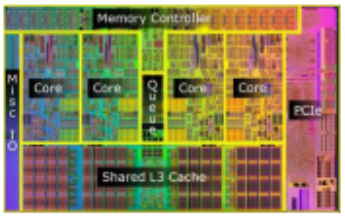
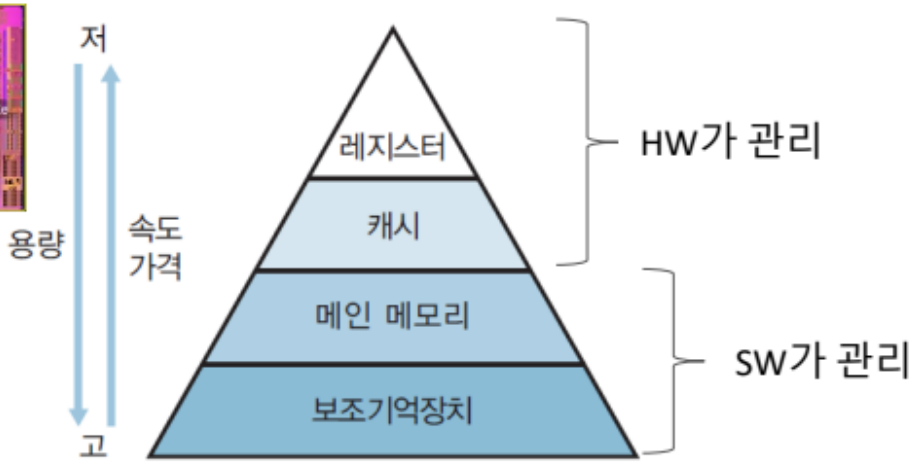
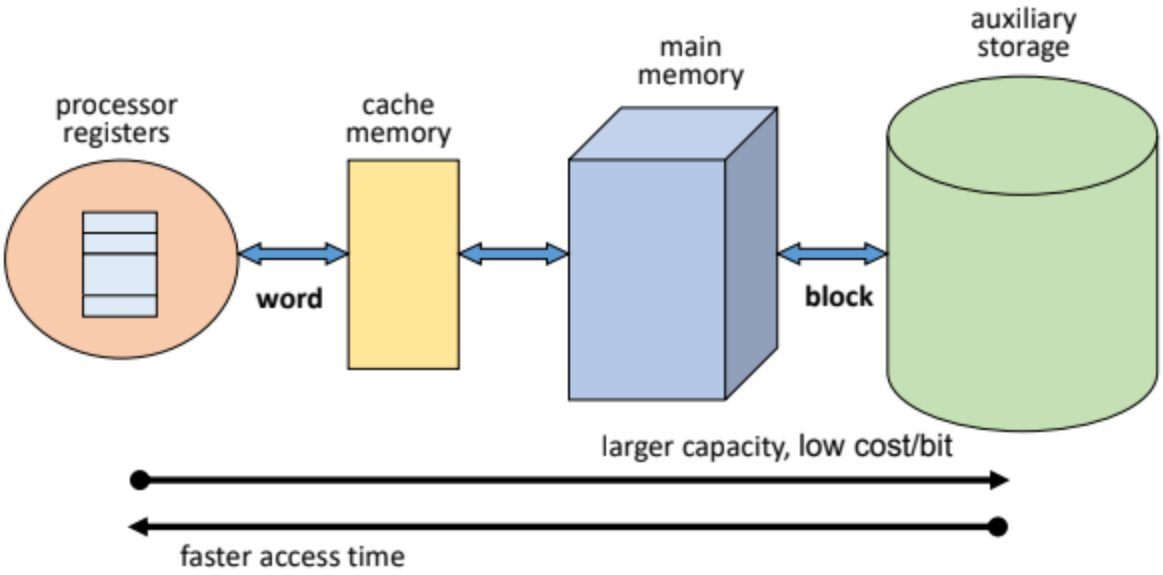
5주차 Memory Management

메모리(기억장치) 종류





메모리 계층구조



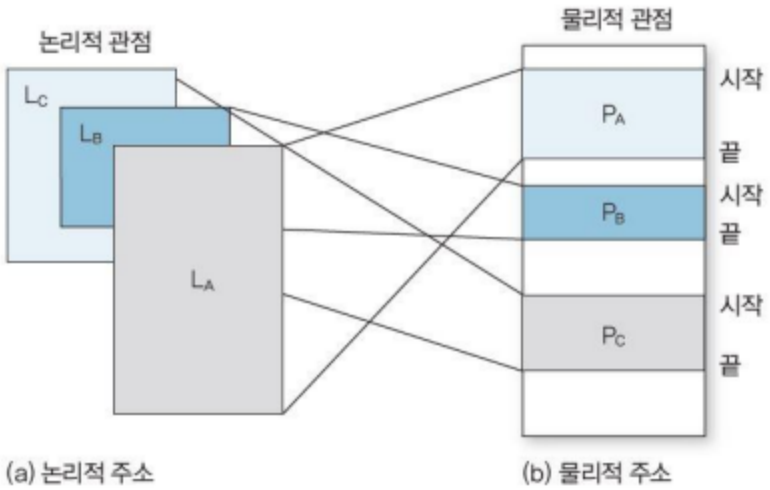
- Block
  - 보조기억장치와 주기억장치 사이의 데이터 전송 단위

- Size: 1 ~ 4KB

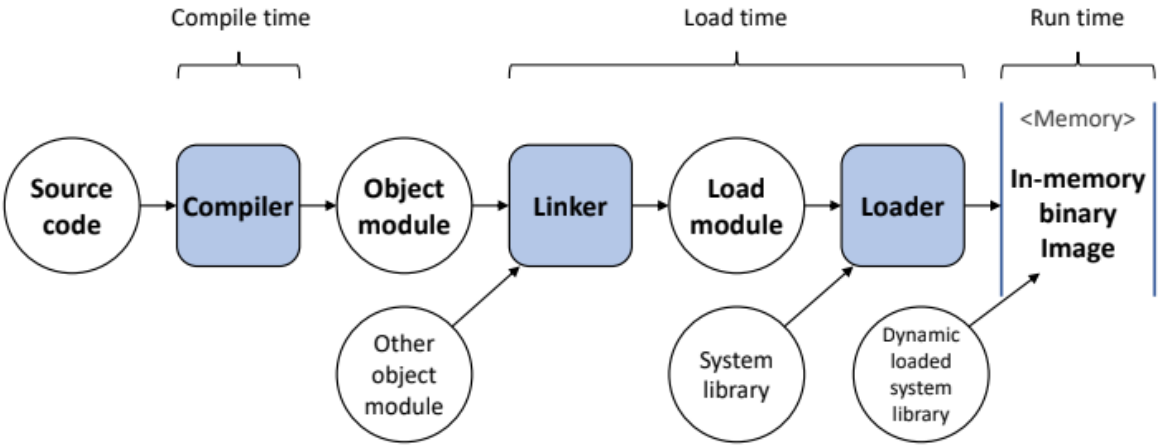
- Word
  - 주기억장치와 레지스터 사이의 데이터 전송 단위
  - Size: 16 ~ 64 bits

Address Binding

- 프로그램의 논리 주소를 실제 메모리의 물리 주소로 매핑(mapping)하는 작업

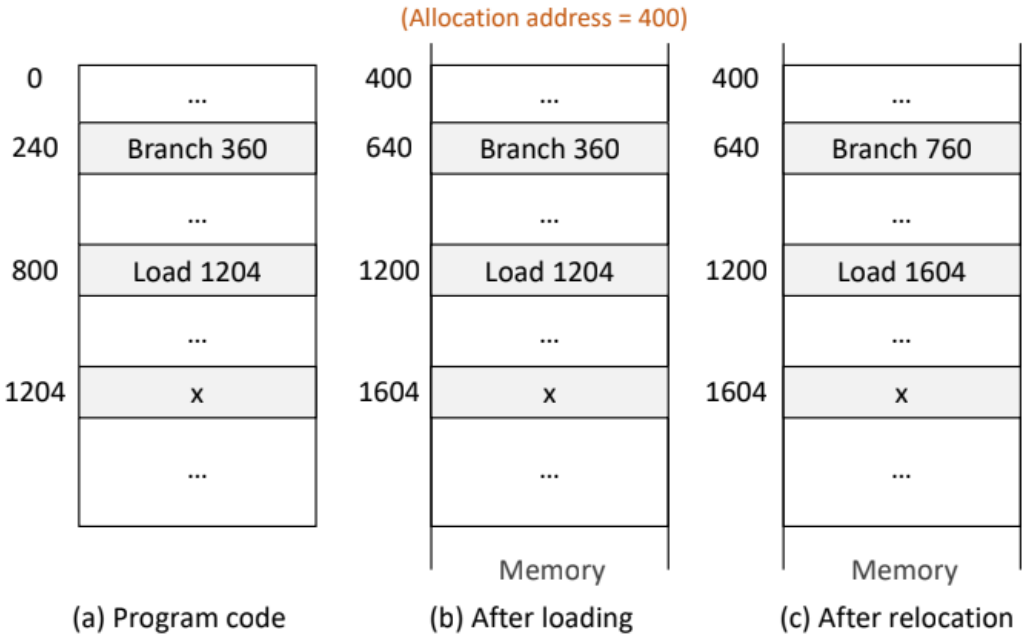


- Binding 시점에 따른 구분



compile : 소스 코드를 object module로 변환
linking : object module을 묶어서 load module로 변환
loading : load module을 메모리에 올림

- Compile time binding
  - 프로세스가 메모리에 적재될 위치를 컴파일러가 알 수 있는 경우, 위치가 변하지 않음
  - 프로그램 전체가 메모리에 올라가야 함
- Load time binding
  - 메모리 적재 위치를 컴파일 시점에서 모르면, 대체 가능한 상대 주소를 생성
  - 적재 시점(load time)에 시작 주소를 반영하여 사용자 코드 상의 주소를 재설정
  - 프로그램 전체가 메모리에 올라가야 함



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

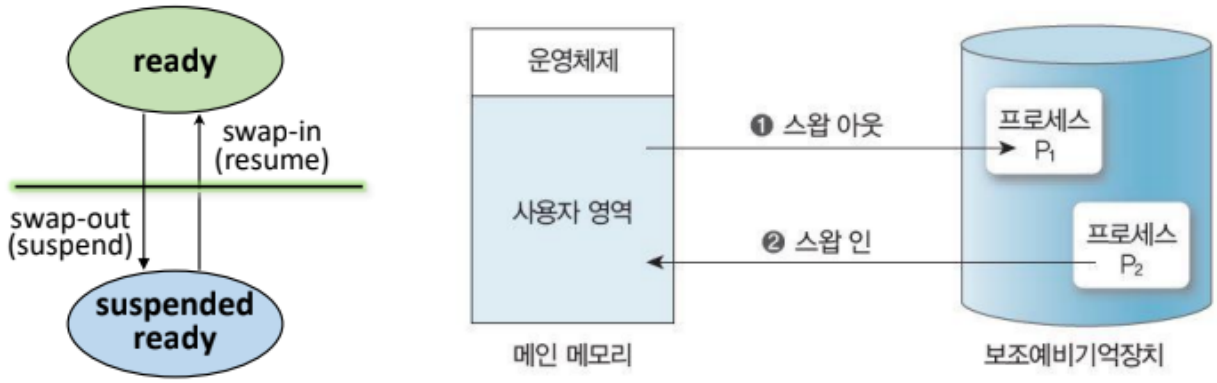
- Run time binding
  - Address binding을 수행 시간까지 연기 (ready 상태에서 running으로 넘어갈 때)
    - 프로세스가 수행 도중 다른 메모리 위치로 이동할 수 있음
  - HW의 도움이 필요
    - MMU: Memory Management Unit
  - 대부분의 OS가 사용

Dynamic Loading

- 모든 루틴을 교체 가능한 형태로 디스크에 저장
- 실제 호출 전까지는 루틴을 적재하지 않음
  - 메인 프로그램만 메모리에 적재하여 수행
  - 루틴의 호출 시점에 address binding 수행
- 장점 : 메모리 공간의 효율적 사용

Swapping

- 프로세서 할당이 끝나고 수행 완료 된 프로세스는 swap-device로 보내고 (Swap-out) 새롭게 시작하는 프로세스는 메모리에 적재 (Swap-in)



Memory Allocation

- Continuous Memory Allocation (연속할당)
  - Uni-programming
  - Multi-programming
    - Fixed partition (FPM)
    - Variable partition (VPM)
- Non-continuous Memory Allocation (비연속할당)

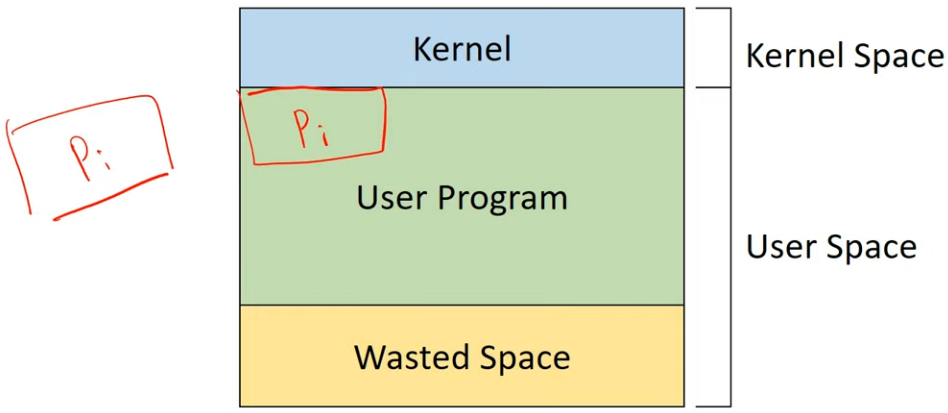
Continuous Memory Allocation

- 프로세스 (context)를 하나의 **연속된** 메모리 공간에 할당하는 정책
  - 프로그램, 데이터, 스택 등
- 메모리 구성 정책
  - 메모리에 동시에 올라갈 수 있는 프로세스 수 (= Multiprogramming degree)
  - 각 프로세스에게 할당되는 메모리 공간 크기
  - 메모리 분할 방법

- Uni-programming
  - Multiprogramming degree = 1
- Multi-programming
  - Fixed(static) partition multi-programming (FPM, 고정 분할)
  - Variable(dynamic) partition multi-programming (VPM, 가변 분할)

Uni-Programming

- 하나의 프로세스만 메모리 상에 존재
- 가장 간단한 메모리 관리 기법



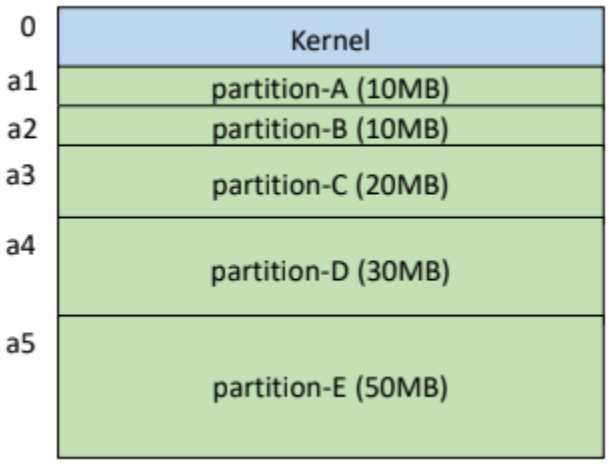
- **문제점 1**
  - 프로그램의 크기 > 메모리 크기
- 해결법
  - Overlay structure
    - 메모리에 현재 필요한 영역만 적재
    - 사용자가 프로그램의 흐름 및 자료구조를 모두 알고 있어야 함
- **문제점 2**
  - 커널(Kernel) 보호 → 프로그램을 올릴 때 커널을 침범하지 않아야 함
- 해결방법

- 경계 레지스터 (boundary register) 사용

- **문제점 3**
  - 시스템과 자원의 활용도가 낮다 → 공간이 낭비된다 → 퍼포먼스가 낮다
- 해결법
  - Multi-programming

Fixed Partition Multiprogramming

- 메모리 공간을 고정된 크기로 분할
  - 미리 분할되어 있음

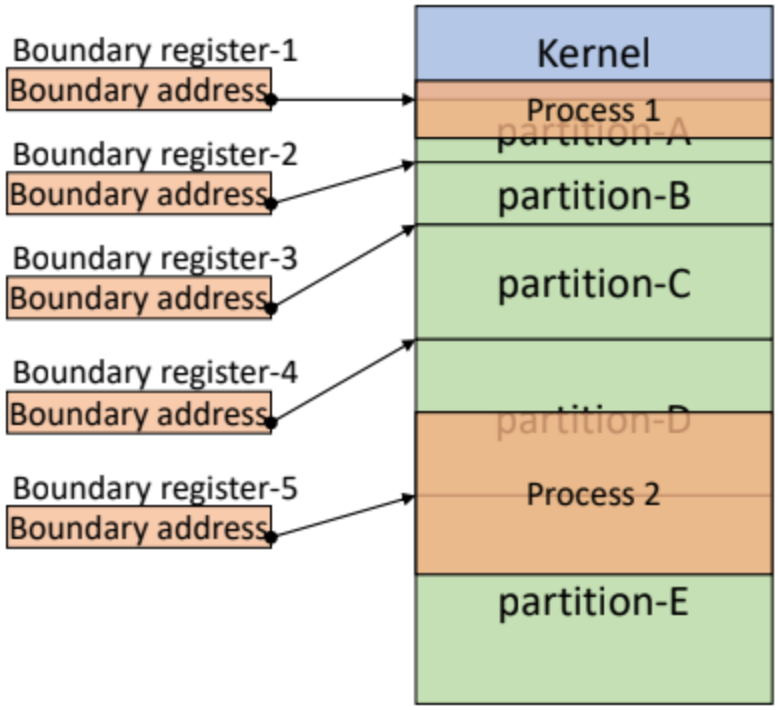


- 각 프로세스는 하나의 partition(분할)에 적재
  - Process : Partition = 1 : 1
- Partition의 수 = K
  - Multiprogramming degree = K
- 자료구조 예시

partition	start address	size	current process ID	other fields
A	a1	10 MB	-	···
B	a2	10 MB	-	···
C	a3	20 MB	-	···
D	a4	30 MB	-	···
E	a5	50 MB	-	···

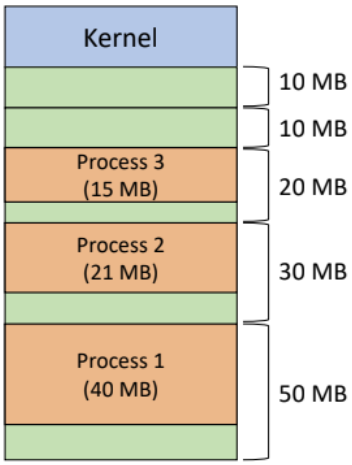
<Partition table or State table>

- 커널 및 사용자 영역 보호

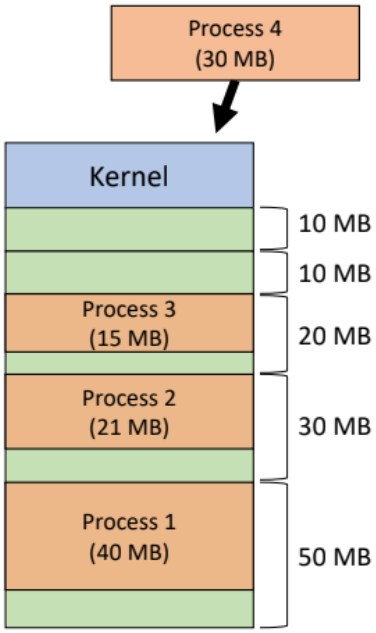


Fragmentation (단편화)

- Internal fragmentation
  - 내부 단편화
  - (Partition 크기 > Process 크기) ⇒ 메모리가 낭비 됨



- External fragmentation



- 외부 단편화
- (남은 메모리 크기 > Process 크기)지만, 연속된 공간이 아님 ⇒ 메모리가 낭비 됨

FPM 요약

- 고정된 크기로 메모리 미리 분할
- 장점
  - 메모리 관리가 간편함 ⇒ Low overhead
- 단점

- 시스템 자원이 낭비 될 수 있음
- Internal/external fragmentation

Variable Partition Multiprogramming

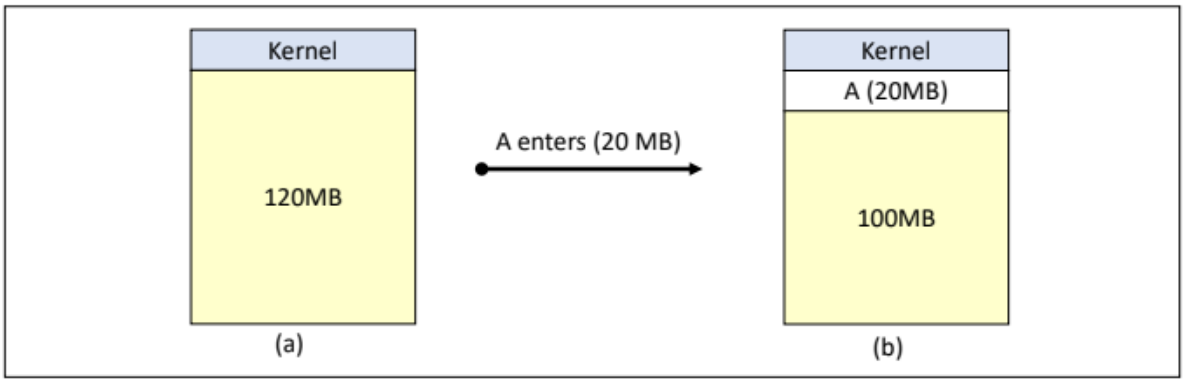
- 초기에는 전체가 하나의 영역
- 프로세스를 처리하는 과정에서 메모리 공간이 동적으로 분할
- No internal fragmentation

VPM 예제

시나리오

Memory space: 120 MB

1. 초기 상태
2. 프로세스 A(20MB) 가 적재 된 후



<State table>

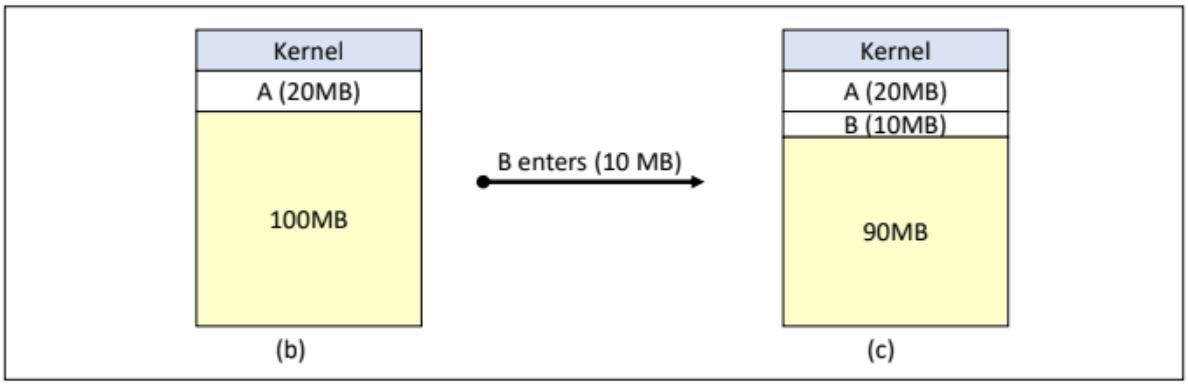
partition	start address	size	current process ID	other field
1	u	120	none	· · ·

(a)

partition	start address	size	current process ID	other field
1	u	20	A	· · ·
2	u+20	100	none	· · ·

(b)

3. 프로세스 B(10MB) 가 적재 된 후



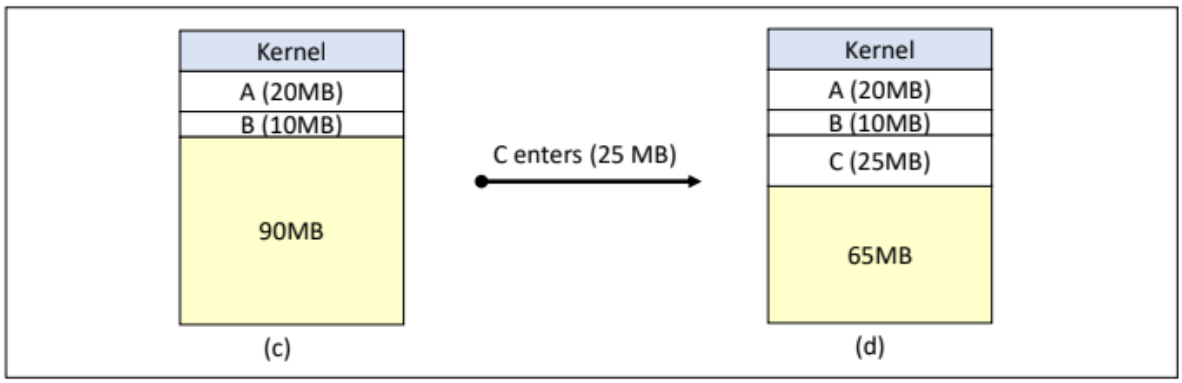
partition	start address	size	current process ID	other field
1	u	20	A	···
2	u+20	100	none	···

(b)

partition	start address	size	current process ID	other field
1	u	20	A	···
2	u+20	10	B	···
3	u+30	90	none	···

(c)

4. 프로세스 C(25MB) 가 적재 된 후



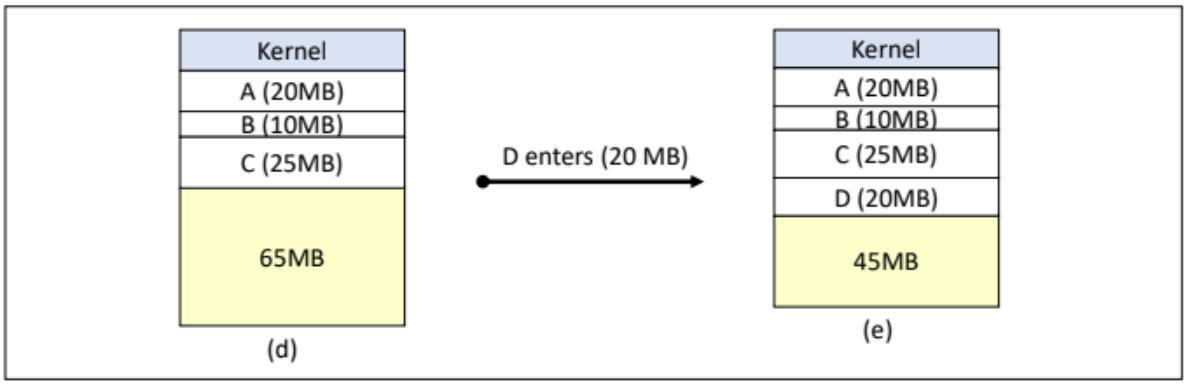
partition	start address	size	current process ID	other field
1	u	20	A	···
2	u+20	10	B	···
3	u+30	90	none	···

(c)

partition	start address	size	current process ID	other field
1	u	20	A	···
2	u+20	10	B	···
3	u+30	25	C	···
4	u+55	65	none	···

(d)

5. 프로세스 D(20MB) 가 적재 된 후



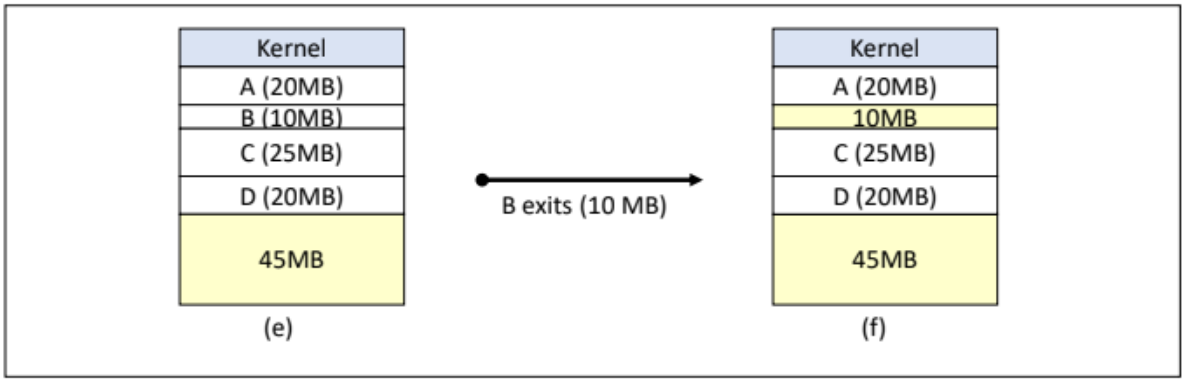
partition	start address	size	current process ID	other field
1	u	20	A	···
2	u+20	10	B	···
3	u+30	25	C	···
4	u+55	65	none	···

(d)

partition	start address	size	current process ID	other field
1	u	20	A	···
2	u+20	10	B	···
3	u+30	25	C	···
4	u+55	20	D	···
5	u+75	45	none	···

(e)

6. 프로세스 B가 주기억장치를 반납한 후



partition	start address	size	current process ID	other field
1	u	20	A	···
2	u+20	10	B	···
3	u+30	25	C	···
4	u+55	20	D	···
5	u+75	45	none	···

(e)

partition	start address	size	current process ID	other field
1	u	20	A	···
2	u+20	10	none	···
3	u+30	25	C	···
4	u+55	20	D	···
5	u+75	45	none	···

(f)

7. 프로세스 E(15MB) 가 적재 된 후

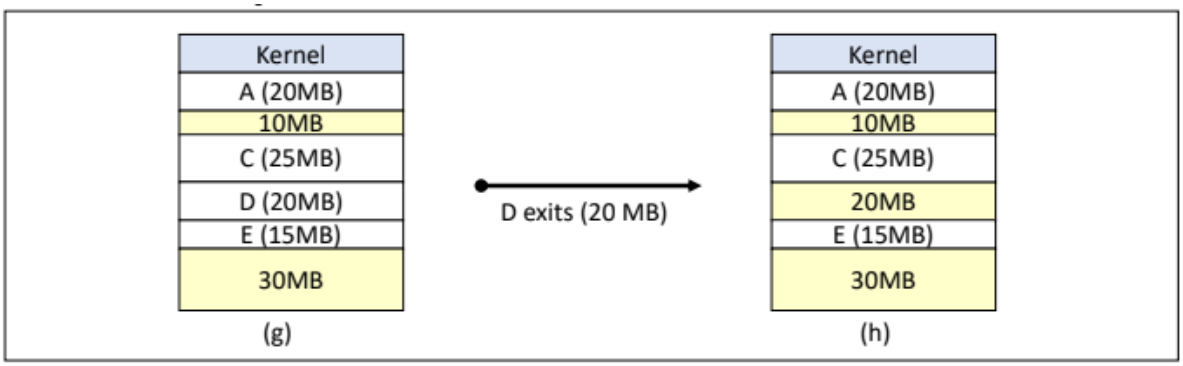
partition	start address	size	current process ID	other field
1	u	20	A	···
2	u+20	10	none	···
3	u+30	25	C	···
4	u+55	20	D	···
5	u+75	45	none	···

(f)

partition	start address	size	current process ID	other field
1	u	20	A	···
2	u+20	10	none	···
3	u+30	25	C	···
4	u+55	20	D	···
5	u+75	15	E	···
6	u+90	30	none	···

(g)

8. 프로세스 D가 주기억장치를 반납한 후



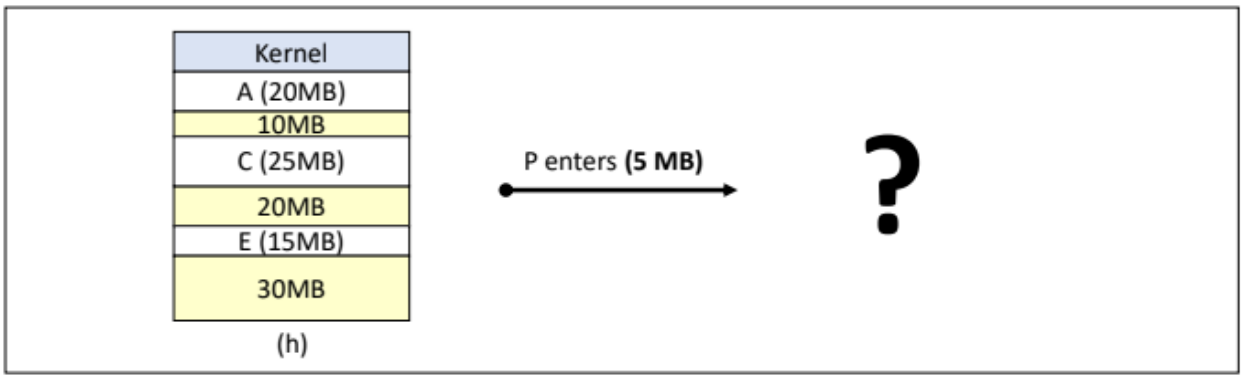
partition	start address	size	current process ID	other field
1	u	20	A	···
2	u+20	10	none	···
3	u+30	25	C	···
4	u+55	20	D	···
5	u+75	15	E	···
6	u+90	30	none	···

(g)

partition	start address	size	current process ID	other field
1	u	20	A	···
2	u+20	10	none	···
3	u+30	25	C	···
4	u+55	20	none	···
5	u+75	15	E	···
6	u+90	30	none	···

(h)

어디에 배치할 것인가?



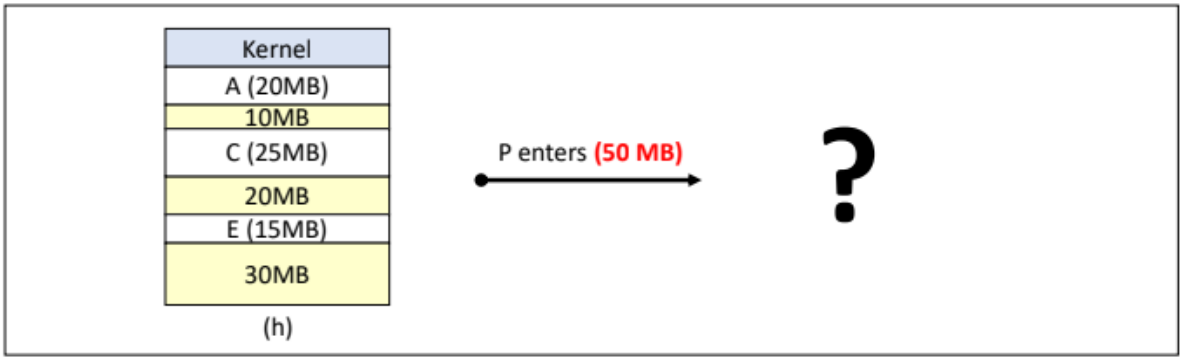
배치 전략 (Placement strategies)

- **First-fit (최초 적합)**
  - 충분한 크기를 가진 첫 번째 partition을 선택
  - Simple and low overhead
  - 공간 활용률이 떨어질 수 있음 → 15MB 공간에 14MB가 들어가면 남는 1MB의 공간은 활용률이 떨어짐
- **Best-fit (최적 적합)**
  - Process가 들어갈 수 있는 partition 중 가장 작은 곳 선택
  - 탐색시간이 오래 걸림 (overhead가 크다) ← 모든 partition을 살펴봐야 함
  - 크기가 큰 partition을 유지 할 수 있음
  - 작은 크기의 partition이 많이 발생
    - 활용하기 너무 작은 partition 발생 ⇒ 공간 활용률 떨어짐
- **Worst-fit (최악 적합)**
  - Process가 들어갈 수 있는 partition 중 가장 큰 곳 선택
  - 탐색시간이 오래 걸림 (overhead가 크다) ← 모든 partition을 살펴봐야 함
  - 작은 크기의 partition 발생을 줄일 수 있음
  - 큰 크기의 partition 확보가 어려움
    - 큰 프로세스가 들어왔을 때 들어갈 공간이 없을 수 있음
- **Next-fit (순차 최초 적합)**

- 최초 적합 전략과 유사
- State table에서 마지막으로 탐색한 위치부터 탐색
- **메모리 영역의 사용 빈도 균등화**
- Low overhead

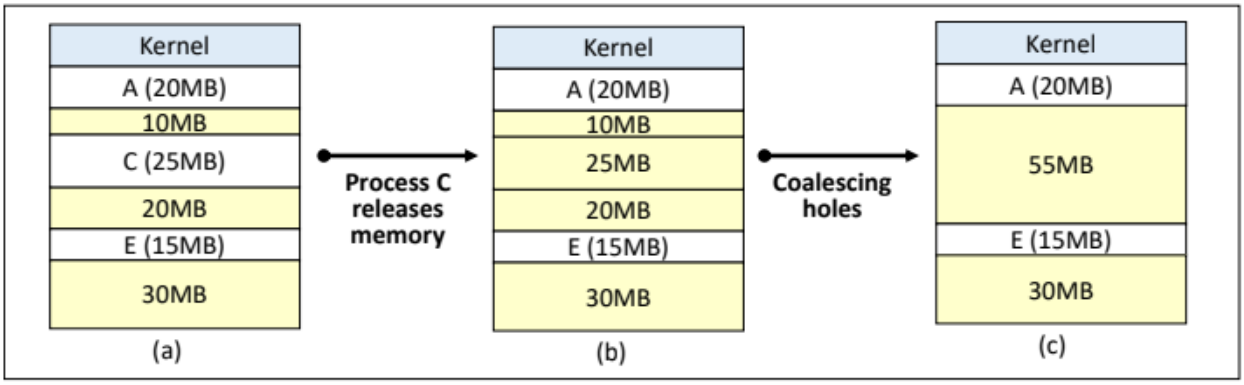
어디에 배치할 것인가?

- External fragmentation issue



Coalescing holes (공간 통합)

- 인접한 빈 영역을 하나의 partition으로 통합
  - Process가 memory를 release하고 나가면 수행
  - Low overhead



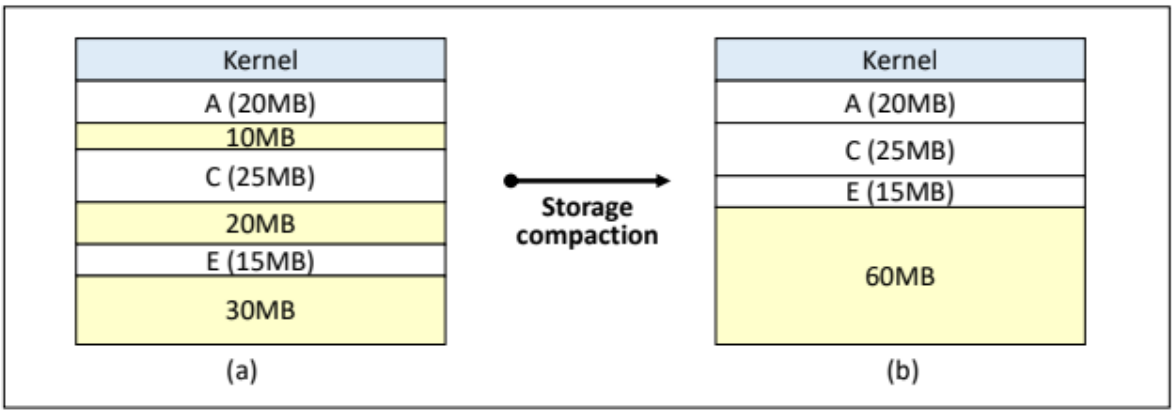
partition	start address	size	current process ID
1	u	20	A
2	u+20	10	none
3	u+30	25	C
4	u+55	20	none
5	u+75	15	E
6	u+90	30	none

partition	start address	size	current process ID
1	u	20	A
2	u+20	10	none
3	u+30	25	none
4	u+55	20	none
5	u+75	15	E
6	u+90	30	none

partition	start address	size	current process ID
1	u	20	A
2	u+20	55	none
3	u+75	15	E
4	u+90	30	none

Storage Compaction (메모리 압축)

- 모든 빈 공간을 하나로 통합
- 프로세스 처리에 필요한 적재 공간 확보가 필요할 때 수행



partition	start address	size	current process ID
1	u	20	A
2	u+20	10	none
3	u+30	25	C
4	u+55	20	none
5	u+75	15	E
6	u+90	30	none

partition	start address	size	current process ID
1	u	20	A
2	u+20	25	C
3	u+45	15	E
4	u+60	60	none

Base images from Prof.

- High overhead
  - 모든 Process 재배치 (Process 중지)
  - 많은 시스템 자원을 소비
  - 자주 해주면 안되고 일정 기간 혹은 요청이 있을 때만 실행

Continuous Memory Allocation 요약

- Uni-programming
  - Simple
  - Fragmentation problem
- Fixed partition multi-programming (FPM)
- Variable partition multi-programming (VPM)
  - Placement strategies
    - First-fit, Best-fit, Worst-fit, Next-fit
  - External fragmentation issue

- Coalescing holes
- Storage compaction