William Stallings
Computer Organization and Architecture
9th Edition

Chapter 17

Parallel Processing

17.1 Multiple Processor Organization

Types of Parallel Processor Systems

- Single instruction, single data **(SISD)** stream
  - Single processor executes a single instruction stream to operate on data stored in a single memory
  - Uniprocessors fall into this category
- Single instruction, multiple data **(SIMD)** stream
  - A single machine instruction controls the simultaneous execution of a number of processing elements on a lockstep basis
  - Vector and array processors fall into this category
- Multiple instruction, single data **(MISD)** stream
  - A sequence of data is transmitted to a set of processors, each of which executes a different instruction sequence
  - Not commercially implemented
- Multiple instruction, multiple data **(MIMD)** stream
  - A set of processors simultaneously execute different instruction sequences on different data sets
  - SMPs, clusters and NUMA systems fit this category

Processor Organizations

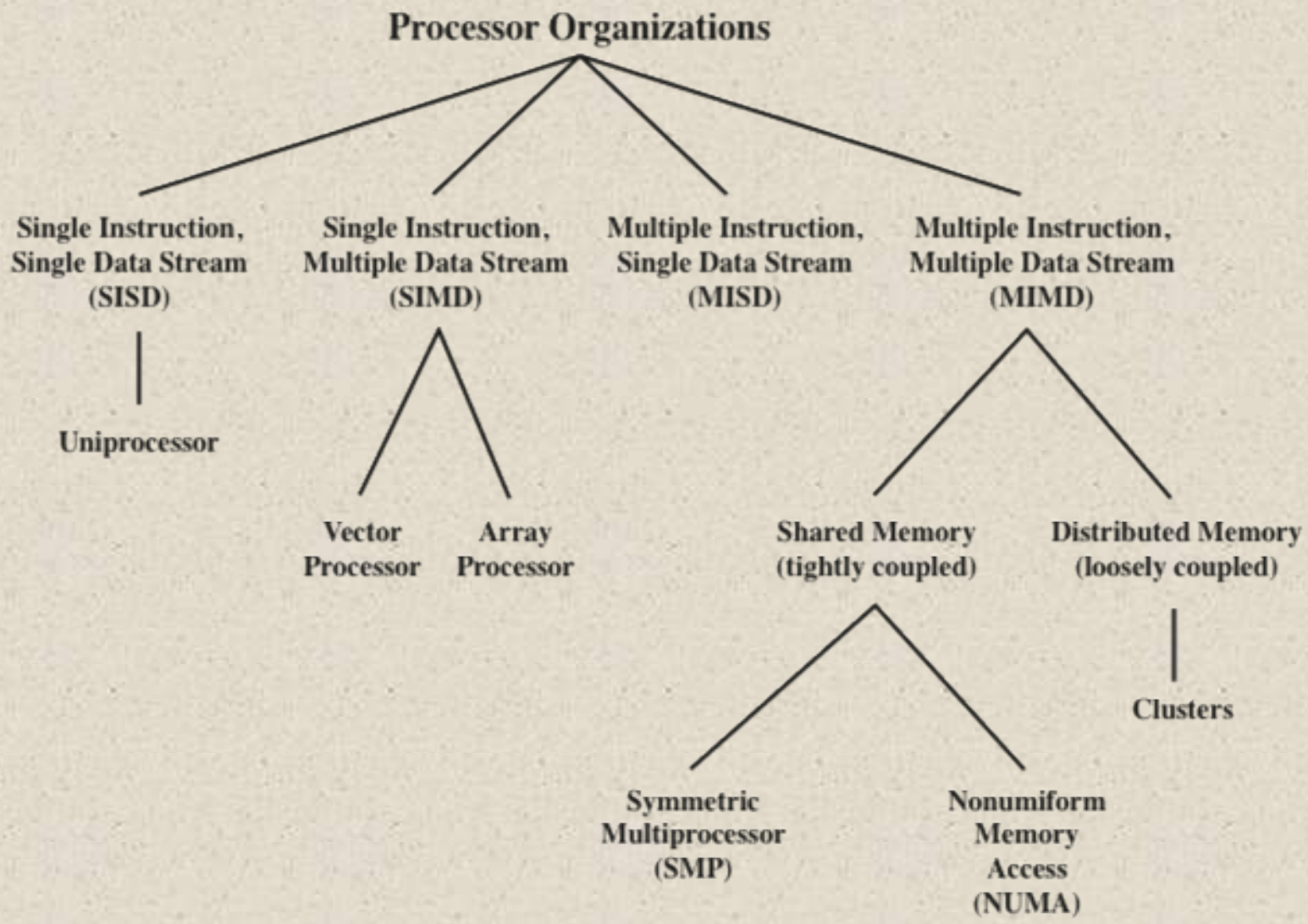


Figure 17.1 A Taxonomy of Parallel Processor Architectures

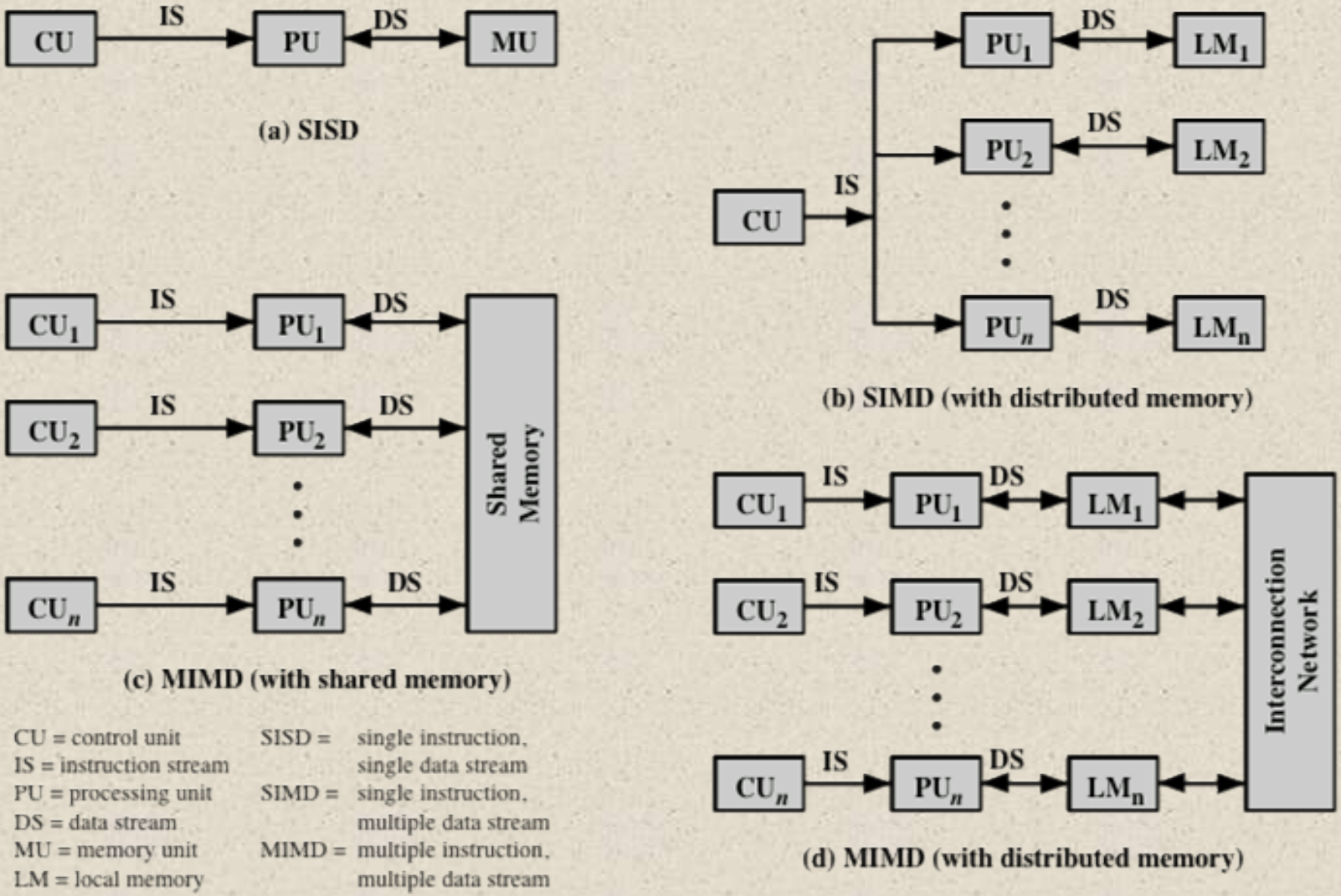


Figure 17.2 Alternative Computer Organizations

17.2 Symmetric Multiprocessor (SMP)

A stand alone computer with the following characteristics:

Two or more similar processors of comparable capacity	Processors share same memory and I/O facilities • Processors are connected by a bus or other internal connection • Memory access time is approximately the same for each processor	All processors share access to I/O devices • Either through same channels or different channels giving paths to same devices	All processors can perform the same functions (hence "symmetric")	System controlled by integrated operating system • Provides interaction between processors and their programs at job, task, file and data element levels

Multiprogramming and Multiprocessing

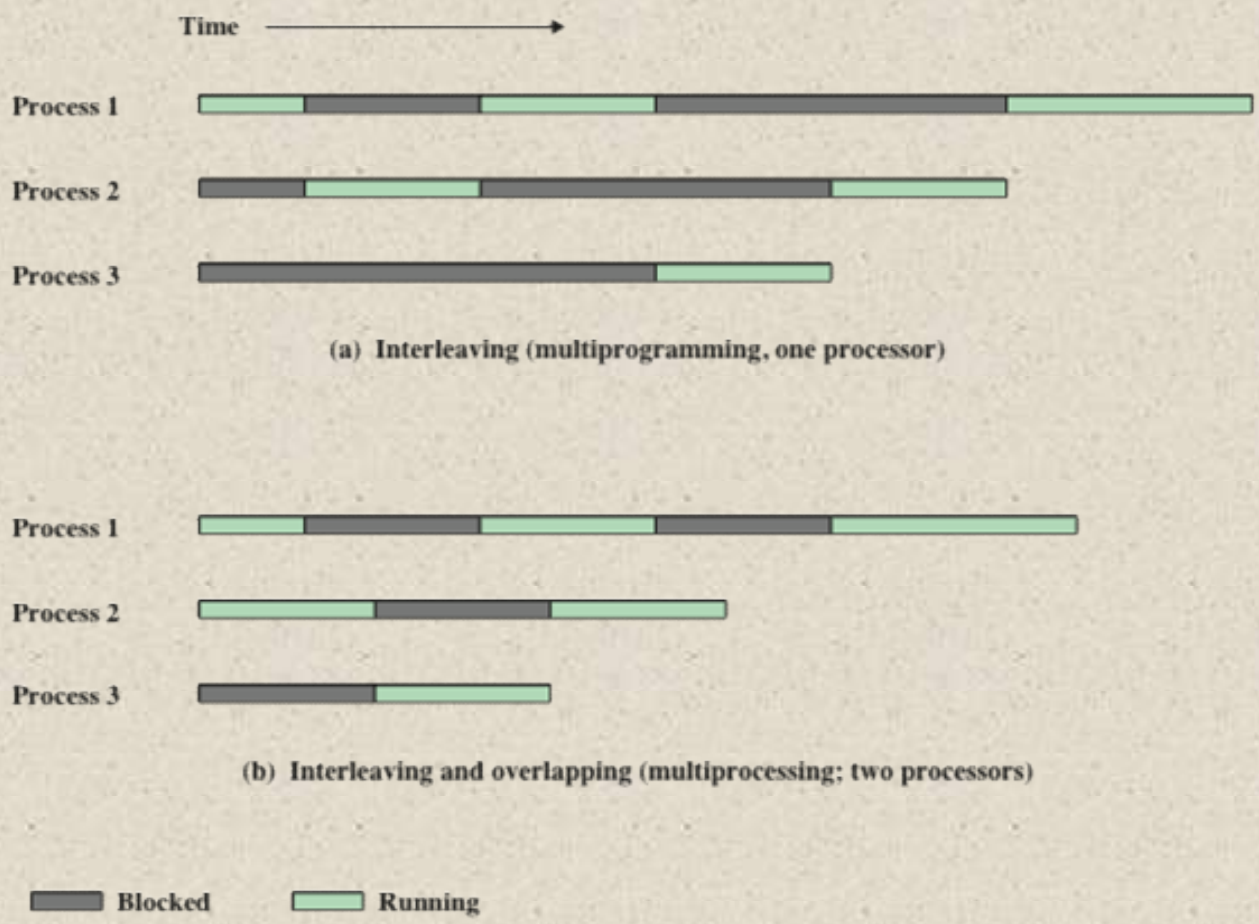


Figure 17.3 Multiprogramming and Multiprocessing

Organization

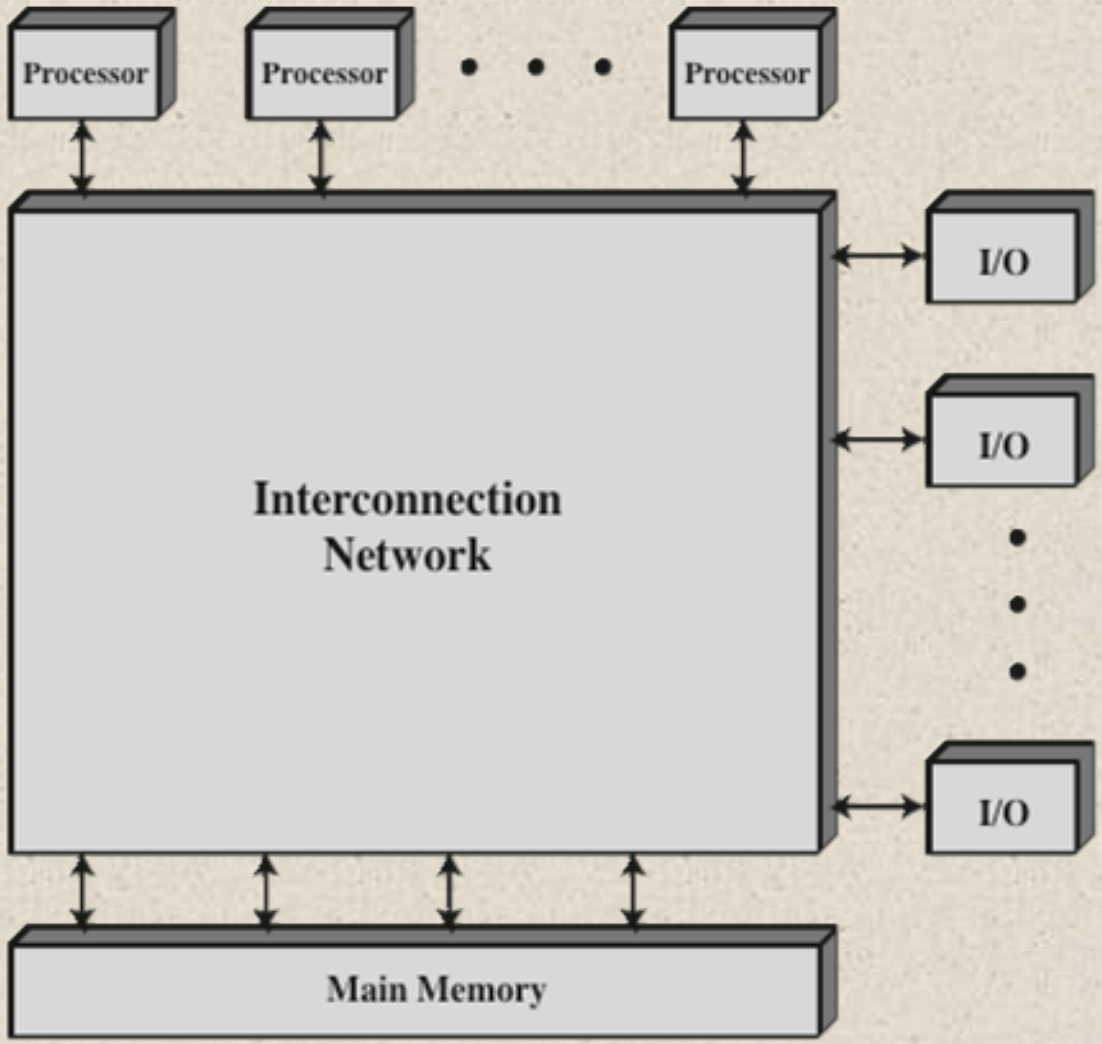


Figure 17.4 Generic Block Diagram of a Tightly Coupled Multiprocessor

Symmetric Multiprocessor Organization

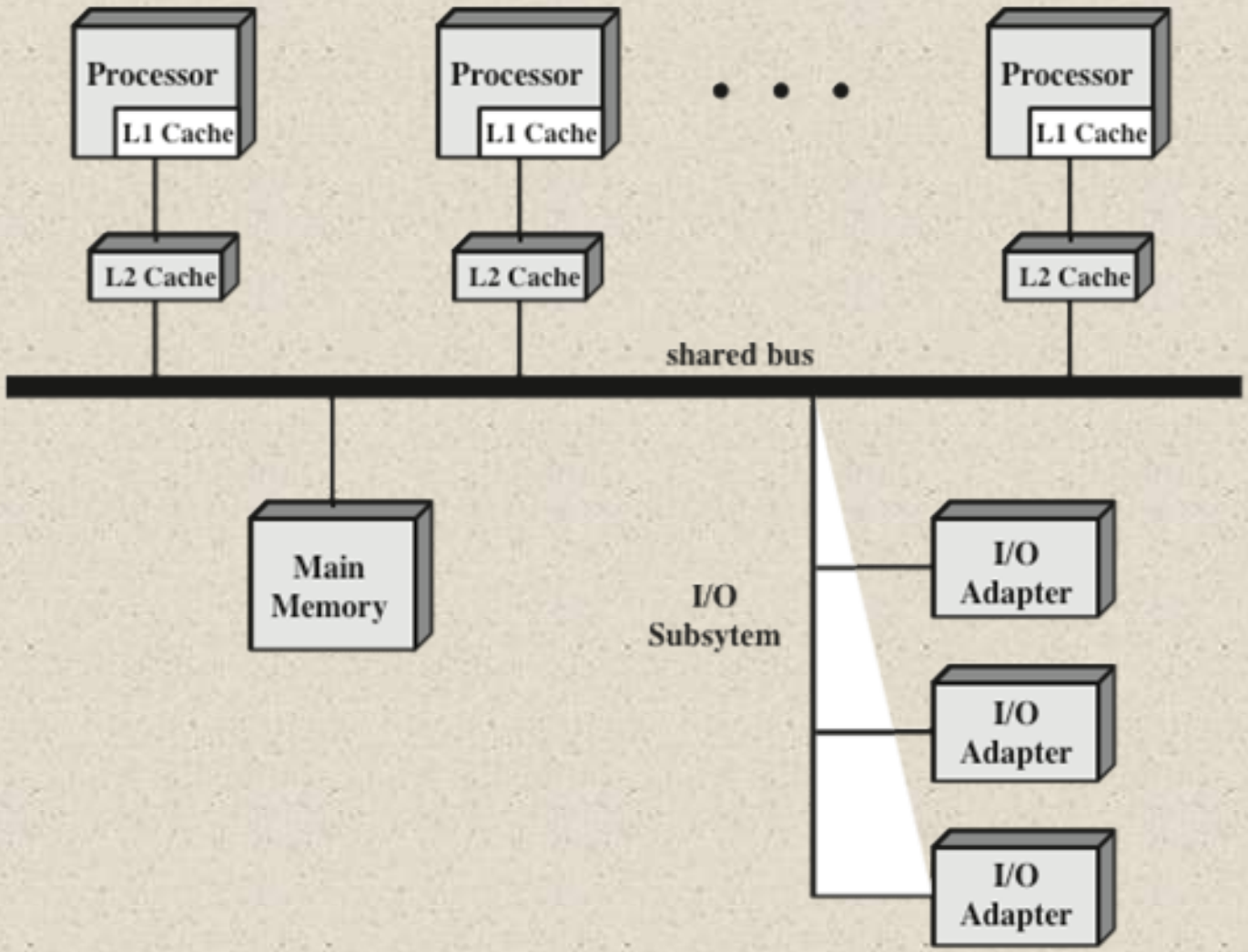
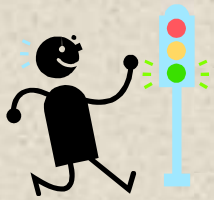


Figure 17.5 Symmetric Multiprocessor Organization

The bus organization has several attractive features:



- Simplicity
  - Simplest approach to multiprocessor organization
- Flexibility
  - Generally easy to expand the system by attaching more processors to the bus
- Reliability
  - The bus is essentially a passive medium and the failure of any attached device should not cause failure of the whole system

Disadvantages of the bus organization:

- Main drawback is performance
  - All memory references pass through the common bus
  - Performance is limited by bus cycle time
- Each processor should have cache memory
  - Reduces the number of bus accesses
- Leads to problems with *cache coherence*
  - If a word is altered in one cache it could conceivably invalidate a word in another cache
    - To prevent this the other processors must be alerted that an update has taken place
  - Typically addressed in hardware rather than the operating system

Multiprocessor Operating System Design Considerations

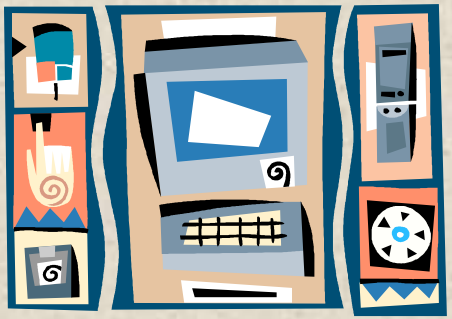
- **Simultaneous concurrent processes**
  - OS routines need to be reentrant to allow several processors to execute the same IS code simultaneously
  - OS tables and management structures must be managed properly to avoid deadlock or invalid operations
- **Scheduling**
  - Any processor may perform scheduling so conflicts must be avoided
  - Scheduler must assign ready processes to available processors
- **Synchronization**
  - With multiple active processes having potential access to shared address spaces or I/O resources, care must be taken to provide effective synchronization
  - Synchronization is a facility that enforces mutual exclusion and event ordering
- **Memory management**
  - In addition to dealing with all of the issues found on uniprocessor machines, the OS needs to exploit the available hardware parallelism to achieve the best performance
  - Paging mechanisms on different processors must be coordinated to enforce consistency when several processors share a page or segment and to decide on page replacement
- **Reliability and fault tolerance**
  - OS should provide graceful degradation in the face of processor failure
  - Scheduler and other portions of the operating system must recognize the loss of a processor and restructure accordingly

17.3 Cache Coherence

Software Solutions

- Attempt to avoid the need for additional hardware circuitry and logic by relying on the compiler and operating system to deal with the problem
- Attractive because the overhead of detecting potential problems is transferred from run time to compile time, and the design complexity is transferred from hardware to software
  - However, compile-time software approaches generally must make conservative decisions, leading to inefficient cache utilization

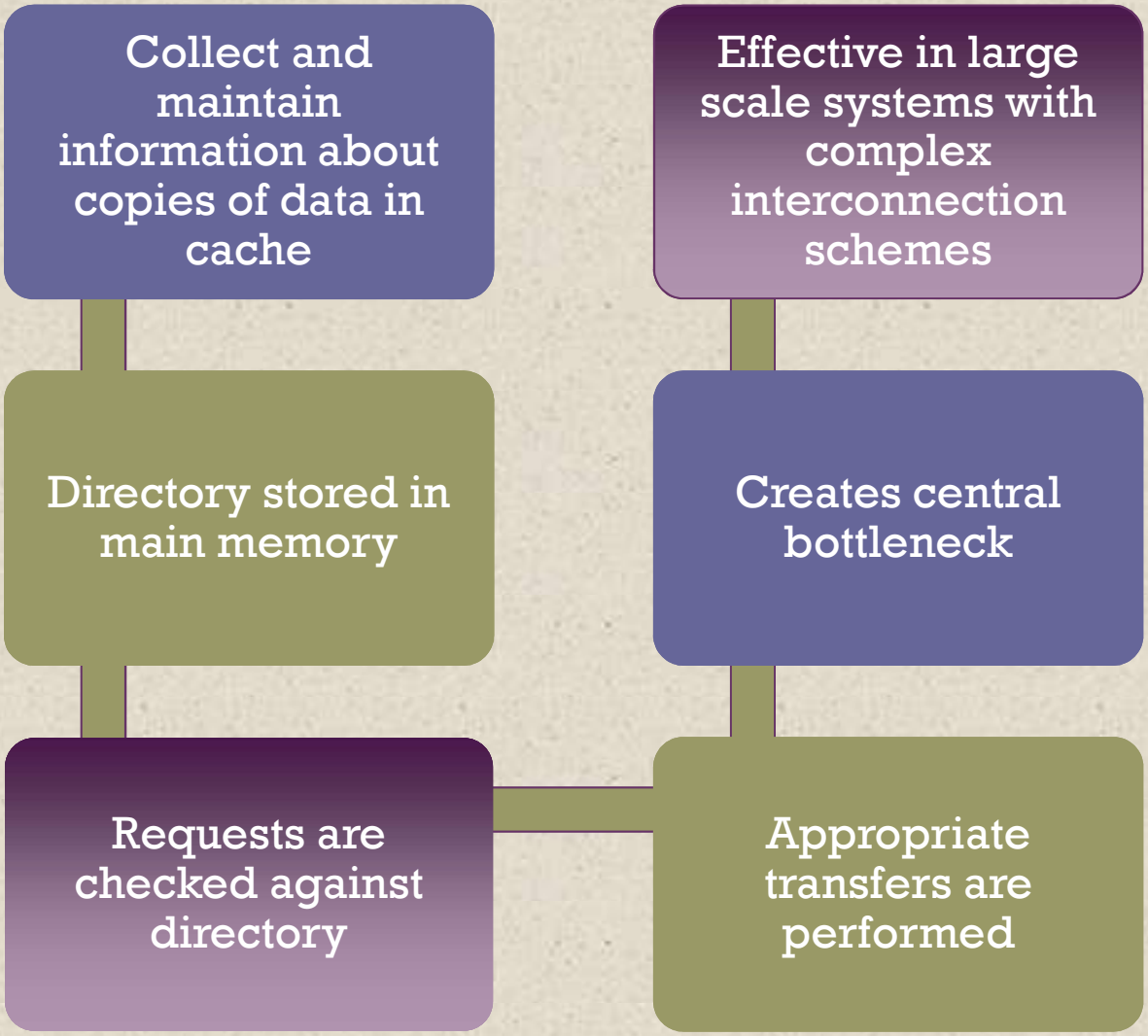
Cache Coherence



Hardware-Based Solutions

- Generally referred to as *cache coherence protocols*
- These solutions provide dynamic recognition at run time of potential inconsistency conditions
- Because the problem is only dealt with when it actually arises there is more effective use of caches, leading to improved performance over a software approach
- Approaches are transparent to the programmer and the compiler, reducing the software development burden
- Can be divided into two categories:
  - Directory protocols
  - Snoopy protocols

Directory Protocols



Snoopy Protocols

- Distribute the responsibility for maintaining cache coherence among all of the cache controllers in a multiprocessor
  - A cache must recognize when a line that it holds is shared with other caches
  - When updates are performed on a shared cache line, it must be announced to other caches by a broadcast mechanism
  - Each cache controller is able to "snoop" on the network to observe these broadcast notifications and react accordingly
- Suited to bus-based multiprocessor because the shared bus provides a simple means for broadcasting and snooping
  - Care must be taken that the increased bus traffic required for broadcasting and snooping does not cancel out the gains from the use of local caches
- Two basic approaches have been explored:
  - Write invalidate
  - Write update (or write broadcast)

Write Invalidate

- Multiple readers, but only one writer at a time
- When a write is required, all other caches of the line are invalidated
- Writing processor then has exclusive (cheap) access until line is required by another processor
- Most widely used in commercial multiprocessor systems such as the Pentium 4 and PowerPC
- State of every line is marked as modified, exclusive, shared or invalid
  - For this reason the write-invalidate protocol is called *MESI*

Write Update

- Can be multiple readers and writers
- When a processor wishes to update a shared line the word to be updated is distributed to all others and caches containing that line can update it
- Some systems use an adaptive mixture of both write-invalidate and write-update mechanisms

MESI Protocol

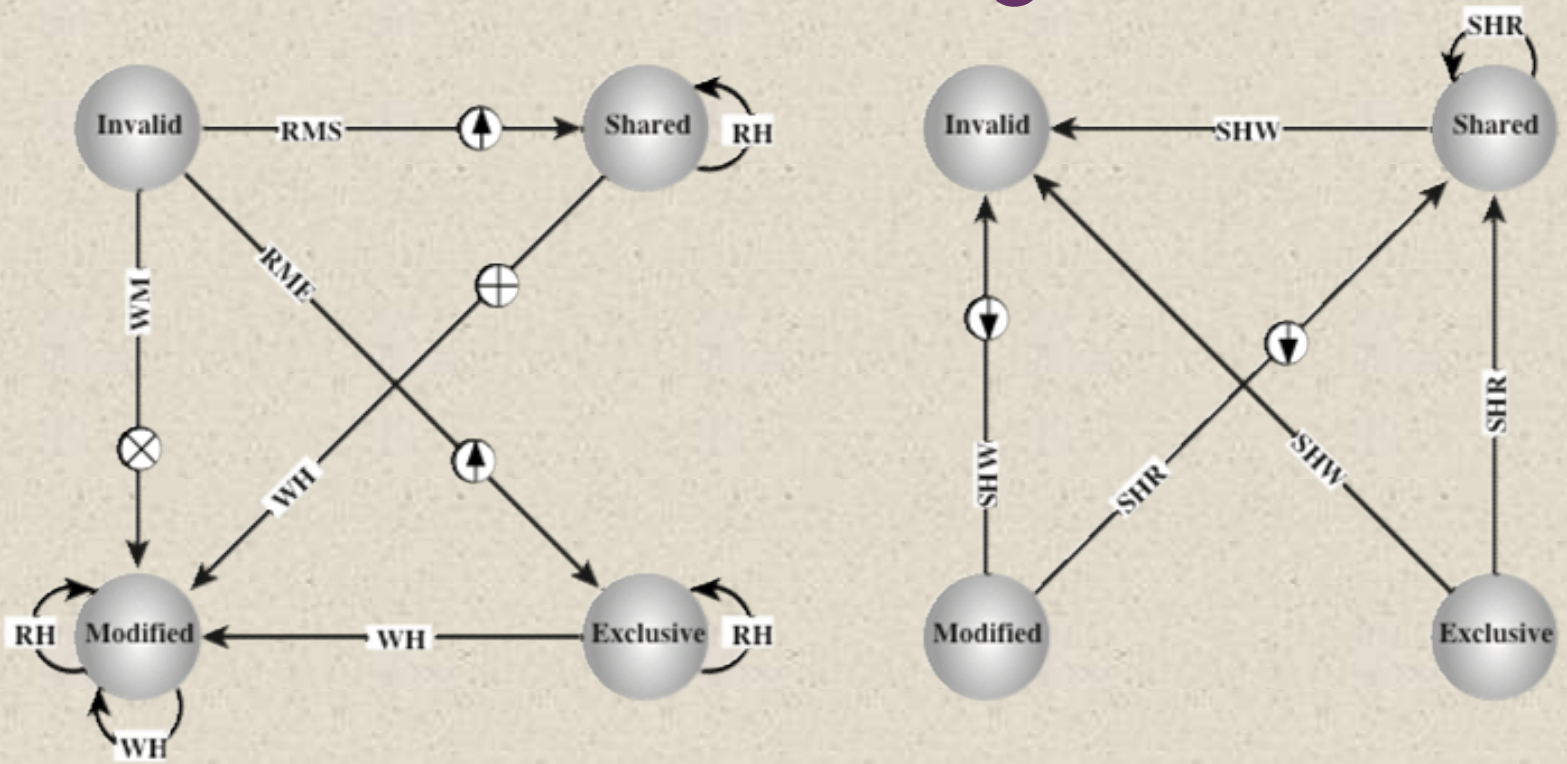
To provide cache consistency on an SMP the data cache supports a protocol known as MESI:

- Modified
  - The line in the cache has been modified and is available only in this cache
- Exclusive
  - The line in the cache is the same as that in main memory and is not present in any other cache
- Shared
  - The line in the cache is the same as that in main memory and may be present in another cache
- Invalid
  - The line in the cache does not contain valid data

Table 17.1
MESI Cache Line States

	M Modified	E Exclusive	S Shared	I Invalid
This cache line valid?	Yes	Yes	Yes	No
The memory copy is…	out of date	valid	valid	—
Copies exist in other caches?	No	No	Maybe	Maybe
A write to this line…	does not go to bus	does not go to bus	goes to bus and updates cache	goes directly to bus

MESI State Transition Diagram



RH	Read hit
RMS	Read miss, shared
RME	Read miss, exclusive
WH	Write hit
WM	Write miss
SHR	Snoop hit on read
SHW	Snoop hit on write or read-with-intent-to-modify

Dirty line copyback

Invalidate transaction

Read-with-intent-to-modify

Cache line fill

Figure 17.6 MESI State Transition Diagram

17.4 Multithreading and Chip Multiprocessors

- Processor performance can be measured by the rate at which it executes instructions
- MIPS rate = f * IPC
  - f = processor clock frequency, in MHz
  - IPC = average instructions per cycle
- Increase performance by increasing clock frequency and increasing instructions that complete during cycle
- Multithreading
  - Allows for a high degree of instruction-level parallelism without increasing circuit complexity or power consumption
  - Instruction stream is divided into several smaller streams, known as threads, that can be executed in parallel

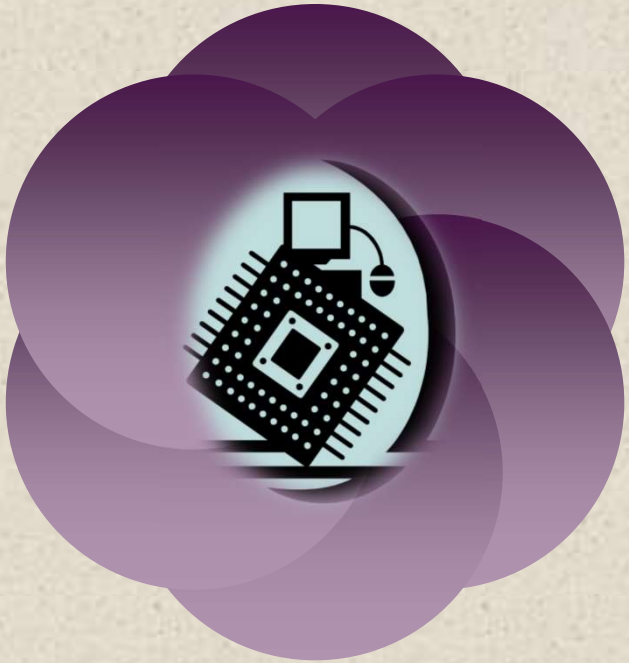
Definitions of Threads and Processes

Thread in multithreaded processors may or may not be the same as the concept of software threads in a multiprogrammed operating system

Thread switch

- The act of switching processor control between threads within the same process
- Typically less costly than process switch

Thread is concerned with scheduling and execution, whereas a process is concerned with both scheduling/execution and resource and resource ownership



Thread:

- Dispatchable unit of work within a process
- Includes processor context (which includes the program counter and stack pointer) and data area for stack
- Executes sequentially and is interruptible so that the processor can turn to another thread

Process:

- An instance of program running on computer
- Two key characteristics:
  - Resource ownership
  - Scheduling/execution

Process switch

- Operation that switches the processor from one process to another by saving all the process control data, registers, and other information for the first and replacing them with the process information for the second

17.4 MULTITHREADING AND CHIP MULTIPROCESSORS

Implicit and Explicit Multithreading

- All commercial processors and most experimental ones use explicit multithreading
  - Concurrently execute instructions from different explicit threads
  - Interleave instructions from different threads on shared pipelines or parallel execution on parallel pipelines
- Implicit multithreading is concurrent execution of multiple threads extracted from single sequential program
  - Implicit threads defined statically by compiler or dynamically by hardware

Approaches to Explicit Multithreading

- Interleaved
  - Fine-grained
  - Processor deals with two or more thread contexts at a time
  - Switching thread at each clock cycle
  - If thread is blocked it is skipped
- Simultaneous (SMT)
  - Instructions are simultaneously issued from multiple threads to execution units of superscalar processor
- Blocked
  - Coarse-grained
  - Thread executed until event causes delay
  - Effective on in-order processor
  - Avoids pipeline stall
- Chip multiprocessing
  - Processor is replicated on a single chip
  - Each processor handles separate threads
  - Advantage is that the available logic area on a chip is used effectively

Approaches to Executing Multiple Threads

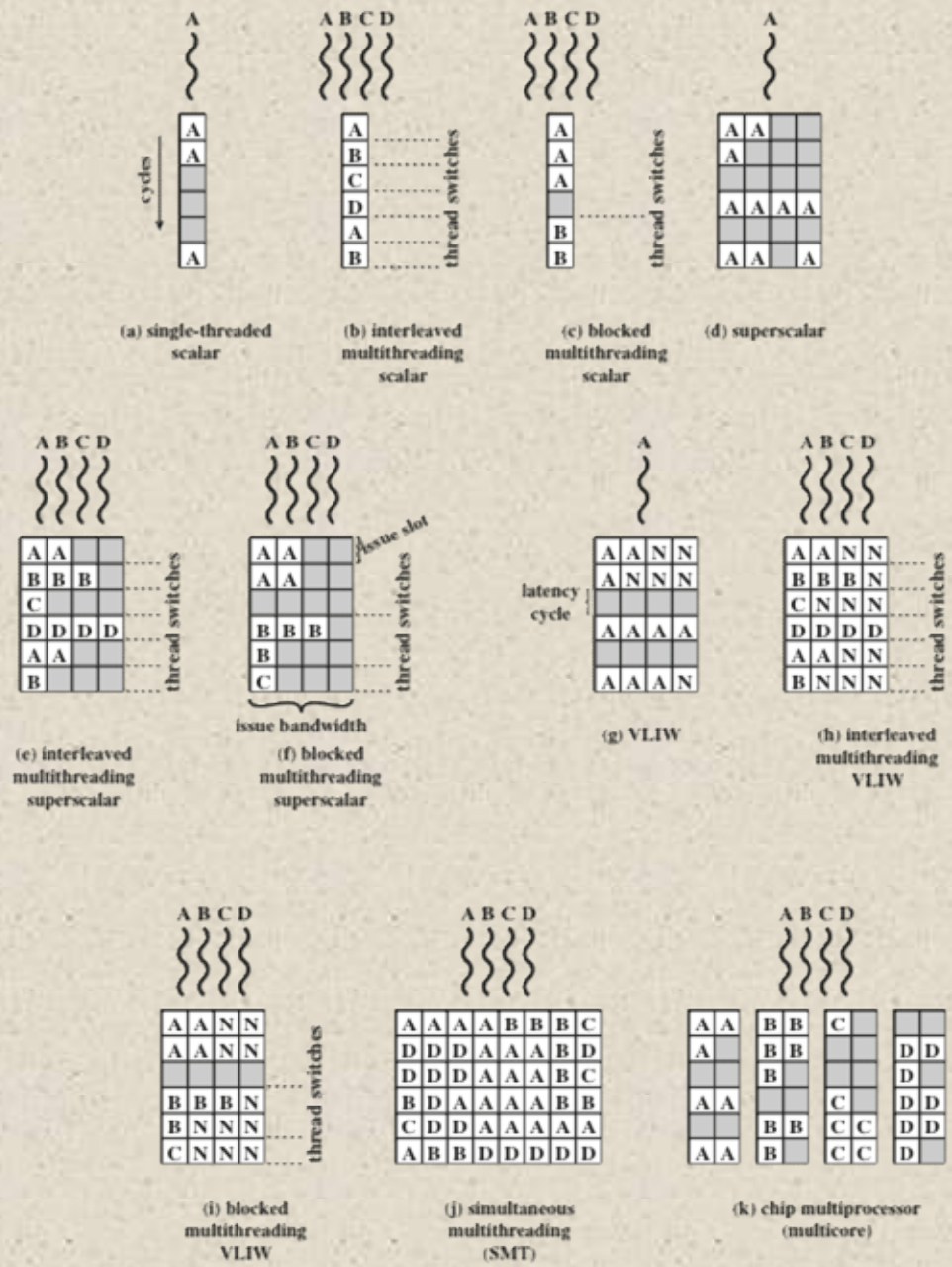


Figure 17.7 Approaches to Executing Multiple Threads

Example Systems

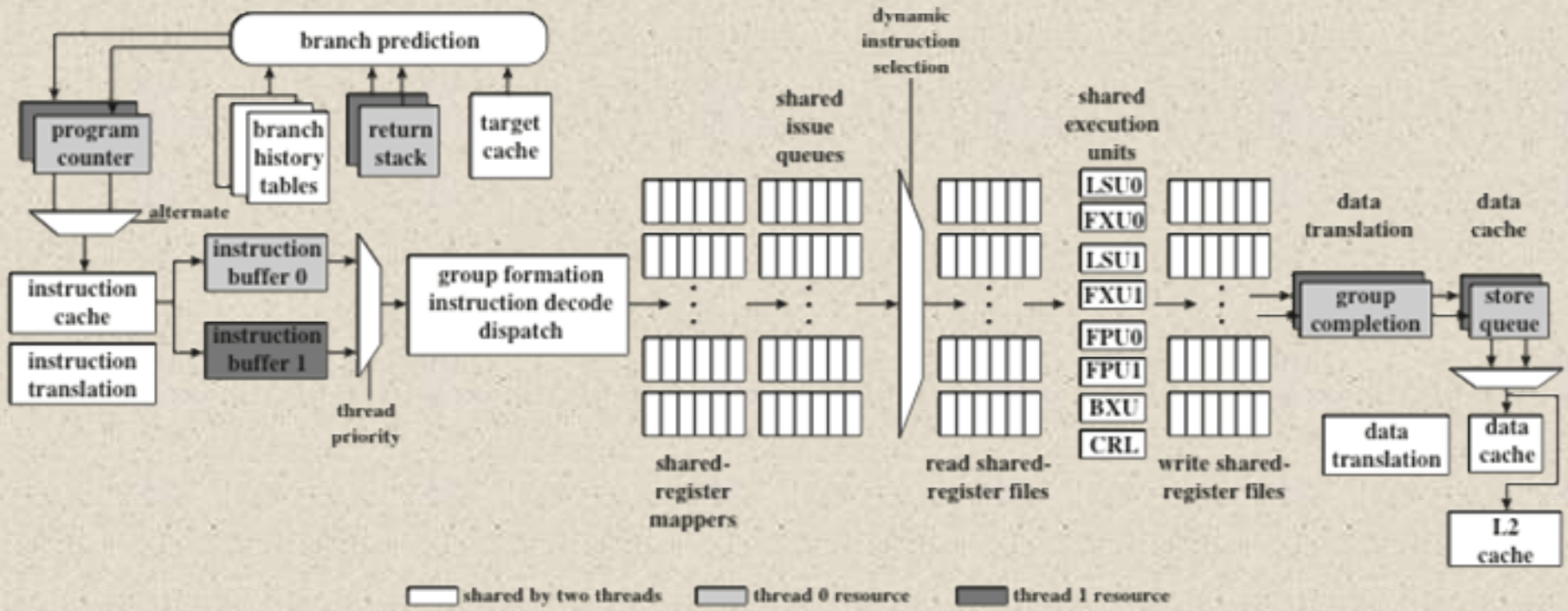
Pentium 4

- More recent models of the Pentium 4 use a multithreading technique that Intel refers to as *hyperthreading*
- Approach is to use SMT with support for two threads
- Thus the single multithreaded processor is logically two processors

IBM Power5

- Chip used in high-end PowerPC products
- Combines chip multiprocessing with SMT
  - Has two separate processors, each of which is a multithreaded processor capable of supporting two threads concurrently using SMT
  - Designers found that having two two-way SMT processors on a single chip provided superior performance to a single four-way SMT processor

Power5 Instruction Data Flow



BXU = branch execution unit and
CRL = condition register logical execution unit
FPU = floating-point execution unit
FXU = fixed-point execution unit
LSU = load/store unit

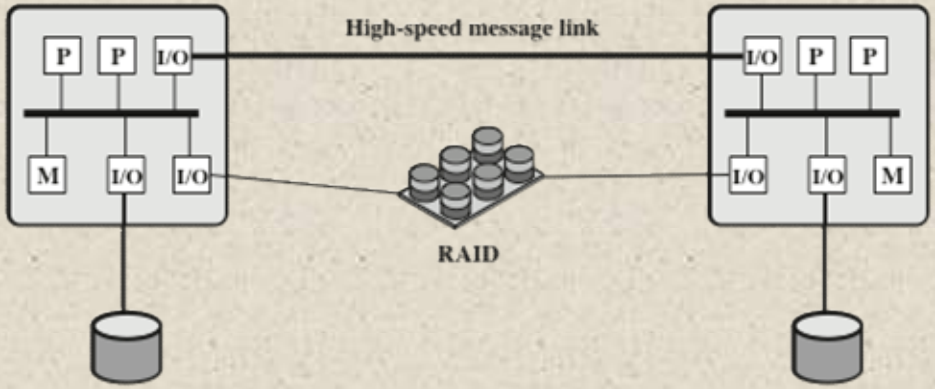
Figure 17.8 Power5 Instruction Data Flow

Clusters

- Alternative to SMP as an approach to providing high performance and high availability
- Particularly attractive for server applications
- Defined as:
  - A group of interconnected whole computers working together as a unified computing resource that can create the illusion of being one machine
  - (The term *whole computer* means a system that can run on its own, apart from the cluster)
- Each computer in a cluster is called a node
- Benefits:
  - Absolute scalability
  - Incremental scalability
  - High availability
  - Superior price/performance

Cluster Configurations

(a) Standby server with no shared disk



(b) Shared disk

Figure 17.9 Cluster Configurations

Table 17.2
Clustering Methods: Benefits and Limitations

Clustering Method	Description	Benefits	Limitations
Passive Standby	A secondary server takes over in case of primary server failure.	Easy to implement.	High cost because the secondary server is unavailable for other processing tasks.
Active Secondary:	The secondary server is also used for processing tasks.	Reduced cost because secondary servers can be used for processing.	Increased complexity.
Separate Servers	Separate servers have their own disks. Data is continuously copied from primary to secondary server.	High availability.	High network and server overhead due to copying operations.
Servers Connected to Disks	Servers are cabled to the same disks, but each server owns its disks. If one server fails, its disks are taken over by the other server.	Reduced network and server overhead due to elimination of copying operations.	Usually requires disk mirroring or RAID technology to compensate for risk of disk failure.
Servers Share Disks	Multiple servers simultaneously share access to disks.	Low network and server overhead. Reduced risk of downtime caused by disk failure.	Requires lock manager software. Usually used with disk mirroring or RAID technology.

Operating System Design Issues

- How failures are managed depends on the clustering method used
- Two approaches:
  - Highly available clusters
  - Fault tolerant clusters
- Failover
  - The function of switching applications and data resources over from a failed system to an alternative system in the cluster
- Failback
  - Restoration of applications and data resources to the original system once it has been fixed
- Load balancing
  - Incremental scalability
  - Automatically include new computers in scheduling
  - Middleware needs to recognize that processes may switch between machines

Parallelizing Computation

Effective use of a cluster requires executing software from a single application in parallel

Three approaches are:

Parallelizing complier

- Determines at compile time which parts of an application can be executed in parallel
- These are then split off to be assigned to different computers in the cluster

Parallelized application

- Application written from the outset to run on a cluster and uses message passing to move data between cluster nodes

Parametric computing

- Can be used if the essence of the application is an algorithm or program that must be executed a large number of times, each time with a different set of starting conditions or parameters

Cluster Computer Architecture

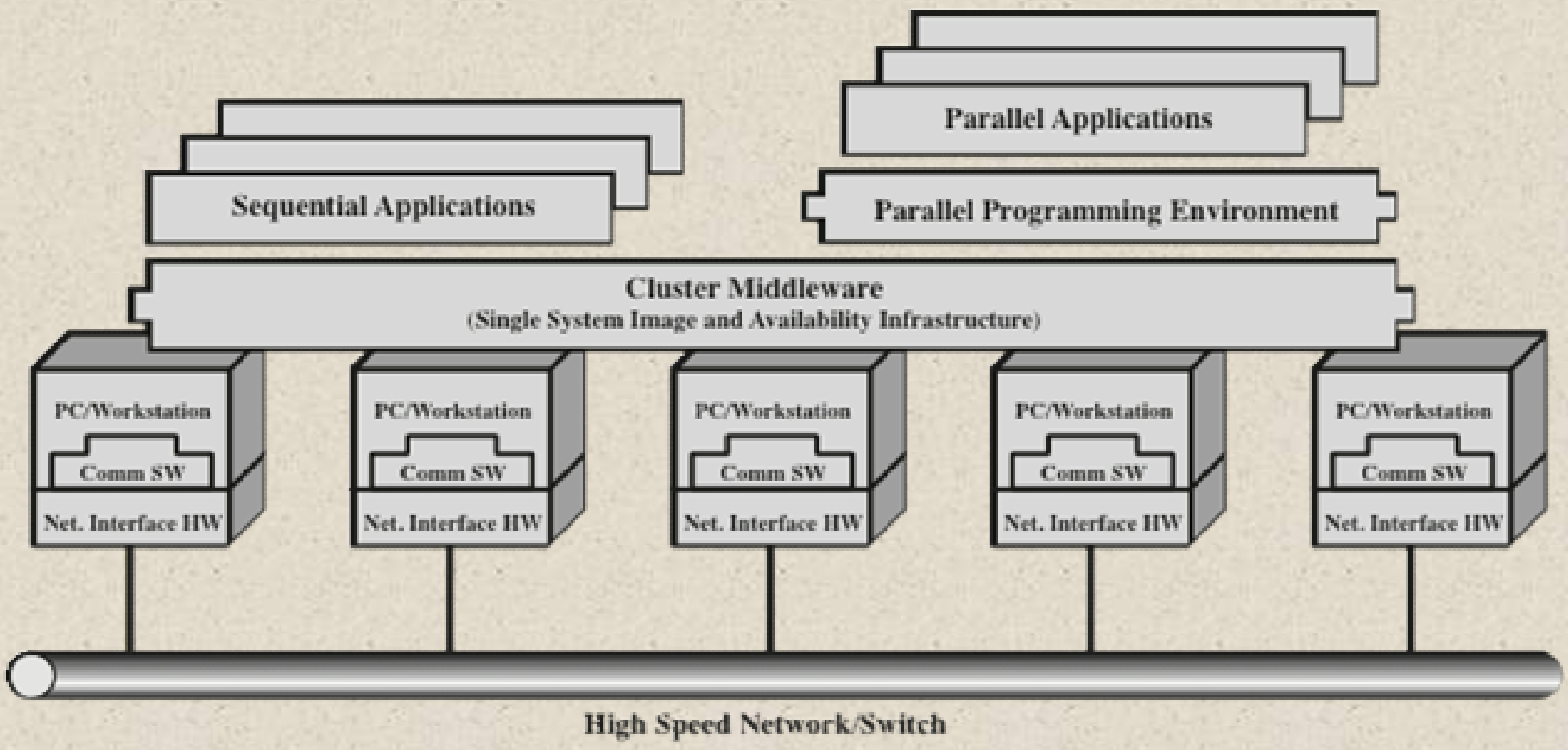


Figure 17.10 Cluster Computer Architecture [BUYY99a]

Example 100-Gbps Ethernet Configuration for Massive Blade Server Site

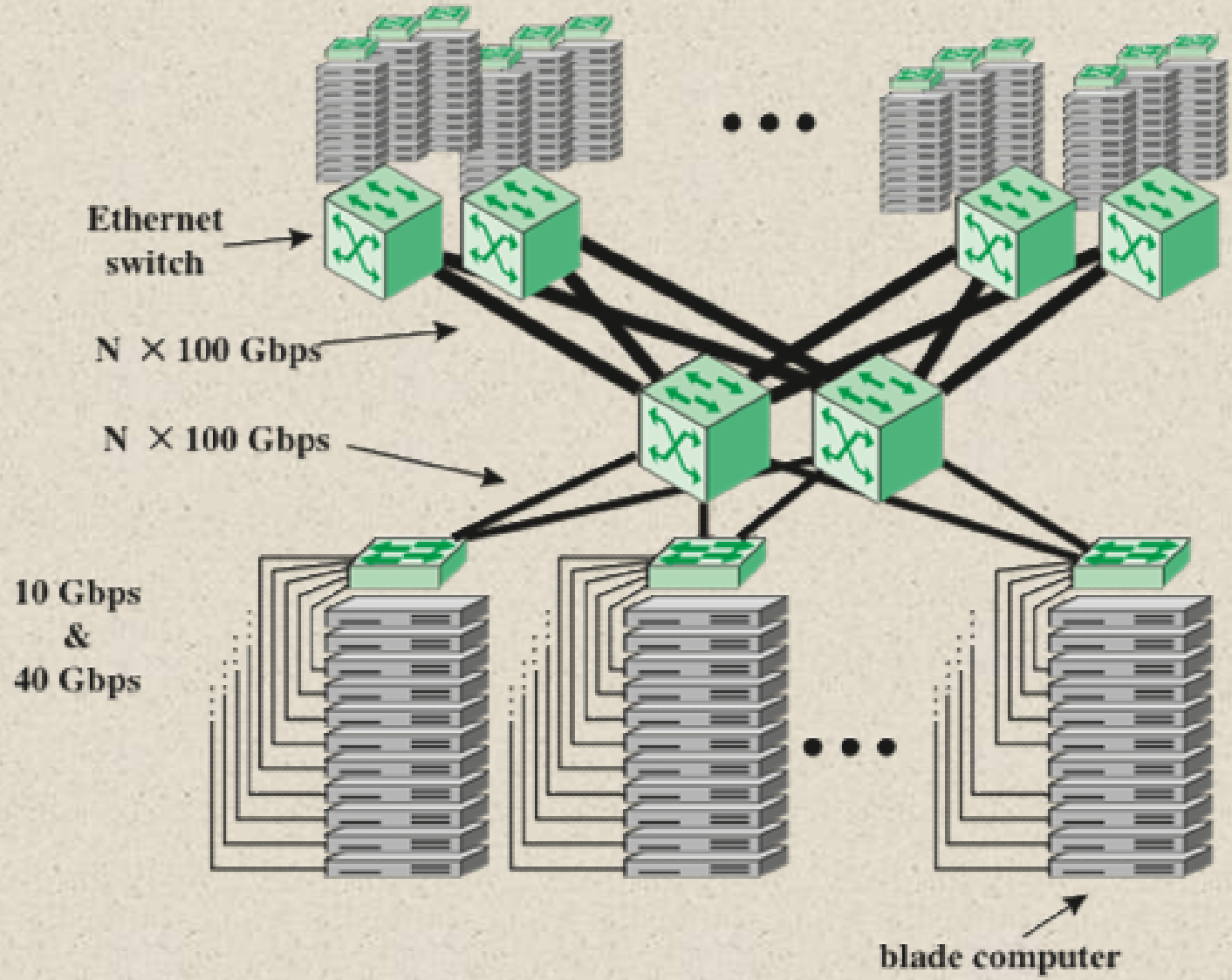


Figure 17.11 Example 100-Gbps Ethernet Configuration for Massive Blade Server Site

Clusters Compared to SMP

- Both provide a configuration with multiple processors to support high demand applications
- Both solutions are available commercially

SMP	Clustering
Easier to manage and configure	Far superior in terms of incremental and absolute scalability
Much closer to the original single processor model for which nearly all applications are written	Superior in terms of availability
Less physical space and lower power consumption	All components of the system can readily be made highly redundant
Well established and stable	

Nonuniform Memory Access (NUMA)

- Alternative to SMP and clustering
- Uniform memory access (UMA)
  - All processors have access to all parts of main memory using loads and stores
  - Access time to all regions of memory is the same
  - Access time to memory for different processors is the same
- Nonuniform memory access (NUMA)
  - All processors have access to all parts of main memory using loads and stores
  - Access time of processor differs depending on which region of main memory is being accessed
  - Different processors access different regions of memory at different speeds
- Cache-coherent NUMA (CC-NUMA)
  - A NUMA system in which cache coherence is maintained among the caches of the various processors

Motivation

SMP has practical limit to number of processors that can be used

- Bus traffic limits to between 16 and 64 processors

In clusters each node has its own private main memory

- Applications do not see a large global memory
- Coherency is maintained by software rather than hardware

NUMA retains SMP flavor while giving large scale multiprocessing

Objective with NUMA is to maintain a transparent system wide memory while permitting multiple multiprocessor nodes, each with its own bus or internal interconnect system

CC-NUMA Organization

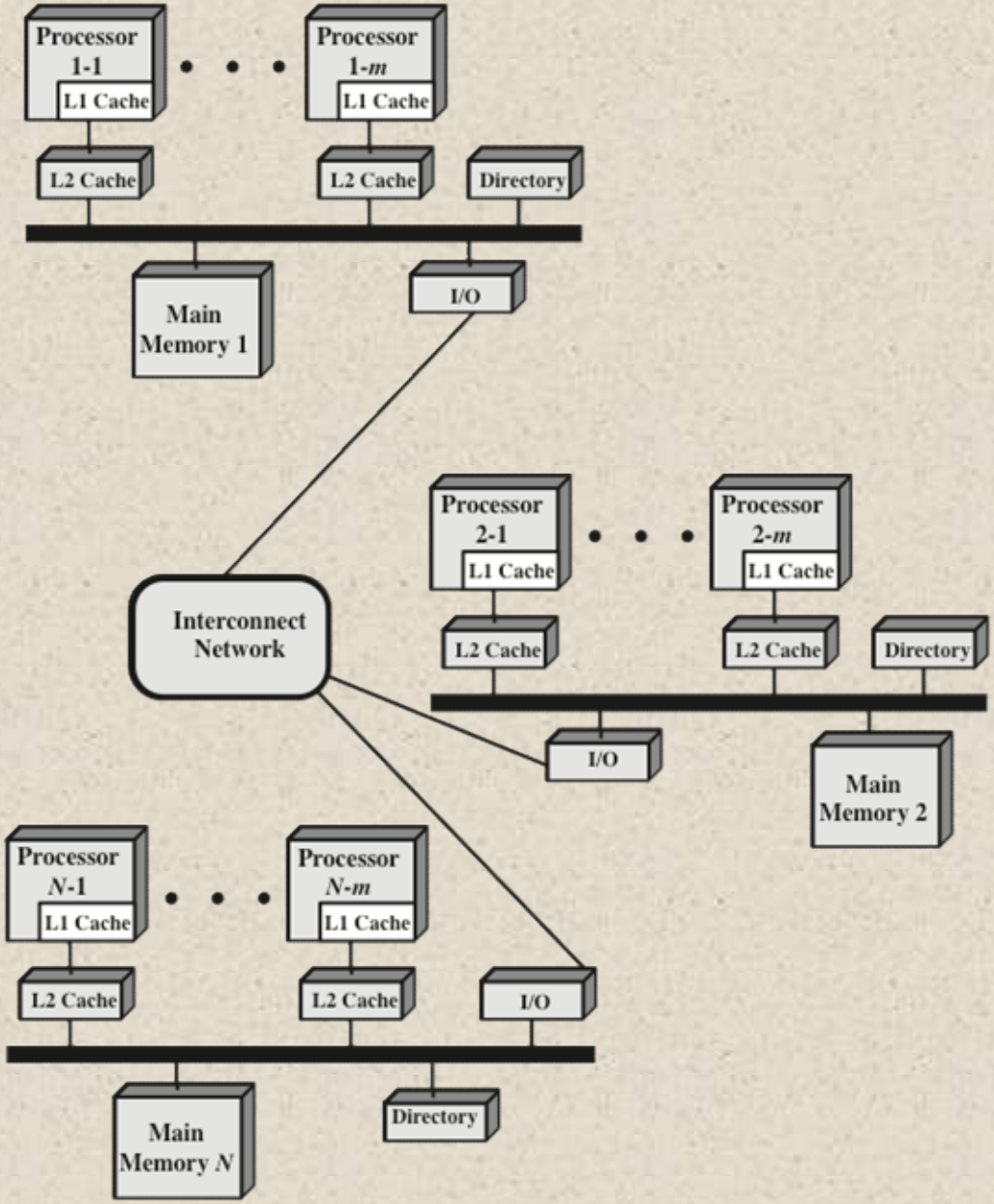


Figure 17.12 CC-NUMA Organization

NUMA Pros and Cons

- Main advantage of a CC-NUMA system is that it can deliver effective performance at higher levels of parallelism than SMP without requiring major software changes
- Bus traffic on any individual node is limited to a demand that the bus can handle
- If many of the memory accesses are to remote nodes, performance begins to break down
- Does not transparently look like an SMP
- Software changes will be required to move an operating system and applications from an SMP to a CC-NUMA system
- Concern with availability

Vector Computation

- There is a need for computers to solve mathematical problems of physical processes in disciplines such as aerodynamics, seismology, meteorology, and atomic, nuclear, and plasma physics
- Need for high precision and a program that repetitively performs floating point arithmetic calculations on large arrays of numbers
  - Most of these problems fall into the category known as *continuous-field simulation*
- Supercomputers were developed to handle these types of problems
  - However they have limited use and a limited market because of their price tag
  - There is a constant demand to increase performance
- Array processor
  - Designed to address the need for vector computation
  - Configured as peripheral devices by both mainframe and minicomputer users to run the vectorized portions of programs

Vector Addition Example

$$\begin{bmatrix} 1.5 \\ 7.1 \\ 6.9 \\ 100.5 \\ 0 \\ 59.7 \end{bmatrix} + \begin{bmatrix} 2.0 \\ 39.7 \\ 1000.003 \\ 11 \\ 21.1 \\ 19.7 \end{bmatrix} = \begin{bmatrix} 3.5 \\ 46.8 \\ 1006.903 \\ 111.5 \\ 21.1 \\ 79.4 \end{bmatrix}$$

$$A \quad + \quad B \quad = \quad C$$

Figure 17.13 Example of Vector Addition

Matrix Multiplication (C = A * B)

```
      DO 100 I = 1, N
      DO 100 J = 1, N
      C(I, J) = 0.0
      DO 100 K = 1, N
      C(I, J) = C(I, J) + A(I, K) + B(K, J)
100   CONTINUE
```

(a) Scalar processing

```
      DO 100 I = 1, N
      C(I, J) = 0.0 (J = 1, N)
      DO 100 K = 1, N
      C(I, J) = C(I, J) + A(I, K) + B(K, J) (J = 1, N)
100   CONTINUE
```

(b) Vector processing

```
      DO 50 J = 1, N – 1
      FORK 100
50    CONTINUE
      J = N
100   DO 200 I = 1, N
      C(I, J) = 0.0
      DO 200 K = 1, N
      C(I, J) = C(I, J) + A(I, K) + B(K, J)
200   CONTINUE
      JOIN N
```

(c) Parallel processing

Figure 17.14 Matrix Multiplication (C = A × B)

Approaches to Vector Computation

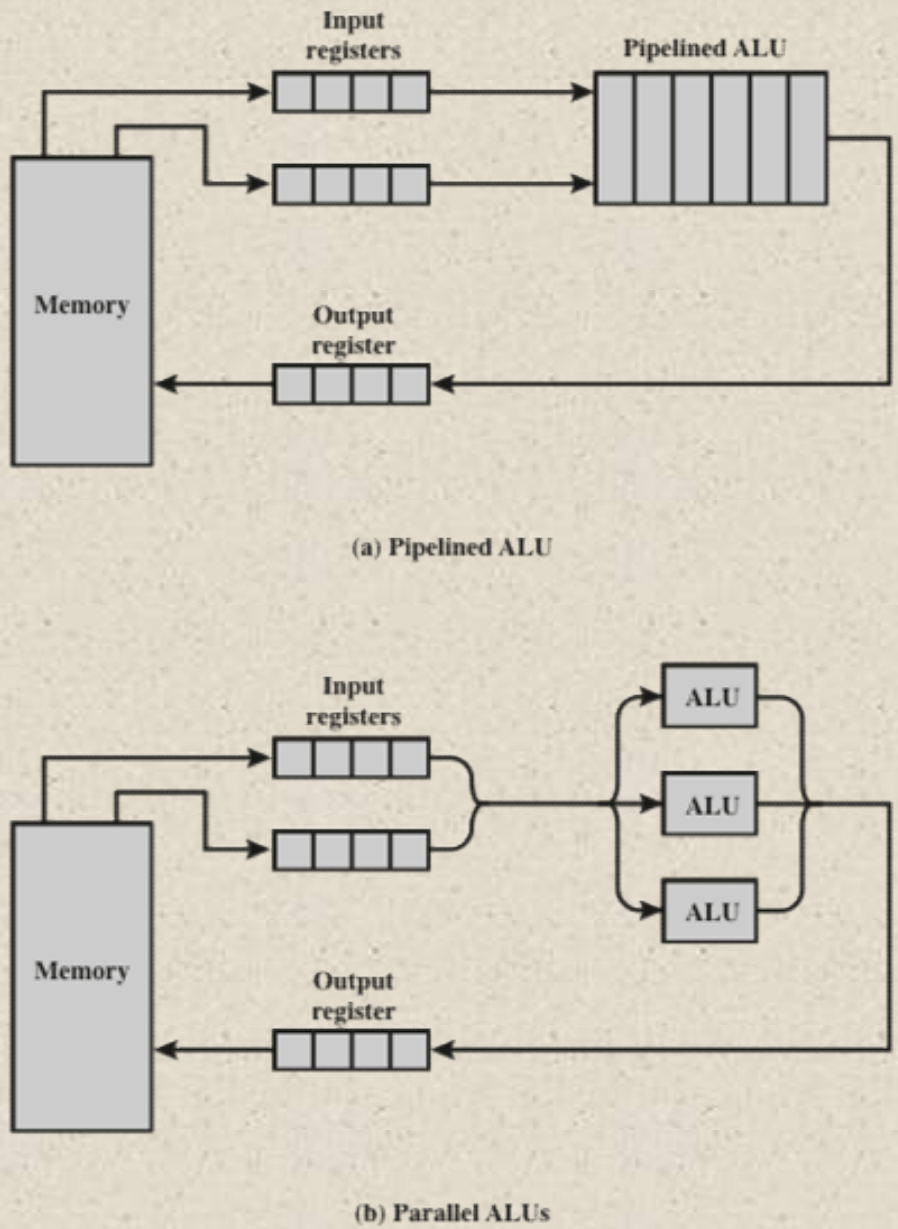


Figure 17.15 Approaches to Vector Computation

Pipelined Processing of Floating-Point Operations

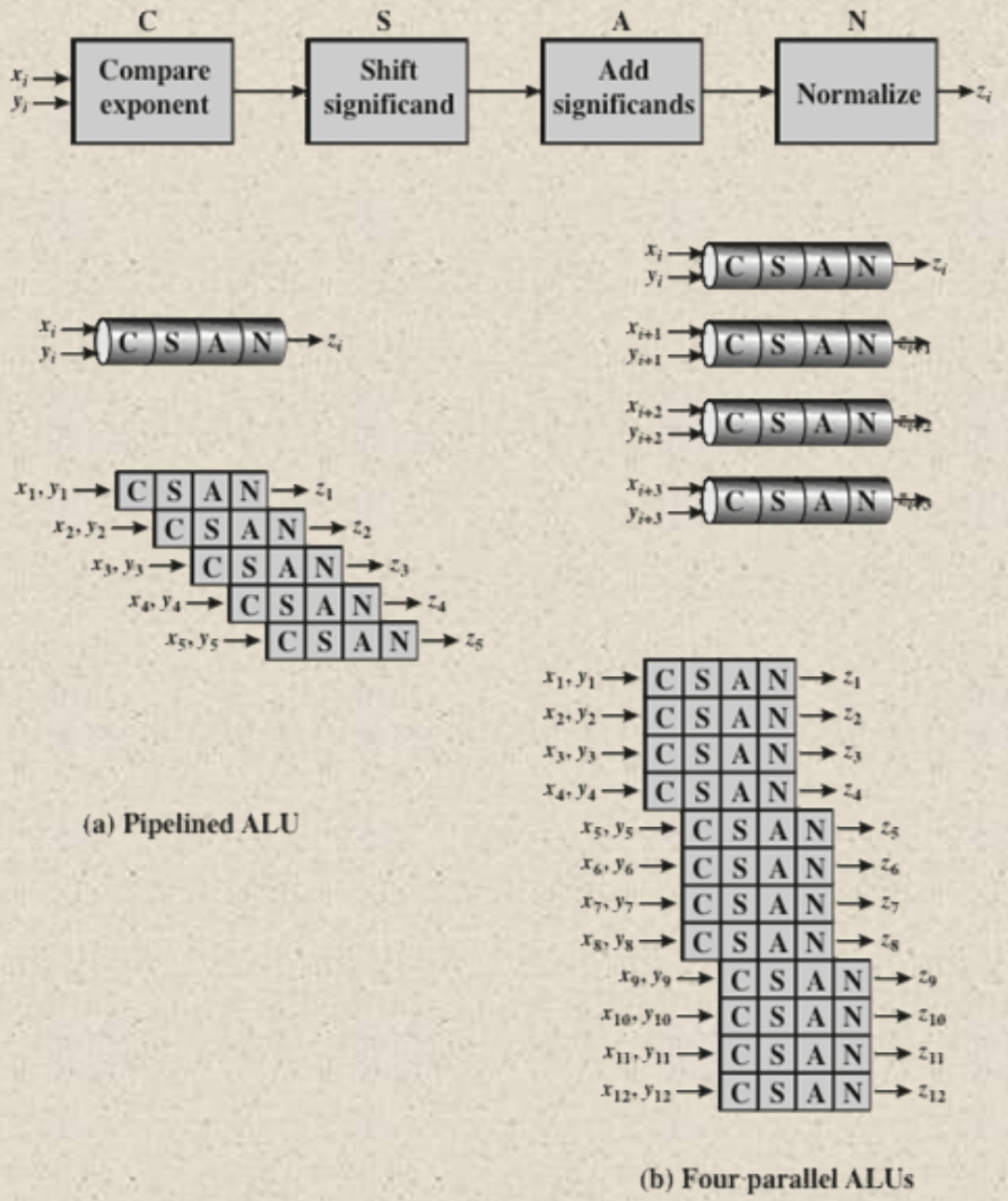


Figure 17.16 Pipelined Processing of Floating-Point Operations

A Taxonomy of Computer Organizations

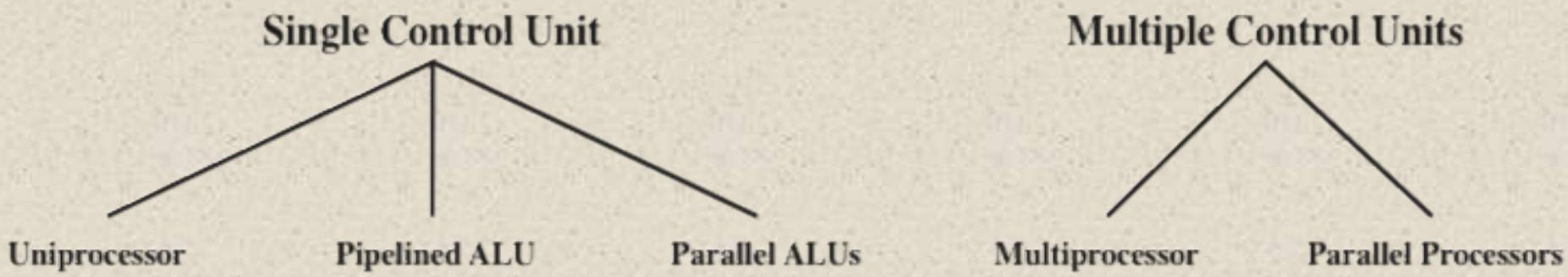


Figure 17.17 A Taxonomy of Computer Organizations

IBM 3090 with Vector Facility

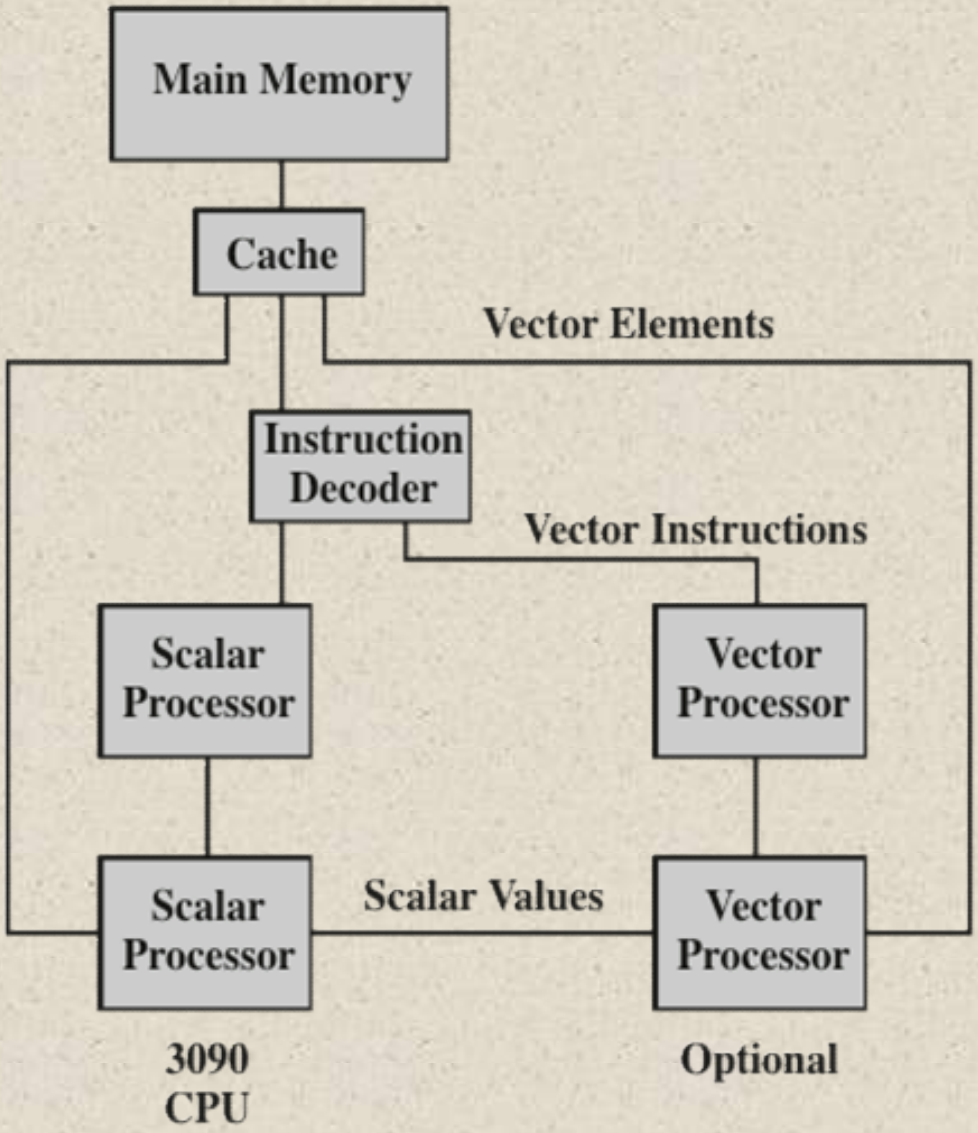


Figure 17.18 IBM 3090 with Vector Facility

Alternative Programs for Vector Calculation

FORTRAN ROUTINE:

```
      DO 100 J = 1, 50
      CR(J) = AR(J) * BR(J) – AI(J) * BI(J)
100   CI(J) = AR(J) * BI(J) + AI(J) * BR(J)
```

Operation	Cycles
AR(J) * BR(J) → T1(J)	3
AI(J) * BI(J) → T2(J)	3
T1(J) – T2(J) → CR(J)	3
AR(J) * BI(J) → T3(J)	3
AI(J) * BR(J) → T4(J)	3
T3(J) + T4(J) → CI(J)	3
TOTAL	18

(a) Storage to storage

Operation	Cycles
AR(J) → V1(J)	1
BR(J) → V2(J)	1
V1(J) * V2(J) → V3(J)	1
AI(J) → V4(J)	1
BI(J) → V5(J)	1
V4(J) * V5(J) → V6(J)	1
V3(J) – V6(J) → V7(J)	1
V7(J) → CR(J)	1
V1(J) * V5(J) → V8(J)	1
V4(J) * V2(J) → V9(J)	1
V8(J) + V9(J) → V0(J)	1
V0(J) → CI(J)	1
TOTAL	12

(b) Register to register

Operation	Cycles
AR(J) → V1(J)	1
V1(J) * BR(J) → V2(J)	1
AI(J) → V3(J)	1
V3(J) * BI(J) → V4(J)	1
V2(J) – V4(J) → V5(J)	1
V5(J) → CR(J)	1
V1(J) * BI(J) → V6(J)	1
V4(J) * BR(J) → V7(J)	1
V6(J) + V7(J) → V8(J)	1
V8(J) → CI(J)	1
TOTAL	10

(c) Storage to register

Operation	Cycles
AR(J) → V1(J)	1
V1(J) * BR(J) → V2(J)	1
AI(J) → V3(J)	1
V2(J) – V3(J) * BI(J) → V2(J)	1
V2(J) → CR(J)	1
V1(J) * BI(J) → V4(J)	1
V4(J) + V3(J) * BR(J) → V5(J)	1
V5(J) → CI(J)	1
TOTAL	8

(d) Compound instruction

Vi = vector registers
AR, BR, AI, BI = operands in memory
Ti = temporary locations in memory

Figure 17.19 Alternative Programs for Vector Calculation

Registers for the IBM 3090 Vector Facility

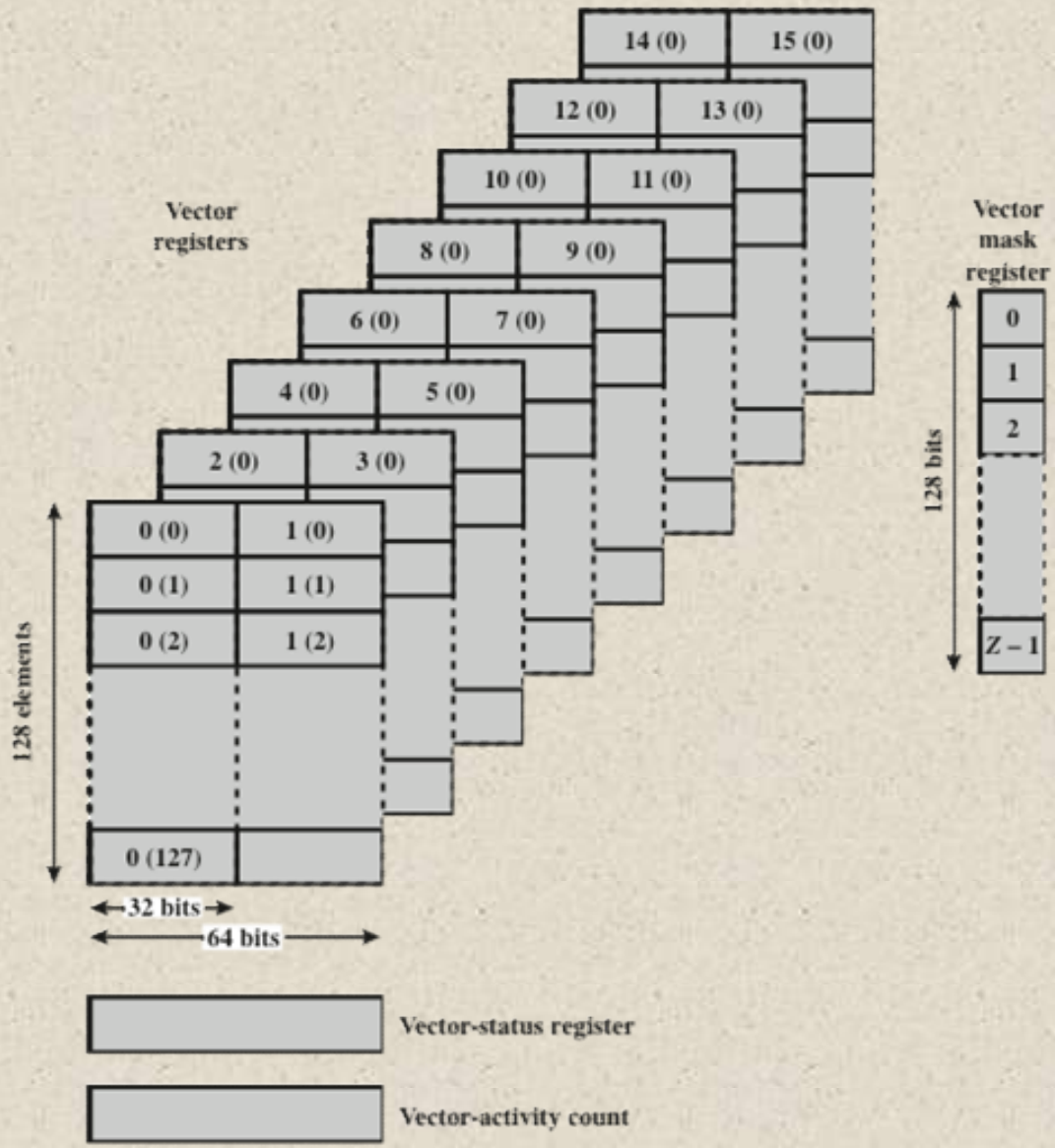


Figure 17.20 Registers for the IBM 3090 Vector Facility

Table 17.3
IBM 3090 Vector Facility: Arithmetic and Logical Instructions

Operation	Data Types: Floating-Point Long	Data Types: Floating-Point Short	Data Types: Binary or Logical	Operand Locations			
Add	FL	FS	BI	V + V → V	V + S → V	Q + V → V	Q + S → V
Subtract	FL	FS	BI	V – V → V	V – S → V	Q – V → V	Q – S → V
Multiply	FL	FS	BI	V × V → V	V × V → V	Q × V → V	Q × S → V
Divide	FL	FS	—	V/V → V	V/S → V	Q/V → V	Q/S → V
Compare	FL	FS	BI	V · V → V	V · S → V	Q · V → V	Q · S → V
Multiply and Add	FL	FS	—		V + V × S → V	V + Q × V → V	V + Q × S → V
Multiply and Subtract	FL	FS	—		V – V × S → V	V – Q × V → V	V – Q × S → V
Multiply and Accumulate	FL	FS	—	P + · V → V	P + · S → V		
Complement	FL	FS	BI	–V → V			
Positive Absolute	FL	FS	BI	|V| → V			
Negative Absolute	FL	FS	BI	–|V| → V			
Maximum	FL	FS	—			Q · V → Q	
Maximum Absolute	FL	FS	—			Q · V → Q	
Minimum	FL	FS	—			Q · V → Q	
Shift Left Logical	—	—	LO	· V → V			
Shift Right Logical	—	—	LO	· V → V			
And	—	—	LO	V & V → V	V & S → V	Q & V → V	Q & S → V
OR	—	—	LO	V | V → V	V | S → V	Q | V → V	Q | S → V
Exclusive-OR	—	—	LO	V ⊕ V → V	V ⊕ S → V	Q ⊕ V → V	Q ⊕ S → V

Explanation:

Data Types
- FL Long floating point
- FS Short floating point
- BI Binary integer
- LO Logical

Operand Locations
- V Vector register
- S Storage
- Q Scalar (general or floating-point register)
- P Partial sums in vector register
- · Special operation

Summary

Chapter 17

Parallel Processing

- Multiple processor organizations
  - Types of parallel processor systems
  - Parallel organizations
- Symmetric multiprocessors
  - Organization
  - Multiprocessor operating system design considerations
- Cache coherence and the MESI protocol
  - Software solutions
  - Hardware solutions
  - The MESI protocol
- Multithreading and chip multiprocessors
  - Implicit and explicit multithreading
  - Approaches to explicit multithreading
  - Example systems
- Clusters
  - Cluster configurations
  - Operating system design issues
  - Cluster computer architecture
  - Blade servers
  - Clusters compared to SMP
- Nonuniform memory access
  - Motivation
  - Organization
  - NUMA Pros and cons
- Vector computation
  - Approaches to vector computation
  - IBM 3090 vector facility