Addr ที่อยู่ใน RAM = Physical Addr , Virtual แยกชั้น จะมองว่า Process เริ่มที่ 0000

→ virtual addr
→ Logical addr

- มองเห็นทั้งหมด
- มองเห็นแค่นี้

No ไม่มองเห็น Phy addr.

Progr รู้ว่า ตัวเองอยู่ ตรงไหนของ mem by Check จาก Addr

- Virtual mem จุดเริ่มต้น ของ mem ต่างกัน

kernel เห็นหมด
- Phy addr
- Vir addr

(Vir addr → map → Phy addr)

* map เพื่อได้รู้ว่าอยู่ตรงไหน ของ phy addr.

Address Translation

kernel เห็นหมด
- Phy addr
- Vir addr

(Vir addr → map → Phy addr)

* map เพื่อได้รู้ว่าอยู่ตรงไหน ของ phy addr.

Main Points

พวกการ map ถือว่ามันคือ overhead
เพราะมันไม่ใช่ job (user สั่งการ)

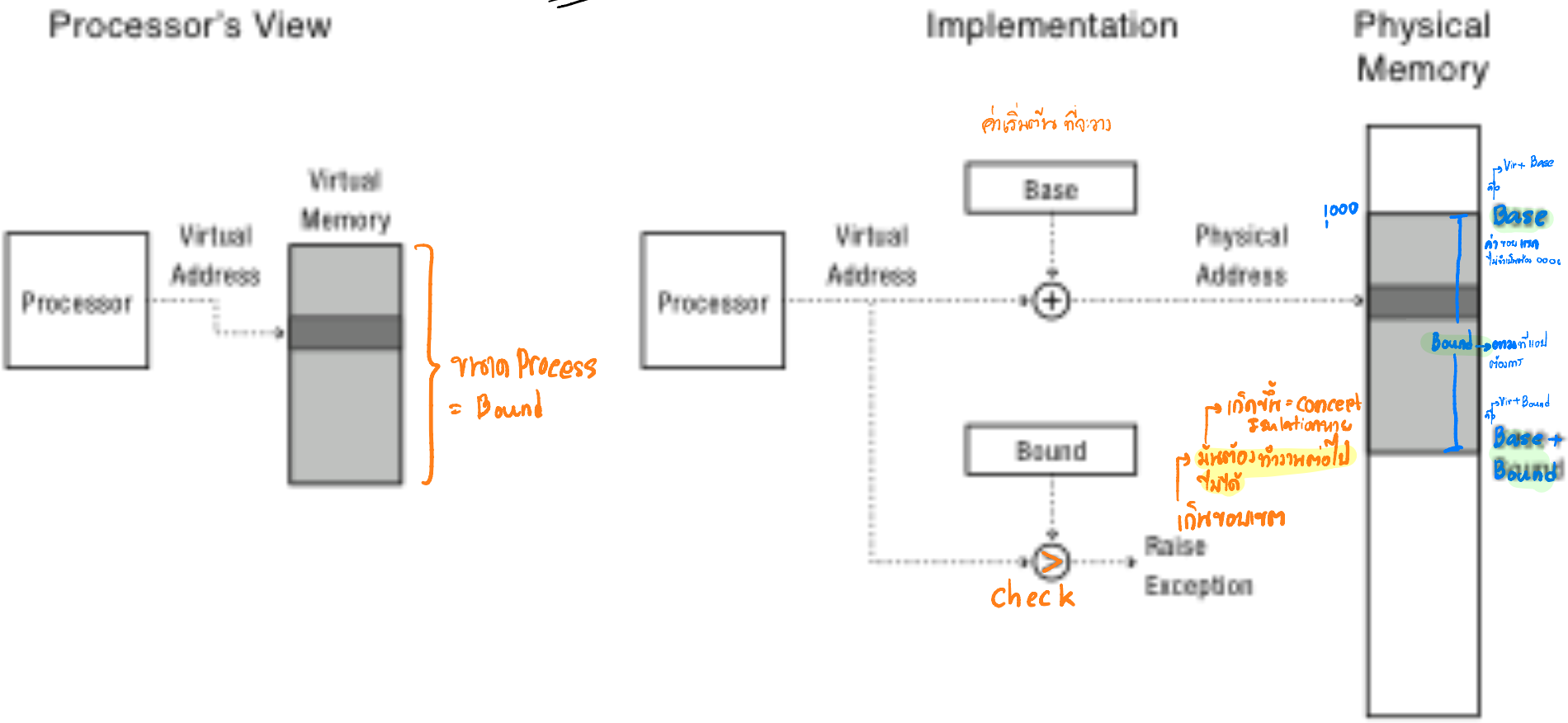
- Address Translation Concept คอนเซปท์ การพิจารณา เร็ว, ยืดหยุ่น
  - How do we convert a virtual address to a physical address?
- Flexible Address Translation 4 วิธีในการทำ addr Translation
  - Base and bound
  - Segmentation
  - Paging
  - Multilevel translation
- Efficient Address Translation การจัดการ เพิ่มประสิทธิภาพยังไงดี
  - Translation Lookaside Buffers
  - Virtually and physically addressed caches

Address Translation Goals

- Memory protection
- Memory sharing
  - Shared libraries, interprocess communication
- Sparse addresses
  - Multiple regions of dynamic allocation (heaps/stacks)
- Efficiency
  - Memory placement
  - Runtime lookup
  - Compact translation tables

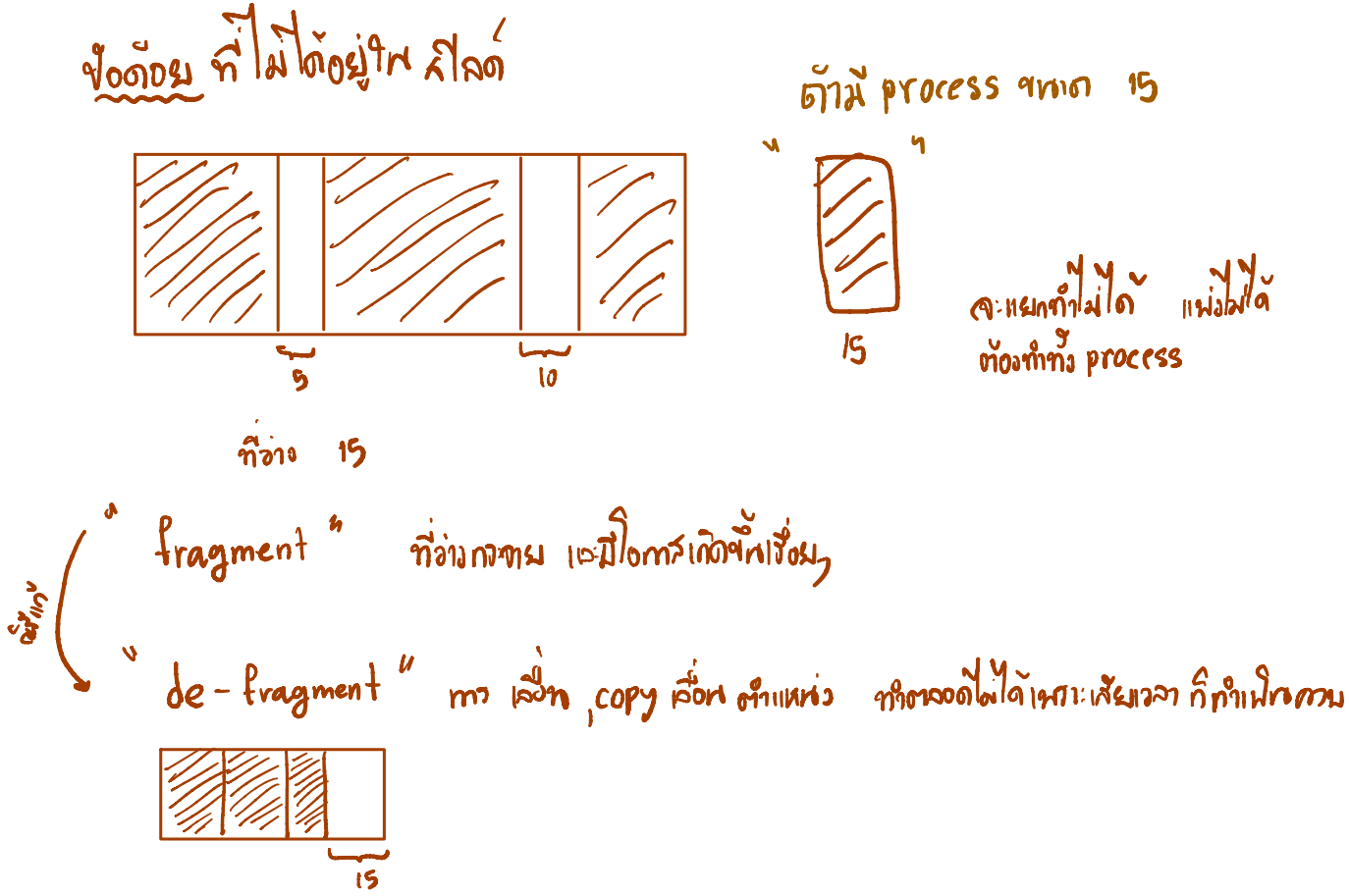
Virtually Addressed Base and Bounds

ข้อเสีย



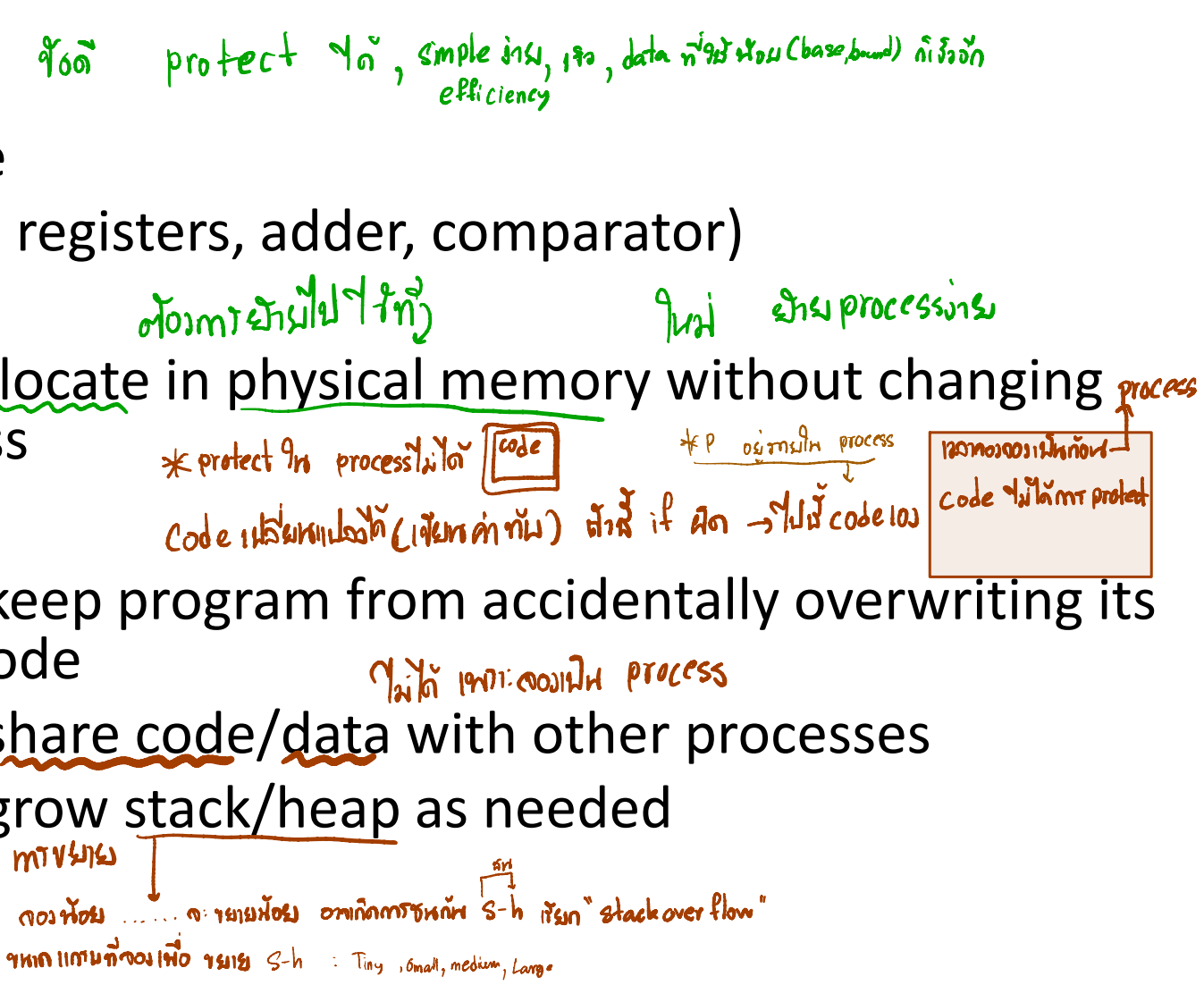
Activity #1

- Drawback of Base and Bounds



Virtually Addressed Base and Bounds

- Pros?
  - Simple
  - Fast (2 registers, adder, comparator)
  - Safe
  - Can relocate in physical memory without changing process
- Cons?
  - Can't keep program from accidentally overwriting its own code
  - Can't share code/data with other processes
  - Can't grow stack/heap as needed

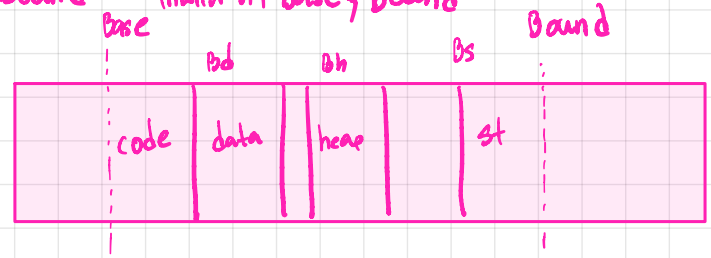


Segmentation

- Segment is a contiguous region of *virtual* memory
- Each process has a segment table (in hardware)
  - Entry in table = segment
- Segment can be located anywhere in physical memory
  - Each segment has: start, length, access permission
- Processes can share segments
  - Same start, length, same/different access permissions

Base - Bound

เก็บแค่ ค่า base, bound



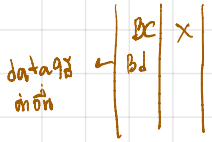
Segment

เก็บค่าเป็นตาราง

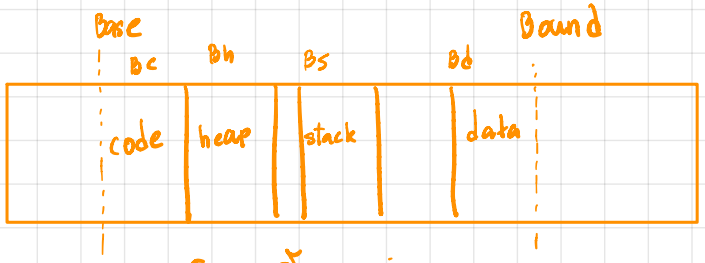
Bc	x	R
Bd	w	R/W
Bh	y	R/W
Bs	z	R/W

→ ถ้าไม่ตรงกับสิ่งนี้ เปลี่ยน แปลงไม่ได้ เพราะมันป้องกัน

ตัวอย่าง Share Bc ใช้ร่วมกัน



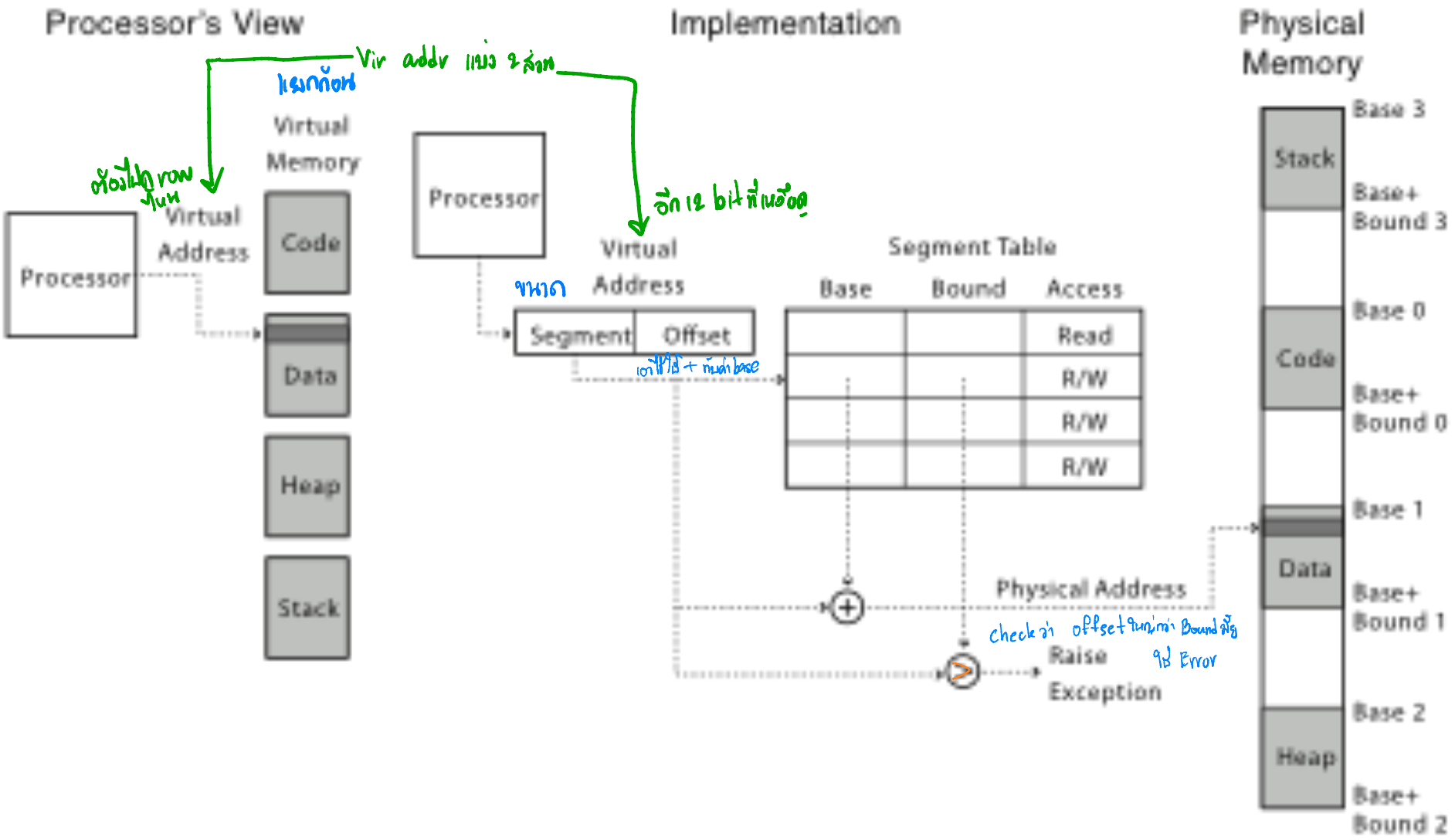
* สิ่งเปลี่ยนแปลงได้ Bd



แตก process เป็น 4 ส่วน เหมือนกับ segment

Base-Bound	Segment
x - Share	✓
x - protect	✓ ยกเว้น Internal segment

Segmentation



Ex. ตัวอย่าง

2 bit segment #
12 bit offset

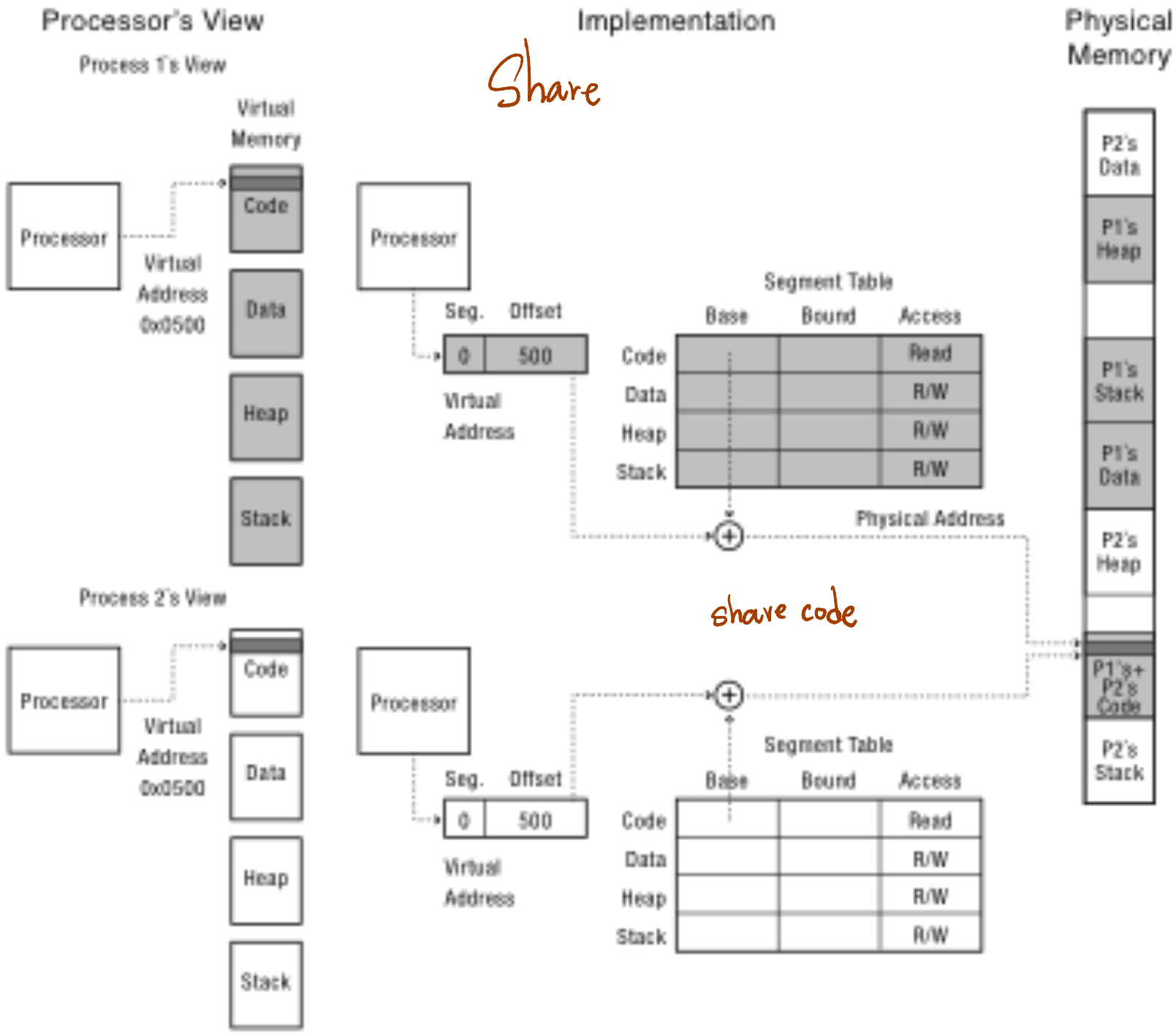
	Base: Segment start	Bound: length
code	0x4000 (เริ่มที่)	0x700
data	0	0x500
heap	-	-
stack	0x2000	0x1000

Virtual Memory

main: 240 h (ฐานสิบหก)	store #1108, r2
244 = 240+4	store pc+8, r31
248	jump 360
24c	
…	
strlen: 360	loadbyte (r2), r3
…	…
420	jump (r31)
…	
x: 1108	a b c \0
…	

Physical Memory

x: 108	a b c \0
…	
main: 4240 (ที่มา 4000b Bound 240h)	store #1108, r2 (4000 0001 0000 1000)
4244 = 4240+4	store pc+8, r31
4248	jump 360
424c	
…	…
strlen: 4360	loadbyte (r2),r3
…	
4420	jump (r31)
…	


Processor's View
Implementation
Physical Memory
Process 1's View
Virtual Memory
Share
Processor
Virtual Address 0x0500
Code
Data
Heap
Stack
Seg.
Offset
0
500
Virtual Address
Segment Table
Base
Bound
Access
Code
Data
Heap
Stack
Read
R/W
R/W
R/W
Physical Address
P2's Data
P1's Heap
P1's Stack
P1's Data
P2's Heap
P1's+ P2's Code
P2's Stack
Process 2's View
share code

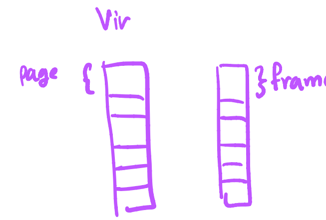
Activity #2

- Drawback of Segmentation

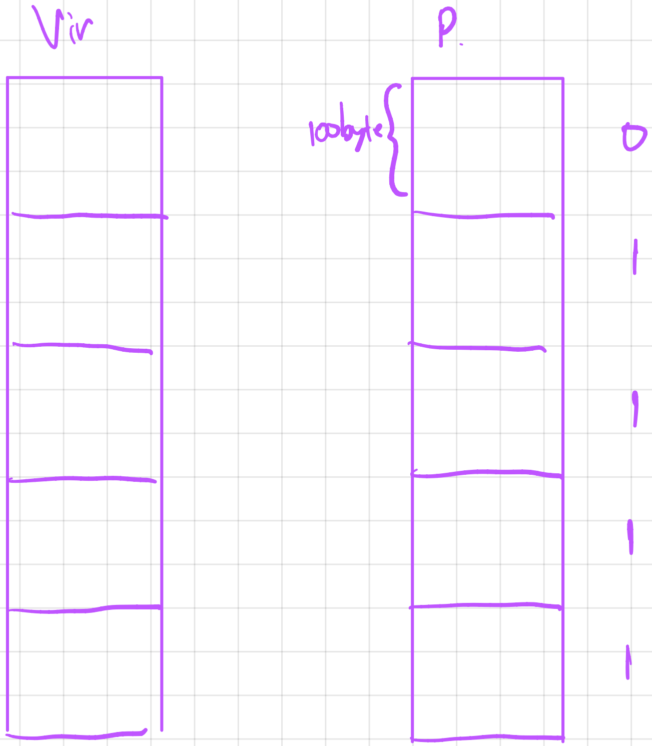
Segmentation

- Pros?
  - Can share code/data segments between processes
  - Can protect code segment from being overwritten
  - Can transparently grow stack/heap as needed
  - Can detect if need to copy-on-write
- Cons?
  - Complex memory management
    - Need to find chunk of a particular size
  - May need to rearrange memory from time to time to make room for new segment or growing segment
    - External fragmentation: wasted space between chunks

Paged Translation



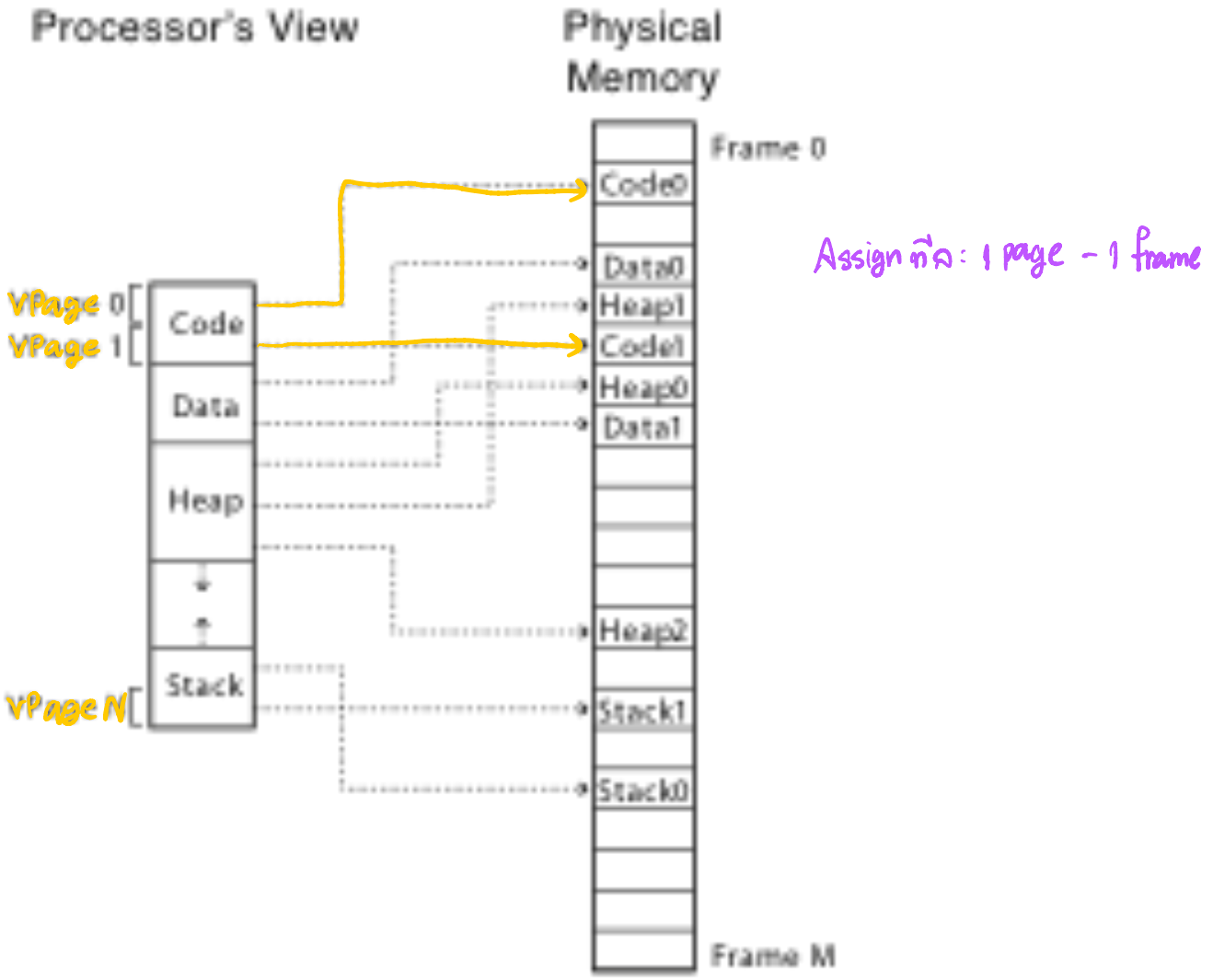
- Manage memory in fixed size units, or pages
- Finding a free page is easy
  - Bitmap allocation: 0011111100000001100
  - Each bit represents one physical page frame
- Each process has its own page table
  - Stored in physical memory
  - Hardware registers
    - pointer to page table start
    - page table length



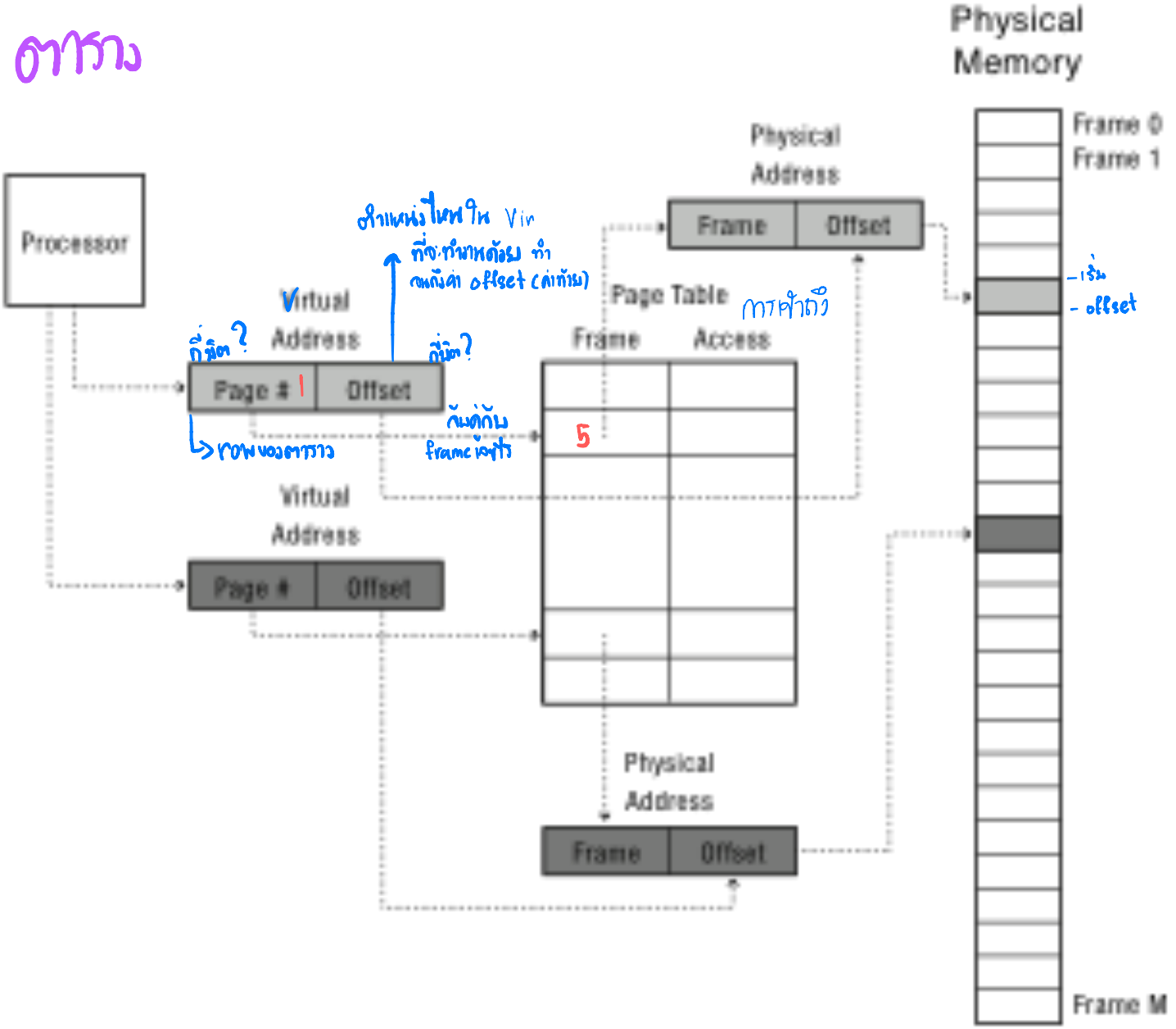
1 0
ใช้บิตกำหนดว่าง / ไม่ว่าง → แล้วแต่ว่า เครื่อง / คนทำกำหนด

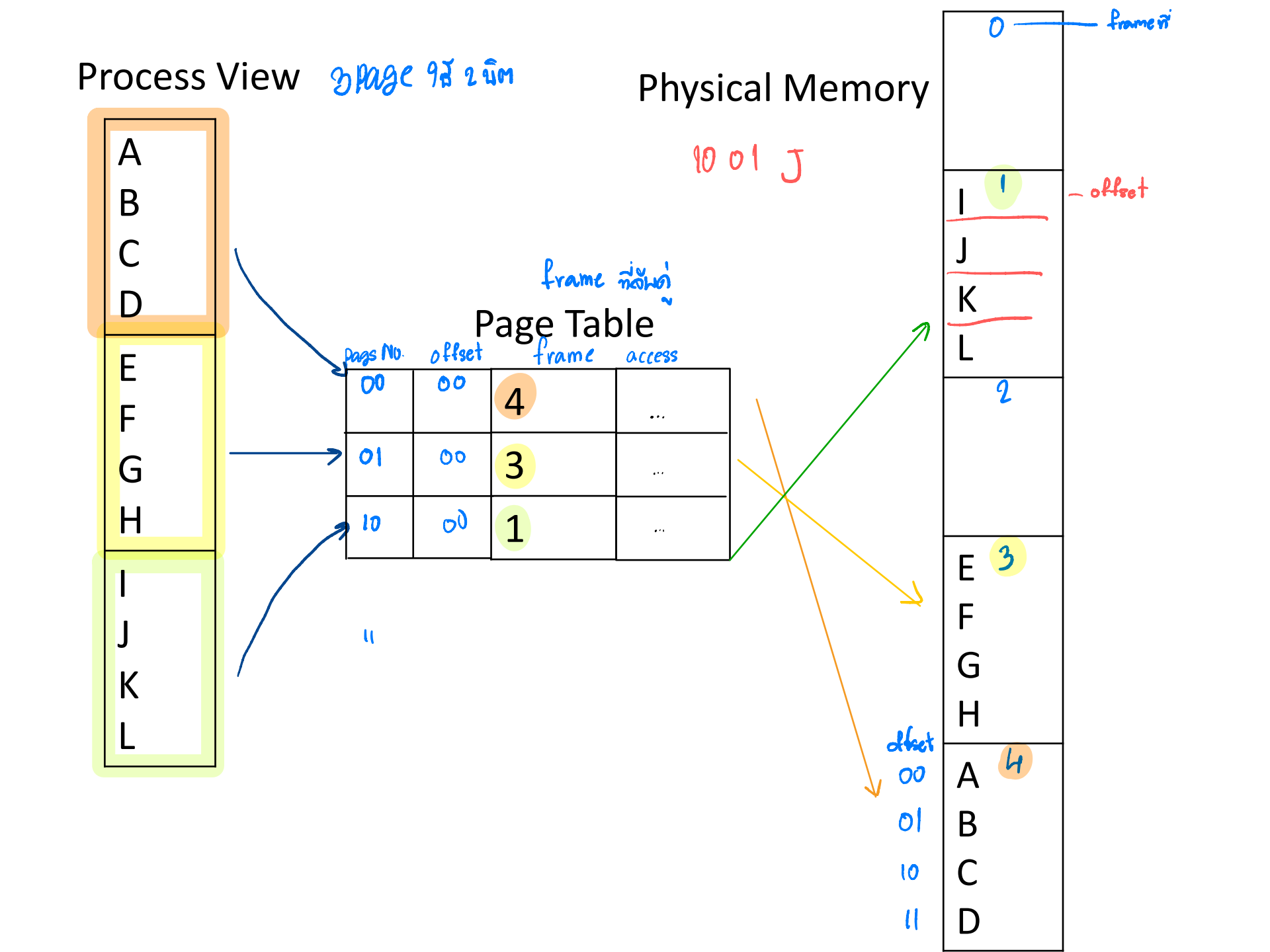
↳ แทนที่ scan ทั้งหมด ก็มา scan bit แทน เรา ที่รู้ 4 bit
แทนที่ scan 100 byte

Paged Translation (Abstract)



Paged Translation (Implementation)




Process View
3 page ขนาด 2 บิต
Physical Memory
10 01 J
A
B
C
D
E
F
G
H
I
J
K
L
frame ที่เข้าหา
Page Table
page No.
offset
frame
access
00
00
4
...
01
00
3
...
10
00
1
...
11
0
frame ที่
1
offset
I
J
K
L
2
3
E
F
G
H
4
A
B
C
D
offset
00
01
10
11

Activity #3

- Drawback of paging

ข้อดี scan เร็ว, เร็วๆการ allocation, การจัดการ segment ที่ขนาดตัว ก็ไป page ที่ว่าง แล้ว assign เข้าไป

ข้อเสีย

Sparse Address Spaces

- Might want many separate dynamic segments
  - Per-processor heaps
  - Per-thread stacks ทำได้ง่าย แตก thr แต่อันที่ต้องการ stack ของตัวเอง ก็หาที่ใส่ได้
  - Memory-mapped files
  - Dynamically linked libraries
- What if virtual address space is large?
  - 32-bits, 4KB pages => 500K page table entries
  - 64-bits => 4 quadrillion (พันล้าน) page table entries

ถ้า process ต้องการพื้นที่มาก pages ก็ใหญ่ขึ้น page table เก็บใน frame ถ้าใหญ่เกิน 1 frame ก็ทำงานแยกส่วนกัน

ก็ต้องย้ายส่วนหลังทำ page ที่เสร็จ

ใน 1page ประสิทธิภาพการใช้พื้น RAM ใช้ได้ไม่เต็มที่

เพราะ 1 segment บางที่ใช้ไม่ครบ page แต่ยังคงพื้นที่ไปใช้ไม่ได้ด้วย

แลกกับความเร็วในการหาว่าตัวไหนอ่าน

Multi-level Translation

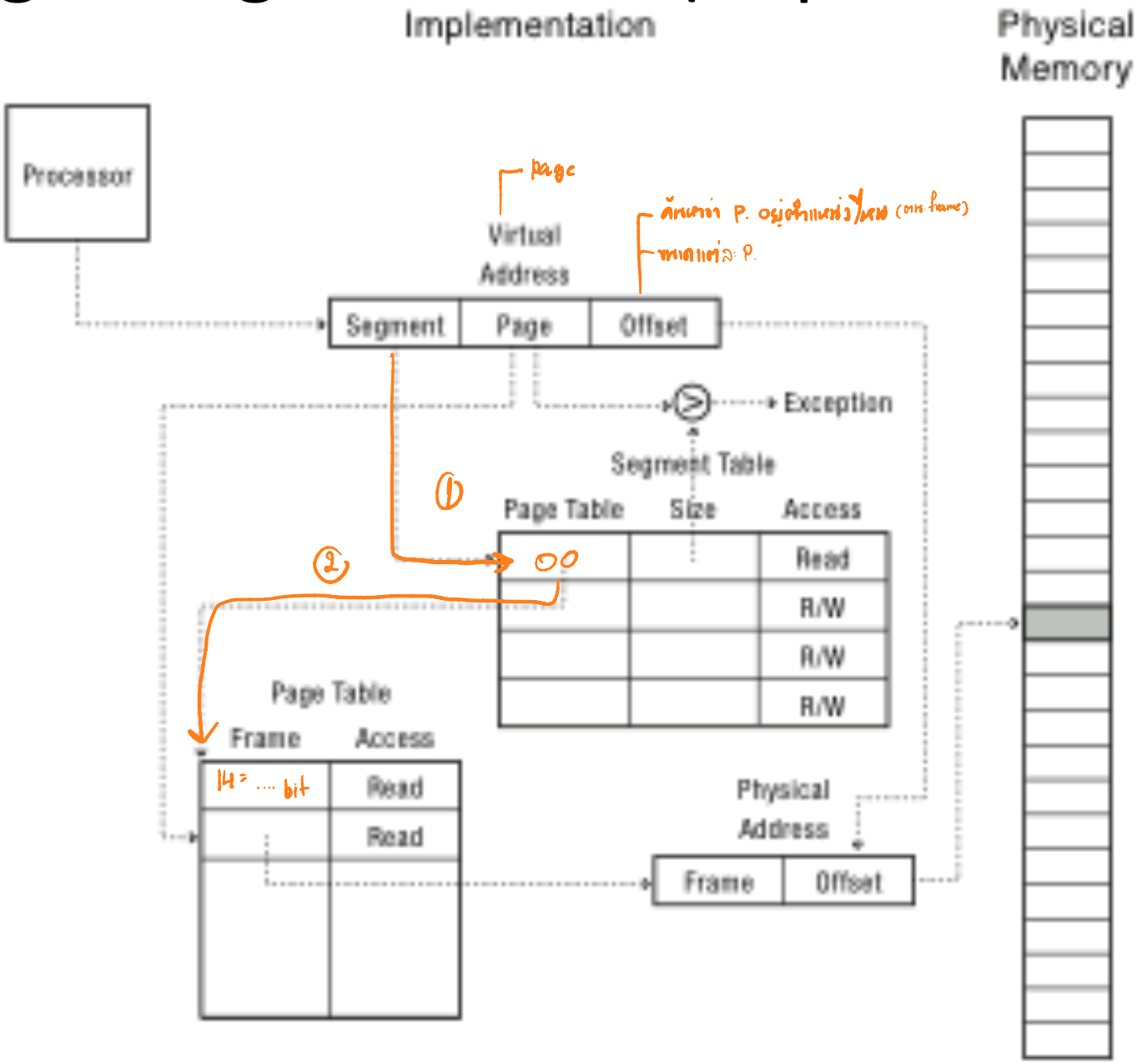
- Tree of translation tables
  - Paged segmentation
  - Multi-level page tables
  - Multi-level paged segmentation
- Fixed-size page as lowest level unit of allocation
  - Efficient memory allocation (compared to segments)
  - Efficient for sparse addresses (compared to paging)
  - Efficient disk transfers (fixed size units)
  - Easier to build translation lookaside buffers
  - Efficient reverse lookup (from physical -> virtual)
  - Variable granularity for protection/sharing

แต่ page ไม่ fix ขนาดเลิว

Paged Segmentation

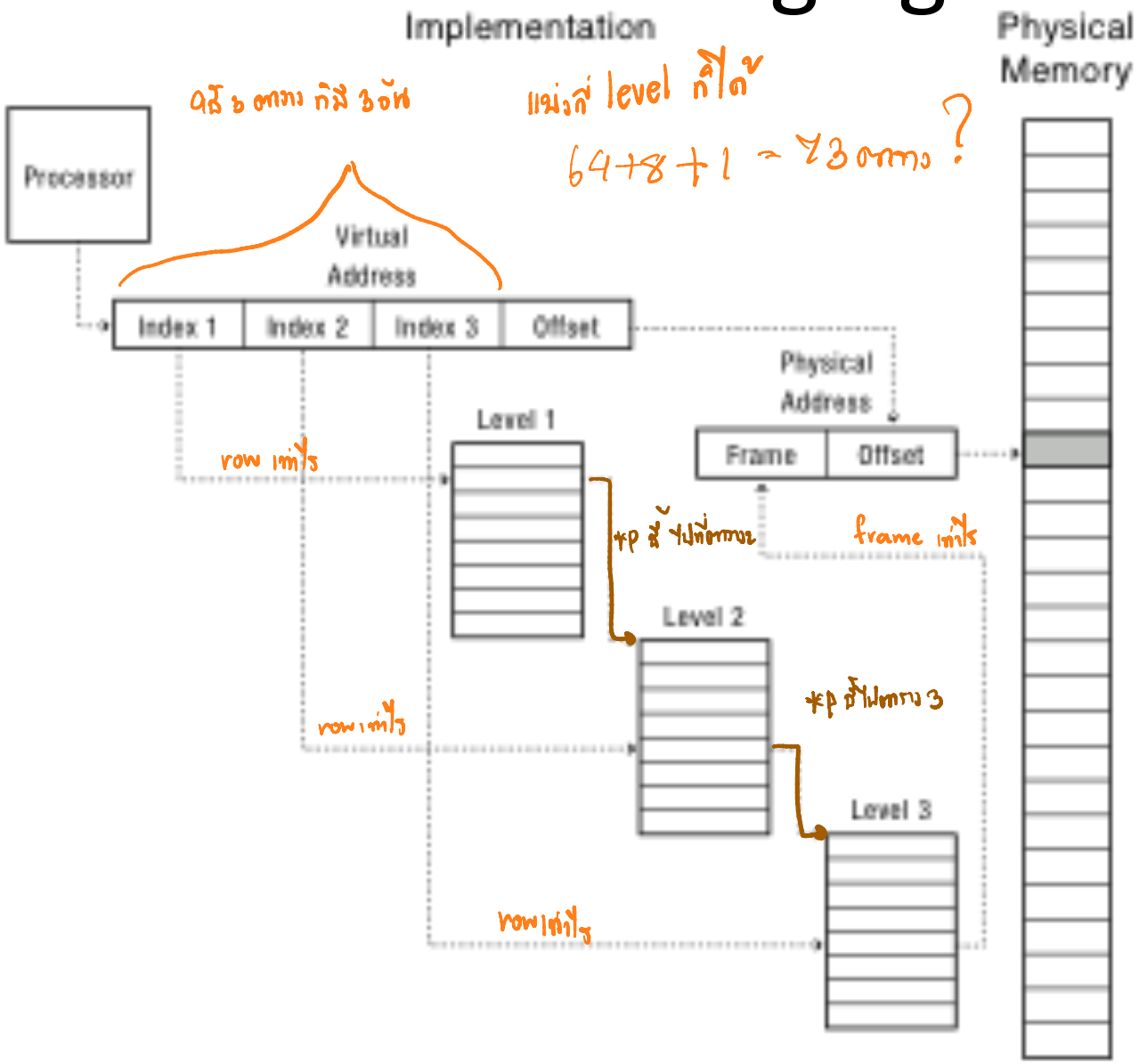
- Process memory is segmented
- Segment table entry:
  - Pointer to page table
  - Page table length (# of pages in segment)
  - Access permissions
- Page table entry:
  - Page frame
  - Access permissions
- Share/protection at either page or segment-level

Paged Segmentation (Implementation)



แตก หลาย table
ทำให้ ขนาด table ไม่เกิน 1 frame

Multilevel Paging



Activity #4

- Drawback of Multilevel translation

ด้อย - วิธีเกิบซับซ้อน มีหลายขั้นตอน ช้า
- ใช้พื้นที่หน่วยความจำที่ RAM ไม่ดี

Multilevel Translation

- Pros:
  - Allocate/fill only page table entries that are in use
  - Simple memory allocation ✓
  - Share at segment or page level
- Cons:
  - Space overhead: one pointer per virtual page
  - Two (or more) lookups per memory reference

ก็เก็บข้อมูลของ page มาด้วย

- มีหลายชั้นหลายทาง
- ดูประสิทธิภาพ RAM ไม่ดี

Activity #5

- Accelerate multilevel translation
  - Possible? ได้
  - How? จำ cache เข้าไป

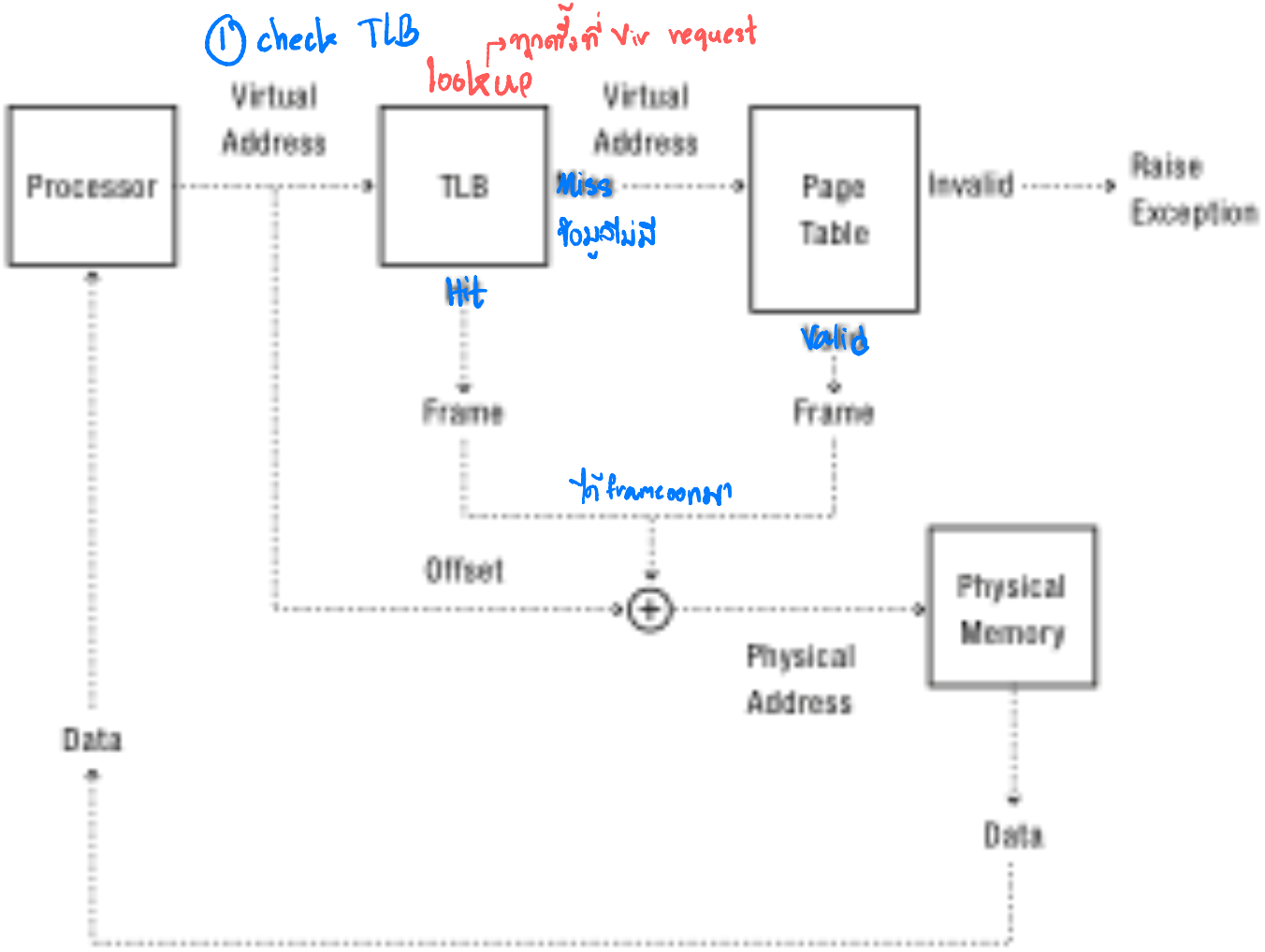
Efficient Address Translation

- Translation lookaside buffer (TLB)
  - Cache of recent virtual page -> physical page translations
  - If cache hit, use translation
  - If cache miss, walk multi-level page table
- Cost of translation =

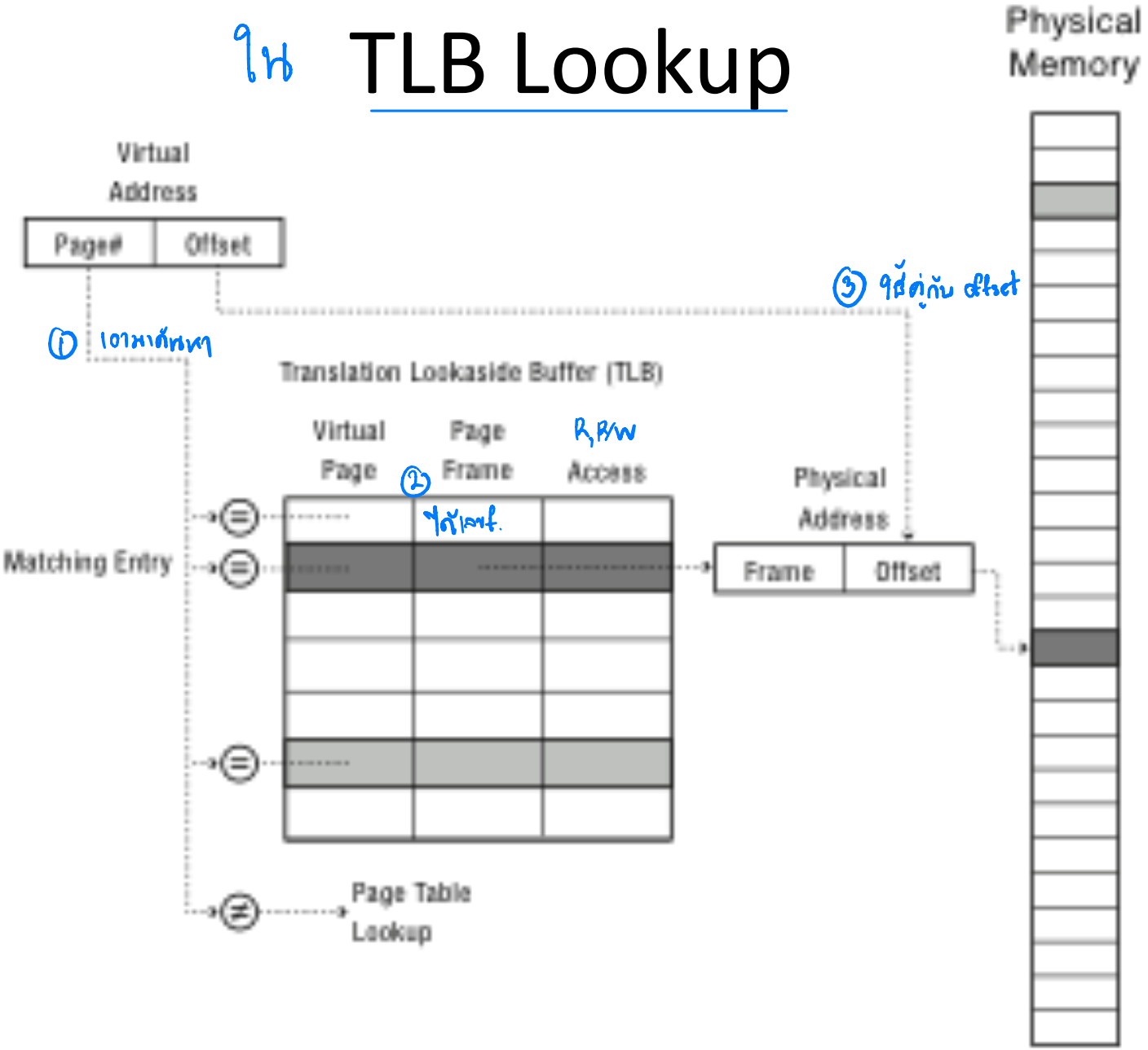
  Cost of TLB lookup +

  Prob(TLB miss) * cost of page table lookup

TLB and Page Table Translation



ใน TLB Lookup



สรุป อ่านเอา

Address Translation Uses

- Process isolation
  - Keep a process from touching anyone else's memory, or the kernel's
- Efficient interprocess communication
  - Shared regions of memory between processes
- Shared code segments
  - E.g., common libraries used by many different programs
- Program initialization
  - Start running a program before it is entirely in memory
- Dynamic memory allocation
  - Allocate and initialize stack/heap pages on demand

Address Translation (more)

- Cache management
  - Page coloring
- Program debugging
  - Data breakpoints when address is accessed
- Zero-copy I/O
  - Directly from I/O device into/out of user memory
- Memory mapped files
  - Access file data using load/store instructions
- Demand-paged virtual memory
  - Illusion of near-infinite memory, backed by disk or memory on other machines