MPI Collectives

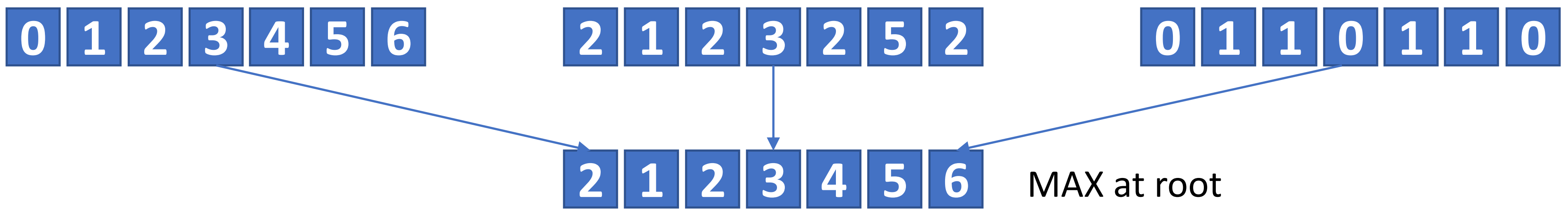
Jan 22, 2019

Communication Cost Model

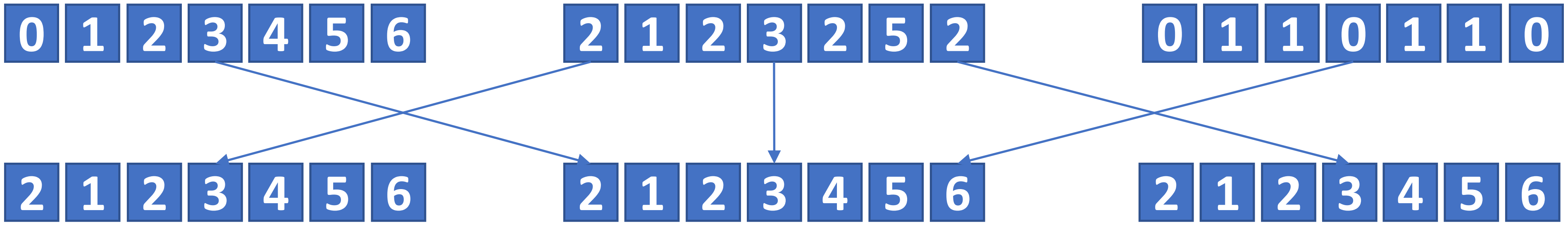
- Message transfer time is modeled as $l+n/b$, where l is the latency (or startup time) per message, and $1/b$ is the transfer time per byte, and n the message size in bytes
- All processes can send and receive one message at the same time

Reduce

- MPI_Reduce (inbuf, outbuf, count, datatype, op, root, comm)
- Combines element in inbuf of each process
- Combined value in outbuf of root
- op: MIN, MAX, SUM, PROD, …



Allreduce

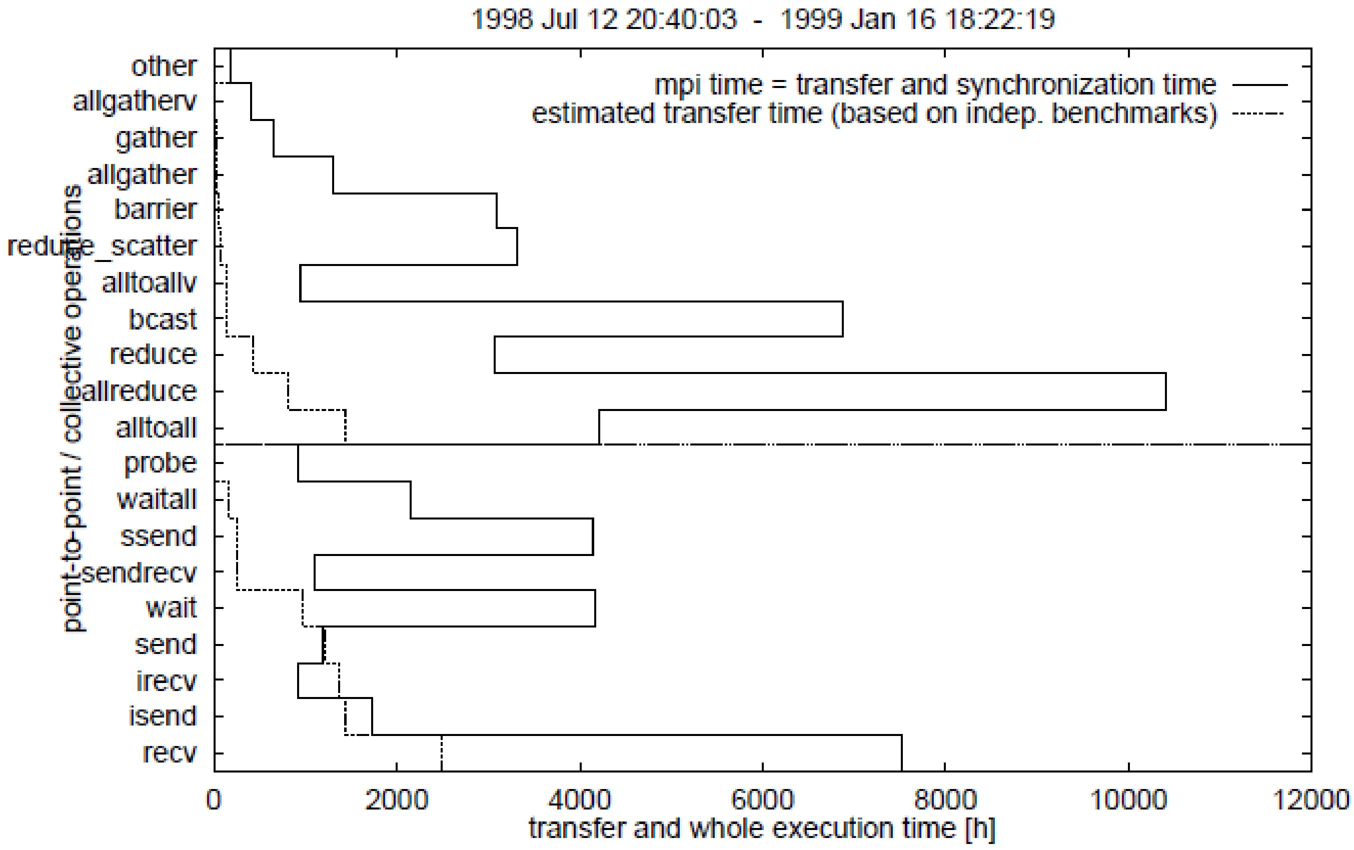


MPI_Allreduce (inbuf, outbuf, count, datatype, op, comm)

Equivalent collective

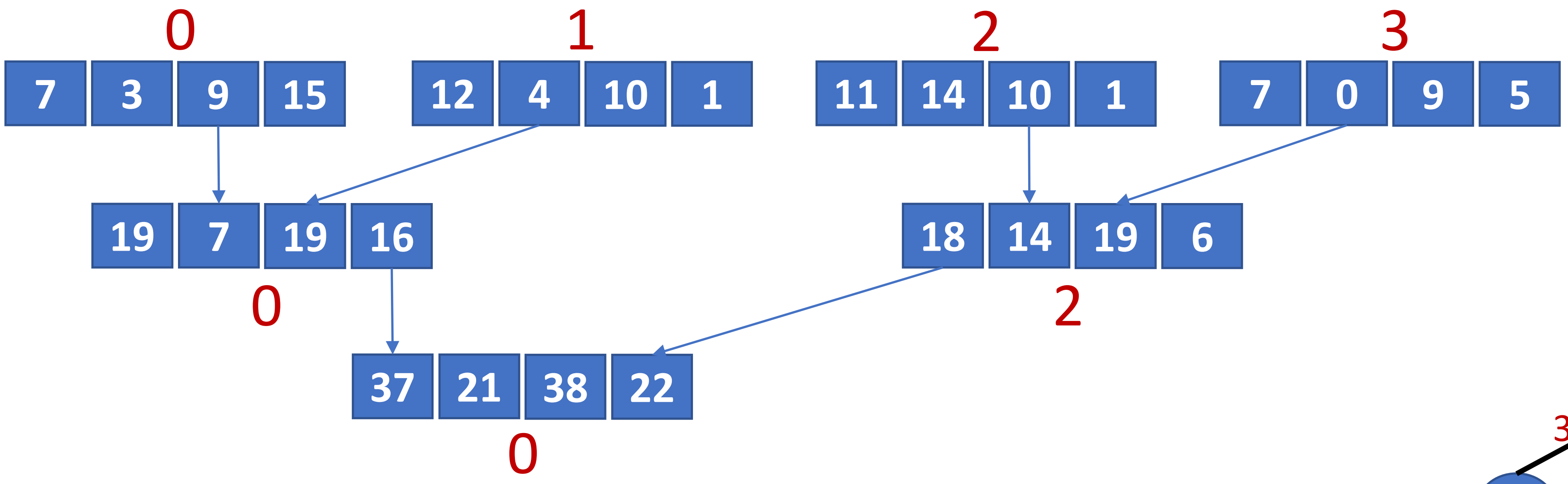
- Reduce+Bcast

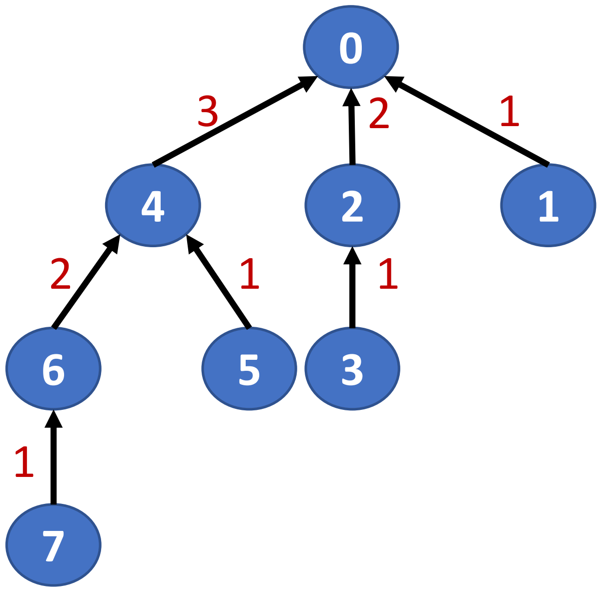
MPI Profiles



[Source: CUG 2000]

Reduce – Recursive doubling

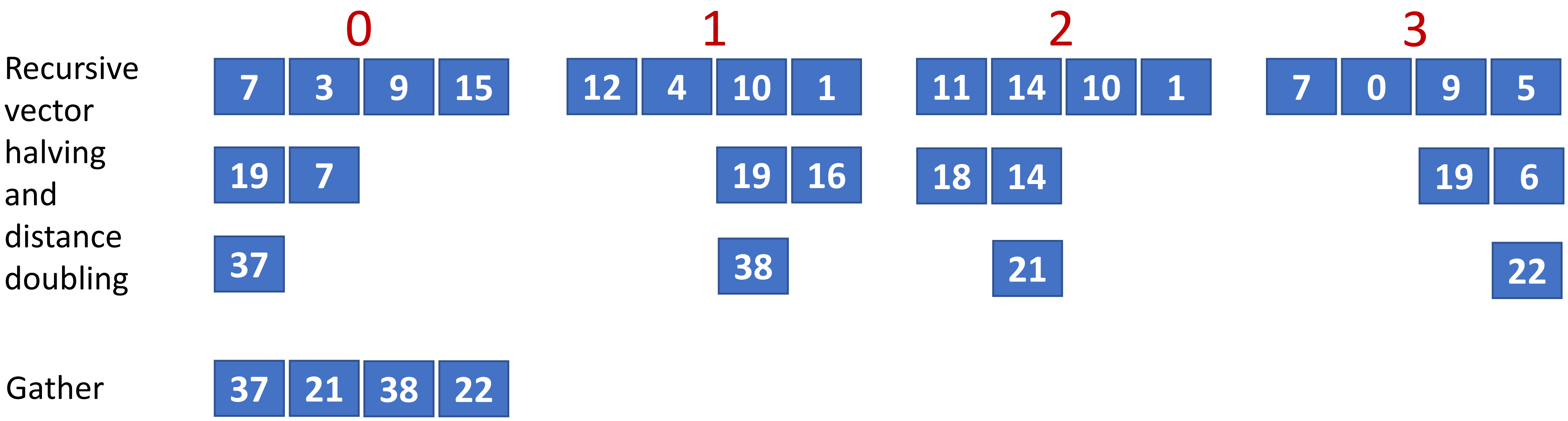




Time: log p (l + n*(1/b) + n*c)
l = latency, b = bandwidth
c = compute time per byte

Used for short messages

Reduce – Rabenseifner’s Algorithm



Time:

log p * l + (p-1)/p*(n/b) + (p-1)/p*n*c (reduce-scatter) +

log p * l + (p-1)/p*(n/b) (gather using binomial)

n = data size

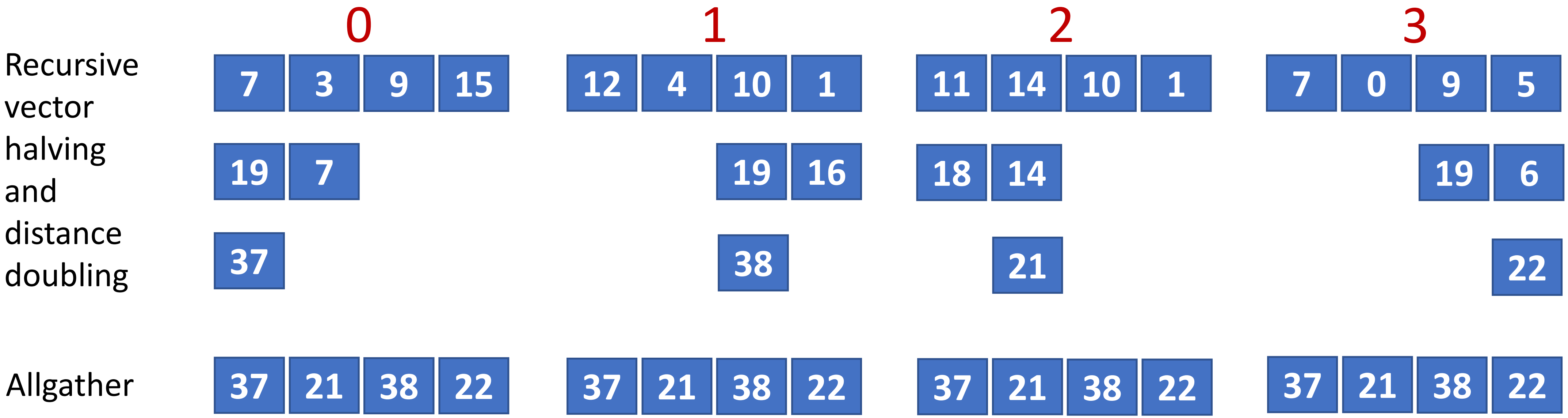
l = latency

p = #processes

b = bandwidth

c = compute time

Allreduce – Rabenseifner's Algorithm



Time:

log p * l + (p-1)/p*(n/b) + (p-1)/p*n*c (reduce-scatter) +

log p * l + (p-1)/p*(n/b) (allgather using recursive vector doubling and distance halving)

n = data size   l = latency

p = #processes   b = bandwidth

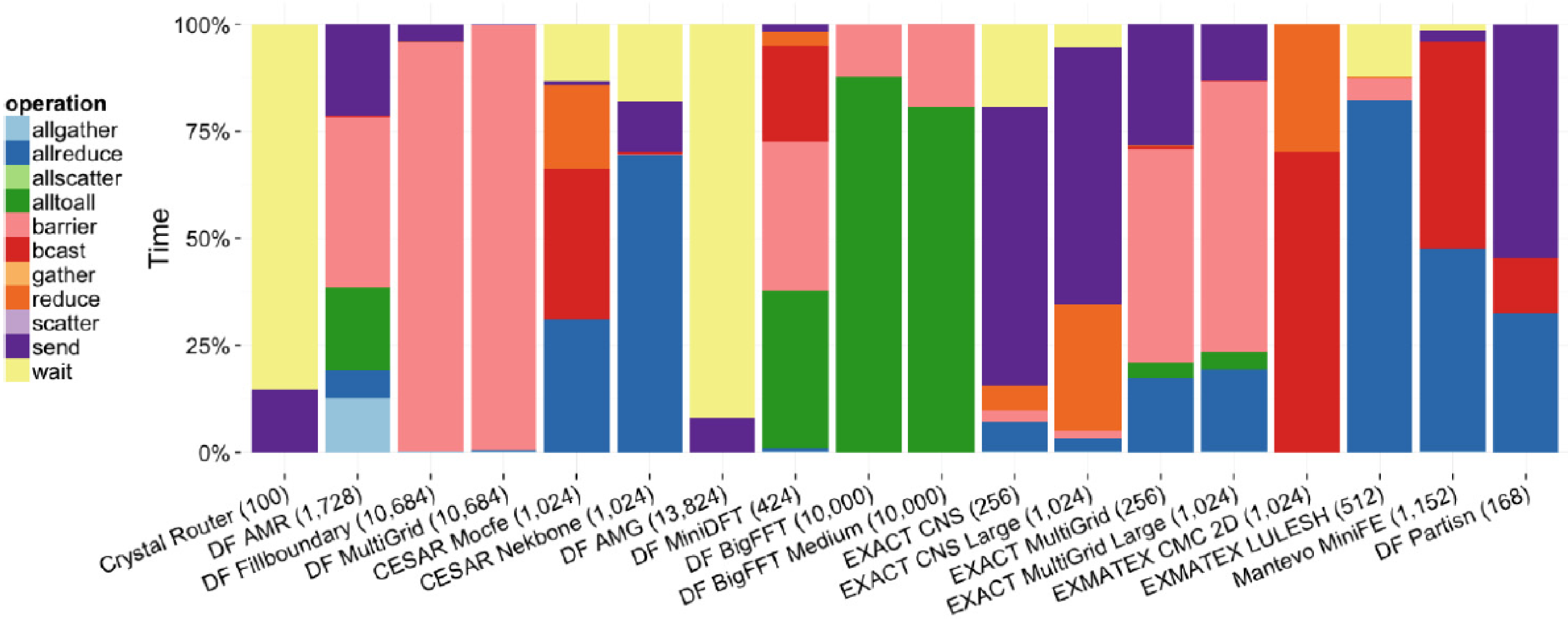
c = compute time

Allreduce Algorithms

- Reduce followed by broadcast
  - Time: [log p (l + n*(1/b) + n*c)] + [log p (l + n*(1/b)]

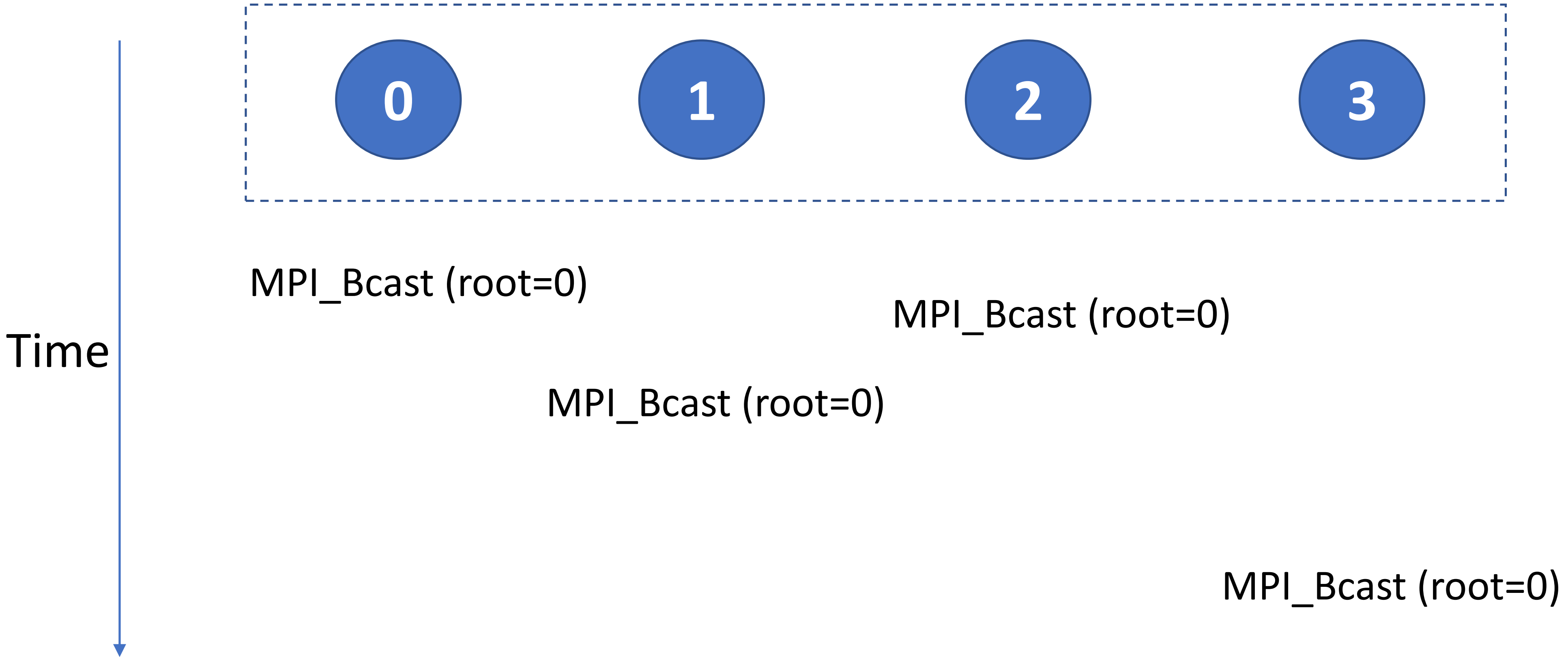
- Reduce-scatter followed by allgather (recursive doubling)
  - Time: log p * l + 2(p-1)/p * (n/b) + (p-1)/p * n * c

MPI Communication Times Survey



Source: Klenk and Froning

Recap


0
1
2
3
Time
MPI_Bcast (root=0)
MPI_Bcast (root=0)
MPI_Bcast (root=0)
MPI_Bcast (root=0)

MPI Examples

```
// … initialization tasks

int sendarr[10], recvarr[10];
if (myrank == 0) MPI_Send (sendarr, 1, MPI_INT, 1, 99, MPI_COMM_WORLD);
if (myrank == 1) MPI_Recv (recvarr, 10, MPI_INT, 0, 99, MPI_COMM_WORLD, status);
printf ("%d\n", myrank);
```

Output for n=2?

0

1

MPI Examples

```
If (myrank == 0) {
  MPI_Bcast (buf1, count, type, 0, comm);
  MPI_Send (buf2, count, type, 1, tag, comm);
}
If (myrank == 1) {
  MPI_Recv (buf2, count, type, 0, tag, comm, status);
  MPI_Bcast (buf1, count, type, 0, comm);
}
```

Will it run?

Incorrect code, may succeed, may deadlock

MPI_Bcast Examples

```
// … initialization tasks

int color=irand();
if (myrank != 1) MPI_Bcast (&color, 1, MPI_INT, 1, MPI_COMM_WORLD);
if (myrank == 1) MPI_Bcast (&color, 1, MPI_INT, 1, MPI_COMM_WORLD);
printf ("%d: %d\n", myrank, color);
```

Output for n=4?

MPI_Bcast Examples

```
// … initialization tasks

int arr[100];
if (myrank != 2) MPI_Bcast (arr, 10, MPI_INT, 1, MPI_COMM_WORLD);
if (myrank == 2) MPI_Bcast (arr, 100, MPI_INT, 1, MPI_COMM_WORLD);
printf (“%d\n”, myrank);
```

Output for n=4?

“Message sizes do not match across processes”

MPI_Bcast Examples

```
// … initialization tasks

int arr[1000];
if (myrank != 2) MPI_Bcast (arr, 100, MPI_INT, 1, MPI_COMM_WORLD);
if (myrank == 2) MPI_Bcast (arr, 10, MPI_INT, 1, MPI_COMM_WORLD);
printf (“%d\n”, myrank);
```

Output for n=4?

“Message truncated error”

MPI_Bcast Examples

```
// … initialization tasks

int color=irand();
if (myrank != 0) MPI_Bcast (&color, 1, MPI_INT, 0, MPI_COMM_WORLD);
printf (“%d: %d\n”, myrank, color);
```

Output for n=4?

0 is not blocked, everyone else is.

MPI_Bcast Examples

```
// … initialization tasks

int color=irand();
if (myrank != 3) MPI_Bcast (&color, 1, MPI_INT, 0, MPI_COMM_WORLD);
printf (“%d: %d\n”, myrank, color);
```

Output for n=4?

May succeed but not safe

MPI_Bcast Examples

```
// … initialization tasks

int color=irand();
if (myrank != 2) MPI_Bcast (&color, 1, MPI_INT, 0, MPI_COMM_WORLD);
MPI_Bcast (&color, 1, MPI_INT, 0, MPI_COMM_WORLD);
printf (“%d: %d\n”, myrank, color);
```

Output for n=4?

Most likely will block, not safe

MPI_Bcast Examples

```
// … initialization tasks

int color=irand();
if (myrank != 3) MPI_Bcast (&color, 1, MPI_INT, 0, MPI_COMM_WORLD);
MPI_Bcast (&color, 1, MPI_INT, 0, MPI_COMM_WORLD);
printf ("%d: %d\n", myrank, color);
```

Output for n=4?

May succeed, not safe

MPI_Bcast Examples

```
// … initialization tasks

int color=irand();
if (myrank != 3) MPI_Bcast (&color, 100000, MPI_INT, 0, MPI_COMM_WORLD);
MPI_Bcast (&color, 100000, MPI_INT, 0, MPI_COMM_WORLD);
printf (“%d: %d\n”, myrank, color);
```

Output for n=4?

Most likely will block, not safe

Non-blocking Collectives

- Introduced in MPI-3
- Similar algorithms
- Benefit of non-blocking point-to-point
- Overlap communication and computation
- Reduce synchronization
- Collective on overlapping communicators
- How do we ensure completion?
    - MPI_Wait (request, status)

Non-blocking Collectives

- MPI_Ibcast (buffer, count, datatype, root, comm, request)
- MPI_Igather (sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, root, comm, request)
- MPI_Igatherv (sendbuf, sendcount, sendtype, recvbuf, recvcounts, displs, recvtype, root, comm, request)
- MPI_Ialltoall (sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, comm, request)
- …

MPI Example

```
If (myrank == 0) {
  MPI_Ibcast (&buf1, 1, MPI_INT, 0, comm, req1);
  MPI_Wait(&req1, MPI_STATUS_IGNORE);
  MPI_Send(buf, count, MPI_INT, 1, tag, comm);
}
If (myrank == 1) {
  MPI_Ibcast (&buf1, 1, MPI_INT, 0, comm, req1);
  MPI_Recv(buf, count, MPI_INT, 0, tag, comm, MPI_STATUS_IGNORE);
  MPI_Wait(&req1, MPI_STATUS_IGNORE);
}
```

Output for n=2?

Valid code

MPI Example

```
If (myrank == 0) {
  MPI_Ibcast (&buf1, 1, MPI_INT, 0, comm, req1[0]);
  MPI_Ibcast (&buf2, 1, MPI_INT, 0, comm, req1[0]);
}
If (myrank == 1) {
  MPI_Ibcast (&buf2, 1, MPI_INT, 0, comm, req1[0]);
  MPI_Ibcast (&buf1, 1, MPI_INT, 0, comm, req1[0]);
}
MPI_Waitall(2, req1, MPI_STATUSES_IGNORE);
```

Correct for n=2?

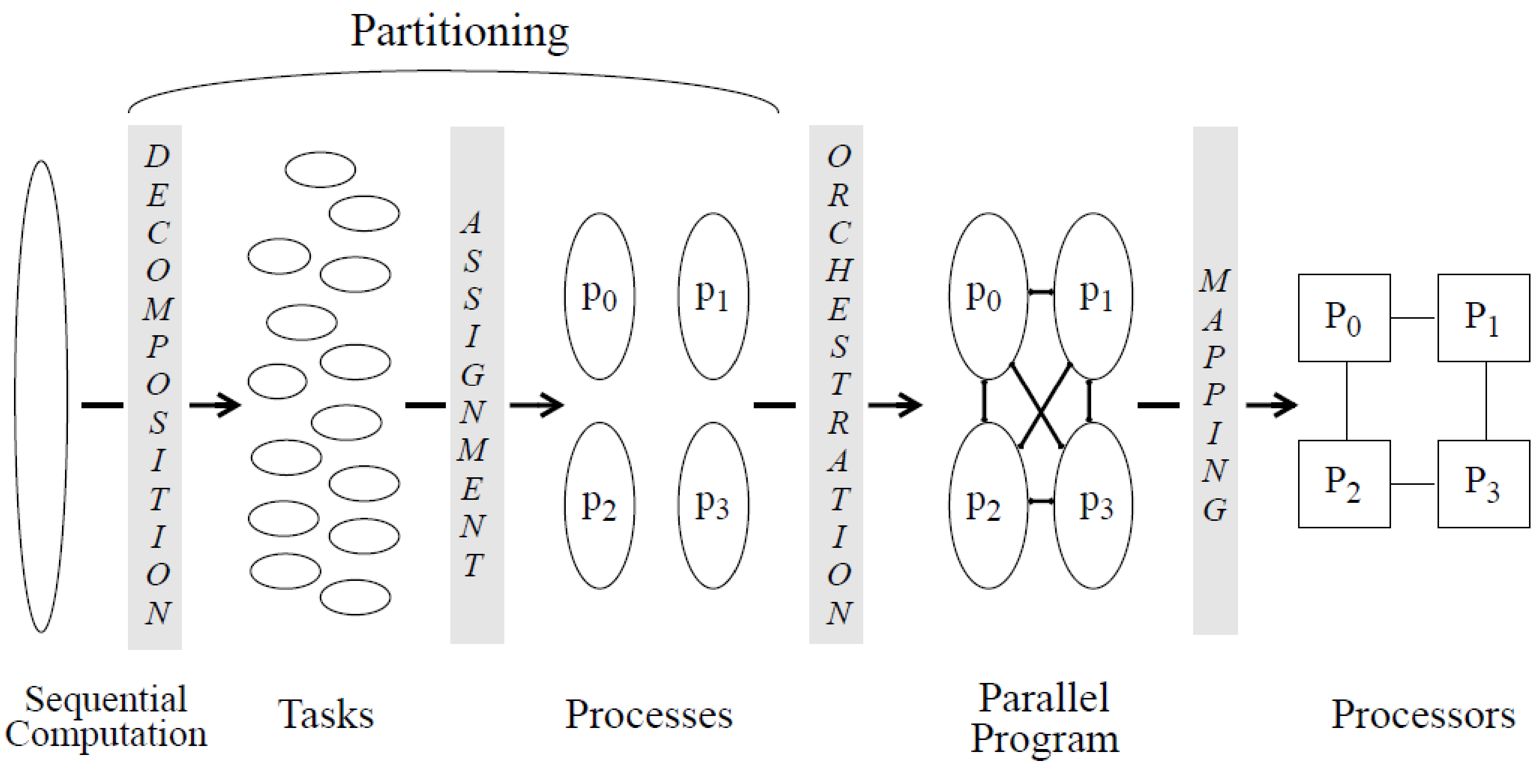
Valid code

Parallelization

Parallelization Steps

1. *Decomposition* of computation into tasks
   - Identifying portions of the work that can be performed concurrently.
2. *Assignment* of tasks to processes
   - Assigning concurrent pieces of work onto multiple processes running in parallel
3. *Orchestration* of data access, communication and synchronization among processes
   - Distributing the data associated with the program
   - Managing access to data shared by multiple processes
   - Synchronizing at various stages of the parallel program execution
4. *Mapping* of processes to processors
   - Placement of processes in the physical processor topology

Illustration of Parallelization Steps



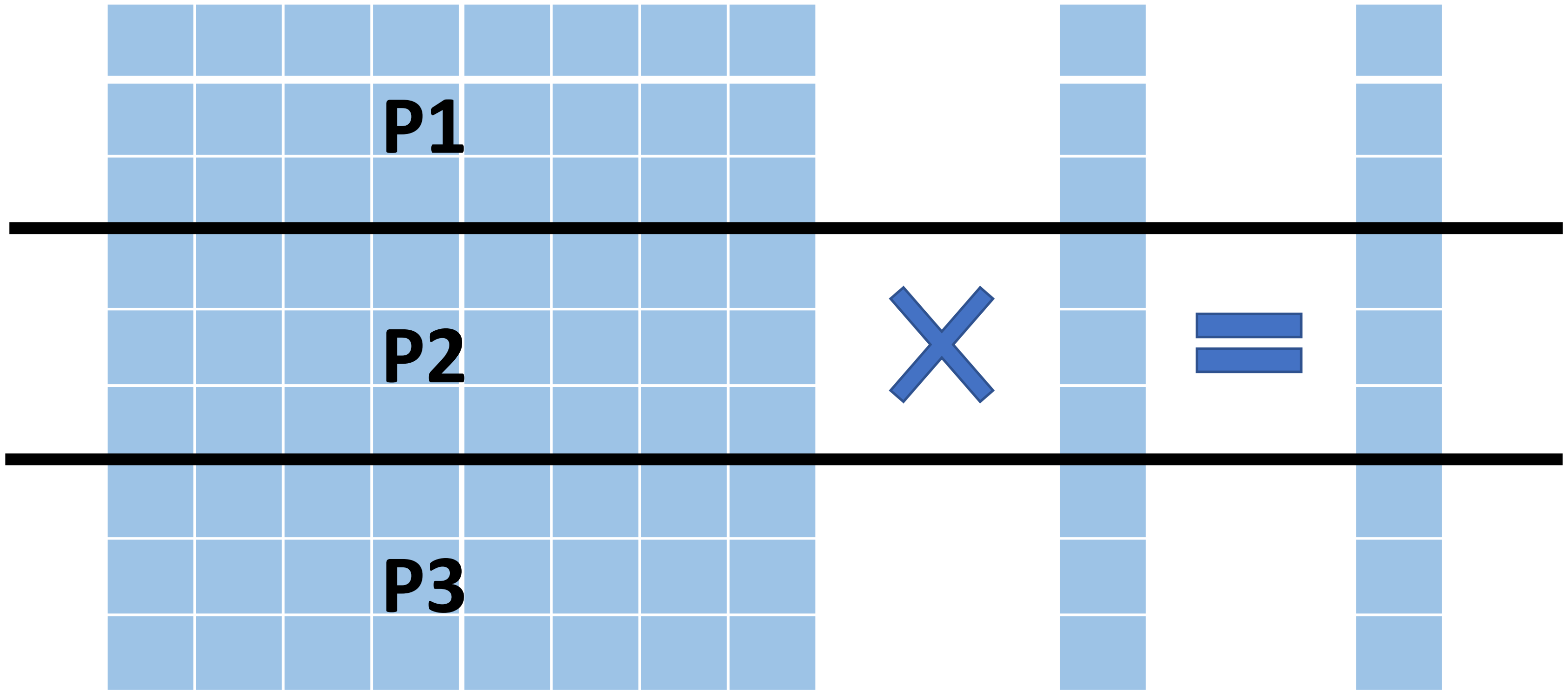
Q: What if number of tasks != allocated processors?

Source: Culler et al.

Desirable properties

- Minimize execution time by minimizing
  - Inter-process communications
  - Load imbalance
  - Synchronization
  - Idling

Parallelization – Matrix Vector Multiplication



P = 3

Decomposition

Assignment

Orchestration

Mapping

Virtual Topology

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31

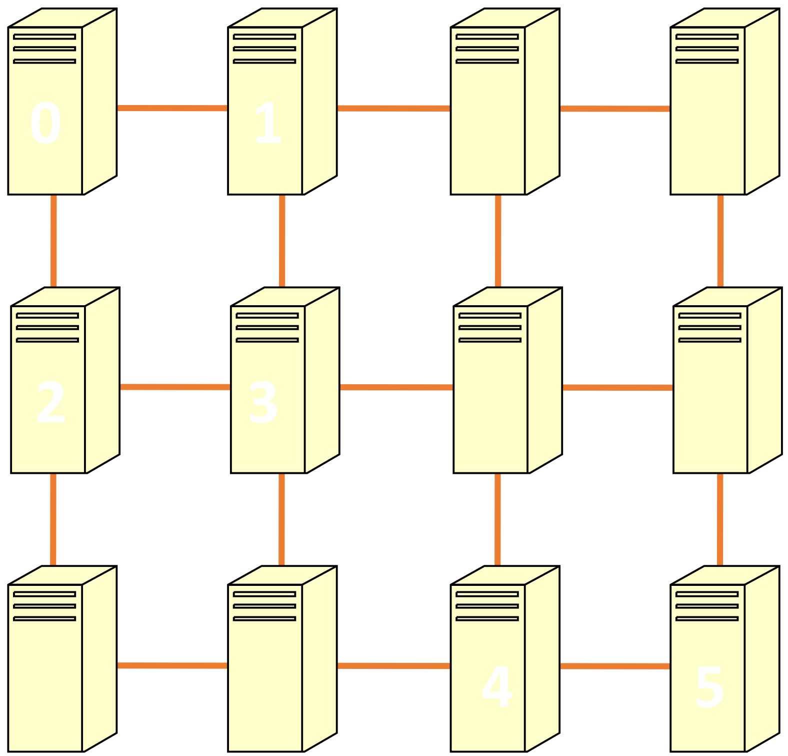
8 x 4 2D virtual process topology

- Communication pattern of MPI processes
- Graphical representation of communications
  - Nearest neighbor in a mesh
  - All-to-all
  - …
- Convenient way to represent communications

Note: Virtual topology set up before execution

Q: One feature of virtual topology with respect to network topology?

Physical Topology



- Connections between allocated cores
- Default placement of ranks based on node IDs
- Mapping: Placement of ranks onto cores
- Topology-aware mapping: Mapping that minimizes all communication times taking into account the physical topology