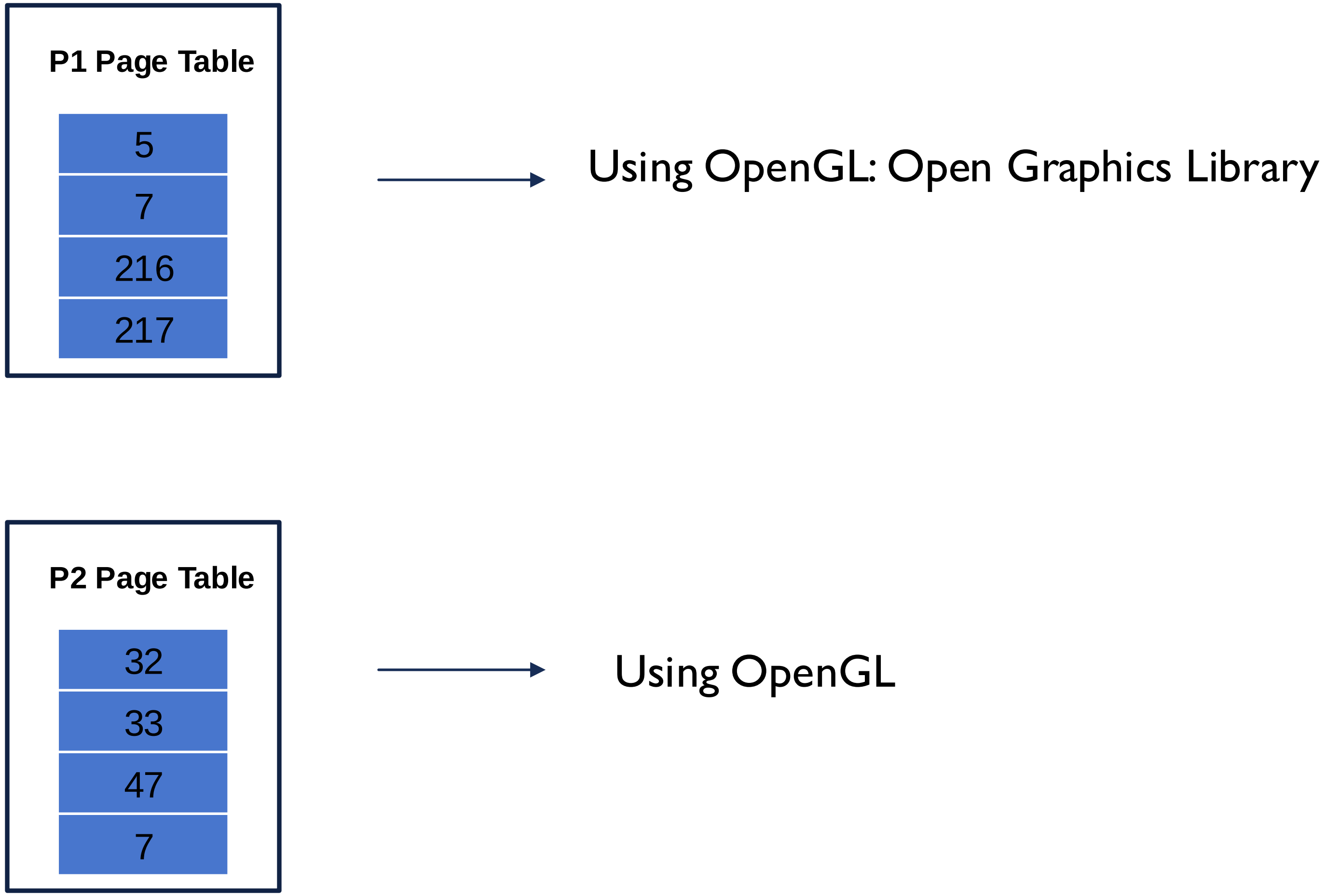
OPERATING SYSTEMS INTERNALS

CSCI 509

PAGE SHARING

- A lot of code is shared by many processed in a system.
- Example: Libraries, routines ..



P1 Page Table

5
7
216
217

Using OpenGL: Open Graphics Library

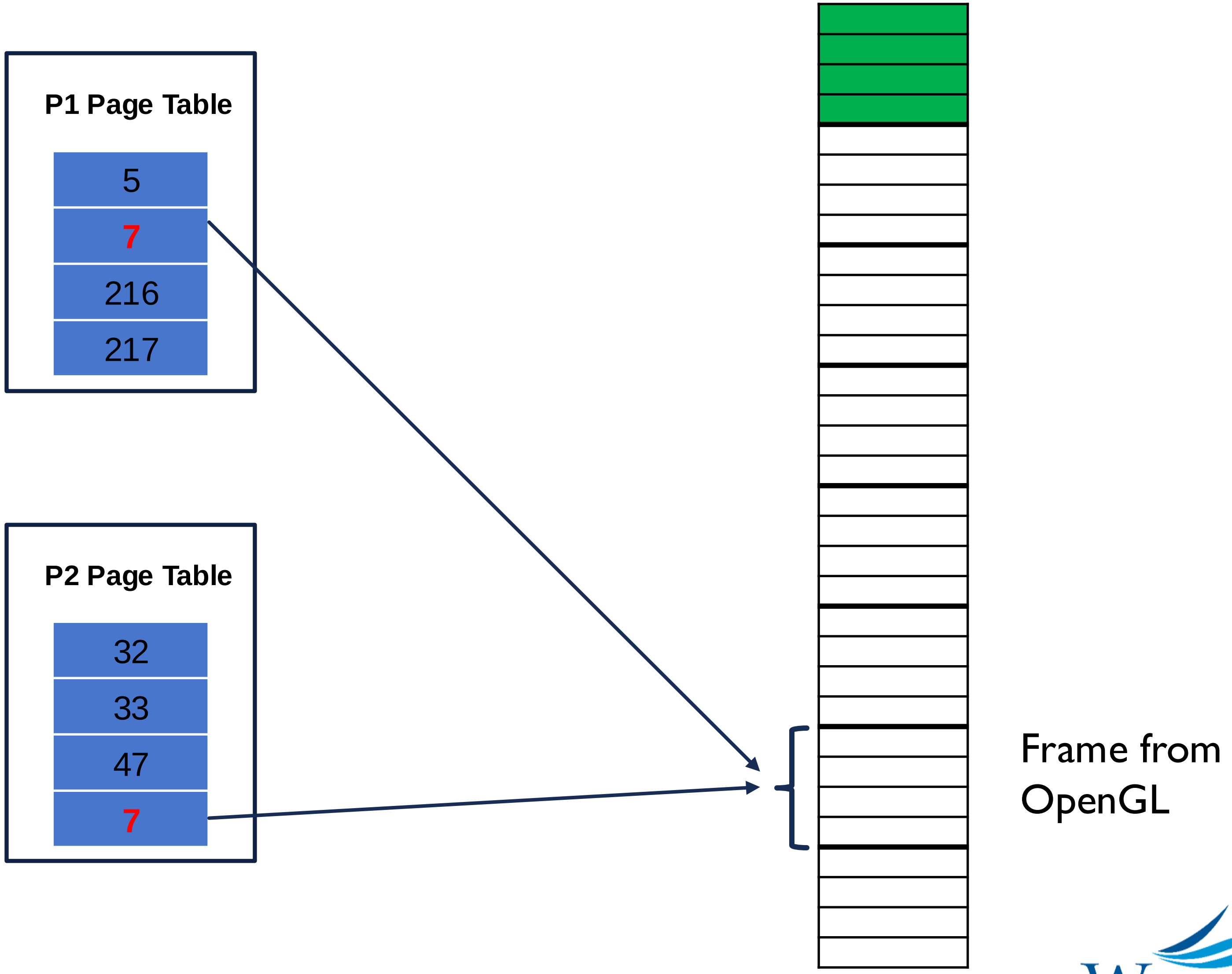
P2 Page Table

32
33
47
7

Using OpenGL

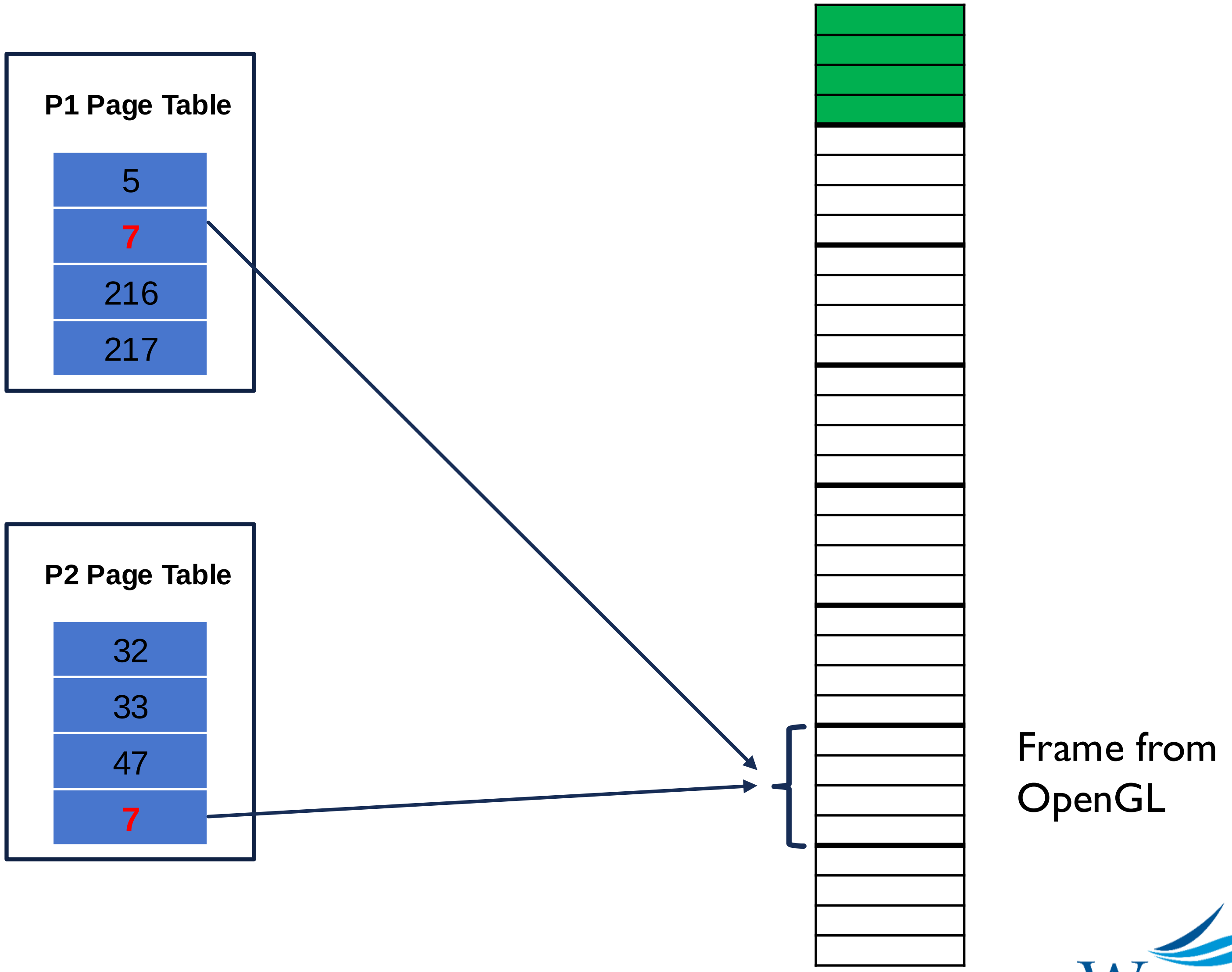
PAGE SHARING

- A lot of code is shared by many processed in a system.
- Example: Libraries, routines ..



PAGE SHARING

- A lot of code is shared by many processed in a system.
- Example: Libraries, routines ..
- Read-Only: Enforced by OS.



P1 Page Table

5
7
216
217

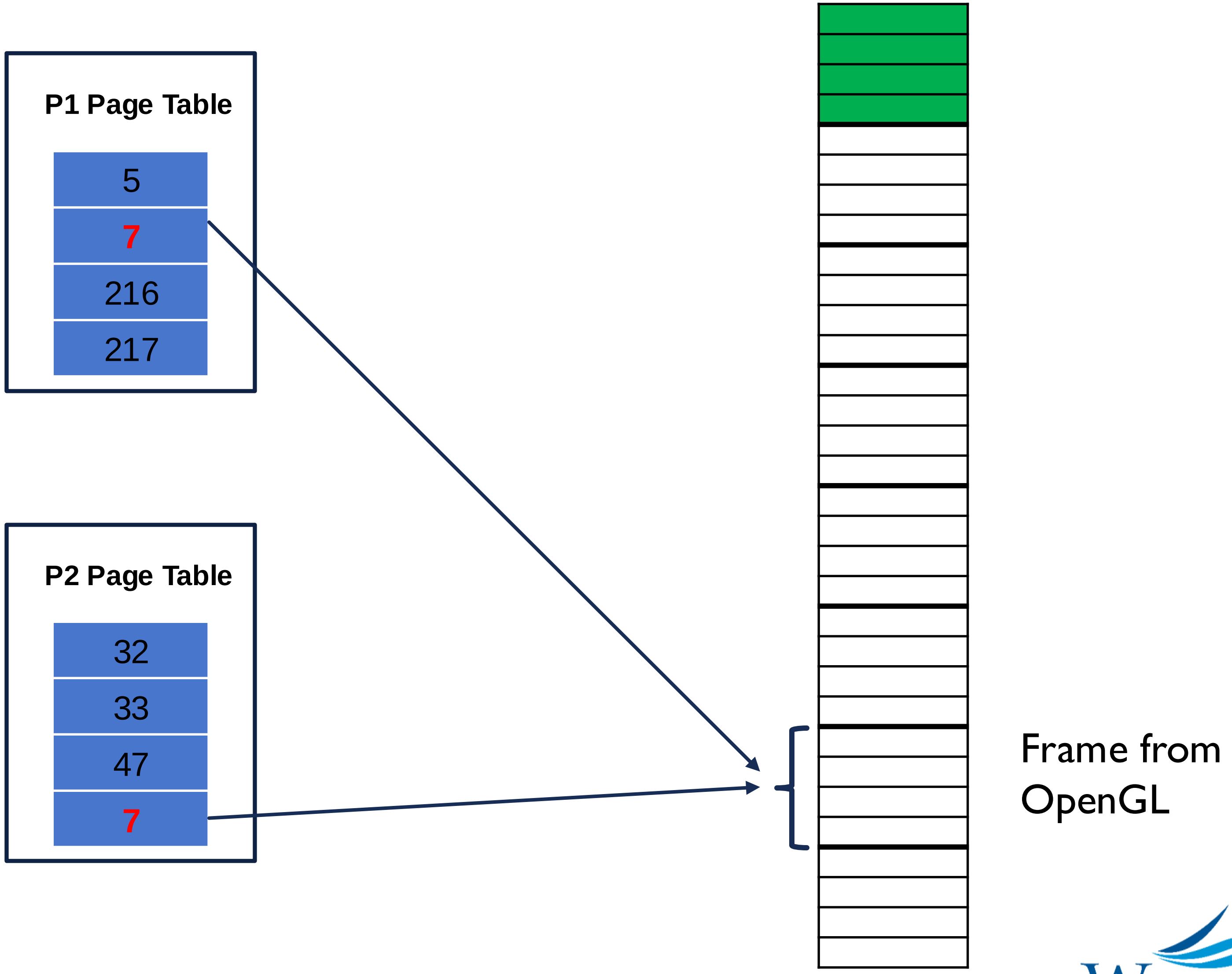
P2 Page Table

32
33
47
7

Frame from OpenGL

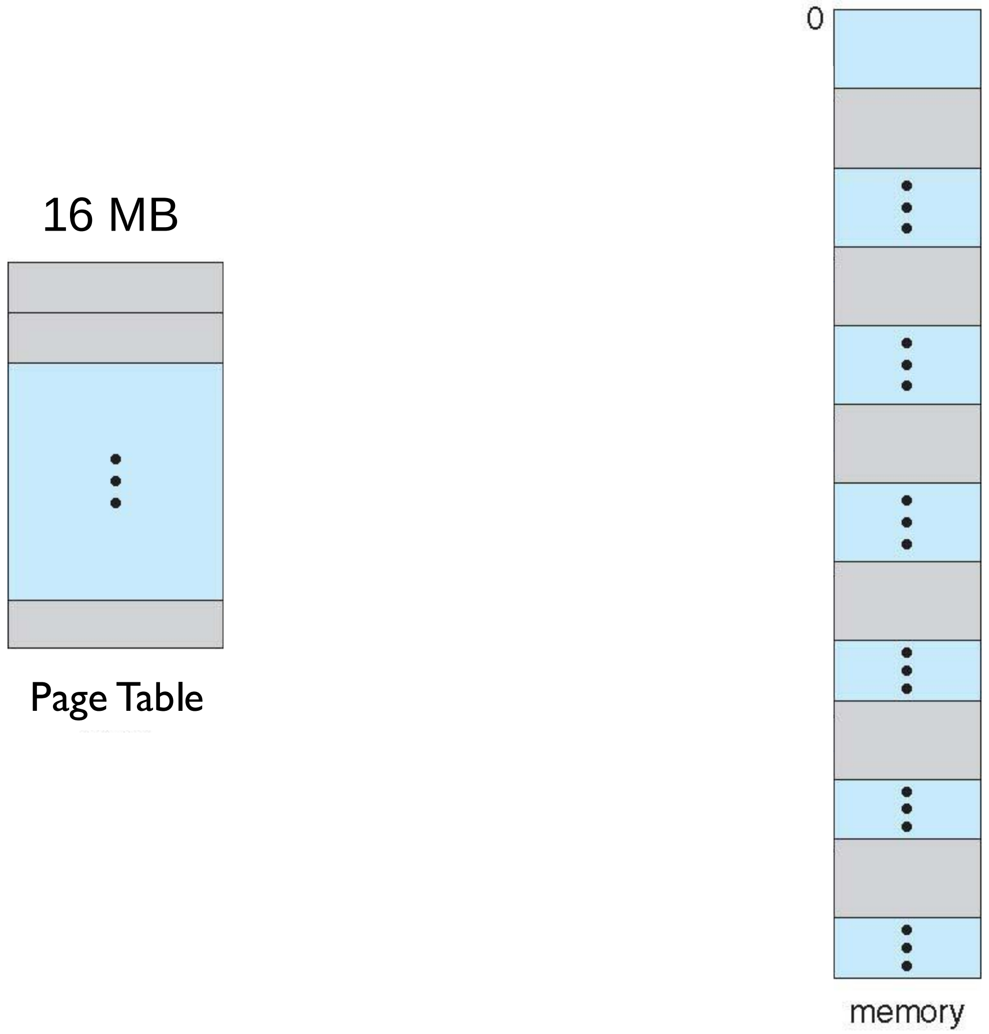
PAGE SHARING

- A lot of code is shared by many processed in a system.
- Example: Libraries, routines ..
- Read-Only: Enforced by OS.
- This is different that shared memory segments and IPC which support read/write on shared memory.



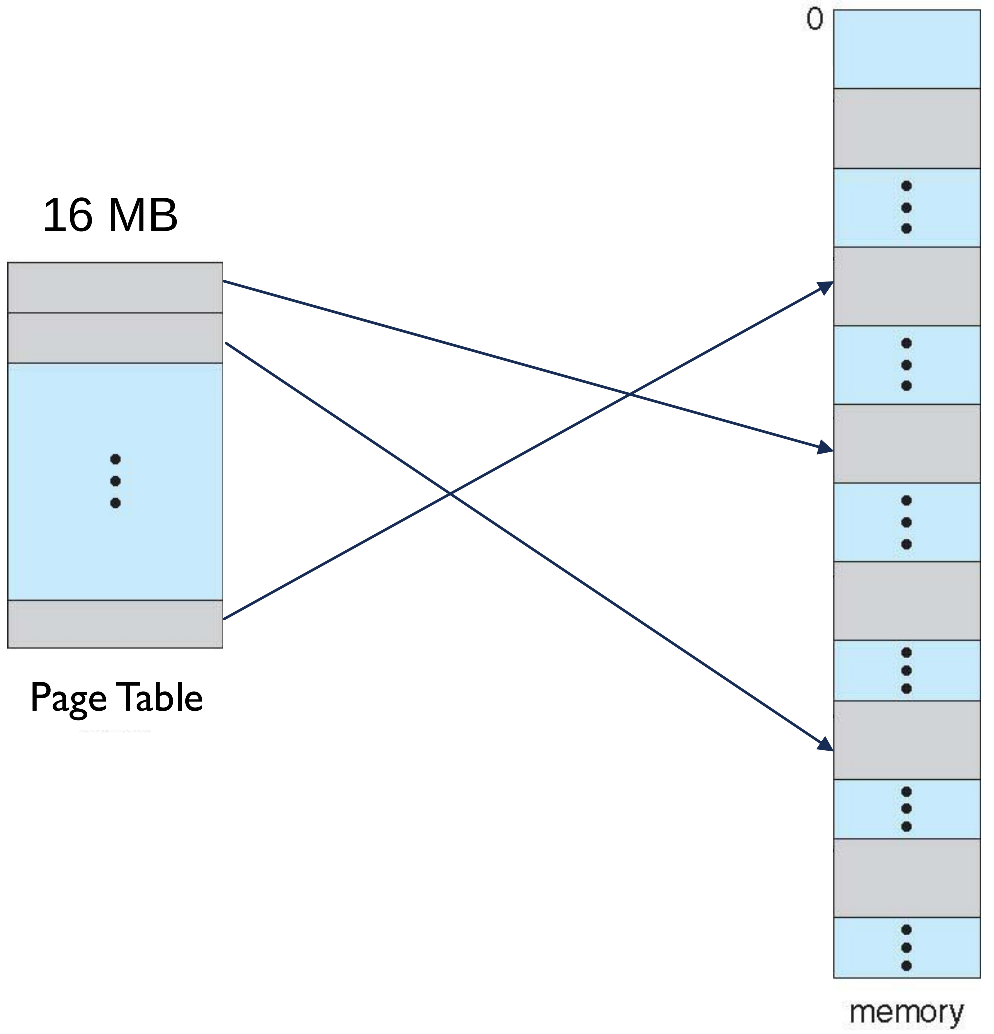
MULTI-LEVEL PAGING

- Assume 32-bit machine.
- If page size is 1 KB, how large would the page table be?
- Size of table: 2^{22}= 4MB for 1 byte per entry.
- For 4 bytes for every entry, that's 16 MB.



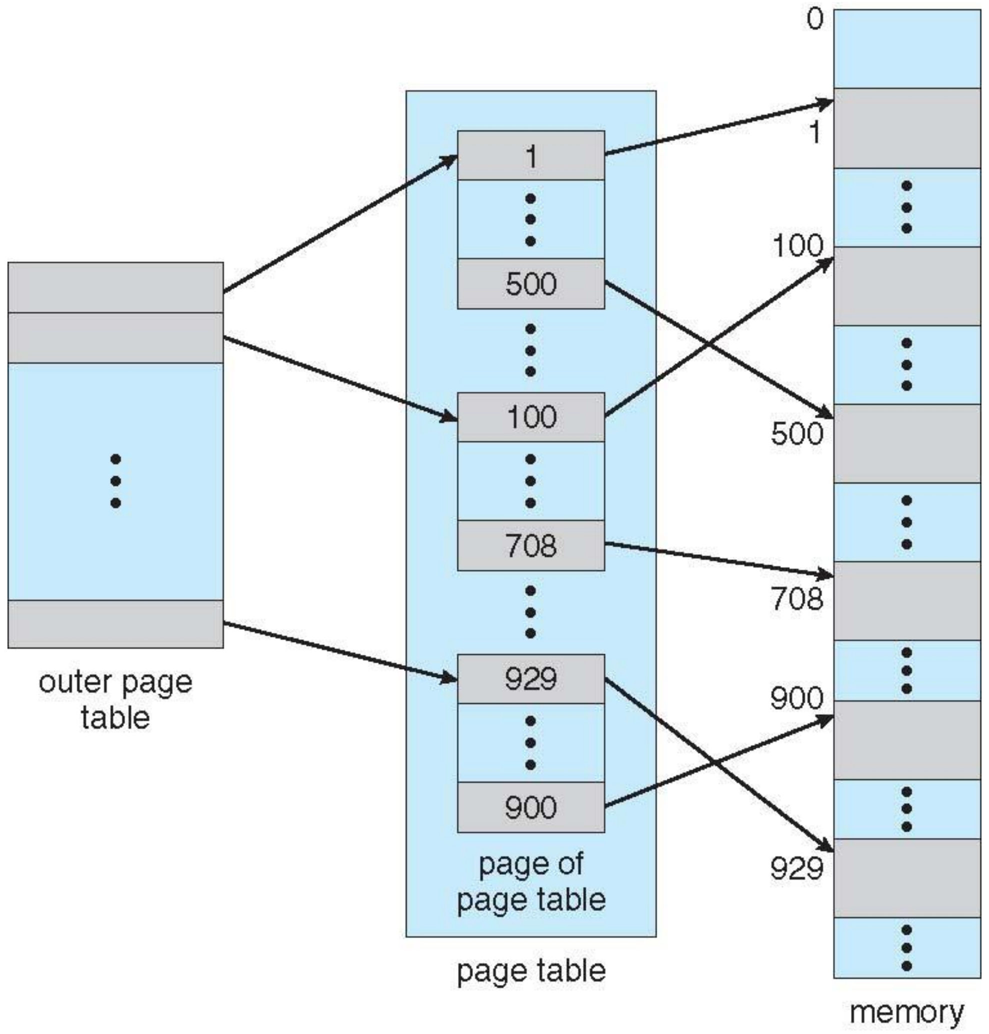
MULTI-LEVEL PAGING

- Assume 32-bit machine.
- If page size is 1 KB, how large would the page table be?
- Size of table: 2^{22}= 4MB for 1 byte per entry.
- For 4 bytes for every entry, that's 16 MB.
- TLB Speeds things up but the page table still needs to be accessed frequently.



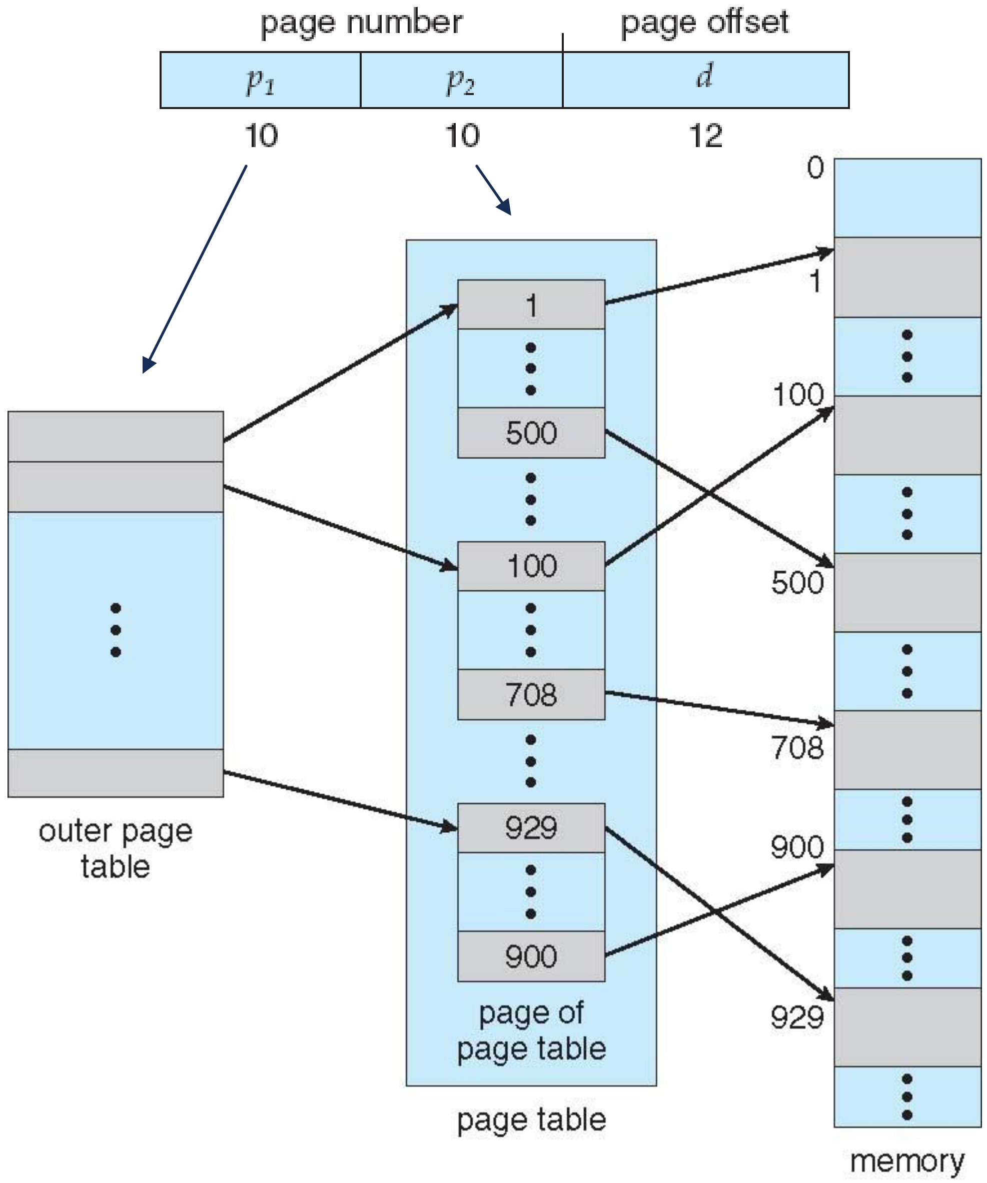
MULTI-LEVEL PAGING

- Solution: Paging the page table.



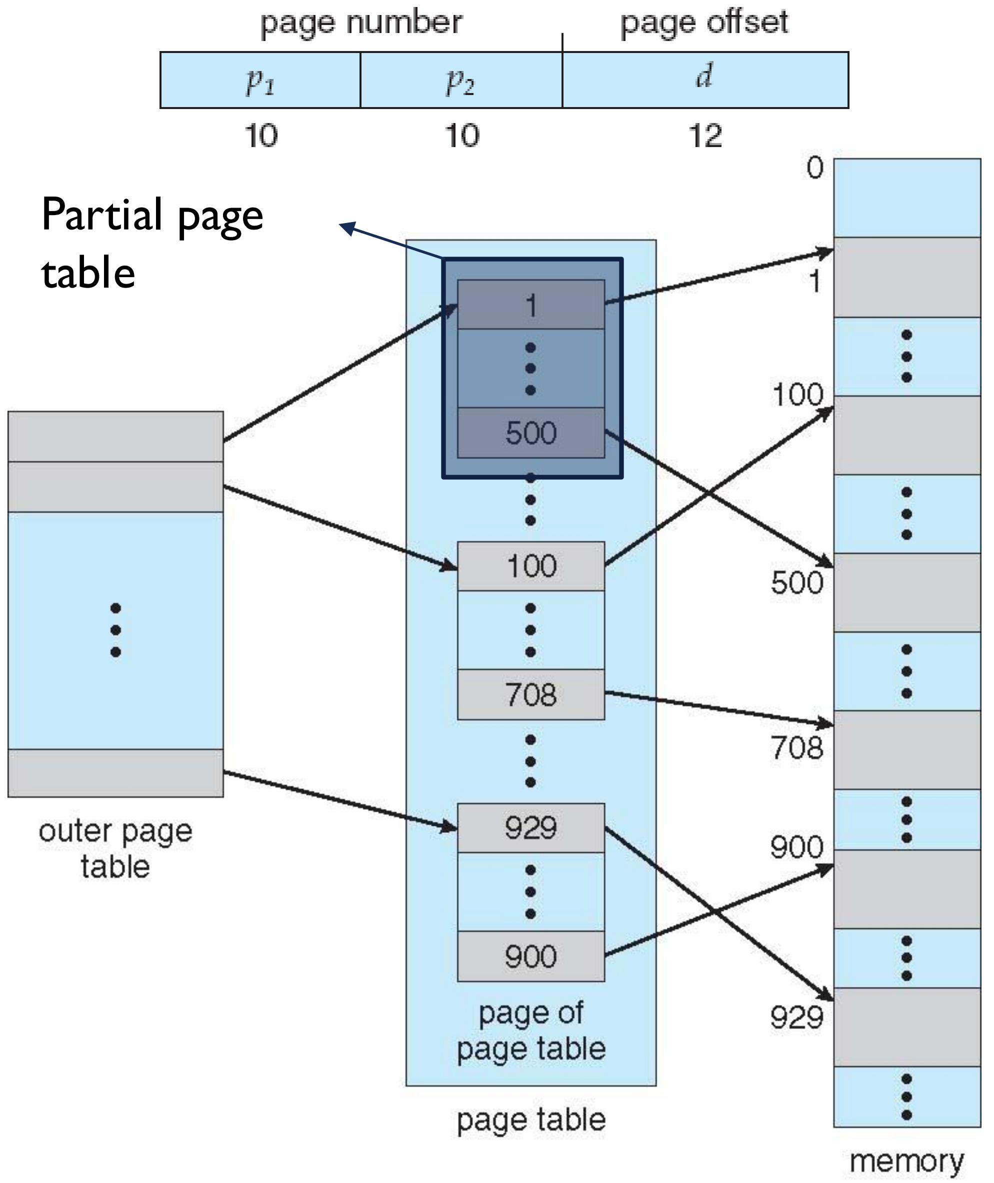
MULTI-LEVEL PAGING

- Solution: Paging the page table.
- The virtual address is now split into outer page, inner page and page offset.



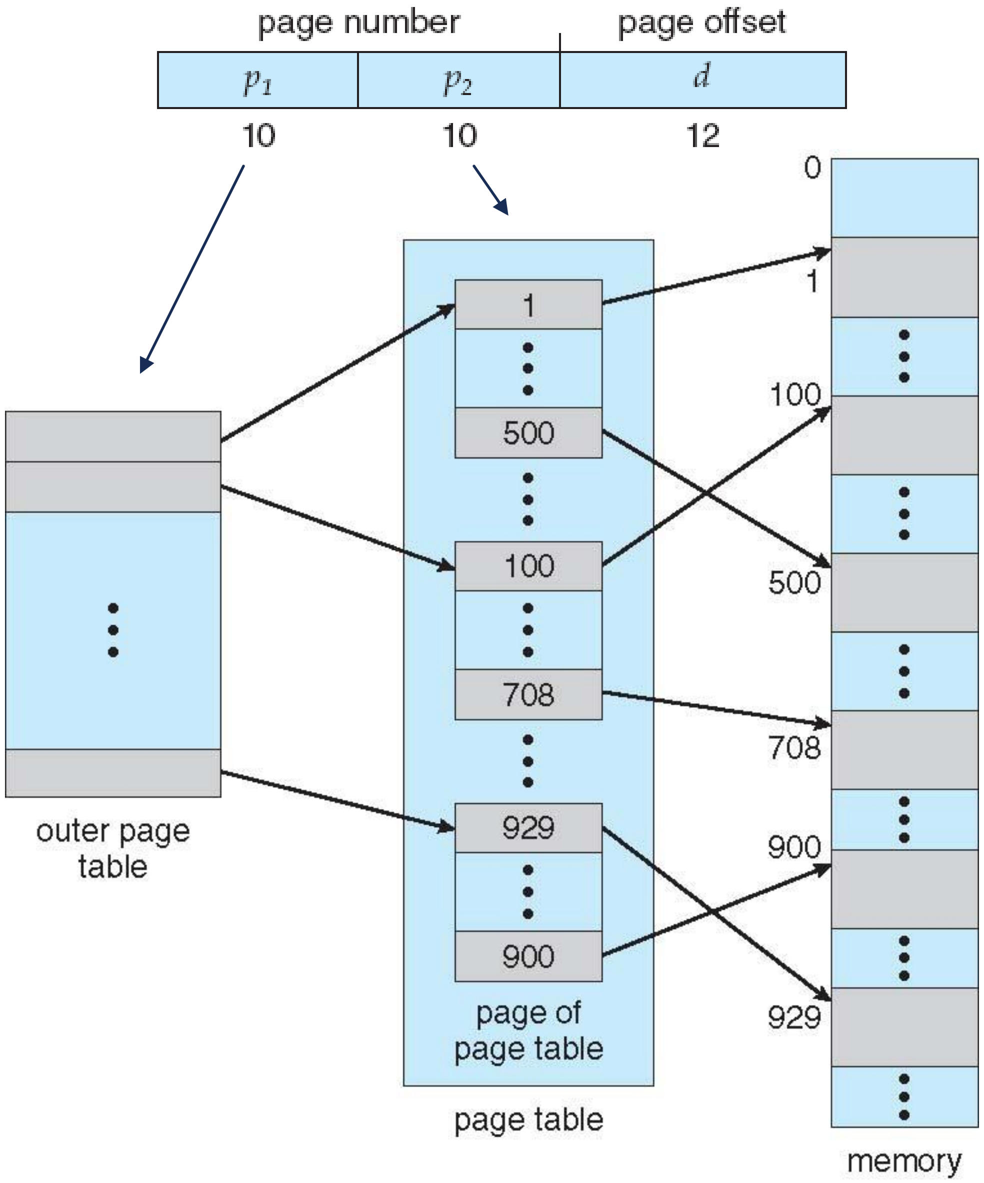
MULTI-LEVEL PAGING

- Solution: Paging the page table.
- The virtual address is now split into outer page, inner page and page offset.



MULTI-LEVEL PAGING

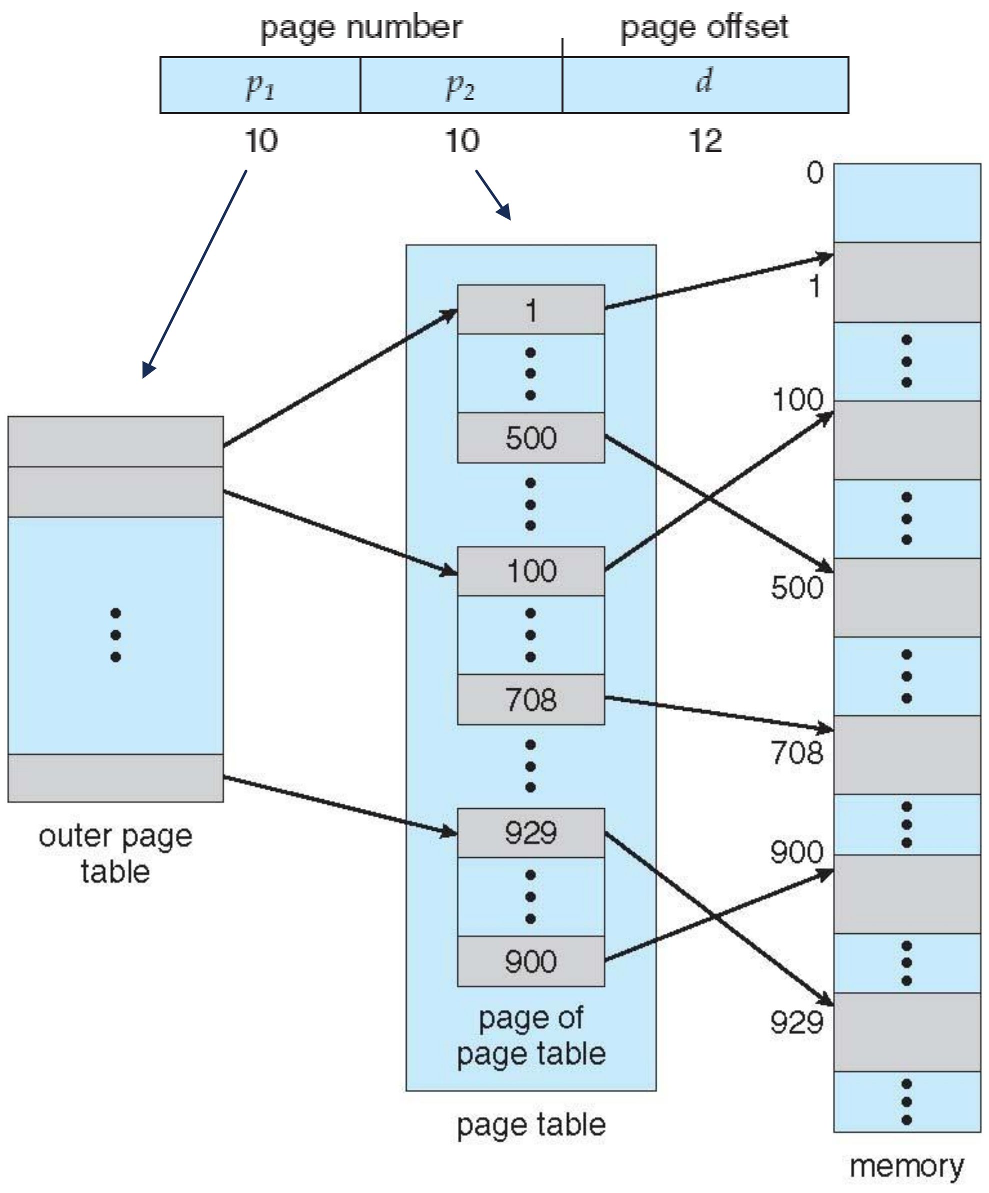
00100011100011011111101101111010



MULTI-LEVEL PAGING

P1 P2 D

00100011100011011111101101111010

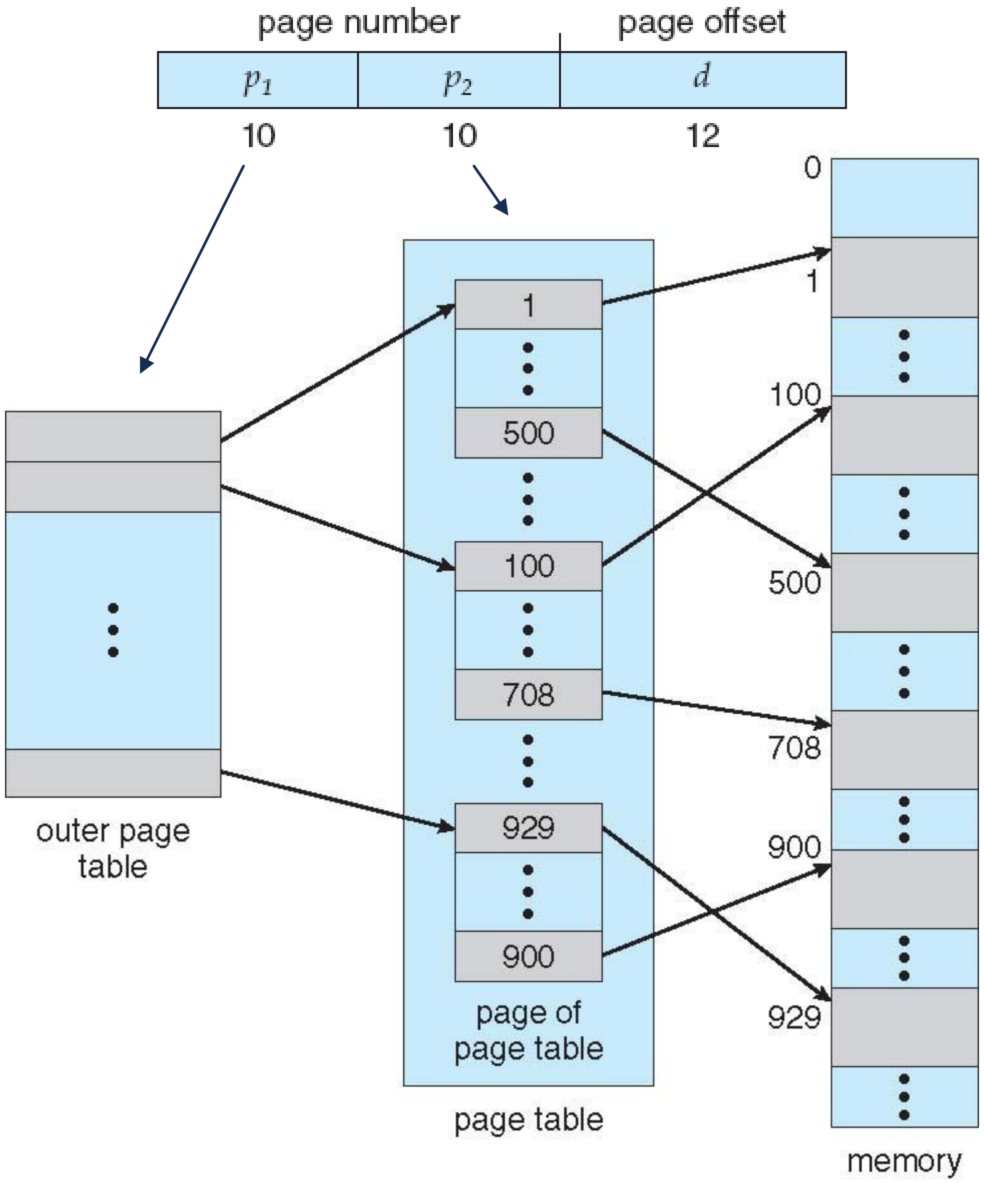


MULTI-LEVEL PAGING

P1 P2 D

0010001110 0011011111 011011111010

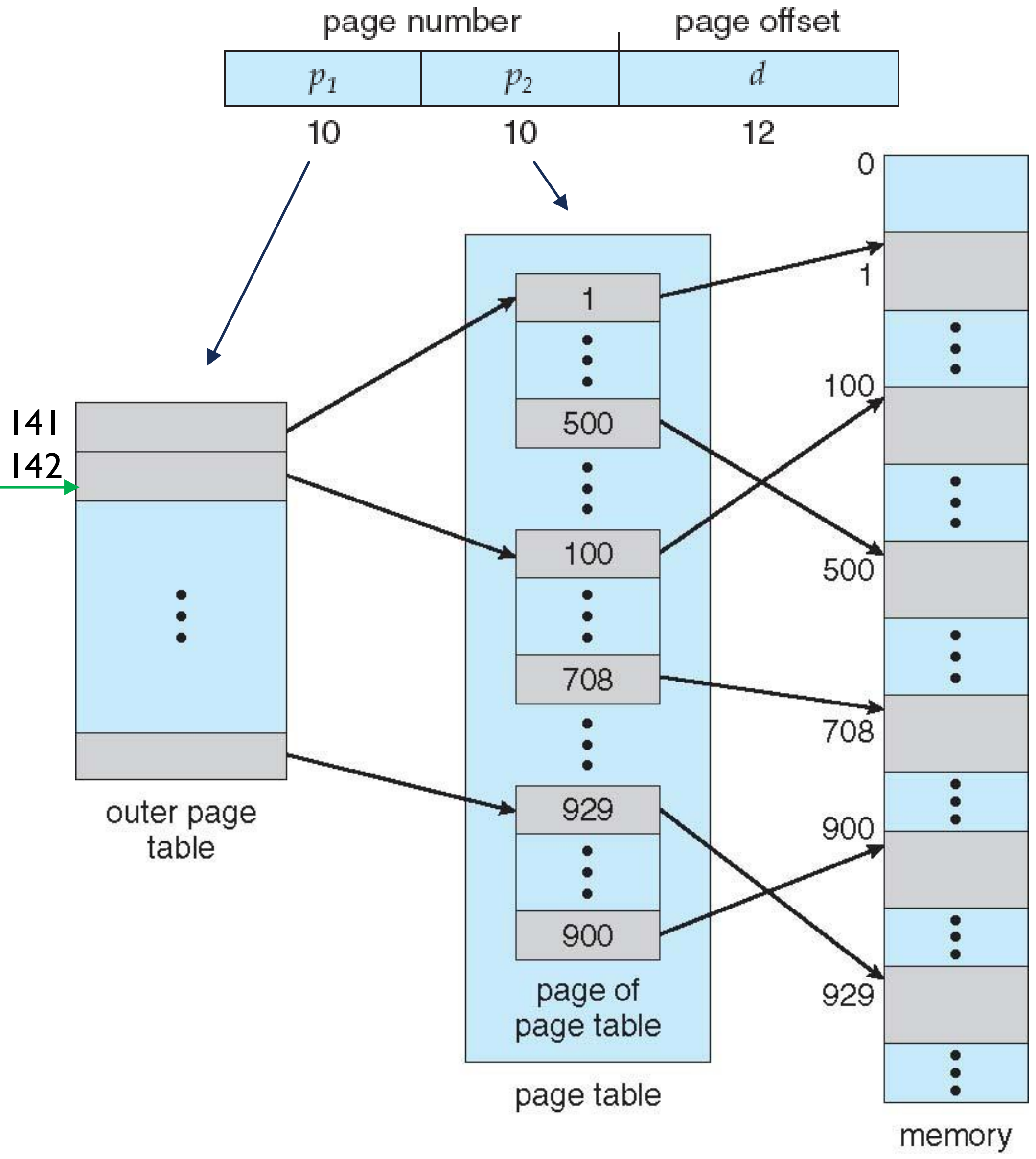
142 447



MULTI-LEVEL PAGING

P1	P2	D
0010001110	0011011111	01101111010
142	447	

Use outer page table to find address of the 142nd inner page table



MULTI-LEVEL PAGING

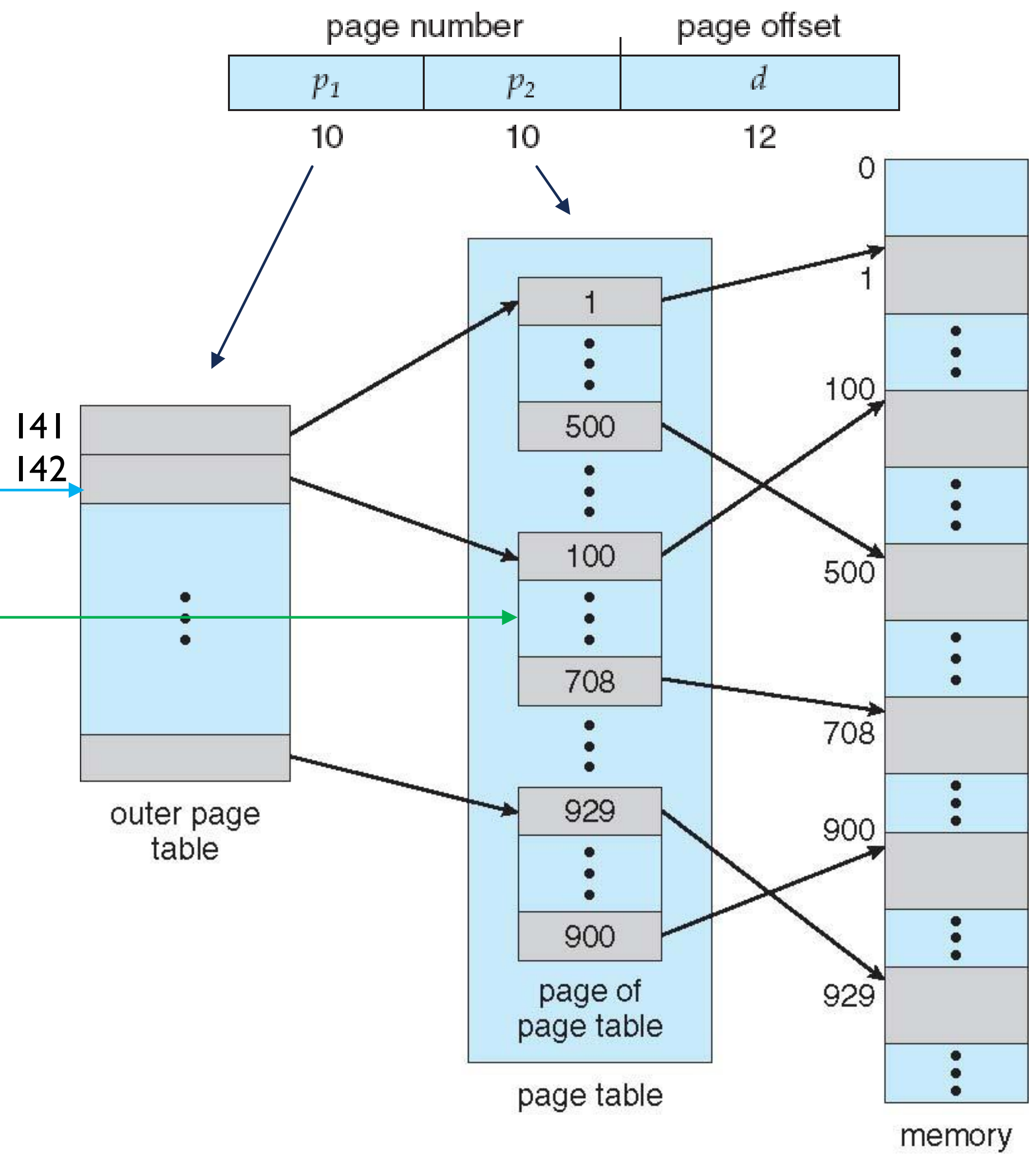
P1 P2 D

00100011100011011111101101111010

142 447

Use inner page table to find address of page 447

Use outer page table to find address of the 142nd inner page table



MULTI-LEVEL PAGING

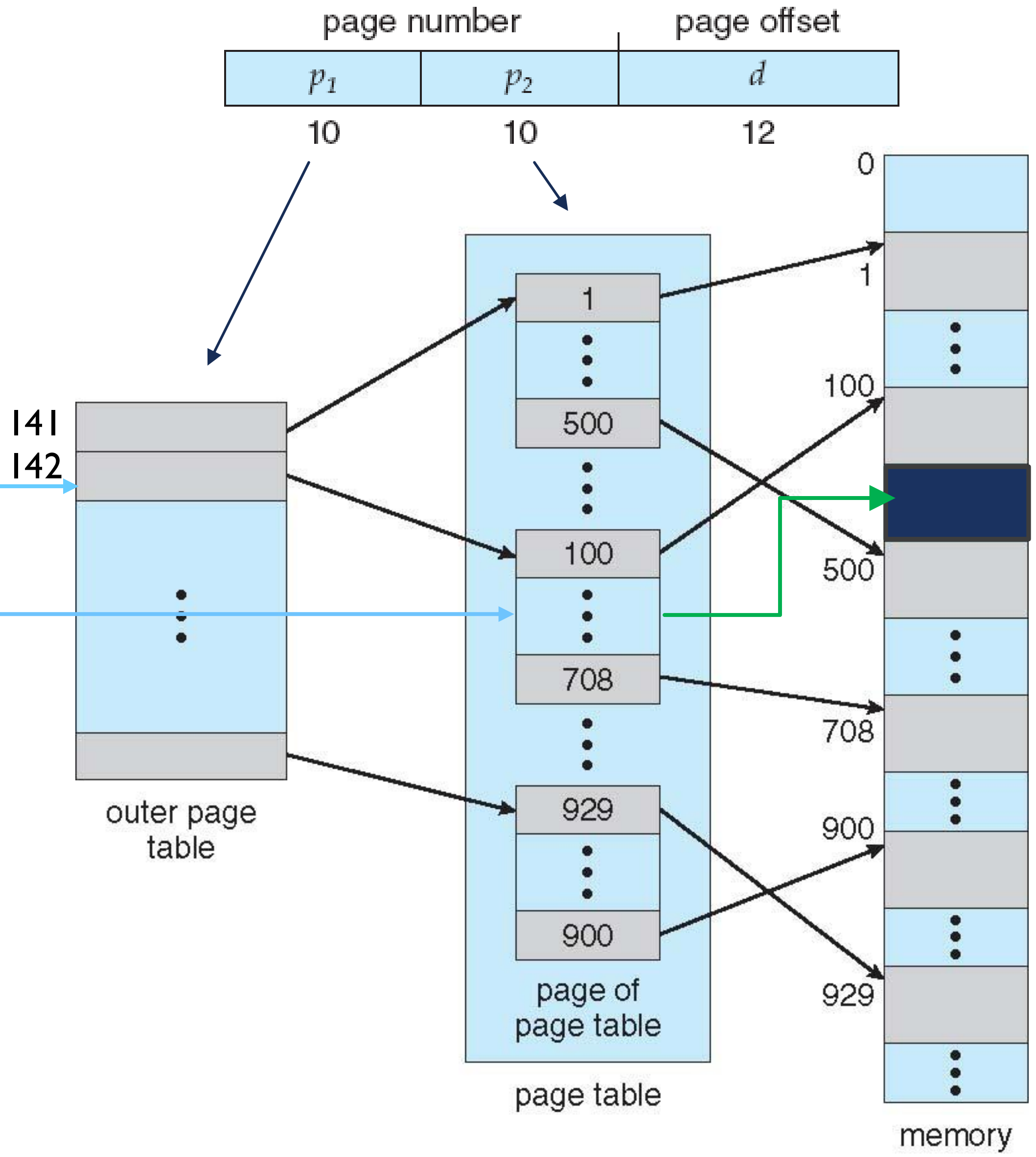
P1 P2 D

001000111000110111111 01101111010

142 447

Use inner page table to find address of page 447

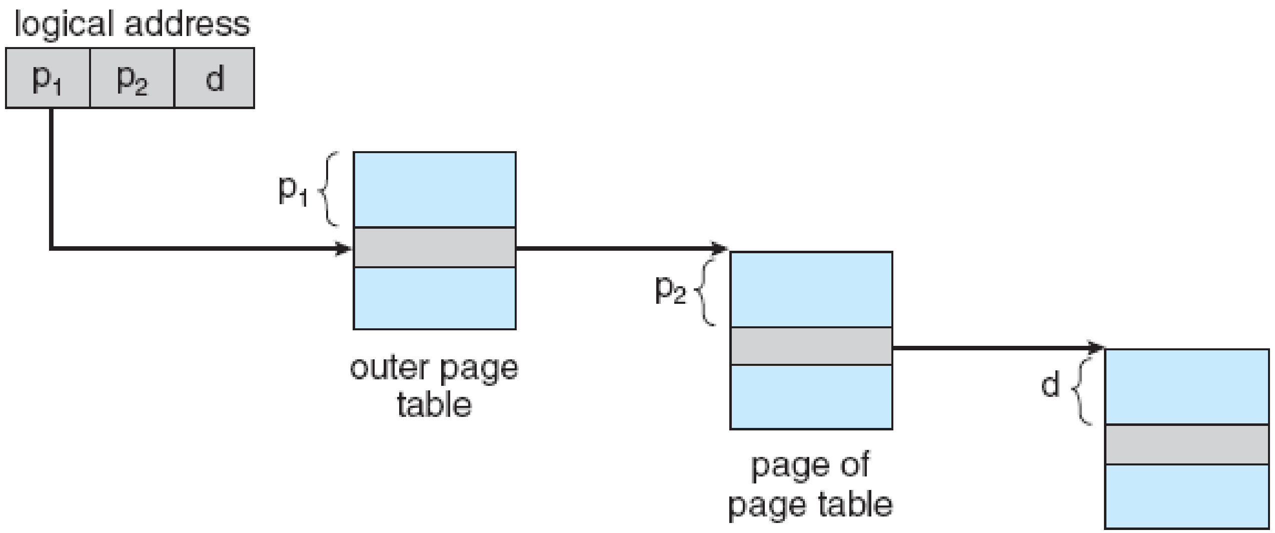
Use outer page table to find address of the 142nd inner page table



Retrieve the page

MULTI-LEVEL PAGING

2-level paging



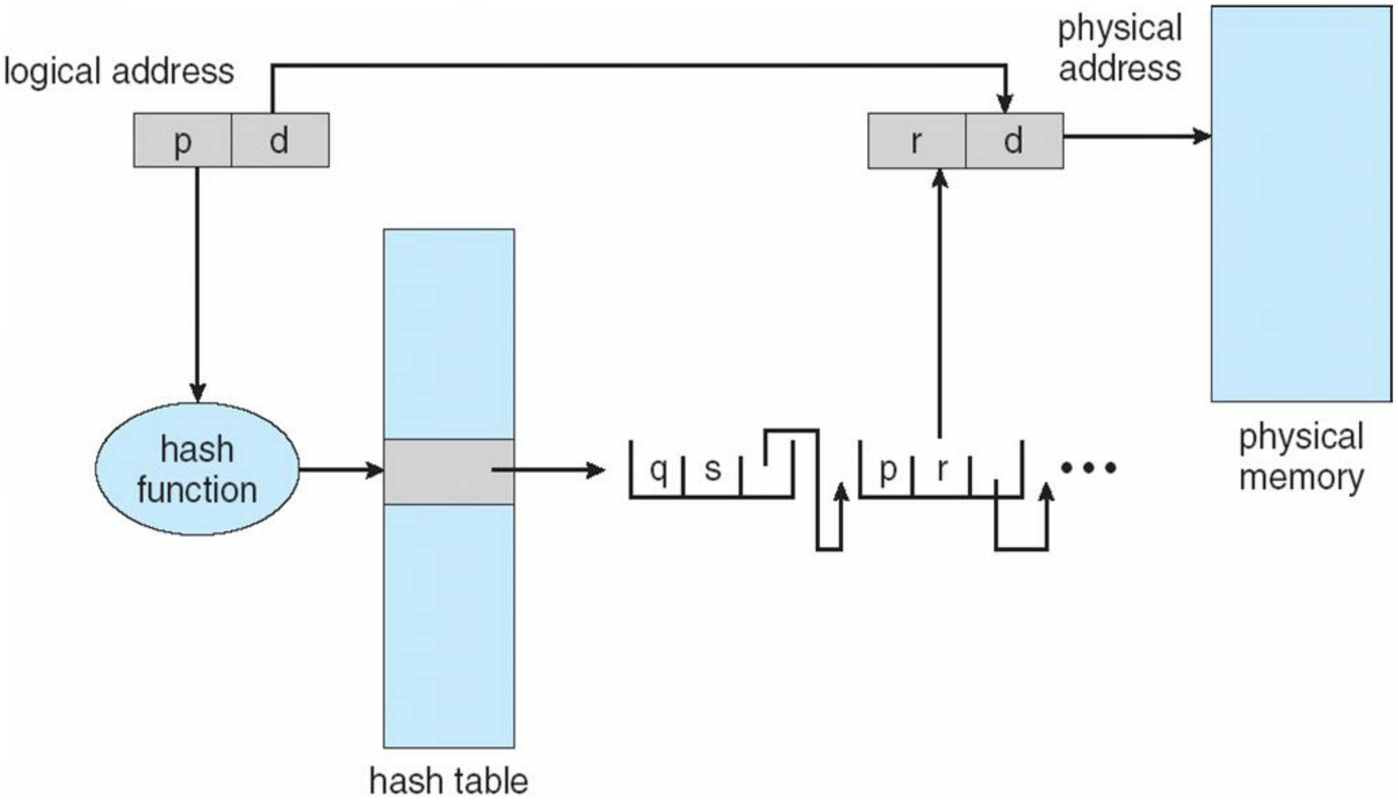
We can have 3 or even more levels of paging …

3-level paging

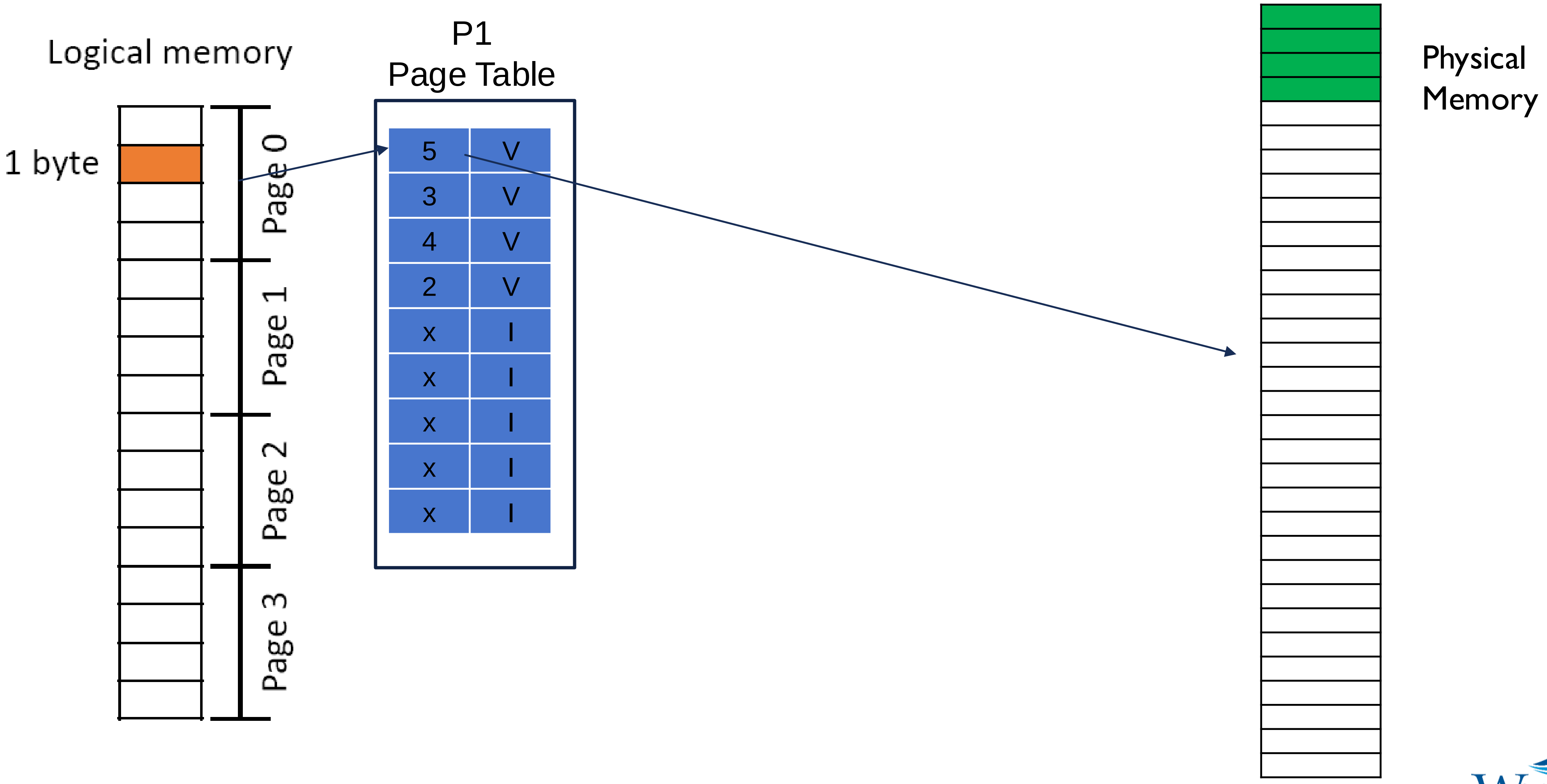
2nd outer page	outer page	inner page	offset
p_1	p_2	p_3	d
32	10	10	12

HASHED PAGED TABLE

- The virtual page number is hashed into a page table
  - This page table contains a chain of elements hashing to the same location
- Each element contains (1) the virtual page number (2) the value of the mapped page frame (3) a pointer to the next element

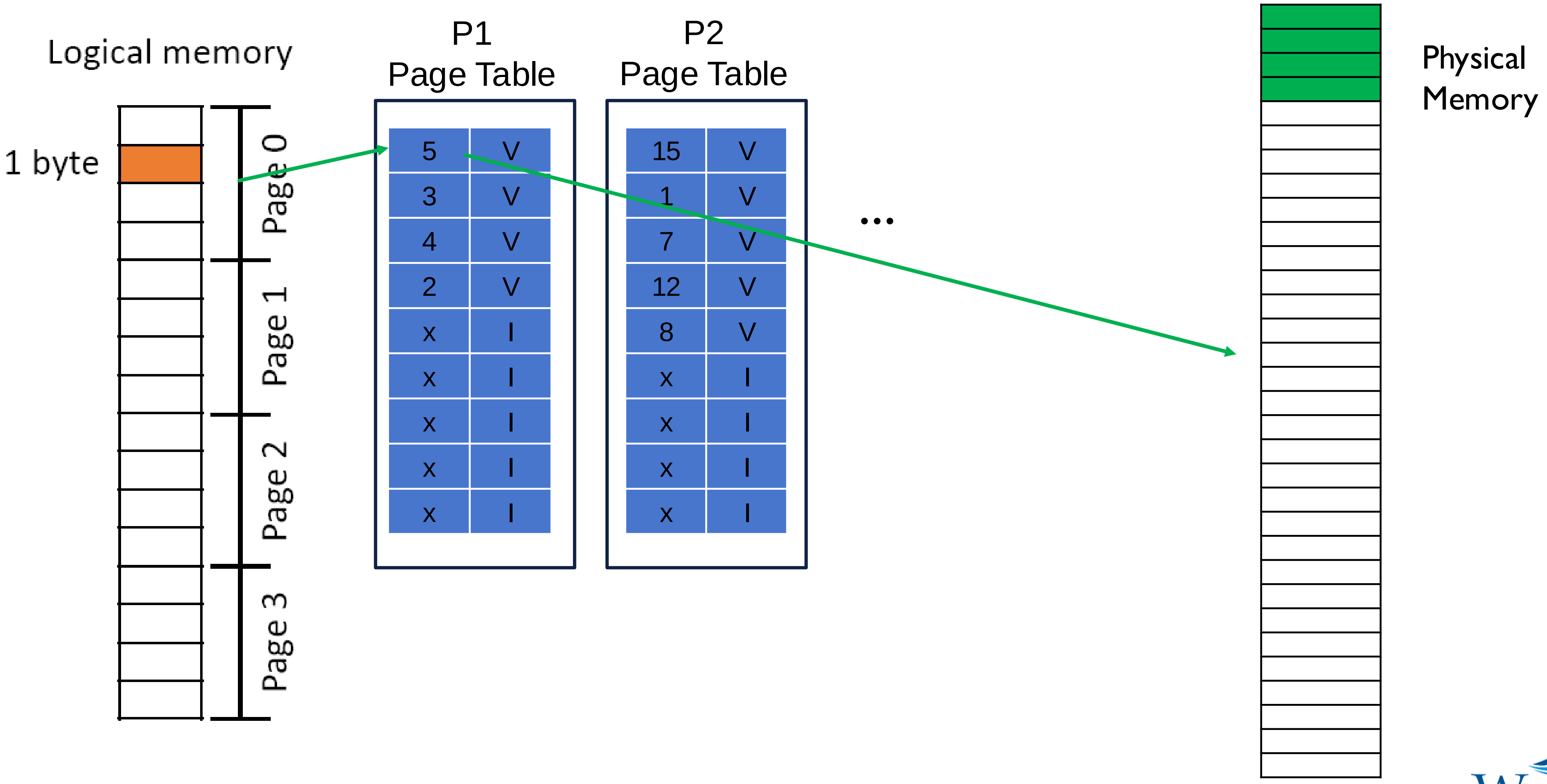


SINGLE PAGE TABLE



5	V
3	V
4	V
2	V
x	I
x	I
x	I
x	I
x	I

SINGLE PAGE TABLE



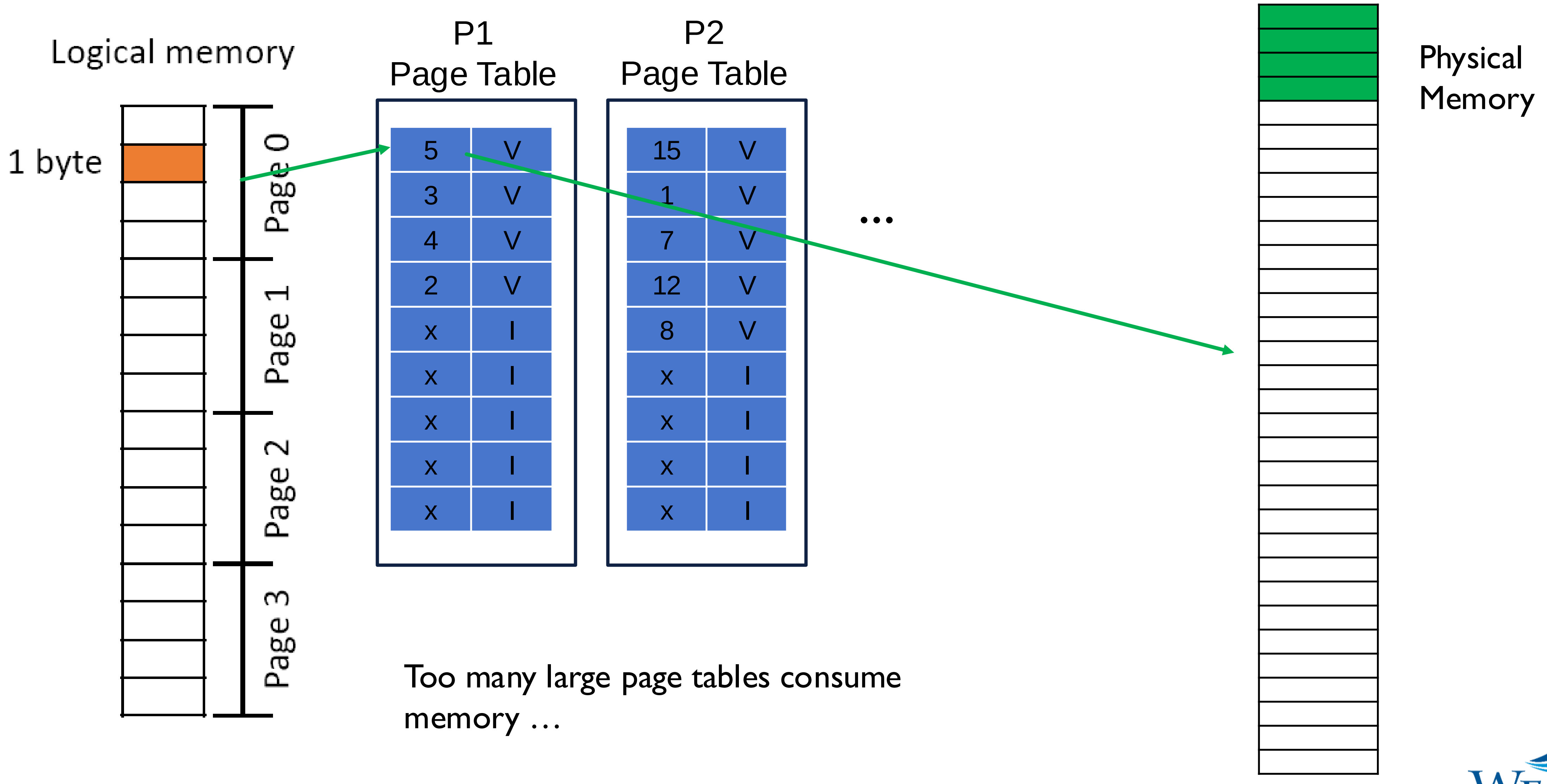
P1
Page Table

5	V
3	V
4	V
2	V
x	I
x	I
x	I
x	I
x	I

P2
Page Table

15	V
1	V
7	V
12	V
8	V
x	I
x	I
x	I
x	I

SINGLE PAGE TABLE



P1
Page Table

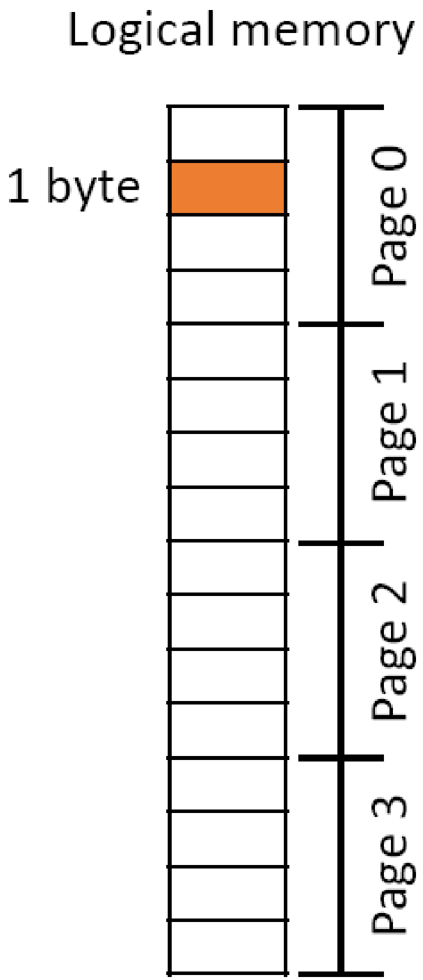
5	V
3	V
4	V
2	V
x	I
x	I
x	I
x	I
x	I

P2
Page Table

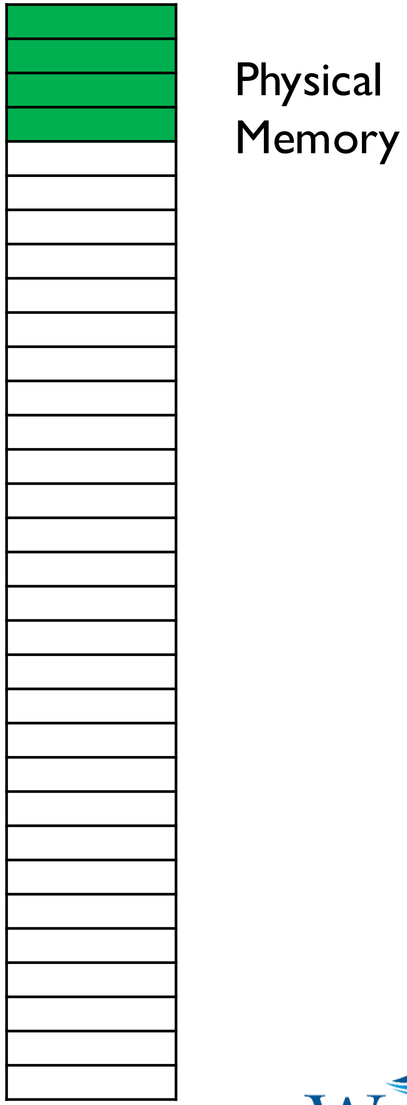
15	V
1	V
7	V
12	V
8	V
x	I
x	I
x	I
x	I

Too many large page tables consume memory …

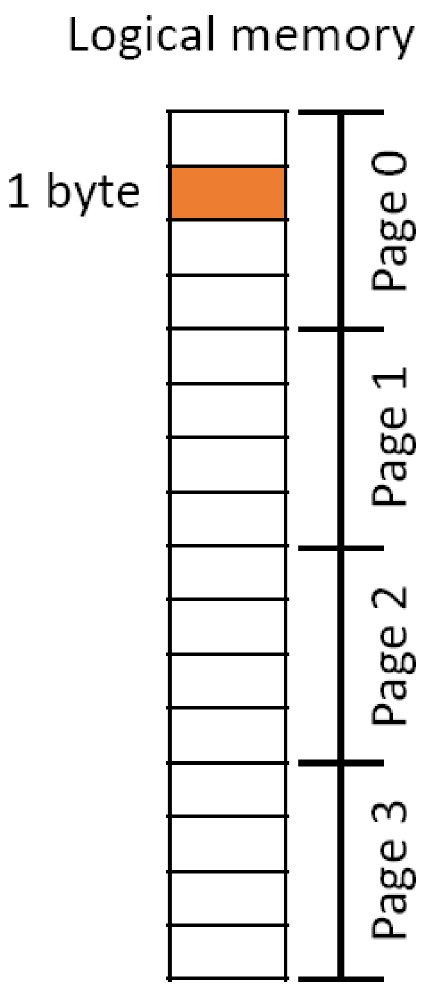
SINGLE PAGE TABLE: INVERTED PAGE TABLE



Instead of a page table, a 'frame' table with all the valid frames in physical memory with process ID and page Number assigned to.



SINGLE PAGE TABLE: INVERTED PAGE TABLE

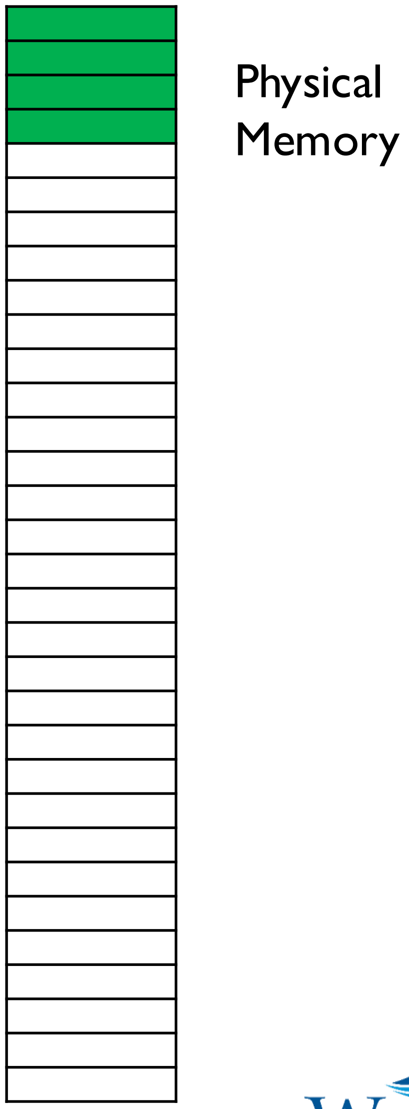


Instead of a page table, a 'frame' table with all the valid frames in physical memory with process ID and page Number assigned to.

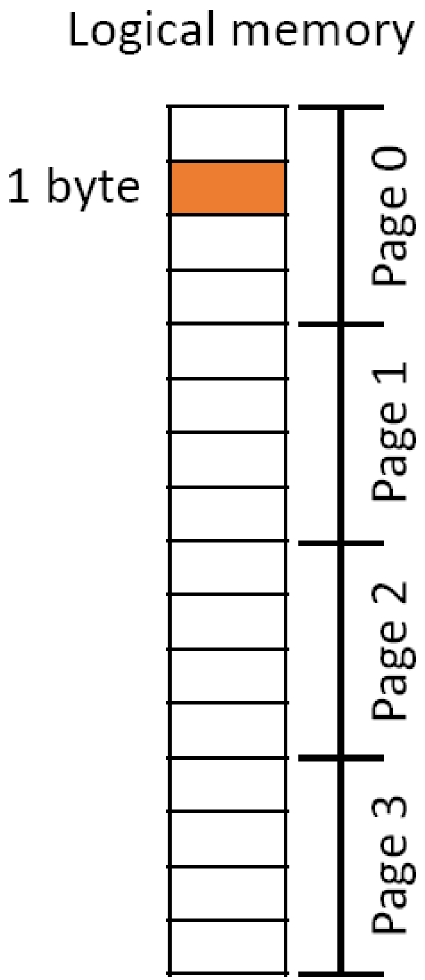
Inverted Page Table

Indexed
By Frame!

	Page No.	Process ID
Frame 0		
Frame 1		
Frame 2		
Frame 3		
Frame 4		
Frame 5		
⋮	⋮	



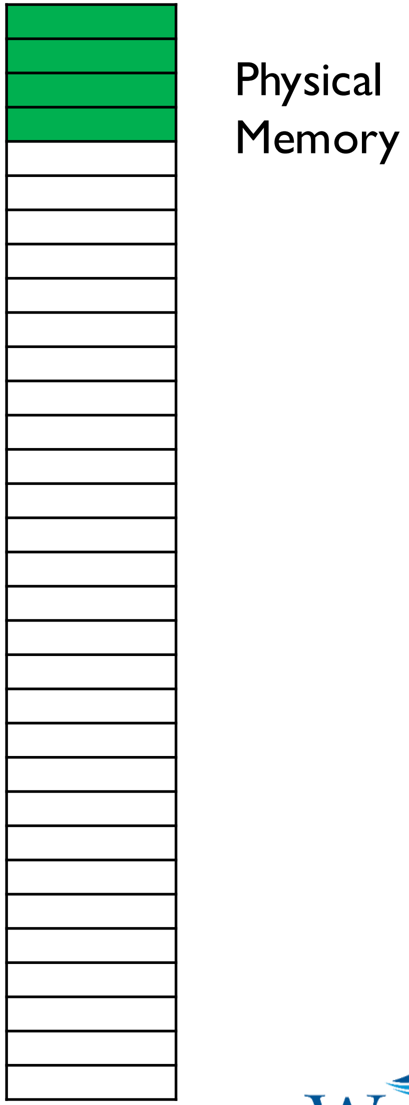
SINGLE PAGE TABLE: INVERTED PAGE TABLE



Instead of a page table, a 'frame' table with all the valid frames in physical memory with process ID and page Number assigned to.

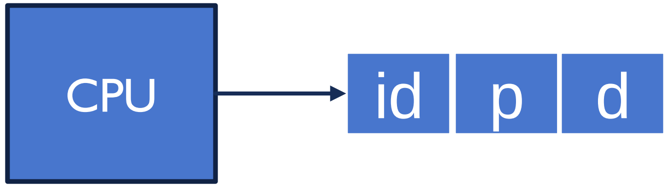
Inverted Page Table

	Page No.	Process ID
Frame 0	Page 1	Process 3
Frame 1	Page 14	Process 1
Frame 2	Page 511	Process 3
Frame 3	Page 13	Process 2
Frame 4	None	None
Frame 5	Page 16	Process 1
⋮	⋮	



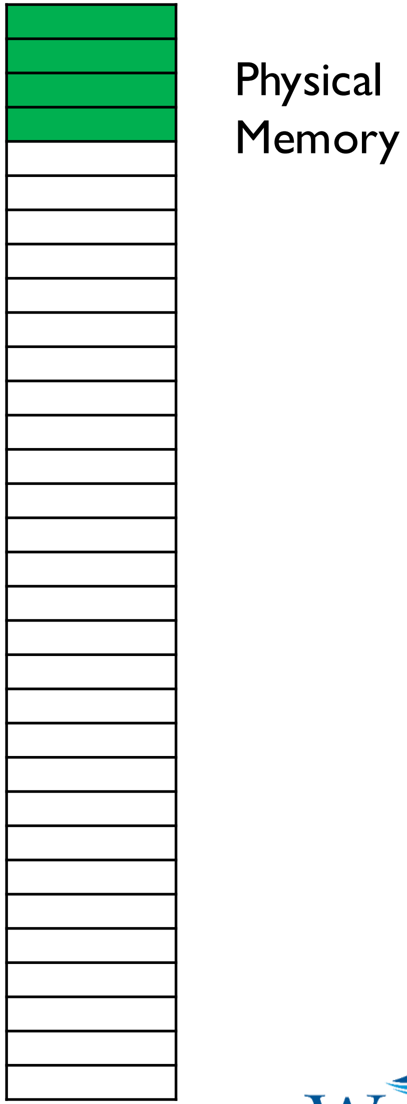
INVERTED PAGE TABLE

Now the address should include the process id, since the table is shared by all processes.



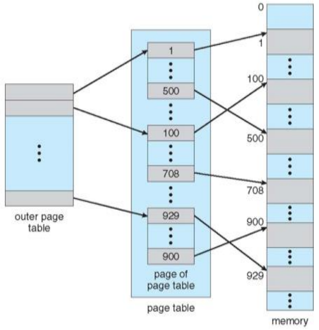
Inverted Page Table

	Page No.	Process ID
Frame 0	Page 1	Process 3
Frame 1	Page 14	Process 1
Frame 2	Page 511	Process 3
Frame 3	Page 13	Process 2
Frame 4	None	None
Frame 5	Page 16	Process 1
⋮	⋮	

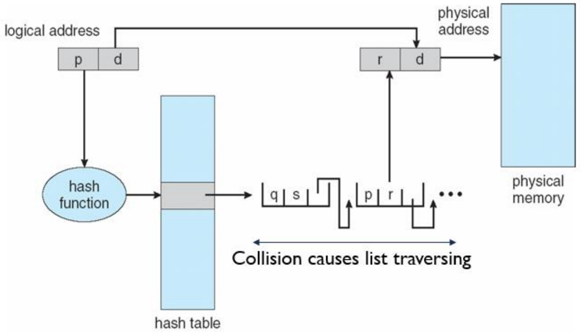


PAGE TABLE STRUCTURES

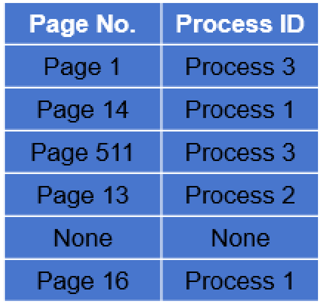
- Multi-Level Paging



- Hashed Page Table



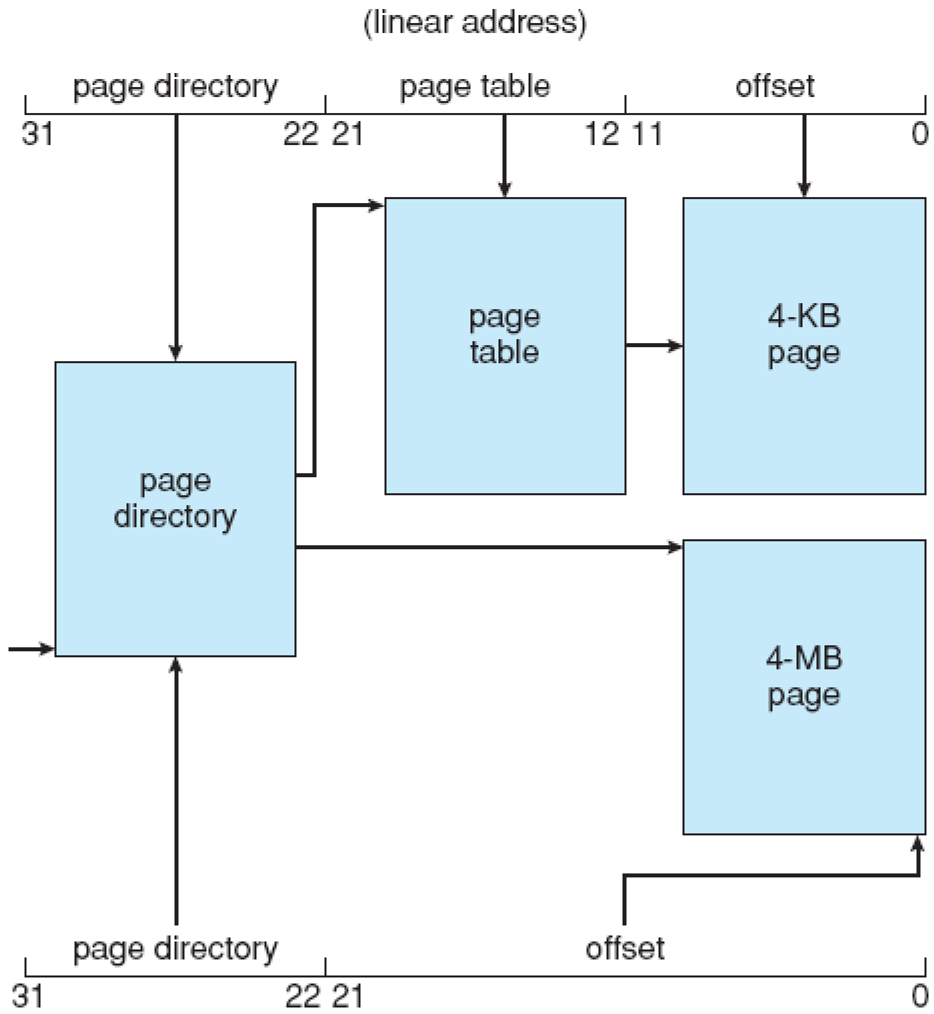
- Inverted Page Table



Page No.	Process ID
Page 1	Process 3
Page 14	Process 1
Page 511	Process 3
Page 13	Process 2
None	None
Page 16	Process 1

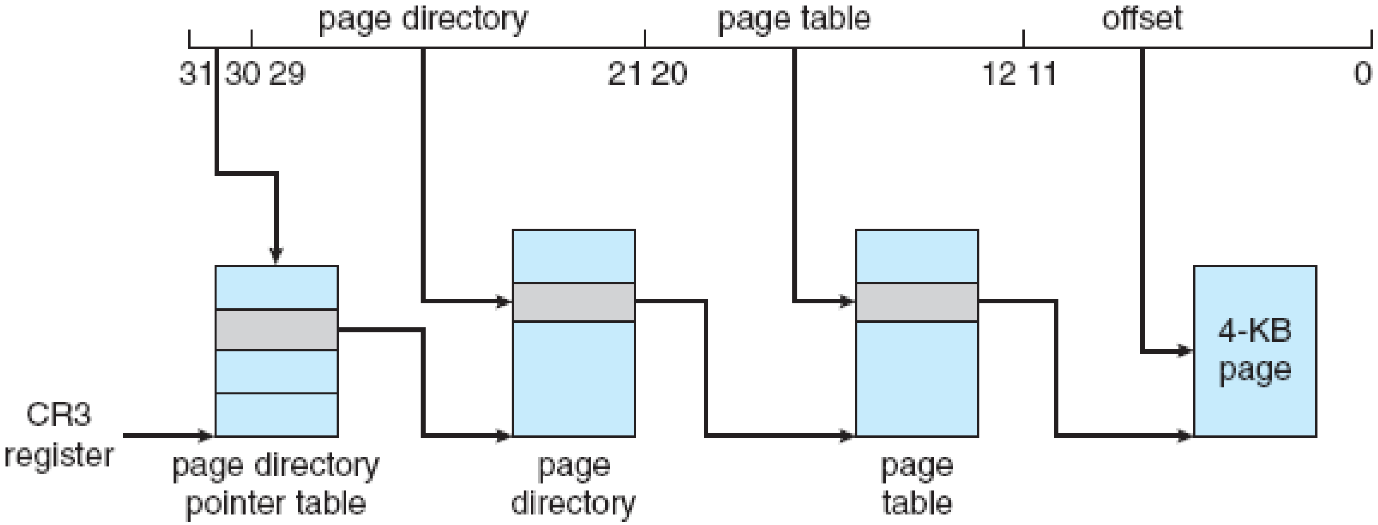
IA-32: INTEL ARCHITECTURE 32-BIT

- Two paging levels.
- Outer page table, called page directory.
- Two page sizes: 4-KB and 4-MB



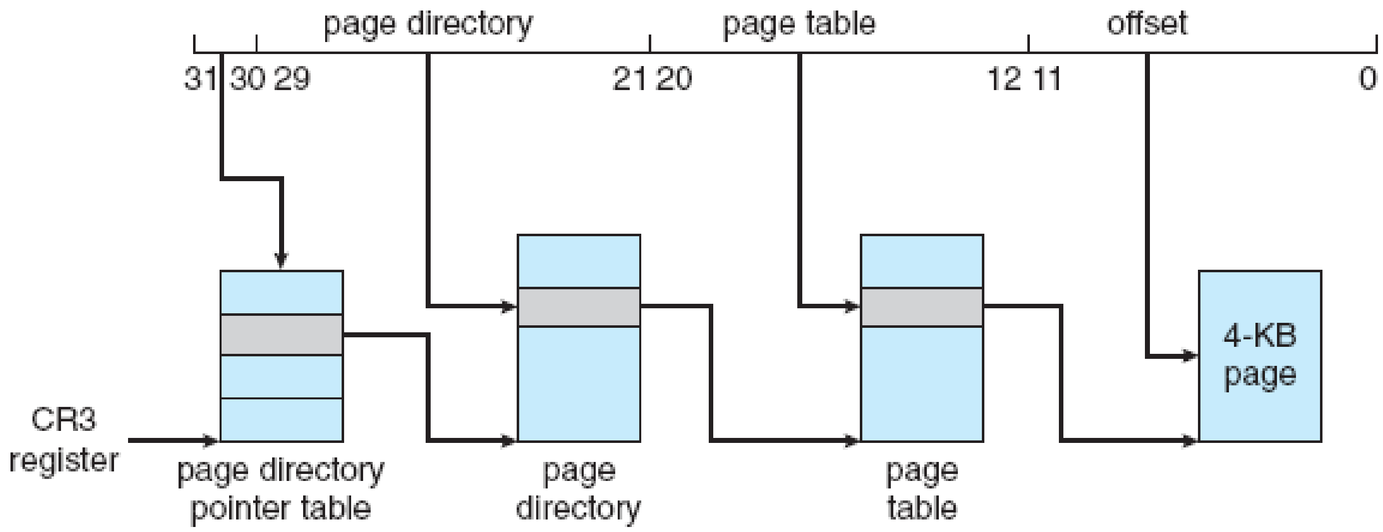
IA-32 PAE

- IA-32 Physical Address Extension.
- With 32-bit of address, only 4GB of ram can be supported.
- To solve this issue, intel added PAE: Physical Address Extension.



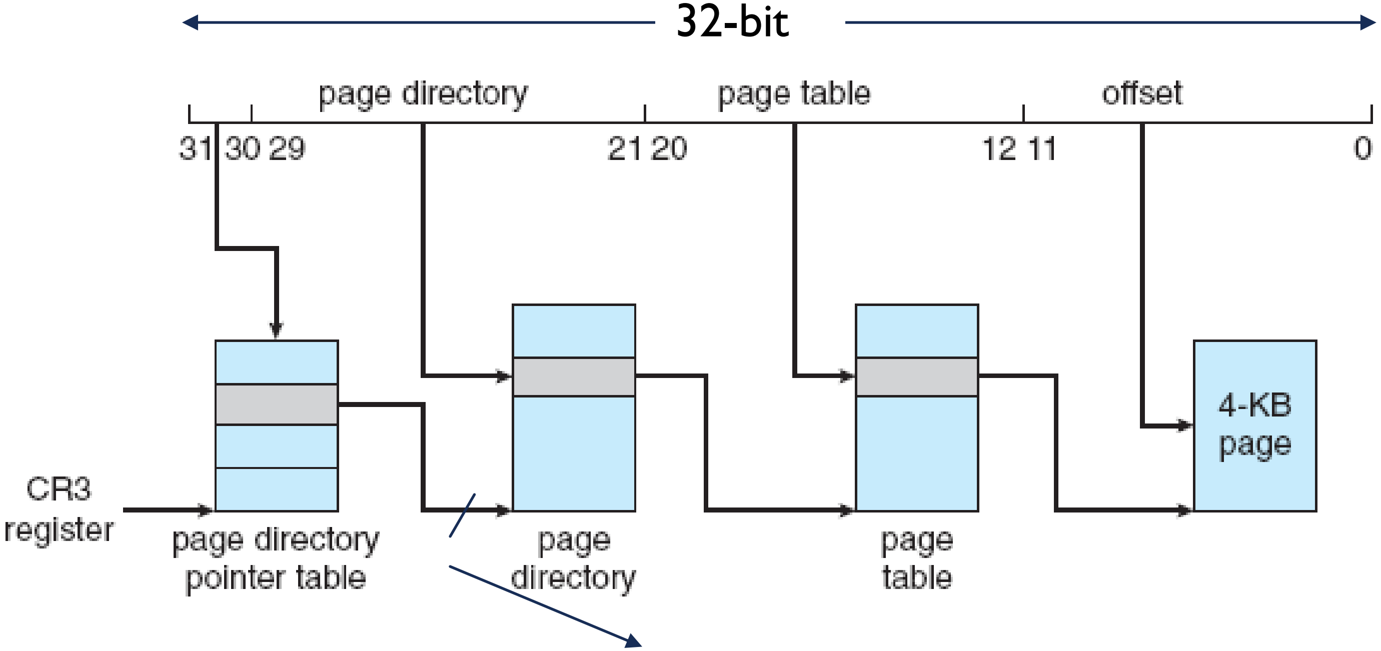
IA-32 PAE

- IA-32 Physical Address Extension.
- With 32-bit of address, only 4GB of ram can be supported.
- To solve this issue, intel added PAE: Physical Address Extension.
- Can be enabled/disabled on *supported* chips.
- The 32-bit address adopts a 3-level paging instead of 2 levels.



IA-32 PAE

- IA-32 Physical Address Extension.
- With 32-bit of address, only 4GB of ram can be supported.
- To solve this issue, intel added PAE: Physical Address Extension.
- Can be enabled/disabled on *supported* chips.
- The 32-bit address adopts a 3-level paging instead of 2 levels.



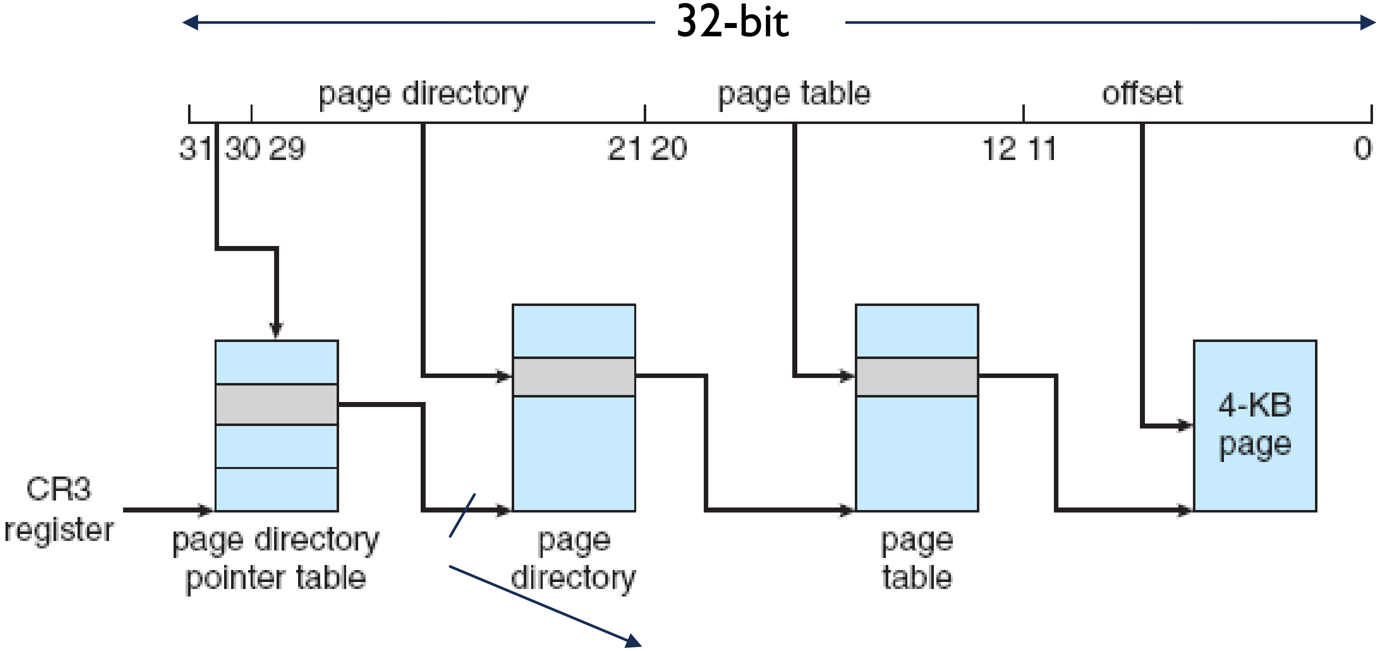
32-bit

64-bit (only 36 are used)

- Virtual address is still 32-bits .. But physical address is now 36-bits.

IA-32 PAE

- IA-32 Physical Address Extension.
- With 32-bit of address, only 4GB of ram can be supported.
- To solve this issue, intel added PAE: Physical Address Extension.
- Can be enabled/disabled on *supported* chips.
- The 32-bit address adopts a 3-level paging instead of 2 levels.



32-bit

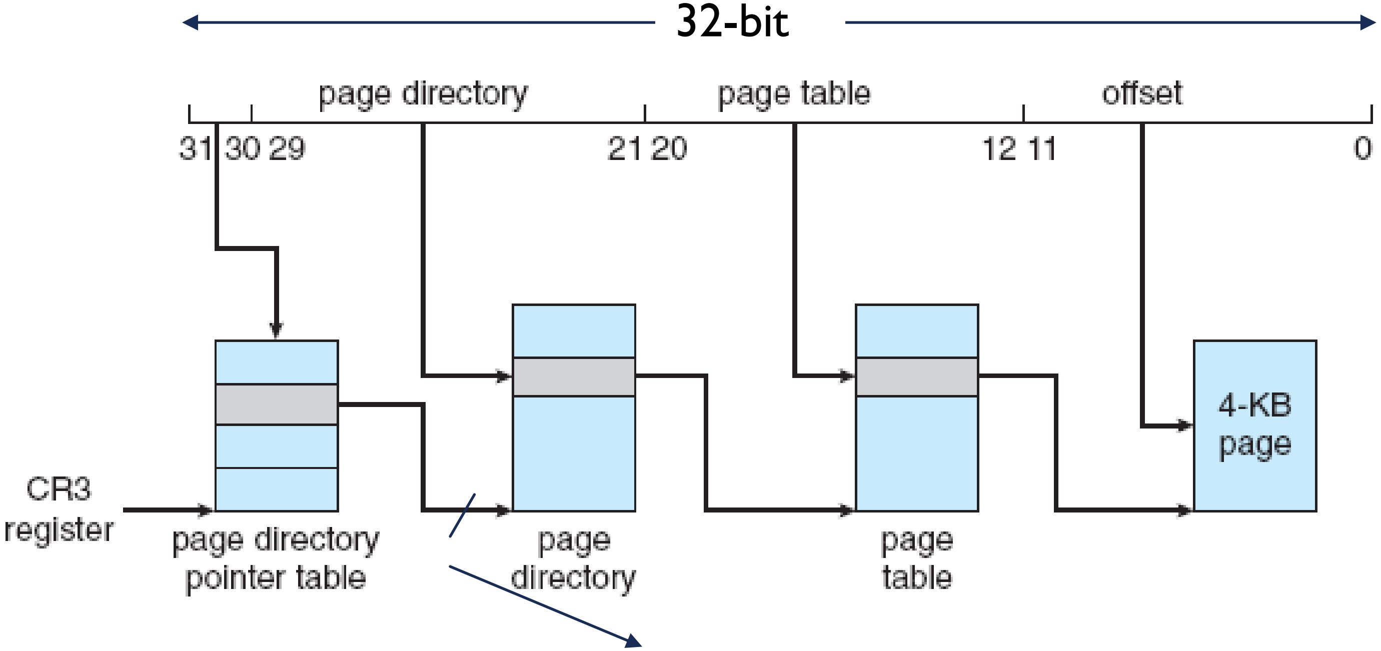
64-bit (only 36 are used)

What does that imply?

- Virtual address is still 32-bits .. But physical address is now 36-bits.

IA-32 PAE

- IA-32 Physical Address Extension.
- With 32-bit of address, only 4GB of ram can be supported.
- To solve this issue, intel added PAE: Physical Address Extension.
- Can be enabled/disabled on *supported* chips.
- The 32-bit address adopts a 3-level paging instead of 2 levels.



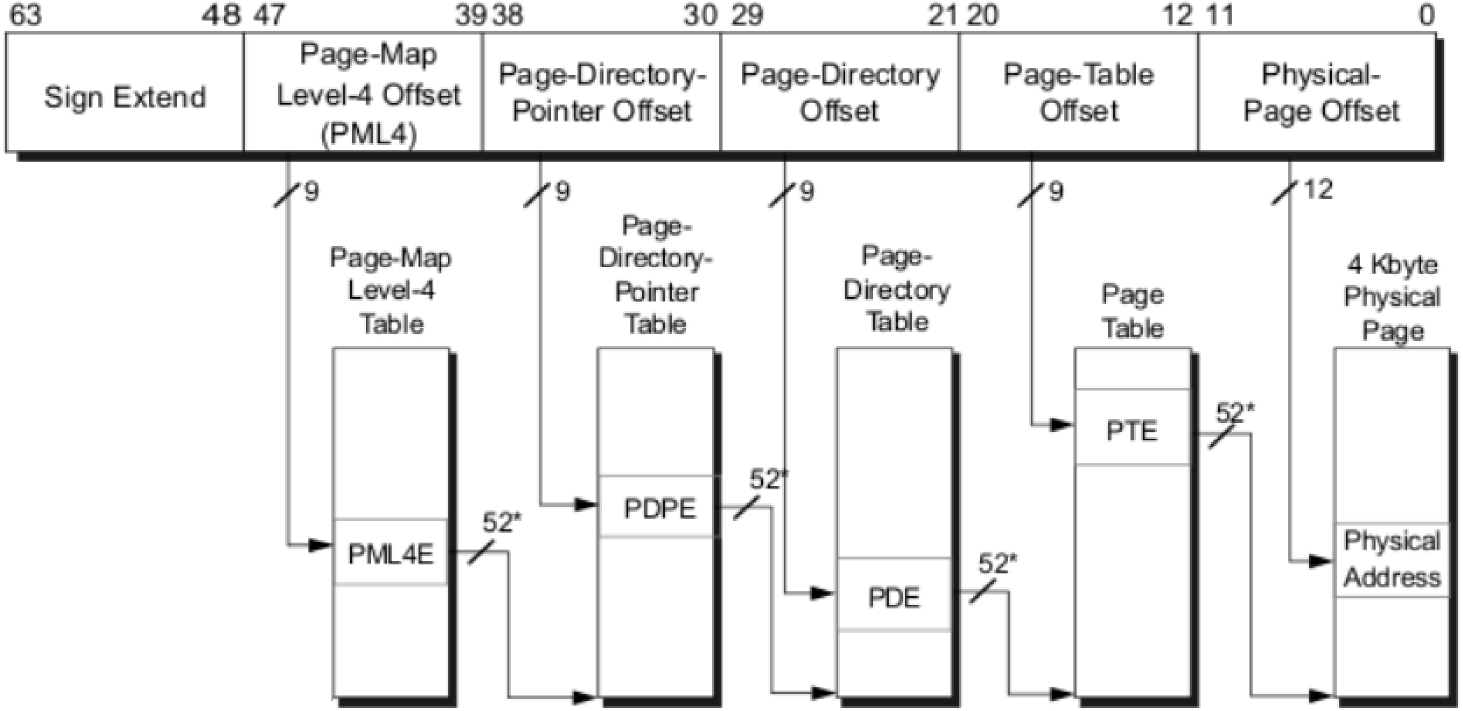
32-bit

64-bit (only 36 are used)

- Virtual address is still 32-bits .. But physical address is now 36-bits.
- Process is still limited to 4-GB, but system is not.

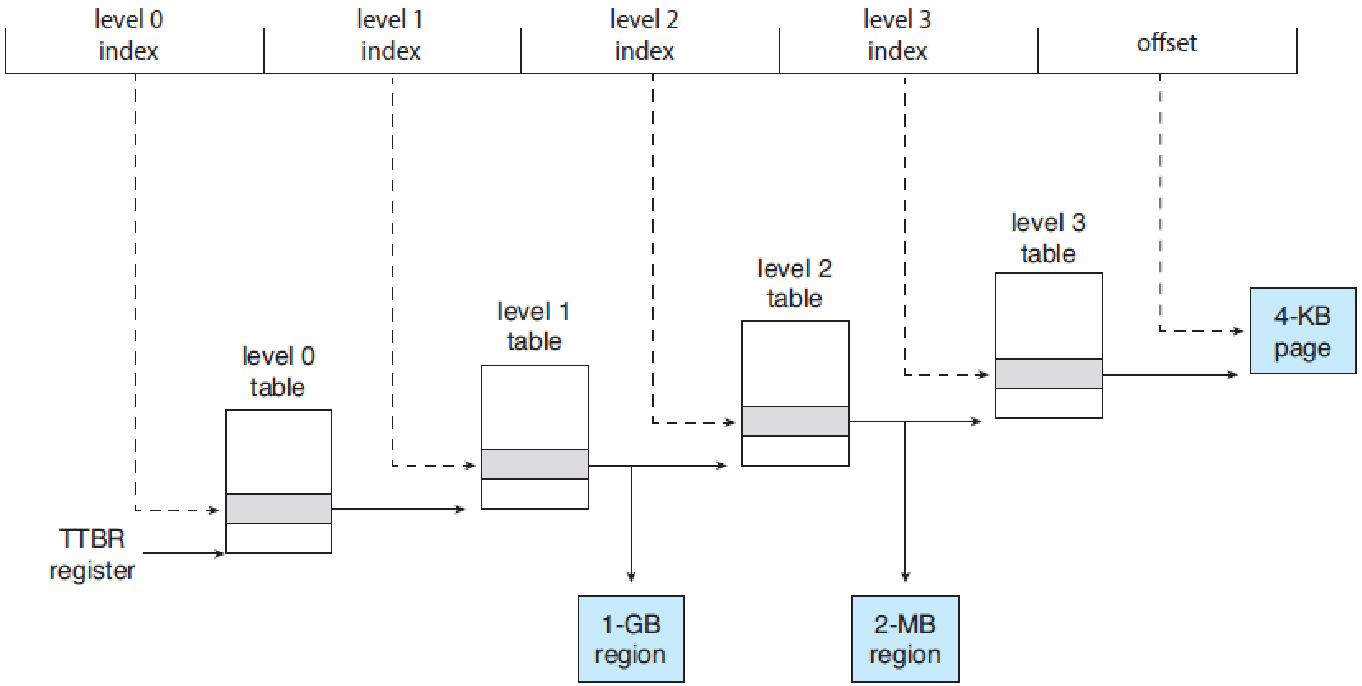
PAGING IMPLEMENTATION 64-X86

- Introduced by AMD in Opteron (2003)
- 64-bit virtual address (only 48-bits are used)
- 4-level paging
- TLB misses: very expensive, each level require a memory access.
- Memory Management Unit has its own cache dedicated for paging.

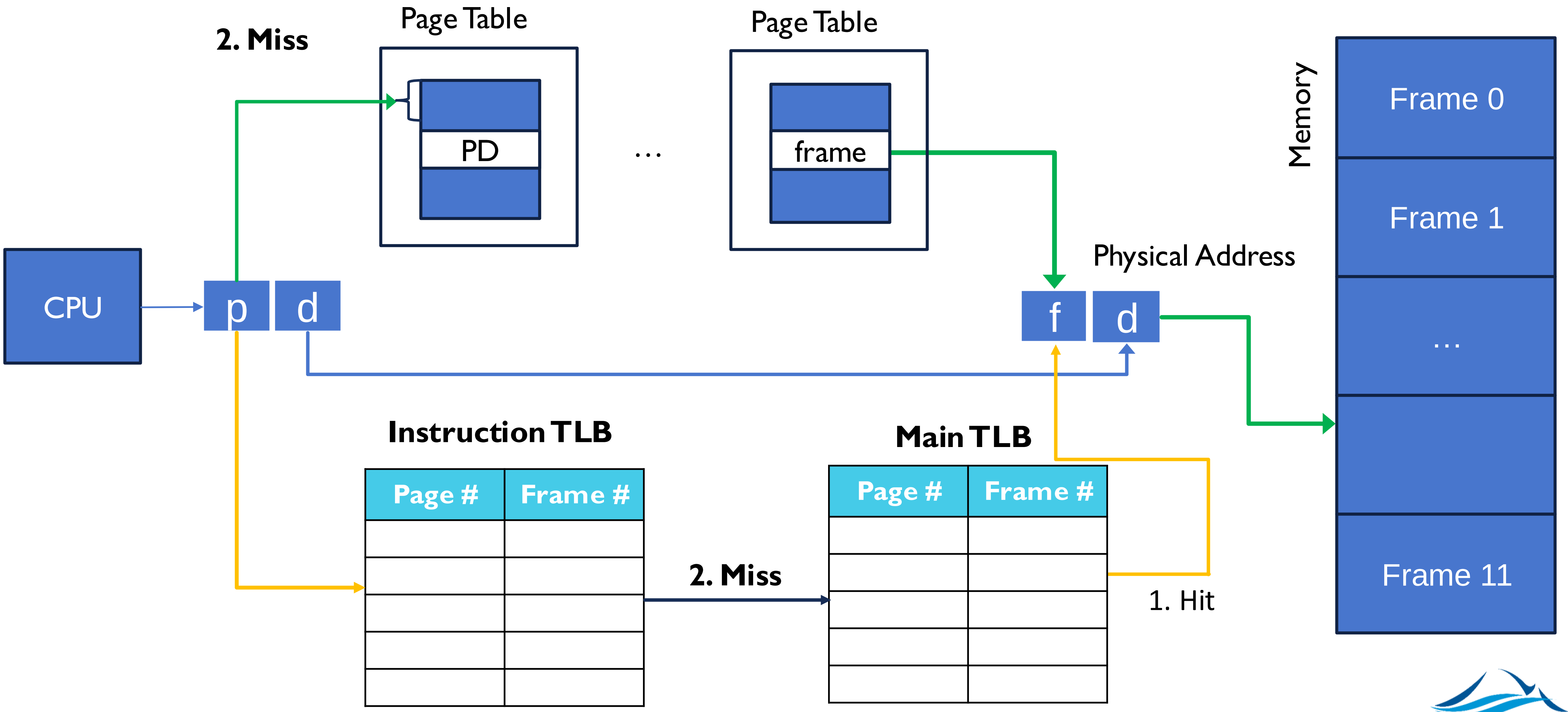


ARM V8 ARCHITECTURE

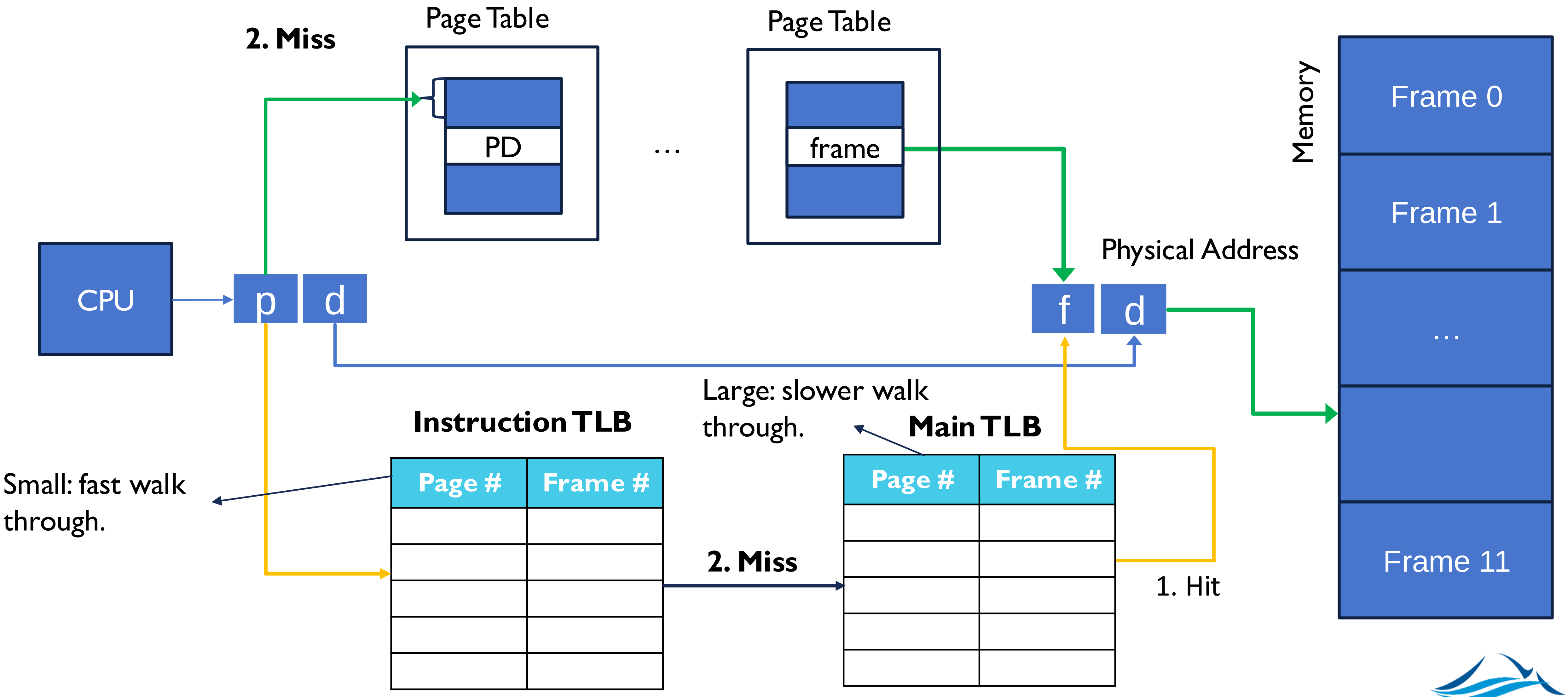
- ARM Architecture dominates mobile devices.
- Uses 64-bit (48-bit used) 4 level paging.
- Can offer contiguous memory in form of 'regions'.
- Two level TLBs:
  - Data TLB
  - Instruction TLB
  - Outer TLB for both Data and Instruction in case of TLB Miss



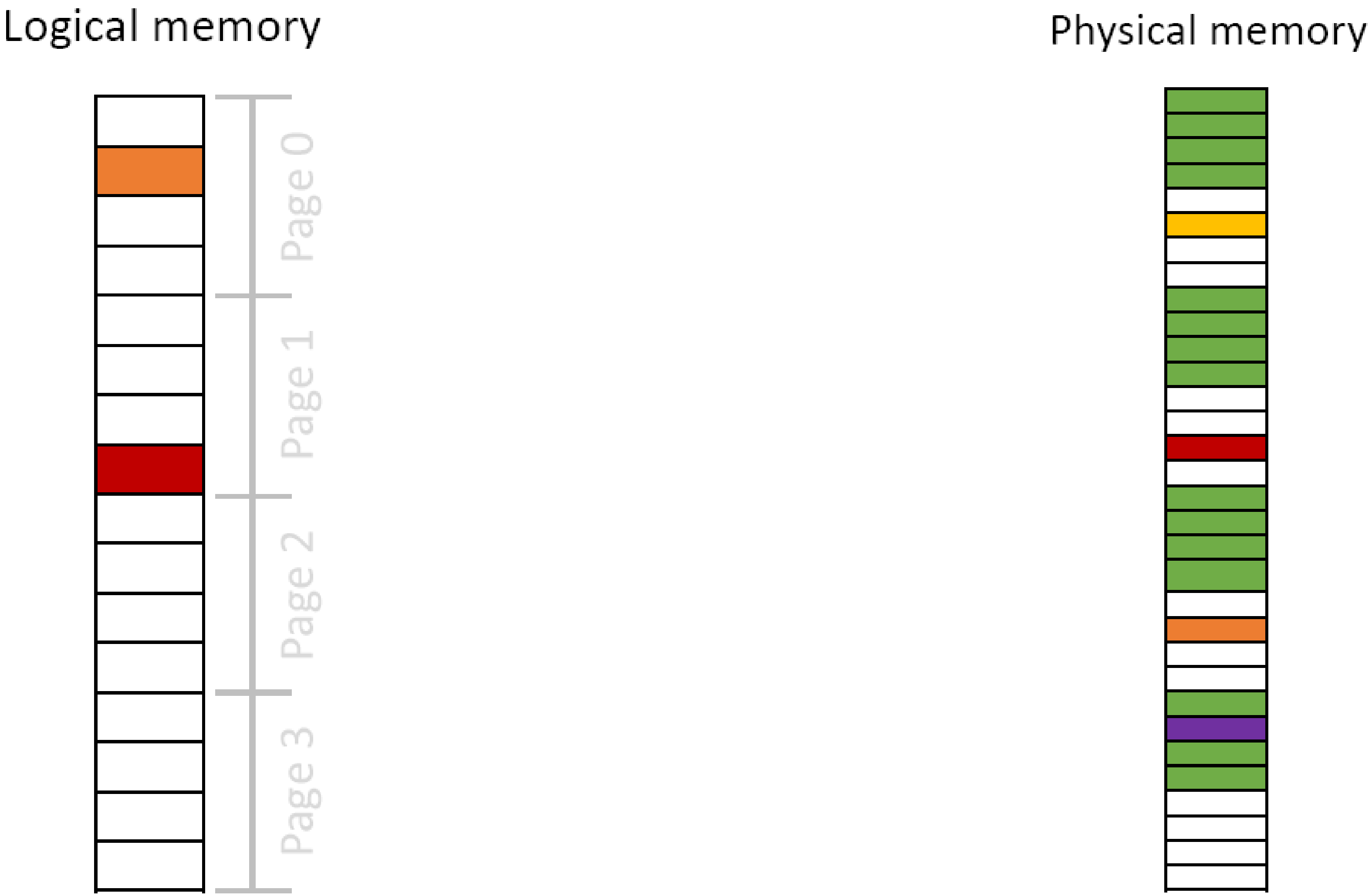
ARM V8 TWO-LEVEL TLB



ARM V8 TWO-LEVEL TLB

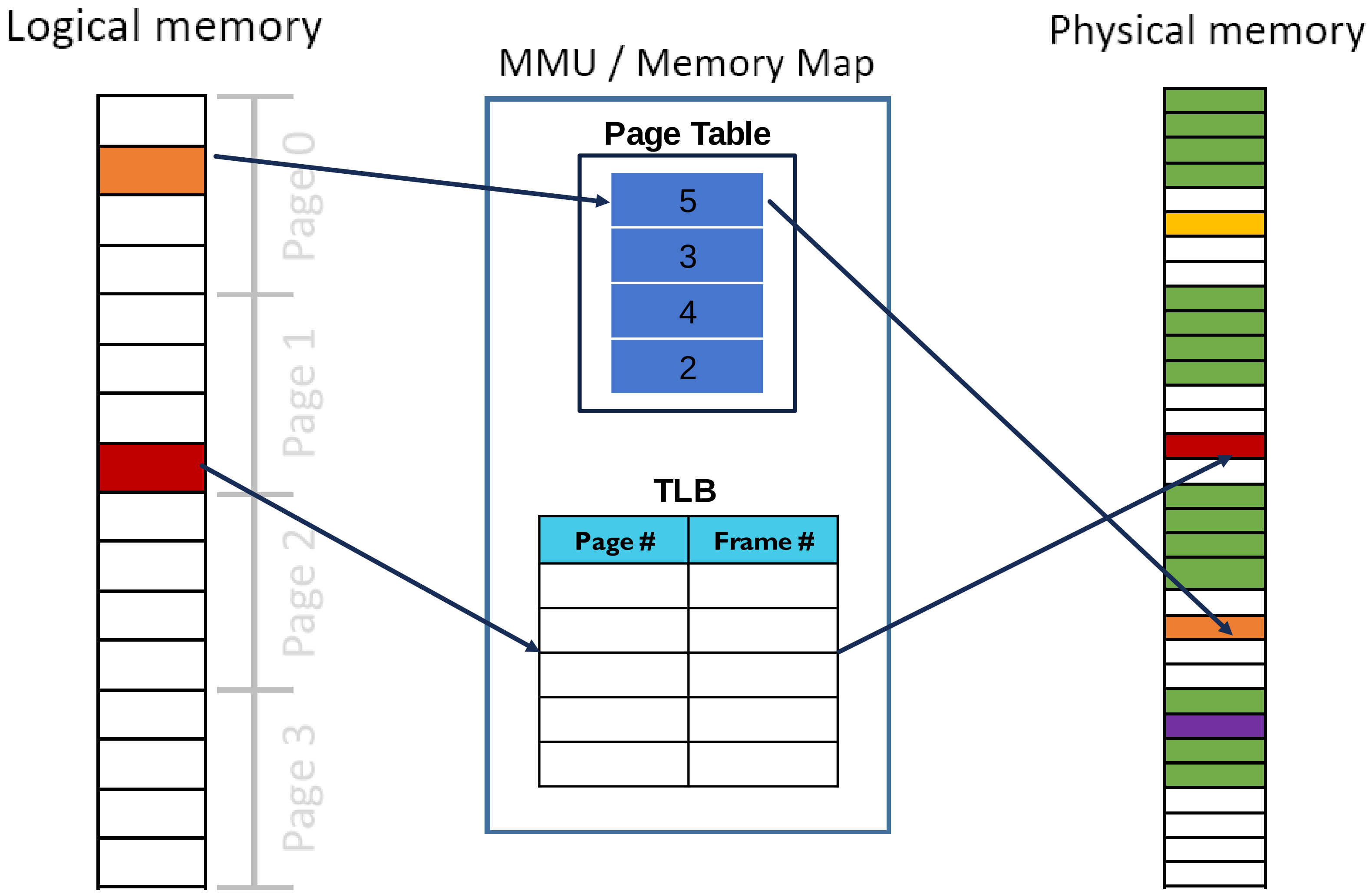


VIRTUAL MEMORY



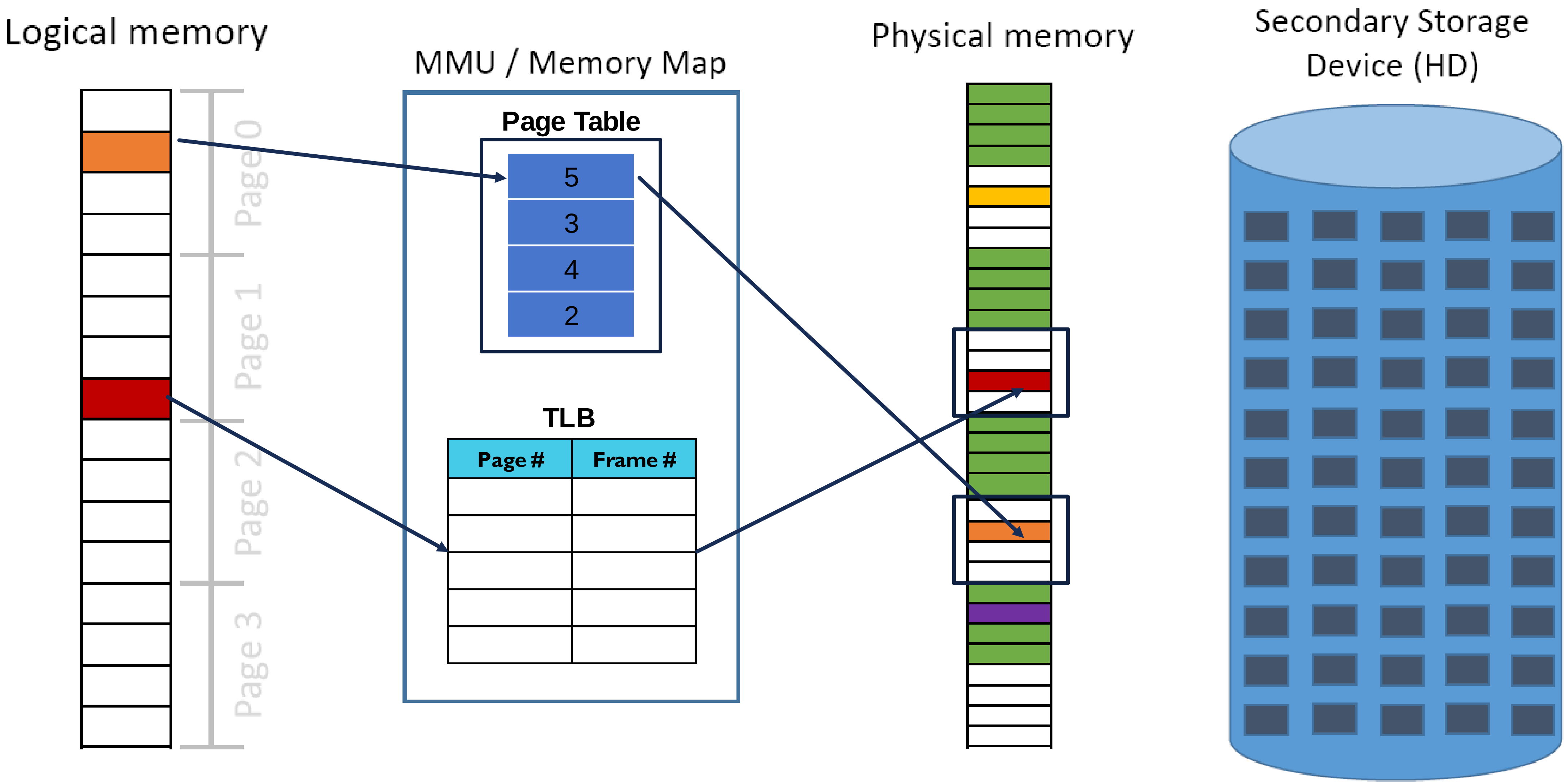
- Q:What are the two methods that can map logical memory to physical ones?

VIRTUAL MEMORY



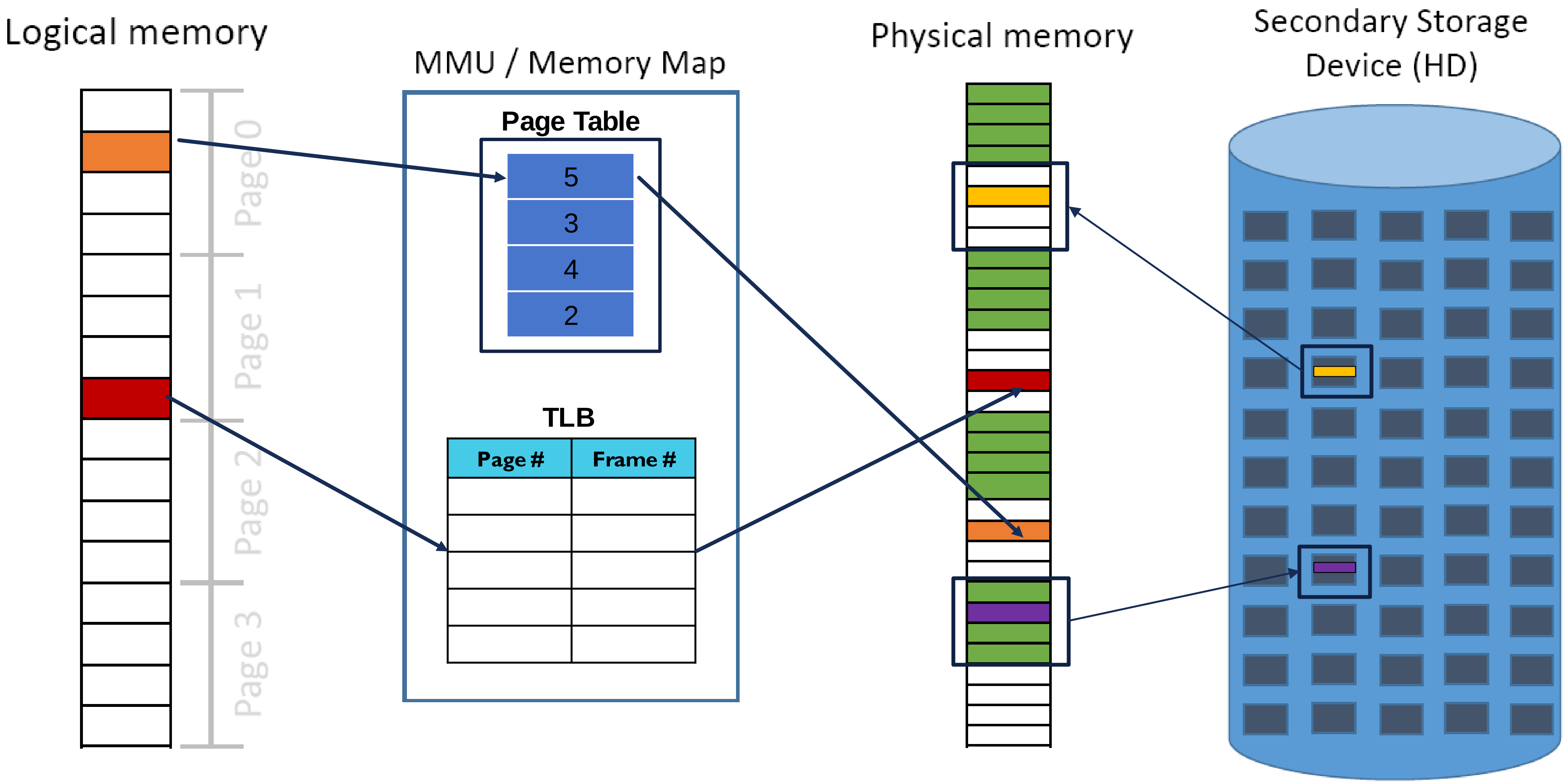
- Q:What are the two methods that can map logical memory to physical ones?
- Page Table and Translation Look Aside Buffers.

VIRTUAL MEMORY

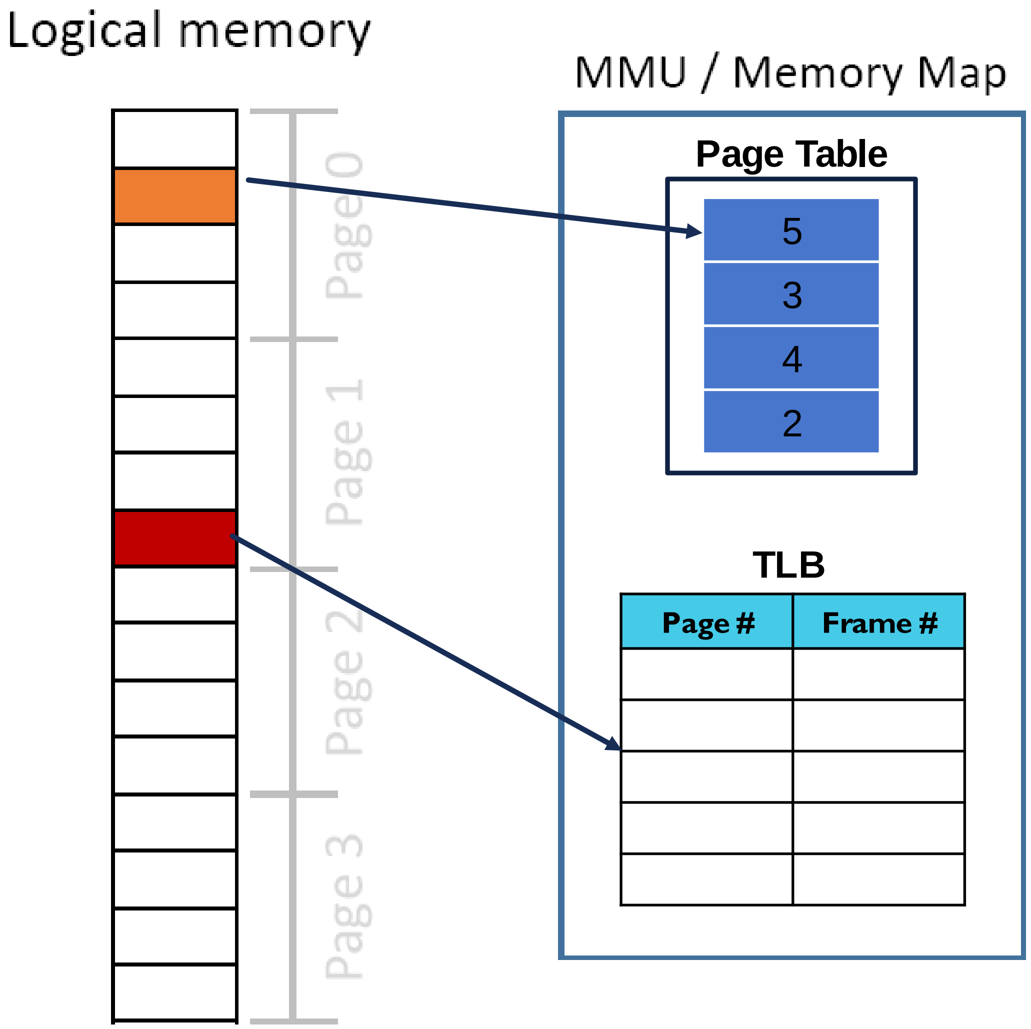


Page #	Frame #

VIRTUAL MEMORY



VIRTUAL MEMORY



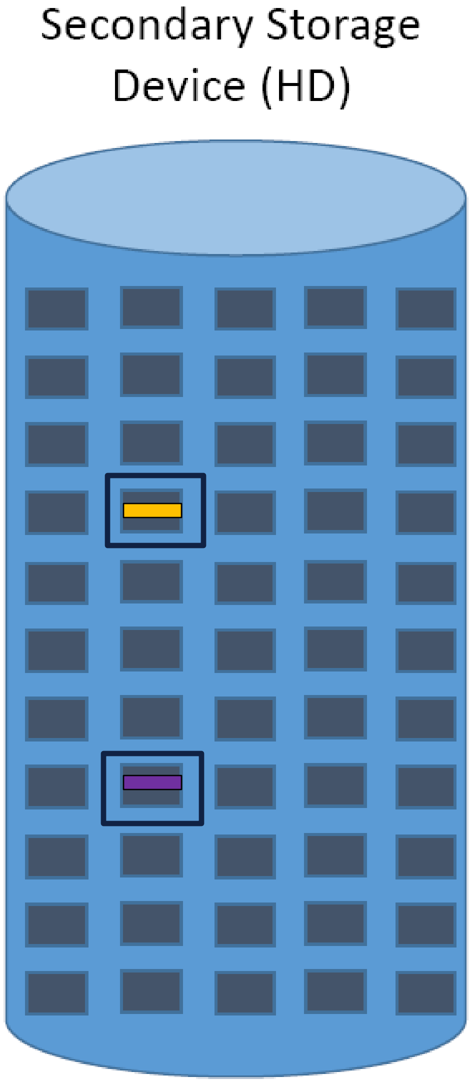
Q: How to allocate frames in physical memory?

Q: What to do if the physical memory is full?

Q: What frames need to be written out?

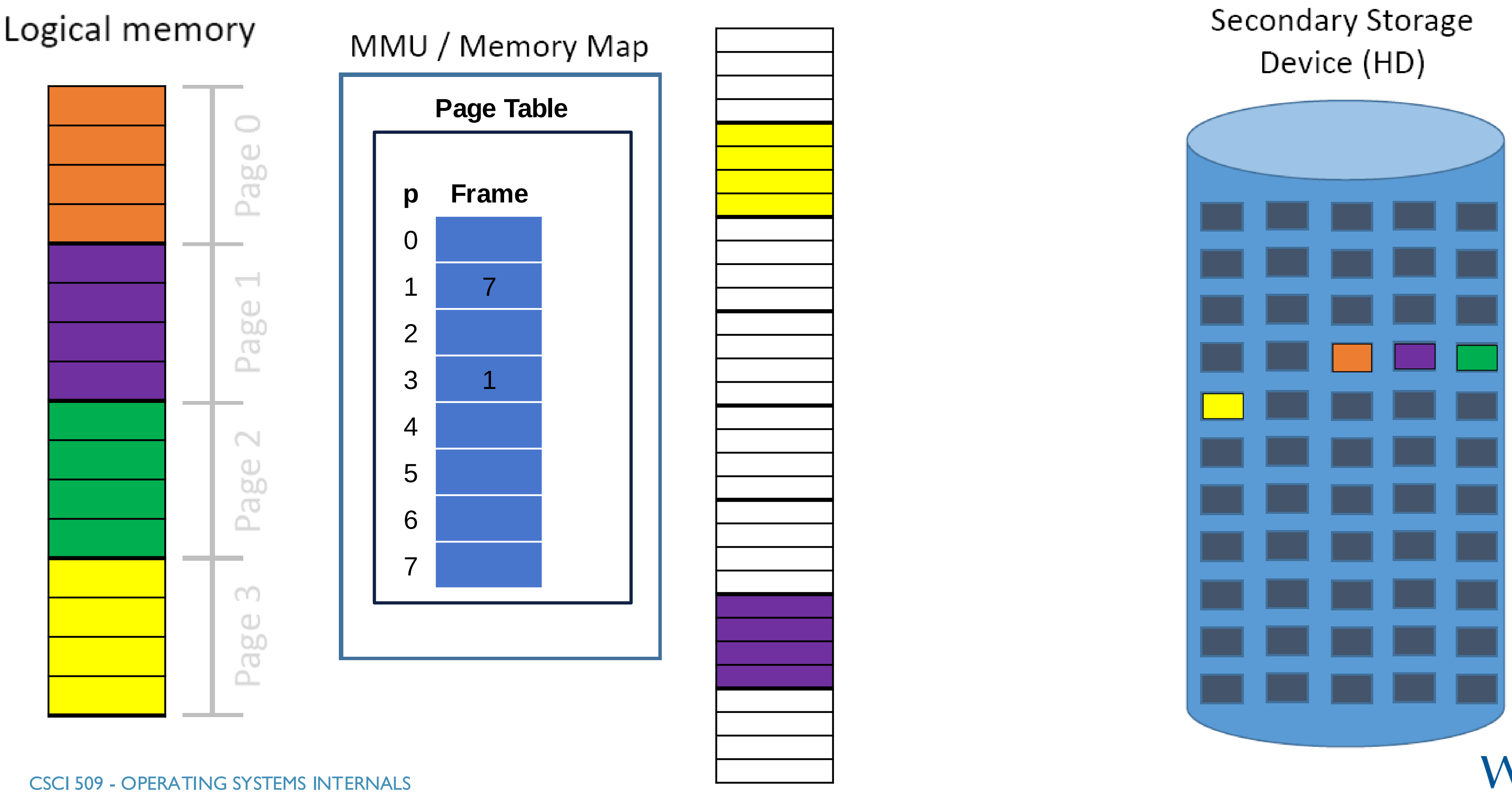
Q: How to decide what page to bring in memory?

Q: What's the impact on performance?



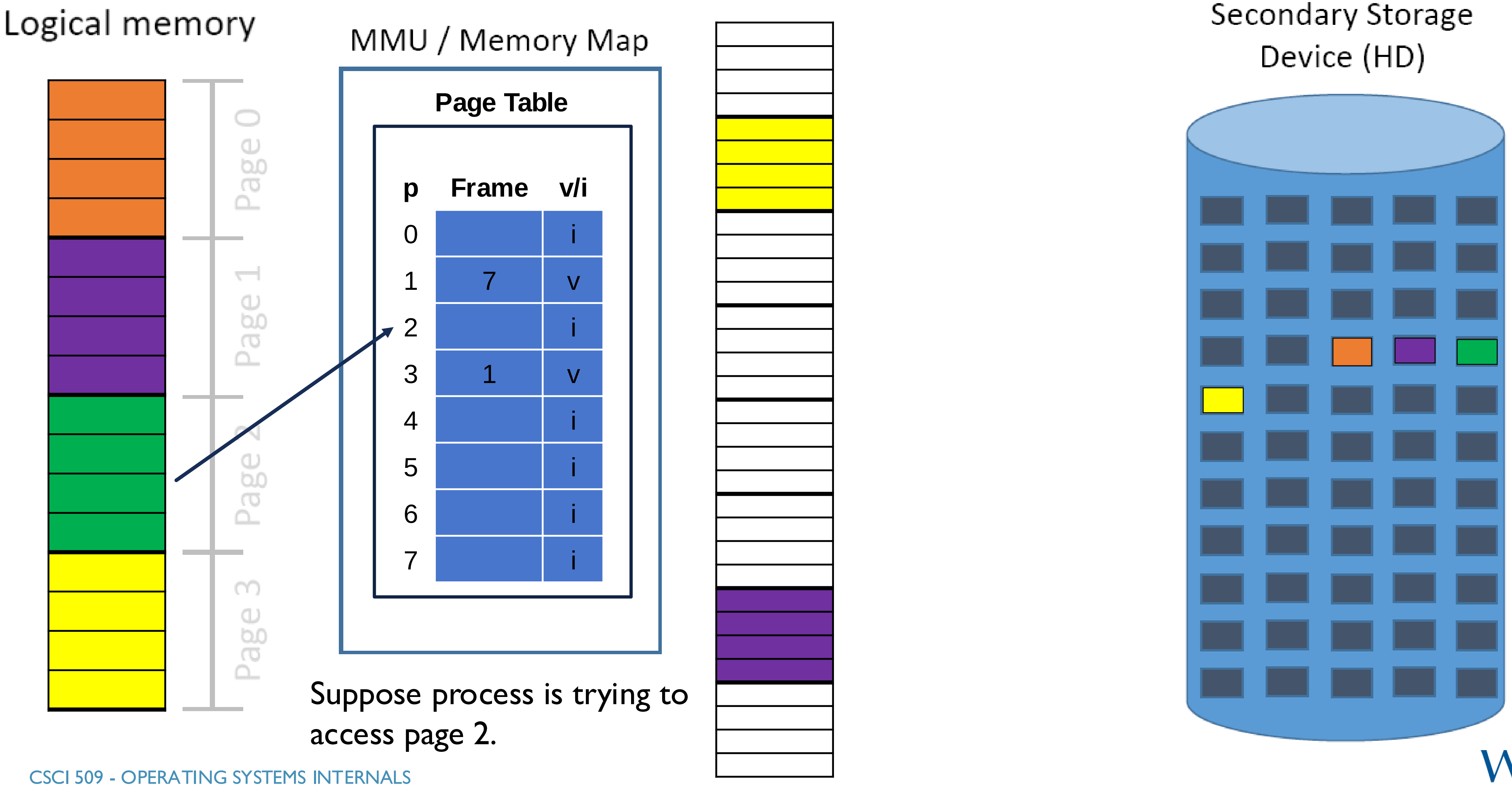
VIRTUAL MEMORY

Q: How to allocate frames in physical memory?



p	Frame
0	
1	7
2	
3	1
4	
5	
6	
7	

FRAME ALLOCATION

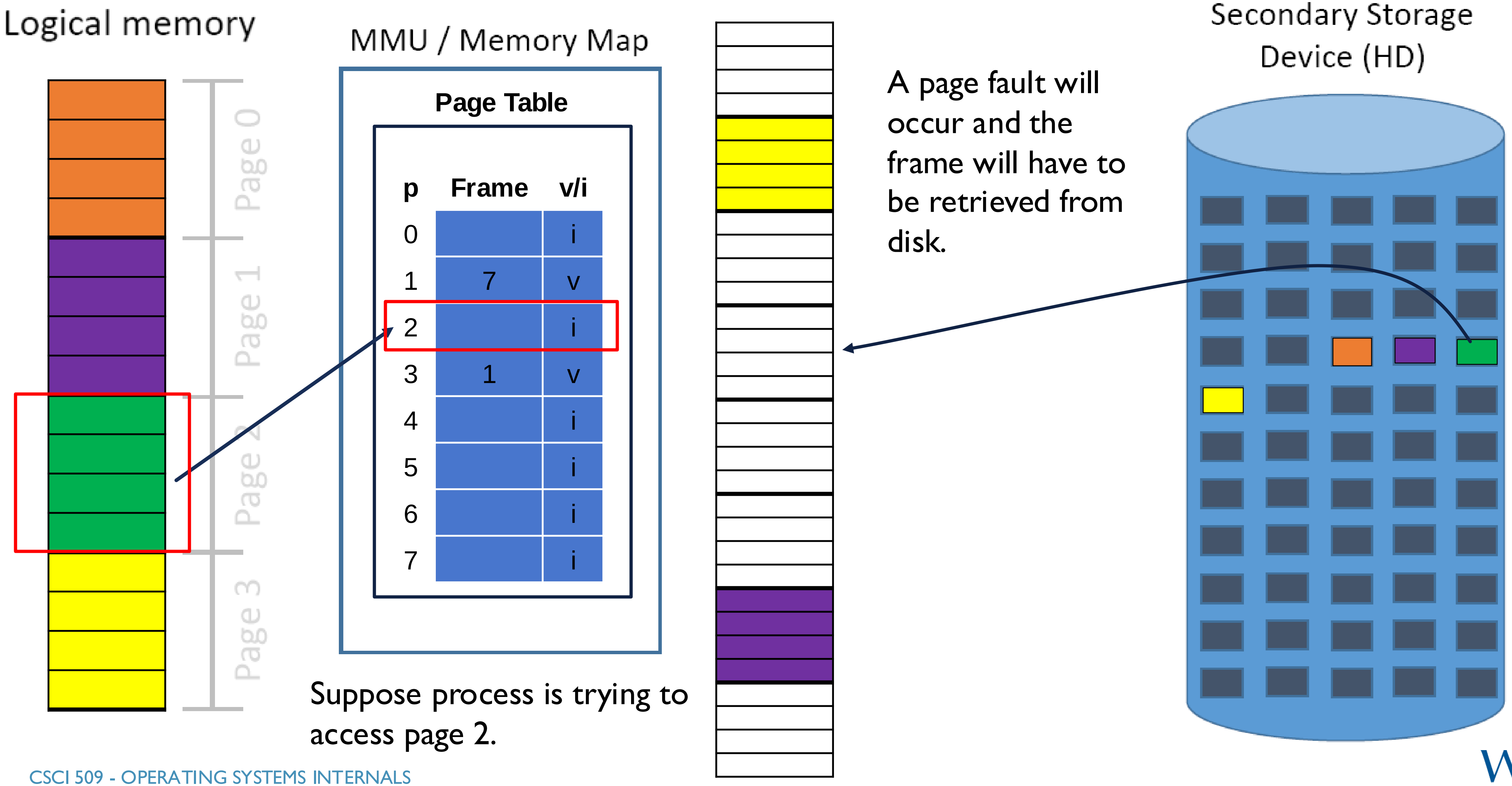


Page Table

p	Frame	v/i
0		i
1	7	v
2		i
3	1	v
4		i
5		i
6		i
7		i

Suppose process is trying to access page 2.

FRAME ALLOCATION



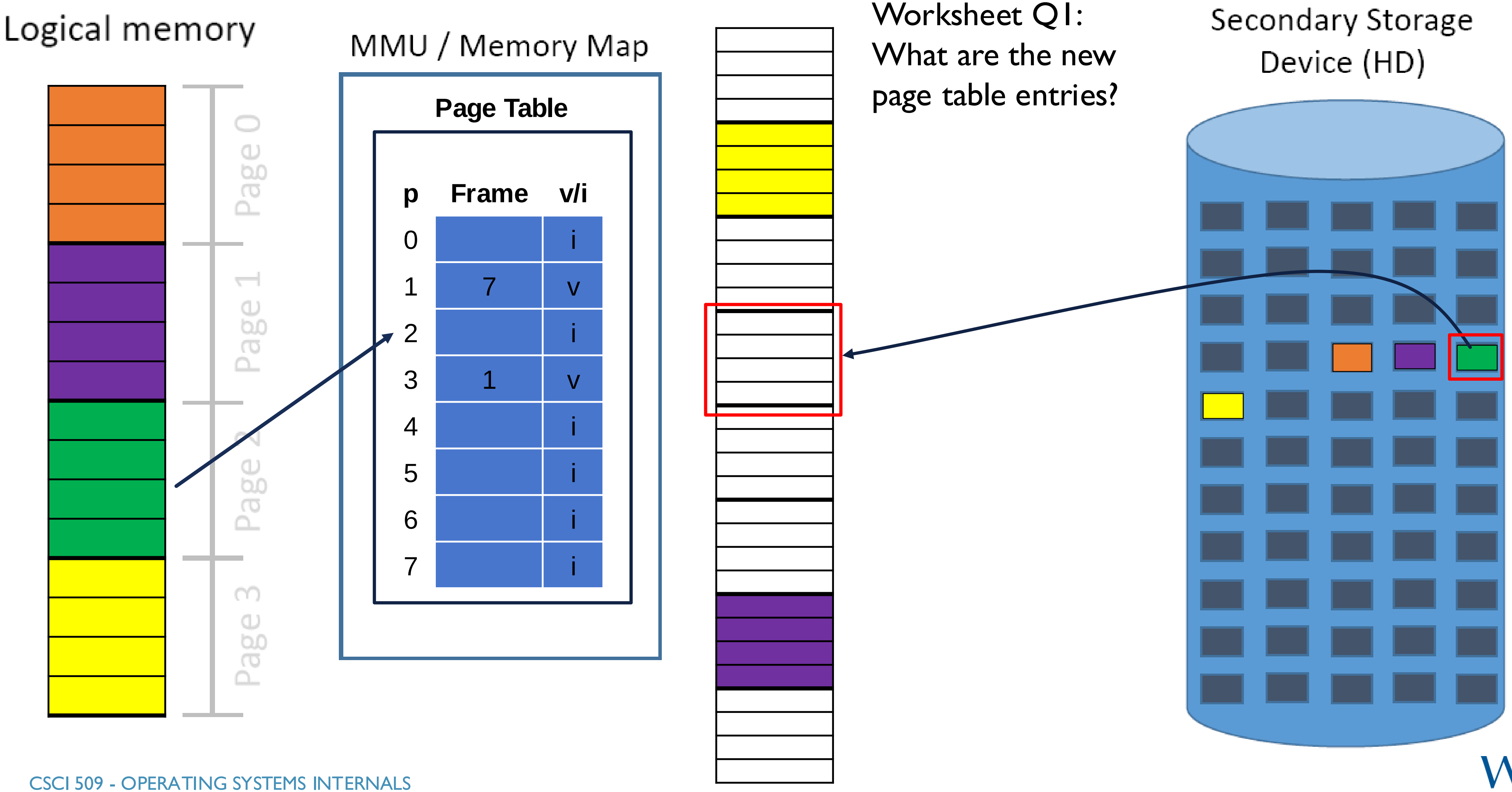
Page Table

p	Frame	v/i
0		i
1	7	v
2		i
3	1	v
4		i
5		i
6		i
7		i

A page fault will occur and the frame will have to be retrieved from disk.

Suppose process is trying to access page 2.

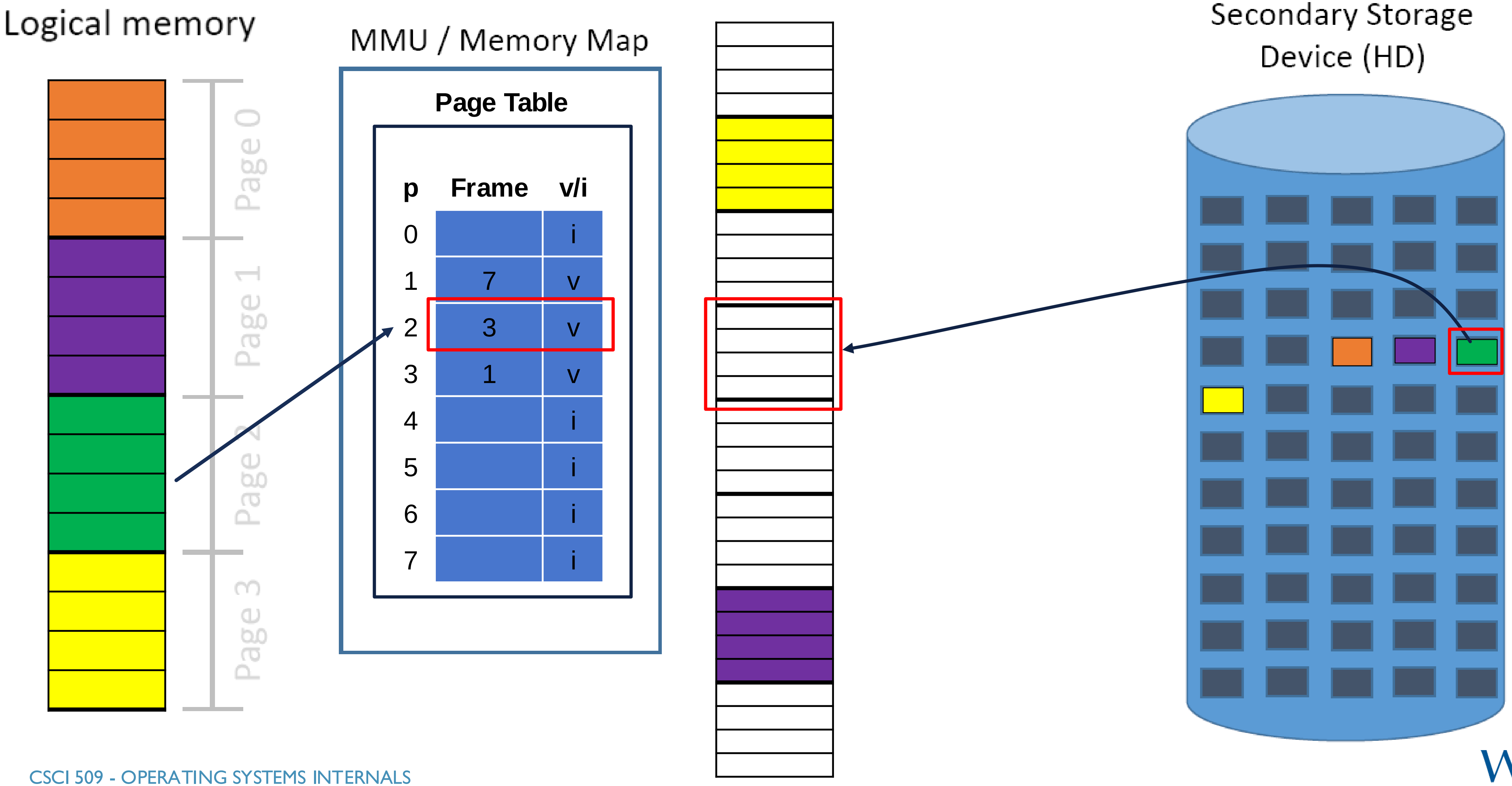
FRAME ALLOCATION



p	Frame	v/i
0		i
1	7	v
2		i
3	1	v
4		i
5		i
6		i
7		i

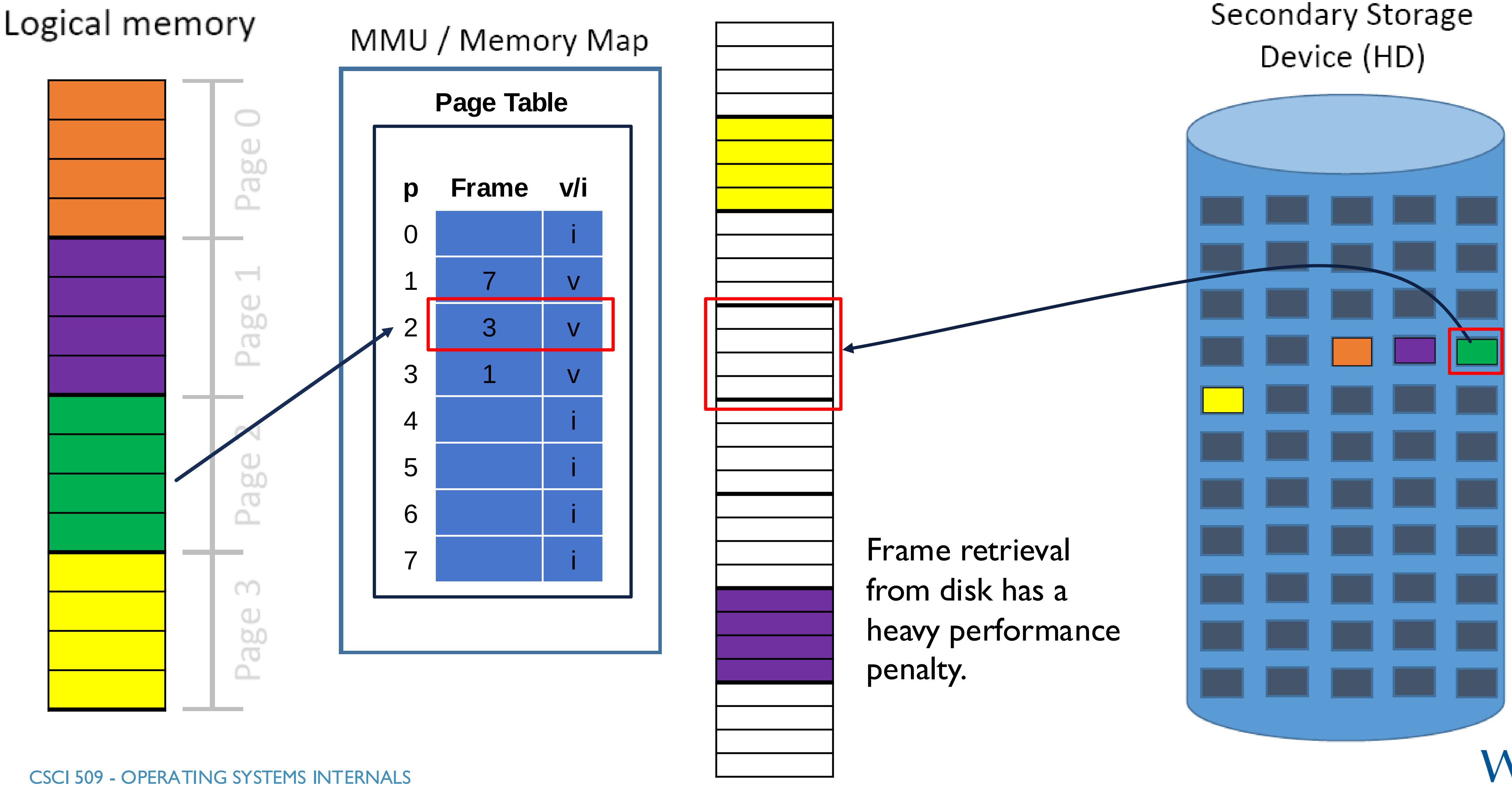
Worksheet Q1:
What are the new page table entries?

FRAME ALLOCATION



p	Frame	v/i
0		i
1	7	v
2	3	v
3	1	v
4		i
5		i
6		i
7		i

FRAME ALLOCATION



p	Frame	v/i
0		i
1	7	v
2	3	v
3	1	v
4		i
5		i
6		i
7		i

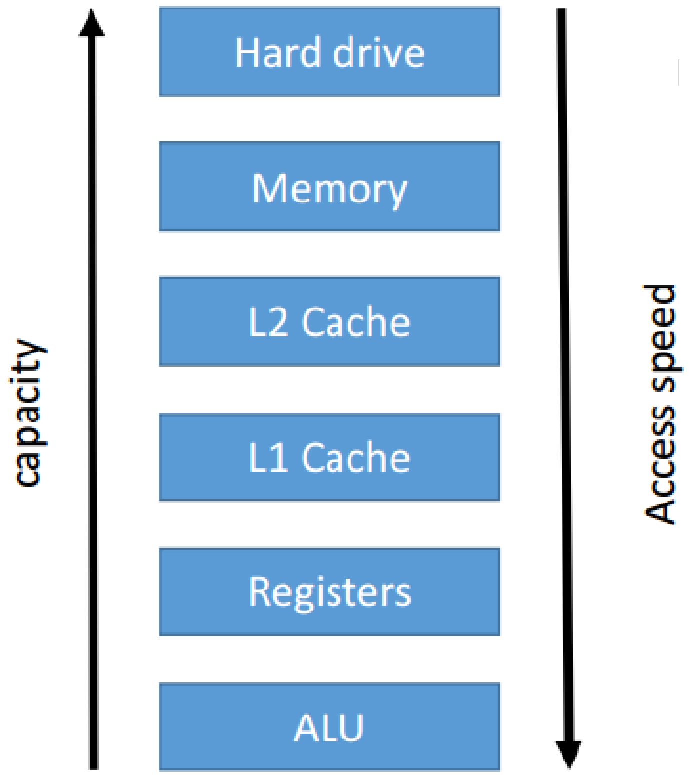
Frame retrieval from disk has a heavy performance penalty.

PAGE FAULT PENALTY

Ideal Case : all of the pages needed by a program are in physical memory

Real-world Case : Swapping in (and out) must occur

Q: What time delay (penalty) does a program incur due to demand paging in response to a single page fault?

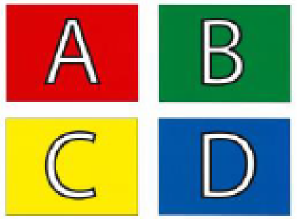


PAGE FAULT PENALTY

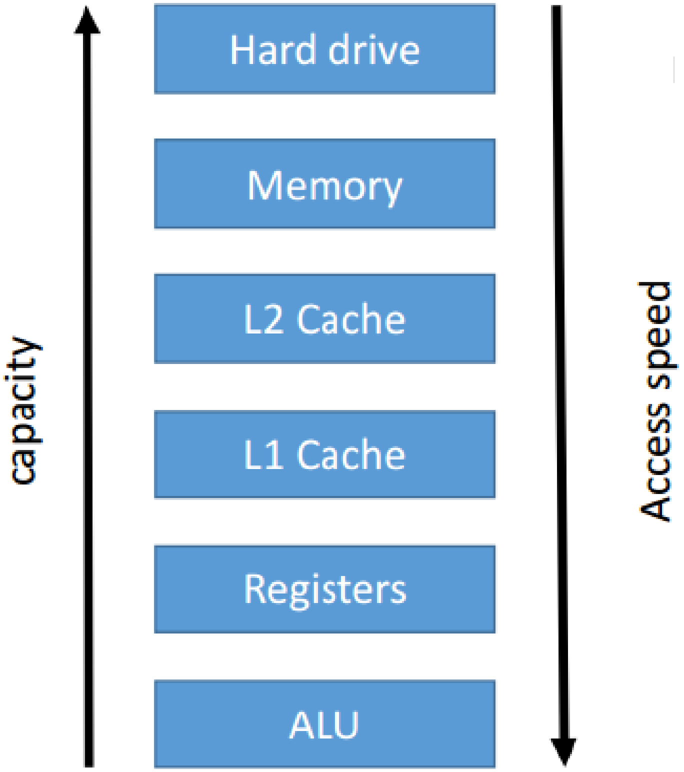
Ideal Case : all of the pages needed by a program are in physical memory

Real-world Case : Swapping in (and out) must occur

Q: What time delay (penalty) does a program incur due to demand paging in response to a single page fault?



A : 1 millisecond
B : 3 nanoseconds
C : 2 microseconds
D : 1 picosecond

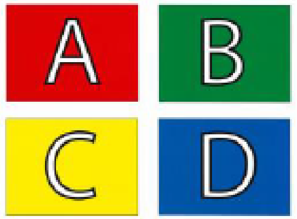


PAGE FAULT PENALTY

Ideal Case : all of the pages needed by a program are in physical memory

Real-world Case : Swapping in (and out) must occur

Q: What time delay (penalty) does a program incur due to demand paging in response to a single page fault?

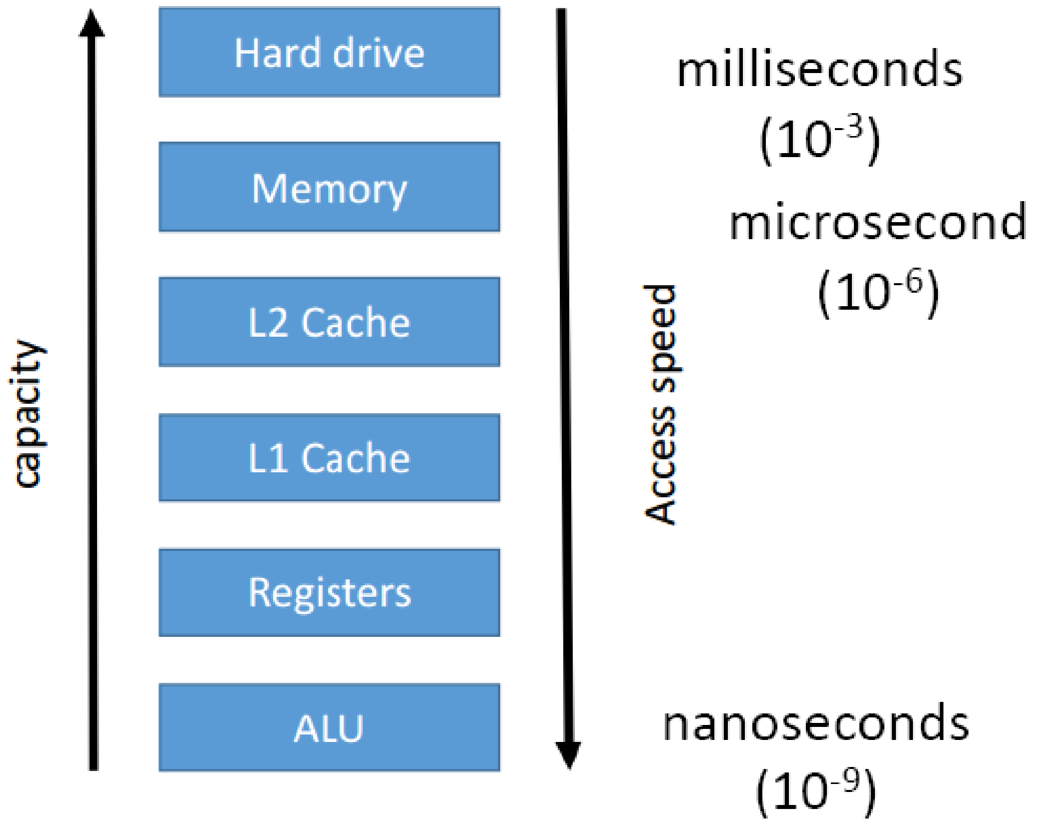


[✓ **A :** 1 millisecond]

B : 3 nanoseconds

C : 2 microseconds

D : 1 picosecond



PAGE FAULT PENALTY

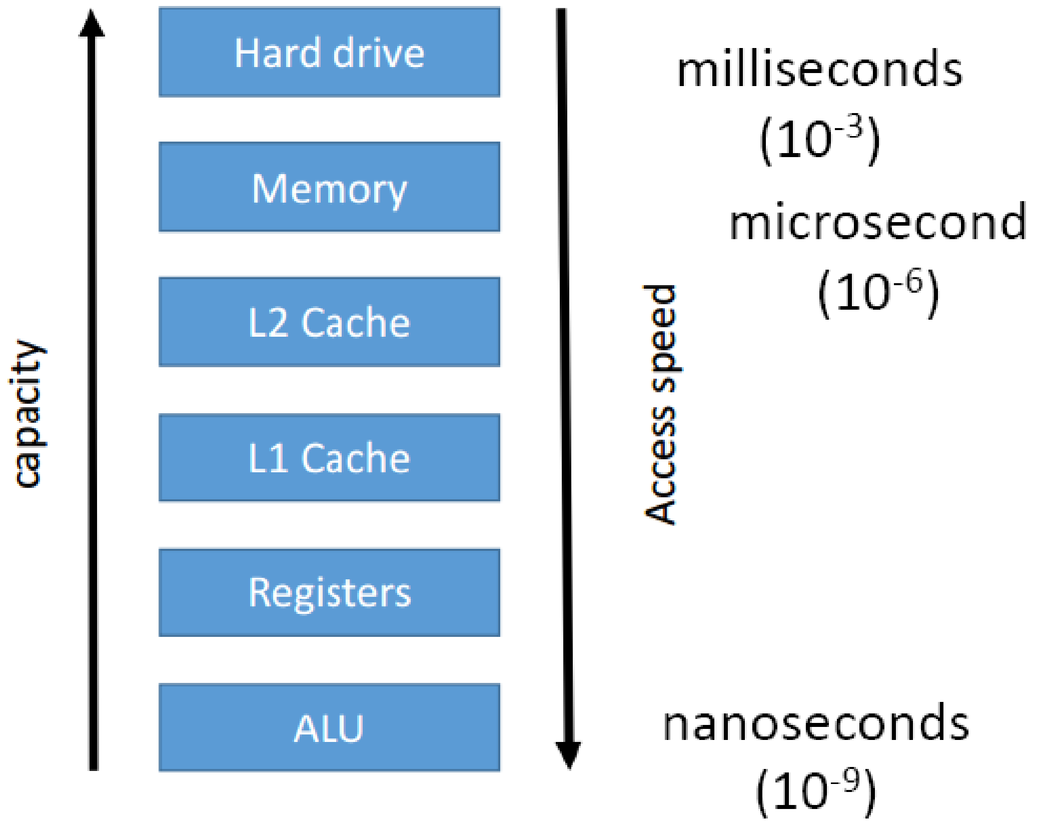
Ideal Case : all of the pages needed by a program are in physical memory

Real-world Case : Swapping in (and out) must occur

Q: What time delay (penalty) does a program incur due to demand paging in response to a single page fault?

- Service page fault
- swap in the needed frame
- Restart the process

The approximate steps taken



PAGE FAULT PENALTY

Ideal Case : all of the pages needed by a program are in physical memory

Real-world Case : Swapping in (and out) must occur

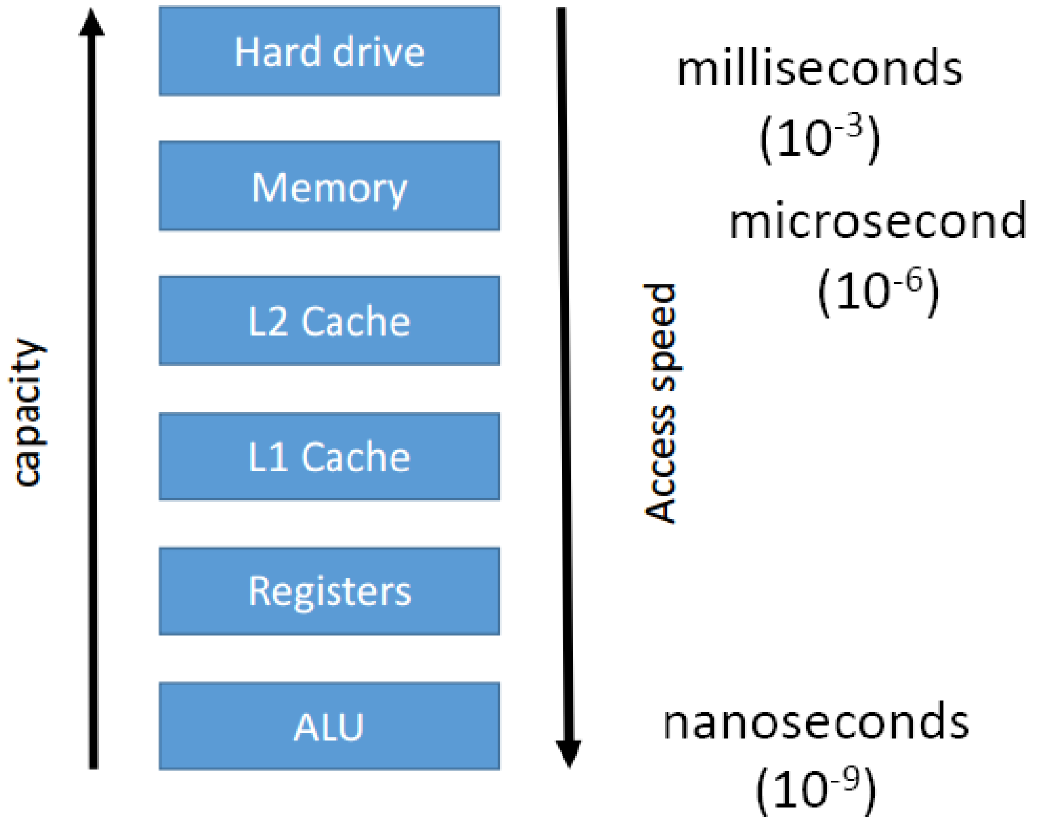
Q: What time delay (penalty) does a program incur due to demand paging in response to a single page fault?

- Service page fault
- swap in the needed frame
- Restart the process

Servicing the page fault means interrupting the process, and saving the process's state (PCB)

Context Switch

~50 microseconds



capacity

Hard drive

Memory

L2 Cache

L1 Cache

Registers

ALU

Access speed

milliseconds (10^{-3})

microsecond (10^{-6})

nanoseconds (10^{-9})

PAGE FAULT PENALTY

Ideal Case : all of the pages needed by a program are in physical memory

Real-world Case : Swapping in (and out) must occur

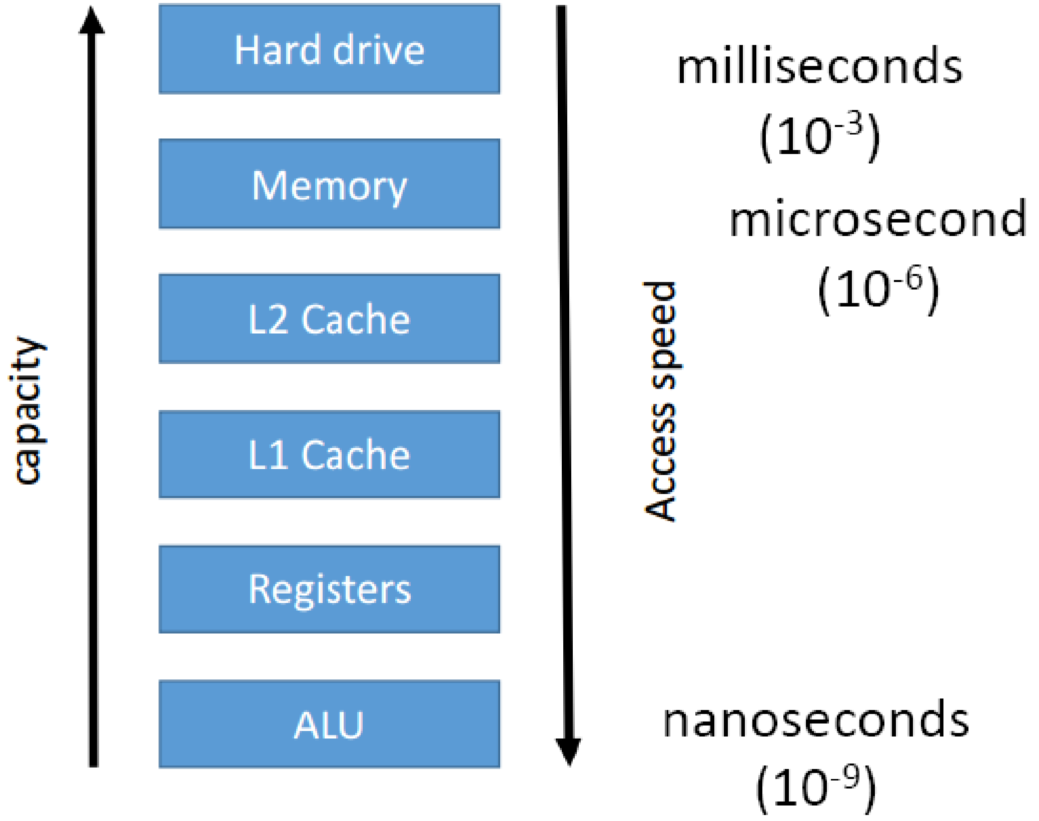
Q: What time delay (penalty) does a program incur due to demand paging in response to a single page fault?

- Service page fault
- swap in the needed frame
- Restart the process

Issue read (I/O), and schedule other processes to run

Bring in needed frame corresponding to needed page, update page table, wait for OS to schedule the process again

~50 microseconds



PAGE FAULT PENALTY

Ideal Case : all of the pages needed by a program are in physical memory

Real-world Case : Swapping in (and out) must occur

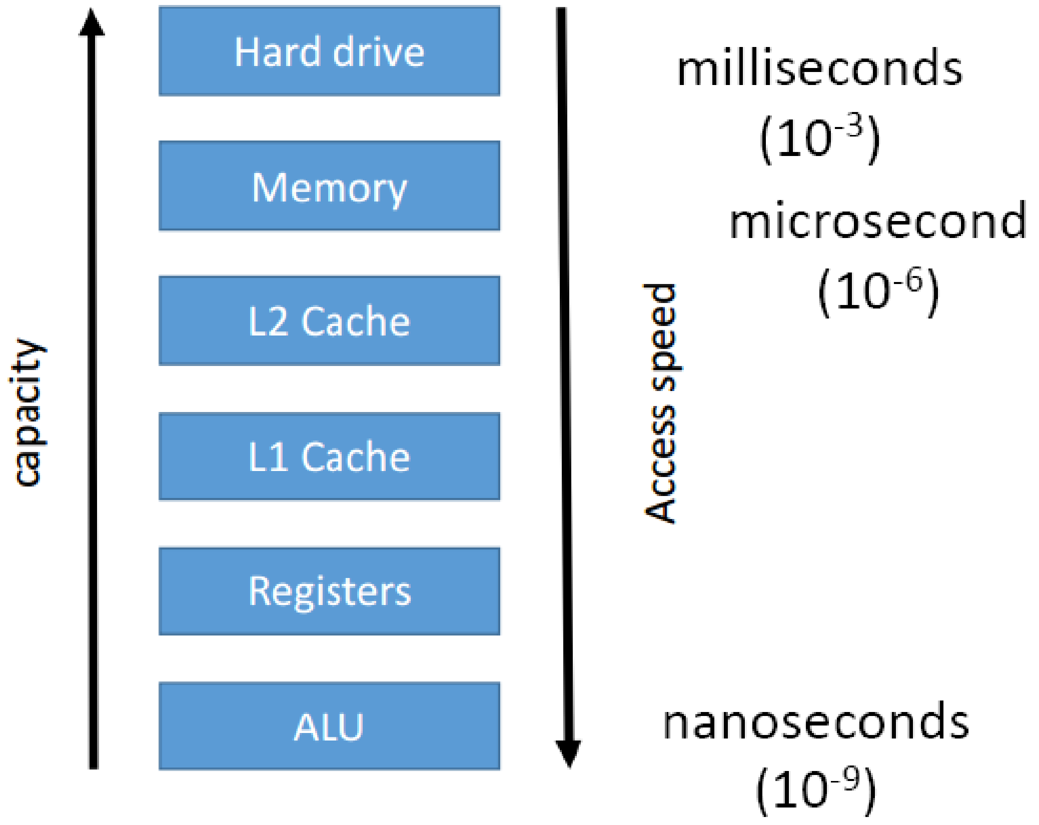
Q: What time delay (penalty) does a program incur due to demand paging in response to a single page fault?

- Service page fault
- swap in the needed frame
- Restart the process

Issue read (I/O), and schedule other processes to run

Bring in needed frame corresponding to needed page, update page table, wait for OS to schedule the process again

~50 microseconds + ~5 milliseconds



capacity

Hard drive

Memory

L2 Cache

L1 Cache

Registers

ALU

Access speed

milliseconds (10^{-3})

microsecond (10^{-6})

nanoseconds (10^{-9})

PAGE FAULT PENALTY

Ideal Case : all of the pages needed by a program are in physical memory

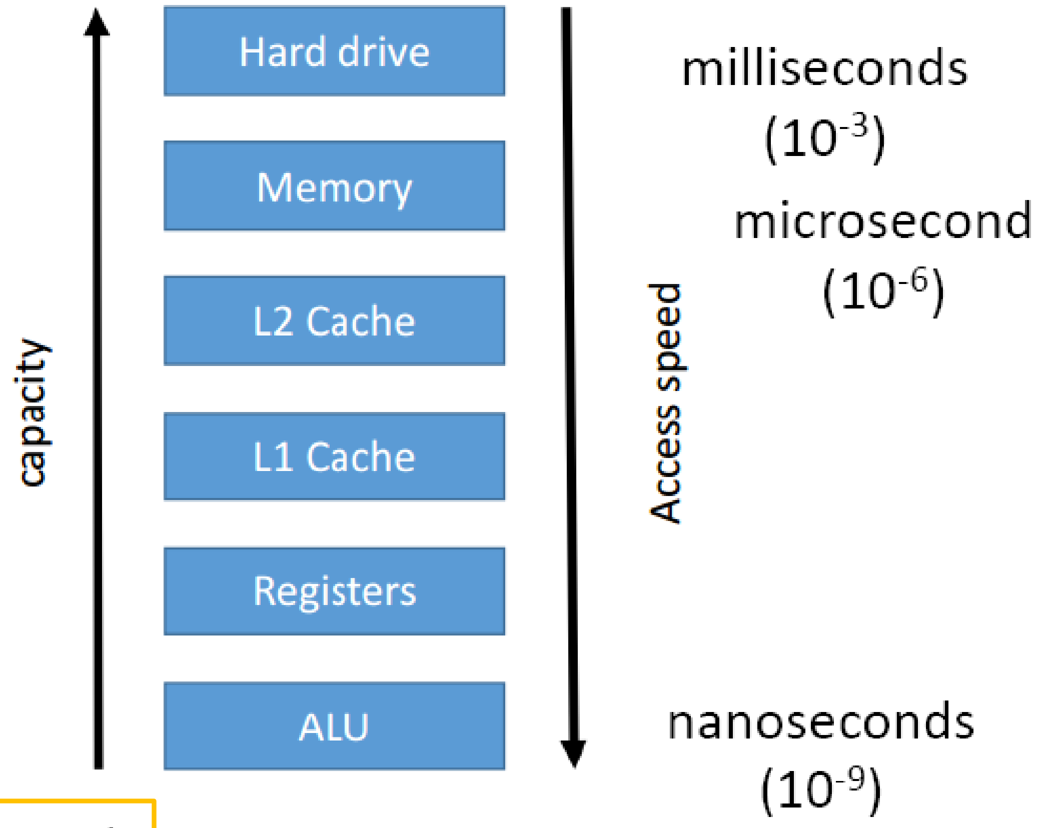
Real-world Case : Swapping in (and out) must occur

Q: What time delay (penalty) does a program incur due to demand paging in response to a single page fault?

- Service page fault
- swap in the needed frame
- Restart the process

Assuming no other processes have a higher priority in the ready queue …

restart the process as soon as "it" is dispatched



~50 microseconds + ~5 milliseconds + ~50 microseconds

PAGE FAULT PENALTY

Ideal Case : all of the pages needed by a program are in physical memory

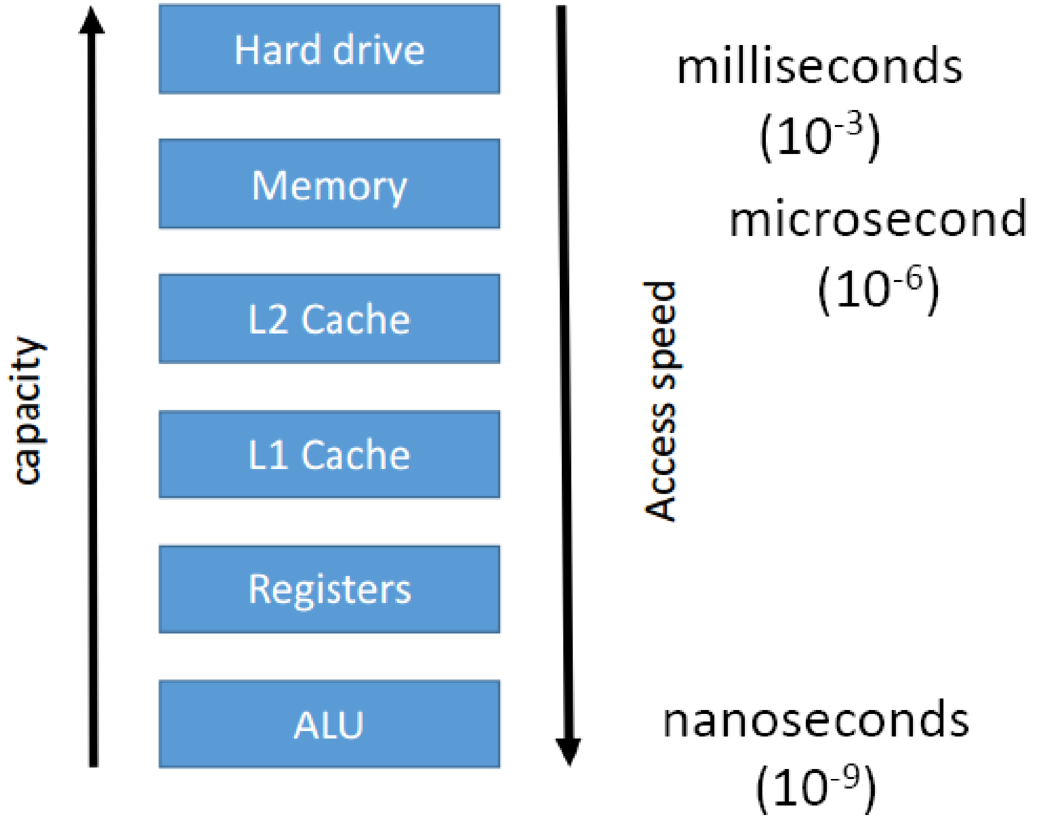
Real-world Case : Swapping in (and out) must occur

Q: What time delay (penalty) does a program incur due to demand paging in response to a single page fault?

- Service page fault
- swap in the needed frame
- Restart the process

Total time for all three, which does not include waiting for dispatch in the ready queue …

About 5 milliseconds



capacity

Hard drive

Memory

L2 Cache

L1 Cache

Registers

ALU

Access speed

milliseconds (10^{-3})

microsecond (10^{-6})

nanoseconds (10^{-9})

~50 microseconds + ~5 milliseconds + ~50 microseconds = 5.1 milliseconds

PAGE FAULT PENALTY

Ideal Case : all of the pages needed by a program are in physical memory

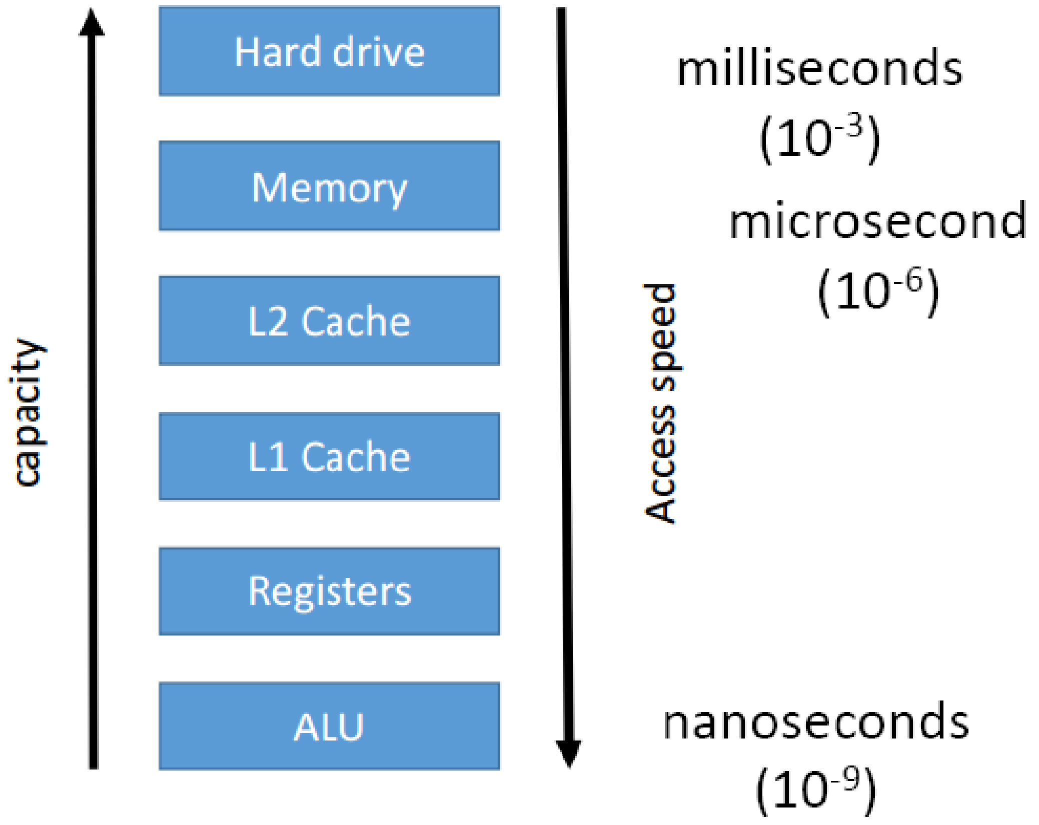
Real-world Case : Swapping in (and out) must occur

Q: What time delay (penalty) does a program incur due to demand paging in response to a single page fault?

- Service page fault
- swap in the needed frame
- Restart the process

Total time for all three, which does not include waiting for dispatch in the ready queue …

About 5 milliseconds



capacity

Hard drive

Memory

L2 Cache

L1 Cache

Registers

ALU

Access speed

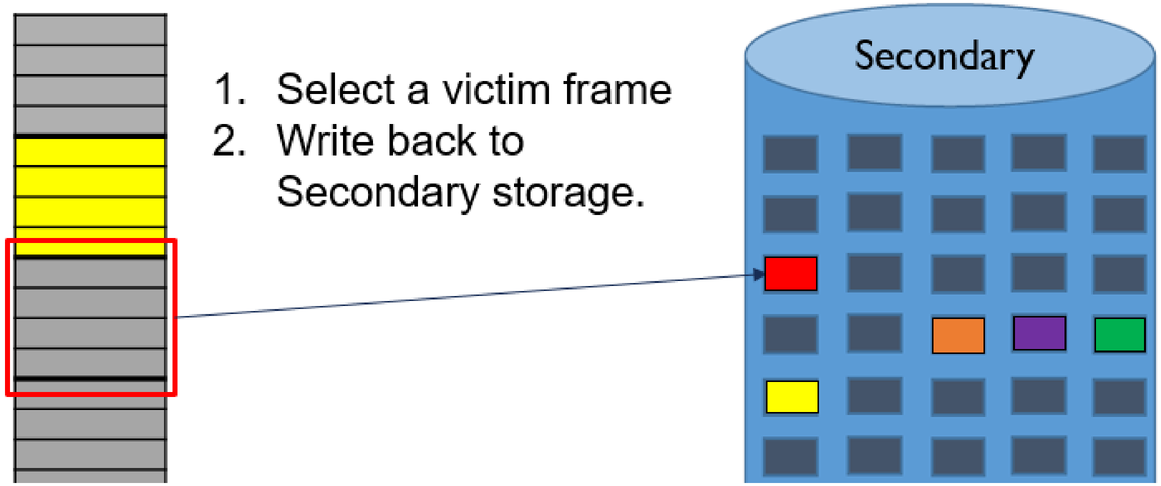
milliseconds (10^{-3})

microsecond (10^{-6})

nanoseconds (10^{-9})

~50 microseconds + ~5 milliseconds + ~50 microseconds = 5.1 milliseconds

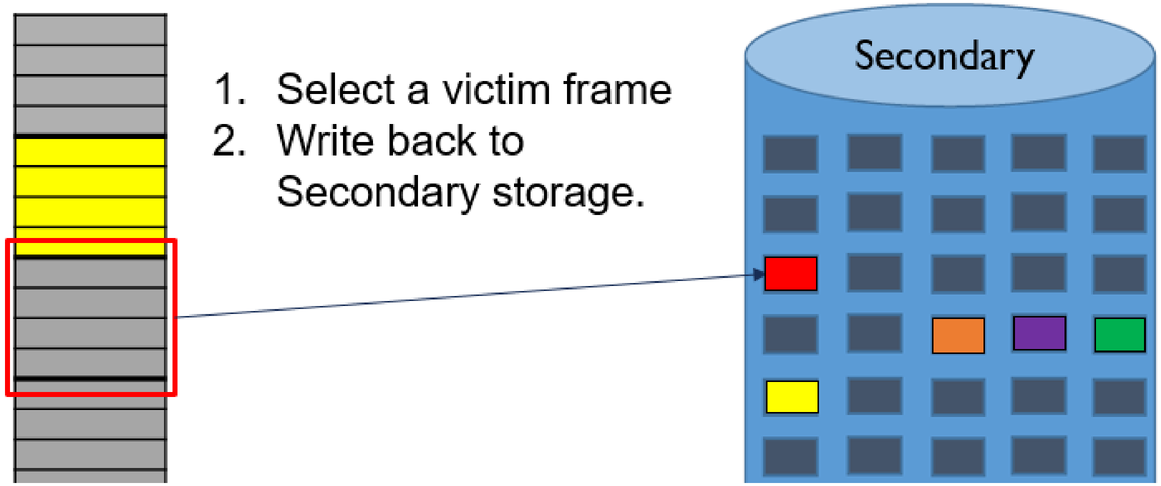
PAGE REPLACEMENT



1. Select a victim frame
2. Write back to Secondary storage.

PAGE REPLACEMENT

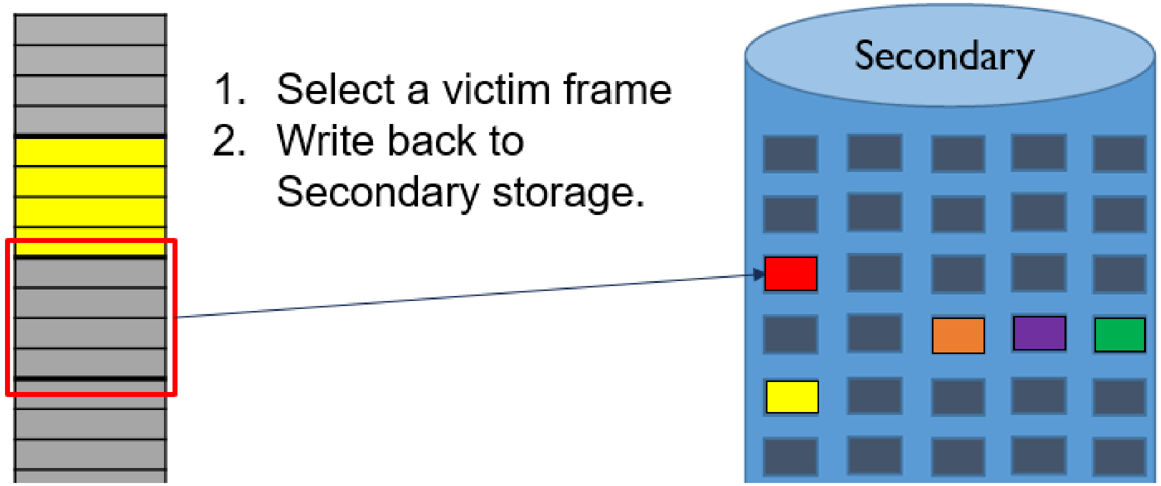
- How is the victim selected?



1. Select a victim frame
2. Write back to Secondary storage.

PAGE REPLACEMENT

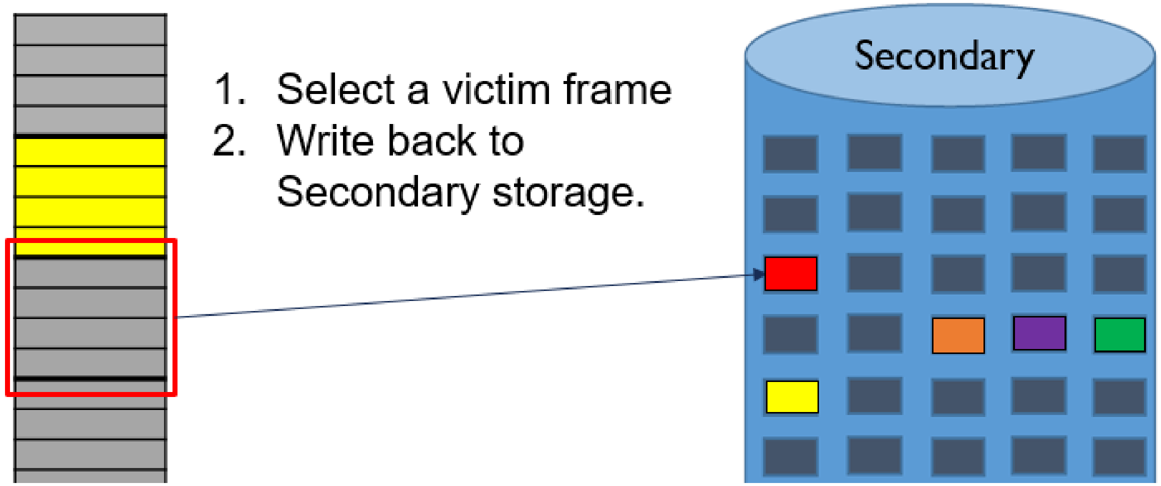
- The choice of the "victim" page to select is important.
- How is the victim selected?



1. Select a victim frame
2. Write back to Secondary storage.

PAGE REPLACEMENT

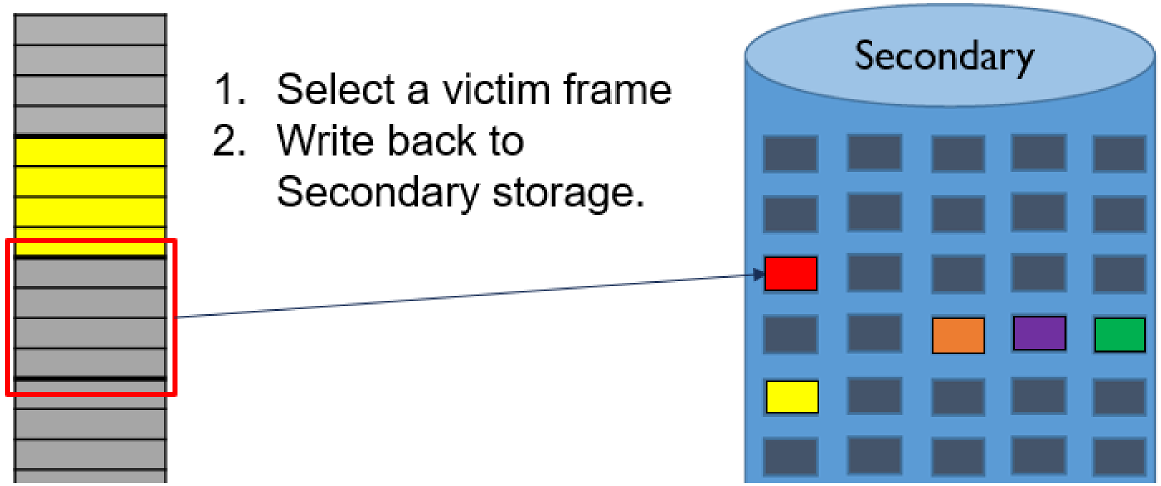
- The choice of the "victim" page to select is important.
- How is the victim selected?
- Worksheet Q2: what are possible criteria or factors for selecting the victim page?



1. Select a victim frame
2. Write back to Secondary storage.

PAGE REPLACEMENT

- The choice of the "victim" page to select is important.
- How is the victim selected?
- Worksheet Q2: what are possible criteria or factors for selecting the victim page?

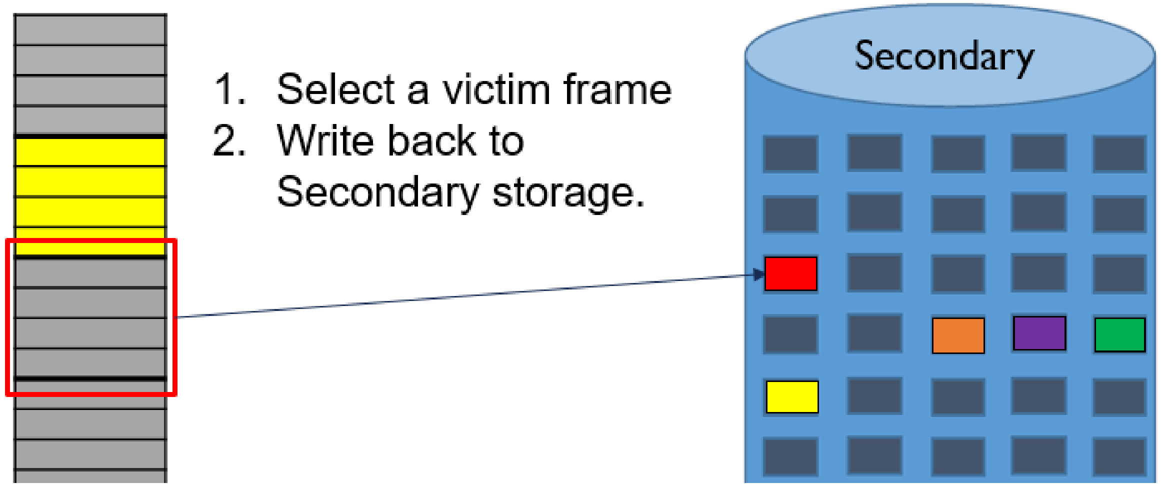


1. Select a victim frame
2. Write back to Secondary storage.

objective: reduce page faults

PAGE REPLACEMENT

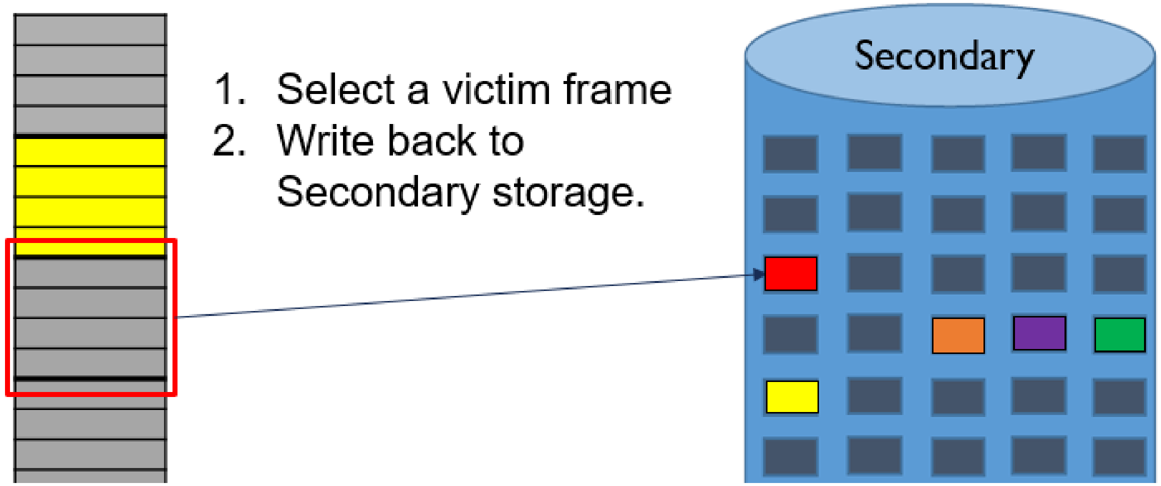
- How is the victim selected?
- Victim Selection: FIFO
- Select the oldest page in the frame and remove it.
- How does it work? How does it perform?



1. Select a victim frame
2. Write back to Secondary storage.

FIFO PAGE REPLACEMENT

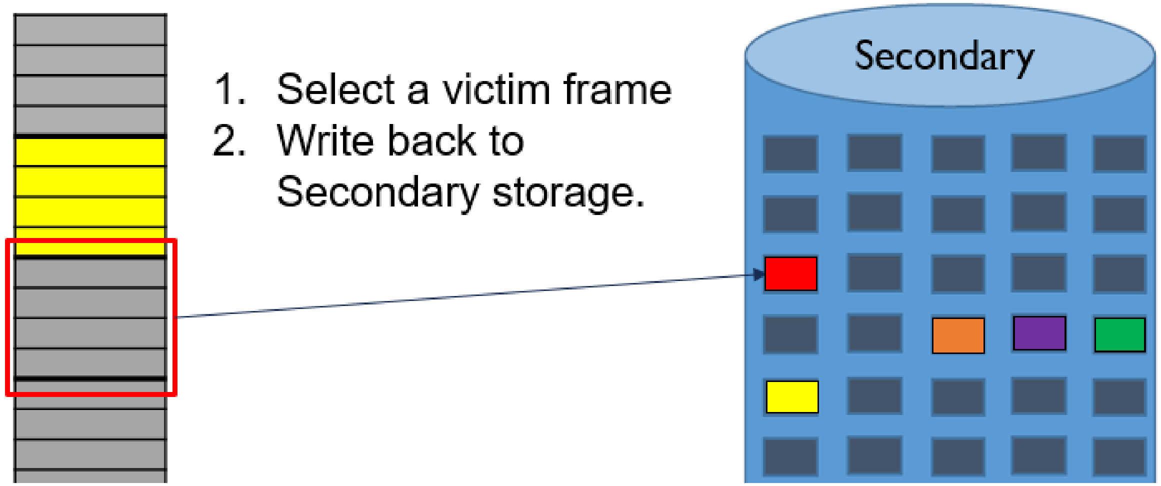
- Victim Selection: FIFO



1. Select a victim frame
2. Write back to Secondary storage.

FIFO PAGE REPLACEMENT

- Victim Selection: FIFO
- Select the oldest page in the frame and remove it.
- How does it work? How does it perform?



1. Select a victim frame
2. Write back to Secondary storage.

FIFO REPLACEMENT

- Assume a page/frame architecture where each page has **100 bytes.** Assume the following byte (address) requests, left to right (written in base 10) generated by a process.

1011 0656 0692 1466 0605 1141 1222

FIFO REPLACEMENT

- Assume a page/frame architecture where each page has **100 bytes.** Assume the following byte (address) requests, left to right (written in base 10) generated by a process.

1011 0656 0692 1466 0605 1141 1222

Starting with an empty 3-frame memory:

- How will the pages be places?
- What is the number of page faults?

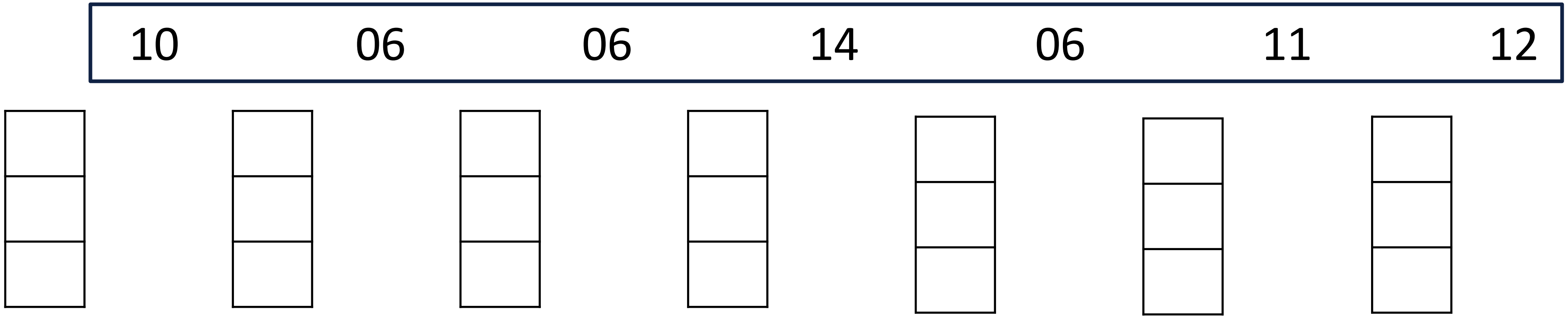
FIFO EXAMPLE

Consider this sequence of pages
Assume a physical memory size of 3 frames (for the process).

10	06	06	14	06	11	12

FIFO EXAMPLE

Consider this sequence of pages
Assume a physical memory size of 3 frames (for the process).

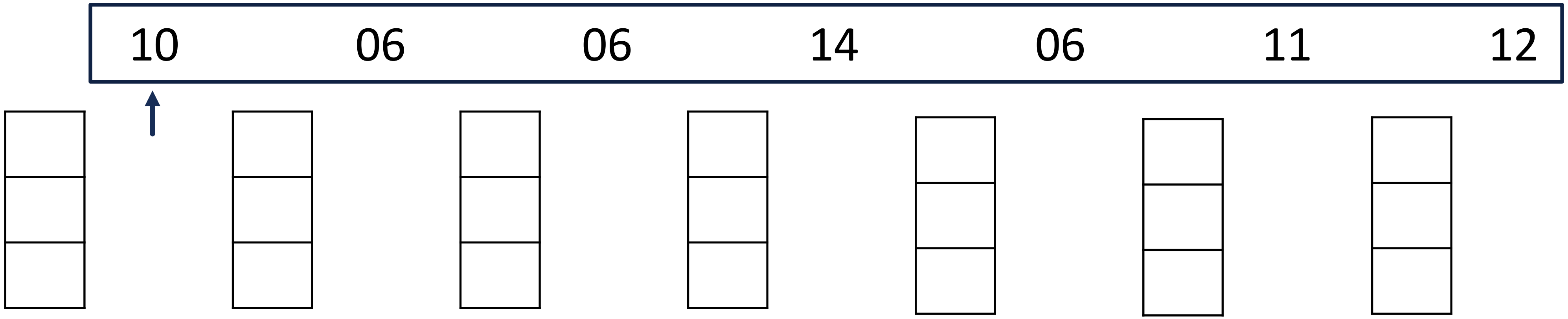


10	06	06	14	06	11	12

Page Fault Count: 0

FIFO EXAMPLE

Consider this sequence of pages
Assume a physical memory size of 3 frames (for the process).

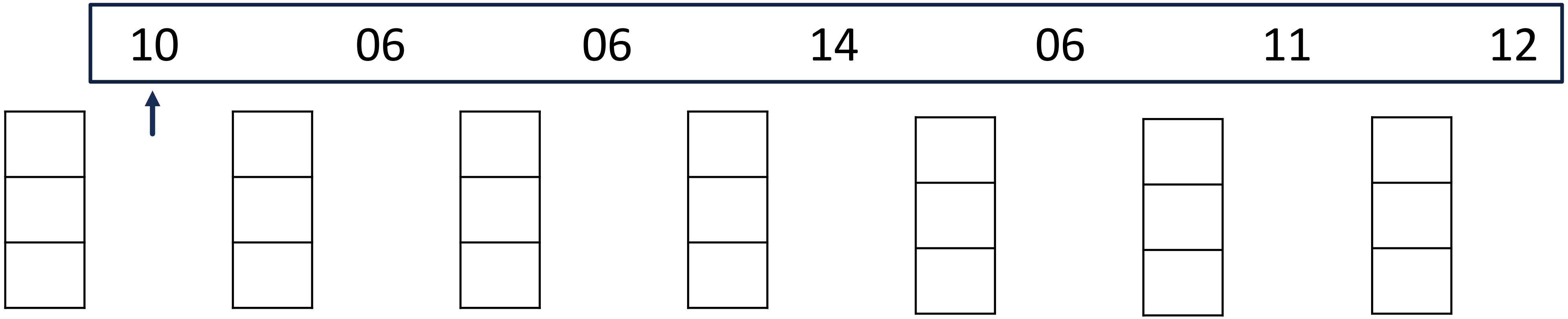


10	06	06	14	06	11	12

Page Fault Count: 0

FIFO EXAMPLE

Consider this sequence of pages
Assume a physical memory size of 3 frames (for the process).



10	06	06	14	06	11	12

Page Fault Count: 0

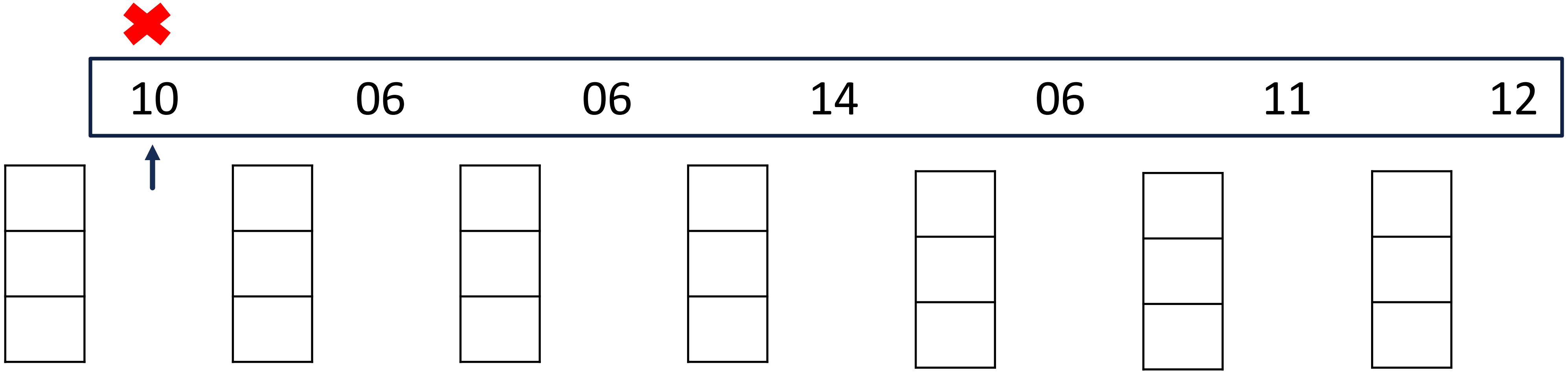
Q: Do we get a page fault?

A	B
C	D

A: Yes
B: No

FIFO EXAMPLE

Consider this sequence of pages
Assume a physical memory size of 3 frames (for the process).



10 06 06 14 06 11 12

Page Fault Count: 1

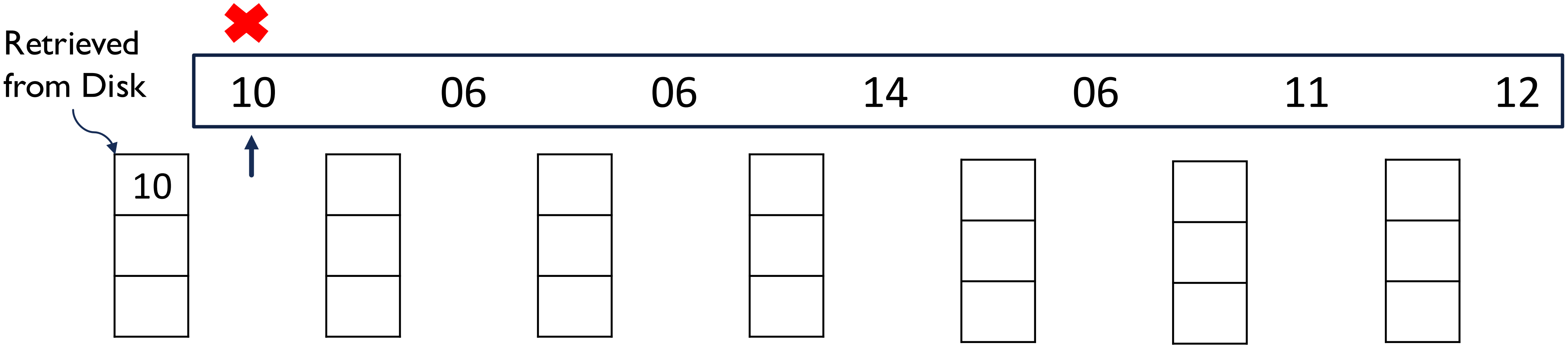
Q: Do we get a page fault?

A B
C D

A: Yes
B: No

FIFO EXAMPLE

Consider this sequence of pages
Assume a physical memory size of 3 frames (for the process).



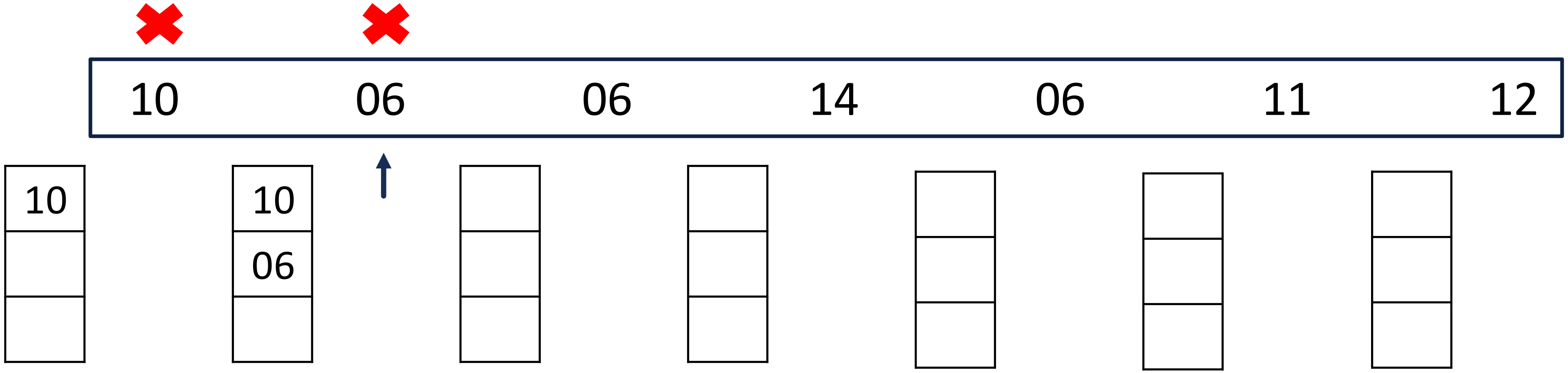
Retrieved from Disk

10 06 06 14 06 11 12

Page Fault Count: 0

FIFO EXAMPLE

Consider this sequence of pages
Assume a physical memory size of 3 frames (for the process).

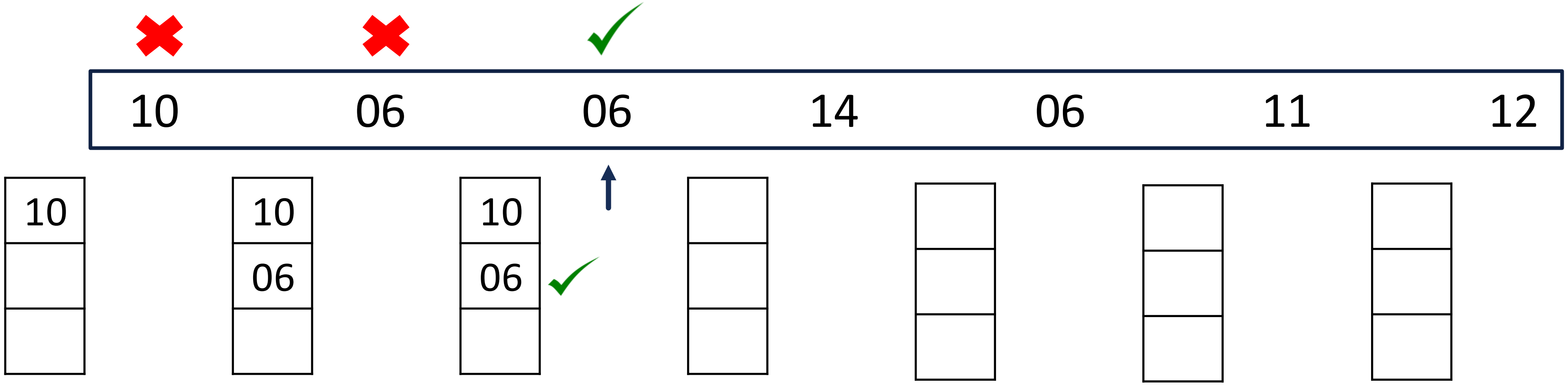


10 06 06 14 06 11 12

Page Fault Count: 2

FIFO EXAMPLE

Consider this sequence of pages
Assume a physical memory size of 3 frames (for the process).



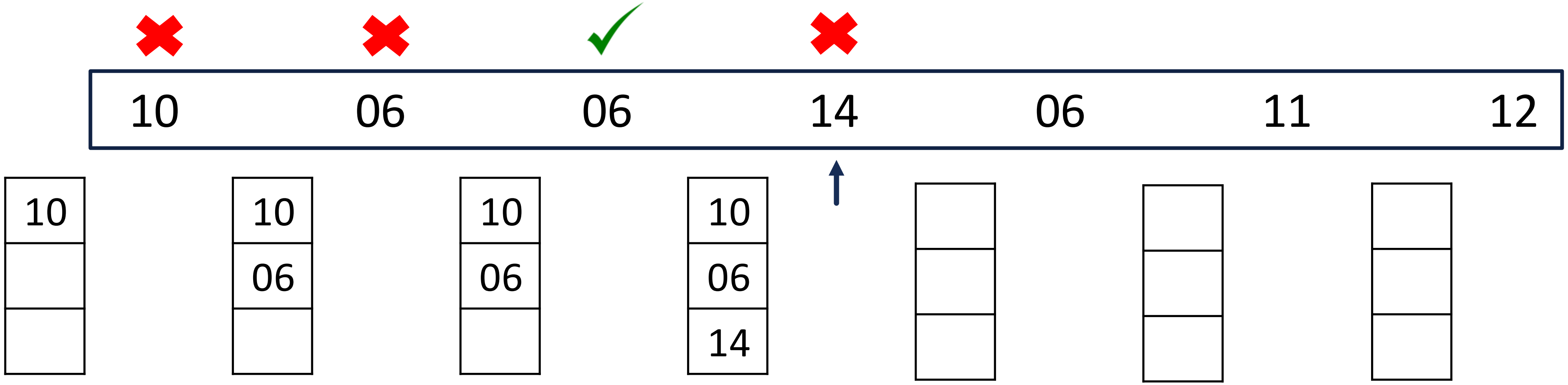
✖	✖	✓				
10	06	06	14	06	11	12

10	10	10				
	06	06 ✓				

Page Fault Count: 2

FIFO EXAMPLE

Consider this sequence of pages
Assume a physical memory size of 3 frames (for the process).

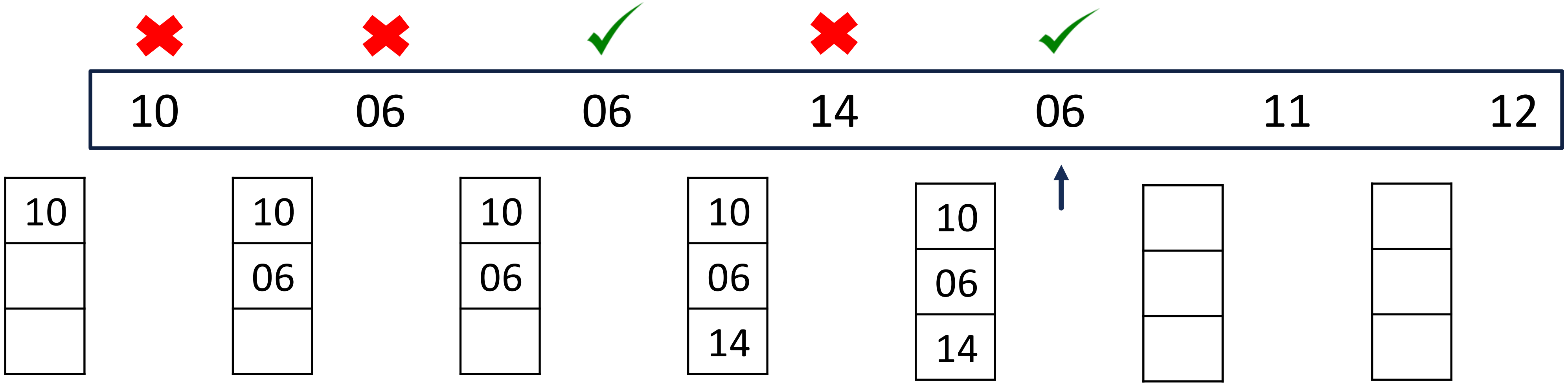


✗	✗	✓	✗			
10	06	06	14	06	11	12
10	10	10	10			
	06	06	06			
			14			

Page Fault Count: 3

FIFO EXAMPLE

Consider this sequence of pages
Assume a physical memory size of 3 frames (for the process).

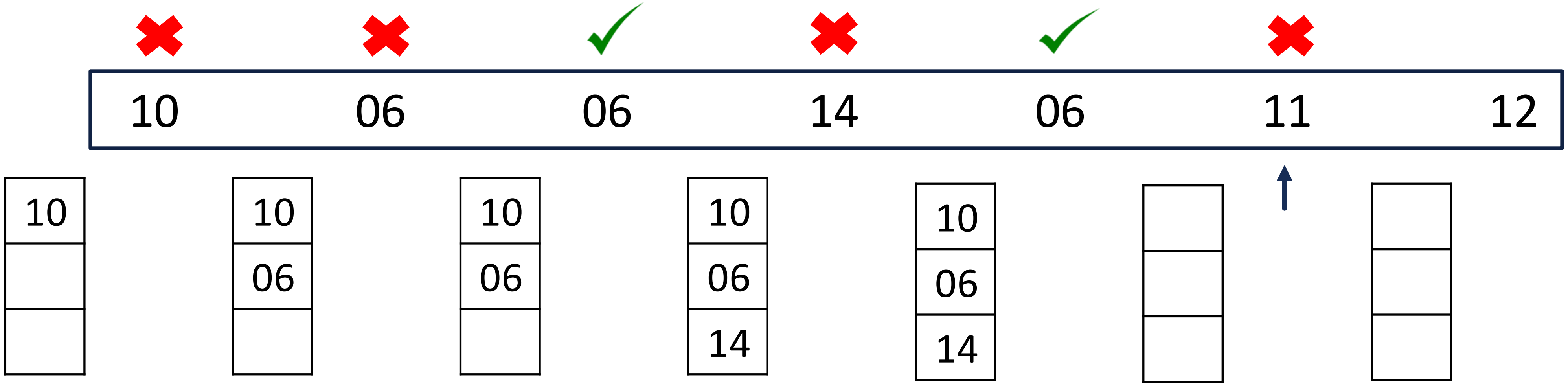


✗	✗	✓	✗	✓		
10	06	06	14	06	11	12
10	10	10	10	10		
	06	06	06	06		
			14	14		

Page Fault Count: 3

FIFO EXAMPLE

Consider this sequence of pages
Assume a physical memory size of 3 frames (for the process).



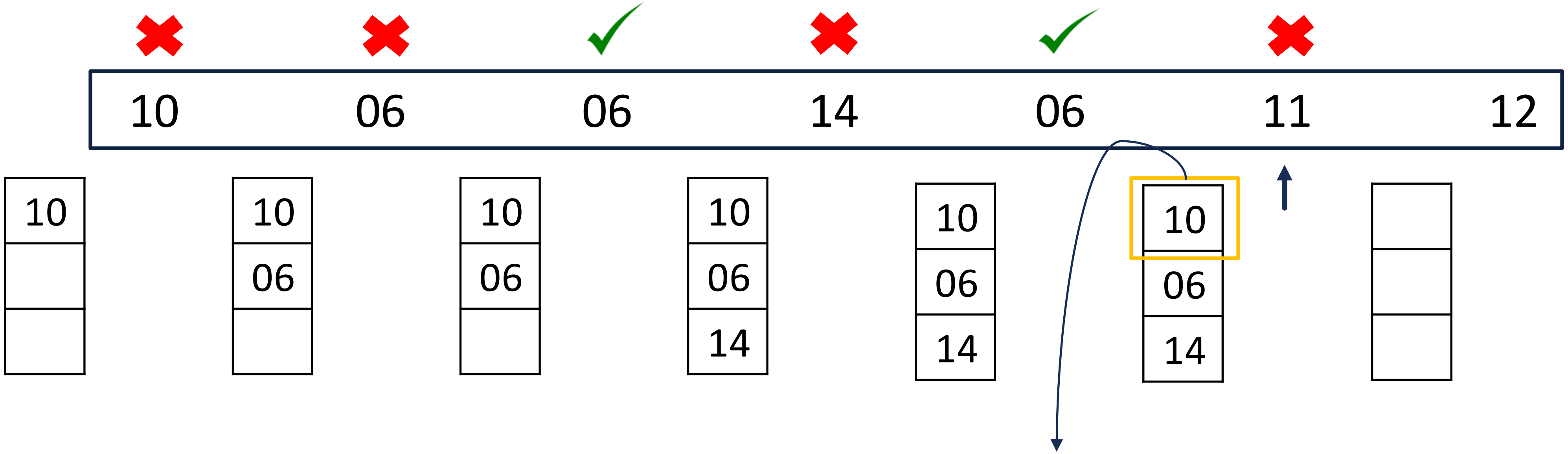
✗	✗	✓	✗	✓	✗	
10	06	06	14	06	11	12

10	10	10	10	10		
	06	06	06	06		
			14	14		

Page Fault Count: 4

FIFO EXAMPLE

Consider this sequence of pages
Assume a physical memory size of 3 frames (for the process).



✗	✗	✓	✗	✓	✗	
10	06	06	14	06	11	12

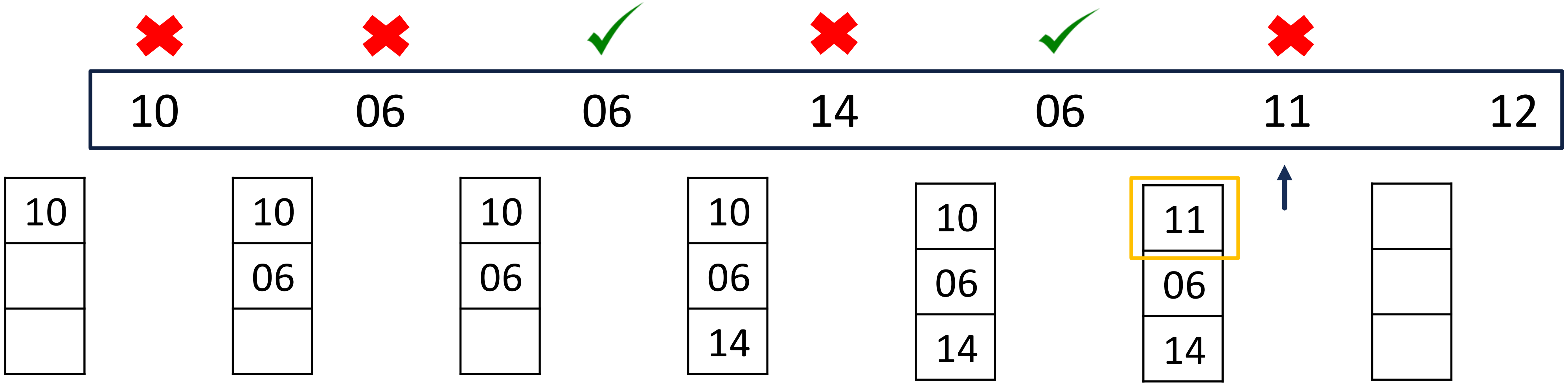
10	10	10	10	10	10	
	06	06	06	06	06	
			14	14	14	

First Frame In
This is our victim.

Page Fault Count: 4

FIFO EXAMPLE

Consider this sequence of pages
Assume a physical memory size of 3 frames (for the process).

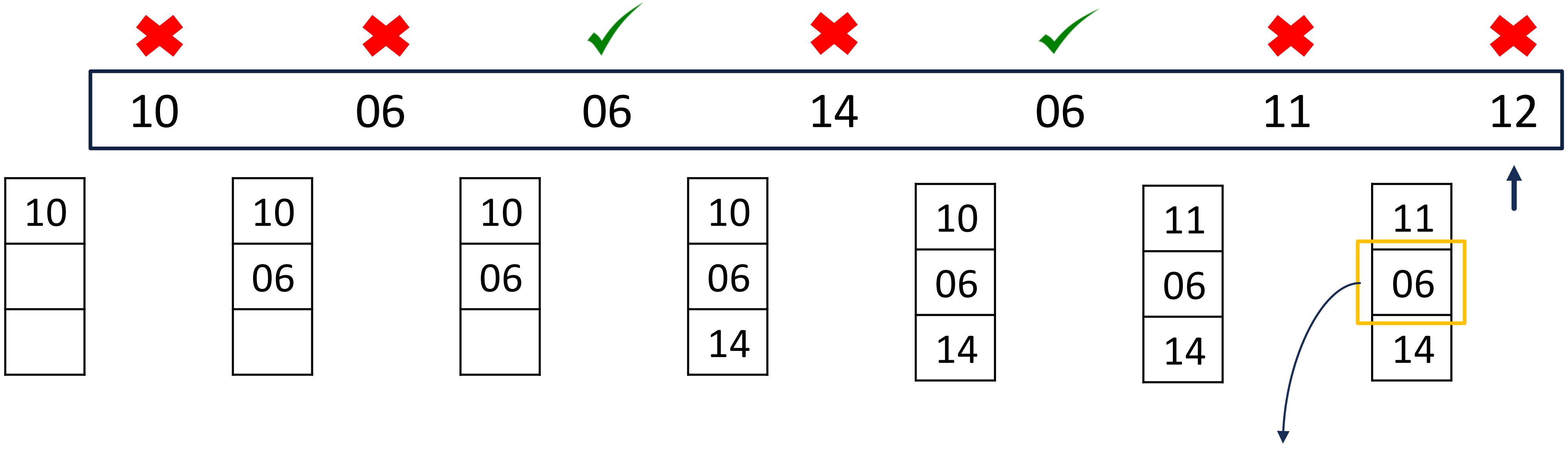


✗	✗	✓	✗	✓	✗	
10	06	06	14	06	11	12
10	10	10	10	10	11	
	06	06	06	06	06	
			14	14	14	

Page Fault Count: 4

FIFO EXAMPLE

Consider this sequence of pages
Assume a physical memory size of 3 frames (for the process).

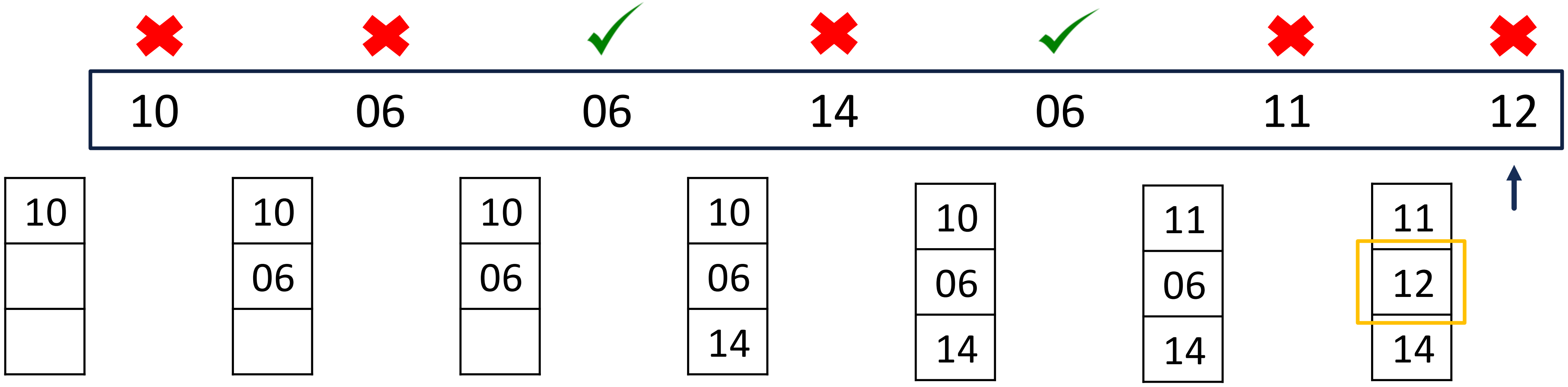


Second in line

Page Fault Count: 5

FIFO EXAMPLE

Consider this sequence of pages
Assume a physical memory size of 3 frames (for the process).



✗	✗	✓	✗	✓	✗	✗
10	06	06	14	06	11	12
10	10	10	10	10	11	11
	06	06	06	06	06	12
			14	14	14	14

Page Fault Count: 5

WORKSHEET

1-Frame Memory:

0110	1206	0613	0697	1206	1606	0614

2-Frame Memory:

0110	1206	0613	0697	1206	1606	0614

3-Frame Memory:

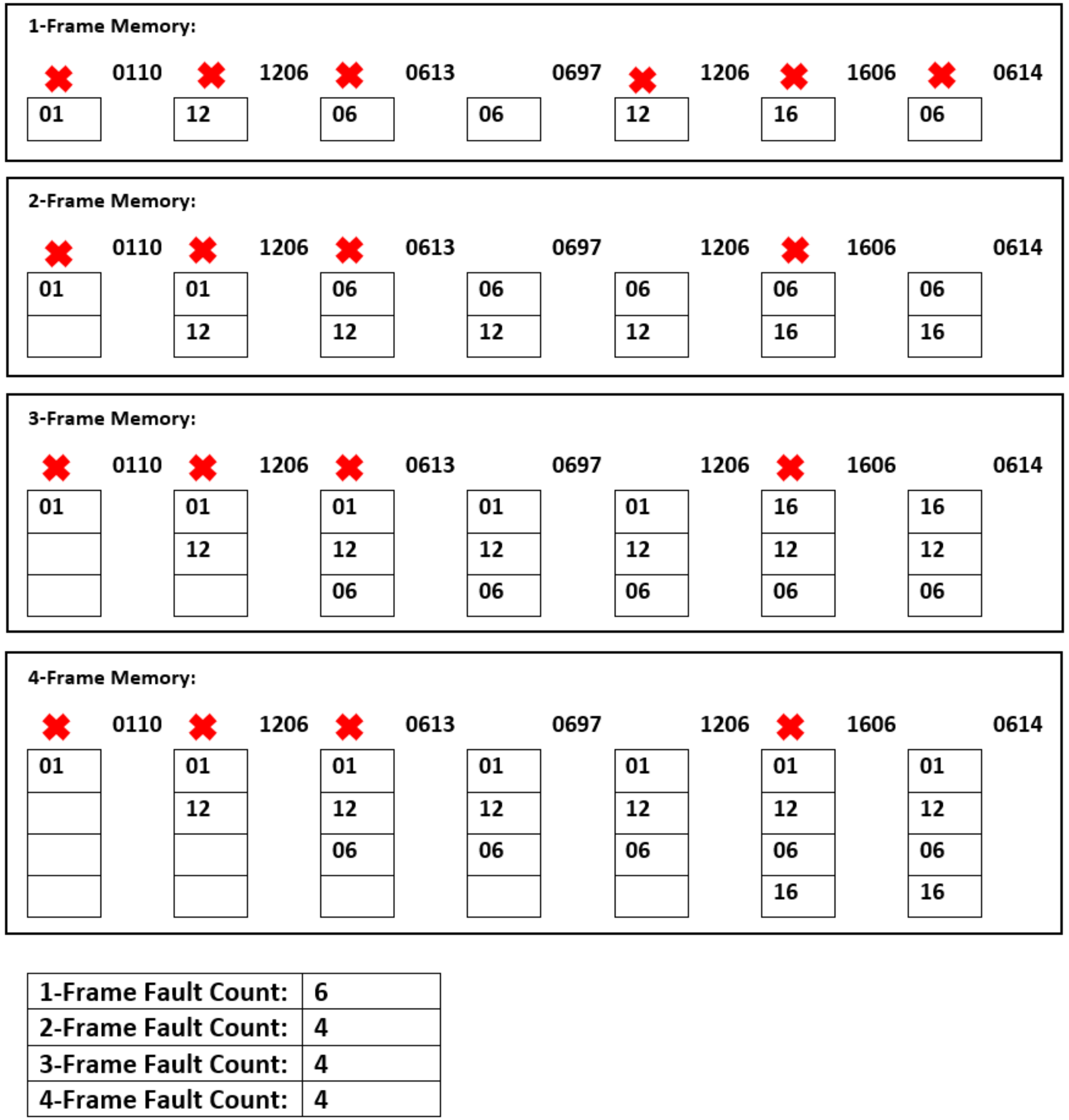
0110	1206	0613	0697	1206	1606	0614

4-Frame Memory:

0110	1206	0613	0697	1206	1606	0614

1-Frame Fault Count:	
2-Frame Fault Count:	
3-Frame Fault Count:	
4-Frame Fault Count:	

WORKSHEET SOLUTION



1-Frame Memory:

0110	1206	0613	0697	1206	1606	0614
✖	✖	✖		✖	✖	✖
01	12	06	06	12	16	06

2-Frame Memory:

0110	1206	0613	0697	1206	1606	0614
✖	✖	✖			✖	
01	01	06	06	06	06	06
	12	12	12	12	16	16

3-Frame Memory:

0110	1206	0613	0697	1206	1606	0614
✖	✖	✖			✖	
01	01	01	01	01	16	16
	12	12	12	12	12	12
		06	06	06	06	06

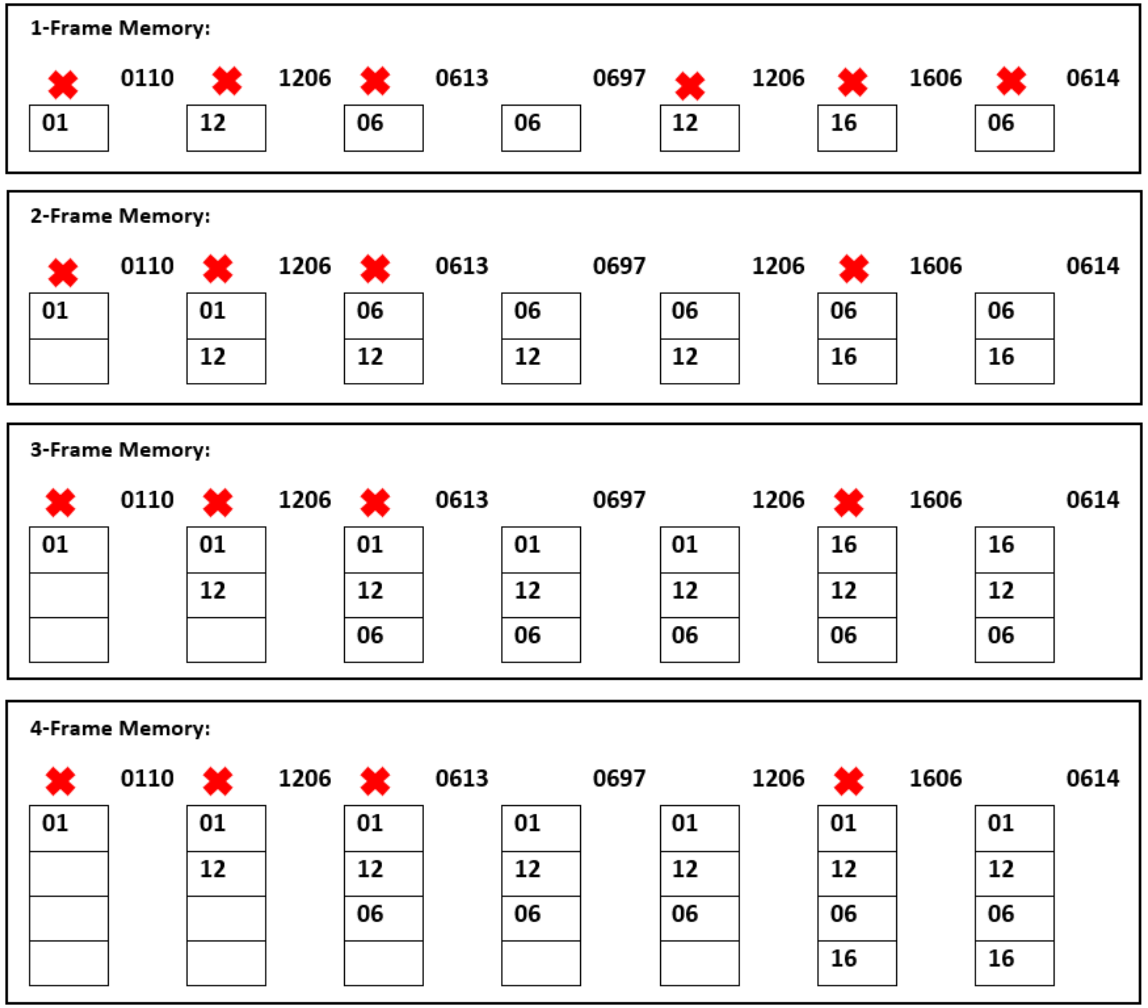
4-Frame Memory:

0110	1206	0613	0697	1206	1606	0614
✖	✖	✖			✖	
01	01	01	01	01	01	01
	12	12	12	12	12	12
		06	06	06	06	06
					16	16

1-Frame Fault Count:	6
2-Frame Fault Count:	4
3-Frame Fault Count:	4
4-Frame Fault Count:	4

WORKSHEET SOLUTION

- It’s a page fault when any frame in memory changes content.



1-Frame Memory:

0110	1206	0613	0697	1206	1606	0614
✖	✖	✖		✖	✖	✖
01	12	06	06	12	16	06

2-Frame Memory:

0110	1206	0613	0697	1206	1606	0614
✖	✖	✖			✖	
01	01	06	06	06	06	06
	12	12	12	12	16	16

3-Frame Memory:

0110	1206	0613	0697	1206	1606	0614
✖	✖	✖			✖	
01	01	01	01	01	16	16
	12	12	12	12	12	12
		06	06	06	06	06

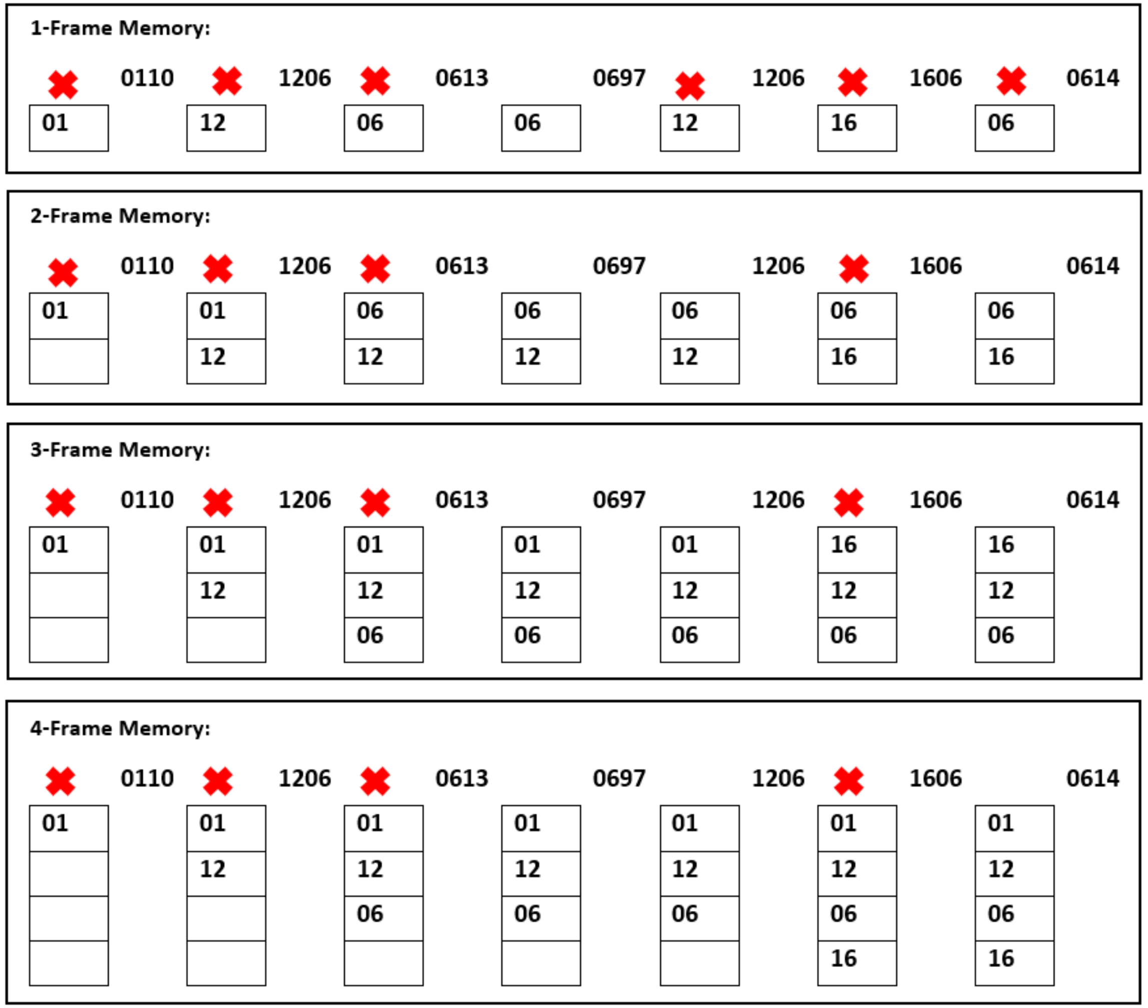
4-Frame Memory:

0110	1206	0613	0697	1206	1606	0614
✖	✖	✖			✖	
01	01	01	01	01	01	01
	12	12	12	12	12	12
		06	06	06	06	06
					16	16

1-Frame Fault Count:	6
2-Frame Fault Count:	4
3-Frame Fault Count:	4
4-Frame Fault Count:	4

WORKSHEET SOLUTION

- It’s a page fault when any frame in memory changes content.
- Page faults can’t be eliminated completely even with infinite memory … every page needs to be retrieved at least once causing a single page fault for each page.



1-Frame Memory:

✖ 0110	✖ 1206	✖ 0613	0697	✖ 1206	✖ 1606	✖ 0614
01	12	06	06	12	16	06

2-Frame Memory:

✖ 0110	✖ 1206	✖ 0613	0697	1206	✖ 1606	0614
01	01	06	06	06	06	06
	12	12	12	12	16	16

3-Frame Memory:

✖ 0110	✖ 1206	✖ 0613	0697	1206	✖ 1606	0614
01	01	01	01	01	16	16
	12	12	12	12	12	12
		06	06	06	06	06

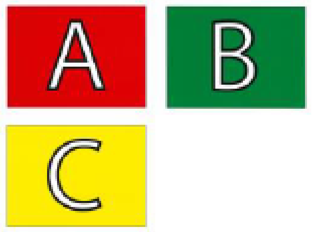
4-Frame Memory:

✖ 0110	✖ 1206	✖ 0613	0697	1206	✖ 1606	0614
01	01	01	01	01	01	01
	12	12	12	12	12	12
		06	06	06	06	06
					16	16

1-Frame Fault Count:	6
2-Frame Fault Count:	4
3-Frame Fault Count:	4
4-Frame Fault Count:	4

INCREASING THE FRAME SIZE

Can increasing the number of memory frames result in the increase of Page Faults in a FIFO Replacement Strategy?



A : Yes

B : No

C : Unsure

3-FRAME FIFO

FIFO page replacement

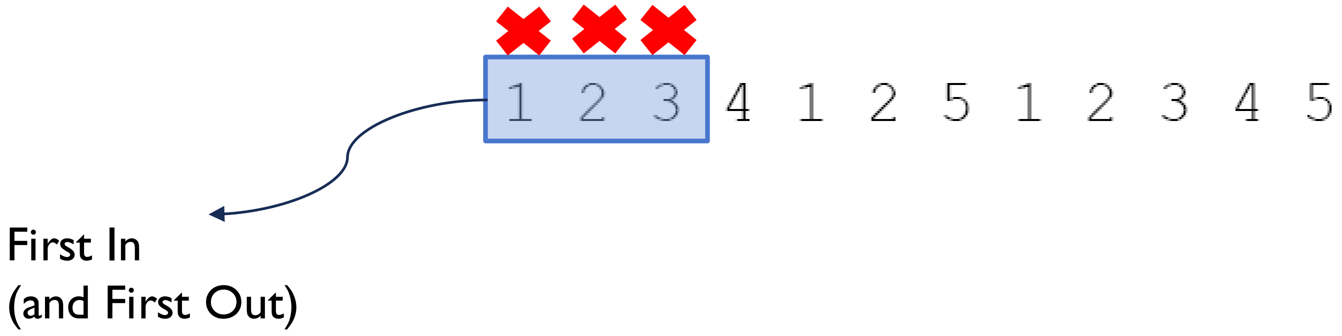
Assume the following are the page components of addresses (left to right, written in base 10) generated by a process

1 2 3 4 1 2 5 1 2 3 4 5

3-FRAME FIFO

FIFO page replacement

Assume the following are the page components of addresses (left to right, written in base 10) generated by a process



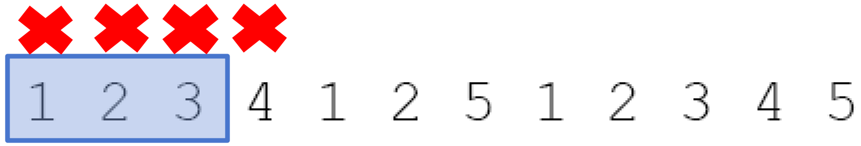
1 2 3 4 1 2 5 1 2 3 4 5

First In
(and First Out)

3-FRAME FIFO

FIFO page replacement

Assume the following are the page components of addresses (left to right, written in base 10) generated by a process

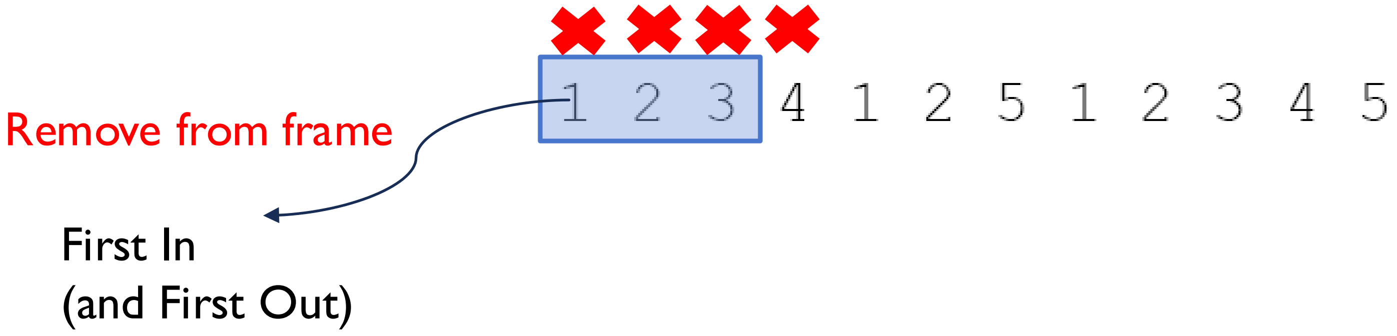


1 2 3 4 1 2 5 1 2 3 4 5

3-FRAME FIFO

FIFO page replacement

Assume the following are the page components of addresses (left to right, written in base 10) generated by a process



1 2 3 4 1 2 5 1 2 3 4 5

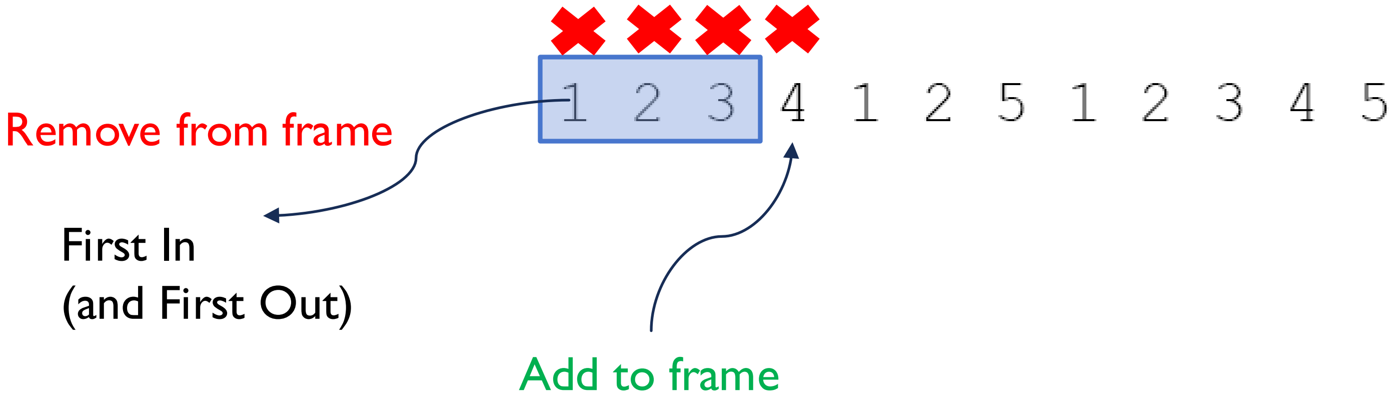
Remove from frame

First In
(and First Out)

3-FRAME FIFO

FIFO page replacement

Assume the following are the page components of addresses (left to right, written in base 10) generated by a process



1 2 3 4 1 2 5 1 2 3 4 5

Remove from frame

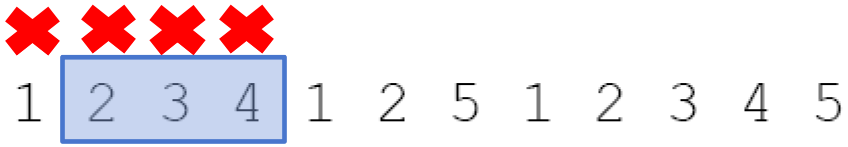
First In
(and First Out)

Add to frame

3-FRAME FIFO

FIFO page replacement

Assume the following are the page components of addresses (left to right, written in base 10) generated by a process

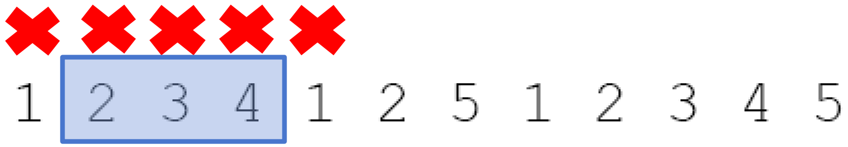


1 2 3 4 1 2 5 1 2 3 4 5

3-FRAME FIFO

FIFO page replacement

Assume the following are the page components of addresses (left to right, written in base 10) generated by a process



1 2 3 4 1 2 5 1 2 3 4 5

3-FRAME FIFO

FIFO page replacement

Assume the following are the page components of addresses (left to right, written in base 10) generated by a process

✖ ✖ ✖ ✖ ✖

1 2 3 4 1 2 5 1 2 3 4 5

3-FRAME FIFO

FIFO page replacement

Assume the following are the page components of addresses (left to right, written in base 10) generated by a process

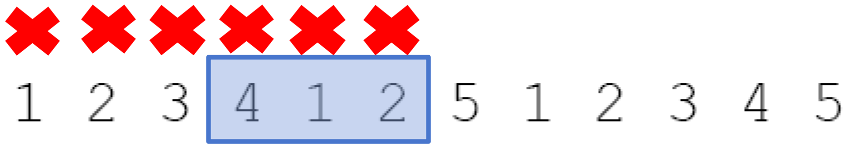
✖ ✖ ✖ ✖ ✖ ✖

1 2 $\boxed{3\ \ 4\ \ 1}$ 2 5 1 2 3 4 5

3-FRAME FIFO

FIFO page replacement

Assume the following are the page components of addresses (left to right, written in base 10) generated by a process



1 2 3 4 1 2 5 1 2 3 4 5

3-FRAME FIFO

FIFO page replacement

Assume the following are the page components of addresses (left to right, written in base 10) generated by a process

✖ ✖ ✖ ✖ ✖ ✖ ✖

1 2 3 4 1 2 5 1 2 3 4 5

3-FRAME FIFO

FIFO page replacement

Assume the following are the page components of addresses (left to right, written in base 10) generated by a process

✖	✖	✖	✖	✖	✖	✖					
1	2	3	4	1	2	5	1	2	3	4	5

3-FRAME FIFO

FIFO page replacement

Assume the following are the page components of addresses (left to right, written in base 10) generated by a process

✖	✖	✖	✖	✖	✖	✖	✓				
1	2	3	4	1	2	5	1	2	3	4	5

3-FRAME FIFO

FIFO page replacement

Assume the following are the page components of addresses (left to right, written in base 10) generated by a process

✖	✖	✖	✖	✖	✖	✖	✓	✓			
1	2	3	4	1	2	5	1	2	3	4	5

3-FRAME FIFO

FIFO page replacement

Assume the following are the page components of addresses (left to right, written in base 10) generated by a process

✗	✗	✗	✗	✗	✗	✗	✓	✓	✗		
1	2	3	4	1	2	5	1	2	3	4	5

3-FRAME FIFO

FIFO page replacement

Assume the following are the page components of addresses (left to right, written in base 10) generated by a process

✖	✖	✖	✖	✖	✖	✖	✓	✓	✖		
1	2	3	4	1	2	5	1	2	3	4	5

3-FRAME FIFO

FIFO page replacement

Assume the following are the page components of addresses (left to right, written in base 10) generated by a process

✖	✖	✖	✖	✖	✖	✖	✓	✓	✖	✖	
1	2	3	4	1	2	5	1	2	3	4	5

3-FRAME FIFO

FIFO page replacement

Assume the following are the page components of addresses (left to right, written in base 10) generated by a process

✗	✗	✗	✗	✗	✗	✗	✓	✓	✗	✗	
1	2	3	4	1	2	5	1	2	3	4	5

3-FRAME FIFO

FIFO page replacement

Assume the following are the page components of addresses (left to right, written in base 10) generated by a process

✗	✗	✗	✗	✗	✗	✗	✓	✓	✗	✗	✓
1	2	3	4	1	2	5	1	2	3	4	5

Page Fault Counter: 9

4-FRAME FIFO

FIFO page replacement

Assume the following are the page components of addresses (left to right, written in base 10) generated by a process

✖ ✖ ✖ ✖

1 2 3 4 1 2 5 1 2 3 4 5

4-FRAME FIFO

FIFO page replacement

Assume the following are the page components of addresses (left to right, written in base 10) generated by a process

✖ ✖ ✖ ✖ ✓

1 2 3 4 1 2 5 1 2 3 4 5

4-FRAME FIFO

FIFO page replacement

Assume the following are the page components of addresses (left to right, written in base 10) generated by a process

✗ ✗ ✗ ✗ ✓ ✓

1 2 3 4 1 2 5 1 2 3 4 5

4-FRAME FIFO

FIFO page replacement

Assume the following are the page components of addresses (left to right, written in base 10) generated by a process

```
✖ ✖ ✖ ✖ ✓ ✓ ✖
1 2 3 4 1 2 5 1 2 3 4 5
```

4-FRAME FIFO

FIFO page replacement

Assume the following are the page components of addresses (left to right, written in base 10) generated by a process

✗	✗	✗	✗	✓	✓	✗					
1	2	3	4	1	2	5	1	2	3	4	5

4-FRAME FIFO

FIFO page replacement

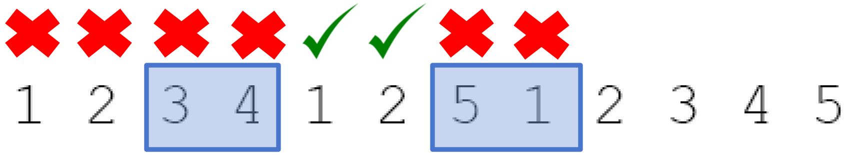
Assume the following are the page components of addresses (left to right, written in base 10) generated by a process

✗	✗	✗	✗	✓	✓	✗	✗				
1	2	3	4	1	2	5	1	2	3	4	5

4-FRAME FIFO

FIFO page replacement

Assume the following are the page components of addresses (left to right, written in base 10) generated by a process



✗ ✗ ✗ ✗ ✓ ✓ ✗ ✗

1 2 3 4 1 2 5 1 2 3 4 5

4-FRAME FIFO

FIFO page replacement

Assume the following are the page components of addresses (left to right, written in base 10) generated by a process

✖	✖	✖	✖	✓	✓	✖	✖	✖			
1	2	3	4	1	2	5	1	2	3	4	5

4-FRAME FIFO

FIFO page replacement

Assume the following are the page components of addresses (left to right, written in base 10) generated by a process

✖	✖	✖	✖	✓	✓	✖	✖	✖			
1	2	3	4	1	2	5	1	2	3	4	5

4-FRAME FIFO

FIFO page replacement

Assume the following are the page components of addresses (left to right, written in base 10) generated by a process

✗	✗	✗	✗	✓	✓	✗	✗	✗	✗		
1	2	3	4	1	2	5	1	2	3	4	5

4-FRAME FIFO

FIFO page replacement

Assume the following are the page components of addresses (left to right, written in base 10) generated by a process

✗	✗	✗	✗	✓	✓	✗	✗	✗	✗		
1	2	3	4	1	2	5	1	2	3	4	5

4-FRAME FIFO

FIFO page replacement

Assume the following are the page components of addresses (left to right, written in base 10) generated by a process

✗	✗	✗	✗	✓	✓	✗	✗	✗	✗	✗	
1	2	3	4	1	2	5	1	2	3	4	5

4-FRAME FIFO

FIFO page replacement

Assume the following are the page components of addresses (left to right, written in base 10) generated by a process

✗	✗	✗	✗	✓	✓	✗	✗	✗	✗	✗	
1	2	3	4	1	2	5	1	2	3	4	5

4-FRAME FIFO

FIFO page replacement

Assume the following are the page components of addresses (left to right, written in base 10) generated by a process

✖	✖	✖	✖	✓	✓	✖	✖	✖	✖	✖	✖
1	2	3	4	1	2	5	1	2	3	4	5

4-FRAME FIFO

FIFO page replacement

Assume the following are the page components of addresses (left to right, written in base 10) generated by a process

✗	✗	✗	✗	✓	✓	✗	✗	✗	✗	✗	✗
1	2	3	4	1	2	5	1	2	3	4	5

Page Fault Counter: 10

- **Increasing the frame size increased the page fault rate!**
- **Belady's Anomaly**

OPTIMAL REPLACEMENT

- What would be the optimal replacement strategy?

OPTIMAL REPLACEMENT

- What would be the optimal replacement strategy?
- Well … optimally, you want to remove the page that won’t be used for the longest period.

OPTIMAL REPLACEMENT

- What would be the optimal replacement strategy?
- Well … optimally, you want to remove the page that won’t be used for the longest period.
- So if we have P1 and P2 .. P1 will be used in 10 clock cycles and P2 in 20, we remove P2.

OPTIMAL REPLACEMENT

- What would be the optimal replacement strategy?
- Well … optimally, you want to remove the page that won't be used for the longest period.
- So if we have P1 and P2 .. P1 will be used in 10 clock cycles and P2 in 20, we remove P2.
- Any particular challenge that you foresee with this algorithm?

OPTIMAL REPLACEMENT

- What would be the optimal replacement strategy?
- Well … optimally, you want to remove the page that won't be used for the longest period.
- So if we have P1 and P2 .. P1 will be used in 10 clock cycles and P2 in 20, we remove P2.
- Any particular challenge that you foresee with this algorithm?
- You need to see the future to know which page will be used when.
- Why is it important then?

OPTIMAL REPLACEMENT

- What would be the optimal replacement strategy?
- Well … optimally, you want to remove the page that won't be used for the longest period.
- So if we have P1 and P2 .. P1 will be used in 10 clock cycles and P2 in 20, we remove P2.
- Any particular challenge that you foresee with this algorithm?
- You need to see the future to know which page will be used when.
- Why is it important then? It provides a benchmark/compass for new strategies.

3-FRAME OPTIMAL REPLACEMENT

Page requests : 1 2 3 4 1 2 5 1 2 3 4 5

3-FRAME OPTIMAL REPLACEMENT

✖ ✖ ✖

Page requests : 1 2 3 4 1 2 5 1 2 3 4 5

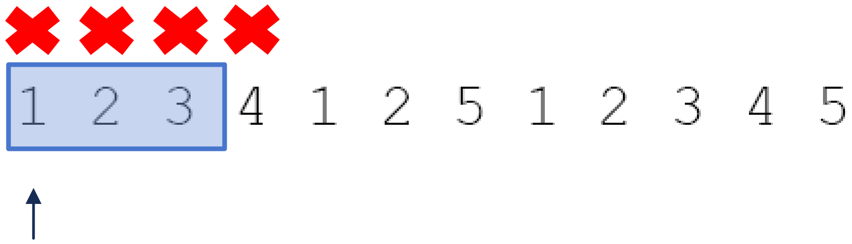
3-FRAME OPTIMAL REPLACEMENT

✖ ✖ ✖ ✖

Page requests : 1 2 3 4 1 2 5 1 2 3 4 5

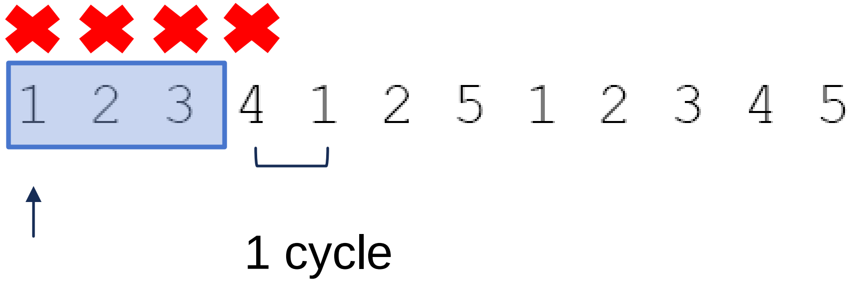
3-FRAME OPTIMAL REPLACEMENT

Page requests :



1 2 3 4 1 2 5 1 2 3 4 5

3-FRAME OPTIMAL REPLACEMENT

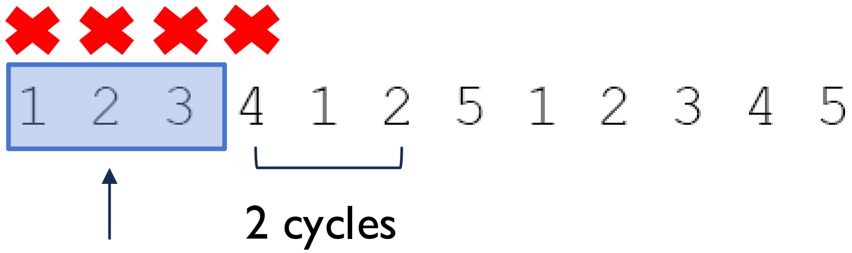


Page requests : 1 2 3 4 1 2 5 1 2 3 4 5

1 cycle

3-FRAME OPTIMAL REPLACEMENT

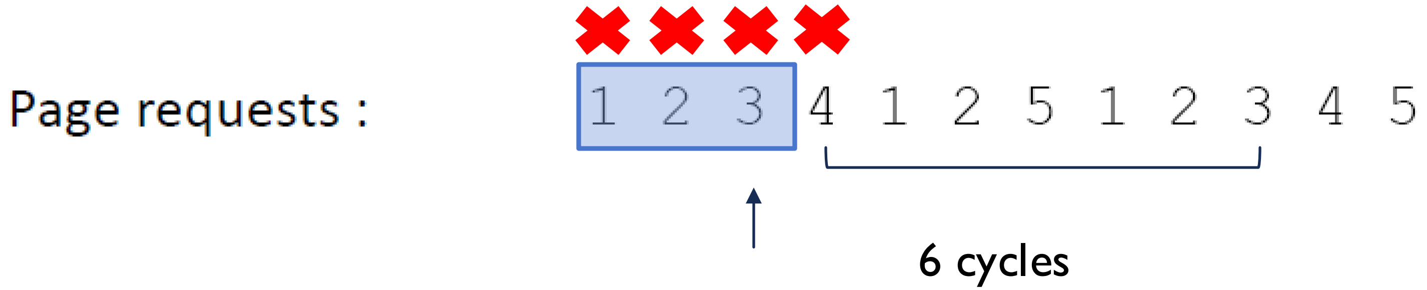
Page requests :



1 2 3 4 1 2 5 1 2 3 4 5

2 cycles

3-FRAME OPTIMAL REPLACEMENT

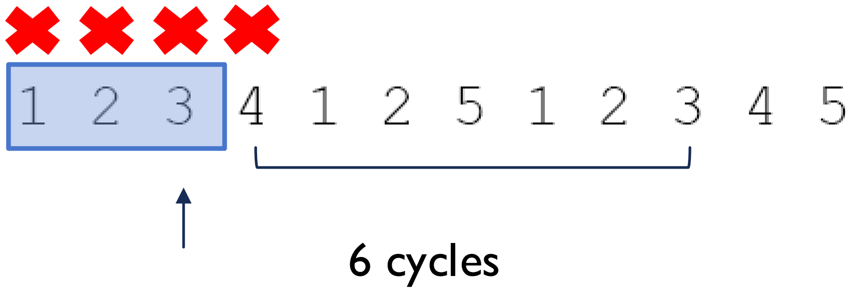


Page requests : 1 2 3 4 1 2 5 1 2 3 4 5

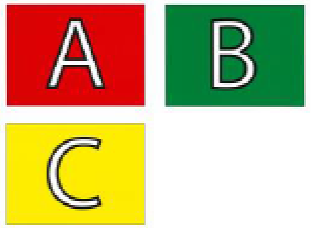
6 cycles

3-FRAME OPTIMAL REPLACEMENT

Page requests :



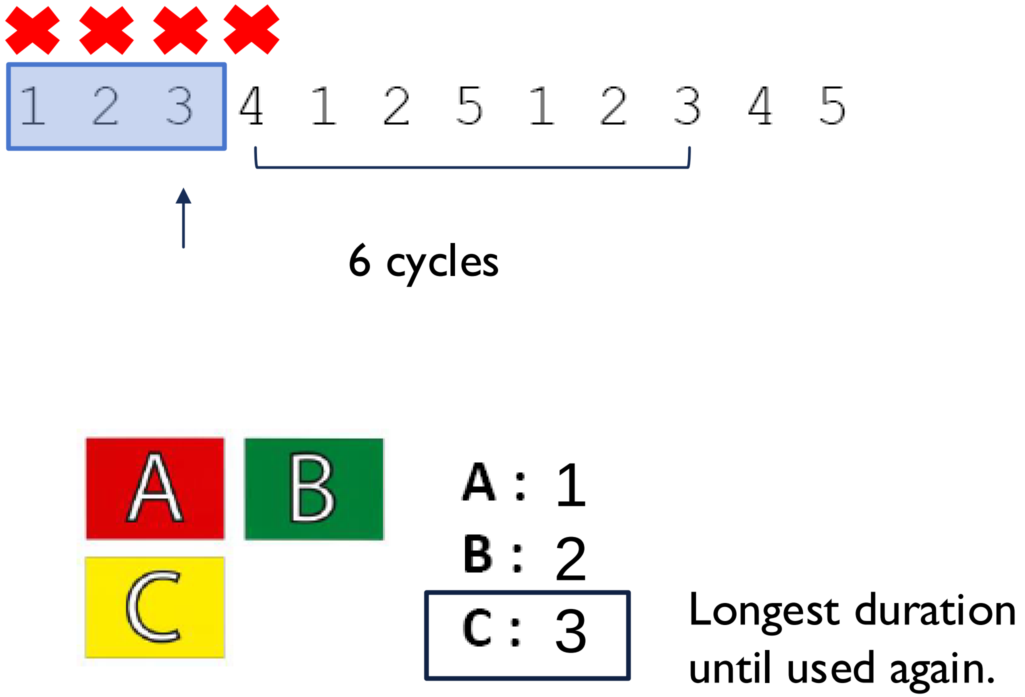
Which page should we remove from memory?



A : 1
B : 2
C : 3

3-FRAME OPTIMAL REPLACEMENT

Page requests :



1 2 3 4 1 2 5 1 2 3 4 5

6 cycles

Which page should we remove from memory?

A B C

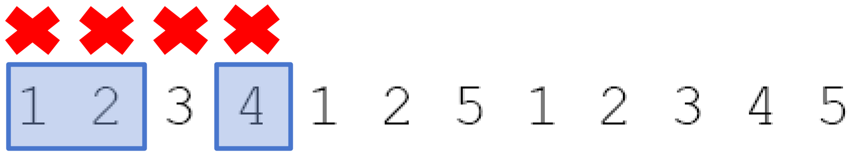
A : 1

B : 2

C : 3

Longest duration until used again.

3-FRAME OPTIMAL REPLACEMENT



Page requests : 1 2 3 4 1 2 5 1 2 3 4 5

3-FRAME OPTIMAL REPLACEMENT

✗ ✗ ✗ ✗ ✓

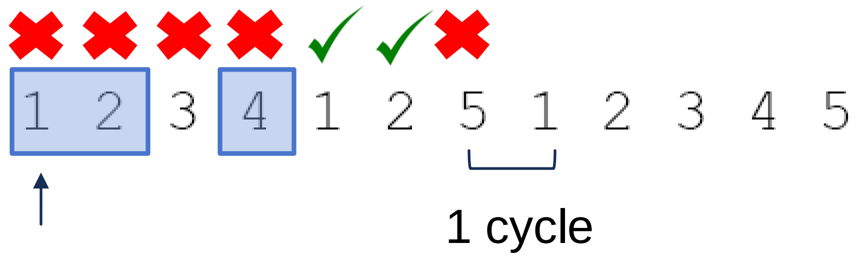
Page requests : 1 2 3 4 1 2 5 1 2 3 4 5

3-FRAME OPTIMAL REPLACEMENT

Worksheet Q1

	✖	✖	✖	✖	✓	✓						
Page requests :	1	2	3	4	1	2	5	1	2	3	4	5

3-FRAME OPTIMAL REPLACEMENT



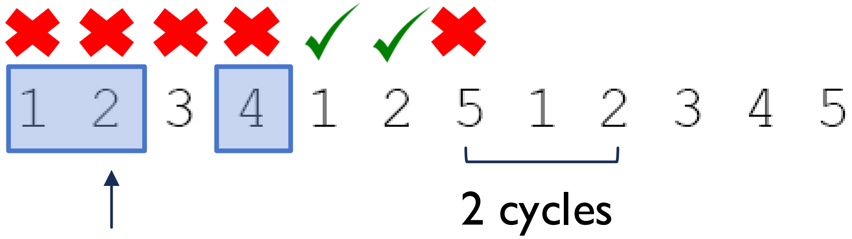
✗ ✗ ✗ ✗ ✓ ✓ ✗

Page requests : 1 2 3 4 1 2 5 1 2 3 4 5

1 cycle

3-FRAME OPTIMAL REPLACEMENT

Page requests :

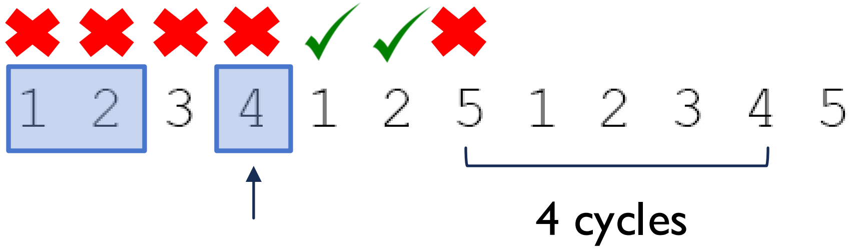


✗ ✗ ✗ ✗ ✓ ✓ ✗

1 2 3 4 1 2 5 1 2 3 4 5

2 cycles

3-FRAME OPTIMAL REPLACEMENT



✗ ✗ ✗ ✗ ✓ ✓ ✗

Page requests : 1 2 3 4 1 2 5 1 2 3 4 5

4 cycles

3-FRAME OPTIMAL REPLACEMENT

	✖	✖	✖	✖	✓	✓	✖					
Page requests :	1	2	3	4	1	2	5	1	2	3	4	5

3-FRAME OPTIMAL REPLACEMENT

	✗	✗	✗	✗	✓	✓	✗	✓				
Page requests :	1	2	3	4	1	2	5	1	2	3	4	5

3-FRAME OPTIMAL REPLACEMENT

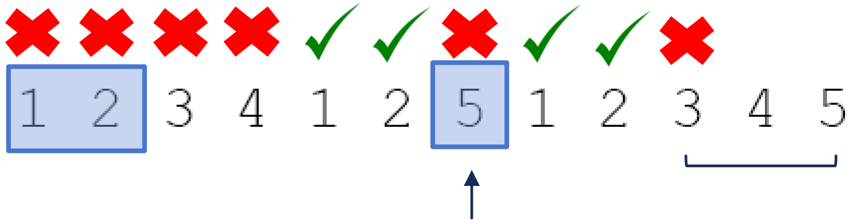
	✗	✗	✗	✗	✓	✓	✗	✓	✓			
Page requests :	1	2	3	4	1	2	5	1	2	3	4	5

3-FRAME OPTIMAL REPLACEMENT

	✗	✗	✗	✗	✓	✓	✗	✓	✓	✗		
Page requests :	1	2	3	4	1	2	5	1	2	3	4	5

3-FRAME OPTIMAL REPLACEMENT

Page requests :



✗ ✗ ✗ ✗ ✓ ✓ ✗ ✓ ✓ ✗

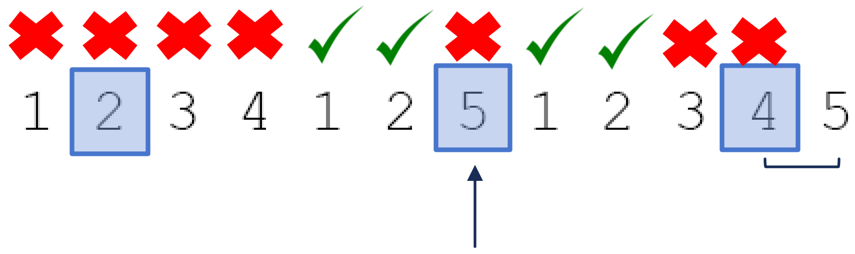
1 2 3 4 1 2 5 1 2 3 4 5

3-FRAME OPTIMAL REPLACEMENT

	✗	✗	✗	✗	✓	✓	✗	✓	✓	✗		
Page requests :	1	$\boxed{2}$	3	4	1	2	$\boxed{5}$	1	2	$\boxed{3}$	4	5

3-FRAME OPTIMAL REPLACEMENT

Page requests :



✗ ✗ ✗ ✗ ✓ ✓ ✗ ✓ ✓ ✗ ✗

1 2 3 4 1 2 5 1 2 3 4 5

3-FRAME OPTIMAL REPLACEMENT

	✗	✗	✗	✗	✓	✓	✗	✓	✓	✗	✗	✓
Page requests :	1	$\boxed{2}$	3	4	1	2	$\boxed{5}$	1	2	3	$\boxed{4}$	5

- Total Page Faults: 7
- 3-Frame FIFO: 9
- 4-Frame FIFO: 10

LEAST RECENTLY USED

- We obviously can’t use Optimal Replacement …
- What’s a good approximation for Optimal Replacement?

LEAST RECENTLY USED

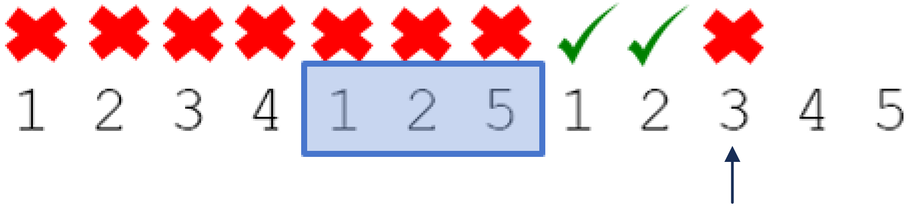
- We obviously can’t use Optimal Replacement …
- What’s a good approximation for Optimal Replacement? Least Recently Used.

LEAST RECENTLY USED

- Least Recently Used attempts to estimate which of the pages in memory would be not needed for the longest period.
- The idea is that a page that we have not accessed for a long while, won't be used any time soon .. We're probably done with it for now.
- How is that different from FIFO replacement?
- FIFO looks at the time the page was ***brought into memory*** … LRU looks at the time the page was ***last accessed.***

FIFO VS LRU

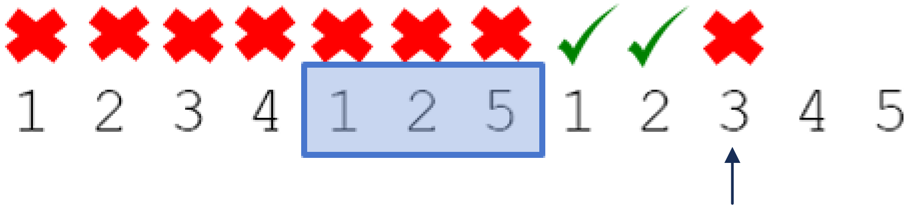
Worksheet Q2



✗ ✗ ✗ ✗ ✗ ✗ ✗ ✓ ✓ ✗

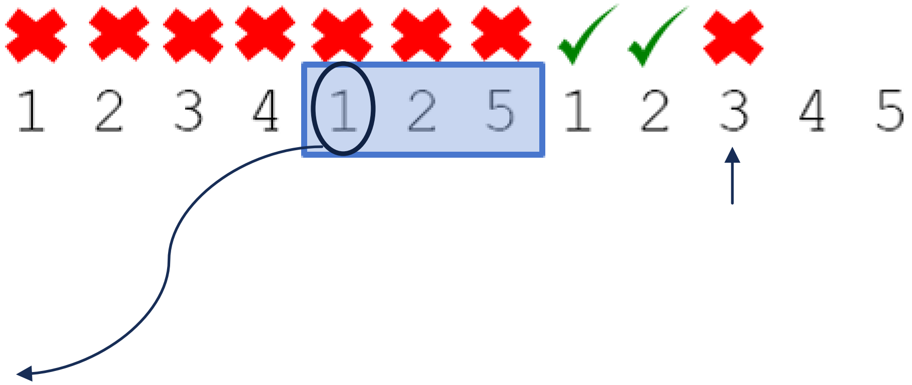
1 2 3 4 1 2 5 1 2 3 4 5

LRU



✗	✗	✗	✗	✗	✗	✗	✓	✓	✗		
1	2	3	4	1	2	5	1	2	3	4	5

FIFO VS LRU

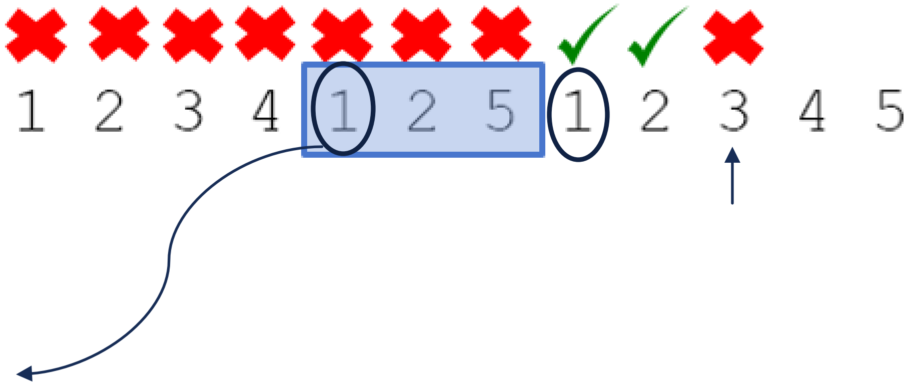


✗ ✗ ✗ ✗ ✗ ✗ ✗ ✓ ✓ ✗

1 2 3 4 1 2 5 1 2 3 4 5

LRU: Page 1 has been recently used (2 cycles ago)

FIFO VS LRU

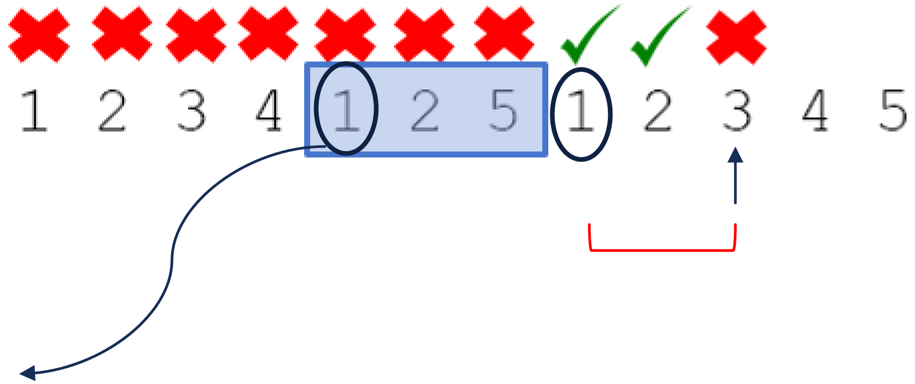


✗ ✗ ✗ ✗ ✗ ✗ ✗ ✓ ✓ ✗

1 2 3 4 1 2 5 1 2 3 4 5

LRU: Page 1 has been recently used (2 cycles ago)

FIFO VS LRU

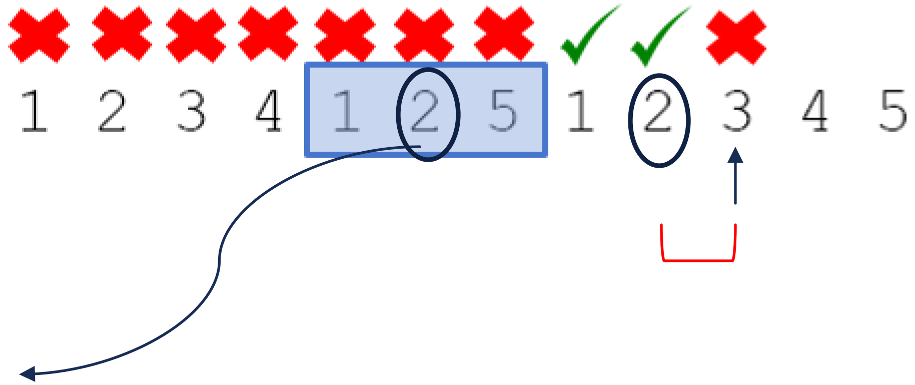


✖ ✖ ✖ ✖ ✖ ✖ ✖ ✓ ✓ ✖

1 2 3 4 1 2 5 1 2 3 4 5

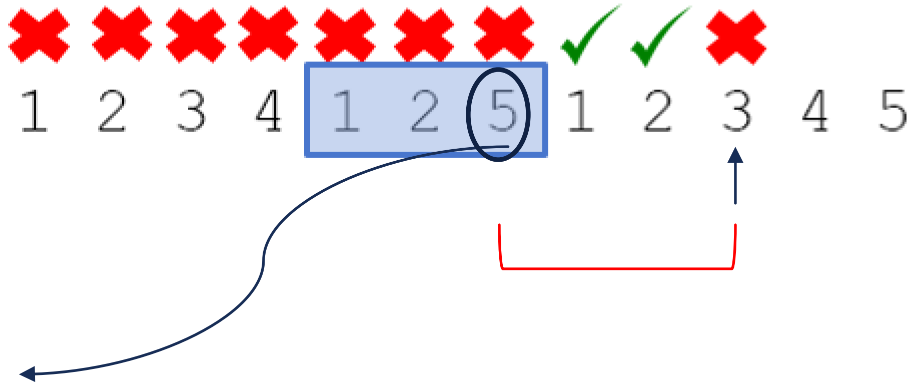
LRU: Page 1 has been recently used (2 cycles ago)

FIFO VS LRU



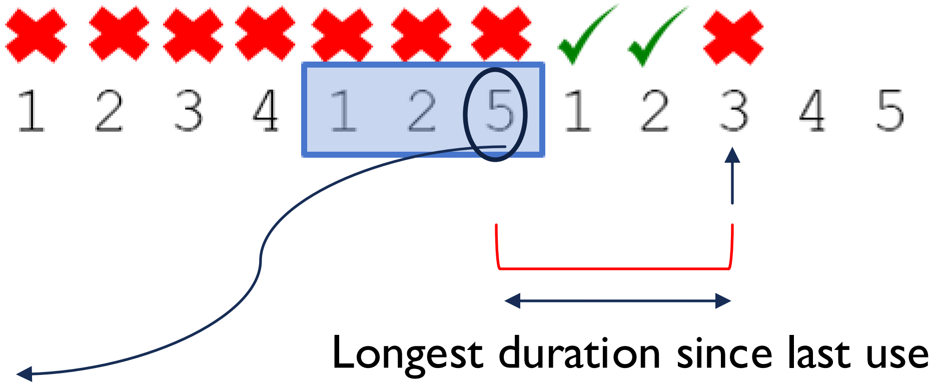
LRU: Page 2 has been recently used (1 cycle ago)

FIFO VS LRU



LRU: Page 5 has been less recently used (3 cycles ago)

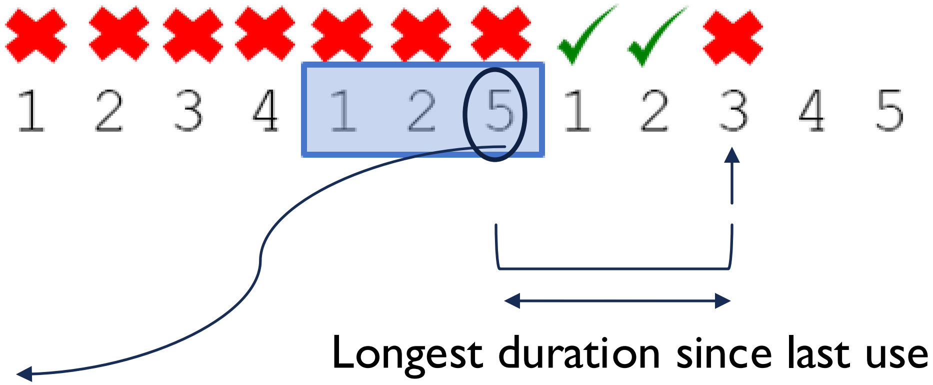
FIFO VS LRU



Longest duration since last use

LRU: Page 5 has been less recently used (3 cycles ago)

FIFO VS LRU

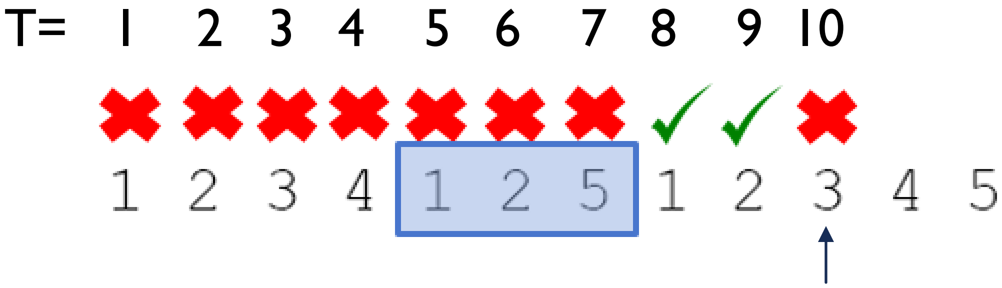


Longest duration since last use

LRU: Page 5 has been less recently used (3 cycles ago)

LRU: Remove 5 (the last page inserted)
FIFO: Remove 1 (the first/oldest page inserted).

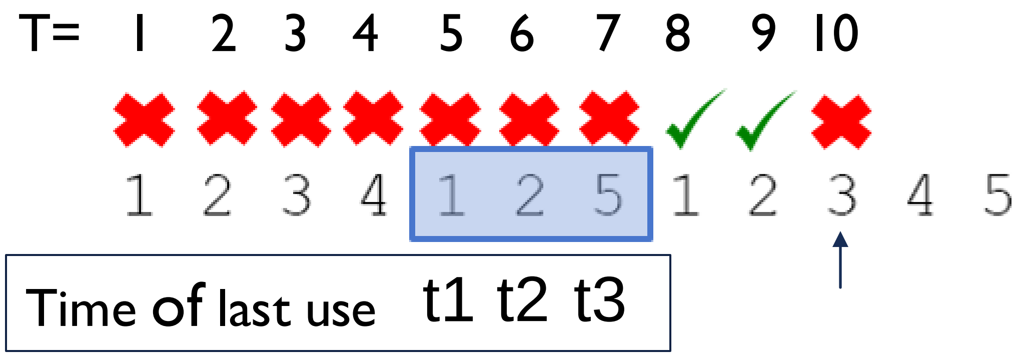
LRU DATA STRUCTURE



T=	1	2	3	4	5	6	7	8	9	10		
	✗	✗	✗	✗	✗	✗	✗	✓	✓	✗		
	1	2	3	4	1	2	5	1	2	3	4	5

Q: How to keep track of last access/reference?

LRU DATA STRUCTURE



T= 1 2 3 4 5 6 7 8 9 10

✗ ✗ ✗ ✗ ✗ ✗ ✗ ✓ ✓ ✗

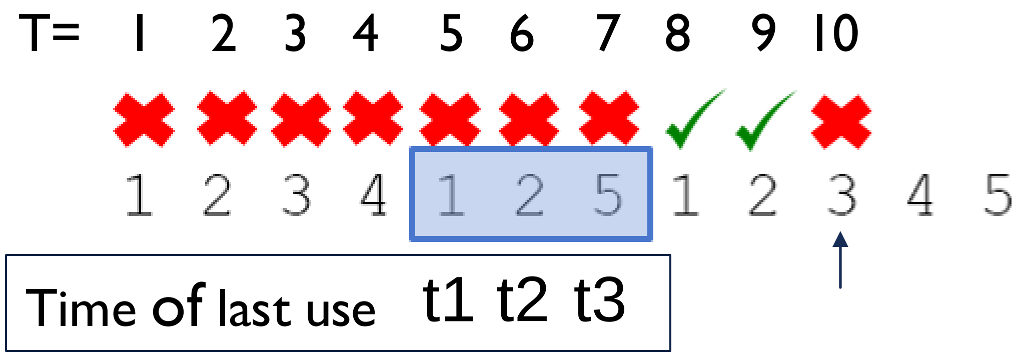
1 2 3 4 1 2 5 1 2 3 4 5

Time of last use t1 t2 t3

Q: How to keep track of last access/reference?

- Need a data structure that keeps track of last time use of each frame in memory.

LRU DATA STRUCTURE



T= 1 2 3 4 5 6 7 8 9 10

✗ ✗ ✗ ✗ ✗ ✗ ✗ ✓ ✓ ✗

1 2 3 4 1 2 5 1 2 3 4 5

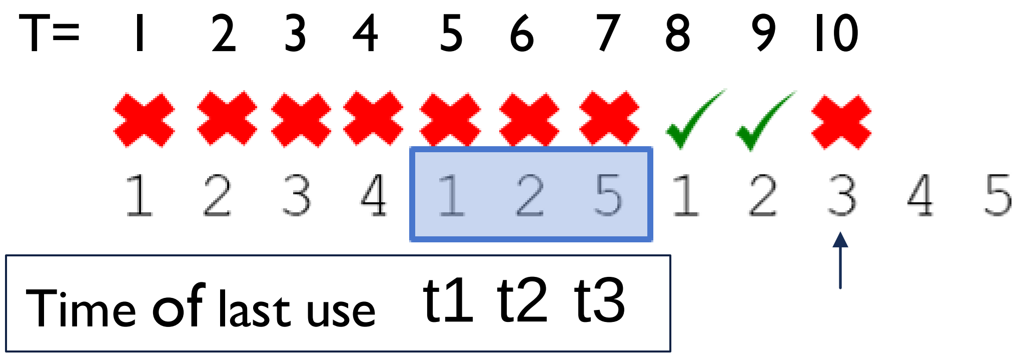
Time of last use t1 t2 t3

Q: How to keep track of last access/reference?

- Need a data structure that keeps track of last time use of each frame in memory.

Worksheet Q3: What would be the complications?

LRU DATA STRUCTURE



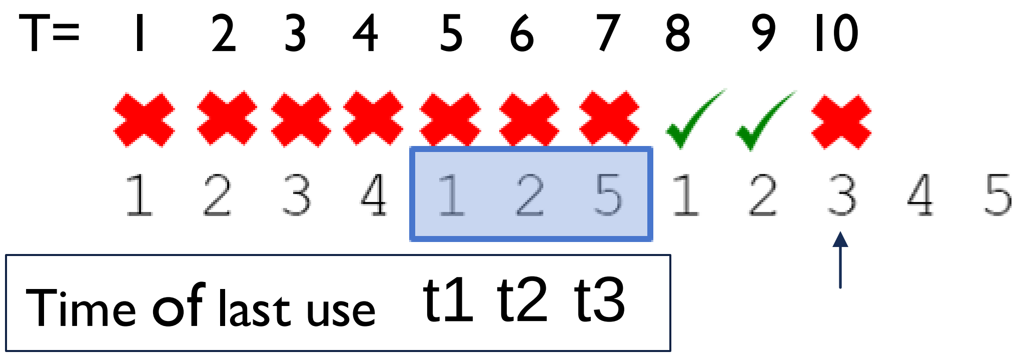
T=	1	2	3	4	5	6	7	8	9	10		
	✗	✗	✗	✗	✗	✗	✗	✓	✓	✗		
	1	2	3	4	1	2	5	1	2	3	4	5

Time of last use t1 t2 t3

Q: How to keep track of last access/reference?

- Need a data structure that keeps track of last time use of each frame in memory.
- How many frames in memory? For an 8GB ram, 4KB page size … that's around 2 million entries.

LRU DATA STRUCTURE



T= 1 2 3 4 5 6 7 8 9 10

✗ ✗ ✗ ✗ ✗ ✗ ✗ ✓ ✓ ✗

1 2 3 4 1 2 5 1 2 3 4 5

Time of last use t1 t2 t3

Q: How to keep track of last access/reference?

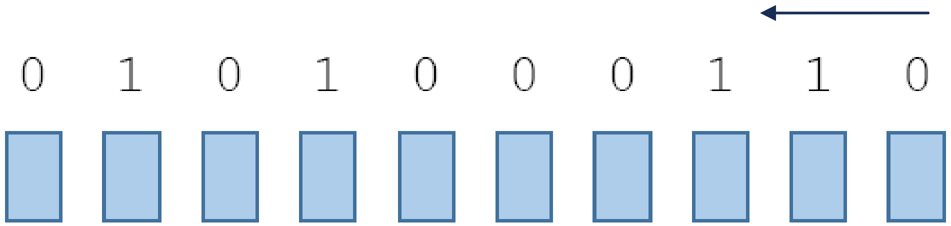
- Need a data structure that keeps track of last time use of each frame in memory.
- How many frames in memory? For an 8GB ram, 4KB page size … that's around 2 million entries.

- Impractical:
  - Too large
  - Need to sort 2 million entries every access!

LRU SUPPORT

LRU support

Common architecture support provides a single reference bit



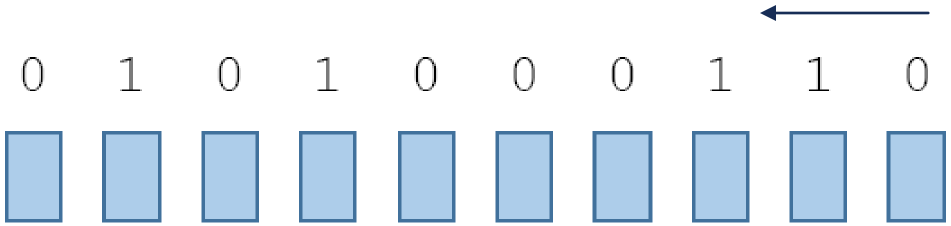
- When a page is swapped in, the bit it set to 0
- When a page is read from (referenced) or written to, the bit is set to 1

LRU SUPPORT

LRU support

- Reduces size of data structure saved.
- No need to sort, remove the first '0': less time traversing.

Common architecture support provides a single reference bit

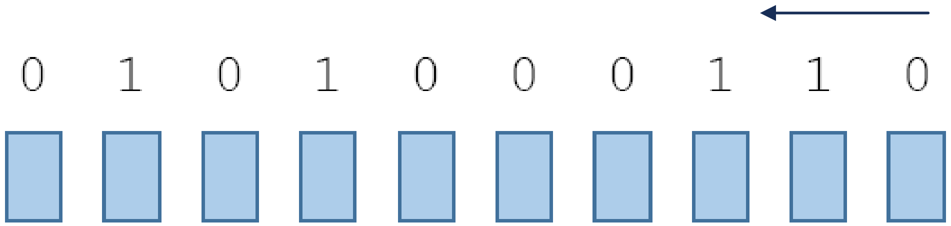


- When a page is swapped in, the bit it set to 0
- When a page is read from (referenced) or written to, the bit is set to 1

LRU SUPPORT

LRU support

Common architecture support provides a single reference bit

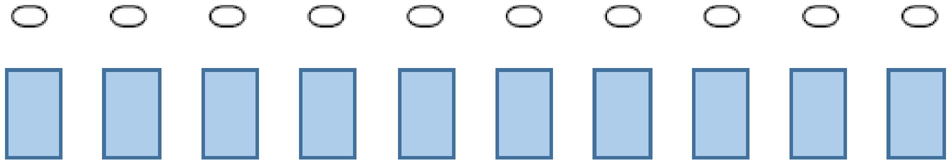


- When a page is swapped in, the bit it set to 0
- When a page is read from (referenced) or written to, the bit is set to 1

- Reduces size of data structure saved.
- No need to sort, remove the first ‘0’: less time traversing.

- Can only track whether a page has been read or not …
- Not very useful

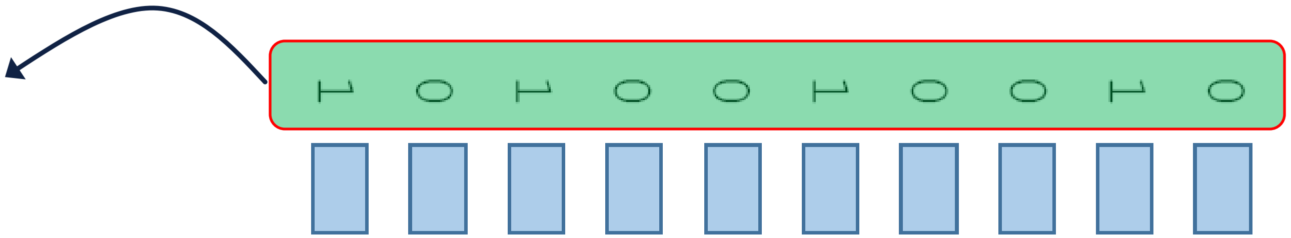
RECENT HISTORY



- Improvement: Last n-cycles history.

RECENT HISTORY

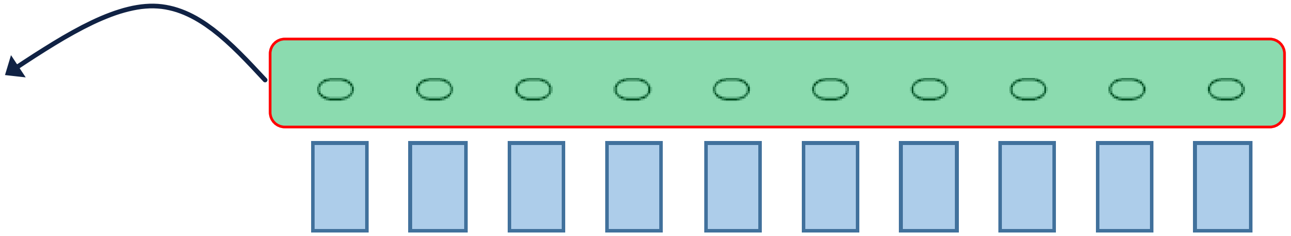
Periodically reset
all reference bits



- Improvement: Last n-cycles history.

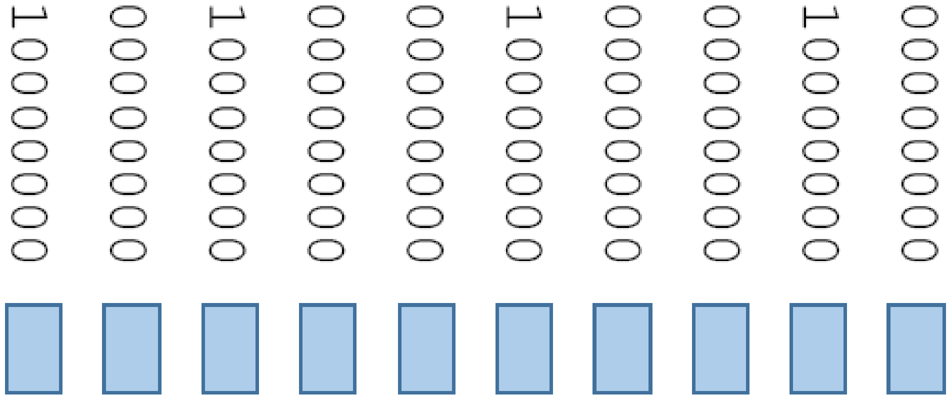
RECENT HISTORY

Periodically reset
all reference bits



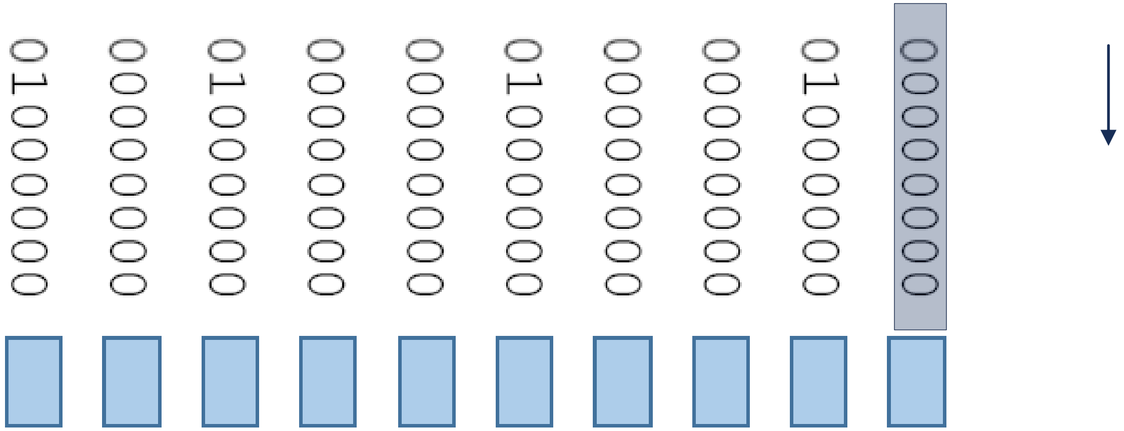
- Now the reference bit stores whether a frame have been referenced since last period started … rather than since it was brought in.

TRACKING RECENT HISTORY



- Tracks a longer history
- Still smaller than data structure.
- Remove first page that has all ‘zeros’ in its history: no need to sort.

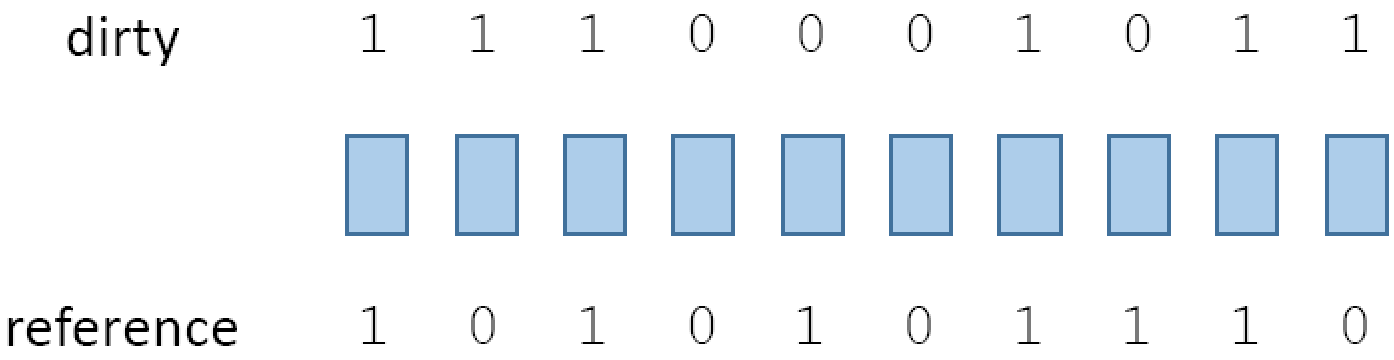
TRACKING RECENT HISTORY



- Tracks a longer history
- Still smaller than data structure.
- Remove first page that has all ‘zeros’ in its history:
  - no need to sort.
  - No need to go through all memory

DIRTY BIT FOR PAGE REPLACEMENT

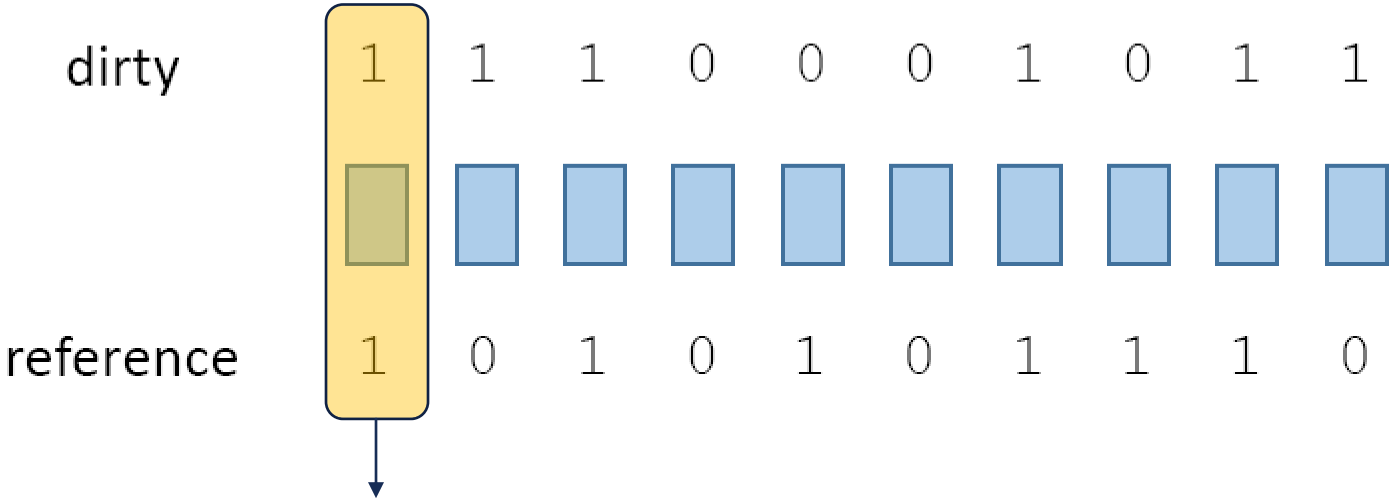
Q: Can we use dirty bit in page replacement algorithm?



dirty	1	1	1	0	0	0	1	0	1	1
reference	1	0	1	0	1	0	1	1	1	0

DIRTY BIT FOR PAGE REPLACEMENT

Q: Can we use dirty bit in page replacement algorithm?



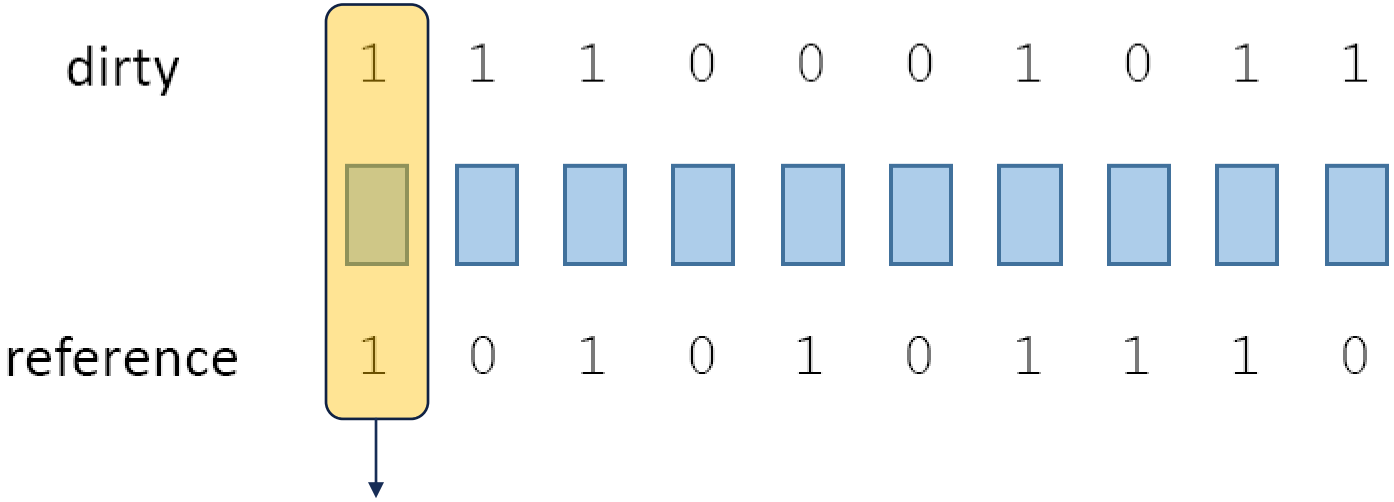
dirty	1	1	1	0	0	0	1	0	1	1
reference	1	0	1	0	1	0	1	1	1	0

Modified
and recently
used.

DIRTY BIT FOR PAGE REPLACEMENT

Q: Can we use dirty bit in page replacement algorithm?

Not Modified: No need to write back!



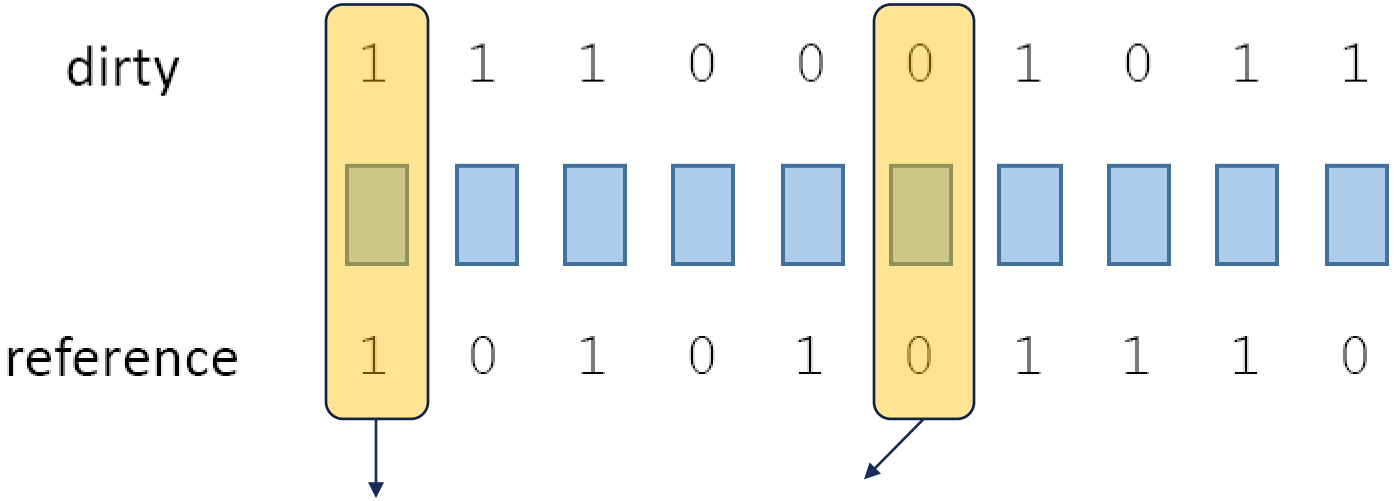
dirty	1	1	1	0	0	0	1	0	1	1
reference	1	0	1	0	1	0	1	1	1	0

Modified and recently used.

DIRTY BIT FOR PAGE REPLACEMENT

Q: Can we use dirty bit in page replacement algorithm?

Not Modified: No need to write back!



dirty	1	1	1	0	0	0	1	0	1	1
reference	1	0	1	0	1	0	1	1	1	0

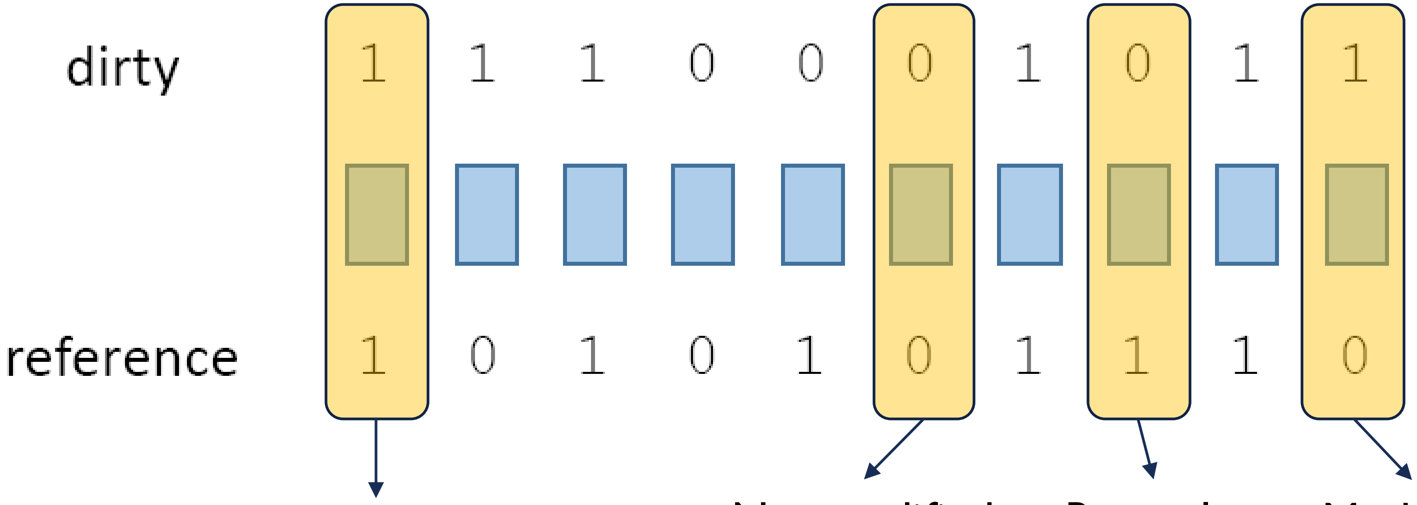
Modified and recently used.

Not modified and not recently used.

DIRTY BIT FOR PAGE REPLACEMENT

Q: Can we use dirty bit in page replacement algorithm?

Not Modified: No need to write back!



Modified and recently used.

Not modified and not recently used.

Recently used but not modified

Modified but not recently used.

DIRTY BIT FOR PAGE REPLACEMENT

(referenced , modified)

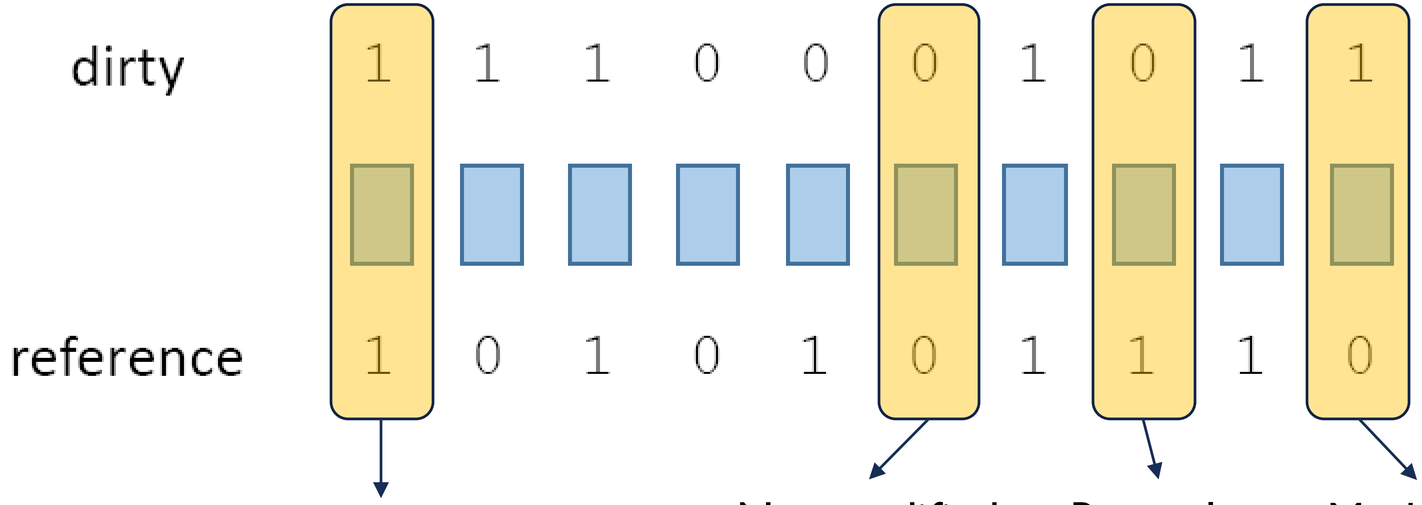
Evict first
“best” choice

(0,0) (0,1) (1,0) (1,1)

Evict last
“worst” choice

Not modified (no need for write back) and not recently used.

Modified (needs write back) and recently used (high probability of causing a page fault).



dirty	1	1	1	0	0	0	1	0	1	1
reference	1	0	1	0	1	0	1	1	1	0

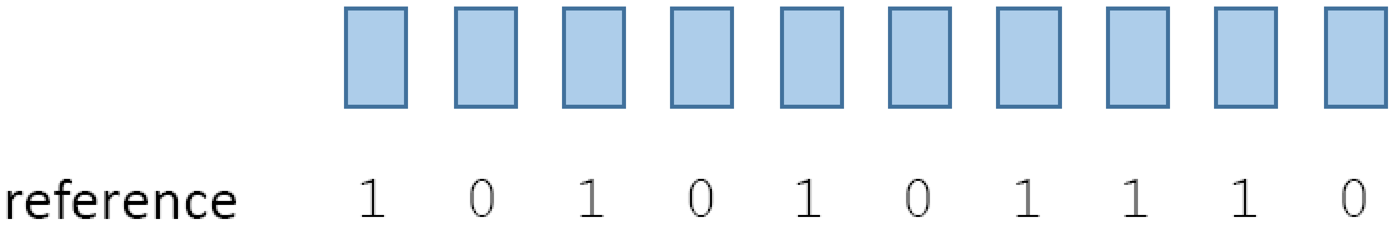
Modified and recently used.

Not modified and not recently used.

Recently used but not modified

Modified but not recently used.

DIRTY BIT FOR PAGE REPLACEMENT



reference 1 0 1 0 1 0 1 1 1 0

- Dirty bits indicate that a page needs to be written back.
- This slows down page replacement …
- Look at both, reference bit (history) and dirty bit.
- Pick a page that doesn’t need replacement!

DIRTY BIT

(referenced , modified)

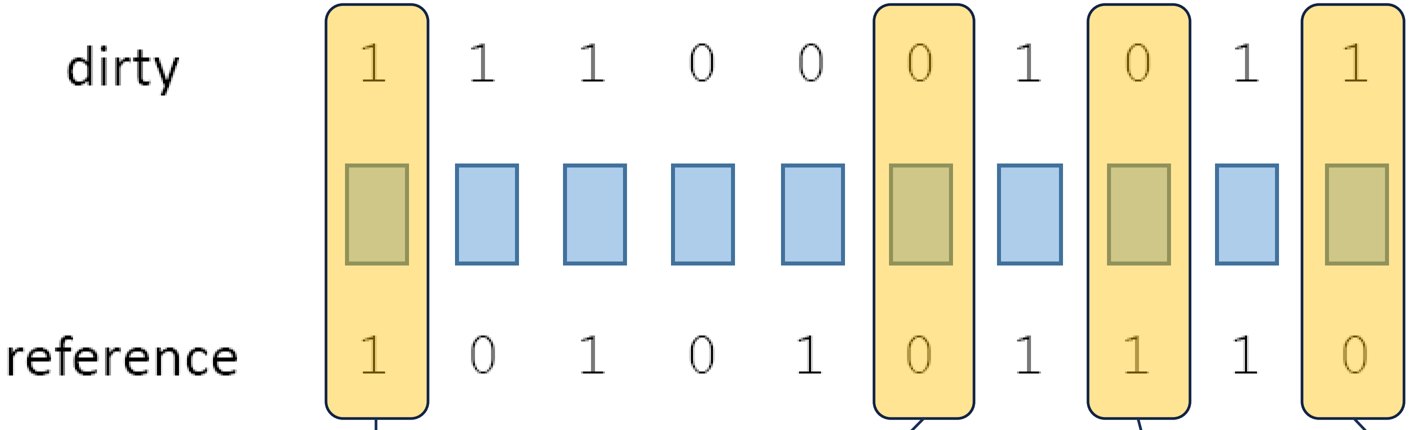
Evict first
“best” choice

(0,0) (0,1) (1,0) (1,1)

Evict last
“worst” choice

Not modified (no need for write back) and not recently used.

Modified (needs write back) and recently used (high probability of causing a page fault).



dirty	1	1	1	0	0	0	1	0	1	1
reference	1	0	1	0	1	0	1	1	1	0

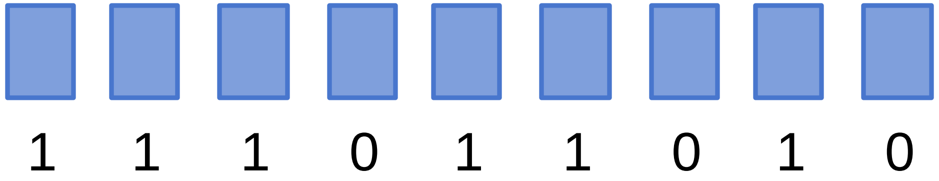
Modified and recently used.

Not modified and not recently used.

Recently used but not modified

Modified but not recently used.

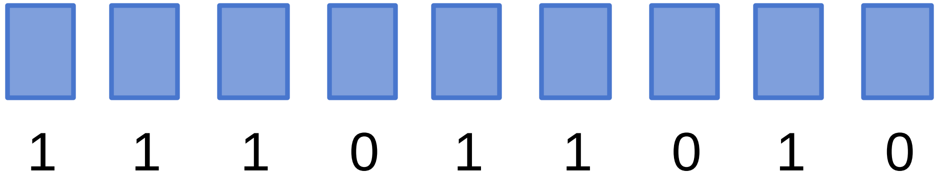
SECOND CHANCE



1 1 1 0 1 1 0 1 0

SECOND CHANCE

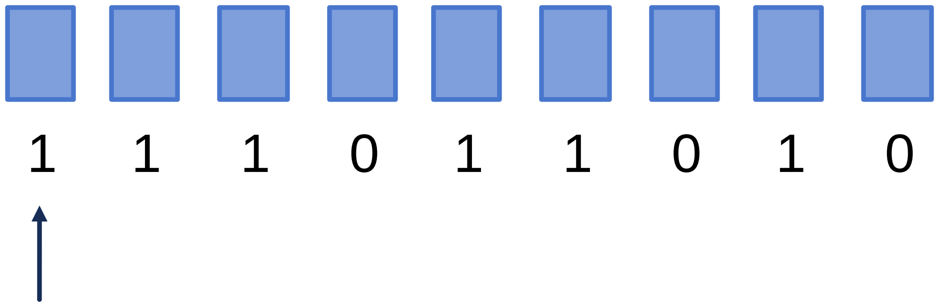
Similar to periodic reset …
with a simpler implementation.



1 1 1 0 1 1 0 1 0

SECOND CHANCE

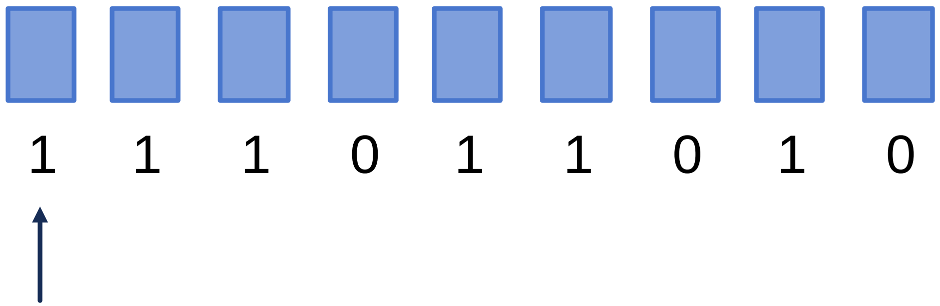
Instead of resetting periodically, we reset reference bits *while* searching for a not-recently-used page.



1 1 1 0 1 1 0 1 0

SECOND CHANCE

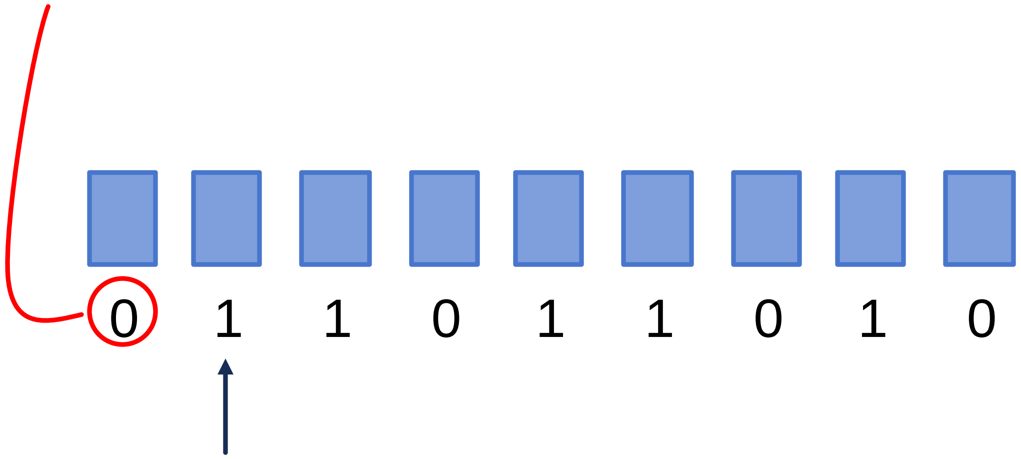
Instead of resetting periodically, we reset reference bits *while* searching for a not-recently-used page.



Page fault just occurred, need to replace frame … walking through memory

SECOND CHANCE

Instead of resetting periodically, we reset reference bits *while* searching for a not-recently-used page.

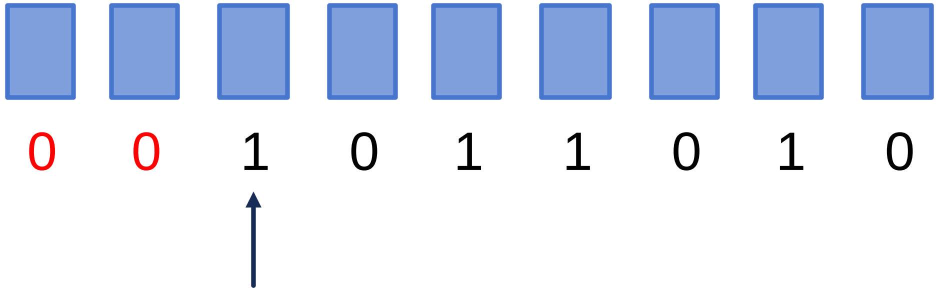


As OS walks through memory, it resets the ‘1’s to ‘0’s.

Page fault just occurred, need to replace frame … walking through memory

SECOND CHANCE

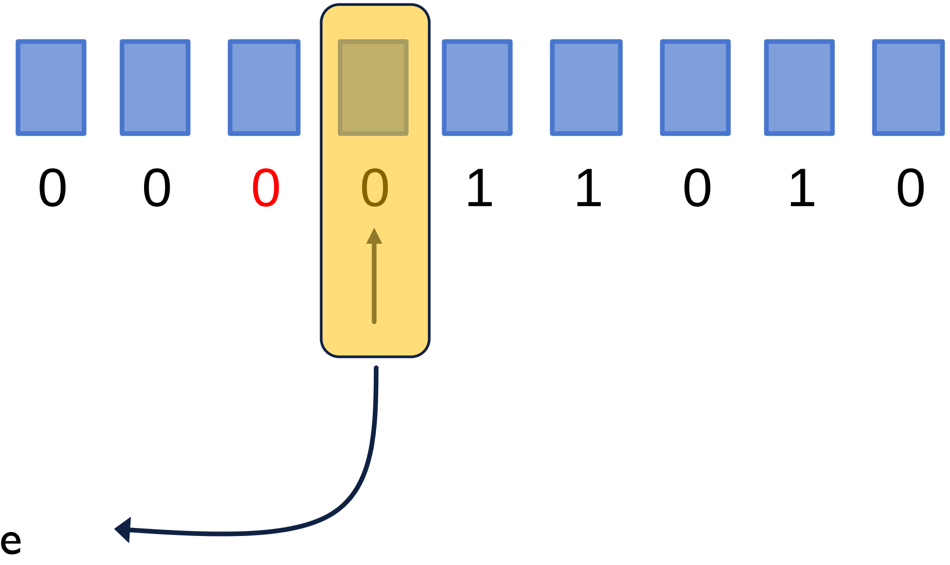
Instead of resetting periodically, we reset reference bits *while* searching for a not-recently-used page.



Page fault just occurred, need to replace frame … walking through memory

SECOND CHANCE

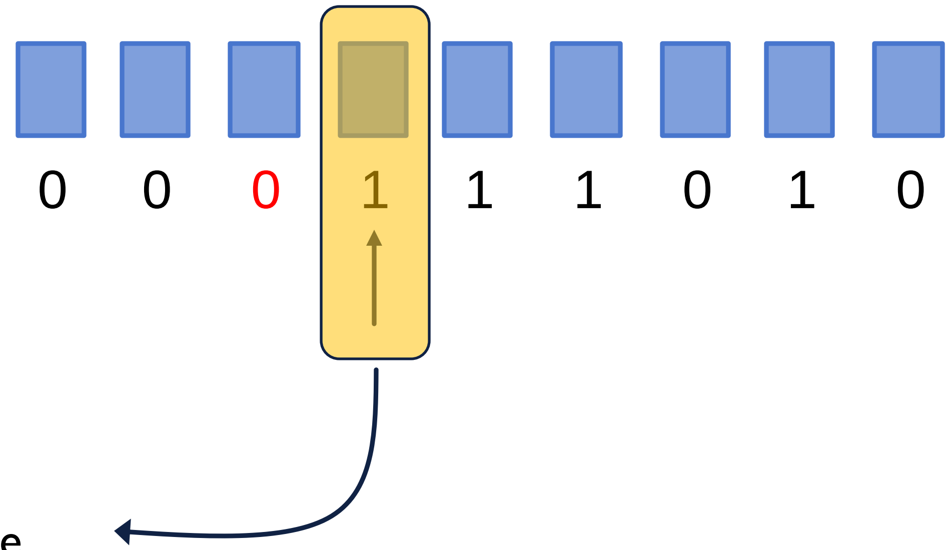
Instead of resetting periodically, we reset reference bits *while* searching for a not-recently-used page.



When we find a zero, we remove the frame.

SECOND CHANCE

Instead of resetting periodically, we reset reference bits *while* searching for a not-recently-used page.

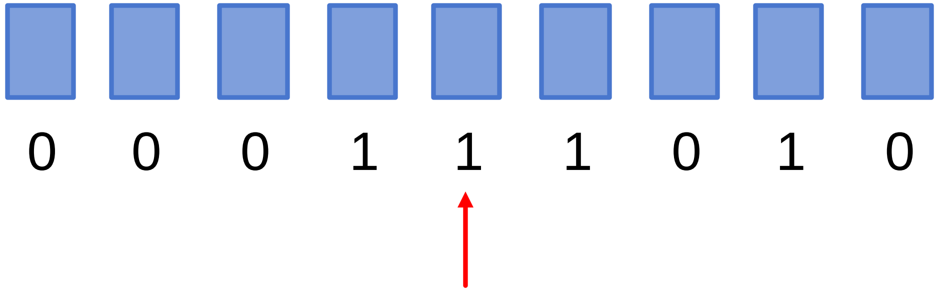


Remove

When we find a zero, we remove the frame.

SECOND CHANCE

Instead of resetting periodically, we reset reference bits *while* searching for a not-recently-used page.

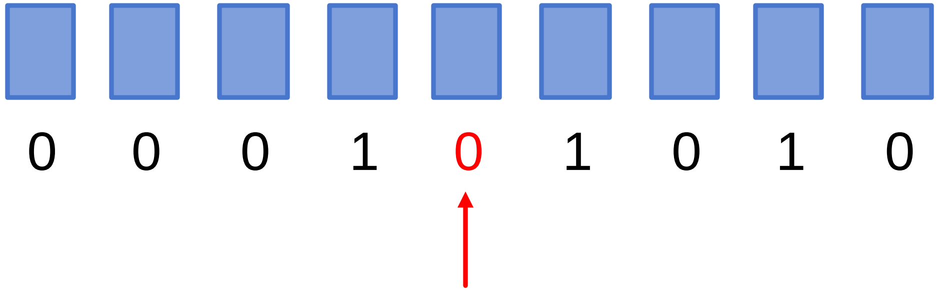


Worksheet Q1

Next replacement search continue from last pointer position.

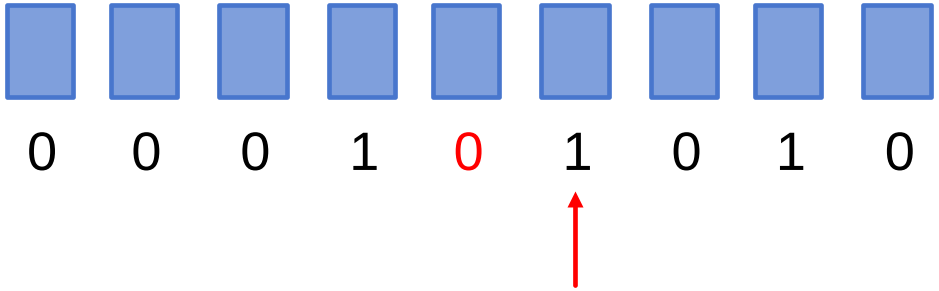
SECOND CHANCE

Instead of resetting periodically, we reset reference bits *while* searching for a not-recently-used page.



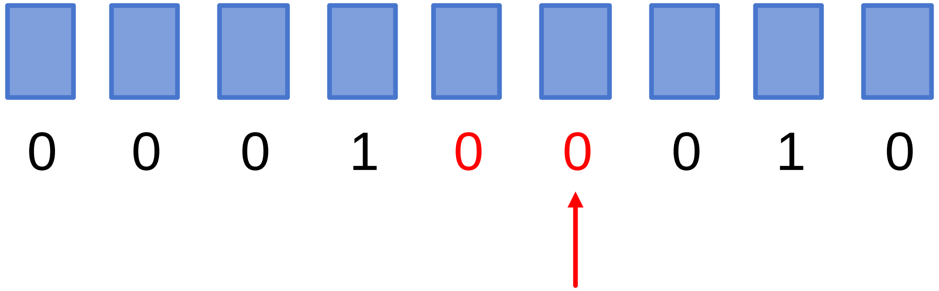
SECOND CHANCE

Instead of resetting periodically, we reset reference bits *while* searching for a not-recently-used page.



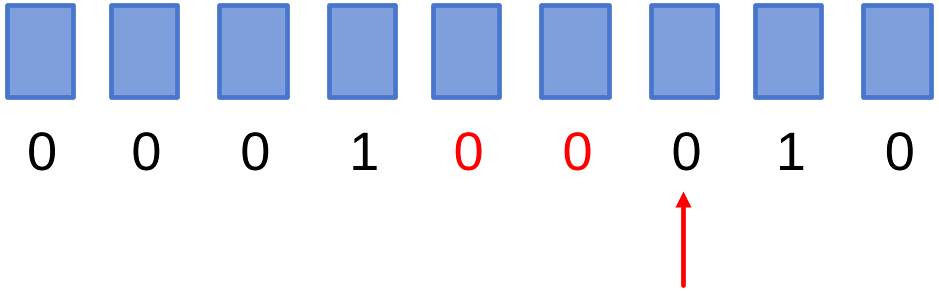
SECOND CHANCE

Instead of resetting periodically, we reset reference bits *while* searching for a not-recently-used page.



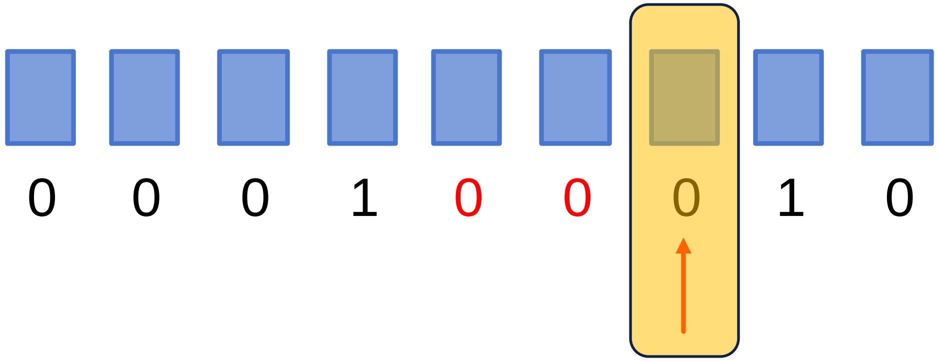
SECOND CHANCE

Instead of resetting periodically, we reset reference bits *while* searching for a not-recently-used page.



SECOND CHANCE

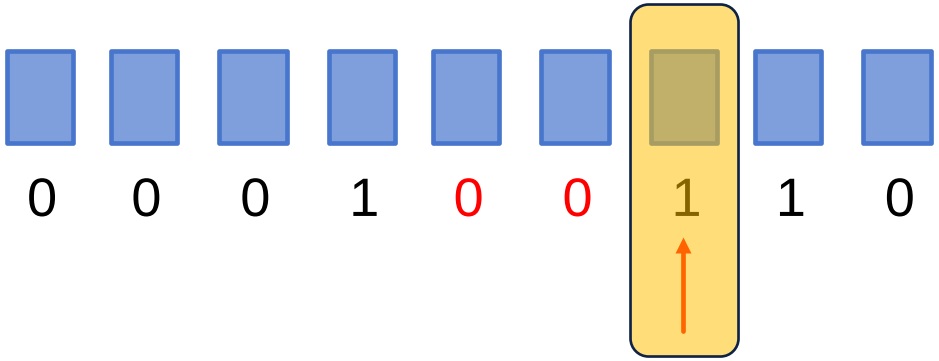
Instead of resetting periodically, we reset reference bits *while* searching for a not-recently-used page.



Next replacement search continue from last pointer position.

SECOND CHANCE

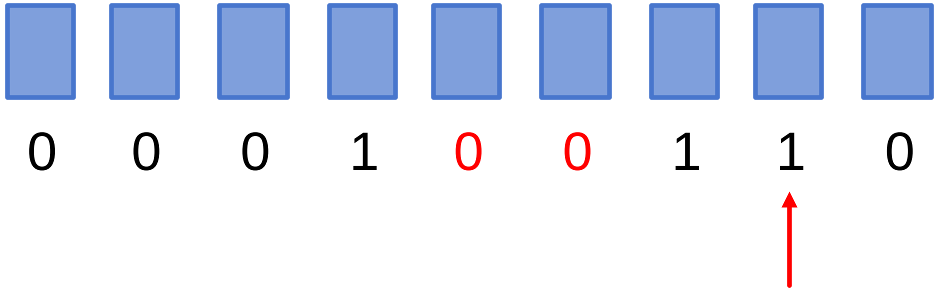
Instead of resetting periodically, we reset reference bits *while* searching for a not-recently-used page.



Next replacement search continue from last pointer position.

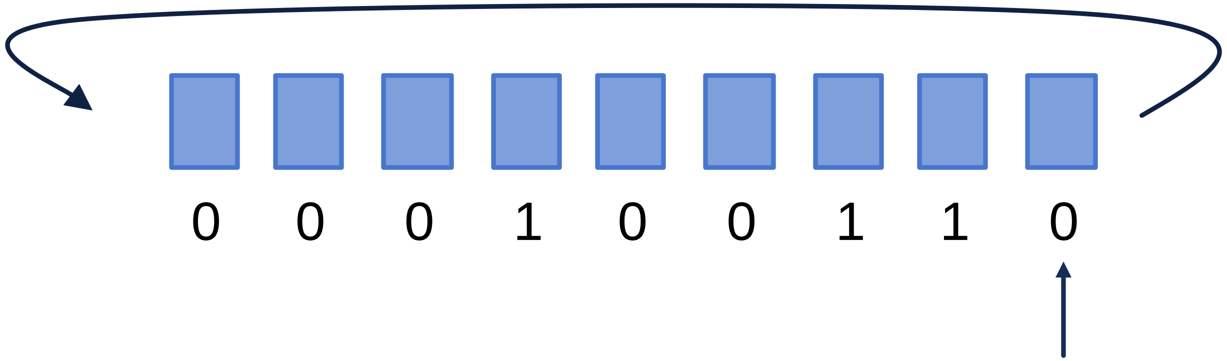
SECOND CHANCE

Instead of resetting periodically, we reset reference bits *while* searching for a not-recently-used page.



SECOND CHANCE

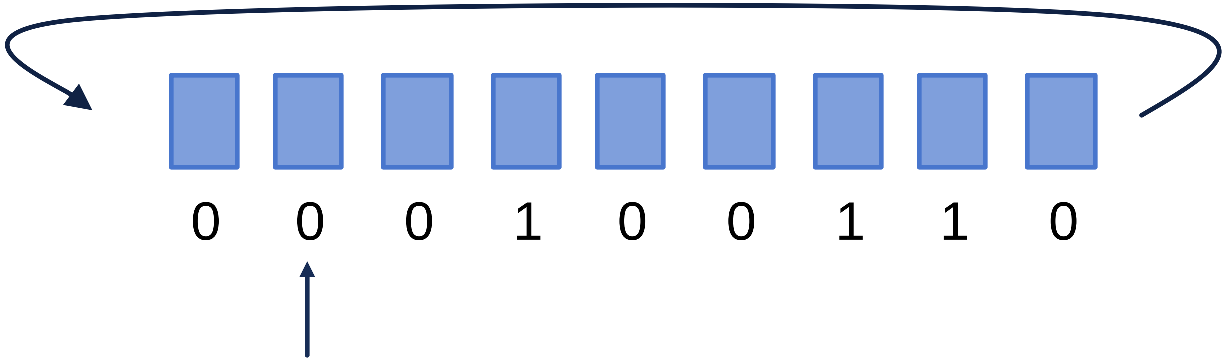
Instead of resetting periodically, we reset reference bits *while* searching for a not-recently-used page.



Circle back once you go through all memory frame …

SECOND CHANCE

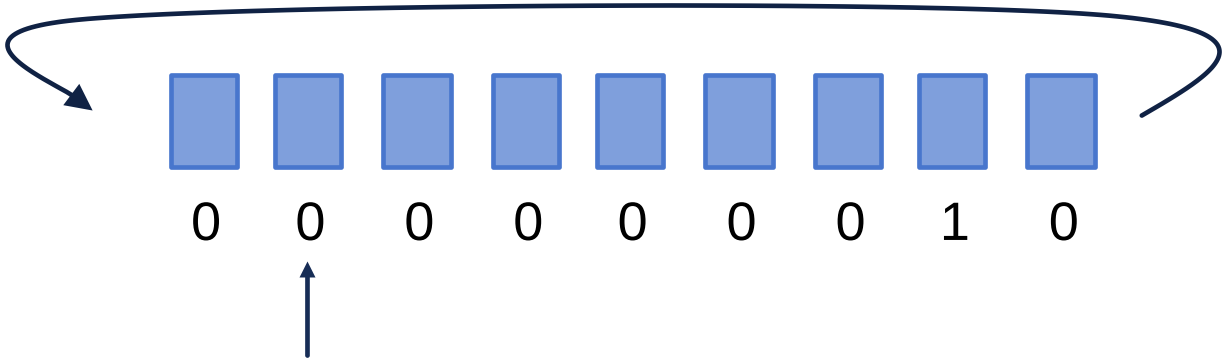
Instead of resetting periodically, we reset reference bits *while* searching for a not-recently-used page.



- Why is it called second chance?
- We remove frames that has not been referenced since our last "visit".
- This frame has stayed unreferenced since we reset it to '0'.

SECOND CHANCE

Instead of resetting periodically, we reset reference bits *while* searching for a not-recently-used page.

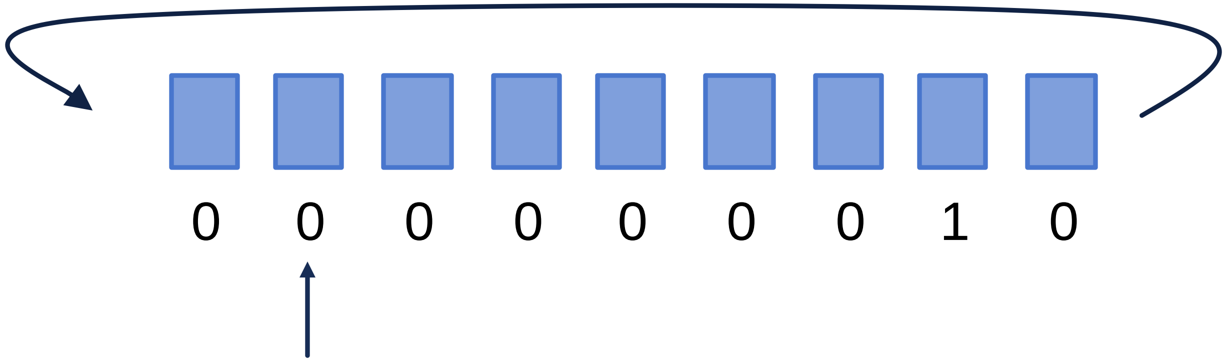


- Advantages?

Worksheet Q2

SECOND CHANCE

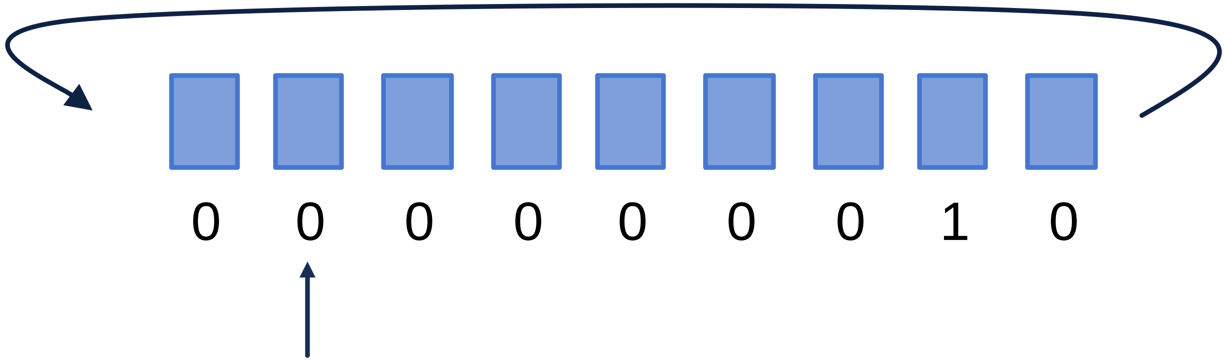
Instead of resetting periodically, we reset reference bits *while* searching for a not-recently-used page.



- Advantages?
- No need to pause programs to reset bits, we reset them during search.

SECOND CHANCE

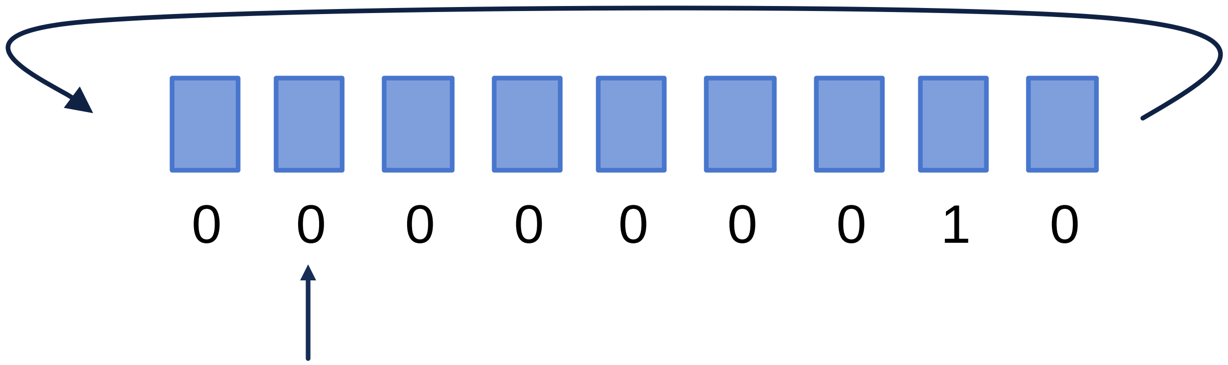
Instead of resetting periodically, we reset reference bits *while* searching for a not-recently-used page.



- Advantages?
- No need to pause programs to reset bits, we reset them during search.
- No need to reset all bits, we rest the ones that are relevant.

SECOND CHANCE

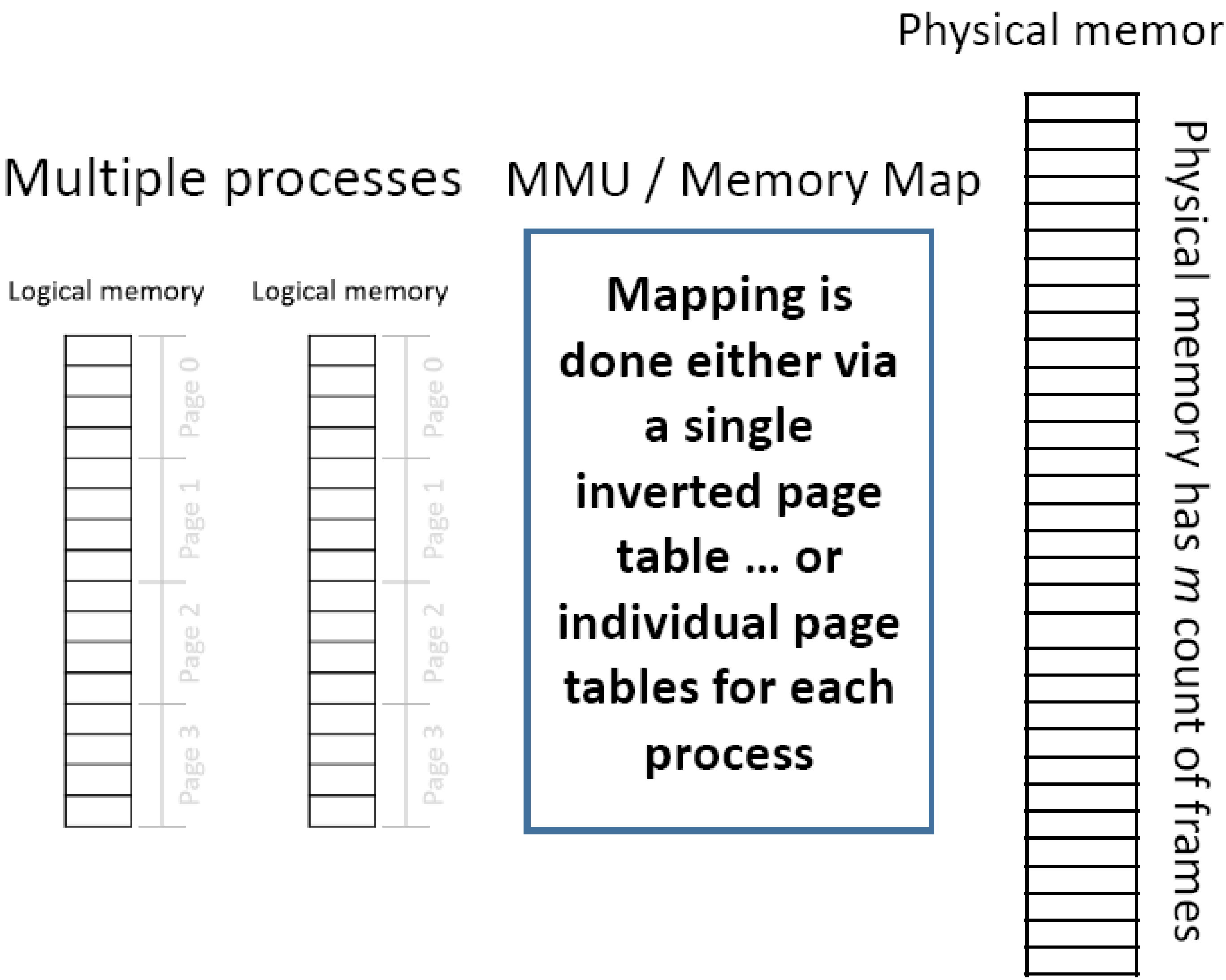
Instead of resetting periodically, we reset reference bits *while* searching for a not-recently-used page.



- Advantages?
- No need to pause programs to reset bits, we reset them during search.
- No need to reset all bits, we rest the ones that are relevant.
- Less stalls and shorter penalty.

FRAME ALLOCATION

- So far we talked about how to replace frames.

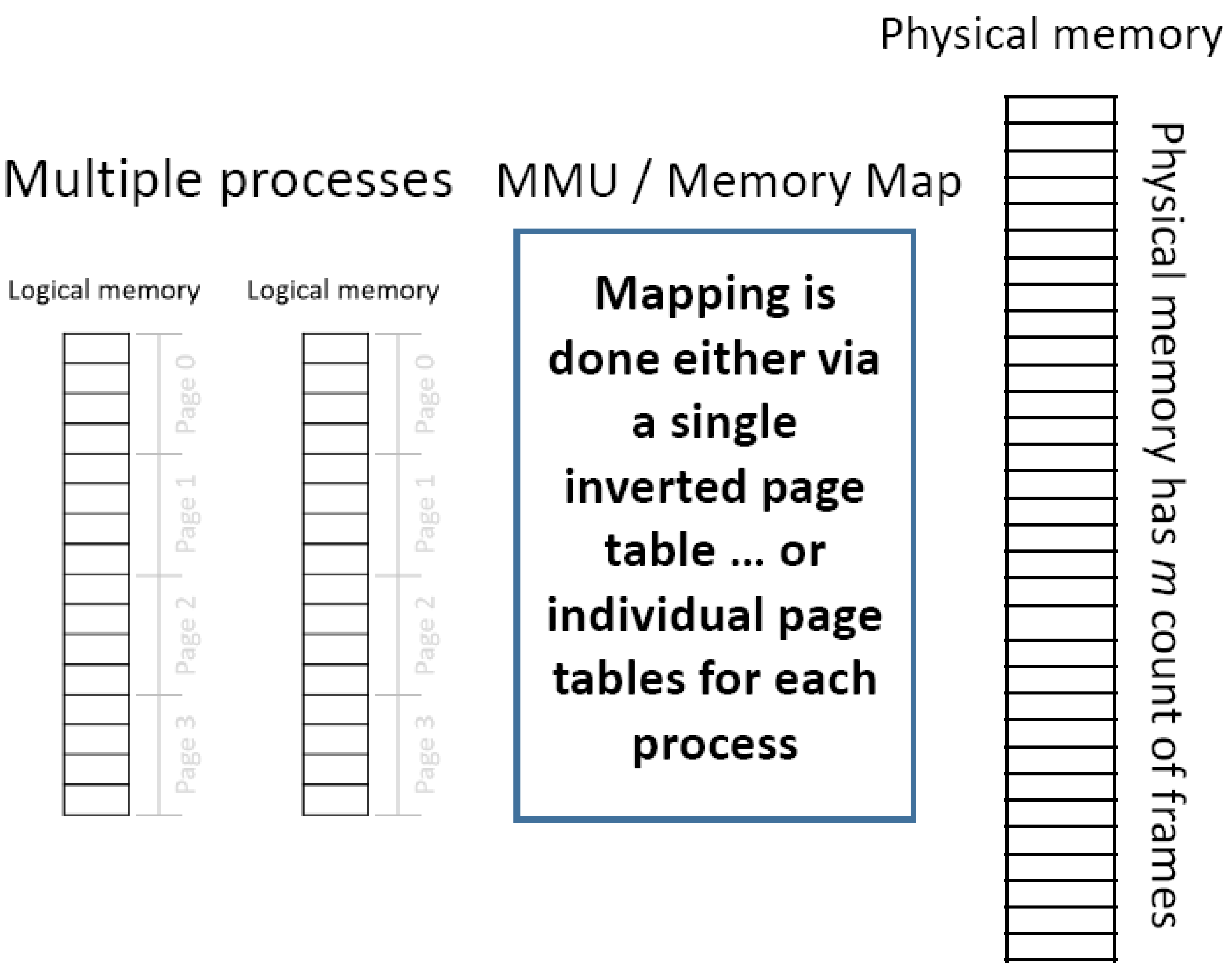


Mapping is done either via a single inverted page table … or individual page tables for each process

Physical memory has *m* count of frames

FRAME ALLOCATION

- How to allocate frames?
- Divide all frames among process equally?
- Allocate on demand?



Physical memory

Multiple processes

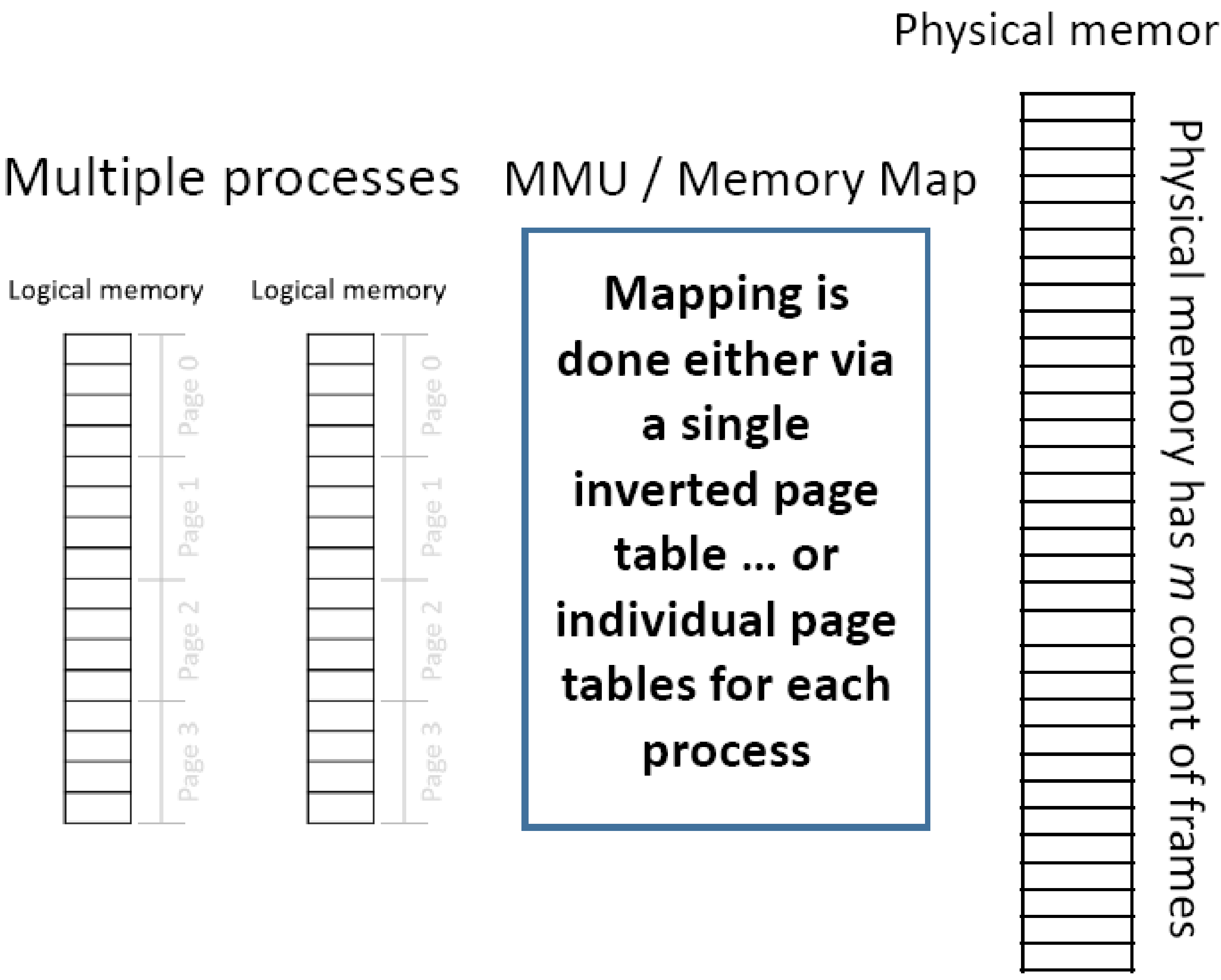
MMU / Memory Map

Mapping is done either via a single inverted page table … or individual page tables for each process

Physical memory has *m* count of frames

FRAME ALLOCATION

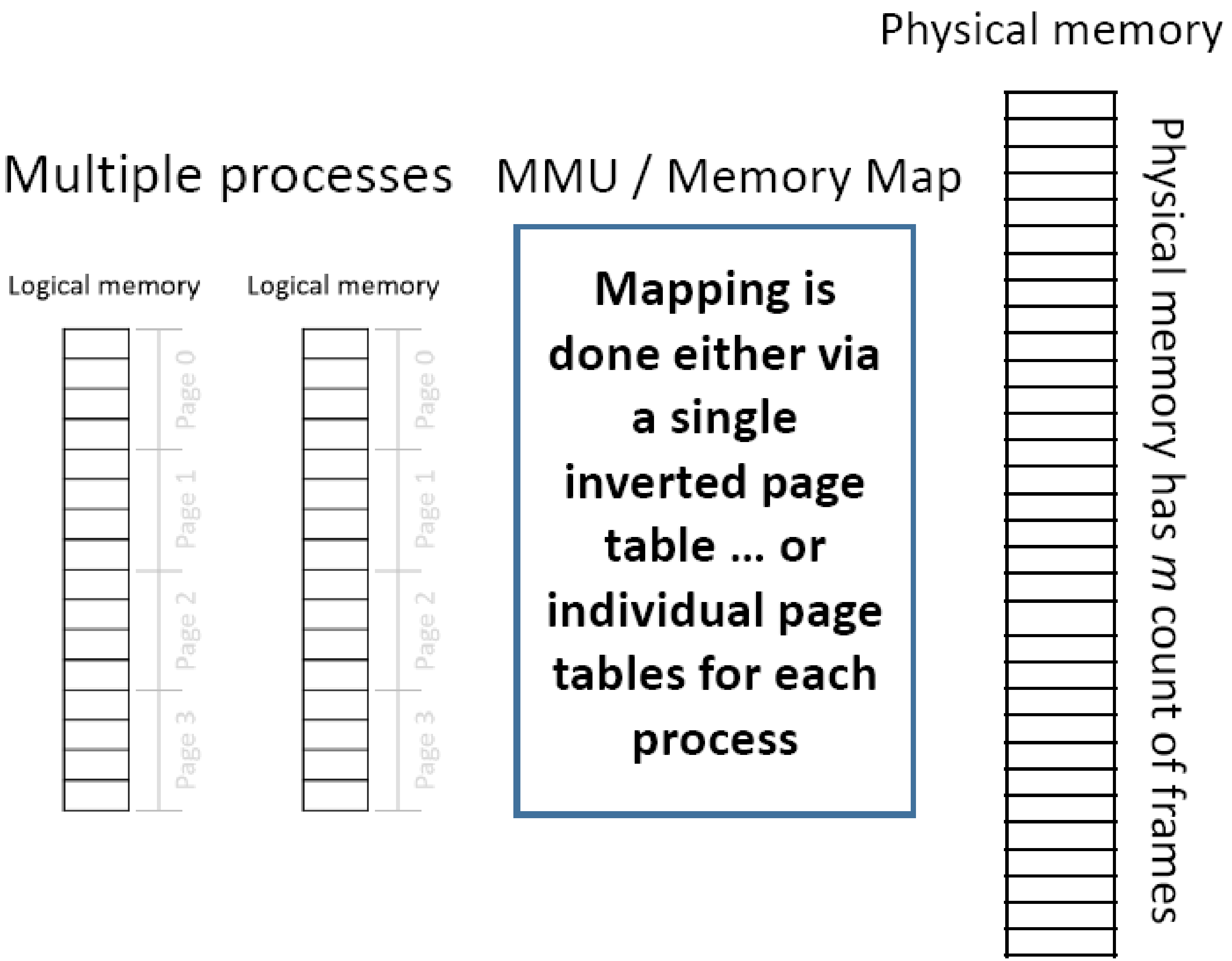
- As the number of frames allocated for each process decreases, page fault rate goes up.



Mapping is done either via a single inverted page table … or individual page tables for each process

FRAME ALLOCATION

- Q: What is the minimum number of frames to allocate for each process?



Mapping is done either via a single inverted page table … or individual page tables for each process

Physical memory has *m* count of frames

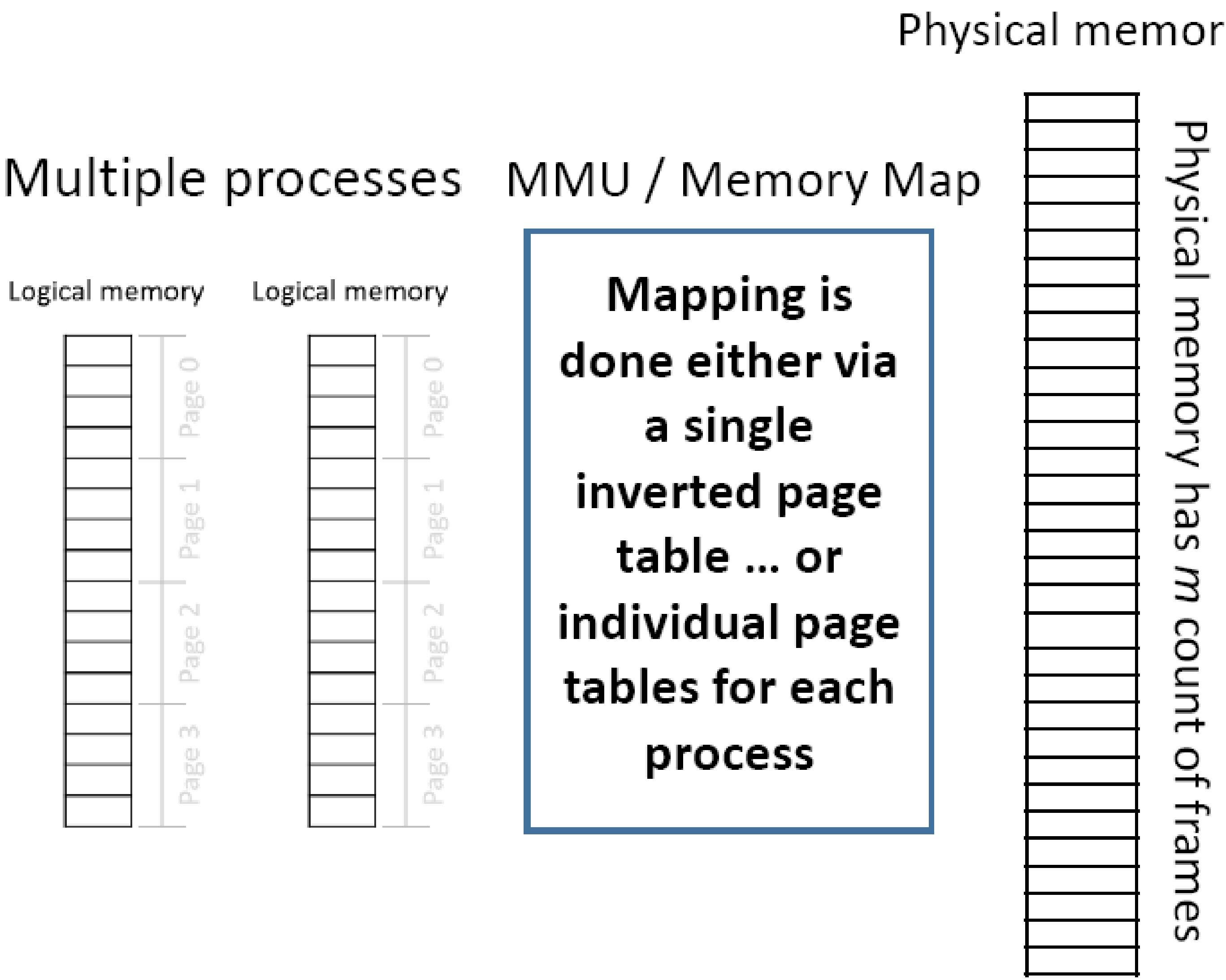
FRAME ALLOCATION

- Q: What is the minimum number of frames to allocate for each process?

Decided by architecture …

Q:How many memory references can occur in a single instruction?

- This can be a lot, 8 or even 16 in some architectures … why?
- Indirect addressing, where address can point to another address ..



Mapping is done either via a single inverted page table … or individual page tables for each process

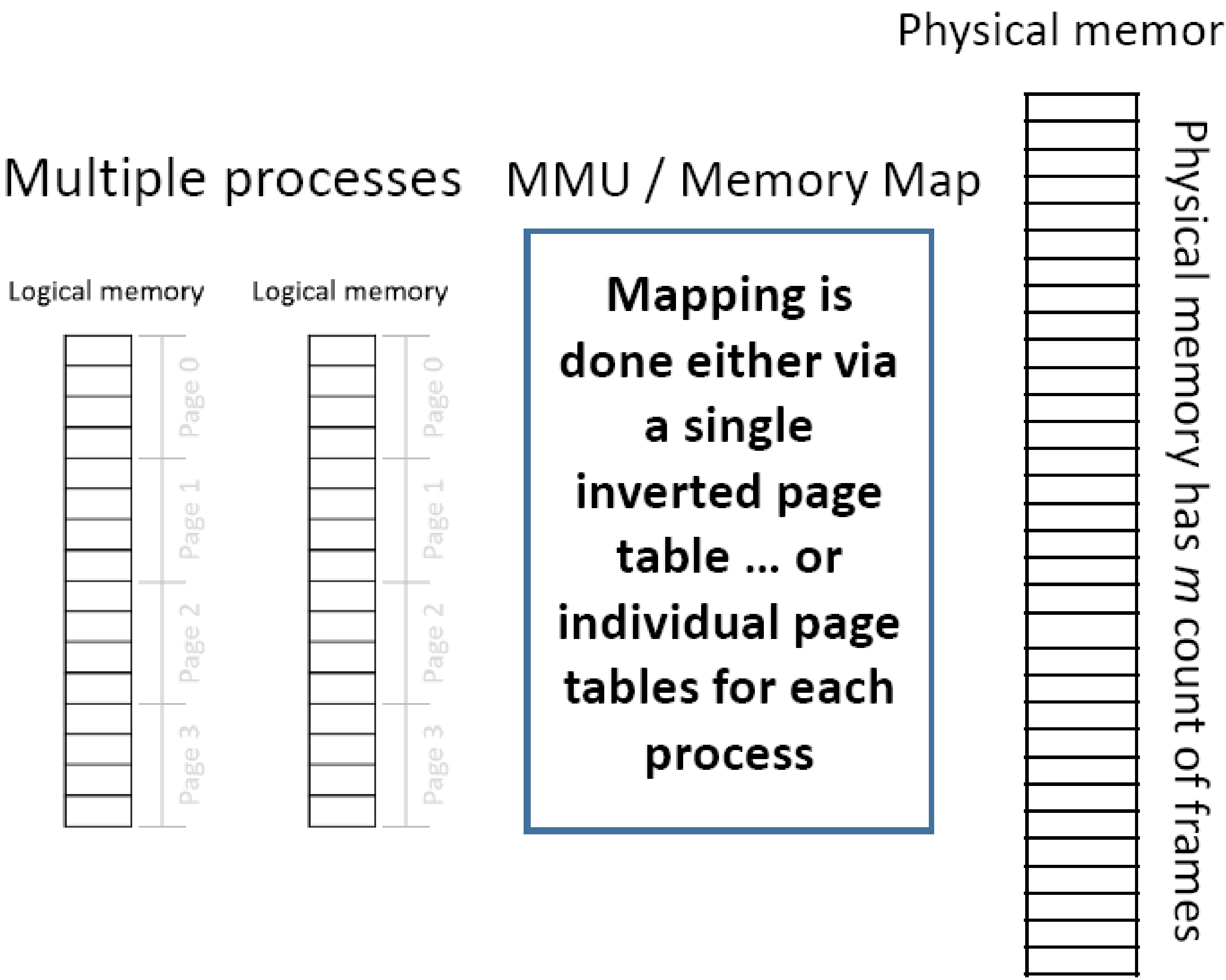
FRAME ALLOCATION

- Q: What is the minimum number of frames to allocate for each process?

Decided by architecture …

Q: How many memory references can occur in a single instruction?

- This can be a lot, 8 or even 16 in some architectures … why?
- Indirect addressing, where address can point to another address ..



Mapping is done either via a single inverted page table … or individual page tables for each process

It's simply the number of different memory locations that can be accessed in a *single* instruction.

EQUAL ALLOCATION

- Split the memory equally among all processes.
- Suppose we have 100 frames and 5 process … each would have 20 frames.
- Pretty simple …
- **Q: What are the drawbacks of this approach?**

EQUAL ALLOCATION

- Split the memory equally among all processes.
- Suppose we have 100 frames and 5 process … each would have 20 frames.
- Pretty simple …
- **Q: What are the drawbacks of this approach?**
  - Not all processes need the same amount of frames!
  - Some might need many others might need fewer pages … not very efficient use of memory

EQUAL ALLOCATION

- Split the memory equally among all processes.
- Suppose we have 100 frames and 5 process … each would have 20 frames.
- Pretty simple …
- **Q: What are the drawbacks of this approach?**
  - Not all processes need the same amount of frames!
  - Some might need many others might need fewer pages … not very efficient use of memory
  - Alternative: proportional memory allocation.

PROPORTIONAL MEMORY ALLOCATION

- Allocate frames proportional to process memory size.
- Process twice the size will get twice the number of frames.

Worksheet Q3: Any inefficiencies?

PROPORTIONAL MEMORY ALLOCATION

- Allocate frames proportional to process memory size.
- Process twice the size will get twice the number of frames.

Worksheet Q3: Any inefficiencies?

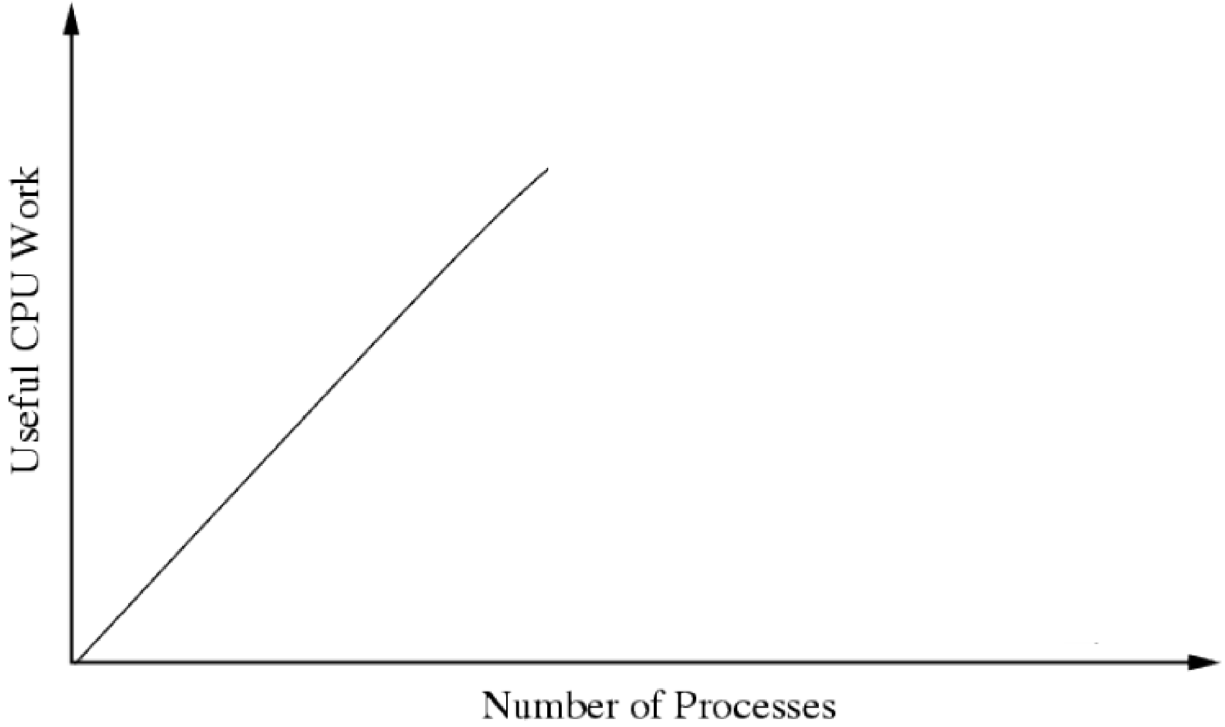
- Processes do not need all their pages in memory …
- The number of frames required is not always proportional to process size …
- The number for frames required *might vary* during run time.

THRASHING

- As we add more processes for multitasking we increase CPU utilization.

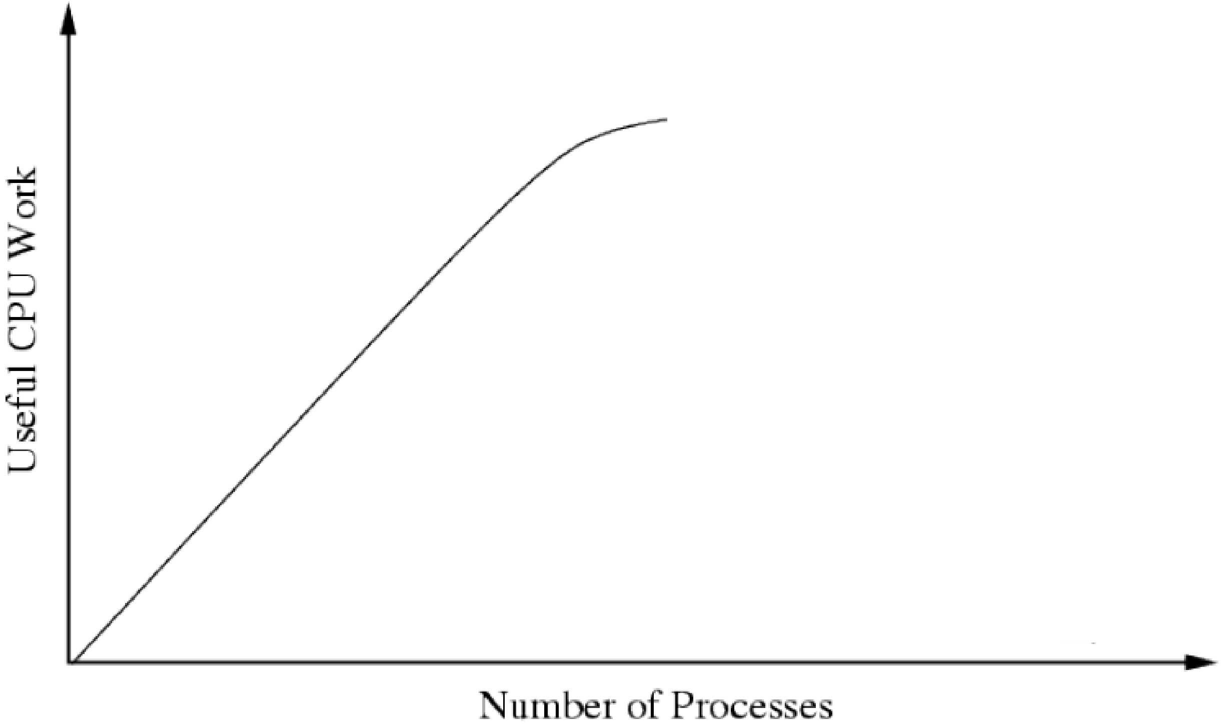
THRASHING

- As we add more processes for multitasking we increase CPU utilization.



THRASHING

- As we add more processes for multitasking we increase CPU utilization.
- We reach a point when adding more processes compete for frames in memory and page faults start to increase.



THRASHING

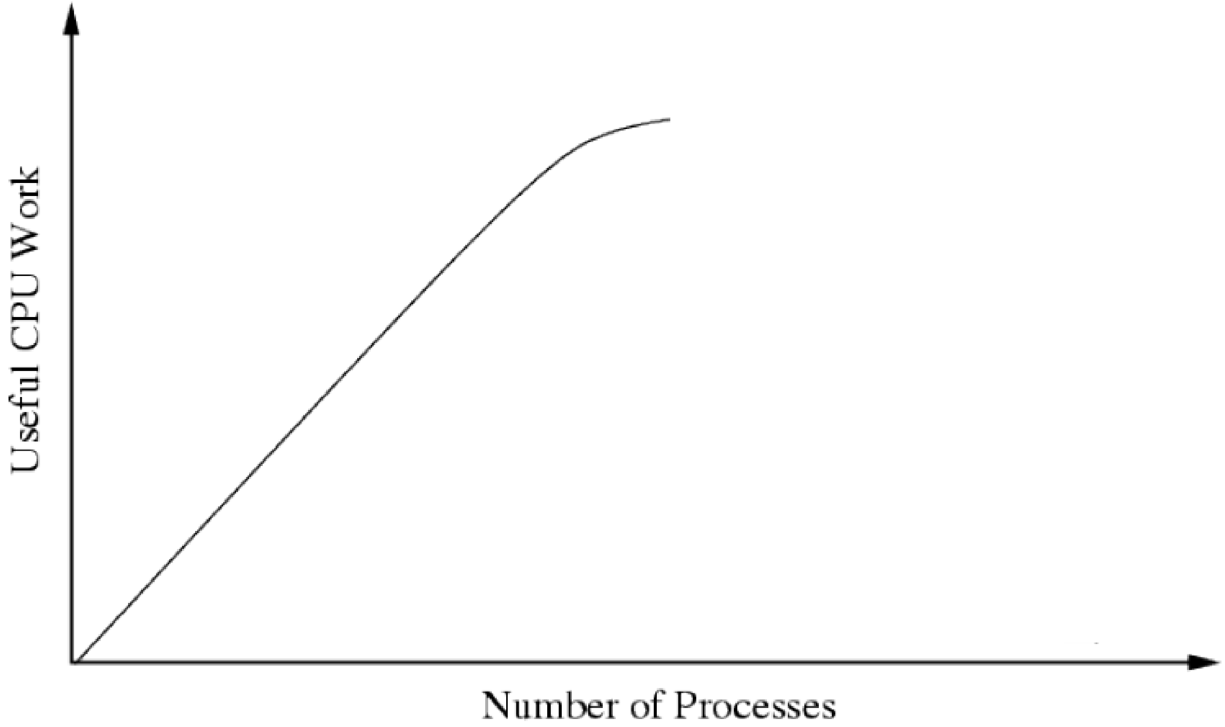
- As we add more processes for multitasking we increase CPU utilization.
- We reach a point when adding more processes compete for frames in memory and page faults start to increase.

Q:When page faults increase, what happens to CPU utilization

A B

A : Increases

B : Decreases



THRASHING

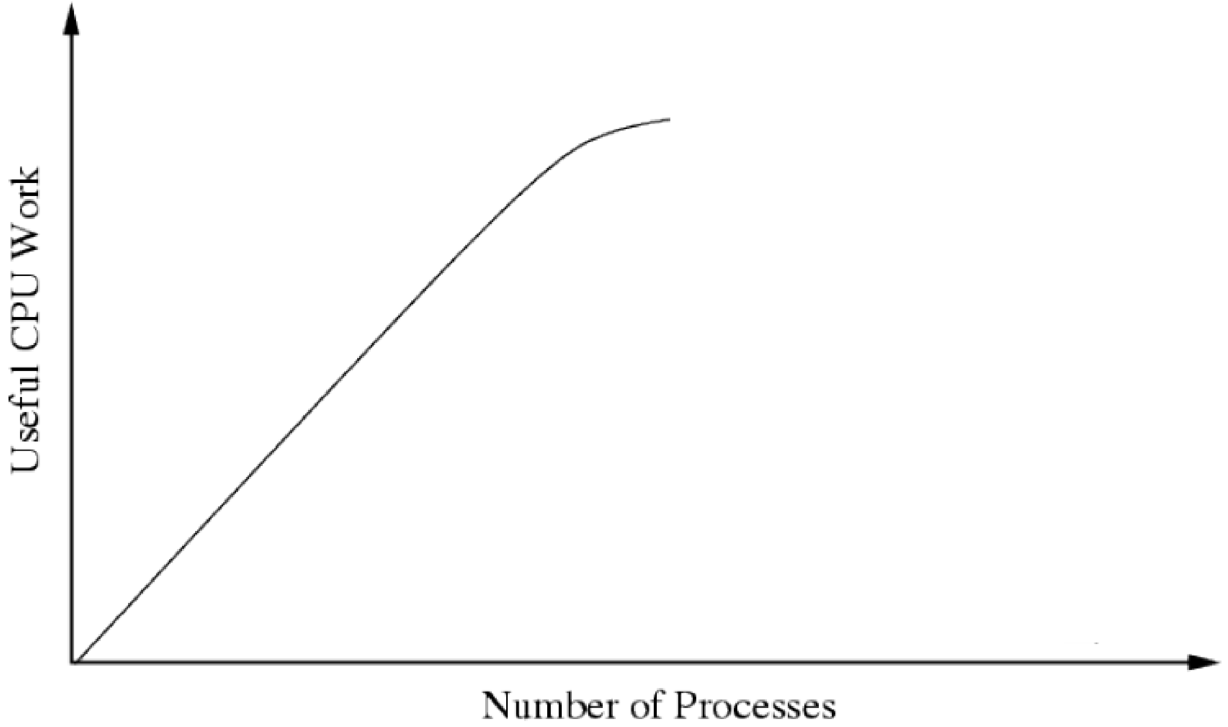
- As we add more processes for multitasking we increase CPU utilization.
- We reach a point when adding more processes compete for frames in memory and page faults start to increase.

Q:When page faults increase, what happens to CPU utilization

A B

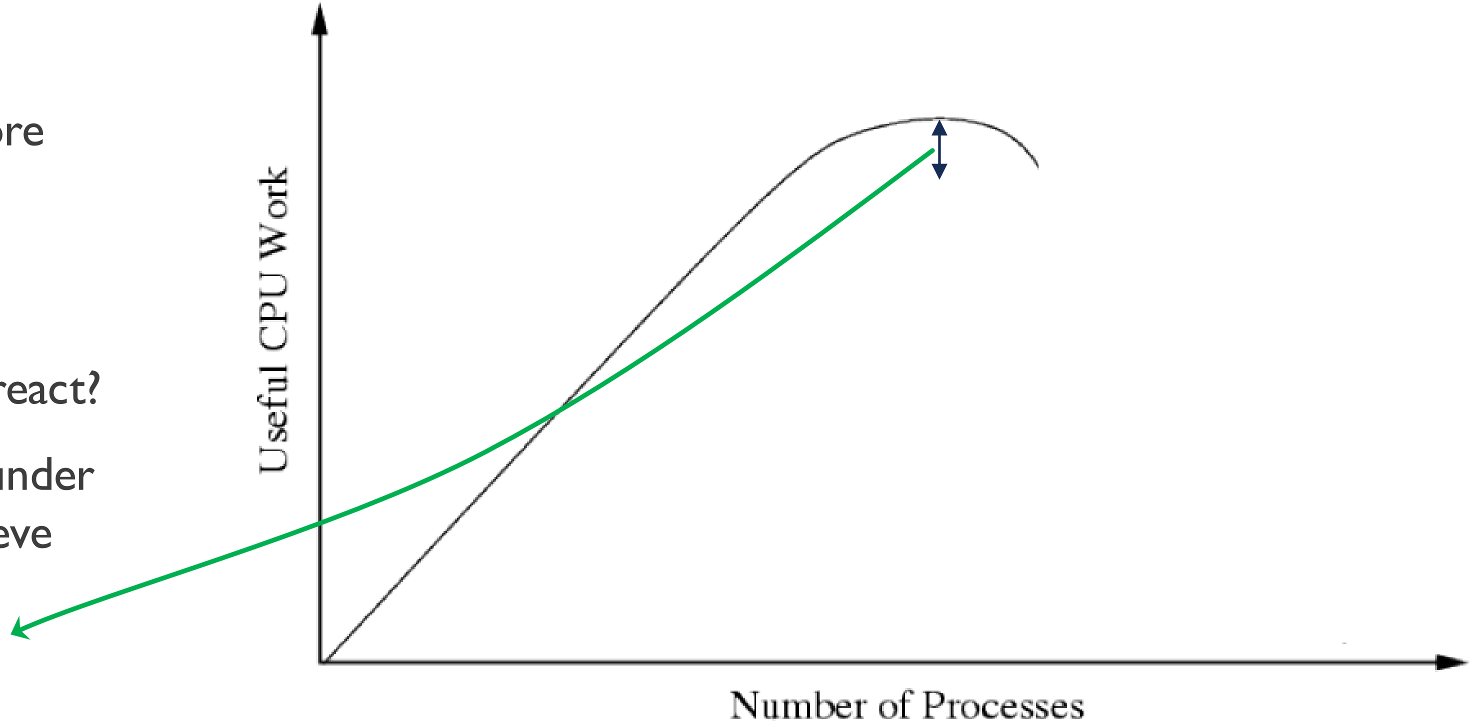
A : Increases

[✓ **B :** Decreases]



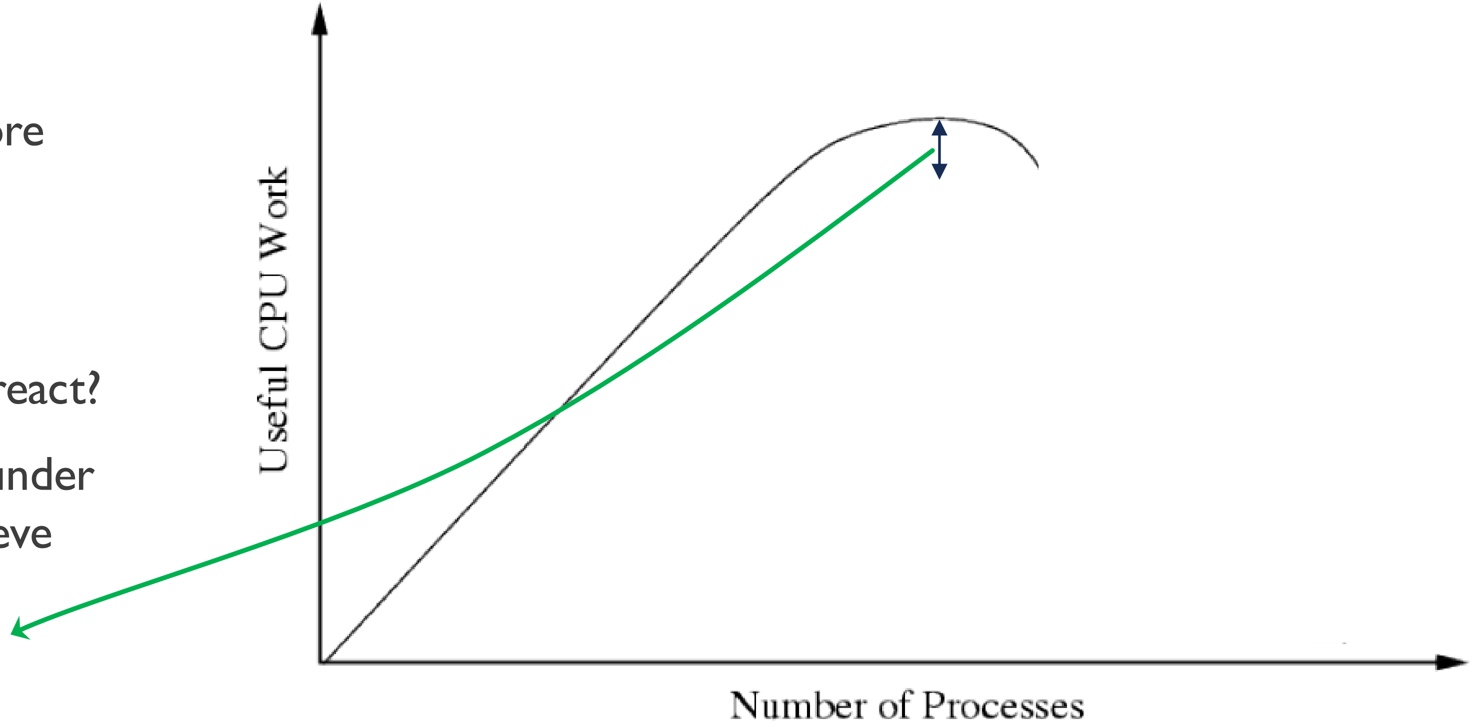
THRASHING

- As we add more processes for multitasking we increase CPU utilization.
- We reach a point when adding more processes compete for frames in memory and page faults start to increase.
- How does the Operating System react?
- The OS would think that CPU is under utilized, and as such it would retrieve more processes for concurrent execution.



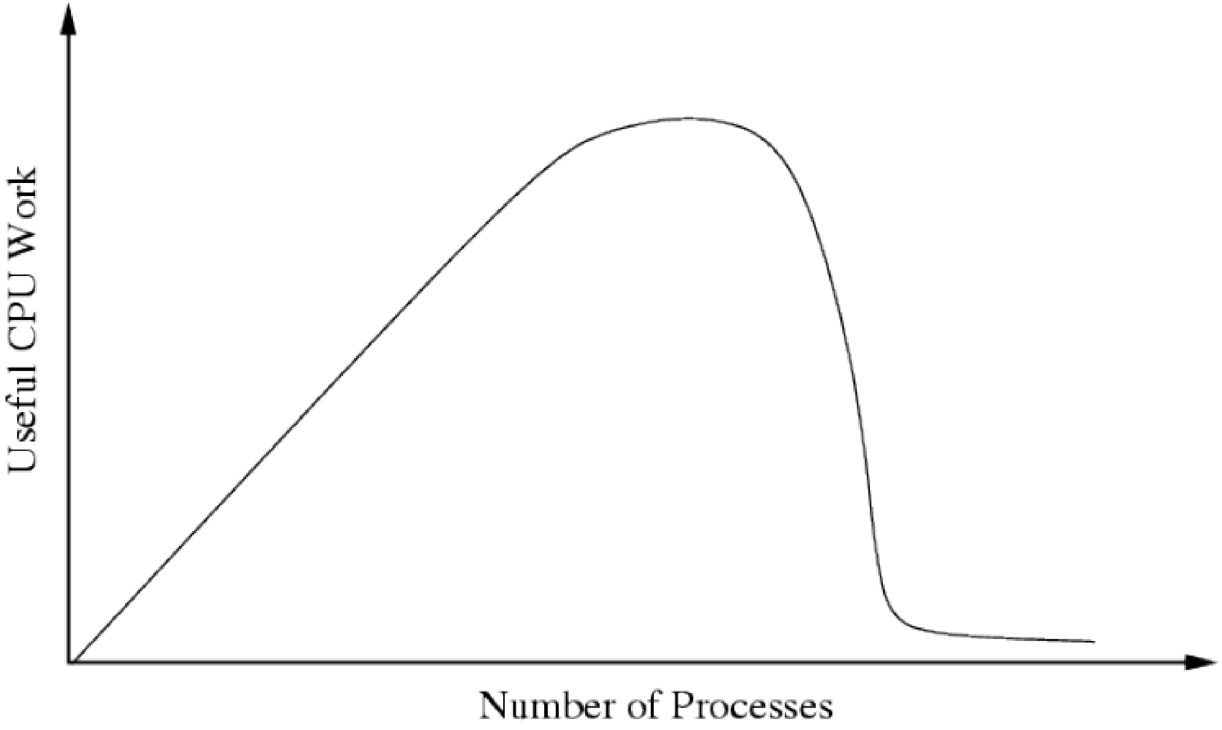
THRASHING

- As we add more processes for multitasking we increase CPU utilization.
- We reach a point when adding more processes compete for frames in memory and page faults start to increase.
- How does the Operating System react?
- The OS would think that CPU is under utilized, and as such it would retrieve more processes for concurrent execution.
- This causes even more page faults.



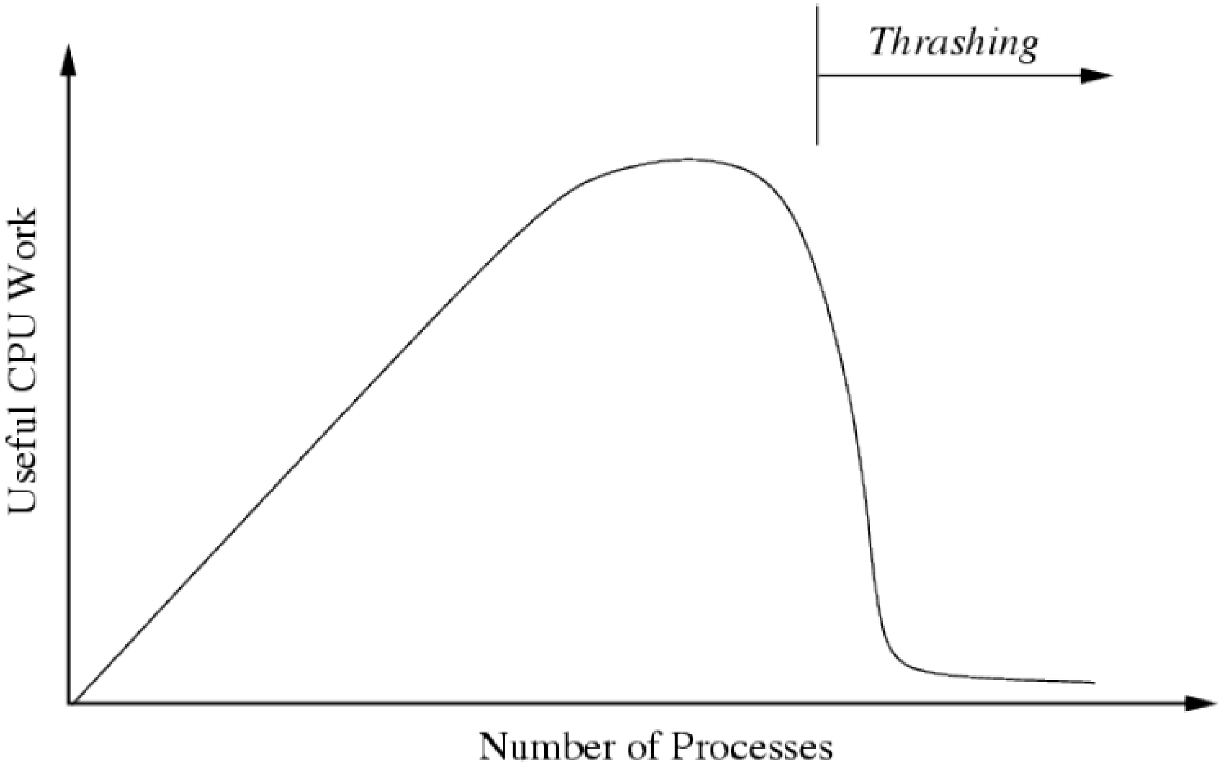
THRASHING

- As we add more processes for multitasking we increase CPU utilization.
- We reach a point when adding more processes compete for frames in memory and page faults start to increase.
- How does the Operating System react?
- The OS would think that CPU is under utilized, and as such it would retrieve more processes for concurrent execution.
- This causes even more page faults.



THRASHING

- As we add more processes for multitasking we increase CPU utilization.
- We reach a point when adding more processes compete for frames in memory and page faults start to increase.
- How does the Operating System react?
- The OS would think that CPU is under utilized, and as such it would retrieve more processes for concurrent execution.



HOW TO PREVENT THRASHING

- We can allow for only local frame replacement … a process can only replace frames it owns.

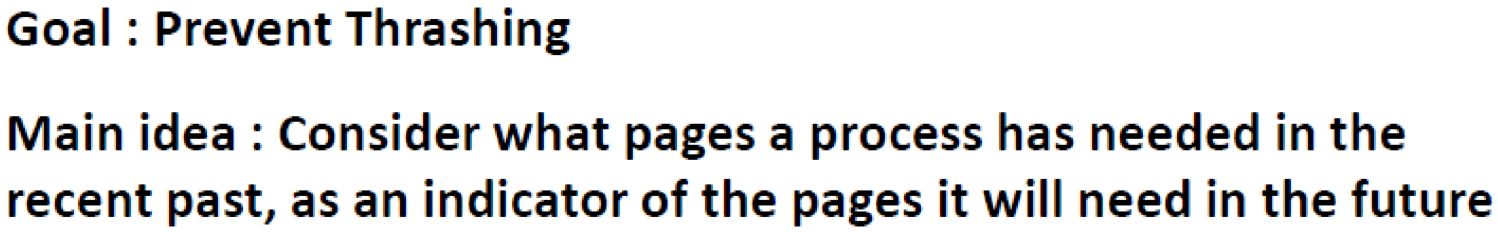
HOW TO PREVENT THRASHING

- We can allow for only local frame replacement … a process can only replace frames it owns.
- To truly prevent thrashing, we need to know how much frames each process needs for “healthy” execution.
- This is different than the minimum amount of frames that is dependent on architecture.

HOW TO PREVENT THRASHING

- We can allow for only local frame replacement … a process can only replace frames it owns.
- To truly prevent thrashing, we need to know how much frames each process needs for "healthy" execution.
- This is different than the minimum amount of frames that is dependent on architecture.
- How to know how much frames do we need for each process?

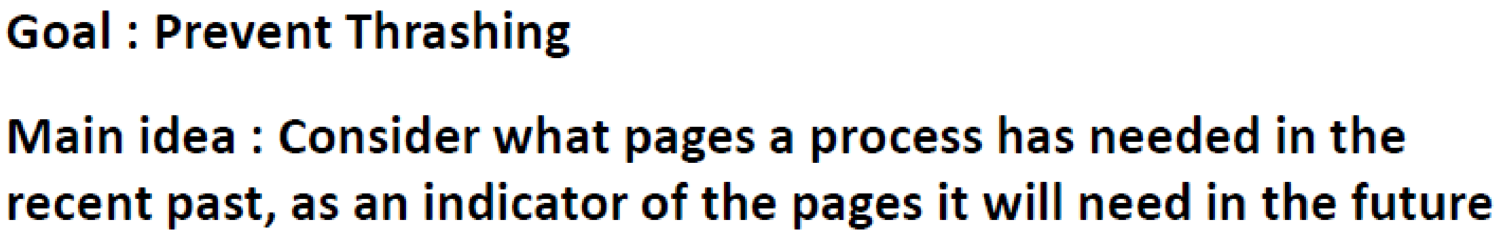
WORKING SET



Goal : Prevent Thrashing

Main idea : Consider what pages a process has needed in the recent past, as an indicator of the pages it will need in the future

WORKING SET

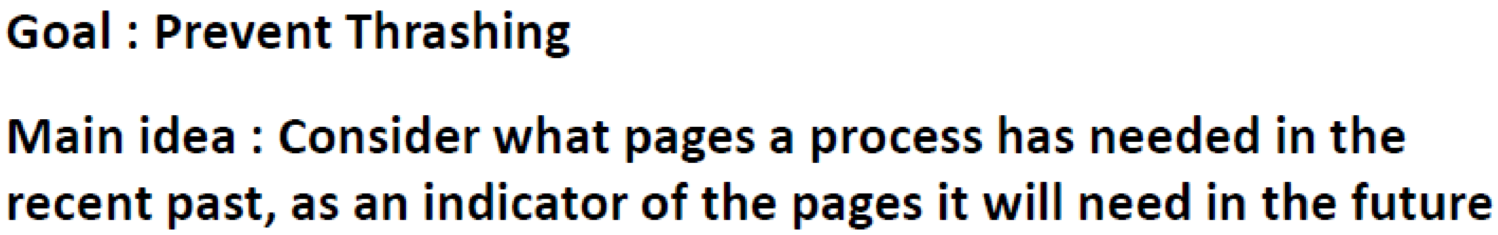


Goal : Prevent Thrashing

Main idea : Consider what pages a process has needed in the recent past, as an indicator of the pages it will need in the future

Working Set: a set of pages that are actively used together

WORKING SET



Goal : Prevent Thrashing

Main idea : Consider what pages a process has needed in the recent past, as an indicator of the pages it will need in the future

Working Set: a set of pages that are actively used together

Recent Memory References: $\Delta = 13$

WORKING SET

Goal : Prevent Thrashing

Main idea : Consider what pages a process has needed in the recent past, as an indicator of the pages it will need in the future

Working Set: a set of pages that are actively used together

Recent Memory References: $\Delta = 13$

1 2 4 5 6 4 2 5 6 5 1 2 6 8 2 4 5 6 7 2 3 3 3 2 2 3 3 3 2 3

WORKING SET

Goal : Prevent Thrashing

Main idea : Consider what pages a process has needed in the recent past, as an indicator of the pages it will need in the future

Working Set: a set of pages that are actively used together

Recent Memory References: $\Delta = 13$

1 2 4 5 6 4 2 5 6 5 1 2 6 8 2 4 5 6 7 2 3 3 3 2 2 3 3 3 2 3

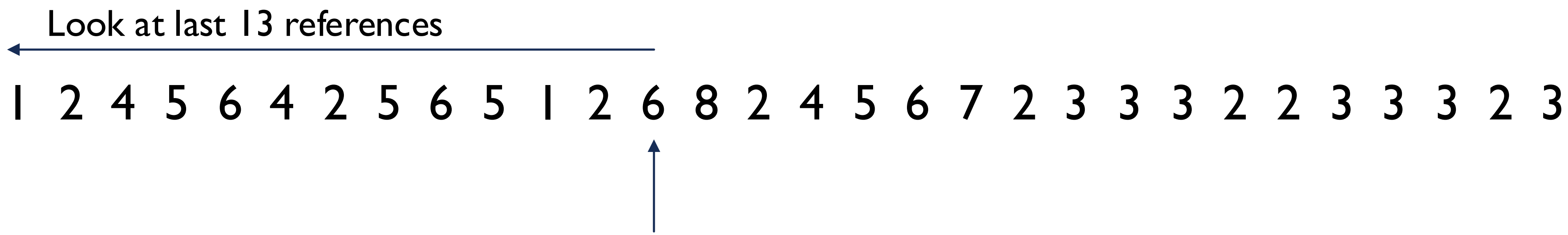
WORKING SET

Goal : Prevent Thrashing

Main idea : Consider what pages a process has needed in the recent past, as an indicator of the pages it will need in the future

Working Set: a set of pages that are actively used together

Recent Memory References: $\Delta = 13$



Look at last 13 references

1 2 4 5 6 4 2 5 6 5 1 2 6 8 2 4 5 6 7 2 3 3 3 2 2 3 3 3 2 3

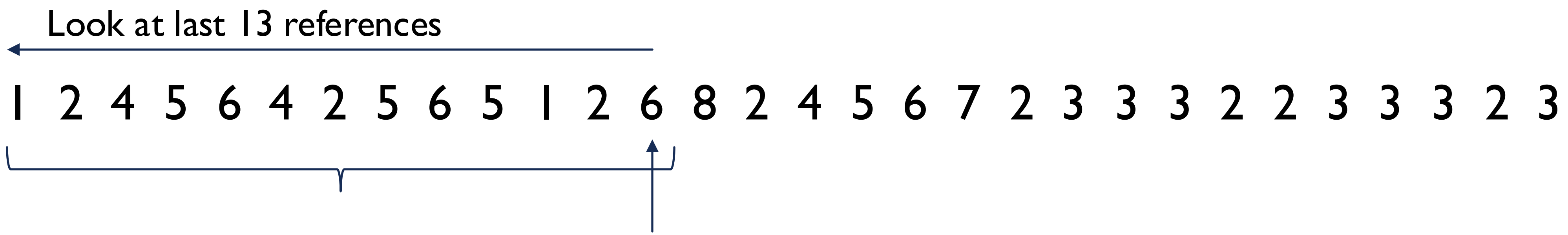
WORKING SET

Goal : Prevent Thrashing

Main idea : Consider what pages a process has needed in the recent past, as an indicator of the pages it will need in the future

Working Set: a set of pages that are actively used together

Recent Memory References: $\Delta = 13$



Look at last 13 references

1 2 4 5 6 4 2 5 6 5 1 2 6 8 2 4 5 6 7 2 3 3 3 2 2 3 3 3 2 3

{1,2,4,5,6} = Working Set

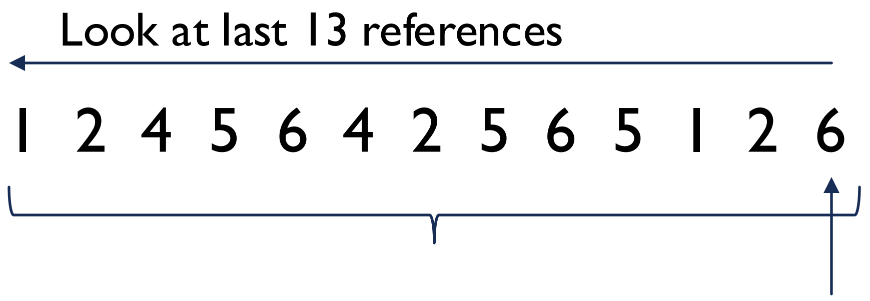
WORKING SET

Goal : Prevent Thrashing

Main idea : Consider what pages a process has needed in the recent past, as an indicator of the pages it will need in the future

Working Set: a set of pages that are actively used together

Recent Memory References: $\Delta = 13$



Look at last 13 references

1 2 4 5 6 4 2 5 6 5 1 2 6 8 2 4 5 6 7 2 3 3 3 2 2 3 3 3 2 3

{1,2,4,5,6} = Working Set
Working Set Size = 5

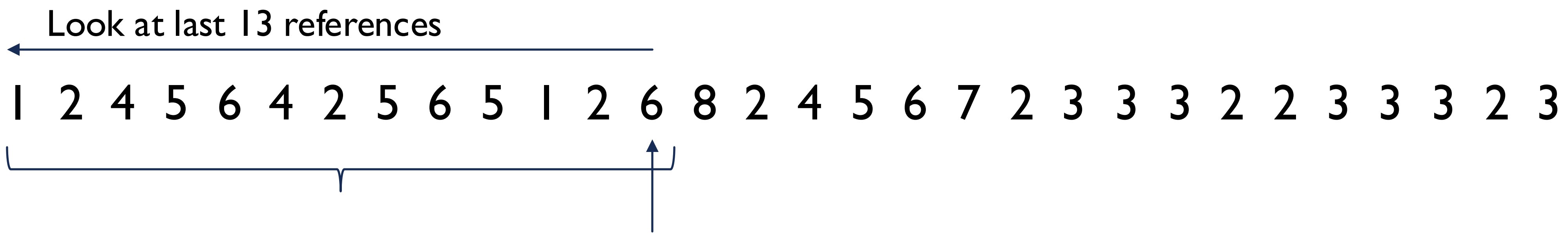
WORKING SET

Goal : Prevent Thrashing

Main idea : Consider what pages a process has needed in the recent past, as an indicator of the pages it will need in the future

Working Set: a set of pages that are actively used together

Recent Memory References: $\Delta = 13$



Look at last 13 references

1 2 4 5 6 4 2 5 6 5 1 2 6 8 2 4 5 6 7 2 3 3 3 2 2 3 3 3 2 3

{1,2,4,5,6} = Working Set
Working Set Size = 5
OS allocates 5 frames for P1

WORKING SET

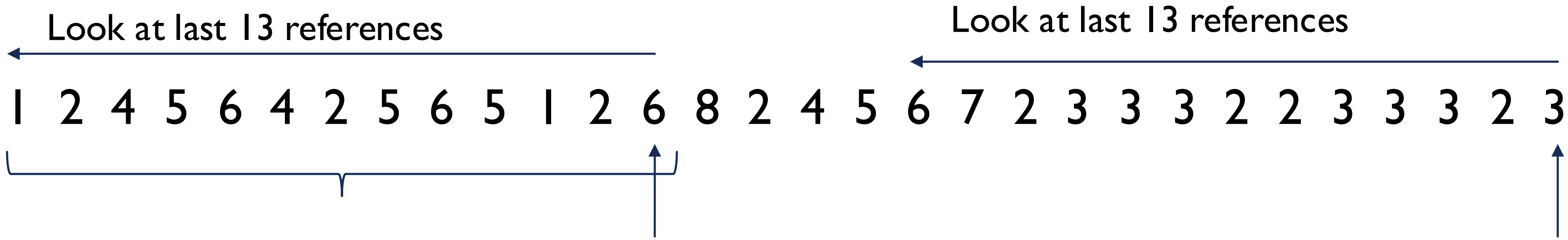
Goal : Prevent Thrashing

Main idea : Consider what pages a process has needed in the recent past, as an indicator of the pages it will need in the future

Working Set: a set of pages that are actively used together

Recent Memory References: $\Delta = 13$

Worksheet Q4



Look at last 13 references

Look at last 13 references

1 2 4 5 6 4 2 5 6 5 1 2 6 8 2 4 5 6 7 2 3 3 3 2 2 3 3 3 2 3

{1,2,4,5,6} = Working Set
Working Set Size = 5
OS allocates 5 frames for P1

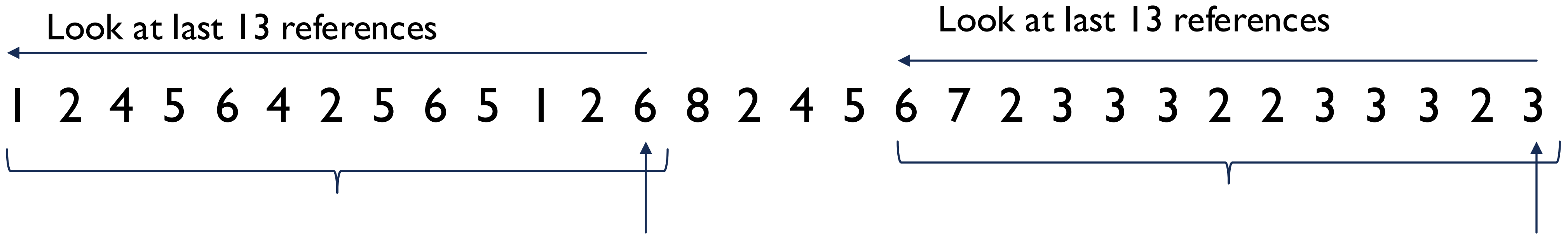
WORKING SET

Goal : Prevent Thrashing

Main idea : Consider what pages a process has needed in the recent past, as an indicator of the pages it will need in the future

Working Set: a set of pages that are actively used together

Recent Memory References: $\Delta = 13$



Look at last 13 references

Look at last 13 references

1 2 4 5 6 4 2 5 6 5 1 2 6 8 2 4 5 6 7 2 3 3 3 2 2 3 3 3 2 3

{1,2,4,5,6} = Working Set
Working Set Size = 5
OS allocates 5 frames for P1

{2,3,6,7} = Working Set

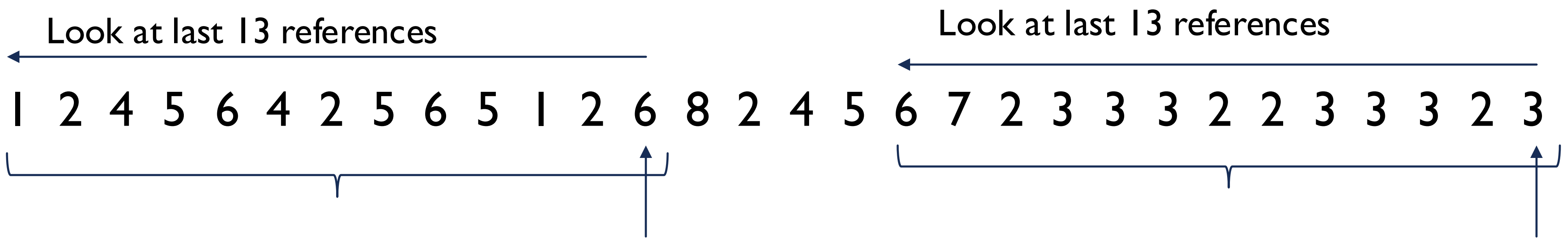
WORKING SET

Goal : Prevent Thrashing

Main idea : Consider what pages a process has needed in the recent past, as an indicator of the pages it will need in the future

Working Set: a set of pages that are actively used together

Recent Memory References: $\Delta = 13$



Look at last 13 references

Look at last 13 references

1 2 4 5 6 4 2 5 6 5 1 2 6 8 2 4 5 6 7 2 3 3 3 2 2 3 3 3 2 3

{1,2,4,5,6} = Working Set
Working Set Size = 5
OS allocates 5 frames for P1

{2,3,6,7} = Working Set
Working Set Size = 4

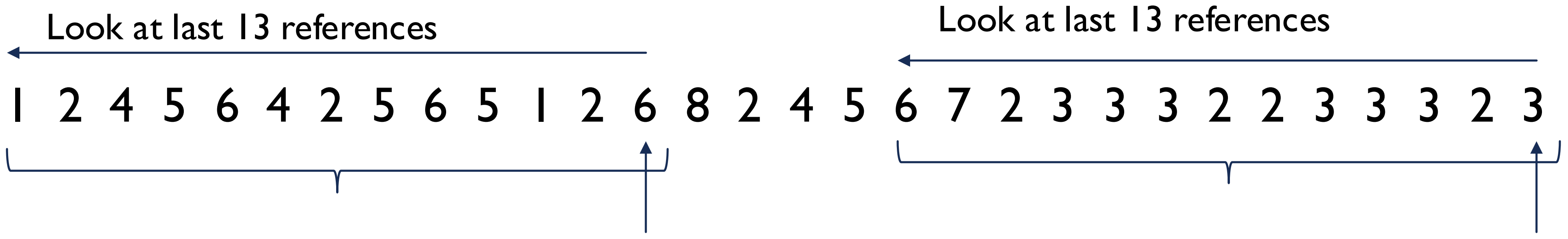
WORKING SET

Goal : Prevent Thrashing

Main idea : Consider what pages a process has needed in the recent past, as an indicator of the pages it will need in the future

Working Set: a set of pages that are actively used together

Recent Memory References: $\Delta = 13$



Look at last 13 references

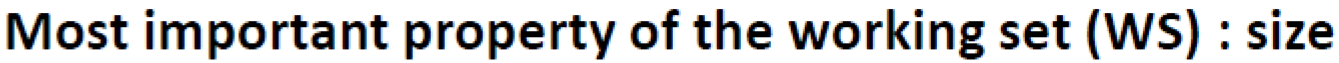
Look at last 13 references

1 2 4 5 6 4 2 5 6 5 1 2 6 8 2 4 5 6 7 2 3 3 3 2 2 3 3 3 2 3

{1,2,4,5,6} = Working Set
Working Set Size = 5
OS allocates 5 frames for P1

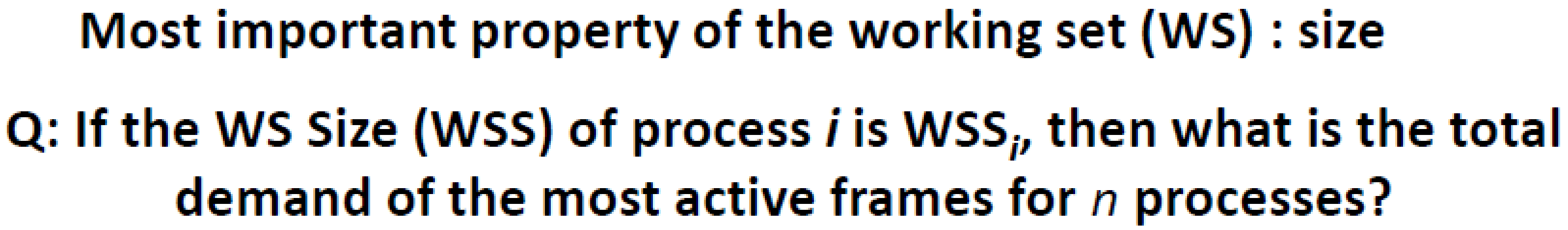
{2,3,6,7} = Working Set
Working Set Size = 4
OS allocates 4 frames for P1

WORKING SIZE SET AND THRASHING



Most important property of the working set (WS) : size

WORKING SIZE SET AND THRASHING



Most important property of the working set (WS) : size

Q: If the WS Size (WSS) of process *i* is WSS$_i$, then what is the total demand of the most active frames for *n* processes?

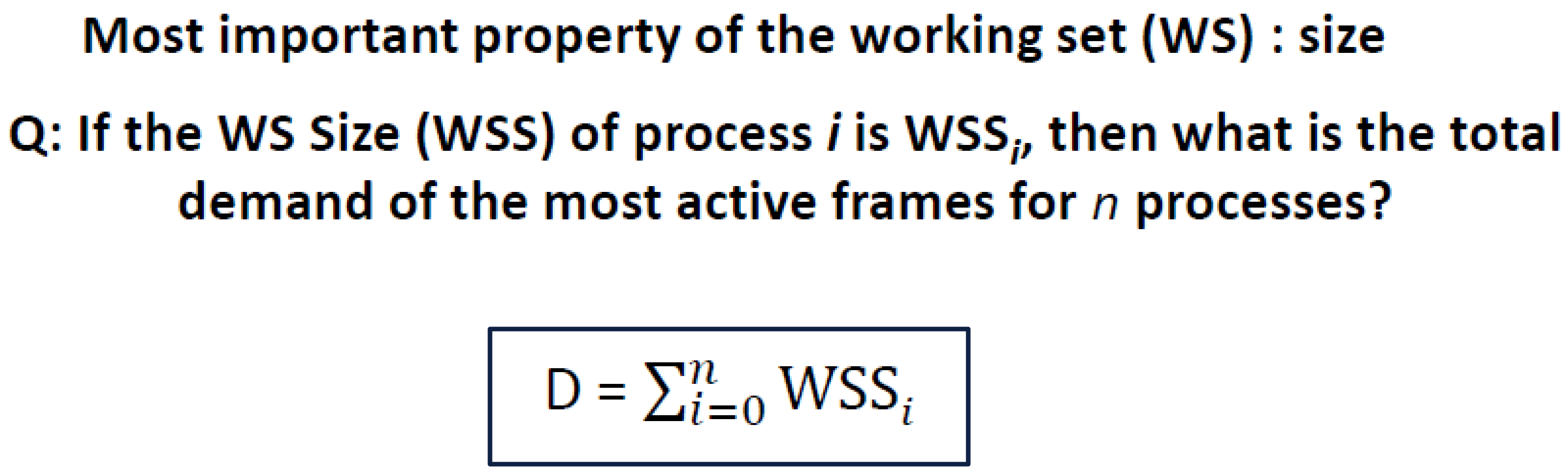
WORKING SIZE SET AND THRASHING

Most important property of the working set (WS) : size

Q: If the WS Size (WSS) of process *i* is WSS_i, then what is the total demand of the most active frames for *n* processes?

$$\boxed{\mathrm{D} = \sum_{i=0}^{n} \mathrm{WSS}_i}$$

WORKING SIZE SET AND THRASHING



Most important property of the working set (WS) : size

Q: If the WS Size (WSS) of process *i* is WSS$_i$, then what is the total demand of the most active frames for *n* processes?

$$\boxed{D = \sum_{i=0}^{n} \mathrm{WSS}_i}$$

If the total demand D is greater than the count of available frames, thrashing will occur.

WORKING SIZE SET AND THRASHING

Most important property of the working set (WS) : size

Q: If the WS Size (WSS) of process *i* is WSS_i, then what is the total demand of the most active frames for *n* processes?

$$\boxed{\text{D} = \sum_{i=0}^{n} \text{WSS}_i}$$

If the total demand D is greater than the count of available frames, thrashing will occur.

Q: What can the OS do?

WORKING SIZE SET AND THRASHING

Most important property of the working set (WS) : size

Q: If the WS Size (WSS) of process *i* is WSS$_i$, then what is the total demand of the most active frames for *n* processes?

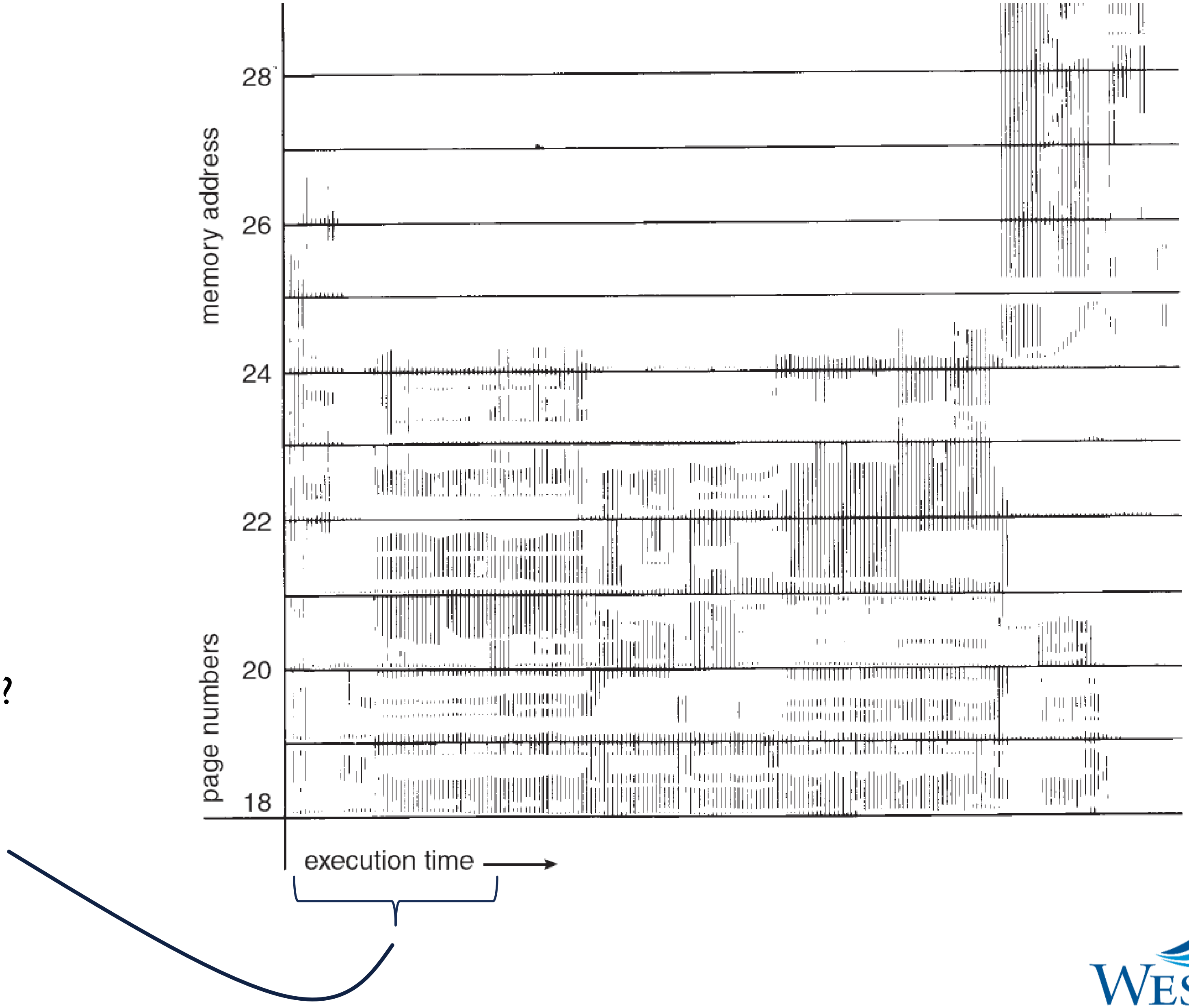
$$\boxed{D = \sum_{i=0}^{n} \mathrm{WSS}_i}$$

If the total demand D is greater than the count of available frames, thrashing will occur.

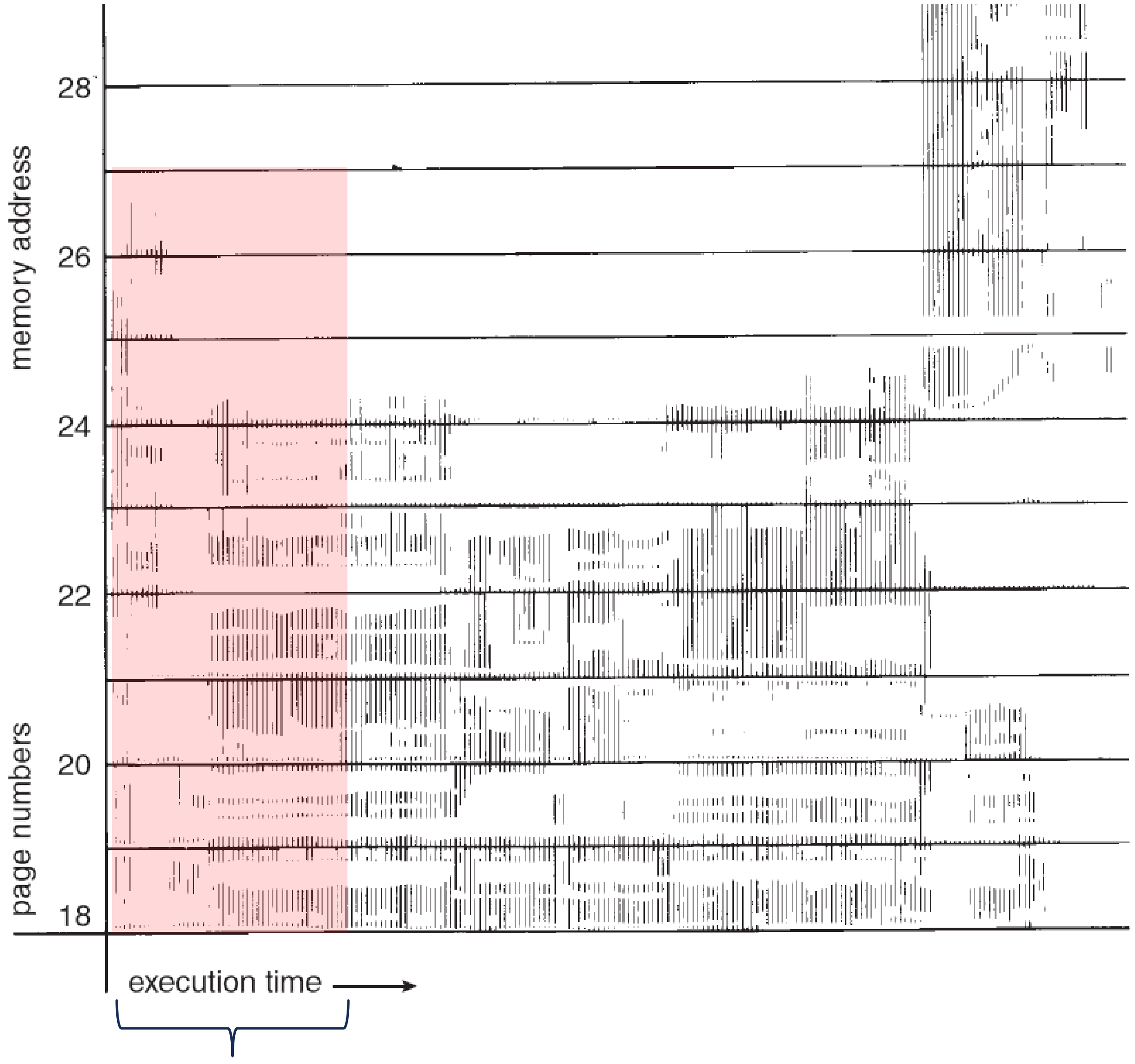
The OS monitors WSS for each process, if D is approaching the memory limit, suspend a process.

WORKING SET

Q:What is the working set size?



WORKING SET

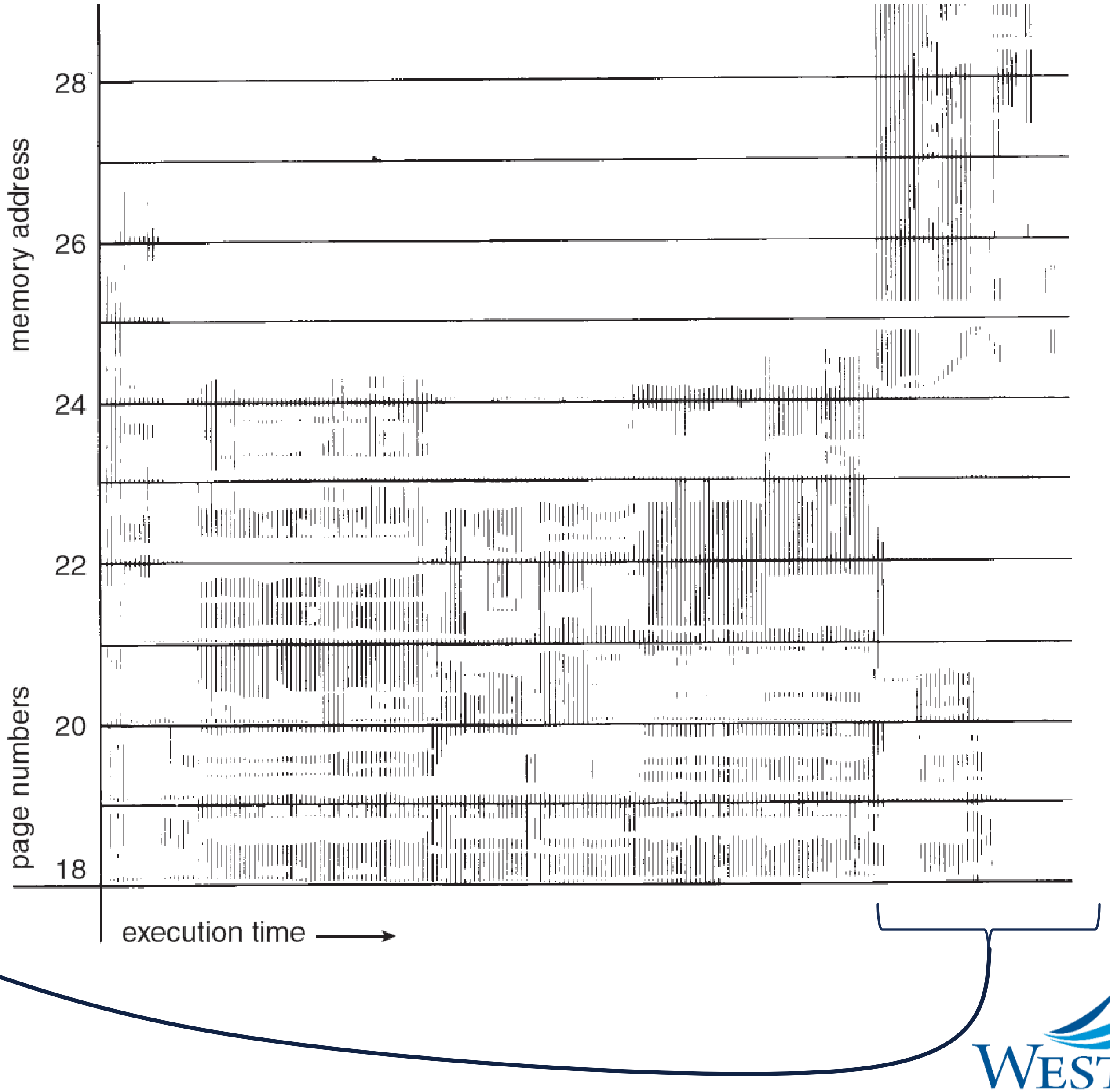


Q:What is the working set size?

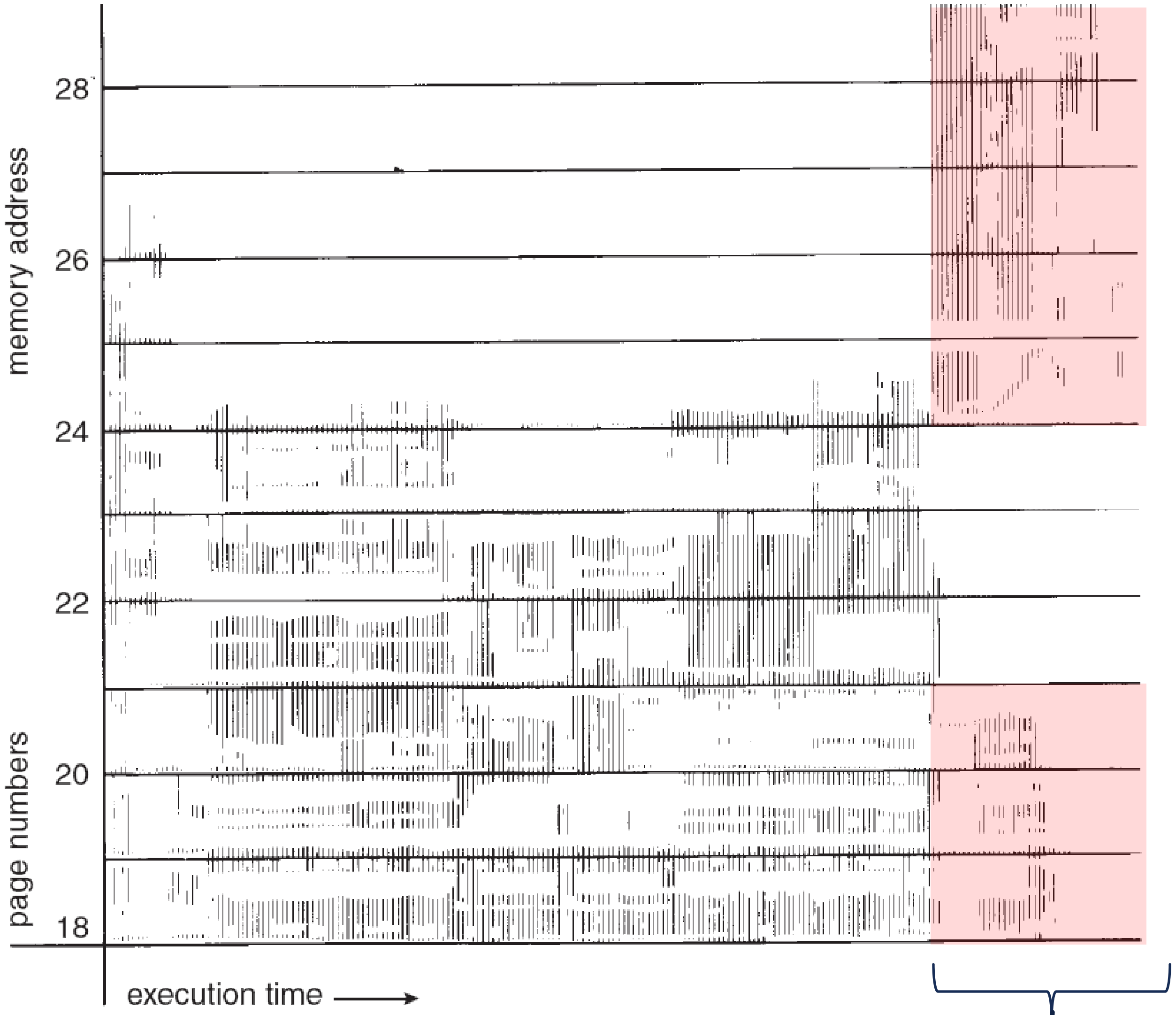
Pages 18 to 26
WSS = 9

WORKING SET

Q:What is the working set size?



WORKING SET



Q:What is the working set size?

Pages 18 to 20 and 24 to 28
WSS = 8