A quick introduction to MPI (Message Passing Interface)

M1IF - APPD

Oguz Kaya Pierre Pradic

École Normale Supérieure de Lyon, France

ENS DE LYON

Introduction

ENS DE LYON

- Standardized and portable message-passing system.
- Started in the 90's, still used today in research and industry.
- Good theoretical model.
- Good performances on HPC networks (InfiniBand ...).

ENS DE LYON

De facto standard for communications in HPC applications.

ENS DE LYON

APIs:

- C and Fortran APIs.
- C++ API deprecated by MPI-3 (2008).

Environment:

- Many implementations of the standard (mainly OpenMPI and MPICH)
- Compiler (wrappers around gcc)
- Runtime (mpirun)

ENS DE LYON

Compiling:

gcc	→	mpicc
g++	→	mpic++ / mpicxx
gfortran	→	mpifort

Executing:

mpirun -n <nb procs> <executable> <args>
ex : mpirun -n 10 ./a.out

note: mpiexec and orterun are synonyms of mpirun
see `man mpirun` for more details

ENS DE LYON

Context limits

All MPI call must be nested in the MPI context delimited by `MPI_Init` and `MPI_Finalize`.

```
1  #include <mpi.h>
2
3  int main(int argc, char *argv[])
4  {
5      MPI_Init(&argc, &argv);
6
7      // ...
8
9      MPI_Finalize();
10
11     return 0;
12 }
```

ENS DE LYON

Hello World

```
1  #include <stdio.h>
2  #include <mpi.h>
3
4  int main(int argc, char *argv[])
5  {
6      int rank, size;
7
8      MPI_Init(&argc, &argv);
9
10     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
11     MPI_Comm_size(MPI_COMM_WORLD, &size);
12
13     printf("Hello from proc %d / %d\n", rank, size);
14
15     MPI_Finalize();
16
17     return 0;
```

ENS DE LYON

Code:

```
printf("[%d] step 1\n", rank);
MPI_Barrier(MPI_COMM_WORLD);
printf("[%d] step 2\n", rank);
```

Output:

```
[0] step 1
[1] step 1
[2] step 1
[3] step 1
[3] step 2
[0] step 2
[2] step 2
[1] step 2
```

ENS DE LYON

Point-to-point communication

ENS DE LYON

Send and Receive

Sending data:

```
int MPI_Send(const void*  data,
             int          count,
             MPI_Datatype datatype,
             int          destination,
             int          tag,
             MPI_Comm     communicator);
```

Receiving data:

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

ENS DE LYON

Example

```
1  int rank, size;
2  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
3  MPI_Comm_size(MPI_COMM_WORLD, &size);
4
5  int number;
6  switch(rank)
7  {
8    case 0:
9      number = -1;
10     MPI_Send(&number, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
11     break;
12   case 1:
13     MPI_Recv(&number, 1, MPI_INT, 0, 0, MPI_COMM_WORLD,
14              MPI_STATUS_IGNORE);
15     printf("received number: %d\n", number);
16     break;
17 }
```

ENS DE LYON

Asynchronious communications

Sending data:

```
int MPI_Isend(const void*  data,
              int          count,
              MPI_Datatype datatype,
              int          destination,
              int          tag,
              MPI_Comm     communicator,
              MPI_Request* request);
```

Receiving data:

```
int MPI_Irecv(void*        data,
              int          count,
              MPI_Datatype datatype,
              int          source,
              int          tag,
              MPI_Comm     communicator,
              MPI_Request* request);
```

ENS DE LYON

Other functions

- MPI_Probe, MPI_Iprobe
- MPI_Test, MPI_Testany, MPI_Testall
- MPI_Cancel
- MPI_Wtime, MPI_Wtick

ENS DE LYON

Simple datatypes

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

ENS DE LYON

Composed datatypes

Composed structure:

- Structures;
- Array.

Possibilities are almost limitless ...

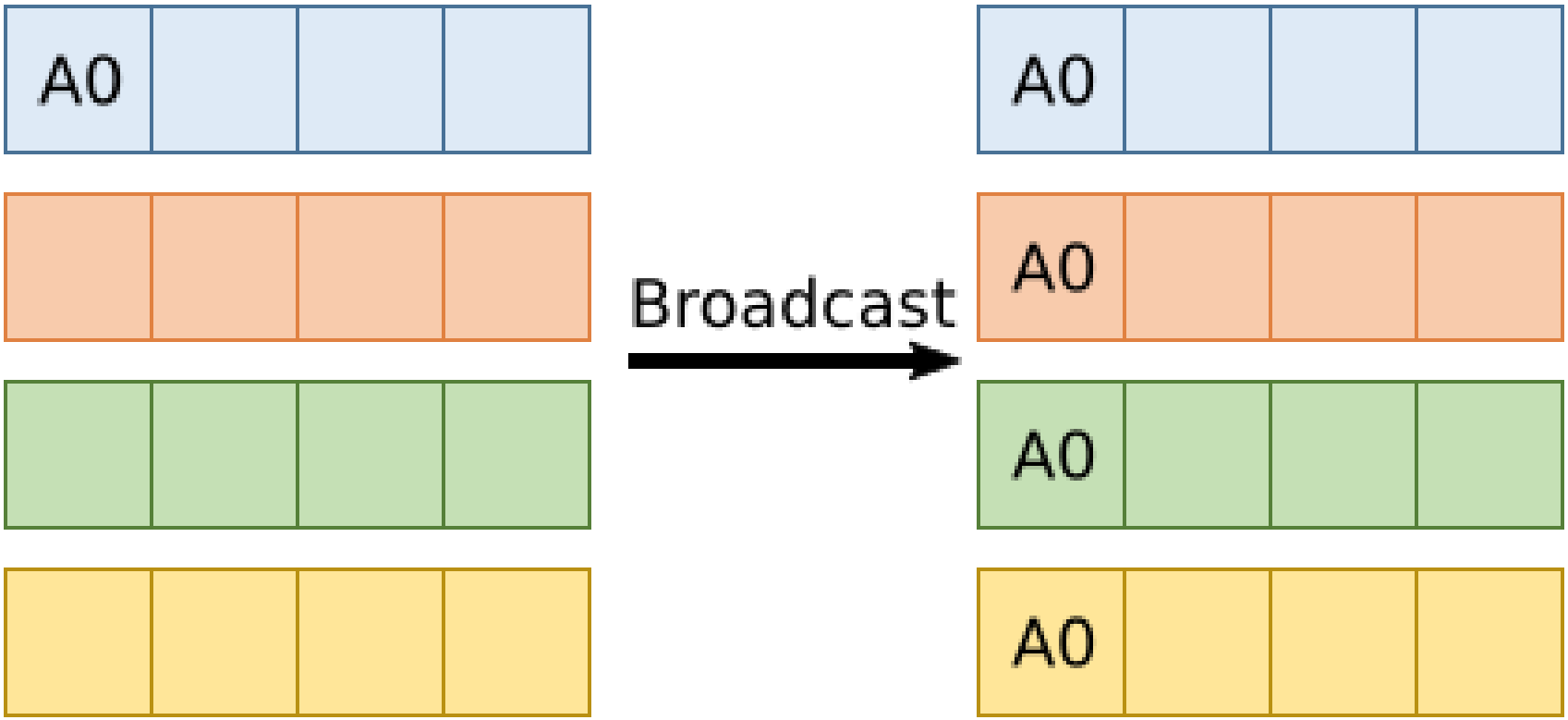
... but sometimes difficult to setup.

ENS DE LYON

Collective communications

ENS DE LYON

Broadcast

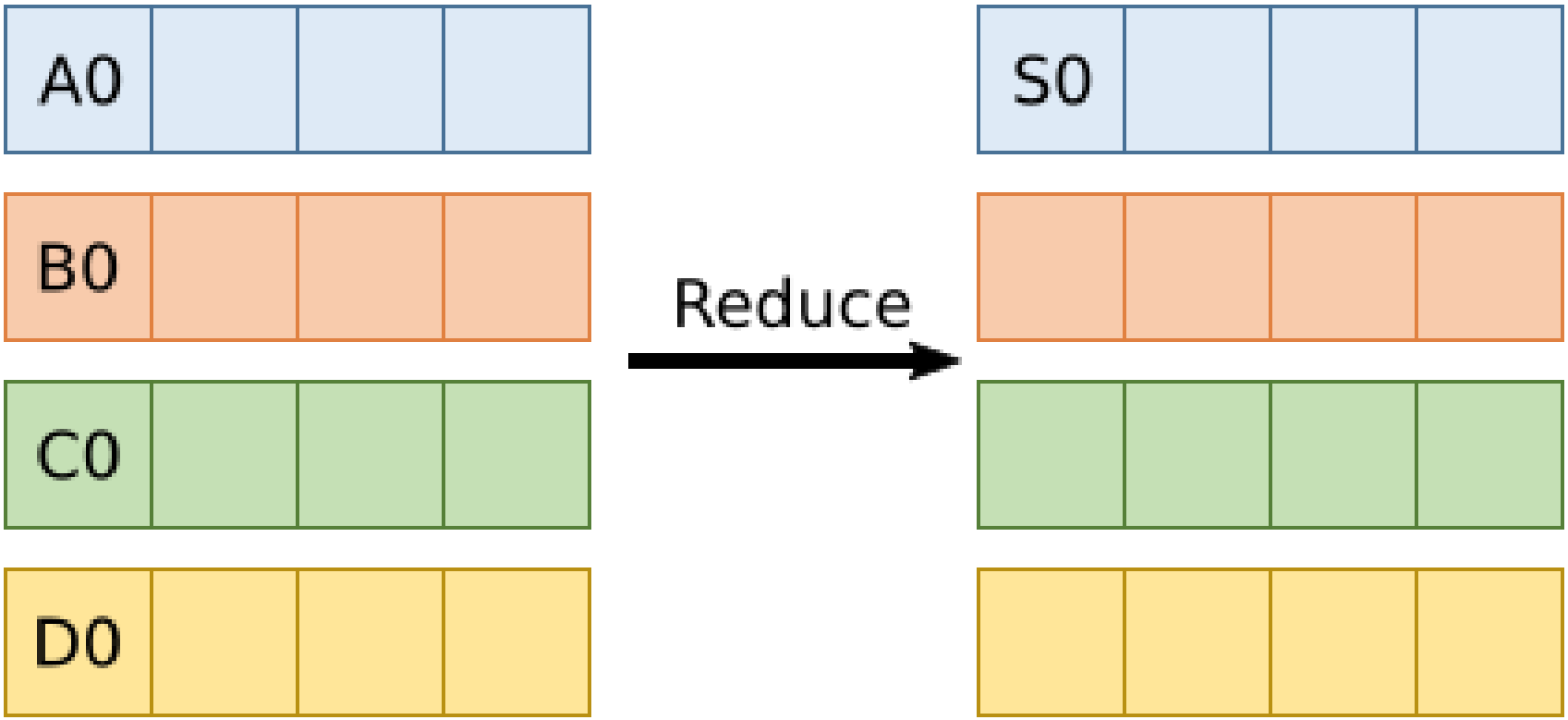


ENS DE LYON

Broadcast

```
int MPI_Bcast(void*         data,
              int           count,
              MPI_Datatype  datatype,
              int           root,
              MPI_Comm      communicator);
```

Reduce



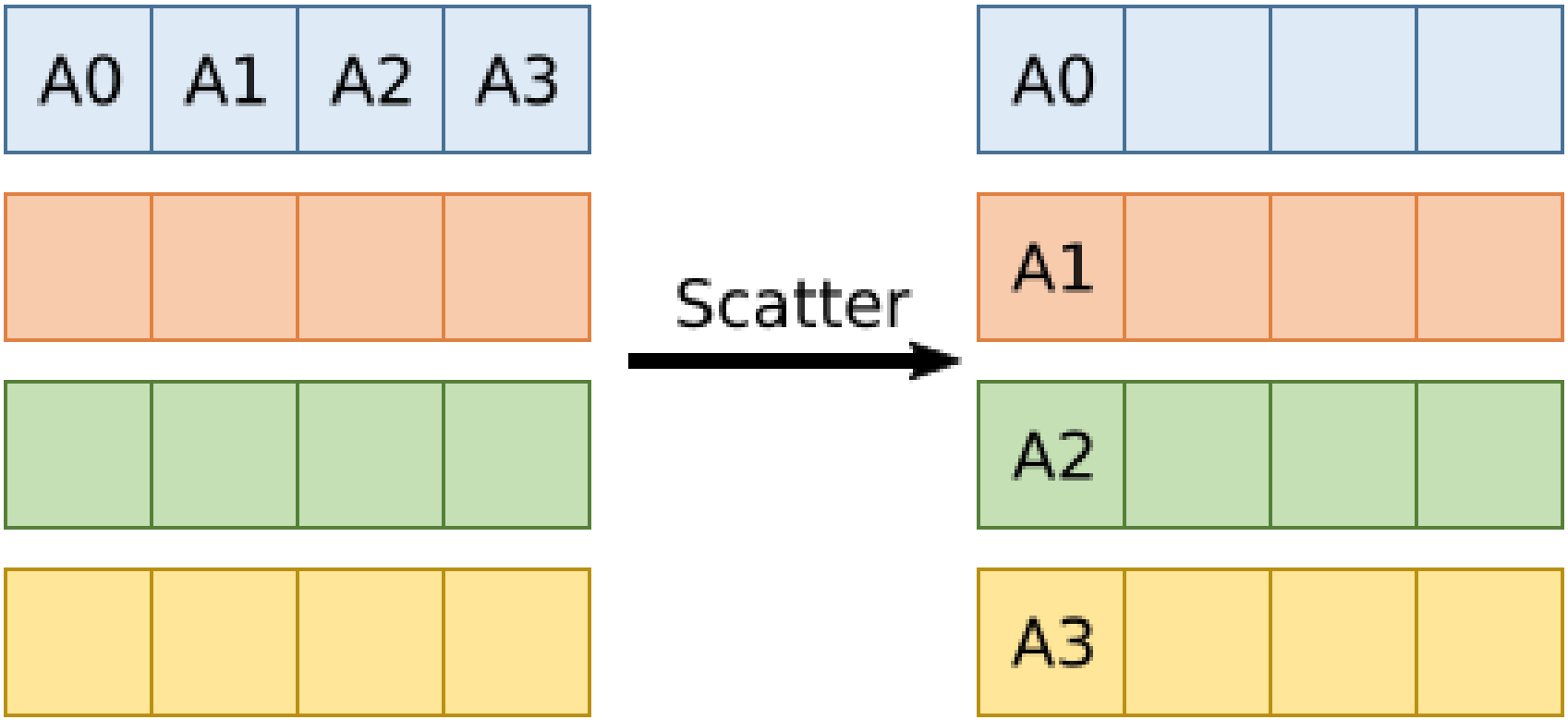
ENS DE LYON

Reduce

```
int MPI_Reduce(const void*   sendbuf,
               void*         recvbuf,
               int           count,
               MPI_Datatype  datatype,
               MPI_Op        operator,
               int           root,
               MPI_Comm      communicator);
```

ENS DE LYON

Scatter



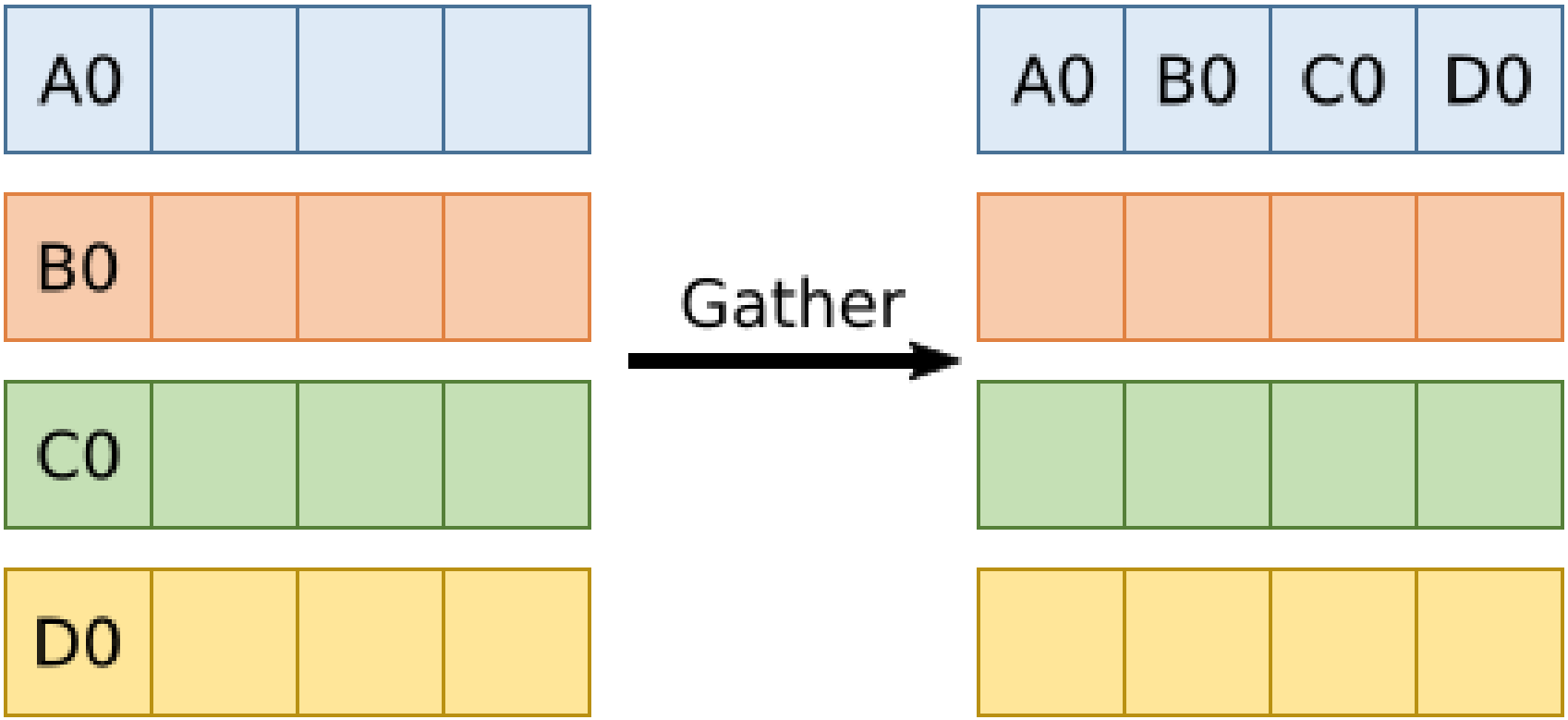
ENS DE LYON

Scatter

```
int MPI_Scatter(const void*  sendbuf,
                int          sendcount,
                MPI_Datatype sendtype,
                void*        recvbuf,
                int          recvcount,
                MPI_Datatype recvtype,
                int          root,
                MPI_Comm     communicator);
```

ENS DE LYON

Gather



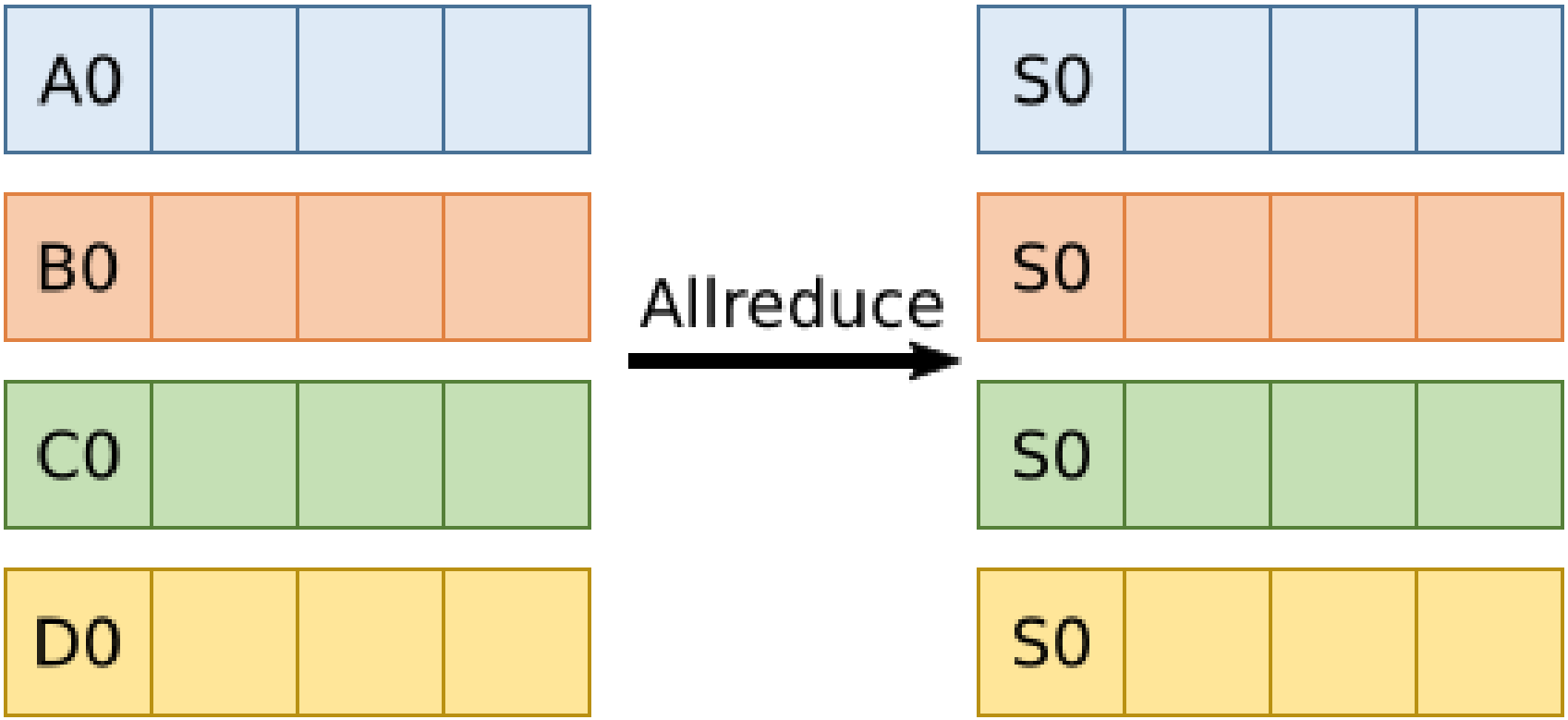
ENS DE LYON

Gather

```
int MPI_Gather(const void*   sendbuf,
               int           sendcount,
               MPI_Datatype  sendtype,
               void*         recvbuf,
               int           recvcount,
               MPI_Datatype  recvtype,
               int           root,
               MPI_Comm      communicator);
```

ENS DE LYON

Allreduce



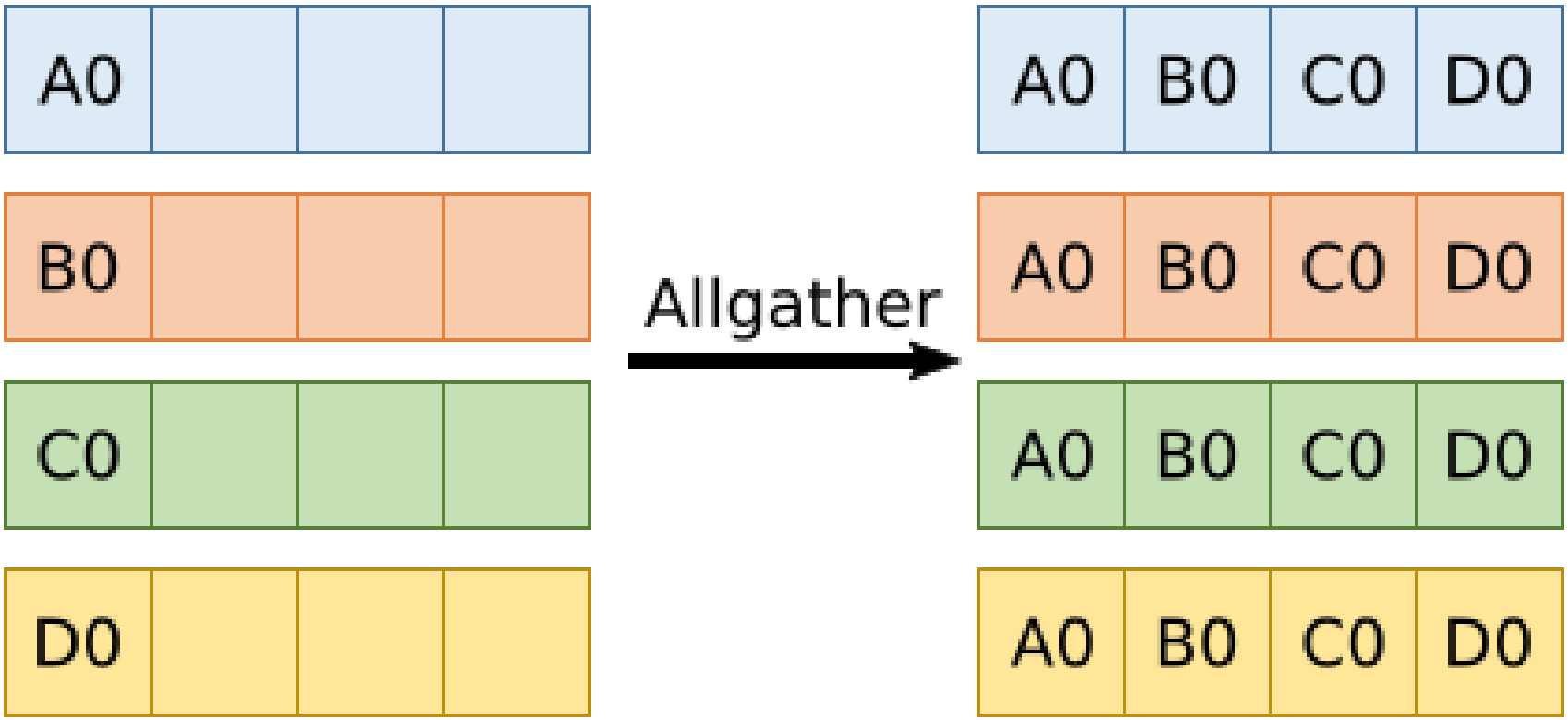
ENS DE LYON

AllReduce

```
int MPI_Allreduce(const void*   sendbuf,
                  void*         recvbuf,
                  int           count,
                  MPI_Datatype  datatype,
                  MPI_Op        operator,
                  MPI_Comm      communicator);
```

ENS DE LYON

Allgather



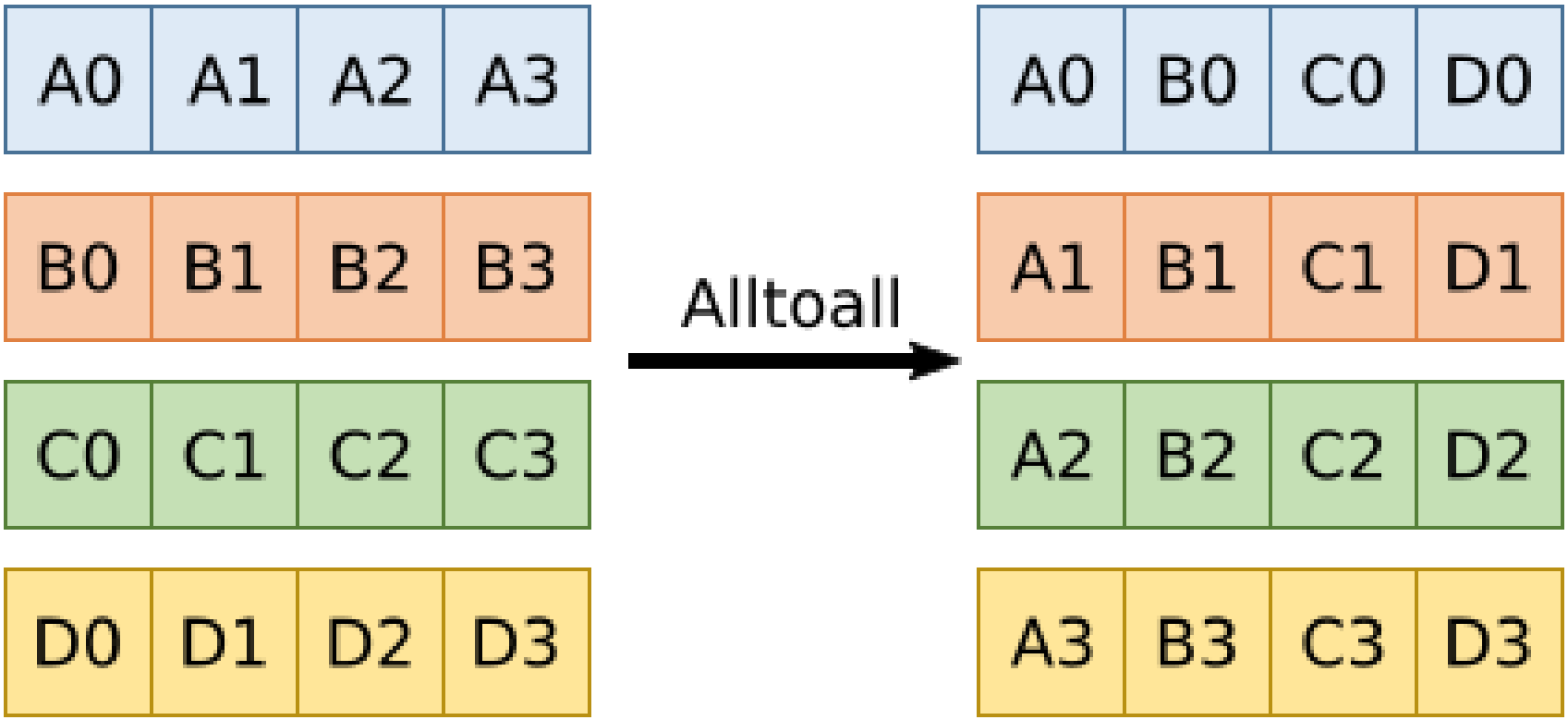
ENS DE LYON

AllGather

```
int MPI_Allgather(const void*   sendbuf,
                  int           sendcount,
                  MPI_Datatype  sendtype,
                  void*         recvbuf,
                  int           recvcount,
                  MPI_Datatype  recvtype,
                  MPI_Comm      communicator);
```

ENS DE LYON

Alltoall



ENS DE LYON

Alltoall

```
int MPI_Alltoall(const void*   sendbuf,
                 int           sendcount,
                 MPI_Datatype  sendtype,
                 void*         recvbuf,
                 int           recvcount,
                 MPI_Datatype  recvtype,
                 MPI_Comm      communicator);
```

ENS DE LYON

Custom communicators

ENS DE LYON

`MPI_COMM_WORLD` can be split into smaller, more appropriate communicators.

```
int MPI_Comm_split(MPI_Comm  communicator,
                   int       color,
                   int       key,
                   MPI_Comm* newcommunicator);
```

ENS DE LYON

Example

```
1  ...
2  int rank, size;
3  MPI_Init(&argc, &argv);
4  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
5  MPI_Comm_size(MPI_COMM_WORLD, &size);
6
7  int        hrank, vrank;
8  int        hsize, vsize;
9  MPI_Comm hcomm, vcomm;
10 MPI_Comm_split(MPI_COMM_WORLD, rank%p, rank, &vcomm);
11 MPI_Comm_split(MPI_COMM_WORLD, rank/p, rank, &hcomm);
12 MPI_Comm_rank(hcomm, &hrank);
13 MPI_Comm_size(hcomm, &hsize);
14 MPI_Comm_rank(vcomm, &vrank);
15 MPI_Comm_size(vcomm, &vsize);
16 ...
```

ENS DE LYON