

CSE 160

Lecture 22

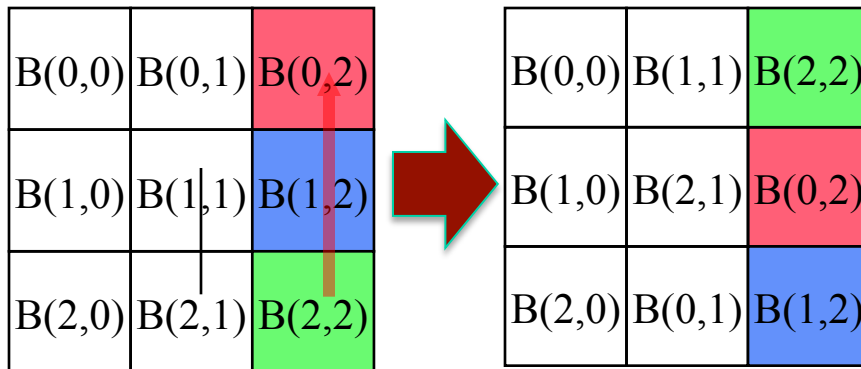
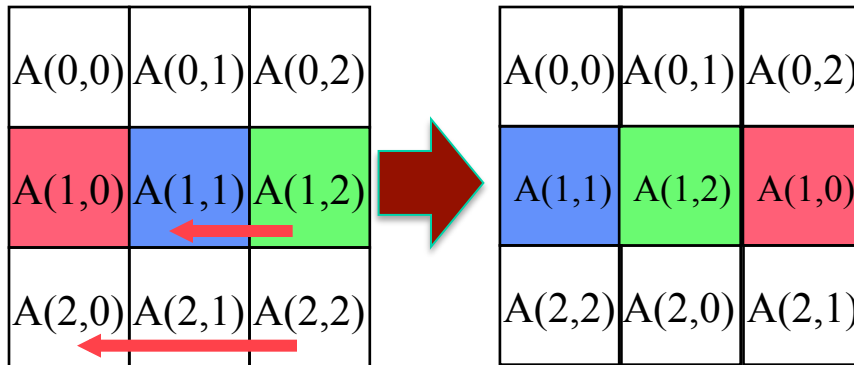
Matrix Multiplication – Continued
Managing communicators
Gather and Scatter (Collectives)

Today's lecture

- Managing communicators in Cannon's Algorithm
- Gather Scatter

Skewing the matrices

$$C[1,2] = A[1,0]*B[0,2] + A[1,1]*B[1,2] + A[1,2]*B[2,2]$$



- We want A[1,0] and B[0,2] to reside on the same processor initially
- Shift rows and columns so the next pair of values A[1,1] and B[1,2] line up
- And so on with A[1,2] and B[2,2]

Shift and multiply

$$C[1,2] = A[1,0]*B[0,2] + A[1,1]*B[1,2] + A[1,2]*B[2,2]$$

- Takes $\sqrt{p}$ steps
- Circularly shift
 - each row by 1 column to the left
 - each column by 1 row to the left
- Each processor forms the product of the two local matrices adding into the accumulated sum

A(0,0)	A(0,1)	A(0,2)
A(1,1)	A(1,2)	A(1,0)
A(2,2)	A(2,0)	A(2,1)

A(0,1)	A(0,2)	A(0,0)
A(1,2)	A(1,0)	A(1,1)
A(2,0)	A(2,1)	A(2,2)



B(0,0)	B(1,1)	B(2,2)
B(1,0)	B(2,1)	B(0,2)
B(2,0)	B(0,1)	B(1,2)



B(1,0)	B(2,1)	B(0,2)
B(2,0)	B(0,1)	B(1,2)
B(0,0)	B(1,1)	B(2,2)

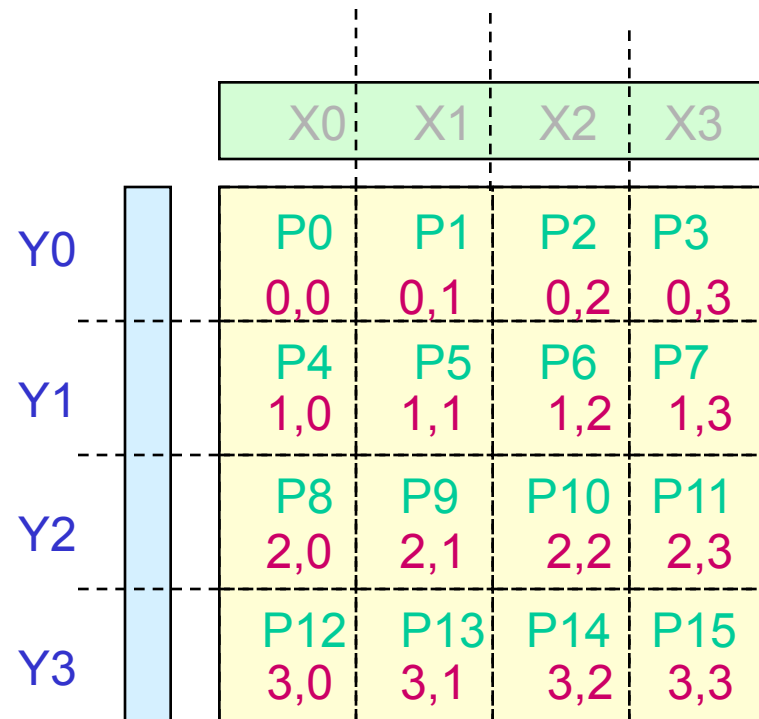
Implementation

Today's lecture

- Managing communicators
- Gather Scatter

Communication domains

- Cannon's algorithm shifts data along rows and columns of processors
- MPI provides communicators for grouping processors, reflecting the communication structure of the algorithm
- An MPI communicator is a name space, a subset of processes that communicate
- Messages remain within their communicator
- A process may be a member of more than one communicator



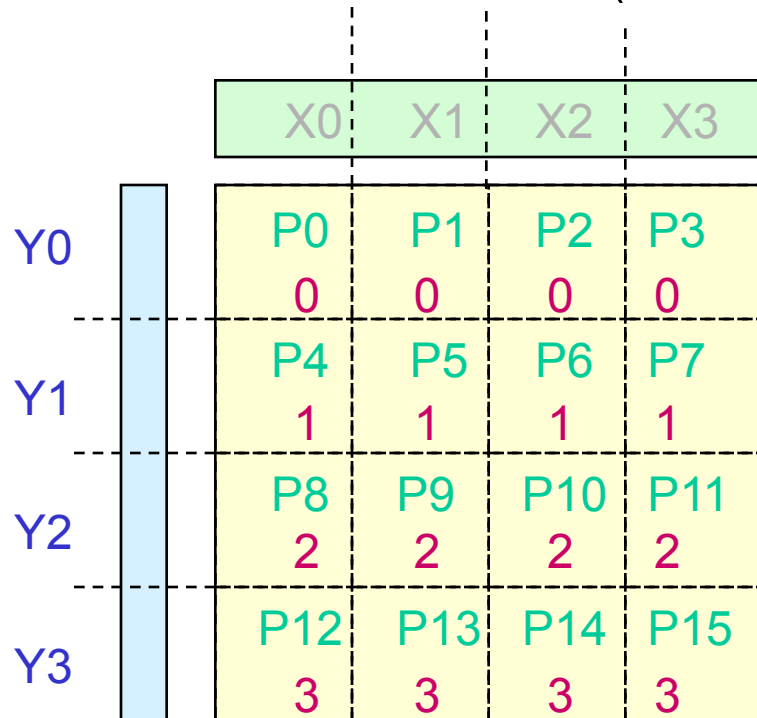
Creating the communicators

- Create a row communicator $\text{key} = \text{myRank} \div \sqrt{P}$

```
MPI_Comm rowComm;
```

```
MPI_Comm_split( MPI_COMM_WORLD,  
               myRank /  $\sqrt{P}$ , myRank, &rowComm);
```

```
MPI_Comm_rank(rowComm, &myRow);
```



- Each process obtains a new communicator
- Each process' rank relative to the new communicator
- Rank applies to the respective communicator only
- Ordered according to **myRank**

More on Comm_split

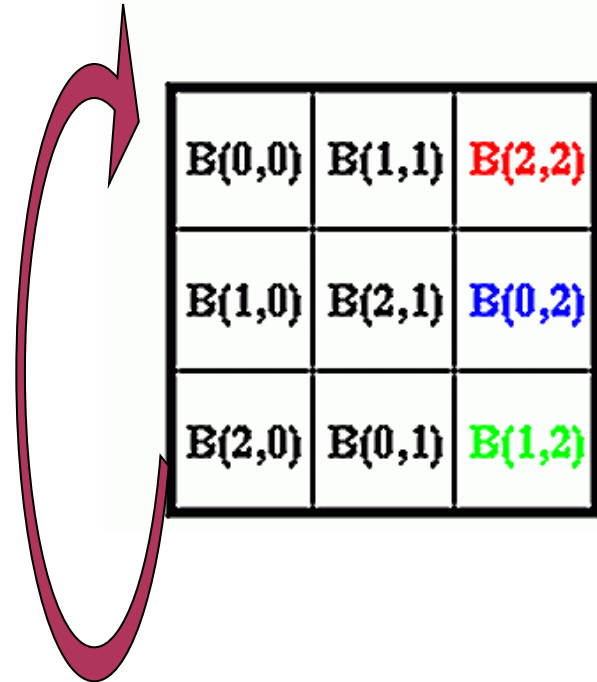
**MPI_Comm_split(MPI_Comm comm, int splitKey,
int rankKey, MPI_Comm* newComm)**

- Ranks assigned arbitrarily among processes sharing the same **rankKey** value
- May exclude a process by passing the constant **MPI_UNDEFINED** as the **splitKey**
- Return a special **MPI_COMM_NULL** communicator
- If a process is a member of several communicators, it will have a rank within each one

Circular shift

- Communication with columns (and rows)

$p(0,0)$	$p(0,1)$	$p(0,2)$
$p(1,0)$	$p(1,1)$	$p(1,2)$
$p(2,0)$	$p(2,1)$	$p(2,2)$

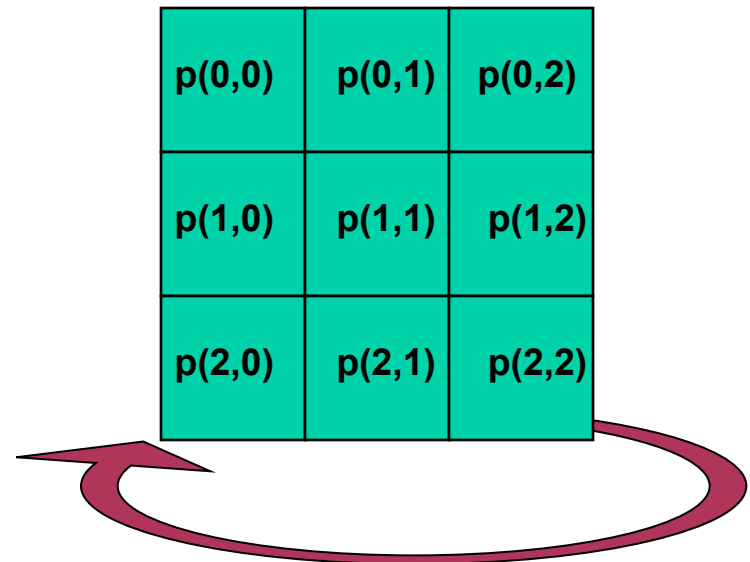


Circular shift

- Communication with columns (and rows)

```
MPI_Comm_rank(rowComm,&myidRing);  
MPI_Comm_size(rowComm,&nodesRing);  
int next = (myidRng + 1 ) % nodesRing;  
MPI_Send(&X,1,MPI_INT,next,0, rowComm);  
MPI_Recv(&XR,1,MPI_INT,  
        MPI_ANY_SOURCE,  
        0, rowComm, &status);
```

- Processes 0, 1, 2 in one communicator because they share the same key value (0)
- Processes 3, 4, 5 are in another (key=1), and so on



Today's lecture

- Managing communicators with Cannon's Algorithm
- **Gather Scatter**

Collective communication

- Basic collectives seen so far
 - Broadcast: distribute data from a designated root process to all the others
 - Reduce: combine data from all processes returning the result to the root process
 - Will revisit these
- Other Useful collectives
 - Scatter/gather
 - All to all
 - Allgather
- Diverse applications
 - Sorting
 - Fast Fourier Transform

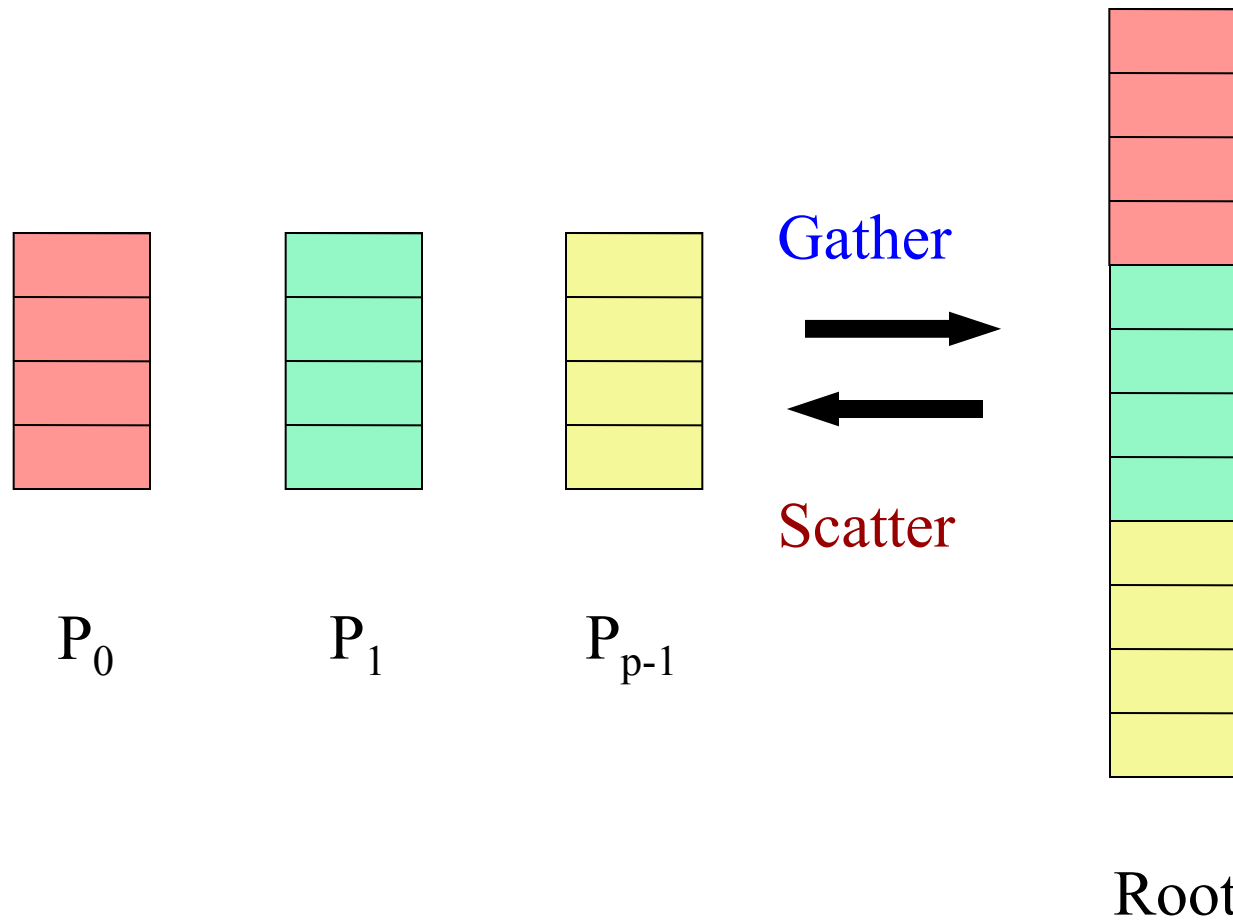
Underlying assumptions

- Fast interconnect structure
 - All nodes are equidistant
 - Single-ported, bidirectional links
- Communication time is $\alpha + \beta n$ in the absence of contention
 - Determined by bandwidth β^{-1} for long messages
 - Dominated by latency α for short messages

Inside MPI-CH

- Tree like algorithm to broadcast the message to blocks of processes, and a linear algorithm to broadcast the message within each block
- Block size may be configured at installation time
- If there is hardware support (e.g. Blue Gene), then it is given responsibility to carry out the broadcast
- Polyalgorithms apply different algorithms to different cases, i.e. long vs. short messages, different machine configurations
- We'll use hypercube algorithms to simplify the special cases when $P=2^k$, k an integer

Scatter/Gather



Scatter

- Simple linear algorithm
 - Root processor sends a chunk of data to all others
 - Reasonable for long messages

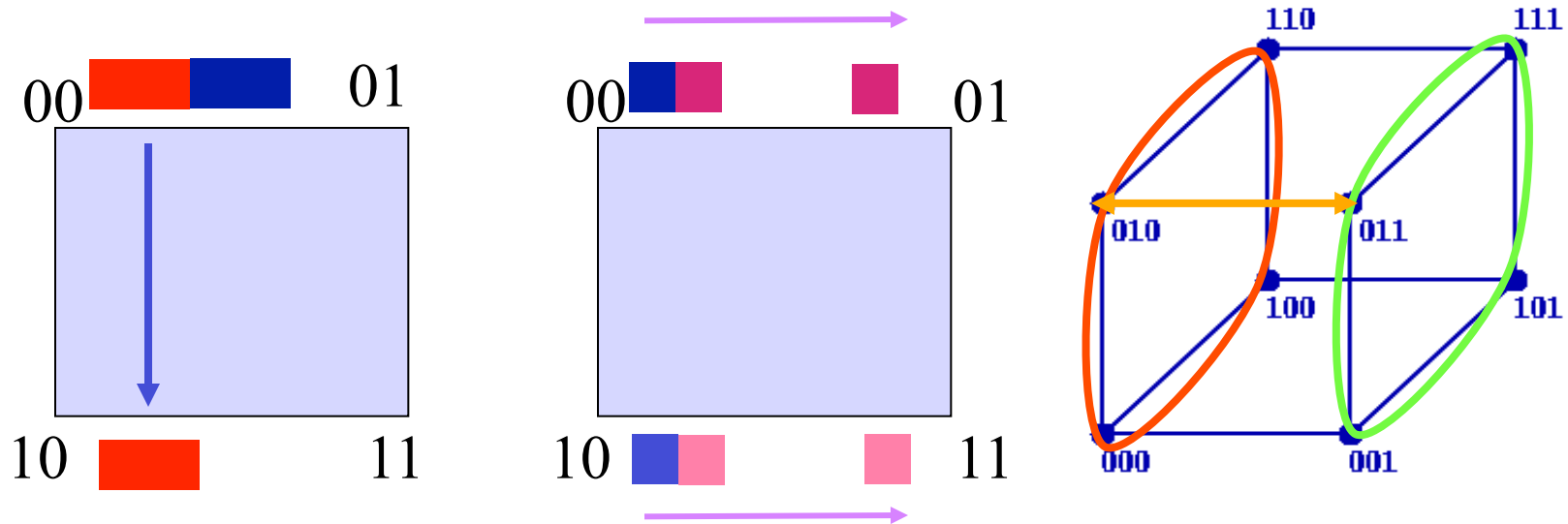
$$(p-1)\alpha + \frac{p-1}{p}n\beta$$

- Similar approach taken for Reduce and Gather
- For short messages, we need to reduce the complexity of the latency (α) term

Minimum spanning tree algorithm

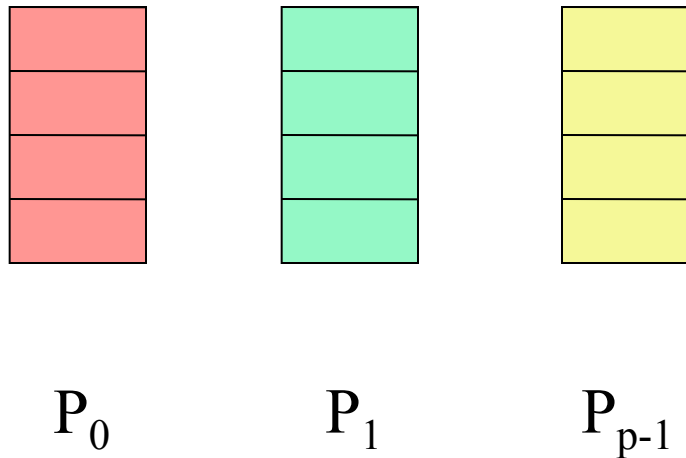
- Recursive hypercube-like algorithm with $\lceil \log P \rceil$ steps
 - Root sends half its data to process $(\text{root} + p/2) \bmod p$
 - Each receiver acts as a root for corresponding half of the processes
 - MST: organize communication along edges of a minimum-spanning tree covering the nodes
- Requires $O(n/2)$ temp buffer space on intermediate nodes
- Running time:

$$\lceil \lg P \rceil \alpha + \frac{p-1}{p} n \beta$$

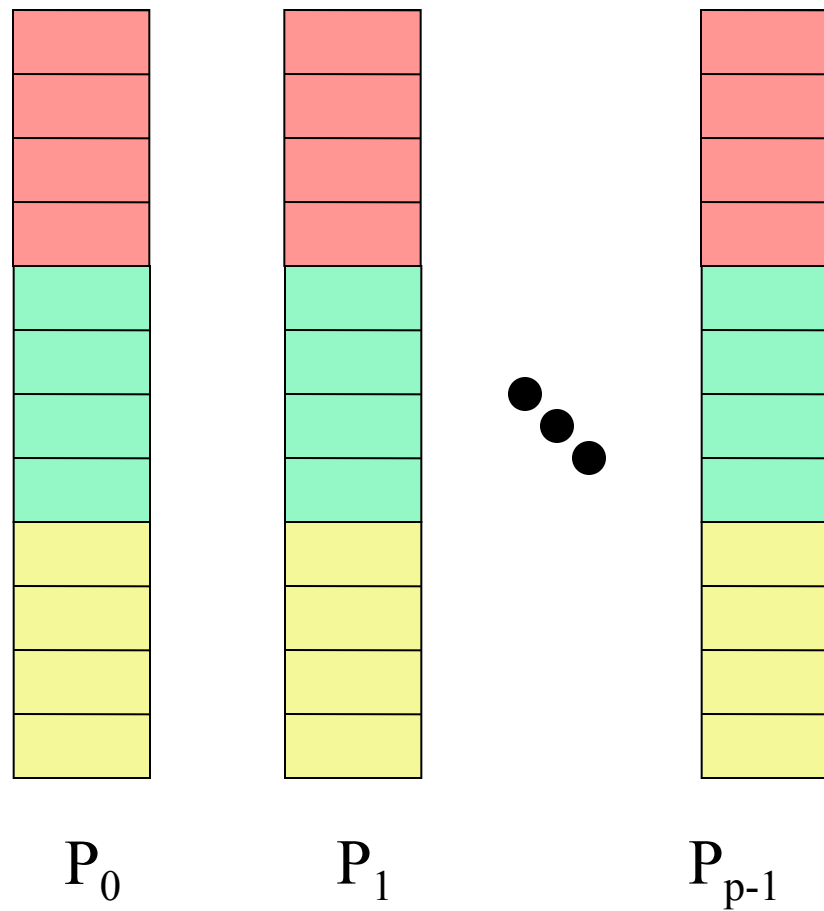


AllGather

- Equivalent to a gather followed by a broadcast
- All processors accumulate a chunk of data from all the others



AllGather



AllGather

- Use the all to all recursive doubling algorithm
- For P a power of two, running time is

$$[\lg P]\alpha + \frac{p-1}{p}n\beta$$

Revisiting Broadcast

- P may not be a power of 2
- MPI-CH uses a binomial tree algorithm for short messages (We can use the hypercube algorithm to illustrate the special case of $P=2^k$)
- We use a different algorithm for long messages

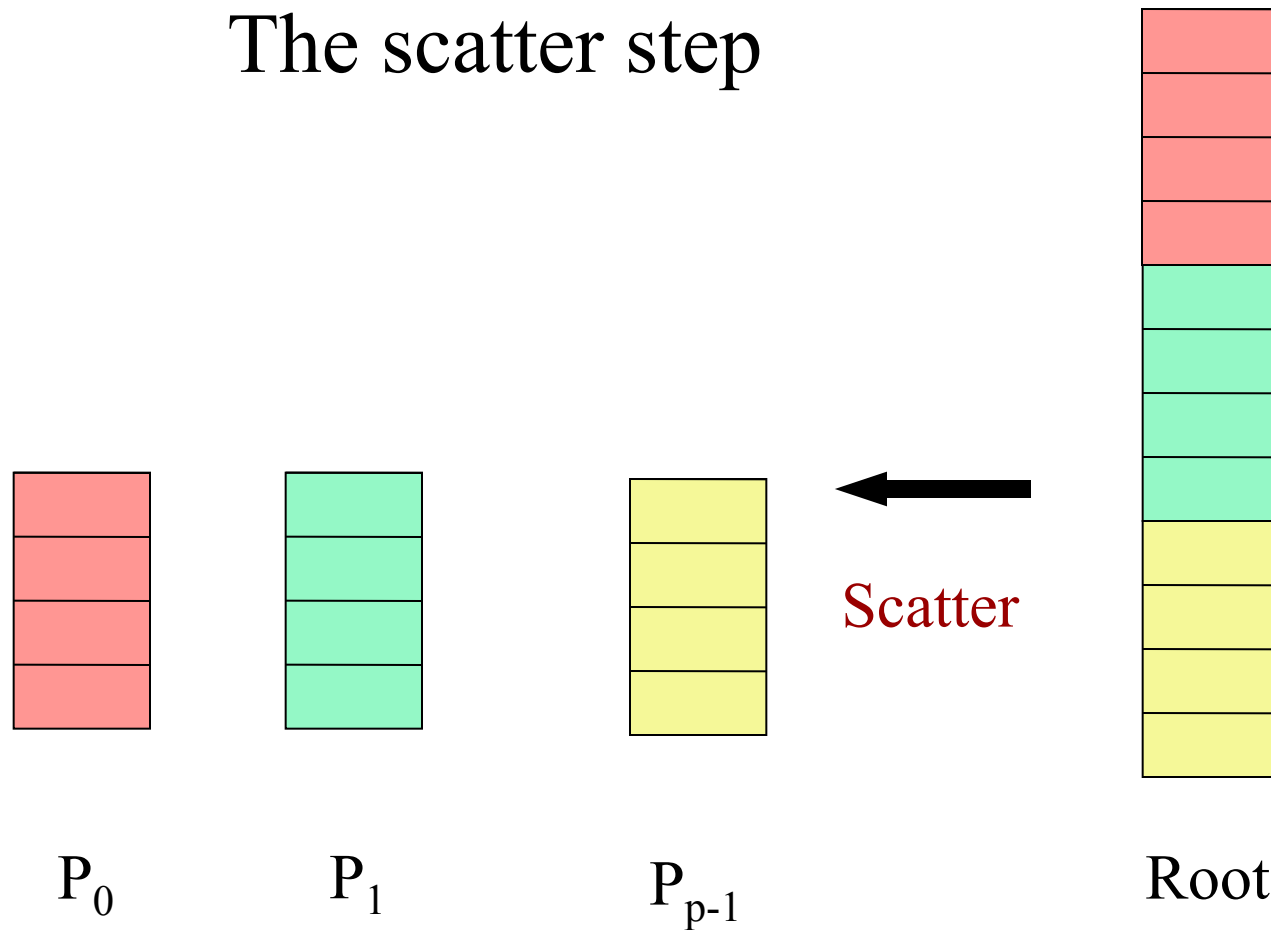
Strategy for long messages

- Based van de Geijn's strategy
- Scatter the data
 - Divide the data to be broadcast into pieces, and fill the machine with the pieces
- Do an Allgather
 - Now that everyone has a part of the entire result, collect on all processors
- Faster than MST algorithm for long messages

$$2 \frac{p-1}{p} n\beta \ll \lceil \lg p \rceil n\beta$$

Algorithm for long messages

The scatter step



Algorithm for long messages

