой топологии коммуникатора **comm** для осуществления сдвига вдоль измерения **direction** на величину **disp**.

```
int MPI_Cart_shift( MPI_Comm comm, int direction, int displ, int  
    *source, int *dest )
```

# MPI\_CART\_SHIFT

---

Для периодических измерений осуществляется циклический сдвиг, для непериодических – линейный сдвиг.

Для  $n$ -мерной декартовой решетки значение **direction** должно быть в пределах от 0 до  $n-1$ .

Значения **source** и **dest** можно использовать, например, для обмена функцией **MPI\_Sendrecv**.

# Пример: Sendrecv в 1D решетке

```
...  
int dim[1], period[1];  
dim[0] = nprocs;  
period[0] = 1;  
MPI_Comm ring_comm;
```

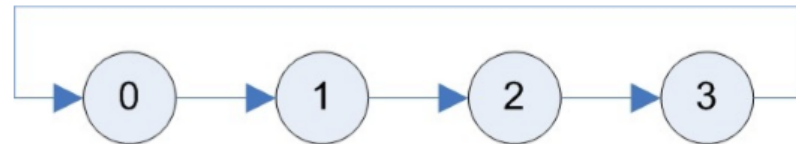
```
MPI_Cart_create(MPI_COMM_WORLD, 1, dim, period, 0, &ring_comm);
```

```
int source, dest;
```

```
MPI_Cart_shift(ring_comm, 0, 1, &source, &dest);
```

```
MPI_Sendrecv(right_boundary, n, MPI_INT, dest, rtag,  
             left_boundary, n, MPI_INT, source, ltag,  
             ring_comm, &status);
```

```
...
```



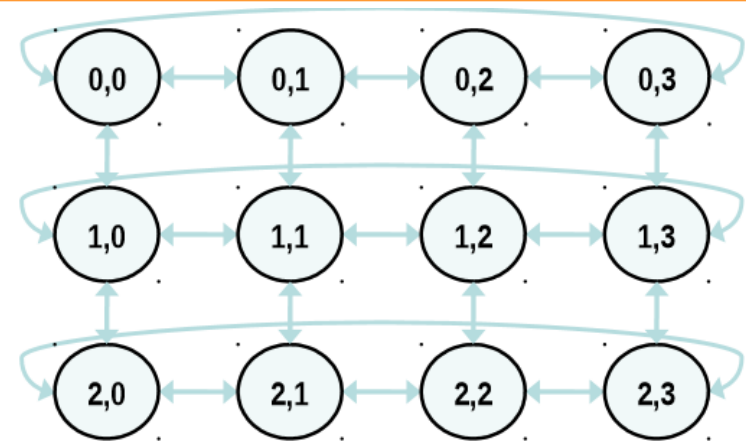
# Пример: Sendrecv в 2D решетке

...

```
int dim[] = {4, 3};  
int period[] = {1, 0};  
MPI_Comm grid_comm;
```

```
MPI_Cart_create(MPI_COMM_WORLD, 2,  
               dim, period, 0, &grid_comm);
```

```
int source, dest;  
for (int dimension = 0; dimension < 2; dimension++) {  
    for (int versus = -1; versus < 2; versus+=2;) {  
        MPI_Cart_shift(ring_comm, dimension, versus, &source, &dest);  
        MPI_Sendrecv(buffer, n, MPI_INT, source, stag,  
                     buffer, n, MPI_INT, dest, dtag,  
                     grid_comm, &status);  
    }  
}
```



# Создание подрешетки

---

```
int MPI_Cart_sub (MPI_Comm comm_old,  
                  int remain_dims[], MPI_Comm *new_comm)
```

```
int dim[] = {2, 3, 4};
```

```
int remain_dims[] = {1, 0, 1}; // 3 comm with 2x4 processes 2D grid
```

```
...
```

```
int remain_dims[] = {0, 0, 1}; // 6 comm with 4 processes 1D topology
```

# MPI\_CARTDIM\_GET

---

- Определение числа измерений в решетке.

```
int MPI_Cartdim_get( MPI_Comm comm, int* ndims )
```

- comm       коммуникатор (решетка)
- ndims       число измерений



# Пример декартовой решетки (send&recv, mesh)

```
MPI_Request reqs[8];
MPI_Status stats[8];
MPI_Comm cartcomm;

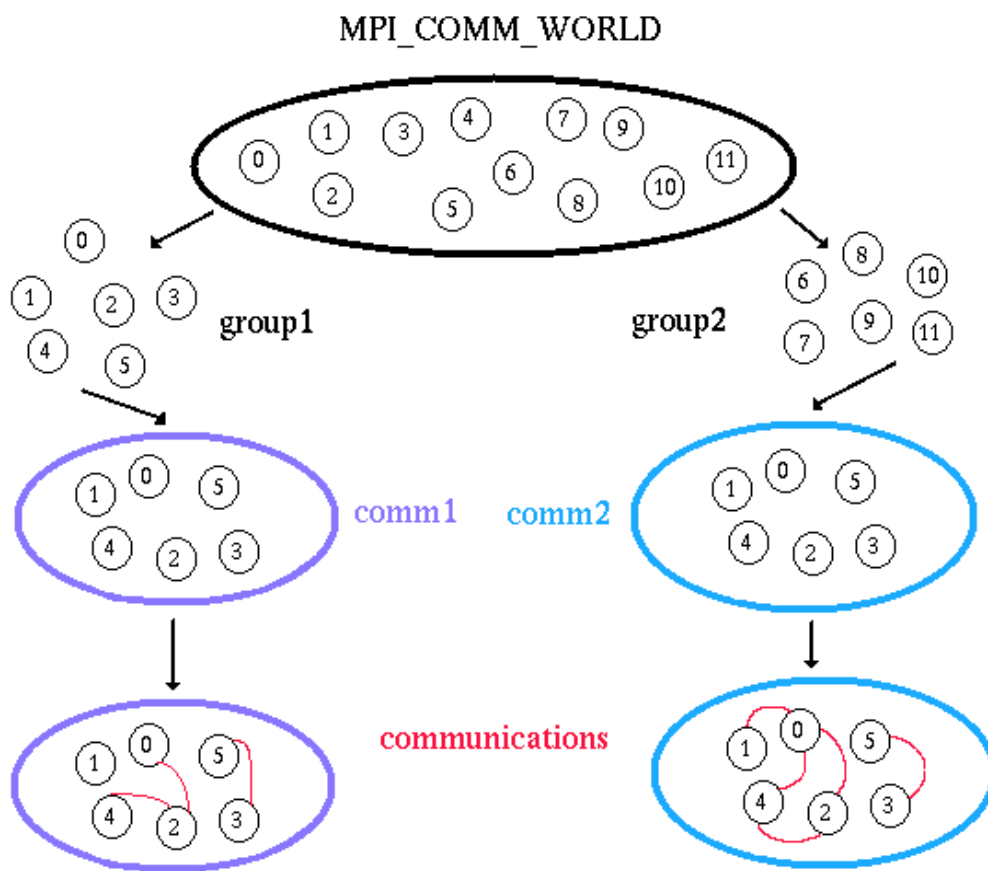
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD, &numtasks);

if (numtasks == SIZE) {
    MPI_Cart_create(MPI_COMM_WORLD, 2, dims,
periods, reorder, &cartcomm);
    MPI_Comm_rank(cartcomm, &rank);
    MPI_Cart_coords(cartcomm, rank, 2, coords);
    MPI_Cart_shift(cartcomm, 0, 1, &nbrs[UP],
&nbrs[DOWN]);
    MPI_Cart_shift(cartcomm, 1, 1, &nbrs[LEFT],
&nbrs[RIGHT]);

    outbuf = rank;
```

```
for (i=0; i<4; i++) {
    dest = nbrs[i];
    source = nbrs[i];
    MPI_Isend(&outbuf, 1, MPI_INT, dest, tag,
MPI_COMM_WORLD, &reqs[i]);
    MPI_Irecv(&inbuf[i], 1, MPI_INT, source, tag,
MPI_COMM_WORLD, &reqs[i+4]);
}
MPI_Waitall(8, reqs, stats);
printf("rank= %d coords= %d %d
neighbors(u,d,l,r)= %d %d %d %d inbuf(u,d,l,r)=
%d %d %d %d\n",
rank,coords[0],coords[1],nbrs[UP],nbrs[DOWN],n
brs[LEFT],inbuf[UP],inbuf[DOWN],inbuf[LEFT],in
buf[RIGHT]);
}
else
    printf("Must specify %d tasks.
Terminating.\n",SIZE);
MPI_Finalize();
}
```

# Группы и коммутаторы

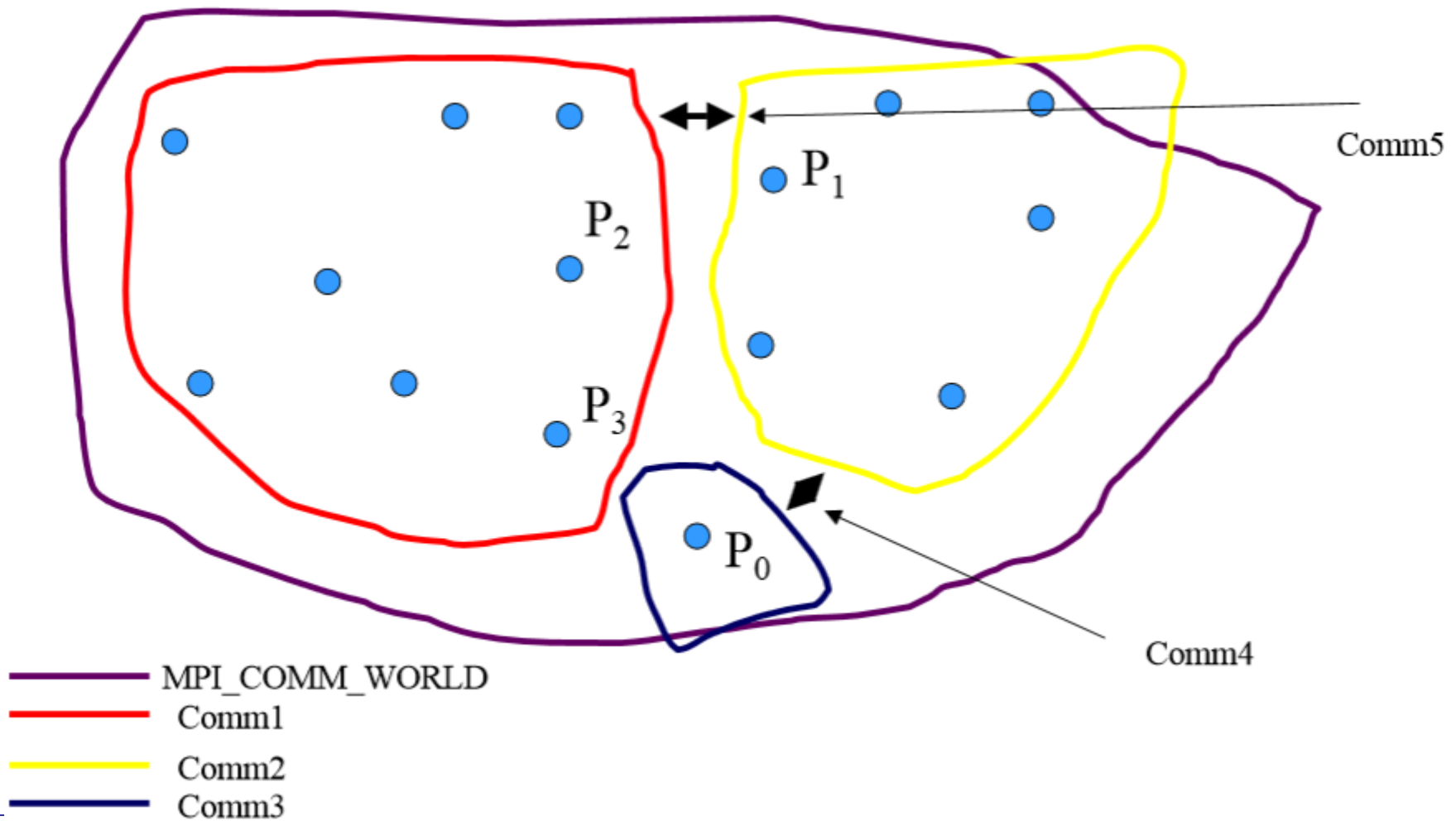


# Группы и коммутаторы

---

- Группа:
  - Упорядоченное множество процессов
  - Каждый процесс в группе имеет уникальный номер
  - Процесс может принадлежать нескольким группам
    - rank всегда относителен группы
- Коммутаторы:
  - Все обмены сообщений всегда проходят в рамках коммутатора
  - С точки зрения программирования группы и коммутаторы эквивалентны
- Группы и коммутаторы – динамические объекты, должны создаваться и уничтожаться в процессе работы программы

# Типы коммуникаторов



# Типы коммуникаторы

---

- Intercommunicator
  - Обмены (только 2-ухточеченные) между процессами из разных коммуникаторов
- Intracommunicator:
  - Все обмены сообщений всегда проходят в рамках одного коммуникатора

Коммуникатор может быть только одного типа: либо inter, либо intra !

# Создание новых коммуникаторов

---

2 способа создания новых коммуникаторов:

- Использовать функции для работы с группами и коммуникаторами (создать новую группу процессов и по новой группе создать коммуникатор, разделить коммуникатор и т.п.)
- Использовать виртуальные топологии

# Типичный шаблон работы

---

1. Извлечение глобальной группы из коммуникатора `MPI_COMM_WORLD`, используя функцию `MPI_Comm_group`
2. Формирование новой группы как подмножества глобальной группы, используя `MPI_Group_incl` или `MPI_Group_excl`
3. Создание новый коммуникатор для новой группы, используя `MPI_Comm_create`
4. Определение номера процесса в новом коммуникаторе, используя `MPI_Comm_rank`
5. Обмен сообщениями, используя функции MPI
6. По окончании освобождение созданных коммуникатора и группы, используя `MPI_Comm_free` и `MPI_Group_free`

```

main(int argc, char **argv) {
    int me, count, count2;
    void *send_buf, *recv_buf, *send_buf2, *recv_buf2;
    MPI_Group MPI_GROUP_WORLD, grprem;
    MPI_Comm commslave;
    static int ranks[] = {0};
    MPI_Init(&argc, &argv);
    MPI_Comm_group(MPI_COMM_WORLD, &MPI_GROUP_WORLD);
    MPI_Comm_rank(MPI_COMM_WORLD, &me);
    MPI_Group_excl(MPI_GROUP_WORLD, 1, ranks, &grprem);
    MPI_Comm_create(MPI_COMM_WORLD, grprem, &commslave);
    if(me != 0){ /* compute on slave */
        MPI_Reduce(send_buf,recv_buf,count, MPI_INT, MPI_SUM, 1,
                    commslave);
    }
    /* zero falls through immediately to this reduce, others do later... */
    MPI_Reduce(send_buf2, recv_buf2, count2, MPI_INT, MPI_SUM, 0,
                MPI_COMM_WORLD);
    MPI_Comm_free(&commslave);
    MPI_Group_free(&MPI_GROUP_WORLD);
    MPI_Group_free(&grprem);
    MPI_Finalize();
}

```



# Специальные типы MPI

---

- MPI\_Comm
  - MPI\_COMM\_WORLD – коммуникатор для всех процессов приложения.
  - MPI\_COMM\_NULL – значение, используемое для ошибочного коммуникатора.
  - MPI\_COMM\_SELF – коммуникатор, включающий только вызвавший процесс.
- MPI\_group
  - MPI\_GROUP\_EMPTY – пустая группа.
  - MPI\_GROUP\_NULL – значение, используемое для ошибочной группы

# Количество процессов в группе

---

- Размер группы (число процессов в группе)

```
int MPI_Group_size(MPI_Group comm, int *size)
```

Результат – число процессов

Если указать MPI\_GROUP\_EMPTY, то size=0

# Номер процесса в группе

---

- Номер процесса в группе

```
int MPI_Group_rank (MPI_Group comm, int *rank)
```

Результат – номер процессов или MPI\_UNDEFINED

# Определение группы по коммуникатору

---

- Группа по коммуникатору

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

*Пример:*

```
MPI_Group commGroup;
```

```
MPI_Comm_group (MPI_COMM_WORLD, &commGroup);
```

# Включение процессов в группу

## ■ Включение процессов в группу

*int MPI\_Group\_incl(MPI\_Group comm, , int n, int \*ranks, MPI\_Group \*newgroup)*

*n* – число процессов в новой группе

*ranks* – номера процессов в группе *group*, которые будут составлять группу *newgroup* (выходной параметр) ;

*newgroup* – новая группа, составленная из процессов из *ranks*, в порядке, определенном *ranks* (выходной параметр) .

В случае *n=0* *MPI\_Group\_incl* вернет *MPI\_GROUP\_EMPTY*.

Функция может применяться для перенумерации процессов в группе.

# Исключение процессов из группы

- Номер процесса в группе

*int MPI\_Group\_excl(MPI\_Group oldgroup, , int n, int \*ranks, MPI\_Group \*newgroup)*

*n* - число процессов в массиве ranks

*ranks* – номера процессов в группе oldgroup, которые будут исключаться из группы oldgroup;

*newgroup* – новая группа , не содержащая процессов с номерами из ranks, порядок процессов такой же, как в группе group (выходной параметр).

Каждый из n процессов с номерами из массива ranks должен существовать, иначе функция вернет ошибку. В случае n=0 MPI\_Group\_excl вернет группу group.

# Сравнение групп процессов

---

```
int MPI_Group_compare(MPI_Group group1,  
MPI_Group group2, int *result)
```

**MPI\_Group\_compare** возвращает результат сравнения двух групп:

**MPI\_IDENT** – состав и порядок одинаковые в обеих группах;

**MPI\_SIMILAR** – обе группы содержат одинаковые процессы, но их порядок в группах разный;

**MPI\_UNEQUAL** – различные состав и порядок групп.

# Трансляция номеров процессов между группами

---

```
int MPI_Group_translate_ranks ( MPI_Group group_a,  
    int n, int *ranks_a, MPI_Group group_b, int  
    *ranks_b )
```

Функция возвращает список номеров процессов из группы *group\_a* в их номера в группе *group\_b*

MPI\_UNDEFINED возвращается для процессов, которых нет в *group\_b*



# Создание коммуникатора по группе

```
int MPI_Comm_create (MPI_Comm comm, MPI_Group group,  
MPI_Comm *newcomm)
```

***comm*** – коммуникатор;

***group*** – группа, представляющая собой подмножество процессов, ассоциированное с коммуникатором *comm*;

***newcomm*** – новый коммуникатор (выходной параметр).

Функция `MPI_Comm_create` создает новый коммуникатор, с которым ассоциирована группа *group*. Функция возвращает `MPI_COMM_NULL` процессам, не входящим в *group*.

`MPI_Comm_create` завершится с ошибкой, если не все аргументы *group* будут одинаковыми в различных вызывающих функцию процессах, или если *group* не является подмножеством группы, ассоциированной с коммуникатором *comm*. **Вызвать функцию должны все процессы, входящие в *comm*, даже если они не принадлежат новой группе.**

# Создание нескольких коммуникаторов

---

```
int MPI_Comm_split(MPI_Comm comm, int color, int key, MPI_Comm  
*newcomm)
```

***comm*** - коммуникатор;

***color*** - признак разделения на группы;

***key*** - параметр, определяющий нумерацию в новых коммуникаторах;

***newcomm*** – новый коммуникатор (выходной параметр).

# MPI\_Comm\_split

---

Функция разбивает все множество процессов, входящих в коммуникатор `comm`, на непересекающиеся подгруппы - одну подгруппу на каждое значение параметра ***color*** (неотрицательное число).

Каждая новая подгруппа содержит все процессы одного цвета. Если в качестве ***color*** указано значение `MPI_UNDEFINED`, то в `newcomm` будет возвращено значение `MPI_COMM_NULL`.

Это **коллективная функция**, но каждый процесс может указывать свои значения для параметров `color` и `key`..

# MPI\_Comm\_split

---

Значение **color** определяет порядок нумерации процессов в новом коммуникаторе:

- процессы с меньшим значением **color** получают меньший rank в новом коммуникаторе;
- если значение **color** одинаково, то нумерация процессов в новом коммуникаторе будет определяться порядком следования в исходном коммуникаторе.

# Пример

---

Rank	0	1	2	3	4	5	6	7	8	9	10
Process	a	b	c	d	e	f	g	h	i	j	k
Color	U	3	1	1	3	7	3	3	1	U	3
Key	0	1	2	3	1	9	3	8	1	0	0

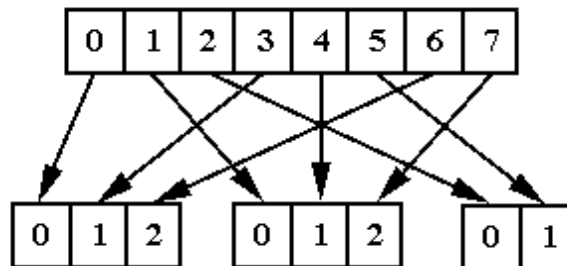
Будет создано 3 группы процессов:  
{l,c,d}, {k,b,e,g,h}, {f}

Процессы a и j получают значение MPI\_COMM\_NULL

# Пример MPI\_Comm\_split

---

```
MPI_comm comm, newcomm;  
int myid, color; . . . . .  
MPI_Comm_rank(comm, &myid);  
color = myid%3;  
MPI_Comm_split(comm, color, myid, &newcomm);
```



# Пример: вычисление числа $P_i$

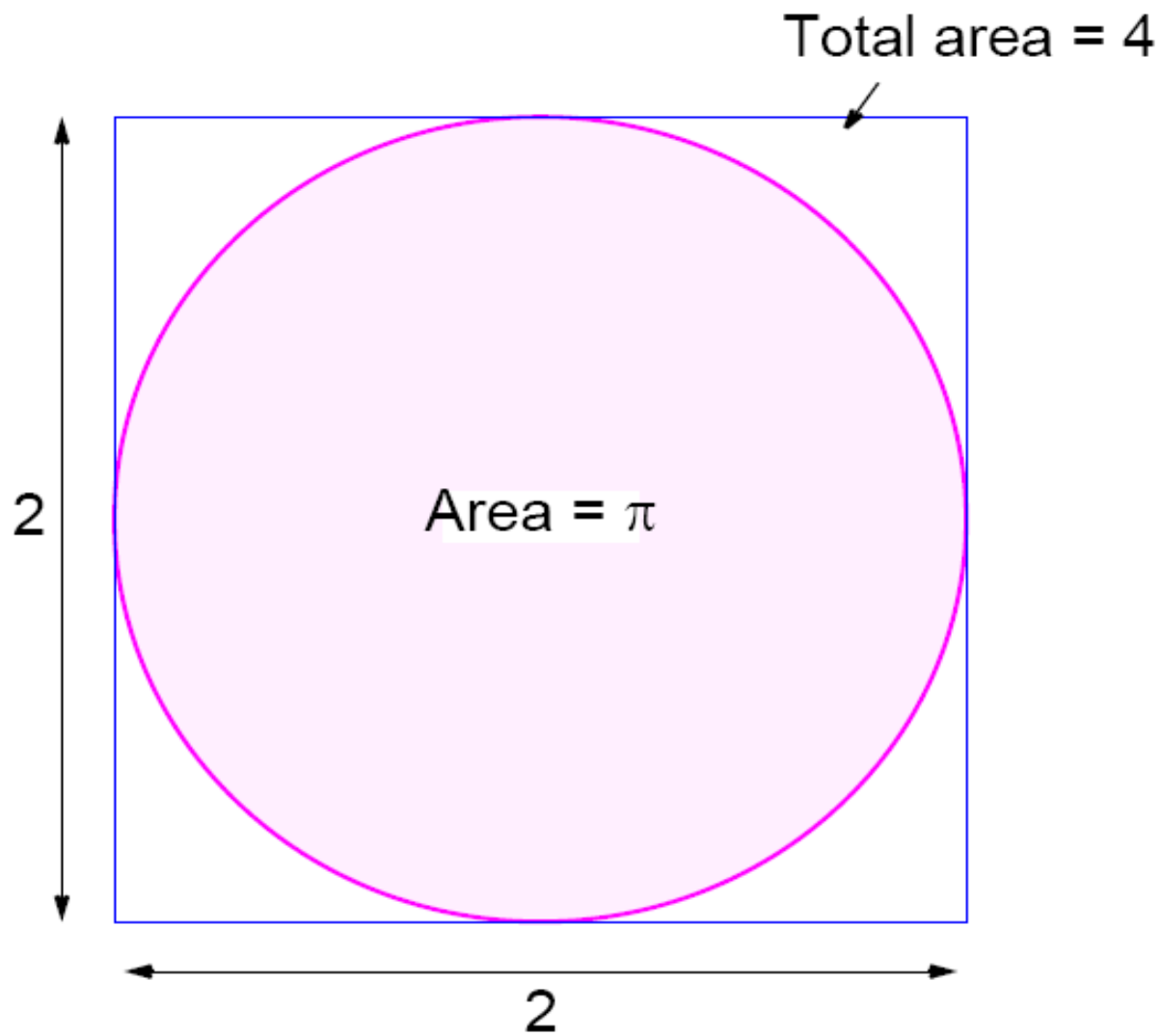
---

Окружность вписывается в квадрат  $2 \times 2$ . Отношение площадей:

$$\frac{\text{Area of circle}}{\text{Area of square}} = \frac{\pi(1)^2}{2 \times 2} = \frac{\pi}{4}$$

Точки внутри квадрата выбираются случайно

**Отношение числа случайно выбранных точек, попавших в квадрат к числу точек, попавших в круг равно  $P_i$ .**





# Монте Карло метод для интегрирования произвольных функций

---

Вычисление случайного значения  $x$  для вычисления  $f(x)$  и суммы  $f(x)$ :

$$\text{Area} = \int_{x_1}^{x_2} f(x) dx = \lim_{N \rightarrow \infty} \frac{1}{N} \sum_{i=1}^N f(x_i)(x_2 - x_1)$$

$x_i$  – случайно сгенерированные значения  $x$ , принадлежащие отрезку  $[x_1, x_2]$ .

Метод Монте Карло полезен для вычисления функций, которые не могут быть проинтегрированы численно (многомерных функций)

# Вычисление числа $\pi$ (1)

```
/* compute pi using Monte Carlo method */  
#include <stdio.h>  
#include <limits.h>  
#include <stdlib.h>  
#include <math.h>  
#include "mpi.h"  
#define CHUNKSIZE    1000  
/* message tags */  
#define REQUEST  1  
#define REPLY    2
```

# Вычисление числа $\Pi$ (2)

---

```
int main(int argc, char *argv[])

    int iter;
    int in, out, i, iters, max, ix, iy, ranks[1], done, temp;
    double x, y, Pi, error, epsilon;
    int numprocs, myid, server, totalin, totalout, workerid;
    int rands[CHUNKSIZE], request;
    MPI_Comm world, workers;
    MPI_Group world_group, worker_group;
    MPI_Status status;
```

# Вычисление числа $\pi$ (3)

```
MPI_Init(&argc, &argv);
world = MPI_COMM_WORLD;
MPI_Comm_size(world, &numprocs);
MPI_Comm_rank(world, &myid);
server = numprocs-1;    /* last proc is server */
if (myid == 0) {
    if (argc < 2) {
        fprintf(stderr, "Usage: %s epsilon\n", argv[0] );
        MPI_Abort(MPI_COMM_WORLD, 1);
    }
    sscanf( argv[1], "%lf", &epsilon );
}
MPI_Bcast(&epsilon, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
```

# Вычисление числа $\Pi$ (4)

---

```
MPI_Comm_group(world, &world_group);  
ranks[0] = server;  
MPI_Group_excl(world_group, 1, ranks, &worker_group);  
MPI_Comm_create(world, worker_group, &workers);  
MPI_Group_free(&worker_group);
```

# Вычисление числа $\pi$ (5)

```
if (myid == server) {          /* I am the rand server */
    do {
        MPI_Recv(&request, 1, MPI_INT, MPI_ANY_SOURCE, REQUEST,
                world, &status);
        if (request) {
            for (i = 0; i < CHUNKSIZE; ) {
                rands[i] = random();
                if (rands[i] <= INT_MAX) i++;
            }
            MPI_Send(rands, CHUNKSIZE, MPI_INT,
                    status.MPI_SOURCE, REPLY, world);
        }
    } while(request > 0);
}
```

# Вычисление числа $\pi$ (6)

```
}  
else {                                /* I am a worker process */  
    request = 1;  
    done = in = out = 0;  
    max = INT_MAX;                    /* max int, for normalization */  
    MPI_Send(&request, 1, MPI_INT, server, REQUEST, world);  
    MPI_Comm_rank(workers, &workerid);  
    iter = 0;  
    while (!done) {  
        iter++;  
        request = 1;  
        MPI_Recv(rands, CHUNKSIZE, MPI_INT, server, REPLY,  
                 world, MPI_STATUS_IGNORE);
```