메모리 관리

IT 융합공학부 김진웅

유튜브 주소

메모리 계층 구조와 메모리 관리 핵심

메모리 주소

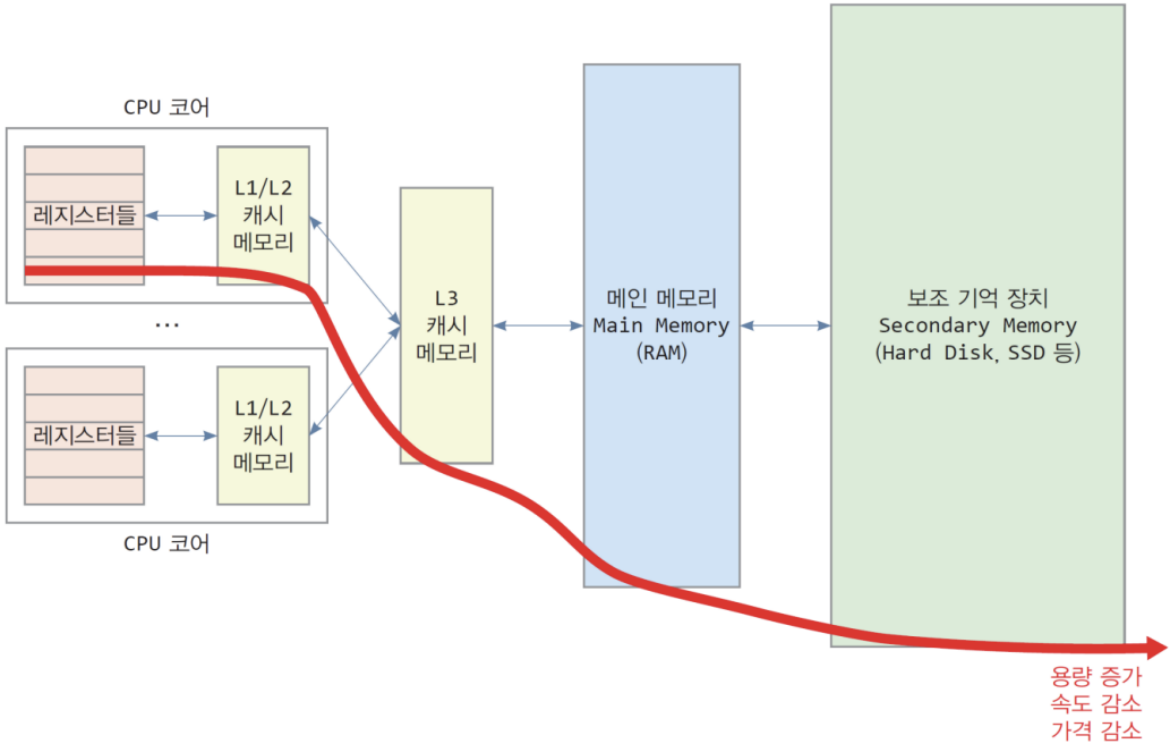
물리 메모리 관리

메모리 할당 방식

메모리 계층 구조

- 메모리는 컴퓨터 시스템 여러 곳에 계층적으로 존재
  - CPU 레지스터 – CPU 캐시 – 메인 메모리 – 보조기억장치
  - CPU 레지스터에서 보조기억장치로 갈수록

    -> 용량 증가, 가격 저렴, 속도 저하



메모리 계층 구조의 특성

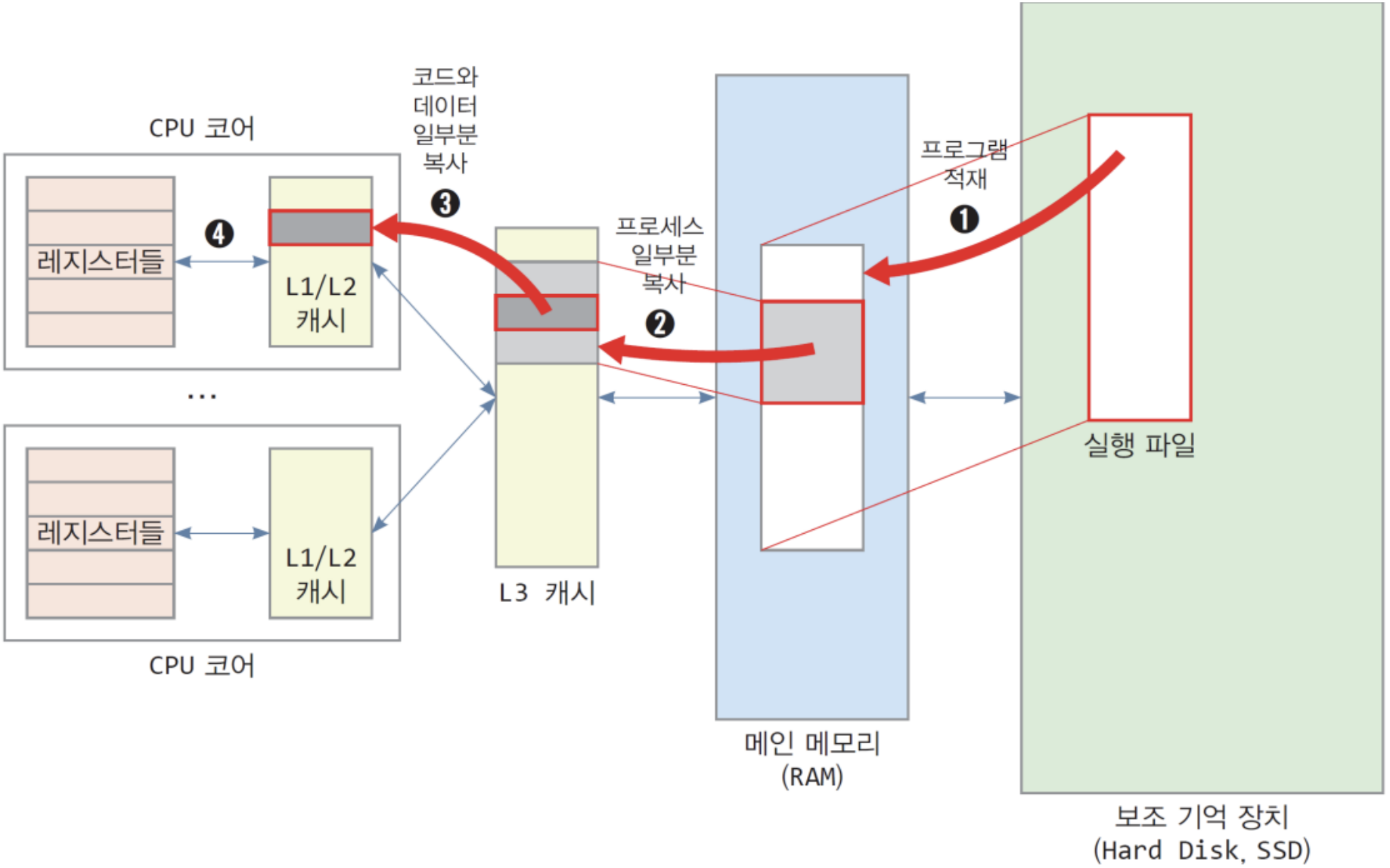
	CPU 레지스터	L1/L2 캐시	L3 캐시	메인 메모리	보조 기억 장치
용도	몇 개의 명령과 데이터 저장	한 코어에서 실행되는 명령과 데이터 저장	멀티 코어들에 의해 공유. 명령과 데이터 저장	실행 중인 전체 프로세스들의 코드와 데이터, 입출력 중인 파일 블록들 저장	파일이나 데이터베이스, 그리고 메모리에 적재된 프로세스의 코드와 데이터의 일시 저장
용량	바이트 단위. 8~30개 정도. 1KB 미만	KB 단위 (Core i7의 경우 32KB/256KB)	MB 단위 (Core i7의 경우 8MB)	GB 단위 (최근 PC의 경우 최소 8GB 이상)	TB 단위
타입		SRAM (Static RAM)	SRAM (Static RAM)	DRAM (Dynamic RAM)	마그네틱 필드나 플래시 메모리
속도	〈1ns	〈5ns	〈5ns	〈50ns	〈20ms
가격		고가	고가	보통	저가
휘발성	휘발성	휘발성	휘발성	휘발성	비휘발성

메모리 계층화의 목적

- 계층화의 역사적 과정
  - CPU 성능 향상 -> 더 빠른 메모리 요구 -> 작지만 빠른 off-chip 캐시 등장 -> 더 빠른 액세스를 위해 on-chip 캐시 -> 멀티 코어의 성능에 적합한 L1, L2, L3 캐시

- 메모리 계층화의 목적
  - CPU의 메모리 액세스 시간을 줄이기 위함

메모리 계층에서 코드와 데이터 이동



메모리 계층화 성공 이유

- 작은 캐시에 당장 실행할 프로그램 코드와 데이터를 일부분만 두는데도 효과적인 이유

  -> 참조의 지역성 때문

  - 코드나 데이터, 자원 등이 아주 짧은 시간 내에 다시 사용되는 특성

    -> CPU는 작은 캐시 메모리에 적재된 코드와 데이터로 한동안 실행

  - 캐시를 채우는 시간의 손해보다 빠른 캐시를 이용하는 이득이 큼

메모리 관리

- **메모리의 역할**
  - 메모리는 실행하고자 하는 프로그램 코드와 데이터 적재
  - CPU는 메모리에 적재된 코드와 데이터만 처리

- **운영체제에 의해 메모리 관리가 필요한 이유**
  - 메모리는 공유 자원이기 때문
  - 메모리는 보호되어야 하기 때문
  - 메모리 용량 한계 극복할 필요
  - 메모리 효율성 증대를 위해

물리 주소와 논리 주소 (1)

- 메모리는 오직 주소로만 접근

- 물리 주소(physical address)
  - 물리 메모리(RAM)에 매겨진 주소, 하드웨어에 의해 고정된 메모리 주소
  - 0에서 시작하여 연속되는 주소 체계
  - 메모리는 시스템 주소 버스를 통해 물리 주소의 신호를 받음

물리 주소와 논리 주소 (2)

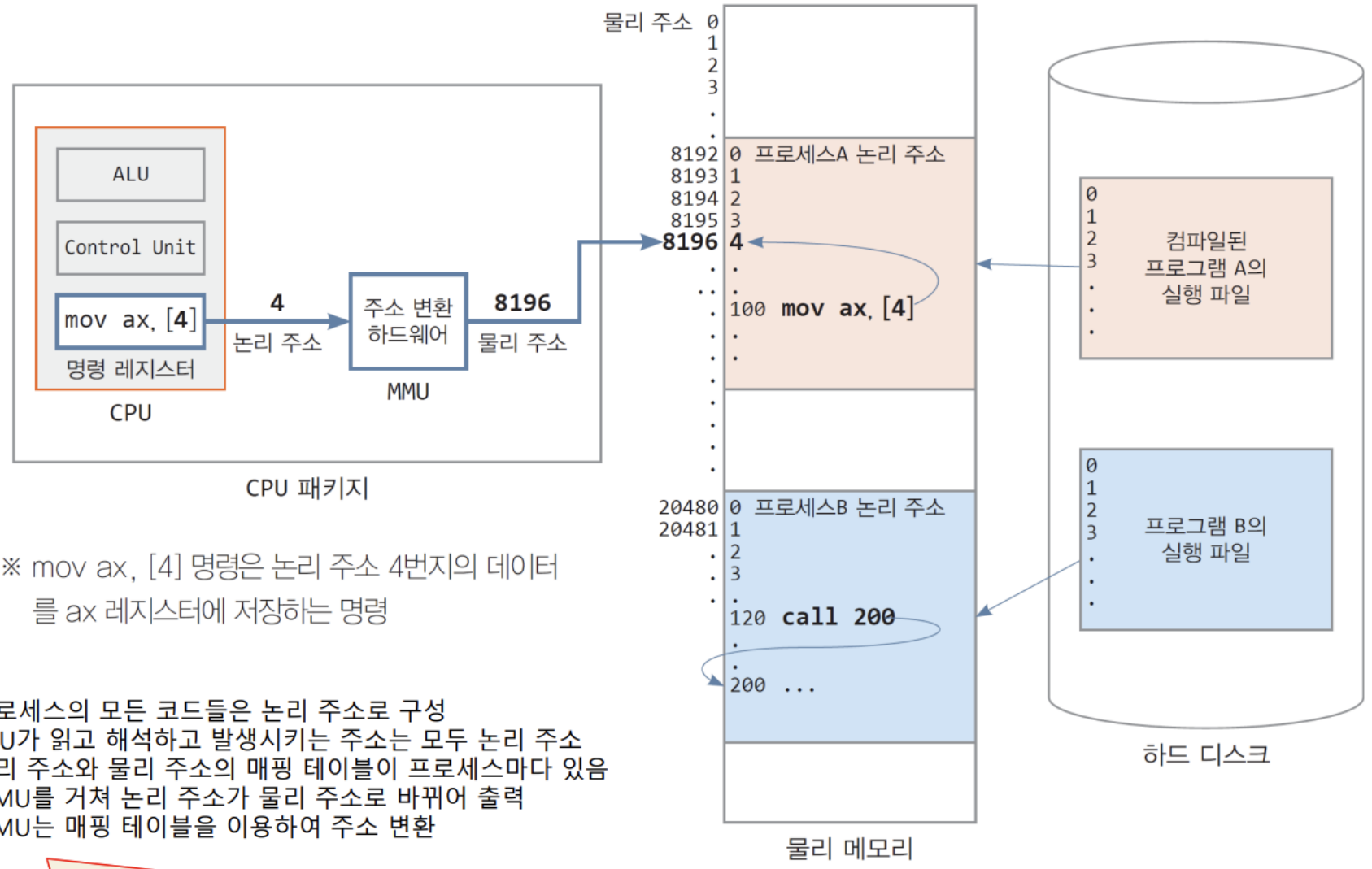
- 논리/가상 주소(logical address/virtual address)
  - 개발자나 프로세스가, 프로세스 내에서 사용하는 주소, 코드나 변수 등에 대한 주소
  - 0에서 시작하여 연속되는 주소 체계, 프로세스 내에서 매겨진 상대 주소
  - 컴파일러와 링커에 의해 매겨진 주소
  - CPU 내에서 프로세스를 실행하는 동안 다루는 모든 주소는 논리 주소
  - 사용자나 프로세스는 결코 물리 주소를 알 수 없음

MMU(Memory Management Unit)

- 논리 주소를 물리 주소로 바꾸는 하드웨어 장치
  - CPU가 발생시킨 논리 주소는 MMU에 의해 물리 주소로 바뀌어 물리 물리 메모리에 도달

- 오늘날 MMU는 CPU 패키지에 내장
  - 인텔이나 AMD의 x86 CPU는 80286부터 MMU를 내장
  - MMU 덕분으로 여러 프로세스가 하나의 물리 메모리에서 실행됨

논리 주소와 물리 주소, MMU에 의한 주소 변환



※ mov ax, [4] 명령은 논리 주소 4번지의 데이터를 ax 레지스터에 저장하는 명령

- 프로세스의 모든 코드들은 논리 주소로 구성
- CPU가 읽고 해석하고 발생시키는 주소는 모두 논리 주소
- 논리 주소와 물리 주소의 매핑 테이블이 프로세스마다 있음
- MMU를 거쳐 논리 주소가 물리 주소로 바뀌어 출력
- MMU는 매핑 테이블을 이용하여 주소 변환

프로세스 A의 코드 실행 중 CPU 안의 mov ax, [4] 명령에 담긴 4 번지는 논리주소이고, 논리 주소 4번지의 물리 주소는 8196임

메모리 할당(memory allocation)

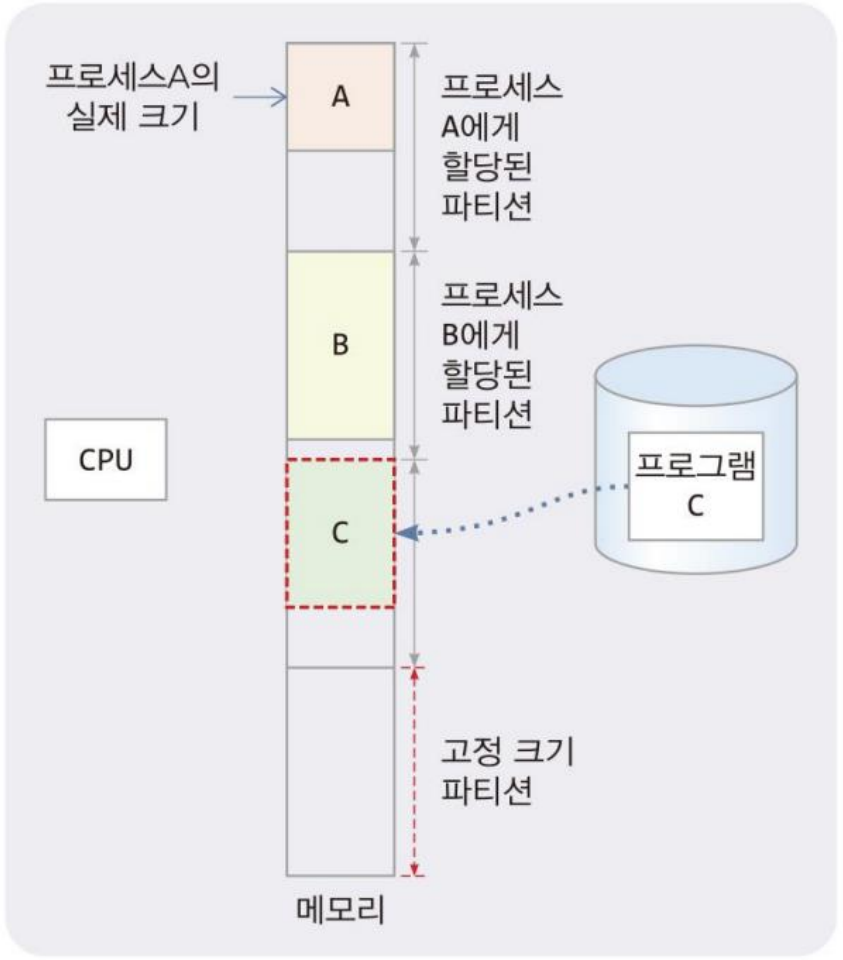
- 메모리 할당
  - 운영체제가 새 프로세스를 실행시키거나 실행 중인 프로세스가 메모리를 필요로 할 때, 물리 메모리 할당
  - 프로세스의 실행은 할당된 물리 메모리에서 이루어짐
    -> 프로세스의 코드(함수), 변수, 스택, 동적 할당 공간 액세스 등

메모리 할당 기법 – 연속 메모리 할당

- 프로세스별로 연속된 한 덩어리의 메모리 할당
- 고정 크기 할당
  - 메모리를 고정 크기의 파티션으로 나누고 프로세스당 하나의 파티션 할당
  - 파티션의 크기는 모두 같거나 다를 수 있음

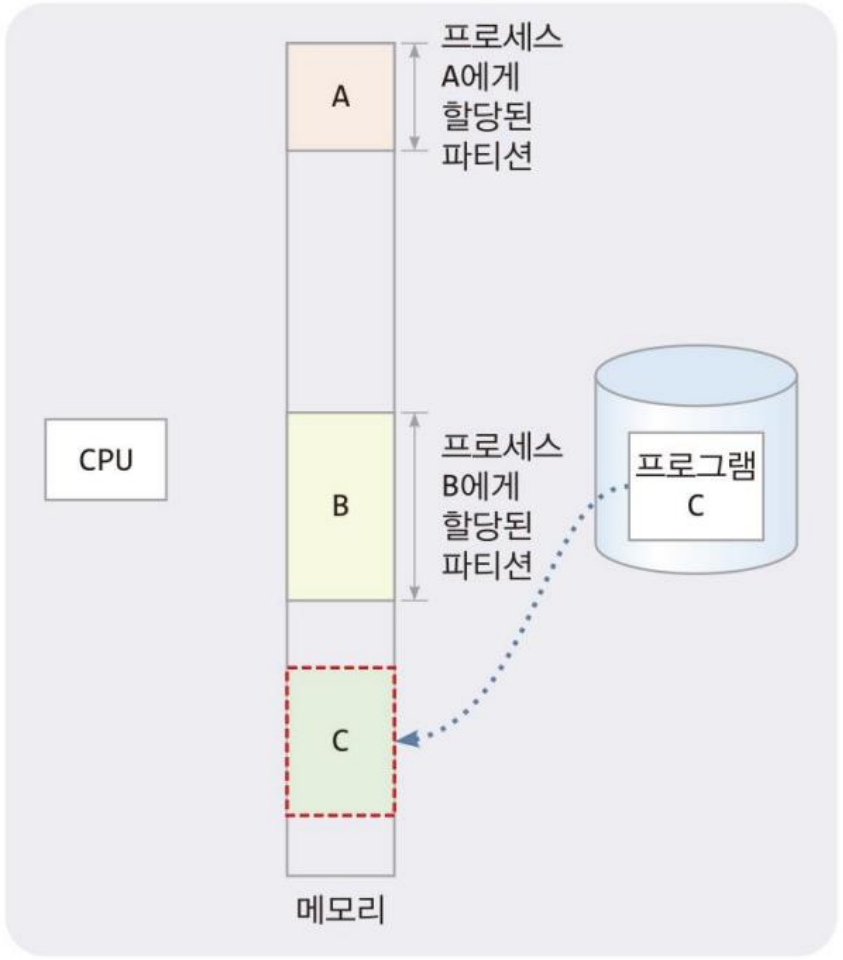
- 가변 크기 할당
  - 메모리를 가변 크기의 파티션으로 나누고 프로세스당 하나의 파티션 할당

메모리 할당 기법 – 연속 메모리 할당



메모리를 고정 크기의 파티션으로 나누고
각 프로세스를 하나의 파티션에 배치

(a) 연속 메모리 할당 – 고정 크기



각 프로세스에게 자신의 크기만한
파티션 동적 할당

(b) 연속 메모리 할당 – 가변 크기

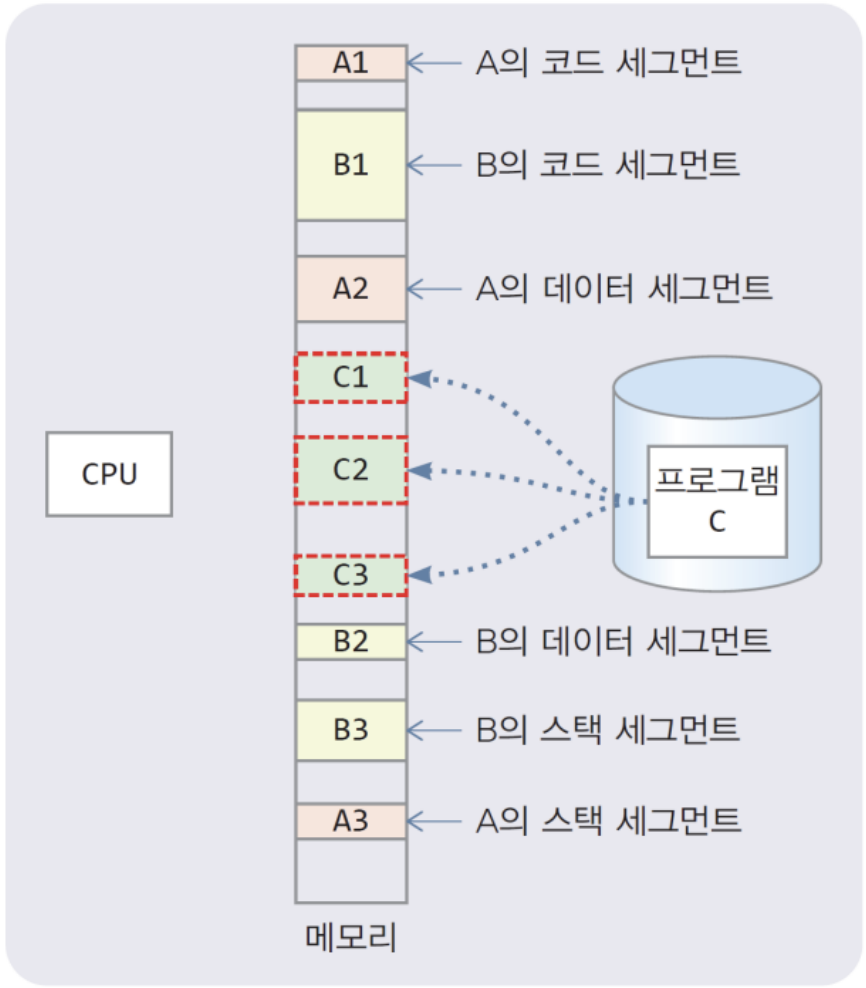
메모리 할당 기법 – 분할 메모리 할당

- 프로세스에게 여러 덩어리의 메모리 할당

- 고정 크기 할당
  - 고정 크기의 덩어리 메모리를 여러 개 분산 할당
  - ex. 페이징(paging) 기법

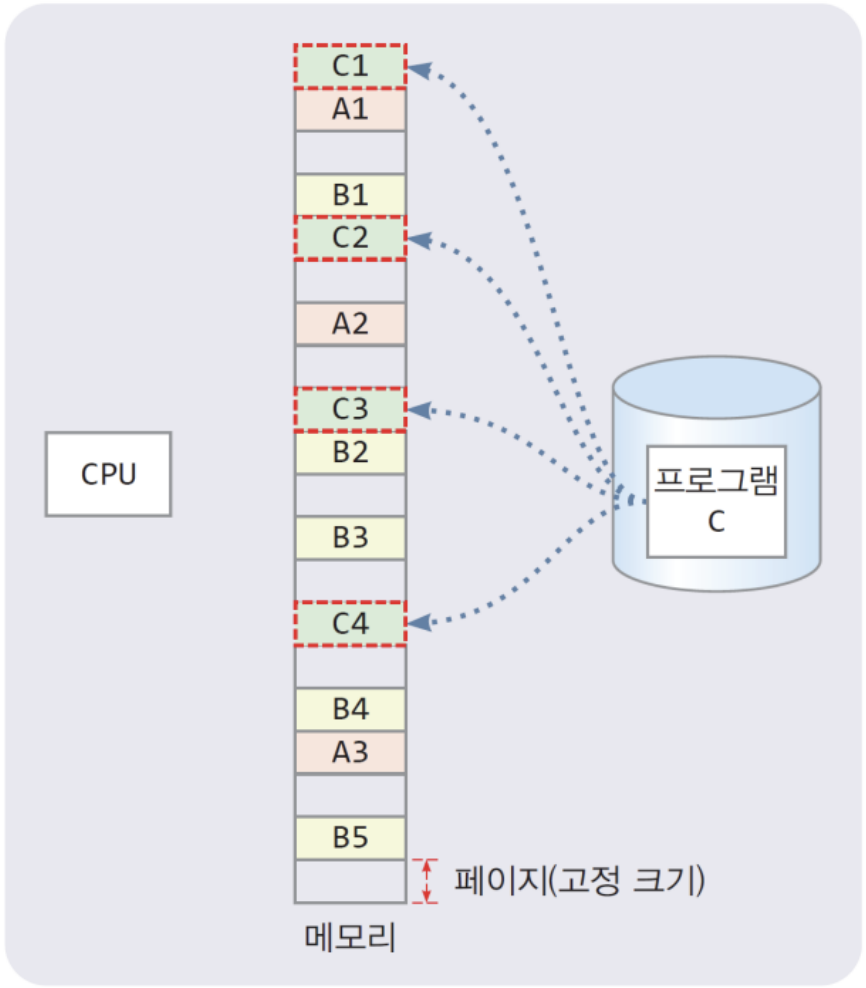
- 가변 크기 할당
  - 가변 크기의 덩어리 메모리를 여러 개 분산 할당
  - ex. 세그먼테이션(segmentation) 기법

메모리 할당 기법 – 분할 메모리 할당



프로세스를 가변 크기의 세그먼트들로 분할 할당

(c) 분할 할당 – 세그먼테이션(segmentation)



프로세스를 고정 크기의 페이지들로 분할 할당

(d) 분할 할당 – 페이징(paging)

Q & A