운영체제

2019.7.11

컴퓨터공학과 이병문
가천대학교 Gachon University

강의일정

- 7/9 메모리관리2
- 10 가상메모리1(수행과제2)
- 11 가상메모리2
- 12 기말고사

페이지대치 알고리즘
- 선입선출, 벨레디의 변이
- OPT알고리즘(MIN)
- LRU, LRU근접알고리즘
- LFU, MFU 알고리즘

페이지할당 알고리즘
- 균일과 비례할당 알고리즘

적재정책
- 스레싱(Thrashing), 지역성,
- Working Set Model
- 페이지부재율

기타고려사항(메모리관리)
-대치범위, 프리페이징
-페이지크기, …

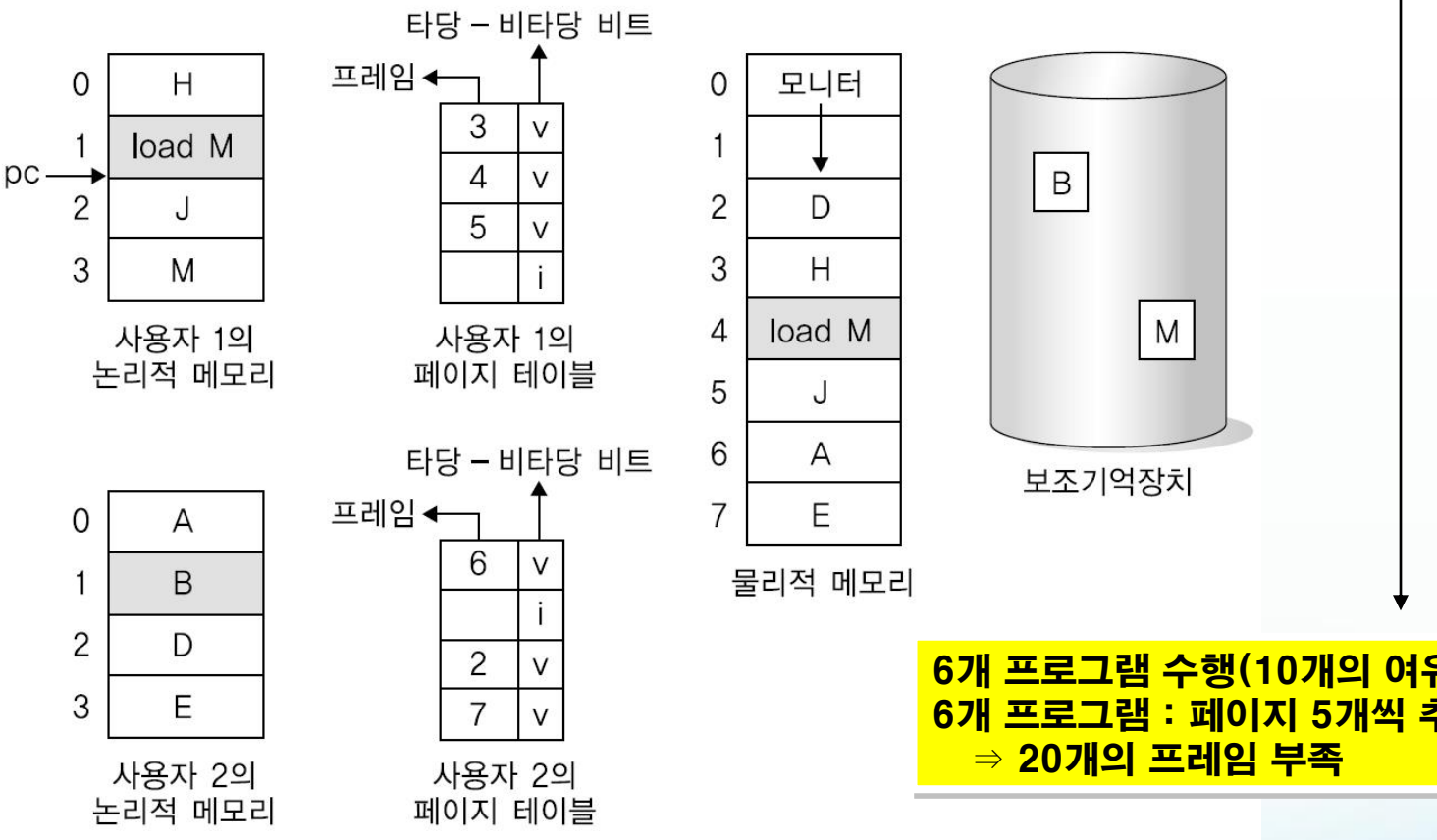
가상메모리 – 페이지대치

■ 페이징 대치의 필요성

☑ 40개의 프레임 시스템

-10개 프레임 요구 ⇒ 4개의 프로그램 수행

- 5개 프레임 요구 ⇒ 8개의 프로그램 수행 가능



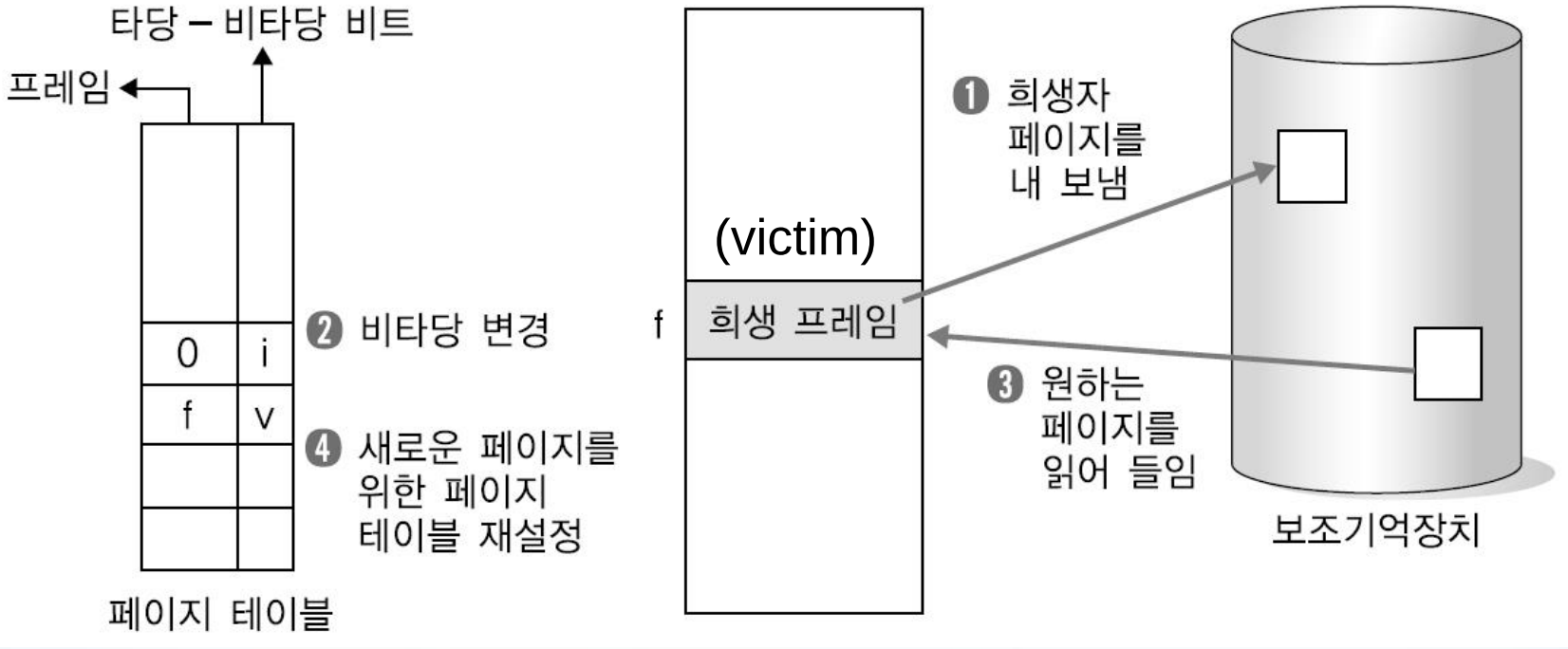
6개 프로그램 수행(10개의 여유)
6개 프로그램 : 페이지 5개씩 추가요구
⇒ 20개의 프레임 부족

가상메모리 – 페이지대치

■ 페이징 대치 (victim – 희생프레임을 선택하는 방법)

☑ 해결방법 : 페이지 대치

☑ 페이지 부재 발생하면 메인메모리에 있으면서 사용되지 않는 페이지를 없애고 새로운 페이지로 바꾸는 행위



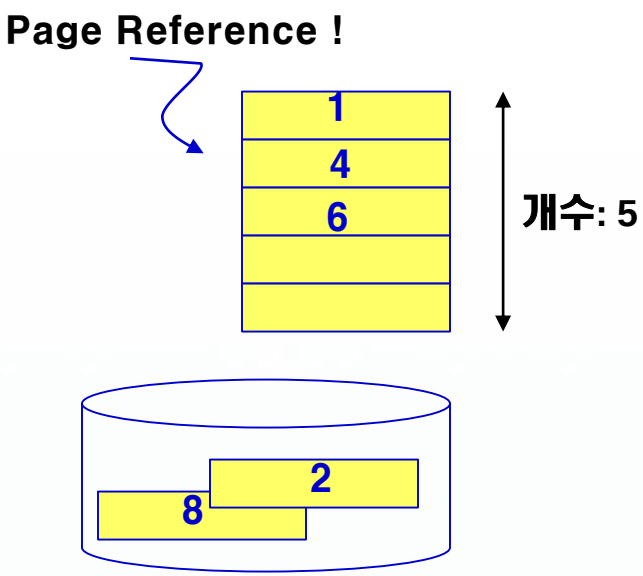
☑ 희생프레임(victim)을 찾는 것은 매우 중요 -> 효율적인 페이지 대치 알고리즘이 필요

가상메모리 – 페이지대치

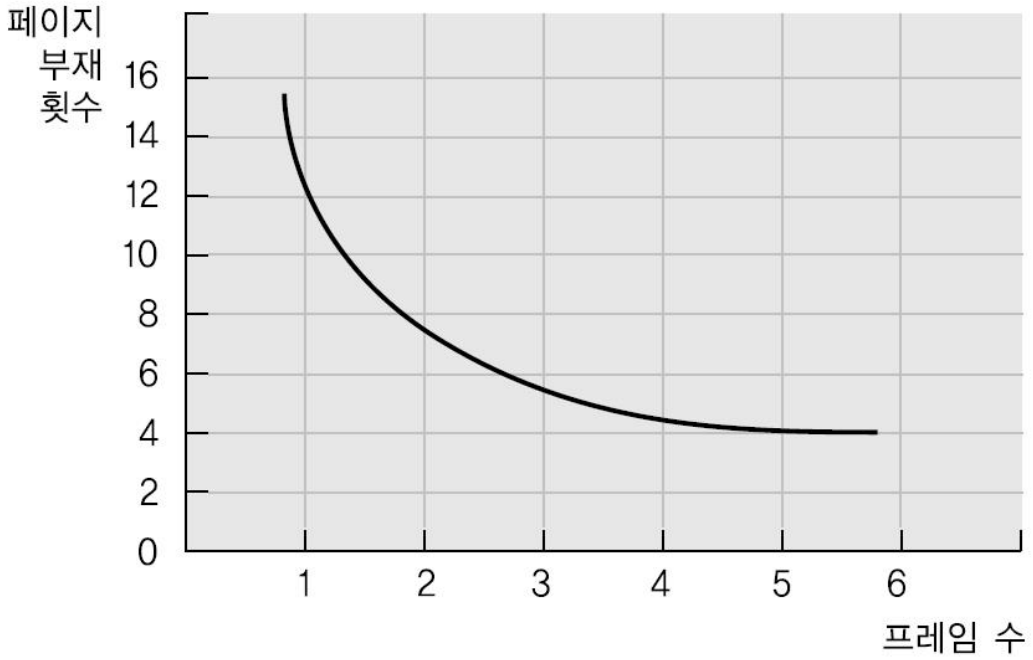
■ 페이지 부재(Page Fault)와 프레임 개수

☑ (페이지) 참조문자열 : **1, 4, 1, 6, 1, 6, 1, 6, 1, 6, 1, 6, 1**

☑ 프레임 수 증가 -> 페이지 부재수 감소



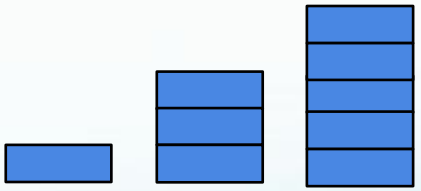
페이지 부재와 프레임 개수



☑ 프레임 개수 5개일 경우 페이지부재수는 ?

☑ 프레임개수 3개일 경우 페이지 부재수는 ?

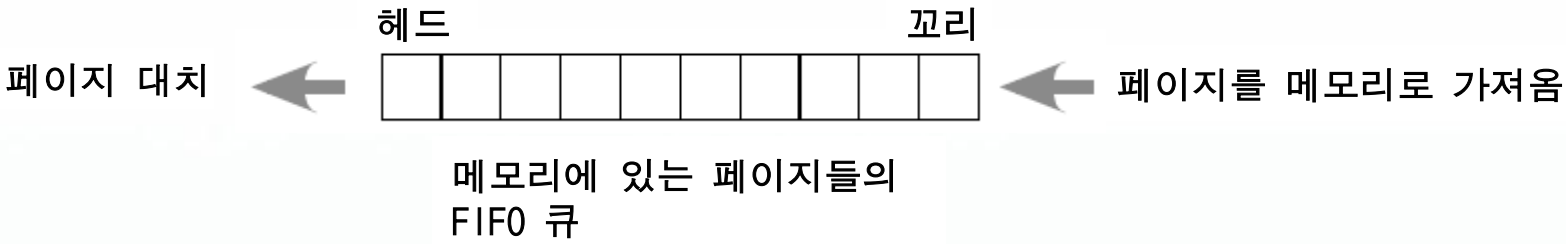
☑ 프레임개수 1개일 경우 페이지부재수는 ?

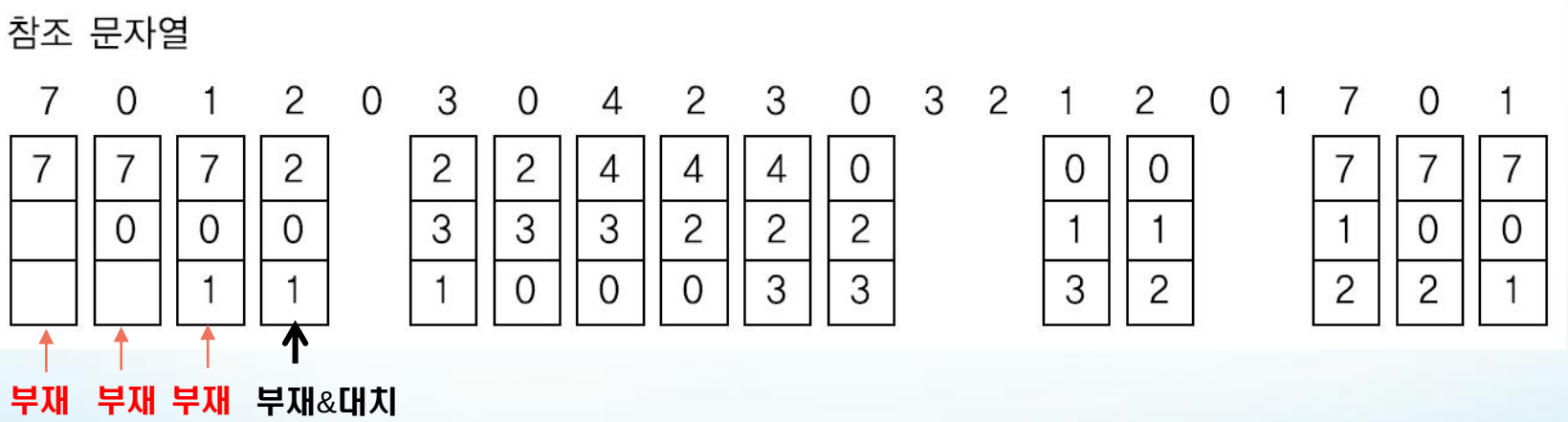


가상메모리 – 페이지 대치알고리즘

■ 선입선출(FIFO, FCFS) 대치 알고리즘

- ☑ 참조문자열 : **7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1**
- ☑ 간단한 페이지 대치 알고리즘
- ☑ 페이지의 **메모리 도착시간** 이용 - **오래된 페이지 대치**
- ☑ 페이지 : 선입선출 큐(**FIFO queue**)에 의해 관리
  큐의 머리 부분에 있는 페이지를 먼저 대치





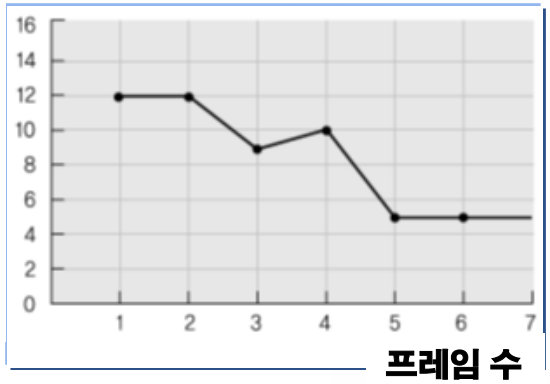
가상메모리 – 페이지 대치알고리즘

■ 벨레디의 변이(Belady's anomaly)

☑ 할당되는 프레임의 수가 증가함에 따라 페이지 부재율 증가
(Frame을 더 많이 할당했으나 Page fault가 오히려 증가)

☑ 최적 페이지 대치 알고리즘이 필요 함.

☑ **(페이지) 참조문자열: 0,1,2,3,0,1,4,0,1,2,3,4**



모든 페이지 프레임은 초기에 비어 있음

	0	1	2	3	0	1	4	0	1	2	3	4
3페이지 프레임	0	1	2	3	0	1	4	4	4	2	3	3
		0	1	2	3	0	1	1	1	4	2	2
			0	1	2	3	0	0	0	1	4	4
	P	P	P	P	P	P	P			P	P	

← 페이지 부재 표시(9회)

	0	1	2	3	0	1	4	0	1	2	3	4
4페이지 프레임	0	1	2	3	3	3	4	0	1	2	3	4
		0	1	2	2	2	3	4	0	1	2	3
			0	1	1	1	2	3	4	0	1	2
				0	0	0	1	2	3	4	0	1
	P	P	P	P			P	P	P	P	P	P

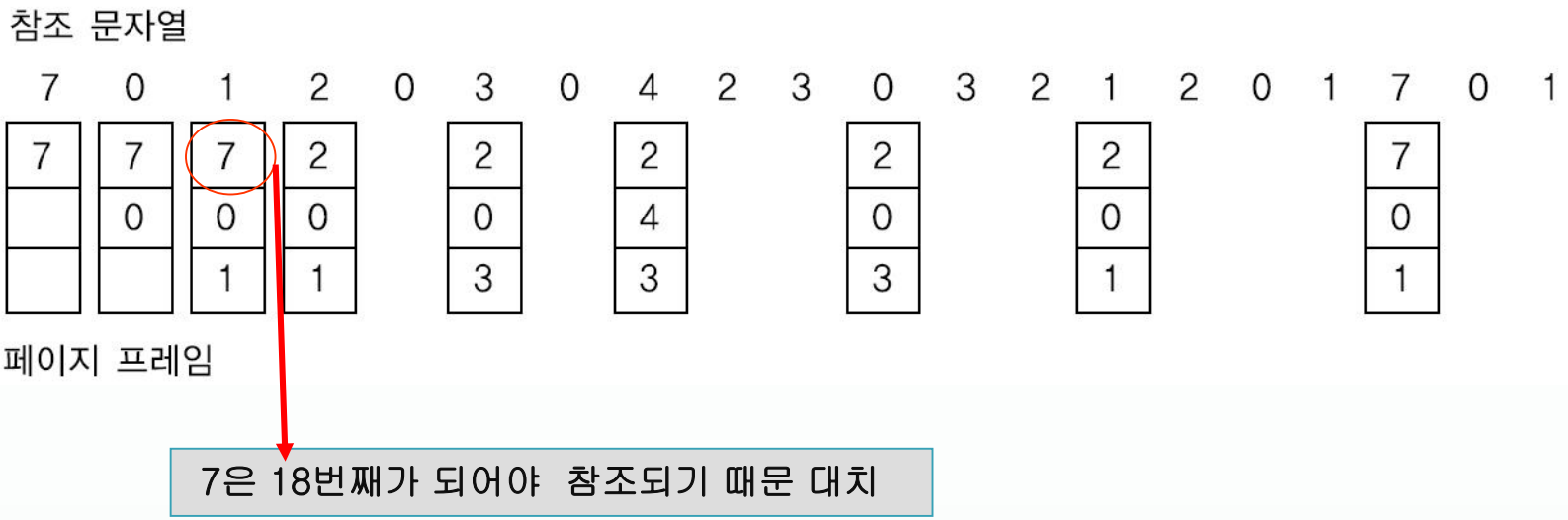
← 페이지 부재 표지(10회)

가상메모리 – 페이지 대치알고리즘

■ 최적 페이지 대치 알고리즘

☑ 페이지 부재율이 가장 낮음 : **OPT(Optimal Algorithm)** 또는 **MIN** 알고리즘
"앞으로 가장 오랜 기간 동안 사용되지 않을 페이지로 대치하라"

☑ 페이지 부재 : **9**회



☑ 구현의 어려움
참조열에 대한 미래의 지식(정보)요구 – 미리 알기 어렵기 때문
최소작업우선(SJF)중앙 처리 장치 스케줄링과 비슷한 경우

가상메모리 – 페이지 대치알고리즘

■ 최근 최소사용 (LRU, Least Recently Used) 대치 알고리즘

- ☑ 가까운 미래의 근사치로서 가장 최근의 과거 사용
  (오랜 기간 동안 사용되지 않은 페이지로 대치하는 효과)
- ☑ 미래의 예측을 과거의 자료로 해결하려는 통계적 개념
- ☑ 메모리의 지역성 이용한 알고리즘
- ☑ 각 페이지와 그 페이지가 최후로 사용된 시간 연관

페이지 부재 :12회

☑ 페이지가 대치되어야 할 때는 “오랫동안 사용하지 않은 페이지 선택”

가상메모리 – 페이지 대치알고리즘

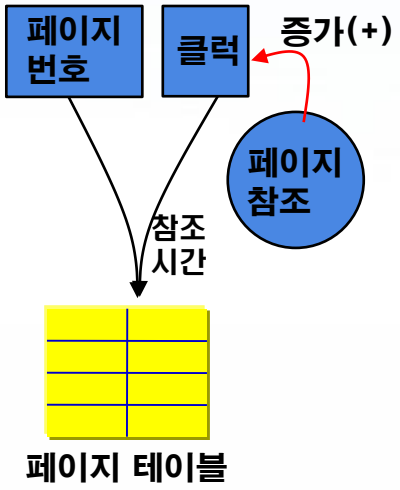
■ LRU**구현방법1**: **계수기**(Counter)

☑ 각 페이지 테이블 항목과 사용시간 레지스터 연관
프로세서에 논리 클럭이나 계수기를 덧붙임

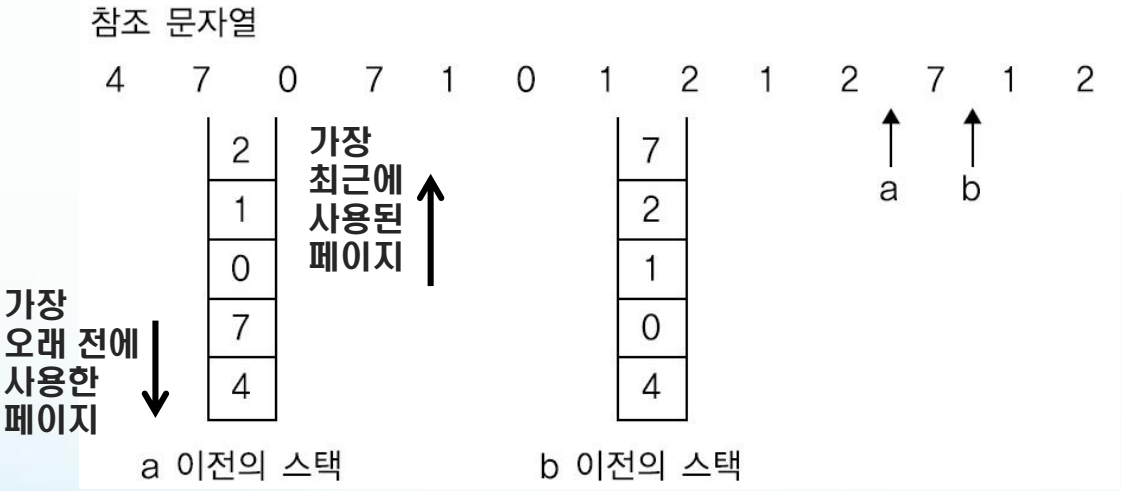
☑ 클럭 - 메모리 참조마다 증가

☑ 사용시간 레지스터 - 페이지의 최후 참조에 대한 시간 유지
가장 작은 시간 값을 가진 페이지 대치

☑ 페이지 탐색시간과 페이지 테이블의 변화 과정 유지 부담



■ LRU**구현방법2**: **스택**(Stack)



최근의 페이지를 참조한 스택

☑ 참조될 때마다 **Top**으로 추가 또는 이동함.

☑ 중간에서 이동할 경우는 오버헤드 -> **Doubly Linked List Stack**으로

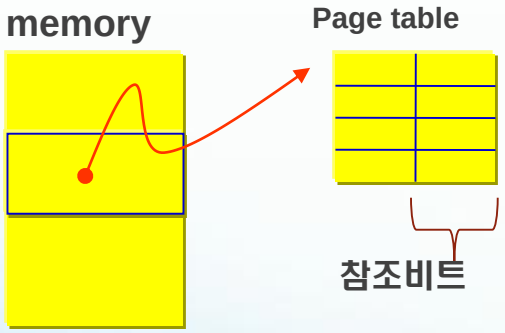
가상메모리 – 페이지 대치알고리즘

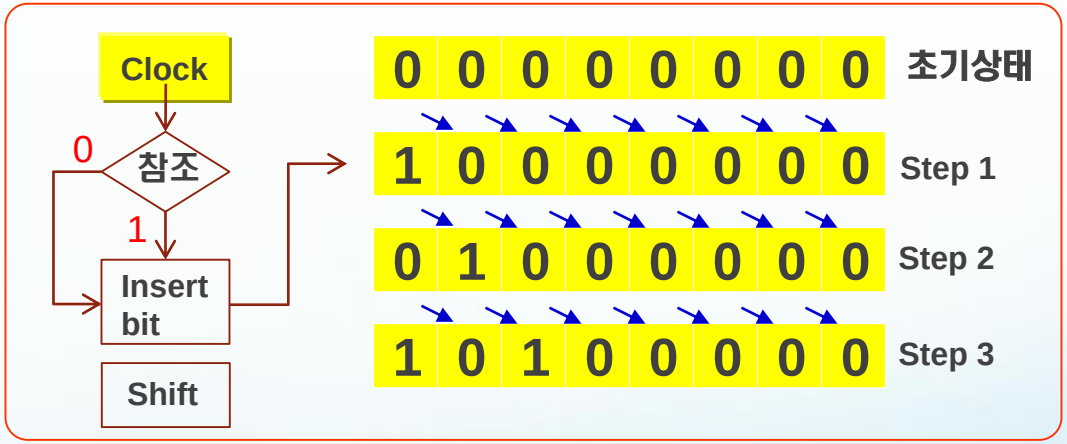
■ LRU근접 대치 알고리즘

- ☑ 부가된 참조비트(Reference bits) 알고리즘
- ☑ 시계(2차적 기회대치) 알고리즘
- ☑ 최소사용빈도(LFU, Least Frequently Used) 알고리즘
- ☑ 최대사용빈도수(MFU, Most Frequently Used) 알고리즘
- ☑ 페이지 버퍼링

■ LRU근접 구현: 부가된 참조비트 알고리즘

- ☑ 페이지 대치 대상 : 가장 낮은 참조 비트를 가진 페이지프레임
- ☑ **11111110 vs 00000001 vs 00001111 vs 01010110**

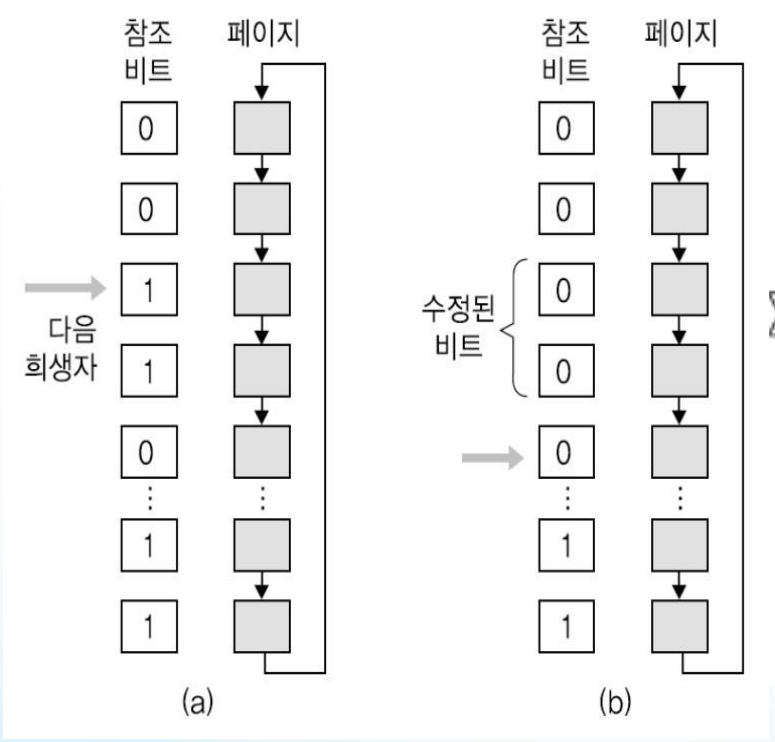


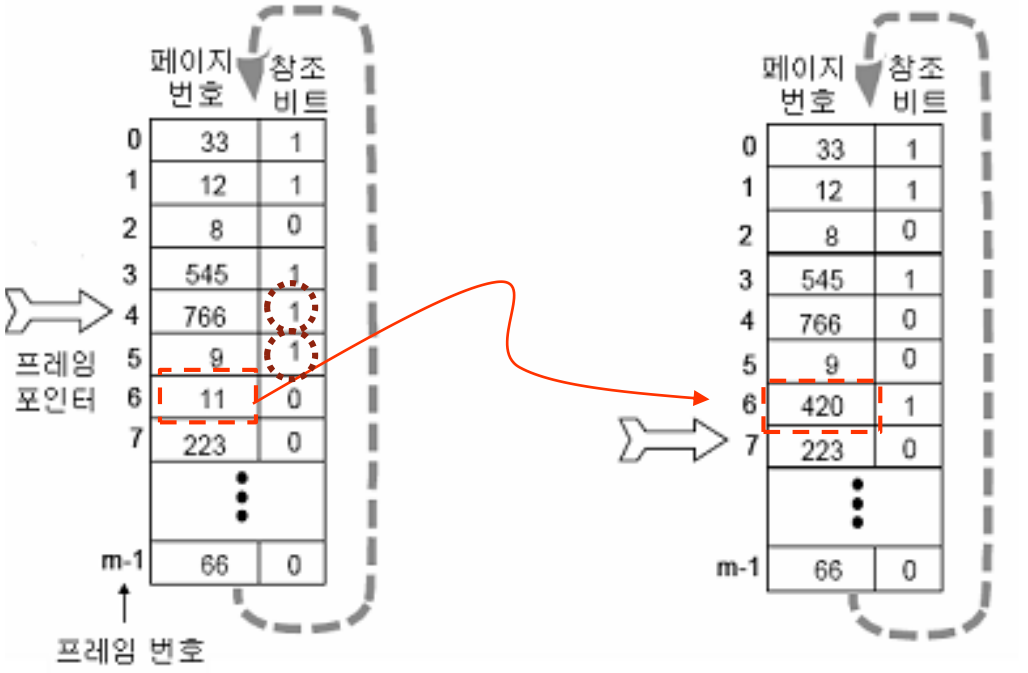


가상메모리 – 페이지 대치알고리즘

■ **LRU근접 구현**: **시계(2차적 기회대치, 2nd-Chance)알고리즘**

- ☑ 원형버퍼로 구성, **FIFO** 스타일
- ☑ 참조비트 ; 페이지가 프레임에 적재(1) 또는 페이지가 참조(1)
  초기값 (0) 또는 페이지대치 필요 시 **2차적 기회제공** (1 -> 0)
- ☑ 페이지 대치대상 ; 참조비트가 0 인 프레임의 페이지



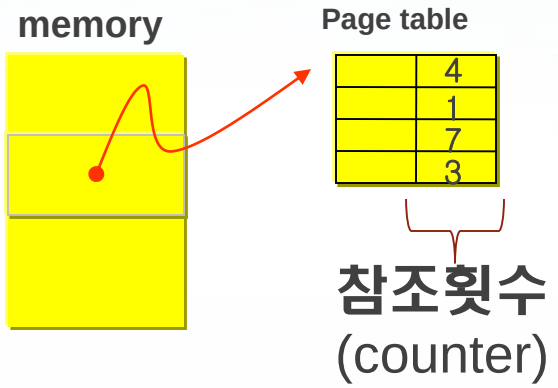


가상메모리 – 페이지 대치알고리즘

■ 최소사용빈도수, LFU(Least Frequently Used)알고리즘

☑ 페이지 대치대상
: **참조횟수가 적은** 프레임의 페이지
(참조횟수가 적은 것은 활발하게
사용되지 않을 것이라는 판단으로..)

☑ 페이지가 참조될 때 마다 해당페이지의
참조횟수가 증가



■ 최대사용빈도수, MFU(Most Frequently Used)알고리즘

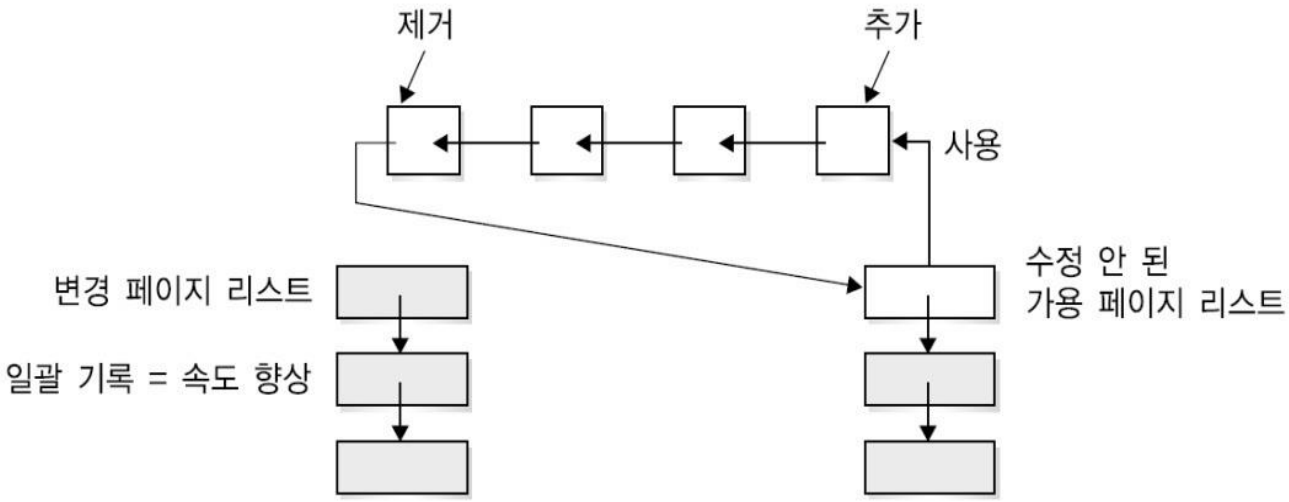
☑ 페이지 대치대상
: **참조횟수가 많은** 프레임의 페이지
(참조횟수가 적은 것은 향후에 사용될
가능성이 높다는 의미)

☑ 페이지가 참조될 때 마다 해당페이지의
참조횟수가 증가

가상메모리 – 페이지 대치알고리즘

■ **페이지 버퍼링**(Page Buffering)

☑ 교체대상으로 선택된 페이지를 잠시 동안 메모리에 유지(버퍼링)



☑ 변경되지 않은 페이지 교체 시,
교체된 페이지는 메모리에 남고 페이지 프레임은 가용 페이지 리스트의 끝에 추가됨.

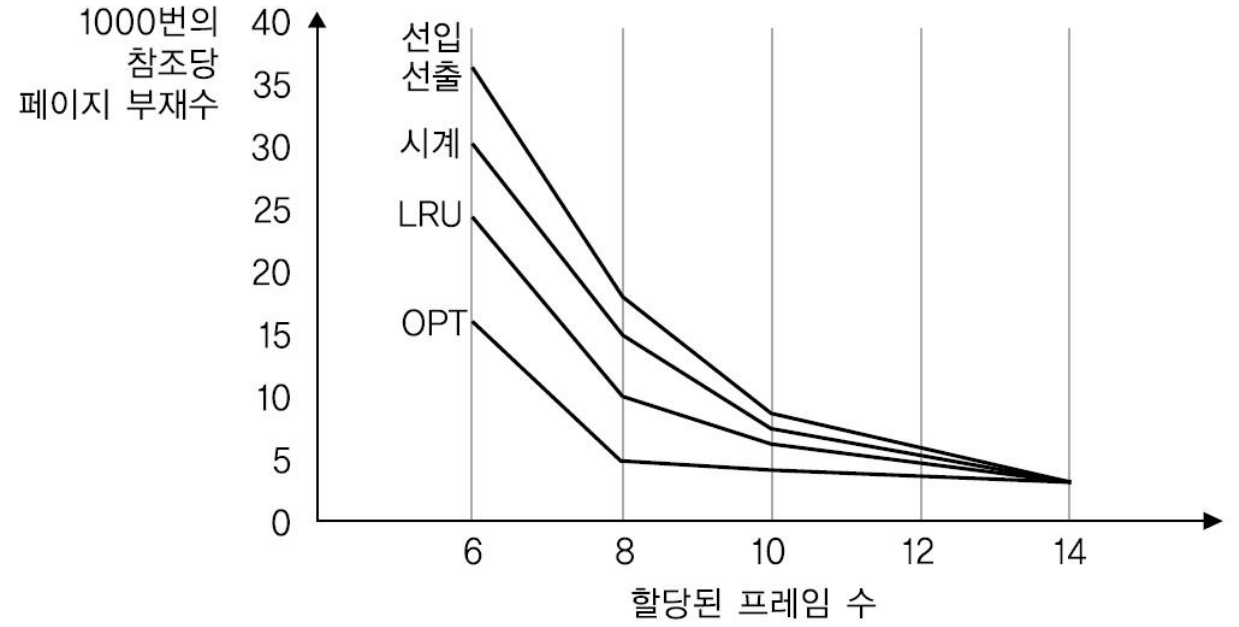
☑ 변경된 페이지 교체 시,
페이지 프레임을 변경 페이지 리스트의 끝에 추가함.

가상메모리 – 페이지 대치알고리즘

■ 알고리즘의 비교 및 분석

☑ 배르(BAER, 1980년대) 발표

- ☑ 분석에 사용한 페이지 크기는 256 word이며
- ☑ 프레임 수를 6, 8, 10, 12, 14개로 변경시키면서 실행.
- ☑ 적은 수의 프레임을 사용할 경우 차이가 크게 나타남.
- ☑ 페이지 부재율(Fault)이 낮을수록 좋음



가상메모리 – 페이지 할당알고리즘

■ 가상메모리의 효율성

☑ 페이지 부재율 감소를 위해 페이지대치와 프레임할당은 매우 중요

☑ n개 프레임중 에서 최적의 할당알고리즘으로 효율성을 증대

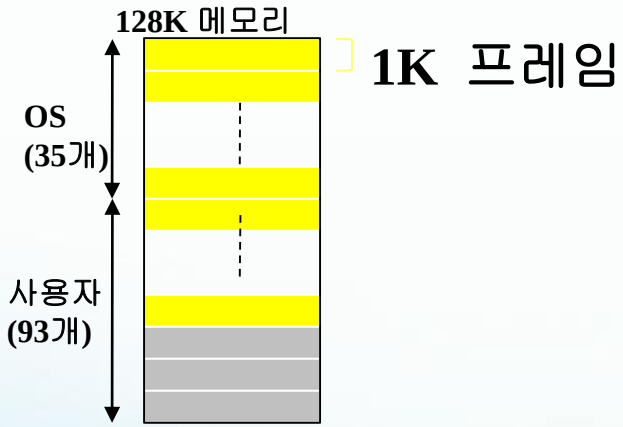
☑ 단독사용자 vs 다중사용자

■ 단독 사용자

☑ OS 영역(35개 프레임)과 단독사용자 영역(93개 프레임)으로 구분

☑ 1번째 프레임 할당 -> 2번째 프레임할당 -> … ->93번째 프레임할당

-> **94번째 프레임할당 요청 ➔ 페이지대치 필요**



■ 다중 사용자 (multi processes)

☑ 여러 프로세스에게 93개 프레임 할당을 어떻게 하여야 효율적으로 할 수 있는지 ?

☑ 각 프로세스마다 최소로 할당되어야 할 프레임수가 존재

☑ 프레임수 ▼ -> 페이지 부재율 ▲

가상메모리 – 페이지 할당알고리즘

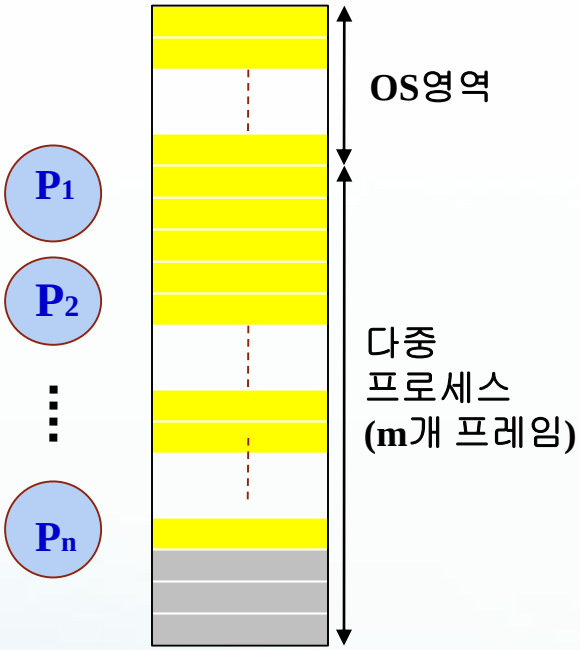
■ 균일과 비례할당 알고리즘

☑ 균일할당 ; 모든 프로세스에게 m/n 개 프레임 할당

☑ 비례할당 ; 각 프로세스의 크기에 비례하여 프레임할당

(각 프로세스가 요구하는 메모리 양이 다름)

큰 프로세스는 프레임 많이, 작은 프로세스는 프레임 적게



■ 예1)

☑ 3개 프로세스, 총 98개 프레임 경우

☑ 균일할당 ?

32 + 32 + 32 + 2

■ 예2)

☑ 2개 프로세스(10K, 127K), 총 62개 프레임 경우

☑ 균일할당 (31개 프레임, 31개 프레임)

☑ 비례할당 필요

10 / (10+127) x 62 프레임 = 약 4개 프레임

127 / (10+127) x 62 프레임 = 약 57개 프레임

가상메모리 – 적재정책(스레싱)

■ 스레싱(Thrashing)

☑ 페이지 부재 연속적 발생, 프로세스는 페이지 교환을 위하여 시간 낭비

⇒계속적으로 페이지 교환이 일어나는 현상

☑ **스레싱(thrashing) : 프로세스 수행 시간 < 페이지 교환에 보내는 시간**

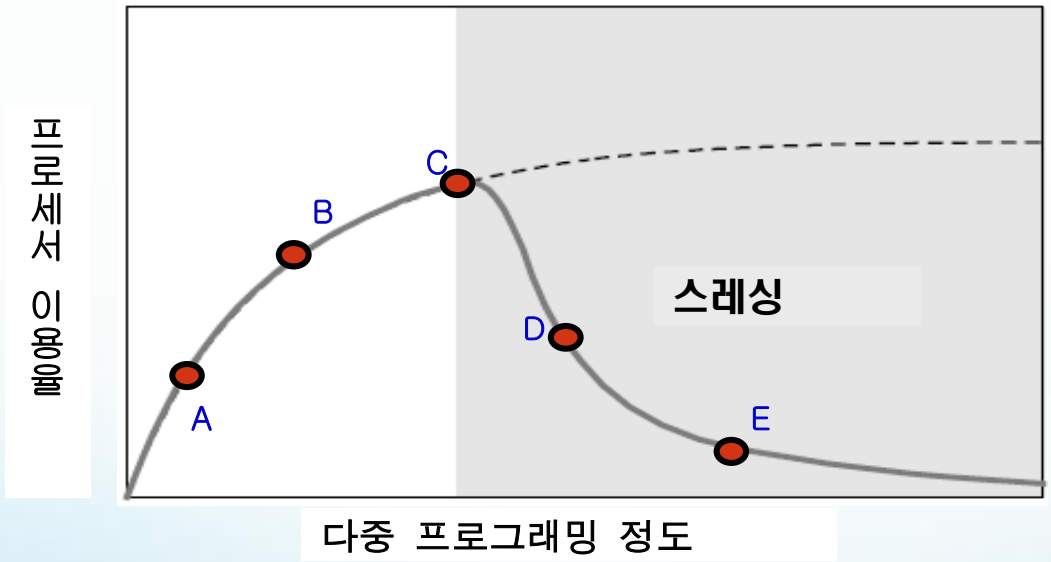
☑ 스레싱의 원인

– 프로세서의 효율성(이용율) 감시 운영

1) 이용율 감소 : 새로운 프로세스 도입(다중 프로그래밍 정도 높임)

2) 페이지 부재 일으킴(수행중인 프로세스로부터 페이지 회수)

3) 프로세스 : 필요한 프레임 미배당



다중 프로그래밍의 정도가 높아짐에 따라 프로세서의 이용율도 최대값이 될 때까지는 높아짐

가상메모리 – 적재정책(스레싱)

■ 스레싱(Thrashing)의 예방

- ☑ 지역 교환 알고리즘, 우선순위 교환 알고리즘 사용하면 제한
- ☑ 지역 교환 알고리즘 이용

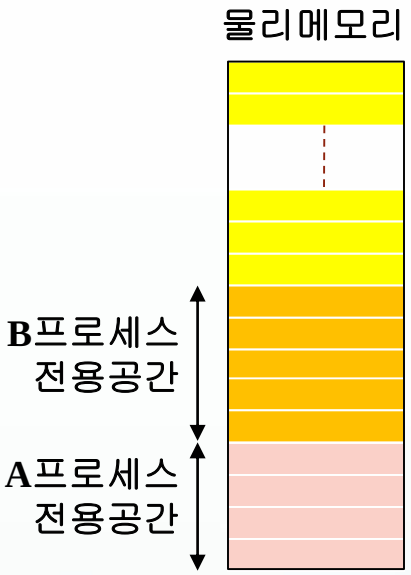
  스레싱 발생 ⇒ 다른 프로세스로부터 프레임 갖고 올 수 없음

  ⇒ 프로세스를 스레싱 현상에 빠지게 할 수 없음
- ☑ 여러 프로세스가 스레싱 일으키게 되면

  시간 : 페이징 처리 장치를 기다리는 큐에서 보내게 됨

  페이지 부재 처리를 위한 평균 시간 증가

  스레싱은 아니지만 유효 접근 시간 증가
- ☑ Denning은 50% 수준의 다중프로그래밍 정도 제안
- ☑ 예방 – 프로세스가 필요로 하는 프레임 수 제공

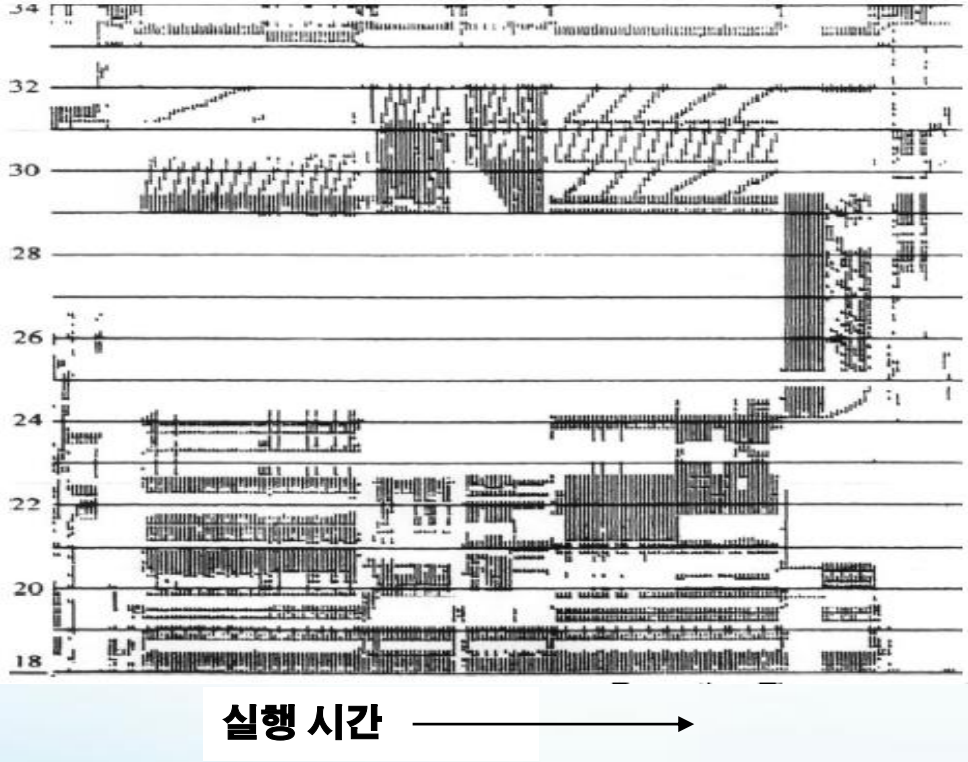


가상메모리 – 적재정책(Working Set)

■ 지역성(Locality)

☑ 프로세스 실행과정 ⇒ 지역성(locality) 또는 국부성 특성 가짐
메모리내의 정보를 균일하게 액세스 하지 않고
페이지중 일부를 선호하여 지역적인 부분을 집중적으로 참조

프로그램의 실행과정과 지역성의 연관관계



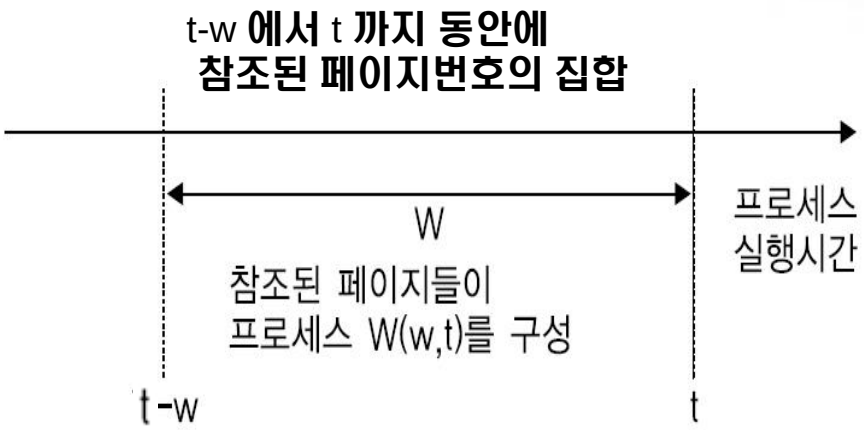
☑ 시간 지역성
참조된 기억장소가 가까운 미래에도
계속 참조될 가능성이 높음
[예] 순환구조의 루틴,
부프로그램, 스택,
계산이나 합계의 변수

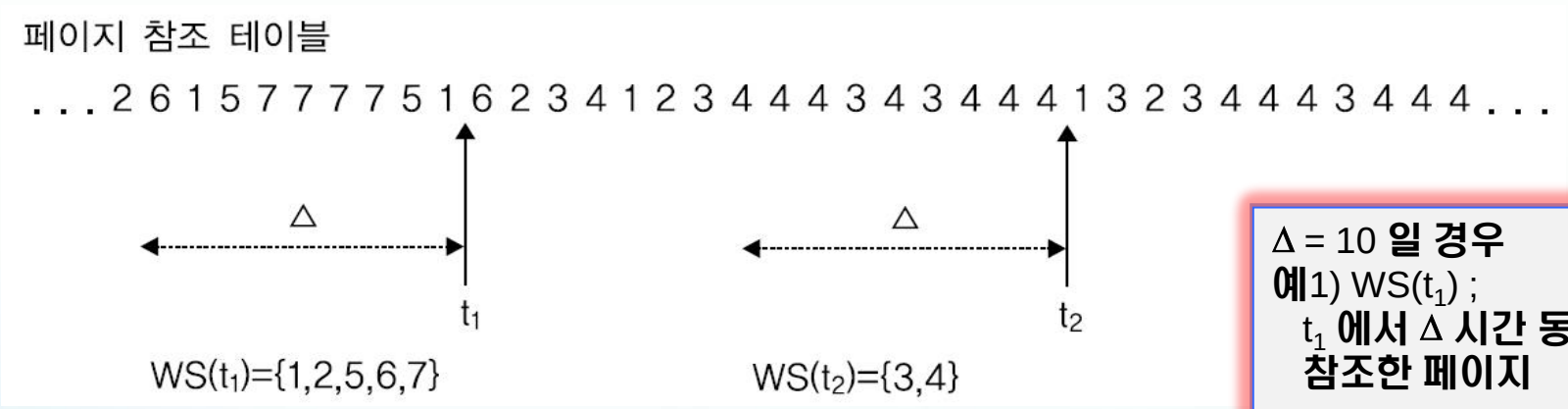
☑ 공간 지역성
프로세스가 어떤 기억 장소를
참조하면 그 근처의 기억장소가
그 후에도 계속 참조될 가능성이 높음
[예] 배열 순례(검색 작업), 순차적
코드의 실행, 근처의 관련 변수 선언

가상메모리 – 적재정책(Working Set)

■ 작업설정 모델(Working Set Model)

- ☑ Denning, 지역성(Locality) 을 설명하기 위한 목적으로 개발
- ☑ 참조가 많은 페이지들의 집합 -> "메모리에 지속적으로 상주"
- ☑ Page Fault 를 줄이는 것이 목적
- ☑ Working-Set(작업설정) ;
  -> Window 를 이용, $w(t, \Delta)$
  -> 시간 t 에서의 작업설정: W(t, w)





Δ = 10 일 경우
예1) $WS(t_1)$;
t_1 에서 Δ 시간 동안
참조한 페이지

가상메모리 – 적재정책(Working Set)

문제1) Δ=3 **일 경우에 다음의 메모리 참조순서에서의** Working Set **을 구하여라. 단, 총 프레임 갯수는** 3**이다.**

4 5 1 3 1 1 2 1 3 5 6 5 6 6 4 5 6 7

t_1 t_2

문제2) Δ=5 **일 경우에 위의 메모리 참조순서에서의** Working Set **을 구하여라. 단, 총 프레임 갯수는** 3**이다.**

가상메모리 – 적재정책(Working Set)

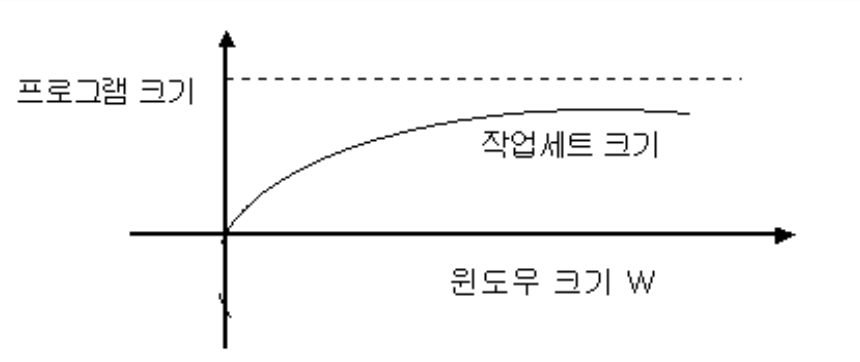
■ 작업설정 모델(Working Set Model)

- ☑ Δ = Working-Set Window , WSSi ; 프로세스i 의 Working-Set Size
- ☑ Δ 가 작게 설정된다면, 지역성을 커버하지 못함.
- ☑ Δ 가 크게 설정된다면, 몇 개 정도의 지역성을 커버함.
- ☑ Δ 가 매우 크게(무한대) 설정된다면, 전체프로세스를 커버
- ☑ $D = \sum WSS_i$ = Total Demand Frames, D 는 전체요구페이지프레임

프로세스가 5개의 프레임을 필요로 하는데

3개의 프레임만 있다면

- ☑ if D > m, Trashing 발생 , m :총 유효프레임수
  따라서, D > m 일 경우는 프로세스 실행을 연기시켜야 함.



적절한 값으로 10,000**으로 제안**
(Madnick**과** Donovan)

가상메모리 – 적재정책(Working Set)

■ **페이지 부재율**(page fault rate)

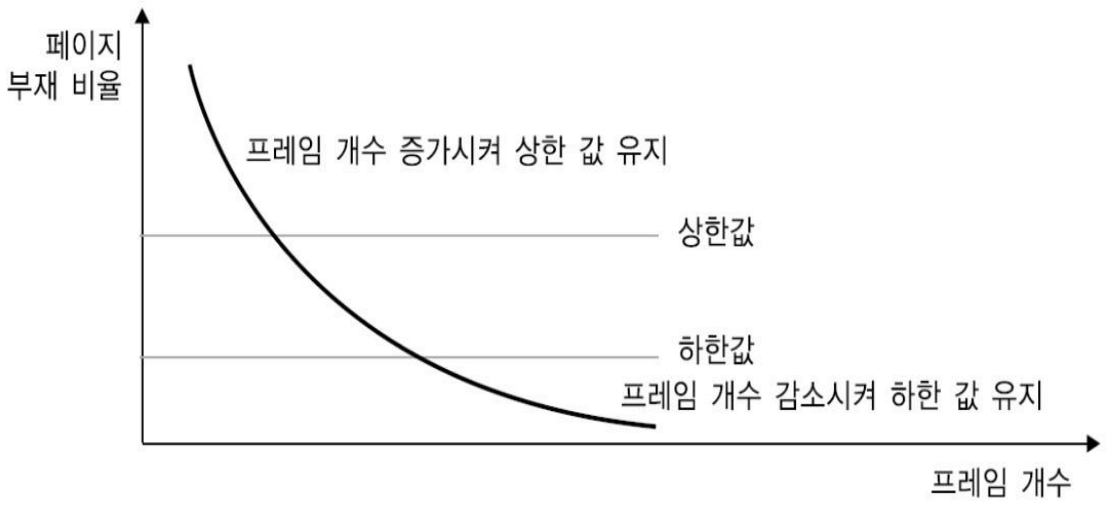
☑ 작업 설정
- 프리페이징(prepaging) 유용
- 스레싱 조절 어려움

☑ 페이지 부재 빈도(PFF;Page Fault Frequency); 스레싱 예방
- 직접적인 접근 방법
- 프로세스 중지시켜야 될 경우도 있음
  - 페이지 부재율이 증가되고 유효 프레임이 없으면
    어떤 프로세스를 골라서 그 프로세스를 중지시켜야 함
- [결과]
  프로세스에게 할당되었던 프레임은 높은 페이지 부재율을 갖는
  프로세스들에게 분배

☑ 스레싱 : 페이지 부재에서 발생
  페이지 부재율 조절 필요

가상메모리 – 적재정책(Working Set)

■ **페이지 부재율**(page fault rate)



페이지 부재율 높음
->프로세스가 더 많은
프레임 필요
페이지 부재율 낮음
-> 프로세스가 많은
프레임 갖고 있음
페이지 부재율의
상한과 하한 값 결정

☑ 페이지 부재율 > 상한 값 ⇒ 다른 프레임을 더 할당
☑ 페이지 부재율 < 하한 값 ⇒ 프레임 회수
☑ 스레싱 방지 ⇒ 페이지 부재율 측정, 조절

[문제점]

☑ 작업설정에서 보았던 일부 프로세스를 중지시키는 과정 발생 가능
☑ 페이지 참조가 새로운 지역성으로 이동하는 과도기
⇒ 작동 어려움, ⇒ 마지막으로 참조된 후 시간 단위(Δ) 경과 까지 그대로 있게 됨

기타고려사항

■ 대치범위

☑ 지역대치 vs 전역대치

☑ 지역대치

-해당 프로세스 프레임내 선택

-프로세스간 영향을 주지 않음

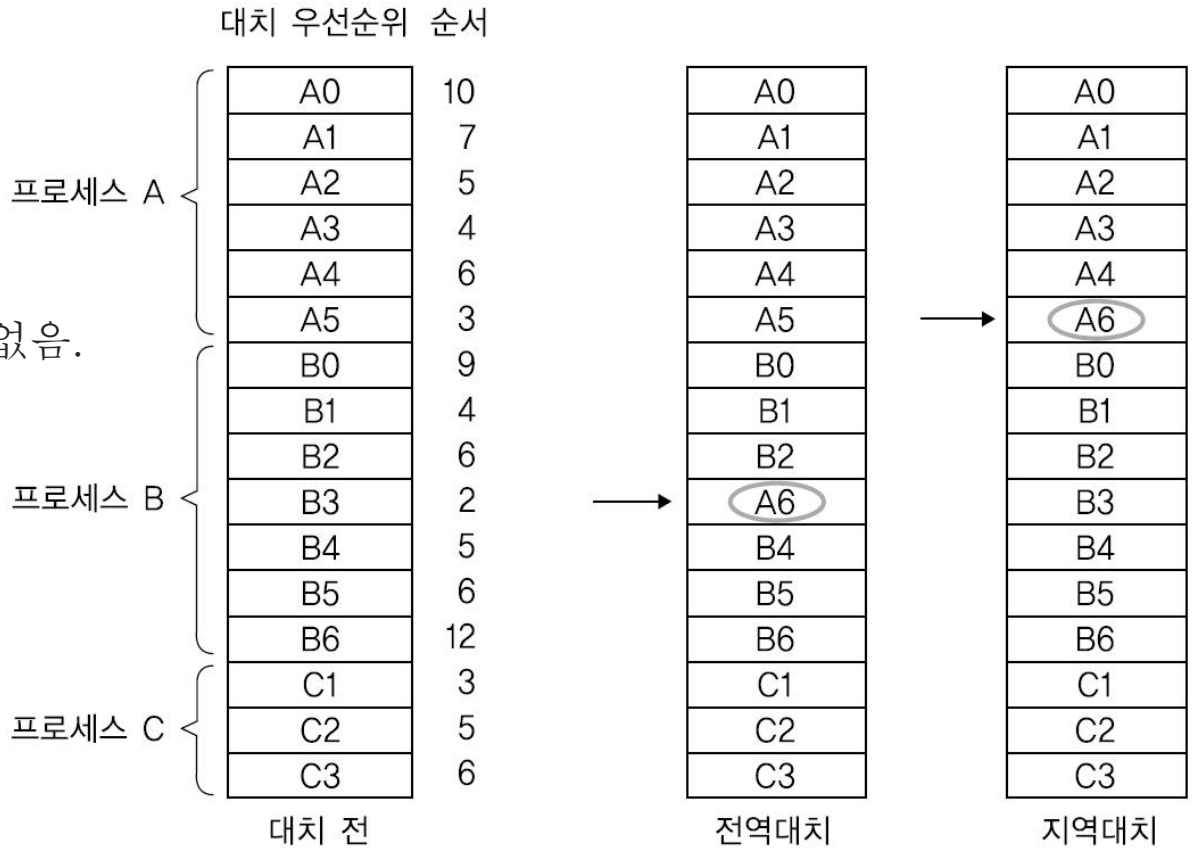
-프로세스의 프레임수는 변화없음.

☑ 전역대치

-프로세스와 상관없이 선택

-구현이 간단

-타 프로세스에 영향을 줌.



기타고려사항

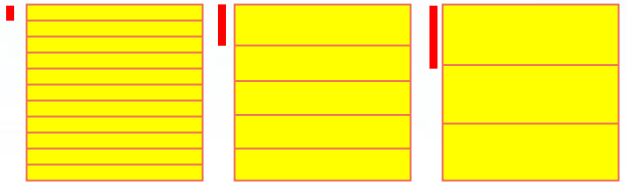
■ 프리페이징(Prepaging)

☑ 프로세스의 실행초기에는 많은 Page Fault 가 발생
☑ Prepaging ; 미리 여러 페이지를 가져온 뒤에 실행하면 Page Fault 를 줄여줌
☑ PrePaging 시점 ; 1) 프로그램 실행직전 2) I/O Waiting 3) 유효프레임이 없어 이를 처리하는 경우

☑ Prepaging 의 성능
☑ 메모리에 돌아온 페이지들 중에서 일부 페이지는 미사용 가능
예) s 페이지 만큼 Prepaging, 그러나 실제 사용율은 a ($0 \leq a \leq 1$).
불필요한 (1-a)s개의 페이지 프리페이징 비용
☑ as개의 페이지 부재 해결 비용 비교
a 가 0에 가까우면⇒ 프리페이징은 좋지 않음
1에 가까우면 ⇒ 프리페이징의 효과 좋음

기타고려사항

■ 페이지 크기(Page Size)



☑ 일반적으로 256Byte ~ 4096Byte

☑ Page 크기 ▲ -> Page Table 크기 ▼

예) 가상메모리 공간 ; 4M Byte이라면

Page Size(1024 byte) -> Page Table entry 개수는 ? 4096

Page Size(8192 byte) -> Page Table entry 개수는 ? 512

Architecture	Page Size
x86	4KB, 2MB, 4MB
X86_64	4KB, 1MB, 2MB
Power ISA	4KB, 64KB, 16MB
UltraSPARC	8KB, 64KB, 512KB,…
ARMv7	4KB, 64KB, 1MB

Page Size 를 크게 하면 ->

페이지 개수가 적어짐
페이지테이블 크기가 적어짐
페이지부재수가 적어짐
페이지교체가 줄어듦
처리속도 증가
디스크 처리시간 증가 페이지

단, 내부 단편화는 많아짐

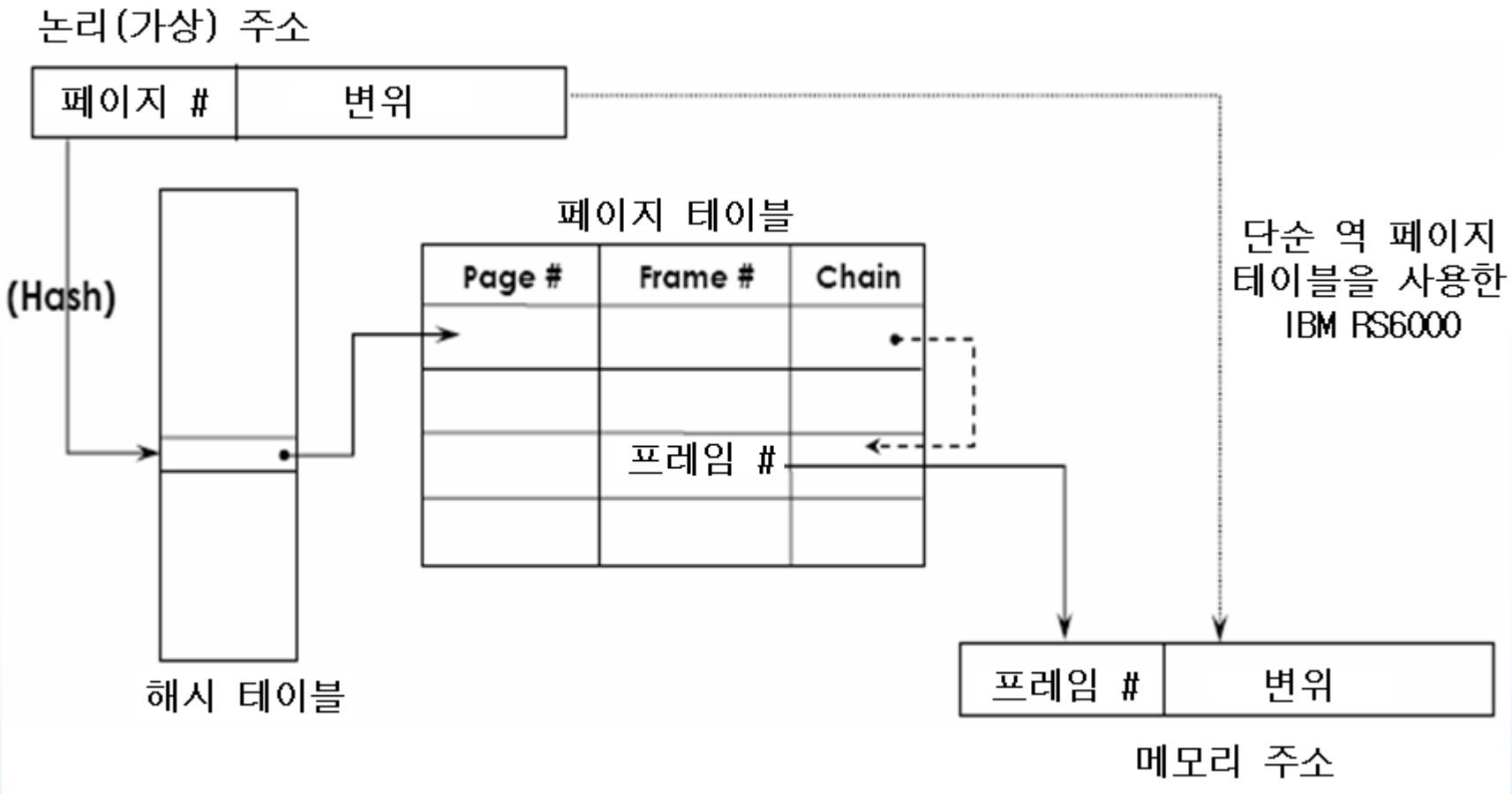
메모리 효율성이 떨어짐

기타고려사항

■ Page Table 구조

☑ 페이지 테이블이 커지면 -> 필요한 용량증가 -> 가상메모리에 저장

☑ 역 페이지 테이블 구조

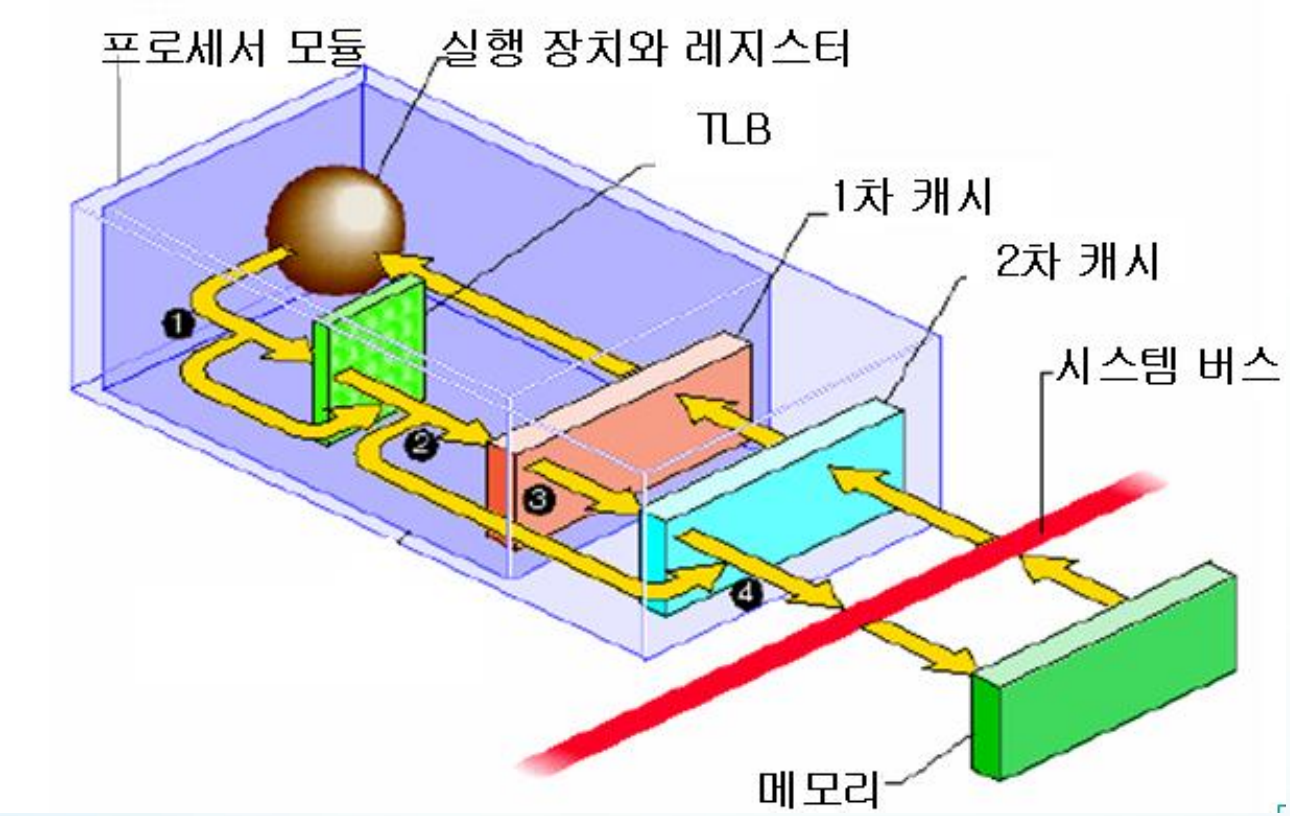


기타고려사항

■ Page Table 구조

☑ 변환 우선참조 버퍼(TLB, Translation Look-aside Buffer)

- 가상메모리 참조를 위한 메모리접근시간이 두배로 증가하는 것을 해결
- 페이지테이블 (메모리 -> TLB) ; 메모리접근시간 단축!



Q&A