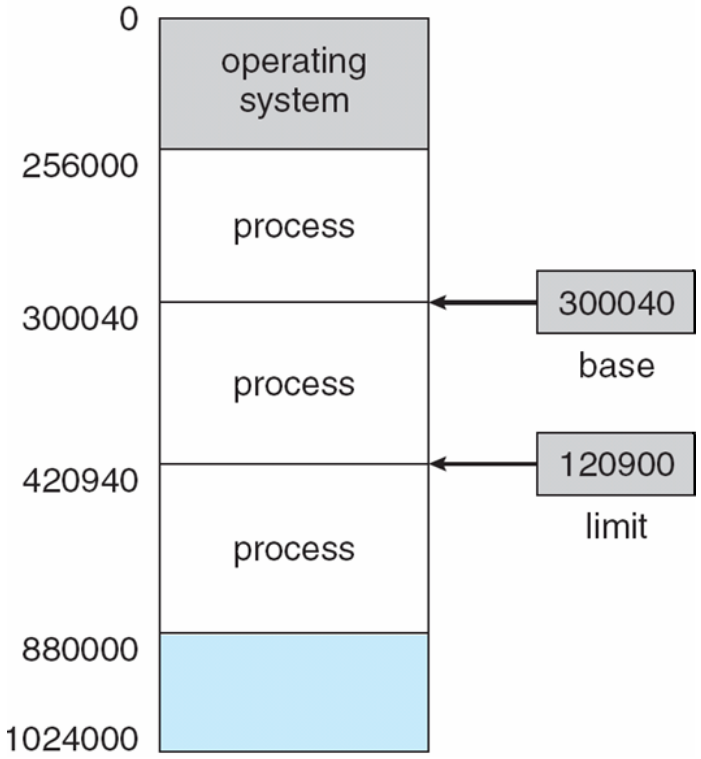
Operating Systems (OS)
Memory Management
(Paging 중심)

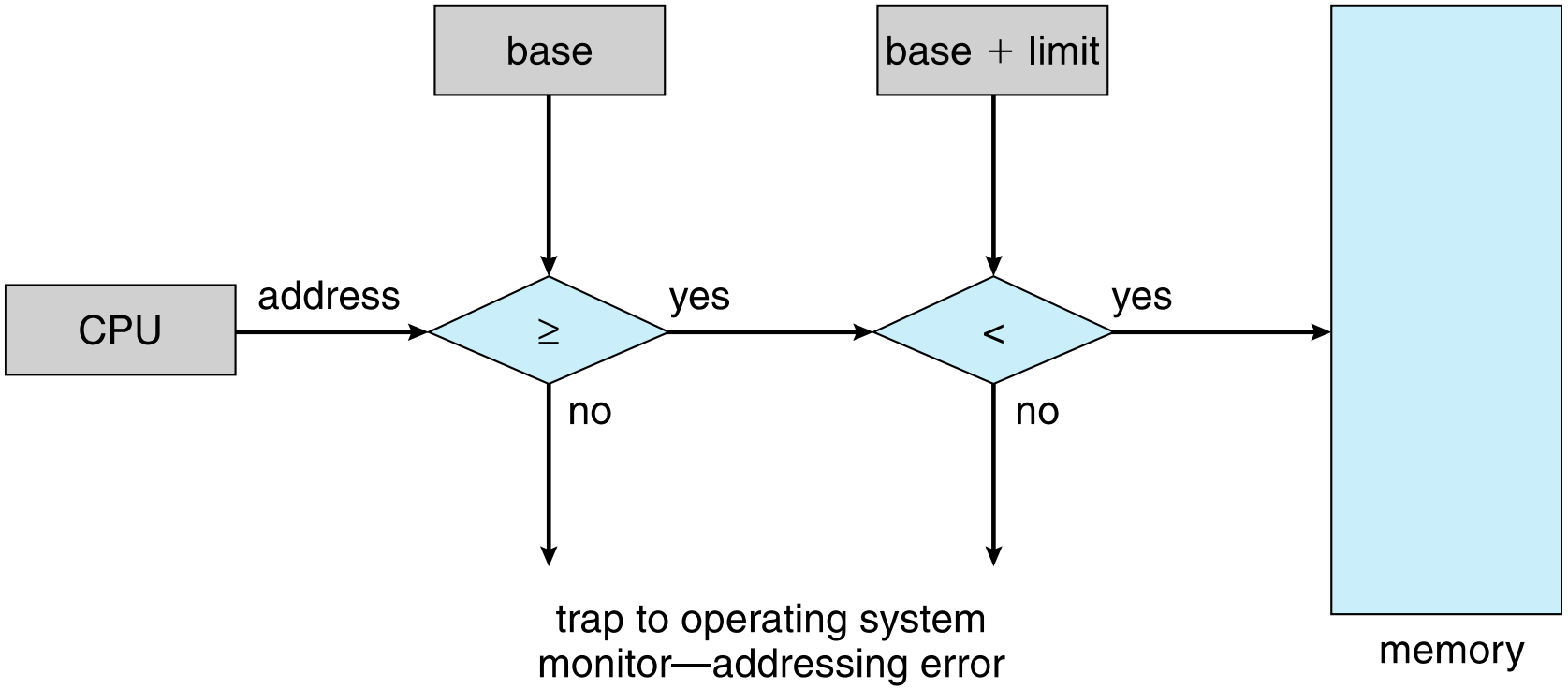
Prof. Eun-Seok Ryu (esryu@skku.edu)
Multimedia Computing Systems Laboratory
http://mcsl.skku.edu
Department of Immersive Media Engineering
Department of Computer Education (J.A.)
Sungkyunkwan University (SKKU)

Base and Limit Registers

- A pair of **base** and **limit registers** define the logical address space
- CPU must check every memory access generated in user mode to be sure it is between base and limit for that user



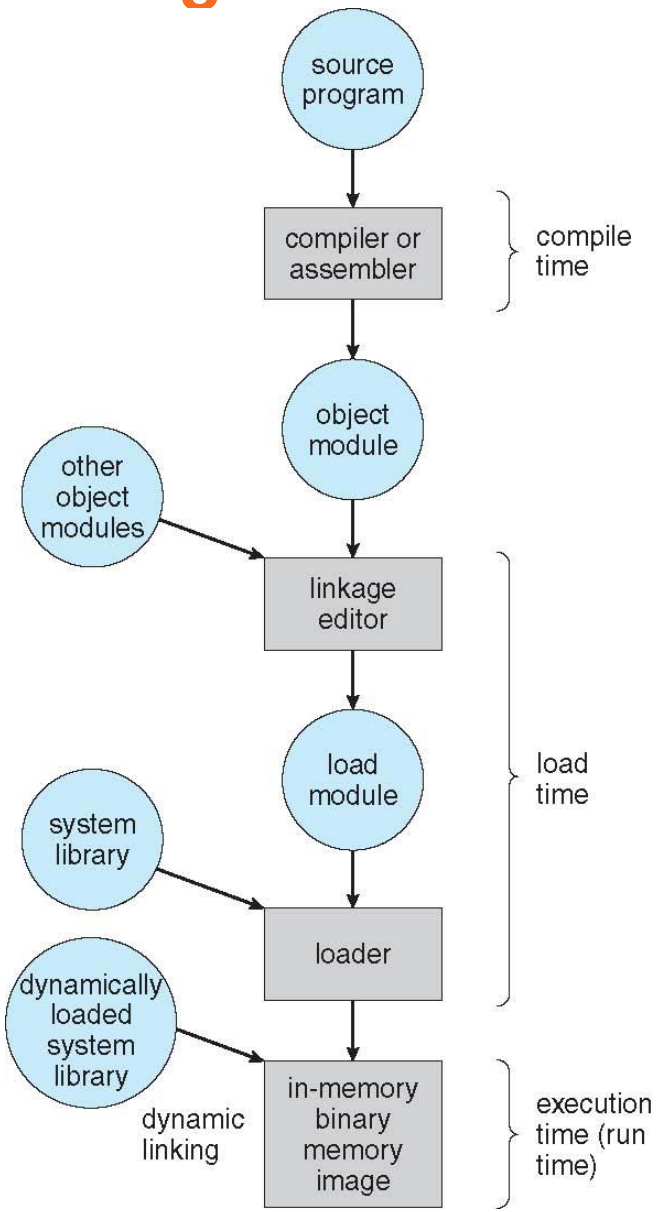
Hardware Address Protection



Binding of Instructions and Data to Memory

- Address binding of instructions and data to memory addresses can happen at three different stages
  - **Compile time**: If memory location known a priori, **absolute code** can be generated; must recompile code if starting location changes
  - **Load time**: Must generate **relocatable code** if memory location is not known at compile time
  - **Execution time**: Binding delayed until run time if the process can be moved during its execution from one memory segment to another
    - Need hardware support for address maps (e.g., base and limit registers)

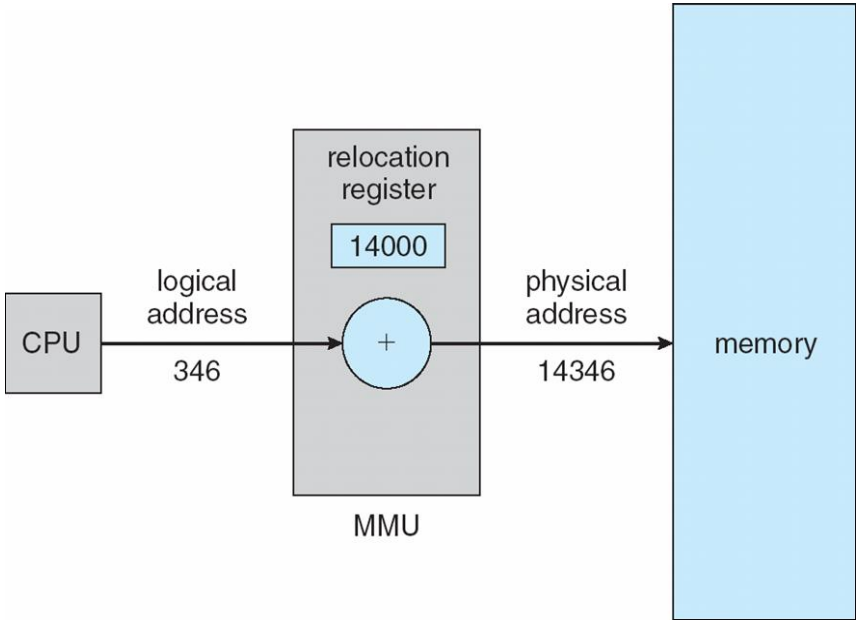
Multistep Processing of a User Program



Memory Management Unit (MMU): Hardware device that at run time maps virtual to physical address

Dynamic relocation using a relocation register

- Routine is not loaded until it is called
- Better memory-space utilization; unused routine is never loaded
- All routines kept on disk in relocatable load format
- Useful when large amounts of code are needed to handle infrequently occurring cases
- No special support from the operating system is required
  - Implemented through program design
  - OS can help by providing libraries to implement dynamic loading



Dynamic Linking

- Static linking: system libraries and program code combined by the loader into the binary program image
- Dynamic linking: linking postponed until execution time
- Small piece of code, **stub**, used to locate the appropriate memory-resident library routine
- Stub replaces itself with the address of the routine, and executes the routine
- Operating system checks if routine is in processes′ memory address
  - If not in address space, add to address space
- Dynamic linking is particularly useful for libraries
- System also known as **shared libraries**
- Consider applicability to patching system libraries
  - Versioning may be needed

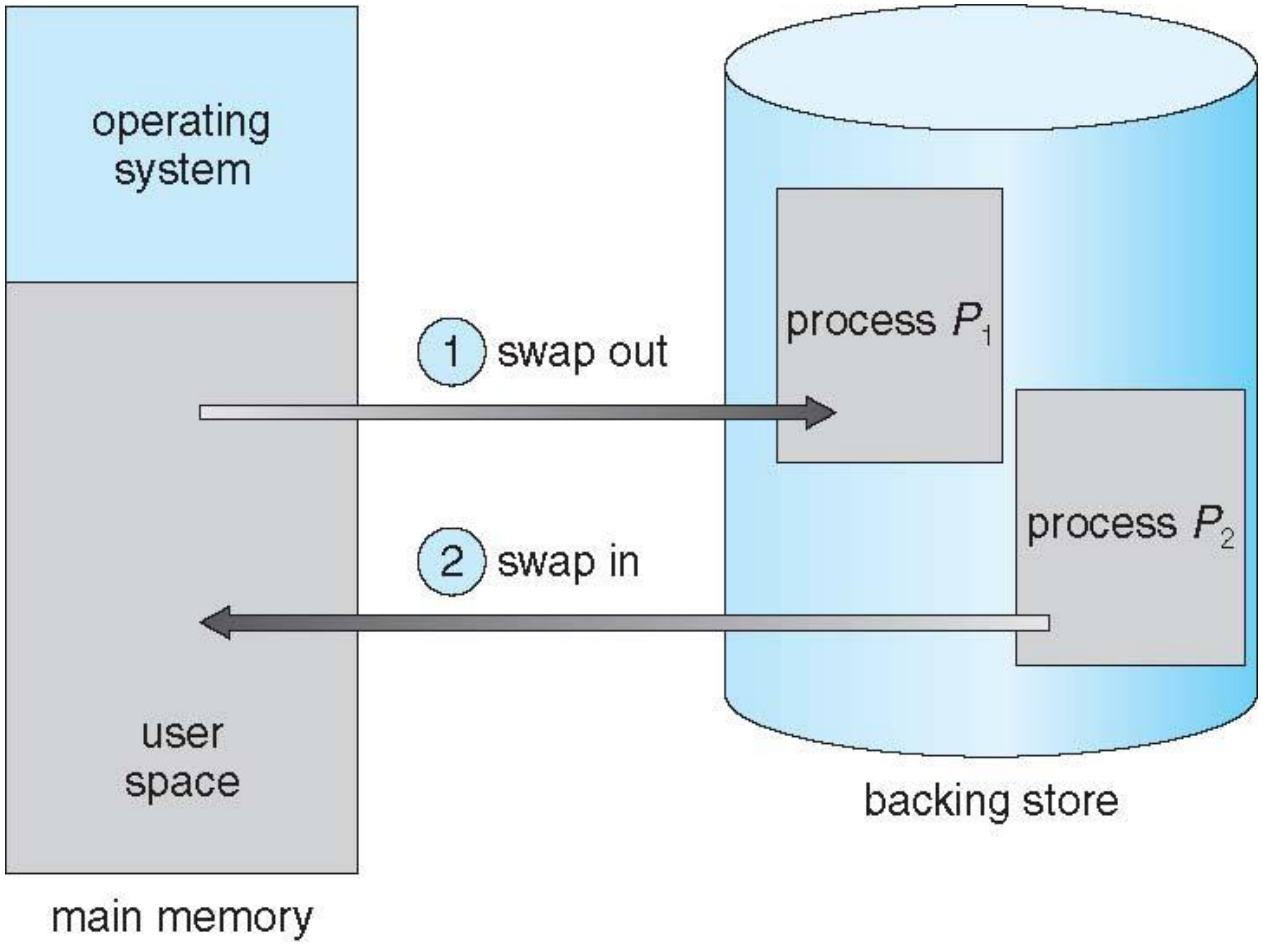
Swapping

- A process can be **swapped** temporarily out of memory to a backing store, and then brought back into memory for continued execution
  - Total physical memory space of processes can exceed physical memory
- **Backing store** – fast disk large enough to accommodate copies of all memory images for all users; must provide direct access to these memory images
- **Roll out, roll in** – swapping variant used for priority-based scheduling algorithms; lower-priority process is swapped out so higher-priority process can be loaded and executed
- Major part of swap time is transfer time; total transfer time is directly proportional to the amount of memory swapped
- System maintains a **ready queue** of ready-to-run processes which have memory images on disk

Swapping (Cont.)

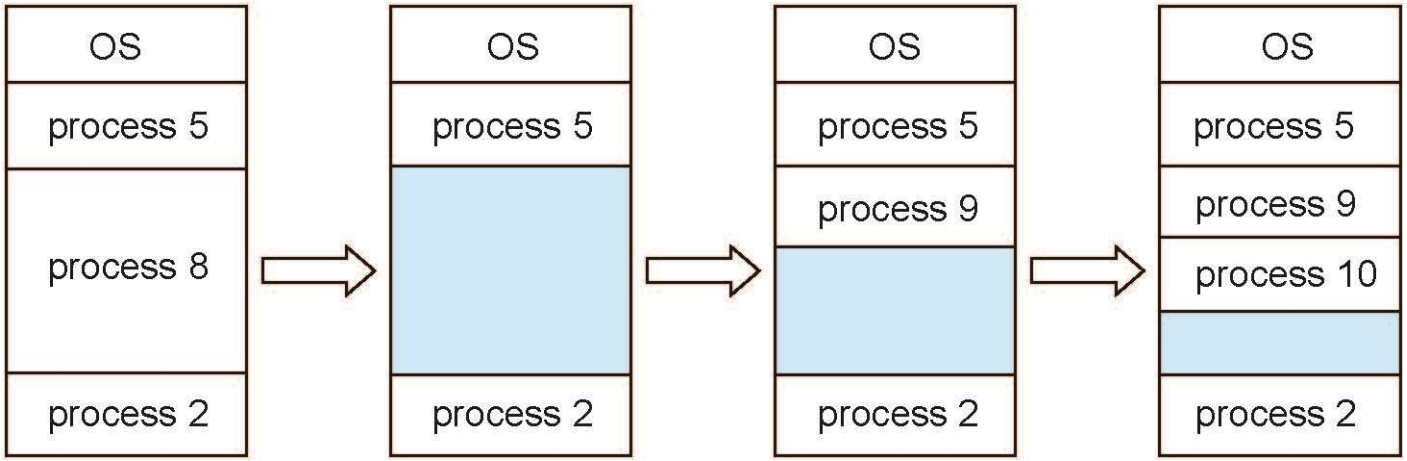
- Does the swapped out process need to swap back in to same physical addresses?
- Depends on address binding method
  - Plus consider pending I/O to / from process memory space
- Modified versions of swapping are found on many systems (i.e., UNIX, Linux, and Windows)
  - Swapping normally disabled
  - Started if more than threshold amount of memory allocated
  - Disabled again once memory demand reduced below threshold

Schematic View of Swapping



Multiple-partition allocation

- Multiple-partition allocation
  - Degree of multiprogramming limited by number of partitions
  - **Variable-partition** sizes for efficiency (sized to a given process' needs)
  - **Hole** – block of available memory; holes of various size are scattered throughout memory
  - When a process arrives, it is allocated memory from a hole large enough to accommodate it
  - Process exiting frees its partition, adjacent free partitions combined
  - Operating system maintains information about:
    a) allocated partitions b) free partitions (hole)



Dynamic Storage-Allocation Problem

How to satisfy a request of size ***n*** from a list of free holes?

- **First-fit**: Allocate the ***first*** hole that is big enough
- **Best-fit**: Allocate the ***smallest*** hole that is big enough; must search entire list, unless ordered by size
  - Produces the smallest leftover hole
- **Worst-fit**: Allocate the ***largest*** hole; must also search entire list
  - Produces the largest leftover hole

First-fit and best-fit are better than worst-fit in terms of speed and storage utilization

Fragmentation

- **External Fragmentation** – total memory space exists to satisfy a request, but it is not contiguous
- **Internal Fragmentation** – allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used

- Reduce external fragmentation by **compaction**
  - Shuffle memory contents to place all free memory together in one large block

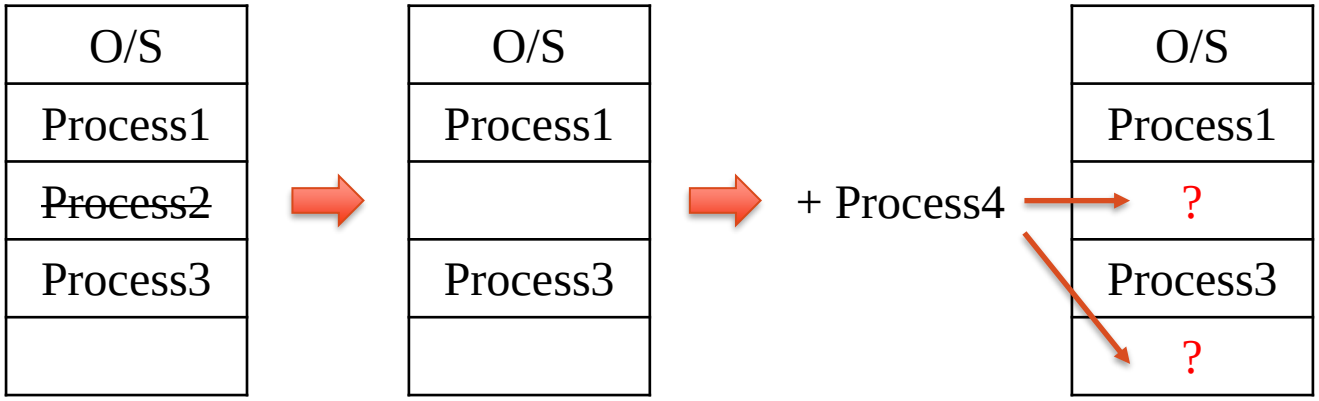
Paging (중요!)

- Physical address space of a process can be noncontiguous; process is allocated physical memory whenever the latter is available
  - Avoids external fragmentation
  - Avoids problem of varying sized memory chunks
- Divide physical memory into fixed-sized blocks called **frames**
  - Size is power of 2, between 512 bytes and 16 Mbytes
- Divide logical memory into blocks of same size called **pages**
- Keep track of all free frames
- To run a program of size ***N*** pages, need to find ***N*** free frames and load program
- Set up a **page table** to translate logical to physical addresses
- Still have Internal fragmentation

Fragmentation Problem

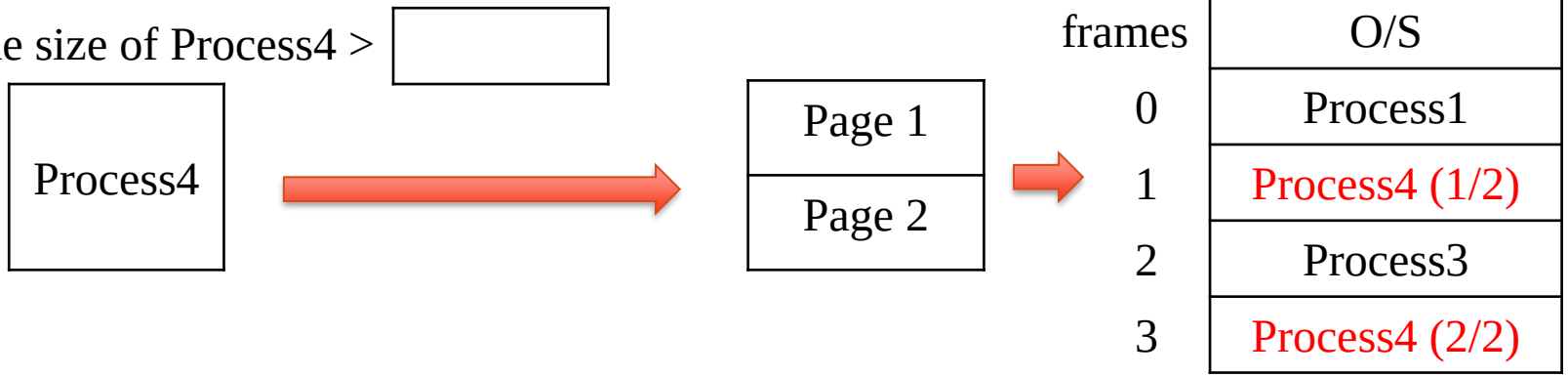
- CPU – Memory – Storage(HDD)
- Fragmentation

memory

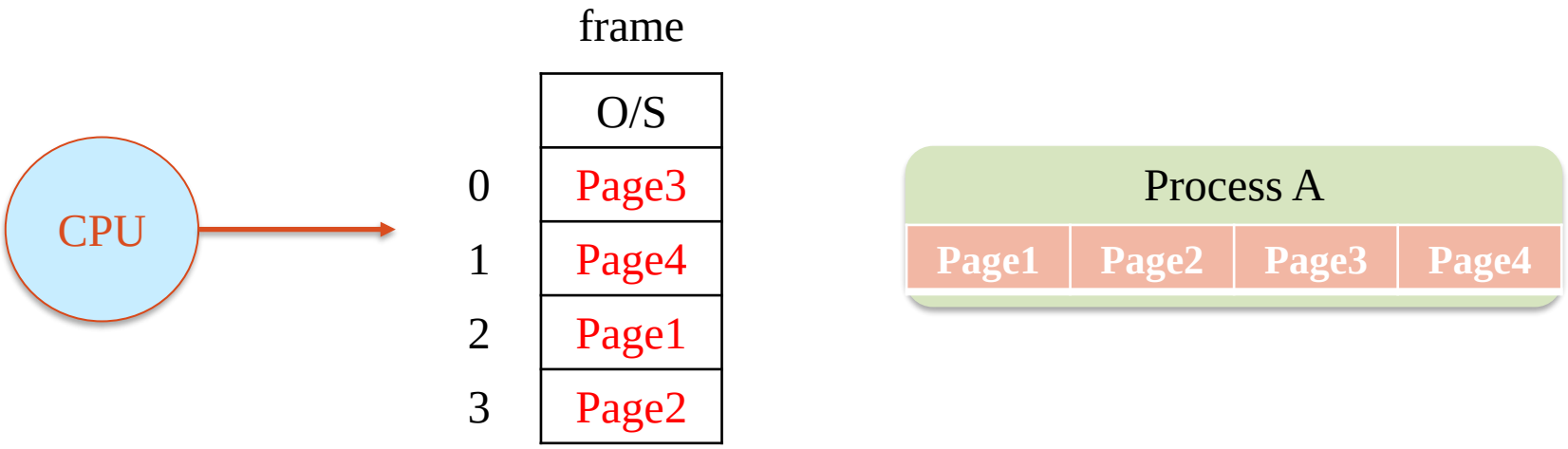


- Noncontiguous Process

If the size of Process4 >



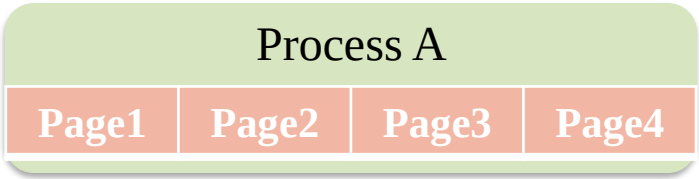
Fragmentation and page address

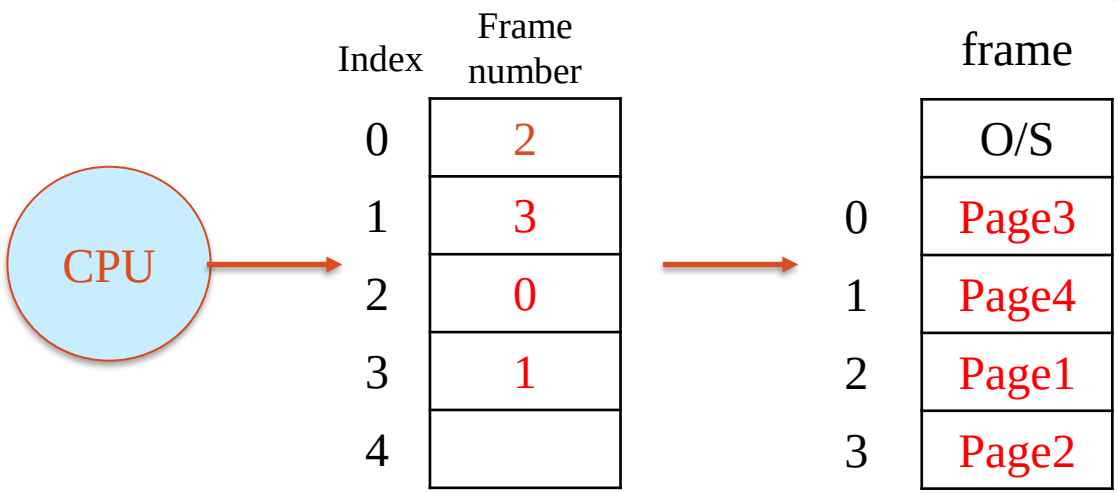


Q&A time

1. Fragmentation 때문에, 프로세스의 page순서가 뒤죽박죽일 수 밖에 없음.
2. CPU는 Process를 순서대로 메모리에 올려서 수행해야 하므로 순서가 맞아야 함.
3. 방법은???

Page Table and Address Translation





From the CPU's point of view, the address is continuous.

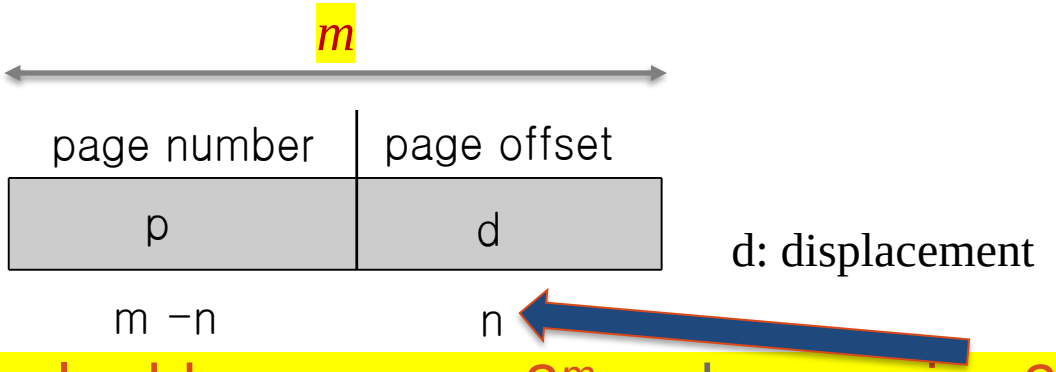
- CPU가 JMP 1000 같은 주소를 제시할 때 필요
- 따라서 페이지 테이블은 각 프로세스 마다 가지고 있음, 연산은 MMU가 함.

- MMU와 TLB는 대부분 CPU에 삽입 되어 있음.
- Page Table은 Memory에 위치

Address Translation

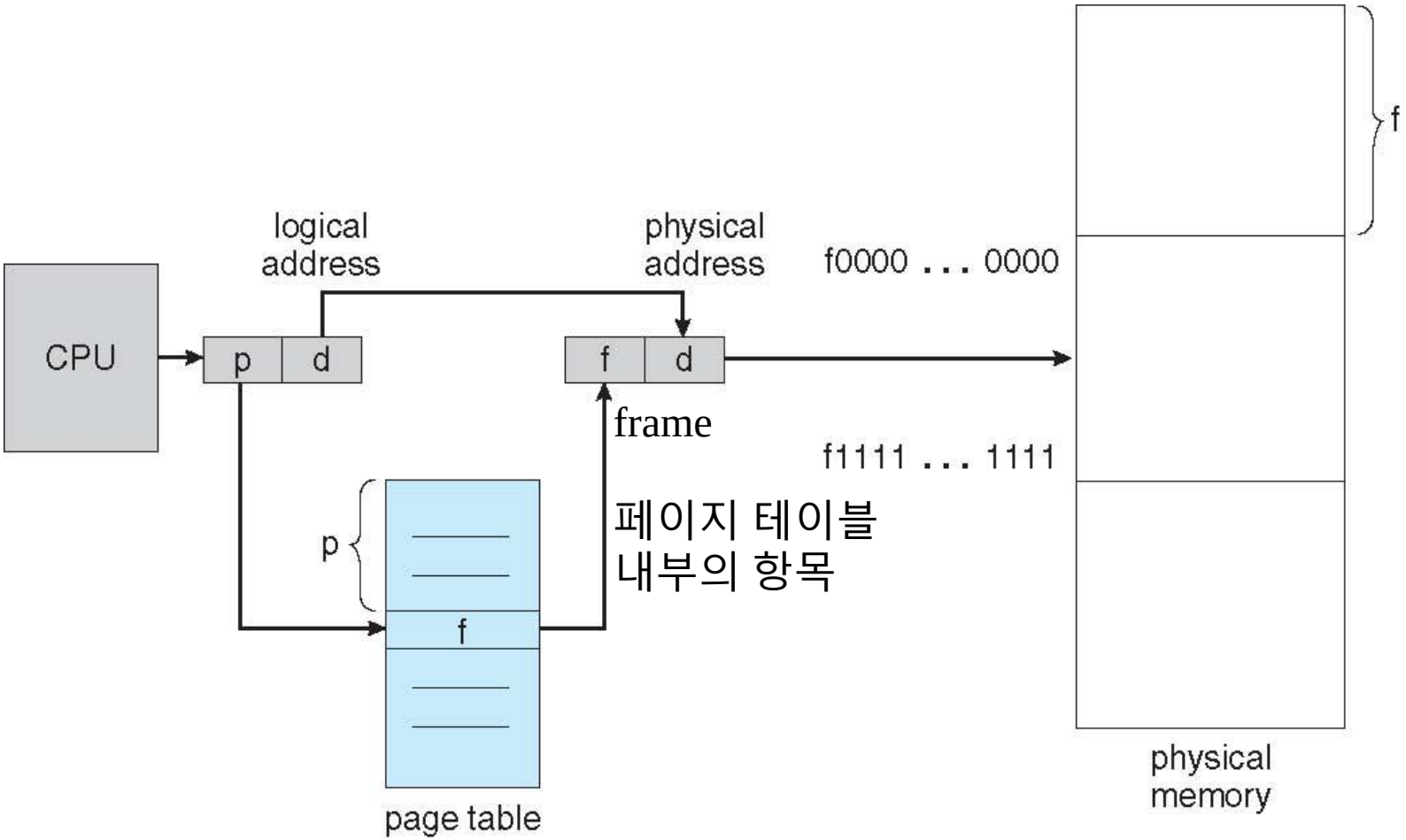
Address Translation Scheme

- Address generated by CPU is divided into:
  - **Page number** (***p***) – used as an index into a **page table** which contains base address of each page in physical memory
  - **Page offset** (***d***) – combined with base address to define the physical memory address that is sent to the memory unit

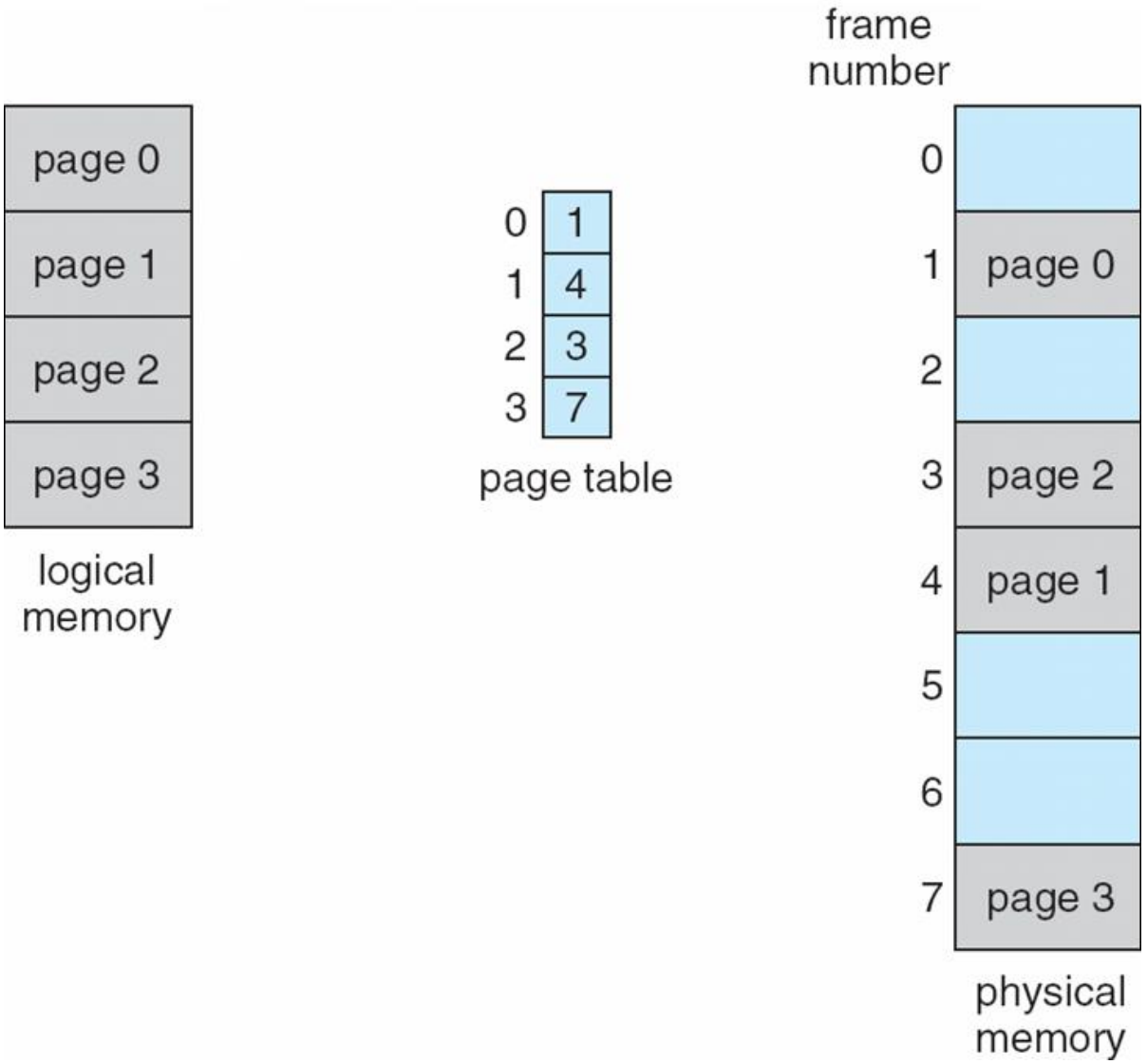


  - For given logical address space 2^m and page size 2^n

Paging Hardware



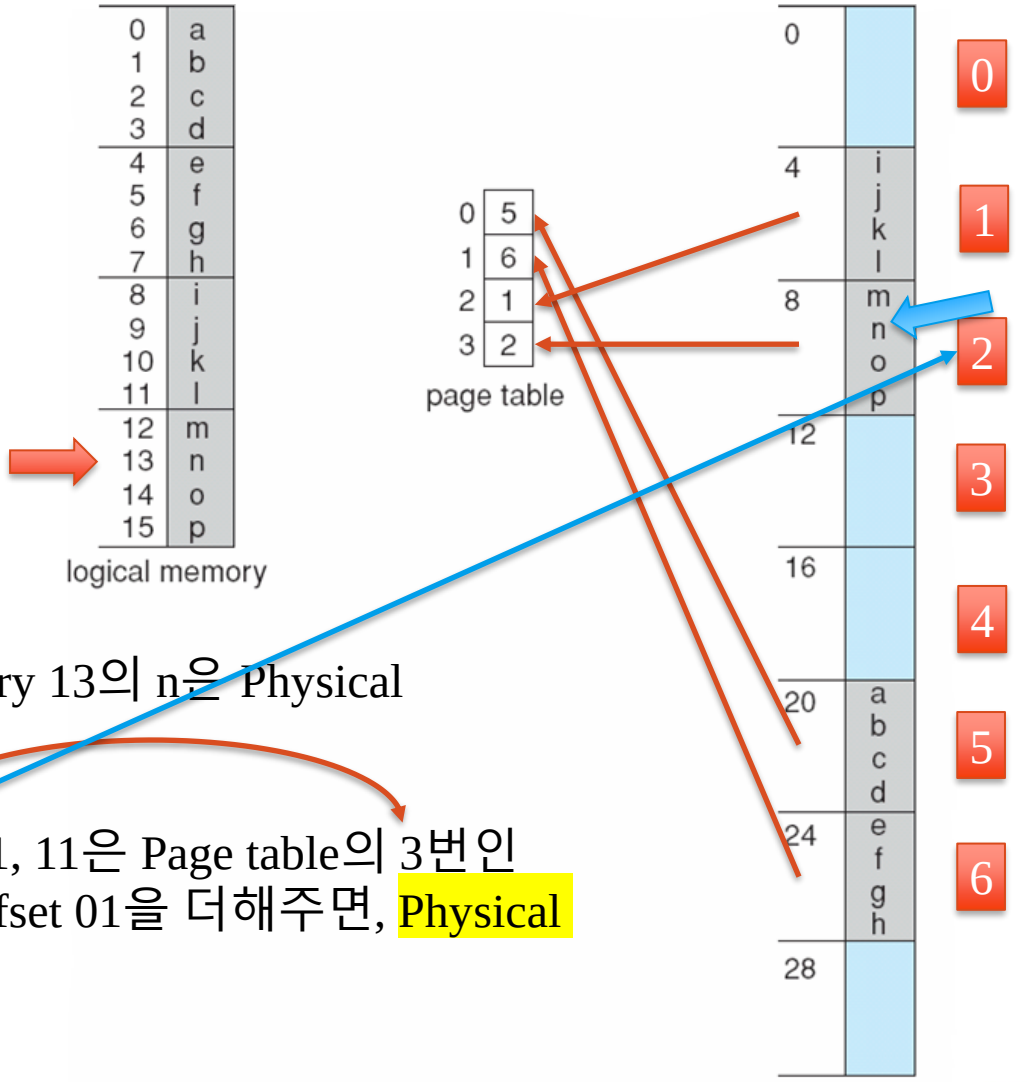
Paging Model of Logical and Physical Memory



Paging Example

- *n*=2 and *m*=4 32-byte memory and 4-byte pages

Slide 17의 m, n 참조



- 예로 Logical memory 13의 n은 Physical memory 9에 있음

- 13 = 1101(2), 11 | 01, 11은 Page table의 3번인 'frame 2' 그리고 offset 01을 더해주면, Physical 주소는 09가 됨

Address Translation

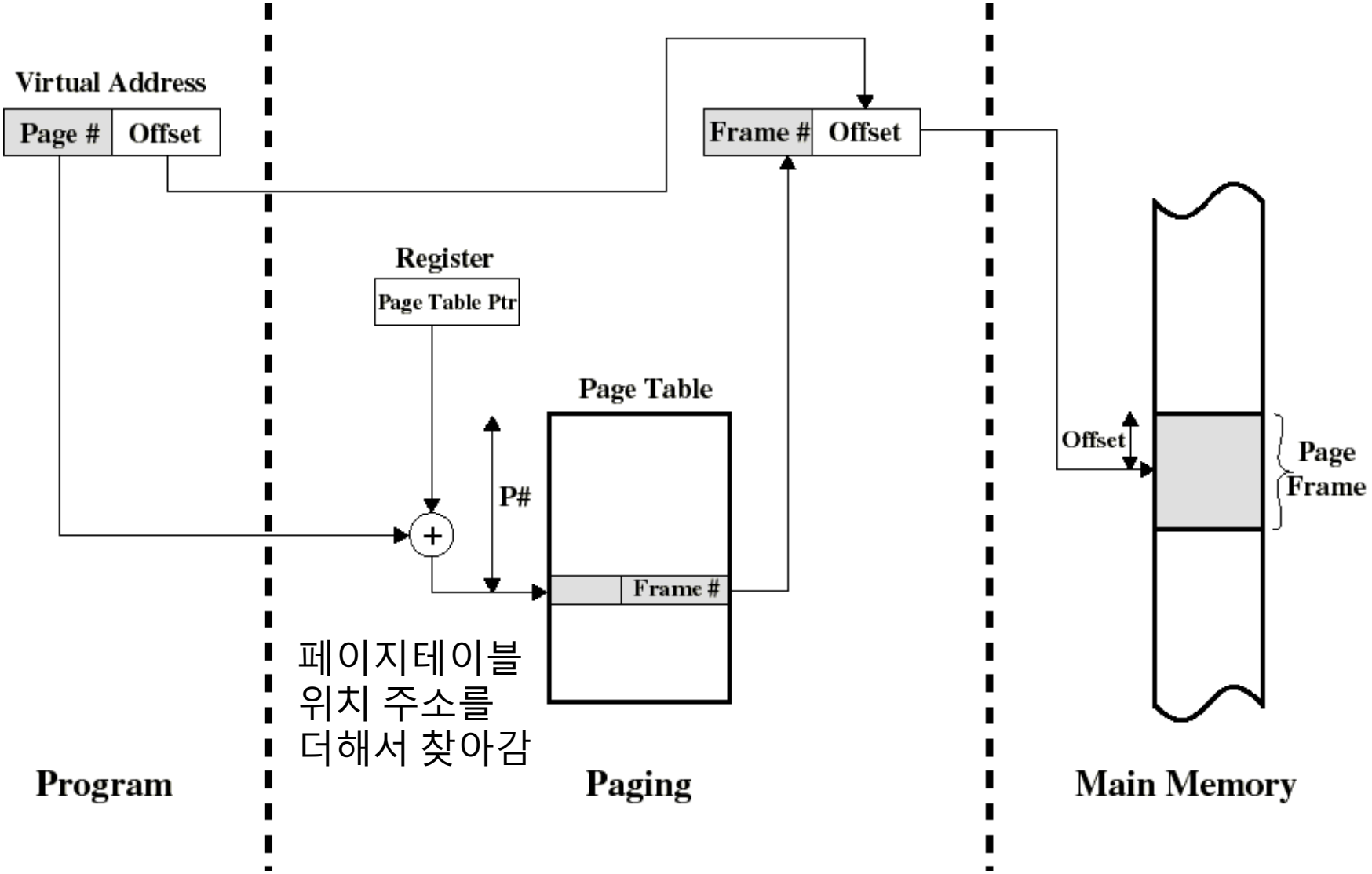


*figure from the mclab of silla Univ.

Q&A time

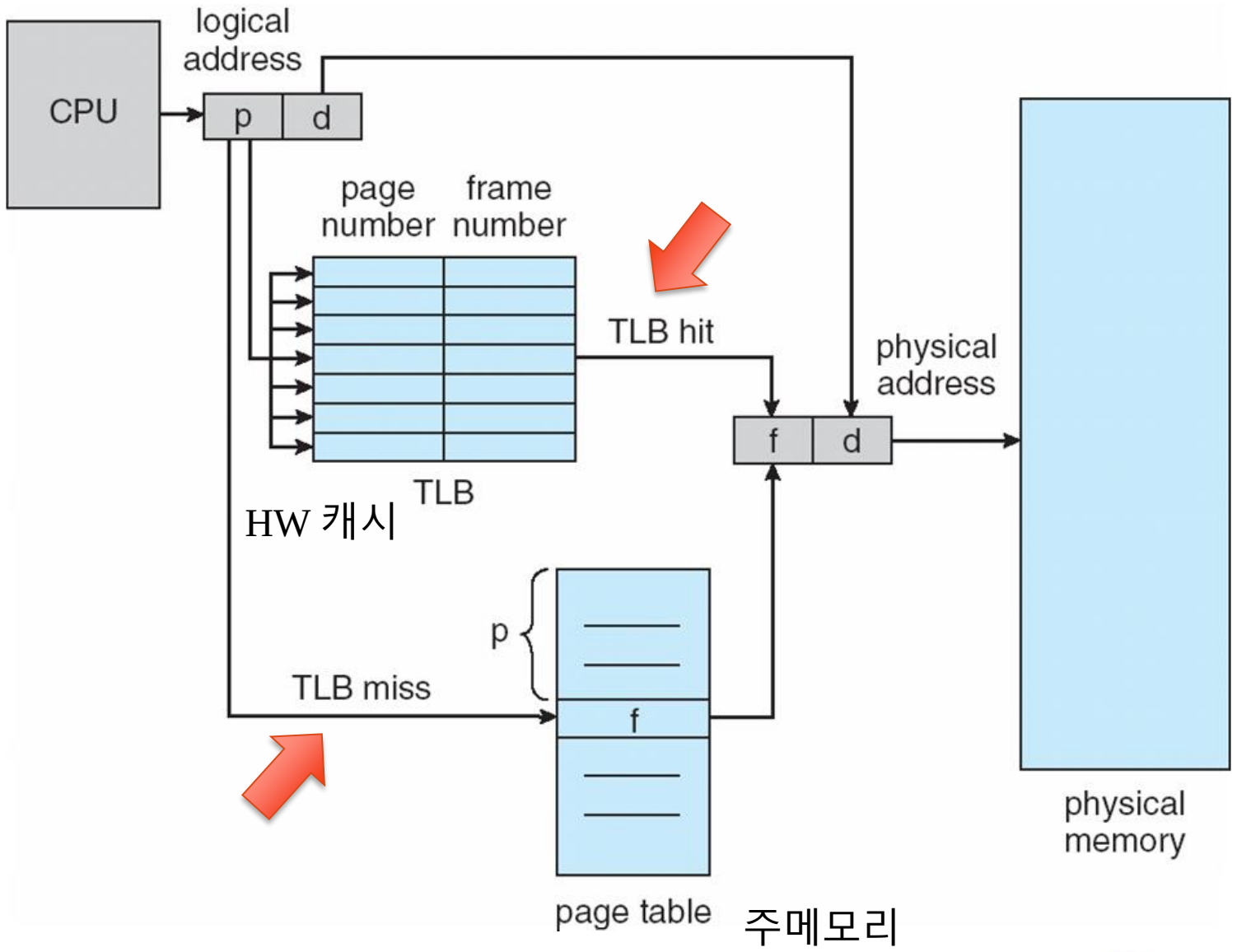
1. Internal fragmentation? Process가 여러 page로 나뉠 때 마지막 page가 다 차지 않고 조금 남을 경우 = 메모리 낭비.
   Then, 최대 internal fragmentation의 낭비는? Page size – 1 byte
   If page size = 1KB, then, 1023 byte.
2. Partial Loading? 장점? (1) 더 많은 프로세스가 메모리에 적재, (2) 프로세스가 메모리 크기보다 더 큰 경우에도 실행 가능
3. 16 bits의 물리적 주소를 사용하는 경우,
   1. Physical memory 접근 가능 사이즈는? 2^16=64KB(65536)의 물리적인 메모리 접근 가능 (2^10 = 1024, 1KB)
   2. Page 크기를 1KB로 정하면 Offset을 나타내기 위한 bit수는? 10 bits 필요
   3. 이 때, 물리적으로 page를 나타내기 위한 bit수는? 6 bits (16-10)
4. Page size = 4 bytes, Page table = 5, 6, 1, 2 Then, logical address 13의 Physical address? 13=1101(2), 11 | 01 (4B고려), 11번 즉 3번엔 2가 있음, 2는 10, d인 01을 합치면 1001, 즉 1001(2)=9. 따라서 답 9.

Q&A time – Cont'd

1. Page size = 1 KB, Page table = 1, 2, 5, 4, 8, 3, 0, 6
   Then, (1) logical address 3000의 Physical address? (2) Physical address 0x1A53의 logical address?
   1KB는 2^10. 즉, 10bit가 displacement가 됨. 3000을 2진수화 10|1110111000(2), 따라서, Page table의 10(2)즉2는 5임. 거기에 나머지 10bit displacement 적용하면, 101110111000(2)임

2. Page table의 위치는? CPU register? Main memory?
   > TLB (Translation Look-aside Buffers) 특수한 소형 하드웨어 캐시

Paging Hardware With TLB



Summary

- 프로세스는 독립적 메모리 공간을 가짐
  - >> Base와 Limit으로 특정 메모리 공간을 사용하게 함
  - 문제점: Task switching 발생 시 메모리 내용을 외부 메모리로의 빈번한 swap
  - 디스크에서 주메모리로 들어오기를 기다리는 프로세스들의 집합은 input queue를 형성
- 실행시간 바인딩(Execution time binding)의 경우 논리주소(가상주소)와 물리주소 다름 -> HW기반의 MMU (Memory Management Unit)필요
- Swapping: roll-in / roll-out : Priority 정책에 따라 swap 가능
  - 실시간 바인딩으로 프로세스가 메모리내의 빈공간 어디든 들어감 (swap-in / swap-out) -> 문제점: Context-switch time
  - 대부분은 평소엔 Swap 안하고, 메모리 부족시에만 swap 작동

Summary – cont'd

- Memory Allocation: 프로세스의 요청대로 필요한 메모리 할당
  - 최초적합/최적적합/최악적합 (가장 큰 공간 선택)
  - 문제점: Fragmentation (단편화): 주로 외부 단편화 => 정렬? (너무 많은 비용)
  - => Paging과 Segmentation: 작은 공간으로 분할하고 요청시 그 공간의 배수 단위로 할당. 즉, 할당하는 메모리 사이즈를 고정하면? => Paging기법
  - 문제: 외부 단편화 없으나, 내부 단편화 발생
  - 해결? => Page 사이즈 줄이기
    - 문제점: 대신, 페이지 테이블 크기가 커짐 (공간 낭비)
- Page Table은 주메모리, Cache로는 TLB에 위치(소형 HW 캐시)
  - 같은 페이지 번호가 발견되면 그에 대응하는 프레임 번호를 알려줌=> TLB에 없으면 주메모리에서 찾음 => TLB관리/교체 이슈 (Virtual Memory 시간에 좀 더 다룸)