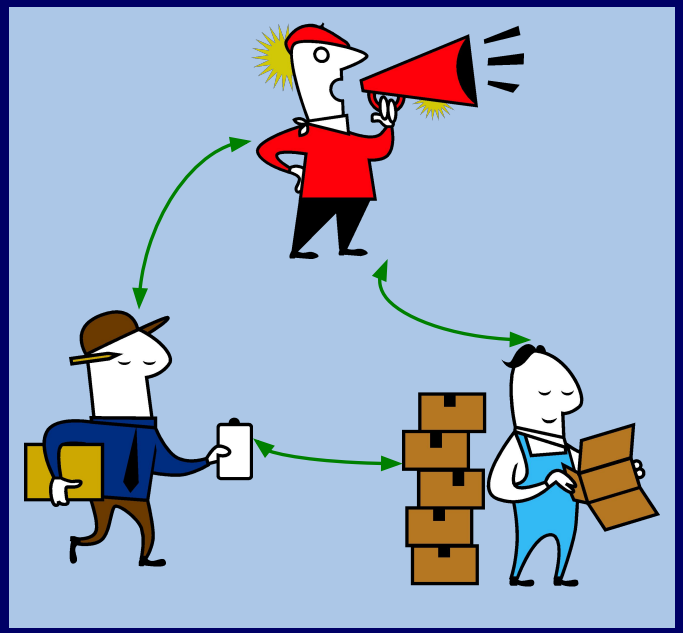
Параллельное программирование в стандарте MPI

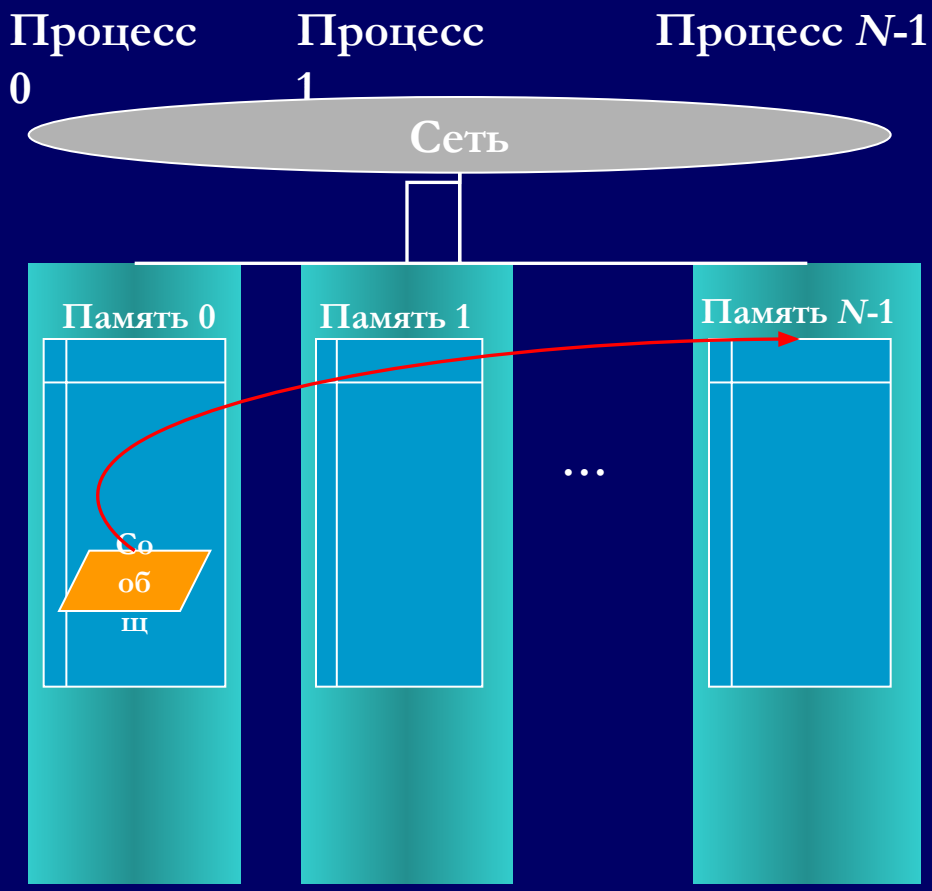


М.Л. Цымблер

Содержание

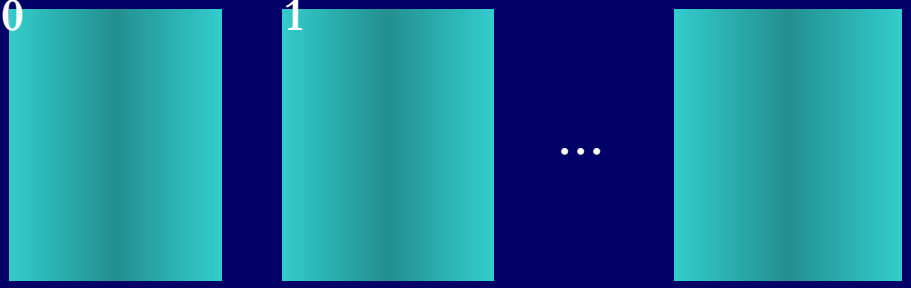
- Модель передачи сообщений
- Модели SPMD и MPMD
- Стандарт MPI
- Основные понятия и функции MPI

Модель передачи сообщений



- *Параллельное приложение* состоит из нескольких *процессов*, выполняющихся одновременно.
- Каждый процесс имеет приватную память.
- Обмены данными между процессами осуществляются посредством явной отправки/получения *сообщений*.
- Процессы могут выполняться как на одном и том же, так и на разных процессорах.

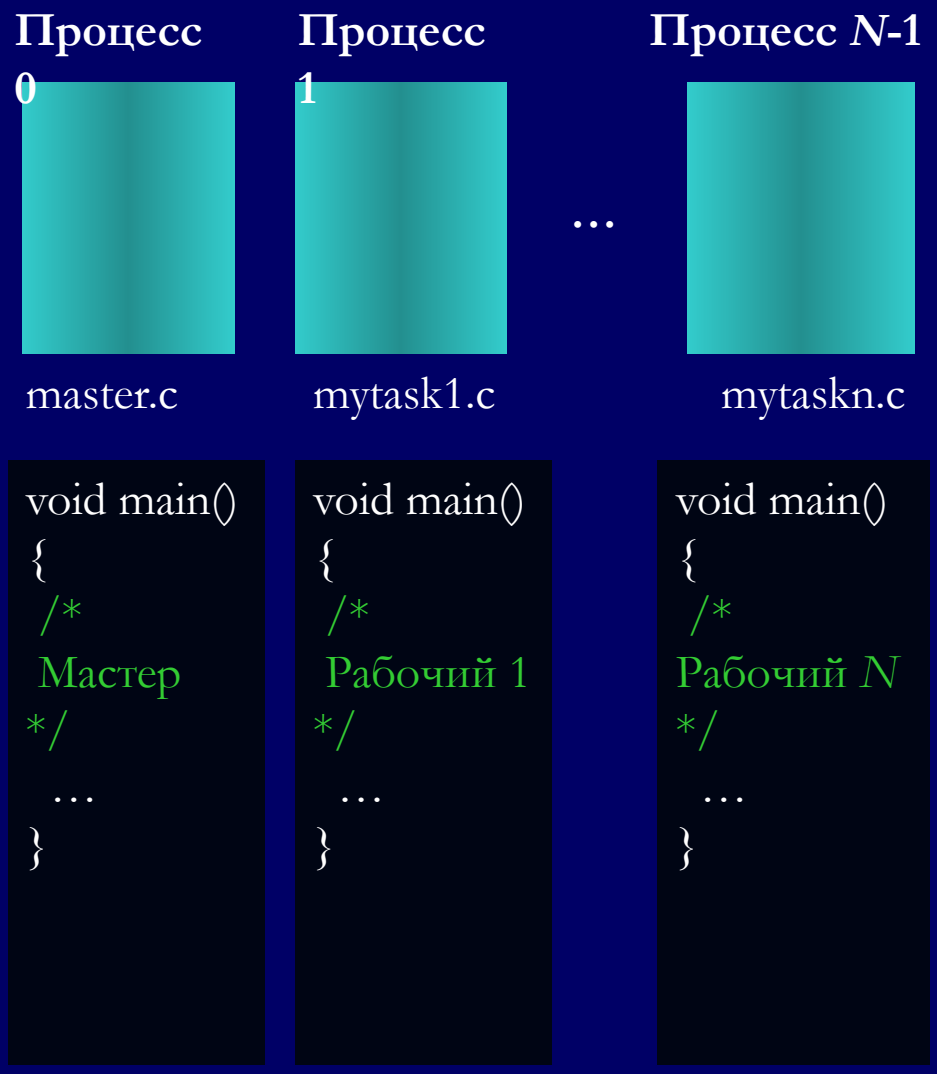
Модель выполнения SPMD




```
void main()
{
…
  switch (myrank) {
  case 0: do_mastertask(); /* Мастер */
  case 1: do_task1(); /* Рабочий 1 */
  case 2: do_task2();  /* Рабочий 2 */
  …
    }
…
}
```

- *Модель SPMD (Single Program Multiple Data)* предполагает, каждый процесс параллельного приложения имеет один и тот же исходный код.

Модель выполнения MPMD



- *Модель MPMD (Multiple Program Multiple Data)* предполагает, каждый процессы параллельного приложения имеют различные исходные коды.

Стандарт MPI

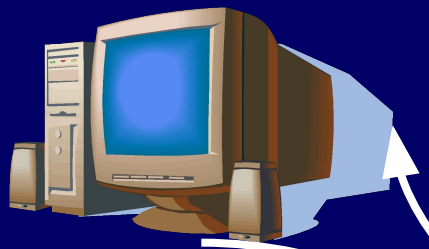
- *MPI (Message Passing Interface)* – стандарт, реализующий модель обмена сообщениями между параллельными процессами. Поддерживает модели выполнения SPMD и (начиная с версии 2.0) MPMD.
- Стандарт представляет собой набор спецификаций подпрограмм (более 120) на языках C, C++ и FORTRAN.
- Стандарт реализуется разработчиками в виде библиотек подпрограмм для различных аппаратно-программных платформ (кластеры, персональные компьютеры, …, Windows, Unix/Linux, …).

Источники информации

- Сервер PARALLEL.RU
  http://parallel.ru/tech/tech_dev/mpi.html
- MPI-форум
  http://www.mpi-forum.org

Цикл разработки MPI-программ

- Персональный компьютер
  - Первоначальная разработка и отладка
  - Отладка по результатам запуска на суперкомпьютере



- Суперкомпьютер
  - Тестирование

MPI-программа

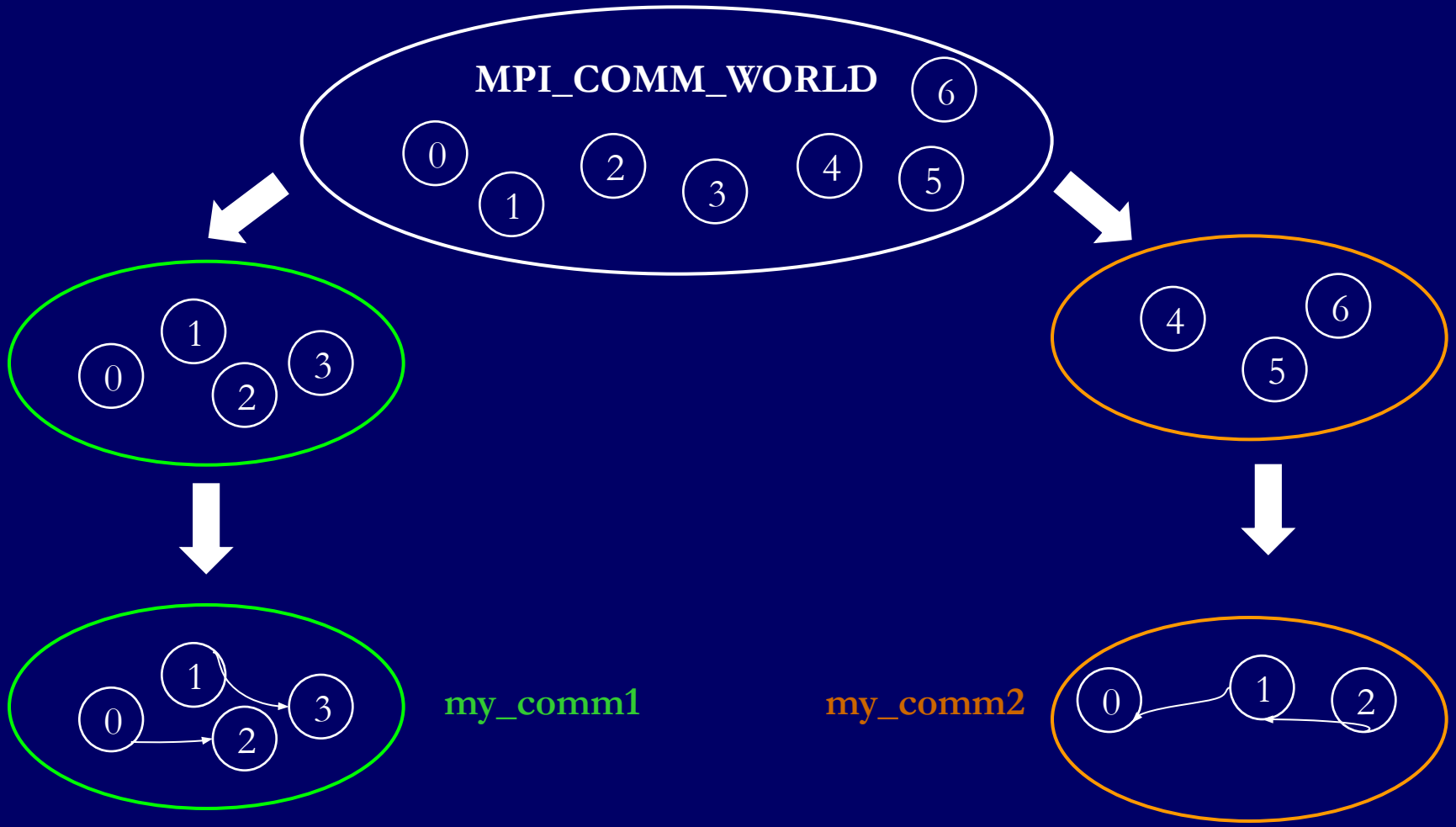
- *MPI-программа* – множество параллельных взаимодействующих процессов.
- Процессы порождаются один раз, во время запуска программы*.
- Каждый процесс работает в своем адресном пространстве, каких-либо общих данных нет. Единственный способ взаимодействия процессов – явный обмен сообщениями.

* Порождение дополнительных процессов и уничтожение существующих возможно только начиная с версии MPI-2.0.

Коммуникаторы

- Для локализации области взаимодействия процессов можно создавать специальные программные объекты – *коммуникаторы*. Процесс может входить в разные коммуникаторы.
- Взаимодействия процессов проходят в рамках некоторого коммуникатора. Сообщения, переданные в разных коммуникаторах, не пересекаются и не мешают друг другу.
- Атрибуты процесса MPI-программы:
  - номер коммуникатора;
  - номер в коммуникаторе (от 0 до n-1, n – число процессов в коммуникаторе).
- Стандартные коммуникаторы:
  - MPI_COMM_WORLD – все процессы приложения
  - MPI_COMM_SELF – текущий процесс приложения
  - MPI_COMM_NULL – пустой коммуникатор

Коммуникаторы



Сообщение

- *Сообщение* процесса – набор данных стандартного (определенного в MPI) или пользовательского типа.
- Основные атрибуты сообщения:
  - номер процесса-отправителя (получателя)
  - номер коммуникатора
  - тег (уникальный идентификатор) сообщения (целое число)
  - тип элементов данных в сообщении
  - количество элементов данных
  - указатель на буфер с сообщением

Структура MPI-программы

```
#include "mpi.h"          /* Подключение библиотеки */

void main(int argc, char * argv[])
{

    MPI_Init(&argc, &argv);   /* Инициализация */

    …                 /* Обмены  */

    MPI_Finalize();       /* Завершение */
}
```

MPI-функции

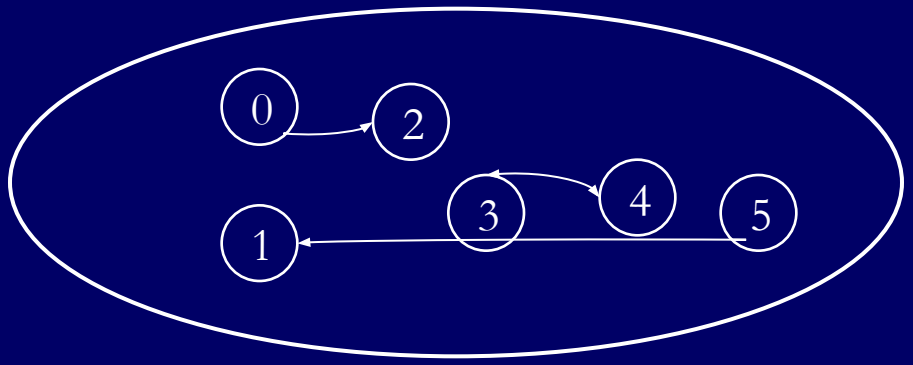
- Имеют имена вида MPI_...
- Возвращают целое число – MPI_SUCCESS или код ошибки.
- Простые функции общего назначения:
  - /* Количество процессов в коммуникаторе */
    int MPI_Comm_size(MPI_Comm comm, int * size);
  - /* Номер (ранг) текущего процесса в коммуникаторе */
    int MPI_Comm_rank(MPI_Comm comm, int * rank);

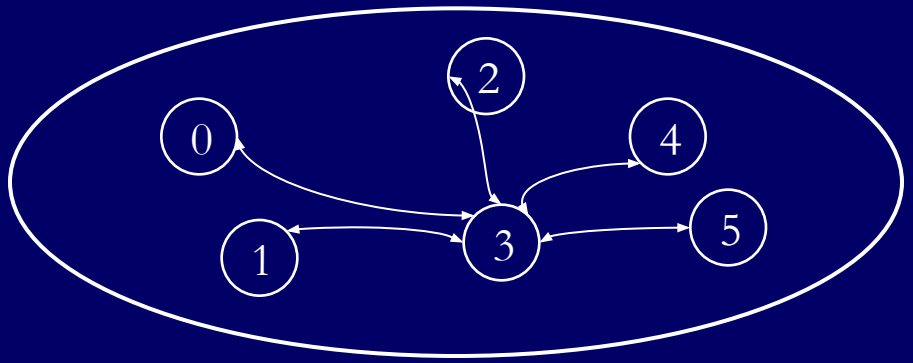
Пример MPI-программы

```
#include "mpi.h"
#include <stdio.h>
int total, iam;
void main(int argc, char * argv[])
{
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &total);
    MPI_Comm_rank(MPI_COMM_WORLD, &iam);
    printf("Привет! Я %d-й процесс из %d.\n", iam, total);
    MPI_Finalize();
}
```

Виды взаимодействия процессов

- *Взаимодействие "точка-точка"* – обмен между двумя процессами одного коммуникатора.
- *Коллективное взаимодействие* – обмен между всеми процессами одного коммуникатора.





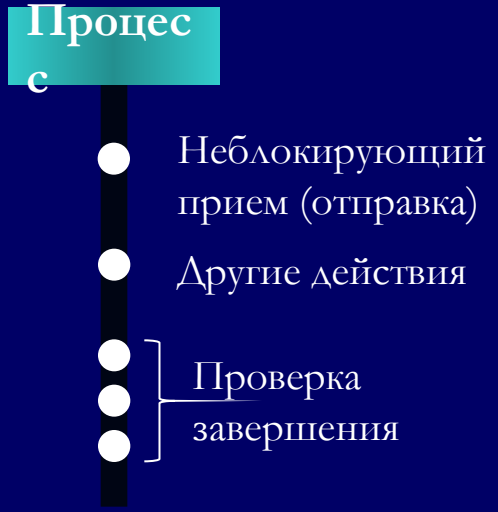
Взаимодействие "точка-точка"

- Участвуют два процесса: отправитель сообщения и получатель сообщения.
  - Отправитель должен вызвать одну из функций отправки сообщения и явно указать атрибуты получателя (коммуникатор и номер в коммуникаторе) и тег сообщения.
  - Получатель должен вызвать одну из функций получения сообщения и указать (тот же) коммуникатор отправителя; получатель может не знать номер отправителя и тег сообщения.
- Свойства:
  - Сохранение порядка (если P0 передает P1 сообщения A и затем B, то P1 получит A, а затем B).
  - Гарантированное выполнение обмена (если P0 вызвал функцию отправки, а P1 вызвал функцию получения, то P1 получит сообщение от P0).

Виды коммуникационных функций "точка-точка"

- *Блокирующая* функция запускает операцию и возвращает управление процессу только после ее завершения.
  - После завершения допустима модификация отправленного (принятого) сообщения.

- *Неблокирующая* функция запускает операцию и возвращает управление процессу немедленно.
  - Факт завершения операции проверяется позднее с помощью другой функции.
  - До завершения операции недопустима модификация отправляемого (получаемого) сообщения.



Отправка сообщений при использовании функций "точка-точка"

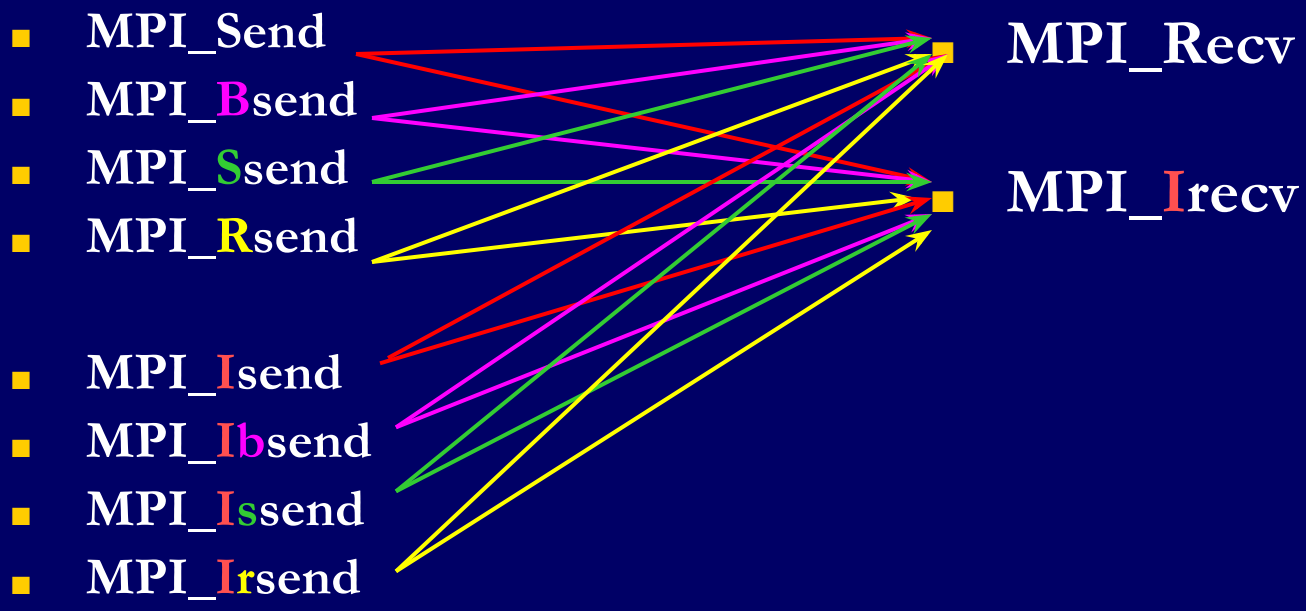
- *Стандартная* – завершается сразу после отправки сообщения.
- *Синхронная* – завершается после приема подтверждения от адресата.
- *Буферизованная* – завершается, как только сообщение копируется в системный буфер для дальнейшей отправки.
- *"По готовности"* – начинается, если адресат инициализировал прием и завершается сразу после отправки.

Коммуникационные функции "точка-точка"

- Отправка: MPI_[I][R, S, B]Send
- Прием: MPI_[I]Recv

Блокирующие			Неблокирующие		
Отправка		*Прием*	*Отправка*		*Прием*
Стандартная	MPI_Send	MPI_Recv	*Стандартная*	MPI_Isend	MPI_Irecv
Синхронная	MPI_Ssend		*Синхронная*	MPI_Issend	
Буферизованная	MPI_Bsend		*Буферизованная*	MPI_Ibsend	
По готовности	MPI_Rsend		*По готовности*	MPI_Irsend	

Коммуникационные функции "точка-точка"



Блокирующая стандартная отправка сообщения

- int MPI_Send
  - IN void * buf – указатель на буфер с сообщением
  - IN int count – количество элементов в буфере
  - IN MPI_Datatype datatype – MPI-тип данных элементов в буфере
  - IN int dest – номер процесса-получателя
  - IN int tag – тег сообщения
  - IN MPI_Comm comm – коммуникатор

Блокирующее стандартное получение сообщения

- int MPI_Recv
  - OUT void * buf – указатель на буфер с сообщением
  - IN int count – количество элементов в буфере
  - IN MPI_Datatype datatype – MPI-тип данных элементов в буфере
  - IN int src – номер процесса-отправителя
  - IN int tag – тег сообщения
  - IN MPI_Comm comm – коммуникатор
  - OUT MPI_Status* status – информация о фактически полученных данных (указатель на структуру с двумя полями: source – номер процесса-источника, tag – тег сообщения)

Неблокирующая стандартная отправка сообщения

- int MPI_Isend
  - IN void * buf – указатель на буфер с сообщением
  - IN int count – количество элементов в буфере
  - IN MPI_Datatype datatype – MPI-тип данных элементов в буфере
  - IN int dest – номер процесса-получателя
  - IN int tag – тег сообщения
  - IN MPI_Comm comm – коммуникатор
  - OUT MPI_Request *request – дескриптор операции (для последующей проверки завершения операции)

Неблокирующее стандартное получение сообщения

- int MPI_Irecv
  - OUT void * buf – указатель на буфер с сообщением
  - IN int count – количество элементов в буфере
  - IN MPI_Datatype datatype – MPI-тип данных элементов в буфере
  - IN int src – номер процесса-отправителя
  - IN int tag – тег сообщения
  - IN MPI_Comm comm – коммуникатор
  - OUT MPI_Request *request – дескриптор операции (для последующей проверки завершения операции)

Завершение неблокирующих обменов

- /* Проверка завершения */
  int MPI_Test
  (MPI_Request *request, int *flag, MPI_Status *status)
  - int MPI_Testany (…)
  - int MPI_Testall (…)
  - int MPI_Testsome (…)
- /* Ожидание завершения */
  int MPI_Wait
  (MPI_Request *request, MPI_Status *status)
  - int MPI_Waitany (…)
  - int MPI_Waitall (…)
  - int MPI_Waitsome (…)

Тупики (deadlocks)

- Гарантированный тупик

P0	P1
MPI_Recv (от процесса P1); MPI_Send (процессу P1);	MPI_Recv (от процесса P0); MPI_Send (процессу P0);

- Возможный тупик

P0	P1
MPI_Send (процессу P1); MPI_Recv (от процесса P1);	MPI_Send (процессу P0); MPI_Recv (от процесса P0);

Разрешение тупиков

P0

```
MPI_Send (процессу P1);
MPI_Recv (от процесса P1);
```

P1

```
MPI_Recv (от процесса P0);
MPI_Send (процессу P0);
```

P0

```
MPI_Send (процессу P1);
MPI_Recv (от процесса P1);
```

P1

```
MPI_Irecv (от процесса P0);
MPI_Send (процессу P0);
MPI_Wait
```

P0

```
/* Совмещенные прием и передача */
MPI_Sendrecv
```

P1

```
/* Совмещенные прием и передача */
MPI_Sendrecv
```

Барьерная синхронизация процессов

- int MPI_Barrier(MPI_Comm comm)
- Вызвавший данную функцию процесс блокируется до тех пор, пока все другие процессы указанного коммуникатора не вызовут эту функцию.
  Завершение данной функции возможно только всеми процессами одновременно (все процессы "преодолевают барьер" одновременно).

Получение сообщений

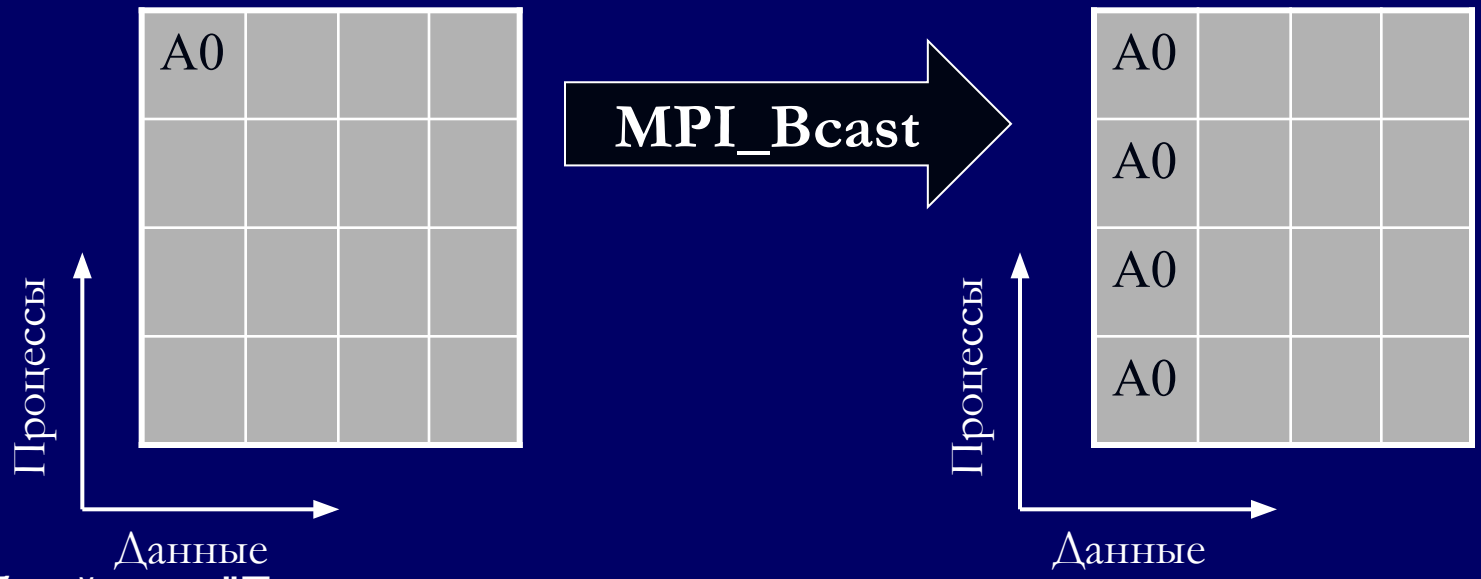
- "Джокеры"
  - MPI_ANY_SOURCE – получить сообщение, отправленное любым процессом
  - MPI_ANY_TAG – получить сообщение, имеющее любой тег
- Информация об ожидаемом сообщении
  - /* С блокировкой */
    int MPI_Probe(int src, int tag, MPI_Comm comm, MPI_Status * status);
  - /* Без блокировки */
    int MPI_Iprobe
    (int src, int tag, MPI_Comm comm, int * flag, MPI_Status * status);
  - /* Количество элементов в принятом сообщении */
    int MPI_Get_count(MPI_Status * status, MPI_Datatype type, int * cnt);

Коллективные операции

- Прием и/или передачу выполняют одновременно *все* процессы коммуникатора.
- Коллективная функция имеет большое количество параметров, часть которых нужна для приема, а часть для передачи. При вызове в разных процессах та или иная часть игнорируется.
- Значения *всех* параметров коллективных функций (за исключением адресов буферов) должны быть идентичными во всех процессах.
- MPI назначает теги для сообщений автоматически.

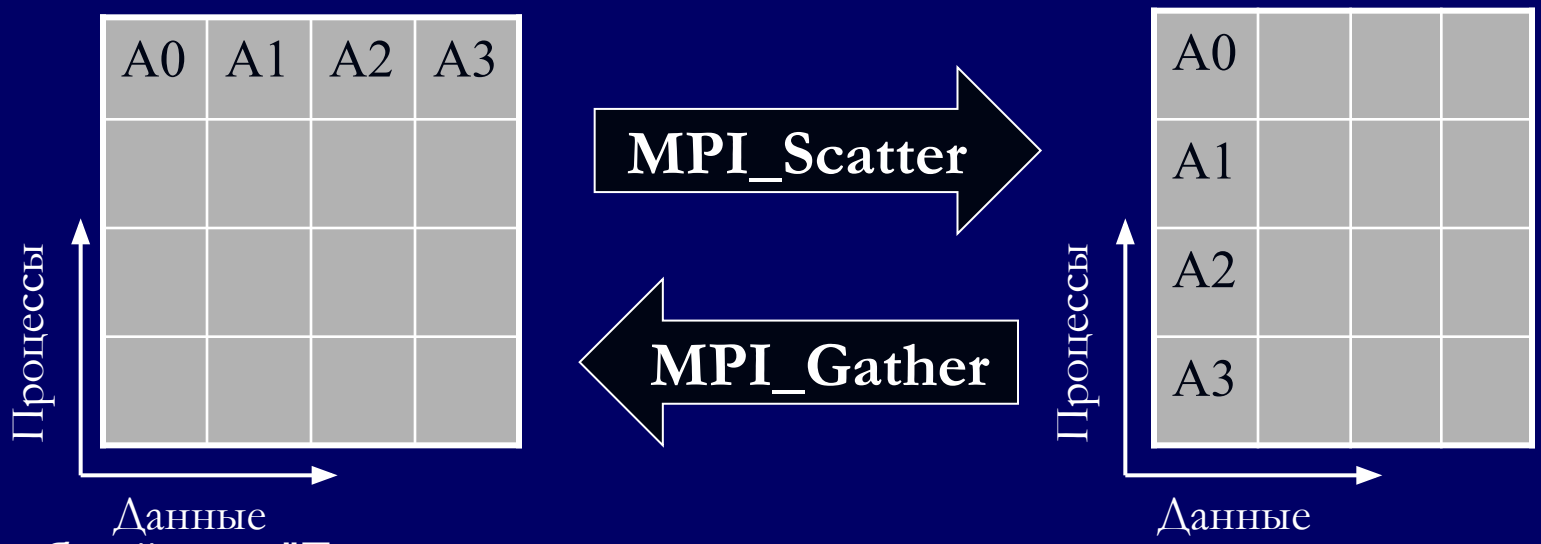
Коллективная рассылка сообщения

- /* Рассылка содержимого буфера из процесса с номером rootRank, во все остальные */
  int MPI_Bcast
  (void * buf, int count, MPI_Datatype type, int rootRank,
  MPI_Comm comm);

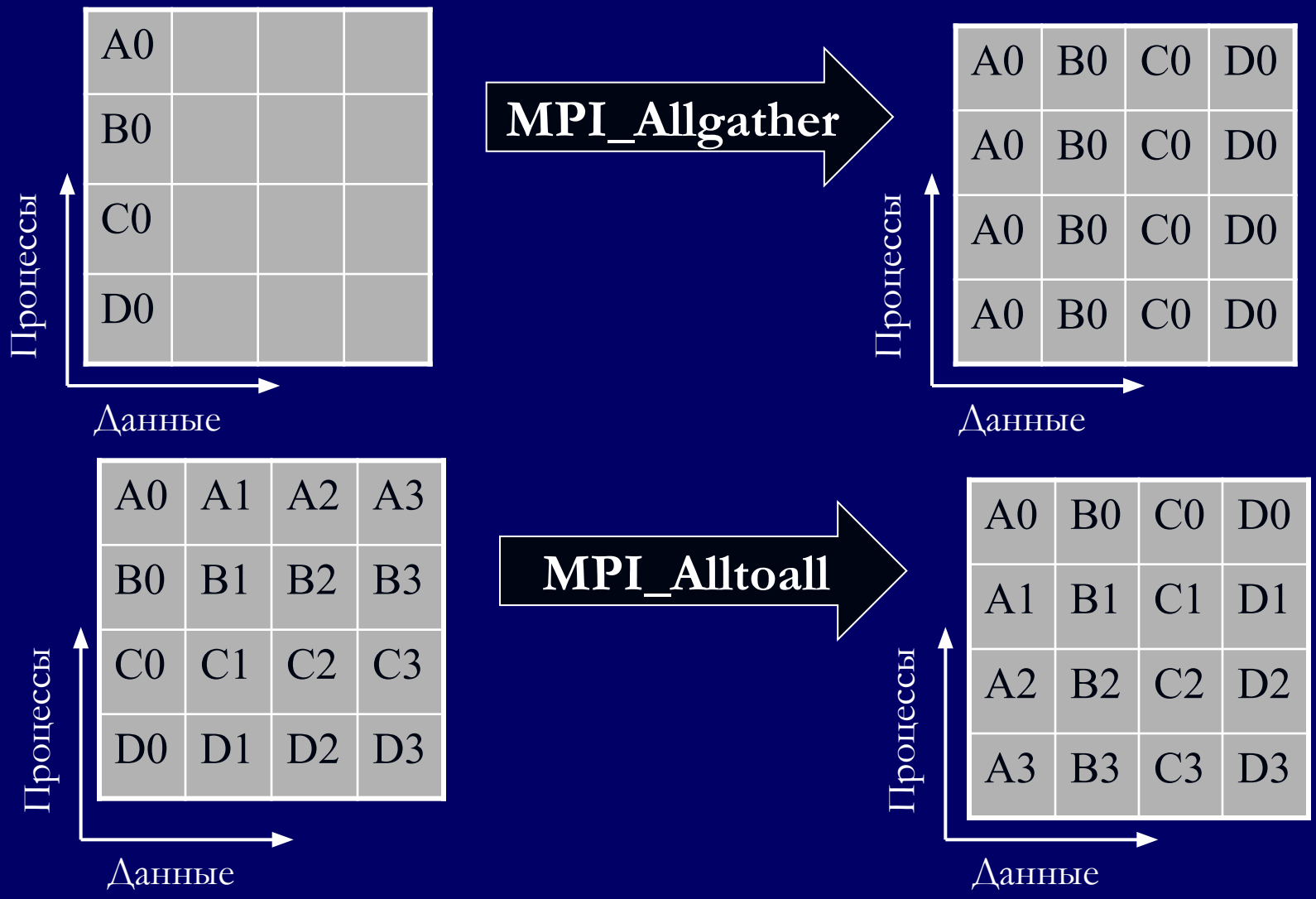


Коллективный прием сообщения

- /* Сборка элементов данных из буферов всех процессов в буфере процесса с номером rootRank */
  int MPI_Gather
  (void * sbuf, int scount, MPI_Datatype stype, void * rbuf,
  int rcount, MPI_Datatype rtype, int rootRank, MPI_Comm comm);
- /* Рассылка элементов данных из буфера процесса с номером rootRank в буфера всех процессов (обратная к MPI_Gather) */
  int MPI_Scatter



Широковещательные прием и передача



Глобальные операции над данными

- /* Выполнение count независимых глобальных операций op над соответствующими элементами массивов sbuf.
  Результат выполнения над *i*-ми элементами sbuf записывается в *i*-й элемент массива rbuf процесса rootRank. */
  int MPI_Reduce
  (void * sbuf, void * rbuf, int count, MPI_Datatype type, MPI_Op op,
  int rootRank, MPI_Comm comm);
- Глобальные операции:
  - MPI_MAX, MPI_MIN
  - MPI_SUM, MPI_PROD
  - …
  - MPI_Op_Create()

Стандартные типы данных в MPI

Тип MPI	Соответствующий тип C
MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_BYTE	-
MPI_PACKED	-

Пользовательские типы данных

- /* Создание типа "массив" */
  int MPI_Type_contiguous(int count, MPI_Datatype oldtype, MPI_Datatype *newtype);

```
#define N 100
int A[N];
MPI_Datatype MPI_INTARRAY100;
…
MPI_Type_contiguous(N, MPI_INT, & MPI_INTARRAY100);
MPI_Type_commit(&MPI_INTARRAY100);
…
MPI_Send(A, 1, MPI_INTARRAY100, ... );
/* то же, что и MPI_Send(A, N, MPI_INT, ... ); */
…
MPI_Type_free( &intArray100 );
```