PARALLEL QUICKSORT

CSE 633: PARALLEL ALGORITHMS

GUIDED BY DR. RUSS MILLER

SRI ABINAYA - 50292993

AGENDA

- QUICKSORT
- SEQUENTIAL QUICKSORT
- IMPLEMENTATION OF SEQUENTIAL QUICKSORT
- PARALLEL QUICKSORT
- IMPLEMENTATION OF PARALLEL QUICKSORT
- CORRECTION
- CHALLENGES
- RESULTS

THE PROBLEM-QUICKSORT

To sort a list of numbers in either increasing or decreasing order.

QuickSort is a Divide and Conquer algorithm.

On the average, it has O(n log n) complexity, making quicksort suitable for sorting big data volumes. So, it is important to make it parallel.

SEQUENTIAL QUICKSORT ALGORITHM

- Select median as pivot from the sample data set picked from the actual data set.
- Divide the list into two sub lists: a “low list” containing numbers smaller than the pivot, and a “high list” containing numbers larger than the pivot
- The low list and high list recursively repeat the procedure to sort themselves
- The final sorted result is the concatenation of the sorted low list, the pivot, and the sorted high list.

0	1	2	3	4	5	6
10	80	30	90	40	50	70

0	1	2	3	4	5	6
10	80	30	90	40	50	70

Pivot

0	1	2	3	4	5	6
10	80	30	90	40	50	70

i

j

Pivot

0	1	2	3	4	5	6
10	80	30	90	40	50	70

i j

Pivot

J IS A LOOP VARIABLE

0	1	2	3	4	5	6
10	80	30	90	40	50	70

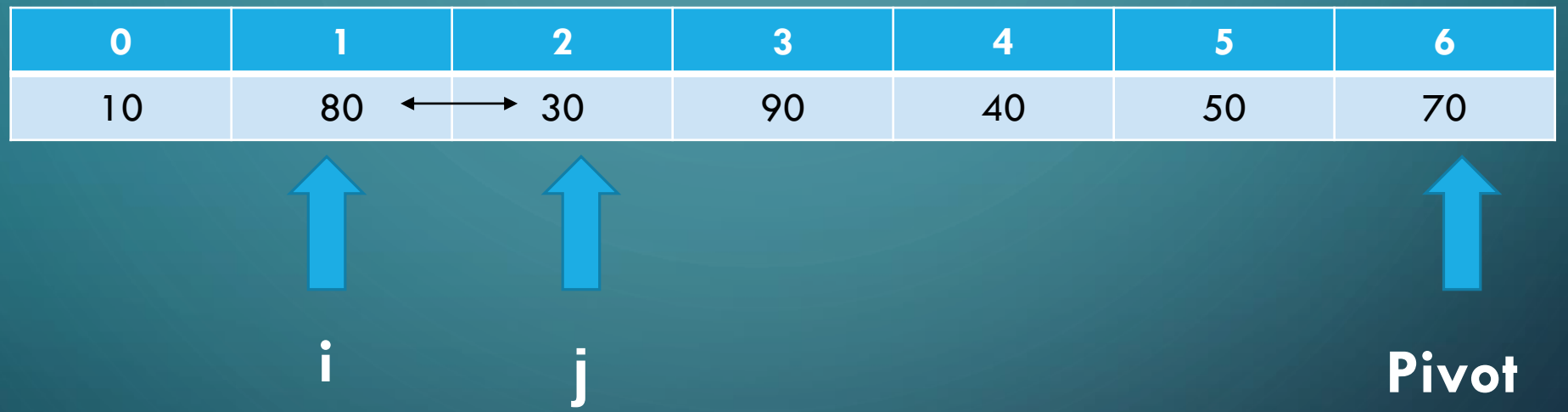
i j Pivot

J IS A LOOP VARIABLE

0	1	2	3	4	5	6
10	80	30	90	40	50	70

i j Pivot

J IS A LOOP VARIABLE

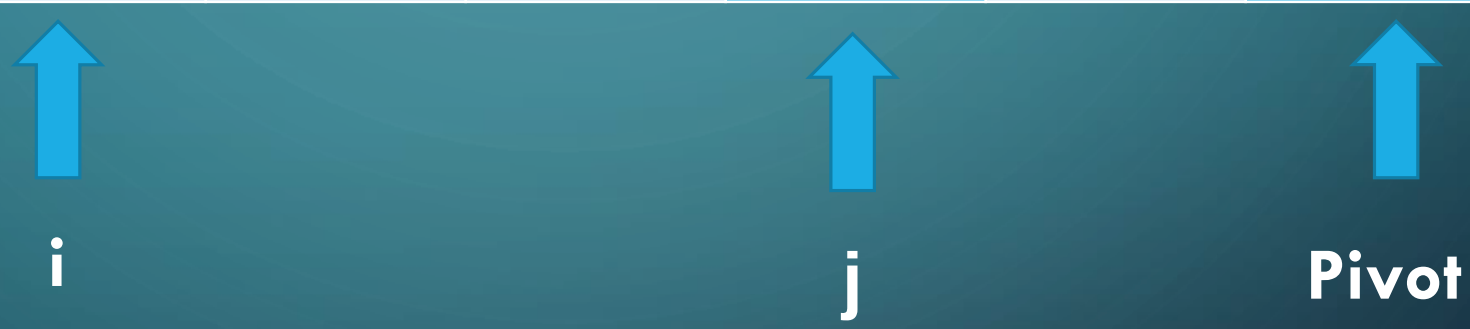


J IS A LOOP VARIABLE

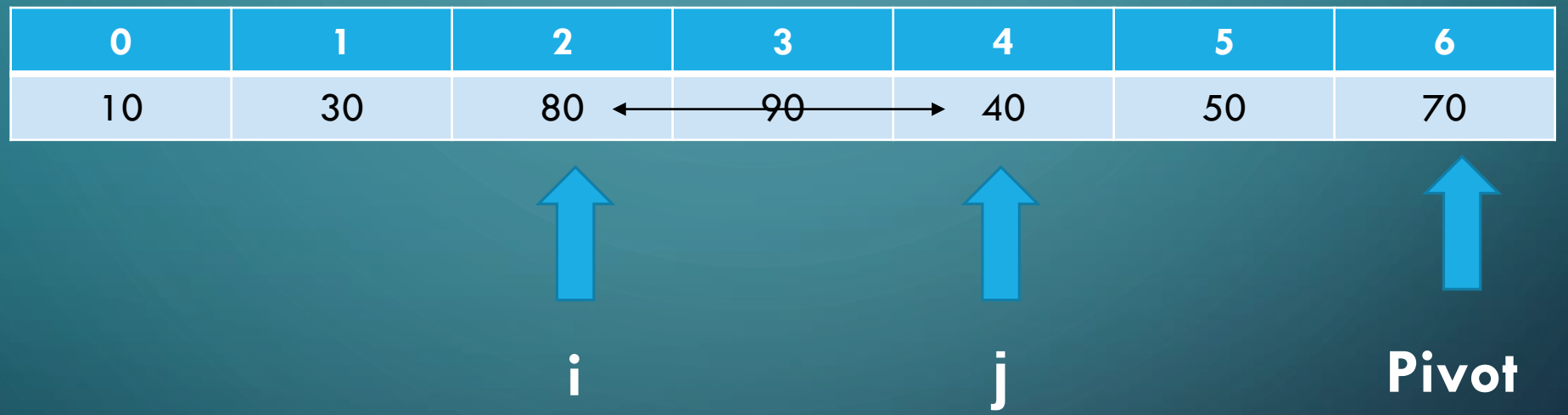
0	1	2	3	4	5	6
10	30	80	90	40	50	70

J IS A LOOP VARIABLE

0	1	2	3	4	5	6
10	30	80	90	40	50	70

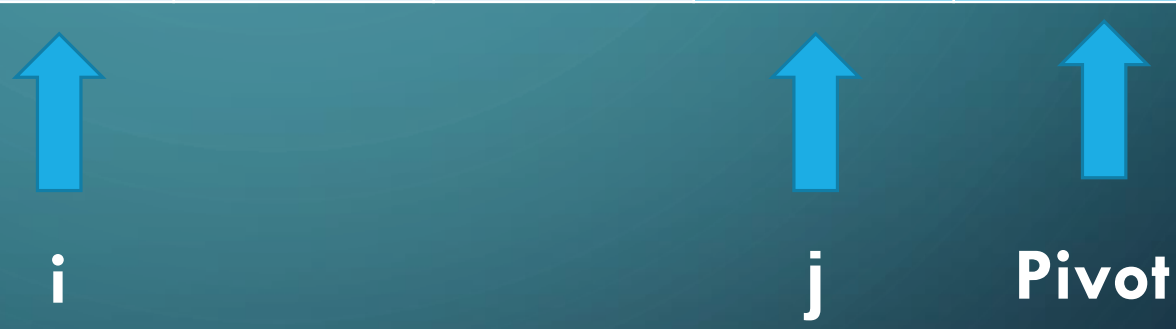


J IS A LOOP VARIABLE

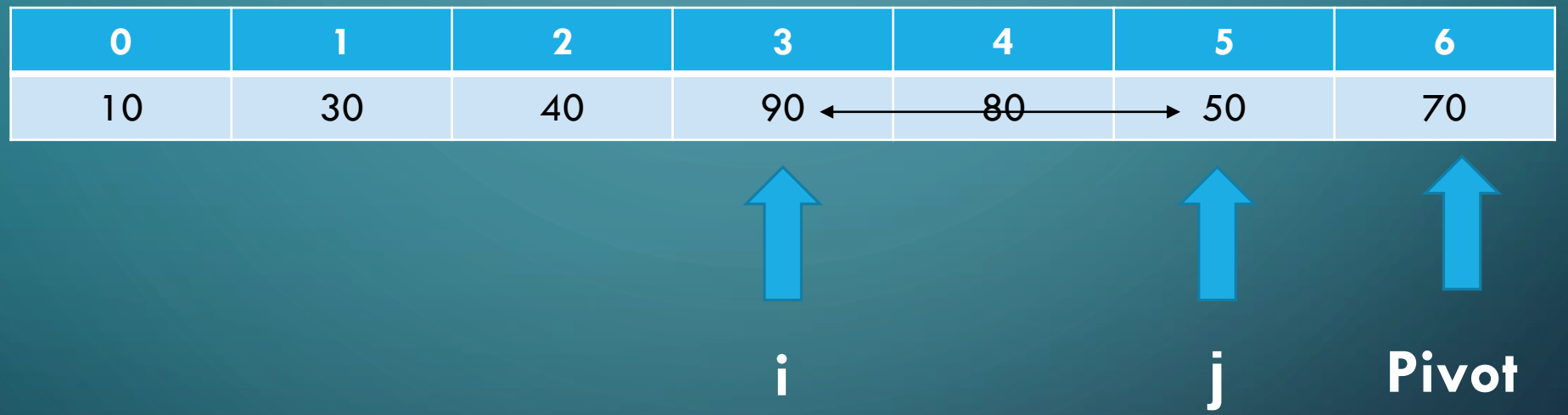


J IS A LOOP VARIABLE

0	1	2	3	4	5	6
10	30	40	90	80	50	70

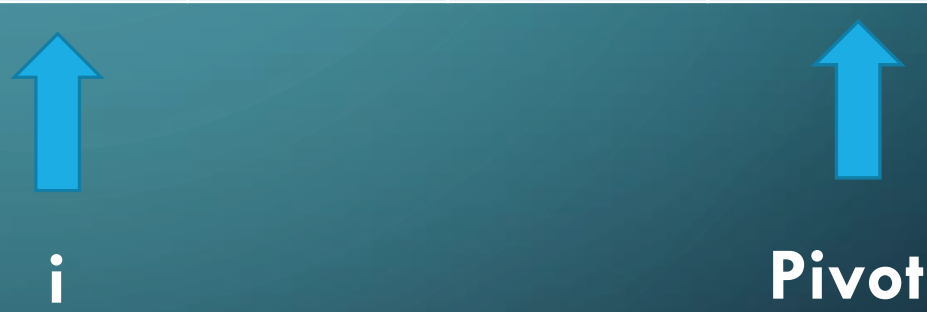


J IS A LOOP VARIABLE



J IS A LOOP VARIABLE

0	1	2	3	4	5	6
10	30	40	50	80	90	70



J IS A LOOP VARIABLE

0	1	2	3
10	30	40	50

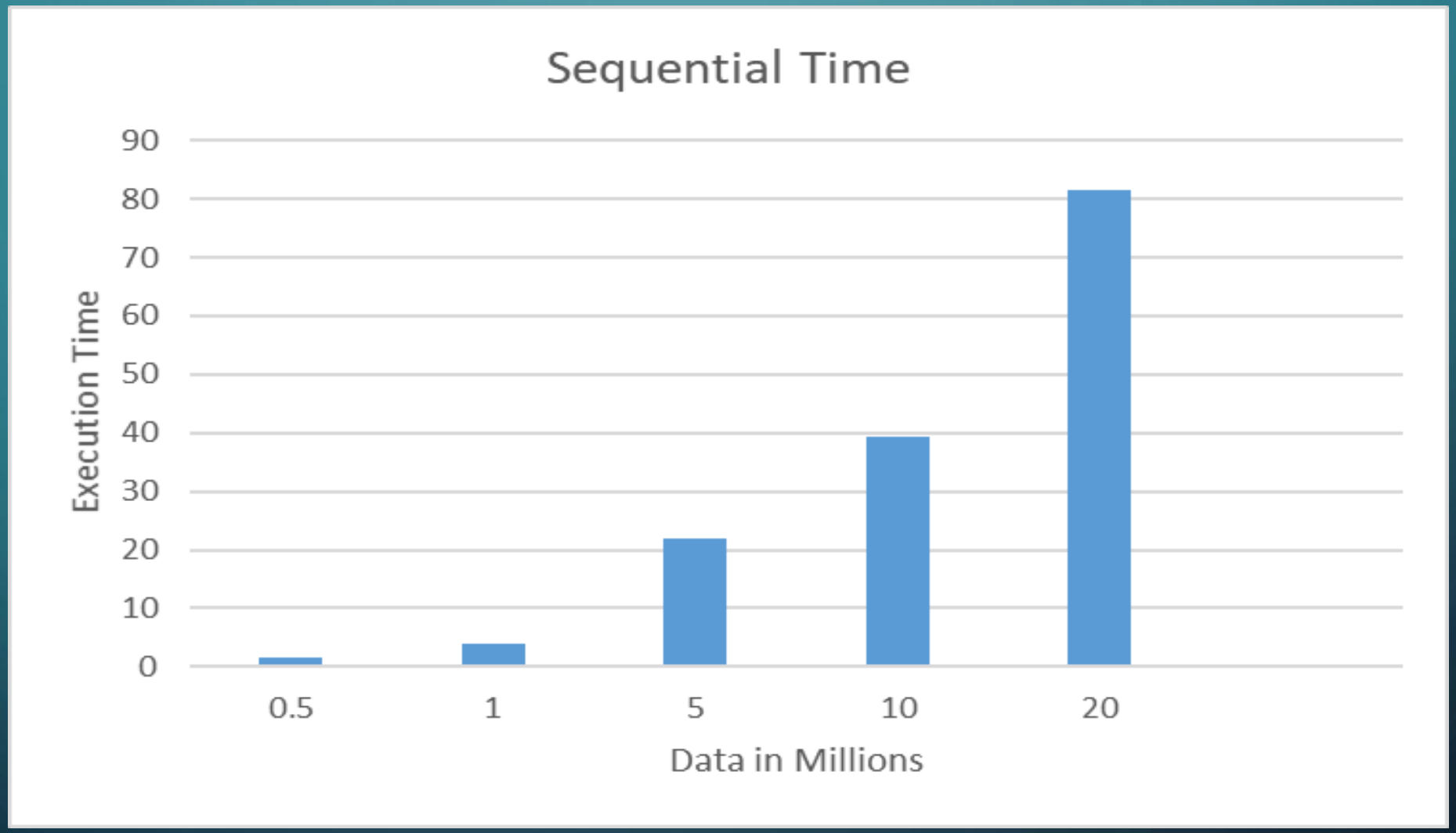
Pivot

4	5	6
70	90	80

Pivot

J IS A LOOP VARIABLE

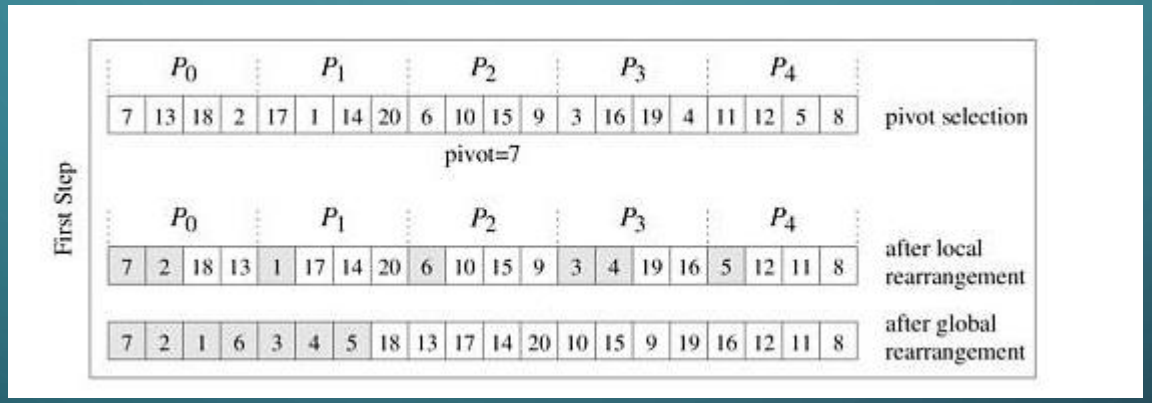
0	1	2	3	4	5	6
10	30	40	50	70	80	90


Sequential Time
Execution Time
90
80
70
60
50
40
30
20
10
0
0.5
1
5
10
20
Data in Millions

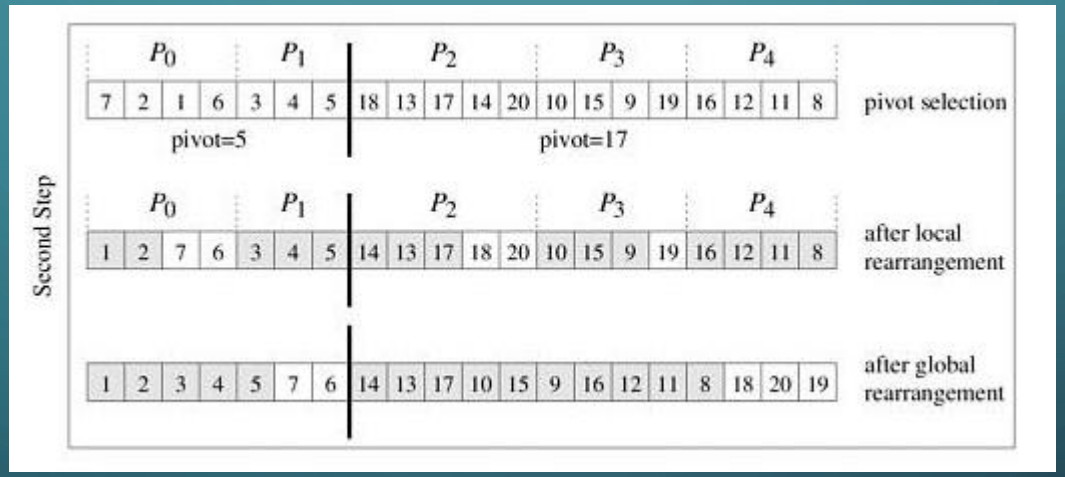
PARALLEL QUICK SORT ALGORITHM

- We choose a pivot which nearer to median by taking samples from one of the processes and broadcast it to every process.
- Each process divides its unsorted list into two lists: those smaller than (or equal) the pivot, those greater than the pivot Each process in the upper half of the process list sends its “low list” to a partner process in the lower half of the process list and receives a “high list” in return
- Now, the upper-half processes have only values greater than the pivot, and the lower-half processes have only values smaller than the pivot.
- Thereafter, the processes divide themselves into two groups and the algorithm recurses.
- After log P recursions, every process has an unsorted list of values completely disjoint from the values held by the other processes.
- The largest value on process i will be smaller than the smallest value held by process i + 1. Each process finally sorts its list using sequential quicksort.

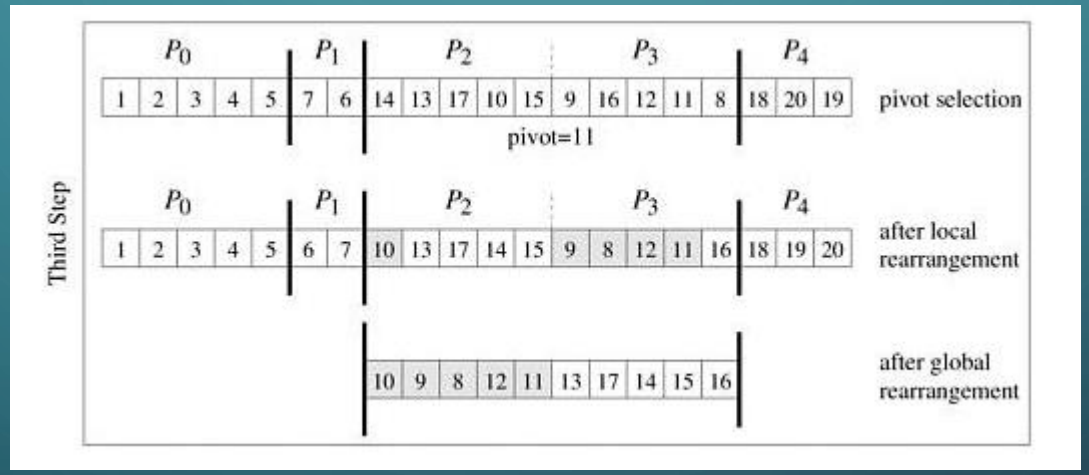
PARALLEL QUICKSORT



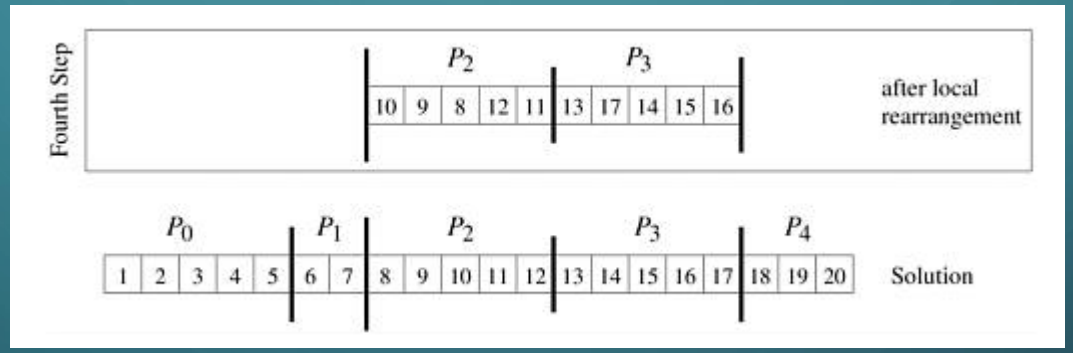
PARALLEL QUICKSORT



PARALLEL QUICKSORT

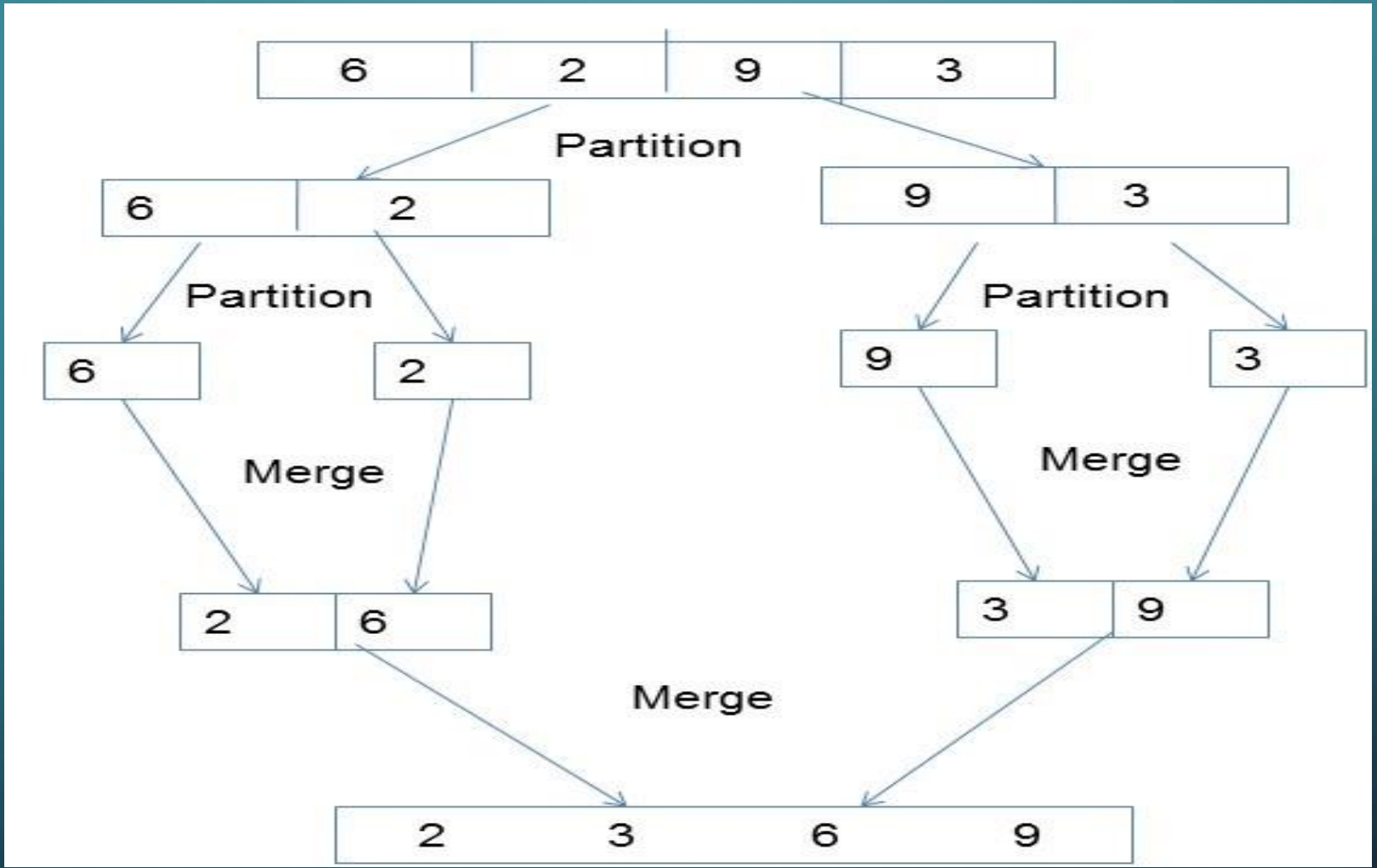


PARALLEL QUICKSORT

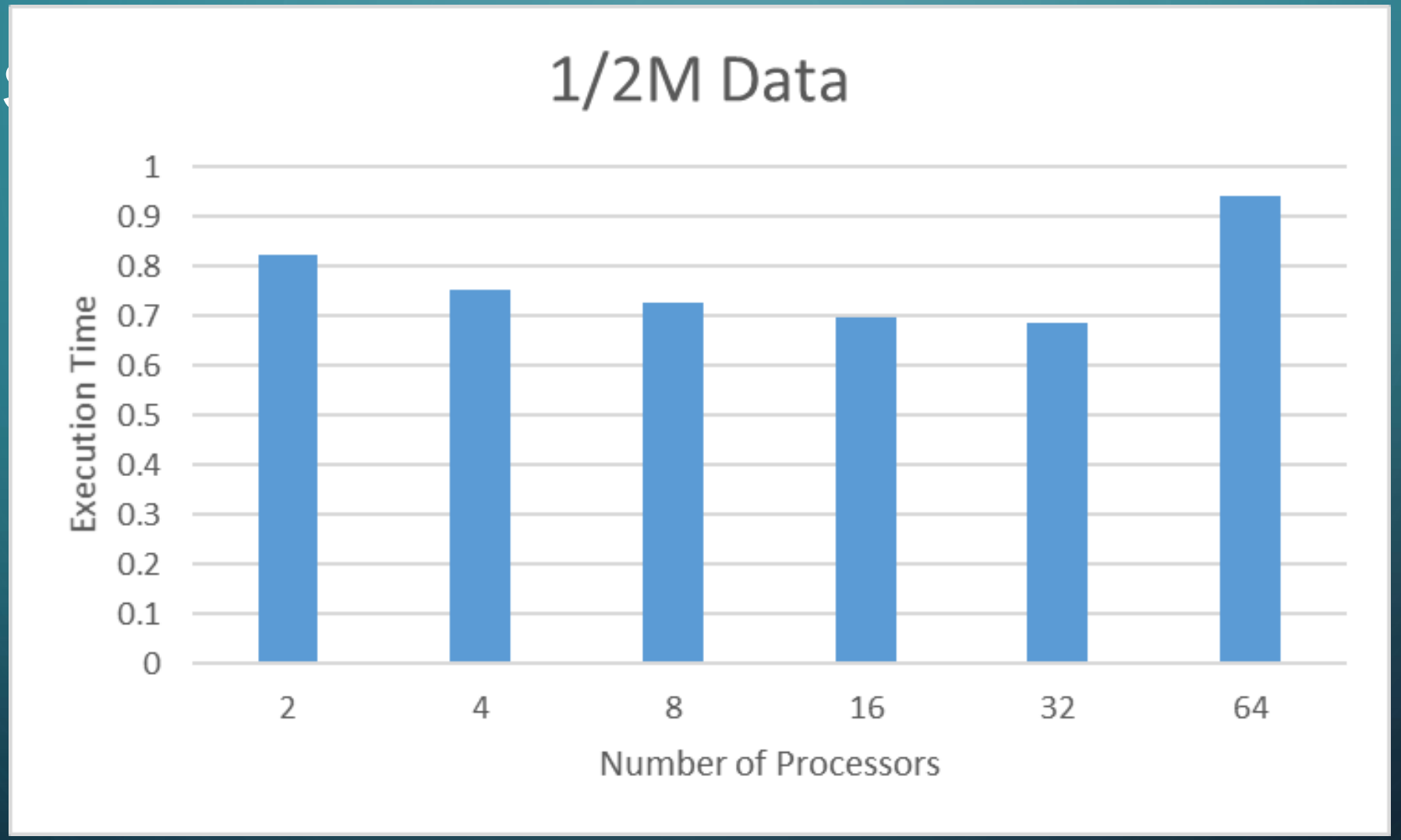


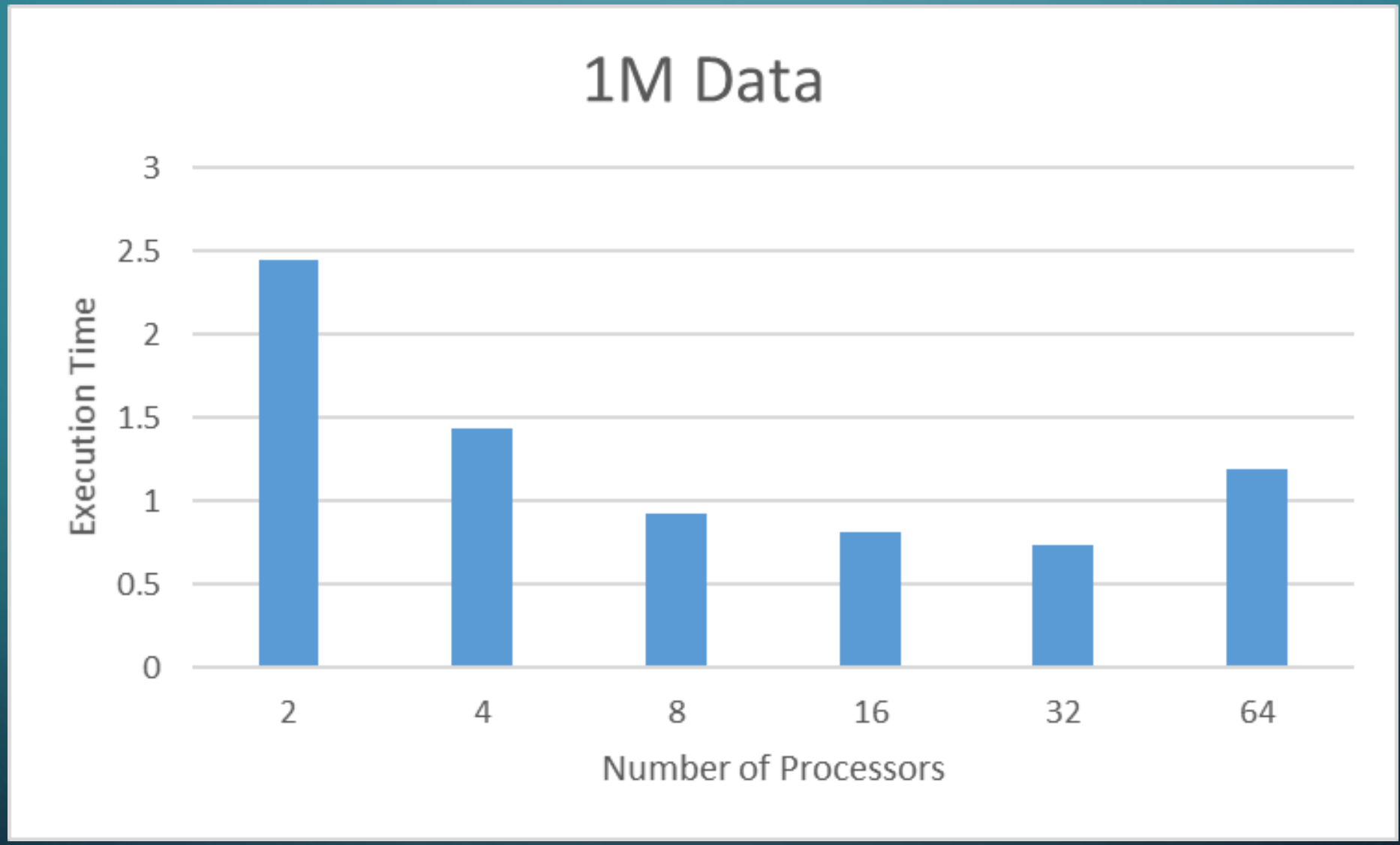
IMPLEMENTATION OF PARALLEL QUICKSORT

- Created sample data set and chosen the median which is the pivot element
- Distributed the data among all the processors using send and receive command.
- The pivot is chosen and sent to all the processors
- Called the parallel quick sort function.
- Function calls the partition function to partition the data.
- Exchanges the low list and up list based on which processor using send and recv command.
- Recursively it calls itself.
- Once the iteration reaches log (number of pes).
- Called sequential quick sort function.


6
2
9
3
Partition
6
2
9
3
Partition
Partition
6
2
9
3
Merge
Merge
2
6
3
9
Merge
2
3
6
9

RE


1/2M Data
Execution Time
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
2
4
8
16
32
64
Number of Processors


1M Data
Execution Time
3
2.5
2
1.5
1
0.5
0
2
4
8
16
32
64
Number of Processors


5M Data
Number of Processors
14
12
10
8
6
4
2
0
2
4
8
16
32
64
Execution Time


10M Data
30
25
20
15
10
5
0
Number of Processors
2
4
8
16
32
Execution Time


20M Data
60
50
40
30
20
10
0
Number of Processors
2
4
8
16
32
Execution Time

CORRECTION

- Included median to find the pivot element.

CHALLENGES

- MPI4py documentation.
- Proper barriers since the data is transferred to particular processor.
- Recursion in parallel.

REFERENCES

- MPI documents
- Miller Algorithms Sequential and Parallel A Unified Approach


Thanks!