Super Algorithms for Supercomputers

Lecture 2 : Parallel Communication Algorithms

08/27/2021

Professor Amanda Bienz

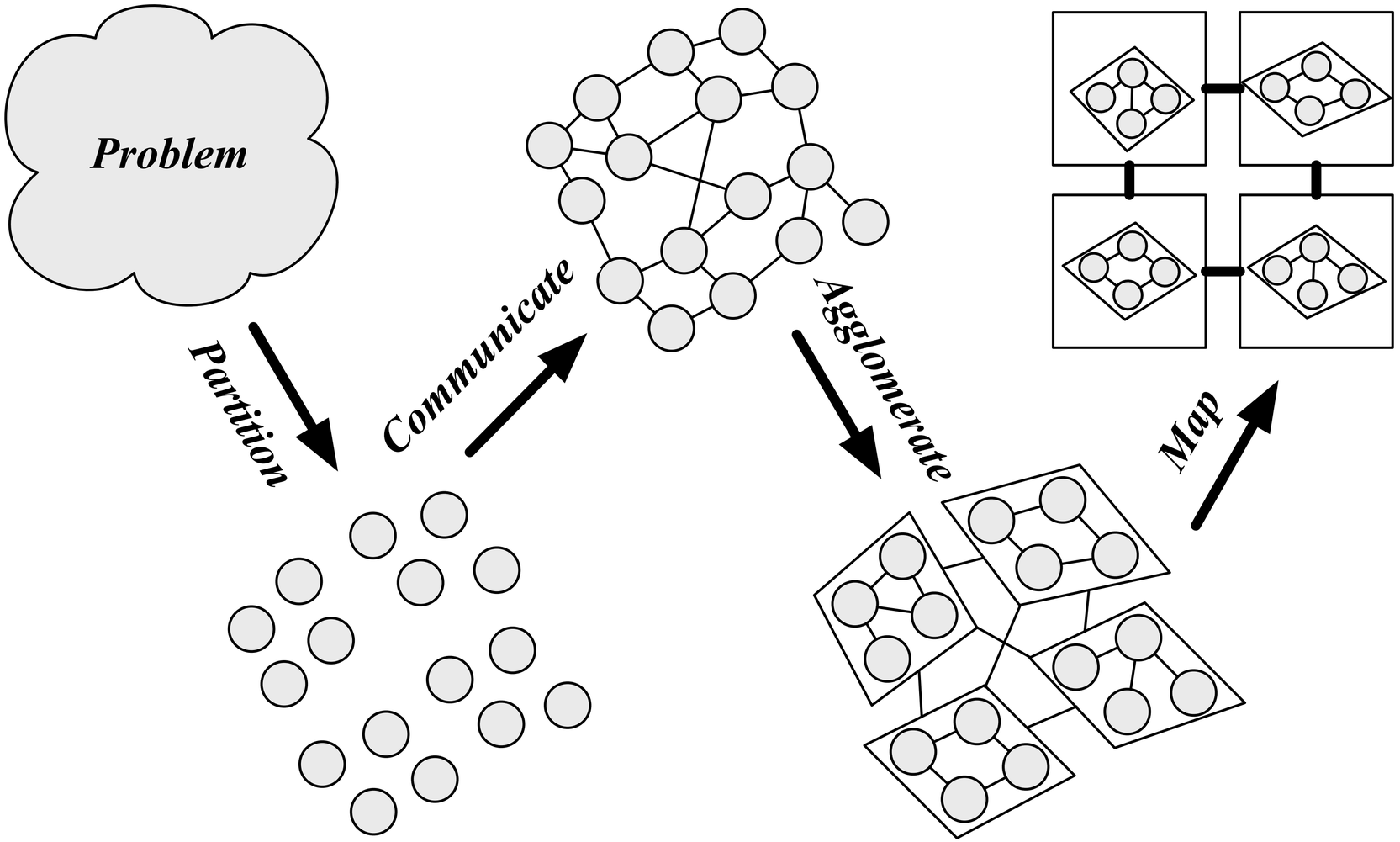
https://courses.grainger.illinois.edu/cs554/fa2015/notes/02_design.pdf

Today's Learning Objectives

- What are the steps in parallel algorithm design?
- What qualities make a partition desirable?
- How do you optimally communication among all processes?
  - E.g. broadcast, reduce, allreduce, allgather, etc

Parallel Algorithm Design

- **Partition:** decompose global problem into smaller tasks
- **Communicate:** determine how tasks must communicate with one another (form a *task graph*)
- **Agglomerate:** combine groups of fine-grain tasks to form coarse-grain tasks (minimizing communication)
- **Map:** assign coarse-grain tasks to processors


Problem
Partition
Communicate
Agglomerate
Map

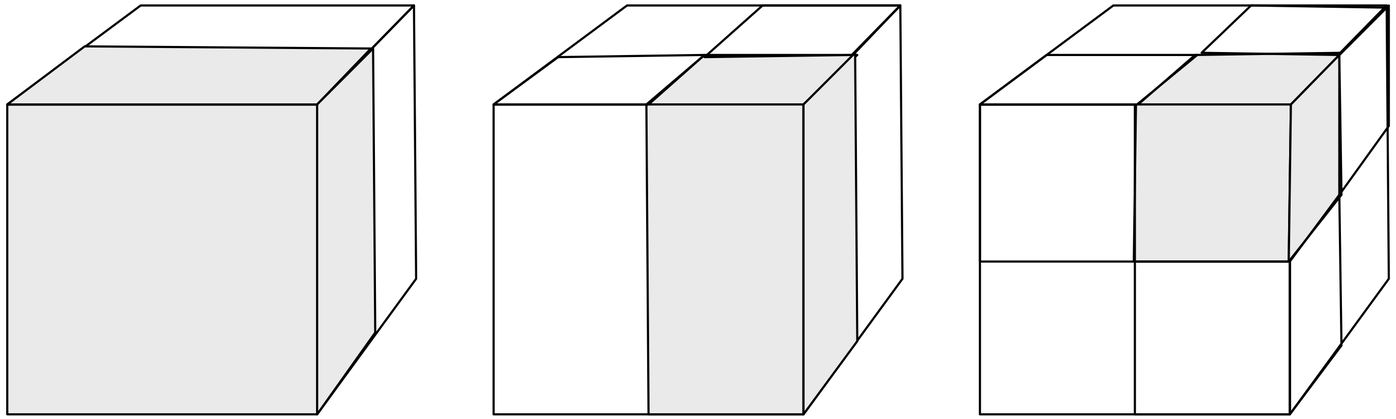
Partitioning Strategies

- **Independent tasks:** subdivide computation into tasks that do not depend on one another (embarrassingly parallel)
- **Domain decomposition:** subdivide geometric domain into subdomains
- **Divide-and-conquer:** recursively divide problem into tree-like hierarchy or subproblems
- **Pipelining:** break problem into sequence of stages for each sequence of objects

Desirable Partitions

- Execute many tasks at same time
- Many more fine-grain tasks than processors
- Number of tasks grows with problem size (not size of tasks)
- Tasks uniform in size
- Avoid redundant computation / storage

Example : Domain Decomposition



- 3D domain with 1D, 2D or 3D partition
- For each case, does task size grow with problem size?
- Which partition is best?

Parallel Communication

- Data dependencies between tasks –> communication pattern
  - Each task holds it’s own data
  - If this data is needed by another task, must be communicated
- Communication patterns:
  - Local or global
  - Structured or irregular
  - Persistent or dynamically changing
  - Synchronous or sporadic

Desirable Communication

- Minimize frequency and volume
- As localized as possible (between neighboring tasks)
- Uniform among tasks
- Allows tasks to be execute concurrently
- Overlap communication and computation as much as possible

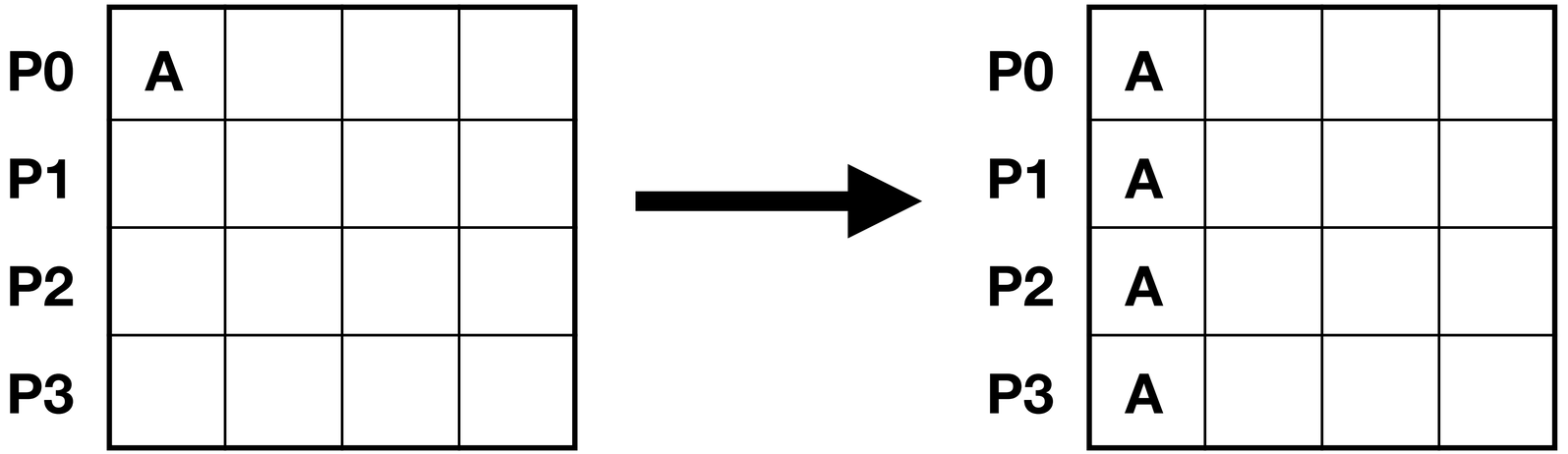
Collective Communication

- **Broadcast :** One process has data and wants to give this data to every other process
- **Reduce :** Each process has data, want to reduce this data onto a single process
  - E.g. : Each process has 1 value, want P0 to hold the sum of these values (or max, min, etc).
- **Gather :** Each process has data, want a single process to gather all of this data
- **Scatter :** A single process holds an array of data, want to give a portion of this data to every other process

Collective Communication

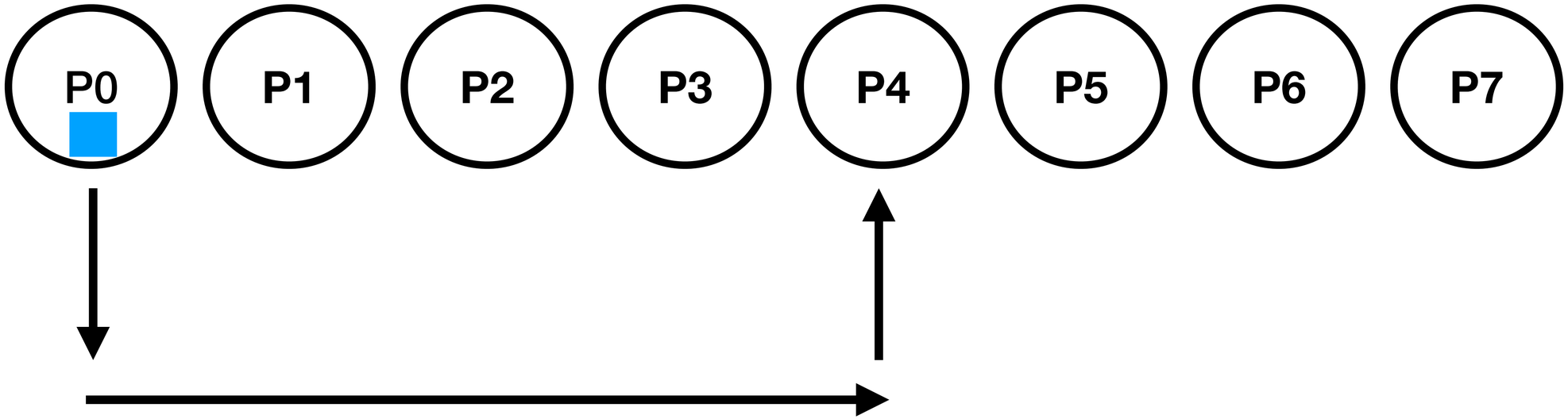
- **Allreduce :** Each process holds data, want all processes to hold reduction of this data
  - E.g. each process holds a value, want all processes to hold the sum of these values
- **Allgather :** Each process holds data, want all processes to gather all data
- **Alltoall :** Each process holds an array of data, want a portion of this data to go to each process

Broadcast

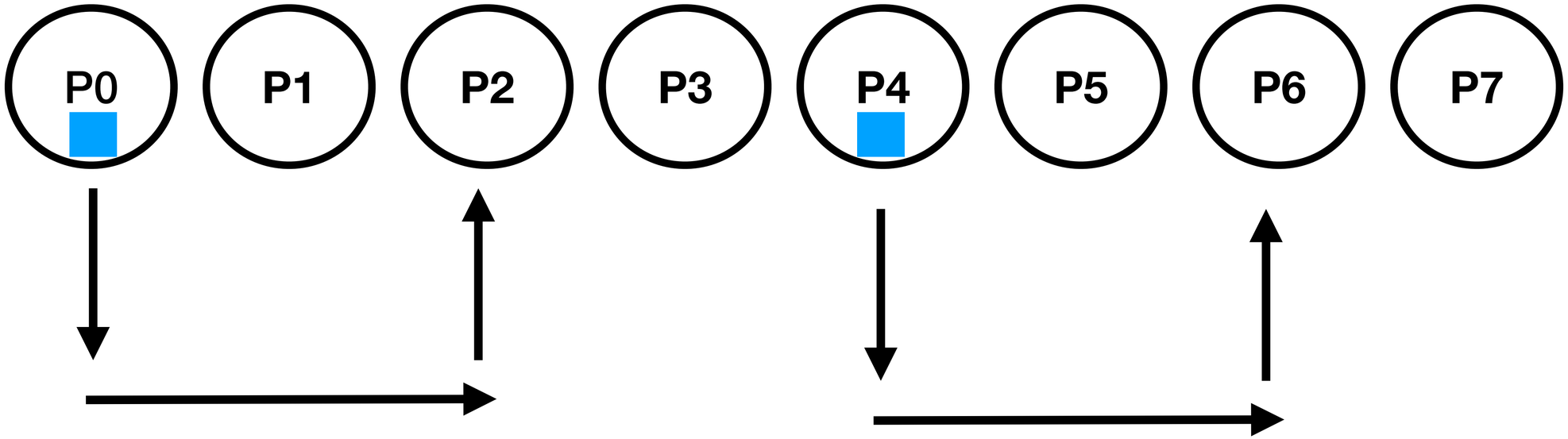


- Process 0 starts with a value A
- After broadcast, all processes hold the value A
- How do we efficiently broadcast data?

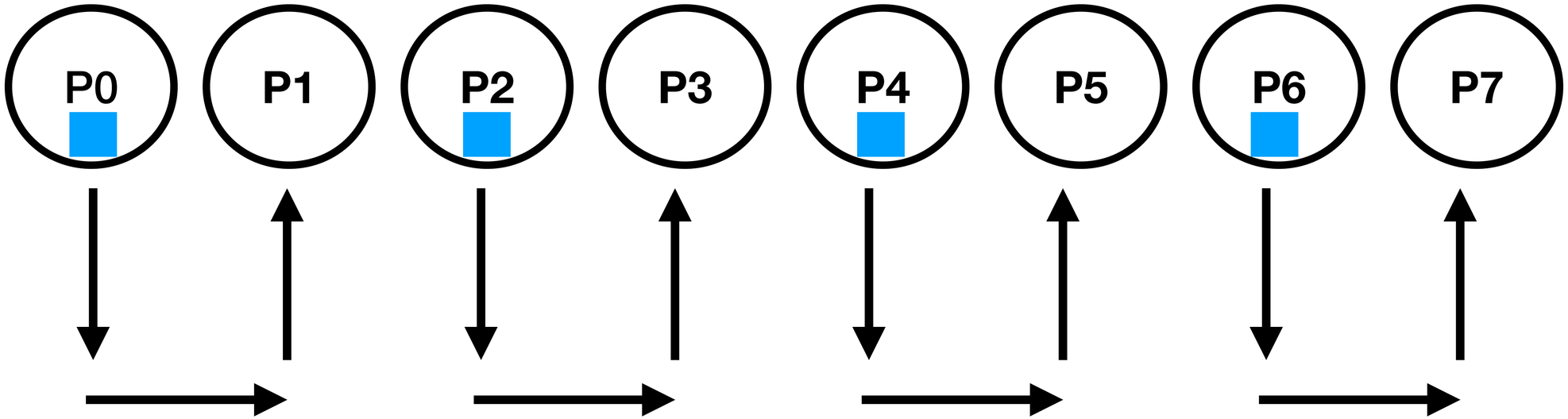
Binomial Tree Algorithm



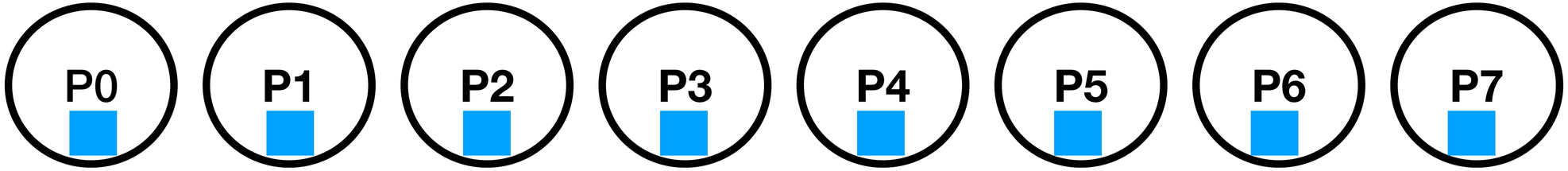
Binomial Tree Algorithm



Binomial Tree Algorithm

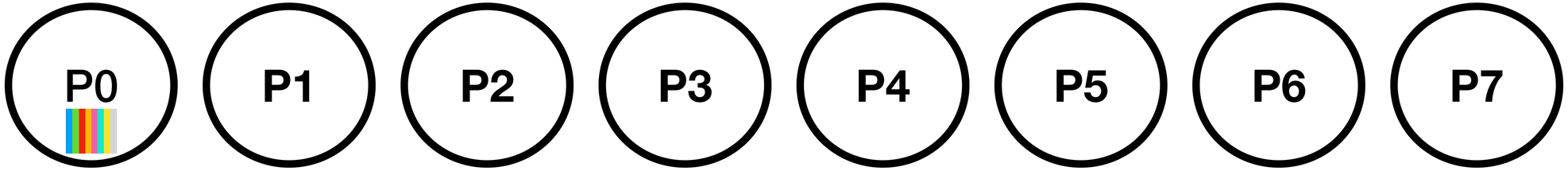


Binomial Tree Algorithm

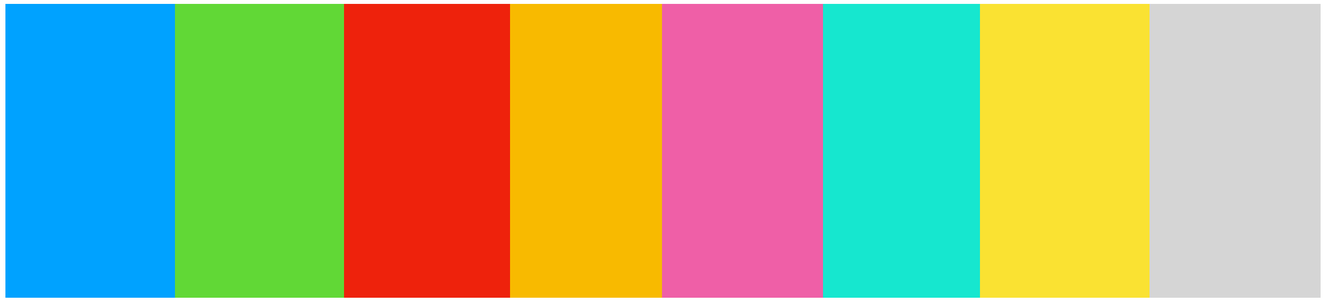


- log(num_procs) steps
- At each step, all n values are communicated
  - i.e. full blue box communicated at each of the log(num_procs) steps
- $T = \log_2(p) \cdot (\alpha + \beta n)$

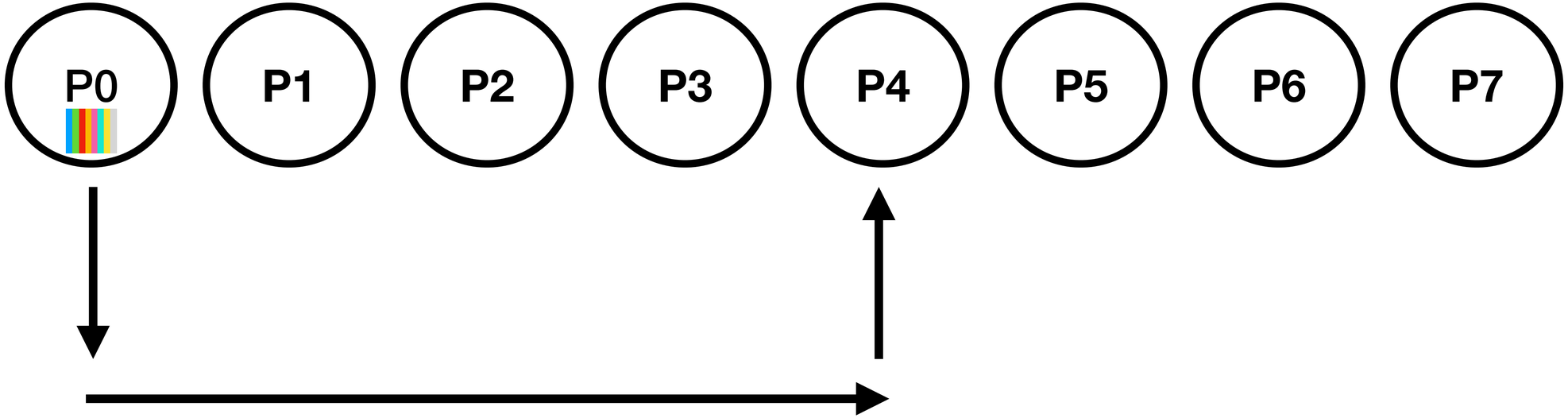
Scatter + Allgather Algorithm

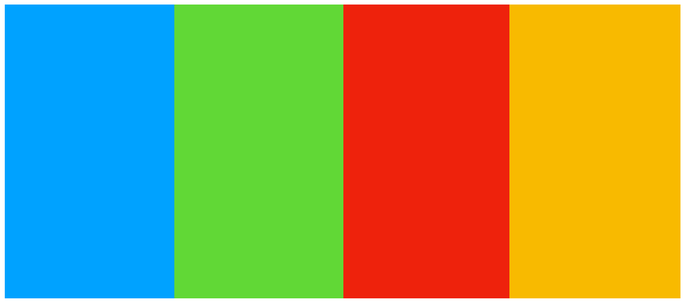


- For larger broadcasts, it is actually cheaper to first scatter the data, and then perform an allgather!

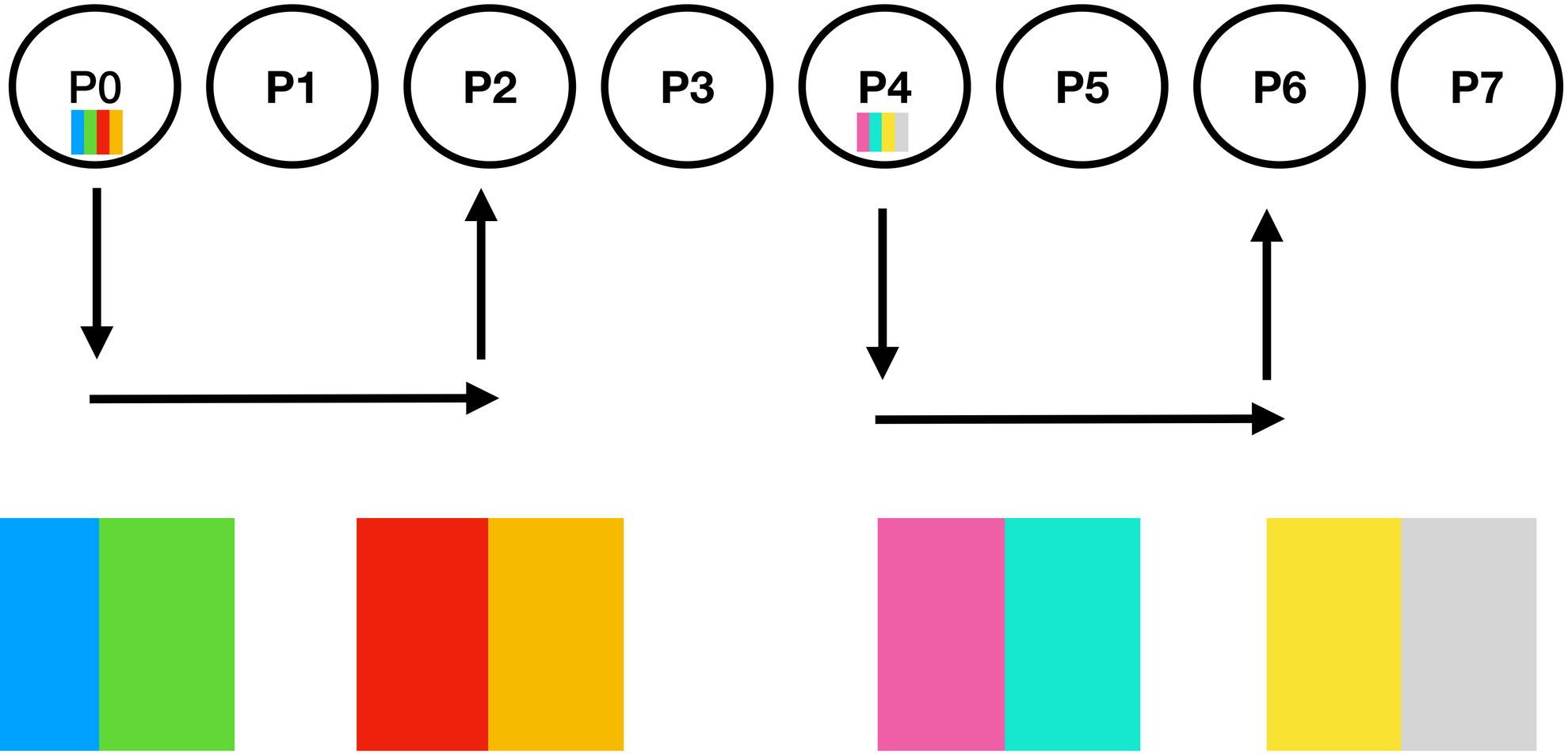


Scatter + Allgather Algorithm

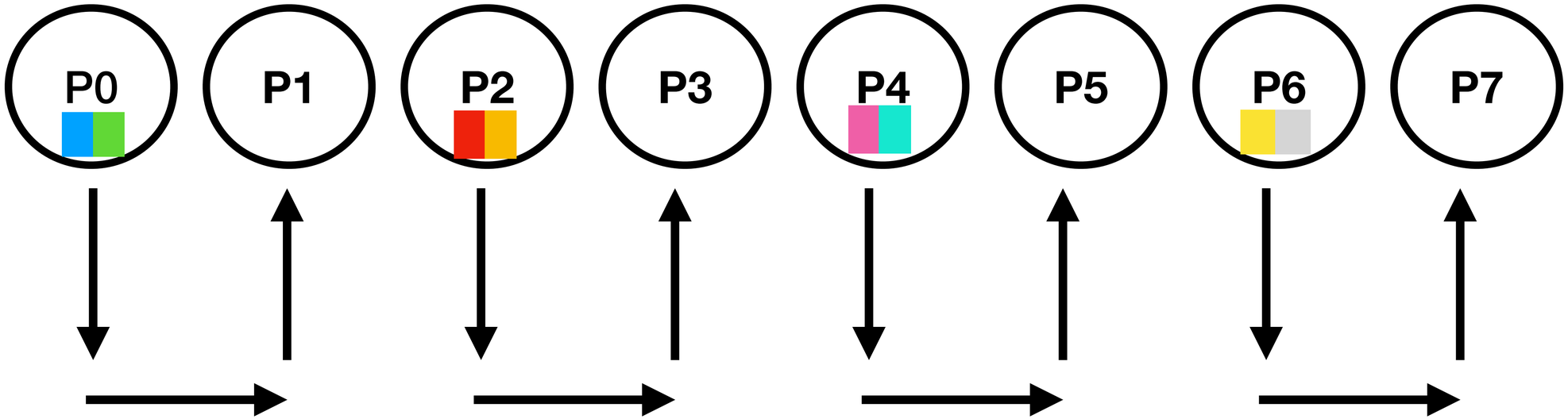




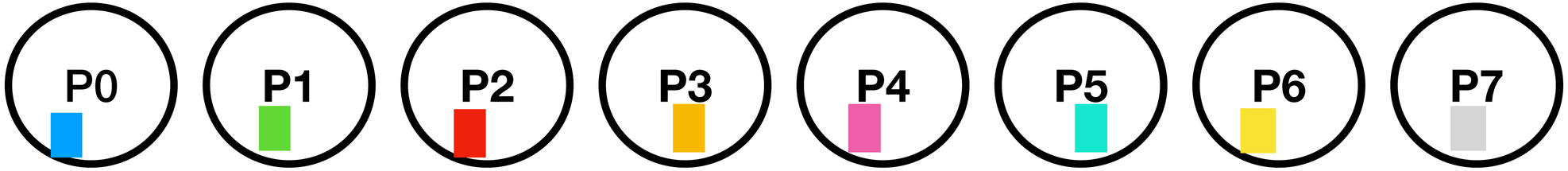
Scatter + Allgather



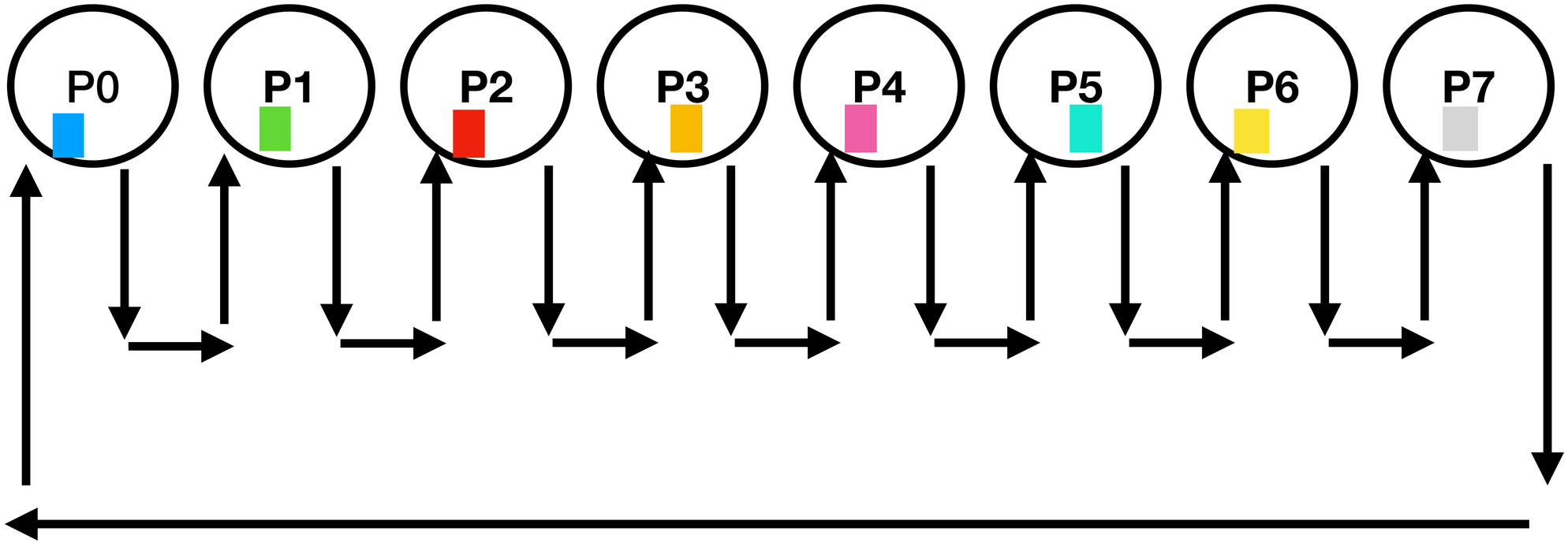
Scatter + Allgather



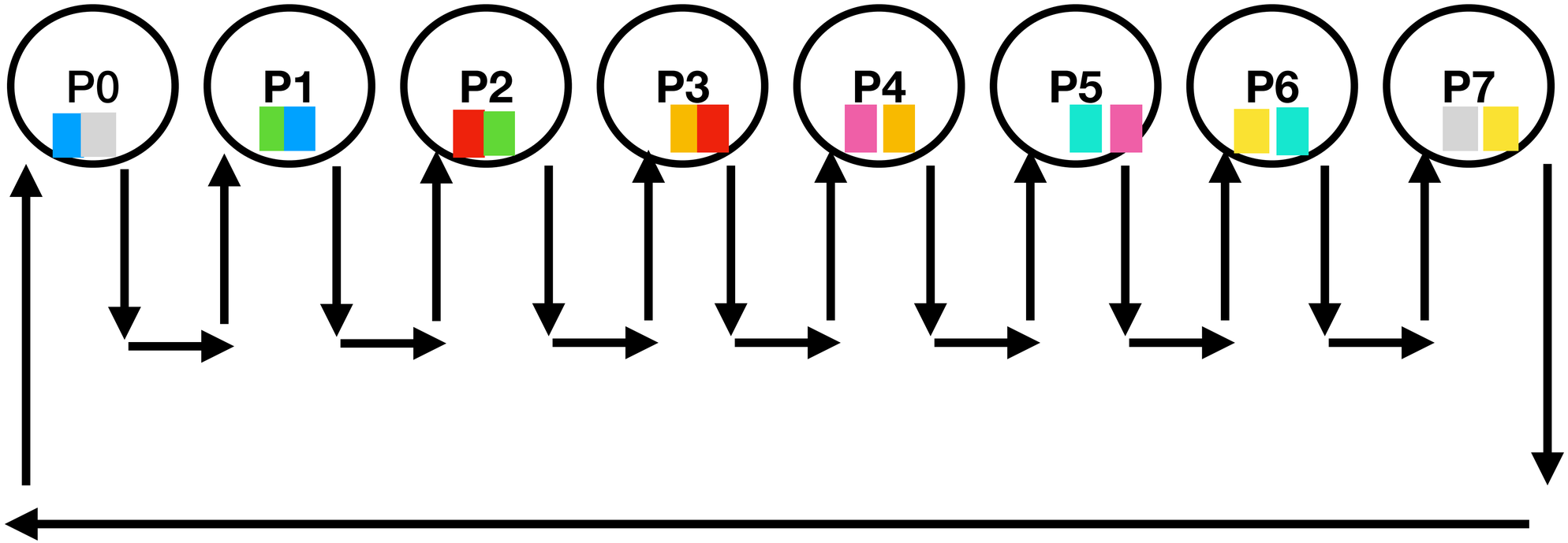
Scatter + Allgather



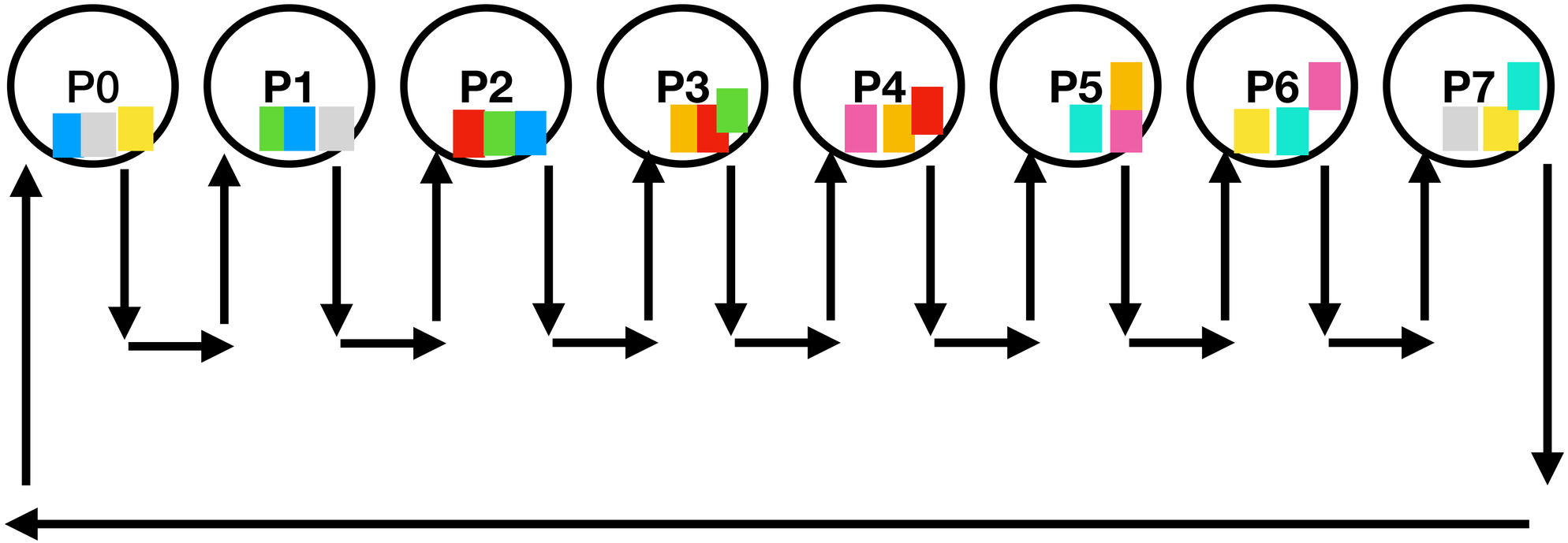
Scatter + Allgather



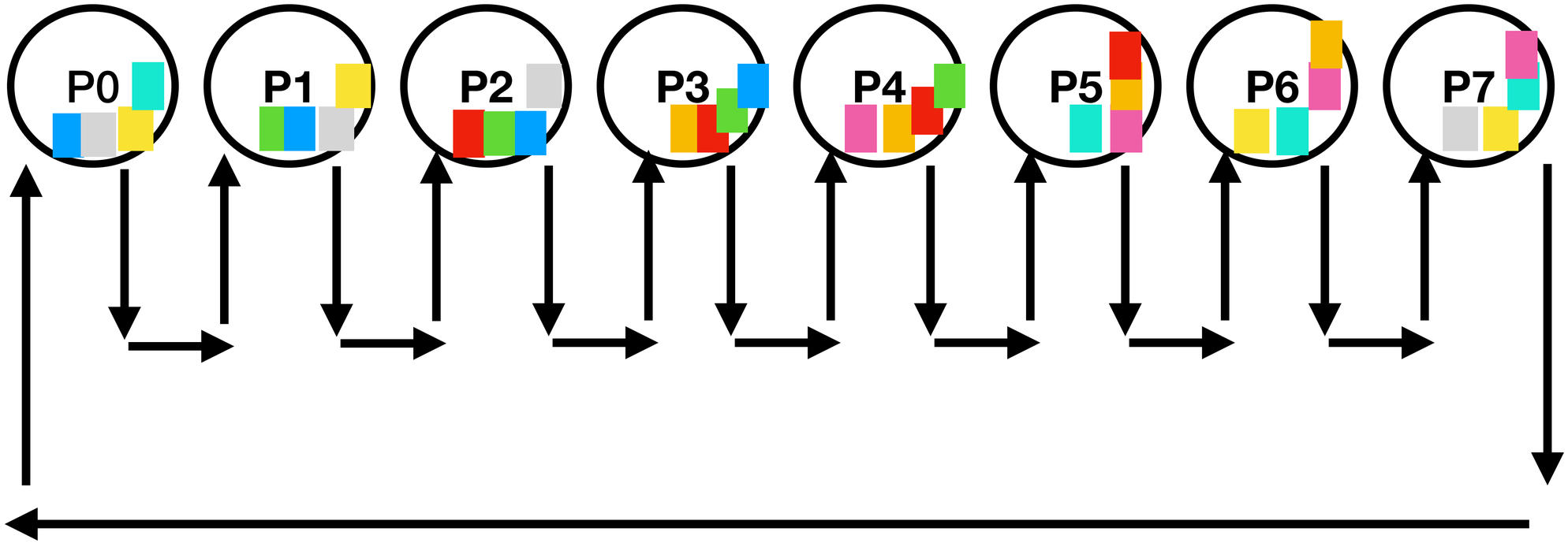
Scatter + Allgather



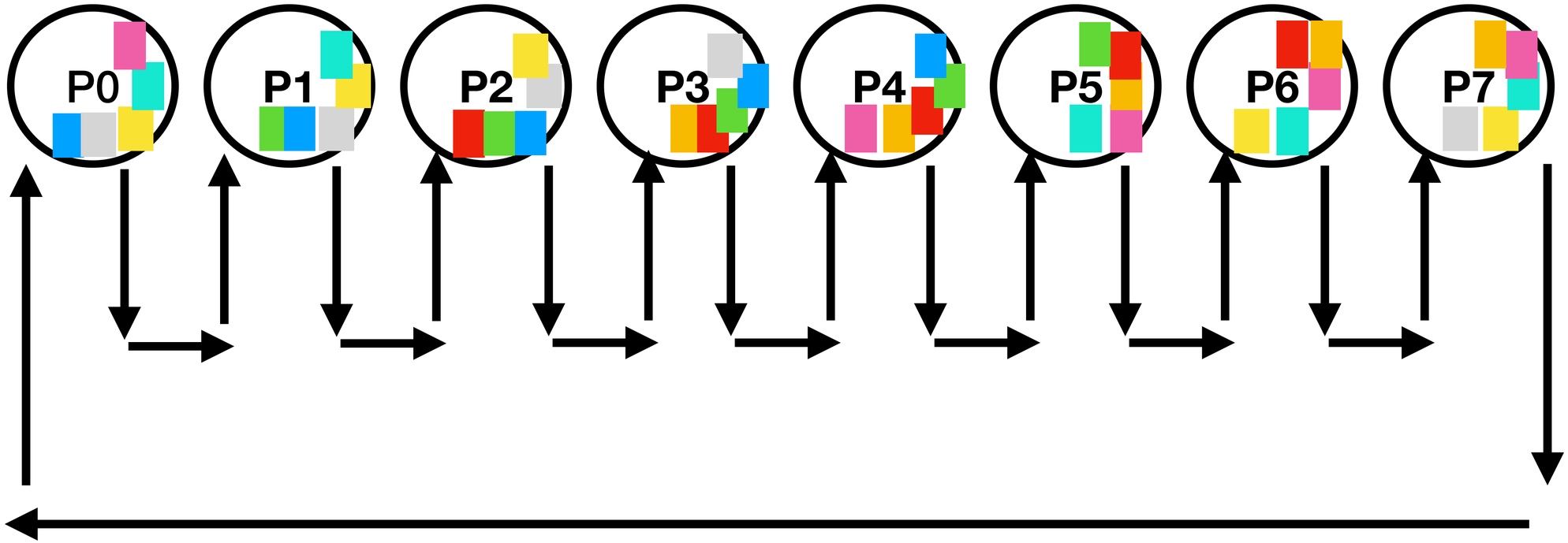
Scatter + Allgather



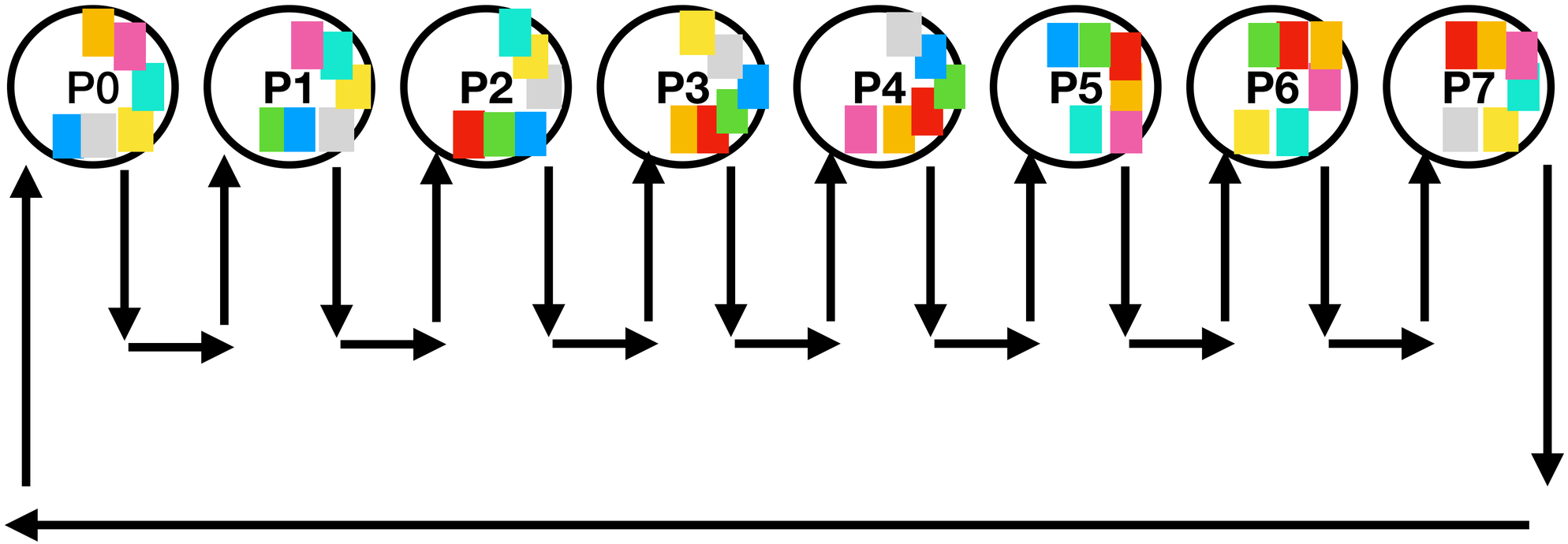
Scatter + Allgather



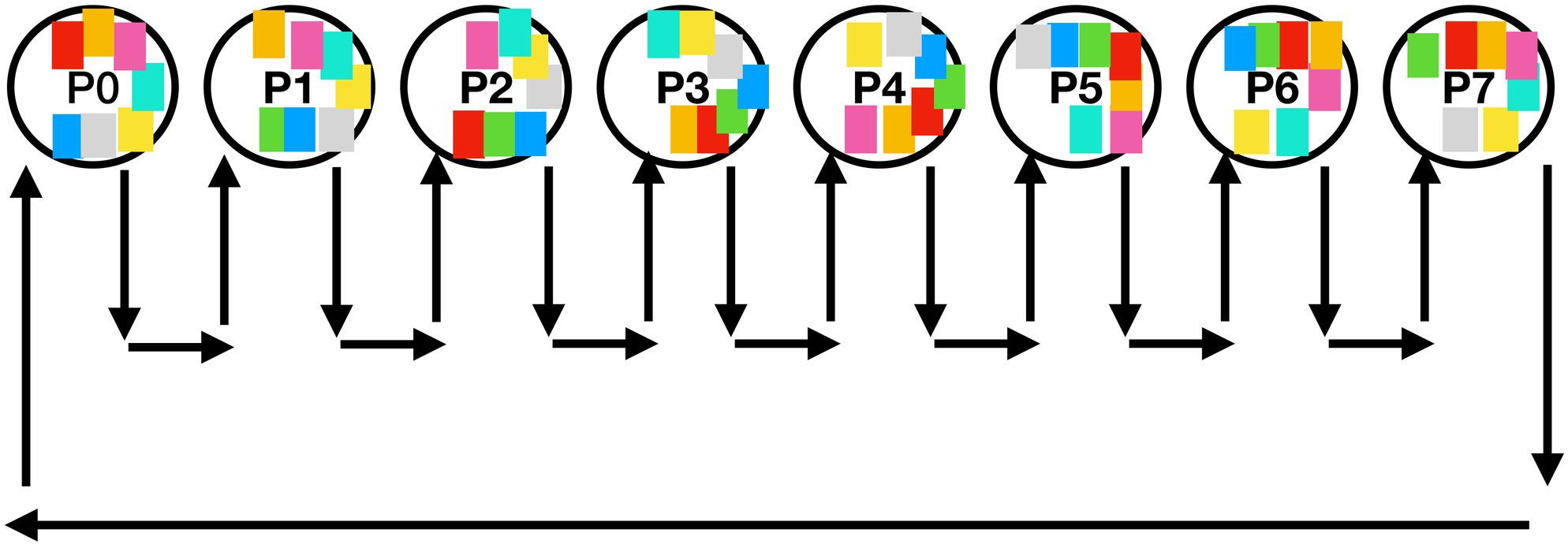
Scatter + Allgather



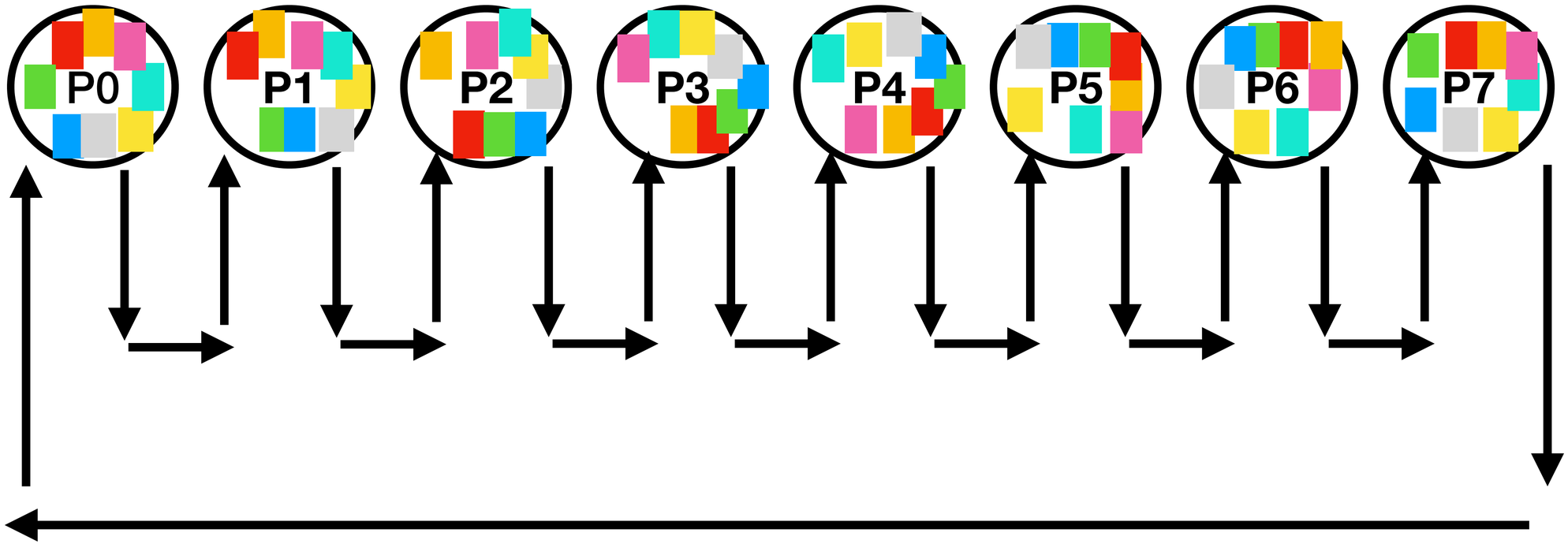
Scatter + Allgather



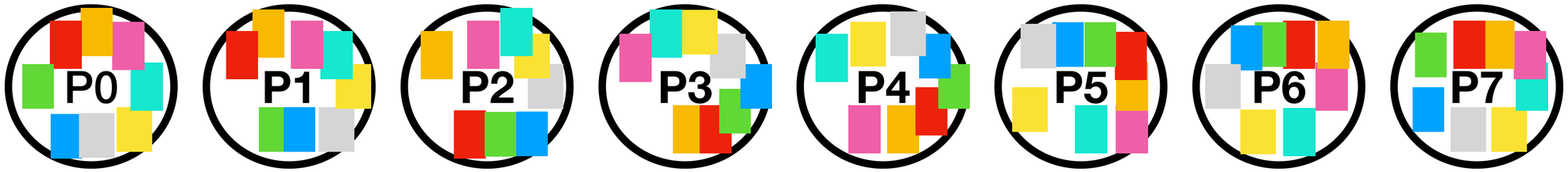
Scatter + Allgather



Scatter + Allgather

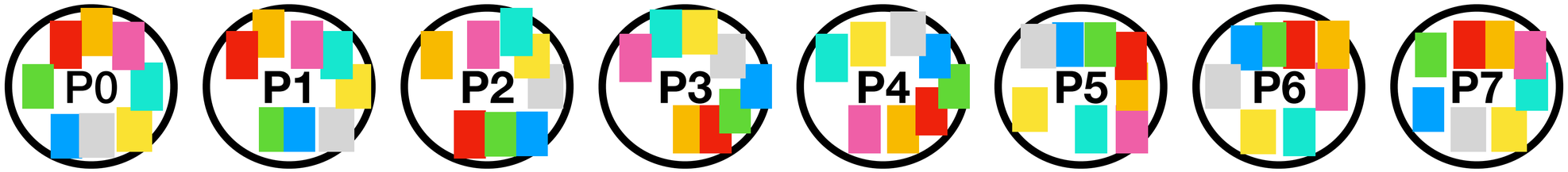


Scatter + Allgather



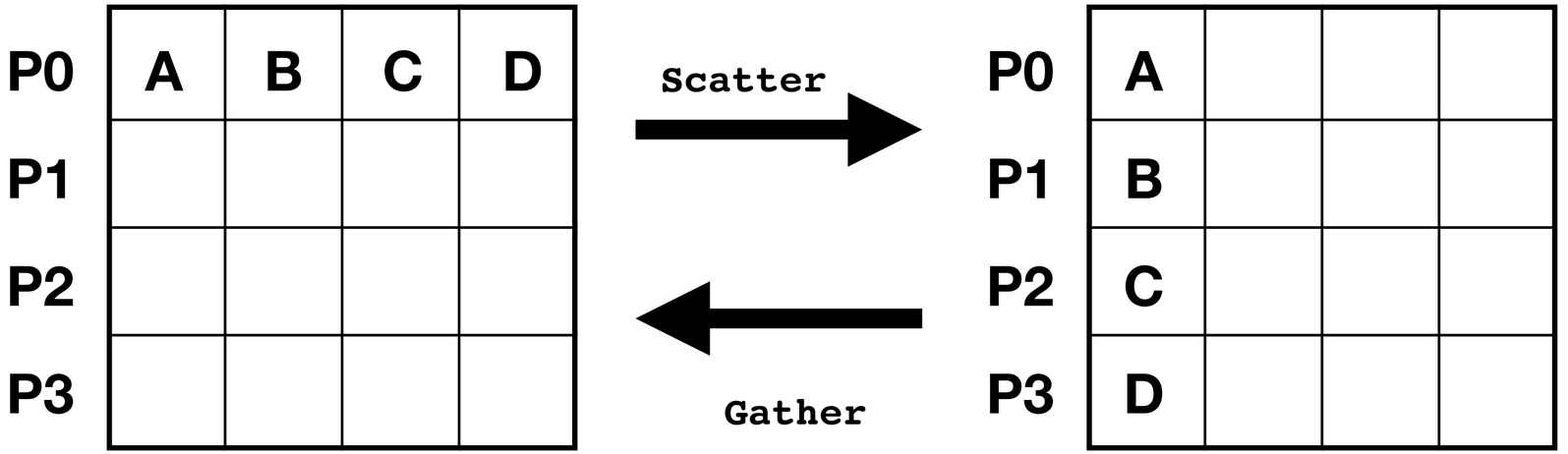
- Scatter has log(num_procs) steps
- Allgather ring algorithm has num_procs - 1 steps
- Scatter sends a total of n values
- Allgather ring algorithm sends n/p values at each step
- $T = (\log_2(p) + p - 1) \cdot \alpha + 2\frac{p-1}{p} n\beta$

Why is Scatter+Allgather ever used?



- Binomial Tree : $T = \log_2(p) \cdot (\alpha + \beta n)$
- Scatter + Allgather : $T = (\log_2(p) + p - 1) \cdot \alpha + 2\frac{p-1}{p} n\beta$

Scatter / Gather



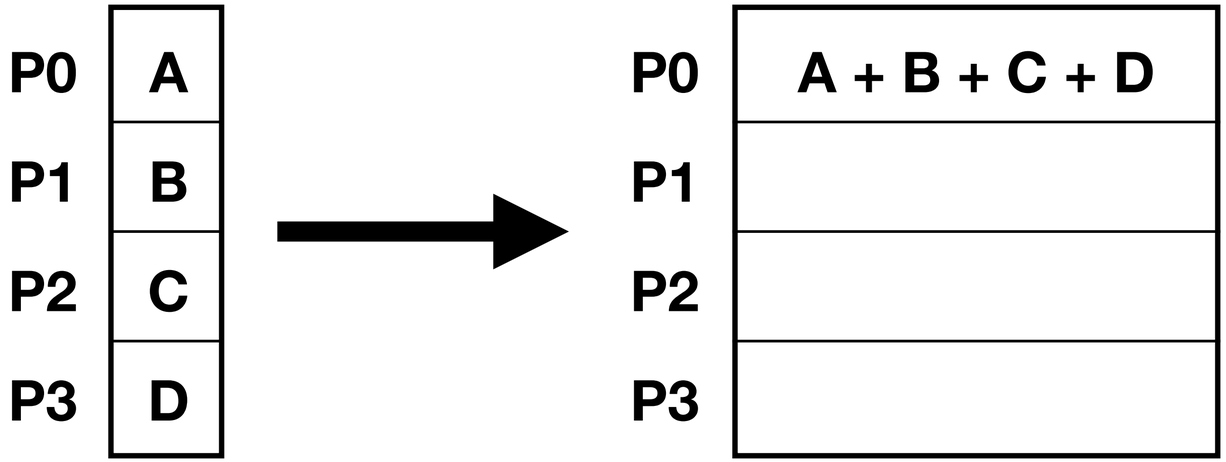
- Scatter : Process 0 starts with values A, B, C, D. After scatter, each process holds one of these values.

- Gather : Each process holds one value (A, B, C, or D). After gather, process 0 holds all of these values.

Scatter / Gather

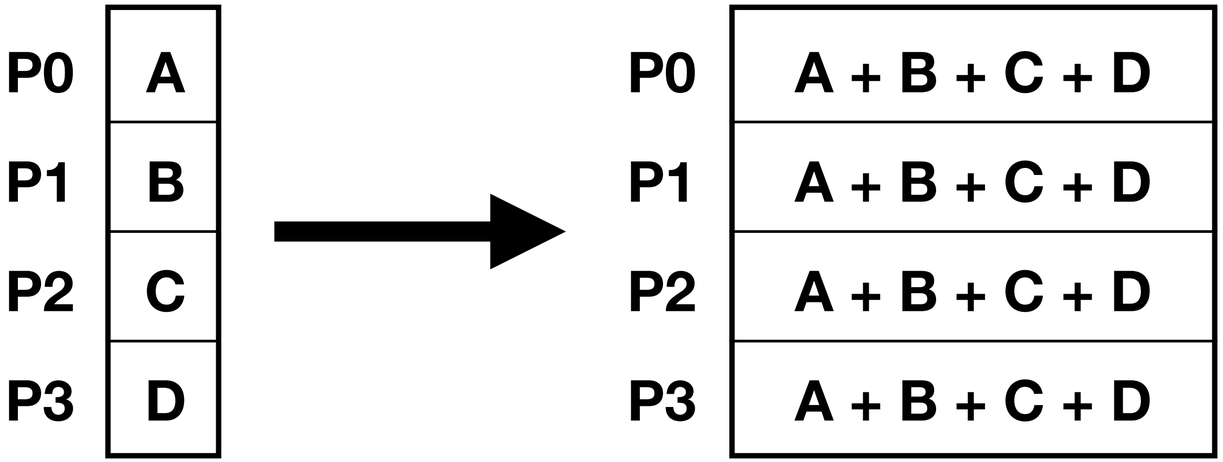
- Both algorithms : binomial trees, just in opposite order
- Scatter on previous slides (first step of scatter+allgather broadcast algorithm)
- Gather is just reverse of scatter

Reduce



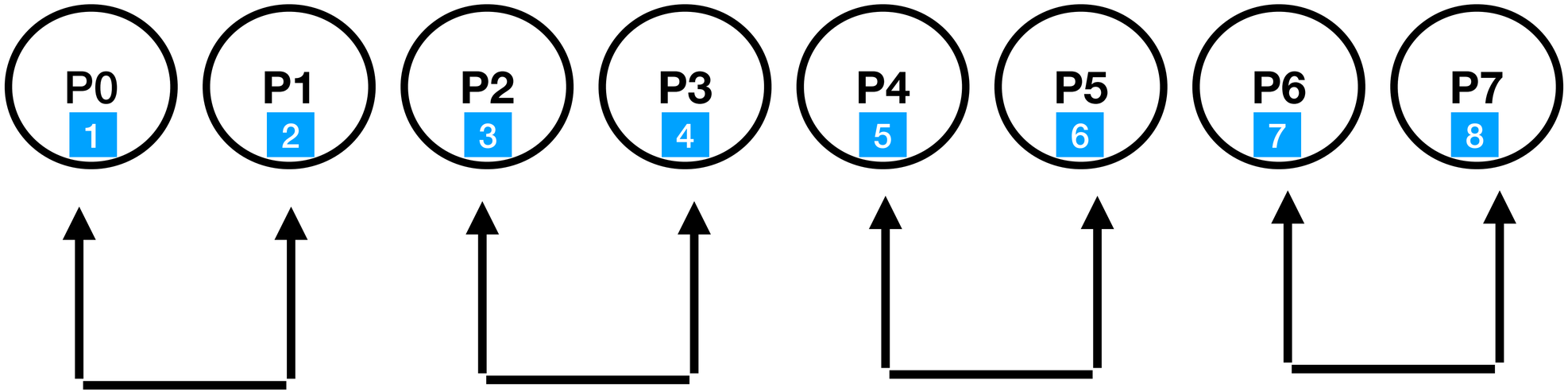
- Each process holds a value (A, B, C, or D)
- After reduce, one process holds the sum of all values
- Similarly, could hold the max or min of the values
- How do we efficiently reduce? **Again, a binomial tree**

Allreduce

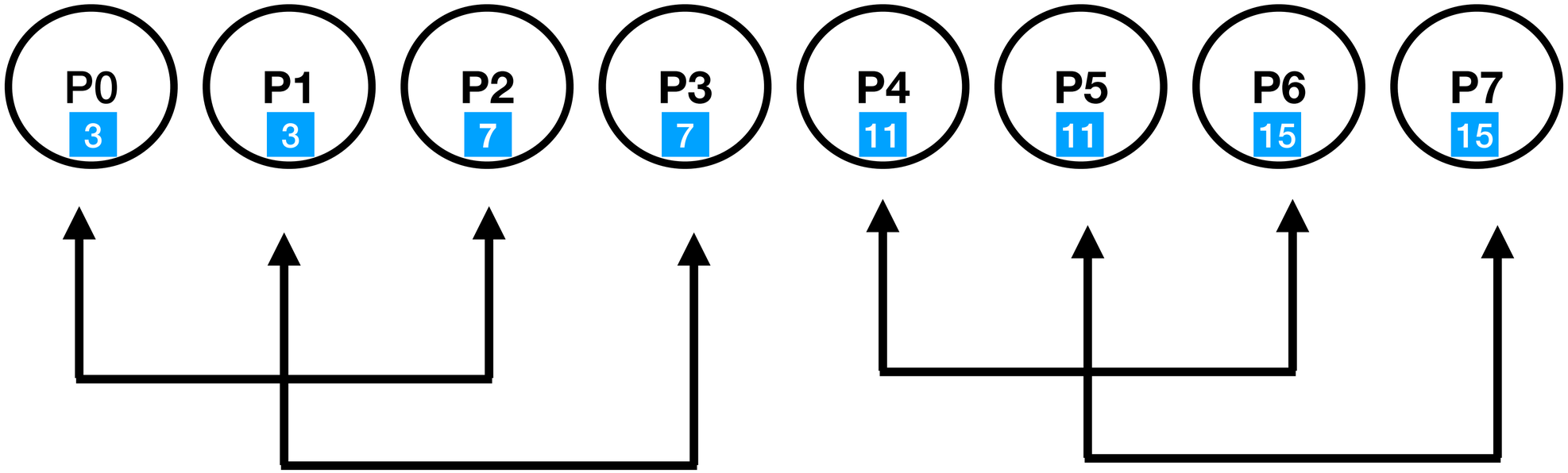


- After all reduce, each process holds sum of the values (or min, max, etc)
- How to efficiently Allreduce?
- Could reduce and then gather. But that's not very efficient.

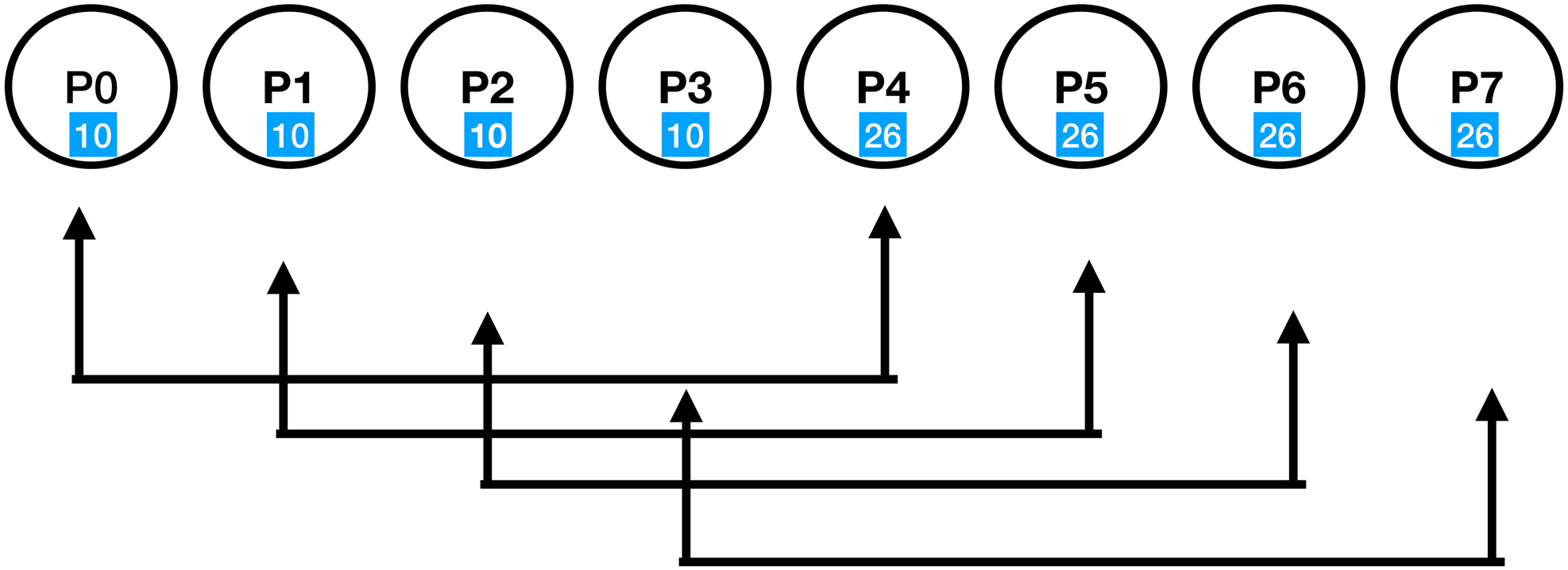
Recursive Doubling



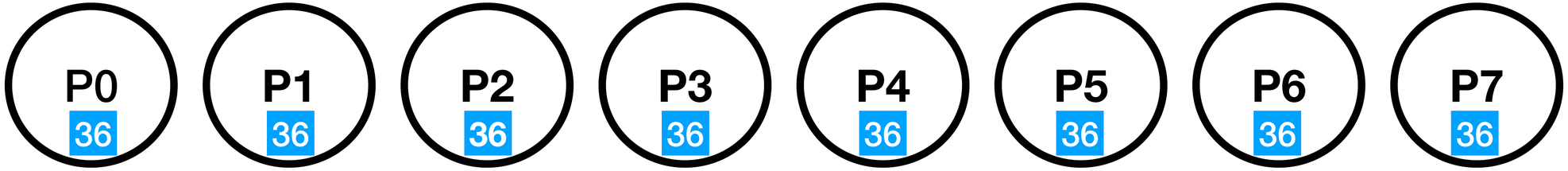
Recursive Doubling



Recursive Doubling

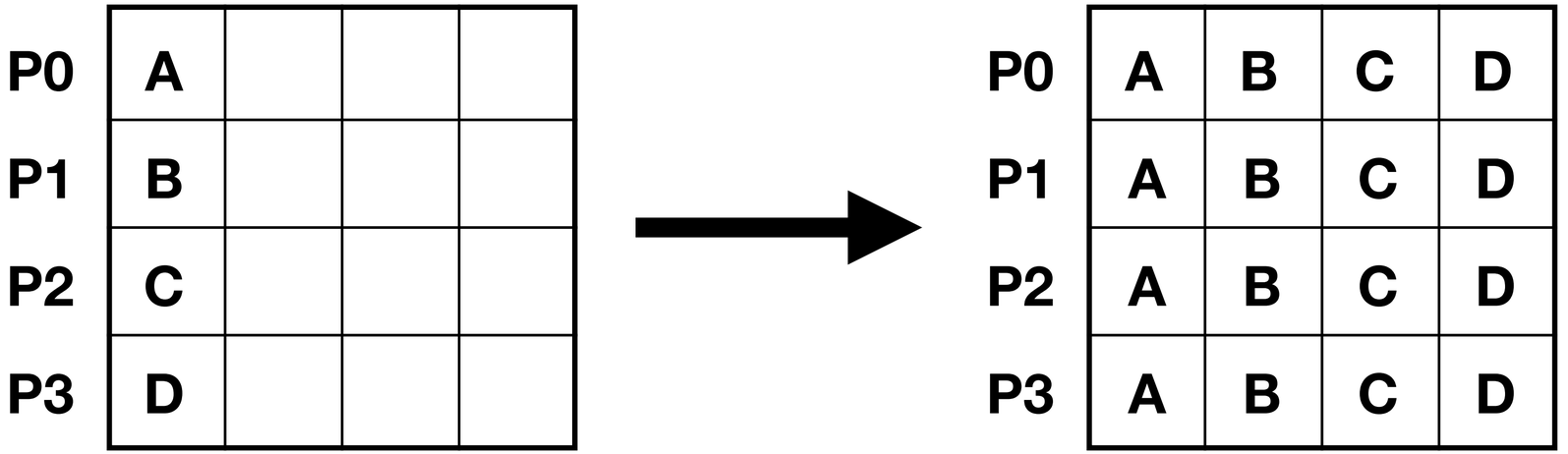


Recursive Doubling



- log(p) steps
- Size of values : n
- $T = (\log_2(p)) \cdot (\alpha + n\beta)$

Allgather

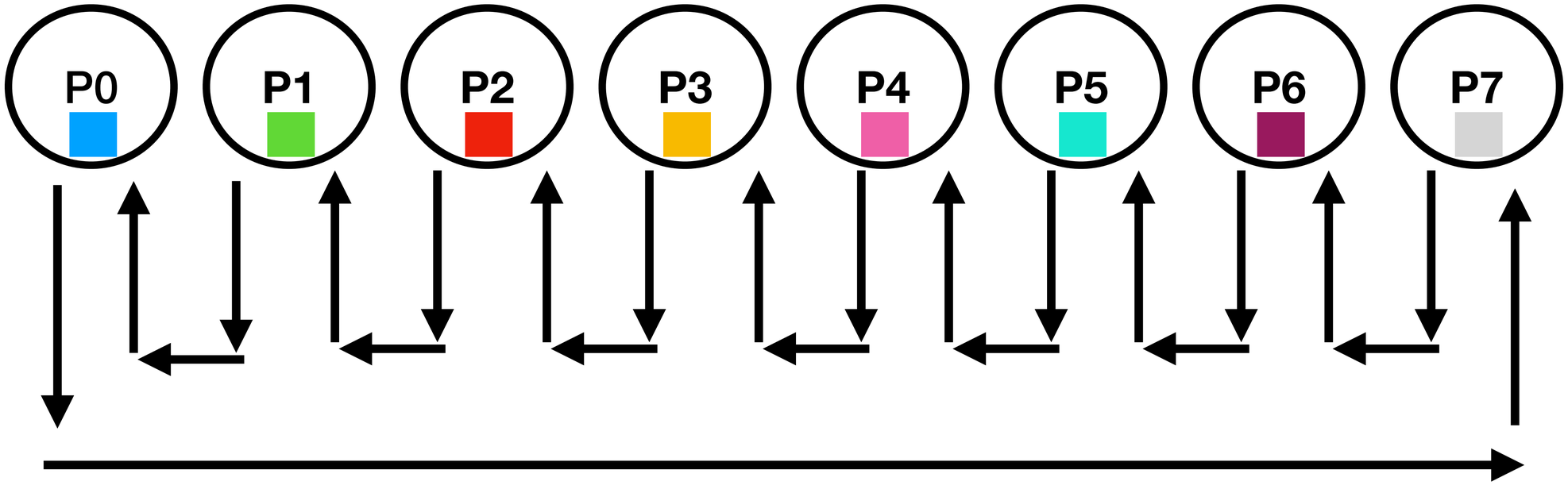


- Each process initially holds a value (A, B, C, or D)
- After allgather, all processes hold all values
- How do we efficiently allgather?

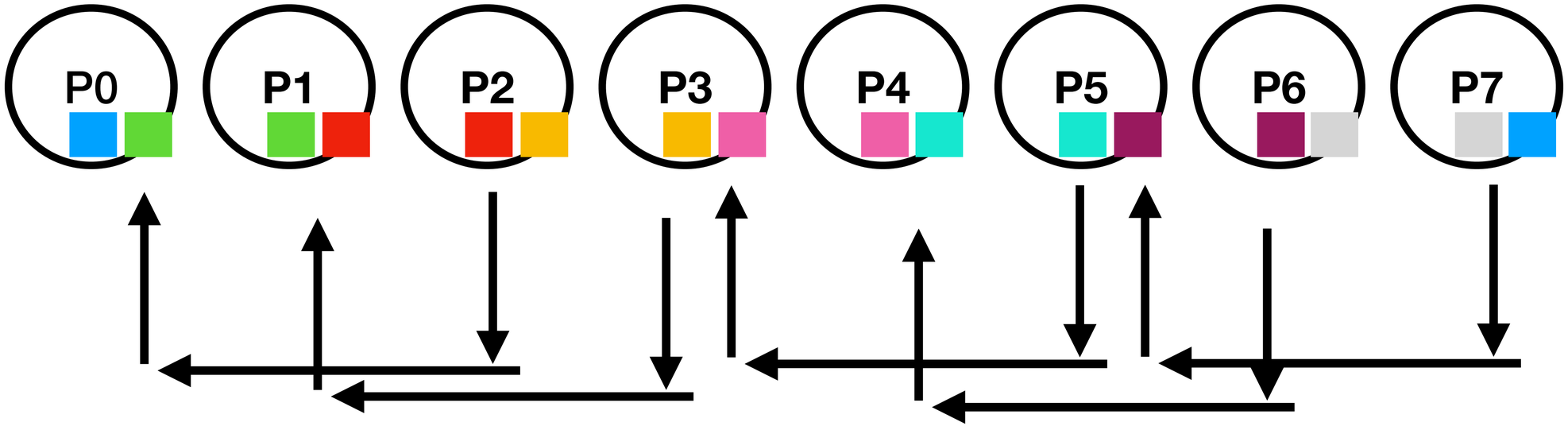
Recursive Doubling

- Could use recursive doubling similar to all reduce.
- log(p) steps
- Message size :
  - Step 1 : n/p
  - Step 2 : 2n/p
  - Last step : 2^{log(p)-1}*n/p
- $T = \log_2(p) \cdot \alpha + \frac{p-1}{p} n \cdot \beta$

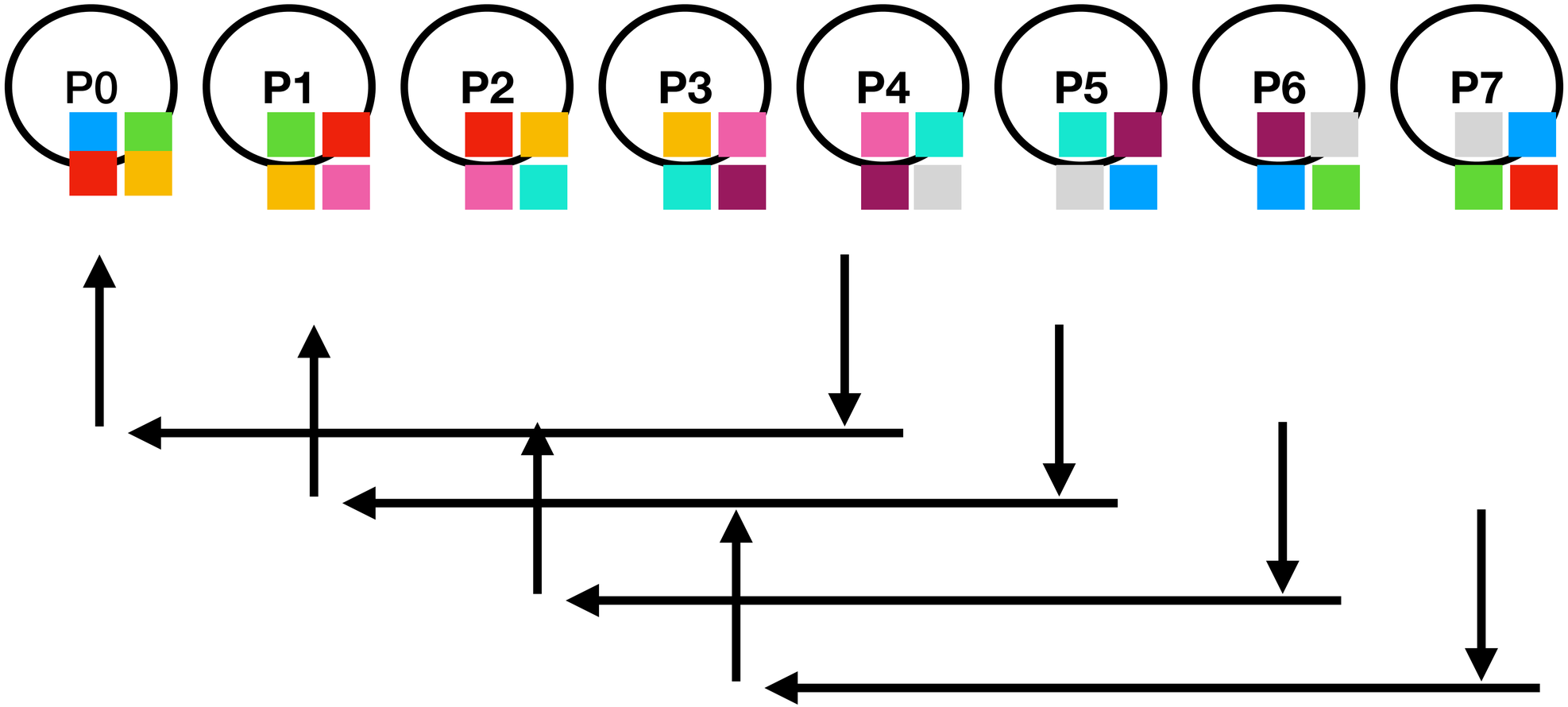
The Bruck Algorithm



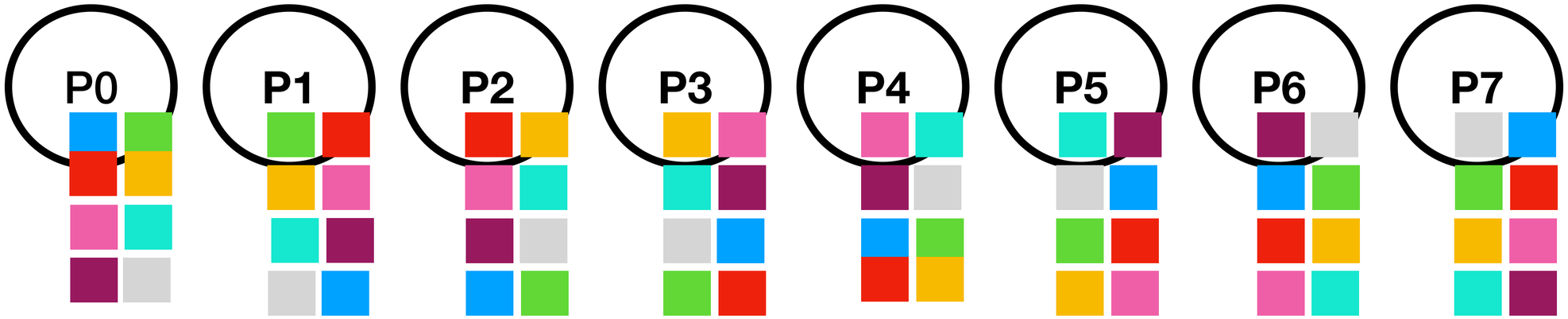
The Bruck Algorithm



The Bruck Algorithm



The Bruck Algorithm



- Same number of steps and message sizes as recursive doubling
- Can anyone spot why this algorithm is preferred?

Other Collectives

- **Barrier :** wait until all processes reach the barrier before any process moves on
  - Dissemination (similar to Bruck algorithm)

Other Collectives

P0	A0	A1	A2	A3
P1	B0	B1	B2	B3
P2	C0	C1	C2	C3
P3	D0	D1	D2	D3

P0	A0	B0	C0	D0
P1	A1	B1	C1	D1
P2	A2	B2	C2	D2
P3	A3	B3	C3	D3

- **Alltoall :** send portion of data to each process (similar to every process doing a scatter)
  - Bruck algorithm for small messages
  - Pairwise exchange for large messages (in step k, send to rank + k, receive from rank - k)
  - **Alltoall is very expensive… avoid when possible**