운 영 체 제

4장 가상 메모리 관리

가상 메모리 개요(1)

4장 가상 메모리 관리

4.1 개요

4.2 페이징(paging)

4.3 세그먼테이션(segmentation)

4.4 세그먼트/페이징 혼용 기법

4.5 페이지 교체 알고리즘

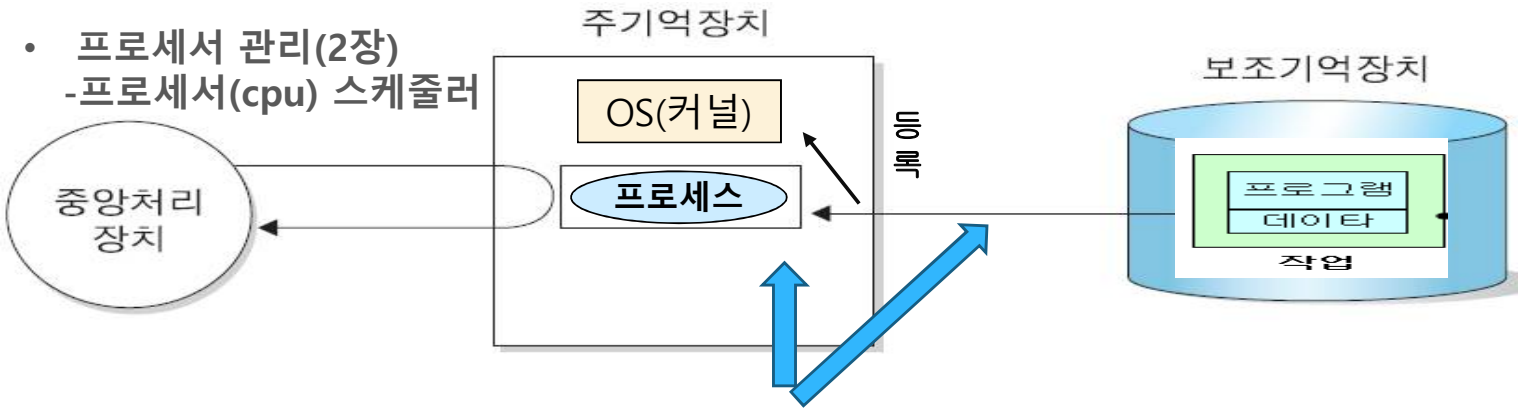
4.6 스레싱(thrashong)

학습 내용

4장 가상 메모리 관리

- 4.1 개요
  - -가상기억장치 개념
  - -주소 사상 기법(주소 변환)
  - -블럭 사상을 통한 가상 주소 변환(동적 주소 변환)

4장 가상 메모리의 개요



- 프로세서 관리(2장)
  -프로세서(cpu) 스케줄러
- **주기억장치의 관리(3장)**
  - **-Job 스케줄링**
  - **-주소 바인딩**
  - **-주기억장치 관리 기법**
    - -인출 기법(요구 인출 기법,예상인출 기법)
    - -배치 기법(최초 적합(first-fit), 최적 적합(best-fit) 및 최악 적합(worst-fit))
    - -교체 기법
  - **-주기억장치 할당 기법**
    - **-연속 할당 기법(단일사용자 연속, 고정 분할, 가변 분할)**
- **가상 메모리 관리(4장)**
  - **-분산 할당 기법(페이징 기법, 세그먼테이션 기법)**
  - -페이지 교체 기법

기억장치 관리의 발전

기억장치 관리의 발전

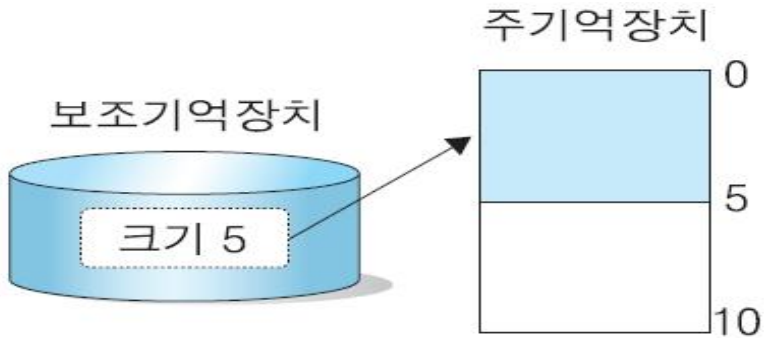
연속 적재 기법 (Contiguous Loading)		분산 적재 기법 (Scatter Loading)						
실기억 공간 (Real Memory)							가상 기억 공간 (Virtual Memory)	
단일 사용자 (Single User)		다중 프로그래밍 (Multi-Programming)						
오버레이 Overlay	교체 Swapping	고정 분할 Fixed Partition	동적 분할 Dynamic Partition	페이징 Paging	세그먼테이션 Segmentation	페이지/세그먼트의 혼합 형태	요구 페이징	요구 세그먼테이션

오버레이 (overlay)기법
=> 프로그램 크기가 클때
스와핑(Swapping)기법
⇒시스템 성능 향상
⇒다중 프로그래밍 지원
⇒페이징 기법으로 발전

가상 메모리의 소개

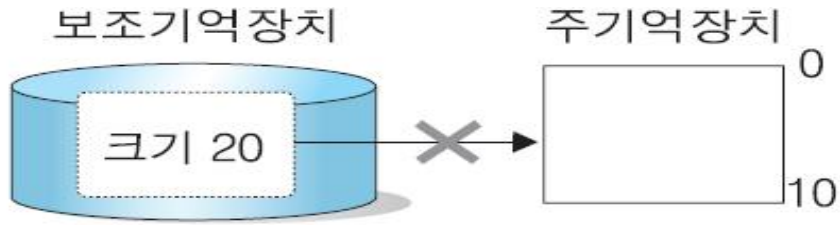
▶ 가상 메모리

- 크기 5의 프로그램이 주기억장치로 : 진입 : 문제없음



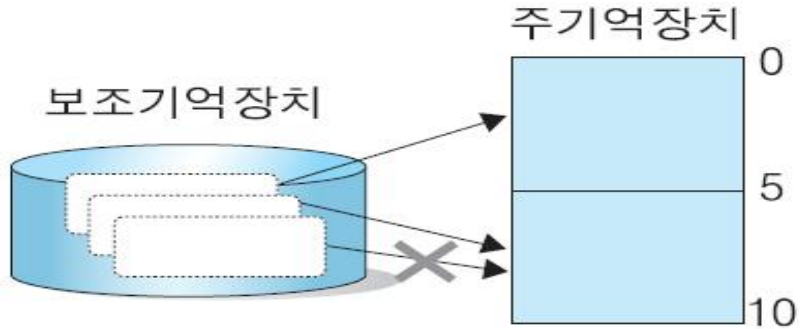
[그림] 크기 5의 프로그램이 주기억장치로 진입

- 주기억장치보다 프로그램의 크기가 큰 경우 : 문제 발생



[그림] 주기억장치보다 프로그램의 크기가 큰 경우

- 실행하려는 전체 프로그램의 크기가 주기억장치보다 클 때 : 문제 발생

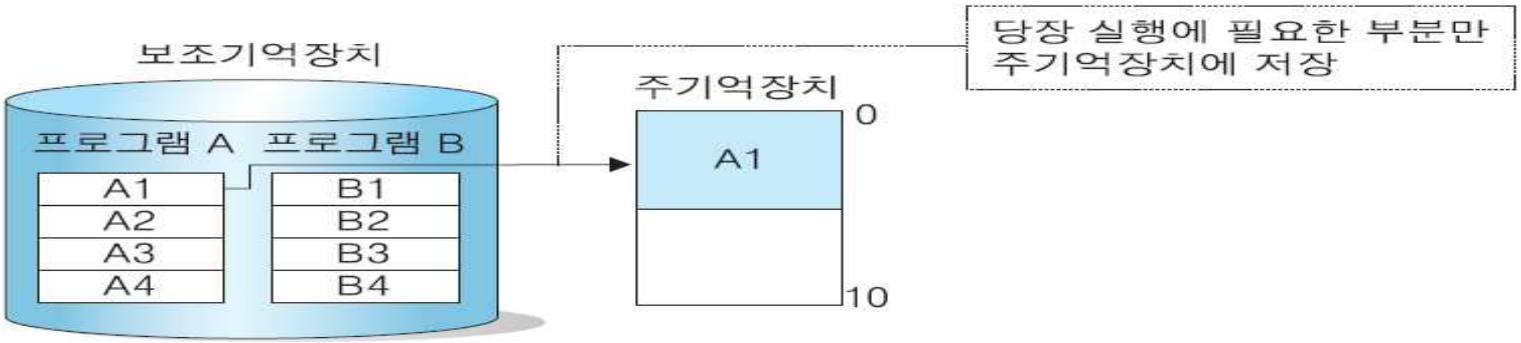


[그림] 실행하려는 전체 프로그램의 크기가주기억장치보다 클 때

가상 메모리의 소개

- 실행하려는 전체 프로그램의 크기가 주기억장치보다 클 때
  - 당장 실행에 필요한 부분만 주기억장치에 저장하고, 당장 필요하지 않은 나머지 부분은 보조기억장치에 넣어 두고 실행
  - 사용자 입장에서는 실제 주기억장치보다 큰 주기억장치를 가지고 있는 것처럼 느끼게 됨

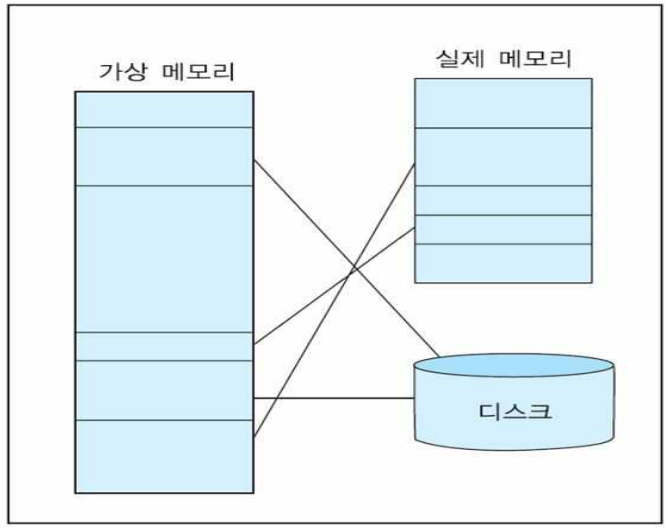
  - 당장 실행될 부분만 주기억장치에 저장한다.
    - ✓ 블럭 : 프로그램을 일정한 크기로 나눈 단위
  - 블록(Block)의 종류
    - 페이징 기법 :사용자 프로그램을 같은 크기의 블럭들로 분할하는 경우
    - 세그먼테이션 기법:사용자 프로그램을 서로 다른 크기의 논리적인 단위로 분할하는 경우

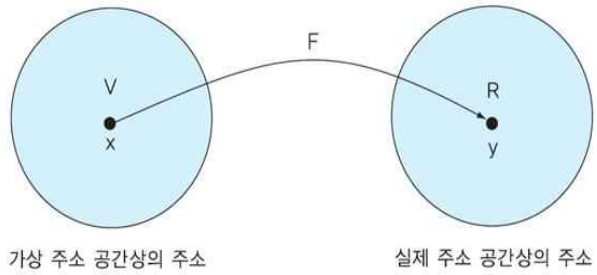


[그림] 당장 실행될 부분만 주기억장치에 저장

가상기억장치 개념

- 메모리에 프로그램의 일부만 적재 수행 시 장점
  - 수행 중인 프로세스를 활동 영역을 메인 메모리에 유지하면서 필요할 때는 디스크와 메모리 사이에 프로세스 코드와 데이터 저장, 다시 자동으로 전송하는 (스왑 인, 스왑 아웃) 과정을 거쳐 프로세스를 재할당, 디스크에 저장된 주소 공간은 캐시로 처리하여 메인 메모리 효율적 사용 가능
  - 메인 메모리의 제한된 용량과 중첩 사용 문제 해결
  - **중첩을 고려하여 프로그래밍할 필요가 없으므로 프로그래밍 용이**
  - **프로세서의 이용률과 처리율 향상(응답시간이나 반환시간은 늦음)**
- 메모리에 프로그램의 일부만 적재 수행 시 문제점
  - 메모리와 디스크 사이에 이동량 증가, **스와핑 공간 필요, 페이지 적재와 복귀할 페이징 알고리즘 결정해야 함**
  - 요구된 프로세스의 페이지가 없을 때(**페이지 부재**)처리하는 방안 등
- 문제점 해결 방법
  - **실행 중인 프로세스가 참조하는 주소와 메인 메모리에서 사용하는 물리적 주소 분리**
    - **가상 주소를 물리적 주소로 변환하는 과정 : 매핑(동적 주소 변환)**
    - 실행 중인 프로세스가 참조하는 주소 : 가상 주소(논리적 주소, 프로그램 주소)
    - **매핑은 가능한 빨리 수행**. 그렇지 않으면 시스템 성능 떨어지고 가상 메모리 사용 효과 낮아짐
    - 매핑은 변환 함수로 표현



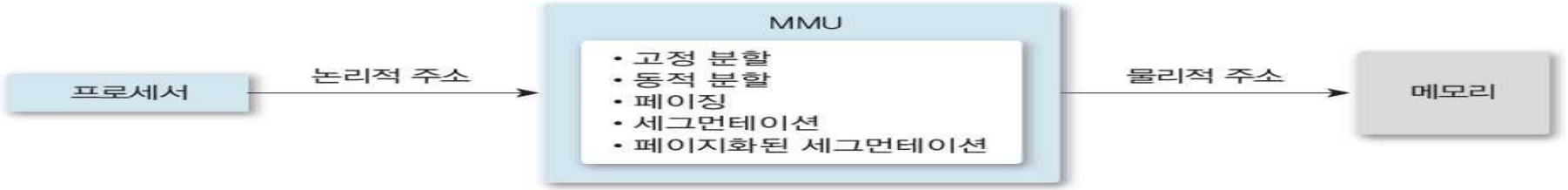


가상기억장치 개념

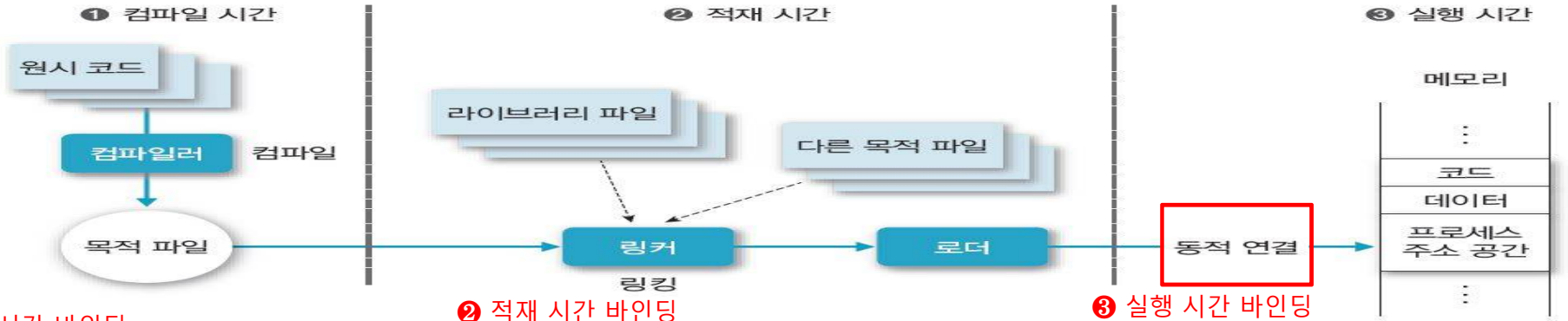
- 가상기억장치(**virtual storage**)
  - 개념
    - 사용자 프로그램을 여러 블럭들로 분할
    - 가상 기억장치는 보조 기억장치의 일부를 주기억장치처럼 사용하는 것으로 용량이 작은 주기억장치를 마치 큰 용량을 가진 것처럼 사용하는 기법
    - 실행 시 필요한 블럭들 만 **비연속적으로** 주기억장치에 적재시킴
    - 가상 기억 장치에서 프로그램이 갖는 연속적인 주소가 실제 기억 장치에서는 연속적일 필요가 없다.
    - 가상 기억장치에 저장된 프로그램을 실행하려면 가상 기억장치의 주소를 주기억장치의 주소로 바꾸는 주소 변환 작업이 필요하다.**(동적 주소 변환)**
    - 블록 단위로 나누어 사용하므로 연속 할당 방식에서 발생 할 수 있는 **단편화를 해결할 수 있다.**
    - **주기억장치의 이용률과 다중프로그래밍의 효율을 높일 수 있다.**
  - 블록의 종류에 따라
    - 페이징 시스템(paging system)
      - 사용자 프로그램을 **같은 크기의 블럭들로 분할**하는 경우
    - 세그먼테이션 시스템(segmentation system)
      - 사용자 프로그램을 **서로 다른 크기의 논리적인 단위로 분할**하는 경우
    - 페이징 시스템과 세그먼테이션 시스템의 합성 기법

주소 사상 기법(주소 변환)

- 매핑**mapping (3**장 기억장치 관리와 **2**장 시스템소프트웨어 종류 참고)
  - 논리적 주소와 물리적 주소의 연결, 매핑 시켜 주는 작업(주소 바인딩 (address binding))



메모리관리장치의 주소 변환(논리적 주소 → 물리적 주소)



❶ 컴파일 시간 바인딩
프로세스가 메모리에 적재될 위치를 컴파일 과정에서 알 수 있다면 컴파일러는 물리적 주소 생성 가능

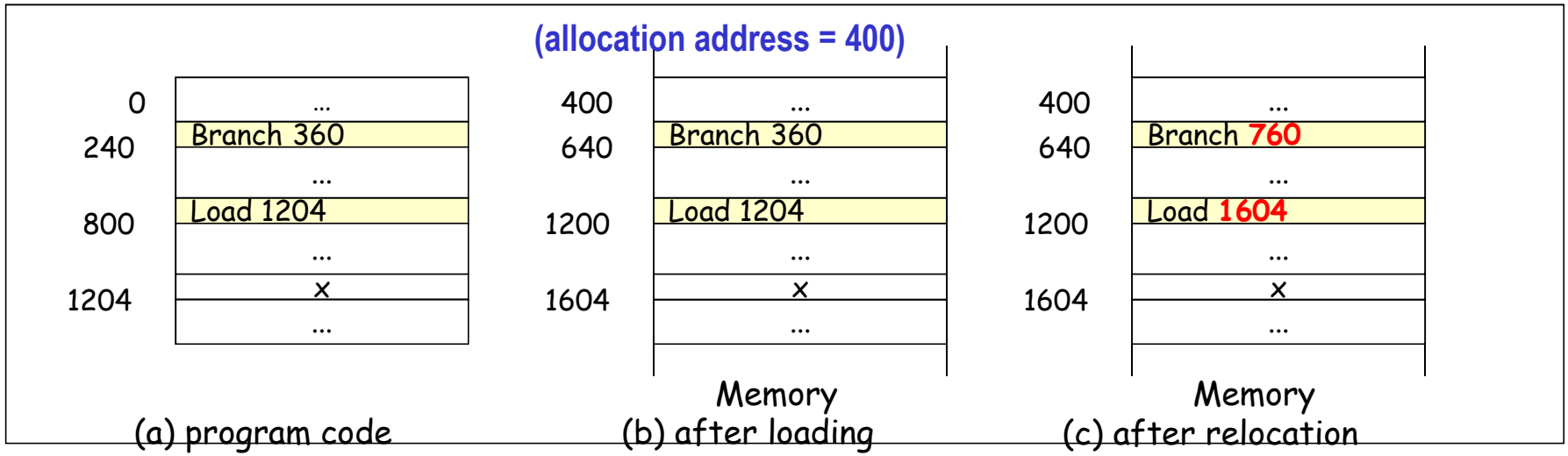
❷ 적재 시간 바인딩
프로세스를 메모리의 어디에 적재해야 할지 컴파일 과정에 알려 주지 않으면 컴파일러는 대체 가능한 상대 주소 생성. 시작 주소가 변하면 단지 변화된 값을 반영하 려고 사용자 코드를 재적재(정적 대치)

❸ 실행 시간 바인딩
프로세스 실행 도중 메모리의 한 세그먼트에서 다른 세그먼트로 이동 한다면 바인딩은 수행 시간까지 연기(지연). 이런 주소 체계는 기본 및 경계(한계) 레지스터 등 특수 하드웨어의 지원 필요. 현재 범용 운영체제 대부분 실행 시간에 바인딩 방법 사용 .

주소 사상 기법(주소 변환)

- 프로그램 전체를 연속으로 적재하는 경우(적재 시간 바인딩)
  - 재배치 과정 한 번 거침
    - 상대 주소(relative address)
      - 프로그램의 시작 지점을 0번지로 가정하는 주소임
    - 재배치(relocation)
      - 주기억장치 할당 후 할당 주소(allocation address)에 따라 상대 주소들을 조정하는 작업

프로그램 재배치 과정(프로그램 전체가 연속적으로 적재된 경우)



주소 사상 기법(주소 변환)

- 프로그램을 분할하여 그 일부만을 비연속 적재하는 경우(실행 시간 바인딩)
  - 가상메모리에서 사용하는 주소를 가상 주소(virtual address)
  - 주기억장치상에서 이용할 수 있는 주소를 실제 주소(real address)
  - 주소 사상 함수(address mapping function) => 동적 주소 변환(Dynamin address translation)
    - 가상 주소를 실제 주소로 변환하는 작업을 실행 중에 동적으로 수행하기 위한 기법
    - 프로세스가 수행될 때 가상 주소를 실제 주소로 변환하는 대표적인 메커니즘

정의 : 주소 사상 함수

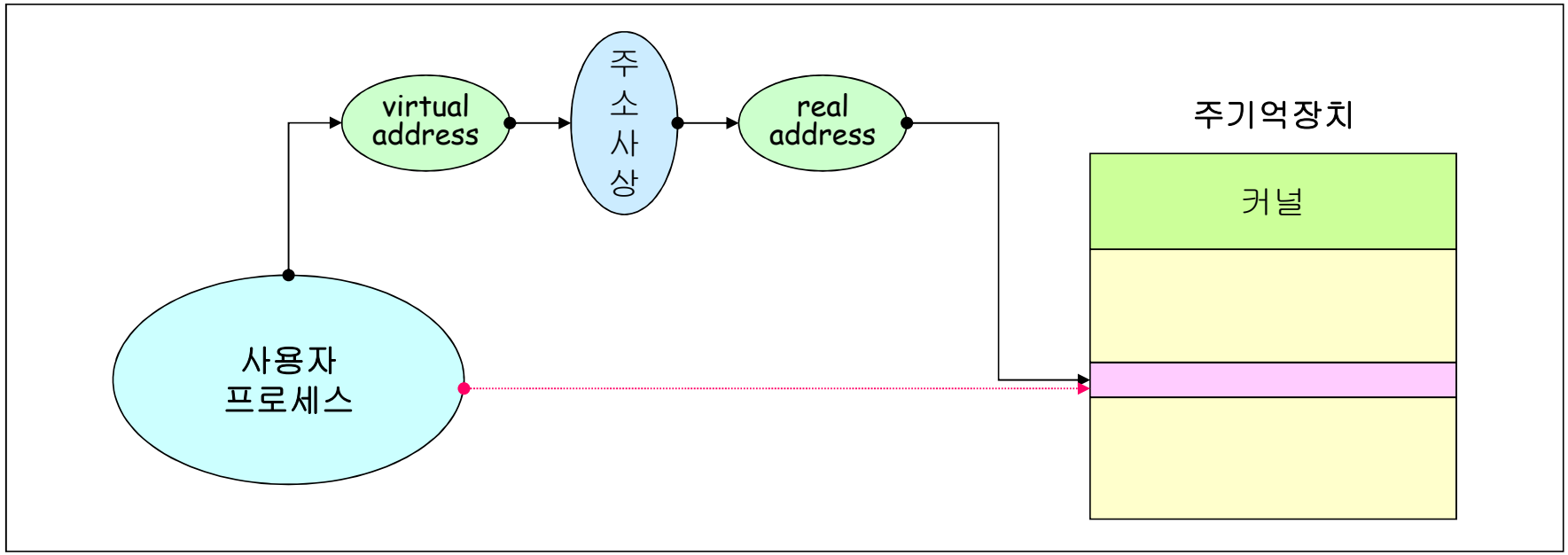
- 프로그램의 실행 중 발생되는 가상 주소 v를 주기억장치 상의 실제 주소 r로 변환하는 함수
- 주소 사상 함수 f() 정의

f : V → R
r = f(v)

- V : 가상 주소 공간(virtual address space)
- R : 실제 주소 공간(real address space)

주소 사상 기법(주소 변환)

주소 사상 과정 : 동적 주소 변환(Dynamin address translation)



- 인위적 연속성(artificial contiguity)
  - 사용자 또는 프로세스 입장에서 실행 프로그램 전체가 주기억장치에 적재되어 있는 것으로 보게 하는 개념
  - 가상 주소 공간상의 연속된 주소들은 실기억 공간에서도 반드시 연속적일 필요는 없다는 특성

블럭 사상을 통한 가상 주소 변환

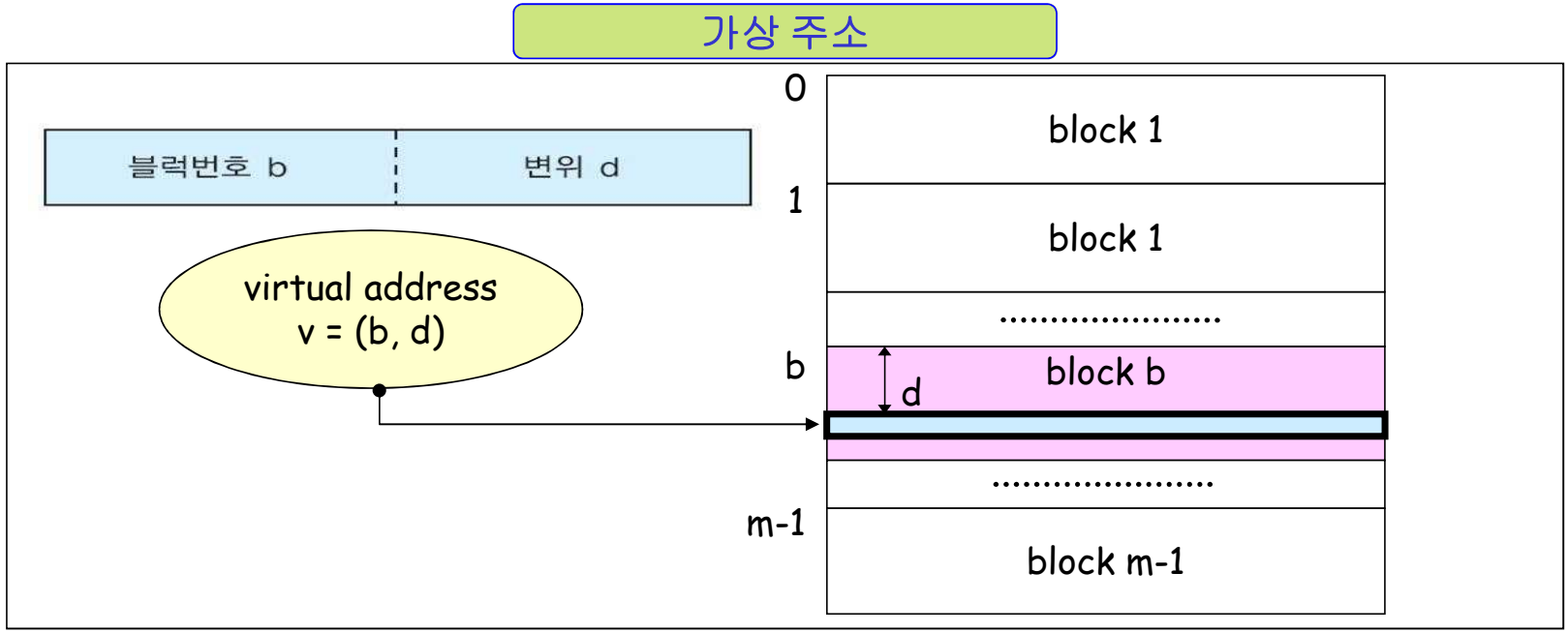
- **블럭 사상(block mapping) 개념**
  - 블럭 사상(block mapping)
    - 프로그램을 블럭 단위로 분할하고 이렇게 분할된 블럭 단위로 주소 사상 정보를 기록하여 사용하는 기법
  - 블럭 사상 테이블(block map table)
    - 각 블럭의 가상 주소와 이에 대응되는 실제 주소 저장

블럭 크기에 따른 주소 사상 오버헤드

블럭의 크기를 크게 하는 경우	사상 테이블에 적재될 사상 정보의 양이 작아짐
	주소 사상에 필요한 시간이 빨라짐
	각 블럭이 차지하는 주기억장치 공간의 양이 많아짐
	필요없는 부분이 주기억장치에 적재될 가능성이 많아짐
	각 블럭의 전송 시간이 길어짐

블럭 사상을 통한 가상 주소 변환

- 블럭 사상 기법
  - 가상 주소 v = (b, d)
    - b = 블럭 번호
    - d = 해당 블럭 내에서의 변위(displacement, offset)



블럭 사상을 통한 가상 주소 변환

- 블럭 사상 테이블(block map table)
  - 주소 사상 정보를 기록, 유지하기 위하여 사용되는 테이블
    - 시스템에 존재하는 각 프로세스마다 하나씩 존재
    - 커널 영역에서 관리

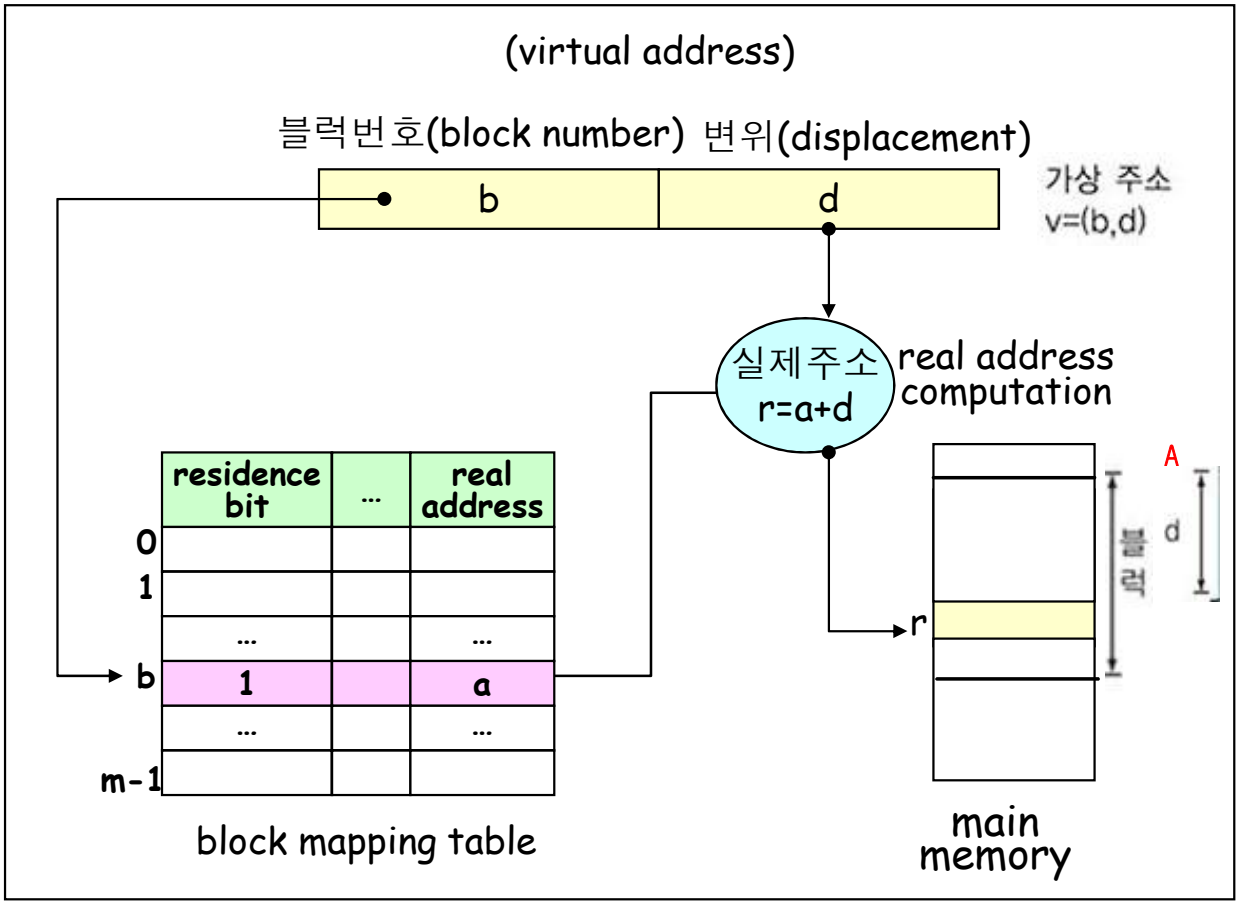
블럭 사상 테이블의 구조

block number	residence bit	...	real address
0 1 2 ...		. . .	
b	1	.....	a
... m-1		:	

- 블록 번호(block number)
- 존재 비트 (**residence bit**): 해당 블럭이 주기억장치에 적재되어 있는지의 여부
- 실제 주소(**real address**) : 해당 블럭이 적재된 주기억장치 주소

블럭 사상을 통한 가상 주소 변환

블럭 사상 과정



⑴ 해당 프로세스의 블럭 사상 테이블에 접근

⑵ 블럭 사상 테이블에서 블럭 b에 대한 엔트리를 찾음

⑶ 찾아진 엔트리의 존재 비트 검사

① 존재 비트가 0 인 경우

디스크로부터 해당 블럭을 주기억장치로 적재

블럭 사상 케이블 갱신 후 ⑶ -② 단계 수행

② 존재 비트가 1인 경우

해당 엔트리에서 실제 주소 필드의 값 a 인출

⑷ 인출된 값 a와 가상 주소의 변위 d를 더하여 실제 주소 r 형성 (r = a + d)

⑸ 실제 주소 r로 주기억장치에 접근

학습 내용

- **4.2 페이징**
  - -페이징 시스템 개요
  - -페이징 시스템 특성
  - -주소 변환
  - -주소 사상 기법(직접 사상기법)
  - -주소 사상 기법(연관 사상기법)
  - -주소 사상 기법(혼합 연관/직접 사상 기법)
  - -페이징 시스템의 공유
  - -페이지 인출 기법
  - -페이지 양도

가상 메모리의 개념

- 개념
  - 사용자 프로그램을 여러 블럭들로 분할
  - 가상 기억장치는 보조 기억장치의 일부를 주기억장치처럼 사용하는 것으로 용량이 작은 주기억장치를 마치 큰 용량을 가진 것처럼 사용하는 기법
  - 실행 시 필요한 블럭들만 비연속적으로 주기억장치에 적재시킴
  - 가상 기억장치에 저장된 프로그램을 실행하려면 가상 기억장치의 주소를 주기억장치의 주소로 바꾸는 주소 변환 작업이 필요하다.(동적 주소 변환)
  - 블록의 종류에 따라
    - 페이징 시스템(paging system)
      - 사용자 프로그램을 같은 크기의 블럭들로 분할하는 경우
    - 세그먼테이션 시스템(segmentation system)
      - 사용자 프로그램을 서로 다른 크기의 논리적인 단위로 분할하는 경우
    - 페이징 시스템과 세그먼테이션 시스템의 합성 기법

[그림] 당장 실행될 부분만 주기억장치에 저장

페이징 시스템

- 페이징 시스템(paging system) 개요
  - 정의
    - 프로그램을 블럭 단위로 분할할 때
      같은 크기의 블럭들로 분할하는 시스템
  - 용어

❑ 페이지(page)

- 실행 프로그램의 분할된 블록
- 일정한 크기의 블록

❑ 페이지 프레임(page frame), 페이지 틀

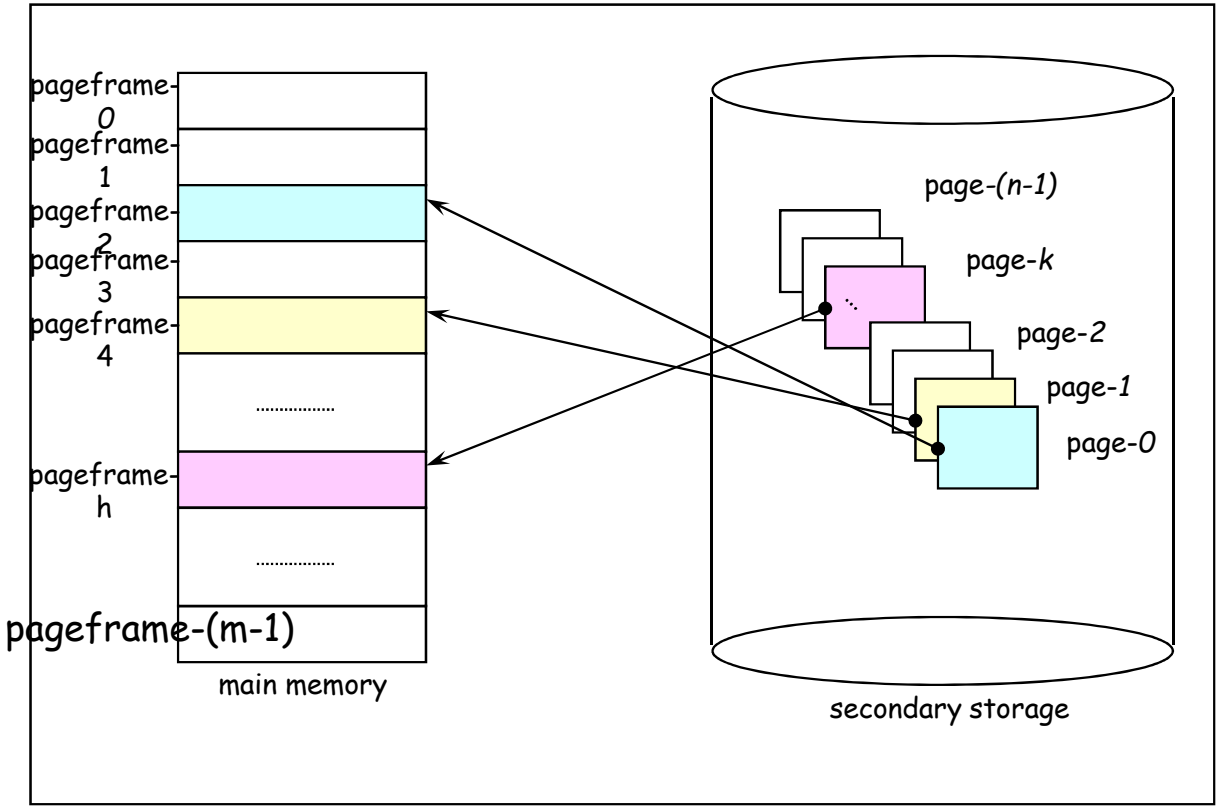
- 주기억장치의 분할 영역
- 페이지 크기(page size)와 같은 크기로 분할됨

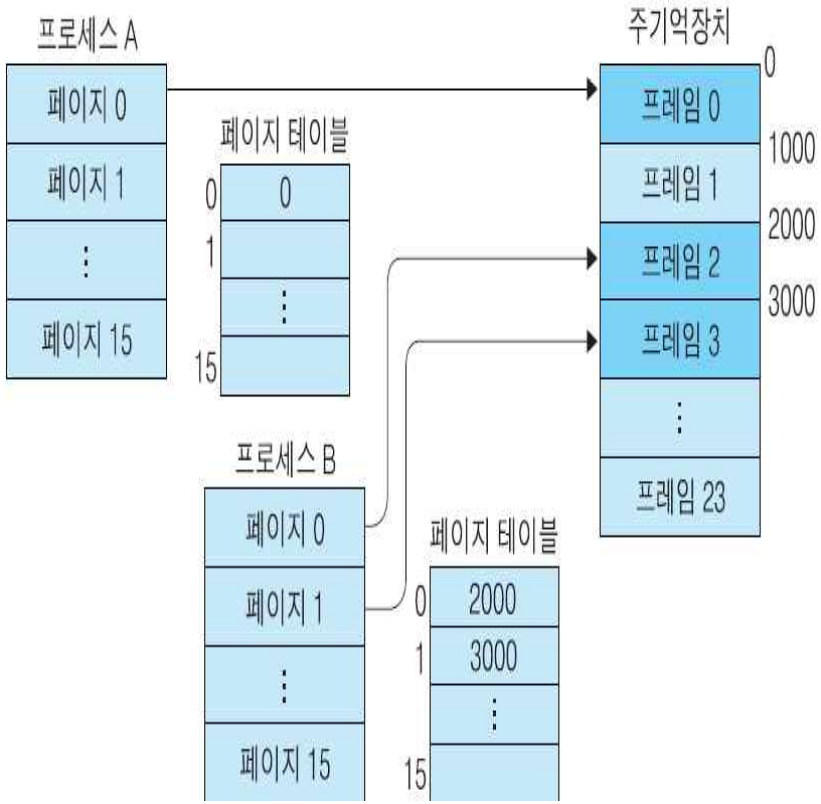
- 페이징 시스템에서의 가상 주소는 순서쌍 v=(p, d)로 표현

- 주소 변환

페이징 시스템

페이징 시스템





페이징 시스템

- **페이징 시스템 특성**
  - 프로그램 분할 시 논리적인 구분을 하지 않음
  - 단순(simple)하고 효율적(efficient)이며, 많은 운영체제에서 사용됨
  - 프로그램 공유(sharing)나 보호(protection)에 있어서 복잡한 문제 발생 가능
- **주소 변환**
  - 페이징 시스템에서의 가상 주소 v = (p, d)
    - p = 페이지 번호
    - d = 해당 페이지 내에서의 변위
  - 주소 사상
    - 페이지 사상 테이블(PMT : Page MappingTable) 이용하여 가상 주소를 실제 주소로 변환
  - 주소 사상 기법
    - 직접 사상(direct mapping)
    - 연관 사상(associative mapping)
    - 혼합 사상(mixed direct/associative mapping)

페이징 시스템

페이지 사상 테이블 (PMT : Page Mapping Table)

page number	residence bit	secondary storage address	other fields	page frame number
0	1	S_0	. . .	2
1	1	S_1	. . .	4
2	0	S_2	. . .	-
. . .	. . .	...	. . .	. . .
k	1	S_k	. . .	h
. . .	. . .	...	. . .	. . .
n-1	0	S_{n-1}	. . .	-

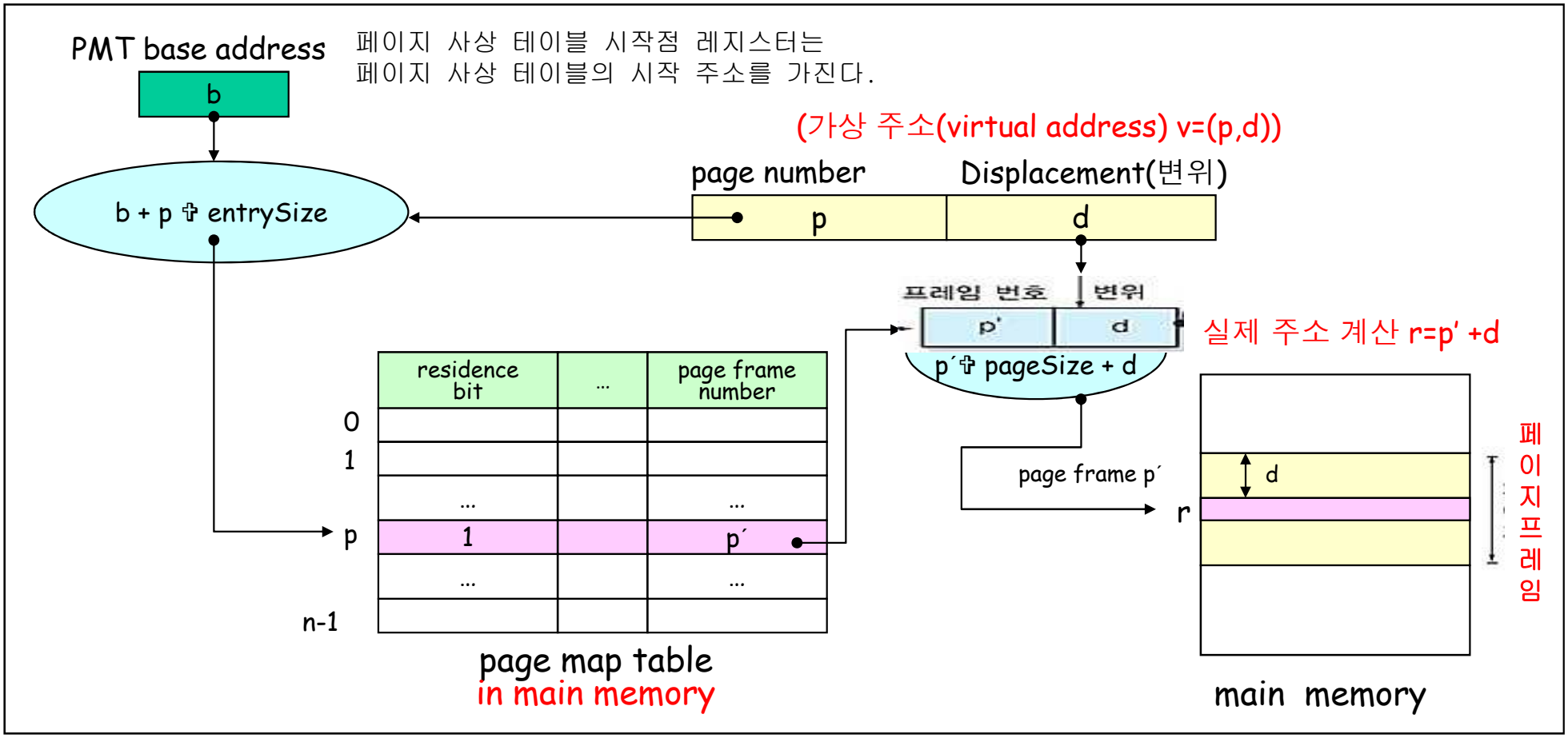
- 존재 비트 : 해당 페이지가 주기억장치에 적재되어 있는지의 여부
- 보조기억장치 주소 : 해당 페이지의 보조기억장치 주소
- 페이지 프레임 번호 : 해당 페이지가 적재되어 있는 페이지 프레임 번호

주소 사상 기법

- 직접 사상(direct mapping) 기법
  - 블럭 사상 기법과 유사함
  - 가정
    - PMT가 주기억장치의 커널 공간 내에 존재함
    - PMT의 엔트리 크기 = entrySize
    - 한 페이지의 크기 = pageSize

주소 사상 기법(직접 사상기법)

페이징 시스템의 직접 사상 기법



주소 사상 기법(직접 사상 기법)

페이징 시스템의 직접 사상 알고리즘

⑴ 해당 프로세스의 PMT가 저장되어 있는 주소 b에 접근
⑵ 해당 PMT에서 페이지 p에 대한 엔트리 찾음
→ 페이지 p의 엔트리 위치 = b + p * entrySize
⑶ 찾아진 엔트리의 존재 비트 검사
① 존재 비트가 0 인 경우 (page fault)
디스크로부터 해당 페이지를 주기억장치로 적재
PMT를 갱신한 후 ⑶ -② 단계 수행
② 존재 비트가 1인 경우
해당 엔트리에서 페이지 프레임 번호 p'를 인출
⑷ 인출된 페이지 프레임 번호 p'와 가상 주소의 변위 d를 사용하여
실 주소 r 형성 (r = p' * pageSize + d)
⑸ 실 주소 r로 주기억장치에 접근

주소 사상 기법(직접 사상기법)

- 직접 사상 기법의 단점
  - 주기억장치 접근 회수 2배로 증가
  - 성능 저하 초래
- 직접 사상 기법의 단점 해결
  - PMT를 캐쉬 기억장치에 적재하는 기법
    - 보다 빠른 주소 변환을 위하여, 속도가 매우 빠른 고속 캐시 기억장치(high speed cache memory)를 이용하여 직접 사상의 페이지 사상 테이블을 구현
  - 연관 사상 기법

- 페이지 부재(page fault)
  - 프로세스가 실행 중 접근하는 페이지가 주기억장치에 적재되어 있지 않는 경우 발생
  - 프로세스의 계속 진행이 불가능함
  - 프로세스의 실행 중에 발생하는 페이지 부재 횟수를 줄이는 기법 필요

주소 사상 기법(연관 사상기법)

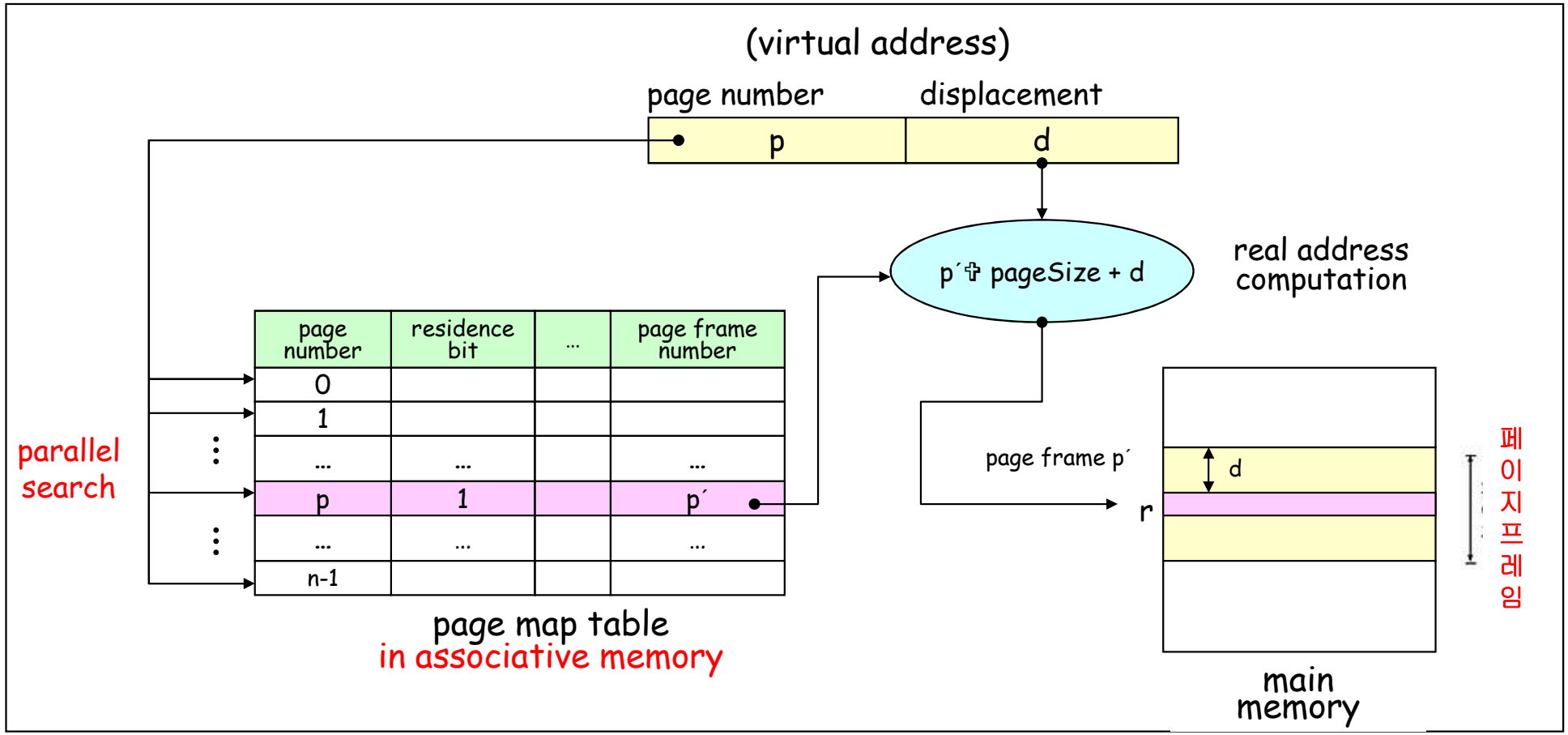
- 연관 사상(**associative mapping**) 기법
  - PMT를 연관 기억장치에 적재하여 사용하는 기법
  - 저장된 값을 이용하여 데이터를 접근
  - 직접 사상 기법의 오버헤드 줄이기 위한 기법
  - 직접 사상 기법의 메커니즘과 유사함

  - 연관 기억장치(associative memory)
    - 내용 주소화 메모리 (content addressable memory)
    - 주소의 개념 없이 지정된 내용으로 데이터에 접근할 수 있도록 하드웨어적으로 구현된 기억장치(고가)

  - 연관 사상 기법의 특징
    - 필요한 내용을 갖는 데이터에 대해 병렬 탐색 가능
    - PMT를 저장할 연관 기억장치를 설치하는 하드웨어 비용 큼(고가)

주소 사상 기법(연관 사상기법)

페이징 시스템의 연관 사상 기법



주소 사상 기법(혼합 연관/직접 사상 기법)

- **혼합 연관/직접 사상(combined direct/associative mapping)** 기법
  - 직접 사상 기법과 연관 사상 기법을 같이 사용
    - 하드웨어 비용 줄이면서 연관 사상 기법의 장점 취하는 기법
  - 소규모의 연관 기억장치 사용(고가이기 때문에)
    - PMT의 전체 내용 : 주기억장치의 커널 공간에 적재
    - PMT의 일부 내용 : 연관 기억장치 내 적재
      - 최근에 사용된 페이지에 대한 엔트리들만 적재
  - 가장 최근에 참조된 페이지는 조만간 다시 사용되기 쉽다는 사실을 이용
  - 연관기억장치에는 페이지 사상 테이블의 전체 항목 중 가장 최근에 참조된 일부 페이지 항목들만을 수용
  - 우선 가상 주소를 연관 페이지 사상 테이블에서 찾는다. 없으면 직접 사상 테이블에서 찾는다.

- **프로그램의 지역성(locality)**
  - 프로그램중에서
    한 번 접근된 부분은 계속해서 다시 접근될 가능성 많고,
    한 번 접근된 부분의 인접부분이 앞으로 접근될 가능성이 많다는 성질

페이징 시스템

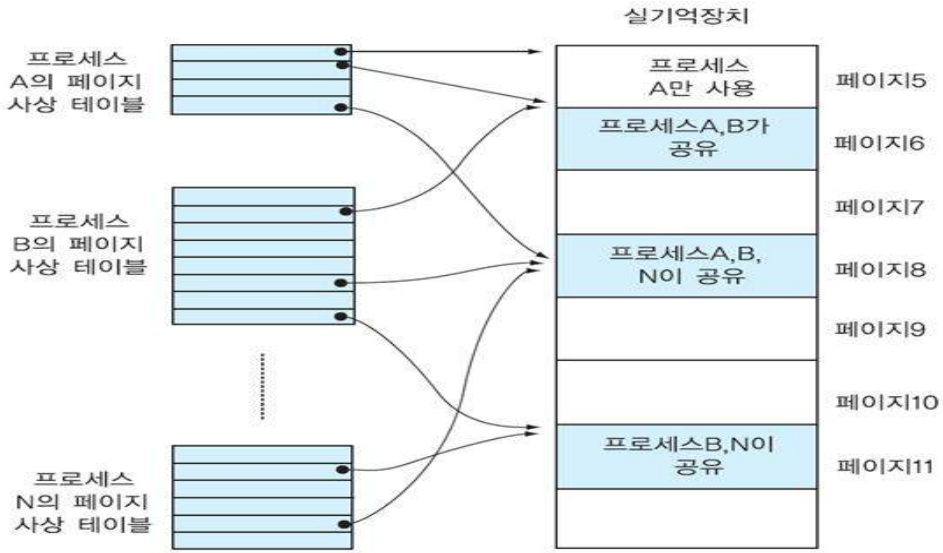
- 페이징 시스템 특성
  - 프로그램 분할 시 논리적인 구분을 하지 않음
  - 단순(simple)하고 효율적(efficient)이며, 많은 운영체제에서 사용됨
  - 프로그램 공유(sharing)나 보호(protection)에 있어서 복잡한 문제 발생 가능

- 페이지 크기
  - 페이지의 크기를 결정함에 있어 고려되어야 할 내용
    - 페이지 크기가 작으면 작을수록 페이지 테이블의 크기가 증가하여 기억 공간이 낭비

    ➔테이블 단편화(table fragmentation)
    - 페이지가 크게 되면 참조되지 않을 많은 정보들까지 주기억장치로 옮겨지게 되어 기억공간의 낭비를 초래
    - 프로그램 실행 중 입출력 전송의 횟수를 줄이기 위해서는 페이지 크기가 클수록 효과적

페이징 시스템의 공유

- 페이징 시스템의 공유
  - 공유(share)가 가능한 페이지를 가능한 한 공유
  - 공유 페이지의 코드를 실행하는 프로세스마다 자신의 가상 주소 공간 내에서의 분기 주소가 다르기 때문에 오류가 발생 할 수가 있다.
  - 프로그램 공유(sharing)에 있어서 복잡한 문제 발생 가능



페이징 시스템에서의 공유

페이징 시스템

- 페이지 인출 기법
  - 요구 페이징(demand paging) 기법
    - 실행 중인 프로세스에 의하여 명백히 참조되는 프로세스만이 보조기억장치로부터 주기억장치로
  - 예상 페이징(anticipatory paging) 기법
    - 운영체제가 예측하여 주기억장치에 여유가 있을 때 이 페이지들을 미리 적재

- 페이지 양도(page release)
  - 더 이상 필요로 하지 않는 특정한 페이지
  - 가장 효율적인 페이지 양도 방법은 컴파일러나 운영체제가 자동으로 페이지 제거

학습 내용

- 4.3 세그먼테이션
  - 세그먼테이션 시스템 개요
  - 주소 사상
  - 주소 사상 기법(직접 사상기법)
  - 세그먼테이션 시스템의 직접 사상 기법
  - 세그먼테이션 시스템의 공유
- 4.4 세그먼트/페이징 혼용 기법
  - 페이징/세그먼테이션 혼합 기법 개요
  - 주소 사상 기법
  - 페이징/세그먼테이션 혼합 기법에서의 직접 사상 기법

기억장치 관리의 발전

기억장치 관리의 발전

연속 적재 기법 (Contiguous Loading)		분산 적재 기법 (Scatter Loading)						
실기억 공간 (Real Memory)							가상 기억 공간 (Virtual Memory)	
단일 사용자 (Single User)		다중 프로그래밍 (Multi-Programming)						
오버레이 Overlay	교체 Swapping	고정 분할 Fixed Partition	동적 분할 Dynamic Partition	페이징 Paging	세그먼테이션 Segmentation	페이지/세그먼트의 혼합 형태	요구 페이징	요구 세그먼테이션

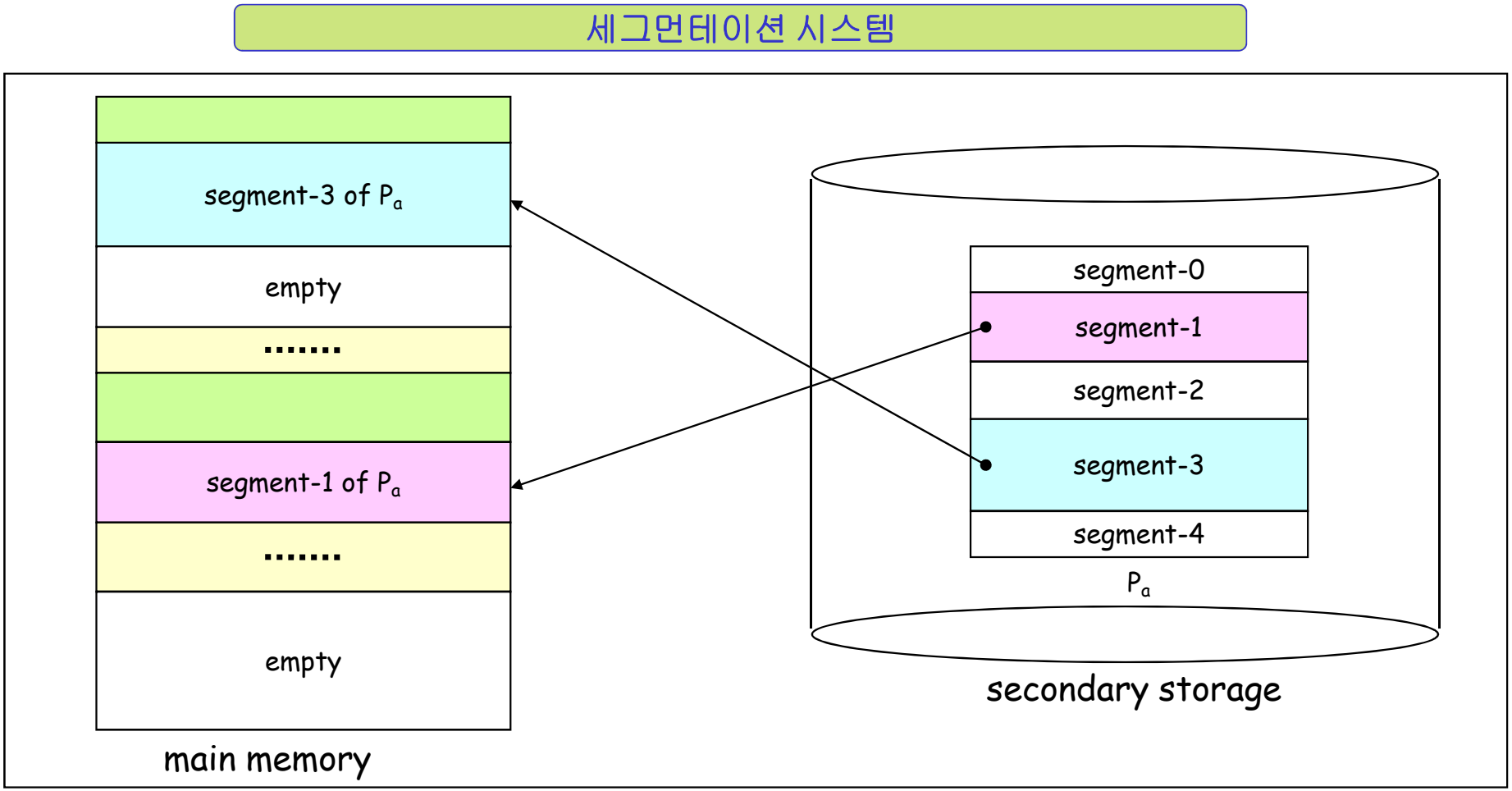
오버레이 (overlay)기법
=> 프로그램 크기가 클때
스와핑(Swapping)기법
⇒시스템 성능 향상
⇒다중 프로그래밍 지원
⇒페이징 기법으로 발전

세그먼테이션 시스템

- 세그먼테이션 시스템(**segmentation system**) 개요
  - 정의
    - 프로그램을 블럭 단위로 분할할 때 논리적인 개념을 가지고 서로 다른 크기의 블럭들로 분할하는 시스템

      ❑ 세그먼트(segment)
      - 분할되는 프로그램 블럭들
      - 프로그램을 배열이나 함수 등과 같은 논리적인 크기로 나눈 단위
  - 특징
    - 주기억장치 영역을 미리 분할 해 둘 수 없음
    - 각 세그먼트를 주기억장치에 적재할 때 빈 공간 찾아 할당함
      - 가변 분할 다중프로그래밍과 유사
    - 세그먼트 공유(**sharing**)나 보호(**protection**)가 쉬움
    - 주소 사상 기법이나 주기억장치 관리 기법 등에서 오버헤드 발생

세그먼테이션 시스템


세그먼테이션 시스템
segment-3 of P_a
empty
.....
segment-1 of P_a
.....
empty
main memory
segment-0
segment-1
segment-2
segment-3
segment-4
P_a
secondary storage

세그먼테이션 시스템

- 주소 사상
  - 세그먼테이션 시스템에서의 가상 주소 **v = (s, d)**
    - **s** = 세그먼트 번호(**segment number**)
    - **d** = 해당 세그먼트 내에서의 변위(**displacement**)
  - 세그먼트 사상 테이블(**SMT : Segment Map Table**) 이용
  - 주소 사상 기법
    - 페이징 시스템에서와 유사함
      - 직접 사상 기법
      - 연관 사상 기법
      - 혼합 사상 기법

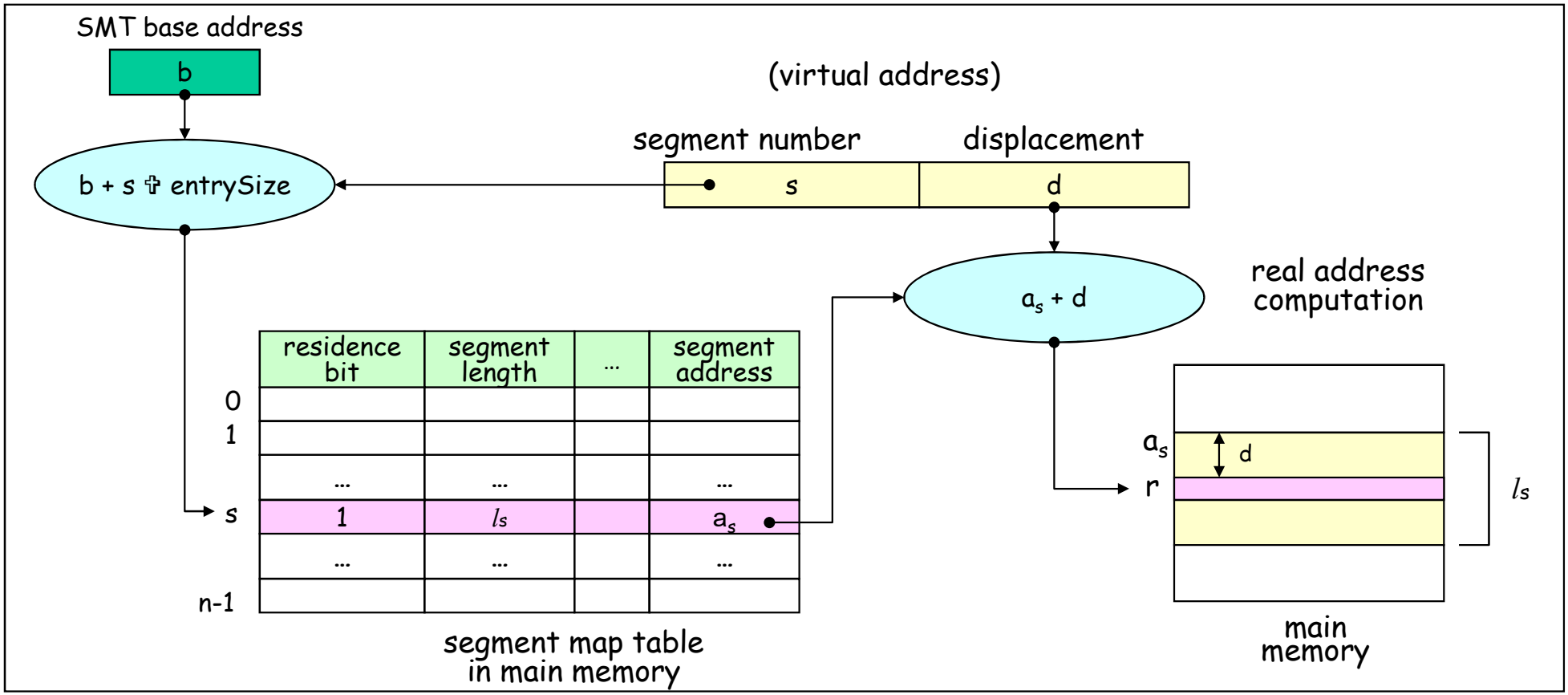
주소 사상 기법

- 직접 사상 기법
  - 가정
    - SMT가 주기억장치의 커널 공간 내에 존재
    - SMT의 엔트리 크기 = entrySize
  - SMT 참조로 인한 오버헤드를 줄이기 위해
    - SMT를 캐쉬 기억장치에 적재하거나
    - 연관 기억장치 사용 가능 (연관 사상 기법)
  - 직접 사상
    - 가상 주소는 v=(s, d)로 표현

  - 세그먼트(segment)
    - 논리적 단위가 되는 프로그램 모듈이나 자료구조 등

세그먼테이션 시스템

세그먼테이션 시스템의 직접 사상 기법



세그먼테이션 시스템

세그먼테이션 시스템의 직접 사상 알고리즘

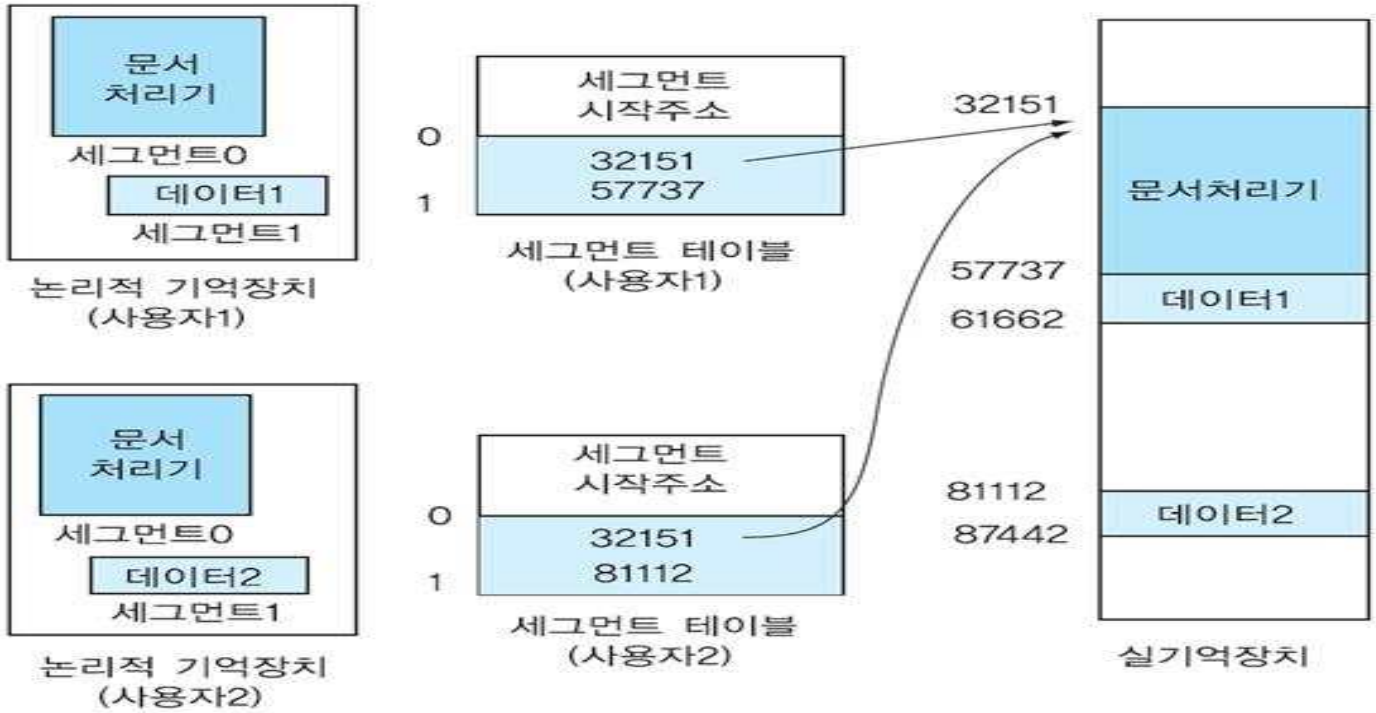
⑴ 해당 프로세스의 SMT가 저장되어 있는 주소 b에 접근
⑵ 접근한 SMT에서 세그먼트 s에 대한 엔트리 찾음
　　s의 엔트리 위치 = b + s * entrySize
⑶ 찾아진 엔트리에 대해 다음 단계들을 순차적으로 실행
　① 존재 비트가 0 인 경우
　　디스크로부터 해당 세그먼트를 주기억장치로 적재
　　SMT를 갱신
　　// missing segment fault
　② 변위가 세그먼트 길이보다 큰 경우 (d > l_s)
　　세그먼트 오버플로우 예외 처리 모듈을 호출
　　// segment overflow exception
　③ 보호 비트 필드를 검사 → 허가되지 않은 연산일 경우
　　세그먼트 보호 예외 처리 모듈을 호출
　　// segment protection exception
⑷ 세그먼트 적재 주소 a_s 를 인출 하고 변위 d와 더하여 실 주소 r 형성
　(r = a_s + d)
⑸ 실 주소 r로 주기억장치에 접근

세그먼테이션 시스템

- 주기억장치 보호 메커니즘
  - 보호 키 필드 사용
    - 프로세스들이 접근할 수 있는지를 기록하도록 함
- 주기억장치 할당을 위한 배치 기법
  - 최초 적합(first-fit) 전략
  - 최적 적합(best-bit) 전략
  - 최악 적합(worst-fit) 전략
  - 순환 최초 적합(next-fit) 전략 등
- 세그먼트의 공유(sharing)를 위한 메커니즘
  - 페이징 시스템과 유사하며 보다 간단함

세그먼테이션 시스템

- 세그먼트 공유 및 보호
- 세그먼테이션 시스템에서의 세그먼트 공유



세그먼테이션 시스템

- 공유 및 보호
  - 보호를 위한 접근(access) 제어
    - 세그먼테이션 시스템의 또 다른 장점 중의 하나는 세심한 접근 제어가 가능
  - 접근 제어 기법
    - 접근 제어 정의
      - 프로세스가 자신의 프로그램 코드나 데이터 등의 **context**에 접근할 때 허가되지 않은 형태의 접근을 금지하는 메커니즘
    - SMT에 보호 비트 필드 사용
      - 4가지 접근 권한
        - 읽기(R, Read) 권한: 해당 세그먼트의 내용을 읽을 수 있음
        - 쓰기(W, Write) 권한 : 해당 세그먼트의 내용을 변경시킬 수 있음(붙이기 가능)
        - 실행(X, eXecute) 권한 : 해당 세그먼트에 저장되어 있는 명령들을 실행시킬 수 있음
        - 붙이기(A,Append) 권한 : 해당 세그먼트의 내용 뒤에 다른 내용 붙일 수 있음(변경 불가)

페이징/세그먼테이션 혼합 기법

- **페이징 시스템**
  - 장점 : 단순하고 오버헤드가 적은 기법
  - 단점 : 논리적인 분할의 개념 없음
    페이지 공유 등과 관련하여 복잡한 문제 발생함

- **세그먼테이션 시스템**
  - 장점 : 사용자 프로그램을 논리적으로 분할함
    세그먼트의 공유 쉬움
  - 단점 : 관리 오버헤드 증가함

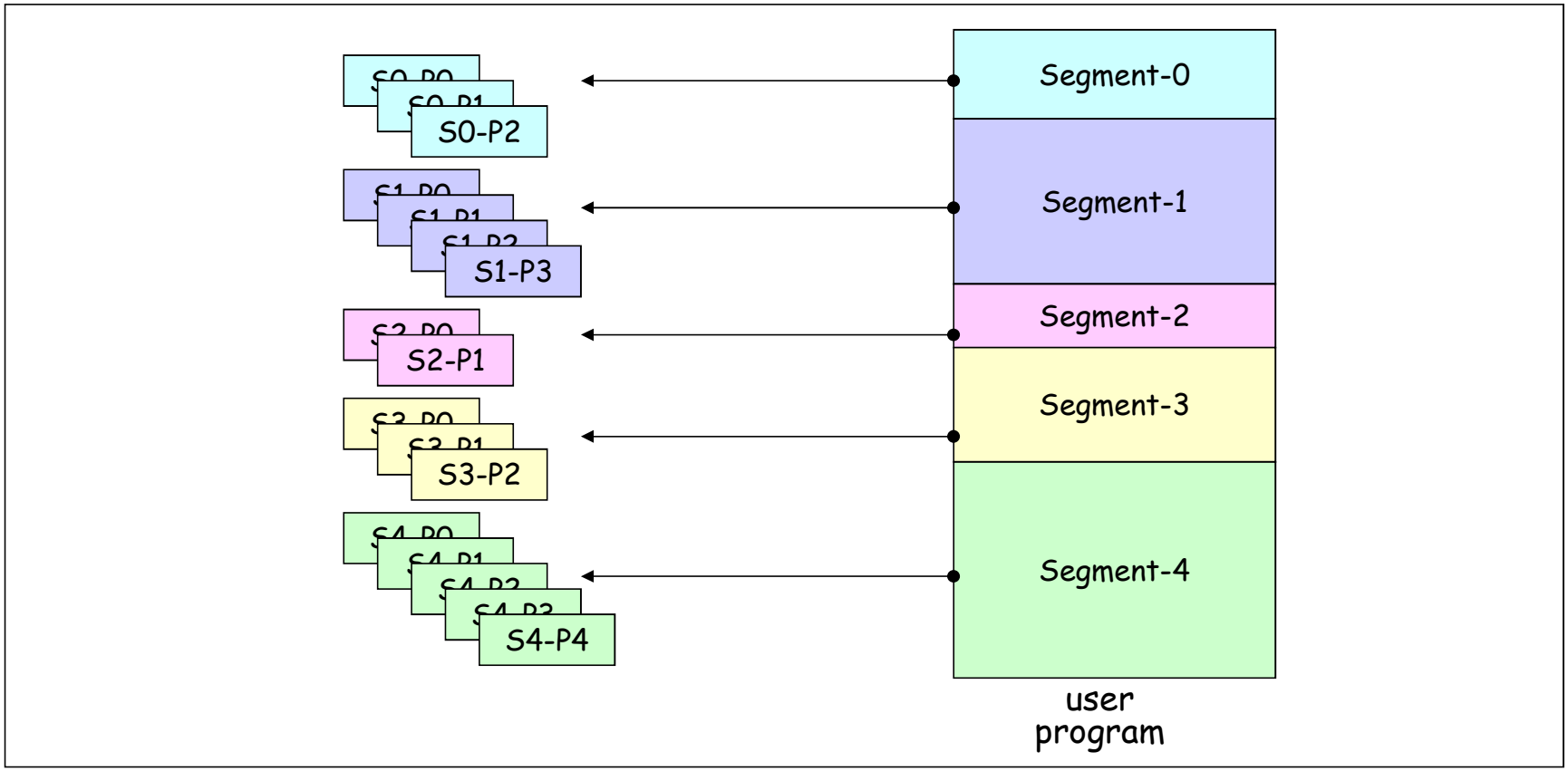
페이징/세그먼테이션 혼합 기법

- **페이징/세그먼테이션 혼합 기법**
  - 두 가상기억장치 관리 기법의 장점을 모두 수용
  - 하나의 세그먼트를 정수 배의 페이지로 다시 분할하는 세그먼트/페이징 혼용 기법
  - 분할
    - 우선 사용자 프로그램을 논리적인 세그먼트 단위로 분할
    - 각 세그먼트들을 다시 페이지 단위로 분할
  - 적재
    - 분할된 페이지 단위로 주기억장치에 적재
  - 가상 주소 v는 3차원의 요소로 구성 : v=(s, p, d)

  - 시스템의 공유
    - 세그먼트의 공유는 서로 다른 프로세스의 세그먼트 사상 테이블의 항목들이 동일 페이지 사상 테이블을 공유

페이징/세그먼테이션 혼합 기법

페이징/세그먼테이션 시스템의 프로그램 분할

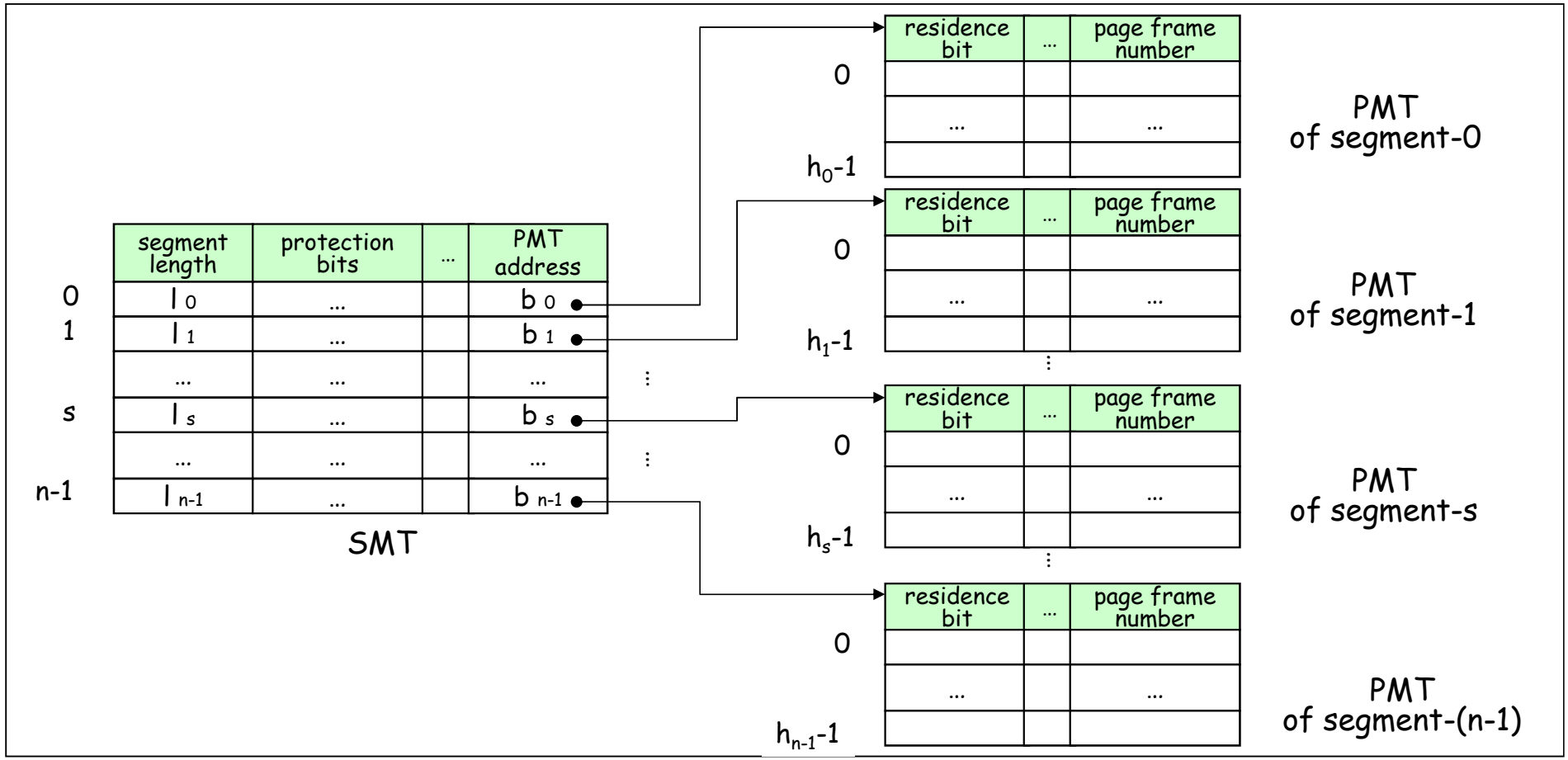


주소 사상 기법

- 주소 사상
  - 페이징/세그먼테이션 혼합기법에서의 가상 주소 **v = (s, p, d)**
    - **s** = 세그먼트 번호
    - **p** = 해당 세그먼트에서의 페이지 번호
    - **d** = 해당 페이지에서의 변위
  - **SMT**와 **PMT** 모두 사용
    - 각 프로세스마다 하나의 **SMT** 존재
    - 그 프로그램의 세그먼트 개수만큼 **PMT** 존재
  - 주소 사상 기법
    - 직접/연관/혼합 사상 기법 모두 가능
  - 주기억장치는 페이지 프레임 단위로 미리 분할되어 있음

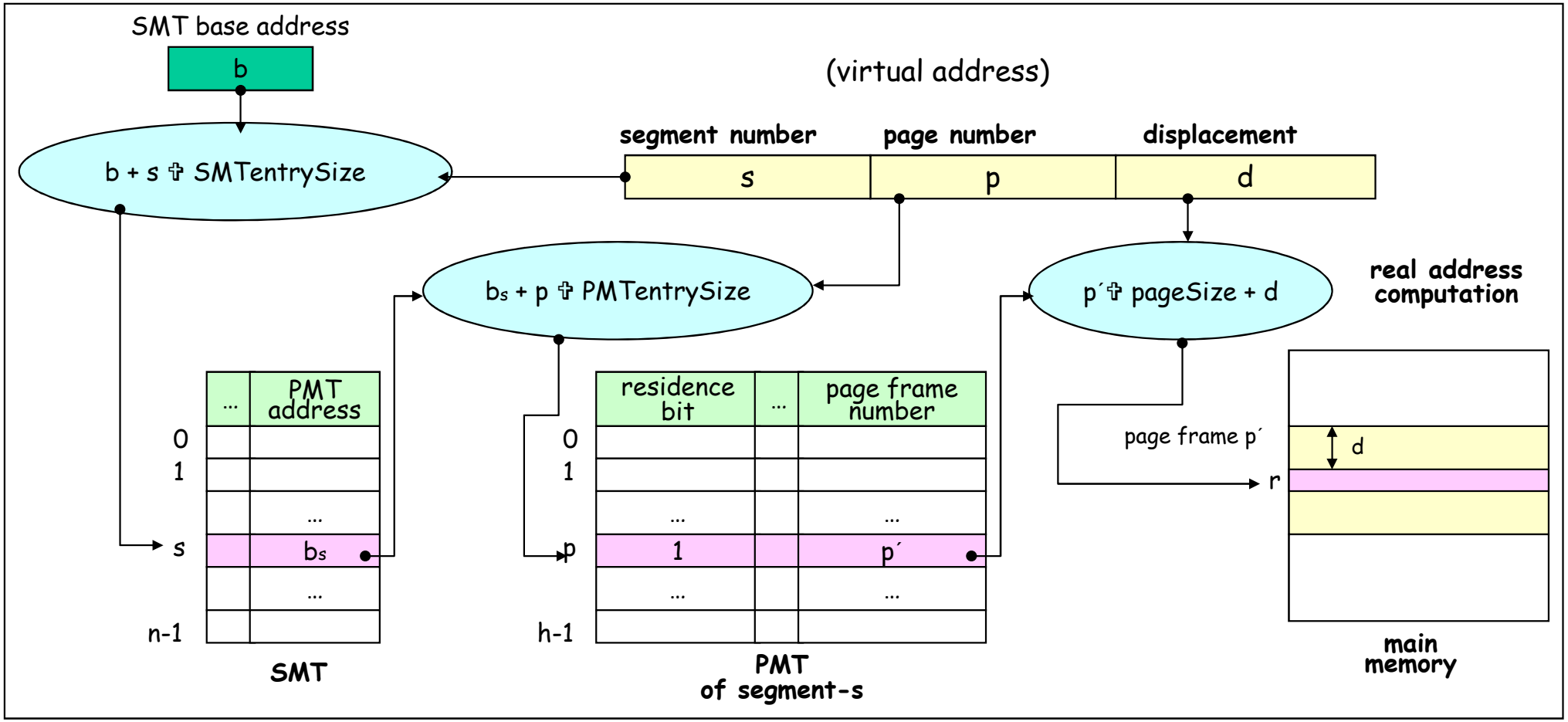
페이징/세그먼테이션 혼합 기법

페이징/세그먼테이션 혼합 기법에서의 주소 사상 테이블



페이징/세그먼테이션 혼합 기법

페이징/세그먼테이션 혼합 기법의 직접 사상 기법



결론

- 페이징 기법
  - 사용자 프로그램을 미리 정해진 일정한 크기로 분할함
  - 단순하고 효과적인 주소 사상이 가능
  - 논리적인 개념 없이 분할되어 프로그램 공유 문제 등 복잡
- 세그먼테이션 기법
  - 프로그램 공유 문제 단순함
  - 서로 다른 크기의 세그먼트 관리 오버헤드 추가
- 페이징/세그먼테이션 혼합 기법
  - 주기억장치 관리의 단순성, 효율성 제공, 프로그램 공유 문제 해결
  - 주소 사상 과정이 복잡하고 주소 사상 테이블의 전체 크기 커짐

문제

다음과 같은 세그먼트 맵 테이블이 있을 때, 실제 주소는 얼마가 되겠는가? (단, 가상주소 =s(2,100))

세그먼트번호	크기	시작주소
0	1200	4000
1	800	5700
2	1000	2000
3	500	3200

가 1500　　나. 1000　　다. 2000　　라. 2100