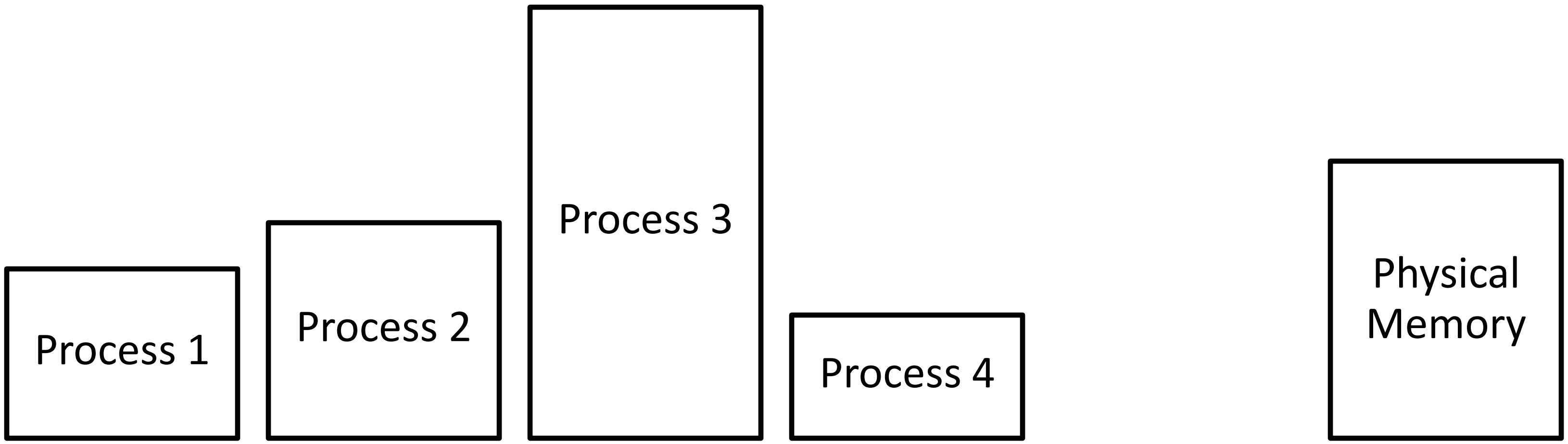
แก้ไขครั้งล่าสุดเมื่อวันที่ 23 ตุลาคม 2567

Virtual Memory

Why?

The need of memory more than the available physical memory.



Physical Memory Limits: Windows 8

The following table specifies the limits on physical memory for Windows 8.

Version	Limit on X86	Limit on X64
Windows 8 Enterprise	4 GB	512 GB
Windows 8 Professional	4 GB	512 GB
Windows 8	4 GB	128 GB

Physical Memory Limits: Windows 7

The following table specifies the limits on physical memory for Windows 7.

Version	Limit on X86	Limit on X64
Windows 7 Ultimate	4 GB	192 GB
Windows 7 Enterprise	4 GB	192 GB
Windows 7 Professional	4 GB	192 GB
Windows 7 Home Premium	4 GB	16 GB
Windows 7 Home Basic	4 GB	8 GB
Windows 7 Starter	2 GB	N/A

Memory Limits for Windows Releases
http://msdn.microsoft.com/en-us/library/windows/desktop/aa366778(v=vs.85).aspx

Physical Memory Limits: Windows 11

The following table specifies the limits on physical memory for Windows 11.

Version	Limit on X64	Limit on ARM64
Windows 11 Enterprise	6 TB	6 TB
Windows 11 Education	2 TB	2 TB
Windows 11 Pro for Workstations	6 TB	6 TB
Windows 11 Pro	2 TB	2 TB
Windows 11 Home	128 GB	128 GB

Physical Memory Limits: Windows 10

The following table specifies the limits on physical memory for Windows 10.

Version	Limit on X86	Limit on X64
Windows 10 Enterprise	4 GB	6 TB
Windows 10 Education	4 GB	2 TB
Windows 10 Pro for Workstations	4 GB	6 TB
Windows 10 Pro	4 GB	2 TB
Windows 10 Home	4 GB	128 GB

https://learn.microsoft.com/en-us/windows/win32/memory/memory-limits-for-windows-releases

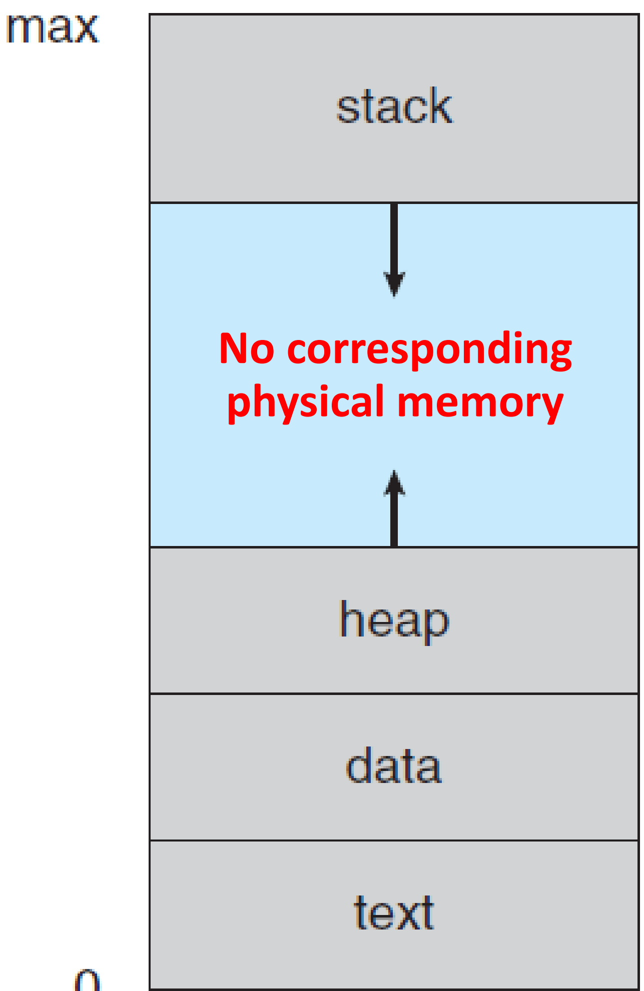
Single-process memory limits

Memory type	Limit on X86	Limit in 64-bit Windows
User-mode virtual address space for each 32-bit process	2 GB Up to 3 GB with IMAGE_FILE_LARGE_ADDRESS_AWARE and 4GT	2 GB with IMAGE_FILE_LARGE_ADDRESS_AWARE cleared (default) 4 GB with IMAGE_FILE_LARGE_ADDRESS_AWARE set
User-mode virtual address space for each 64-bit process	Not applicable	With IMAGE_FILE_LARGE_ADDRESS_AWARE set (default): x64: Windows 8.1 and Windows Server 2012 R2 or later: 128 TB x64: Windows 8 and Windows Server 2012 or earlier 8 TB Intel Itanium-based systems: 7 TB 2 GB with IMAGE_FILE_LARGE_ADDRESS_AWARE cleared

The memory limit of 2 GB is over!

https://learn.microsoft.com/en-us/windows/win32/memory/memory-limits-for-windows-releases

Virtual Memory



Virtual memory
(a process)

<u>ประโยชน์หลักของ virtual memory</u>

- จอง memory ได้มากกว่าที่มีอยู่จริง (ใช้ backing store ช่วย)

<u>ประโยชน์รอง</u>

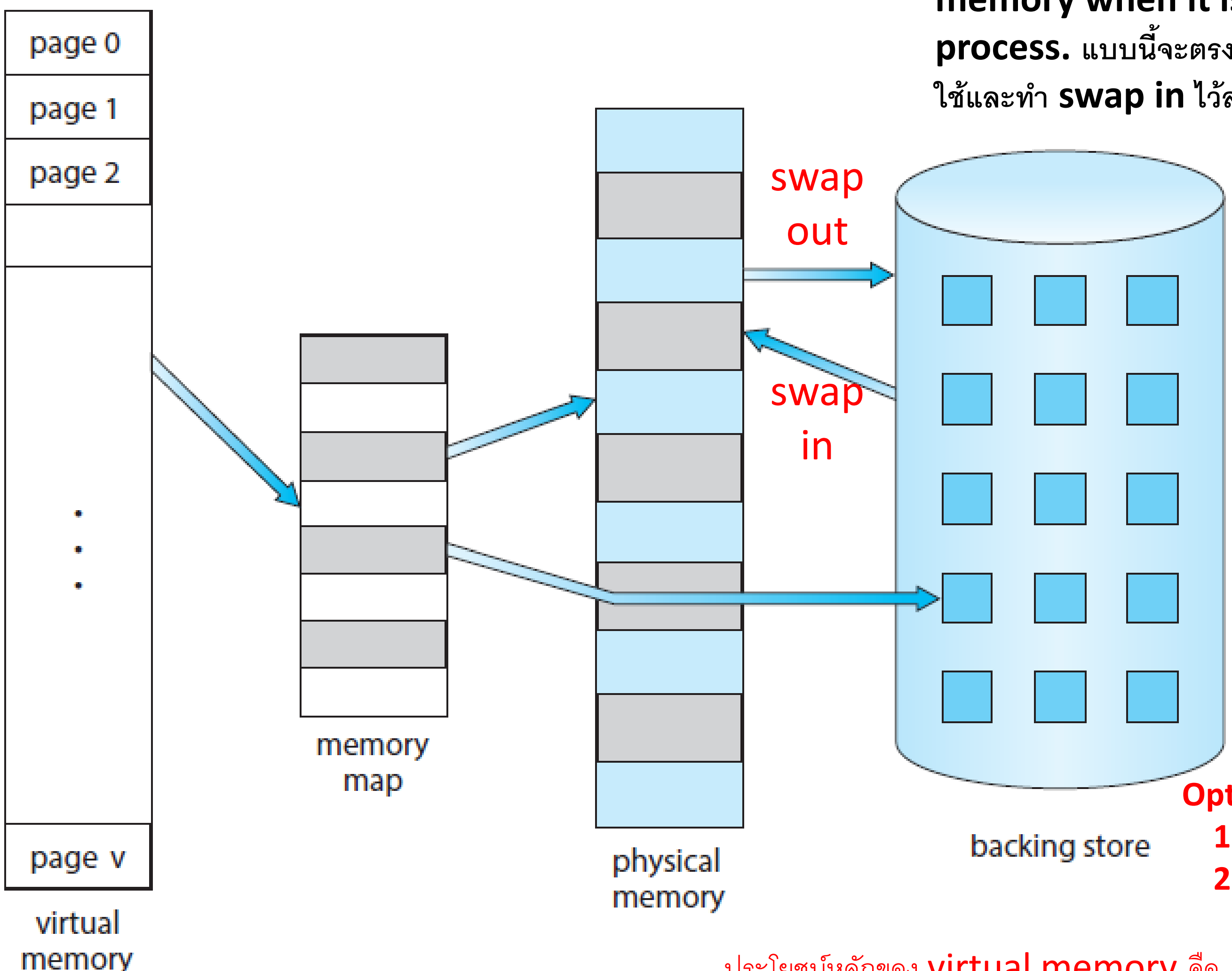
- ทุก process มี address space เริ่มจาก 0 (ไม่ต้องทำ relocation)
- ไม่ต้องให้ frame กับ page ทั้งหมด ค่อยให้เมื่อต้องการใช้

As stack and heap grow, more pages will be allocated and mapped to physical memory.

Virtual Memory

Demand paging (lazy swapping)

The OS only swaps a page into memory when it is required by a process. แบบนี้จะตรงข้ามกับการคาดว่าจะต้องใช้และทำ **swap in** ไว้ล่วงหน้า



Options

1. **Raw disk**
2. **File system**

ประโยชน์หลักของ virtual memory คือ
process สามารถจอง memory มากกว่า memory ที่มีจริง ๆ ได้

Figure 10.1 Diagram showing virtual memory that is larger than physical memory.

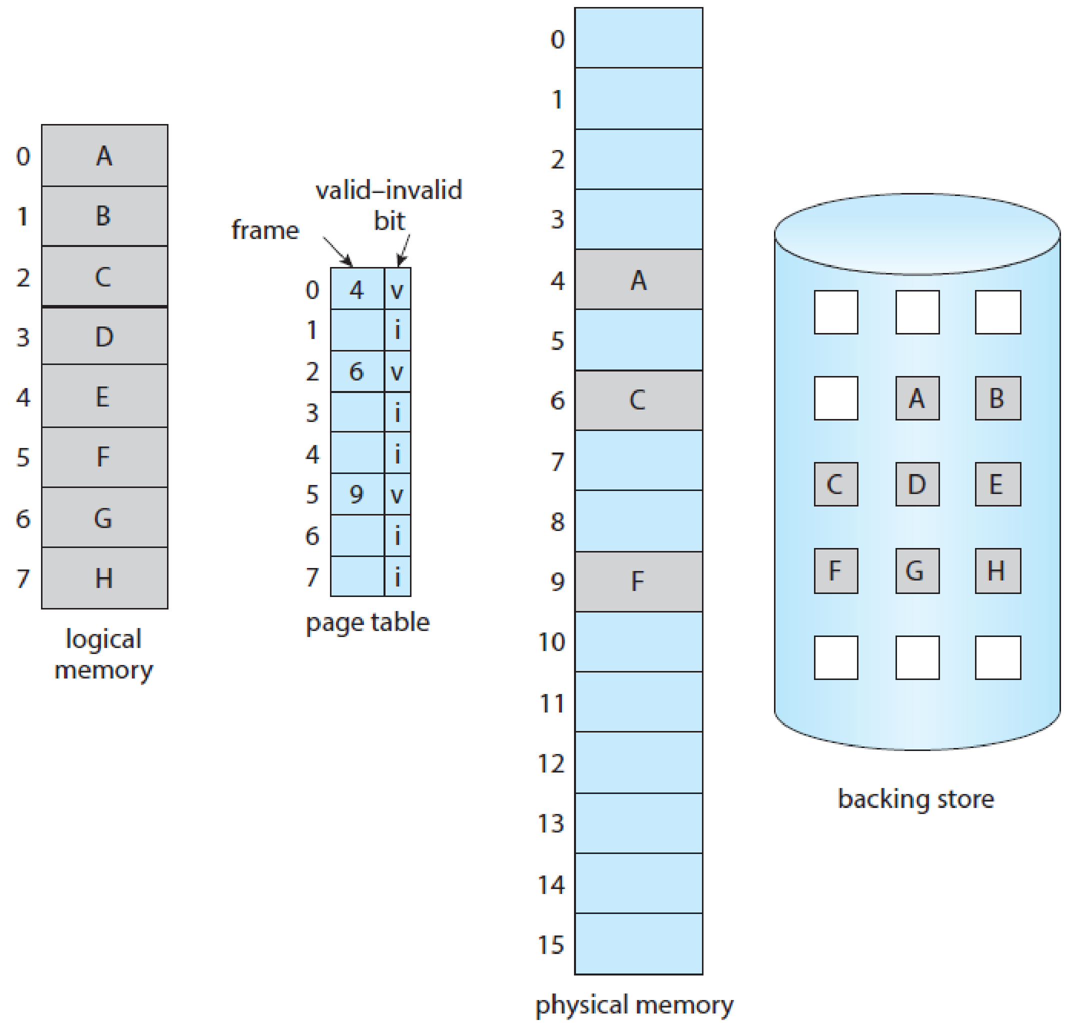
Summary

มี logical
กับ physical

แบ่งเป็น page
ไม่ใช่ contiguous

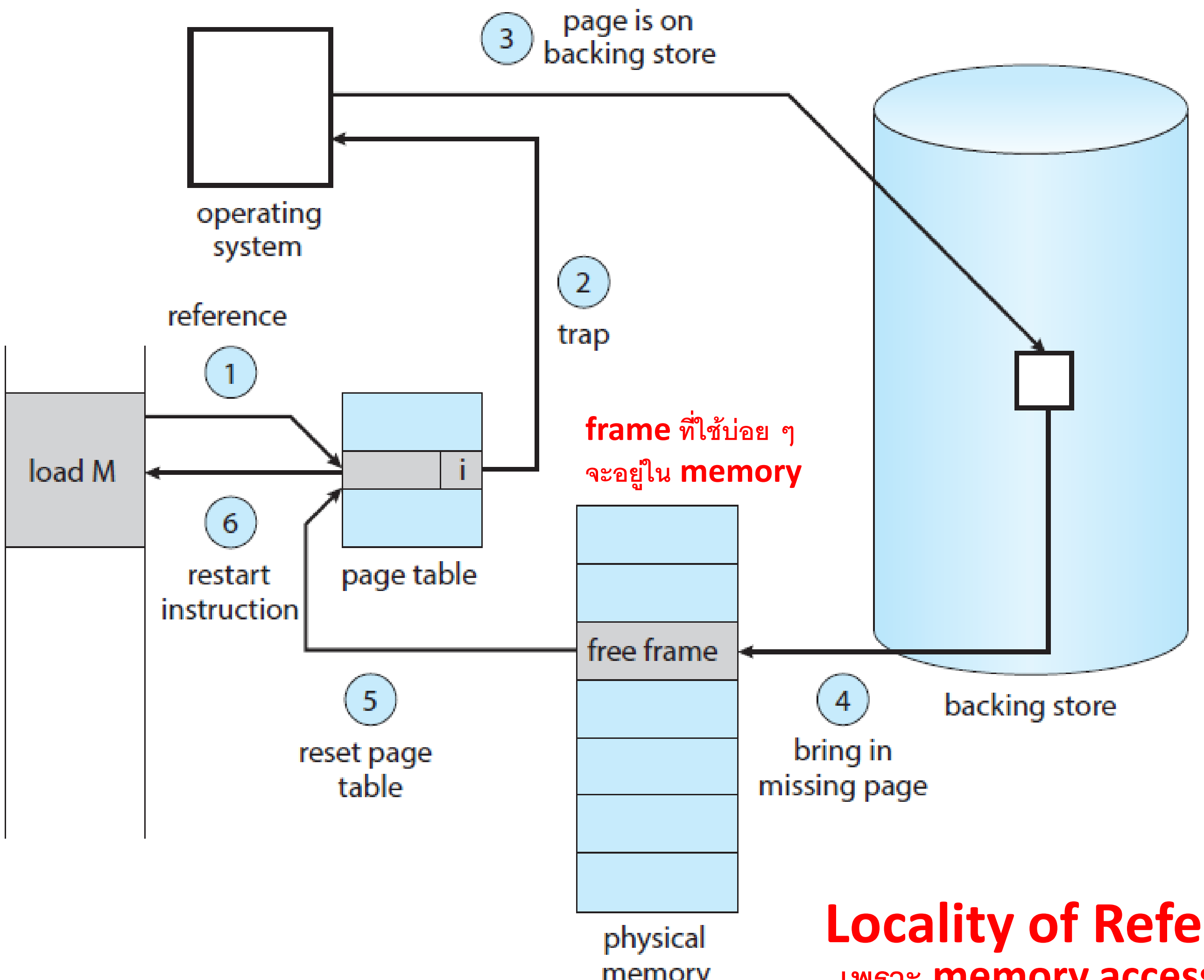
swap กับ
backing store

Virtual memory = mapping + paging + swapping



valid คือ อยู่บน memory
invalid คือ ไม่อยู่บน memory
อยู่บน backing store

Page Fault



frame ที่ใช้บ่อย ๆ จะอยู่ใน memory

Locality of Reference

เพราะ **memory access** ไม่ได้ใช้ **random address** ถึงใช้ **paging** และ **virtual memory** ได้

Performance of Demand Paging

$$\text{effective access time} = (1 - p) \times ma + p \times \text{page fault time}.$$

With an average page-fault service time of 8 milliseconds and a memory-access time of 200 nanoseconds, the effective access time in nanoseconds is

Page fault ช้าลง
8ms / 200 ns
= 40,000 เท่า

$$\begin{aligned}\text{effective access time} &= (1 - p) \times (200) + p\ (8 \text{ milliseconds})\\ &= (1 - p) \times 200 + p \times 8{,}000{,}000\\ &= 200 + 7{,}999{,}800 \times p.\end{aligned}$$

We see, then, that the effective access time is directly proportional to the **page-fault rate**. If one access out of 1,000 causes a page fault, the effective access time is 8.2 microseconds. The computer will be slowed down by a factor of 40 because of demand paging! If we want performance degradation to be less than 10 percent, we need to keep the probability of page faults at the following level:

p = 0.001 ช้าลง
8.2 µs / 200 ns
= 41 เท่า

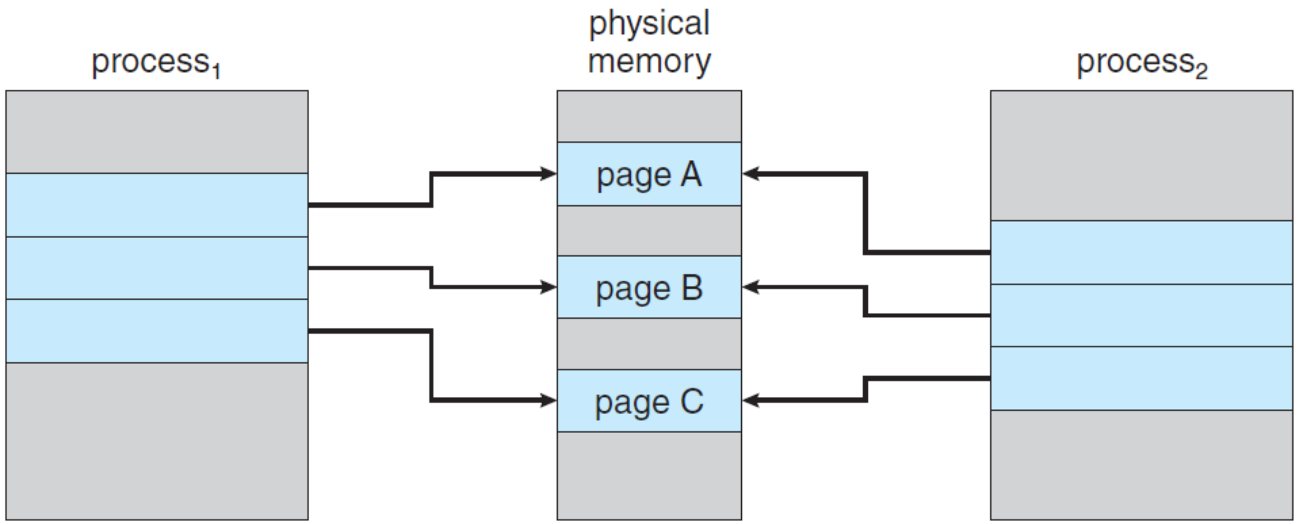
$$\begin{aligned}220 &> 200 + 7{,}999{,}800 \times p,\\ 20 &> 7{,}999{,}800 \times p,\\ p &< 0.0000025.\end{aligned}$$

$= 2.5 \times 10^{-6}$ เพื่อให้ effective access time เพิ่มขึ้นไม่เกิน 20%

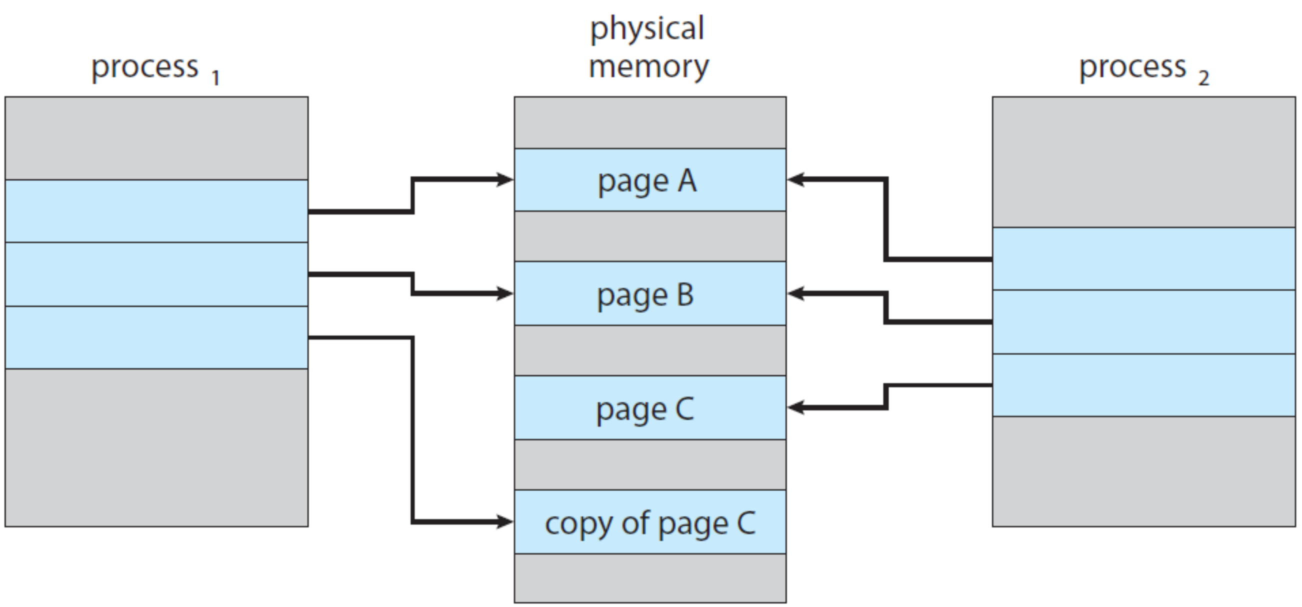
That is, to keep the slowdown due to paging at a reasonable level, we can allow fewer that one memory access out of 399,990 to page-fault. In sum, it is important to keep the page-fault rate low in a demand-paging system. Otherwise, the effective access time increases, slowing process execution dramatically.

ใส่ RAM ให้พอกับการใช้งาน

Copy-on-Write

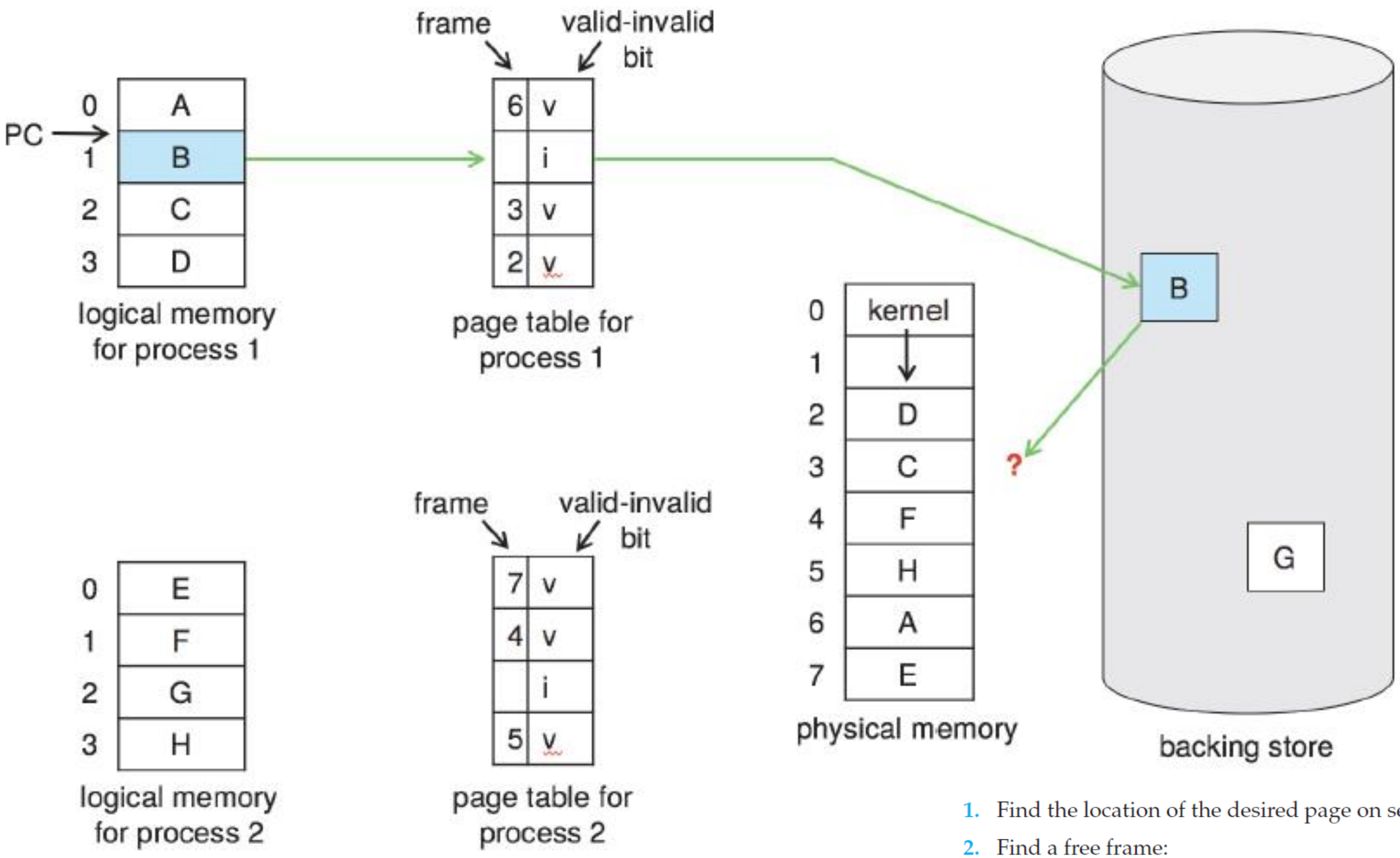


After forking. Parent and child share the same copy.

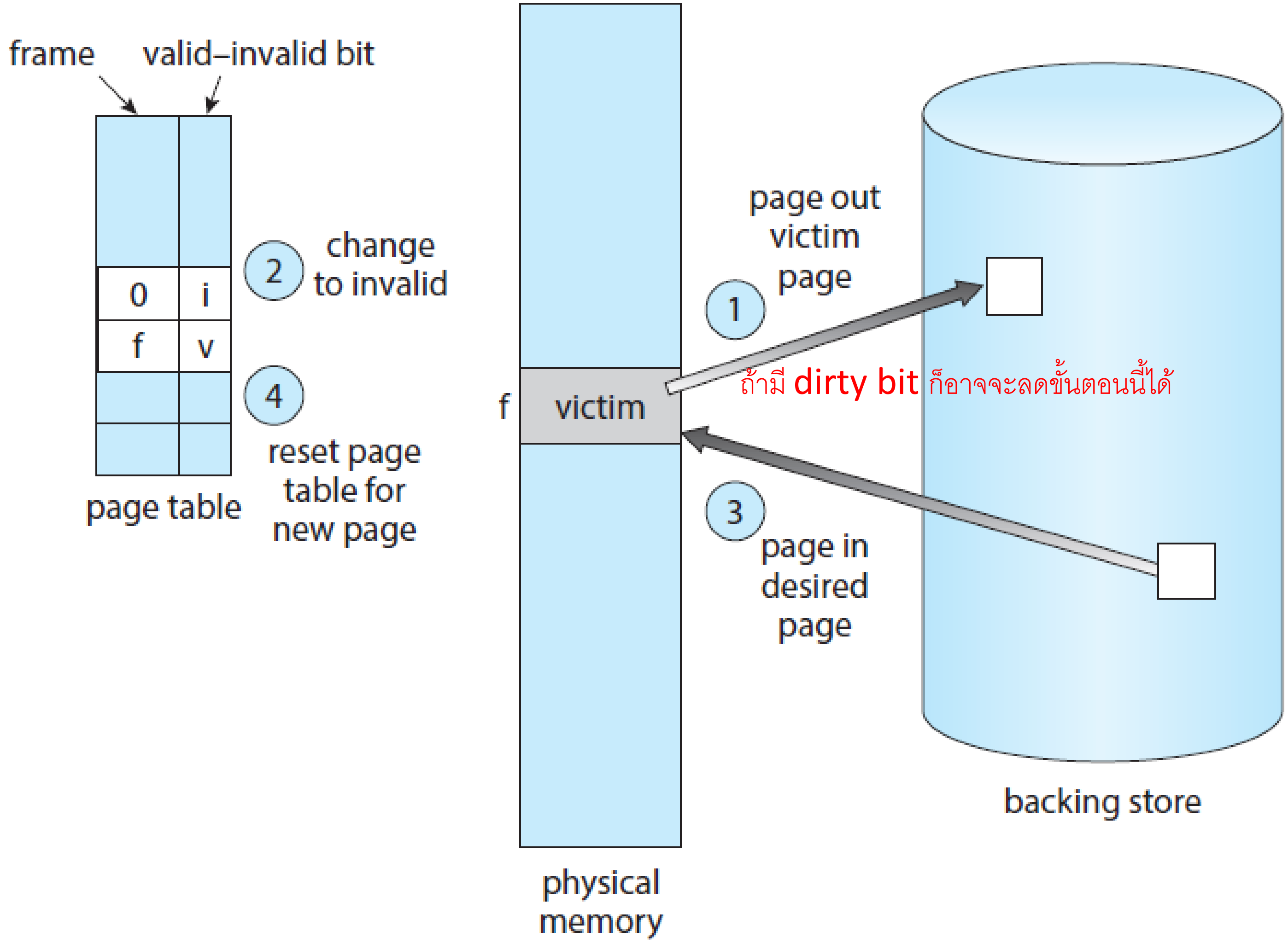


Make another copy of page C when a process writes.

Need for Page Replacement



1. Find the location of the desired page on secondary storage.
2. Find a free frame:
   a. If there is a free frame, use it.
   b. If there is no free frame, use a page-replacement algorithm to select a victim frame.
   c. Write the victim frame to secondary storage (if necessary); change the page and frame tables accordingly.
3. Read the desired page into the newly freed frame; change the page and frame tables.
4. Continue the process from where the page fault occurred.



OS เพิ่ม modify bit หรือ dirty bit ให้ทุก frame เพื่อที่ว่าถ้า frame นั้นยังไม่ถูก write (ไม่ dirty) จะได้ไม่ต้องเขียนลง backing store ในหนังสือบอกว่า dirty bit อยู่กับฮาร์ดแวร์ เดาว่าน่าจะเป็น memory controller คือทุกครั้งที่มีการเขียน frame นั้น ฮาร์ดแวร์จะอัปเดต dirty bit = 1

Demand paging requires

1) frame-allocation algorithm จะให้กี่ max. frame per process

2) page-replacement algorithm page ไหนจะเป็น victim

For example, if we trace a particular process, we might record the following address sequence:

0100, 0432, 0101, 0612, 0102, 0103, 0104, 0101, 0611, 0102, 0103,
0104, 0101, 0610, 0102, 0103, 0104, 0101, 0609, 0102, 0105

At 100 bytes per page, this sequence is reduced to the following reference string:

Reference string

1, 4, 1, 6, 1, 6, 1, 6, 1, 6, 1

max. frame = 2 เกิด **page fault**
max. frame = 3 ไม่เกิด **page fault**

Page Replacement Algorithms

1) **FIFO page replacement**
   - **Belady's anomaly**
2) **Optimal page replacement**
   - **Replace the page that will not be used for the longest period of time.**
   - **Similar to SJF, requiring future knowledge.**
3) **Least-recently-used (LRU) page replacement**
   - **Counter, equip a counter for each entry in page table**
   - **Stack, move the referenced page to TOS**
4) **LRU-approximation page replacement**
   - **Additional-reference-bits algorithm**
   - **Second-chance algorithm**
   - **Enhanced second-chance algorithm**

Page Replacement Algorithms (cont.)

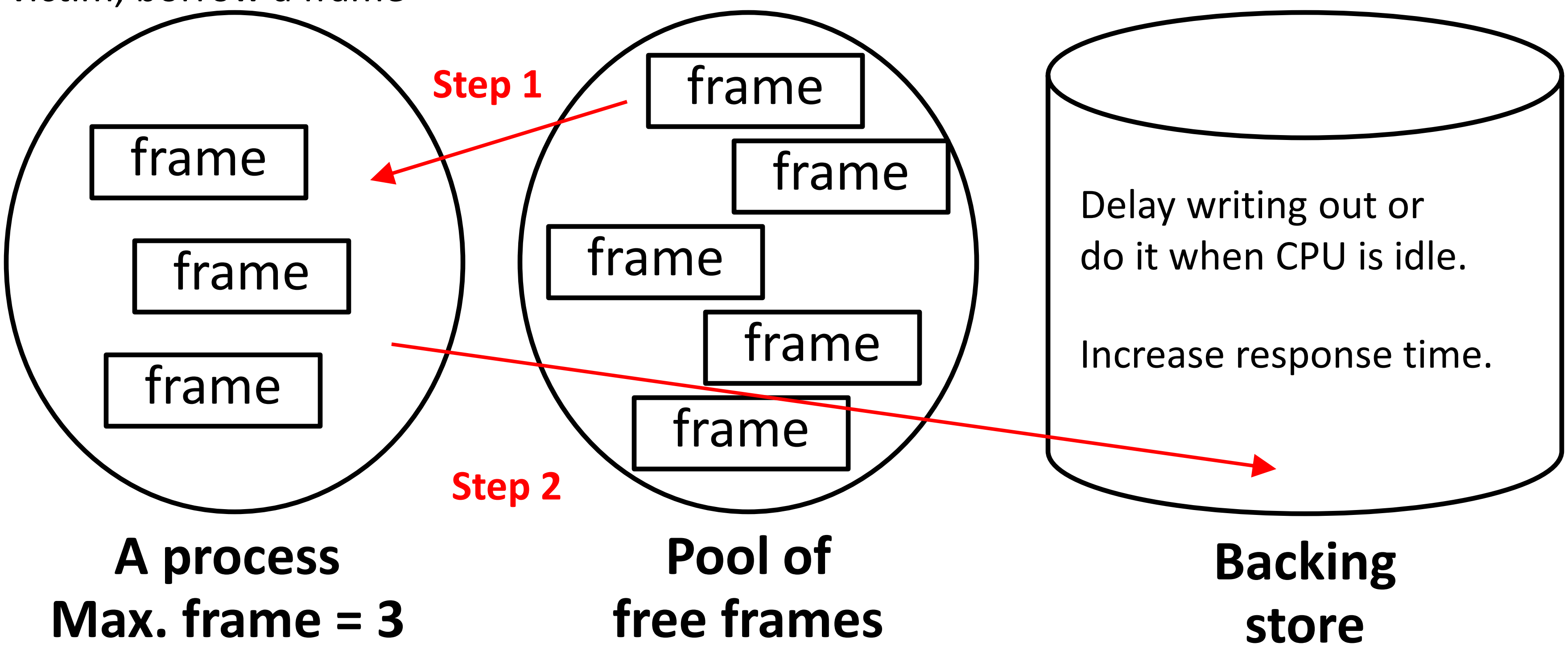
5) Counting-based page replacement frequently ≠ recently

- Least frequently used (LFU) page-replacement algorithm

ถูกใช้บ่อย แสดงว่าเก่าแล้ว - Most frequently used (MFU) page-replacement algorithm

6) Page-buffering algorithms (เป็นเทคนิคเสริม)

Page fault, not choose a victim, borrow a frame



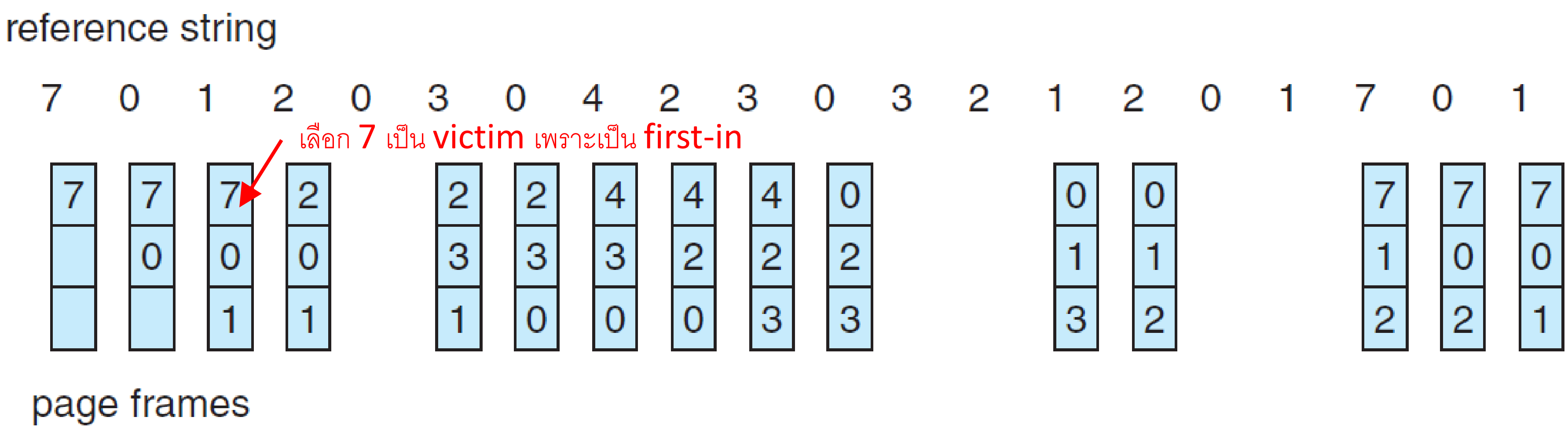


Figure 10.12 FIFO page-replacement algorithm.

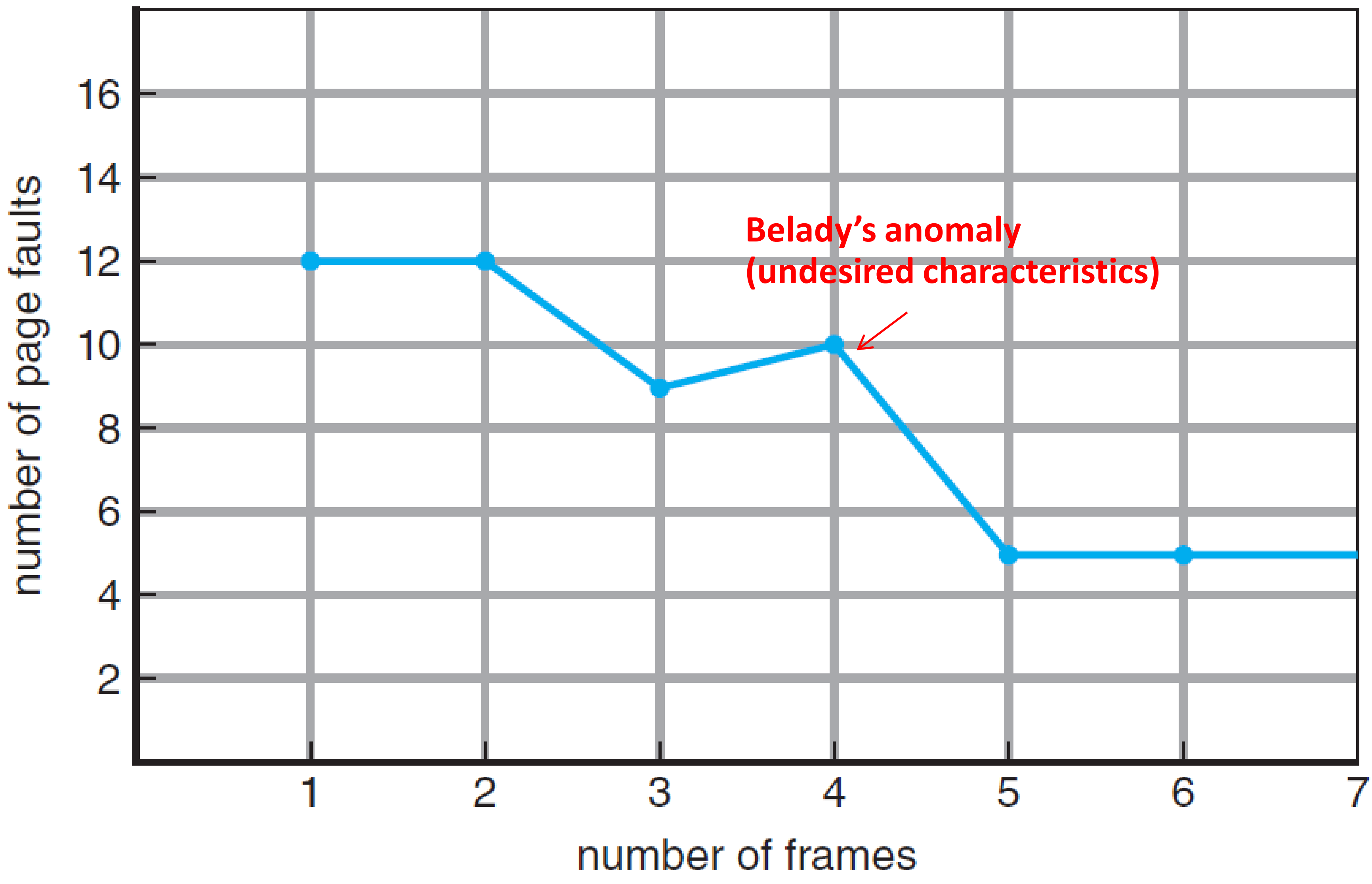


Figure 10.13 Page-fault curve for FIFO replacement on a reference string.

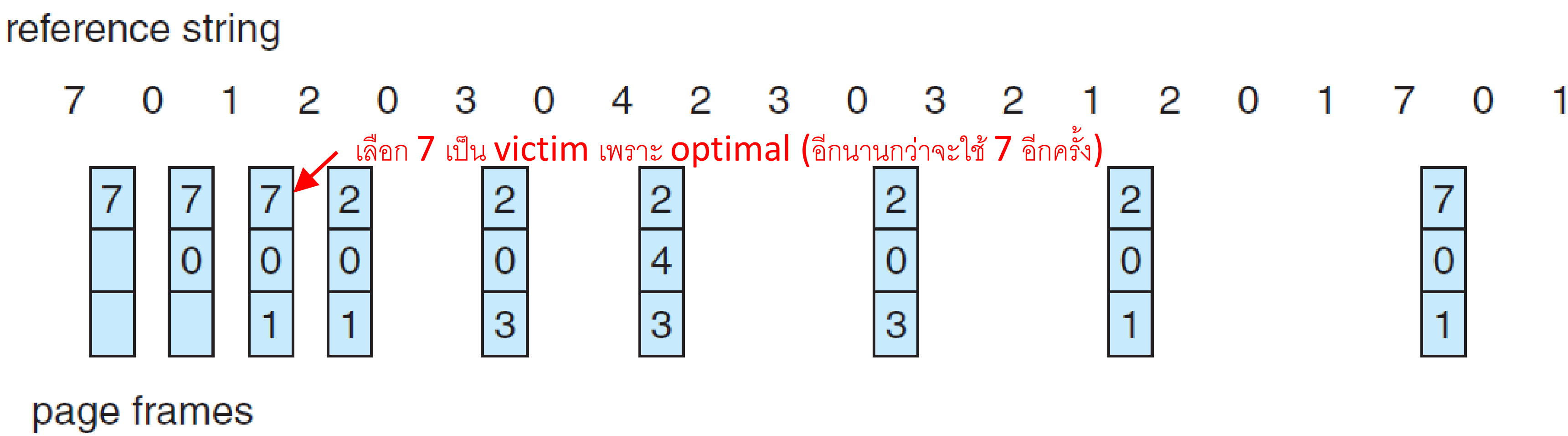


Figure 10.14 Optimal page-replacement algorithm.

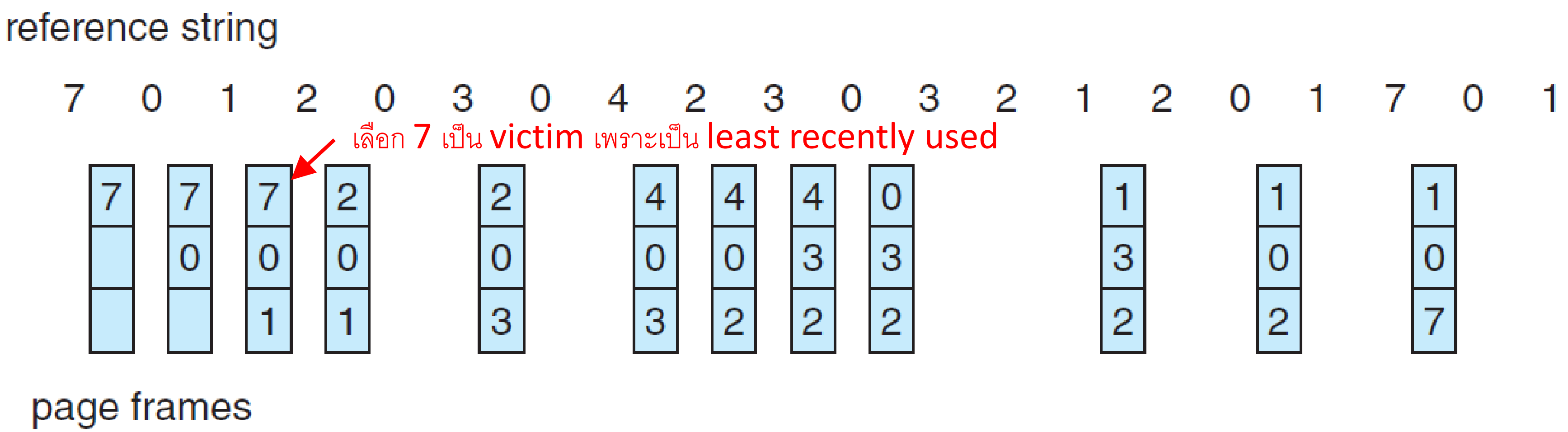
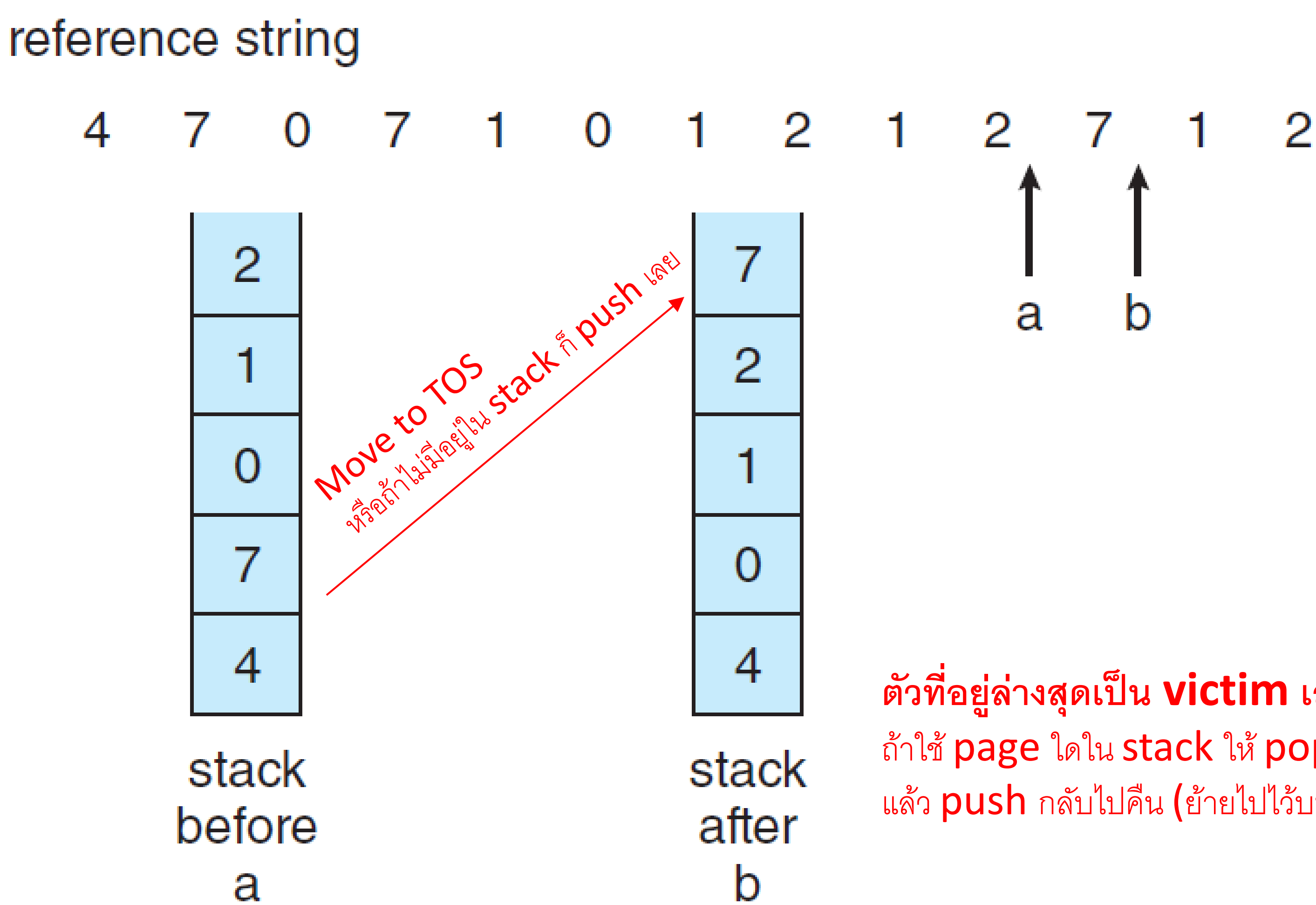


Figure 10.15 LRU page-replacement algorithm.

เหมือนเลือกเสื้อผ้าตัวเก่าทิ้ง

1. ติด counter ไว้กับทุก frame
2. counter จะมีค่าเพิ่มขึ้นตามเวลา
3. counter จะถูก reset ให้มีค่าเท่ากับ 0 เมื่อมีการใช้ frame นั้น
4. LRU คือเลือก frame ที่ counter มีค่ามากที่สุด



ตัวที่อยู่ล่างสุดเป็น victim เสมอ
ถ้าใช้ page ใดใน stack ให้ pop
แล้ว push กลับไปคืน (ย้ายไปไว้บนสุด)

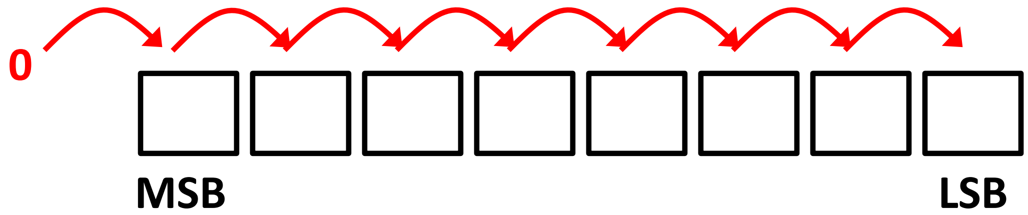
Figure 10.16 Use of a stack to record the most recent page references.

เหมือนตู้เสื้อผ้า ชุดเก่าอยู่ล่างชุด

Additional-reference-bits algorithm

Counting is more expensive then shifting

Shift right (less registers than counter)



Example

1000 0000

100ms

0100 0000

100ms

0010 0000

access

1010 0000

100ms

0101 0000

unsigned int!

- **Each page has a corresponding 8-bit register.**
- **If the page is accessed, MSB is set to 1.**
- **Every 100 ms, shift-right (÷2) all registers.**
- **The page with the lowest number is the LRU page.**

มองไปในอดีตได้ไกลมาก เท่าไรก็ได้ (ปรับให้ >100ms) แต่ใช้ที่ 8 บิตเสมอ

เหมือนการใช้ counter แต่เป็นการประมาณค่า

Second-chance algorithm

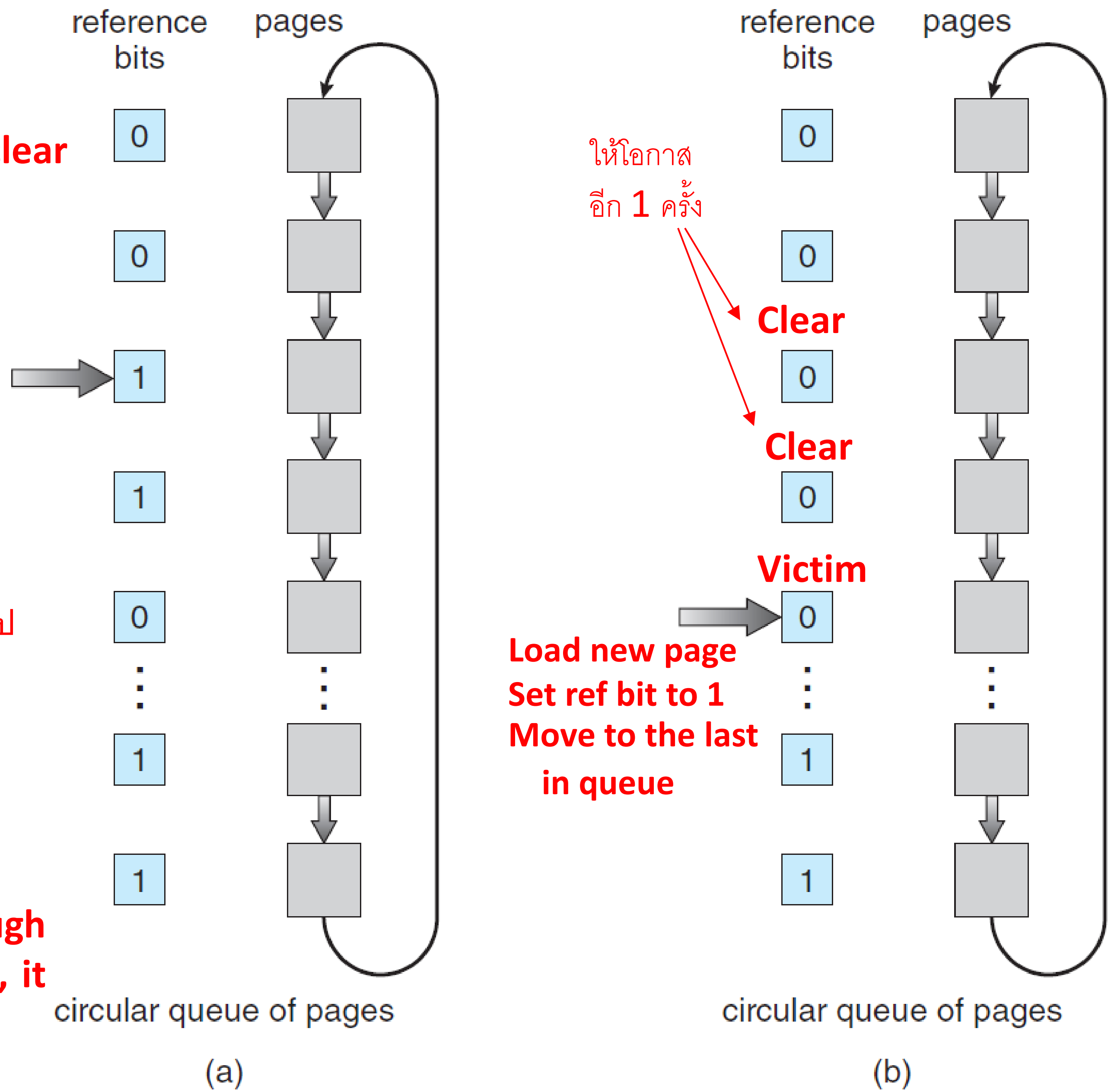
FIFO + Second Chance

Referenced Set ref. bit to 1

1 Give the second chance, clear

0 Replace

The first item in queue



Step

1. ใช้หลัก FIFO เริ่มจากหัวคิว
2. ถ้า ref bit = 0, ได้ victim แล้ว
   ถ้า ref bit = 1, clear ไปตัวถัดไป
3. อาจจะต้องวนมาเริ่มใหม่
   (ได้ second chance กันหมด)
4. ถ้าได้ victim แล้ว ให้โหลด new page มาทับ victim และให้ไปต่อท้ายคิว ตามหลักการ FIFO

If a page is used often enough to keep its reference bit set, it will never been replaced.

Enhanced second-chance algorithm (modify bit)

reference bit (0 คือเก่าแล้ว ได้ second chance ไปแล้ว)

modify bit (0 = not modified, 1 = modified)

first choice	(0, 0)	neither recently used nor modified – best page to replace
	(0, 1)	not recently used but modified – not quite as good and need writing disk
	(1, 0)	recently used but clean – probably will be used again soon
last choice	(1, 1)	recently used and modified – used again soon and need writing disk

ประโยชน์ของการเลือก **modify bit = 0** คือ **reduce I/O traffic.**

Allocation of Frames

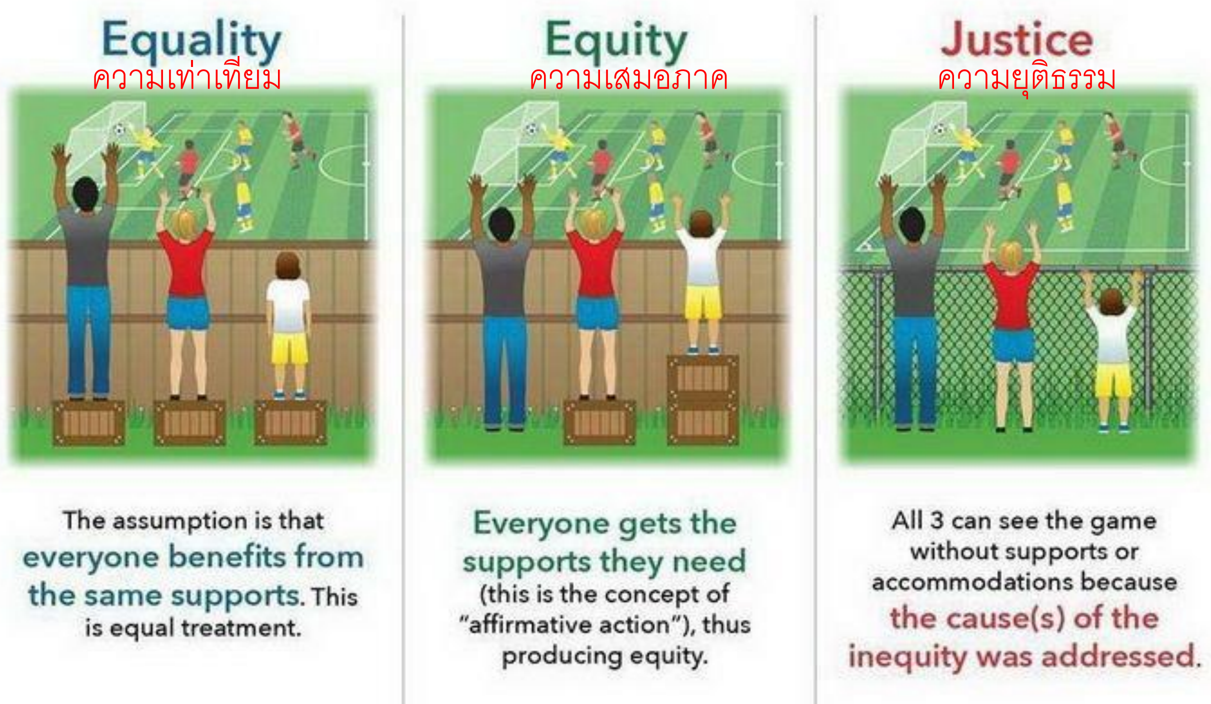
1) **Minimum number of frames**

- **Instruction set architecture:**

add a1 a2 a3	**min = 3**
ld r1 a4	**min = 1**

2) **Allocation algorithms**

- **Equal allocation** ให้ทุก **process** เท่า ๆ กัน
- **Proportional allocation** เกลี่ยให้ตามความต้องการใช้ **memory** ของแต่ละ **process**



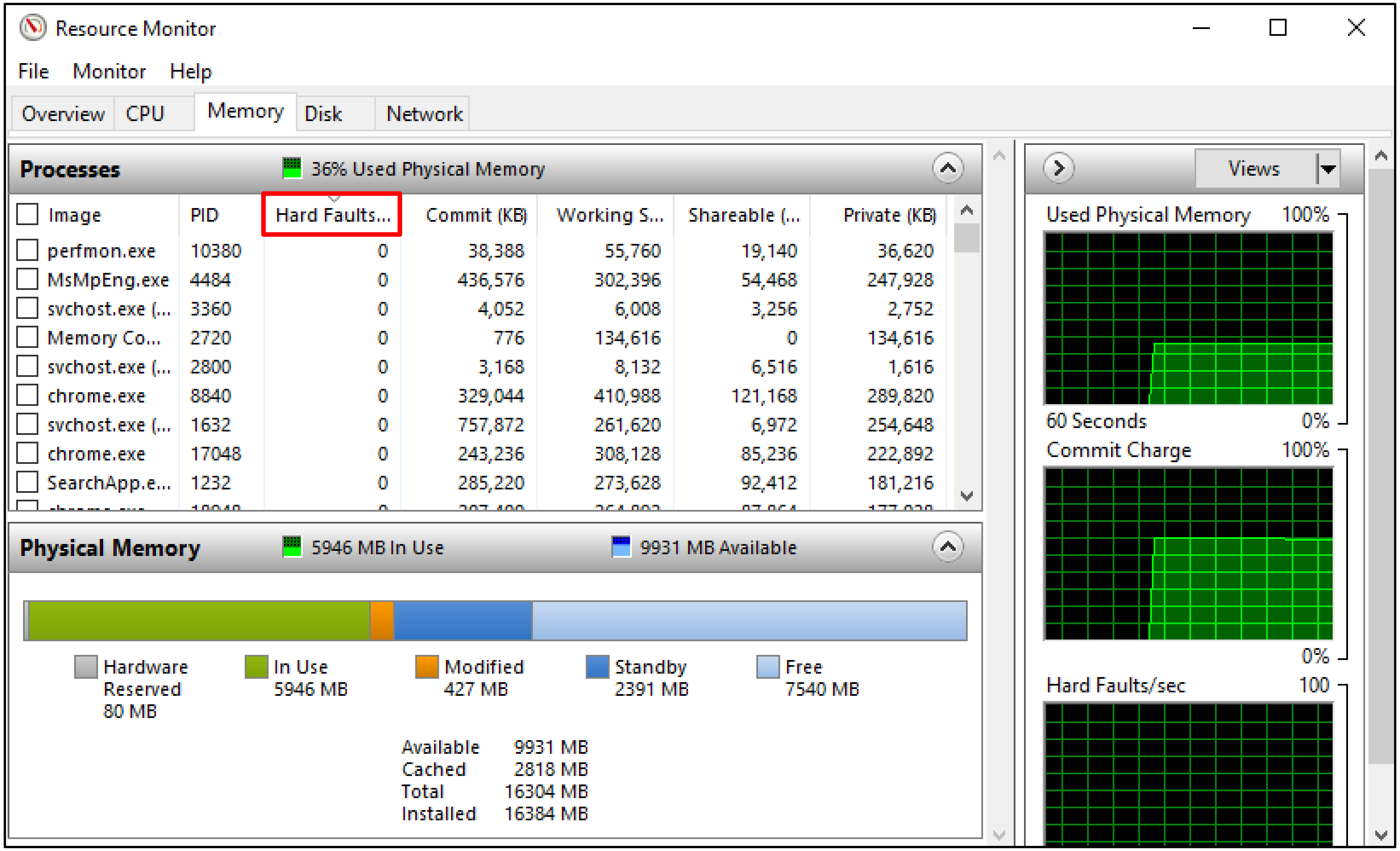
Global vs. Local allocation

1) Local allocation
   - a process uses ***max*** frames.
   - when requesting a free frame, choose a victim from its own set of allocated frame.
2) Global allocation
   - choose a victim from the set of all frames, even if that frame is currently allocated to some other process.

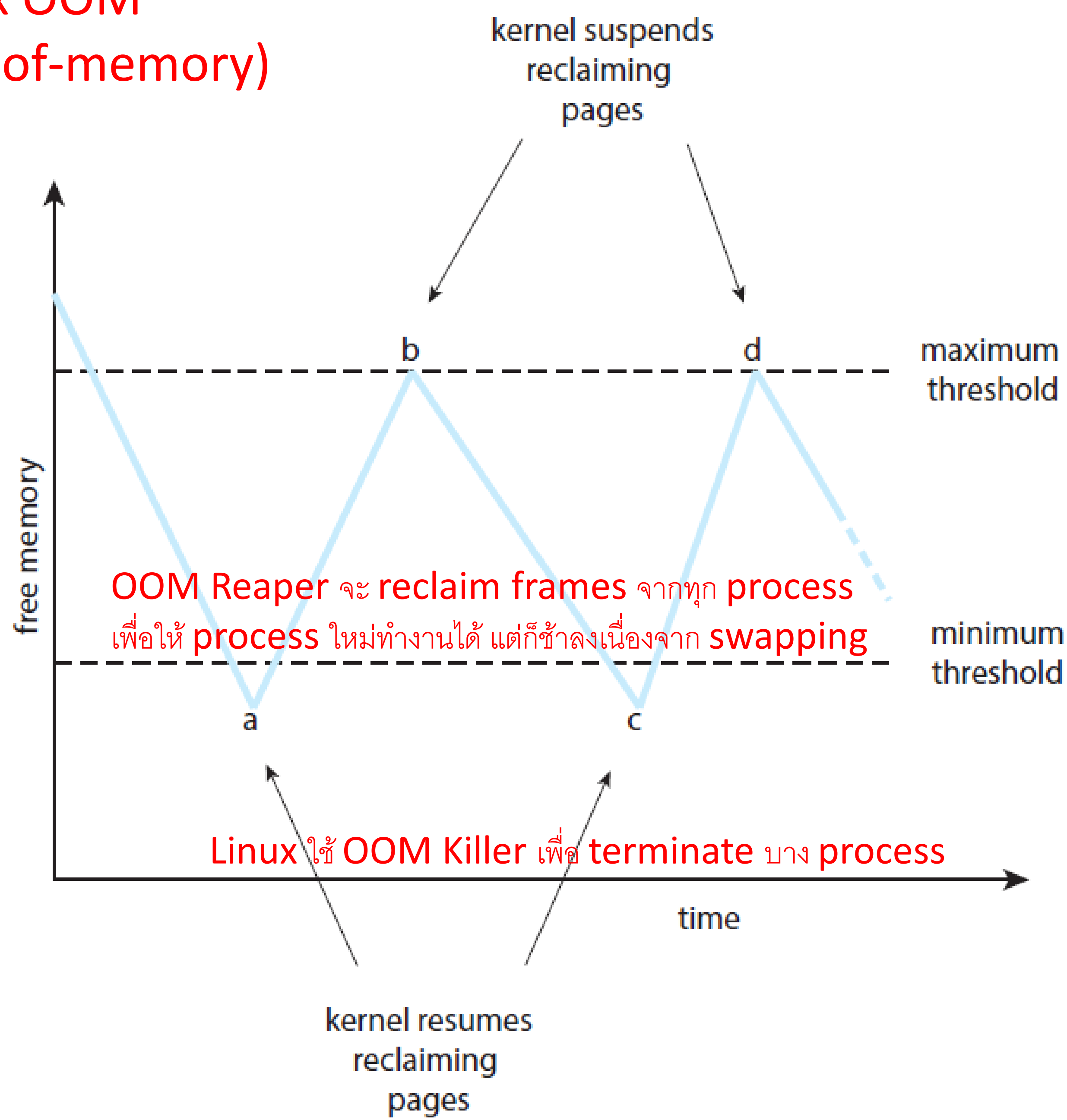
MAJOR AND MINOR PAGE FAULTS

As described in Section 10.2.1, a page fault occurs when a page does not have a valid mapping in the address space of a process. Operating systems generally distinguish between two types of page faults: **major** and **minor** faults. (Windows refers to major and minor faults as **hard** and **soft** faults, respectively.) A major page fault occurs when a page is referenced and the page is not in memory. Servicing a major page fault requires reading the desired page from the backing store into a free frame and updating the page table. Demand paging typically generates an initially high rate of major page faults.

Minor page faults occur when a process does not have a logical mapping to a page, yet that page is in memory. Minor faults can occur for one of two reasons. First, a process may reference a shared library that is in memory, but the process does not have a mapping to it in its page table. In this instance, it is only necessary to update the page table to refer to the existing page in memory. A second cause of minor faults occurs when a page is reclaimed from a process and placed on the free-frame list, but the page has not yet been zeroed out and allocated to another process. When this kind of fault occurs, the frame is removed from the free-frame list and reassigned to the process. As might be expected, resolving a minor page fault is typically much less time consuming than resolving a major page fault.

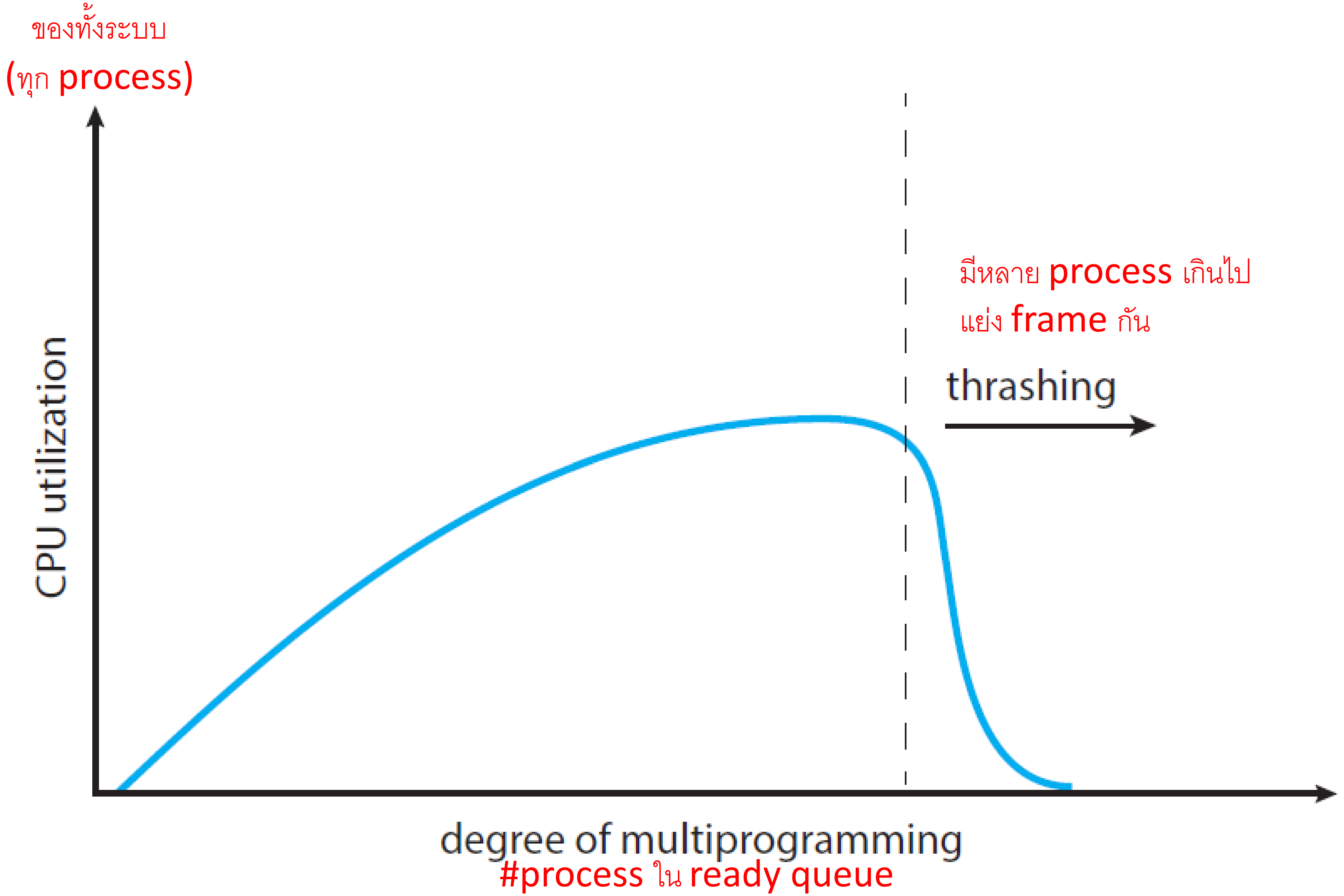

Resource Monitor
File Monitor Help
Overview CPU Memory Disk Network
Processes
36% Used Physical Memory
Image PID Hard Faults... Commit (KB) Working S... Shareable (... Private (KB)
perfmon.exe 10380 0 38,388 55,760 19,140 36,620
MsMpEng.exe 4484 0 436,576 302,396 54,468 247,928
svchost.exe (... 3360 0 4,052 6,008 3,256 2,752
Memory Co... 2720 0 776 134,616 0 134,616
svchost.exe (... 2800 0 3,168 8,132 6,516 1,616
chrome.exe 8840 0 329,044 410,988 121,168 289,820
svchost.exe (... 1632 0 757,872 261,620 6,972 254,648
chrome.exe 17048 0 243,236 308,128 85,236 222,892
SearchApp.e... 1232 0 285,220 273,628 92,412 181,216
Physical Memory
5946 MB In Use
9931 MB Available
Hardware Reserved 80 MB
In Use 5946 MB
Modified 427 MB
Standby 2391 MB
Free 7540 MB
Available 9931 MB
Cached 2818 MB
Total 16304 MB
Installed 16384 MB
Views
Used Physical Memory 100%
60 Seconds 0%
Commit Charge 100%
0%
Hard Faults/sec 100

Linux OOM (out-of-memory)



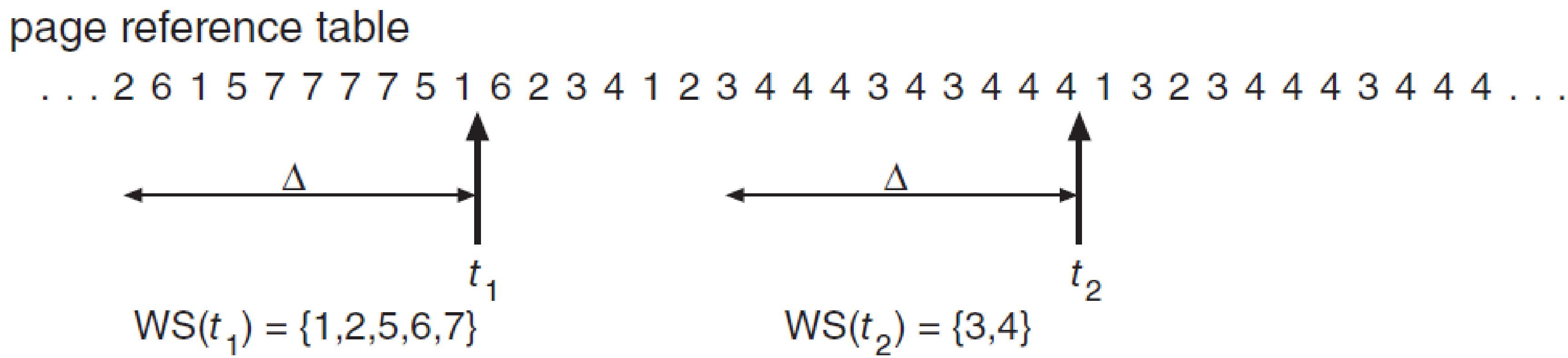
Thrashing

Definition: high paging activity.



Working-Set Model

Working set = set of pages in the most recent Δ page references.



The most important property of the working set, then, is its size. If we compute the working-set size, WSS_i, for each process in the system, we can then consider that

เช่น P1 ใช้ WSS = 8, P2 ใช้ WSS = 6

Demand $$D = \sum WSS_i,$$

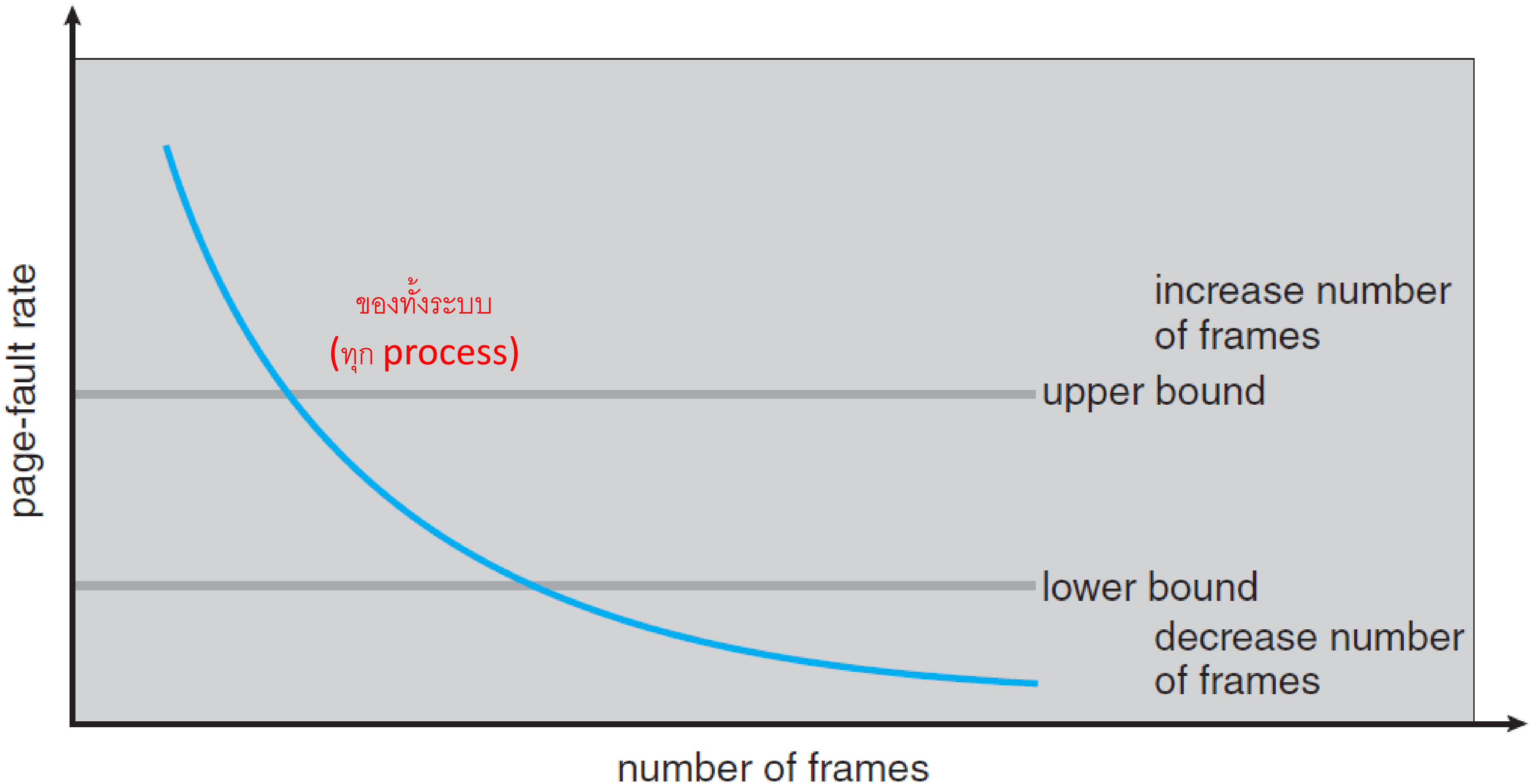
= 8 + 6 =14

where D is the total demand for frames. Each process is actively using the pages in its working set. Thus, process i needs WSS_i frames. If the total demand is greater than the total number of available frames ($D > m$), thrashing will occur, because some processes will not have enough frames.

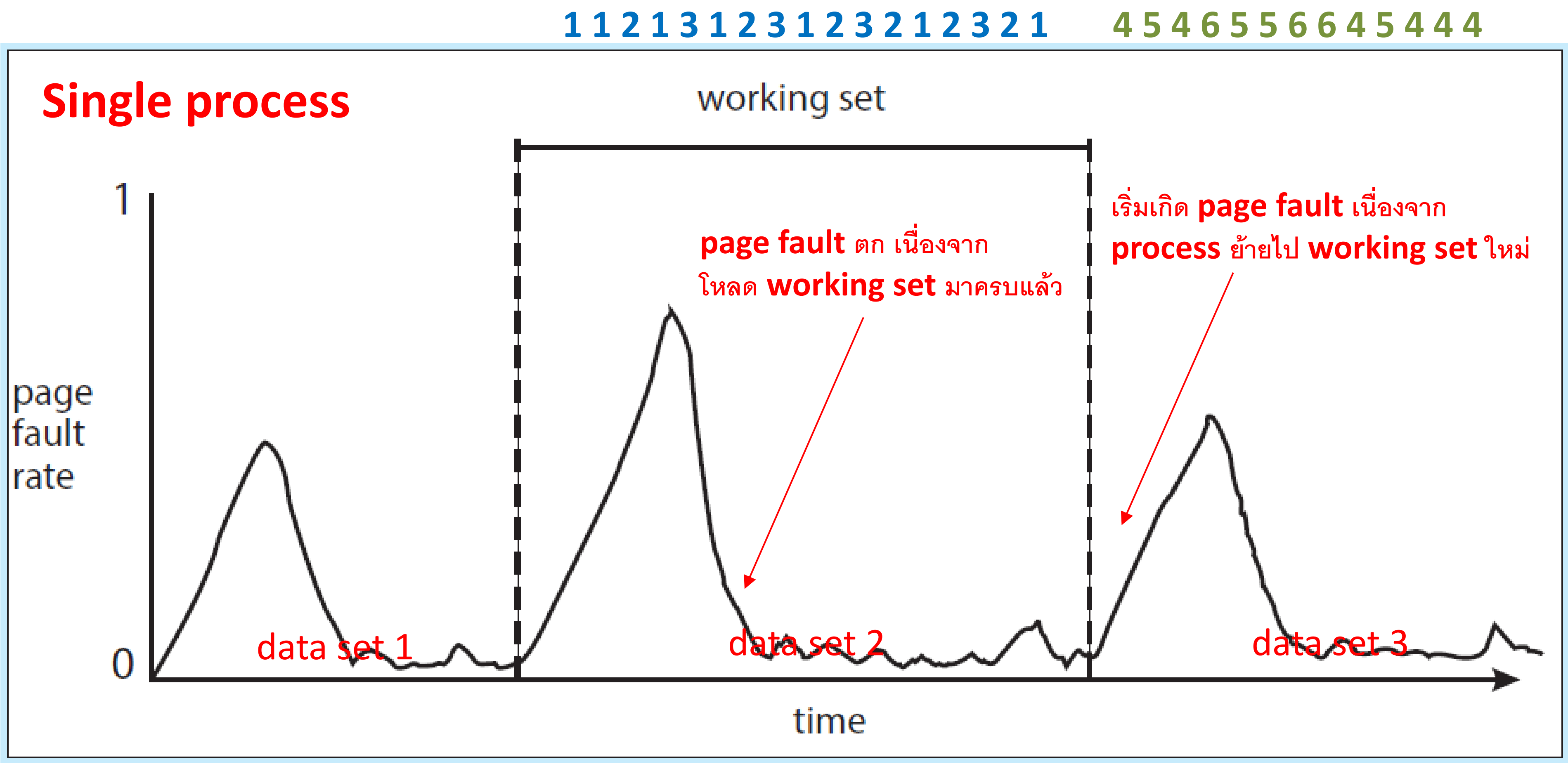
ถ้า demand > supply (allocated frames) ก็เพิ่ม frame ให้แต่ละ process (equally or proportionally)

แต่ถ้าไม่มี frame เหลือแล้ว ให้ลด degree of multiprogramming หรือลดจำนวน process ใน ready queue

Page-Fault Frequency



Working sets and page fault rates



Chapter 10 Virtual Memory