CSE 6220/CX 4220
Introduction to HPC

# Lecture 12: Sample Sort

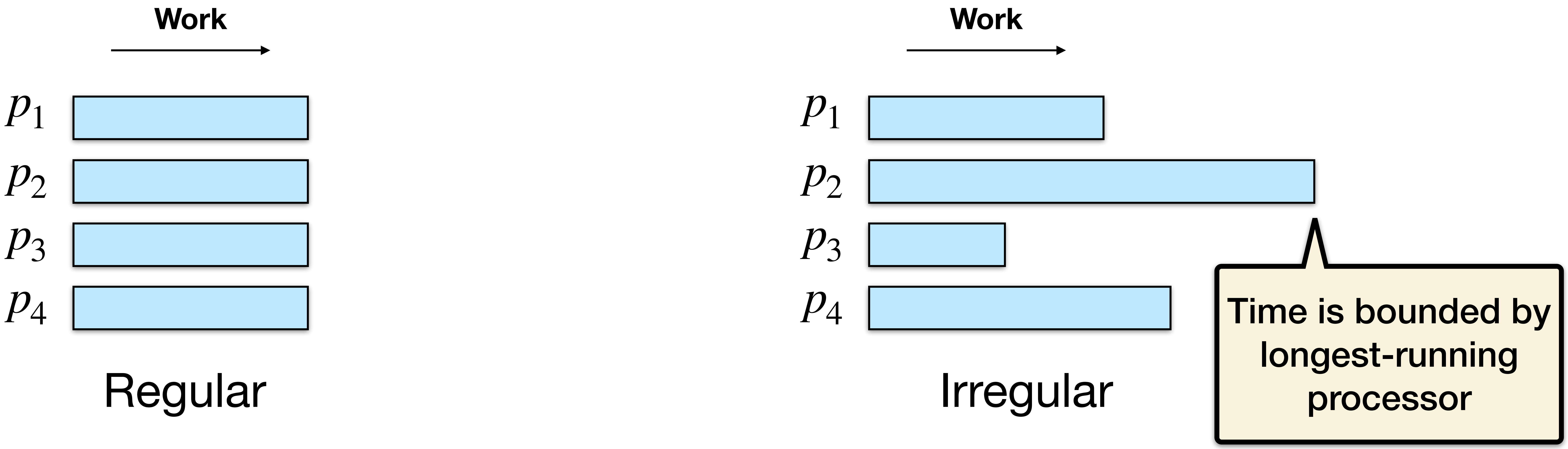
**Helen Xu**
**hxu615@gatech.edu**

Georgia Tech College of Computing
School of Computational Science and Engineering

Slide credits mostly to Prof. Srinivas Aluru, with major contributions from Patrick Flick

# Regular vs Irregular Algorithms

- Regular algorithms (e.g., bitonic sort) **the same amount** of data / work on each processor
- Irregular algorithms (e.g., sample sort) have **variable amounts** of data / work on each processor.



# Trading Some Computation for Communication

- To use parallel computing in practice, we want to **mitigate the communication cost** relative to the "ideal" parallel algorithm time $T_1/p$.
- Computation time is relatively cheap compared to communication.
- Idea: Do some extra **computation in exchange for communication**.

$$1 < \mu < \tau$$

# Recap: Bitonic Sorting

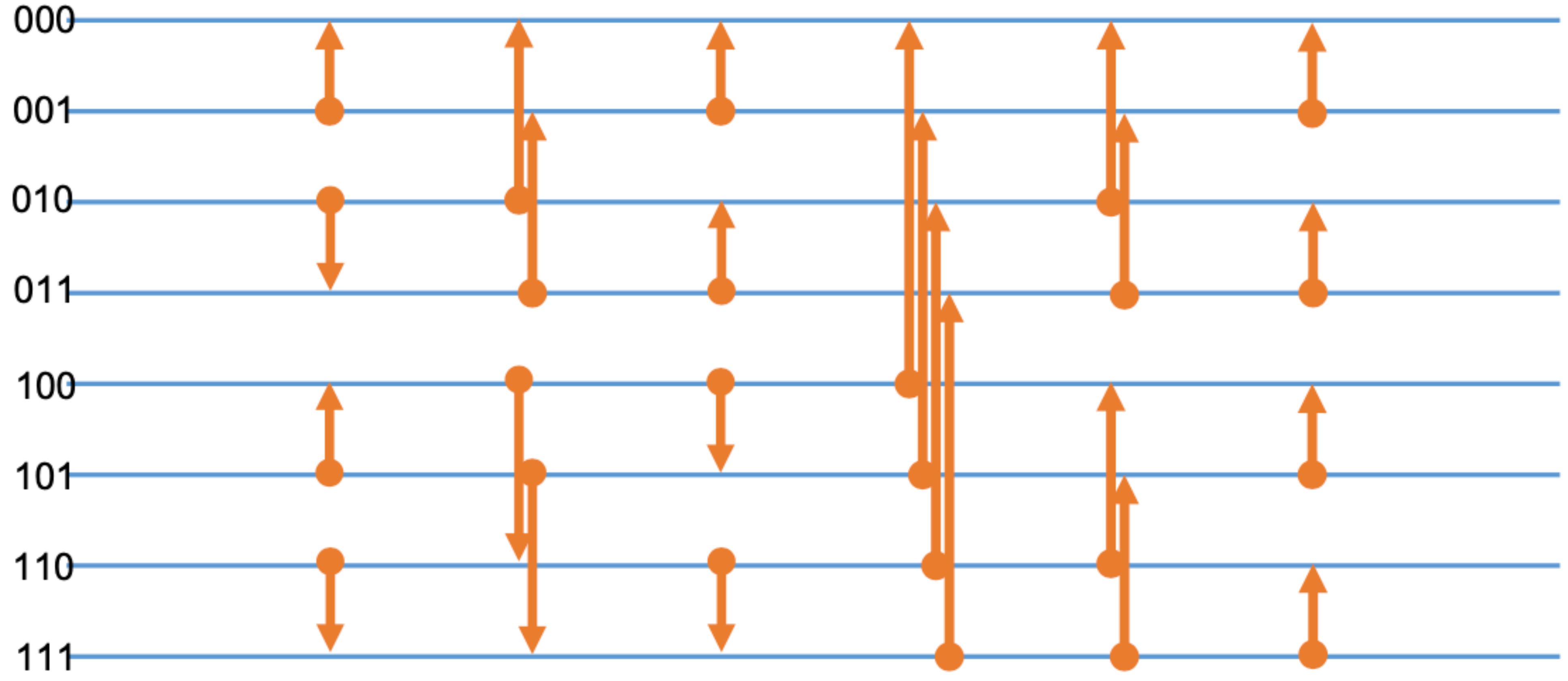
Second term is not ideal - may dominate for large p

- Comp. Time

$$O\left(\frac{n}{p}\log\frac{n}{p} + \boxed{\frac{n}{p}\log^2 p}\right)$$

- Comm. Time

$$O\left(\tau\log^2 p + \boxed{\mu\frac{n}{p}\log^2 p}\right)$$




Also shows up in the communication time

# Lower Bounds for Parallel Sorting

How would you prove lower bounds on communication / computation?

**(serial sorting lower bound in the comparison model is $\Omega(n \lg n)$)**

**Hint: bound computation via $n, p$**

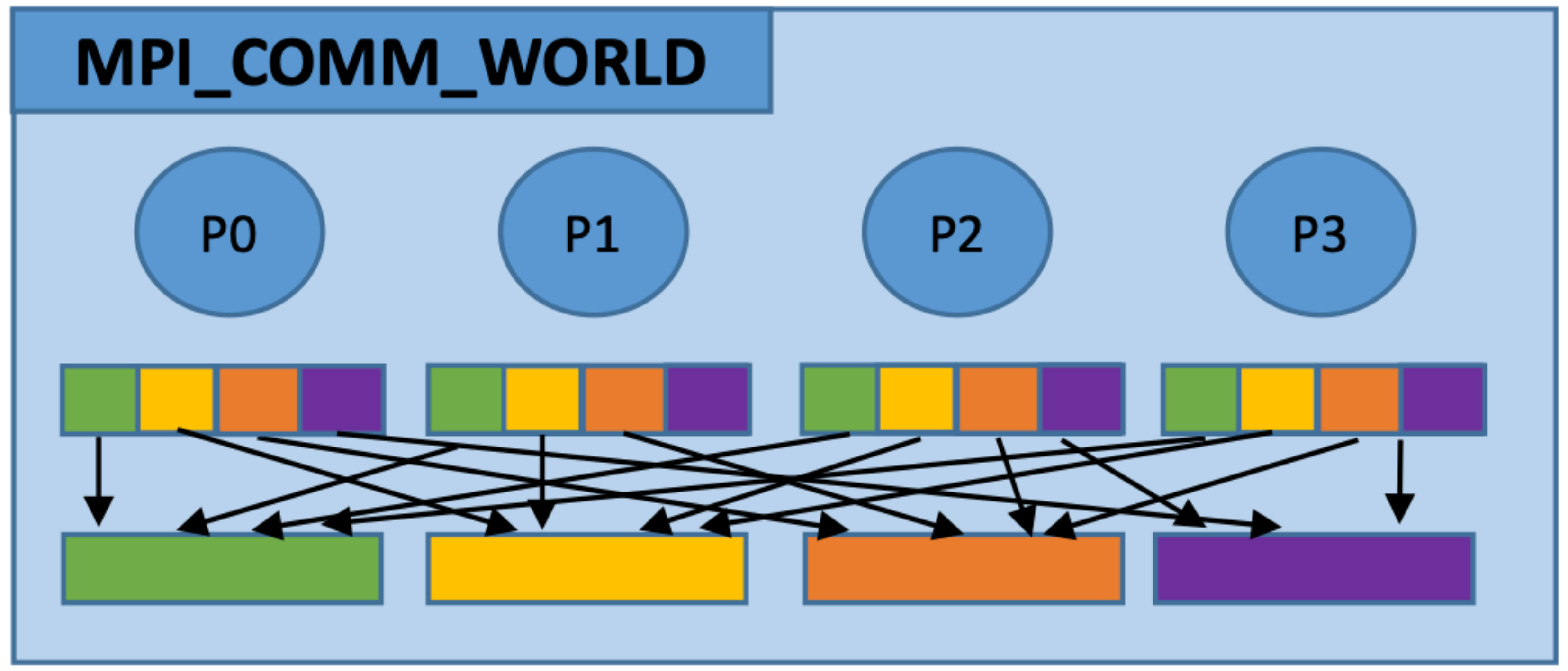
**bound communication via $\tau, \mu, n, p$**

# Lower Bounds for Parallel Sorting

- Sequential sorting lower bound
  - Comparison based sorting (using only <, > , ==)
  - Sorting $n$ elements requires at least $\Omega(n \log n)$ comparisons

- Distributed memory parallel sorting
  - Assuming $\frac{n}{p}$ elements per processor
  - Worst-case: all $\frac{n}{p}$ elements belong to other processes
  - Average case: $\frac{n}{p}\left(1 - \frac{1}{p}\right) = \Theta\left(\frac{n}{p}\right)$
  - Thus any sorting algorithm has communication complexity at least:

$$\Omega\left(\mu \frac{n}{p}\right)$$




- Lower bound:
  - Computation $\Omega\left(\frac{n \log n}{p}\right)$
  - Communication $\Omega\left(\mu \frac{n}{p}\right)$

# Lower Bounds for Parallel Sorting

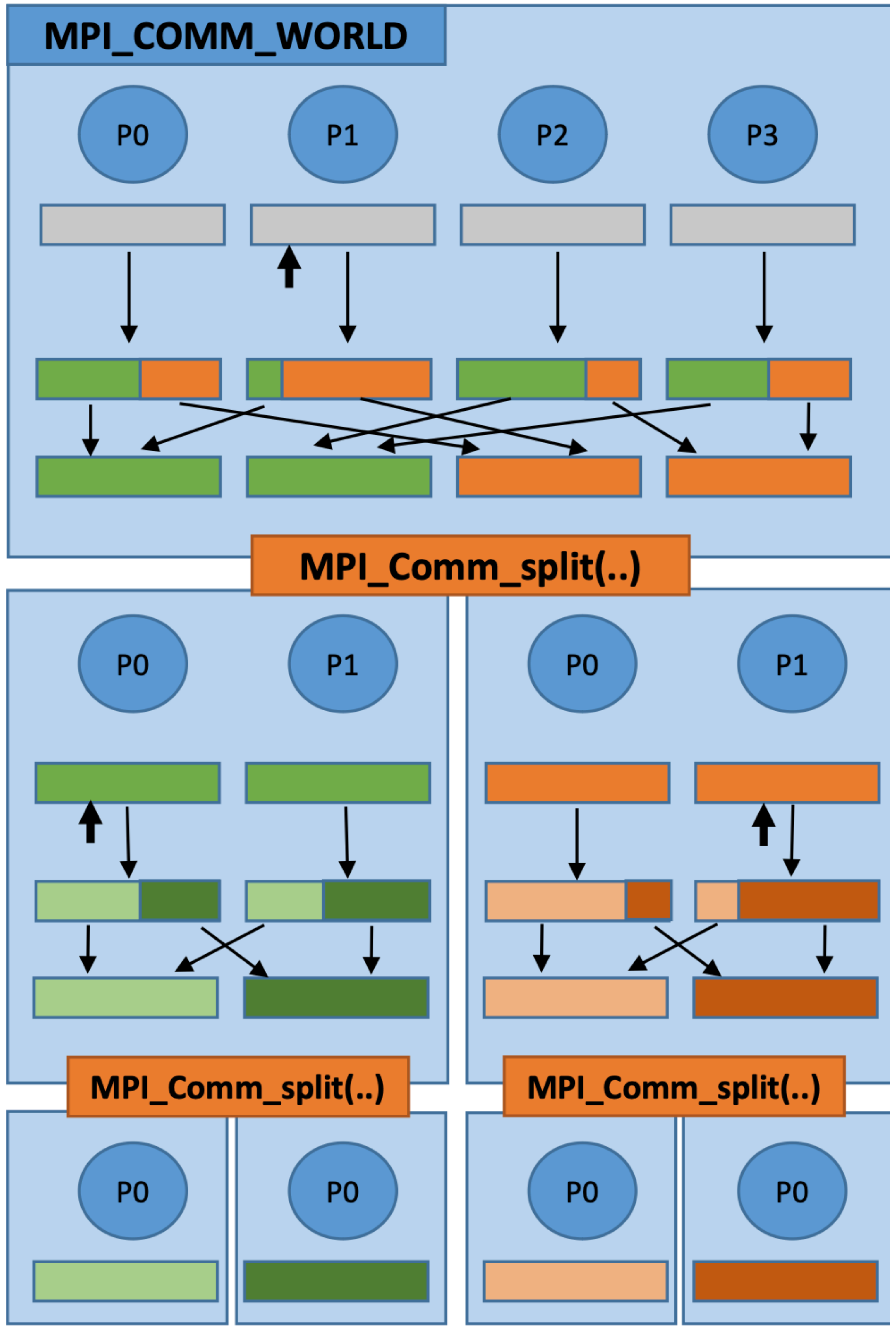
- Lower bound for distributed memory parallel sort:
  - Computation: $\Omega\left(\frac{n \log n}{p}\right)$
  - Communication: $\Omega\left(\mu \frac{n}{p}\right)$
- Bitonic Sort:
  - Computation: $O\left(\frac{n}{p} \log \frac{n}{p} + \frac{n}{p} \log^2 p\right)$
  - Communication: $O\left(\log^2 p \left(\tau + \mu \frac{n}{p}\right)\right)$ => $\log^2 p \times$ data movements
- Can we do better?
  - Can we somehow determine which processor each element should go to?

Ideal case: everyone gets equal blocks from everyone, and the blocks happen to be in order

# Parallel Quicksort

## Parallel Quicksort

1. Pick pivot and broadcast $O(\log p(\tau + \mu))$
2. Partition locally $O\left(\frac{n}{p}\right)$
3. Re-arrange partitions $O\left(\tau + \mu\frac{n}{p}\right)$
4. Split processors and recurse
5. If $p$ = 1, locally sort




Best/expected case (assuming no imbalance):

- $O(\log p)$ iterations before local sort
- Computation: $O\left(\frac{n}{p}\log p + \frac{n}{p}\log\frac{n}{p}\right) = O\left(\frac{n\log n}{p}\right)$
- Communication: $O\left(\tau\lg^2 p + \mu\frac{n}{p}\log p\right)$

Worst case:

- Much worse!

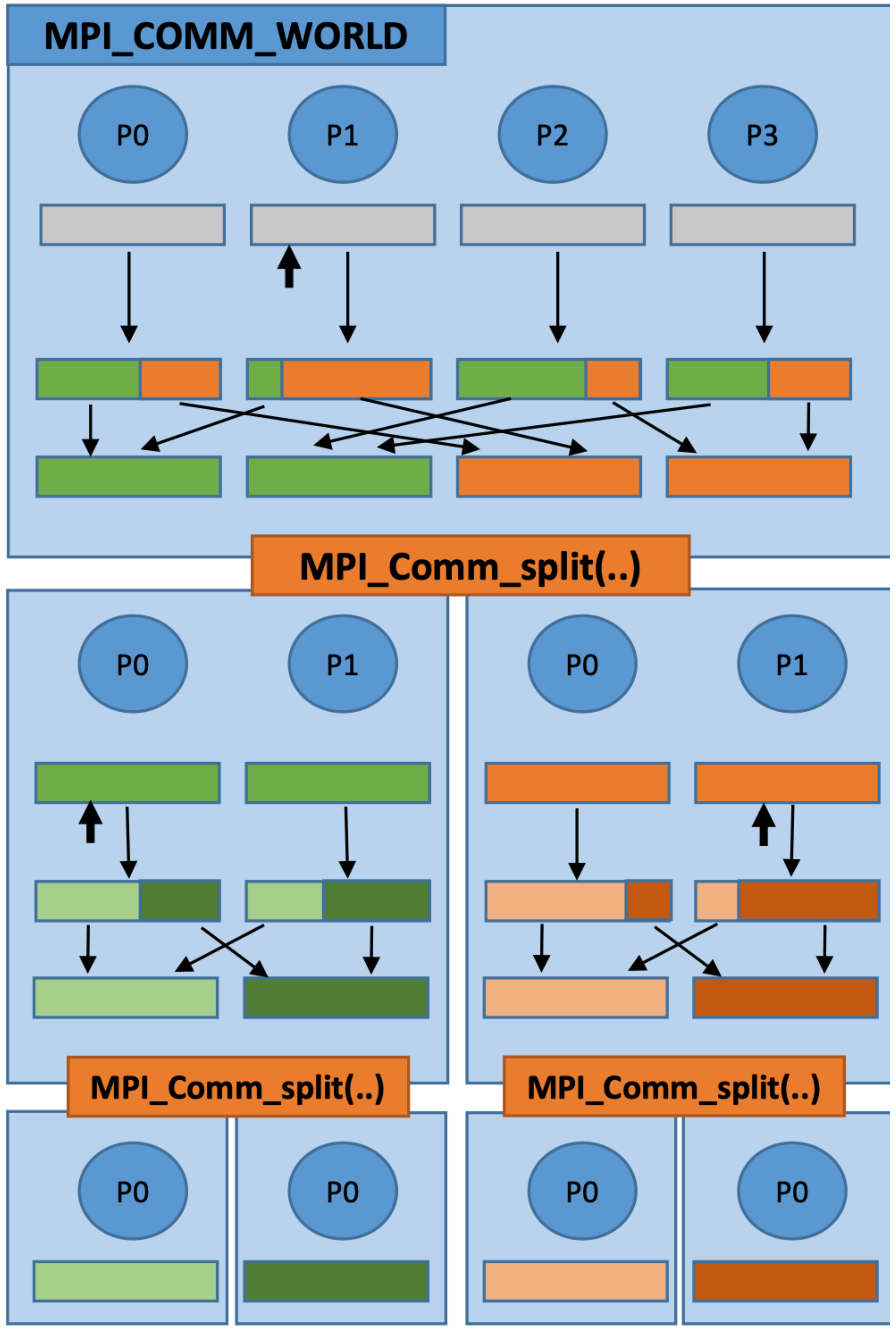
# Parallel Quicksort

Lower bounds:
Computation = $\Omega((n \lg n)/p)$
Communication = $\Omega((\mu n)/p)$

Could also use the same RNG

Parallel Quicksort

1. Pick pivot and broadcast $O(\log p(\tau + \mu))$
2. Partition locally $O\left(\frac{n}{p}\right)$
3. Re-arrange partitions $O\left(\tau + \mu\frac{n}{p}\right)$
4. Split processors and recurse
5. If $p$ = 1, locally sort




Best/expected case (assuming no imbalance):

- $O(\log p)$ iterations before local sort
- Computation: $O\left(\frac{n}{p}\log p + \frac{n}{p}\log\frac{n}{p}\right) = O\left(\frac{n\log n}{p}\right)$
- Communication: $O\left(\tau \lg^2 p + \mu\frac{n}{p}\log p\right)$

Worst case:

- Much worse!

# Parallel Quicksort

Lower bounds:
Computation = $\Omega((n \lg n)/p)$
Communication = $\Omega((\mu n)/p)$

Could also use the same RNG

## Parallel Quicksort

1. Pick pivot and broadcast $O(\log p(\tau + \mu))$
2. Partition locally $O\left(\frac{n}{p}\right)$
3. Re-arrange partitions $O\left(\tau + \mu\frac{n}{p}\right)$
4. Split processors and recurse
5. If $p$ = 1, locally sort

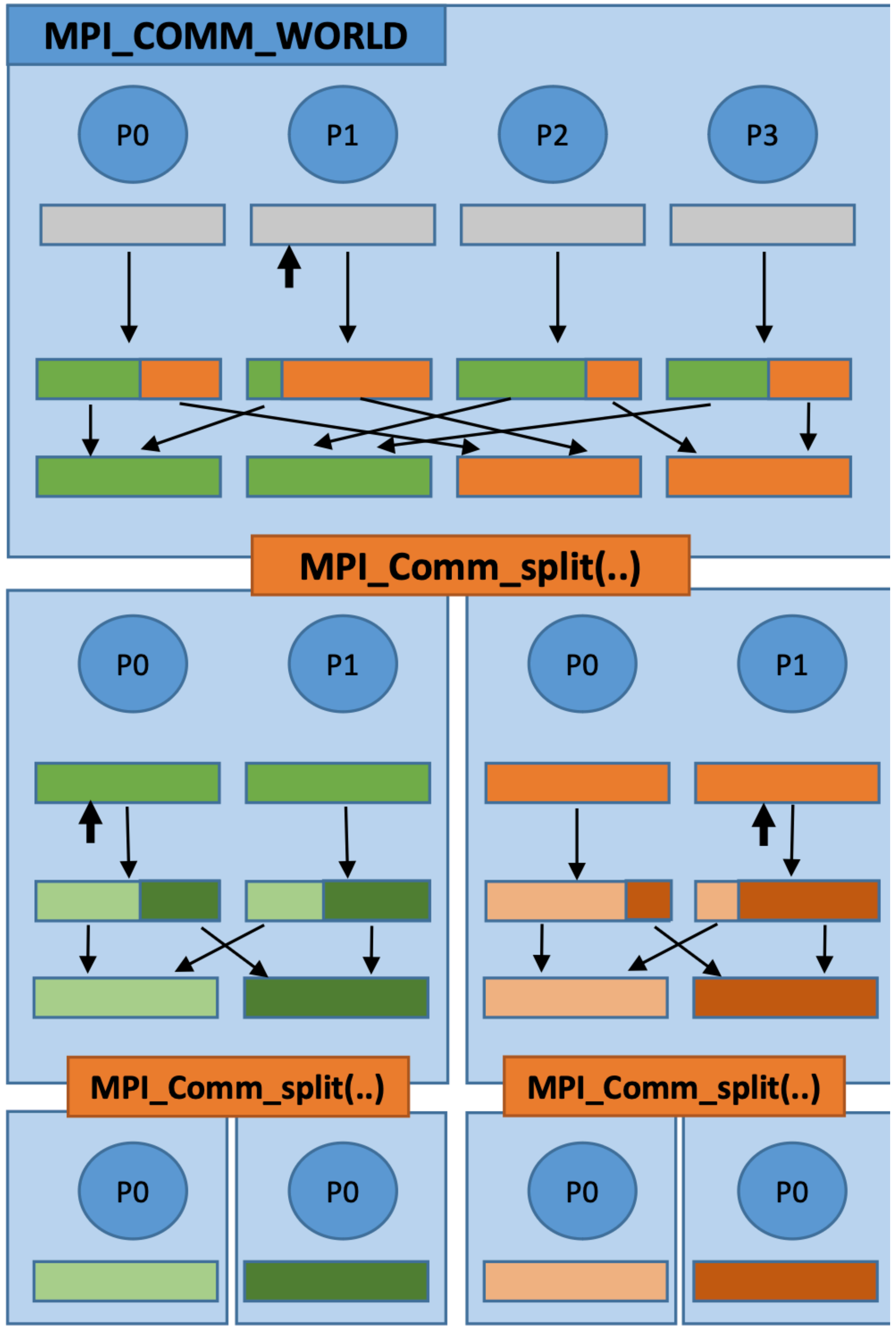
Ideally, choose the global median

Best/expected case (assuming no imbalance):

- $O(\log p)$ iterations before local sort
- Computation: $O\left(\frac{n}{p}\log p + \frac{n}{p}\log\frac{n}{p}\right) = O\left(\frac{n \log n}{p}\right)$
- Communication: $O\left(\tau \lg^2 p + \mu\frac{n}{p}\log p\right)$

Worst case:

- Much worse!



# Parallel Quicksort

Lower bounds:
Computation = $\Omega((n \lg n)/p)$
Communication = $\Omega((\mu n)/p)$

Could also use the same RNG

## Parallel Quicksort

1. Pick pivot and broadcast $O(\log p(\tau + \mu))$
2. Partition locally $O\left(\frac{n}{p}\right)$
3. Re-arrange partitions $O\left(\tau + \mu \frac{n}{p}\right)$
4. Split processors and recurse
5. If $p$ = 1, locally sort

Ideally, choose the global median

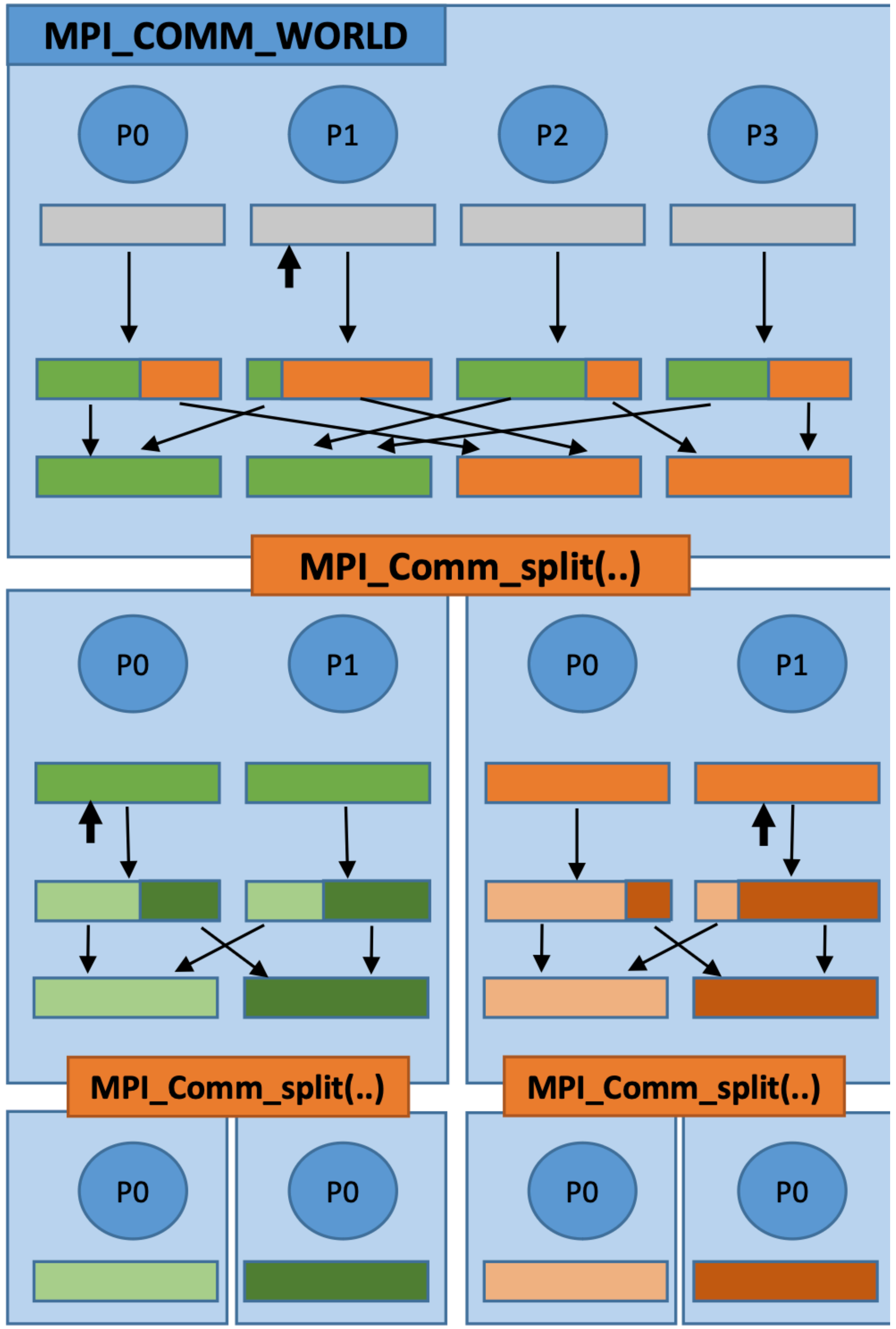
Best/expected case (assuming no imbalance):

- $O(\log p)$ iterations before local sort
- Computation: $O\left(\frac{n}{p}\log p + \frac{n}{p}\log\frac{n}{p}\right) = O\left(\frac{n \log n}{p}\right)$
- Communication: $O\left(\tau \lg^2 p + \mu \frac{n}{p}\log p\right)$

Worst case:

- Much worse!

Can we lower this log p?

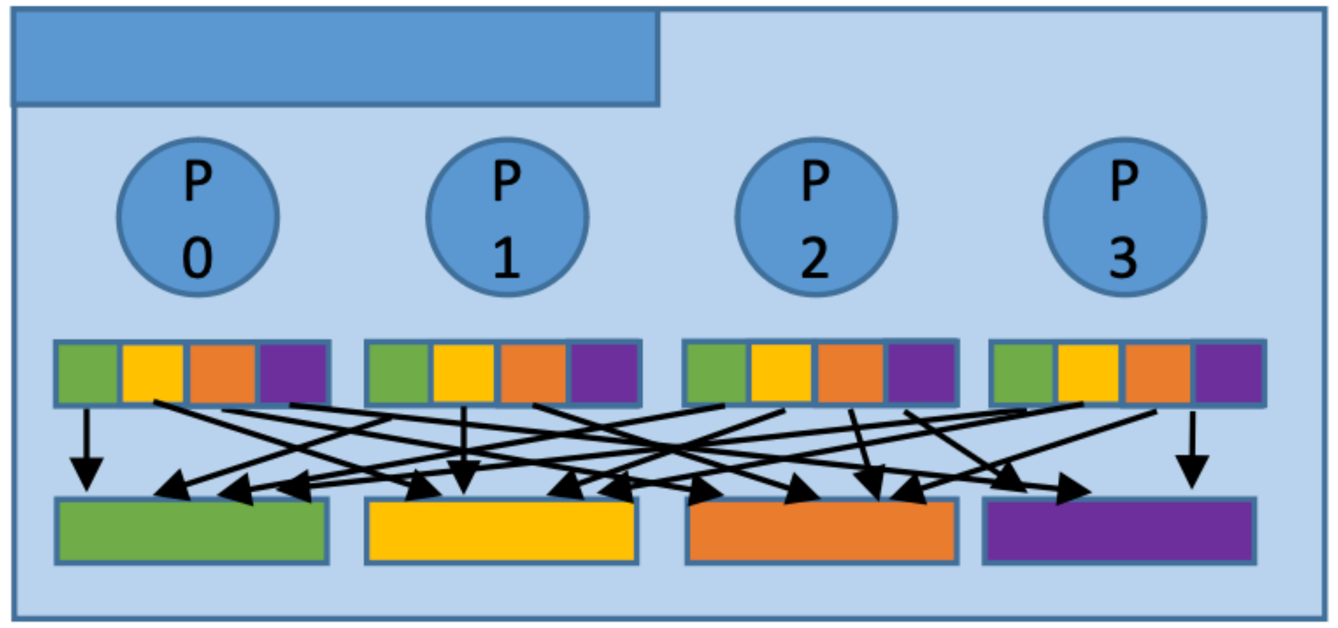


# Sample sort - first attempt

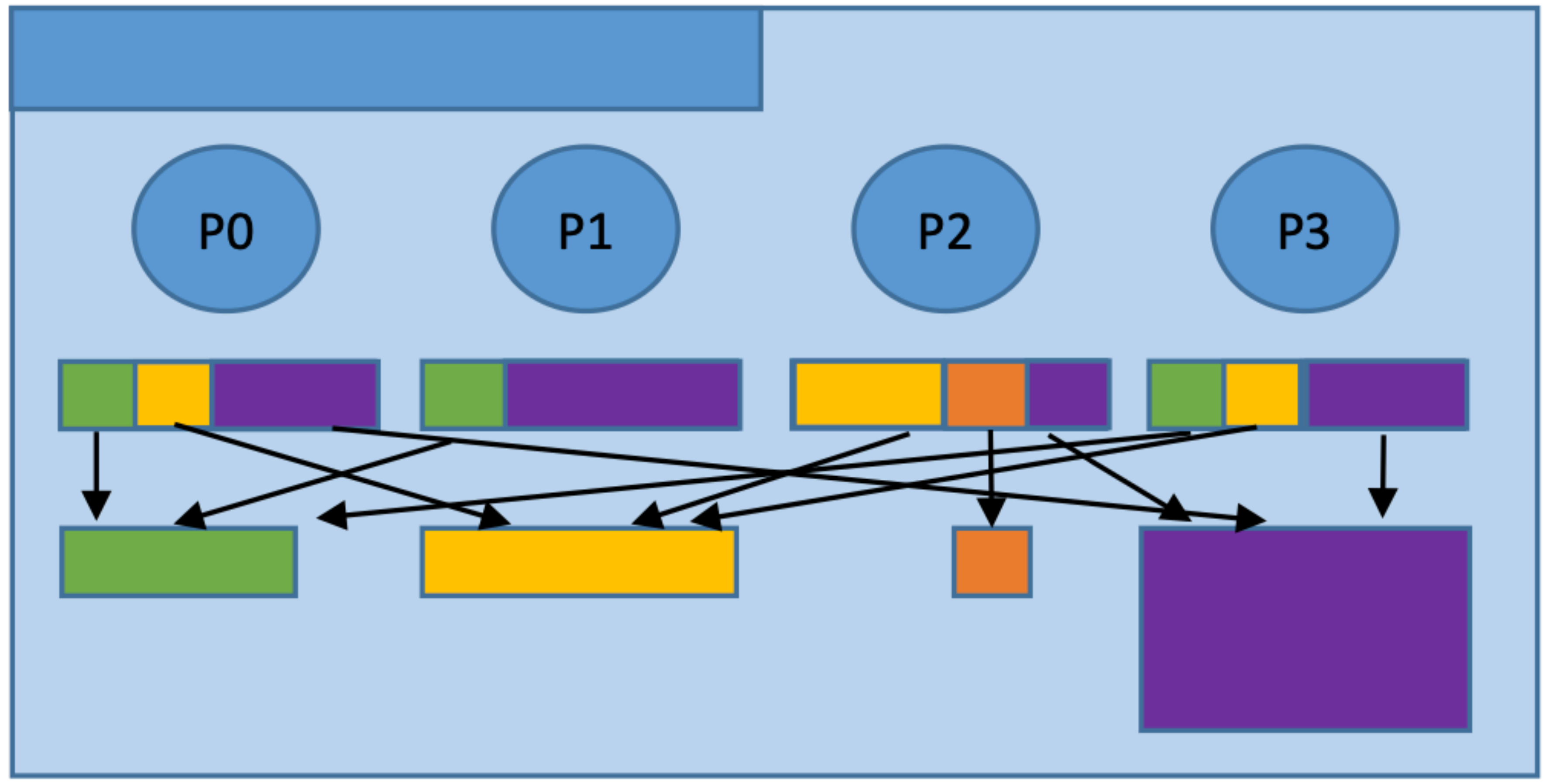
**Lower bounds:**
Computation = $\Omega((n \lg n)/p)$
Communication = $\Omega((\mu n)/p)$

**Attempt:**

1. Sort locally — $O\left(\frac{n}{p}\log\frac{n}{p}\right)$
2. Pick p-1 random samples globally (like quicksort) — $O(p \log p)$
3. Split local sequence into $p$ segments according to samples — $O\left(\frac{n}{p}\right)$
4. All-to-all communication — $\cancel{O\left(\tau p + \mu\frac{n}{p}\right)}$
5. Locally merge p sequences — $\cancel{O\left(\frac{n}{p}\log p\right)}$

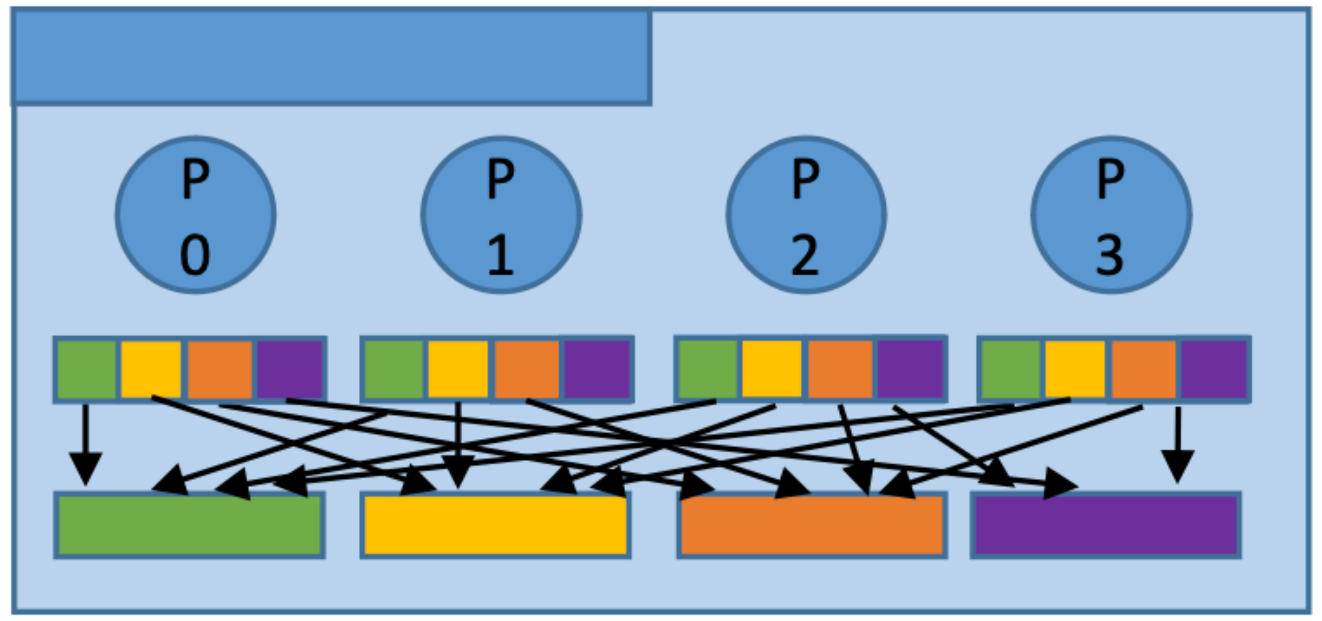



**Problem: worst-case load imbalance up to Ω(n)**



# Ideal sample sort

Lower bounds:
Computation = $\Omega((n \lg n)/p)$
Communication = $\Omega((\mu n)/p)$




## Sample sort

1. Sort locally $O\left(\frac{n}{p}\log\frac{n}{p}\right)$
2. **Find $p-1$ "*good*" splitters**
3. Split local sequence into $p$ segments according to splitters
4. Many-to-many communication $O\left(\tau p + \mu\frac{n}{p}\right)$
5. Locally merge p sequences $O\left(\frac{n}{p}\log p\right)$

## Good splitters:

- Each processor should receive at most $m \leq c\frac{n}{p}$ elements in Step 4.
  - where $c \geq 1$ is a small constant
- Finding splitters should not be too expensive (complexity)

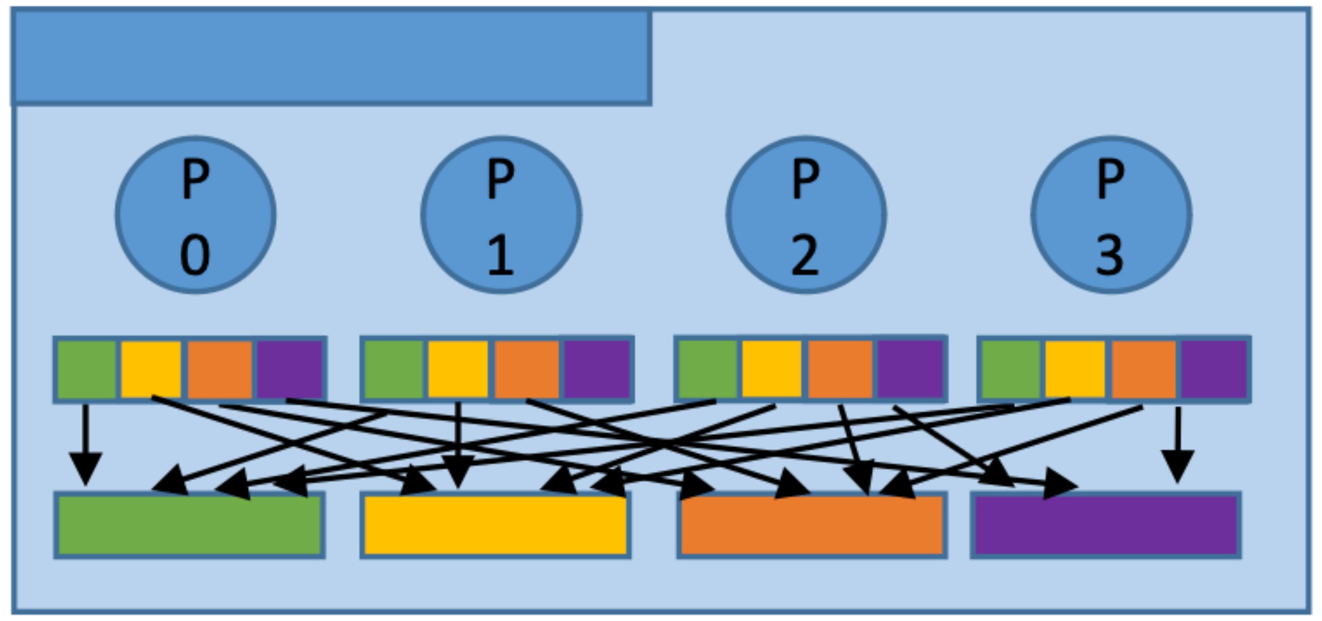
# Ideal sample sort

Lower bounds:
Computation = $\Omega((n \lg n)/p)$
Communication = $\Omega((\mu n)/p)$

**Sample sort**

Ideally deterministic

1. Sort locally $O\left(\frac{n}{p}\log\frac{n}{p}\right)$
2. **Find $p-1$ "*good*" splitters**
3. Split local sequence into $p$ segments according to splitters
4. Many-to-many communication $O\left(\tau p + \mu\frac{n}{p}\right)$
5. Locally merge p sequences $O\left(\frac{n}{p}\log p\right)$




**Good splitters:**

- Each processor should receive at most $m \leq c\frac{n}{p}$ elements in Step 4.
  - where $c \geq 1$ is a small constant
- Finding splitters should not be too expensive (complexity)

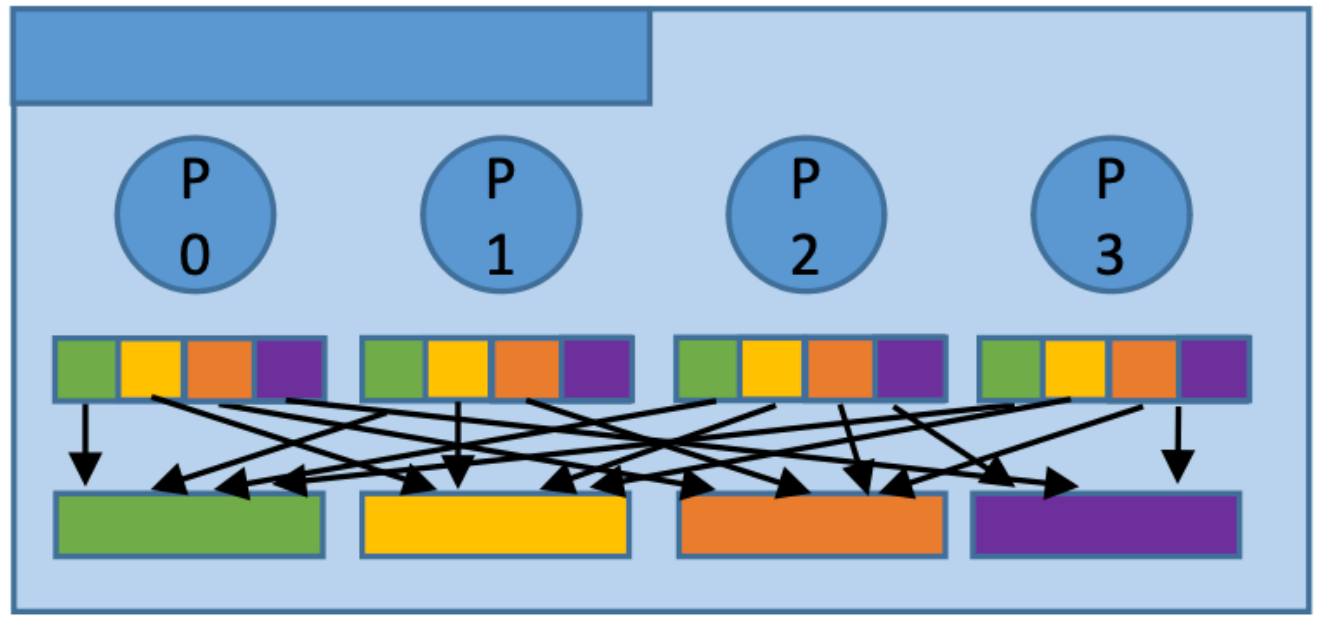
# Ideal sample sort

Lower bounds:
Computation = $\Omega((n \lg n)/p)$
Communication = $\Omega((\mu n)/p)$

**Sample sort**

1. Sort locally $O\left(\frac{n}{p}\log\frac{n}{p}\right)$
2. **Find $p-1$ "*good*" splitters**
3. Split local sequence into $p$ segments according to splitters
4. Many-to-many communication $O\left(\tau p + \mu\frac{n}{p}\right)$
5. Locally merge p sequences $O\left(\frac{n}{p}\log p\right)$

Ideally deterministic




How to efficiently merge p sequences?

**Good splitters:**

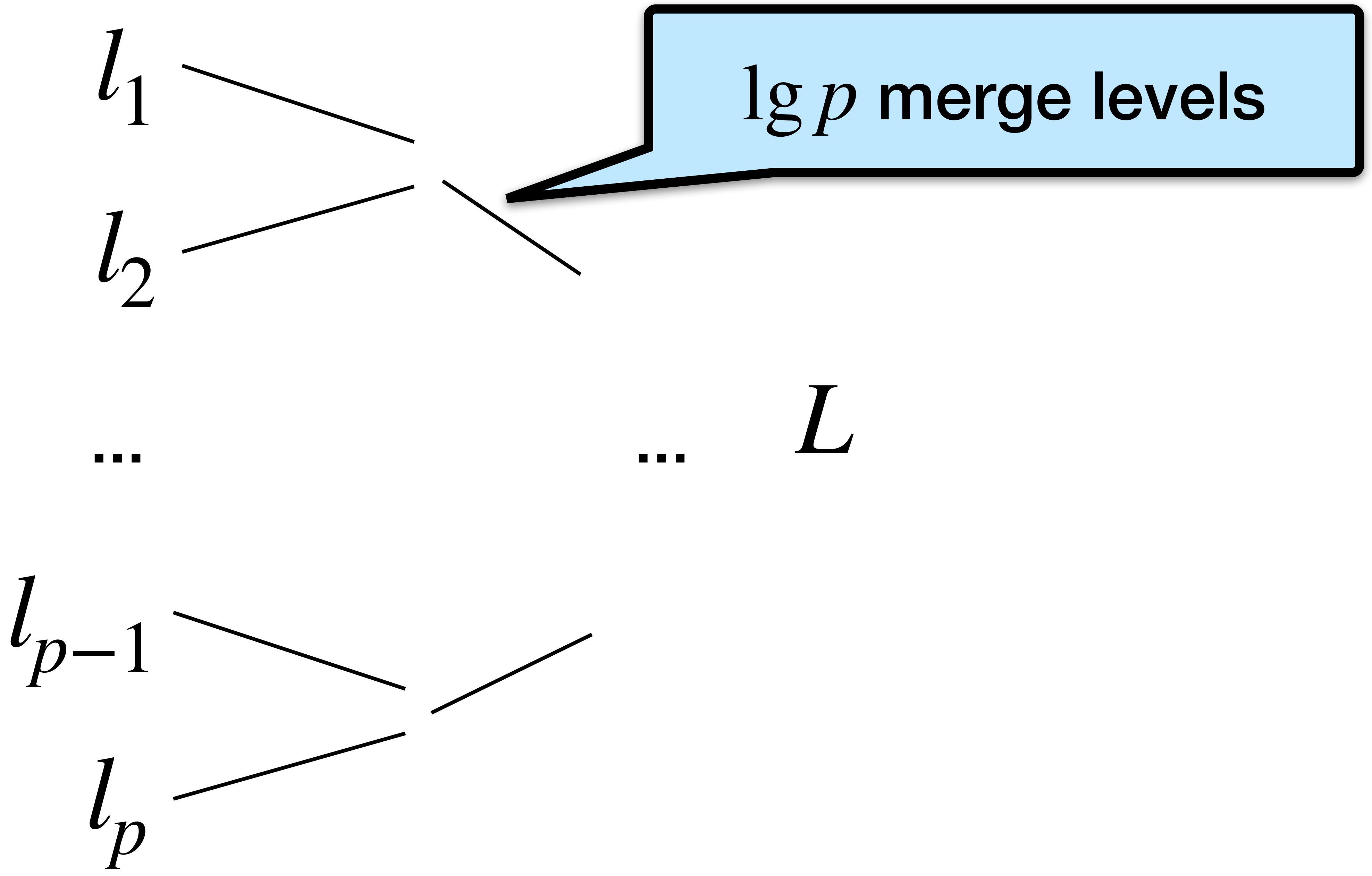
- Each processor should receive at most $m \le c\frac{n}{p}$ elements in Step 4.
  - where $c \ge 1$ is a small constant
- Finding splitters should not be too expensive (complexity)

# Tournament merge

Suppose we have lists $l_1, l_2, \ldots, l_p$ that we want to merge into one big list $L$.

The size of the sum of lists is bounded by $\sum_{i=1}^{p} |l_i| = O(n/p)$.

How can we do it in time $O((n/p)\lg p)$? (ignoring communication, assume the list are already located on one processor.)



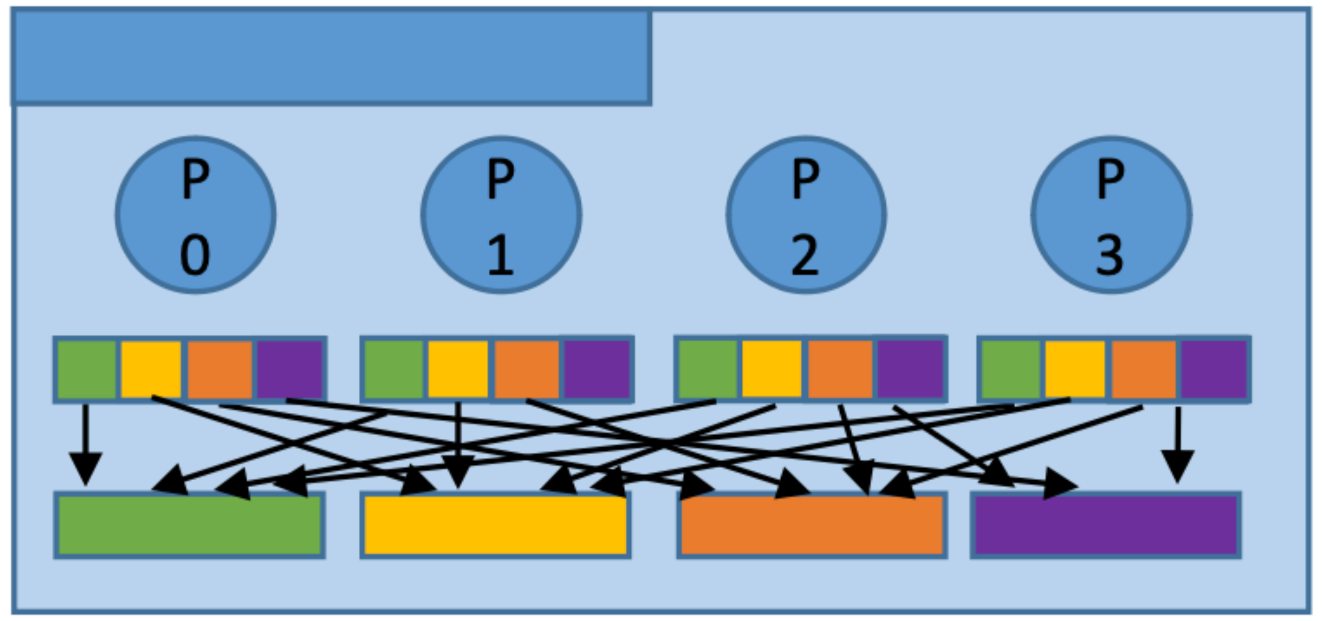
# Ideal sample sort

Lower bounds:
Computation = $\Omega((n \lg n)/p)$
Communication = $\Omega((\mu n)/p)$

**Sample sort**

1. Sort locally $O\left(\frac{n}{p}\log\frac{n}{p}\right)$
2. **Find $p-1$ "*good*" splitters**
3. Split local sequence into $p$ segments according to splitters
4. Many-to-many communication $O\left(\tau p + \mu\frac{n}{p}\right)$
5. Locally merge p sequences $O\left(\frac{n}{p}\log p\right)$

Ideally deterministic




How to efficiently merge p sequences?

**Good splitters:**

- Each processor should receive at most $m \le c\frac{n}{p}$ elements in Step 4.
  - where $c \ge 1$ is a small constant
- Finding splitters should not be too expensive (complexity)

How to find "good" splitters?

# Oversampling in sample sort

Idea: choose more than p, sort them, and choose "good" ones

## Finding "good" splitters by regular sampling

1. Pick $p-1$ equally spaced elements from the local sorted array on each processor (called local splitters from here onwards)
2. Sort all $p(p-1)$ local splitters using bitonic sort
3. Pick $p-1$ global splitters: From the sorted local splitter array, pick last local splitter on each processor (excl. last processor)
4. Allgather global splitters

$S_0, S_1, \ldots, S_{p-1}$ from **sorted**

$s_{0,1}, \ldots, s_{0,p-1}, s_{1,0}, \ldots, s_{1,p-1}, \ldots, s_{p,p-1}$

Initial element distribution:

| $p_0$ | | | | | | | | $p_1$ | | | | | | | | $p_2$ | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 22 | 7 | 13 | 18 | 2 | 17 | 1 | 14 | 20 | 6 | 10 | 21 | 15 | 9 | 24 | 3 | 16 | 19 | 23 | 4 | 11 | 12 | 5 | 8 |

Local sort and choose local splitters:

| $p_0$ | | | | | | | | $p_1$ | | | | | | | | $p_2$ | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | **7** | 13 | 14 | **17** | 18 | 22 | 3 | 6 | **9** | 10 | 15 | **20** | 21 | 24 | 4 | 5 | **8** | 11 | 12 | **16** | 19 | 23 |

Sample combining:

| | | | | | |
|---|---|---|---|---|---|
| 7 | 17 | 9 | 20 | 8 | 16 |

Sort samples and select global splitters:

| | | | | | |
|---|---|---|---|---|---|
| 7 | **8** | 9 | 16 | **17** | 20 |

Partition elements according to global splitters:

| $p_0$ | | | | | | | $p_1$ | | | | | | | | | $p_2$ | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |

# Proving even load distribution

Theorem: Each processor receives at most 2n/p elements.

***Proof:*** Number of local splitters contained in the elements received in each processor is $p-1$.

Let $s_i$ denote number of these local splitters that originally came from $P_i$ $\quad (\sum s_i = p-1)$.

Maximum number of elements that came from $P_i < (s_i+1)\frac{n}{p^2}$

Total number of elements received by a processor

$$< \sum_{i=0}^{p-1} (s_i+1)\frac{n}{p^2}$$

$$= \sum_{i=0}^{p-1} s_i \frac{n}{p^2} + \sum_{i=0}^{p-1} 1 \frac{n}{p^2}$$

$$= (p-1)\frac{n}{p^2} + p\frac{n}{p^2} < \frac{2n}{p}$$

# Sampling "good" splitters

**Finding "good" splitters by regular sampling**

1. Pick $p-1$ equally spaced elements (local splitters) on each processor
2. Sort all $p(p-1)$ local splitters using bitonic sort
3. Pick $p-1$ global splitters: last local splitter on each processor (excl. last)
4. Allgather global splitters

What is the complexity of selecting the global splitters?

**Bitonic sort for n > p:**

Computation: $O\left(\frac{n}{p}\log\frac{n}{p} + \frac{n}{p}\log^2 p\right)$

Communication: $O\left(\left(\tau + \mu\frac{n}{p}\right)\log^2 p\right)$

# Sampling "good" splitters

## Finding "good" splitters by regular sampling

1. Pick $p-1$ equally spaced elements (local splitters) on each processor
2. Sort all $p(p-1)$ local splitters using bitonic sort
3. Pick $p-1$ global splitters: last local splitter on each processor (excl. last)
4. Allgather global splitters

**Bitonic sort for n > p:**

Computation: $O\left(\frac{n}{p}\log\frac{n}{p} + \frac{n}{p}\log^2 p\right)$

Communication: $O\left(\left(\tau + \mu\frac{n}{p}\right)\log^2 p\right)$

***Complexity:***

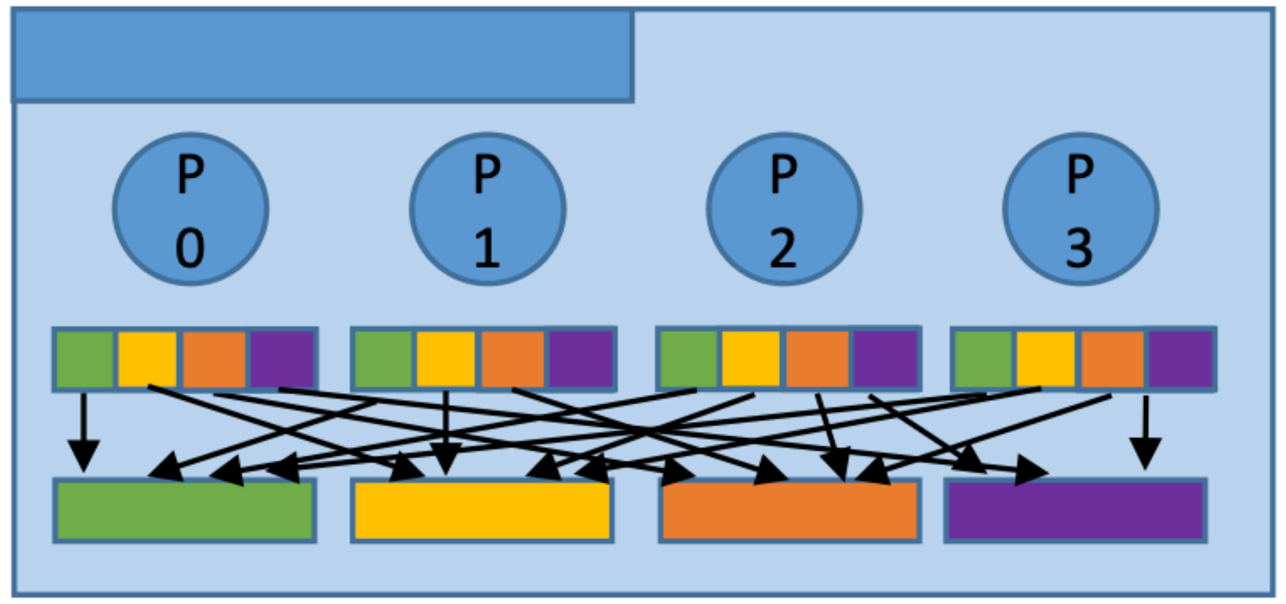
- Bitonic sort $n = O(p^2)$:
  - Computation: $O(p\log^2 \mathrm{p})$
  - Communication: $O((\tau + \mu p)\log^2 p)$
- Allgather: $O(\tau\log p + \mu p)$

# Sample sort complexity

**Lower bounds:**
Computation = $\Omega((n \lg n)/p)$
Communication = $\Omega((\mu n)/p)$

## Sample sort

1. Sort locally $O\left(\frac{n}{p}\log\frac{n}{p}\right)$
2. **Find $p-1$ "*good*" splitters**
3. Split local sequence into $p$ segments according to splitters
4. Many-to-many communication $O\left(\tau p + \mu\frac{n}{p}\right)$
5. Locally merge p sequences $O\left(\frac{n}{p}\log p\right)$




Computation: $O\left(\frac{n}{p}\log\frac{n}{p} + \frac{n}{p}\log p + p\log^2 p\right) = O\left(\frac{n\log n}{p} + p\log^2 p\right)$

Communication: $O\left(\tau p + \mu\frac{n}{p} + (\tau + \mu p)\log^2 p\right) = O\left(\tau p + \mu\left(\frac{n}{p} + p\log^2 p\right)\right)$

Optimal for $n > p^2\log^2 p$.