Казанский федеральный
УНИВЕРСИТЕТ
ВЫСШАЯ ШКОЛА
информационных технологий
и информационных систем

Введение в технологию MPI

Эдуард Храмченков

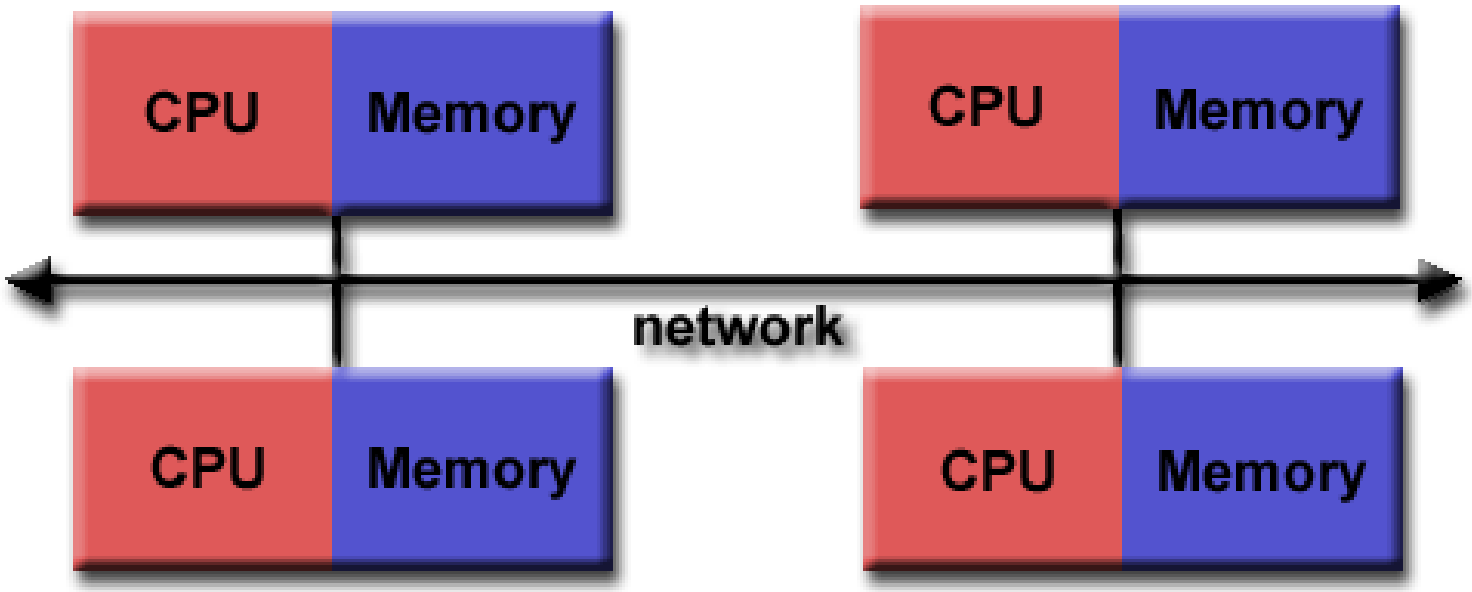
MPI

- Message Passing Interface – стандарт передачи сообщений между узлами параллельного компьютера
- Наиболее распространённый стандарт параллельного программирования
- Поддерживается практически всеми параллельными системами
- Предназначен, в первую очередь, для систем с распределенной памятью

MPI

- Открытый стандарт с несколькими свободными и коммерческими реализациями
- Наиболее распространённые свободные реализации MPI – OpenMPI и MPICH
- Коммерческие реализации от HP, Intel
- MS-MPI – разработка Microsoft для Windows HPC Cluster
- Поддержка основных семейств ОС
- Поддержка языков: Fortran, C/C++, Java*(MPJ)

Модель MPI



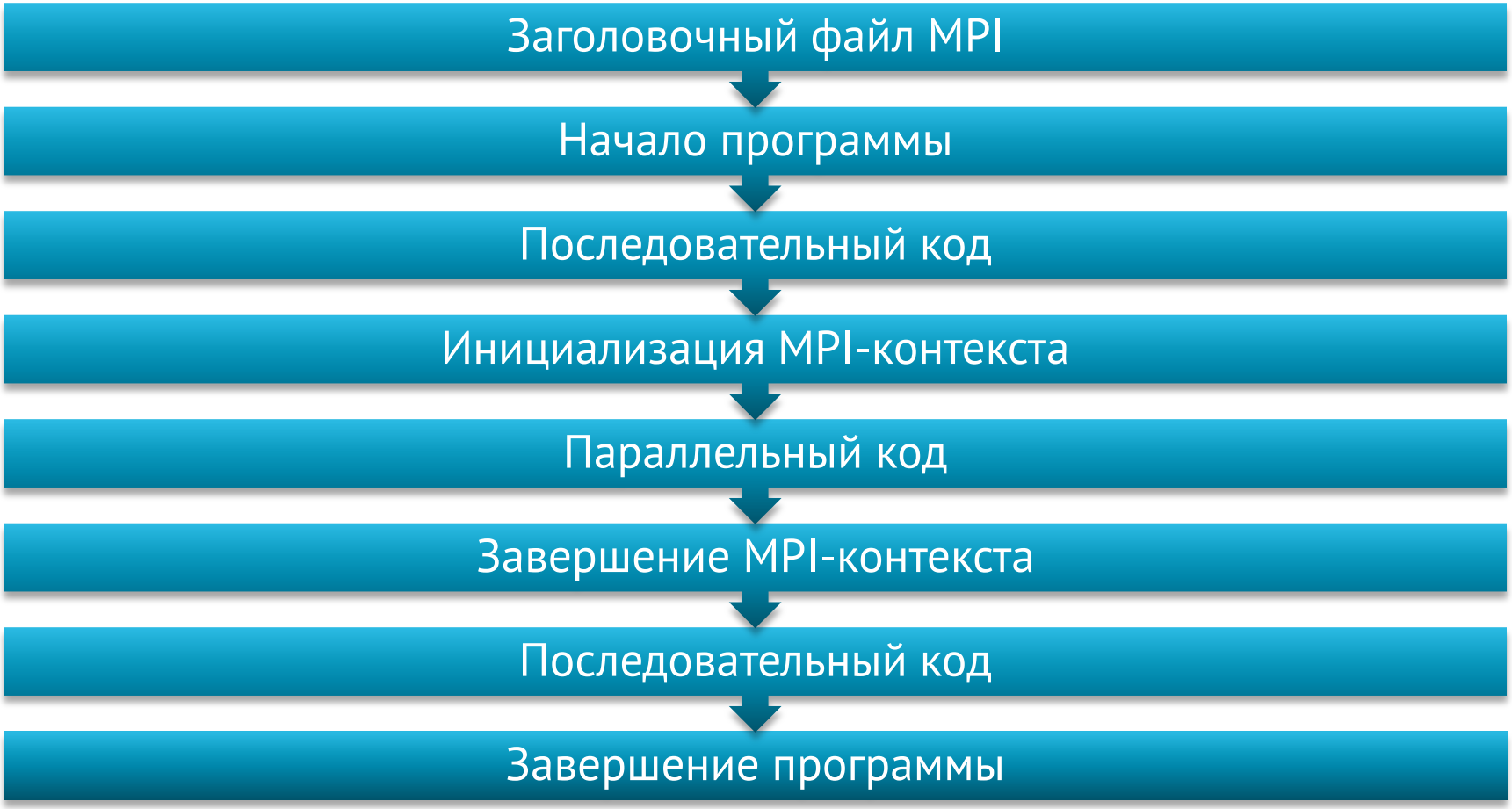
Модель MPI

- Среда выполнения MPI-кода представляется как набор вычислительных узлов
- Узлы связанны между собой сетью
- MPI-узлами могут быть узлы кластера, процессоры многопроцессорной системы, ядра одного процессора, GPU/MIC, etc.
- Сеть связывающая узлы может иметь различную физическую основу – Ethernet, InfiniBand, процессорная шина, etc.

Модель MPI

- MPI-программа – набор процессов, каждый из которых выполняет один и тот же код
- Процессы распределяются по вычислительным узлам и работают только с локальной памятью узла
- Каждый процесс имеет свой номер и может работать над частью общей задачи
- Процессы обмениваются своими локальными данными через сеть

Структура MPI-приложения



Компиляция и запуск кода

- Компиляция кода осуществляется при помощи MPI-обертки компилятора для конкретного языка

$mpicc/mpic++ [опции] file.c/file.cpp

- Запуск MPI-программы осуществляется с помощью команды

$mpirun [-n/np <число процессов>] file

Пример 1

- Функция main может содержать стандартные аргументы для их передачи в функцию MPI_Init() (зависит от реализации MPI)
- Запускается n экземпляров программы, каждый из которых выводит сообщение
- Нумерация процессов начинается с 0
- Почти все MPI-функции в качестве возвращаемого значения отдают код успеха/ошибки (MPI_SUCCESS/MPI_ERR_*)

Основные функции MPI

```
int MPI_Init (int *argc, char ***argv)
```

- Первая MPI-функция вызываемая в программе
- Инициализирует MPI-контекст, до вызова это функции работа с MPI невозможна
- После инициализации MPI-контекста, каждый созданный процесс включается в коммуникатор MPI_COMM_WORLD

Основные функции MPI

```
int MPI_Comm_size ( MPI_Comm comm, int *size )
```

- Возвращает в переменную size количество потоков в коммуникаторе comm

```
int MPI_Comm_rank ( MPI_Comm comm, int *rank )
```

- Возвращает в переменную rank порядковый номер потока в коммуникаторе comm

Основные функции MPI

```
int MPI_Finalize()
```

- Последняя MPI-функция вызываемая в программе
- Завершает MPI-контекст, освобождает ресурсы и уничтожает коммуникатор MPI_COMM_WORLD
- Количество потоков после вызова функции неопределено

Коммуникаторы MPI

- Специальные объекты, объединяющие процессы в группы
- В рамках коммуникатора происходит обмен сообщениями и данными, редукция данных
- MPI_COMM_WORLD – коммуникатор по умолчанию, он всегда создается при инициализации MPI-контекста
- Почти все MPI-функции принимают коммуникатор как аргумент

Коммуникаторы MPI

- В сложных программах с большим количеством потоков иногда выгодно создать несколько меньших коммуникаторов
- Новые коммуникаторы создаются на базе существующего
- При этом происходит распределение процессов по новым коммуникаторам
- Изначальный коммуникатор при этом не уничтожается

Двухточечная коммуникация

- Базовый вид коммуникации – двухточечная (point-to-point, от процесса к процессу)
- Процесс А помещает данные для отправки процессу В в специальный буфер
- После этого буфер с данными передается по коммуникационной сети
- Процесс В должен подтвердить, что принимает это сообщение
- Процесс А получает уведомление о доставке

Двухточечная коммуникация

- До тех пор, пока процесс А не получит подтверждения о доставке сообщения, он не может выполнять дальнейшие операции
- Такая передача называется блокирующей (blocking)
- Блокирующая передача может быть синхронной и асинхронной
- После возврата функции отправки, буфер доступен для изменения

Двухточечная коммуникация

```
MPI_Send(
        void* data,// //Адрес первого передаваемого элемента
        int count, //Количество передаваемых элементов
        MPI_Datatype datatype, //Тип передаваемых элементов
        int destination, //Номер целевого процесса
        int tag, //Метка сообщения
        MPI_Comm communicator) //Коммуникатор
```

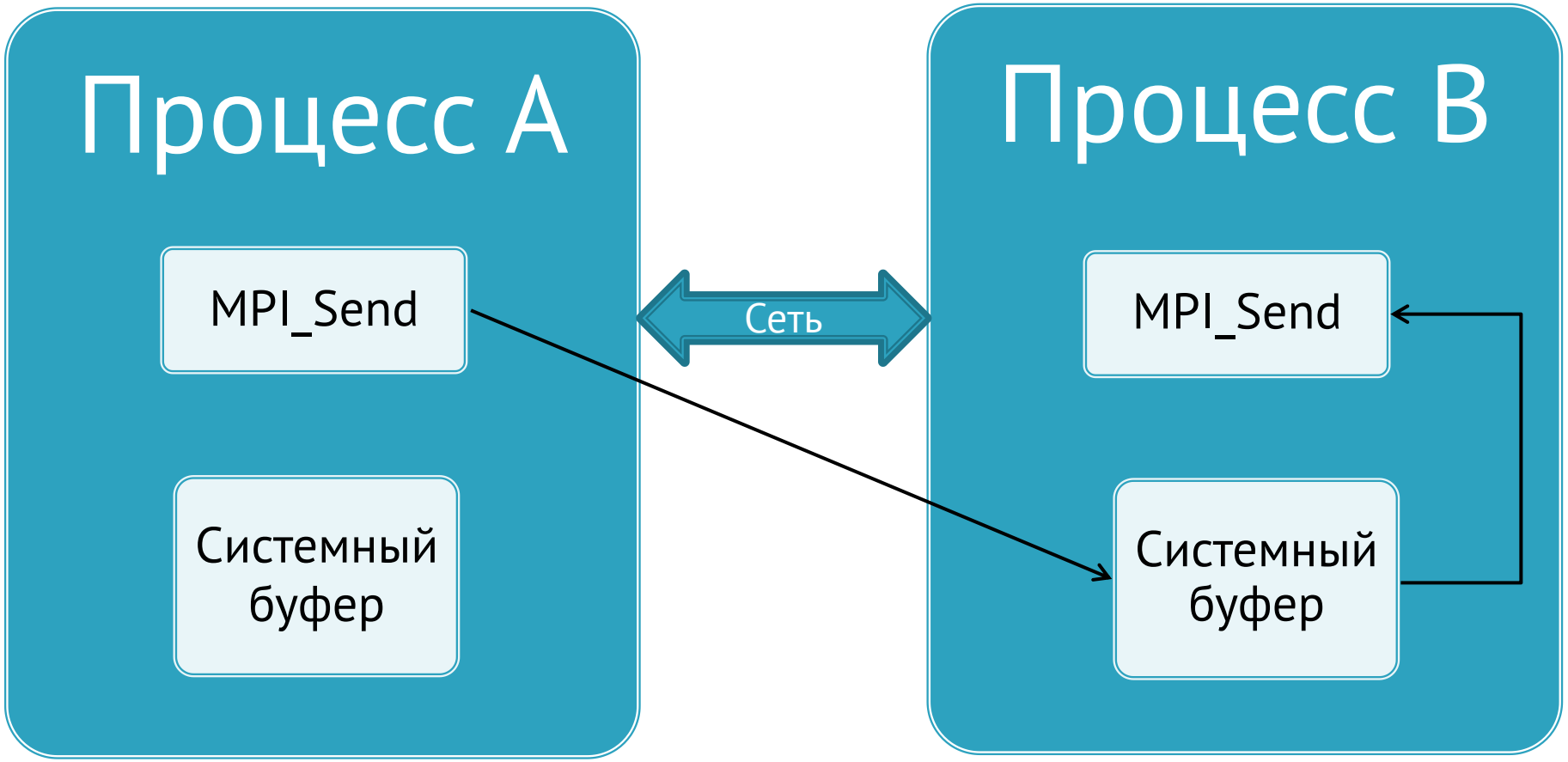
Двухточечная коммуникация

```
MPI_Recv(
        void* data,// //Адрес принимающего буфера
        int count, //Количество передаваемых элементов
        MPI_Datatype datatype, //Тип передаваемых элементов
        int source, //Номер отправляющего процесса
        int tag, //Метка сообщения
        MPI_Comm communicator, //Коммуникатор
        MPI_Status* status ) //Статус передачи
```

Двухточечная коммуникация

- В случае, если процесс A посылает процессу B несколько сообщений они помечаются метками, с помощью параметра tag
- Отправленные данные помещаются в системный буфер на целевом узле
- Как только процесс B готов принять эти данные, они передаются из буфера процессу
- Это позволяет передавать и получать сообщения без синхронизации процессов

Двухточечная коммуникация



Двухточечная коммуникация

- Системный буфер зависит от имплементации MPI стандарта, не прописан в самом стандарте
- Управляется библиотекой MPI и недоступен программисту
- Это конечный ресурс, который может быть быстро исчерпан
- Позволяет повысить производительность MPI-приложения за счет асинхронности

Некоторые типы данных MPI

MPI_CHAR	//signed char
MPI_UNSIGNED_CHAR	//unsigned char
MPI_BYTE	//char
MPI_INT	//int
MPI_UNSIGNED	//unsigned int
MPI_LONG	//signed long int
MPI_SHORT	//signed short int
MPI_UNSIGNED_LONG	//unsigned long int
MPI_UNSIGNED_SHORT	//unsigned short int
MPI_FLOAT	//float
MPI_DOUBLE	//double
MPI_LONG_DOUBLE	//long double

Пример 2

- Передача числа между процессами

Двухточечная коммуникация

- Кроме описанных базовых блокирующих асинхронных функций приема/передачи есть функции
  - Блокирующего синхронного приема/передачи
  - Блокирующего буферизованного приема/передачи
  - Неблокирующего синхронного приема/передачи
  - Неблокирующего асинхронного приема/передачи
  - Неблокирующего буферизованного приема/передачи

Коллективная коммуникация

- Возможна коллективная широковещательная коммуникация
- Коммуникация должна охватывать все процессы коммуникатора
- В случае, если не все процессы участвуют в коллективной коммуникации, поведение программы становится неопределенным(UB)
- Программист должен удостовериться, что все процессы участвуют в коммуникации

Коллективная коммуникация

- Существуют несколько типов коллективной коммуникации
  - Синхронизация – процессы ждут, пока все участники коммуникатора не достигнут контрольной точки
  - Передача данных – широковещательная рассылка, распределение/сбор данных
  - Коллективные вычисления – один участник собирает данные от остальных и производит вычисления (сумма, произведение, etc.)

Синхронизация в MPI

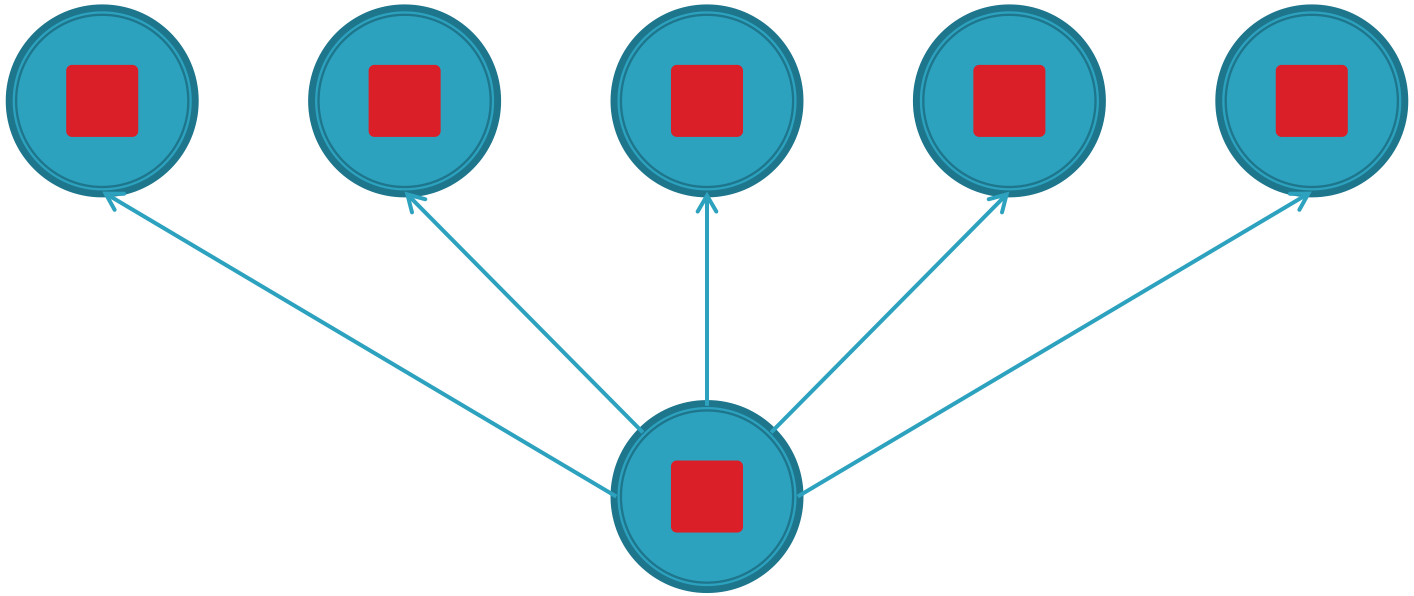
int MPI_Barrier **(MPI_Comm comm)**

- Специальная функция создающая в точке вызова барьер синхронизации для всех процессов коммуникатора comm
- Функцию вызывает первый достигший этой точки процесс
- Все процессы коммуникатора должны достигать этой точки

Пример 3

- В случае, если не все потоки достигают точки синхронизации, поведение программы становится неопределенным
- В данном случае программа останавливается и не завершается корректным образом

Широковещательная рассылка



Широковещательная рассылка

```
int MPI_Bcast(
        void *data, //Адрес первого передаваемого элемента
        int count, //Количество передаваемых элементов
        MPI_Datatype datatype, //Тип передаваемых элементов
        int source, //Номер передающего процесса
        MPI_comm comm) //Коммуникатор
```

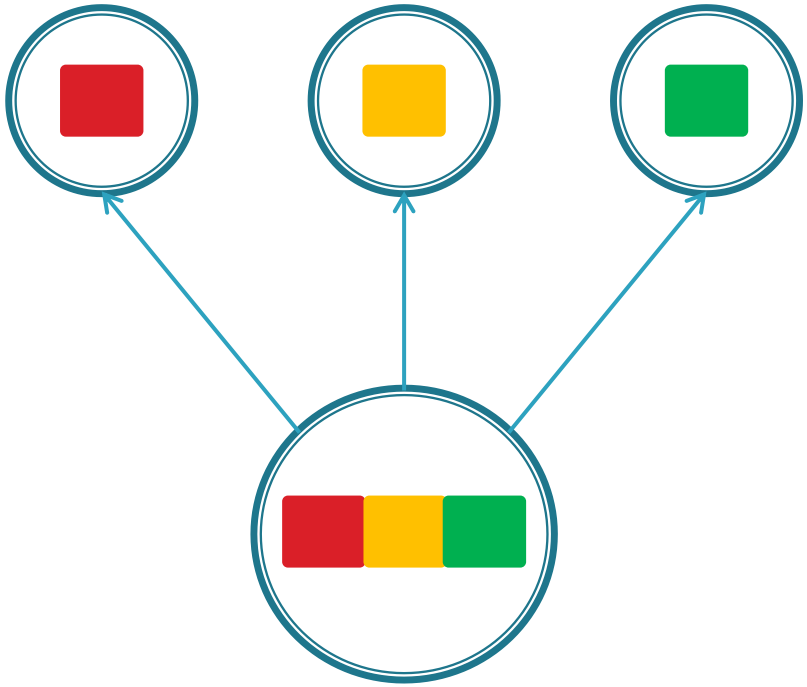
Широковещательная рассылка

- Процесс, указанный в параметре source рассылает всем остальным процессам буфер данных хранящийся в параметре data
- Эту функцию вызывают все процессы коммуникатора
- Принимающие процессоры, после вызова этой функции, получают в свой параметр data значения от рассылающего процесса

Пример 4

- Один из процессов изменяет данные, потом делает рассылку по всем остальным членам коммуникатора

Распределение данных



Распределение данных

```
int MPI_Scatter (
        void *send_data, //Адрес передаваемого буфера
        int send_count, //Количество передаваемых элементов
        MPI_Datatype send_type, //Тип отправляемых данных
        void *recv_data, //Адрес принимающего буфера
        int recv_count, //Количество принимаемых элементов
        MPI_Datatype recv_type, //Тип принимаемых  данных
        int root, //Номер передающего процесса
        MPI_Comm comm ) //Коммуникатор
```

Распределение данных

- Суть функции схожа с широковещательной рассылкой
- Отличия в том, что функция распределения данных рассылает не идентичный набор данных, а разные блоки данных для разных процессов
- Размеров блоков определяется параметрами send_count и send_type

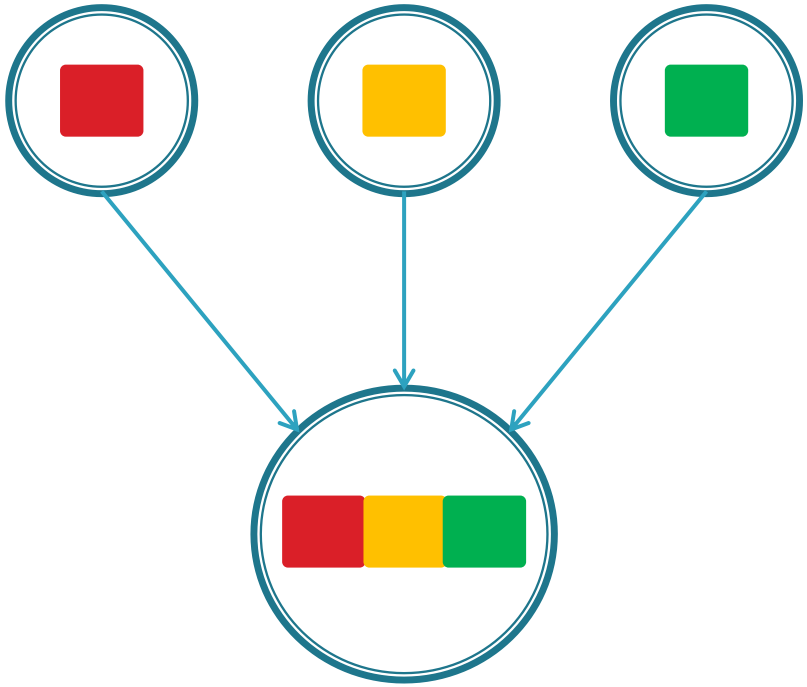
Распределение данных

```
int MPI_Scatterv (
        void *send_data, //Адрес передаваемого буфера
        int *send_count, //Массив количества передаваемых элементов
        int *displ, //Массив со смещениями передаваемого буфера
        MPI_Datatype send_type, //Тип отправляемых данных
        void *recv_data, //Адрес принимающего буфера
        int recv_count, //Количество принимаемых элементов
        MPI_Datatype recv_type, //Тип принимаемых  данных
        int root, //Номер передающего процесса
        MPI_Comm comm ) //Коммуникатор
```

Распределение данных

- Размеры массивов send_count и displ должны совпадать с размером коммуникатора comm
- Индекс элемента массивов соответствует порядковому номеру процесса
- Массив send_count содержит количества отправляемых элементов для каждого из процессов
- Массив displ содержит смещения относительно начала массива send_data

Сбор данных



Сбор данных

```
int MPI_Gather (
        void *send_data, //Адрес передаваемого буфера
        int send_count, //Количество передаваемых элементов
        MPI_Datatype send_type, //Тип отправляемых данных
        void *recv_data, //Адрес принимающего буфера
        int recv_count, //Количество принимаемых элементов
        MPI_Datatype recv_type, //Тип принимаемых  данных
        int root, //Номер принимающего процесса
        MPI_Comm comm ) //Коммуникатор
```

Сбор данных

- Синтаксис идентичен функции MPI_Scatter
- Параметр root означает не рассылающий, а принимающий процесс
- Процесс root собирает данные со всех процессов в свой принимающий буфер
- Параметр recv_count отражает количество элементов принимаемых от одного процесса, а не общее количество принимаемых элементов

Пример 5

- Распределенный подсчет среднего
- Сначала рассылаем блоки данных для каждого из процессов
- Собираем итоговые локальные средние и считаем конечный результат

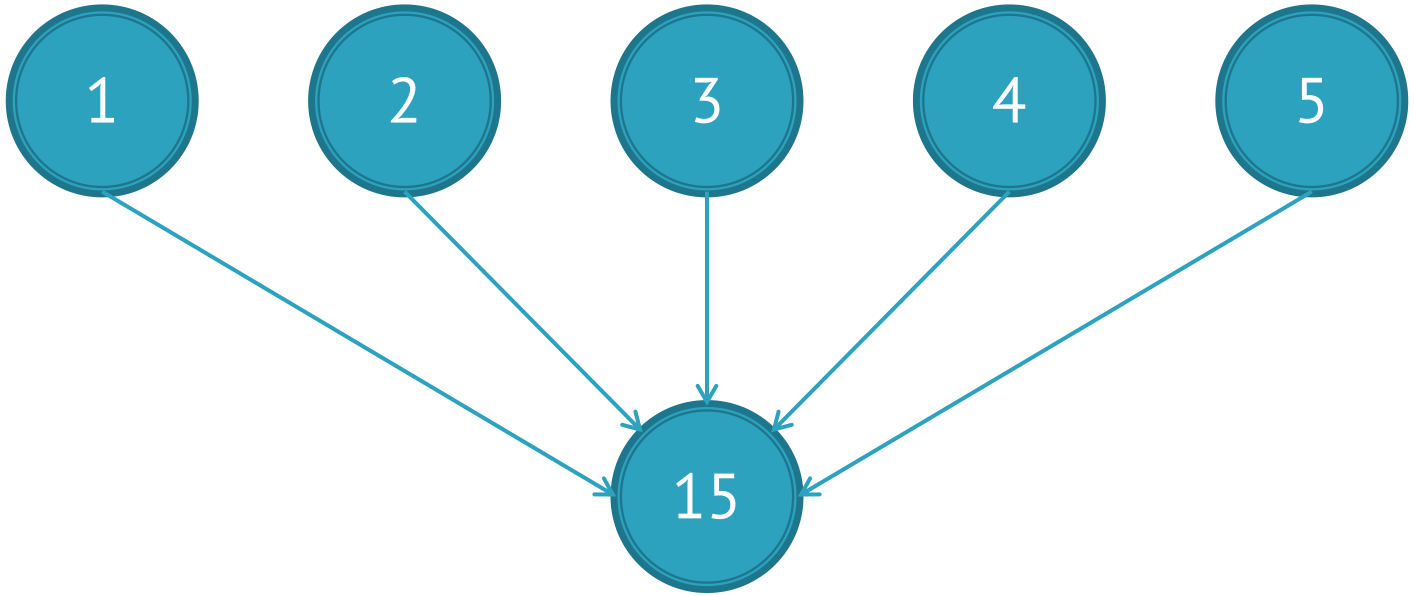
Сбор данных

```
int MPI_Allgather (
        void *send_data, //Адрес передаваемого буфера
        int send_count, //Количество передаваемых элементов
        MPI_Datatype send_type, //Тип отправляемых данных
        void *recv_data, //Адрес принимающего буфера
        int recv_count, //Количество принимаемых элементов
        MPI_Datatype recv_type, //Тип принимаемых  данных
        MPI_Comm comm ) //Коммуникатор
```

Сбор данных

- Данные собираются на всех процессах
- При использовании данной функции, все указатели должны указывать на корректные области памяти

Редукция



Редукция

```
MPI_Reduce(
        void *operand , //Адрес первого элемента редукции
        void *result, //Адрес первого элемента результата
        int count, //Количество элементов редукции
        MPI_Datatype type, //Тип элементов
        MPI_Op operator, //Оператор редукции
        int root, //Процесс получающий результат редукции
        MPI_Comm comm) //Коммуникатор
```

Редукция

- Функция принимает массив элементов и производит над ними операцию, определенную параметром operator
- В случае, если каждый из процессов содержит несколько элементов массива, будет произведена поэлементная редукция
- За корректность состояния всех соответствующих массивов отвечает программист

Операторы редукции

MPI_BAND	//Побитовое И
MPI_BOR	//Побитовое ИЛИ
MPI_BXOR	//Побитовое исключающее ИЛИ
MPI_LAND	//Логическое И
MPI_LOR	//Логическое ИЛИ
MPI_LXOR	//Логическое исключающее ИЛИ
MPI_MAX	//Максимум
MPI_MAXLOC	//Максимум и позиция максимума
MPI_MIN	//Минимум
MPI_MINLOC	//Минимум и позиция минимума
MPI_PROD	//Произведение
MPI_SUM	//Сумма

Пример 6

- Подсчет среднего с использованием редукции
- Частичные суммы подсчитываются каждым из процессов
- Затем суммы редуцируются при помощи соответствующей функции

Измерение времени в MPI

- Функция MPI_Wtime() – возвращает количество секунд прошедших от какого-то «момента в прошлом»
- Данный «момент в прошлом» гарантированно не изменяется на протяжении жизни процесса
- Время локально относительно узла
- Одинаковость времени не гарантирована для разных узлов

Пример 7

- Сравнение производительности различных вариантов подсчета среднего

Казанский федеральный
УНИВЕРСИТЕТ
ВЫСШАЯ ШКОЛА
информационных технологий
и информационных систем

Вопросы

ekhramch@kpfu.ru