

# Operating System

---

Subject 1. Basic

Subject 2. Contiguous Alloc

Subject 3. Paging

Subject 4. Page Table

Subject 5. Virtual Memory

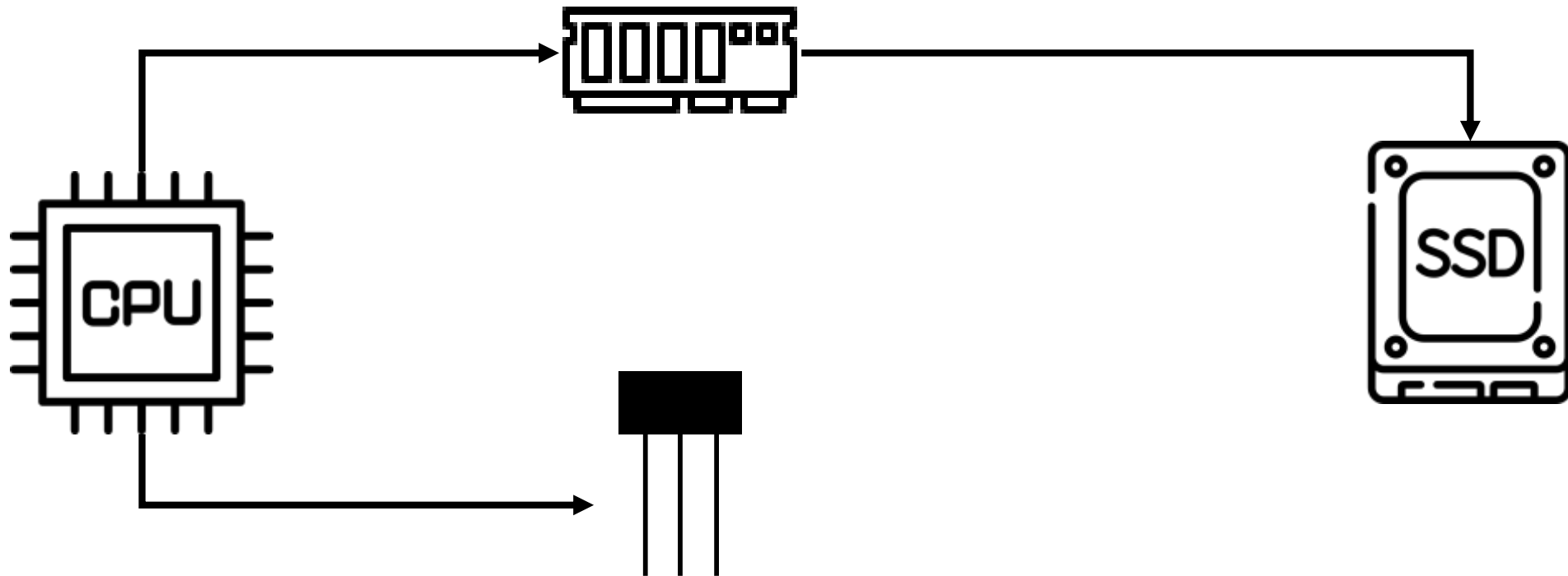
# subject 1.

## Basic

---

# Hardware

- CPU는 메인 메모리와 레지스터에만 직접 접근할 수 있다.
- 따라서 모든 실행되는 명령어와 데이터는 디스크에서 메모리나 레지스터에 있어야 한다.



# Hardware

---

1 byte = 8 bits

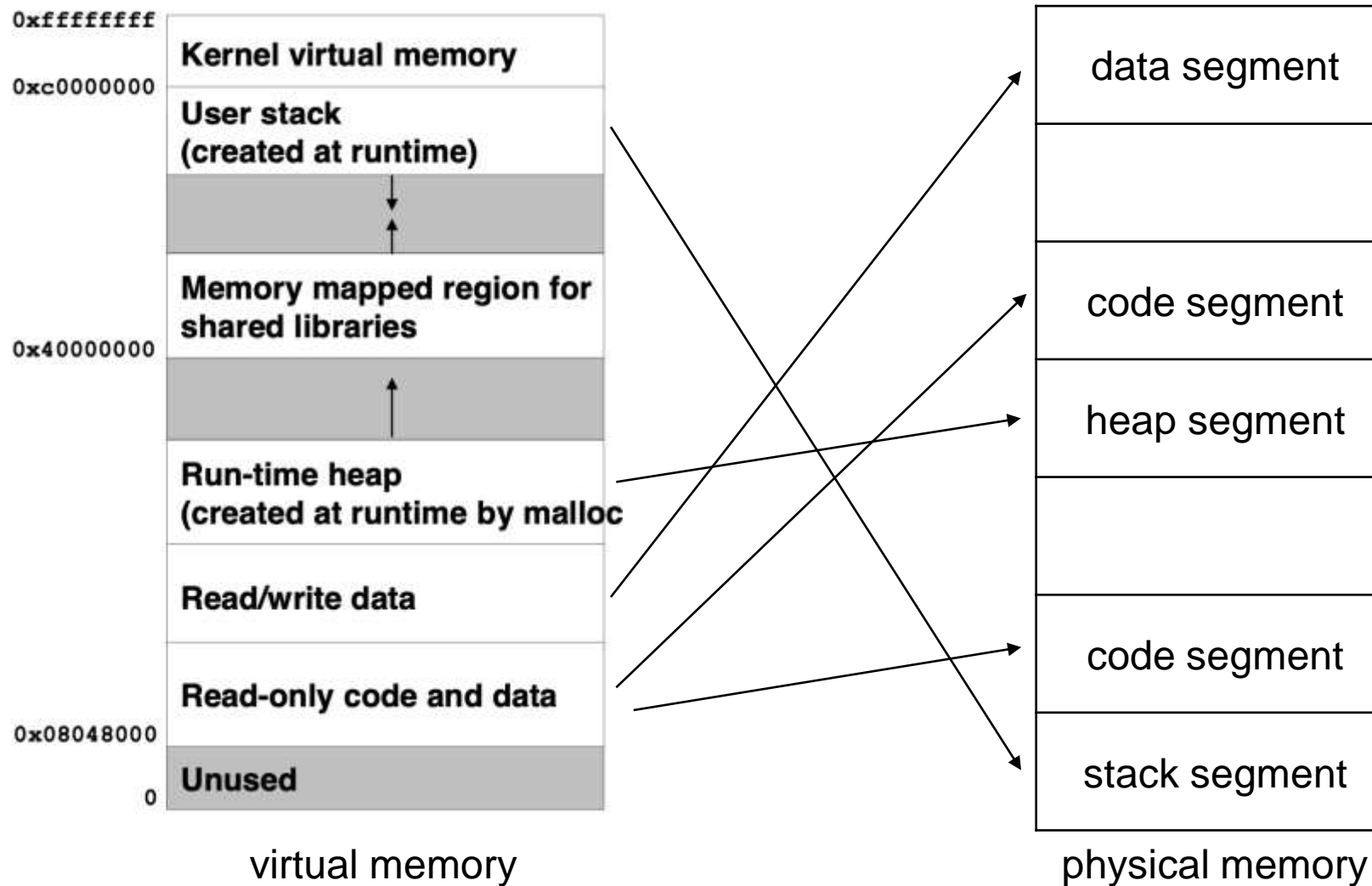
1 KB =  $2^{10}$  bytes

1 MB =  $2^{10}$  KB =  $2^{20}$  bytes

1 GB =  $2^{10}$  MB =  $2^{20}$  KB =  $2^{30}$  bytes

# Address Space

32bit system



# Address Space



made by Seungtaek

# Address Space

---

- Address Space: 프로세스에서 참조 할 수 있는 주소들의 범위
- **1 바이트마다 주소가 붙는다.** (byte address)
- 32bit 시스템
  - $2^{32}$ 개의 주소 지정 가능
  - 주소 1개당 1byte 이므로,  $2^{32} * 1 \text{ byte} = 4\text{GB}$ . 4GB의 메모리 크기를 addressing 할 수 있다.
- 64bit 시스템
  - $2^{64} = 16 \text{ EB}$  (엑사 바이트)

# Address Space

## Physical Memory Limits: Windows 11


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

 Expand table

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 Server 2016

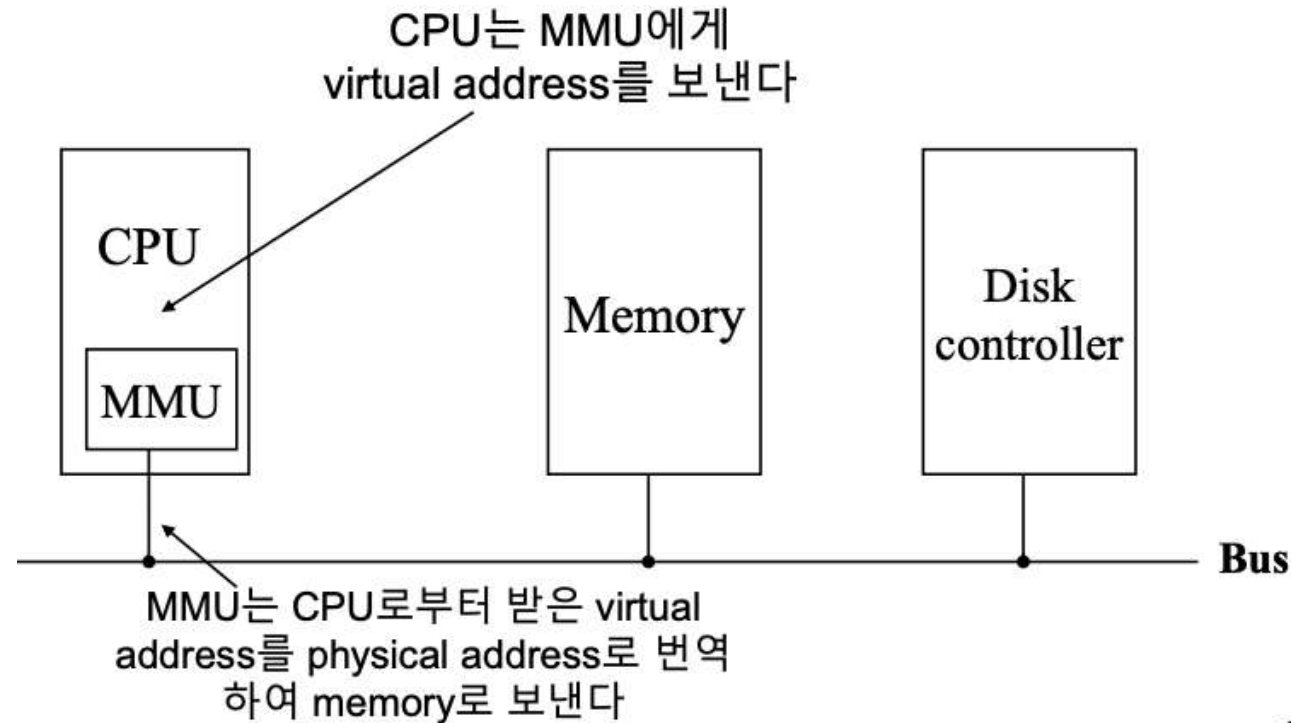
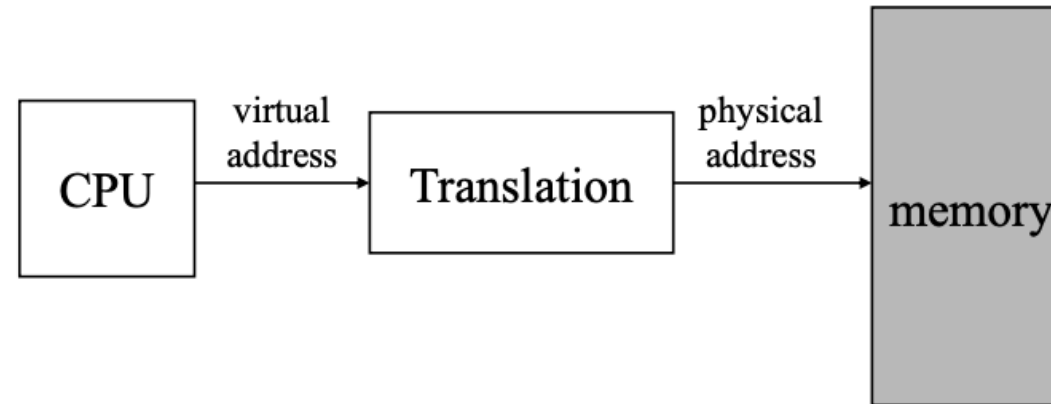
The following table specifies the limits on physical memory for Windows Server 2016.

 Expand table

Version	Limit on X64
Windows Server 2016 Datacenter	24 TB
Windows Server 2016 Standard	24 TB



# Address translation

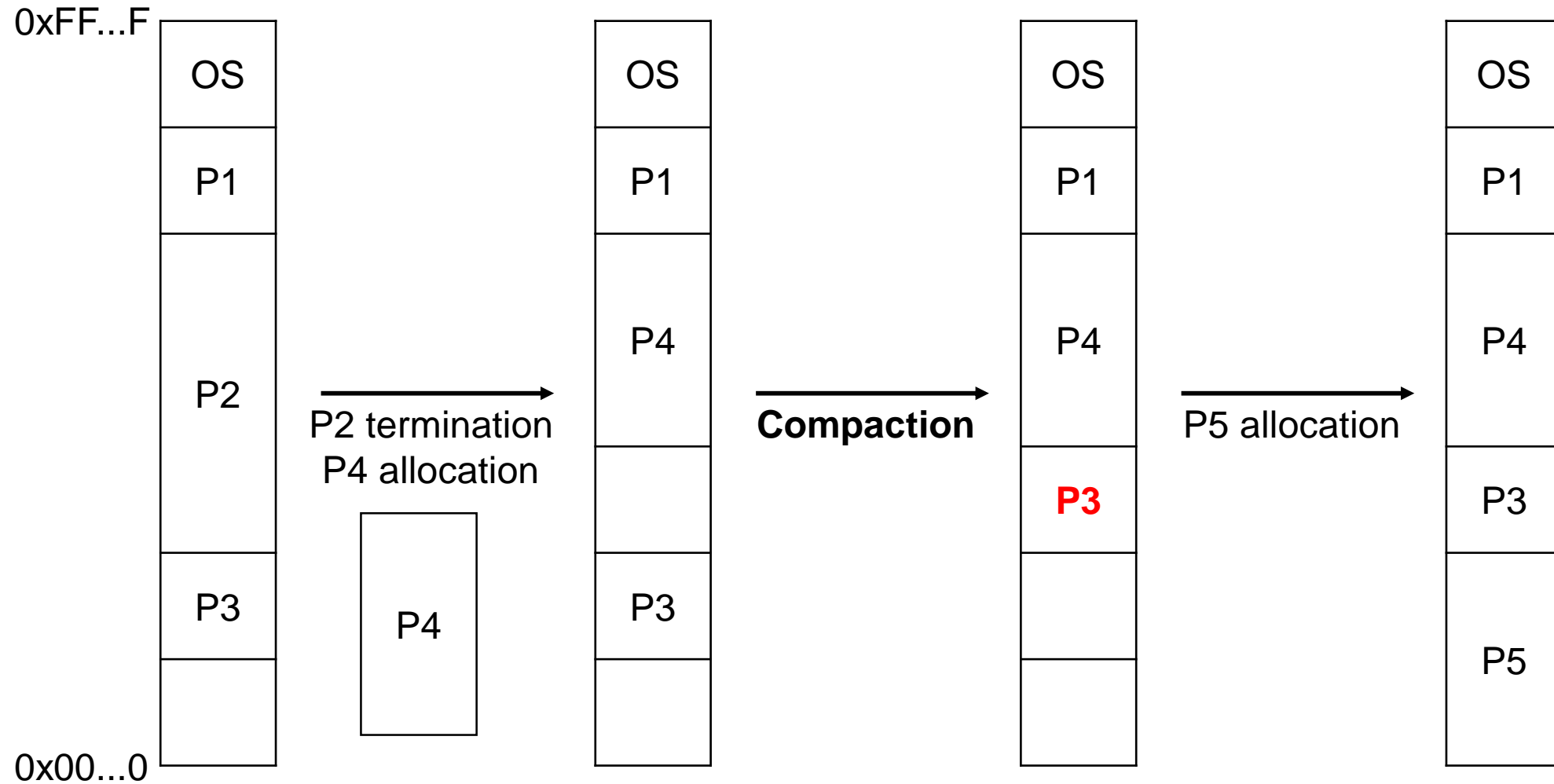


# subject 2.

## Contiguous Memory Allocation

---

# Variable Partition



# Dynamic storage allocation

---

First fit

첫 번째 사용 가능한 가용 공간에 할당

Best fit

사용 가능한 공간 중에서 가장 작은 곳에 할당

Worst fit

가장 큰 가용 공간에 할당

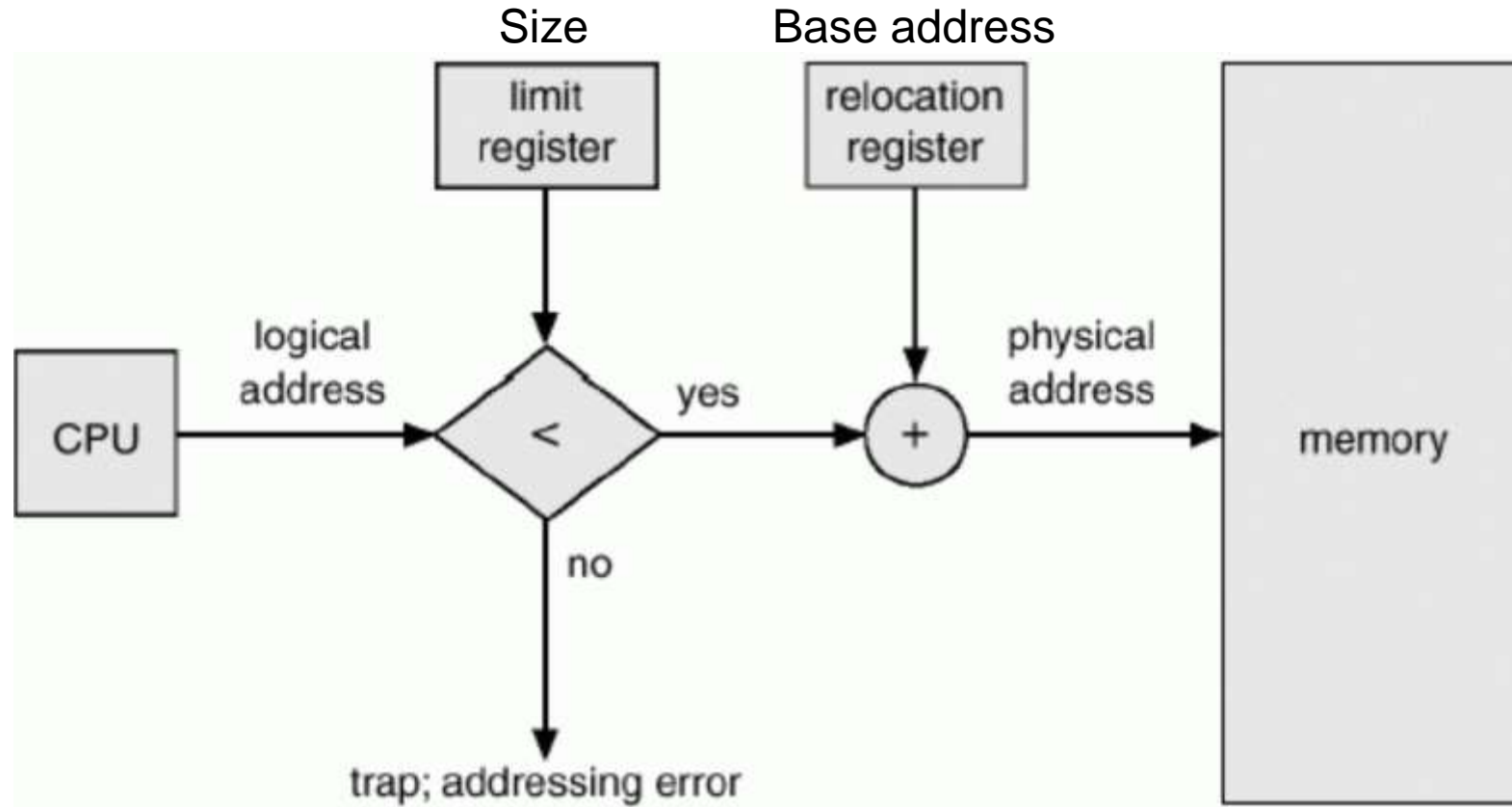
Simulation..

Time : First fit > Best fit > Worst fit

Utilization : First fit, Best fit > Worst fit

# Memory Protection

PCB에 해당 값 저장 필요



# External fragmentation

## Statistical analysis of First-fit

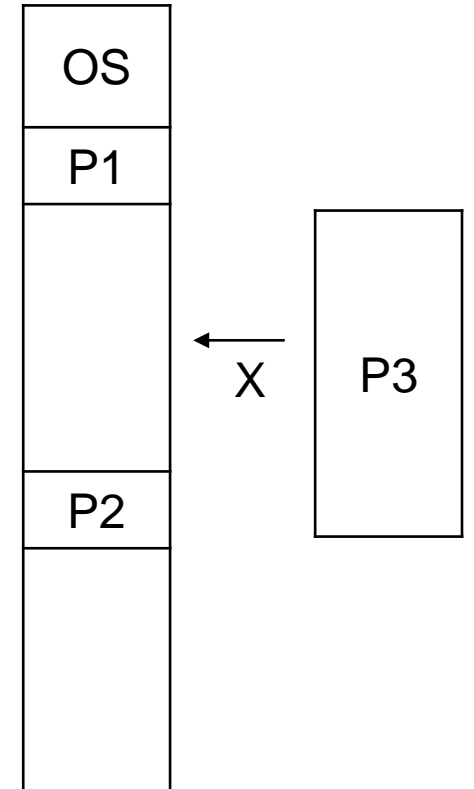
### 50-percent rule

N개의 블록이 할당되었을 때,  $0.5N$ 개의 블록이 단편화 때문에 손실  
즉, 메모리의  $1/3$  을 쓸 수 없게 됨

ex) 전체가 30개 블록일 때 : 20개 할당되면 10개는 손실.

### Compaction

비용! 비용! 비용!



# subject 3.

## Paging

---

# Basic Method

---

주소 공간을 동일한 크기인 page로 나누어 관리

보통 1 page의 크기는 4KB

- **프레임** (Frame) : 물리 메모리를 고정된 크기로 나누었을 때, 하나의 단위
  - **페이지** (Page) : 논리 메모리를 고정된 크기로 나누었을 때, 하나의 단위
- 특별한 이유가 없으면 page와 frame 크기는 같다.

페이지가 하나의 프레임을 할당 받으면, 물리 메모리에 위치하게 된다.

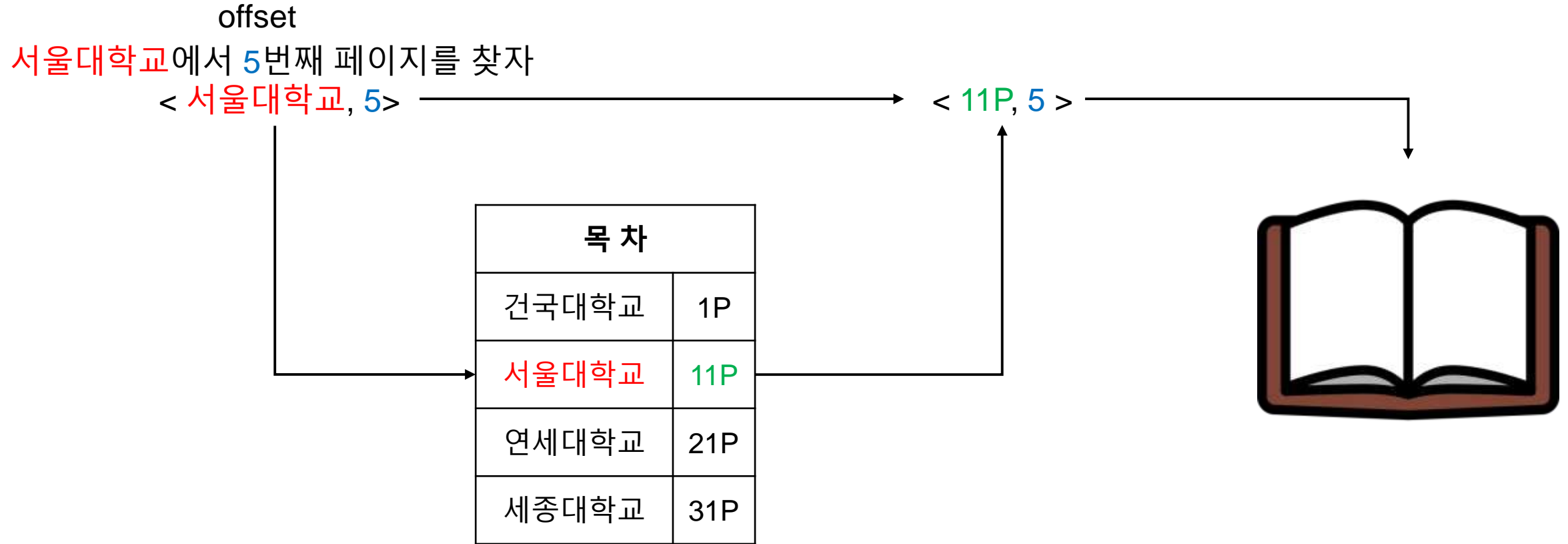
프레임을 할당 받지 못한 페이지들은 외부 저장장치(Backing store, secondary storage)에 저장된다.

4KB 감잡기

$2^{12}$  Bytes : 하나의 Page는 4096개의 address를 포함



# Allegory



# Page Table

프로세스마다 하나의 페이지 테이블을 가짐

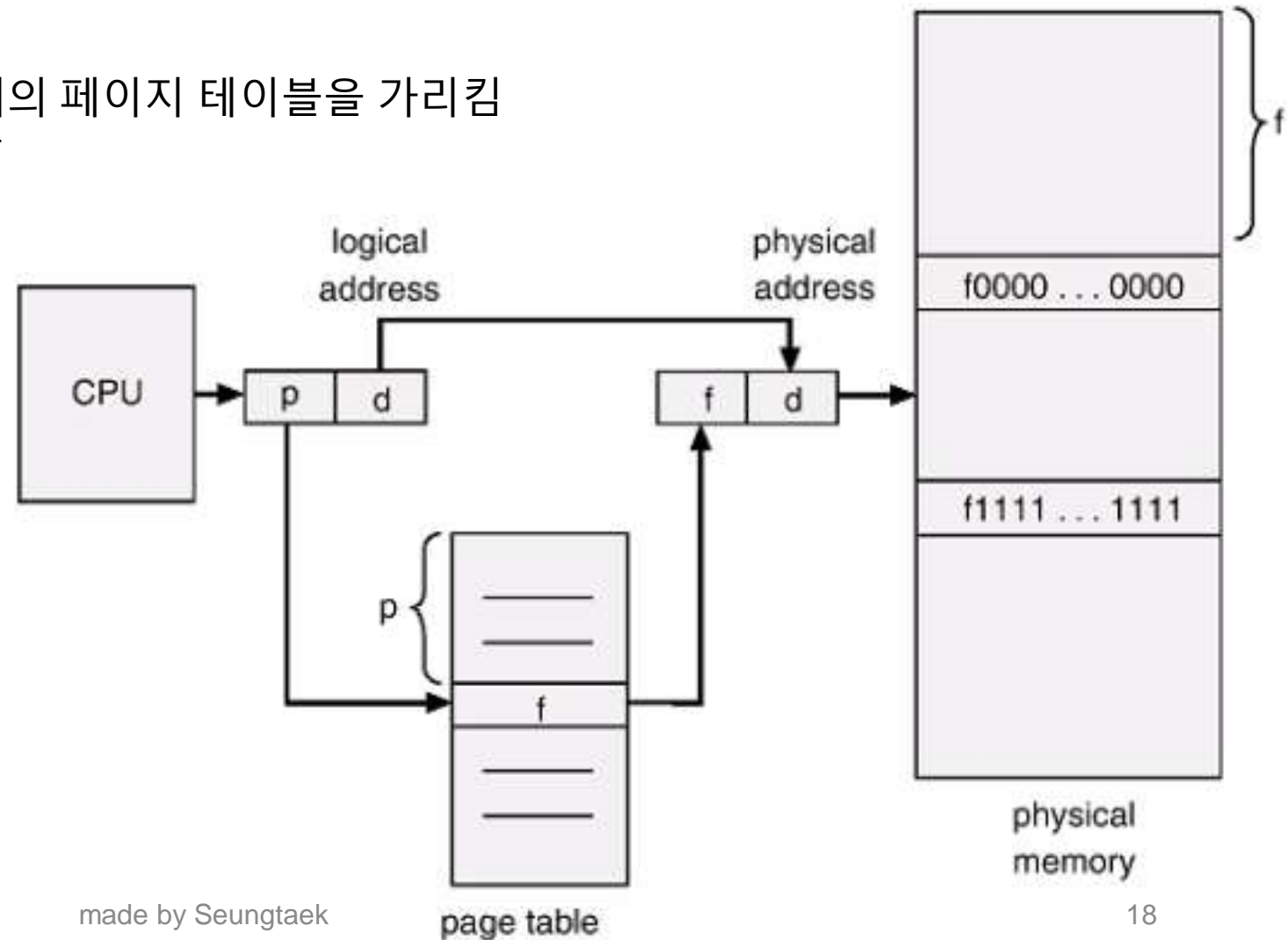
page table은 Linear한 형태로 생각

페이지 테이블은 물리 메모리(RAM)에 위치함

PTBR(Page-table base register)가 물리 메모리 내의 페이지 테이블을 가리킴

→ 문맥 교환 시 PCB에 저장 필요함

CF) 스레드 스위칭은 PTBR을 바꾸지 않는다.



< A, B > : A concatenation B

Logical address = < p, d >

Physical address = < pageTable[p], d >

# Quiz

32 bit 시스템, 페이지 크기 4KB, 프레임 크기 4KB

Quiz 1. 논리주소 중 몇 비트를 오프셋으로 사용?

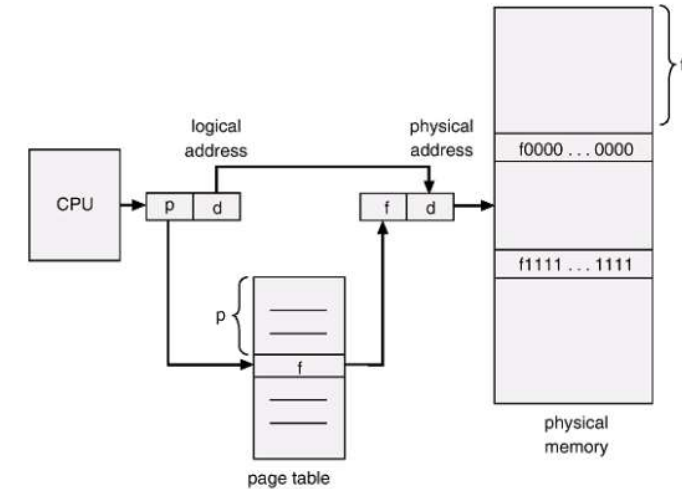
답: 12비트

Quiz 2. 128MB 물리메모리는 몇 개의 Frame으로 쪼개지는가?

답:  $128\text{MB} / 4\text{KB} = 32\text{K}$  개

Quiz 3. 논리적으로 4GB를 사용하기 위해서는 몇 개의 Page가 필요한가?

답:  $4\text{GB} / 4\text{KB} = \text{M}$ 개



# Page Table Entry (PTE)

Page base address : 프레임의 시작 주소

Flag bits

- Accessed bit: 페이지에 대한 접근이 있었는지
- Dirty bit: 페이지 내용의 변경이 있었는지
- Present bit: 현재 페이지에 할당된 프레임이 있는지      Virtual Memory..
- Read/Write bit: 읽기/쓰기에 대한 권한 표시
- User/Supervisor bit : 사용자 모드에서 접근이 가능한지

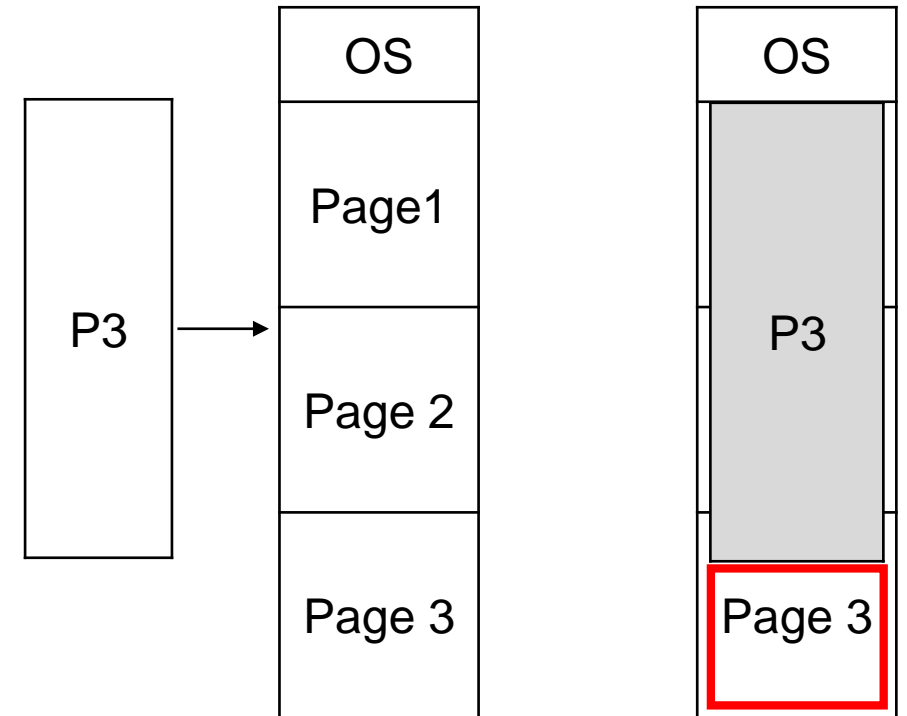
일반적으로 PTE는 **4 Byte** (32bit system)

Page base address	Accessed bit	Dirty bit	Present bit	Read bit	Write bit	Mode bit	...
-------------------	--------------	-----------	-------------	----------	-----------	----------	-----

# Internal fragmentation

Page 크기를  $S$ 라 할 때,  
평균적으로 프로세스당  $S/2$  크기의 내부 단편화 발생.

$S$ 가 작은게 좋을까?  
 $S$  작다 = Page 개수가 많아진다 = Page Table 크기가 커진다.



# Internal fragmentation

## Page Table이 커지면 얼마나 커지겠어?

페이지 크기가 4KB일 때, PTE 개수는  $2^{20}$ 개.

Page Table의 크기는  $4 \text{ byte} * 2^{20} = 4\text{MB}$

페이지 크기가 1Byte 일 때, PTE 개수는  $2^{32}$  개.

Page Table의 크기는  $4 \text{ byte} * 2^{32} = 16\text{GB}$

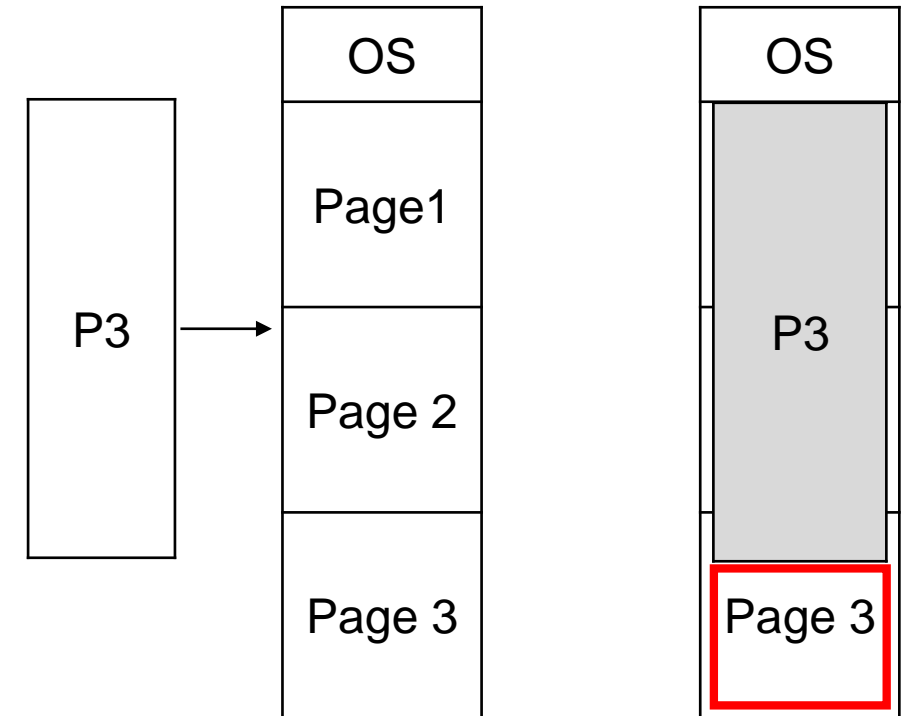
프로세스가 1000개라면?

사실은요..

프로세스가 자신의 모든 주소 범위를 사용하는 경우는 드물다

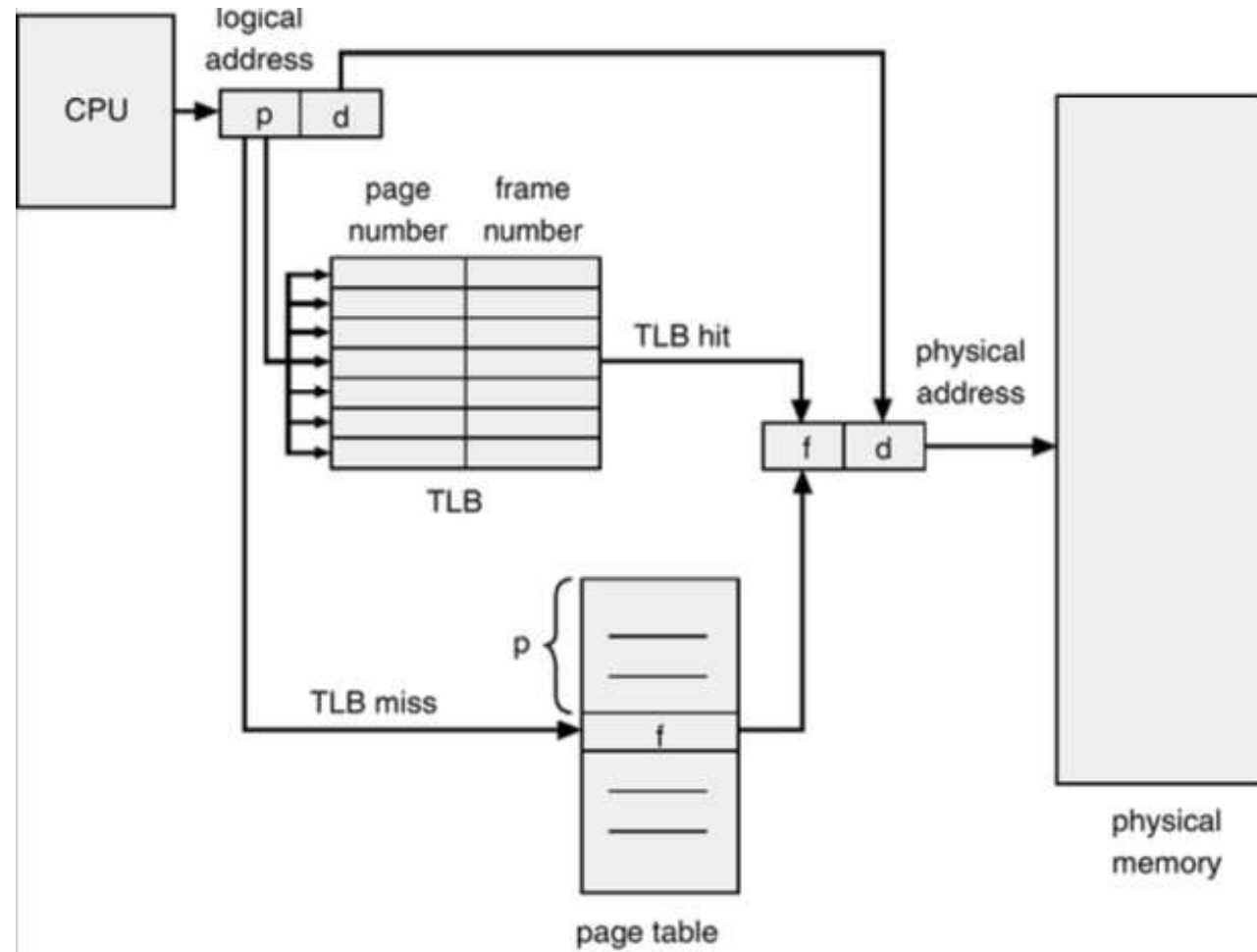
모든 PTE를 사용할 필요는 없음

PTLR(page table length register) : 페이지 테이블 길이



# Translation Look-aside Buffers (TLB)

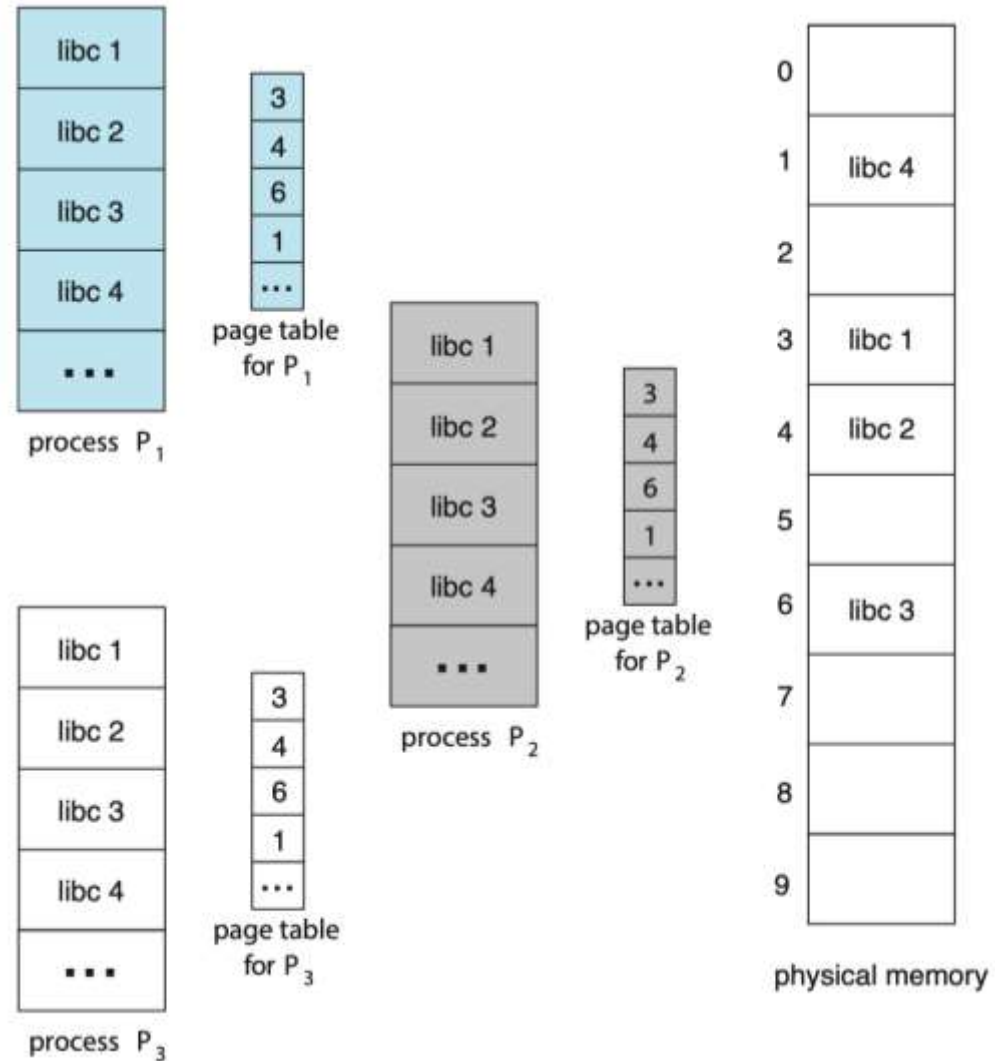
페이징 방법에서는 데이터 액세스를 위해 항상 두 번의 메모리 접근을 거쳐야 함



Cache는 다음에

# Shared Pages

Shared library, compiler, OS, IPC(shared memory), ...



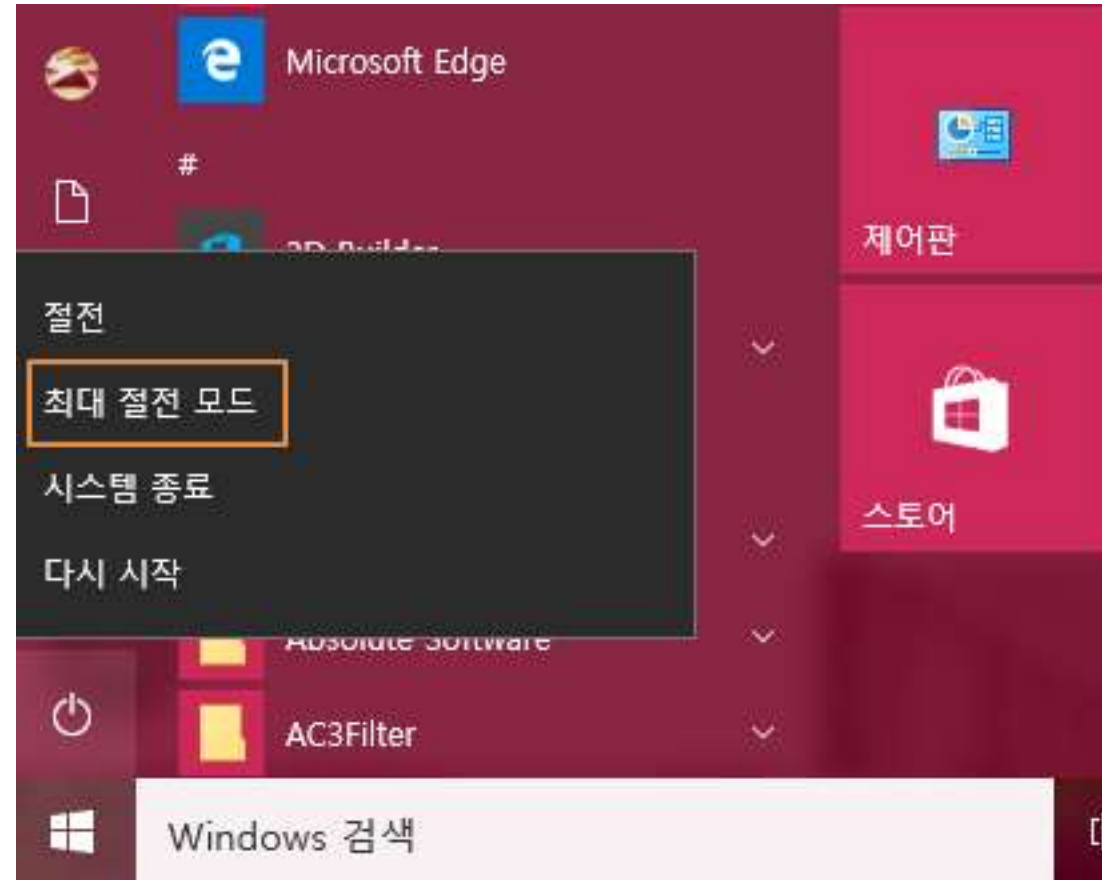


CS

**Common Sense**

---

# 최대 절전 모드



# Page size in Linux

---

## 4KB

```
root@ithinkso:~# getconf PAGESIZE  
4096
```

## 16KB

```
hongseungtaeg ~  
getconf PAGESIZE  
16384
```

# Memory bandwidth

## M4 Max 칩

최대 16코어 CPU(성능 코어 12개 및 효율 코어 4개)

최대 40코어 GPU

하드웨어 가속형 레이 트레이싱

16코어 Neural Engine

최대 546GB/s 메모리 대역폭

SoC(System on a Chip)

## KLEVV DDR5-6000

KD5AGUA80-60A300J

CAPACITY

16GBx2

SPEED

6000MT/s

$6000 \text{ MHz} \times 64\text{bits} \times 2 / 8 = 96\text{GB/s}$

Memory Bandwidth (GB/s)=Memory Clock Speed (MHz)×Memory Bus Width (bits)×2/8

# subject 4.

## Structure of the Page Table

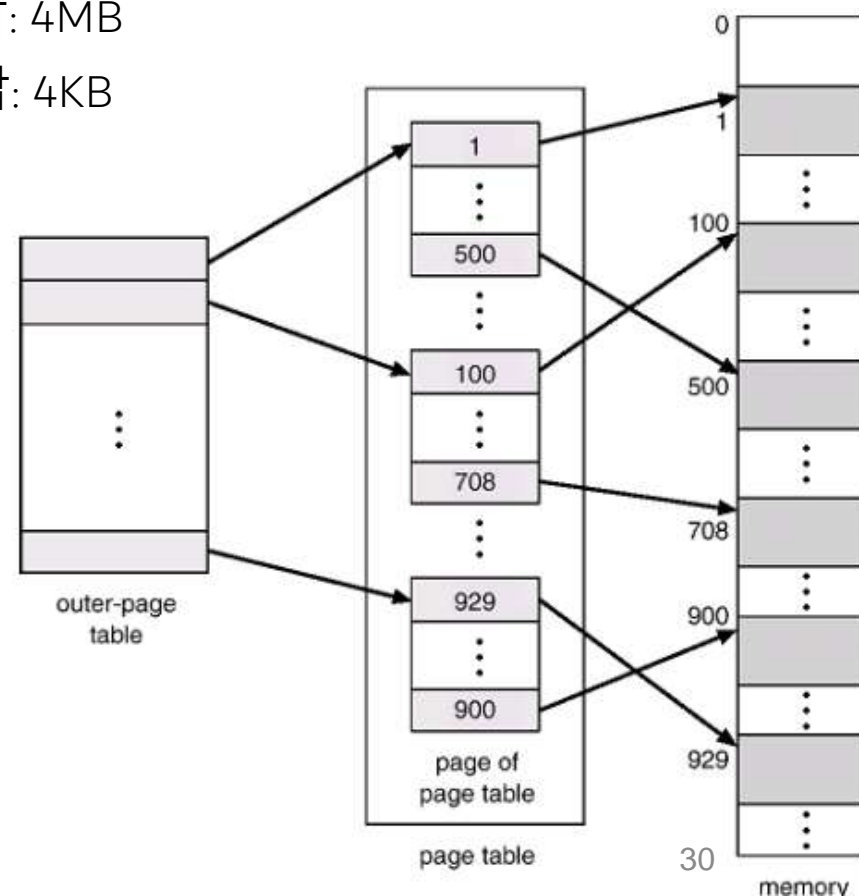
---

# 2-level Page Table

**장점: 필요한 연속된 공간이 적어진다.**

- 원소의 개수가  $2^{10}$ 개이고, 각 원소 크기가 4 byte라면 4KB 만큼만 필요
- 더군다나 모든 page table이 있어야 하는건 아님 (사용하는 주소 공간은 sparse 하다)
- Quiz 1. Outer-page table의 원소 1개가 매핑하는 주소 공간의 크기는? 답: 4MB
- Quiz 2. Inner-page table의 원소 1개는 매핑하는 주소 공간의 크기는? 답: 4KB

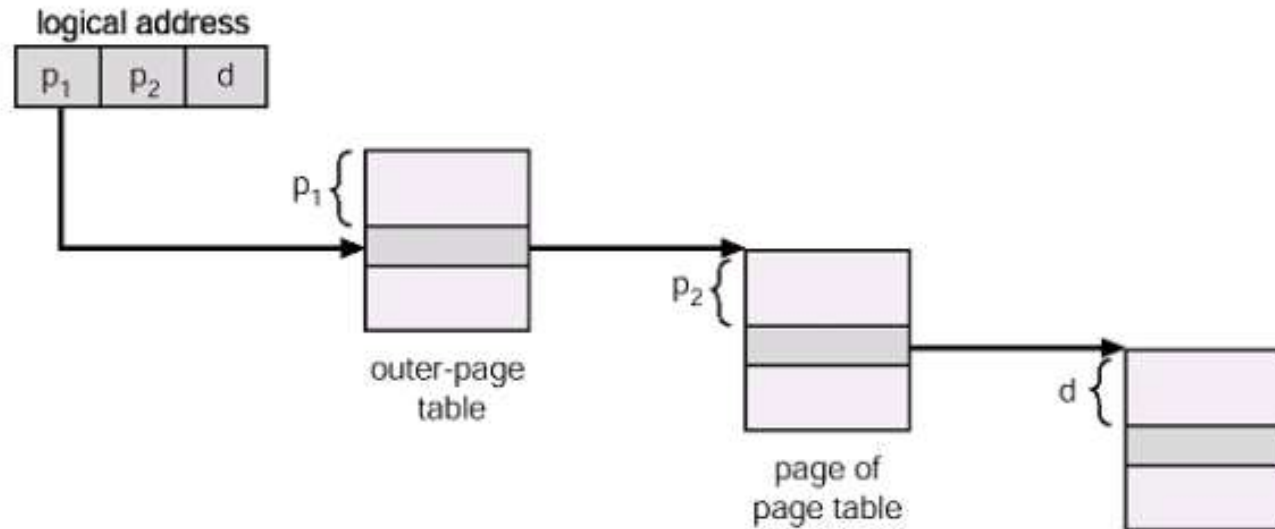
page number		page offset
$p_i$	$p_2$	$d$
10	10	12



# 2-level Page Table Walk

1. MMU는 32bit logical 주소를 받는다.
2. outer-page table 접근 (메모리 접근 1번)
3. page of page table 접근 (메모리 접근 2번)
4. 실제 메모리 접근 (메모리 접근 3번)

메모리 1회 접근을 위해 실제로는 3번을 접근해야 한다. → 그래서 한번에 갈 수 있는 TLB가 중요하다.



# Inverted Page Table

왜 page table의 용량이 큰가?

- logical address와 physical address가 1:1로 매핑되어야 하기 때문이다.
- 아무리 가상 메모리 공간이 크더라도, 물리 메모리의 크기에는 한계가 있음
- 64bit 주소 공간에 모든 물리 메모리가 매핑되어 있지 않음
- 반대로 모든 물리 메모리는 가상 메모리의 페이지에 매핑될 확률이 높음

Page #	Frame #

Frame #	Page ID	
	PID	P
1		
2		
3		
4		
5		

<기존 방식 – Multilevel Page Table>

<새로운 방식- Inverted Page Table>

실제 많은 연구와 실험이 수행됐는데, 생각보다 성능이 안좋아서 사용하지 않는다.



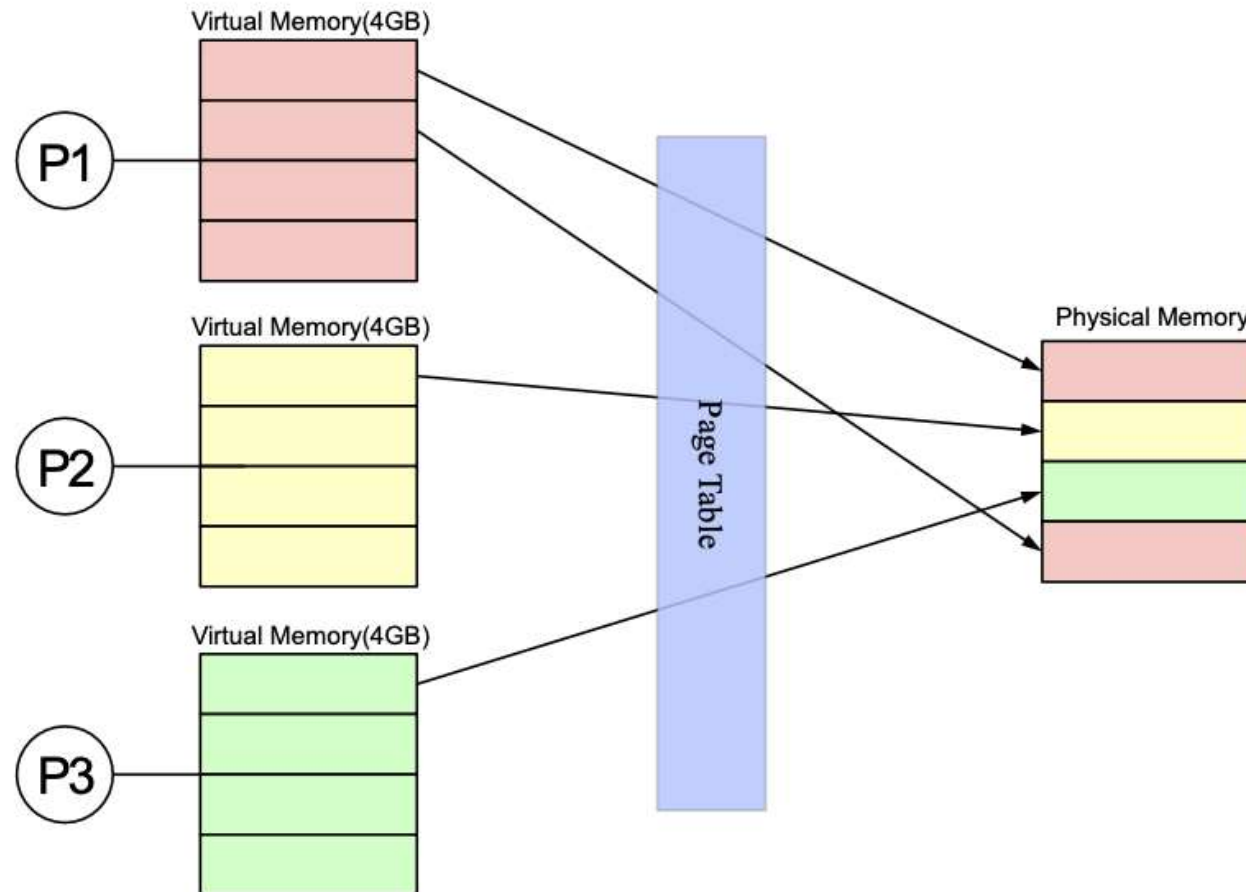
# subject 5.

## Virtual Memory

---

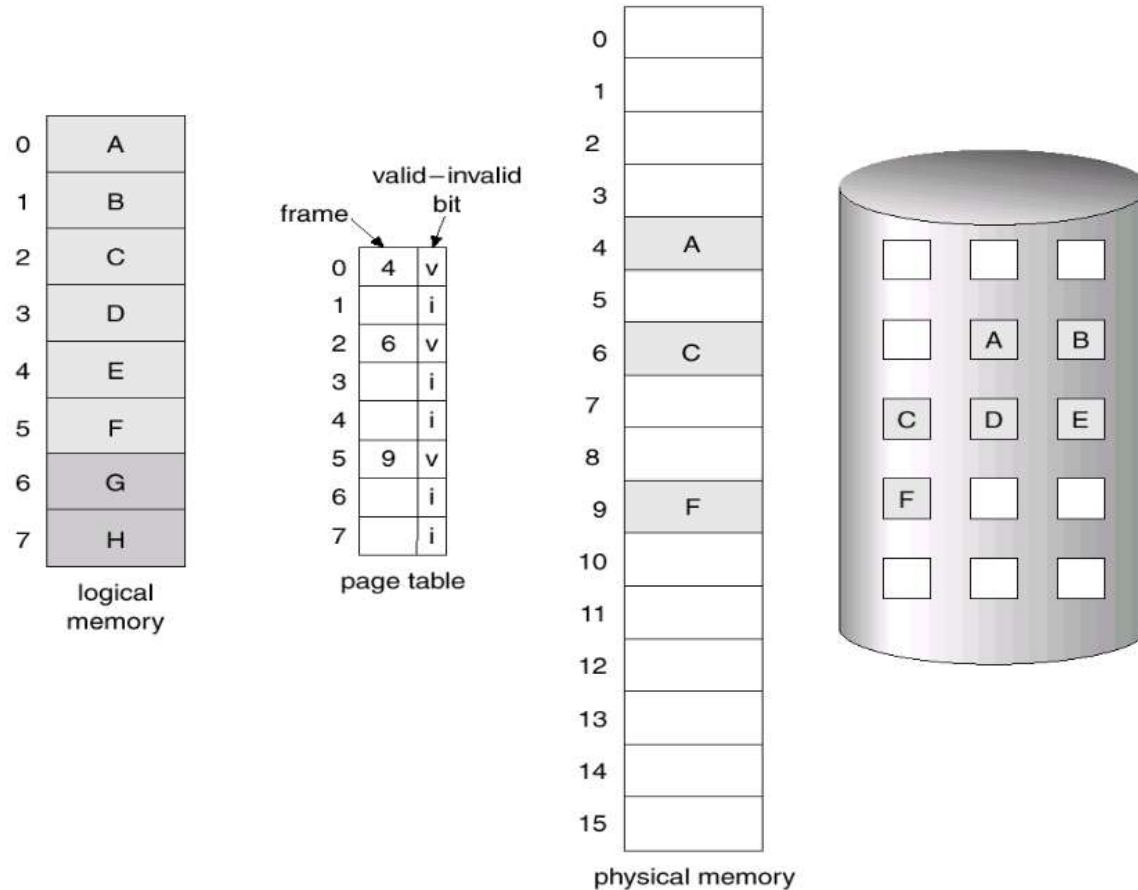
# Virtual memory

프로세스가 수행 되기 위해서 프로그램의 모든 부분이 물리 메모리(physical memory)에 있을 필요는 없다.



# Demand Paging

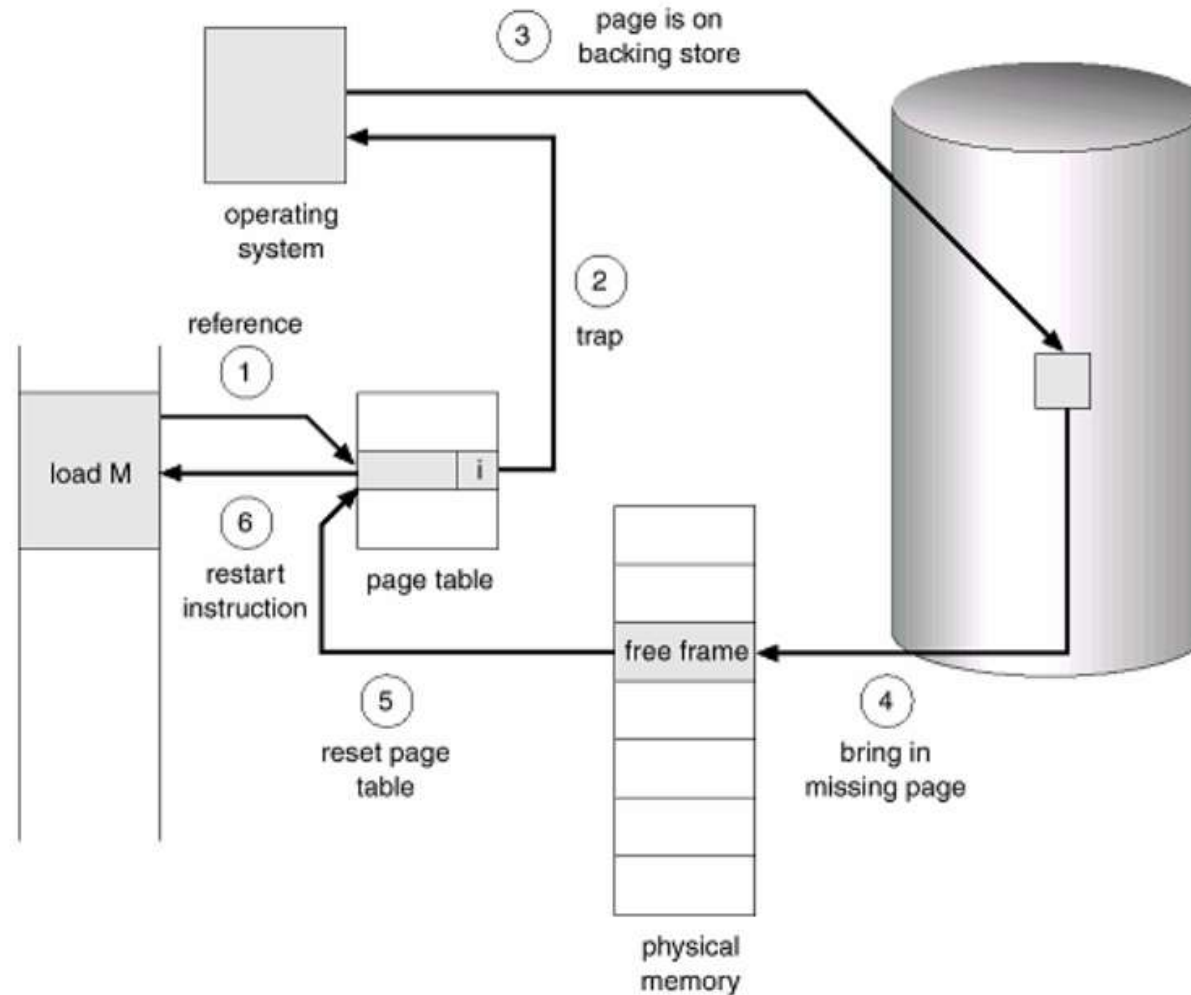
- 프로세스의 실행을 위한 모든 page를 메모리에 올리지 않고, 필요한 page의 요청이 발생할 때 메모리에 올리는 paging 기법
- 한 프로세스에 필요한 page를 memory와 secondary storage 간에 이동시킴
  - Valid, invalid page 존재



# Page Fault

프로세스가 페이지를 참조하였을 때 해당 페이지가 할당 받은 프레임이 없는 경우 (valid bit == false)

- Page Fault handler 수행



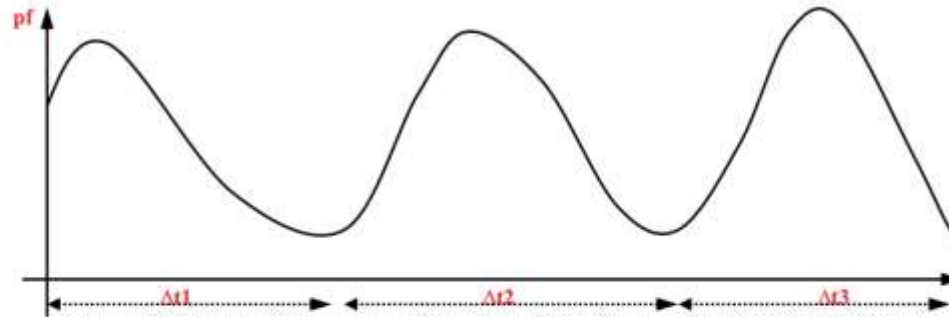
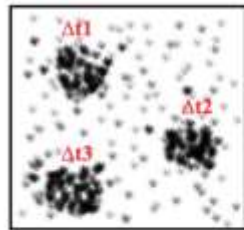
Swap in / out  
Page in / out

# Working set

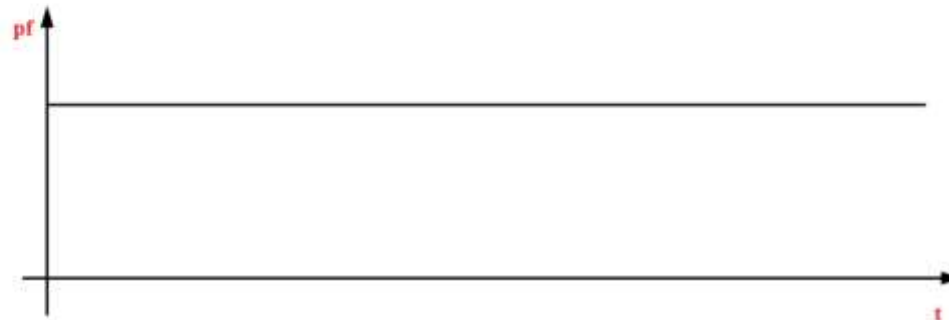
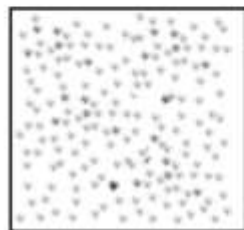
- 어떤 시간 window 동안에 접근한 page들의 집합
- 시간마다 working set이 변함
  - ... 25737456745680120123456 :  $Ws = \{4,5,6,7,8,0\}$
- “적당한” window를 잡으면 working set은 locality를 나타냄

## Locality에 따른 Page fault

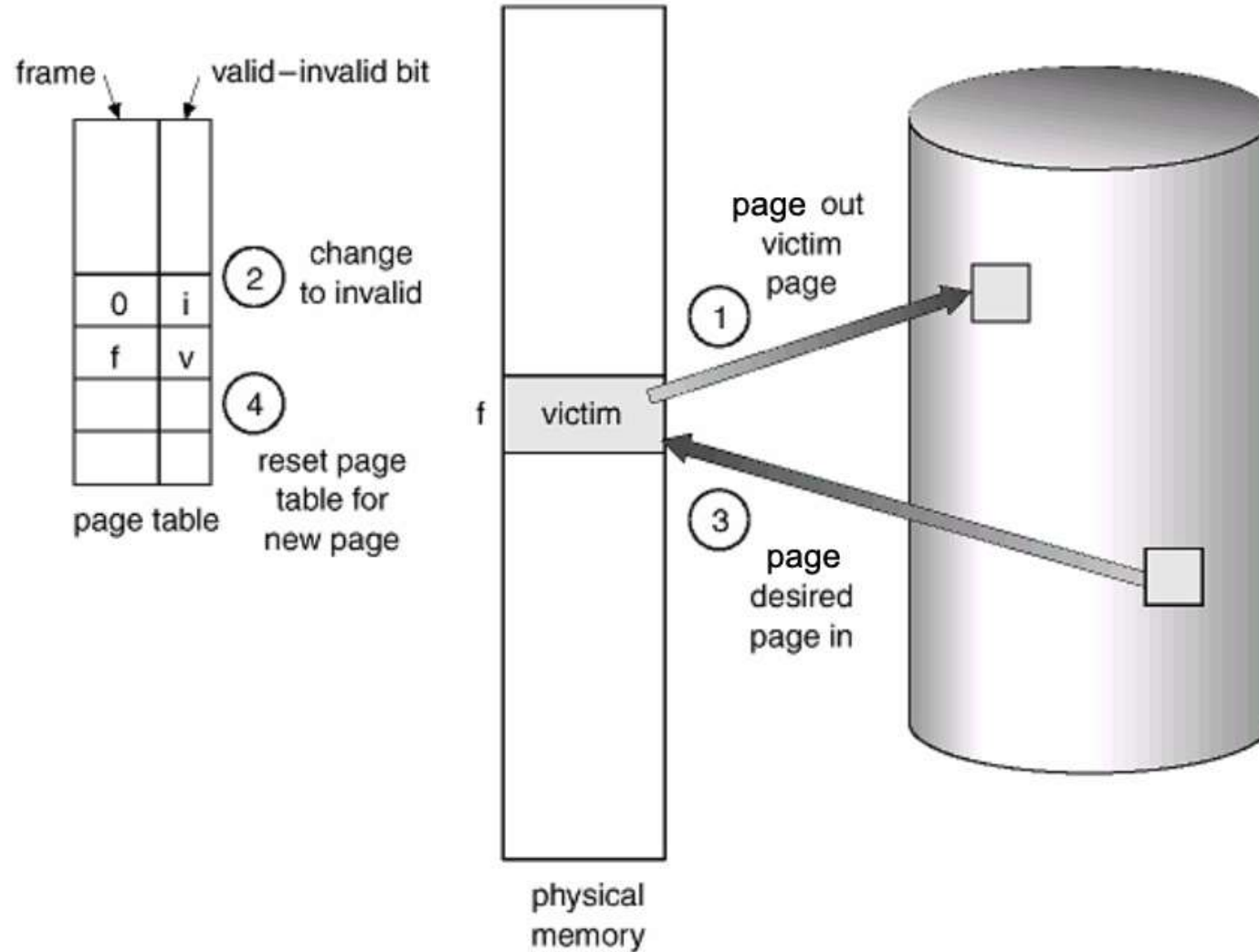
### ■ 일반적인 경우



### ■ VOD



# Page Fault with Page Replacement



# Text and Stack(heap) Replacement

---

## Text segment

- Swap 영역에 저장할 필요 없음 (not swap-out)
- 어차피 디스크에 저장되어 있는 영역 + 변하지 않음

## Stack, Heap, BSS, ... (Anonymous memory)

- 반드시 swap space에 저장을 해놔야 나중에 복구할 수 있다.

swap space 관련해서는 사실 엄청 복잡함. 그래서 다른 곳에서도 설명을 잘 안한다.

# Page Replacement Algorithms

---

- Optimal Page Replacement
- FIFO
- LRU
- LRU Approximation
  - Additional-Reference
  - Second-Change
  - Enhanced Second-Change



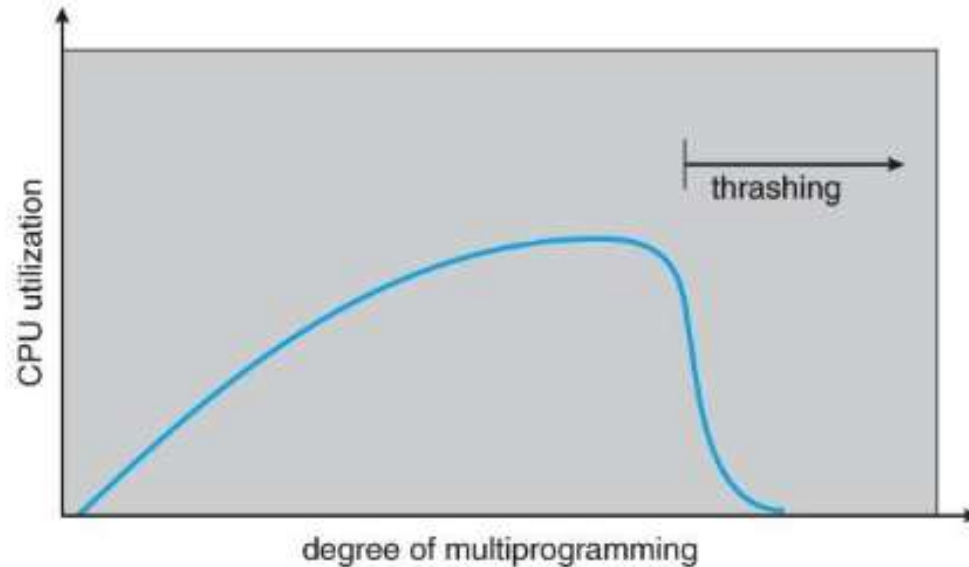
# Thrashing

프로세스들이 실행되는 시간보다 페이지를 교체하는 데에 더 많은 시간이 소요되는 현상

Thrashing은 Page-fault Frequency 문제

임의 프로세스에 대하여 원하는 페이지 오류율의 상한선과 하한선을 정해놓고

- 상한선을 넘으면 더 많은 프레임을 할당
- 하한선을 넘으면 프레임을 회수



궁극적 해결책은 충분한 메모리를 제공하는 것

# Reference

---

operating system(korea university, 유혁)

Operating System Concepts (10/E, Silberschatz)

Computer Systems A Programmer's Perspective (3/E, Randal E. Bryan)

Computer Organization And Design (6/E, David A. Patterson)

# Index

---

## 1부: 소켓 프로그래밍과 HTTP

내일 13시, 19시

Java] Socket, Stream

HTTP] Basic

## 2부: Tomcat

Java] Multi-threads

HTTP] Add requirement

HTTP] HttpRequest, HttpResponse

HTTP] Servlet

## 3부: Spring

Java] Reflection, Annotation

HTTP] Final