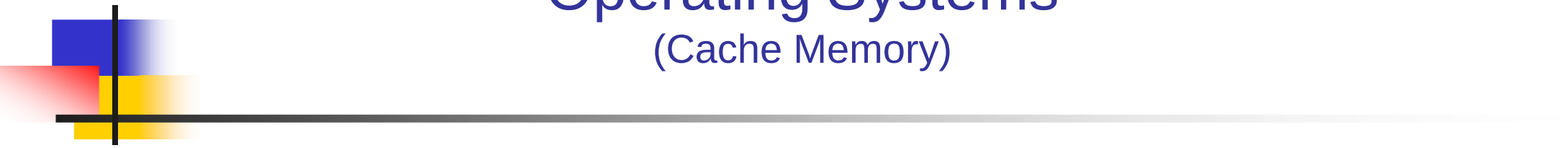
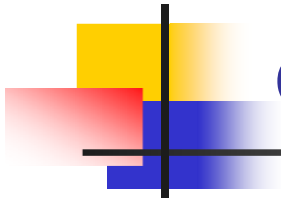


Operating Systems
(Cache Memory)

Chapter 12

These lecture materials are modified from the lecture notes written by A. Silberschatz, P. Galvin and G. Gagne.

August, 2022



Outline

- Cache Memory Concept
- Cache Placement Policy
- Cache Writing Policy

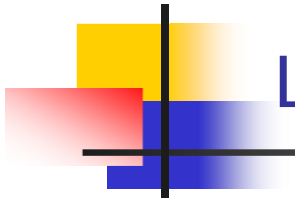
Cache Memory Concept

Cache Memory Concept

- 캐시 메모리(Cache Memory)는 속도가 빠른 장치와 느린 장치 간의 속도차에 따른 병목 현상을 줄이기 위한 범용 메모리
- 메인 메모리와 CPU 사이에 위치

CPU -> 캐시-> 메인 메모리

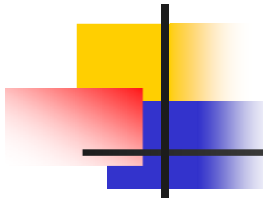
- CPU의 속도에 버금갈 만큼 메모리 계층에서 가장 속도가 빠르지만, 용량이 적고 비쌈
  - 보통 캐시는 SRAM, 메인 메모리는 DRAM



Locality

- 캐시 메모리는 메인 메모리에서 자주 사용하는 프로그램과 데이터를 저장해두어 속도를 빠르게 함
- CPU가 어떤 데이터를 원하는지 어느 정도 예측할 수 있어야 함

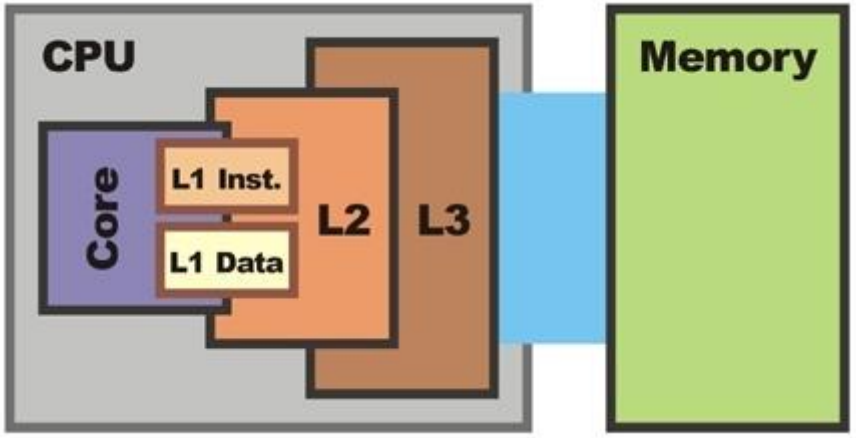
- 캐시의 지역성(Locality)을 이용



Locality

- **공간 지역성 (Spatial Locality) :**
- 최근에 사용했던 데이터와 인접한 데이터가 참조될 가능성이 높다는 특성
  - 예) 배열 A[0], A[1]과 같은 연속 접근의 경우 그다음 원소들에 접근할 가능성이 높다.
- **시간 지역성 (Temporal Locality) :**
- 최근에 사용했던 데이터가 재참조될 가능성이 높은 특성
  - 예) for, while 같은 반복문은 특정 부분을 반복해서 접근하기 때문에 다시 참조할 확률이 높음

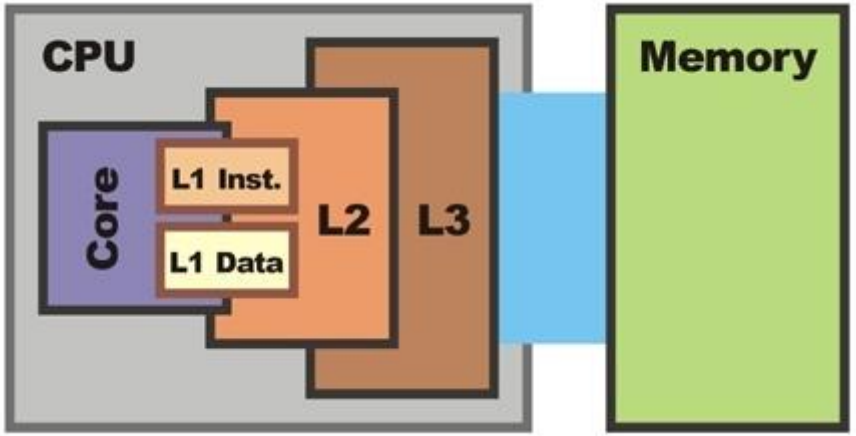
Level



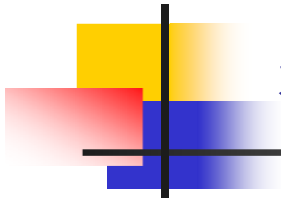
- 시스템에 장착된 캐시의 용량과 성능이 점점 증가하면서 캐시의 캐시로 사용되는 메모리가 추가됨
- 이것을 적용된 순서대로 L(Level) 1, L2, L3 ... 라고 호칭.

- L1에 가장 고성능이자 고가인 작은 용량의 집적회로가 사용
  - L1캐시에서 데이터를 퍼가기 위한 캐시로 사용하기 위해 그보다는 용량이 크지만 그 대신 약간 더 느린 저장공간 L2가 추가됨

Level

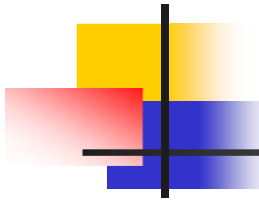


- L1 캐시 메모리에는 명령어 캐시와 데이터 캐시가 따로 존재
- L1 캐시 메모리는 CPU에 직접 데이터를 공급해 주기 때문에 빠른 접근 지연 시간(Access latency)이 매우 중요
- 명령어는 공간 지역성이, 데이터는 시간 지역성이 높음
- 명령어와 데이터를 동시에 읽어올 수 있게 함으로써 CPU의 성능을 향상



캐시 메모리의 동작

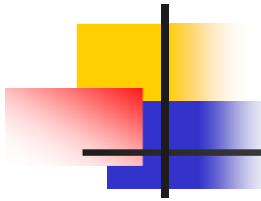
- CPU가 데이터가 필요할 때, 제일 먼저 캐시 메모리를 검색함.
  - cache hit라면 CPU는 캐시에서 데이터가 있는 블록을 가져옴.
  - cache miss라면 캐시가 메인 메모리에서 필요한 블록을 가져옴. 다음, CPU는 캐시에서 블록을 가져옴.
- 1 블록 = 1 캐시 라인



Caching Line

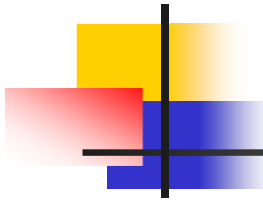
- 캐시 메모리는 메인 메모리에 비해 크기가 매우 작기 때문에 메인 메모리와 1:1 매칭이 불가능
- 캐시가 아무리 CPU에 가깝게 위치하더라도, 데이터가 캐시 내의 어느 곳에 저장되어 있는지 찾기가 어려워 모든 데이터를 순회해야 한다면 캐시의 장점을 잃기 때문에 쉽게 찾을 수 있는 구조가 필요함

- Caching Line: 빈번하게 사용되는 데이터의 주소들이 흩어져 있기 때문에 캐시에 저장하는 데이터에는 데이터의 주소 등을 기록해둔 태그를 달아둘 필요가 있음
- 이러한 태그들의 묶음을 의미



Caching Line

- **Caching Line:**
- 빈번하게 사용되는 데이터의 주소들이 흩어져 있기 때문에 캐시에 저장하는 데이터에는 데이터의 주소 등을 기록해둔 태그를 달아둘 필요가 있음
- 이러한 태그들의 묶음을 의미



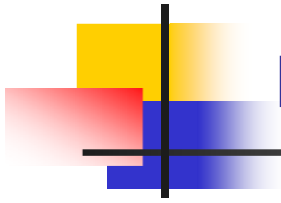
Performance

- Cache Hit Ratio
  - (Cache Hit 수 / (Cache Hit 수 + Cache Miss 수))× 100
- Cache Miss Ratio
  - 1 - (Cahe Hit Ratio)
- Hit Time
  - 캐시 라인의 데이터를 CPU로 전달하는 데까지 걸리는 시간
- Miss Penalty
  - 캐시 미스로 메인메모리에서 블록을 캐시라인에 가져오는 시간.
  - 캐시 미스시 Hit Time + Miss Penalty 가 캐시 레이턴시

캐시 미스의 종류

- 1) Compulsory Miss
  - 특정 데이터에 처음 접근할 때 발생하는 cache miss
- 2) Capacity Miss
  - 캐시 메모리의 공간이 부족해서 발생하는 cache miss
- 3) Conflict Miss
  - 캐시 메모리에 A와 B 데이터를 저장해야 하는데, A와 B가 같은 캐시 메모리 주소에 할당되어 있어서 발생하는 cache miss
  - 캐시 공간이 작아서 벌어지는 일이므로 캐시 크기를 키우면 해결되지만 비용 문제가 발생

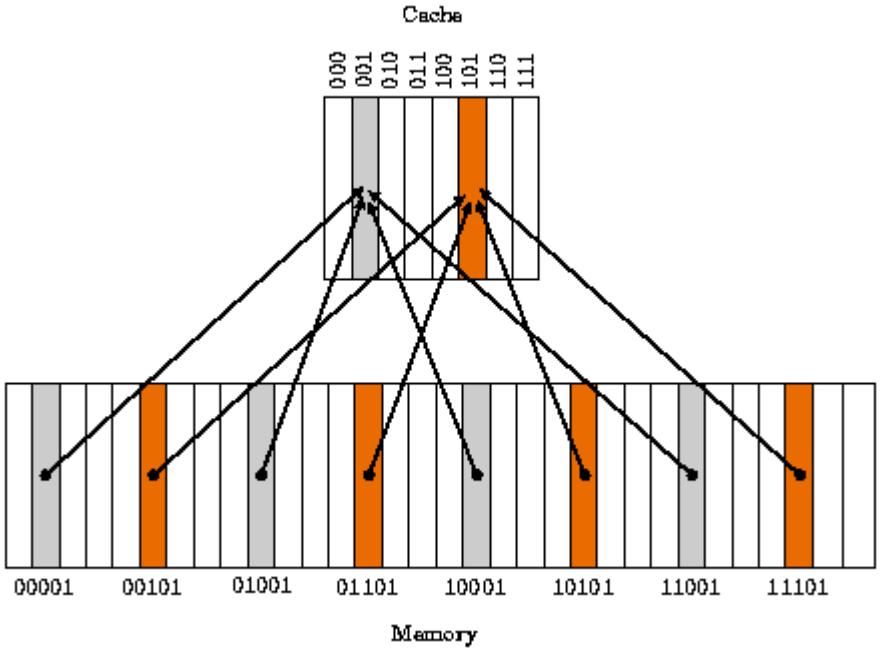
Cache Placement Policy

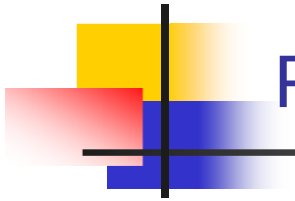


Direct Mapping

- 메인 메모리를 일정한 크기의 블록으로 나누어 각각의 블록을 캐시의 정해진 위치에 매핑하는 방식.
  - 예) 메인 메모리의 2번째 블록 (00001) 은 캐시의 2번째 집합(001) 에만 들어갈 수 있음
- 가장 간단하고 구현도 쉬움
- 하지만 적중률(Hit rate)이 낮아질 수 있음.
- 동일한 캐시 메모리에 할당된 여러 데이터를 사용할 때 충돌이 발생하게 되는 단점이 있음

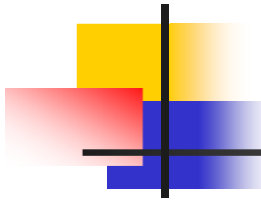
Direct Mapping





Full Associative Mapping

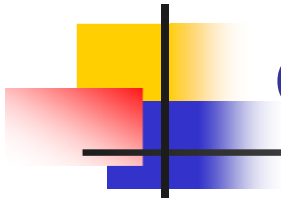
- 캐시 메모리의 빈 공간에 마음대로 주소를 저장하는 방식
- 저장이 매우 간단함
- 원하는 데이터가 있는지 찾기 위해서는 모든 태그를 병렬적으로 검사해야 하기 때문에 복잡하고 비용이 높음



Set Associative Mapping

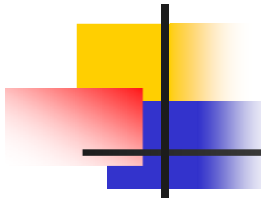
- Direct Mapping과 Full Associative Mapping의 장점을 결합한 방식
- 빈 공간에 마음대로 주소를 저장하되, 미리 정해둔 특정 행에만 저장하는 방식.
- Direct에 비해 검색 속도는 느리지만 저장이 빠르고 Full에 비해 저장이 느리지만 검색은 빠름.
- 주로 사용되는 방식

Cache Writing Policy



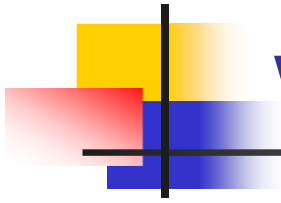
Cache Writing Policy

- CPU에 의해 캐시의 데이터가 변경 되었을 때, 그 근원인 메인 메모리의 값도 변경되어야함.
- 언제 메인 메모리의 값을 변경 할 것인가?



Cache Coherency(캐시 일관성)

- 멀티코어 시스템에서는 다른 코어에 의해 데이터가 변경될 수 있음.
- 이 때 이 사실을 다른 코어들에게 전달하고 각 코어의 데이터를 업데이트 하여 동일하게 하는 것

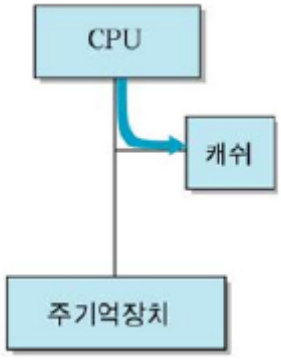


Write-through 정책

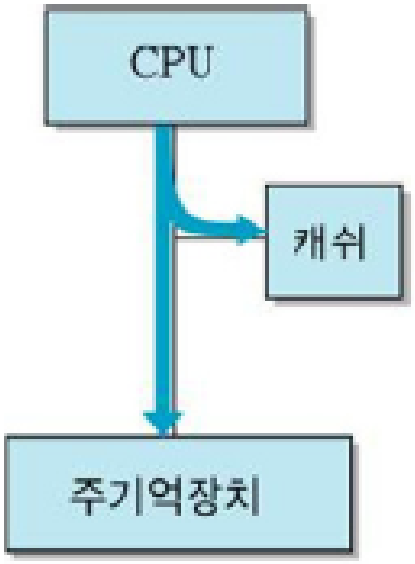
- 캐시 Hit 시 쓰기 정책
- 캐시의 내용도 수정하고 곧바로 메인 메모리의 내용도 수정
- Cache Coherency이 보장되지만 버스 병목현상이 존재
  - 즉 느리다는 문제가 있음!

Write-back 정책

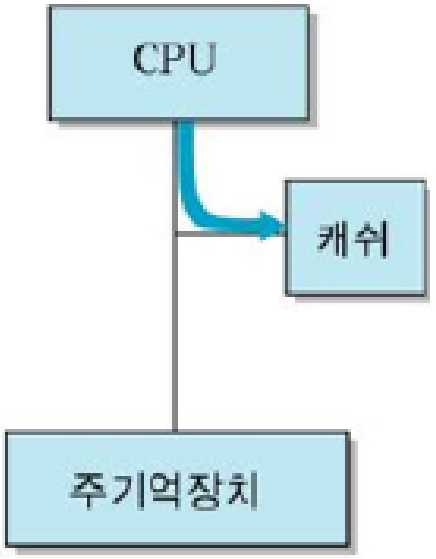
- 캐시 Hit 시 쓰기 정책
- 캐시의 내용만 수정하고, 해당 캐시라인이 Replacement 될 때 메인 메모리의 내용도 수정.
- 메인 메모리에 쓰기 동작이 최소화 되기 때문에 속도 측면에서 이득
- 캐시라인에서 내려올 때 메인 메모리의 수정이 필요하다는 것을 알려주기 위해 추가적인 비트 필요
- Cache Coherency을 어기는 문제점이 있음



캐시 Hit 시 쓰기 정책



Write-through



Write-back

캐시 Miss 시 쓰기 정책

- **Write allocate:**
- 메인 메모리의 블록을 수정해준 다음 캐시로 Fetch.
- 해당 데이터가 자주 수정된다면 이 정책을 사용하여 캐시에 올려두는 것이 성능에서 이득

- **No-write allocate:**
- 메인 메모리의 블록만 수정만 하는 정책.
- 캐시에 따로 올리지 않음

Chapter 12
Finish