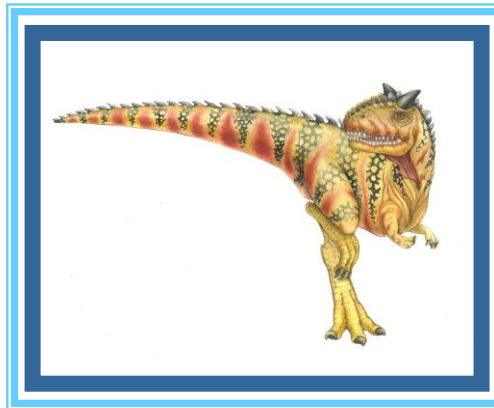
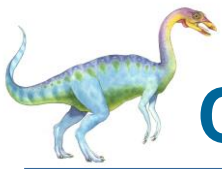


# Chapter 8: Virtual-Memory Management

---



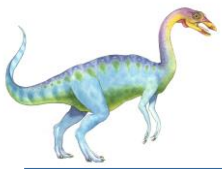


# Chapter 8: Virtual-Memory Management

---

- Background
- Demand Paging
- Page Replacement
- Allocation of Frames





# Objectives

---

- To describe the benefits of a virtual memory system
- To explain the concepts of demand paging, page-replacement algorithms, and allocation of page frames





# Background

---

- Code needs to be in memory to execute, but entire program rarely used
  - Error code, unusual routines, large data structures
- Entire program code not needed at same time
- Consider ability to execute partially-loaded program
  - Program no longer constrained by limits of physical memory
  - Each program takes less memory while running -> more programs run at the same time
    - ▶ Increased CPU utilization and throughput with no increase in response time or turnaround time
  - Less I/O needed to load or swap programs into memory -> each user program runs faster





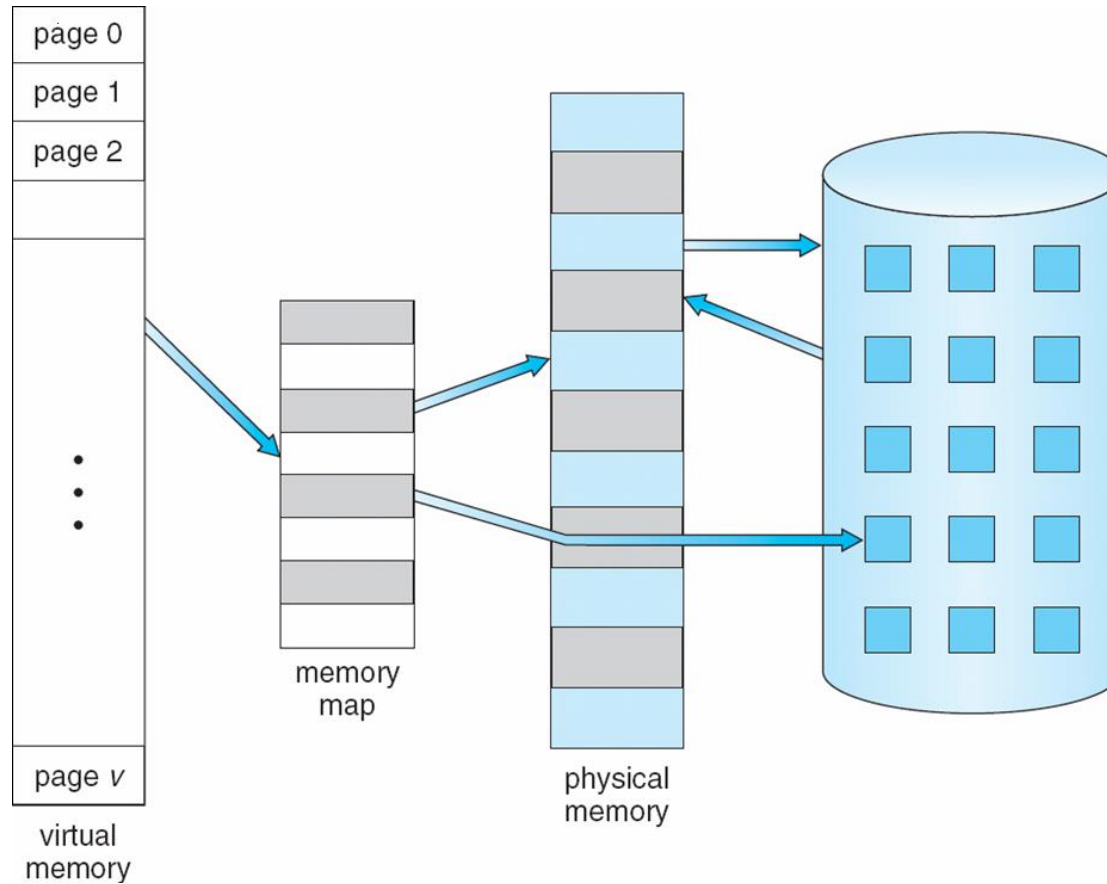
# Background

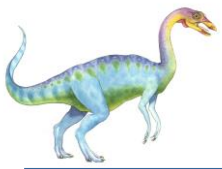
- **Virtual memory** – separation of user logical memory from physical memory.
  - Only part of the program needs to be in memory for execution
  - Logical address space can therefore be much larger than physical address space
  - Allows address spaces to be shared by several processes
  - Allows for more efficient process creation
  - More programs running concurrently
  - Less I/O needed to load or swap processes
  
- **Virtual address space** – logical view of how process is stored in memory
  - Usually start at address 0, contiguous addresses until end of space
  - Meanwhile, physical memory organized in page frames
  - MMU must map logical to physical
  
- Virtual memory can be implemented via:
  - Demand paging
  - Demand segmentation



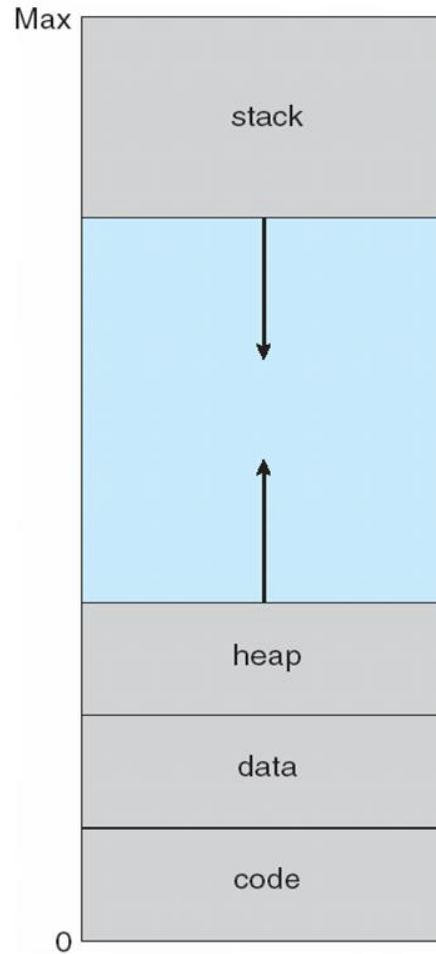


# Virtual Memory That is Larger Than Physical Memory



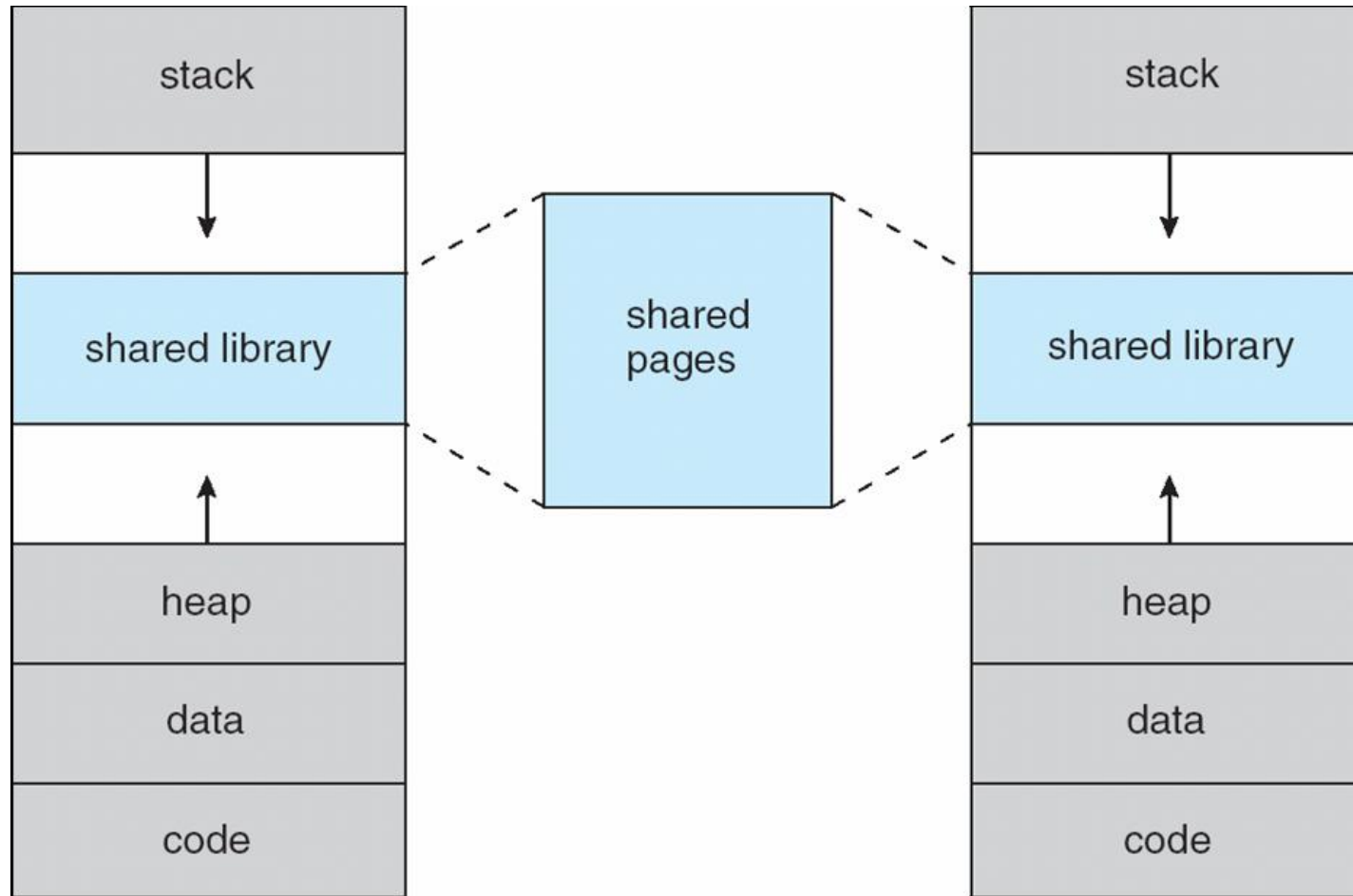


# Virtual-address Space





# Shared Library Using Virtual Memory





# Demand Paging

(การจัดสรร **Paging** ตามความต้องการที่ร้องขอ)

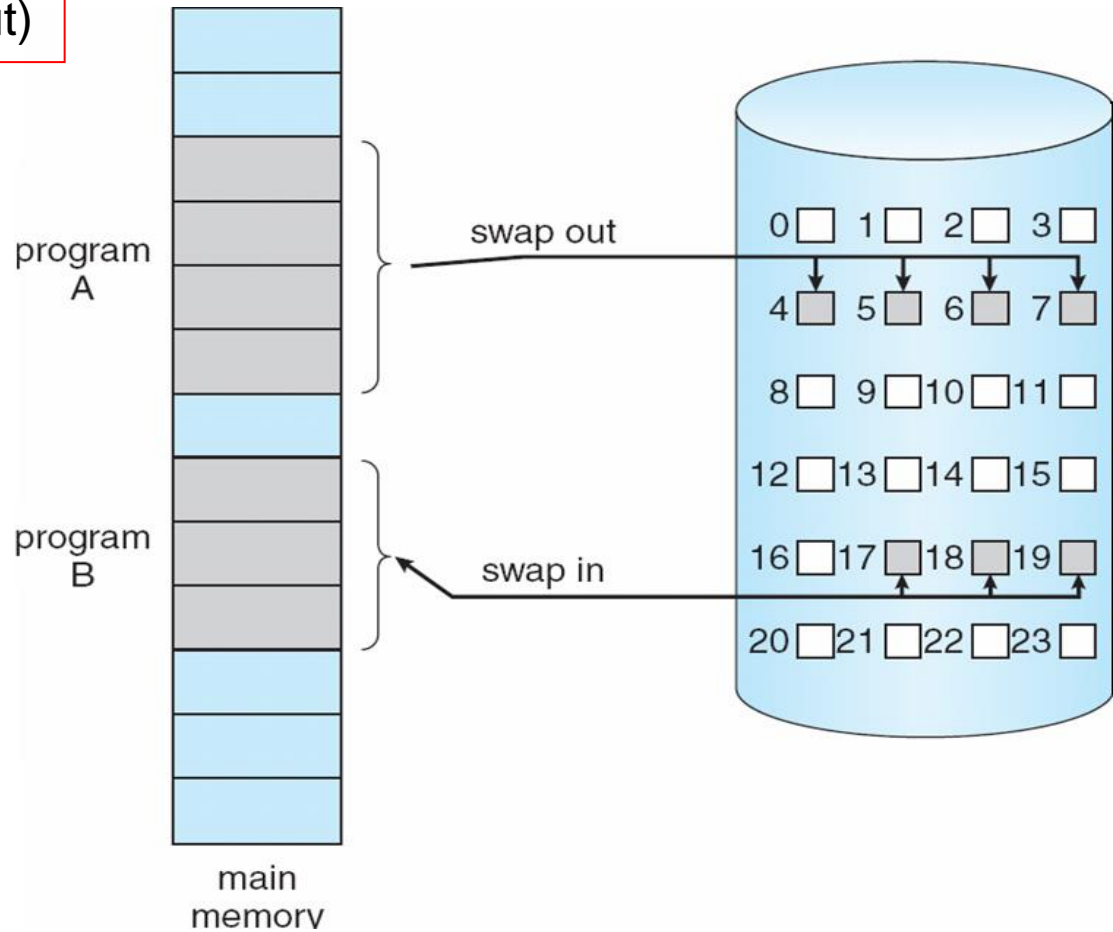
- ❑ Bring a page into memory **only when it is needed**
  - ❑ Less I/O needed
  - ❑ Less memory needed
  - ❑ Faster response
  - ❑ More users
- ❑ Page is needed  $\Rightarrow$  reference to it (หากต้องการใช้ page ให้อ้างตำแหน่งถึง page ที่ต้องการ)
  - ❑ invalid reference  $\Rightarrow$  abort (หากอ้างตำแหน่งไม่ถูกต้องให้ยกเลิก)
  - ❑ not-in-memory  $\Rightarrow$  bring to memory (หากอ้างแล้วไม่มีใน memory ให้นำเข้ามาไว้ใน memory)
- ❑ **Lazy swapper** – never swaps a page into memory unless page will be needed
  - ❑ Swapper that deals with pages is a **pager**

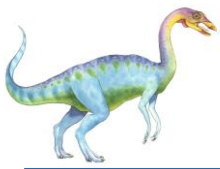




# Transfer of a Paged Memory to Contiguous Disk Space

มี pager เป็นตัวย้าย page  
เข้า (swap in) หรือออก (swap out)





# Valid-Invalid Bit

- With each page table entry a valid–invalid bit is associated (**v** ⇒ in-memory, **i** ⇒ not-in-memory)
- Initially valid–invalid bit is set to **i** on all entries
- Example of a page table snapshot:

Frame #	valid-invalid bit
	<b>v</b>
	<b>v</b>
	<b>v</b>
	<b>v</b>
	<b>i</b>
....	
	<b>i</b>
	<b>i</b>

page table

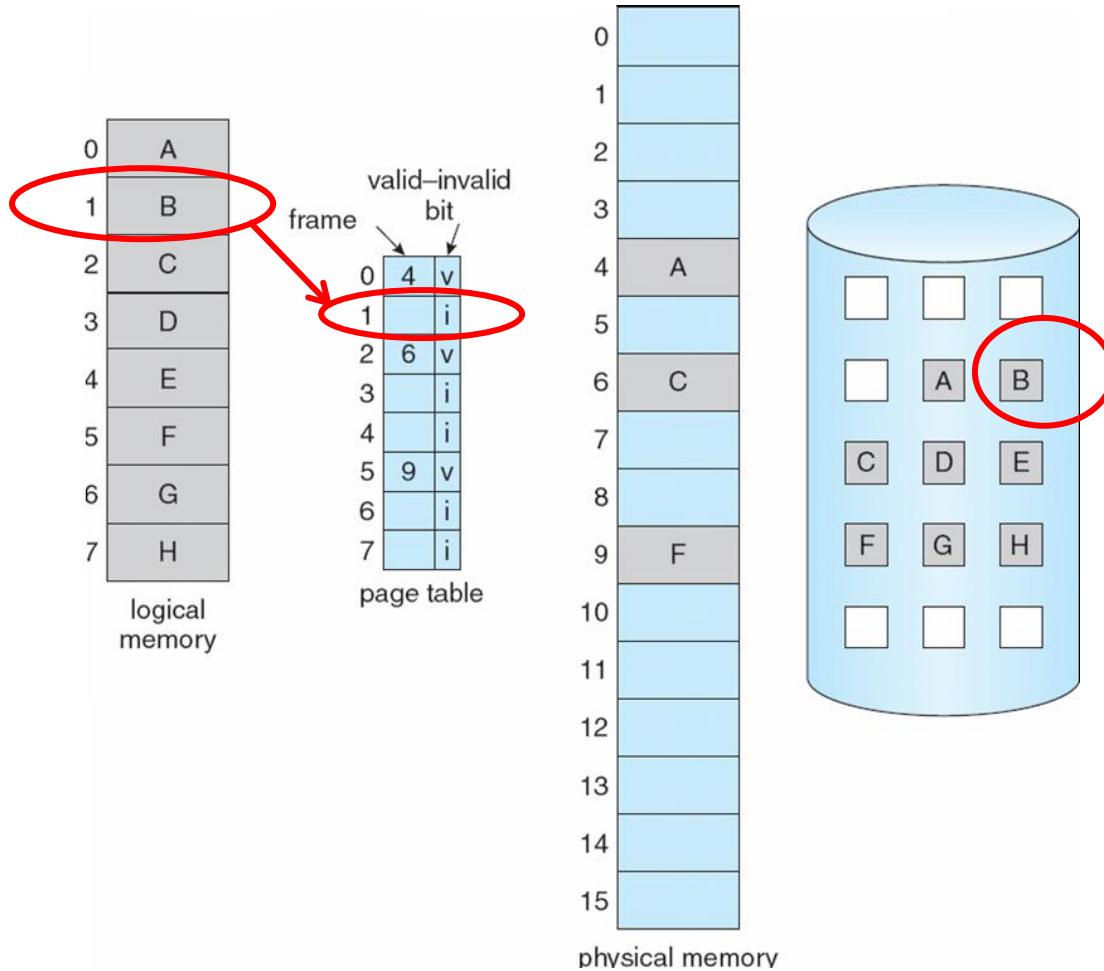
page fault

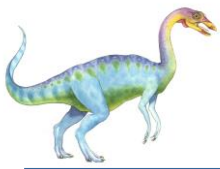
- During address translation, if valid–invalid bit in page table entry is **i** ⇒ **page fault** (การอ้างอิงผิดหน้าหรือไม่พบหน้าที่ต้องการ)





# Page Table When Some Pages Are Not in Main Memory





# Page Fault

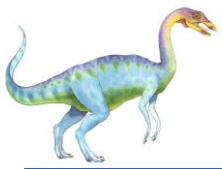
เมื่อมีการอ้างอิงผิดหน้าหรือไม่พบหน้าที่ต้องการในหน่วยความจำหลัก (**page fault**) จะมีขั้นตอนดำเนินการดังนี้

- If there is a reference to a page, first reference to that page will trap to operating system:

## page fault

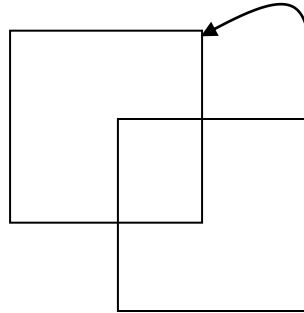
1. Operating system looks at another table to decide:
  - Invalid reference  $\Rightarrow$  abort
  - Just not in memory
2. Get empty frame
3. Swap page into frame
4. Reset tables
5. Set validation bit = **V**
6. Restart the instruction that caused the page fault





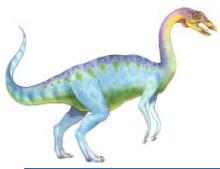
# Page Fault (Cont.)

- Restart instruction (ทำต่อจากจุดที่ได้ทำมาแล้วล่าสุด หรือ ทำต่อจากจุดที่เกิด page fault ขึ้น)
  - block move

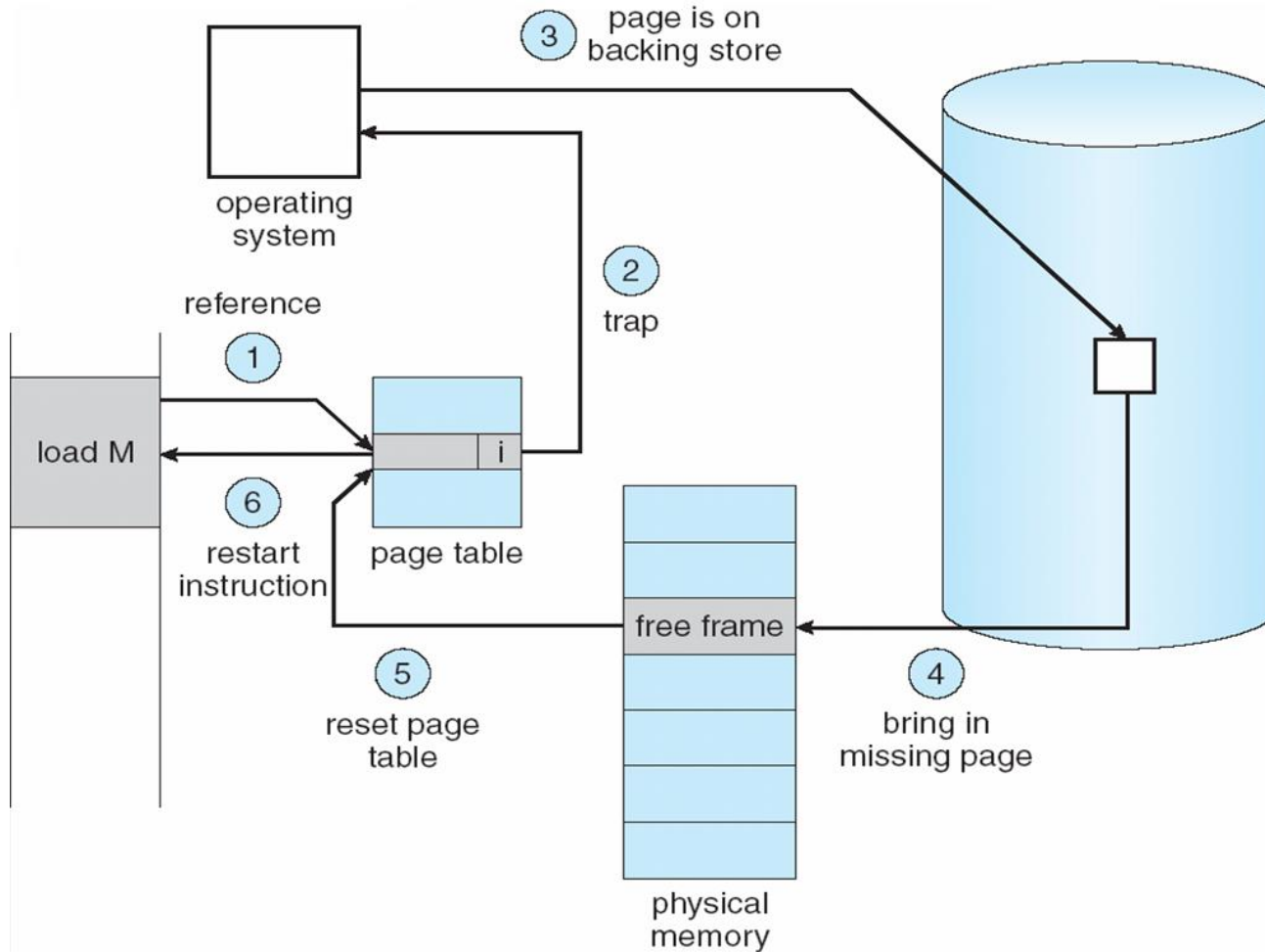


- auto increment/decrement location





# Steps in Handling a Page Fault





# What happens if there is no free frame?

---

- **Page replacement** – find some page in memory, but not really in use, swap it out
  - algorithm
  - performance – want an algorithm which will result in minimum number of page faults
- Same page may be brought into memory several times





# Page Replacement

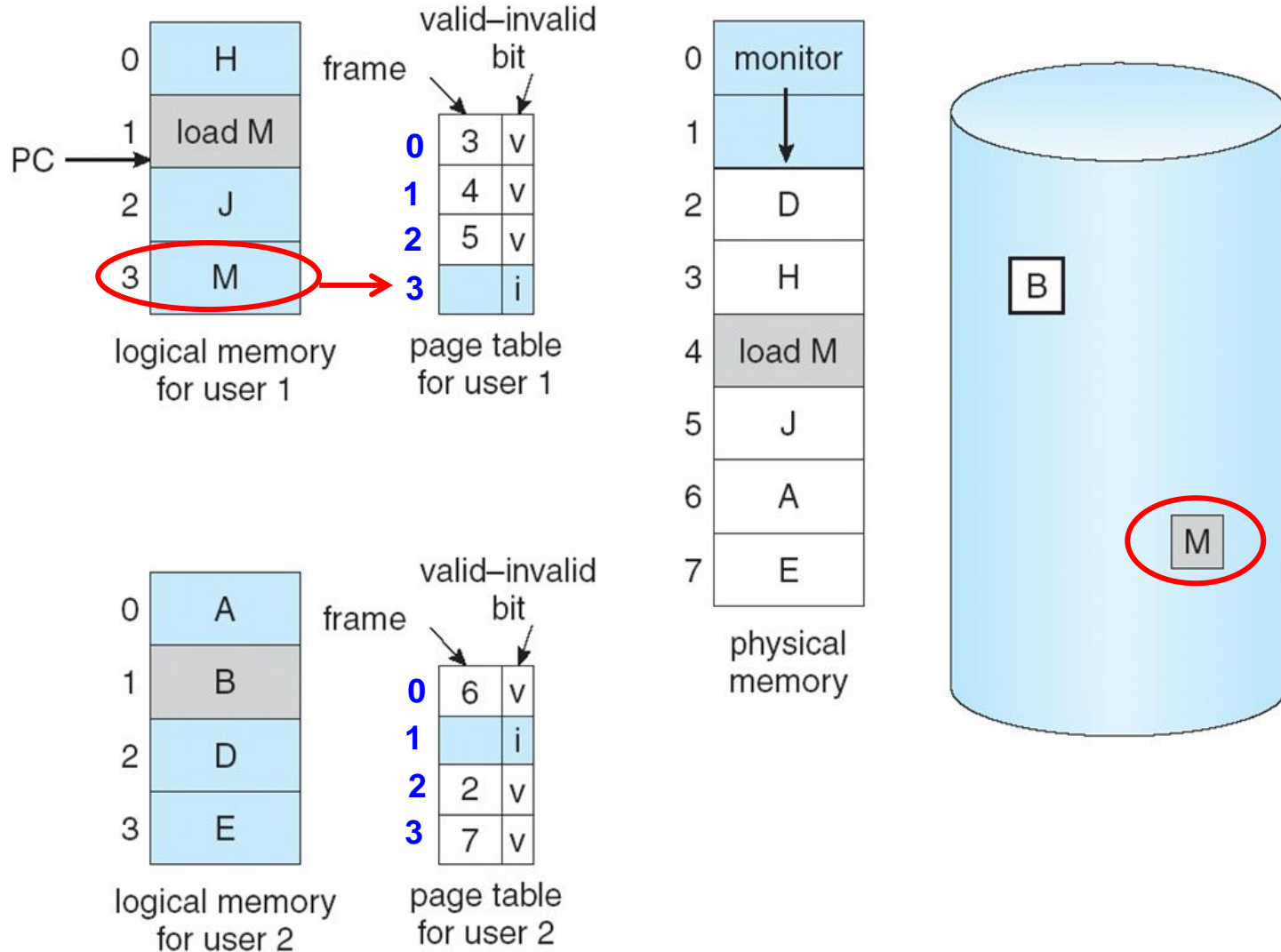
---

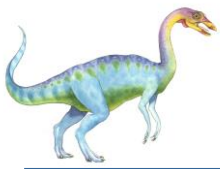
- **Prevent over-allocation of memory** by modifying page-fault service routine to include page replacement
- Use **modify (dirty) bit** to reduce overhead of page transfers – only modified pages are written to disk
- Page replacement completes separation between logical memory and physical memory – large virtual memory can be provided on a smaller physical memory





# Need For Page Replacement





# Basic Page Replacement

---

1. Find the location of the desired page on disk
2. Find a free frame:
  - If there is a free frame, use it
  - If there is no free frame, use a page replacement algorithm to select a **victim** frame
3. Bring the desired page into the (newly) free frame; update the page and frame tables
4. Restart the process





# Page Replacement

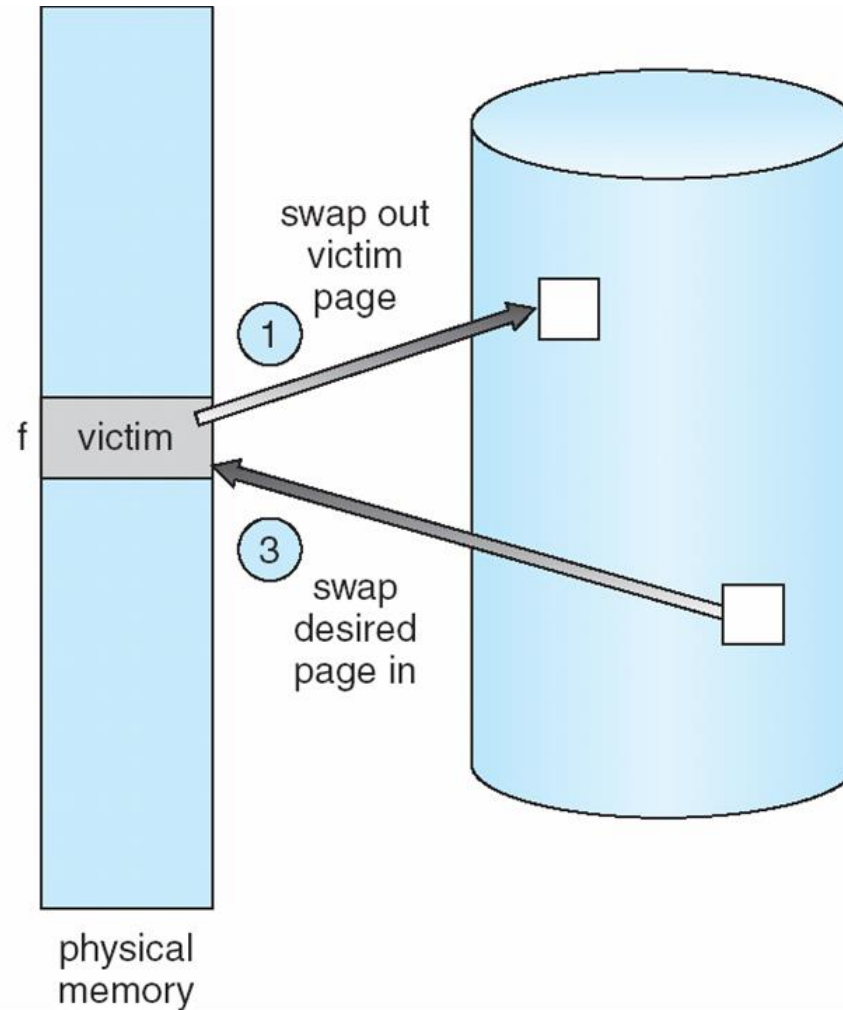
frame    valid-invalid bit

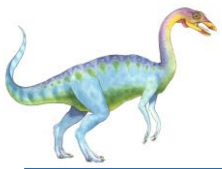
0	i
f	v

page table

② change to invalid

④ reset page table for new page





# Page Replacement Algorithms

---

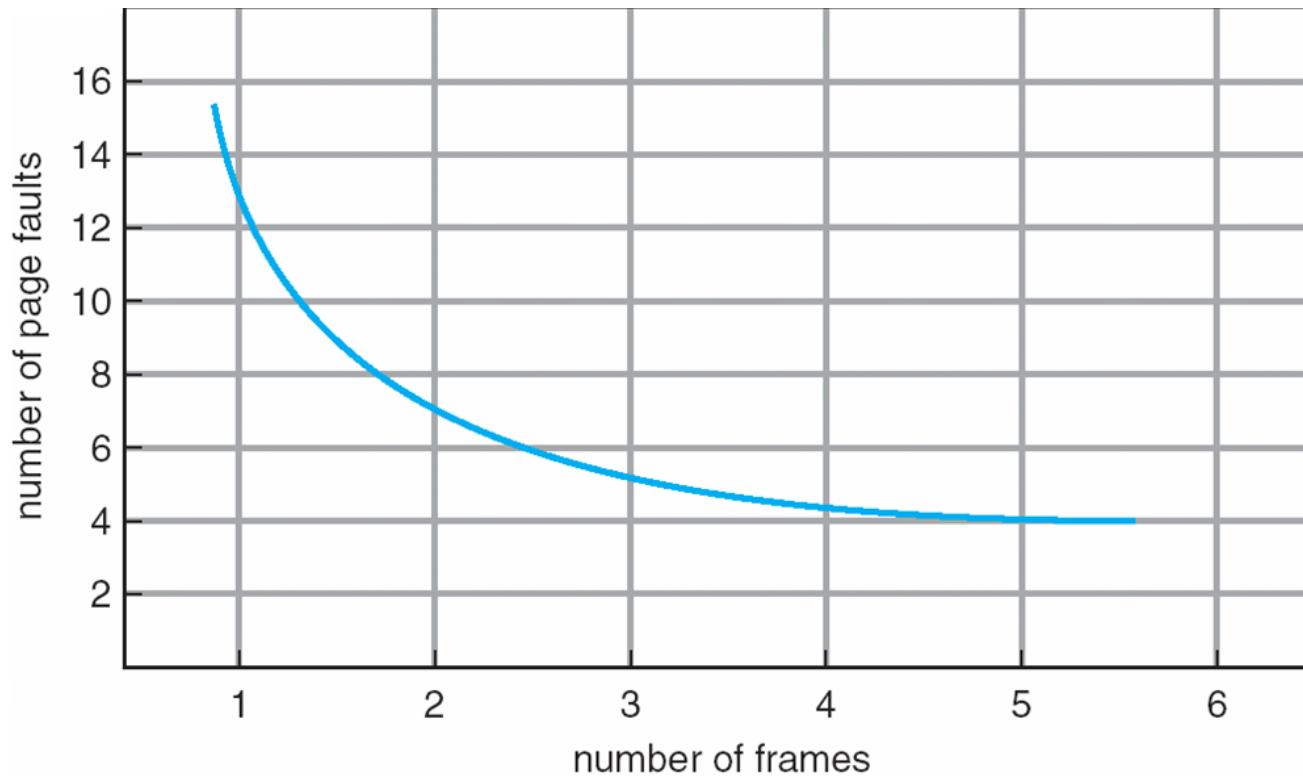
- **Want lowest page-fault rate**
- Evaluate algorithm by running it on a particular string of memory references (reference string) and computing the number of page faults on that string
- In all our examples, the reference string is

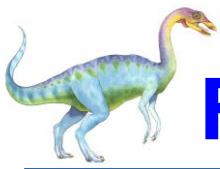
**1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5**





# Graph of Page Faults Versus The Number of Frames





# First-In-First-Out (FIFO) Algorithm

- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- 3 frames (3 pages can be in memory at a time per process)

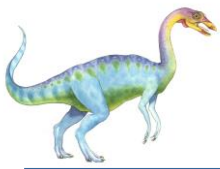
1	1	4	5	
2	2	1	3	เกิด 9 page faults
3	3	2	4	

- 4 frames

1	1	5	4	
2	2	1	5	เกิด 10 page faults
3	3	2		
4	4	3		

- Belady's Anomaly: more frames  $\Rightarrow$  more page faults



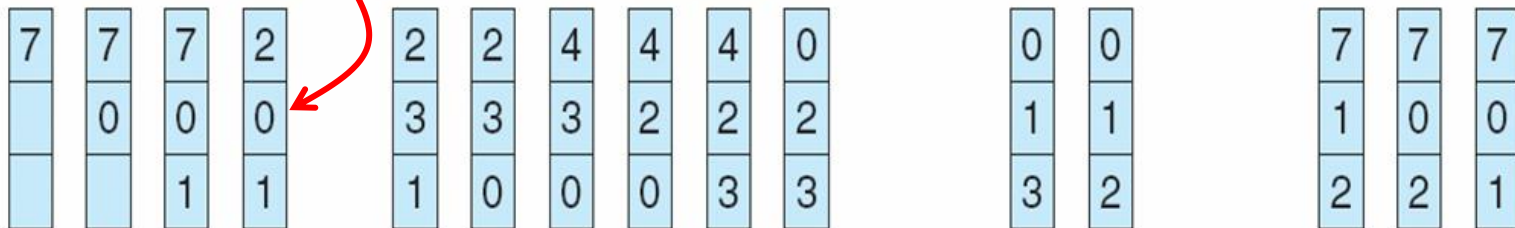


# FIFO Page Replacement

มี 0 อยู่แล้วใน memory จึงไม่ต้องไปดึงข้อมูลมาใหม่อีก  
จึงไม่เกิด page fault สำหรับ page 0

reference string

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1



page frames

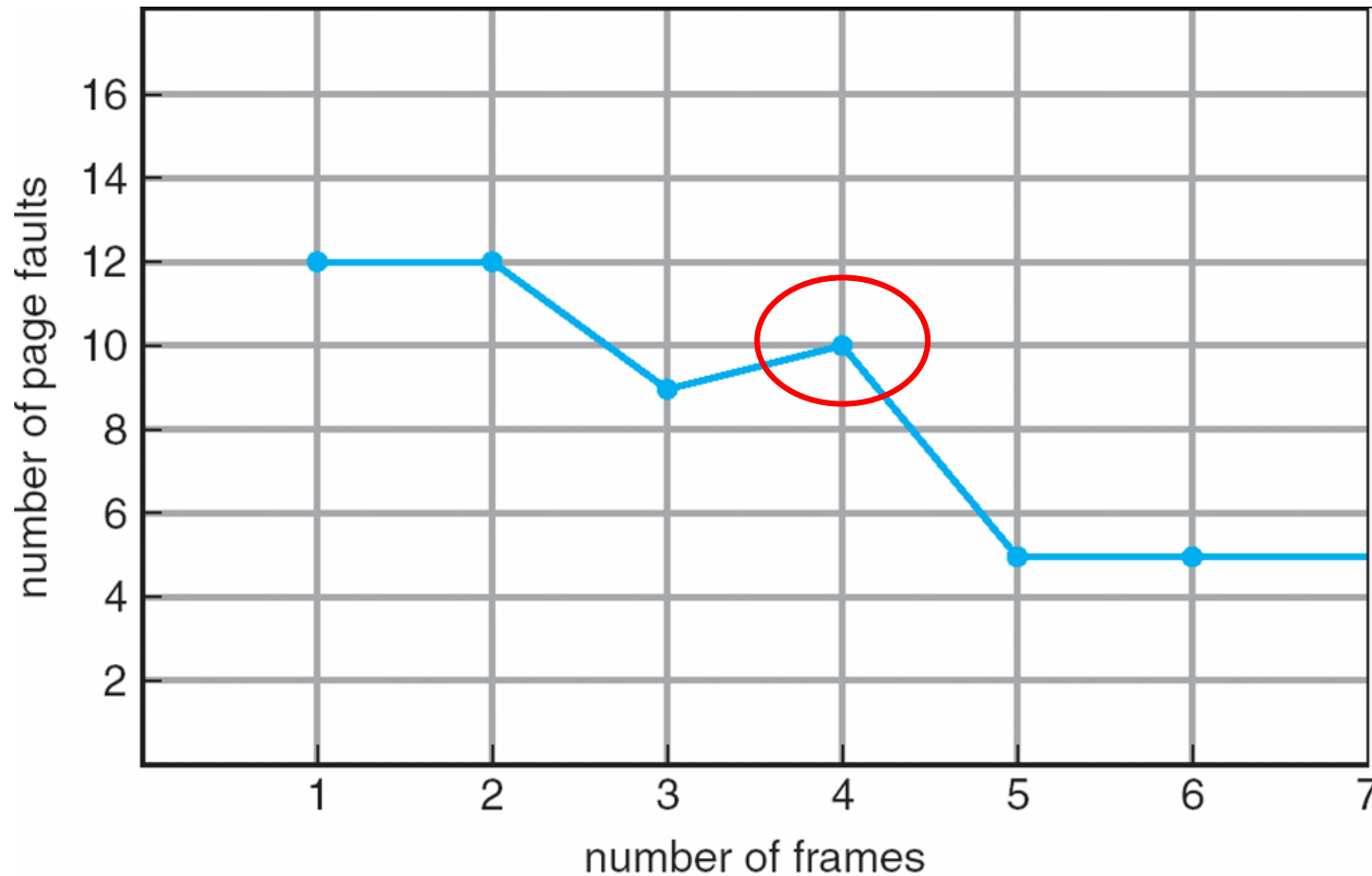
เมื่อ physical memory 3 frames และใช้ reference string  
ตามที่กำหนดให้ จะเกิด page fault ทั้งหมด **15 page faults**

- ครั้งแรกไม่มี page อยู่ใน Physical Memory จะเกิด page fault





# FIFO Illustrating Belady's Anomaly



**Belady's Anomaly** : เป็นเหตุการณ์ที่เมื่อมี Physical Memory เพิ่มขึ้น แต่จะเกิด page fault เพิ่มขึ้นด้วย (เป็นข้อยกเว้นสำหรับ FIFO Algorithm)





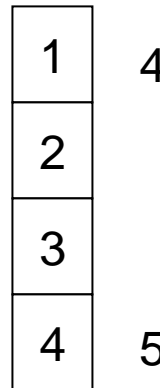
# Optimal Algorithm

- Replace page that will not be used for longest period of time

(มองใน list ของ reference string ถัดไปในอนาคตว่า page ใดอีกนานที่จะถูกใช้งาน จะโดน replace )

- 4 frames example

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

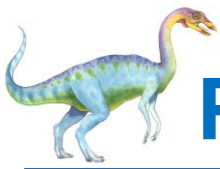


เกิด 6 page faults

ไม่ make sence เราไม่สามารถรู้อนาคตได้

- How do you know this?
- Used for measuring how well your algorithm performs

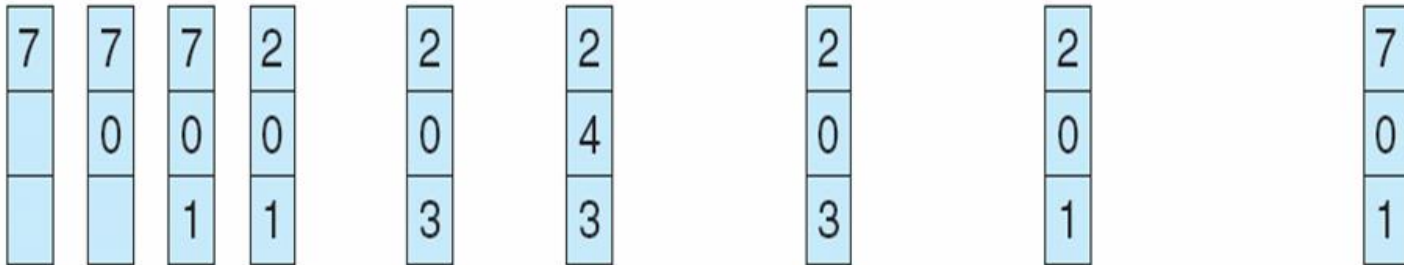




# Practice: Optimal Page Replacement

reference string

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1



page frames

เกิดขึ้น page faults ???



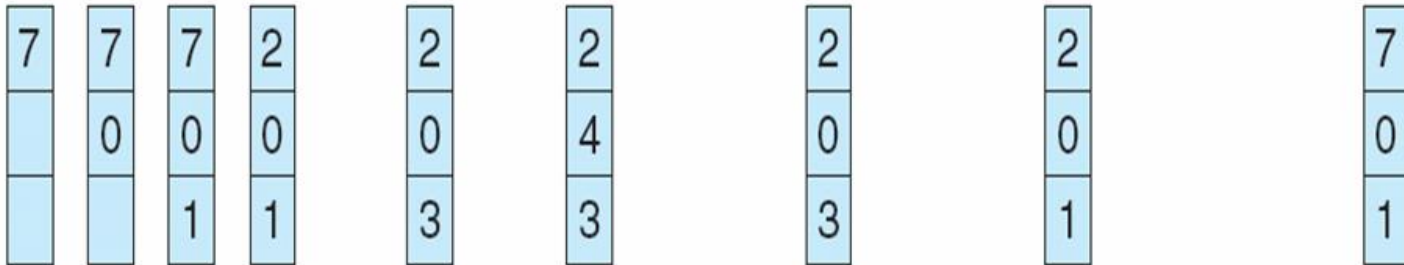


# Practice: Optimal Page Replacement

Answer!!

reference string

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1



page frames

เกิด **9** page faults





# Least Recently Used (LRU) Algorithm

(มองย้อนกลับไปในอดีตว่า **page** ใดไม่ได้ถูกใช้มานานที่สุดจะถูก **replace**)

- Reference string: 1, 2, 3, 4, 1, 2, **5**, 1, 2, **3**, **4**, **5**

1	1	1	1	<b>5</b>
2	2	2	2	2
3	<b>5</b>	5	4	4
4	4	<b>3</b>	3	3

เกิด **8** page faults

- Counter implementation
  - Every page entry has a counter; every time page is referenced through this entry, copy the clock into the counter
  - When a page needs to be changed, look at the counters to determine which are to change

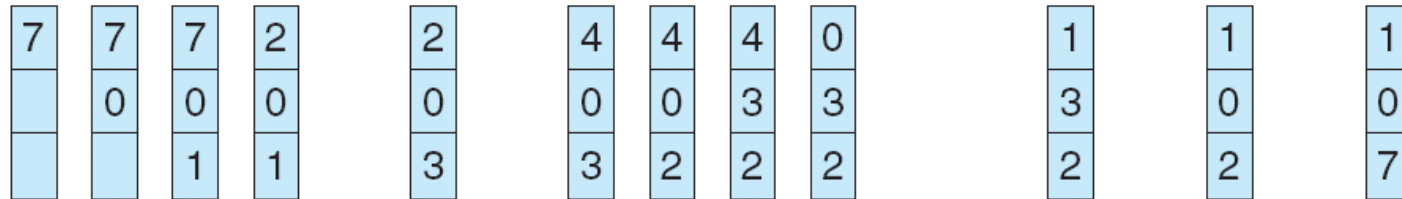




# Practice: LRU Page Replacement

reference string

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1



page frames

เกิดขึ้นที่ page fault ???



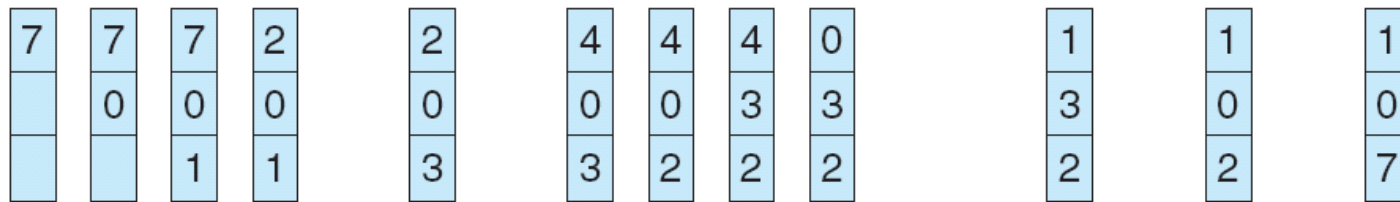


# Practice: LRU Page Replacement

Answer!!

reference string

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

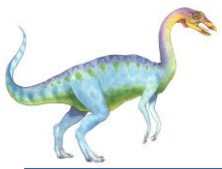


page frames

เกิด **12** page faults

- Better than FIFO but worse than Optimal Algorithm



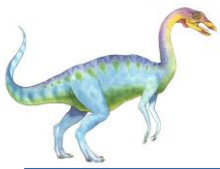


# Counting Algorithms

- Keep a counter of the number of references that have been made to each page
- **LFU Algorithm**: replaces page with smallest count
- **MFU Algorithm**: based on the argument that the page with the smallest count was probably just brought in and has yet to be used

**LFU : Least Frequently Used** (ความถี่ในการถูกใช้น้อยที่สุด)  
**MFU: Most Frequently Used** (ความถี่ในการถูกใช้มากที่สุด)





# Allocation of Frames

---

- Each process needs *minimum* number of pages
- Example: IBM 370 – 6 pages to handle SS MOVE instruction:
  - instruction is 6 bytes, might span 2 pages
  - 2 pages to handle *from*
  - 2 pages to handle *to*
- Two major allocation schemes
  - fixed allocation
  - priority allocation





# Fixed Allocation

- Equal allocation – For example, if there are 100 frames and 5 processes, give each process 20 frames. (ได้จาก (frame / process) = (100 / 5))
- Proportional allocation – Allocate according to the size of process

- $s_i$  = size of process  $p_i$
- $S = \sum s_i$
- $m$  = total number of frames
- $a_i$  = allocation for  $p_i = \frac{s_i}{S} \times m$

$m = 64$

$s_i = 10$

$s_2 = 127$

$a_1 = \frac{10}{137} \times 64 \approx 5$

$a_2 = \frac{127}{137} \times 64 \approx 59$

**P<sub>i</sub> ได้ 5 frames**

**P<sub>2</sub> ได้ 59 frames**

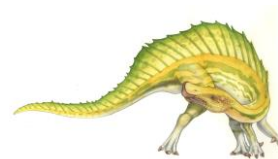




# Priority Allocation

---

- Use a proportional allocation scheme using priorities rather than size
  
- If process  $P_i$  generates a page fault,
  - select for replacement one of its frames
  - select for replacement a frame from a process with lower priority number





# Global vs. Local Allocation

---

- **Global replacement** – process selects a replacement frame from the set of all frames; one process can take a frame from another
- **Local replacement** – each process selects from only its own set of allocated frames



# End of Chapter 8

---

