ADVANCED TOPICS IN ALGORITHMS (ATIA)

Parallel Processing: An Introduction,

Parallel Sorting

INTRODUCTION TO PARALLEL PROCESSING

What is Parallel Processing?

Parallel processing is another method used to improve performance in a computer system,

When a system processes two different instructions simultaneously, it is performing parallel processing

Kinds of Parallel Processing

Parallelism in Uniprocessor Systems

Parallelism in Multiprocessor Systems

- Flynn's Classification
- System Topologies
- MIMD System Architectures

Parallelism in Uniprocessor Systems

A uniprocessor (one CPU) system can perform two or more tasks simultaneously.
The tasks are not related to each other.
So, a system that processes two different instructions simultaneously could be considered to perform parallel processing

Example 1: The instruction pipeline
Example 2: A vectored arithmetic unit

Parallelism in Multiprocessor Systems

More than one processor performing tasks simultaneously. Since multiprocessor systems are more complicated than uniprocessor systems, there are many different ways to organize the processors and memory, so a researcher, Michael J. Flynn proposed a classification based on the flow of instructions and data within the computer called Flynn's classification:

A computer is classified by whether it processes a single instruction at a time or multiple instructions simultaneously, and whether it operates on one or multiple data sets.

Categories of Flynn's Classification

SISD: Single instruction with single data
SIMD: Single instruction with multiple data
MISD: Multiple instruction with single data
MIMD: Multiple instruction with multiple data

System Topologies

The pattern of connections between its processors.
Various factors, typically involving a cost-performance tradeoff, determine which topology a computer designer will select for a multiprocessor system.

Although topologies differ greatly, standard metrics-diameter and bandwidth-are used to quantify them

Diameter: the maximum distance between two processors in the computer system.
Bandwidth: the capacity of a communication link multiplied by the number of such links in the system.
Bisection bandwidth: the maximum data transfer that could occur at the bottleneck in the topology.

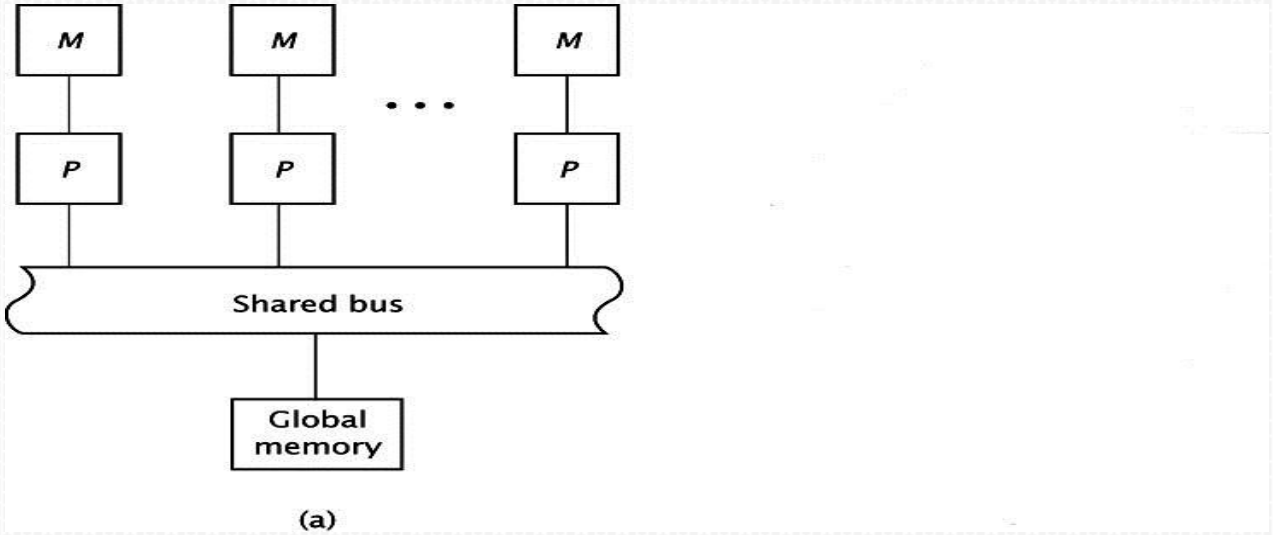
Examples of Topology

Shared Bus Topology

Processors communicate with each other exclusively through this bus.

The bus can only handle only one data transmission at a time.

In most shared busses, processors directly communicate with their own local memory.



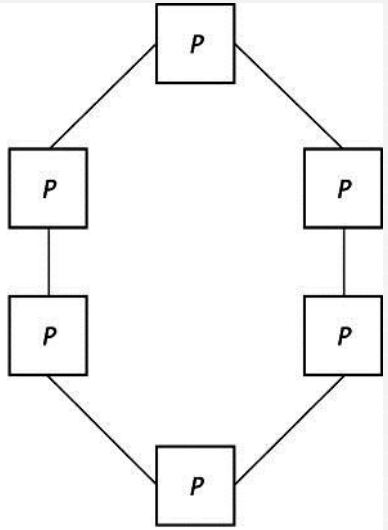
Examples of Topology

Ring Topology

The ring topology uses direct connections between processors instead of a shared bus.

This allows all communication links to be active simultaneously.

Data may have to travel through several processors to reach its destination



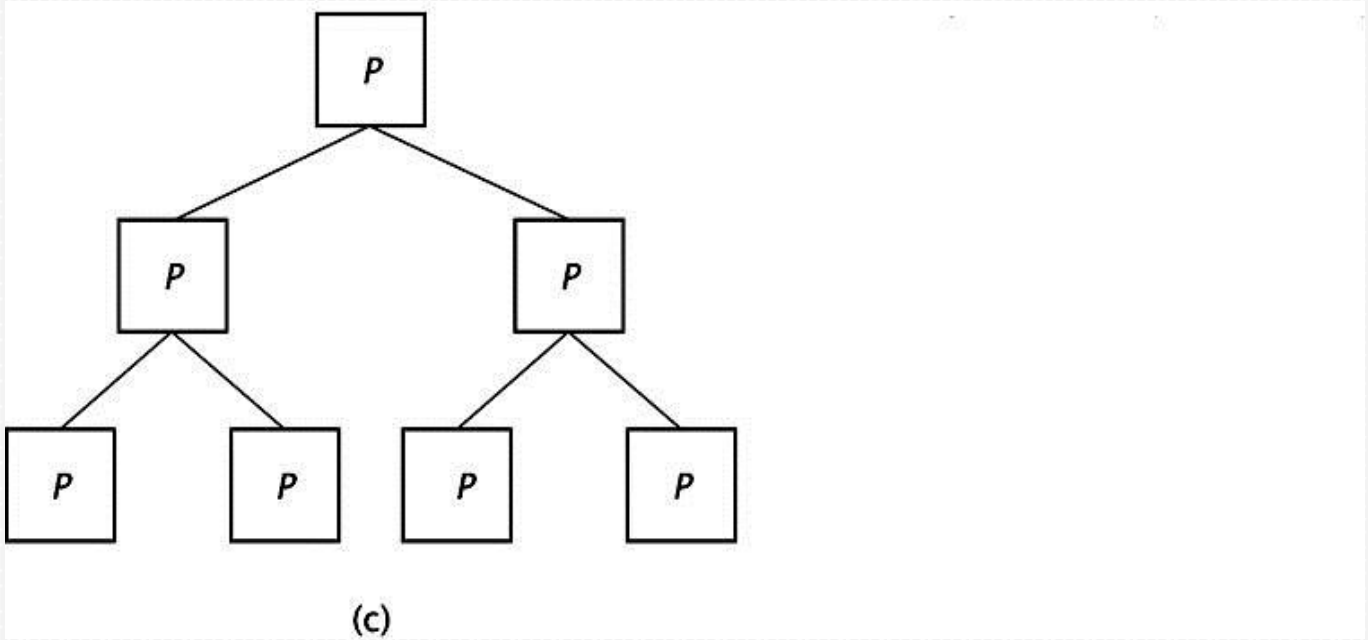
(b)

Examples of Topology

Tree Topology

Like the ring, it uses direct connections between processors; each having three connections.

There is only one unique path between any pair of processors

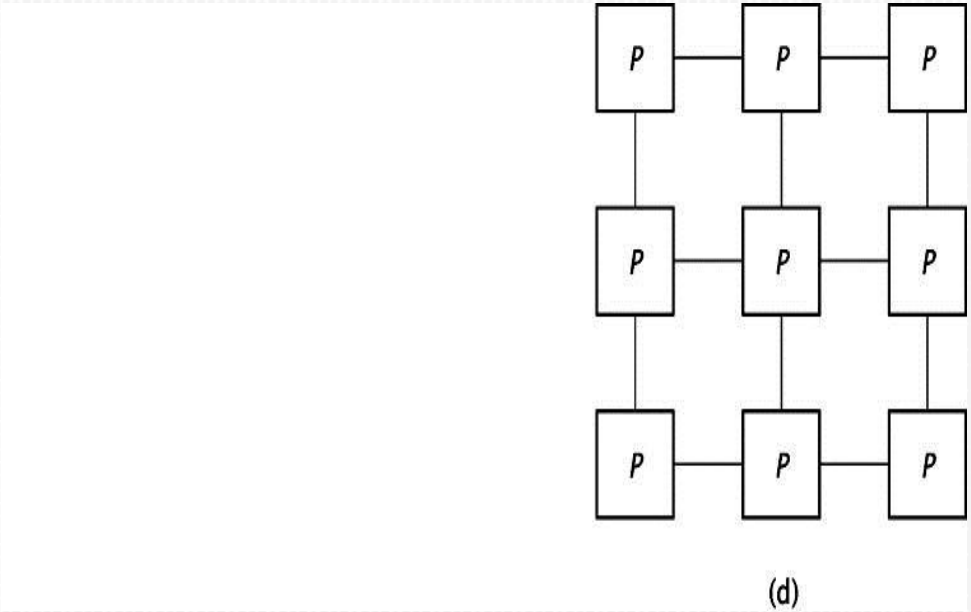


(c)

Examples of Topology

Mesh Topology

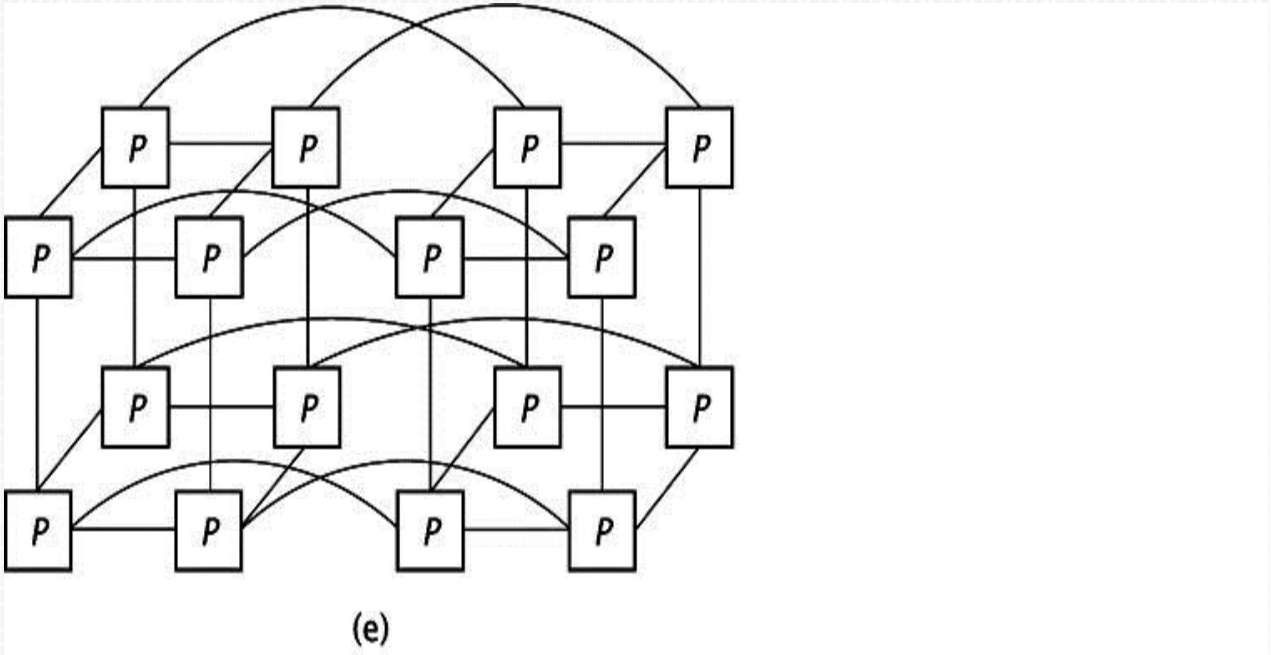
Every processor connects to the processors above and below it, and to its right and left.



Examples of Topology

Hypercube Topology

The hypercube is a multidimensional mesh topology. Each processor connects to all other processors whose binary values differ by one bit.



(e)

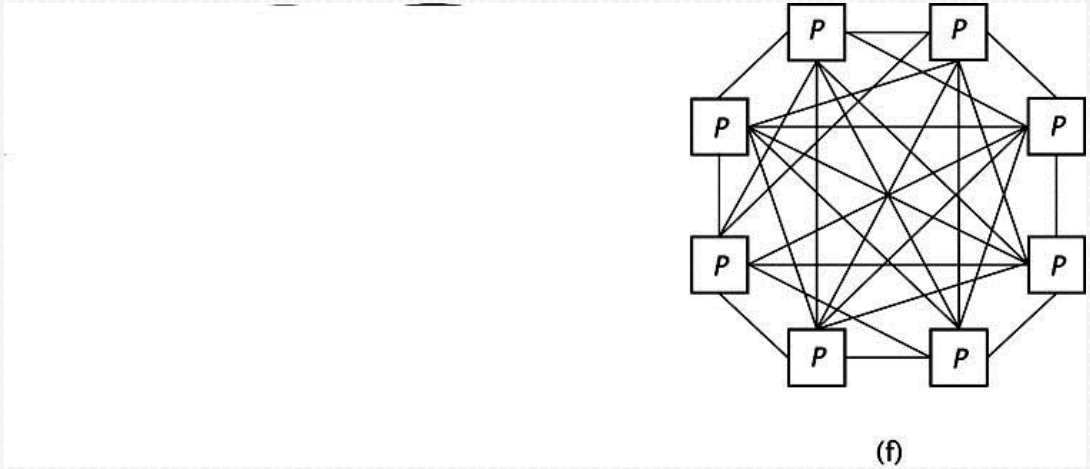
Examples of Topology

Completely connected Topology

In the most extreme connection scheme, the processors are completely connected.

Every processor has (n-1) connections, one to each of the other processors.

This increases the complexity of the processors as the system grows, but offers maximum communication capabilities.



(f)

MIMD System Architectures

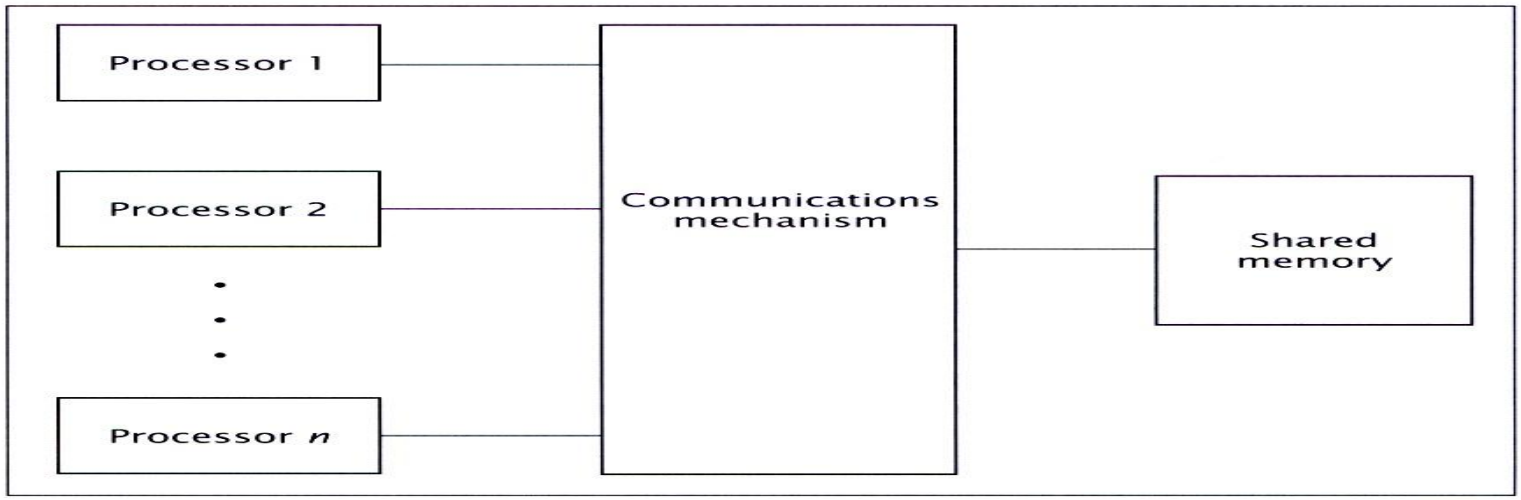
The architecture of an MIMD system, as opposed to its topology, refers to its connections with respect to system memory.

There are two types of architectures:

- Uniform memory access(UMA)
- Non-uniform memory access(NUMA)

Uniform Memory Access

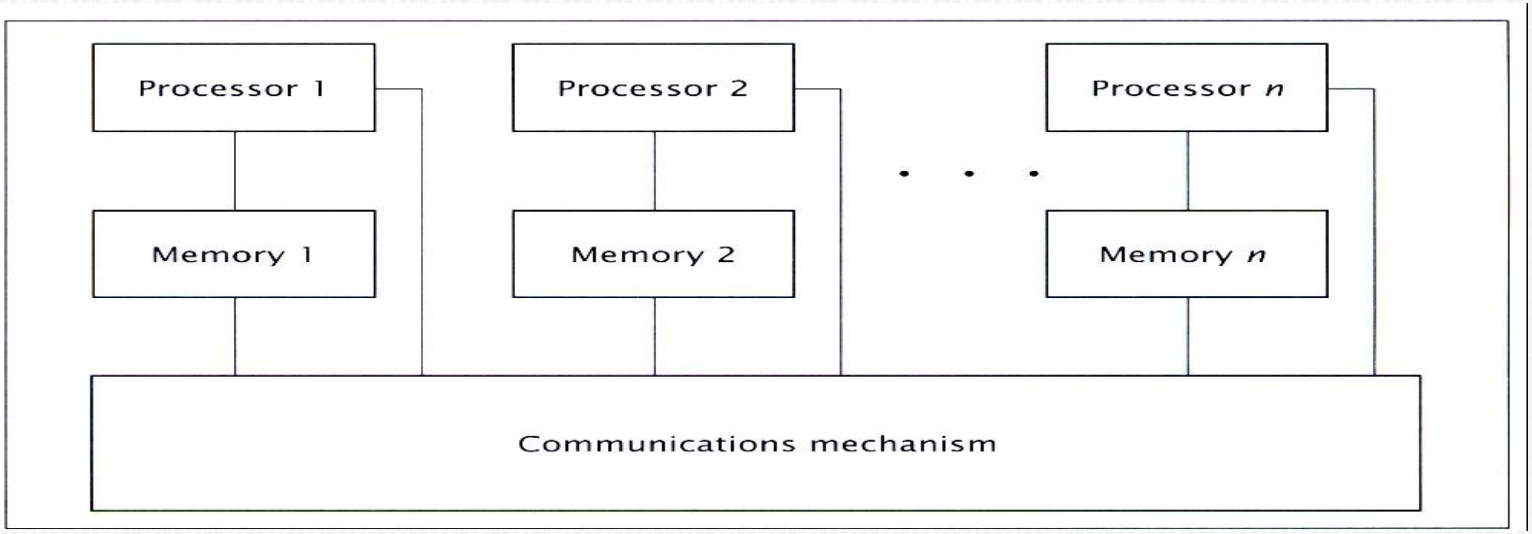
The UMA gives all CPUs equal (uniform) access to all locations in shared memory.
They interact with shared memory by some communications mechanism like a simple bus or a complex multistage interconnection network



Non-Uniform Memory Access

In contrast to UMA architectures, NUMA do not allow uniform access to all shared memory locations, this architecture still allows all processors to access all shared memory locations.

However, each processor can access the memory module closest to it, its local shared memory, more quickly than the other modules, so the memory access times are non-uniform.



Parallel Sorting Algorithms

Potential Speedup

O(n log n) optimal sequential sorting algorithm

Best we can expect based upon a sequential sorting algorithm using n processors is:

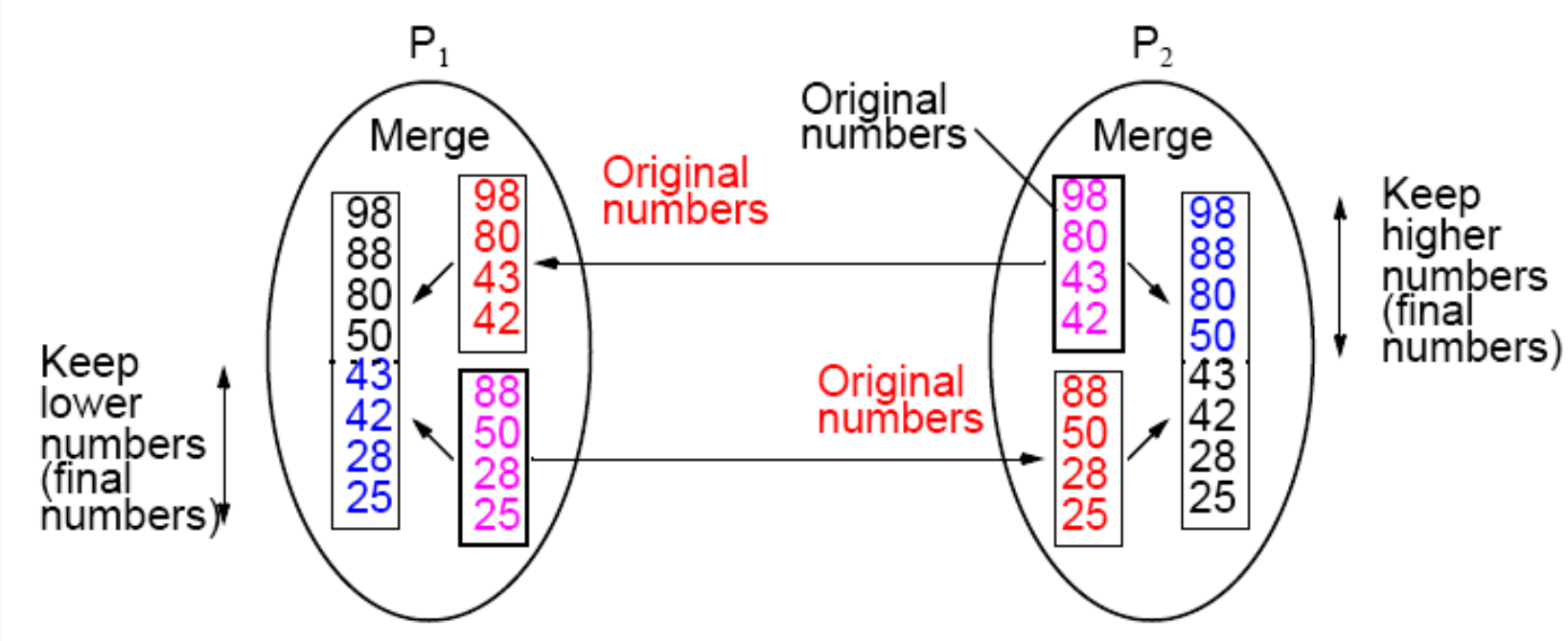
$$\textbf{optimal parallel time complexity} = \frac{O(n \log n)}{n} = O(\log n)$$

Compare-and-Exchange Sorting Algorithms

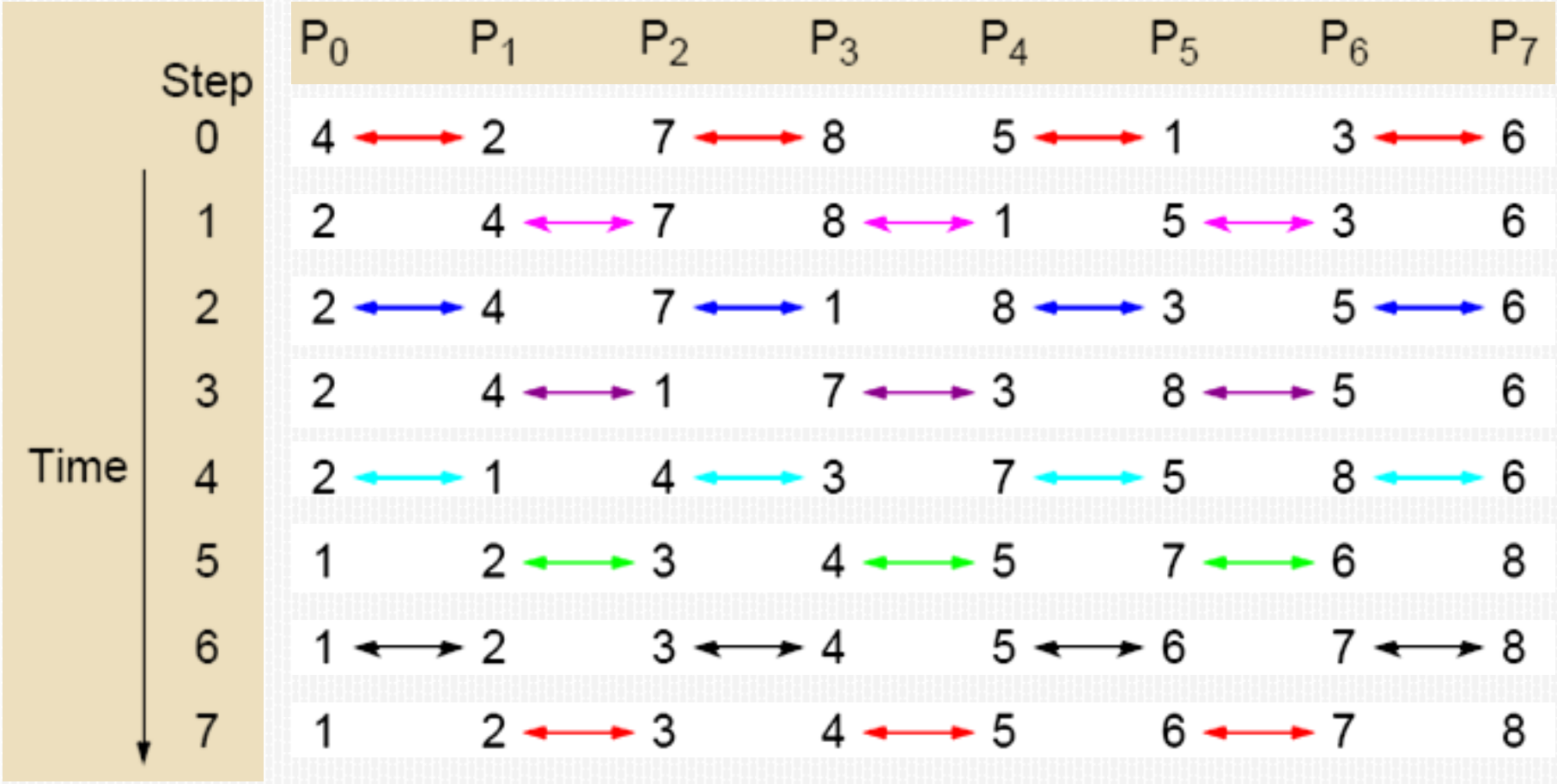
Form the basis of several, if not most, classical sequential sorting algorithms.

Two numbers, say *A* and *B*, are compared between P_0 and P_1.

Compare-and-Exchange Two Sublists



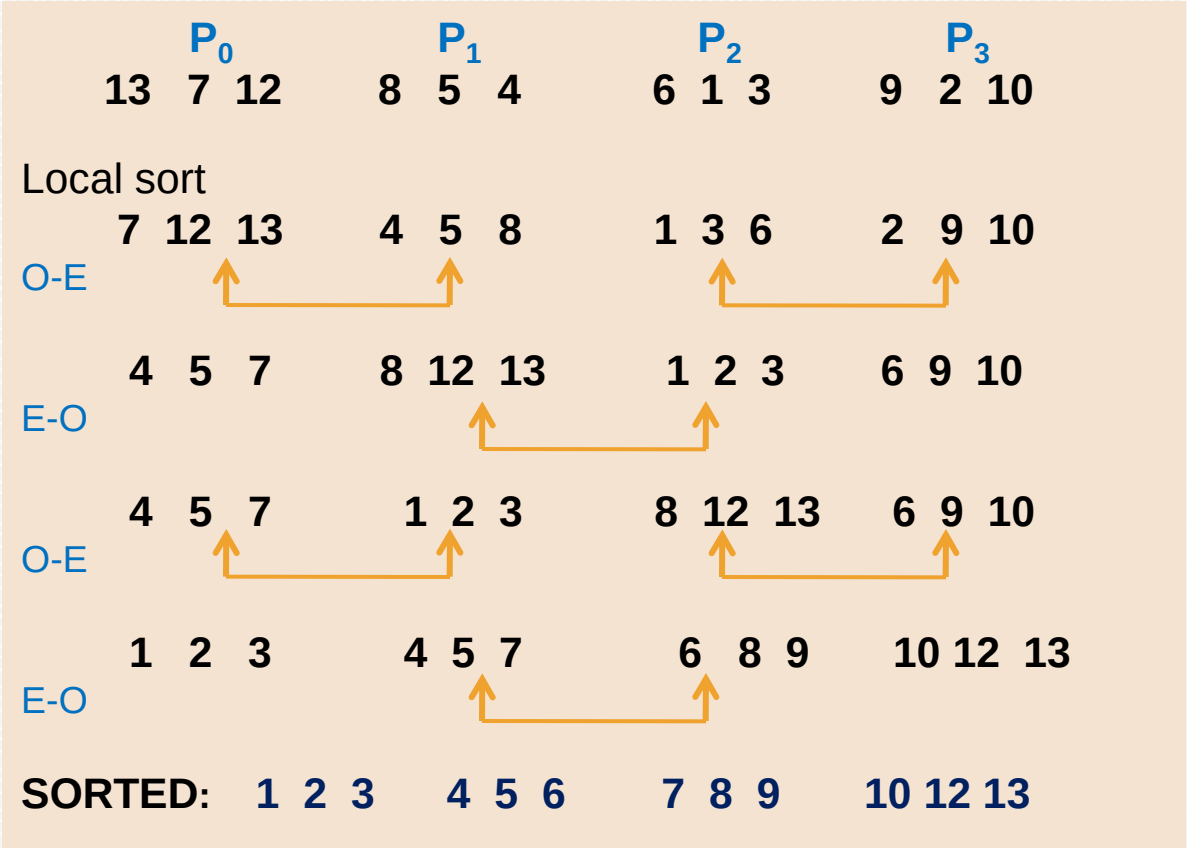
Odd-Even Transposition Sort - example



Parallel time complexity: $T_{par} = O(n)$ (for P=n)

Odd-Even Transposition Sort – Example (N >> P)

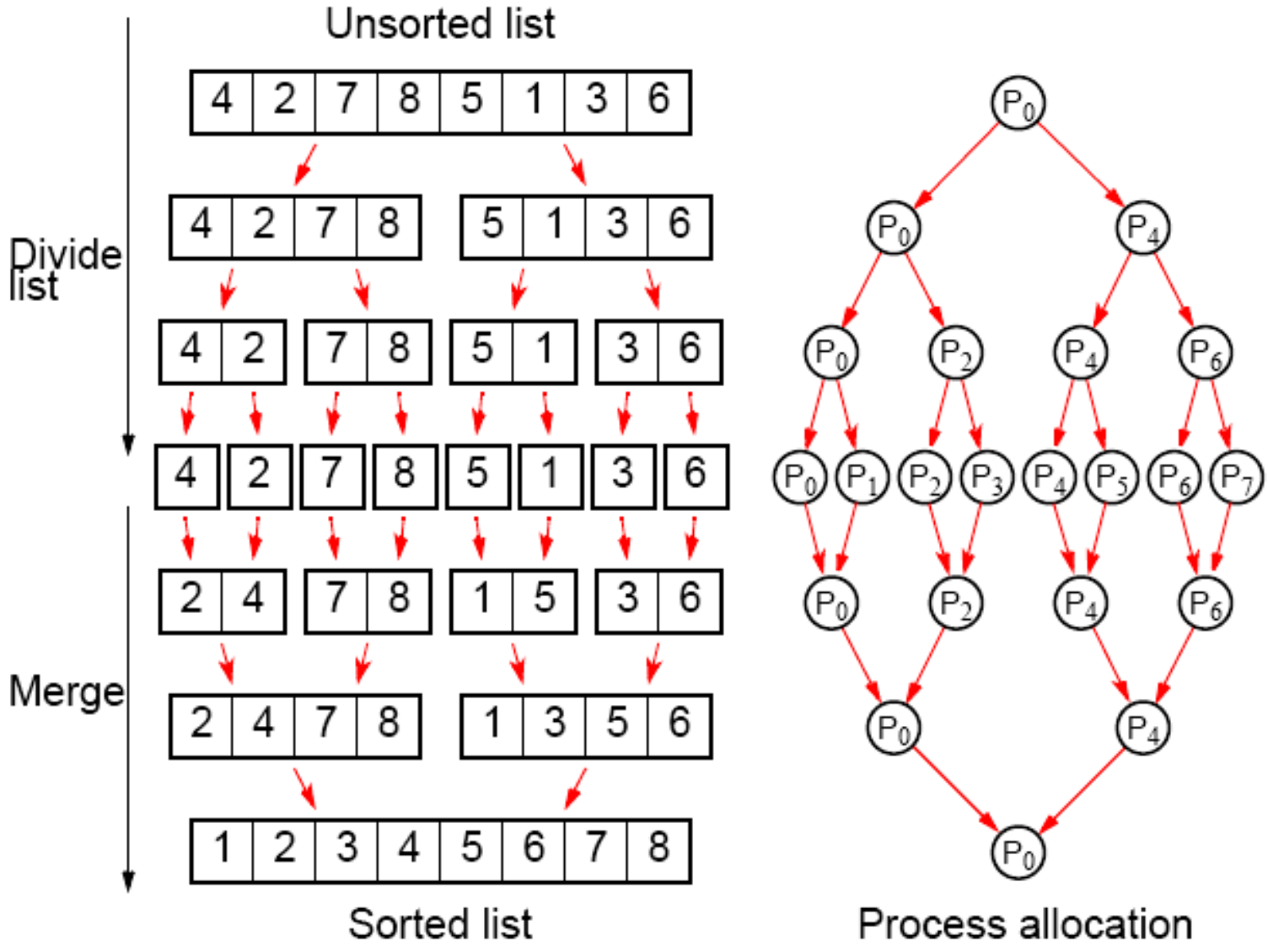
Each PE gets n/p numbers. First, PEs sort n/p locally, then they run odd-even trans. algorithm each time doing a merge-split for 2n/p numbers.



Time complexity: T_{par} = (Local Sort) + (p merge-splits) +(p exchanges)

$$T_{par} = (n/p)\log(n/p) + p*(n/p) + p*(n/p) = (n/p)\log(n/p) + 2n$$

Parallelizing Mergesort



Mergesort - Time complexity

Sequential:

$$T_{seq} = 1 * n + 2 * \frac{n}{2} + 2^2 * \frac{n}{2^2} + \ldots + 2^{\log n} * \frac{n}{2^{\log n}}$$

$$T_{seq} = O(n \log n)$$

Parallel:

$$T_{par} = 2\left(\frac{n}{2^0} + \frac{n}{2^1} + \frac{n}{2^2} + \ldots + \frac{n}{2^k} \ldots + 2 + 1 \right)$$

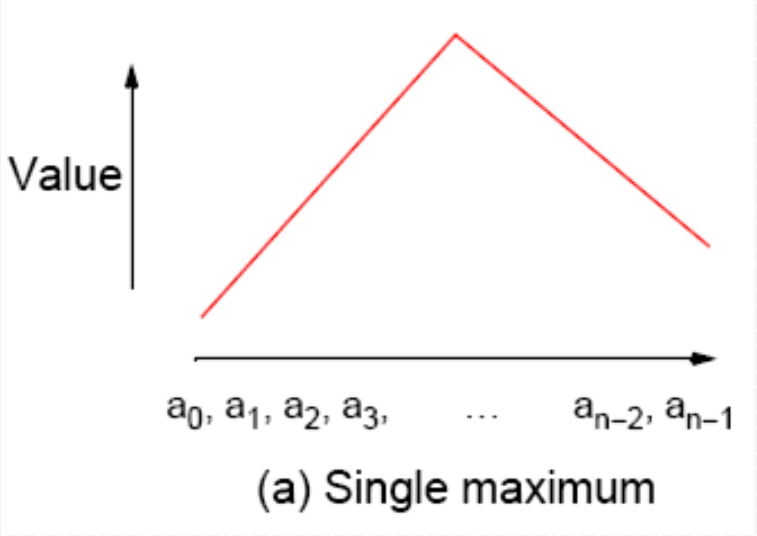
$$= 2n\left(2^0 + 2^{-1} + 2^{-2} + \ldots + 2^{-\log n}\right)$$

$$T_{par} = O(4n)$$

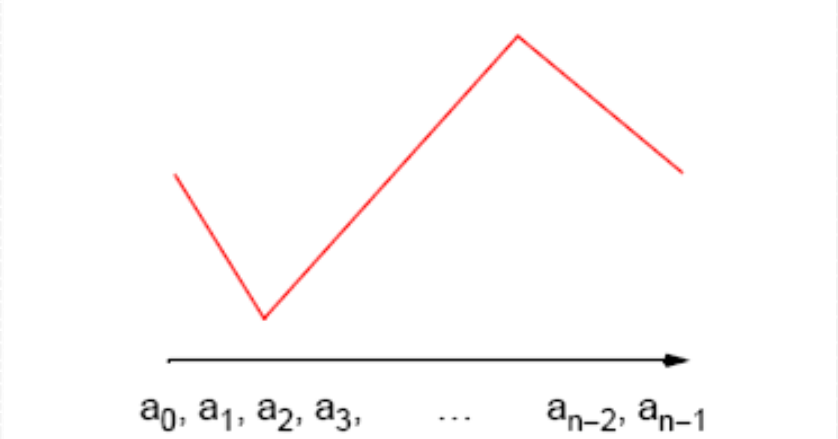
Bitonic Mergesort

Bitonic Sequence

A *bitonic sequence* is defined as a list with no more than one **LOCAL MAXIMUM** and no more than one **LOCAL MINIMUM**. (Endpoints must be considered - wraparound)

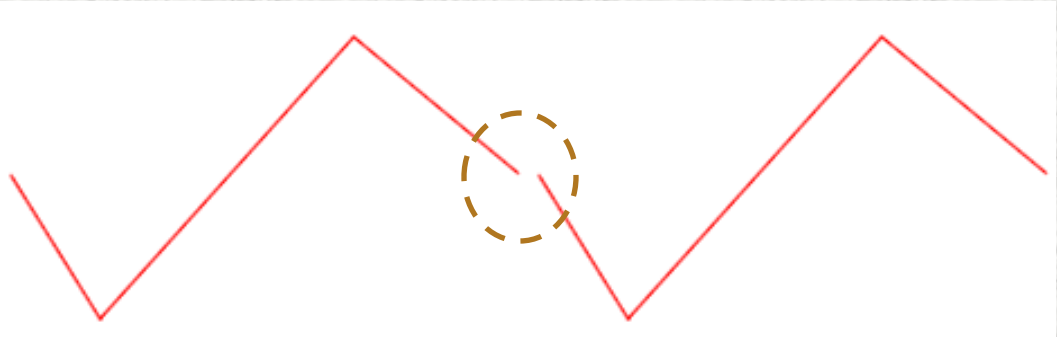


(a) Single maximum



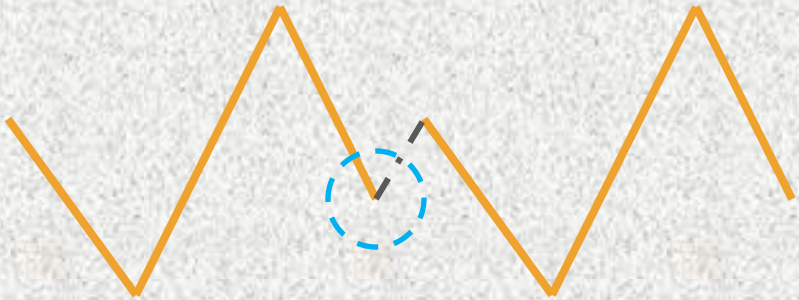
(b) Single maximum and single minimum

A *bitonic sequence* is a list with no more than one **LOCAL MAXIMUM** and no more than one **LOCAL MINIMUM**.
(**Endpoints must be considered - wraparound**)



This is ok!
1 Local MAX; 1 Local MIN
The list is bitonic!

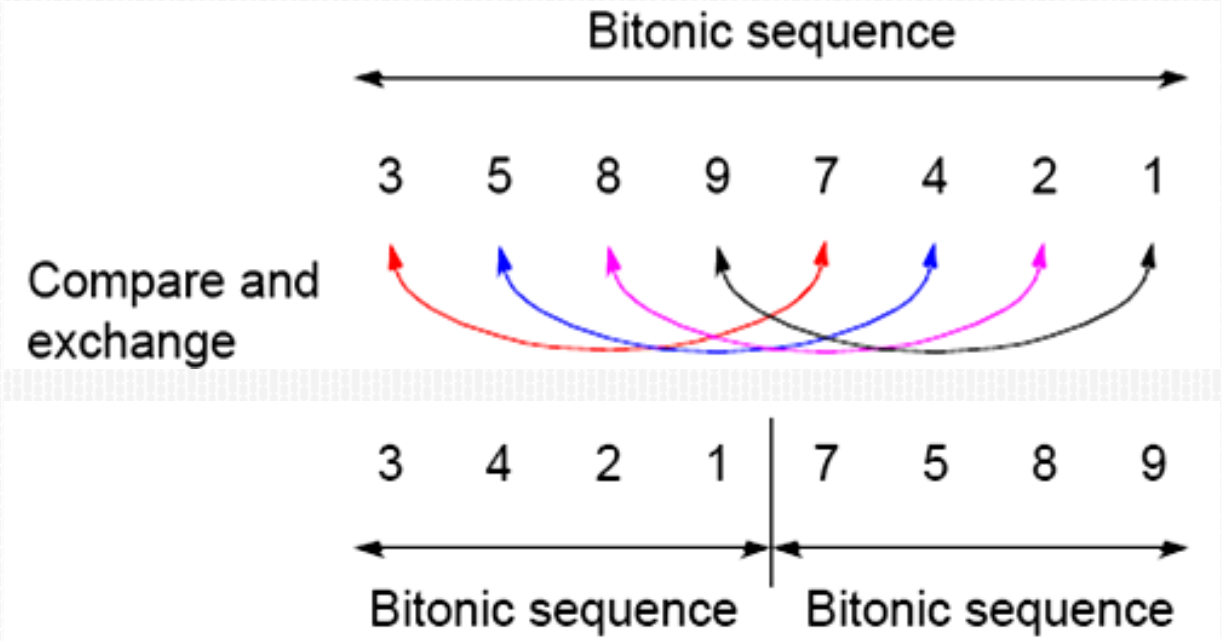
This is **NOT** bitonic! Why?



1 Local MAX; 2 Local MINs

Binary Split

1. Divide the **bitonic list** into two equal halves.
2. Compare-Exchange each item on the first half with the corresponding item in the second half.



Result:

Two bitonic sequences where the numbers in one sequence are all less than the numbers in the other sequence.

Repeated application of binary split

Bitonic list:

24 20 15 9 4 2 5 8 | 10 11 12 13 22 30 32 45

Result after Binary-split:

10 11 12 9 4 2 5 8 | 24 20 15 13 22 30 32 45

If you keep applying the BINARY-SPLIT to each half repeatedly, you will get a SORTED LIST !

10 11 12 9 . 4 2 5 8 | 24 20 15 13 . 22 30 32 45

4 2 . 5 8 10 11 . 12 9 | 22 20 . 15 13 24 30 . 32 45

4 . 2 5 . 8 10 . 9 12 . 11 15 . 13 22 . 20 24 . 30 32 . 45

2 4 5 8 9 10 11 12 13 15 20 22 24 30 32 45

Q: How many parallel steps does it take to sort ?

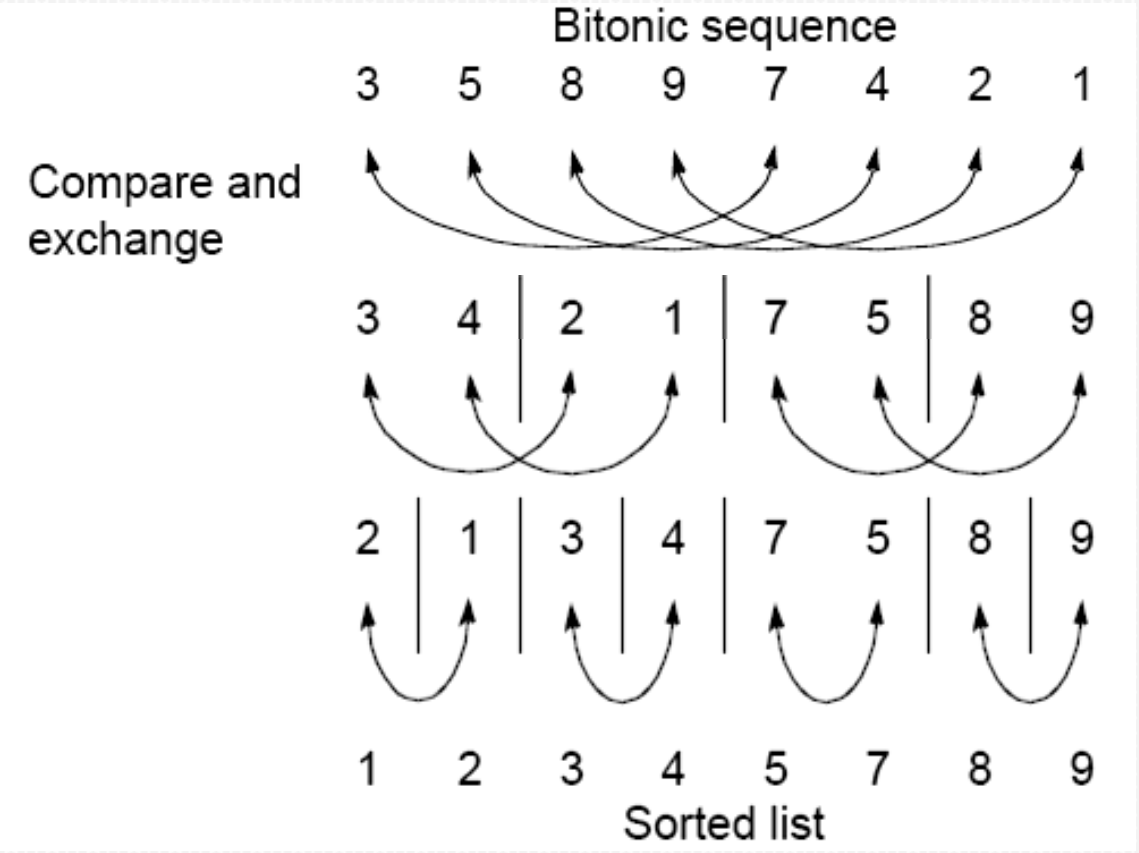
A: **log *n***

Sorting a bitonic sequence

Compare-and-exchange moves smaller numbers of each pair to left and larger numbers of pair to right.

Given a bitonic sequence,

recursively performing '*binary split*' will sort the list.



Sorting an arbitrary sequence

To sort an unordered sequence, sequences are merged into larger bitonic sequences, starting with pairs of adjacent numbers.

By a compare-and-exchange operation, pairs of adjacent numbers formed into increasing sequences and decreasing sequences. Pairs form a bitonic sequence of twice the size of each original sequences.

By repeating this process, bitonic sequences of larger and larger lengths obtained.

In the final step, a single bitonic sequence sorted into a single increasing sequence.

Bitonic Sort

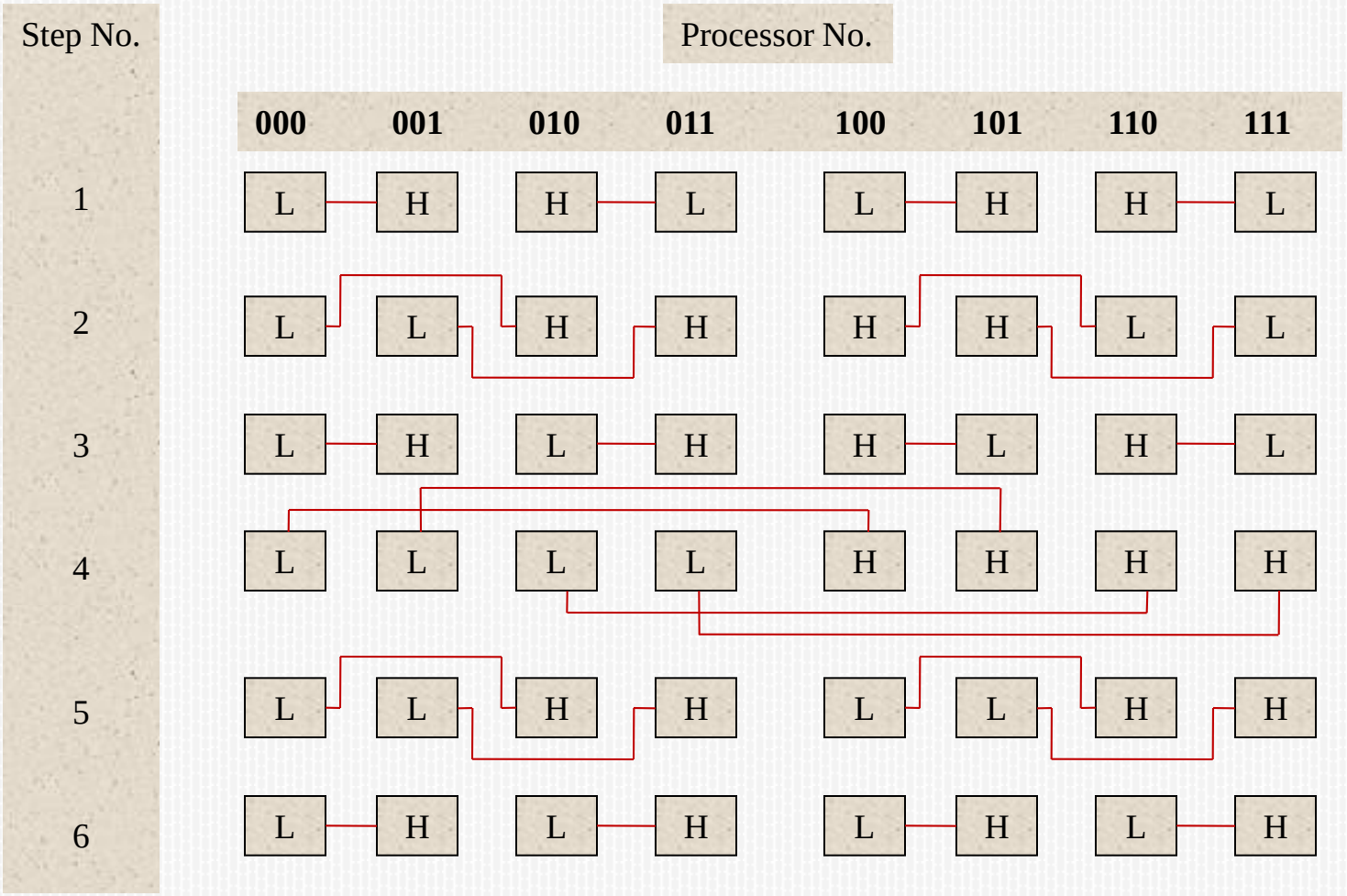
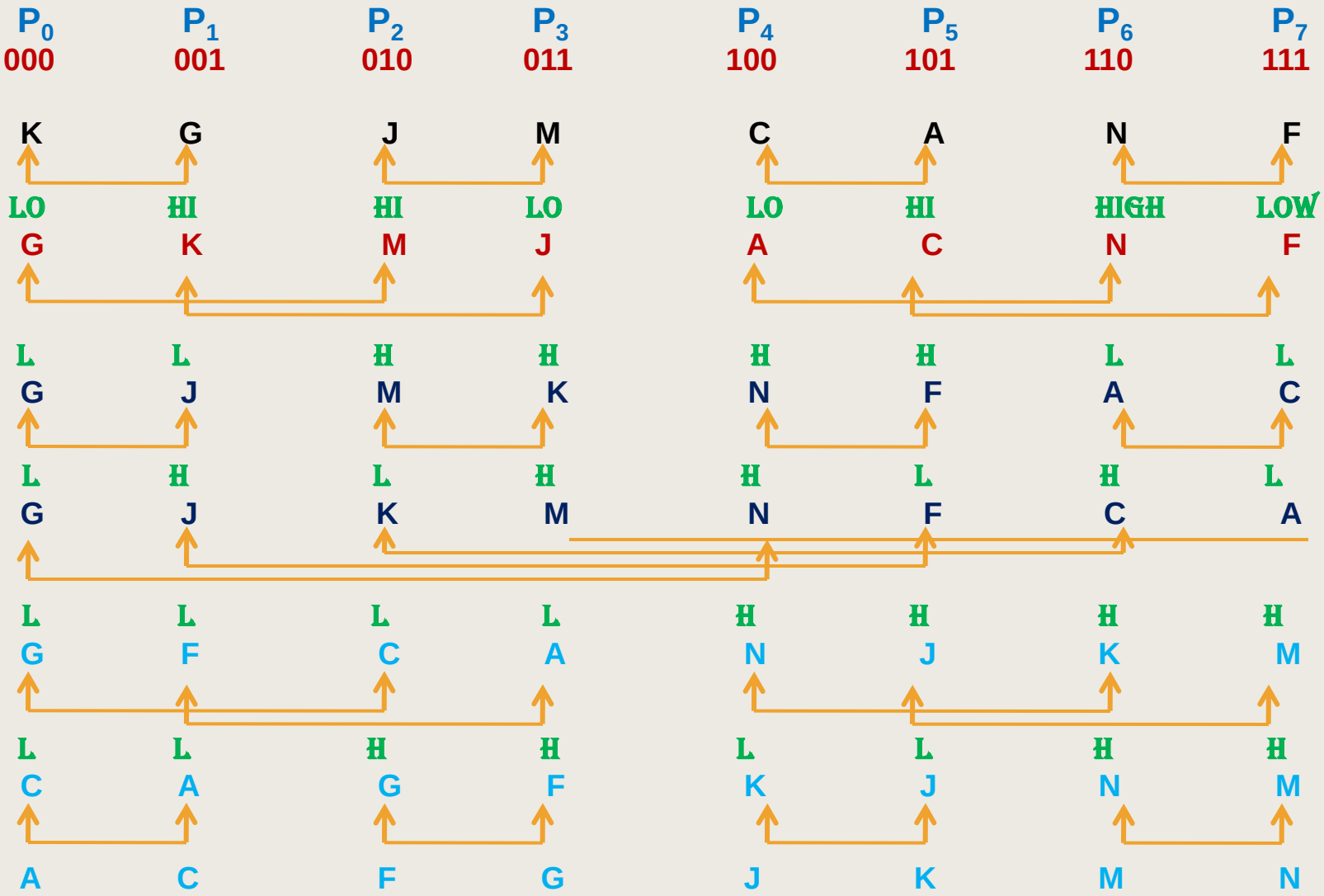


Figure 2: Six phases of Bitonic Sort on a hypercube of dimension 3

Bitonic sort (for N = P)

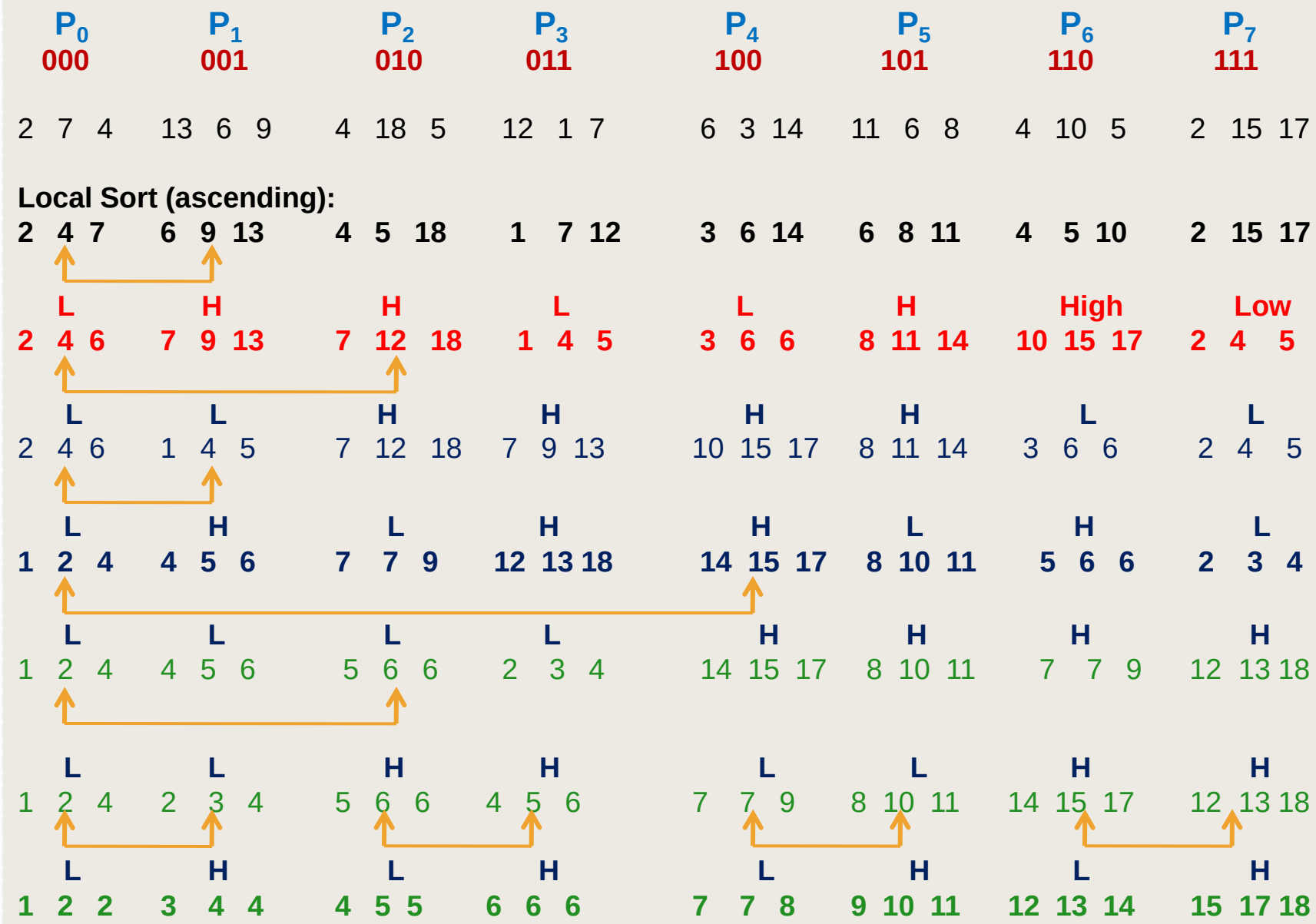


Number of steps (P=n)

In general, with $n = 2^k$, there are k phases, each of 1, 2, 3, …, k steps. Hence the total number of steps is:

$$T_{par}^{bitonic} = \sum_{i=1}^{i=\log n} i = \frac{\log n(\log n + 1)}{2} = O(\log^2 n)$$

Bitonic sort (for N >> P)



Number of steps (for N >> P)

$$T_{par}^{bitonic} = \text{Local Sort} + \text{Parallel Bitonic Merge}$$

$$= \frac{N}{P}\log\frac{N}{P} + 2\frac{N}{P}(1+2+3+\ldots+\log P)$$

$$= \frac{N}{P}\{\log\frac{N}{P} + 2(\frac{\log P(1+\log P)}{2})\}$$

$$= \frac{N}{P}(\log N - \log P + \log P + \log^2 P)$$

$$T_{par}^{bitonic} = \frac{N}{P}(\log N + \log^2 P)$$

Parallel sorting - summary

Computational time complexity using P=n processors

- Odd-even transposition sort - **O(n)**
- Parallel mergesort - **O(n)**
  unbalanced processor load and Communication
- **Bitonic Mergesort - O(log^2n)** **(** BEST! **)**
- ??? Parallel Shearsort - **O(√n *log*n)**
- Parallel Rank sort - **O(n)** (for P=n)