

# Chapter 17

## **Parallel Processing**

William Stallings, Computer Organization and Architecture, 9<sup>th</sup> Edition

# + Objectives

You are profiting from multiple CPU computers, You should know about them.

After studying this chapter, you should be able to:

- Summarize the types of parallel processor organizations.
- Present an overview of design features of symmetric multiprocessors. Understand the issue of cache coherence in a multiple processor system.
- Explain the key features of the MESI protocol.
- Explain the difference between implicit and explicit multithreading. Summarize key design issues for clusters.

# + Contents

- 17.1 Multiple Processor Organizations
- 17.2 Symmetric Multiprocessors
- 17.3 Cache Coherence and the MESI Protocol
- 17.4 Multithreading and Chip Multiprocessors



# 17.1- Multiple Processor Organization



- 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

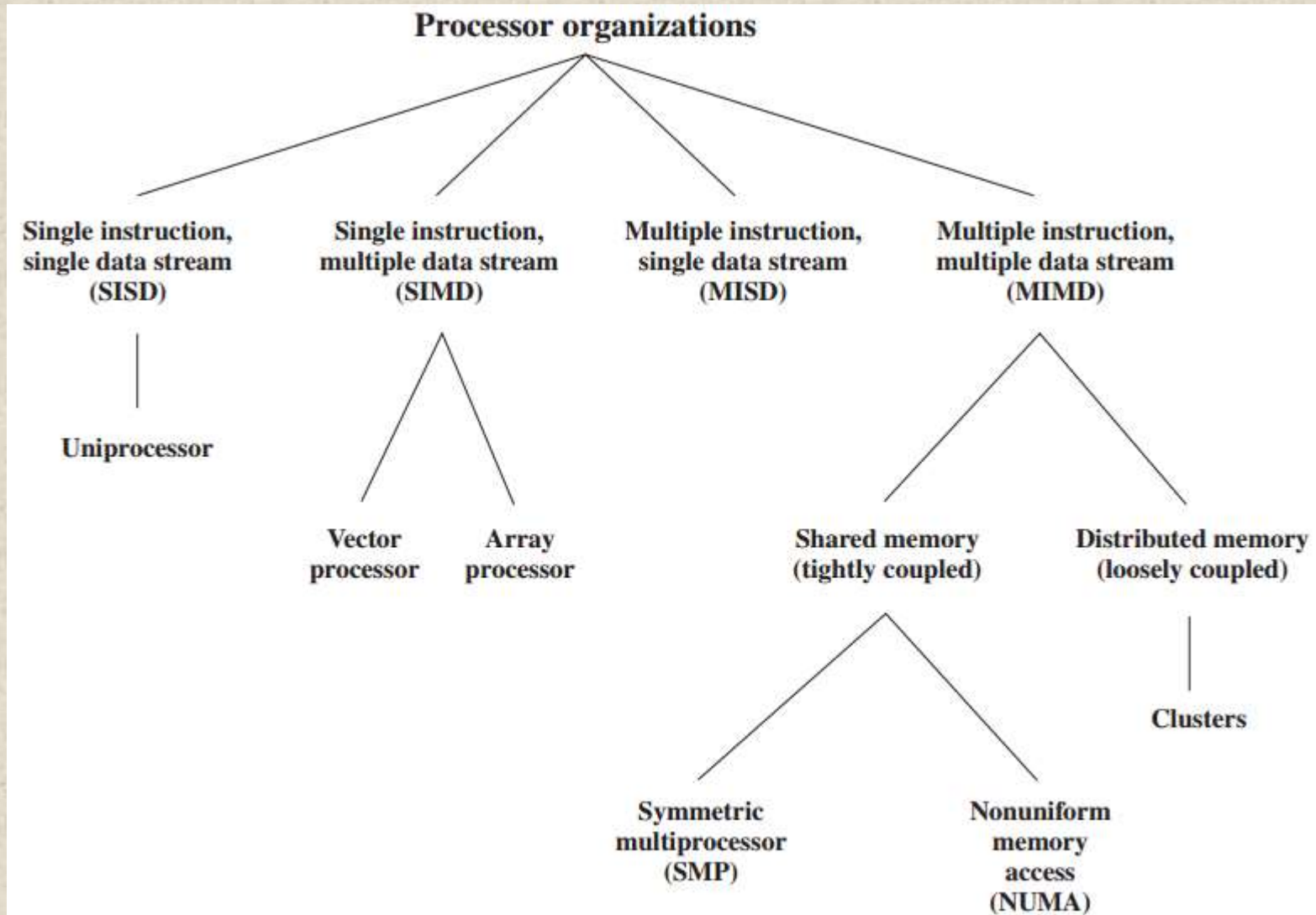
- Single instruction, multiple data (**SIMD**) stream
  - **A single machine** instruction controls the simultaneous execution of a number of processing elements on a lockstep basis
  - Vector and array processors fall into this category

- 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



# Parallel Organizations

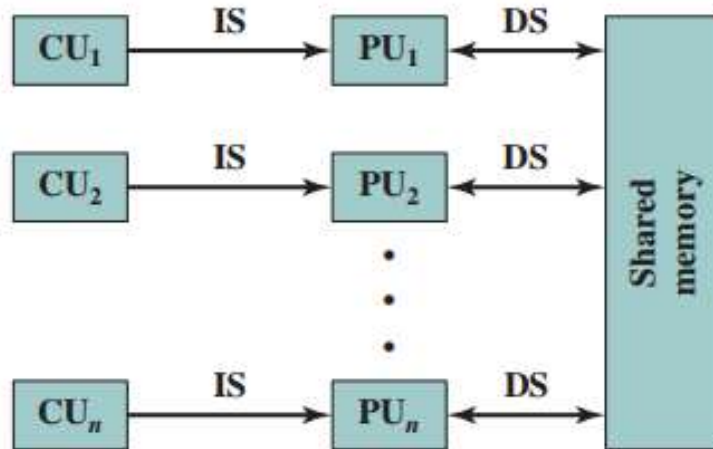


**Figure 17.1** A Taxonomy of Parallel Processor Architectures

# Parallel Organizations

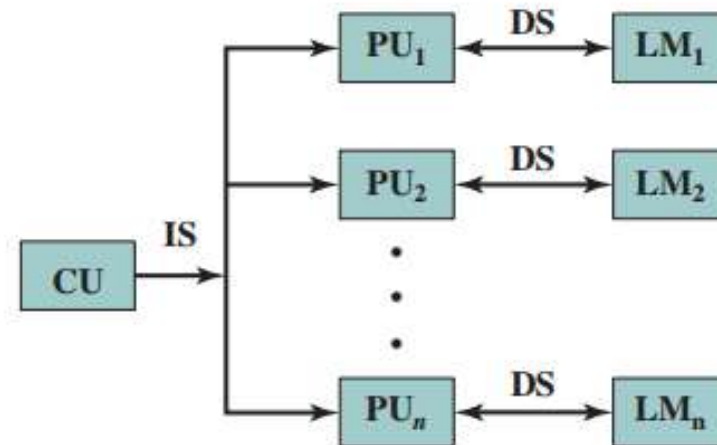


(a) SISD

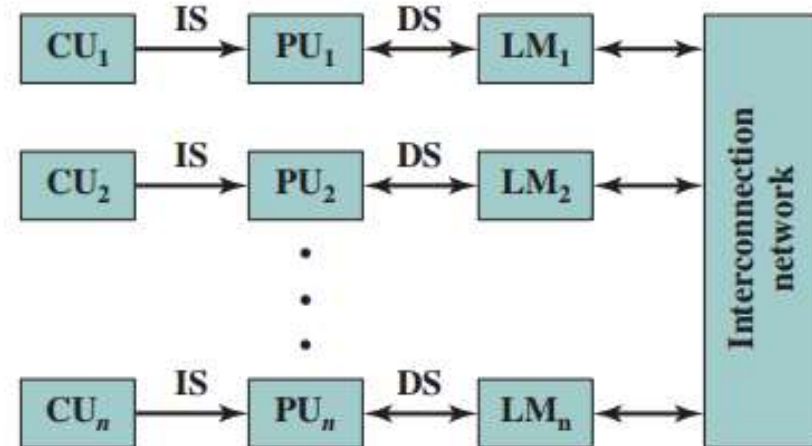


(c) MIMD (with shared memory)

CU = Control unit	SISD = Single instruction,
IS = Instruction stream	= single data stream
PU = Processing unit	SIMD = Single instruction,
DS = Data stream	multiple data stream
MU = Memory unit	MIMD = Multiple instruction,
LM = Local memory	multiple data stream



(b) SIMD (with distributed memory)



(d) MIMD (with distributed memory)

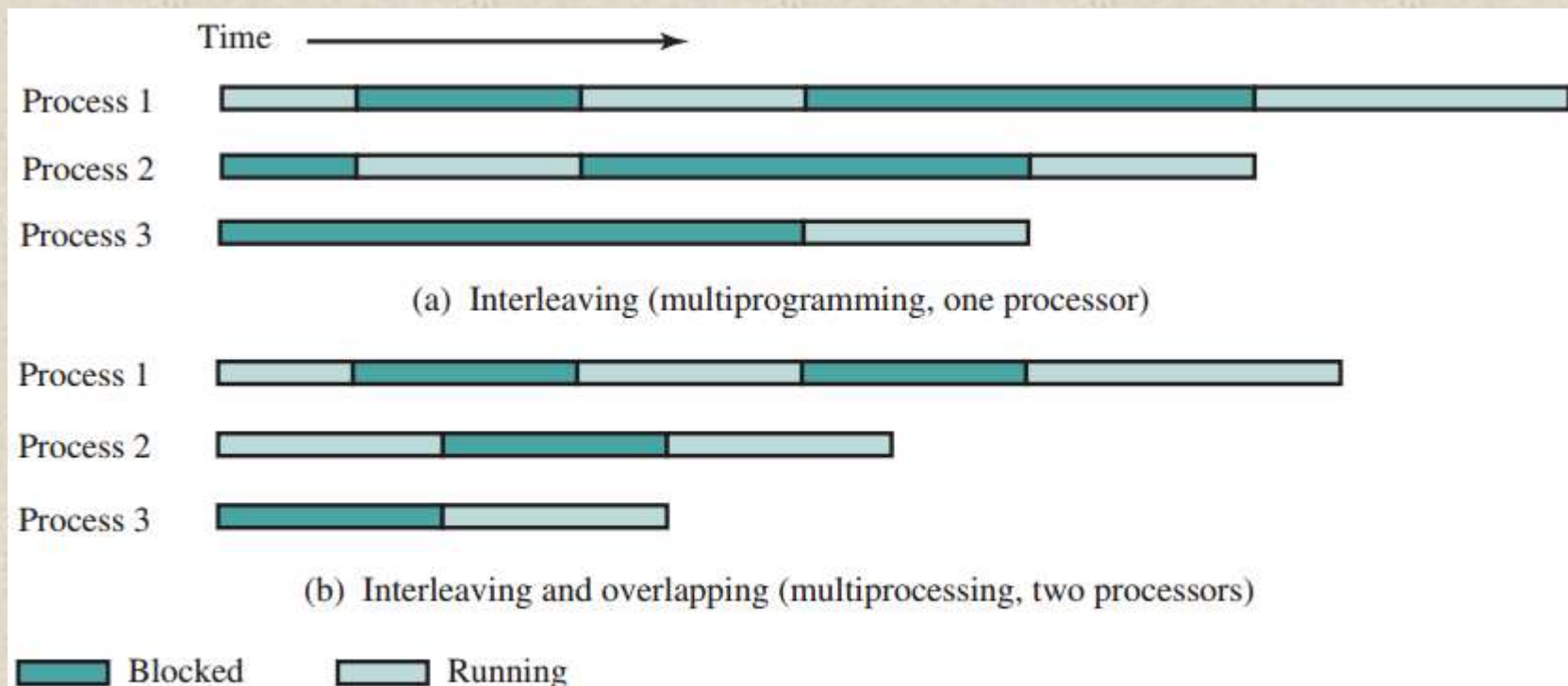
**Figure 17.2** Alternative Computer Organizations

## 17.2- Symmetric Multiprocessor (SMP)

A SMP can be defined as a stand alone computer with the following characteristics:

Two or more similar processors of comparable capacity	<p>Processors share same memory and I/O facilities</p> <ul style="list-style-type: none"><li>• Processors are connected by a bus or other internal connection</li><li>• Memory access time is approximately the same for each processor</li></ul>	<p>All processors share access to I/O devices</p> <ul style="list-style-type: none"><li>• Either through same channels or different channels giving paths to same devices</li></ul>	<p>All processors can perform the same functions (hence “<u>symmetric</u>”)</p>	<p>System controlled by integrated operating system</p> <ul style="list-style-type: none"><li>• Provides interaction between processors and their programs at job, task, file and data element levels</li></ul>
---	---	---	---	---

# Multiprogramming and Multiprocessing

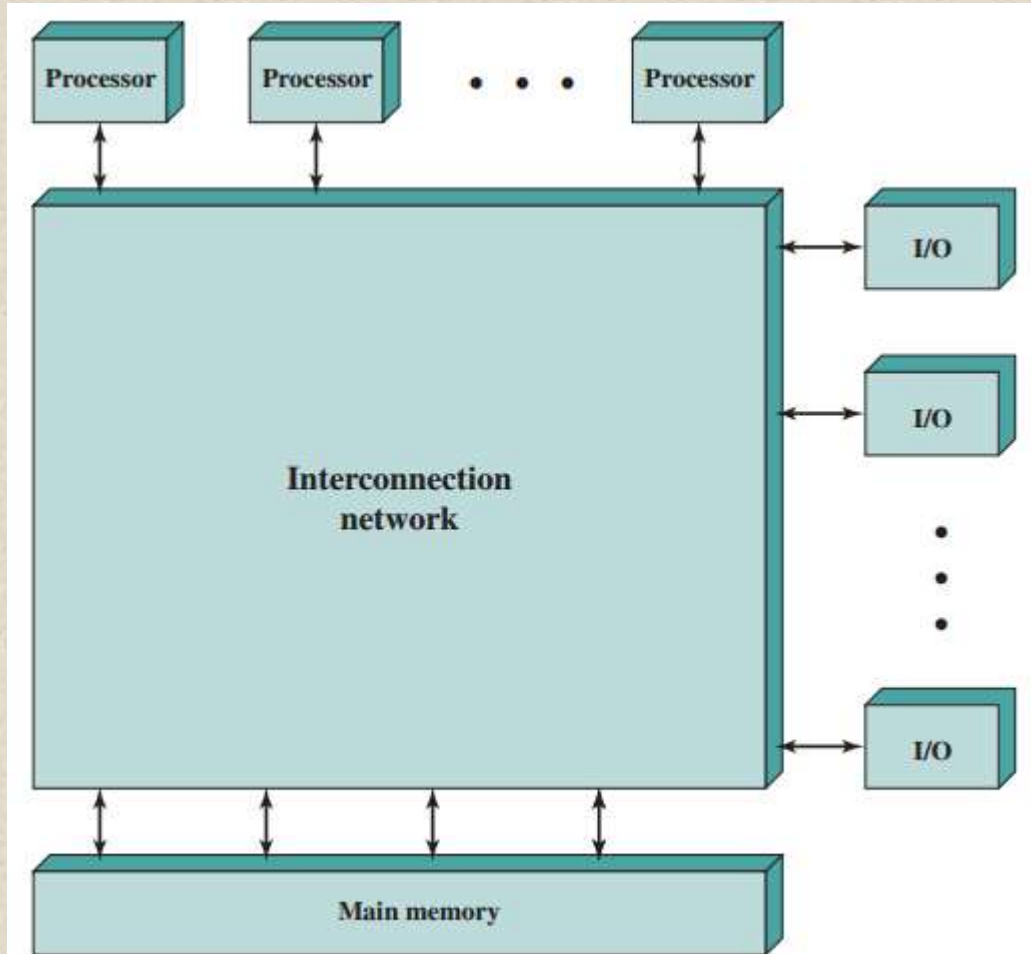


**Figure 17.3** Multiprogramming and Multiprocessing

The operating system of an SMP schedules processes or threads across all of the processors. SMP has a number of potential advantages over a uni-processor organization, including the following: **Performance**, **availability**, **incremental growth** (user can add processors), **scaling** (Vendors can offer a range of products with different configures)



# Organization: Tightly Coupled



**Figure 17.4** Generic Block Diagram of a Tightly Coupled Multiprocessor

- Each processor is self-contained (CU, registers, one or more caches).
- Shared main memory and I/O devices through some form of interconnection mechanism.
- Processors can communicate with each other through memory.
- A processor can exchange signals directly to each other.
- The memory is often organized so that multiple simultaneous accesses to separate blocks of memory are possible.
- In some configurations, each processor may also have its own private main memory and I/O channels in addition to the shared resources.

# Organization: Symmetric Multiprocessor

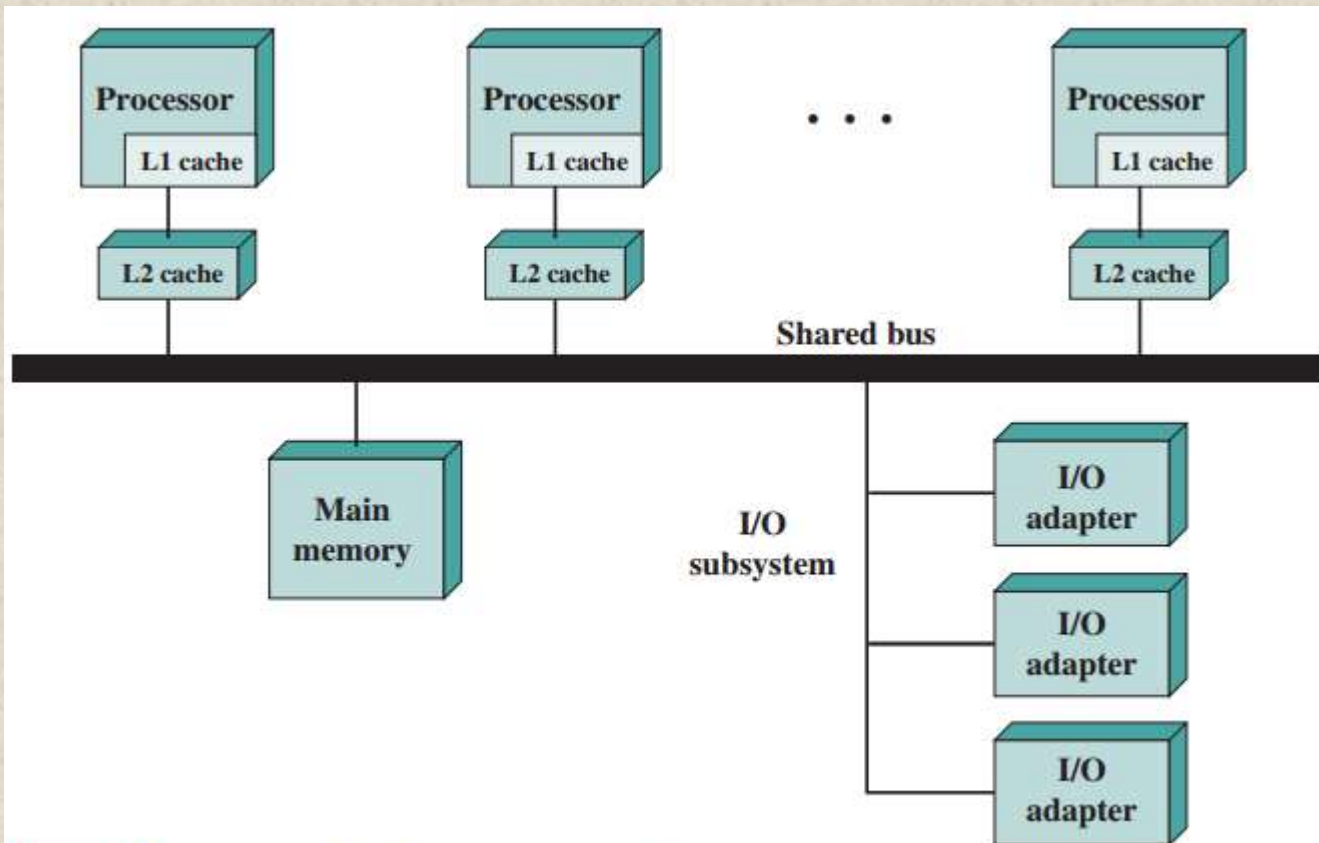


Figure 17.5 Symmetric Multiprocessor Organization

## DMA:

- Addressing: <source, destination>
- Arbitration: Any I/O module can be “master.”
- Time-sharing

- The most common organization for personal computers, workstations, and servers is the time-shared bus. The time-shared bus is the simplest mechanism for constructing a multiprocessor system.
- The structure and interfaces are basically the same as for a single-processor system that uses a bus interconnection. The bus consists of control, address, and data lines



## 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





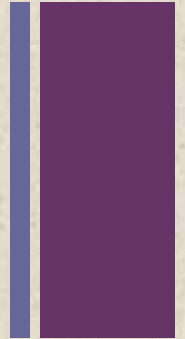
## Disadvantages of the bus organization:



- **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 (center) to allow several processors to **execute** the same OS code (OS service) **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**

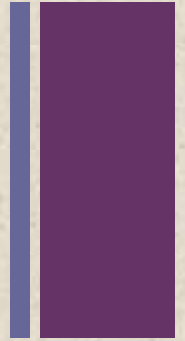
## ■ Synchronization

- 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

mutual exclusion: loại trừ lẫn nhau, cơ chế độc chiếm tài nguyên, một nguyên nhân gây deadlock



# Multiprocessor Operating System Design Considerations...



## ■ Memory management

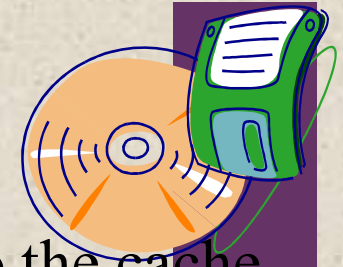
- In addition to dealing with all of the issues found on uniprocessor machines, the OS needs to **exploit** the available **hardware parallelism** to achieve the best performance
- **Paging** mechanisms on different processors must **be coordinated** to enforce **consistency** when several processors share a page or segment and to decide on page replacement

## ■ Reliability and fault tolerance

- OS should provide graceful degradation (suy giảm) 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 and the MESI Protocol

### Review:



**Write back**: Write operations are usually made only to the cache. Main memory is only updated when the corresponding cache line is flushed from the cache → can result in inconsistency

**Write through**: All write operations are made to main memory as well as to the cache, ensuring that main memory is always valid. Even with the write-through policy, inconsistency can occur unless other caches monitor the memory traffic or receive some direct notification of the update



**MESI (modified/exclusive/shared/invalid)** protocol is recommended here.

*Coherent: sticking together – cố kết*

*Consistency: disambiguation- nhất quán, không nhập nhằng*

*Protocol: way including some steps for communication- giao thức*



## + Cache Coherence...



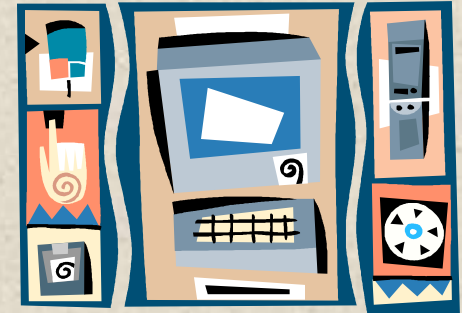
### Software Solutions

- Attempt to avoid the need for additional hardware circuitry and logic by relying on the compiler and operating system to deal with the problem (*không muốn thêm phần cứng*)
- **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



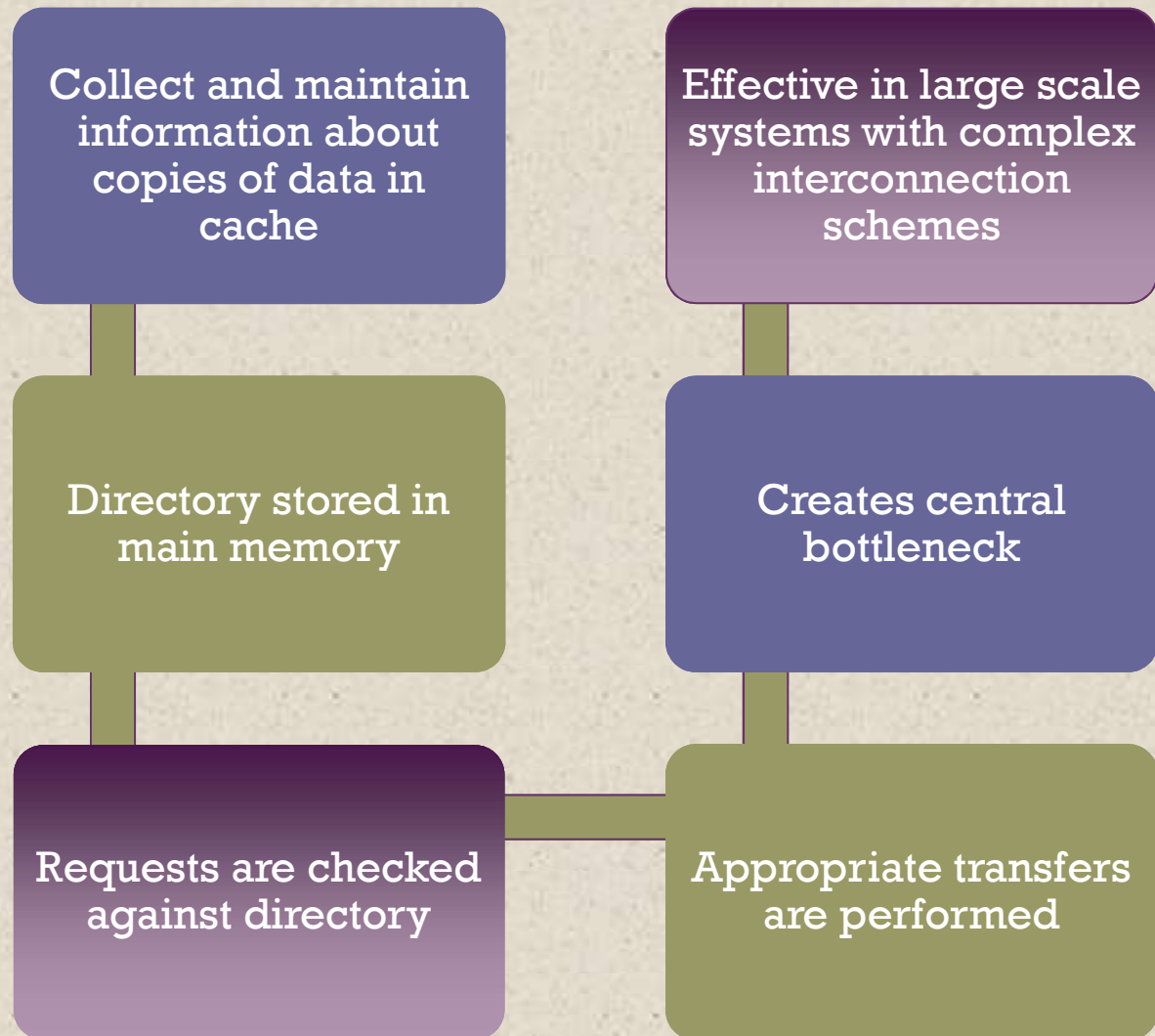
- Generally referred to as **cache coherence protocols**
- These solutions provide **dynamic recognition at run time** of potential **inconsistency** conditions
- Because the problem is only dealt with when it actually arises there is **more effective use of caches**, leading to improved performance 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

*Transparent: unable to see- trong suốt*

*Snoop: spy, rình mò*

# Directory Protocols

There is a centralized controller that is part of the main memory controller



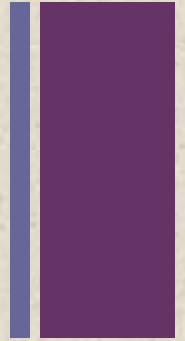
# Snoopy Protocols

- **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)





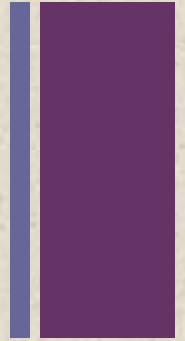
# + Write Invalidate



- **Multiple readers**, but only **one writer** at a time
- When **a write is required**, all other caches of the **line are invalidated (marked)**
- Writing processor then has **exclusive** (độc chiếm-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 (symmetric multi-processor) 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

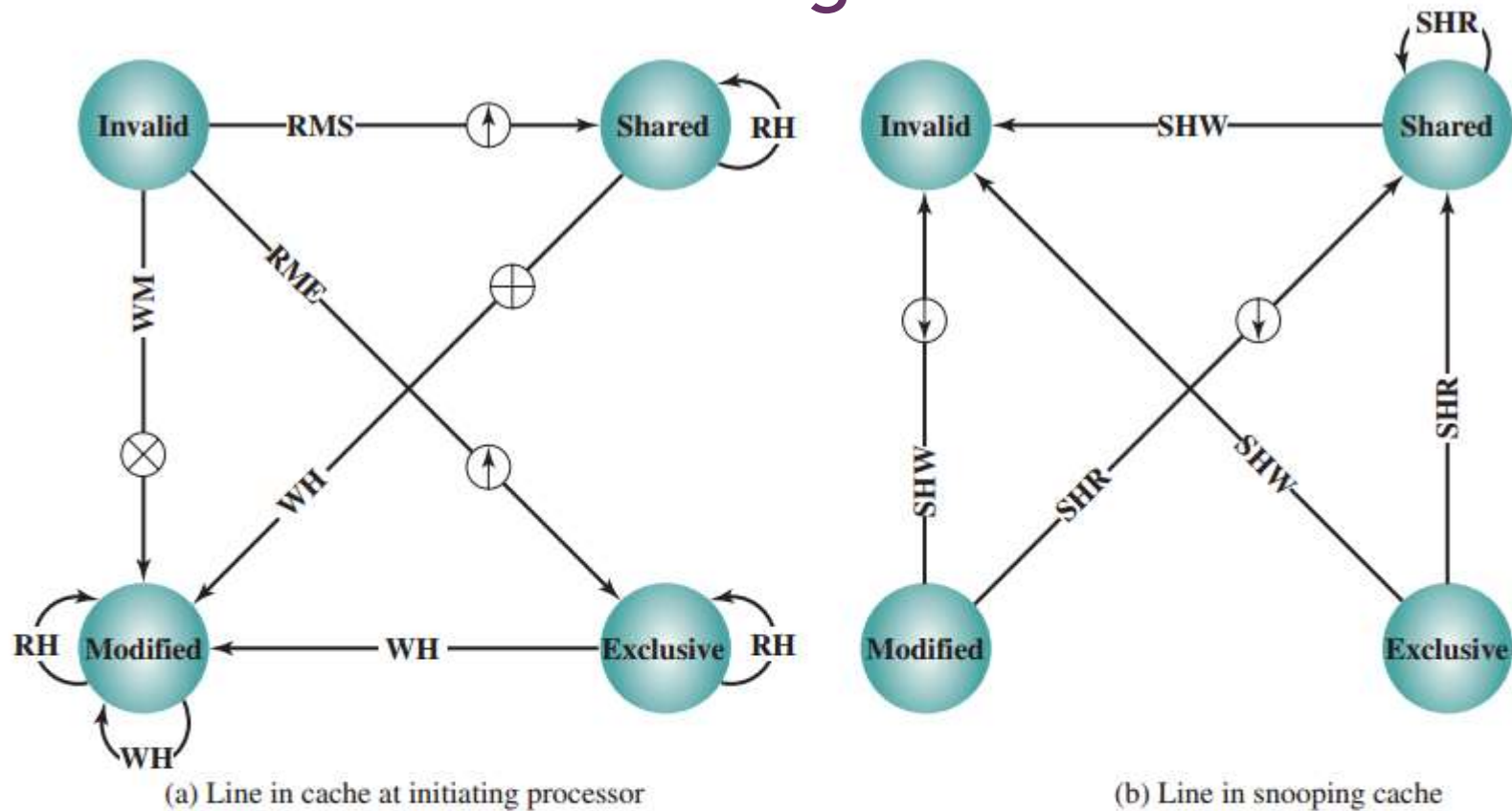
**Table 17.1** MESI Cache Line States

	<b>M</b> <b>Modified</b>	<b>E</b> <b>Exclusive</b>	<b>S</b> <b>Shared</b>	<b>I</b> <b>Invalid</b>
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





Table 17.1 summarizes the meaning of the four states.



# 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

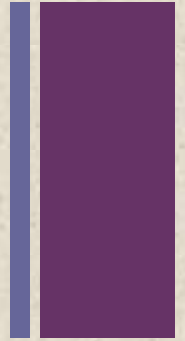
 Dirty line copyback  
 Invalidate transaction  
 Read-with-intent-to-modify  
 Cache line fill

**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 =  $f * IPC$**  // Million Instructions Per second
  - $f$  = 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 → Increase IPC
  - Instruction stream is divided into several smaller streams, known as threads, that can be executed in parallel

# Definitions of Threads and Processes

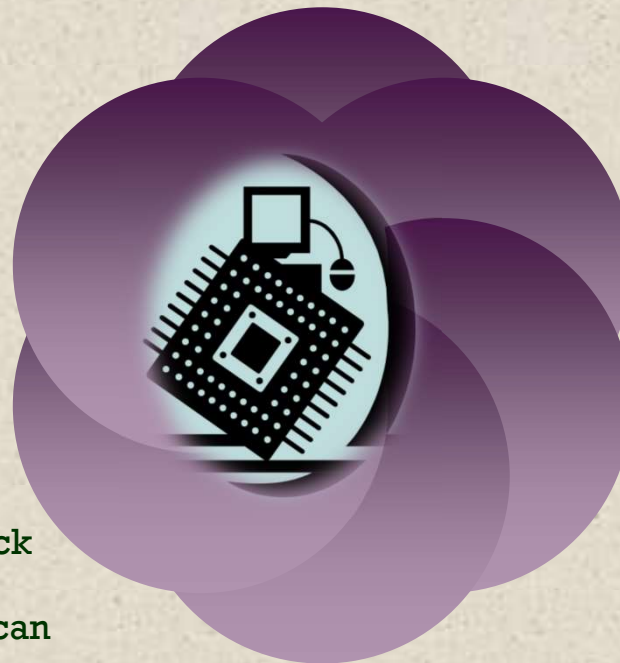
Thread in multithreaded processors may or may not be the same as the concept of software threads in a multiprogrammed operating system

## Thread switch

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

## Thread:

- 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



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

## Process:

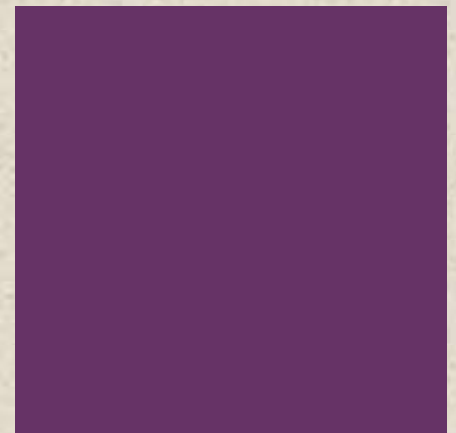
- An instance of program running on computer
- Two key characteristics:
  - Resource ownership
  - Scheduling/execution

## 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

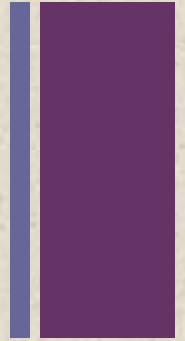
# Implicit and Explicit Multithreading

- **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 (divided)
- 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 (no fine)
- Thread executed until event causes delay (IO)
- Effective on in-order processor
- Avoids pipeline stall (failure)

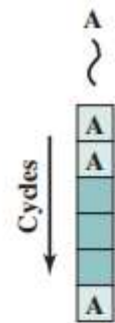
## ■ Chip multiprocessing

- 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

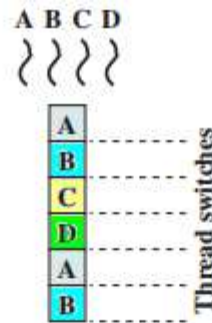
*SMT: Simultaneous Multithreading*



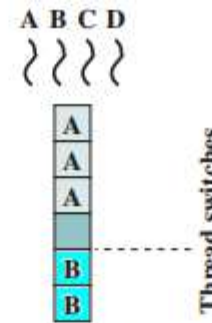
# Approaches to Executing Multiple Threads



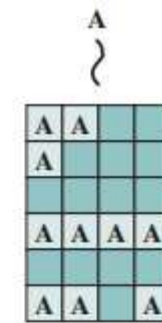
(a) Single-threaded scalar



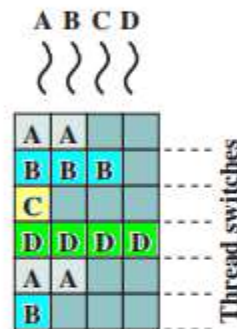
(b) Interleaved multithreading scalar



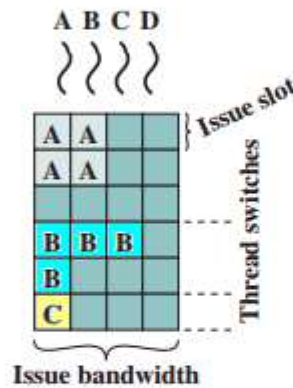
(c) Blocked multithreading scalar



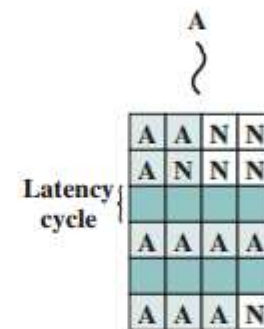
(d) Superscalar



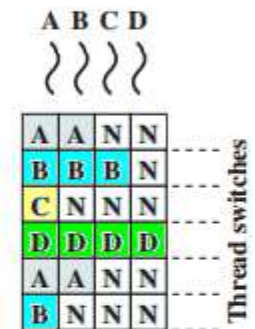
(e) Interleaved multithreading superscalar



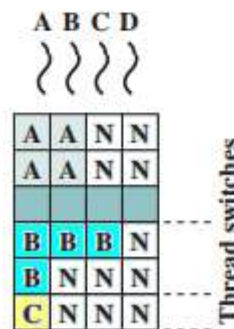
(f) Blocked multithreading superscalar



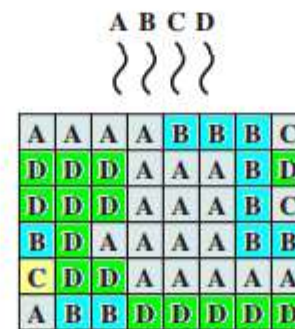
(g) VLIW



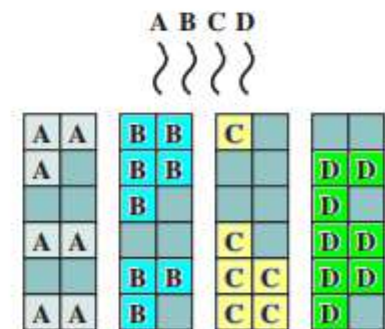
(h) Interleaved multithreading VLIW



(i) Blocked multithreading VLIW



(j) Simultaneous multithreading (SMT)



(k) Chip multiprocessor (multicore)



# 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

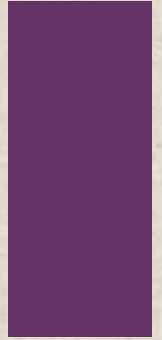
## IBM Power5

- 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





# Exercises



- 17.1 List and briefly define three types of computer system organization.
- 17.2 What are the chief characteristics of an SMP(symmetric multiprocessor)?
- 17.3 What are some of the potential advantages of an SMP compared with a uniprocessor?
- 17.4 What are some of the key OS design issues for an SMP?
- 17.5 What is the difference between software and hardware cache coherent schemes?
- 17.6 What is the meaning of each of the four states in the MESI protocol?

# + 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