

CS 5803

*Introduction to High Performance Computer
Architecture: Address Accessible Memories*

A.R. Hurson

323 CS Building,

Missouri S&T

hurson@mst.edu

Introduction to High Performance Computer Architecture



◆ Outline

- * Memory System: A taxonomy
- * Memory Hierarchy
- * Access Gap and Size Gap
 - Definition
 - How to reduce the access gap
 - Cache memory
 - Interleaved memory
 - How to reduce the size gap
 - Paging
 - Segmentation

Introduction to High Performance Computer Architecture

Note, this unit will be covered in three weeks. In case you finish it earlier, then you have the following options:

- 1) Take the early test and start CS5803.module5
- 2) Study the supplement module
(supplement CS5803.module4)
- 3) Act as a helper to help other students in studying CS5803.module4

Note, options 2 and 3 have extra credits as noted in course outline.

Enforcement of background

Glossary of prerequisite topics

Familiar with the topics? **No** → Review CS5803.module4.background

Yes

Take Test

Pass? **No** → Remedial action

Yes

Glossary of topics

Familiar with the topics? **No** → Take the Module

Yes

Take Test

Pass? **No** → Take the Module

Yes

Options

Study next module?

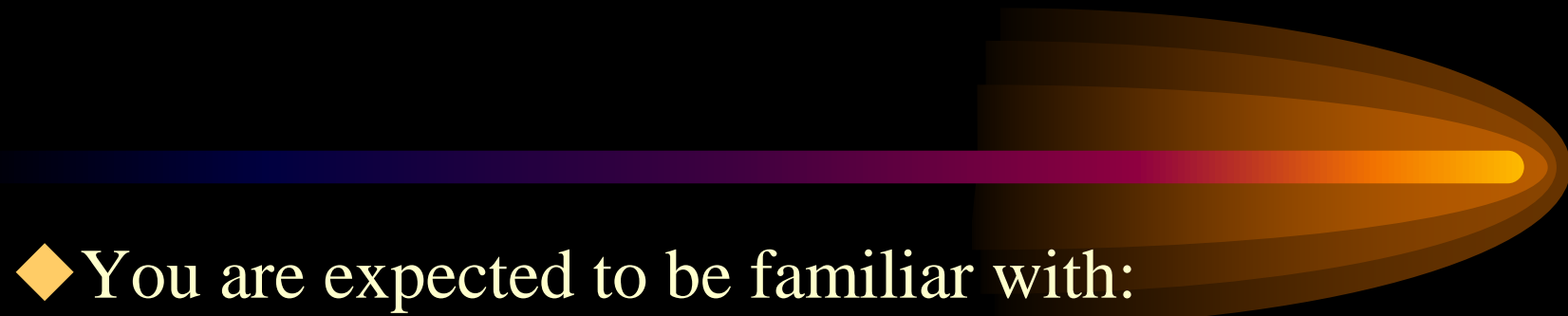
Lead a group of students in this module (extra credits)?

Study more advanced related topics (extra credits)?

At the end: take exam, record the score, and impose remedial action if not successful

Extra Curricular activities

Introduction to High Performance Computer Architecture

- 
- ◆ You are expected to be familiar with:
 - ✦ Basic hardware requirements of a random access memory,
 - ✦ Basic functionality of a random access memory,
 - ✦ The sequence of basic operations (μ -operations) of a random access memory, and
 - ✦ Basic definition and concept of Interleaved memory and Cache memory
 - ◆ If not then you need to study
CS5803.module4.background

◆ Memory System

- ★ In pursuit of improving the performance and hence to reduce the CPU time

$$T = I_c * CPI * \tau = I_c * (p + m * k) * \tau$$

in this section we will talk about the memory system.

- ★ The goal is to develop means to reduce m and k .

◆ Memory System

- ★ Different parameters can be used in order to classify the memory systems.
- ★ In the following we will use the **access mode** in order to classify memory systems
- ★ Access mode is defined as the way the information stored in the memory is accessed.

◆ Memory System — Access Mode

- ★ **Address Accessible Memory:** Where information is accessed by its address in the memory space.
- ★ **Content Addressable Memory:** Where information is accessed by its contents (or partial contents).

◆ Memory System — Access Mode

- ★ Within the scope of address accessible memory we can distinguish several sub-classes;
 - **Random Access Memory (RAM)**: Access time is independent of the location of the information.
 - **Sequential Access Memory (SAM)**: Access time is a function of the location of the information.
 - **Direct Access Memory (DAM)**: Access time is partially independent of and partially dependent on the location of the information.

◆ Memory System — Access Mode

- ★ Even within each subclass, we can distinguish several sub subclasses.
- ★ For example within the scope of Direct Access Memory we can recognize different groups:
 - Movable head disk,
 - Fixed head disk,
 - Parallel disk

◆ Memory System

- ★ **Movable head disk:** Each surface has just one read/write head. To initiate a read or write, the read/write head should be positioned on the right track first — **seek time**.
- ★ Seek time is a mechanical movement and hence, relatively, very slow and time consuming.

◆ Memory System

- ★ **Fixed head disk:** Each track has its own read/write head. This eliminates the seek time. However, this performance improvement comes at the expense of cost.

◆ Memory System

- ★ **Parallel disk:** To respond the growth in performance and capacity of semiconductor, secondary storage technology, introduced **RAID** — **R**edundant **A**rray of **I**nexpensive **D**isks.
- ★ In short RAID is a large array of **small independent disks** acting as a single high performance logical disk.

◆ Memory System

- ★ Within the scope of Random Access Memory we are concerned about two major issues:
 - **Access Gap:** Is the difference between the CPU cycle time and the main memory cycle time.
 - **Size Gap:** Is the difference between the size of the main memory and the size of the information space.

◆ Memory System

- ★ Within the scope of the memory system, the goal is to design and build a system with low cost per bit, high speed, and high capacity. In other words, in the design of a memory system we want to:
 - Match the rate of the information access with the processor speed.
 - Attain adequate performance at a reasonable cost.

◆ Memory System

- ★ The appearance of a variety of hardware as well as software solutions represents the fact that in the worst cases the trade-off between **cost**, **speed**, and **capacity** can be made more attractive by combining different hardware systems coupled with special features — **memory hierarchy**.

Introduction to High Performance Computer Architecture

◆ Access gap

★ How to reduce the access gap bottleneck:

- Software Solutions:

- Devise algorithmic techniques to reduce the number of accesses to the main memory.

- $$T = I_c * CPI * \tau = I_c * (p + m * k) * \tau$$

- Advances in technology
- Interleaved memory
- Application of registers
- Cache memory

◆ Access gap — Interleaved Memory

- ★ A memory is n -way interleaved if it is composed of n independent modules, and a word at address i is in module number $i \bmod n$.
- ★ This implies consecutive words in consecutive memory modules.
- ★ If the n modules can be operated independently and if the memory bus line is time shared among memory modules then one should expect an increase in bandwidth between the main memory and the CPU.

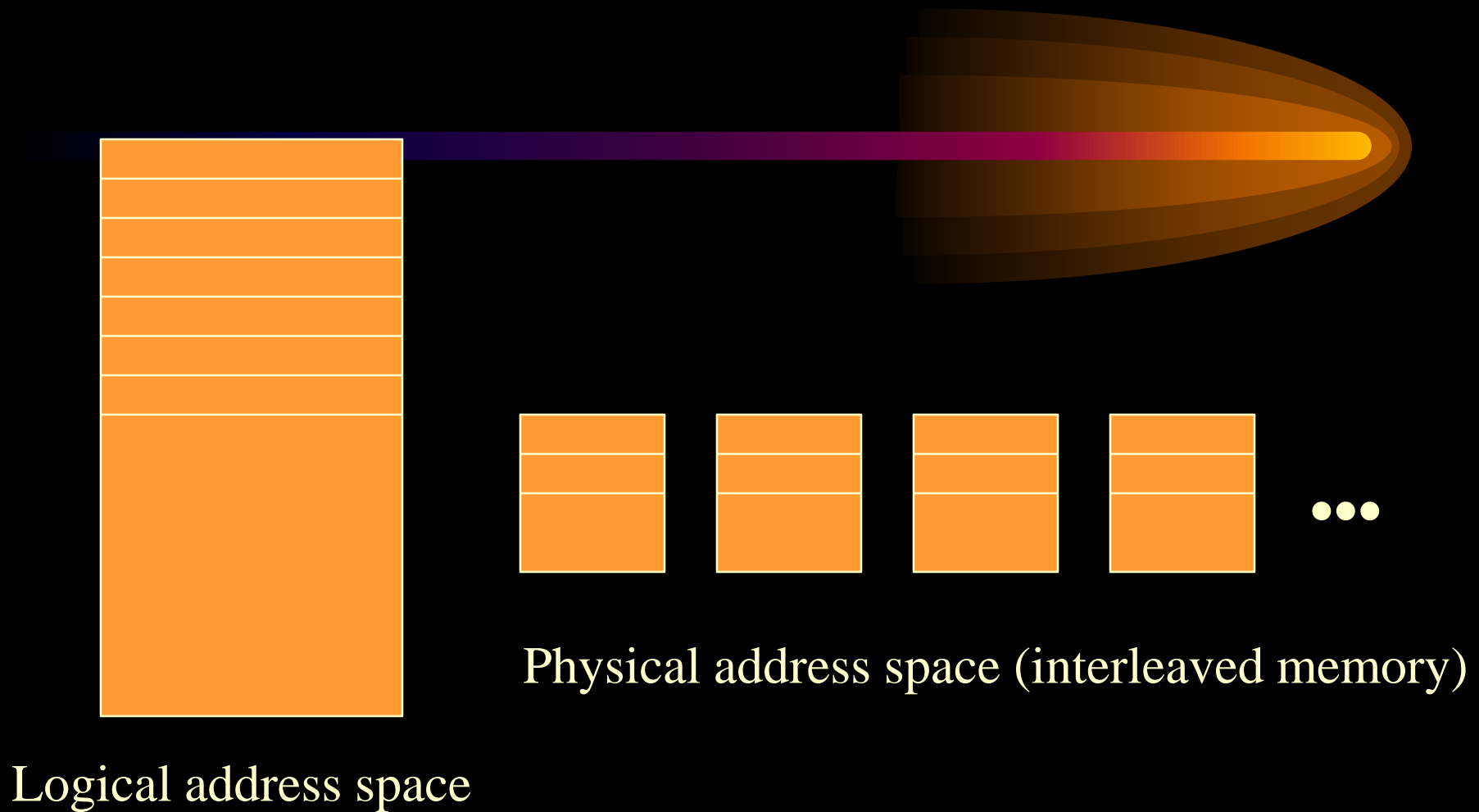
◆ Access gap — Interleaved Memory

- ★ Dependencies in the programs — **branches** and **randomness** in accessing the data will degrade the effect of memory interleaving.

◆ Access gap — Interleaved Memory

- ★ To show the effectiveness of memory interleaving, assume a pure sequential program of m instructions.
- ★ For a conventional system in which main memory is composed of a single module, the system has to go through m -fetch cycles and m -execute cycles in order to execute the program.
- ★ For a system in which main memory is composed of n modules, the system executes the same program by executing $\lceil m/n \rceil$ -fetch cycles and m -execute cycles.

Introduction to High Performance Computer Architecture



Mapping of logical addresses to physical addresses

◆ **Access gap — Interleaved Memory**

- ★ In general when the main memory is composed of n different modules, the addresses in the address space can be distributed among the memory modules in two different fashions:
 - Consecutive addresses in consecutive memory modules — **Low Order Interleaving**.
 - Consecutive addresses in the same memory module — **High Order Interleaving**.

◆ Access gap — Interleaved Memory

★ Whether low order interleaving or high order interleaving, a word address is composed of two parts:

- Module Address, and
- Word Address.



Address format in interleaved memory organization

◆ Questions

- ★ Name and discuss the factors which influence the **speed**, **cost**, and **capacity** of the main memory the most.
- ★ Compare and contrast low-order interleaving against high-order interleaving.
- ★ Dependencies in the program degrade the effectiveness of the memory interleaving — justify it.

◆ **Access gap — Interleaved Memory**

- ★ Within the scope of interleaved memory, a **memory conflict** (contention, interference) is defined if two or more addresses are issued to the same memory module.
- ★ In the worst case all the addresses issued are referred to the same memory module.
- ★ In this case the system's performance will be degraded to the level of a single module memory organization.

◆ **Access gap — Interleaved Memory**

- ★ To take advantage of interleaving, CPU should be able to perform **look ahead** fetches — issuing addresses before they are really needed.
- ★ In the case of **straight line programs** and lack of random-data-access such a look ahead policy can be enforced very easily and effectively.

◆ Interleaved Memory — Effect of Branch

- ★ Assume λ is the probability of a successful branch. Hence $1-\lambda$ is the probability of a sequential instruction (in case of a straight line program then λ is zero).
- ★ In the case of interleaving where memory is composed of n modules, CPU employs a look-ahead policy and issues n -instruction fetches to the memory.
- ★ Naturally, memory utilization will be degraded if one of these n instructions generate a successful branch.

◆ Interleaved Memory — Effect of Branch

- $P(1) = \lambda$ Prob. of the 1st instruction to generate a successful branch.
- $P(2) = \lambda(1-\lambda)$ Prob. of the 2nd instruction to generate a successful branch.
- \vdots
- $P(k) = \lambda(1-\lambda)^{k-1}$ Prob. of the k^{th} instruction to generate a successful branch.
- \vdots
- $P(n) = (1-\lambda)^{n-1}$ Prob. of 1st $(n-1)$ instructions to be sequential instructions.

◆ Interleaved Memory — Effect of Branch

- ★ Note in case of $P(n)$, it does not matter whether or not the last instruction is a sequential instruction.
- ★ The average number of memory modules to be used effectively

$$IB_n = \sum_{k=1}^n k P(k) = \lambda + 2 \lambda(1-\lambda) + \cdots + n (1-\lambda)^{n-1} = \frac{1 - (1-\lambda)^n}{\lambda}$$

◆ Interleaved Memory — Effect of Branch

★ Example

- For $\lambda = 5\%$ and $n = 4$ then $IB_n = 3.8$
- For $\lambda = 5\%$ and $n = 8$ then $IB_n = 6.8$
- For $\lambda = 10\%$ and $n = 4$ then $IB_n = 3.4$
- For $\lambda = 10\%$ and $n = 8$ then $IB_n = 5.7$
- Less branching, as expected, implies higher memory utilization.
- Memory utilization is not linear in the number of memory.

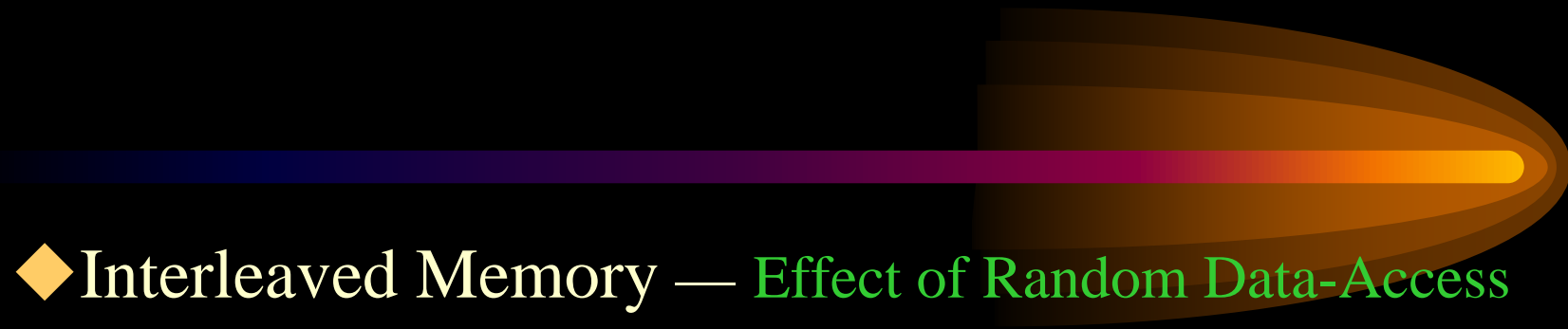
◆ Interleaved Memory — Effect of Branch

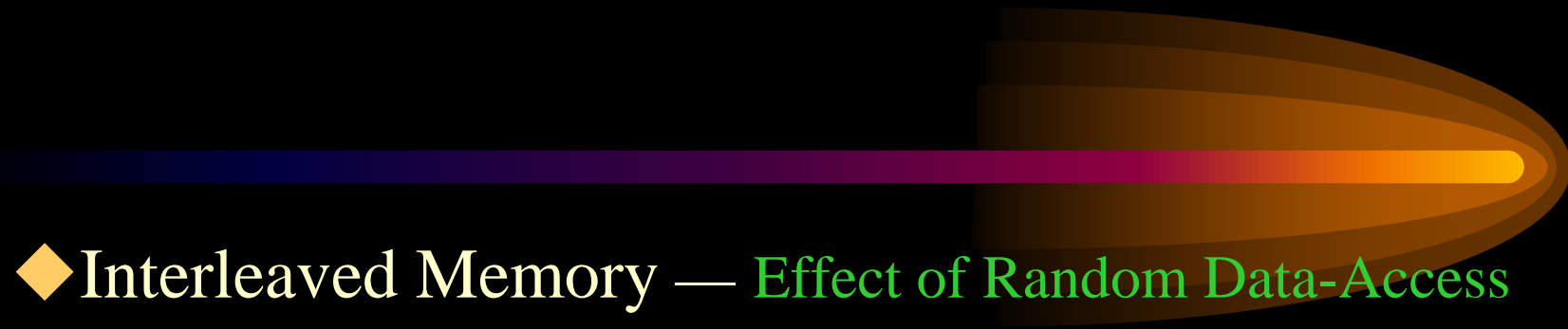
$$\begin{aligned}
 IB_n &= \lambda + 2\lambda(1-\lambda) + \dots + n(1-\lambda)^{n-1} = \\
 &= \lambda[1+2(1-\lambda) + \dots + (n-1)(1-\lambda)^{n-2}] + n(1-\lambda)^{n-1} = \\
 &= \lambda \left[\sum_{i=0}^{n-2} (i+1)(1-\lambda)^i \right] + n(1-\lambda)^{n-1} = \\
 &= \lambda \left(\sum_{i=0}^{n-2} \frac{d}{d\lambda} (1-\lambda)^{i+1} \right) + n(1-\lambda)^{n-1} \\
 &= -\lambda \frac{d}{d\lambda} \left(\sum_{i=0}^{n-2} (1-\lambda)^{i+1} \right) + n(1-\lambda)^{n-1} = \\
 &= -\lambda \frac{d}{d\lambda} \left(\frac{(1-\lambda) - (1-\lambda)(1-\lambda)^{n-1}}{1-(1-\lambda)} \right) + n(1-\lambda)^{n-1}
 \end{aligned}$$

◆ Interleaved Memory — Effect of Branch

⋮

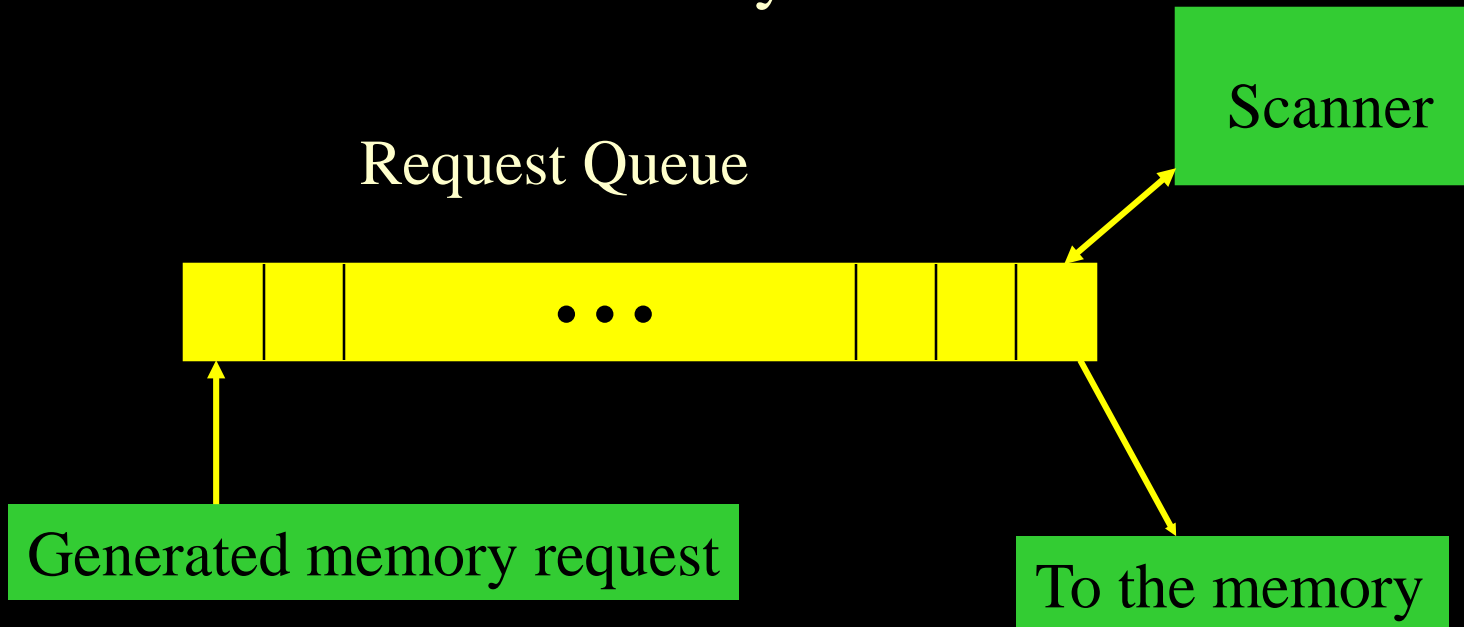
$$= \frac{1 - (1 - \lambda)^n}{\lambda}$$

- 
- ◆ Interleaved Memory — Effect of Random Data-Access
 - ✳ In case of data-access, the effectiveness of the interleaved memory will be compromised if among the n requests made to the memory, some are referred to the same memory module.

- 
- ◆ Interleaved Memory — Effect of Random Data-Access
 - ✳ Access requests are queued,
 - ✳ A **scanner** will check the request in the head of the queue:
 - If **no conflict**, the request is passed to the memory,
 - If **conflict** then the scanning is suspended as long as the conflict is not resolved.

Introduction to High Performance Computer Architecture

◆ Interleaved Memory — Effect of Random Data-Access



◆ Interleaved Memory — Effect of Random Data-Access

$$P(1) = \frac{n}{n} \frac{1}{n}$$

Prob. of one successful access.

$$P(2) = \frac{n(n-1)}{n^2} \frac{2}{n}$$

Prob. of two successful accesses.

•
•
•

$$P(k) = \frac{n(n-1) \dots (n-k+1)}{n^k} \frac{k}{n}$$

Prob. of k successful accesses.

◆ Interleaved Memory — Effect of Random Data-Access

★ The average number of active memory modules is:

$$\begin{aligned} DB_n &= \sum_{k=1}^n k P(k) = \sum_{k=1}^n \frac{k^2}{n^k} (n-1)(n-2) \dots (n-k+1) = \\ &= \sum_{k=1}^n \frac{k^2 (n-1)!}{n^k (n-k)!} \approx \sqrt{\frac{\pi n}{2}} - \frac{1}{3} + \frac{1}{12} \sqrt{\frac{\pi}{2n}} + O(n^{-1}) \end{aligned}$$

- For $1 \leq n \leq 45$ $DB_n \approx n^{.56}$
- If $n = 16$ one can conclude that on average, just 4 modules can be kept busy under randomly generated access requests.

Introduction to High Performance Computer Architecture

◆ Interleaved Memory — *Effect of Random Data-Access*

- ★ Naturally, performance of the interleaved memory under random data-access can be improved by **not allowing** an access to a busy module to stop other accesses to the main memory.
- ★ In another words, the conflicting access is queued and retried again.
- ★ This concept was first implemented in the design of CDC6600 — *Stunt Box*.

◆ Interleaved Memory — **Stunt Box**

- ★ **Stunt Box** is designed to provide a maximum flow of addresses to the Main memory.
- ★ **Stunt Box** is a piece of hardware that controls and regulates accesses to the main memory.
- ★ **Stunt Box** allows access **out-of-order** to the main memory.

◆ Interleaved Memory — **Stunt Box**

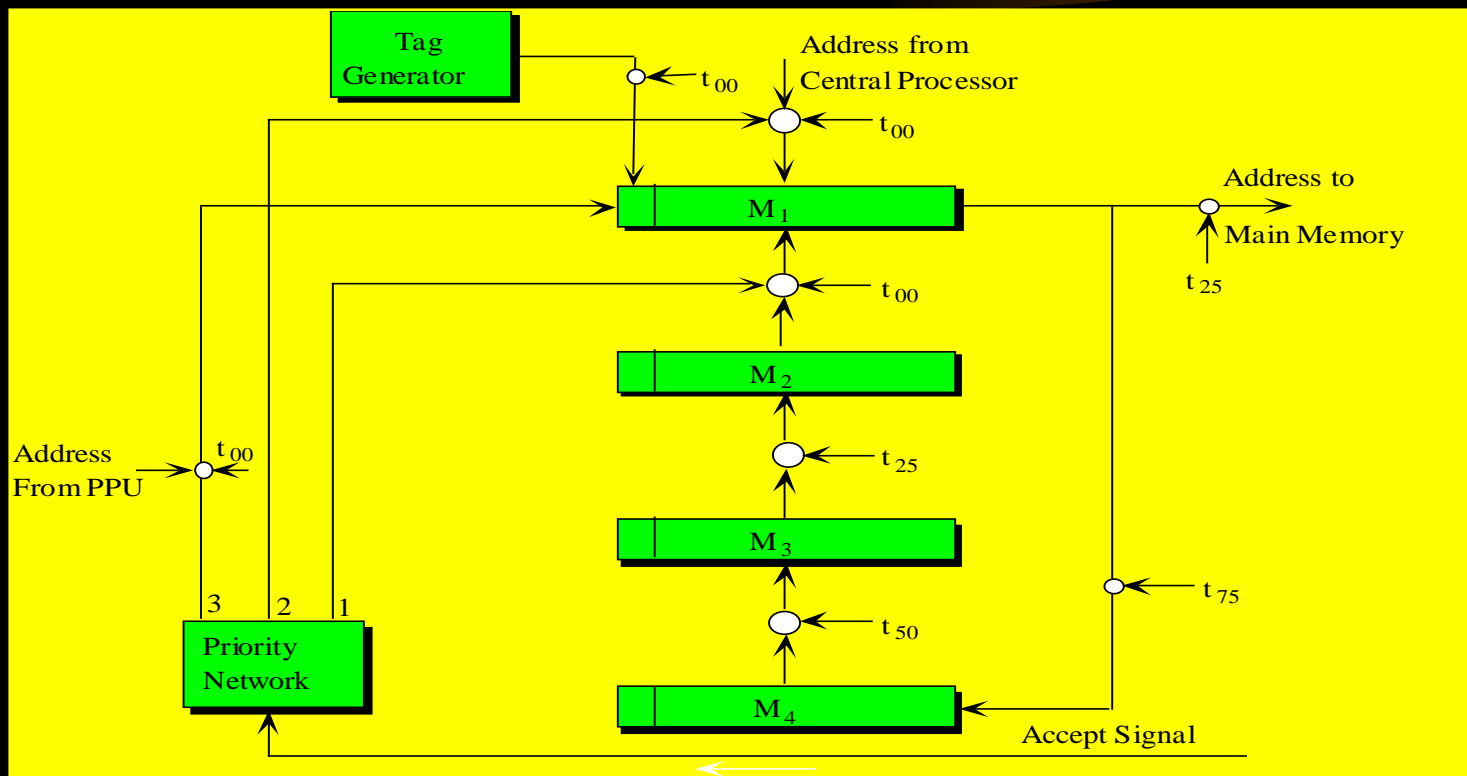
★ **Stunt Box** is composed of three parts:

- Hopper
- Priority Network
- Tag Generator and Distributor

◆ Interleaved Memory — **Stunt Box**

- ★ **Hopper** is an assembly of four registers to retain storage reference information until storage conflicts are resolved.
- ★ **Priority Network** prioritizes the requests to the memory generated by the central processor and the peripheral processors.
- ★ **Tag Generator** is used to control read/write conflict.

◆ Interleaved Memory — Stunt Box



◆ Stunt Box — Flow of Data and Control

- Assuming an empty hopper, a storage address from one of the sources is entered in register M_1 .
- The access request in M_1 is issued to the main memory.
- The contents of the registers in hopper are circulated every 75 nano seconds.
- If a request is accepted by the main memory, it will not be re-circulated back to the M_1 . Otherwise, after each 300 nano seconds it will be sent back to the main memory for a possible access.

◆ **Stunt Box** — Flow of Data and Control

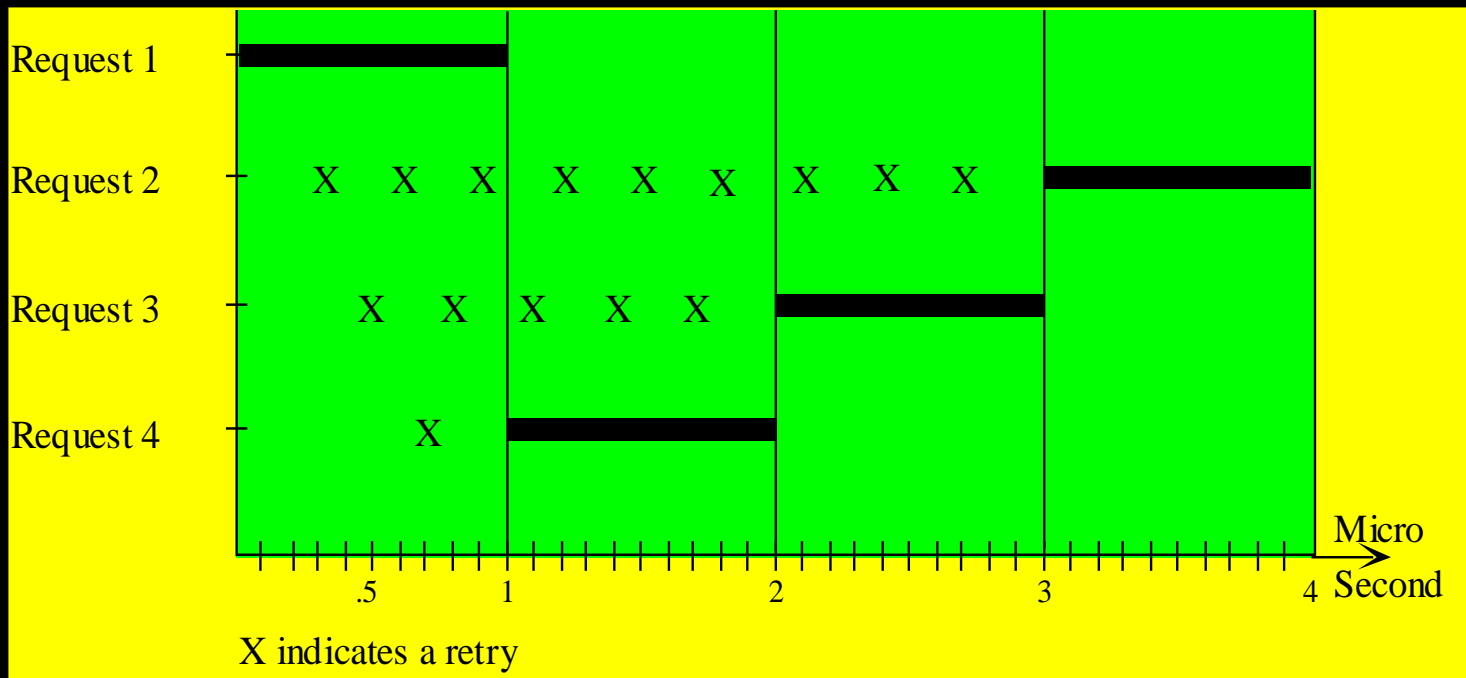
★ Time events of a request:

- t_{00} - Enter M_1
- t_{25} - Send to the central storage
- t_{75} - M_1 to M_4
- t_{150} - M_4 to M_3
- T_{225} - M_3 to M_2
- t_{300} - M_2 to M_1 (if not accepted)

Introduction to High Performance Computer Architecture

◆ Stunt Box — Example

- ★ Assume a sequence of access requests is initiated to the same memory module:





◆ **Stunt Box** — Example

★ The previous chart indicated;

- Access out-of-order,
- A request to the memory, sooner or later, will be granted.

◆ Memory System — Cache Memory

★ Principle of Locality

- Analysis of a large number of typical programs has shown that most of their execution time is spent in a few main routines.
- As a result, a number of instructions are executed repeatedly. This maybe in the form of a single loop, nested loops, or a few subroutines that repeatedly call each other.

◆ Memory System — Cache Memory

★ Principle of Locality

- It has been observed that a program spends 90% of its execution time in only 10% of the code — **principle of locality**.
- The main observation is that many instructions in each of a few **localized areas** of the program are repeatedly executed, while the remainder of the program is accessed relatively infrequently.

◆ **Memory System — Cache Memory**

★ **Principle of Locality** — locality can be represented in two forms:

- **Temporal Locality**: If an item is referenced, it will tend to be referenced again soon.
- **Spatial Locality**: If an item is referenced, nearby items tend to be referenced soon.



◆ Memory System — Cache Memory

★ Principle of Locality

- Now, if it can be arranged to have the **active segments** of a program in a fast memory, then the total execution time can be significantly reduced.
- Such a fast memory is called a **cache** (slave, buffer) memory.

◆ Memory System — Cache Memory

★ Principle of Locality

- Cache is a level of memory inserted between the main memory and the CPU.
- Due to economical reasons, cache is relatively much smaller than main memory.
- To make the cache effective, it must be considerably faster than the main memory.



◆ Memory System — Cache Memory

★ Principle of Locality

- The main memory and the cache are **partitioned** into blocks of equal sizes.
- Naturally, because of the size gap between the main memory and the cache at each moment of time a portion of the main memory is resident in the cache.

◆ Memory System — Cache Memory



Main Memory



Cache

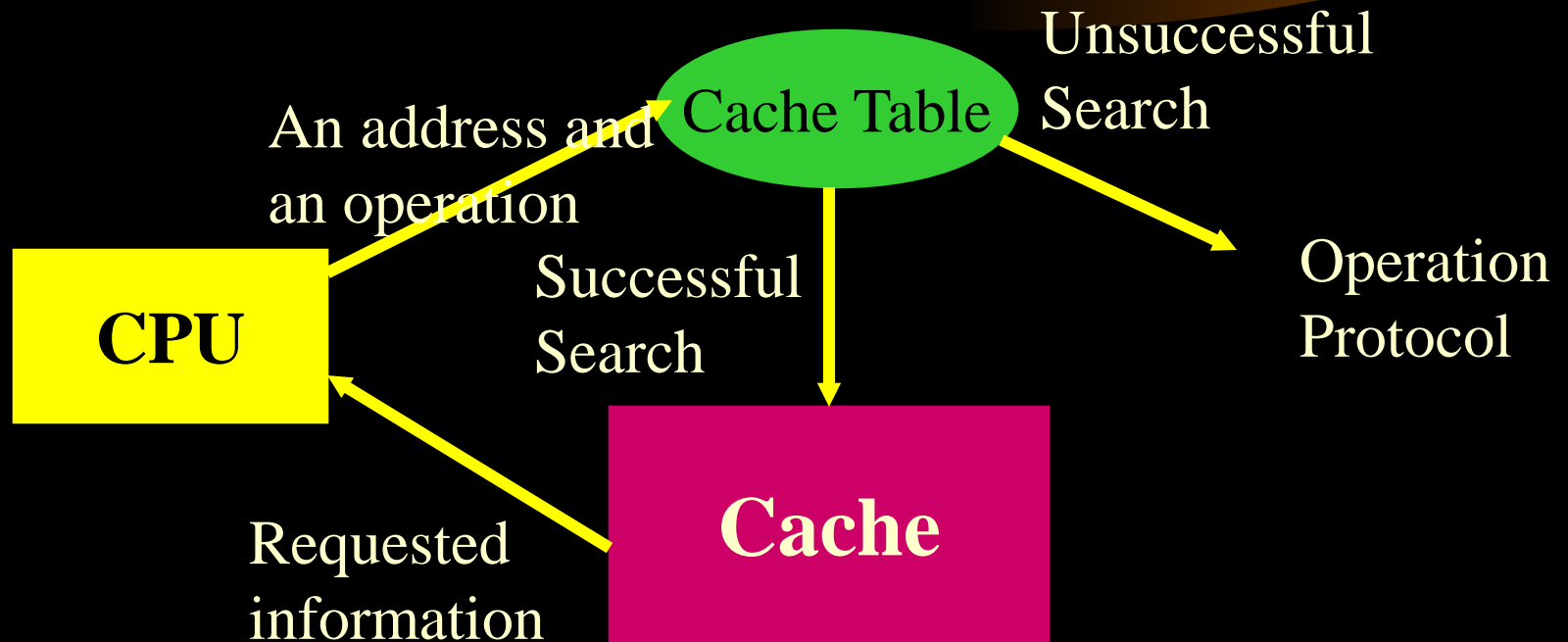


◆ Memory System — Cache Memory

★ Address Mapping

- Each reference to a memory word is presented to the cache.
- The cache searches its directory:
 - If the item is in the cache, then it will be accessed from the cache.
 - Otherwise, a miss occurs.

Introduction to High Performance Computer Architecture



◆ Memory System — Cache Memory

- ★ The concept of the cache was introduced in mid 1960s by Wilkes.
- ★ When a memory request is generated, it is first presented to the cache memory, and if the cache cannot respond, the request is then presented to the main memory.

◆ Memory System — Cache Memory

- ★ The idea of cache is similar to virtual memory in that some active portion of a low-speed memory is stored in duplicate in a higher-speed memory.
- ★ The difference between cache and virtual memory is a matter of implementation, the two approaches are conceptually the same because they both rely on the correlation properties observed in sequences of address references.

◆ Memory System — Cache Memory

- ★ Cache implementations are totally different from virtual memory implementation because of the speed requirements of cache. If we assume that cache memory has an access time of one machine cycle, then main memory typically has an access time anywhere from 4 to 20 times longer, not 500 times larger for the delay due to a page fault in virtual memory.
- ★ In general caches are controlled by **hardware algorithms**.

◆ Memory System — Cache Memory

★ Cache vs. Virtual Memory

	Cache/ Main Memory	Main/ Secondary Memory
Access time ratios	$\approx 10/1$	$\approx 1000/1$
Memory Management System	Implemented in Hardware	Mainly Implemented in Software
Typical Block Size	Few Words	Hundreds of Words
Access of Processor to Second Level	Direct Access to Both Cache and Main Memory	All Accesses via Main Memory

◆ **Memory System — Cache Memory**

★ Ranges of parameters for cache

Block size	4-128 Bytes
Hit time	1-4 clock cycles
Miss penalty	8-32 clock cycles
access time	6-10 clock cycles
Transfer time	2-22 clock cycles
Miss ratio	1%-20%
Cache size	1KB-256KB

◆ Memory System — Cache Memory

★ Replacement Policy

- For each read operation that causes a cache miss, the item is retrieved from the main memory and copied into the cache. This forces some other item in cache to be identified and removed from the cache to make room for the new item (if cache is full).
- The collection of rules which allows such activities is referred to as the **Replacement Algorithm**.



◆ Memory System — Cache Memory

★ Replacement Policy

- The cache-replacement decision is critical, a good replacement algorithm, naturally, can yield somewhat higher performance than can a bad replacement algorithm.

◆ Memory System — Cache Memory

★ Let h be the probability of a cache hit — **hit ratio**

$$h = \frac{\text{\# of accesses responded by cache}}{\text{Total \# of accesses to the memory}}$$

and t_{cache} and t_{main} be the respective cycle times of cache and main memory then:

$$t_{\text{eff}} = t_{\text{cache}} + (1-h)t_{\text{main}}$$

★ $(1-h)$ is the probability of a miss — **miss ratio**.

◆ Cache Memory — Issues of Concern

★ Read Policy

- Load Through

★ Write policy (on hit)

- Write through
- Write back \Rightarrow dirty bit

★ Write policy (on miss)

- Write allocate
- No-write allocate

★ Placement/replacement policy

★ Address Mapping



◆ **Cache Memory** — Issues of Concern

★ In Case of Miss-Hit

- For read operation, the block containing the referenced address is moved to the cache.
- For write operation, the information is written directly into the main memory.



◆ Questions

- ★ Compare and contrast different write policies against each other.
- ★ In case of miss-hit, why are **read** and **write** operations treated differently?



◆ **Cache Memory** — Issues of Concern

★ **Sources for cache misses:**

- Compulsory — cold start misses
- Capacity
- Conflict — placement/replacement policy

◆ **Cache Memory** — Issues of Concern

- ★ It has been shown that increasing the **cache sizes** and/or **degree of associativity** will reduce cache miss ratio.
- ★ Naturally, **compulsory** miss ratios are independent of cache size.



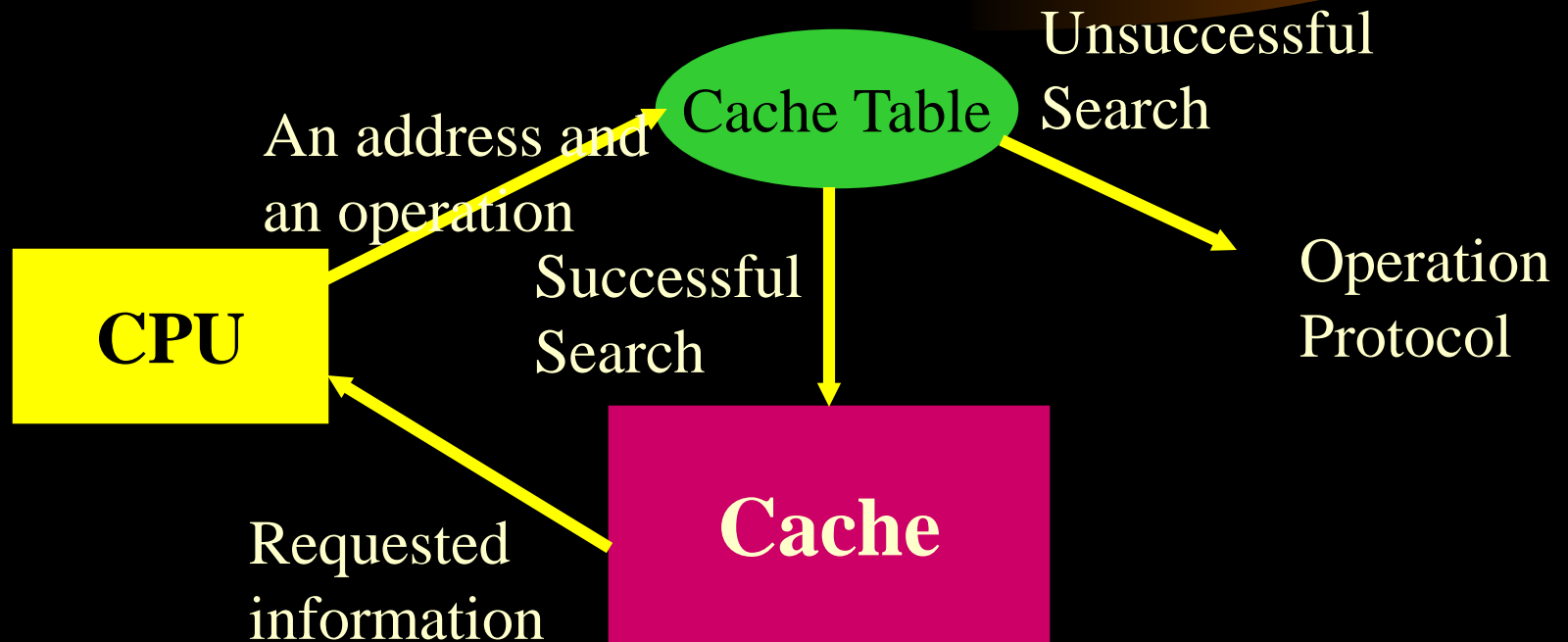
◆ Memory System — Cache Memory

- ★ **Mixed caches:** Cache contains both instruction and data — **Unified caches.**
- ★ **Instruction-only and Data-only caches:** Dedicated caches for instructions and data.

◆ Memory System — Cache Memory

- ★ In general, miss ratios for instruction caches are lower than miss ratios for data caches.
- ★ For smaller cache sizes, unified caches offer higher miss ratios than dedicated caches. However, as the cache size increases, miss ratio for unified caches relative to the dedicated caches reduces.

Introduction to High Performance Computer Architecture





◆ Memory System — Cache Memory

★ Address Mapping

- Direct Mapping
- Associative Mapping
- Set Associative Mapping

◆ **Cache Memory** — Address Mapping

★ In the following discussion assume:

- B = block size (2^b)
- C = number of blocks in cache (2^c)
- M = number of blocks in main memory (2^m)
- S = number of sets in cache (2^s)

◆ Cache Memory — Address Mapping

★ Direct Mapping

- Block K of main memory maps into block $(K \text{ modulo } C)$ of the cache.
- Since more than one main memory block is mapped into a given cache block, contention may arise even when the cache is not full.



◆ **Cache Memory** — Address Mapping

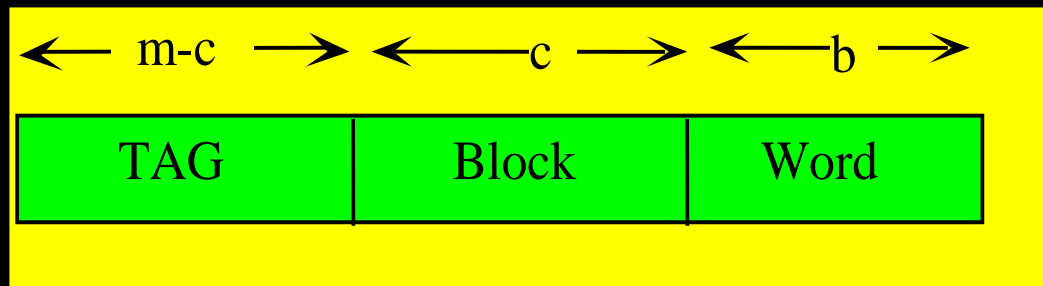
★ **Direct Mapping**

- Address mapping can be implemented very easily.
- Replacement policy is very simple and trivial.
- In general, cache utilization is low.

◆ Cache Memory — Address Mapping

★ Direct Mapping

- Main memory address is of the following form:



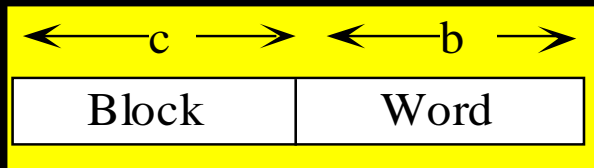
- A **Tag-register** of length $m-c$ is dedicated to each cache block

◆ Cache Memory — Address Mapping

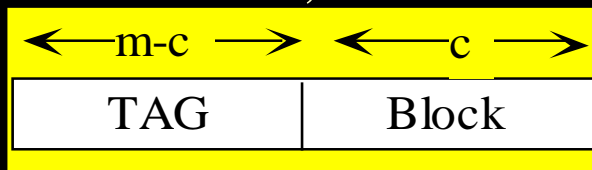
★ Direct Mapping

- Content of (tag-register)_c is compared against the tag portion of the address:

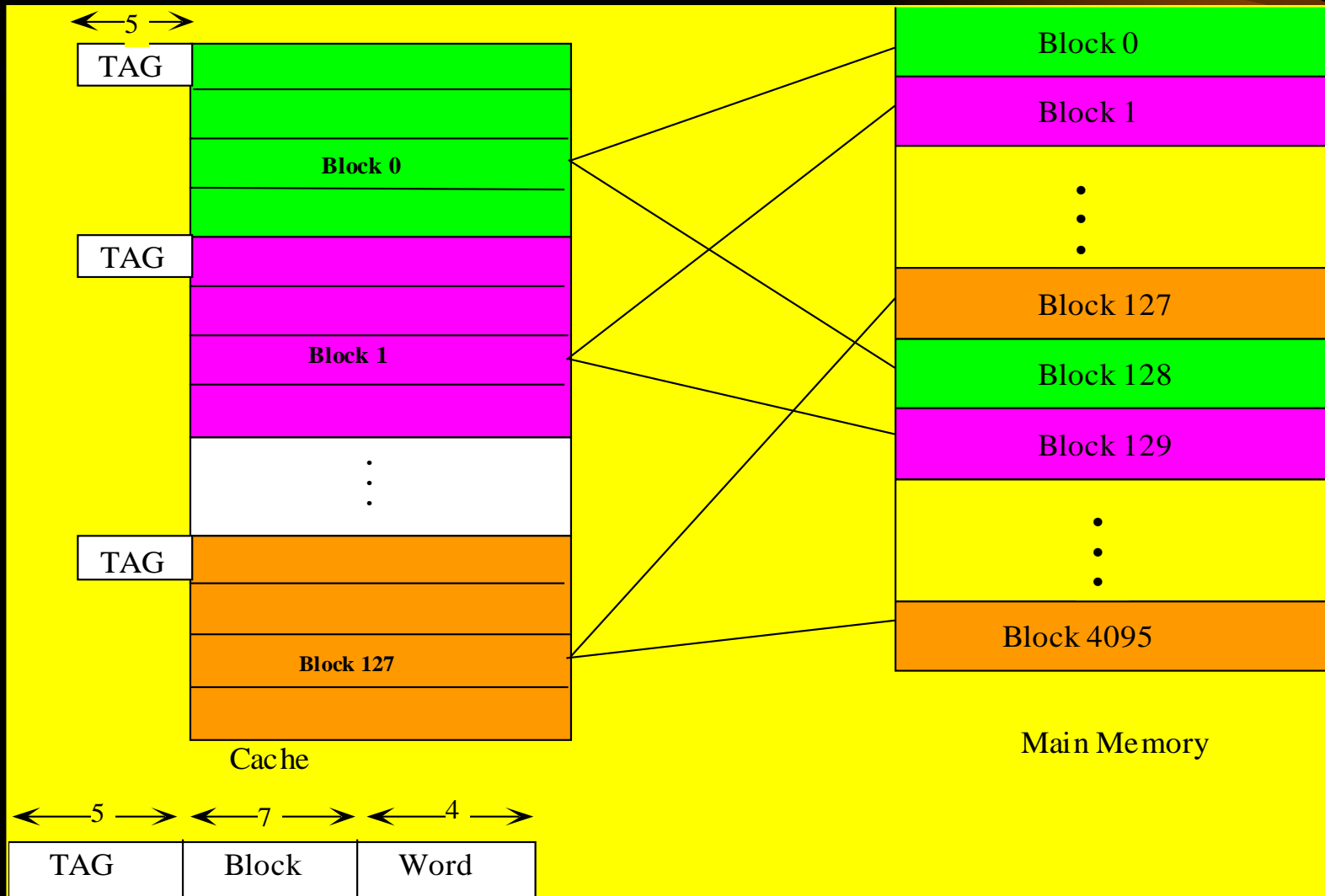
- If match then hit; and access information at address from the cache.



- If no-match, then miss-hit; bring block from main memory into block *c* of cache



Introduction to High Performance Computer Architecture



◆ Cache Memory — Address Mapping

★ Associative Mapping

- A block of main memory can potentially reside in any cache block. This flexibility can be achieved by utilizing a **wider Tag-Register**.
- Address mapping requires **hardware facility** to allow simultaneous search of tag-registers.
- A **reasonable replacement policy** can be adopted (least recently used).
- Cache can be used very effectively.

◆ **Cache Memory** — Address Mapping
★ **Associative Mapping**

- Main memory address is of the following form:

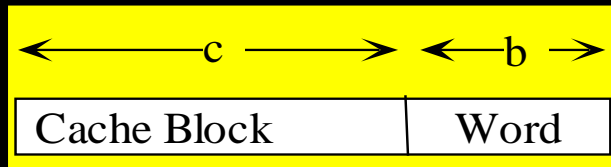


- A **tag-register** of length m is dedicated to each cache block.

◆ Cache Memory — Address Mapping

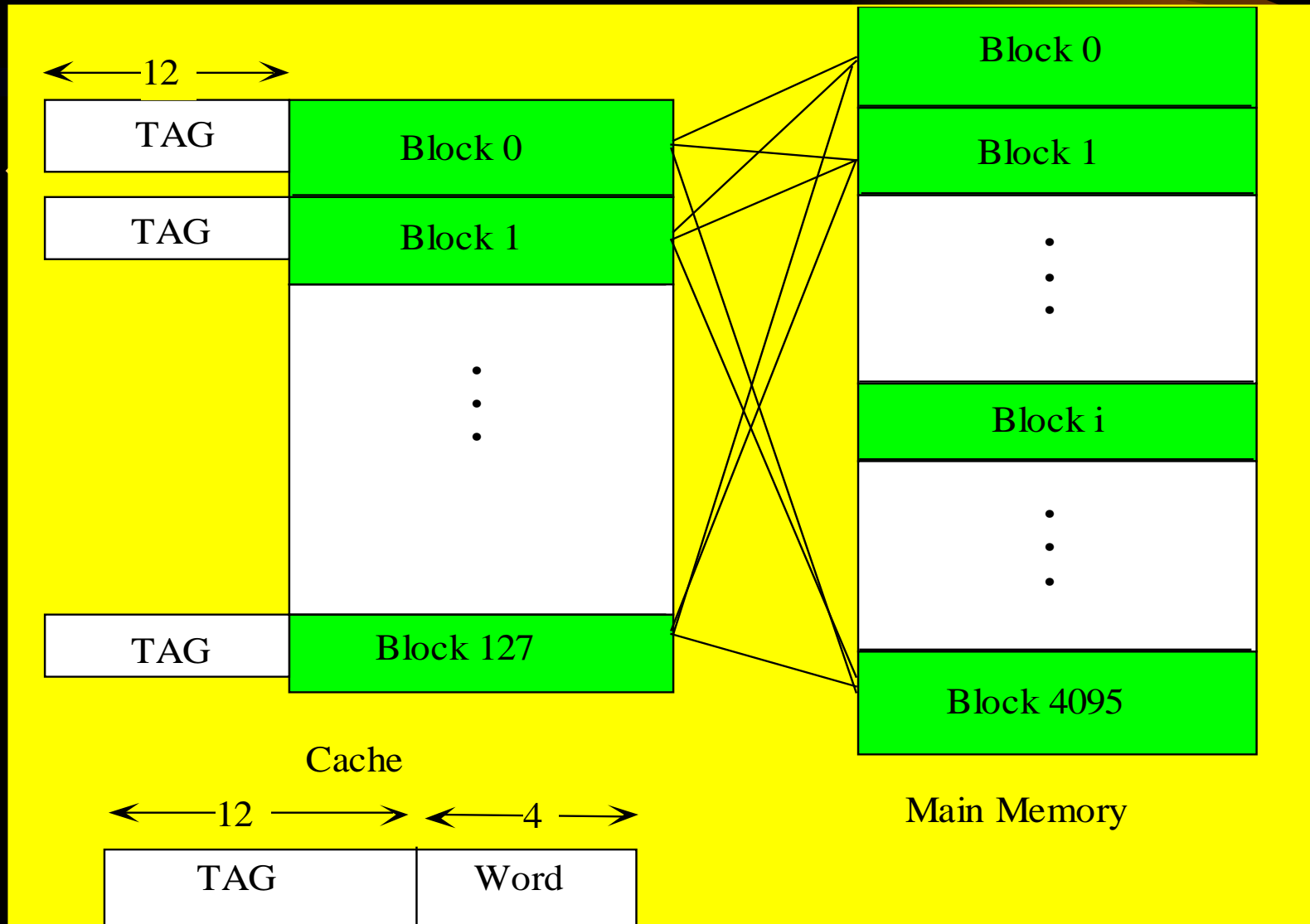
★ Associative Mapping

- Contents of Tag portion of the address is searched (in parallel) against the contents of the Tag-registers:
- If match, then hit; access information at address from the cache.



- If no-match, then miss-hit; bring block from memory into the **proper** cache block.

Introduction to High Performance Computer Architecture



◆ Cache Memory — Address Mapping

★ Set Associative Mapping

- Is a compromise between Direct-Mapping and Associative Mapping.
- Blocks of cache are grouped into **sets** (S), and the mapping allows a block of main memory (K) to reside in any block of the set ($K \bmod S$).
- Address mapping can be implemented easily at a more **reasonable hardware cost** relative to the associative mapping.

◆ **Cache Memory** — Address Mapping

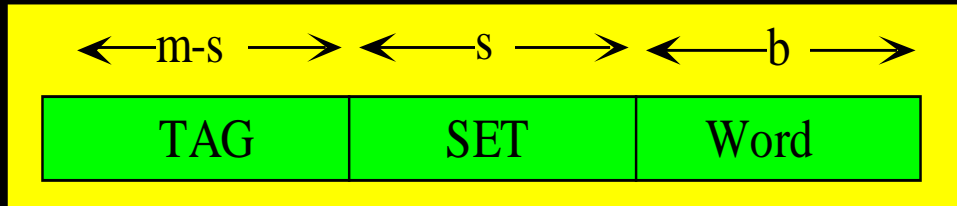
★ **Set Associative Mapping**

- This scheme allows one to employ a reasonable replacement policy within the blocks of a set and hence offers **better cache utilization** than the direct-mapping scheme.

◆ **Cache Memory** — Address Mapping

★ **Set Associative Mapping**

- Main memory address is of the following form:

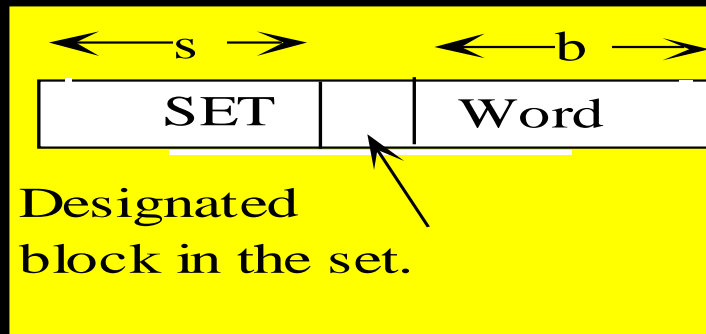


- A **tag-register** of length $m-s$ is dedicated to each block in the cache.

◆ Cache Memory — Address Mapping

★ Set Associative Mapping

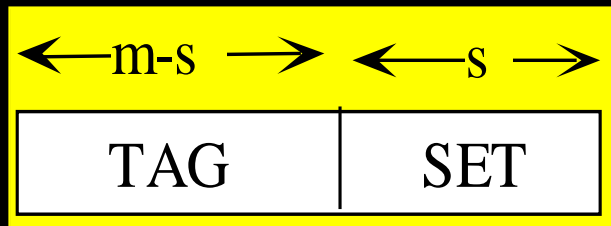
- Contents of Tag-registers_s are compared simultaneously against the tag portion of the address:
- If match, then hit; access information at address from the cache.



◆ **Cache Memory** — Address Mapping

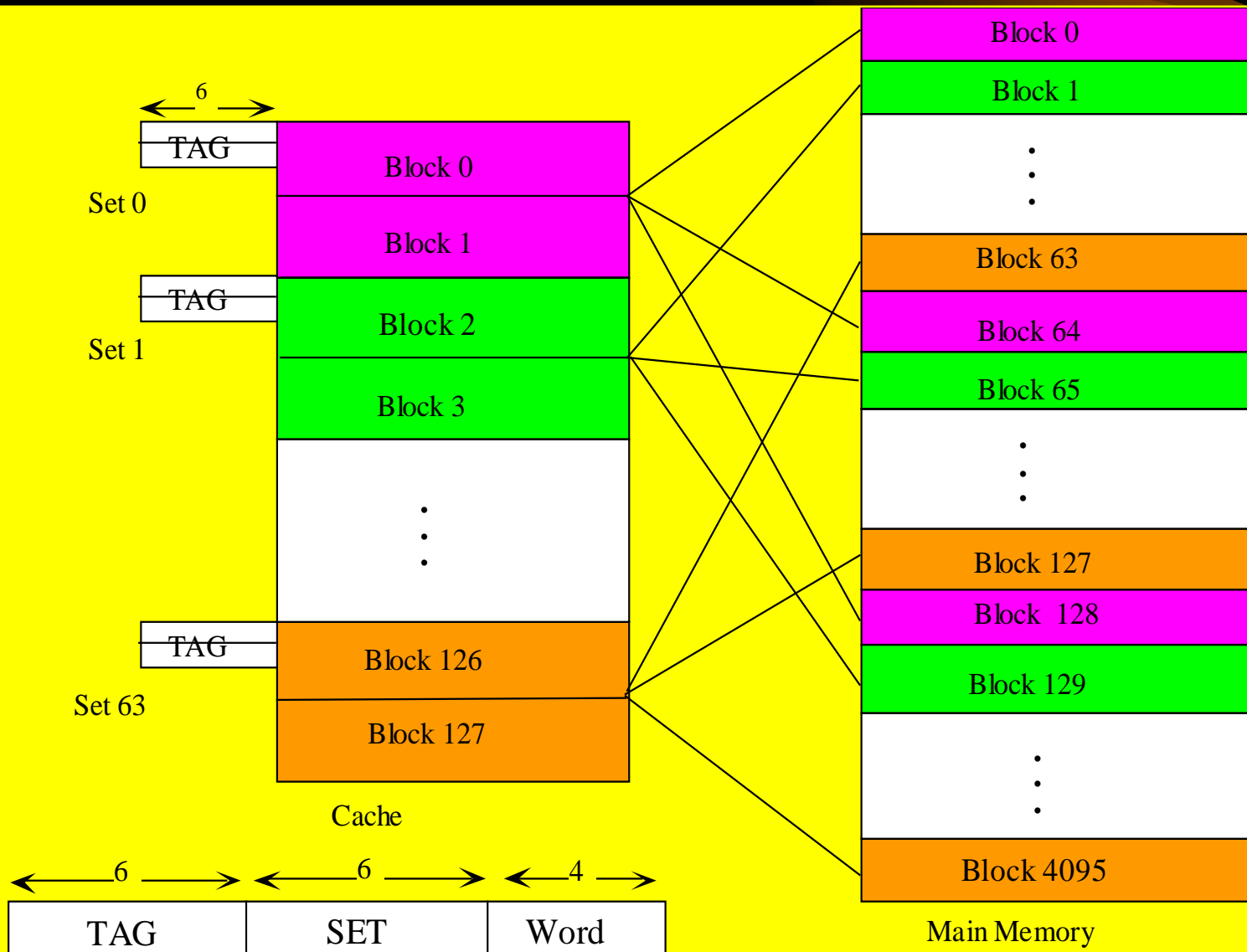
★ **Set Associative Mapping**

- If no-match, then miss-hit; bring block



from the main memory into the proper block of set s of the cache.

Introduction to High Performance Computer Architecture





◆ Questions

- ★ Compare and contrast unified cache against dedicated caches.
- ★ Compare and contrast Direct Mapping, Associative Mapping, and Set Associative Mapping against each other.

◆ **Cache Memory** — IBM 360/85

★ **Main Memory (4-way interleaved)**

- Size 512-4096 k bytes
- Cycle Time 1.04 μ sec
- Block Size 1 k bytes

◆ Cache Memory — IBM 360/85

★ Cache

- Size 16 k bytes
- Access Time 80 η sec
- Block Size 1 k bytes
- Address Mapping — Associative Mapping
- Replacement Policy — Least Recently Used
- Read Policy — Read-Through
- Write Policy — Write-Back, write access to the main memory does not cause any cache reassignment.

◆ Cache Memory — IBM 360/85

★ Hardware Configuration

- An associative memory of 16 words, each 14 bits long, represents the collection of the **tag-registers**.
- Each block is a collection of 16 **units** each of length 64 bytes.
- Each block has a **validity register** of length 16.



◆ **Cache Memory** — IBM 360/85

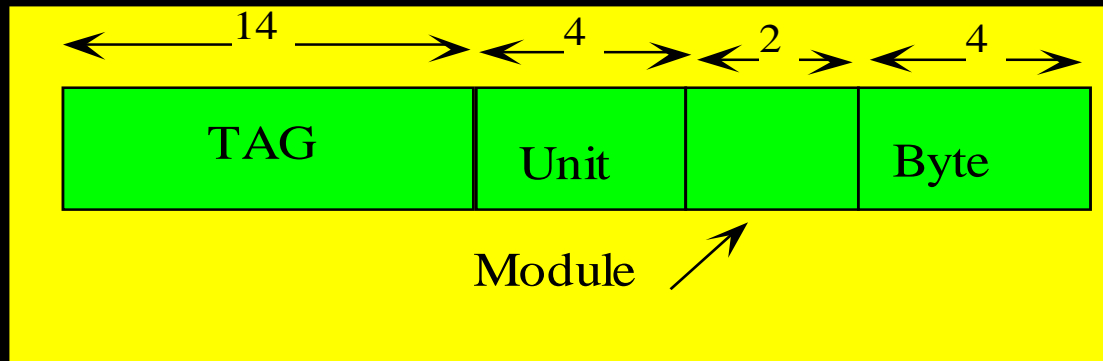
★ **Hardware Configuration**

- A **validity bit** represents the availability of a unit within a block in the cache.
- A **unit** is the smallest granule of information which is transferred between the main memory and the cache.
- The units in a block are brought in on a **demand basis**.

Introduction to High Performance Computer Architecture

◆ Cache Memory — IBM 360/85

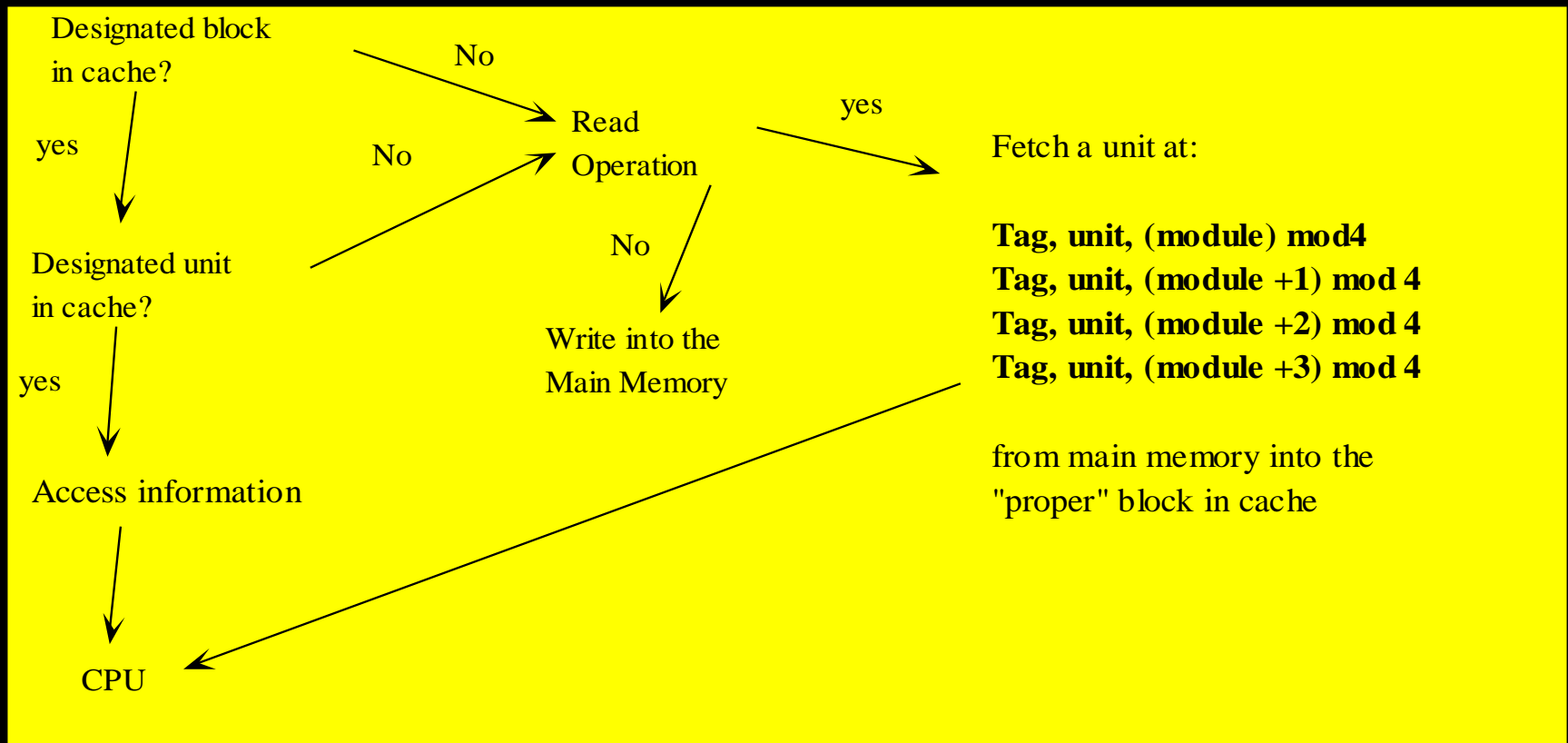
★ Main memory address format:



Introduction to High Performance Computer Architecture

◆ Cache Memory — IBM 360/85

★ Flow of Operations



◆ **Cache Memory** — IBM 370/155

★ **Main Memory (4-way interleaved)**

- Size 256 - 2048 k bytes
- Cycle time 2.100 μ sec
- Block size 32 bytes

◆ Cache Memory — IBM 370/155

★ Cache

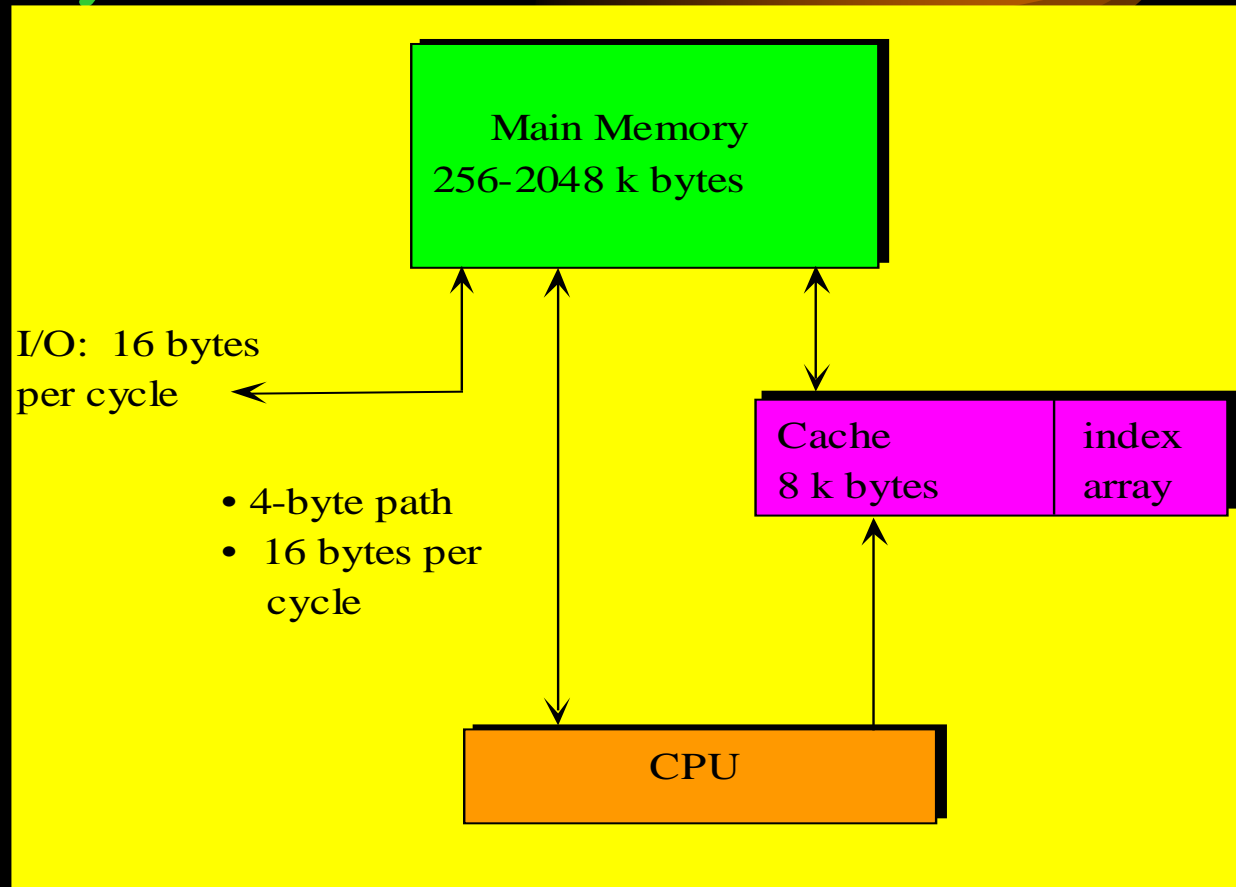
- Size 8 k bytes
- Cycle time 230 η sec
- Block size 32 bytes

★ Address Mapping

- Set associative mapping
- Set-size – 2

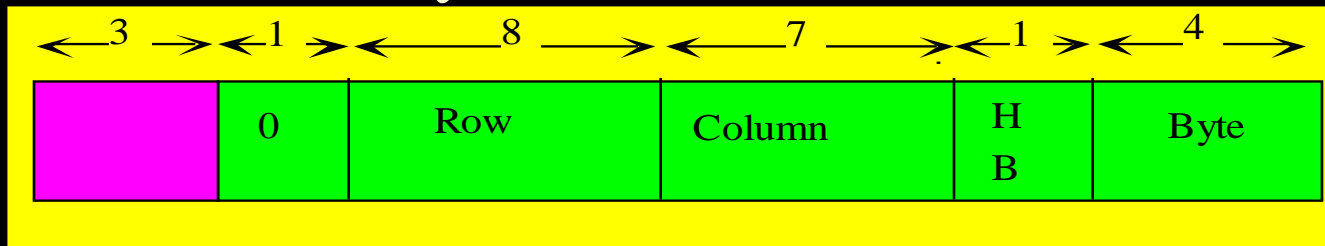
◆ Cache Memory — IBM 370/155

Hardware Configuration

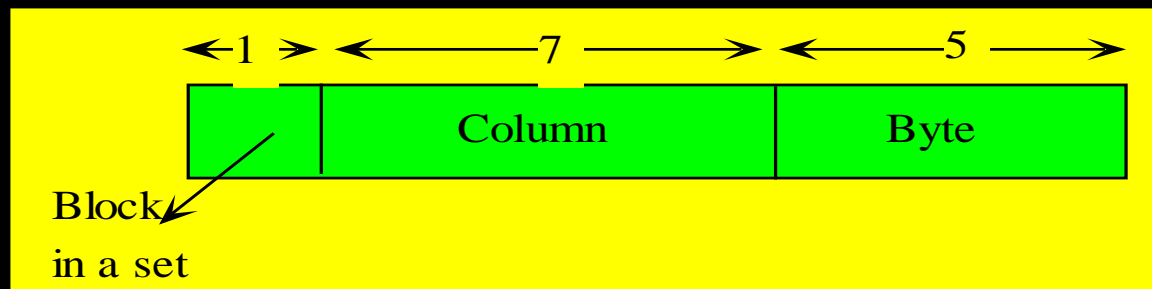


◆ Cache Memory — IBM 370/155

★ Main memory address format:



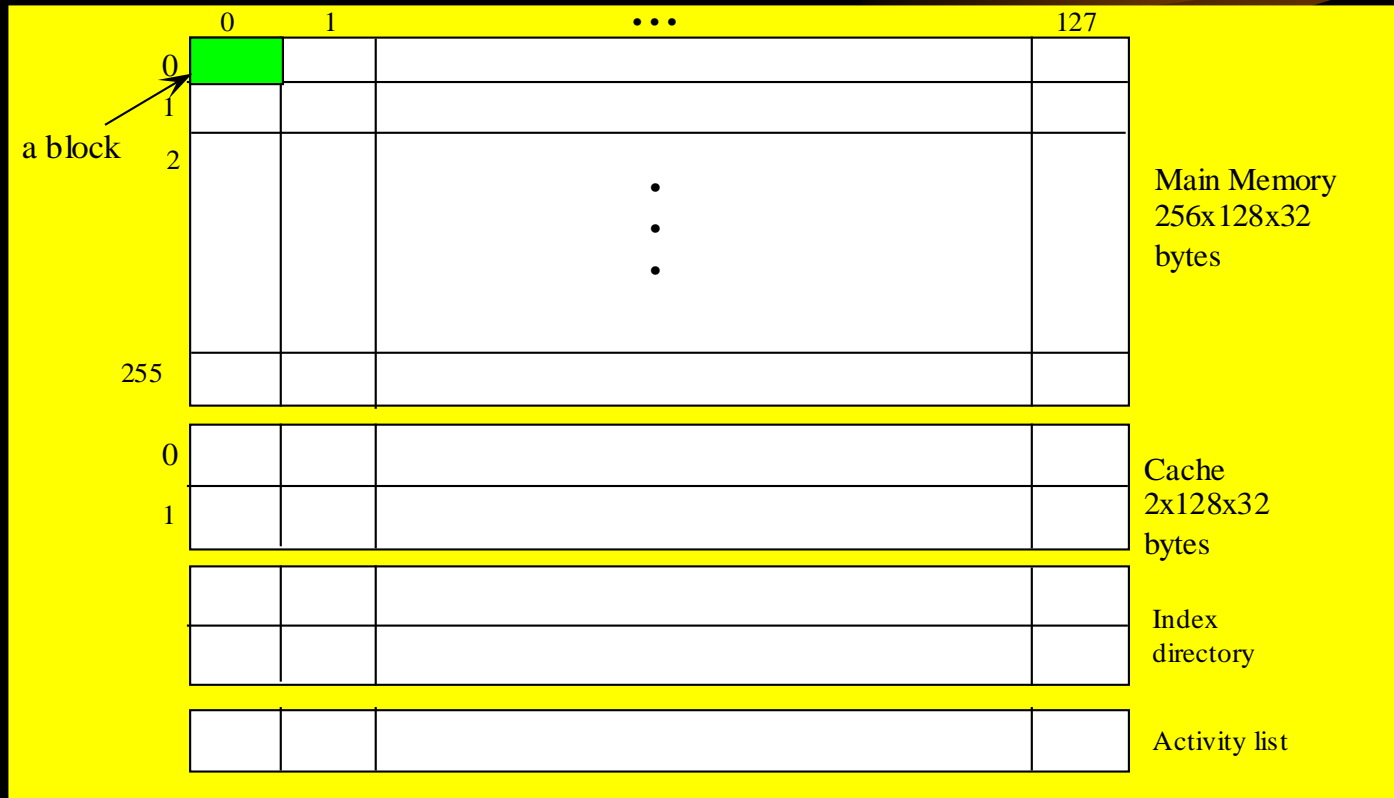
★ Cache address format:



Introduction to High Performance Computer Architecture

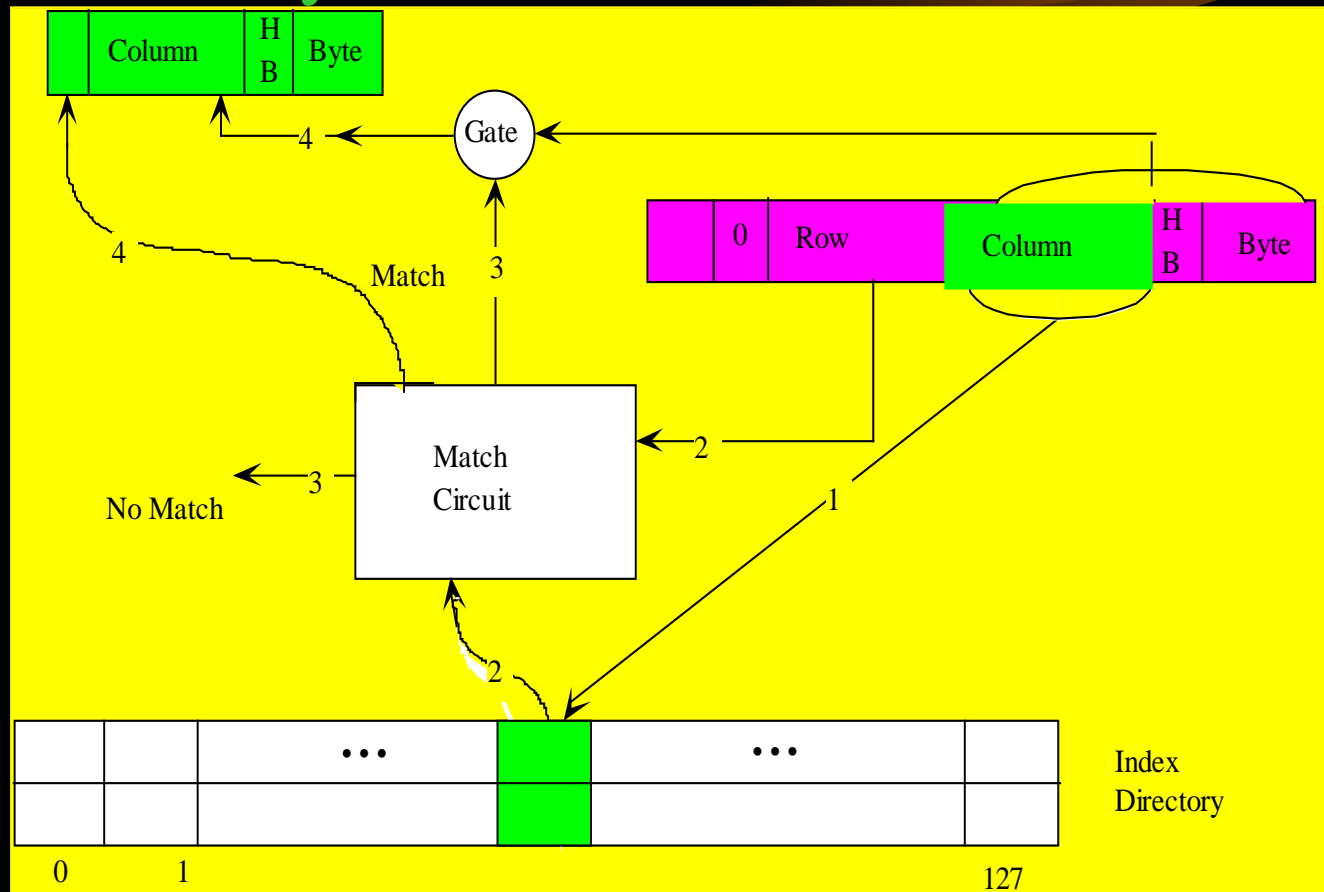
◆ Cache Memory — IBM 370/155

★ Memory Organization and View



Introduction to High Performance Computer Architecture

◆ Cache Memory — IBM 370/155



◆ Cache Memory — 68040

- ★ Two dedicated caches on processor chip
 - Instruction cache
 - Data cache
- ★ Each cache is of size 4K bytes with 4-way set associative organization.
- ★ Each cache is a collection of 64 sets of 4 blocks each.

◆ **Cache Memory** — 68040

- ★ Each block is a collection of 4 long words — a long word is 4 bytes long.
- ★ Each cache block has a **valid bit** and a **dirty bit**.
- ★ Either **write back** or **write through** policy can be employed.
- ★ It uses a randomly selected block in each set as a **replacement policy**.

◆ Cache Memory — 68040

★ Main Memory Address Format:

Tag	Set#	Byte#
22 bits	6 bits	4 bits

◆ Cache Memory — Pentium III

★ Has two cache Levels:

● Level1

- Has **dedicated caches** for instructions and data.
- Each cache is 16K bytes.
- **Data cache** is 4-way set associative organization.
- **Instruction cache** is 2-way set associative organization.
- Both **write back** and **write through** policies can be adopted.

◆ Cache Memory — Pentium III

★ Level2

- It is a **unified cache**, either external to the processor chip (Pentium III — Katmai) or internal to the processor chip (Pentium III — Coppermine).
- If **internal**, it is of size 256 Kbytes SRAM, 4-way set associative organization.
- If **external**, it is of size 512 Kbytes, 8-way set associative organization.
- Either **write back** or **write through** policy can be employed.

◆ Cache Memory

★ How to make cache faster?

- Make the cache faster — Better **technology**,
- Make the cache larger,
- **Sub block** cache blocks — A portion of a block is the granule of information transferred between the main memory and cache,
- Use a **write buffer** — Care should be taken for write-read order.

◆ Cache Memory

★ How to make cache faster?

- Early restart — Allow the CPU to continue as soon as the requested data is in cache (**read through**),
- Out-of-order fetch — Attempt to fetch the requested information first should be used in conjunction with read through.
- Multi-level cache memory organization.

◆ Cache Memory

★ Two-Level Cache Organization

- As the name suggests, the cache is a hierarchy of two levels:
 - Level 1
 - Level 2
- Average memory-access time:

$$\text{Hit time}_{L1} + \text{Miss ratio}_{L1} * (\text{Hit time}_{L2} + \text{Miss ratio}_{L2} * \text{Miss penalty}_{L2})$$

- ◆ **Cache Memory** — Two-Level Cache Organization
 - ★ Parameters for Level 2 Cache memory

Block size	32-256 Bytes
Hit time	4-10 clock cycles
Miss penalty	30-80 clock cycles
access time	14-18 clock cycles
transfer time	16-62 clock cycles
Local miss ratio	15%-30%
Cache size	256KB-4MB

◆ Cache Memory — Cache Coherency

- ★ Cache coherency problem refers to the status where multiple copies of the same information block contain different data values.
- ★ Cache coherency occurs when the same information is shared among different resources that can perform Read/Write operations on them — I/O devices and the CPU, Multiprocessor systems with private caches.



◆ **Cache Memory** — Cache Coherency

★ I/O Operations

- In Case of **Output operation, write through** policy for cache resolves the possibility of cache coherency — Always the same information is maintained in cache and main memory.

◆ Cache Memory — Cache Coherency

★ I/O Operations

- In Case of **input operation**, we should guarantee that no blocks of the I/O buffer designated for input are the cache resident — **non-cacheable blocks**.
 - **Flush** the buffer address for the cache before initiating any input operation — This will be done by the operating system (**software solution**).
 - **Invalidate** cache entries with the same I/O addresses — **Hardware solution**.



◆ **Cache Memory** — Cache Coherency

★ **Multiprocessor Systems**

- The coherency problem arises for a processor with exclusive write access to an information that might have several copies in other private caches.



◆ **Cache Memory** — Cache Coherency

★ **Multiprocessor Systems**

- **In case of write**, the **coherency protocol** should locate all the caches that have a copy of the information and either:
 - **Invalidate** all other copies, or
 - **Broadcast** the write to the shared information.
- **In case of read miss**, the coherency protocol must transfer the **most up-to-date** copy in.

◆ **Cache Memory** — Cache Coherency

★ **Multiprocessor Systems**

- **Directory Based System** — The information about blocks of main memory resident in caches is kept in just one location (an entry for each physical block).
- **Snoopy System** — Every cache that has a copy of a physical block also has a copy of the information about it (coherency information is proportional to the number of blocks in cache).



◆ **Cache Memory** — Cache Coherency

★ **Snoopy Systems**

- **In case of read miss**, all caches check to see if they have a copy of the requested block and supply a copy to the requesting cache.



◆ **Cache Memory** — Cache Coherency

★ **Snoopy Systems**

- **In case of write**, all caches check to see if they have a copy of the block involved, then either:
 - **Write invalidate**, shared copies are invalidated and local copy is updated.
 - **Write broadcast**, writing processor broadcasts the new data and all copies are updated with the new value.

◆ Memory System — Virtual Memory

- ★ Recall from our earlier discussion that within the scope of memory organization we are concern about two issues:
 - Access gap, and
 - Size gap.
- ★ Access gap issue was discussed in detail. In this section we will concentrate on the size gap issue.

◆ Memory System — Virtual Memory

- ★ Since the earliest days of the computer, it was recognized that due to many factors, memories must be organized in a **hierarchical fashion**.
 - Because of such an organization, the **memory allocation issue** becomes important.
 - **Memory allocation** is the ability to distribute information among the levels of the memory system.

◆ Memory System — Virtual Memory

- ★ The **memory allocation issue** even became more important after the introduction of **high level Programming Languages** (say why?).
- ★ Along with the introduction of high level programming languages, two approaches were proposed for the memory allocation issue:
 - **Static**
 - **Dynamic.**

◆ Memory System — Virtual Memory

★ Both **static** and **dynamic** memory allocations differ on their assumptions about the prediction of:

- availability of **memory resources**, and
- creation of the program's **reference string** (execution thread).

◆ Memory System — Virtual Memory

- ★ **Static approach** assumes that *memory resources* are given or can be pre-specified and *reference string* can be determined during the compilation time.
- ★ **Dynamic approach** assumes that *memory resources* can not be pre-specified and *reference string* can only be determined during the execution time.

◆ Memory System — Virtual Memory

- ★ Concepts such as **machine independence**, **modularity**, and **list processing** revealed that the static approach is not a suitable solution.

◆ Memory System — Virtual Memory

★ Dynamic Allocation Objectives:

- The ability to store program into a **space of arbitrary size**.
- The ability to run a **partially loaded program**.
- The ability to **relocate a program**.
- The ability to change system resources without having to reprogram or recompile.
- The ability to vary the amount of space in use by a given program.

◆ Memory System — Virtual Memory

★ Even in the arena of dynamic allocation, there were two ideas about who should handle the memory allocation:

- Programmer,
- Automatic storage allocation, influenced by multiprogramming and time-sharing concepts.

◆ Memory System — Virtual Memory

- ★ Concept of the virtual memory grew out of the automatic storage allocation policy, where programmer has the **illusion** of having a very large main memory.

◆ Memory System — Virtual Memory

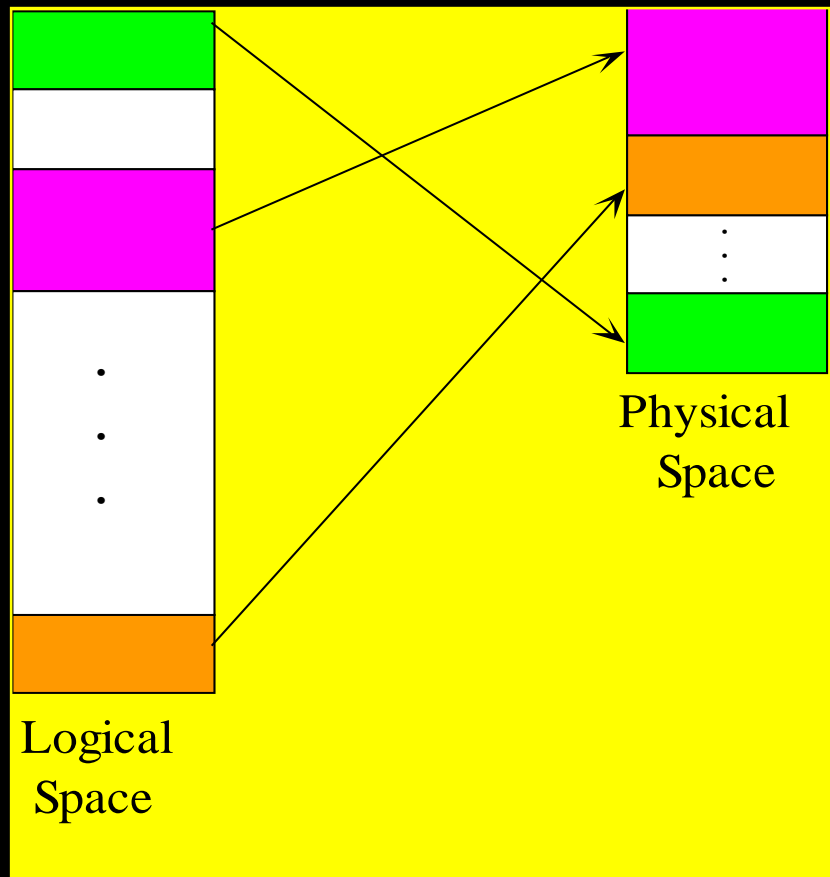
★ However, this new concept created some problems:

- Execution time overhead,
- Space overhead,
- Severe cost overhead in time sharing system,
- Thrashing in multiprogramming system.

◆ Memory System — Virtual Memory

- ★ Within the concept of the virtual memory one should distinguished between the **physical space** and **logical space**.
- ★ Memory allocation is a **mapping** from logical space to physical space.
- ★ This mapping function is called the **address translation** which translates programmer addresses to physical addresses.

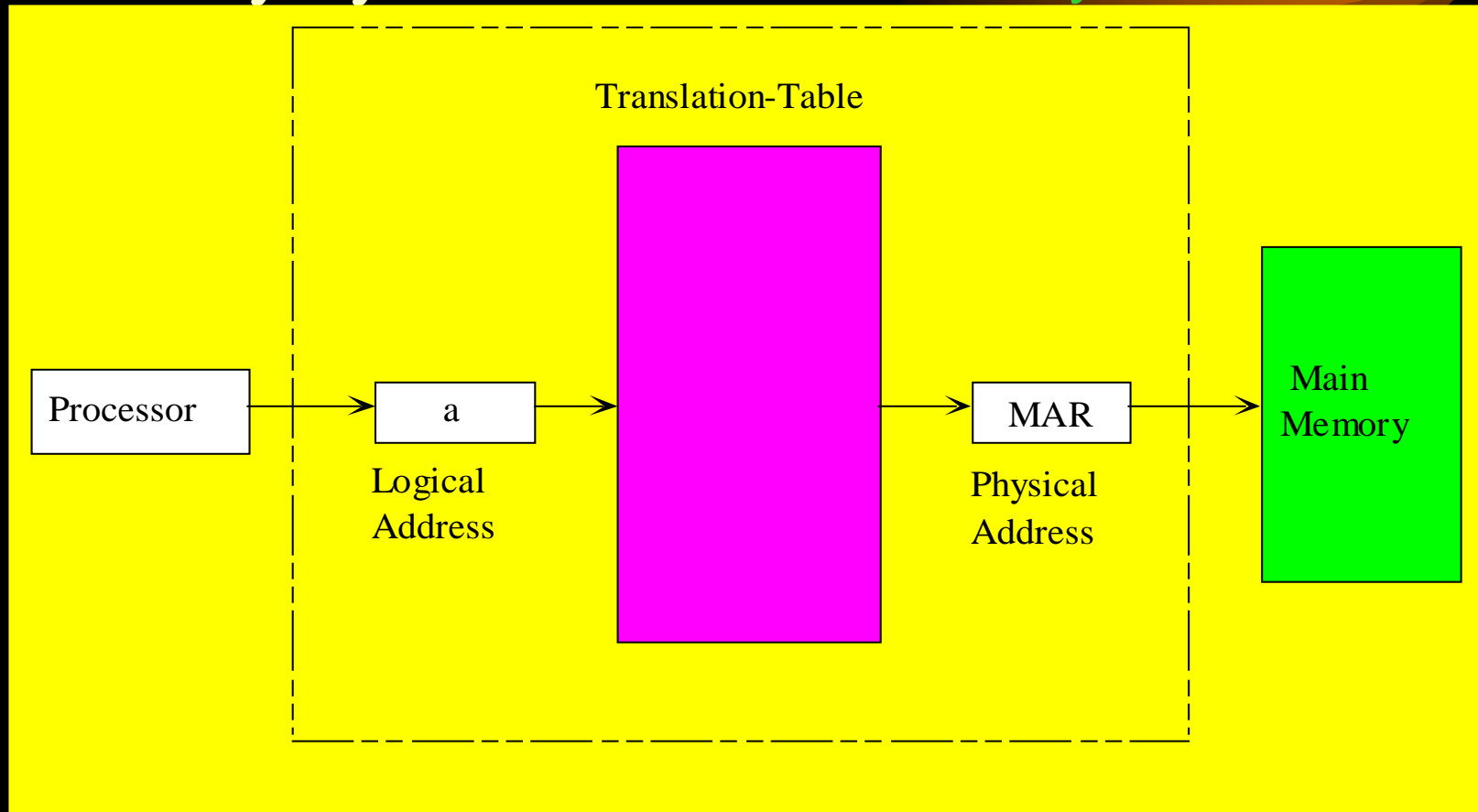
◆ Memory System — Virtual Memory



◆ Memory System — Virtual Memory

- ★ Implementation of the translation function implies the existence of a **table** which holds the virtual (logical) and memory (physical) addresses.
- ★ Due to the sheer size of the mapping table, it would be impractical if virtual memory has to map each word.
- ★ A practical solution is to **group** information into **blocks** and have just one entry for each block in the mapping table.

◆ **Memory System — Virtual Memory**



◆ **Memory System — Virtual Memory**

★ **Ranges of parameters for virtual memory**

Block(page) size	512-8192 Bytes
Hit time	1-10 clock cycles
Miss penalty	10^5-$6 \cdot 10^5$ clock cycles
access time	10^5-$5 \cdot 10^5$ clock cycles
transfer time	10^4-10^5 clock cycles
Miss ratio	10-5%-10-3%
Main memory size	4MB-2048MB



◆ Virtual Memory

- ★ The literature has addressed two approaches for handling virtual memory:
 - Segmentation
 - Paging

◆ Virtual Memory — Segmentation

- ★ Logical space is grouped into blocks of **varying sizes** (e.g., segments).
- ★ Segmentation is in response to the requirements of some applications which need to **group information** based on the **contents**.

◆ Virtual Memory — Segmentation

- ★ Each entry in the **segment table** has two parts:
 - The beginning address of the segment (A), and
 - Its length (b).
- ★ If a segment is not in the physical space, its corresponding entry in the segment table is empty.

◆ Virtual Memory — Segmentation

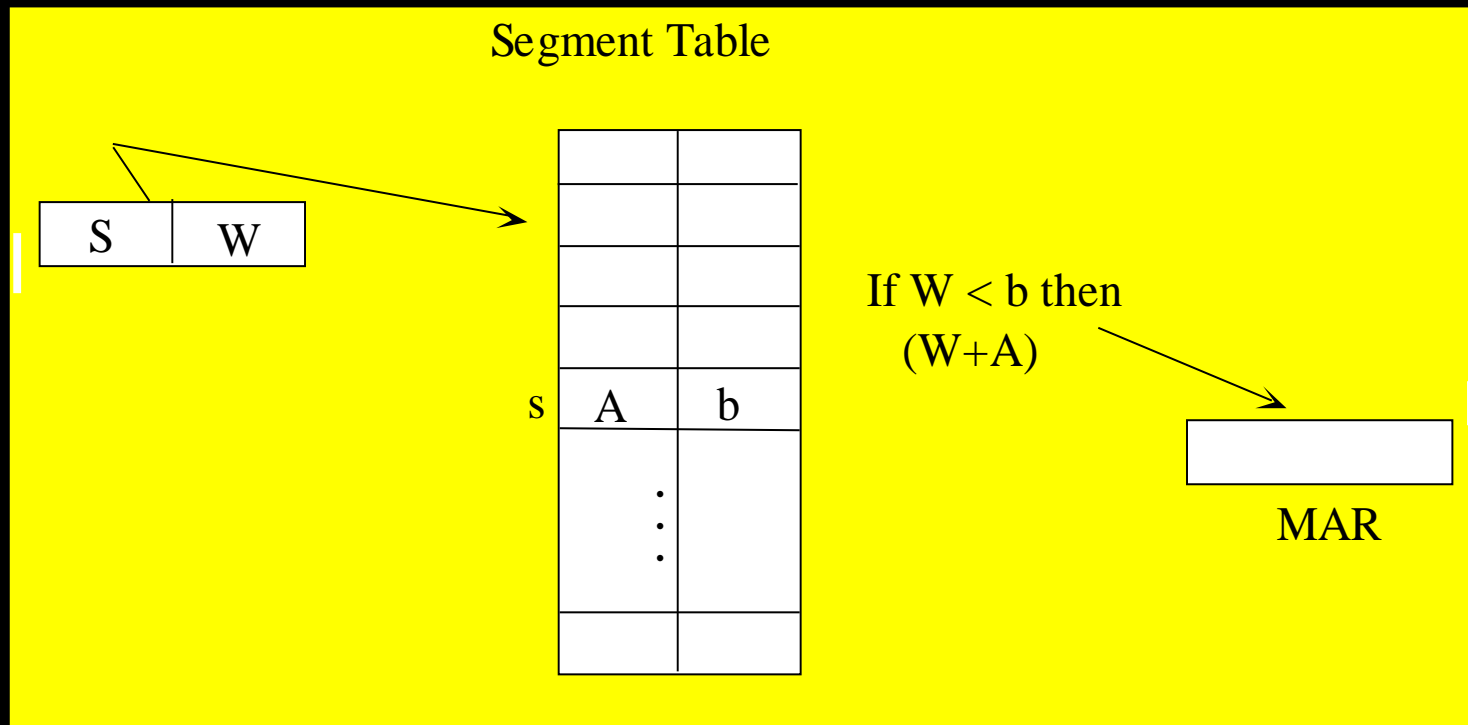
- ★ Each logical address has also two parts:
 - A segment number (S), and
 - A relative location to the beginning of the segment (W).

◆ Virtual Memory — Segmentation

★ Address Translation

- For each logical address (S, W):
 - If the S^{th} entry of the segment table is empty, then a **segment fault**.
 - If $W > b$ then **overflow fault**.
- $(A + W)$ is loaded into MAR.

◆ Virtual Memory — Segmentation



◆ Virtual Memory — Paging

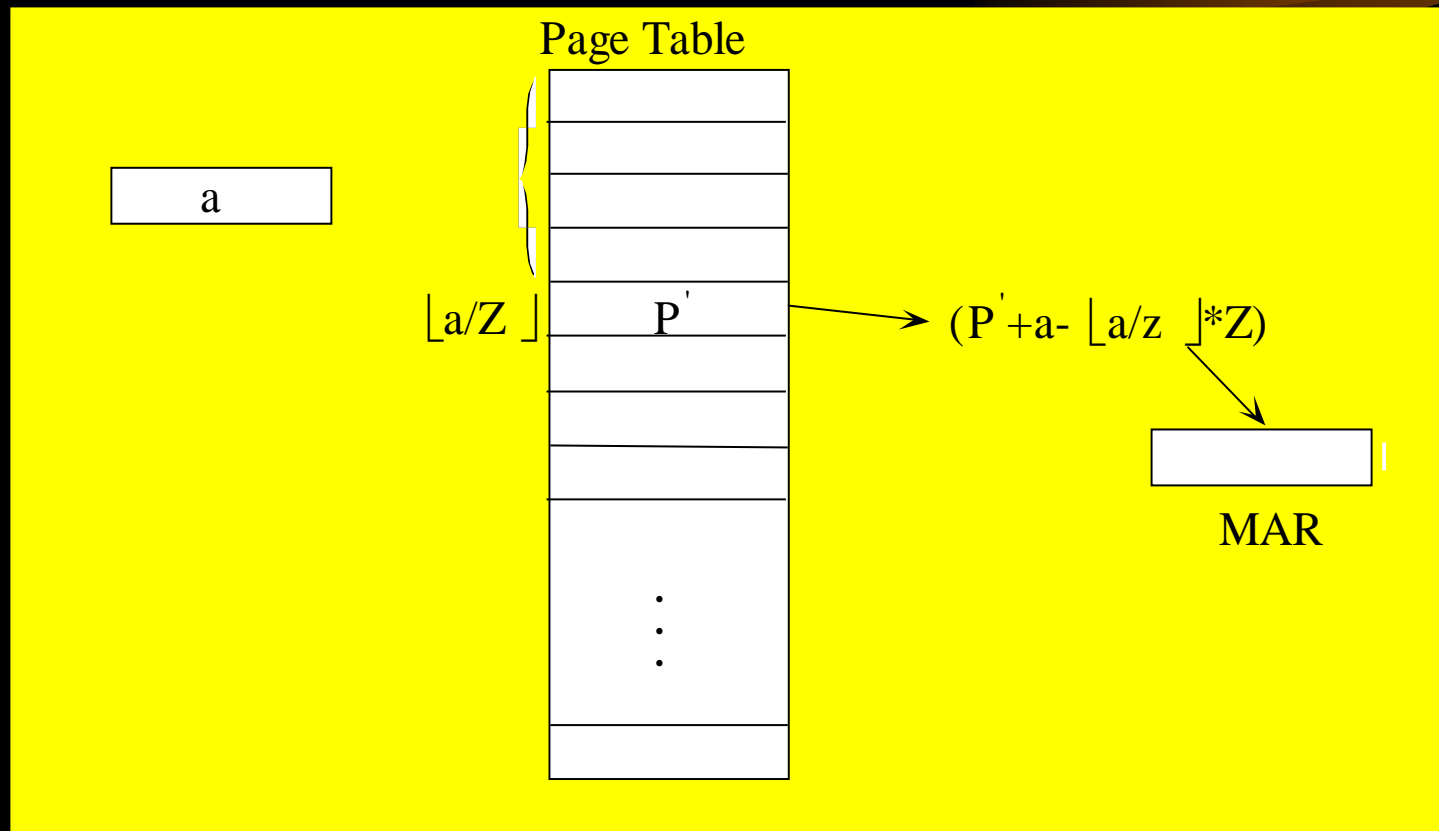
- ★ Logical and physical space are organized into **equal size blocks** (e.g., pages).
- ★ Each entry in the **page table** has one entry (P') — the beginning address of the page in the physical address.
- ★ If a page is not in the main memory its corresponding entry in the page table is empty.
- ★ Each logical address is also one entry (a).

◆ Virtual Memory — Paging

★ Address Translation

- For each logical address (a) - assume Z is the page size:
 - If the P^{th} entry ($P = \lfloor a/z \rfloor$) of page table empty then a **page fault**.
- ($P' + a - PZ$) is loaded into MAR.

◆ Virtual Memory — Paging



◆ Virtual Memory

- ★ Inability to assign physical spaces to logical addresses is called **Fragmentation**.
 - Both segmentation and paging suffer from storage as well as execution time overhead.
 - Both segmentation and paging suffer from **Table Fragmentation**.
 - Segmentation suffers from **External Fragmentation**.
 - Paging suffers from **Internal Fragmentation**.

◆ Virtual Memory

★ Issues of Concern:

- Replacement Policy — Locality of reference
- Write policy — Write back with dirty bit
- Optimum Page Size (small vs. large),
- Reducing the Overheads,



◆ Virtual Memory

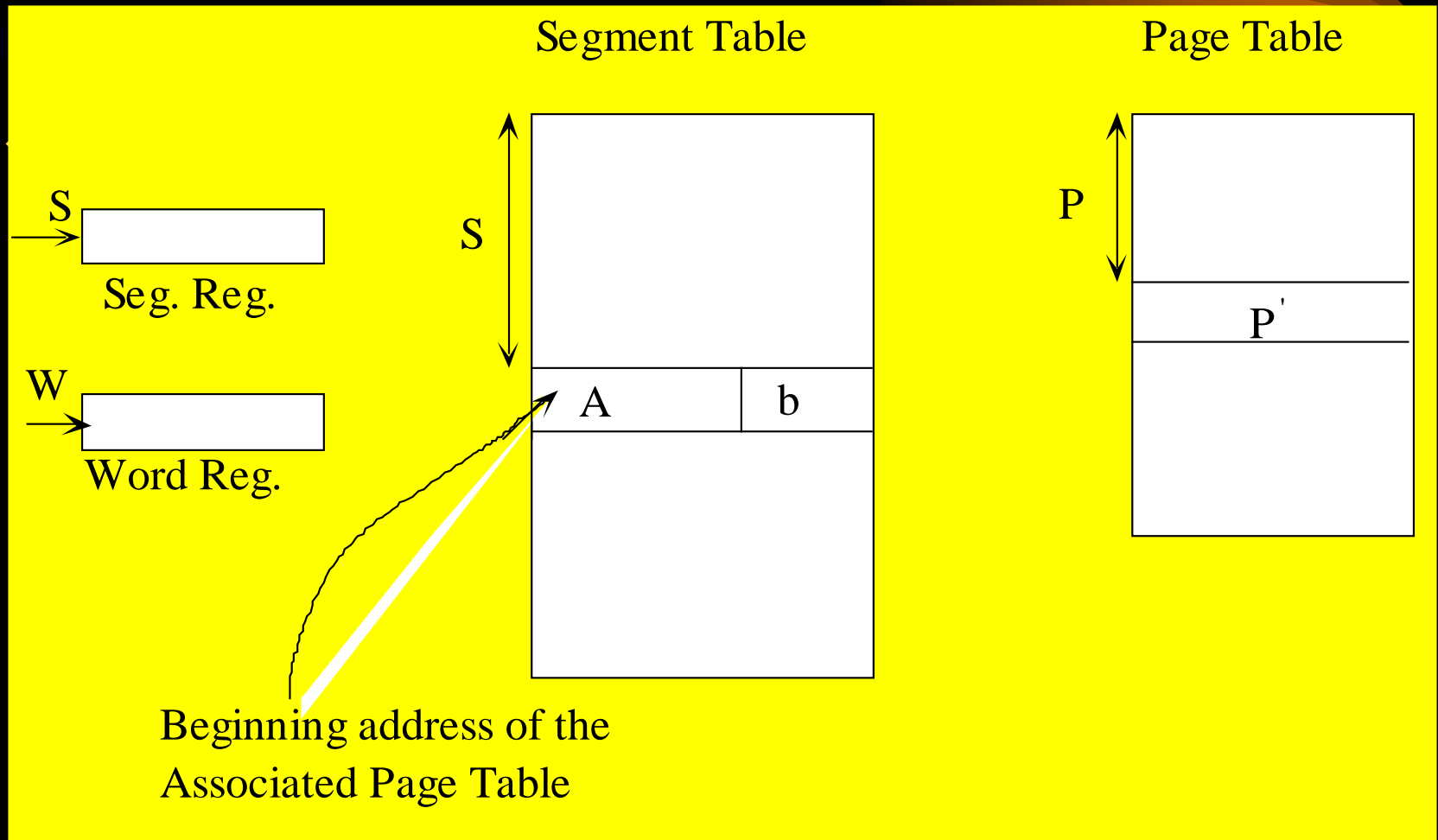
- ★ Segment or page table can be treated as a segment or a page, respectively.
- ★ Concept of paging and segmentation can be combined in order to get advantages of both approaches.
- ★ Associative memory can be used to speed up the address translation step.

◆ Virtual Memory — Segmentation & Paging

★ (S, W) is loaded into segment and word registers

- If S^{th} entry in the segment table is empty then **segment fault**.
- If $W > b$ then **overflow fault**.
- If P^{th} entry ($P = \lfloor W/Z \rfloor$) of the associated page table empty then **page fault**.
- $(P' + W - P * Z)$ is loaded into MAR.

Introduction to High Performance Computer Architecture



◆ Questions

- ★ What is an optimum page size?
- ★ What is the address translation procedure if the segment table is treated as a segment?
- ★ Within the scope of virtual memory: given the choice between lower miss ratio and a simple placement/replacement policy, which one would you choose?
- ★ Compare and contrast smaller and larger page sizes against each other.