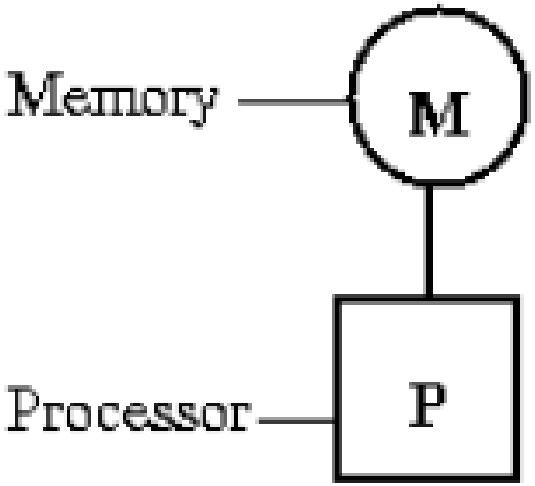
Message Passing Interface (MPI)

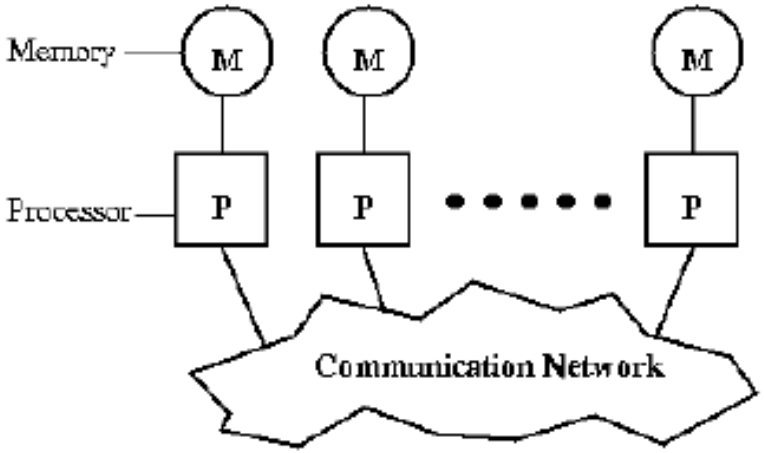
Последователно програмиране

- Опростен изглед на целевата машина – единичен процесор с памет
- Единична програма за изпълнение
- Програмата може да се пренесе на друга последователна архитектура



Message-passing парадигма

- Развитие с цел паралелно програмиране
- Множество процесори със собствена памет
- Програмата се изпълнява на всеки процесор
- Комуникация между процесите чрез съобщения



Message-passing модел

- Простота
- Универсалност
- Лесно дебъгване
- Производителност

Message Passing Interface

- Стандарт за интерфейс, базиран на съобщения
- Реализация като библиотека
- Поддържа Fortran, C, C++ езици
- Без комуникационни сървъри
- Платформа – високо производителни компютри
- Проектиран за паралелни приложения

Message Passing Interface - особености

- Предаване на съобщения от тип Point-to-point:
  - две базови операции *send* и *receive*
  - различни техни вариации

Message Passing Interface - особености

- Колективни комуникации – дава възможност голям брой процеси да си комуникират:
  - Операции за преместване на данни между процесите
  - Колективни операции по изчисления

Message Passing Interface - особености

- Поддръжка на групи от процеси
  - Група - подредена съвкупност от процеси
  - Всеки процес се идентифицира с *rank*
  - Описва кои процеси са въвлечени в колективни комуникации
  - Различни групи могат да изпълняват паралелно различни задачи

Message Passing Interface - особености

- Поддръжка на контекст на комуникацията:
  - Разделя фамилии от съобщения
  - Създаване на независими комуникационни потоци между процеси
  - Независимост на паралелните библиотеки от потребителския код

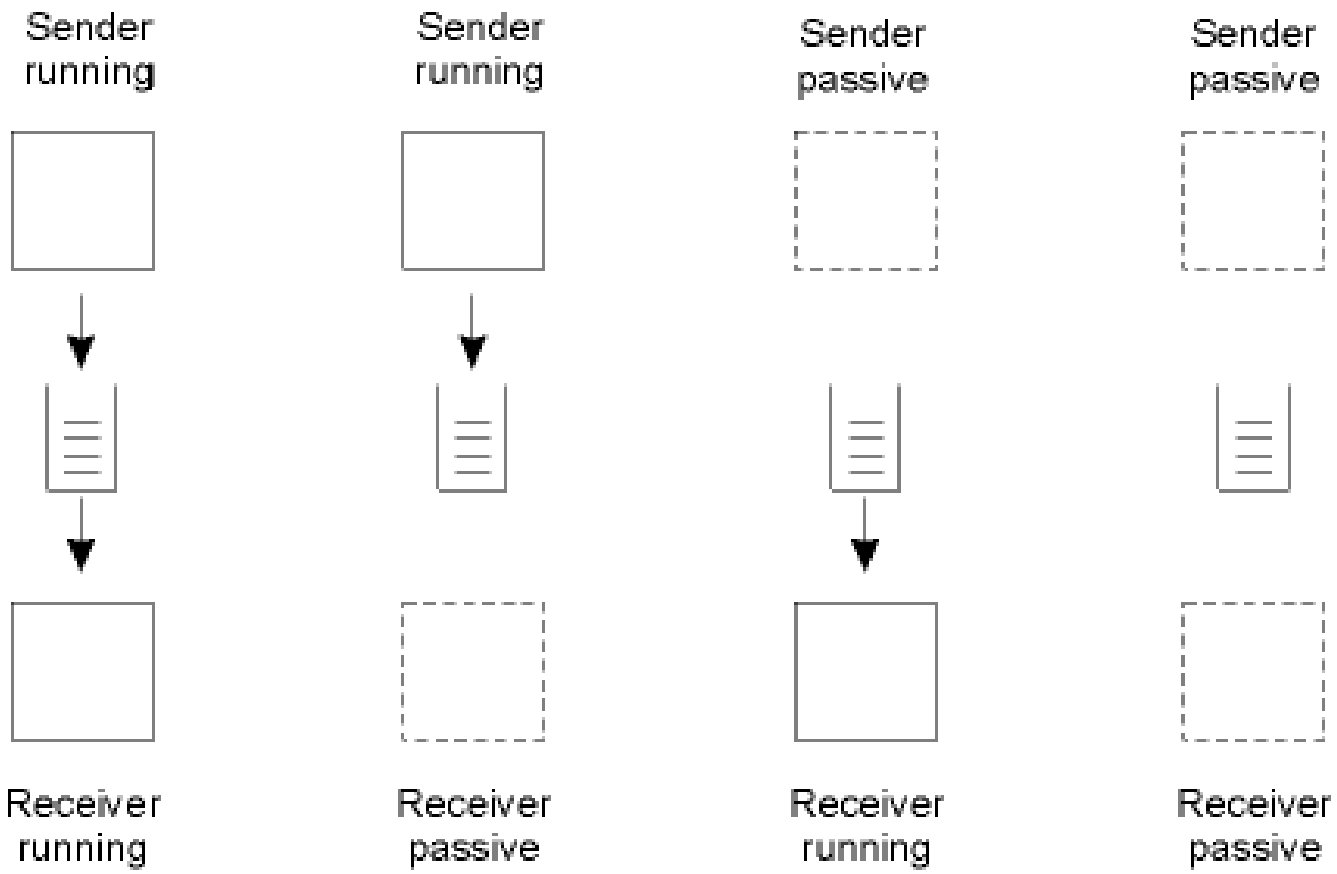
Message Passing Interface - особености

- Поддръжка на приложни топологии:
  - Процесите за организирани в топологии (grid)
  - Дава възможност за метод от високо ниво за манипулиране на групи процеси

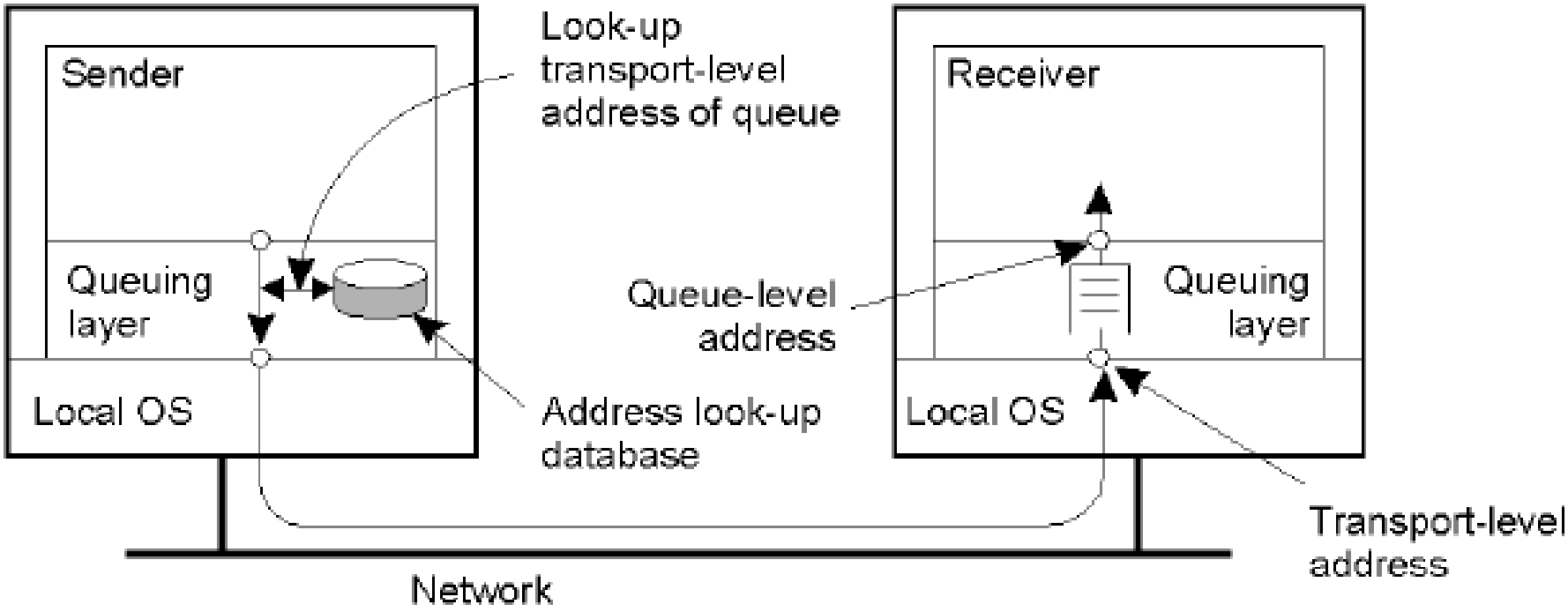
Процесен модел

- Основната изчислителна единица е *процес*
- MPI моделът е статичен – след стартиране на приложението съществува фиксиран брой процеси
- Процесите принадлежат на групи
- Групите са динамични
- Процесите се идентифицират с номер на група и rank

Message-queuing Модел

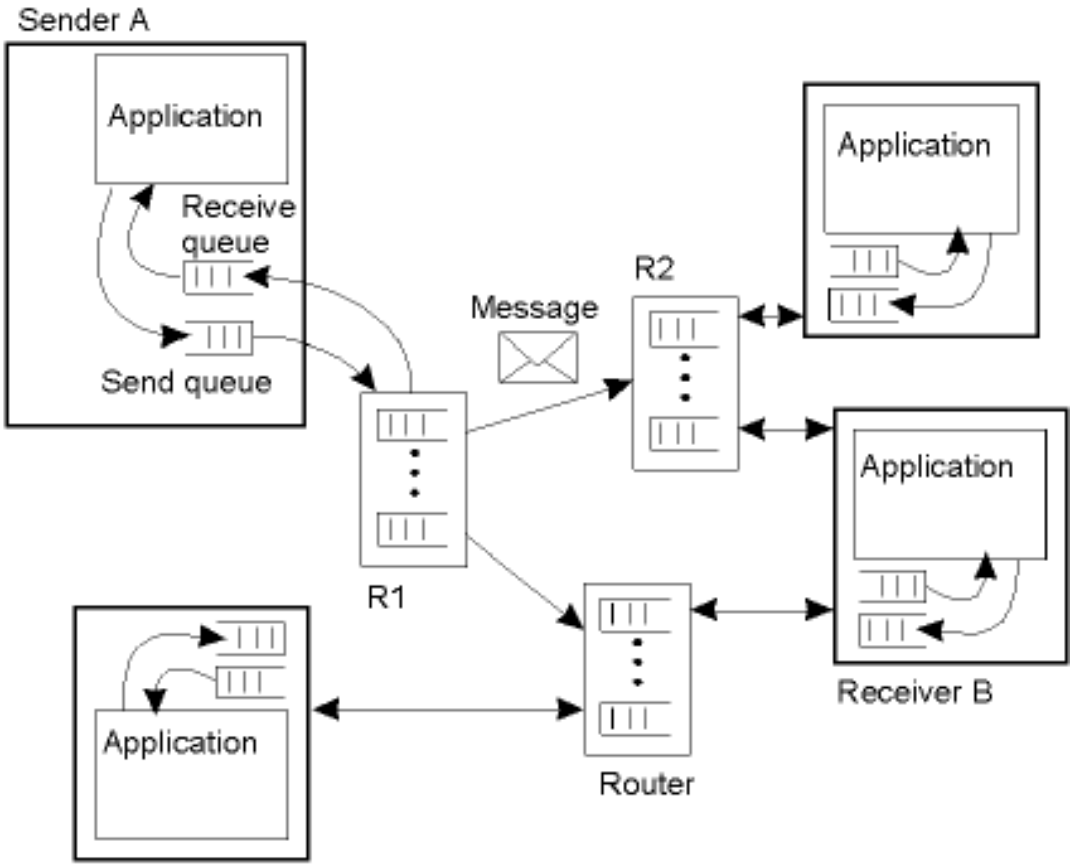


Архитектура на Message-queuing система



Адресиране на ниво опашка и на ниво мрежа

Архитектура на Message-queuing система



Организация на система с маршрутизатори

Създаване на MPI програма

- Създава се само една програма
- Всеки възел изпълнява тази програма
- Ако е необходимо някой възел да изпълни част от кода, това се указа явно

Пример:

```
if (MyID = 2) {
...
}
```

Основни моменти

- Брой процеси/възли – *MPI_Comm_size()*
- Област на комуникационните операции – контекст
- Контекстът и групата са комбинирани в един обект - комуникатор
- В повечето случаи се използва един комуникатор: MPI_COMM_WORLD
- Ранг на текущ процес: *MPI_Comm_rank()*

Структура на MPI програма

```
#include "mpi.h"
...
main(int argc, char** argv) {
  ...
  /* No MPI functions called before this */
  MPI_Init(&argc, &argv);
  ...
  MPI_Finalize();
  /* No MPI functions called after this */
  ...
} /* main */
```

Пример на MPI програма

```
#include <stdio.h>
#include "mpi.h"
int main(argc,argv)
  int argc;
  char *argv[]; {
  int myrank; /* Rank of process */
  int numprocs; /* Number of processes */
  MPI_Init(&argc,&argv);
  MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
  MPI_Comm_rank(MPI_COMM_WORLD,&myrank);
  printf( "Process %d of %d says Hello!\n",
            myrank, numprocs);
  MPI_Finalize();
}
```

Стартиране на MPI програма

- Компилиране и свързване с MPI библиотека
- Стартиране:

 # hello –np 4

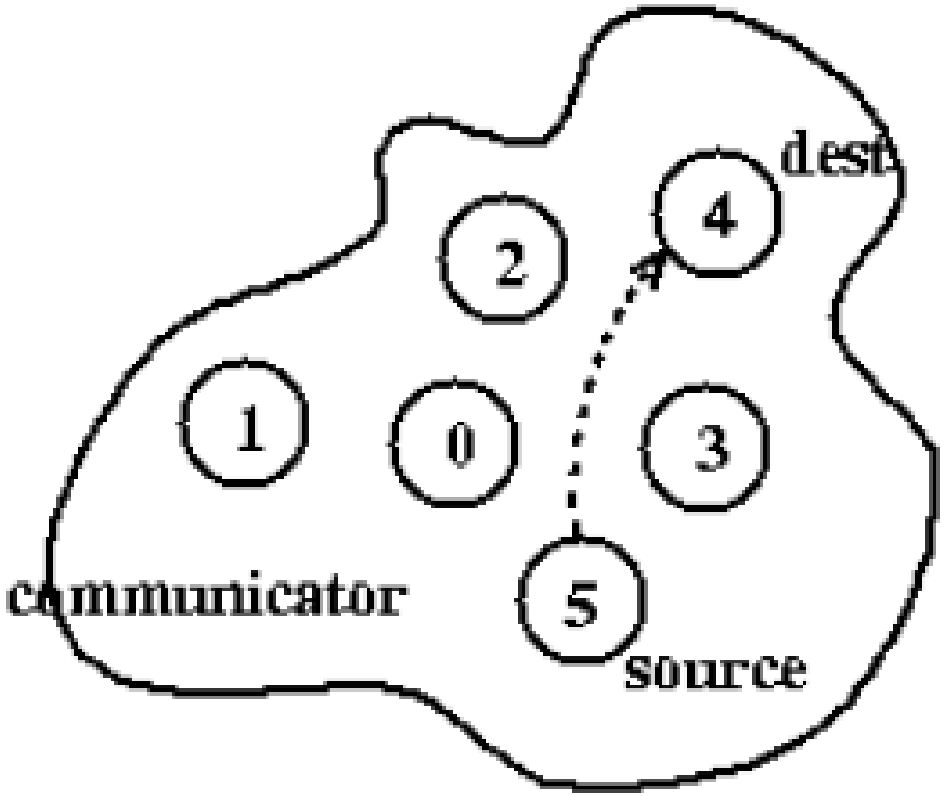
Process 1 of 4 says Hello!

Process 2 of 4 says Hello!

Process 0 of 4 says Hello!

Process 3 of 4 says Hello!

Point-to-point комуникации



Point-to-point комуникации - send

```
int MPI_Send(void *buf,
        int count,
        MPI_Datatype datatype,
        int dest,
        int tag,
        MPI_Comm comm)
```

Блокираща операция

За неблокираща – MPI_Isend()

Point-to-point комуникации - receive

```
int MPI_Recv(void *buf,
        int count,
        MPI_Datatype datatype,
        int source,
        int tag,
        MPI_Comm comm,
        MPI_Status *status);
```

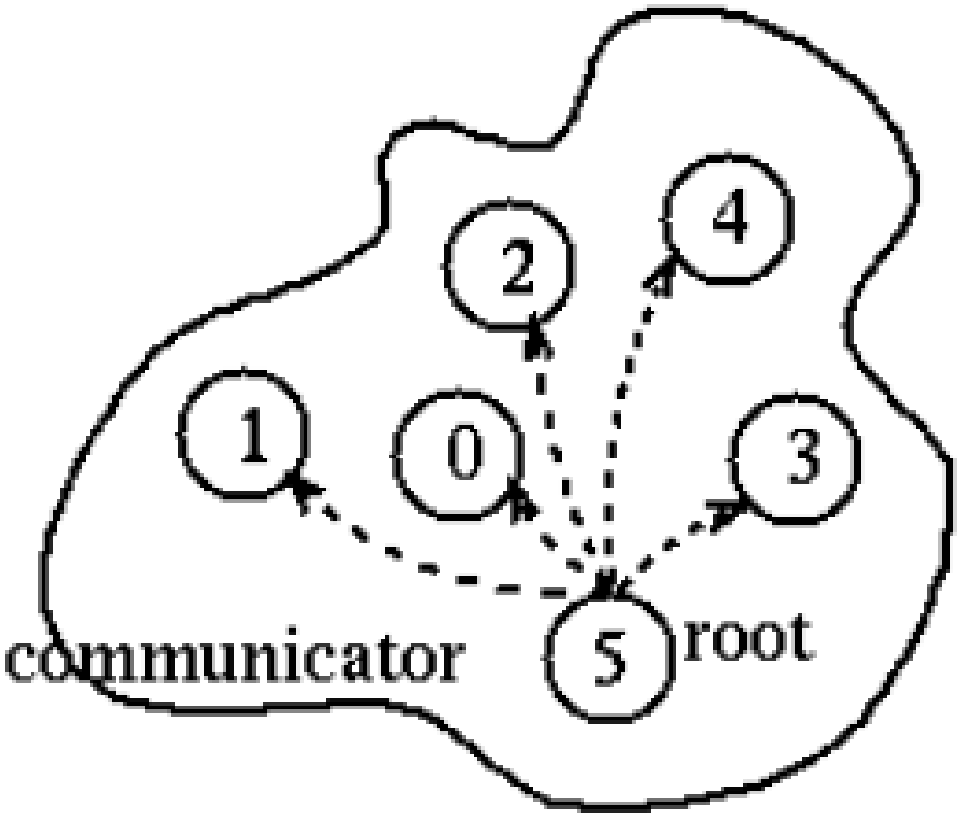
Блокираща операция

За неблокираща – MPI_Irecv()

Основни типове данни

MPI datatype	C datatype
MPI_CHAR	signed char
MPI_SHORT	signed short int
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

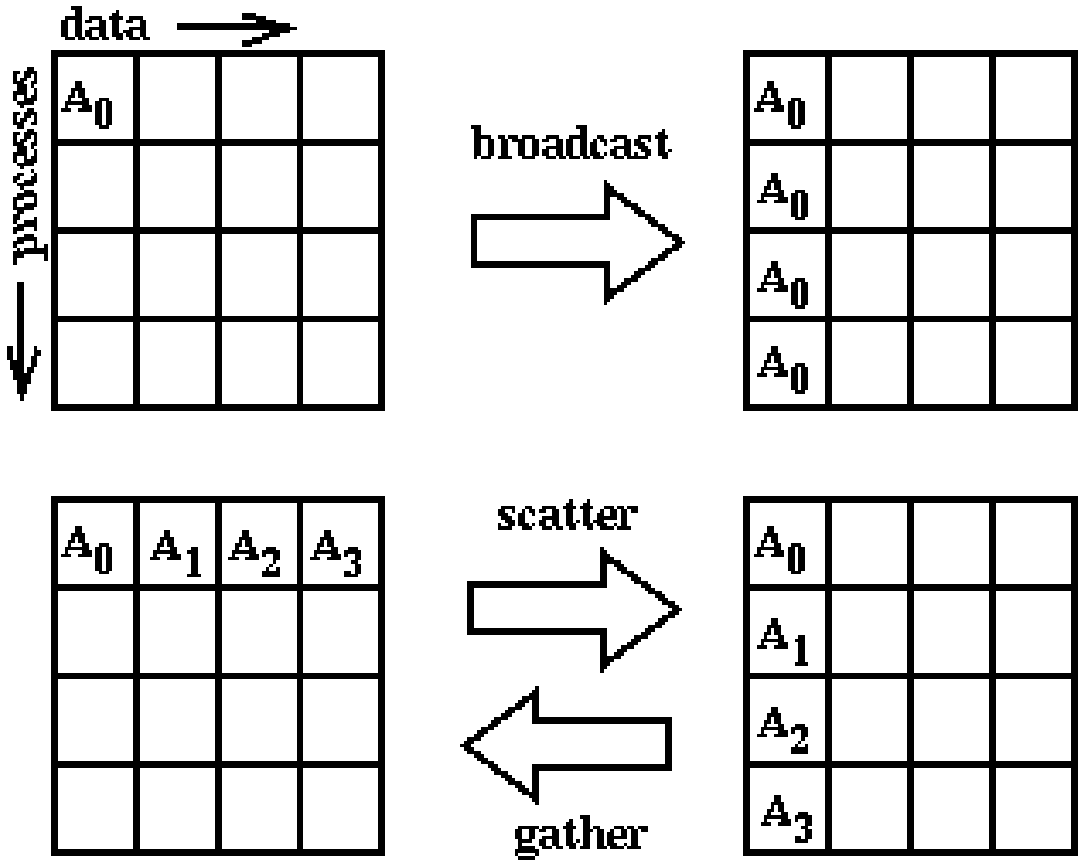
Колективни комуникации



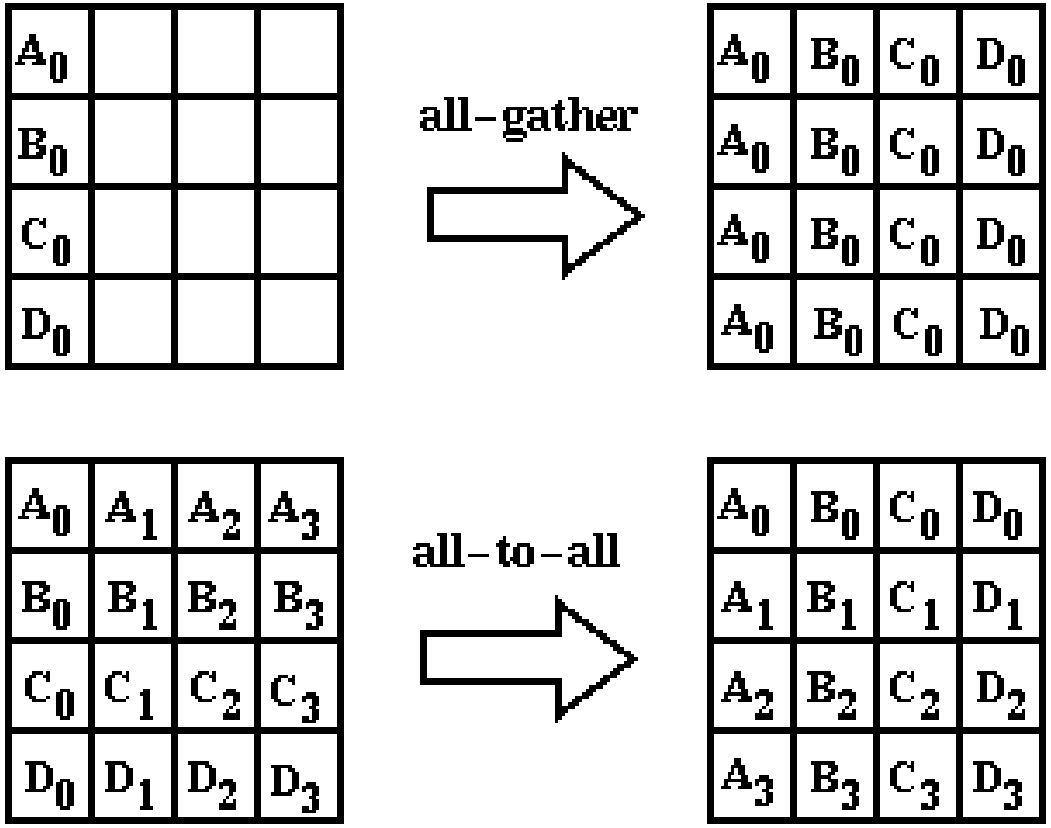
Колективни комуникации

- **Broadcast**
- **Gatter, Scatter**
- **All-gatter**
- **All-to-all**

Колективни комуникации



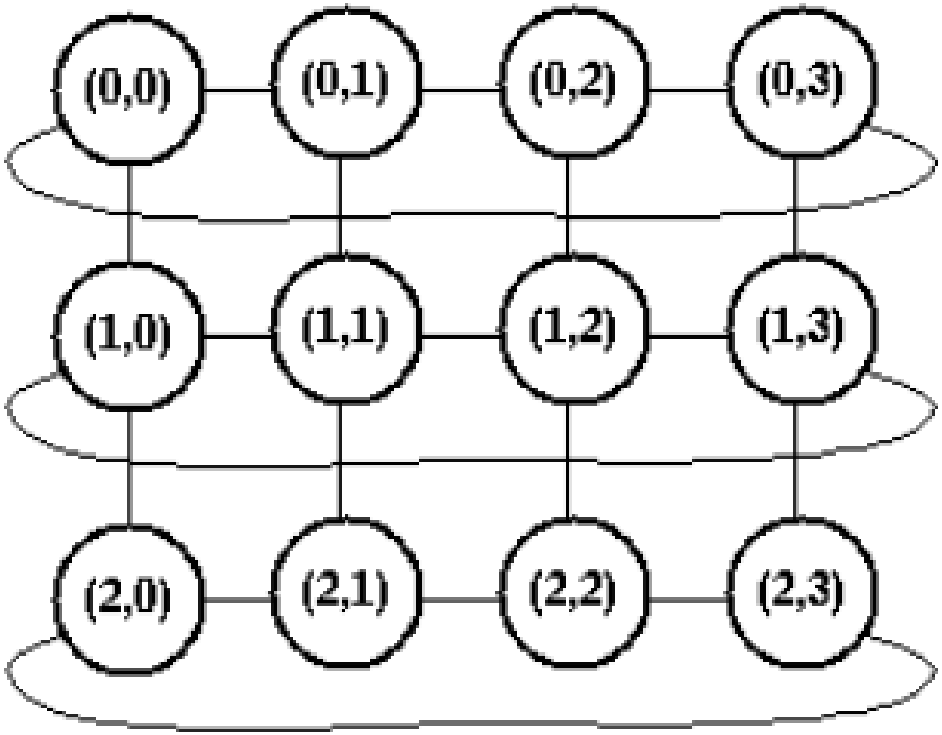
Колективни комуникации



Виртуални топологии

- Механизъм за именуване на процесите в комуникатор с цел по-добра комуникация
- Опростяване на кода
- Оптимизиране свързването на процес към възел

Виртуални топологии



2-D Cartesian Virtual Topology