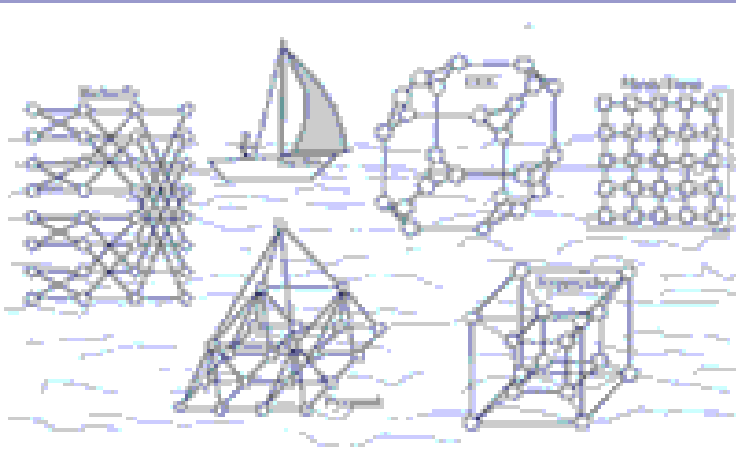


Plenum Series in Computer Science

# Introduction to Parallel Processing

Algorithms and Architectures



Behrooz Parhami

## Part I Fundamental Concepts

Architectural Variations	Part I: Fundamental Concepts	Background and Motivation Complexity and Models	1. Introduction to Parallelism 2. A Taste of Parallel Algorithms 3. Parallel Algorithm Complexity 4. Models of Parallel Processing
	Part II: Extreme Models	Abstract View of Shared Memory Circuit Model of Parallel Systems	5. PRAM and Basic Algorithms 6. More Shared-Memory Algorithms 7. Sorting and Selection Networks 8. Other Circuit-Level Examples
	Part III: Mesh-Based Architectures	Data Movement on 2D Arrays Mesh Algorithms and Variants	9. Sorting on a 2D Mesh or Torus 10. Routing on a 2D Mesh or Torus 11. Numerical 2D Mesh Algorithms 12. Other Mesh-Related Architectures
	Part IV: Low-Diameter Architectures	The Hypercube Architecture Hypercubic and Other Networks	13. Hypercubes and Their Algorithms 14. Sorting and Routing on Hypercubes 15. Other Hypercubic Architectures 16. A Sampler of Other Networks
	Part V: Some Broad Topics	Coordination and Data Access Robustness and Ease of Use	17. Emulation and Scheduling 18. Data Storage, Input, and Output 19. Reliable Parallel Processing 20. System and Software Issues
	Part VI: Implementation Aspects	Control-Parallel Systems Data Parallelism and Conclusion	21. Shared-Memory MIMD Machines 22. Message-Passing MIMD Machines 23. Data-Parallel SIMD Machines 24. Past, Present, and Future

# About This Presentation

This presentation is intended to support the use of the textbook *Introduction to Parallel Processing: Algorithms and Architectures* (Plenum Press, 1999, ISBN 0-306-45970-1). It was prepared by the author in connection with teaching the graduate-level course ECE 254B: Advanced Computer Architecture: Parallel Processing, at the University of California, Santa Barbara. Instructors can use these slides in classroom teaching and for other educational purposes. Any other use is strictly prohibited. © Behrooz Parhami

<b>Edition</b>	<b>Released</b>	<b>Revised</b>	<b>Revised</b>
<b>First</b>	Spring 2005	Spring 2006	Fall 2008

# I Fundamental Concepts

Provide motivation, paint the big picture, introduce the 3 Ts:

- Taxonomy (basic terminology and models)
- Tools for evaluation or comparison
- Theory to delineate easy and hard problems

## Topics in This Part

Chapter 1 Introduction to Parallelism

Chapter 2 A Taste of Parallel Algorithms

Chapter 3 Parallel Algorithm Complexity

Chapter 4 Models of Parallel Processing

# 1 Introduction to Parallelism

Set the stage for presenting the course material, including:

- Challenges in designing and using parallel systems
- Metrics to evaluate the effectiveness of parallelism

## Topics in This Chapter

1.1 Why Parallel Processing?

1.2 A Motivating Example

1.3 Parallel Processing Ups and Downs

1.4 Types of Parallelism: A Taxonomy

1.5 Roadblocks to Parallel Processing

1.6 Effectiveness of Parallel Processing

# 1.1 Why Parallel Processing?

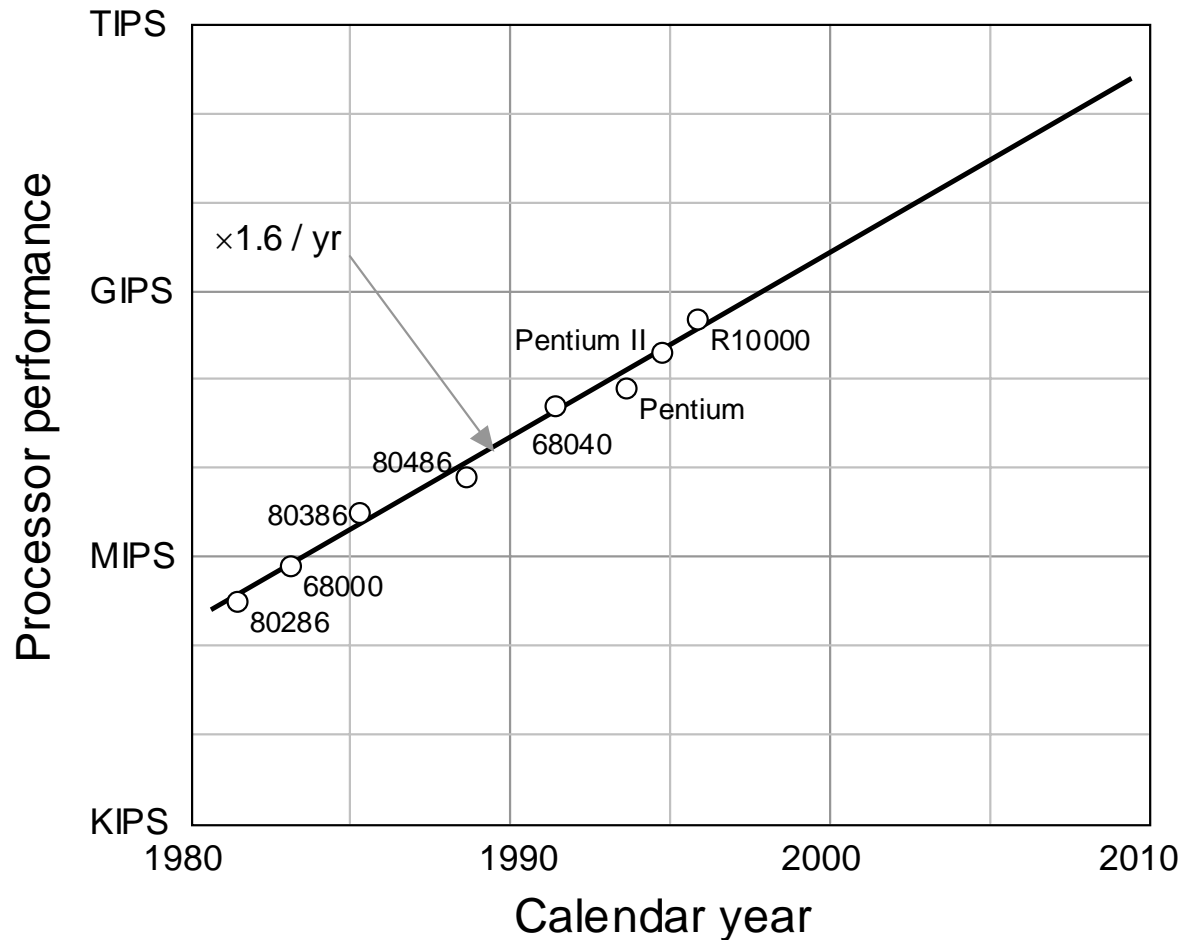
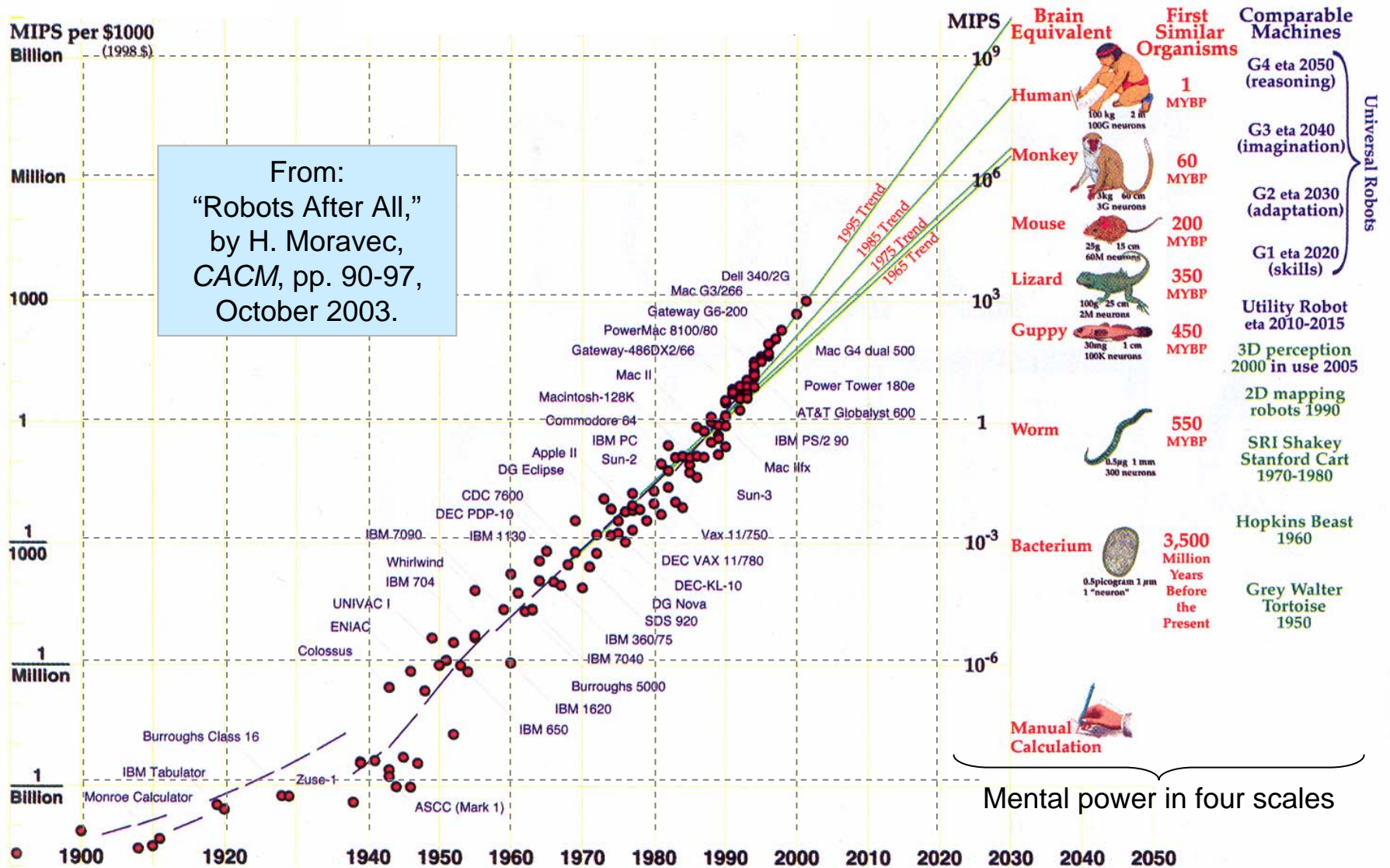


Fig. 1.1 The exponential growth of microprocessor performance, known as Moore's Law, shown over the past two decades (extrapolated).

# Evolution of Computer Performance/Cost



# The Semiconductor Technology Roadmap

Calendar year →	2001	2004	2007	2010	2013	2016
Halfpitch (nm)	140	90	65	45	32	22
Clock freq. (GHz)	2	4	7	12	20	30
Wiring levels	7	8	9	10	10	10
Power supply (V)	1.1	1.0	0.8	0.7	0.6	0.5
Max. power (W)	130	160	190	220	250	290

From the 2001 edition of the roadmap [Alla02]

Factors contributing to the validity of Moore's law

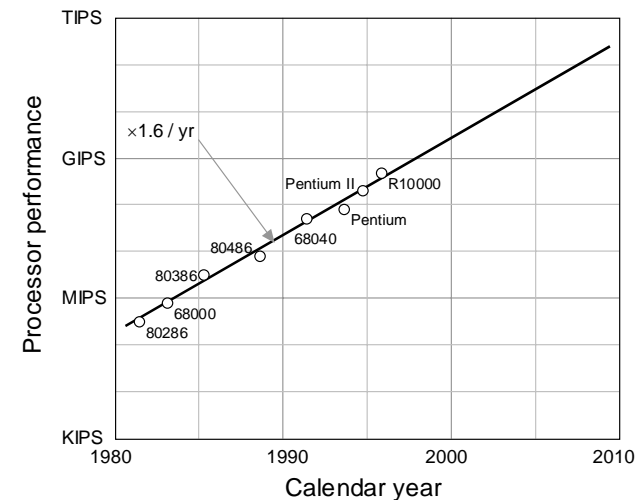
Denser circuits; Architectural improvements

Measures of processor performance

Instructions/second (MIPS, GIPS, TIPS, PIPS)

Floating-point operations per second  
(MFLOPS, GFLOPS, TFLOPS, PFLOPS)

Running time on benchmark suites





# Why High-Performance Computing?

Higher speed (solve problems faster)

Important when there are “hard” or “soft” deadlines;  
e.g., 24-hour weather forecast

Higher throughput (solve more problems)

Important when there are many similar tasks to perform;  
e.g., transaction processing

Higher computational power (solve larger problems)

e.g., weather forecast for a week rather than 24 hours,  
or with a finer mesh for greater accuracy

## Categories of supercomputers

Uniprocessor; aka vector machine

Multiprocessor; centralized or distributed shared memory

Multicomputer; communicating via message passing

Massively parallel processor (MPP; 1K or more processors)



# The Speed-of-Light Argument

The speed of light is about 30 cm/ns.

Signals travel at a fraction of speed of light (say, 1/3).

If signals must travel 1 cm during the execution of an instruction, that instruction will take at least 0.1 ns; thus, performance will be limited to 10 GIPS.

This limitation is eased by continued miniaturization, architectural methods such as cache memory, etc.; however, a fundamental limit does exist.

How does parallel processing help? Wouldn't multiple processors need to communicate via signals as well?

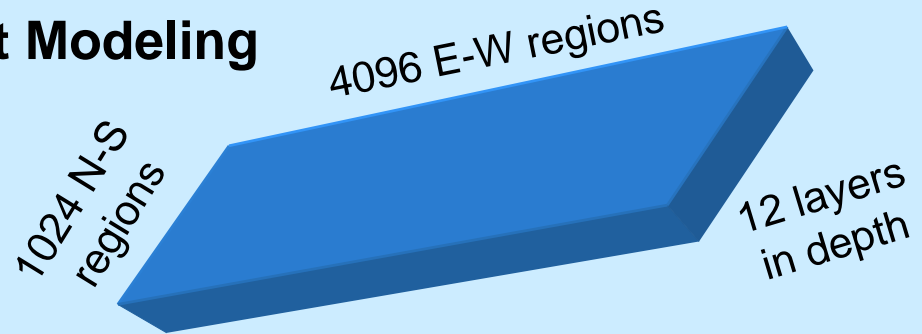
# Why Do We Need TIPS or TFLOPS Performance?

Reasonable running time = Fraction of hour to several hours ( $10^3$ - $10^4$  s)  
In this time, a TIPS/TFLOPS machine can perform  $10^{15}$ - $10^{16}$  operations

## Example 1: Southern oceans heat Modeling

(10-minute iterations)

300 GFLOP per iteration  $\times$   
300 000 iterations per 6 yrs =  
 $10^{16}$  FLOP



## Example 2: Fluid dynamics calculations ( $1000 \times 1000 \times 1000$ lattice)

$10^9$  lattice points  $\times$  1000 FLOP/point  $\times$  10 000 time steps =  $10^{16}$  FLOP

## Example 3: Monte Carlo simulation of nuclear reactor

$10^{11}$  particles to track (for 1000 escapes)  $\times$   $10^4$  FLOP/particle =  $10^{15}$  FLOP

## Decentralized supercomputing ( from *Mathworld News*, 2006/4/7 ):

Grid of tens of thousands networked computers discovers  $2^{30\,402\,457} - 1$ , the 43<sup>rd</sup> Mersenne prime, as the largest known prime (9 152 052 digits )

# The ASCI Program

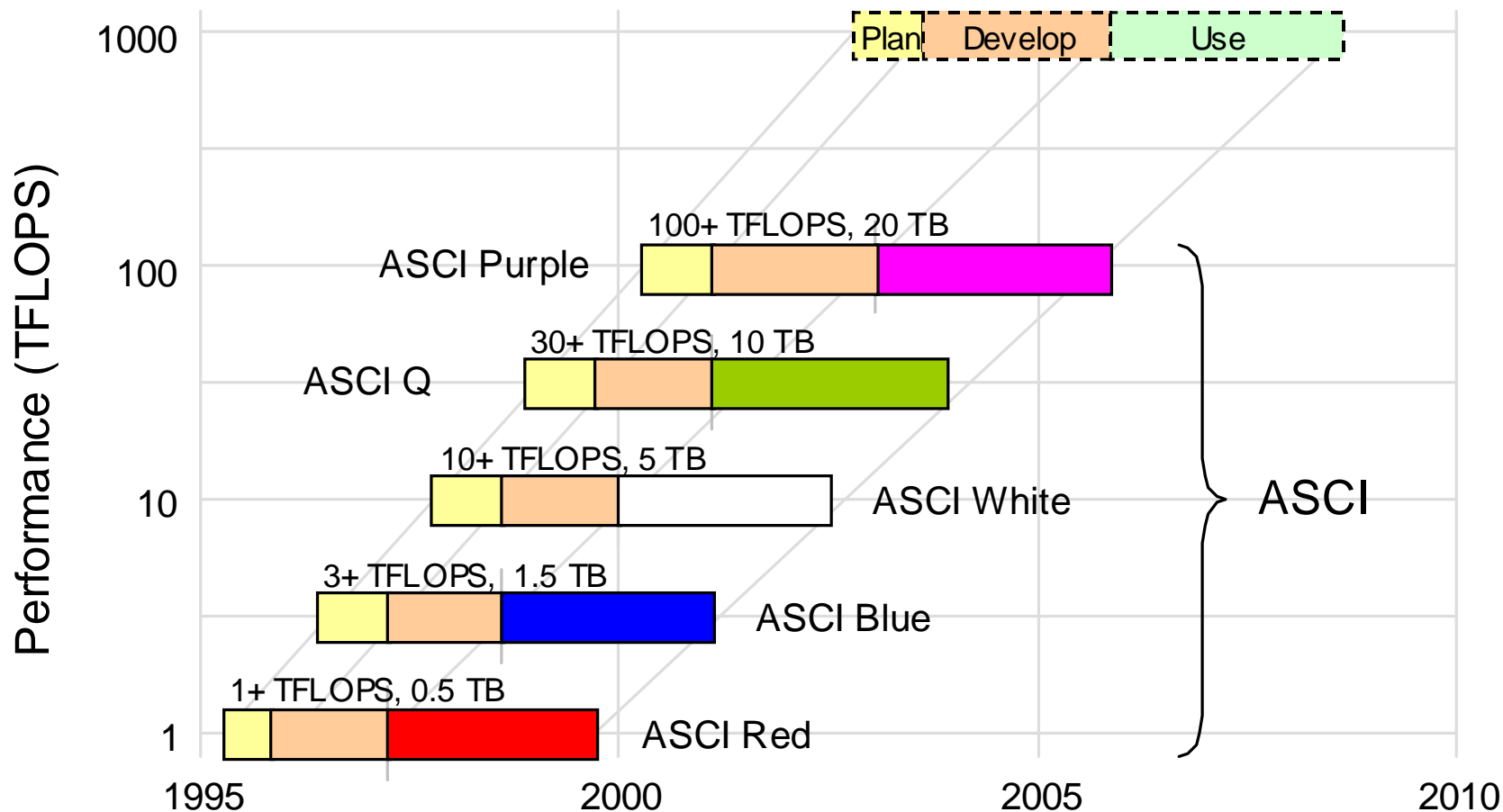


Fig. 24.1 Milestones in the Accelerated Strategic (Advanced Simulation &) Computing Initiative (ASCI) program, sponsored by the US Department of Energy, with extrapolation up to the PFLOPS level.

# The Quest for Higher Performance

Top Three Supercomputers in 2005 (*IEEE Spectrum*, Feb. 2005, pp. 15-16)

1. IBM Blue Gene/L	2. SGI Columbia	3. NEC Earth Sim
LLNL, California	NASA Ames, California	Earth Sim Ctr, Yokohama
Material science, nuclear stockpile sim	Aerospace/space sim, climate research	Atmospheric, oceanic, and earth sciences
32,768 proc's, 8 TB, 28 TB disk storage	10,240 proc's, 20 TB, 440 TB disk storage	5,120 proc's, 10 TB, 700 TB disk storage
Linux + custom OS	Linux	Unix
<b>71 TFLOPS</b> , \$100 M	<b>52 TFLOPS</b> , \$50 M	<b>36 TFLOPS*</b> , \$400 M?
Dual-proc Power-PC chips (10-15 W power)	20x Altix (512 Itanium2) linked by Infiniband	Built of custom vector microprocessors
Full system: 130k-proc, 360 TFLOPS (est)		Volume = 50x IBM, Power = 14x IBM

\* Led the top500 list for 2.5 yrs

# The Quest for Higher Performance: 2008 Update

Top Three Supercomputers in June 2008 (<http://www.top500.org>)

1. IBM Roadrunner	2. IBM Blue Gene/L	3. Sun Blade X6420
LANL, New Mexico	LLNL, California	U Texas Austin
Nuclear stockpile calculations, and more	Advanced scientific simulations	Open science research
122,400 proc's, 98 TB, 0.4 TB/s file system I/O	212,992 proc's, 74 TB, $\approx$ 2 PB disk storage	62,976 proc's, 126 TB
Red Hat Linux	CNK/SLES 9	Linux
<b>1.38 PFLOPS</b> , \$130M	<b>0.596 PFLOPS</b> , \$100M	<b>0.504 PFLOPS*</b>
PowerXCell 8i 3.2 GHz, AMD Opteron (hybrid)	PowerPC 440 700 MHz	AMD X86-64 Opteron quad core 2 GHz
2.35 MW power, expands to 1M proc's	1.60 MW power, expands to 0.5M proc's	2.00 MW power, Expands to 0.3M proc's

\* Actually 4<sup>th</sup> on top-500 list, with the 3<sup>rd</sup> being another IBM Blue Gene system at 0.557 PFLOPS

# Supercomputer Performance Growth

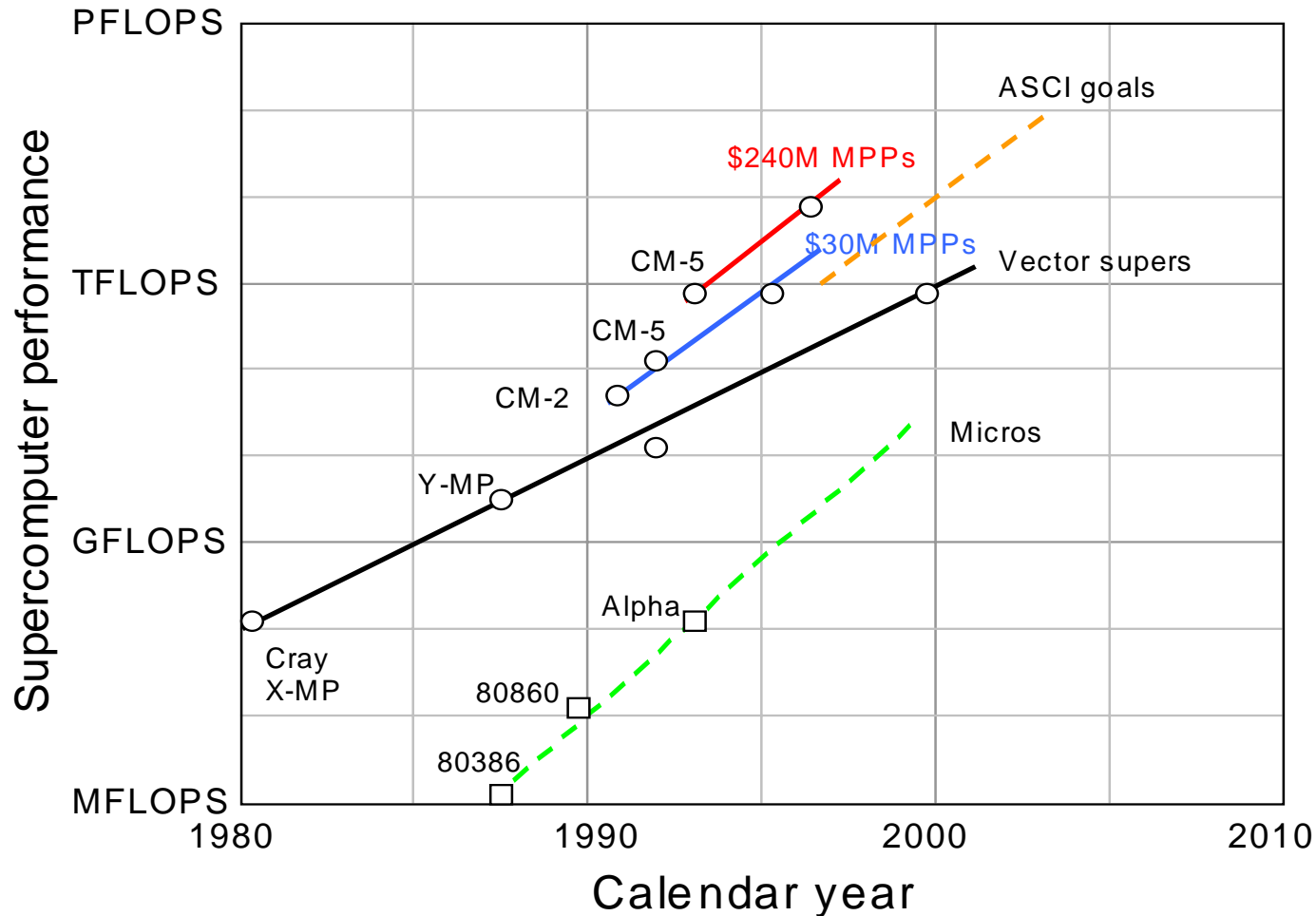


Fig. 1.2 The exponential growth in supercomputer performance over the past two decades (from [Bell92], with ASCI performance goals and microprocessor peak FLOPS superimposed as dotted lines).

# What Exactly is Parallel Processing?

Parallelism = Concurrency

Doing more than one thing at a time

Has been around for decades, since early computers

I/O channels, DMA, device controllers, multiple ALUs

The sense in which we use it in this course

Multiple agents (hardware units, software processes) collaborate to perform our main computational task

- Multiplying two matrices
- Breaking a secret code
- Deciding on the next chess move



# 1.2 A Motivating Example

Fig. 1.3 The sieve of Eratosthenes yielding a list of 10 primes for  $n = 30$ . Marked elements have been distinguished by erasure from the list.

Any composite number has a prime factor that is no greater than its square root.

<u>Init.</u>	<u>Pass 1</u>	<u>Pass 2</u>	<u>Pass 3</u>
2 ← $m$	2	2	2
3	3 ← $m$	3	3
4			
5	5	5 ← $m$	5
6			
7	7	7	7 ← $m$
8			
9	9		
10			
11	11	11	11
12			
13	13	13	13
14			
15	15		
16			
17	17	17	17
18			
19	19	19	19
20			
21	21		
22			
23	23	23	23
24			
25	25	25	
26			
27	27		
28			
29	29	29	29
30			

# Single-Processor Implementation of the Sieve

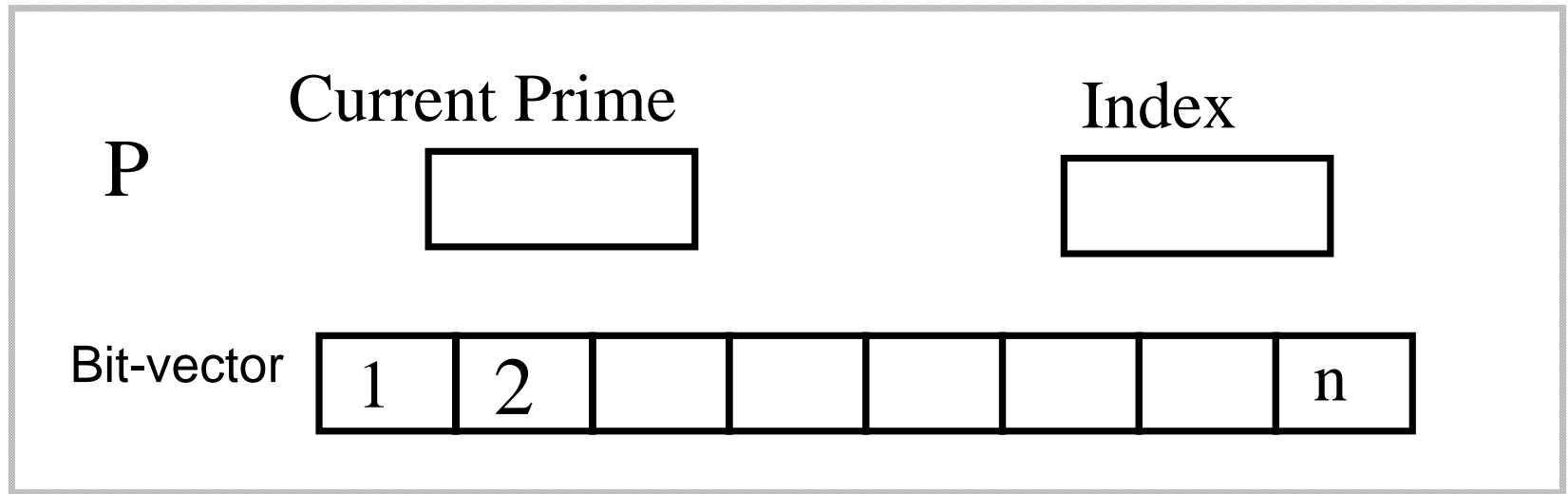


Fig. 1.4 Schematic representation of single-processor solution for the sieve of Eratosthenes.

# Control-Parallel Implementation of the Sieve

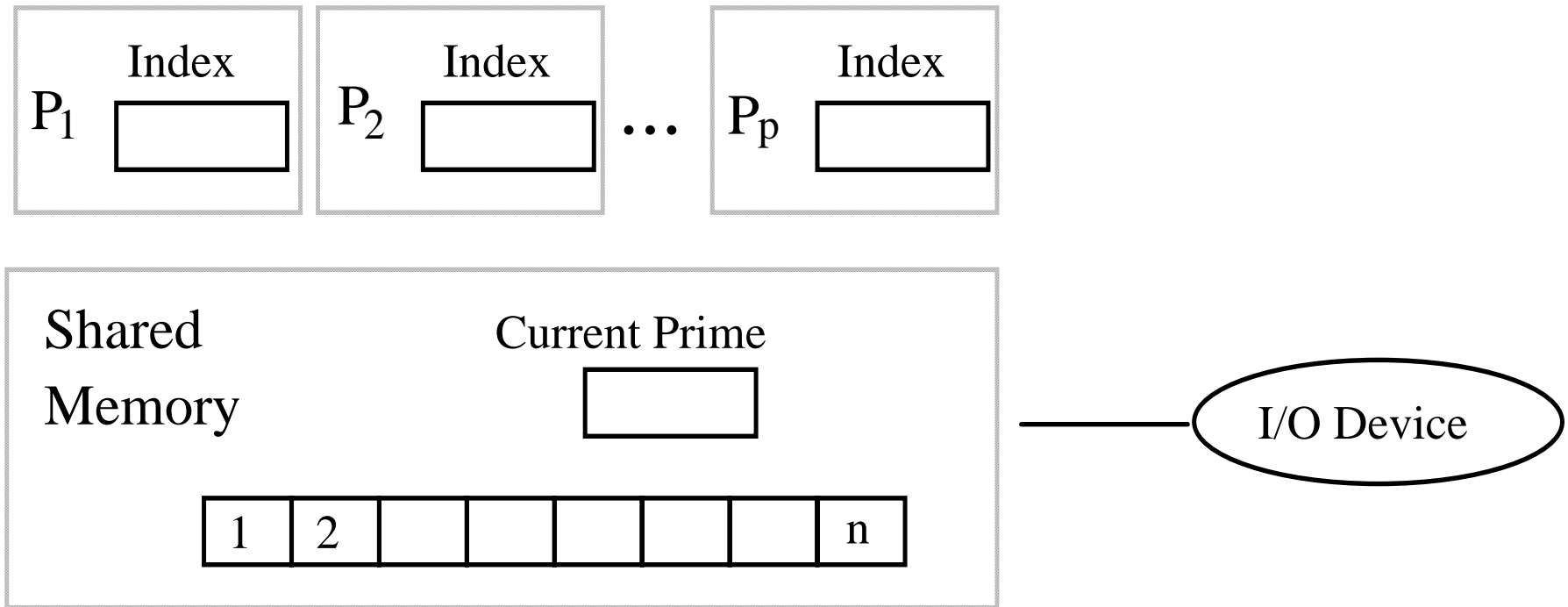


Fig. 1.5 Schematic representation of a control-parallel solution for the sieve of Eratosthenes.

# Running Time of the Sequential/Parallel Sieve

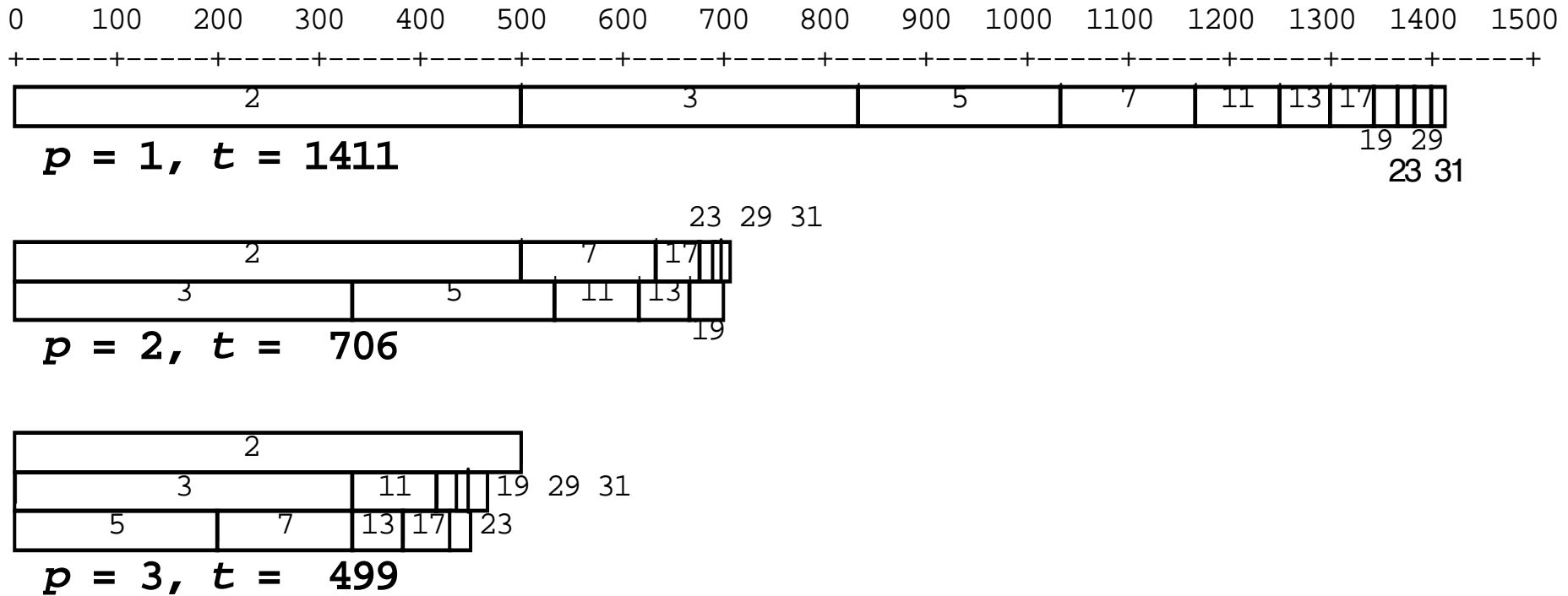


Fig. 1.6 Control-parallel realization of the sieve of Eratosthenes with  $n = 1000$  and  $1 \leq p \leq 3$ .

# Data-Parallel Implementation of the Sieve

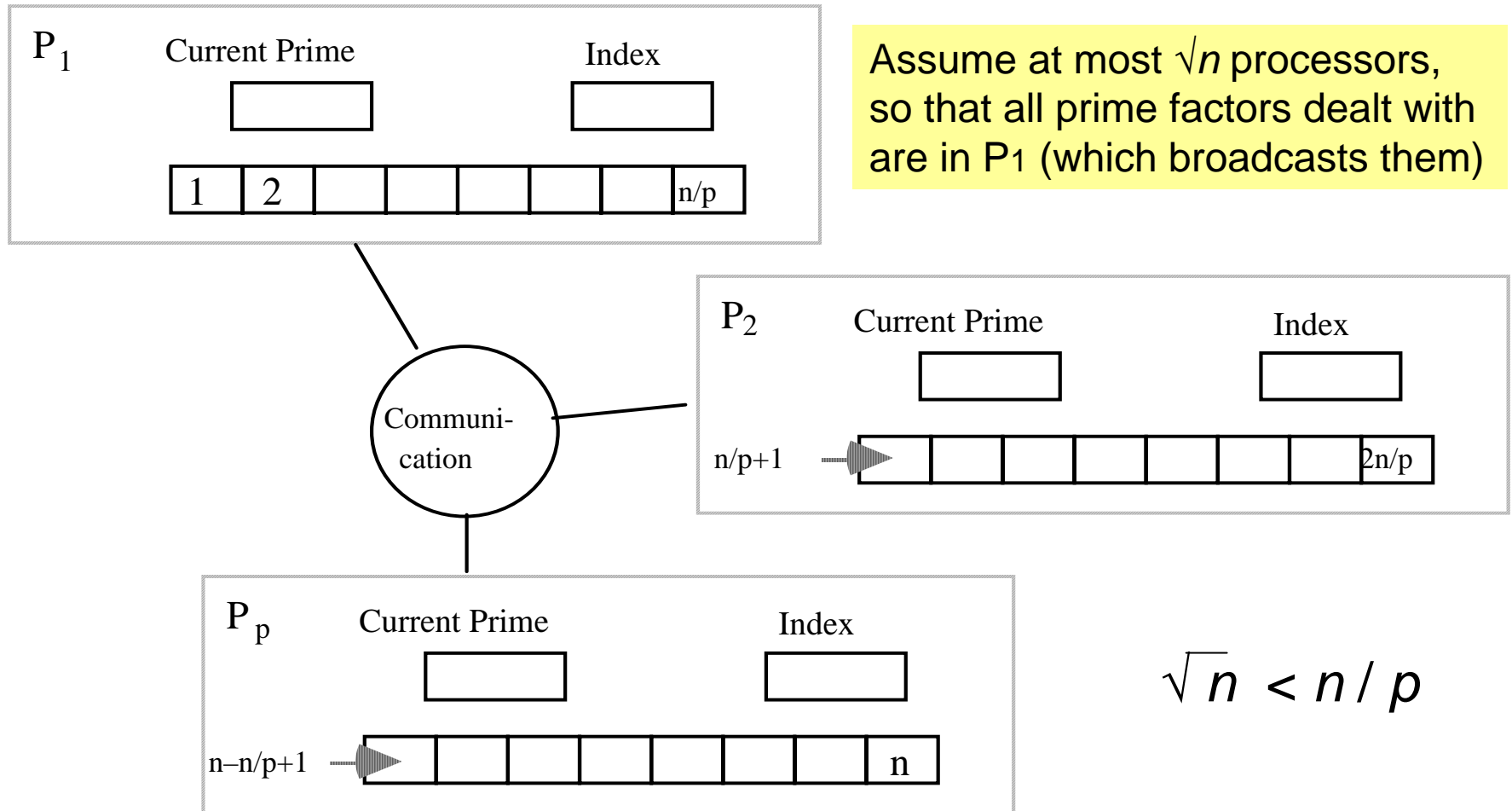


Fig. 1.7 Data-parallel realization of the sieve of Eratosthenes.

# One Reason for Sublinear Speedup: Communication Overhead

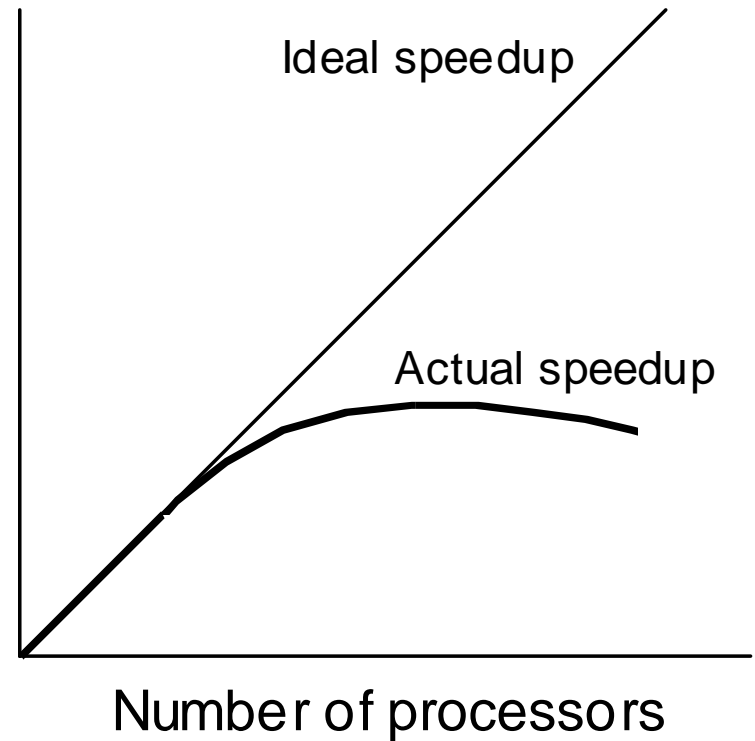
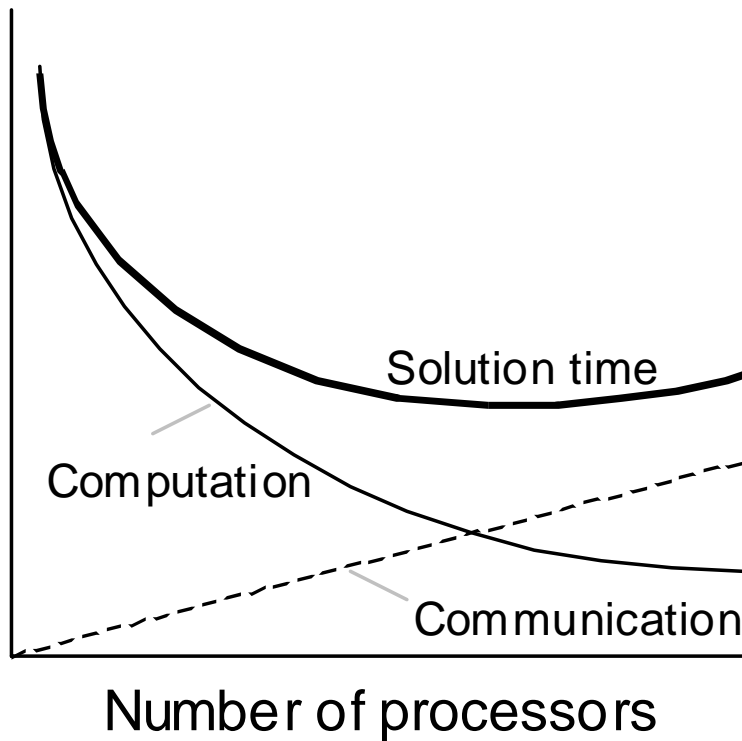


Fig. 1.8 Trade-off between communication time and computation time in the data-parallel realization of the sieve of Eratosthenes.

# Another Reason for Sublinear Speedup: Input/Output Overhead

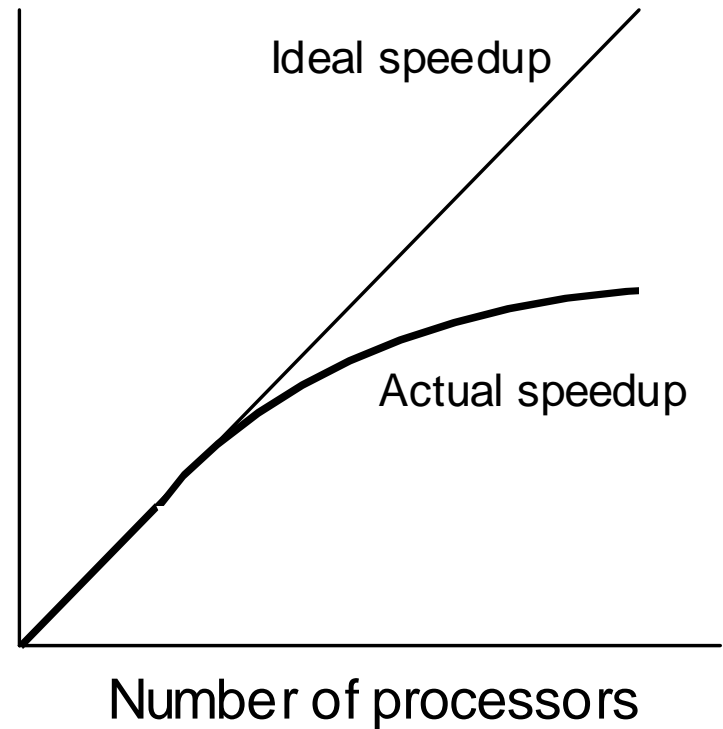
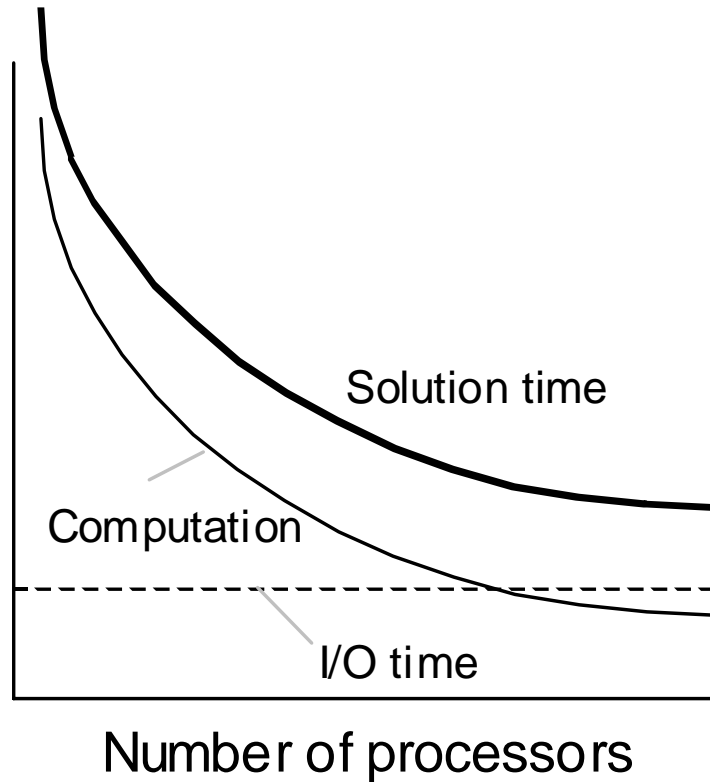
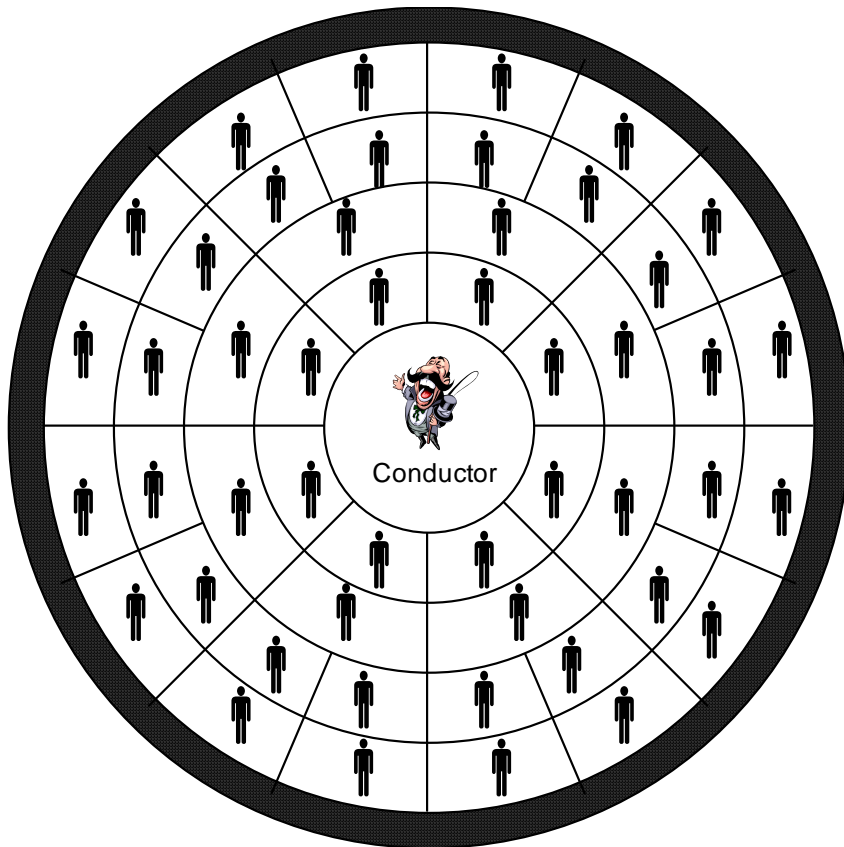


Fig. 1.9 Effect of a constant I/O time on the data-parallel realization of the sieve of Eratosthenes.



# 1.3 Parallel Processing Ups and Downs

Fig. 1.10 Richardson's circular theater for weather forecasting calculations.



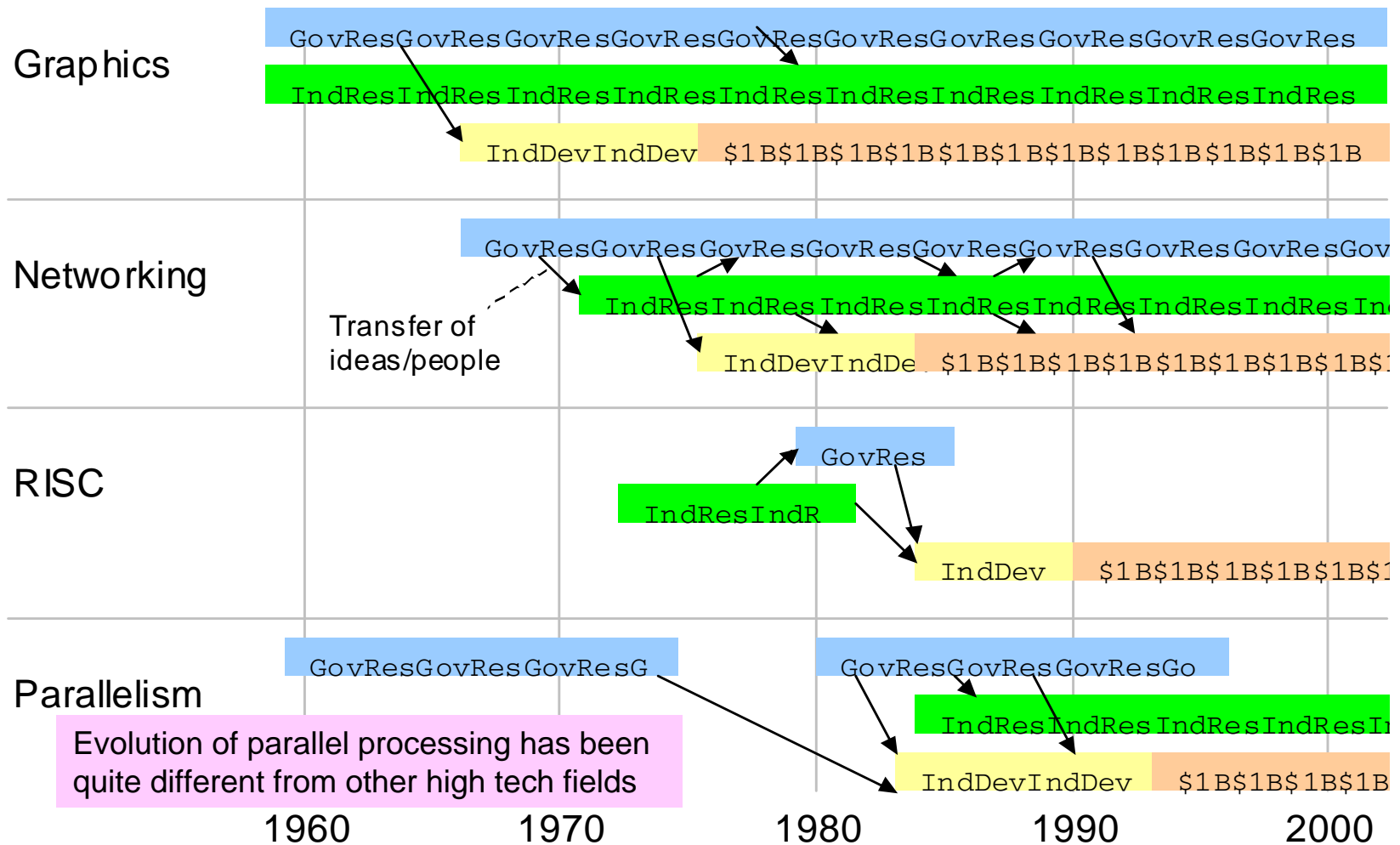
Using thousands of “computers” (humans + calculators) for 24-hr weather prediction in a few hours

1960s: ILLIAC IV (U Illinois) – four  $8 \times 8$  mesh quadrants, SIMD

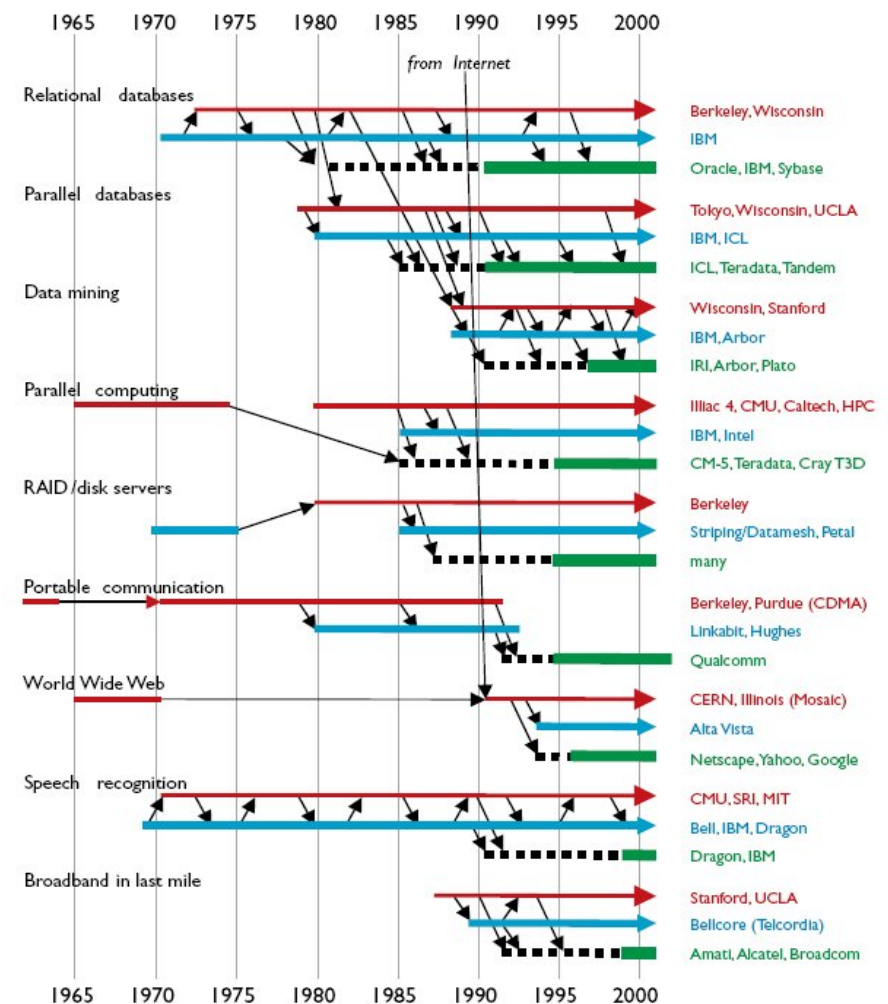
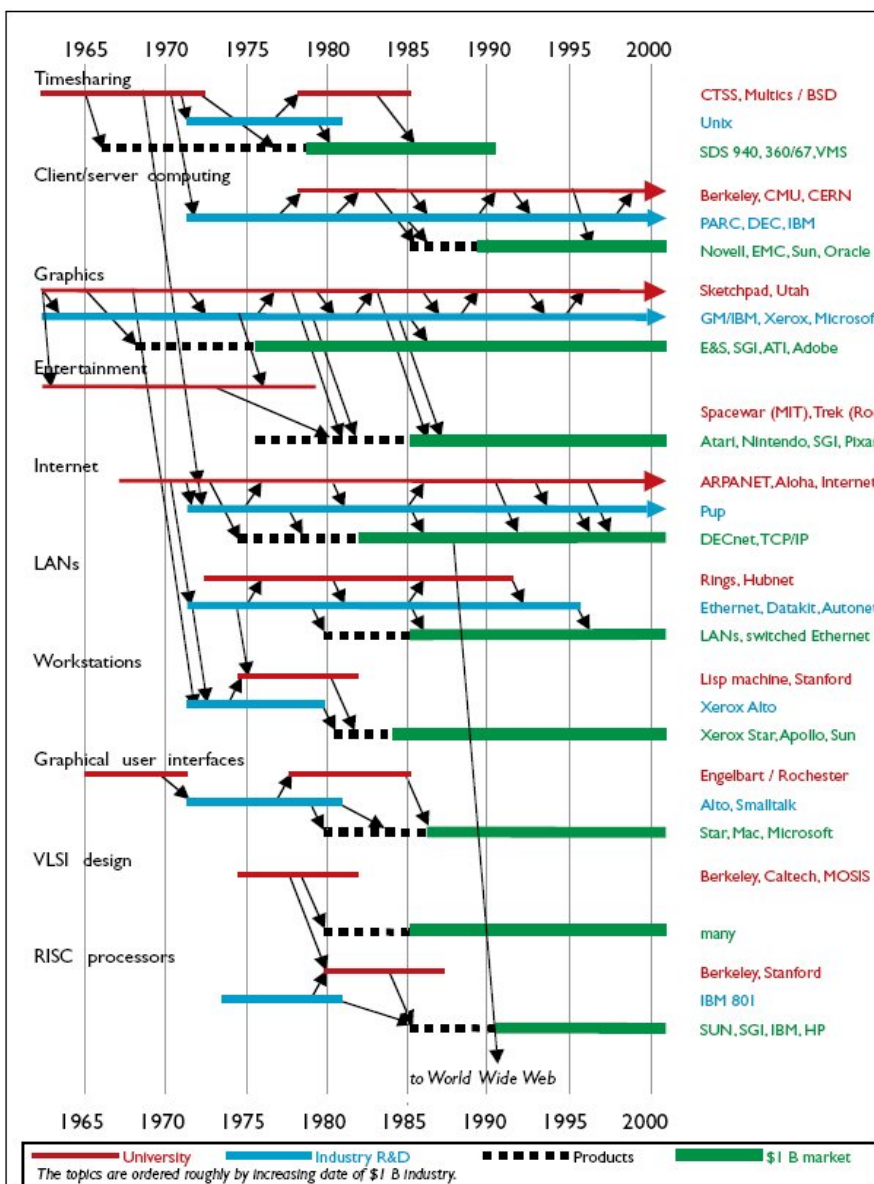
1980s: Commercial interest – technology was driven by government grants & contracts. Once funding dried up, many companies went bankrupt

2000s: Internet revolution – info providers, multimedia, data mining, etc. need lots of power

# Trends in High-Technology Development



Development of some technical fields into \$1B businesses and the roles played by government research and industrial R&D over time (*IEEE Computer*, early 90s?).



## Trends in Hi-Tech Development (2003)

Source: From [6], reprinted with permission from the National Academy of Sciences, courtesy of the National Academies Press, Washington D.C. ©2003.

# Status of Computing Power (circa ~~2000~~<sup>2010</sup>)

~~TFLOPS~~

~~GFLOPS~~ on desktop: Apple Macintosh, with G4 processor

~~PFLOPS~~

~~TFLOPS~~ in supercomputer center:

1152-processor IBM RS/6000 SP (switch-based network)

Cray T3E, torus-connected

~~EFLOPS~~ (Exa =  $10^{18}$ )

~~PFLOPS~~ on drawing board:

1M-processor IBM Blue Gene (2005?)

32 proc's/chip, 64 chips/board, 8 boards/tower, 64 towers

Processor: 8 threads, on-chip memory, no data cache

Chip: defect-tolerant, row/column rings in a  $6 \times 6$  array

Board:  $8 \times 8$  chip grid organized as  $4 \times 4 \times 4$  cube

Tower: Boards linked to 4 neighbors in adjacent towers

System:  $32 \times 32 \times 32$  cube of chips, 1.5 MW (water-cooled)

# 1.4 Types of Parallelism: A Taxonomy

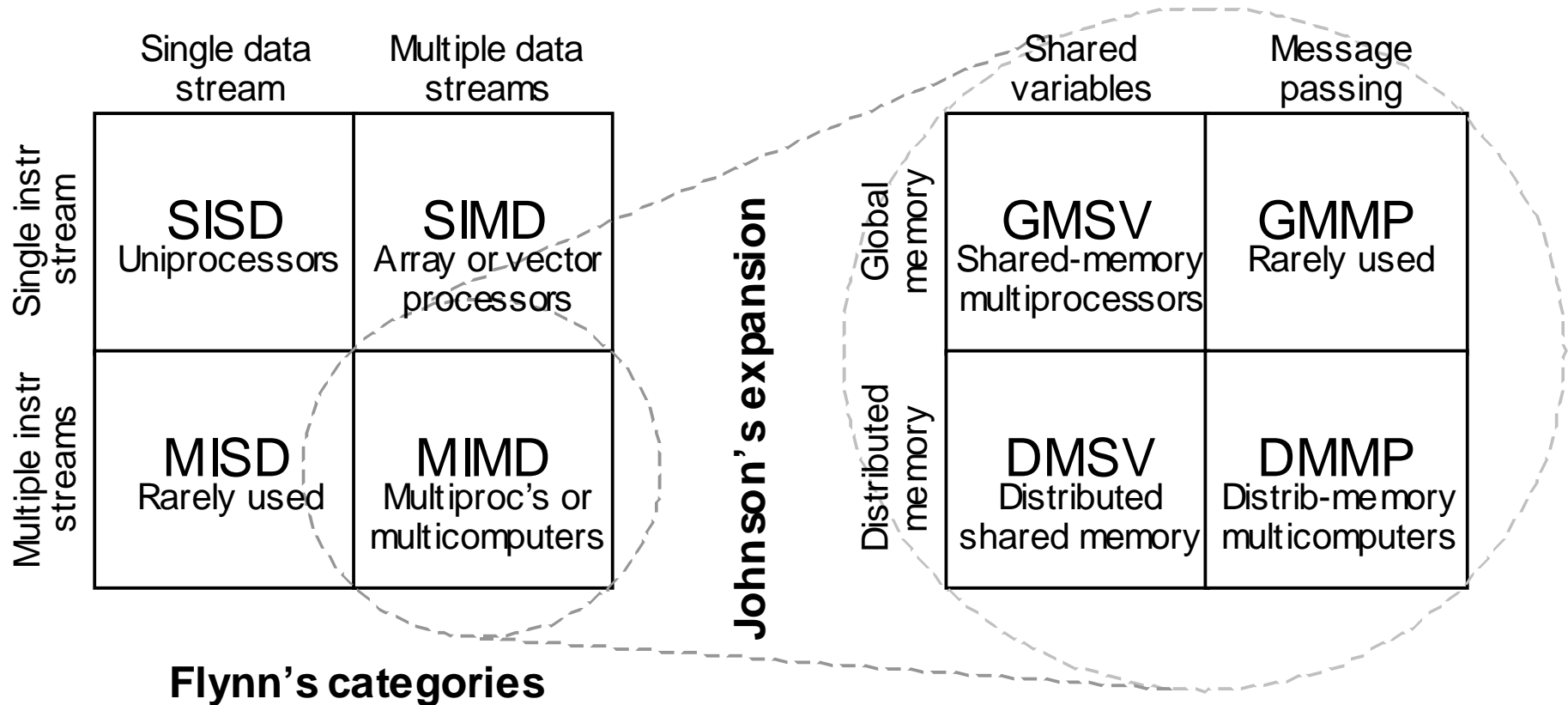
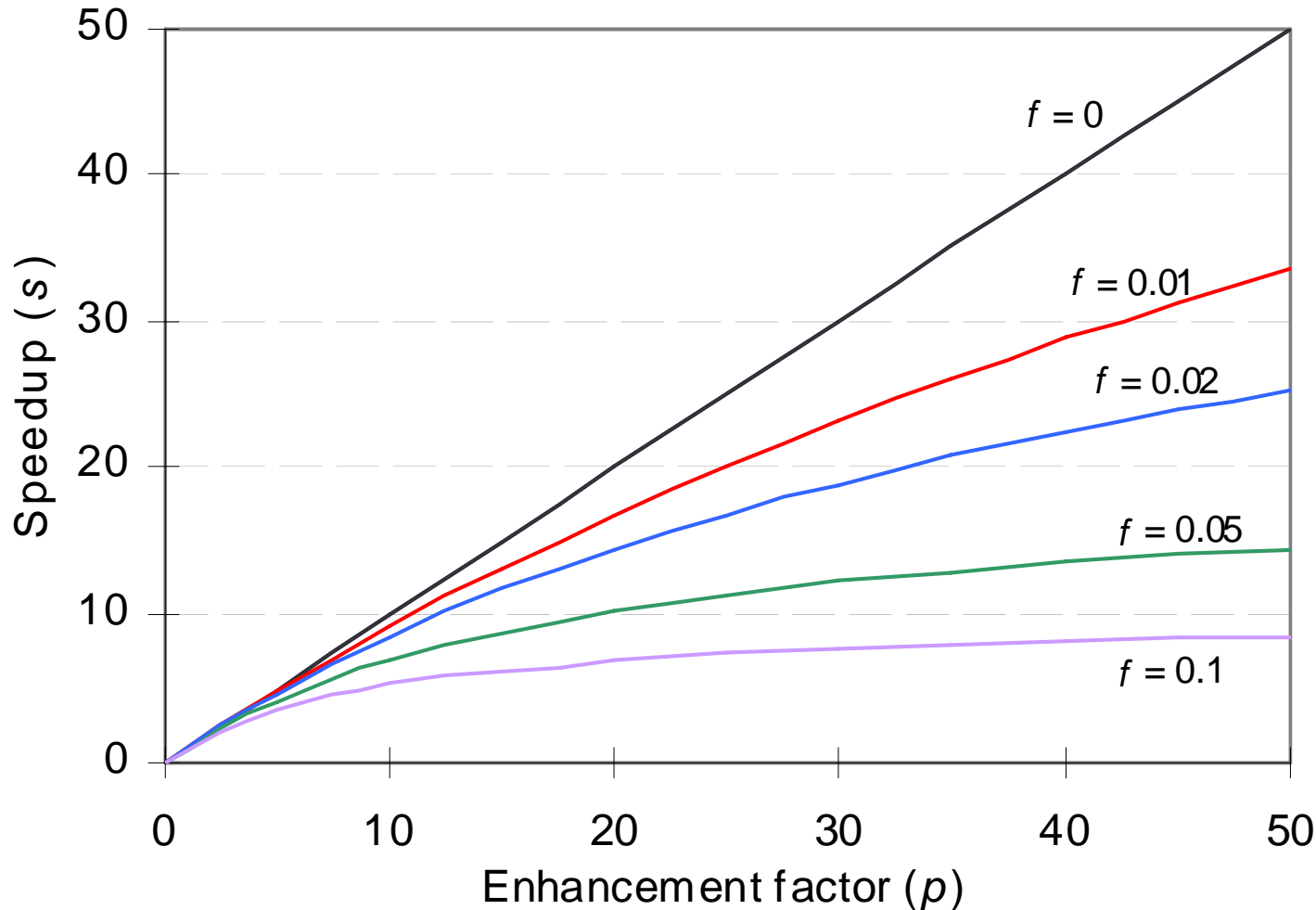


Fig. 1.11 The Flynn-Johnson classification of computer systems.

# 1.5 Roadblocks to Parallel Processing

- Grosch's law: Economy of scale applies, or power = cost<sup>2</sup>  
No longer valid; in fact we can get more bang per buck in micros
- Minsky's conjecture: Speedup tends to be proportional to  $\log p$   
Has roots in analysis of memory bank conflicts; can be overcome
- Tyranny of IC technology: Uniprocessors suffice (x10 faster/5 yrs)  
Faster ICs make parallel machines faster too; what about x1000?
- Tyranny of vector supercomputers: Familiar programming model  
Not all computations involve vectors; parallel vector machines
- Software inertia: Billions of dollars investment in software  
New programs; even uniprocessors benefit from parallelism spec
- Amdahl's law: Unparallelizable code severely limits the speedup

# Amdahl's Law



$f$  = fraction  
unaffected

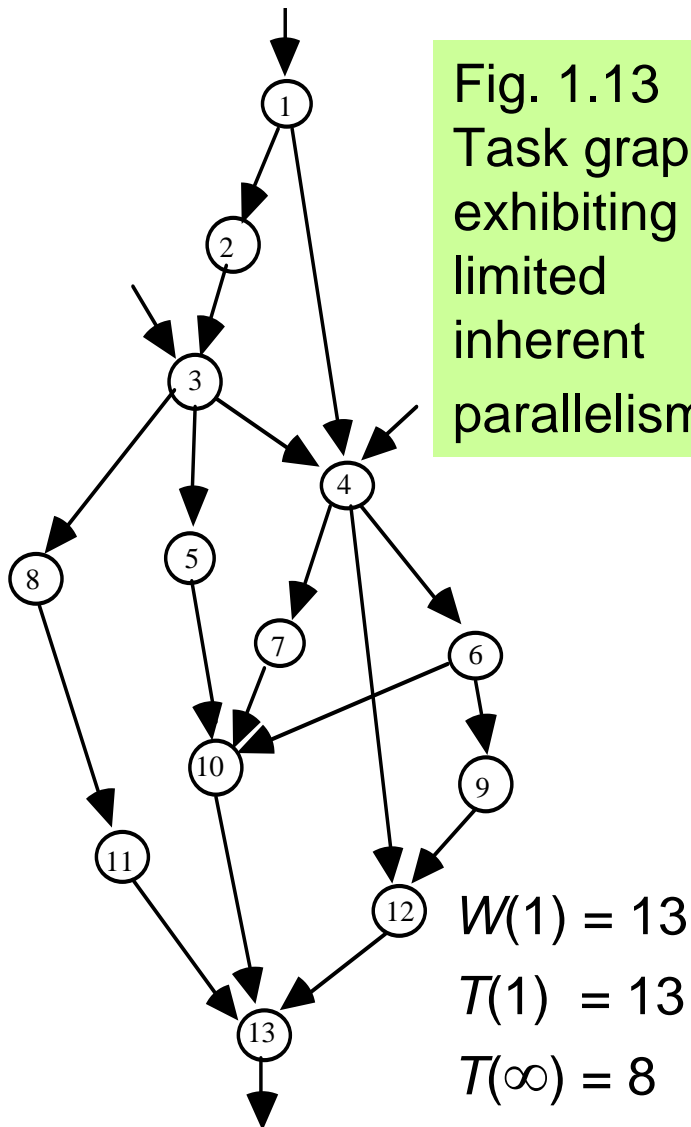
$p$  = speedup  
of the rest

$$s = \frac{1}{f + (1 - f)/p}$$
$$\leq \min(p, 1/f)$$

Fig. 1.12 Limit on speed-up according to Amdahl's law.



# 1.6 Effectiveness of Parallel Processing



$p$  Number of processors

$W(p)$  Work performed by  $p$  processors

$T(p)$  Execution time with  $p$  processors  
 $T(1) = W(1); \quad T(p) \leq W(p)$

$S(p)$  Speedup =  $T(1) / T(p)$

$E(p)$  Efficiency =  $T(1) / [p T(p)]$

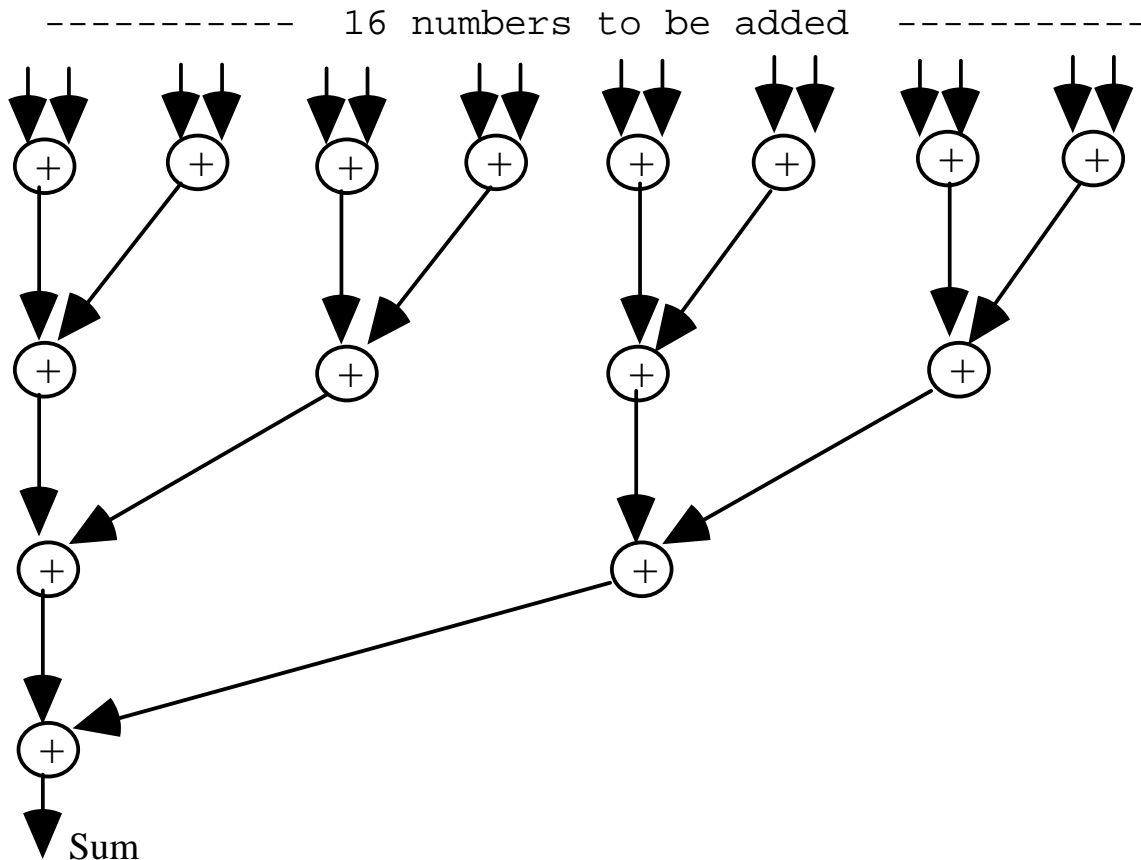
$R(p)$  Redundancy =  $W(p) / W(1)$

$U(p)$  Utilization =  $W(p) / [p T(p)]$

$Q(p)$  Quality =  $T^3(1) / [p T^2(p) W(p)]$

# Reduction or Fan-in Computation

Example: Adding 16 numbers, 8 processors, unit-time additions



*Zero-time communication*

$$E(8) = 15 / (8 \times 4) = 47\%$$

$$S(8) = 15 / 4 = 3.75$$

$$R(8) = 15 / 15 = 1$$

$$Q(8) = 1.76$$

*Unit-time communication*

$$E(8) = 15 / (8 \times 7) = 27\%$$

$$S(8) = 15 / 7 = 2.14$$

$$R(8) = 22 / 15 = 1.47$$

$$Q(8) = 0.39$$

Fig. 1.14 Computation graph for finding the sum of 16 numbers .

# ABCs of Parallel Processing in One Slide

## A Amdahl's Law (Speedup Formula)

**Bad news** – Sequential overhead will kill you, because:

$$\text{Speedup} = T_1/T_p \leq 1/[f + (1 - f)/p] \leq \min(1/f, p)$$

**Morale:** For  $f = 0.1$ , speedup is at best 10, regardless of peak OPS.

## B Brent's Scheduling Theorem

**Good news** – Optimal scheduling is very difficult, but even a naive scheduling algorithm can ensure:

$$T_1/p \leq T_p < T_1/p + T_\infty = (T_1/p)[1 + p/(T_1/T_\infty)]$$

**Result:** For a reasonably parallel task (large  $T_1/T_\infty$ ), or for a suitably small  $p$  (say,  $p < T_1/T_\infty$ ), good speedup and efficiency are possible.

## C Cost-Effectiveness Adage

**Real news** – The most cost-effective parallel solution may not be the one with highest peak OPS (communication?), greatest speed-up (at what cost?), or best utilization (hardware busy doing what?).

**Analogy:** Mass transit might be more cost-effective than private cars even if it is slower and leads to many empty seats.

# 2 A Taste of Parallel Algorithms

Learn about the nature of parallel algorithms and complexity:

- By implementing 5 building-block parallel computations
- On 4 simple parallel architectures (20 combinations)

## Topics in This Chapter

2.1 Some Simple Computations

2.2 Some Simple Architectures

2.3 Algorithms for a Linear Array

2.4 Algorithms for a Binary Tree

2.5 Algorithms for a 2D Mesh

2.6 Algorithms with Shared Variables

## 2.1 Some Simple Computations

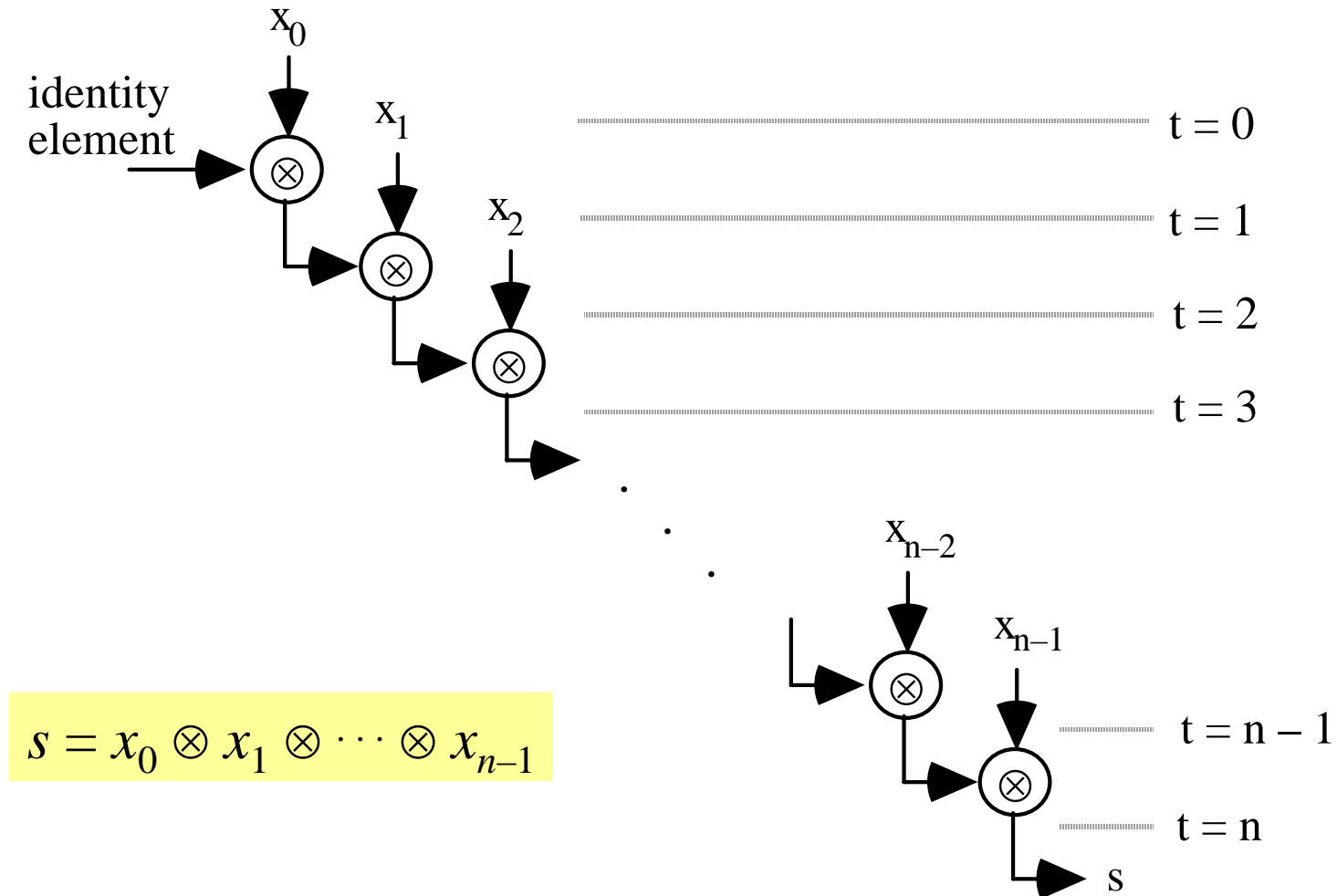
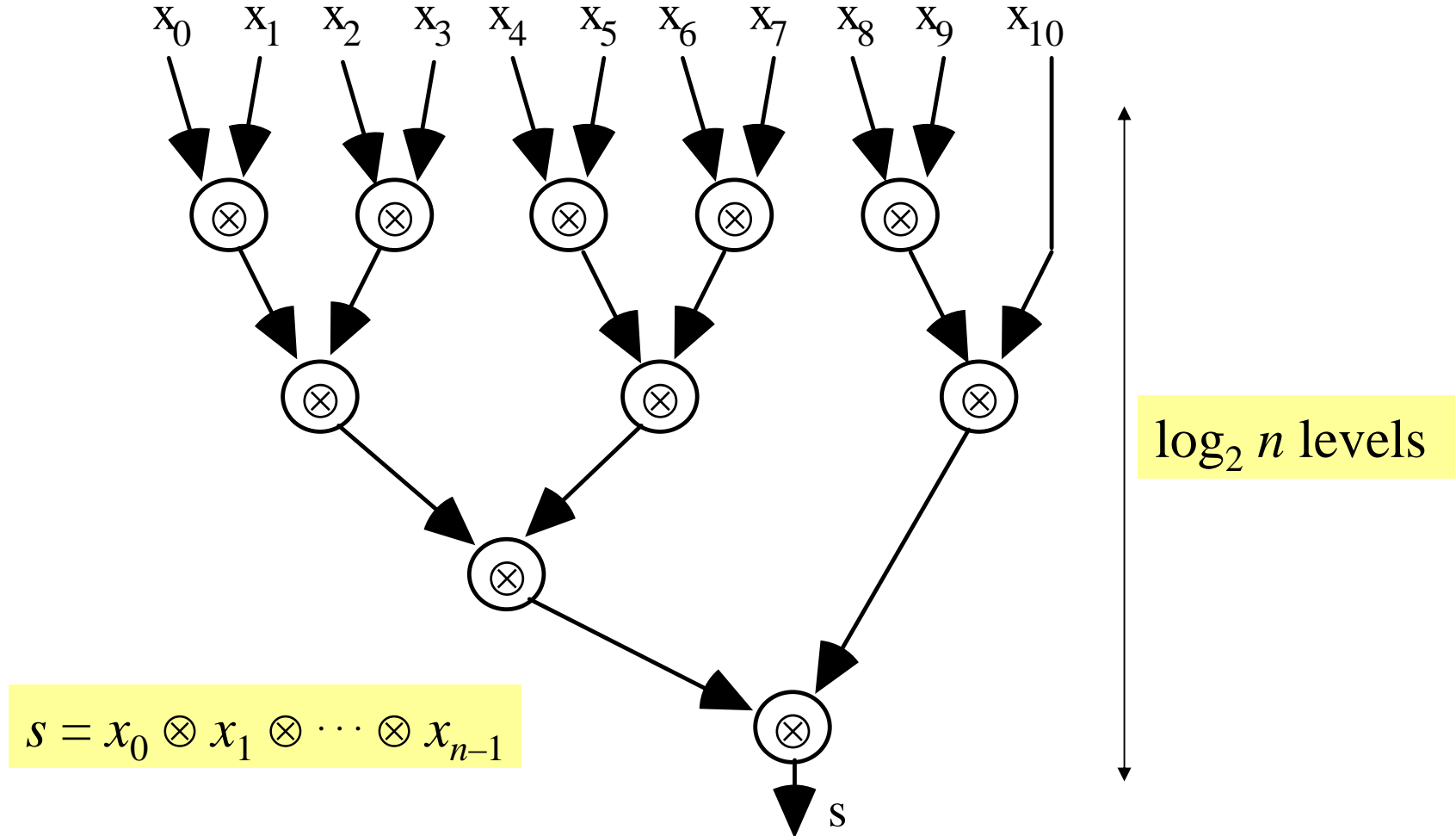


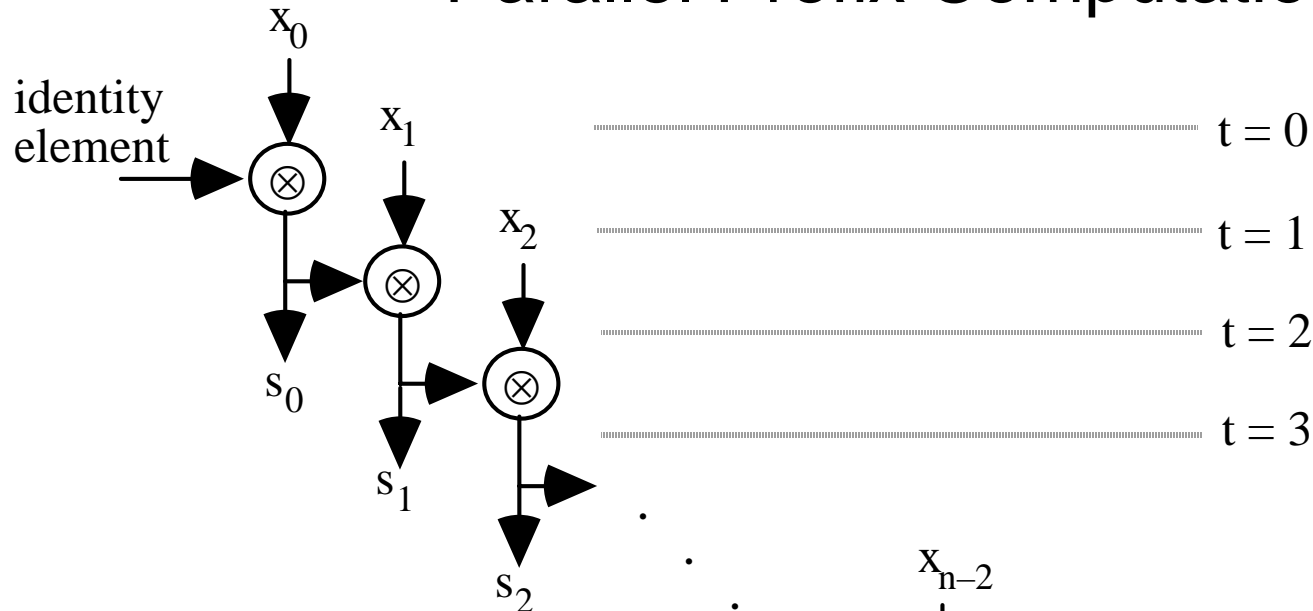
Fig. 2.1 Semigroup computation on a uniprocessor.

# Parallel Semigroup Computation



Semigroup computation viewed as tree or fan-in computation.

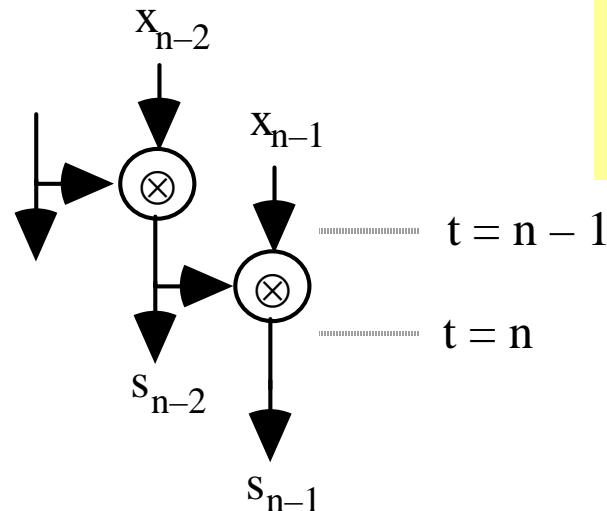
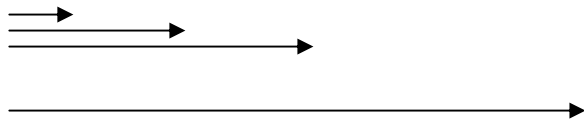
# Parallel Prefix Computation



Parallel version  
much trickier  
compared to that  
of semigroup  
computation

Requires a  
minimum of  
 $\log_2 n$  levels

$$s = x_0 \otimes x_1 \otimes x_2 \otimes \cdots \otimes x_{n-1}$$



Prefix computation on a uniprocessor.



# The Five Building-Block Computations

**Semigroup computation:** aka tree or fan-in computation  
All processors to get the computation result at the end

**Parallel prefix computation:**  
The  $i$ th processor to hold the  $i$ th prefix result at the end

**Packet routing:**  
Send a packet from a source to a destination processor

**Broadcasting:**  
Send a packet from a source to all processors

**Sorting:**  
Arrange a set of keys, stored one per processor, so that the  $i$ th processor holds the  $i$ th key in ascending order

## 2.2 Some Simple Architectures

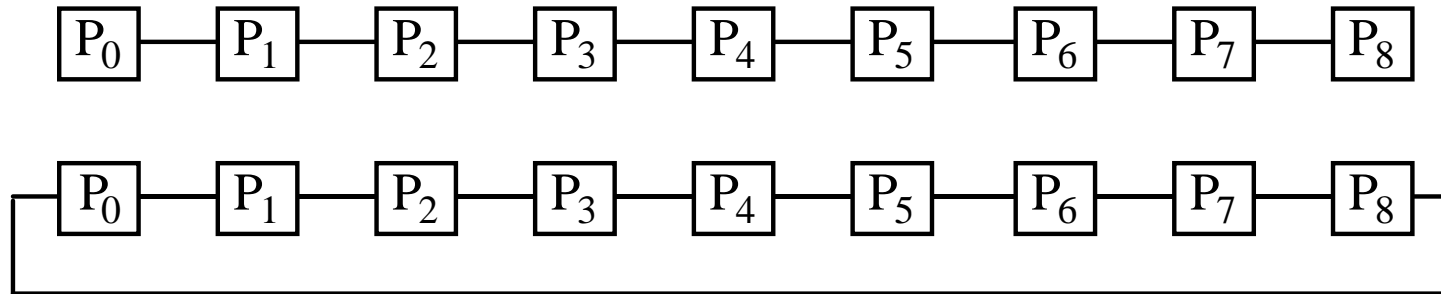


Fig. 2.2 A linear array of nine processors and its ring variant.

Max node degree	$d = 2$	
Network diameter	$D = p - 1$	$(\lfloor p/2 \rfloor)$
Bisection width	$B = 1$	$(2)$

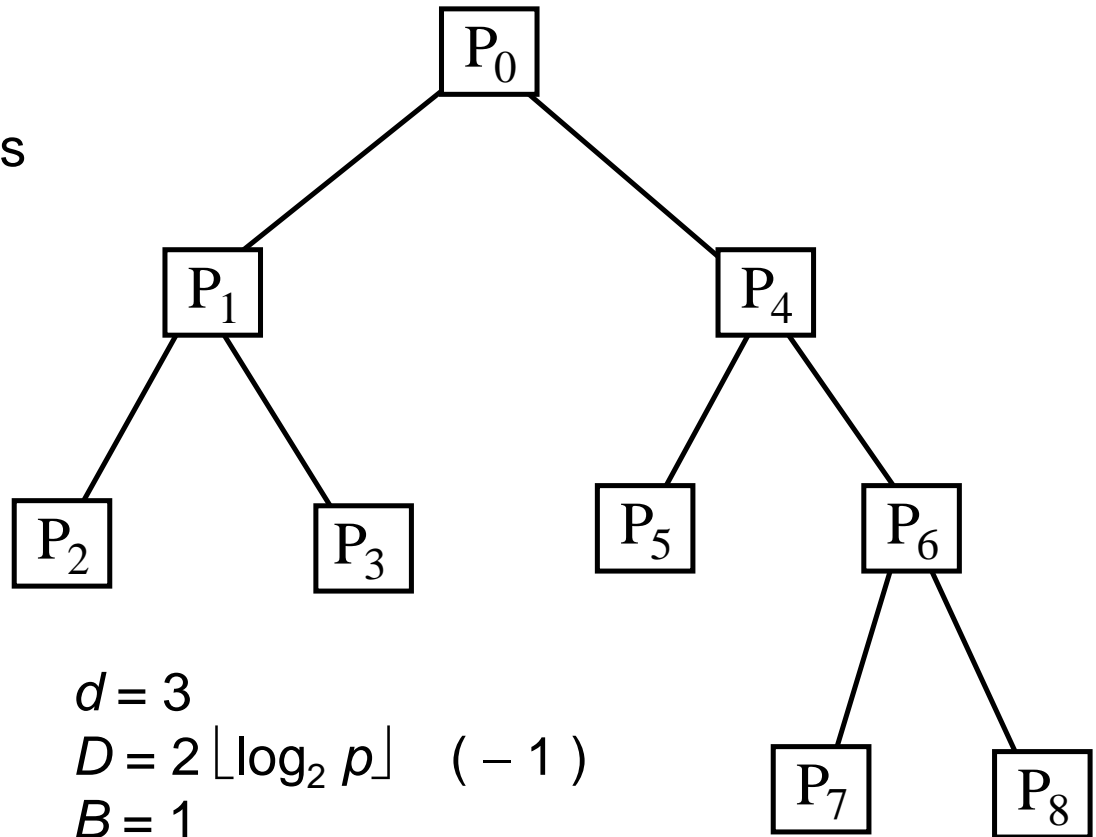
# (Balanced) Binary Tree Architecture

Complete binary tree

$2^q - 1$  nodes,  $2^{q-1}$  leaves

Balanced binary tree

Leaf levels differ by 1



Max node degree

$$d = 3$$

Network diameter

$$D = 2 \lfloor \log_2 p \rfloor - 1$$

Bisection width

$$B = 1$$

Fig. 2.3 A balanced (but incomplete) binary tree of nine processors.

# Two-Dimensional (2D) Mesh

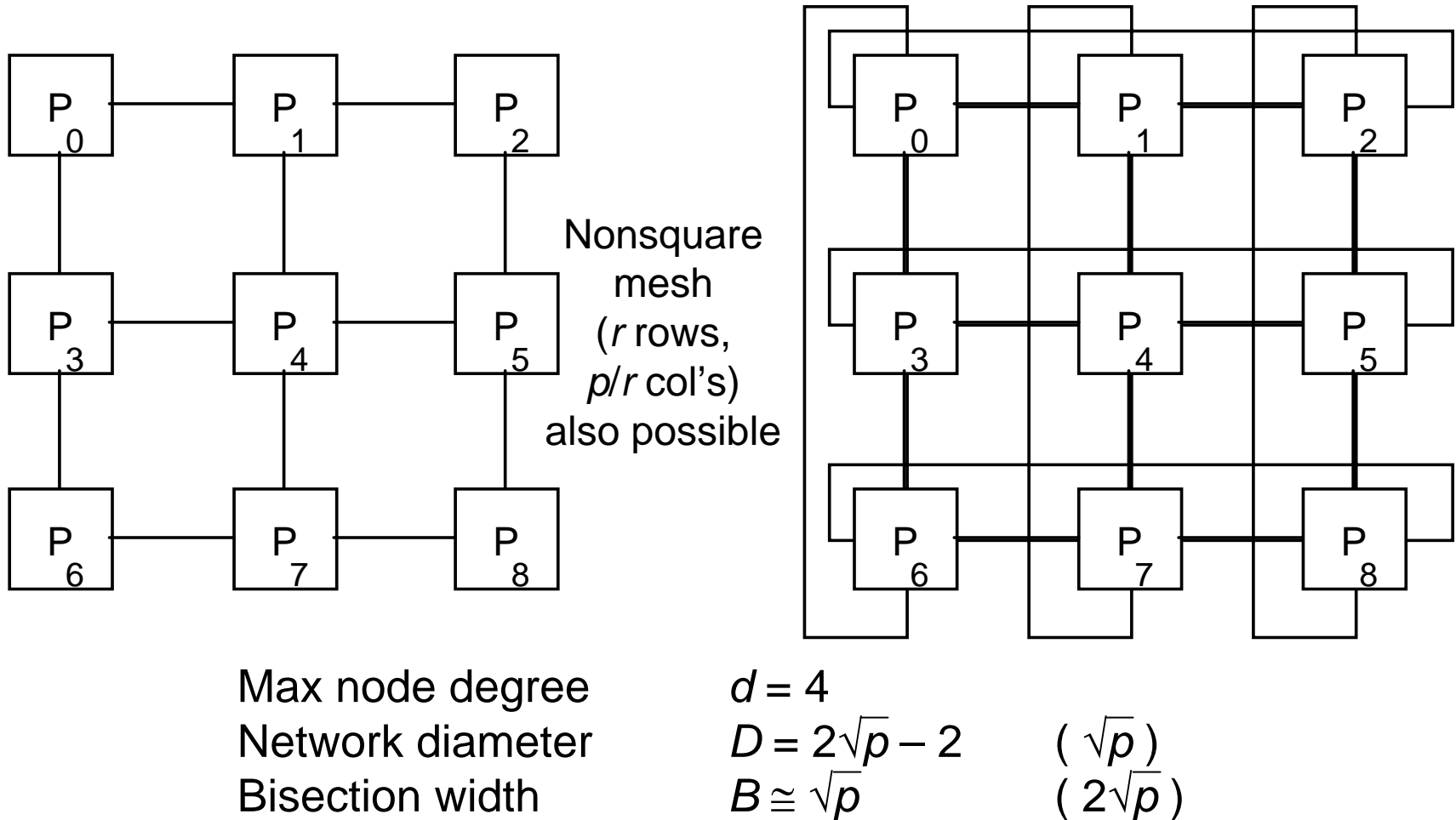
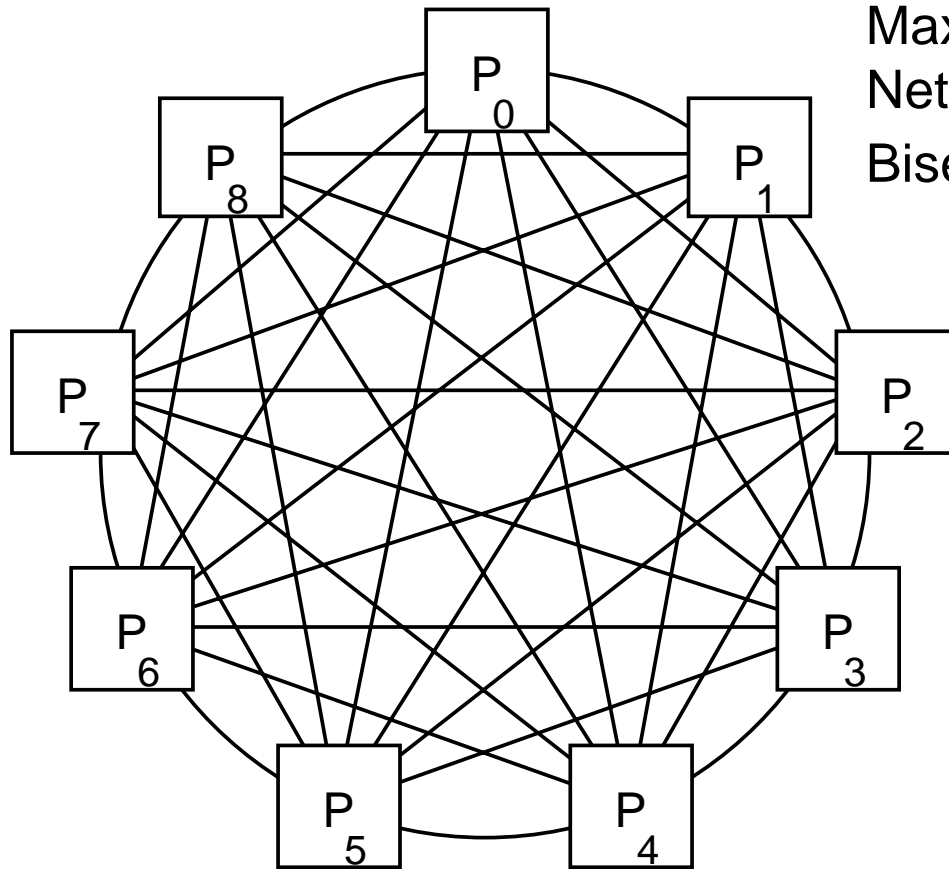


Fig. 2.4 2D mesh of 9 processors and its torus variant.

# Shared-Memory Architecture



Max node degree

$$d = p - 1$$

Network diameter

$$D = 1$$

Bisection width

$$B = \lfloor p/2 \rfloor \lceil p/2 \rceil$$

Costly to implement  
Not scalable

But . . .  
Conceptually simple  
Easy to program

Fig. 2.5 A shared-variable architecture modeled as a complete graph.

# Architecture/Algorithm Combinations

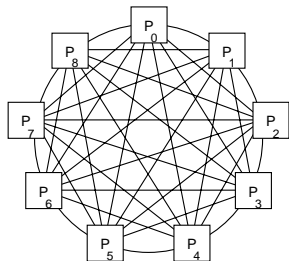
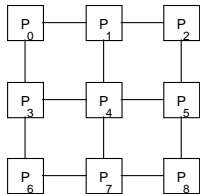
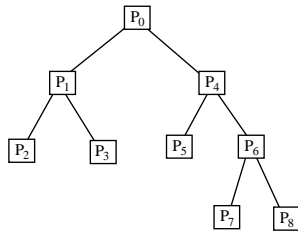
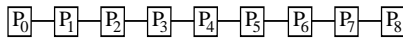
Semi-group

Parallel prefix

Packet routing

Broadcasting

Sorting



We will spend more time on linear array and binary tree

and less time on mesh and shared memory (studied later)

## 2.3 Algorithms for a Linear Array

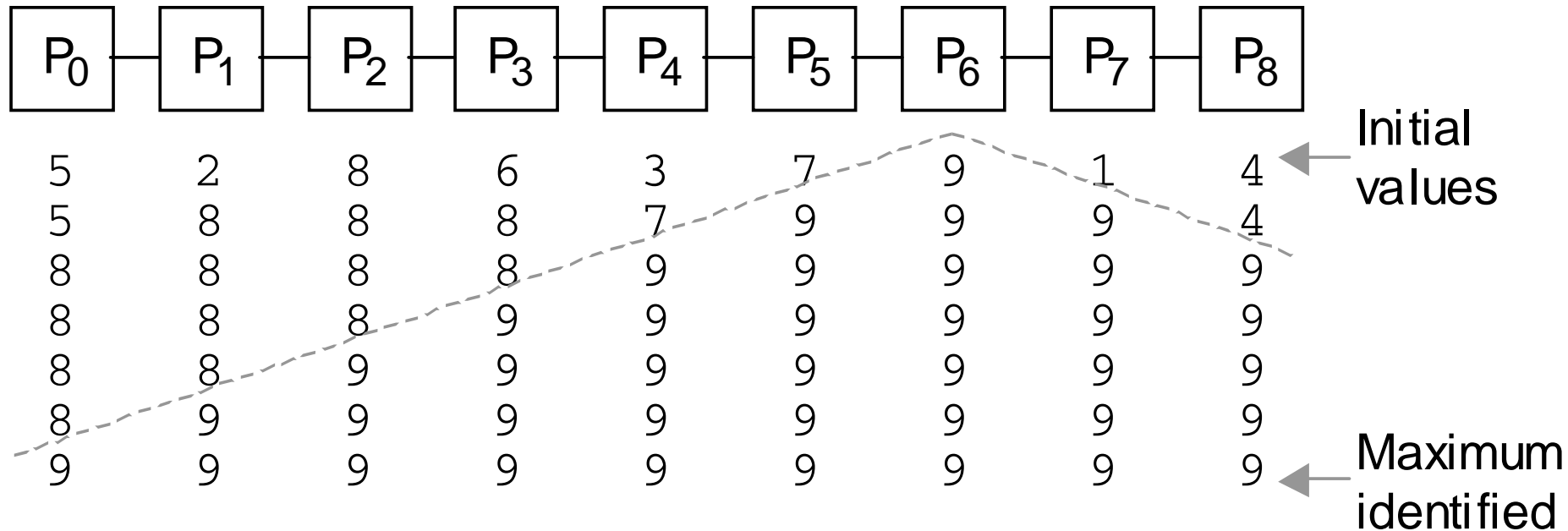


Fig. 2.6 Maximum-finding on a linear array of nine processors.

**For general semigroup computation:**

Phase 1: Partial result is propagated from left to right

Phase 2: Result obtained by processor  $p - 1$  is broadcast leftward

# Linear Array Prefix Sum Computation

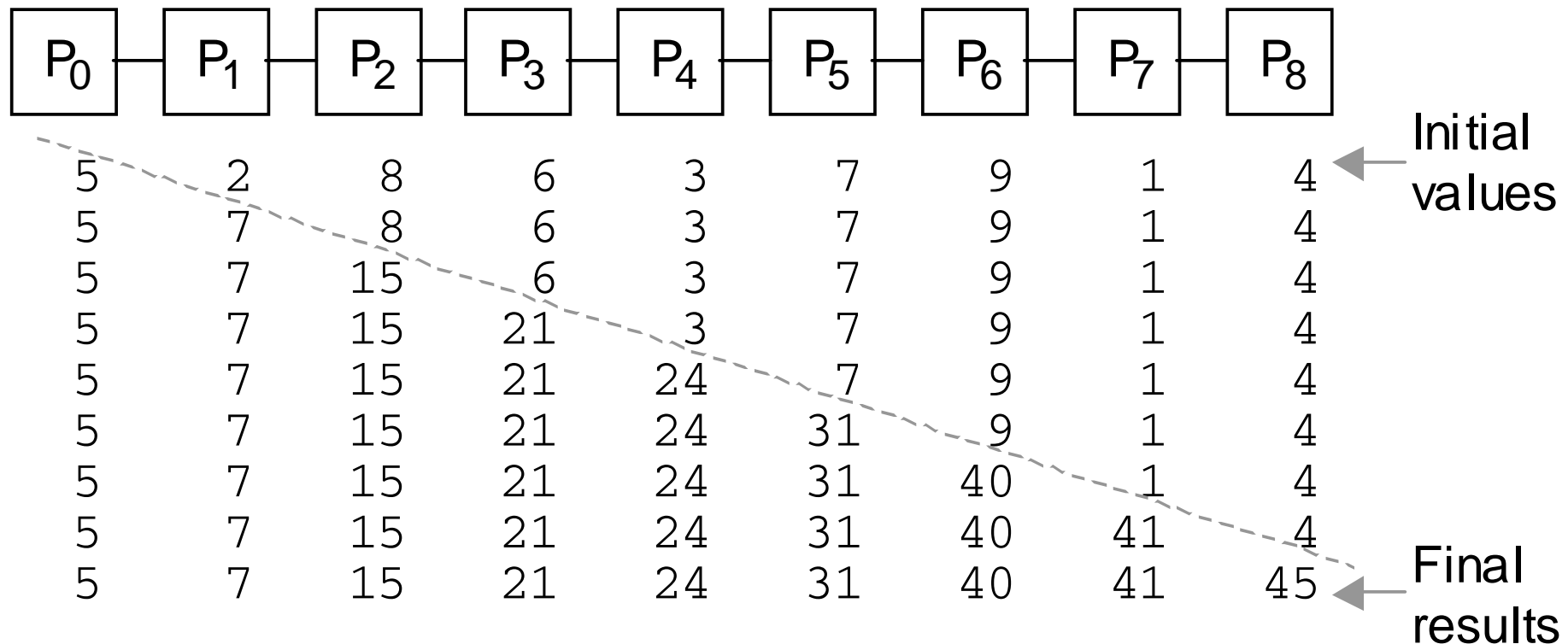


Fig. 2.7 Computing prefix sums on a linear array of nine processors.

**Diminished parallel prefix computation:**

The  $i$ th processor obtains the result up to element  $i - 1$



# Linear-Array Prefix Sum Computation

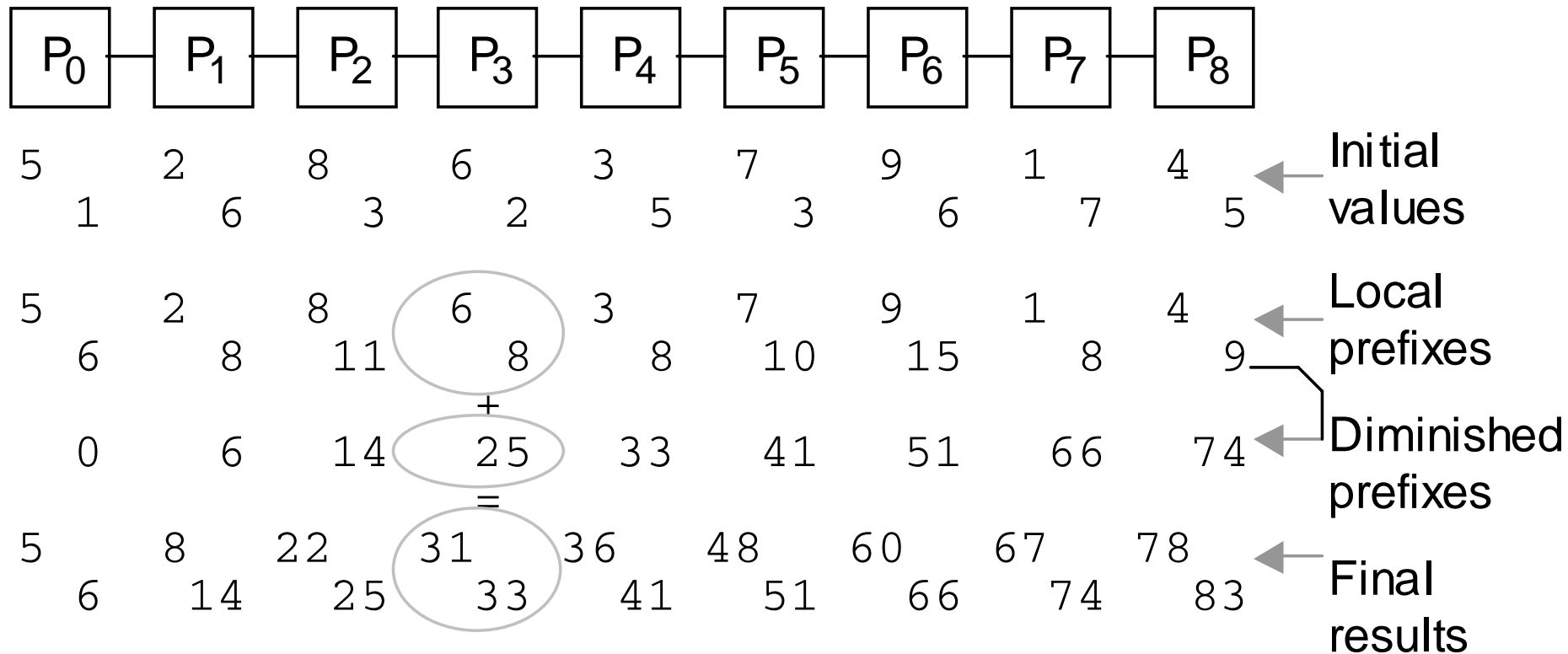
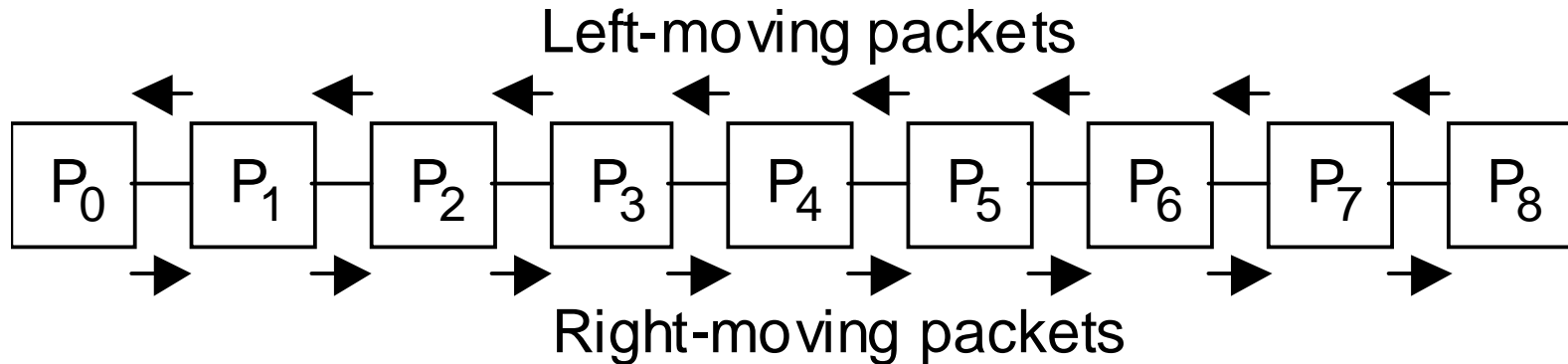


Fig. 2.8 Computing prefix sums on a linear array with two items per processor.

# Linear Array Routing and Broadcasting



Routing and broadcasting on a linear array of nine processors.

To route from processor  $i$  to processor  $j$ :

Compute  $j - i$  to determine distance and direction

To broadcast from processor  $i$ :

Send a left-moving and a right-moving broadcast message

# Linear Array Sorting (Externally Supplied Keys)

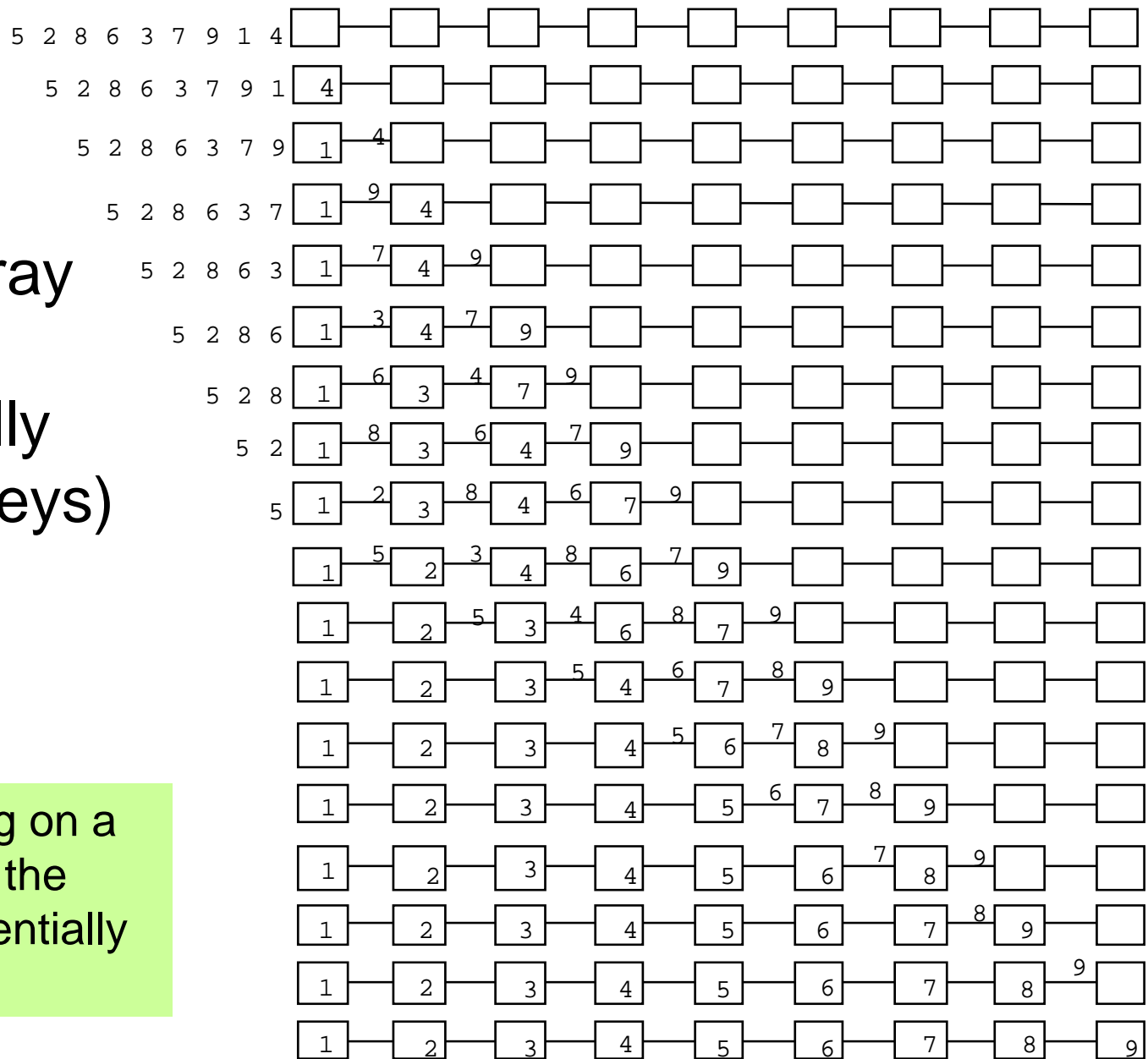


Fig. 2.9 Sorting on a linear array with the keys input sequentially from the left.

# Linear Array Sorting (Internally Stored Keys)

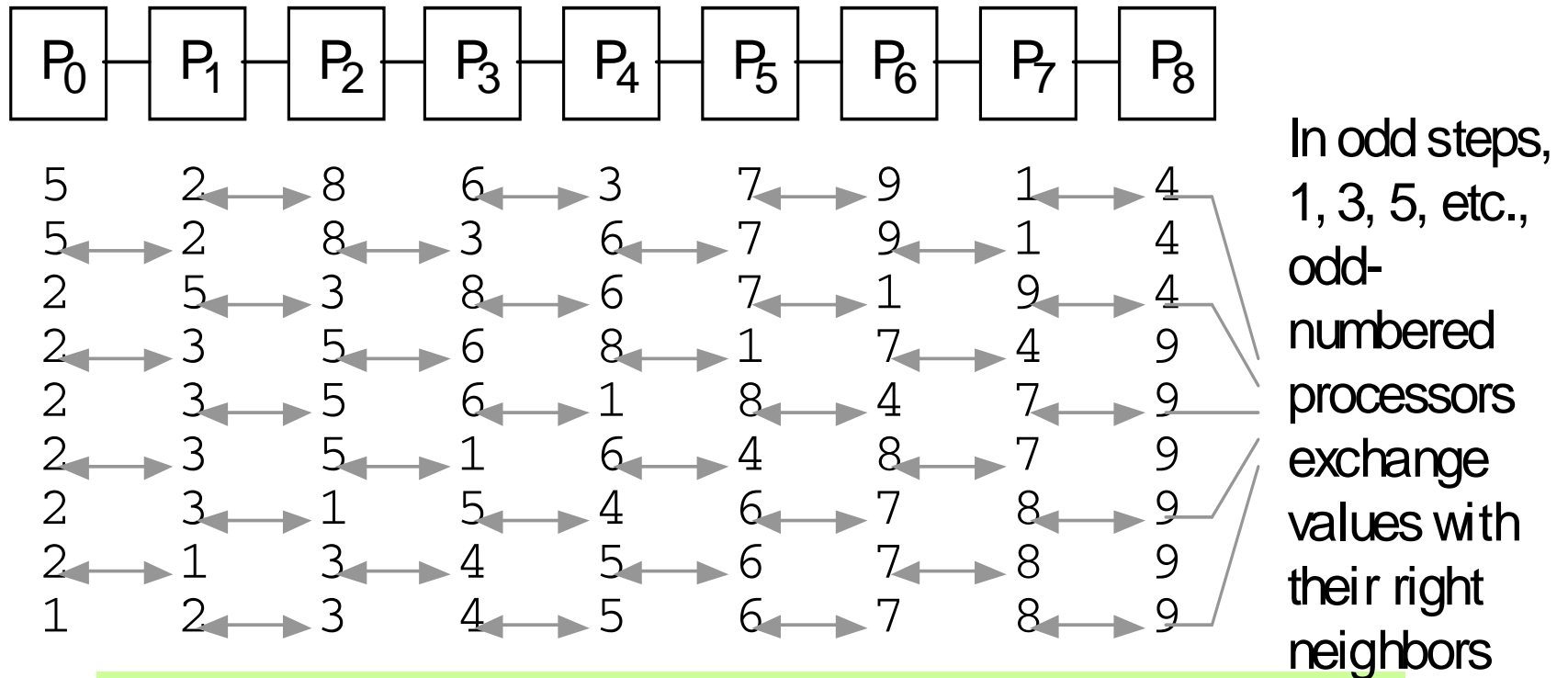


Fig. 2.10 Odd-even transposition sort on a linear array.

$$T(1) = W(1) = p \log_2 p$$

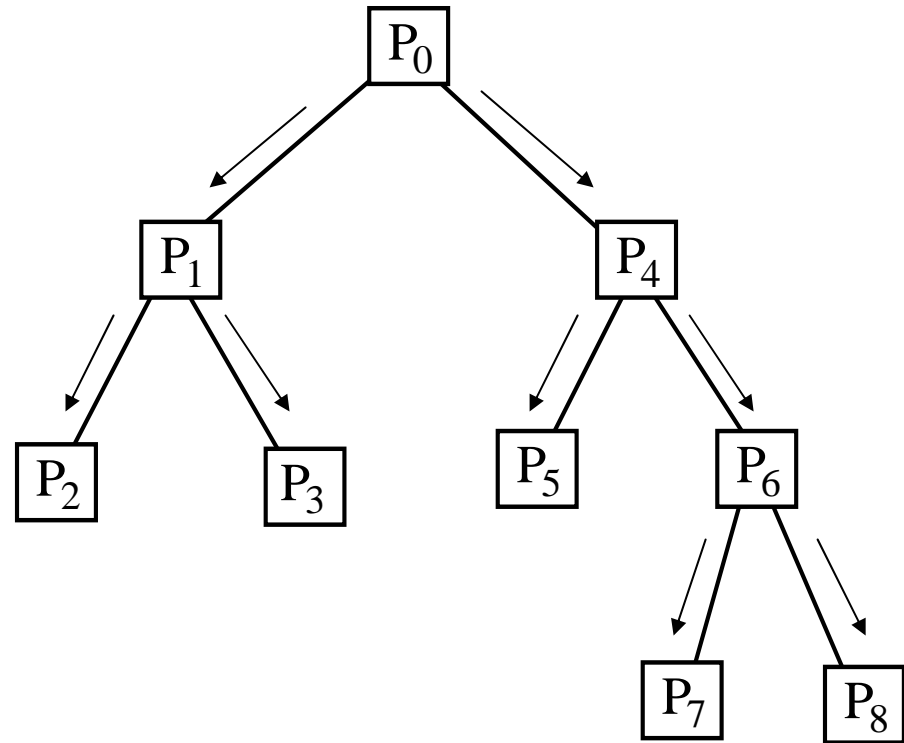
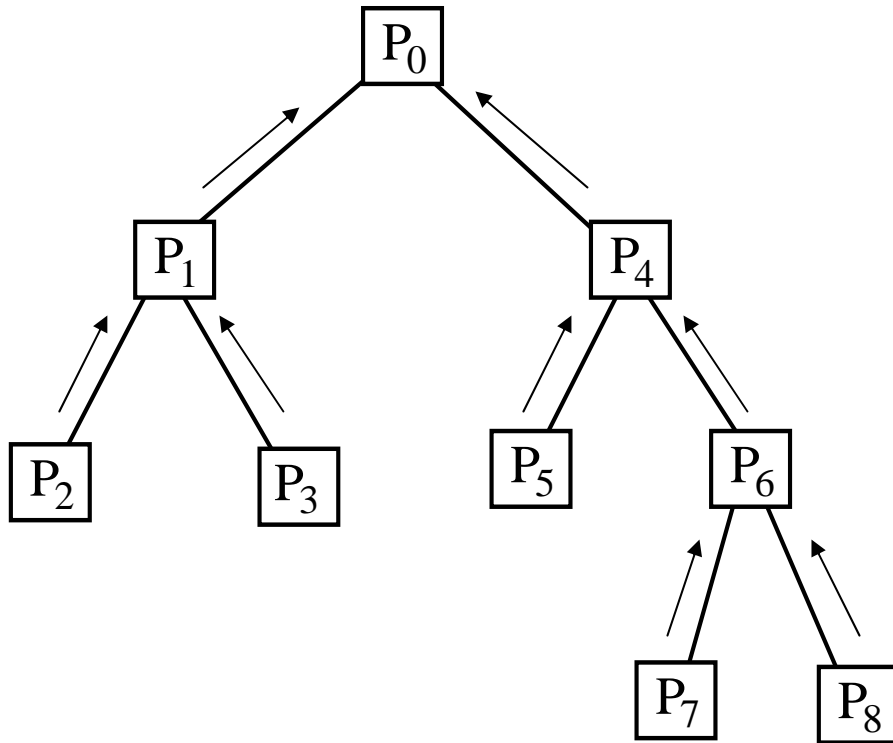
$$T(p) = p$$

$$W(p) \cong p^2/2$$

$$S(p) = \log_2 p \quad (\text{Minsky's conjecture?})$$

$$R(p) = p/(2 \log_2 p)$$

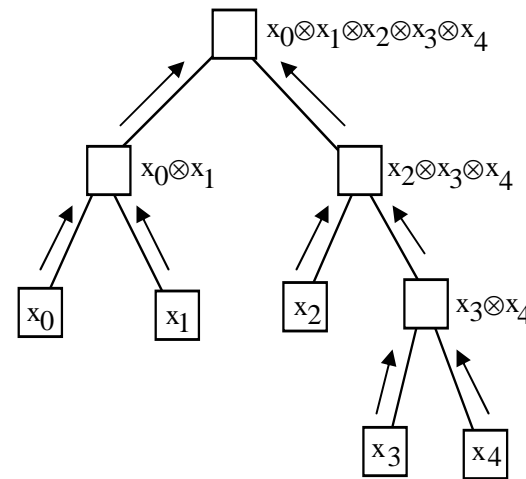
## 2.4 Algorithms for a Binary Tree



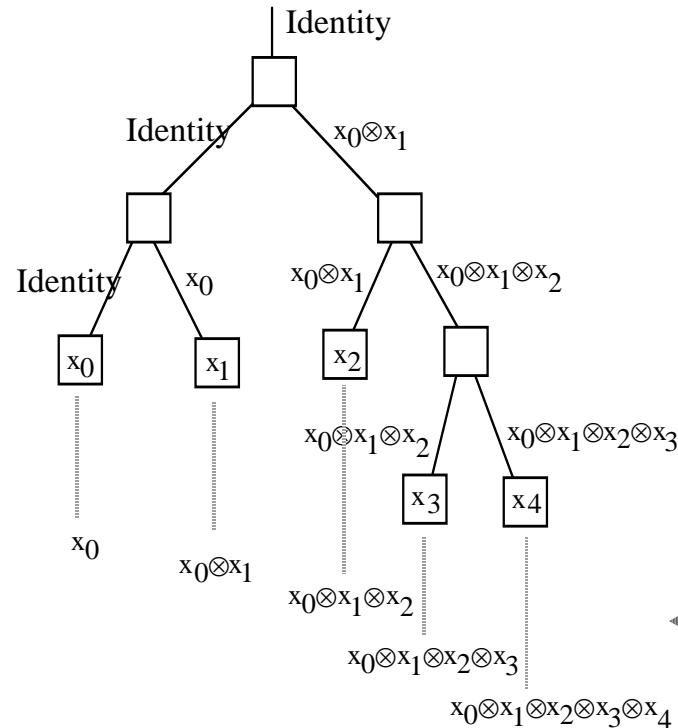
Semigroup computation and broadcasting on a binary tree.

# Binary Tree Parallel Prefix Computation

Fig. 2.11 Parallel prefix computation on a binary tree of processors.



Upward  
propagation



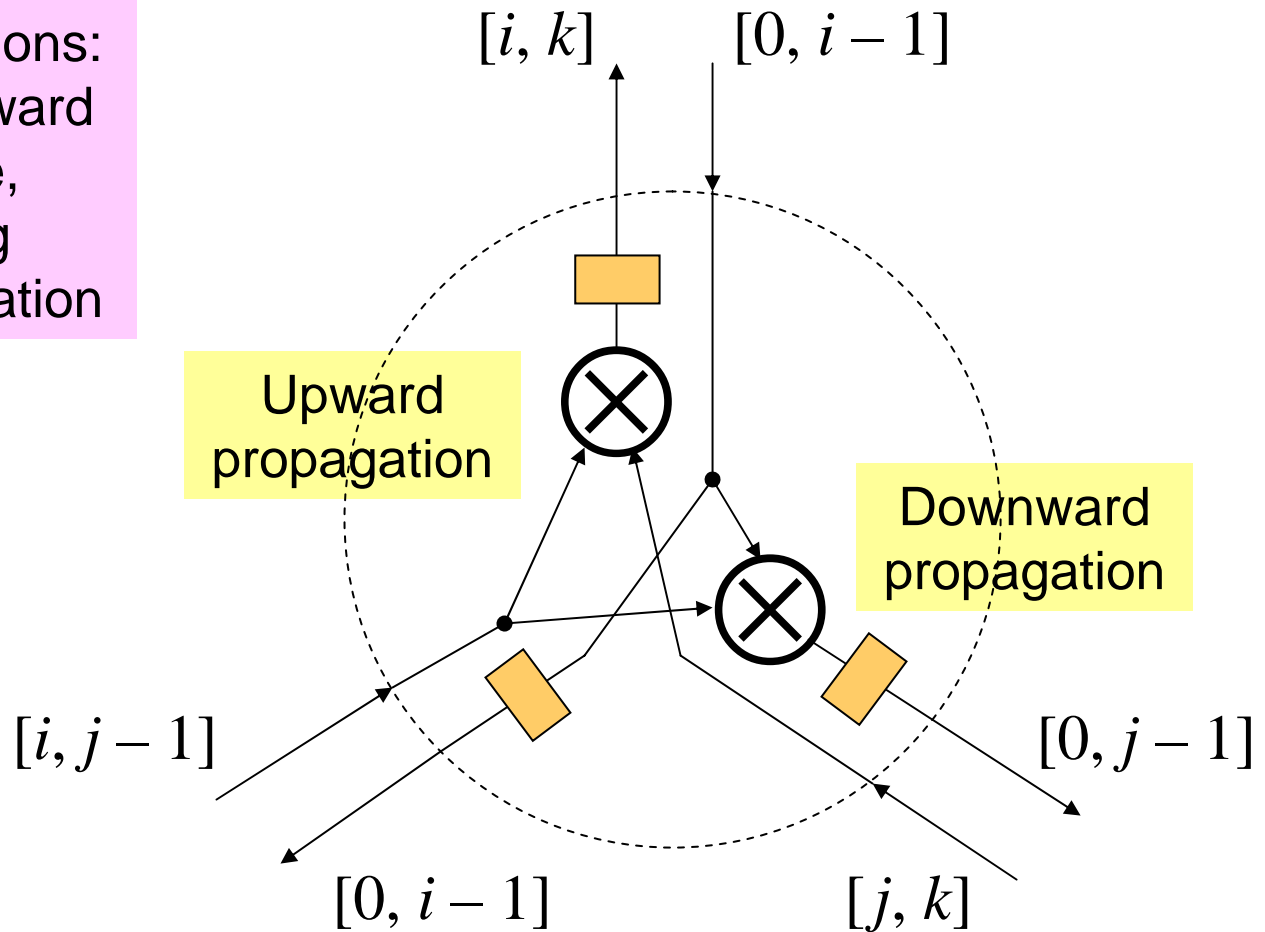
Downward  
propagation

Result

# Node Function in Binary Tree Parallel Prefix

Two binary operations:  
one during the upward  
propagation phase,  
and another during  
downward propagation

Insert latches for  
systolic operation  
(no long wires or  
propagation path)



# Usefulness of Parallel Prefix Computation

## Ranks of 1s in a list of 0s/1s:

Data:	0	0	1	0	1	0	0	1	1	1	0
Prefix sums:	0	0	1	1	2	2	2	3	4	5	5
Ranks of 1s:			1		2			3	4	5	

## Priority arbitration circuit:

Data:	0	0	1	0	1	0	0	1	1	1	0
Dim'd prefix ORs:	0	0	0	1	1	1	1	1	1	1	1
Complement:	1	1	1	0	0	0	0	0	0	0	0
AND with data:	0	0	1	0	0	0	0	0	0	0	0

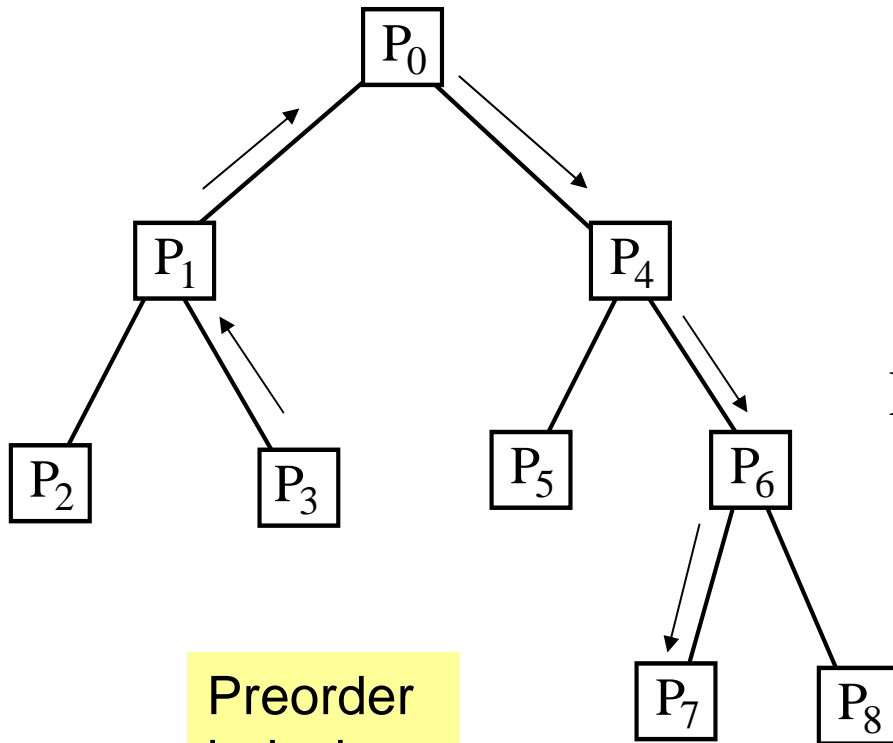
## Carry-lookahead network:

$p \not\subset x = x$   
 $a \not\subset x = a$   
 $g \not\subset x = g$

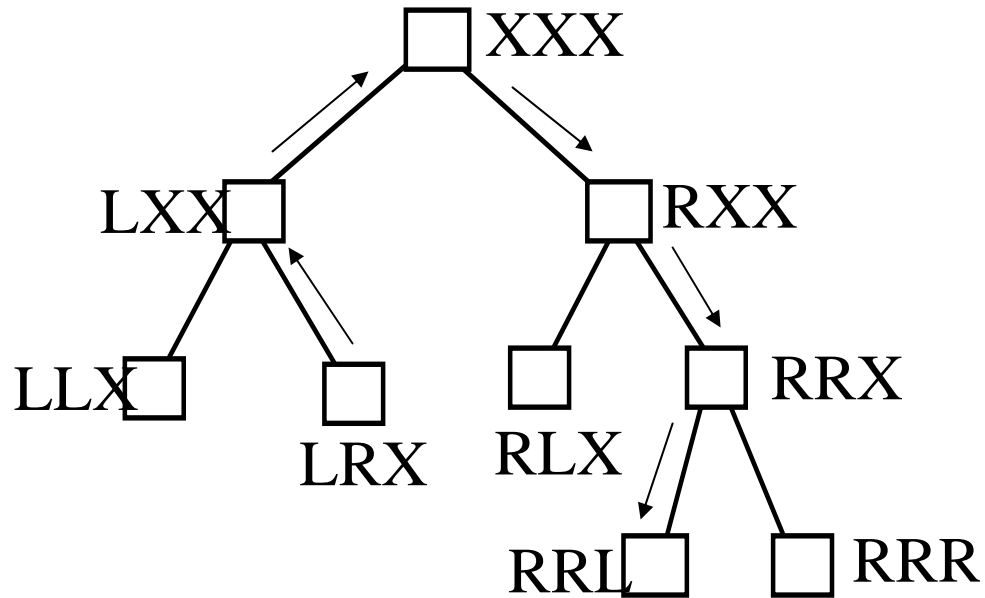




# Binary Tree Packet Routing



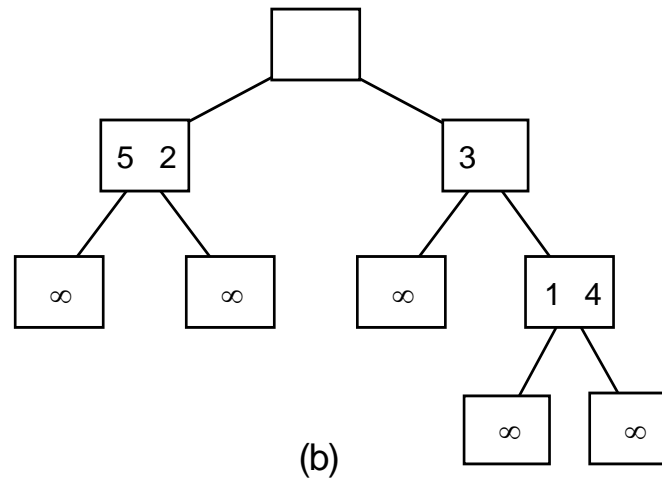
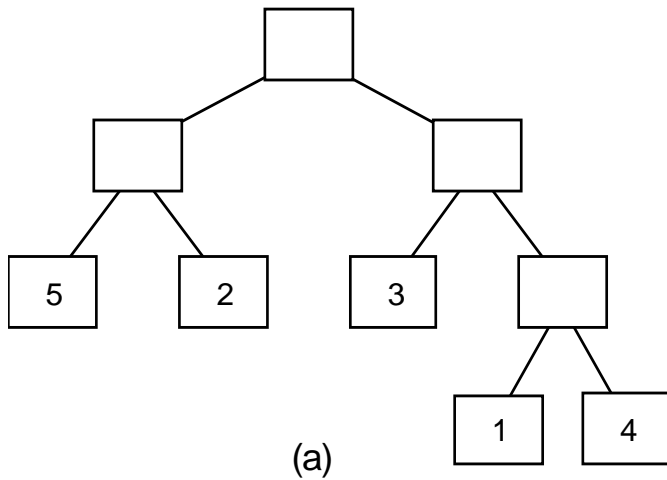
Preorder  
indexing



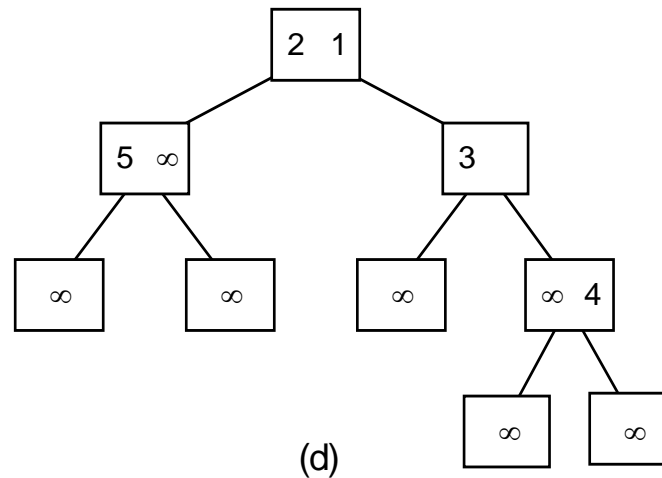
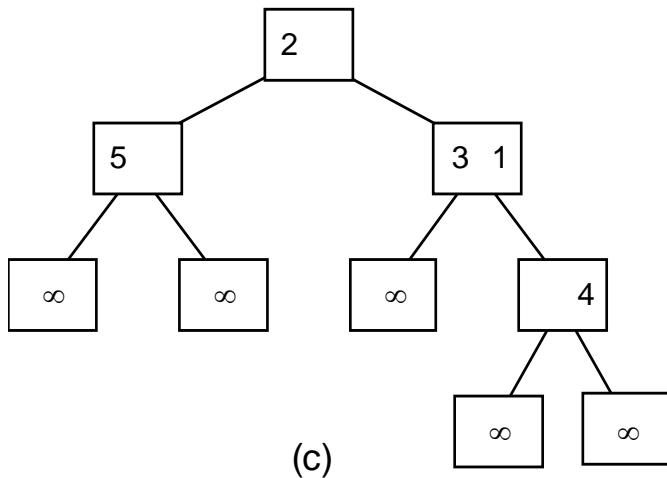
Node index is a representation  
of the path from the tree root

Packet routing on a binary tree with two indexing schemes.

# Binary Tree Sorting



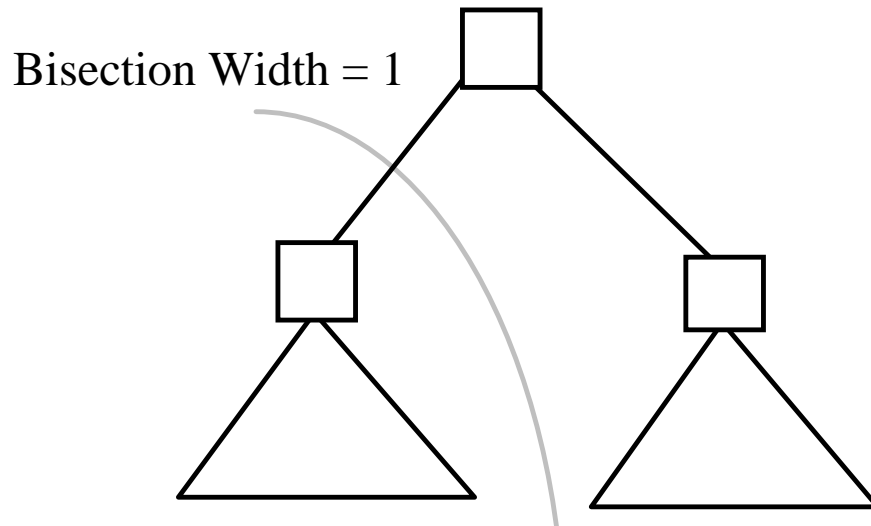
Small values  
“bubble up,”  
causing the  
root to “see”  
the values in  
ascending  
order



Linear-time  
sorting (no  
better than  
linear array)

Fig. 2.12 The first few steps of the sorting algorithm on a binary tree.

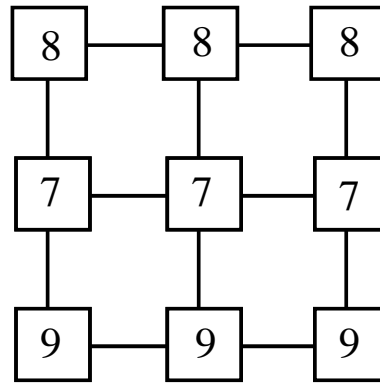
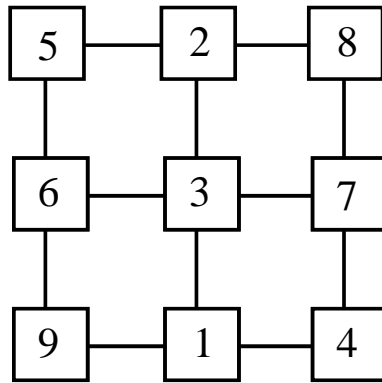
# The Bisection-Width Bottleneck in a Binary Tree



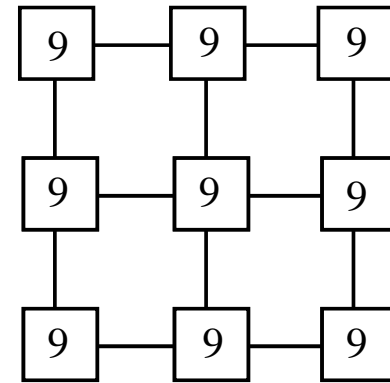
Linear-time  
sorting is the  
best possible  
due to  $B = 1$

Fig. 2.13 The bisection width of a binary tree architecture.

## 2.5 Algorithms for a 2D Mesh

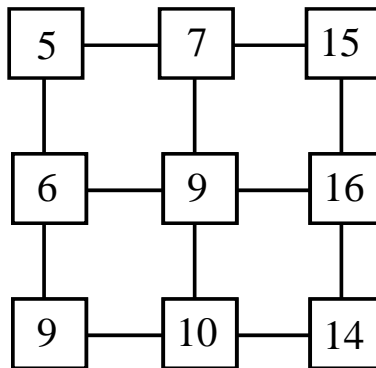


Row maximums

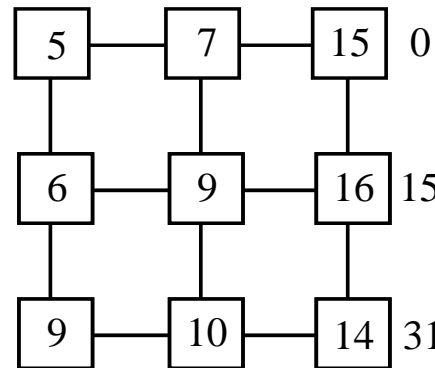


Column maximums

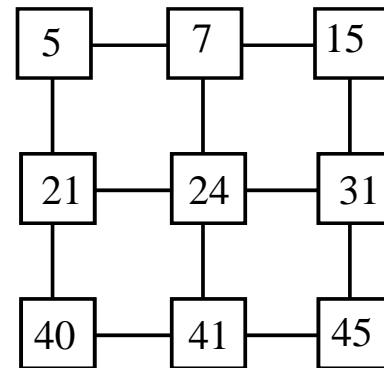
Finding the max value on a 2D mesh.



Row prefix sums



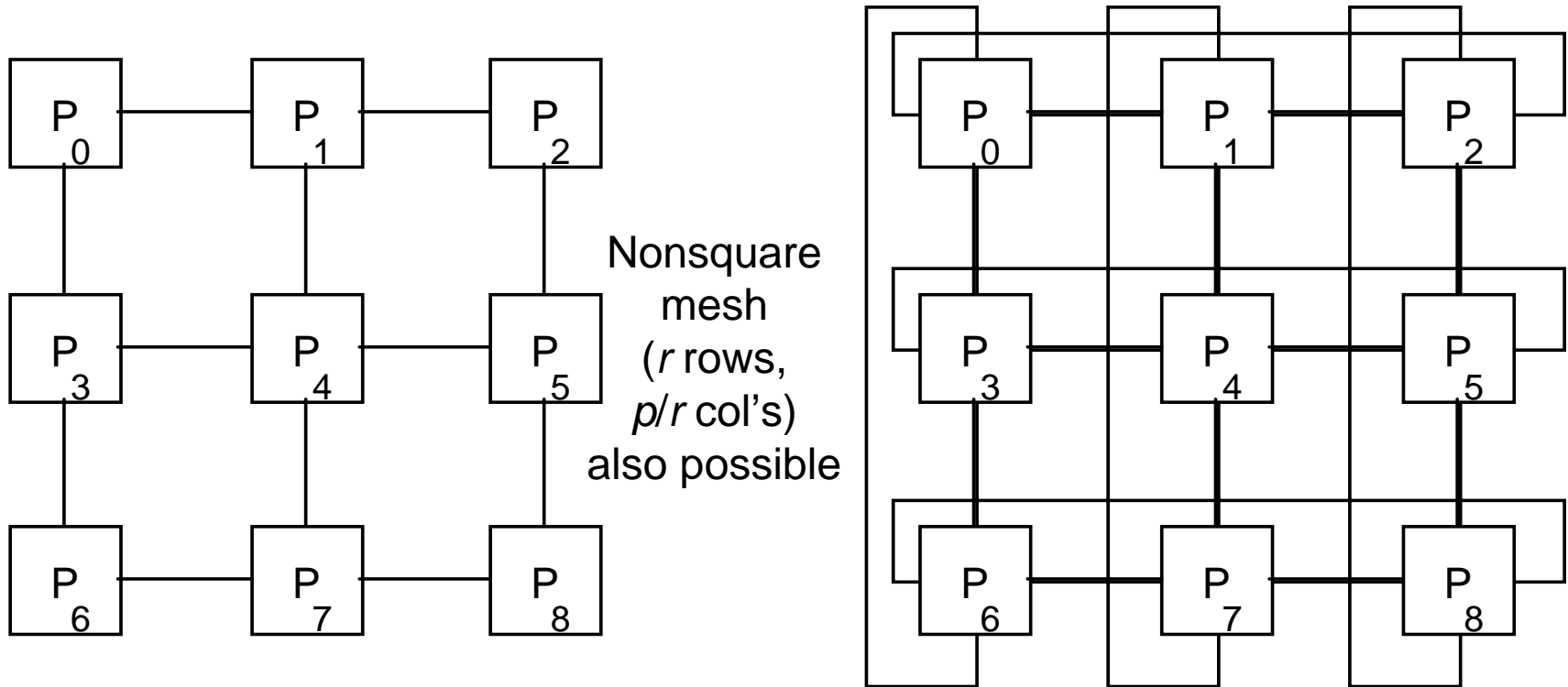
Diminished prefix sums in last column



Broadcast in rows and combine

Computing prefix sums on a 2D mesh

# Routing and Broadcasting on a 2D Mesh



**Routing:** Send along the row to the correct column; route in column

**Broadcasting:** Broadcast in row; then broadcast in all column

Routing and broadcasting on a 9-processors 2D mesh or torus

# Sorting on a 2D Mesh Using Shearsort

5	2	8
6	3	7
9	1	4

Initial values

Number of iterations =  $\log_2 \sqrt{p}$

Compare-exchange steps in each iteration =  $2\sqrt{p}$

Total steps =  $(\log_2 p + 1) \sqrt{p}$

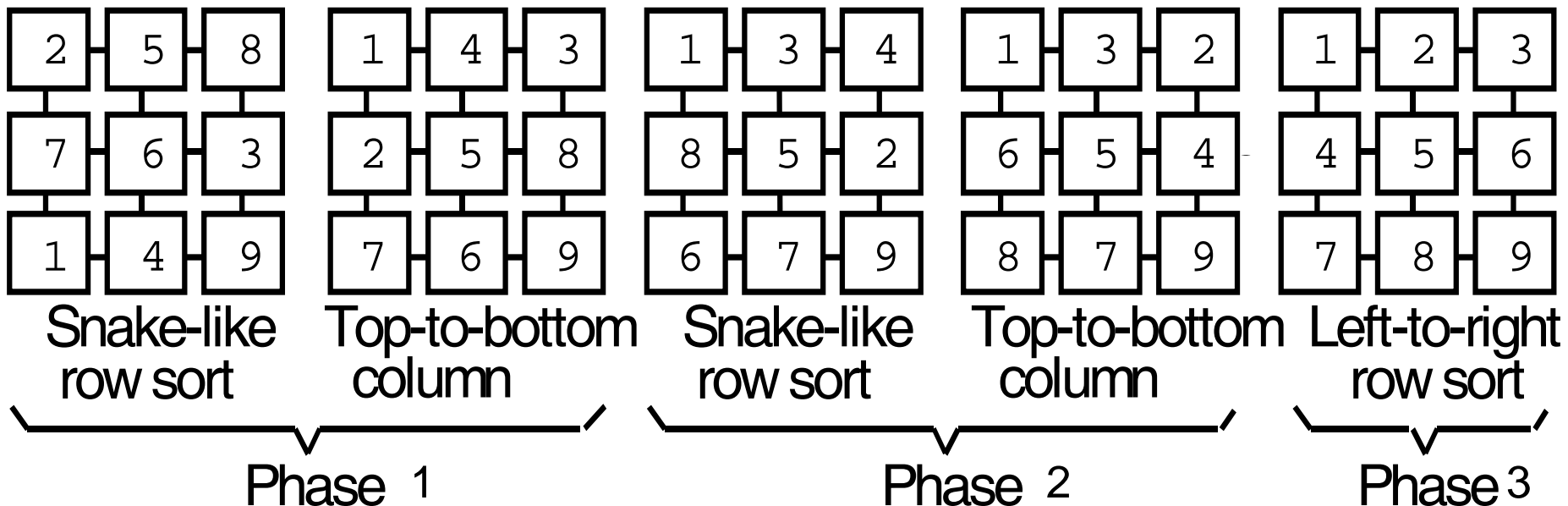
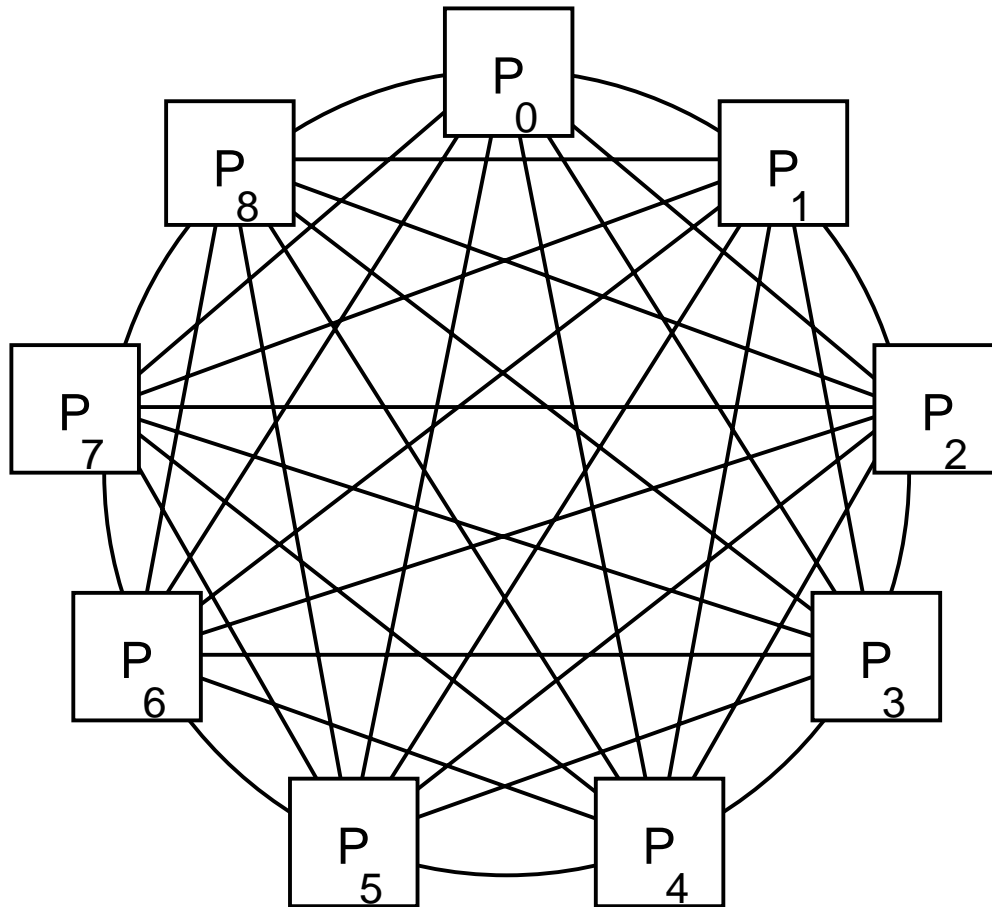


Fig. 2.14 The shearsort algorithm on a  $3 \times 3$  mesh.

## 2.6 Algorithms with Shared Variables



### **Semigroup computation:**

Each processor can perform the computation locally

### **Parallel prefix computation:**

Same as semigroup, except only data from smaller-index processors are combined

### **Packet routing:** Trivial

**Broadcasting:** One step with all-port ( $p - 1$  steps with single-port) communication

**Sorting:** Each processor determines the rank of its data element; followed by routing

# 3 Parallel Algorithm Complexity

Review algorithm complexity and various complexity classes:

- Introduce the notions of time and time/cost optimality
- Derive tools for analysis, comparison, and fine-tuning

## Topics in This Chapter

3.1 Asymptotic Complexity

3.2 Algorithms Optimality and Efficiency

3.3 Complexity Classes

3.4 Parallelizable Tasks and the NC Class

3.5 Parallel Programming Paradigms

3.6 Solving Recurrences



# 3.1 Asymptotic Complexity

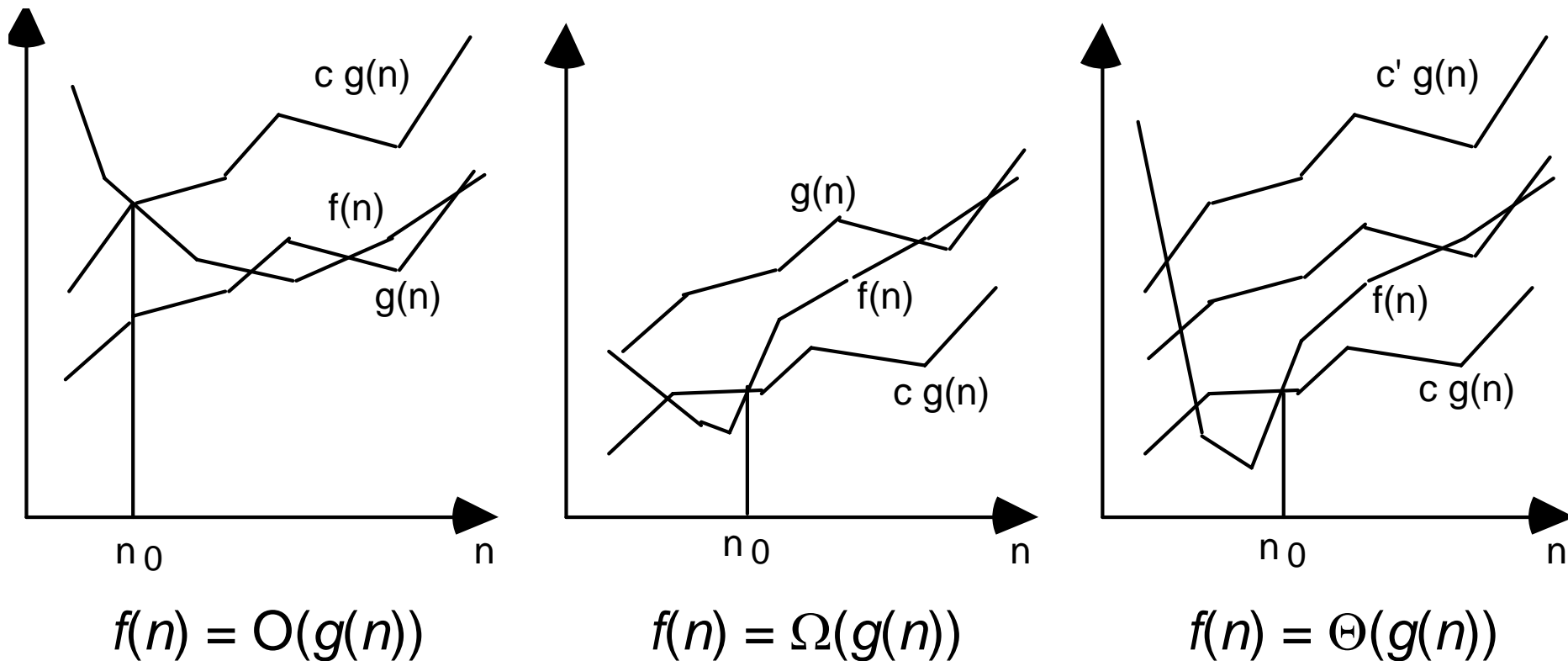


Fig. 3.1 Graphical representation of the notions of asymptotic complexity.

$$3n \log n = O(n^2)$$

$$\frac{1}{2} n \log^2 n = \Omega(n)$$

$$200 n^2 + n = \Theta(n^2)$$

# Little Oh, Big Oh, and Their Buddies

## **Notation**

$$f(n) = o(g(n))$$

$$f(n) = O(g(n))$$

$$f(n) = \Theta(g(n))$$

$$f(n) = \Omega(g(n))$$

$$f(n) = \omega(g(n))$$

<

≤

=

≥

>

## **Growth rate**

strictly less than

no greater than

the same as

no less than

strictly greater than

## **Example of use**

$$T(n) = cn^2 + o(n^2)$$

$$T(n, m) = O(n \log n + m)$$

$$T(n) = \Theta(n \log n)$$

$$T(n) = \Omega(\sqrt{n})$$

$$T(n) = \omega(\log n)$$

# Growth Rates for Typical Functions

Table 3.1 Comparing the Growth Rates of Sublinear and Superlinear Functions (K = 1000, M = 1 000 000).

Sublinear		Linear	Superlinear	
$\log^2 n$	$n^{1/2}$	$n$	$n \log^2 n$	$n^{3/2}$
-----	-----	-----	-----	-----
9	3	10	90	30
36	10	100	3.6 K	1 K
81	31	1 K	81 K	31 K
169	100	10 K	1.7 M	1 M
256	316	100 K	26 M	31 M
361	1 K	1 M	361 M	1000 M

$n$	$(n/4) \log^2 n$	$n \log^2 n$	$100 n^{1/2}$	$n^{3/2}$
-----	-----	-----	-----	-----
10	20 s	2 min	5 min	30 s
100	15 min	1 hr	15 min	15 min
1 K	6 hr	1 day	1 hr	9 hr
10 K	5 day	20 day	3 hr	10 day
100 K	2 mo	1 yr	9 hr	1 yr
1 M	3 yr	11 yr	1 day	32 yr

Table 3.3 Effect of Constants on the Growth Rates of Running Times Using Larger Time Units and Round Figures.

Warning: Table 3.3 in text needs corrections.

# Some Commonly Encountered Growth Rates

<b>Notation</b>	<b>Class name</b>	<b>Notes</b>
$O(1)$	Constant	Rarely practical
$O(\log \log n)$	Double-logarithmic	Sublogarithmic
$O(\log n)$	Logarithmic	
$O(\log^k n)$	Polylogarithmic	$k$ is a constant
$O(n^a), a < 1$		e.g., $O(n^{1/2})$ or $O(n^{1-\varepsilon})$
$O(n/\log^k n)$		Still sublinear
<hr/>		
$O(n)$	Linear	
<hr/>		
$O(n \log^k n)$		Superlinear
$O(n^c), c > 1$	Polynomial	e.g., $O(n^{1+\varepsilon})$ or $O(n^{3/2})$
$O(2^n)$	Exponential	Generally intractable
$O(2^{2^n})$	Double-exponential	Hopeless!

## 3.2 Algorithm Optimality and Efficiency

**Lower bounds:** Theoretical arguments based on bisection width, and the like

**Upper bounds:** Deriving/analyzing algorithms and proving them correct

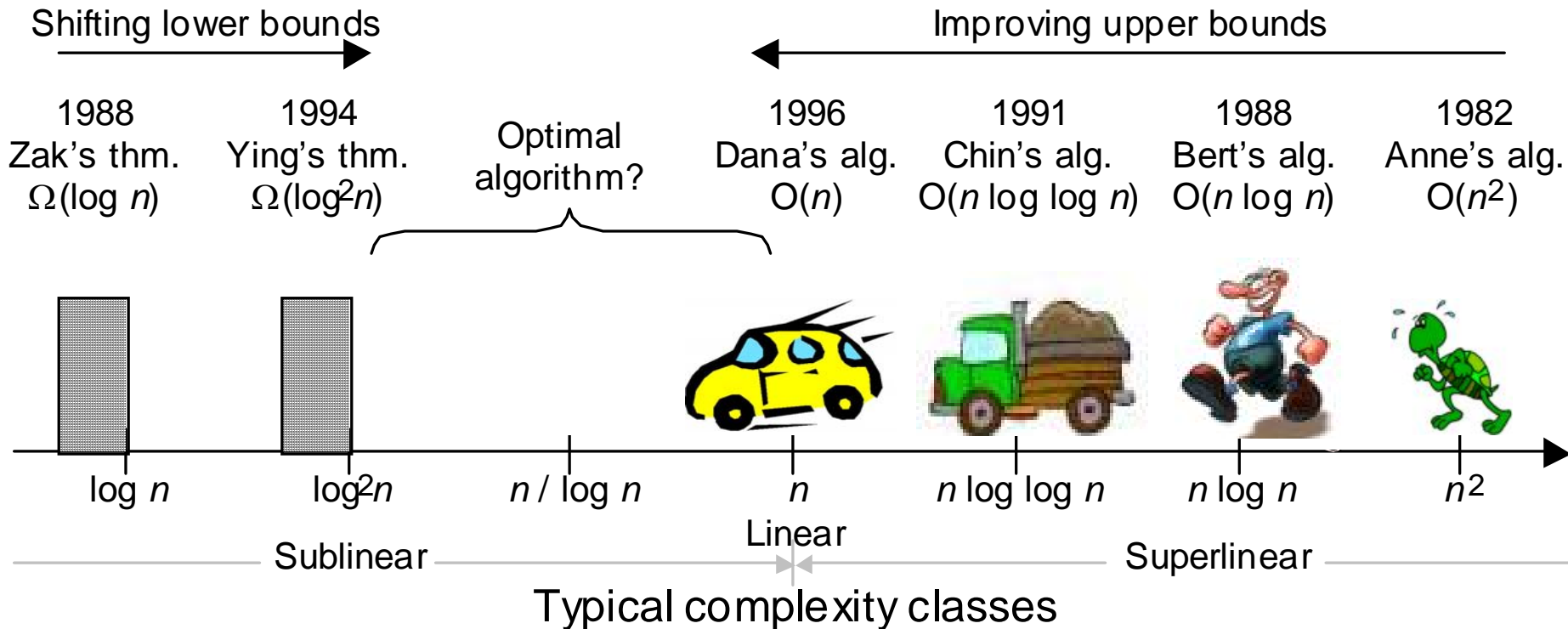


Fig. 3.2 Upper and lower bounds may tighten over time.

# Complexity History of Some Real Problems

Examples from the book *Algorithmic Graph Theory and Perfect Graphs* [GOLU04]:  
Complexity of determining whether an  $n$ -vertex graph is planar

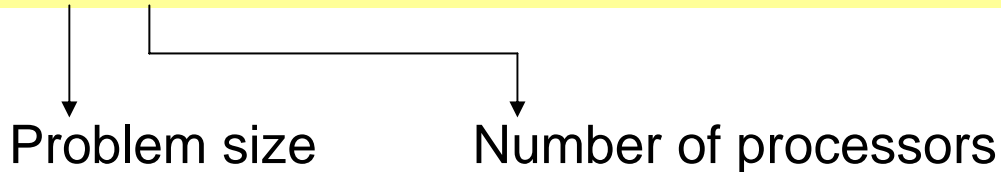
Exponential	Kuratowski	1930
$O(n^3)$	Auslander and Porter	1961
	Goldstein	1963
	Shirey	1969
$O(n^2)$	Lempel, Even, and Cederbaum	1967
$O(n \log n)$	Hopcroft and Tarjan	1972
$O(n)$	Hopcroft and Tarjan	1974
	Booth and Leuker	1976

A second, more complex example: Max network flow,  $n$  vertices,  $e$  edges:  
 $ne^2 \rightarrow n^2e \rightarrow n^3 \rightarrow n^2e^{1/2} \rightarrow n^{5/3}e^{2/3} \rightarrow ne \log^2 n \rightarrow ne \log(n^2/e)$   
 $\rightarrow ne + n^{2+\varepsilon} \rightarrow ne \log_{e/(n \log n)} n \rightarrow ne \log_{e/n} n + n^2 \log^{2+\varepsilon} n$

# Some Notions of Algorithm Optimality

## Time optimality (*optimal algorithm, for short*)

$T(n, p) = g(n, p)$ , where  $g(n, p)$  is an established lower bound



## Cost-time optimality (*cost-optimal algorithm, for short*)

$pT(n, p) = T(n, 1)$ ; i.e., redundancy = utilization = 1

## Cost-time efficiency (*efficient algorithm, for short*)

$pT(n, p) = \Theta(T(n, 1))$ ; i.e., redundancy = utilization =  $\Theta(1)$

# Beware of Comparing Step Counts

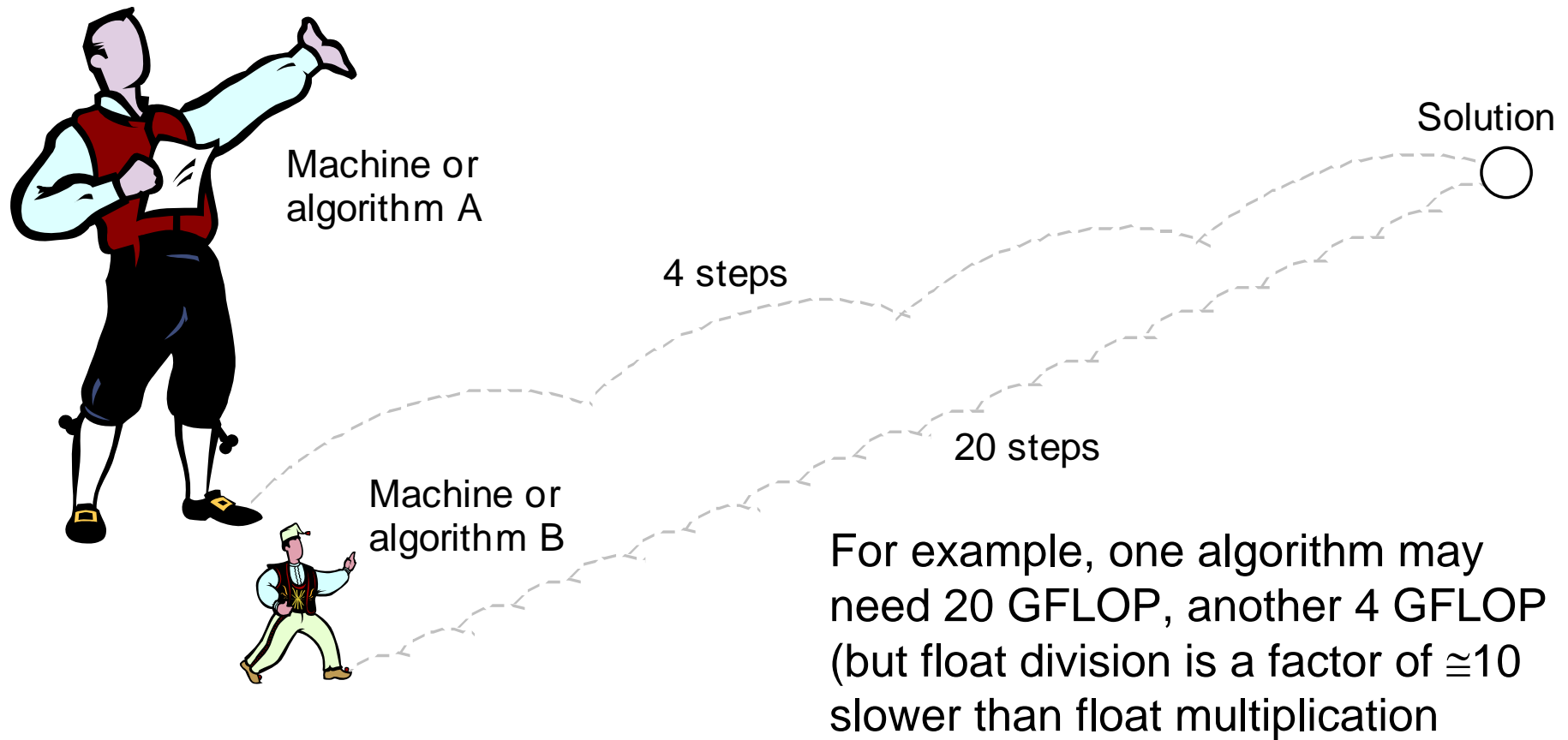
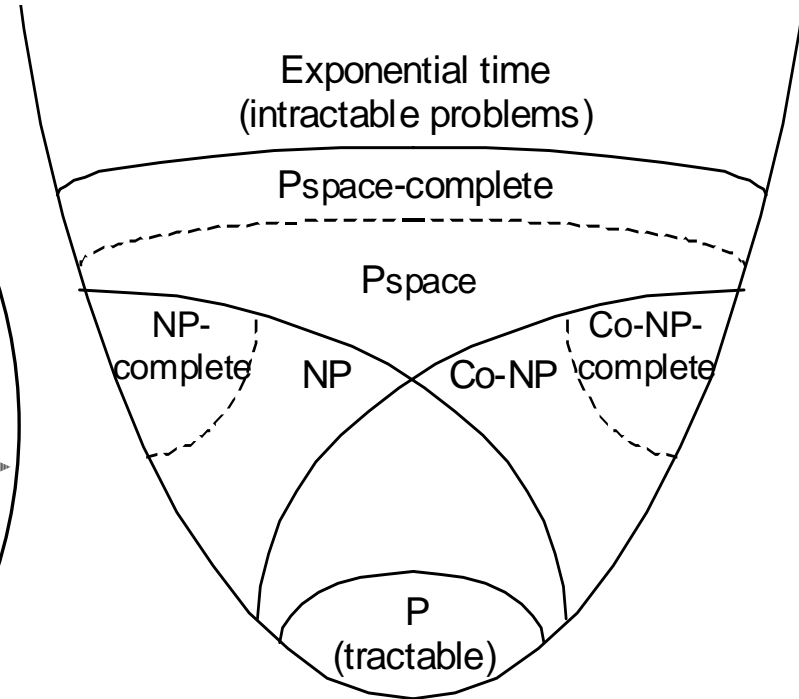
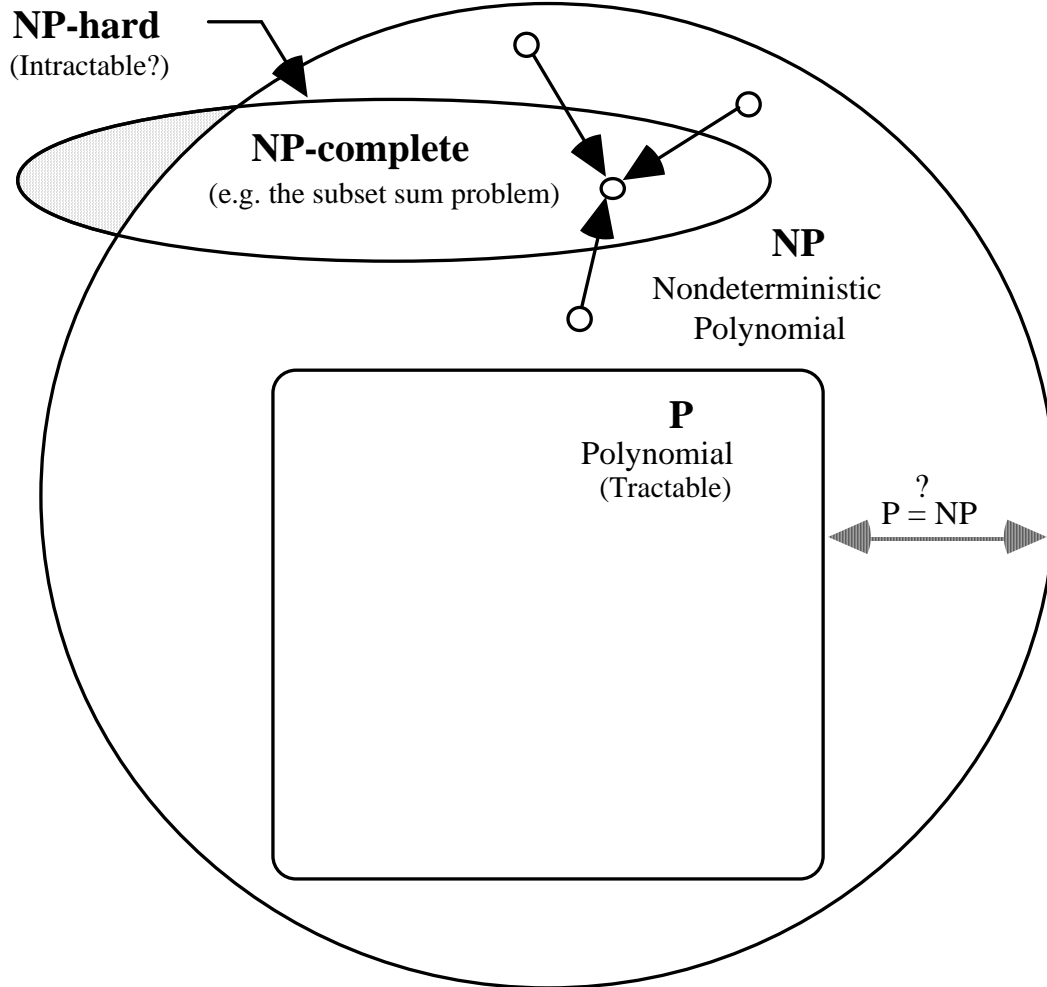


Fig. 3.2 Five times fewer steps does not necessarily mean five times faster.



# 3.3 Complexity Classes



A more complete view of complexity classes

Conceptual view of the P, NP, NP-complete, and NP-hard classes.

# Some NP-Complete Problems

**Subset sum problem:** Given a set of  $n$  integers and a target sum  $s$ , determine if a subset of the integers adds up to  $s$ .

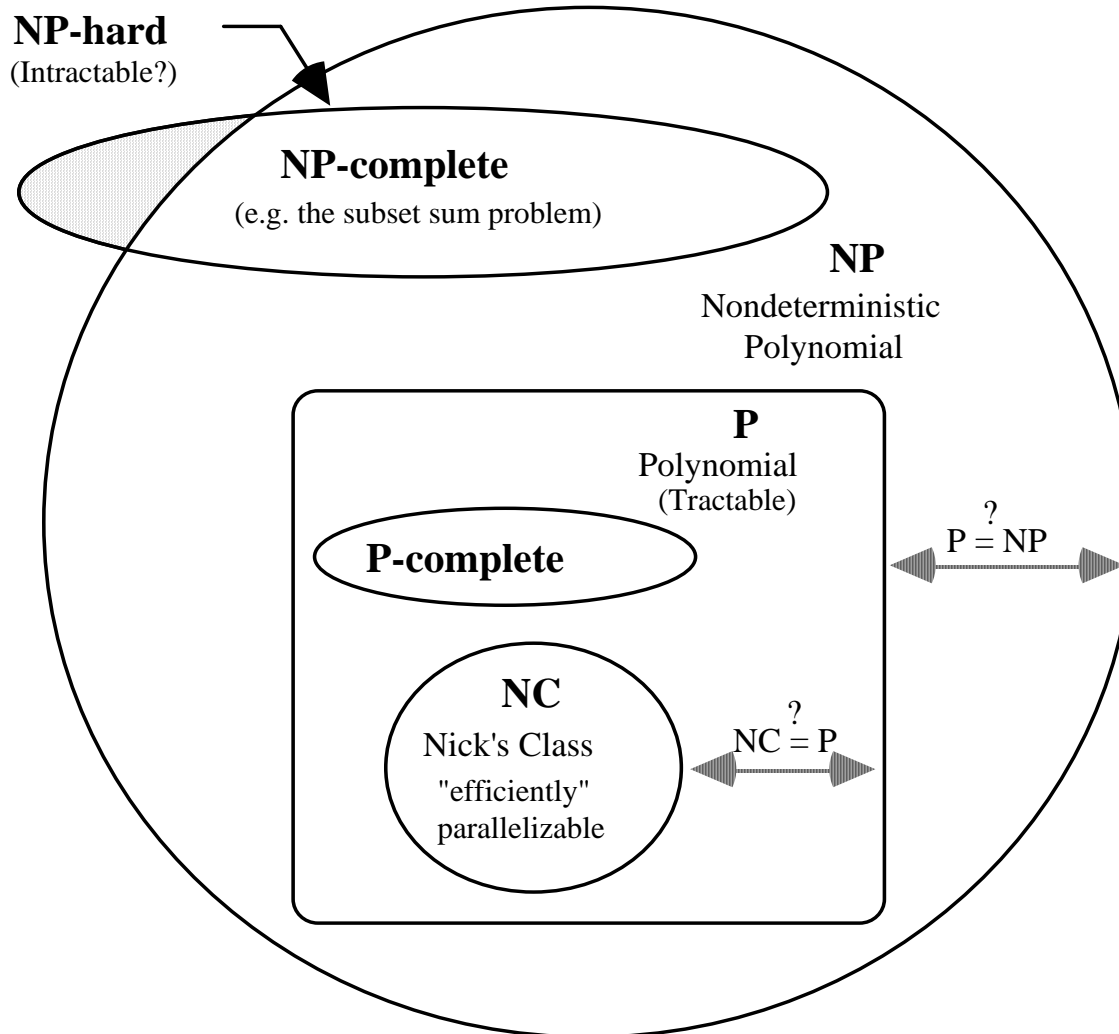
**Satisfiability:** Is there an assignment of values to variables in a product-of-sums Boolean expression that makes it true?  
(Is in NP even if each OR term is restricted to have exactly three literals)

**Circuit satisfiability:** Is there an assignment of 0s and 1s to inputs of a logic circuit that would make the circuit output 1?

**Hamiltonian cycle:** Does an arbitrary graph contain a cycle that goes through all of its nodes?

**Traveling salesman:** Find a lowest-cost or shortest-distance tour of a number of cities, given travel costs or distances.

# 3.4 Parallelizable Tasks and the NC Class



**NC (Nick's class):**  
Subset of problems in P for which there exist parallel algorithms using  $p = n^c$  processors (polynomially many) that run in  $O(\log^k n)$  time (polylog time).

**P-complete problem:**  
Given a logic circuit with known inputs, determine its output (*circuit value prob.*).

Fig. 3.4 A conceptual view of complexity classes and their relationships.

# 3.5 Parallel Programming Paradigms

## Divide and conquer

Decompose problem of size  $n$  into smaller problems; solve subproblems independently; combine subproblem results into final answer

$$T(n) = \underbrace{T_d(n)}_{\text{Decompose}} + \underbrace{T_s}_{\text{Solve in parallel}} + \underbrace{T_c(n)}_{\text{Combine}}$$

## Randomization

When it is impossible or difficult to decompose a large problem into subproblems with equal solution times, one might use random decisions that lead to good results with very high probability.

*Example:* sorting with random sampling

Other forms: Random search, control randomization, symmetry breaking

## Approximation

Iterative numerical methods may use approximation to arrive at solution(s).

*Example:* Solving linear systems using Jacobi relaxation.

Under proper conditions, the iterations converge to the correct solutions; more iterations  $\Rightarrow$  greater accuracy

## 3.6 Solving Recurrences

$$\begin{aligned} f(n) &= f(n-1) + n && \{\text{rewrite } f(n-1) \text{ as } f((n-1)-1) + (n-1)\} \\ &= f(n-2) + n-1 + n \\ &= f(n-3) + n-2 + n-1 + n \\ &\dots \\ &= f(1) + 2 + 3 + \dots + n-1 + n \\ &= n(n+1)/2 - 1 = \Theta(n^2) \end{aligned}$$

**This method is  
known as unrolling**

$$\begin{aligned} f(n) &= f(n/2) + 1 && \{\text{rewrite } f(n/2) \text{ as } f((n/2)/2 + 1)\} \\ &= f(n/4) + 1 + 1 \\ &= f(n/8) + 1 + 1 + 1 \\ &\dots \\ &= f(n/n) + 1 + 1 + 1 + \dots + 1 \\ &\quad \text{----- } \log_2 n \text{ times -----} \\ &= \log_2 n = \Theta(\log n) \end{aligned}$$

# More Example of Recurrence Unrolling

$$\begin{aligned}f(n) &= 2f(n/2) + 1 \\&= 4f(n/4) + 2 + 1 \\&= 8f(n/8) + 4 + 2 + 1 \\&\dots \\&= n f(n/n) + n/2 + \dots + 4 + 2 + 1 \\&= n - 1 = \Theta(n)\end{aligned}$$

$$\begin{aligned}f(n) &= f(n/2) + n \\&= f(n/4) + n/2 + n \\&= f(n/8) + n/4 + n/2 + n \\&\dots \\&= f(n/n) + 2 + 4 + \dots + n/4 + n/2 + n \\&= 2n - 2 = \Theta(n)\end{aligned}$$

## **Solution via guessing:**

Guess  $f(n) = \Theta(n) = cn + g(n)$   
 $cn + g(n) = cn/2 + g(n/2) + n$   
Thus,  $c = 2$  and  $g(n) = g(n/2)$

# Still More Examples of Unrolling

$$\begin{aligned}
 f(n) &= 2f(n/2) + n \\
 &= 4f(n/4) + n + n \\
 &= 8f(n/8) + n + n + n \\
 &\dots \\
 &= n f(n/n) + \underbrace{n + n + n + \dots + n}_{\log_2 n \text{ times}} \\
 &= n \log_2 n = \Theta(n \log n)
 \end{aligned}$$

**Alternate solution method:**

$$f(n)/n = f(n/2)/(n/2) + 1$$

$$\text{Let } f(n)/n = g(n)$$

$$g(n) = g(n/2) + 1 = \log_2 n$$

$$\begin{aligned}
 f(n) &= f(n/2) + \log_2 n \\
 &= f(n/4) + \log_2(n/2) + \log_2 n \\
 &= f(n/8) + \log_2(n/4) + \log_2(n/2) + \log_2 n \\
 &\dots \\
 &= f(n/n) + \log_2 2 + \log_2 4 + \dots + \log_2(n/2) + \log_2 n \\
 &= 1 + 2 + 3 + \dots + \log_2 n \\
 &= \log_2 n (\log_2 n + 1)/2 = \Theta(\log^2 n)
 \end{aligned}$$

# Master Theorem for Recurrences

## Theorem 3.1:

Given  $f(n) = a f(n/b) + h(n)$ ;  $a, b$  constant,  $h$  arbitrary function  
the asymptotic solution to the recurrence is ( $c = \log_b a$ )

$$f(n) = \Theta(n^c) \quad \text{if } h(n) = O(n^{c-\varepsilon}) \text{ for some } \varepsilon > 0$$

$$f(n) = \Theta(n^c \log n) \quad \text{if } h(n) = \Theta(n^c)$$

$$f(n) = \Theta(h(n)) \quad \text{if } h(n) = \Omega(n^{c+\varepsilon}) \text{ for some } \varepsilon > 0$$

### Example:

$$f(n) = 2f(n/2) + 1$$

$$a = b = 2; c = \log_b a = 1$$

$$h(n) = 1 = O(n^{1-\varepsilon})$$

$$f(n) = \Theta(n^c) = \Theta(n)$$



# Intuition Behind the Master Theorem

## Theorem 3.1:

Given  $f(n) = a f(n/b) + h(n)$ ;  $a, b$  constant,  $h$  arbitrary function  
the asymptotic solution to the recurrence is ( $c = \log_b a$ )

$$f(n) = \Theta(n^c) \quad \text{if } h(n) = O(n^{c-\varepsilon}) \text{ for some } \varepsilon > 0$$

$$\begin{aligned} f(n) &= 2f(n/2) + 1 = 4f(n/4) + 2 + 1 = \dots \\ &= n f(n/n) + n/2 + \dots + 4 + 2 + 1 \end{aligned}$$

**The last term  
dominates**

$$f(n) = \Theta(n^c \log n) \quad \text{if } h(n) = \Theta(n^c)$$

$$\begin{aligned} f(n) &= 2f(n/2) + n = 4f(n/4) + n + n = \dots \\ &= n f(n/n) + n + n + n + \dots + n \end{aligned}$$

**All terms are  
comparable**

$$f(n) = \Theta(h(n)) \quad \text{if } h(n) = \Omega(n^{c+\varepsilon}) \text{ for some } \varepsilon > 0$$

$$\begin{aligned} f(n) &= f(n/2) + n = f(n/4) + n/2 + n = \dots \\ &= f(n/n) + 2 + 4 + \dots + n/4 + n/2 + n \end{aligned}$$

**The first term  
dominates**

# 4 Models of Parallel Processing

Expand on the taxonomy of parallel processing from Chap. 1:

- Abstract models of shared and distributed memory
- Differences between abstract models and real hardware

## Topics in This Chapter

4.1 Development of Early Models

4.2 SIMD versus MIMD Architectures

4.3 Global versus Distributed Memory

4.4 The PRAM Shared-Memory Model

4.5 Distributed-Memory or Graph Models

4.6 Circuit Model and Physical Realizations

# 4.1 Development of Early Models

Associative memory

Parallel masked search of all words

Bit-serial implementation with RAM

Associative processor

Add more processing logic to PEs

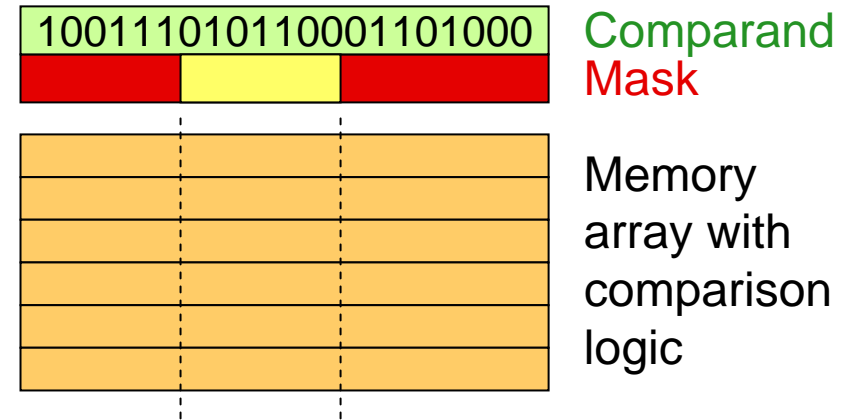


Table 4.1 Entering the second half-century of associative processing

Decade	Events and Advances	Technology	Performance
1940s	Formulation of need & concept	Relays	
1950s	Emergence of cell technologies	Magnetic, Cryogenic	Mega-bit-OPS
1960s	Introduction of basic architectures	Transistors	
1970s	Commercialization & applications	ICs	Giga-bit-OPS
1980s	Focus on system/software issues	VLSI	Tera-bit-OPS
1990s	Scalable & flexible architectures	ULSI, WSI	Peta-bit-OPS

# The Flynn-Johnson Classification Revisited

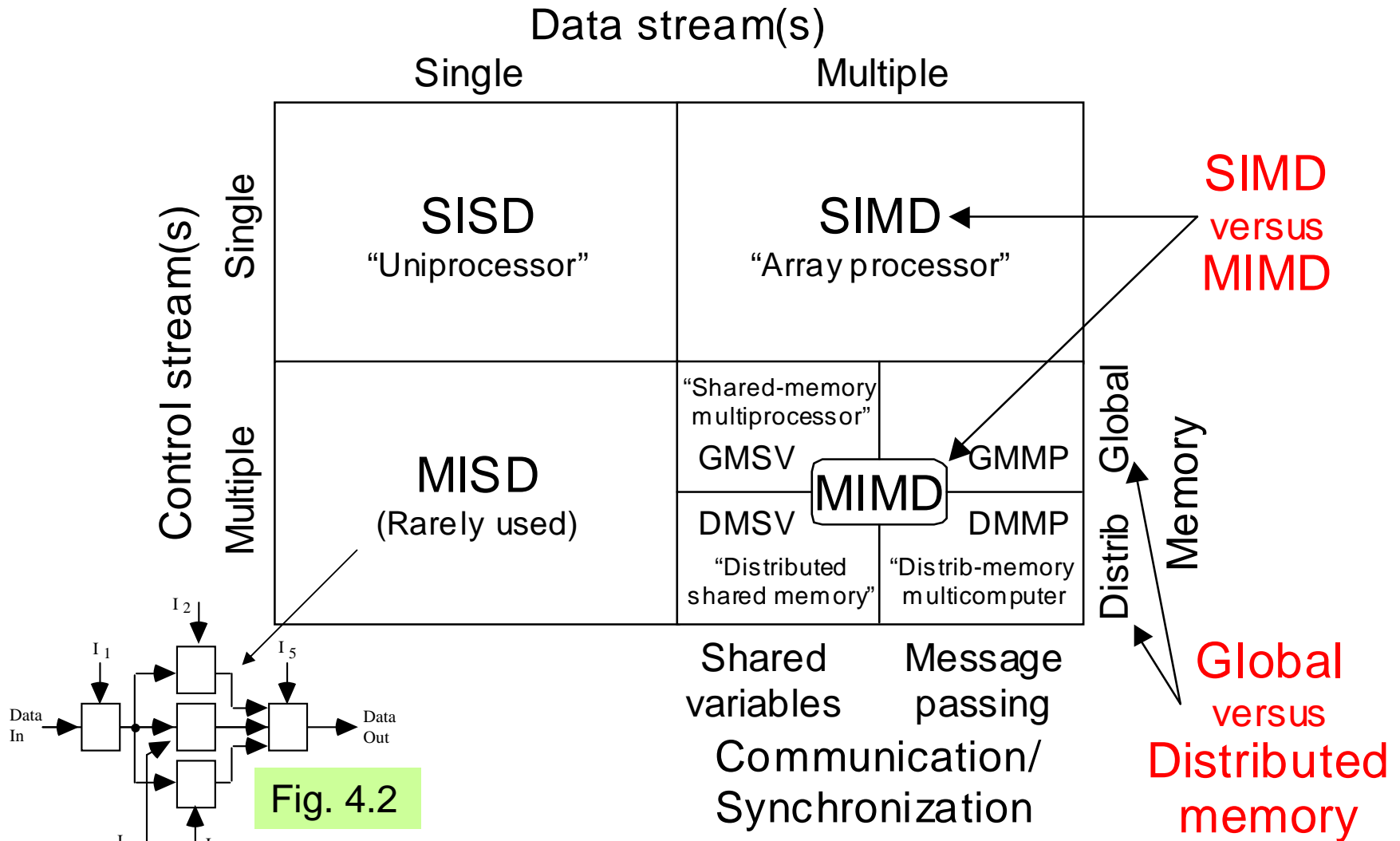


Fig. 4.1 The Flynn-Johnson classification of computer systems.

## 4.2 SIMD versus MIMD Architectures

Most early parallel machines had SIMD designs

- Attractive to have skeleton processors (PEs)

- Eventually, many processors per chip

- High development cost for custom chips, high cost

- MSIMD and SPMD variants

Most modern parallel machines have MIMD designs

- COTS components (CPU chips and switches)

- MPP: Massively or moderately parallel?

- Tightly coupled versus loosely coupled

- Explicit message passing versus shared memory

Network-based NOWs and COWs

- Networks/Clusters of workstations

Grid computing

- Vision: Plug into wall outlets for computing power

### SIMD Timeline

1960	
1970	ILLIAC IV
	DAP
1980	Goodyear MPP
	TMC CM-2
1990	MasPar MP-1
2000	Clearspeed array coproc
2010	

## 4.3 Global versus Distributed Memory

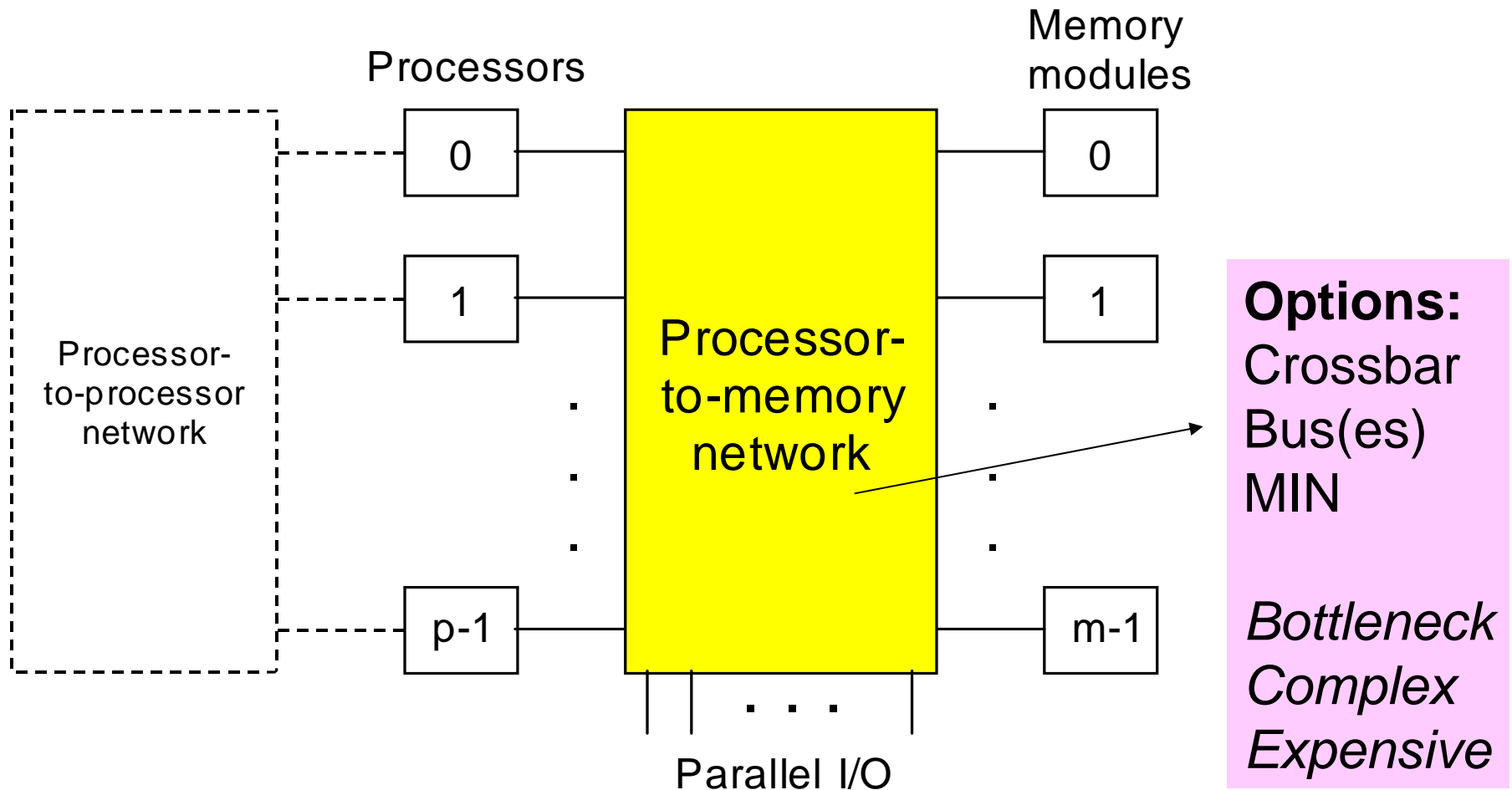


Fig. 4.3 A parallel processor with global memory.

# Removing the Processor-to-Memory Bottleneck

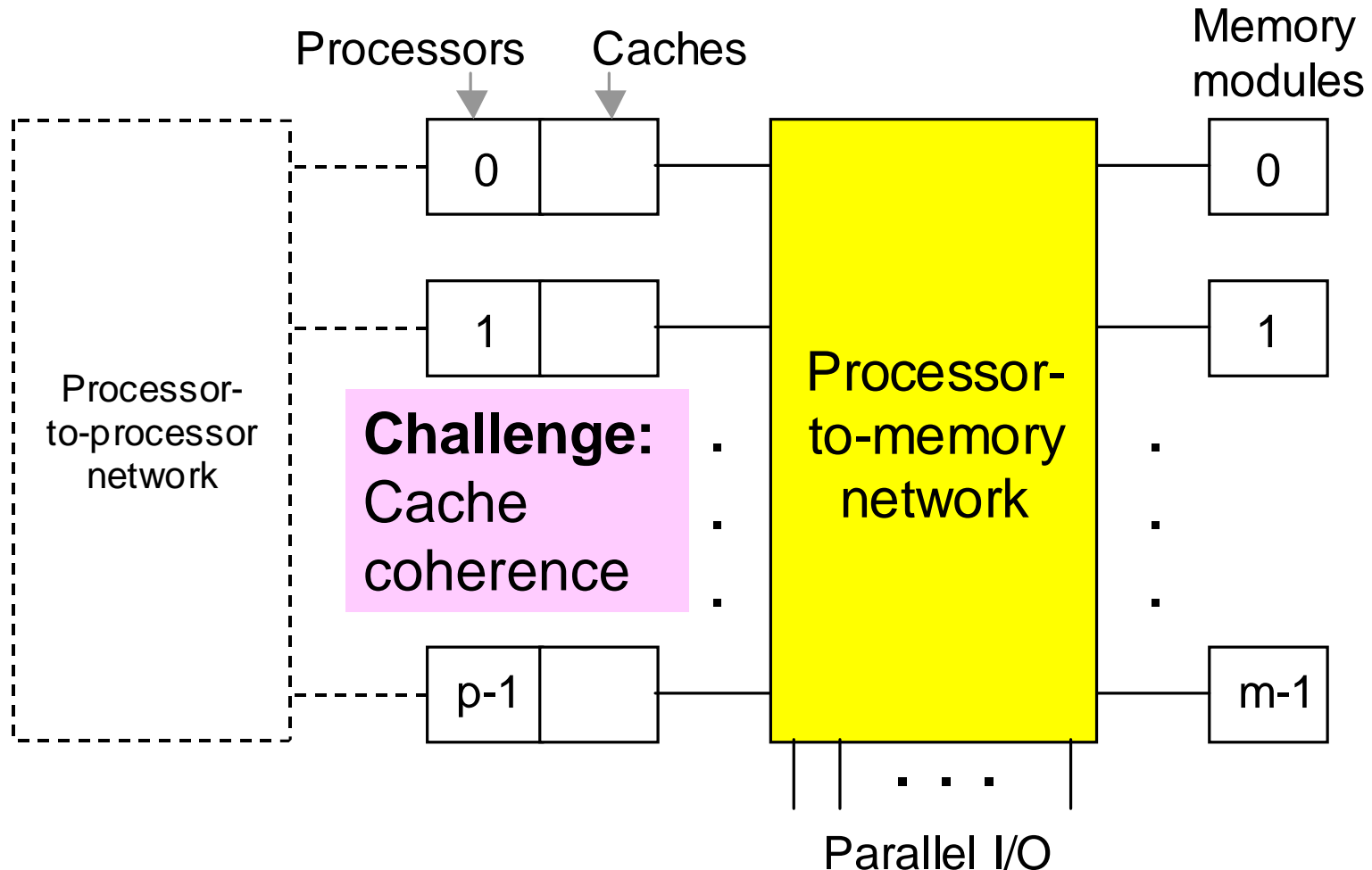
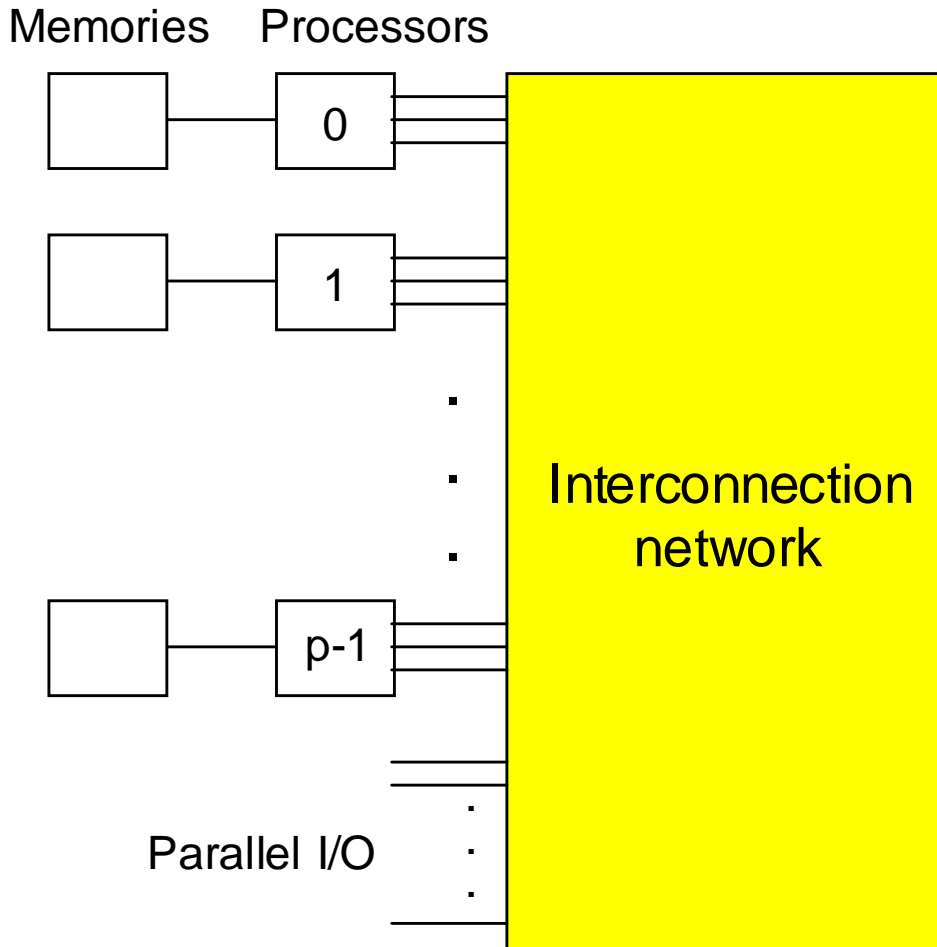


Fig. 4.4 A parallel processor with global memory and processor caches.

# Distributed Shared Memory



## Some Terminology:

NUMA

Nonuniform memory access  
(distributed shared memory)

UMA

Uniform memory access  
(global shared memory)

COMA

Cache-only memory arch

Fig. 4.5 A parallel processor with distributed memory.



## 4.4 The PRAM Shared-Memory Model

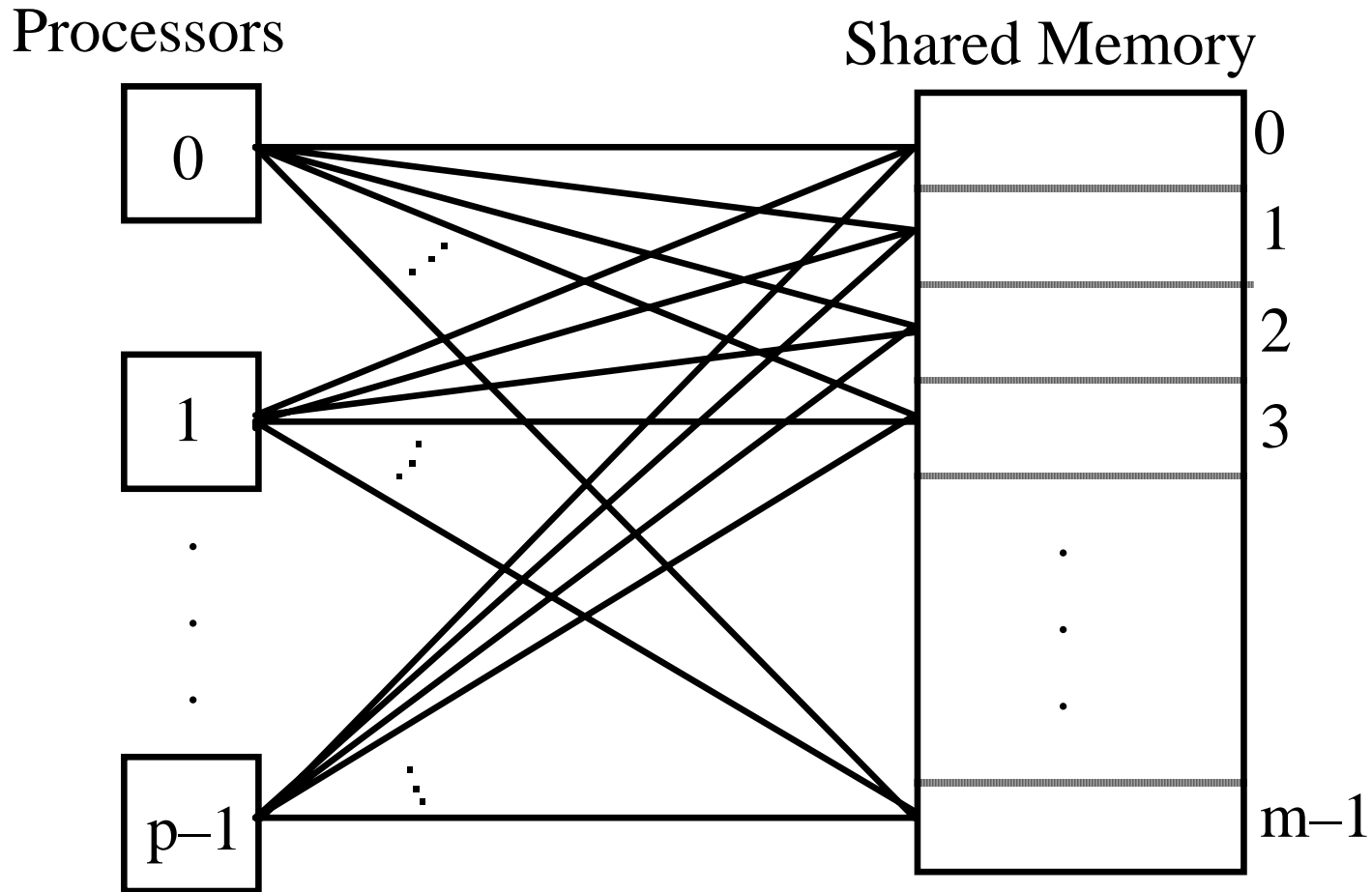
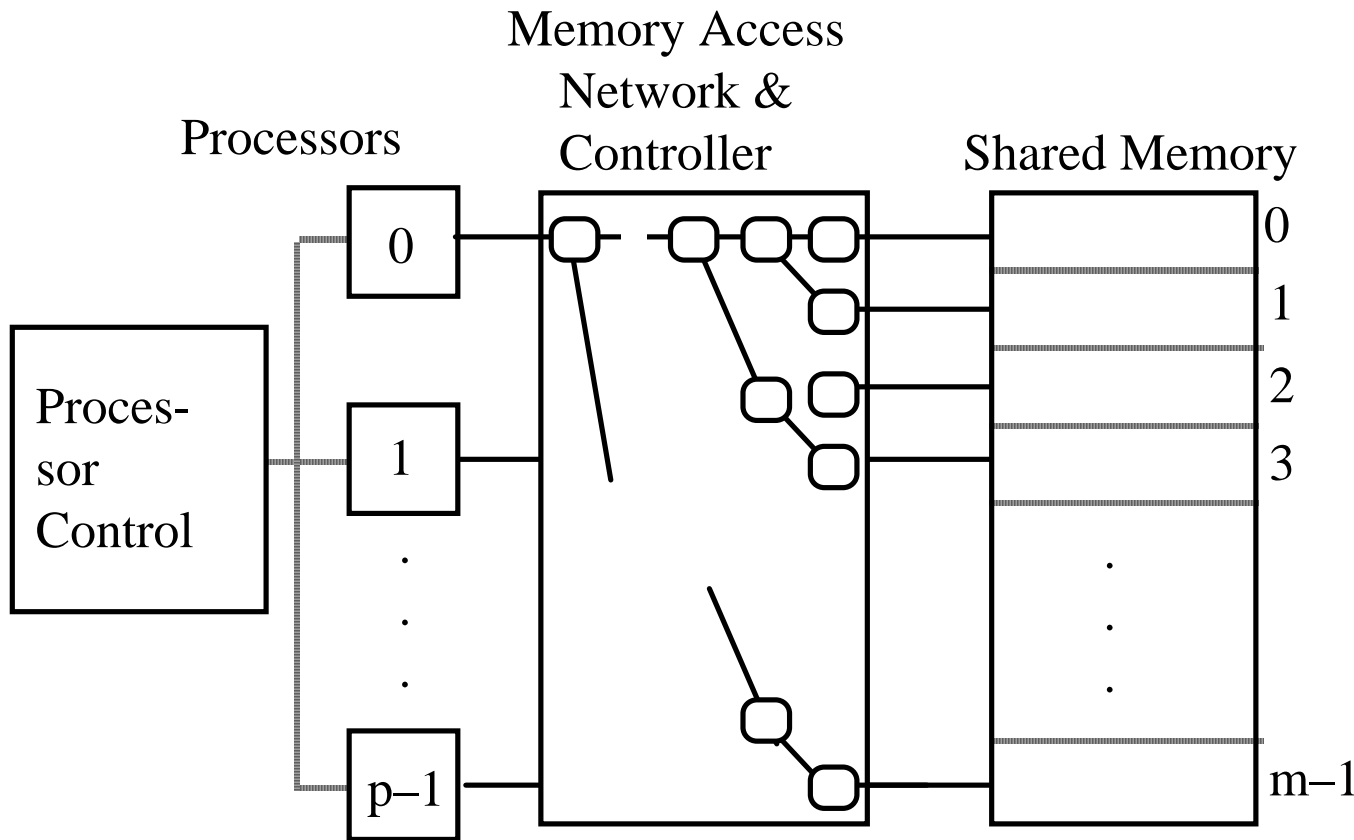


Fig. 4.6 Conceptual view of a parallel random-access machine (PRAM).

# PRAM Implementation and Operation



## PRAM Cycle:

All processors read memory locations of their choosing

All processors compute one step independently

All processors store results into memory locations of their choosing

Fig. 4.7 PRAM with some hardware details shown.

## 4.5 Distributed-Memory or Graph Models

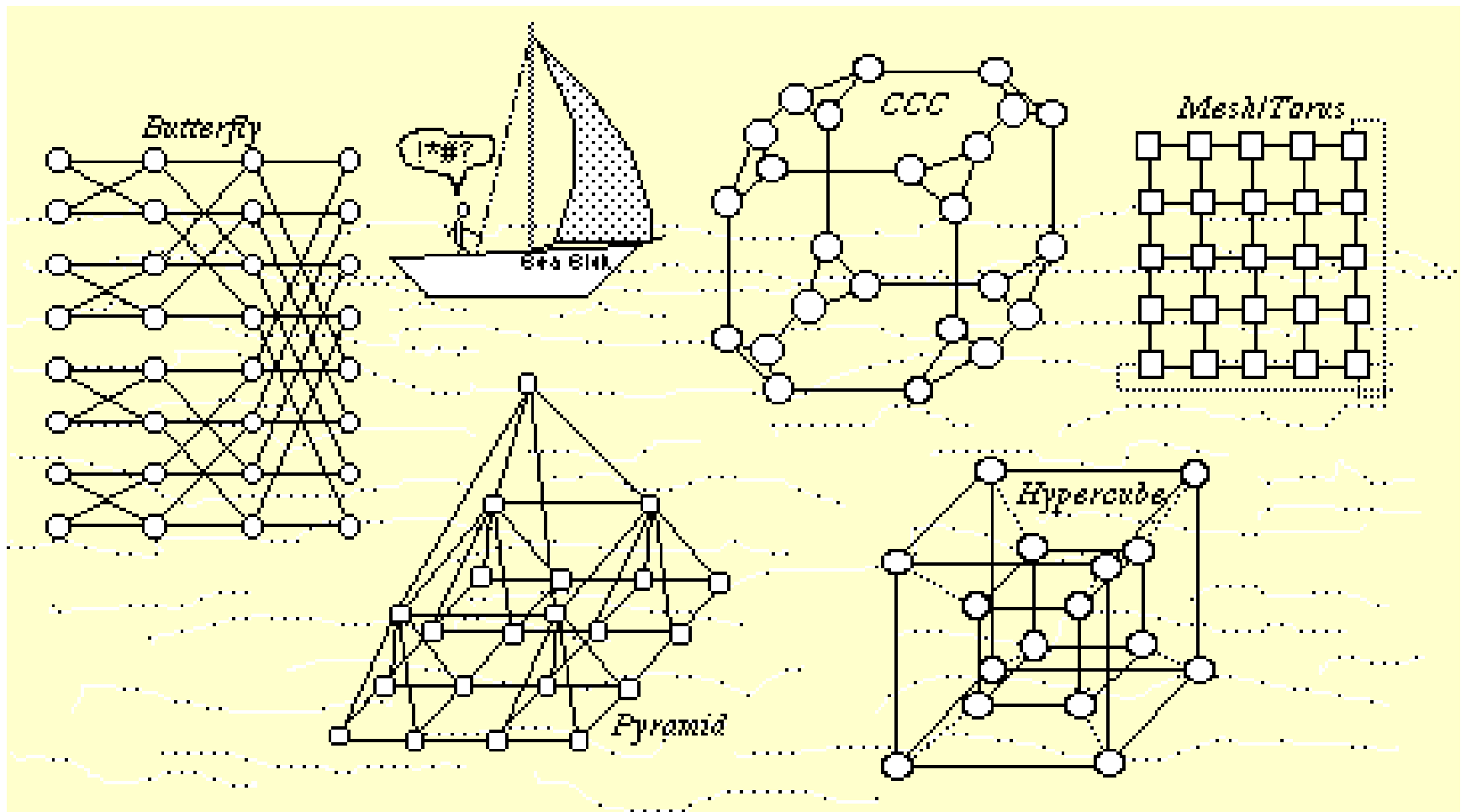


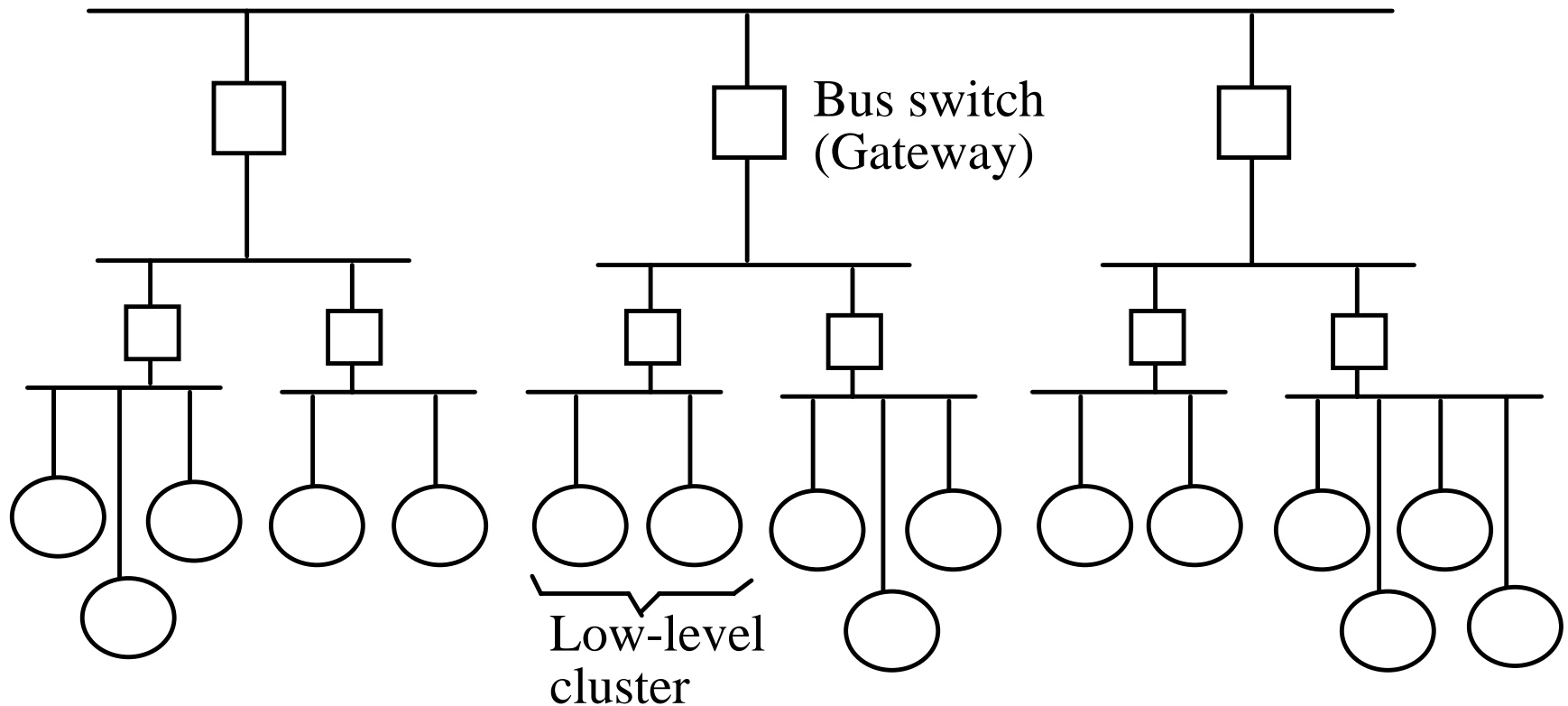
Fig. 4.8 The sea of interconnection networks.

# Some Interconnection Networks (Table 4.2)

Network name(s)	Number of nodes	Network diameter	Bisection width	Node degree	Local links?
1D mesh (linear array)	$k$	$k - 1$	1	2	Yes
1D torus (ring, loop)	$k$	$k/2$	2	2	Yes
2D Mesh	$k^2$	$2k - 2$	$k$	4	Yes
2D torus ( $k$ -ary 2-cube)	$k^2$	$k$	$2k$	4	Yes <sup>1</sup>
3D mesh	$k^3$	$3k - 3$	$k^2$	6	Yes
3D torus ( $k$ -ary 3-cube)	$k^3$	$3k/2$	$2k^2$	6	Yes <sup>1</sup>
Pyramid	$(4k^2 - 1)/3$	$2 \log_2 k$	$2k$	9	No
Binary tree	$2^l - 1$	$2l - 2$	1	3	No
4-ary hypertree	$2^l(2^{l+1} - 1)$	$2l$	$2^{l+1}$	6	No
Butterfly	$2^l(l + 1)$	$2l$	$2^l$	4	No
Hypercube	$2^l$	$l$	$2^{l-1}$	$l$	No
Cube-connected cycles	$2^l l$	$2l$	$2^{l-1}$	3	No
Shuffle-exchange	$2^l$	$2l - 1$	$\geq 2^{l-1}/l$	4 unidir.	No
De Bruijn	$2^l$	$l$	$2^l/l$	4 unidir.	No

<sup>1</sup> With folded layout

## 4.6 Circuit Model and Physical Realizations



Scalability dictates hierarchical connectivity

Fig. 4.9 Example of a hierarchical interconnection architecture.

# Signal Delay on Wires No Longer Negligible

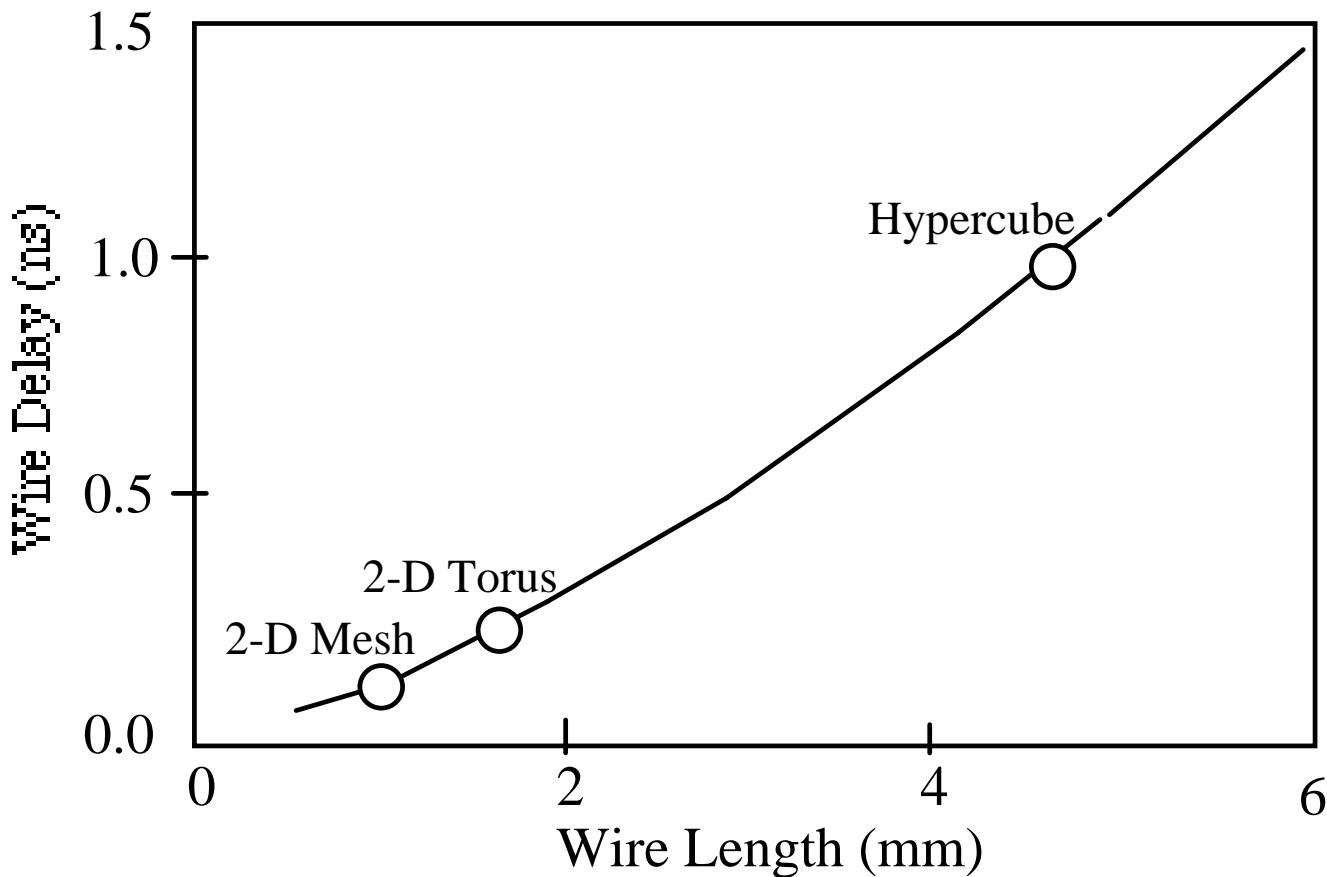
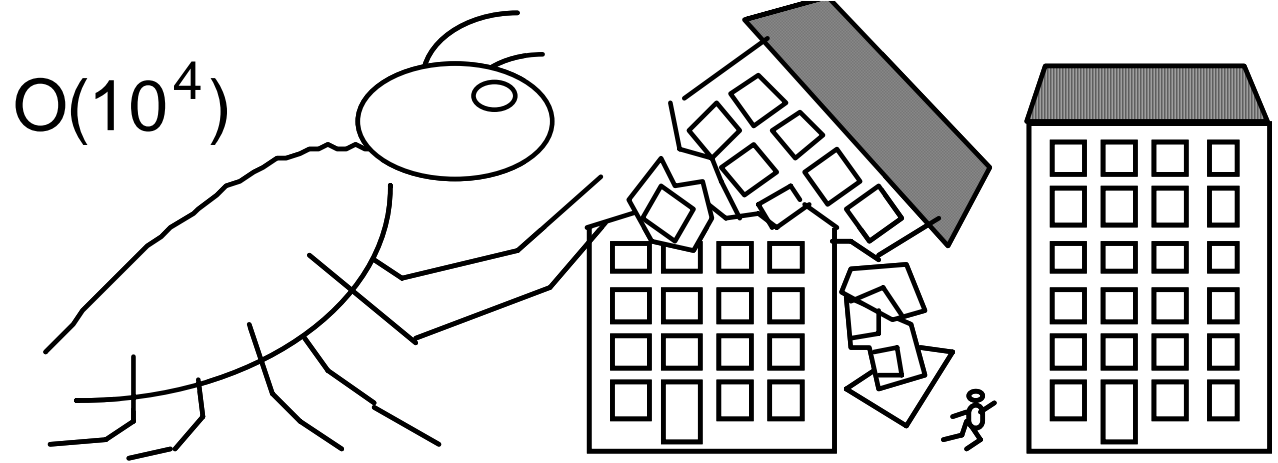


Fig. 4.10 Intrachip wire delay as a function of wire length.

# Pitfalls of Scaling up (Fig. 4.11)

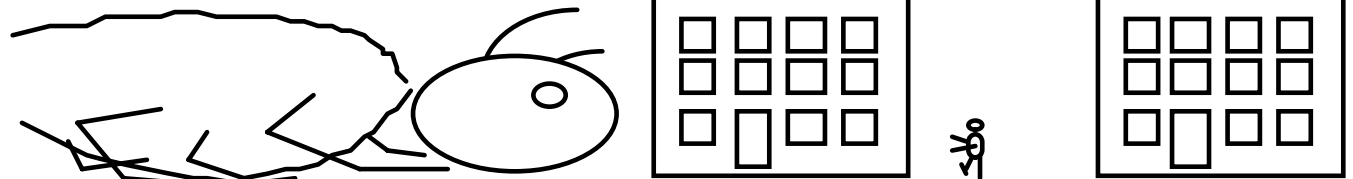
If the weight of ant grows by a factor of one trillion, the thickness of its legs must grow by a factor of one million to support the new weight



Scaled up ant on the rampage!  
What is wrong with this picture?

Ant scaled up in length  
from 5 mm to 50 m

Leg thickness must grow  
from 0.1 mm to 100 m



Scaled up ant collapses under own weight.