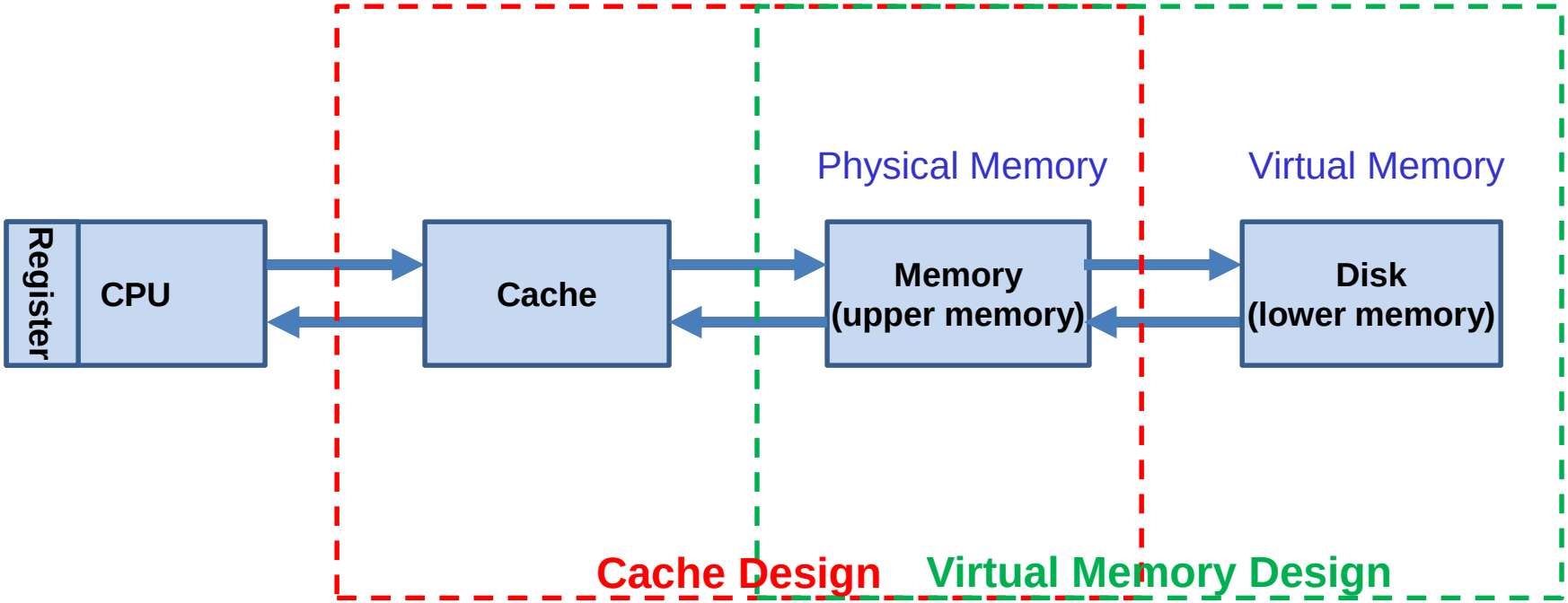
Chapter 9: Virtual Memory and TLB

Covered Topics

1. Virtual Memory
2. Virtual memory paging
3. Translation Lookaside Buffer (TLB)
4. LRU stack

– Chapter 6 by Null
– Chapter 14 by Berger (Available online)

Accessing Memories in a Computer System



Virtual Memory

- Virtual memory provides the method to **map lower memory to upper memory** in the hierarchy
  - Large quantities of slower memory (disk) supplies instructions and data to physical memory
  - Allows programs to **pretend** that they have **larger memory** resources to run in and store data

- The relationship between **virtual memory** (large) and **physical memory** (small) is similar to that of **main memory** (large) and an **associative cache** (small)
  - Any page in a virtual memory can be **mapped to any** page frame on a physical memory
  - **No need of search**: you can **directly translate** the address in virtual memory to physical memory using the ***page table***

Paging

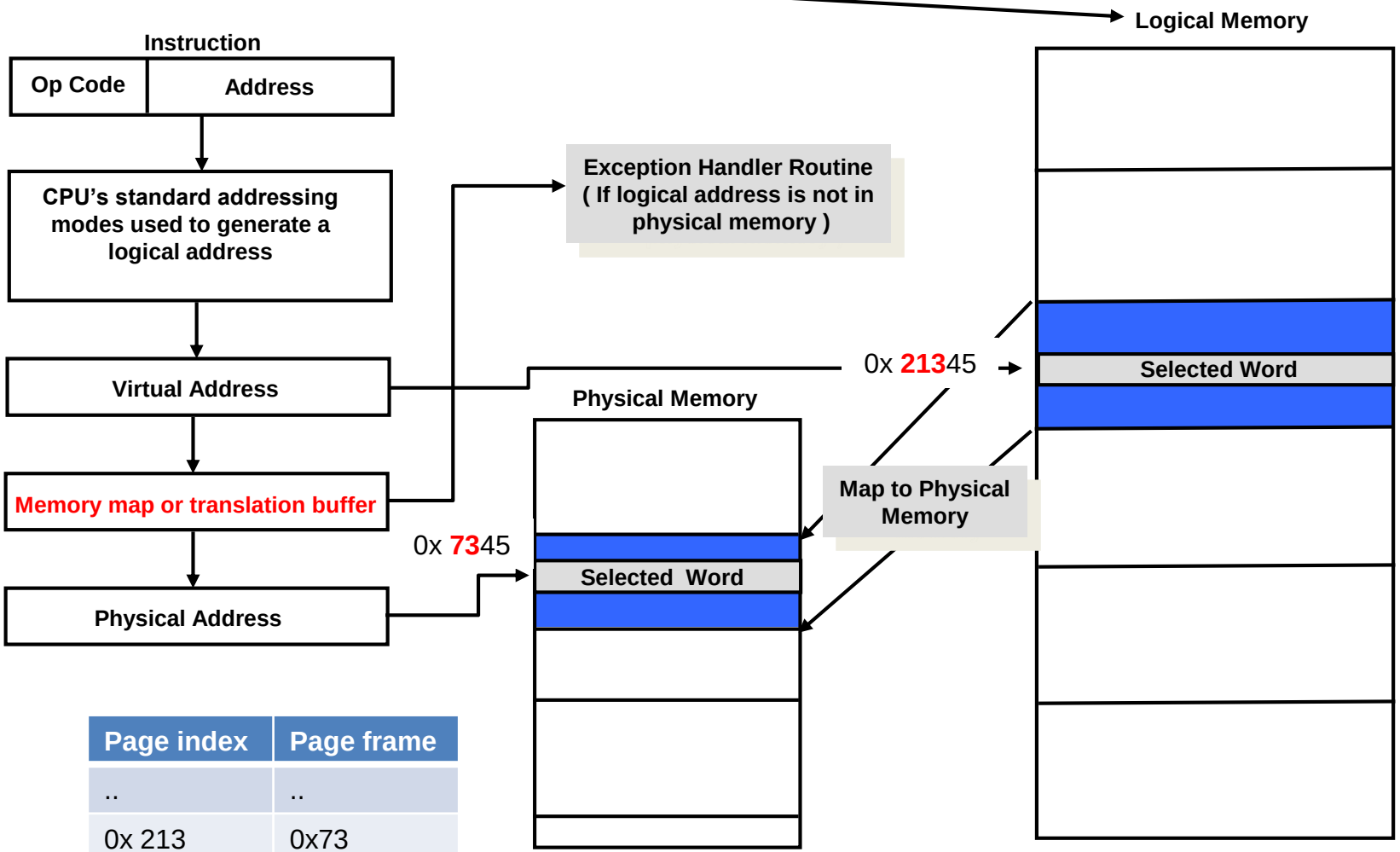
- **Pages**
  - Blocks of memory that can map as a **the minimum unit** between **physical memory** and **secondary storage** (e.g., disk)
- Virtual memory system only loads **specific parts (pages)** of the program into physical memory
  - ***Pages that has been allocated*** by the operating system
  - ***Pages that is currently in use*** (hit)
- Virtual (Logical) address is divided into **two parts**
  - ***Virtual page number*** (or page number)
  - ***Word offset*** (byte offset) within the page
  - The **page number** in **virtual** address is translated into a **frame number** in **physical** address
  - The offset is preserved

Paging: Define Terms

- ***Demand-paged virtual memory***
  - O/S loads the pages into physical memory on demand
- ***Page Number (Virtual Page Number)***
  - Most significant (upper) bits of the virtual address
- ***Word Offset****:* Byte or word address within a page
  - Least significant (lower) bits within the page
- ***Frame***: One unit of physical memory that holds a page
  - Always loads onto page boundaries
  - Tend to be from 512 bytes to 4096 bytes
- ***Page Map***
  - Memory map that stores relationship (mapping) between pages in virtual memory and frames in physical memory

Components of a Virtual Memory System

- ***Virtual (Logical) Memory*** is the memory space that the program thinks it is available to use



Page index	Page frame
..	..
0x 213	0x73
..	..

Page Table: Keeping Track of Pages

- ***Page table*:** The part of the page map owned by the O/S
- ***Page table entries**:* Holds information about a specific page
  - ***Virtual page number***
    - Index in page table
  - ***Page frame number***
    - Page frame number in physical memory
  - ***Validity bit**:* Whether or not the page is currently in physical memory
  - ***Dirty bit**:* Whether or not the program has modified (written to) the page
  - ***Protection bits**:* Which user (process or program) may access the page
    - Linux Users: what CHMOD really does

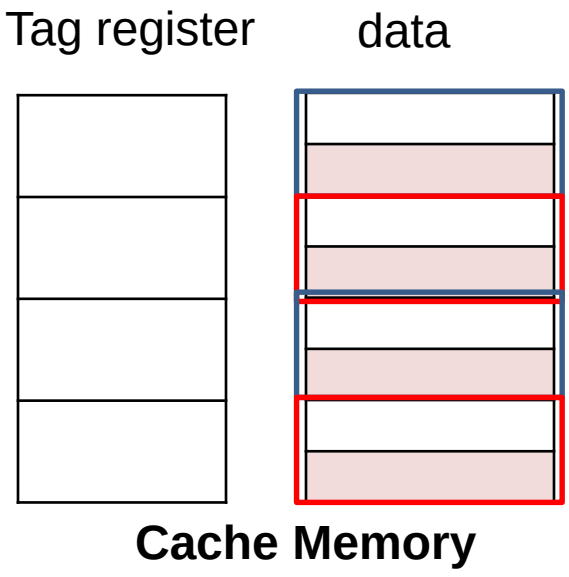
- For simplicity, we will use the following table for page table.

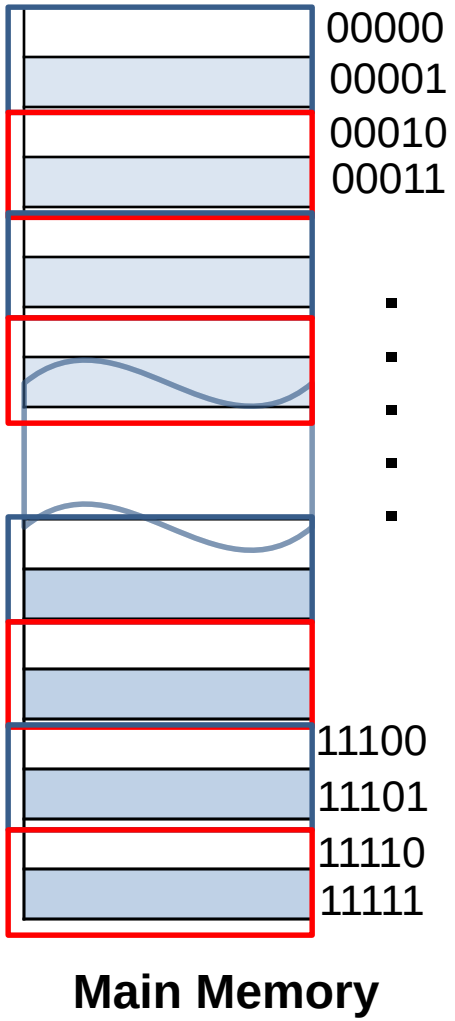
Page (virtual)	Frame (physical)	Validity bit
0	0	1
1	--	0
2	1	1

Associative Cache vs. Virtual Memory

Associative Cache vs. Virtual Memory

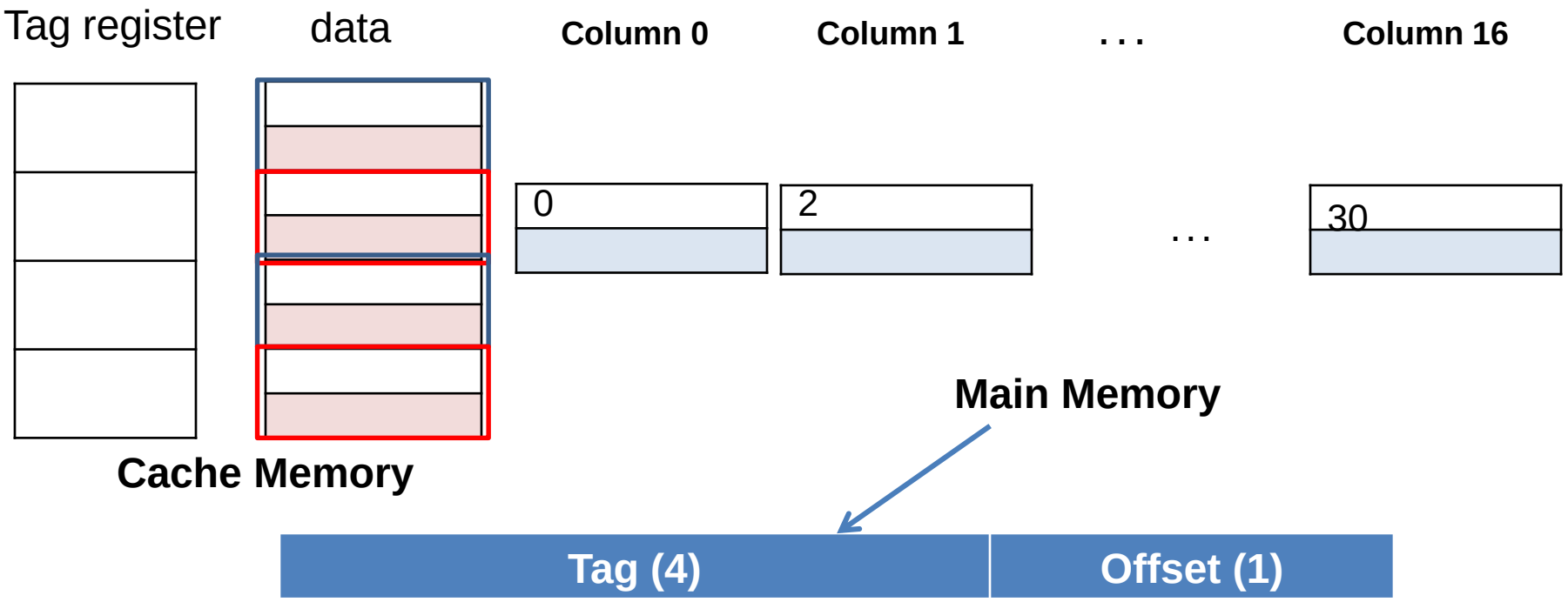
- The main memory size is 32 bytes
- The size of cache is 8 bytes
- The size of block is 2 bytes
- Main memory is divided with the same size of blocks





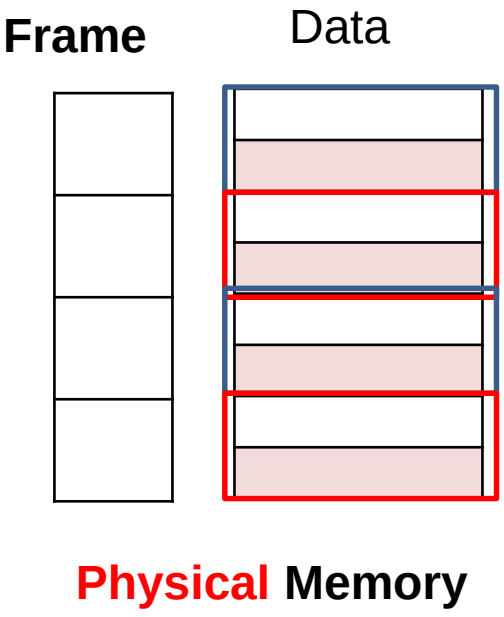
Associative Cache vs. Virtual Memory

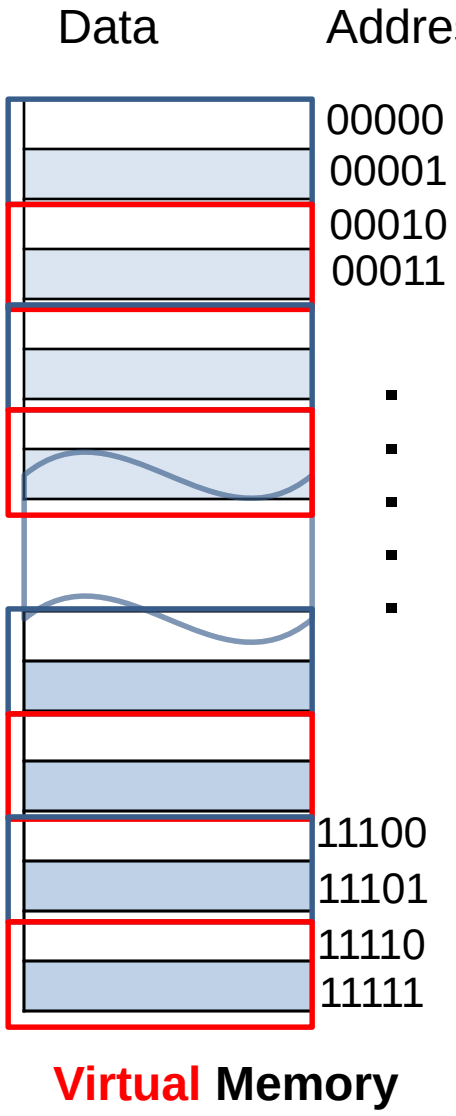
- Rearrange the main memory based on the size of block (refill line)
- Conceptually the number of columns = number of blocks
- Each block can be mapped to any available block



Associative Cache vs. Virtual Memory

- The virtual memory size is 32 bytes
- The size of physical memory is 8 bytes
- The size of page frame is 2 bytes

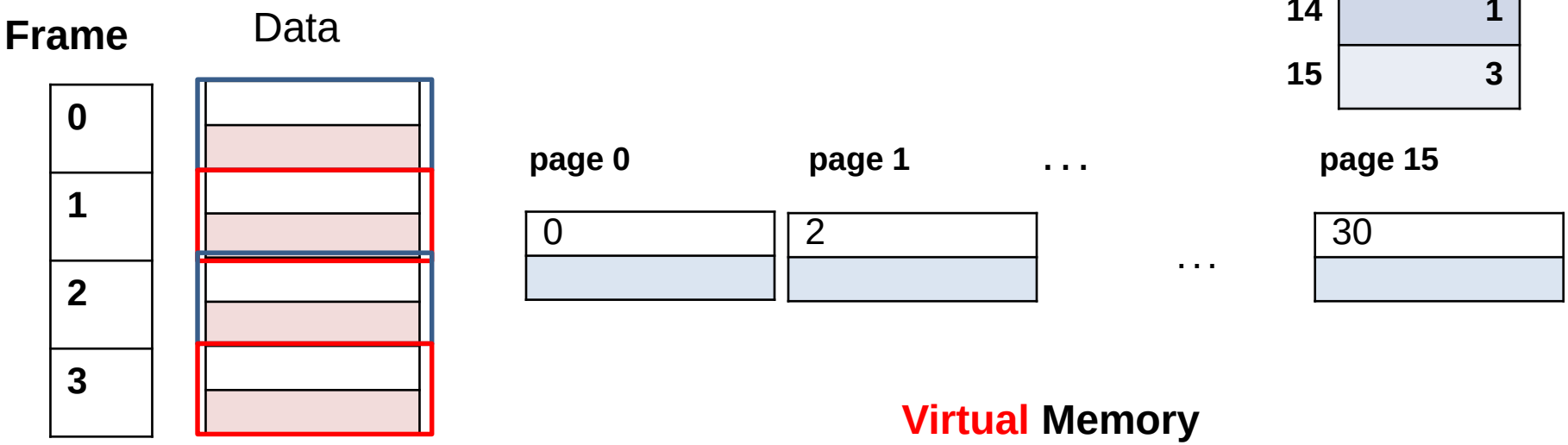




Associative Cache vs. Virtual Memory

- The number of pages = VM size / frame size = 32/2 = 16
- The number of frames = PM size / frame size = 8/2 = 4
- Map the page into one of the 4 frames (page table)

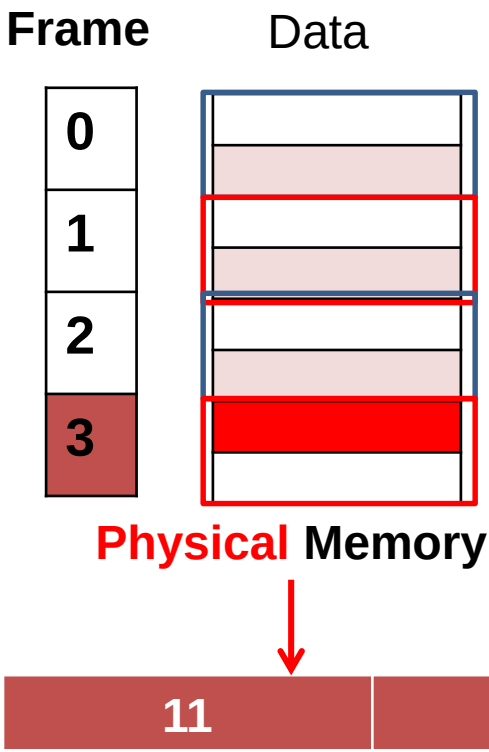
Page (virtual)	Frame (physical)
0	0
1	--
2	2
..	..
14	1
15	3

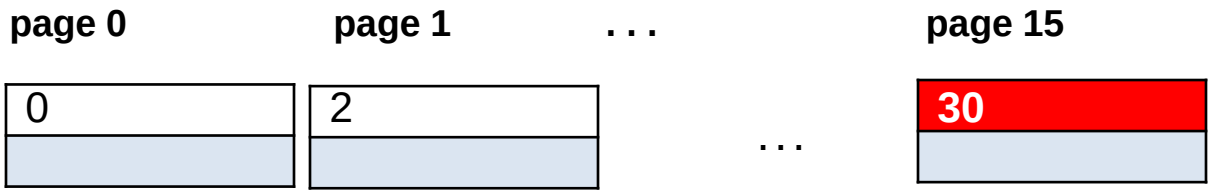


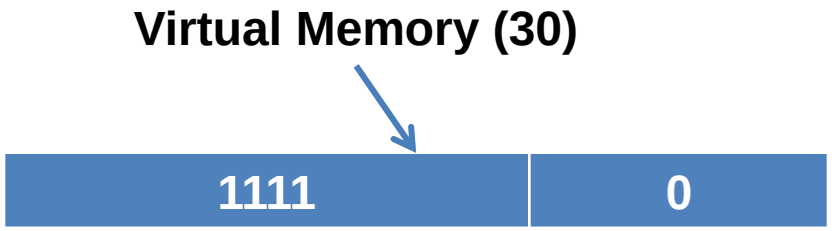
Associative Cache vs. Virtual Memory

- The number of pages = VM size / frame size = 32/2 = 16
- The number of frames = PM size / frame size = 8/2 = 4
- Map the page into one of the 4 frames (page table)

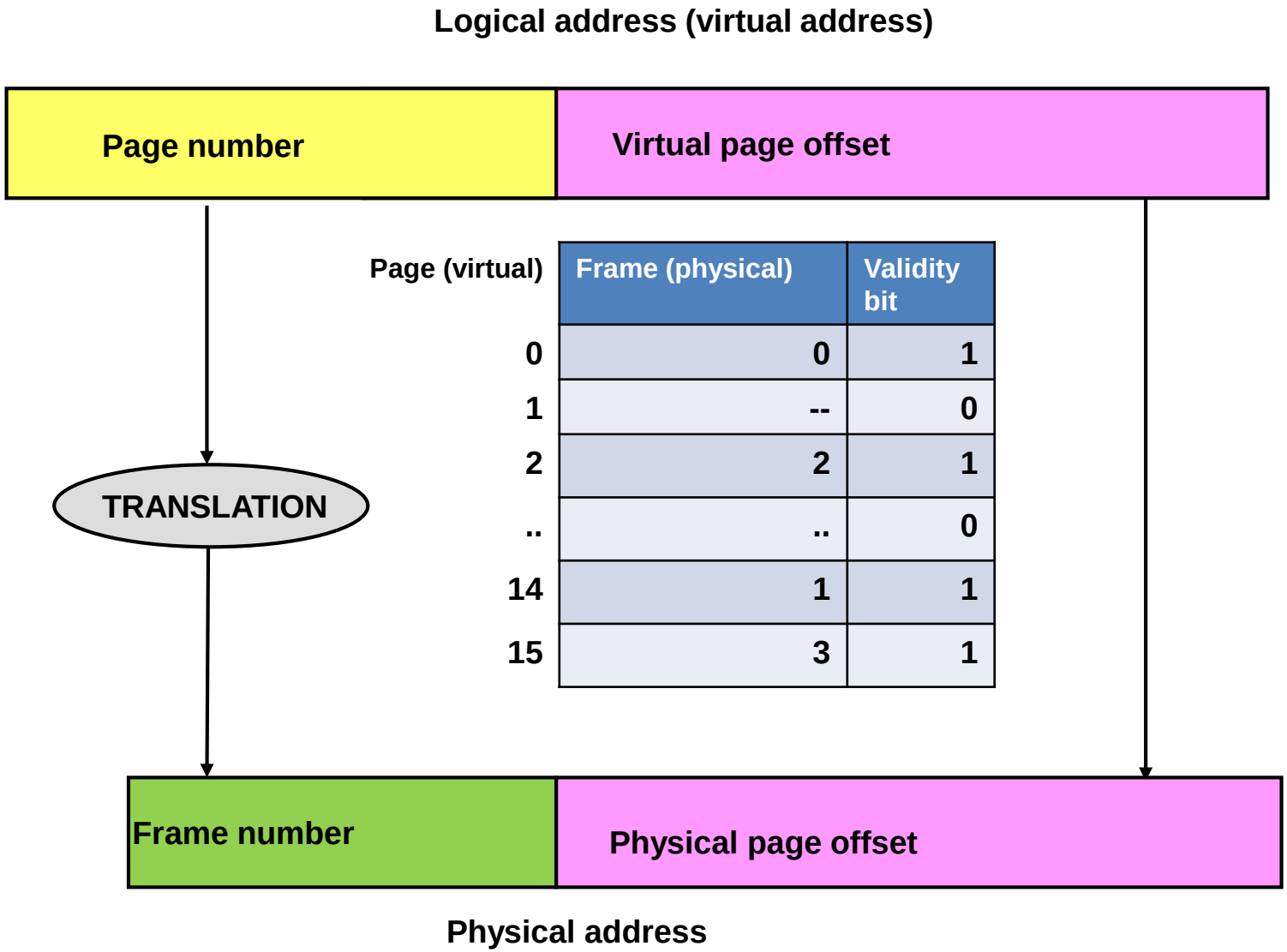
Page (virtual)	Frame (physical)
0	0
1	--
2	2
..	..
14	1
15	3







Page Table: Map Pages to Frames (O/S)



Class Exercise

- Suppose a system has a **virtual address space** of **8K** and a **physical address space** of **4K**, and the system uses **byte addressing**. The **size of page frame** is **1K bytes**.

Q1: How many bits in a physical address?

Q2: How many page frames in physical memory?

Q3: How many bits for frame bits in physical address?

Q4: How many bits in a virtual address?

Q5: How many pages in virtual memory?

Q6: How many bits for page bits in virtual address?

Class Exercise

- Suppose a system has a **virtual address space** of **8K** and a **physical address space** of **4K**, and the system uses **byte addressing**. The **size of page frame** is **1K bytes**.

Q1: How many bits in a physical address?

Since the physical address space is 4K, 12 bits for physical address

Q2: How many page frames in physical memory?

The size of a page frame is 1K = 2^{10}, then $2^{12}/2^{10} = 2^2$ number of frames.

Q3: How many bits for frame bits in physical address?

Based on the answer of Q2, 2 bits for page frame.

Q4: How many bits in a virtual address?

Since it is 8K, 13 bits for virtual address.

Q5: How many pages in virtual memory?

The size of a page frame is 1K = 2^{10}, then $2^{13}/2^{10} = 2^3$ number of pages.

Q6: How many page bits in virtual address?

Based on the answer of Q5, 3 bits for page in virtual address.

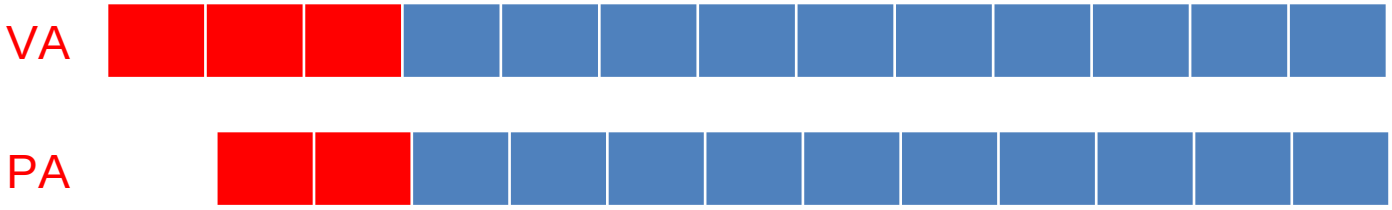
Class Exercise

Map the logical address **0x0404** into physical address using the following page table.

Page (virtual)	Frame (physical)	Validity bit
0	--	0
1	3	1
2	0	1
3	--	0
4	--	0
5	1	1
6	2	1
7	--	0

Class Exercise

Map the logical address **0x0404** into physical address using the following page table.



Page (virtual)	Frame (physical)	Validity bit
0	--	0
1	3	1
2	0	1
3	--	0
4	--	0
5	1	1
6	2	1
7	--	0

Since the page number 1, which corresponds to frame number 3, the frame bits should be 11_2

Therefore, logical address 0x0404 is translated to physical address 0xC04

Class Exercise

Map the logical address **0x1553** into physical address using the following page table.

Page (virtual)	Frame (physical)	Validity bit
0	--	0
1	3	1
2	0	1
3	--	0
4	--	0
5	1	1
6	2	1
7	--	0

Class Exercise

Map the logical address **0x1553** into physical address using the following page table.

Page (virtual)	Frame (physical)	Validity bit
0	--	0
1	3	1
2	0	1
3	--	0
4	--	0
5	1	1
6	2	1
7	--	0

Since the page number 5, which corresponds to frame number 1, the frame bits should be 1_2

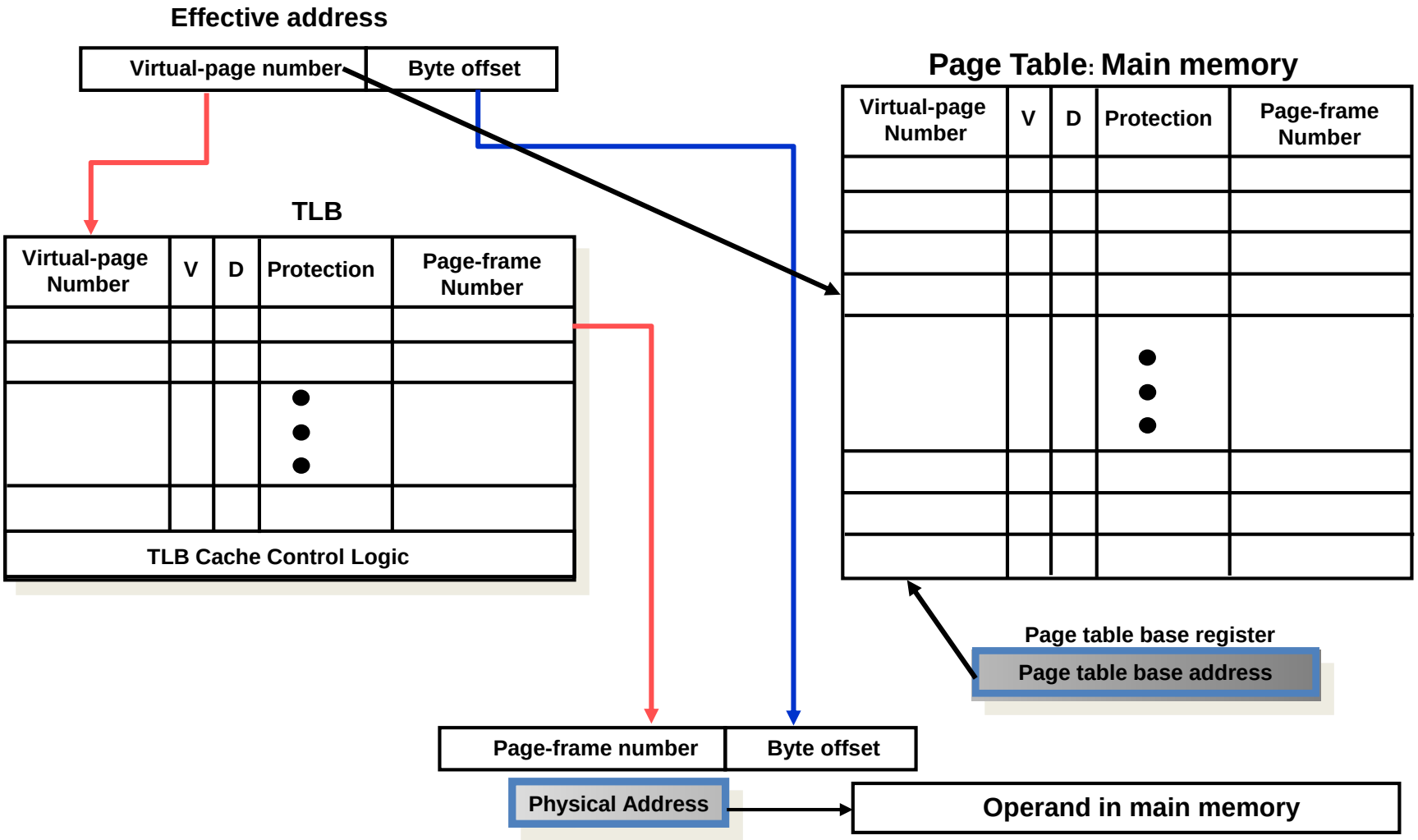
Therefore, logical address 0x1553 is translated to physical address 0x553

Translation Lookaside Buffer (TLB)

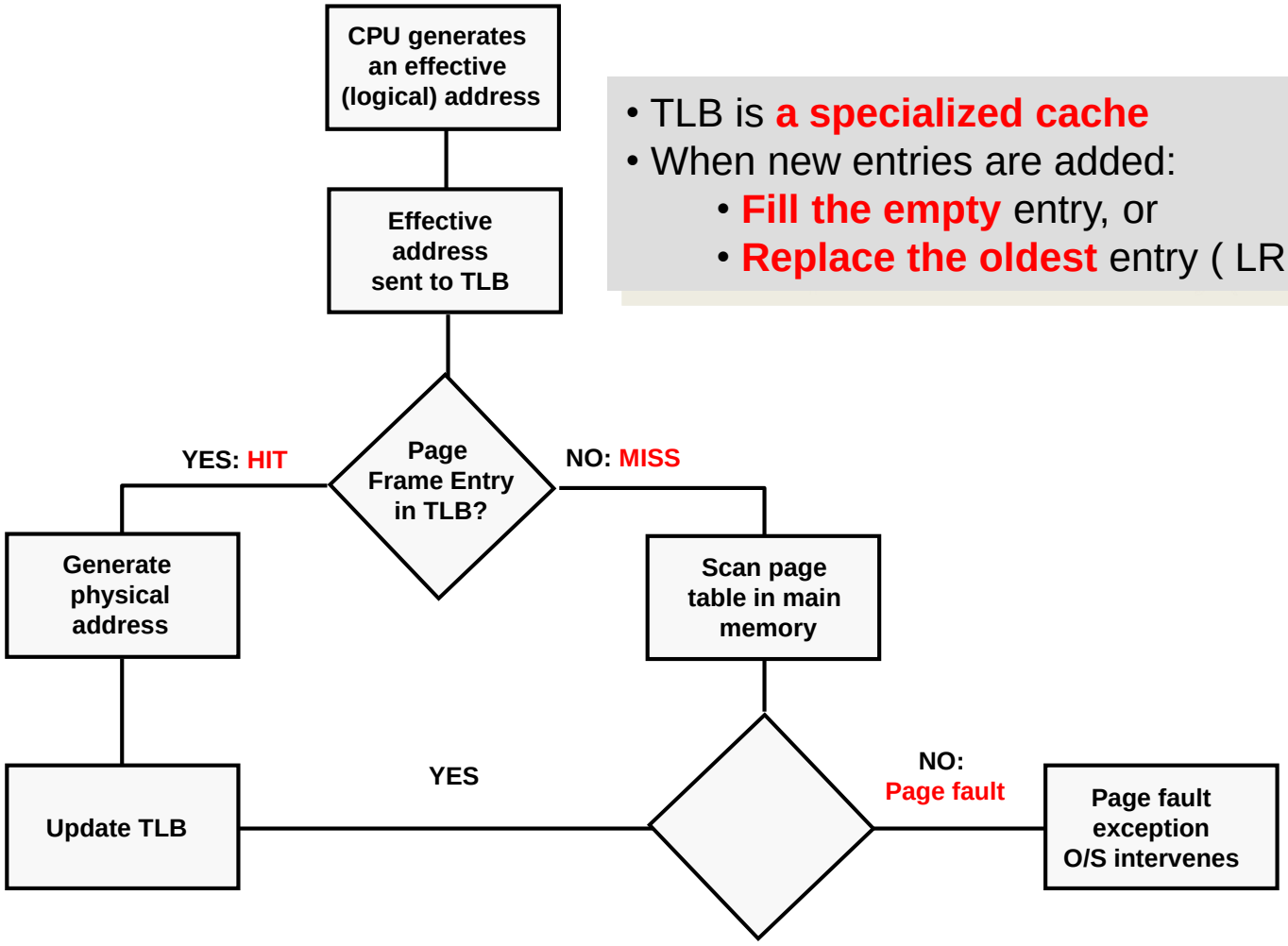
- Most computer systems keep their page tables in main memory
  - ***Page-table base register*** points to the beginning of the table
  - O/S can modify the page table base register using supervisor mode instructions
  - In theory, main memory (non-cached) accesses could take **twice** as long, because the page table must be accessed first

- Modern processors maintain a ***Translation Lookaside Buffer (TLB)***
  - TLB holds partial **information of the page table (less rows)**
    - ***cache for page table***
  - TLB cache algorithm holds **only most recently accessed pages**
    - Flushes ***Least Recently Used*** *(LRU)* entries from TLB
    - Holds **only mapping for valid pages**

Components of a Paging System



Virtual Paging Process



- TLB is **a specialized cache**
- When new entries are added:
  - **Fill the empty** entry, or
  - **Replace the oldest** entry (LRU)

Replacement Policy

- When all the page frames of physical memory are full, we need to find a "***victim***" frame to throw out, where we need a ***Replacement Policy***

- *Optimal replacement policy*: impossible to implement as it should predict all the future memory references completely

- **The replacement policy also applies to the caches as well**

Three Most Common Replacement Policies

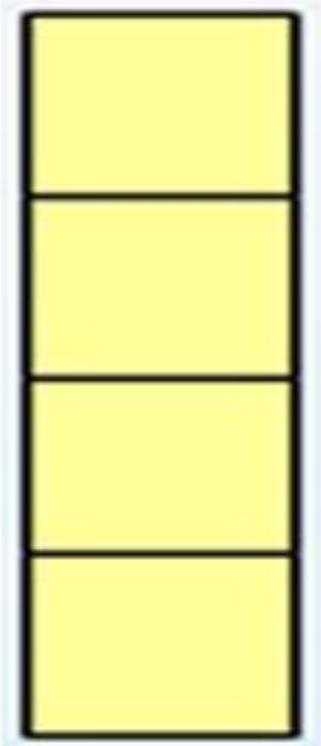
- ***First-In, First-Out* (FIFO)**
  - Throw away the frame that has been in the cache the longest, regardless of when it was last used

- ***Random***
  - Picks a frame at random and replaces it with a new one
  - Disadvantage: Can certainly evict a frame that will be needed often or needed soon

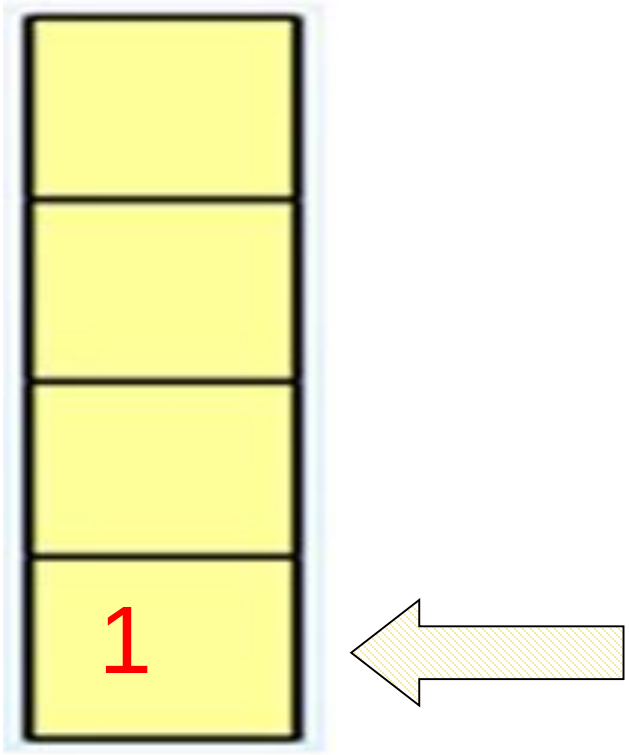
- ***Least Recently Used* (LRU)**
  - Evicts the frame that has been unused for the longest period of time
  - Disadvantage: Complex, as it has to maintain an access history for each block, which ultimately slows down the cache.

Implementing an LRU Stack

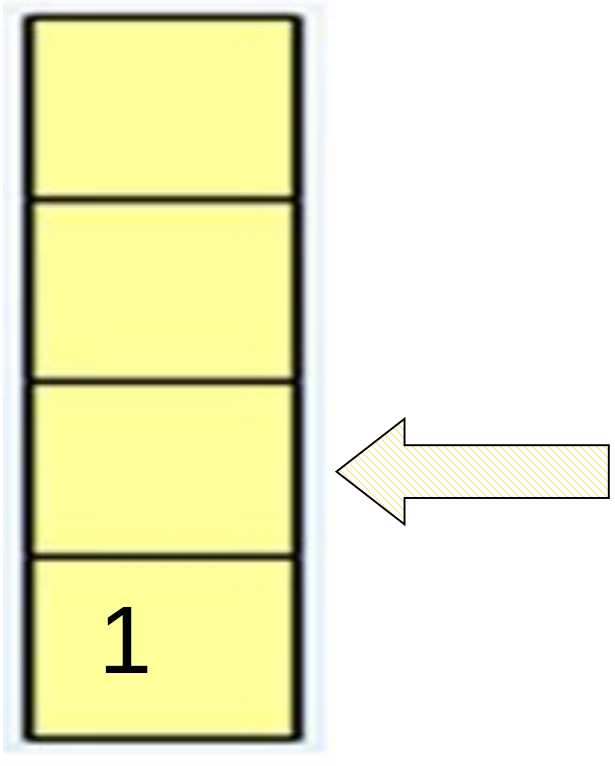
REF string: 1 5 3 1 6 2 3



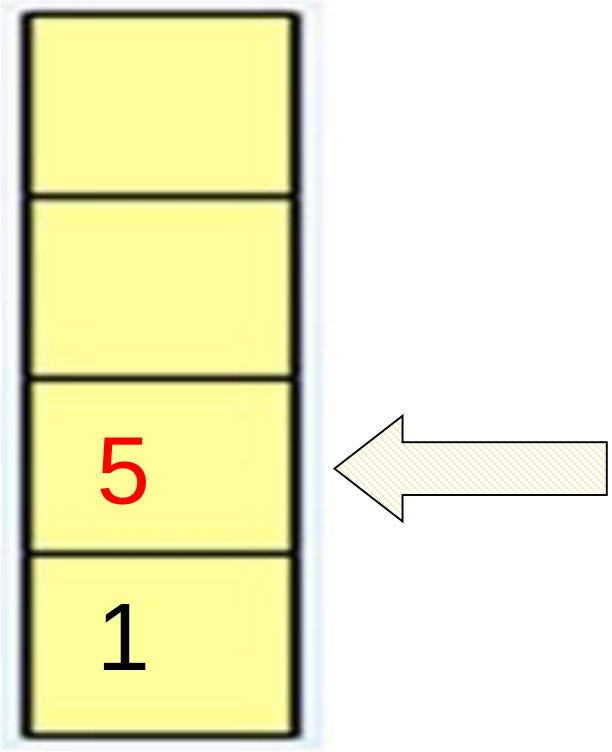
REF string: 1 5 3 1 6 2 3



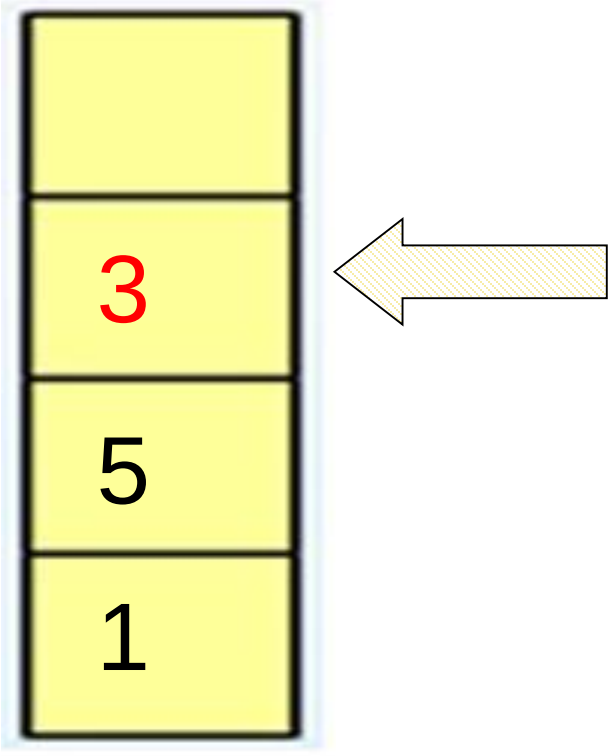
REF string: 1 5 3 1 6 2 3



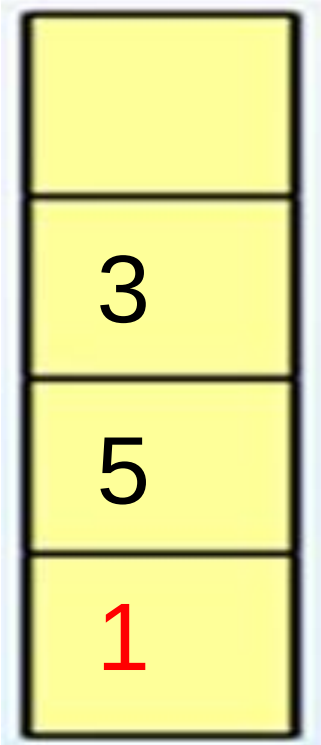
REF string: 1 5 3 1 6 2 3



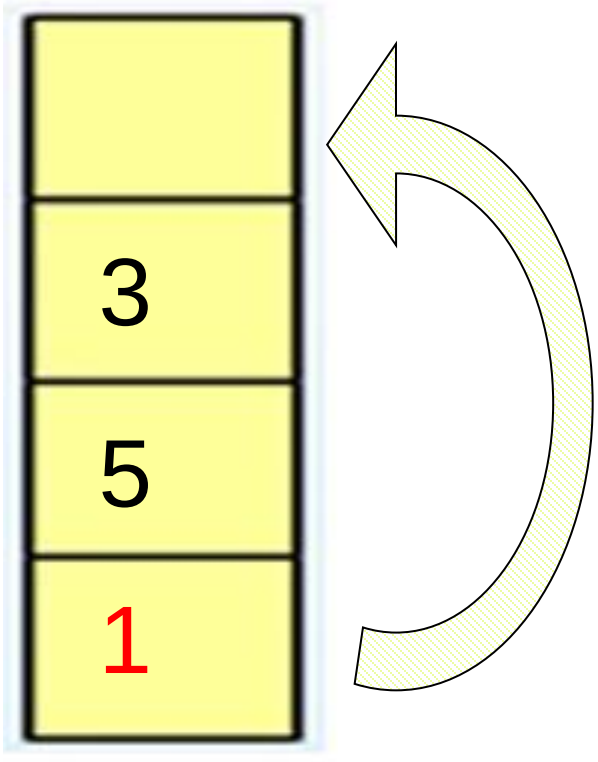
REF string: 1 5 3 1 6 2 3



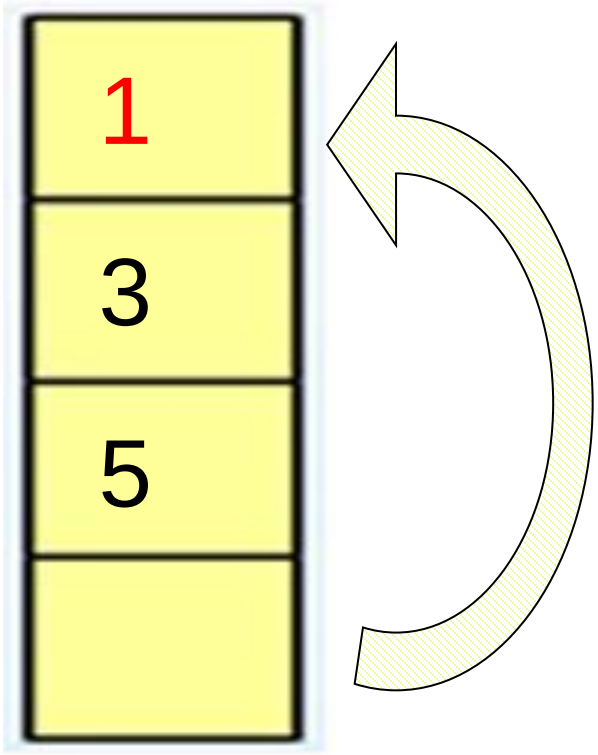
REF string: 1 5 3 1 6 2 3



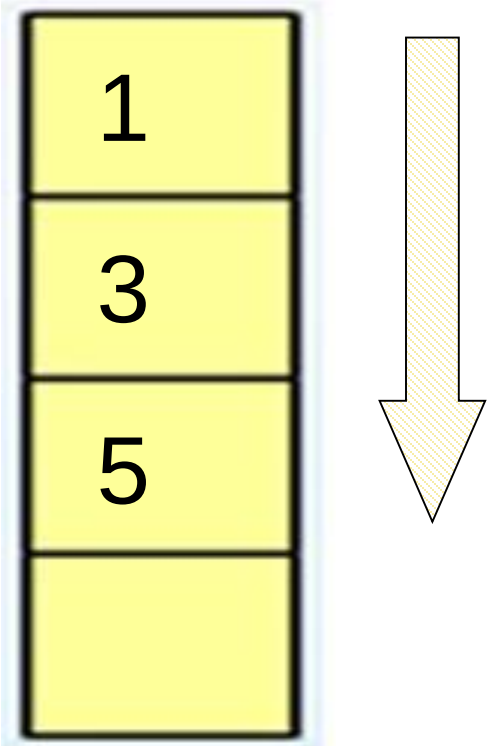
REF string: 1 5 3 1 6 2 3



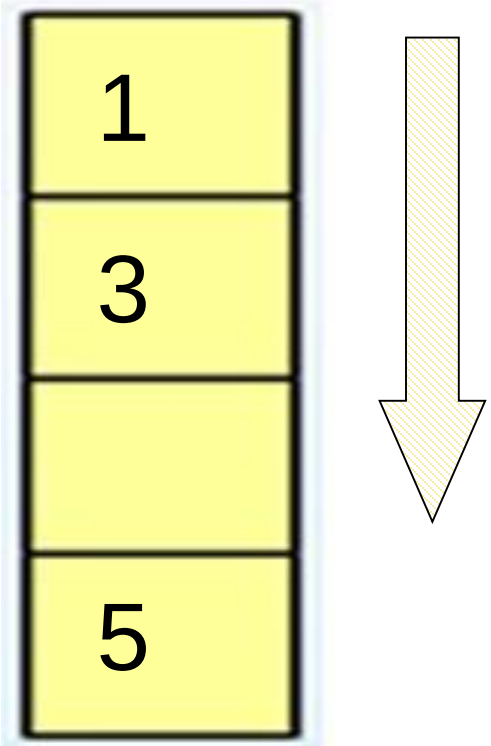
REF string: 1 5 3 1 6 2 3



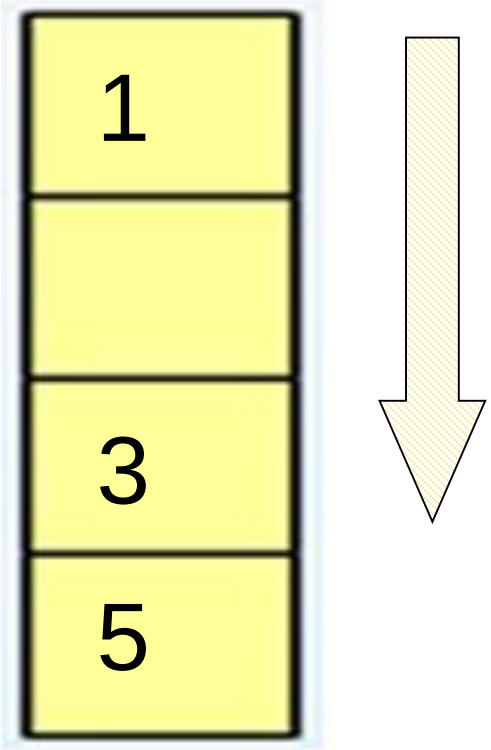
REF string: 1 5 3 1 6 2 3



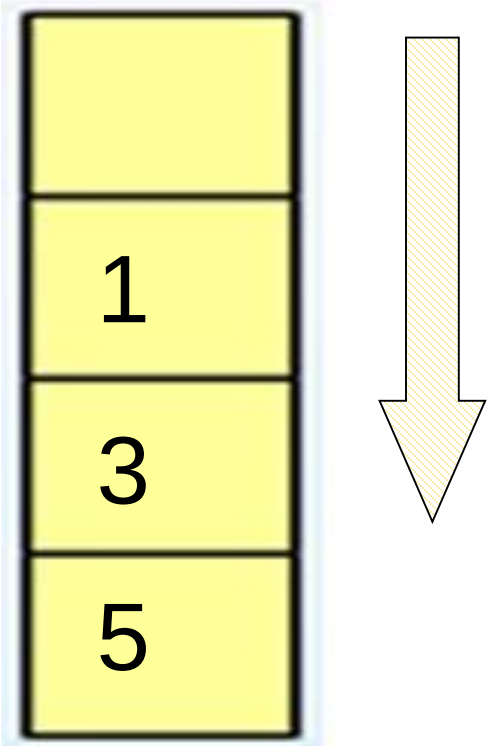
REF string: 1 5 3 1 6 2 3



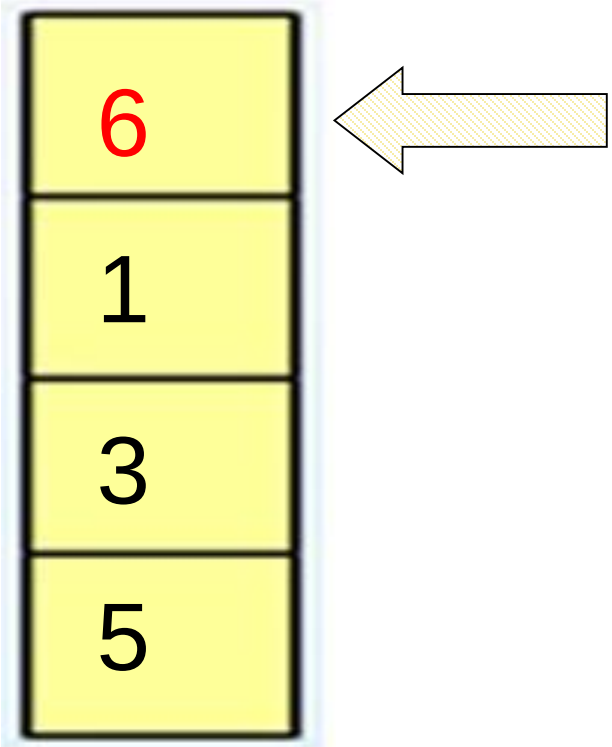
REF string: 1 5 3 1 6 2 3



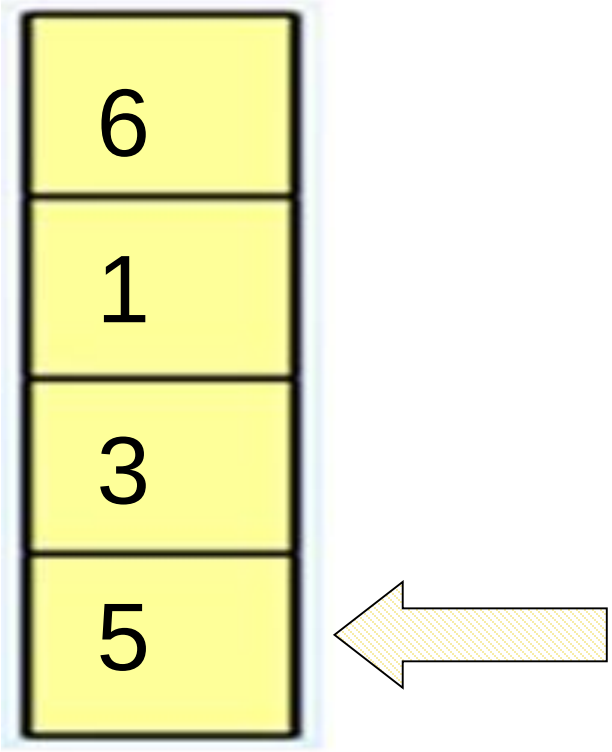
REF string: 1 5 3 1 6 2 3



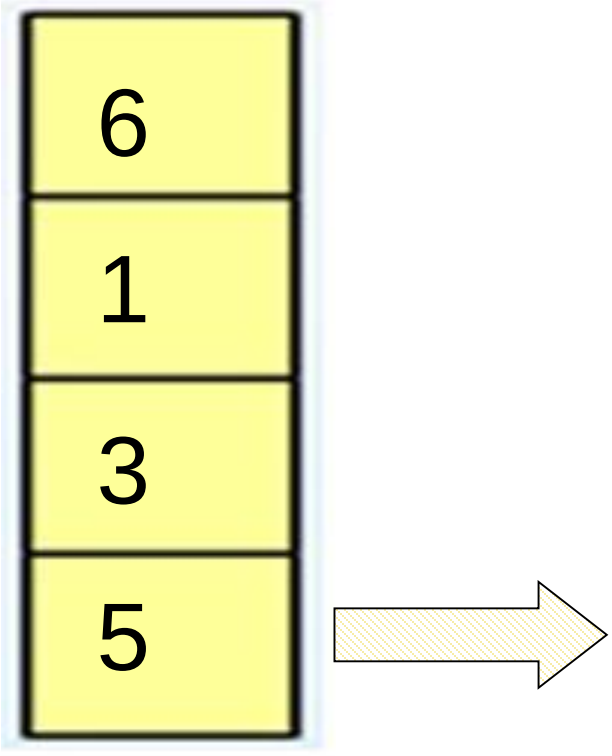
REF string: 1 5 3 1 6 2 3



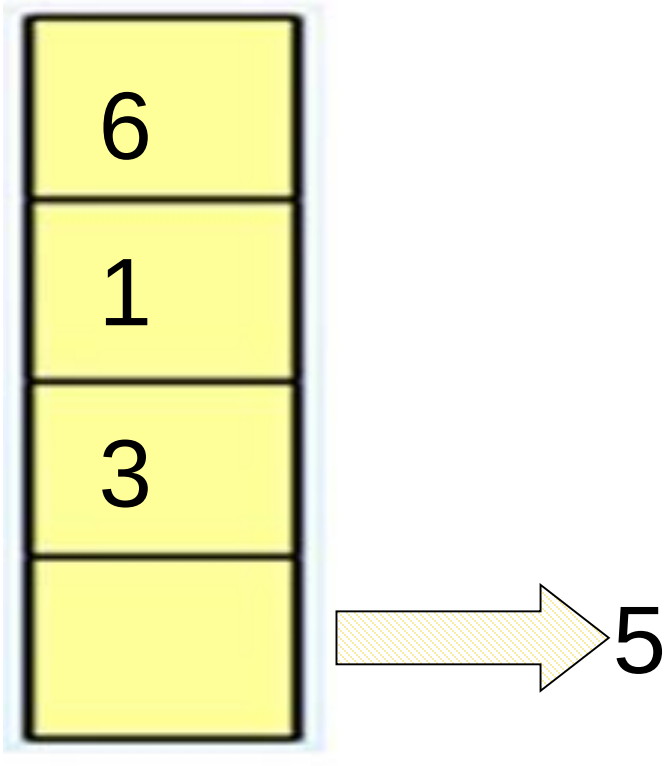
REF string: 1 5 3 1 6 2 3



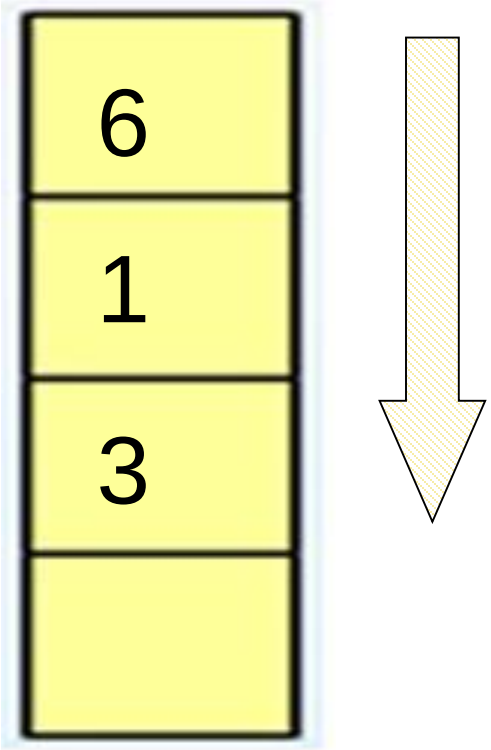
REF string: 1 5 3 1 6 2 3



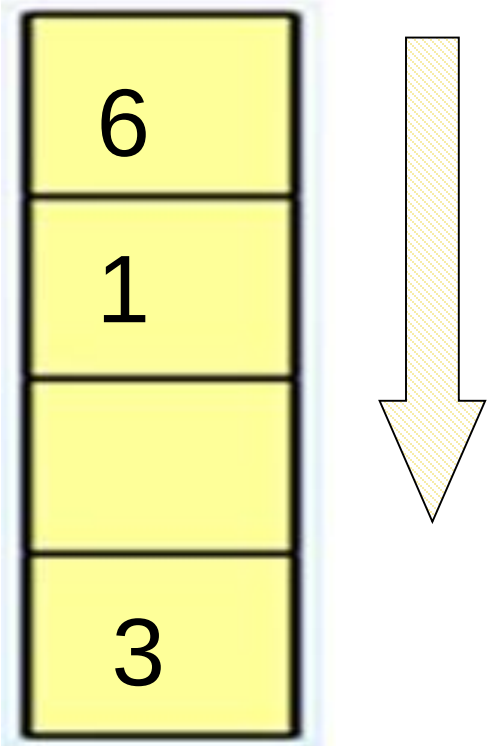
REF string: 1 5 3 1 6 2 3



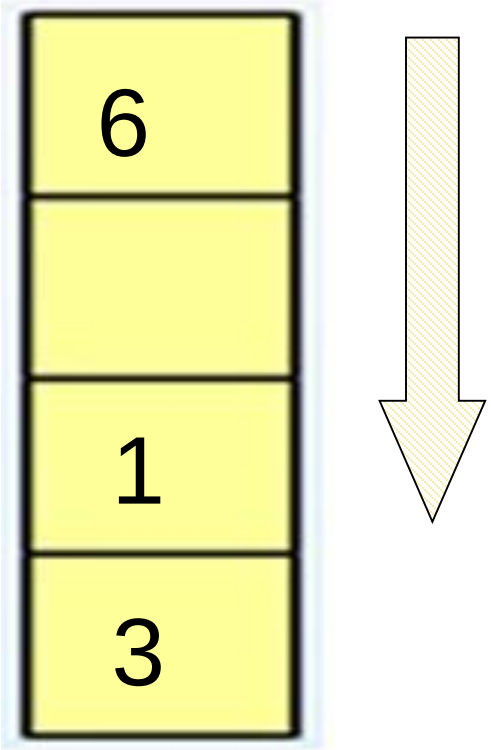
REF string: 1 5 3 1 6 2 3



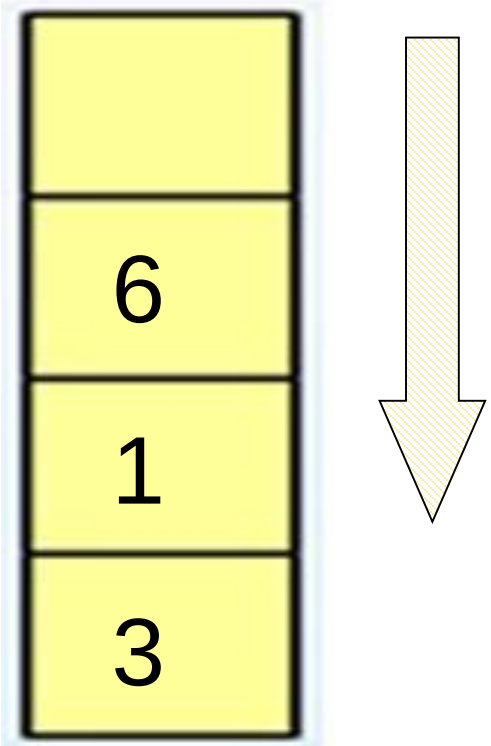
REF string: 1 5 3 1 6 2 3



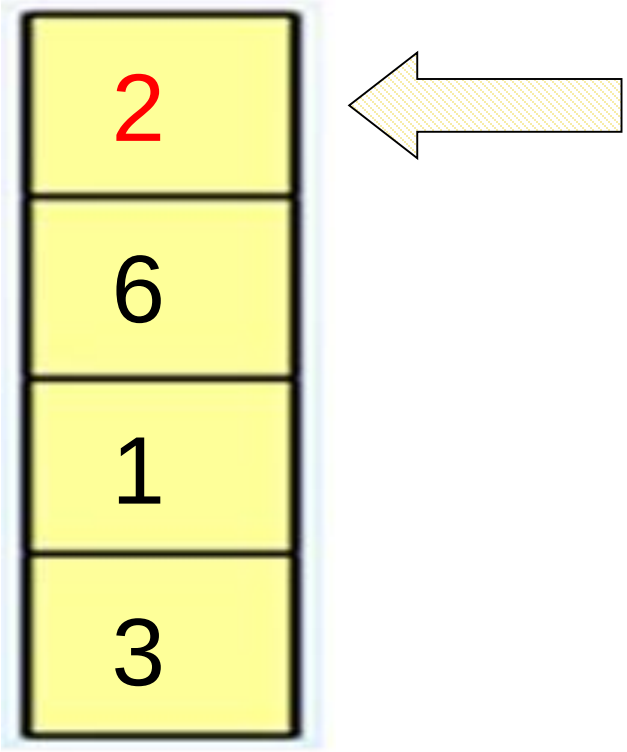
REF string: 1 5 3 1 6 2 3



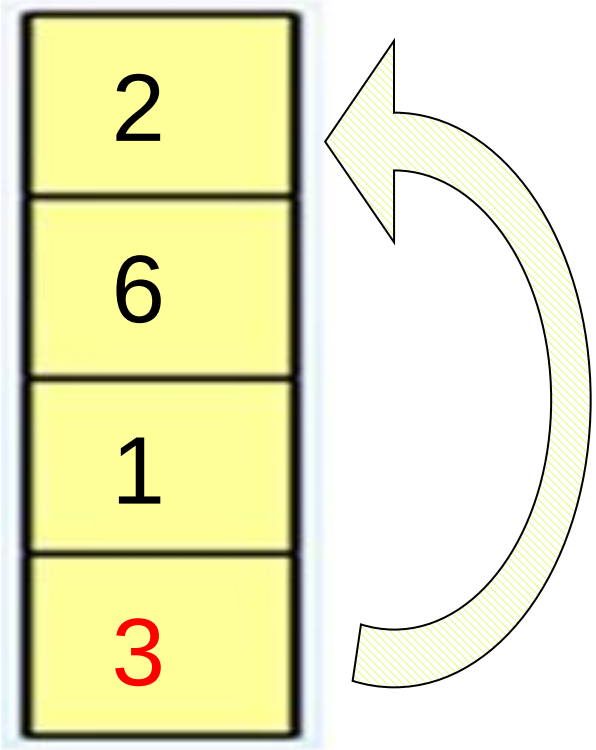
REF string: 1 5 3 1 6 2 3



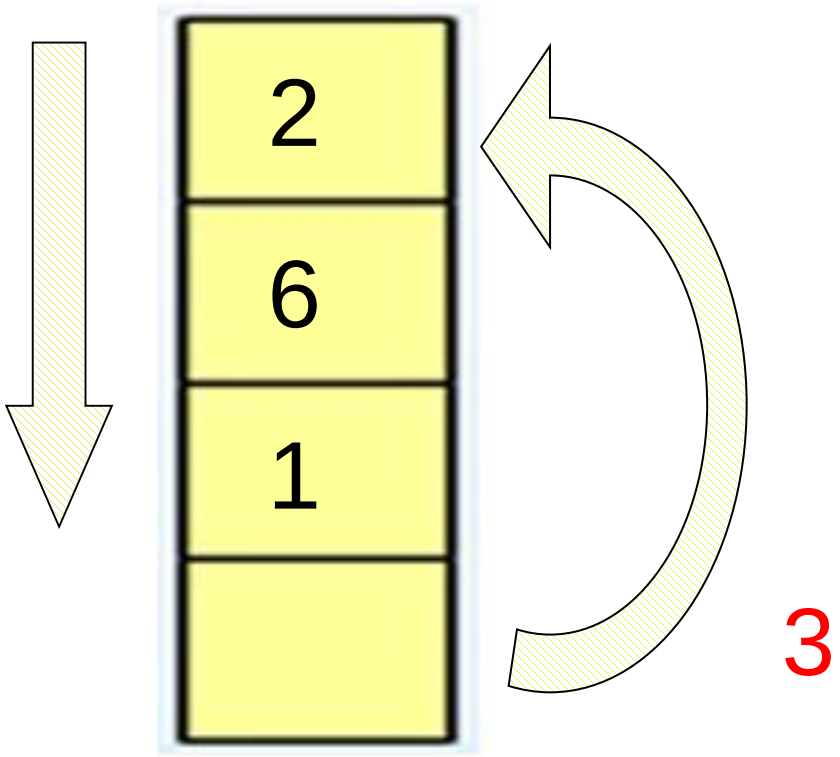
REF string: 1 5 3 1 6 2 3



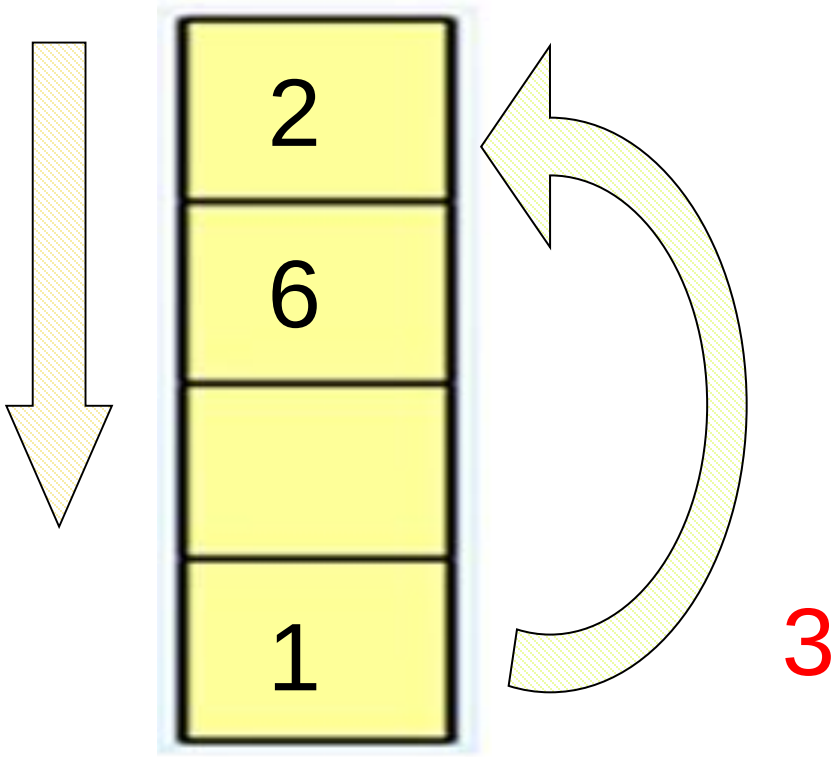
REF string: 1 5 3 1 6 2 3



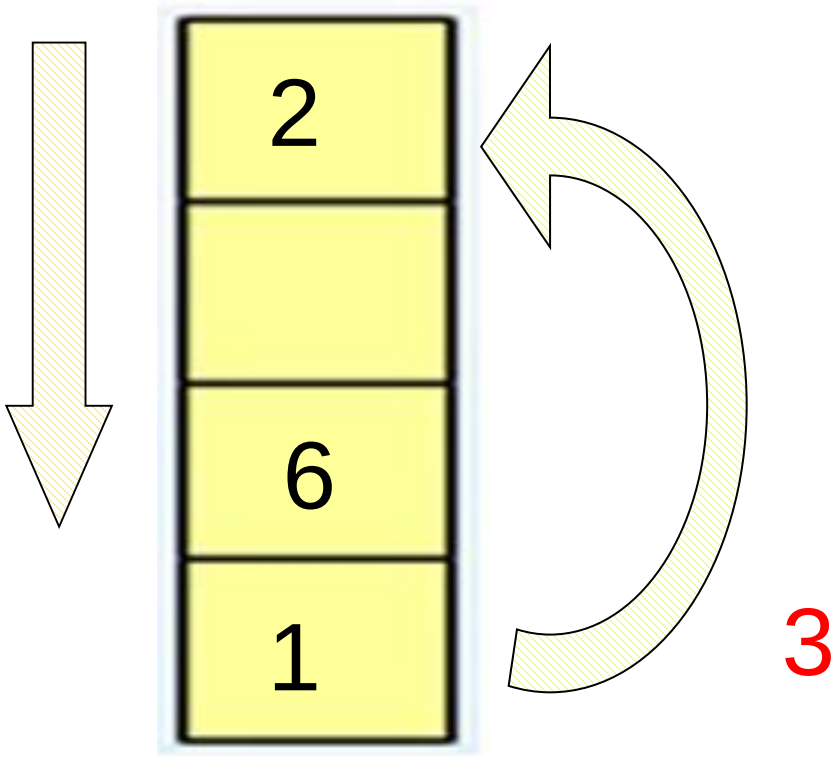
REF string: 1 5 3 1 6 2 3



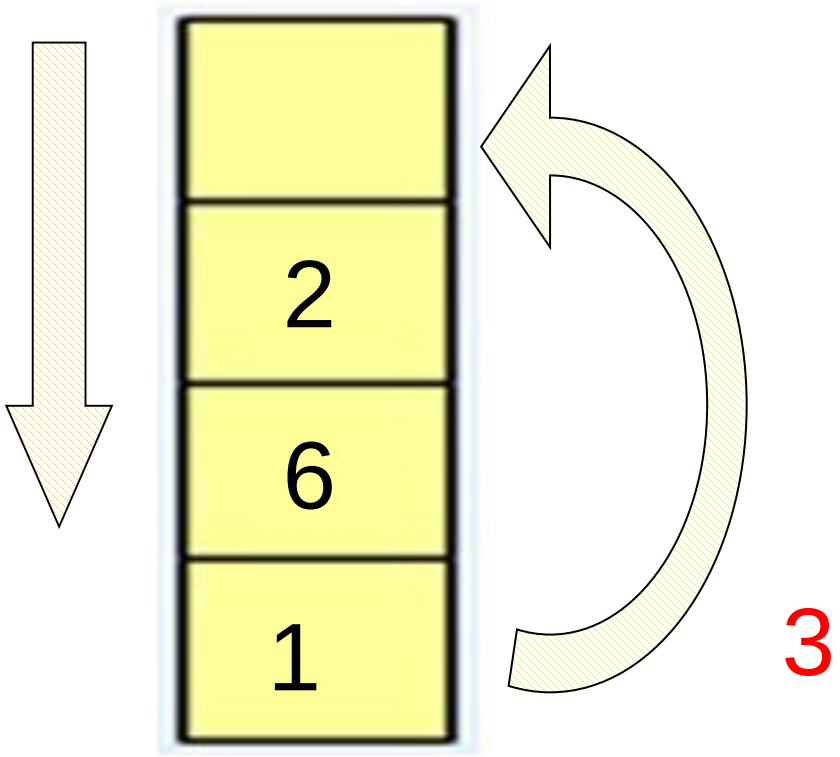
REF string: 1 5 3 1 6 2 3



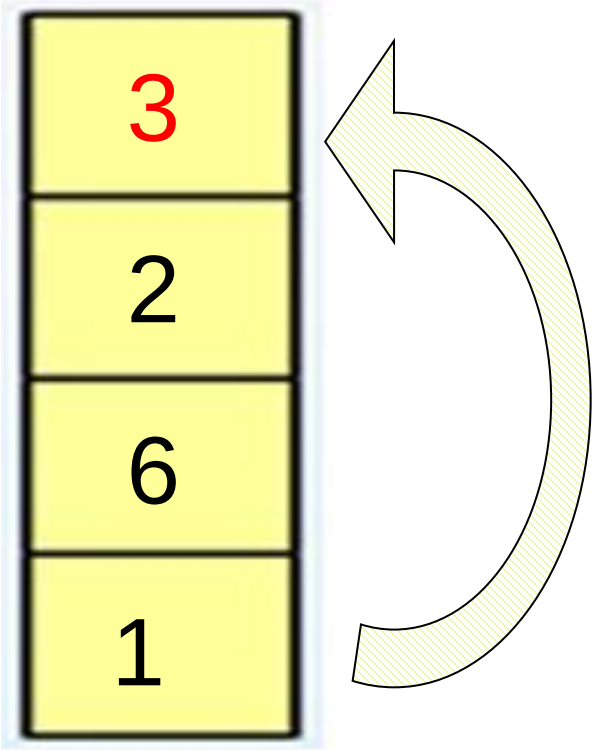
REF string: 1 5 3 1 6 2 3



REF string: 1 5 3 1 6 2 3



REF string: 1 5 3 1 6 2 3



TLB LRU Stack

- TLB LRU stack
  - TLB stores **only a part of** page table information
  - When TLB is full, the victim page is the page at the bottom of the TLB LRU stack
  - The victim page is removed from the stack and the newly used one goes on top of the stack. The rest are shifted to down

TLB

4	3
6	2

page 1 is referenced

TLB LRU Stack

6
4

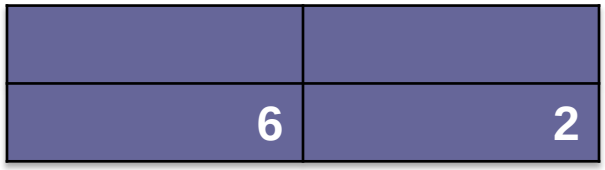
Page table

page	frame
0	--
1	0
2	1
3	--
4	3
5	--
6	2
7	--

TLB LRU Stack

- TLB LRU stack
  - TLB stores **only a part of** page table information
  - When TLB is full, the victim page is the page at the bottom of the TLB LRU stack
  - The victim page is removed from the stack and the newly used one goes on top of the stack. The rest are shifted to down

TLB



TLB LRU Stack

page 1 is referenced

page 4 is removed

Page table

page	frame
0	--
1	0
2	1
3	--
4	3
5	--
6	2
7	--

TLB LRU Stack

- TLB LRU stack
  - TLB stores **only a part of** page table information
  - When TLB is full, the victim page is the page at the bottom of the TLB LRU stack
  - The victim page is removed from the stack and the newly used one goes on top of the stack. The rest are shifted to down

TLB

1	0
6	2

TLB LRU Stack

6

page 1 is referenced

page 4 is removed

page 1 is inserted

Page table

page	frame
0	--
1	0
2	1
3	--
4	3
5	--
6	2
7	--

TLB LRU Stack

- TLB LRU stack
  - TLB stores **only a part of** page table information
  - When TLB is full, the victim page is the page at the bottom of the TLB LRU stack
  - The victim page is removed from the stack and the newly used one goes on top of the stack. The rest are shifted to down

TLB

1	0
6	2

TLB LRU Stack

1
6

page 1 is referenced

page 4 is removed

page 1 is inserted

TLB LRU is updated

Page table

page	frame
0	--
1	0
2	1
3	--
4	3
5	--
6	2
7	--

Mem LRU Stack

- Mem LRU stack
  - Some pages are not mapped with frames in the page table
  - If the unmapped page is referenced, a page fault occurs
  - When a page fault happened, the victim page is at the bottom of the Mem LRU stack
  - TLB, TLB LRU, Page table, and Mem LRU are updated

TLB

4	3
6	2

Mem LRU Stack

6
4
1
2

page 3 is referenced

TLB LRU Stack

6
4

Page table

page	frame
0	--
1	0
2	1
3	--
4	3
5	--
6	2
7	--

Mem LRU Stack

- Mem LRU stack
  - Some pages are not mapped with frames in the page table
  - If the unmapped page is referenced, a page fault occurs
  - When a page fault happened, the victim page is at the bottom of the Mem LRU stack
  - TLB, TLB LRU, Page table, and Mem LRU are updated

TLB

4	3
6	2

Mem LRU Stack

6
4
1
2

page 3 is referenced → Page fault

TLB LRU Stack

6
4

Page table

page	frame
0	--
1	0
2	1
3	**--**
4	3
5	--
6	2
7	--

Mem LRU Stack

- Mem LRU stack
  - Some pages are not mapped with frames in the page table
  - If the unmapped page is referenced, a page fault occurs
  - When a page fault happened, the victim page is at the bottom of the Mem LRU stack
  - TLB, TLB LRU, Page table, and Mem LRU are updated

TLB

4	3
6	2

Mem LRU Stack

6
4
1
2

TLB LRU Stack

6
4

page 3 is referenced → Page fault

page 2 is a victim page

Page table

page	frame
0	--
1	0
2	**1**
3	**--**
4	3
5	--
6	2
7	--

Mem LRU Stack

- Mem LRU stack
  - Some pages are not mapped with frames in the page table
  - If the unmapped page is referenced, a page fault occurs
  - When a page fault happened, the victim page is at the bottom of the Mem LRU stack
  - TLB, TLB LRU, Page table, and Mem LRU are updated

TLB

4	3
6	2

Mem LRU Stack

6
4
1
2

TLB LRU Stack

6
4

page 3 is referenced → Page fault
page 2 is a victim page
page 3 is mapped with frame 1

Page table

page	frame
0	--
1	0
2	**--**
3	**1**
4	3
5	--
6	2
7	--

Mem LRU Stack

- Mem LRU stack
  - Some pages are not mapped with frames in the page table
  - If the unmapped page is referenced, a page fault occurs
  - When a page fault happened, the victim page is at the bottom of the Mem LRU stack
  - TLB, TLB LRU, Page table, and Mem LRU are updated

TLB

3	1
6	2

Mem LRU Stack

3
6
4
1

TLB LRU Stack

3
6

page 3 is referenced → Page fault
page 2 is a victim page
page 3 is mapped with frame 1
Update Mem LRU, TLB, TLB LRU

Page table

page	frame
0	--
1	0
2	**--**
3	**1**
4	3
5	--
6	2
7	--

Class Exercise

Consider a system with 256K bytes of physical memory, the size of each page is 64K bytes, and the virtual address space is 1M. The system is byte-accessible. The processor has a 2 entry fully associative TLB.

Q1) Show the breakdown of the virtual addresses:

Q2) Show the breakdown of the physical addresses:

Class Exercise

Consider a system with 256K bytes of physical memory, the size of each page is 64K bytes, and the virtual address space is 1M. The system is byte-accessible. The processor has a 2 entry fully associative TLB.

Q1) Show the breakdown of the virtual addresses:

Virtual address space is 1MB: so it has 20 bits. The size of each page is 64KB. That is, offset bits is 16, and the other 4 bits are page bits.

page	offset
4	16

Q2) Show the breakdown of the physical addresses:

Physical address is 256KB: so it has 18 bits. The size of each page is 64KB. That is, offset bits is 16, and the other 2 bits are page frame bits (4 frames).

frame	offset
2	16

Class Exercise

Current values in each TLB and page table is as follows.

TLB

	Virtual page	Frame (physical)
0	0x5	2
1	0XA	3

TLB LRU
5
A

MEM LRU
5
A
B
3

Page Table

	Frame (physical)	Validity bit
0		
1		
2		
3	0	1
4		
5	2	1
6		
7		
8		
9		
A	3	1
B	1	1
C		
D		
E		
F		

Q3) Show the complete state of the memory hierarchy after the following virtual address been processed: **0xA4620**

Class Exercise

Current values in each TLB and page table is as follows.

TLB

	Virtual page	Frame (physical)
0	0x5	2
1	0XA	3

TLB LRU
A
5

MEM LRU
A
5
B
3

Page Table

	Frame (physical)	Validity bit
0		
1		
2		
3	0	1
4		
5	2	1
6		
7		
8		
9		
A	3	1
B	1	1
C		
D		
E		
F		

Q3) Show the complete state of the memory hierarchy after the following virtual address been processed: **0xA4620**

1) Page = 0xA: page is in TLB (hit)

2) Update LRUs

Class Exercise

Current values in each TLB and page table is as follows.

TLB

	Virtual page	Frame (physical)
0	0x5	2
1	0XA	3

TLB LRU
A
5

MEM LRU
A
5
B
3

Page Table

	Frame (physical)	Validity bit
0		
1		
2		
3	0	1
4		
5	2	1
6		
7		
8		
9		
A	3	1
B	1	1
C		
D		
E		
F		

Q4) Show the complete state of the memory hierarchy after the following virtual address been processed: **0xBBB08**

Class Exercise

Current values in each TLB and page table is as follows.

TLB

	Virtual page	Frame (physical)
0	**0xB**	**1**
1	0xA	3

TLB LRU
B
A

MEM LRU
B
A
5
3

Page Table

	Frame (physical)	Validity bit
0		
1		
2		
3	0	1
4		
5	2	1
6		
7		
8		
9		
A	3	1
B	**1**	**1**
C		
D		
E		
F		

Q4) Show the complete state of the memory hierarchy after the following virtual address been processed: **0xBBB08**

1) Page = 0xB: page is not in TLB (miss)

2) Refer page table to get frame for B

3) Update TLB and LRUs

Class Exercise

Current values in each TLB and page table is as follows.

TLB

	Virtual page	Frame (physical)
0	0xB	1
1	0XA	3

TLB LRU
B
A

MEM LRU
B
A
5
3

Page Table

	Frame (physical)	Validity bit
0		
1		
2		
3	0	1
4		
5	2	1
6		
7		
8		
9		
A	3	1
B	1	1
C		
D		
E		
F		

Q5) Show the complete state of the memory hierarchy after the following virtual address been processed: **0x9AC10**

Class Exercise

Current values in each TLB and page table is as follows.

TLB

	Virtual page	Frame (physical)
0	0xB	1
1	0X9	0

TLB LRU
9
B

MEM LRU
9
B
A
5

Page Table

	Frame (physical)	Validity bit
0		
1		
2		
3	**--**	**0**
4		
5	2	1
6		
7		
8		
9	**0**	**1**
A	3	1
B	1	1
C		
D		
E		
F		

Q5) Show the complete state of the memory hierarchy after the following virtual address been processed: **0x9AC10**

1) Page = 0x9: page is not in TLB (miss)

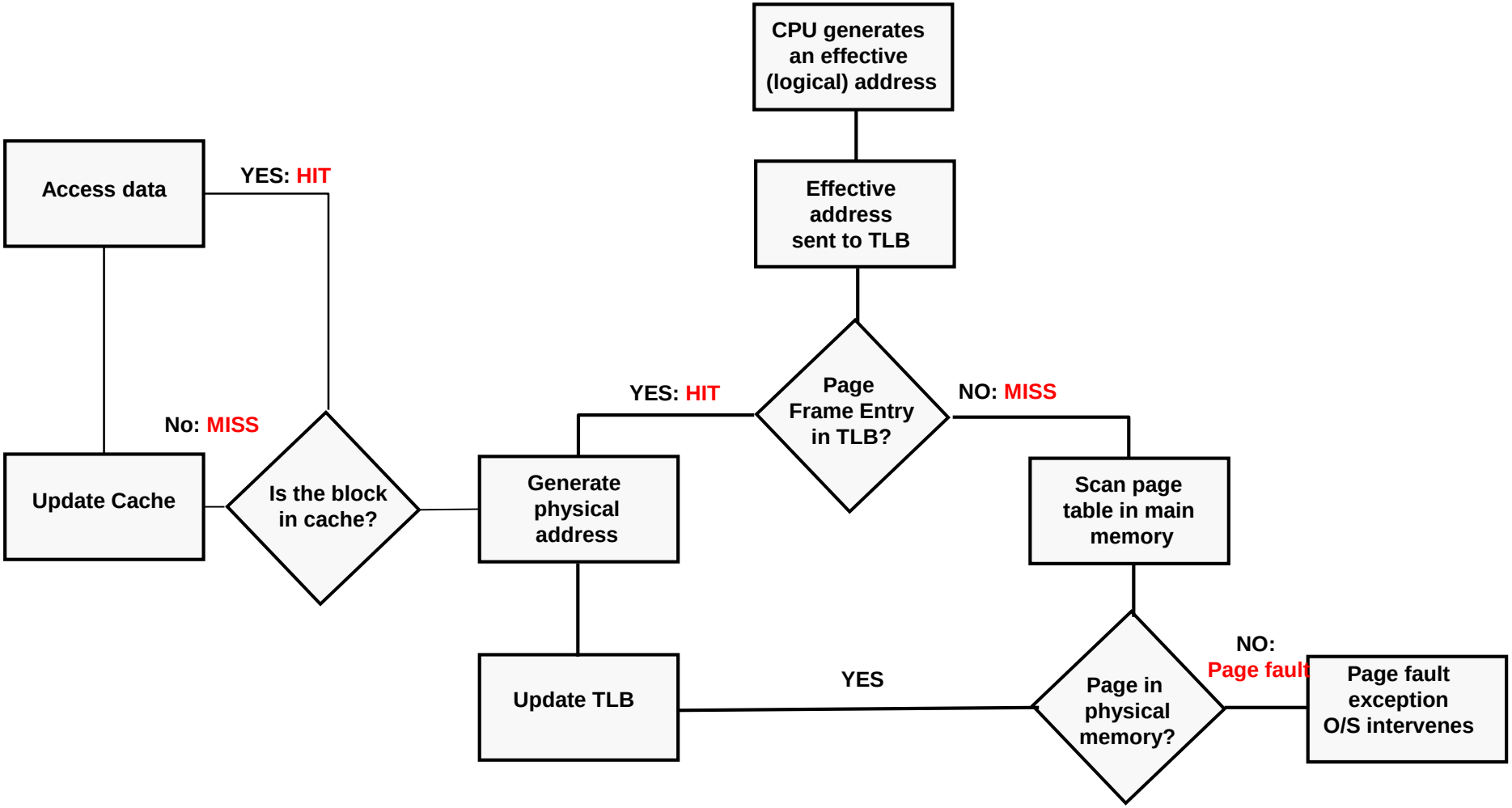
2) Refer page table. Not in page table: page fault

3) Replace page 3 with page 9

3) Update LRUs and TLB

Put All Together

The TLB, Page Table, Main, and Cache Memory



Class Exercise

Consider a system with 256K bytes of physical memory, the size of each page is 64K bytes, and the virtual address space is 1M. The system is byte-accessible. The processor has a 2 entry fully associative TLB.

The processor has a 4Kbyte **direct mapped L1 cache** with 256 bytes/block. There is also a 16Kbyte **2-way set associative L2 cache** with 1024 bytes/block.

Q1) Show the breakdown of the physical addresses as interpreted by L1 cache.

Q2) Show the breakdown of the physical addresses as interpreted by L2 cache.

Class Exercise

Consider a system with 256K bytes of physical memory, the size of each page is 64K bytes, and the virtual address space is 1M. The system is byte-accessible. The processor has a 2 entry fully associative TLB.

Q1) Show the breakdown of the virtual addresses:

Virtual address space is 1MB: so it has 20 bits. The size of each page is 64KB. That is, offset bits is 16, and the other 4 bits are page bits.

page	offset
4	16

Q2) Show the breakdown of the physical addresses:

Physical address is 256KB: so it has 18 bits. The size of each page is 64KB. That is, offset bits is 16, and the other 2 bits are page frame bits.

frame	offset
2	16

Class Exercise

Consider a system with 256K bytes of physical memory, the size of each page is 64K bytes, and the virtual address space is 1M. The system is byte-accessible. The processor has a 2 entry fully associative TLB.

The processor has a 4Kbyte direct mapped L1 cache with 256 bytes/block. There is also a 16Kbyte 2-way set associative L2 cache with 1024 bytes/block.

Q1) Show the breakdown of the physical addresses as interpreted by L1 cache.

L1 cache is 4KB. Each L1 block is 256B → Offset is 8-bit. The number of blocks is 4K/256 = 16 → 4 bits for Row. The number of pages is 256K/4K = 64

Tag	Row	offset
6	4	8

Q2) Show the breakdown of the physical addresses as interpreted by L2 cache.

L2 cache is 16KB. Each L1 block is 1024B → Offset is 10-bit. The number of blocks = 16K/1K = 16. But total 16/2 = 8 sets → Set is 3-bit. The rest of bits is for tag (5 bits)

Tag	Set	offset
5	3	10

Class Exercise – cont'd

Q3) How to find 0xA4620 (virtual address) in L1 cache?

Q4) How to find 0xA4620 (virtual address) in L2 cache?

Page Table

	Frame (physical)	Validity bit
0		
1		
2		
3	0	1
4		
5	2	1
6		
7		
8		
9		
A	3	1
B	1	1
C		
D		
E		
F		

MEM LRU
5
A
B
3

TLB

	Virtual page	Frame
0	5	2
1	A	3

TLB LRU
5
A

Class Exercise – cont'd

Q3) How to find 0xA4620 in L1 cache?

Convert Virtual Address to Physical Address:

A4620 → 34620:

11 0100 0110 0010 0000:

Row 6, tag 110100 = 0x34:

if match with tag, then hit.

Q4) How to find 0xA4620 in L2 cache?

Convert Virtual Address to Physical Address:

A4620 → 34620:

11 010 0 01 10 0010 0000:

Set 1, tag 11010 = 0x1A:

if one of the 2 blocks in set 1 matches with tag, then hit.

Page Table

	Frame (physical)	Validity bit
0		
1		
2		
3	0	1
4		
5	2	1
6		
7		
8		
9		
A	**3**	**1**
B	1	1
C		
D		
E		
F		

MEM LRU
A
5
B
3

TLB

	Virtual page	Frame
0	5	2
1	**A**	**3**

TLB LRU
A
5

Class Exercise – cont'd

Q5) How to find 0x94620 in L1 cache?

Q6) How to find 0x94620 in L2 cache?

Page Table

	Frame (physical)	Validity bit
0		
1		
2		
3	0	1
4		
5	2	1
6		
7		
8		
9		
A	3	1
B	1	1
C		
D		
E		
F		

MEM LRU
5
A
B
3

TLB

	Virtual page	Frame
0	5	2
1	A	3

TLB LRU
5
A

Class Exercise – cont'd

Q5) How to find 0x94620 in L1 cache?

Convert Virtual Address to Physical Address:

94620 → ? : page fault:

Remove page 3 from MEM LRU

Update page 9 with frame 0:

Physical Address: 04620

00 0100 0110 0010 0000:

Row 6, tag 000100= 0x04:

if match with tag then hit.

Q6) How to find 0x94620 in L2 cache?

00 010 0 01 10 0010 0000

Set 1, tag 00010= 0x02:

if one of 2 blocks in the set 1 matches

With tag then hit.

MEM LRU
9
5
A
B

TLB

	Virtual page	Frame
0	5	2
1	9	0

TLB LRU
9
5

Page Table

	Frame (physical)	Validity bit
0		
1		
2		
3	--	0
4		
5	2	1
6		
7		
8		
9	0	1
A	3	1
B	1	1
C		
D		
E		
F		

Modern Systems

Characteristic	Intel Pentium Pro	PowerPC 604
Virtual address	32 bits	52 bits
Physical address	32 bits	32 bits
Page size	4 KB, 4 MB	4 KB, selectable, and 256 MB
TLB organization	A TLB for instructions and a TLB for data Both four-way set associative Pseudo-LRU replacement Instruction TLB: 32 entries Data TLB: 64 entries TLB misses handled in hardware	A TLB for instructions and a TLB for data Both two-way set associative LRU replacement Instruction TLB: 128 entries Data TLB: 128 entries TLB misses handled in hardware

Characteristic	Intel Pentium Pro	PowerPC 604
Cache organization	Split instruction and data caches	Split intruction and data caches
Cache size	8 KB each for instructions/data	16 KB each for instructions/data
Cache associativity	Four-way set associative	Four-way set associative
Replacement	Approximated LRU replacement	LRU replacement
Block size	32 bytes	32 bytes
Write policy	Write-back	Write-back or write-through

Wrap-up

- Virtual memory provides the method to map lower memory in the hierarchy to upper memory in the hierarchy
- Page table maps virtual memory and physical memory addresses
- TLB is a cache for page table. Usually use LRU for the replacement policy
- Virtual memory, physical memory and caches are put together to speed up the process