

# What is Parallel Computing?

---

- Consider the problem of stacking (reshelving) a set of library books.
  - A single worker trying to stack all the books in their proper places cannot accomplish the task faster than a certain rate.
  - We can speed up this process, however, by employing more than one worker.

# Solution 1

---

- **Assume that books are organized into shelves** and that the shelves are grouped into bays
- **One simple way** to assign the task to the workers is:
  - **To divide the books equally among them.**
  - **Each worker stacks the books one a time**
- **This division of work may not be most efficient** way to accomplish the task since
  - **The workers must walk all over the library to stack books.**

# Solution 2

Instance of  
**task  
partitioning**

- An alternative way to divide the work is to assign a fixed and disjoint set of bays to each worker.
- As before, each worker is assigned an equal number of books arbitrarily.
  - If the worker finds a book that belongs to a bay assigned to him or her,
    - he or she places that book in its assignment spot
  - Otherwise,
    - He or she passes it on to the worker responsible for the bay it belongs to.
- The second approach requires less effort from individual workers

Instance of  
Communication  
task

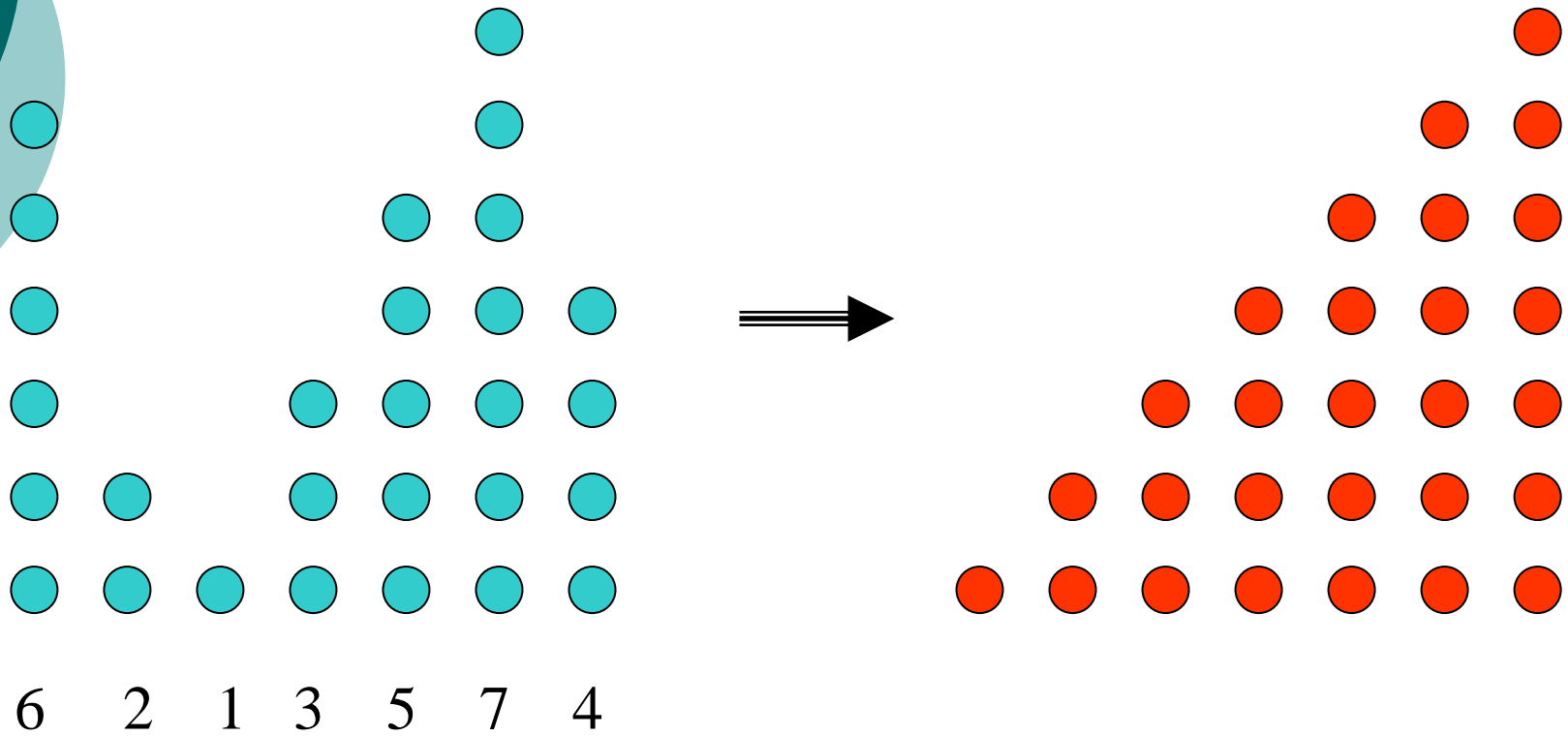
# Problems are parallelizable to different degrees

---

- **For some problems**, assigning partitions to other processors might be more time-consuming than performing the processing locally.
- **Other problems may be completely serial.**
  - For example, consider the **task of digging a post hole.**
    - Although one person can dig a hole in a certain amount of time,
    - Employing more people does not reduce this time

# Sorting in nature

---



# Parallel Processing

(Several processing elements working to solve a single problem)

---

Primary consideration: **elapsed time**

- **NOT:** throughput, sharing resources, etc.
- Downside: **complexity**
  - system, algorithm design
- Elapsed Time = **computation time + communication time + synchronization time**

# Design of efficient algorithms

---

A parallel computer is of little use unless efficient parallel algorithms are available.

---

- The issue in designing parallel algorithms are very different from those in designing their sequential counterparts.
- A significant amount of work is being done to develop efficient parallel algorithms for a variety of parallel architectures.

# The main open question

---



- The basic parallel complexity class is **NC**.
- **NC** is a class of problems computable in poly-logarithmic time ( $\log^c n$ , for a constant  $c$ ) using a polynomial number of processors.
- **P** is a class of problems computable sequentially in a polynomial time

**The main open question in parallel computations is**

$$\mathbf{NC = P ?}$$

# Efficient and optimal parallel algorithms

---

- **A parallel algorithm is efficient** iff
  - it is fast (e.g. polynomial time) and
  - the product of the parallel time and number of processors is close to the time of at the best know sequential algorithm
$$T_{\text{sequential}} \approx T_{\text{parallel}} \cdot N_{\text{processors}}$$
- **A parallel algorithms** is optimal iff this product is of the same order as the best known sequential time

# Processor Trends

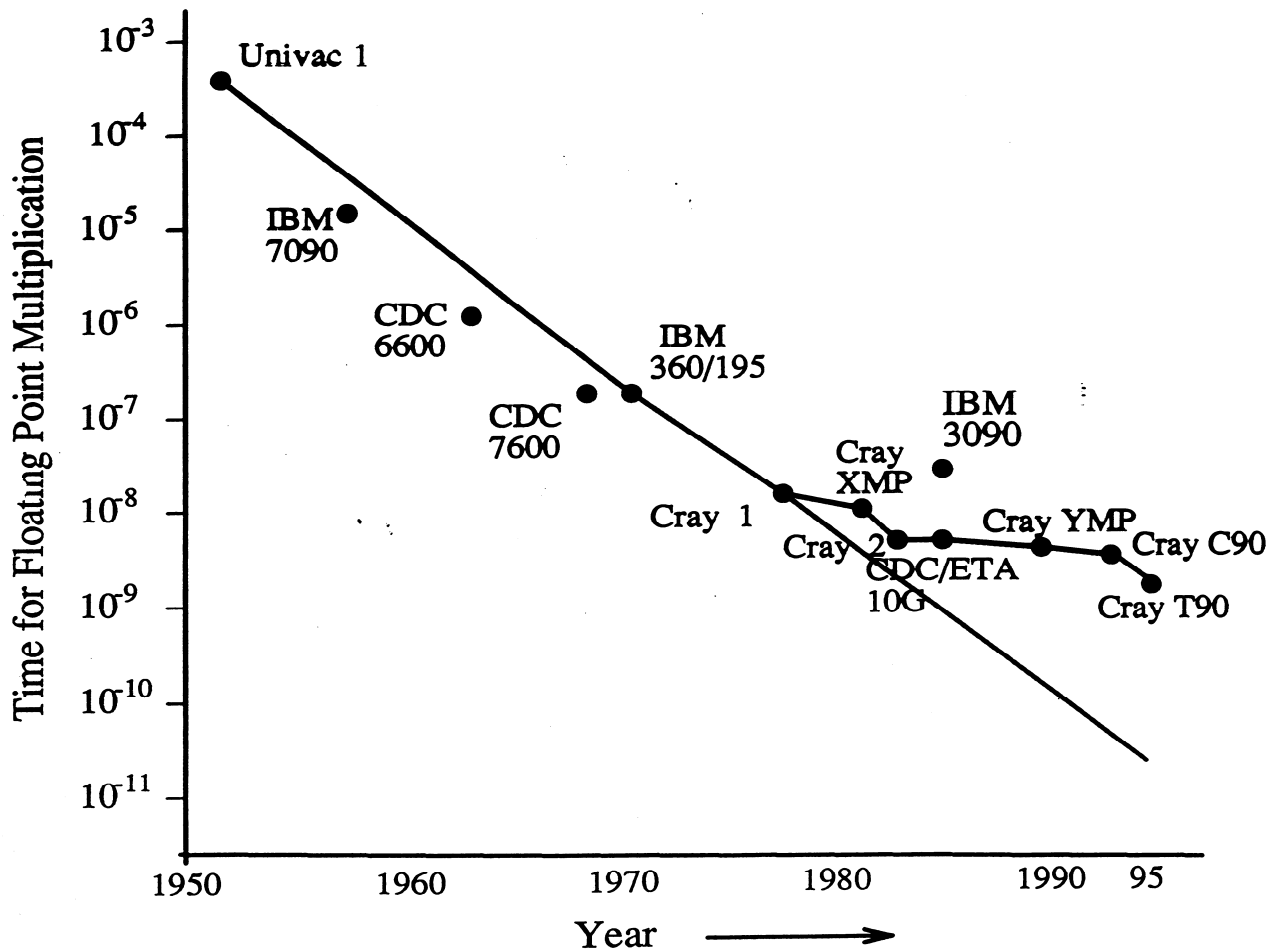
---

- **Moore's Law**

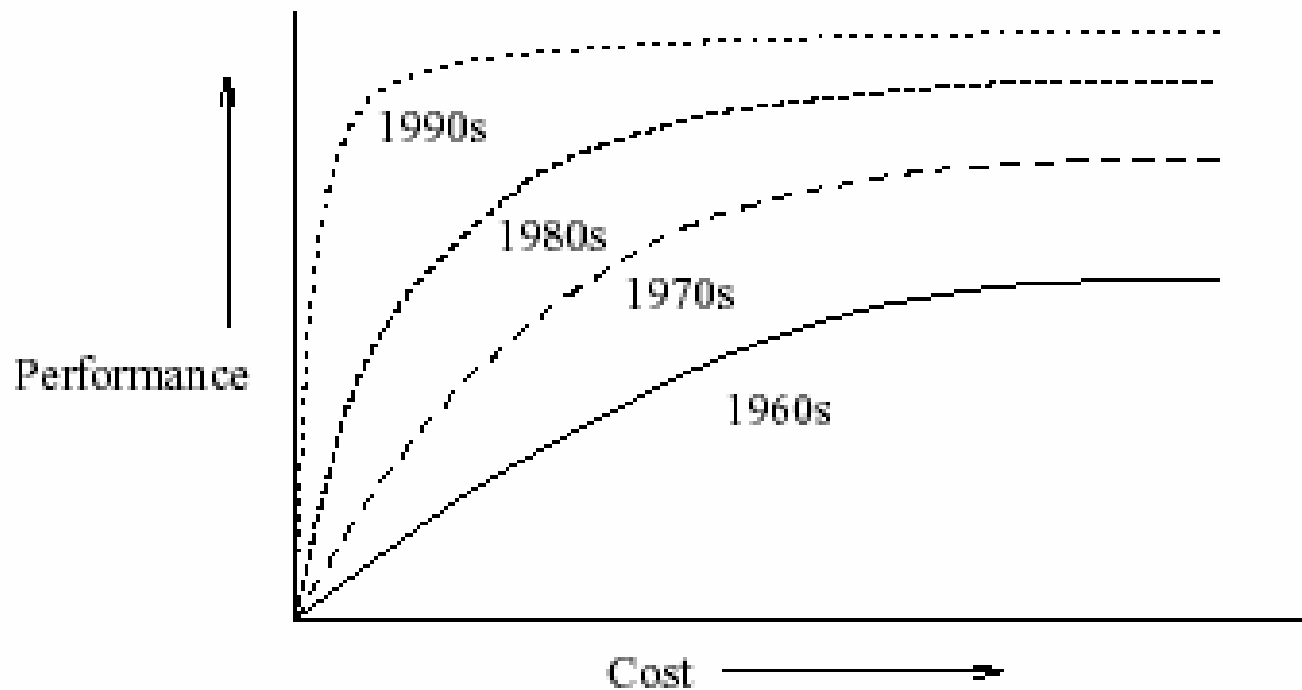
- performance doubles every 18 months

- **Parallelization within processors**

- pipelining
- multiple pipelines



**UNIPROCESSOR PERFORMANCE HISTORY**



**Figure 1.1** Cost versus performance curve and its evolution over the decades.

Copyright (r) 1994 Benjamin/Cummings Publishing Co.

# Why Parallel Computing

---

## ○ **Practical:**

- Moore's Law cannot hold forever
- Problems must be solved immediately
- Cost-effectiveness
- Scalability

## ○ **Theoretical:**

- challenging problems



# Some Complex Problems

---

- $N$ -body simulation
- Atmospheric simulation
- Image generation
- Oil exploration
- Financial processing
- Computational biology

# Some Complex Problems

---

## ○ ***N*-body simulation**

- $O(n \log n)$  time
- galaxy  $\approx 10^{11}$  stars  $\Rightarrow$  approx. one year / iteration

## ○ **Atmospheric simulation**

- 3D grid, each element interacts with neighbors
- 1x1x1 mile element  $\Rightarrow 5 \times 10^8$  elements
- 10 day simulation requires approx. 100 days

# Some Complex Problems

---

## ○ **Image generation**

- animation, special effects
- several minutes of video  $\Rightarrow$  50 days of rendering

## ○ **Oil exploration**

- large amounts of seismic data to be processed
- months of sequential exploration



# Some Complex Problems

---

- **Financial processing**

- market prediction, investing
- Cornell Theory Center, Renaissance Tech.

- **Computational biology**

- drug design
- gene sequencing (Celera)
- structure prediction (Proteomics)



# Fundamental Issues

---

- Is the problem amenable to parallelization?
- How to decompose the problem to exploit parallelism?
- What machine architecture should be used?
- What parallel resources are available?
- What kind of speedup is desired?

# Two Kinds of Parallelism

---

## ○ **Pragmatic**

- goal is to speed up a given computation as much as possible
- problem-specific
- techniques include:
  - overlapping instructions (multiple pipelines)
  - overlapping I/O operations (RAID systems)
  - “traditional” (asymptotic) parallelism techniques

# Two Kinds of Parallelism

---

## ○ **Asymptotic**

- studies:
  - architectures for general parallel computation
  - parallel algorithms for fundamental problems
  - limits of parallelization
- can be subdivided into three main areas

# Asymptotic Parallelism

---

## ○ **Models**

- comparing/evaluating different architectures

## ○ **Algorithm Design**

- utilizing a given architecture to solve a given problem

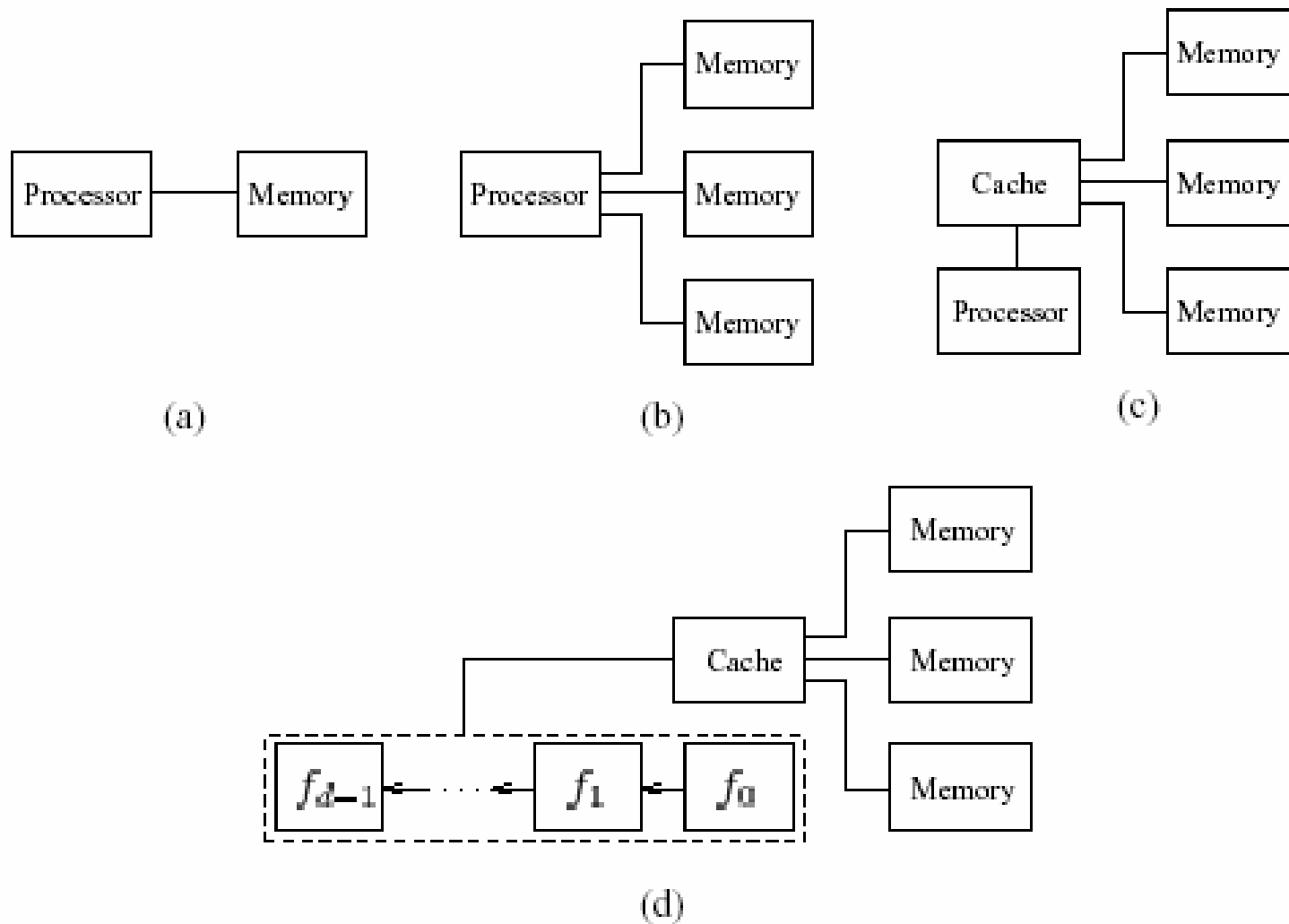
## ○ **Computational Complexity**

- classifying problems according to their difficulty

# Architecture

---

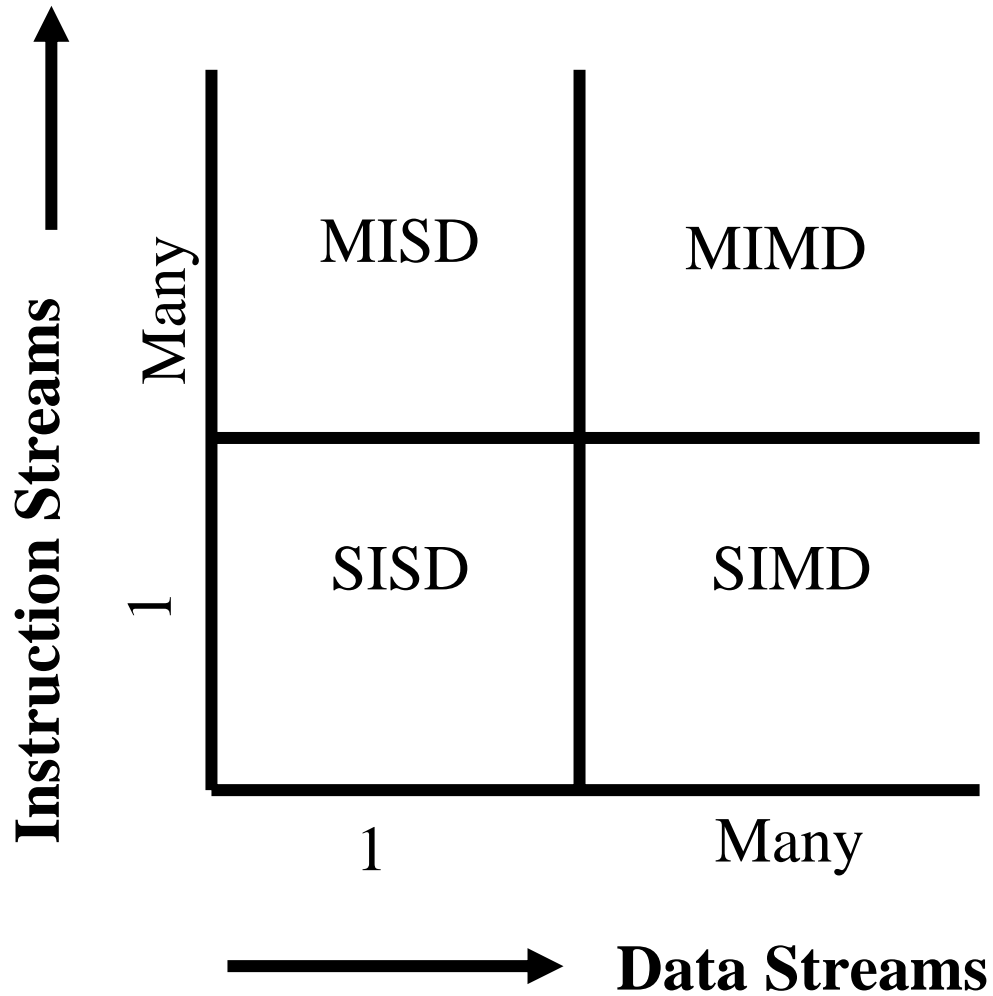
- **Single processor:**
  - single instruction stream
  - single data stream
  - von Neumann model
  
- **Multiple processors:**
  - Flynn's taxonomy



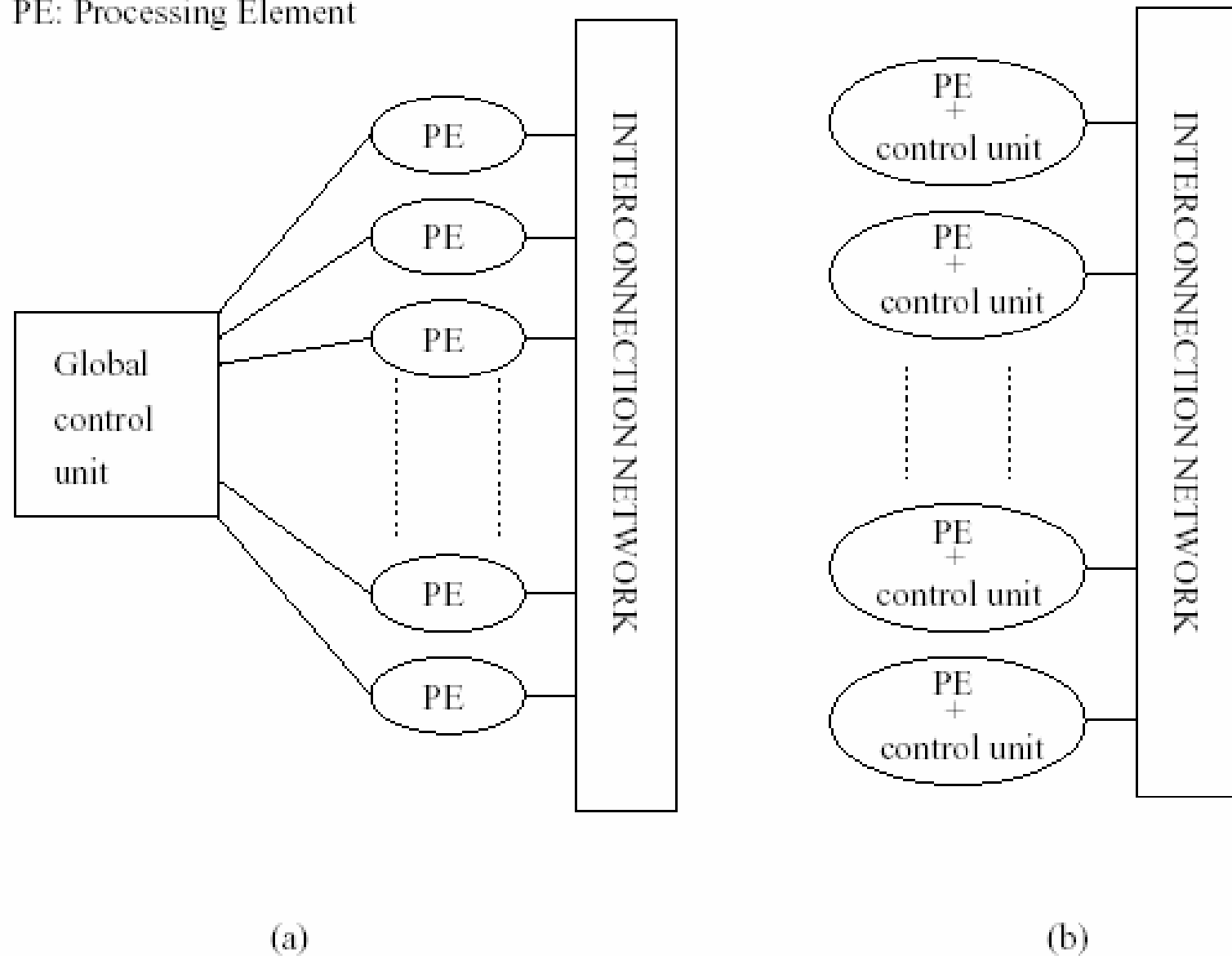
**Figure 2.1** The evolution of a typical sequential computer: (a) a simple sequential computer; (b) a sequential computer with memory interleaving; (c) a sequential computer with memory interleaving and cache; and (d) a pipelined processor with  $d$  stages.

# Flynn's Taxonomy

---



PE: Processing Element



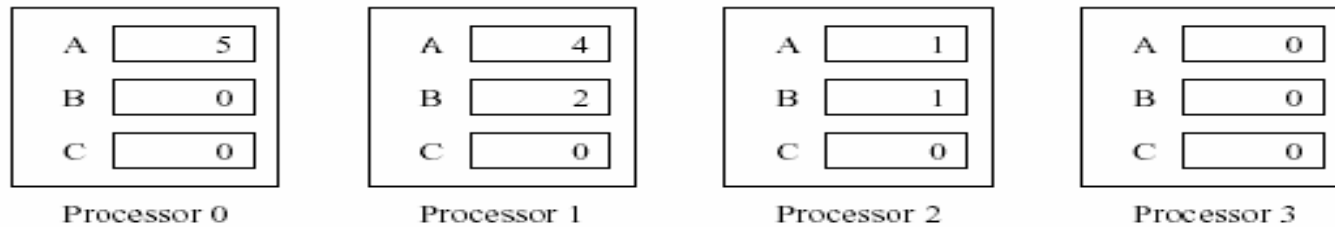
**Figure 2.2** A typical SIMD architecture (a) and a typical MIMD architecture (b).  
Copyright (r) 1994 Benjamin/Cummings Publishing Co.

```

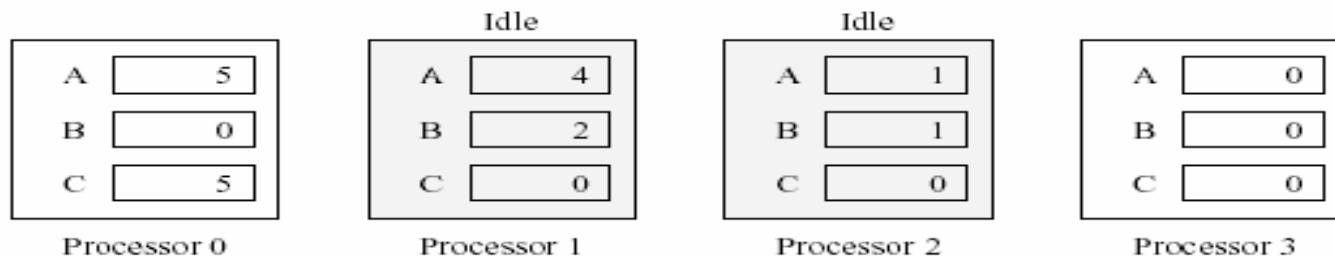
if (B == 0)
    C = A;
else
    C = A/B;

```

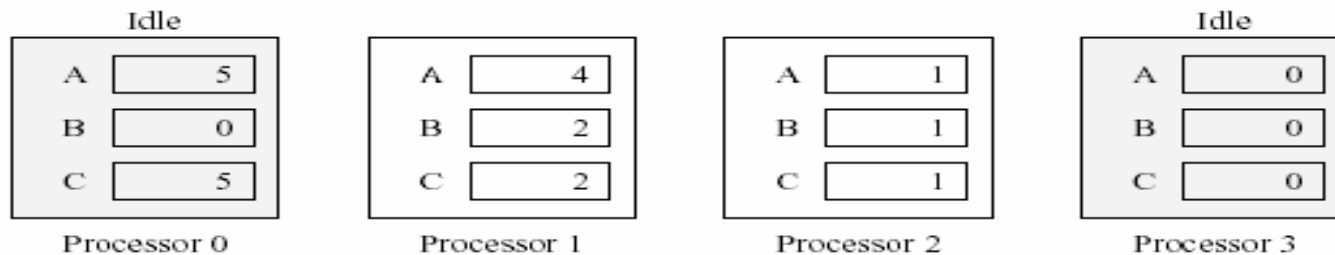
(a)



Initial values



Step 1



Step 2

(b)

**Figure 2.3** Executing a conditional statement on an SIMD computer with four processors: (a) The conditional statement; (b) The execution of the statement in two steps.



# Parallel Architectures

---

- **Multiple processing elements**
- **Memory:**
  - shared
  - distributed
  - hybrid
- **Control:**
  - centralized
  - distributed

# Parallel vs Distributed Computing

---

- **Parallel:**

- several processing elements concurrently solving a single same problem

- **Distributed:**

- processing elements do not share memory or system clock

- **Which is the subset of which?**

- distributed is a subset of parallel

# Parallelization

---

- **Control vs Data parallel:**

- control: different operations on different data elements
- data: same operations on different data elements

- **Coarse vs Fine grained:**

- algorithm granularity: ratio of computation to communication time
- architecture granularity: ratio of computation to communication cost

# An Idealized Parallel Computer

---

- PRAM
  - EREW
  - CREW
  - ERCW
  - CRCW