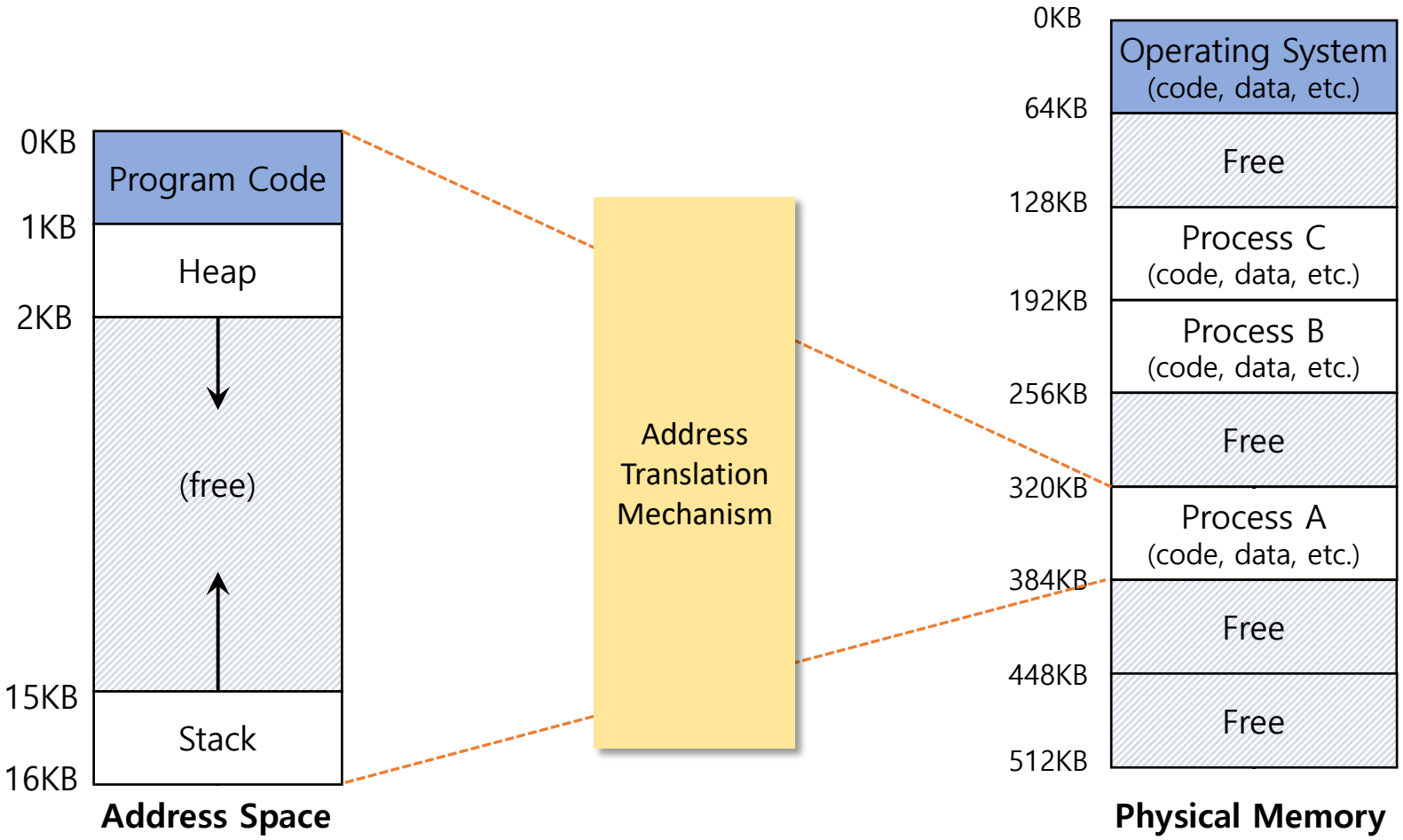
CDA0017: Operating Systems

Donghyun Kang (donghyun@changwon.ac.kr)

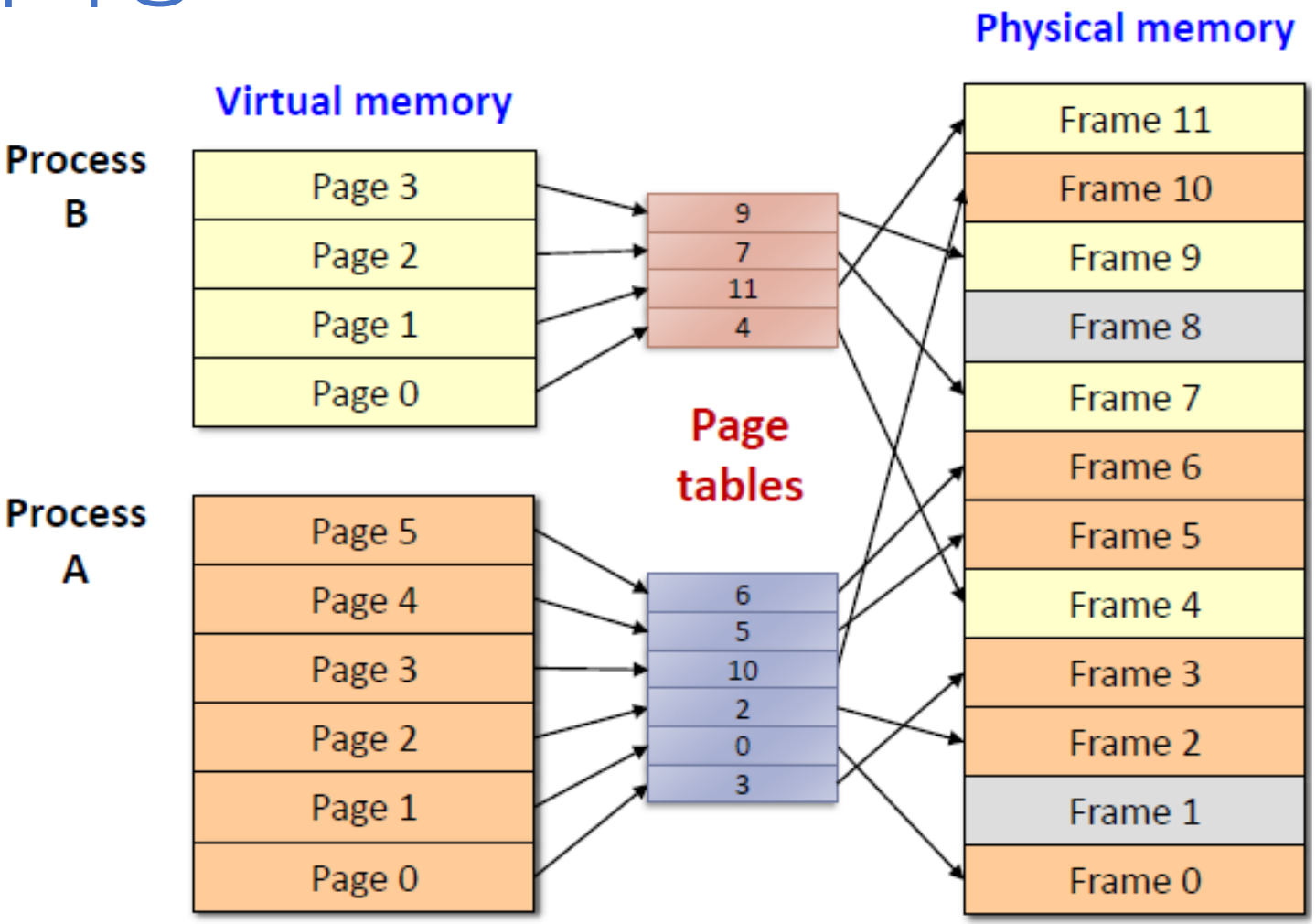
NOSLab (https://noslab.github.io)

Changwon National University

가상 주소 공간


0KB
Program Code
1KB
Heap
2KB
(free)
15KB
Stack
16KB
Address Space
Address Translation Mechanism
0KB
Operating System
(code, data, etc.)
64KB
Free
128KB
Process C
(code, data, etc.)
192KB
Process B
(code, data, etc.)
256KB
Free
320KB
Process A
(code, data, etc.)
384KB
Free
448KB
Free
512KB
Physical Memory

페이징


Physical memory
Virtual memory
Process B
Page 3
Page 2
Page 1
Page 0
Process A
Page 5
Page 4
Page 3
Page 2
Page 1
Page 0
Page tables
9
7
11
4
6
5
10
2
0
3
Frame 11
Frame 10
Frame 9
Frame 8
Frame 7
Frame 6
Frame 5
Frame 4
Frame 3
Frame 2
Frame 1
Frame 0

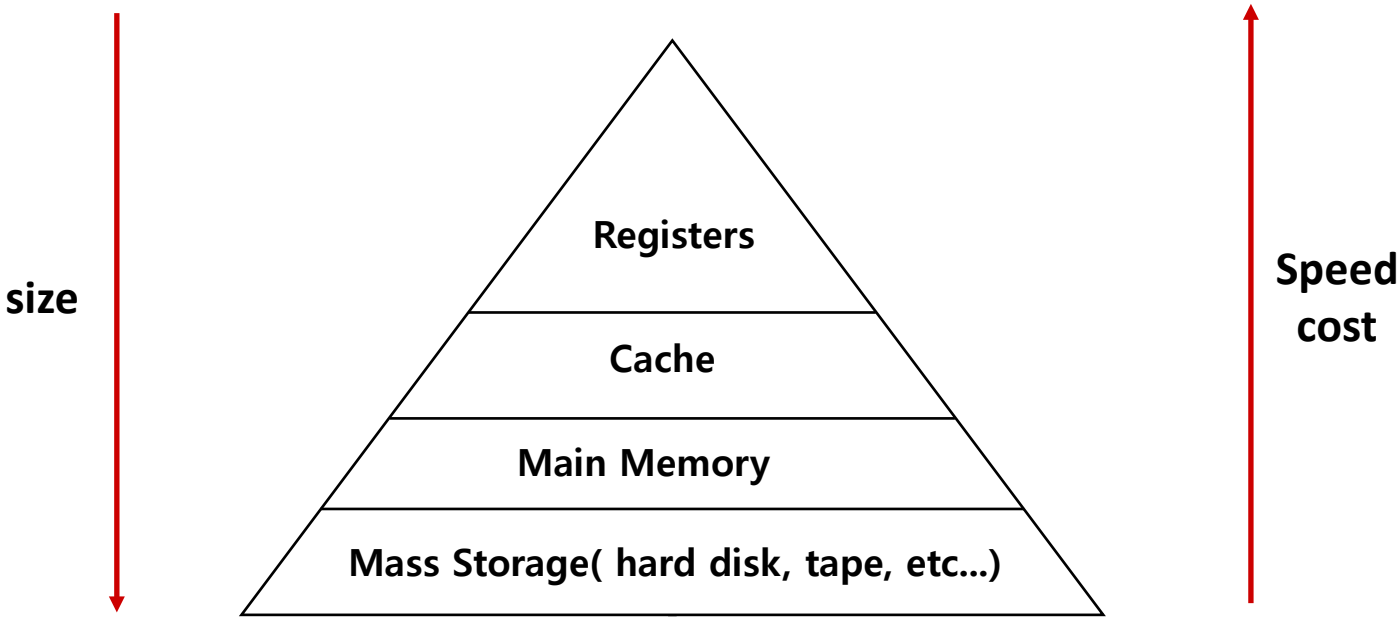
물리 메모리 크기의 극복: 메커니즘

- 많은 프로세스가 동시에 수행되는 경우, 물리 메모리는 항상 충분하지 않음

- 추가적인 메모리 구조를 요구함
  - 운영체제는 사용되지 않는 물리 메모리 공간의 일부를 지우고 싶음.
  - HDD가 물리 메모리 역할을 지원할 수 있음

메모리 구조

- 메모리 계층 구조



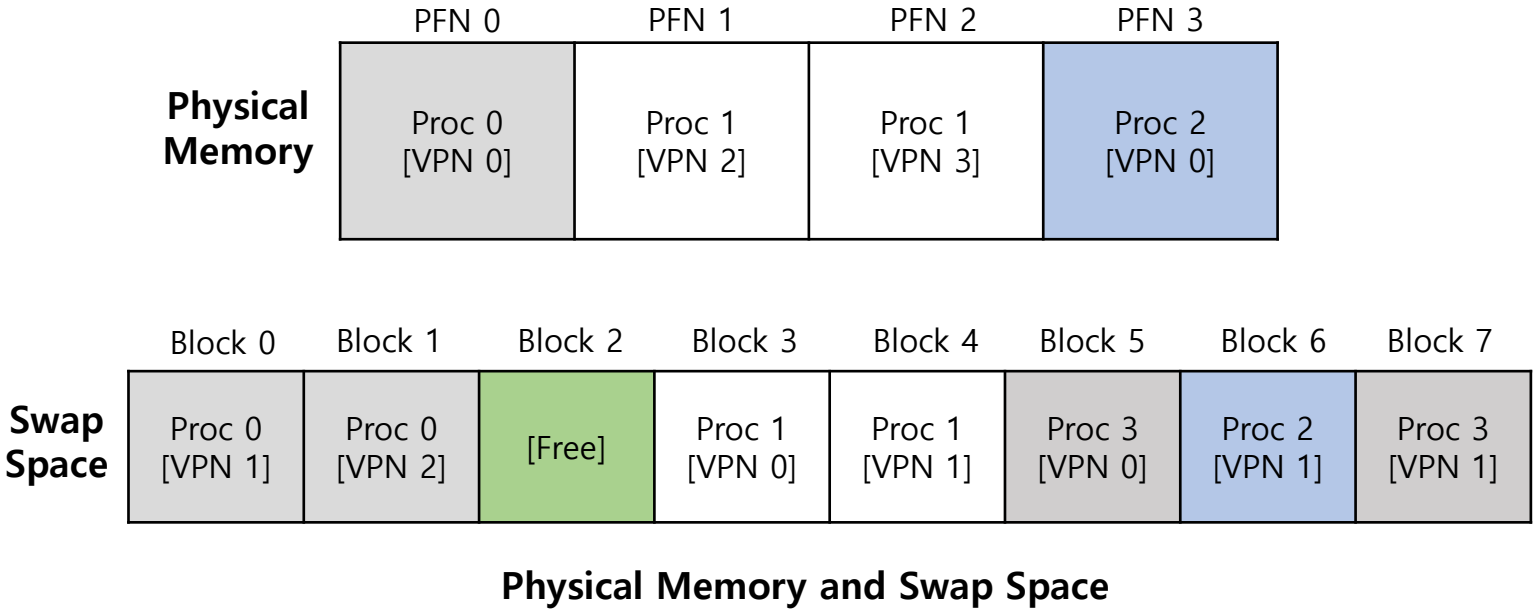
Memory Hierarchy in modern system

스왑 (Swap)

- 오버레이(Overlays)
  - 프로그래머가 코드 또는 데이터의 일부를 메모리에 적재할 지 여부를 판단함
  - 운영체제의 지원 불필요
- 프로세스 단위 스왑(Process-level swapping)
  - 프로세스에서 사용하고 있는 메모리를 메모리 계층 구조 하위로 이동
  - 프로세스가 재 실행되는 경우, 다시 하위 메모리 계층에서 상위 계층으로 이동해야함
- 페이지 단위 스왑(Page-level swapping)
  - 일부 페이지를 하위 메모리 계층으로 이동 (swap-out)
  - 하위 메모리 계층의 페이지를 상위 메모리 계층으로 이동 (swap-in)

스왑은 어디에 저장되는가?

- 스왑 공간
  - HDD/SSD의 일부 공간을 스왑 공간으로 예약
  - 스왑 공간은 페이지 크기와 동일한 블록 크기를 가짐
  - 특정 파일 시스템 또는 파티션으로 분할될 수 있음



Physical Memory and Swap Space

Present Bit

- 페이지가 메모리 계층 구조의 어디에 존재하는지 확인이 필요함
  - 메모리 공간 vs. 스왑 공간
  - PTE의 present bit이 해당 페이지의 위치를 표시함

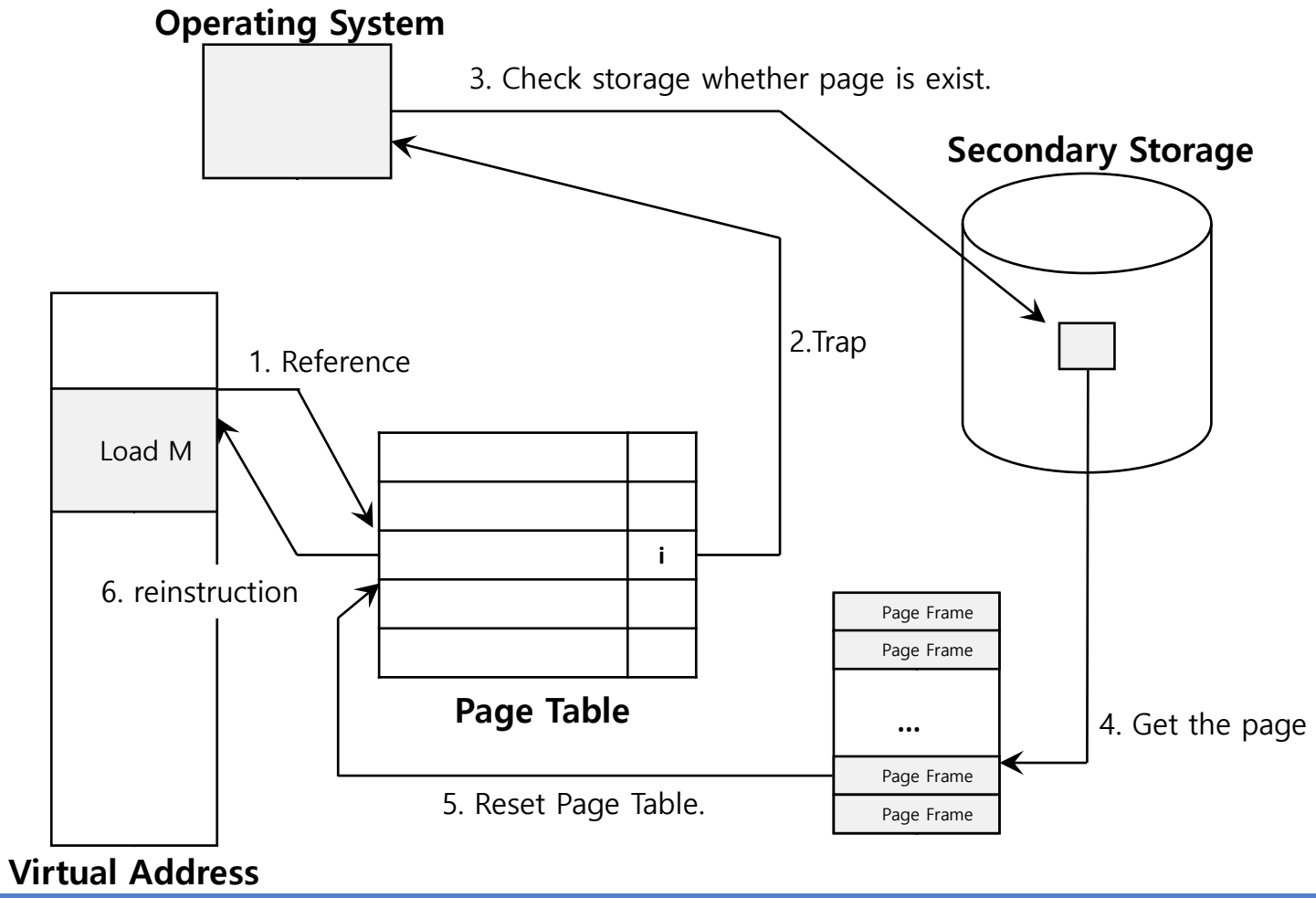
Value	Meaning
1	page is present in physical memory
0	The page is not in memory but rather on disk.

Page Fault

- 접근하는 페이지가 물리 메모리에 적재되지 않음
  - 스왑 공간에 존재하는 경우, 운영체제는 page fault을 발생시켜 해당 페이지를 메모리로 적재함
  - Page-fault handler 사용

Page Fault Control Flow

- PTE used for data such as the PFN of the page for a disk address.



메모리가 부족한 경우 (Full)?

- 운영체제는 현재 메모리에 적재된 페이지를 page out 하고 새로운 공간을 할당함
  - 페이지 교체 정책이 어떤 페이지를 page out할지 결정함

- Page out
  - 원래 파일 위치로 쓰기
  - 스왑 공간으로 쓰기
  - 특별한 읽기/쓰기 없음

Page Fault Control Flow – Hardware

```
1:      VPN = (VirtualAddress & VPN_MASK) >> SHIFT
2:      (Success, TlbEntry) = TLB_Lookup(VPN)
3:      if (Success == True) // TLB Hit
4:          if (CanAccess(TlbEntry.ProtectBits) == True)
5:              Offset = VirtualAddress & OFFSET_MASK
6:              PhysAddr = (TlbEntry.PFN << SHIFT) | Offset
7:              Register = AccessMemory(PhysAddr)
8:          else RaiseException(PROTECTION_FAULT)
```

Page Fault Control Flow – Hardware

```
9:      else // TLB Miss
10:     PTEAddr = PTBR + (VPN * sizeof(PTE))
11:     PTE = AccessMemory(PTEAddr)
12:     if (PTE.Valid == False)
13:         RaiseException(SEGMENTATION_FAULT)
14:     else
15:         if (CanAccess(PTE.ProtectBits) == False)
16:             RaiseException(PROTECTION_FAULT)
17:         else if (PTE.Present == True)
18:             // assuming hardware-managed TLB
19:             TLB_Insert(VPN, PTE.PFN, PTE.ProtectBits)
20:             RetryInstruction()
21:         else if (PTE.Present == False)
22:             RaiseException(PAGE_FAULT)
```

Page Fault Control Flow – Software

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

- 운영체제는 새로운 페이지를 위한 빈 물리 메모리 공간(페이지 프레임)을 찾아야 함
- 만약, 빈 물리 메모리 공간이 없다면, 페이지 교체 정책에 따라서 일부 페이지를 교체해야 함

교체는 실제 언제 일어나는가?

- 운영체제가 전체 메모리를 모두 사용하는 시점에서 페이지 교체를 수행함

- 스왑 데몬(Swap Daemon), 페이지 데몬(Page Daemon)
  - 남은 페이지의 개수가 최소값(LW: Low Watermark)에 도달하면 백그라운드 쓰레드가 동작하여 페이지를 page out 함
  - 해당 쓰레드는 최대값(HW: High Watermark)에 도달하면 자동으로 종료됨

스왑의 대상

- 어떤 타입의 페이지 프레임이 스왑되는가?
  - 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**

물리 메모리 크기의 극복: 정책

- 메모리 사용량의 증가는 사용되고 있는 페이지의 일부에 대한 page out을 유발함

- 어떤 페이지를 추출(evict)할지 결정하는 정책이 페이지 교체 정책임

캐시 관리 정책

- 캐시 미스(cache miss) 최소화가 목적임
- 평균 메모리 접근 시간으로 계산 가능
  - AMAT: Average Memory Access Time

$$AMAT = (P_{Hit} * T_M) + (P_{Miss} * T_D)$$

Arguement	Meaning
T_M	The cost of accessing memory
T_D	The cost of accessing disk
P_{Hit}	The probability of finding the data item in the cache(a hit)
P_{Miss}	The probability of not finding the data in the cache(a miss)

최적의 페이지 교체 정책

- 가장 적은 캐시 미스를 발생시킴
  - 가장 나중에 접근할 페이지를 교체 대상으로 선택함
  - 그 결과로 가장 적은 캐시 미스를 발생 시킴

- 비교하기 위한 지표로만 사용됨

Tracing the Optimal Policy

Reference Row

0 1 2 0 1 3 0 3 1 2 1

cold-start miss →

Access	Hit/Miss?	Evict	Resulting Cache State
0	Miss		0
1	Miss		0,1
2	Miss		0,1,2
0	Hit		0,1,2
1	Hit		0,1,2
3	Miss	2	0,1,3
0	Hit		0,1,3
3	Hit		0,1,3
1	Hit		0,1,3
2	Miss	3	0,1,2
1	Hit		0,1,2

Hit rate is $\frac{Hits}{Hits+Misses} = \mathbf{54.6\%}$

Future is not known.

A Simple Policy: FIFO

- 가장 먼저 들어온 페이지가 가장 먼저 추출(evict)됨
  - 교체가 필요할 때 가장 마지막에 존재(첫번째 입력된)하는 페이지 제거
  - 구현이 쉬움
  - 그러나, 페이지에 대한 중요성이 전혀 고려되지 않음

Tracing the FIFO Policy

Reference Row

0 1 2 0 1 3 0 3 1 2 1

Access	Hit/Miss?	Evict	Resulting Cache State
0	Miss		0
1	Miss		0,1
2	Miss		0,1,2
0	Hit		0,1,2
1	Hit		0,1,2
3	Miss	0	1,2,3
0	Miss	1	2,3,0
3	Hit		2,3,0
1	Miss		3,0,1
2	Miss	3	0,1,2
1	Hit		0,1,2

Hit rate is $\frac{Hits}{Hits+Misses} = \mathbf{36.4\%}$

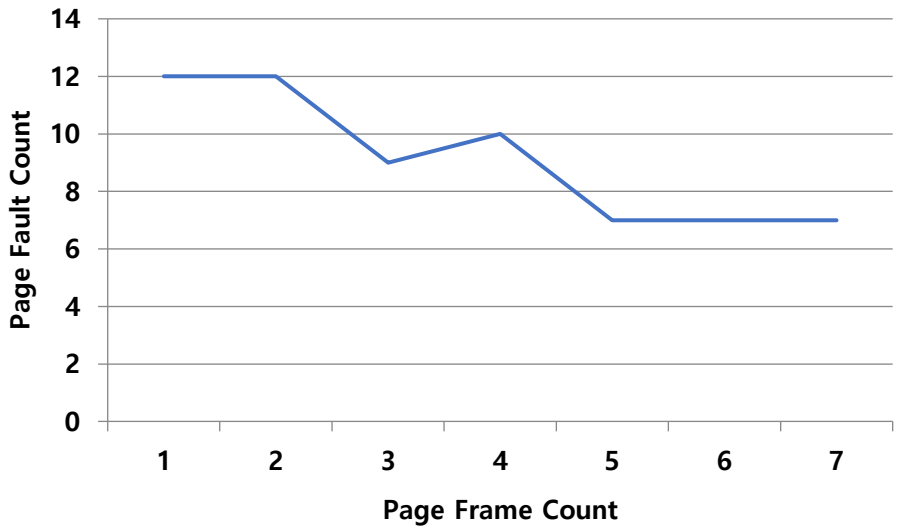
Even though page 0 had been accessed a number of times, FIFO still kicks it out.

BELADY'S ANOMALY

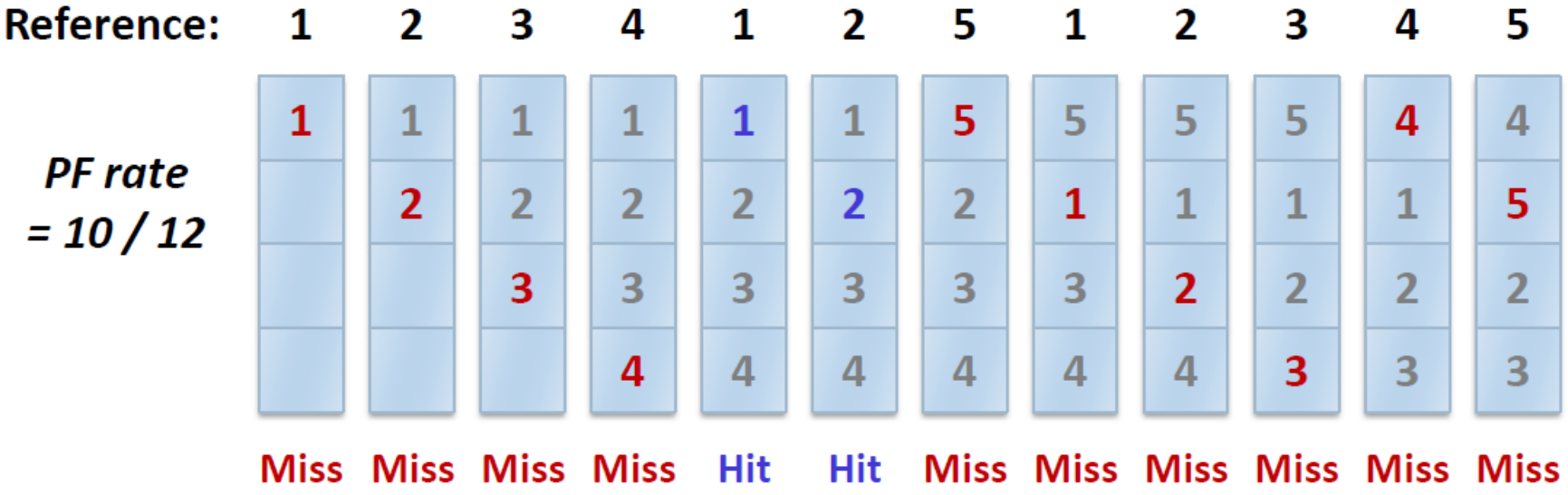
- 캐시의 양이 증가할 수록 일반적으로 캐시 적중률(cache hit)이 증가한다고 생각하지만 FIFO는 더 감소함

Reference Row

1 2 3 4 1 2 5 1 2 3 4 5



BELADY'S ANOMALY (Cont'd)



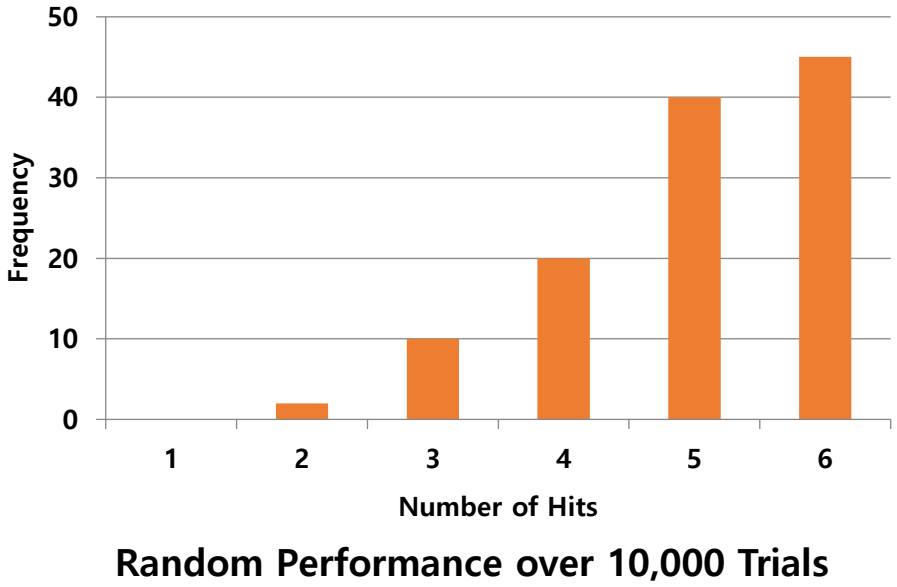
임의(Random) 선택

- 교체할 페이지를 임의(Random)하게 선택
  - 추출할 페이지를 선택하는 시간 소모를 제거함

Access	Hit/Miss?	Evict	Resulting Cache State
0	Miss		0
1	Miss		0,1
2	Miss		0,1,2
0	Hit		0,1,2
1	Hit		0,1,2
3	Miss	0	1,2,3
0	Miss	1	2,3,0
3	Hit		2,3,0
1	Miss	3	2,0,1
2	Hit		2,0,1
1	Hit		2,0,1

임의(Random) 선택 성능

- 임의 선택이 최적의 교체 정책과 비슷한 경우도 존재함



Q&A