Parallel Algorithms

PART 2

Last time …

- Introduction to Parallel Algorithms
- Complexity analysis
- Work/Depth model
- Prefix Sum, Parallel Select

Questions?

Parallel Select

- Select numbers < pivot

$$A \leftarrow [1 \quad 2 \quad 3 \quad 0 \quad 4 \quad 0 \quad 2 \quad 3 \quad 0 \quad 1 \quad 3 \quad 4]$$

pivot ← 2

$$[1 \quad 0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0 \quad 1 \quad 1 \quad 0 \quad 0]$$

$$[1 \quad 1 \quad 1 \quad 2 \quad 2 \quad 3 \quad 3 \quad 3 \quad 4 \quad 5 \quad 5 \quad 5]$$

Parallel Select the $a_i < \text{pv}$

```
[l,m] ← select_lower (a, n, pv)
// t = t[0,…,n-1]
parfor (i=0; i<n; ++i)   t[i] ← a[i] < pv;
s ← scan (t);  m ← s(n-1);
parfor (i=0; i<n; ++i)   if t[i]  l[s[i] – 1] ← a[i];
```

$$W(n) = O(n)$$
$$D(n) = O(\log n)$$

Today …

Intro to Parallel Algorithms

- Parallel Search
- Parallel Sorting
  - Merge sort
  - Sample sort
  - Bitonic sort
- Communication costs

Parallel Search

- Problem Description
  - Given a sorted list X of size n and an element y
  - Find the index $i \mid x_i \leq y < x_{i+1}$
- Sequential
  - Use binary search
  - $O(\log n)$ time
- Work depth
  - `parfor(i) if` $x_i \leq y < x_{i+1}$ `return i; // no duplicates` $W = n, D = 1$
- PRAM
  - $O\left(\frac{\log n}{\log p}\right)$ using p processes

Ranking

- Given ordered lists, A, B of lengths s, t
- Define:

$$\text{rank}(z{:}\,A) \leftarrow \text{number of elements } \; a_i \mid a_i \leq z$$

- Define:

$$\text{rank}(B{:}\,A) \coloneqq (r_1, r_2, \ldots, r_t)$$
$$r_i \leftarrow \text{rank}(b_i{:}\,A)$$

Ranking

- $A = [7 \quad 13 \quad 25 \quad 26 \quad 31 \quad 54]$
- $B = [1 \quad 8 \quad 13 \quad 27]$
- $\text{rank}(B{:}\,A) = [0 \quad 1 \quad 2 \quad 4]$
- $\text{rank}(A{:}\,B) = [1 \quad 3 \quad 3 \quad 3 \quad 4 \quad 4 \quad 4]$

- Use binary search
- Consider a multithreaded vs Hadoop implementation

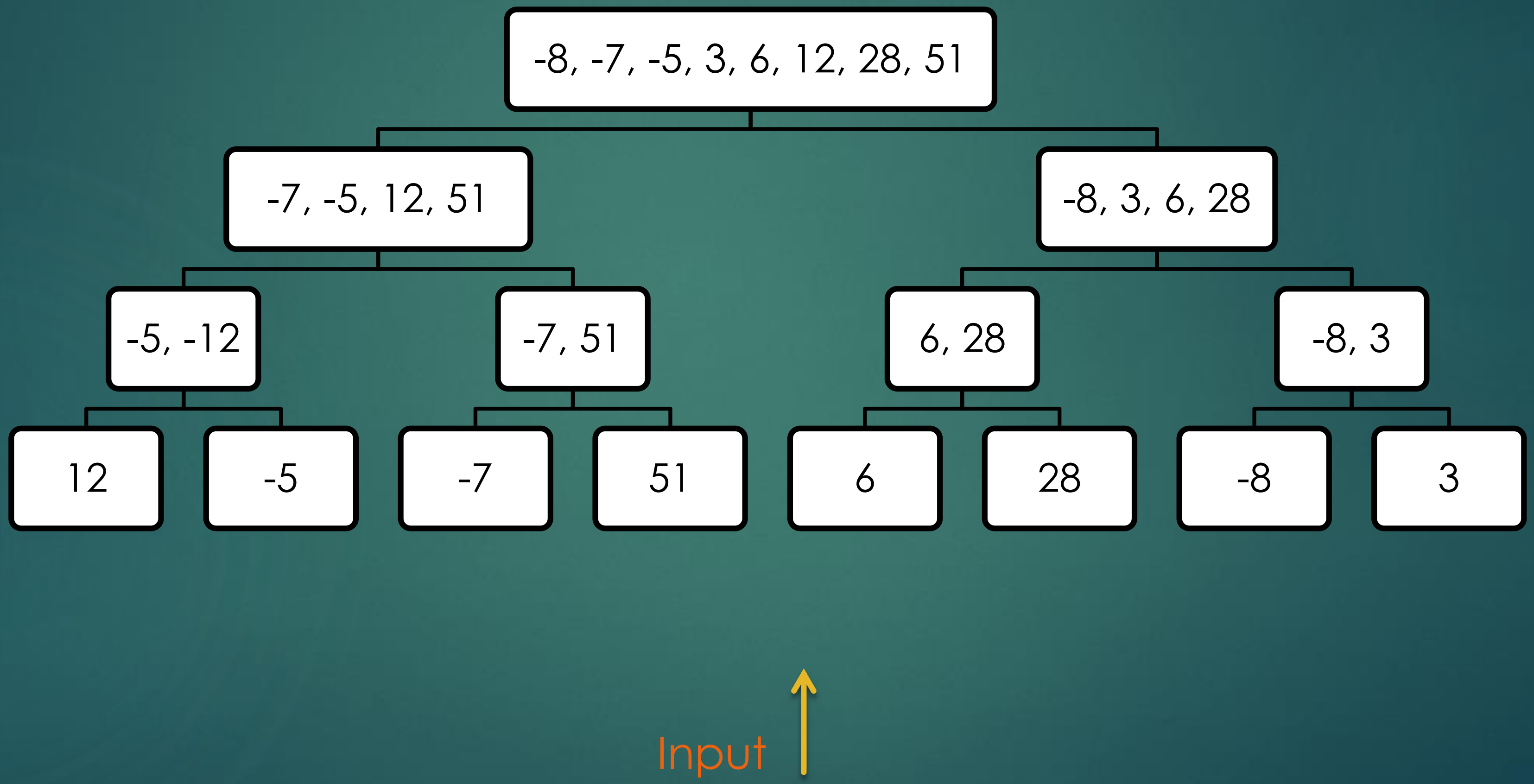
Parallel Sort

MERGE SORT

Divide & Conquer Merge Sort

- Divide X into X_1 and X_2
- Sort X_1 and X_2
- Merge X_1 and X_2
- Uses a Binary Tree
  - Bottom-up approach
  - Start with the leaves
  - Climb to the root
  - Merge the branches
- Requires parallel **Merge**

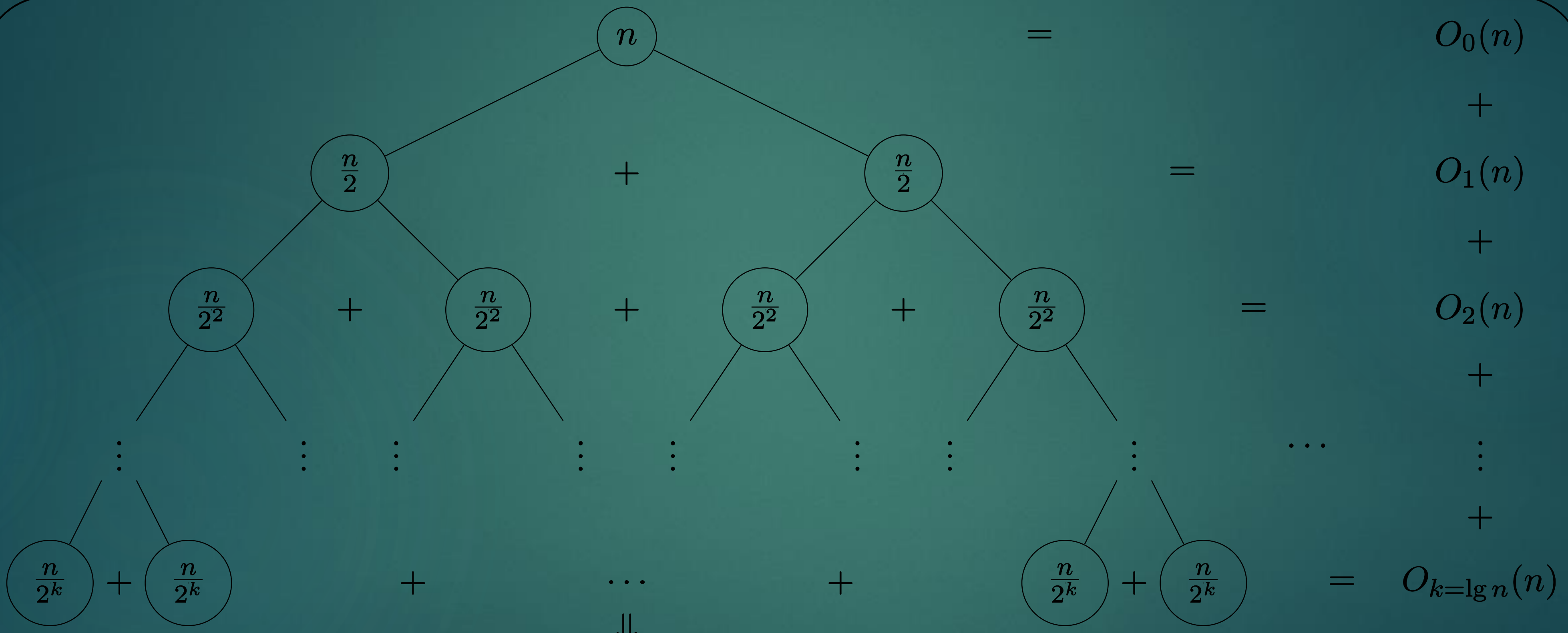
example



Merge sort

```
b = Merge_Sort(a,n)
    if n < 100
        return seqSort(a, n);
    b1 = Merge_Sort(a[0,…,n/2-1], n/2);
    b2 = Merge_Sort(a[n/2,…,n-1], n/2);
    return Merge (b1, b2);
```

Merge Sort - Complexity



$$O\left(\sum_{i=0}^{k} 2^i \cdot \frac{n}{2^i}\right) = O\left(\sum_{i=0}^{k} n\right) = O(k \cdot n) \Leftrightarrow O(n \cdot \lg n)$$

Parallel Merge

Merging two lists of lengths n, m

- Problem description ($m \leq n$)
  - Given, $A = (a_1, a_2, \ldots, a_n)$ and $B = (b_1, b_2, \ldots, b_m)$
  - $a_i < a_{i+1} \quad \forall i$
  - $b_i < b_{i+1} \quad \forall i$
  - $A \cap B = \emptyset$
  - Build $C = (c_1, c_2, \ldots, c_{n+m})$
  - $c_i \in A \cup B$
  - $c_i < c_{i+1} \quad \forall i$

Merging two sorted lists

- Best Sequential Time: $O(n)$
- Parallel Merge:
  - Tradeoffs between
    - Depth-Optimal
    - Work-Optimal

Merging using Ranking

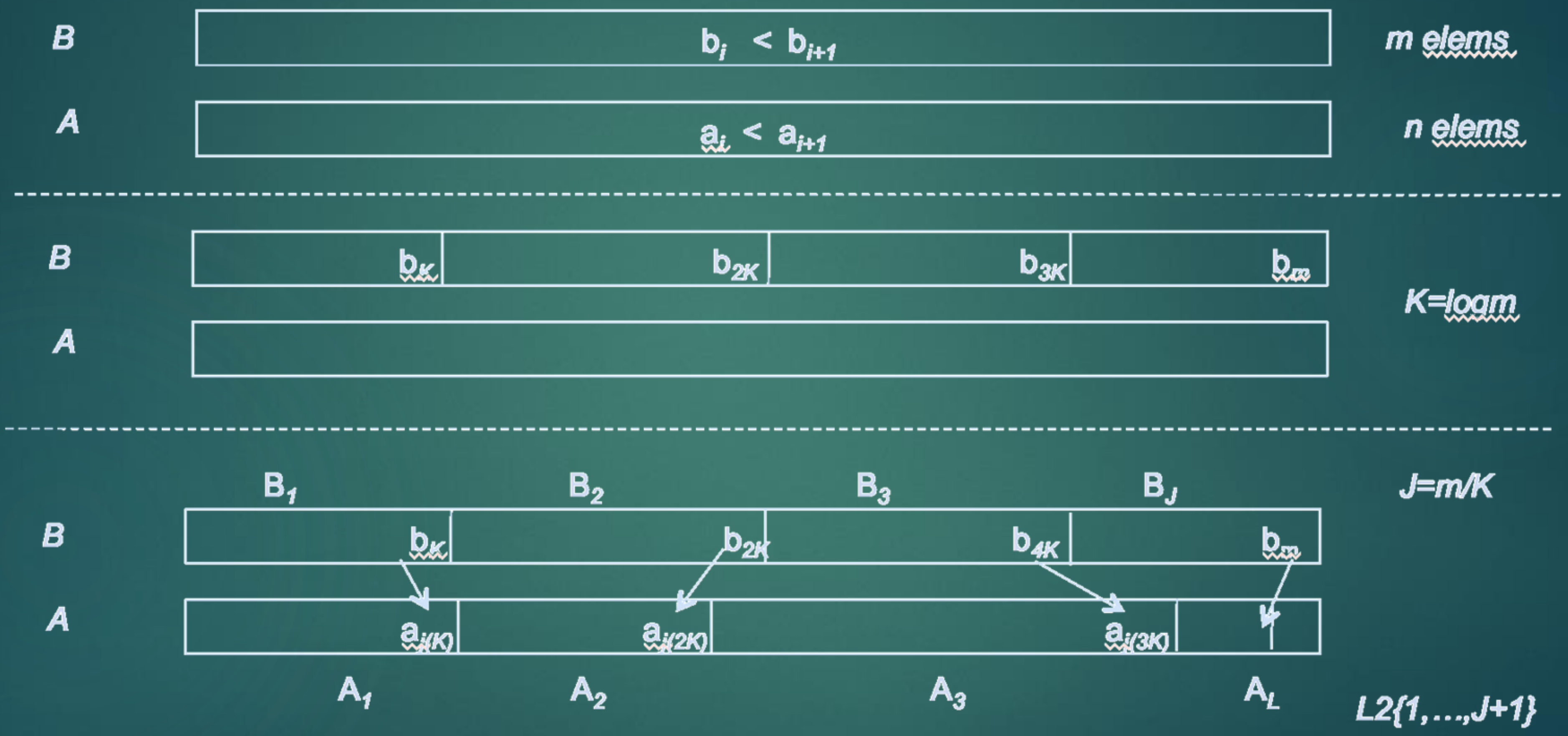
- Assume elements in A and B are distinct
- Let C be the merged result. Given,
  - $x \in C$
  - $\text{rank}(x : C) = i$
  - $c_i = x$
- Property

$$\text{rank}(x : C) = \text{rank}(x : A) + \text{rank}(x : B)$$

- Solution to the merging problem,
  - Find $\text{rank}(A : B)$ and $\text{rank}(B : A)$
  - Parallel searches using $p = nm$, $D = O(1)$ but $W = O(n^2)$
  - Concurrent binary searches, $D = O(\log n)$ and $W = O(n \log n)$
  - Goal: Parallelize with optimal work

Recall that an algorithm is work optimal iff $W_p = W_{seq}$

Example



B
$b_i < b_{i+1}$
m elems
A
$a_i < a_{i+1}$
n elems
B
b_K
b_{2K}
b_{3K}
b_m
K=logm
A
B_1
B_2
B_3
B_J
J=m/K
B
b_K
b_{2K}
b_{4K}
b_m
A
$a_{i(K)}$
$a_{i(2K)}$
$a_{i(3K)}$
A_1
A_2
A_3
A_L
L2{1,…,J+1}

Work-optimal merge - `Merge1`

- $A = [a_1, \dots, a_n], \quad B = [b_1, \dots, b_m], \quad n \geq m$

1. Partition B into $\frac{m}{\log m}$ blocks
   - Size of each block → $\log m$
2. `parallel for i =` 1 : $m / \log m$
   - $R_i \leftarrow \mathrm{rank}(b_{i \log m} : A)$ using sequential binary search
3. Partition A accordingly
   - Block $A_i : (a_{R_{i-1}+1}, \dots, a_{R_i})$
4. Merge blocks A_i and B_i in $O(\log m)$ time using sequential merge
   - But if $|A_i| \gg |B_i| = \log m$, then recurse … `Merge1`(B_i, A_i)

Work ?
Depth ?

Sequential Sorting

- What is the complexity ?

$$\mathcal{O}(?)$$

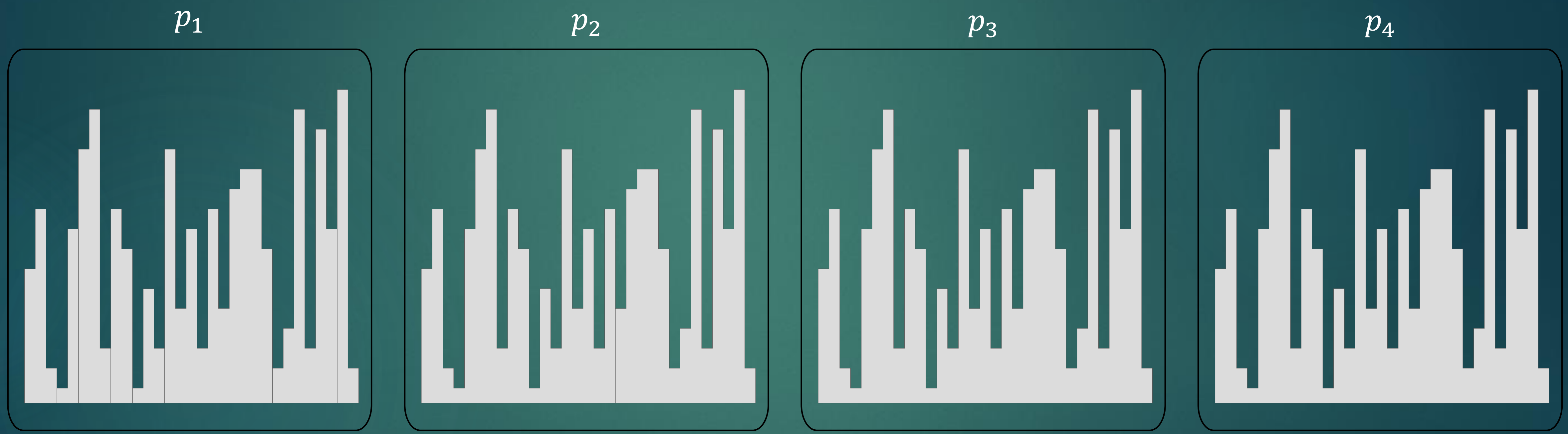
Sequential Sorting

- Comparison based
  - $\mathcal{O}(n \log n)$
  - Can we sort faster than $\mathcal{O}(n \log n)$?
- Non-comparison based
  - $\mathcal{O}(n)$

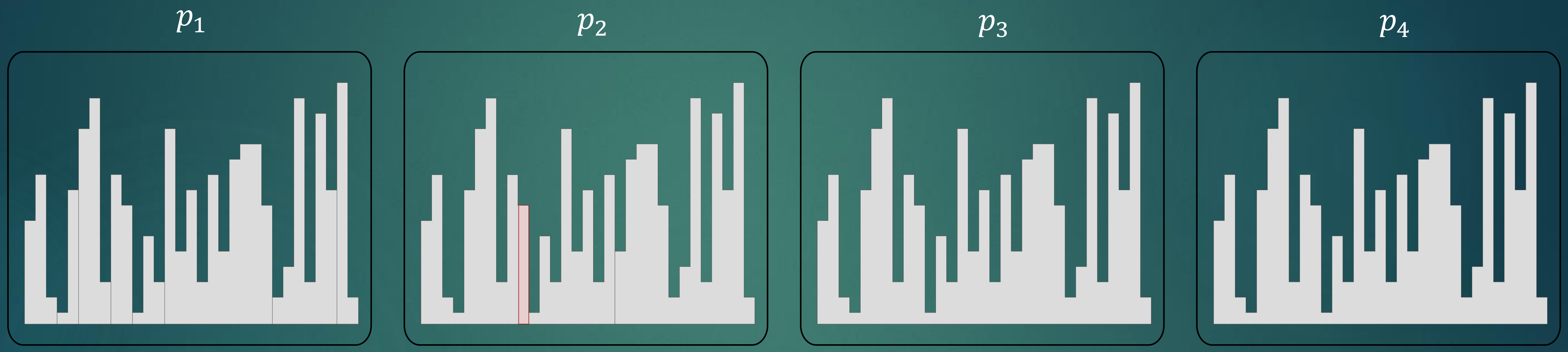
Bucket sort

- Assume input is uniformly distributed over an interval $[a, b]$
- Divide interval into m equal sized intervals (**buckets**)
- Drop numbers into appropriate buckets
- Sort each bucket (say using quicksort)
  - $\mathcal{O}\left(n \log\left(\frac{n}{m}\right)\right)$
  - For $m = \mathcal{O}(n)$ → $\mathcal{O}(n)$ sorting
- Radix sort
- dense, uniform distribution

Parallel Quicksort

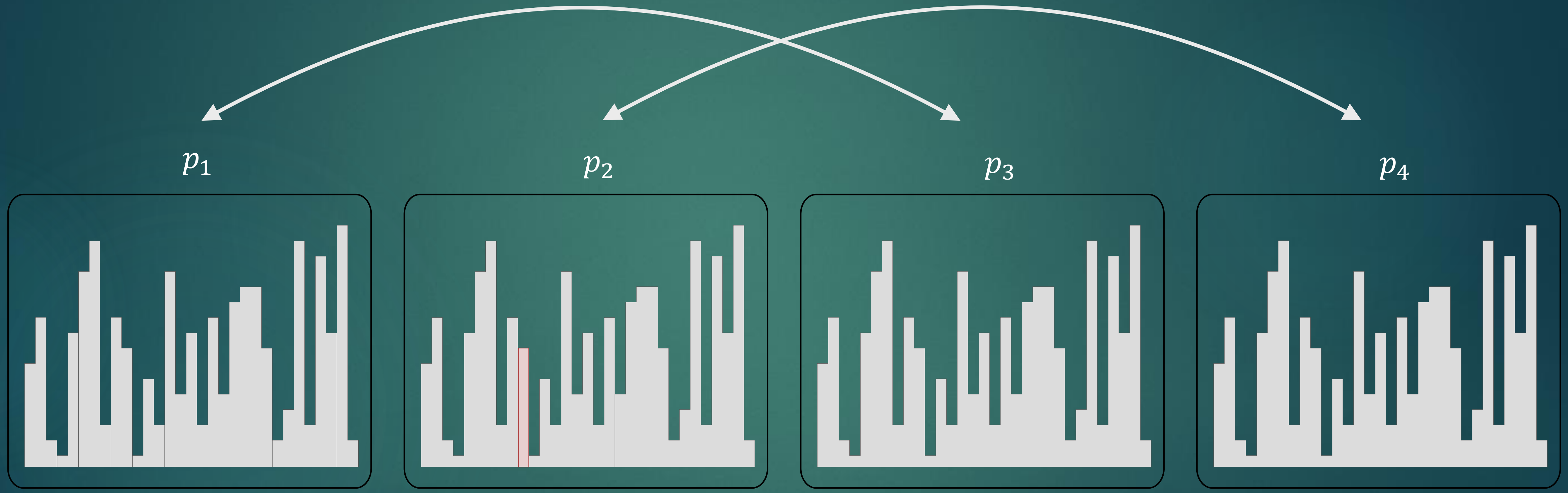


Parallel Quicksort



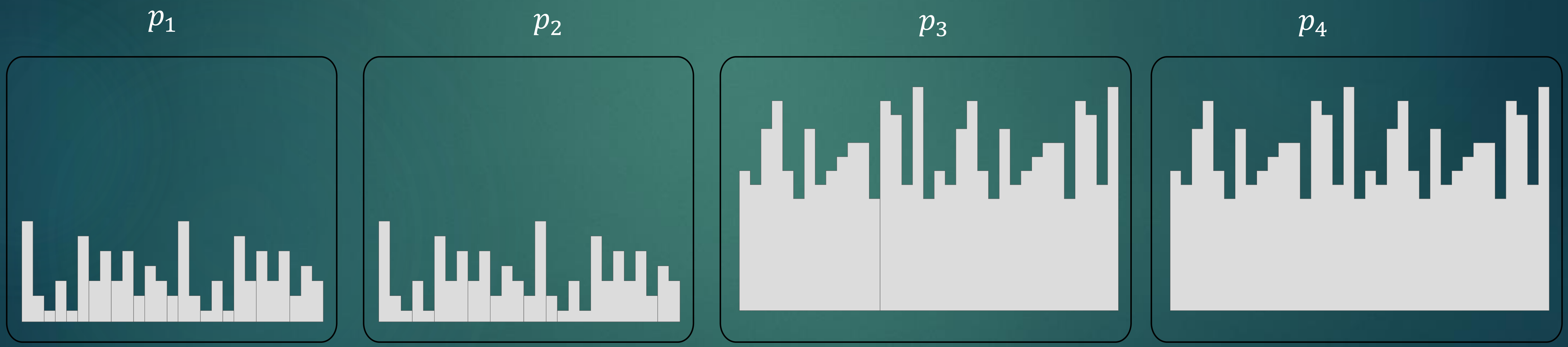
parallel median selection

Parallel Quicksort

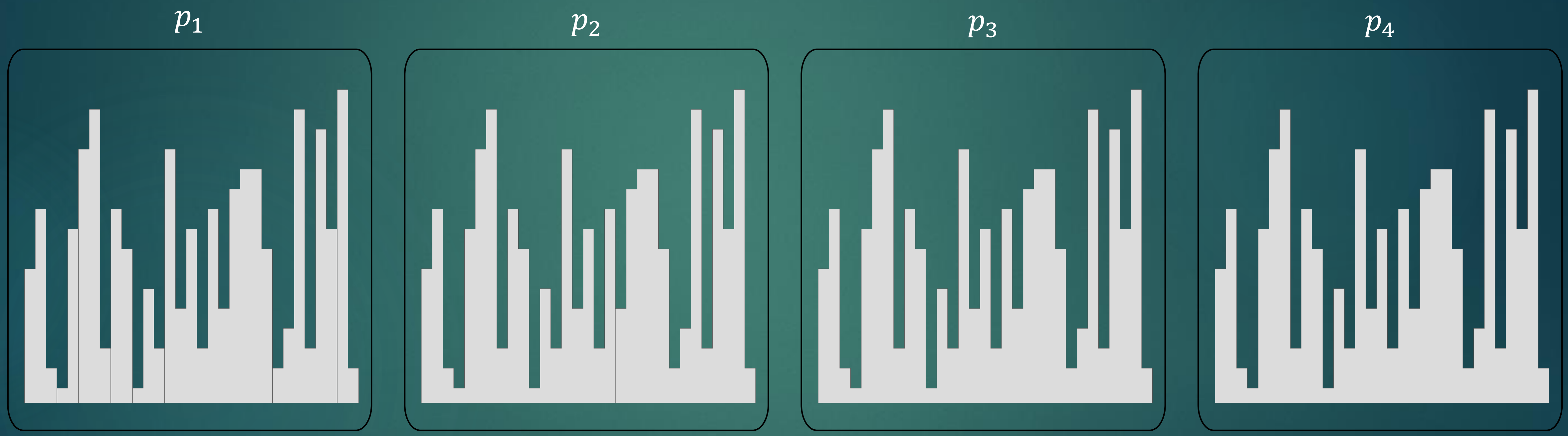


parallel exchange

Parallel Quicksort

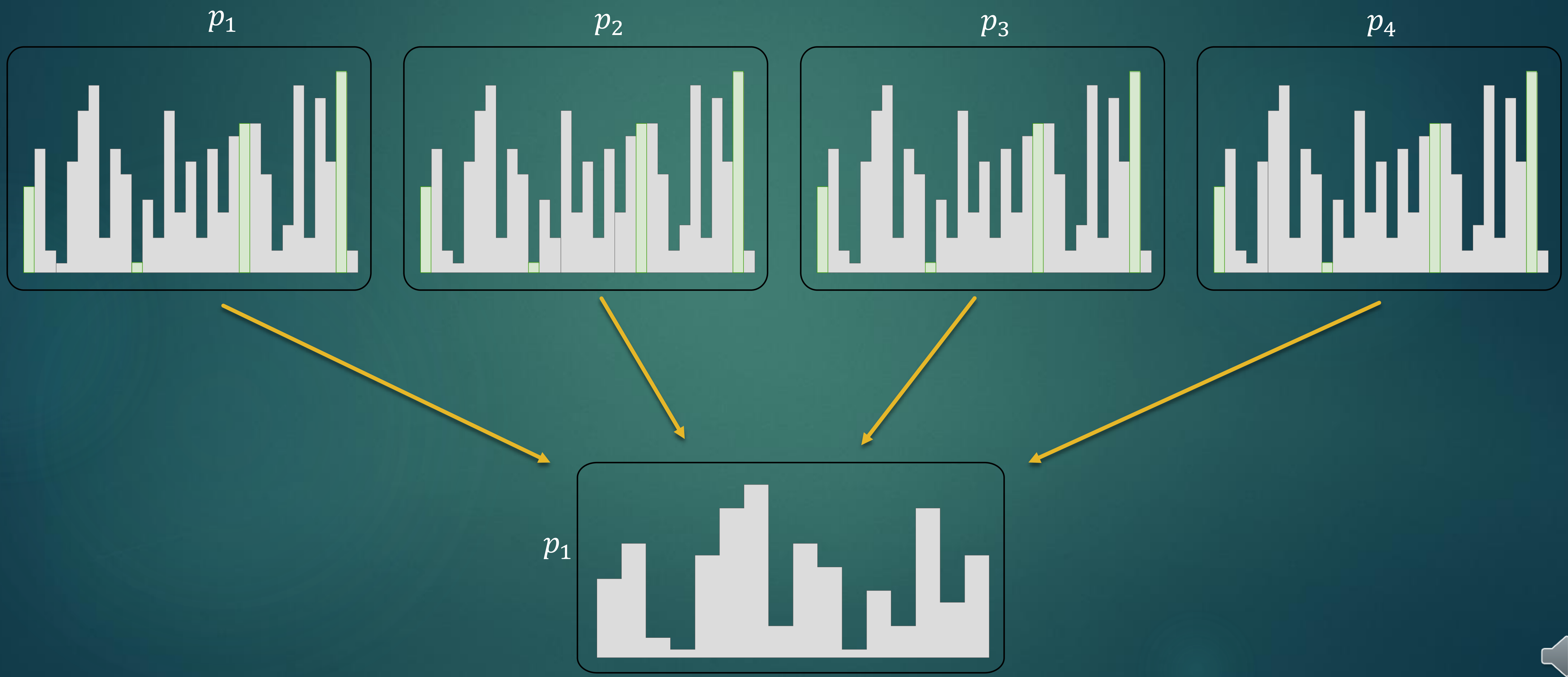


Sample Sort

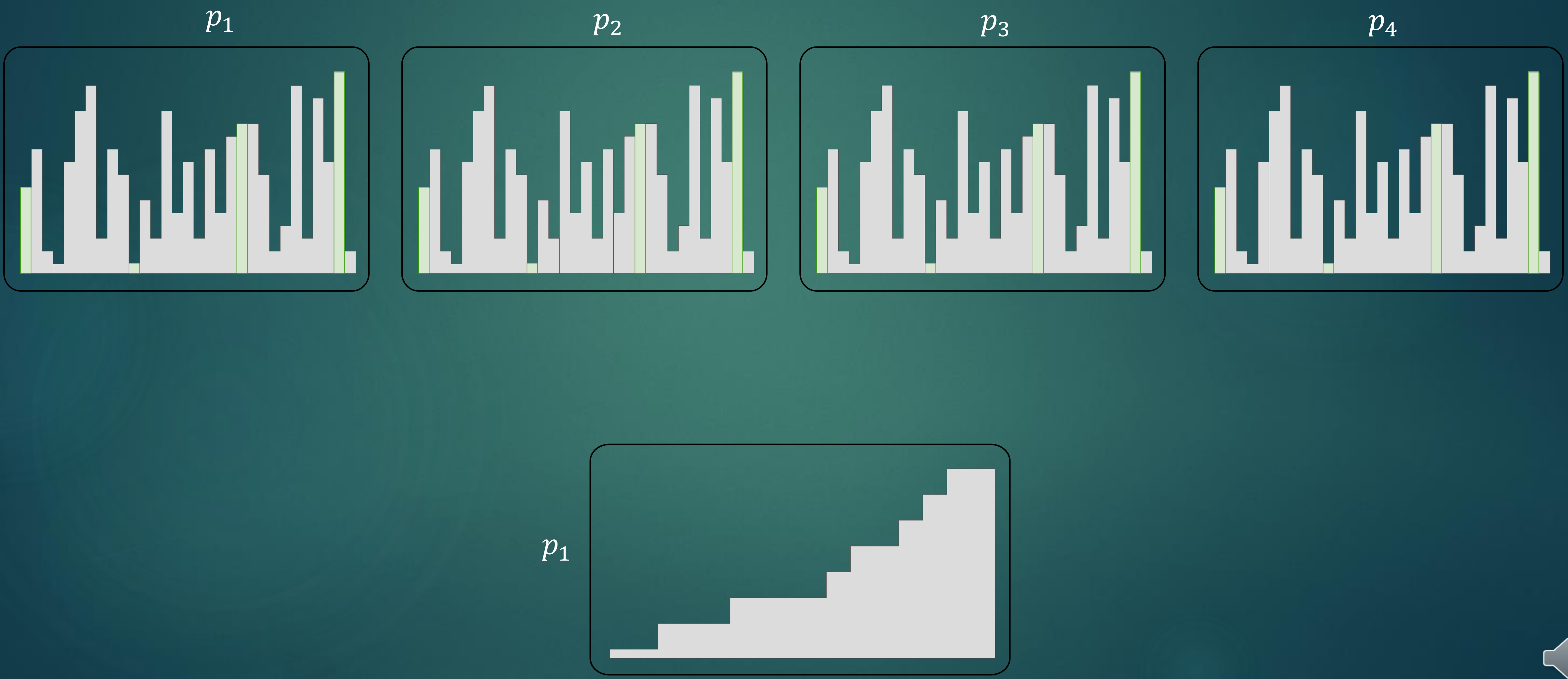


Sample Sort

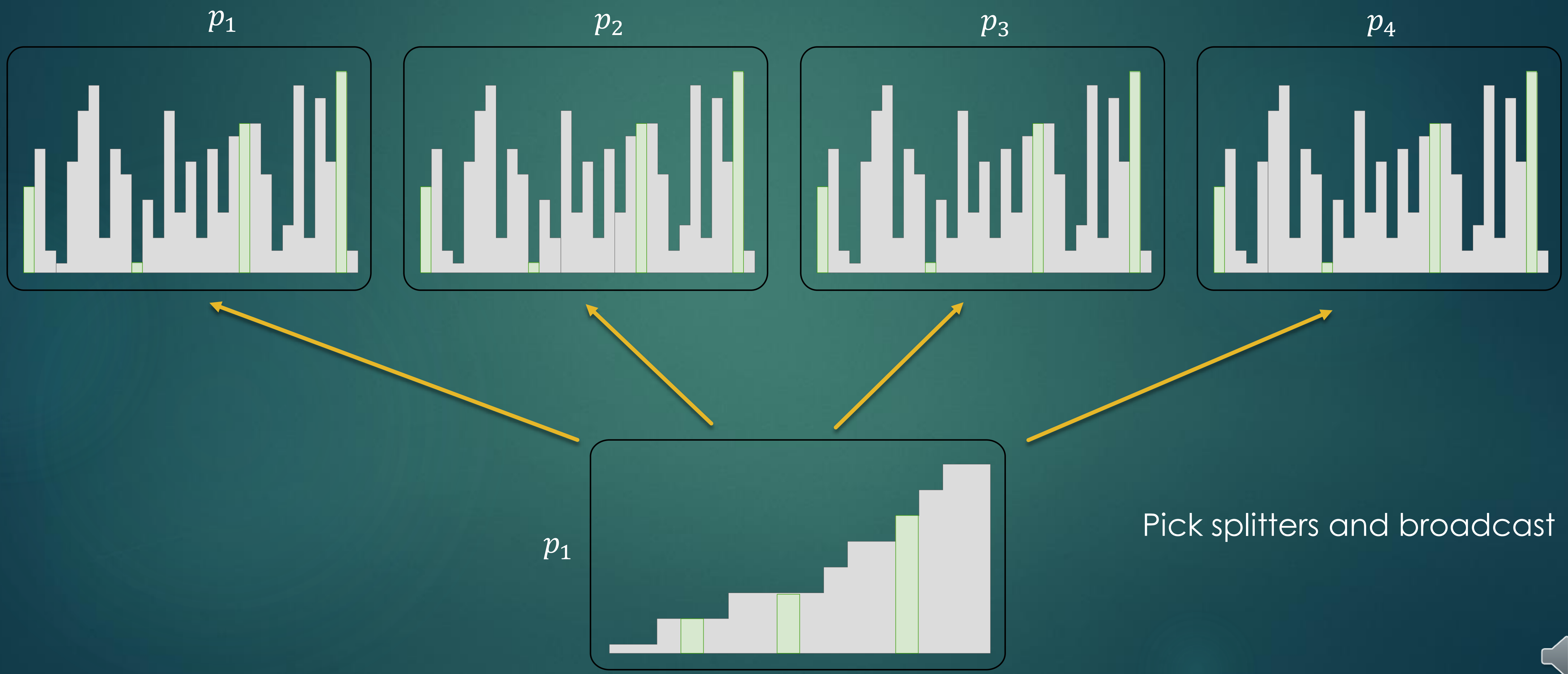
Sample Sort



Sample Sort

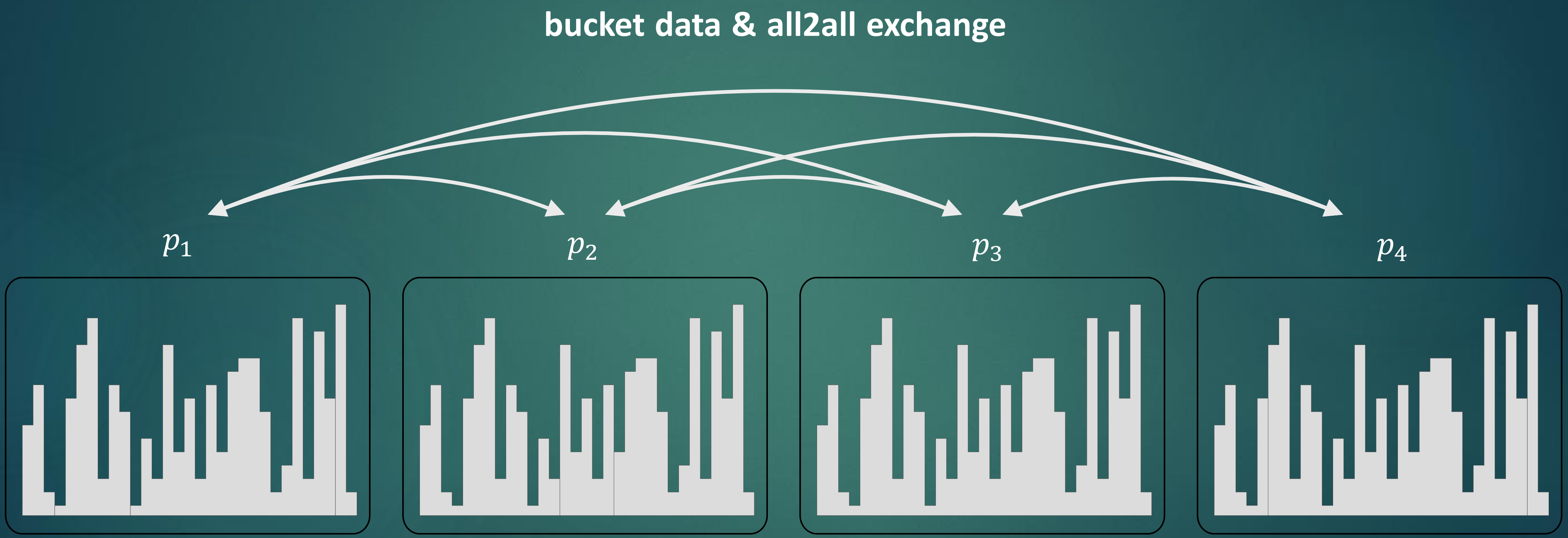


Sample Sort



Pick splitters and broadcast

Sample Sort



Sample sort

- randomly partition input in $\frac{n}{p}$ points
- sort locally
- select p splitters/processor (evenly)
  - guarantees no more 2*n/p elements / bucket (**proof?**)
- gather(splitters) in p_0
- sort splitters in p_0 and create buckets
  - block partition using p binary search on n/p sorted seq.
- Exchange data
- sort

Sample Sort

Sort locally	$\mathcal{O}\left(\frac{n}{p}\log\frac{n}{p}\right)$
Select $p-1$ splitters per process	$\mathcal{O}(p)$
Gather splitters in p_0	$\mathcal{O}(p^2)$
Sort splitters in p_0	$\mathcal{O}(p^2\log p)$
Broadcast splitters	$\mathcal{O}(p\log p)$
Sort again	$\mathcal{O}\left(\frac{n}{p}\log\frac{n}{p}\right)$

Sample Sort – load balance

- Guarantees no more $\frac{2n}{p}$ elements / bucket
- Proof:

All entries on p_i must be $> s_{i-1}$ and $\leq s_i$

$(i-2)p + \frac{p}{2}$ elements of the sample $\leq s_i$

→ lower bound elements = $\frac{\left((i-2)p+\frac{p}{2}\right)n}{p^2}$

$(p-i)p - \frac{p}{2}$ elements of the sample $> s_i$

→ upper bound elements = $\frac{\left((p-i)p-\frac{p}{2}\right)n}{p^2} + \frac{n}{p^2} - 1$

Maximum number of elements on processor i,

$$n - ub - lb = \frac{2n}{p} - \frac{n}{p^2} + 1 \leq \frac{2n}{p} \qquad \blacksquare$$