Parallel Algorithm Design

Case Study: Sorting

Xian-He Sun
Department of Computer Science
Illinois Institute of Technology
sun@iit.edu

Tridiagonal Solvers

- Parallel Algorithms
  - The Partition Method, The PPT Algorithm, The PDD Algorithm, The LU Pipelining Algorithm, The PTH Method and PPD Algorithm
- They are the best parallel (scalable) algorithm, but not the best sequential algorithms
- Some design optimizations are in the communication
- General considerations, and non-numerical algorithms

Parallelizing A Sequential Algorithm

Method Description :

–Detect and exploit sny inherent parallelism in an Existing Sequential Algorithm.

–Parallel Implementation of a parallelizable Code Segment.

Remark :

–A parallelizing method Is in Most Common Use and effective in common.

–Not all sequential algorithms Can Be parallelized.

–A Best Sequential Algorithm is Uncertain to Be parallelized a Good Parallel Algorithm.

–Many Sequential Numerical Algorithms Can Be parallelized Directly into Effective Parallel Numerical Algorithms.

Designing A New Parallel Algorithm

Method Description :

– In terms of the description of a given problem, we redesign or invent a new parallel algorithm without regard to the related a sequential algorithm.

Remark :

– Usually, a very efficient parallel algorithm can be obtained using this method because of investigating inherent feature of the problem.
– To invent a new parallel algorithm is a challenge and creative work.
– To do this work, the designer' s comprehensive background should be good enough.
– To design a parallel string matching algorithm on SIMD-SM model in terms of the periodicity in strings is a good example.

Borrowing Other Well-known Algorithms

Method Description :

- To find relationship between to be solved problem and well-known problem.
- To design a similar algorithm that solves a given problem using a well-known algorithm.

Remark :

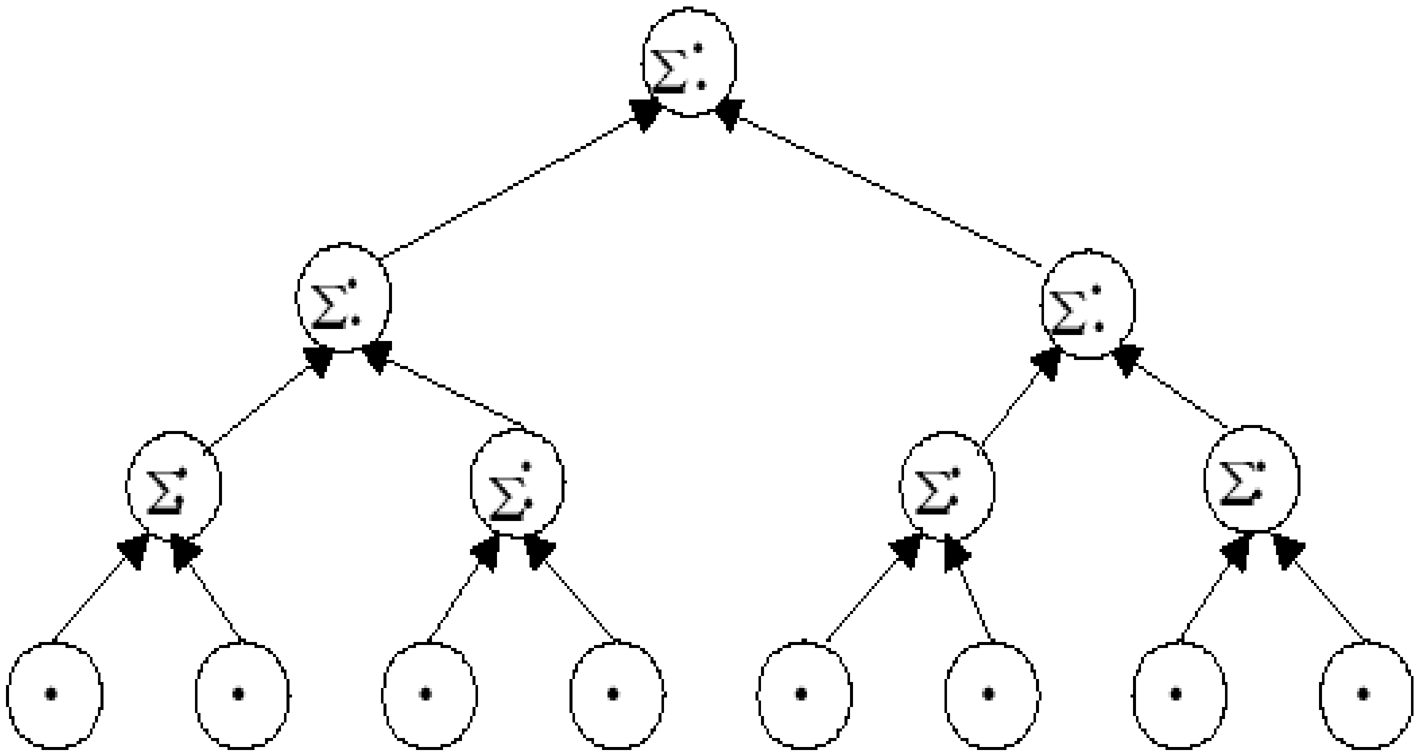
- This work is very skilled work.
- The designer should have a rich and practical experience of designing the different kind of parallel algorithms.
- To find all-pairs shortest-path using a matrix multiplication algorithm is a good example.

Balanced Trees

Introduced here techniques are just as general guidelines for designing parallel algorithms, rather than as a design manual of directly applicable methods.

- Description :
  - Building a balanced binary tree on input elements.
  - Traversing the tree forward and backward to and from the root.
- Remark :
  - It is the most elementary and useful method for computing prefix-sum, finding the maximum or minimum of n elements etc.
  - Complexity of such an algorithm: $t(n)=O(\log n)$, $p(n)=n/2$.

Balanced Trees



© Sun

Doubling Technique

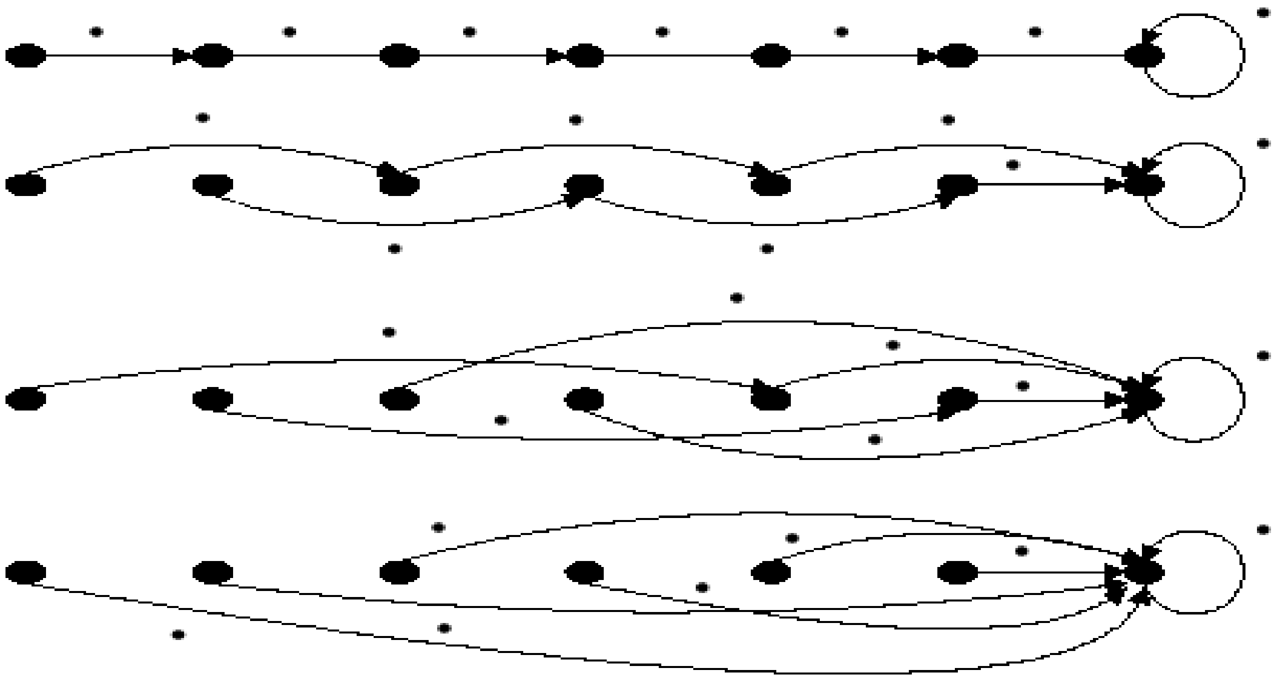
Description :

- It is also called pointer jumping, path doubling.
- The computation proceeds by a recursive application of the calculation in hand to all elements over a certain distance from each individual element.
- This distance doubles in successive steps. After k steps the computation has been performed over all elements within a distance of 2 k .

Remark :

- This technique is normally applied to an array or a list of elements. Ex. Computing list ranking, finding the roots of a forest etc.
- Complexity: t(n)=O(logn), p(n)=O(n).

Doubling Technique



© Sun

Partitioning Strategy

Description :

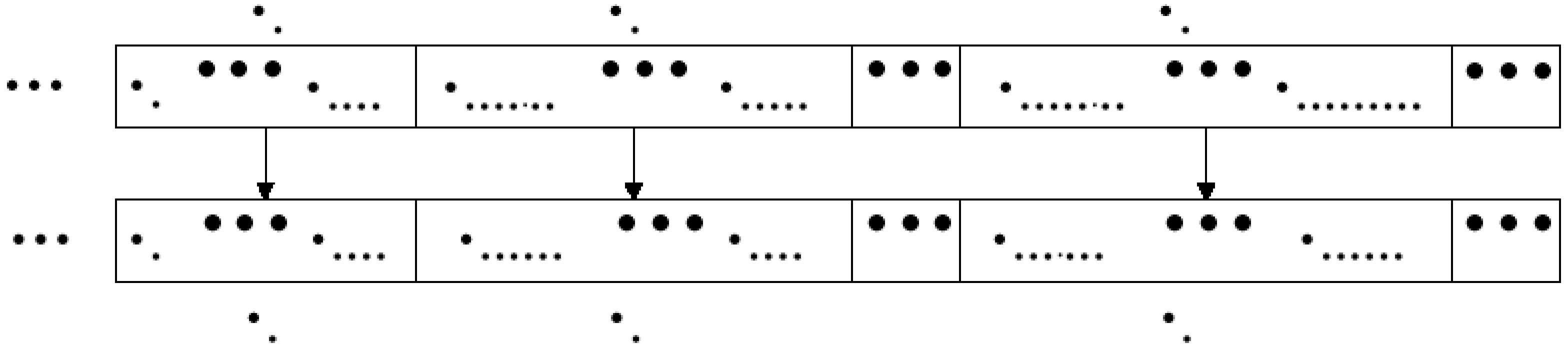
- Breaking up the given problem into several nonoverlapping subproblems of almost equal sizes.
- Solving concurrently these subproblems.

Remark :

- Usual partitioning method includes uniform partition, square root partition, logarithmic partition and functional partition etc.
- It is different from divide-and-conquer.
- Ex. Merging sort etc. (see below, where

  j(i)=rank(b ilogm :A)).

Partitioning Strategy



© Sun

Divide-and-Conquer

Description :

- Dividing the problem into several subproblems.
- Solving recursively the subproblems.
- Merging solutions of subproblems into a solution for original problem.

Remark :

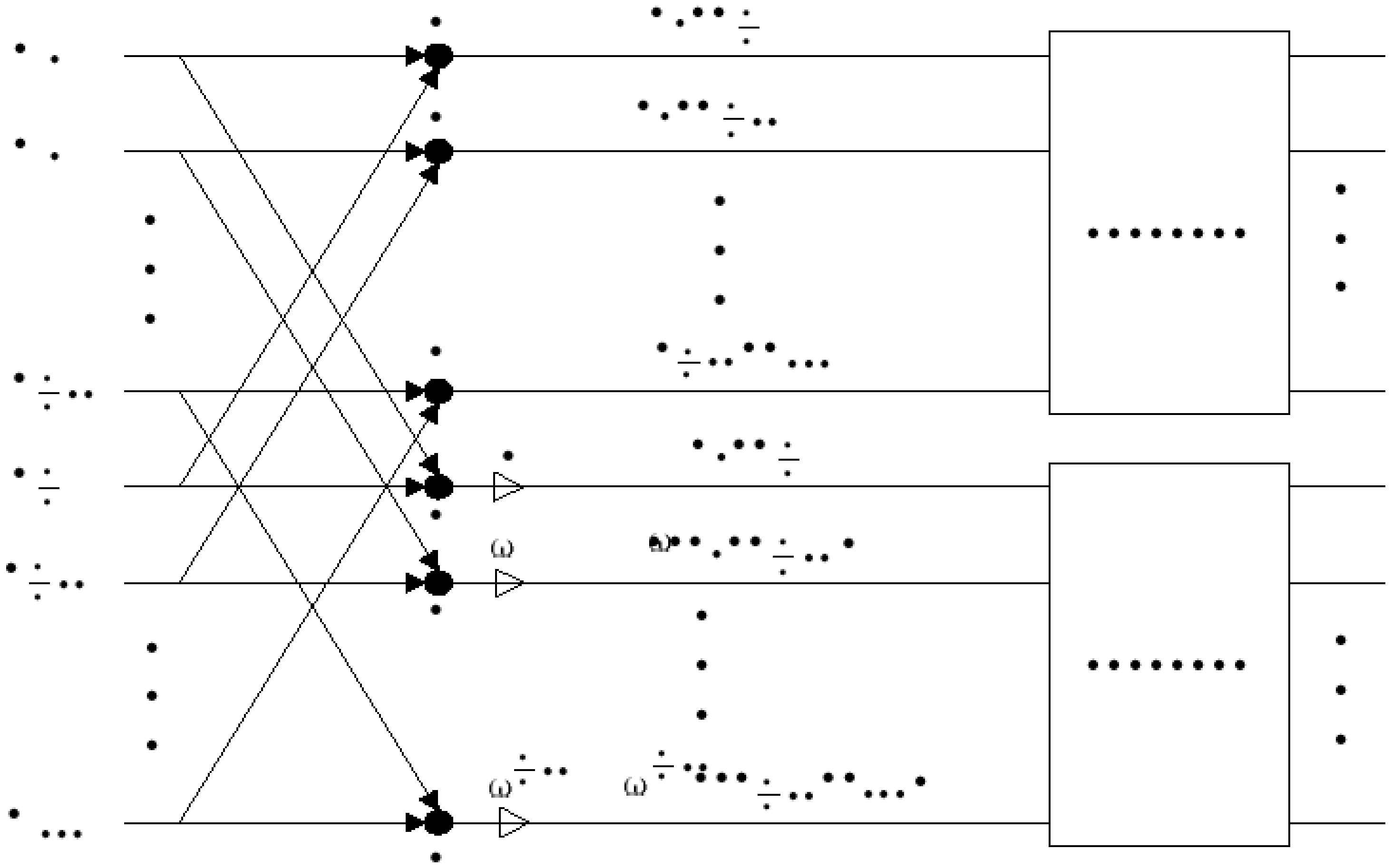
- This strategy has been shown to be effective in the development of fast sequential algorithms.
- It also leads to the most natural way of exploiting parallelism. Ex. Constructing bitonic merging network, finding convex hull etc.

More About Divide-and-Conquer

Difference between Divide-and-Conquer and Partitioning :

- They both have same goal of decomposing the problem into a set of subproblems that can be solved in parallel.
- The main work of the former lies in the merging of solution of subproblems.
- The main work of the latter lies in carefully decomposing the problem so that the solutions of subproblems can be combined easily to generate the original problem solution.

More About Divide-and-Conquer



Pipelining

Description :

- Breaking an algorithm into a sequence of segments in which the output of each segment is the input of its successor.
- All segments must produce results at the same rate.

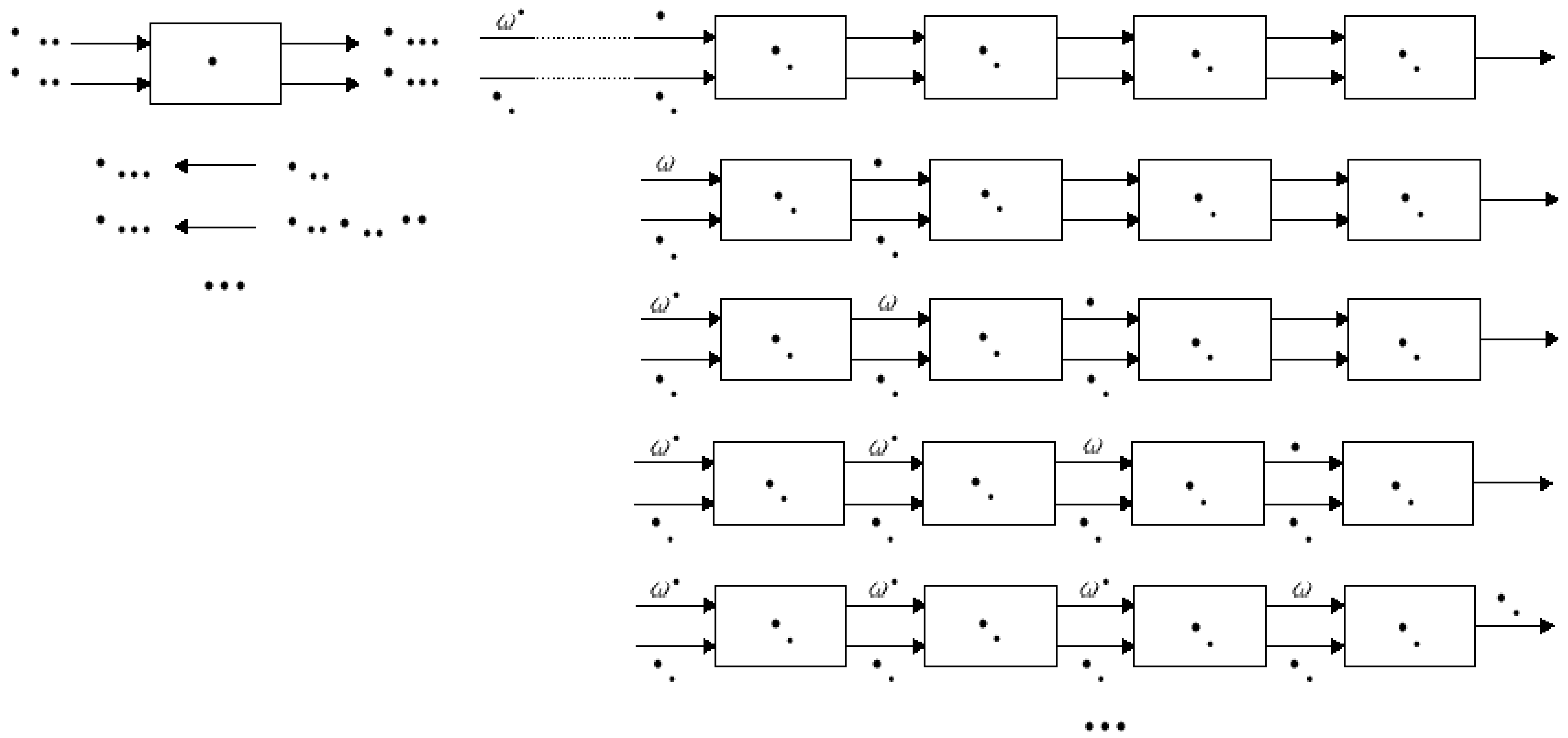
Remark :

- Pipelining is an important parallel technique that has been used extensively in parallel processing.
- Systolic algorithm is a special kind of pipelining algorithms.
- Ex. Using Horner rule to compute 4-point DFT:

$$y(\quad ^{j}) = \sum_{k=0}^{3} a_k \quad ^{jk}$$

© Sun

Pipelining



© Sun

Tridiagonal Solvers

- Parallel Algorithms
  - The Partition Method, The PPT Algorithm, The PDD Algorithm, The LU Pipelining Algorithm, The PTH Method and PPD Algorithm
- Design new parallel algorithm, borrowing existing mathematical results, Divide conquer, Balanced Tree, pipelining, hybrid design

Parallel Algorithms - sorting

(Slides are based on Fernando Silva note and the book “Parallel Programming Techniques & Applications Using Networked Workstations & Parallel Computers, 2nd ed.” de B. Wilkinson)

Sorting in Parallel

Why?

it is a frequent operation in many applications

Goal?

sorting a sequence of values in increasing order using n processors

Potential speedup?

best sequential algorithm has complexity $O(n \log n)$

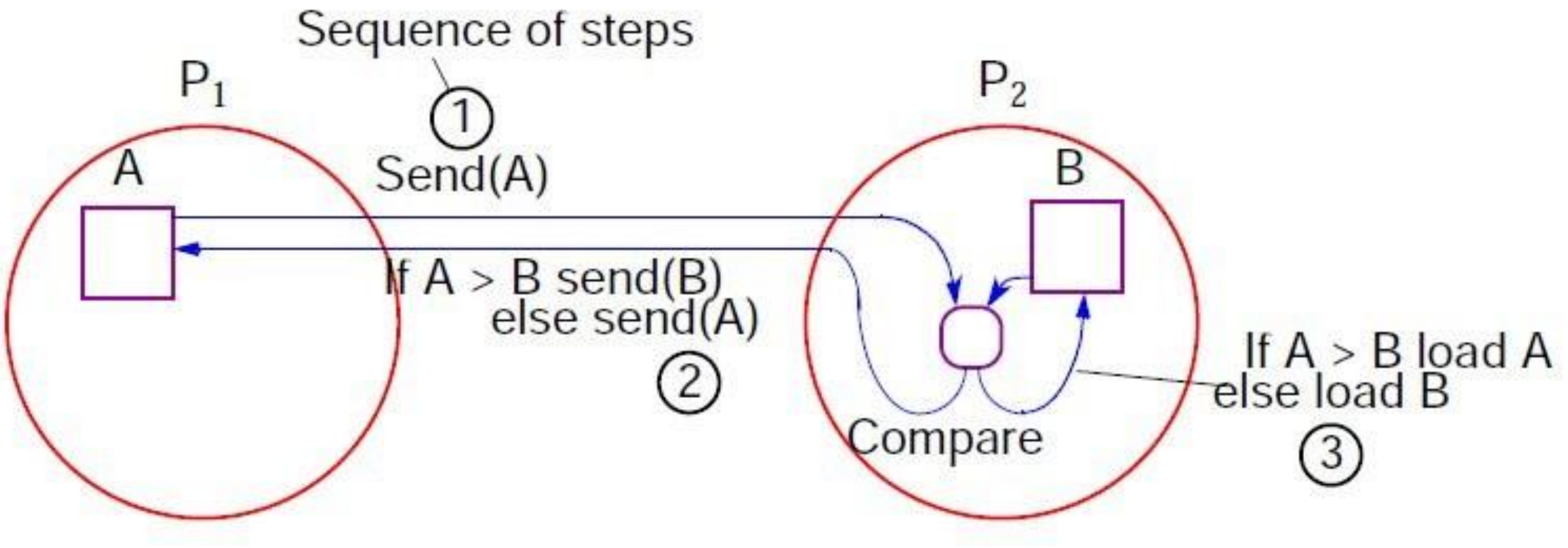
the best we can aim with a parallel algorithm, using n processors is: optimal complexity of a parallel sorting algorithm: $O(n \log n)/n = O(\log n)$

Compare-and-swap with message exchange(1/2)

Sequential sorting requires the comparison of values and swapping in the positions they occupy in the sequence. And, if it is in parallel? And, if the memory is distributed?

version 1:

P_1 send A to P_2, this compares B with A and sends to P_1 the $min(A, B)$.

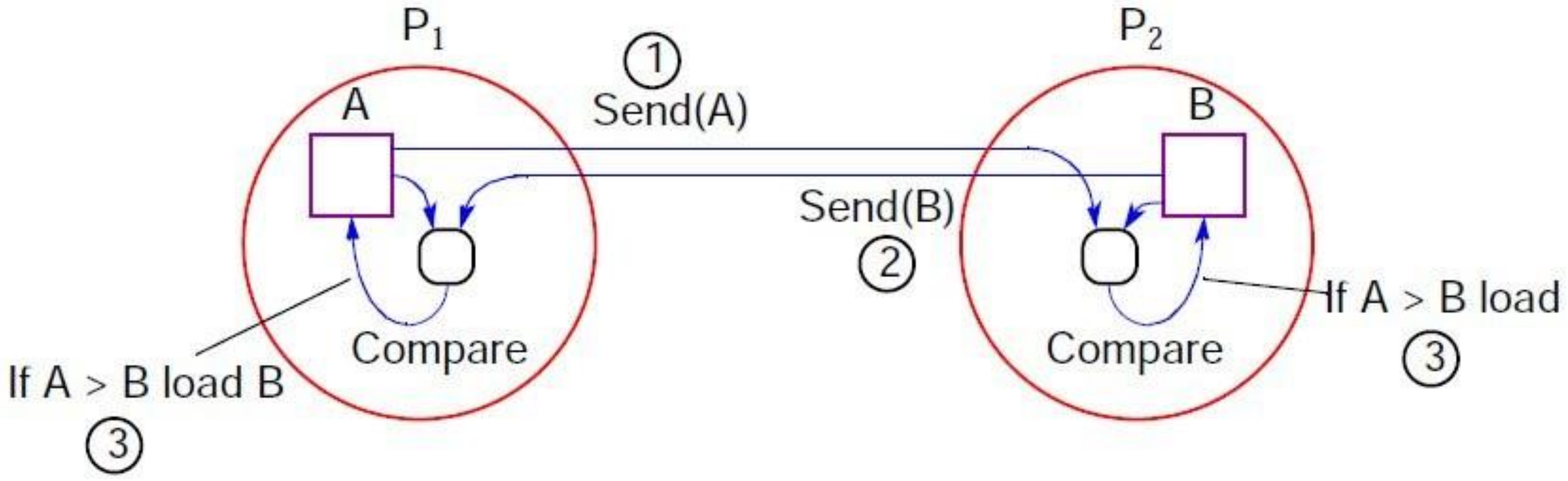


© Sun

Compare-and-swap with message exchange(2/2)

version 2:

P_1 sends A to P_2; P_2 sends B to P_1; P_1 does $A = min(A, B)$ and P_2 does $B = max(A, B)$.

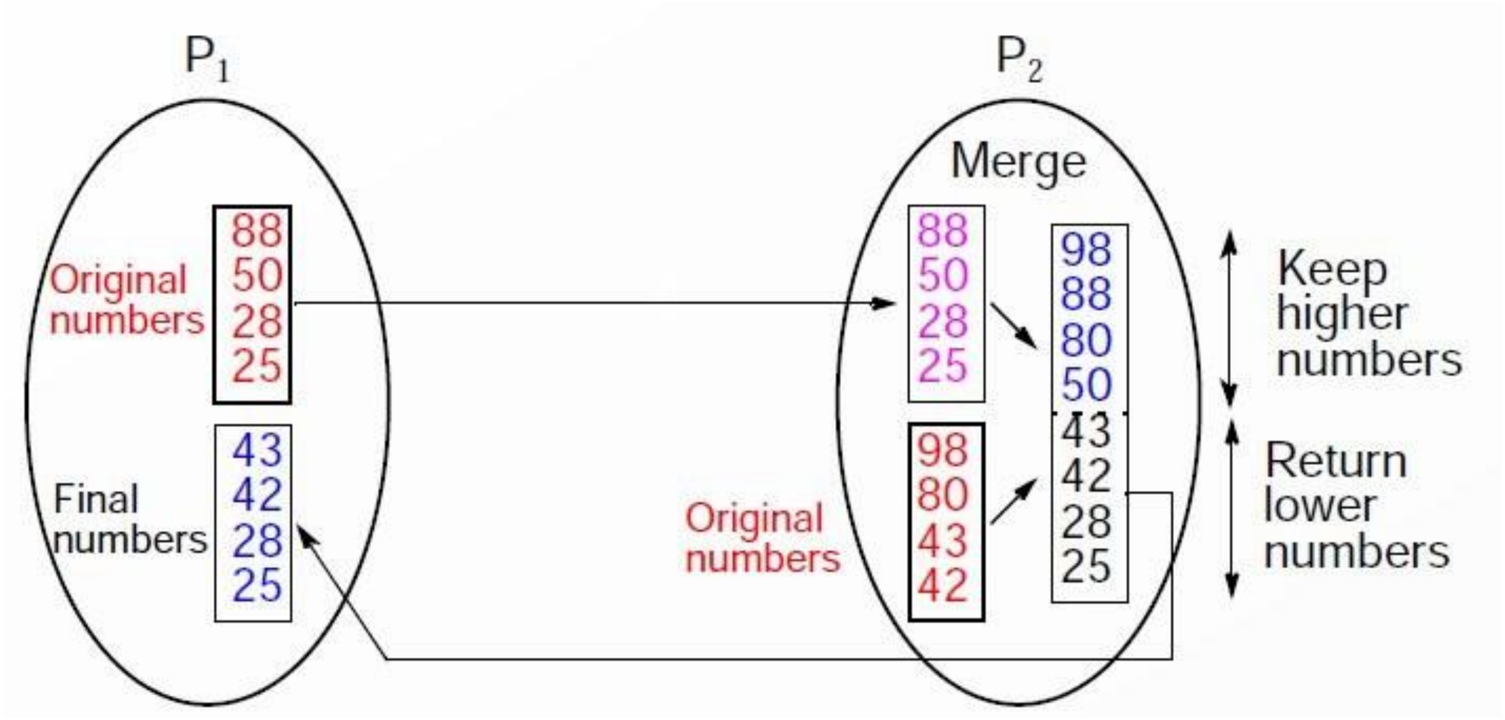


Data partition

Version 1:

n numbers and *p* processors

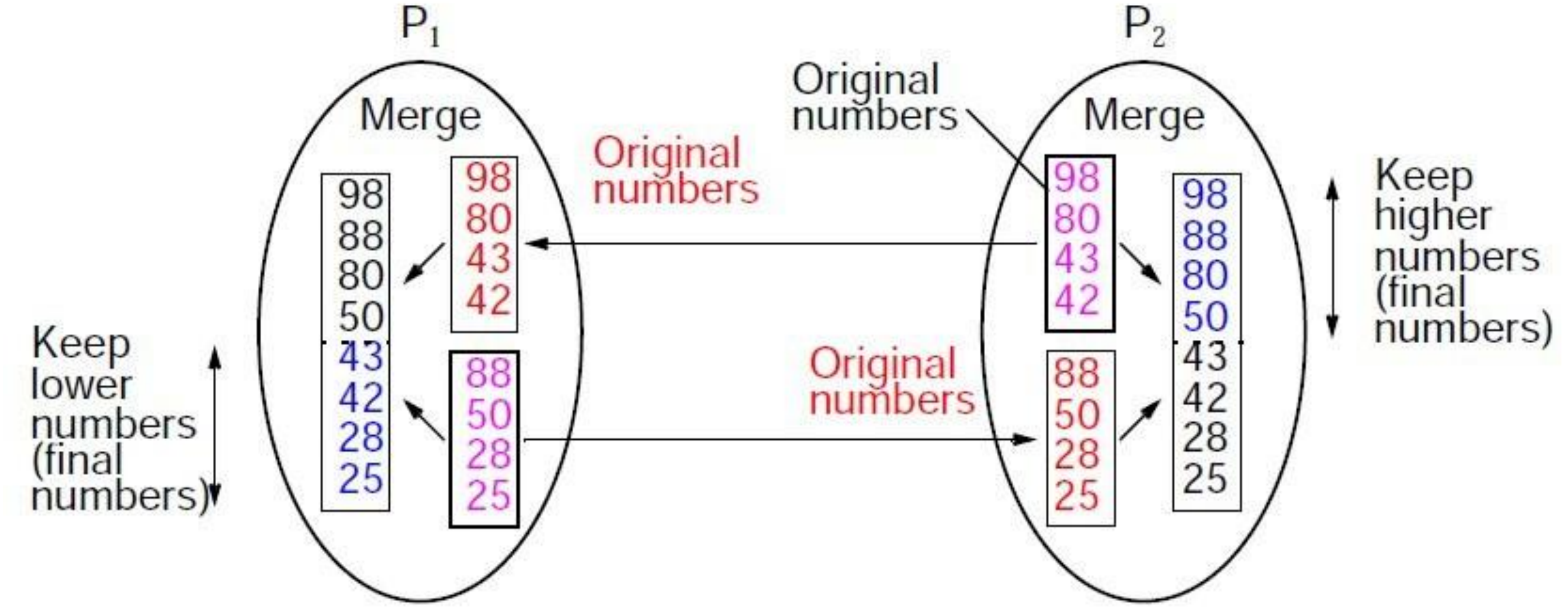
n/p numbers assigned to each processor.



© Sun

Merging two sub-lists

version 2:



Bubble Sort

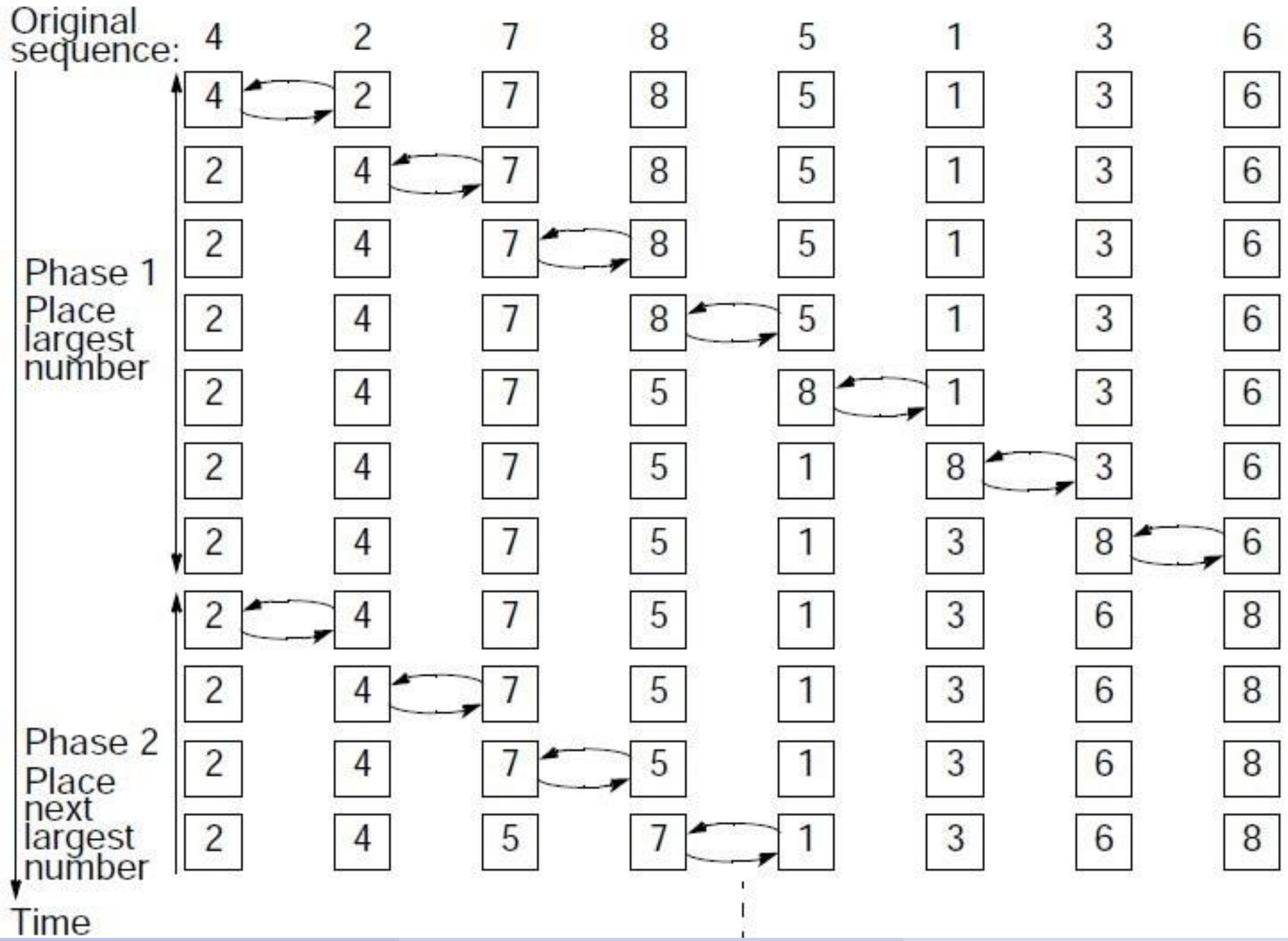
compares two consecutive values at a time and swaps them if they are out of order.

greater values are being moved towards the end of the list.

number of comparisons and swaps: $\sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}$

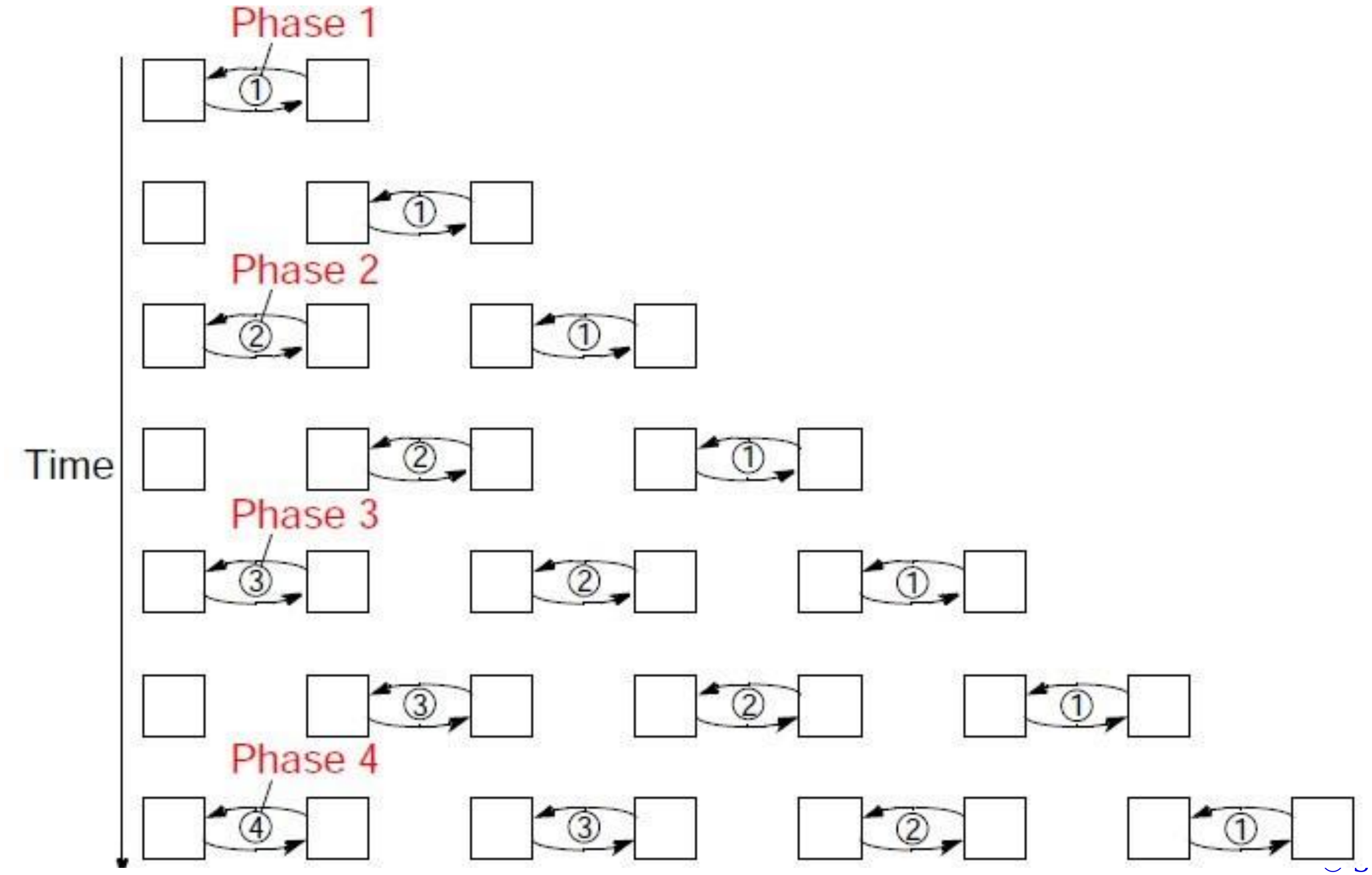
which corresponds to a time complexity $O(n^2)$.

Bubble-sort example



Parallel Bubble-sort

The idea is to run multiple iterations in parallel.



Odd-Even with transposition (1/2)

it is a variant of the bubble-sort operates in two

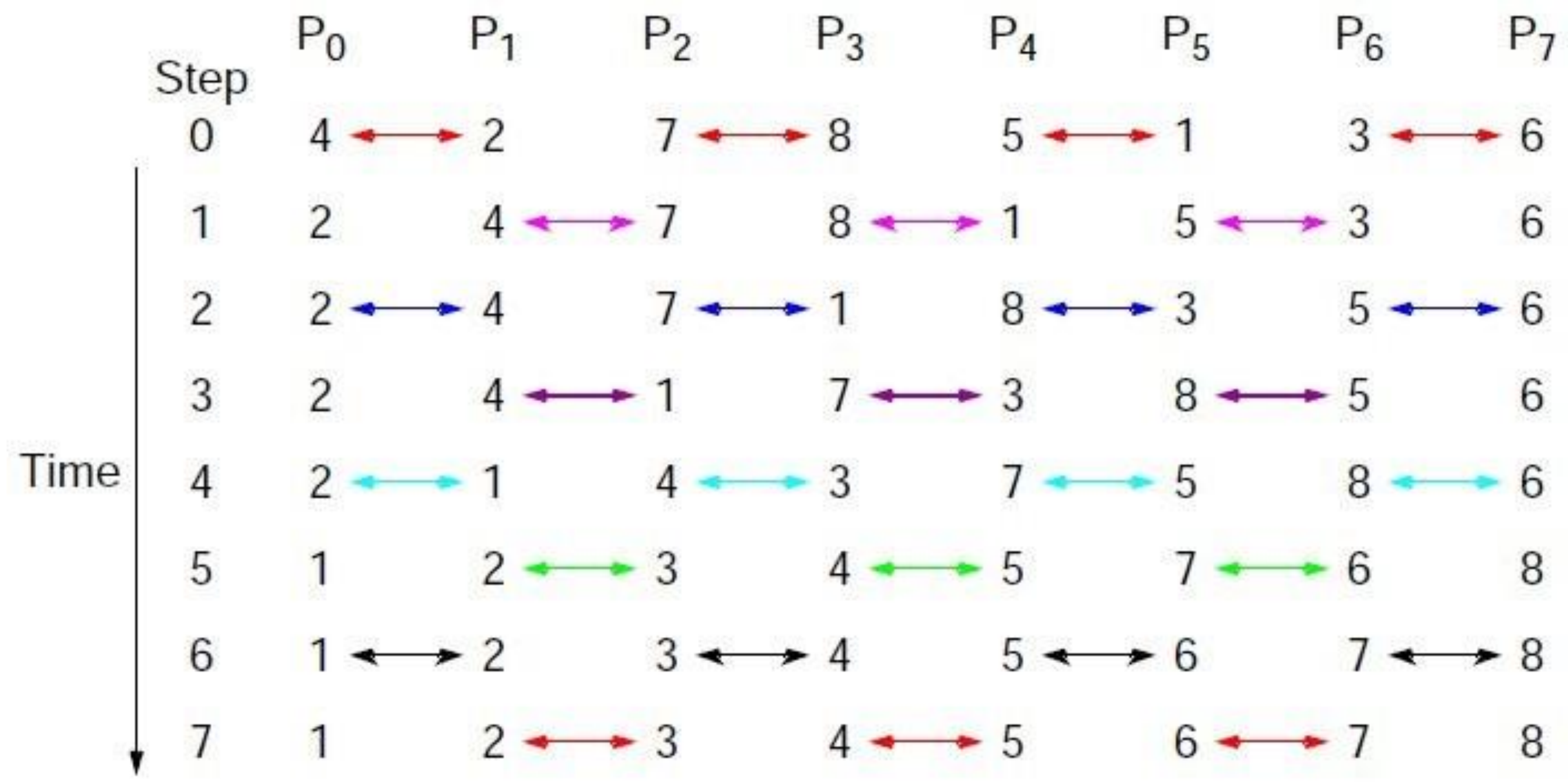
alternate phases: **Phase-even:**

- even processes exchange values with right neighbors.

Phase-odd:

- odd processes exchange values with right neighbors.

Odd-Even with transposition (2/2)



Step	P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7
0	4	2	7	8	5	1	3	6
1	2	4	7	8	1	5	3	6
2	2	4	7	1	8	3	5	6
3	2	4	1	7	3	8	5	6
4	2	1	4	3	7	5	8	6
5	1	2	3	4	5	7	6	8
6	1	2	3	4	5	6	7	8
7	1	2	3	4	5	6	7	8

© Sun

Parallel algorithm - Odd-Even with transposition

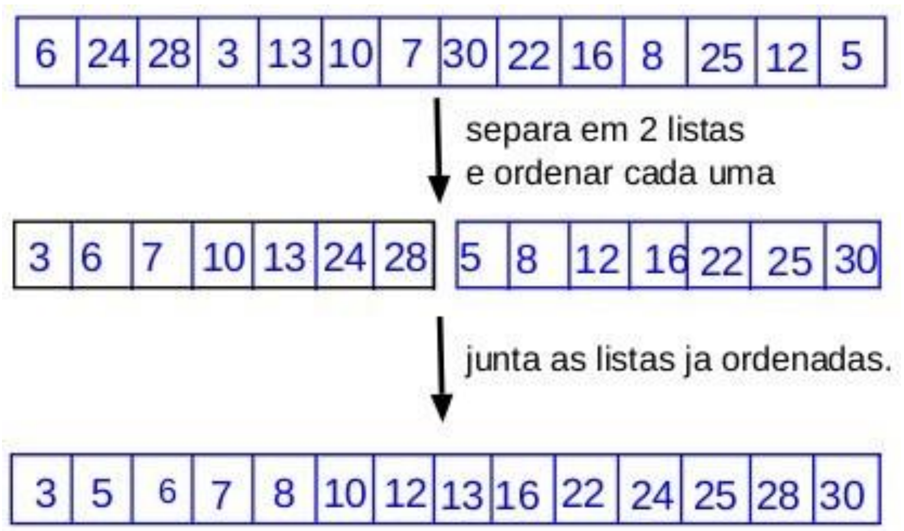
```
void ODD-EVEN-PAR(n)
{
id =  process  label
for (i= 1; i<= n; i++) {  if (i is odd)
compare-and-exchange-min(id+1);  else
compare-and-exchange-max(id-1);  if (i is even)
compare-and-exchange-min(id+1);  else
compare-and-exchange-max(id-1);
}
}
```

Mergesort (1/2)

Example of a *divide-and-conquer* algorithm

Sorting method to sort a vector; first subdivides it in two parts, applies again the same method to each part and when they are both sorted (2 sorted vectors/lists) with *m* and *n* elements, they are merged to produce a sorted vector that contains *m* + *n* elements of the initial vector.

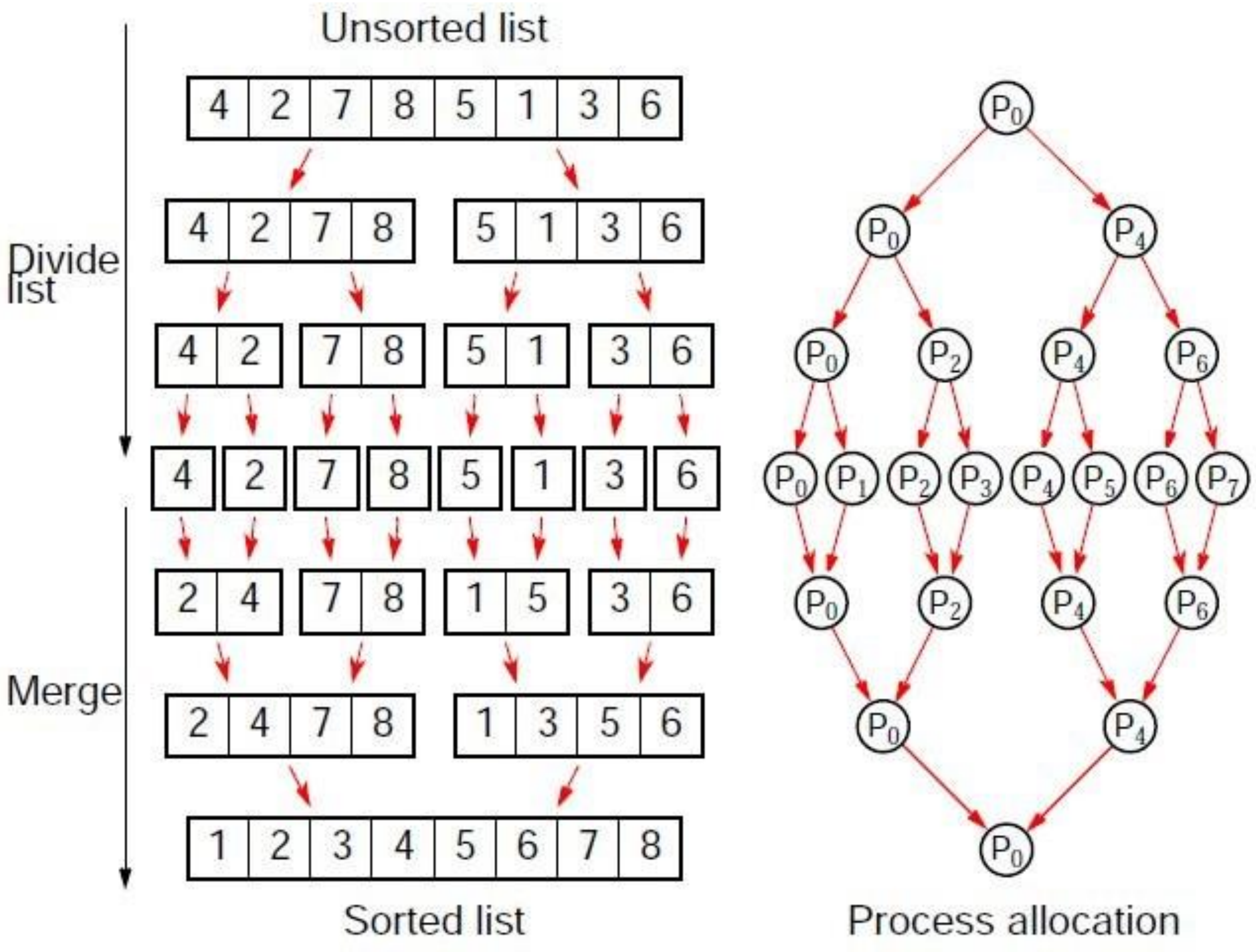
The average complexidade is $O(n \log n)$.



© Sun

Parallel Mergesort (2/2)

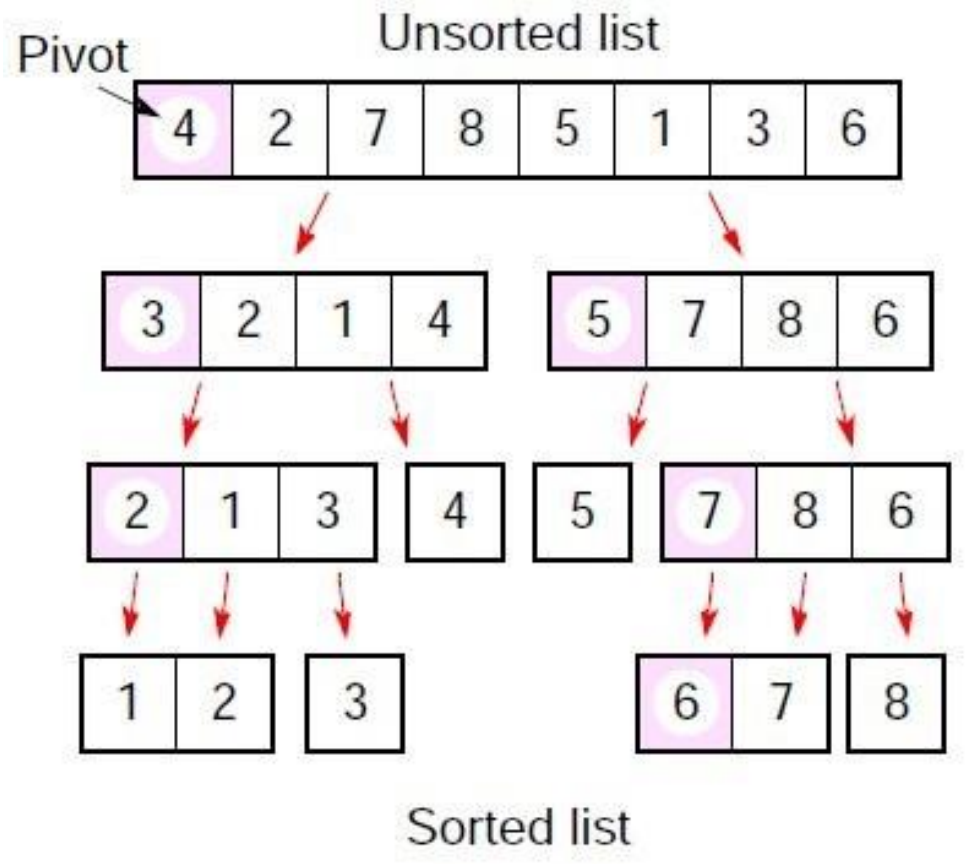
Using a strategy to assign work to processors organized in a tree.

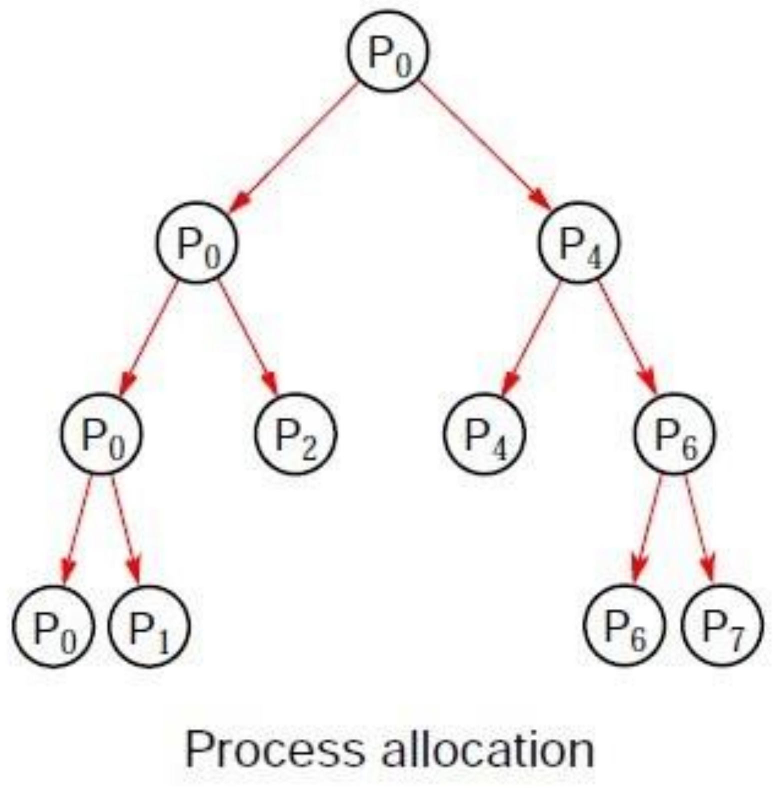


© Sun

Parallel Quicksort

Using a strategy for work-assignment in a tree-fashion.





© Sun

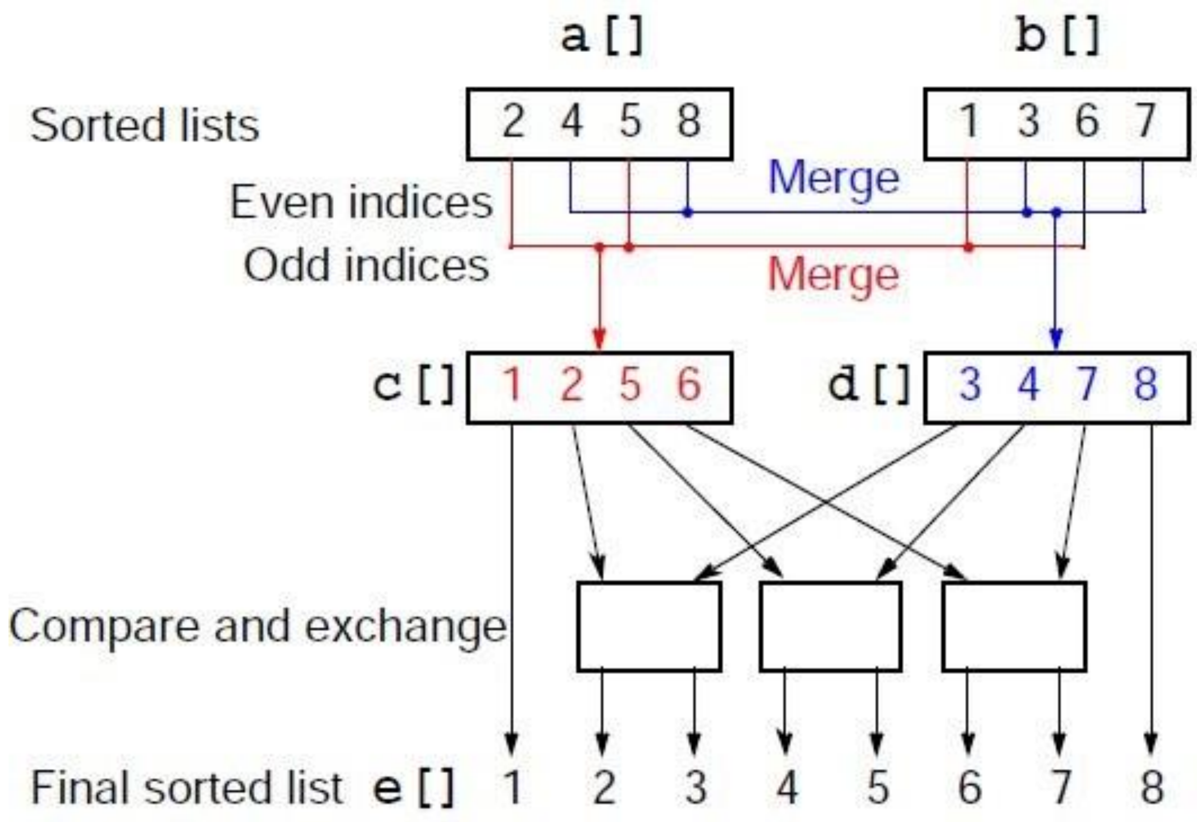
Difficulties with the allocation of processes organized in a tree

- the initial division starts with just one process, which is imitating. the search tree of quicksort is not, in general, balanced
- selecting the pivot is very important for efficiency

Odd-Even mergesort

complexity: $O(\log^2 n)$

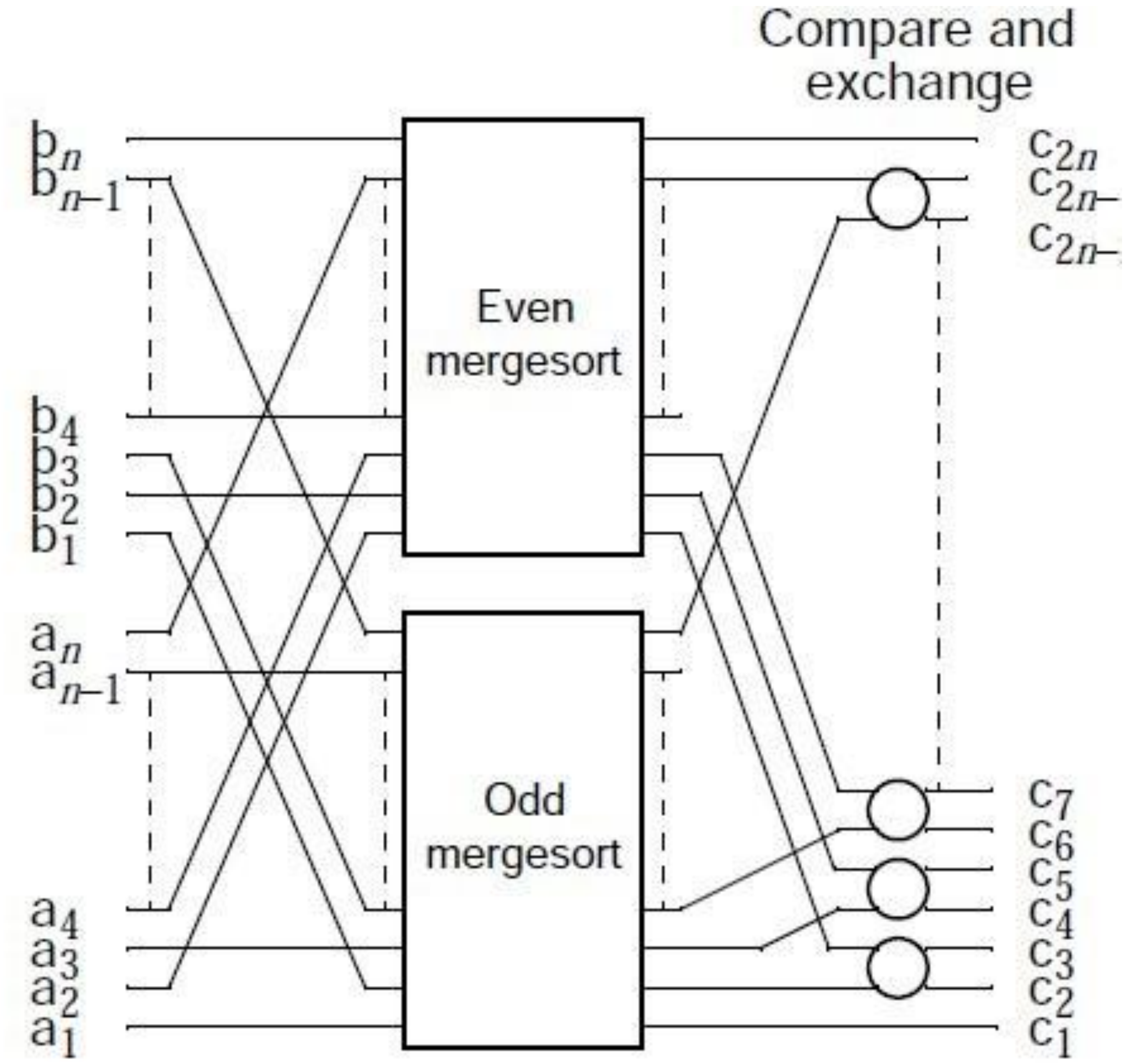
merging the two lists $a_1, a_2, \ldots, a_n$ and $b_1, b_2, \ldots, b_n$, where n is a power of 2.



© Sun

Odd-Even mergesort

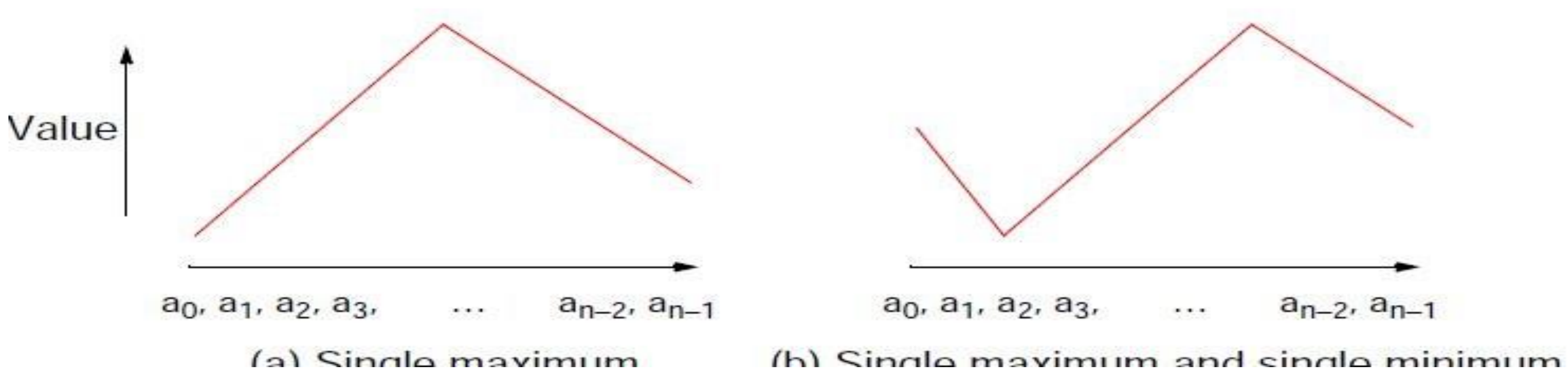
Apply recursively odd-even merge:



© Sun

Bitonic Sort (1/7)

- complexity: $O(\log^2 n)$
- a sequence is bitonic if it contains two sequences, one increasing and one decreasing, i.e.
- $a_1 < a_2 < . . . < a_{i-1} < a_i > a_{i+1} > a_{i+2} > . . . > a_n$
- for some i such that $(0 \le i \le n)$
- a sequence is bitonic if the property described is attained by a circular rotation to the right of its elements.
- Examples:

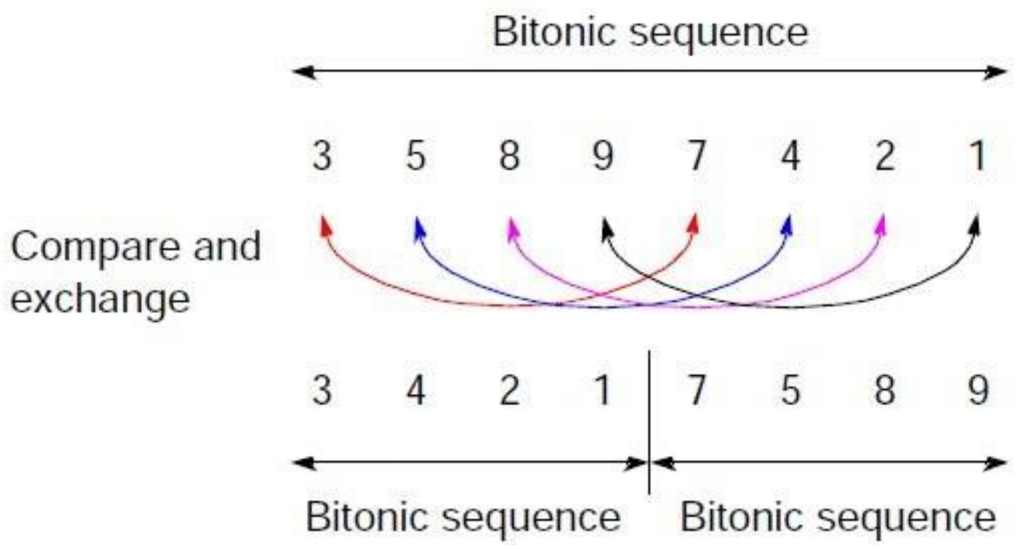

Value
a0, a1, a2, a3, ... an−2, an−1
(a) Single maximum
a0, a1, a2, a3, ... an−2, an−1
(b) Single maximum and single minimum

Bitonic Sort (2/7)

Special characteristic of bitonic sequences:

if we do a compare-and-exchange operation with elements a_i and $a_{i+n/2}$, for all i, in a sequence of size n,
we obtain *two bitonic sequences* in which all the values in one sequence are smaller then the values of the other.

Example: start with sequence 3, 5, 8, 9, 7, 4, 2, 1 and we obtain:

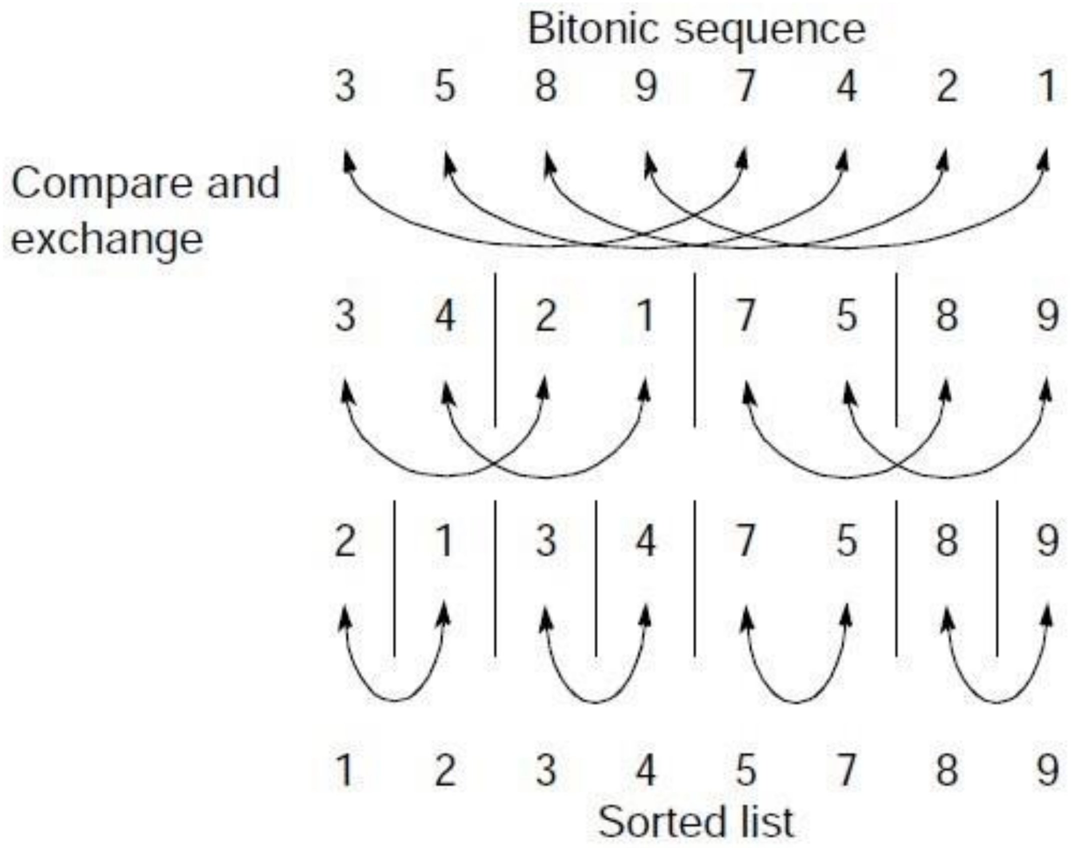


© Sun

Bitonic Sort (3/7)

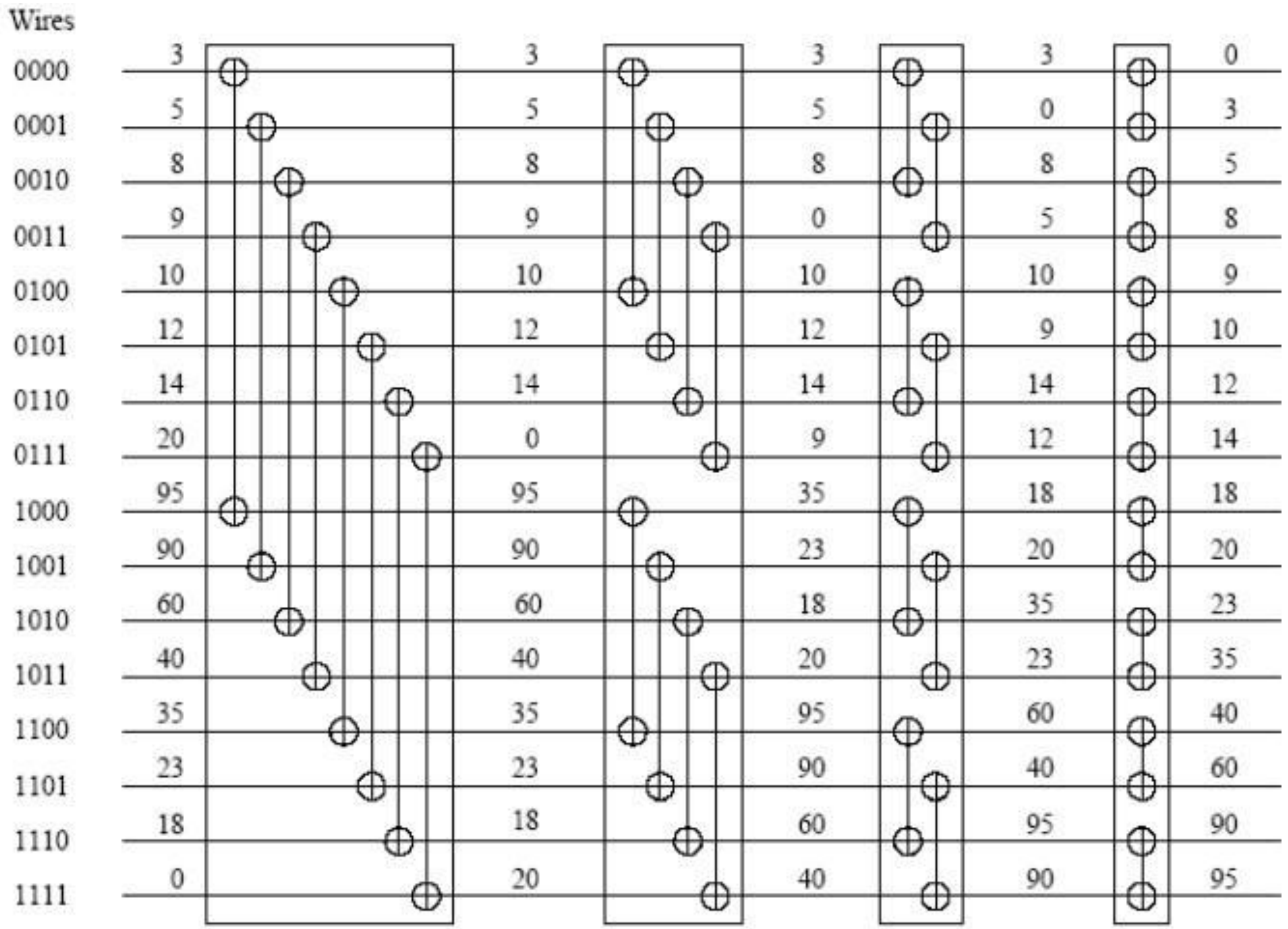
the compare-and-exchange operation moves smaller values to the left and greater values to the right.

given a bitonic sequence, if we apply recursively these operations we get a sorted sequence.



© Sun

Bitonic Sort example (4/7)



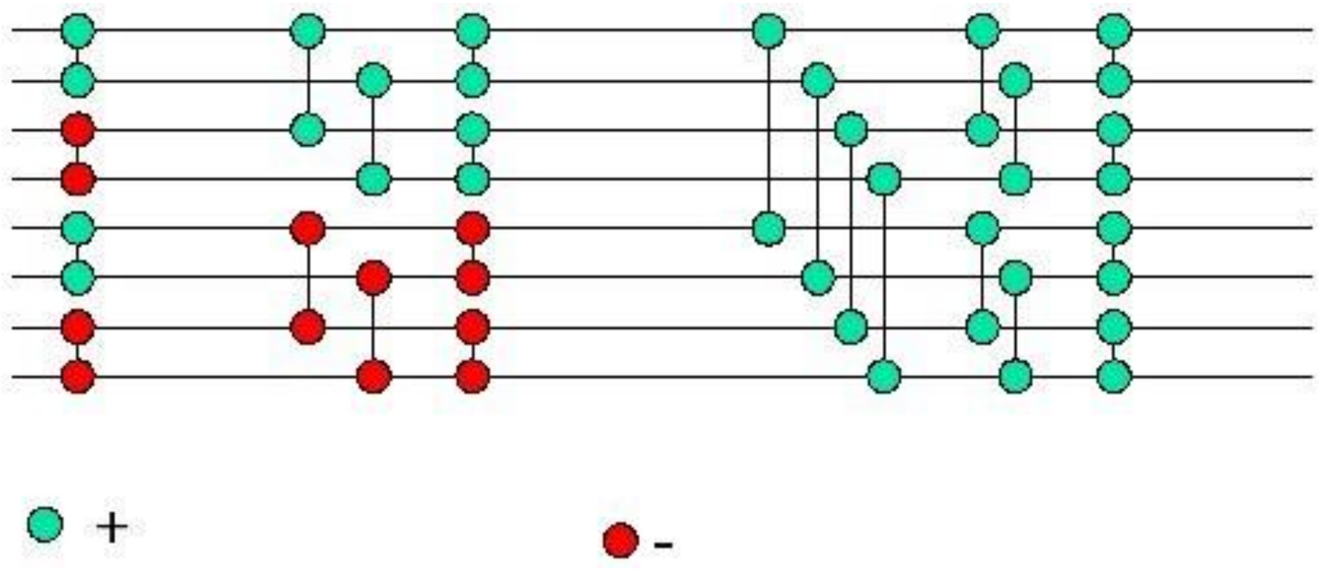
© Sun

Bitonic Sort (5/7)

To sort an unsorted sequence

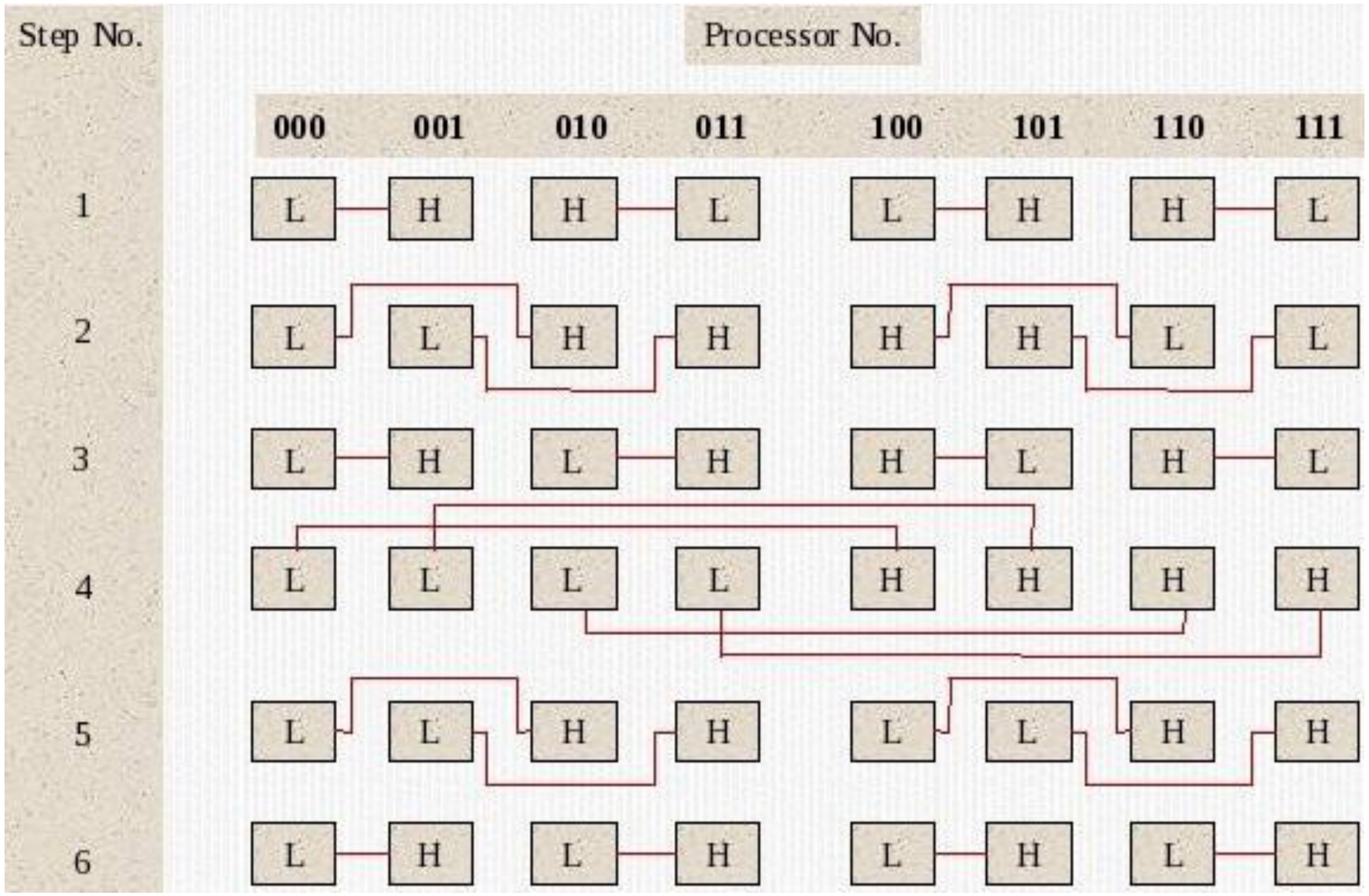
merge sequences in larger bitonic sequences, starting with adjacent pairs, alternating monotonicity.

in the end, the bitonic sequence becomes sorted in a unique increasing sequence.



© Sun

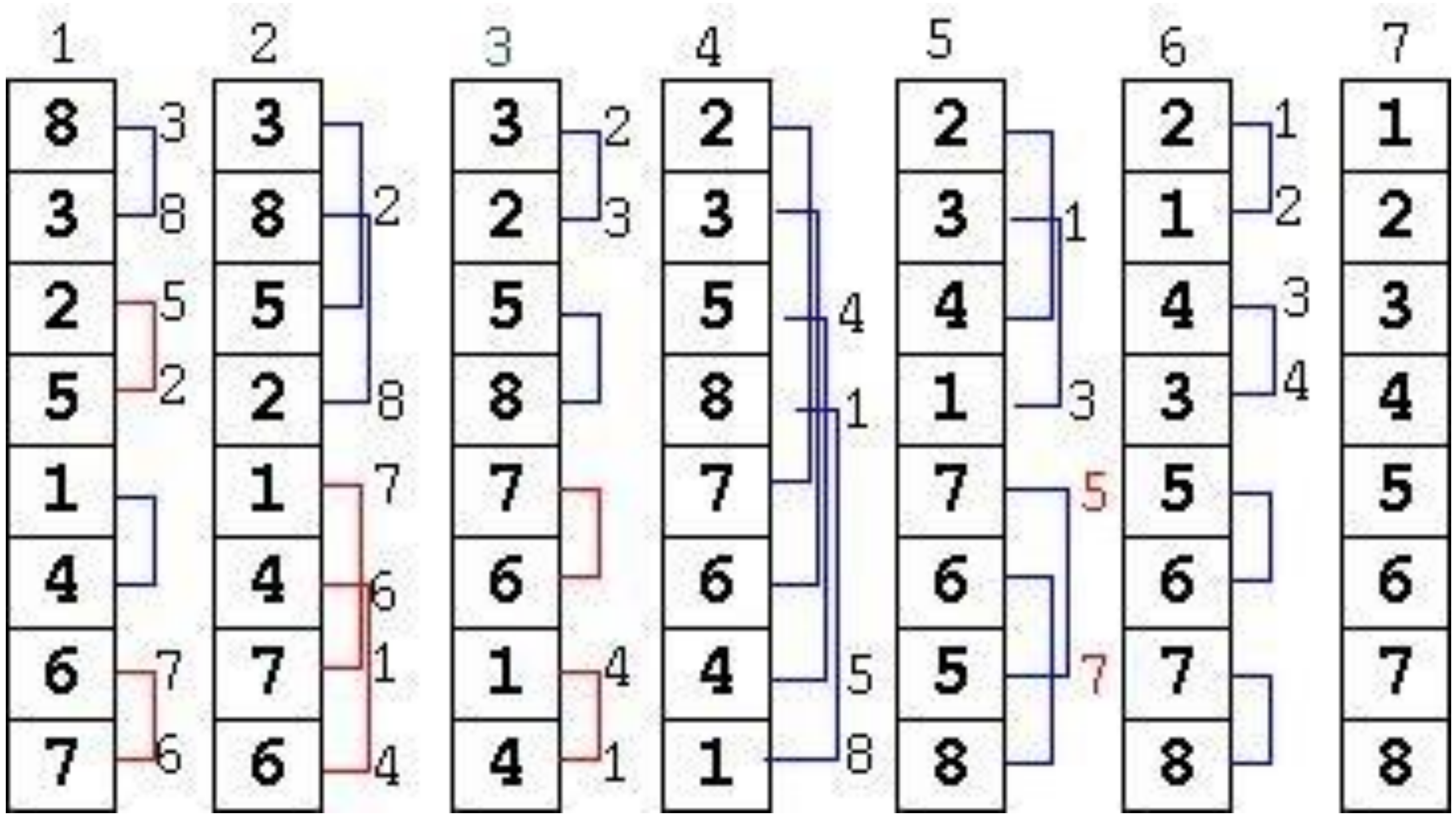
Bitonic Sort (6/7)



© Sun

Bitonic Sort (7/7)

Unsorted sequence ⇒ bitonic sequence ⇒ sorted sequence.



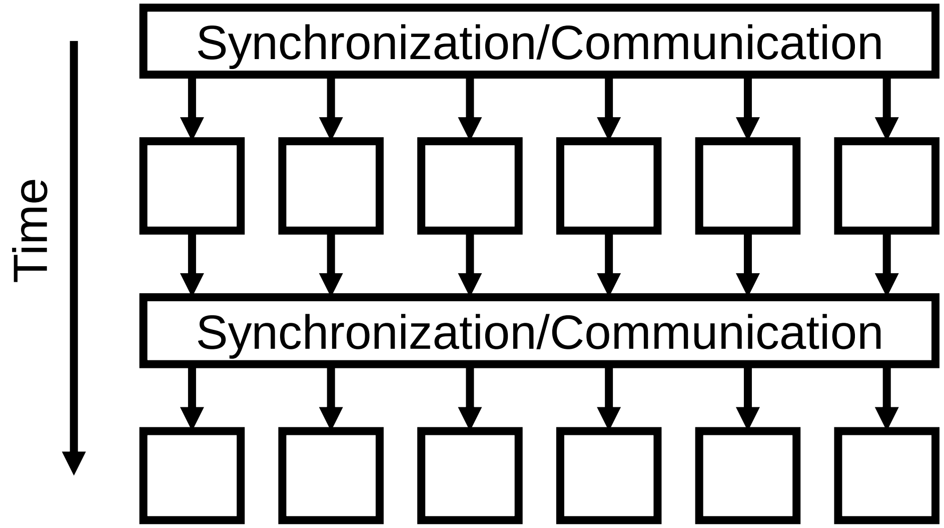
© Sun

Application Structure

- Frequently used patterns for parallel applications:
  - Single Program Multiple Data – SPMD (Domain Decomposition)
  - Embarrassingly Parallel
  - Master / Slave
  - Work Pool
  - Divide and Conquer
  - Pipeline
  - Competition

Structure: Single Program Multiple Data

- Single program is executed in a replicated fashion.
- Processes or threads execute same operations on different data.
- Loosely-synchronous: Sequence of phases of computation and communication/synchronization.



SPMD: Shared Memory Version (OpenMP)

- All processes execute the same program and have direct access to a shared address space.
- A program is a sequence of one or more sections.
- A section may be serial, parallel or replicate:
  - A **serial section** is executed by exactly one process. The first process arriving at a serial section is assigned to it; all others skip it.
  - A **parallel section** is expressed by a doall loop. All iterations can be executed in parallel, without synchronization.
  - A **replicate section** is executed by all processes.
- The model provides **barrier synchronization**.

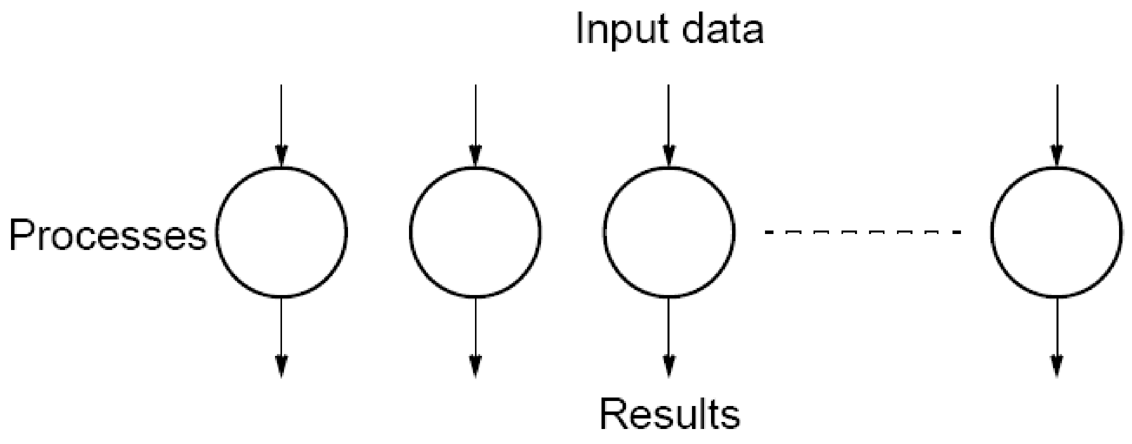
SPMD: Distributed Memory Version (MPI)

- This model assumes a distributed-memory multiprocessing (DMMP) or a cluster architecture. All processes are created when a program is initiated, and terminate at its end.
  - Each process operates in a separate address space.
  - Data are distributed to processors
  - All processes execute the same program (but on separate sets of data)

- The basic **compilation** steps :
  - Distribute data to processors
  - Enforce Owner Computes Paradigm: On each processor, perform exactly those computations which assign values to local data
  - Insert communication for all accesses to non local data.

Embarrassingly Parallel Computations

- A computation that can obviously be divided into a number of completely independent parts, each of which can be executed by a separate processor



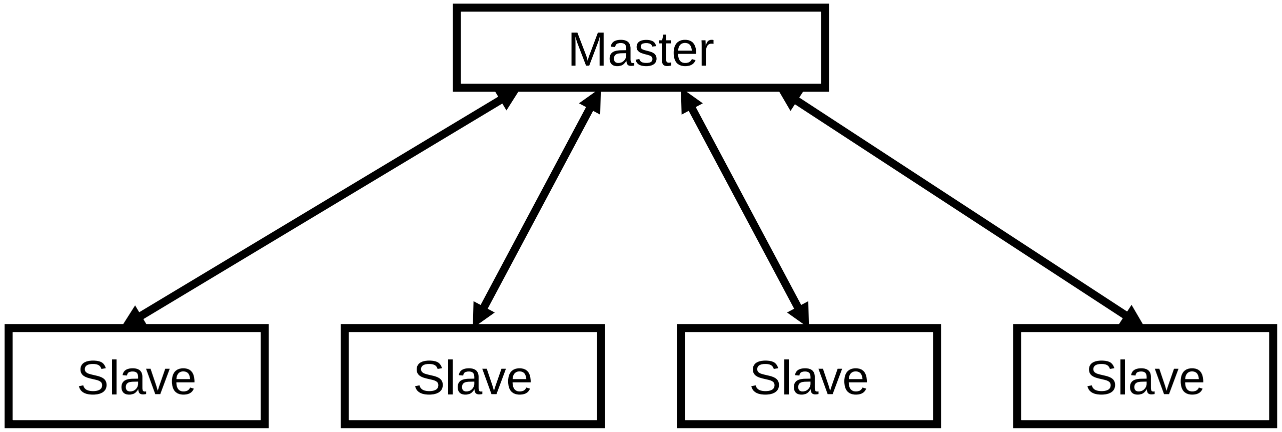
- No communication or very little communication between processes
- Each process can do its tasks w/o any interaction with others

Embarrassingly Parallel Examples

- Low level image processing
- Graphics rendering
  - Each frame may be rendered independently
- Face recognition
  - Comparing a large number of videos/photos
- Monte Carlo calculations
  - Calculation of pi
- Data analysis - cross validation
- Many more

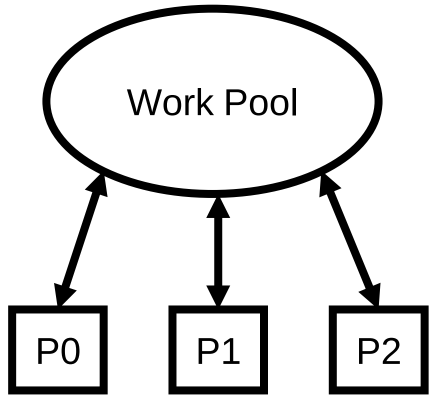
Structure: Master / Slave

- One process executes as a master. It distributes tasks to the slaves and receives the results from the slaves.
- Slaves execute the assigned tasks usually independent of the other slaves.
- Frequently used on workstation networks.



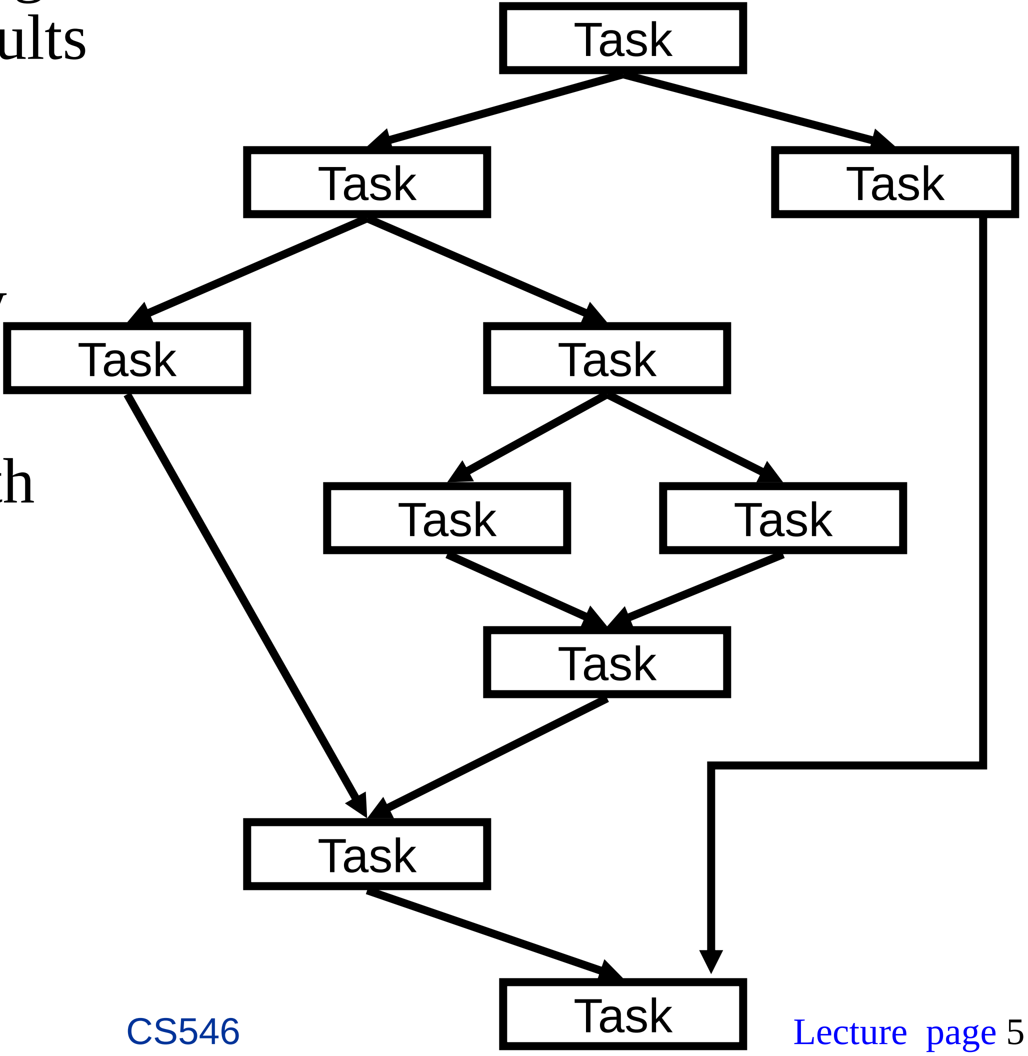
Structure: Work Pool

- Processes fetch tasks from a pool and insert new tasks into the pool.
- Pool requires synchronization.
- Large parallel machines require a distributed work pool.
- Leads to better load balancing.



Structure: Divide and Conquer

- Recursive partitioning of tasks and collection of results
- Problems:
  - o load balancing
  - o least granularity
- Used on systems with background load


Task
Task
Task
Task
Task
Task
Task
Task
Task
Task