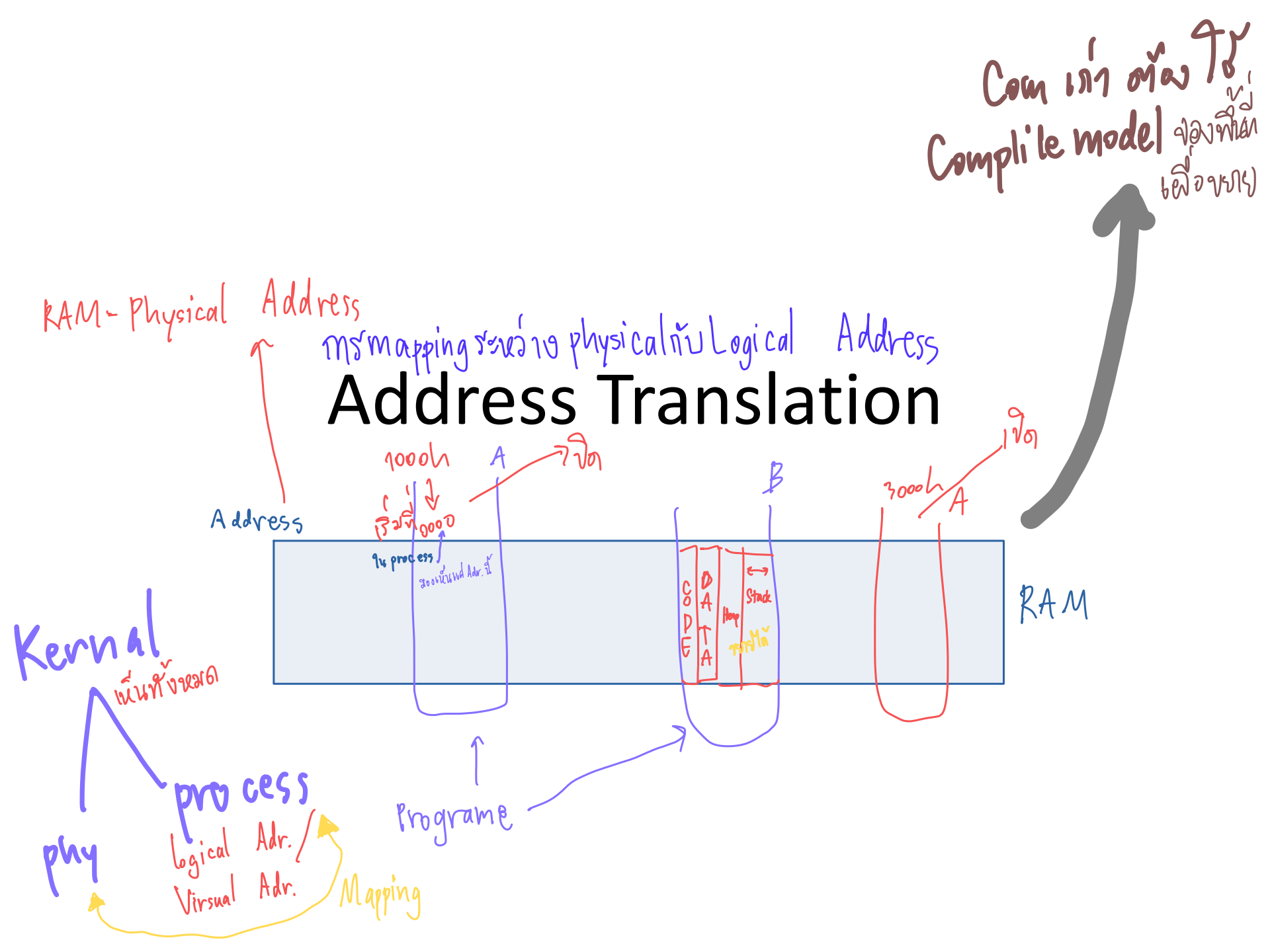
Address Translation



Main Points

- Address Translation Concept
  - How do we convert a virtual address to a physical address?
- Flexible Address Translation
  - Base and bound
  - Segmentation
  - Paging
  - Multilevel translation
- Efficient Address Translation
  - Translation Lookaside Buffers
  - Virtually and physically addressed caches

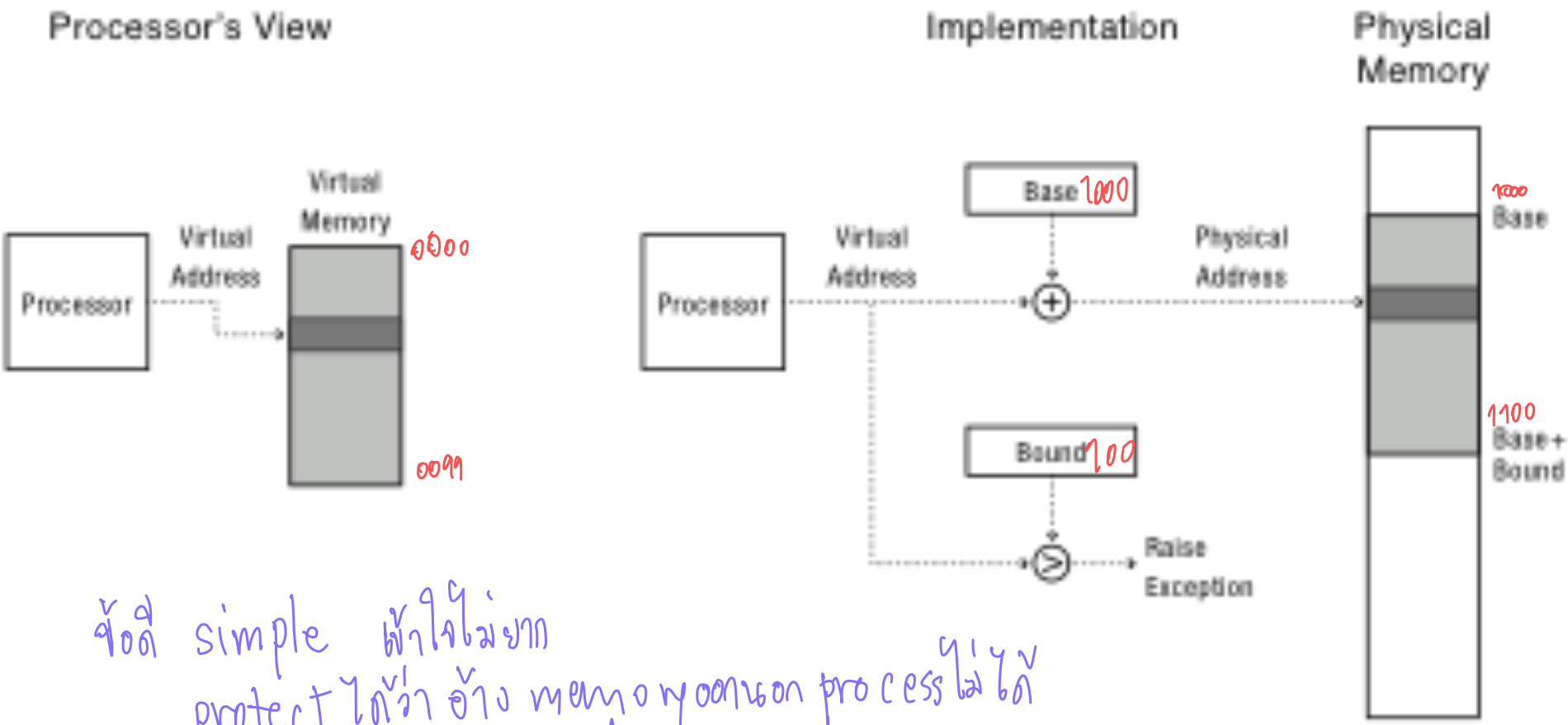
Address Translation Goals

- 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

ตำแหน่งเริ่มต้นที่ assign ให้กับ process

Virtually Addressed Base and Bounds

ขนาด



ข้อดี simple เข้าใจไม่ยาก
protect ได้ว่า อ้าง memory ของ process ไม่ได้
ใช้ข้อมูลน้อย ทำงานได้เร็ว

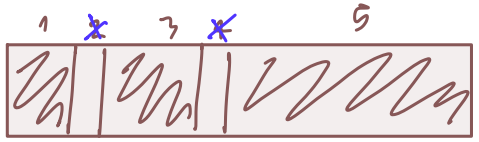
ข้อเสีย เกิด fragmentation ช่องว่าง ไม่ defragment ยาก / Share code ของ process ไม่ได้
Grow stack heap ที่ต้องการไม่ได้

Activity #1

- Drawback of Base and Bounds

Virtually Addressed Base and Bounds

ต้องเป็นสีนเดียวกัน Allocate ไม่ได้



ข้อด้อยยังมีอีก

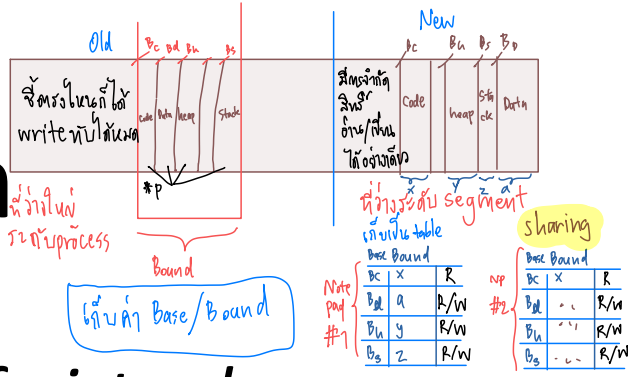
วิธีแก้ต้องเลื่อน 15 unit จะได้หมด

สูญเสียครั้งเดียว [de-fragment] เสียเวลาต้องรอรอบ copy → เลื่อน

- Pros? ข้อดี
  - Simple simple เข้าใจไม่ยาก
  - Fast (2 registers, adder, comparator) ใช้ข้อมูลน้อย ทำงานได้เร็ว
  - Safe
  - Can relocate in physical memory without changing process protect ได้ว่า อ้าง memory ของ process ไม่ได้
- Cons? ข้อเสีย
  - Can't keep program from accidentally overwriting its own code pointer ชี้ผิด เขียนทับ code เดิม เกิด A. เสียหายได้
  - Can't share code/data with other processes แชร์ code ไม่ได้
  - Can't grow stack/heap as needed ขยายไม่ได้

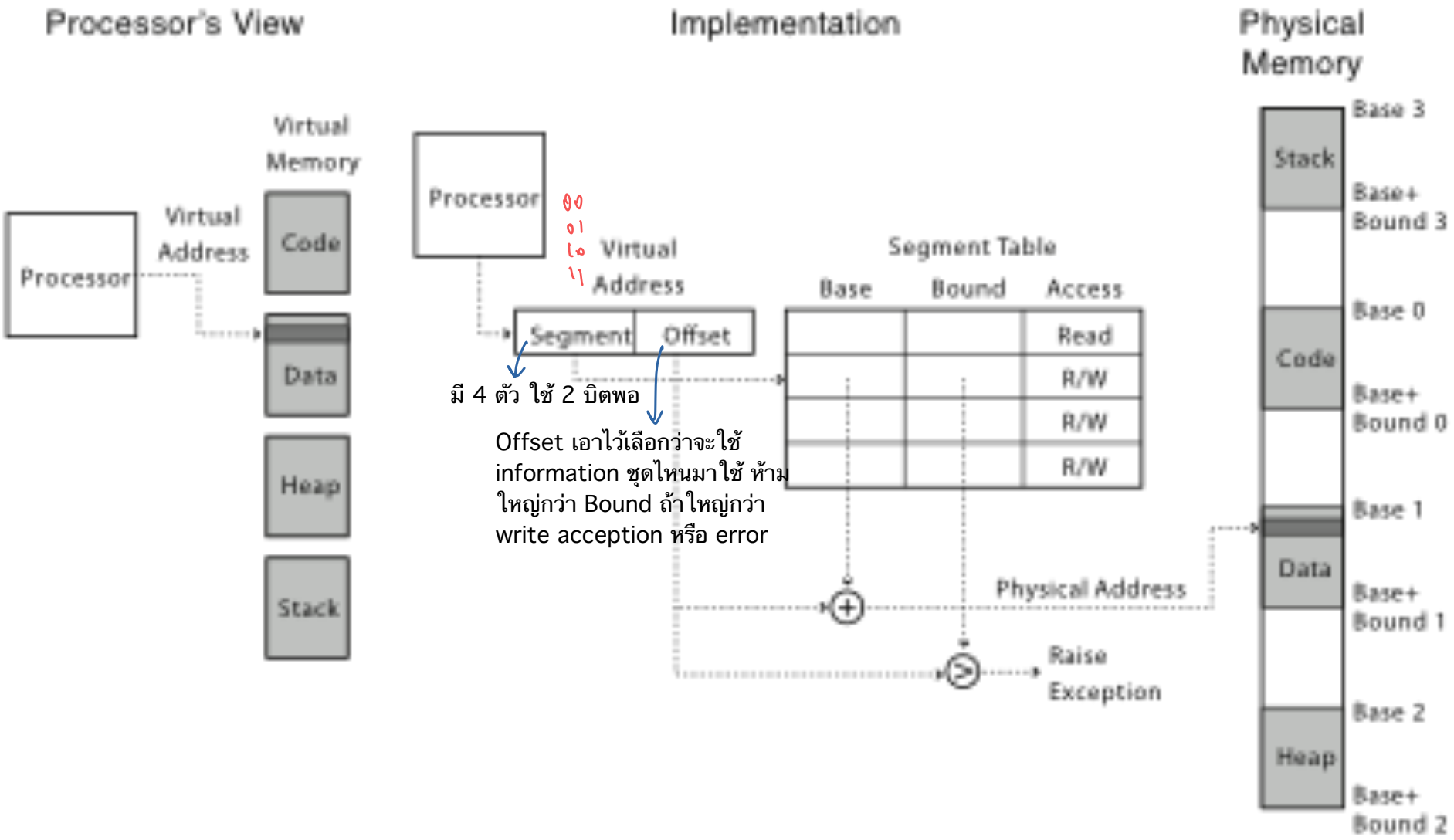
เพราะไม่รู้ว่า Code Data Heap Stack อยู่ตรงไหน เลย protect ไม่ได้

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

Segmentation



2 bit segment #
12 bit offset

	Segment start	length
code	0x4000	0x700
data	0	0x500
heap	-	-
stack	0x2000	0x1000

2A0
1 4H +
ระวังการบวกเลขฐาน 16!!!!!!

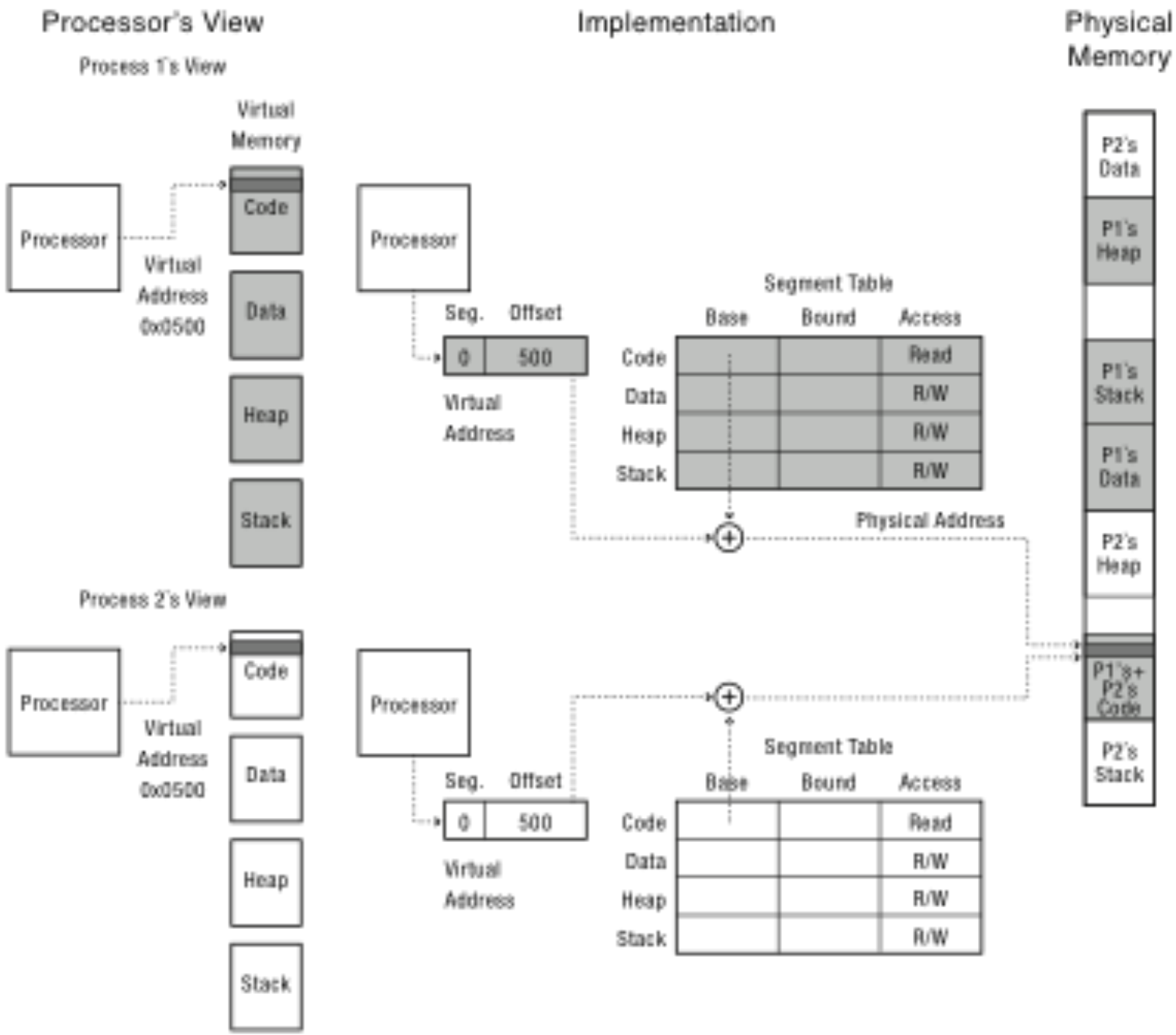
Virtual Memory

แปลงเป็นเลขฐาน 2 = 00 0010 0100 0000

main: 240 ฐาน 16	store #1108, r2
244	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	store #1108, r2
4244	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
Processor
Virtual Address 0x0500
Code
Data
Heap
Stack
Segment Table
Base
Bound
Access
Seg.
Offset
0
500
Virtual Address
Read
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

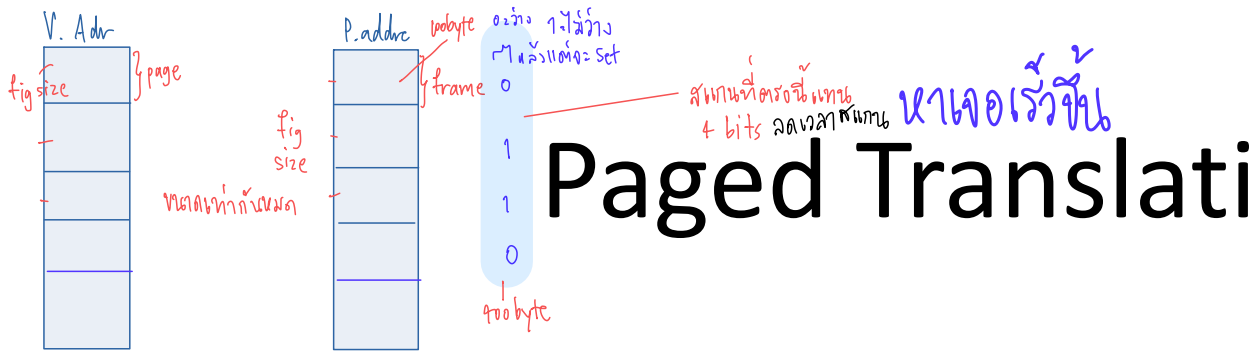
Activity #2

ข้อด้อย

- Drawback of Segmentation

Segmentation

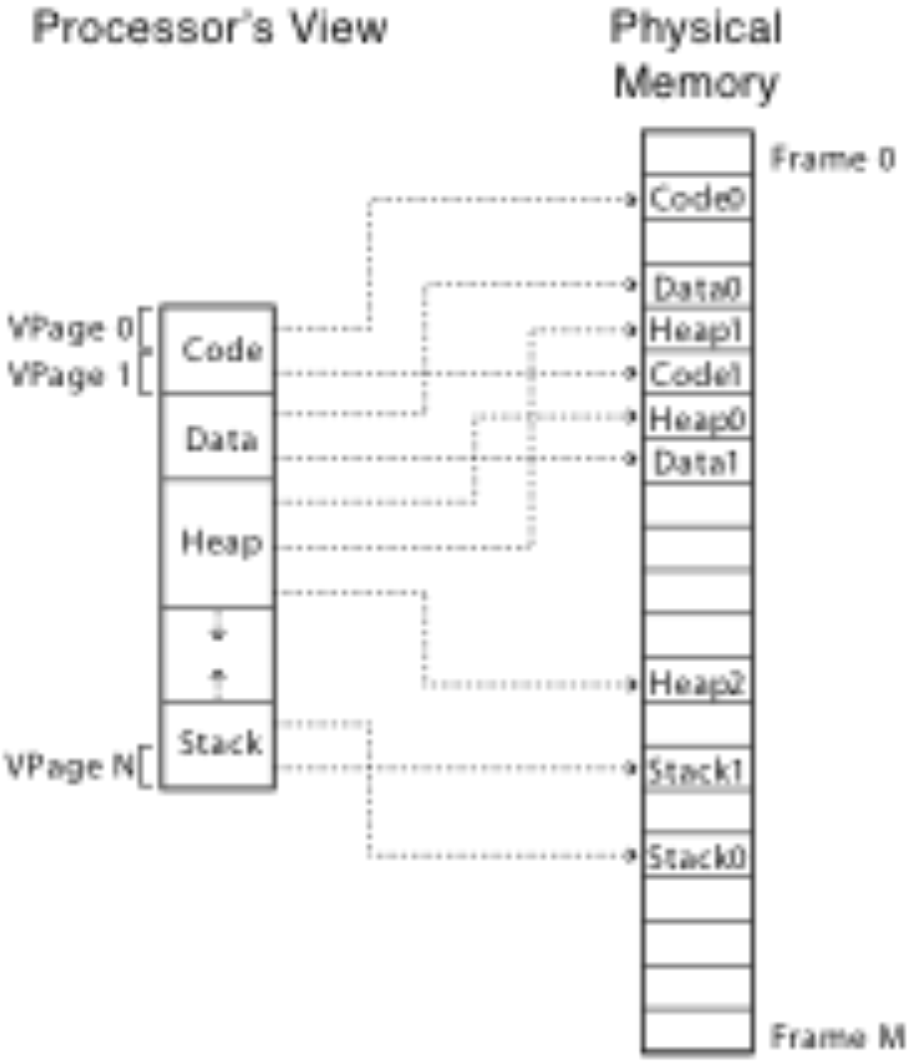
- Pros?
  - Can share code/data segments between processes
  - Can protect code segment from being overwritten
  - Can transparently grow stack/heap as needed
  - Can detect if need to copy-on-write
- Cons?
  - Complex memory management
    - Need to find chunk of a particular size
  - May need to rearrange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



Paged Translation

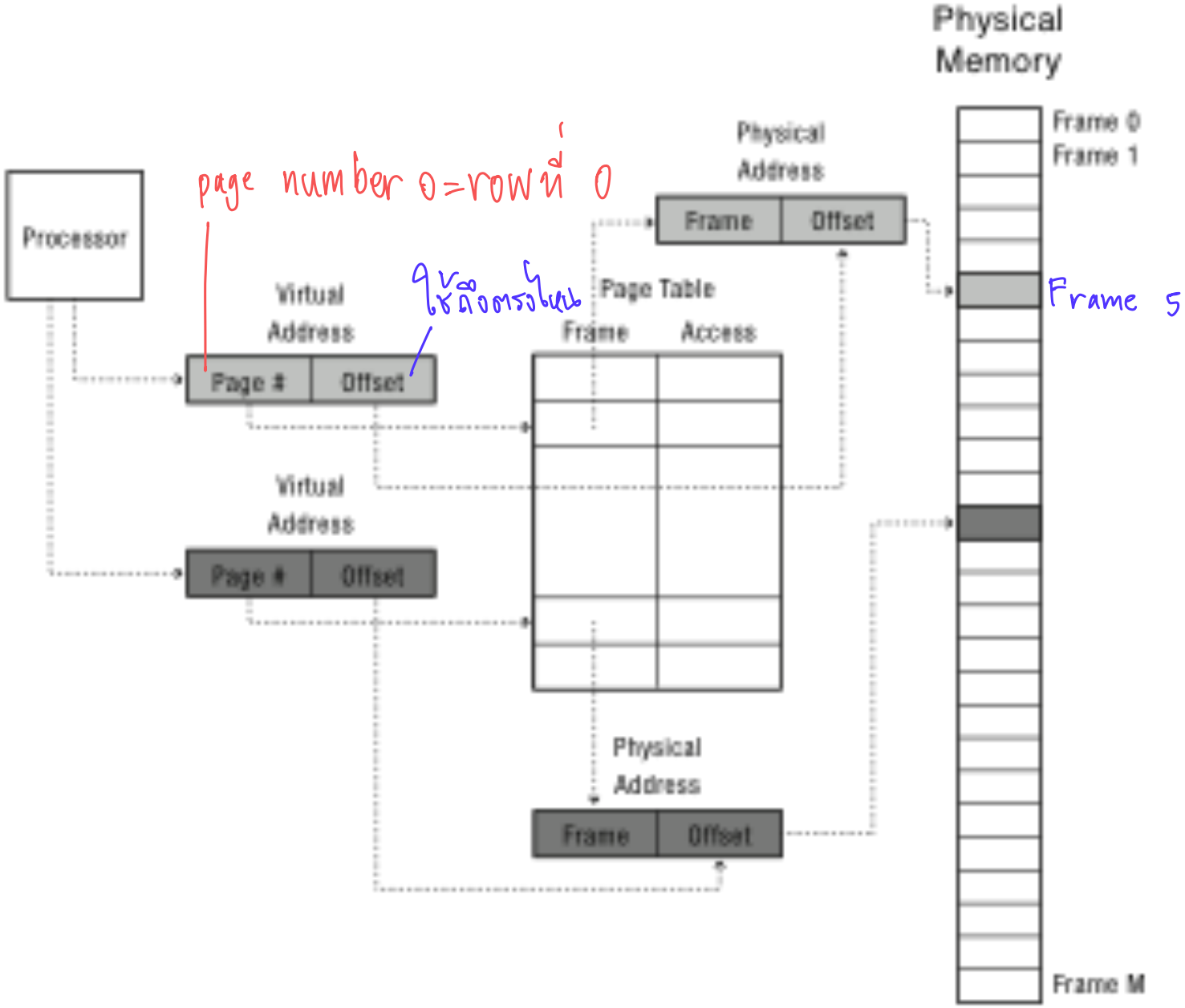
- Manage memory in fixed size units, or pages
- Finding a free page is easy
  - Bitmap allocation: 00111111000000001100
  - Each bit represents one physical page frame
- Each process has its own page table
  - Stored in physical memory
  - Hardware registers
    - pointer to page table start
    - page table length

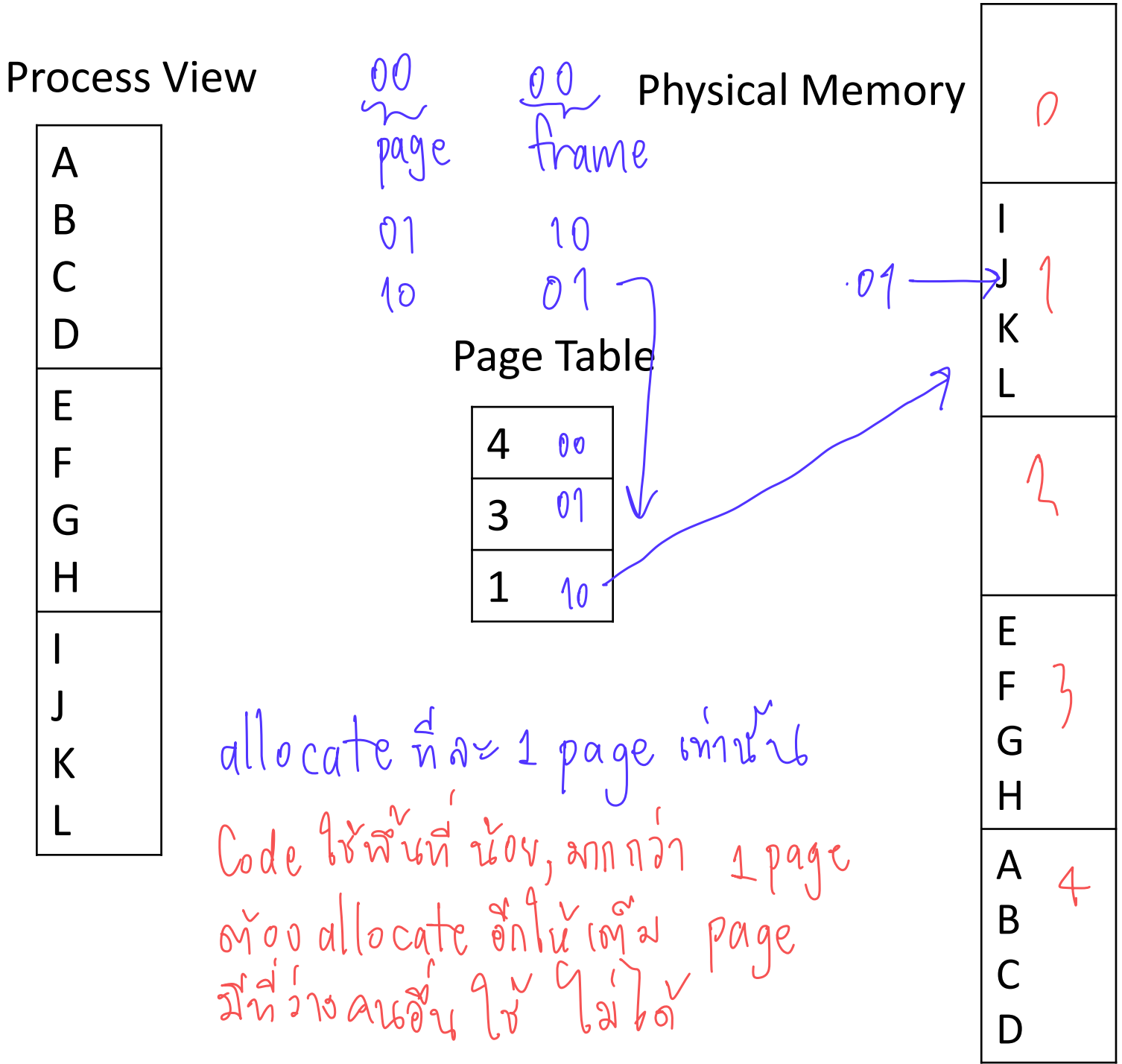
Paged Translation (Abstract)



Assignment: 2 page
1 frame

Paged Translation (Implementation)

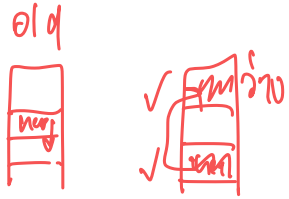



Process View
A
B
C
D
E
F
G
H
I
J
K
L
00
page
00
frame
01
10
10
01
Page Table
4 00
3 01
1 10
Physical Memory
0
I
J
K
L
1
2
E
F
G
H
3
A
B
C
D
4
01
allocate ทีละ 1 page เท่านั้น
Code ใช้พื้นที่ น้อย, มากกว่า 1 page
ต้อง allocate อีกให้เต็ม page
มีที่ว่างคนอื่น ใช้ ไม่ได้

Activity #3

ข้อด้อย ประสิทธิภาพในการใช้พื้นที่RAMใช้ได้ไม่เต็มที่

- Drawback of paging



ข้อดี

1 เร็ว ในการหาพื้นที่ว่าง

2 ข้อดีของ segment ที่ขยายตัว heap/stack ทำได้ดีขึ้น ไม่ต้อง relocate ทำเรื่อยๆ

Sparse Address Spaces

- Might want many separate dynamic segments
  - Per-processor heaps
  - Per-thread stacks
  - 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 page table entries

ถ้าต้องการใช้ memory มากขึ้นเรื่อยๆ page size ใหญ่ page table ใหญ่ขึ้น ซึ่งเก็บใน RAM (หลาย frame) ขนาดใหญ่ เวลา look-up table ทำได้ไม่ต่อเนื่อง

500,000 rows

4,000,000,000 page / bits สแกนที่เยอะ

Multi-level Translation

break down ตรงๆ ลงมาให้เล็ก

- 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

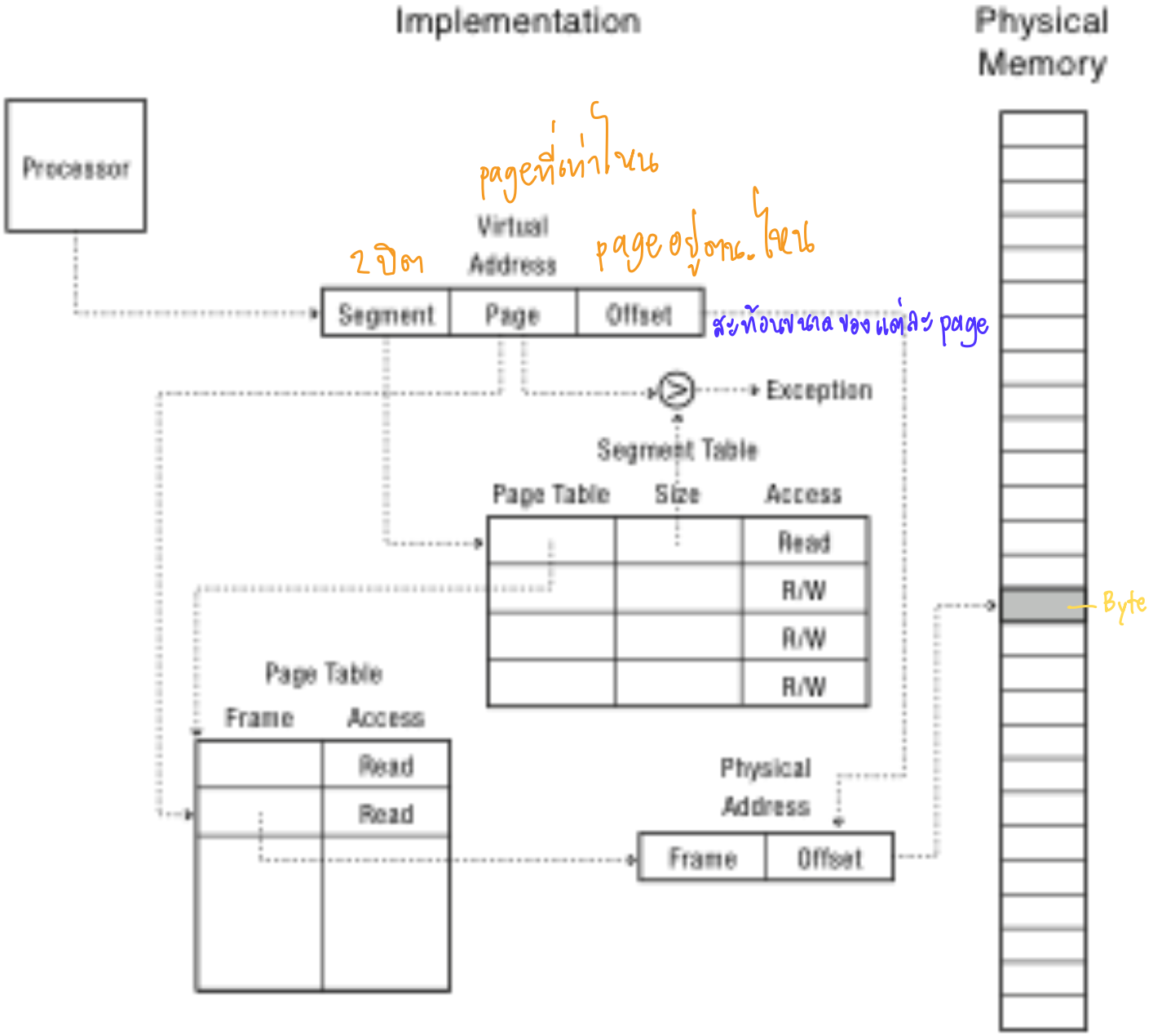
Paged Segmentation

อาจมาง 2 Level

Level 0 = segment table
เก็บค่า pointer ชี้ตรงที่เก็บ page สำหรับ segment

- 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

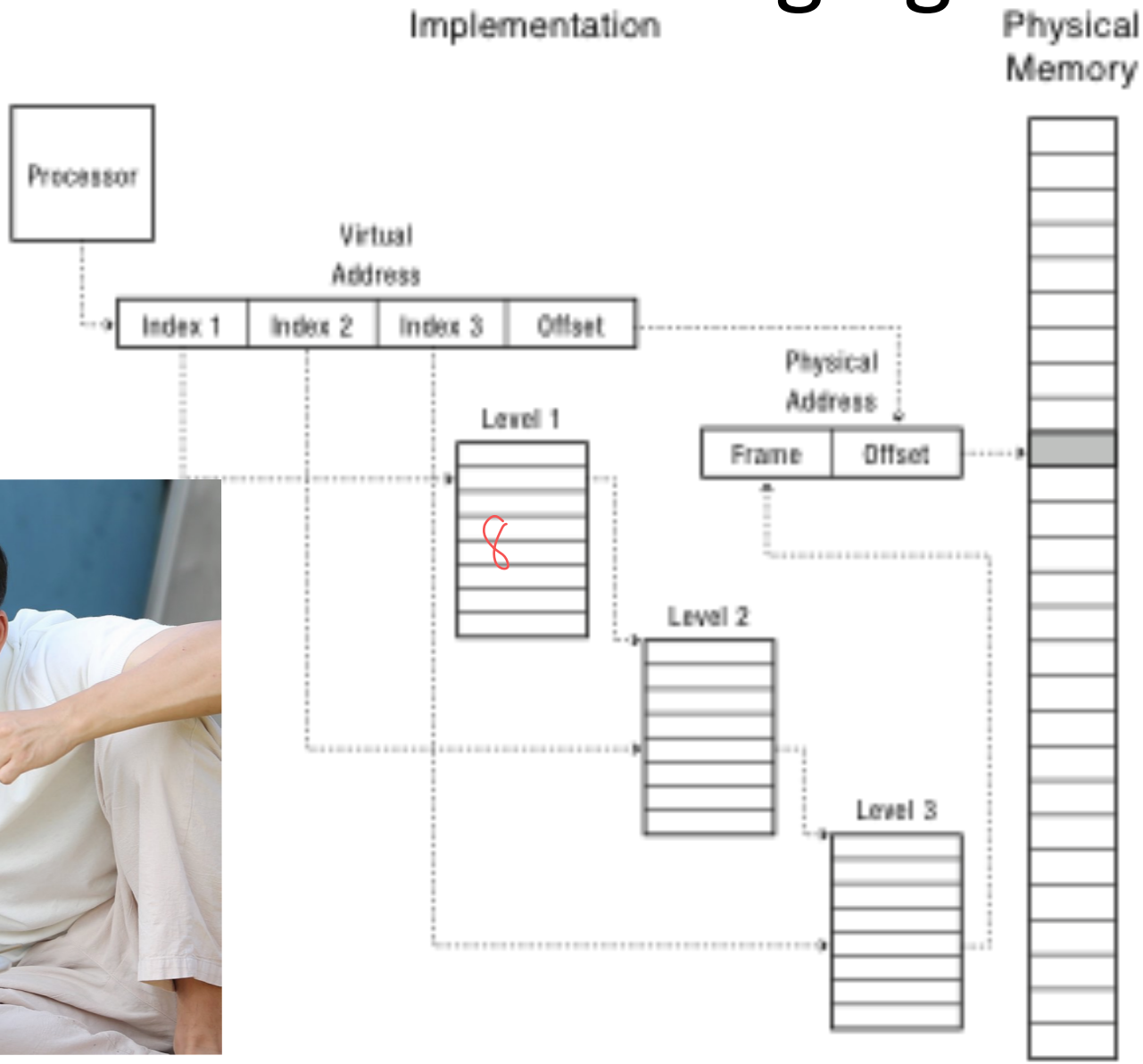
Paged Segmentation (Implementation)



มีหลาย page แก้ปัญหา ข้อมูลใหญ่กว่า 1 frame

Multilevel Paging

มีหลายตาราง



Activity #4

- Drawback of Multilevel translation

1 ประสิทธิภาพในการใช้พื้นที่ RAM ไม่ดี
2 ขนาดของข้อมูลใช้ใน ms translation ซับซ้อนมากขึ้นกว่าเดิม
3 มีความซับซ้อนในการอ่าน ทำให้ช้าและช้ากว่าเดิม

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

Activity #5

- Accelerate multilevel translation
  - Possible?
  - How?

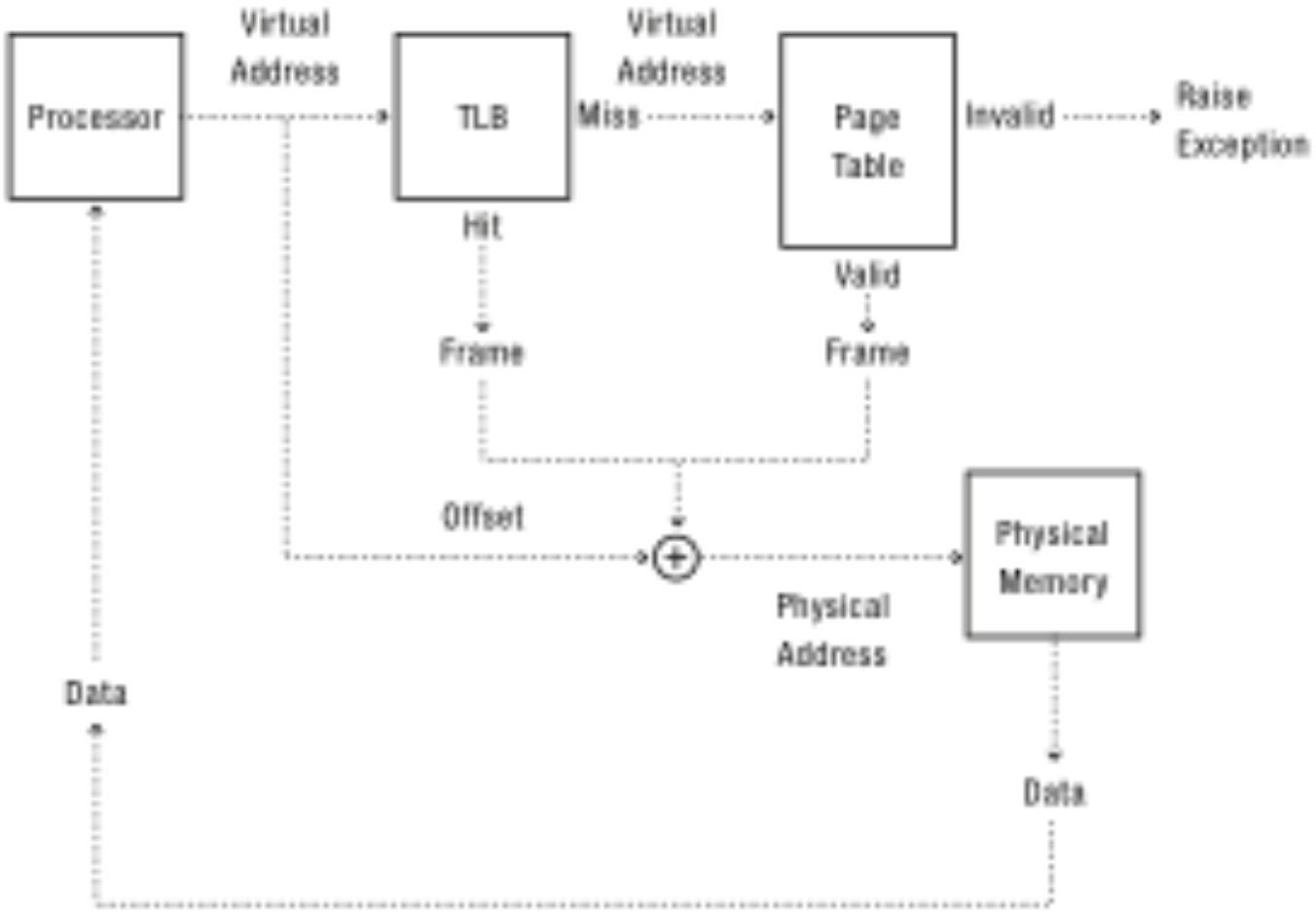
Efficient Address Translation

- 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 =

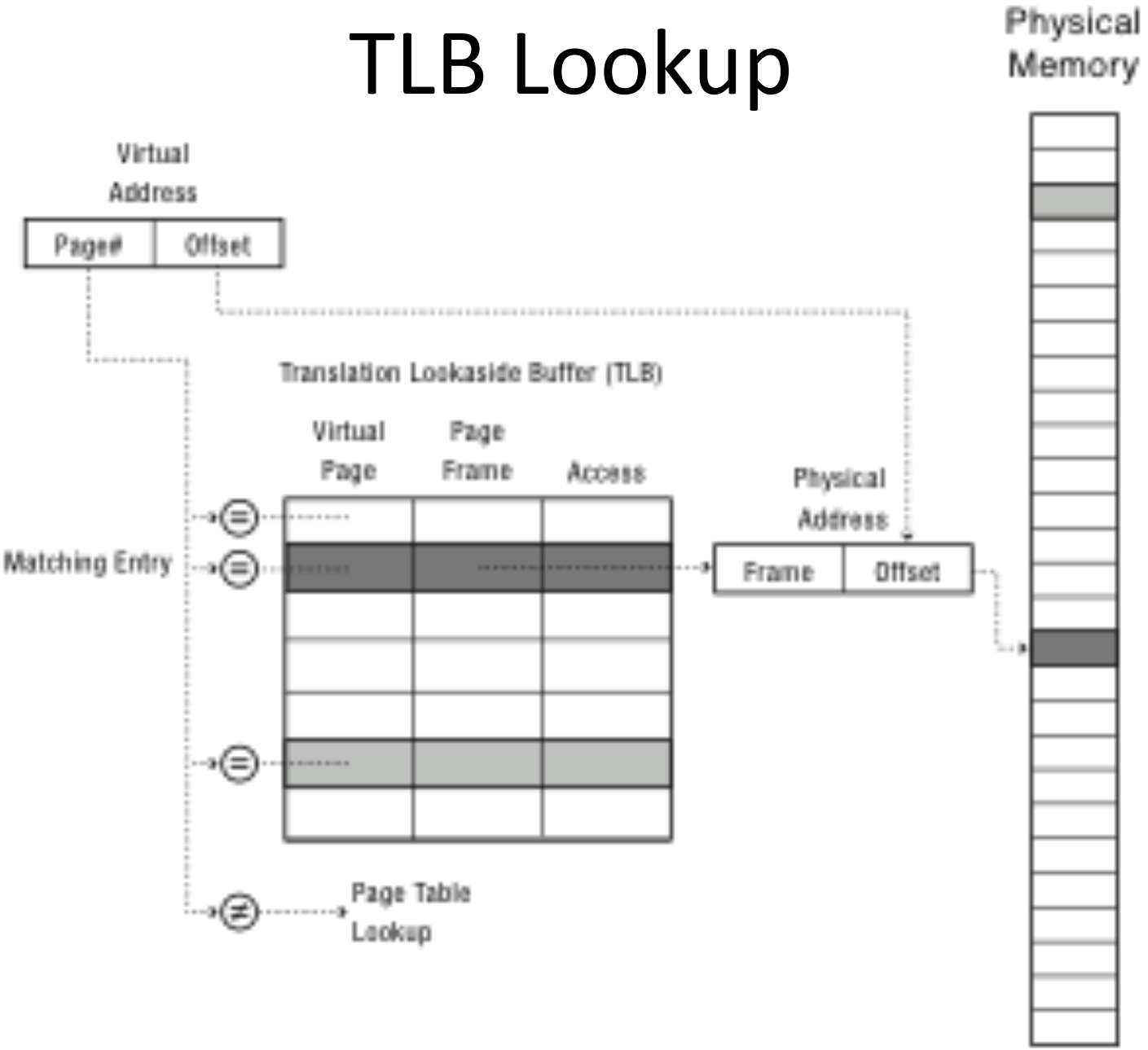
  Cost of TLB lookup +

  Prob(TLB miss) * cost of page table lookup

TLB and Page Table Translation



TLB Lookup


Physical Memory
Virtual Address
Page#
Offset
Translation Lookaside Buffer (TLB)
Virtual Page
Page Frame
Access
Physical Address
Matching Entry
Frame
Offset
Page Table Lookup

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