KOÇ UNIVERSITY

Modeling and Optimizing Communication

Didem Unat

COMP 429/529 Parallel Programming

Communication Cost

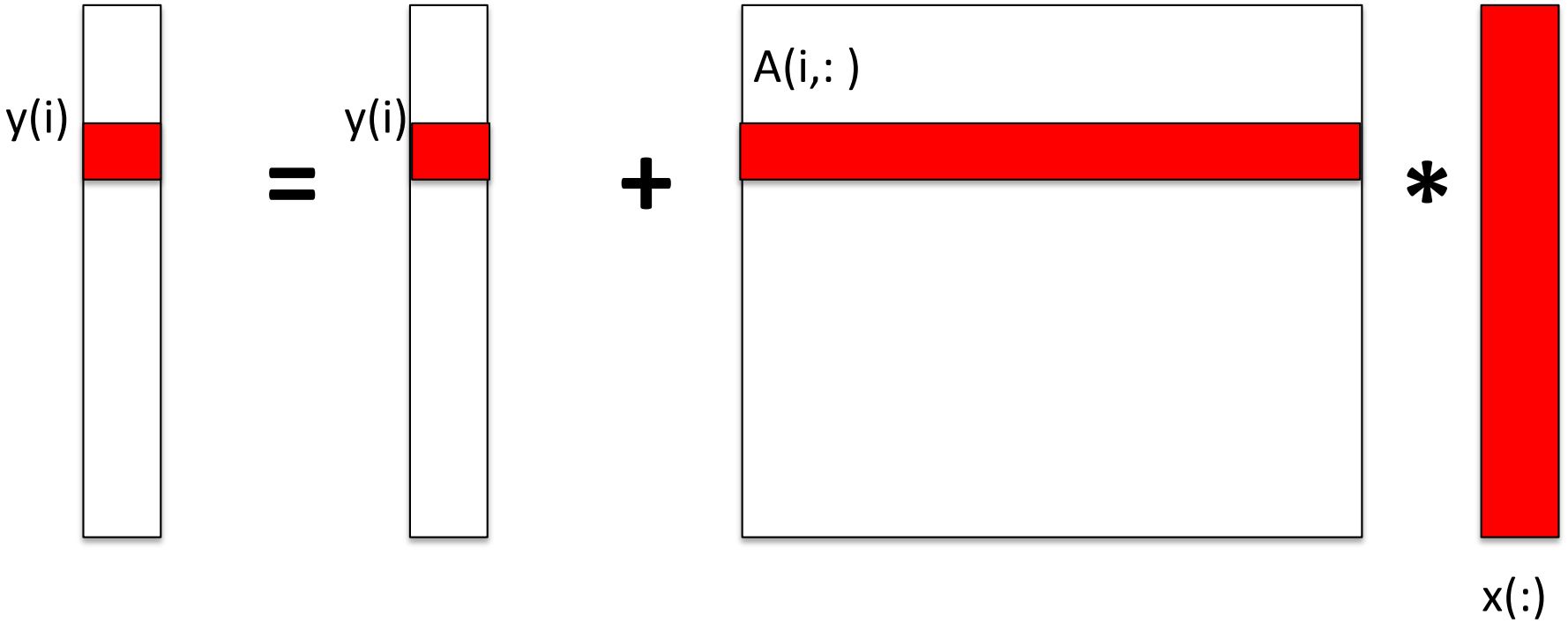
- Communication performance can be a major factor in determining application performance
- **Message passing time = α + N/β**

α = message startup time and overhead (latency)
β = network bandwidth (bytes/sec)
N = message size

- Short messages: startup term dominates
- Long messages: bandwidth term dominates

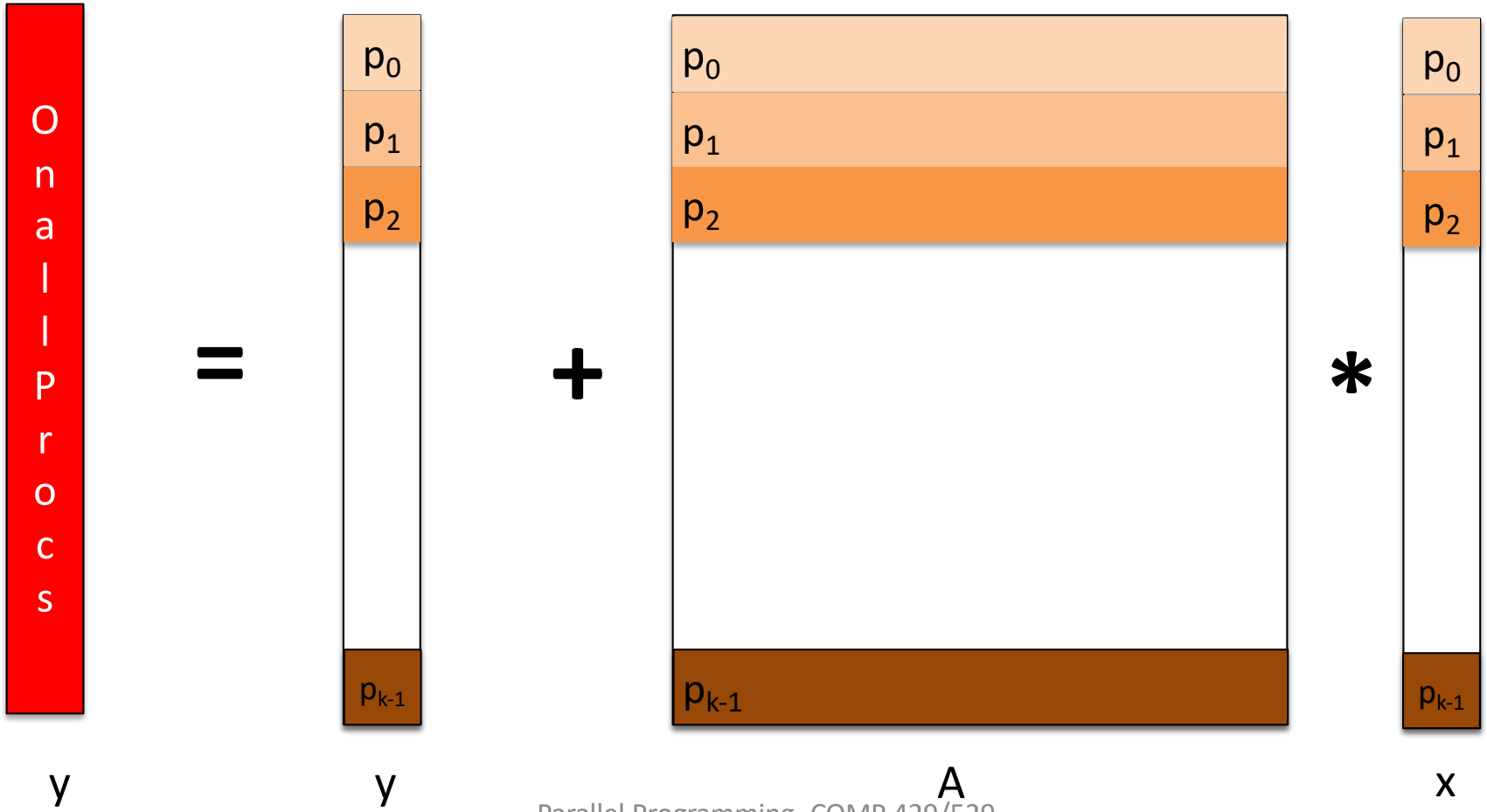
An MPI Example: Matrix Vector Multiply

```
{implements y = y + A*x}
for i = 1:n
    for j = 1:n
       y(i) = y(i) + A(i,j)*x(j)
```



An MPI Example: Matrix Vector Multiply

Assume we have *P* processors with the following initial data layout
Ex. P_0 has a number of rows of A, x and y.

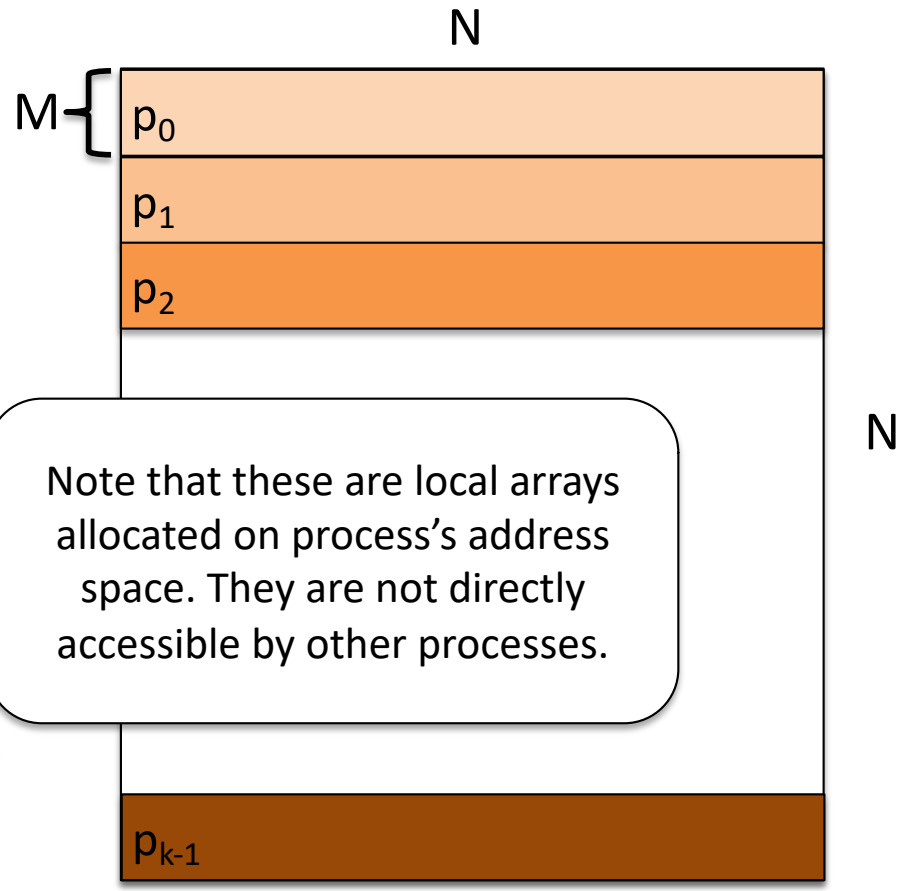


An MPI Example: Matrix Vector Multiply

- A high-level parallel algorithm
  - Collect the entire x vector on all processors
  - Perform local matrix-vector multiplications
    - y(i) = y(i) + A(i, :) * x (:)
  - Collect partial output vectors y on all processors

MatVec Multiply

```
#include <mpi.h>
#include <stdio.h>
#include <sys/time.h>

#define N 3000

int main(int argc, char *argv[])
{
  int P, i, myrank, M;
  int i, j;
  double t_start, t_end;
  double **my_A, *my_x, *my_y, *x, *y;

  /* Initializations */
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &P);
  MPI_Comm_rank(MPI_COMM_WORLD, &myrank);

  M = N / P; // Assuming N is a multiple of P

  my_x = (double*) malloc(M * sizeof(double));
  my_y = (double*) malloc(M * sizeof(double));
  x = (double*) malloc(N * sizeof(double));
  y = (double*) malloc(N * sizeof(double));

  A = (double**) malloc(M * sizeof(double*));
  for (i = 0; i < M; ++i)
    A[i] = (double*) malloc(N * sizeof(double));

  //initialize local arrays (e.g. from a file)
  ...
```

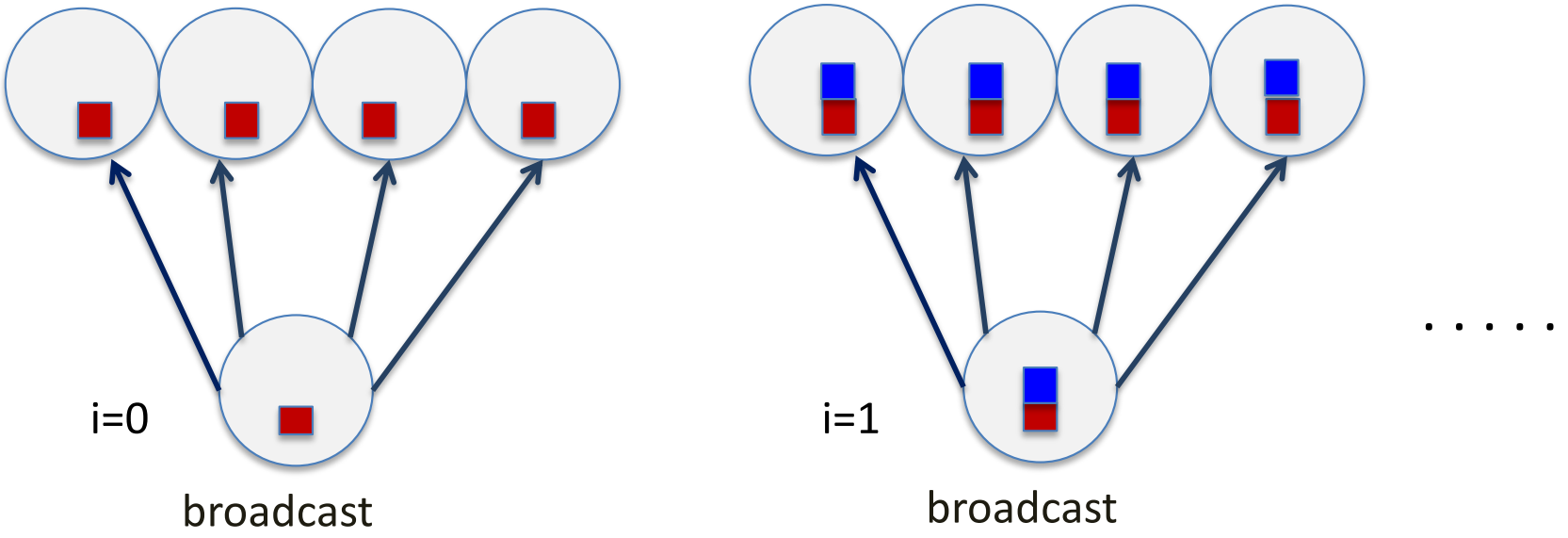


Note that these are local arrays allocated on process's address space. They are not directly accessible by other processes.

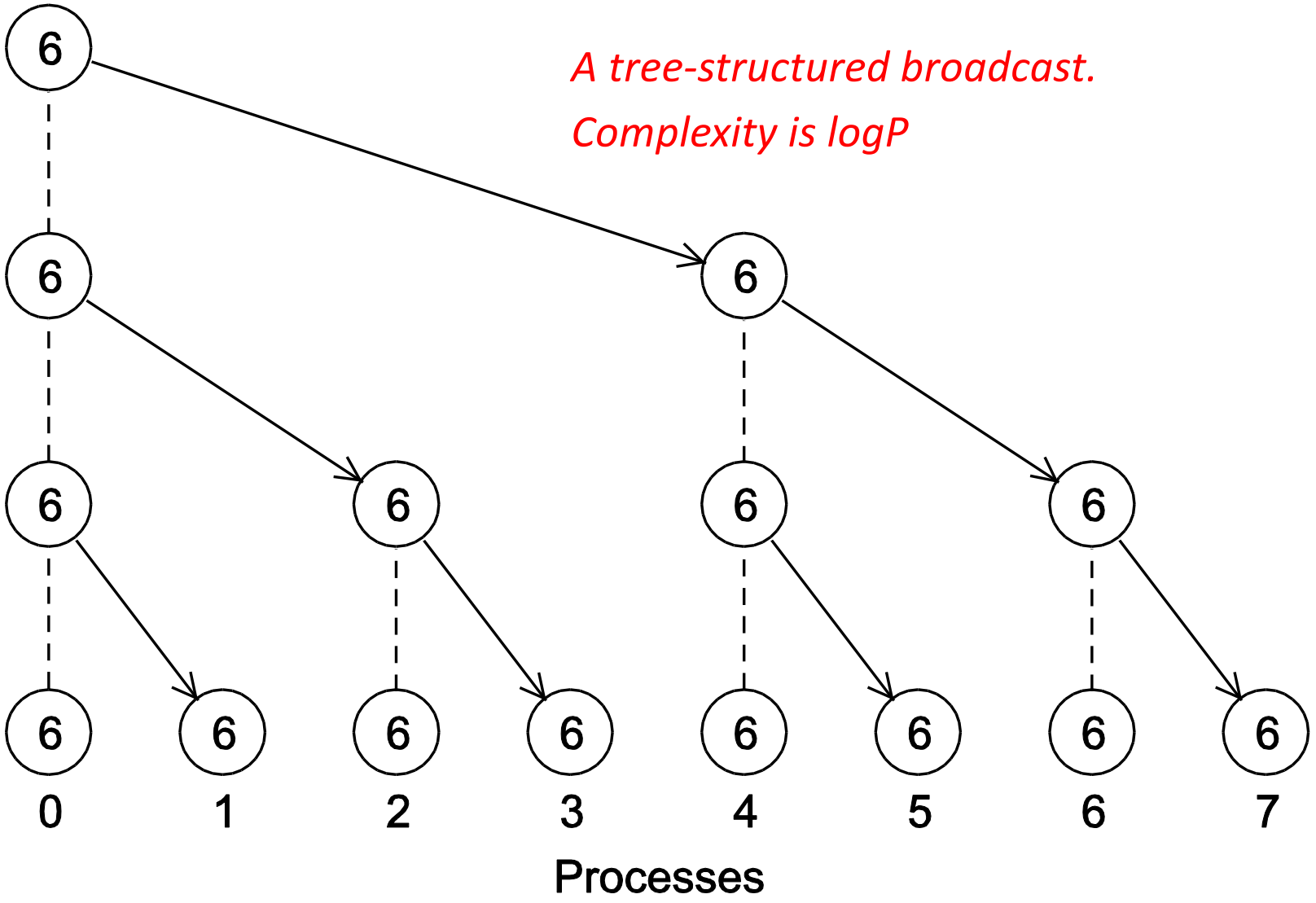
A, my_x and my_y represents the portion of the vectors assigned to this process.
X and Y are the entire global vectors

An MPI Example: Matrix Vector Multiply

```
/* collect x vector on all processors */
/* first copy my_x into x */
 for (j = 0; j < M; ++j)
   x[myrank*M +j] = my_x[j];

 for (i = 0; i < P; ++i)
   MPI_Bcast(&(x[myrank*M]), M, MPI_DOUBLE, i, MPI_COMM_WORLD);
```



A tree-structure Broadcast Communication

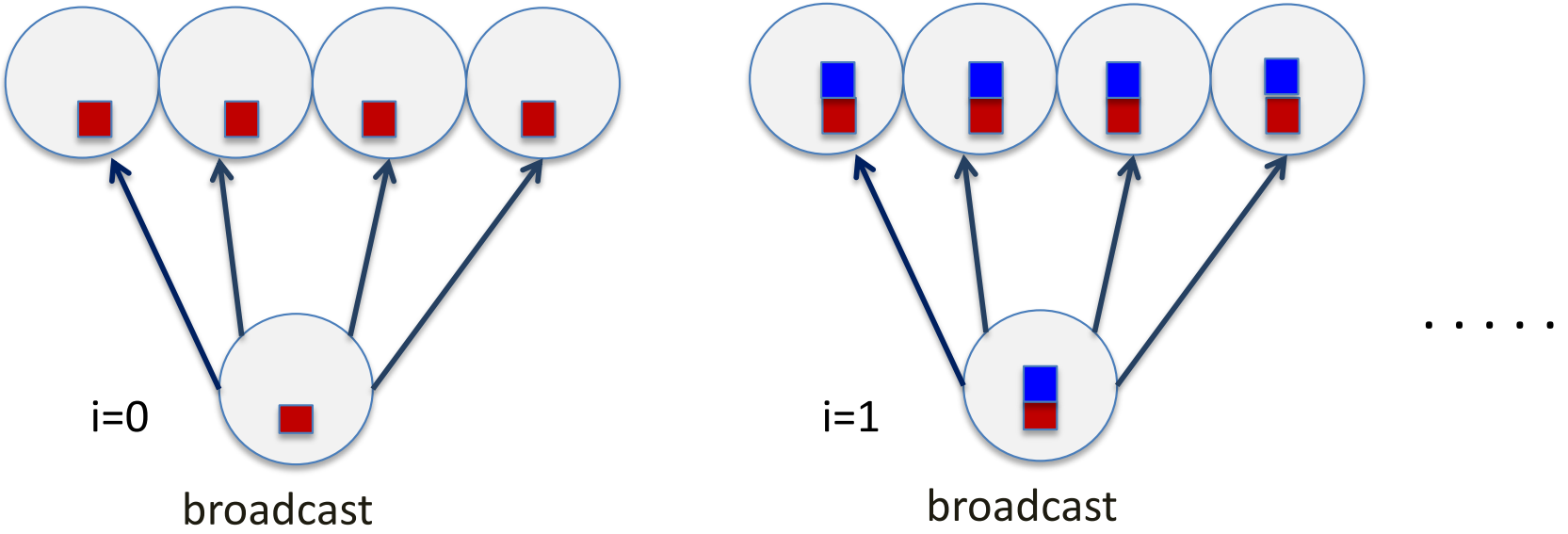


A tree-structured broadcast.

Complexity is logP

An MPI Example: Matrix Vector Multiply

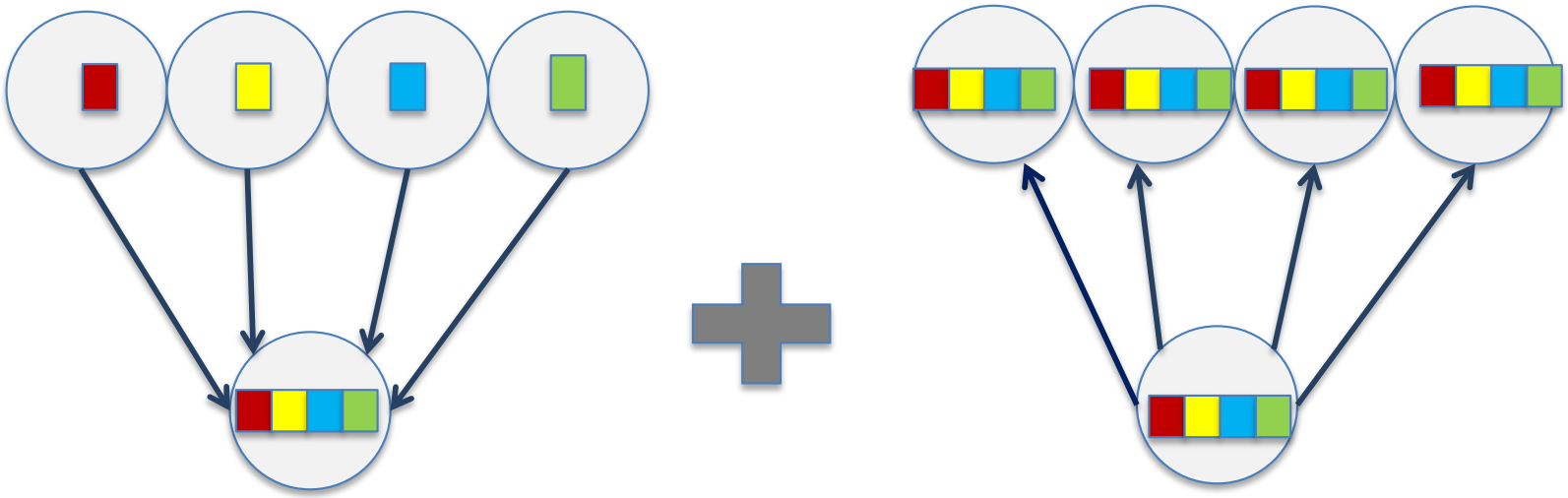
```
/* collect x vector on all processors */
/* first copy my_x into x */
 for (j = 0; j < M; ++j)
   x[myrank*M +j] = my_x[j];

 for (i = 0; i < P; ++i)
   MPI_Bcast(&(x[myrank*M]), M, MPI_DOUBLE, i, MPI_COMM_WORLD);
```



- What is the total communication volume?
  Ω(PN) - Each process sends N/P elements to all P processes

An MPI Example: Matrix Vector Multiply

```
/* collect x vector on all processors */

MPI_Gather(my_x, M, MPI_DOUBLE, x, N, MPI_DOUBLE, 0, MPI_COMM_WORLD);

MPI_Bcast(x, N, MPI_DOUBLE, 0, MPI_COMM_WORLD);
```



gather broadcast

```
//Or even better
/* collect x vector on all processors */

MPI_Allgather(my_x, M, MPI_DOUBLE, x, N, MPI_DOUBLE, MPI_COMM_WORLD);
```

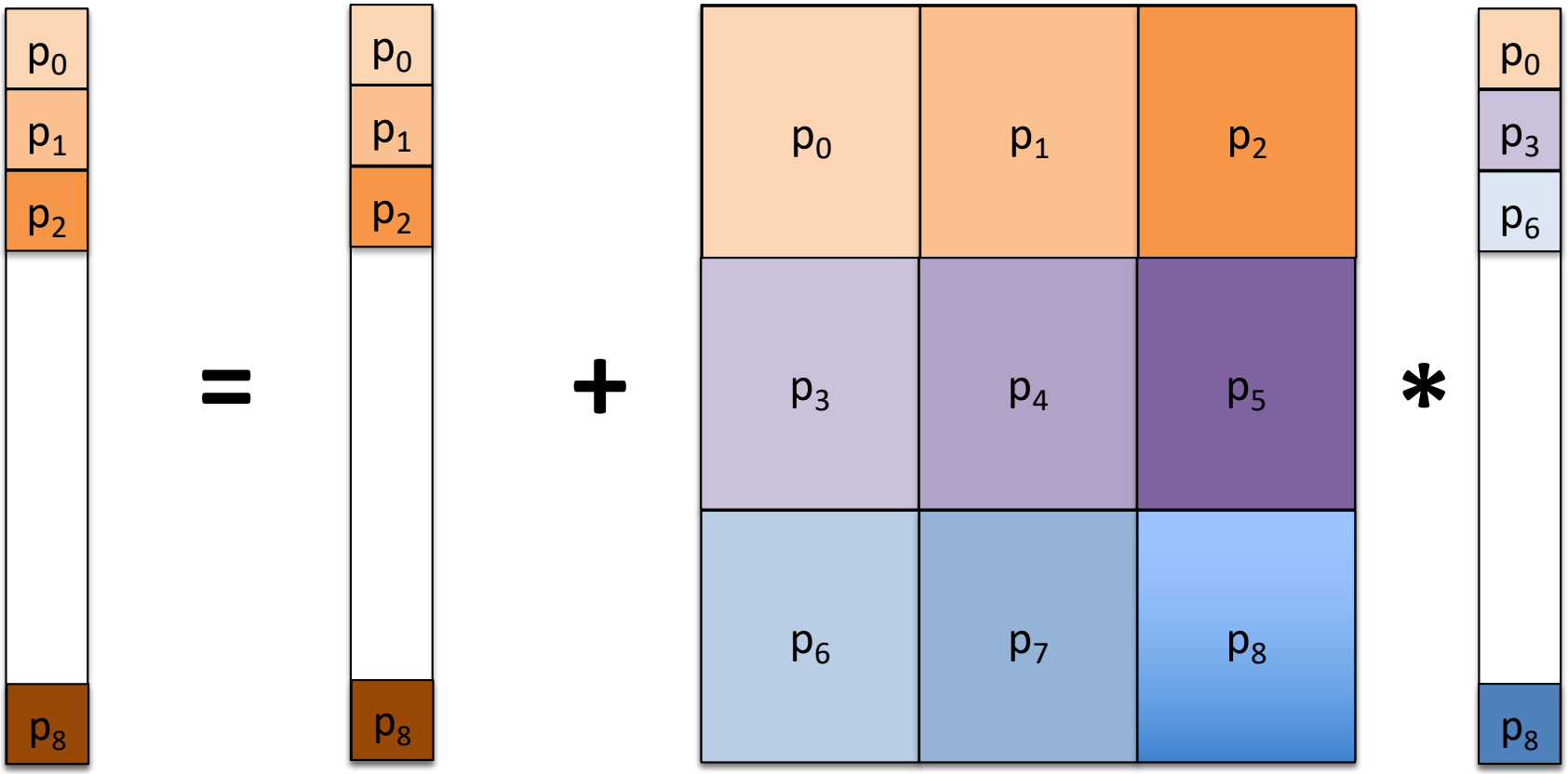
An MPI Example: Matrix Vector Multiply

```
/* local computations */
for (i = 0; i < M; ++i)
   for (j = 0; j < N; ++j)
      my_y[i] += A[i][j] * x[j];

/* collect partial results at all processors */
 MPI_Allgather(my_y, M, MPI_DOUBLE, y, N, MPI_DOUBLE, MPI_COMM_WORLD);
```

- MPI_Allgather is simpler to implement and potentially more efficient than a series of broadcasts
  - But it still performs Ω(PN) operations.
- Nevertheless, Ω(PN) is the lower bound on the communication volume and it is not likely to scale well!
- Can we do better?

An MPI Example: Matrix Vector Multiply

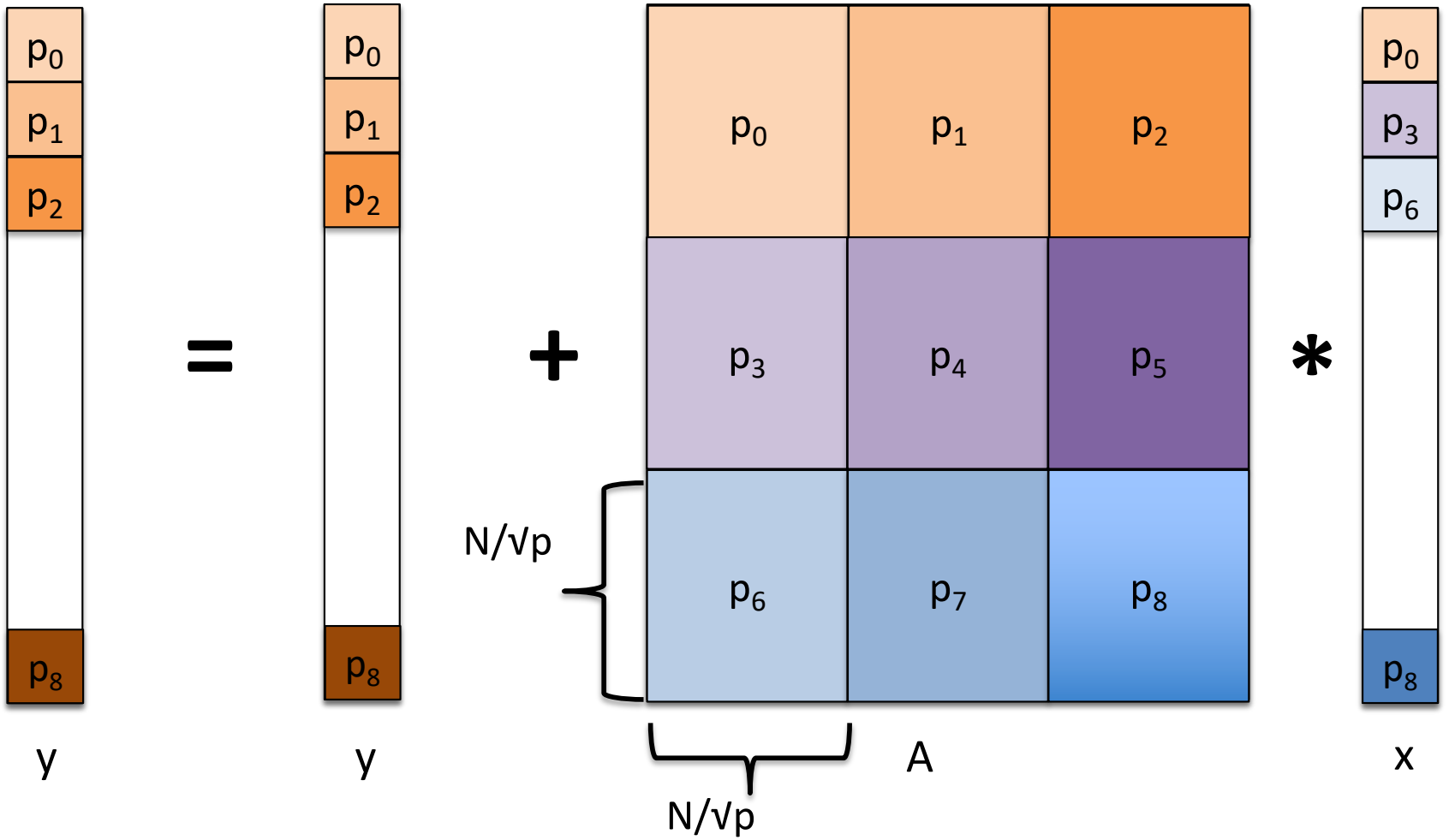
- Let's decompose the matrix A into 2D blocks
- √p processes are in the same row block
- √p processes are in the same column block



Colors indicate the initial storage in each processor.

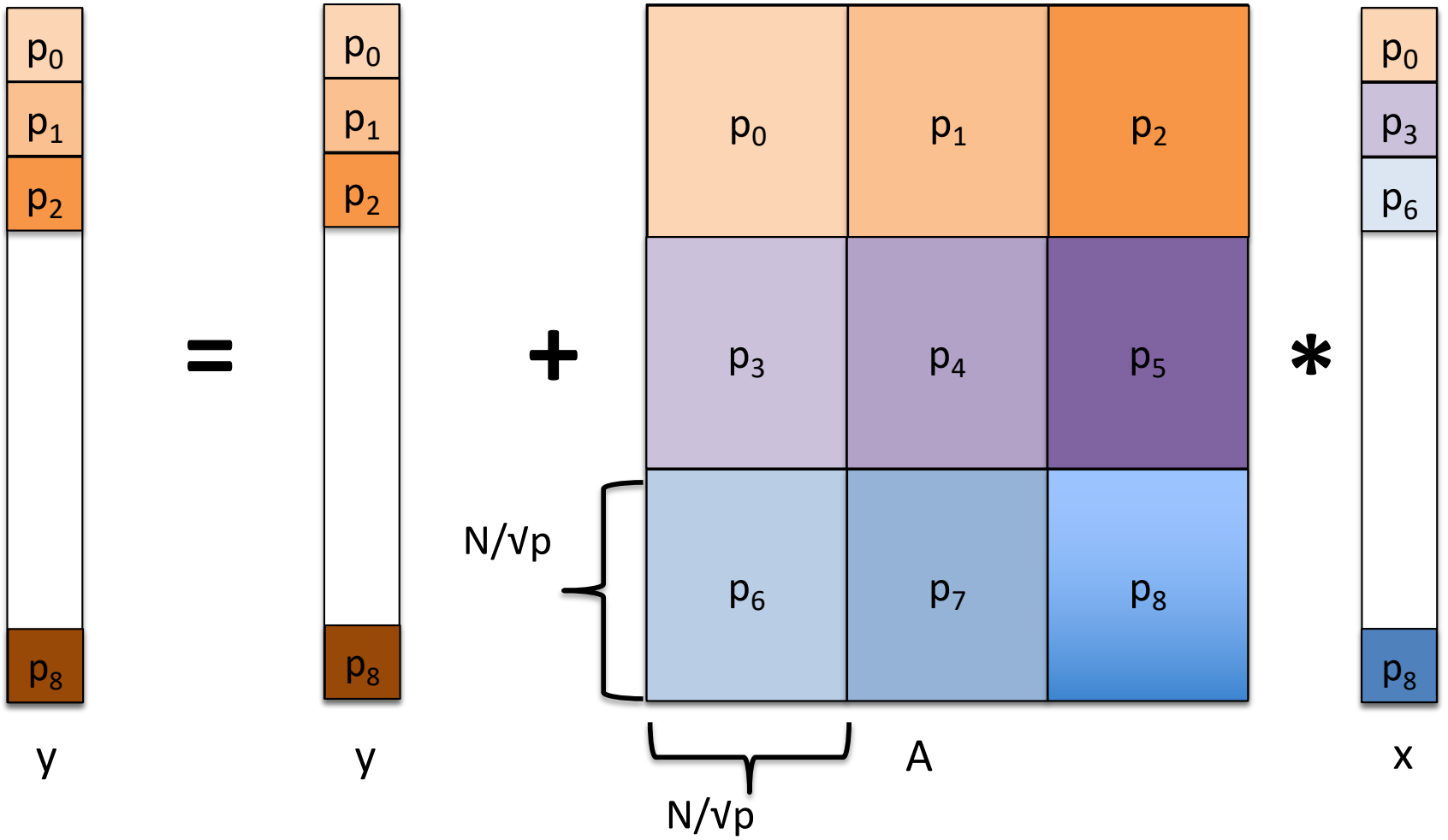
An MPI Example: Matrix Vector Multiply

- How is a 2D decomposition different from 1D decomposition?
  - Number of matrix elements?
  - Same, $(N/P^{1/2})^2 = N^2/P$. So the computational load per process does not change!



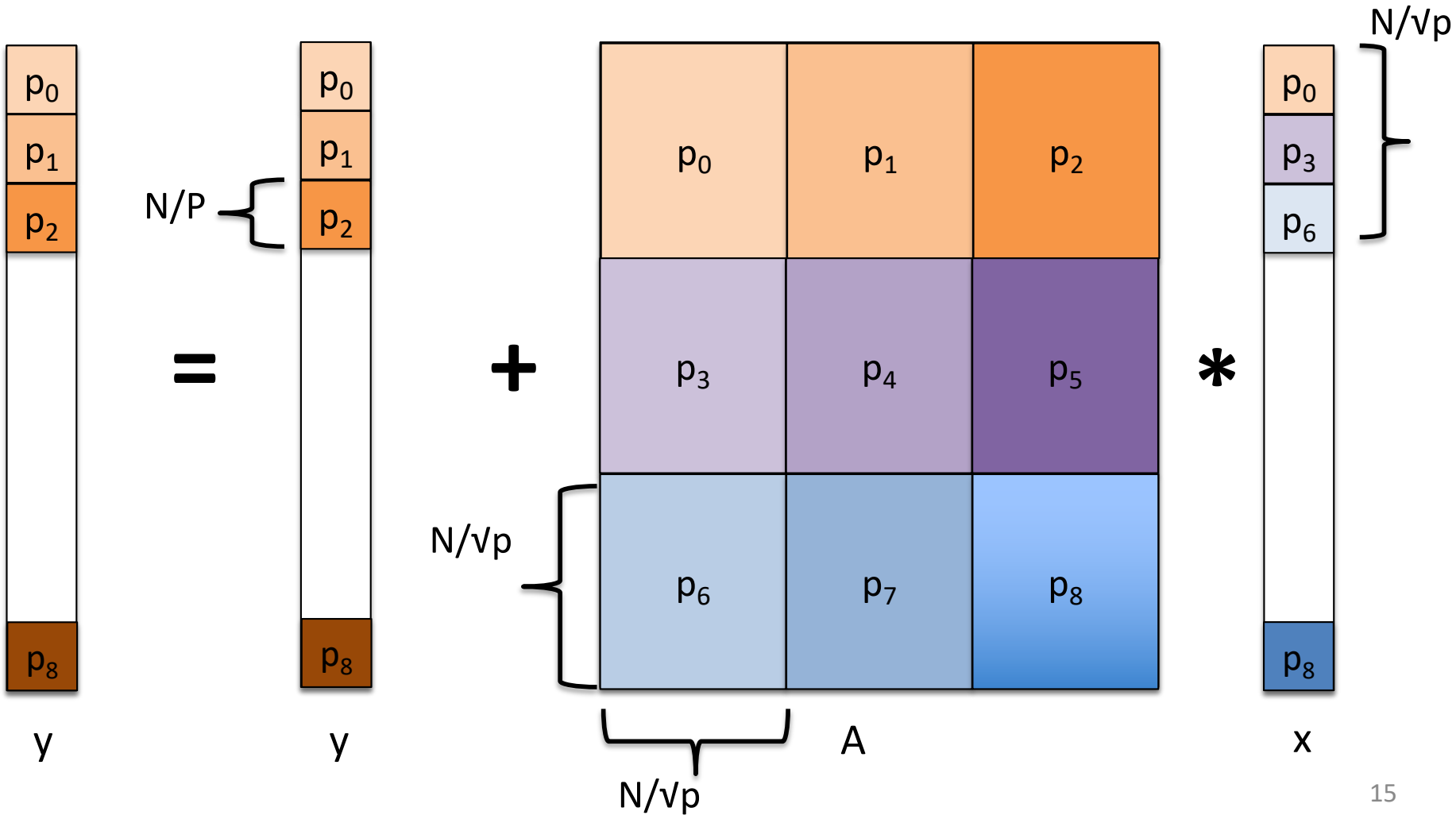
An MPI Example: Matrix Vector Multiply

- How about communication volume?



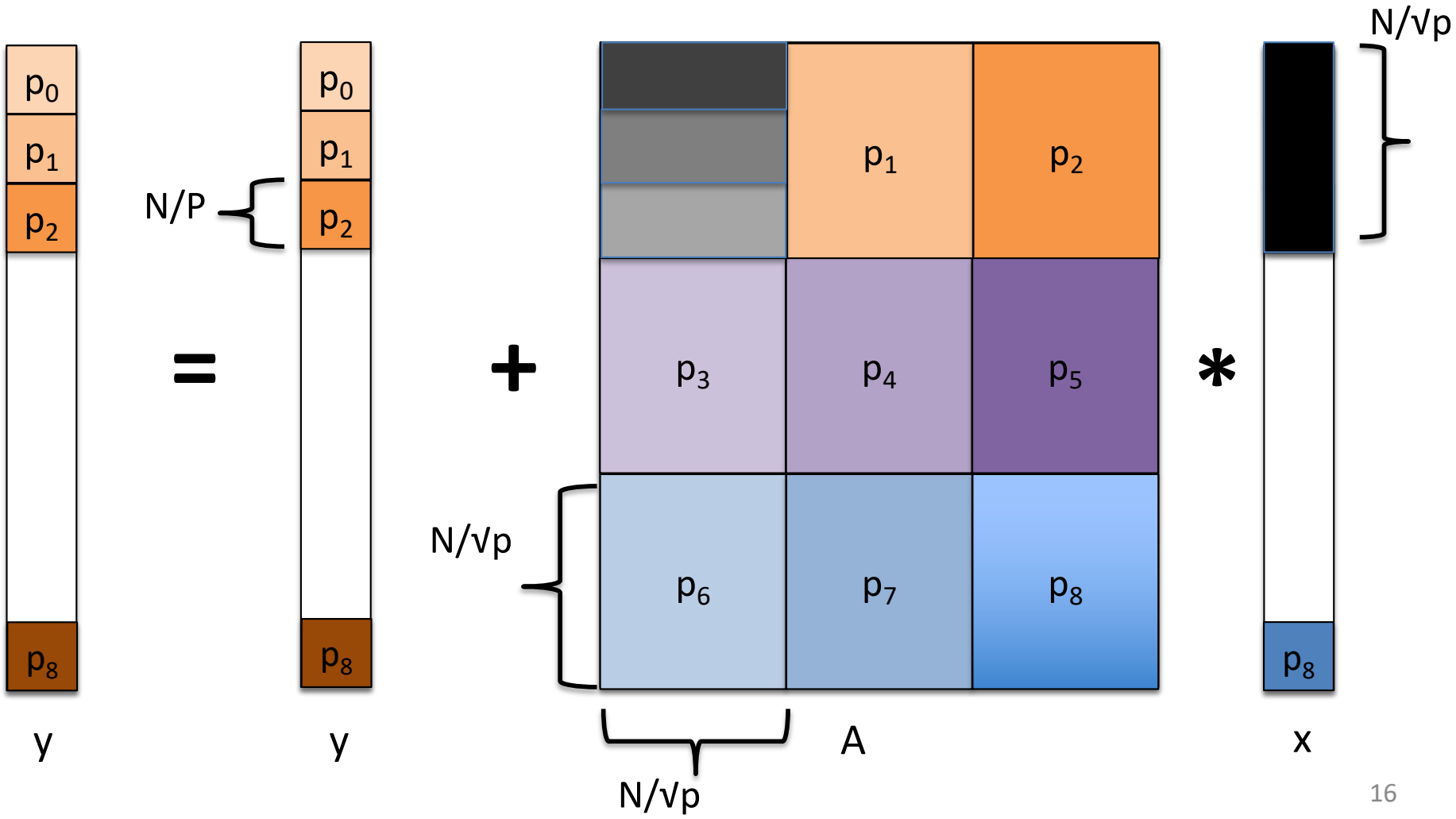
An MPI Example: Matrix Vector Multiply

- Each processor still starts with an N/P partition of *y*,
- But each one needs only a partition of *x* which is of length $N/P^{1/2}$
  - As opposed to entire x in 1D partition



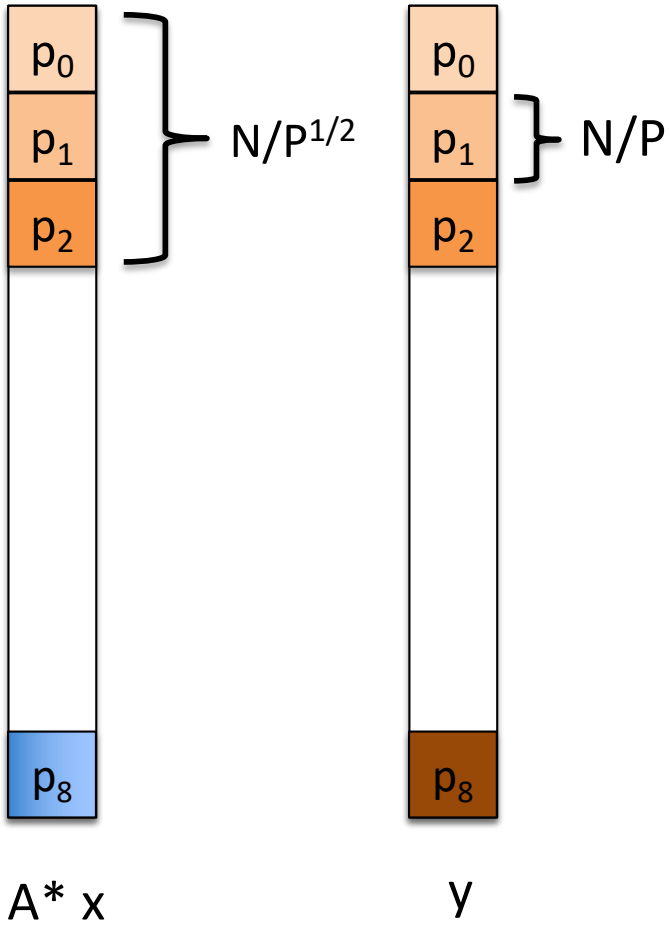
An MPI Example: Matrix Vector Multiply

- Local *Ax* creates a partial result of *y*
  - Need to combine the results from other processes in the same row block
  - Each process has $N/P^{1/2}$ of *Ax* but only needs compute N/P of *y*



An MPI Example: Matrix Vector Multiply

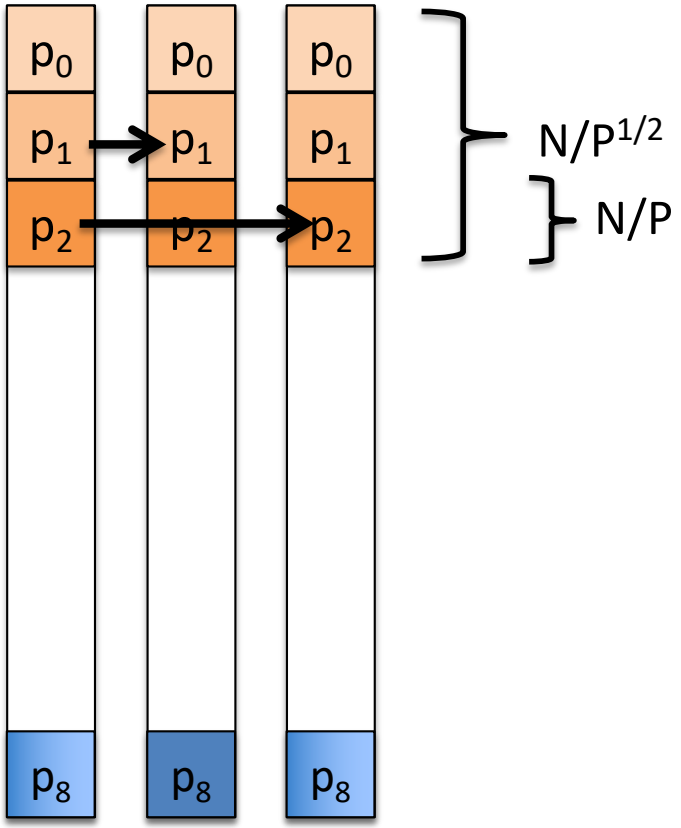
- Local *Ax* creates a partial result of *y*
  - Need to combine the results from other processes in the same row block
  - Each process has $N/P^{1/2}$ of *Ax* but only needs compute N/P of *y*



$N/P^{1/2}$ - N/P

A process keeps N/P elements for itself and sends the rest of the partial result to other processes in the same row block.

An MPI Example: Matrix Vector Multiply



A* x

Per process comm volume

$P^{1/2} * P^{1/2} * (N/P^{1/2} - N/P)$

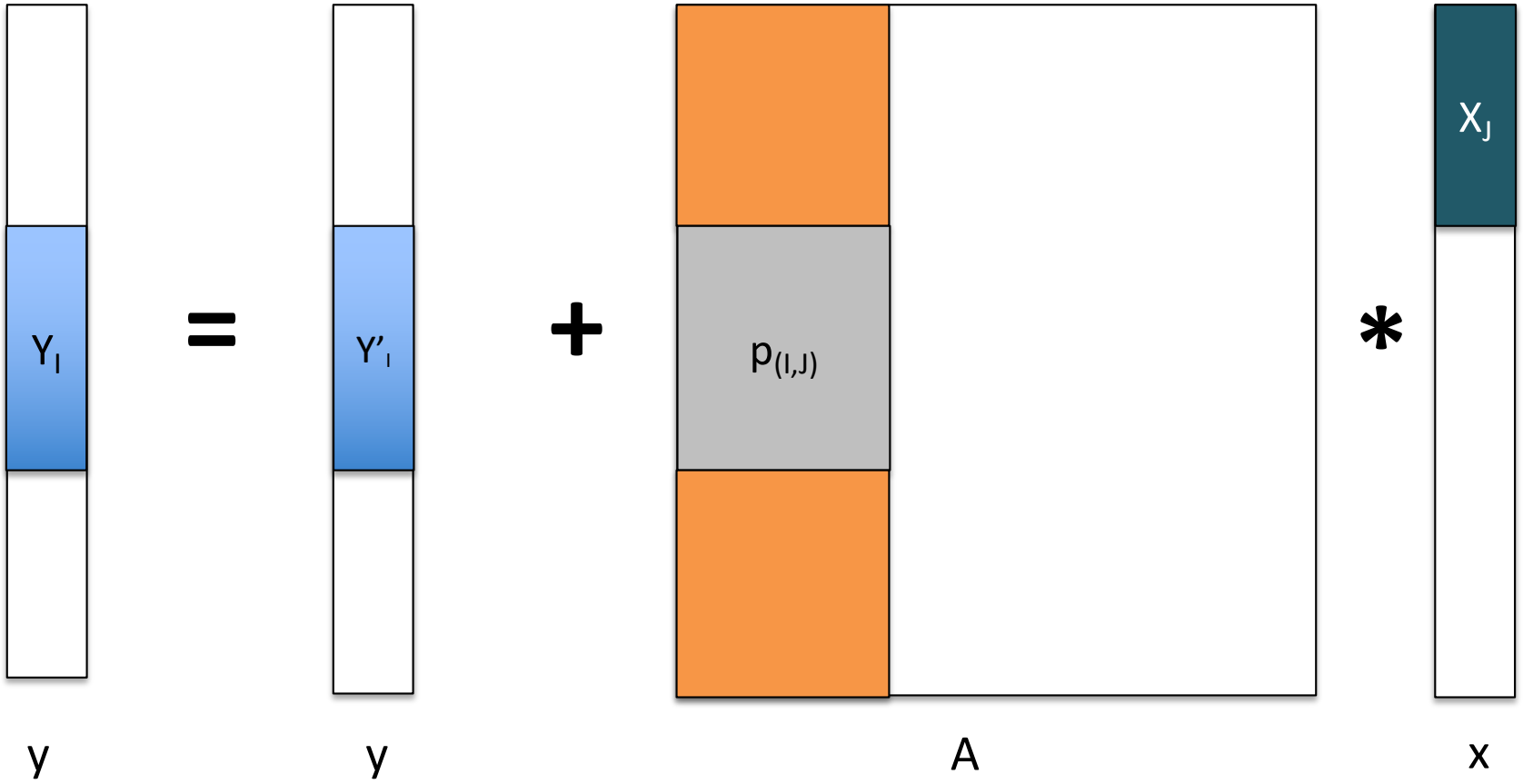
There are $P^{1/2}$ processes in the same row block.

There are $P^{1/2}$ processes in the same column block.

- Input communication volume is now $\Omega(P(N/P^{1/2} - N/P)) = \Omega(P^{1/2}N)$
  - compare with the 1D partitioning case which was $\Omega(PN)$

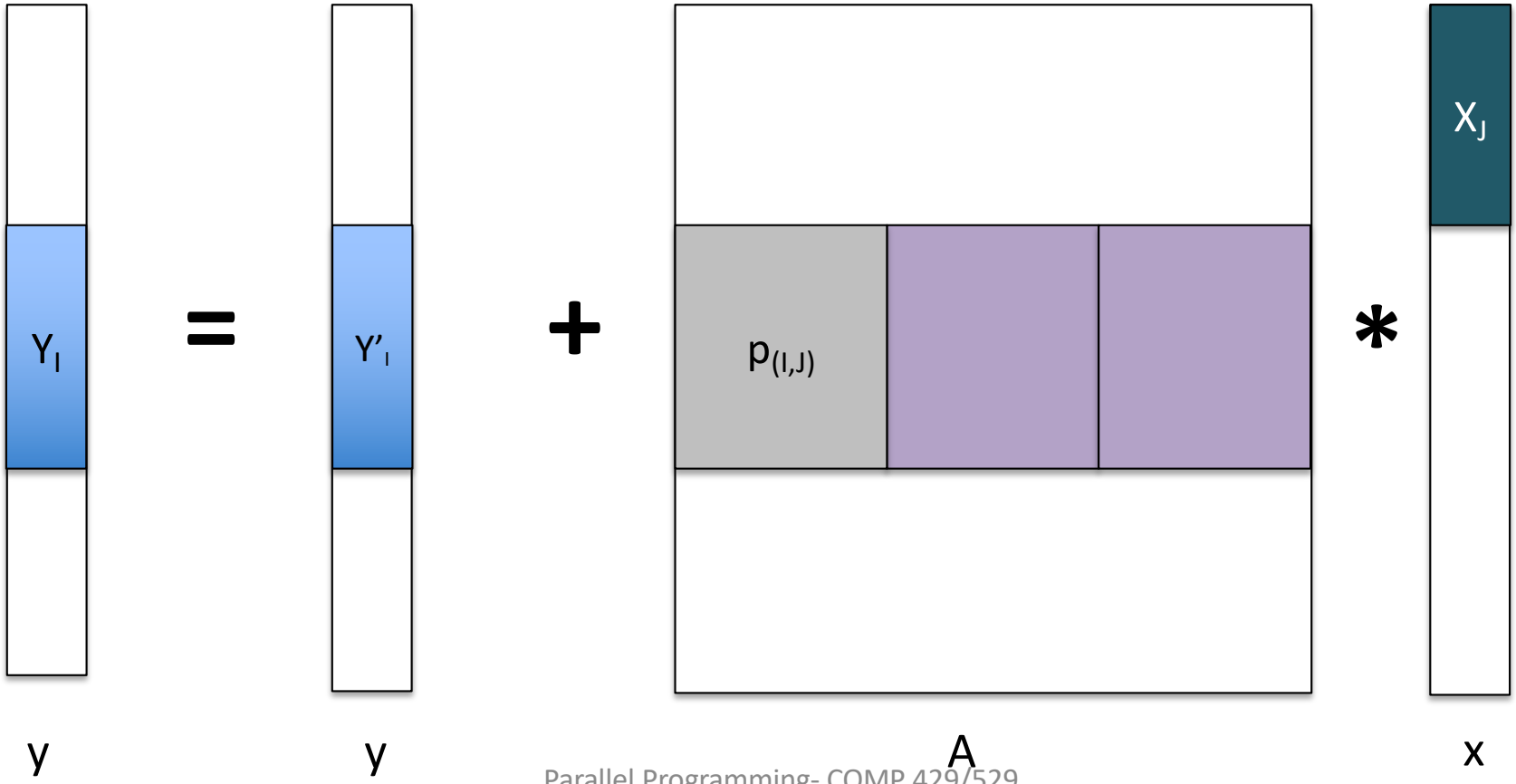
An MPI Example: Matrix Vector Multiply

- Let us consider the processor at the *i*th row and *j*th column
- It needs to talk to $P^{1/2}$ other processors on its column for input!
- So the input vector can be collected through a **gather**



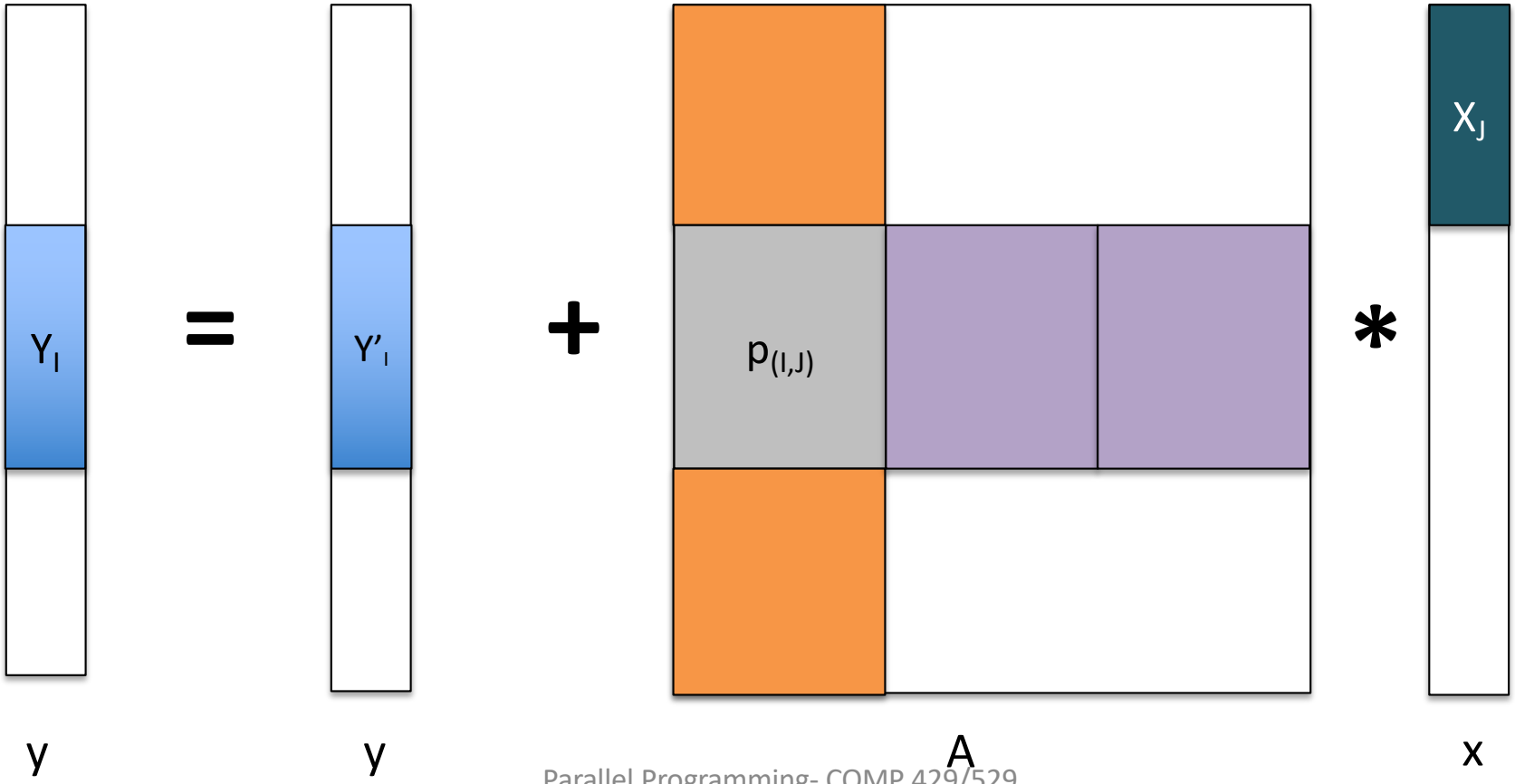
An MPI Example: Matrix Vector Multiply

- But what about the output vector?
  - Matrix-vector multiply in block notation: Y'[I] += A[I][J] * X[J]
- The output of each processor is only **partial**, hence denoted by Y'
- Need a **reduction** this time along rows



An MPI Example: Matrix Vector Multiply

- So instead of the default communicator `MPI_COMM_WORLD`, we now need column and row communicators!
- This can be implemented using `MPI_Comm_split`!
- Use the row and column index values as colors



Groups and Communicators

- In many parallel algorithms, communication operations need to be restricted to certain subsets of processes.
  - MPI provides mechanisms for partitioning the group of processes that belong to a communicator into subgroups each corresponding to a different communicator.
- The simplest such mechanism is:

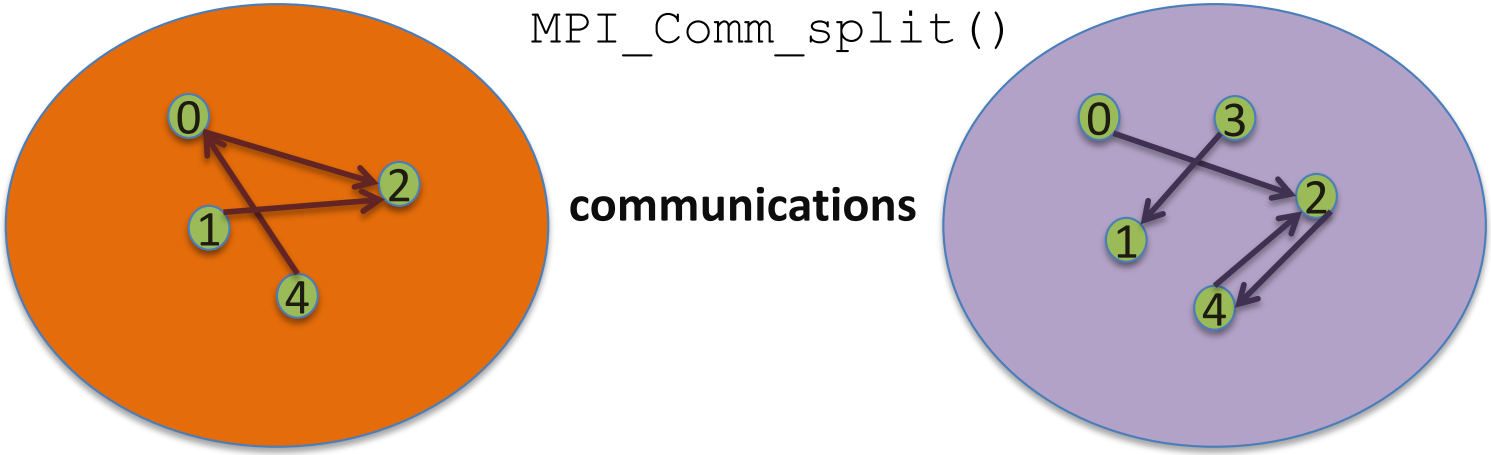
```
int MPI_Comm_split(MPI_Comm comm, int color, int key,
                   MPI_Comm *newcomm)
```

- This operation groups processors by color and sorts resulting groups on the key.

Communicators

Original rank 0, 1, 2, 3
New rank 0, 1, 2, 3

Original rank 4, 5, 6, 7, 8
New rank 0, 1, 2, 3, 4



MatVec Multiply

```
#include <mpi.h>
#include <stdio.h>
#include <sys/time.h>

#define N 3000

int main(int argc, char *argv[])
{
  int P, myrank, M, orange, purple;
  int i, j, sqrtP;
  double t_start, t_end;
  double **my_A, *my_x, *my_y, *x, *y;
  MPI_Comm  rowcomm, colcomm;

  /* Initializations */
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &P);
  MPI_Comm_rank(MPI_COMM_WORLD, &myrank);


  sqrtP = sqrt(P); // Assuming P is a perfect square
  M = N / P; // Assuming N is a multiple of P
  bigM = N / sqrtP;
  my_x = (double*) malloc(M * sizeof(double));
  my_y = (double*) malloc(M * sizeof(double));
  x = (double*) malloc(bigM * sizeof(double));
  y = (double*) malloc(bigM * sizeof(double));

  A = (double**) malloc(bigM * sizeof(double*));

  for (i = 0; i < bigM; ++i)
    A[i] = (double*) malloc(bigM * sizeof(double));

  //initialize local arrays my_x, my_y and A (e.g. from a file or randomly)
  ...
```

Two new communicators

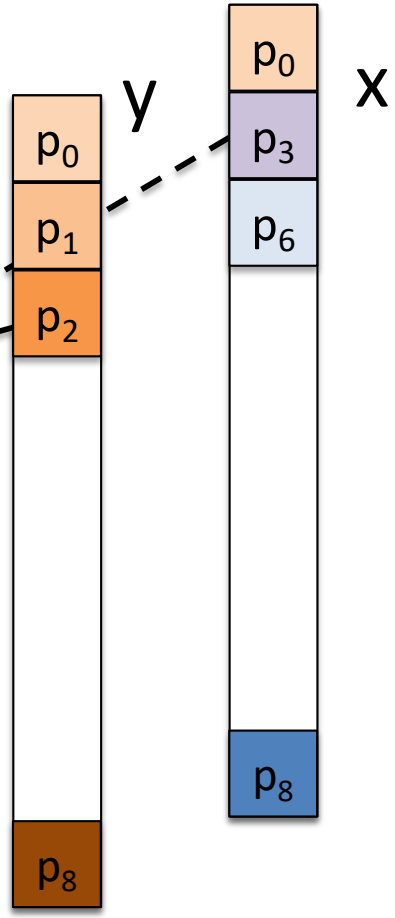
Note that matrix A has a dimension of square block bigM* bigM
Each process only knows the values for my_x, my_y and A.

Need to get the values for x and y from other processors

MatVec Multiply-2

Split communicator into two communicators: rowcomm and colcomm

```
/* setup communication groups */
purple = myrank / sqrtP; //rows
orange = myrank % sqrtP; //columns

MPI_Comm_split(MPI_COMM_WORLD, purple, myrank, &rowcomm);
MPI_Comm_split(MPI_COMM_WORLD, orange, myrank, &colcomm);

/* collect x vectors along column comms, y vector along row comms */
MPI_Allgather(my_x, M, MPI_DOUBLE, x, bigM, MPI_DOUBLE, colcomm);
MPI_Allgather(my_y, M, MPI_DOUBLE, y, bigM, MPI_DOUBLE, rowcomm);

/* local computations */
for (i = 0; i < bigM; ++i)
  for (j = 0; j < bigM; ++j)
     y[i] += A[i][j] * x[j];



/* collect partial results along rows */
for (i = 0; i < sqrtP; ++i)
  MPI_Reduce(&(y[M*i]), my_y, M, MPI_DOUBLE, MPI_SUM, i, rowcomm);
```




```
/* Alternative to MPI_Reduce potentially more efficient */
/* collect partial results along rows */

MPI_Reduce_scatter(y, my_y, array_of_M, MPI_DOUBLE, MPI_SUM, rowcomm);
```

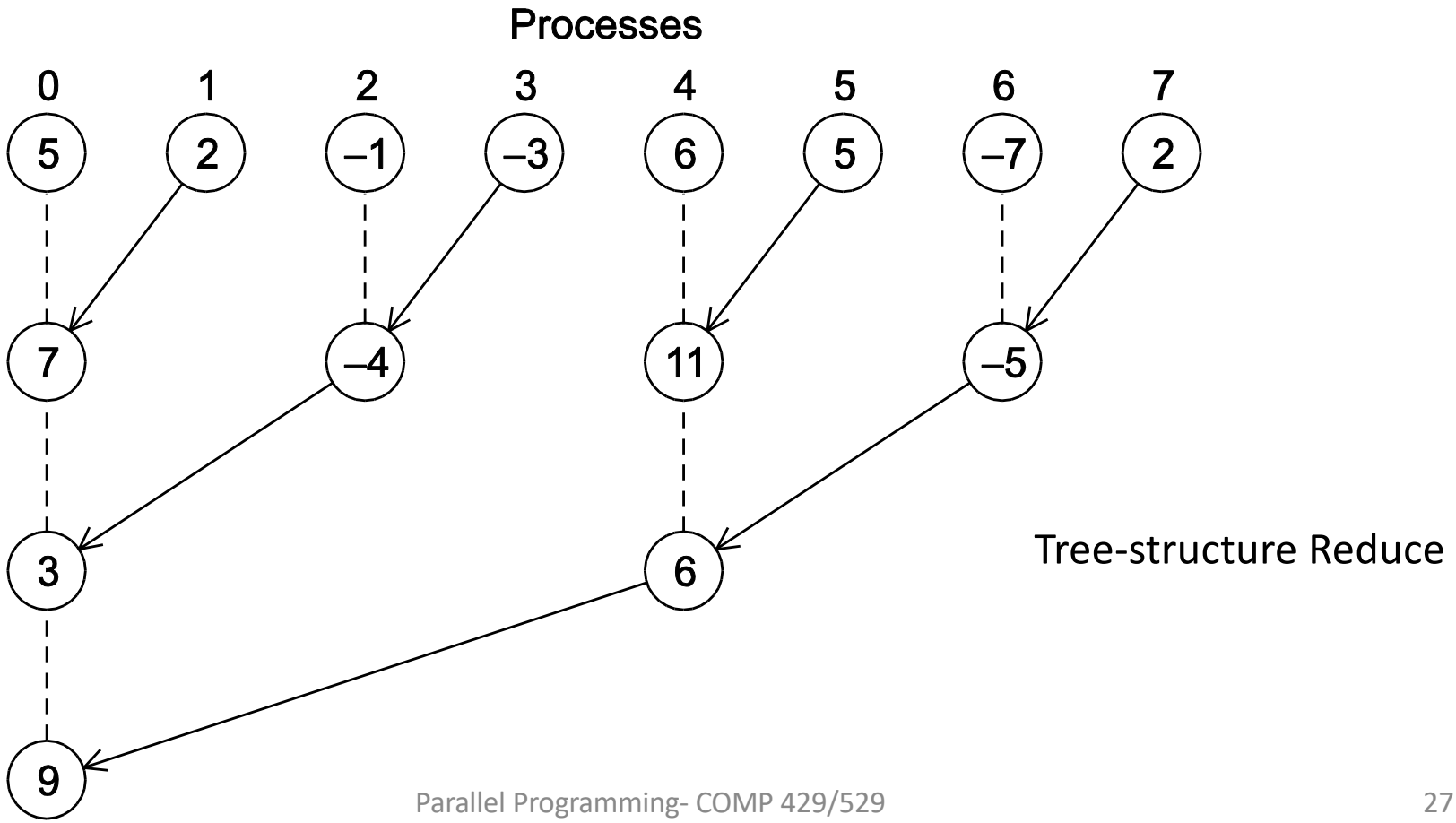
Mat-Vec Summary

- We have seen that decisions made at an algorithmic level have important consequences in terms of communication overheads!
  - We didn't change the compute load at all

- Not only that, but how we implement the algorithm is important! (a series of bcasts vs. allgather, or a series of reduces vs. reduce_scatter)

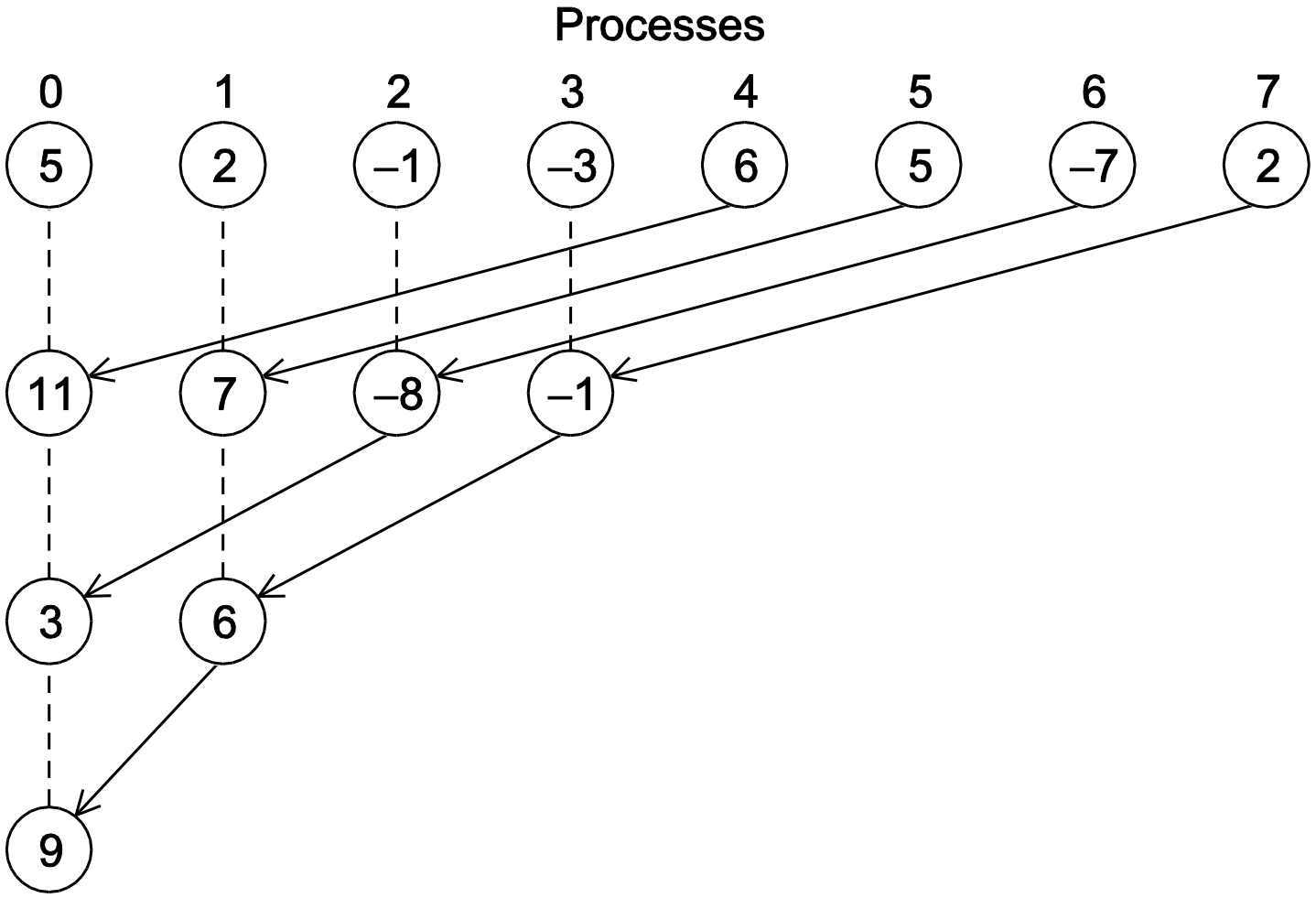
A tree-structure Communication

- MPI has an efficient implementation of collective communication operations
  - Use them over send/receive in a for loop

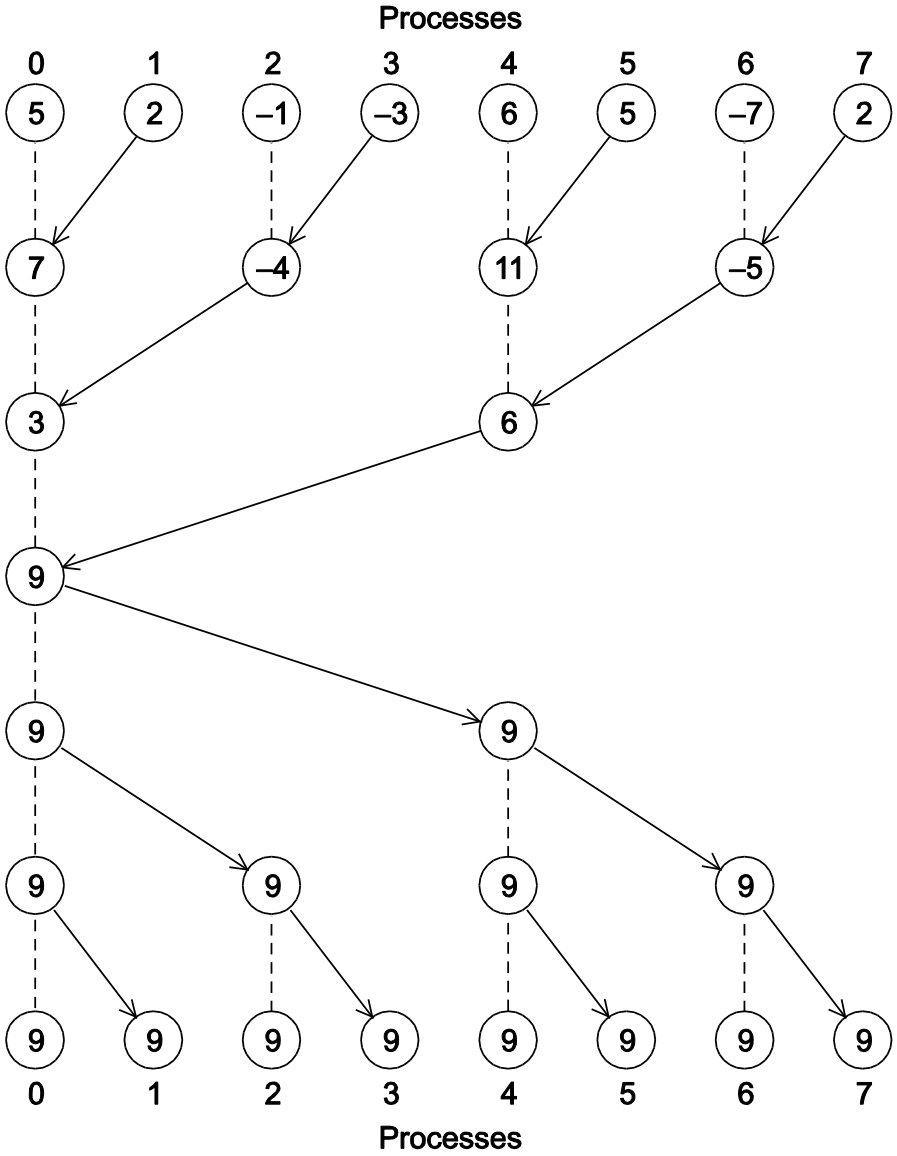


A tree-structure Communication

- MPI may use an alternative tree-structure

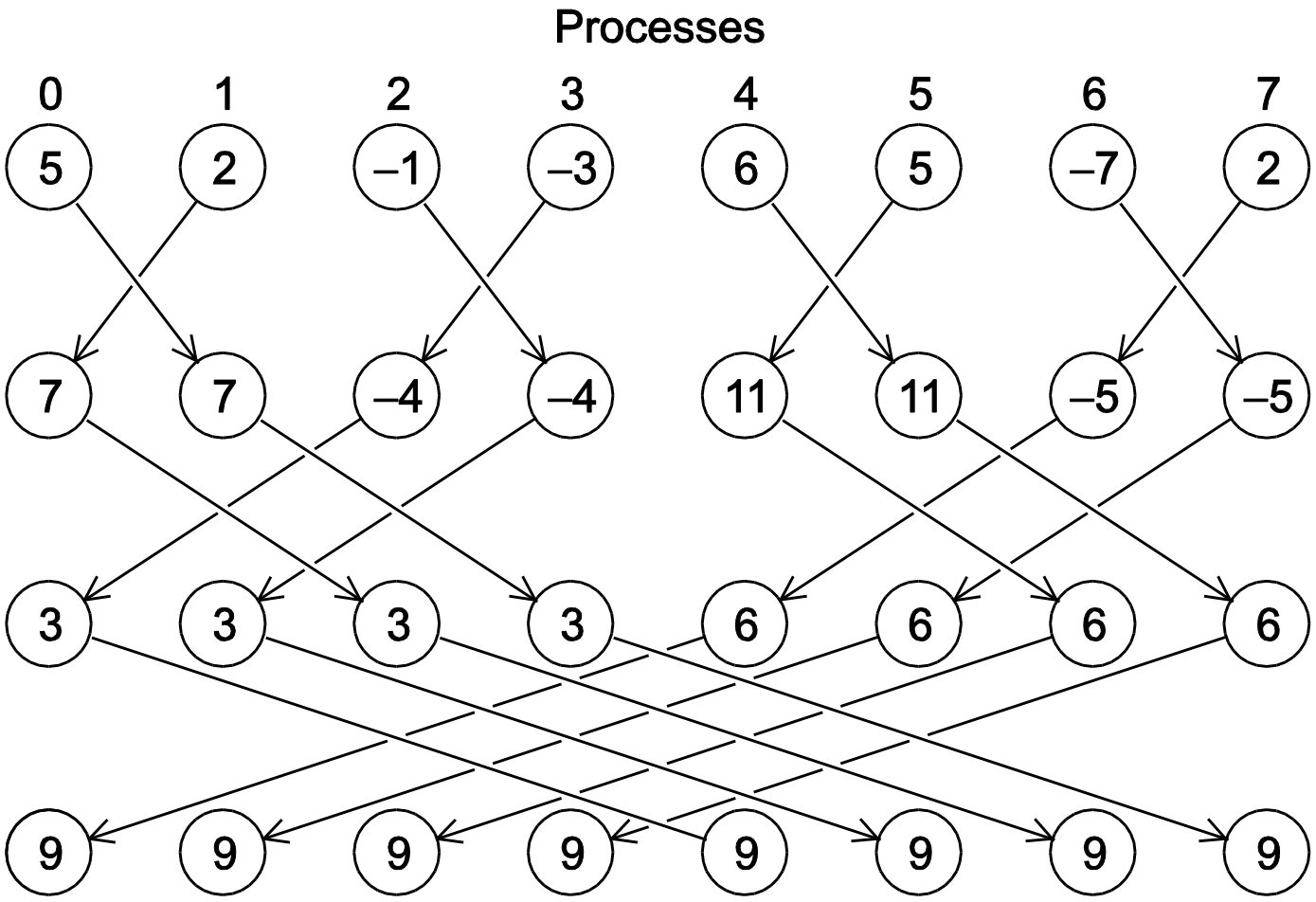


MPI_AllReduce()



- Reduce followed by a broadcast

MPI_AllReduce()



- A butterfly-structured global sum

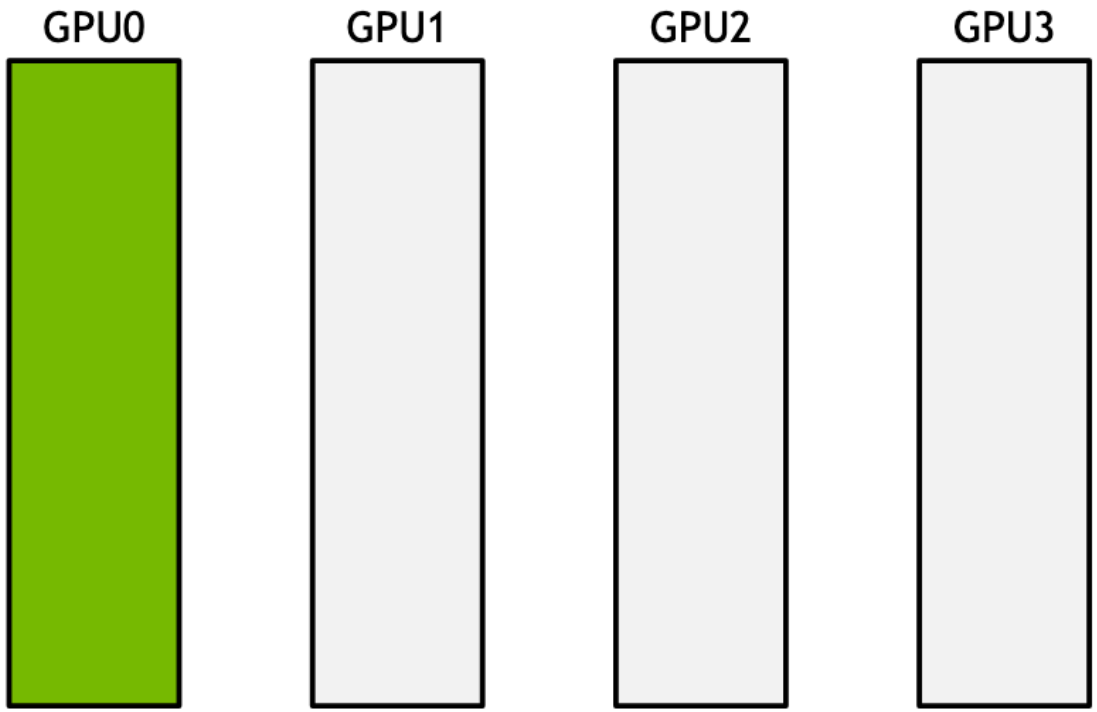
NCCL

- Nvidia Collective Communication Library
  - Accelerated collectives that are topology-aware
  - Mostly designed to improve the scalability of multi-GPU applications (DL applications)
- Pattern the library after MPI's collectives
- Handle the intra-node communication and inter-node communication in an optimal way
- Host-side API
- Asychronous calls

NCCL

BROADCAST

with unidirectional ring

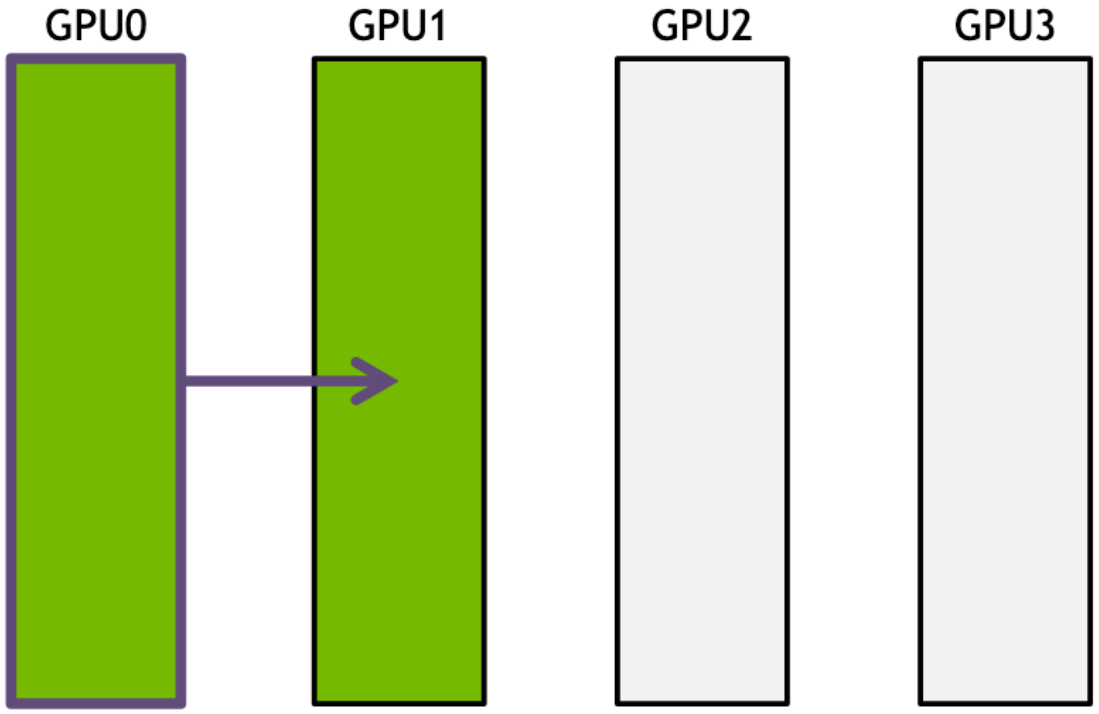


NVIDIA

NCCL

BROADCAST

with unidirectional ring



Step 1: $\Delta t = N/B$

N: bytes to broadcast

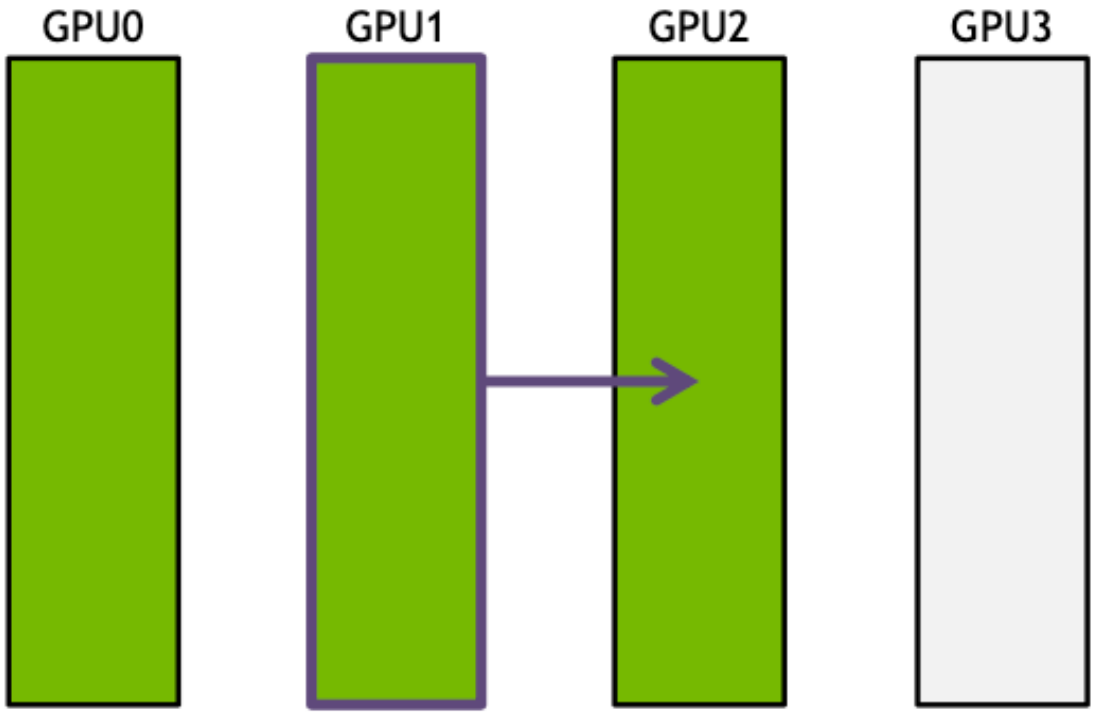
B: bandwidth of each link

NVIDIA

NCCL

BROADCAST

with unidirectional ring



Step 1: $\Delta t = N/B$

Step 2: $\Delta t = N/B$

N: bytes to broadcast

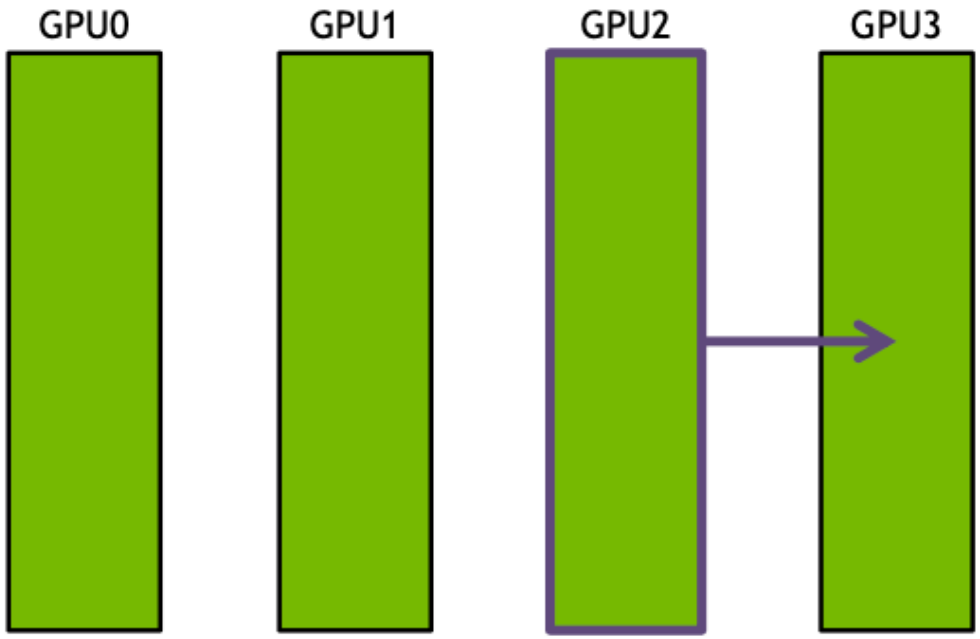
B: bandwidth of each link

NVIDIA

NCCL

BROADCAST

with unidirectional ring



Step 1: $\Delta t = N/B$

Step 2: $\Delta t = N/B$

Step 3: $\Delta t = N/B$

N: bytes to broadcast

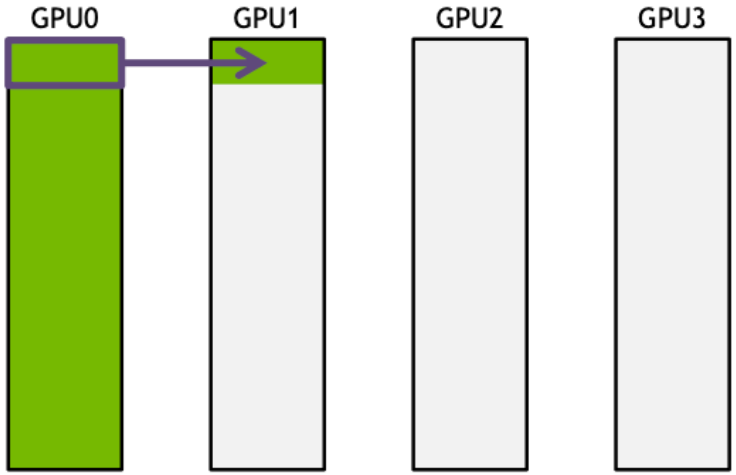
B: bandwidth of each link

NVIDIA

NCCL

BROADCAST

with unidirectional ring



Split data into S messages

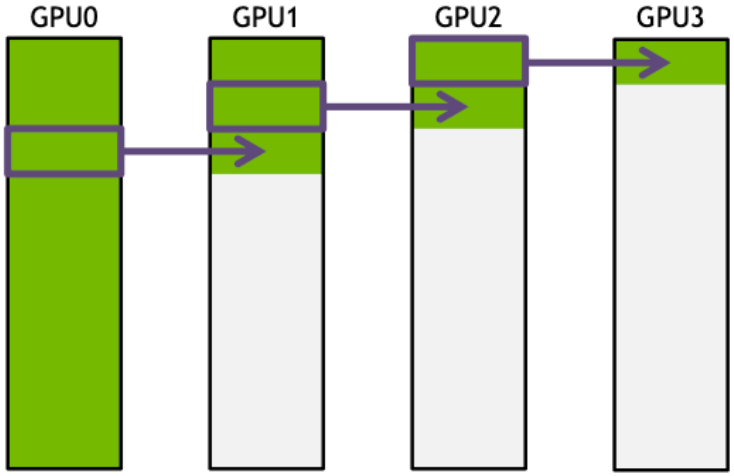
Step 1: $\Delta t = N/(SB)$

NVIDIA

NCCL

BROADCAST

with unidirectional ring



Split data into S messages

Step 1: $\Delta t = N/(SB)$

Step 2: $\Delta t = N/(SB)$

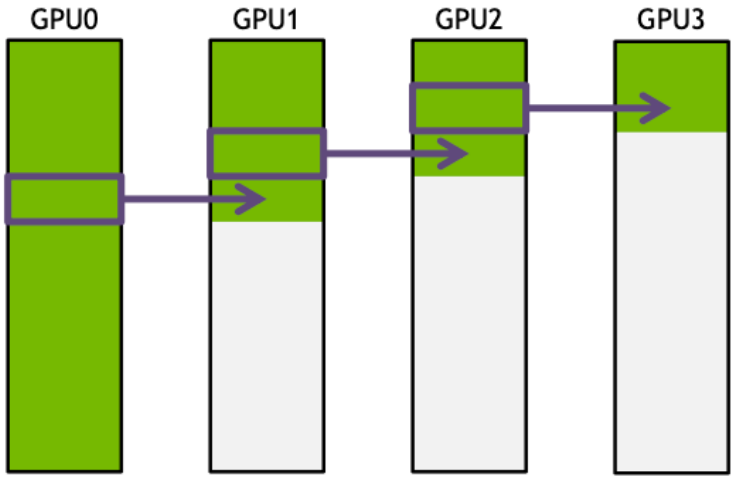
Step 3: $\Delta t = N/(SB)$

NVIDIA

NCCL

BROADCAST

with unidirectional ring



Split data into S messages

Step 1: $\Delta t = N/(SB)$

Step 2: $\Delta t = N/(SB)$

Step 3: $\Delta t = N/(SB)$

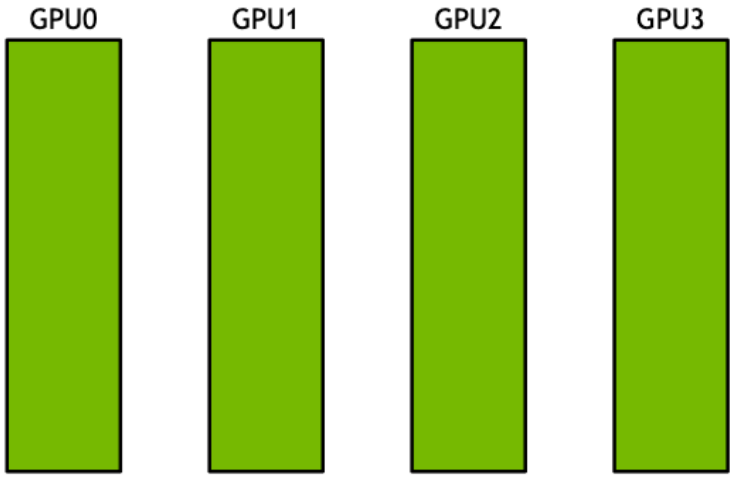
Step 4: $\Delta t = N/(SB)$

NVIDIA

NCCL

BROADCAST

with unidirectional ring



Split data into S messages

Step 1: $\Delta t = N/(SB)$

Step 2: $\Delta t = N/(SB)$

Step 3: $\Delta t = N/(SB)$

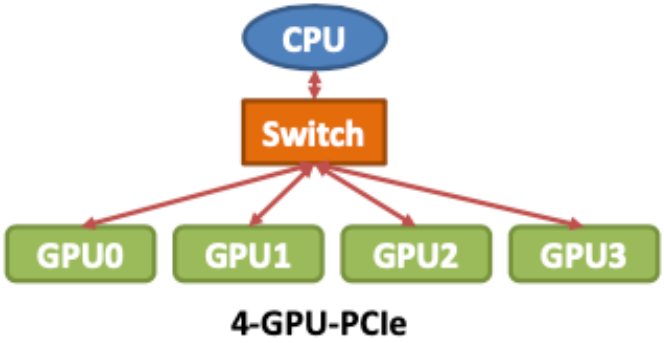
Step 4: $\Delta t = N/(SB)$

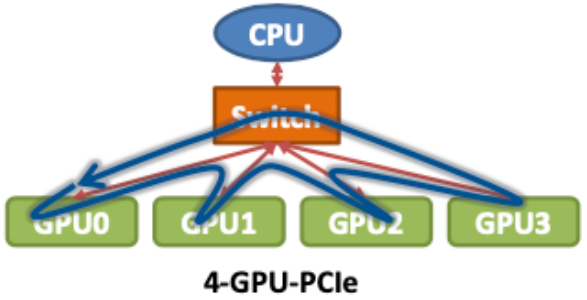
...

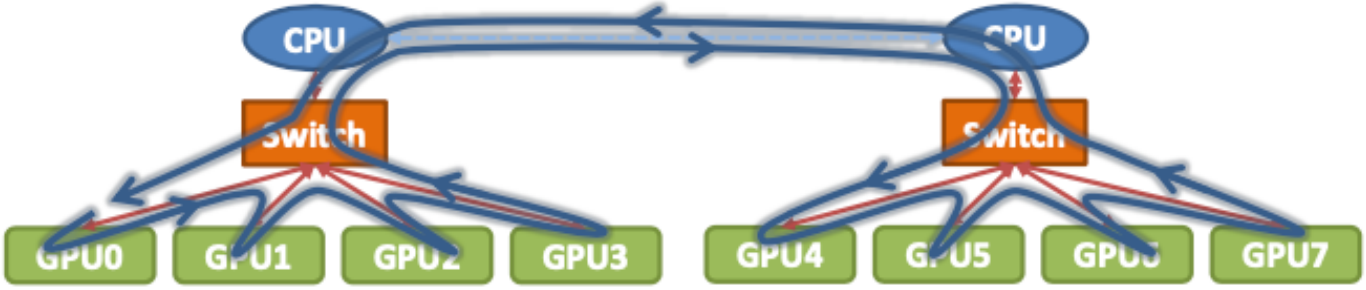
Total time:
$SN/(SB) + (k-2)\,N/(SB)$
$= N(S+k-2)/(SB) \rightarrow N/B$

NVIDIA

Ring Topology







Acknowledgments

- These slides are inspired and partly adapted from
  - Metin Aktulga (Michigan State Univ.)
  - Scott Baden (UCSD)
  - The course book (Pacheco)
  - https://computing.llnl.gov/tutorials/mpi/
  - Nvidia