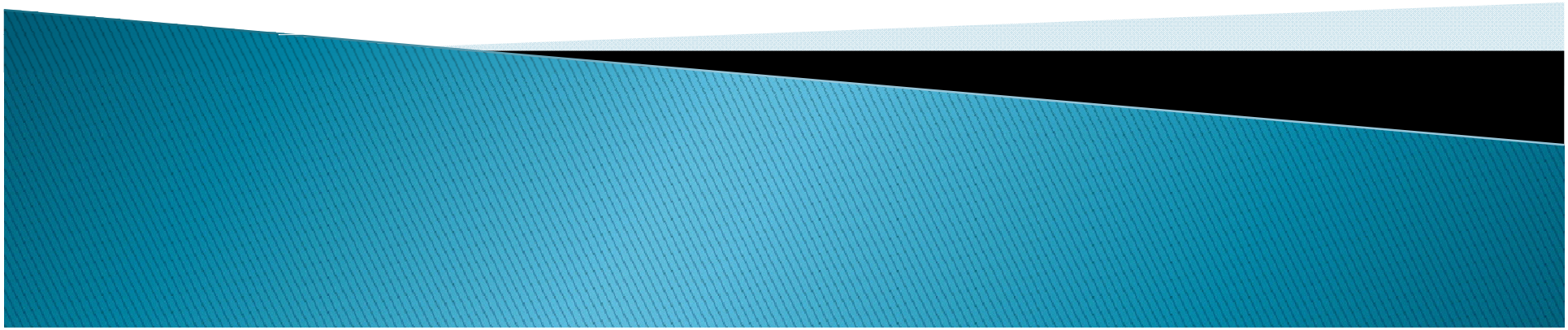
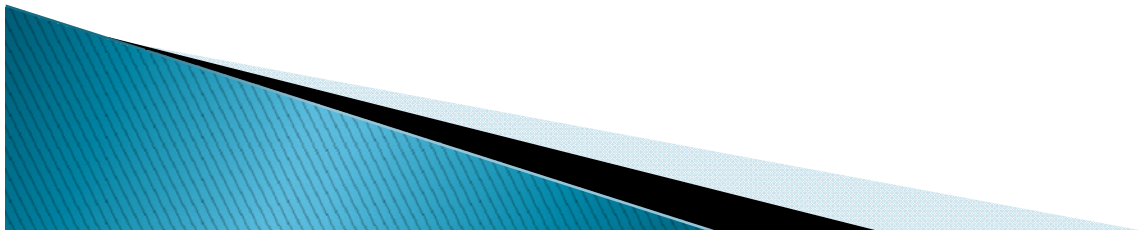
Обзор технологии параллельного программирования MPI



План лекции

- Стандарт MPI
- Основные понятия
- Блокирующие двухточечные обмены
- Двухточечные обмены с буферизацией, другие типы двухточечных обменов
- Коллективные операции



Рейтинг TOP-500

TOP 500 The List.

PRESENTED BY

ICL INNOVATIVE COMPUTING LABORATORY

BERKELEY LAB Lawrence Berkeley National Laboratory

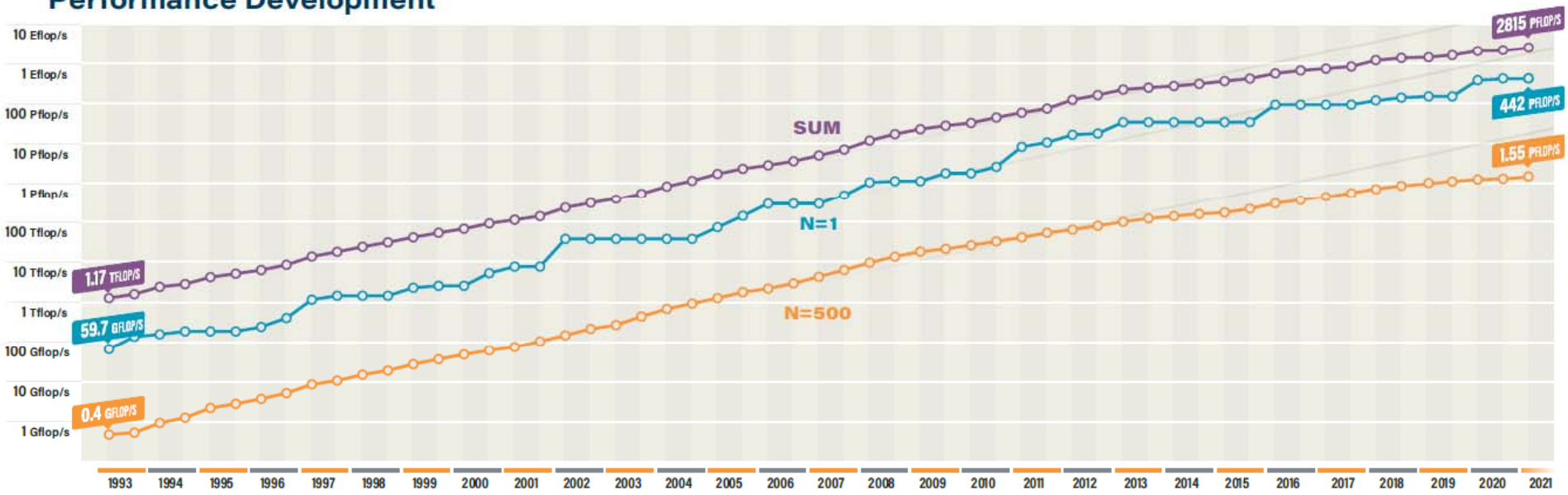
ISC GROUP Moving Forward.

FIND OUT MORE AT

top500.org

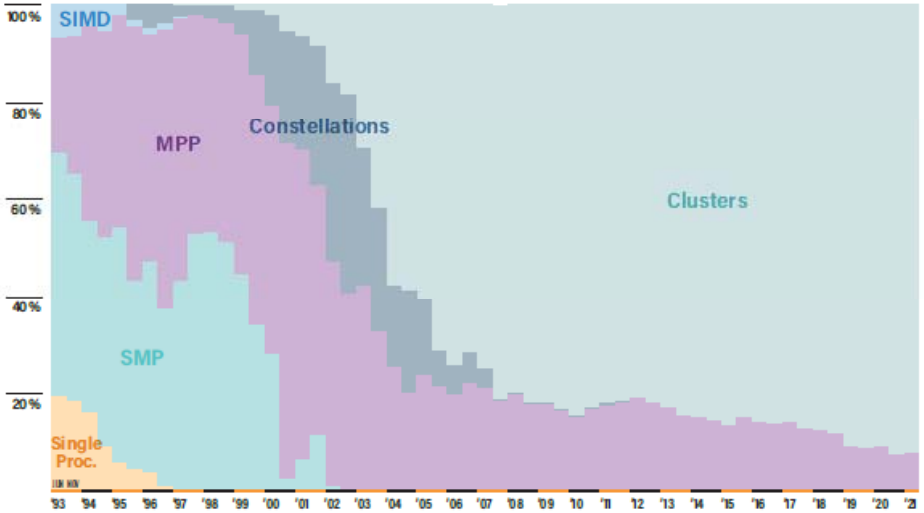
JUNE 2021	SYSTEM	SPECS	SITE	COUNTRY	CORES	RMAX PFLOP/S	POWER MW
1	Fugaku	Fujitsu A64FX (48C, 2.2GHz), Tofu Interconnect D	RIKEN R-CCS	Japan	7,630,848	442.0	29.9
2	Summit	IBM POWER9 (22C, 3.07GHz), NVIDIA Volta GV100 (80C), Dual-Rail Mellanox EDR Infiniband	DOE/SC/ORNL	USA	2,414,592	148.6	10.1
3	Sierra	IBM POWER9 (22C, 3.1GHz), NVIDIA Tesla V100 (80C), Dual-Rail Mellanox EDR Infiniband	DOE/NNSA/LLNL	USA	1,572,480	94.6	7.44
4	Sunway TaihuLight	Shenwei SW26010 (260C, 1.45 GHz) Custom Interconnect	NSCC in Wuxi	China	10,649,600	93.0	15.4
5	Perlmutter	HPE Cray EX235n, AMD EPYC 7763 (64C, 2.45GHz), NVIDIA A100 SXM4 40 GB, Slingshot-10 (274 GB)	DOE/SC/LBNL	USA	761,856	64.9	2.53

Performance Development

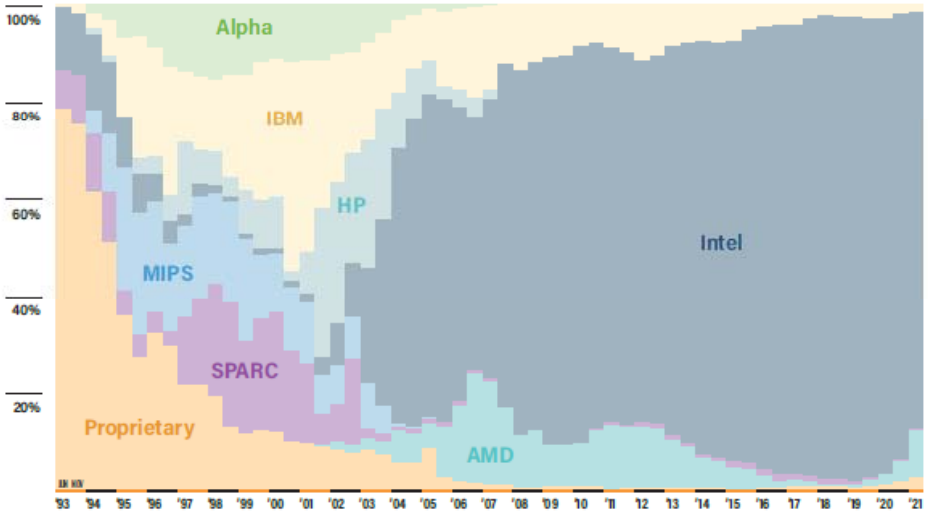


Рейтинг TOP-500

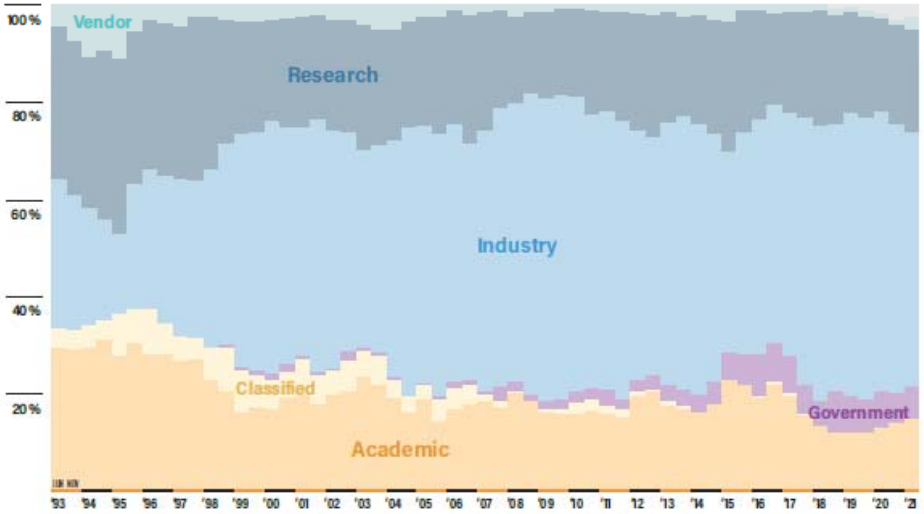
Architectures



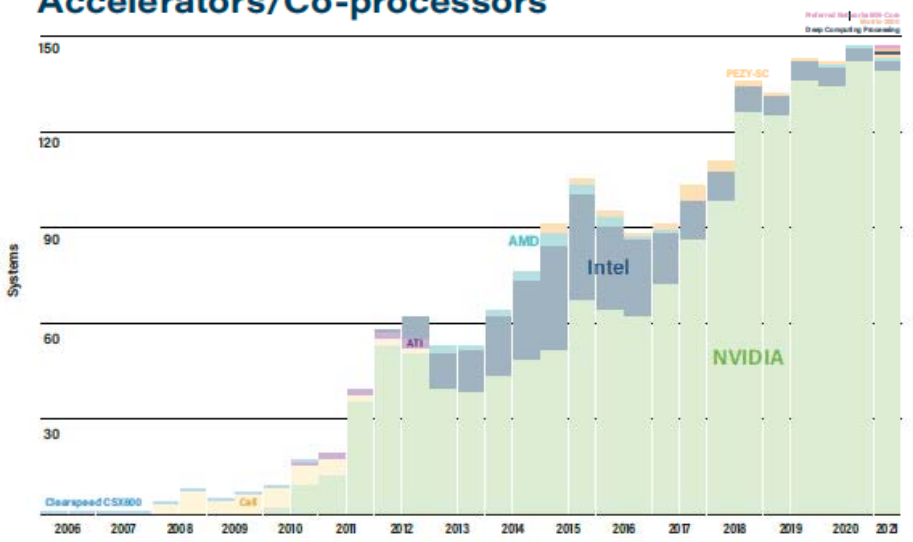
Chip Technology



Installation Type



Accelerators/Co-processors



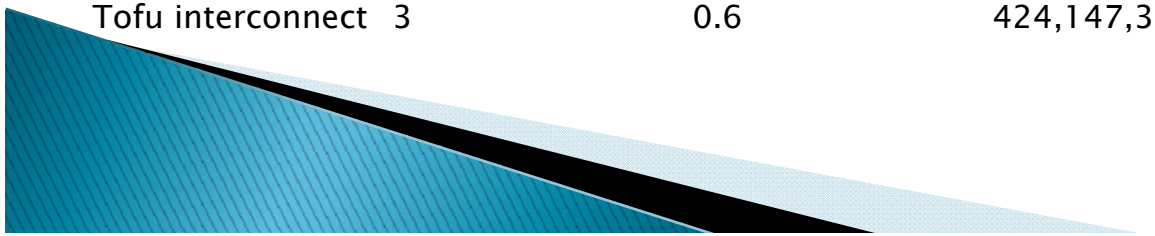
TOP-500. Ноябрь 2021

- Три суперкомпьютера «Яндекса» вошли в мировой рейтинг Top500. «Червоненкис», «Галушкин» и «Ляпунов» заняли 19, 36 и 40 строчки соответственно.

- «Червоненкис» включает 199 вычислительных узлов, во время теста он показал производительность 21,53 петафлопса. Производительность «Галушкина» и «Ляпунова» составила 16,02 и 12,81 петафлопса соответственно. Все три суперкомпьютера построены на базе процессоров AMD EPYC и графических ускорителей NVIDIA A100.
- Также в рейтинг попали системы «Кристофари Neo»(43) и «Кристофари»(72) от Сбера, «Ломоносов-2» разработки МГУ(241) и MTS Grom(294) от оператора МТС.

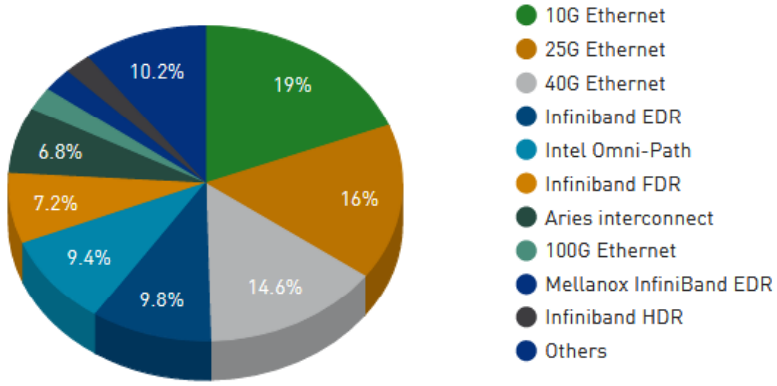
Рейтинг TOP-500

Interconnect	Count	System Share (%)	Rmax (GFlops)	Rpeak (GFlops)	Cores
10G Ethernet	95	19	160,861,930	433,349,075	9,015,160
25G Ethernet	80	16	159,047,540	318,120,729	3,946,560
40G Ethernet	73	14.6	120,385,030	154,736,640	4,204,800
Infiniband EDR	49	9.8	144,217,082	277,283,465	4,951,850
Intel Omni-Path	47	9.4	183,378,118	292,177,907	4,682,000
Infiniband FDR	36	7.2	80,012,743	119,387,030	2,750,824
Aries interconnect	34	6.8	155,747,832	234,957,652	5,500,164
100G Ethernet	12	2.4	23,436,900	44,971,200	501,940
Mellanox InfiniBand EDR	12	2.4	62,946,240	92,035,501	1,095,392
Infiniband HDR	11	2.2	22,818,088	33,917,463	666,320
Dual-rail Mellanox EDR Infiniband	9	1.8	313,174,000	421,169,933	5,167,176
Mellanox InfiniBand HDR100	8	1.6	33,900,690	49,045,299	929,856
InfiniBand HDR100	7	1.4	16,655,550	28,202,803	453,888
Mellanox HDR Infiniband	5	1	85,978,470	122,766,902	1,522,656
Custom Interconnect	3	0.6	7,311,408	8,642,512	587,232
Tofu interconnect	3	0.6	424,147,300	523,999,642	7,446,528

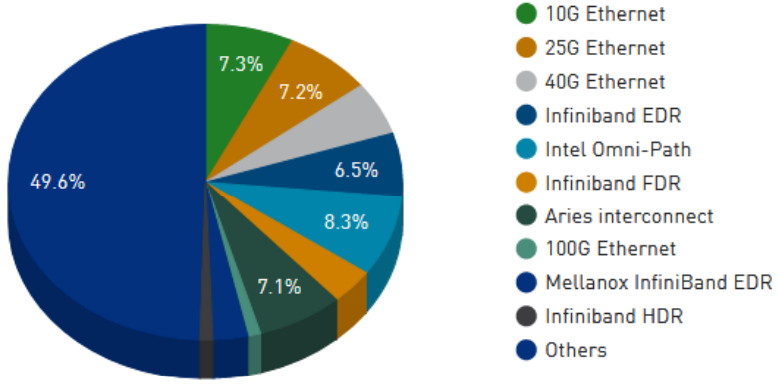


Рейтинг TOP-500

Interconnect System Share



Interconnect Performance Share



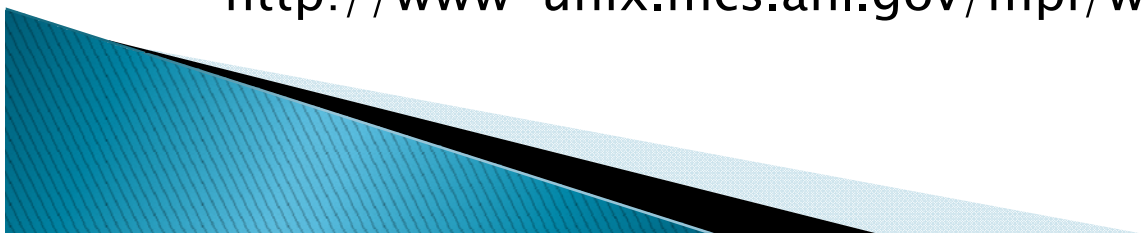
MPI

- MPI 1.1 Standard разрабатывался 92-94
- MPI 2.0 – 95-97
- MPI 2.1 – сентябрь 2008 г.
- MPI 3.0 – сентябрь 2012 г.
- MPI 3.1 – июнь 2015 г.
- MPI 4.0 – июнь 2021 г.

Более 450 процедур

- Стандарты
  http://www.mpi-forum.org/
  https://computing.llnl.gov/tutorials/mpi/

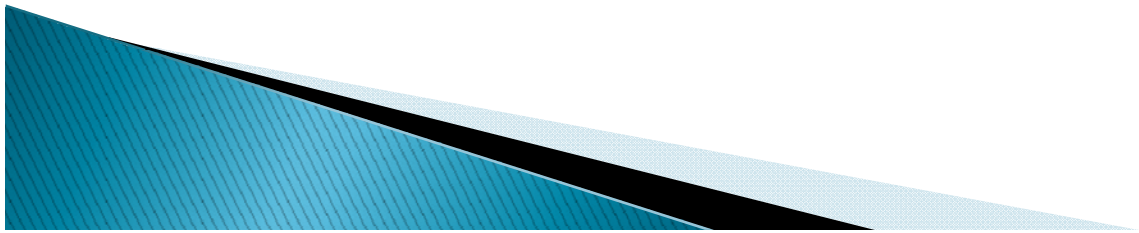
- Описание функций
  http://www-unix.mcs.anl.gov/mpi/www/



Цель MPI

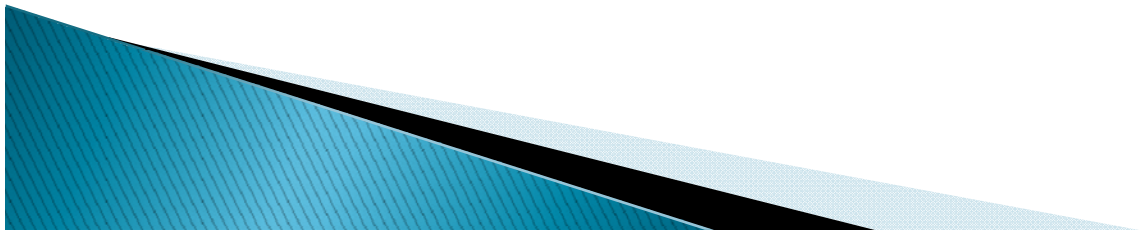
- Основная цель:
  - Обеспечение переносимости исходных кодов
  - Эффективная реализация

- Кроме того:
  - Большая функциональность
  - Поддержка неоднородных параллельных архитектур

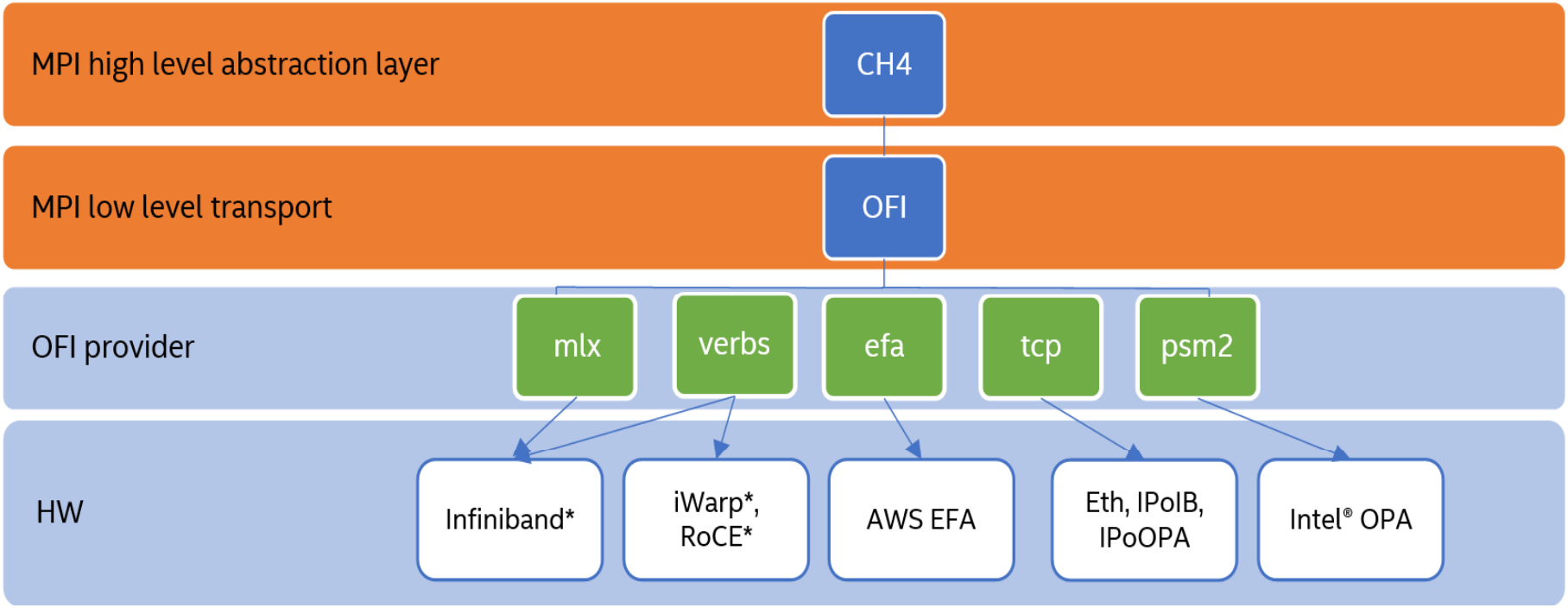


Реализации MPI

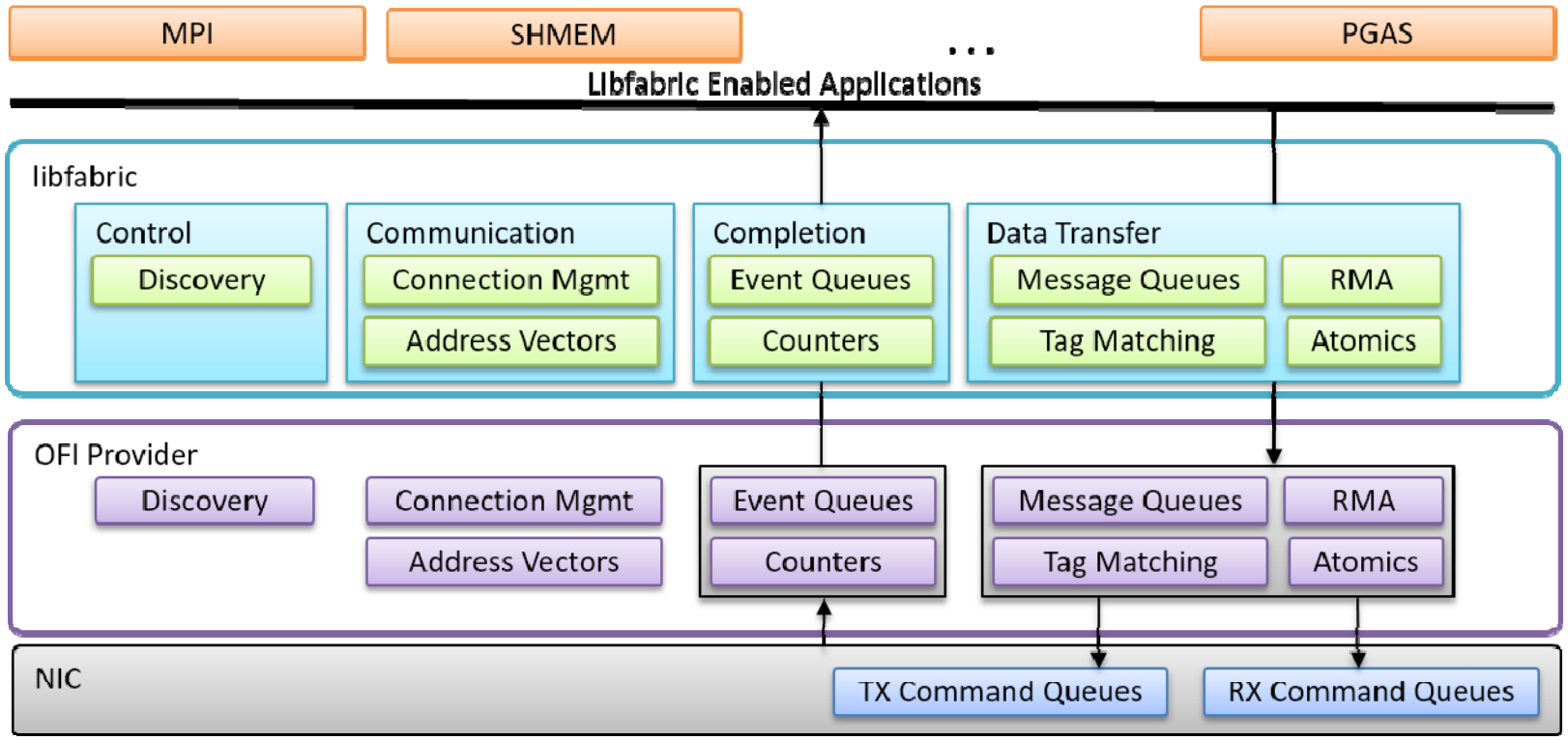
- MPICH
- LAM/MPI
- Mvapich
- OpenMPI
- Коммерческие реализации Intel, IBM и др.



Реализация MPI



Реализация MPI. Libfabric OpenFabrics

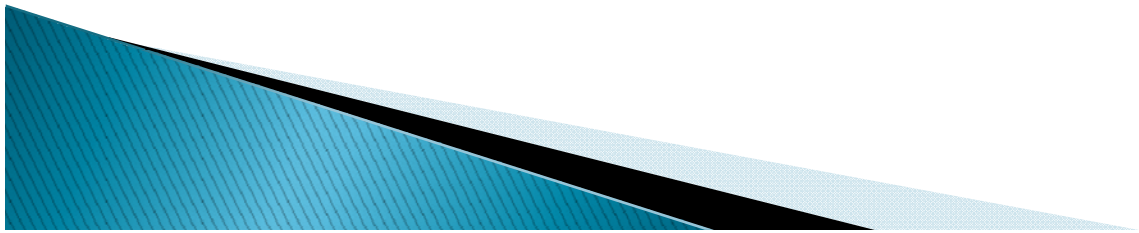


Модель MPI

- Параллельная программа состоит из процессов, процессы могут быть многопоточными.
- MPI реализует передачу сообщений между процессами.
- Межпроцессное взаимодействие предполагает:
  - синхронизацию
  - перемещение данных из адресного пространства одного процесса в адресное пространство другого процесса.

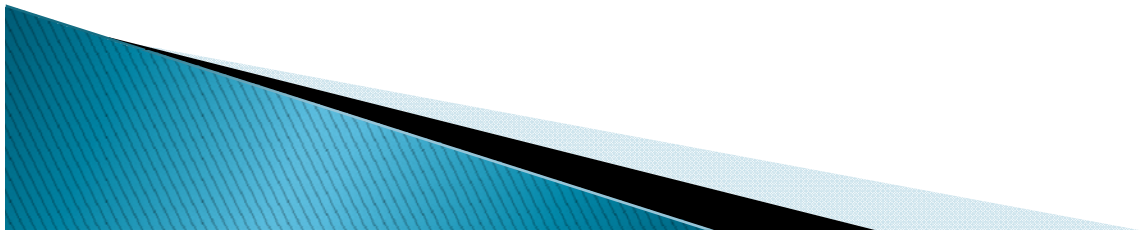
Основные понятия

- Процессы объединяются в группы.
- Каждое сообщение посылается в рамках некоторого контекста и должно быть получено в том же контексте.
- Группа и контекст вместе определяют коммуникатор.
- Процесс идентифицируется своим номером в группе, ассоциированной с коммуникатором.
- Коммуникатор, содержащий все начальные процессы, называется MPI_COMM_WORLD.



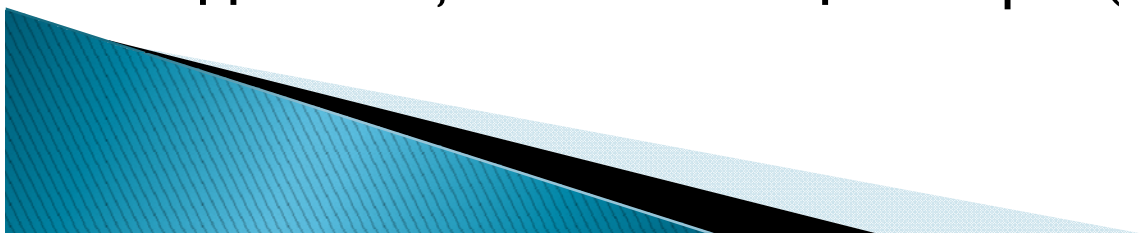
Понятие коммуникатора MPI

- Управляющий объект, представляющий группу процессов, которые могут взаимодействовать друг с другом
- Все обращения к MPI функциям содержат коммуникатор, как параметр
- Наиболее часто используемый коммуникатор MPI_COMM_WORLD
- Определяется при вызове MPI_Init
- Содержит ВСЕ процессы программы



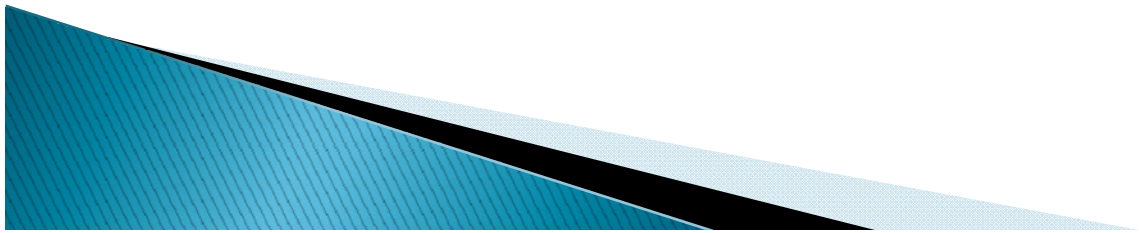
Типы данных MPI

- Данные в сообщении описываются тройкой: (address, count, datatype), где datatype определяется рекурсивно как:
  - Предопределенный базовый тип, соответствующий типу данных в базовом языке (например, MPI_INT, MPI_DOUBLE_PRECISION)
  - Непрерывный массив MPI типов
  - Векторный тип
  - Индексированный тип
  - Произвольные структуры
- MPI включает функции для построения пользовательских типов данных, например, типа данных, описывающих пары (int, float).



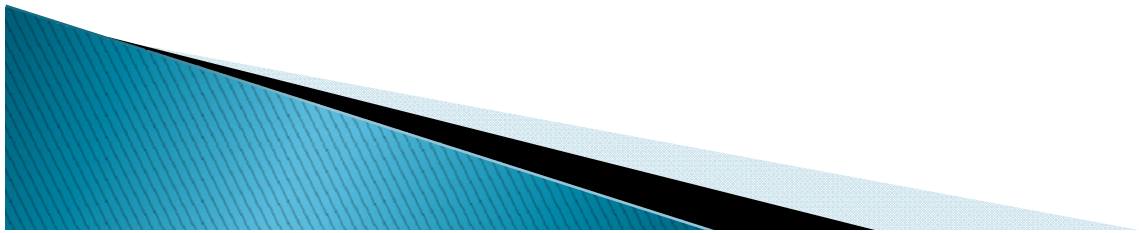
Базовые типы данных

MPI datatype	C datatype
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



Понятие тега

- Сообщение сопровождается определяемым пользователем признаком для идентификации принимаемого сообщения.
- Теги сообщений у отправителя и получателя должны быть согласованы.
- Можно указать в качестве значения тэга константу MPI_ANY_TAG.
- Некоторые не-MPI системы передачи сообщений называют тэг типом сообщения.
- MPI вводит понятие тэга, чтобы не путать это понятие с типом данных MPI.



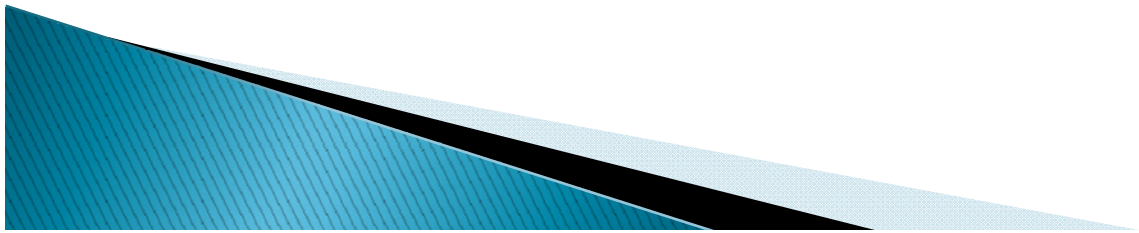
Формат MPI-функций

error = MPI_Xxxxx(parameter,...);
MPI_Xxxxx(parameter,...);

- Возвращаемое значение – код ошибки. Определяется константой MPI_SUCCESS

```
int error;
.....
error = MPI_Init(&argc, &argv));
if (error != MPI_SUCCESS)
{
        fprintf (stderr, " MPI_Init error \n");
        return 1;

}
```

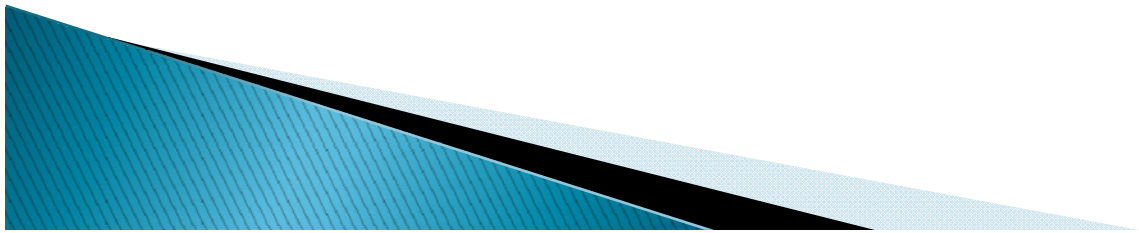


Выполнение MPI-программы

- При запуске указываем число требуемых процессоров np и название программы
  mpirun -np 3 prog
- На выделенных для расчета узлах запускается np копий указанной программы
- Каждая копия программы получает два значения:
  - **np**
  - **rank** из диапазона [0 ... np-1]
- Любые две копии программы могут непосредственно обмениваться данными с помощью функций передачи сообщений

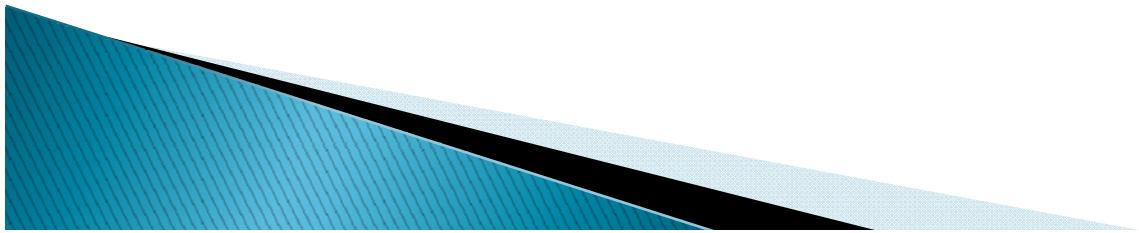
C: MPI helloworld.c

```
#include <mpi.h>
main(int argc, char **argv)
{
  int numtasks, rank;
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &
  numtasks);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  printf("Hello World from process %d of %d\n“,
  rank, numtasks);
  MPI_Finalize();
}
```



Функции определения среды

- **int MPI_Init(int *argc, char ***argv)**
  должна первым вызовом, вызывается только один раз
- **int MPI_Comm_size(MPI_Comm comm, int *size)**
  число процессов в коммуникаторе
- **int MPI_Comm_rank(MPI_Comm comm, int *rank)**
  номер процесса в коммуникаторе (нумерация с 0)
- **int MPI_Finalize()**
  завершает работу процесса
- **int MPI_Abort (MPI_Comm comm, int*errorcode)**
  завершает работу программы



Функции определения среды

- **int MPI_Initialized(int *flag)**

В аргументе **flag** возвращает 1, если вызвана после процедуры **MPI_Init**, и 0 в противном случае.

- **int MPI_Finalized(int *flag)**

В аргументе **flag** возвращает 1, если вызвана после процедуры **MPI_Finalize** , и 0 в противном случае.

Эти процедуры можно вызвать до **MPI_Init** и после **MPI_Finalize.**

- **int MPI_Get_processor_name(char *name, int *len)**

Возвращает в строке **name** имя узла, на котором запущен вызвавший процесс. В переменной **len** возвращается количество символов в имени, не превышающее константы **MPI_MAX_PROCESSOR_NAME.**

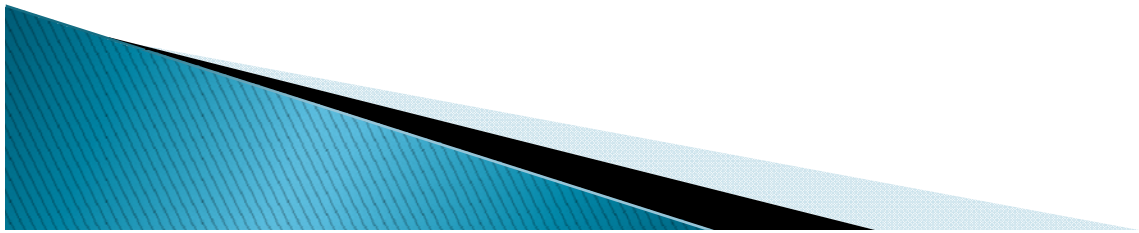
Функции работы с таймером

- **double MPI_Wtime(void)**

  Возвращает для каждого вызвавшего процесса астрономическое время в секундах (вещественное число двойной точности), прошедшее с некоторого момента в прошлом. Момент времени, используемый в качестве точки отсчёта, не будет изменён за время существования процесса.

- **double MPI_Wtick(void)**

  Возвращает разрешение таймера в секундах.



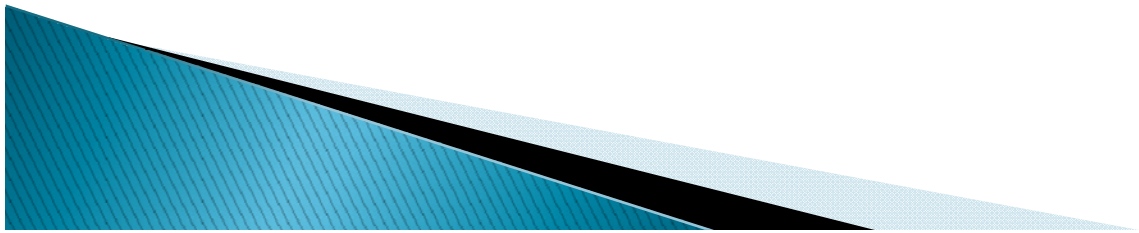
Использование таймера

```
#include <stdio.h>
#include "mpi.h"
#define NTIMES 1000
int main(int argc, char **argv)
{
   double time_start, time_finish, tick;
   int rank, i;
   int len;
   char *name;
   name = (char*)malloc(MPI_MAX_PROCESSOR_NAME*sizeof(char));
   MPI_Init(&argc, &argv);
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
   MPI_Get_processor_name(name, &len);
   tick = MPI_Wtick();
   time_start = MPI_Wtime();
   for (i = 0; i<NTIMES; i++)time_finish = MPI_Wtime();
   printf ("node %s, process %d: tick= %lf, time= %lf\n",
        name, rank, tick, (time_finish-time_start)/NTIMES);
   MPI_Finalize();
}
```

Информация о статусе сообщения

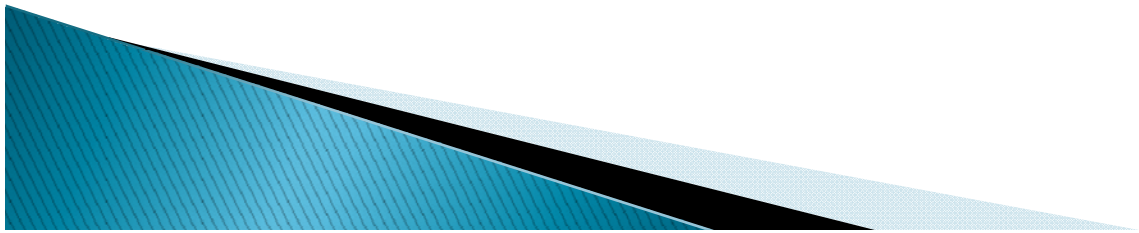
Содержит:

- Source: status.MPI_SOURCE
- Tag: status.MPI_TAG
- Код ошибки: status.MPI_ERROR
- Count: MPI_Get_count

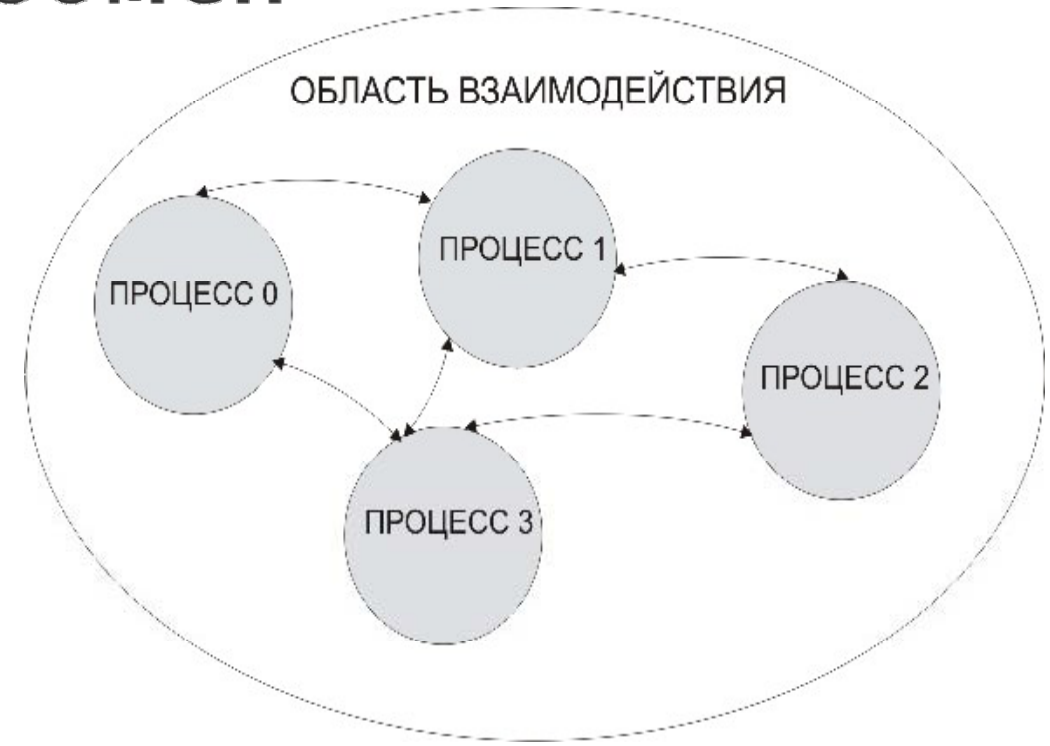


Двухточечный (point-to-point, p2p) обмен

- В двухточечном обмене участвуют только два процесса, процесс-отправитель и процесс-получатель (источник сообщения и адресат).
- Двухточечные обмены используются для организации локальных и неструктурированных коммуникаций.



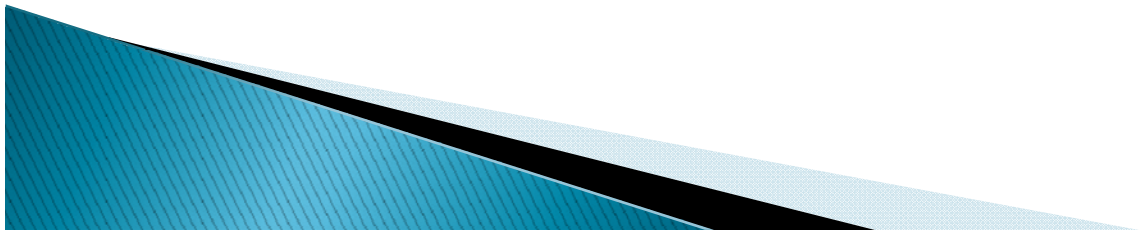
Двухточечный (point-to-point, p2p) обмен



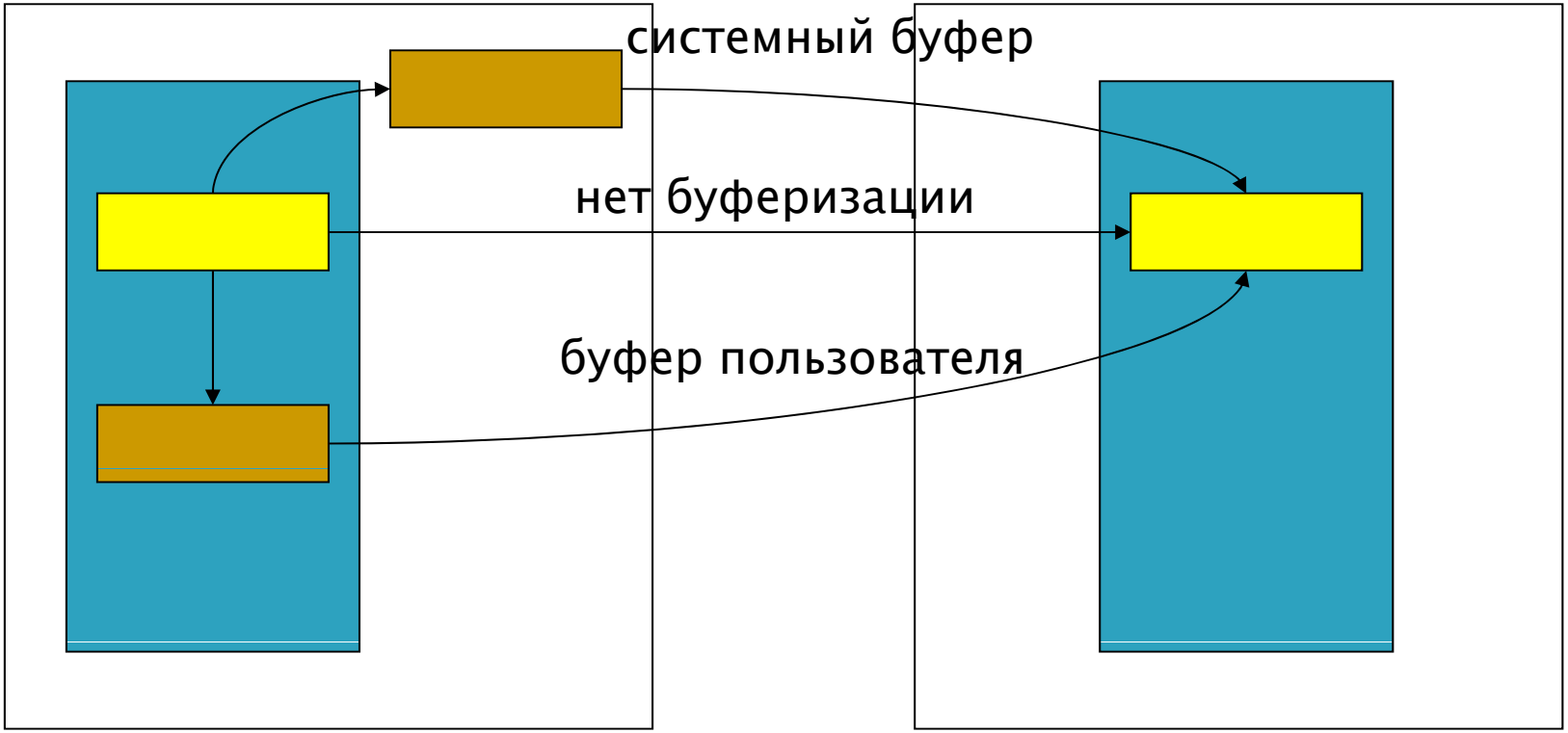
- Двухточечный обмен возможен только между процессами, принадлежащими одной области взаимодействия (одному коммуникатору).

Условия успешного взаимодействия точка-точка

- Отправитель должен указать правильный rank получателя
- Получатель должен указать верный rank отправителя
- Одинаковый коммуникатор
- Тэги должны соответствовать друг другу
- Буфер у процесса-получателя должен быть достаточного объема



Выполнение двухточечных обменов



Разновидности двухточечного обмена

- *блокирующие* прием/передача, которые приостанавливают выполнение процесса на время приема или передачи сообщения;
- *неблокирующие* прием/передача, при которых выполнение процесса продолжается в фоновом режиме, а программа в нужный момент может запросить подтверждение завершения приема сообщения.