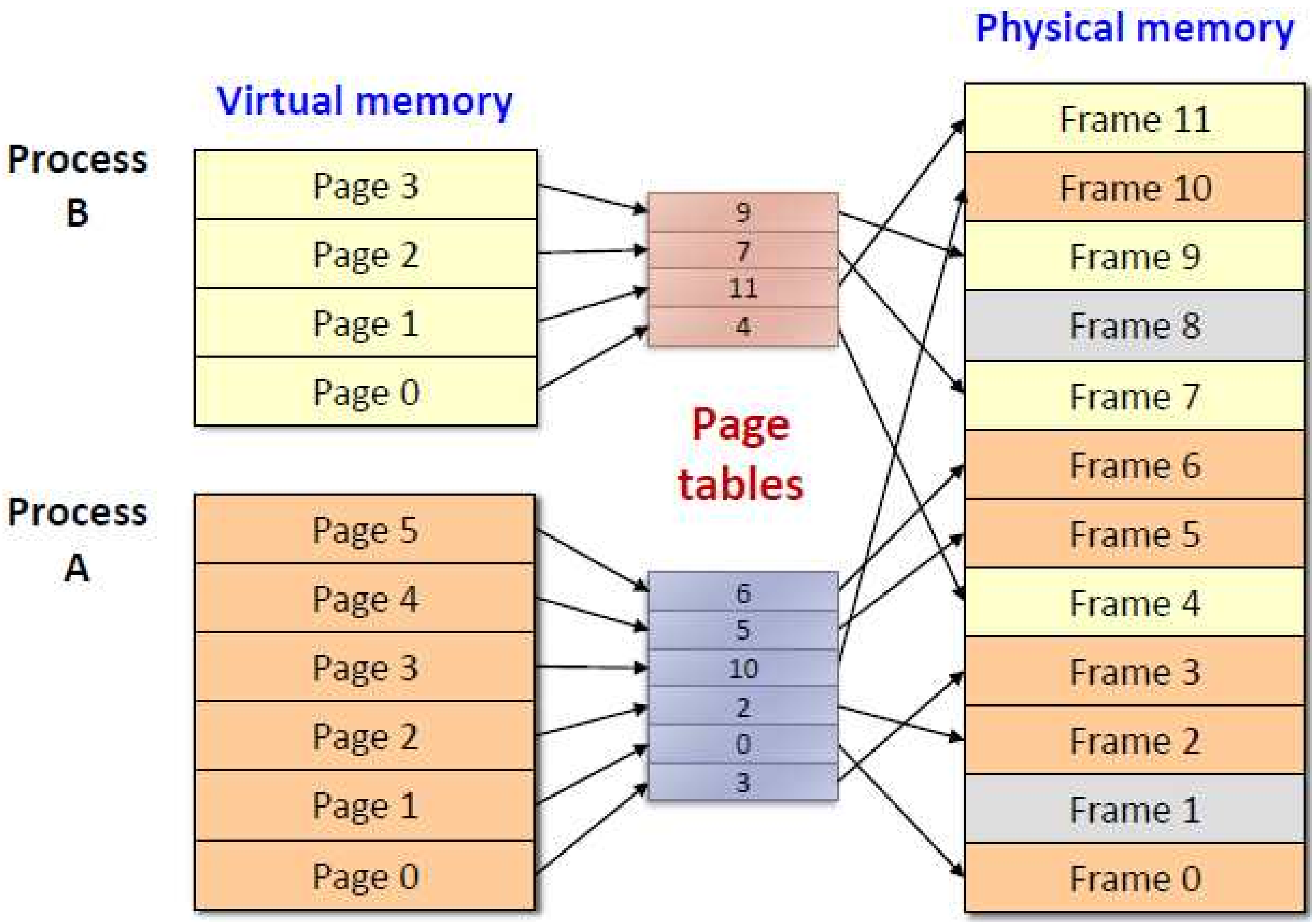
Operating System: Swapping Mechanisms

Sang Ho Choi (shchoi@kw.ac.kr)
School of Computer & Information Engineering
KwangWoon University

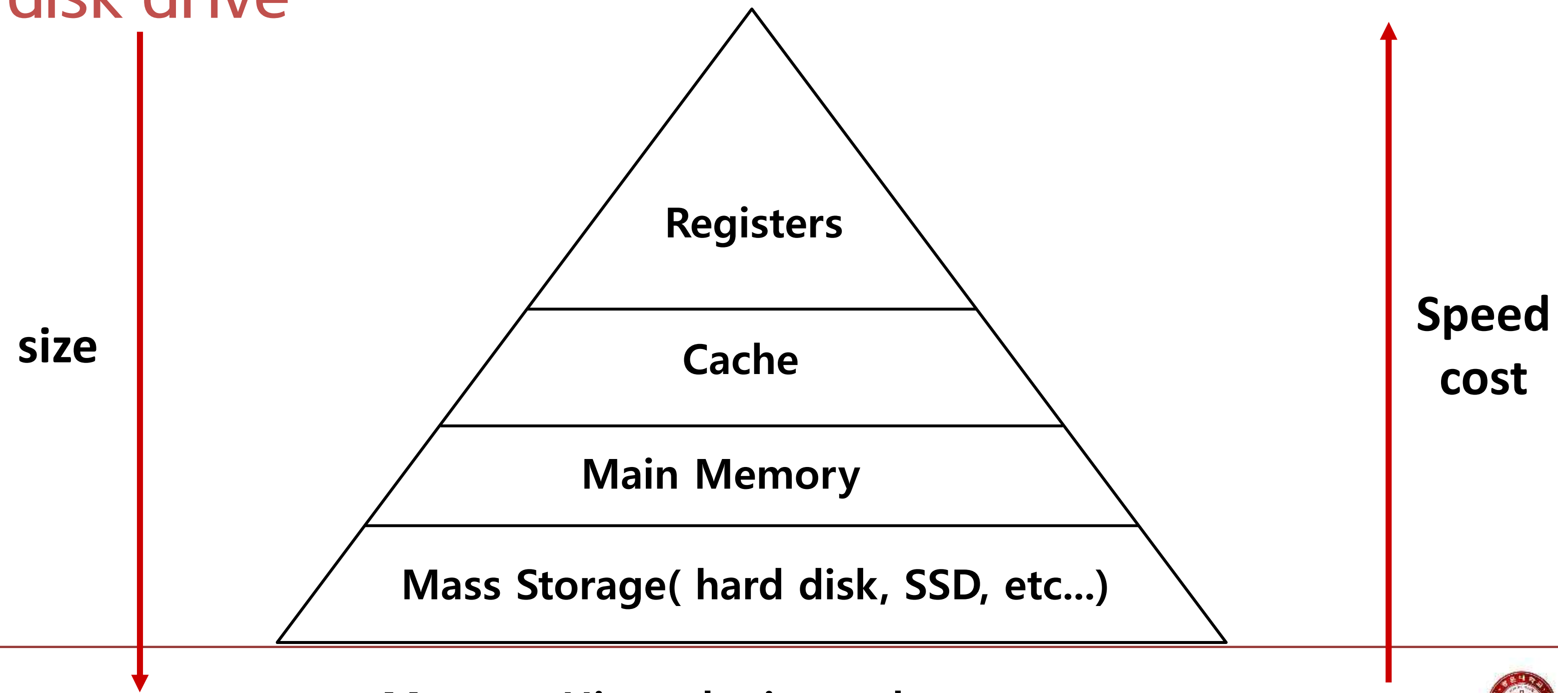
Virtual Memory



We have assumed that all pages reside in physical memory

Beyond Physical Memory: Mechanisms

- Support many concurrently-running large address spaces when not enough physical memory
- Require an additional level in the memory hierarchy
  - OS needs a place to stash away pages that currently aren't in great demand 페이지 저장할곳이 필요함
  - In modern systems, this role is usually served by a hard disk drive



Memory Hierarchy in modern system

How to Swap

Swapping 하는 방법.

- Overlays
  - Programmers manually move pieces of code or data in and out of memory as they were needed
  - No special support needed from OS

  코드작성시 직접적으로 이동시킴.

- Process-level swapping
  - A process is swapped temporarily out of memory to a backing store

    process 단위로 스와핑. 추후실행시 다시가져옴.
  - It's brought back into memory later for continued execution
- Page-level swapping

  페이지 단위로 보내고 가져옴
  - Swap pages out of memory to a backing store (swap-out)
  - Swap pages into memory from the backing store (swap-in)

Where to Swap

- Swap space
  - Disk space reserved for moving pages back and forth
  - The size of the swap space determines the maximum number of memory pages that can be in use
  - Block size is same as the page size → 페이지 단위로 움직이니 block과 size 같다.
  - Can be a dedicated partition or a file in the file system

파티션이나 파일형태로 파일시스템에서 할당.

	PFN 0	PFN 1	PFN 2	PFN 3
Physical Memory	Proc 0 [VPN 0]	Proc 1 [VPN 2]	Proc 1 [VPN 3]	Proc 2 [VPN 0]

	Block 0	Block 1	Block 2	Block 3	Block 4	Block 5	Block 6	Block 7
Swap Space	Proc 0 [VPN 1]	Proc 0 [VPN 2]	[Free]	Proc 1 [VPN 0]	Proc 1 [VPN 1]	Proc 3 [VPN 0]	Proc 2 [VPN 1]	Proc 3 [VPN 1]

Physical Memory and Swap Space

Present Bit

- Add some machinery higher up in the system in order to support swapping pages to and from the disk
  - When the hardware looks in the PTE, it may find that the page is not present in physical memory

	Value	Meaning	
page가 memory에	1	page is present in physical memory	
disk에	0	The page is not in memory but rather on disk.	page fault

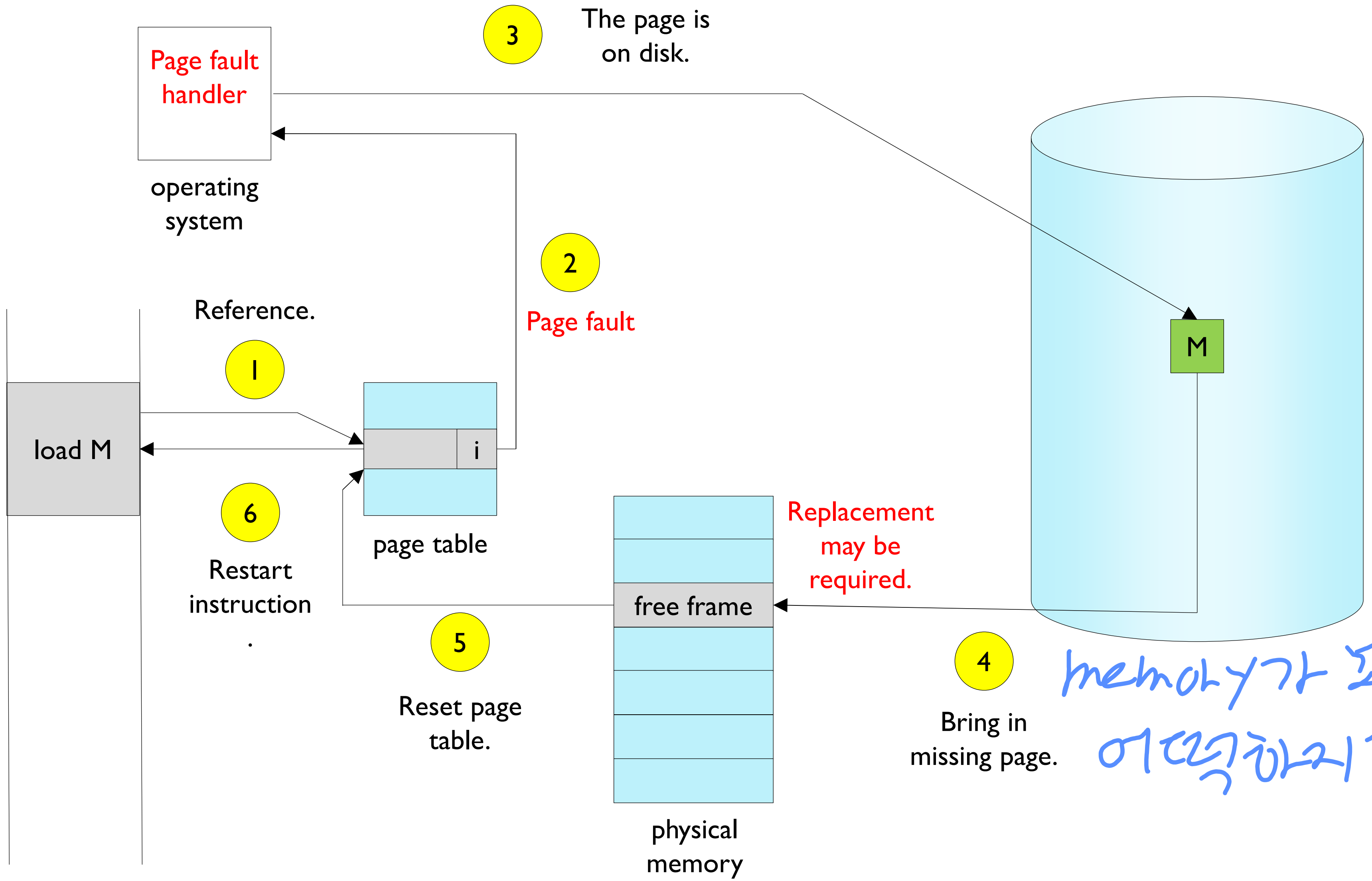
PTE의 비트를 보고.

The Page Fault

- Accessing page that is not in physical memory
  - If a page is not present and has been swapped disk, the OS need to swap the page into memory in order to service the page fault
  - A particular piece of code, known as a page-fault handler, runs, and must service the page fault

Page fault handler

- Page fault handling



memory가 꽉 차있으면 어떡하지?

What If Memory Is Full ?

- The OS like to page out pages to make room for the new pages the OS is about to bring in
  - The process of picking a page to kick out, or replace is known as page-replacement policy

= swap out

어떤거랑 대체할지?

Page Fault Control Flow - Hardware

```
VPN = (VirtualAddress & VPN_MASK) >> SHIFT
(Success, TlbEntry) = TLB_Lookup(VPN)
if (Success == True) // TLB Hit
    if (CanAccess(TlbEntry.ProtectBits) == True)
        Offset = VirtualAddress & OFFSET_MASK
        PhysAddr = (TlbEntry.PFN << SHIFT) | Offset
        Register = AccessMemory(PhysAddr)
    else RaiseException(PROTECTION_FAULT)
```

Page Fault Control Flow - Hardware

```
else // TLB Miss
PTEAddr = PTBR + (VPN * sizeof(PTE))
PTE = AccessMemory(PTEAddr)
if (PTE.Valid == False)
    RaiseException(SEGMENTATION_FAULT)
else
    if (CanAccess(PTE.ProtectBits) == False)
        RaiseException(PROTECTION_FAULT)
    else if (PTE.Present == True)
        // assuming hardware-managed TLB
        TLB_Insert(VPN, PTE.PFN, PTE.ProtectBits)
        RetryInstruction()
    else if (PTE.Present == False)
        RaiseException(PAGE_FAULT)
```

Page Fault Control Flow - Software

swap-in을 하기 위한 공간 찾기.

```
PFN = FindFreePhysicalPage()
if (PFN == -1) // no free page found
    PFN = EvictPage() // run replacement algorithm
    DiskRead(PTE.DiskAddr, pfn) // sleep (waiting for I/O)
    PTE.present = True // update page table with present
    PTE.PFN = PFN // bit and translation (PFN)
    RetryInstruction() // retry instruction
```

어떤 공간과 교체할지에 대한 알고리즘

가져오고

비트 update

재실행

swapping 흐름

- The OS must find a physical frame for the soon-be-faulted-in page to reside within
- If there is no such page, waiting for the replacement algorithm to run and kick some pages out of memory

When Replacements Really Occur

- OS waits until memory is entirely full, and only then replaces a page to make room for some other page
  - This is a little bit unrealistic, and there are many reason for the OS to keep a small portion of memory free more proactively

는 교체가 일어난다는 것을 가정하고 메모리의 일부분을 swap을 위한 빈공간으로 사용.

- Swap Daemon, Page Daemon
  - There are fewer than LW pages available, a background thread that is responsible for freeing memory runs
  - The thread evicts pages until there are HW pages available

LW page ≤ 메모리 빈공간 ≤ HW pages 이걸 유지

LW page 보다 작으면 background에서 메모리 해제하고 이를 HW page까지 만큼 빈공간 확보

What to Swap

- What happens to each type of page frame on low mem
  - Kernel code → Not swapped
  - Kernel data → Not swapped
  - Page tables for user processes → Not swapped
  - Kernel stack for user processes → Not swapped
  - User code pages → Dropped
  - User data pages → Dropped or swapped
  - User heap/stack pages → Swapped
  - Files mmap'ed to user processes → Dropped or go to file system
  - Page cache pages → Dropped or go to file system

이런건 유지 중요하고 자주쓰니

참고

? Mem에서 쫓기면 swap

즉각적인 반응이 필요해서 swap

Memory

swap

Drop

Swap | File system

- Page replacement policy chooses the pages to evict

  ➔ next class!!