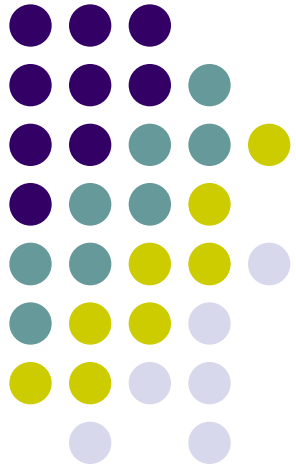
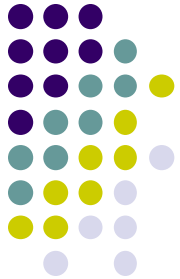
Message Passing Interface: Краткий обзор

Киреев Сергей
ИВМиМГ СО РАН



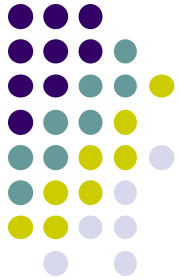
Параллельное программирование



- Параллельные программы – программы, которые имеют параллельно работающие части.

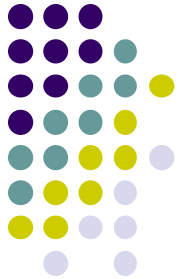
- Средства параллельного программирования
  - Библиотеки
    - MPI, PVM, PThreads, shmem, …
  - Расширения «последовательных» языков
    - OpenMP, Cilk+, UPC, HPF, …
  - «Параллельные» языки
    - X10, Chapel, Occam, Erlang, …

Параллельное программирование



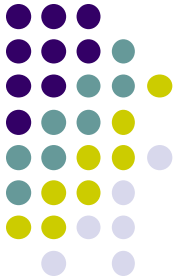
- Лекции
  - Формальные основы параллельного программирования
  - Математические модели параллельных программ: взаимодействующие последовательные процессы, сети Петри, асинхронные программы
  - Синхронизация параллельных процессов: базовые средства, классические проблемы синхронизации
  - Проблемы создания параллельных программ: отображение на ресурсы, балансировка загрузки ресурсов
- Практика
  - MPI
  - OpenMP

MPI (Message Passing Interface)



- MPI – стандарт интерфейса обмена сообщениями между параллельно работающими процессами
  - Текущая версия стандарта: MPI 3.0
- Существуют много реализаций стандарта MPI:
  - Открытые: MPICH, OpenMPI, MVAPICH, LAM/MPI, …
  - Коммерческие: Intel MPI, HP-MPI, SGI MPI, …
- MPI представляет собой библиотеку подпрограмм для языков C (C++) и Fortran
  - Есть реализации для Java: MPJ, MPJ Express

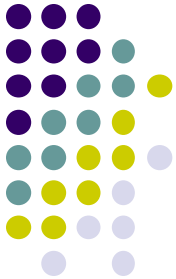
MPI: основы



- Программа «Hello, World!» с использованием MPI

```
#include<mpi.h>                          // Подключение библиотеки MPI
#include<stdio.h>
int main(int argc,char *argv[])
{ int size,rank;
  MPI_Init(&argc,&argv);                 // Инициализация MPI
  MPI_Comm_size(MPI_COMM_WORLD,&size);   // Получение числа процессов
  MPI_Comm_rank(MPI_COMM_WORLD,&rank);   // Получение номера процесса
  printf("Hello from process #%d of %d\n",rank,size);
  MPI_Finalize();                        // Завершение работы MPI
  return 0;
}
```

MPI: основы



- Программа «Hello, World!» с использованием MPI

```
#include<mpi.h>                                  // Подключение библиотеки MPI
#include<stdio.h>
int main(int argc,char *argv[])
{ int size,rank;
  MPI_Init(&argc,&argv);                         // Инициализация MPI
  MPI_Comm_size(MPI_COMM_WORLD,&size);           // Получение числа процессов
  MPI_Comm_rank(MPI_COMM_WORLD,&rank);           // Получение номера процесса
  printf("Hello from process #%d of %d\n",rank,size);
  MPI_Finalize();                                // Завершение работы MPI
  return 0;
}
```

- Компиляция MPI-программ:

  C: `mpicc -o prog prog.c`

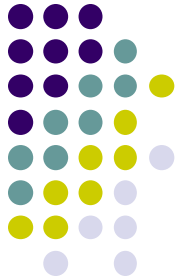
  C++: `mpicxx -o prog prog.cpp`

- Запуск MPI-программ:

  `mpirun -np 4 ./prog`

  или `mpiexec -n 4 ./prog`

MPI: основы



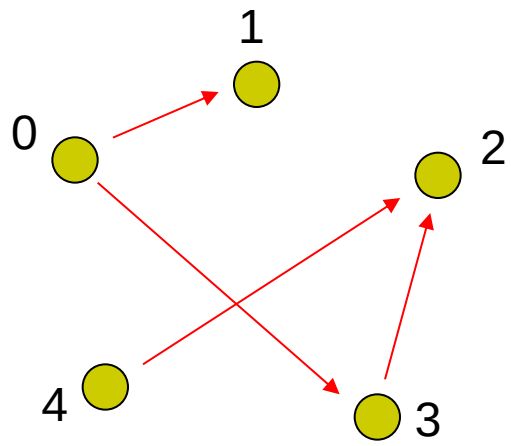
- При запуске MPI-программы запускается указанное число копий программы на указанных ресурсах.

- Примеры:

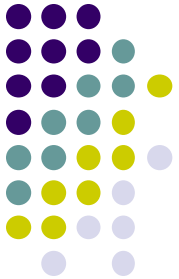
```
>mpirun -np 3 hostname
server
server
server

>mpirun -np 5 ./prog
Hello from process #3 of 5
Hello from process #0 of 5
Hello from process #1 of 5
Hello from process #4 of 5
Hello from process #2 of 5
```

Модель MPI-программы

- MPI-программа – это множество параллельно работающих процессов. Каждый процесс имеет свои код и данные.
- Каждый процесс имеет уникальный номер от 0 до N-1 (N – число процессов)
- Процессы могут в произвольные моменты времени передавать друг другу сообщения



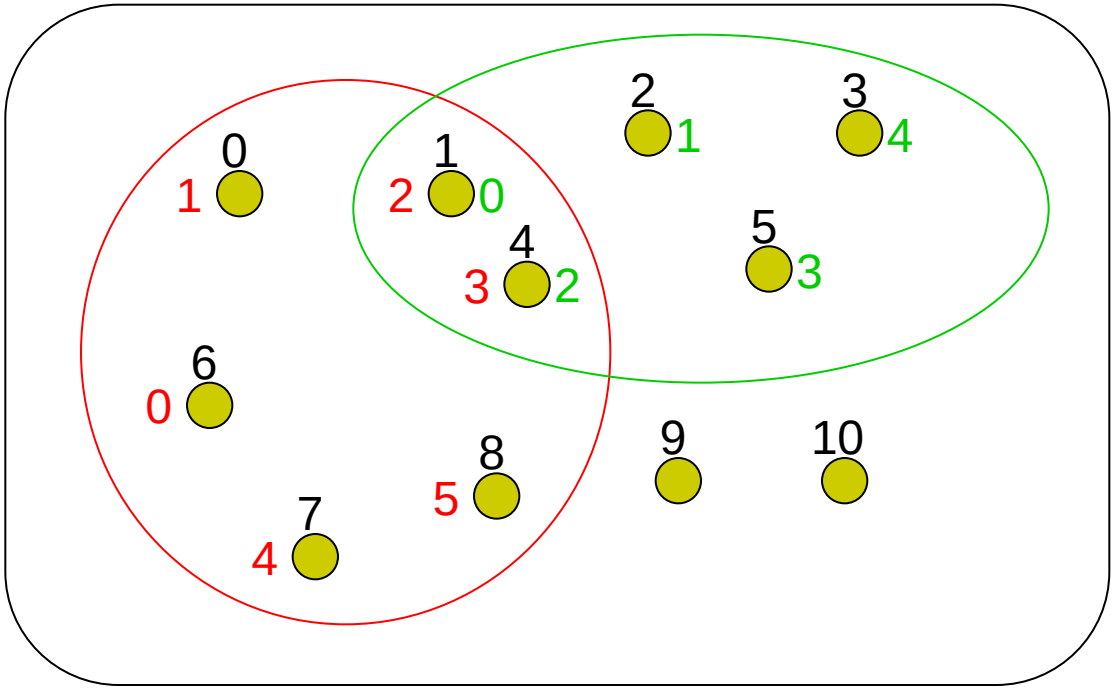
Стандарт MPI включает



- Коммуникации точка-точка
- Коллективные коммуникации
- Односторонние коммуникации
- Типы данных
- Группы процессов, топологии, коммуникаторы
- Динамическое управление процессами
- Средства профилирования

Группы процессов, топологии, коммуникаторы

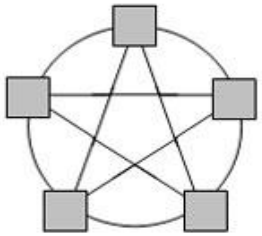
- MPI-процессы могут объединяться в группы
- Внутри каждой группы MPI-процессы имеют свою нумерацию



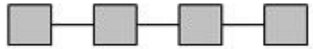
Группы процессов, топологии, коммуникаторы

- Для группы MPI-процессов можно задать виртуальную топологию сети связи
- Виртуальная топология используется для наилучшего отображения на топологию реальной сети

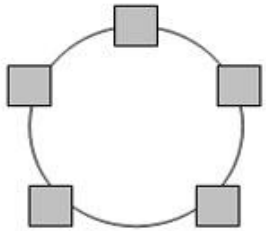
- Примеры топологий:



1) Полный граф

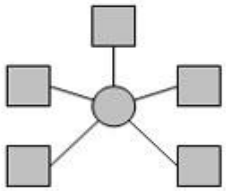


2) Линейка

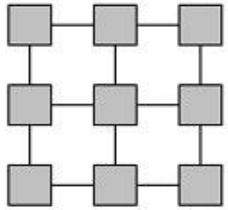


3) Кольцо

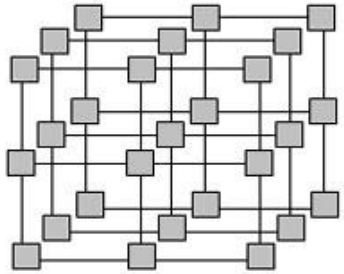
- Типы топологий MPI:
  - Декартова топология
  - Произвольный граф



4) Звезда

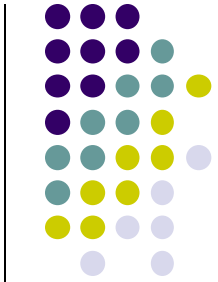


5) 2-мерная решетка



6) 3-мерная решетка

Группы процессов, топологии, коммуникаторы

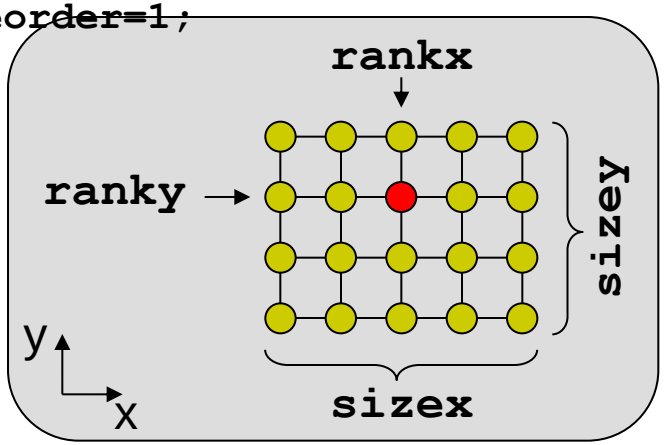


- Коммуникатор – это группа процессов с заданной на ней топологией сети связи
- Пример: MPI_COMM_WORLD
  - Включает все запущенные MPI-процессы
  - Топология сети – полный граф
- В операциях передачи сообщений используются именно коммуникаторы

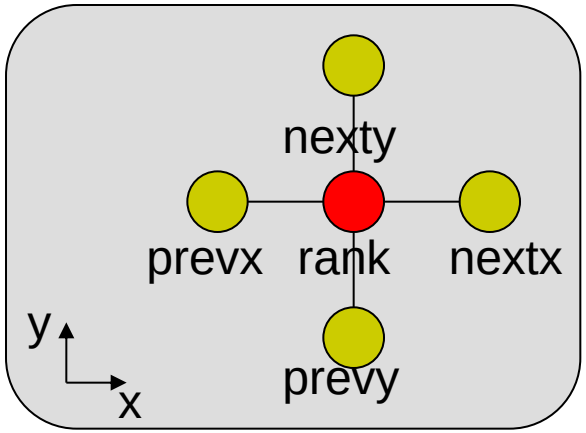
Группы процессов, коммуникаторы, топологии

- **Пример: создание коммуникатора с топологией двумерная решетка**

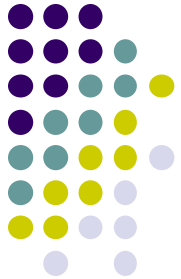
```
int dims[2]={0,0},periods[2]={0,0},coords[2],reorder=1;
int size,rank,sizey,sizex,ranky,rankx;
int prevy,prevx,nexty,nextx;
MPI_Comm comm2d; // коммуникатор
MPI_Comm_size(MPI_COMM_WORLD,&size);
```




```
// определение размеров решетки: dims
MPI_Dims_create(size,2,dims);
sizey = dims[0]; sizex = dims[1];
// создание коммуникатора: comm2d
MPI_Cart_create(MPI_COMM_WORLD,2,dims,periods,reorder,&comm2d);
// получение своего номера в comm2d: rank
MPI_Comm_rank(comm2d,&rank);
// получение своих координат в двумерной решетке: coords
MPI_Cart_get(comm2d,2,dims,periods,coords);
ranky=coords[0]; rankx=coords[1];
// определение номеров соседей: prevy, nexty, prevx, nextx
MPI_Cart_shift(comm2d,0,1,&prevy,&nexty);
MPI_Cart_shift(comm2d,1,1,&prevx,&nextx);
```

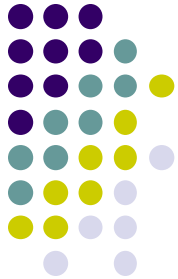


Группы процессов, коммуникаторы, топологии



- Стандарт MPI 1.0
  - Допускаются только подмножества MPI_COMM_WORLD
- Стандарт MPI 2.0 и дальше
  - Запуск новых процессов
  - Взаимодействие групп процессов
    - **интра**коммуникаторы
    - **интер**коммуникаторы

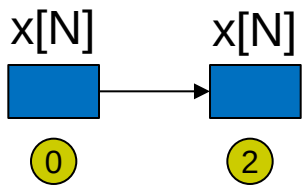
Передача сообщений точка-точка



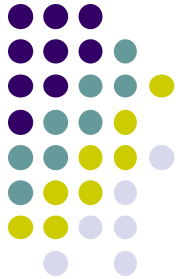
- Базовые операции
  - MPI_Send – отправка сообщения
  - MPI_Recv – прием сообщения

- Пример: передача данных от 0-го процесса 2-му

```
double x[N];
if (size>=3)
{ if (rank==0)
    MPI_Send(x,N,MPI_DOUBLE,2,123,comm);
  if (rank==2)
    MPI_Recv(x,N,MPI_DOUBLE,0,123,comm,MPI_STATUS_IGNORE);
}
```

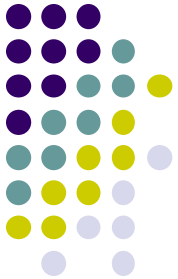


Передача сообщений точка-точка



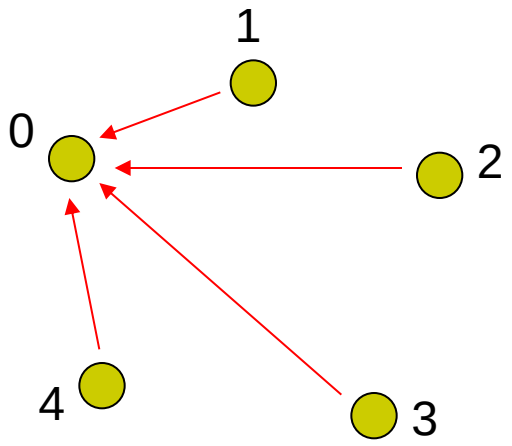
- Варианты отправки сообщения
  - MPI_Send
  - MPI_Ssend – синхронная отправка
  - MPI_Rsend – отправка по готовности парного Recv
  - MPI_Bsend – отправка через пользовательский буфер
    - MPI_Buffer_attach (buffer, size); – подключить буфер
    - MPI_Buffer_detach (buffer, size); – отключить буфер
- Приём сообщения
  - MPI_Recv
- Проверка сообщения без приёма
  - MPI_Probe

Передача сообщений точка-точка

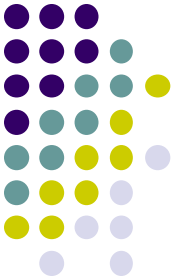


- Пример: приём произвольного сообщения 0-м процессом

```
MPI_Status st;
char x[1000];
if (size>1)
{ if (rank==0)
  { int i,idx = 0;
    for (i=1;i<size;i++)
    { MPI_Probe(MPI_ANY_SOURCE, MPI_ANY_TAG, comm, &st);
      MPI_Recv(x+idx, st.count, MPI_BYTE,
               st.MPI_SOURCE, st.MPI_TAG, comm, MPI_STATUS_IGNORE);
      idx += st.count;
    }
  }
  else
    MPI_Send(x,rank,MPI_CHAR,0,rank,comm);
}
```



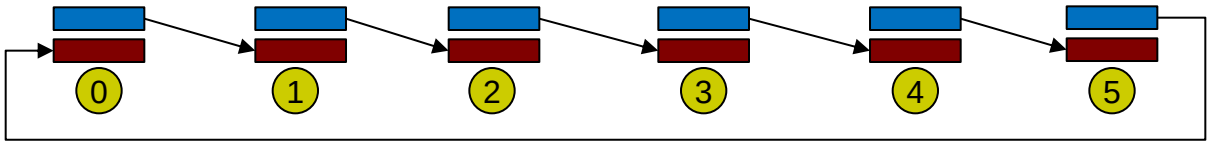
Передача сообщений точка-точка



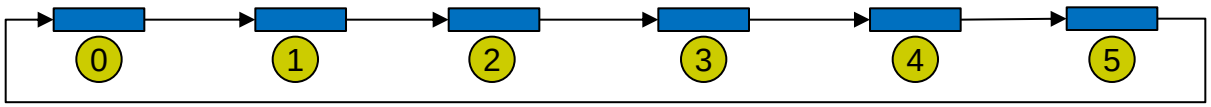
- Совмещение отправки и приёма:
  - MPI_Sendrecv = MPI_Send + MPI_Recv
  - MPI_Sendrecv_replace – используется один буфер

- Пример: сдвиг данных по кольцу

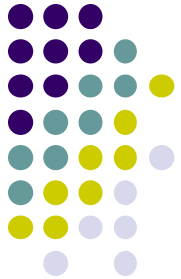
```
double x[N],y[N];
MPI_Sendrecv(x,N,MPI_DOUBLE,(rank+1)%size,123,
             y,N,MPI_DOUBLE,(rank+size-1)%size,123,
             MPI_COMM_WORLD,MPI_STATUS_IGNORE);
```




```
MPI_Sendrecv_replace(x,N,MPI_DOUBLE,
             (rank+1)%size,123, (rank+size-1)%size,123,
             MPI_COMM_WORLD,MPI_STATUS_IGNORE);
```



Передача сообщений точка-точка

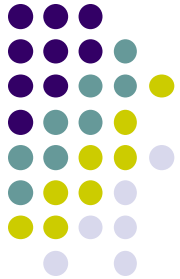


- Асинхронная передача сообщений. Базовые операции:
  - MPI_Isend – запустить отправку сообщения
  - MPI_Irecv – запустить приём сообщения
  - MPI_Test – проверить завершение операции
  - MPI_Wait – дождаться завершения операции

- Пример: передача по кольцу на фоне счёта

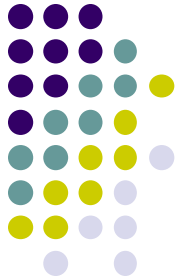
```
int x[N],y[N],done;
MPI_Request reqs,reqr;
MPI_Irecv(x,N,MPI_INT, (rank+size-1)%size,123, comm, &reqr);
MPI_Isend(y,N,MPI_INT, (rank+1)%size,123, comm, &reqs);
do
{ do_some_work();
  MPI_Test(reqr, &done, MPI_STATUS_IGNORE);
} while (!done);
MPI_Wait(reqs, MPI_STATUS_IGNORE);
```

Передача сообщений точка-точка



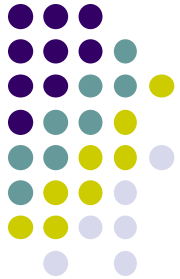
- Асинхронная передача сообщений
  - Запуск отправки сообщения
    - **`MPI_Isend, MPI_Issend, MPI_Irsend, MPI_Ibsend`**
  - Запуск приёма сообщения
    - **`MPI_Irecv`**
  - Запуск проверки сообщения
    - **`MPI_Iprobe`**
  - Проверка завершения операции (группы операций)
    - **`MPI_Test, MPI_Testall, MPI_Testany, MPI_Testsome, MPI_Test_cancelled`**
  - Ожидание завершения операции (группы операций)
    - **`MPI_Wait, MPI_Waitall, MPI_Waitany, MPI_Waitsome`**
  - Отмена операции
    - **`MPI_Cancel`**

Передача сообщений точка-точка



- Многократно повторяющиеся операции
  - Подготовка к отправке
    - **`MPI_Send_init, MPI_Ssend_init, MPI_Rsend_init, MPI_Bsend_init`**
  - Подготовка к приёму
    - **`MPI_Recv_init`**
  - Запуск операции (группы операций)
    - **`MPI_Start, MPI_Start_all`**

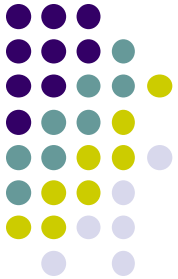
Передача сообщений точка-точка



- Многократно повторяющиеся операции. Пример:

```
MPI_Request req[2];
int x[N],y[N],i;

MPI_Send_init(x,N,MPI_INT,(rank+1)%size,123,comm,&req[0]);
MPI_Recv_init(y,N,MPI_INT,(rank+size-1)%size,
                                        123,comm,&req[1]);
for (i=0;i<100;i++)
{
  do_some_work();
  MPI_Startall(2, req);
  do_more_work();
  MPI_Waitall(2, req, MPI_STATUSES_IGNORE);
}
```

Коллективные операции



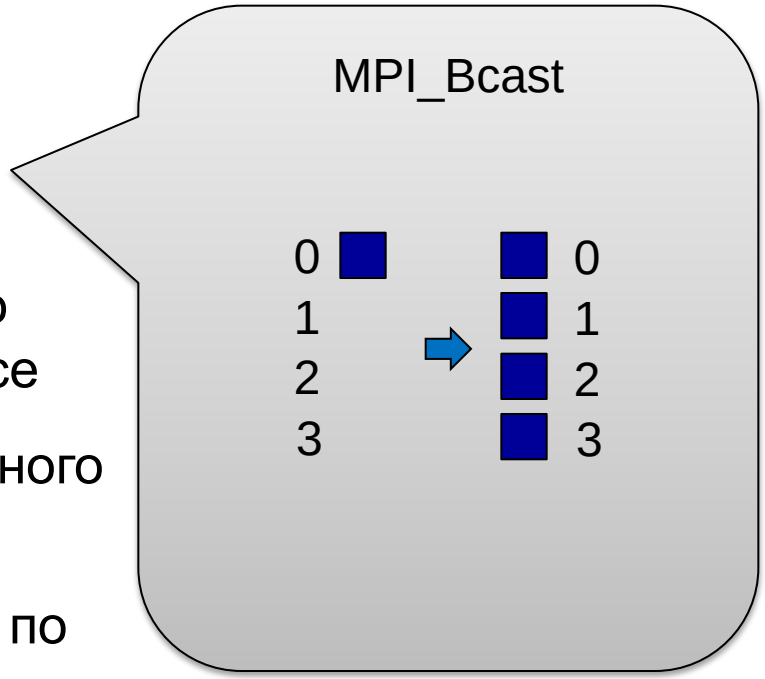
- Выполняются всеми процессами группы
- Базовые операции:

- **MPI_Barrier** – барьерная синхронизация
- **MPI_Bcast** – рассылка данных от одного процесса всем остальным
- **MPI_Gather** – сборка распределенных по всем процессам данных в одном процессе
- **MPI_Scatter** – распределение данных одного процесса между всеми процессами
- **MPI_Allgather** – сборка распределенных по всем процессам данных во всех процессах
- **MPI_Alltoall** – обмен данными «все со всеми»

Коллективные операции

- Выполняются всеми процессами группы
- Базовые операции:

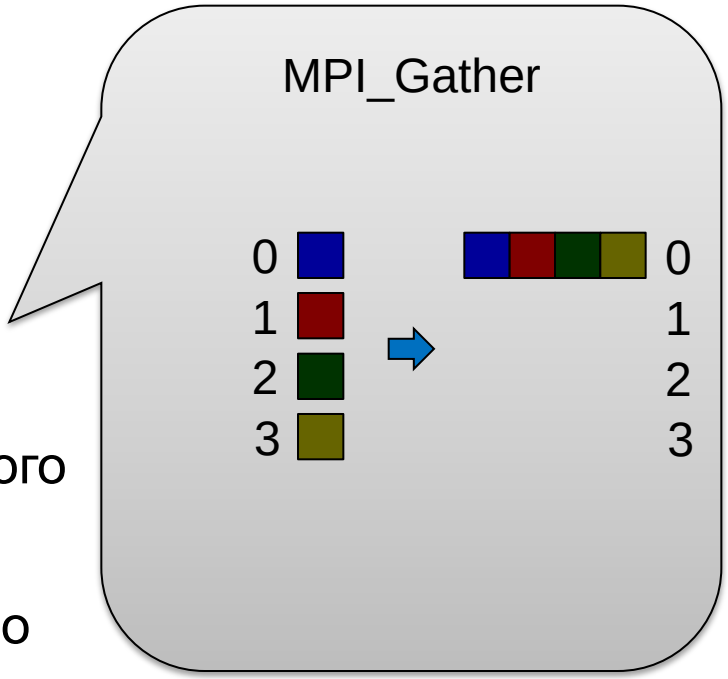
- **MPI_Barrier** – барьерная синхронизация
- **MPI_Bcast** – рассылка данных от одного процесса всем остальным
- **MPI_Gather** – сборка распределенных по всем процессам данных в одном процессе
- **MPI_Scatter** – распределение данных одного процесса между всеми процессами
- **MPI_Allgather** – сборка распределенных по всем процессам данных во всех процессах
- **MPI_Alltoall** – обмен данными «все со всеми»



Коллективные операции

- Выполняются всеми процессами группы
- Базовые операции:

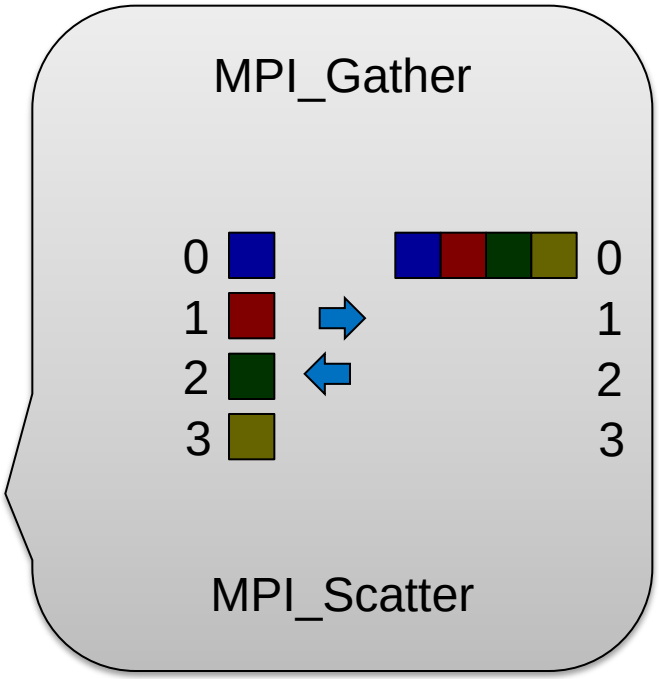
- **MPI_Barrier** – барьерная синхронизация
- **MPI_Bcast** – рассылка данных от одного процесса всем остальным
- **MPI_Gather** – сборка распределенных по всем процессам данных в одном процессе
- **MPI_Scatter** – распределение данных одного процесса между всеми процессами
- **MPI_Allgather** – сборка распределенных по всем процессам данных во всех процессах
- **MPI_Alltoall** – обмен данными «все со всеми»



Коллективные операции

- Выполняются всеми процессами группы
- Базовые операции:

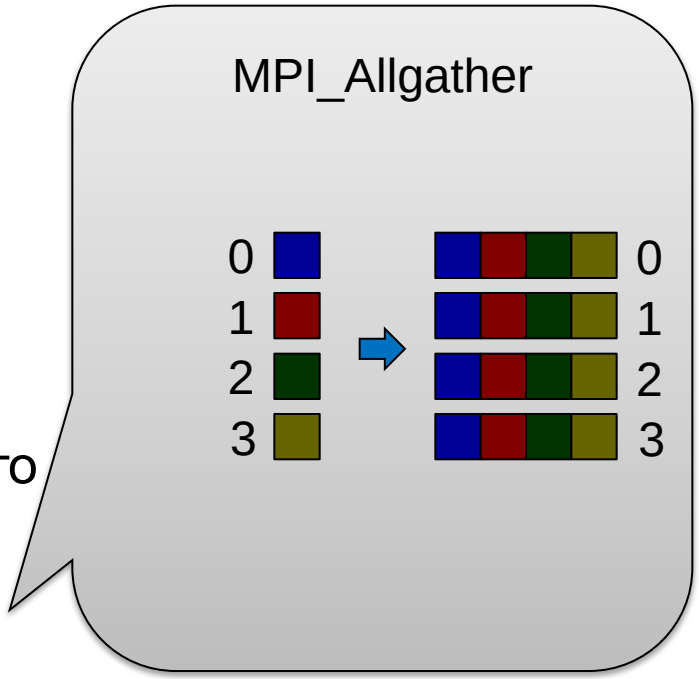
- **MPI_Barrier** – барьерная синхронизация
- **MPI_Bcast** – рассылка данных от одного процесса всем остальным
- **MPI_Gather** – сборка распределенных по всем процессам данных в одном процессе
- **MPI_Scatter** – распределение данных одного процесса между всеми процессами
- **MPI_Allgather** – сборка распределенных по всем процессам данных во всех процессах
- **MPI_Alltoall** – обмен данными «все со всеми»



Коллективные операции

- Выполняются всеми процессами группы
- Базовые операции:

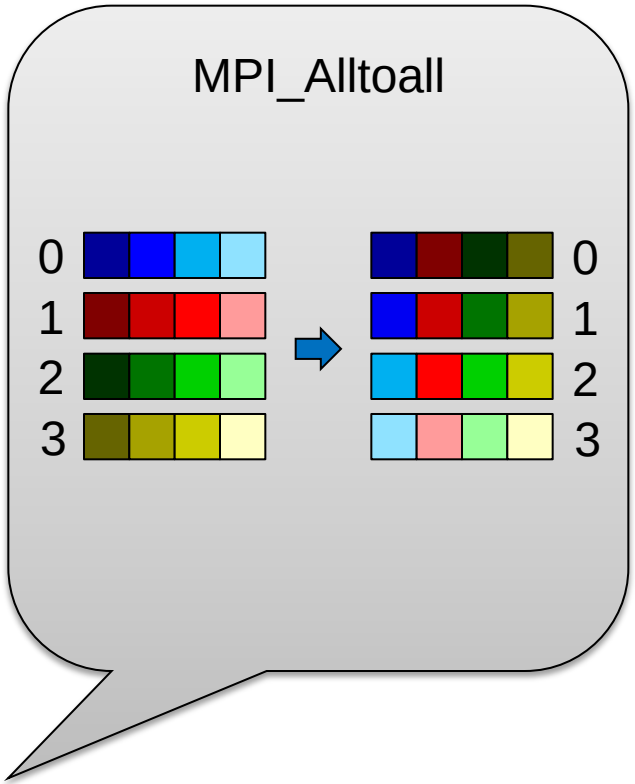
- **MPI_Barrier** – барьерная синхронизация
- **MPI_Bcast** – рассылка данных от одного процесса всем остальным
- **MPI_Gather** – сборка распределенных по всем процессам данных в одном процессе
- **MPI_Scatter** – распределение данных одного процесса между всеми процессами
- **MPI_Allgather** – сборка распределенных по всем процессам данных во всех процессах
- **MPI_Alltoall** – обмен данными «все со всеми»



Коллективные операции

- Выполняются всеми процессами группы
- Базовые операции:

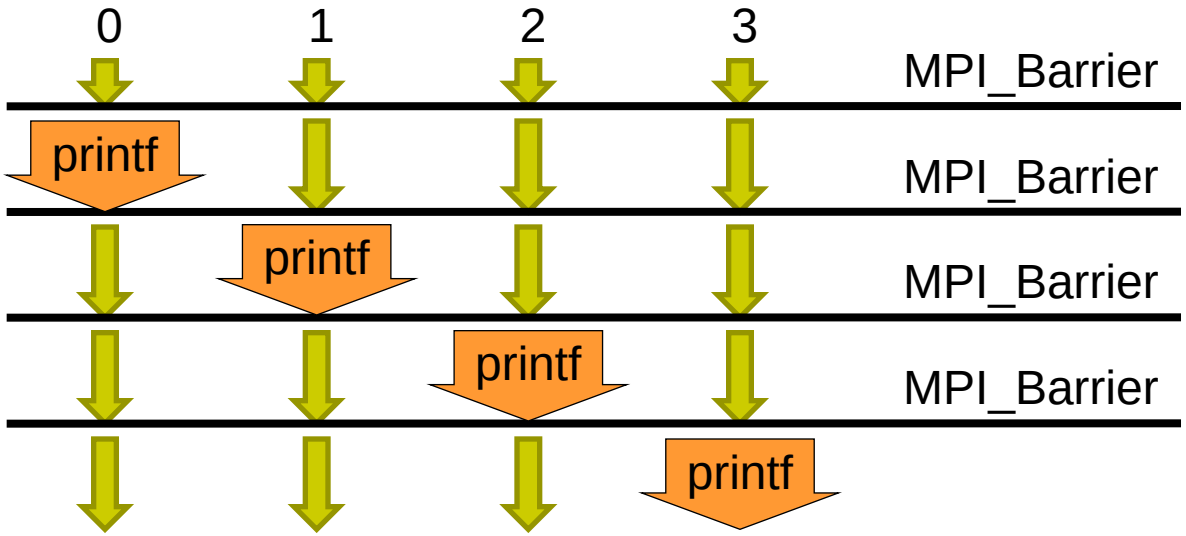
- **MPI_Barrier** – барьерная синхронизация
- **MPI_Bcast** – рассылка данных от одного процесса всем остальным
- **MPI_Gather** – сборка распределенных по всем процессам данных в одном процессе
- **MPI_Scatter** – распределение данных одного процесса между всеми процессами
- **MPI_Allgather** – сборка распределенных по всем процессам данных во всех процессах
- **MPI_Alltoall** – обмен данными «все со всеми»



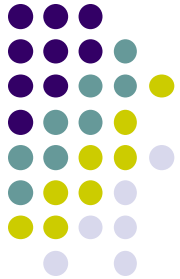
Коллективные операции

- Пример: упорядоченный вывод

```
int size,rank;
MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
for (i=0;i<size;i++)
{ MPI_Barrier(MPI_COMM_WORLD);
  if (rank==i) printf("%d\n", rank);
}
```



Коллективные операции

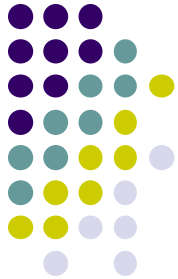


- Редукционные операции
  - MPI_Reduce – редукция по всем процессам с сохранением результата в одном процессе

- Пример: вычисление скалярного произведения

```
double x[N],y[N], s=0.0, sum;
for (i=0;i<N;i++) s += x[i]*y[i];
MPI_Reduce(&s,&sum,1,MPI_DOUBLE,
            MPI_SUM,0,comm);
if (rank==0) printf("Result: %lf\n",sum);
```

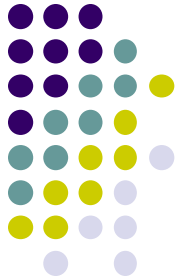
Коллективные операции



- Редукционные операции
  - **MPI_Reduce** – редукция по всем процессам с сохранением результата в одном процессе
  - **MPI_Allreduce** – редукция по всем процессам с сохранением результата во всех процессах
  - **MPI_Reduce_scatter** = MPI_Reduce + MPI_Scatter
  - **MPI_Reduce_scatter_block** – то же, для блоков равного размера
  - **MPI_Scan** – префиксная инклюзивная редукционная операция по всем процессам с сохранением частичных результатов во всех процессах
  - **MPI_Exscan** – префиксная эксклюзивная редукционная операция по всем процессам с сохранением частичных результатов во всех процессах
  - **MPI_Reduce_local** – локальная редукционная операция в одном процессе

- Операции редукции
  - **Встроенные:** MPI_MAX, MPI_MIN, MPI_SUM, MPI_PROD, MPI_LAND, MPI_LOR, MPI_LXOR, MPI_BAND, MPI_BOR, MPI_BXOR, MPI_MAXLOC, MPI_MINLOC
  - **Пользовательские**

Коллективные операции



- Асинхронные операции

Базовые

- `MPI_Ibarrier`
- `MPI_Ibcast`
- `MPI_Igather`
- `MPI_Iscatter`
- `MPI_Iallgather`
- `MPI_Ialltoall`

Редукционные

- `MPI_Ireduce`
- `MPI_Iallreduce`
- `MPI_Ireduce_scatter`
- `MPI_Ireduce_scatter_block`
- `MPI_Iscan`
- `MPI_Iexscan`

Коллективные операции

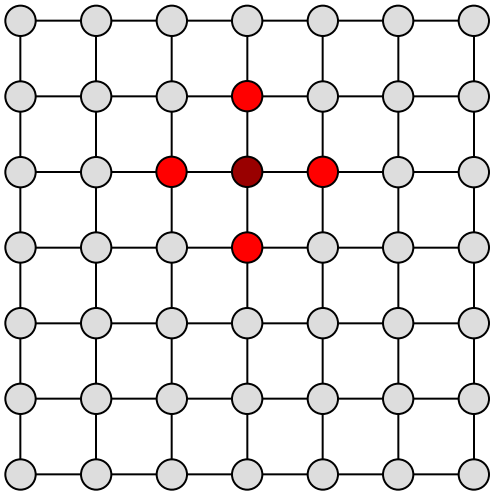
- Коллективные операции с соседними процессами

Блокирующие

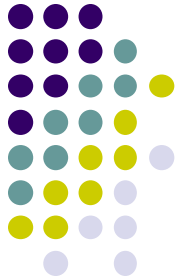
- **`MPI_Neighbor_gather`**
- **`MPI_Neighbor_alltoall`**

Асинхронные (неблокирующие)

- **`MPI_Ineighbor_gather`**
- **`MPI_Ineighbor_alltoall`**



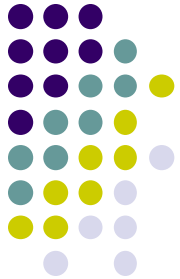
Односторонние коммуникации



- Участвует в передаче только один процесс (источник / приёмник)

- Порядок работы:
  - Каждый процесс заводит «окно» в своей памяти, которое видно другим процессам
  - Все процессы асинхронно читают и пишут данные в своих и чужих окнах (put/get)
  - Операции синхронизации гарантируют завершение передач данных

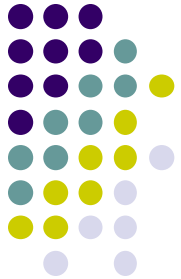
Односторонние коммуникации



- Пример: создание окна

```
MPI_Win win;
int *x;
int elemsize = sizeof(int), winsize = N*elemsize;
MPI_Alloc_mem(winsize, MPI_INFO_NULL, &x);
MPI_Win_create(x, winsize, elemsize,
               MPI_INFO_NULL, comm, &win);

// обмен данными

MPI_Win_free(&win);
MPI_Free_mem(x);
```

Односторонние коммуникации

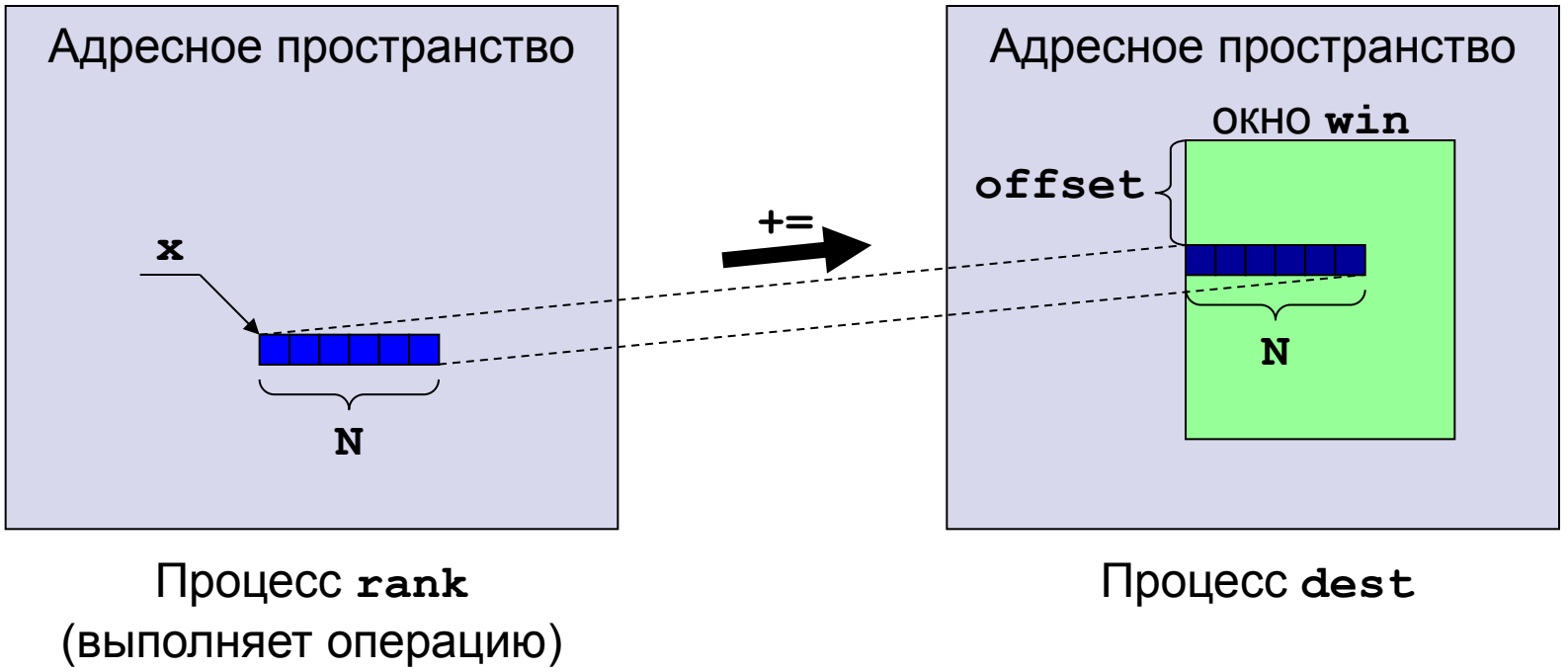


- Операции передачи данных:
  - MPI_Put – запись данных в удалённую память
  - MPI_Get – чтение данных из удалённой памяти
  - MPI_Accumulate () – выполнение заданной операции и запись в удалённую память
- Все операции асинхронные
- Завершение операций передачи проверяется командами синхронизации

Односторонние коммуникации

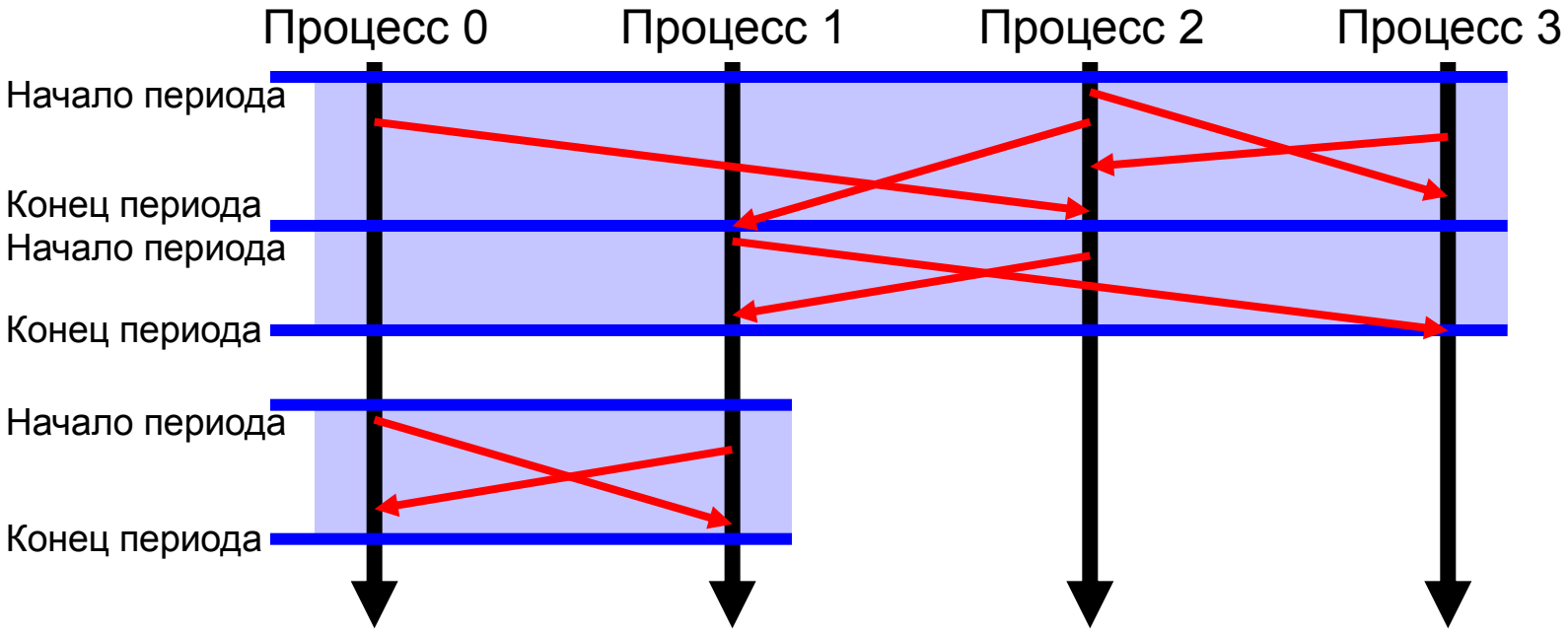
- Пример:

```
MPI_Accumulate(x,N,MPI_INT,
               dest,offset,N,MPI_INT,
               MPI_SUM,comm,win);
```

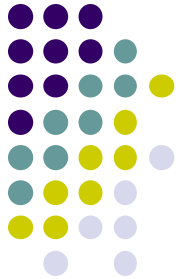


Односторонние коммуникации

- Синхронизация передач данных
  - Время работы программы разделено на периоды, в которые происходят асинхронные передачи.
  - В конце каждого периода происходит ожидание всех запущенных в нём команд передачи данных.
  - Начало и конец периода определяется специальными командами синхронизации.



Односторонние коммуникации



- Синхронизация передач данных
  - Время работы программы разделено на периоды, в которые происходят асинхронные передачи.
  - В конце каждого периода происходит ожидание всех запущенных в нём команд передачи данных.
  - Начало и конец периода определяется специальными командами синхронизации.

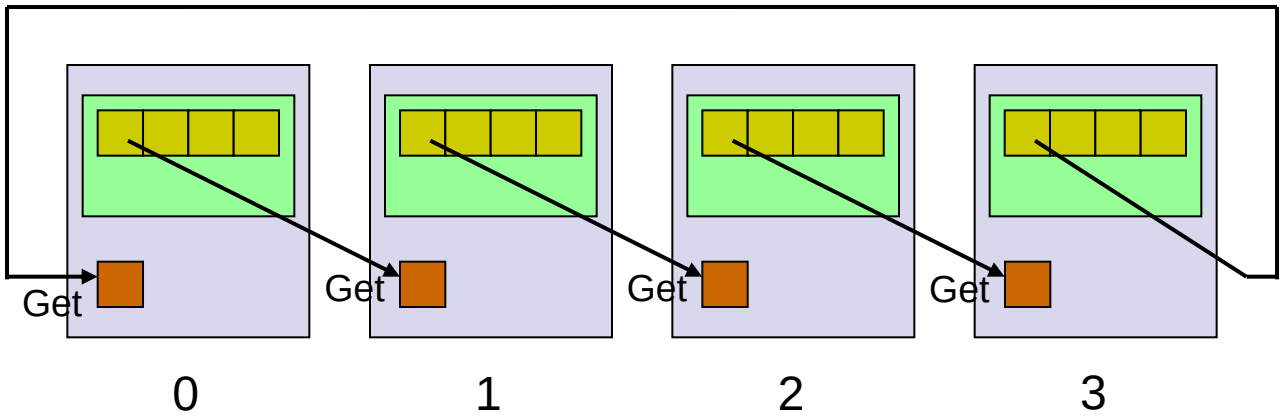
- Два типа периодов обменов
  - Период глобальных обменов – в обменах участвуют все процессы
  - Период локальных обменов – процесс сам выбирает, с кем он обменивается

- Команды синхронизации обменов:
  - **`MPI_Win_fence`** – граница периода глобальных обменов
  - **`MPI_Win_start`** / **`MPI_Win_post`** / **`MPI_Win_lock`** – начало периода локальных обменов
  - **`MPI_Win_complete`** / **`MPI_Win_wait`** / **`MPI_Win_unlock`** – конец периода локальных обменов

Односторонние коммуникации

- Период глобальных обменов. Пример:

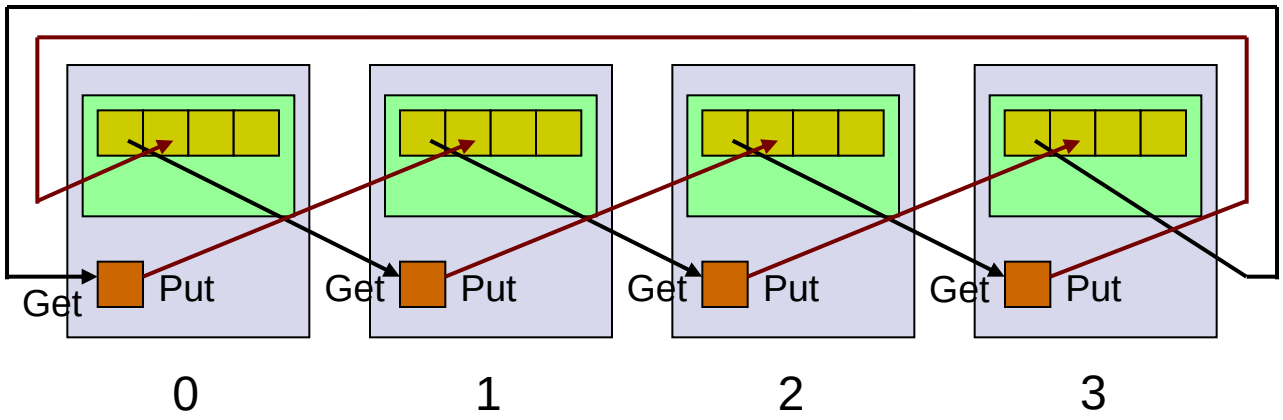
```
MPI_Win win;
int data;
MPI_Win_fence(MPI_MODE_NOPRECEDE | MPI_MODE_NOPUT, win);
MPI_Get(&data,1,MPI_INT,
        (rank+size-1)%size, 0 ,1 ,MPI_INT, win);
MPI_Win_fence(MPI_MODE_NOSTORE, win);
MPI_Put(&data,1,MPI_INT,
        (rank+1)%size, 1, 1, MPI_INT, win);
MPI_Win_fence(MPI_MODE_NOSTORE | MPI_MODE_NOSUCCEED, win);
```



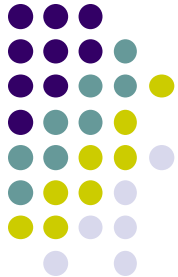
Односторонние коммуникации

- Период глобальных обменов. Пример:

```
MPI_Win win;
int data;
MPI_Win_fence(MPI_MODE_NOPRECEDE | MPI_MODE_NOPUT, win);
MPI_Get(&data,1,MPI_INT,
        (rank+size-1)%size, 0 ,1 ,MPI_INT, win);
MPI_Win_fence(MPI_MODE_NOSTORE, win);
MPI_Put(&data,1,MPI_INT,
        (rank+1)%size, 1, 1, MPI_INT, win);
MPI_Win_fence(MPI_MODE_NOSTORE | MPI_MODE_NOSUCCEED, win);
```



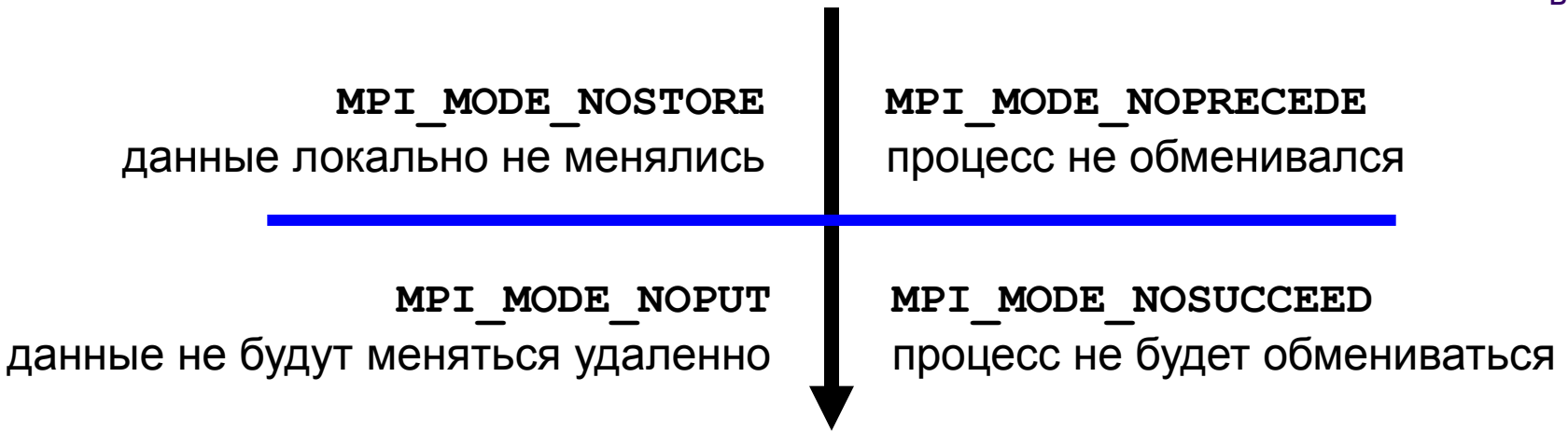
Односторонние коммуникации



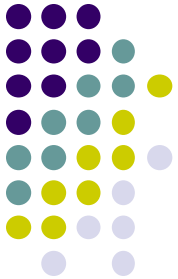
- Задание режима синхронизации (параметр MPI_Info в операции синхронизации), может её ускорить
  Однако значение 0 всегда работает правильно.

- Значения MPI_Info:
  - **MPI_MODE_NOSTORE** – данные в локальном окне локально не изменялись,
  - **MPI_MODE_NOPUT** – данные в локальном окне не будут изменяться удаленными операциями Put / Accumulate
  - **MPI_MODE_NOPRECEDE** – не было локальных Get / Put / Accumulate
  - **MPI_MODE_NOSUCCEED** – не будет локальных Get / Put / Accumulate

  } только все процессы вместе



Односторонние коммуникации



- Период локальных обменов групп процессов
  - Процесс явно указывает группу процессов, которые будут обращаться в локальное окно.
  - Процесс явно указывает группу процессов, в окно к которым он сам будет обращаться.

Начало периода, намерение обращаться в окна группе grp_to

```
MPI_Win_start(grp_to,0,win);
```

Операции доступа

```
MPI_Put/Get/Accumulate(…,win);
```

Конец периода

```
MPI_Win_complete(win);
```

Начало периода, предоставление локального окна группе grp_from

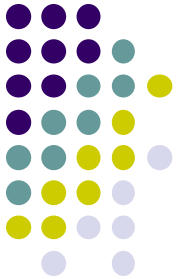
```
MPI_Win_post(grp_from,0,win);
```

Здесь в наше окно происходят обращения

Конец периода

```
MPI_Win_wait(win);
```

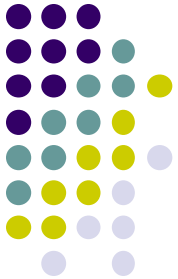
Односторонние коммуникации



- Период локальных обменов групп процессов. Пример:

```
MPI_Group grp_all,grp_from,grp_to; MPI_Win win;
int x,rank,prev,next;
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_group(MPI_COMM_WORLD, &grp_all);
prev = (rank+size-1)%size;
next = (rank+1)%size;
MPI_Group_incl(grp_all, 1, &prev, &grp_from); // создаём группу
MPI_Group_incl(grp_all, 1, &next, &grp_to);   // создаём группу
MPI_Win_create(&x, sizeof(int), sizeof(int),
               MPI_INFO_NULL, MPI_COMM_WORLD, &win);
MPI_Win_post(grp_from, 0, win); // открываем доступ к себе
MPI_Win_start(grp_to, 0, win);  // заявляем, куда будем обращаться
MPI_Put(&rank,1,MPI_INT, next,0,1,MPI_INT, win); // отправляем
MPI_Win_complete(win);          // дожидаемся завершения отправки
MPI_Win_wait(win);              // дожидаемся завершения приёма
printf("%d : %d \n",rank,x);
MPI_Win_free(&win);
```

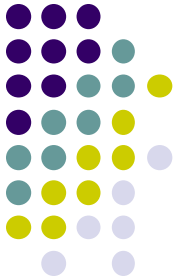
Односторонние коммуникации



- «Защищённый» доступ в окно другого процесса
- Пример:

```
int *x, y, esize = sizeof(int);
MPI_Alloc_mem(2*esize,MPI_INFO_NULL,&x);
MPI_Win_create(x,2*esize,esize,MPI_INFO_NULL,
                  MPI_COMM_WORLD,&win);
x[0] = rank; x[1] = rank;

MPI_Win_lock(MPI_LOCK_SHARED,prev, 0, win);
MPI_Get(&y,1,MPI_INT, prev,0,1, MPI_INT,win);
MPI_Win_unlock(prev, win);
MPI_Win_lock(MPI_LOCK_EXCLUSIVE,next,0,win)
MPI_Put(&y,1,MPI_INT, next,1,1, MPI_INT,win);
MPI_Win_unlock(next, win);

printf("%d : %d\n", rank, x[1]);
```

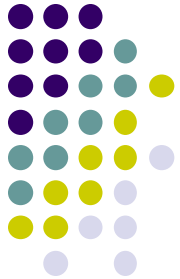
Односторонние коммуникации



- «Защищённый» доступ в окно другого процесса
- Пример:

```
int *x, y, esize = sizeof(int);
MPI_Alloc_mem(2*esize,MPI_INFO_NULL,&x);
MPI_Win_create(x,2*esize,esize,MPI_INFO_NULL,
                  MPI_COMM_WORLD,&win);
x[0] = rank; x[1] = rank;
MPI_Barrier(MPI_COMM_WORLD);
MPI_Win_lock(MPI_LOCK_SHARED,prev, 0, win);
MPI_Get(&y,1,MPI_INT, prev,0,1, MPI_INT,win);
MPI_Win_unlock(prev, win);

MPI_Win_lock(MPI_LOCK_EXCLUSIVE,next,0,win)
MPI_Put(&y,1,MPI_INT, next,1,1, MPI_INT,win);
MPI_Win_unlock(next, win);
MPI_Barrier(MPI_COMM_WORLD);
printf("%d : %d\n", rank, x[1]);
```

Односторонние коммуникации (MPI 3.0)



Создание «окна» в памяти

- **`MPI_Win_create`**
- **`MPI_Win_allocate`**

Удаление «окна»

- **`MPI_Win_free`**

Чтение удаленных данных

- **`MPI_Get`**
- **`MPI_Rget`**

Запись удаленных данных

- **`MPI_Put`**
- **`MPI_Rput`**

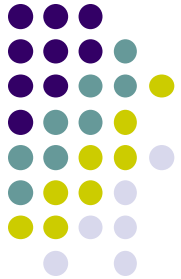
Чтение с редукцией

- **`MPI_Get_accumulate`**
- **`MPI_Rget_accumulate`**

Запись с редукцией

- **`MPI_Accumulate`**
- **`MPI_Raccumulate`**

Односторонние коммуникации (MPI 3.0)



- `MPI_Win_fence`
- `MPI_Win_start, MPI_Win_complete, MPI_Win_post, MPI_Win_wait`
- `MPI_Win_lock, MPI_Win_lock_all, MPI_Win_unlock, MPI_Win_unlock_all`
- `MPI_Win_flush, MPI_Win_flush_all, MPI_Win_flush_local, MPI_Win_flush_local_all, MPI_Win_sync`

MPI-типы данных

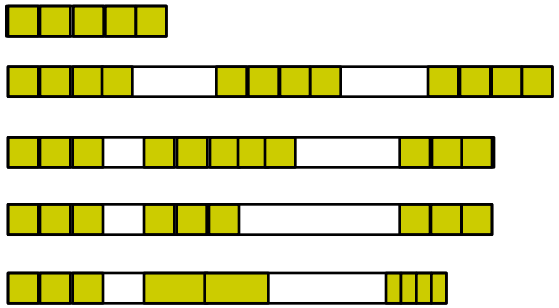
- Обеспечивают правильное преобразование данных при пересылке с одной архитектуры на другую
  - Стандартные MPI-типы

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

  - Пользовательские MPI-типы
    - Конструируются пользователем из базовых типов

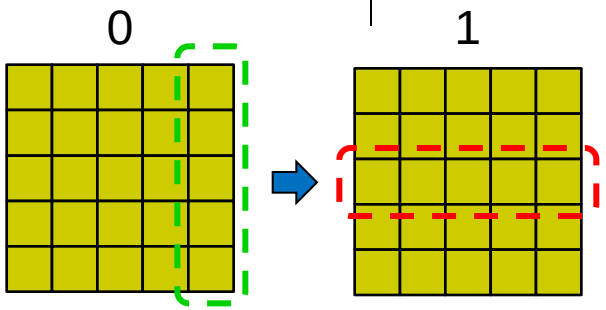
MPI-типы данных

- Пользовательские MPI-типы позволяют пересылать сложные структуры данных как один элемент
- Создание MPI-типа:
  - **`MPI_Type_contiguous`**
  - **`MPI_Type_vector`**
  - **`MPI_Type_indexed`**
  - **`MPI_Type_create_indexed_block`**
  - **`MPI_Type_create_struct`**
  - **`MPI_Type_create_subarray`**
  - **`MPI_Type_create_darray`**
- Завершение создания MPI-типа:
  - **`MPI_Type_commit`**
- Удаление MPI-типа:
  - **`MPI_Type_free`**



MPI типы данных

- Пример: строка и столбец матрицы

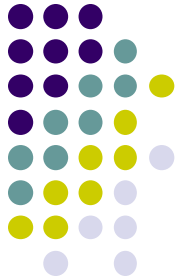



```
double x[N][N];
MPI_Datatype row_type, col_type;

MPI_Type_contiguous(N,MPI_DOUBLE,&row_type);
MPI_Type_vector(N,1,N,MPI_DOUBLE,&col_type);
MPI_Type_commit(&row_type);
MPI_Type_commit(&col_type);

if (rank==0)
  MPI_Send(&x[0][N-1],1,col_type, 1,123, comm);
if (rank==1)
  MPI_Recv(&x[N/2][0],1,row_type, 0,123, comm,
           MPI_STATUS_IGNORE);
```

MPI-IO: Ввод/вывод

- MPI-IO используется для одновременного чтения/записи файлов многими процессами
- Базовые операции:
  - MPI_File_open
  - MPI_File_read
  - MPI_File_write
  - MPI_File_close
- Для задания подмножества записей файла, относящегося к данному процессу, используются MPI-типы данных.

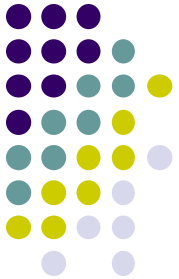
Измерение времени в MPI



- **`double MPI_Wtime();`** – время в секундах по отношению к некоторому моменту в прошлом

- **`double MPI_Wtick();`** – разрешающая способность MPI_Wtime() в секундах

Литература по MPI



- http://www.mpi-forum.org/
- man pages
- Google: “MPI tutorial” – 1-я ссылка
- …