

+ Lecture B - 07

Multiprocessing



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

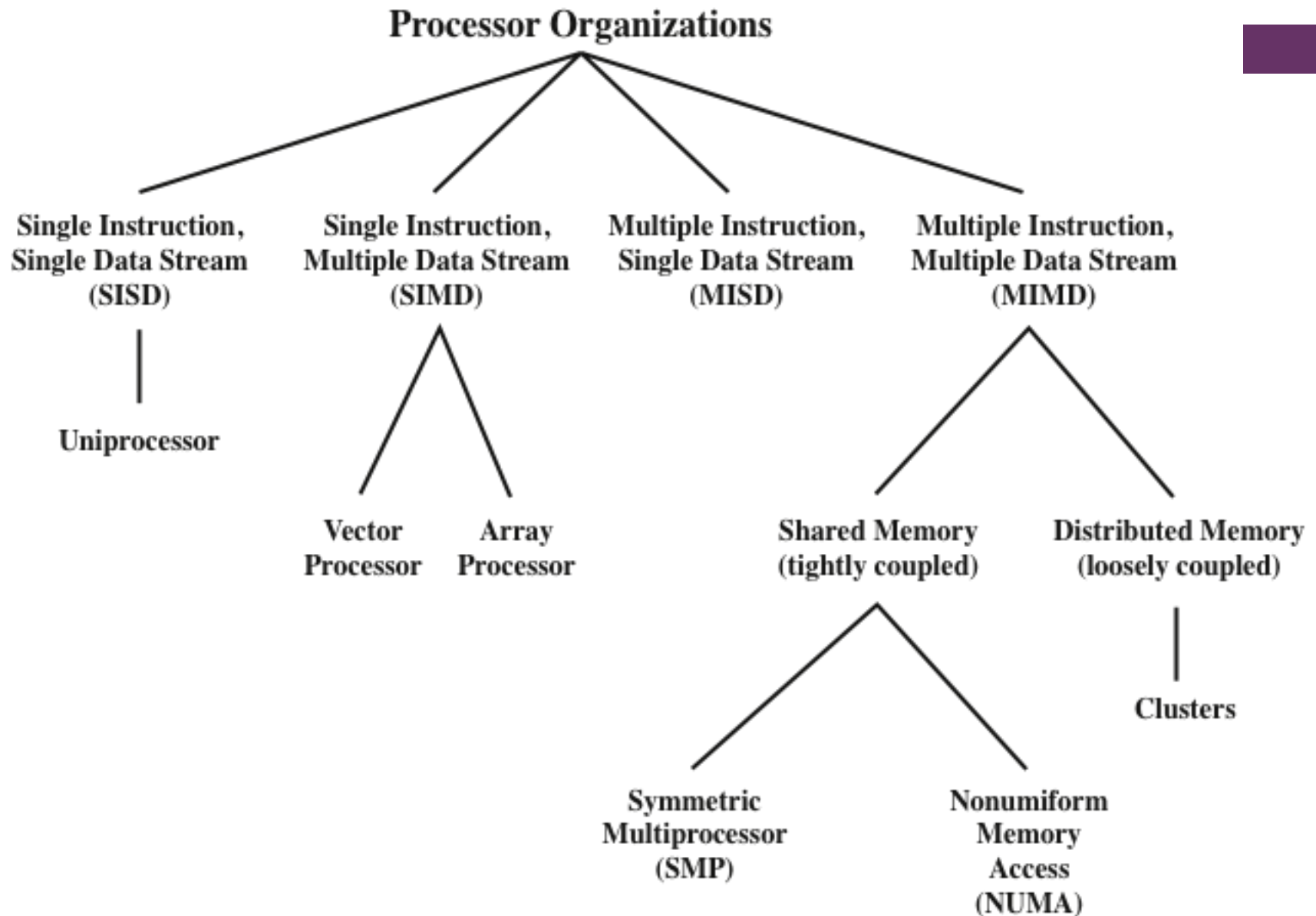
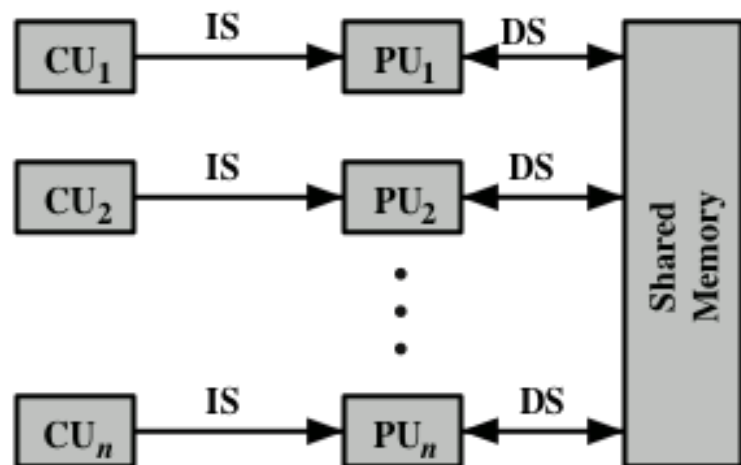


Figure 17.1 A Taxonomy of Parallel Processor Architectures



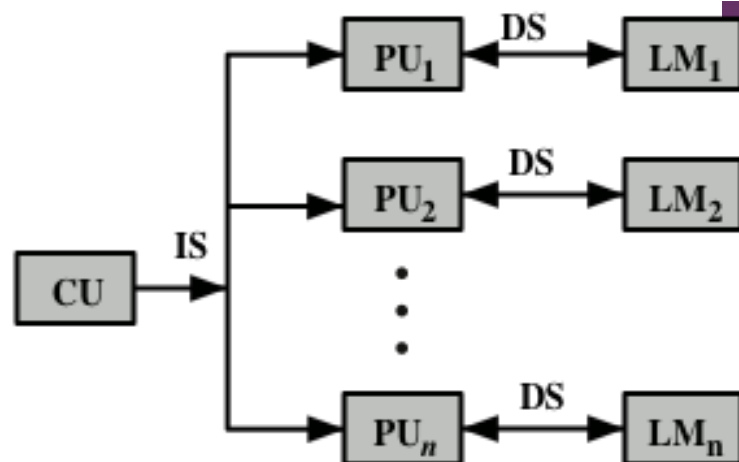
(a) SISD



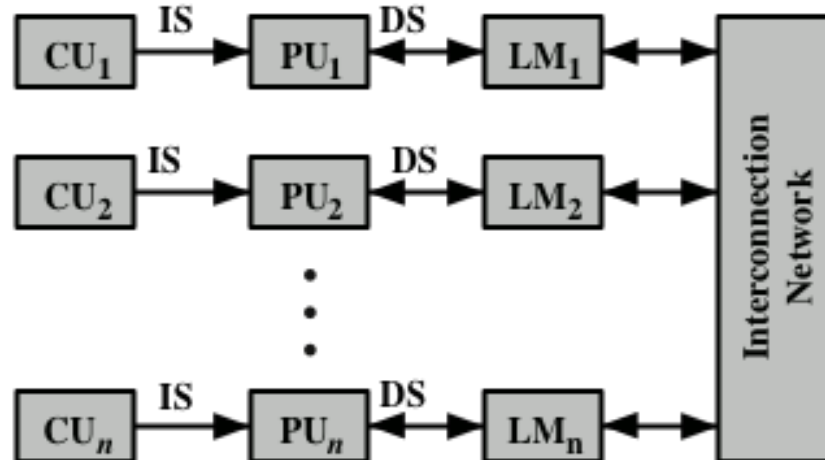
(c) MIMD (with shared memory)

CU = control unit
 IS = instruction stream
 PU = processing unit
 DS = data stream
 MU = memory unit
 LM = local memory

SISD = single instruction,
 single data stream
 SIMD = single instruction,
 multiple data stream
 MIMD = multiple instruction,
 multiple data stream



(b) SIMD (with distributed memory)



(d) MIMD (with distributed memory)

Figure 17.2 Alternative Computer Organizations

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

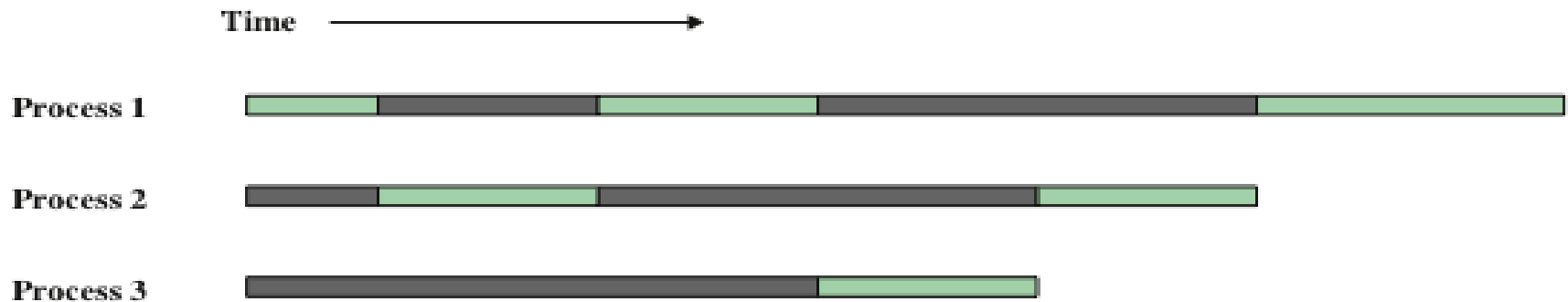
All processors share access to I/O devices

- 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



(a) Interleaving (multiprogramming, one processor)



(b) Interleaving and overlapping (multiprocessing; two processors)

Blocked Running

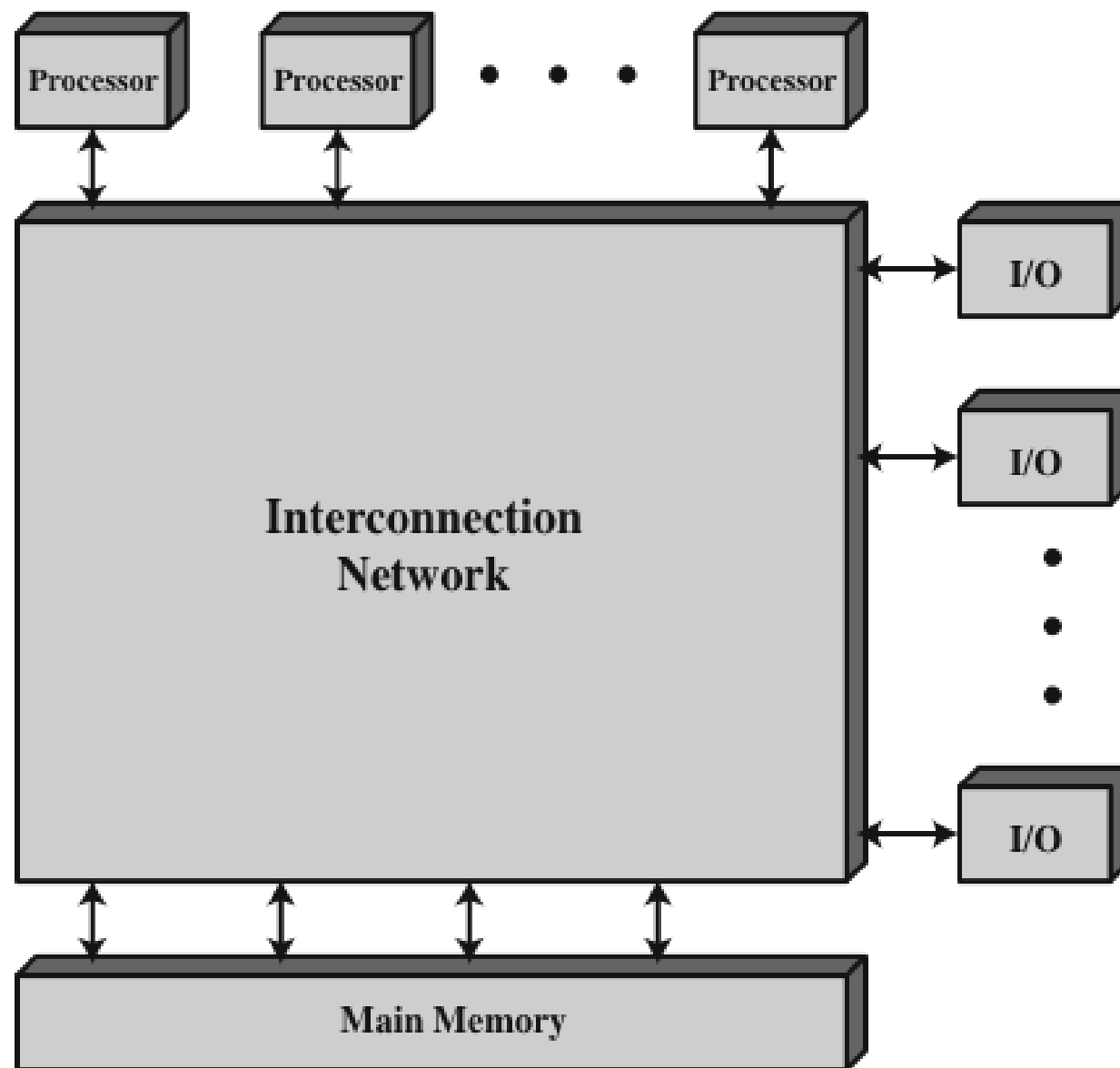


Figure 17.4 Generic Block Diagram of a Tightly Coupled Multiprocessor

Symmetric Multiprocessor Organization

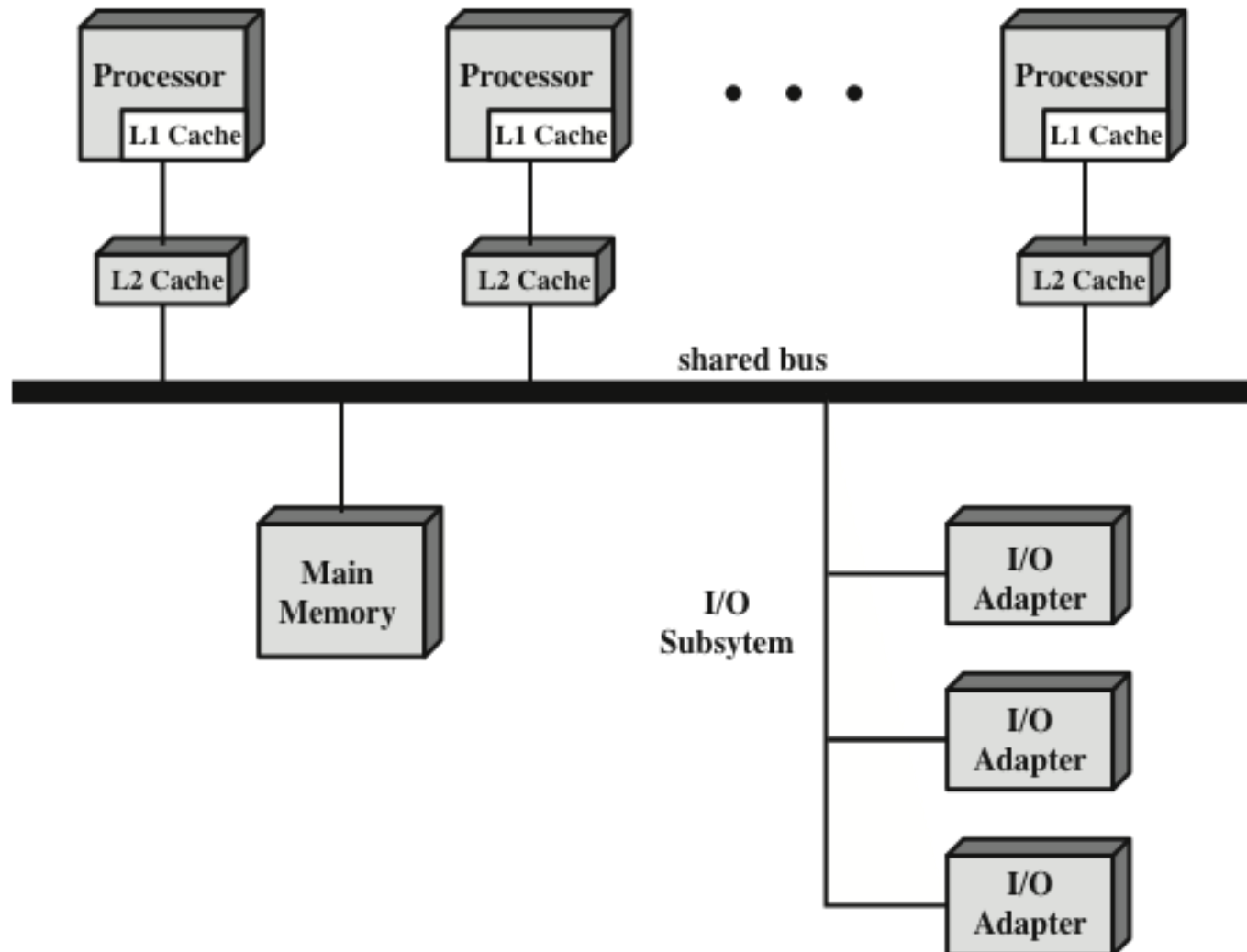


Figure 17.5 Symmetric Multiprocessor Organization



The bus organization has several attractive features:



- Simplicity

- Simplest approach to multiprocessor organization

- Flexibility

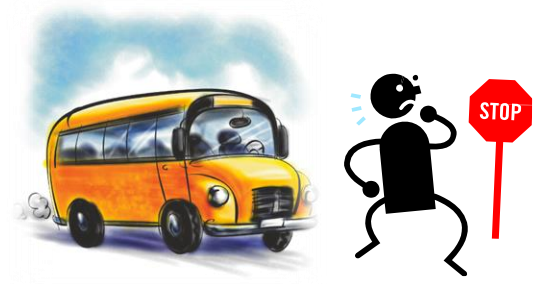
- Generally easy to expand the system by attaching more processors to the bus

- 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 –a..

■ **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

+ Multiprocessor Operating System Design Considerations –b..

■ 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



Cache Coherence

Software Solutions

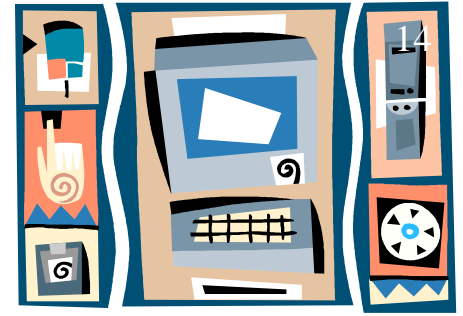


- Attempt to avoid the need for additional hardware circuitry and logic by relying on the compiler and operating system to deal with the problem
- Attractive because the overhead of detecting potential problems is transferred from run time to compile time, and the design complexity is transferred from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



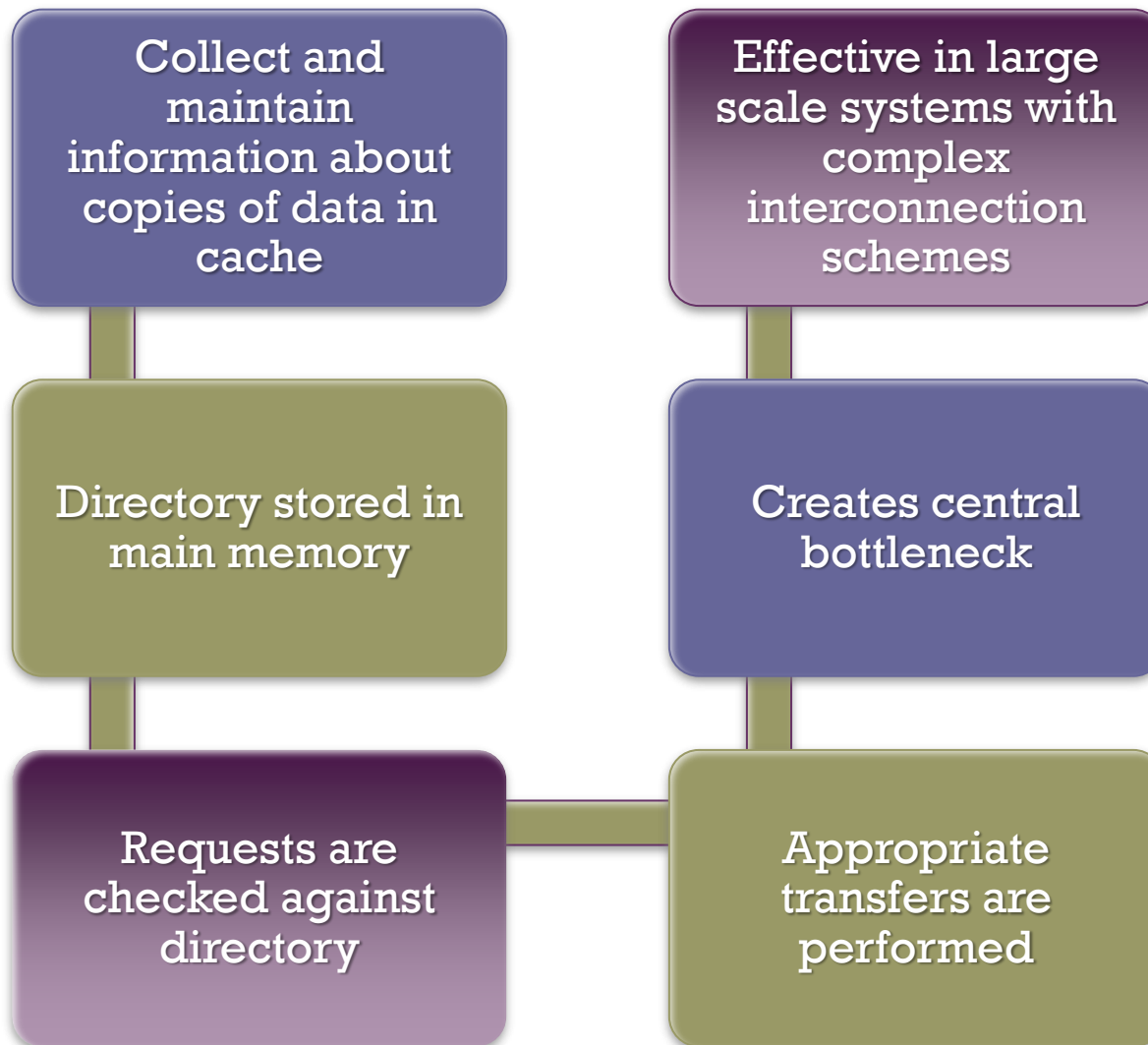
Cache Coherence

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:

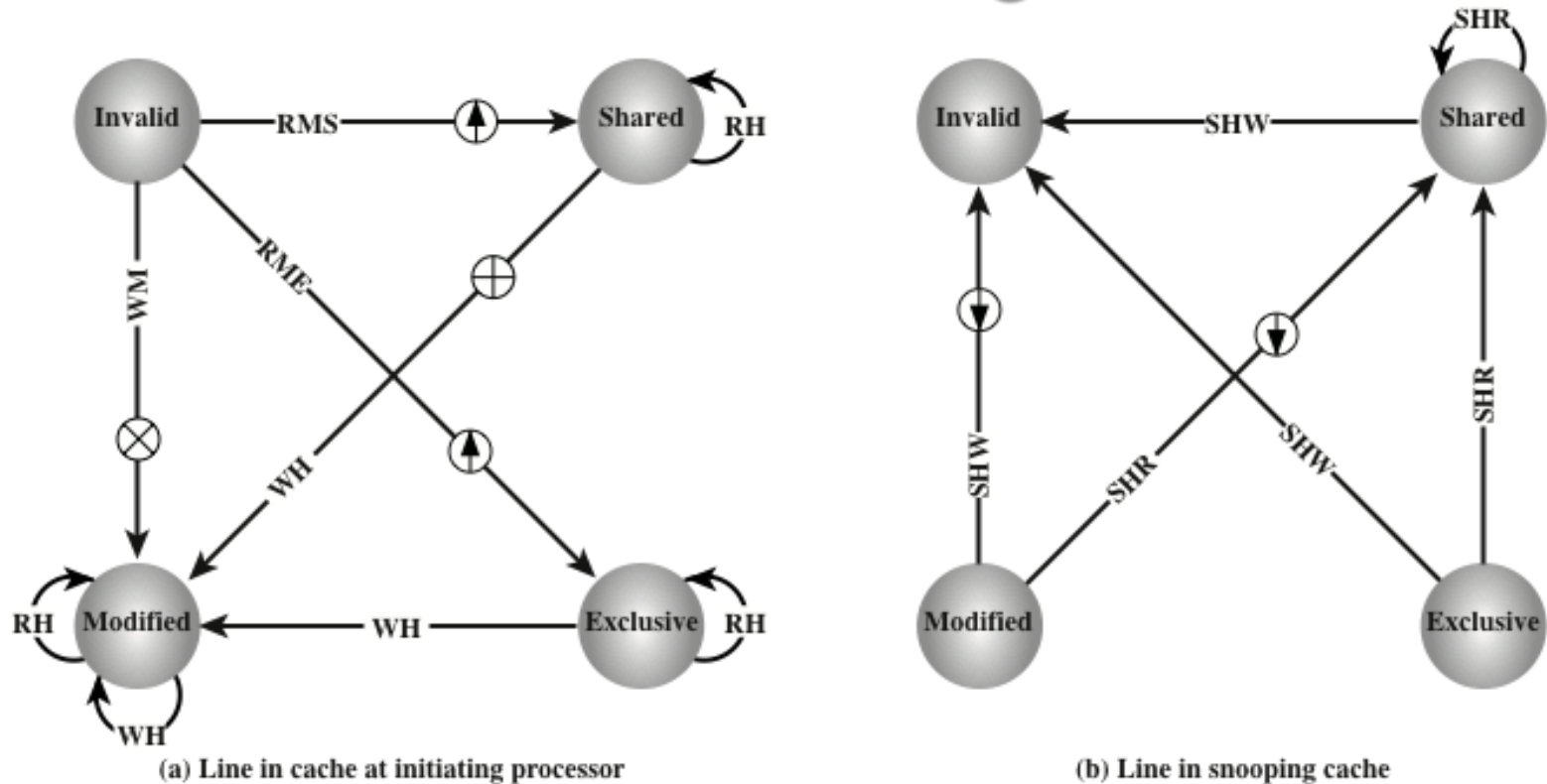
- **Modified**
 - The line in the cache has been modified and is available only in this cache
- **Exclusive**
 - The line in the cache is the same as that in main memory and is not present in any other cache
- **Shared**
 - The line in the cache is the same as that in main memory and may be present in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







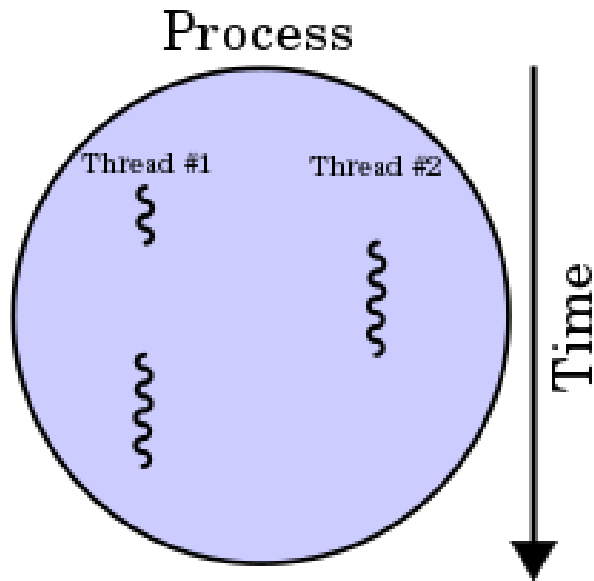
RH	Read hit		Dirty line copyback
RMS	Read miss, shared		
RME	Read miss, exclusive		Invalidate transaction
WH	Write hit		
WM	Write miss		Read-with-intent-to-modify
SHR	Snoop hit on read		
SHW	Snoop hit on write or read-with-intent-to-modify		Cache line fill

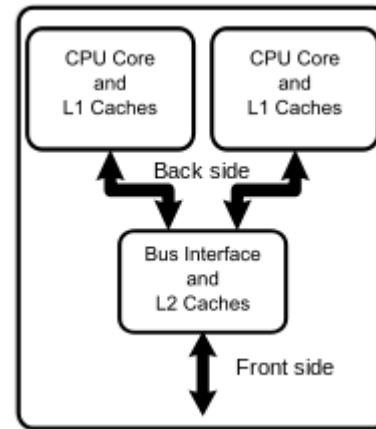
Figure 17.6 MESI State Transition Diagram

Multithreading and Chip Multiprocessors

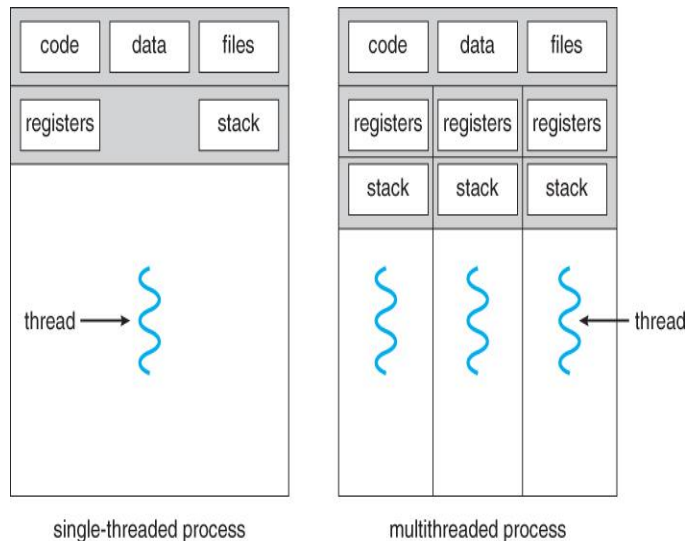
- Processor performance can be measured by the rate at which it executes instructions
- $\text{MIPS rate} = f * \text{IPC}$
 - f = processor clock frequency, in MHz
 - IPC = average instructions per cycle
- Increase performance by increasing clock frequency and increasing instructions that complete during 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
 - multithreading is the ability of a central processing unit (CPU) (or a single core in a multi-core processor) to execute multiple processes or threads concurrently, supported by the operating system.



A process with two threads of execution, running on a single processor

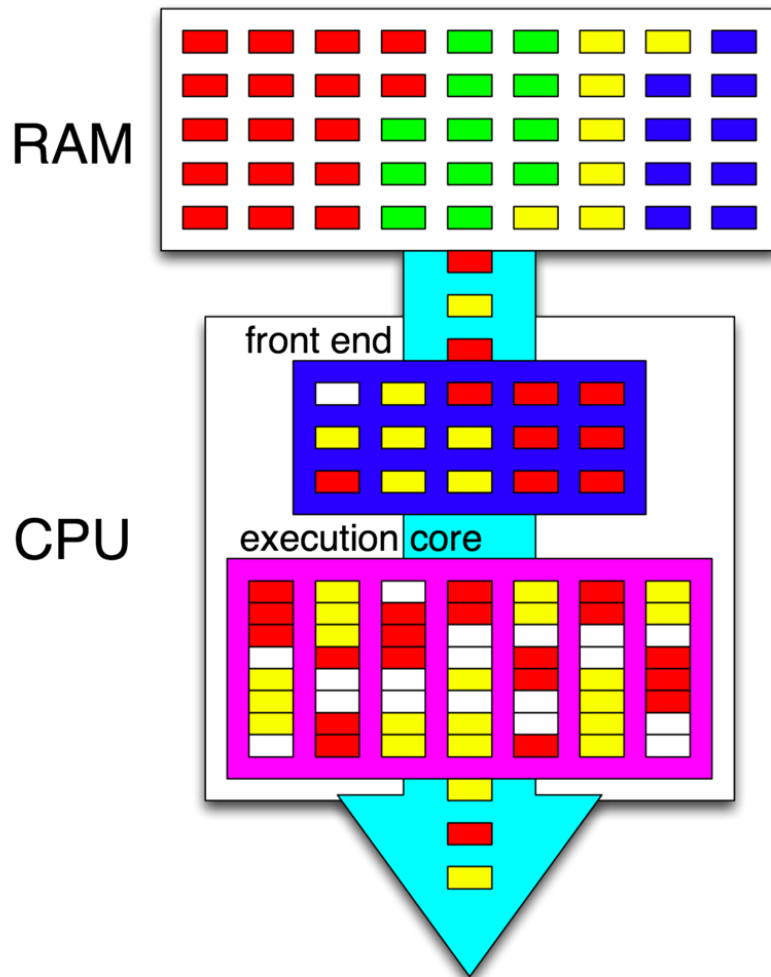


a generic dual-core processor with CPU-local level-1 caches and a shared, on-die level-2 cache



A thread is a lightweight process that can be managed independently by a scheduler. It improves the application performance using parallelism. A thread shares information like data segment, code segment, files etc. with its peer threads while it contains its own registers, stack, counter etc.

Hyper-threading



Hyper-threading (officially called Hyper-Threading Technology or HT Technology and abbreviated as HTT or HT) is Intel's proprietary simultaneous multithreading (SMT) implementation used to improve parallelization of computations (doing multiple tasks at once) performed on x86 microprocessors. It was introduced on Xeon server processors in February 2002 and on Pentium 4 desktop processors in November 2002.[4] Since then, Intel has included this technology in Itanium, Atom, and Core 'i' Series CPUs, among others.[citation needed]

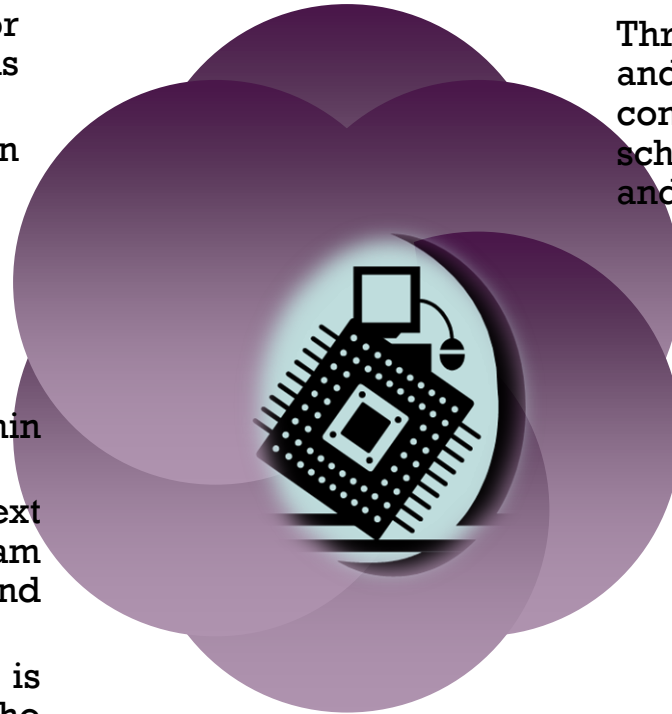
instructions are fetched from RAM (differently colored boxes represent the instructions of four different processes), decoded and reordered by the front end (white boxes represent pipeline bubbles), and passed to the execution core capable of executing instructions from two different programs during the same clock cycle.^{[1][2][3]}

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

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
 - It is a computer science paradigm for building and programming parallel computers designed around the parallel random-access machine (PRAM) parallel computational model. Any single instruction available for execution in a serial program executes immediately
- Implicit multithreading is concurrent execution of multiple threads extracted from single sequential program
 - Implicit threads defined statically by compiler or dynamically by hardware
 - It transfers the creation and management of threading from application developers to compilers and run-time libraries. This, termed implicit threading, is a popular trend today. Implicit threading is mainly the use of libraries or other language support to hide the management of threads

+ Approaches to Explicit Multithreading

- A) 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
- B) Blocked
 - Coarse-grained
 - Thread executed until event causes delay
 - Effective on in-order processor
 - Avoids pipeline stall
- C) Simultaneous (SMT)
 - Instructions are simultaneously issued from multiple threads to execution units of superscalar processor
- D) 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

- For the first two approaches (a and b) , **instructions from different threads are not executed simultaneously**. Instead, the processor is able **to rapidly switch from one thread to another**, using a different set of registers and other context information. This results in a better utilization of the processor's execution resources and avoids a **large penalty due to cache misses and other latency events**.
- The **SMT approach** involves true **simultaneous execution of instructions from different threads**, using **replicated execution resources**.
- **Chip multiprocessing** also enables simultaneous execution of instructions from **different threads**.

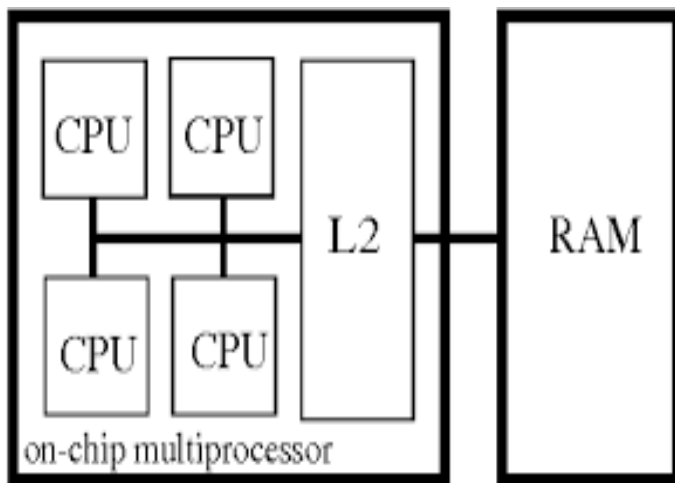
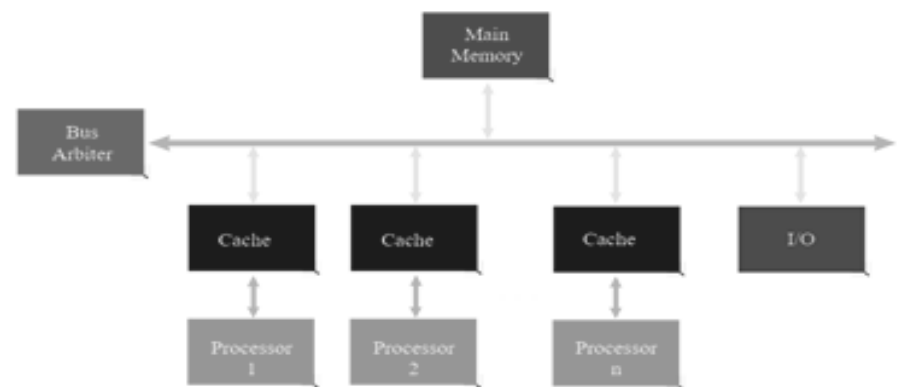
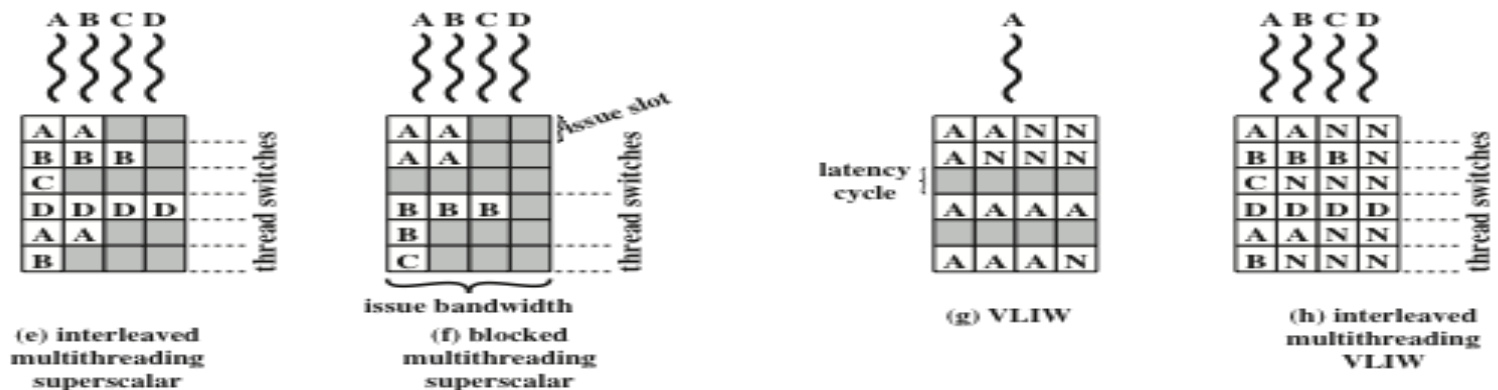
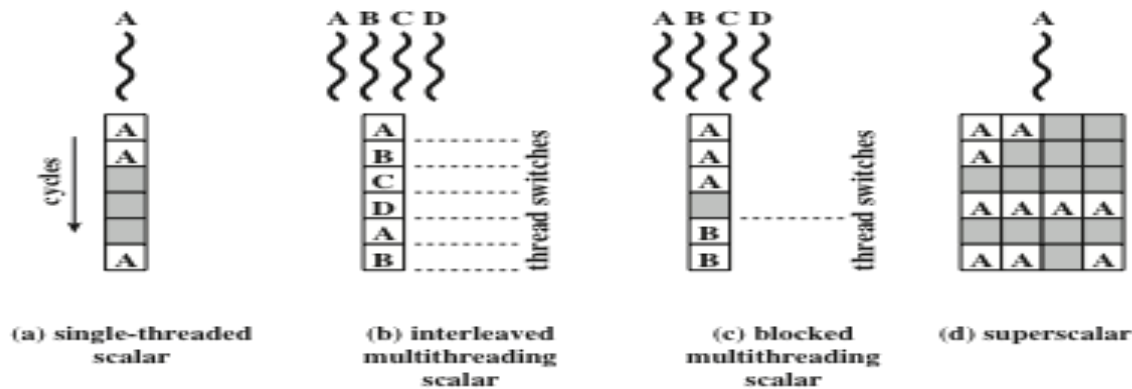


Figure 1. An ACMP with four embedded cores.



SMT approach



Superscalar – several instructions are loaded at once and, as far as possible, are executed simultaneously, shortening the time taken to run the whole program. Multiple Pipelines

Approaches to Executing Multiple Threads

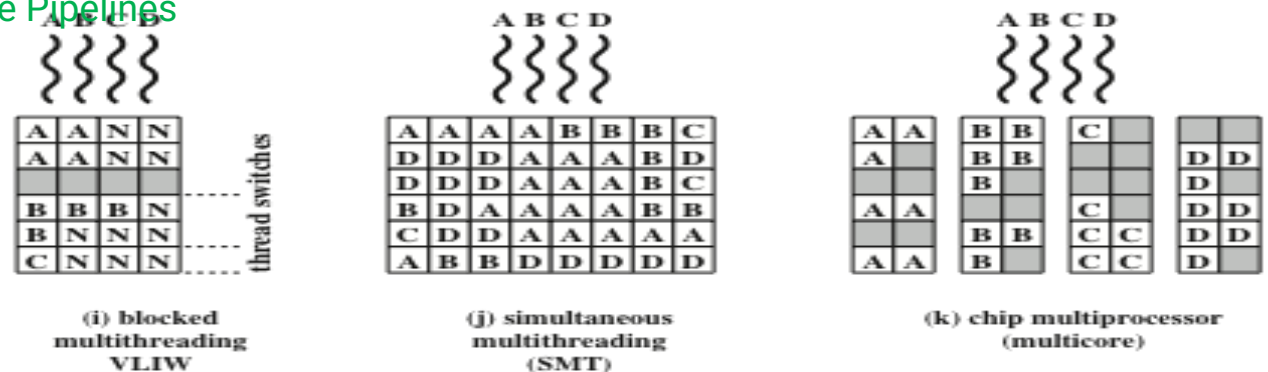


Figure 17.7 Approaches to Executing Multiple Threads

- Figure 17.7 illustrates some of the possible pipe-line architectures that involve multithreading and contrasts these with approaches that do not use multithreading.
- Each horizontal row represents the potential issue slot or slots for a single execution cycle; that is, **the width of each row corresponds to the maximum number of instructions that can be issued in a single clock cycle.**
- The vertical dimension represents the time sequence of clock cycles. An **empty (shaded) slot** represents an **unused execution slot** in one pipeline. A no operation is indicated by N.

The first three illustrations in Figure 17.7 show different approaches with a scalar (i.e., single-issue) processor:

1. **Single-threaded scalar:** This is the simple pipeline found in traditional RISC and CISC machines, with **no multithreading**.
2. **Interleaved multithreaded scalar:** This is the easiest multithreading approach to implement. **By switching from one thread to another at each clock cycle**, the pipeline stages can be kept fully occupied, or close to fully occupied. The hardware must be capable of switching from one thread context to another between cycles.
3. **Blocked multithreaded scalar:** In this case, a single thread is executed until a latency event occurs that would stop the pipeline, at which time the processor switches to another thread.

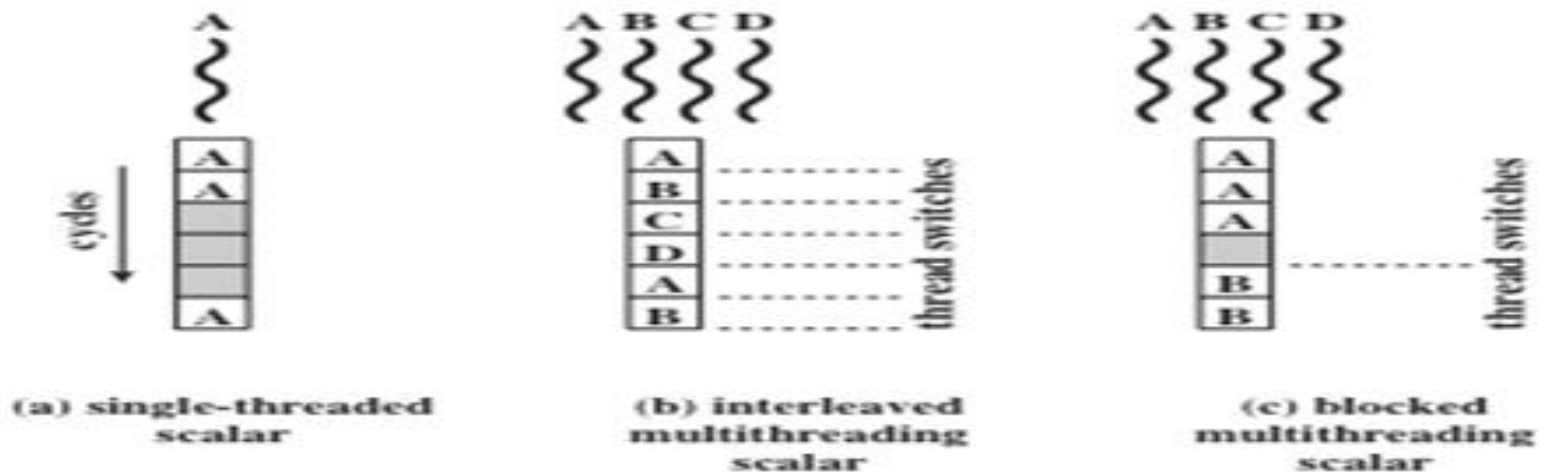
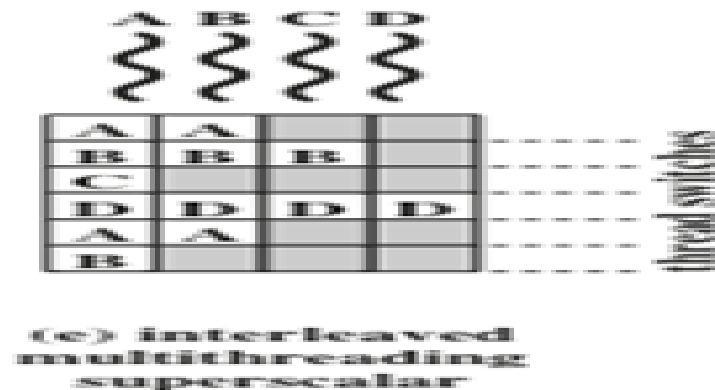


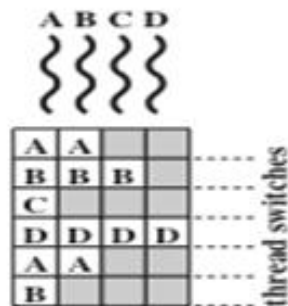
Figure 17.7c shows a situation in which the time to perform a thread switch is one cycle, whereas Figure 17.7b shows that thread switching occurs in zero cycles.

- In the case of **interleaved multithreading**, it is assumed that there are **no control or data dependencies between threads**, which simplifies the pipeline design and therefore should allow a thread switch with no delay. However, depending on the specific design and implementation, block multithreading may require a clock cycle to perform a thread switch
- Although **interleaved multithreading appears to offer better processor utilization** than **blocked multithreading**, it does so at the sacrifice of single-thread performance. The multiple threads compete for cache resources, which raises the probability of a cache miss for a given thread.

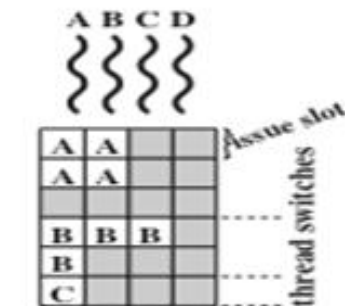


More opportunities for parallel execution are available if the processor can issue multiple instructions per cycle. Figures 17.7d through 17.7i illustrate a number of variations among processors that have hardware for issuing four instructions per cycle.

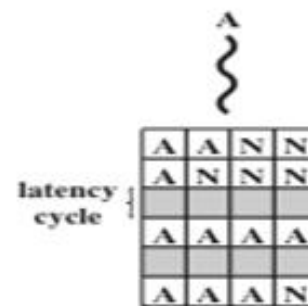
Very long instruction word (VLIW) processor allows programs to explicitly specify instructions to execute in parallel. This design is intended to allow higher performance without the complexity inherent in some other designs



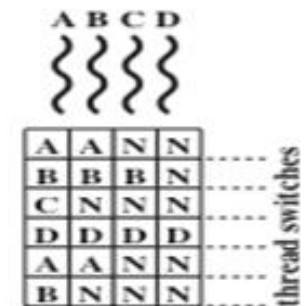
(e) interleaved multithreading superscalar



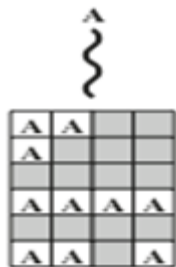
(f) blocked multithreading superscalar



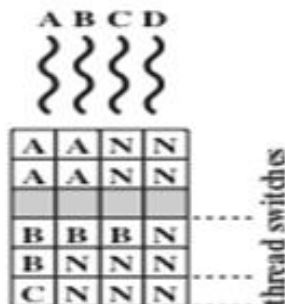
(g) VLIW



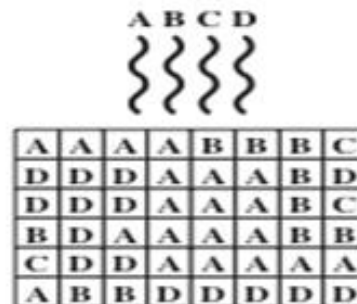
(h) interleaved multithreading VLIW



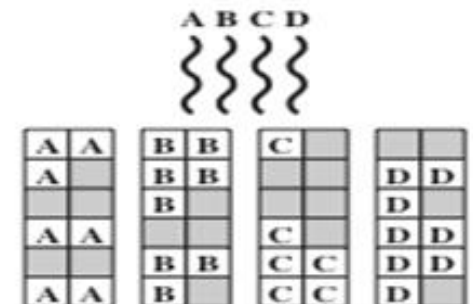
(d) superscalar



(i) blocked multithreading VLIW



(j) simultaneous multithreading (SMT)



(k) chip multiprocessor (multicore)

Example Systems

34

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

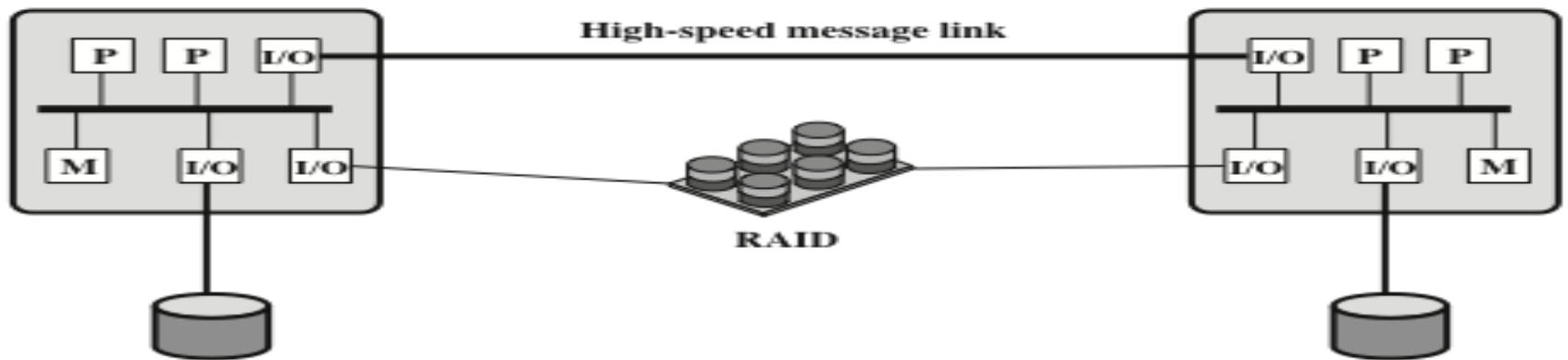
Clusters

- 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

- **Absolute scalability:** It is possible to create large clusters that far surpass the power of even the largest standalone machines. A cluster can have tens, hundreds, or even thousands of machines, each of which is a multiprocessor.
- **Incremental scalability:** A cluster is configured in such a way that it is possible to add new systems to the cluster in small increments. Thus, a user can start out with a modest system and expand it as needs grow, without having to go through a major upgrade in which an existing small system is replaced with a larger system.
- **High availability:** Because each node in a cluster is a standalone computer, the failure of one node does not mean loss of service. In many products, fault tolerance is handled automatically in software.
- **Superior price/performance:** By using commodity building blocks, it is possible to put together a cluster with equal or greater computing power than a single large machine, at much lower cost.



(a) Standby server with no shared disk



(b) Shared disk

Figure 17.9 Cluster Configurations

Clustering Methods: Benefits and Limitations

38

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

39

- How failures are managed depends on the clustering method used: Highly available clusters and Fault tolerant clusters

1. A highly available cluster offers a high probability that all resources will be in service. If a failure occurs, such as a system goes down or a disk volume is lost, then the queries in progress are lost. Any lost query, if retried, will be serviced by a different computer in the cluster. However, the cluster operating system makes no guarantee about the state of partially executed transactions. This would need to be handled at the application level.
2. A fault-tolerant cluster ensures that all resources are always available. This is achieved by the use of redundant shared disks and mechanisms for backing out uncommitted transactions and committing completed transactions.

Operating System Design Issues

■ 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

41

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

Three approaches are:

Parallelizing compiler

- 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

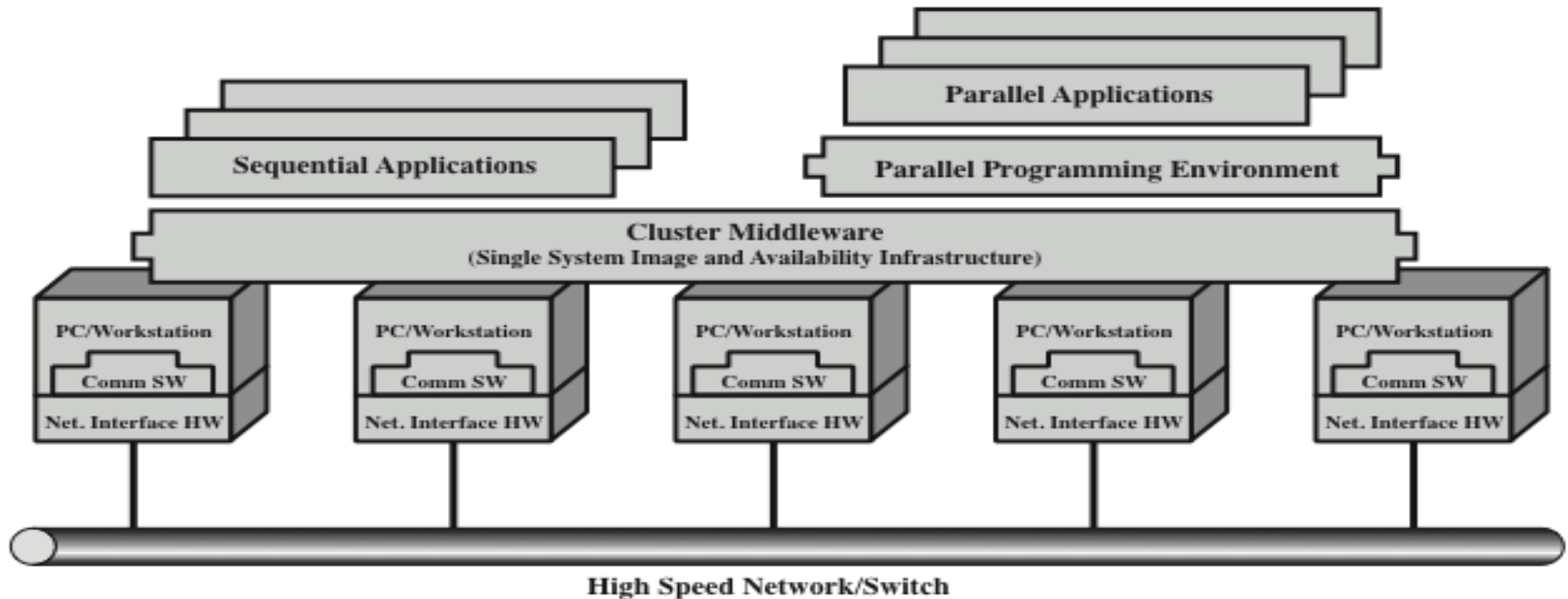
Parallelized application

- 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



The individual computers are connected by some high-speed LAN or switch hardware. Each computer is capable of operating independently. In addition, a **middleware layer** of software is installed in each computer to enable cluster operation. The cluster middleware provides a unified system image to the user, known as a **single-system image**. The middleware is also responsible for providing high availability, by means of **load balancing and responding to failures in individual components**.

A cluster will also include software tools for enabling the efficient execution of programs that are capable of parallel execution.

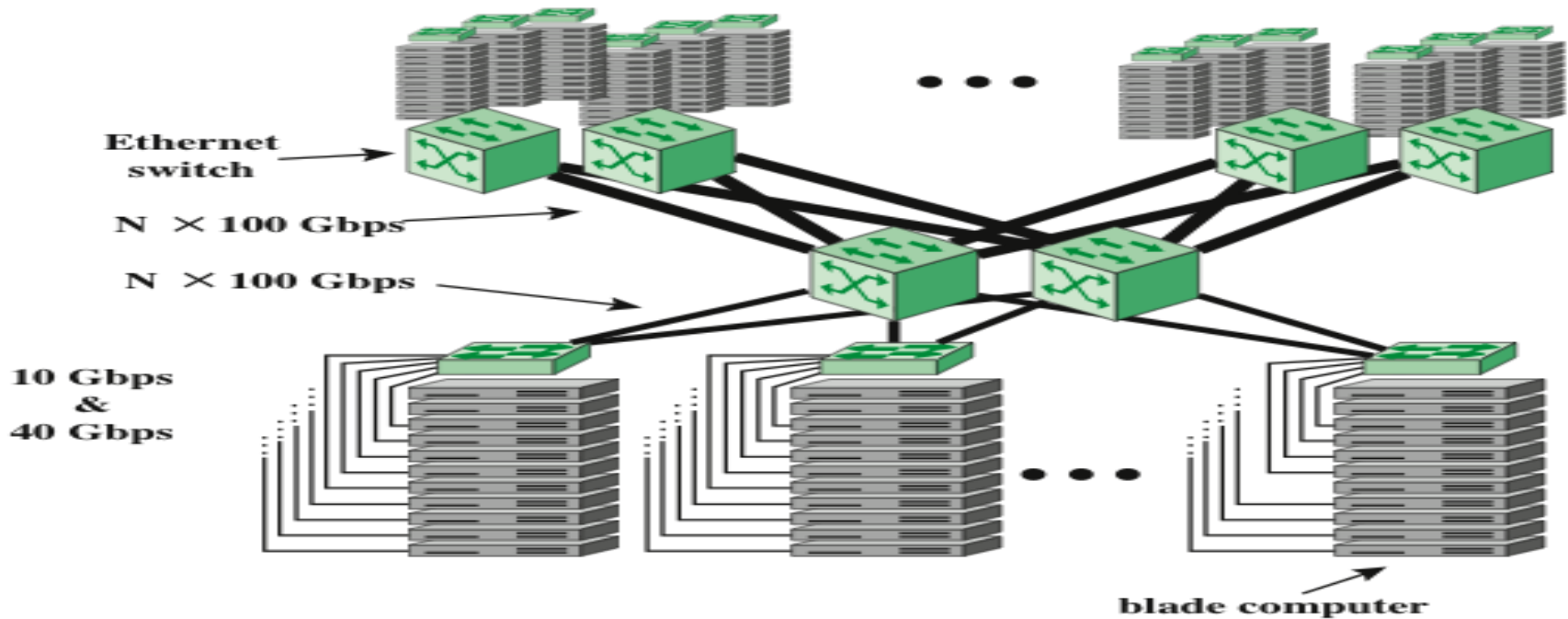


Figure 17.11 Example 100-Gbps Ethernet Configuration for Massive Blade Server Site

A blade server is a server architecture that **houses multiple server modules** (“blades”) in a single chassis. It is widely used in **data centers** to save space and improve system management. Either self-standing or rack mounted, the chassis provides the power supply, and each blade has its own processor, memory, and hard disk.

The trend at large data centers, with substantial banks of blade servers, is the **deployment of 10-Gbps ports on individual servers to handle the massive multimedia traffic** provided by these servers. Such arrangements are stressing the on-site **Ethernet switches needed to interconnect large numbers of servers**. A **100-Gbps rate provides the bandwidth required to handle the increased traffic load**. The 100-Gbps Ethernet switches are deployed in switch uplinks inside the data center as well as providing interbuilding, intercampus, wide area connections for enterprise networks.

Clusters Compared to SMP

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

SMP

- Easier to manage and configure
- Much closer to the original single processor model for which nearly all applications are written
- Less physical space and lower power consumption
- Well established and stable

Clustering

- Far superior in terms of incremental and absolute scalability
- Superior in terms of availability
- All components of the system can readily be made highly redundant

Over the long run, however, the advantages of the cluster approach are likely to result in clusters dominating the **high-performance server market**. Clusters are far superior to SMPs in terms of incremental and absolute scalability. Clusters are also superior in terms of availability, because all components of the system can readily be made highly redundant.

Nonuniform Memory Access (NUMA)

45

- 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

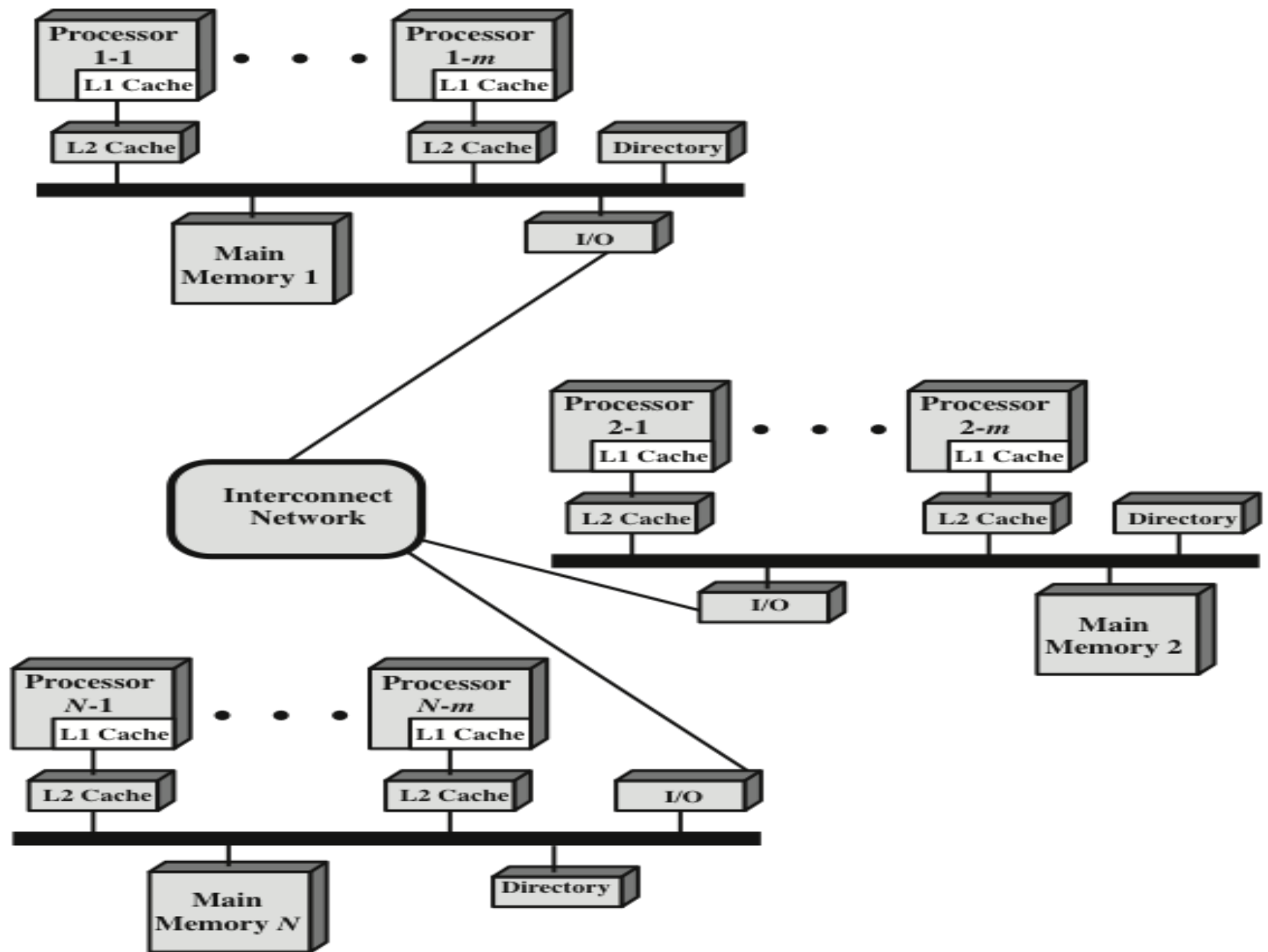


Figure 17.12 CC-NUMA Organization

Figure 17.12 depicts a typical CC-NUMA organization. There are multiple independent nodes, each of which is, in effect, an SMP organization. Thus, each node contains multiple processors, each with its own L1 and L2 caches, plus main memory. The node is the basic building block of the overall CC-NUMA organization. For example, each Silicon Graphics Origin node includes two MIPS R10000 processors; each Sequent NUMA-Q node includes four Pentium II processors. The nodes are interconnected by means of some communications facility, which could be a switching mechanism, a ring, or some other networking facility.

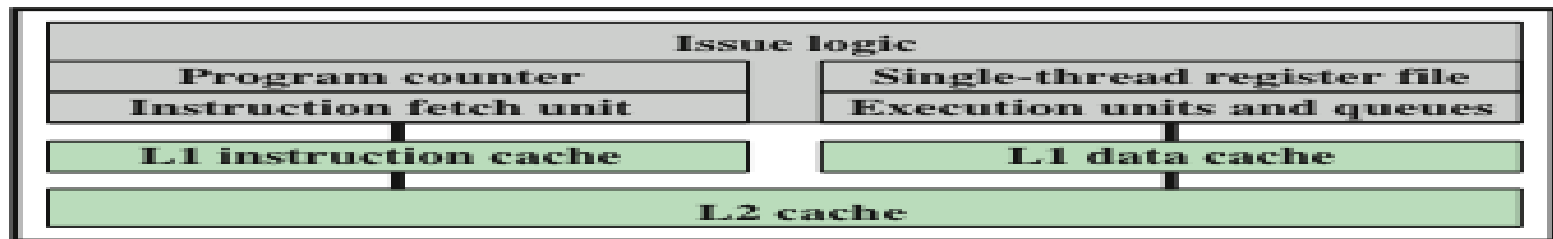
Each node in the CC-NUMA system includes some main memory. From the point of view of the processors, however, there is only a single addressable memory, with each location having a unique system wide address. When a processor initiates a memory access, if the requested memory location is not in that processor's cache, then the L2 cache initiates a fetch operation. If the desired line is in the local portion of the main memory, the line is fetched across the local bus. If the desired line is in a remote portion of the main memory, then an automatic request is sent out to fetch that line across the interconnection network, deliver it to the local bus, and then deliver it to the requesting cache on that bus. All of this activity is automatic and transparent to the processor and its cache.

In this configuration, cache coherence is a central concern. Although implementations differ as to details, in general terms we can say that each node must maintain some sort of directory that gives it an indication of the location of various portions of memory and also cache status information.

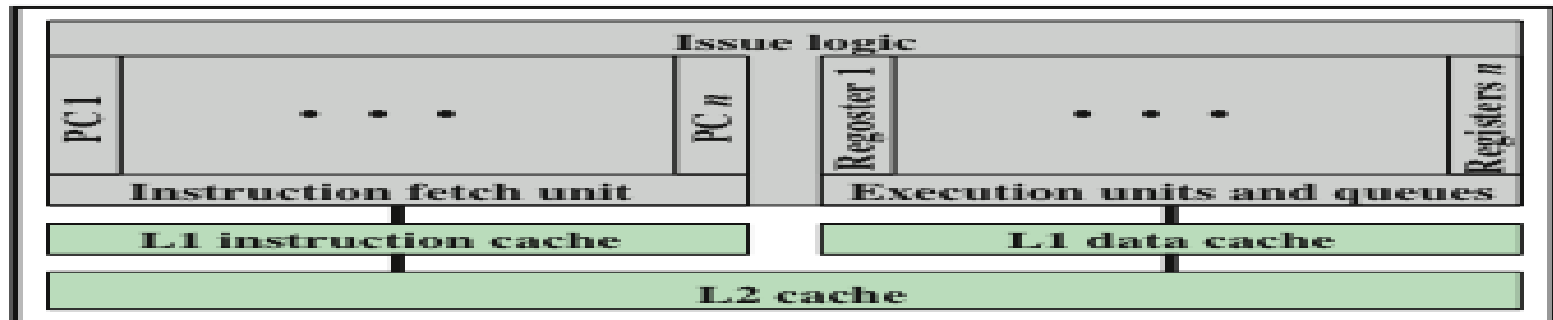
NUMA Pros and Cons

- 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

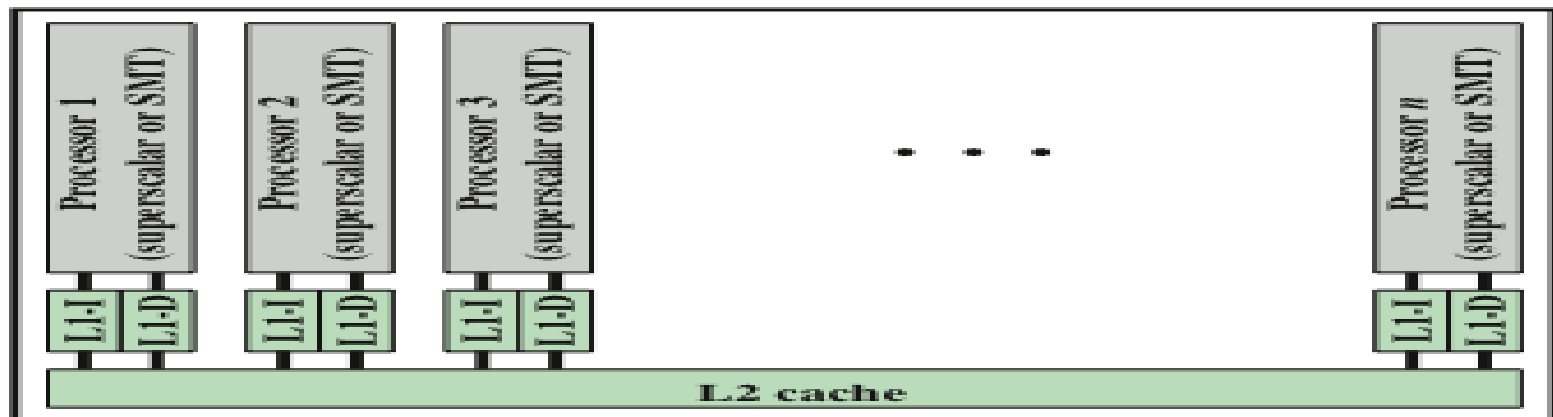
Multi-Core Computing: Alternative Chip Organization



(a) Superscalar



(b) Simultaneous multithreading

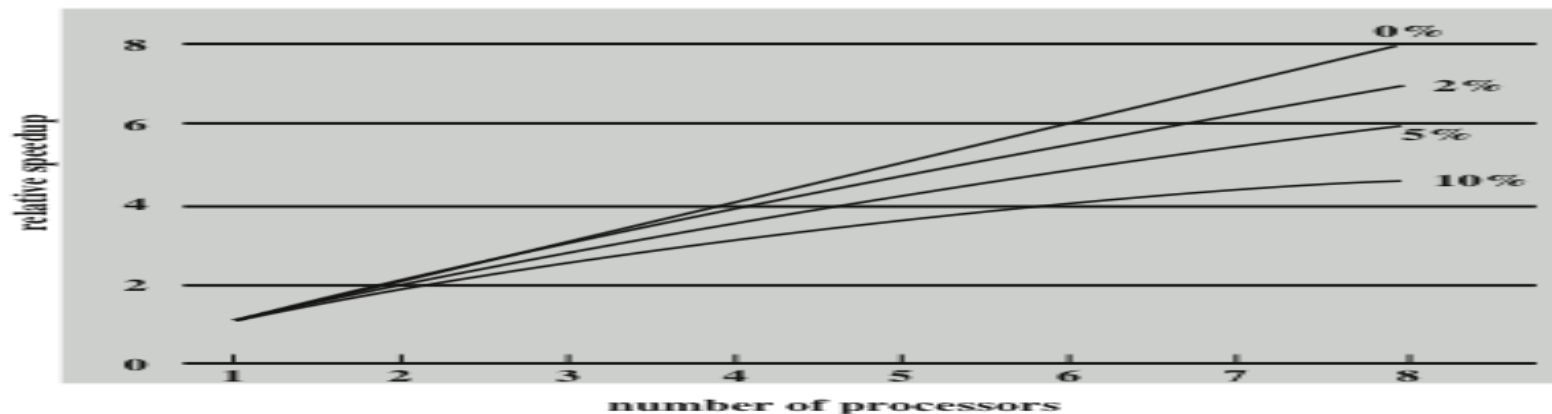


(c) Multicore

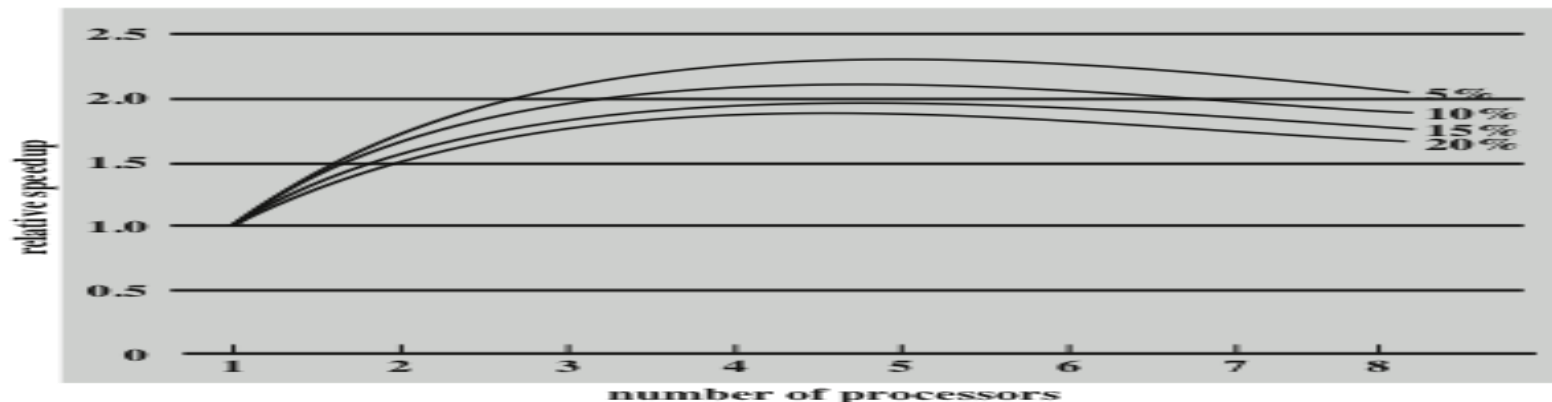
Pipelining: Individual instructions are executed through a pipeline of stages so that while one instruction is executing in one stage of the pipeline, another instruction is executing in another stage of the pipeline.

* **Superscalar:** Multiple pipelines are constructed by replicating execution resources. This enables parallel execution of instructions in parallel pipelines, so long as hazards are avoided.

* **Simultaneous multithreading (SMT):** Register banks are replicated so that multiple threads can share the use of pipeline resources.



(a) Speedup with 0% , 2% , 5% , and 10% sequential portions



(b) Speedup with overheads

Figure 18.5 Performance Effect of Multiple Cores

et said 10% of your code is sequential.!

so if U have 100 processors! the max speed of your code is still limited by % of sequential code in your program that is the limitation. thus to best utilize your processor is to run multiple program

Performance Effect of Multiple Cores

The law assumes a program in which a fraction $(1 - f)$ of the execution time involves code that is inherently serial and a fraction f that involves code that is infinitely parallelizable with no scheduling overhead. This law appears to make the prospect of a multicore organization attractive

But as Figure 18.5a shows, even a small amount of serial code has a noticeable impact. If only 10% of the code is inherently serial ($f = 0.9$), running the program on a multi-core system with 8 processors yields a performance gain of only a factor of 4.7. In addition, software typically incurs overhead as a result of communication and distribution of work among multiple processors and as a result of **cache coherence overhead**.

This results in a curve where performance peaks and then begins to degrade because of the increased burden of the overhead of using multiple processors (e.g., coordination and OS management). Figure 18.5b, is a representative example.

Effective Applications for Multicore Processors

■ **Multi-threaded native applications**

- Characterized by having a small number of highly threaded processes
- Lotus Domino, Siebel CRM (Customer Relationship Manager)

■ **Multi-process applications**

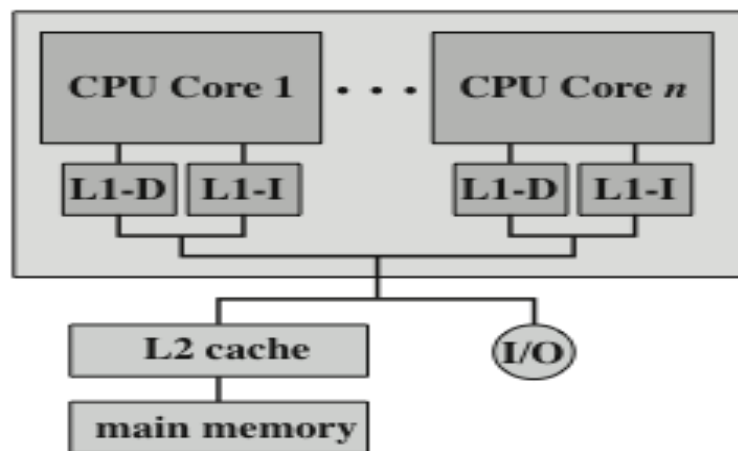
- Characterized by the presence of many single-threaded processes
- Oracle, SAP, PeopleSoft

■ **Java applications**

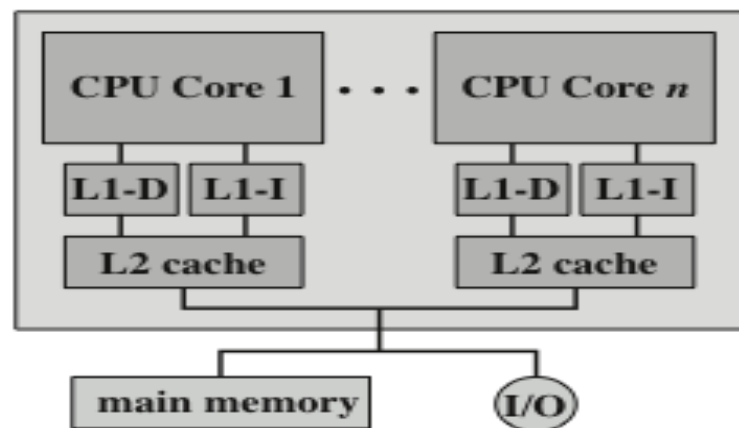
- Java Virtual Machine is a multi-threaded process that provides scheduling and memory management for Java applications
- Sun's Java Application Server, BEA's Weblogic, IBM Websphere, Tomcat

■ **Multi-instance applications**

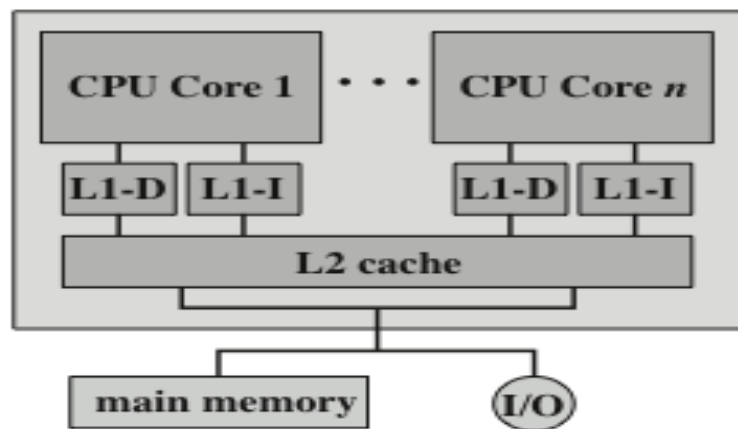
- One application running multiple times
- If multiple application instances require some degree of isolation, virtualization technology can be used to provide each of them with its own separate and secure environment



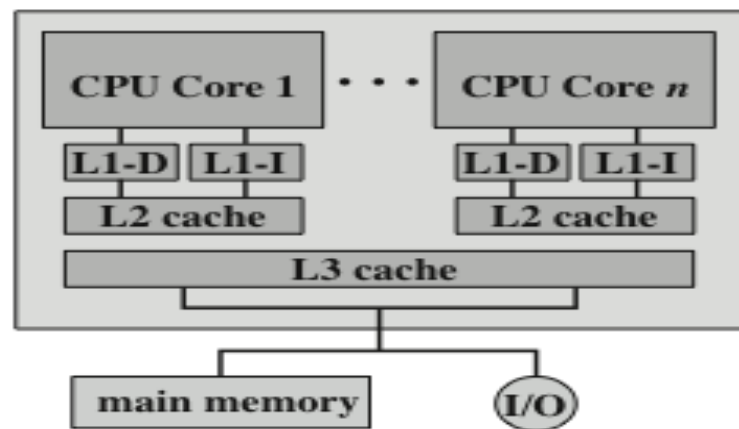
(a) Dedicated L1 cache



(b) Dedicated L2 cache



(c) Shared L2 cache



(d) Shared L3 cache

Figure 18.8 Multicore Organization Alternatives

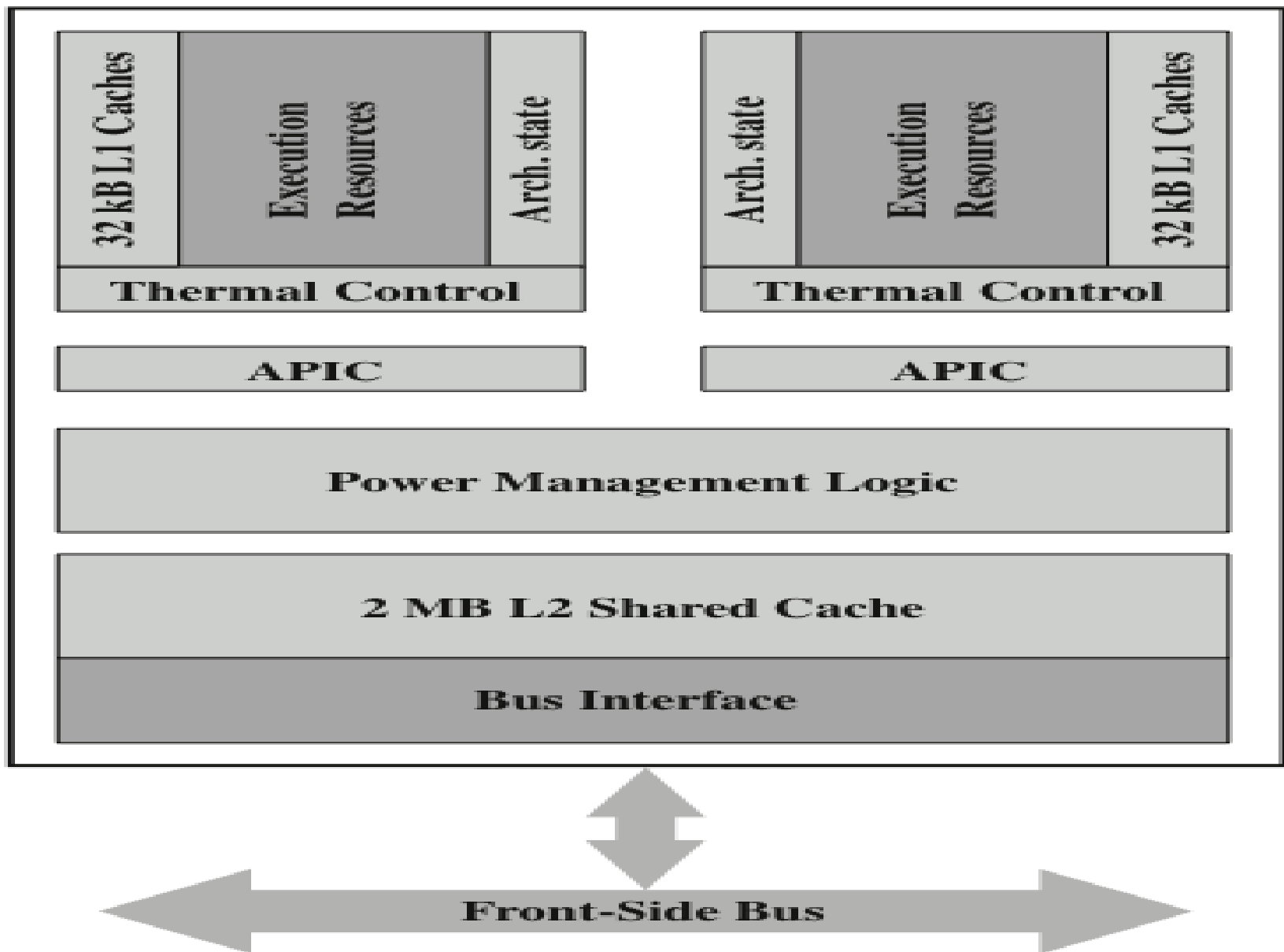


Figure 18.9 Intel Core Duo Block Diagram



Intel x86 Multicore Organization Core Duo

■ **Advanced Programmable Interrupt Controller (APIC)**

- Provides inter-processor interrupts which allow any process to interrupt any other processor or set of processors
- Accepts I/O interrupts and routes these to the appropriate core
- Includes a timer which can be set by the OS to generate an interrupt to the local core

■ **Power management logic**

- Responsible for reducing power consumption when possible, thus increasing battery life for mobile platforms
- Monitors thermal conditions and CPU activity and adjusts voltage levels and power consumption appropriately
- Includes an advanced power-gating capability that allows for an ultra fine grained logic control that turns on individual processor logic subsystems only if and when they are needed

Continued . . .



Intel x86 Multicore Organization Core Duo

■ 2MB shared L2 cache

- Cache logic allows for a dynamic allocation of cache space based on current core needs
- MESI support for L1 caches
- Extended to support multiple Core Duo in SMP
- L2 cache controller allows the system to distinguish between a situation in which data are shared by the two local cores, and a situation in which the data are shared by one or more caches on the die as well as by an agent on the external bus

■ Bus interface

- Connects to the external bus, known as the Front Side Bus, which connects to main memory, I/O controllers, and other processor chips

+ Summary

Lecture B - 07

- 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

Parallel Processing

- Multithreading and chip multiprocessors
 - Implicit and explicit multithreading
 - Approaches to explicit multithreading
 - Example systems
- Clusters
 - Cluster configurations
 - Operating system design issues
 - Cluster computer architecture
 - Clusters compared to SMP
- Nonuniform memory access
 - Motivation
 - Organization
 - NUMA Pros and cons
- Multi-core computing



- Slides adopted from:
 - Computer Organization and Architecture, 9th Edition
William Stallings
ISBN-10: 013293633X | ISBN-13: 9780132936330