K. S. Institute of Technology

Department of Computer Science and Engineering

Advanced Computer Architecture – 18CS733

Faculty Name: Dr. Vijayalaxmi Mekali

Associate Professor, Dept. of CSE

KSIT, Bangalore

Bus Systems

- System bus of a computer operates on contention basis.
- Several active devices such as processors may request use of the bus at the same time.
- Only one of them can be granted access to bus at a time
- The Effective bandwidth available to each processor is inversely proportional to the number of processors contending for the bus.
- For this reason, most bus-based commercial multiprocessors have been small in size.
- The simplicity and low cost of a bus system made it attractive in building small multiprocessors ranging from 4 to 16 processors.

Backplane Bus Specification

- A backplane bus interconnects processors, data storage and peripheral devices in a tightly coupled hardware.
- The system bus must be designed to allow communication between devices on the bus without disturbing the internal activities of all the devices attached to the bus.
- Timing protocols must be established to arbitrate among multiple requests. Operational rules must be set to ensure orderly data transfers on the bus.
- Signal lines on the backplane are often functionally grouped into several buses as shown in Fig 5.1. Various functional boards are plugged into slots on the backplane. Each slot is provided with one or more connectors for inserting the boards as demonstrated by the vertical arrows.

Data Transfer Bus (DTB)

- Data address and control lines form the data transfer bus (DTB) in VME bus.
- Address lines broadcast data and device address
  - Proportional to log of address space size
- Data lines proportional to memory word length
  - Control lines specify read/write, timing, and bus error conditions

Module-III

Bus, Cache and Shared Memory

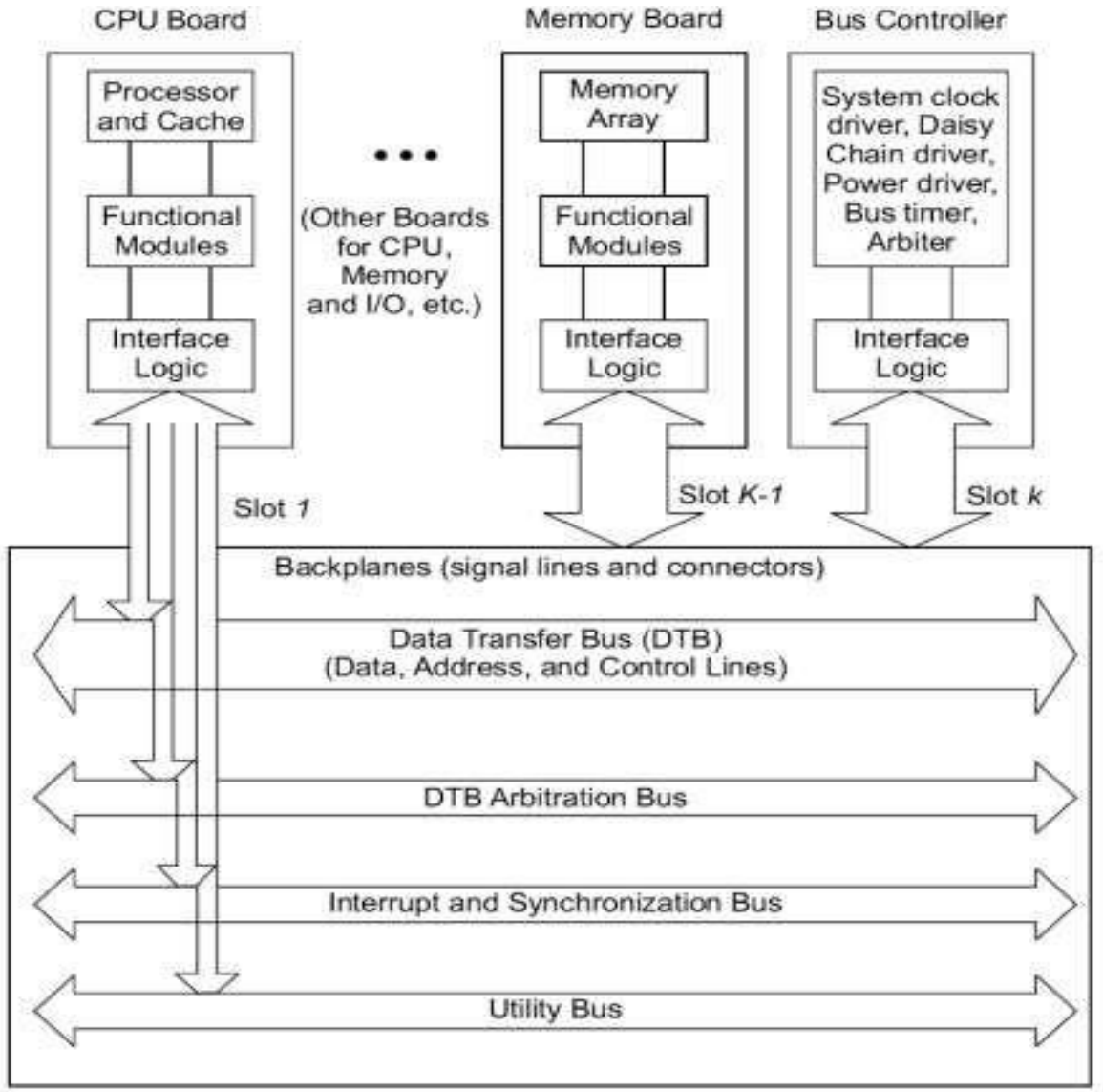


Fig. 5.1 Backplane buses, system interfaces, and slot connections to various functional boards in a multiprocessor system

Bus Arbitration and Control

- The process of assigning control of the DTB to a requester is called arbitration. Dedicated lines are reserved to coordinate the arbitration process among several requesters.
- The requester is called a master, and the receiving end is called a slave.
- Interrupt lines are used to handle interrupts, which are often prioritized. Dedicated lines may be used to synchronize parallel activities among the processor modules.
- Utility lines include signals that provide periodic timing (clocking) and coordinate the power-up and power-down sequences of the system.
- The backplane is made of signal lines and connectors.
- A special bus controller board is used to house the backplane control logic, such as the system clock driver, arbiter, bus timer, and power driver.

Functional Modules

- A functional module is a collection of electronic circuitry that resides on one functional board (Fig. 5.1) and works to achieve special bus control functions.
- Special functional modules are introduced below:
- Arbiter is a functional module that accepts bus requests from the requester module and grants control of the DTB to one requester at a time.
- Bus timer measures the time each data transfer takes on the DTB and terminates the DTB cycle if a transfer takes too long.
- Interrupter module generates an interrupt request and provides status/ID information when an interrupt handler module requests it.
- Location monitor is a functional module that monitors data transfers over the DTB. A power monitor watches the status of the power source and signals when power becomes unstable.
- System clock driver is a module that provides a clock timing signal on the utility bus. In addition, board interface logic is needed to match the signal line impedance, the propagation time, and termination values between the backplane and the plug-in boards.

➢ **Physical Limitations**

- **Due to electrical, mechanical, and packaging limitations, only a limited number of boards can be plugged into a single backplane.**
- **Multiple backplane buses can be mounted on the same backplane chassis.**
- **The bus system is difficult to scale, mainly limited by packaging constraints.**

➢ **Addressing and Timing Protocols**

- **Two types of printed circuit boards connected to a bus: active and passive**
- **Active devices like processors can act as bus masters or as slaves at different times.**
- **Passive devices like memories can act only as slaves.**
- **The master can initiate a bus cycle**
  - **Only one can be in control at a time**
- **The slaves respond to requests by a master**
  - **Multiple slaves can respond**

➢Bus Addressing

- The backplane bus is driven by a digital clock with a fixed cycle time: bus cycle
- Backplane has limited physical size, so will not skew information
- Factors affecting bus delay: –
  - Source's line drivers, destination's receivers, slot capacitance, line length, and bus loading effects
- Design should minimize overhead time, so most bus cycles used for useful operations
- Identify each board with a slot number
- When slot number matches contents of high-order address lines, the board is selected as a slave (slot addressing)

➢Broadcall and Broadcast

- Most bus transactions have one slave/master
- Broadcall: read operation where multiple slaves place data on bus
  - detects multiple interrupt sources

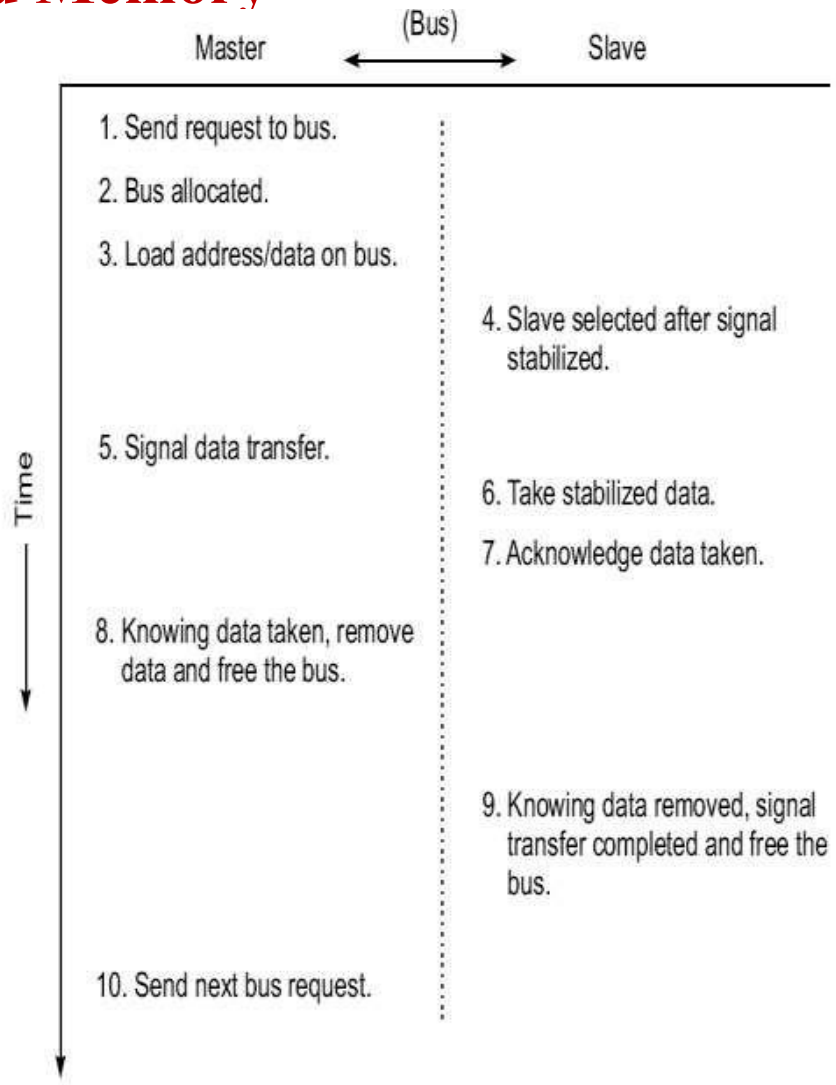


Fig. 5.2 Typical time sequence for information transfer between a master and a slave over a system bus

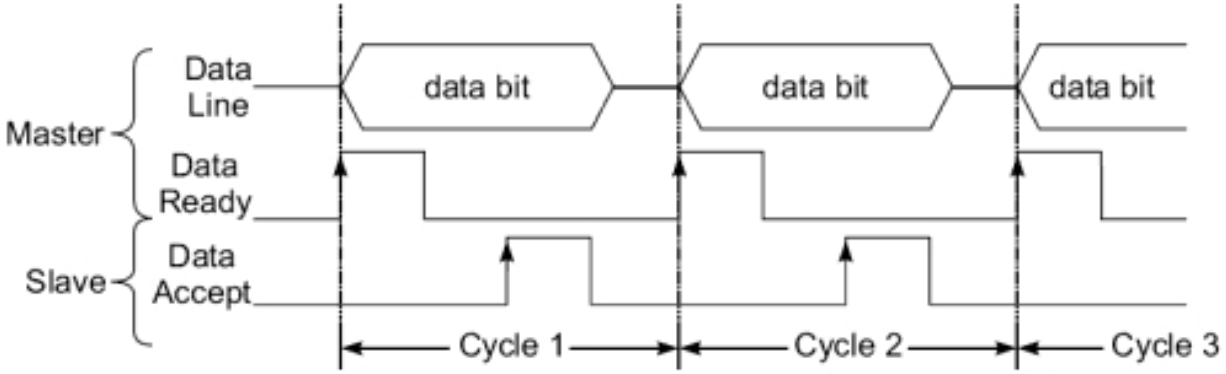
Broadcast: write operation involving multiple slaves

– Implements multicache coherence on the bus

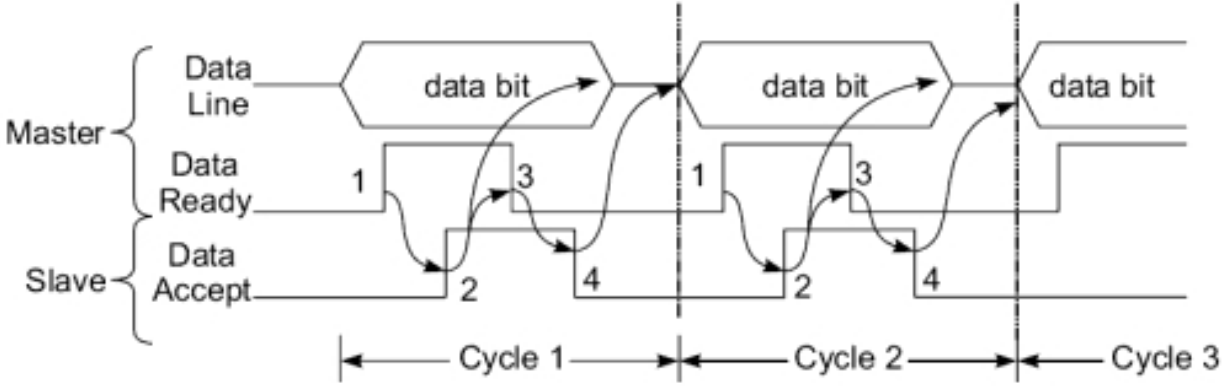
- Timing protocols are needed to synchronize master and slave operations.
- Figure 5.2 shows a typical timing sequence when information is transferred over a bus from a source to a destination.
- Most bus timing protocols implement such a sequence.

Synchronous Timing

- All bus transaction steps take place at fixed clock edges as shown in Fig. 5.3a.
- The clock signals are broadcast to all potential masters and slaves.
- Clock cycle time determined by slowest device on bus
- Once the data becomes stabilized on the data lines, the master uses Data-ready pulse to initiate the transfer
- The Slave uses Data-accept pulse to signal completion of the information transfer.
- Simple, less circuitry, suitable for devices with relatively the same speed.



(a) Synchronous bus timing with fixed-length clock signals for all devices



(b) Asynchronous bus timing using a four-edge handshaking (interlocking with variable length signals for different speed devices.

Fig. 5.3 Synchronous versus asynchronous bus timing protocols

Asynchronous Timing

- **Based on handshaking or interlocking mechanism as shown in Fig. 5.3b.**
- **No fixed clock cycle is needed.**
- **The rising edge (1) of the data-ready signal from the master triggers the rising (2) of the data- accept signal from the slave.**
- **The second signal triggers the falling (3) of the data-ready clock and removal of data from the bus.**
- **The third signal triggers the trailing edge (4) of the data accept clock.**
- **This four-edge handshaking (interlocking) process is repeated until all the data is transferred.**

Advantages:

- **Provides freedom of variable length clock signals for different speed devices**
- **No response time restrictions**
- **More flexible**

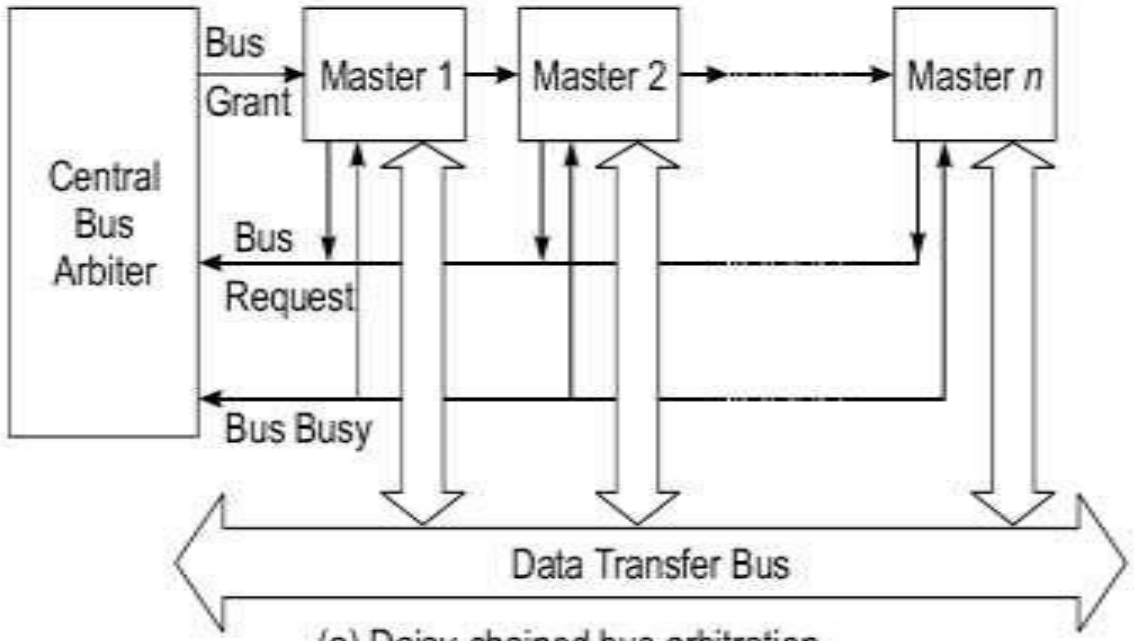
- **Disadvantage: More complex and costly**

Arbitration, Transaction and Interrupt

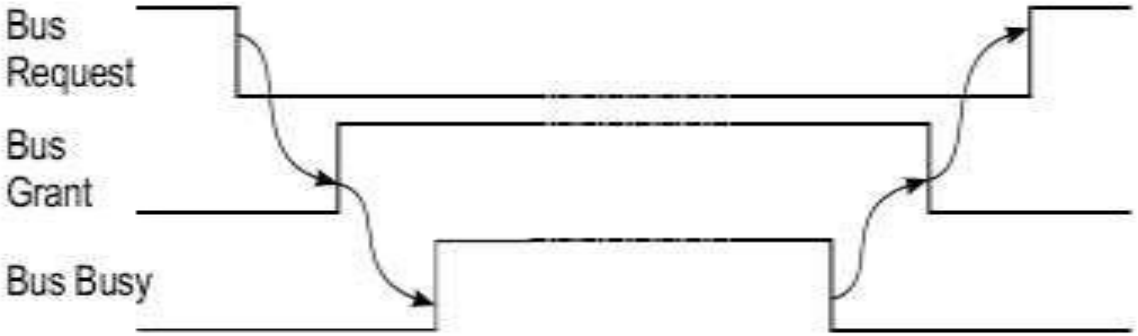
- Arbitration
- Process of selecting next bus master
- Bus tenure is duration of master's control
- It restricts the tenure of the bus to one master at a time.
- Competing requests must be arbitrated on a fairness or priority basis
- Arbitration competition and bus transactions take place concurrently on a parallel bus over separate lines
- Three Arbitration process
  1. Central Arbitration
  2. Distributed Arbitration
  3. Independent Requests and Grants

1. Central Arbitration

- Uses a central arbiter as shown in Fig 5.4a
- Potential masters are daisy chained in a cascade
- A special signal line propagates bus-grant from first master (at slot 1) to the last master (at slot n).
- All requests share the same bus-request line
- The bus-request signals the rise of the bus-grant level, which in turn raises the bus-busy level as shown in Fig. 5.4b.



(a) Daisy-chained bus arbitration



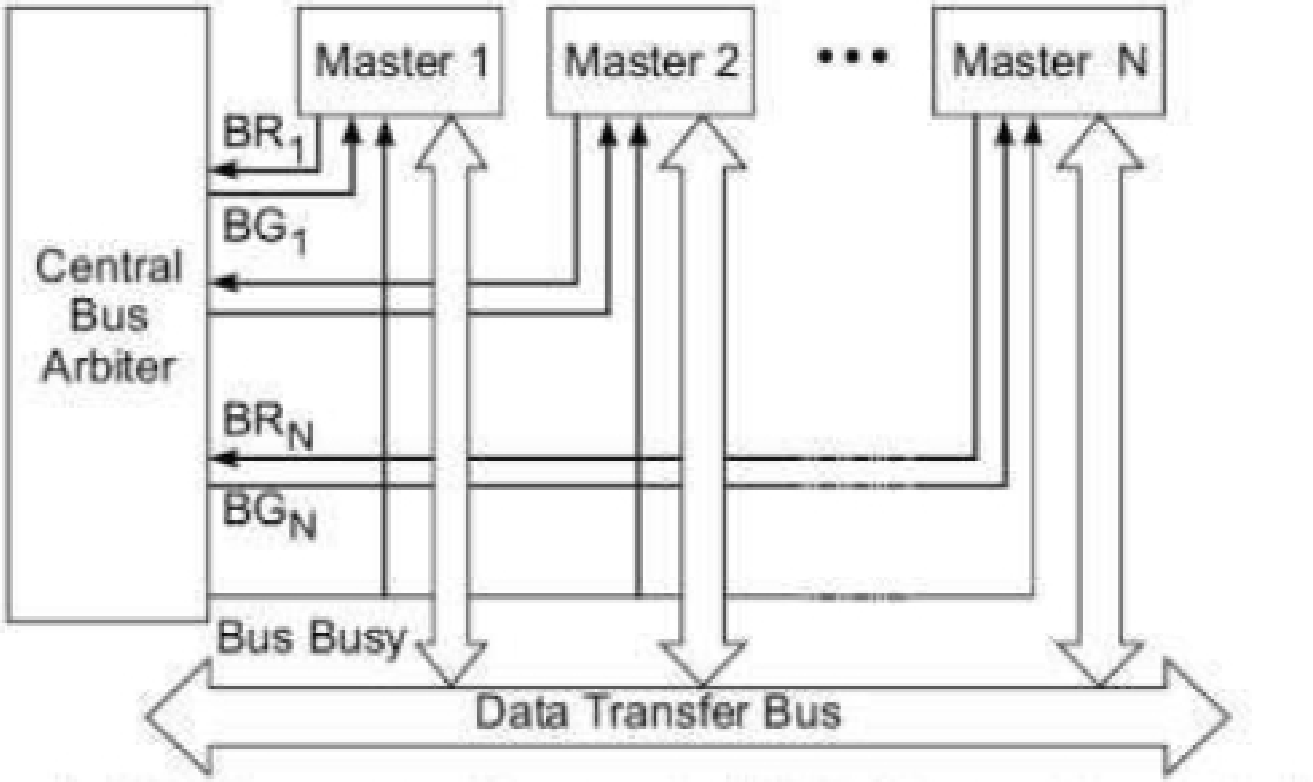
(b) Bus transaction timing

- Simple scheme
- Easy to add devices
- Fixed-priority sequence – not fair
- Propagation of bus-grant signal is slow
- Not fault tolerant

2. Independent Requests and Grants

- Provide independent bus-request and grant signals for each master as shown in Fig5.5a.
- No daisy chaining is used in this scheme.
- Require a central arbiter, but can use a priority or fairness based policy
- More flexible and faster than a daisy-chained policy
- Larger number of lines – costly

Independent Requests and Grants



Legends: BR_i (Bus request from master i) BG_i (Bus grant to master i)

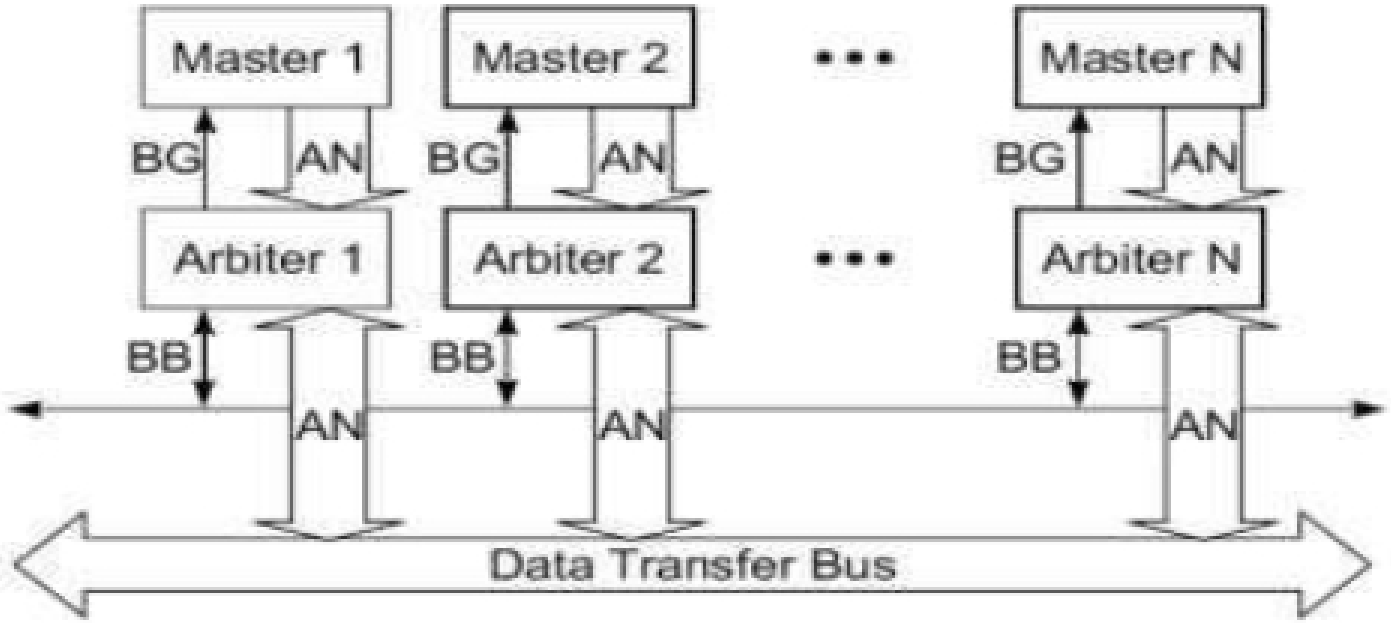
(a) Independent requests with a central arbiter

Fig. 5.5 Two bus arbitration schemes using independent requests and distributed arbiters, respectively

3. Distributed Arbitration

- Each master has its own arbiter and unique arbitration number as shown in Fig. 5.5b.
- Uses arbitration number to resolve arbitration competition
- When two or more devices compete for the bus, the winner is the one whose arbitration number is the largest determined by Parallel Contention Arbitration..
- All potential masters can send their arbitration number to shared-bus request/grant (SBRG) lines and compare its own number with SBRG number.
- If the SBRG number is greater, the requester is dismissed. At the end, the winner's arbitration number remains on the arbitration bus. After the current bus transaction is completed, the winner seizes control of the bus.
- Priority based scheme

3. Distributed Arbitration



Legends: BG (Bus grant) BB (Bus busy) AN (Arbitration number)

(b) Using distributed arbiters

Fig. 5.5 Two bus arbitration schemes using independent requests and distributed arbiters, respectively

Transfer Modes

- **Address-only transfer**: no data
- **Compelled-data transfer:** Address transfer followed by a block of one or more data transfers to one or more contiguous address.
- **Packet-data transfer:** Address transfer followed by a fixed-length block of data transfers from set of continuous address.
- **Connected:** carry out master's request and a slave's response in a single bus transaction
- **Split:** splits request and response into separate transactions
  - Allow devices with long latency or access time to use bus resources more efficiently
  - May require two or more connected bus transactions

Transfer Modes

- **Address-only transfer**: no data
- **Compelled-data transfer:** Address transfer followed by a block of one or more data transfers to one or more contiguous address.
- **Packet-data transfer:** Address transfer followed by a fixed-length block of data transfers from set of continuous address.
- **Connected:** carry out master's request and a slave's response in a single bus transaction
- **Split:** splits request and response into separate transactions
  - Allow devices with long latency or access time to use bus resources more efficiently
  - May require two or more connected bus transactions

Interrupt Mechanisms

- **Interrupt:** is a request from I/O or other devices to a processor for service or attention
- A priority interrupt bus is used to pass the interrupt signals
- Interrupter must provide status and identification information
- Have an interrupt handler for each request line
- Interrupts can be handled by message passing on data lines on a time-sharing basis.
  - Save lines, but use cycles
  - Use of time-shared data bus lines is a virtual interrupt

IEEE and other Standards

- **Open bus standard Futurebus+ to support:**

– 64 bit address space
– Throughput required by multi-RISC or future generations of multiprocessor architectures
• Expandable or scalable
• Independent of particular architectures and processor technologies

Standard Requirements

The major objectives of the Futurebus+ standards committee were to create a bus standard that would provide a significant step forward in improving the facilities and performance available to the designers of multiprocessor systems.

Below are the design requirements set by the IEEE 896.1-1991 Standards Committee to provide a stable platform on which several generations of computer systems could be based:

• Independence for an open standard
• Asynchronous timing protocol
• Optional packet protocol
• Distributed arbitration protocols

Standard Requirements cond

- Support of high reliability and fault tolerant applications
- Ability to lock modules without deadlock or livelock
- Circuit-switched and split transaction protocols
- Support of real-time mission critical computations w/multiple priority levels
- 32 or 64 bit addressing
- Direct support of snoopy cache-based multiprocessors.
- Compatible message passing protocols

Direct Mapping Cache and Associative Cache

The transfer of information from main memory to cache memory is conducted in units of cache blocks or cache lines.

- Four block placement schemes are presented below. Each placement scheme has its own merits and demerits.
- The ultimate performance depends upon cache access patterns, organization, and management policy
- Blocks in caches are called block frames, and blocks in main memory are called blocks Bi ($i \leq m$), Bj ($i \leq n$), $n >> m$, $n=2^s$, $m=2^r$
- Each block has b words $b=2^w$, for cache total of $mb=2^{r+w}$ words, main memory of $nb= 2^{s+w}$ words

Direct Mapping Cache

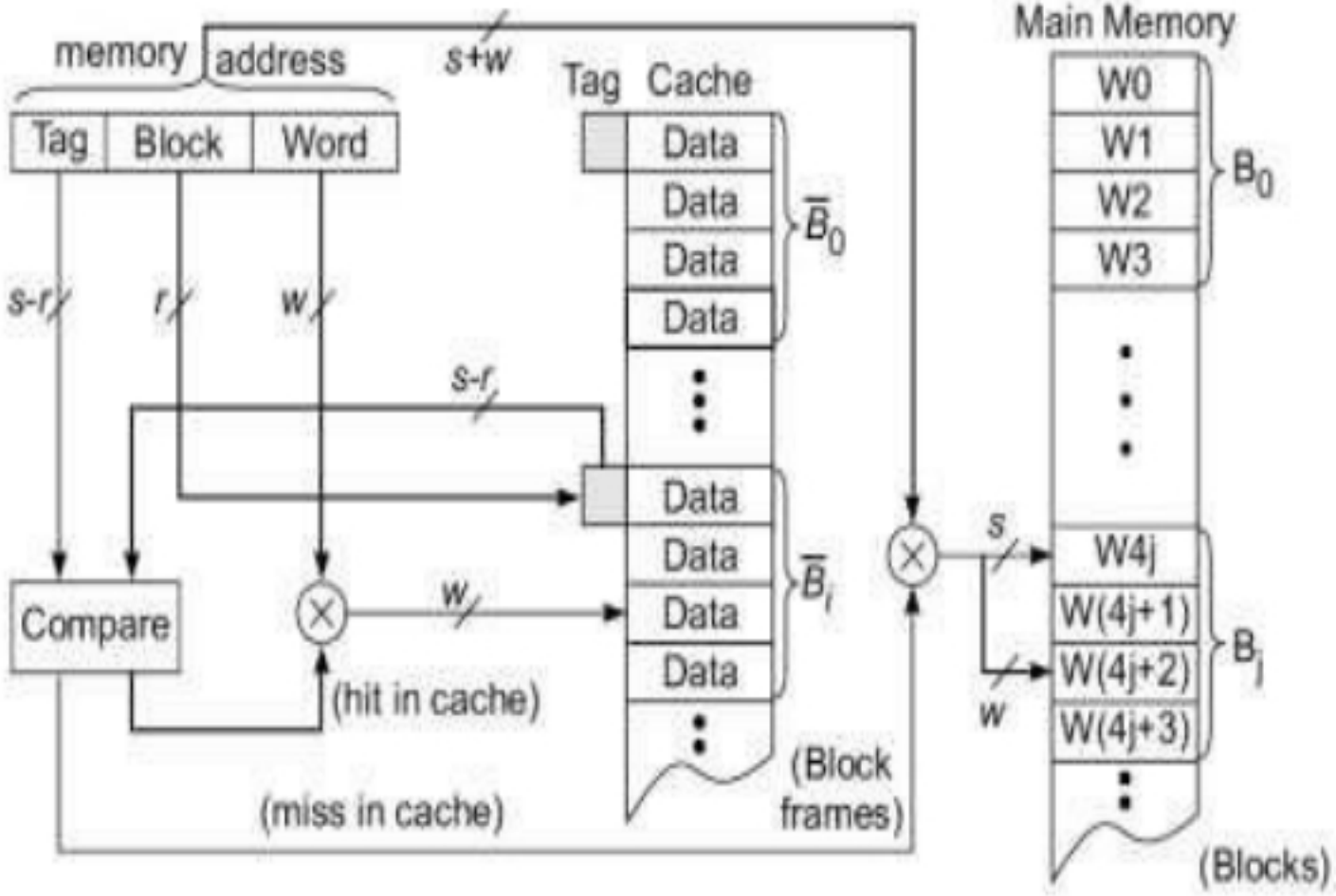
- Direct mapping of $n/m = 2^{s-r}$ memory blocks to one block frame in the cache
- Placement is by using modulo-m function. Block Bj is mapped to block frame Bi
- There is a unique block frame Bi that each Bj can load into.
- There is no way to implement a block replacement policy.
- This Direct mapping is very rigid but is the simplest cache organization to implement.

The memory address is divided into 3 fields:

Bj → Bi if i=j mod m

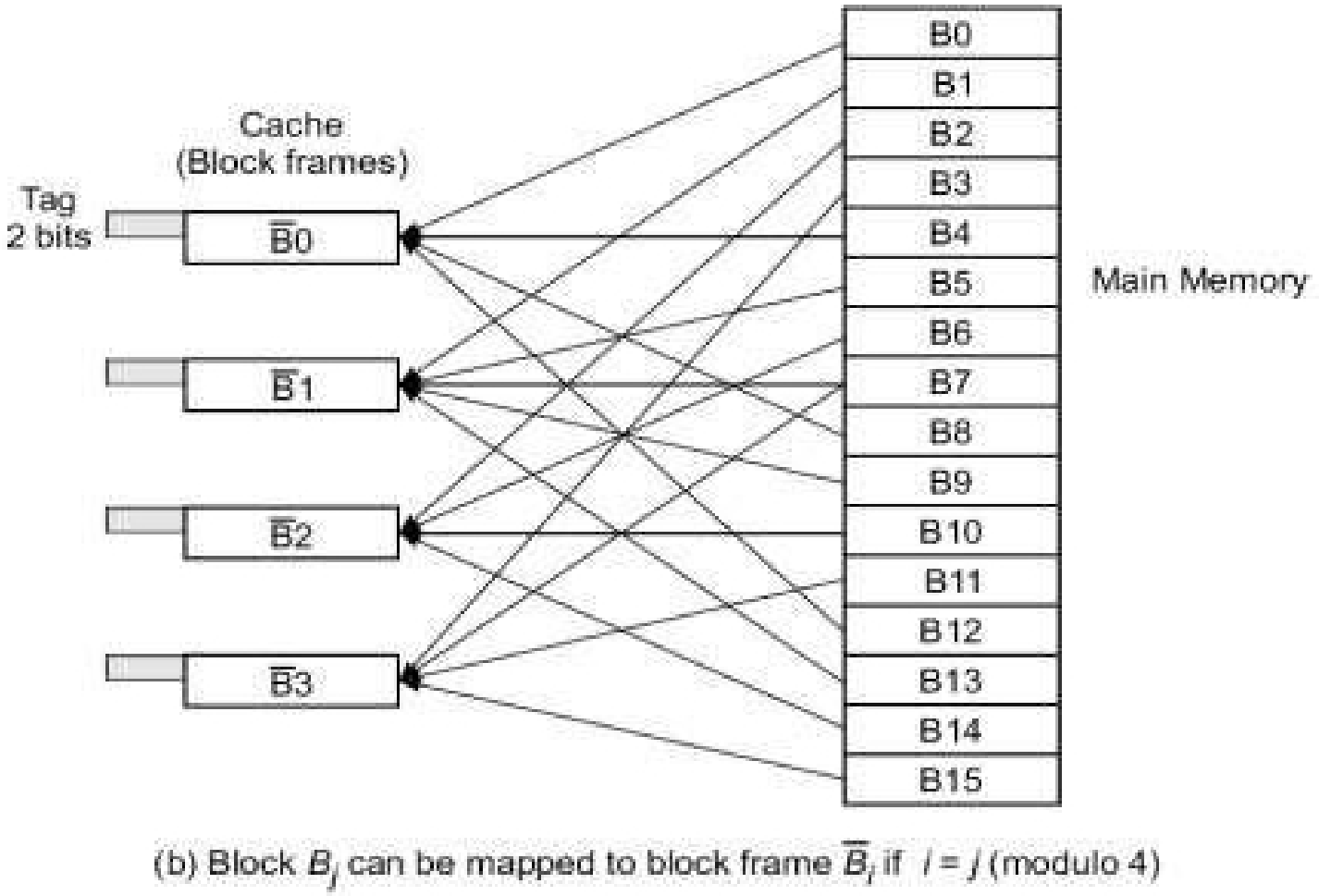
– The lower w bits specify the word offset within each block.
– The upper s bits specify the block address in main memory
– The leftmost (s-r) bits specify the tag to be matched

Direct Mapping Cache



(a) The cache/memory addressing

Direct Mapping Cache



(b) Block B_j can be mapped to block frame $\overline{B}_i$ if $i = j$ (modulo 4)

Fig. 5.9 Direct-mapping cache organization and a mapping example

Direct Mapping Cache

The block field (r bits) is used to implement the (modulo-m) placement, where $m=2^r$

Once the block Bi is uniquely identified by this field, the tag associated with the addressed block is compared with the tag in the memory address.

Advantages

- Simple hardware
- No associative search
- No page replacement policy
- Lower cost
- Higher speed

Disadvantages

- Rigid mapping
- Poorer hit ratio
- Prohibits parallel virtual address translation
- Use larger cache size with more block frames to avoid contention

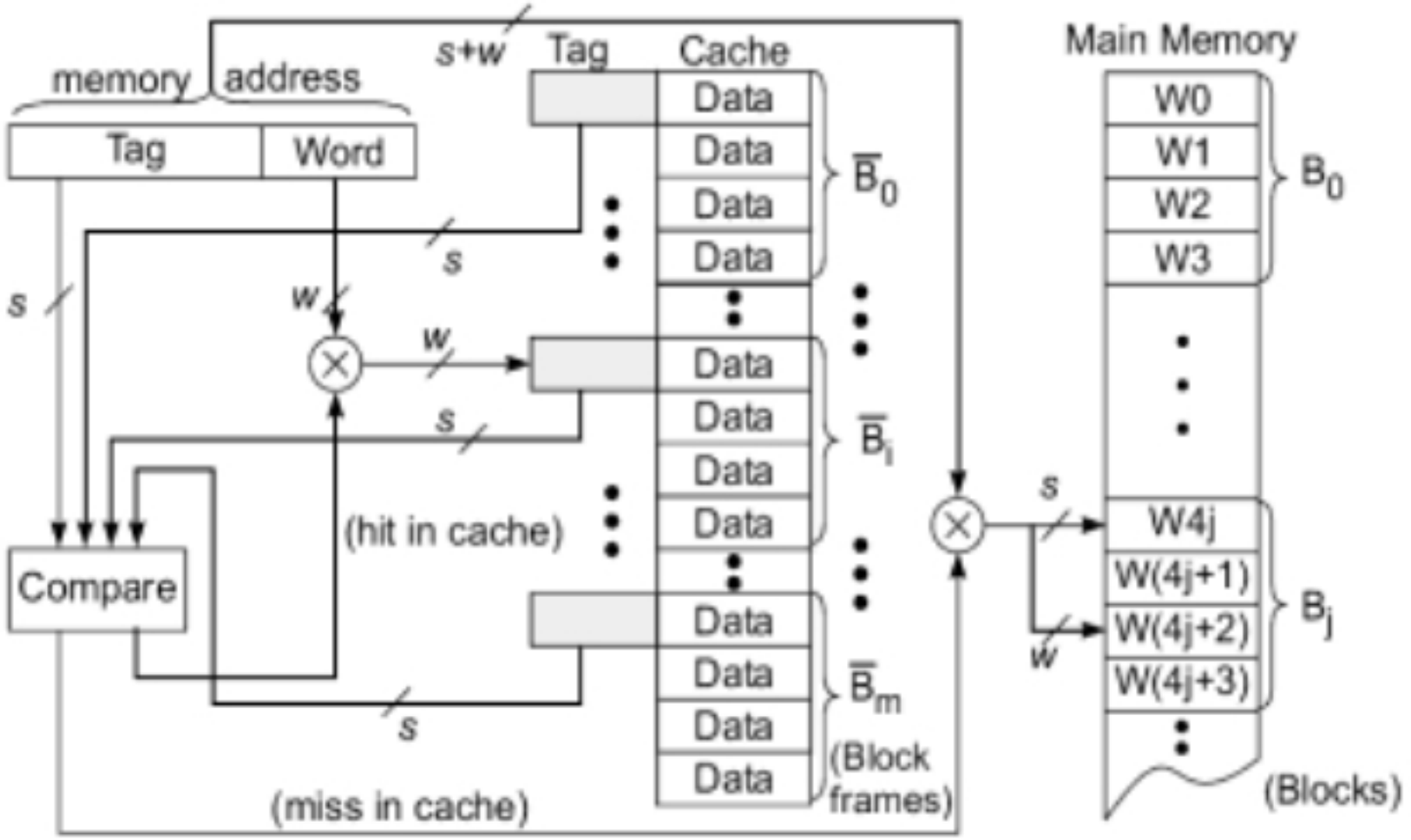
Fully Associative Cache

Each block in main memory can be placed in any of the available block frames as shown in Fig. 5.10a.

- Because of this flexibility, an s-bit tag needed in each cache block.
- As s > r, this represents a significant increase in tag length.
- The name fully associative cache is derived from the fact that an m-way associative search requires tag to be compared with all block tags in the cache. This scheme offers the greatest flexibility in implementing block replacement policies for a higher hit ratio.
- An m-way comparison of all tags is very time consuming if the tags are compared sequentially using RAMs. Thus an associative memory is needed to achieve a parallel comparison with all tags simultaneously.
- This demands higher implementation cost for the cache. Therefore, a Fully Associative Cache has been implemented only in moderate size.

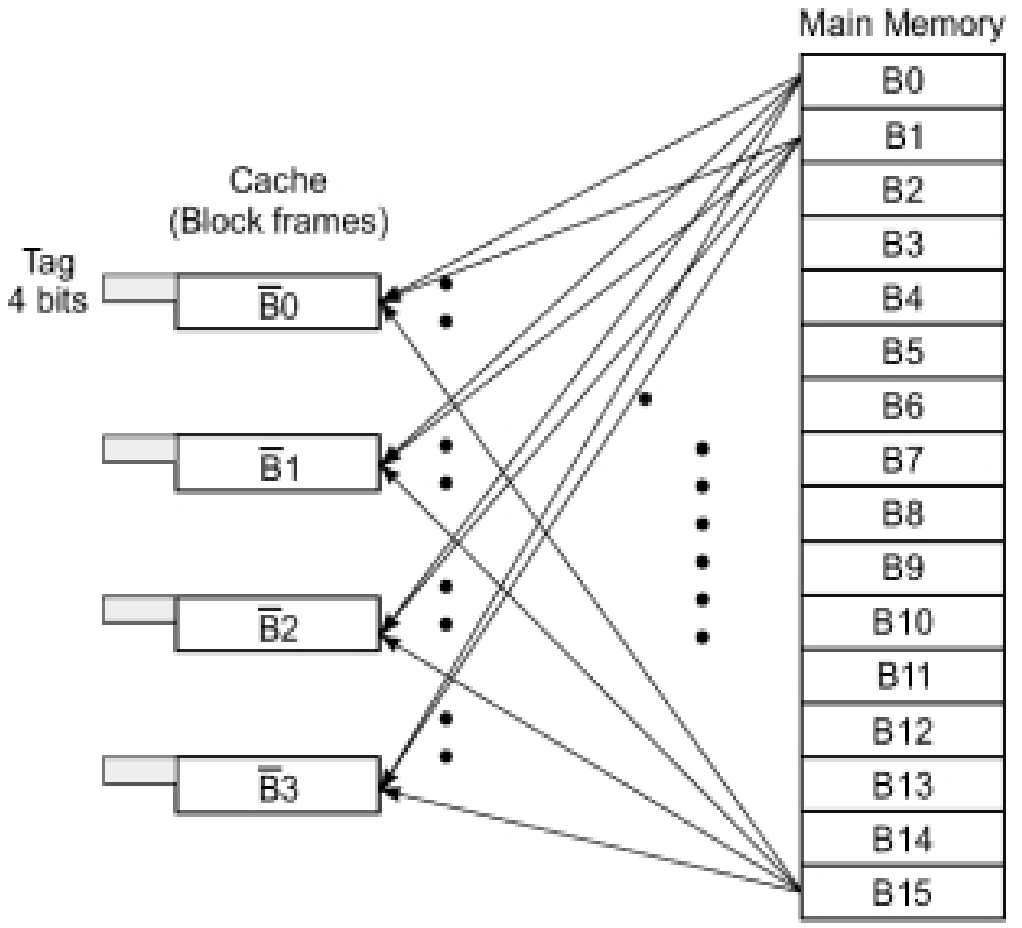
Fig. 5.10b shows a four-way mapping example using a fully associative search. The tag is 4-bits long because 16 possible cache blocks can be destined for the same block frame.

Fully Associative Cache



(a) Associative search with all block tags

Fully Associative Cache



(b) Every block is mapped to any of the four block frames identified by the tag

Fig. 5.10 Fully associative cache organization and a mapping example

Fully Associative Cache

Advantages:

- Offers most flexibility in mapping cache blocks
- Higher hit ratio
- Allows better block replacement policy with reduced block contention
- Higher hardware cost
- Only moderate size cache
- Expensive search process

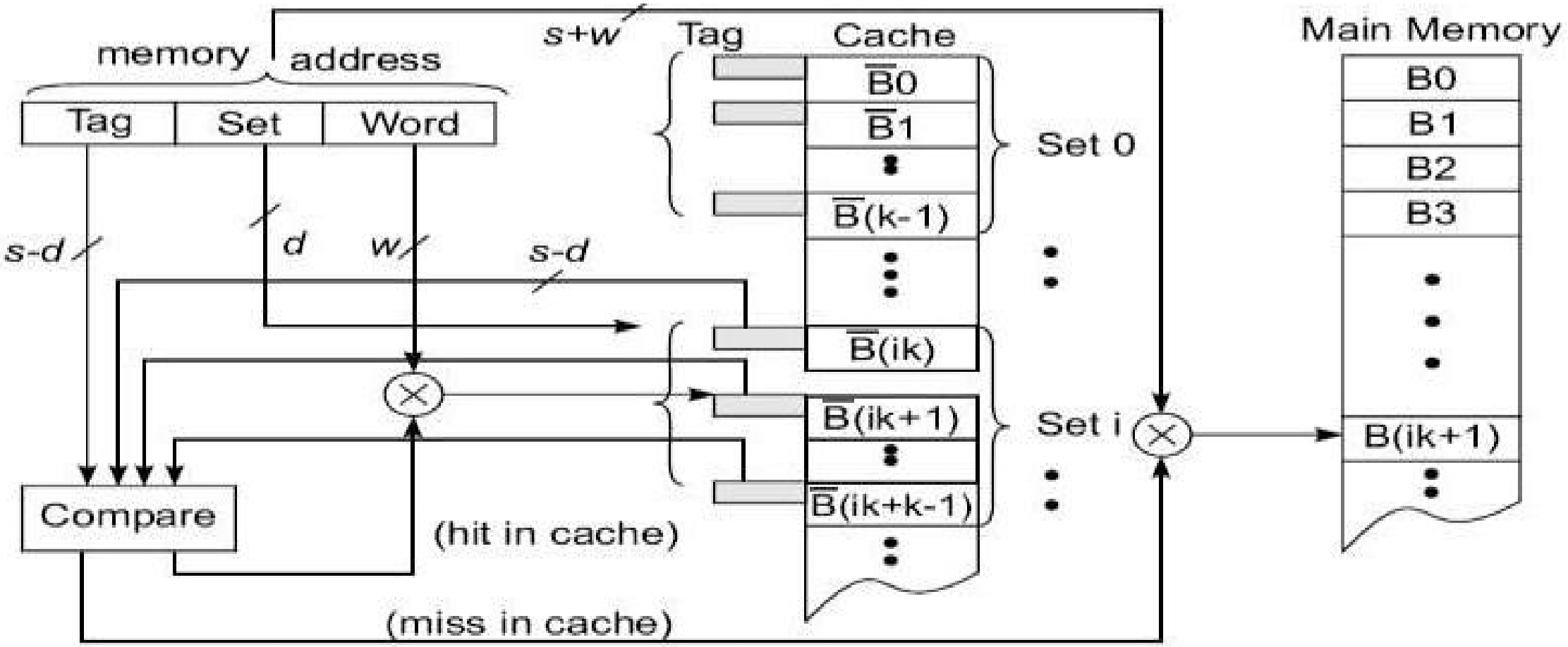
Disadvantages:

- Higher hardware cost
- Only moderate size cache
- Expensive search process

Set Associative Caches

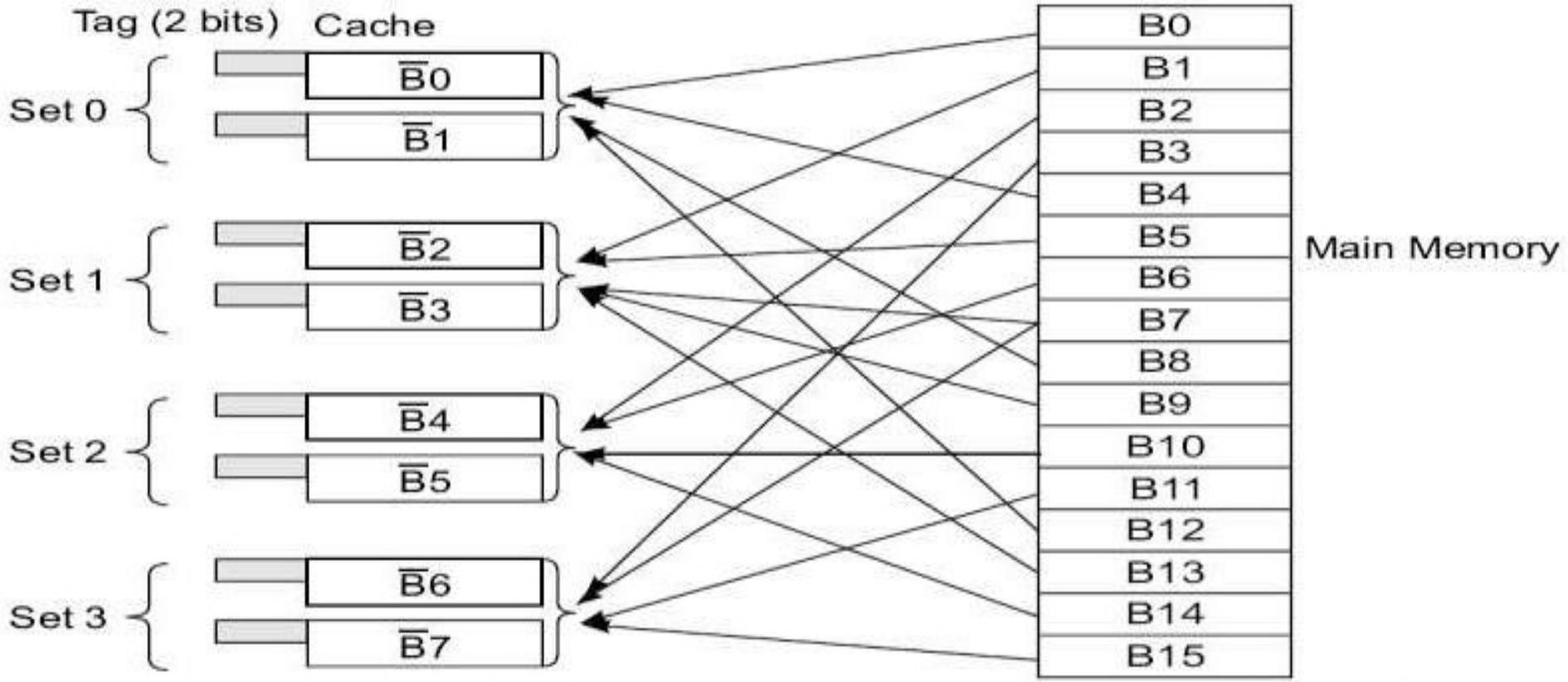
- In a k-way associative cache, the m cache block frames are divided into v=m/k sets, with k blocks per set
- Each set is identified by a d-bit set number, where $2^d = v$.
- The cache block tags are now reduced to s-d bits.
- In practice, the set size k, or associativity, is chosen as 2, 4, 8, 16 or 64 depending on a tradeoff among block size w, cache size m and other performance/cost factors.

Set Associative Caches



(a) A *k*-way associative search within each set of *k* each blocks

Set Associative Caches



(b) Mapping cache blocks in a two-way associative cache wit four sets

Fig. 5.11 Set-associative cache organization and a two-way associative mapping example

Set Associative Caches

- Compare the tag with the k tags within the identified set as shown in Fig 5.11a.
- Since k is rather small in practice, the k-way associative search is much more economical than the full associativity.
- In general, a block Bj can be mapped into any one of the available frames Bf in a set Si defined below.

$$B_j \rightarrow B_f \in S_i \text{ if } j(\text{mod } v) = i$$

The matched tag identifies the current block which resides in the frame.

Sector Mapping Cache

- Partition both the cache and main memory into fixed size sectors.
- Then use fully associative search ie., each sector can be placed in any of the available sector frames.
- The memory requests are destined for blocks, not for sectors.
- This can be filtered out by comparing the sector tag in the memory address with all sector tags using a fully associative search.
- If a matched sector frame is found (a cache hit), the block field is used to locate the desired block within the sector frame.
- If a cache miss occurs, the missing block is fetched from the main memory and brought into a congruent block frame in available sector.
- That is the i^{th} block in a sector must be placed into the ith block frame in a destined sector frame.
- Attach a valid bit to each block frame to indicate whether the block is valid or invalid.
- When the contents of the block frame are replaced from a new sector, the remaining block frames in the same sector are marked invalid. Only the block frames from the most recently referenced sector are marked valid for reference.

Sector Mapping Cache

-

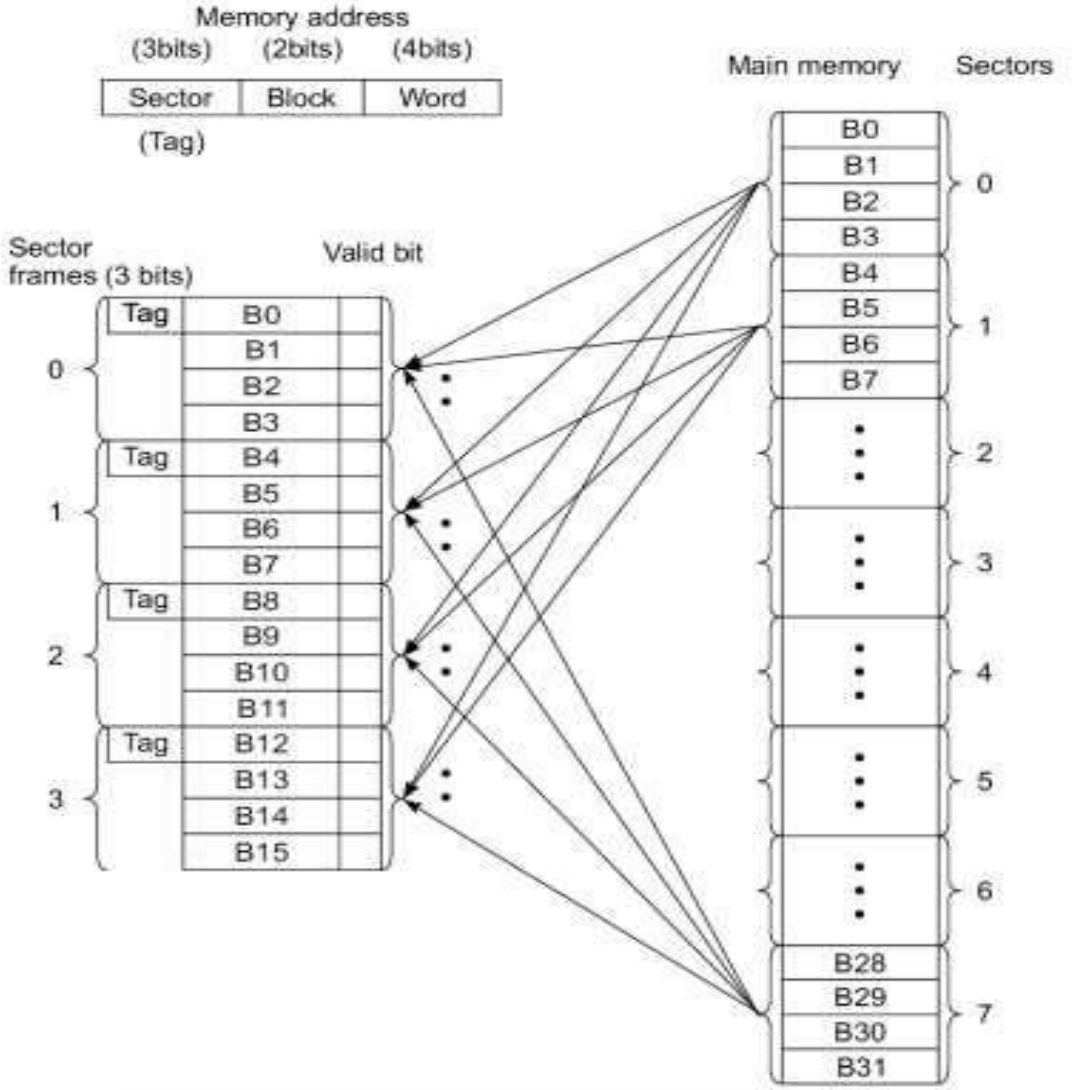


Fig. 5.12 A four-way sector mapping cache organization

Sector Mapping Cache advantages

- Flexible to implement various bkock replacement algorithms
- Economical to perform a fully associative search a limited number of sector tags.
- Sector partitioning offers more freedom in grouping cache lines at both ends of the mapping.

Cache Performance Issues

As far the performance of cache is considered the trade off exist among the cache size, set number, block size and memory speed. Important aspect in cache designing with regard to performance are :

1. Cycle counts

- This refers to the number of basic machine cycles needed for cache access, update and coherence control.
- Cache speed is affected by underlying static or dynamic RAM technology, the cache organization and the cache hit ratios.
- The write through or write back policy also affect the cycle count.
- Cache size, block size, set number, and associativity affect count
- The cycle count is directly related to the hit ratio, which decreases almost linearly with increasing values of above cache parameters.

Cache Performance Issues

2. Hit ratio

- The hit ratio is number of hits divided by total number of CPU references to memory (hits plus misses).
- Hit ratio is affected by cache size and block size
- Increases w.r.t. increasing cache size
- Limited cache size, initial loading, and changes in locality prevent 100% hit ratio

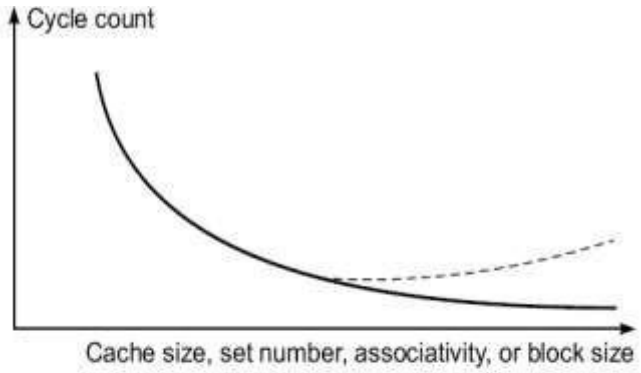
3. Effect of Block Size:

- With a fixed cache size, cache performance is sensitive to the block size.
- As block size increases, hit ratio improves due to spatial locality
- Peaks at optimum block size, then decreases
- If too large, many words in cache not used

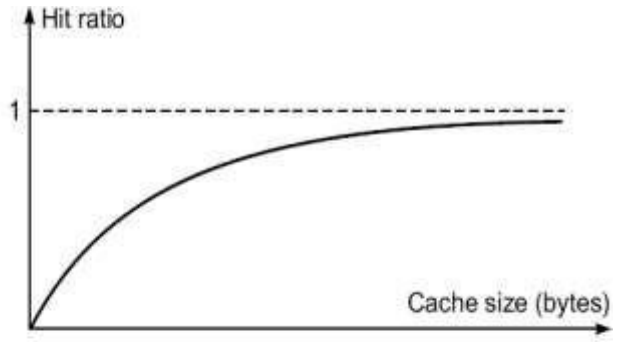
4. Effect of set number

- In a set associative cache, the effects of set number are obvious.
- For a fixed cache capacity, the hit ratio may decrease as the number of sets increases.
- As the set number increases from 32 to 64, 128 and 256, the decrease in the hit ratio is rather small.
- When the set number increases to 512 and beyond, the hit ratio decreases faster.

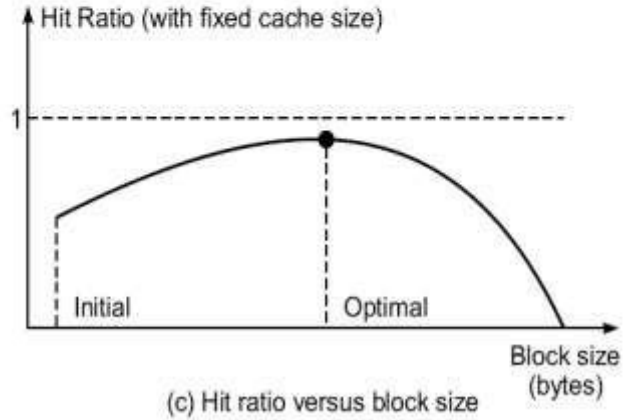
Cache Performance Issues



(a) The total cycle count for cache access (Courtesy of S. A. Przybylski; reprinted with permission from *Cache and Memory Hierarchy Design*, Morgan Kaufmann Publishers, 1990)



(b) Hit ratio versus cache size



(c) Hit ratio versus block size

Fig. 5.13 Cache performance versus design parameters used

Shared Memory Organizations

- **Memory interleaving provides a higher bandwidth for pipelined access of continuous memory locations.**
- **Methods for allocating and deallocating main memory to multiple user programs are considered for optimizing memory utilization.**

Interleaved Memory Organization

• In order to close up the speed gap between the CPU/cache and main memory built with RAM modules, an interleaving technique is presented below which allows pipelined access of the parallel memory modules.

• The memory design goal is to broaden the effective memory bandwidth so that more memory words can be accessed per unit time.

• The ultimate purpose is to match the memory bandwidth with the bus bandwidth and with the processor bandwidth.

Memory bandwidth

Memory bandwidth is the rate at which data can be read from or stored into a semiconductor memory by a processor. Memory bandwidth is usually expressed in units of bytes/second

Interleaved Memory Organization cont..

Memory Interleaving

- The main memory is built with multiple modules.
- These memory modules are connected to a system bus or a switching network to which other resources such as processors or I/O devices are also connected.
- Once presented with a memory address, each memory module returns with one word per cycle.
- It is possible to present different addresses to different memory modules so that parallel access of multiple words can be done simultaneously or in a pipelined fashion.

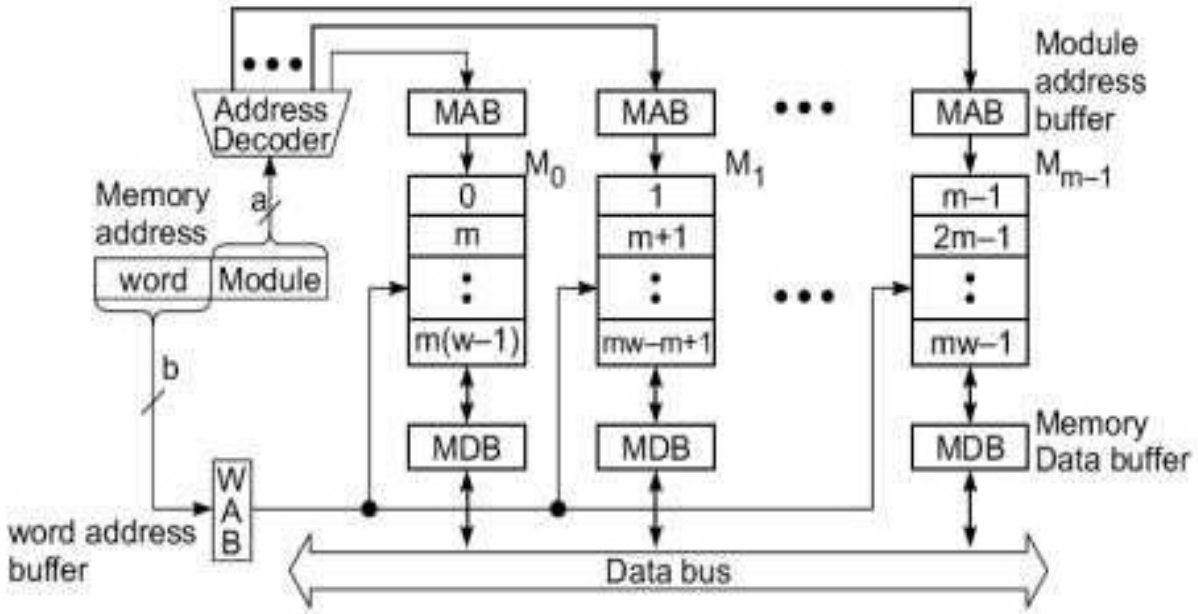
Both parallel access and pipelined access are form of parallelism practiced in a memory organization.

- Consider a main memory formed with $m = 2^a$ memory modules, each containing $w = 2^b$ words of memory cells. The total memory capacity is $m.w = 2^{a+b}$ words.
- These memory words are assigned linear addresses. Different ways of assigning linear addresses result in different memory organizations.
- Besides random access, the main memory is often block-accessed at consecutive addresses.
- Block access is needed for fetching a sequence of instructions or for accessing a linearly ordered data structures.
- Each block access may corresponds to the size of a cache block.

Memory Interleaving

- Low-order interleaving
- High-order interleaving

Figure 5.15 shows Lower order memory interleaving.



(a) Low-order *m*-way interleaving (the C-access memory scheme)

Memory Interleaving

Low-order interleaving

- Low-order interleaving spreads contiguous memory locations across the m modules horizontally (Fig. 5.15a).
- This implies that the low-order a bits of the memory address are used to identify the memory module.
- The high-order b bits are the word addresses (displacement) within each module.
- Note that the same word address is applied to all memory modules simultaneously. A module address decoder is used to distribute module addresses.

Memory Interleaving
High-order interleaving

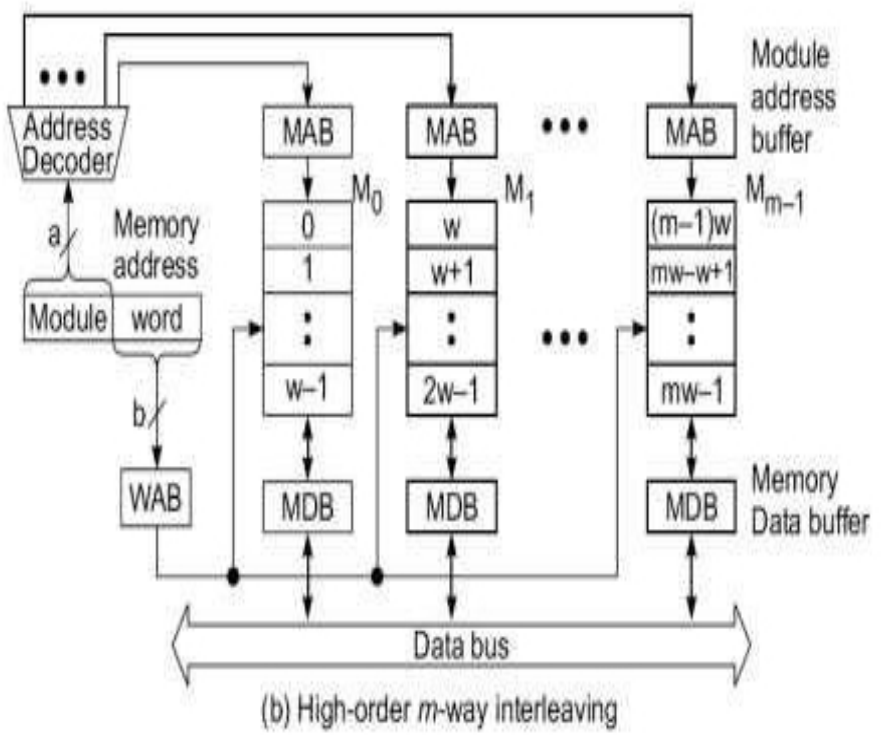


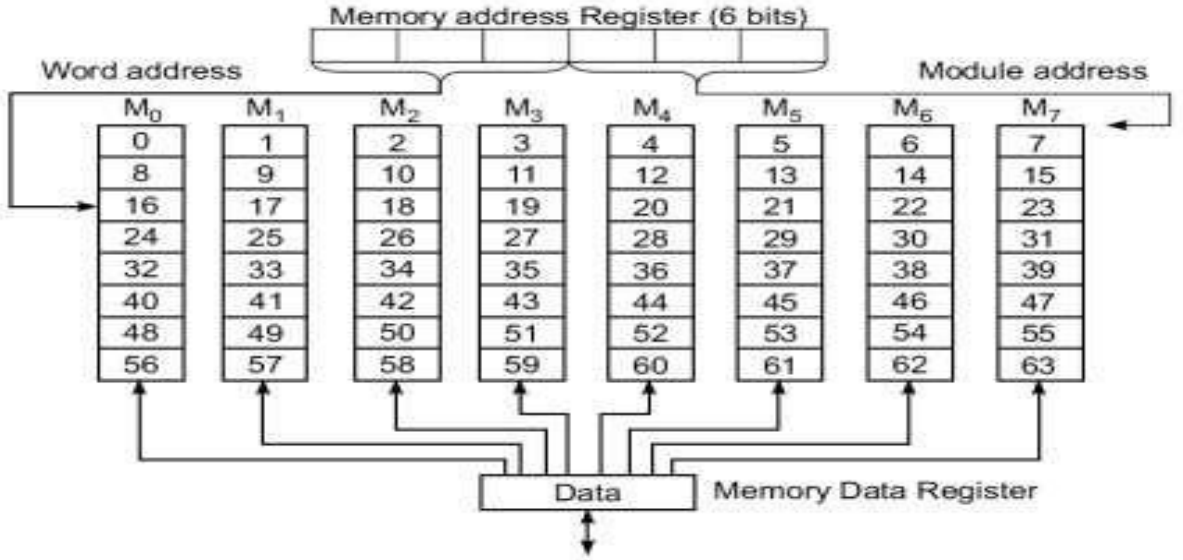
Fig. 5.15 Two interleaved memory organizations with $m = 2^a$ modules and $w = 2^b$ words per module (word addresses shown in boxes)

Memory Interleaving

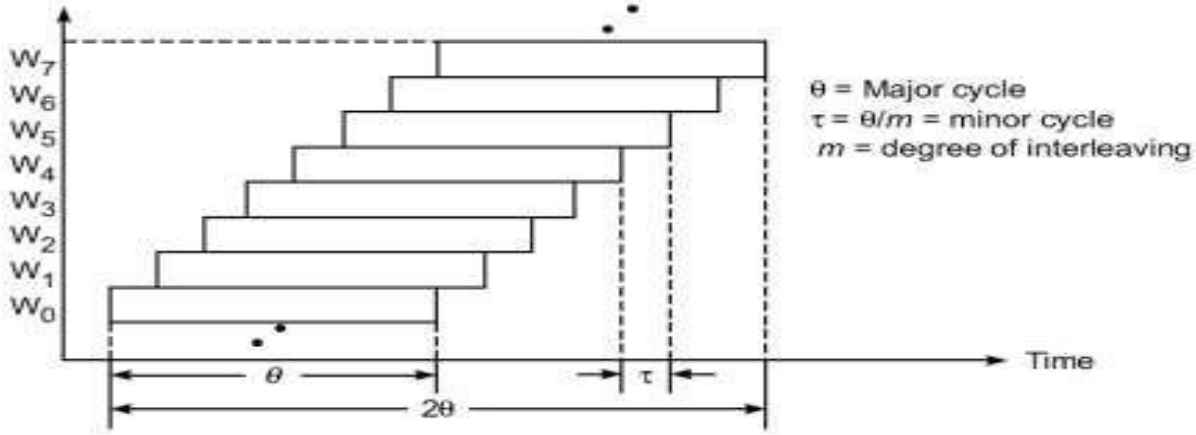
High-order interleaving

- High-order interleaving uses the high-order a bits as the module address and the low-order b bits as the word address within each module (Fig. 5.15b).
- Contiguous memory locations are thus assigned to the same memory module. In each memory cycle, only one word is accessed from each module.
- Thus the high-order interleaving cannot support block access of contiguous locations.

Memory Interleaving
Pipelined Memory Access



(a) Eight-way low-order interleaving (absolute address shown in each memory word)



(b) Pipelined access of eight consecutive words in a C-access memory

Fig. 5.16 Multiway interleaved memory organization and the C-access timing chart

Memory Interleaving

Pipelined Memory Access

- Access of the m memory modules can be overlapped in a pipelined fashion.
- For this purpose, the memory cycle (called the major cycle) is subdivided into m minor cycles.

An eight-way interleaved memory (with m=8 and w=8 and thus a=b=3) is shown in Fig. 5.16a.

- Let θ be the major cycle and τ the minor cycle. These two cycle times are related as follows:

$$\tau = \theta / m$$

m=degree of interleaving

θ =total time to complete access of one word

τ =actual time to produce one word. Then Total block access time is 2θ

Effective access time of each word is τ

The timing of the pipelined access of the 8 contiguous memory words is shown in Fig. 5.16b.

- This type of concurrent access of contiguous words has been called a C-access memory scheme.

Memory Interleaving

Pipelined Memory Access

- Access of the m memory modules can be overlapped in a pipelined fashion.
- For this purpose, the memory cycle (called the major cycle) is subdivided into m minor cycles.

An eight-way interleaved memory (with m=8 and w=8 and thus a=b=3) is shown in Fig. 5.16a.

- Let θ be the major cycle and τ the minor cycle. These two cycle times are related as follows:

$$\tau = \theta / m$$

m=degree of interleaving

θ =total time to complete access of one word

τ =actual time to produce one word. Then Total block access time is 2θ

Effective access time of each word is τ

The timing of the pipelined access of the 8 contiguous memory words is shown in Fig. 5.16b.

- This type of concurrent access of contiguous words has been called a C-access memory scheme.

Memory Interleaving

Bandwidth and Fault Tolerance

- Hellerman (1967) has derived an equation to estimate the effective increase in memory bandwidth through multiway interleaving. A single memory module is assumed to deliver one word per memory cycle and thus has a bandwidth of 1.

Memory Bandwidth

The memory bandwidth B of an m-way interleaved memory is upper-bounded by m and lower- bounded by I.
The Hellerman estimate of B is.

$$B = m^{0.56} \approx \sqrt{m}$$

where m is the number of interleaved memory modules.

- This equation implies that if 16 memory modules are used, then the effective memory bandwidth is approximately four times that of a single module.
- This pessimistic estimate is due to the fact that block access of various lengths and access of single words are randomly mixed in user programs.
- Hellerman's estimate was based on a single-processor system. If memory-access conflicts from multiple processors (such as the hot spot problem) are considered, the effective memory bandwidth will be further reduced.

Memory Interleaving

Bandwidth and Fault Tolerance

In a vector processing computer, the access time of a long vector with n elements and stride distance 1 has been estimated by Cragon (1992) as follows:

- **It is assumed that the elements are stored in contiguous memory locations in an m-way interleaved memory system.**

The average time t1 required to access one element in a vector is estimated by

$$t_1 = \frac{\theta}{m}\left(1 + \frac{m-1}{n}\right)$$

When n → ∞ (very long vector), t1→ θ/m = τ.
As n → 1 (scalar access), t1→ θ.
Equation 5.6 conveys the message that interleaved memory appeals to pipelined access of long vectors; the longer the better.

Memory Interleaving

Fault Tolerance

- High- and low-order interleaving can be combined to yield many different interleaved memory organizations.
- Sequential addresses are assigned in the high-order interleaved memory in each memory module.
- This makes it easier to isolate faulty memory modules in a memory bank of m memory modules.
- When one module failure is detected, the remaining modules can still bo used by opening a window in the address space.
- This fault isolation cannot be carried out in a low-order interleaved memory, in which a module failure may paralyze the entire memory bank.
- Thus low-order interleaving memory is not fault-tolerant.

Memory Allocation Schemes

- The idea of virtual memory is to allow many software processes time-shared use of main memory (precious with limited capacity resource).
- Memory manager: The portion of OS kernel that handles the allocation and deallocation of main memory to executing processes is called memory manager.
- The memory manager monitors the amount of available main memory and decides which processes should reside in main memory and which should be put back to disk if the main memory reaches its limit.
- Allocation policies

  Memory swapping is the process of moving blocks of information between the levels of a memory hierarchy.

  There are different swapping policies exits.

1. Nonpreemptive allocation
   - The incoming block cab be placed only in free region of the main memory.
   - When the main memory is full, this allocation scheme swaps out some of the allocated processes (or pages) to vacate space for the incoming block.
   - Easier to implement.
   - Not going to yield the good memory utilization.

Memory Allocation Schemes

2. **Preemptive allocation**

- **This allocation scheme allows the placement of an incoming block in a region presently occupied by another process.**
- **In both cases memory manager should try to allocate the free space first.**
- **This scheme has a freedom to pre-empt the executing process**
- **Offers more flexibility, but it requires mechanism be established to determine which pages or processes are to be swapped out and to avoid thrashing caused by excessive amount of swapping between the memory levels.**
- **Allocation policies are either local or global**

➢ **local Allocation policy**

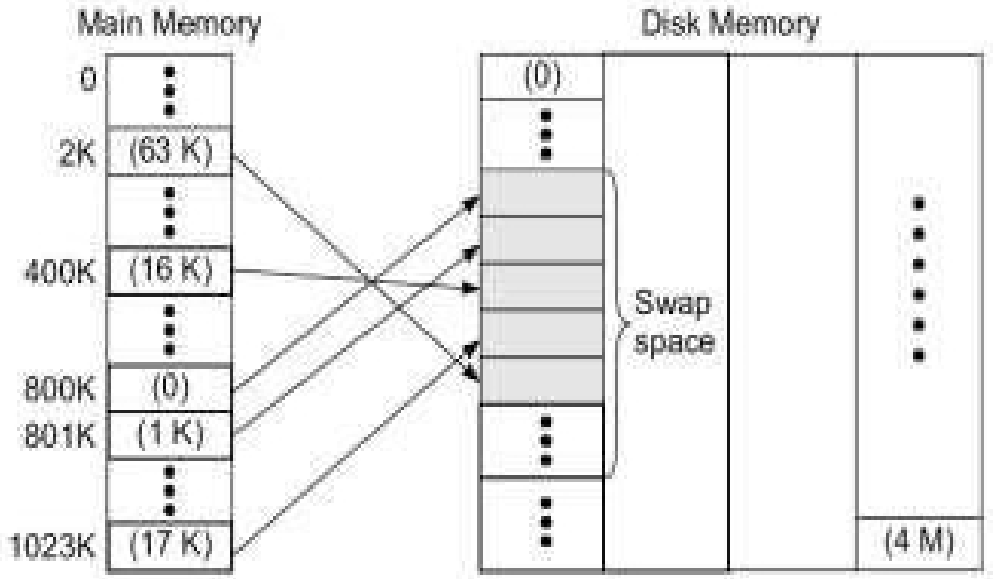
This involves only the resident working set of the faulty process.

➢ **Global Allocation policies**

This scheme considers the history of the working sets of all resident processes in making a swapping decision.
Most computer uses local policy.

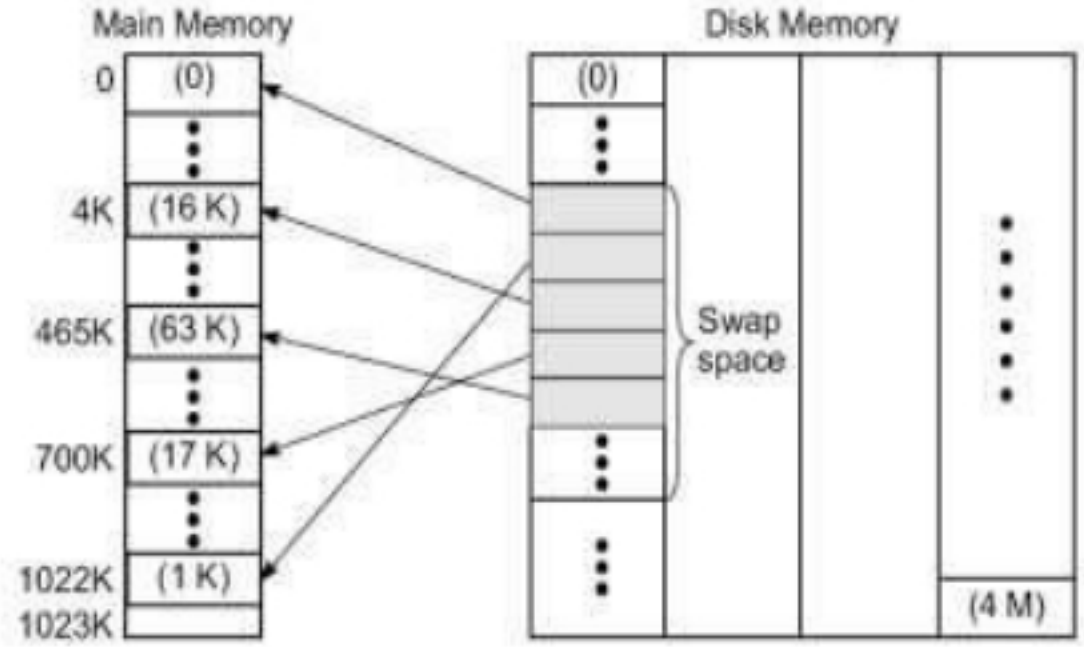
Swapping Systems

- Allow swapping only at entire process level
- **Swap space**: It is configurable section of a disk which is set aside for temporary storage of information being swapped out of the main memory. The portion of disk memory space set aside for a swap device is called swap space.
- Depending on system, may swap entire processes only, or the necessary pages



(a) Moving a process (or pages) onto the swap space on a disk

Swapping Systems



(b) Swapping in a process (or pages) to the memory

Fig. 5.18 The concept of memory swapping in a virtual memory hierarchy (virtual page addresses are identified by numbers within parentheses, assuming a page size of 1 K words)

Swapping Systems

- Memory manager allocates the disk space for program files one block at a time., but it allocates space on the swap device in groups of contiguous blocks.
- The virtual address space of a process may occupy a number of pages. The size of the process address space is limited by the amount of physical memory available on a swapping system.

Swapping in UNIX

- System calls that result in a swap: – Allocation of space for child process being created
– Increase in size of a process address space
– Increased space demand by stack for a process
– Demand for space by a returning process swapped out previously
- Special process 0 is the swapper: The swapper must swap the process into main memory before the kernel can schedule it for execution..

Only when there are process to swap in and eligible processes to swap out can the swapper do its work, otherwise the swapper goes to sleep.

Swapping Systems

- **Demand Paging Systems**

A paged memory system often uses a demand paging memory allocation policy.

- Allows only pages to be transferred b/t main memory and swap device. In fig 5.18 individual pages of a process are allowed to be independently swapped out and in. and we have demand paging system.
- In a demand paging system, the entire process does not have to move into the main memory to execute. Pages are brought in only on demand
- This Allows process address space to be larger than physical address space
- The major advantages of the demand paging is that it offers flexibility to dynamically accommodate large number of processes in physical memory on time-sharing or multiprogrammed basis with significantly enlarged address space.

- **Working Sets**
- **The idea of demand paging matches with working set concept.**

Set of pages referenced by the process during last n memory references (n=window size)

- Only working sets of active processes are resident in memory.

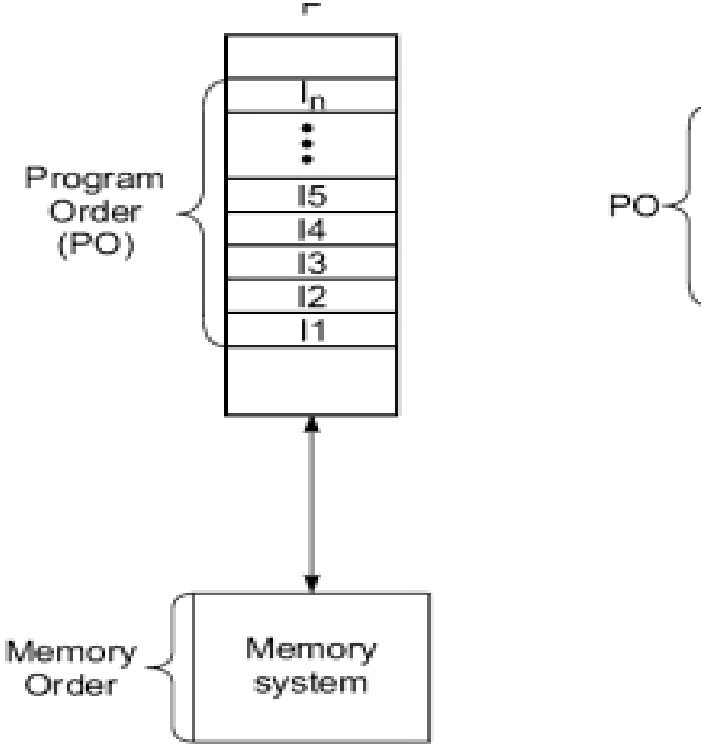
Sequential and Weak Consistency Models

- To study the shared memory behavior in relation to program execution order and memory access order.
- With respect program execution order and memory access order.

Memory inconsistency: when memory access order differs from program execution order

a. Uniprocessor system

- As shown in fig 5.19a, a uniprocessor system maps an SISD sequence into similar execution sequence.
- Thus memory accesses (instruction and data) are consistent with the program execution order. This property is called sequence consistency.

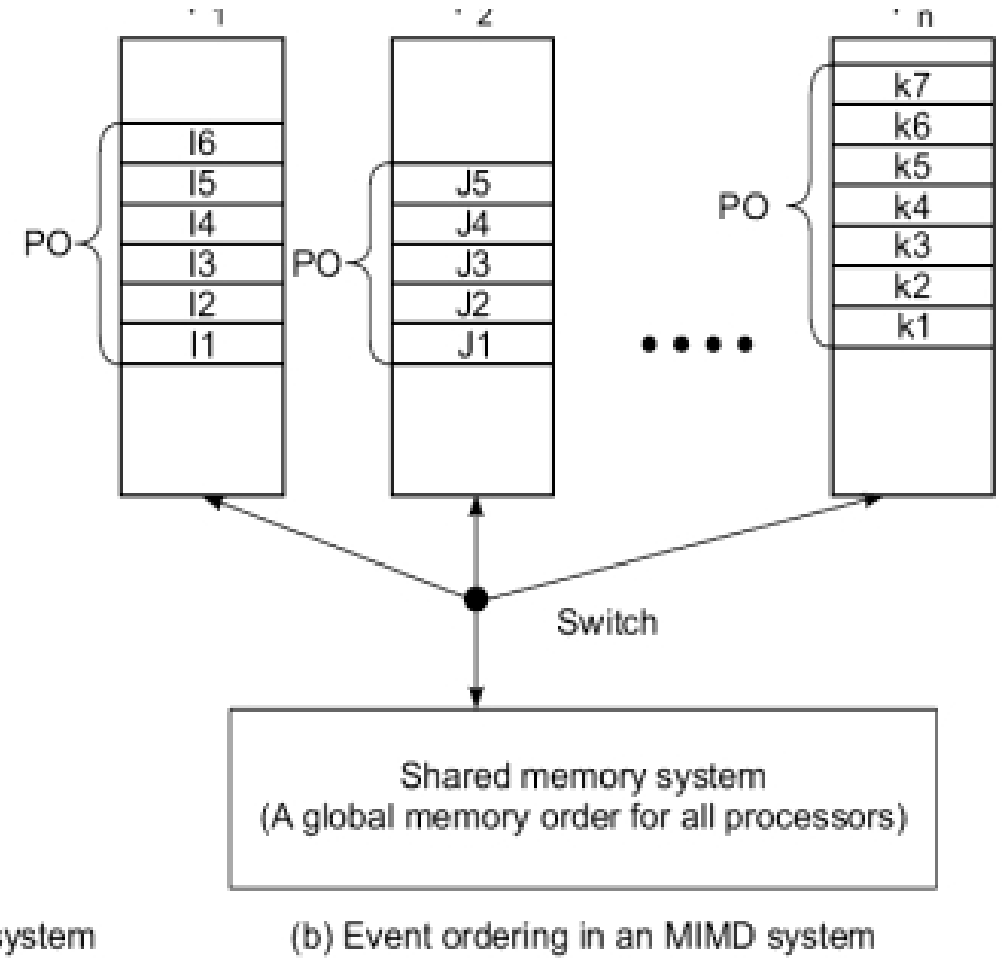


(a) Sequential consistency in an SISD system

Sequential and Weak Consistency Models cond…

b. Shared-memory multiprocessor: There are multiple instructions sequences in different processors as shown in fig 5.19b.

- **Different ways of interleaving the MIMD instruction Sequences into a global memory access sequence lead to Different shared memory behaviors.**



) system (b) Event ordering in an MIMD system

Sequential and Weak Consistency Models cond…

Memory Consistency Issues

• **Memory model:** behavior of a shared memory system as observed by processors is called Memory model.

Specification of memory model answers three fundamental questions

1. What behavior should a programmer/compiler expect from a shared memory multiprocessor?
2. How can a definition of the expected behavior guarantee coverage of all contingencies?
3. How must processors and the memory system behave to ensure consistent adherence to the expected behavior of the multiprocessor?

• **Choosing a memory model** – involves making a compromise between a strong model minimally restricting s/w and a weak model offering efficient implementation

• **Primitive memory operations**: load (read), store (write), and one or more synchronization operations such as swap (atomic load-store) or conditional store

Event Orderings

Processes: concurrent instruction streams executing on different • **Memory events** corresponds to shared memory access. Consistency models specify the order by which the events from one process should be observed by other processes in the machine. Event ordering helps determine if a memory event is legal for concurrent accesses

Program order: order by which memory access occur for execution of a single process, w/o any reordering

Event Orderings

Processes: Processes are concurrent instruction streams executing on different processors. Each process executes a code segment

• The order in which shared memory operations are performed by one process may be observed by other processes. •

Memory events corresponds to shared memory access. Consistency models specify the order by which the events from one process should be observed by other processes in the machine.

Event ordering helps determine if a memory event is legal or illegal for concurrent accesses, when several processes are accessing a common set of memory locations.

Program order: order by which memory access occur for execution of a single process, provided that no program reordering has taken place.

Three primitive memory operations for the purpose of specifying memory consistency models are defined:

1. **A load by processor Pi: is considered performed with respect to processor Pk at a point of time when the issuing of a store to the same location by Pk cannot affect the value returned by the load.**
2. **A store by Pi, is considered performed with respect to Pk at one time when an issued load to the same address by Pk returns the value by this store.**
3. **A load is globally performed if it is performed with respect to all processors and if the store that is the source of the returned value has been performed with respect to all processors.**

•

Atomicity

- Program order preserved and uniform observation sequence by all processors
- Out-of-program-order allowed and uniform observation sequence by all processors
- Out-of-program-order allowed and nonuniform sequences observed by different processors

Three categories of multiprocessor memory behavior:

Atomic memory accesses: memory updates are known to all processors at the same time. Thus store is atomic if the value stored becomes readable to all processors at the same time. Thus necessary and sufficient condition for an atomic memory to be sequentially consistent is that all memory accesses must be performed to preserve all individual program order.

Non-atomic: having individual program orders that conform is not a sufficient condition for sequential consistency. In a cache/network based multiprocessor a system can be nonatomic if an invalidation signal does reach all processors at a same time. Thus store nonatomic in this case.

Linear Pipeline Processors

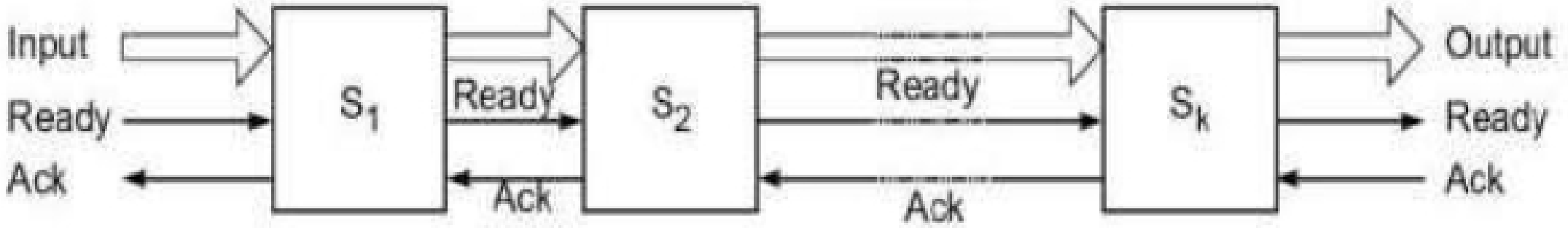
A linear pipeline processor is a cascade of processing stages which are linearly connected to perform a fixed function over a stream of data flowing from one end to the other.

In modern computers, linear pipelines are applied for instruction execution, arithmetic computation, and memory-access operations.

6.1.l Asynchronous & Synchronous models

- A linear pipeline processor is constructed with k processing stages. External inputs(operands) are fed into the pipeline at the first stage S1.
- The processed results are passed from stage Si to stage Si+1, for all i=1,2,….,k-1. The final result emerges from the pipeline at the last stage Sn.
- Depending on the control of data flow along the pipeline, we model linear pipelines in two categories:
  - Asynchronous
  - Synchronous.

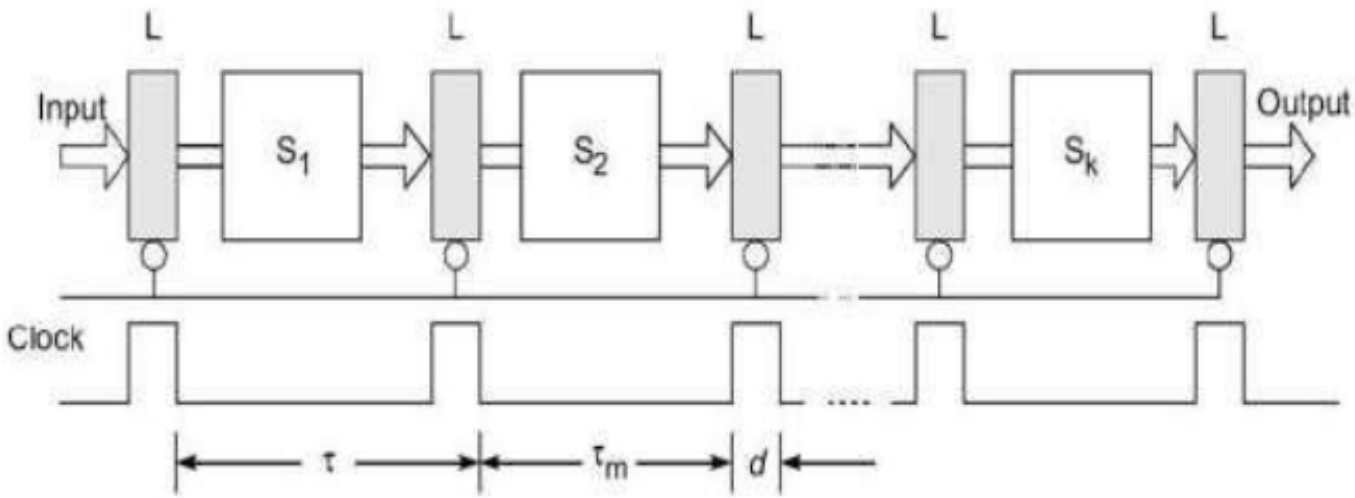
Asynchronous Pipeline Model



(a) An asynchronous pipeline model

- As shown in the figure data flow between adjacent stages in an asynchronous pipeline is controlled by a handshaking protocol.
- When stage Si is ready to transmit, it sends a ready signal to stage Si+1. After stage receives the incoming data, it returns an acknowledge signal to Si.
- Asynchronous pipelines are useful in designing communication channels in message- passing multicomputers where pipelined wormhole routing is practiced
- Asynchronous pipelines may have a variable throughput rate.
- Different amounts of delay may be experienced in different stages.

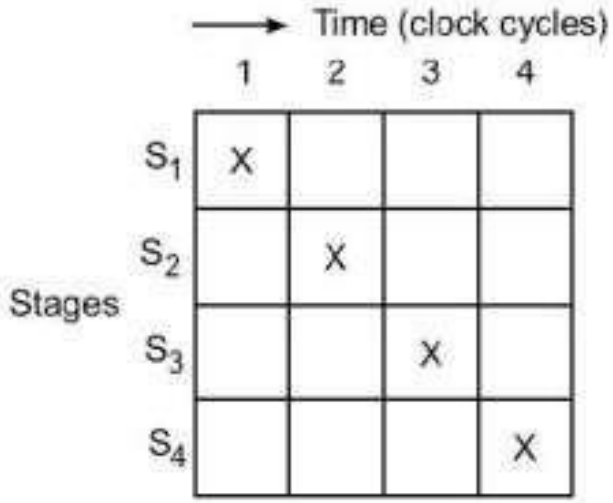
Synchronous Pipeline Model



(b) A synchronous pipeline model

- Synchronous pipelines are illustrated in Fig. Clocked latches are used to interface between stages.
- The latches are made with master-slave flip-flops, which can isolate inputs from outputs.
- Upon the arrival of a clock pulse All latches transfer data to the next stage simultaneously.
- The pipeline stages are combinational logic circuits. It is desired to have approximately equal delays in all stages.
- These delays determine the clock period and thus the speed of the pipeline. Unless otherwise specified, only synchronous pipelines are studied.
- The utilization pattern of successive stages in a synchronous pipeline is specified by a reservation table.

Synchronous Pipeline Model



Captions:
S_i = stage i
L = Latch
τ = Clock period
τ_m = Maximum stage delay
d = Latch delay
Ack = Acknowledge signal.

(c) Reservation table of a four-stage linear pipeline

- For a linear pipeline, the utilization follows the diagonal streamline pattern shown in Fig. 6.1c.
- This table is essentially a space-time diagram depicting the precedence relationship in using the pipeline stages.
- Successive tasks or operations are initiated one per cycle to enter the pipeline. Once the pipeline is filled up, one result emerges from the pipeline for each additional cycle.
- This throughput is sustained only if the successive tasks are independent of each other.

Ssynchronous Pipeline Model cond…

Clocking and Timing Control

The clock cycle τ of a pipeline is determined below. Let τi be the time delay of the circuitry in stage Si and d the time delay of a latch, as shown in Fig 6.1b.

a) Clock Cycle and Throughput :

Denote the maximum stage delay as τm ,and we can write τ as

$$\tau = \max_{i}\{\tau_i\}_1^k + d = \tau_m + d$$

At the rising edge of the clock pulse, the data is latched to the master flip-flops of each latch register. The clock pulse has a width equal to d.

- In general, τm >> d by one to two orders of magnitude.
- This implies that the maximum stage delay τm dominates the clock period. The pipeline frequency is defined as the inverse of the clock period.

$$f = 1 / \tau$$

- If one result is expected to come out of the pipeline per cycle, f represents the maximum throughput of the pipeline.

Synchronous Pipeline Model cond…

Depending on the initiation rate of successive tasks entering the pipeline, the actual throughput of the pipeline may be lower than f.

- This is because more than one clock cycle has elapsed between successive task initiations.

b) Clock Skewing:

- Ideally, we expect the clock pulses to arrive at all stages (latches) at the same time.
- However, due to a problem known as clock skewing the same clock pulse may arrive at different stages with a time offset of s.
- Let tmax be the time delay of the longest logic path within a stage and tmin is the shortest logic path within a stage.
- To avoid a race in two successive stages, we must choose

$$\tau m >= tmax + s \text{ and } d <= tmin - s$$

- These constraints translate into the following bounds on the clock period when clock skew takes effect:

$$d + tmax + s <= \tau <= \tau m + tmin - s$$

- In the ideal case $s = 0$, $tmax = \tau m$, and $tmin = d$. Thus, we have $\tau = \tau m + d$

Synchronous Pipeline Model cond…

c) Speedup, Efficiency and Throughput of Pipeline

Ideally, a linear pipeline of k stages can process n tasks in k + (n — 1) clock cycles, where k cycles are needed to complete the execution of the very first task and the remaining n-1 tasks require n - 1 cycles
Thus the total time required is

$$T_k = [k + (n - 1)]\tau$$

- where τ is the clock period.
- Consider an equivalent-function nonpipelined processor which has a flow-through delay of kτ. The amount of time it takes to execute n tasks on this nonpipelined processor is,

$$T1 = nk\tau$$

Synchronous Pipeline Model cond…

d) Speedup Factor

The speedup factor of a k-stage pipeline over an equivalent nonpipelined processor is defined as

$$S_k = \frac{T_1}{T_k} = \frac{nk\tau}{k\tau + (n-1)\tau} = \frac{nk}{k + (n-1)}$$

e) Efficiency and Throughput

The efficiency Ek of a linear k-stage pipeline is defined as $E_k = \frac{S_k}{k} = \frac{n}{k + (n-1)}$

The efficiency approaches 1 when $n \rightarrow \infty$, and a lower bound on Ek is 1/k when n = 1.

The pipeline throughput Hk is defined as the number of tasks (operations) performed per unit time:

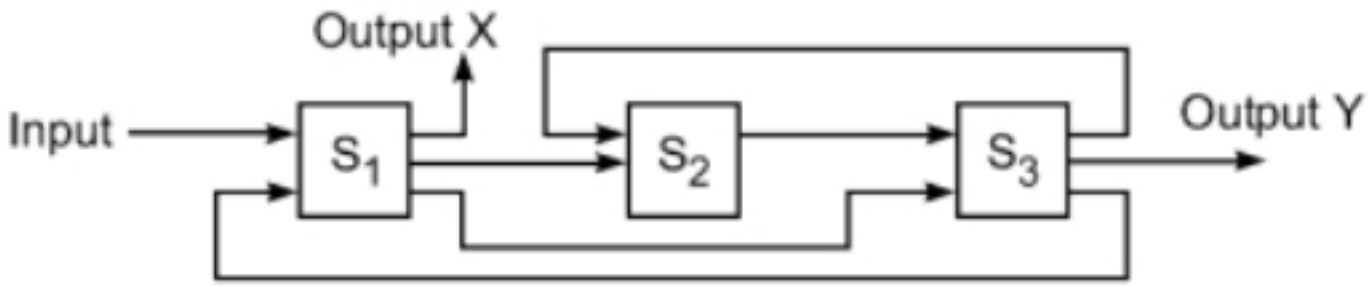
The maximum throughput f occurs when $E_k \rightarrow 1$ as $n \rightarrow \infty$.

Non Linear Pipeline Processors

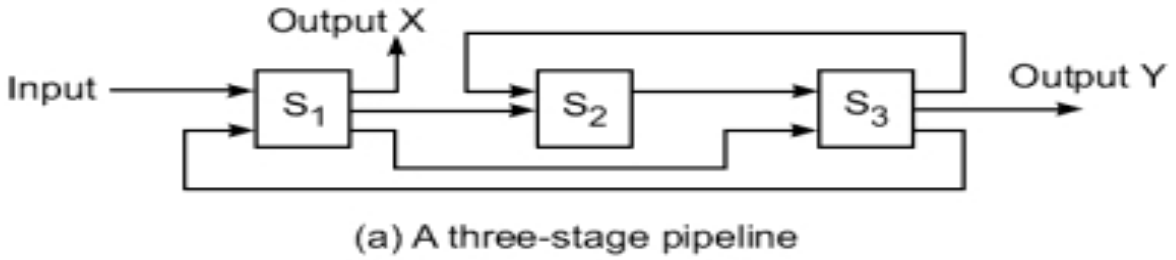
- A dynamic pipeline can be reconfigured to perform variable functions at different times.
- The traditional linear pipelines are static pipelines because they are used to perform fixed functions.
- A dynamic pipeline allows feed forward and feedback connections in addition to the streamline connections.

Reservation and Latency analysis:

- In a static pipeline, it is easy to partition a given function into a sequence of linearly ordered subfunctions.
- However, function partitioning in a dynamic pipeline becomes quite involved because the pipeline stages are interconnected with loops in addition to streamline connections.
- A multifunction dynamic pipeline is shown in Fig 6.3a. This pipeline has three stages.
-



(a) A three-stage pipeline

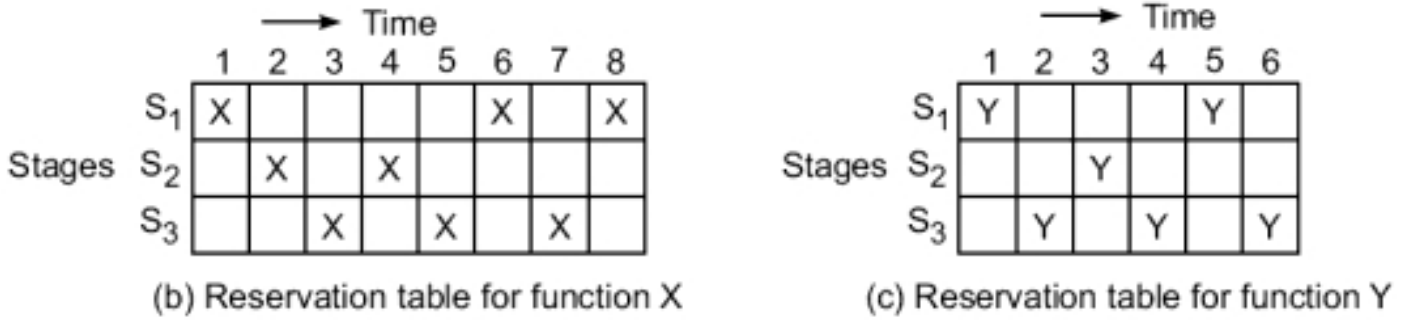


(a) A three-stage pipeline

Non Linear Pipeline Processors cond..

Reservation and Latency analysis:

- A multifunction dynamic pipeline is shown in Fig 6.3a. This pipeline has three stages.
- Besides the streamline connections from S1 to S2 and from S2 to S3,
- Feed forward connection from S1 to S3 and
- Two feedback connections from S3 to S2 and from S3 to S1.
- These feed forward and feedback connections make the scheduling of successive events into the pipeline a nontrivial task.
- With these connections, the output of the pipeline is not necessarily from the last stage.
- In fact, following different dataflow patterns, one can use the same pipeline to evaluate different functions



(b) Reservation table for function X

(c) Reservation table for function Y

Fig. 6.3 A dynamic pipeline with feed forward and feedback connections for two different functions

Non Linear Pipeline Processors cond…

Reservation and Latency analysis:

- **Latency Analysis**

➢ **Pipeline latency** The number of time units (clock cycles) between two initiations of a pipeline is the latency between them.

• Latency values must be non negative integers. A latency of k means that two initiations are separated by k clock cycles.

➢ **Collision** Any attempt by two or more initiations to use the same pipeline stage at the same time will cause a collision.

• A collision implies resource conflicts between two initiations in the pipeline. Therefore, all collisions must be avoided in scheduling a sequence of pipeline initiations.

• Some latencies will cause collisions, and some will not.

➢ **Forbidden latencies** Latencies that cause collisions are called forbidden latencies.

Non Linear Pipeline Processors cond…

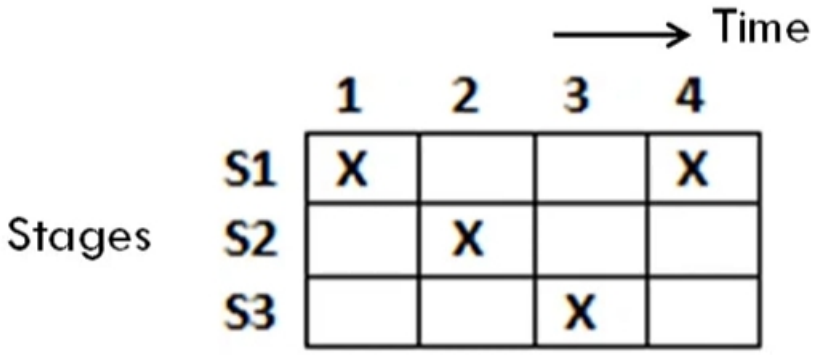
Reservation and Latency analysis:

- **Latency Analysis**

Collision Free Scheduling

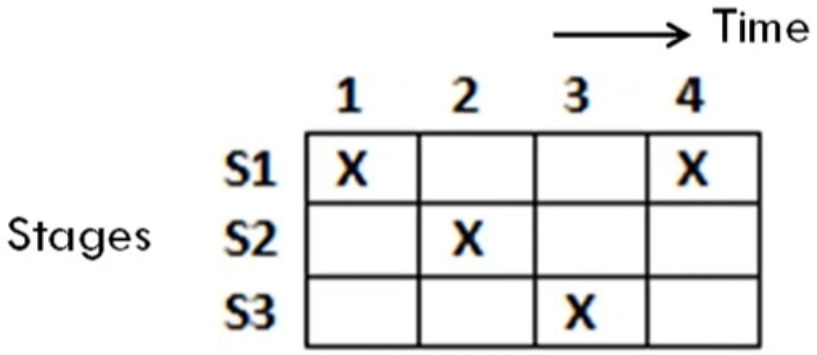
- When scheduling events in a nonlinear pipeline, the main objective is to obtain the shortest average latency between initiations without causing collisions.
- **Collision Vector**: By examining the reservation table, one can distinguish the set of permissible latencies from the set of forbidden latencies.
- For a reservation table with n columns, the maximum forbidden latency in m<=n-1. The permissible latency p should be as small as possible.
- The choice is made in the range 1 <= p <= m-1.
- A permissible latency of p = 1 corresponds to the ideal case. In theory, a latency of 1 can always be achieved in a static pipeline which follows a linear (diagonal or streamlined) reservation table.

SEPT/AUG 2020



1. What are the forbidden latencies and what is initial collision vector?
2. Draw the state transition diagram for scheduling the pipeline.
3. List all the simple cycles and greedy cycles.
4. Determine the optimal constant cycle and the MAL.
5. Let the pipeline clock period be $\tau = 20ns$. Determine the throughput of this pipeline

SEPT/AUG 2020



1. What are the forbidden latencies and what is initial collision vector?
2. Draw the state transition diagram for scheduling the pipeline.
3. List all the simple cycles and greedy cycles.
4. Determine the optimal constant cycle and the MAL.
5. Let the pipeline clock period be $\tau = 20ns$. Determine the throughput of this pipeline

Solution

	1	2	3	4
S1	X			X
S2		X		
S3			X	

1. Forbidden latencies

S1:{(4-1)} = {3}

S2:{0}

S3:{0}

Forbidden latencies {3} It causes collision

Maximum Forbidden latencies is {3}

2. Permissible latencies $\{1,2,4^{+}\}$ It does not cause causes collision

3. Collision vector

As Maximum Forbidden latencies is {3}, Collision vector contains 3 bits

Collision vector: {C3, C2, C1} = {1, 0, 0} or Initial Collision Vector (ICV)={1, 0, 0}

1→ Forbidden latencies , 0→ Permissible latencies

Solution

	1	2	3	4
S1	X			X
S2		X		
S3			X	

How to draw state transition diagram

Ans:

The permissible latency = {1, 2, 4+}, so we have to find the next states of collosion vector for the transitions 1, 2, 4+

Method

To find the collision vector of the next state the collision vector of the present state is shifted to right for each transition {1, 2, 4+}. The shifted collision vector of present state is bitwise OR with the Initial Collision Vector

Step1: Calculating next state with Present state (PS)=1 0 0

Permissible latencies (or transitions) = {1, 2, 4+}.

Initial Collision Vector = 1 0 0

With latency 1

```
  1 0 0
+ 0 1 0  (After right shift of PS by 1)
  1 1 0  Next state
```

New state is 1 1 0 with latency 1

Solution

Step1: Calculating next state with Present state (PS)=1 0 0
Permissible latencies (or transitions) = {1, 2, 4+}.

With latency 1

```
  1 0 0
+ 0 1 0  (After right shift of PS by 1)
  1 1 0  Next state
```

New state is 1 1 0 with latency 1

With latency 2

```
  1 0 0
+ 0 0 1  (After right shift of PS by 2)
  1 0 1  Next state
```

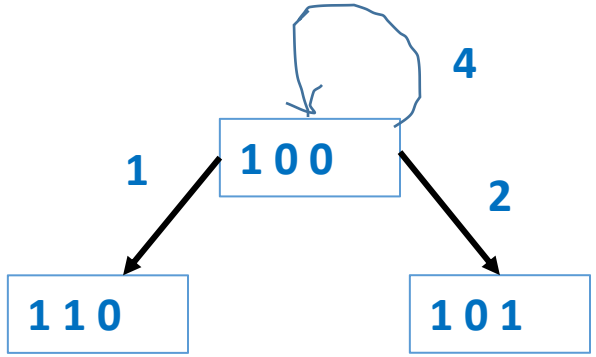
New state is 1 0 1 with latency 2

With latency 4

```
  1 0 0
+ 0 0 0  (After right shift of PS by 4)
  1 0 0  Next state
```

New state is 1 0 0 with latency 4+, it is it returns to initial state 1 0 0

Initial Collision Vector = 1 0 0



Initial state transition diagram

Solution

Step2: Calculating next state with Present state (PS) = 1 1 0 consider as 0 1 1 0 (4 is permissible latency)

(0 1 1 0 here 1→ Forbidden latencies , 0→ Permissible latencies)

Permissible latencies (or transitions) = {1, 4+}.

With latency 1

1 0 0

\+ 0 1 1 (After right shift of PS by 1)

1 1 1 Next state

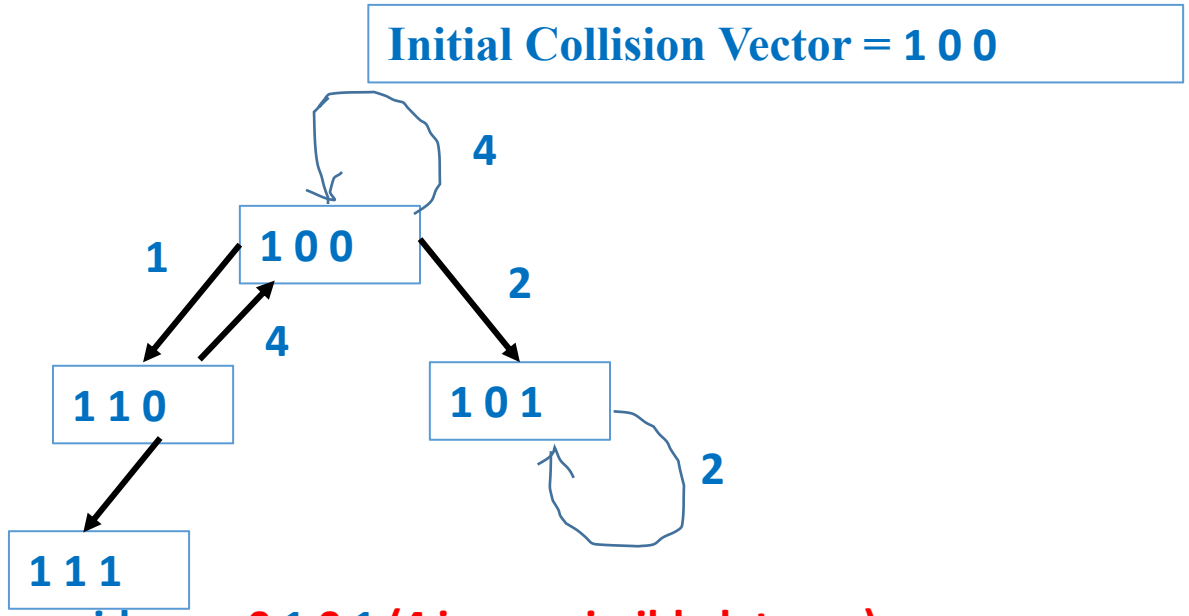
New state is 1 1 1 with latency 1

With latency 4^+

1 0 0

\+ 0 0 0 (After right shift of PS by 4)

1 0 0 Next state



Step3: Calculating next state with Present state (PS) = 1 0 1 consider as 0 1 0 1 (4 is permissible latency)

(0 1 1 0 here 1→ Forbidden latencies , 0→ Permissible latencies)

Permissible latencies (or transitions) = {2, 4+}.

With latency 2

1 0 0

\+ 0 0 1 (After right shift of PS by 2)

1 0 1 Next state New state is 1 0 1 with latency 2

With latency 2

1 0 0

\+ 0 0 1 (After right shift of PS by 1)

1 0 1 Next state New state is 1 0 1 with latency 2

Solution

Step3: Calculating next state with Present state (PS) = 1 0 1 consider as 0 1 0 1 (4 is permissible latency)

(0 1 1 0 here 1→ Forbidden latencies , 0→ Permissible latencies)

Permissible latencies (or transitions) = {2, 4^+}.

With latency 4

1 0 0

\+ 0 0 0 (After right shift of PS by 4)

1 0 0 Next state New state is 1 0 0 with latency 4

Step3: Calculating next state with Present state (PS) = 1 1 1 consider as 0 1 1 1 (4 is permissible latency)

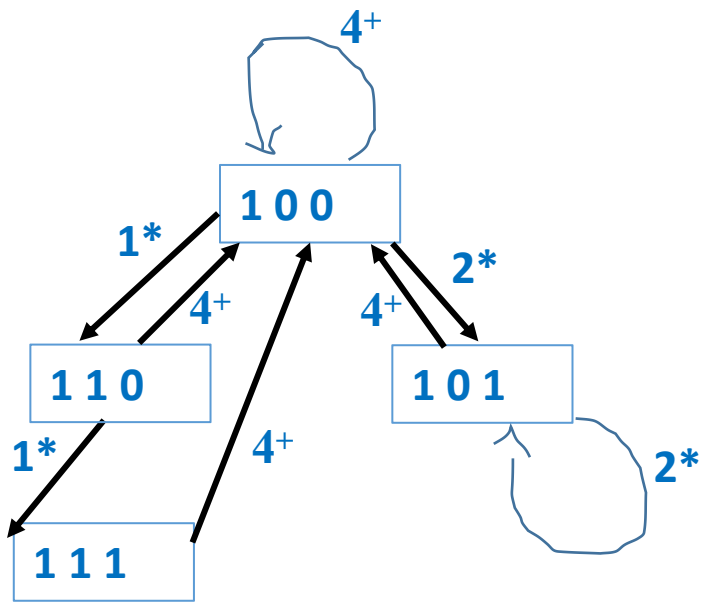
Permissible latencies (or transitions) = {4^+}.

With latency 4

1 0 0

\+ 0 0 0 (After right shift of PS by 0)

1 0 0 Next state New state is 1 0 0 with latency 4



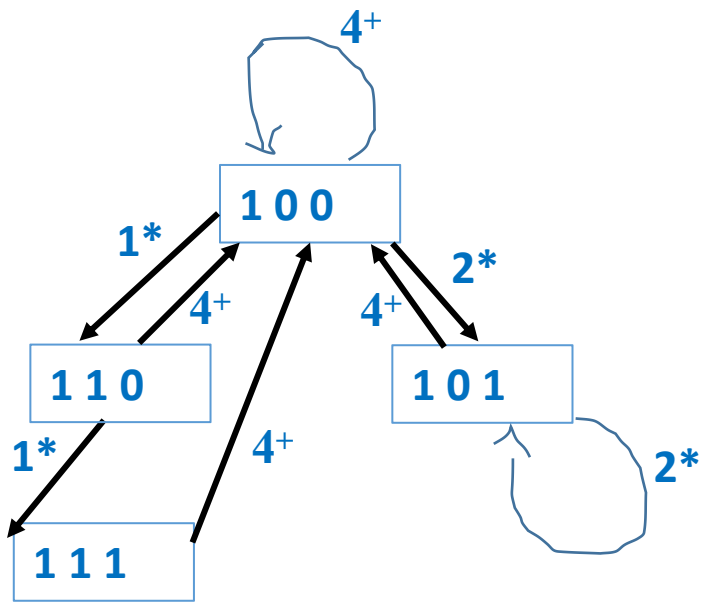
Final state transition diagram

Solution

Simple cycle: It is a latency cycle in which each state appears only one

Greedy cycle: It is simple cycle whose edges are all made with minimum latencies from their respective starting state

Latency Cycle	Types of cycle	Average latency
(2)	Simple cycle (greedy cycle)	2
(4)	Simple cycle	4
(1, 4)	Simple cycle (greedy cycle)	2.5
(2, 4)	Simple cycle	3
(1, 1, 4)	Simple cycle	2
(2, 2, 4)	Not Simple cycle	
(2, 4, 4)	Not Simple cycle	



Final state transition diagram

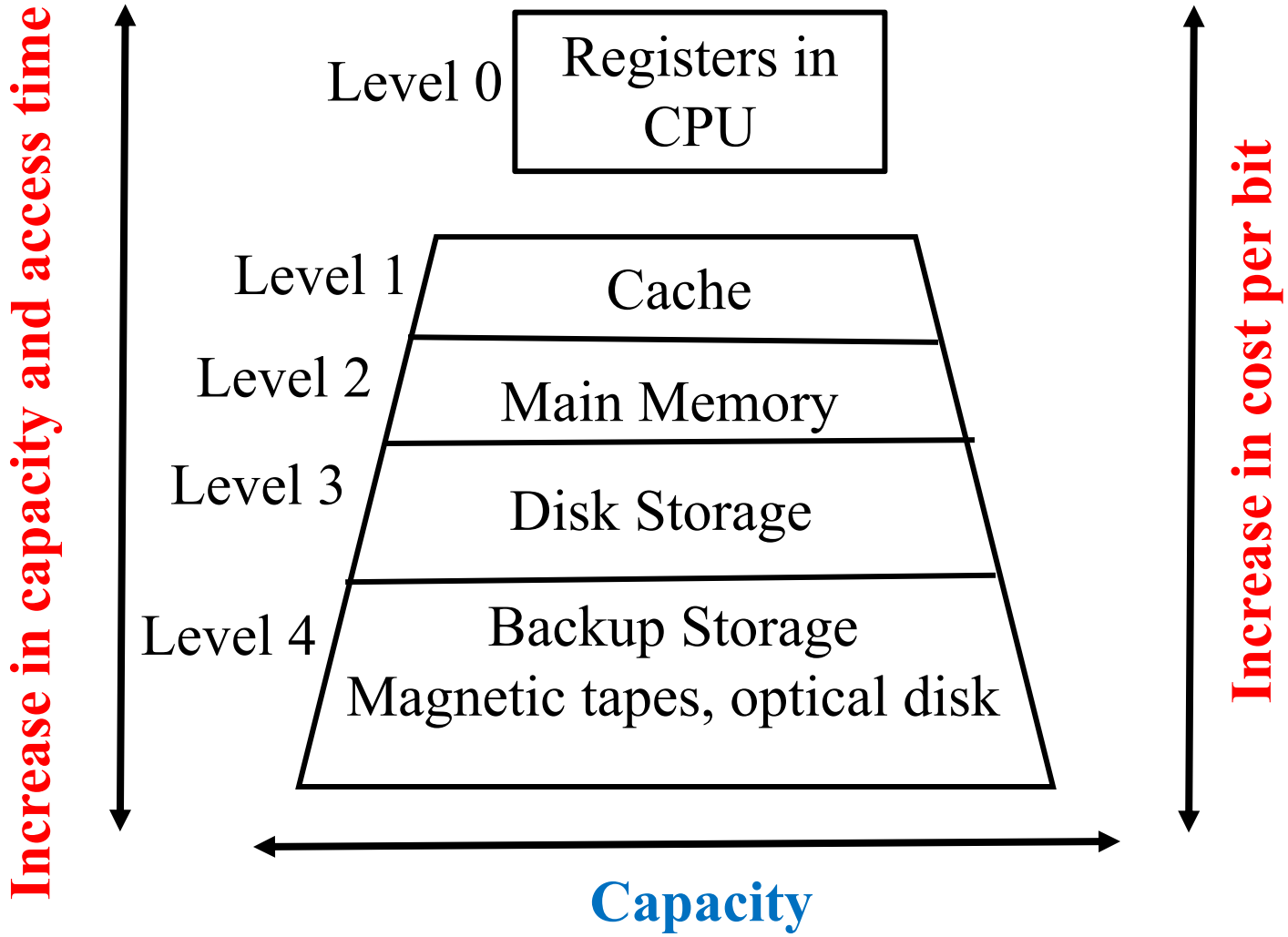
Minimum Average Latency (MAL)=2

Solution

$$\text{Throughput} = \frac{f}{\text{MAL}} = \frac{1}{\text{MAL}*\tau}$$

$$= \frac{1}{2*20\text{ns}} = \frac{1}{40*10^{-9}} = \frac{10^9}{40} = \frac{10^6*10^3}{40} = 25 \text{ MBPS}$$

Four Level Memory



Four Level Memory cond…

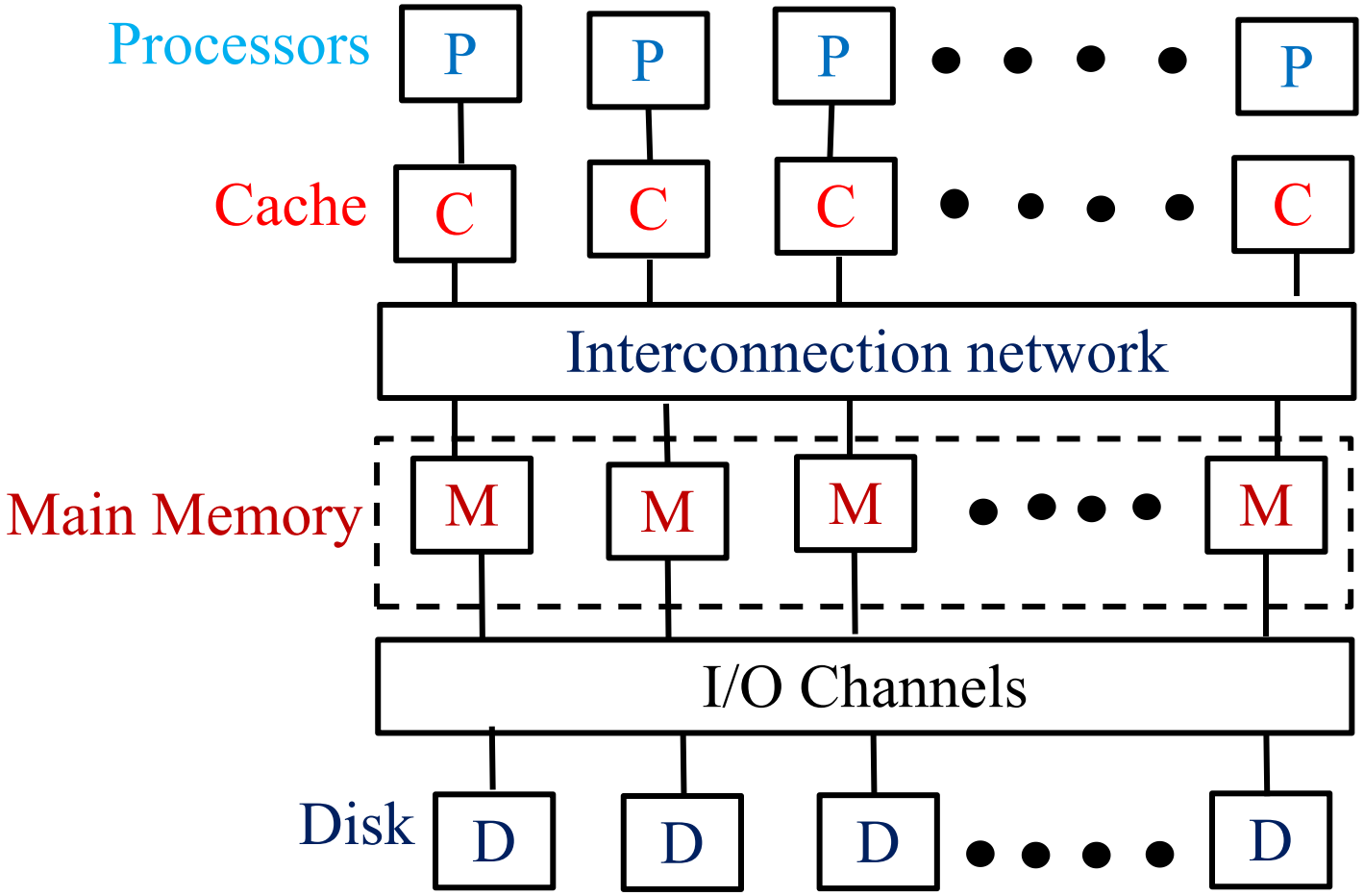


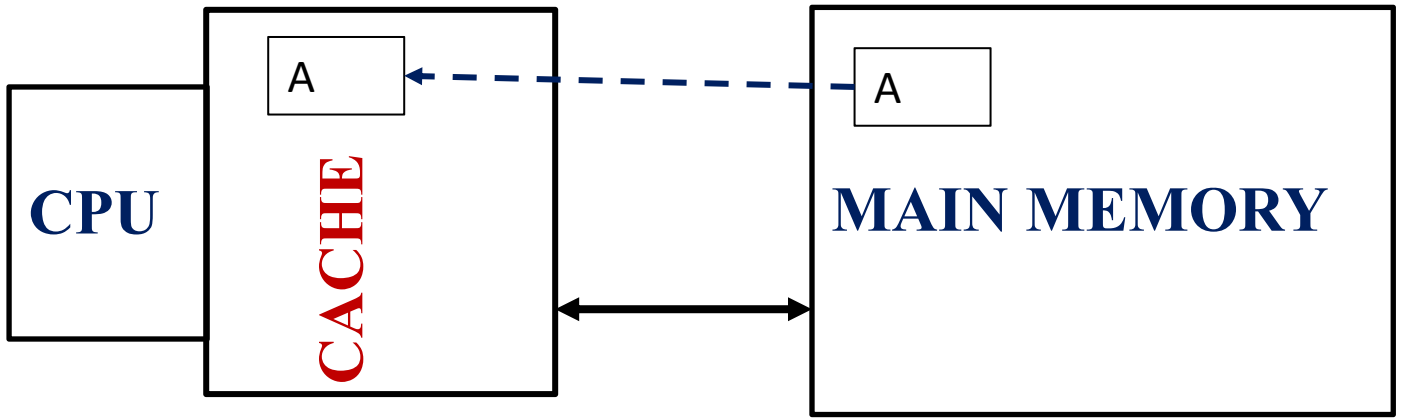
Fig 5.6 Memory Hierarchy for shared-Memory Multi processor

Memory is divide into equal size blocks

Cache is divide into blocks (also be know as lines) whose size is same as memory size.

Cache Memory

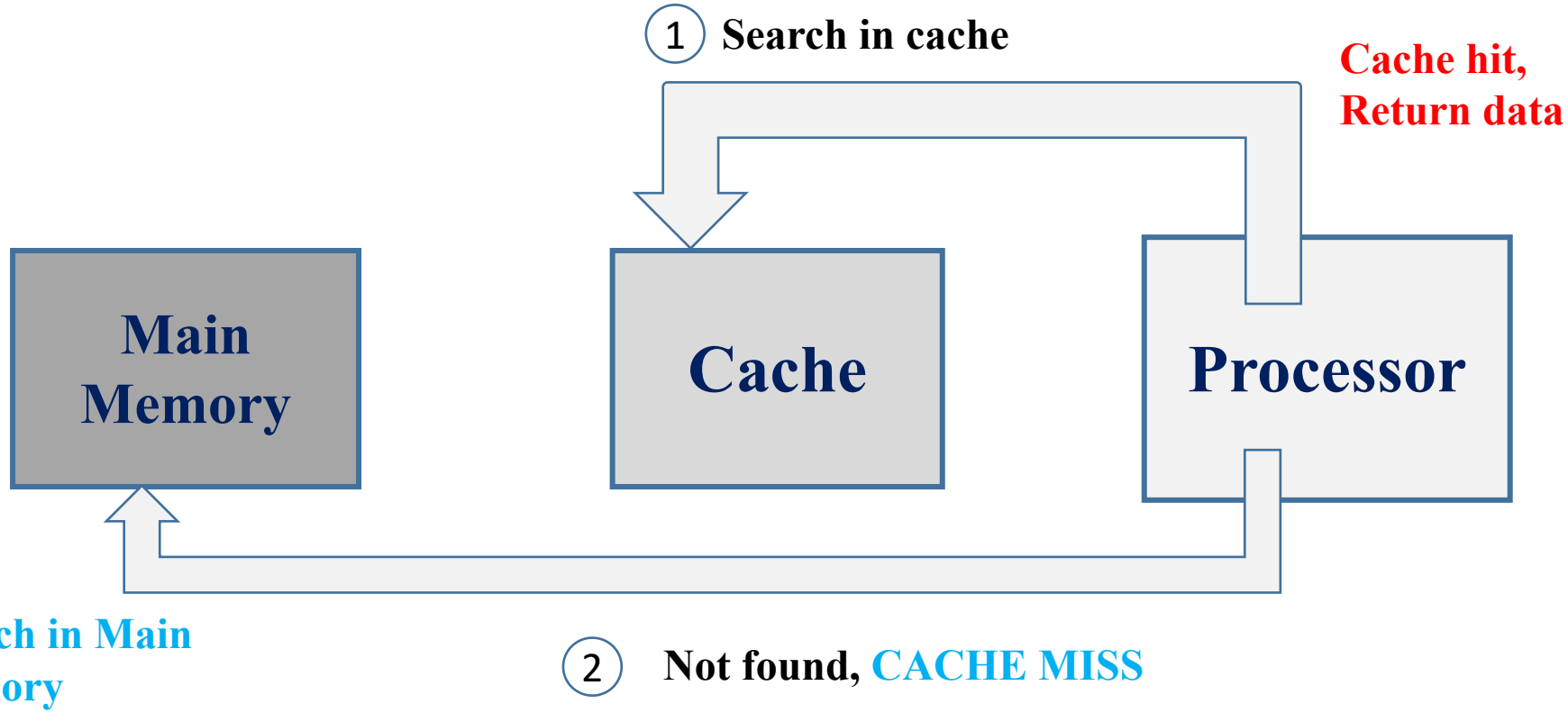
- A cache is a compact, fast memory.
- It lies between registers and RAM in memory hierarchy.
- It holds recently used data and/or instructions.



- Cache acts as a temporary stand-in Temporary copy of main memory.
- CPU can read from and write to cache in much less time than memory.

Cache memory cond…

- **Cache Hit-** Requested data is successfully retrieved from the cache.
- **Cache miss-** It is an event in which a processor makes a request to retrieve data from a cache, but that specific data is not currently in cache memory.



Cache memory cond…

A cache miss occurs either because the data was never placed in the cache, or because the data was removed ("evicted") from the cache by the caching system

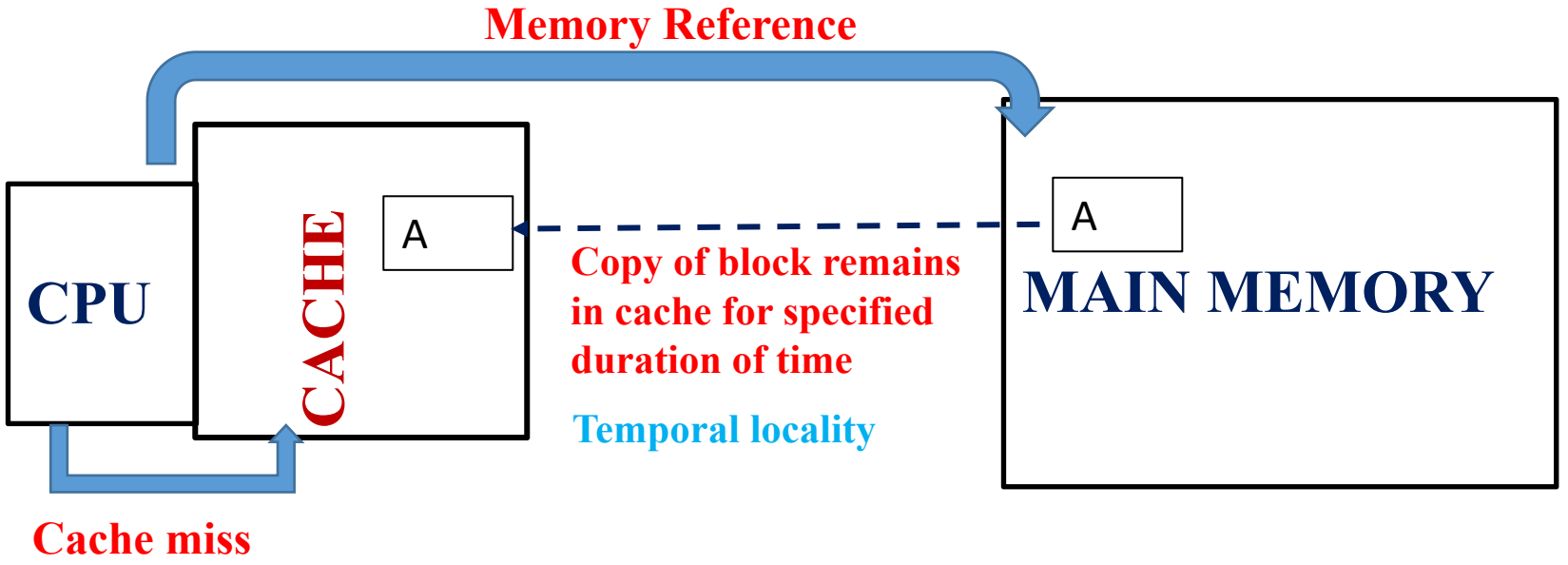
Locality of Reference or principle of locality: It is the tendency of a processor to access the same set of memory locations (same memory blocks) repetitively over a short period of time

- **Temporal locality (time)**
  Reuse of specific data and/or resources within a relatively small time duration
- **Spatial Locality (Space)**
  Spatial locality (also termed data locality) refers to the use of data elements within relatively close storage locations
- **Sequential Locality**
  Special case of spatial locality, occurs when data elements are arranged and accessed linearly, such as traversing the elements in a one-dimensional array.

Locality of Reference or principle of locality cond…

- **Temporal locality (time)**

  Reuse of specific data and/or resources within a relatively small time duration



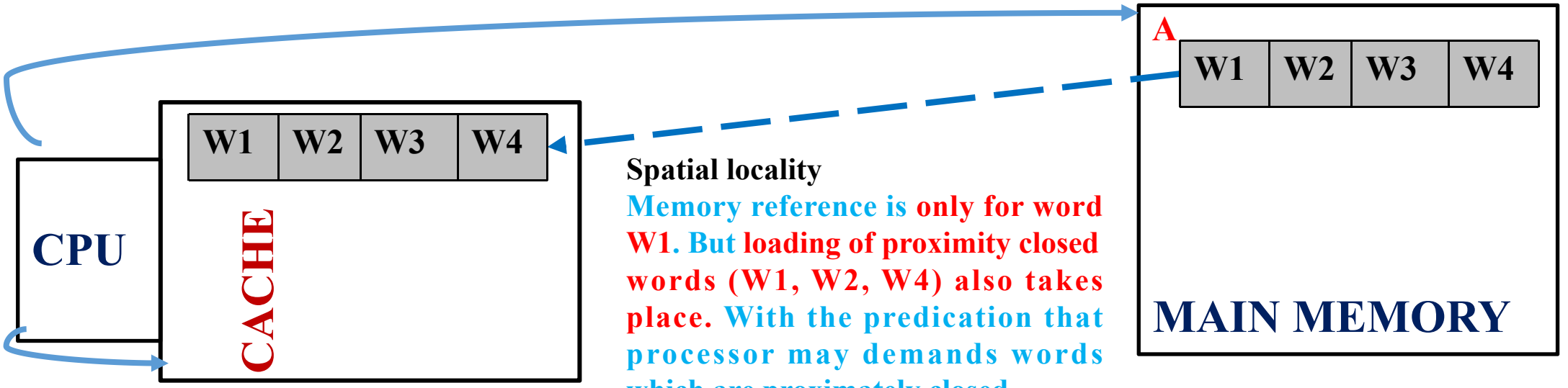
Copy of block remains in cache for specified duration of time with a predication that processor may demand the copy of data in very nearer future

Locality of Reference or principle of locality cond…

- **Spatial Locality (Space)**

  Spatial locality (also termed data locality) refers to the use of data elements within relatively close storage locations

Result of CACHE MISS leads to Memory Reference for ONLY word W1



Spatial locality
Memory reference is only for word W1. But loading of proximity closed words (W1, W2, W4) also takes place. With the predication that processor may demands words which are proximately closed.

Processor checks the Cache for word W3. As word W3 is not in cache "CACHE MISS" occurs

Locality of Reference cont…

Locality is a type of predictable behavior that occurs in computer systems. Systems that exhibit strong locality of reference are great candidates for performance optimization through the use of techniques such as

- **Caching**
- **Prefetching**
- **Branch predictors**
- **Pipelining stage of a processor core.**

Locality of Reference cont

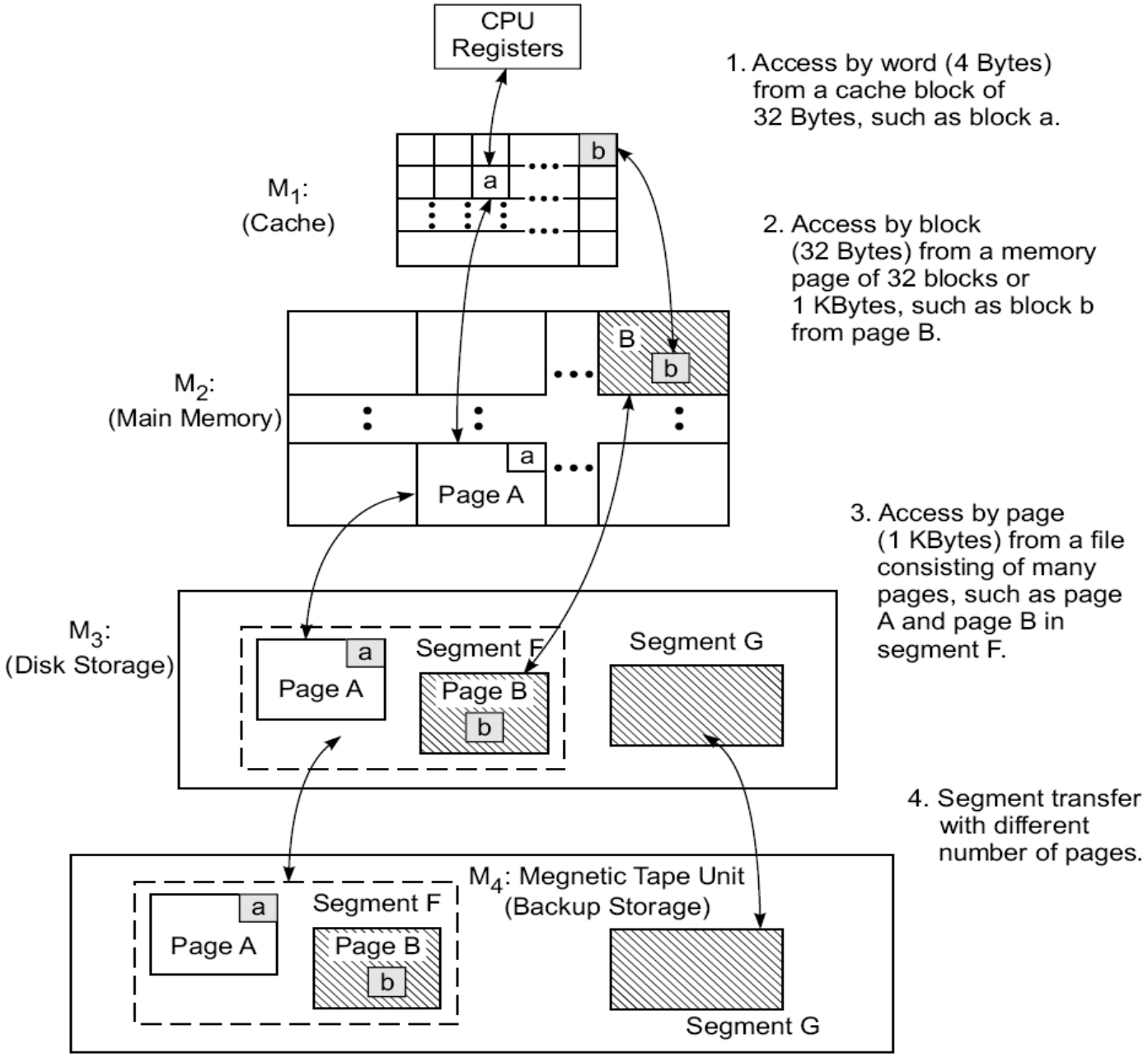


Fig. 4.18 The inclusion property and data transfers between adjacent levels of a memory hierarchy