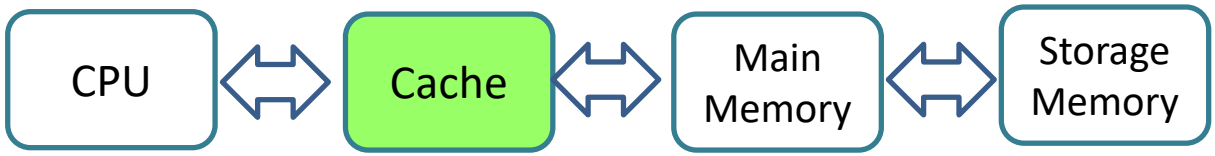
Cx1106
Computer Organization and Architecture

Cache

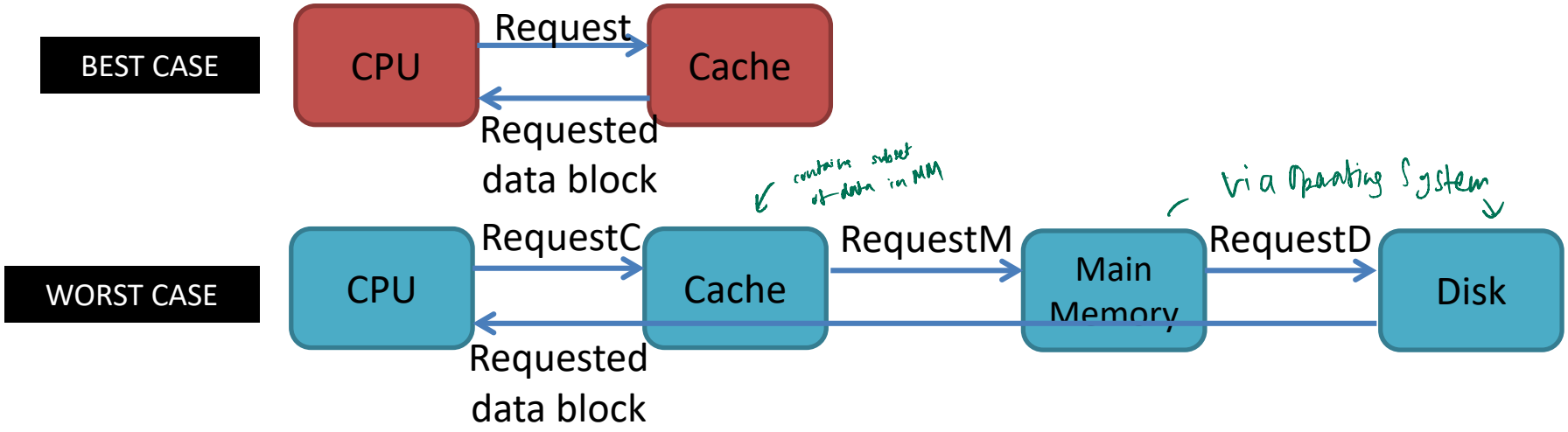
Oh Hong Lye
Lecturer
School of Computer Science and Engineering, Nanyang Technological University.
Email: hloh@ntu.edu.sg

Cache



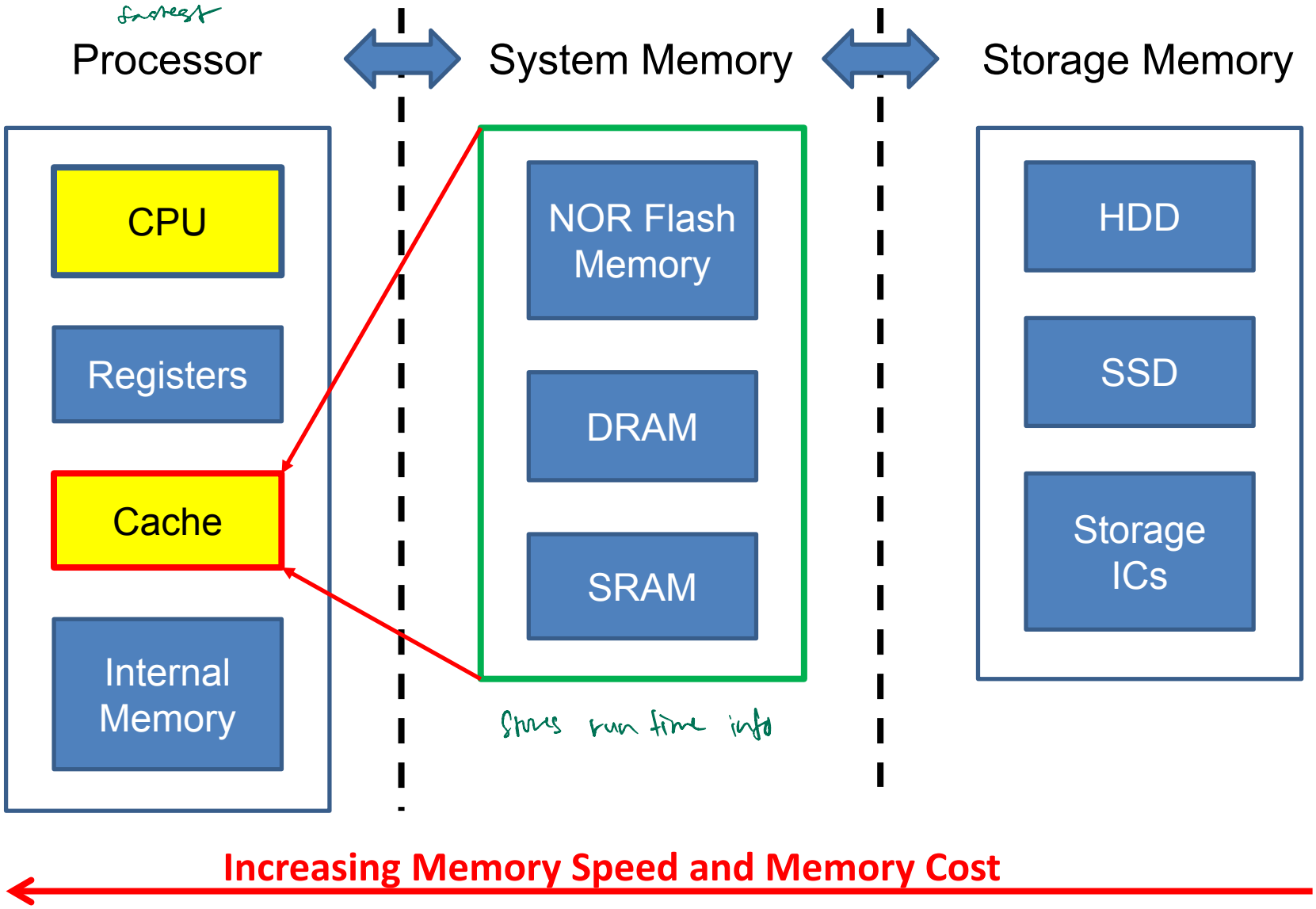
- Issue: **CPU** speed is typically much **faster** than **external memory**.
  - CPU speed in Ghz region
  - External Memory Speed in 100s of Mhz region.
- **Need a fast memory** to act as a **buffer** between the Main memory and CPU.
- The purpose of cache memory is to speed up accesses by storing/fetching **recently used data** closer to the CPU instead of the main memory (slower access).
- Potentially able to **improve the overall system performance** drastically.

CPU Memory Access

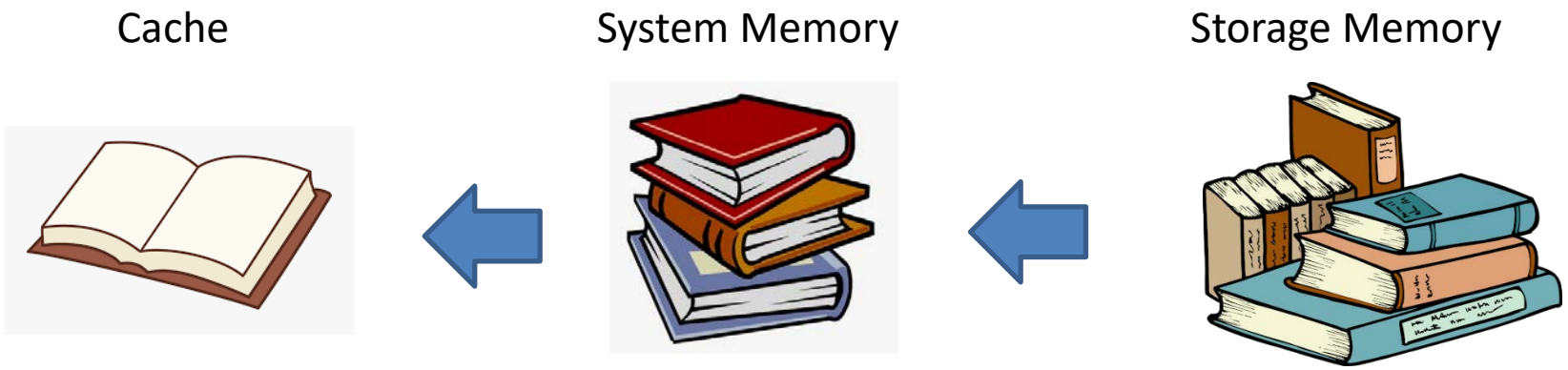


- To access a particular piece of data, the CPU first sends a request to its nearest memory, usually cache.
- If the data is not in cache, a query is then sent to the main memory.
- If the data is not in main memory, then the request goes to disk.
- Once the data is located, the **required data and a number of its nearby data elements** are fetched into cache memory simultaneously.

Computer Memory (Programmer's View)



Cache Design - Introduction

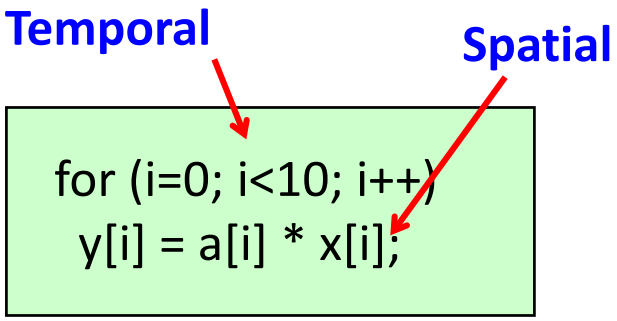


- Analogy: Open Book Exam
- Stacks of notes and text book
  - Entire data collection => Storage memory
- Can only tackle a question at a time during runtime
  - Relevant notes and section of text book => System memory
- At any instance, only a part/section of the question is attempted and only those information are required at that instance
  - Subset of the relevant notes and section of text book => Cache memory
- At any instance, only a small piece of information is required even though we are dealing with a huge set of data.

Principle of Locality

- Now that we know cache only needs to contain a subset of the main memory to work
- In what way should the data be transfer between main memory and cache in order to maximise the efficiency of a cache?
- Efficiency meaning how much improvement can the cache brings to the system
- For that, we need to understand the attributes of how information is organised and accessed when a program is executed.
- This is summarised in the concept called Principle of Locality

Principle of Locality



- The **principle of locality** tells us that once a byte in a program is accessed, it is likely a **nearby data element** will be needed soon.
- There are two principle of locality governing this behaviour
- Locality of **Space** (or Spatial Locality)
  - **Code/Data that is nearby each other** is likely to be accessed together
  - Transfer of data between System memory and cache is done in blocks to leverage on this behaviour
- Locality of **Time** (or Temporal Locality)
  - **Recently accessed code/data** is more likely to be accessed again
  - Used to decide which item to replace in the Cache

purge those not used for a long time

Cache Memory Replacement Policy

- When there is a need to transfer a block of data to the cache but the cache is fully occupied, there is a need to decide which cache block to evict/purge in order to free up space for the new data.
- The algorithm used to decide which block gets evicted is designed based on some Cache Replacement Policy. Two examples illustrated below
- **Least recently used (LRU)** algorithm keeps track of the last time that a block was assessed and evicts the block that has been unused for the longest period of time. The disadvantage of this approach is its complexity: LRU has to maintain an access history for each block, which ultimately slows down the cache.
- **First-in, first-out (FIFO)** is a popular cache replacement policy. In FIFO, the block that has been in the cache the longest, regardless of when it was last used.

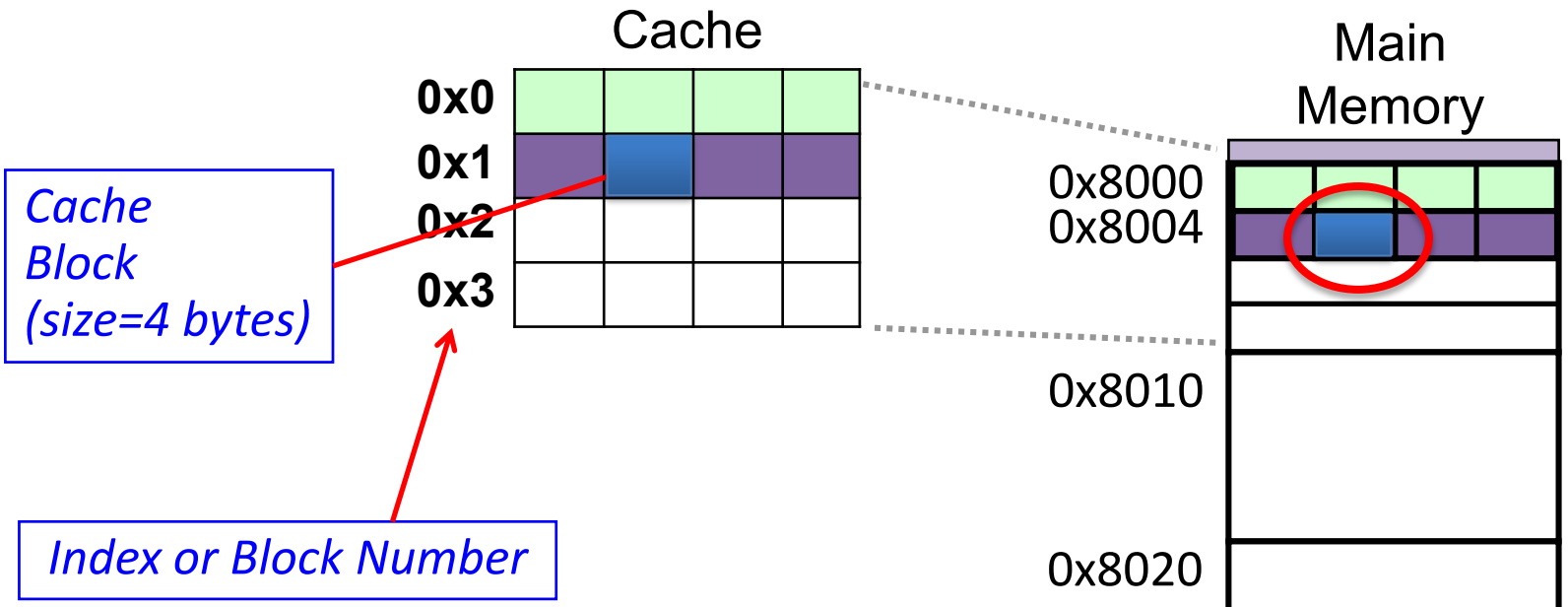
faster implementation through ⇐ FIFO less efficient than LRU

Cache Mapping Scheme

- We understand now that data transfer between main memory and cache is done in blocks instead of individual bytes to leverage on the principle of locality.
- Another attribute of cache design that affects the efficiency of the cache is the cache mapping scheme.
- Cache mapping scheme deals with how each main memory block is mapped to the a particular cache block, e.g. Main memory block #0x80 is mapped to cache block #0 (index 0).
- There are three basic cache mapping schemes
  - Direct Mapped
  - Set Associative
  - Fully Associative
- We will only touch on Direct Mapped Cache in this course, rest of the mapping scheme will be discussed in Advanced Computer Architecture Course.

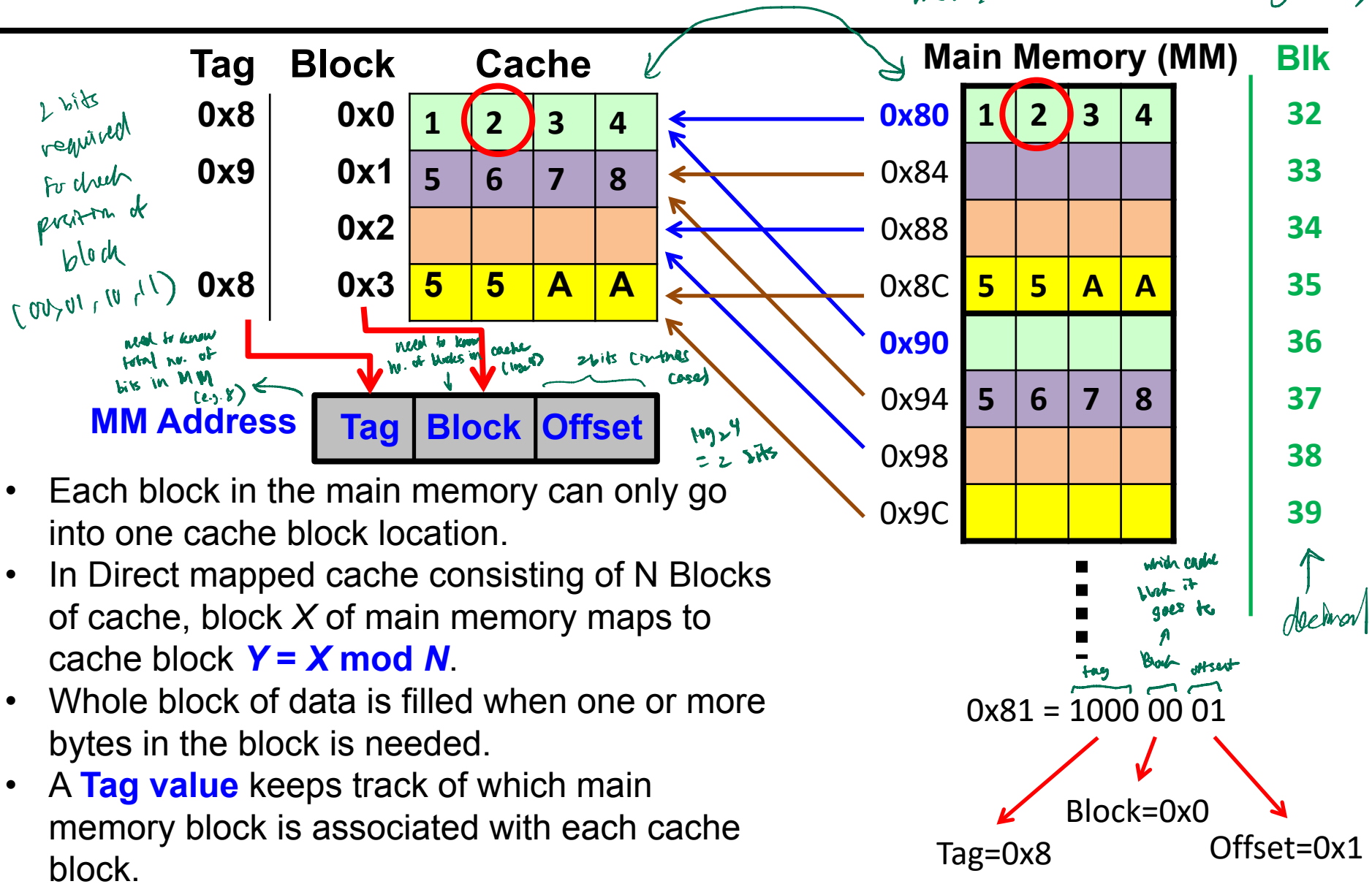
main / system memory
⇒ store runtime code/data

Terms used in Cache Mapping



- Conceptually, a cache is divided into smaller storage locations called **Blocks** (also known as **Cache Lines**).
- The term Index or Block-Number is used to specify a specific cache block/line.
- The **whole cache block** is transferred when one or more bytes in the block is needed.

Direct Mapped Cache



- Each block in the main memory can only go into one cache block location.
- In Direct mapped cache consisting of N Blocks of cache, block *X* of main memory maps to cache block ***Y = X mod N***.
- Whole block of data is filled when one or more bytes in the block is needed.
- A **Tag value** keeps track of which main memory block is associated with each cache block.

Cache Mapping (Elaboration on basic principle)

- Cache mapping
  - Allocates the data in the entire system memory into the cache which is of a much smaller size.
  - Able to uniquely identify each and every system memory location within the cache via the cache way of addressing: Block Index, Offset of the data within the block and the corresponding Tag Value of the block.
- To start doing the mapping, we need an attribute of the data that is unique to it within the entire system memory, for that, the system memory address of the data is chosen as each data's address is unique to itself.
- So the data's address is partitioned into the three fields: TAG, BLOCK and OFFSET so that proper allocation to cache can be done.

Cache Mapping (Elaboration on basic principle)

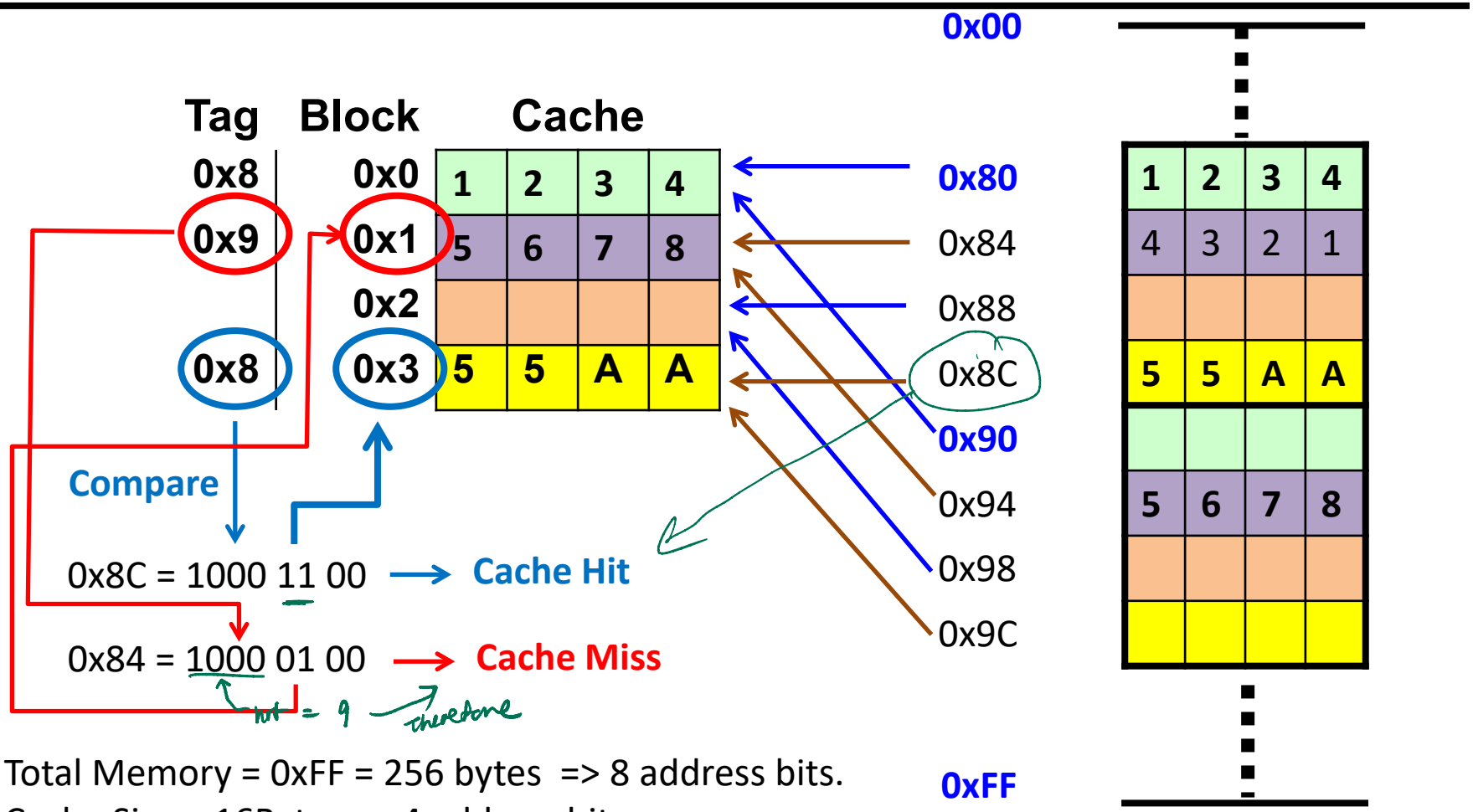
- The **number of bits allocated to each field** is a result of the structure of the cache.
  - **Offset** refers to the targeted data's location offset from the start of the cache block. Something like an address within the cache block. So if the size of a cache block is 16, one would need 4 bits in the OFFSET field to address these 16 locations.
  - **Block** refers to the index of the cache block that the targeted data will be mapped to. If there are 8 cache blocks for example, then one would need 3 bits in the BLOCK field to fully represent the cache block index.
  - **Tag** bits are the left over of target data's system memory address after partitioning for Offset and Block Index. Having this information will allow the cache system to **uniquely identify** the data in the cache.

Direct-Mapped Cache Mapping Example

- Given a system with following attribute
  - **Main/System Memory size = 64KBytes**
  - **Cache Size = 256Bytes**
  - **Cache Block Size = 16Bytes**

  } can used to obtain cache block
- Where would Data at main memory address **0x1106** be mapped to?
- Derivation of cache mapping format
  - Cache Block Size = 16Bytes => #**Offset** bits = log2(16) = **4 bits**.
  - Number of Cache Blocks = 256/16 = 16 Blocks
    => #**Blk** bits = log2(16) = **4 bits**.
  - Main memory size = 64KBytes => log2(64*1024) = 16bits address.
    #**Tag** bits = 16-4-4 = **8 bits**
  - Mapping Format = 8:4:4
- Applying to the main memory address 0x1106
  - 0x1106 = **0001 0001 0000 0110b** ← convert to binary
  - **BLK = 0x0**, **OFFSET=0x6**, **TAG=0x11**

Data Retrieval Example (Direct-Mapped Cache)



Total Memory = 0xFF = 256 bytes => 8 address bits.

Cache Size = 16Bytes => 4 address bits.

Cache Block Size = 4 Bytes => 2 address bits for offset.

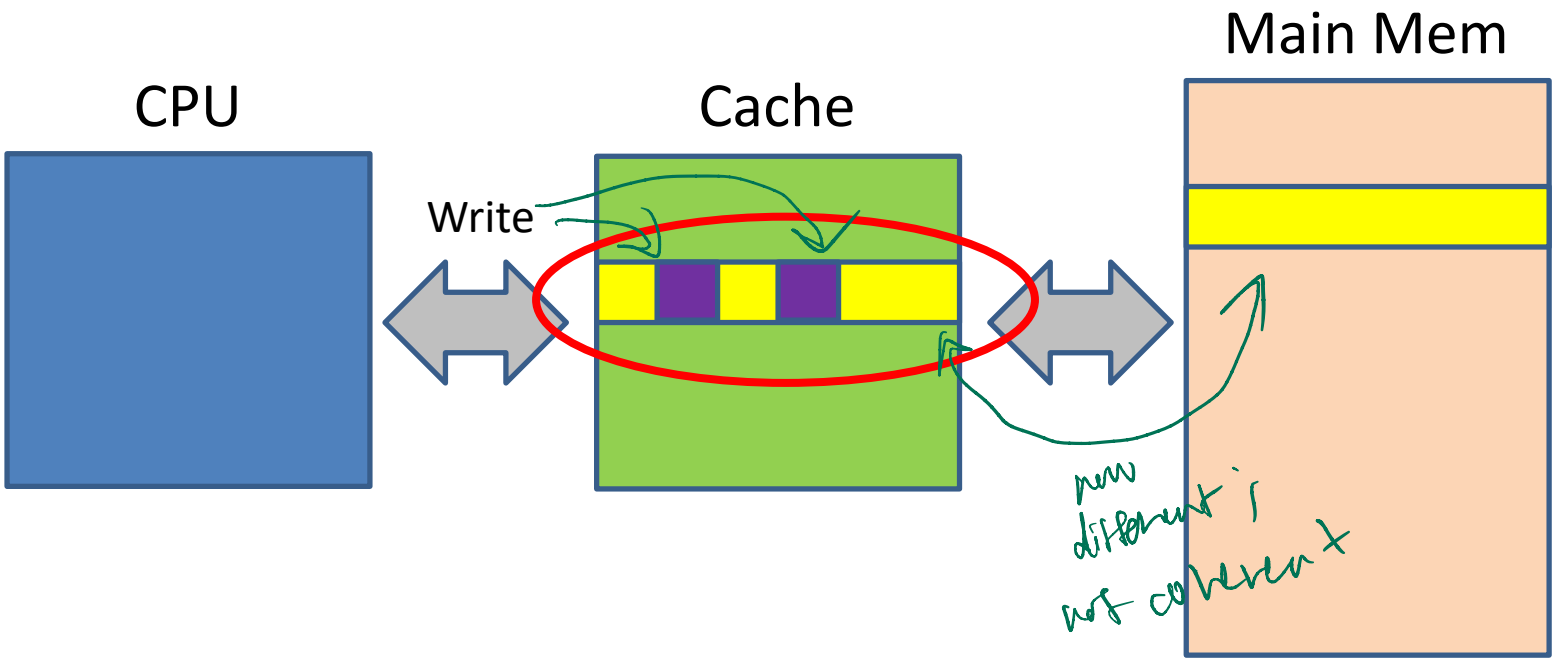
# of Blocks in Cache = 16/4 = 4 Blocks => 2 address bits (Block index)

# of TAG bits = 8-2-2 = 4 bits.

4 : 2 : 2

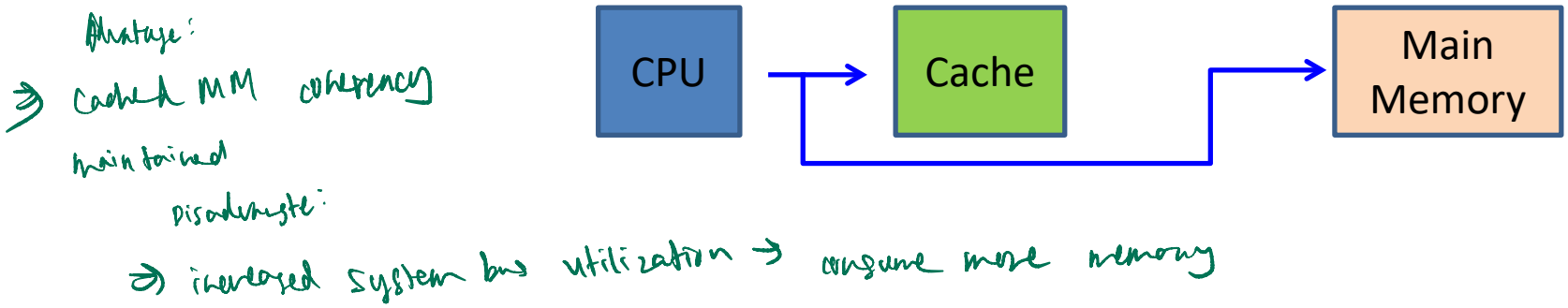
Cache Write Policy

- Locations in cache that are written into by CPU is known as **Dirty Block**
- Cache replacement policies must take into account **dirty blocks**, those blocks that have been updated while they were in the cache.
- Dirty blocks must be written back to memory. A **write policy** determines how this will be done.
- There are two types of write policies: **write through** and **write back**.

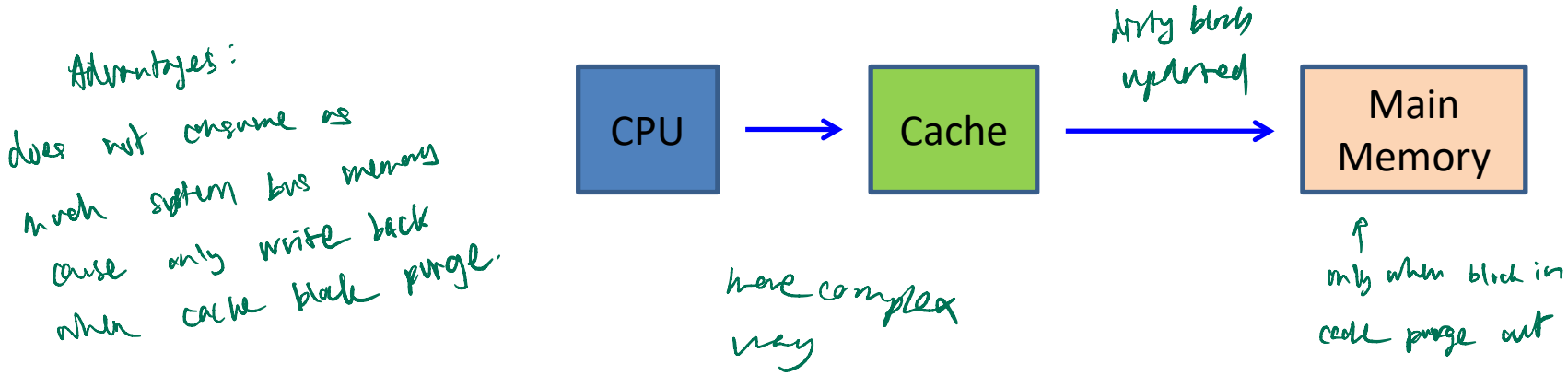


Cache Write Policy (Write Hit)

- **Write through** updates cache and main memory **simultaneously** on every write.



- **Write back** (also called copyback) updates memory **only** when the block is selected for replacement.

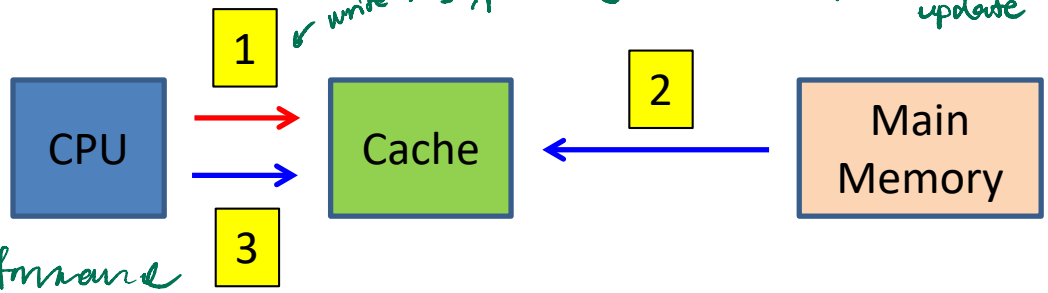


Cache Write Policy (Write Miss)

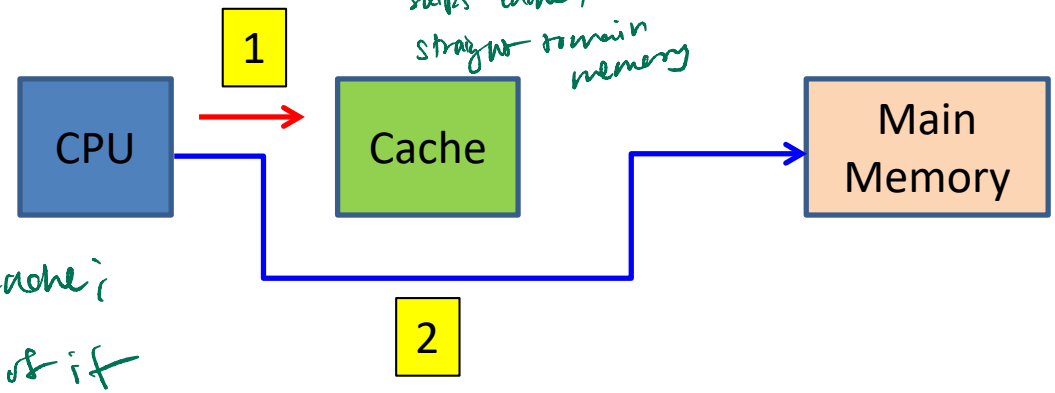
- **Write allocate** => fetch on write. A write miss will cause the data block at the write-miss address to be **loaded to the cache**.

more data transfer
⇒ slower

principle of locality used to improve cache efficiency/performance



- **Write-no-allocate** => A write miss will not cause the data block to be loaded to the cache. Data **Write** is done **directly** to its location in **main memory**.



never no coherency issue.
does not make use of principal of locality.
⇒ data from memory not given to cache; subsequent calls cannot make use of it

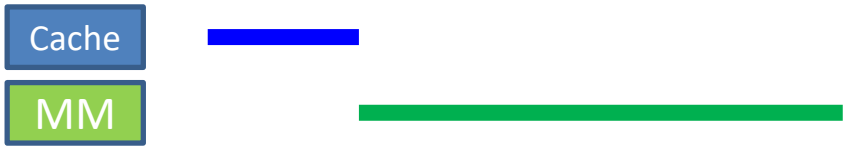
Cache Performance Related Terminologies

- **Cache Hit**
  - Data is found in the cache.
- **Cache Miss**
  - Data is not found in the cache.
- **Cache Hit rate (H)**
  - Percentage of time data found in the cache
- **Cache Miss rate**
  - Percentage of time data not found in cache.
- **Miss rate = 1 - Hit rate = (1 – H).**

↳ probability

Effective Access Time

Sequential Access of Cache and Main Memory, i.e. access do not overlap.



- The performance of hierarchical memory is measured by its effective access time (EAT).
- EAT is a weighted average that takes into account the hit ratio and relative access times of successive levels of memory.
- The EAT for a two-level memory is given by

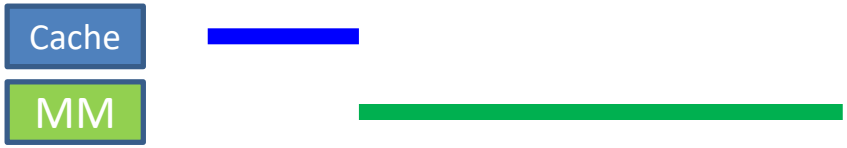
$$EAT = H \times Access_C + (1-H) \times \textit{Miss Penalty}$$

- $Access_C$ = Access times for cache
- Miss Penalty = Time need to access the data when there is Cache Miss
- If we assume that data access to Cache and Main memory do not overlap, then Miss Penalty = $Access_C + Access_{MM}$, where $Access_{MM}$ is the access times for main memory

$$EAT = H \times Access_C + (1-H) \times (Access_C + Access_{MM})$$

Effective Access Time (Example)

Sequential Access of Cache and Main Memory



- Consider a system with a **main memory access time of 200ns** supported by a **cache** having a **10ns access time** and a **hit rate of 99%.**
- If the **accesses do not overlap**,

  The EAT is:

  $$0.99(10\text{ns}) + 0.01(10\text{ns} + 200\text{ns}) = 9.9\text{ns} + 2.1\text{ns} = 12\text{ns}.$$
- This equation for determining the effective access time can be extended to any number of memory levels.

M3 Lecture

**Which of the following is/are true for Cache Memory?

- ✓A. Build with Fast Memory — looking at access time
- ✓B. Contain a subset of data/code in main and storage memory
- ✓C. Contain frequently used data/code
- ✓D. Transfer to cache are in blocks instead of single word.

Cache Levels

L1 : fastest, small size (more expensive as well)

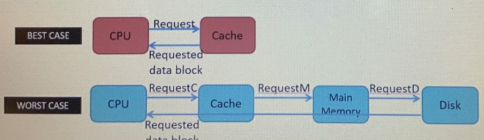
L2 : slower than L1, bigger size than L1

L3 : slower than L2, bigger size than L2

When CPU needs some data/code, which module does it check first for the information required?

- A. Main Memory
- B. Storage Memory
- ✓C. Cache
- D. Virtual Memory

if not in cache → main memory → storage memory

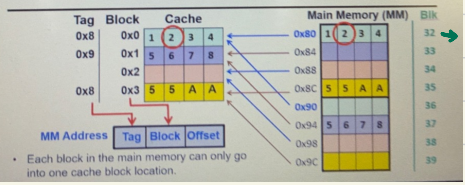


How many location(s) within the cache can a main memory block be mapped to in a direct mapped cache?

✔A. 1
B. 2
C. 4
D. Any block within the cache

main memory block mapped to 1 cache memory block

Direct Mapped Cache



- Each block in the main memory can only go into one cache block location.

only go to cache block 0

Does Direct Mapped Cache require a cache replacement policy?

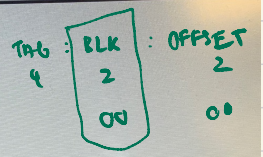
✔A. No
B. Yes

Cache Memory Replacement Policy

- When there is a need to transfer a block of data to the cache but the cache is fully occupied, there is a need to decide which cache block to evict/purge in order to free up space for the new data.
- The algorithm used to decide which block gets evicted is designed based on some Cache Replacement Policy. Two examples illustrated below
- **Least recently used (LRU)** algorithm keeps track of the last time that a block was assessed and evicts the block that has been unused for the longest period of time. The disadvantage of this approach is its complexity: LRU has to maintain an access history for each block, which ultimately slows down the cache.
- **First-in, first-out (FIFO)** is a popular cache replacement policy. In FIFO, the block that has been in the cache the longest, regardless of when it was last used.

What is cache thrashing?

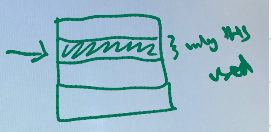
A. Cache erasure process
✓B. Same cache block being replaced frequently
C. Damaging the Cache memory
D. Cache removal



e.g. Cache with
- cache block size = 4
- number of cache blocks = 4
- MM address range = 8bits
- => 4:2:2
- Address access sequence
  - 0x00, 0x10, 0x20, 0x30,...
  - Cache Block 0 kept being purged
  - => thrashing

What is/are the effect of cache thrashing.

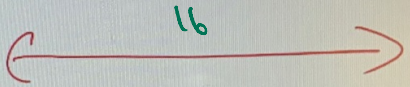
✓A. Lower cache hit rate
B. Higher cache hit rate
✓C. Cache memory not fully utilized
D. Low cache replacement rate



Imagine if the processor is always using the same block out of 32 blocks available in a cache to cache the data it is accessing.

Given a MM size of 64KByte, Cache size of 256Bytes and block size of 16 bytes. Direct Mapped Cache. What is the format for Tag:Block:Offset of the cache?

2^{16}



A. 4:4:8
✓B. 8:4:4
C. 4:8:4
D. None of the above

offset : look at cache blk size

$\Rightarrow \log_2 16 = 4$

$16 - 4 - 4 = 8$

64K => 16bits address
Cache Block size 16bytes => 4bits offset
Cache Size 256bytes => 16 blocks => 4bit block index
=> 8 bit Tag

What is the Tag value of MM block with starting address 0x20. Assume a Direct-Mapped Cache with 4 blocks, block size 4 bytes and 0xFF MM address range.

?

✓A. 0x2
B. 0x10
C. 0x0
D. 0x20

Find Tag blk offset first ↓ corresponding to cache blk size

0xFF => 8 bits address
⇒ 4:2:2
0x20 = 0010 0000b
=> TAG = 0010b = 0x2

$\log_2 4 = 2$

$0x20 = 0010 \mid 00 \mid 00$

0x2

Always operand in BINARY ⇒

**Which of the following statement(s) illustrate the principle of spatial locality?

A. Code in a loop gets executed multiple times
✓B. Code are often executed sequentially
✓C. Data in an array are accessed one after another typically
✓D. Variables used within a function are typically assigned near to each other by the compiler

**Which of the following statements illustrate the principle of temporal locality?

for(...)
{ func ?
}

✓A. Code in a loop gets executed multiple times
B. Code are often executed sequentially
C. Data in an array are accessed one after another typically
✓D. Variables within a function are typically used multiple times in the function

LRU: Least Recently Used

⇒ If need to replace, use the one that has not been used for the longest time

Which principle of locality is the LRU replacement policy based on?

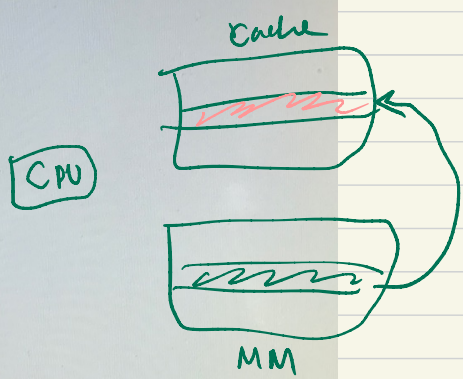
A. Locality of space
✓B. Locality of time

- Locality of **Time** (or Temporal Locality)
  - **Recently accessed code/data** is more likely to be accessed again
  - Used to decide which item to replace in the Cache
- **Least recently used (LRU)** algorithm keeps track of the last time that a block was assessed and evicts the block that has been unused for the longest period of time. The disadvantage of this approach is its complexity: LRU has to maintain an access history for each block, which ultimately slows down the cache.

What is cache coherency issue?

Not about the access

✓A. Content in the cache is different from the content of corresponding locations in the main memory
B. Too many access to the cache
C. Cache access not in sync with memory access
D. Cache data corrupted



How does cache coherency issue occur?

A. Too many reads from Cache.

✓B. CPU writes to cached data but main memory is not updated accordingly

C. Using Write-Through Policy

D. Using LRU Cache replacement policy

What is the advantage that Write-Through policy has over Write-Back?

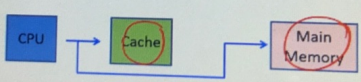
A. Less traffic on system bus

B. Faster write throughput → Write through slower

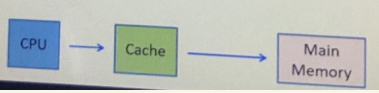
✓C. No cache coherency issue

D. No additional main memory write needed for write to cached data

- **Write through** updates cache and main memory **simultaneously** on every write.



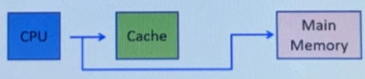
- **Write back** (also called copyback) updates memory **only** when the block is selected for replacement.



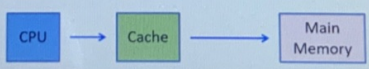
What is the disadvantage that Write-Through policy has over Write-Back?

✔A. Generate more traffic on system bus
B. More complex cache data manipulation
C. Worsen the cache coherency issue
D. Delayed updating of cached data

- **Write through** updates cache and main memory **simultaneously** on every write.



- **Write back** (also called copyback) updates memory **only** when the block is selected for replacement.



Cx1106

Cx1106
Computer Organization and Architecture

Virtual Memory

Oh Hong Lye
Lecturer
School of Computer Science and Engineering, Nanyang Technological University.
Email: hloh@ntu.edu.sg

→ facilitates development of individual devices

→ with seperate memory address space, devices able to extract themselves from how other connecting devices store its data

MEMORY ADDRESS SPACE

Memory Address Space (Analogy)

Address Space 3
(Employee)

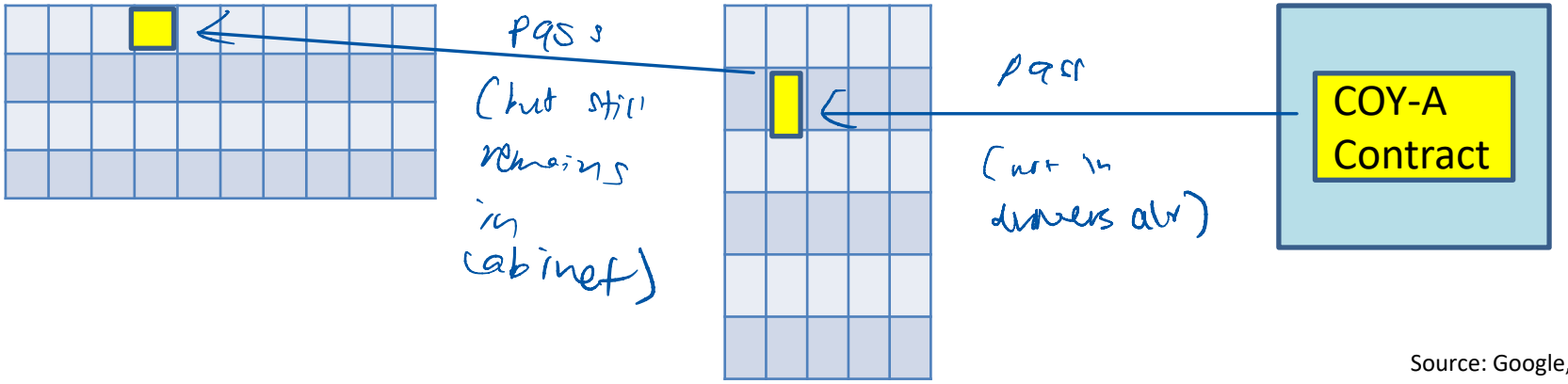
Address Space 2
(Manager)

Address Space 1
(Boss)

Cabinet (Company Name)

Cabinet (Market Segment)

Drawer (all-in-one)



Source: Google, Baby Boss.

Memory Address Space (Computing)

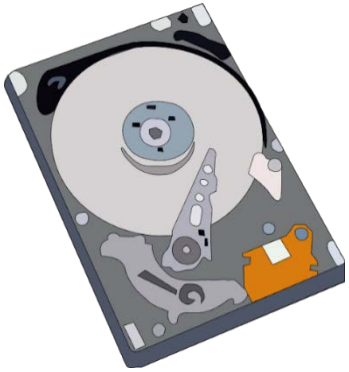
Address Space 3 (Storage Memory Controller)

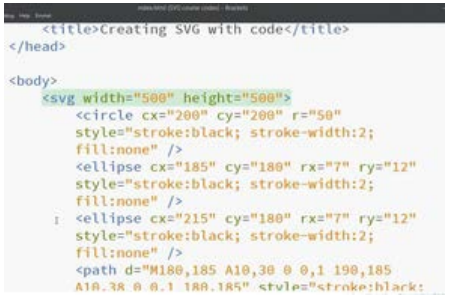
Address Space 2 (System Memory Controller)

translations done here (paging, referencing, etc.)

Address Space 1 (Program)

OS



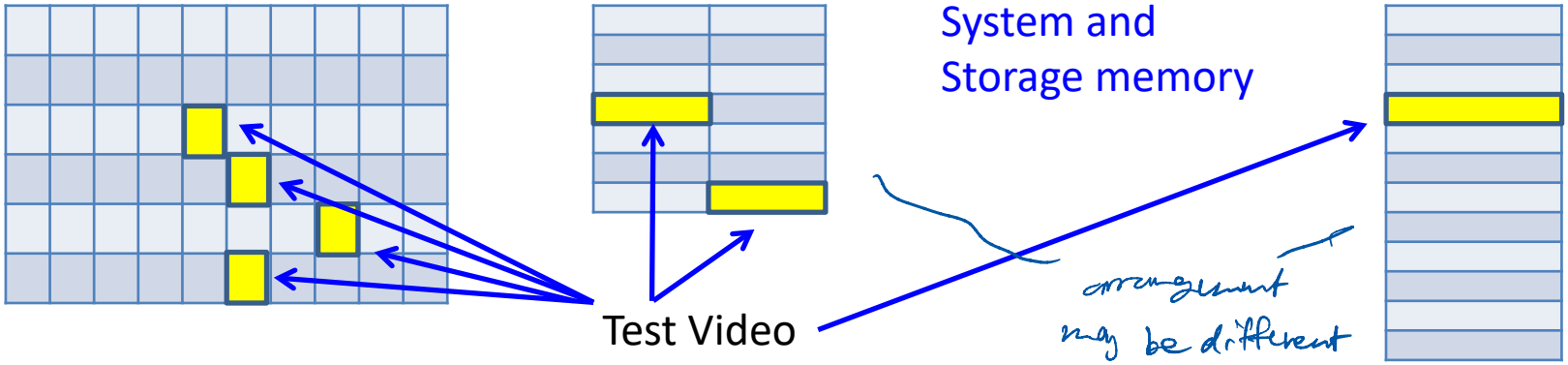


OS keeps track of the location of Test Video in the System and Storage memory

Storage Memory Addressing (LBA)

System Memory Addressing

Virtual Memory Addressing

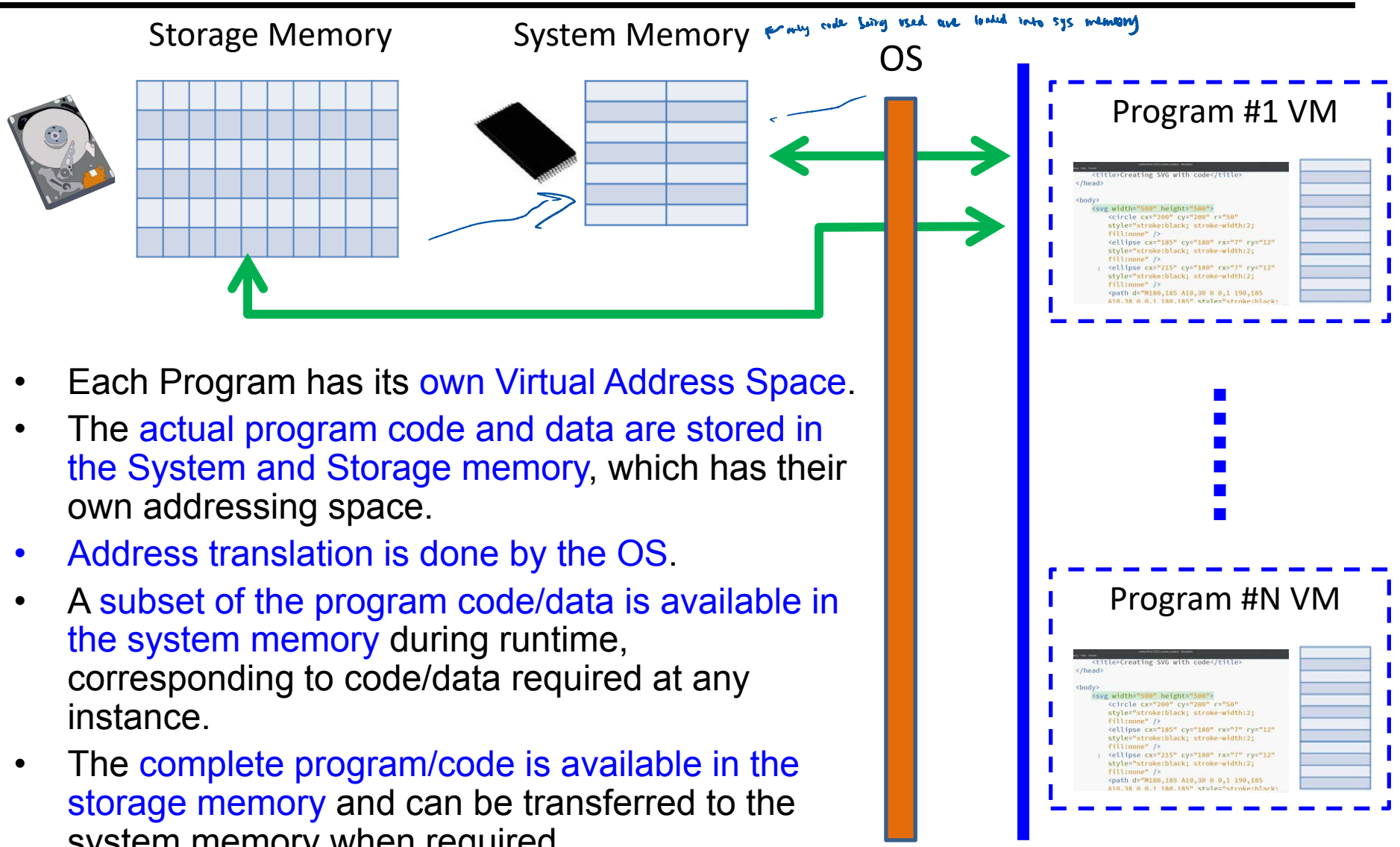


Basic Concepts

- In computing, address space is a range of discrete addresses corresponding to some physical or logical entity, e.g. program virtual memory, physical system memory, logical layout of a HDD etc.
- Every piece of data/code in a program is assigned an address by the compiler during the program compilation.
- When executing a program, the information it requires is obtained by issuing a request to the operating system (OS) using the addresses assigned by the compiler.
- The program doesn't need to know how the required information is organised in the system memory and rely on a middle man to do the corresponding address translation in order to fetch the correct information.
- In this case, the operating system is the middle man, translating the compiler generated address to the system memory address where the required information is stored.
- Program will still work as long as it gets the code/data it requested.

VIRTUAL MEMORY MANAGEMENT

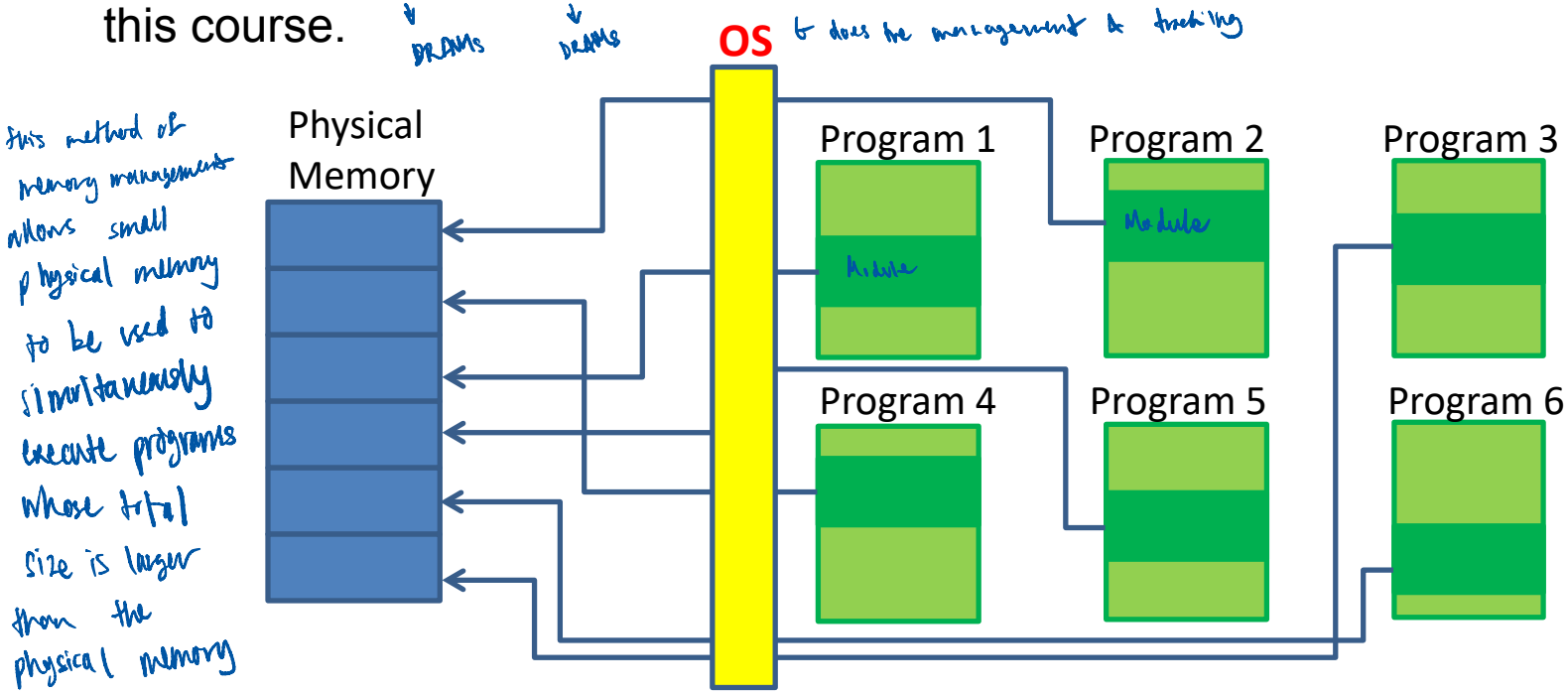
Virtual Memory (VM) Management



- Each Program has its own Virtual Address Space.
- The actual program code and data are stored in the System and Storage memory, which has their own addressing space.
- Address translation is done by the OS.
- A subset of the program code/data is available in the system memory during runtime, corresponding to code/data required at any instance.
- The complete program/code is available in the storage memory and can be transferred to the system memory when required.

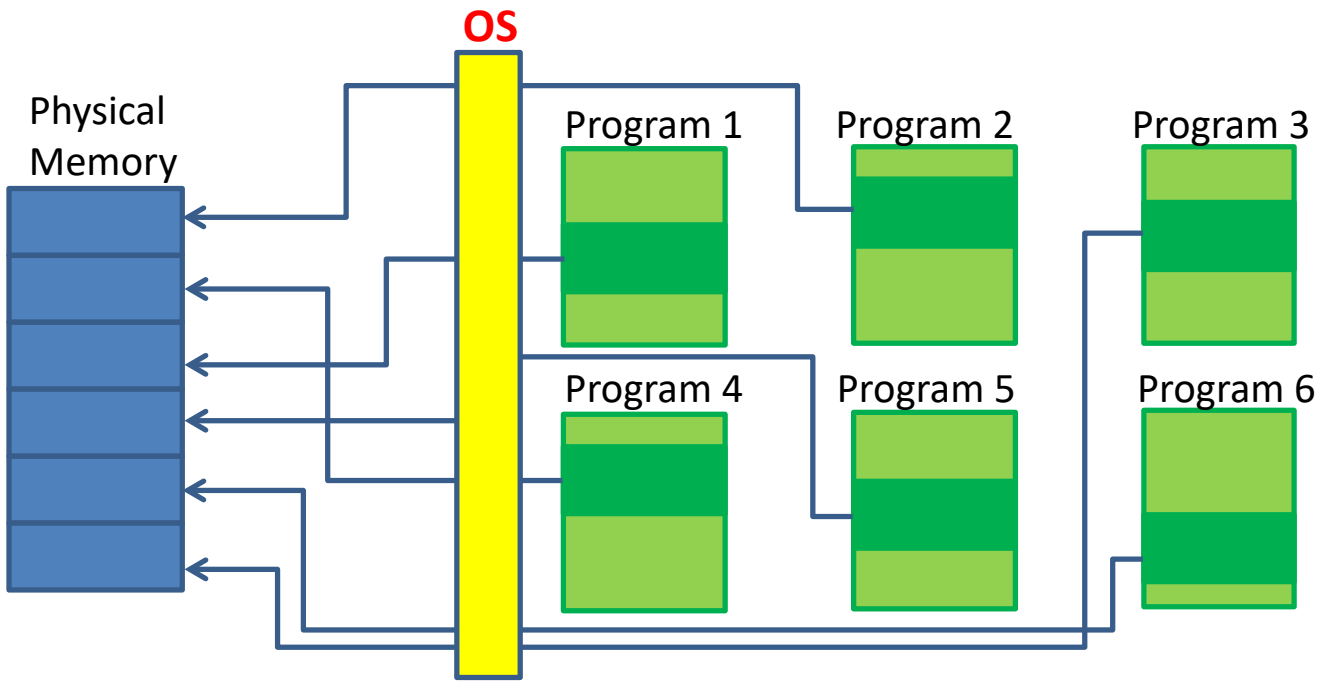
Virtual vs Physical Memory Address Space

- The addresses used by the program is generated by the compiler and is known as the virtual address.
- The virtual address of the code/data is typically different from the Physical Memory Address that they will be resided.
- Address Translation is thus required and is done in hardware and/or software, managed by the Operating System (OS).
- Note that Physical, System and Main memory refers to the same memory for this course.



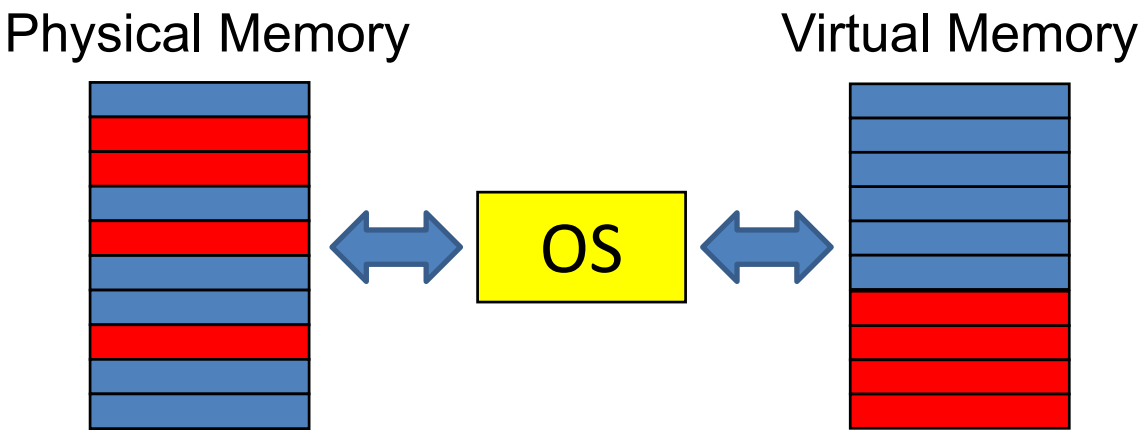
Advantages of Virtual Memory

- OS is able to isolate each virtual memory space and prevent corruption between these spaces.
- Allow efficient and safe sharing among different programs within the shared physical memory.
- Allow one or more programs to run in the physical memory simultaneously even if the total size of all the programs is larger than the actual physical memory size.



Advantages of Virtual Memory

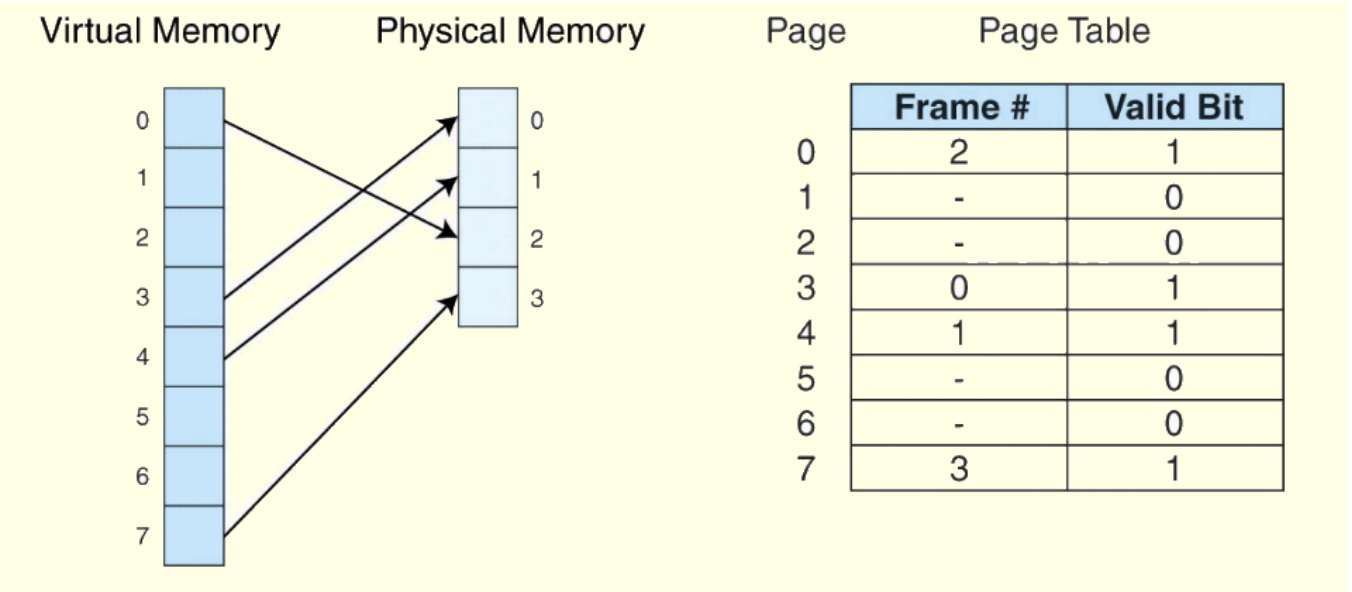
- OS is able to relocate virtual memory blocks to any physical memory blocks. This allows the virtual address space seen by the application to appear to be contiguous when it is actually spread across fragmented blocks in the physical memory.



↓
fragmented across different blocks ; maximise the utilisation of all empty spaces in physical memory

Address Mapping

- There are two common schemes used in the industry for address mapping
  - Paging. Memory space partitioned into Fixed sized blocks
  - Segmentation. Memory space partitioned into variable sized segments.
- For our course, we will only deal with paging with single level page table.

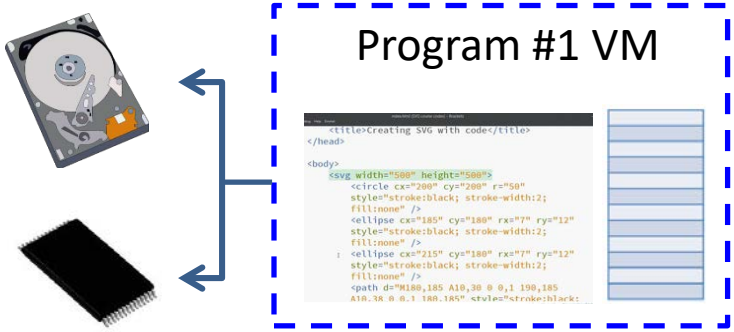


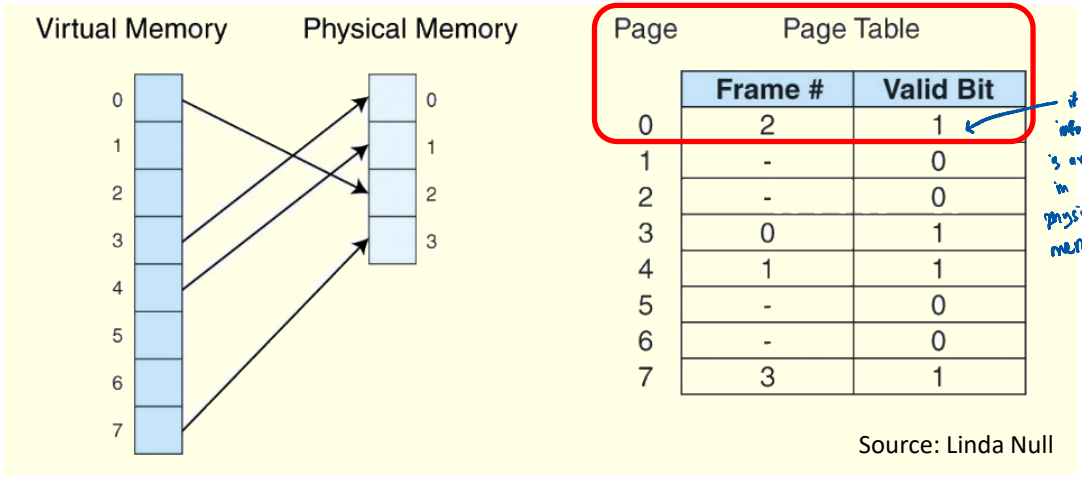
Page	Frame #	Valid Bit
0	2	1
1	-	0
2	-	0
3	0	1
4	1	1
5	-	0
6	-	0
7	3	1

Source: Linda Null

Paging Method

- In a system that uses paging, the memory space is partitioned into fixed size blocks known as a Page/Frame.
- Information concerning the location of each page, whether on disk or in memory, is maintained in a data structure called a page table (shown below).
- In the Page Table below
  - Frame refers to the physical frame number in the main memory.
  - Page refers to the virtual page number used by program code.
  - Valid Bit (VB) indicates whether the Virtual Page is in the main memory (VB=1) or not (VB=0).





Page Table

Page	Frame #	Valid Bit
0	2	1
1	-	0
2	-	0
3	0	1
4	1	1
5	-	0
6	-	0
7	3	1

Source: Linda Null

Address Translation

log2 can be used

4 bits of binary address

3 bits of PM locations

Virtual Memory

	Addr (Binary)	Offset	Value
Page0	0000	00	00
	0001	01	11
	0010	10	22
	0011	11	33
Page1	0100	00	44
	0101	01	55
	0110	10	66
	0111	11	77
Page2	1000	00	88
	1001	01	99
	1010	10	AA
	1011	11	BB
Page3	1100	00	CC
	1101	01	DD
	1110	10	EE
	1111	11	FF

11_2 ⇐

Physical Memory

Offset	Value	Addr (Binary)	
00	CC	000	Frame0
01	DD	001	
10	EE	010	
11	FF	011	
00	88	100	Frame1
01	99	101	
10	AA	110	
11	BB	111	

Page Size = 4Bytes. Virtual Address 1101b.

3 offset

1101b

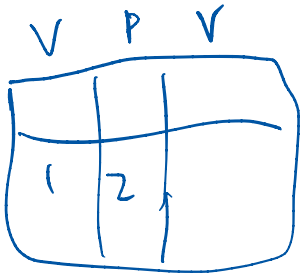
001b

- Work in Binary Numbers
- Offset remain unchanged
- Virtual Page Num replaced with Physical Frame Num

Address Translation

E.g. address : 0x100
Page size: 256 bit = 2^8
↓
Virtual Page number is "1"

Work in Binary Numbers format



Virtual Memory Address

last 8 bits

Virtual Page Number	Offset

Virtual Page Num replaced with Physical Frame Num

Translation via Page Table

0x200

Offset remain unchanged

10

0000 0000

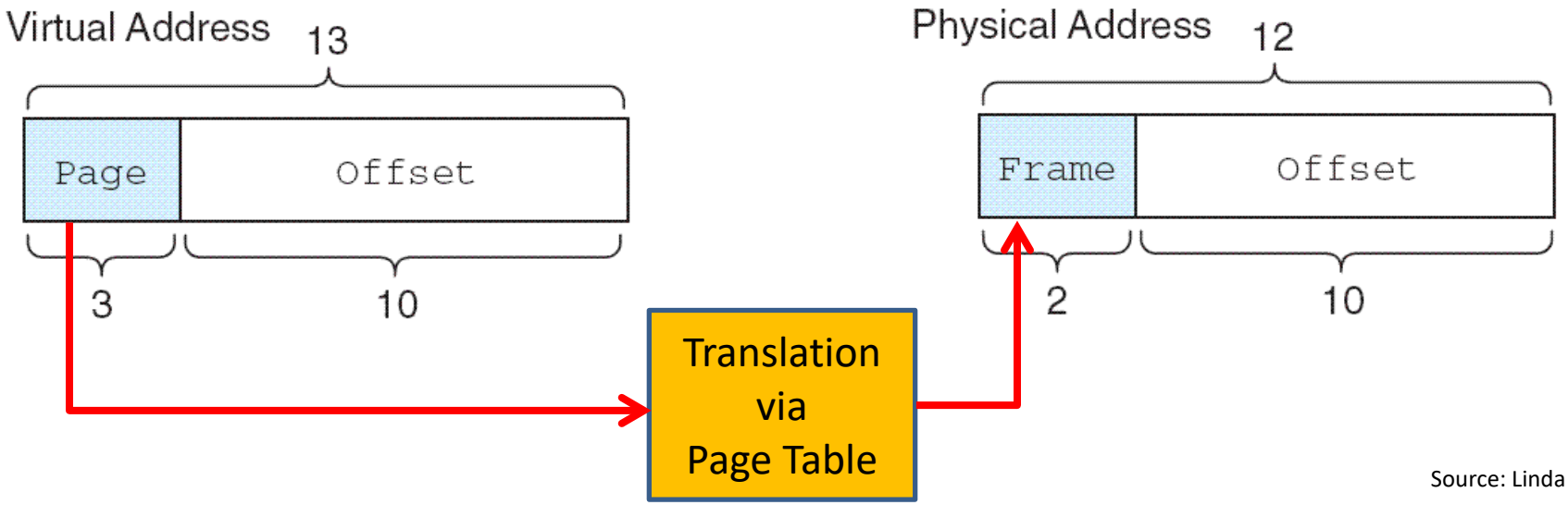
Physical Frame Number	Offset

Physical Memory Address

Paging Example

$k = 2^{10}$ (in this course)

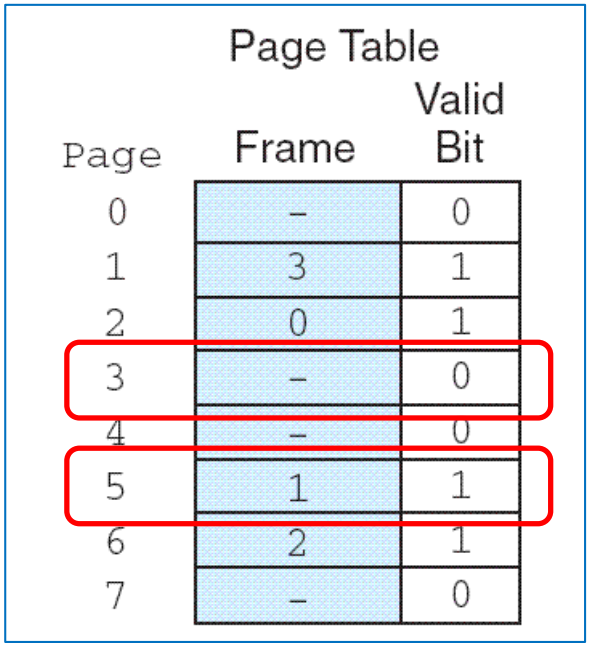
- Consider a system with a virtual address space of 8K and a physical address space of 4K, and the system uses byte addressing.
- If page size is 1KByte, we have 8 virtual pages mapping to 4 physical frames.
- Note that Virtual Page Size is always equal to Physical Frame Size.
- A virtual address has 13 bits ($8K = 2^{13}$) with 3 bits for the page field and 10 for the offset, because the page size is 1024.
- A physical memory address requires 12 bits, the first two bits for the page frame and the trailing 10 bits the offset.



Source: Linda Null

Paging Example

- Suppose we have the page table shown below.
- What happened when CPU access virtual address location
  - 0x1553
  - 0x0FA0



Source: Linda Null

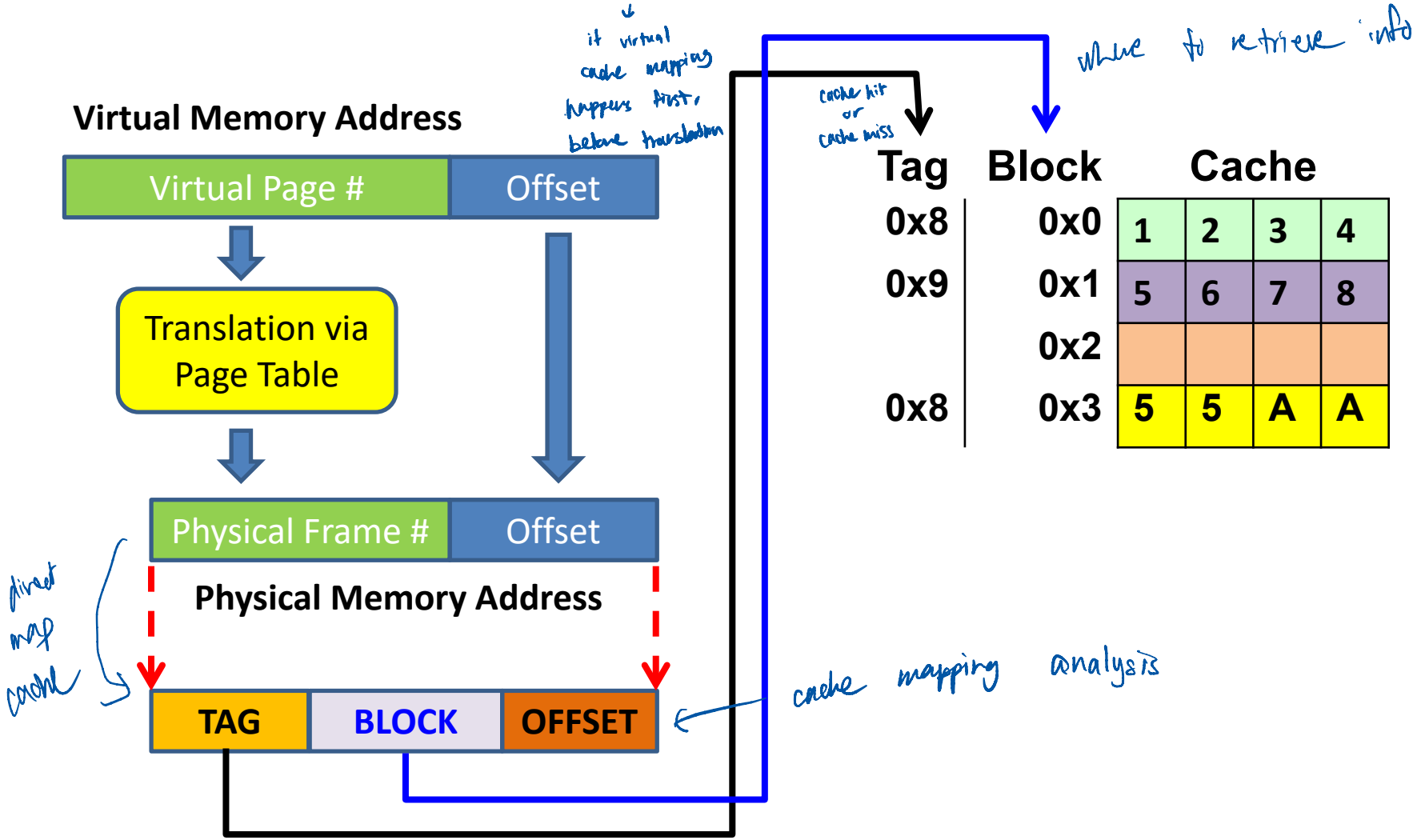
- 0x1553 = 1010101010011b
  - Data resides in Virtual Page 5
  - From Page Table, Virtual Page 5 is mapped to Physical Frame 1 and Valid bit is 1.
  - Physical Address = 010101010011b = **0x0553**
- 0x0FA0 = 0111110100000b
  - Data resides in Virtual Page 3
  - From Page Table, Valid bit is 0
  - Data not in Physical Memory
  - **Page Fault**

Paging Example

- In the previous slide, when accessing virtual address 0x1553 (which is in virtual page 5), the page table shows the valid bit is 1. From the page table, physical frame 1 will be used for subsequent address translation needed to access the actual data.
- When accessing virtual address 0xFA0 (virtual page 3), the page table shows that the valid bit is zero. If the valid bit is zero in the page table entry for the virtual address, this means that the page is not in memory and must be fetched from Storage Memory.
  - This is a page fault.
  - If necessary, a page is evicted from memory and is replaced by the page retrieved from storage memory, and the valid bit is set to 1.
- For both cases above, the data is then accessed by appending the offset to the physical frame number (address translation is simply replacing the virtual page number with the corresponding physical frame number in the page table).

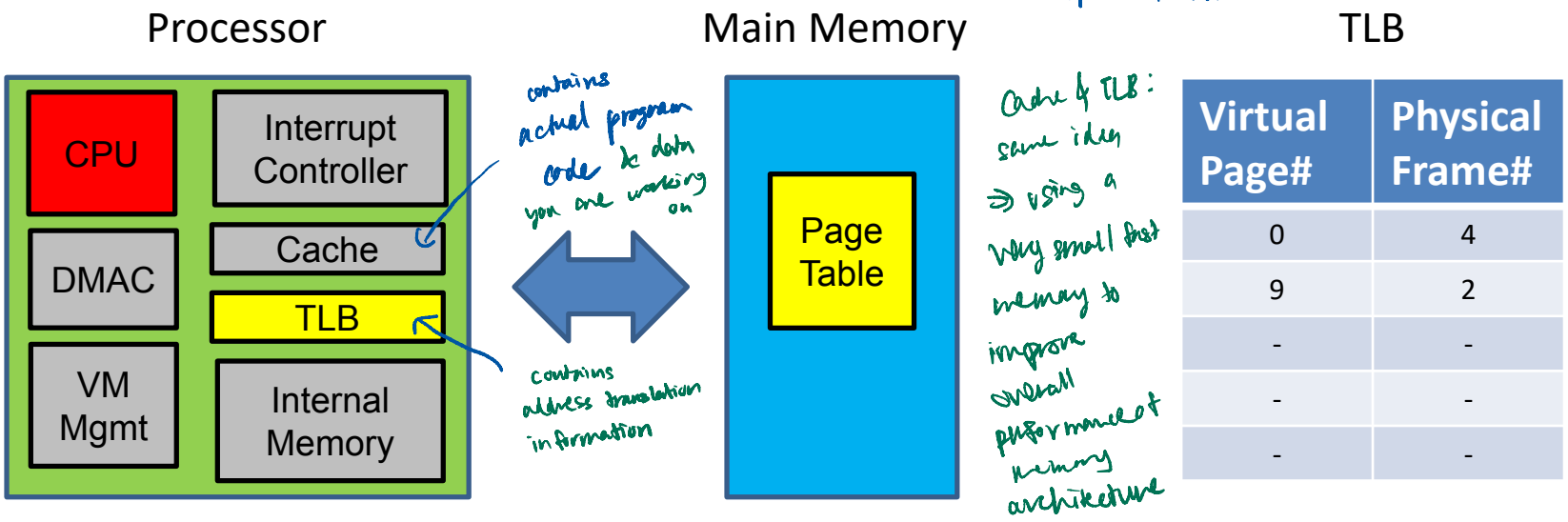
Integrating Cache and Virtual Memory

- Assumption: Cache uses Physical Memory Address to perform Mapping.

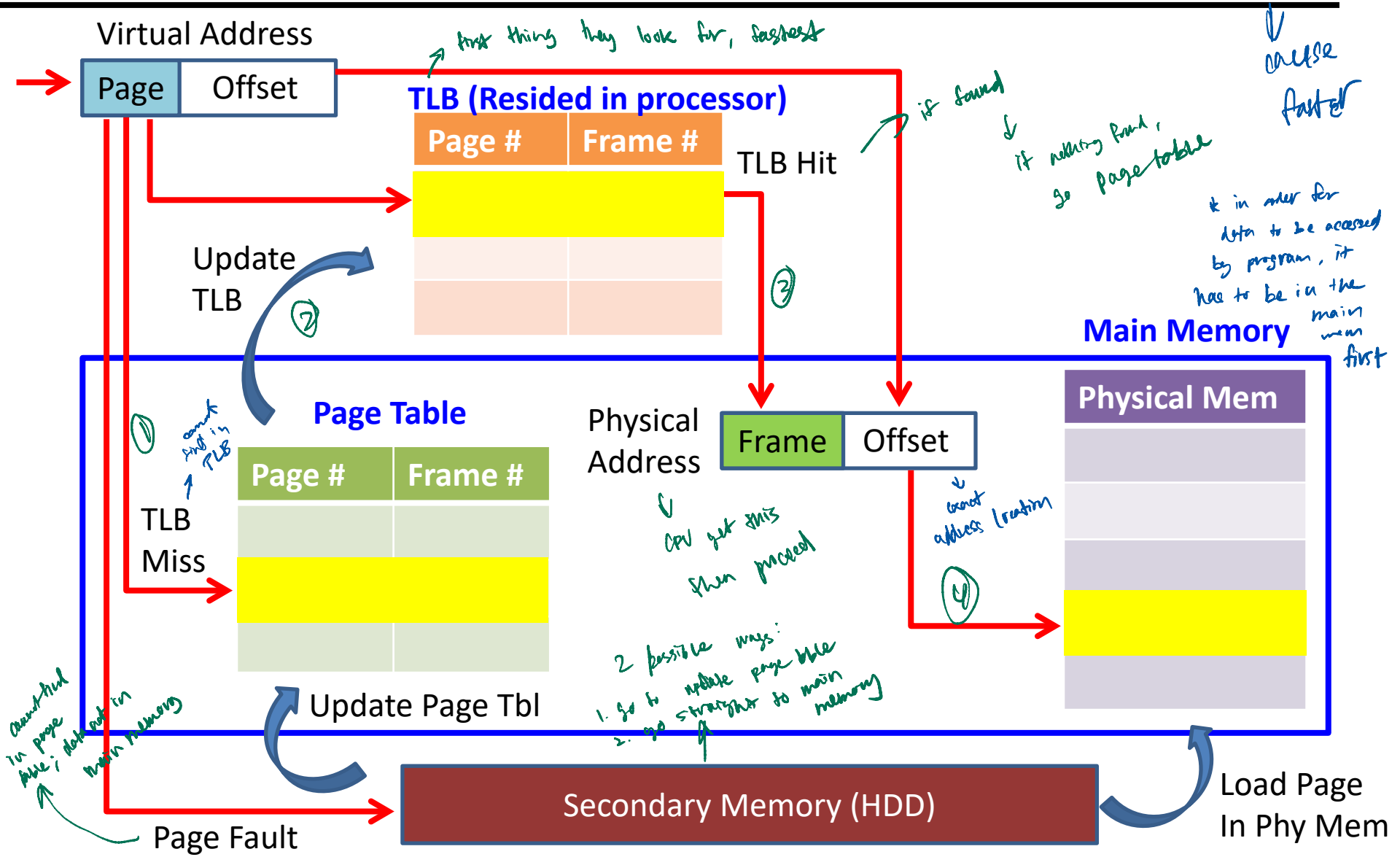


Translation Lookaside Buffer (TLB)

- Page Table is located in Main Memory so access is relatively slower than internal memory access.
- To speed up the page translation process, a page table cache (TLB) is used to stored a list of most recently/frequently used address translation entries in the page table.
- TLB is implement with fast memory such as SRAM and reside within the processor.
- Each TLB entry includes a Virtual Page number and its corresponding Frame Number.



TLB Lookup Process



In a system with 32 virtual pages and 8 physical frames, what is the maximum number of valid entries in the paging table?

A. 32
B. 8
C. 5
D. None of the above

found inside main memory; corresponds to physical frames

Page table contains entries for *Every* Virtual Pages BUT Not all are valid.
Only virtual pages that are mapped to a Physical Frame may be valid.

In a system using paging scheme with 32 virtual pages, 8 physical frames and virtual address range of 64KByte, what is the size of one physical frame?

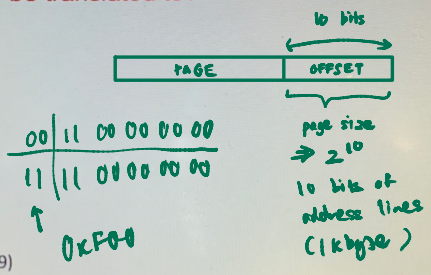
A. 1 KByte
✓B. 2 Kbyte
C. 4 KByte
D. Cannot be determined

Phy Frame can not be < vir memory
⇒ lose memory

Paging Scheme
⇒ Virtual Page Size = Phy Frame Size *
64KByte divided into 32 virtual pages
⇒ Each page = 64KByte/32 = 2 Kbyte
⇒ Phy Frame size = 2KByte

In a system with 1 Kbyte page size, if the virtual page 0 is mapped to physical frame 3, what physical address will virtual address 0x300 be translated to?

✓A. 0x0F00
B. 0x3300
C. 0x0600
D. 0x3000



1 Kbyte => 10 bits offset (A0-A9)
⇒ Virtual page and Phy Frame number are derived from address bits 10 and beyond
0x300 = 0011 0000 0000b
⇒ Virtual page 0
⇒ Mapped to Phy Frame 3
⇒ 1111 0000 0000

**Which of the following is/are true for a translation lookaside buffer?

A. Cache for program data
✓B. Implemented with fast memory
✓C. Contains a subset of the content in the paging table
D. Larger than a typical cache

Storage memory not mentioned at all, can't really acct for page table miss

PYP

Consider a computer system with the following characteristics.

- Cache uses physical address to derive its block and tag field
- Main memory access time = 50 ns
- Translation Lookaside Buffer (TLB)
  - TLB access time = 10 ns
  - Average TLB hit rate = 80%
- TLB and Page Table access do not overlap
- Cache
  - Cache access time = 5 ns
  - Average cache hit rate = 90%
- Cache and Main memory access do not overlap

Starting from virtual addresses of code and data, what would be the effective access time of this hierarchical memory system?

cache hit ↓ (0.9)(5ns) + cache miss ↓ (0.1)(50+5)
↑ cache access time; ↑ mm access; ↑ cache access

CPU → TLB → Main Mem

↓

access TLB first, then proceed with cache

First scenario

best case:

TLB hit + cache hit → probability

Second scenario

TLB hit + cache miss

third

TLB miss + cache hit

fourth

TLB miss + cache miss