CSE 6220/CX 4220
Introduction to HPC

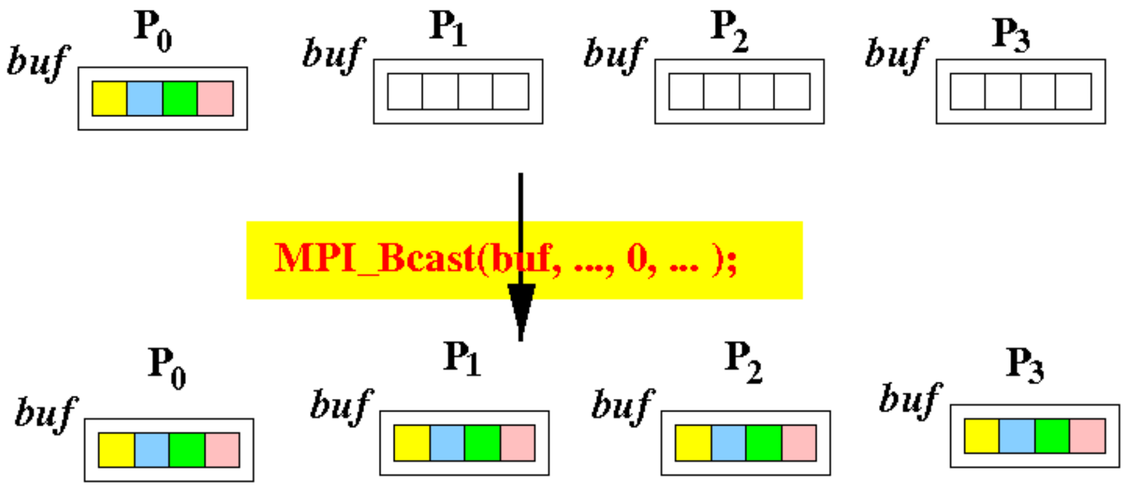
Lecture 9: Communication Primitives

Helen Xu
hxu615@gatech.edu

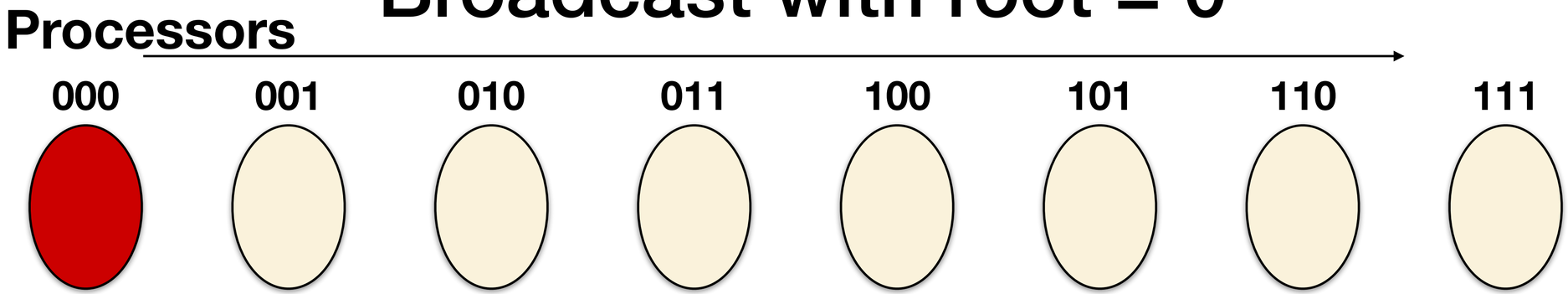
Georgia Tech College of Computing
School of Computational
Science and Engineering

Communication Primitives

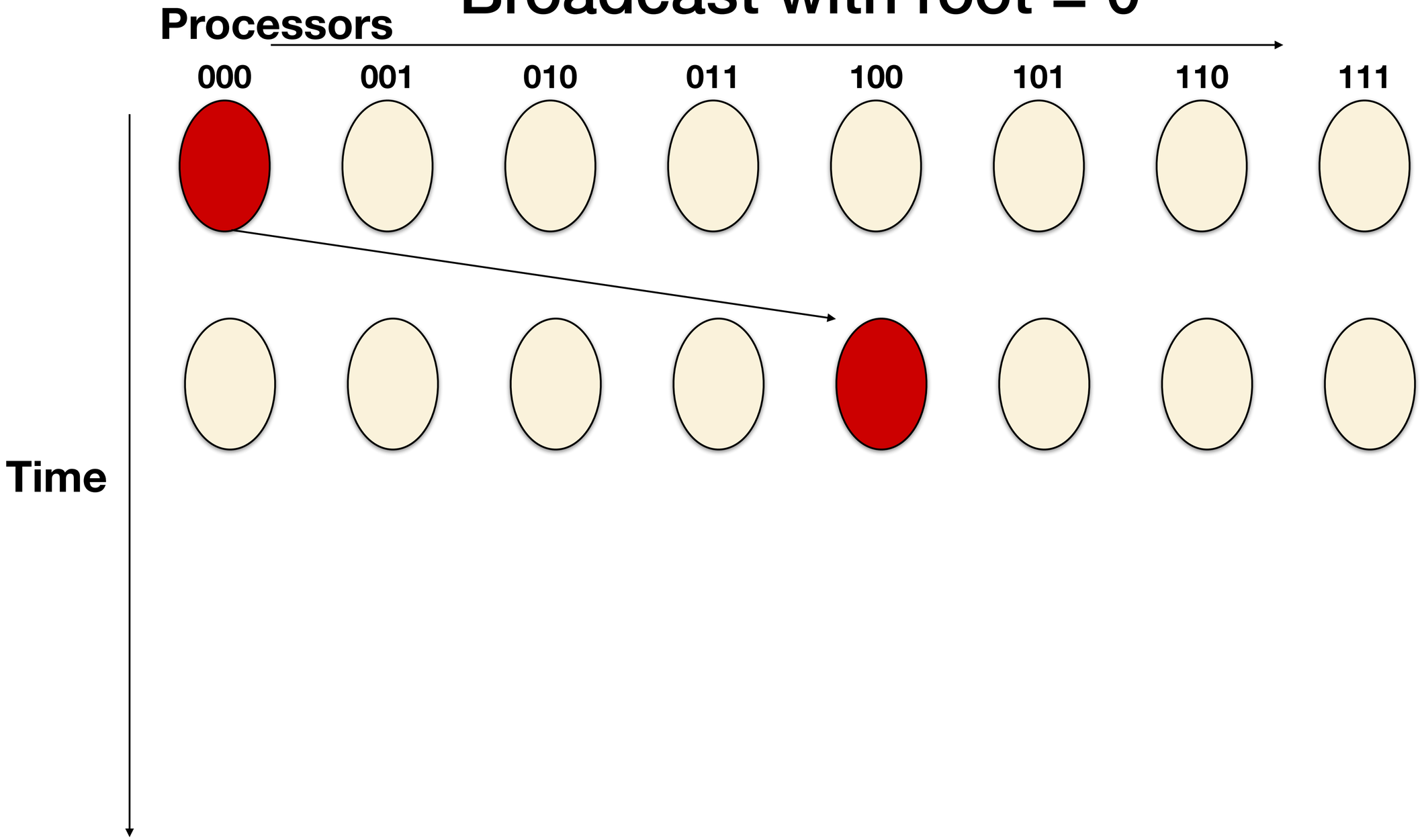
- Building blocks for algorithms that involve communication (sending data)
- Can sometimes include a small amount of computation (e.g., reduce, scan)



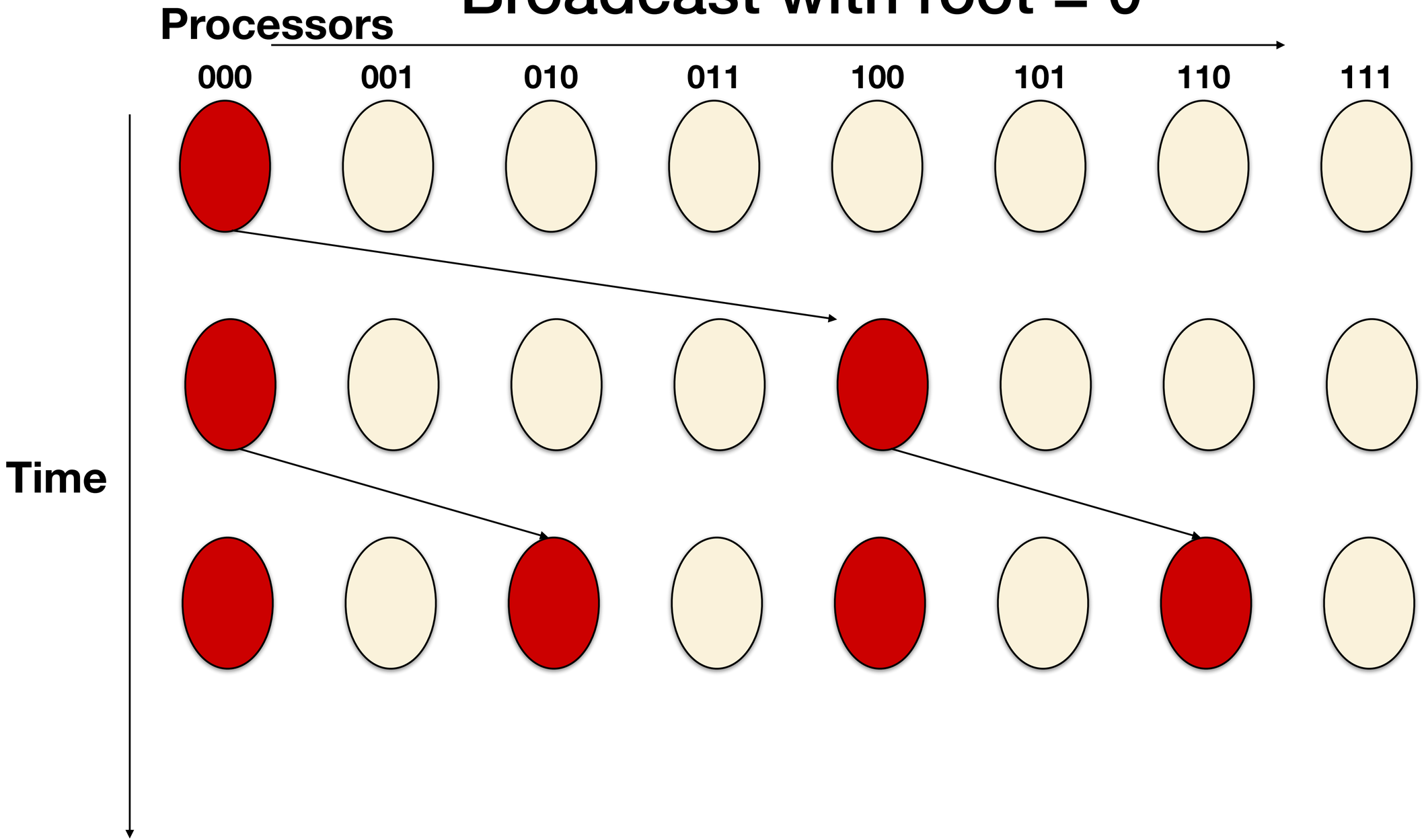
Broadcast with root = 0



Broadcast with root = 0

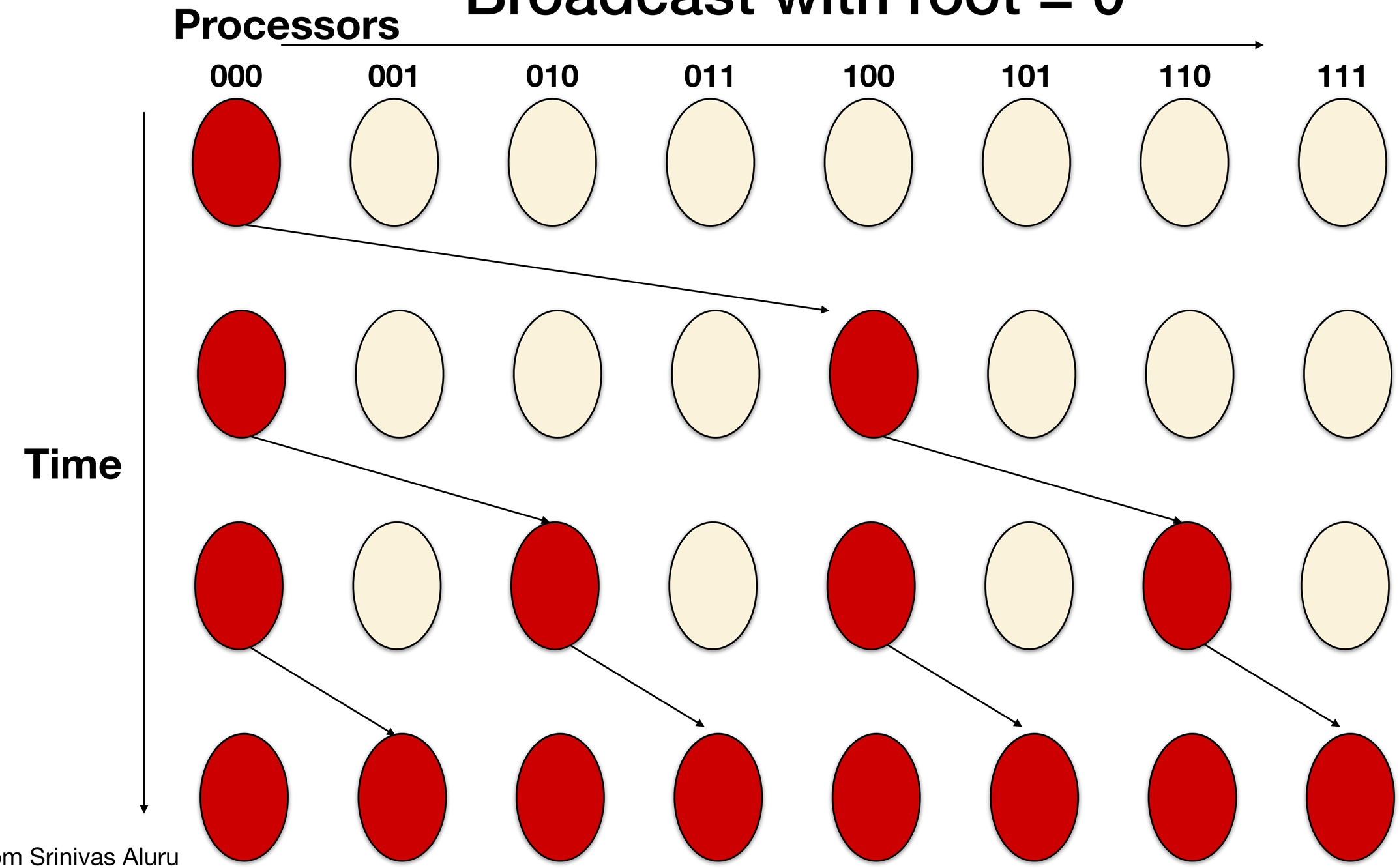


Broadcast with root = 0



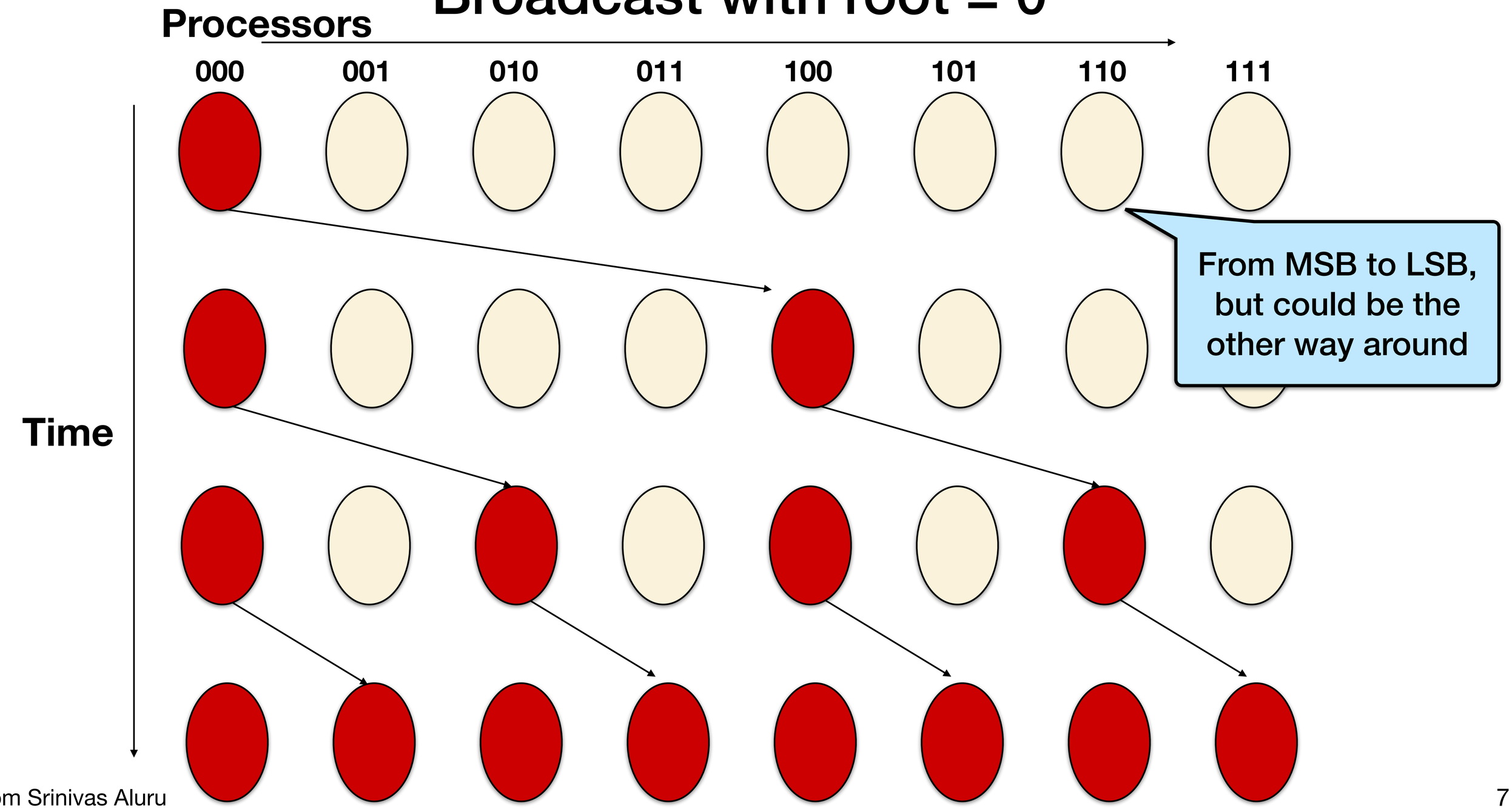
From Srinivas Aluru

Broadcast with root = 0



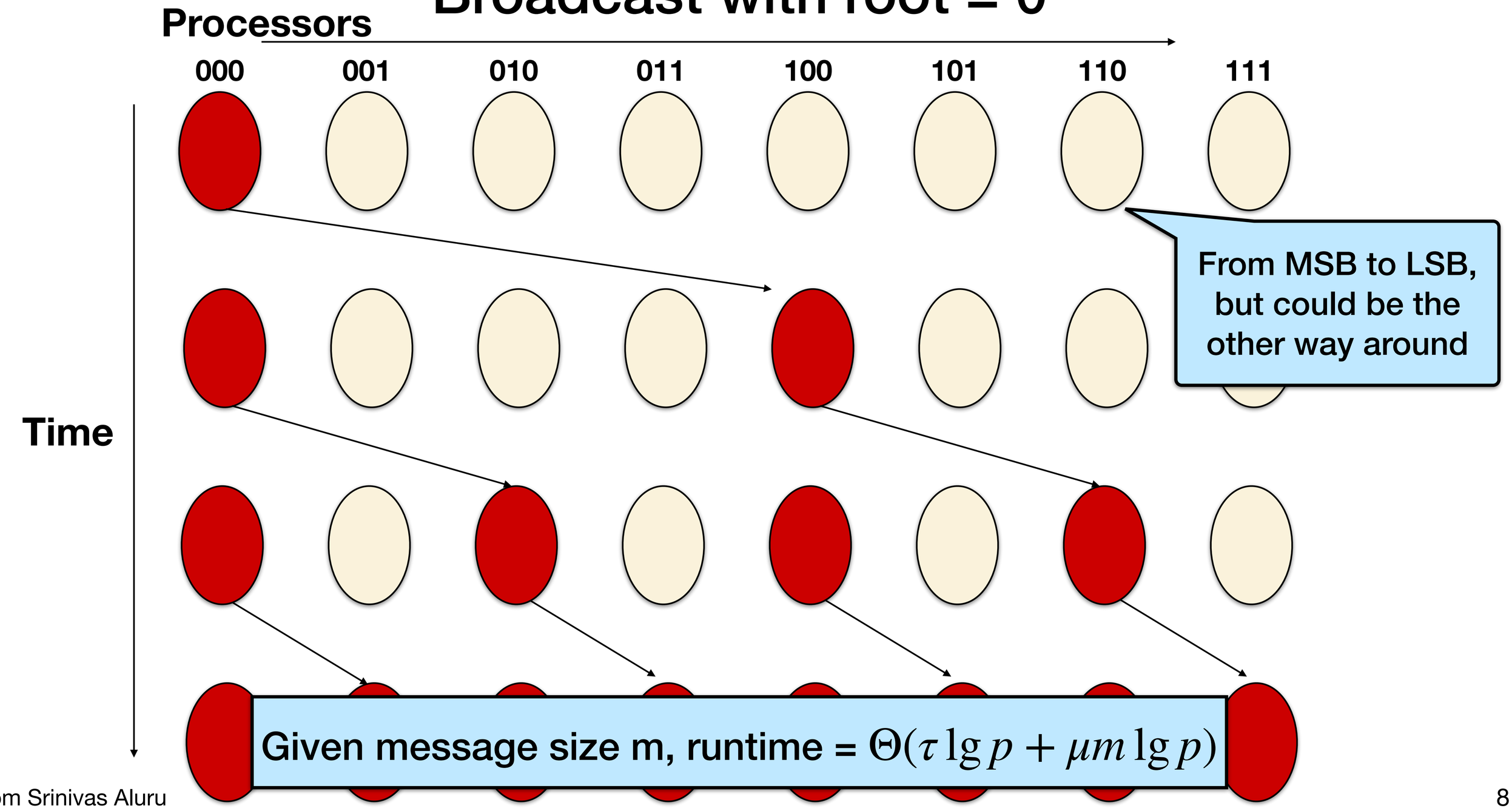
From Srinivas Aluru

Broadcast with root = 0



From Srinivas Aluru

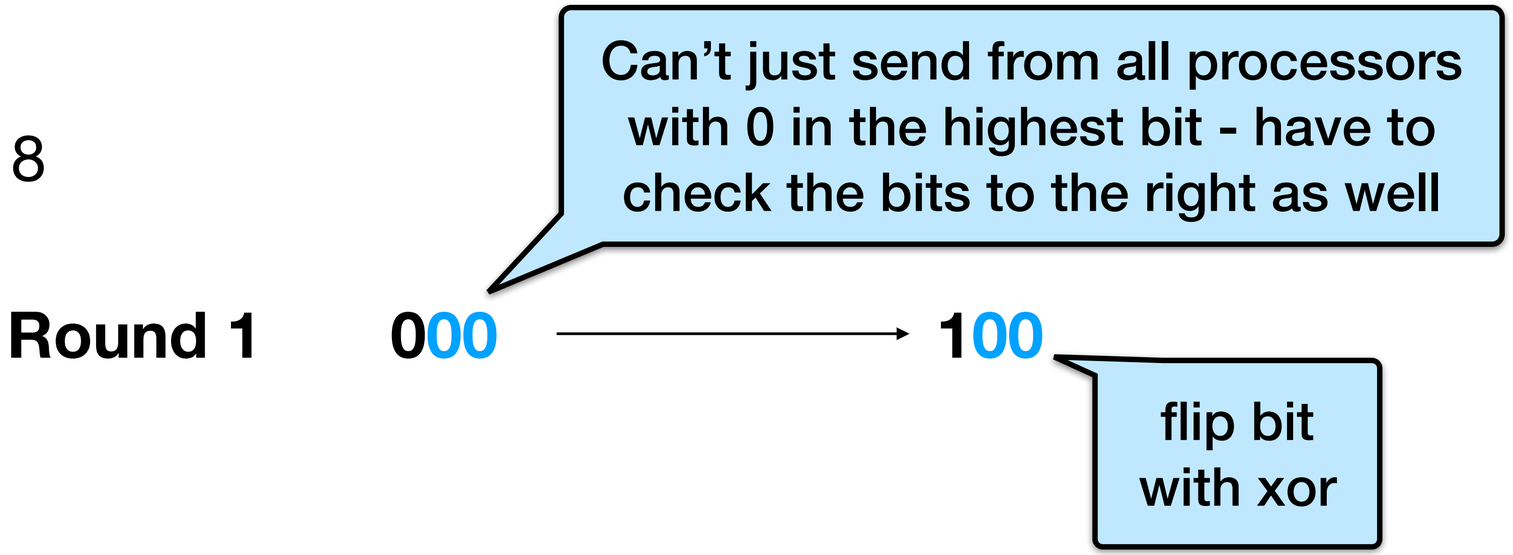
Broadcast with root = 0



From Srinivas Aluru

Broadcast with root = 0

Example: p = 8



Broadcast with root = 0

Example: p = 8

Can't just send from all processors with 0 in the highest bit - have to check the bits to the right as well

Round 1 **000** → **100**

flip bit with xor

To get the bitmask, do left shift then -1

_ _ _ x x x

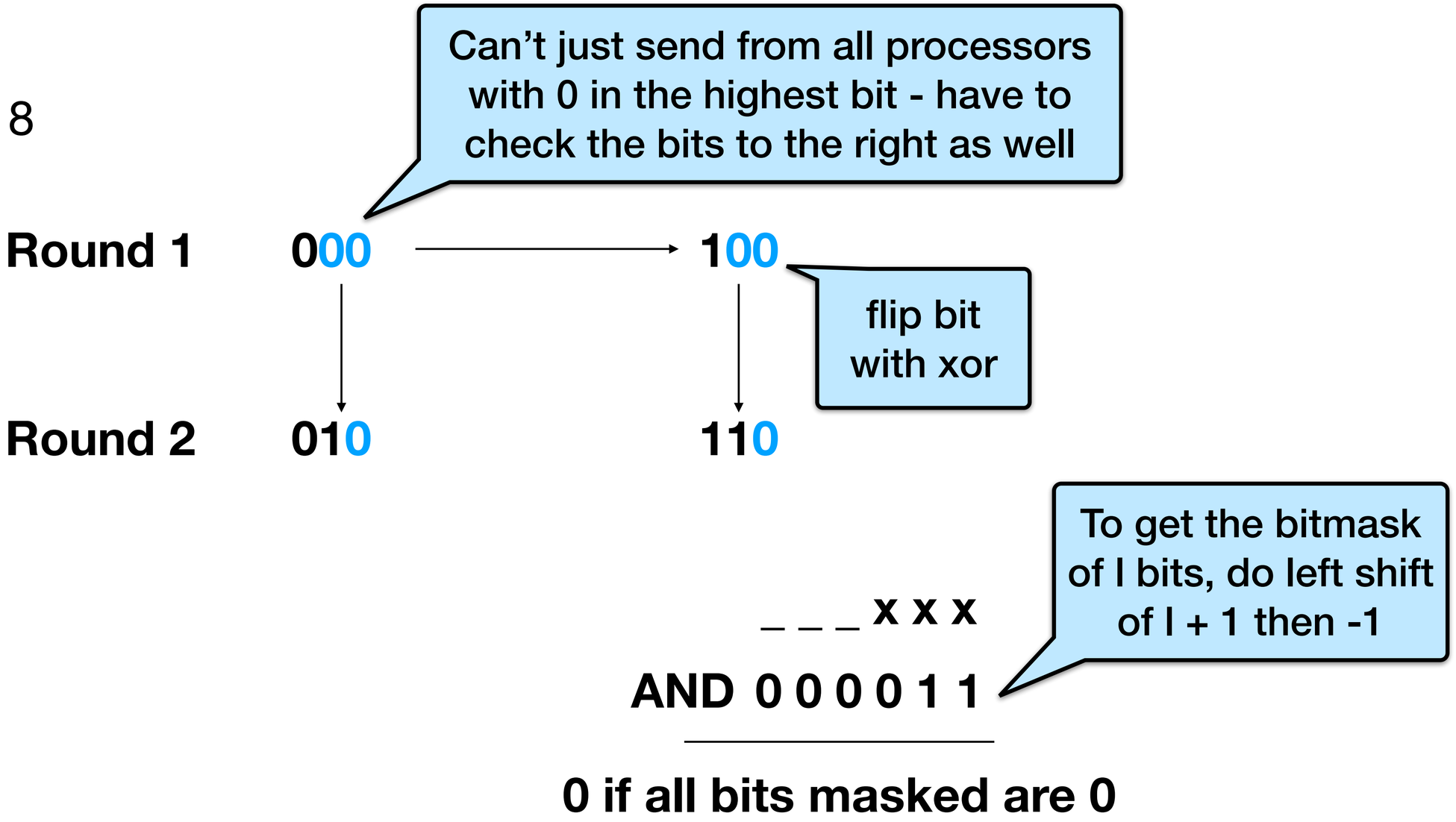
AND 0 0 0 0 1 1

---

0 if all bits masked are 0

Broadcast with root = 0

Example: p = 8



Broadcast with root = 0

Algorithm (for P_{rank})

```
flip ← 1 << (d-1)     // flip is 2^(d-1)
mask ← flip - 1
```

e.g., if d=3
flip = 100b (4)

mask = 11b (3)

Broadcast with root = 0

Algorithm (for P_{rank})

```
flip ← 1 << (d-1)      // flip is 2^(d-1)
mask ← flip - 1
for j=d-1 to 0 do
    if ((rank AND mask) == 0)
        if ((rank AND flip) == 0)
            send x to (rank XOR flip)
        else
            receive x from (rank XOR flip)
    mask ← mask >> 1
    flip ← flip >> 1
endfor
```

e.g., if d=3
flip = 100b (4)

mask = 11b (3)

for j=2, only
000b (0) will send &
100b (4) will receive

From Srinivas Aluru

Broadcast with root = 0

Algorithm (for P_{rank})

```
flip ← 1 << (d-1)      // flip is 2^(d-1)
mask ← flip - 1
for j=d-1 to 0 do
   if ((rank AND mask) == 0)
      if ((rank AND flip) == 0)
         send x to (rank XOR flip)
      else
         receive x from (rank XOR flip)
   mask ← mask >> 1
end
```

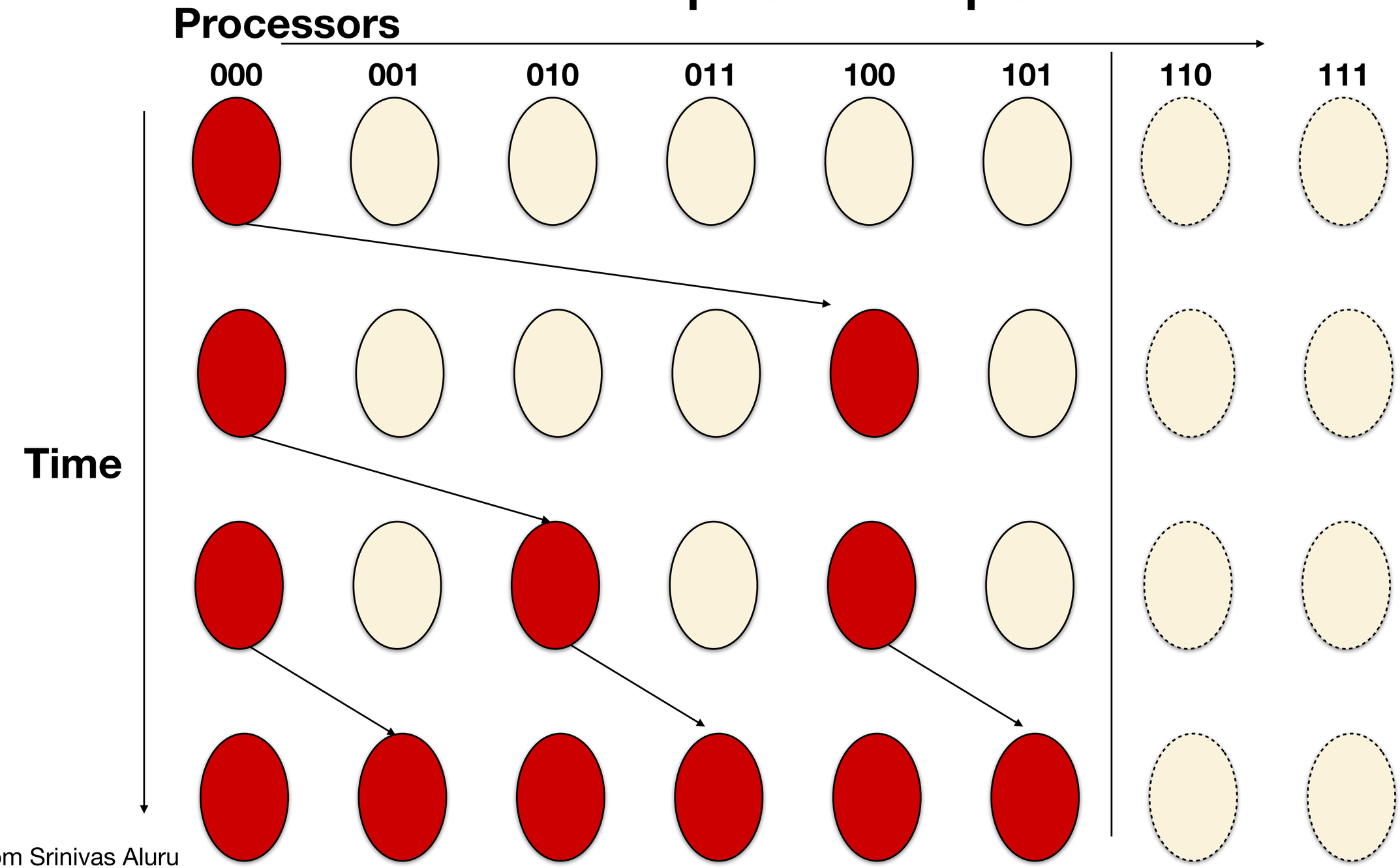
e.g., if d=3
flip = 100b (4)

mask = 11b (3)

for j=2, only
000b (0) will send &
100b (4) will receive

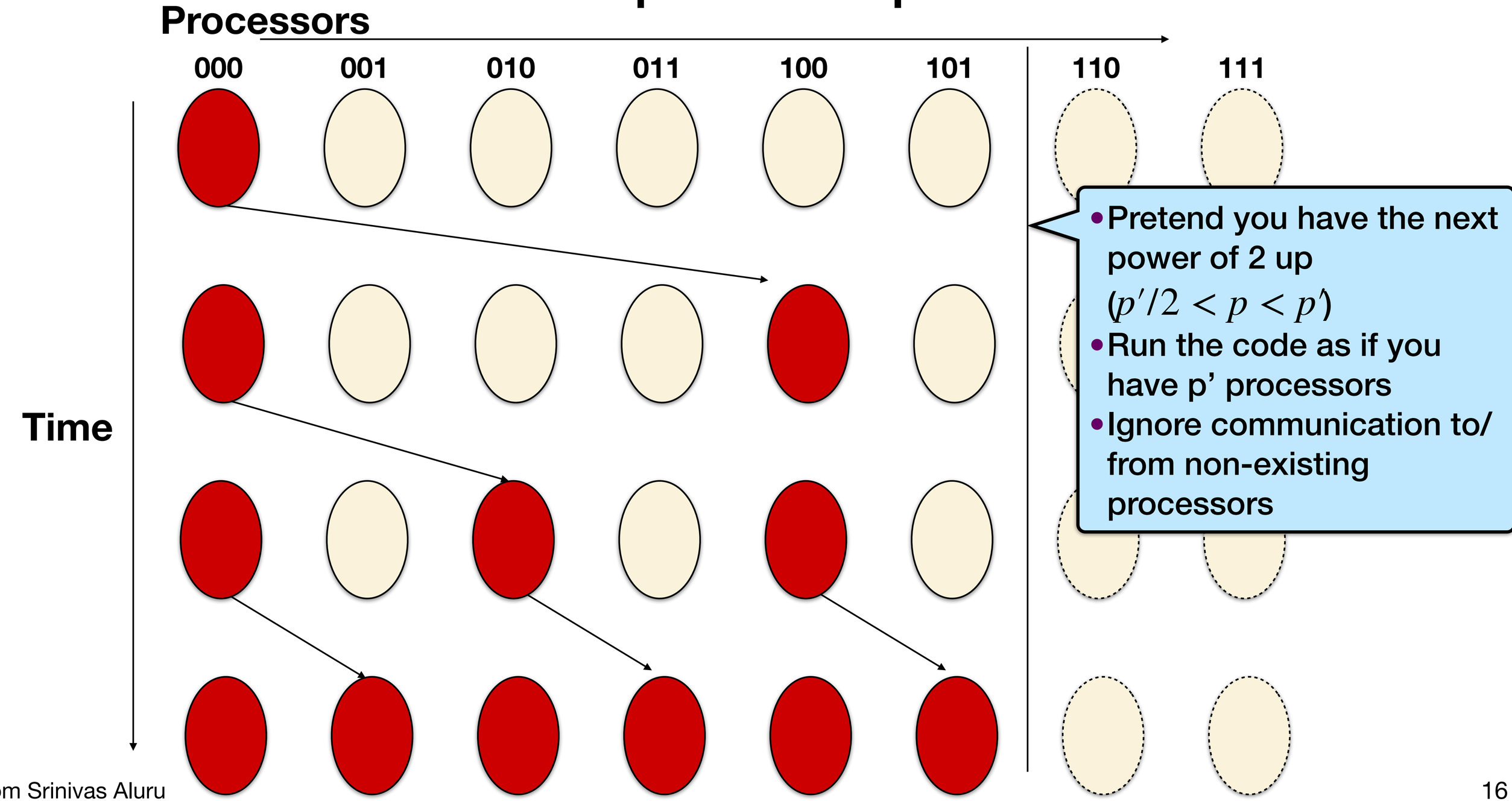
What if the number of processors isn't a power of 2?
What if processor 0 is not the root?

Broadcast if p is not a power of 2



From Srinivas Aluru

Broadcast if p is not a power of 2



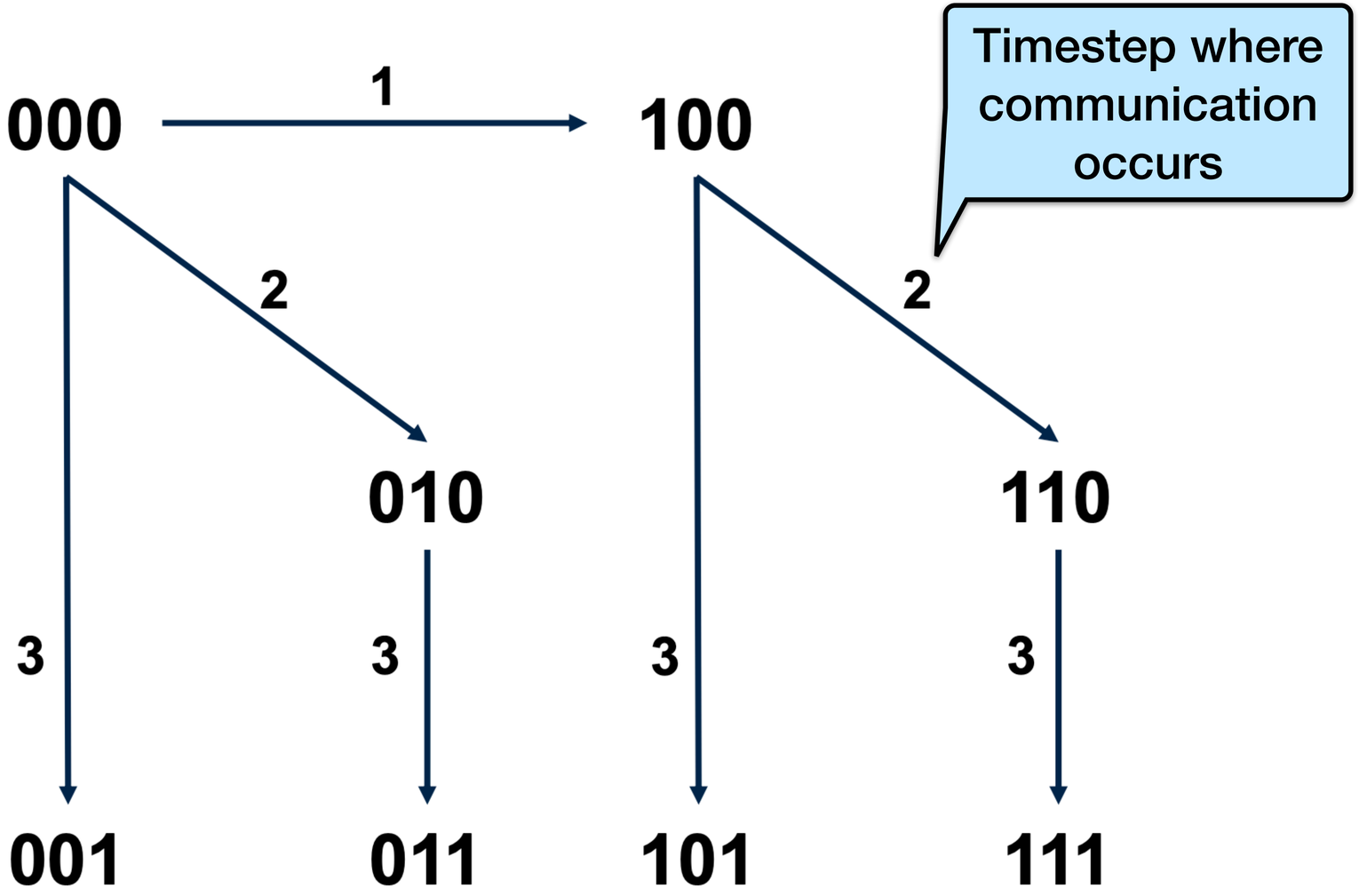
From Srinivas Aluru

What if the root is not processor 0?

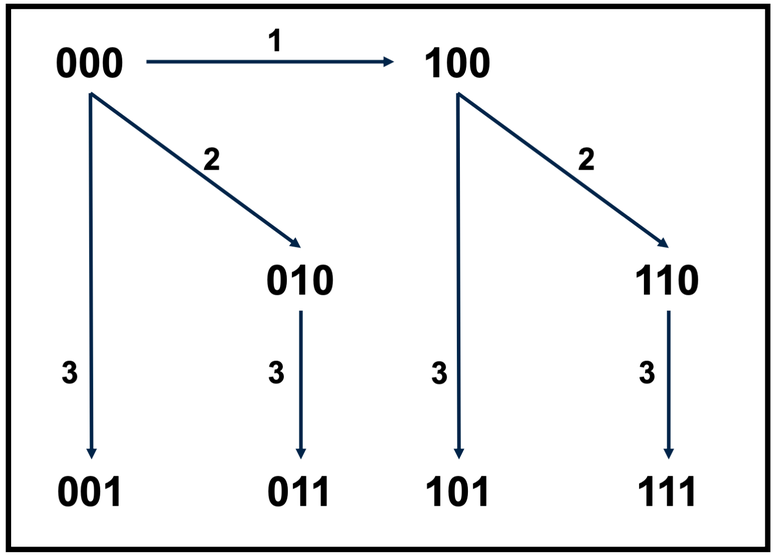
- Easiest way: root sends to processor 0. One communication in $\Theta(\tau + \mu m)$, which does not affect the theoretical complexity.

- But we can do even better - with no additional communication.

Broadcast Tree



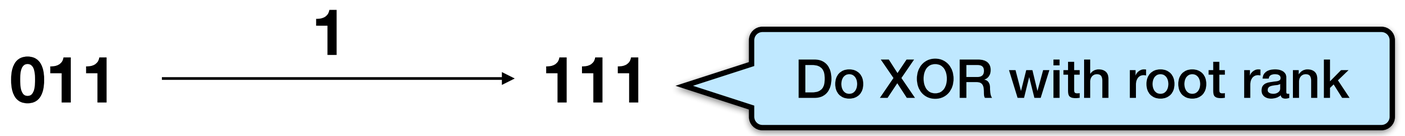
Broadcast Tree



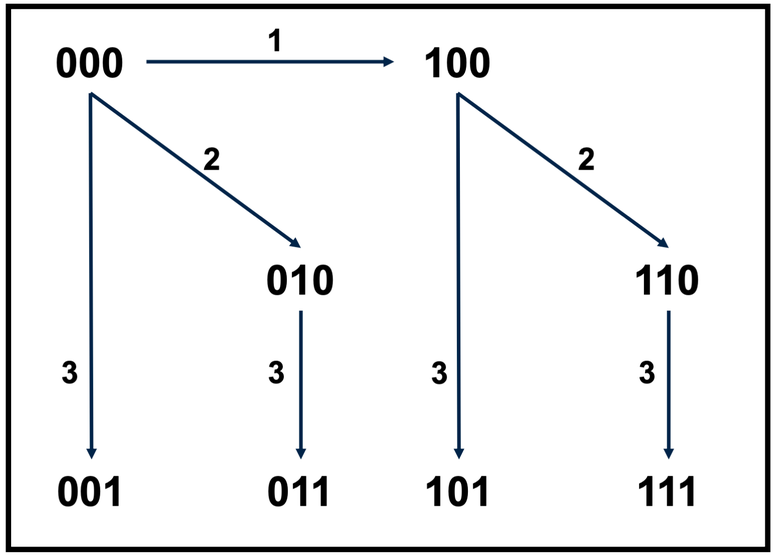
Goal: Make a broadcast tree with any root

- Every rank is represented once in the tree
- Respect hypercubic permutations

Example: Make “011” as root



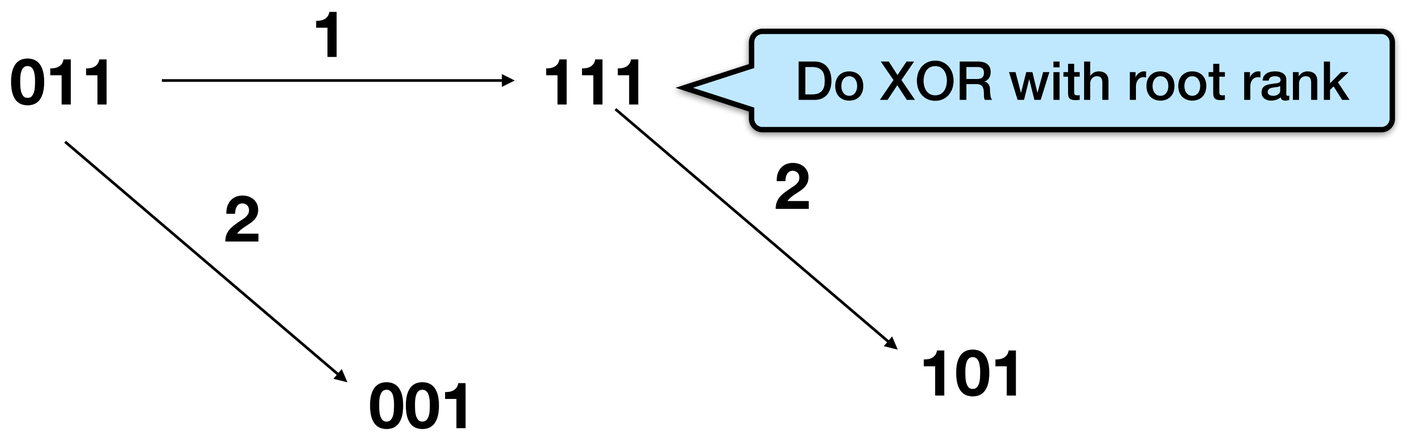
Broadcast Tree



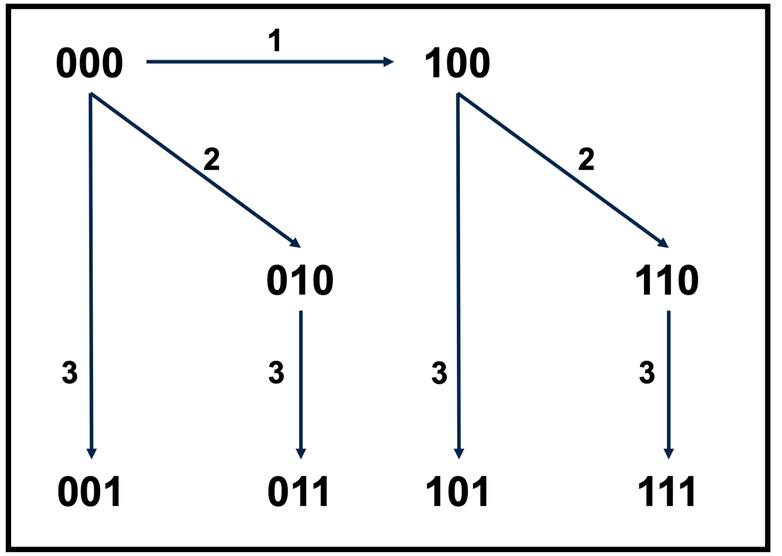
Goal: Make a broadcast tree with any root

- Every rank is represented once in the tree
- Respect hypercubic permutations

Example: Make “011” as root



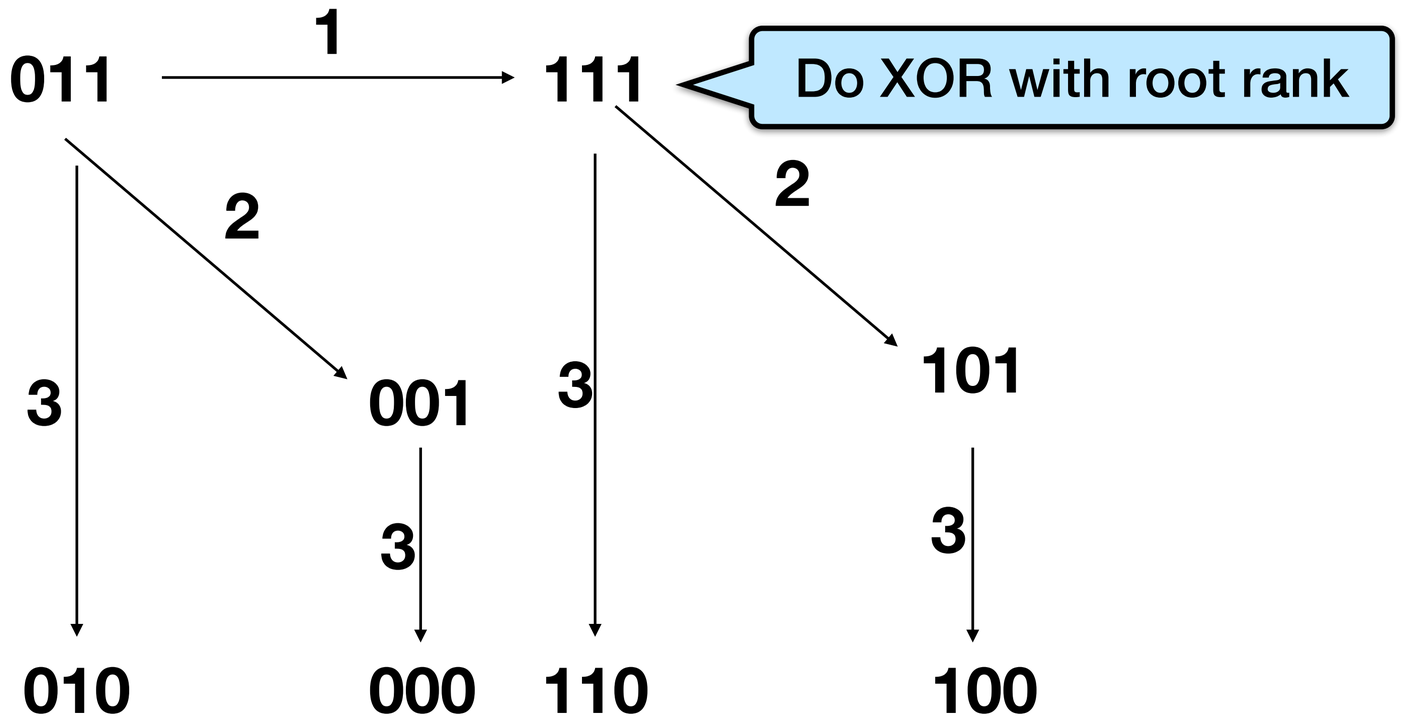
Broadcast Tree



Goal: Make a broadcast tree with any root

- Every rank is represented once in the tree
- Respect hypercubic permutations

Example: Make “011” as root

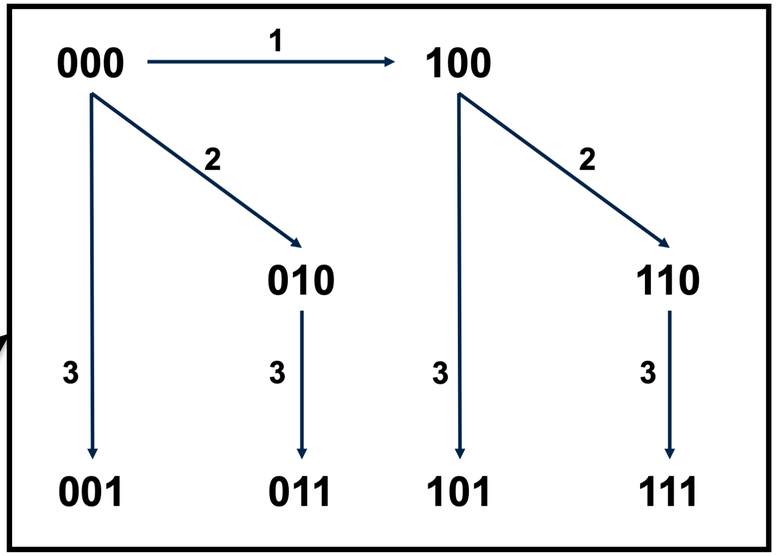


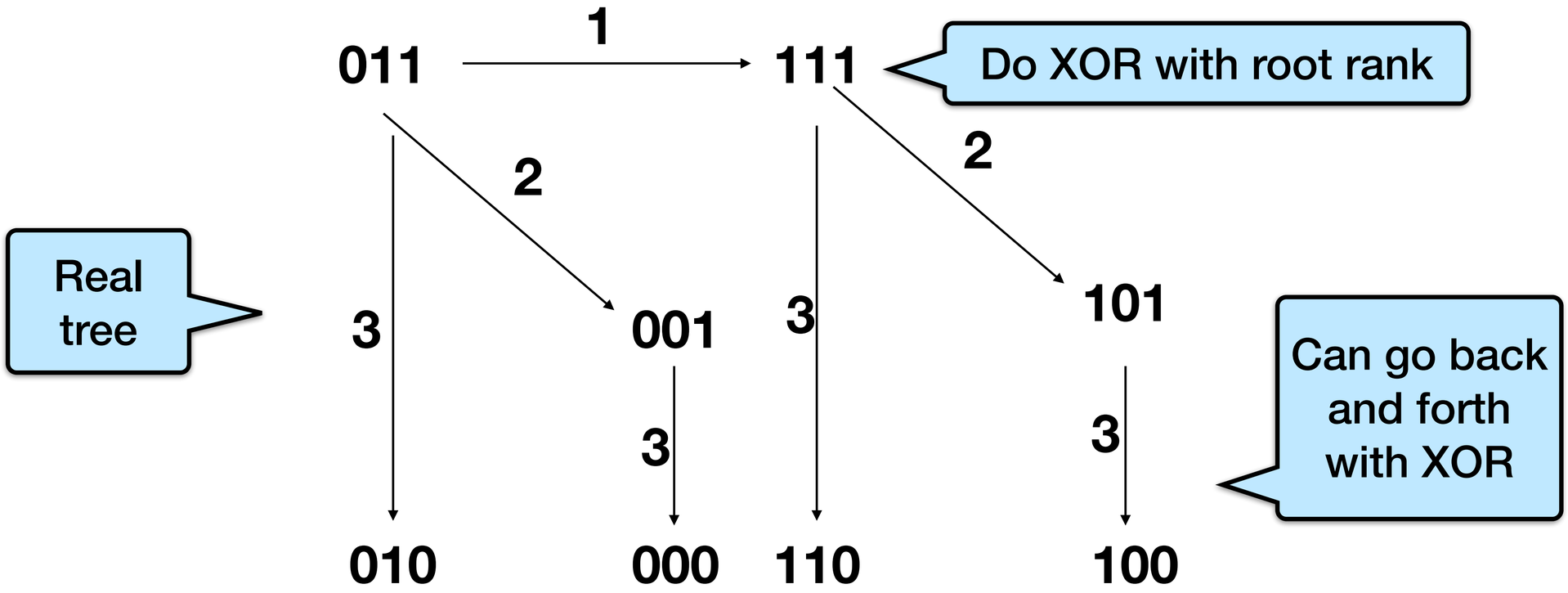
Broadcast Tree

Goal: Make a broadcast tree with any root

- Every rank is represented once in the tree
- Respect hypercubic permutations

Example: Make “011” as root





Broadcast with root ≠ 0

Algorithm (for P_{rank})

```
flip ← 1 << (d-1)
mask ← flip - 1
for j=d-1 to 0 do
   if (((rank XOR root) AND mask) == 0)
```

Convert true rank to rank in the virtual tree

Broadcast with root ≠ 0

Algorithm (for P_{rank})

```
flip ← 1 << (d-1)
mask ← flip - 1
for j=d-1 to 0 do
   if (((rank XOR root) AND mask) == 0)
      if (((rank XOR root) AND flip) == 0)
```

Convert true rank to rank in the virtual tree

Decide whether to send or receive

Broadcast with root ≠ 0

Algorithm (for P_{rank})

```
flip ← 1 << (d-1)
mask ← flip - 1
for j=d-1 to 0 do
   if (((rank XOR root) AND mask) == 0)
      if (((rank XOR root) AND flip) == 0)
         send x to (((rank XOR root) XOR flip) XOR root)
      else
         receive x from (((rank XOR root) XOR flip) XOR root)
   mask ← mask >> 1
   flip ← flip >> 1
endfor
```

Convert true rank to rank in the virtual tree

Decide whether to send or receive

Do communication on the real tree

XOR of the same thing cancels out

Broadcast with root ≠ 0

Algorithm (for P_{rank})

```
flip ← 1 << (d-1)     // flip is 2^(d-1)
mask ← flip - 1
for j=d-1 to 0 do
   if (((rank XOR root) AND mask) == 0)
      if (((rank XOR root) AND flip) == 0)
         send x to (rank XOR flip)
      else
         receive x from (rank XOR flip)
   mask ← mask >> 1
   flip ← flip >> 1
endfor
```

Hypercubic permutation by flipping 1 bit

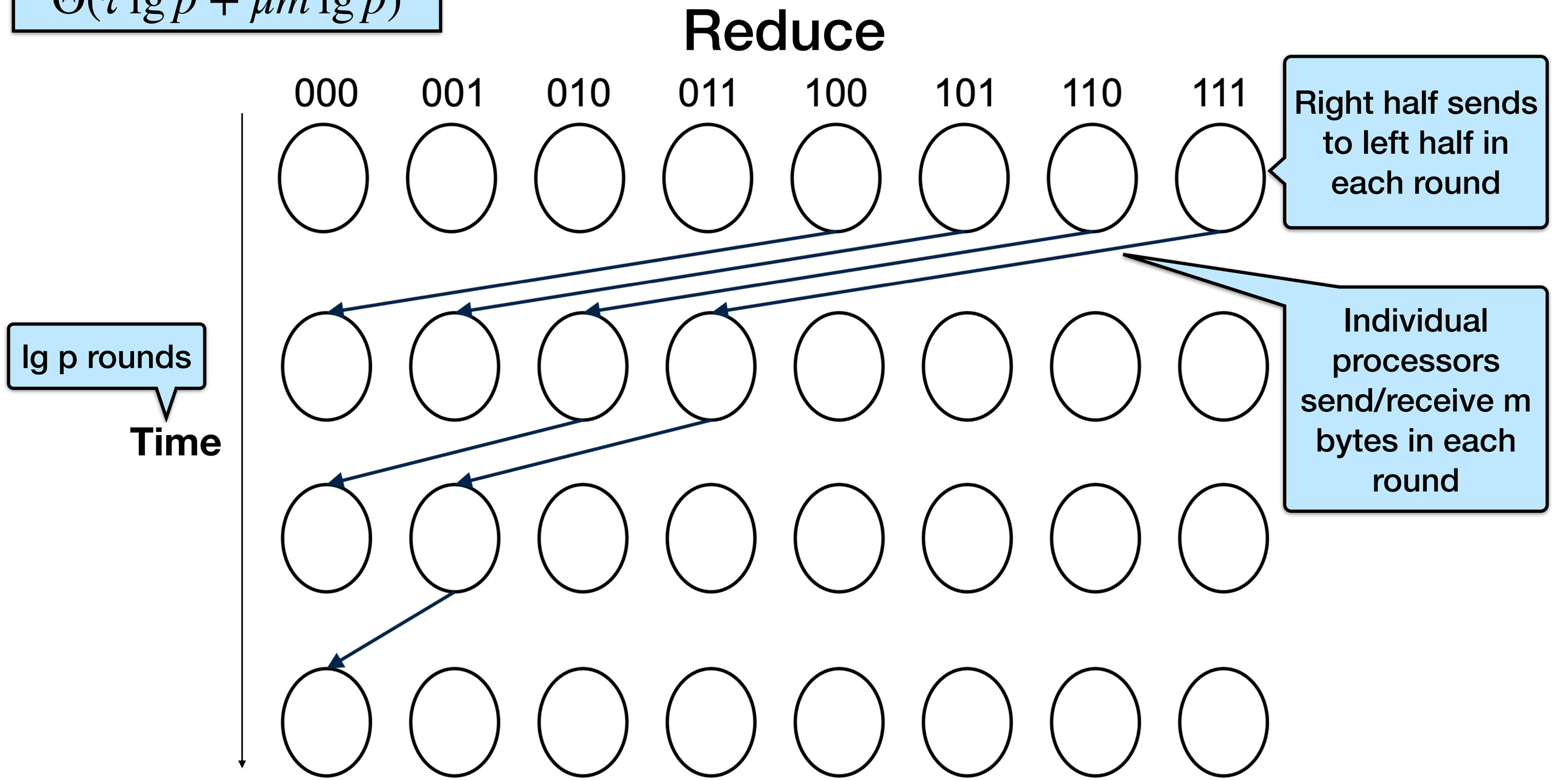
Communication Primitives

- Broadcast
- Reduce - aggregate all elements into one element using some operator
- Allreduce - all ranks get results of reduce
- Scan - prefix sum

All have communication complexity

$$\Theta(\tau \lg p + \mu m \lg p)$$

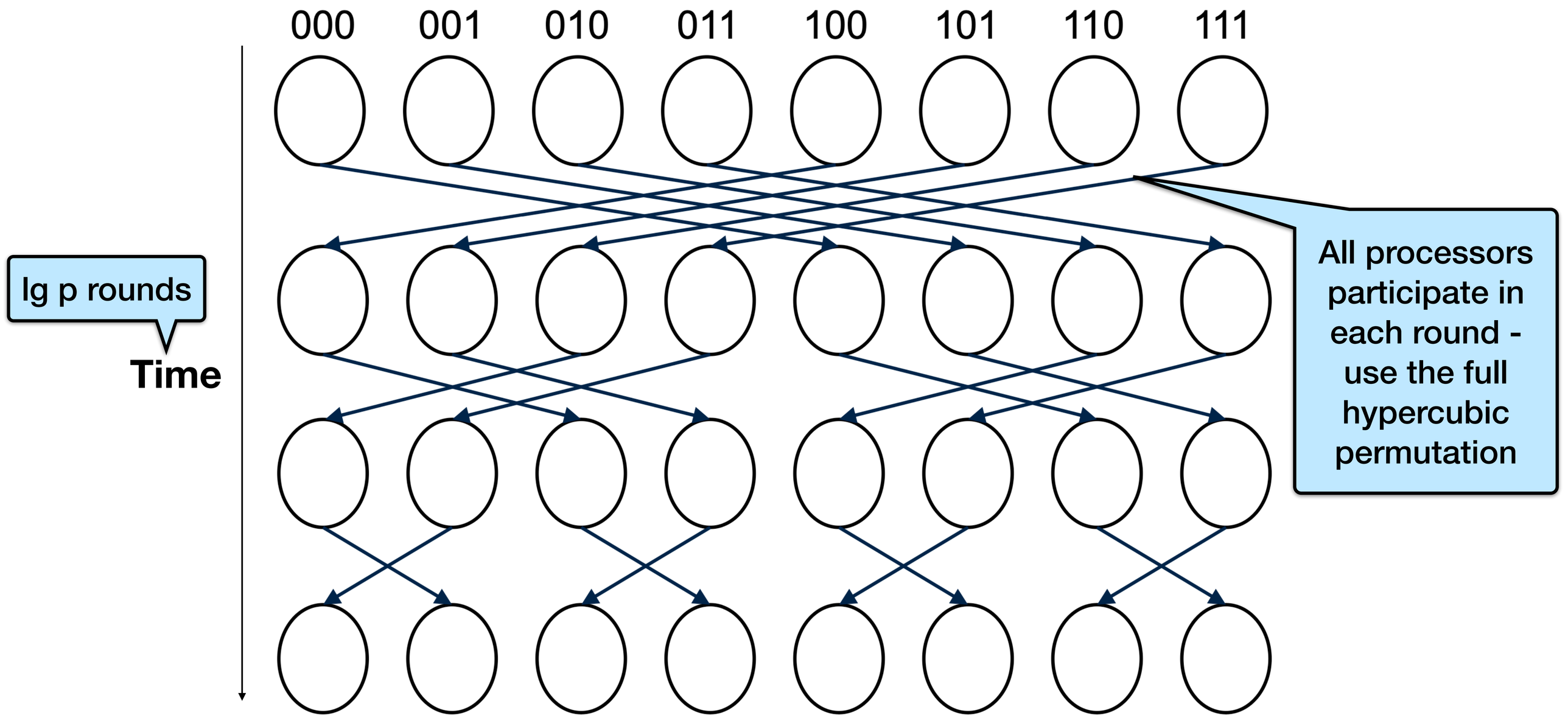
$$\Theta(\tau \lg p + \mu m \lg p)$$



From Srinivas Aluru

$$\Theta(\tau \lg p + \mu m \lg p)$$

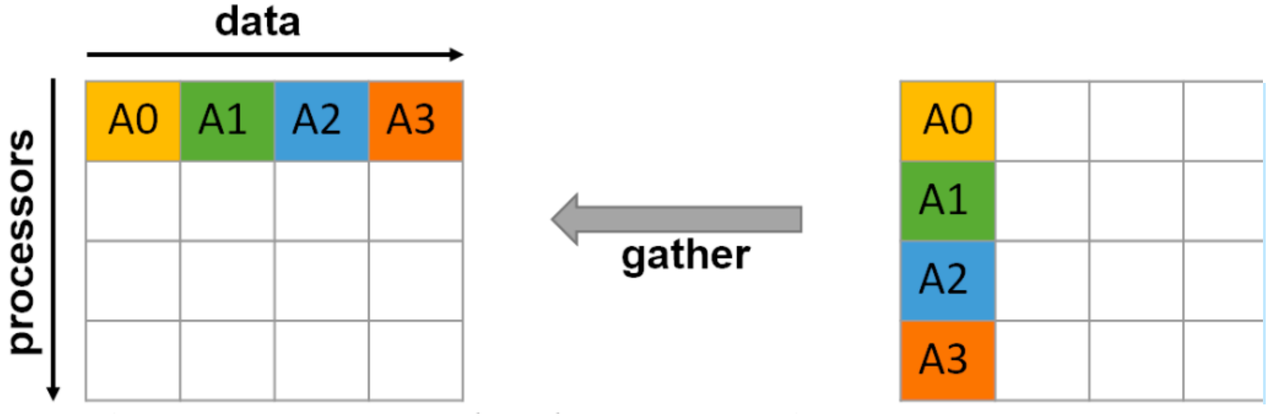
Allreduce



From Srinivas Aluru

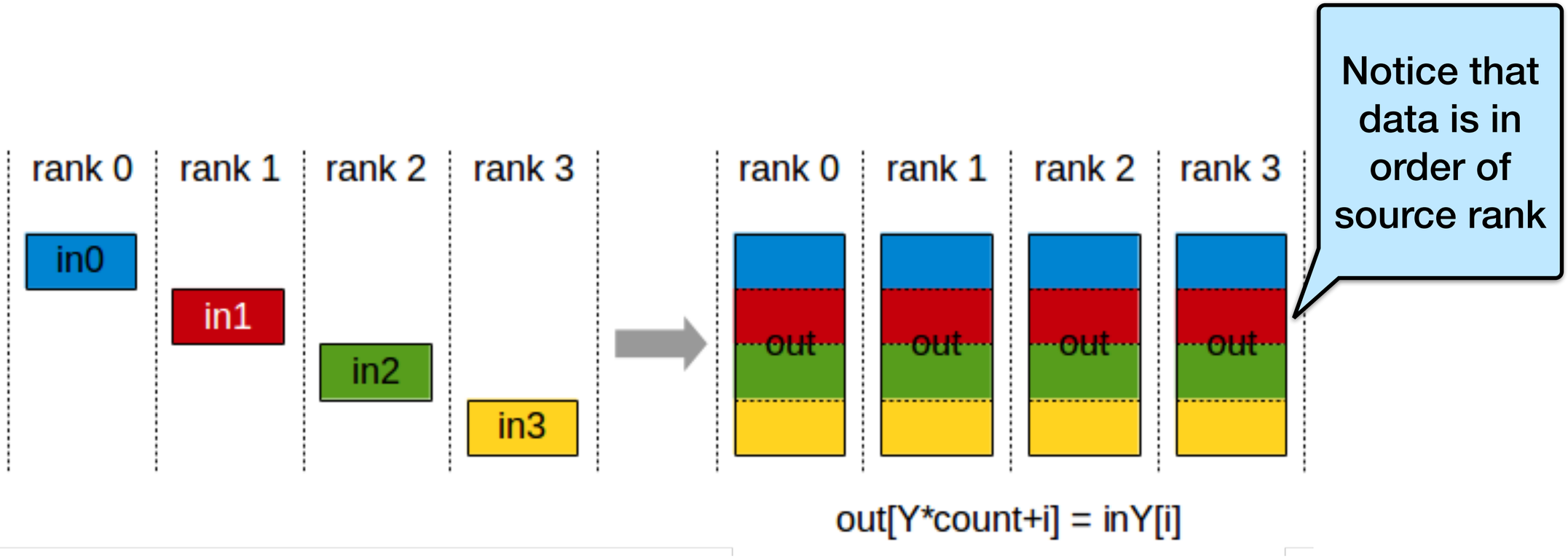
Gather

- Like reduce, but not aggregating them with an operator.
- One processor ends up with all the data that all of the processors are sending.
- Data appears **ordered by source processor** on the root processor.
- Data to be gathered should be much less than the problem size so that the whole algorithm is not worse than serial processing (when accounting for communication).



AllGather

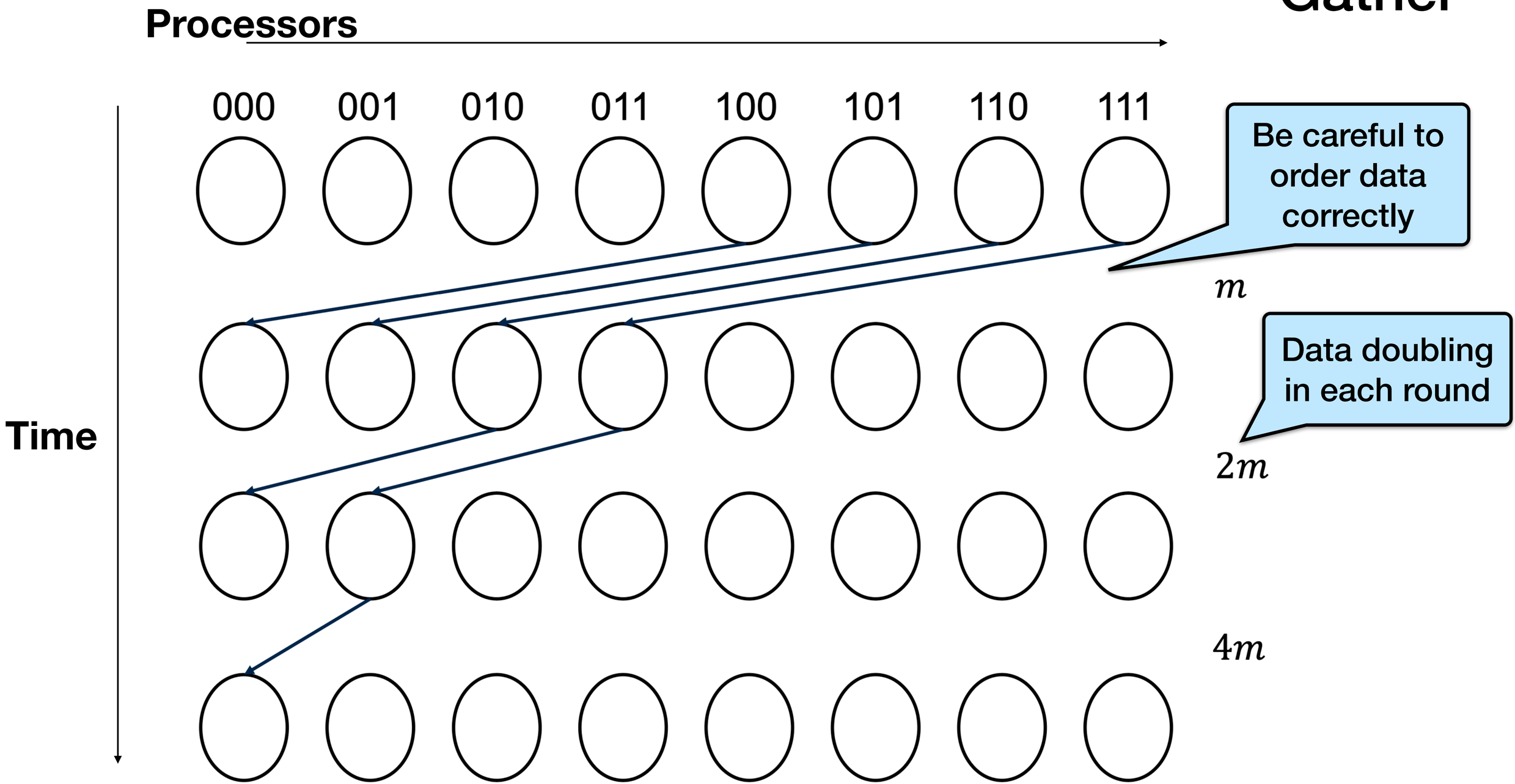
Like gather, but every processor gets the entire answer.



https://docs.nvidia.com/deeplearning/nccl/user-guide/docs/usage/collectives.html

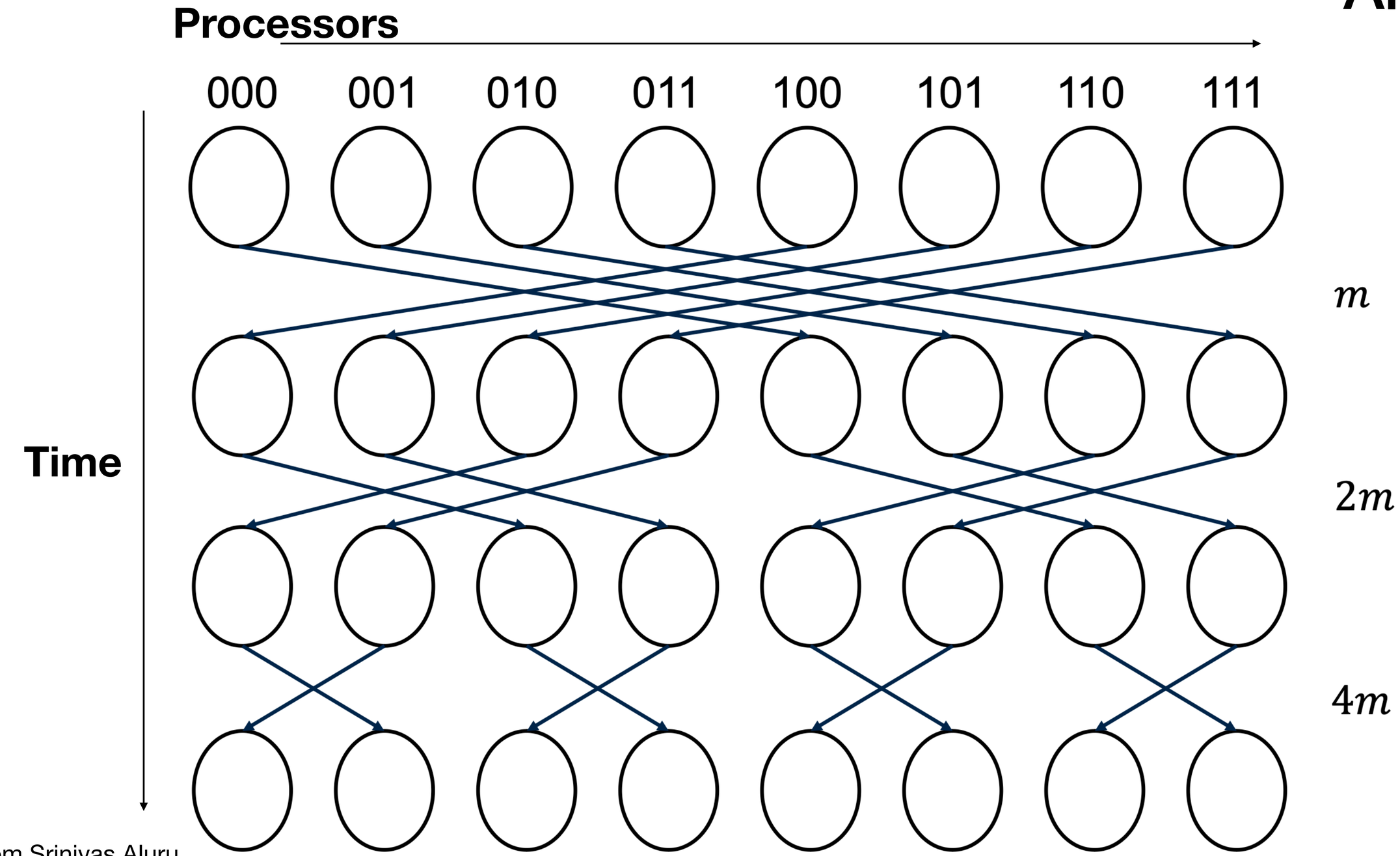
From Srinivas Aluru

Gather



From Srinivas Aluru

AllGather

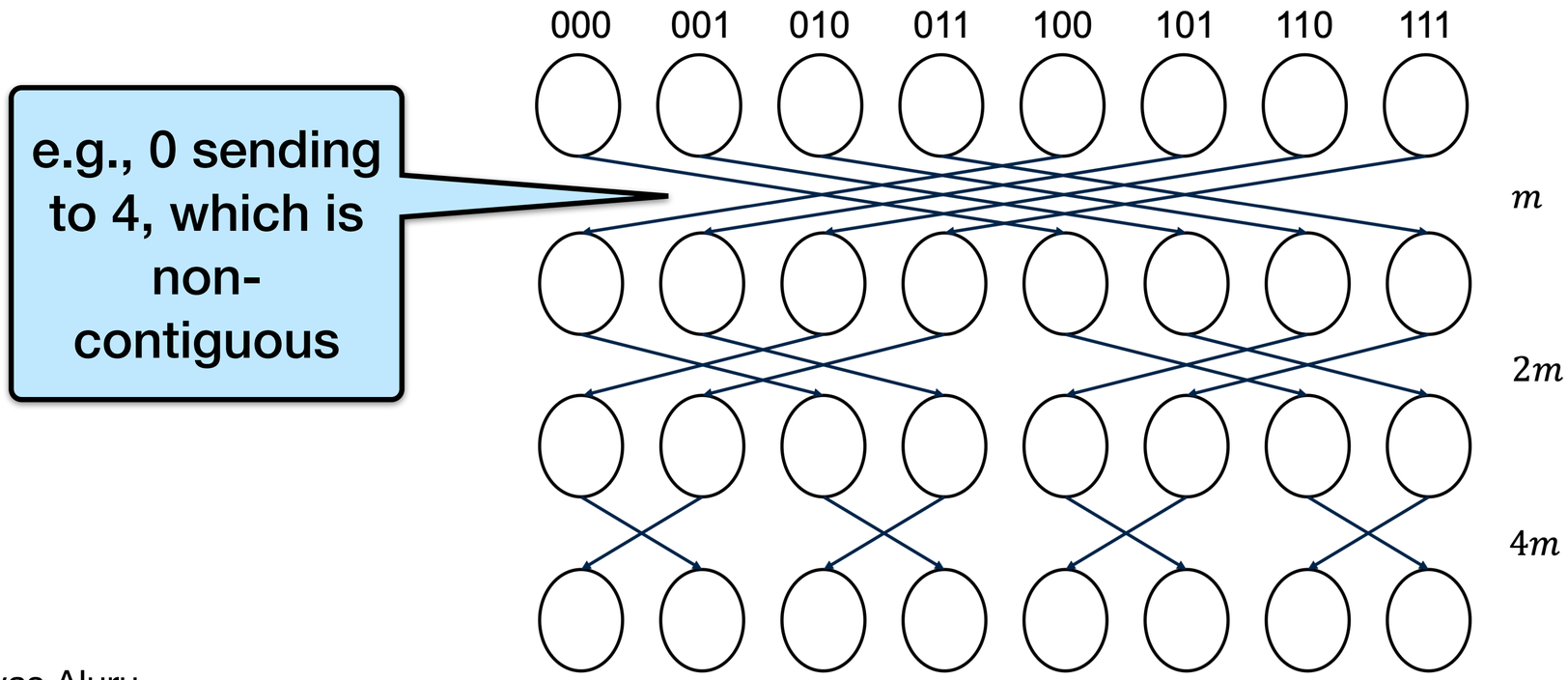


From Srinivas Aluru

Inplace AllGather

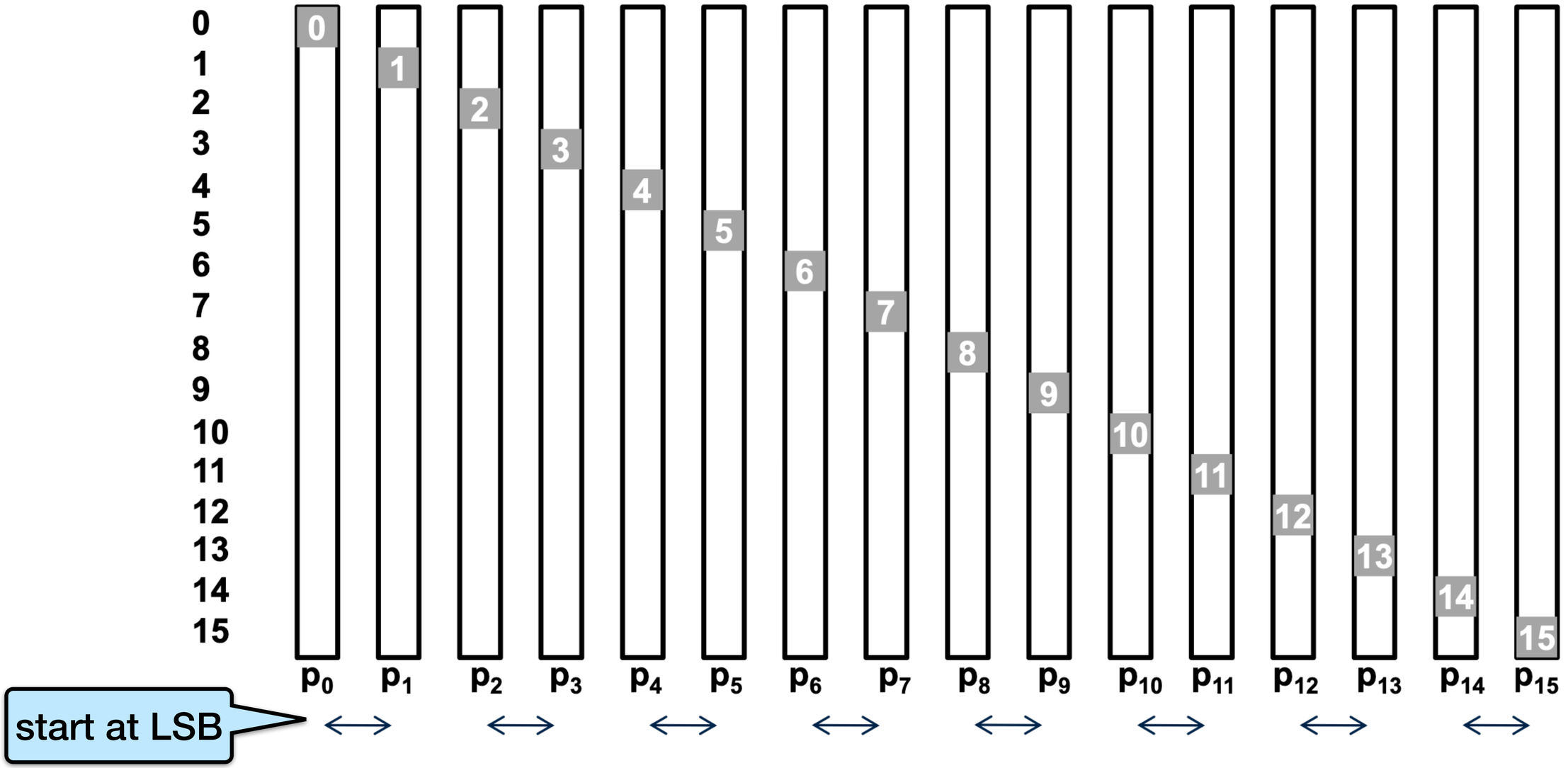
- Goal: Put results of gather on every processor with minimal reshuffling and as much **contiguous data access** as possible.

- Can just directly index into output array, but we want to do better (random indexing would be far away every time).

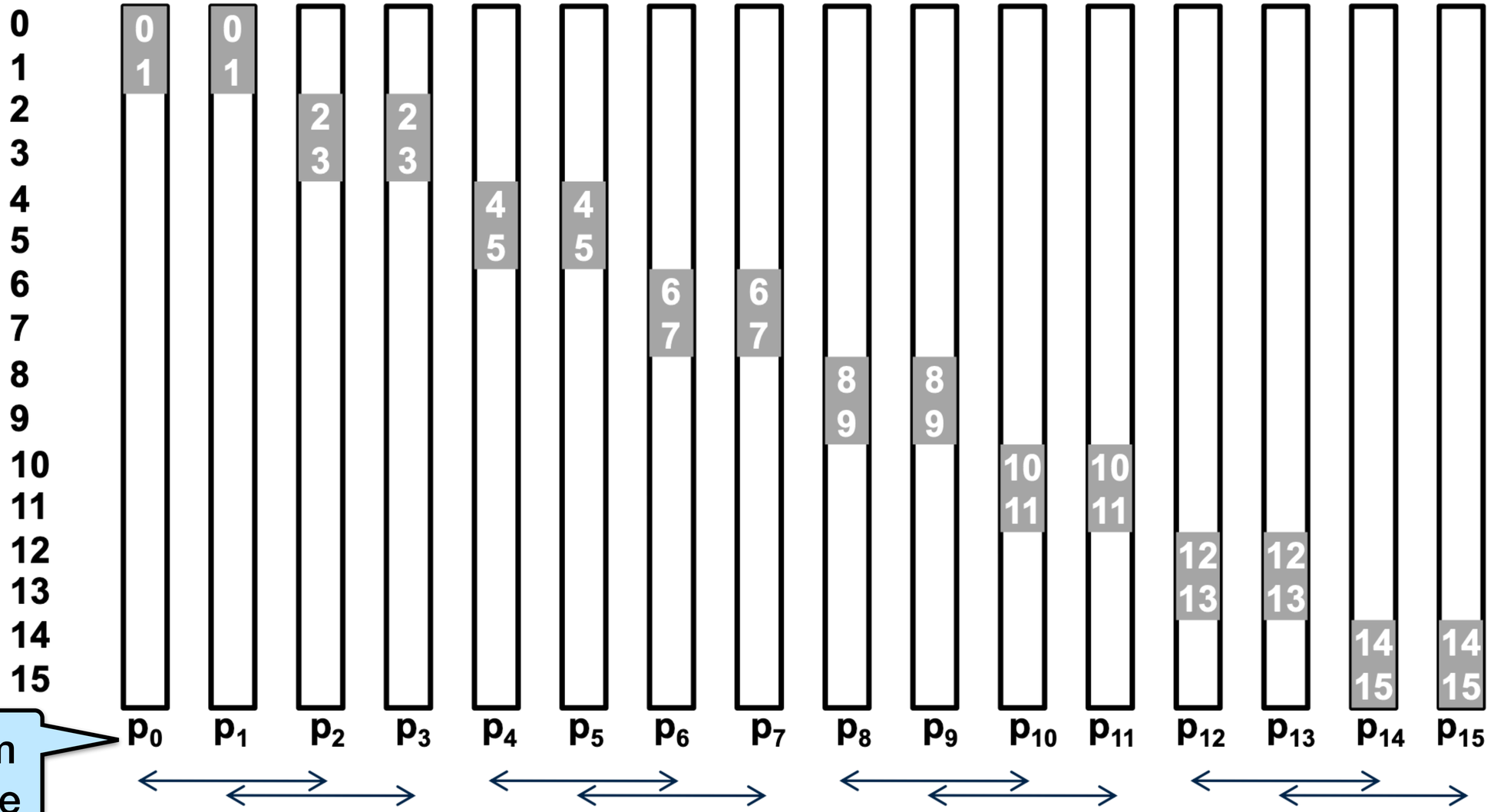


From Srinivas Aluru

Inplace AllGather

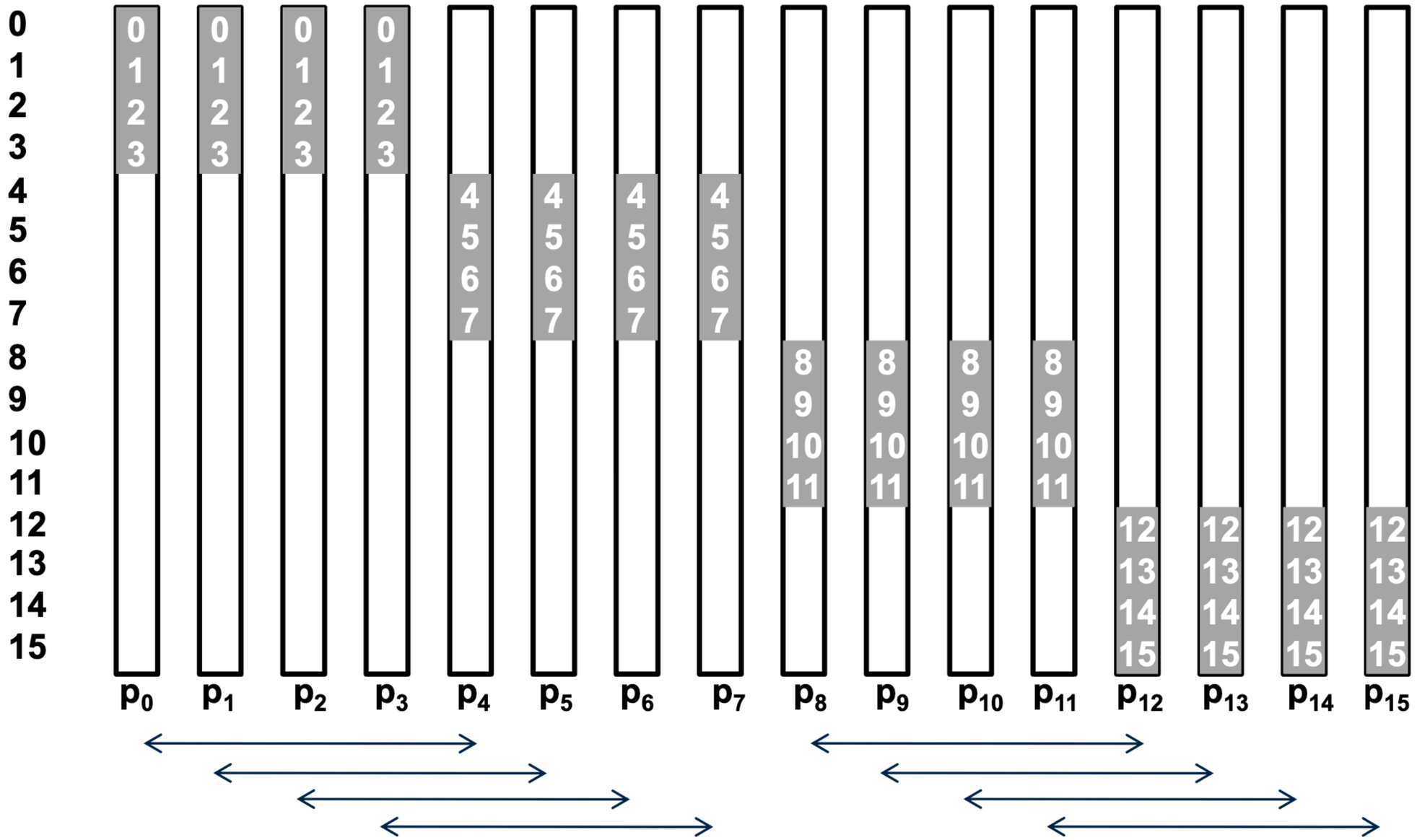


Inplace AllGather



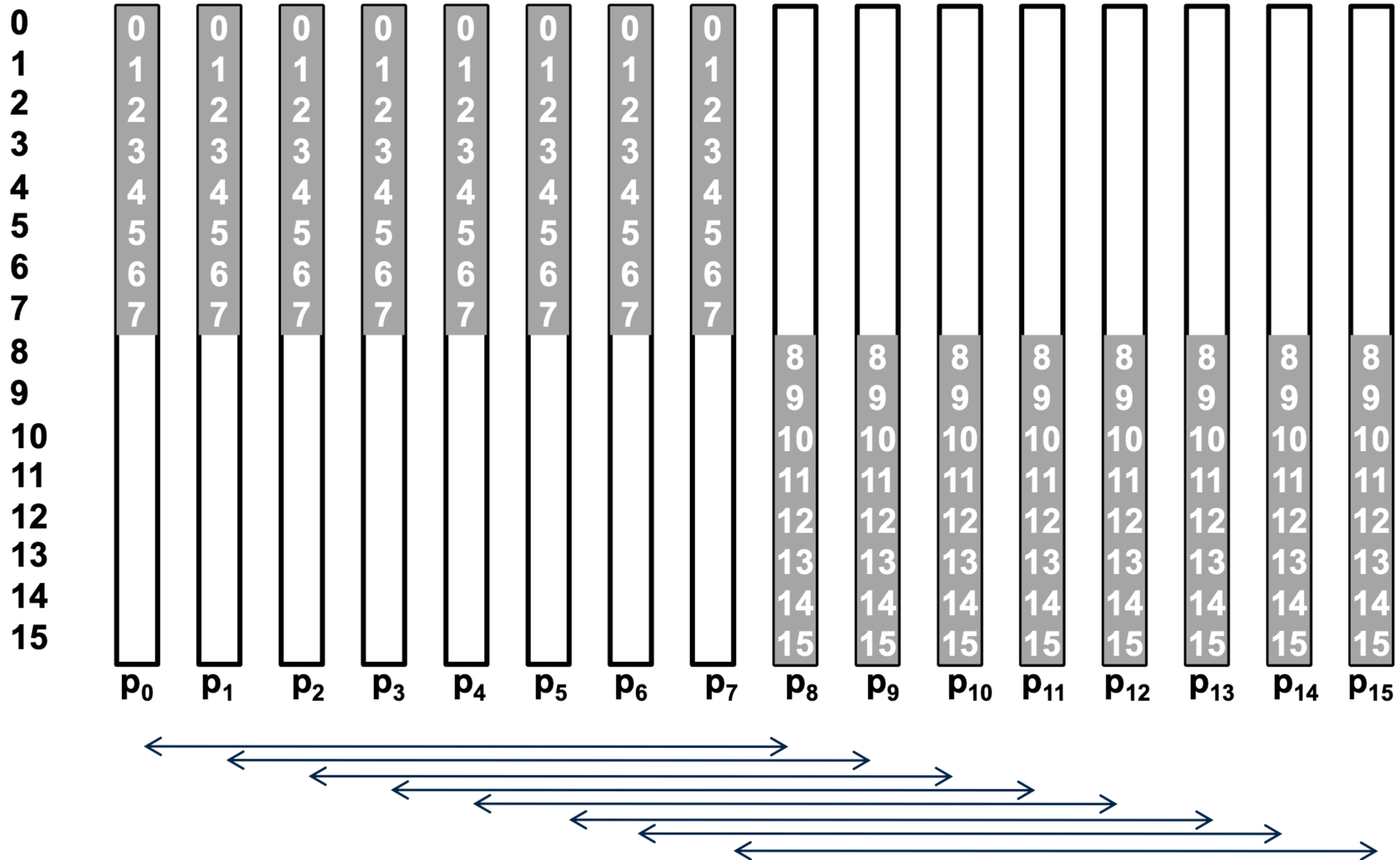
From Srinivas Aluru

Inplace AllGather



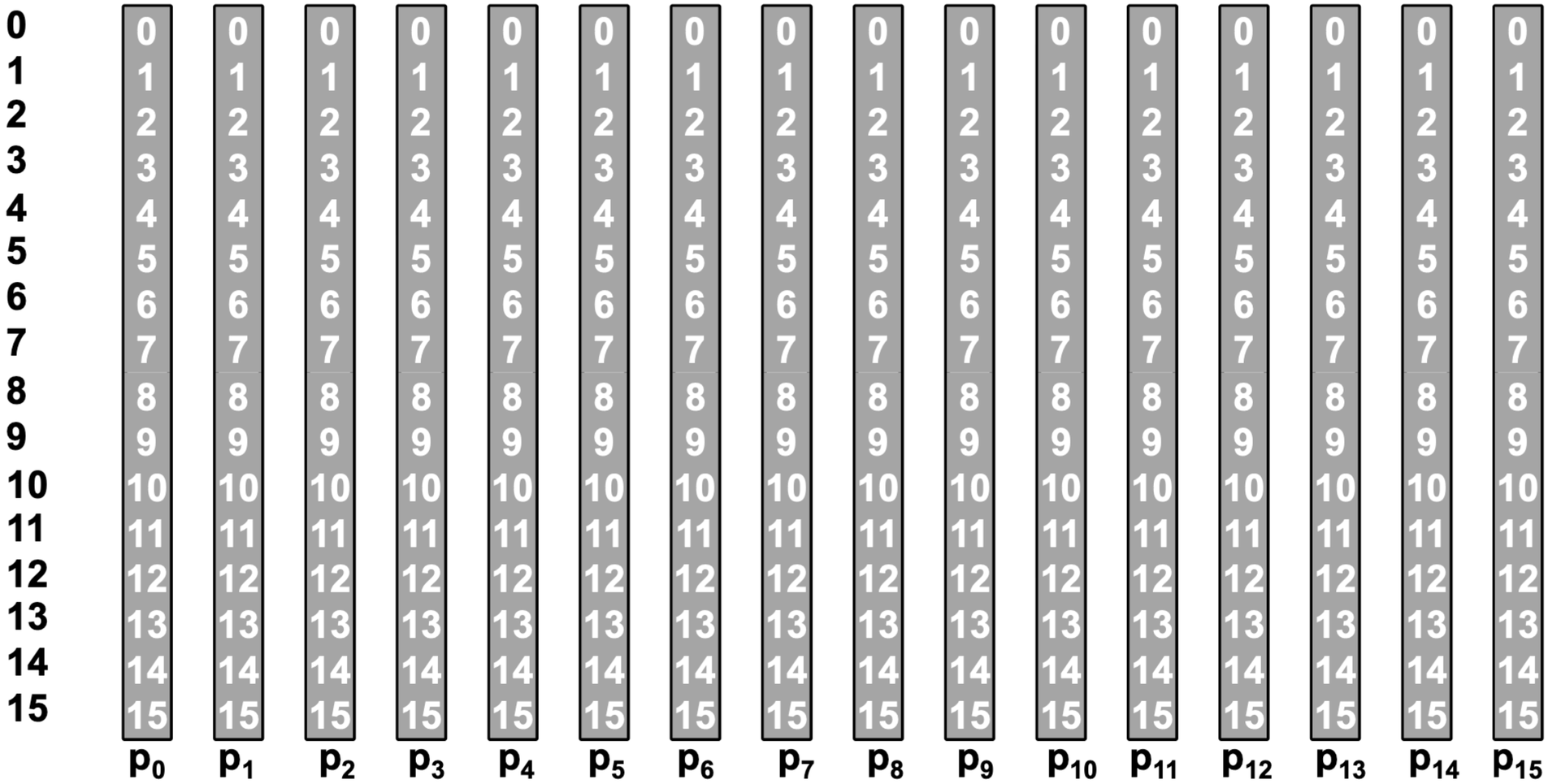
From Srinivas Aluru

Inplace AllGather



From Srinivas Aluru

Inplace AllGather



Final AllGather arrays

AllGather/Gather Runtime

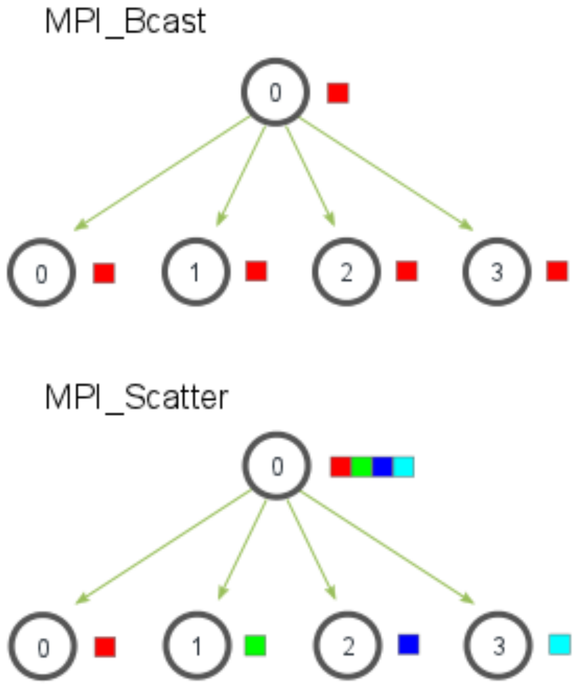
- Run-time for message size m
- Run-time $= \sum_{i=0}^{\log p - 1} (\tau + \mu \cdot m \cdot 2^i)$
- Run-time $= \Theta\left(\tau \log p + \mu \cdot m \cdot \left(1 + 2 + \cdots + \frac{p}{2}\right)\right)$
- Run-time $= \Theta(\tau \log p + \mu \cdot m \cdot p)$

From Srinivas Aluru

Scatter

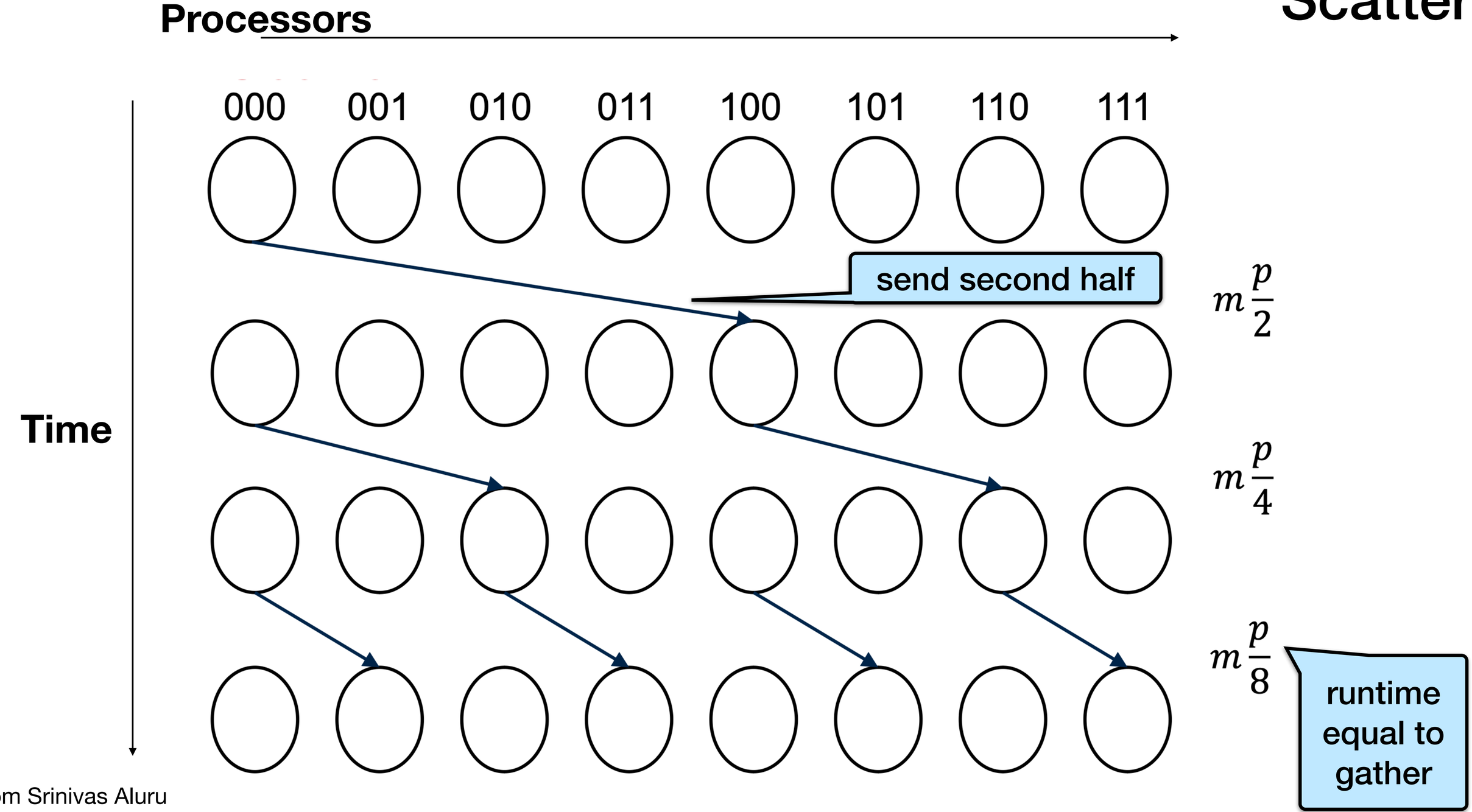
- Scatter is the reverse of gather - **distribute data from root** and partition it across processors.

- Message size should be much less than input size for overall parallel speedup.



From Srinivas Aluru https://mpitutorial.com/tutorials/mpi-scatter-gather-and-allgather/

Scatter



From Srinivas Aluru

Runtime Summary

- Broadcast
- Reduce
- AllReduce
- Scan

$$\Theta(\tau \log p + \mu\, m \log p)$$

- Gather
- AllGather
- Scatter

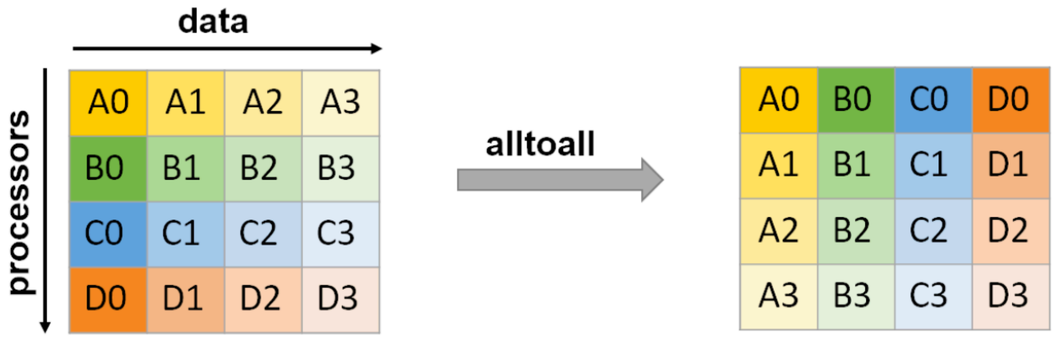
$$\Theta(\tau \log p + \mu\, m\, p)$$

Try to use the primitive with the smallest runtime when possible

From Srinivas Aluru

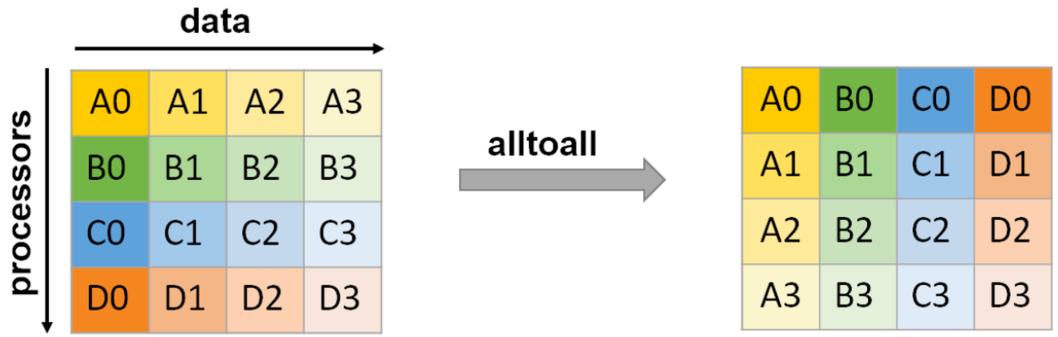
All-to-All

- Example $p = 4$



All-to-All

- Example $p = 4$



$$
\begin{array}{ll}
0: & \begin{bmatrix} m_{00} & m_{01} & m_{02} & m_{03} \end{bmatrix} \\
1: & \begin{bmatrix} m_{10} & m_{11} & m_{12} & m_{13} \end{bmatrix} \\
2: & \begin{bmatrix} m_{20} & m_{21} & m_{22} & m_{23} \end{bmatrix} \\
3: & \begin{bmatrix} m_{30} & m_{31} & m_{32} & m_{33} \end{bmatrix}
\end{array}
\Longrightarrow
\begin{array}{ll}
0: & \begin{bmatrix} m_{00} & m_{10} & m_{20} & m_{30} \end{bmatrix} \\
1: & \begin{bmatrix} m_{01} & m_{11} & m_{21} & m_{31} \end{bmatrix} \\
2: & \begin{bmatrix} m_{02} & m_{12} & m_{22} & m_{32} \end{bmatrix} \\
3: & \begin{bmatrix} m_{03} & m_{13} & m_{23} & m_{33} \end{bmatrix}
\end{array}
$$

- m_{ij} : Message from P_i to P_j

All_to_All: Arbitrary Permutations

Algorithm (for P_i)

```
for j=0 to (p-1) do
```

P_i sends $m_{i,j}$ to P_j

- Is this an efficient algorithm?

All_to_All: Arbitrary Permutations

Algorithm (for P_i)

```
for j=0 to (p-1) do
```

P_i sends $m_{i,j}$ to P_j

Everyone sends to the same processor at the same time

- Is this an efficient algorithm?
- Runtime?
  - $O(\tau \cdot p^2 + \mu \cdot m \cdot p^2)$
- Serializes the communication! Not good.

From Srinivas Aluru

All_to_All: Arbitrary Permutations

Algorithm (for P_i)

```
for j=1 to (p-1) do
```

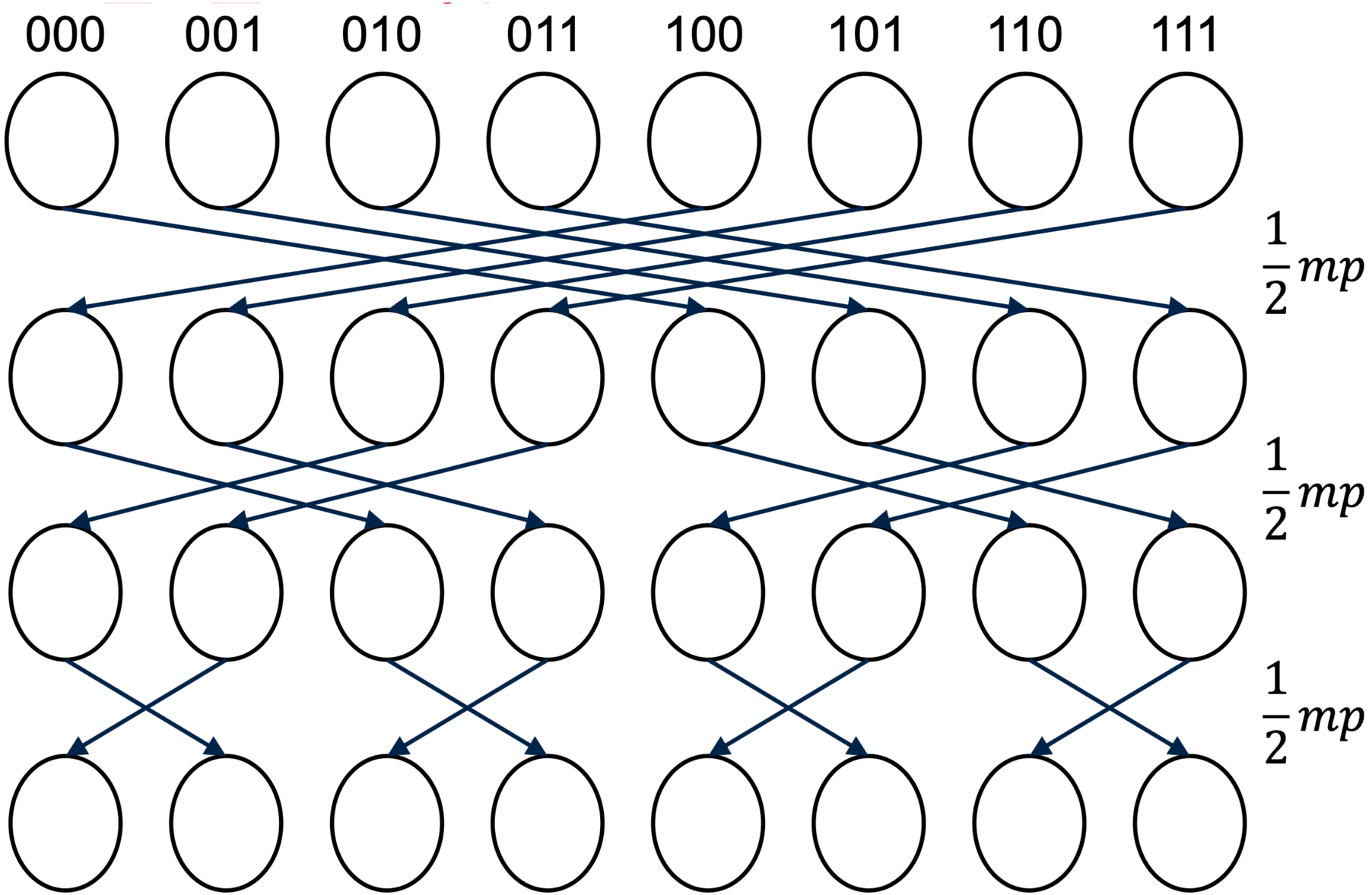
P_i sends $m_{i,(i+j) \bmod p}$ to $P_{(i+j) \bmod p}$

- $O(\tau \cdot p + \mu \cdot m \cdot p)$

Was $\tau \lg p$ for gather/scatter

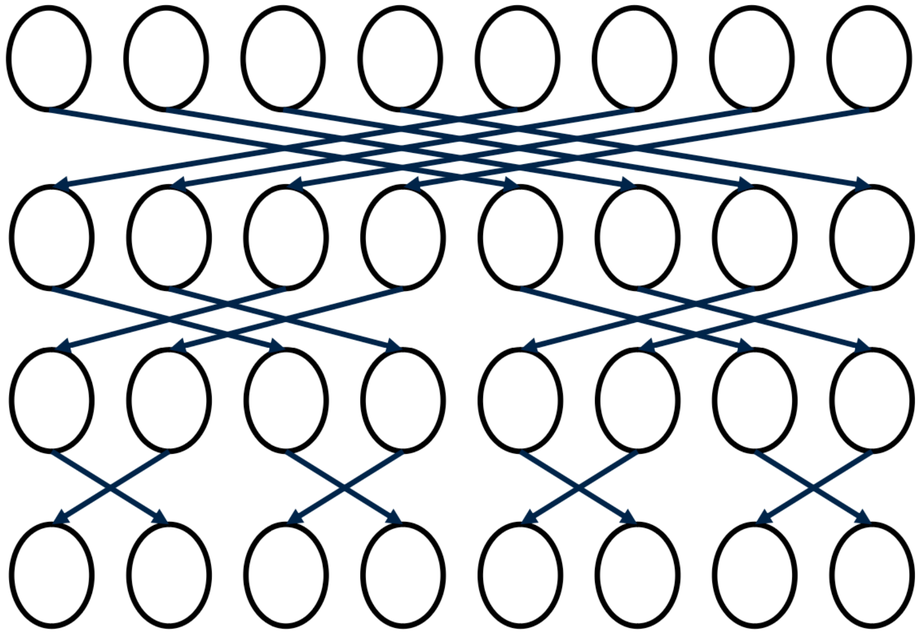
From Srinivas Aluru

All_to_All: Hypercubic Permutations



From Srinivas Aluru

All_to_All: Hypercubic Permutations



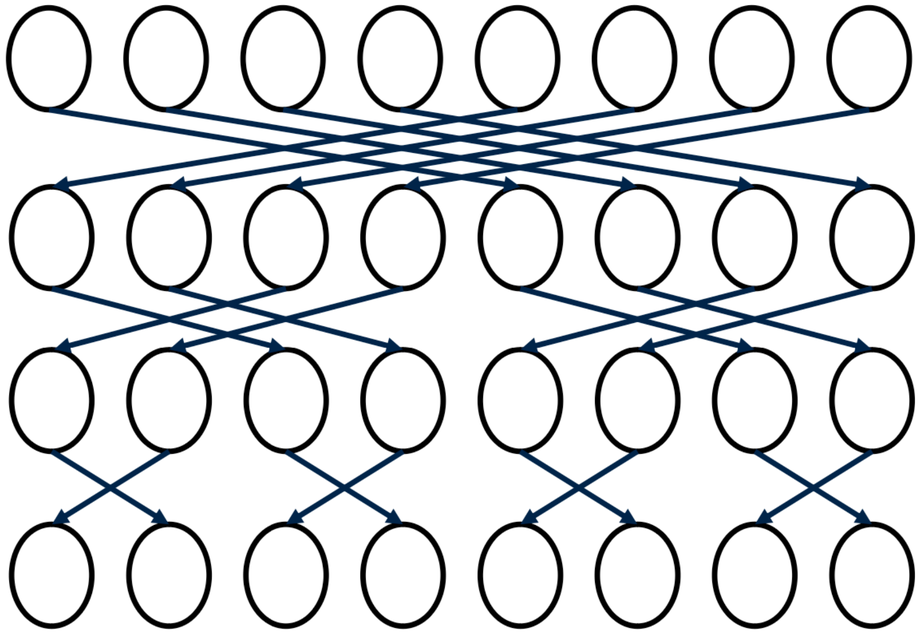
- P_0: m_{00} m_{01} m_{02} m_{03} m_{04} m_{05} m_{06} m_{07}
- P_0: m_{00} m_{40} m_{01} m_{41} m_{02} m_{42} m_{03} m_{43}

Send away right half

Receive left half, interleave results

Red = recv
blue = send

All_to_All: Hypercubic Permutations



- P_0: m_{00} m_{01} m_{02} m_{03} m_{04} m_{05} m_{06} m_{07}
- P_0: m_{00} m_{40} m_{01} m_{41} m_{02} m_{42} m_{03} m_{43}
- P_0: m_{00} m_{20} m_{40} m_{60} m_{01} m_{21} m_{41} m_{61}

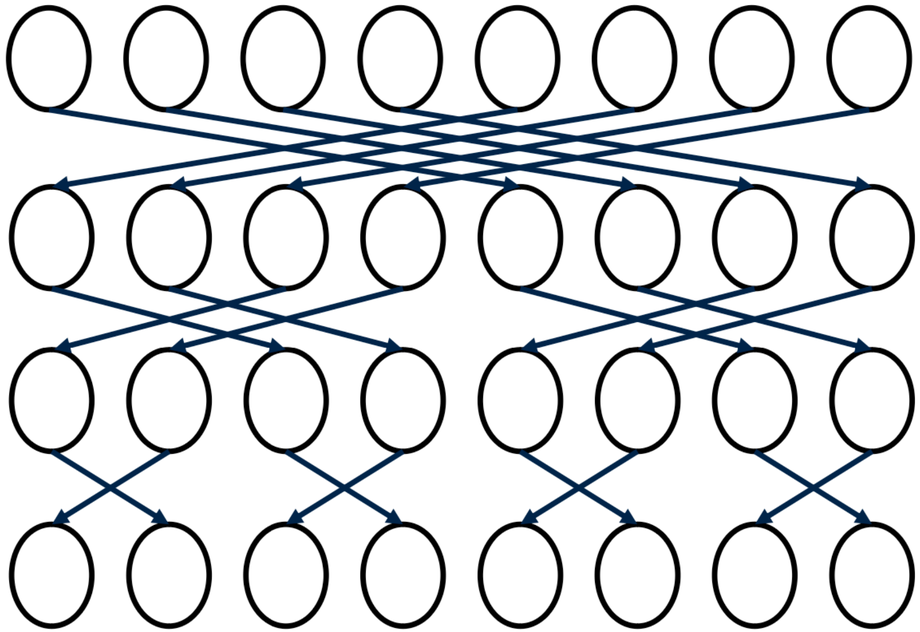
Send away right half

Red = recv
blue = send

Receive left half, interleave results

From Srinivas Aluru

All_to_All: Hypercubic Permutations



- P_0: m_{00} m_{01} m_{02} m_{03} m_{04} m_{05} m_{06} m_{07}
- P_0: m_{00} m_{40} m_{01} m_{41} m_{02} m_{42} m_{03} m_{43}
- P_0: m_{00} m_{20} m_{40} m_{60} m_{01} m_{21} m_{41} m_{61}
- P_0: m_{00} m_{10} m_{20} m_{30} m_{40} m_{50} m_{60} m_{70}
- $O(\tau \cdot \log p + \mu \cdot m \cdot p \cdot \log p)$

Choose depending on message size

Send away right half

Receive left half, interleave results

Red = recv
blue = send

From Srinivas Aluru

Runtime Summary

- Broadcast
- Reduce
- AllReduce
- Scan

$\Theta(\tau \log p + \mu\, m \log p)$

- Gather
- AllGather
- Scatter

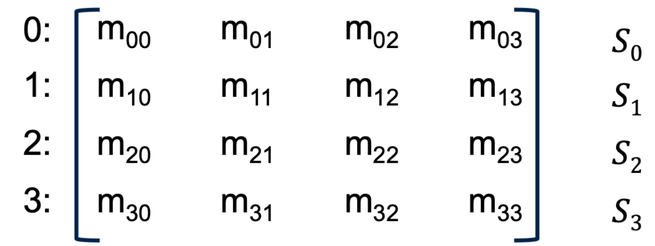
$\Theta(\tau \log p + \mu\, m\, p)$

- All_to_All
  - Arbitrary Permutations: $O(\tau \cdot p + \mu \cdot m \cdot p)$
  - Hypercubic Permutations: $O(\tau \cdot \log p + \mu \cdot m \cdot p \cdot \log p)$

From Srinivas Aluru

Many_to_Many With Bounded Traffic

- Generalized version of all-to-all
- Processors can have variable amount of data to send to all other processors (may be 0, in some cases)
  - m_{ij} : message from P_i to P_j
  - $|m_{ij}|$ = size of the message
    - $|m_{ij}| = 0$: message doesn't exist
  - $\max_i \sum_j |m_{ij}| \leq S$ — Total message size to send per processor
  - $\max_j \sum_i |m_{ij}| \leq R$ — Total message size to receive per processor



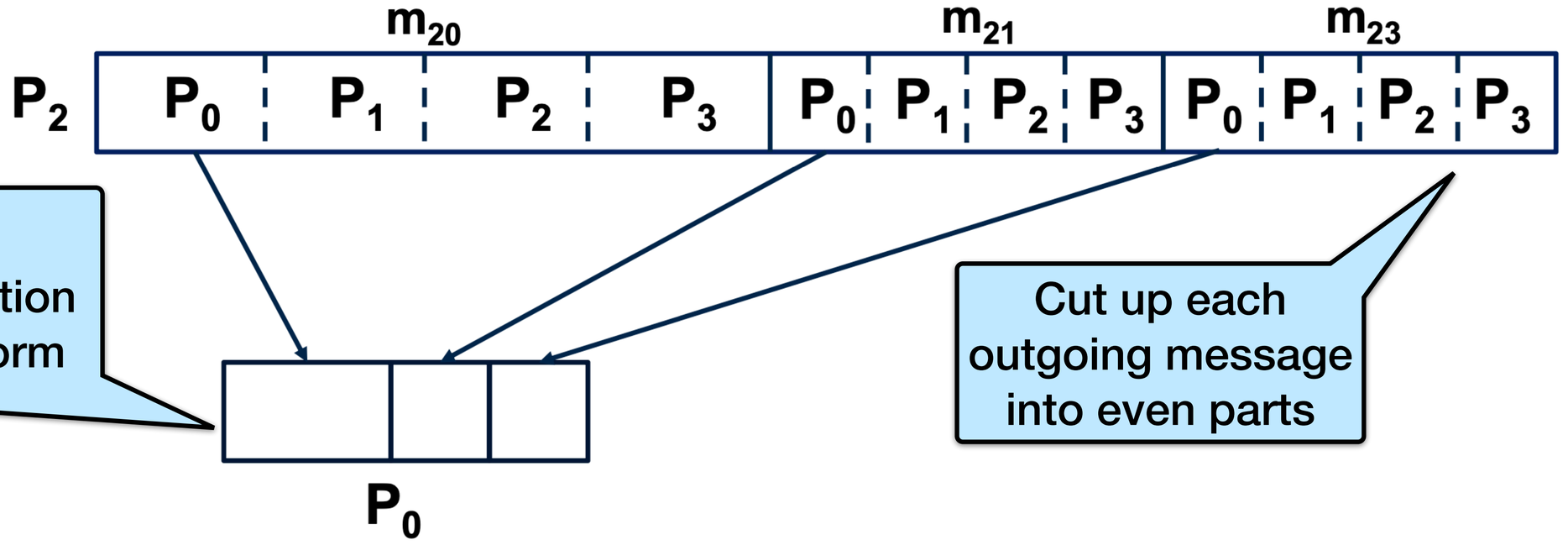
Many_to_Many

- Initial setup: Variable message size destined for each processor
- Example with p = 4

Many_to_Many

Converting to fixed-size all-to-alls

- **Stage 1**: example with $p = 4$



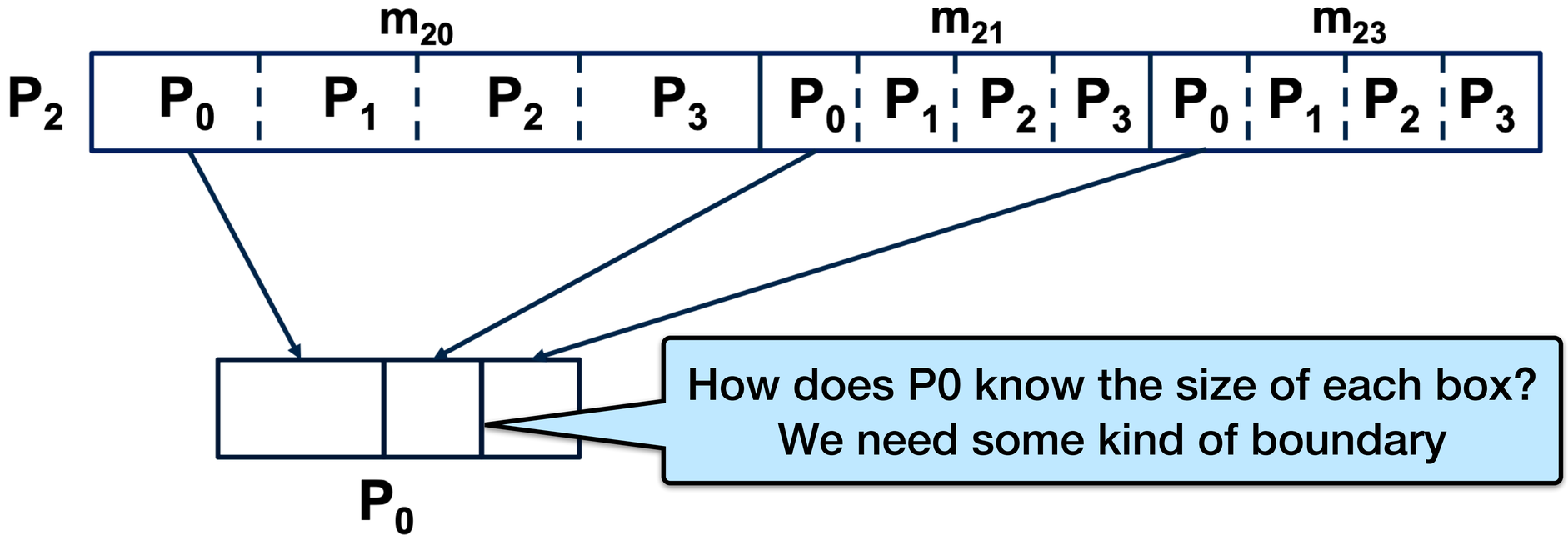
- All_to_All with max message size $\leq \frac{S}{p}$

Many_to_Many

- **Stage 2**: Route the message fragments to actual destinations and assemble
  - All_to_All with max message size $\leq \frac{R}{p}$
- Runtime for All_to_All: $\Theta(\tau \cdot p + \mu \cdot m \cdot p)$
- Stage 1: $\Theta(\tau \cdot p + \mu \cdot \frac{S}{\cancel{p}} \cdot \cancel{p})$
- Stage 2: $\Theta(\tau \cdot p + \mu \cdot \frac{R}{\cancel{p}} \cdot \cancel{p})$
- Total Runtime: $\Theta(\tau \cdot p + \mu \cdot (R + S))$

Many_to_Many

- **Stage 1**: example with $p = 4$



- All_to_All with max message size $\leq \frac{S}{p}$

Many_to_Many

- Stage 1: $\Theta\left(\tau \cdot p + \mu \cdot \left(\frac{S}{p} + p\right) \cdot p\right)$
- Stage 2: $\Theta\left(\tau \cdot p + \mu \cdot \left(\frac{R}{p} + p\right) \cdot p\right)$

Space required to mark the boundary is constant per processor

- Total: $\Theta(\tau \cdot p + \mu \cdot (R + S + p^2))$
- Total: $\Theta\left(\tau \cdot p + \mu \cdot (R + S)\right)$ provided $p^2 = O(S + R)$

Runtime Summary

- Broadcast
- Reduce
- AllReduce
- Scan

$\Theta(\tau \log p + \mu\, m \log p)$

- Gather
- AllGather
- Scatter

$\Theta(\tau \log p + \mu\, m\, p)$

- All_to_All
  - Arbitrary Permutations: $O(\tau \cdot p + \mu \cdot m \cdot p)$
  - Hypercubic Permutations: $O(\tau \cdot \log p + \mu \cdot m \cdot p \cdot \log p)$
- Many_to_Many: $\Theta\big(\tau \cdot p + \mu \cdot (R + S)\big)$ provided $p^2 = O(S + R)$

From Srinivas Aluru