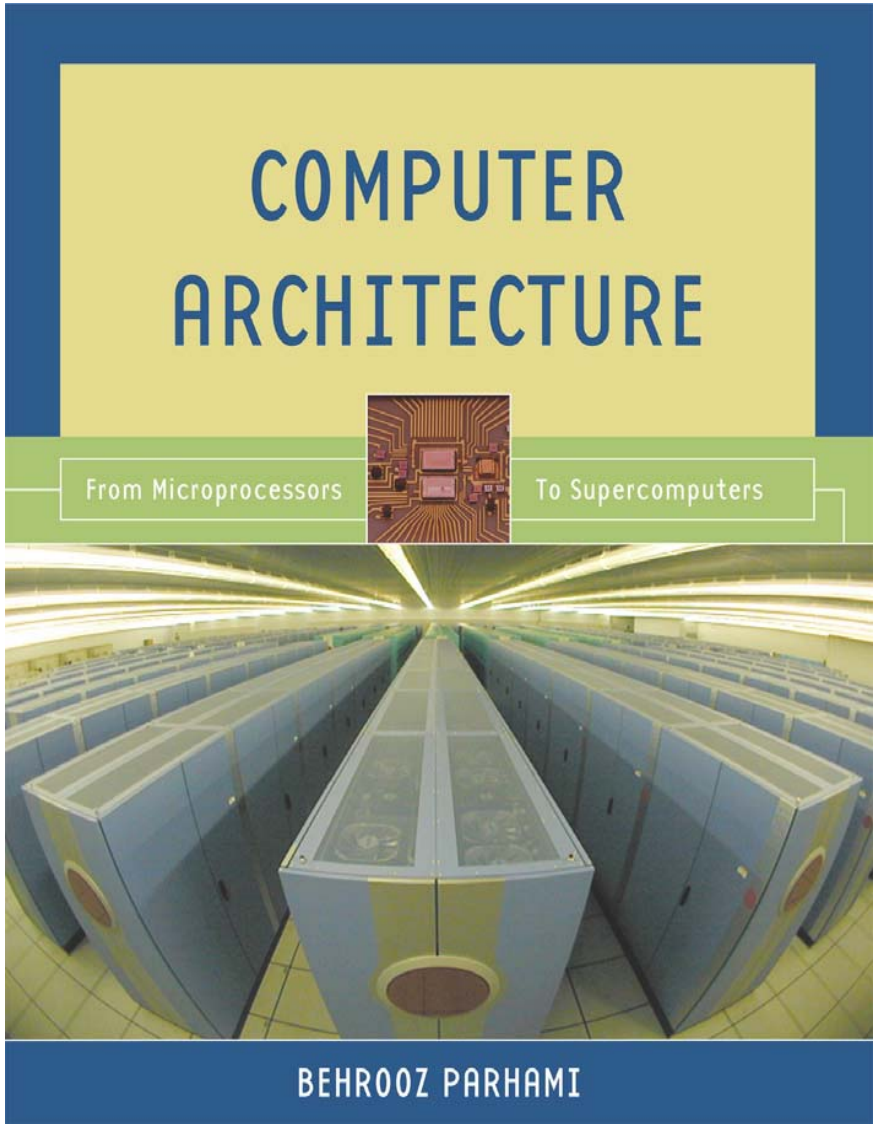


Part VII
Advanced Architectures

	Parts	Chapters
CPU	I. Background and Motivation	1. Combinational Digital Circuits 2. Digital Circuits with Memory 3. Computer System Technology 4. Computer Performance
CPU	II. Instruction-Set Architecture	5. Instructions and Addressing 6. Procedures and Data 7. Assembly Language Programs 8. Instruction-Set Variations
CPU	III. The Arithmetic/Logic Unit	9. Number Representation 10. Adders and Simple ALUs 11. Multipliers and Dividers 12. Floating-Point Arithmetic
CPU	IV. Data Path and Control	13. Instruction Execution Steps 14. Control Unit Synthesis 15. Pipelined Data Paths 16. Pipeline Performance Limits
	V. Memory System Design	17. Main Memory Concepts 18. Cache Memory Organization 19. Mass Memory Concepts 20. Virtual Memory and Paging
	VI. Input/Output and Interfacing	21. Input/Output Devices 22. Input/Ouput Programming 23. Buses, Links, and Interfacing 24. Context Switching and Interrupts
	VII. Advanced Architectures	25. Road to Higher Performance 26. Vector and Array Processing 27. Shared-Memory Multiprocessing 28. Distributed Multicomputing

About This Presentation

This presentation is intended to support the use of the textbook *Computer Architecture: From Microprocessors to Supercomputers*, Oxford University Press, 2005, ISBN 0-19-515455-X. It is updated regularly by the author as part of his teaching of the upper-division course ECE 154, Introduction to Computer Architecture, at the University of California, Santa Barbara. Instructors can use these slides freely in classroom teaching and for other educational purposes. Any other use is strictly prohibited.

Edition	Released	Revised	Revised	Revised	Revised
First	July 2003	July 2004	July 2005	Mar. 2007	

VII Advanced Architectures

Performance enhancement beyond what we have seen:

- What else can we do at the instruction execution level?
- Data parallelism: vector and array processing
- Control parallelism: parallel and distributed processing

Topics in This Part	
Chapter 25	Road to Higher Performance
Chapter 26	Vector and Array Processing
Chapter 27	Shared-Memory Multiprocessing
Chapter 28	Distributed Multicomputing

25 Road to Higher Performance

Review past, current, and future architectural trends:

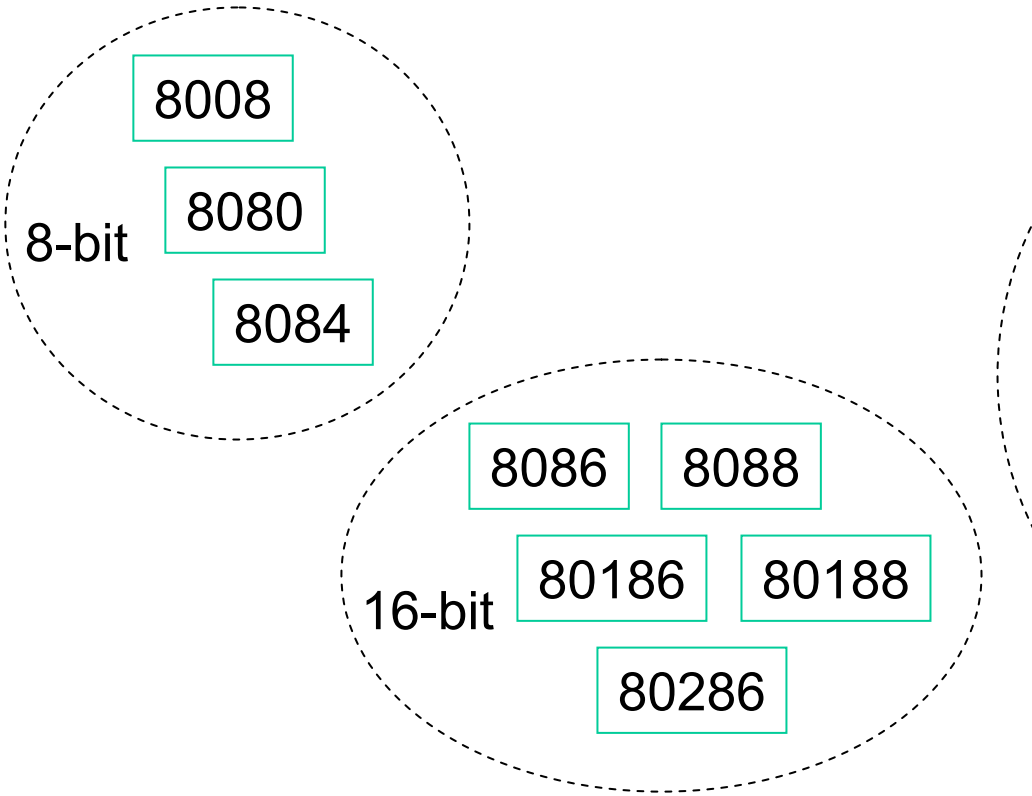
- General-purpose and special-purpose acceleration
- Introduction to data and control parallelism

Topics in This Chapter	
25.1	Past and Current Performance Trends
25.2	Performance-Driven ISA Extensions
25.3	Instruction-Level Parallelism
25.4	Speculation and Value Prediction
25.5	Special-Purpose Hardware Accelerators
25.6	Vector, Array, and Parallel Processing

25.1 Past and Current Performance Trends

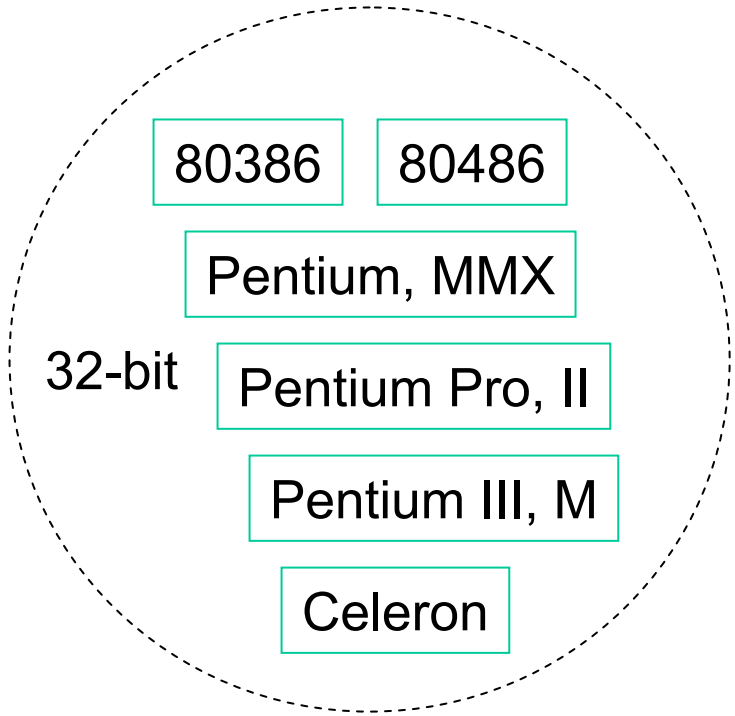
Intel 4004: The first μp (1971)

0.06 MIPS (4-bit processor)



Intel Pentium 4, circa 2005

10,000 MIPS (32-bit processor)



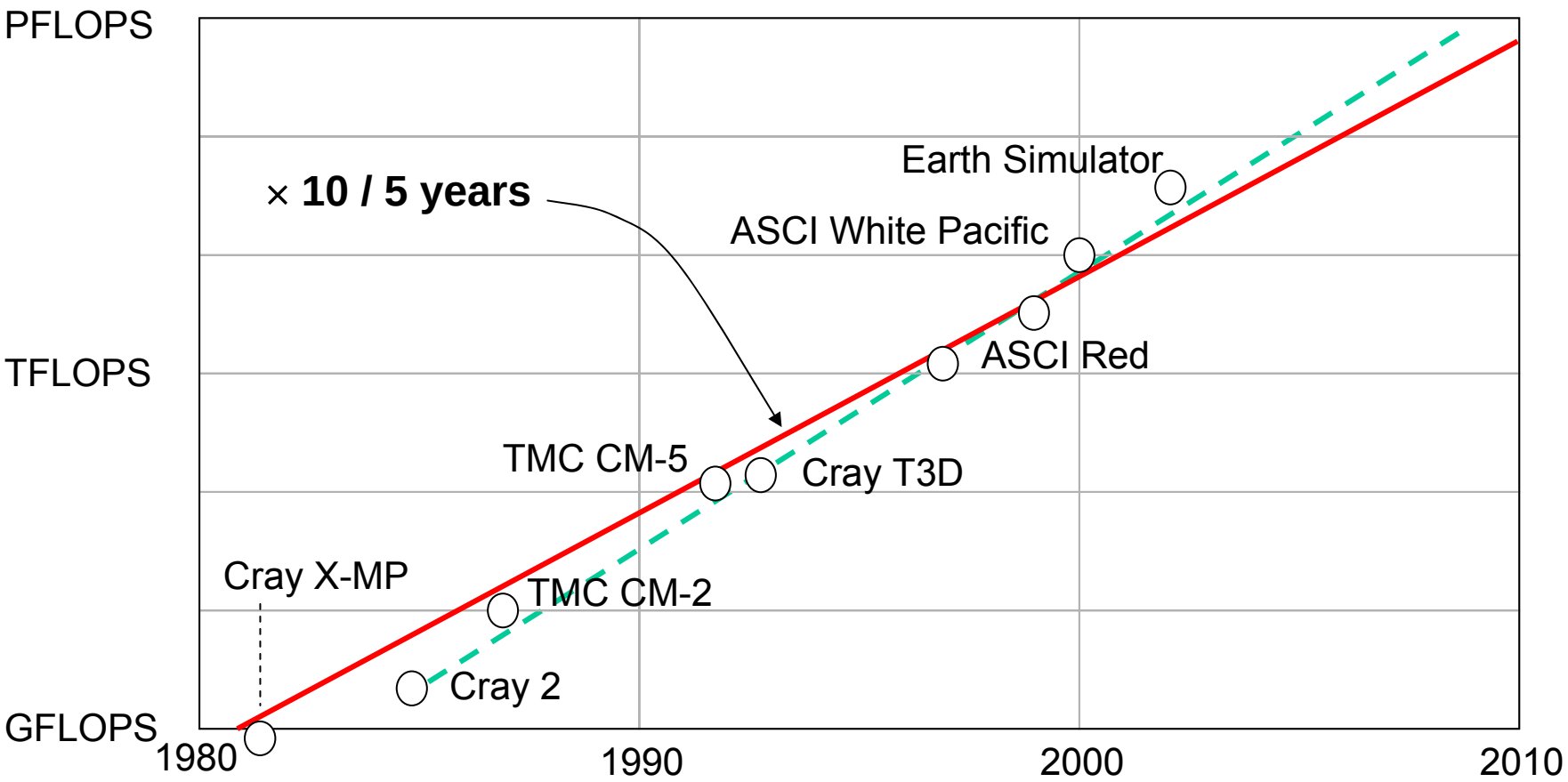
Architectural Innovations for Improved Performance

Computer performance grew by a factor of about 10000 between 1980 and 2000
- 100 due to faster technology
- 100 due to better architecture

Available computing power ca. 2000:
- GFLOPS on desktop
- TFLOPS in supercomputer center
- PFLOPS on drawing board

	Architectural method	Improvement factor	
Established methods	1. Pipelining (and superpipelining)	3-8 √	Previously discussed
	2. Cache memory, 2-3 levels	2-5 √	
	3. RISC and related ideas	2-3 √	
	4. Multiple instruction issue (superscalar)	2-3 √	
	5. ISA extensions (e.g., for multimedia)	1-3 √	Covered in Part VII
Newer methods	6. Multithreading (super-, hyper-)	2-5 ?	
	7. Speculation and value prediction	2-3 ?	
	8. Hardware acceleration	2-10 ?	
	9. Vector and array processing	2-10 ?	
	10. Parallel/distributed computing	2-1000s ?	

Peak Performance of Supercomputers



Dongarra, J., "Trends in High Performance Computing," *Computer J.*, Vol. 47, No. 4, pp. 399-403, 2004. [Dong04]

Energy Consumption is Getting out of Hand

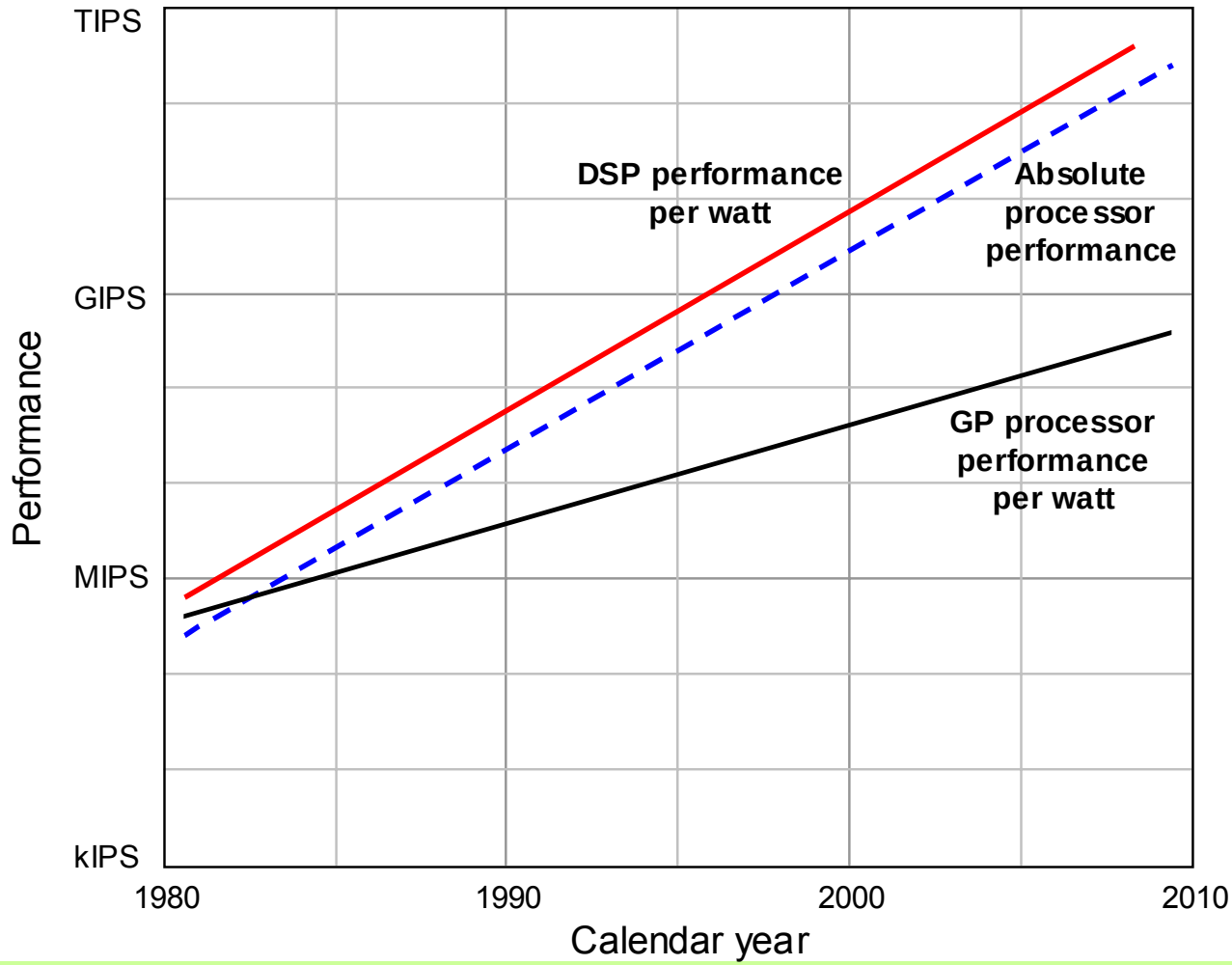


Figure 25.1 Trend in energy consumption for each MIPS of computational power in general-purpose processors and DSPs.

25.2 Performance-Driven ISA Extensions

Adding instructions that do more work per cycle

Shift-add: replace two instructions with one (e.g., multiply by 5)

Multiply-add: replace two instructions with one ($x := c + a \times b$)

Multiply-accumulate: reduce round-off error ($s := s + a \times b$)

Conditional copy: to avoid some branches (e.g., in if-then-else)

Subword parallelism (for multimedia applications)

Intel MMX: multimedia extension

64-bit registers can hold multiple integer operands

Intel SSE: Streaming SIMD extension

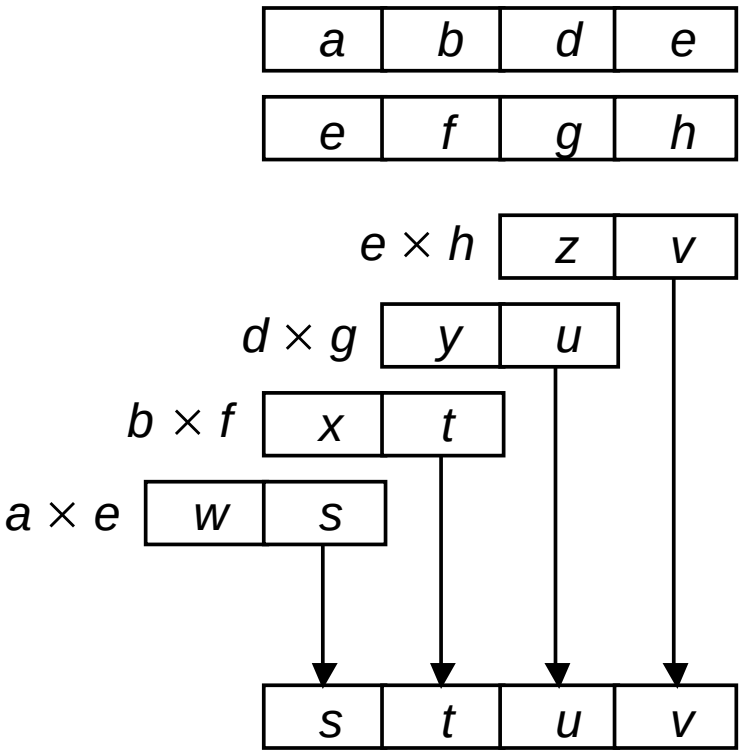
128-bit registers can hold several floating-point operands

Intel MMX ISA Exten-sion

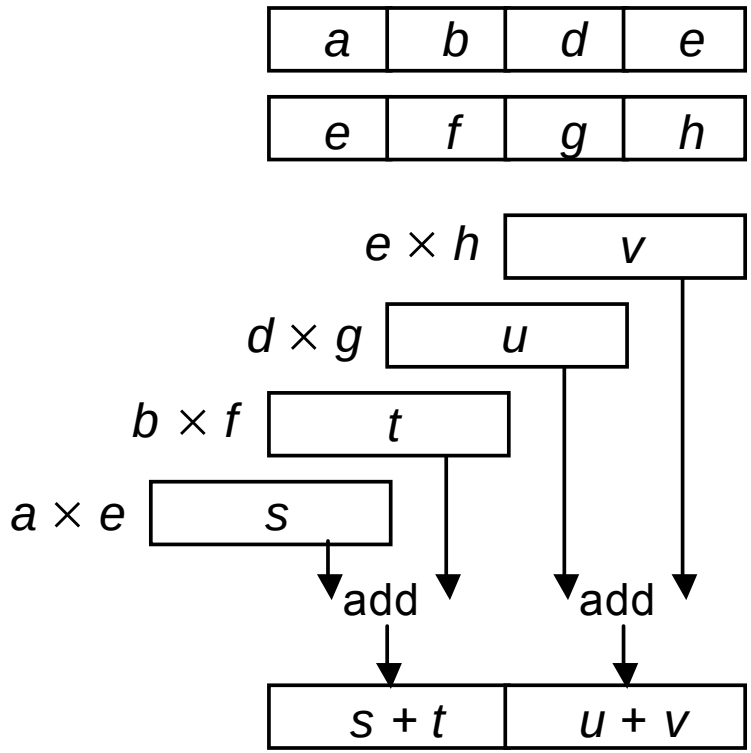
Class	Instruction	Vector	Op type	Function or results
Copy	Register copy		32 bits	Integer register ↔ MMX register
	Parallel pack	4, 2	Saturate	Convert to narrower elements
	Parallel unpack low	8, 4, 2		Merge lower halves of 2 vectors
	Parallel unpack high	8, 4, 2		Merge upper halves of 2 vectors
Arith-metic	Parallel add	8, 4, 2	Wrap/Saturate#	Add; inhibit carry at boundaries
	Parallel subtract	8, 4, 2	Wrap/Saturate#	Subtract with carry inhibition
	Parallel multiply low	4		Multiply, keep the 4 low halves
	Parallel multiply high	4		Multiply, keep the 4 high halves
	Parallel multiply-add	4		Multiply, add adjacent products*
	Parallel compare equal	8, 4, 2		All 1s where equal, else all 0s
	Parallel compare greater	8, 4, 2		All 1s where greater, else all 0s
Shift	Parallel left shift logical	4, 2, 1		Shift left, respect boundaries
	Parallel right shift logical	4, 2, 1		Shift right, respect boundaries
	Parallel right shift arith	4, 2		Arith shift within each (half)word
Logic	Parallel AND	1	Bitwise	dest ← (src1) ∧ (src2)
	Parallel ANDNOT	1	Bitwise	dest ← (src1) ∧ (src2)′
	Parallel OR	1	Bitwise	dest ← (src1) ∨ (src2)
	Parallel XOR	1	Bitwise	dest ← (src1) ⊕ (src2)
Memory access	Parallel load MMX reg		32 or 64 bits	Address given in integer register
	Parallel store MMX reg		32 or 64 bit	Address given in integer register
Control	Empty FP tag bits			Required for compatibility$

Table 25.1

MMX Multiplication and Multiply-Add



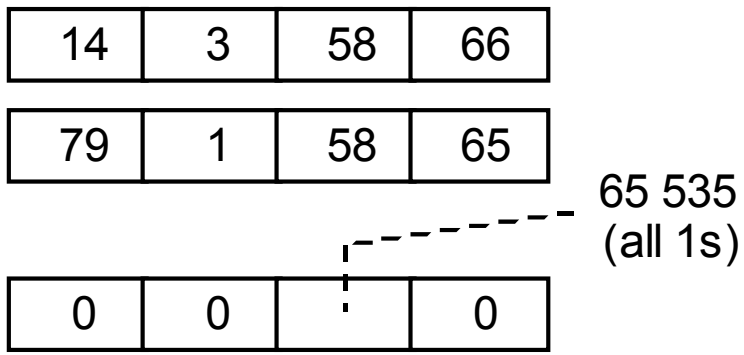
(a) Parallel multiply low



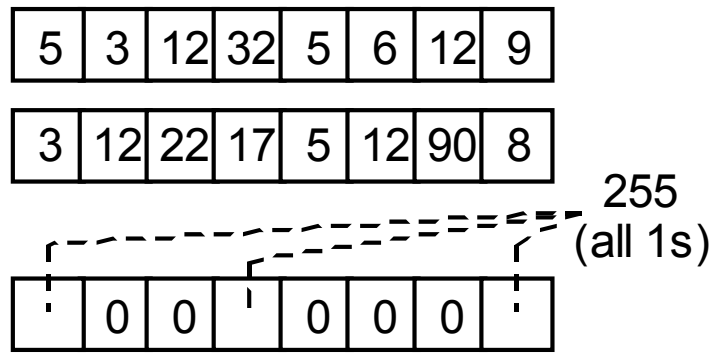
(b) Parallel multiply-add

Figure 25.2 Parallel multiplication and multiply-add in MMX.

MMX Parallel Comparisons



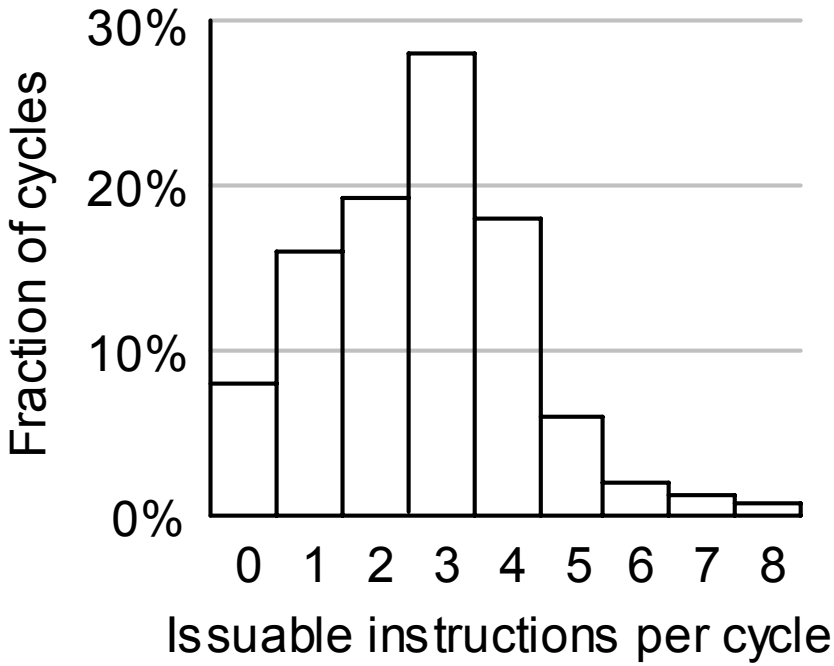
(a) Parallel compare equal



(b) Parallel compare greater

Figure 25.3 Parallel comparisons in MMX.

25.3 Instruction-Level Parallelism



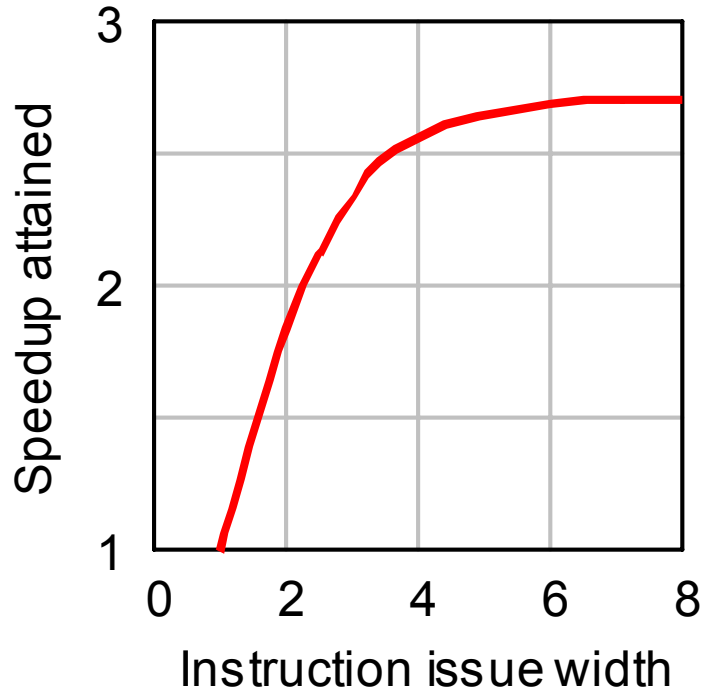


Figure 25.4 Available instruction-level parallelism and the speedup due to multiple instruction issue in superscalar processors [John91].

Instruction-Level Parallelism

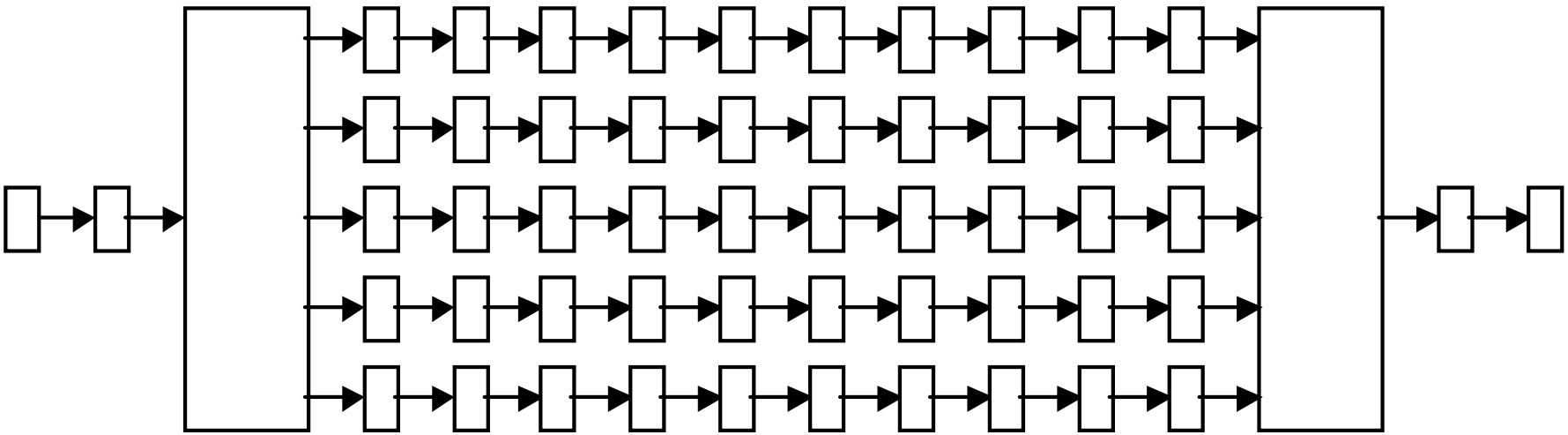


Figure 25.5 A computation with inherent instruction-level parallelism.

VLIW and EPIC Architectures

VLIW Very long instruction word architecture
EPIC Explicitly parallel instruction computing

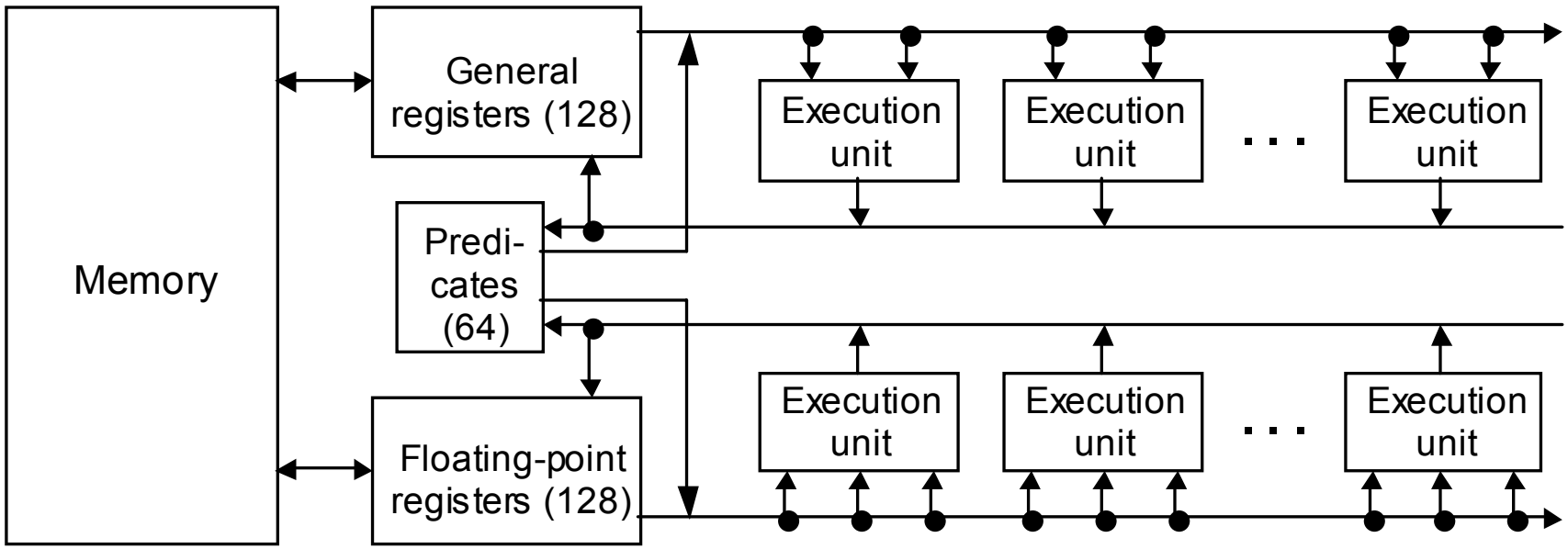
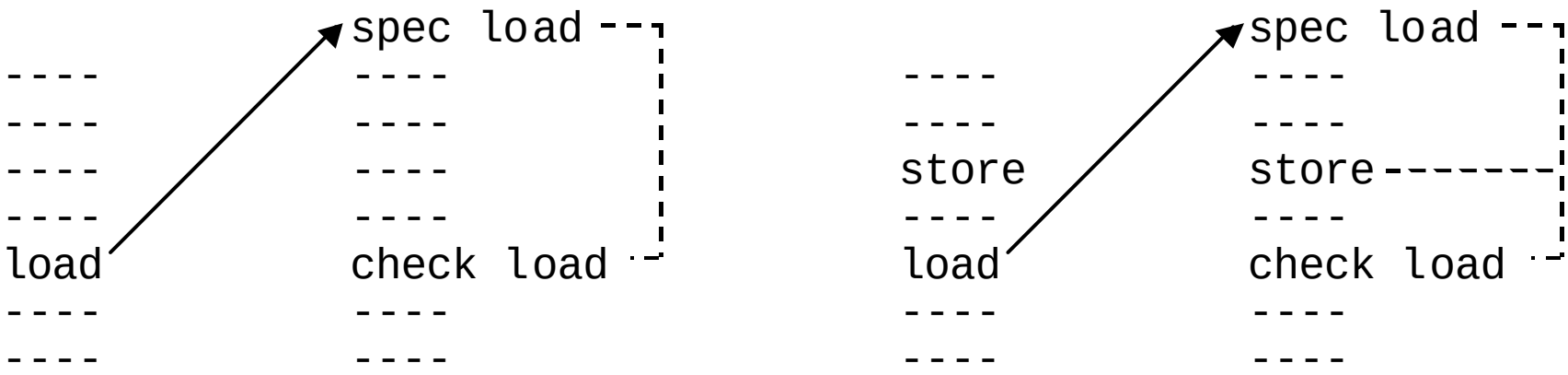


Figure 25.6 Hardware organization for IA-64. General and floating-point registers are 64-bit wide. Predicates are single-bit registers.

25.4 Speculation and Value Prediction



(a) Control speculation

(b) Data speculation

Figure 25.7 Examples of software speculation in IA-64.

Value Prediction

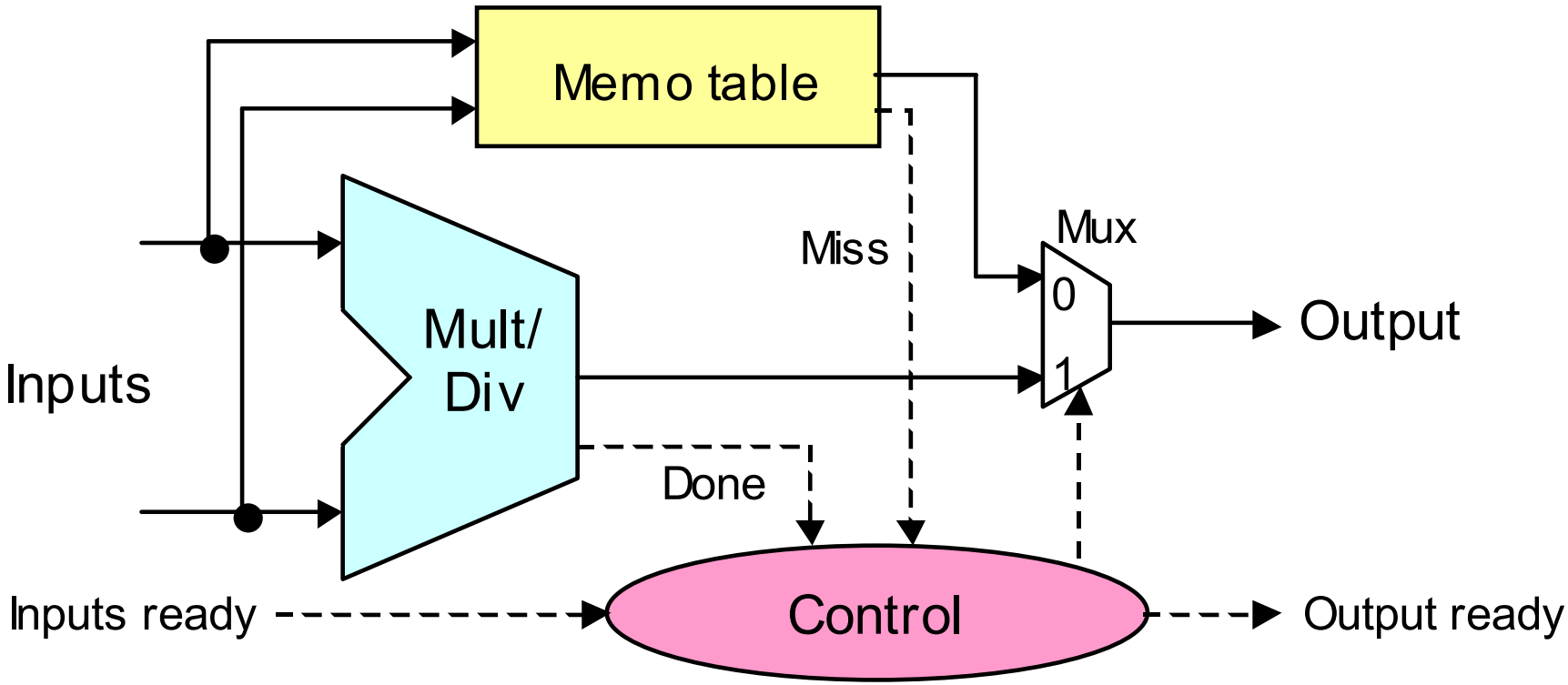


Figure 25.8 Value prediction for multiplication or division via a memo table.

25.5 Special-Purpose Hardware Accelerators

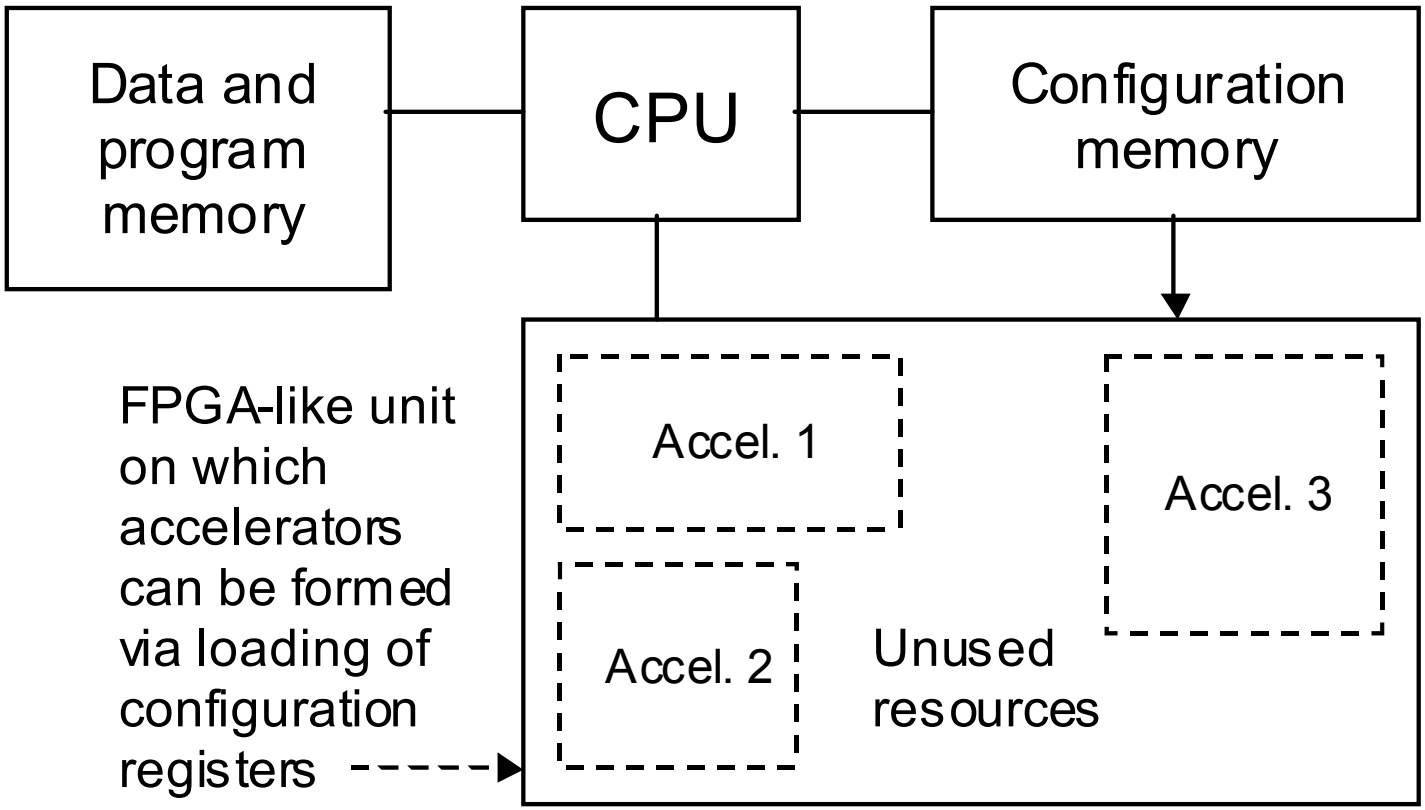


Figure 25.9 General structure of a processor with configurable hardware accelerators.

Graphic Processors, Network Processors, etc.

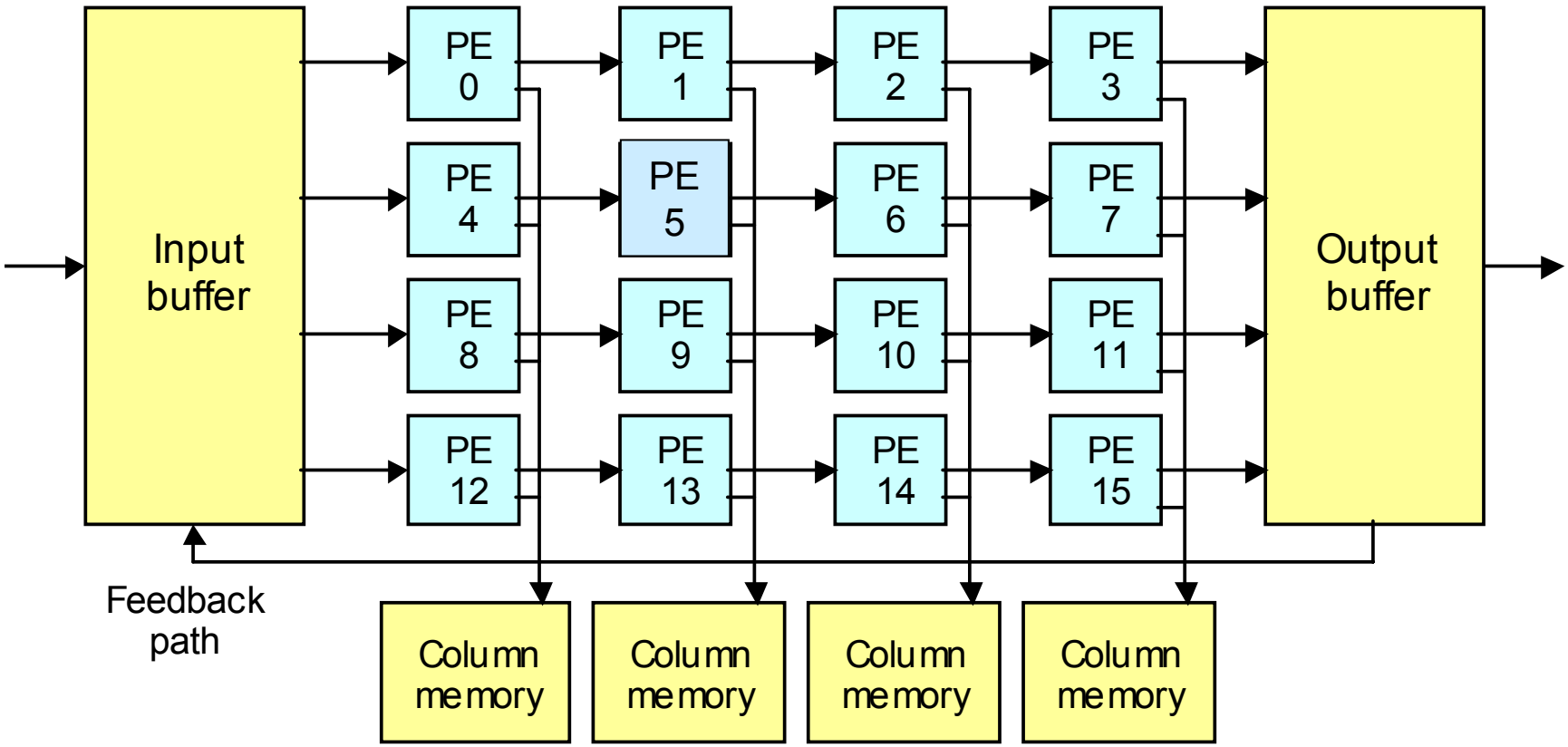


Figure 25.10 Simplified block diagram of Toaster2, Cisco Systems' network processor.

25.6 Vector, Array, and Parallel Processing

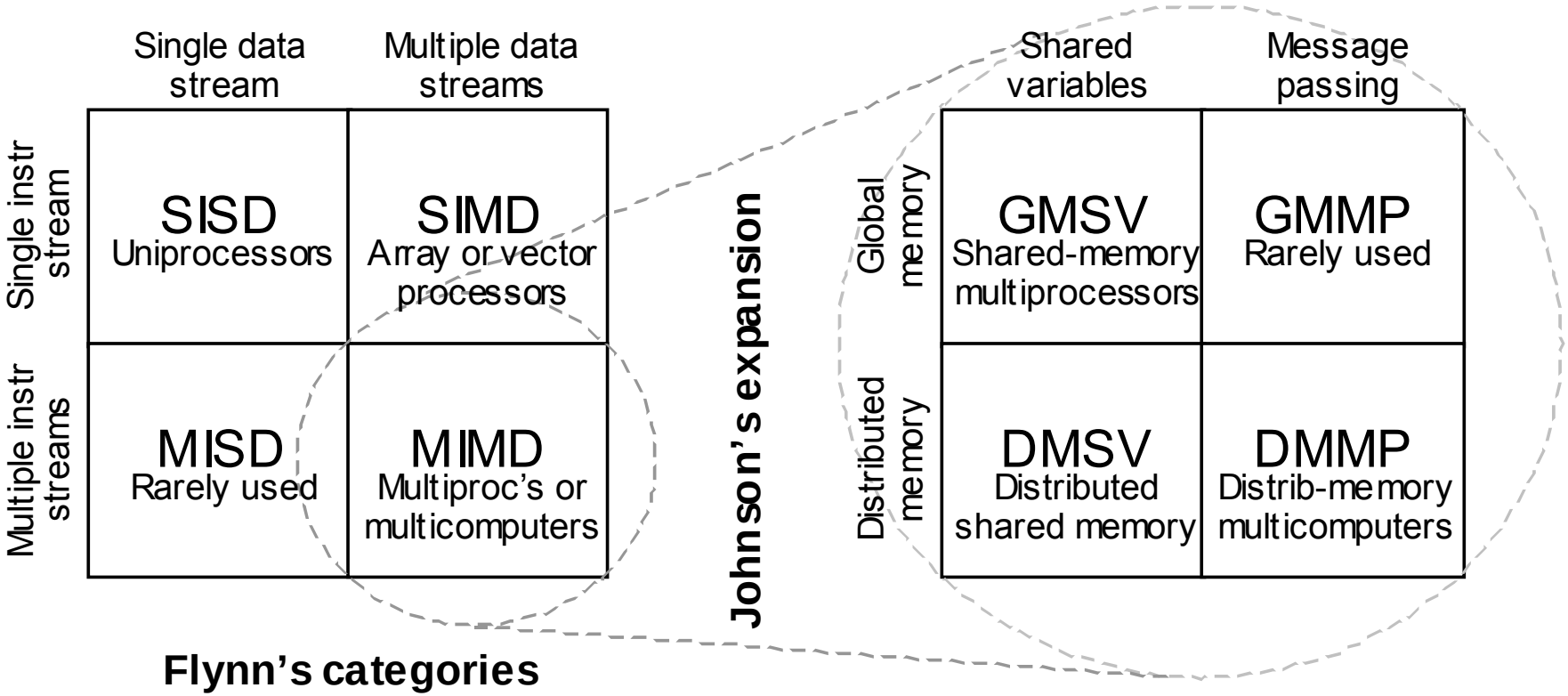


Figure 25.11 The Flynn-Johnson classification of computer systems.

SIMD Architectures

Data parallelism: executing one operation on multiple data streams

Concurrency in time – vector processing
Concurrency in space – array processing

Example to provide context

Multiplying a coefficient vector by a data vector (e.g., in filtering)
$y[i] := c[i] \times x[i], \ 0 \leq i < n$

Sources of performance improvement in vector processing
(details in the first half of Chapter 26)

One instruction is fetched and decoded for the entire operation
The multiplications are known to be independent (no checking)
Pipelining/concurrency in memory access as well as in arithmetic

Array processing is similar (details in the second half of Chapter 26)

MISD Architecture Example

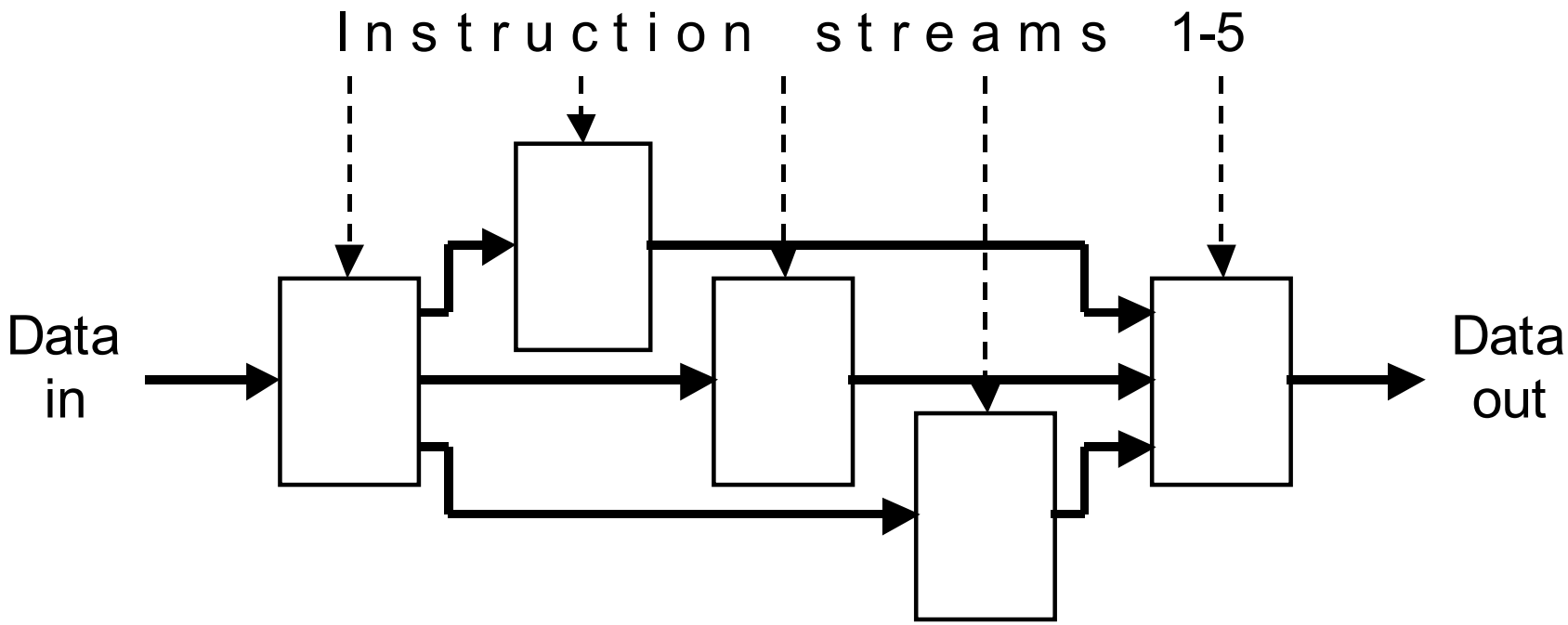


Figure 25.12 Multiple instruction streams operating on a single data stream (MISD).

MIMD Architectures

Control parallelism: executing several instruction streams in parallel

GMSV: Shared global memory – symmetric multiprocessors
DMSV: Shared distributed memory – asymmetric multiprocessors
DMMP: Message passing – multicomputers

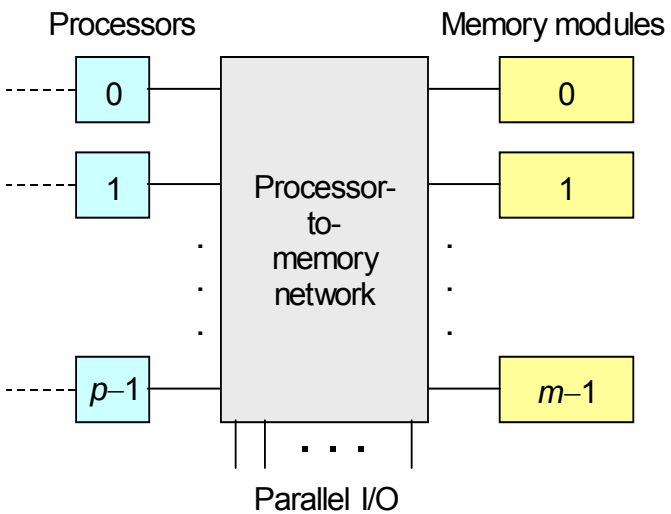


Figure 27.1 Centralized shared memory.

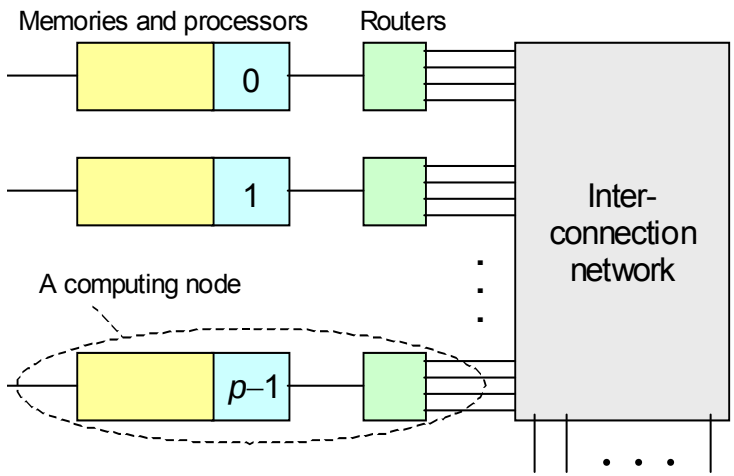
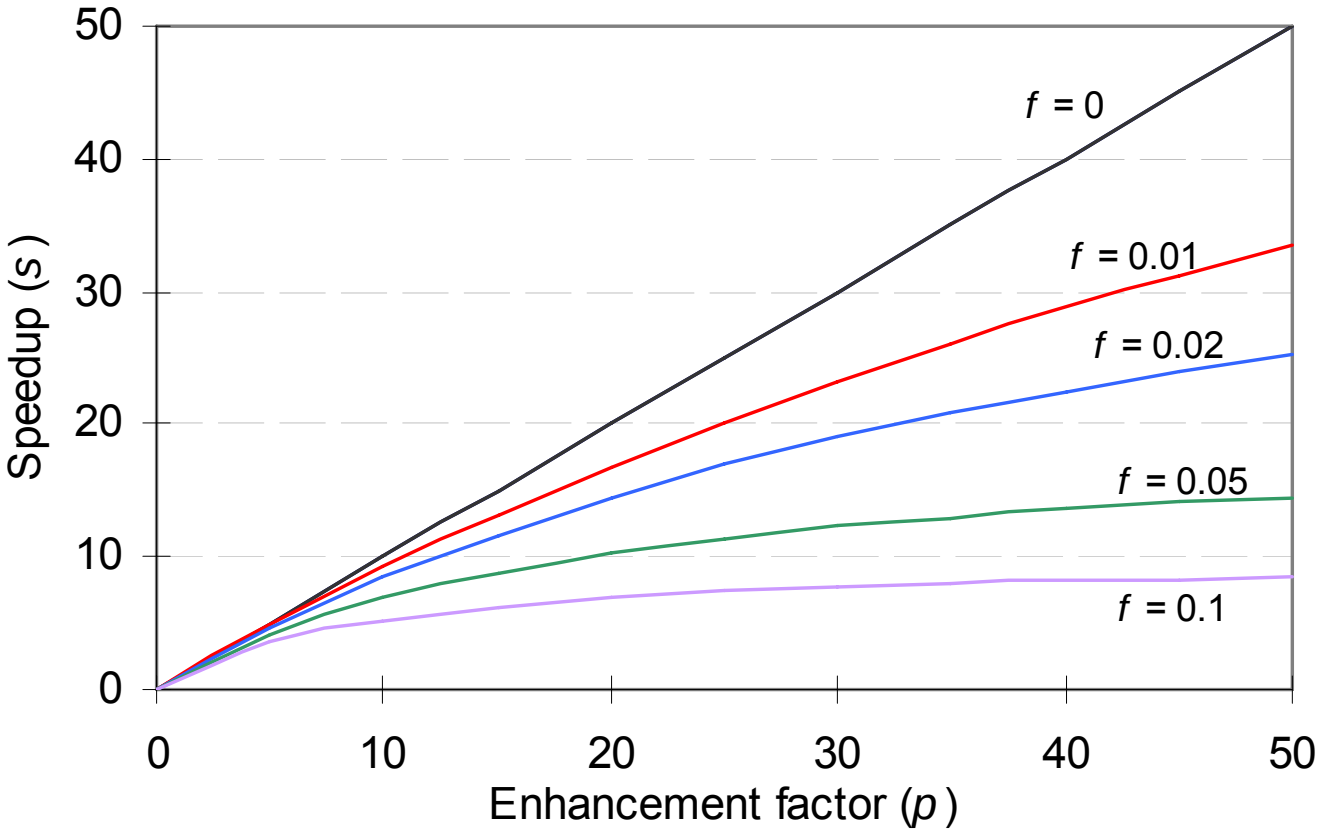


Figure 28.1 Distributed memory.

Amdahl's Law Revisited



f = sequential fraction

p = speedup of the rest with p processors

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

$$\leq \min(p, 1/f)$$

Figure 4.4 Amdahl's law: speedup achieved if a fraction f of a task is unaffected and the remaining 1 – f part runs p times as fast.

26 Vector and Array Processing

Single instruction stream operating on multiple data streams

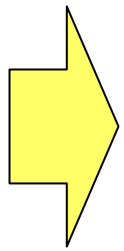
- Data parallelism in time = vector processing
- Data parallelism in space = array processing

Topics in This Chapter	
26.1	Operations on Vectors
26.2	Vector Processor Implementation
26.3	Vector Processor Performance
26.4	Shared-Control Systems
26.5	Array Processor Implementation
26.6	Array Processor Performance

26.1 Operations on Vectors

Sequential processor:

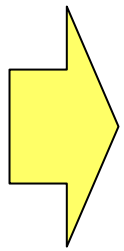
```
for i = 0 to 63 do
  P[i] := W[i] × D[i]
endfor
```



Vector processor:

```
load W
load D
P := W × D
store P
```

```
for i = 0 to 63 do
  X[i+1] := X[i] + Z[i]
  Y[i+1] := X[i+1]+Y[i]
endfor
```



Unparallelizable

26.2 Vector Processor Implementation

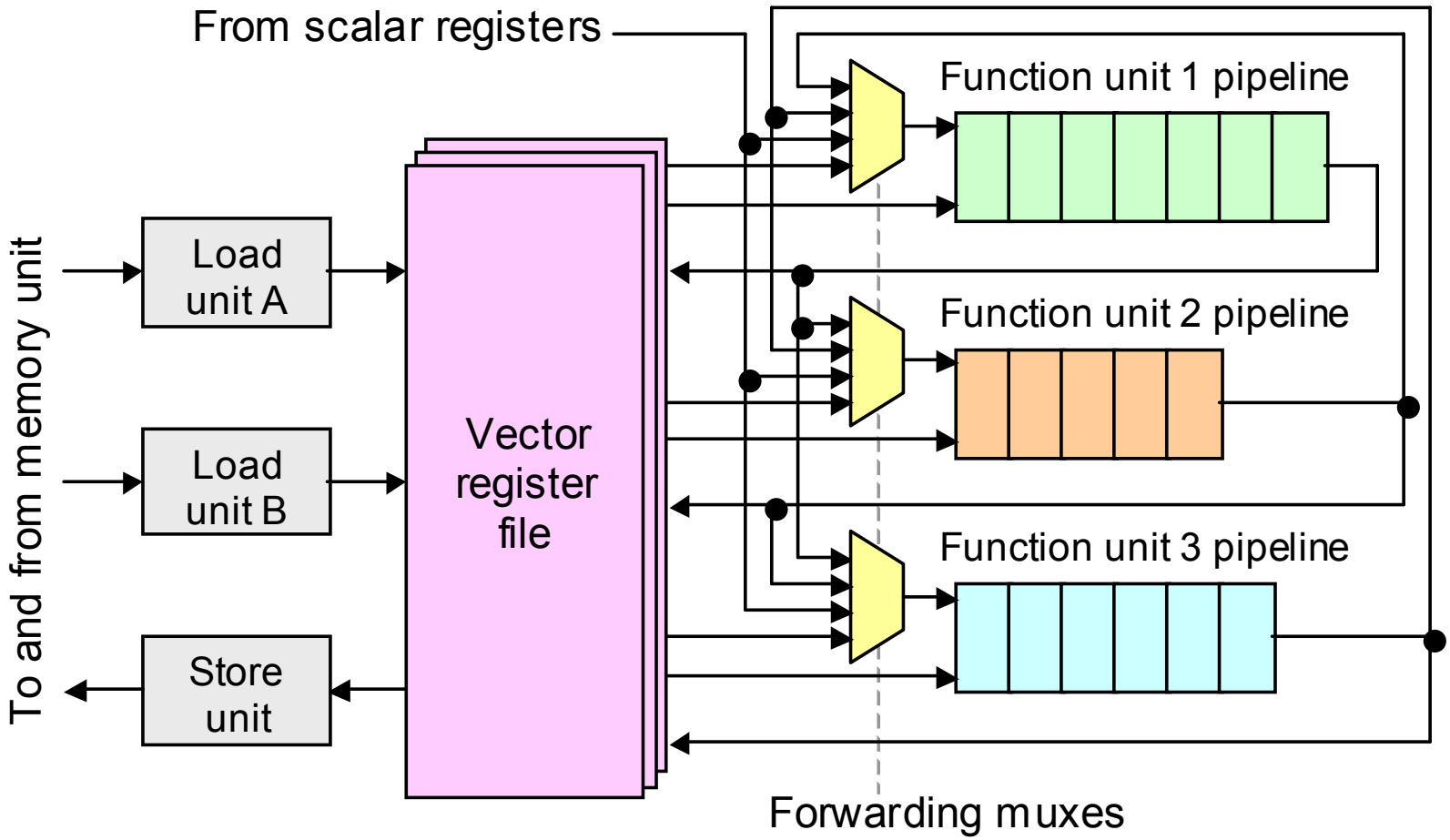
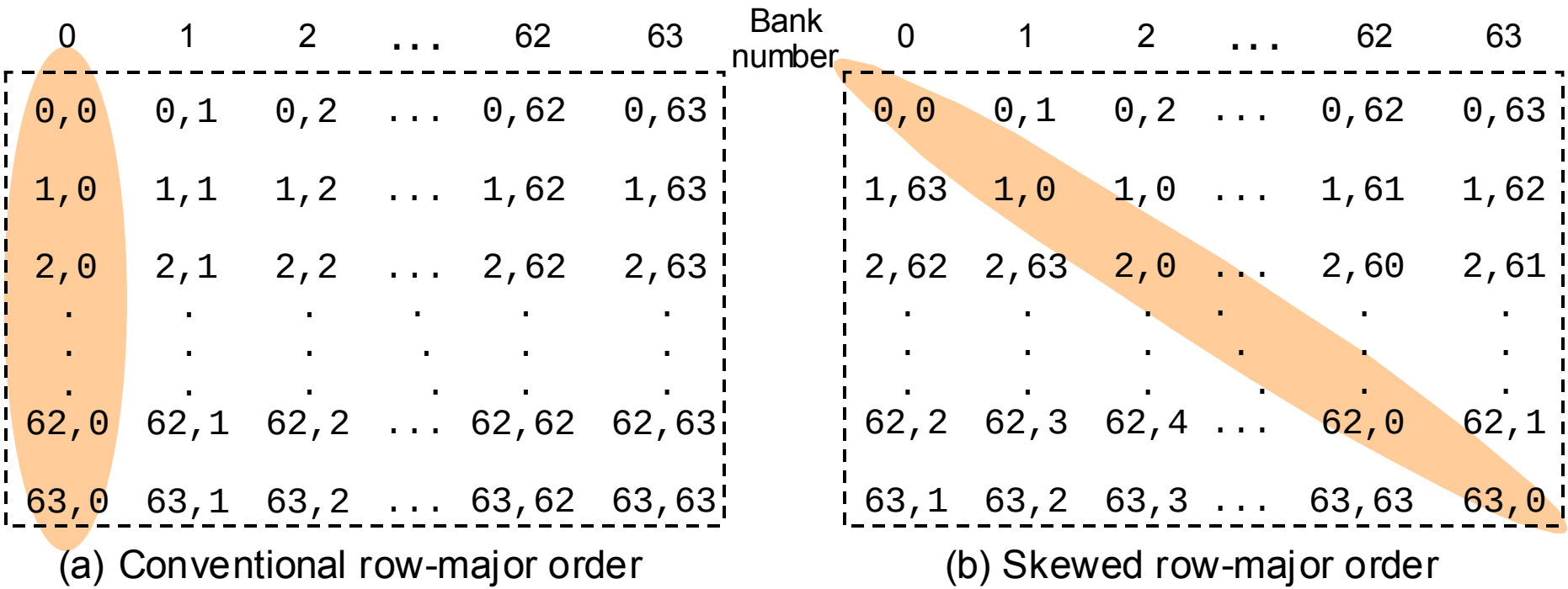


Figure 26.1 Simplified generic structure of a vector processor.

Conflict-Free Memory Access



(a) Conventional row-major order

(b) Skewed row-major order

Figure 26.2 Skewed storage of the elements of a 64 × 64 matrix for conflict-free memory access in a 64-way interleaved memory. Elements of column 0 are highlighted in both diagrams .

Overlapped Memory Access and Computation

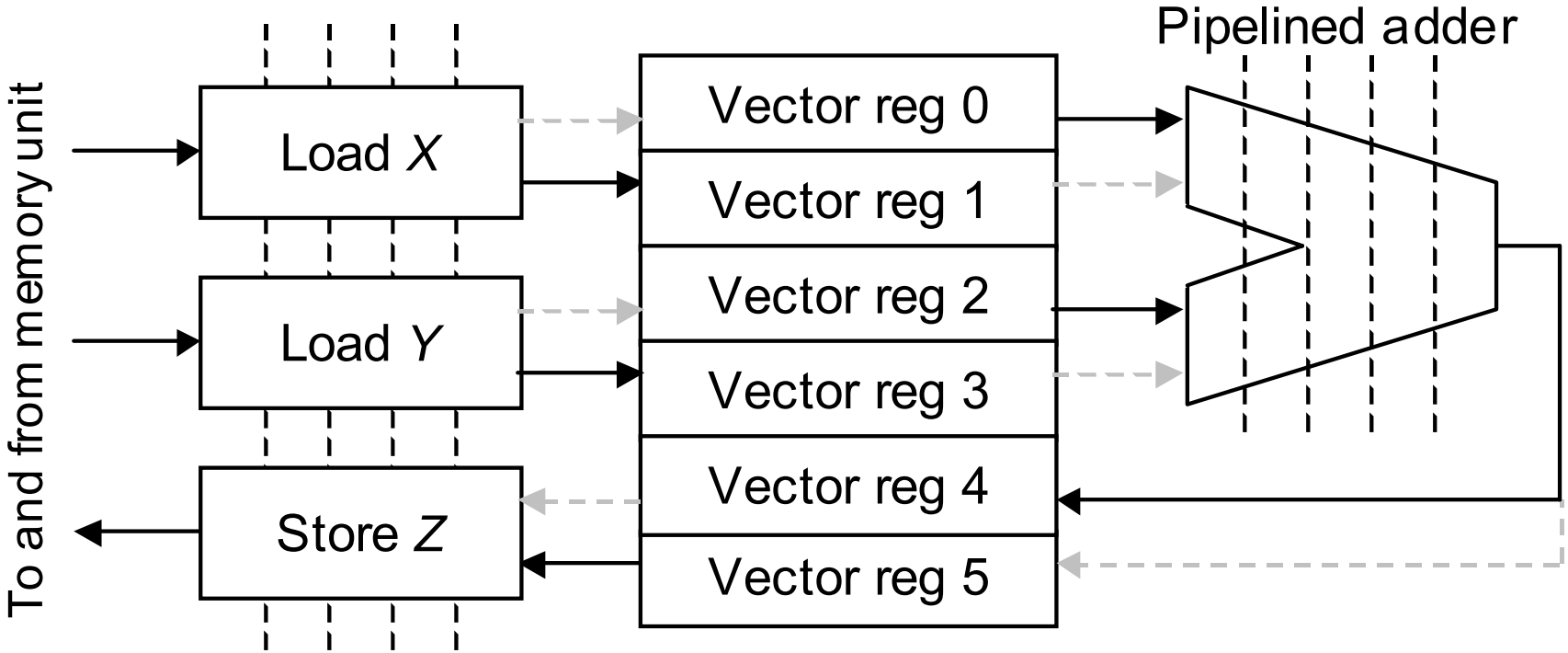


Figure 26.3 Vector processing via segmented load/store of vectors in registers in a double-buffering scheme. Solid (dashed) lines show data flow in the current (next) segment.

26.3 Vector Processor Performance

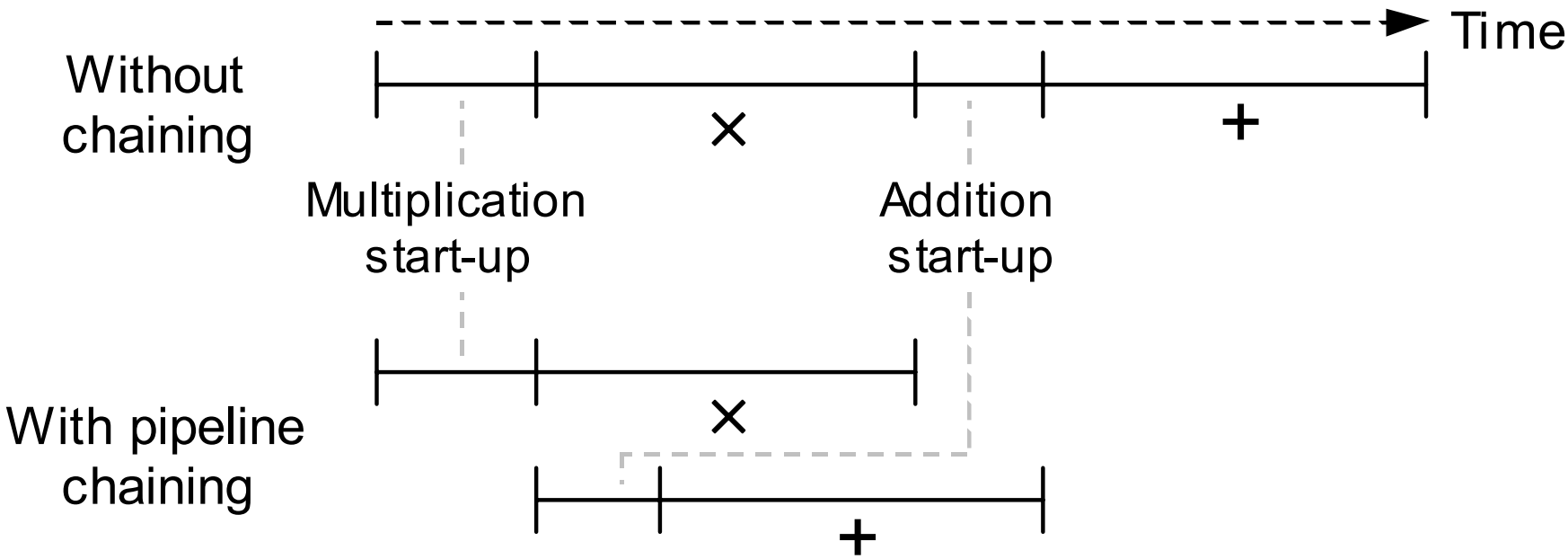


Figure 26.4 Total latency of the vector computation $S := X \times Y + Z$, without and with pipeline chaining.

Performance as a Function of Vector Length

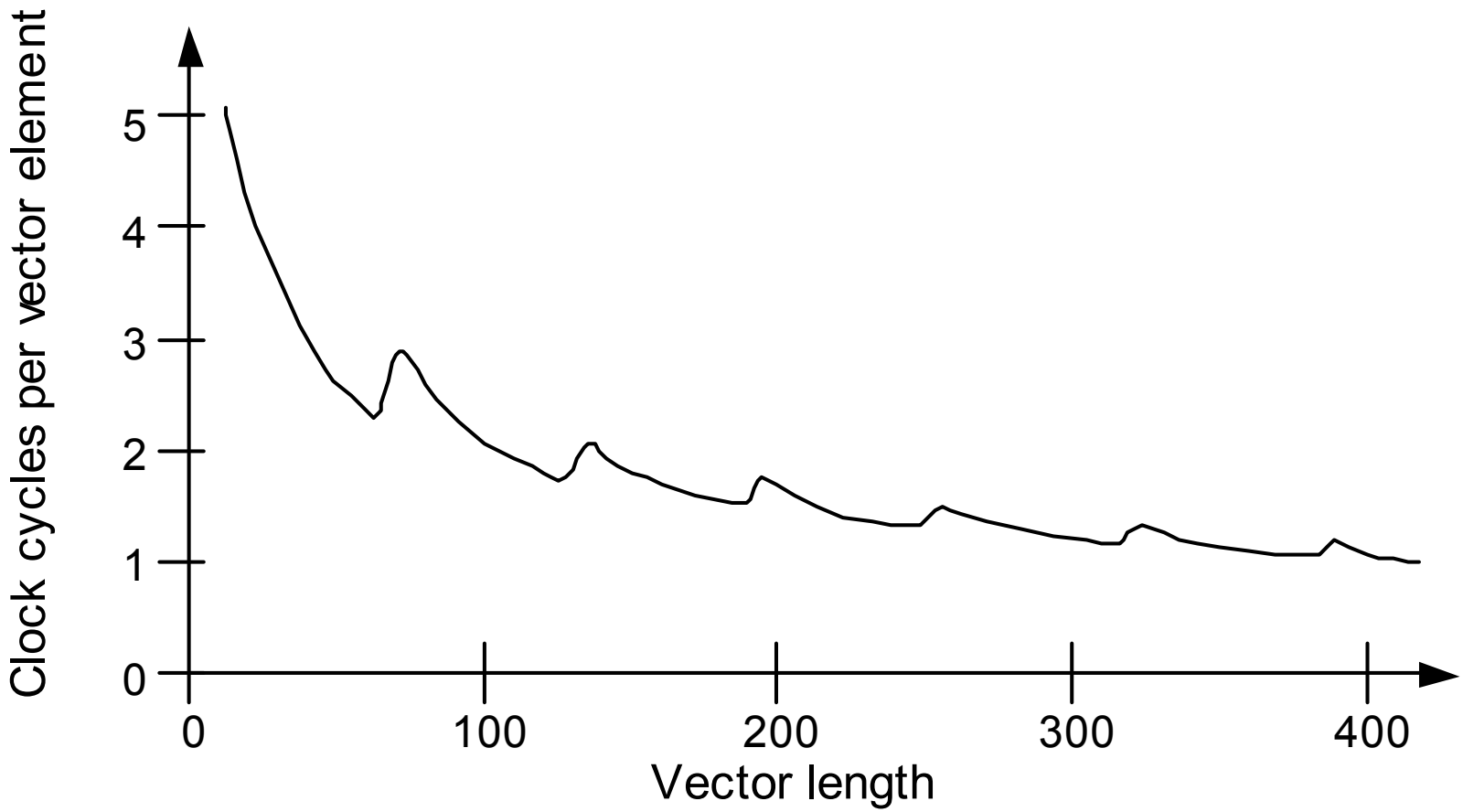
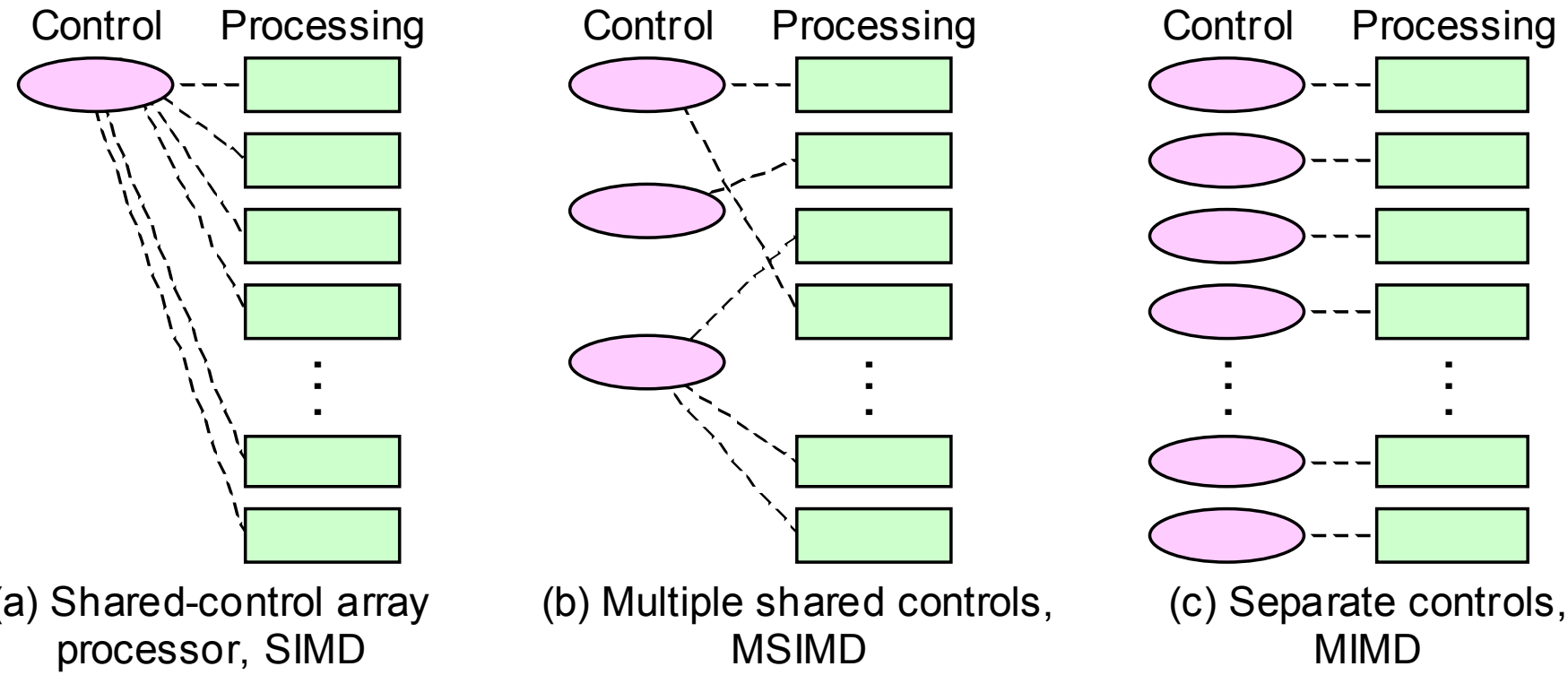


Figure 26.5 The per-element execution time in a vector processor as a function of the vector length.

26.4 Shared-Control Systems



(a) Shared-control array processor, SIMD

(b) Multiple shared controls, MSIMD

(c) Separate controls, MIMD

Figure 26.6 From completely shared control to totally separate controls.

Example Array Processor

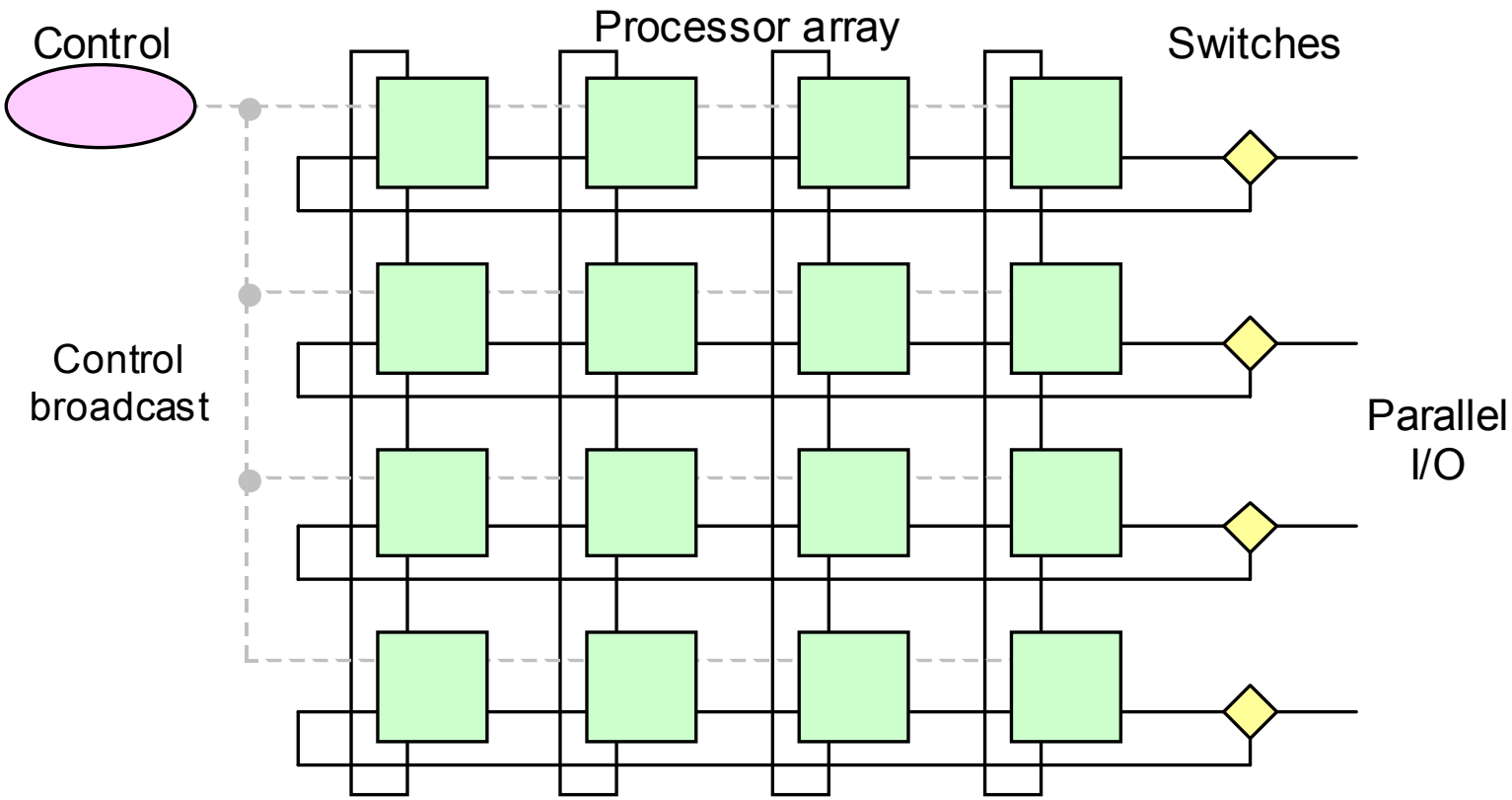


Figure 26.7 Array processor with 2D torus interprocessor communication network.

26.5 Array Processor Implementation

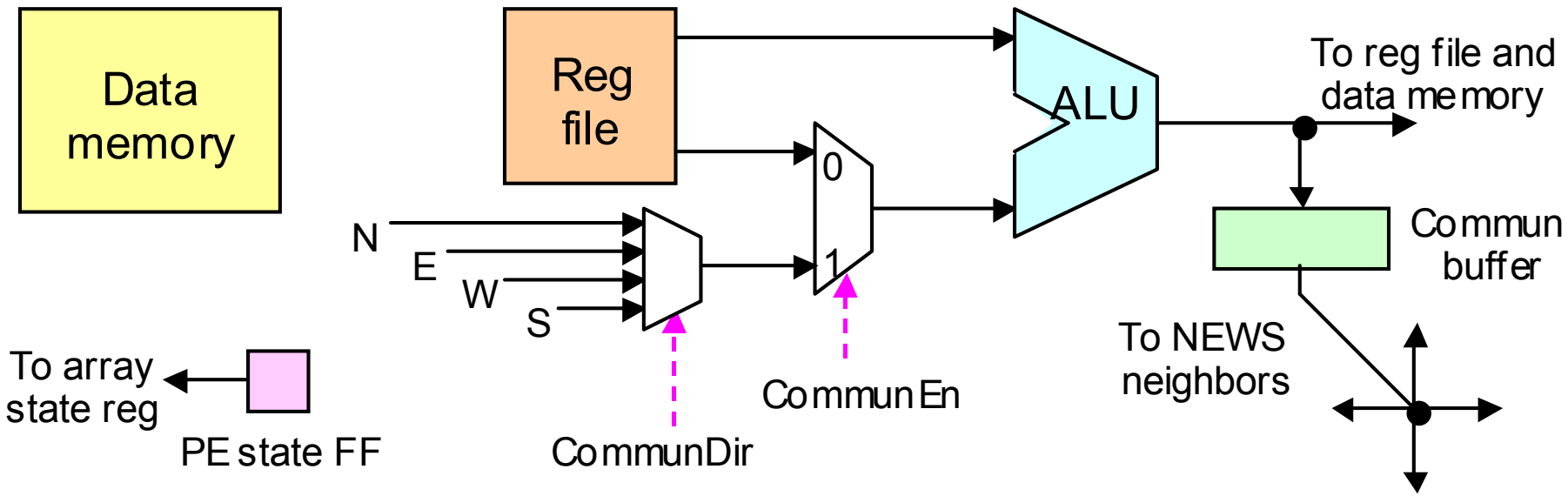


Figure 26.8 Handling of interprocessor communication via a mechanism similar to data forwarding.

Configuration Switches

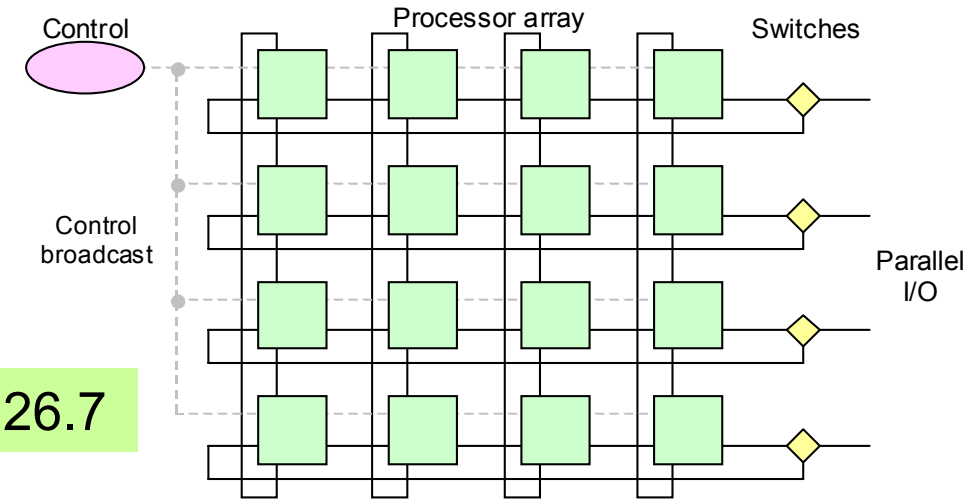
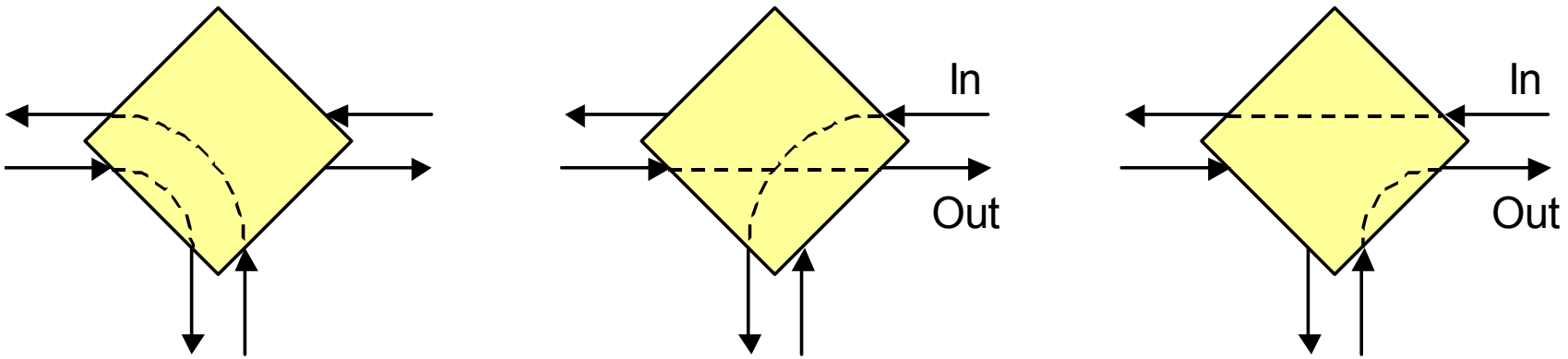


Figure 26.7



(a) Torus operation (b) Clockwise I/O (c) Counterclockwise I/O

Figure 26.9 I/O switch states in the array processor of Figure 26.7.

26.6 Array Processor Performance

Array processors perform well for the same class of problems that are suitable for vector processors

For *embarrassingly* (*pleasantly*) *parallel* problems, array processors can be faster and more energy-efficient than vector processors

A criticism of array processing:
For conditional computations, a significant part of the array remains idle while the "then" part is performed; subsequently, idle and busy processors reverse roles during the "else" part

However:
Considering array processors inefficient due to idle processors is like criticizing mass transportation because many seats are unoccupied most of the time

It's the total cost of computation that counts, not hardware utilization!

27 Shared-Memory Multiprocessing

Multiple processors sharing a memory unit seems naïve

- Didn't we conclude that memory is the bottleneck?
- How then does it make sense to share the memory?

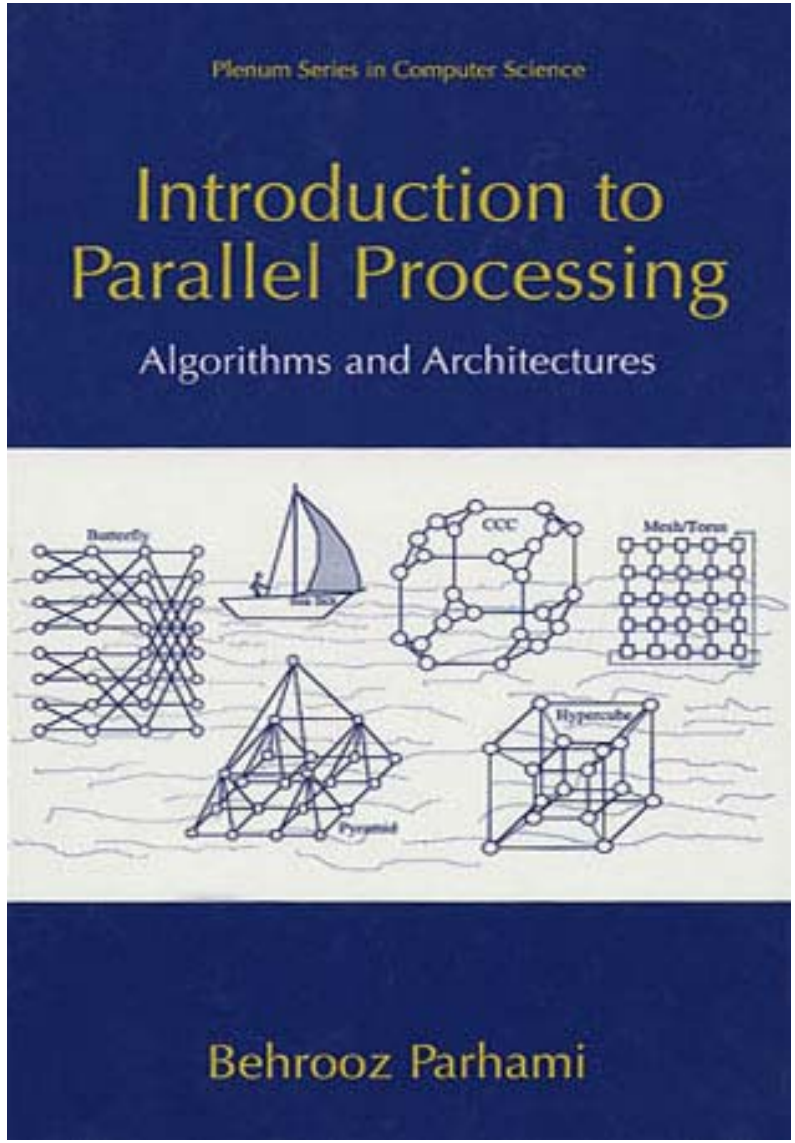
Topics in This Chapter	
27.1	Centralized Shared Memory
27.2	Multiple Caches and Cache Coherence
27.3	Implementing Symmetric Multiprocessors
27.4	Distributed Shared Memory
27.5	Directories to Guide Data Access
27.6	Implementing Asymmetric Multiprocessors

Parallel Processing as a Topic of Study

An important area of study that allows us to overcome fundamental speed limits

Our treatment of the topic is quite brief (Chapters 26-27)

Graduate course ECE 254B: Adv. Computer Architecture – Parallel Processing



27.1 Centralized Shared Memory

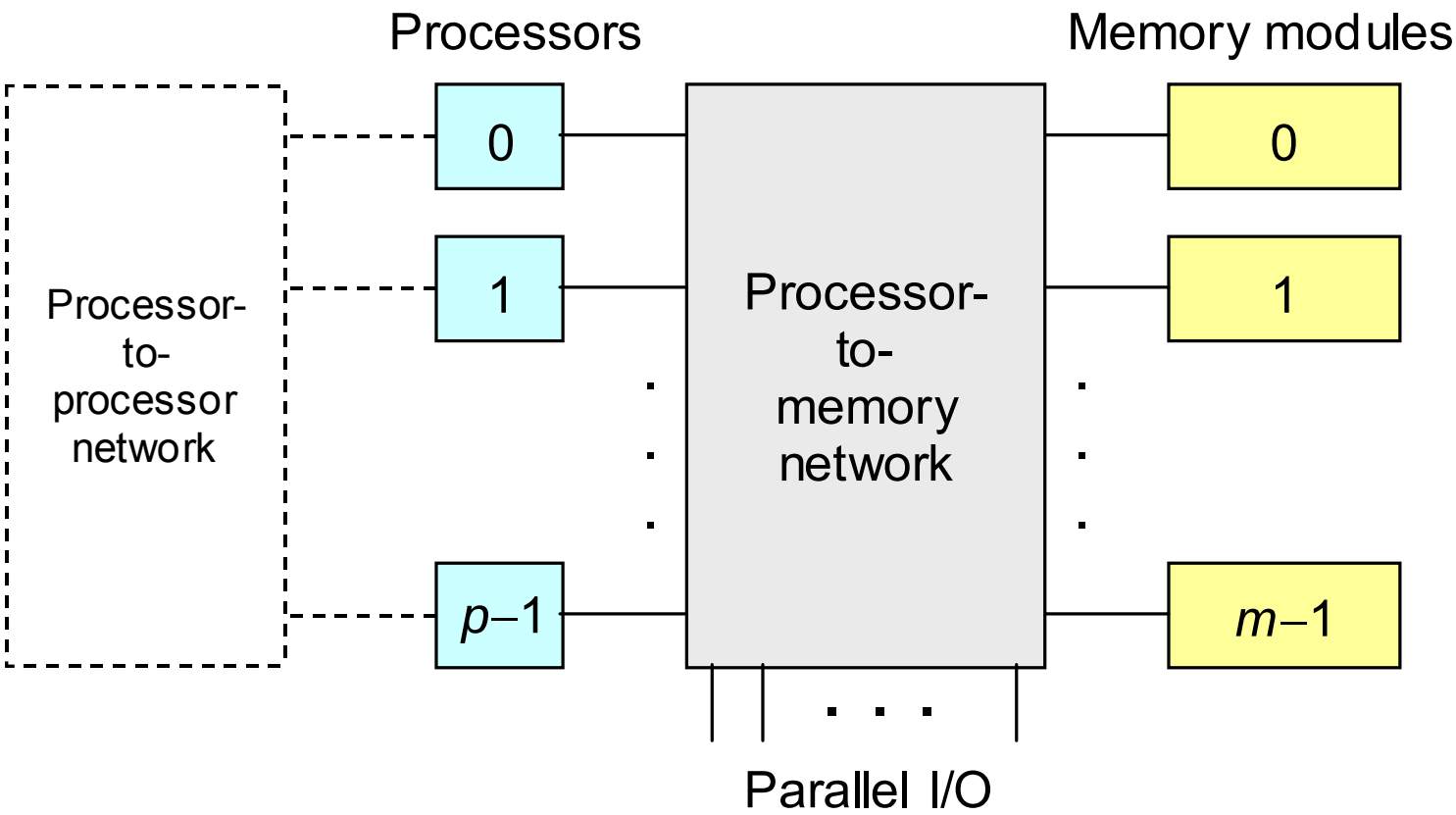


Figure 27.1 Structure of a multiprocessor with centralized shared-memory.

Processor-to-Memory Interconnection Network

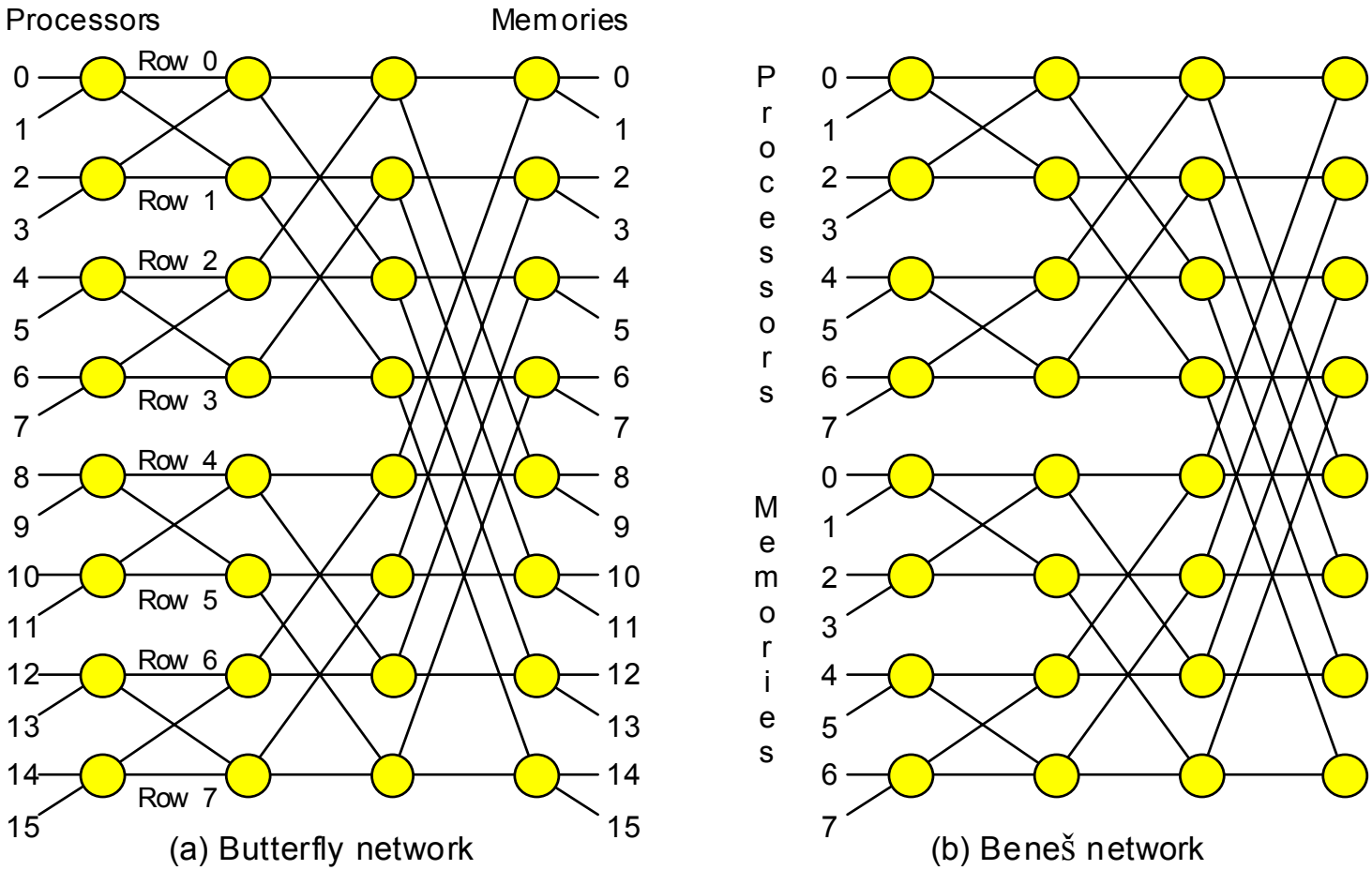


Figure 27.2 Butterfly and the related Beneš network as examples of processor-to-memory interconnection network in a multiprocessor.

Processor-to-Memory Interconnection Network

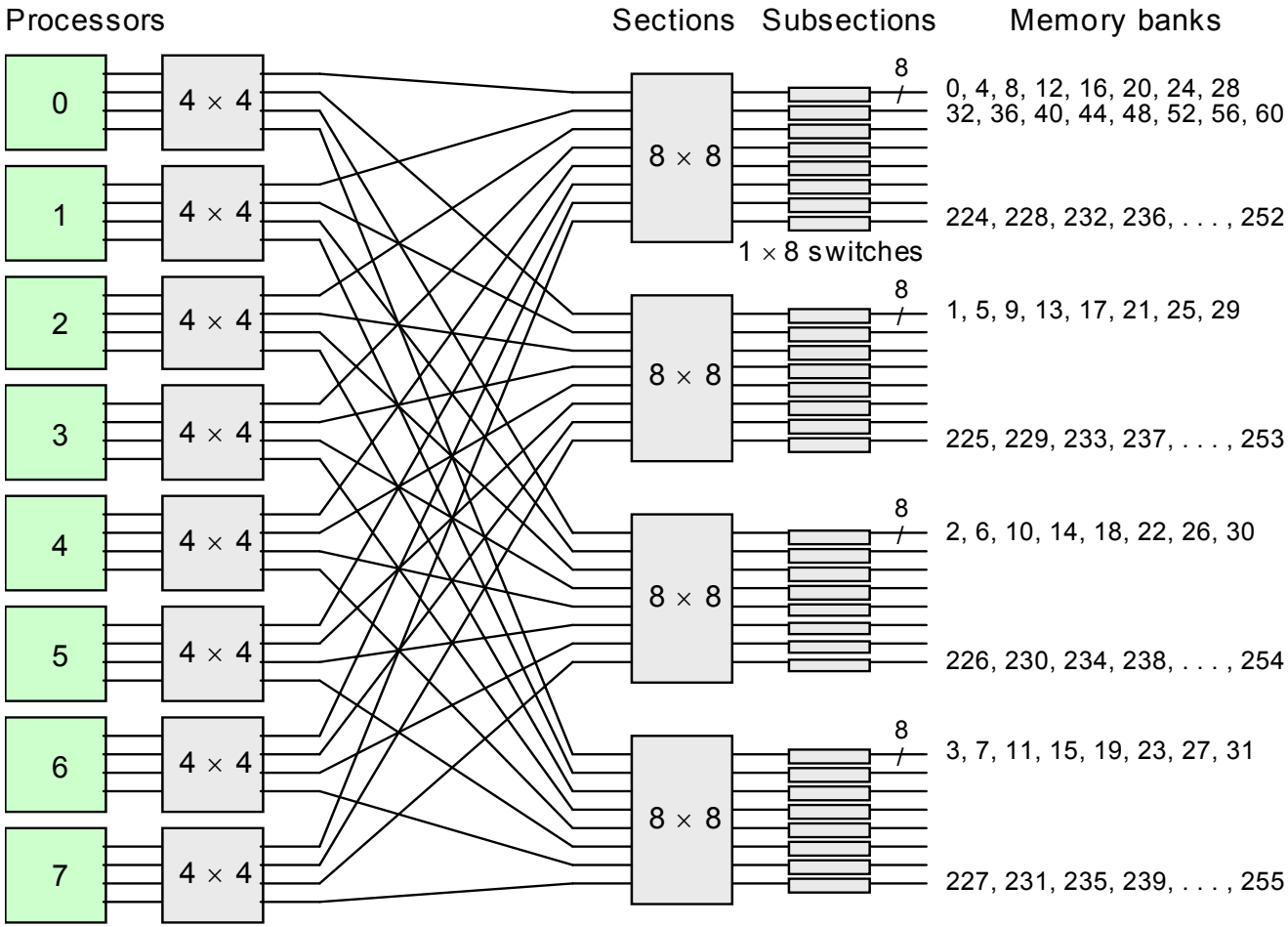


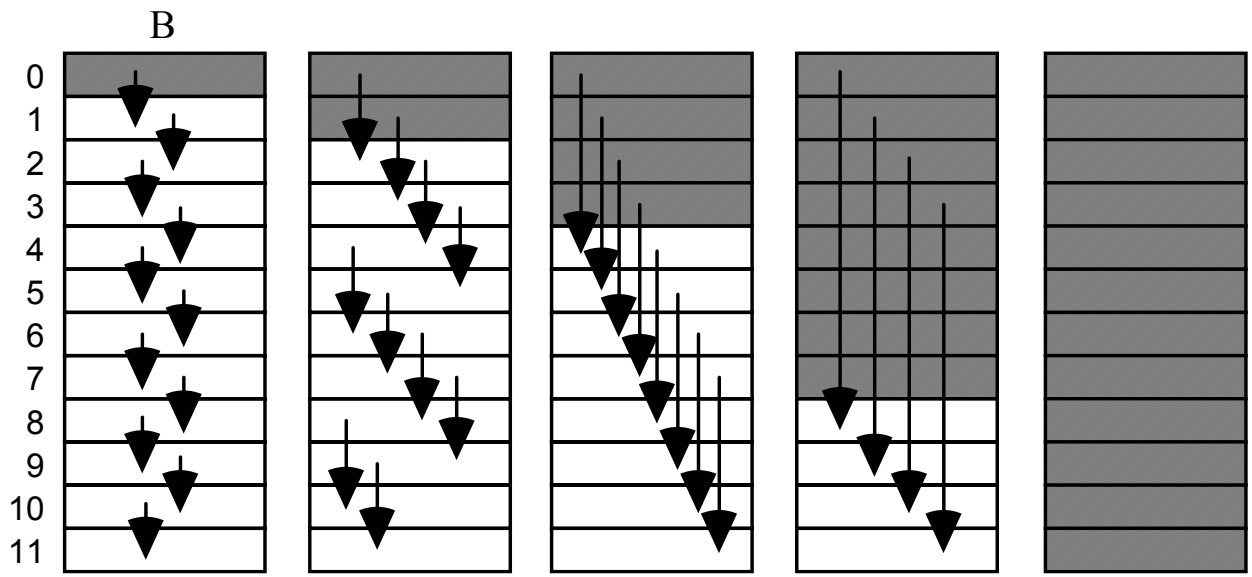
Figure 27.3 Interconnection of eight processors to 256 memory banks in Cray Y-MP, a supercomputer with multiple vector processors.

Shared-Memory Programming: Broadcasting

Copy `B[0]` into all `B[i]` so that multiple processors can read its value without memory access conflicts

```
for k = 0 to ⌈log2 p⌉ – 1 processor j, 0 ≤ j < p, do
    B[j + 2^k] := B[j]
endfor
```

Recursive doubling

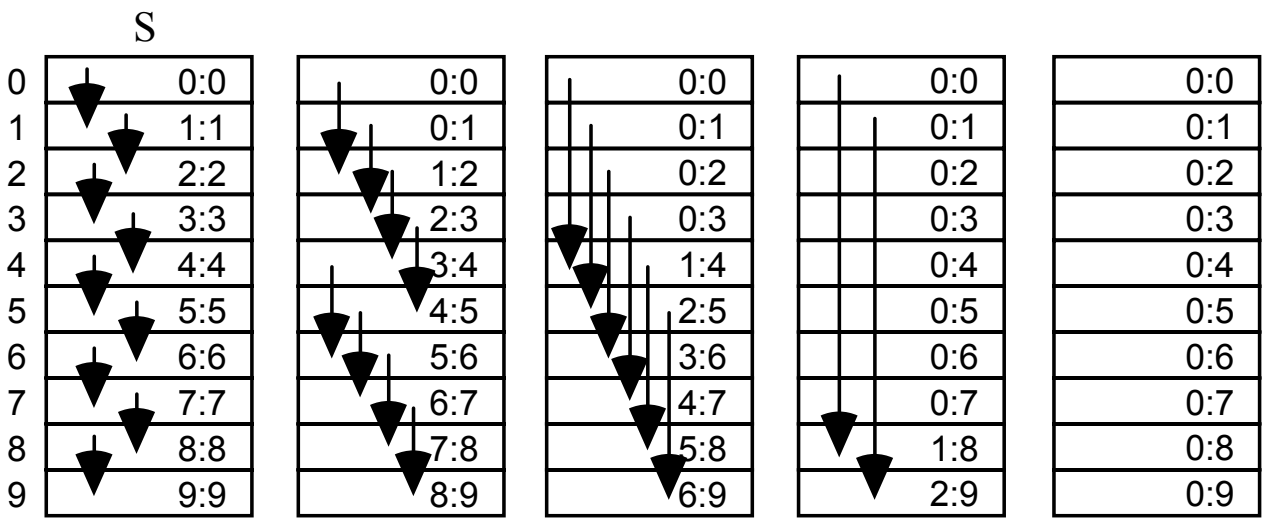


Shared-Memory Programming: Summation

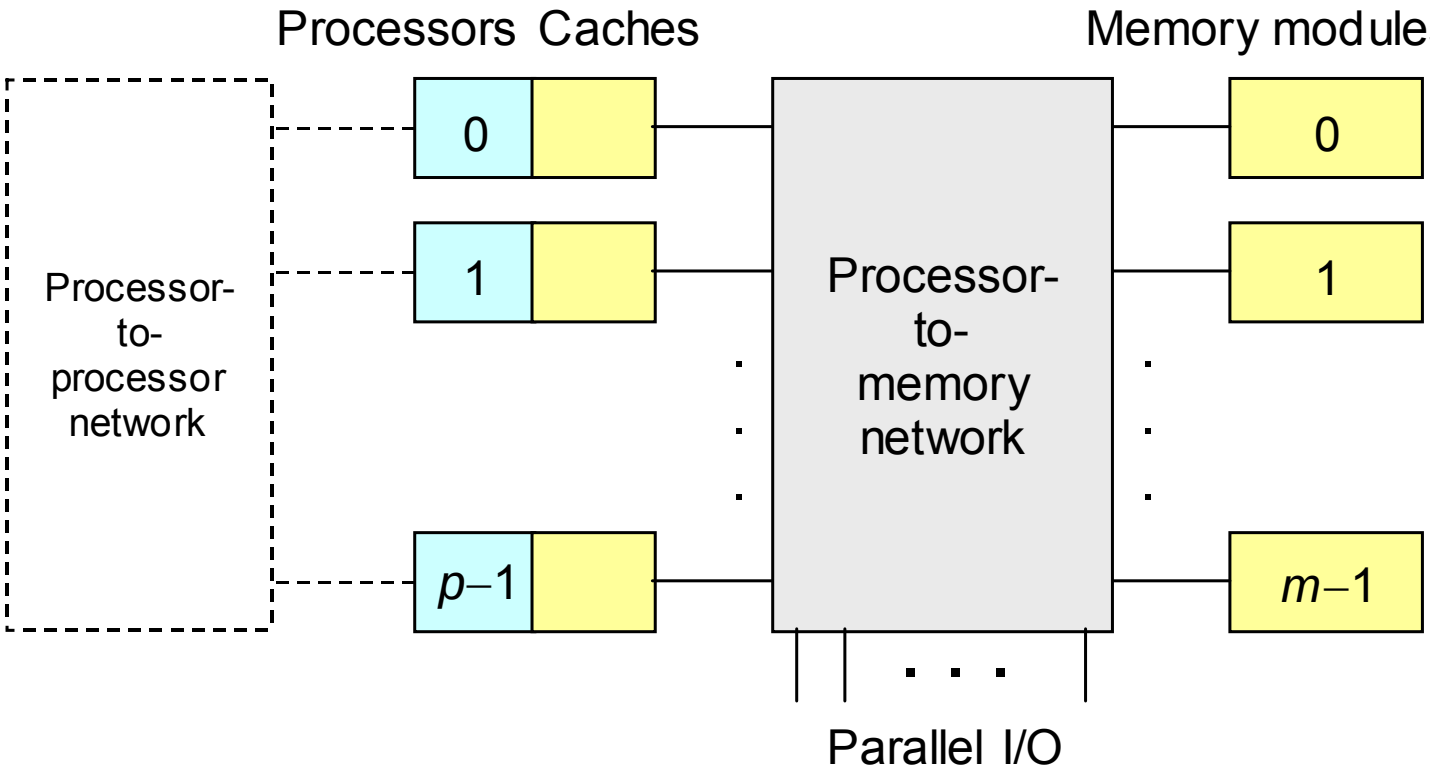
Sum reduction of vector X

```
processor j, 0 ≤ j < p, do Z[j] := X[j]
s := 1
while s < p processor j, 0 ≤ j < p – s, do
  Z[j + s] := X[j] + X[j + s]
  s := 2 × s
endfor
```

Recursive doubling



27.2 Multiple Caches and Cache Coherence



Private processor caches reduce memory access traffic through the interconnection network but lead to challenging consistency problems.

Status of Data Copies

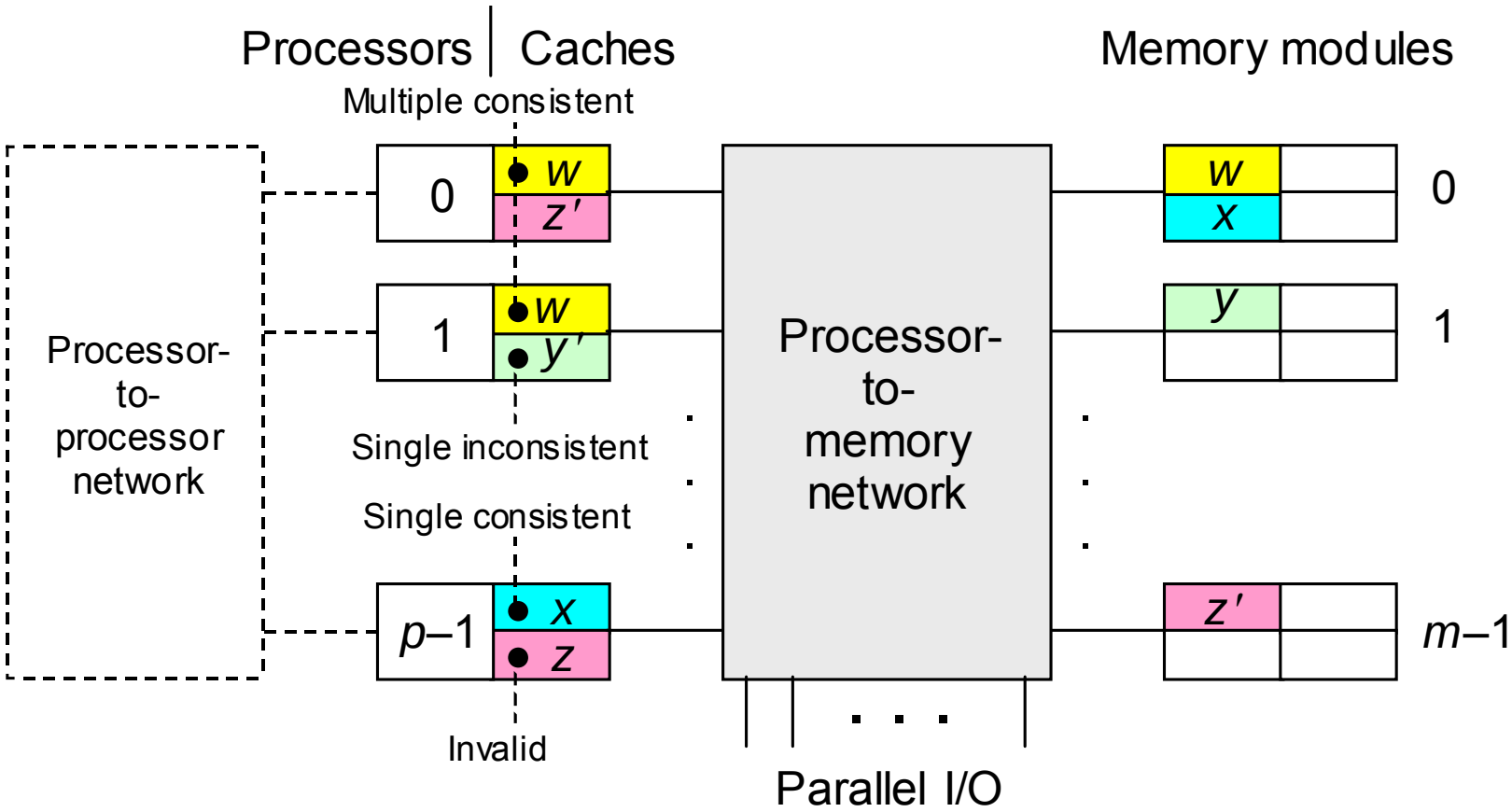
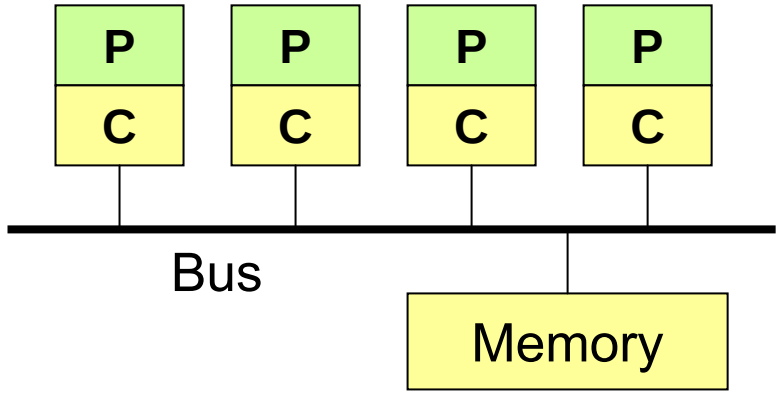


Figure 27.4 Various types of cached data blocks in a parallel processor with centralized main memory and private processor caches.

A Snoopy Cache Coherence Protocol



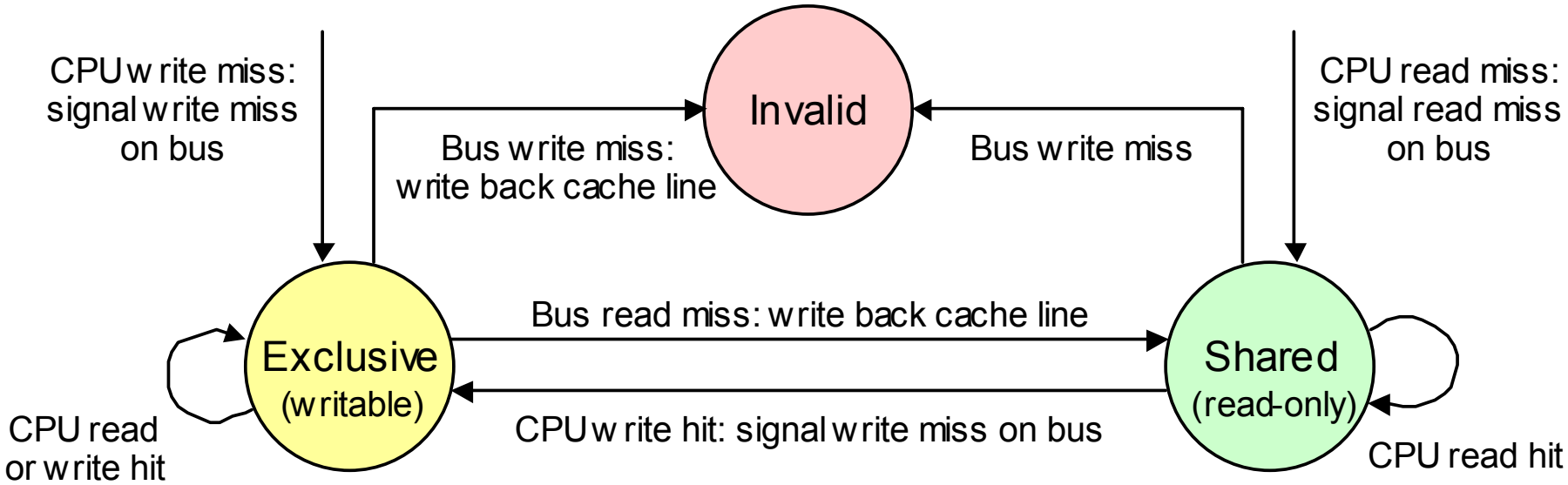


Figure 27.5 Finite-state control mechanism for a bus-based snoopy cache coherence protocol with write-back caches.

27.3 Implementing Symmetric Multiprocessors

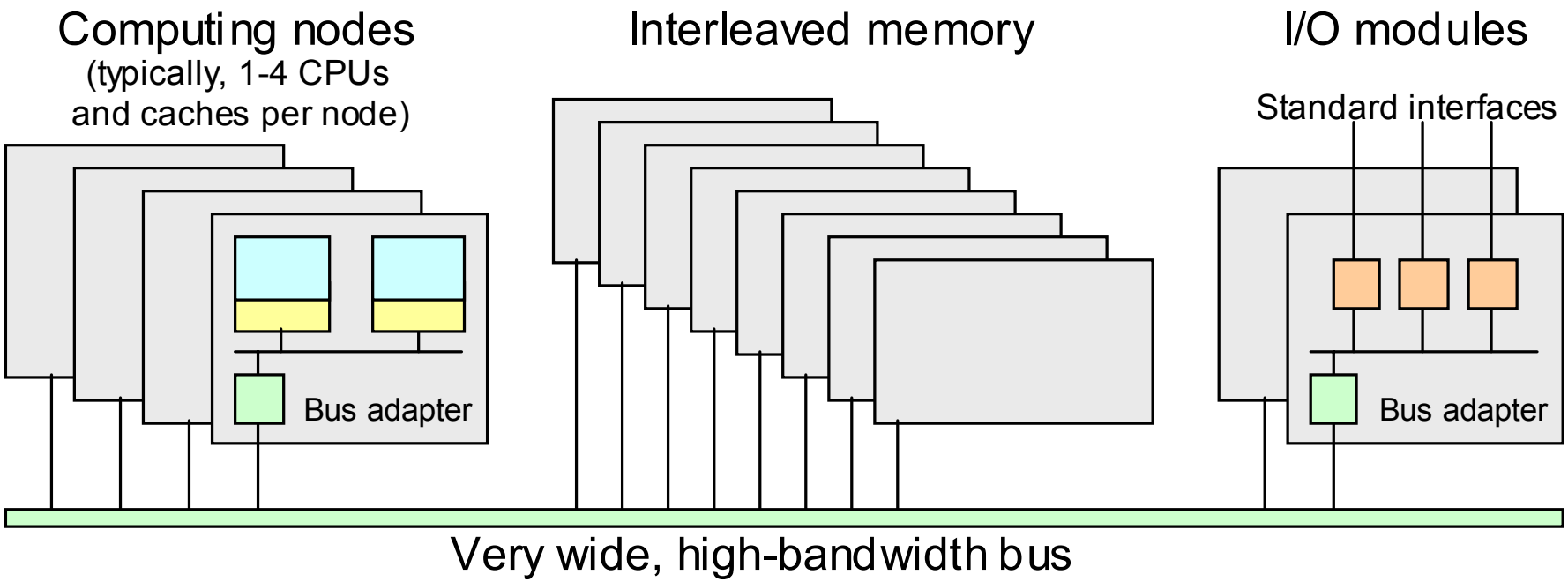


Figure 27.6 Structure of a generic bus-based symmetric multiprocessor.

Bus Bandwidth Limits Performance

Example 27.1

Consider a shared-memory multiprocessor built around a single bus with a data bandwidth of *x* GB/s. Instructions and data words are 4 B wide, each instruction requires access to an average of 1.4 memory words (including the instruction itself). The combined hit rate for caches is 98%. Compute an upper bound on the multiprocessor performance in GIPS. Address lines are separate and do not affect the bus data bandwidth.

Solution

Executing an instruction implies a bus transfer of $1.4 \times 0.02 \times 4 = 0.112$ B. Thus, an absolute upper bound on performance is $x/0.112 = 8.93x$ GIPS. Assuming a bus width of 32 B, no bus cycle or data going to waste, and a bus clock rate of *y* GHz, the performance bound becomes $286y$ GIPS. This bound is highly optimistic. Buses operate in the range 0.1 to 1 GHz. Thus, a performance level approaching 1 TIPS (perhaps even ¼ TIPS) is beyond reach with this type of architecture.

Implementing Snoopy Caches

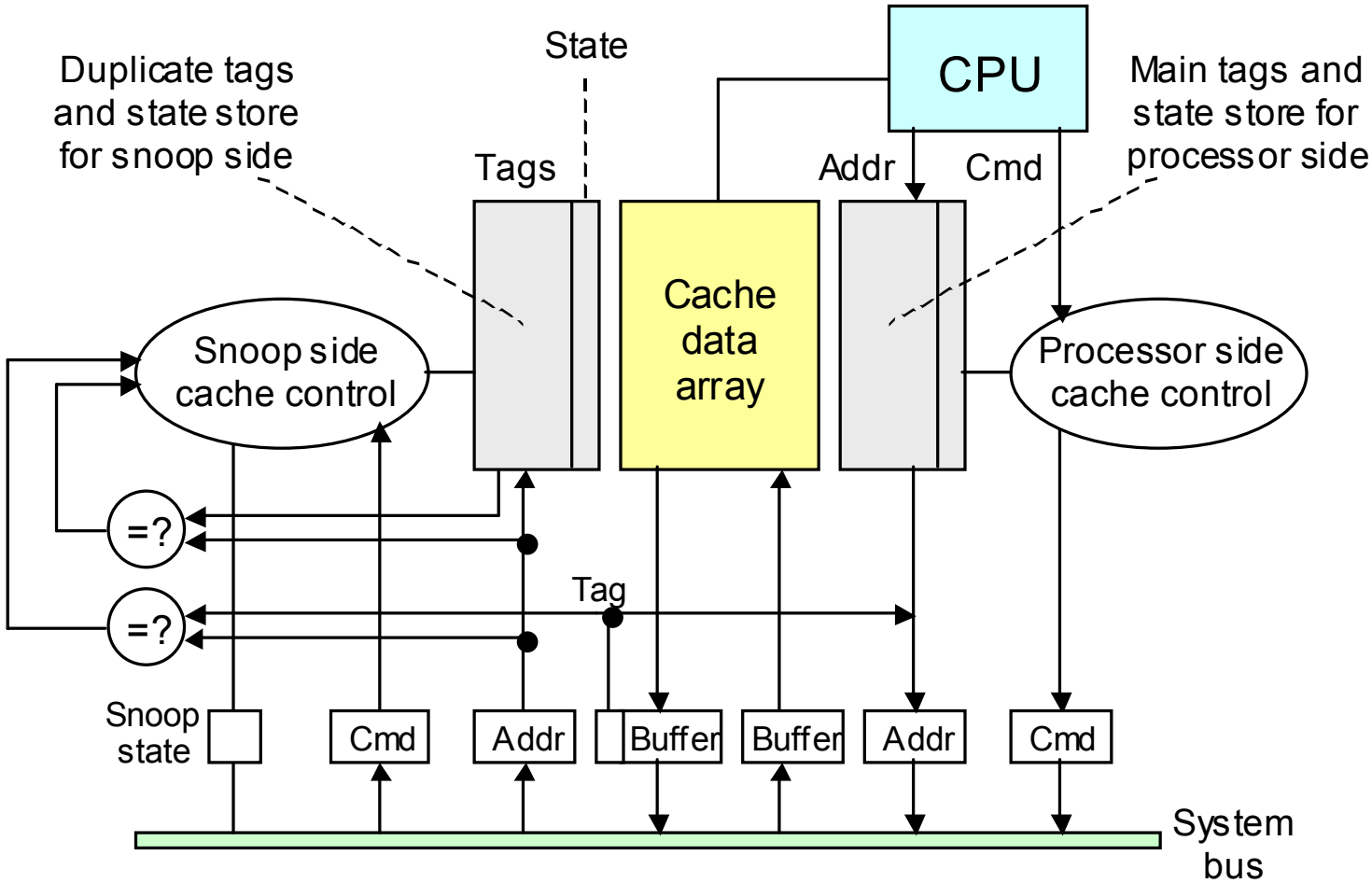


Figure 27.7 Main structure for a snoop-based cache coherence algorithm.

27.4 Distributed Shared Memory

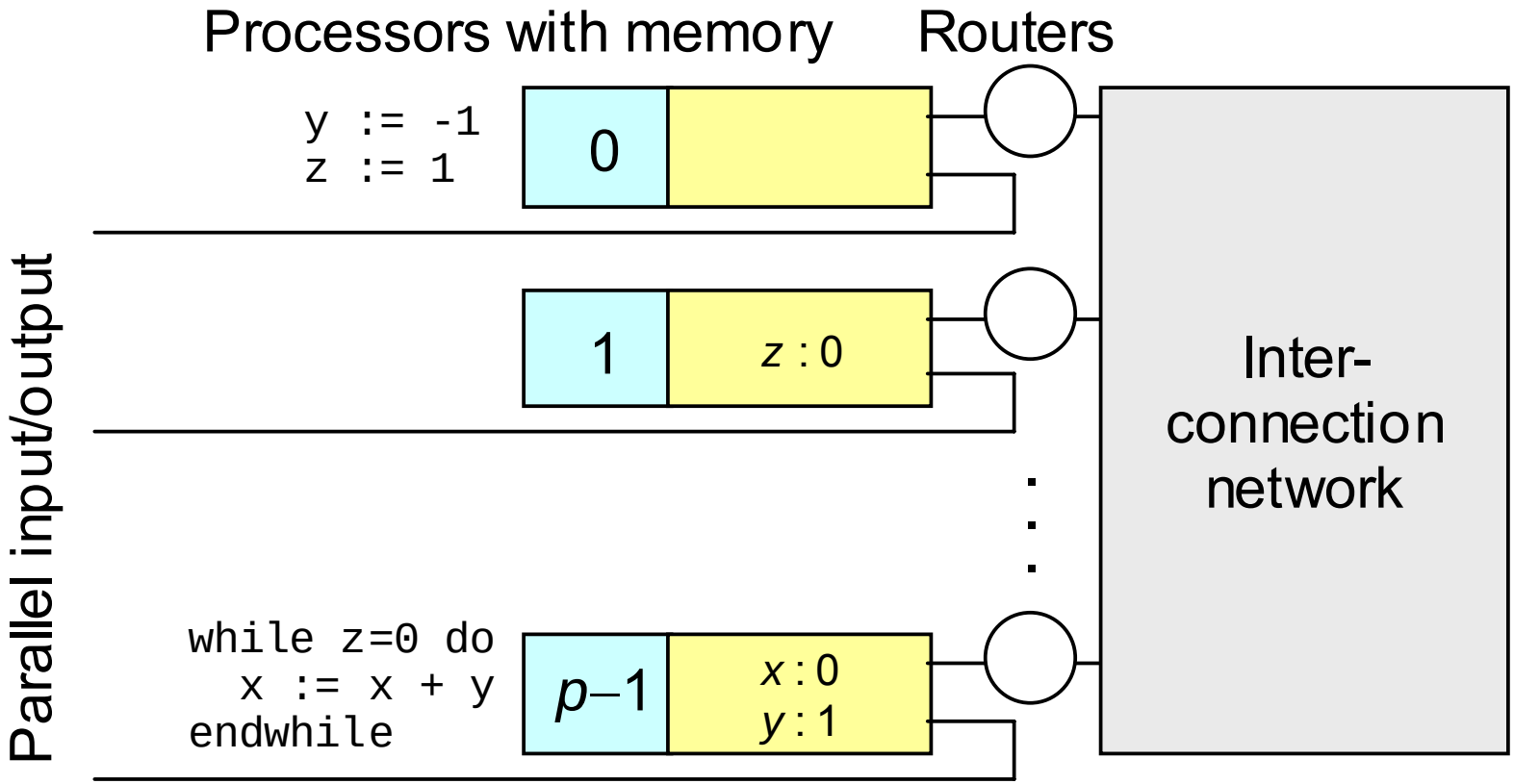


Figure 27.8 Structure of a distributed shared-memory multiprocessor.

27.5 Directories to Guide Data Access

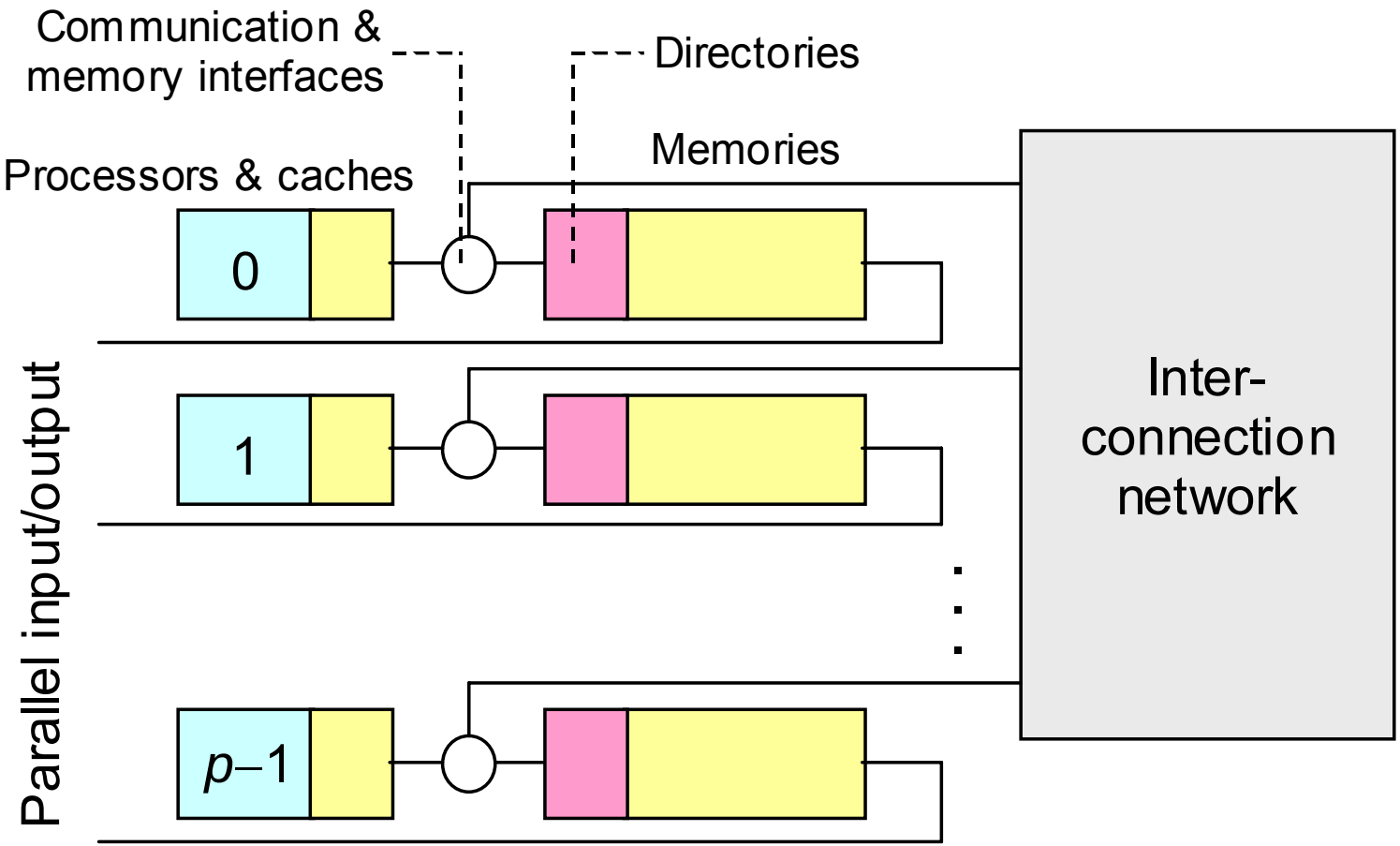


Figure 27.9 Distributed shared-memory multiprocessor with a cache, directory, and memory module associated with each processor.

Directory-Based Cache Coherence

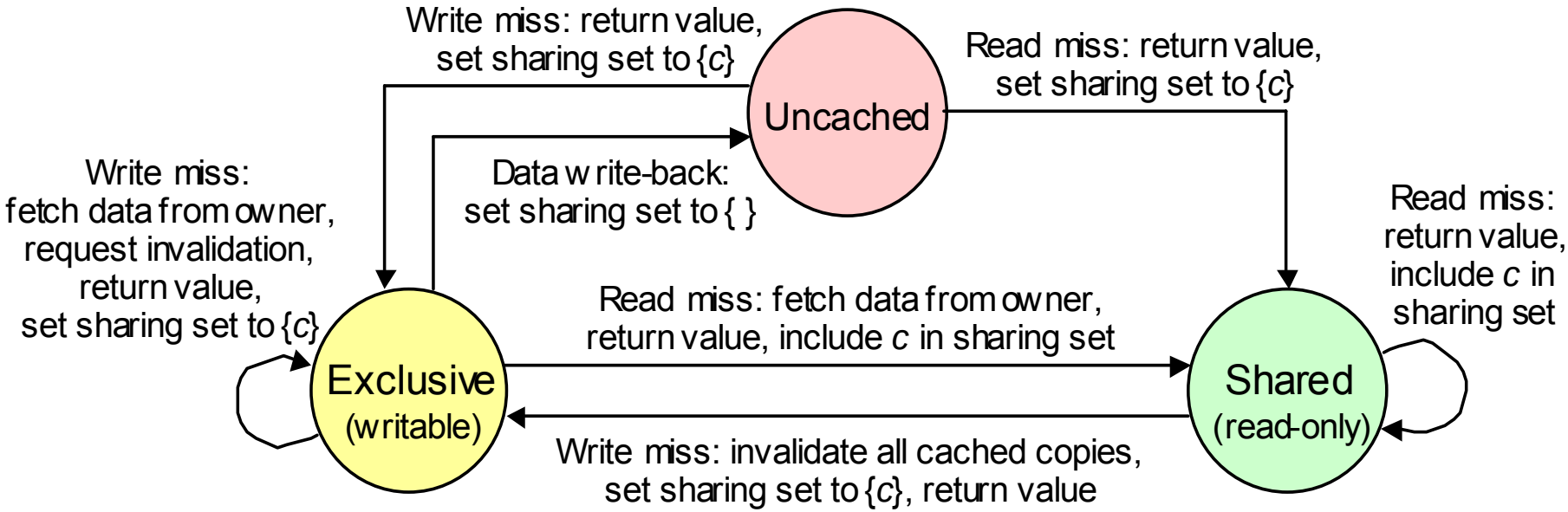


Figure 27.10 States and transitions for a directory entry in a directory-based cache coherence protocol (*c* is the requesting cache).

27.6 Implementing Asymmetric Multiprocessors

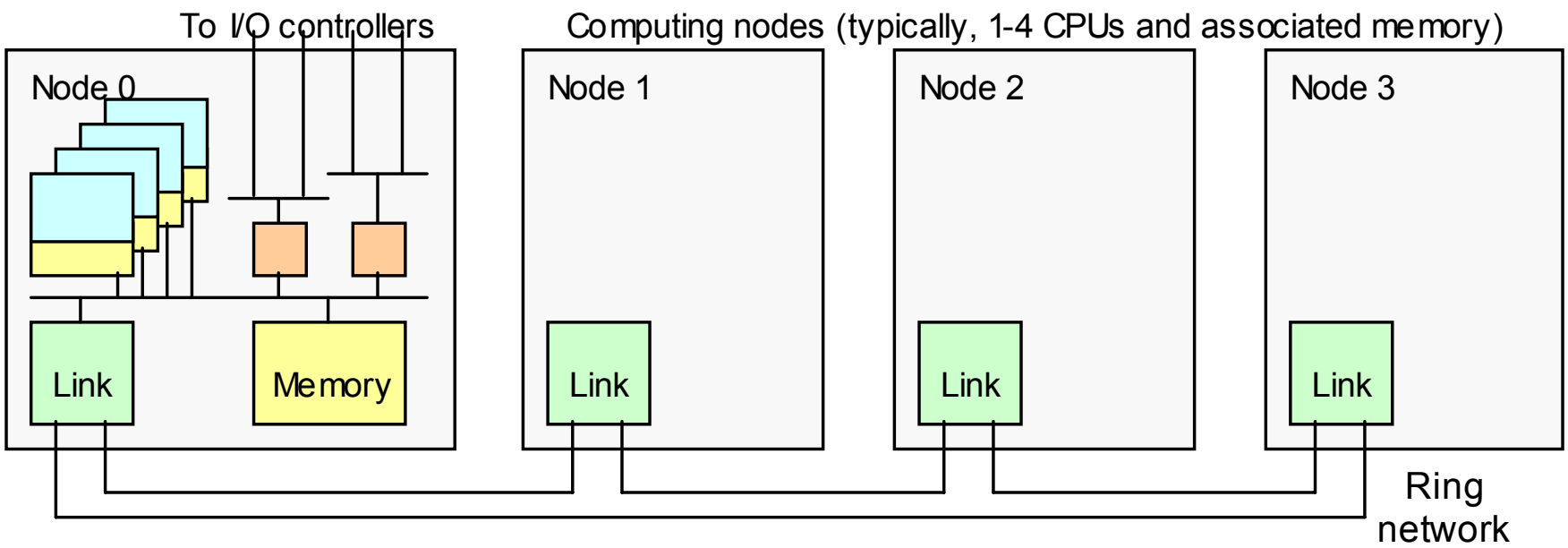


Figure 27.11 Structure of a ring-based distributed-memory multiprocessor.

Scalable Coherent Interface (SCI)

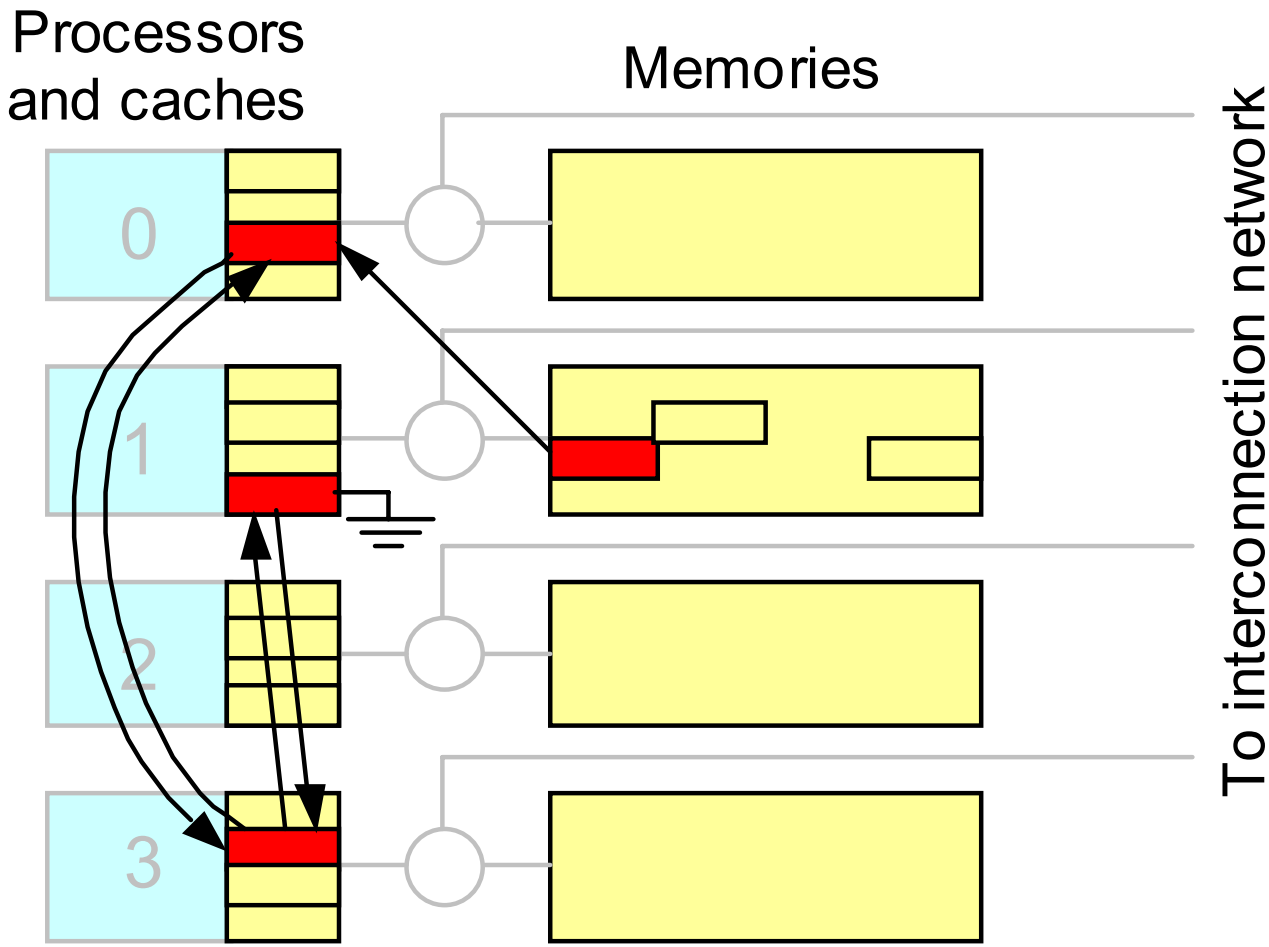


Figure 27.11 Structure of a ring-based distributed-memory multiprocessor.

28 Distributed Multicomputing

Computer architects' dream: connect computers like toy blocks

- Building multicomputers from loosely connected nodes
- Internode communication is done via message passing

Topics in This Chapter	
28.1	Communication by Message Passing
28.2	Interconnection Networks
28.3	Message Composition and Routing
28.4	Building and Using Multicomputers
28.5	Network-Based Distributed Computing
28.6	Grid Computing and Beyond

28.1 Communication by Message Passing

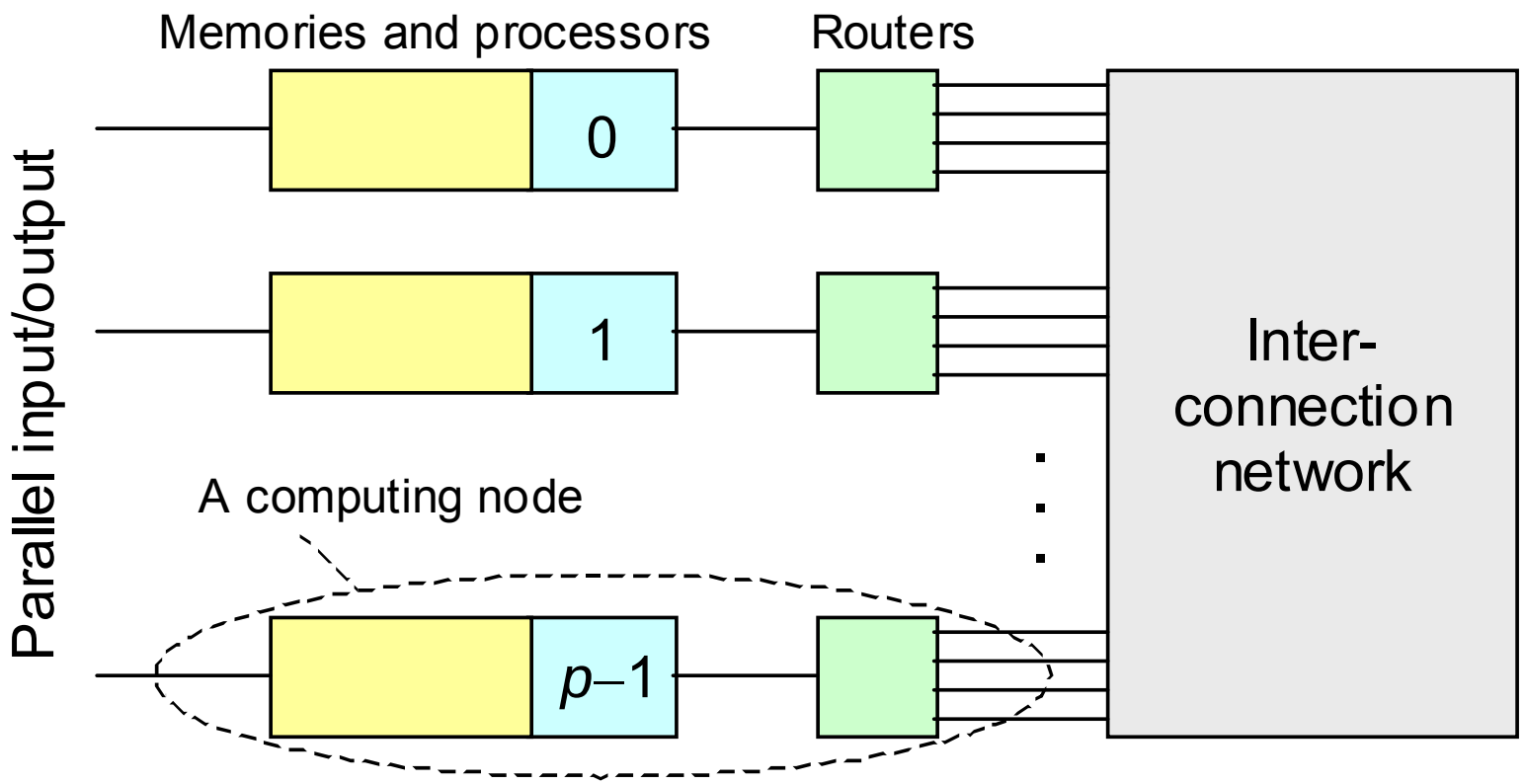


Figure 28.1 Structure of a distributed multicomputer.

Router Design

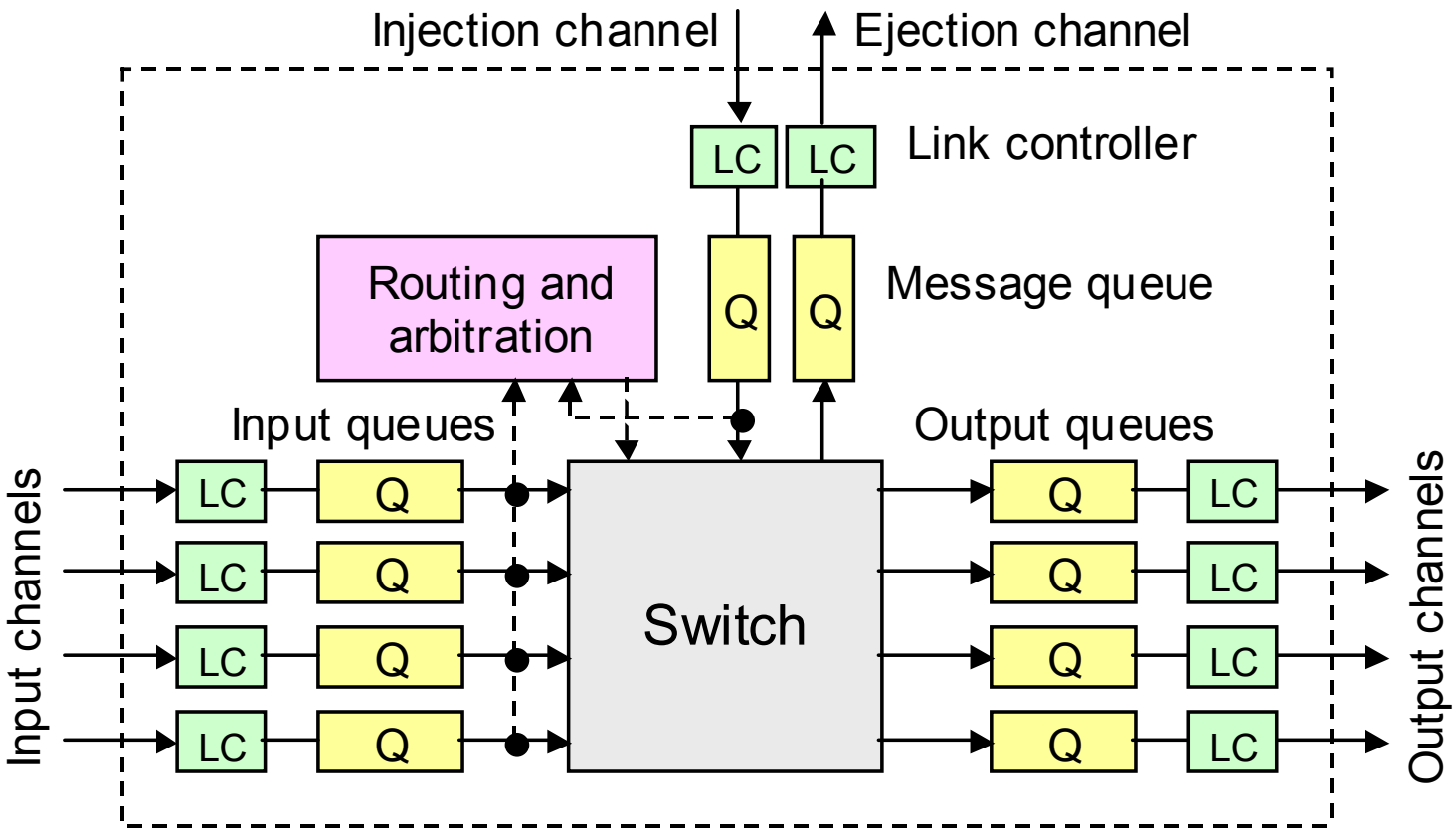


Figure 28.2 The structure of a generic router.

Building Networks from Switches

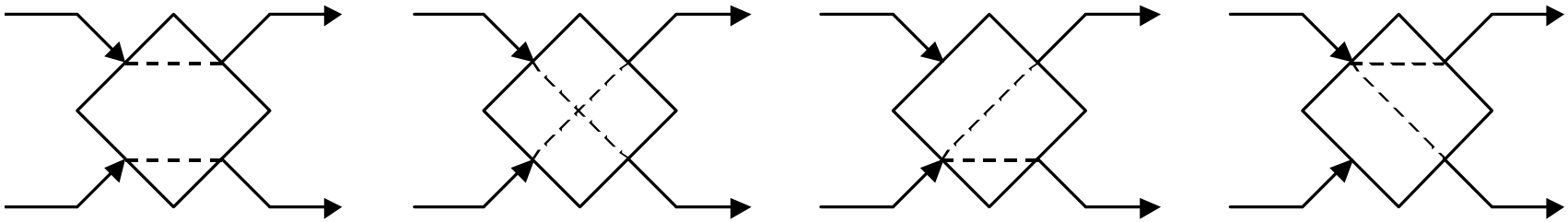
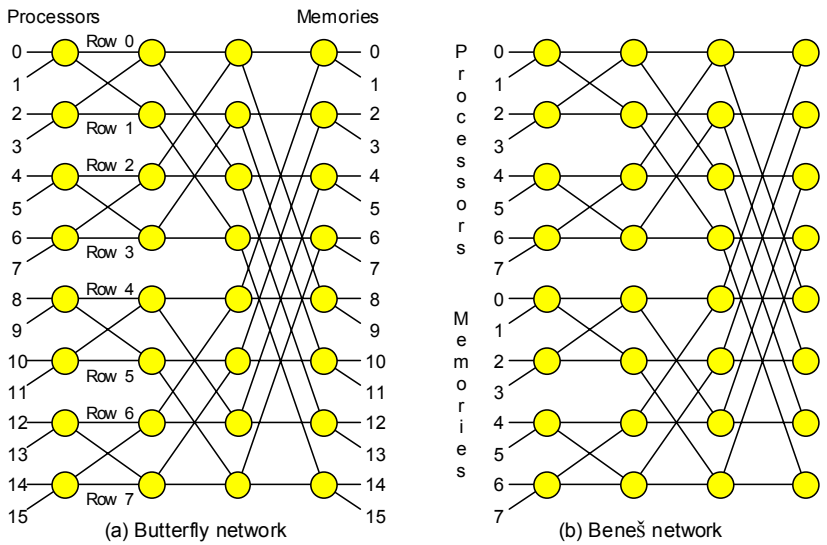


Figure 28.3 Example 2 × 2 switch with point-to-point and broadcast connection capabilities.

Figure 27.2
Butterfly and
Beneš networks



Interprocess Communication via Messages

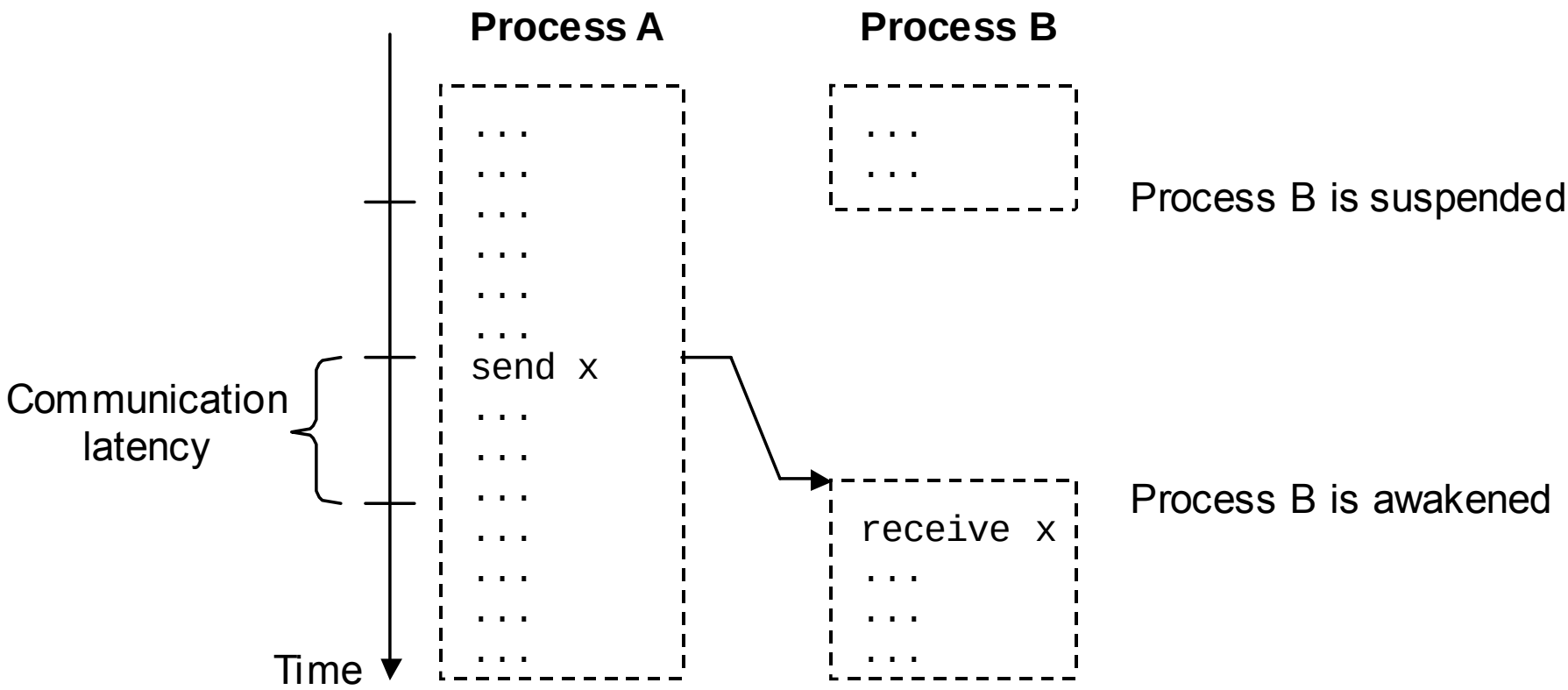


Figure 28.4 Use of send and receive message-passing primitives to synchronize two processes.

28.2 Interconnection Networks

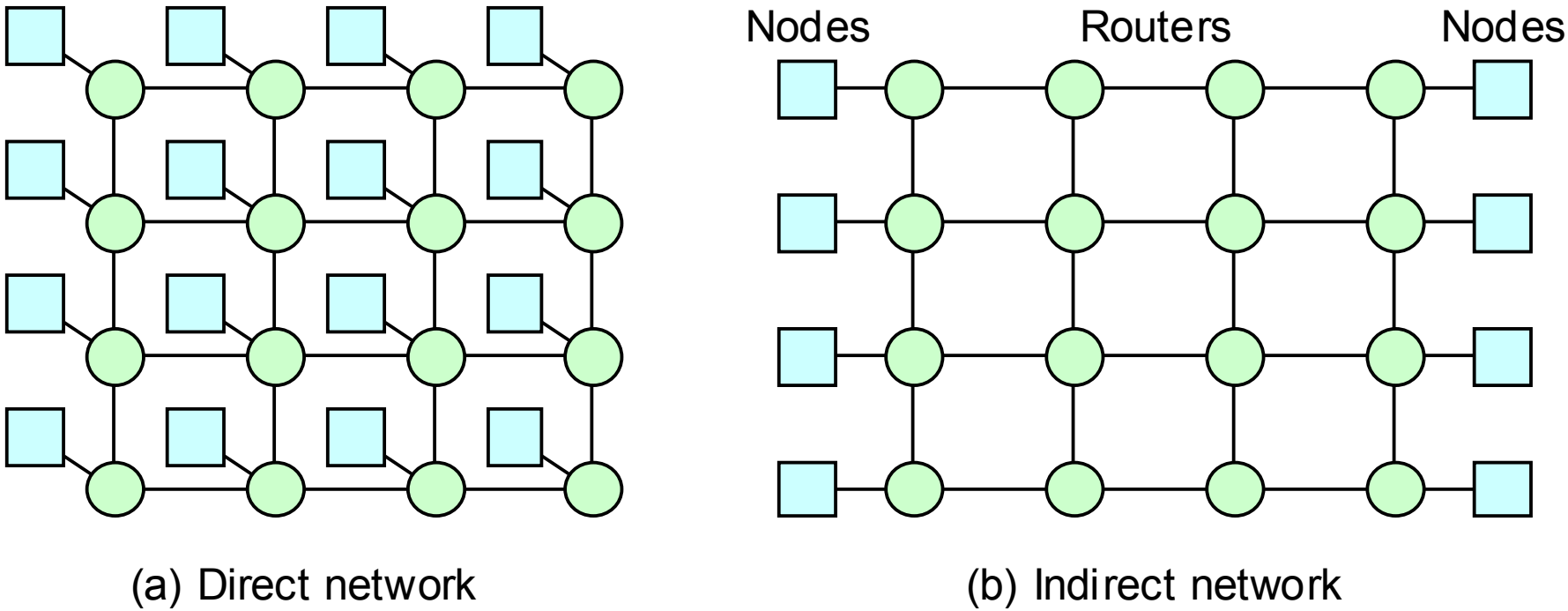
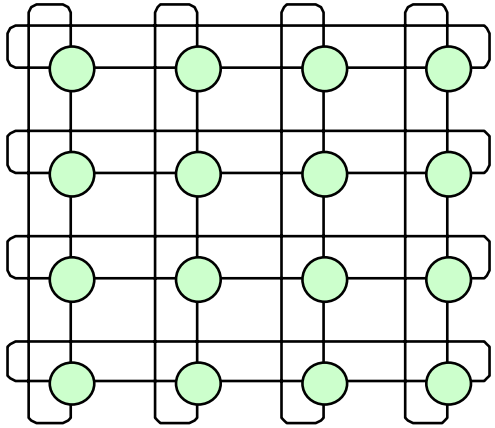
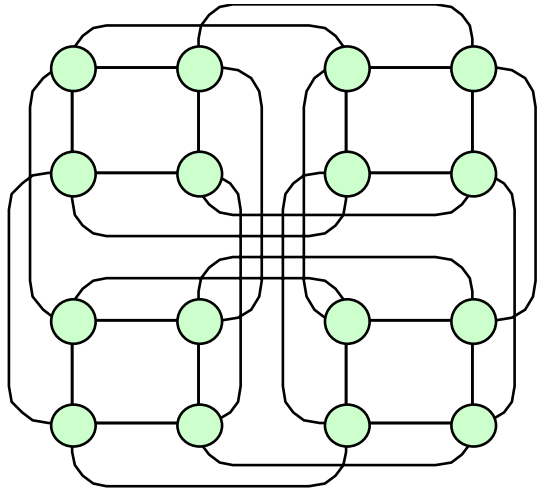


Figure 28.5 Examples of direct and indirect interconnection networks.

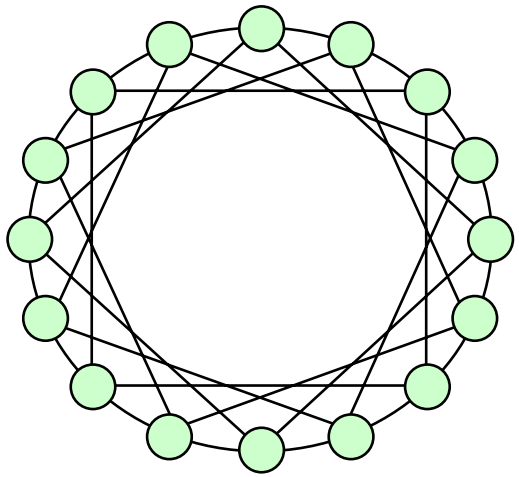
Direct Interconnection Networks



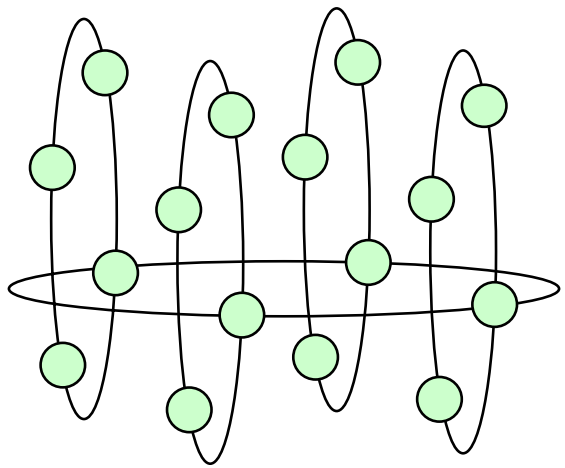
(a) 2D torus



(b) 4D hypercube



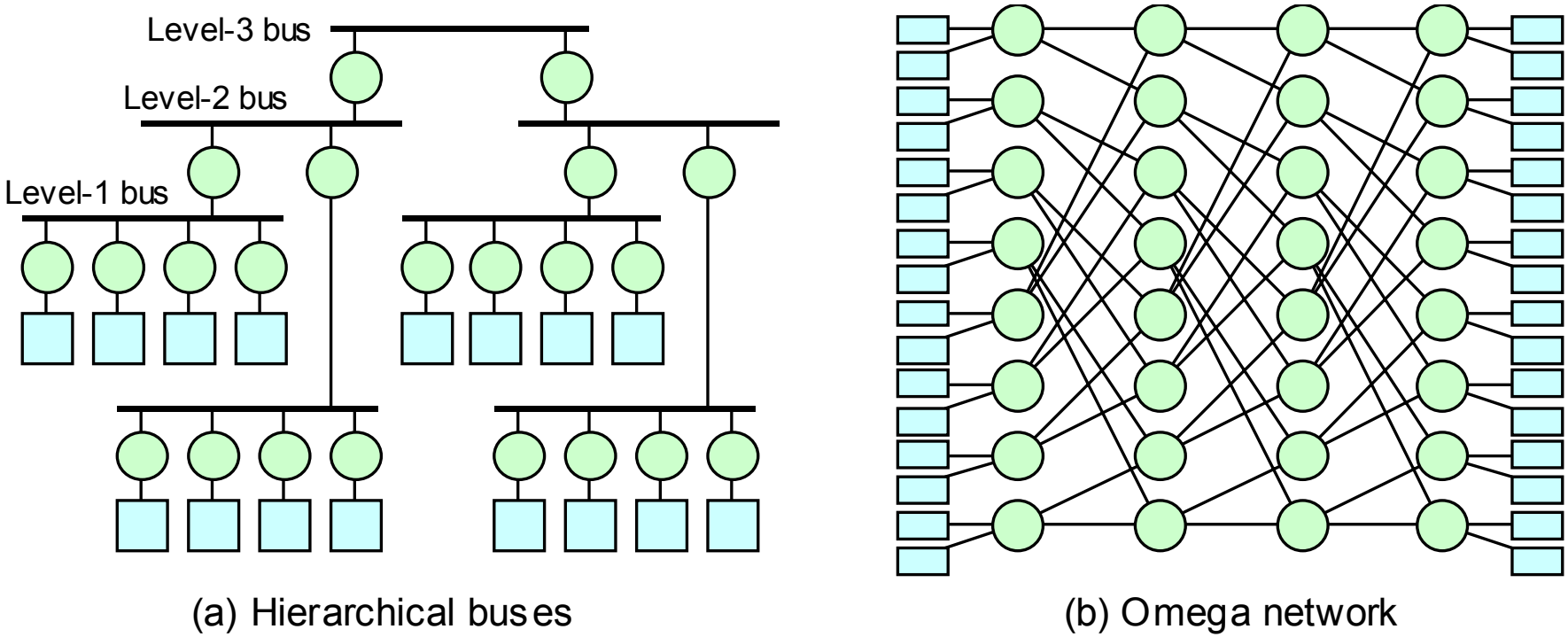
(c) Chordal ring



(d) Ring of rings

Figure 28.6 A sampling of common direct interconnection networks. Only routers are shown; a computing node is implicit for each router.

Indirect Interconnection Networks



(a) Hierarchical buses

(b) Omega network

Figure 28.7 Two commonly used indirect interconnection networks.

28.3 Message Composition and Routing

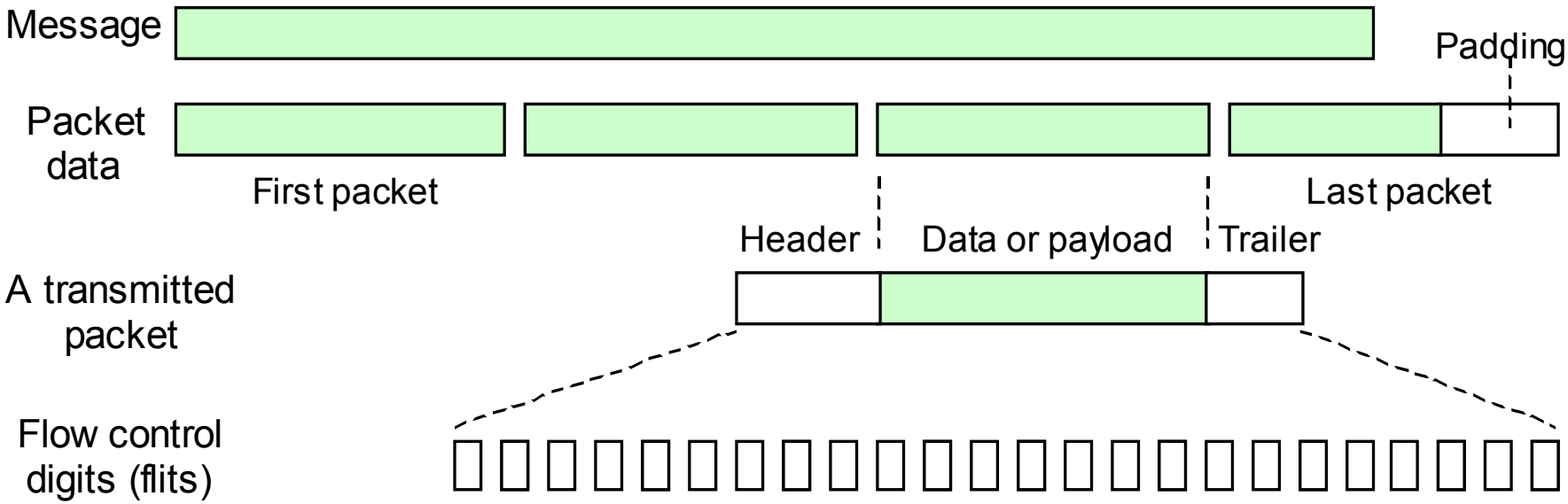
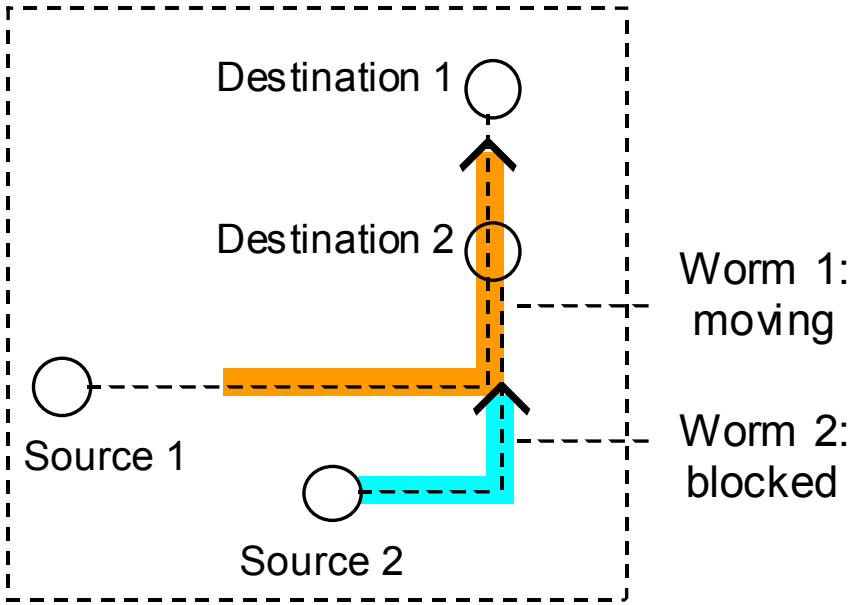
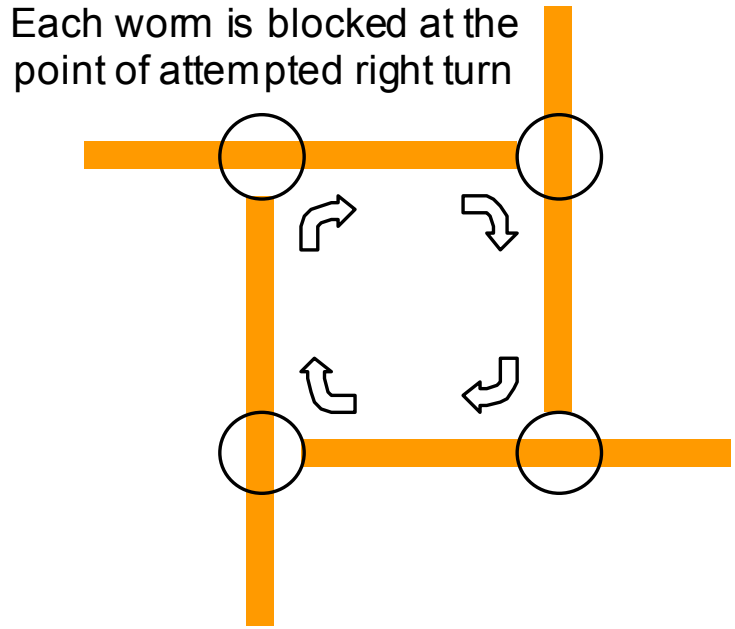


Figure 28.8 Messages and their parts for message passing.

Wormhole Switching



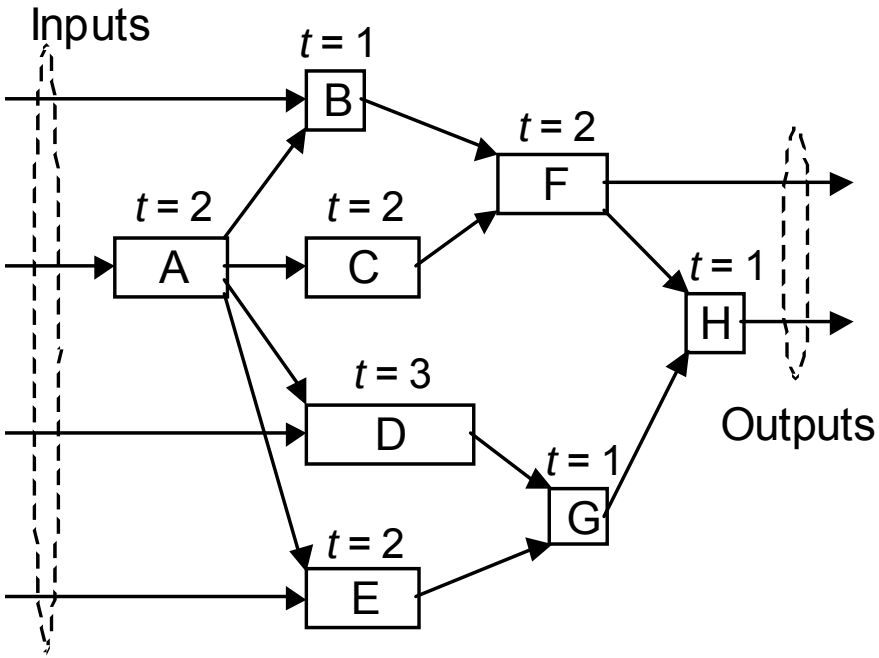
(a) Two worms en route to their respective destinations



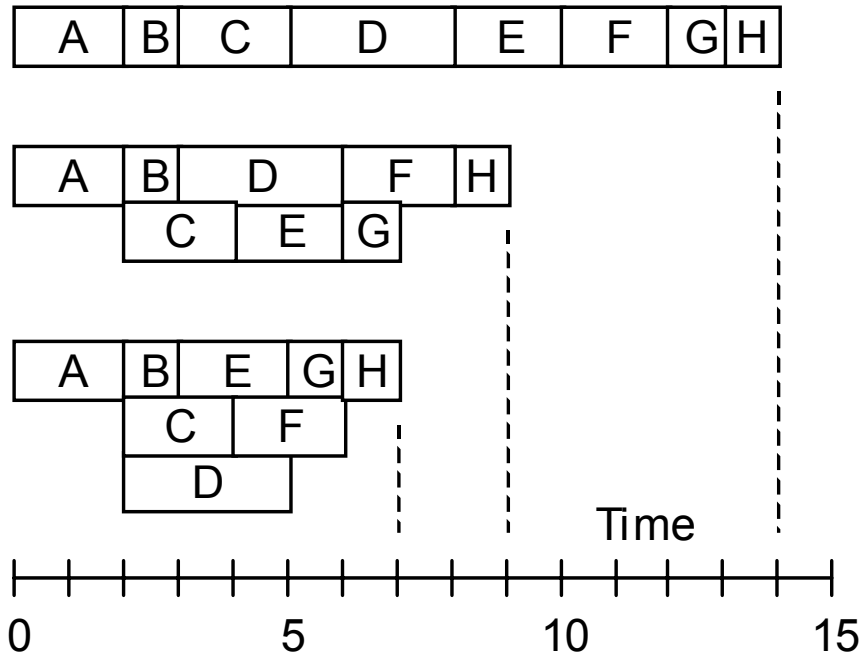
(b) Deadlock due to circular waiting of four blocked worms

Figure 28.9 Concepts of wormhole switching.

28.4 Building and Using Multicomputers



(a) Static task graph



(b) Schedules on 1-3 computers

Figure 28.10 A task system and schedules on 1, 2, and 3 computers.

Building Multicomputers from Commodity Nodes

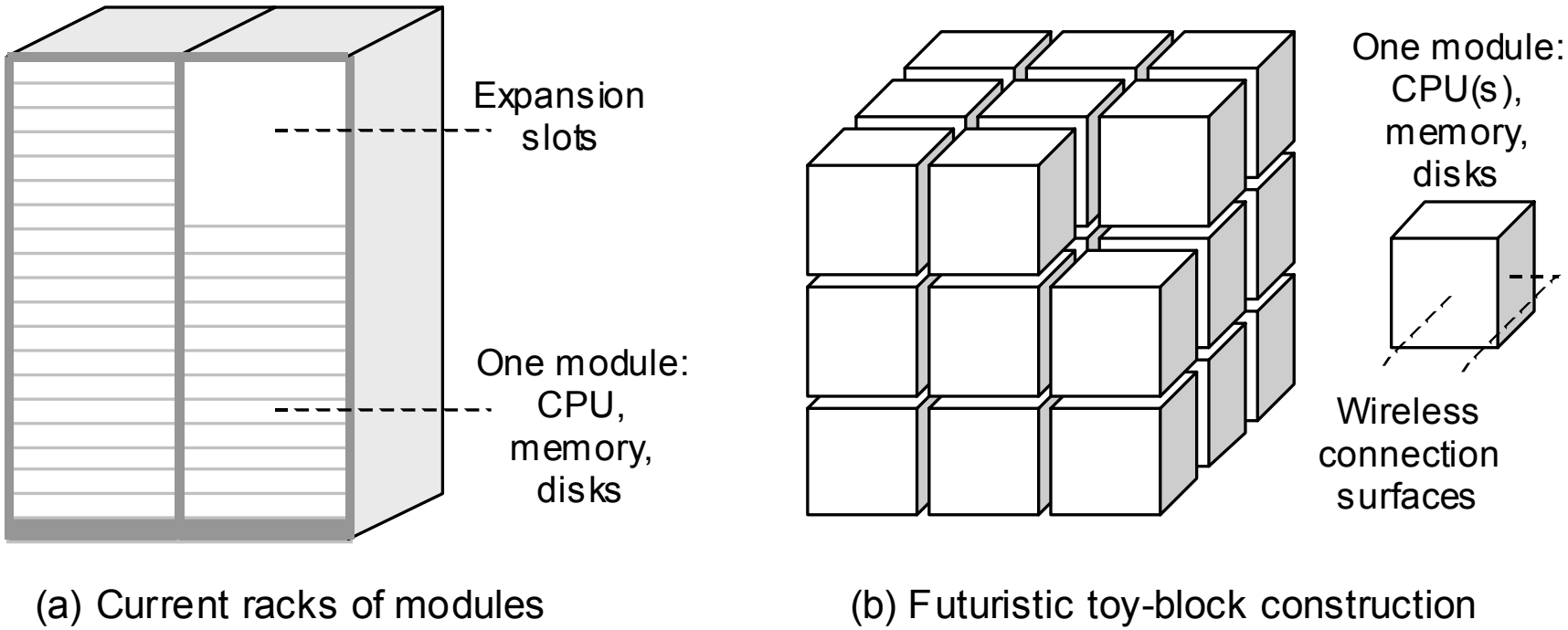


Figure 28.11 Growing clusters using modular nodes.

28.5 Network-Based Distributed Computing

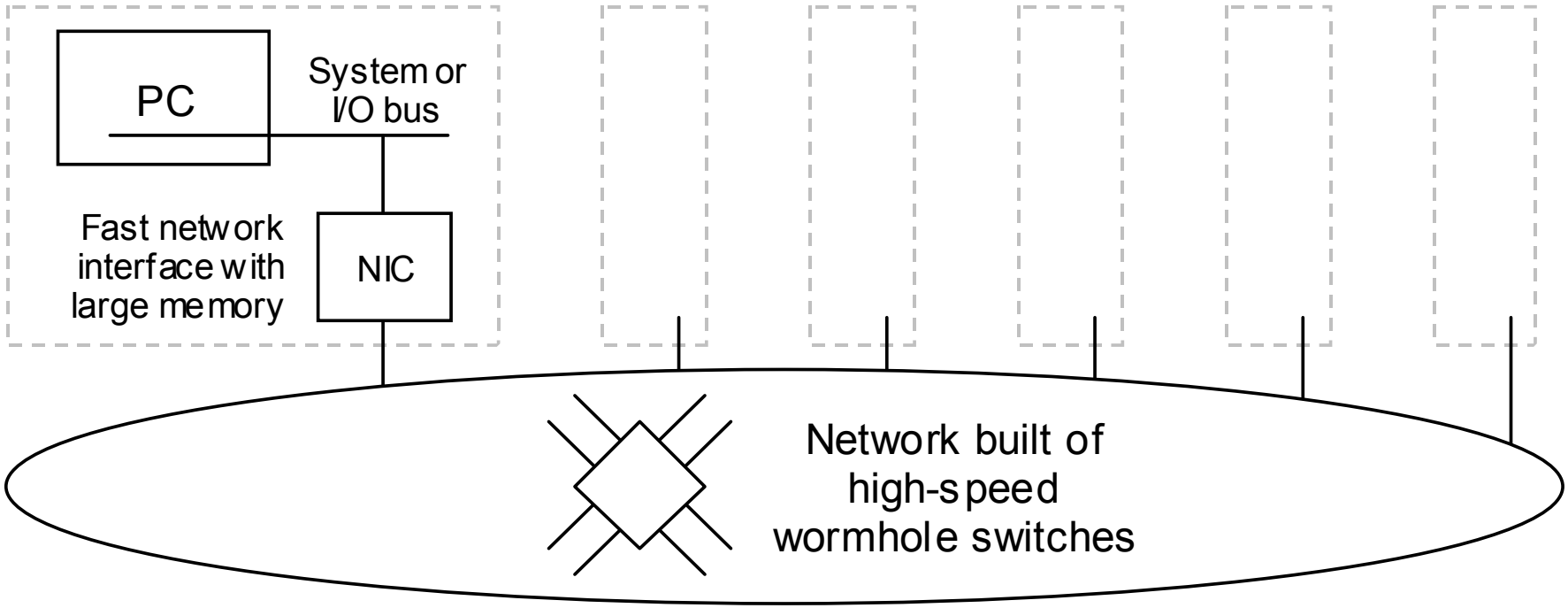


Figure 28.12 Network of workstations.

28.6 Grid Computing and Beyond

Computational grid is analogous to the power grid

Decouples the “production” and “consumption” of computational power

Homes don’t have an electricity generator; why should they have a computer?

Advantages of computational grid:

Near continuous availability of computational and related resources
Resource requirements based on sum of averages, rather than sum of peaks
Paying for services based on actual usage rather than peak demand
Distributed data storage for higher reliability, availability, and security
Universal access to specialized and one-of-a-kind computing resources

Still to be worked out: How to charge for computation usage