Chapter 8.
Virtual Memory

8장의 강의 목표

- 가상메모리(virtual memory)의 필요성과 개념을 이해한다.
- 하드웨어와 소프트웨어의 상호작용 수준에서 가상메모리의 작동원리를 이해한다.
- 페이징(paging)과 세그먼테이션(segmentation)을 비교·이해한다.
- 가상메모리의 효율성을 높이기 위한 운영체제의 기능을 이해한다.
- Unix, Solaris, Linux, Windows 7의 메모리 관리 기법을 이해한다.

목 차

가상메모리의 특성

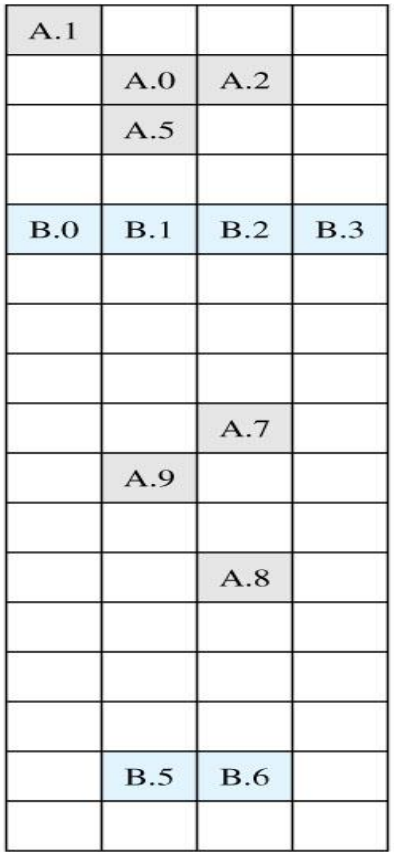
- 프로세스의 모든 메모리 참조는 논리 주소 (logical address)이며, 이는 동적으로 물리주소 (physical address)로 변환된다.
- 프로세스의 주소공간이 여러 블록(페이지나 세그먼트)으로 분할되어 그들 간의 순서/인접관계와 무관하게 주기억장치 상에 배치되어 수행될 수 있다.
- 프로세스의 주소공간을 구성하는 블록 중 일부만 주기억장치 상에 적재한 채 수행할 수 있다.
  - ✓ 적재집합(resident set): 특정 프로세스의 주소공간 중 주기억장치에 적재된 블록들의 집합
  - ✓ 메모리접근 오류(memory access fault) : 주기억장치에 적재되지 않은 블록이 참조될 때 발생하는 하드웨어 이벤트로, 발생시 해당 프로세스는 블록 상태로 참조할 블록이 디스크로부터 적재될 때까지 대기한 후 다시 준비 상태가 된다.

가상메모리와 물리메모리

- 프로세스의 전체 주소공간은 디스크에 설정되고, 그 중 일부분이 주기억장치에 적재(캐시)되어 CPU에 의해 참조된다.
  - ✓ 디스크의 방대한 저장 공간과 연계된 가상메모리
  - ✓ CPU가 실제로 참조할 부분이 적재되는 물리메모리(실메모리)

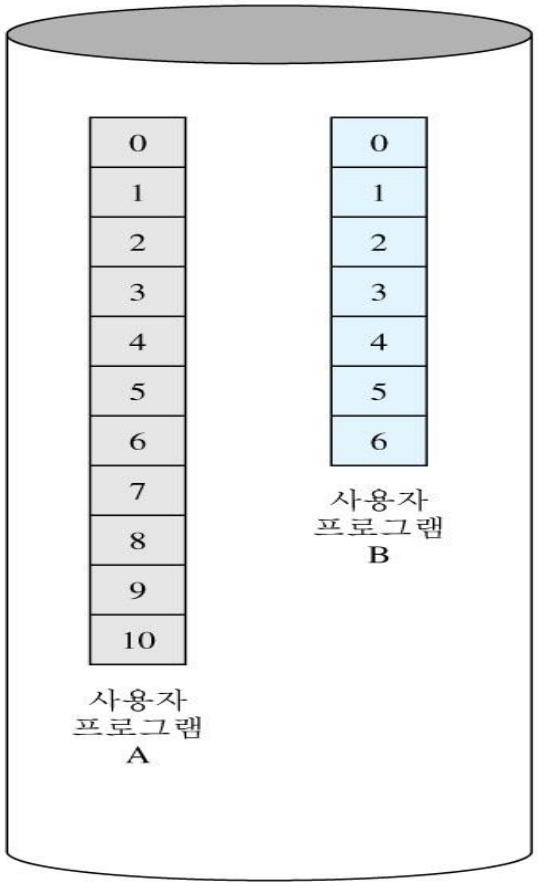
- 부분적재 수행의 이점
  - ✓ 보다 많은 프로세스를 주기억장치에 유지할 수 있다
    - 준비 상태 프로세스가 존재할 가능성이 높아져 처리기 활용도 높아짐
  - ✓ 주기억장치보다 큰 프로세스를 수행할 수 있다.
    - 오버레이 기법을 적용하기 위한 프로그래머의 부담(가용 메모리 크기 파악, 주소공간의 블록 분할, 언제 어떤 블록이 필요한지 파악, 주기억장치와 디스크 간의 블록 교체 작업 등)을 운영체제가 담당

가상 메모리



주기억장치

주기억장치는 페이지와 동일한
크기의 고정길이 프레임들로 구성된다.
프로그램 수행을 위해,
그 페이지들 중 일부 혹은 전부가
주기억장치에 있어야 한다.

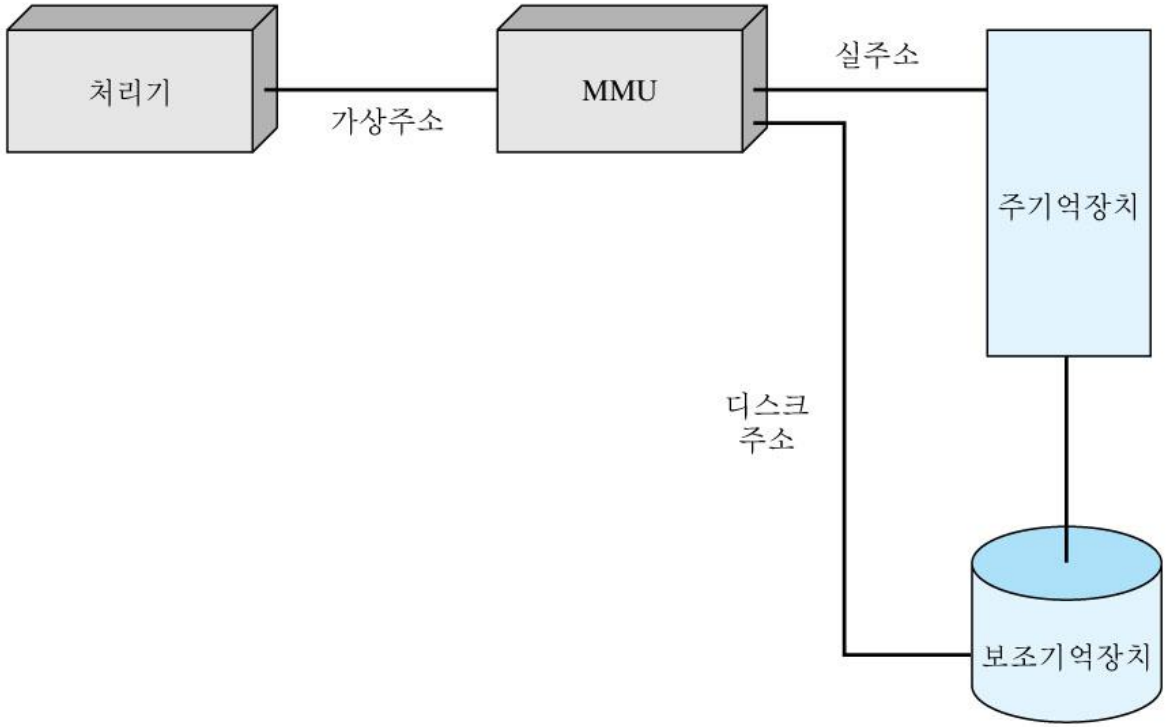


디스크

보조기억장치(디스크)는 고정길이
페이지들을 보유한다.
사용자 프로그램은 몇 개의
페이지들로 구성된다.
모든 프로그램과 운영체제의 페이지들은
파일들처럼 디스크에 있다.

가상 메모리 주소지정

프로세스는 단지 주기억장치에서 수행될 수 있기 때문에, 주기억장치는 실기억장치(real memory) 라고 함.



가상메모리 관련 용어

표 8.1 | 가상 메모리 관련 용어

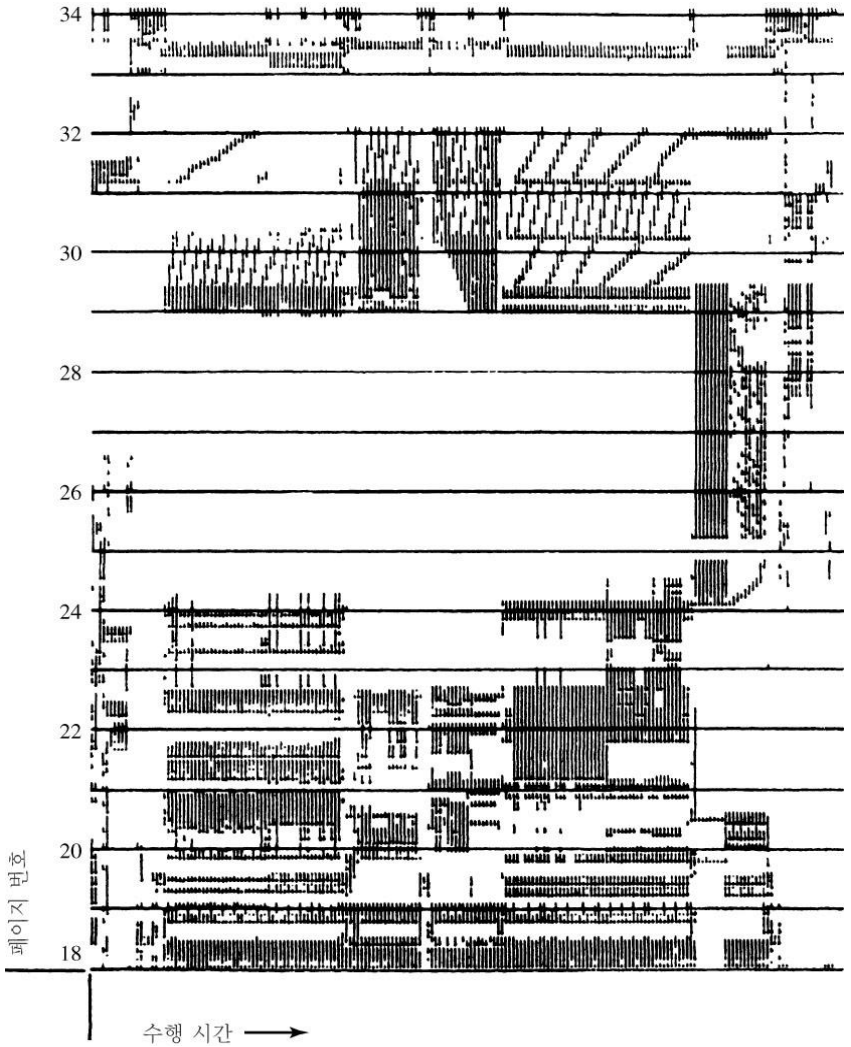
가상 메모리 (virtual memory)	보조기억장치를 주기억장치처럼 주소지정 가능하게 만든 저장공간 할당 체제. 프로그램이 메모리를 참조할 때 사용하는 주소는 메모리 시스템이 물리 메모리의 특정 위치를 식별할 때 사용하는 주소와 구별된다. 프로그램이 생성한 주소는 그에 대응된 물리 주소로 자동 변환된다. 가상 메모리의 크기는 주기억장치의 크기가 아니라 컴퓨터 시스템의 주소지정 체제와 보조기억장치의 가용 크기에 의해 제한된다.
가상주소(virtual address)	주기억장치처럼 참조될 수 있도록 가상 메모리의 특정 위치에 배정된 주소
가상 주소 공간	특정 프로세스에 할당된 가상 주소의 영역
주소 공간	특정 프로세스에 가용한 메모리 주소의 영역
실주소(real address)	주기억장치 상의 특정 위치의 주소

가상메모리의 실용성

- 프로그램 수행에 필요한 블록(메모리접근 오류가 발생한 블록)만 적재하므로, 비사용 블록 적재로 인한 낭비 절감
- 블록 적재 요구가 얼마나 빈번한지가 가상메모리 실용성의 최대 관건
  - ✓ 프로세스 수행 시간 중 임의의 '짧은' 구간을 관찰했을 때 메모리 참조 행태가 너무 분산되지 않아야 하며, 한 번 주기억장치에 적재된 블록들이 내보내지기 전까지 최대한 많이 참조되도록 관리해야 함
  - ✓ 시스템이 안정상태에 있을 경우 주기억장치 전체가 각 프로세스의 블록들로 채워지며, 이 때 주기억장치에 적재되지 않은 블록이 참조되면 이미 적재된 블록을 교체해야 함(교체 대상을 잘못 선정할 경우 바로 재적재해야 하는 상황 발생)
  - ✓ 결론적으로 스레싱(thrashing: 시스템이 프로세스 수행보다 블록 교체에 대부분의 시간을 소비하게 되는 현상) 방지가 관건

지역성의 원리(principle of locality)

- 프로세스의 메모리 참조가 군집을 이루는 특성
  - ✓ Temporal locality: 반복문, 스택, 카운터 변수, 함수 호출
  - ✓ Spatial locality: 순차수행, 배열 순회
- 지역성이 전제될 때, 가상메모리 실용성의 필요조건
  - ✓ 가상메모리 지원 하드웨어의 효율성
  - ✓ 주기억장치와 보조기억장치 간 블록 이동 관리의 효율성



Hardware and Control Structures

- Paging
  - ✓ Details of page table structure
    - Two-level page table
    - Inverted page table
  - ✓ Translation
  - ✓ TLB (Translation Lookaside Buffer)
  - ✓ Issue of page size
- Segmentation
  - ✓ Details of segment table structure
  - ✓ Translation
- Combined Paging and Segmentation
- Protection and Sharing

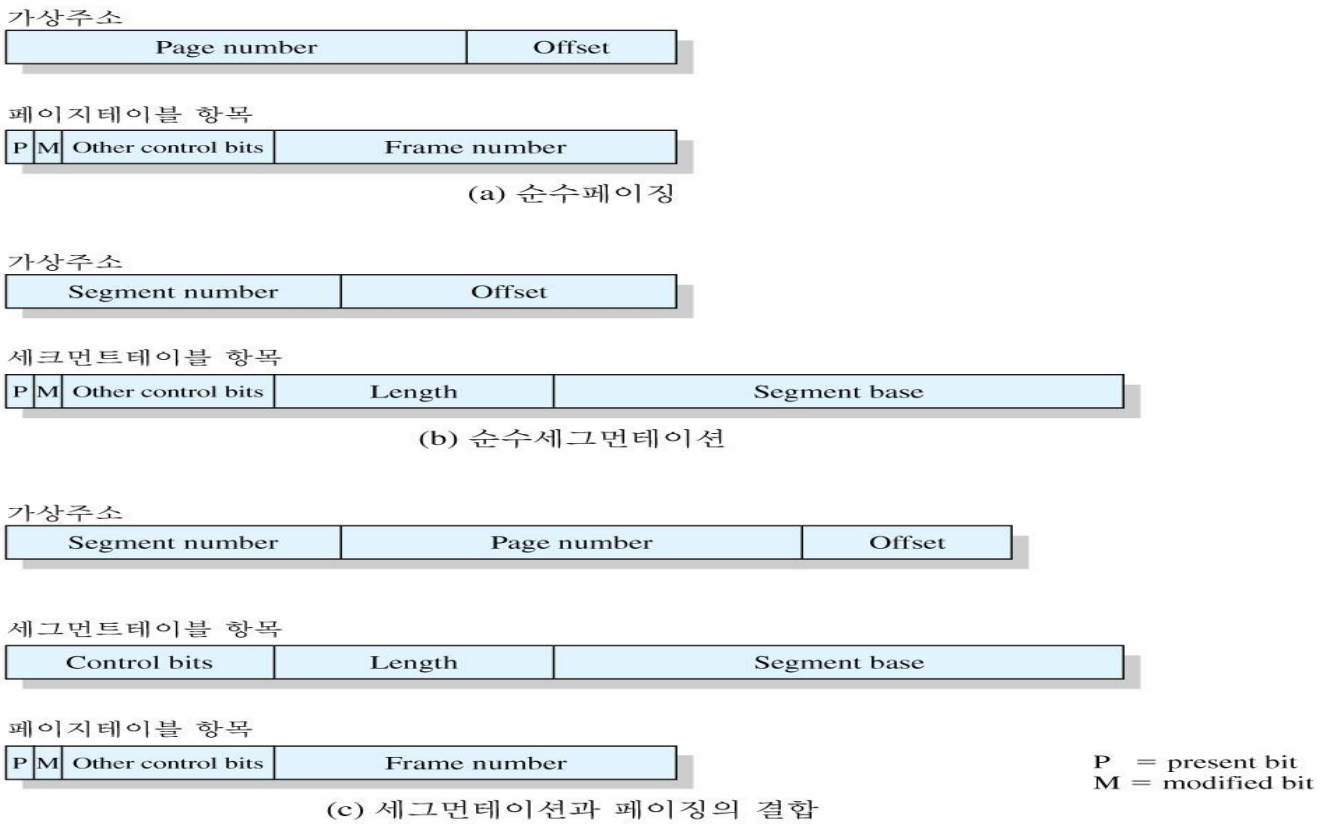
페이징(paging)

- 가상메모리는 고정크기 페이지로 분할되고, 물리메모리는 페이지 크기의 페이지프레임(혹은 프레임)들로 분할
- 페이지테이블(또는 역 페이지테이블)을 매개로 하드웨어/소프트웨어가 상호작용하여, 프로세스가 특정 가상주소(해당 페이지 상의 한 주소)를 참조할 경우 그에 대응된 물리주소(해당 페이지가 적재된 프레임 상의 대응 주소)가 참조되도록 주소 사상(mapping)/변환

- 일반적으로 프로세스별 페이지 테이블을 설정하여, 주소변환 하드웨어가 활용할 주소사상 정보 제공
- 페이지 테이블에 페이지 적재 여부가 표시되고, 적재되지 않은 페이지가 참조될 경우 하드웨어에 의해 페이지폴트(page fault) 이벤트 발생

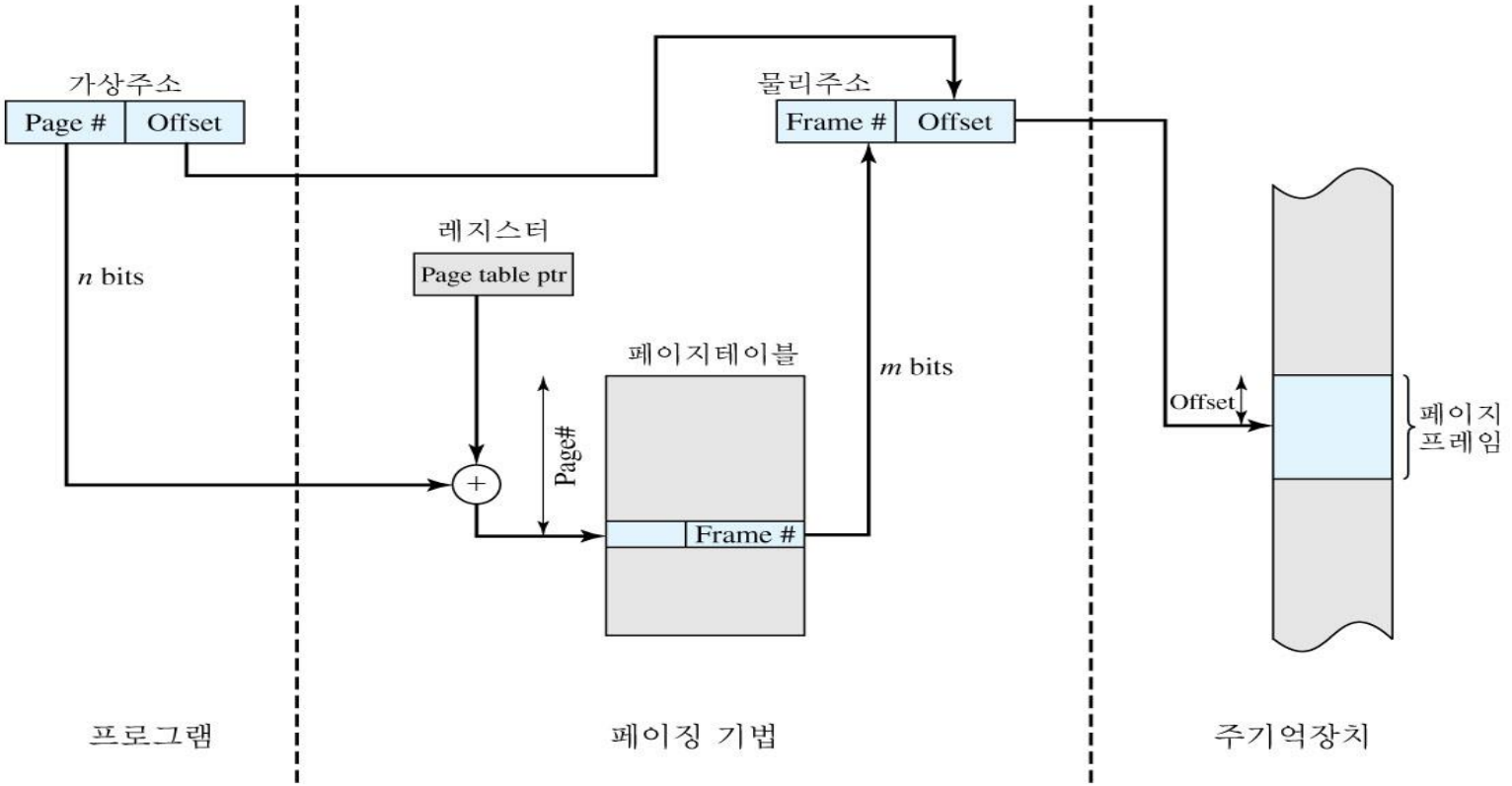
페이징: 페이지테이블 항목 구성

- 가상주소의 페이지 번호를 이용하여 대응된 페이지테이블 항목을 찾고, 그로부터 해당 페이지가 적재된 프레임의 번호를 얻음.
- 존재비트(Present bit)는 적재 여부를, 변경비트(Modify bit)는 적재된 이후 내용 변경 여부를 나타냄.



페이징: 주소변환

- 메모리에서 한 워드를 읽을 때마다 (페이지#, 오프셋) 으로 구성된 가상주소(또는 논리주소)가 물리주소(프레임#, 오프셋) 로 변환



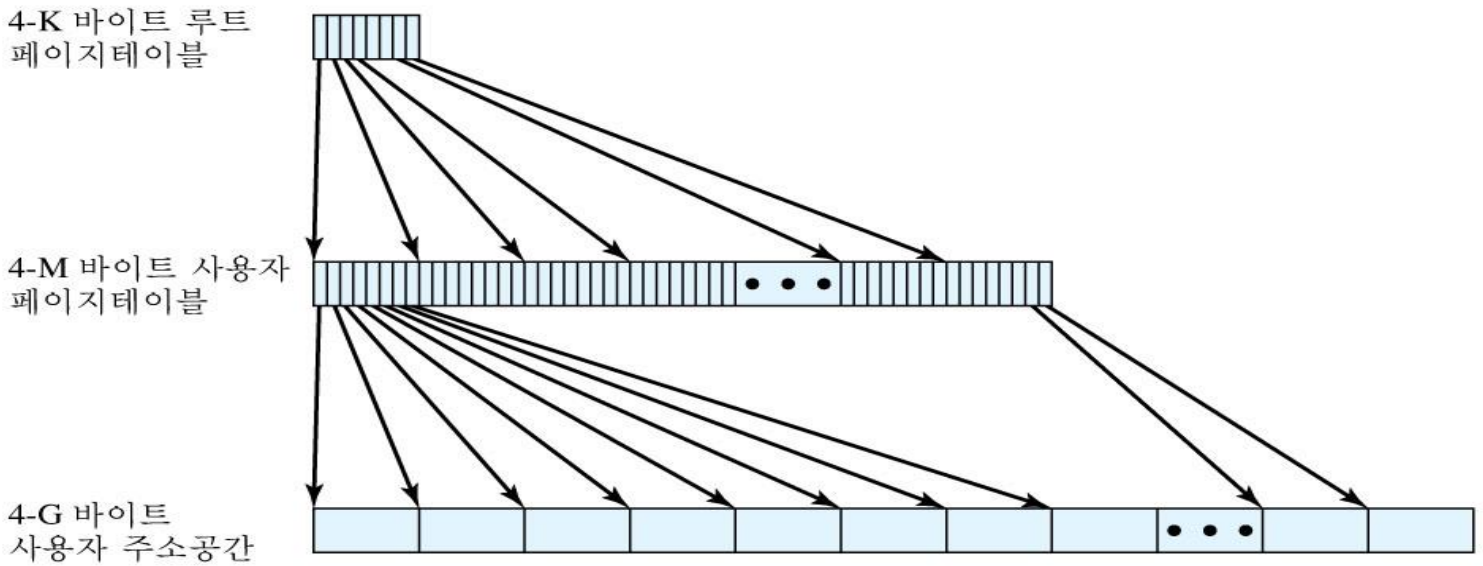
페이징: 페이지테이블 크기

- 가상주소 공간의 크기가 커질수록 페이지테이블의 크기가 증가하고, 주기억장치 상의 보다 큰 적재 공간 요구

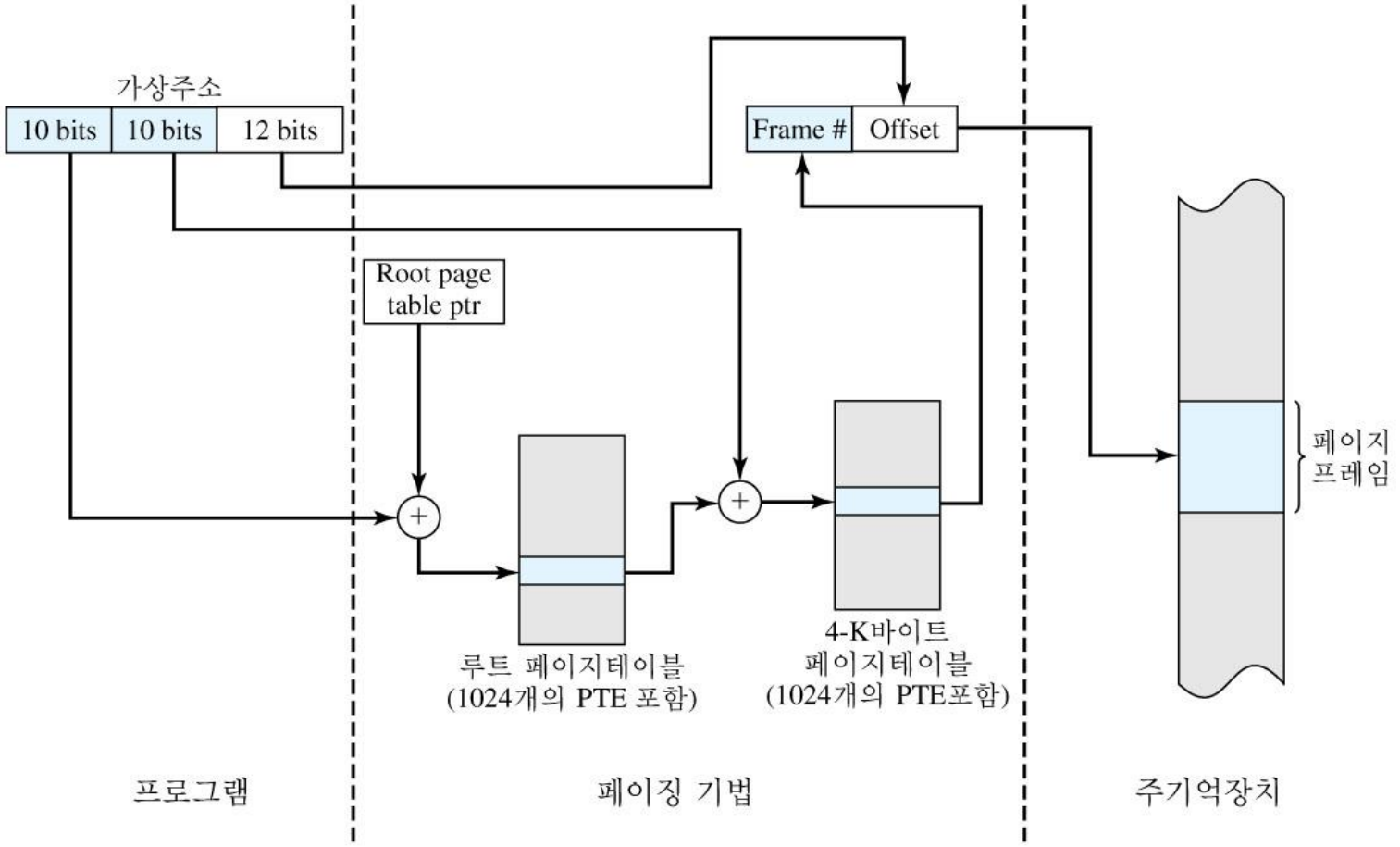
- 해결을 위한 접근 방법
  - ✓ 페이지테이블을 *n*-단계 계층구조로 구성: 선형구조가 아니므로, 전체 가상주소공간 중 실제 참조할 페이지들이 존재하는 영역에 대해서만 단계별 페이지테이블 설정 가능
  - ✓ 페이지테이블 자체를 가상메모리에 적재: 페이지테이블 중 일부분만 주기억장치에 적재한 채 해당 프로세스 수행 가능
  - ✓ 역페이지 테이블(*inverted page table*) 이용: 각 프레임에 어떤 페이지가 적재되어 있는지 사상시키므로, 역페이지 테이블의 크기는 주기억장치의 크기에 비례함

페이징: 2-단계 페이지테이블

- 페이지 디렉토리가 있어 디렉토리 항목 각각이 페이지테이블을 point
  - ✓ 페이지 디렉토리의 길이(항목 수) 가 X 이고 페이지테이블의 최대길이가 Y 일때, 한 프로세스는 최대 X * Y 페이지로 구성됨
- 32비트 주소체계에 적용 가능한 2단계 기법
  - ✓ 4K (2^{12}) byte 크기의 페이지를 가정하면 2^{20}개의 페이지로 구성된 4G(2^{32}) byte 크기의 가상주소 공간 설정



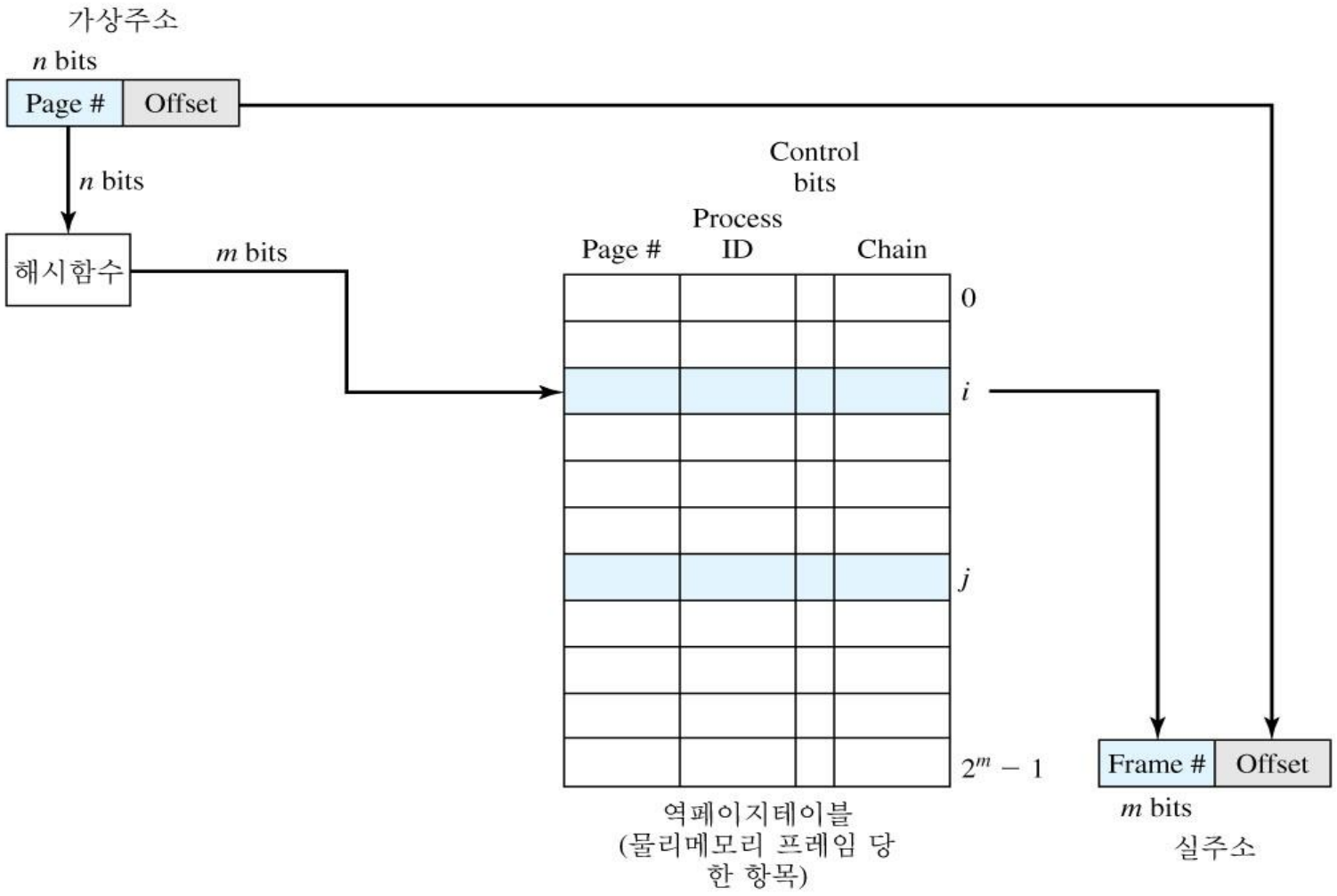
페이징: 주소변환(2-단계 페이지테이블)



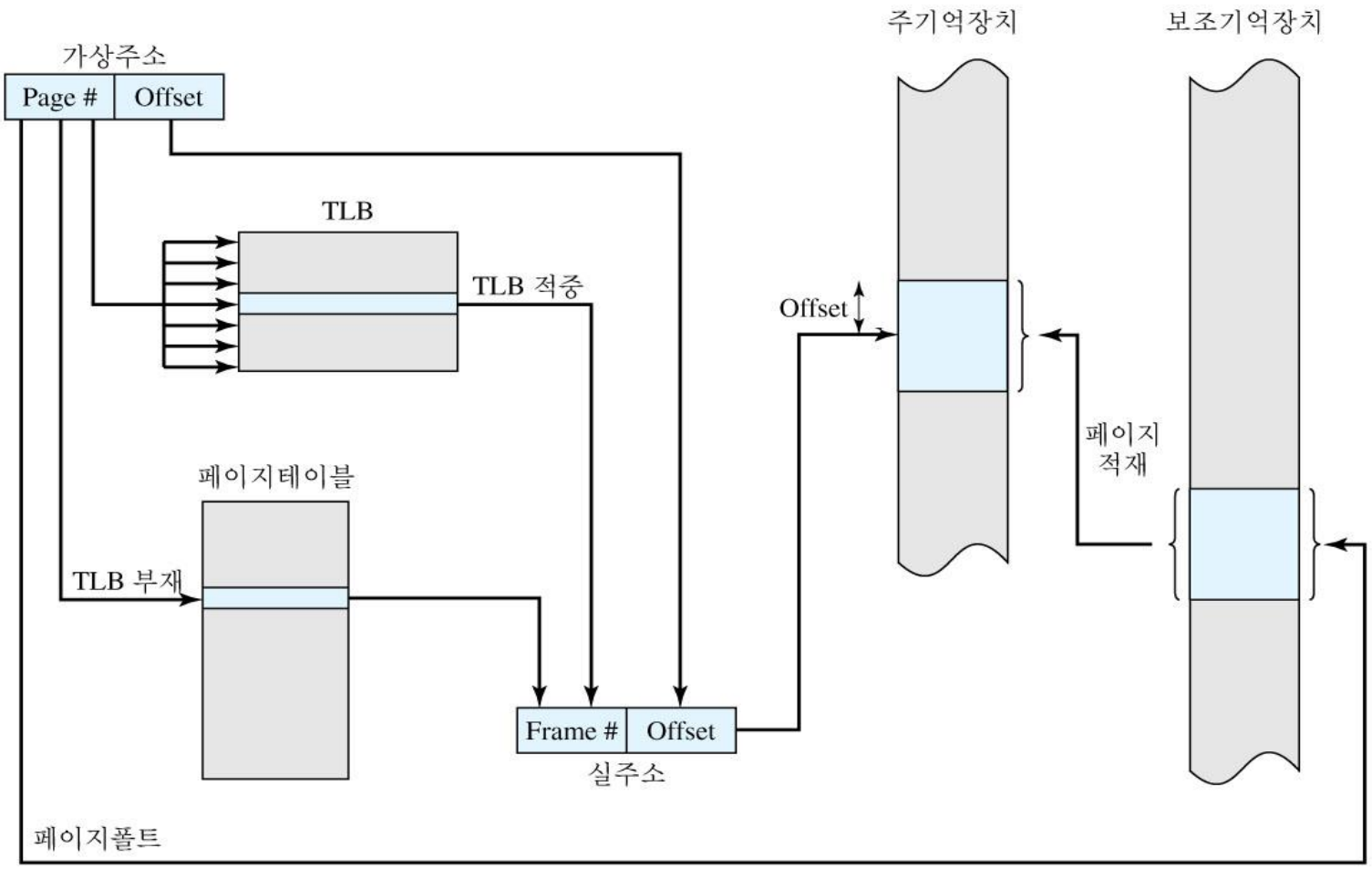
페이징: 역페이지 테이블

- 역 페이지 테이블 (inverted page table)
- 페이지당 하나가 아니라 프레임당 하나의 페이지테이블 항목 설정
  - ✓ 2^n개의 페이지, 2^m개의 프레임($n>m$), 2^m개의 테이블 항목
- 페이지 번호에 대한 해시값(*hash value*)을 해당 페이지가 적재된 프레임의 번호로 간주
  - ✓ *n*-비트 페이지 번호를 *m*-비트 값으로 사상시키는 해시함수 사용
- 충돌(collision) 해결을 위해 연결기법 사용
  - ✓ 전형적으로 한두 개의 테이블 항목 간에 체인 설정

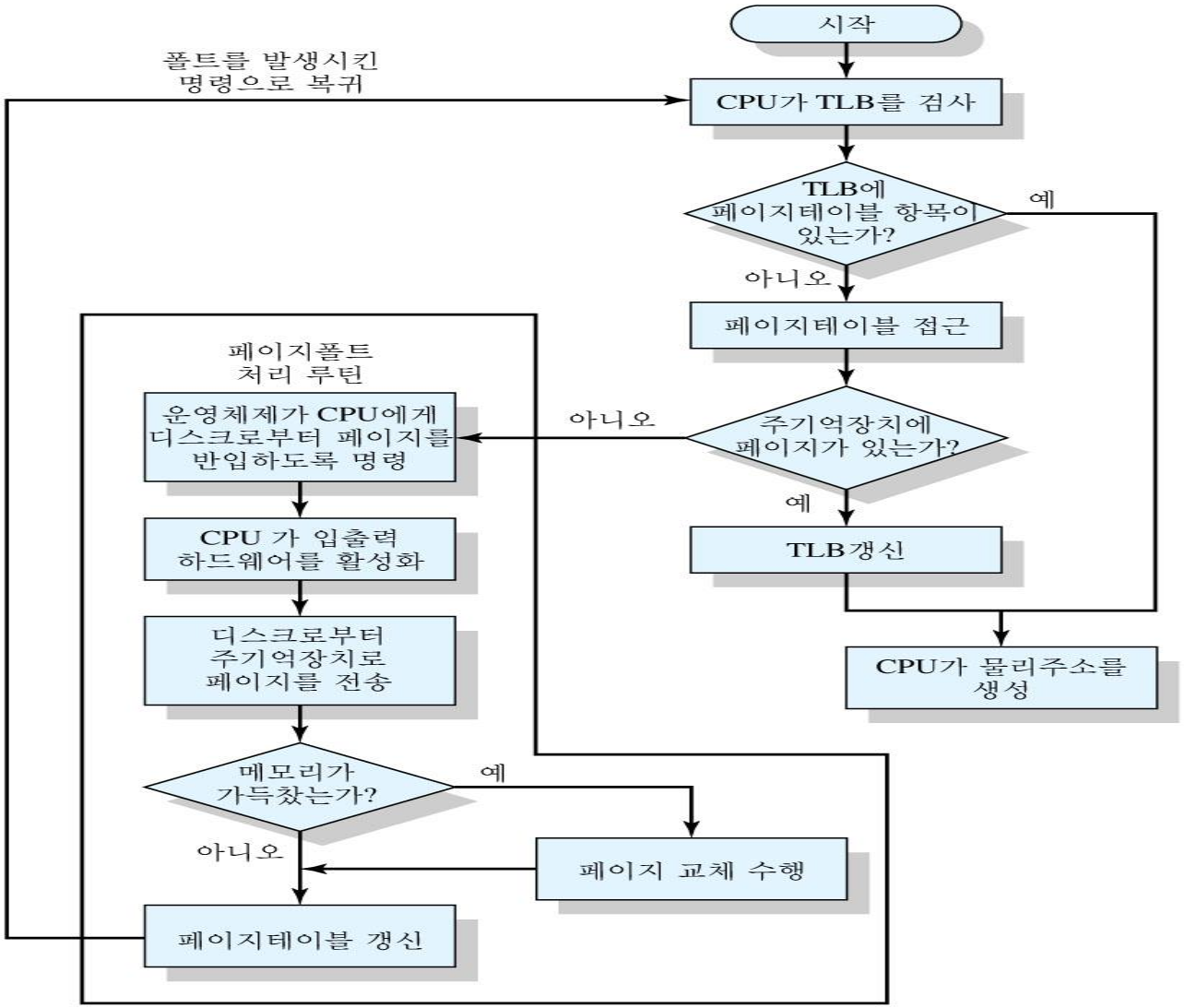
페이징: 역페이지 테이블의 구조



페이징: TLB의 사용

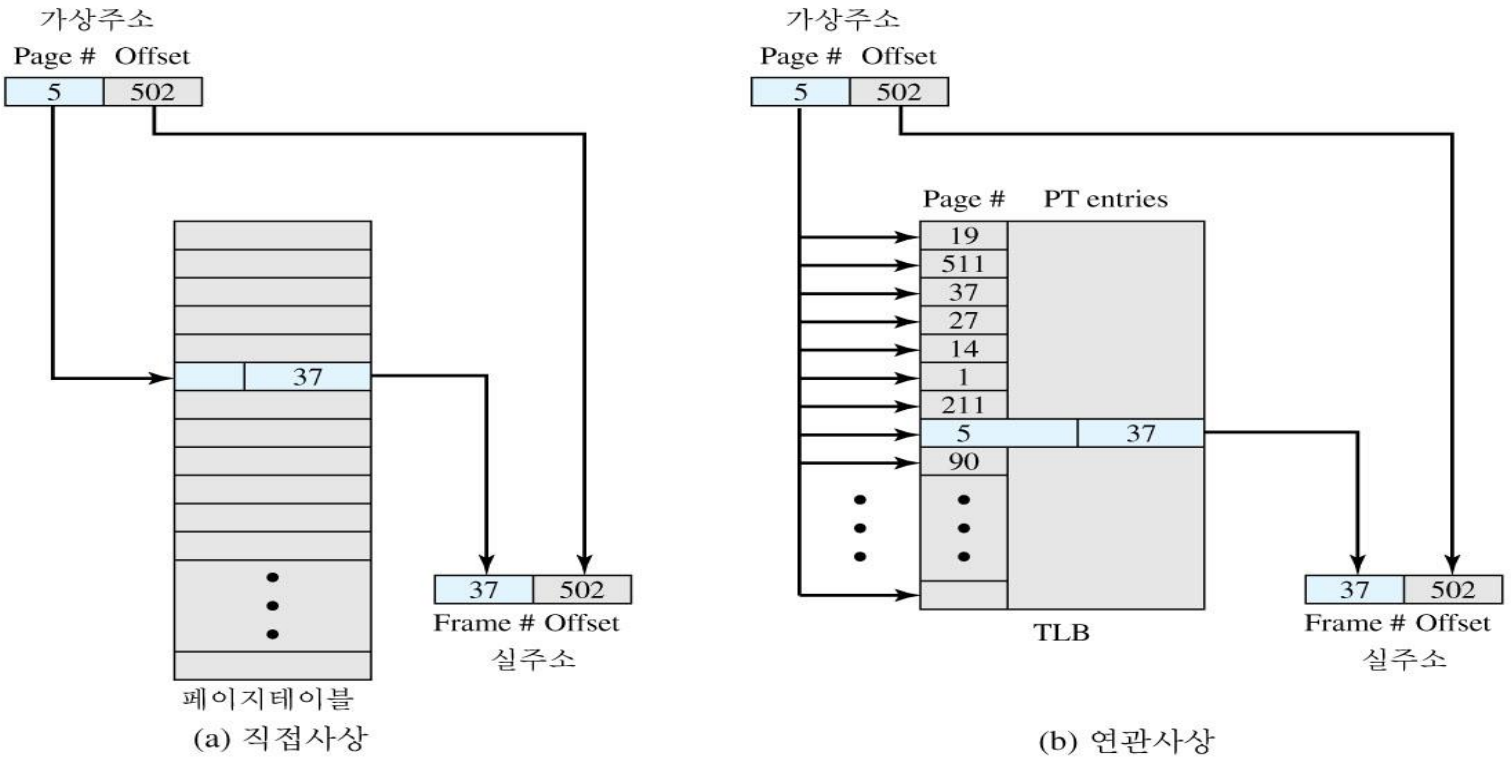

주기억장치
보조기억장치
가상주소
Page #
Offset
TLB
TLB 적중
Offset
페이지
적재
페이지테이블
TLB 부재
Frame #
Offset
실주소
페이지폴트

페이징: TLB의 동작

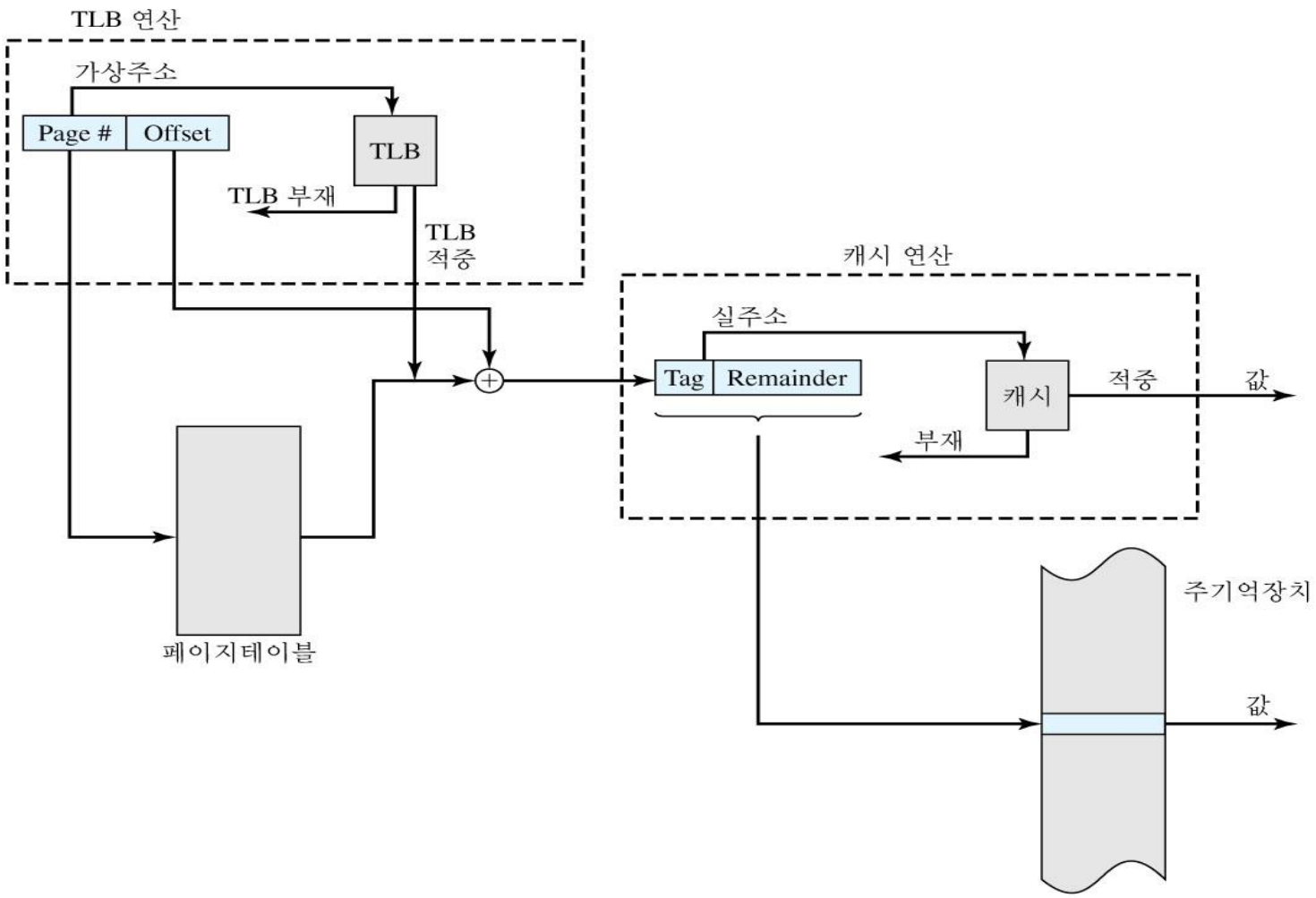

시작
폴트를 발생시킨 명령으로 복귀
CPU가 TLB를 검사
TLB에 페이지테이블 항목이 있는가?
예
아니오
페이지테이블 접근
페이지폴트 처리 루틴
운영체제가 CPU에게 디스크로부터 페이지를 반입하도록 명령
아니오
주기억장치에 페이지가 있는가?
예
CPU 가 입출력 하드웨어를 활성화
TLB갱신
디스크로부터 주기억장치로 페이지를 전송
CPU가 물리주소를 생성
메모리가 가득찼는가?
예
아니오
페이지 교체 수행
페이지테이블 갱신

페이징: TLB에 대한 연관사상

- 연관사상 (Associative mapping)
  - ✓ CPU에는 특정 페이지# 와 일치하는 TLB 항목이 있는지 결정하기 위해 다수의 TLB 항목들을 동시에 조사하는 HW 장착
- TLB 설계시 TLB 항목들을 어떻게 구성할지, 새로운 항목 반입시 어떤 항목을 교체할지 결정해야 함



페이징: TLB와 캐시의 동작

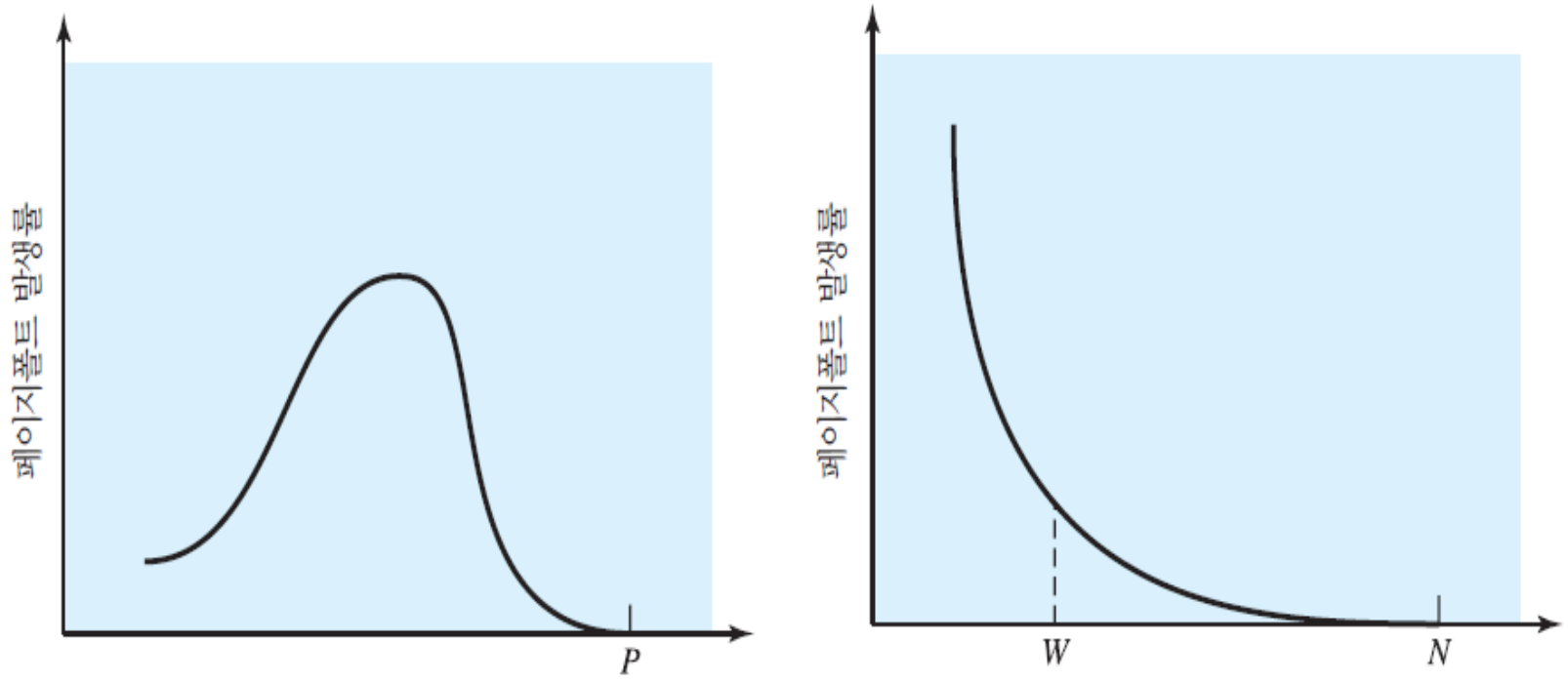


페이징: 페이지/프로그램 크기

- 페이지 크기가 작을 경우
  - ✓ 내부단편화의 양 감소
  - ✓ 페이지테이블 크기 증가:
    수행 프로그램의 크기나 다중 프로그래밍의 차수가 너무 클 때,
    페이지테이블을 부분적으로만 적재해야 하고,
    이 경우 페이지테이블 부재로 인한 페이지폴트 부담 발생
  - ✓ 동일 크기 데이터 적재에 필요한 입출력 회수 증가
  - ✓ 지역성과 관련된 부분만으로 적재집합 구성 가능
  - ✓ 프로세스 당 할당된 프레임이 많을수록, 페이지폴트 발생률 감소

- 프로그램 크기 커질 경우
  - ✓ 개발시 적용하는 객체지향기술, 다중쓰레딩 기술이 참조 지역성을 감소시킴
  - ✓ TLB 적중률이 감소하므로, 성능 감소

페이징: 페이지폴트 발생률



(a) 페이지 크기 (b) 할당된 페이지프레임 수

P = 전체 프로세스의 크기
W = 작업집합의 크기
N = 프로세스의 총 페이지 수

그림 8.10 프로그램의 전형적 페이징 행태

페이징: 페이지/프로그램 크기

표 8.3 페이지 크기의 사례

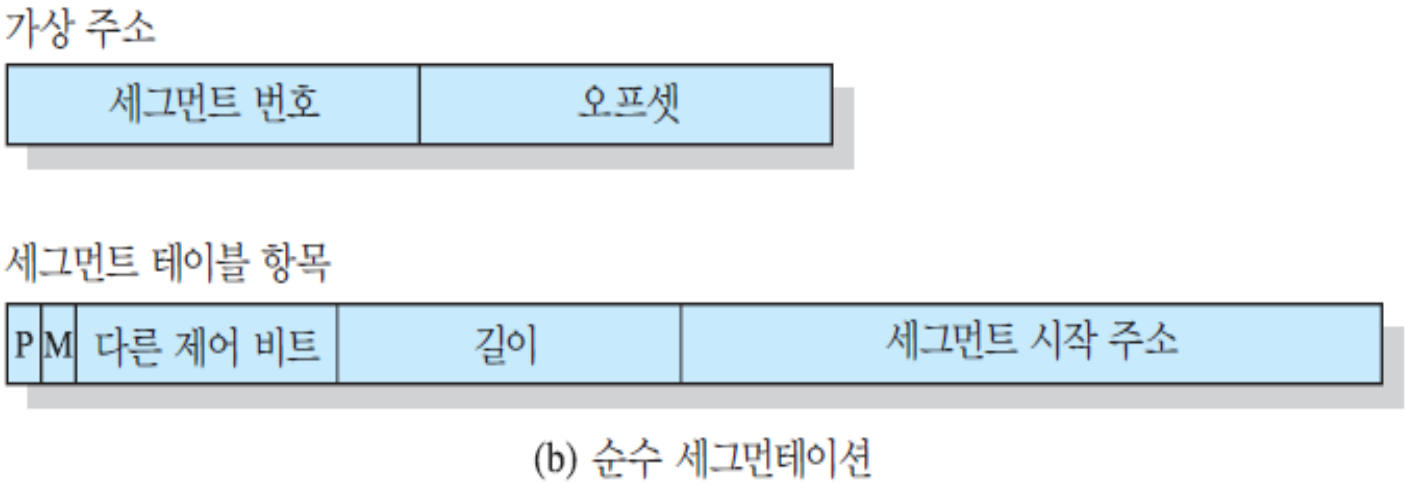
컴퓨터	페이지 크기
Atlas	512개의 48비트 워드
Honeywell-Multics	1024개의 36비트 워드
IBM 370	4 KB
VAX family	512 바이트
IBM AS/400	512 바이트
DEC Alpha	8 KB
MIPS	4 KB ~ 16 MB
UltraSPARC	8 KB ~ 4 MB
Pentium	4 KB 혹은 4 MB
Intel Itanium	4 KB ~ 256MB
Intel core i7	4 KB ~ 1 GB

세그먼테이션(segmentation)

- 프로세스의 주소공간을 동적으로 설정되는 서로 다른 크기의 블록들로 분할
- 세그먼테이션 장점
  - ✓ 확장성 자료구조에 대한 처리 단순화
  - ✓ 세그먼트 별로 독립적인 변경·재컴파일 가능
  - ✓ 논리적 개체(유틸리티 코드, 데이터 테이블 등)를 세그먼트로 설정하여 공유/보호 가능

세그먼테이션: 세그먼트테이블 항목 구성

- 가상주소의 세그먼트 번호를 이용하여 대응된 세그먼트 테이블 항목을 찾고, 그로부터 해당 세그먼트가 적재된 물리메모리 블록의 시작 주소를 얻음.
- 존재비트(Present bit)는 적재 여부를, 변경비트(Modify bit)는 적재된 이후 내용 변경 여부를 나타냄.



(b) 순수 세그먼테이션

세그먼테이션: 주소변환

- 특정 레지스터가 있어 현재 수행중인 프로세스의 세그먼트테이블을 point
- 가상주소의 세그먼트#를 index로 사용하여 로드된 세그먼트의 시작 위치에 해당하는 주기억장치의 주소를 찾음

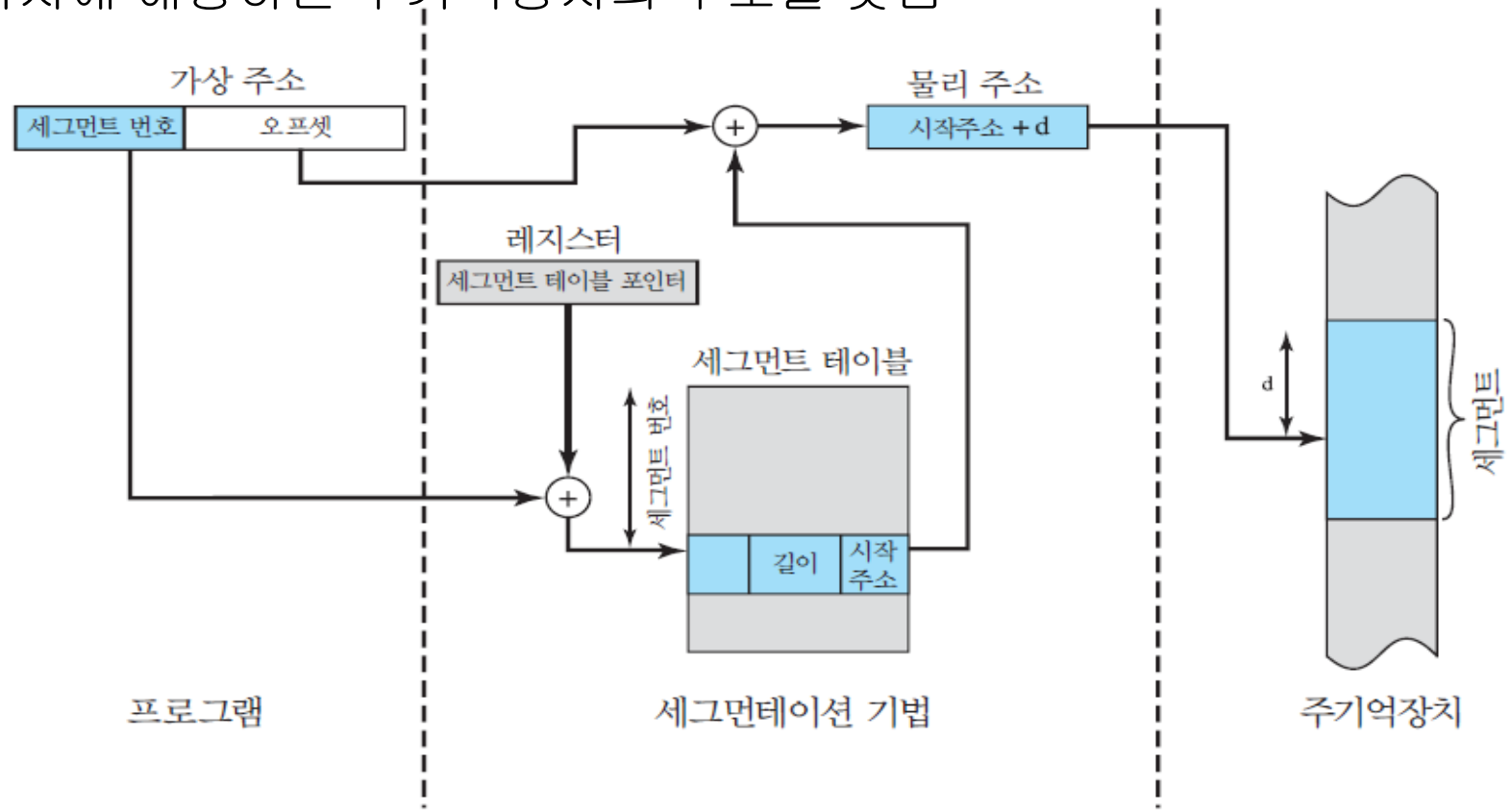


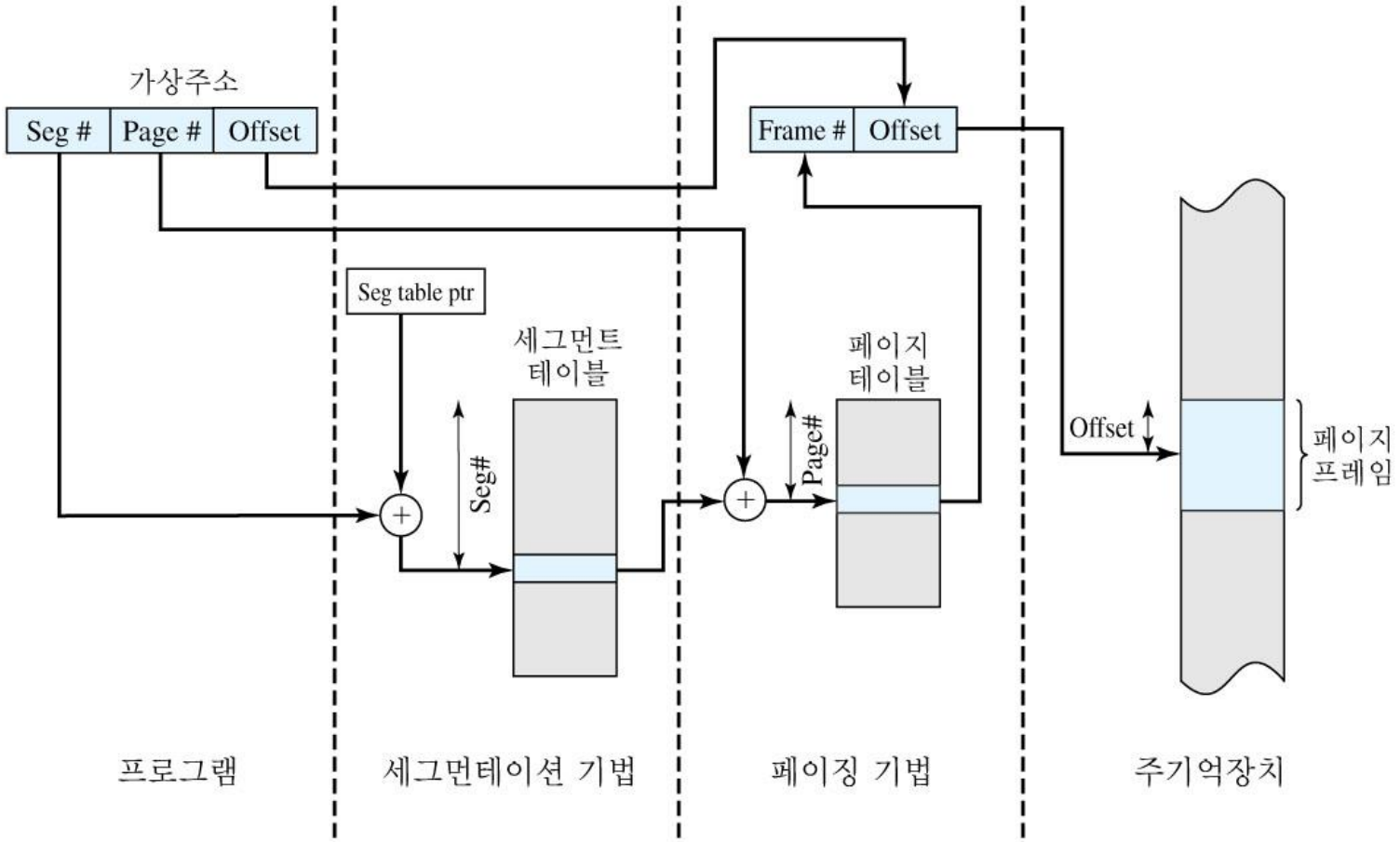
그림 8.11 세그먼테이션 시스템에서의 주소변환

세그먼테이션: 페이징과의 결합

- 두 시스템의 장점 수용
  - ✓ 페이징의 장점: 프로그래머에게 투명, 외부단편화 제거, 정교한 메모리 관리 알고리즘 적용 가능
  - ✓ 세그먼테이션의 장점: 확장성 자료구조 처리 단순화, 모듈단위 관리, 논리적 개체 단위의 공유/보호
- 각 세그먼트는 고정 크기의 페이지들로 분할
- 가상주소에 대한 관점
  - ✓ 프로그래머 관점:
    - 가상(논리)주소 = 세그먼트 번호 + 세그먼트 오프셋
  - ✓ 시스템 관점: 세그먼트 오프셋 = 페이지 번호 + 페이지 오프셋

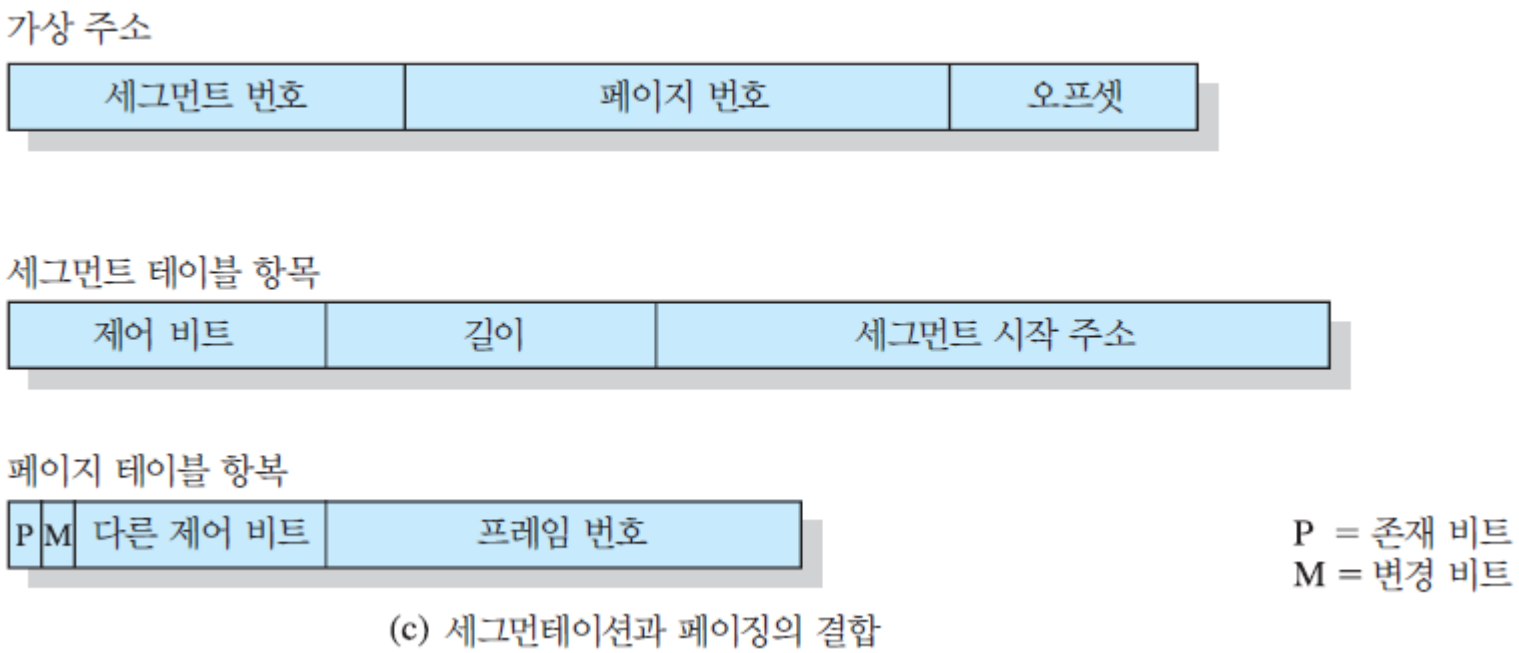
세그먼테이션: 페이징과 결합시의 주소변환

- 프로세스마다 1개의 세그먼트 테이블, 세그먼트마다 1개의 페이지테이블이 각각 연계됨
  - ✓ 특정 레지스터가 있어 현재 수행중인 프로세스의 세그먼트테이블을 point



세그먼테이션: 페이징과 결합시의 테이블 항목

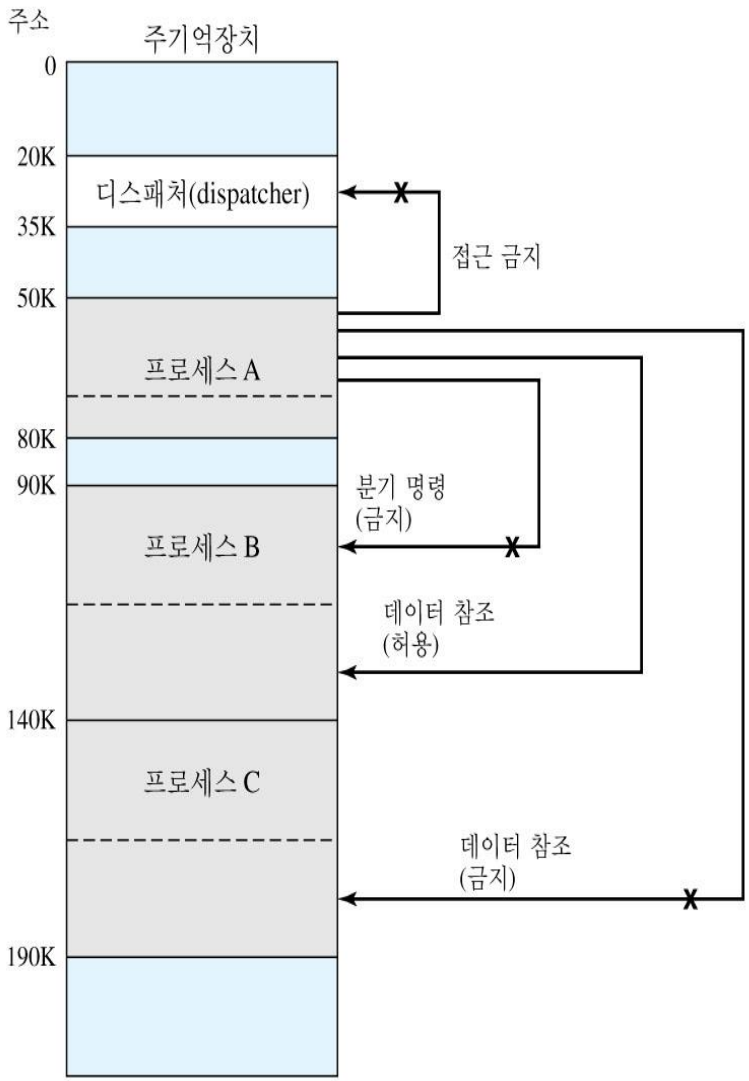
- '세그먼트 베이스' 필드는 세그먼트 자체의 시작주소가 아니라 해당 세그먼트에 대한 페이지테이블의 시작 주소
- 적재 및 변경 여부는 페이지 수준에서 관리되므로, 세그먼트테이블에는 존재(P) 비트나 변경(M) 비트가 없음



(c) 세그먼테이션과 페이징의 결합

세그먼테이션: 보호와 공유

- 세그먼트테이블 항목이 시작주소와 길이를 제한하기 때문에, 세그먼트 범위 밖의 주기억장치 영역에 접근 불가
- 페이지 구조가 프로그래머에게 투명한데 반해, 세그먼트 구조는 가시적이기 때문에 보호나 공유 요건을 정확히 표현 가능



페이징과 세그먼테이션의 특성

표 8.2 | 페이징과 세그먼테이션의 특성

단순 페이징	가상 메모리 페이징	단순 세그먼테이션	가상 메모리 세그먼테이션
주기억장치는 프레임이라 불리는 고정 크기의 작은 블록으로 분할됨.	주기억장치는 프레임이라 불리는 고정 크기의 작은 블록으로 분할됨.	주기억장치는 분할되지 않음.	주기억장치는 분할되지 않음.
프로그램은 컴파일러나 메모리 관리 시스템에 의해 다수의 페이지로 분할됨.	프로그램은 컴파일러나 메모리 관리 시스템에 의해 다수의 페이지로 분할됨.	프로그래머가 컴파일러에게 프로그램 세그먼트를 명시해 줌(분할 방식은 프로그래머에 의해 결정됨).	프로그래머가 컴파일러에게 프로그램 세그먼트를 명시해 줌(분할 방식은 프로그래머에 의해 결정됨).
내부 단편화	내부 단편화	내부 단편화 없음.	내부 단편화 없음.
외부 단편화 없음.	외부 단편화 없음.	외부 단편화	외부 단편화 없음.
각 페이지가 어느 프레임에 적재되어 있는지 나타내기 위해, 운영체제는 프로세스 각각을 위한 페이지 테이블을 관리해야 함.	각 페이지가 어느 프레임에 적재되어 있는지 나타내기 위해, 운영체제는 프로세스 각각을 위한 페이지 테이블을 관리해야 함.	각 세그먼트의 적재 위치와 길이를 나타내기 위해, 운영체제는 프로세스 각각을 위한 세그먼트 테이블을 관리해야 함.	각 세그먼트의 적재 위치와 길이를 나타내기 위해, 운영체제는 프로세스 각각을 위한 세그먼트 테이블을 관리해야 함.

페이징과 세그먼테이션의 특성

단순 페이징	가상 메모리 페이징	단순 세그먼테이션	가상 메모리 세그먼테이션
운영체제는 가용 프레임 리스트(free frame list)를 관리해야 함.	운영체제는 가용 프레임 리스트를 관리해야 함.	운영체제는 메모리 상에 있는 가용 공간(free hole)들의 리스트를 관리해야 함.	운영체제는 메모리 상에 있는 가용 공간들의 리스트를 관리해야 함.
처리기는 절대 주소 계산을 위한 오프셋으로 페이지 번호를 사용함.	처리기는 절대 주소 계산을 위한 오프셋으로 페이지 번호를 사용함.	처리기는 절대 주소 계산을 위한 오프셋으로 세그먼트 번호를 사용함.	처리기는 절대 주소 계산을 위한 오프셋으로 세그먼트 번호를 사용함.
오버레이를 사용하지 않을 경우, 수행할 프로세스의 모든 페이지가 주기억장치에 있어야 함.	수행할 프로세스의 모든 페이지가 주기억장치의 프레임 상에 있을 필요 없음. 필요할 때 적재 가능함.	오버레이를 사용하지 않을 경우, 수행할 프로세스의 모든 세그먼트가 주기억장치에 있어야 함.	수행할 프로세스의 모든 세그먼트가 주기억장치의 프레임 상에 있을 필요 없음. 필요할 때 적재 가능함.
	주기억장치로 한 페이지를 읽어들이기 위해, 한 페이지를 디스크에 기록해야 할 수도 있음.		주기억장치로 한 세그먼트를 읽어들이기 위해, 하나 이상의 세그먼트를 디스크에 기록해야 할 수도 있음.

8.2 운영체제의 가상메모리 관리 정책

- 가상메모리 관리 정책에 있어 주요 이슈는 성능
- 페이징 관련 이슈에 집중
- 가상메모리 관리 정책의 범주
  - ✓ 반입정책(fetch policy)
  - ✓ 배치정책(placement policy)
  - ✓ 교체정책(replacement policy)
  - ✓ 적재집합 (resident set) 관리정책
  - ✓ 클리닝정책(cleaning policy)
  - ✓ 부하제어(load control)

반입정책(Fetch Policy)

- 각 페이지를 언제 주기억장치로 적재할지 결정하는 정책
- 요구반입(*demand paging*)
  - ✓ 페이지폴트(적재되지 않은 페이지 중 일부분 참조)시 적재
  - ✓ 일반적인 경우 지역성에 의해 안정적 운용 가능
- 선반입(*prepaging*)
  - ✓ 페이지폴트에 의해 요구된 페이지 이외의 페이지도 적재
    - 프로그램 수행을 시작할 때나 페이지폴트시 적용
    - 한 프로세스의 페이지들이 보조기억장치에 연속적으로 저장되어 있을 경우 그들을 한꺼번에 반입하는 것이 나중에 필요할 때 따로따로 반입하는 것보다 효율적임.
  - ✓ 스와핑(*swapping*)과 구분
    - Swap-out시 프로세스관련 로드된 페이지 전체가 M.M 밖으로 나갔다가, 프로세스 수행 재개시 swap-in

배치정책(Placement Policy)

- 적재될 블록이 주기억장치의 어디에 위치할 것인지 결정하는 정책
- 페이징 시스템의 경우, 주소변환 하드웨어와 주기억장치 접근 하드웨어들이 어떠한 페이지/프레임 조합에 대해서도 같은 효율로 기능하기 때문에 배치정책은 무의미
- NUMA(NonUniform Memory Access) 구조의 다중처리기인 경우, 각 페이지를 그것을 참조할 처리기와 가까운 메모리 모듈에 배치시키는 배치전략 필요

교체정책(Replacement Policy)

- 가용 프레임이 없을 경우, 새로운 페이지를 반입하기 위해 현재 적재되어 있는 페이지들 중 어떤 페이지를 교체할 것인지 결정하는 정책
- 가까운 미래에 참조될 가능성이 가장 적은 페이지를 선택하여 교체하는 것이 교체정책의 이상적 목표
- 지역성의 원리(principle of locality)를 전제로 과거의 참조 행태에 근거하여 미래의 참조 가능성 예측

교체정책: 프레임 잠금

- 교체 대상에서 배제시키기 위해 프레임 잠금(locking) 설정
  - ✓ 프레임 별로 잠금비트 설정하여 잠금 상태 표시
  - ✓ 프레임테이블, 페이지테이블 등에 잠금비트 유지 가능
- 프레임 잠금 대상
  - ✓ 운영체제 커널 중 주요 자료구조
  - ✓ 입출력 버퍼
  - ✓ 시간이 중요한 영역

교체정책: Optimal과 LRU

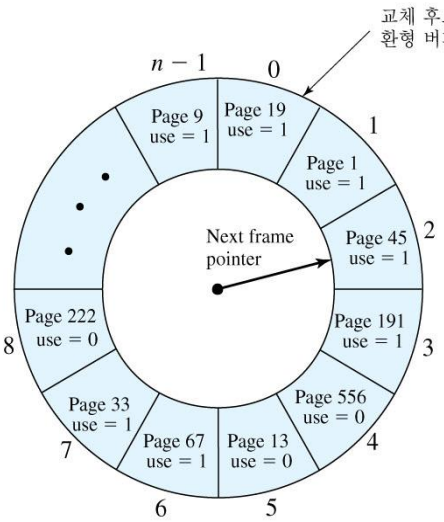
- 최적(Optimal)
  - ✓ 가장 오랫동안 참조되지 않을 페이지 교체
  - ✓ 가장 낮은 페이지폴트율
  - ✓ 미래에 대한 정확한 지식이 없어 구현 불가능
- LRU(Least Recently Used)
  - ✓ 가장 오랫동안 참조되지 않은 페이지 교체
  - ✓ 최적에 근접한 성능
  - ✓ 구현이 어렵고 큰 오버헤드

교체정책: FIFO와 Clock

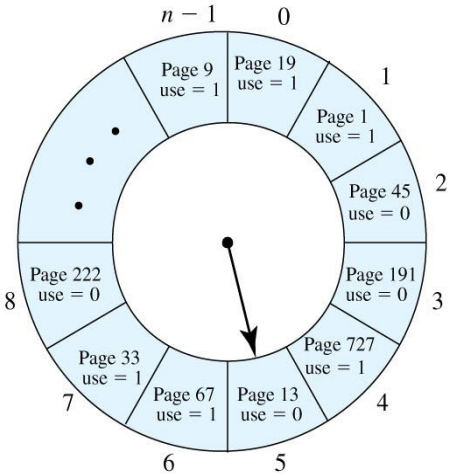
- FIFO(First-In First-Out)
  - ✓ 가장 오래 전에 적재된 페이지 교체
  - ✓ 가장 쉽게 구현
  - ✓ FIFO's anomaly: 특정 참조열에 대해 패턴에 대해 할당된 프레임 수를 증가시켰음에도 보다 많은 페이 더 많은 페이지폴트가 발생하는 현상
- 클록(Clock)
  - ✓ 프레임 별로 use 비트 연계시켜, 처음 적재시 1, 참조시 1로 설정
  - ✓ 페이지를 적재한 프레임들이 환형으로 배치되어 있다고 간주하고, 첫 교체후보를 가리키는 포인터(시계바늘) 설정
  - ✓ 시계 방향으로 포인터를 이동시키면서 포인터가 가리키는 프레임 중 use 비트가 0인 첫 프레임 상의 페이지를 교체(use 비트가 1인 경우 그 값을 0으로 변경하고 다음 프레임으로 이동)

교체정책: 클록 정책의 적용 예

- 시계바늘이 프레임 2를 가리키고 있는 상황에서, 페이지 227을 적재하기 위해 클록정책 작동
- 프레임 2와 3의 use 비트 값이 1이므로 차례로 그 값을 0으로 변경한 후, use 비트 값이 0인 프레임 4를 발견하여 교체페이지로 선택하게 됨.
- 프레임 4에 페이지 227을 적재하고 그 use 비트 값을 1로 한 후, 시계바늘이 그 다음 프레임을 가리키게 설정함.

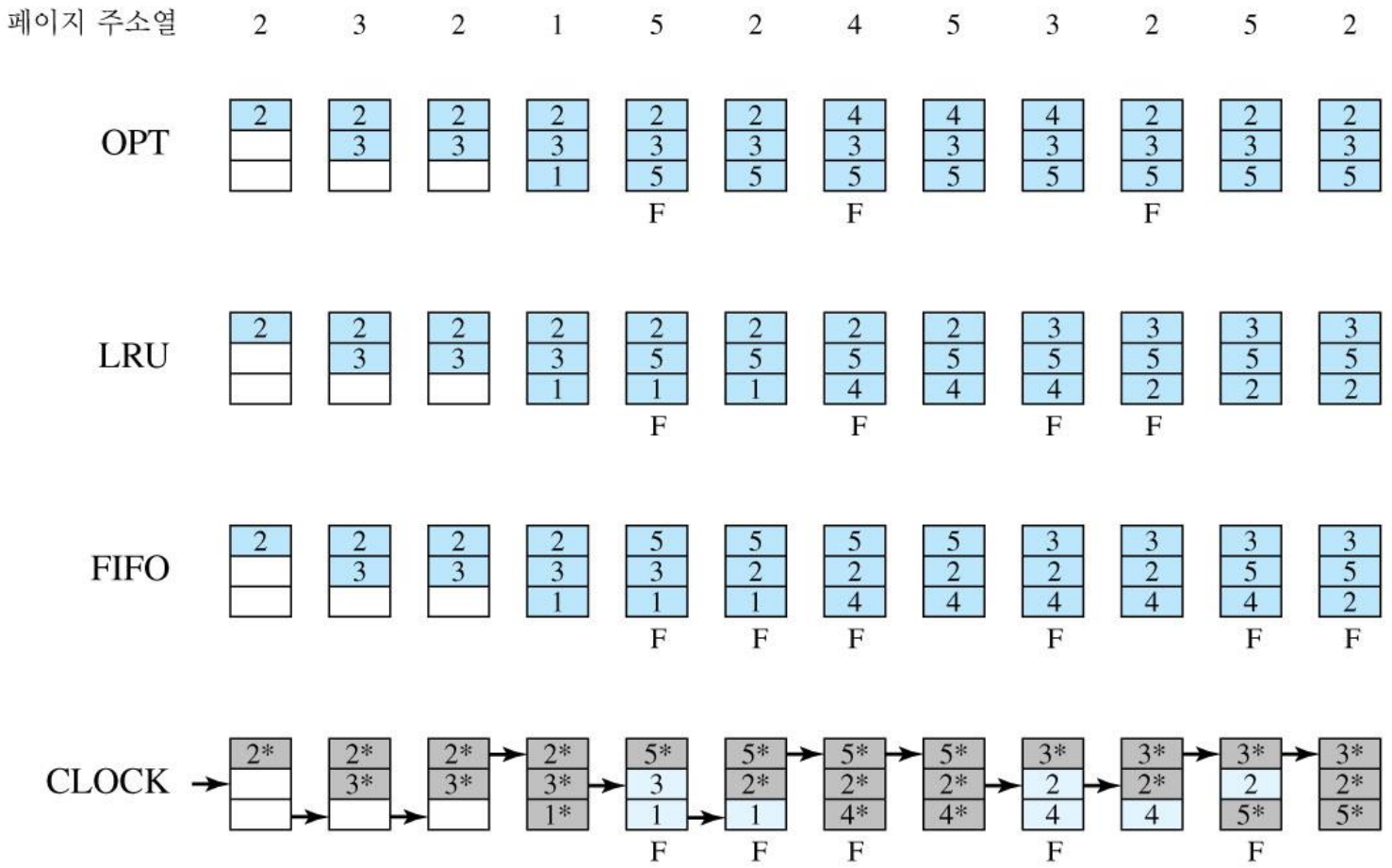


(a) 페이지 교체 직전의 버퍼 상태



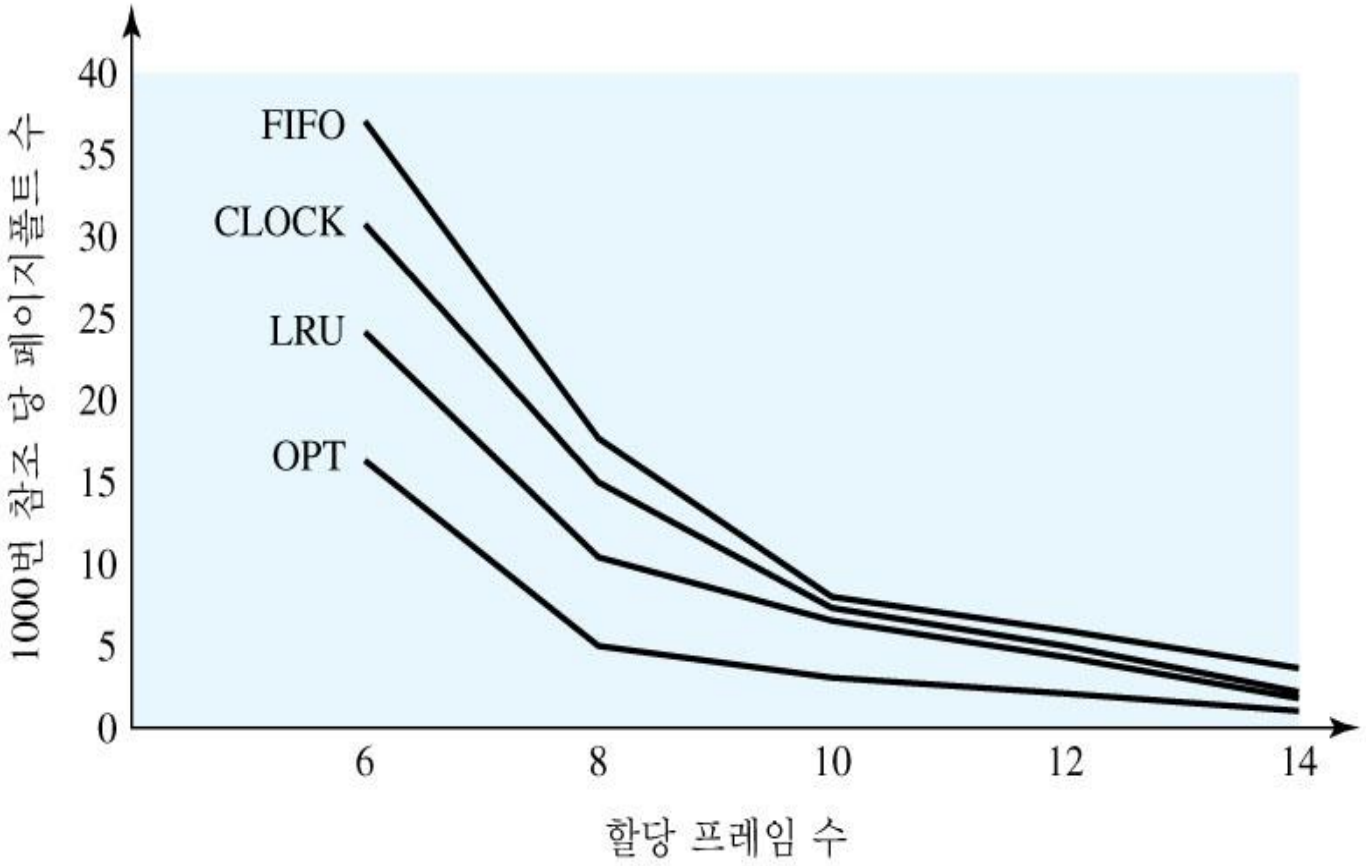
(b) 페이지 교체 직후의 버퍼 상태

교체정책: 기본 알고리즘의 적용 예



F = 할당 프레임이 초기에 채워진 이후 발생한 페이지폴트

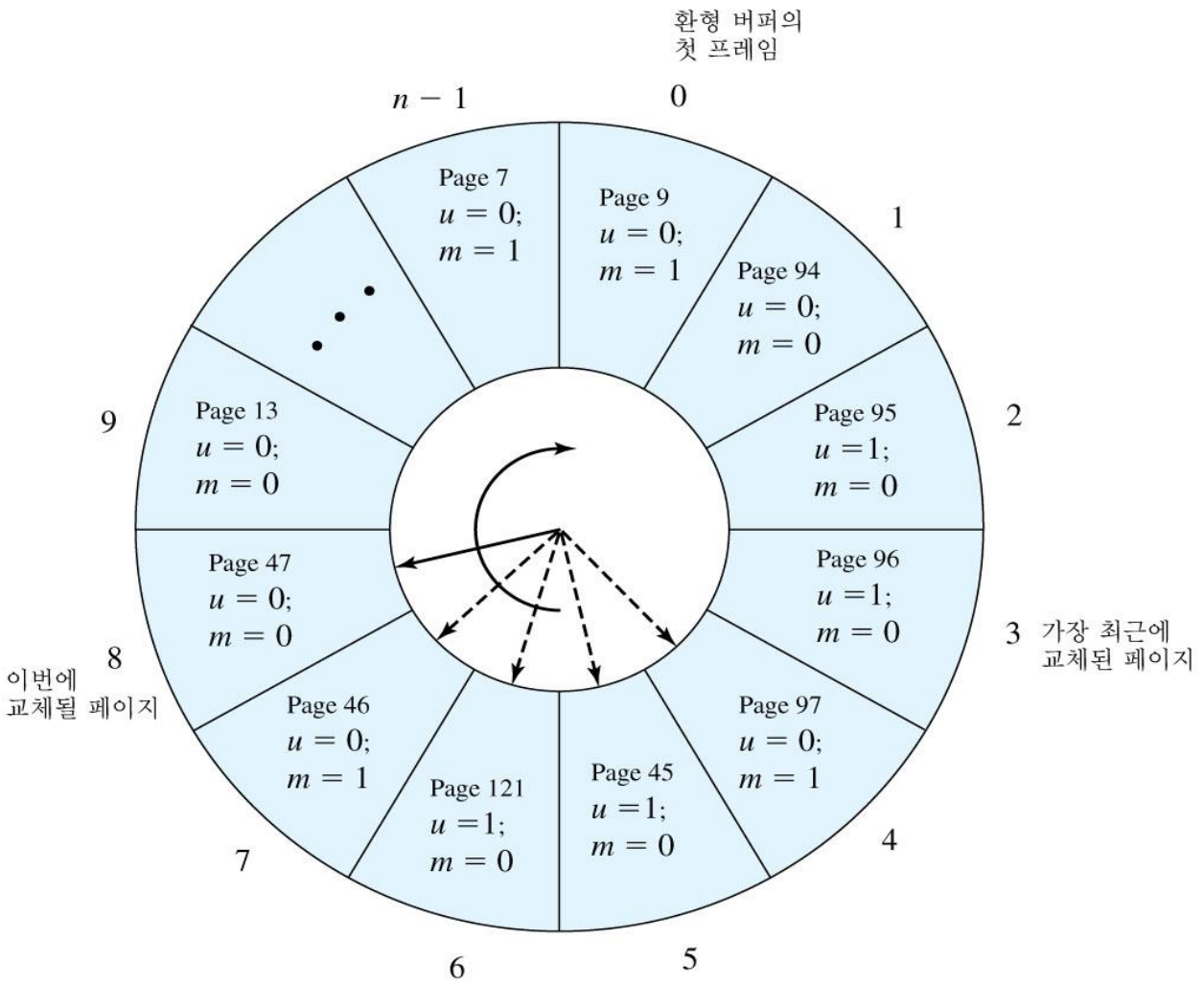
교체정책: 고정할당/지역교체 알고리즘의 비교



교체정책: 변경 비트가 추가된 클록정책

- 사용비트 u와 변경 비트 m의 값에 따라 4 가지 유형의 프레임으로 구분하여 u 값이 동일할 때 m = 1인 페이지에 앞서 m = 0인 페이지 교체함으로써, 교체 당할 페이지를 디스크에 기록하는 부담 절감
- 작동 과정
  1. 첫 번째 스캔: (u = 0; m = 0)인 첫 프레임 선택
  2. 단계 1 실패시 두 번째 스캔: (u = 0; m = 1)인 첫 프레임 선택(스캔된 모든 프레임의 u 값을 0으로 설정)
  3. 단계 2 실패시 단계 1부터 반복(재차 단계1이 실패하여 단계 2가 반복될 경우, 모든 프레임에 대해 u 값은 0, m 값은 1이라는 조건 성립이 보장되므로 첫 스캔 대상 프레임이 교체 대상으로 선택됨)

교체정책: 변경 비트가 추가된 클록정책



교체정책: 페이지 버퍼링

- 교체 대상으로 선택된 페이지를 즉시 교체하지 않고 가용 페이지리스트 뒤쪽에 연결시켜 어느 정도 기간 동안 주기억장치 상에 유지
  - ☞ 변경된 페이지들을 구별하여 변경페이지 리스트에서 디스크 기록을 대기하게 할 수 있음
- 일정 범위의 가용페이지 수를 유지하고, 실제 새로운 페이지를 적재해야 할 때 가용페이지 리스트 상의 첫 페이지를 교체
  - ☞ 변경페이지 리스트 상의 페이지가 디스크에 기록될 경우 가용페이지 리스트로 이동
- 페이지 버퍼링의 유익
  - ✓ 실제로 교체되기 이전에 참조될 경우 적은 비용으로 페이지폴트 해결 가능
  - ✓ 변경페이지들에 대한 클러스터 입출력 적용 가능

적재집합 관리

- 적재집합 관리 역할에 있어서의 교체정책
- 적재집합의 크기 관리
  - ✓ 고정할당(fixed allocation) : 프로세스에 할당된 프레임 수 고정
  - ✓ 가변할당(variable allocation): 프로세스에 할당된 프레임 수의 변경 허용
- 교체범위 관리
  - ✓ 지역범위(local scope) : 해당 프로세스의 적재집합 내에서 교체 대상 선택
  - ✓ 전역범위(global scope) : 주기억장치 상의 모든 페이지 중에 교체 대상 선택

적재집합 관리: 고정할당/지역범위

- 응용의 타입이나 프로그램 요구 등에 의거하여 프로세스에 대한 할당량 결정
- 할당량이 너무 적을 경우, 높은 페이지폴트 발생률
- 할당량이 많아질 경우, 다중 프로그래밍의 차수가 적어져 처리기 유휴시간이나 스와핑 시간 증가

표 8.5 | 적재 집합 관리

	지역 교체	전역 교체
고정 할당	• 프로세스에 할당된 프레임 수 고정 • 해당 프로세스에 할당된 프레임 중에서 교체될 페이지 선택	• 불가
가변 할당	• 프로세스에 할당된 프레임 수는 프로세스의 작업 집합을 유지하기 위해 수시로 변경 가능 • 해당 프로세스에 할당된 프레임 중에서 교체될 페이지 선택	• 주기억장치의 모든 프레임 중에 교체될 페이지 선택. 이로 인해 프로세스의 적재 집합 크기가 변경됨.

적재집합 관리: 가변할당/전역범위

- 구현이 쉽고, 많은 운영체제에 의해 채택
- 전형적 구현 전략
  - ✓ 운영체제는 가용프레임 리스트 유지
  - ✓ 페이지폴트 발생시 해당 프로세스의 적재집합에 가용프레임 추가
  - ✓ 가용프레임이 없을 경우, 잠긴 프레임 이외의 모든 프레임을 대상으로 교체 대상 페이지를 선택(선택된 페이지를 소유한 프로세스의 적재집합 축소 ⇨ 최적의 프로세스가 아닐 수 있음)하여 교체
- 잘못된 페이지 선택 문제 해소 방안: 페이지 버퍼링
  - ✓ 페이지가 덮여 쓰이기 전에 참조될 경우 효율적으로 회수될 수 있어, 페이지 선택의 중요성이 어느정도 감소됨

적재집합 관리: 가변할당/지역범위

- 전형적 구현 전략
  - ✓ 프로세스를 처음 적재할 때, 응용의 타입이나 프로그램 요구, 또 다른 척도 등에 의거하여 어느 정도의 프레임들을 적재집합으로 할당
  - ✓ 페이지폴트 발생시 해당 프로세스의 적재집합 내에서 교체
  - ✓ 수시로, 프로세스에 대한 할당량을 재평가하고 전체적 성능 개선이라는 측면에서 할당량 증감
- 이 전략의 주 요소는 적재집합의 크기와 그 변경 시점을 결정하는 규칙
- 대표적 전략
  - ✓ 작업집합 전략(*working set strategy*)
  - ✓ PFF (*Page Fault Frequency*)
  - ✓ VSWS (*Variable-interval Sampled Working Set*)

적재집합 관리: 작업집합 전략

- 특정 프로세스(페이지 수 N)에 대한 작업집합 $W(t, \Delta)$
  - ✓ 해당 프로세스가 가상시간(매 가상주소 참조에 대해 1씩 증가) 상의 시점 $t - \Delta$ 부터 t 까지 참조한 페이지들의 집합
  - ✓ $W(t, \Delta+1) \supseteq W(t, \Delta)$, $1 \leq |W(t, \Delta)| \leq min(\Delta, N)$
- 작업집합 관리 방법
  - ✓ 각 프로세스의 작업집합을 모니터링
  - ✓ 주기적으로(극단적으로 매 참조시점마다) 프로세스의 적재집합 중 작업집합에 있지 않은 페이지들을 제거(기본적으로 LRU 정책)
  - ✓ 프로세스는 주기억장치에 그 작업집합이 있을 때(적재집합이 작업집합을 포함할 때)만 수행
- 문제점
  - ✓ 작업집합 모니터링 비용의 비현실성
  - ✓ 최적의 Δ값이 알려져 있지 않고 어떤 경우라도 가변적

적재집합 관리: 작업집합 전략

페이지 참조열	윈도우 크기, Δ = 2	3	4	5
24	24	24	24	24
15	24 15	24 15	24 15	24 15
18	15 18	24 15 18	24 15 18	24 15 18
23	18 23	15 18 23	24 15 18 23	24 15 18 23
24	23 24	18 23 24	•	•
17	24 17	23 24 17	18 23 24 17	15 18 23 24 17
18	17 18	24 17 18	•	18 23 24 17
24	18 24	•	24 17 18	•
18	•	18 24	•	24 17 18
17	18 17	24 18 17	•	•
17	17	18 17	•	•
15	17 15	17 15	18 17 15	24 18 17 15
24	15 24	17 15 24	17 15 24	•
17	24 17	•	•	17 15 24
24	•	24 17	•	•
18	24 18	17 24 18	17 24 18	15 17 24 18

그림 8.19 윈도우 크기에 따른 프로세스의 작업 집합

적재집합 관리: 작업집합 크기의 변화

적재집합 관리: PFF 정책

- PFF (Page Fault Frequency) 정책
  - ✓ 페이지폴트 발생률을 근거로 적재집합 크기 결정
- 적재집합 관리 방법
  - ✓ 메모리 상의 각 페이지와 연계된 사용비트 설정(사용비트 값은 해당 페이지가 참조될 때 1이 됨)
  - ✓ 페이지폴트 발생시 해당 프로세스가 직전에 페이지폴트를 발생시켰던 시점으로부터 경과된 가상시간 측정(페이지 참조 카운터를 유지할 경우 가능)
  - ✓ 경과 시간이 정의된 임계치 보다 작을 경우 해당 프로세스의 적재집합에 한 페이지 추가. 그렇지 않을 경우 사용비트가 0인 모든 페이지를 적재집합에서 제거시켜 적재집합 축소(제거되지 않은 페이지의 사용비트는 모두 0으로 설정)
- 주요 단점
  - ✓ 작업집합이 전이되는 과도 기간에 적재집합을 효과적으로 제어하지 못함

적재집합 관리: VSWS 정책

- VSWS (Variable-interval Sampled Working Set) 정책
  - ✓ 샘플링 간격 설정하여 샘플링 기간 시작할 때 적재집합 내 모든 페이지의 사용비트를 0으로 설정하고, 샘플링 기간이 끝난 후 사용비트 0인 페이지 제거
  - ✓ 샘플링 기간 중 페이지폴트 발생시 새로운 페이지를 작업집합에 추가
  - ✓ 페이지폴트 증가시 샘플링 간격을 줄여 사용되지 않은 페이지 제거 가속화 : 샘플링 기간 설정 인자 *M*(샘플링 간격의 최소 기간), *L*(샘플링 간격의 최대 기간), *Q*(샘플링 간에 발생이 허용된 페이지폴트의 회수) 사용
- 샘플링 간격 설정 방법
  - ✓ 최종 샘플링으로부터 경과된 가상시간이 *L*에 이르면, 프로세스를 보류시키고 사용비트들을 살펴본다.
  - ✓ 가상시간 *L*이 지나기 전에 *Q*개의 페이지폴트가 발생했을 때,
    - 최종 샘플링 이후 경과된 가상시간이 *M*보다 작으면, 경과된 가상시간이 *M* 이 될 때까지 기다려 해당 프로세스를 보류시키고 사용비트들을 살펴본다.
    - 최종 샘플링 이후 경과된 가상시간이 *M* 보다 크거나 같으면, 프로세스를 보류시키고 사용비트들을 살펴본다.

클리닝 정책(Cleaning Policy)

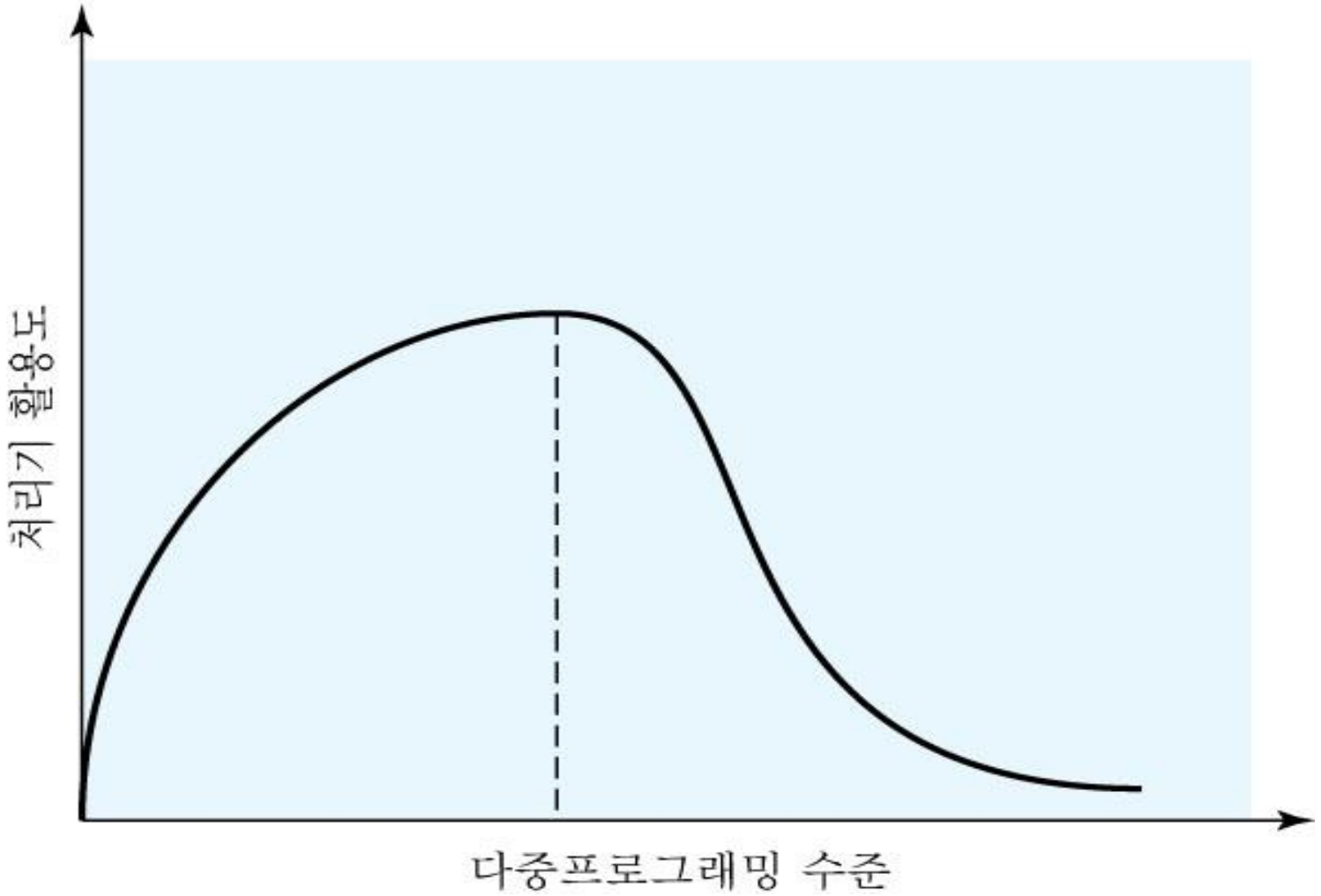
- 변경된 페이지들 중 어떤 것을 보조기억장치에 기록할 것인지 결정하는 정책(반입정책과 상반된 정책)
  - ✓ 요구클리닝(demand cleaning): 교체될 때 기록
  - ✓ 선클리닝(precleaning): 교체되기 전에 기록(일괄 기록 가능)

- 페이지 버퍼링과 접목하여 적용할 경우 효과적
  - ✓ 변경된 페이지가 교체 후보로 선택되면 변경리스트에 연결시킨 후, 주기적으로 일괄 기록한 후 비변경 리스트로 이동
  - ✓ 새로운 페이지 반입시, 비변경 리스트 상의 프레임을 할당하여 적재
  - ✓ 기록되기 이전에 참조될 경우 기록/적재 없이 재활용 가능
  - ✓ 페이지 교체시 비변경 리스트 상의 프레임을 이용하므로, 교체 당시 기존 페이지의 기록작업은 없음.

부하제어(Load Control)

- 주기억장치에 적재될 프로세스 수 결정
- 메모리에 적재된 프로세스의 수가 너무 적으면, 모든 프로세스가 블록 상태인 경우가 잦아 스와핑에 많은 시간 소요
- 메모리에 적재된 프로세스의 수가 너무 많으면, 각 프로세스의 적재집합을 구성하는 평균 페이지 수가 불충분해져 잦은 페이지폴트를 유발하게 되고, 궁극적으로 스레싱 발생

부하제어: 다중프로그래밍과 처리기활용도



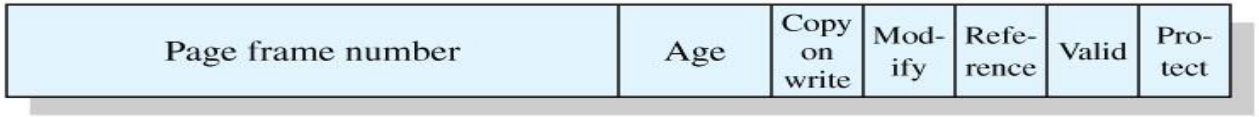
부하제어: 다중프로그래밍 수준 조절

- 작업집합(working set)/PFF
  - 적재집합이 충분히 갖추어진 활성프로세스만 수행을 허용하므로, 그 관리 과정에서 자동/동적으로 활성프로세스의 수 결정
- *L*=*S* 규범(criterion)
  - 폴트간 평균시간 *L*과 그 평균 서비스시간 *S*가 일치되게
- 50% criterion
  - 페이징 장치의 활용도가 50%로 유지되게
- 클록 정책의 변형
  - '바늘'의 스캔 속도가 너무 빠르면 다중프로그래밍 차수를 낮추고, 너무 느리면 높이는 방법 적용

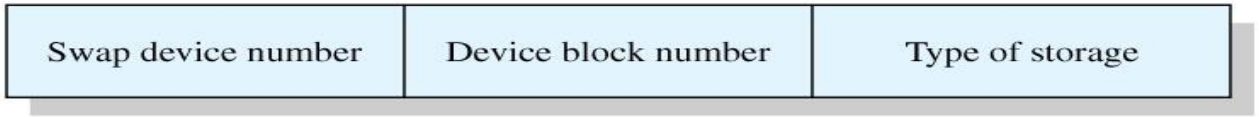
부하제어: 프로세스 보류

- 다중프로그래밍의 차수를 낮추기 위해 어떤 프로세스를 보류(suspension)시킬지 결정
- 보류 대상으로 고려할 만한 프로세스
  - ✓ 최저 우선순위 프로세스: 스케줄링 정책과 일관성 유지
  - ✓ 폴트 발생 프로세스: 작업집합 적재 가능성이 적어 보류시 입게 될 손해도 최소일 가능성(직접적 유익: 바로 블록될 프로세스를 블록시키는 것이므로, 페이지 교체나 입출력 비용 절감)
  - ✓ 가장 최근에 활성화된 프로세스: 작업집합 확보 가능성 최소
  - ✓ 최소 적재집합을 가진 프로세스: 미래의 재적재 비용을 최소화하지만, 지역성이 강한 프로세스에게 불리
  - ✓ 가장 큰 프로세스: 가장 많은 가용 프레임 확보, 추가 보류의 필요성 감소
  - ✓ 잔여 수행 윈도우가 가장 큰 프로세스: 최소 처리시간 우선(*shortest-processing-time-first*) 스케줄링과 일관성 유지

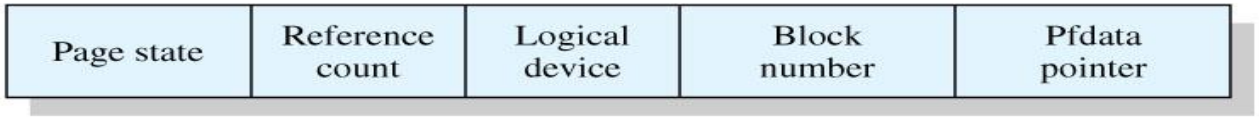
8.3 UNIX와 Solaris의 메모리 관리(1/3)



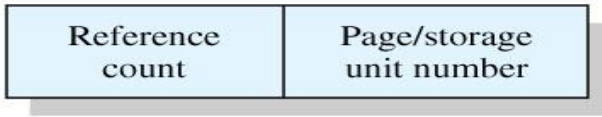
(a) 페이지테이블 항목



(b) 디스크 블록 디스크립터



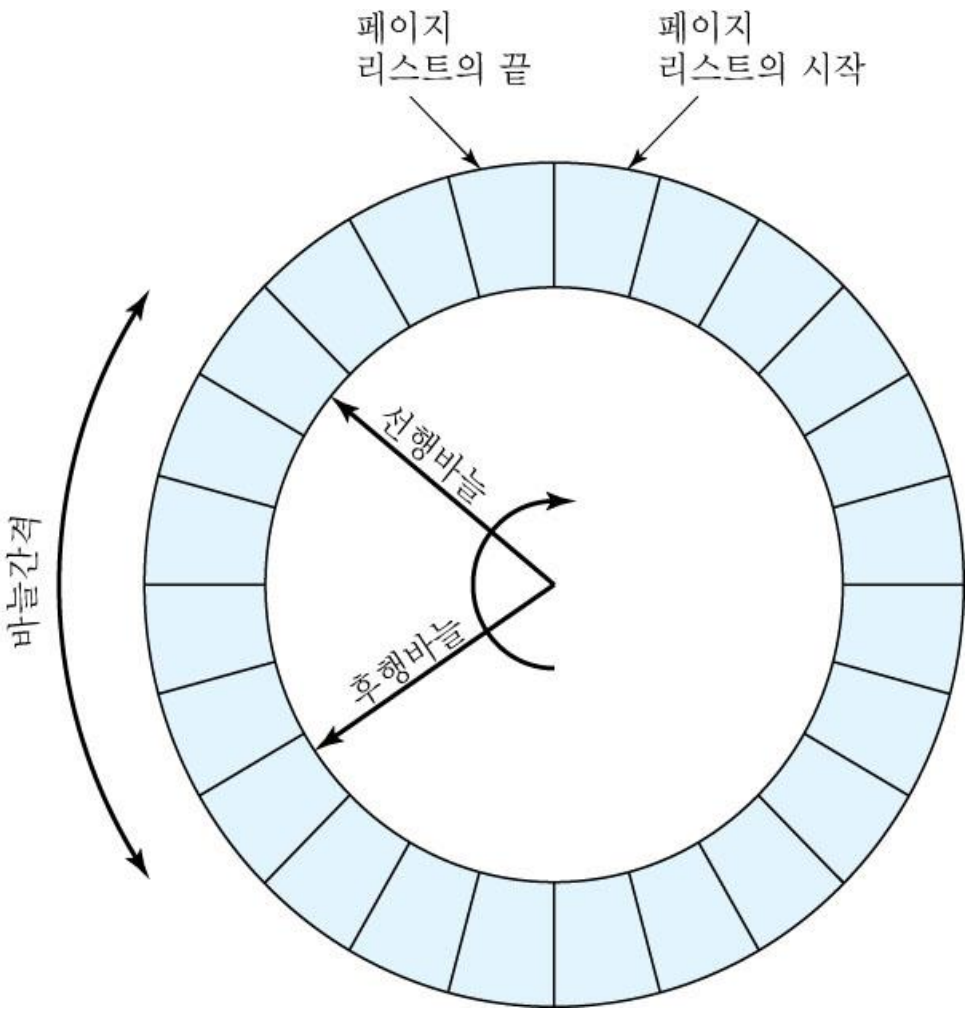
(c) 페이지프레임 데이터 테이블 항목



(d) 테이블 항목

UNIX와 Solaris의 메모리 관리(2/3)

- ‘두 바늘’ 클록 알고리즘
  - ✓ 처음 적재된 페이지의 참조 비트는 0, 참조될 때 1
  - ✓ ‘선행바늘(*fronthand*)’은 프레임들을 스캔하면서 각 페이지의 참조비트를 0으로 만들고, ‘어느 정도’의 시간이 지난 후, ‘후행바늘(*backhand*)’이 동일 페이지들을 스캔하면서 참조비트 값 조사
  - ✓ 참조비트 값이 1인 프레임은 교체 대상에서 제외하고, 0이면 해당 페이지를 반출될 페이지들의 리스트로 이동



UNIX와 Solaris의 메모리 관리(3/3)

- **Lazy buddy system algorithm**
  - **특정 크기의 블록에 대한 요구량이 시간상으로 완만하게 변함으로 분할과 통합을 늦춘다.**

D_i의 값을 0으로 초기화
연산 후에, D_i의 값은 다음과 같이 갱신된다.

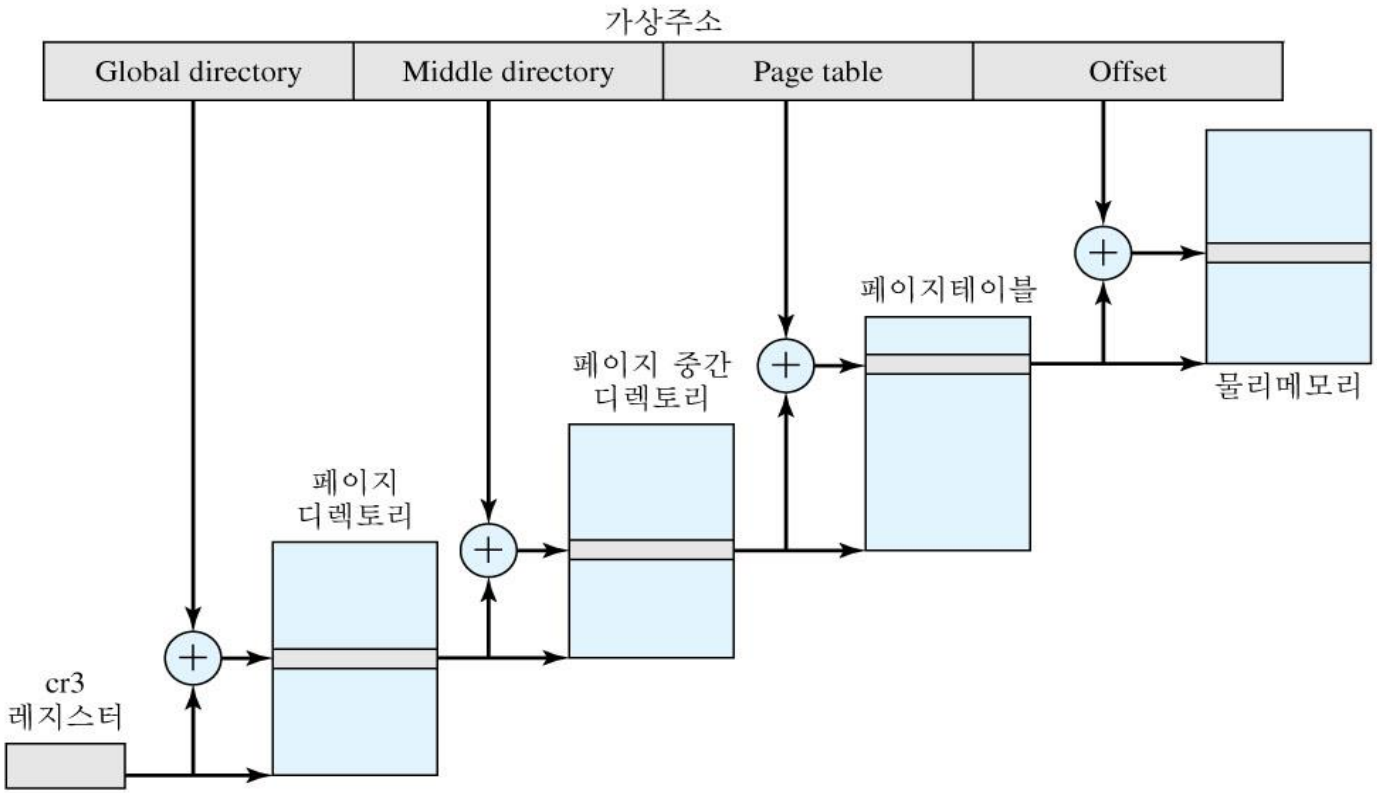
(I) 다음 연산이 블록 할당 요청일 경우:
  만약 가용 블록이 있다면, 할당할 한 블록을 선택한다.
    만약 선택된 블록이 지역적 가용 상태이면
      $D_i := D_i + 2$
    그렇지 않으면
      $D_i := D_i + 1$
  그렇지 않으면
    더 큰 블록을 두 개로 분할(재귀적 연산)하여 원하는 크기의 두 블록을 얻는다.
    하나는 할당하고, 다른 하나는 지역적 가용 상태로 표시한다.
    D_i는 변경되지 않는다(그러나 재귀 호출로 인해 다른 블록 크기에 대한 D 값이 변경될 수 있다).

(II) 다음 연산이 블록 반환 요청일 경우:
  $D_i \geq 2$일 경우
    지역적 가용 상태로 표시하고 지역적으로 반환한다.
    $D_i := D_i - 2$
  $D_i = 1$일 경우
    전역적 가용 상태로 표시하고 전역적으로 반환한다(가능한 경우 통합).
    $D_i := 0$
  $D_i = 0$일 경우
    전역적 가용 상태로 표시하고 전역적으로 반환한다(가능한 경우 통합).
    지역적 가용 상태의 2^i 크기 블록 하나를 선택하여, 전역적으로 반환한다(가능한 경우 통합).
    $D_i := 0$

그림 8.24 게으른 버디 시스템 알고리즘

8.4 LINUX 메모리 관리(1/2)

- LINUX 가상메모리 방식에서의 주소변환



LINUX 메모리 관리(2/2)

- 페이지 할당
  - ☞ 페이지 반입/반출 효율 높이기 위해 연속된 페이지 블록들을 연속된 프레임에 사상
- 페이지교체 알고리즘(*Least Frequently Used* 정책 범주)
  - ✓ 각 페이지에 대해 8비트 변수 age 설정
  - ✓ 참조될 때 age 값 증가, 주기적으로 전체 페이지 스캔하며 감소
  - ✓ age 값이 적을수록 보다 적합한 교체 후보
- 커널 메모리 할당
  - ✓ 페이지 할당: 버디 알고리즘을 이용해 페이지 배수 크기 할당
  - ✓ 슬랩 할당(slab allocation): 할당된 페이지 내에 작은 메모리 블록(*chunks*)을 수용하기 위해 도입. 버디 알고리즘과 유사하게 분할/통합하며, 블록 크기 별로 연결리스트 유지

애니메이션

- http://williamstallings.com/OS/Animations.html
  - ✓ Paging hardware with TLB
  - ✓ Two level paging
  - ✓ Inverted page tables
  - ✓ Virtual memory
  - ✓ Demand paging
  - ✓ Steps in handling a page fault
  - ✓ Page replacement
  - ✓ Second chance algorithm for page replacement
  - ✓ The working set model for page replacement
  - ✓ Considerations For Page Replacement in Virtual Memory Mode
  - ✓ Page replacement algorithms:
    - FIFO
    - Optimal
    - LRU
  - ✓ Virtual page reference strings

Summary

- Hardware and Control Structures
  - ✓ Paging
    - 2-level page table structure, inverted page table
    - TLB (Translation Lookaside Buffer)
  - ✓ Segmentation
  - ✓ Combined Paging and Segmentation
  - ✓ Protection and Sharing
- Operating System Software
  - ✓ fetch, placement, replacement
  - ✓ OPT, LRU, FIFO, clock, LFU, …
  - ✓ Resident set management: working set
- Case study
  - ✓ Unix and Solaris, Linux, Windows Memory Management