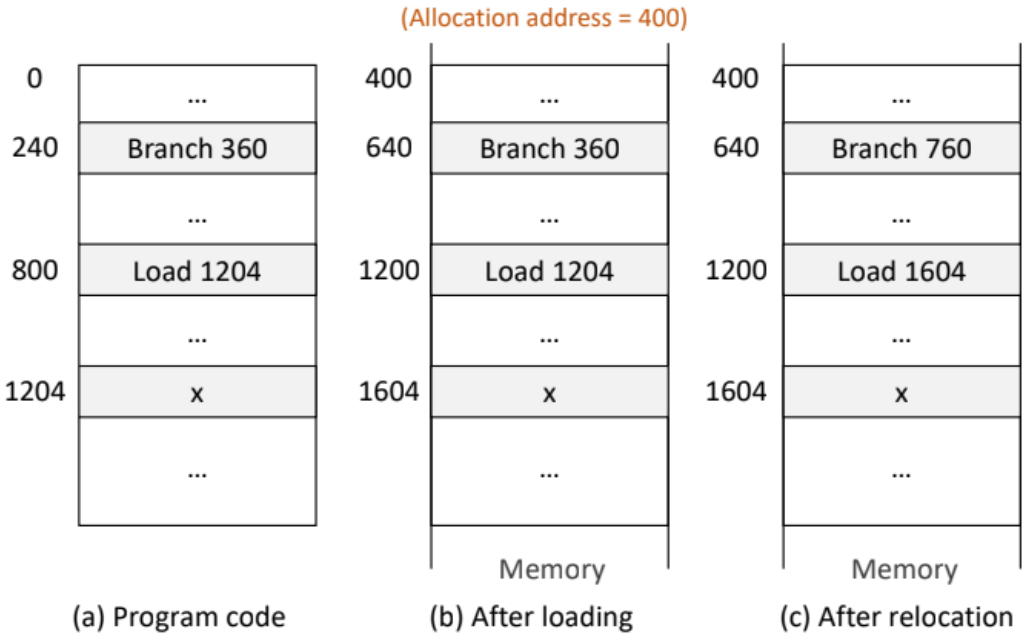
6주차 Virtual Memory

Virtual Storage (Memory)

- Non-continuous allocation (Memory Allocation)
- 사용자 프로그램을 여러 개의 block으로 분할
- 실행 시, 필요한 block들만 메모리에 적재
  - 나머지 block 들은 swap device에 존재
- 기법들
  - Paging system
  - Segmentation system
  - Hybrid paging/segmentation system

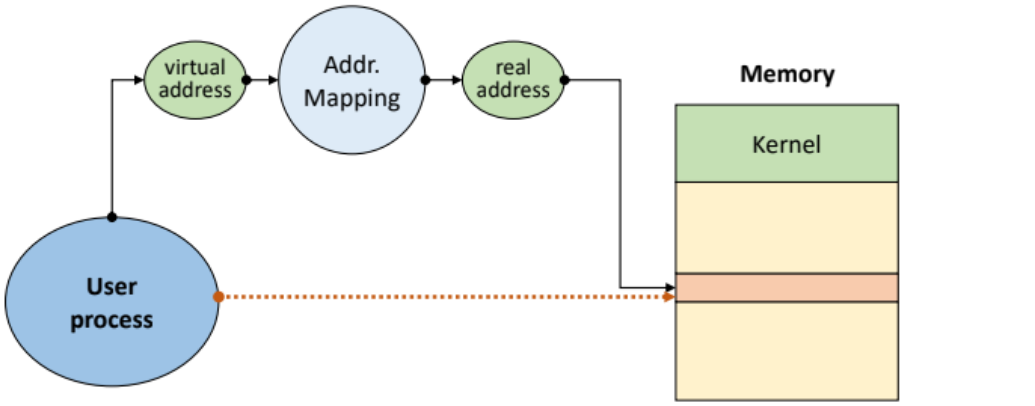
Address Mapping

- Continuous allocation
  - Relative address (상대 주소)
    - 프로그램의 시작 주소를 0으로 가정한 주소
  - Relocation (재배치)
    - 메모리 할당 후, 할당된 주소(allocation address)에 따라 상대 주소들을 조정하는 작업



  - 프로그램이 0번에서 시작한다고 가정하고 주소 할당 ⇒ 실제로 메모리에 올렸더니 시작 주소가 400이면 400을 더해주면 됨

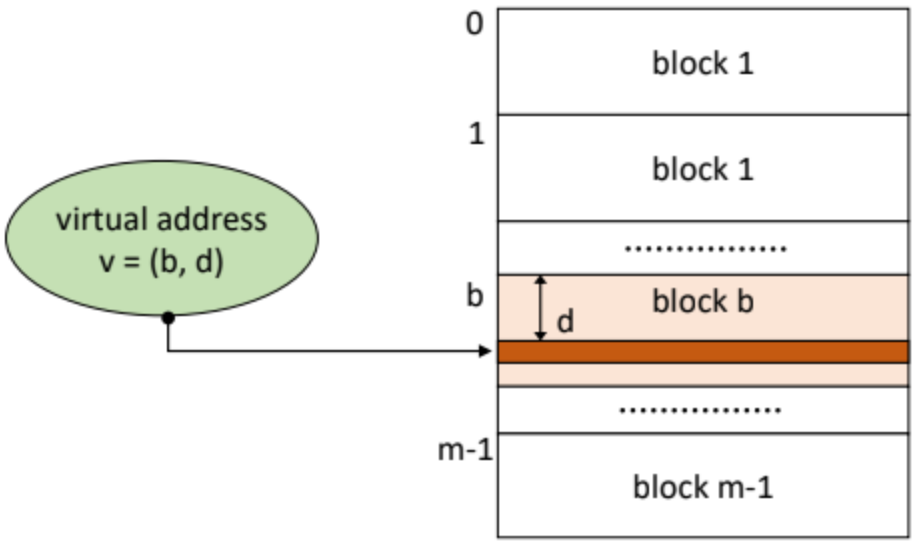
- **Non-continuous allocation**
  - Virtual address (가상주소) = relative address
    - Logical address (논리주소)
    - 연속된 메모리 할당을 가정한 주소
  - Real address (실제주소) = absolute (physical)
    - 실제 메모리에 적재된 주소
  - Address mapping : Virtual address를 real address로 바꿔주는 것



- 사용자/프로세스는 실행 프로그램 전체가 메모리에 연속적으로 적재되었다고 가정하고 실행 할 수 있음

Block Mapping

- Address Mapping 기법 중 하나
- 사용자 프로그램을 block 단위로 분할/관리
  - 각 block에 대한 address mapping 정보 유지
- Virtual address : v = (b,d)
  - b = block number
  - d = displacement(offset) in a block (시작점으로부터 얼마나 떨어져 있는가)

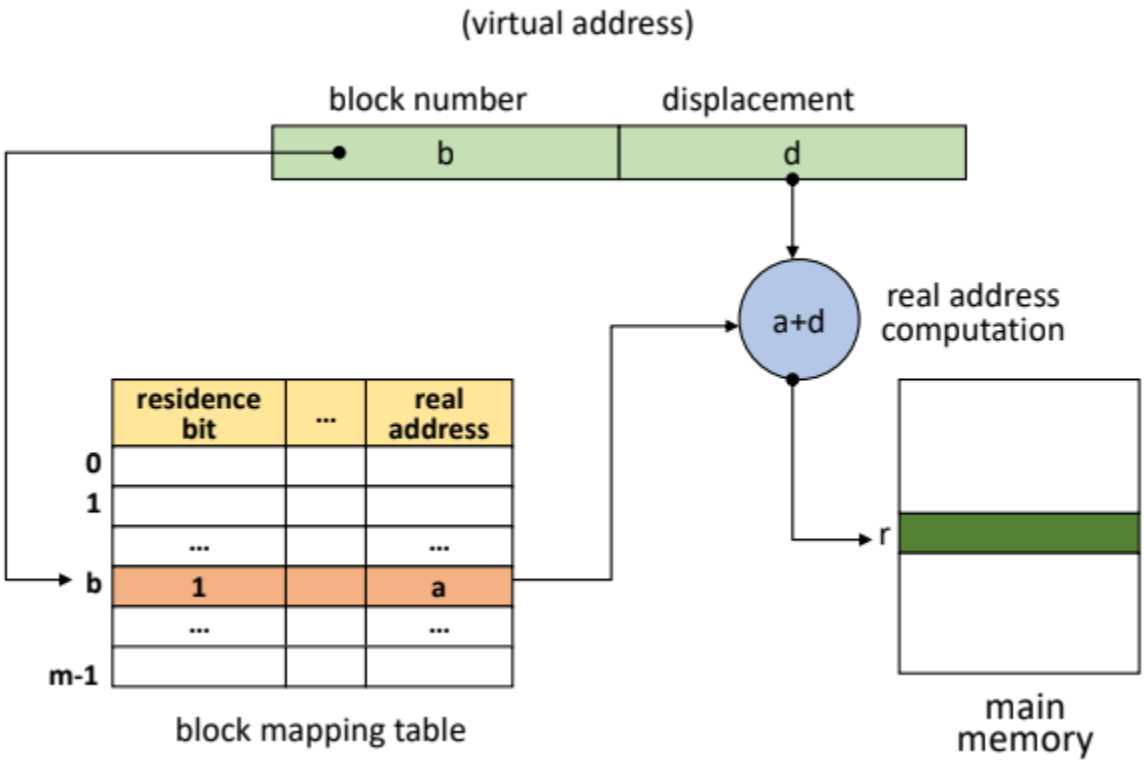


- Block map table (BMT)
  - Address mapping 정보 관리
  - Kernel 공간에 프로세스마다 하나의 BMT를 가짐

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

- Residence bit: 해당 블록이 메모리에 적재되었는지 여부 (0/1)

Block Mapping 과정



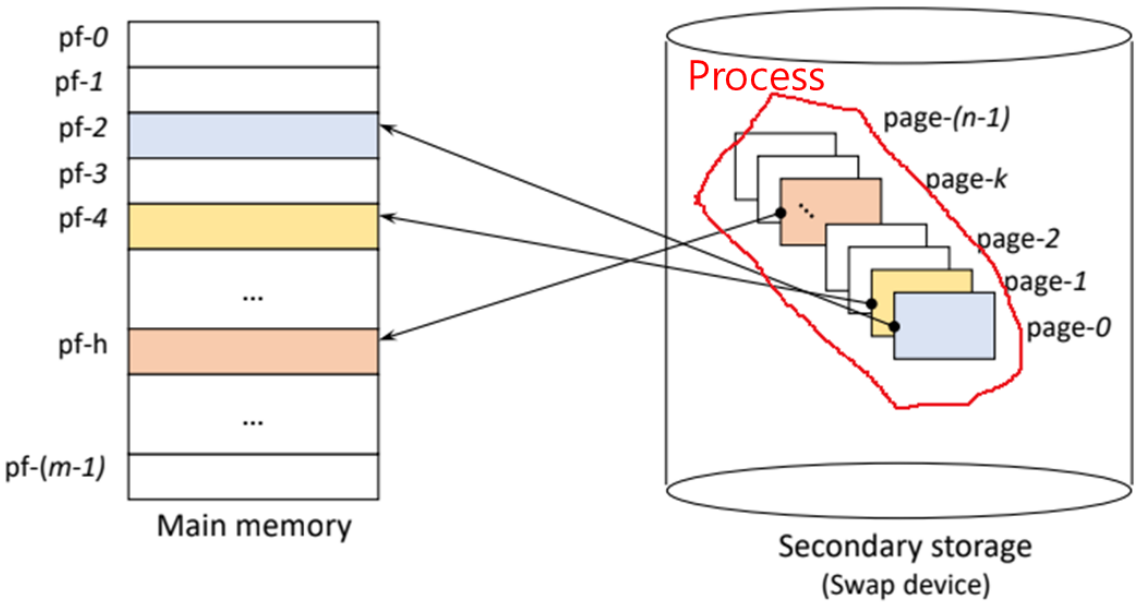
1. **프로세스의 BMT에 접근**
2. **BMT에서 block b에 대한 항목(entry)를 찾음**
3. **Residence bit 검사**
   ① Residence bit = 0 경우,
   swap device에서 해당 블록을 메모리로 가져 옴
   BTM 업데이트 후 3-② 단계 수행
   ② Residence bit = 1 경우,
   BMT에서 b에 대한 real address 값 a 확인
4. **실제 주소 r 계산 (r = a + d)**
5. **r을 이용하여 메모리에 접근**

Virtual Storage Methods

- Paging system
- Segmentation system
- Hybrid paging/segmentation system

Paging System

- 프로그램을 같은 크기의 블록으로 분할 (Pages)
- 용어
  - Page : 프로그램의 분할된 block
  - Page frame : 메모리의 분할 영역 (Page와 같은 크기로 분할)



- 특징
  - 논리적 분할이 아님 (크기에 따른 분할)
    - Segmentation에 비해 Page 공유(sharing) 및 보호(protection) 과정이 복잡함
  - Segmentation에 비해 Simple and Efficient
  - 외부 단편화 발생하지 않음 (page frame과 page의 크기가 같기 때문)
    - 내부 단편화는 발생 가능
  - 윈도우에서 실제로 사용

Paging System에서의 Address Mapping

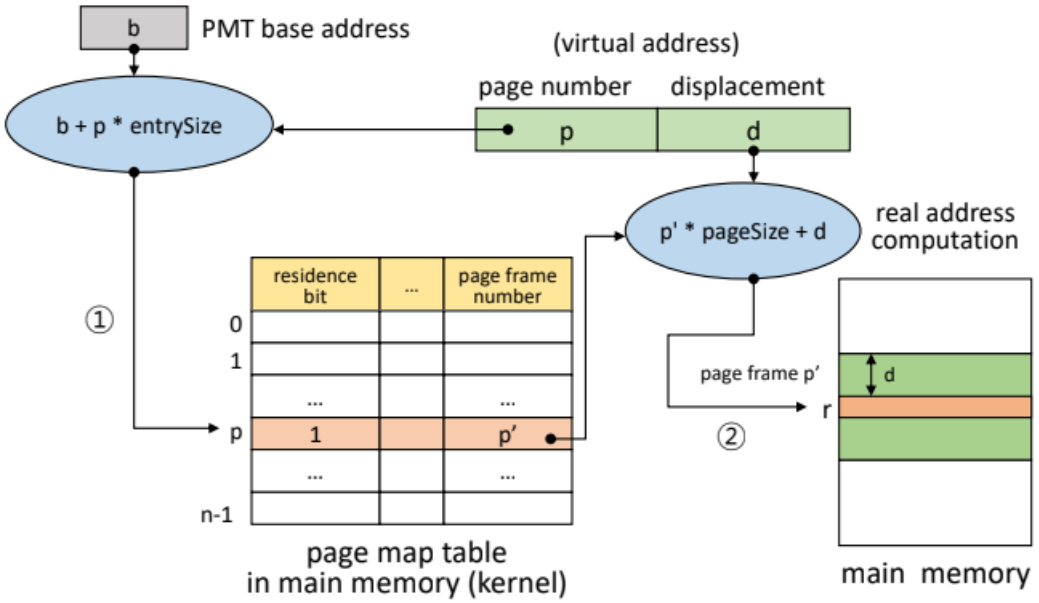
- Virtual address : v = (p,d)
  - p : page number
  - d : displacement(offset)
- Address mapping
  - PMT(Page Map Table) 사용

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}	...	-

    - residence bit : 메모리에 올라가 있으면 1, 아니면 0
    - page frame number : 메모리에 올라가 있다면 어디에 올라가 있는지
    - secondary storage address : swap device의 어디에 저장되어 있는지
- Address mapping mechanism
    - Direct mapping (직접 사상)
    - Associative mapping (연관 사상)
        - TLB(Translation Look-aside Buffer)
    - Hybrid direct/associative mapping

Direct Mapping

- Block mapping 방법과 유사
- 가정
    - PMT를 커널 안에 저장
    - PMT entry size = entrySize
    - Page size = pageSize



1. 해당 프로세스의 PMT가 저장되어 있는 주소 b에 접근

2. 해당 PMT에서 page p에 대한 entry 찾음

- p의 entry 위치 = b + p * entrySize

3. 찾아진 entry의 존재 비트 검사

Context switching 발생 (I/O)
➔Overhead

① Residence bit = 0 인 경우 **(page fault)**,
swap device에서 해당 page를 메모리로 적재
PMT를 갱신한 후 3-② 단계 수행

② Residence bit = 1인 경우,
해당 entry에서 page frame 번호 p'를 확인

4. pf와 가상 주소의 변위 d를 사용하여 실제 주소 r 형성

- r = p' * pageSize + d

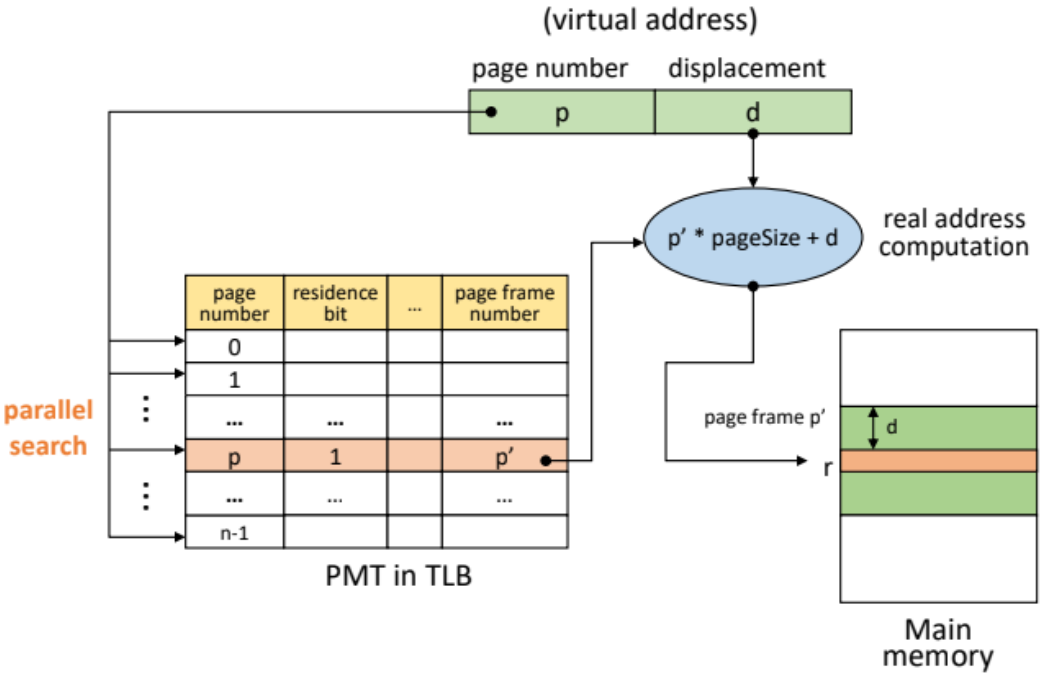
5. 실제 주소 r로 주기억장치에 접근

- 문제점
  - 메모리 접근 횟수가 2배 ⇒ 성능 저하 (performance degradation)
  - PMT를 위한 메모리 공간 필요
- 해결방안
  - **Associative mapping (TLB)**

- **PMT를 위한 전용 기억장치(공간) 사용**
  - **Dedicated register or cache memory**
- Hierarchical paging
- Hashed page table
- Inverted page table

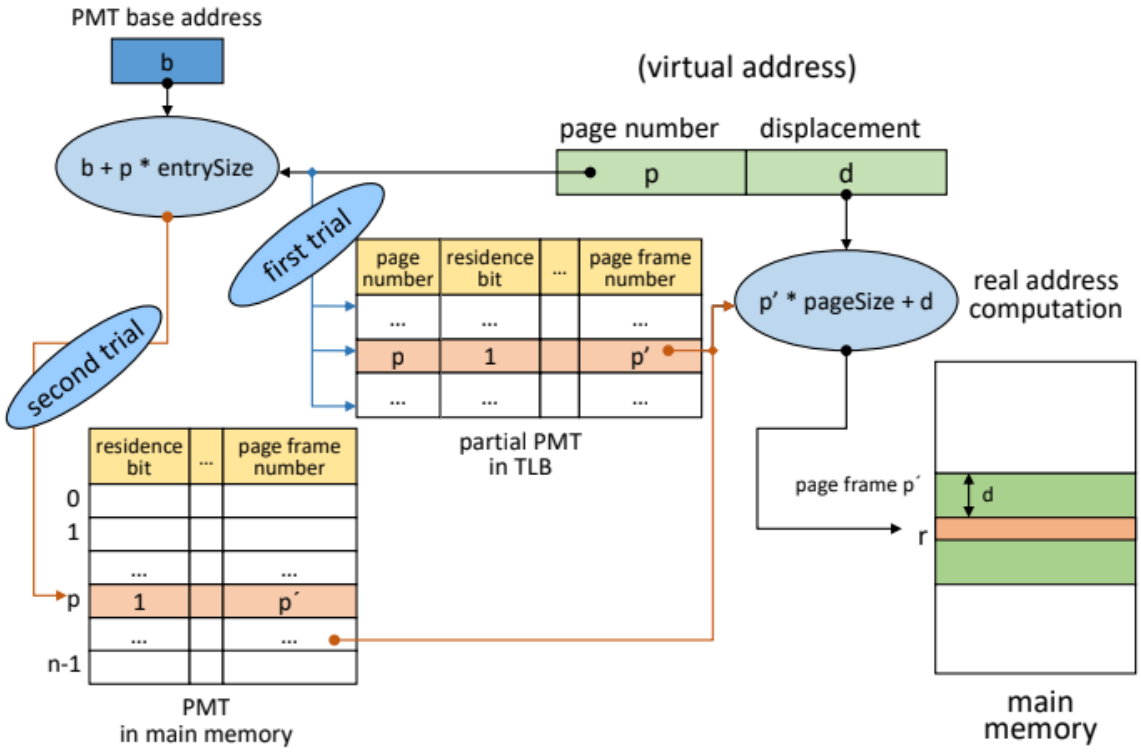
Associative Mapping

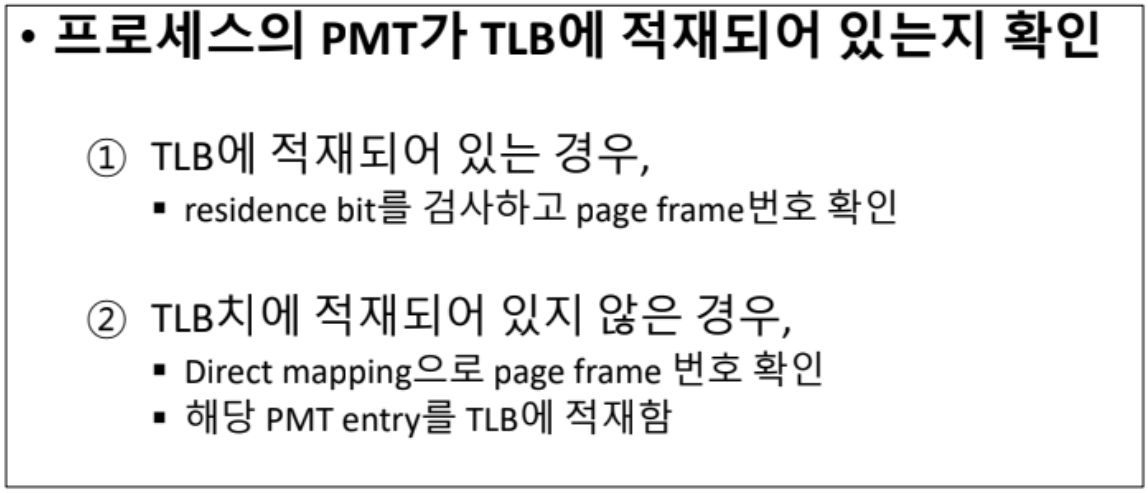
- TLB(Translation Look-aside Buffer)에 PMT 적재
  - Associative high-speed memory
  - PMT를 탐색하기 위한 전용 하드웨어
- PMT를 병렬 탐색
- Low overhead, high speed
- Expensive hardware
  - 큰 PMT를 다루기가 어려움



Hybrid Direct/Associative Mapping

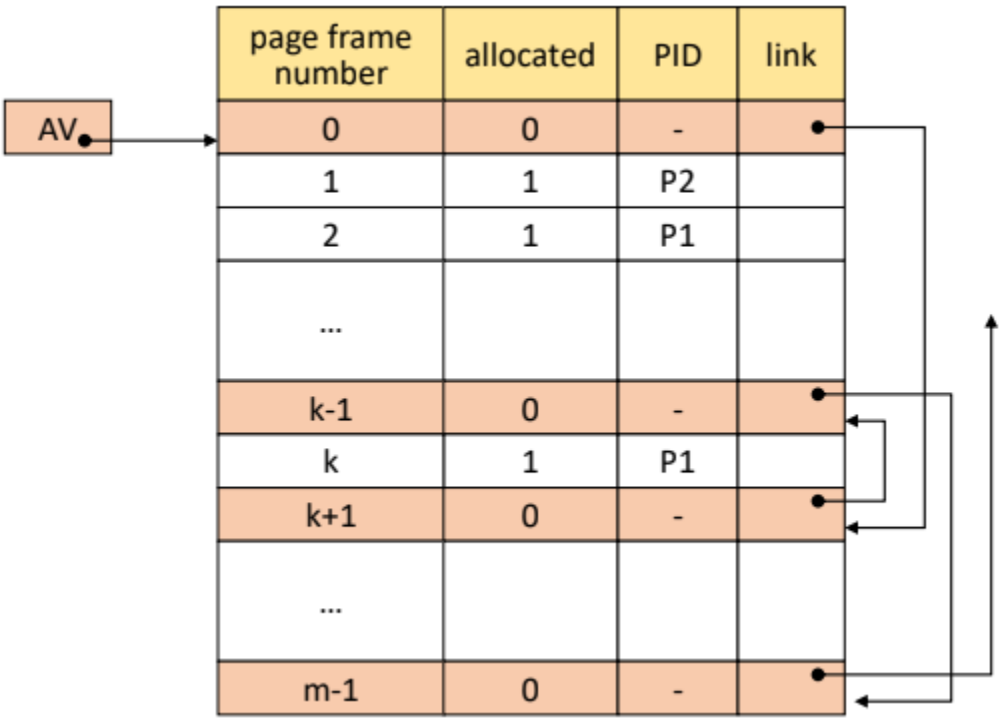
- 두 기법을 혼합하여 사용 (Direct Mapping + Associative Mapping)
  - HW 비용은 줄이고, Associative mapping의 장점 활용
- 작은 크기의 TLB 사용
  - PMT : 메모리(커널 공간)에 저장
  - TLB : PMT 중 일부 entry들을 적재
    - 최근에 사용된 page들에 대한 entry 저장
  - Locality (지역성) 활용
    - 프로그램의 수행과정에서 한번 접근한 영역을 다시 접근 (temporal locality) 또는 인접 영역을 다시 접근(spatial locality)할 가능성이 높음





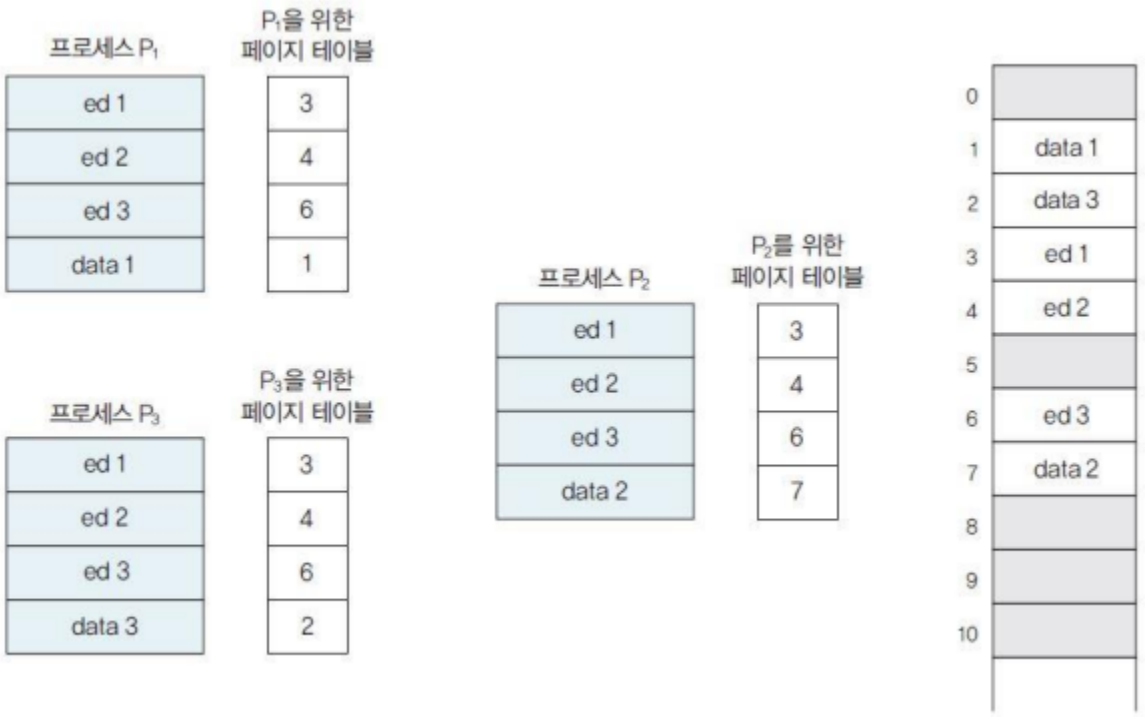
Paging System에서의 Memory Management

- Page와 같은 크기로 미리 분할 하여 관리/사용
  - Page frame
  - FPM 기법과 유사
- Frame table
  - Page frame당 하나의 entry
  - 구성
    - Allocated/available field : page frame이 할당되었는지 / 사용할 수 있는지
    - PID field : 어떤 page가 올라와 있는지
    - Link field : For free list (사용가능 한 fp들을 연결)
    - AV : Free list header (free list의 시작점)

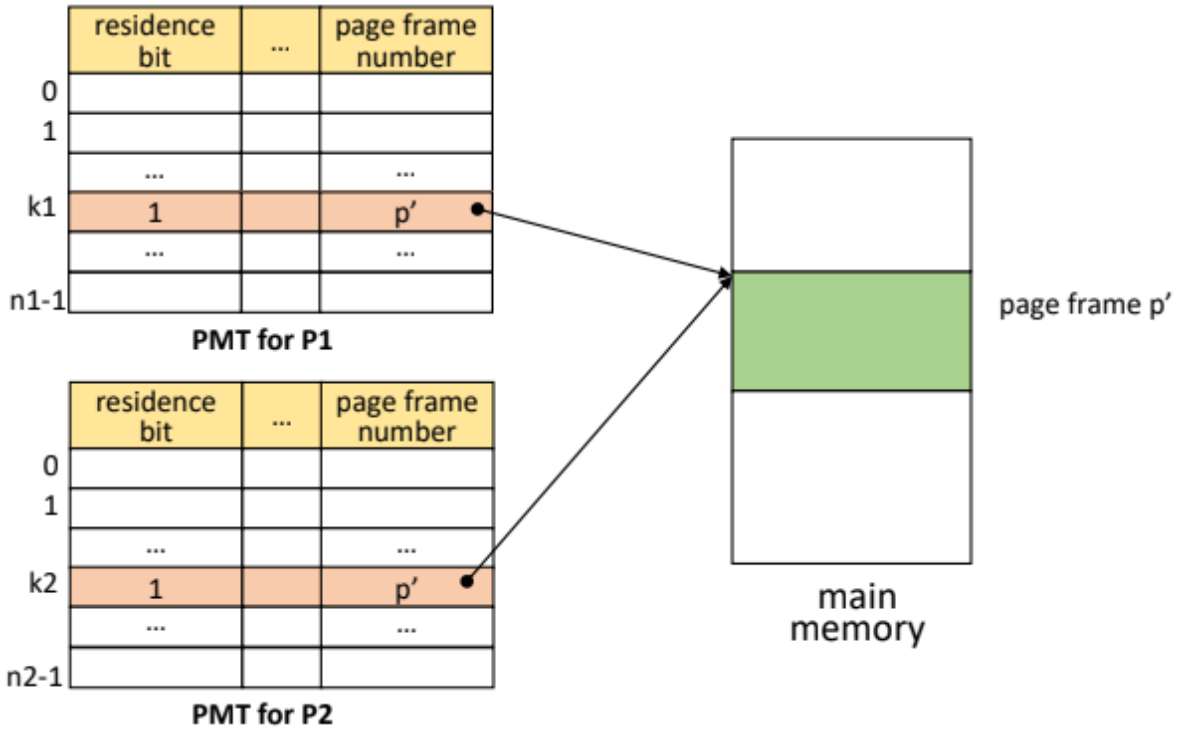


Paging System에서의 Page Sharing

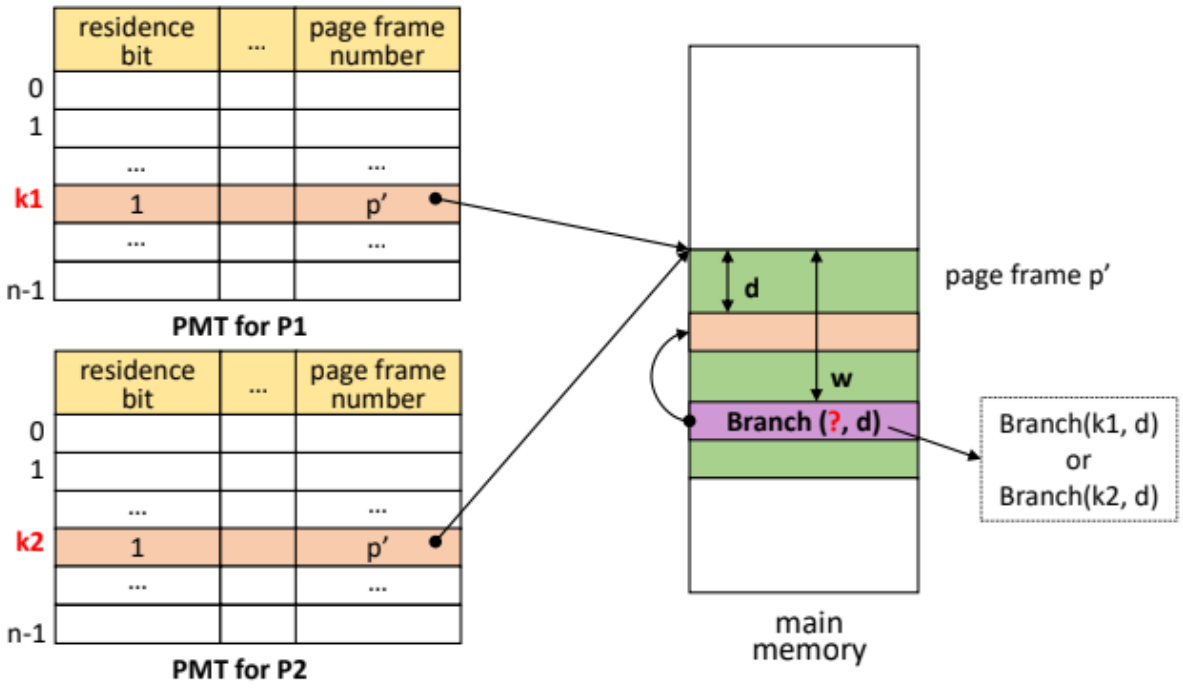
- 여러 프로세스가 특정 page를 공유 가능
  - Non-continuous allocation이기 때문에 가능
- 공유 가능 page
  1. Procedure pages
     - Pure code (reenter code)
  2. Data page
     - Read-only data
     - Read-write data
       - 병행성(concurrency) 제어 기법 관리하에서만 가능
- 예시 - Editor 프로그램을 3명이 사용하는 경우



- Data page sharing

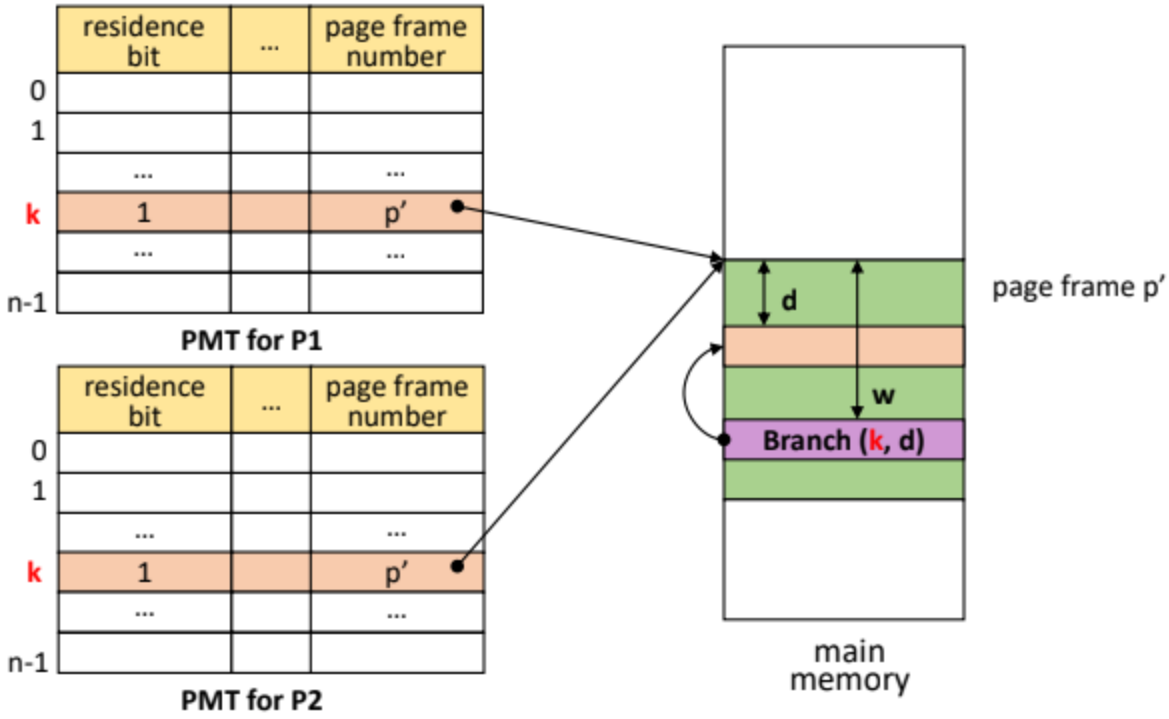


- Procedure Page Sharing (Problem)



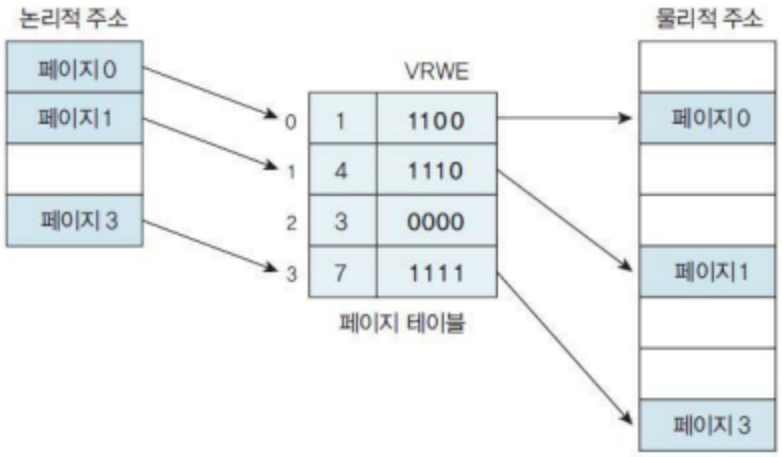
  - 같은 곳에 있는데 서로 다른 페이지 번호를 가리킴

- Procedure Page Sharing (Solution)
  - 프로세스들이 shared page에 대한 정보를 PMT의 같은 entry에 저장하도록 함 ⇒ 같은 이름을 부여한다.



Paging System에서의 Page Protection

- 여러 프로세스가 page를 공유 할 때 ⇒ Protection bit 사용



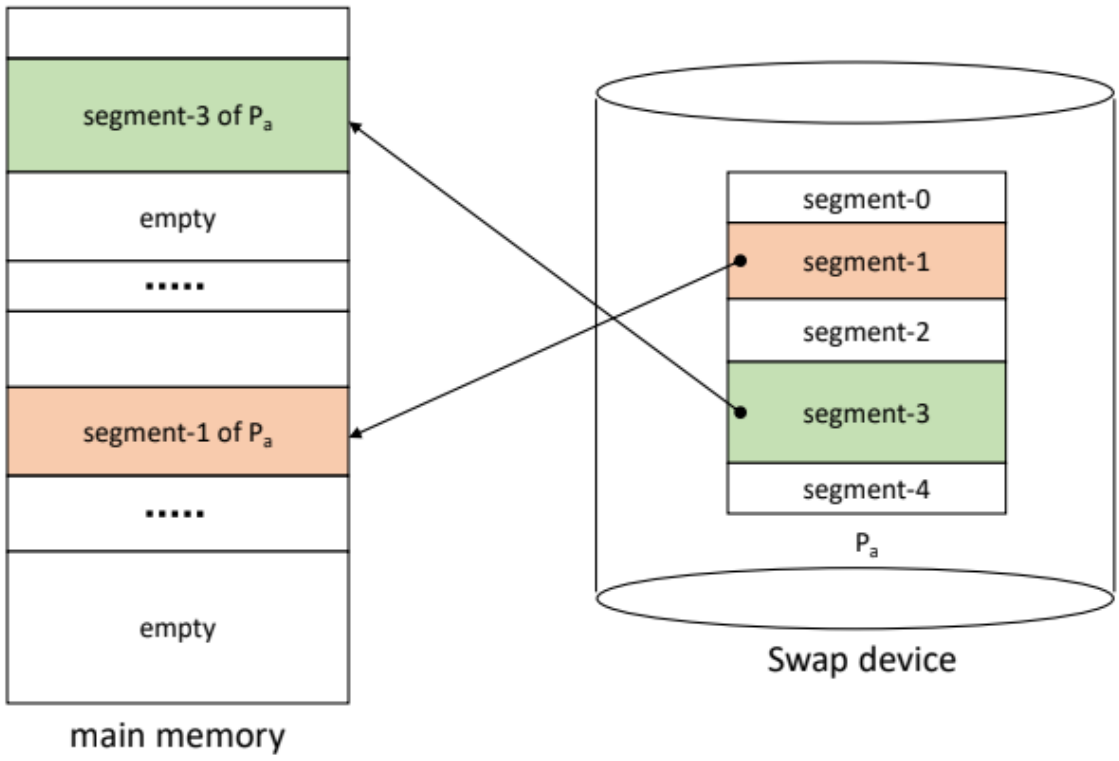
- 타당 · 비타당(V) 비트 : 메인 메모리의 적재 여부
- 읽기(R) 비트 : 읽기 여부
- 쓰기(W) 비트 : 수정 여부
- 실행(E) 비트 : 실행 여부

Paging System – Summary

- 프로그램을 고정된 크기의 block으로 분할 (page) / 메모리를 block size로 미리 분할 (page frame)
  - 외부 단편화 문제 없음
  - 메모리 통합/압축 불필요
  - 프로그램의 논리적 구조 고려하지 않음 ⇒ Page sharing/protection이 복잡
- 필요한 page만 page frame에 적재하여 사용
  - 메모리의 효율적 활용
- Page mapping overhead
  - 메모리 공간 및 추가적인 메모리 접근이 필요
  - 전용 HW 활용으로 해결 가능 ⇒ 하드웨어 비용 증가

Segmentation System

- 프로그램을 논리적 block으로 분할 (segment)
  - Block의 크기가 서로 다를 수 있음
  - 예) stack, heap, main procedure, shared lib, Etc.
- 특징
  - 메모리를 미리 분할 하지 않음
    - VPM과 유사
  - Segment sharing/protection이 용이함
  - Address mapping 및 메모리 관리의 overhead가 큼
  - 내부 단편화 발생하지 않음 (미리 분할해놓는 것이 아니라 필요한만큼 분할하기 때문)
    - 외부 단편화는 발생 가능



Segmentation System에서의 Address Mapping

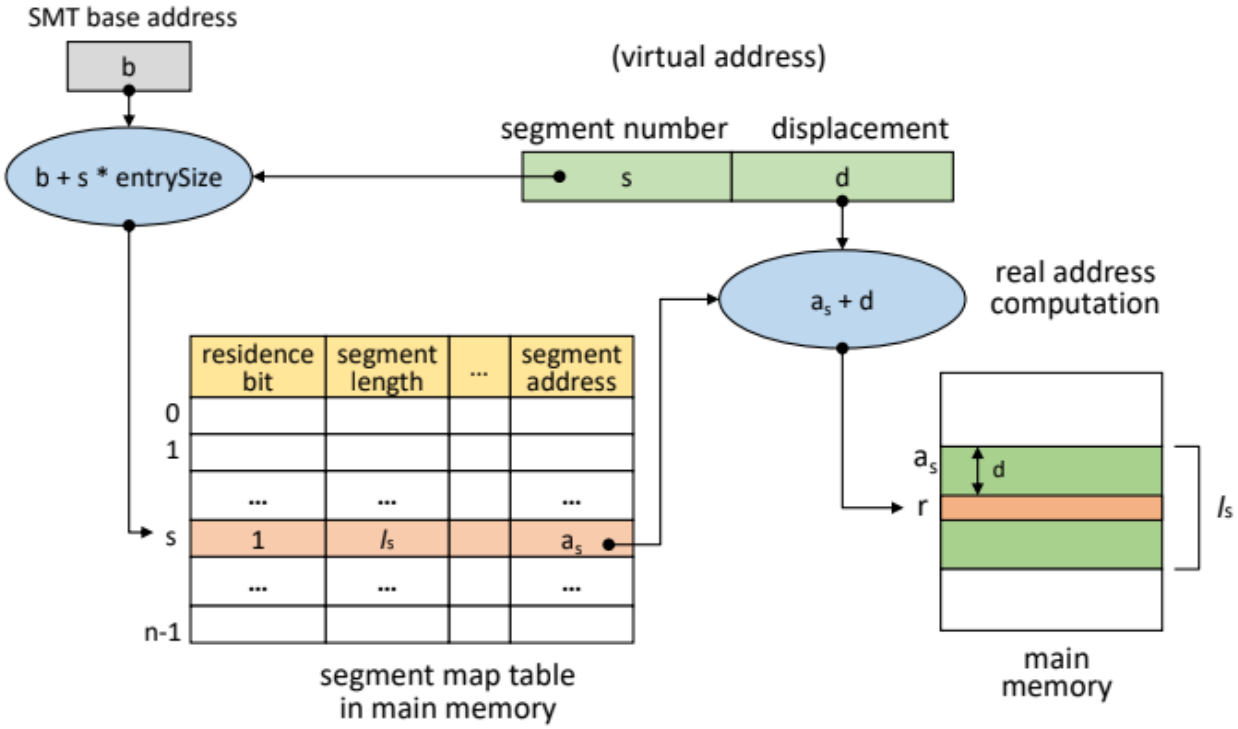
- Virtual address: v = (s , d)
  - s : segment number
  - d : displacement in a segment
- Segment Map Table (SMT)

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	···	-

- sement length : segment의 길이가 다 다르므로 길이를 기록해놔야 함
- protection bits : 논리적으로 나눠진 부분에 대해서 프로세스가 갖는 권한을 기록

- Address mapping mechanism
  - Paging system과 유사

Segmentation System에서의 Direct Mapping



1. **프로세스의 SMT가 저장되어 있는 주소 b에 접근**
2. **SMT에서 segment s의 entry 찾음**
   - s의 entry 위치 = b + s * entrySize
3. **찾아진 Entry에 대해 다음 단계들을 순차적으로 실행**
   ① 존재 비트가 0 인 경우,
   // missing segment fault
   swap deive로부터 해당 segment를 메모리로 적재
   SMT를 갱신
   ② 변위(d)가 segment 길이보다 큰 경우 ($d > l_s$),
   segment overflow exception 처리 모듈을 호출
   ③ 허가되지 않은 연산일 경우 (protection bit field 검사),
   segment protection exception 처리 모듈을 호출
4. **실제 주소 r 계산 ($r = a_s + d$)**
5. **r로 메모리에 접근**

Segmentation System에서의 Memory management

- VPM과 유사
  - Segment 적재 시, 크기에 맞추어 분할 후 적재

<Partition table or State table>

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	...	...	...	...	...	...

Segmentation System에서의 Segment sharing/protection

- 논리적으로 분할되어 있어, 공유 및 보호가 용이함

Segmentation System – Summary

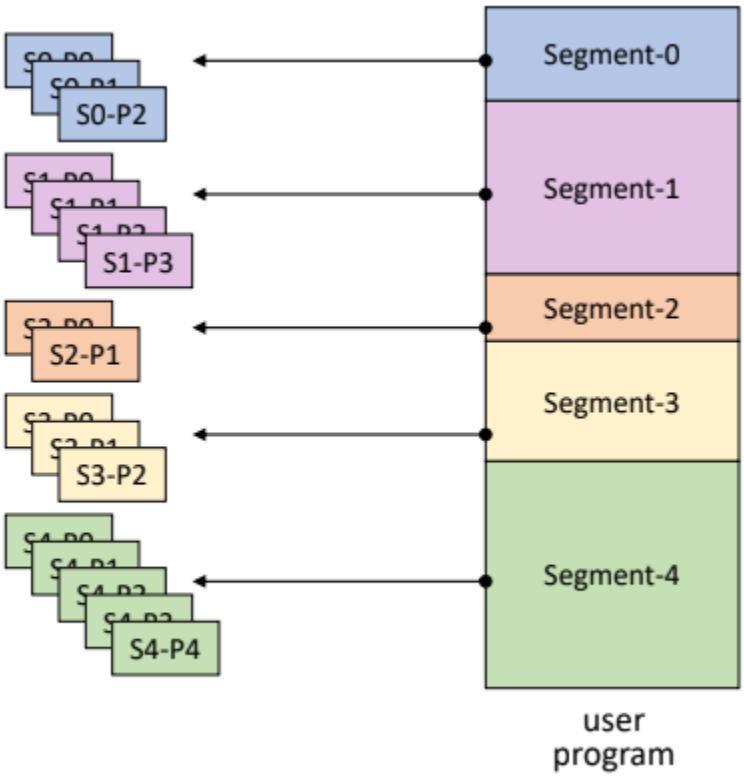
- 프로그램을 논리 단위로 분할 (segment) / 메모리를 동적으로 분할
  - 내부 단편화 문제 없음
  - Segment sharing/protection이 용이함
  - Paging system 대비 관리 overhead가 큼
- 필요한 segment만 메모리에 적재하여 사용
  - 메모리의 효율적 활용
- Segment mapping overhead
  - 메모리 공간 및 추가적인 메모리 접근이 필요
  - 전용 HW 활용으로 해결 가능

Paging vs Segmentation

Paging system	Segmentation System
간편하다, 관리의 오버헤드가 작다	관리의 오버헤드가 크다
논리 단위로 분할한 것이 아니기 때문에 공유 / 보호가 복잡하다	논리 단위로 분할한 것이 아니기 때문에 공유 / 보호가 쉽다

Hybrid Paging/Segmentation

- Paging과 Segmentation의 장점 결합
- 프로그램 분할
  1. 논리 단위의 segment로 분할
  2. 각 segment를 고정된 크기의 page들로 분할
- Page단위로 메모리에 적재



Hybrid Paging/Segmentation에서의 Address Mapping

- Virtual address : v = (s, p, d)
  - s : segment number
  - p : page number
  - d : offset in a page
- SMT와 PMT 모두 사용

- 각 프로세스마다 하나의 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}

* No residence bit

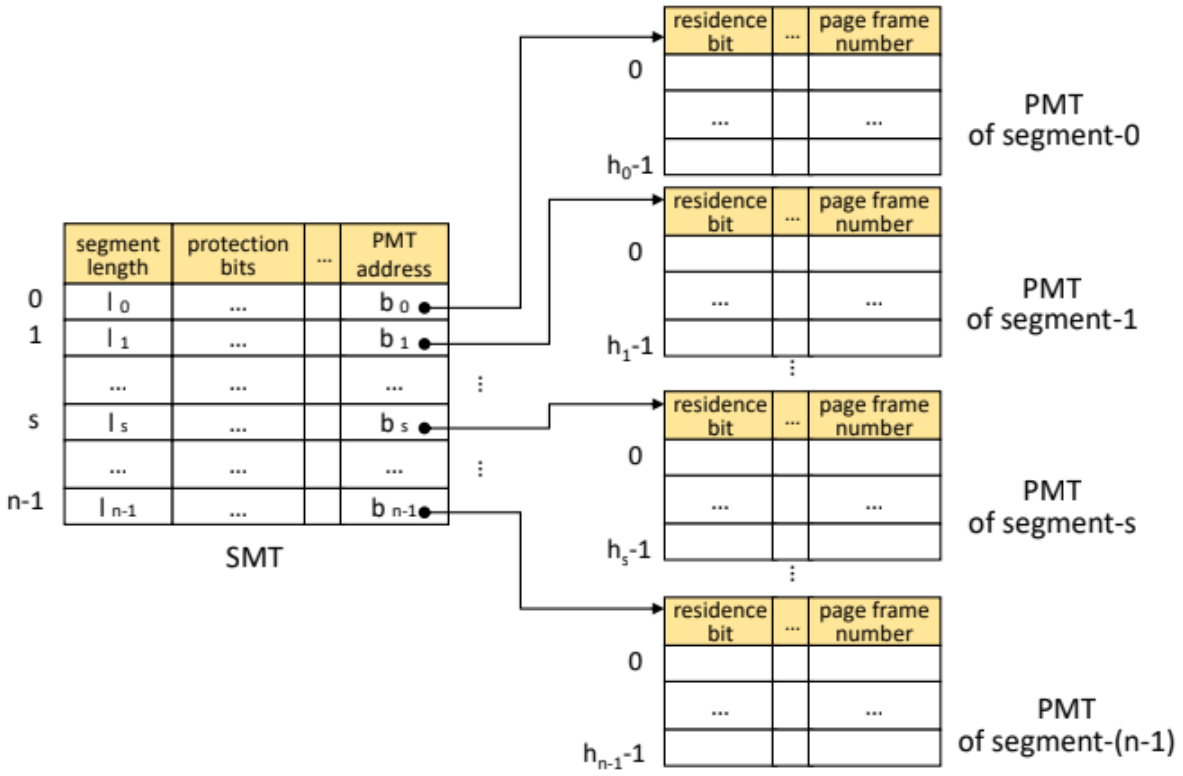
  - 메모리에는 segment가 아닌 페이지가 올라가기 때문에 residence bit가 필요없다.
  - PMT address : 각 segment의 PMT 가 어디에 있는지

- 각 segment마다 하나의 PMT

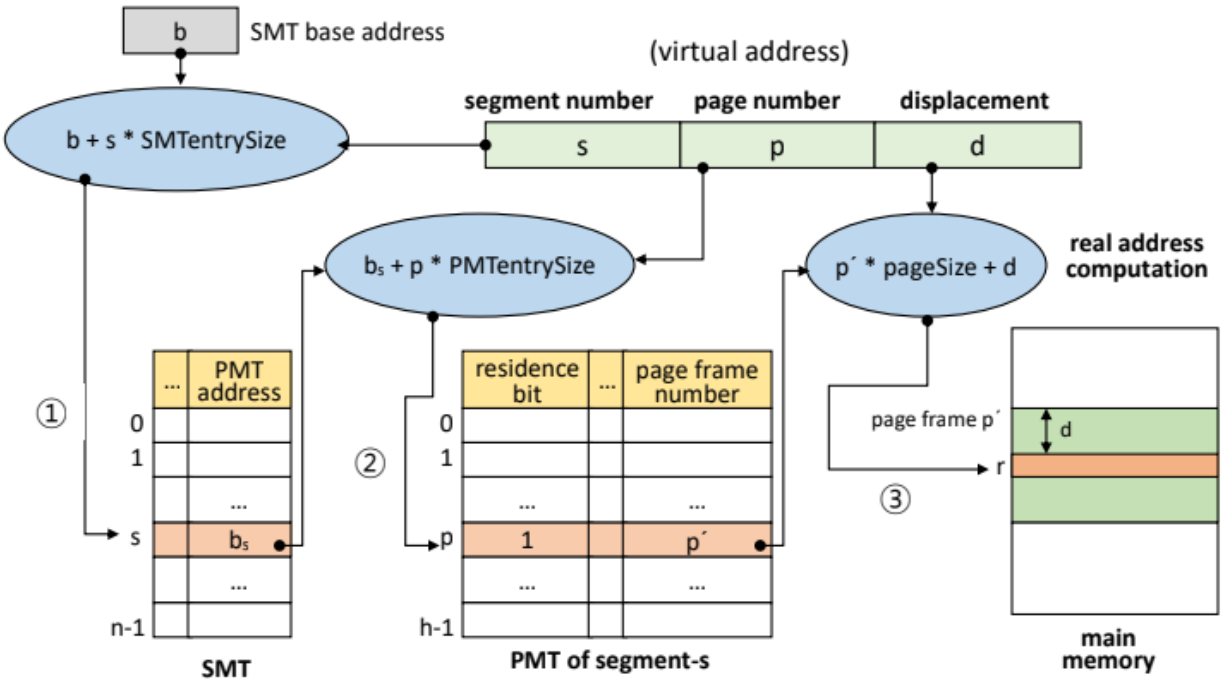
page number	**residence bit**	secondary storage address	other fields	page frame number
0	1	S_{k0}	···	P_{k0}
1	1	S_{k1}	···	P_{k1}
2	0	S_{k2}	···	-
···	···	...	···	···
i	1	S_{ki}	···	P_{ki}
···	···	...	···	···
h-1	0	$S_{k, h-1}$	···	-

- Address mapping
  - Direct, associated 등
- 메모리 관리
  - FPM과 유사

- Address mapping tables



Hybrid Paging/Segmentation에서의 Direct Mapping



Segmentation System – Summary

- 논리적 분할(segment)와 고정 크기 분할(page)을 결합
  - Page sharing/protection이 쉬움
  - 메모리 할당/관리 overhead가 작음
  - 외부 단편화 발생하지 않음
    - 내부 단편화는 발생 가능
- 전체 테이블의 수 증가
  - 메모리 소모가 큼
  - Address mapping 과정이 복잡
- Direct mapping의 경우, 메모리 접근이 3배
  - 성능이 저하될 수 있음