제8장
가상기억장치 구성

컴퓨터 운영체제 OS Operating Systems

Network Security Technology Lab.

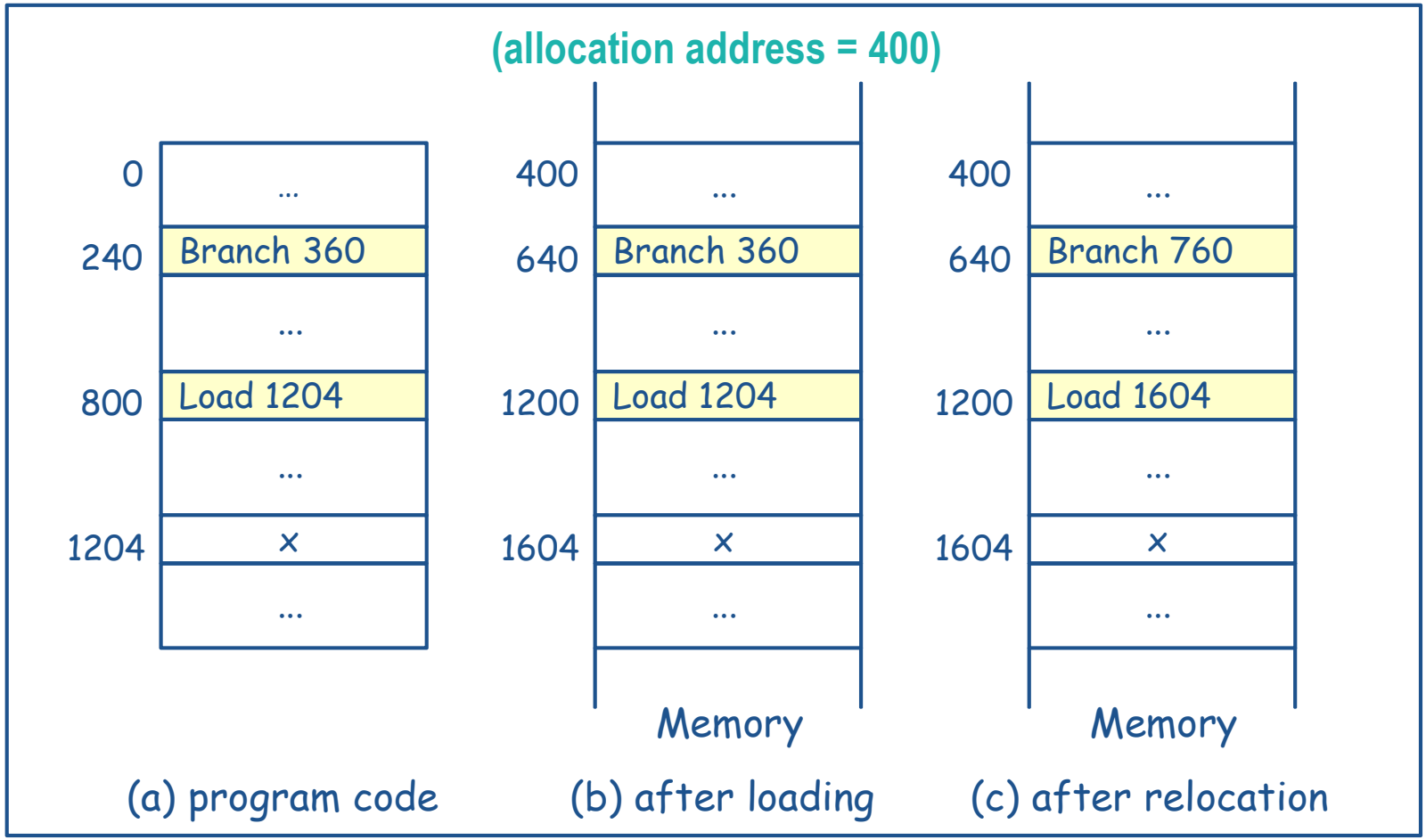
가상기억장치 개념

- 가상기억장치(virtual storage)
  - 개념
    - 사용자 프로그램을 여러 블럭들로 분할
    - 실행시 필요한 블럭들만 비연속적으로 주기억장치에 적재시킴
  - 분류
    - 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

주소 사상 기법

- 프로그램 전체를 연속으로 적재하는 경우
  - 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)



- 프로그램을 분할하여 그 일부만을 비연속 적재하는 경우
  - 가상 주소(virtual address) = 상대 주소
  - 실 주소(real address) = 절대 주소
  - 주소 사상 함수(address mapping function)
    - 가상 주소를 실 주소로 변환하는 작업을 실행 중에 동적으로 수행하기 위한 기법

정의 : 주소 사상 함수

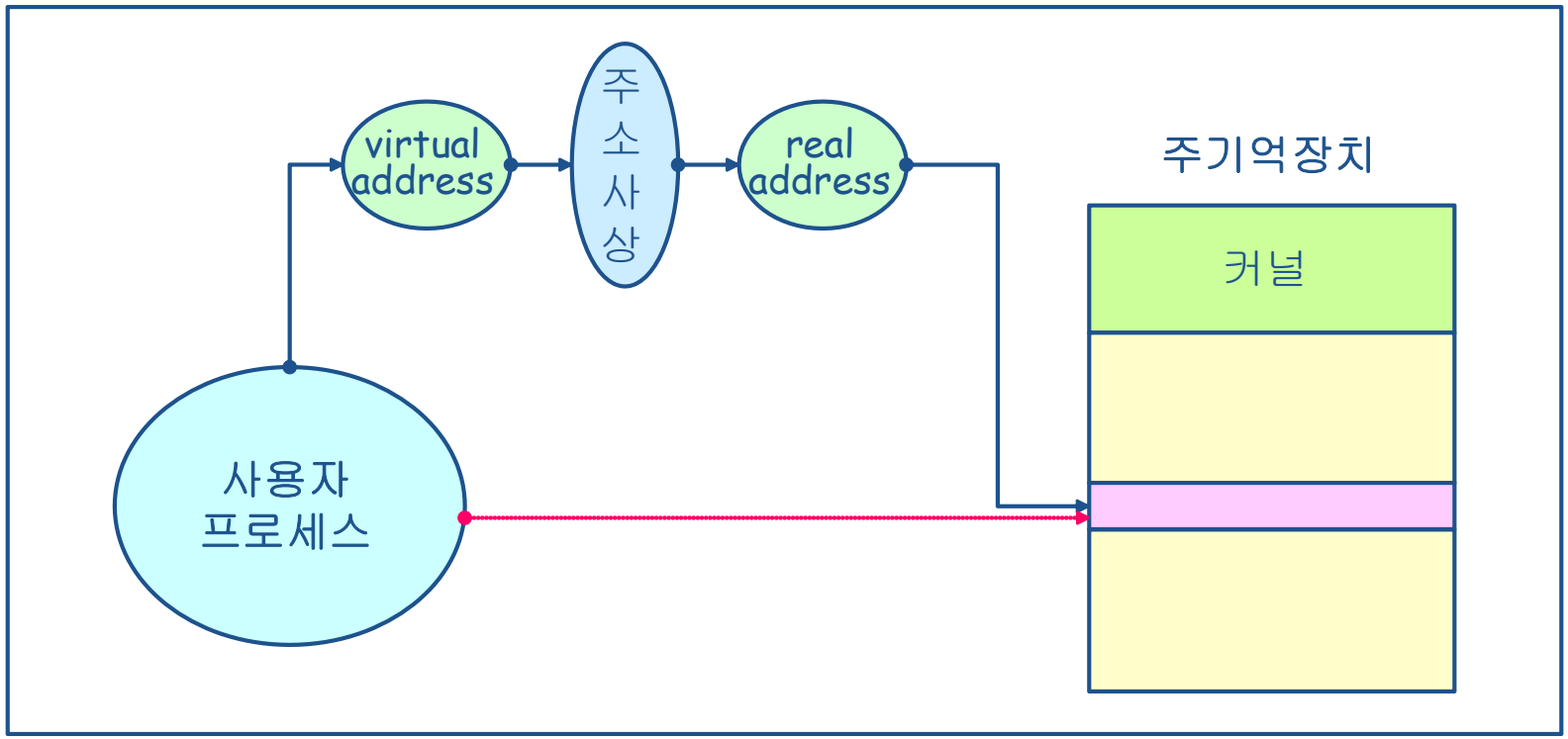
- 프로그램의 실행 중 발생되는 가상 주소 v를 주기억장치 상의 실 주소 r로 변환하는 함수
- 주소 사상 함수 f() 정의

  f : V → R - V : 가상 주소 공간(virtual address space)

  r = f(v) - R : 실 주소 공간(real address space)

- 일반적으로 |V| >> |R| 임을 가정

주소 사상 과정



- 인위적 연속성(artificial contiguity)
  - 사용자 또는 프로세스 입장에서 실행 프로그램 전체가 주기억장치에 적재되어 있는 것으로 보게 하는 개념

블럭 사상

블럭 사상(block mapping) 개념

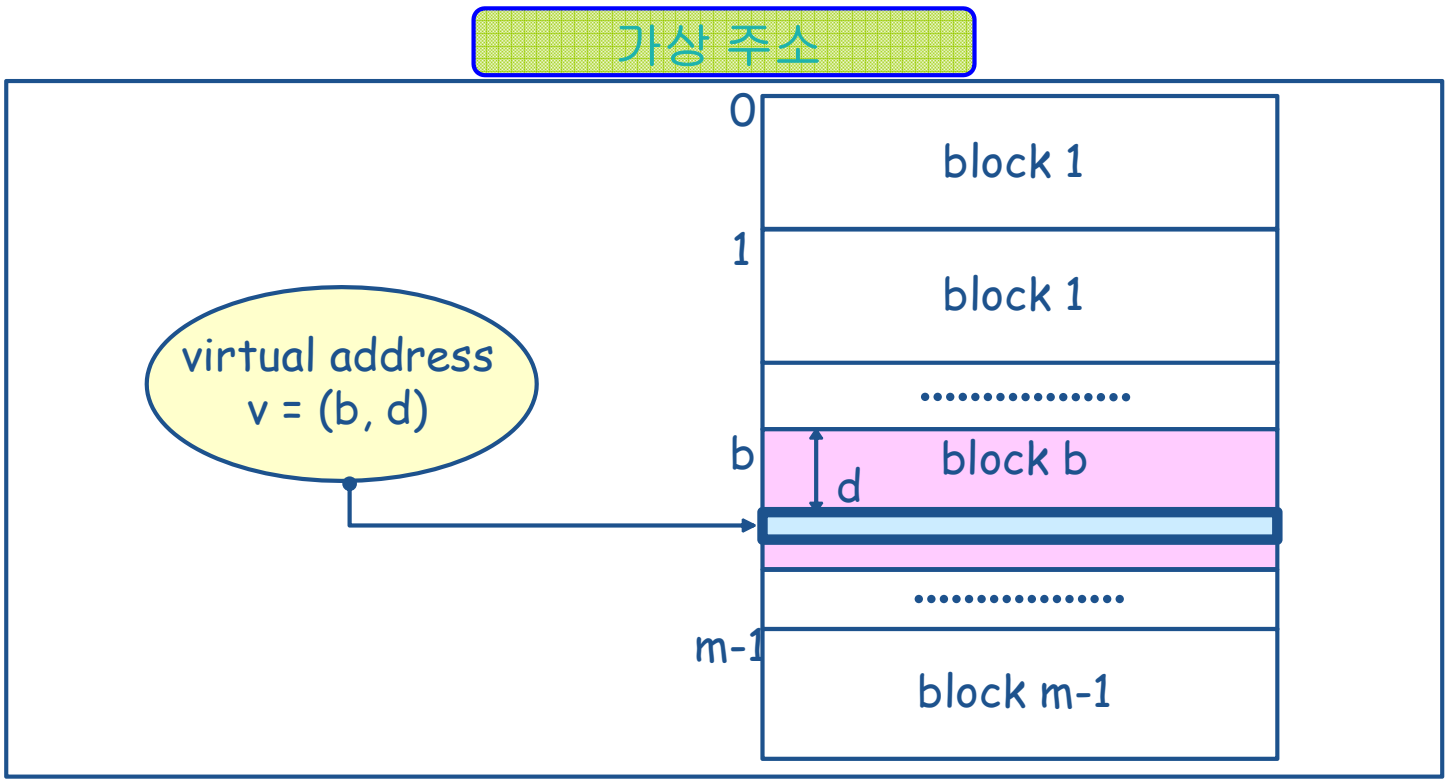
- 블럭 사상(block mapping)
  - 프로그램을 블럭 단위로 분할하고 이렇게 분할된 블럭 단위로 주소 사상 정보를 기록하여 사용하는 기법
- 블럭 사상 테이블(block map table)
  - 각 블럭의 가상 주소와 이에 대응되는 실 주소 저장

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 사상 기법

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



- 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

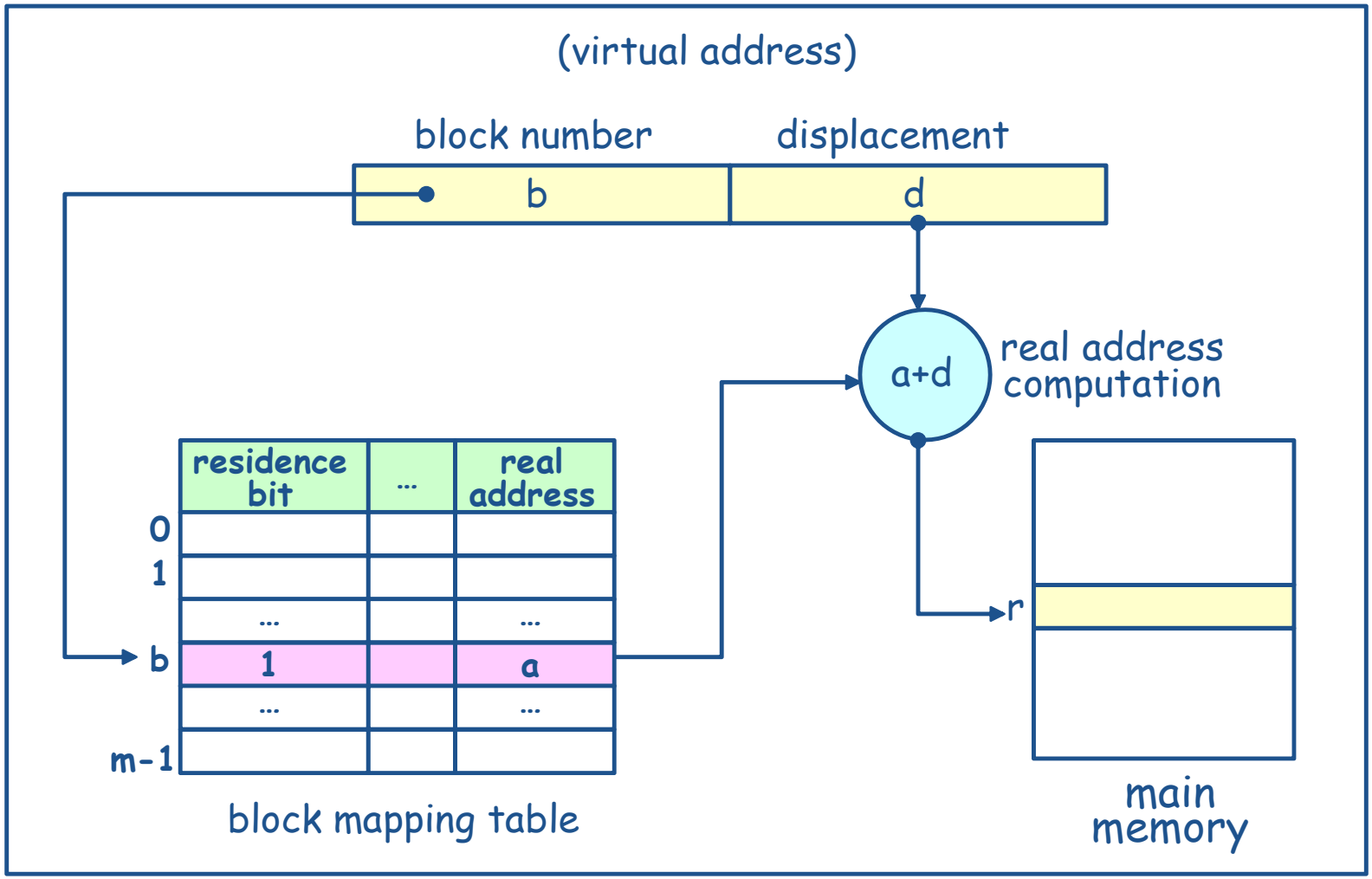
block number	residence bit	...	real address
0 1 2 ...		. . .	
b	1	.....	a
... m-1		: :	

- 존재 비트 : 해당 블럭이 주기억장치에 적재되어 있는지의 여부
- 실 주소 : 해당 블럭이 적재된 주기억장치 주소

가상 주소 v = (b, d)를 실 주소로 변환하는 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 실 주소 필드의 값 a 인출
⑷ 인출된 값 a와 가상 주소의 변위 d를 더하여 실 주소 r 형성 (r = a + d)
⑸ 실 주소 r로 주기억장치에 접근

블럭 사상 과정


(virtual address)
block number
displacement
b
d
a+d
real address computation
residence bit
...
real address
0
1
...
b
1
a
...
m-1
r
block mapping table
main memory

페이징 시스템

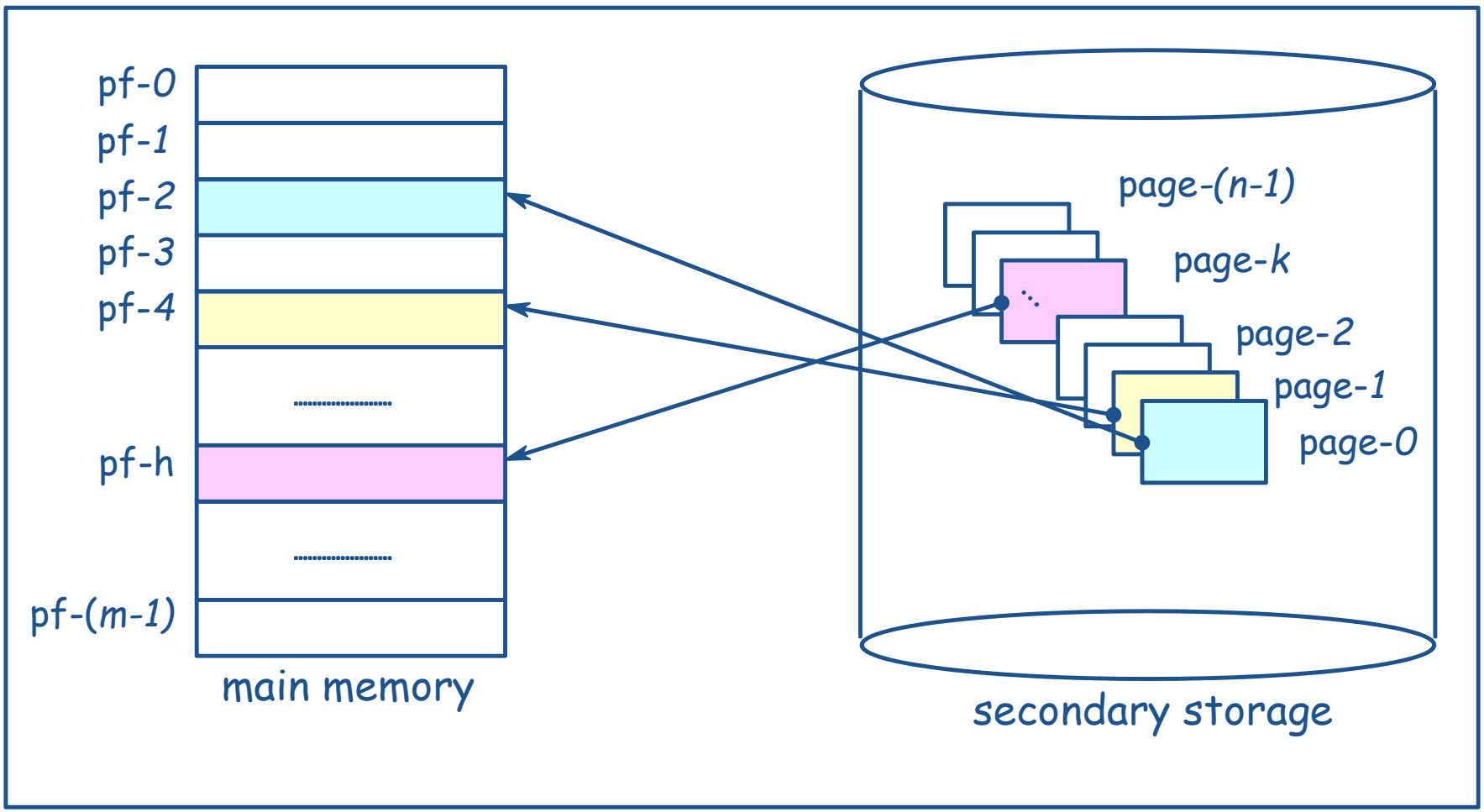
❖ 페이징 시스템(paging system) 개요

- 정의
  - 프로그램을 블럭 단위로 분할할 때
    같은 크기의 블럭들로 분할하는 시스템
- 용어

❑ 페이지(page)
- 실행 프로그램의 분할된 블럭

❑ 페이지 프레임(page frame), 페이지 틀
- 주기억장치의 분할 영역
- 페이지 크기(page size)와 같은 크기로 분할됨

페이징 시스템


pf-0
pf-1
pf-2
pf-3
pf-4
pf-h
pf-(m-1)
main memory
page-(n-1)
page-k
page-2
page-1
page-0
secondary storage

페이징 시스템

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

페이징 시스템

- 주소 사상
  - 페이징 시스템에서의 가상 주소 v = (p, d)
    - p = 페이지 번호
    - d = 해당 페이지 내에서의 변위
  - 주소 사상
    - 페이지 사상 테이블(PMT : Page Map Table) 이용
  - 주소 사상 기법
    - 직접 사상(direct mapping)
    - 연관 사상(associative mapping)
    - 혼합 사상(mixed direct/associative mapping)

페이지 사상 테이블 (PMT : Page Mapping Table)

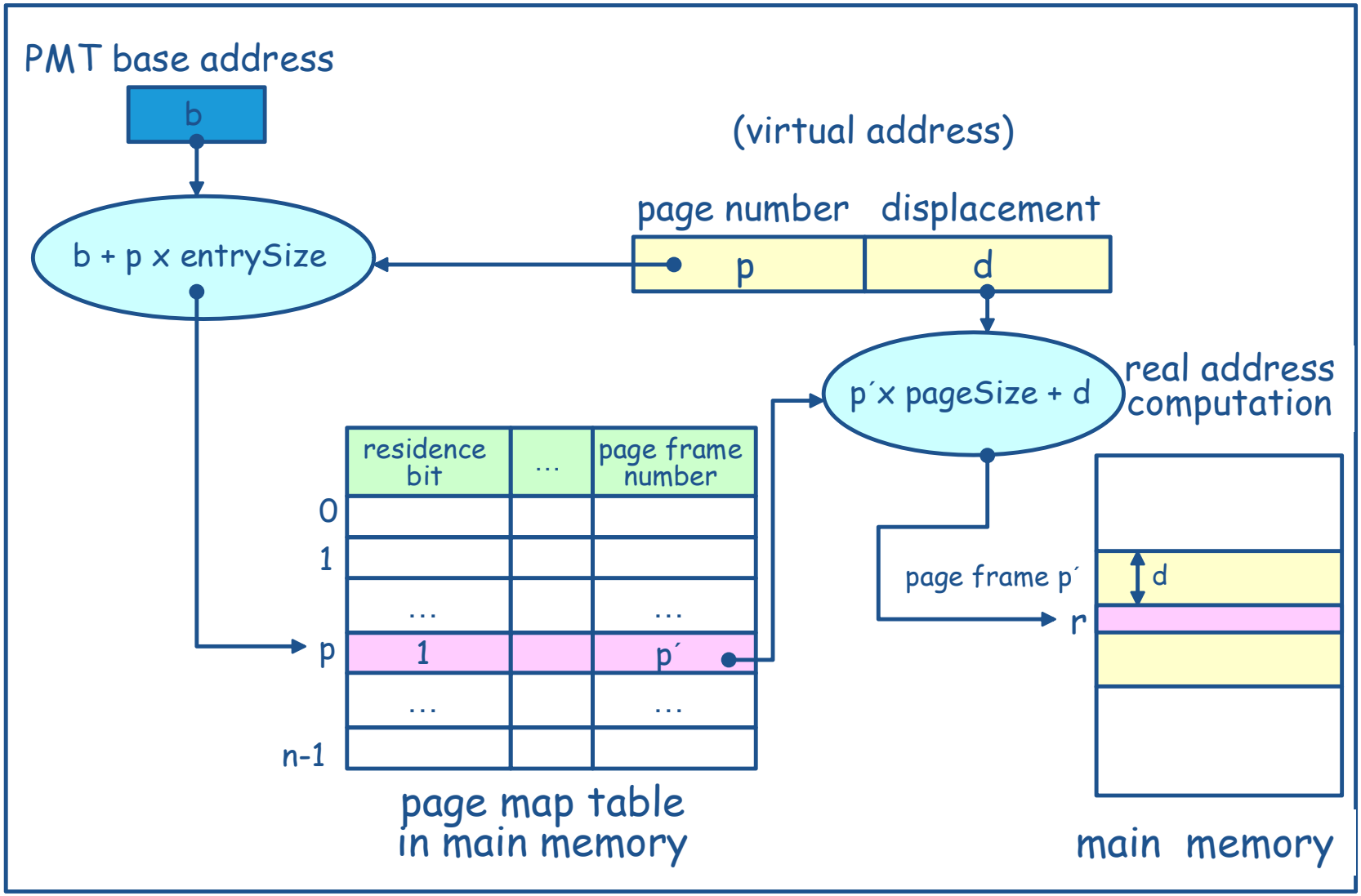
page number	residence bit	secondary storage address	other fields	page frame number
0	1	S_0	. . .	2
1	1	S_1	. . .	4
2	0	S_2	. . .	-
. . .	. . .	...	. . .	. . .
k	1	S_k	. . .	h
. . .	. . .	...	. . .	. . .
n-1	0	S_{n-1}	. . .	-

- 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

페이징 시스템의 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)
디스크로터 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

- 직접 사상 기법의 단점
  - 주기억장치 접근 회수 2배로 증가
  - 성능 저하 초래
- 직접 사상 기법의 단점 해결
  - PMT를 캐쉬 기억장치에 적재하는 기법
  - 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 (↑ 제9장 참조)

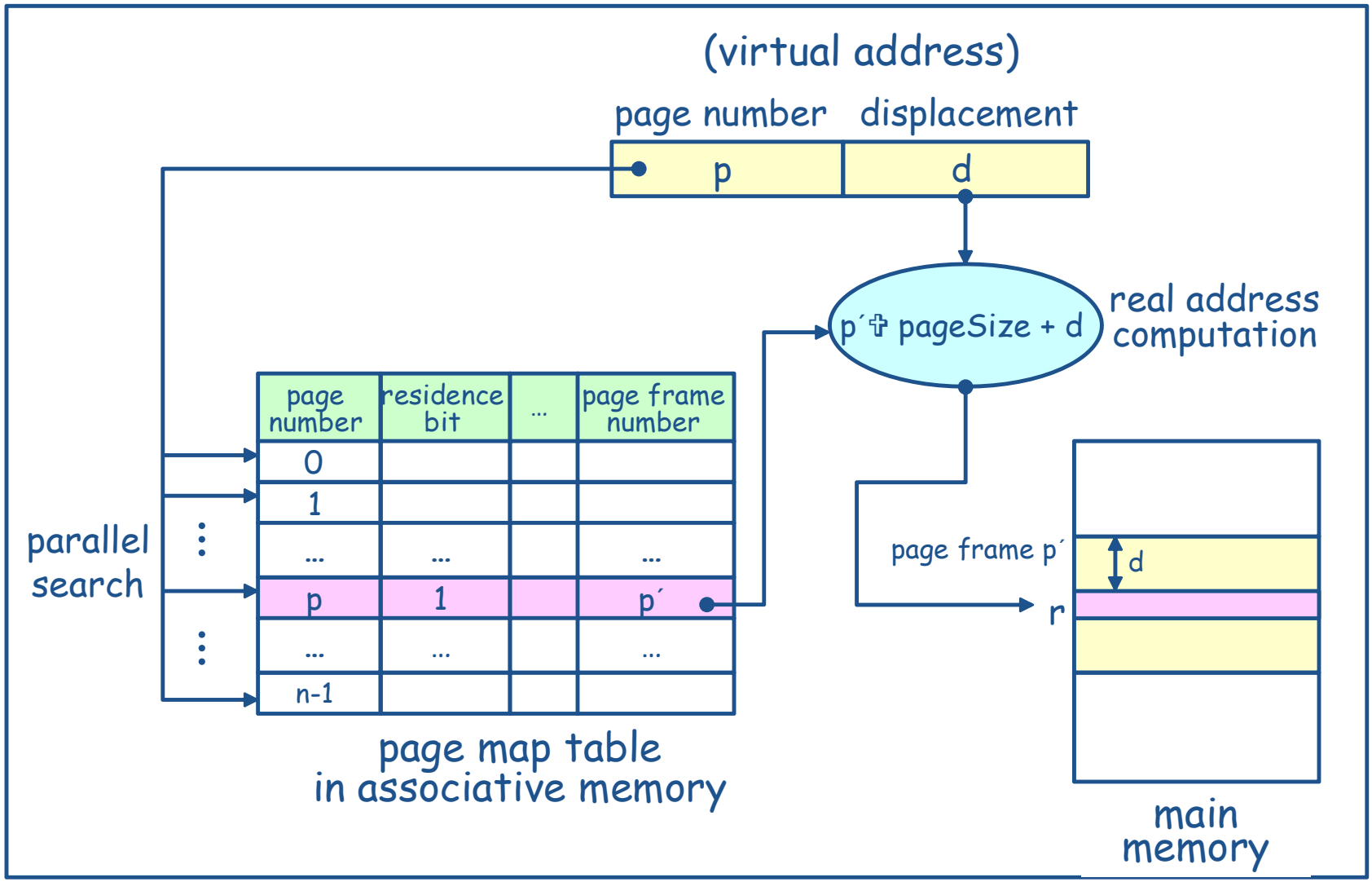
연관 사상(associative mapping) 기법

- PMT를 연관 기억장치에 적재하여 사용하는 기법
- 직접 사상 기법의 오버헤드 줄이기 위한 기법
- 직접 사상 기법의 메커니즘과 유사함

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

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

페이징 시스템의 연관 사상 기법



❖ 혼합 사상(combined direct/associative mapping) 기법

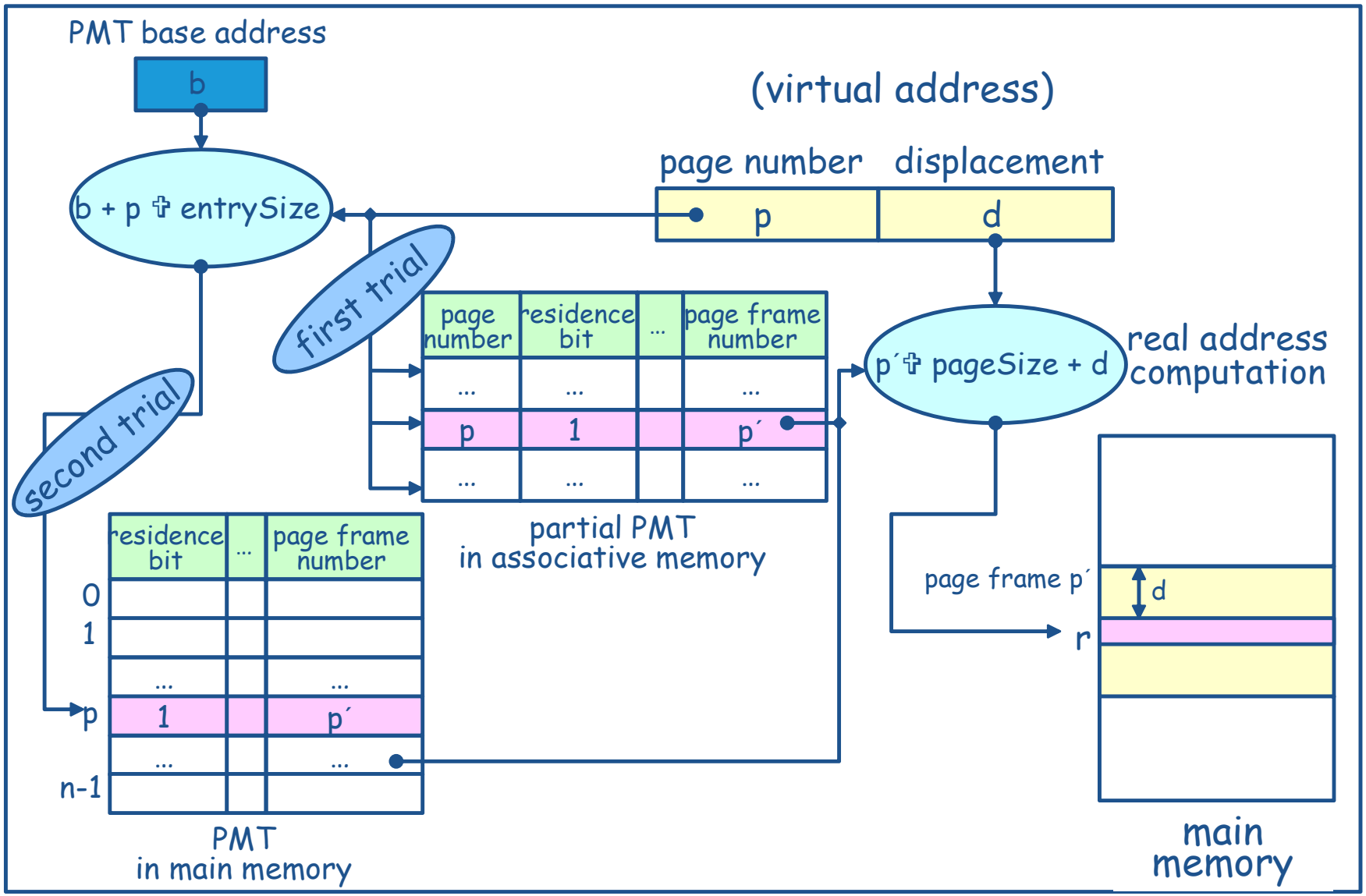
- 직접 사상 기법과 연관 사상 기법을 같이 사용
  - 하드웨어 비용 줄이면서 연관 사상 기법의 장점 취하는 기법
- 소규모의 연관 기억장치 사용
  - PMT의 전체 내용 : 주기억장치의 커널 공간에 적재
  - PMT의 일부 내용 : 연관 기억장치 내 적재
    - 최근에 사용된 페이지에 대한 엔트리들만 적재

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

페이징 시스템의 혼합 사상 알고리즘

- 해당 프로세스의 PMT가 연관 기억장치에 적재되어 있는지 확인
  - ① 연관 기억장치에 적재되어 있는 경우
    존재 비트 검사하고 페이지 프레임 번호 인출
  - ② 연관 기억장치에 적재되어 있지 않은 경우
    직접 사상 기법으로 페이지 프레임 번호 인출
    접근된 PMT 엔트리를 연관 기억장치에 적재함

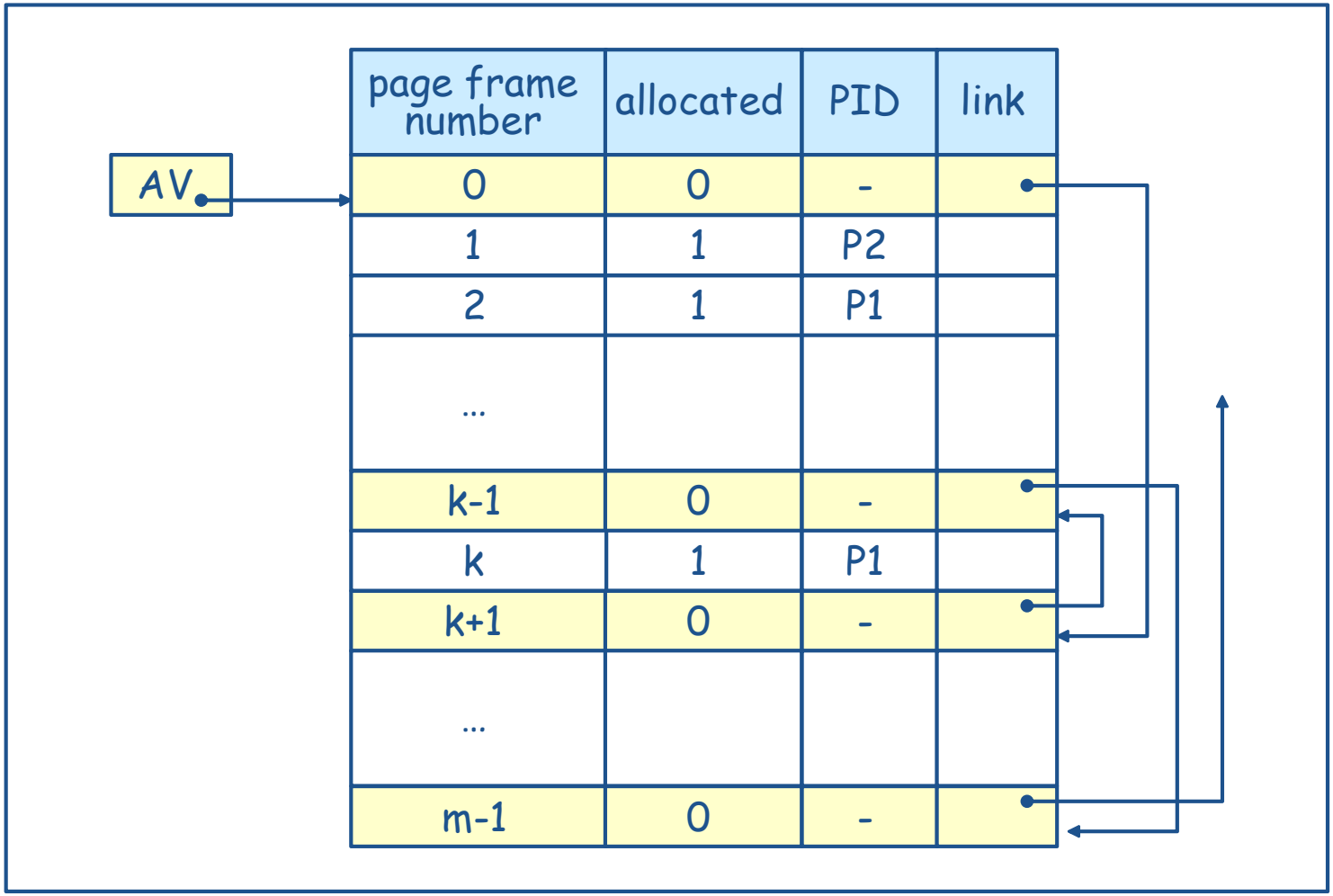
페이징 시스템의 혼합 사상 기법



주기억장치 관리

- 페이징 시스템에서의 주기억장치 관리
  - 주기억장치를 페이지 크기와 같은 크기의 페이지 프레임 단위로 미리 분할하여 관리함
    - FPM 기법과 유사함
  - 주기억장치 관리 테이블
    - allocated 필드
      - 해당 페이지 프레임이 현재 프로세스에게 할당되어 있는지
    - PID 필드
      - 어느 프로세스에게 할당되어 있는지
    - link 필드
      - 현재 할당되어 있지 않은 페이지 프레임들을 하나의 연결 리스트로 연결시키기 위해 사용
    - AV
      - 빈 페이지 프레임 리스트의 헤더

페이징 시스템에서의 주기억장치 관리 테이블

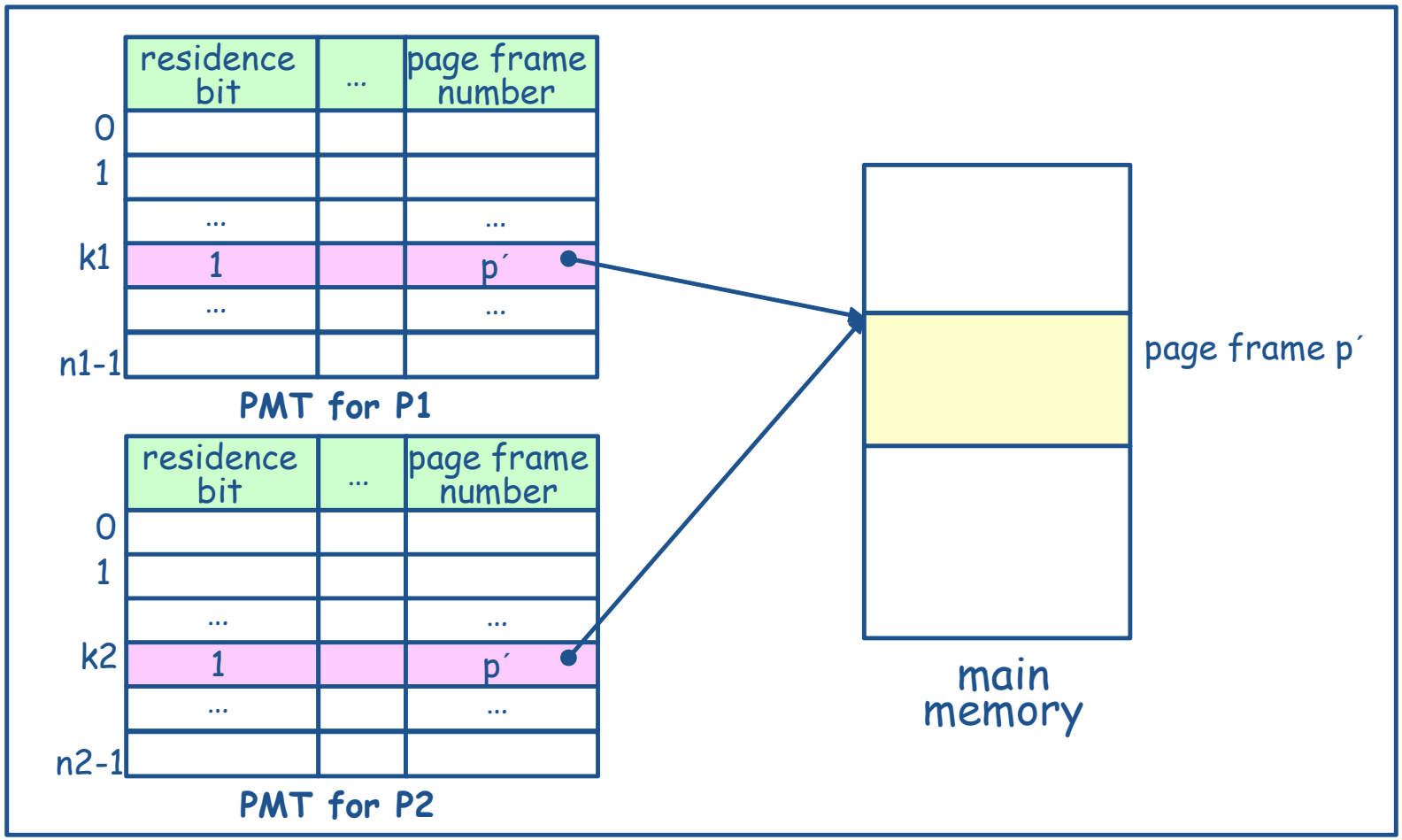


페이지 공유

페이지 공유

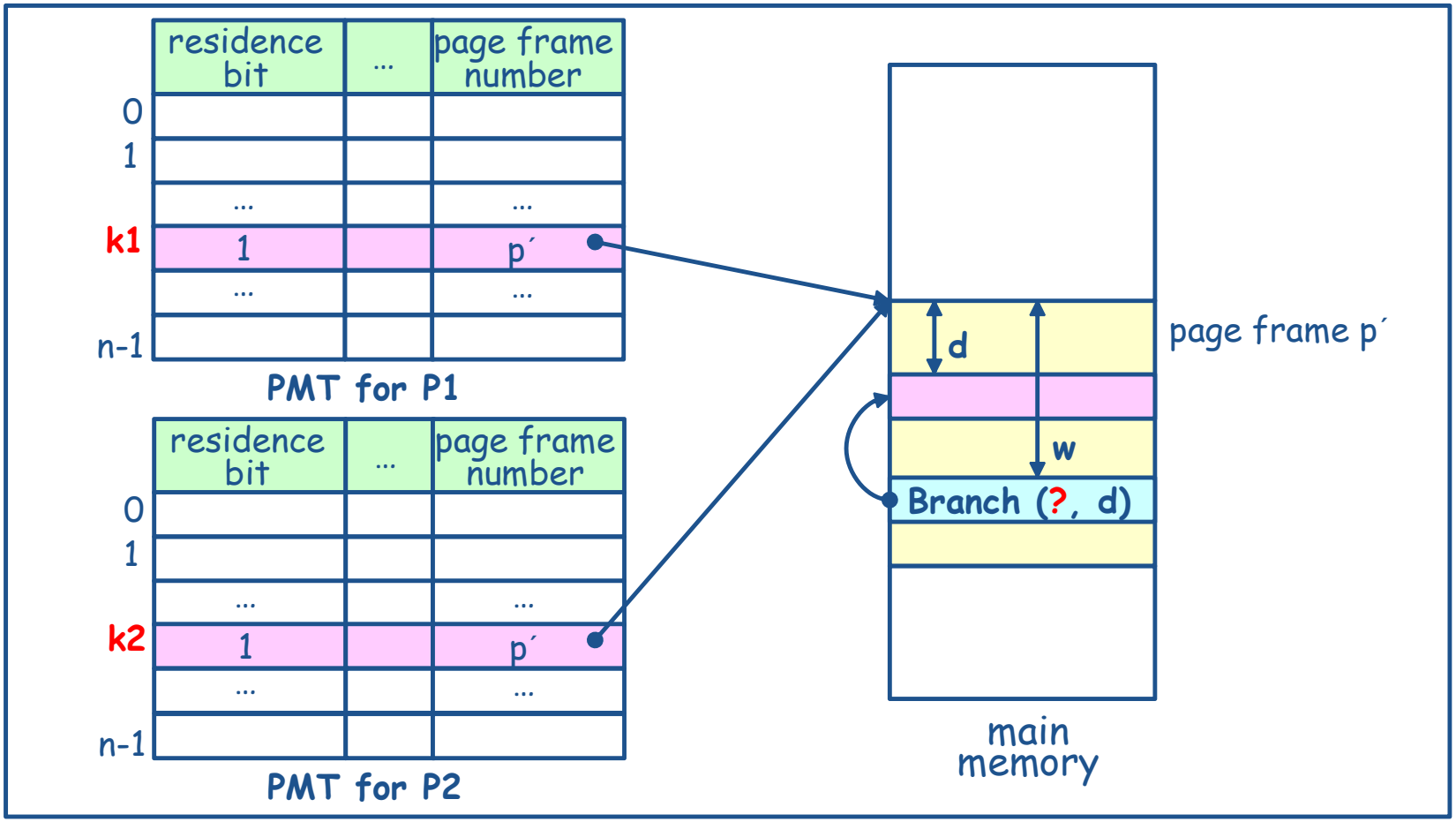
- 공유의 대상이 되는 페이지
  - 프로시져 페이지(procedure page)
    - 순수 코드(재진입가능 코드)일 경우에만 공유 가능
  - 데이터 페이지(data page)
    - 순수 데이터(읽기-전용 데이터)의 경우
      - » 그대로 공유 가능
    - 비순수 데이터(읽기-쓰기 데이터)의 경우
      - » 병행성 제어기법 관리하에 공유 가능

페이지 공유



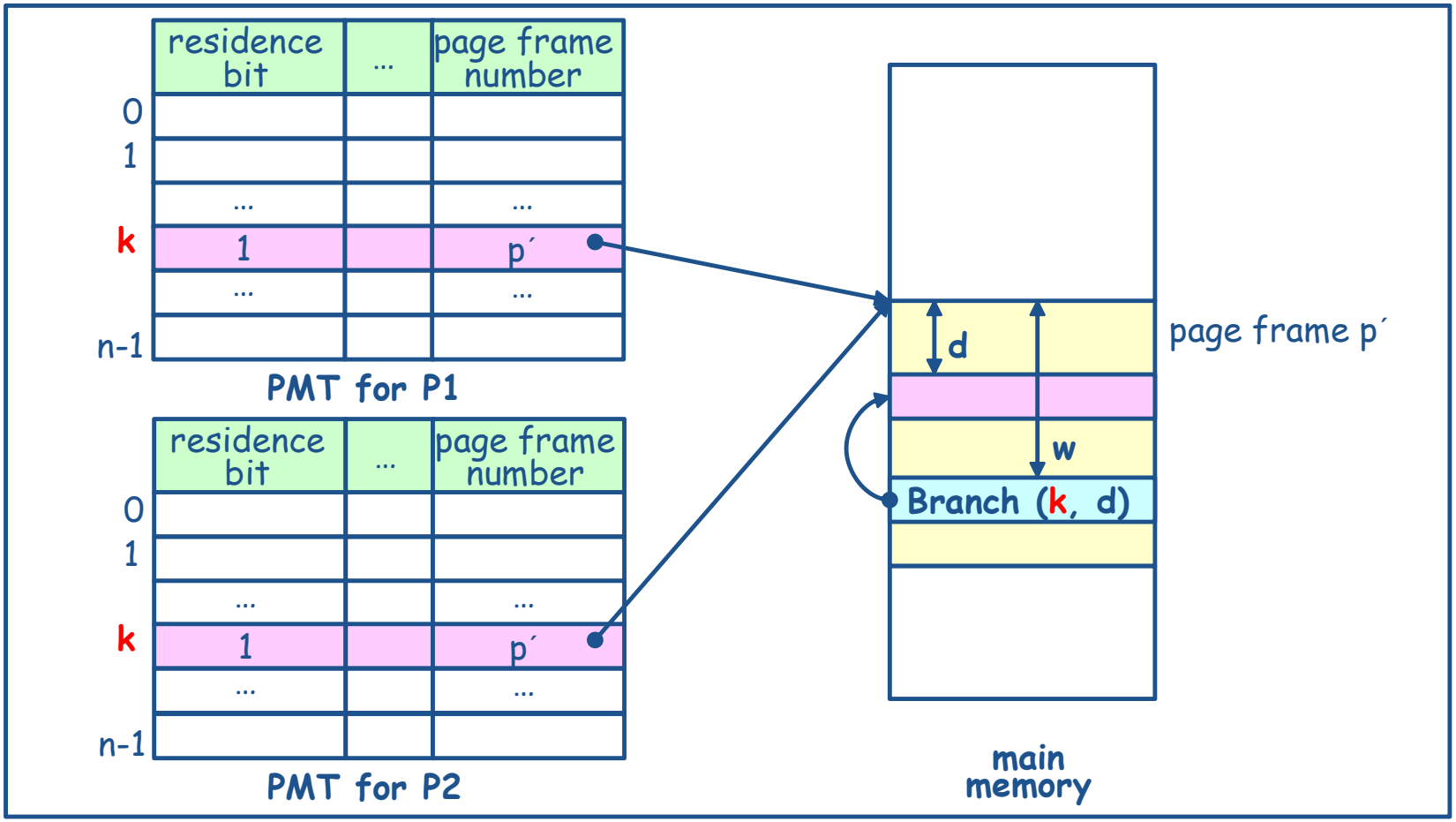
- 공유 대상 페이지가 데이터 페이지인 경우

프로시져 페이지 공유 오류



- p'의 변위 w에 있는 Branch 명령에서 같은 페이지의 변위 d의 위치로 분기하려면
  - Branch(k1, d) or Branch(k2. d) ?
- 공유 페이지의 코드를 실행하는 프로세스마다
  자신의 가상 주소 공간 내에서의 분기 주소가 다르기 때문에 오류 발생

프로시저 페이지 공유



- p'의 변위 w에 있는 Branch 명령에서 같은 페이지의 변위 d의 위치로 분기하려면
  - Branch(k, d)
- 각 프로세스들이 공유 페이지에 대한 정보를 PMT의 같은 엔트리에 저장하도록 함

세그먼테이션 시스템

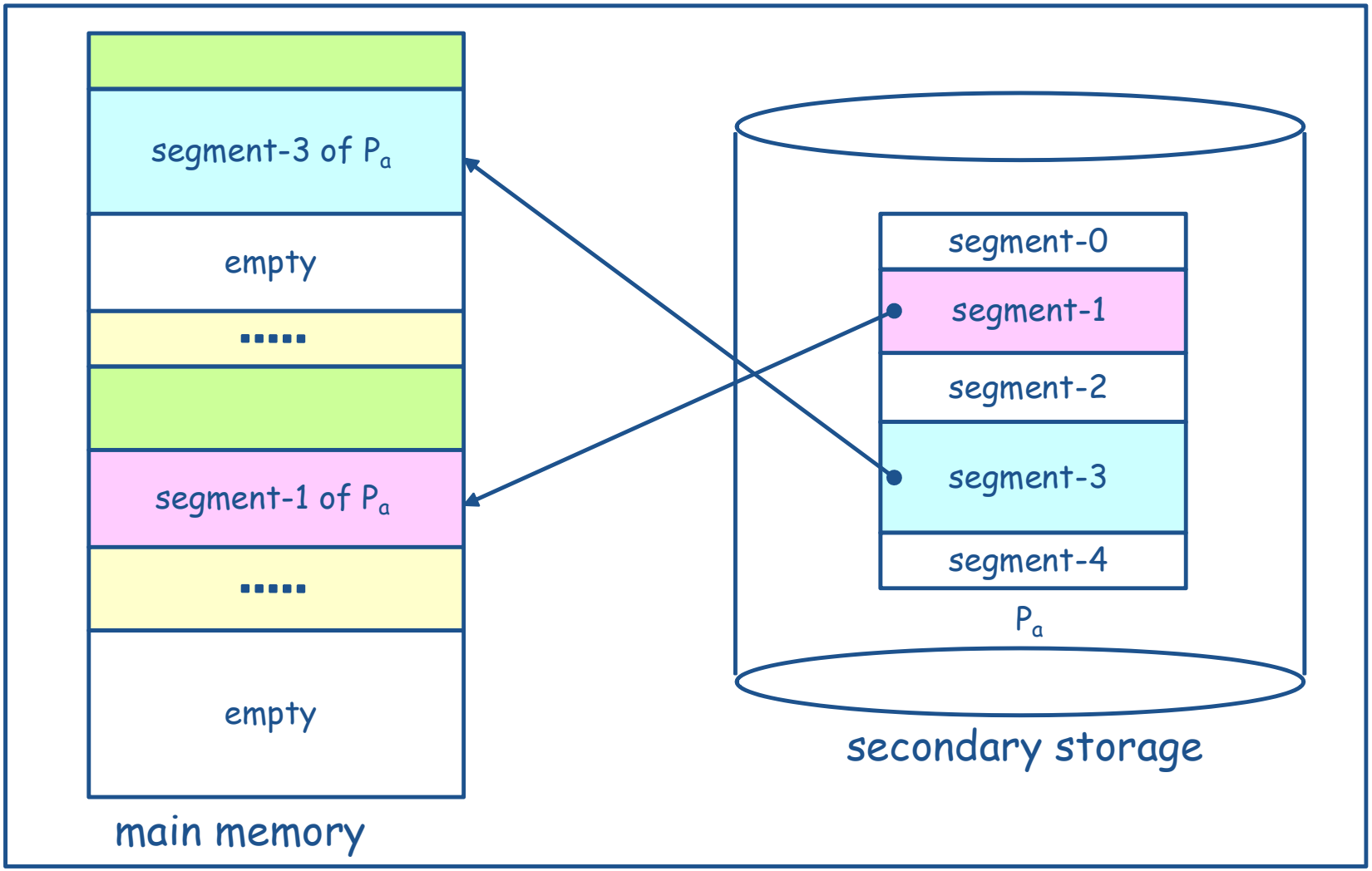
❖ 세그먼테이션 시스템(segmentation system) 개요

- 정의
  - 프로그램을 블럭 단위로 분할할 때 논리적인 개념을 가지고 서로 다른 크기의 블럭들로 분할하는 시스템

> ❑ 세그먼트(segment)
> - 분할되는 프로그램 블럭들

- 특징
  - 주기억장치 영역을 미리 분할 해 둘 수 없음
  - 각 세그먼트를 주기억장치에 적재할 때 빈 공간 찾아 할당함
    - 가변 분할 다중프로그래밍(VPM)과 유사
  - 세그먼트 공유(sharing)나 보호(protection)가 쉬움
  - 주소 사상 기법이나 주기억장치 관리 기법 등에서 오버헤드 발생

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

세그먼트 사상 테이블 (SMT : Segment Map Table)

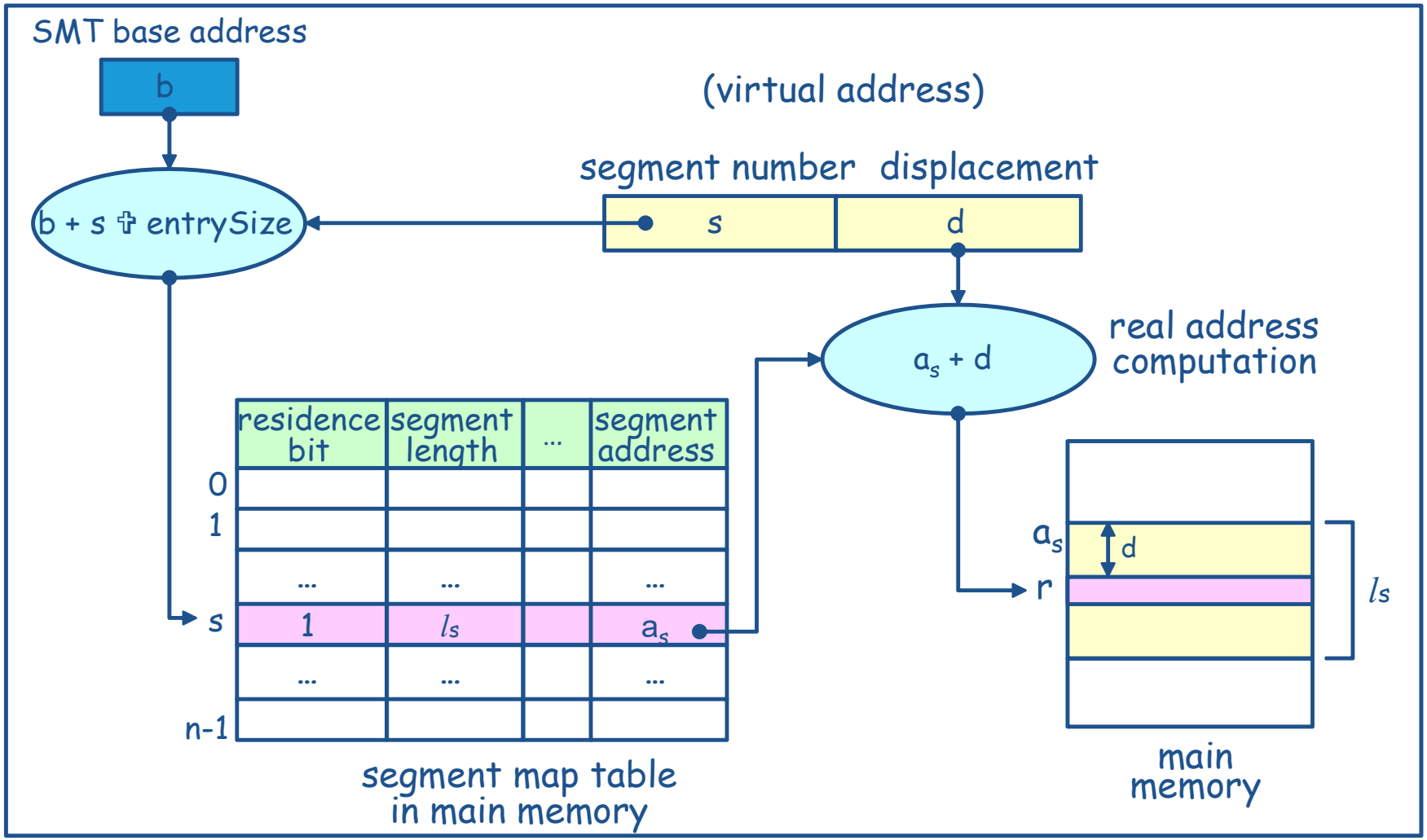
segment number	residence bit	secondary storage address	segment length	protection bits (R/W/X/A)	other fields	segment address in memory
0	1	S_0	l_0	RW	···	a_0
1	1	S_1	l_1	RW	···	a_1
2	0	S_2	l_2	RX	···	-
···	···	···	···	···	···	···
k	1	S_k	l_k	RX	···	a_k
···	···	···	···	···	···	···
n-1	0	S_{n-1}	l_{n-1}	RWA	···	-

- 존재 비트 : 해당 세그먼트가 주기억장치에 적재되어 있는지의 여부
- 보조기억장치 주소 : 해당 세그먼트가 존재하는 디스크 상의 주소
- 세그먼트 길이 : 각 세그먼트의 크기에 대한 정보
- 보호 비트 : 해당 세그먼트에 대해 프로세스가 접근할 수 있는 연산모드 정보
- 세그먼트 주소 : 해당 세그먼트가 적재되어 있는 주기억장치의 주소

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)

세그먼테이션 시스템의 직접 사상 기법



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

접근 제어 기법

❖ 접근 제어 기법

- 접근 제어 정의
  - 프로세스가 자신의 프로그램 코드나 데이터 등의 context에 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)

주기억장치 관리

- 세그먼테이션 시스템에서의 주기억장치 관리
  - 주기억장치의 동적인 관리방법
    - 초기에는 전체를 하나의 영역으로 설정
    - 세그먼트 적재 시마다 세그먼트의 크기에 맞게 분할하여 할당함
  - VPM 기법과 유사함
    - 다만, 세그먼트 단위로 주기억장치에 적재됨

주기억장치 관리

세그먼테이션 시스템의 주기억장치 관리 자료 구조

partition	start address	size	current process ID	segment number	storage protection key	other fields
0	. . .	. . .	Px	x1	. . .	. . .
1	. . .	. . .	none	. . .	. . .	. . .
2	. . .	. . .	Py	y1	. . .	. . .
4	. . .	. . .	Px	x2	. . .	. . .
5	. . .	. . .	none	. . .	. . .	. . .
6	. . .	. . .	. . .	. . .	. . .	. . .

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

페이징/세그먼테이션 혼합 기법

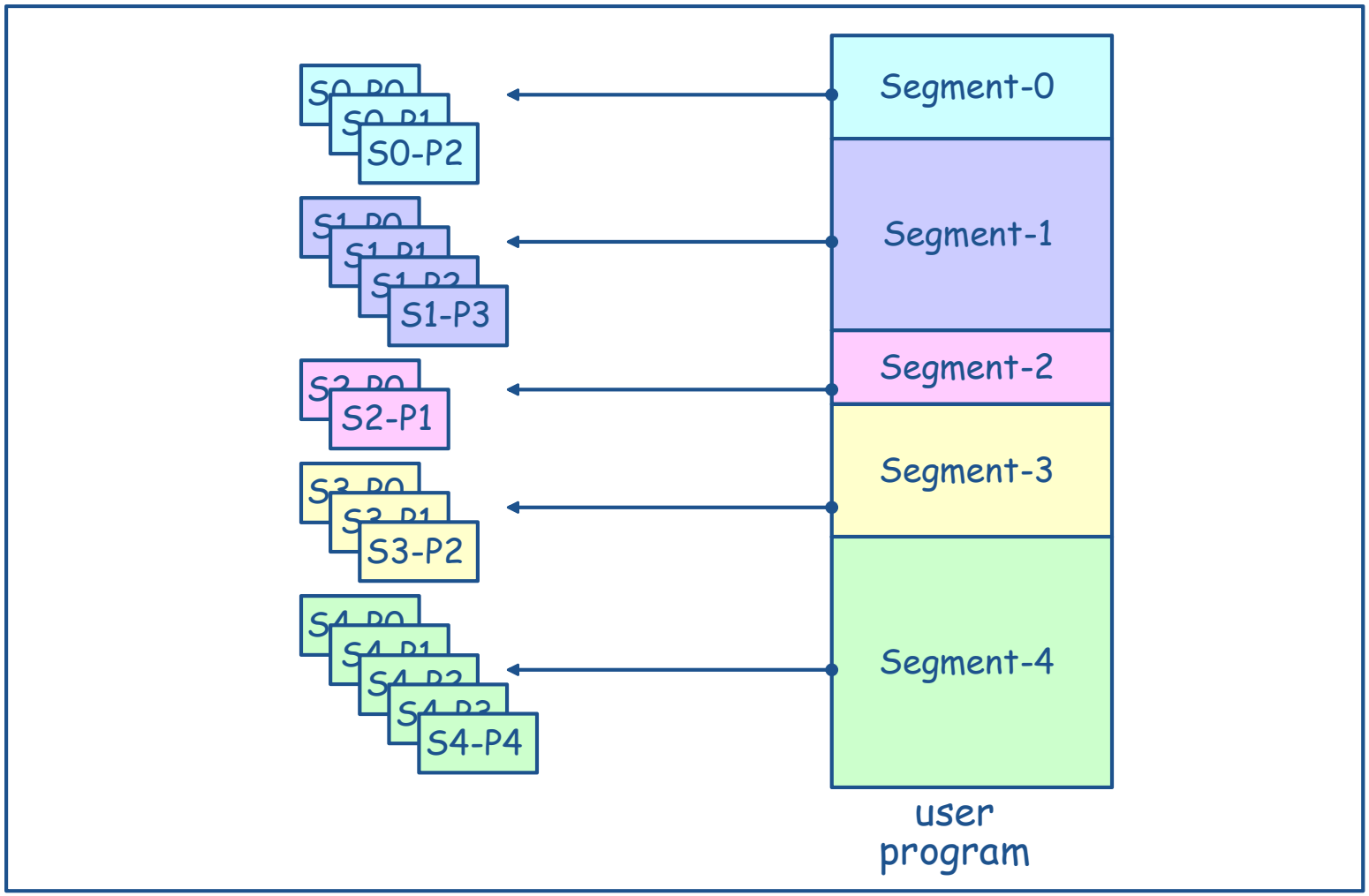
- 페이징 시스템
  - 장점 : 단순하고 오버헤드가 적은 기법
  - 단점 : 논리적인 분할의 개념 없음
    페이지 공유 등과 관련하여 복잡한 문제 발생함

- 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
  - 분할
    - 우선 사용자 프로그램을 논리적인 세그먼트 단위로 분할
    - 각 세그먼트들을 다시 페이지 단위로 분할
  - 적재
    - 분할된 페이지 단위로 주기억장치에 적재

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
  - SMT와 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 기법에서의 SMT

segment number	secondary storage address	segment length	protection bits (R/W/X/A)	other fields	PMT address
0	S_0	l_0	RW	...	b_0
1	S_1	l_1	RW	...	b_1
2	S_2	l_2	RX	...	b_2
...	...	...	...	...	...
k	S_k	l_k	RX	...	b_k
...	...	...	...	...	...
n-1	S_{n-1}	l_{n-1}	RWA	...	b_{n-1}

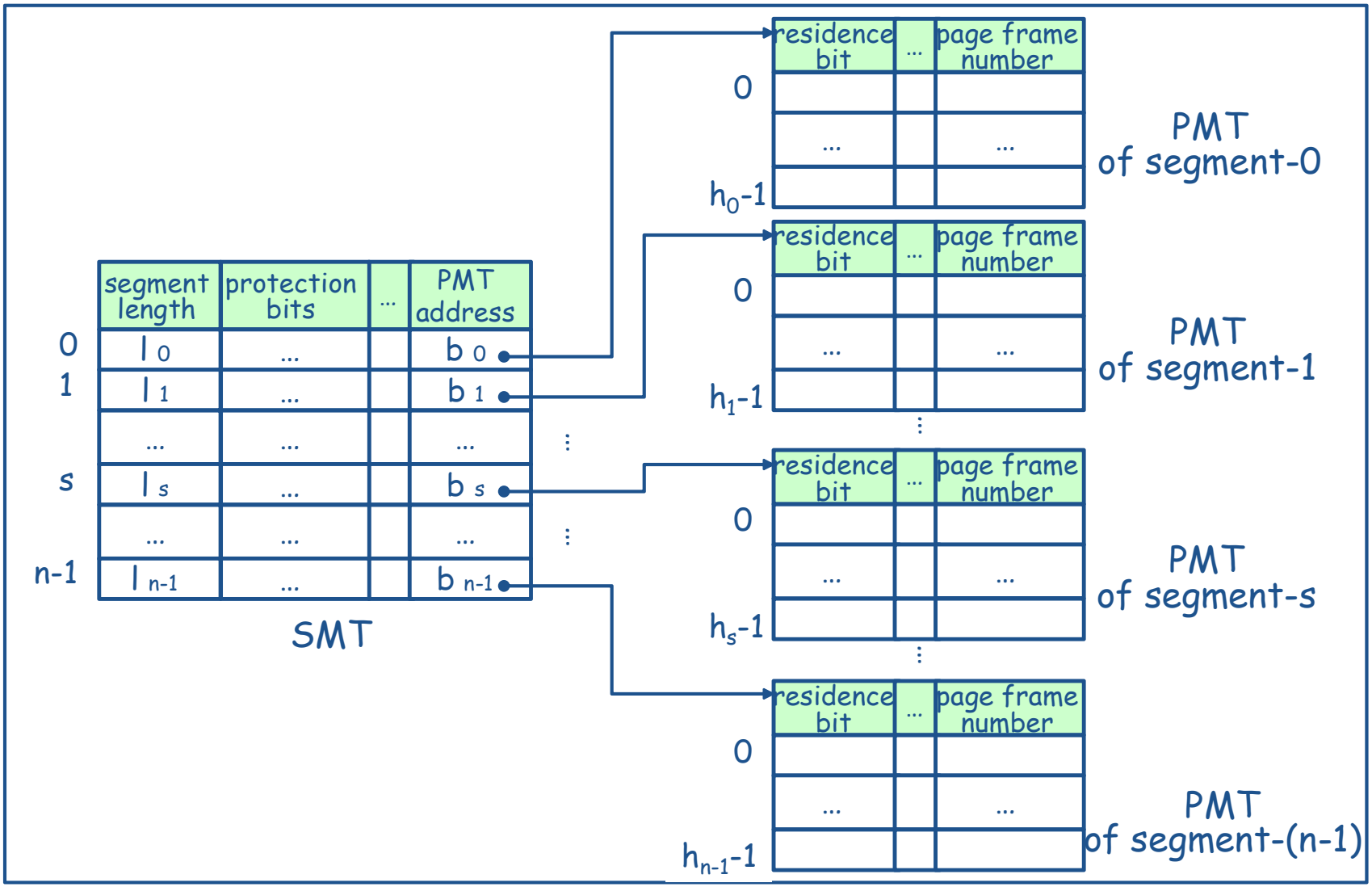
- 존재 비트 없음
  - 프로그램의 세그먼트 단위로 주기억장치에 적재되는 것이 아니고 각 세그먼트를 다시 분할한 페이지 단위로 주기억장치에 적재하기 때문

페이징/세그먼테이션 혼합 기법에서의 세그먼트-k에 대한 PMT

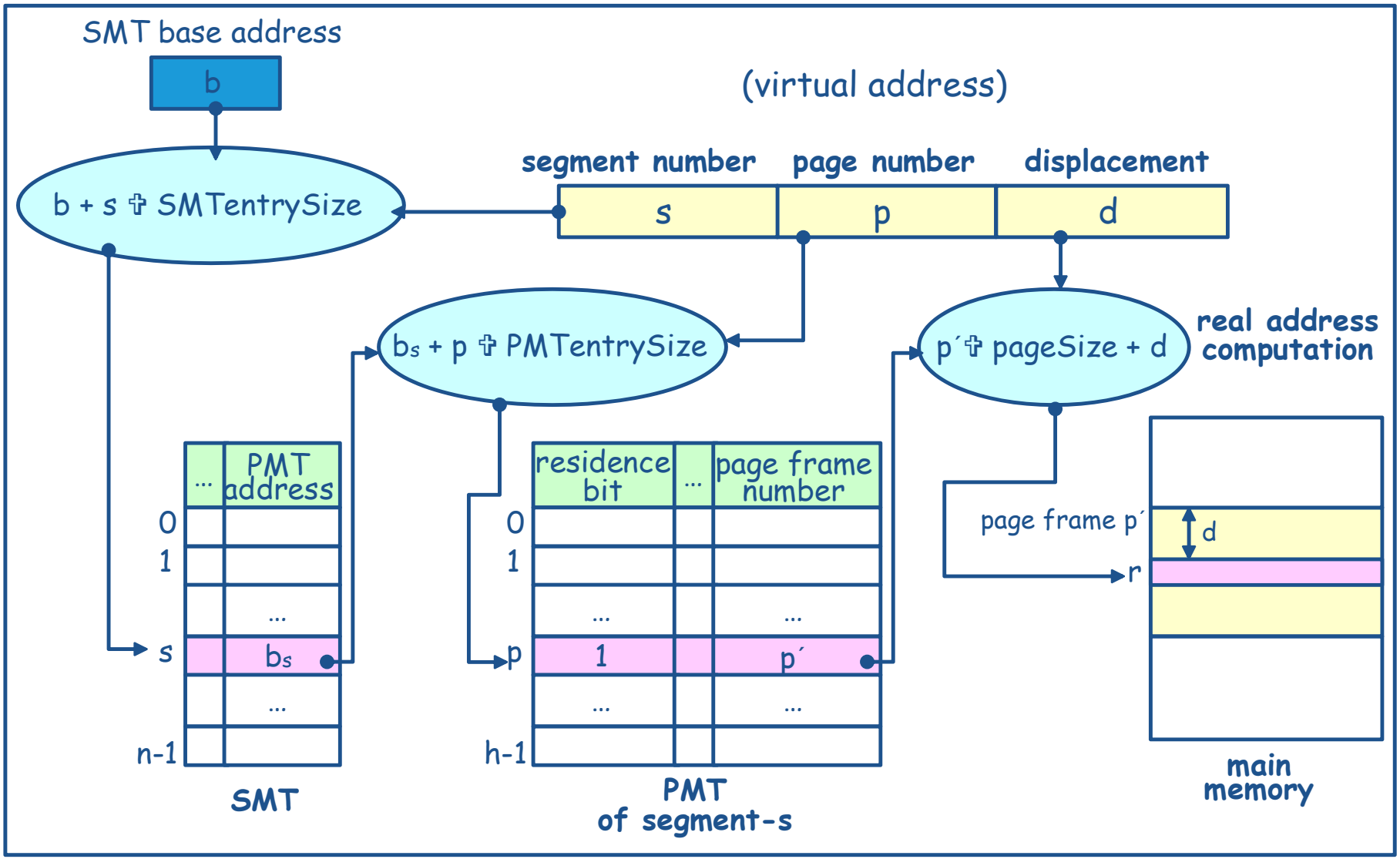
page number	residence bit	secondary storage address	other fields	page frame number
0	1	S_{k0}	. . .	P_{k0}
1	1	S_{k1}	. . .	P_{k1}
2	0	S_{k2}	. . .	-
. . .	. . .	...	. . .	. . .
i	1	S_{ki}	. . .	P_{ki}
. . .	. . .	...	. . .	. . .
h-1	0	$S_{k, h-1}$	. . .	-

- 프로그램의 세그먼트-k에 대한 PMT
  - 가정 : 세그먼트-k가 h개의 페이지들로 다시 분할되었음
  - h개의 각 페이지들에 대한 정보 기록

페이징/세그먼테이션 혼합 기법에서의 주소 사상 테이블



페이징/세그먼테이션 혼합 기법의 직접 사상 기법



페이징/세그먼테이션 혼합 기법에서의 직접 사상 알고리즘

- 가상 주소 v = (s, p, d)
  - SMTentrySize = SMT의 엔트리 크기, PMTentrySize = PMT의 엔트리 크기
  - pageSize = 페이지 크기

(1) 해당 프로세스의 SMT가 저장되어 있는 주소 b에 접근

(2) 접근한 SMT에서 세그먼트 s에 대한 엔트리 찾음
s의 엔트리 위치 = b + s * SMTentrySize

(3) 보호 비트 필드를 검사하여 허가되지 않은 연산일 경우
세그먼트 보호 예외 처리 모듈을 호출 // segment protection exception

(4) 찾아진 SMT 엔트리의 PMT 주소 필드값 b_s를 이용하여
해당 세그먼트의 PMT에 접근

(5) 접근한 PMT에서 페이지 p에 대한 엔트리 찾음
p의 엔트리 위치 = b_s + p * PMTentrySize

(6) 찾아진 PMT 엔트리의 존재 비트 검사
① 존재 비트가 0 인 경우
디스크로부터 해당 페이지를 주기억장치로 적재
PMT를 갱신한 후 (6)-②의 단계 수행
② 존재 비트가 1 인 경우
해당 엔트리에서 페이지 프레임 번호 p'를 인출

(7) 인출된 페이지 프레임 번호 p'와 가상 주소의 변위 d를 사용하여
실 주소 r 형성 (r = p' * pageSize + d)

(8) 실 주소 r로 주기억장치에 접근

페이징/세그먼테이션 혼합 기법에서의 연관 사상 기법

페이징/세그먼테이션 혼합 기법의 연관 사상을 위한 사상 테이블

key		residence bit	. . .	page frame number
s	p			
0	0			
0	1			
. . .	. . .			
0	Si			
1	0			
1	1			
. . .	. . .			
1	Sj			
. . .	. . .			
n-1	0			
n-1	1			
. . .	. . .			
n-1	Sk			

- 연관기억장치에 대한 탐색 키 = (s, p)
  - s = 세그먼트 번호, p = 페이지 번호
- 병렬 탐색

페이징/세그먼테이션 혼합 기법의 고려사항

- 사상 테이블의 개수와 전체적인 크기 증가
  - 기억장소의 소모 많음
  - 주소 사상에 많은 시간이 요구됨
- 직접 사상 기법을 사용할 경우
  - 시스템 성능이 1/3 수준으로 저하될 수 있음

결론

- 가상기억장치 개념
  - FPM 기법, VPM 기법 등과의 차이점
    - 사용자 프로그램을 분할된 블럭 단위로 주기억장치에 적재
    - 비연속적으로 적재

- 각 프로세스에게 할당되는 주기억장치 공간의 평균 크기가 작아짐
- 다중프로그래밍 정도가 높아짐
- 시스템 전체의 성능 향상을 기대할 수 있음

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