power 측면.
성능이 어떻게 되는가
SRAM!! 아다" 하고 끝날 것이 아닌.

Introduction to Cache Memory – Part#1

지금까지 배워온 컴퓨터 속의
cache memory에 대해 알아볼것임.

Yoonjin Kim

Full Professor

Dept. of Computer Science
Sookmyung Women's University

Outline

- **Before beginning**
- **What is cache memory?** (⇒ cache memory 개념)
- **Why do CPUs need cache memory?** (⇒ 왜 CPU는 캐시메모리를 필요로 하는가)
- ☆☆☆ 이론의 시작이될.
  What data is stored into cache memory? ↑ 이번주 분량
  (⇒ 캐시메모리에 어떤 데이터가 저장되는가)
- **How does cache memory work?**
  - Basic concepts
  - Cache read-operation
  - Cache write-operation
  - Cache memory space allocation
    ↳ 메모리 공간 할당

Before Beginning

(구글링해서 찾은 결과)

- Let's search for commercial CPU specification.

(기본 상용 CPU의 Spec)

Cache가 어떤지알면 (해당 CPU의 전체적인 성능의 valence)를 명확히 알 수 있음.

CPU for Desktop PC

MANUFACTURER
Intel

MODEL
Core i7-6700

PART #
BX80662I76700

DATA WIDTH
64-bit

SOCKET
LGA1151

OPERATING FREQUENCY
3.4GHz

Clk 주파수

MAX TURBO FREQUENCY
4GHz

CORES
4

L1 CACHE
4 x 32KB Instruction
4 x 32KB Data

L2 CACHE
4 x 256KB

L3 CACHE
1 x 8MB

Apple A10 Fusion

Produced	From September 7, 2016 to Present
Designed by	Apple Inc.
Common manufacturer(s)	TSMC[1]
Max. CPU clock rate	to 2.34 GHz[2]
Min. feature size	16 nm
Instruction set	A64, A32, T32
Microarchitecture	Hurricane and Zephyr both ARMv8-A-Compatible
Product code	APL1W24
Cores	Quad-core (2× Hurricane + 2× Zephyr)
L1 cache	Per core: 64 KB instruction + 64 KB data
L2 cache	3 MB shared
L3 cache	4 MB shared

Cache ?

끝이없는여정의시작.

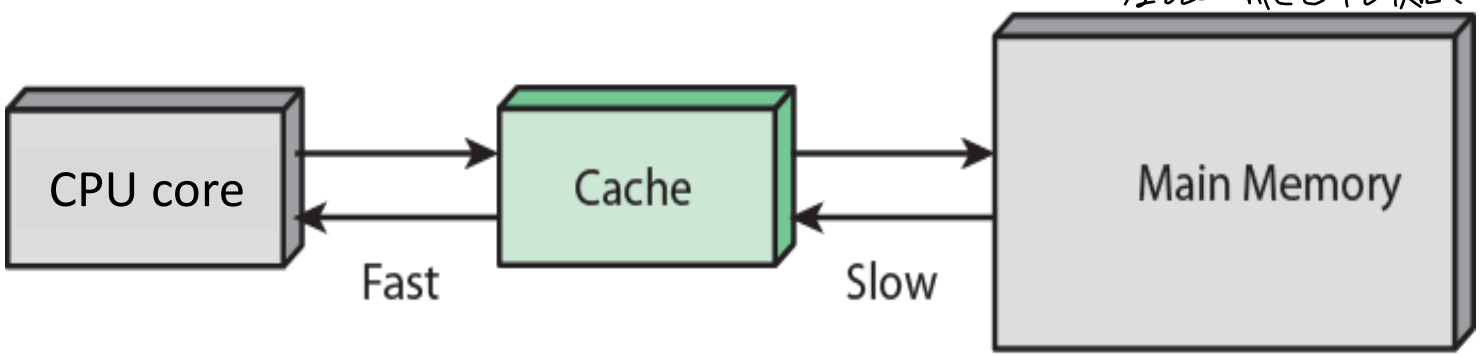
왜 이렇게 구성이 차이있는가!?
L1, L2, L3는 뭐지?
Size 차이는 왜나는거지?

What is Cache Memory?

- Cache memory is an intermediate buffer between CPU Core and main memory (DRAM).
  - CPU = (CPU core + Cache memory) on a chip

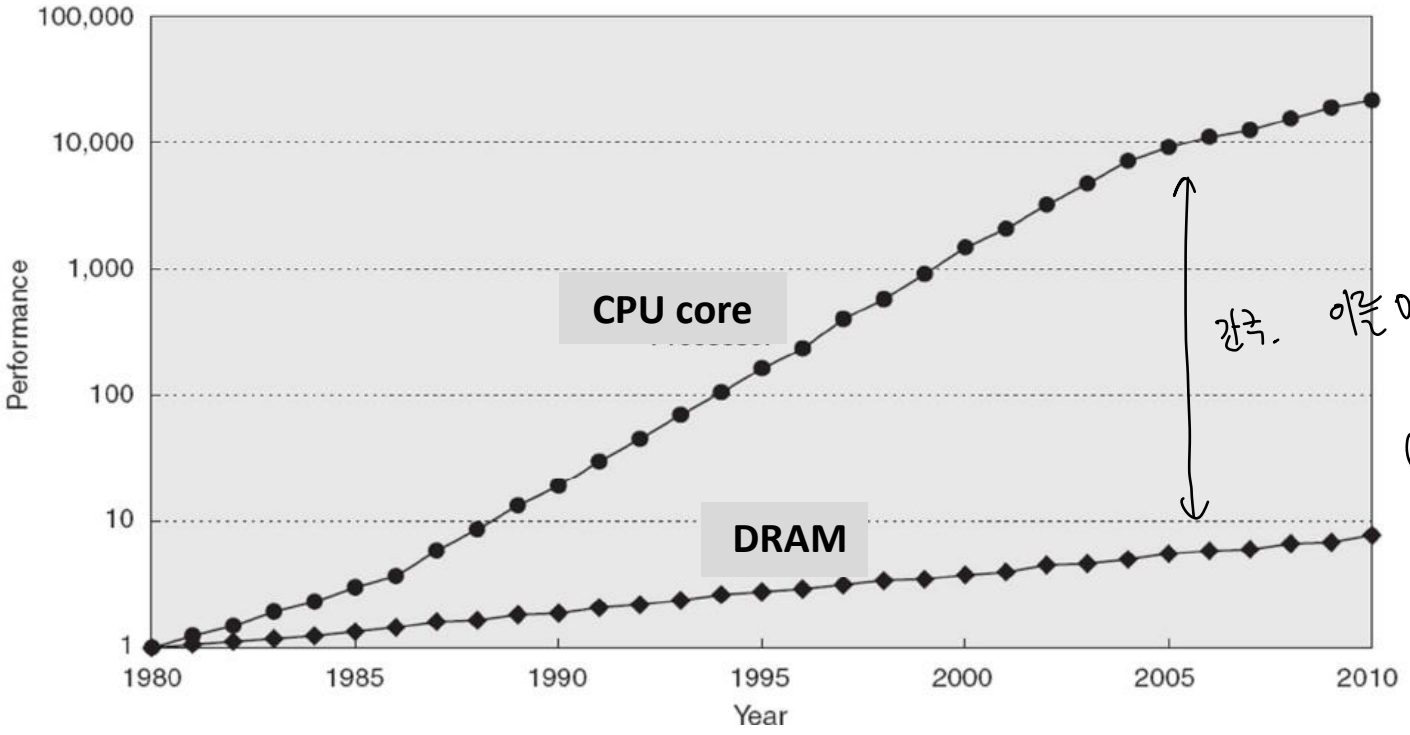
- It is small amount of fast memory (SRAM).

- It may be located on a chip with CPU core or off the chip.



Why do CPUs need Cache Memory?

- **CPU core - DRAM performance gap** (CPU와 DRAM의 성능 gap)
  - Main memory (DRAM) access time is higher than CPU core clock period.
  - Therefore, high speed core's time is wasted during memory access.
  - So it results in significant delay.



Why do CPUs need Cache Memory?

- **To minimize the waiting time for the CPU core**
  - We can consider the fastest memory (SRAM) as main memory.
  - However, unfortunately, SRAM is too expensive for most people to buy a lot of.
  - There is a tradeoff between speed, cost and capacity as follows.

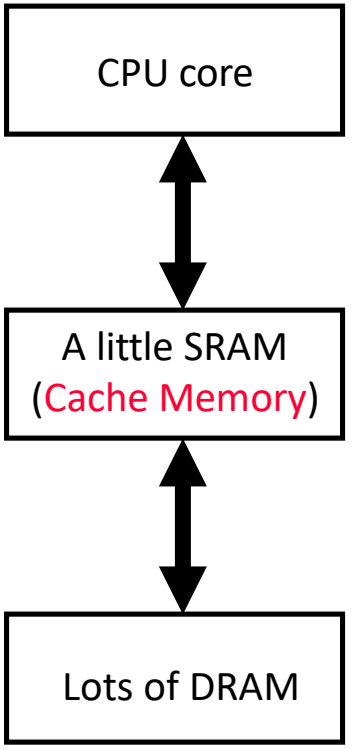
Storage	Speed (Delay)	Cost (Price/MB)	Capacity (Size)
SRAM (Static RAM)	Fastest (1-10 cycles)	Expensive (~$5)	Smallest (24KB-12MB)
DRAM (Dynamic RAM)	Slow (100-200 cycles)	Cheap (~$0.10)	Large (512MB-64GB)

Why do CPUs need Cache Memory?

- **To minimize the waiting time for the CPU core**

  ↳ SRAM과 DRAM의 balance를 잘 조절하면 그게 더 좋지 않을까?

  - Wouldn't it be nice if we could find a balance between fast & expensive memory (SRAM) and slow & cheap memory (DRAM)?
  - We do this by introducing a cache memory, which is a small amount of fast, expensive memory (SRAM).
  - The cache memory goes between the CPU core and main memory (DRAM). ⇒ CPU와 DRAM 사이에 있는 캐시메모리
  - The cache memory keeps a copy of the most frequently used data from the main memory.



메인메모리에서 가장 많이 사용되는 메모리를 캐시메모리에 Keep하는 것

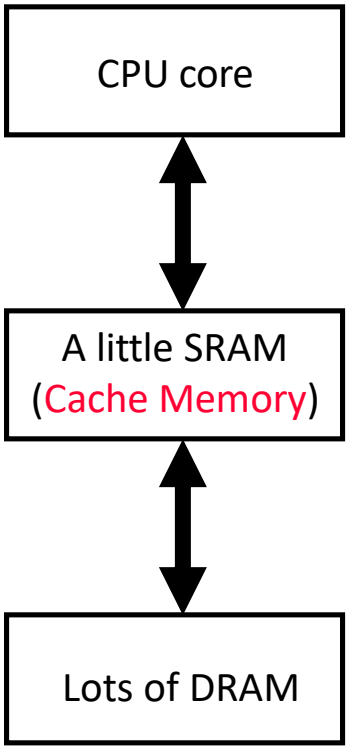
⇓

캐시메모리의 역할

CPU와 DRAM의 gap을 cache memory를 통해서 메꿔준다.

Why do CPUs need Cache Memory?

- **The waiting time for the CPU core is minimized because** (⇒ CPU의 waiting time이 줄어든다)
  - Reads and writes to the most frequently used addresses will be serviced by the cache memory.
    ⇒ 캐시메모리가 자주 사용하는 데이터를 읽고 쓸 수 있게 해줄.
  - We only need to access the slower main memory for less frequently used data.
    ⇒ 덜 사용하는 데이터를 메인메모리는 이용해서 느리게 접근.



일반적 성능개선을 말함.

모든 컴퓨터가 발전할 수 있는 여지 (성능개선)

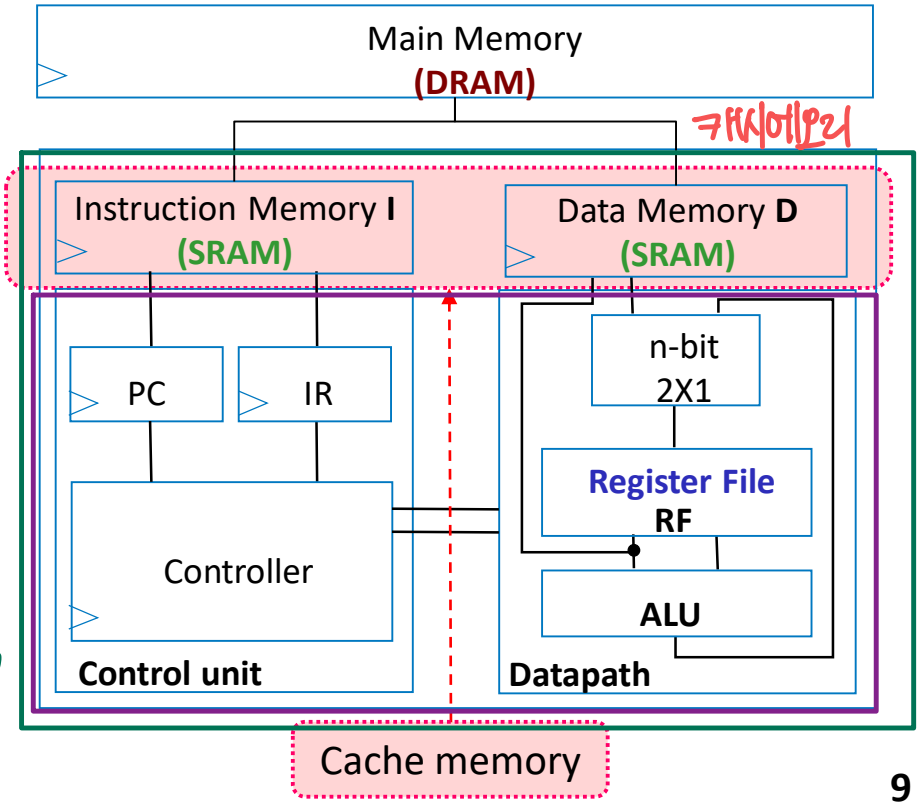
1) CPU clk증가
2) core개수 증가
3) Cache메모리

Why do CPUs need Cache Memory?

- **Recall - Basic Architecture of Programmable Processor**
  (Memory Hierarchy Aspect)

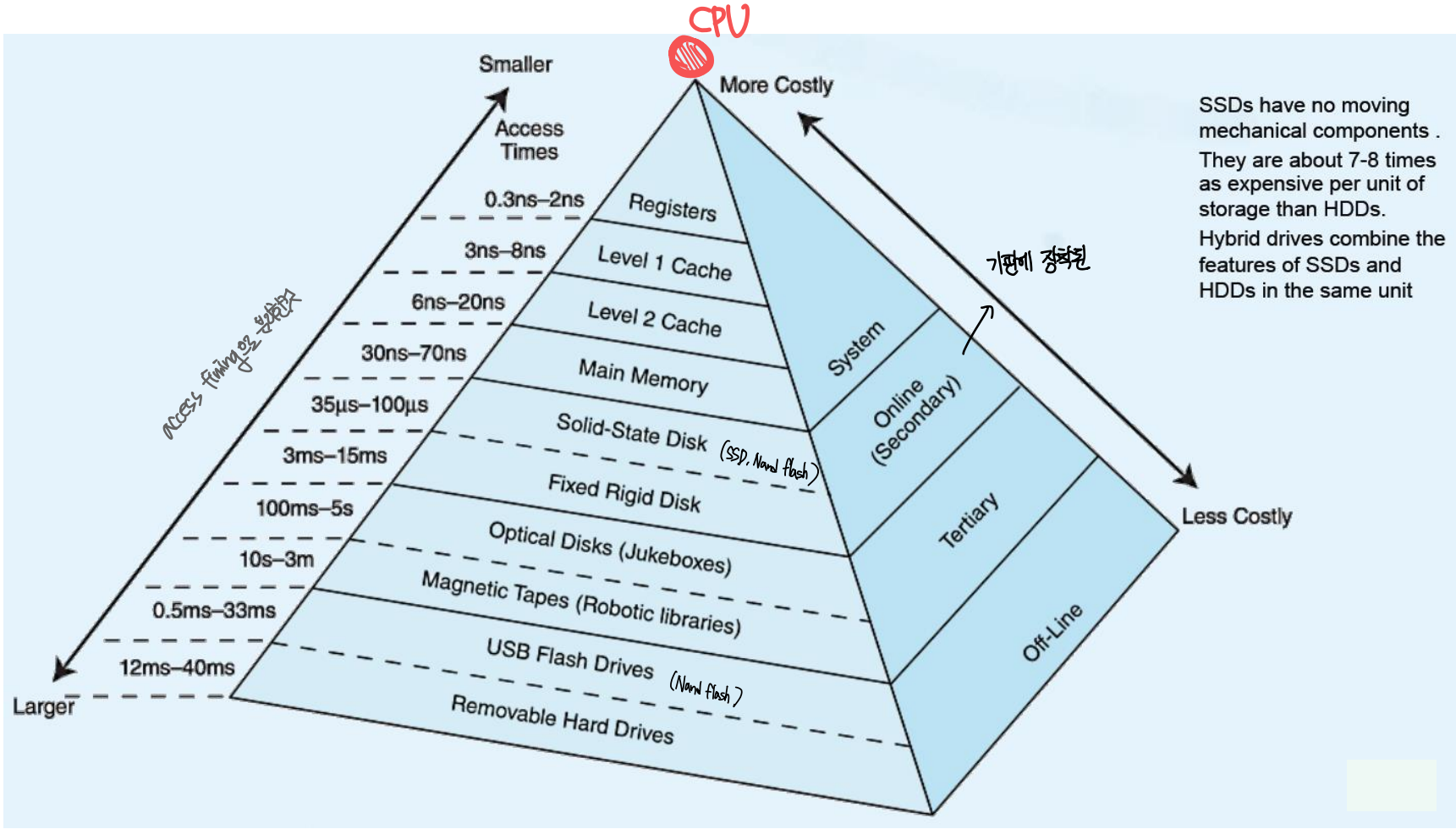
→ 같은 CPU에 만들어지는 것들.
(SRAM, DRAM, 레지스터)

  - **Register File** – 38 transistors for 1-bit
    - Fastest
    - But biggest size
    - Very expensive
  - **SRAM** – 6 transistors for 1-bit
    - Fast
    - More compact than register file
    - Expensive
  - **DRAM** – 1 transistor for 1-bit
    - Slowest
      - And refreshing takes time
    - But very compact
    - Cheap



Why do CPUs need Cache Memory?

- **Memory hierarchy with cache memory**



→ 캐시메모리가 어떤 데이터를 저장하는가?

What Data is stored into Cache Memory?

- **How to know the most frequently used data?** → 어떻게 가장 많이 사용되는 데이터인지알고 캐시에 데이터를 놓는 것인가?
  - The cache memory keeps a copy of the most frequently used data from the main memory.
  - However, It's usually difficult or impossible to figure out what data will be "most frequently used" before a program actually runs. (프로그램의 동작전에)
- **The principle of locality** (접근성의 특징)

  이것에 의해서 캐시메모리에 저장될지 결정됨.
  - In practice, most programs exhibit *locality*, (which the cache memory can take advantage of)
    → 모든 프로그램은 locality를 가지고 있음. 이를 캐시 메모리에서 이용함.
  - ① The principle of temporal locality says that if a program accesses one memory address, there is a good chance that it will access the same address again.
    → 프로그램이 한 메모리 주소에 접근하면, 또 접근할 가능성이 높음 (같은 cycle을 주기로)
  - ② The principle of spatial locality says that if a program accesses one memory address, there is a good chance that it will also access other nearby addresses.
    → 프로그램이 한 메모리 주소에 접근하면, 근처의 다른 주소에 접근할 가능성이 높음. (프로그램은 순차적으로 접근해서)

What Data is stored into Cache Memory?

- **Temporal locality in programs** (⇒ 여기서 program은 "명령어"를 의미함)
  - The principle of temporal locality says that if a program accesses one memory address, there is a good chance that it will access the same address again.
  - Loops are excellent examples of temporal locality in programs.
    - The loop body will be executed many times. ⇒ 계속돌림
    - The computer will need to access those same few locations of the instruction memory repeatedly.
    - For example:

```
Loop: lw   $t0, 0($s1)
      add  $t0, $t0, $s2
      sw   $t0, 0($s1)
      addi $s1, $s1, -4
      bne  $s1, $0, Loop
```

    - Each instruction will be fetched over and over again, once on every loop iteration.

What Data is stored into Cache Memory?

- **Temporal locality in data**
  - Programs often access the same variables over and over, especially within loops. Below, sum and i are repeatedly read and written.

```
sum = 0;
for (i = 0; i < MAX; i++)
    sum = sum + a[i];
```

  - Commonly-accessed variables can sometimes be kept in registers, but this is not always possible because of a limited number of registers.

What Data is stored into Cache Memory?

- **Spatial locality in programs**
  - The principle of spatial locality says that if a program accesses one memory address, there is a good chance that it will also access other nearby addresses.

```
sub  $sp, $sp, 16
sw   $ra, 0($sp)
sw   $s0, 4($sp)
sw   $a0, 8($sp)
sw   $a1, 12($sp)
```

  - Nearly every program exhibits spatial locality, because instructions are usually executed in sequence—if we execute an instruction at memory location *i*, then we will probably also execute the next instruction, at memory location *i+1*.
  - Code fragments such as loops exhibit *both* temporal and spatial locality.

What Data is stored into Cache Memory?

② **Spatial locality in data** (⇒ 여기서 data는 명령어를 제외한 data를 의미함)

- Programs often access data that is stored contiguously.
  - Arrays, like a in the right code, are stored in memory contiguously.
  - The individual fields of a record or object like employee are also kept contiguously in memory.
- Can data have both spatial and temporal locality?
  - Yes, array like b in the right code

"배열" 특징

```
sum = 0;
for (i = 0; i < MAX; i++)
    sum = sum + a[i];
```

배열은 연속적인 메모리 할당

배열.

```
employee.name = "Petter";
employee.boss = "Linda";
employee.age = 45;
```

field ↳ 구조적으로 인접해있음. 한번에 읽어옴.

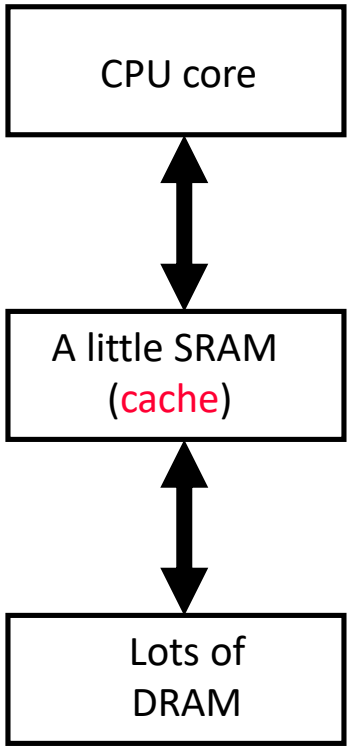
```
sum = 0;
for (i = 0; i < MAX; i++)
   for (j = 0; j < MAX; j++)
    sum = a[i] + b[j];
```

배열 → spatial locality

for loop → temporal locality

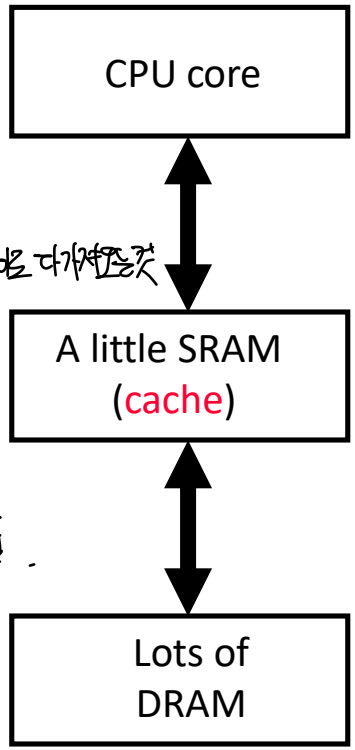
What Data is stored into Cache Memory?

- **How caches take advantage of temporal locality** ⇒ 캐시 메모리가 어떻게 "temporal locality"를 이용하는가!?
  - The first time the CPU core reads from an address in main memory, a copy of that data is also stored in the cache. → 캐시에 메인메모리 데이터를 읽어올 때, 그리고 복사본 저장.
  - The next time that same address is read, we can use the copy of the data in the cache *instead* of accessing the slower dynamic memory.
    ↳ 동일한 주소가 또 들어오면, DRAM에서 읽지 않고 캐시에서 읽어온다.
  - So the first read is a little slower than before since it goes through both main memory and the cache, but subsequent reads are much faster.
    ↳ 처음에는 느리지만 이후 같은 값을 읽어오는데는 매우 빠르다.



What Data is stored into Cache Memory?

- **How caches take advantage of spatial locality**
  - When the CPU core reads location *i* from main memory, a copy of that data is placed in the cache.
  - But instead of just copying the contents of location *i*, we can copy *several* values into the cache at once, such as the four bytes from locations *i* through *i* + 3.
    - If the CPU core later does need to read from locations *i* + 1, *i* + 2 or *i* + 3, it can access that data from the cache and not the slower main memory.
  - Again, the initial load incurs a performance penalty, but we're gambling on spatial locality and the chance that the CPU core will need the contiguous data.



How does Cache Memory Work?

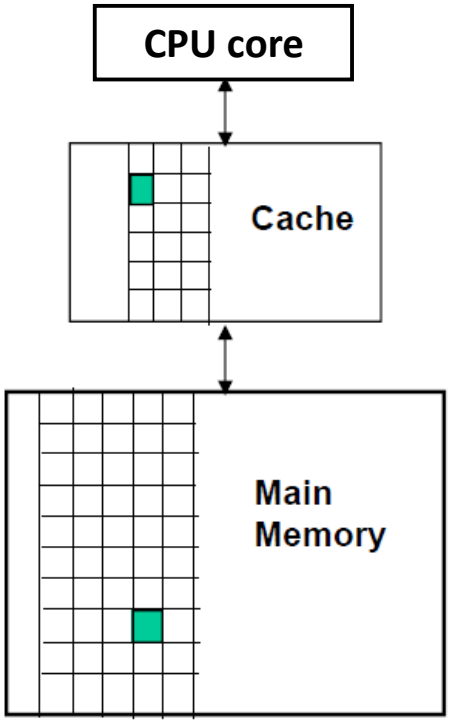
ㅋ 캐시 메모리가 어떻게 작동하는가?

- Basic Concepts

Cache Hit

- If the requested data is found in one of the cache blocks (upper level)
  ⇒ there is a ***hit*** in the cache access

CPU core가 요청한 데이터가 캐시 메모리 안에 들어 있는 상황 ⇒ 캐시 메모리가 히트 했다!!!



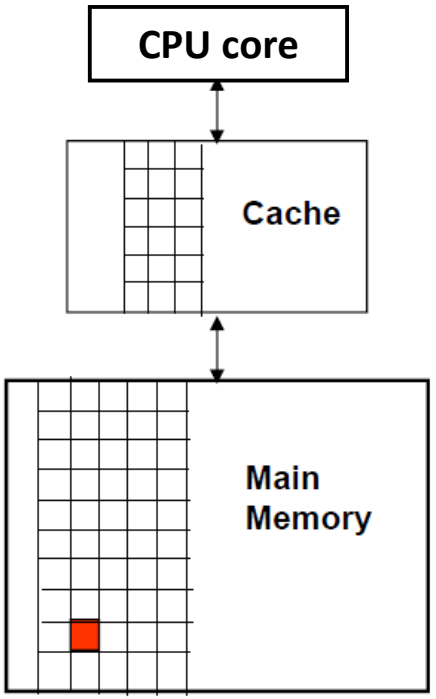
How does Cache Memory Work?

- Basic Concepts

Cache Miss

- If the requested data is not found in in one of the cache blocks (upper level) ⇒ there is a ***miss*** in the cache access
  ⇒ to find the block, we need to access the lower level of the memory hierarchy
- In case of a data miss, we need:
  - ***To stall*** the CPU core; ⇒ CPU core 멈추고
  - To require block from the main memory ⇒ 메인 메모리에 block을 요청함
  - To copy the block in cache; ⇒ 캐쉬에 block copy
  - To repeat the cache access (hit).
    ⇒ 다시 캐시에 접근함.

일반적으로 캐시메모리를 활용하는 방안 (Miss가났을때)



CPU에서 필요한 데이터가 캐시메모리안에 없을경우 ⇓ 캐시 메모리가 Miss !!

How does Cache Memory Work?

- Basic Concepts

⇒ 캐시메모리는 hit, miss를 가지고 성능을 평가할 수 있음.

- **There are two basic measurements of cache performance.**
  - The hit rate is the percentage of memory accesses that are handled by the cache memory.
  - The miss rate (1 − hit rate) is the percentage of accesses that must be handled by the slower main memory.
    ↳ 늦어진 좋지 않은 것임.

(hit-rate) + (miss-rate) = 1

- **Typical caches have a hit rate of 95% or higher, so in fact most memory accesses will be handled by the cache memory and will be dramatically faster.**

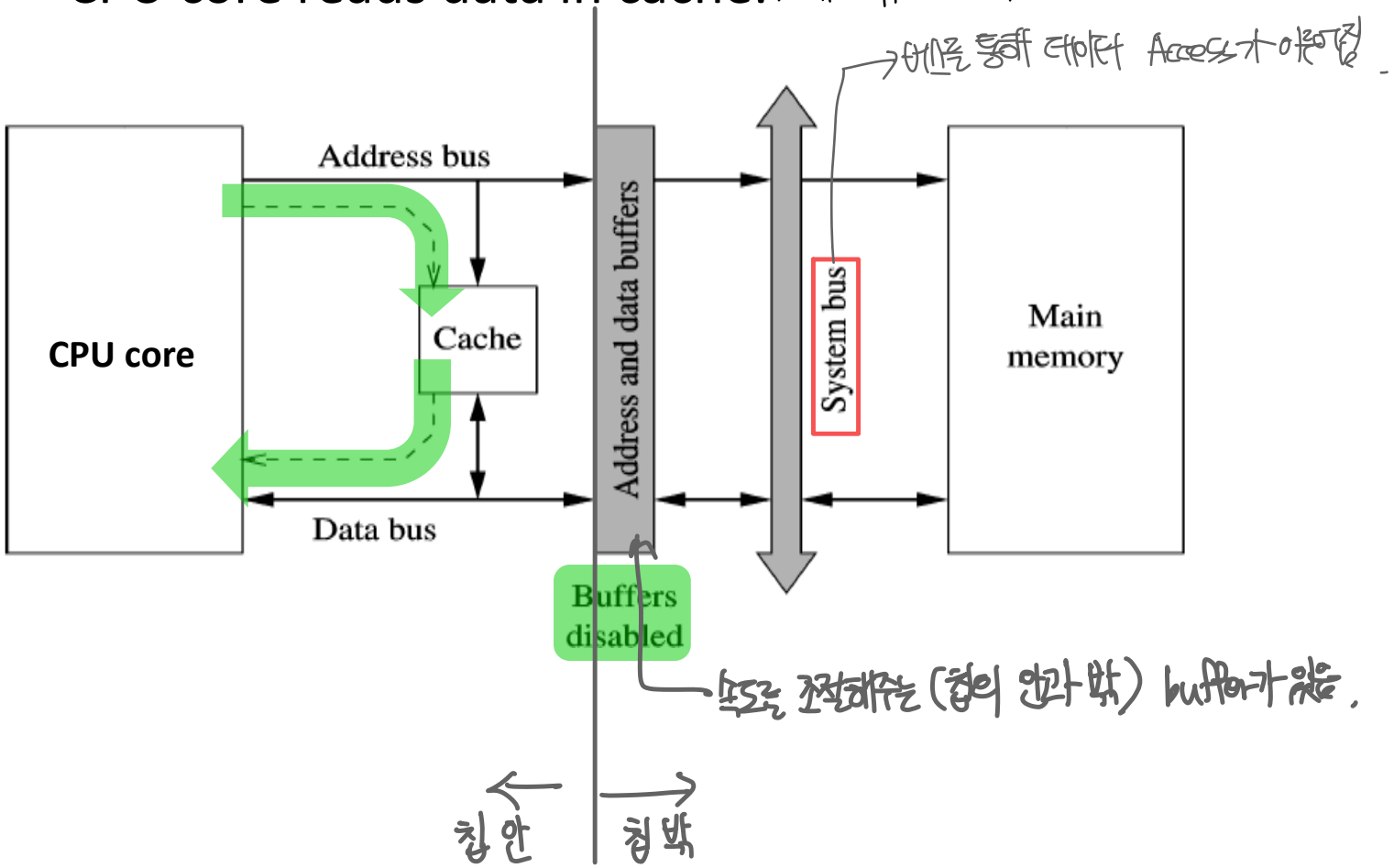
⇒ 일반적으로 hit-rate은 95%를 넘음.
  ↳ 이를 높이면 성능 개선이 가능함.
  ↳ 개선은 5%임.

How does Cache Memory Work?

HIT

- Cache-Read Operation

- **Read hit**
  - CPU core reads data in cache. ⇒ 버스를 통해서 캐시에서 데이터를 읽어옴.

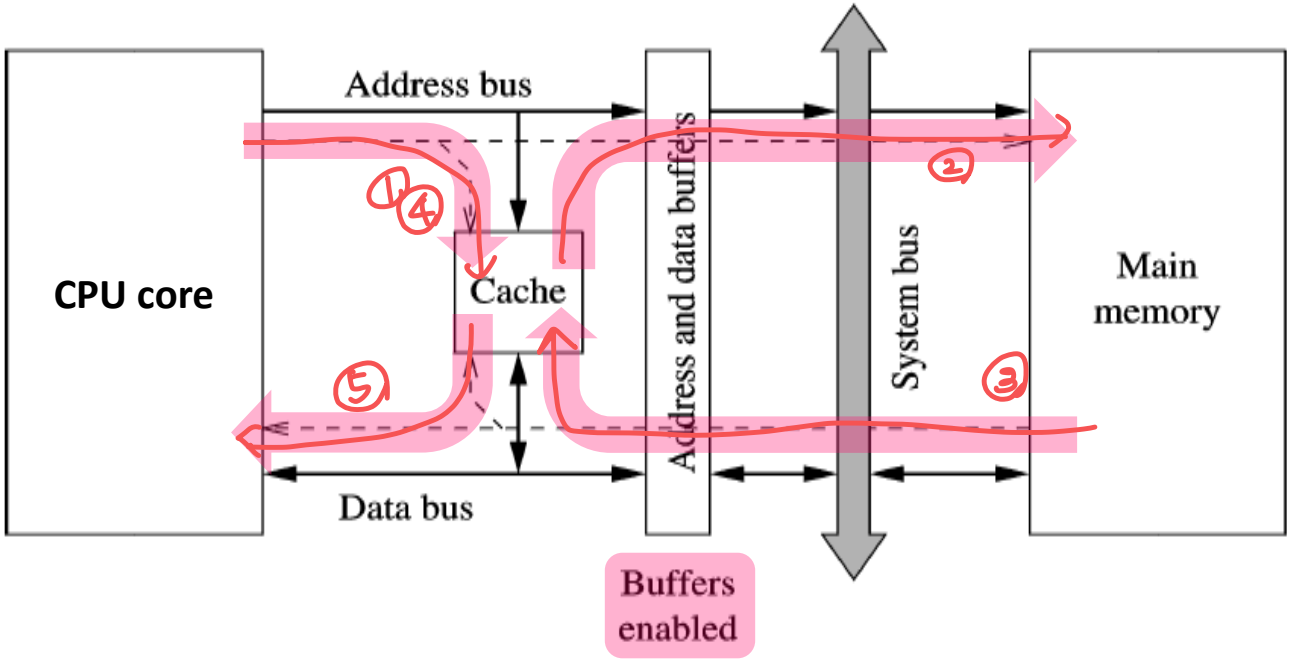


How does Cache Memory Work?

MISS

- Cache-Read Operation

- **Read miss** ⇒ 캐시메모리에 필요한 데이터가 없는 상황
  - CPU core stalls and requests the data to main memory. Core 멈추고 메인메모리에 요청.
  - The data is copied in cache memory. 캐시메모리에 데이터 복사
  - CPU core repeats cache-read operation. CPU core가 캐시메모리로 읽어옴.



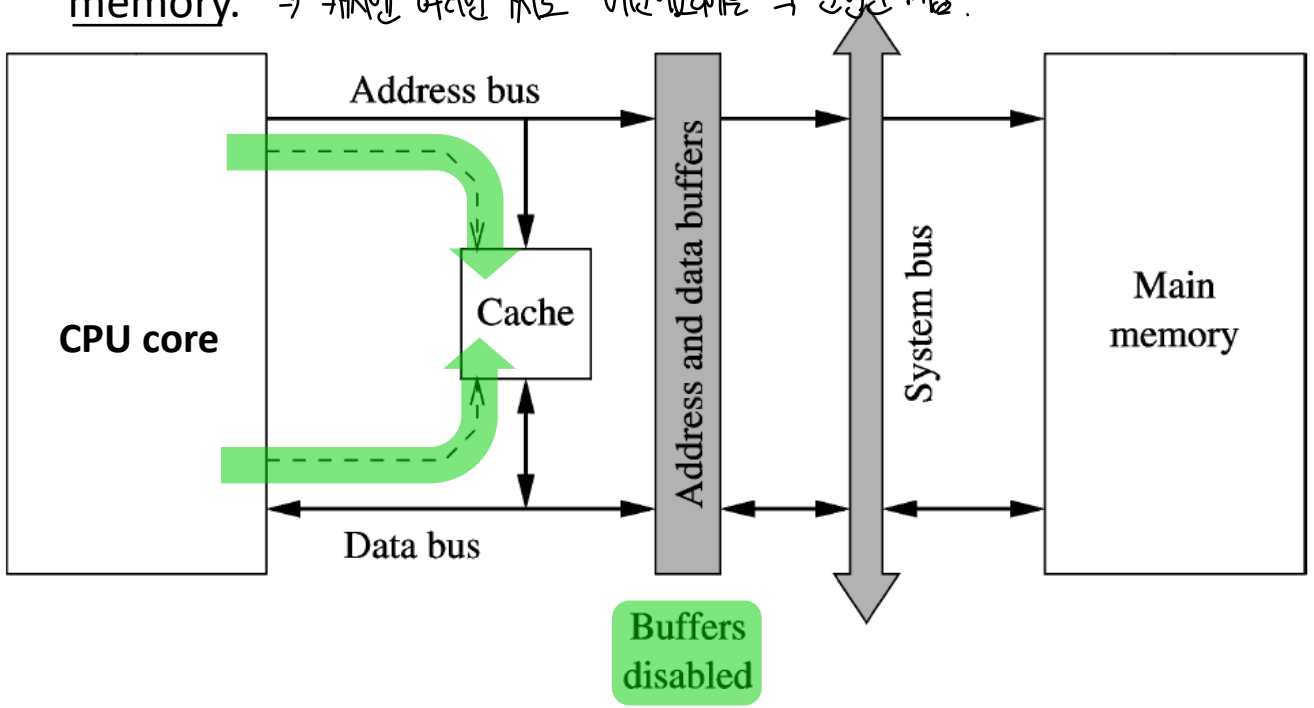
How does Cache Memory Work?

HIT

write는 hit 방식 2가지, miss방식 2가지가있음.

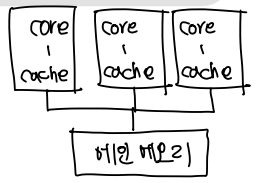
- Cache-Write Operation

- **Write hit** : write-back ① ⇒ 변경된 데이터는 바로 메인메모리에 쓰지 않음.
  - CPU core writes data in cache memory only. ⇒ 캐시 메모리에 CPU core가 데이터를 저장.
  - The modified data is written to the main memory when it is replaced due to a miss ⇒ miss가 난 경우에만 메인메모리에 변경된 데이터를 써줌 (1번만) → 그렇지않은경우 계속 캐시와 core만 주고받음.
    - Multiple writes to the same data require only a single write to the main memory. ⇒ 캐시엔 여러번 써도 메인메모리에는 딱 한번만 써줌.



메인메모리에 접근하는 횟수줄임

* CPU가 multi core인경우



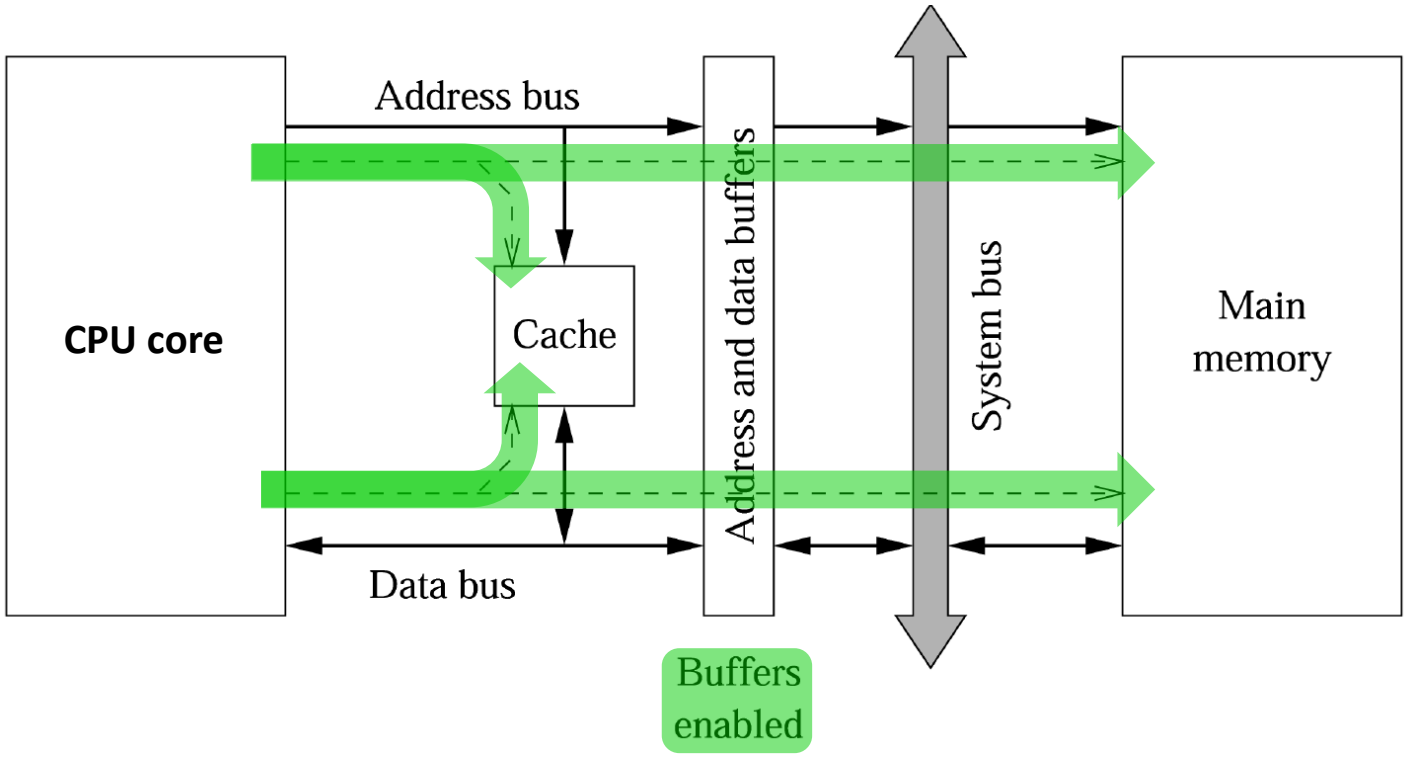
메인메모리를 공유하므로 write back 방식이인경우 두번씀.

How does Cache Memory Work?

HIT

- Cache Write Operation

- **Write hit**: write-through ② ⇒ 캐시메모리에도 값을 쓰고 동시에 메인메모리에도 값을 쓰는 것.
  - CPU core writes data both in cache memory and in main memory.
    ↳ 둘다 업데이트 되므로 multi-core에서도 사용가능.



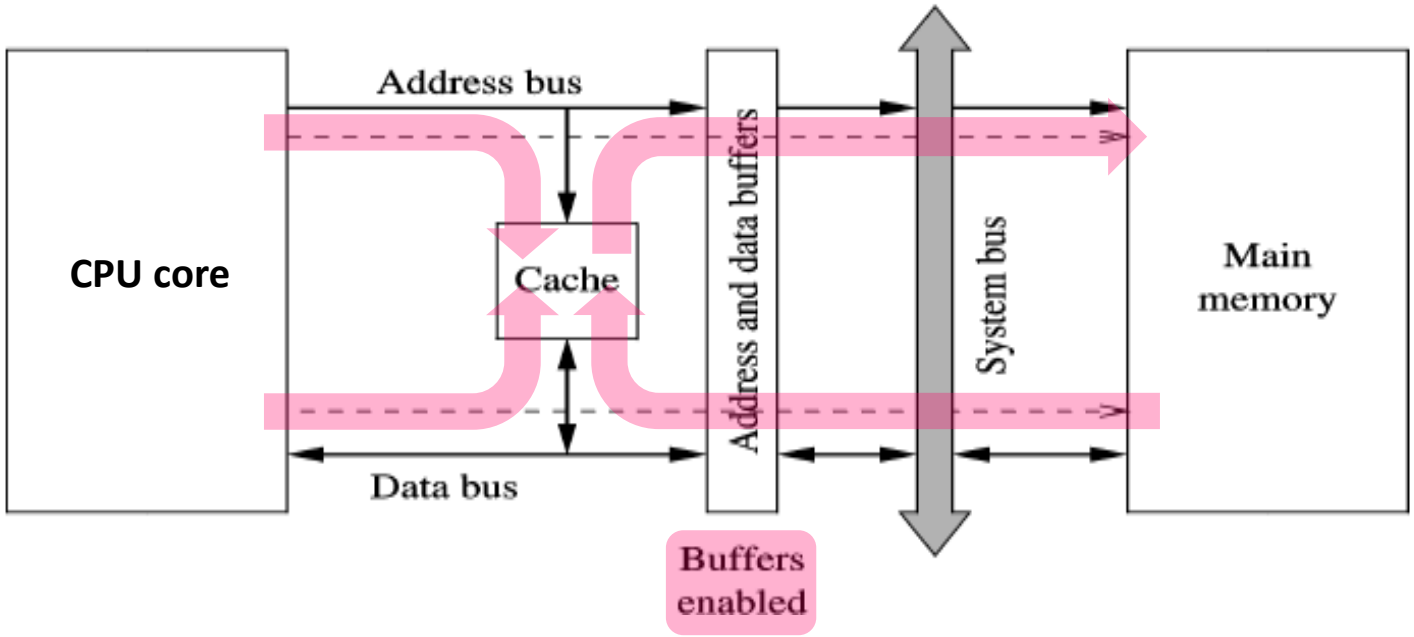
How does Cache Memory Work?

MISS

- Cache-Write Operation

→ 캐시메모리에 쓰고자 하는데 공간이 없는 경우

- **Write miss** ① : write-allocate
  - CPU core stalls and requests the data to main memory. ⇒ core 멈추고 메인메모리에서 읽어옴.
  - The data is copied in cache memory. ⇒ 캐시메모리에 데이터 복사함.
  - CPU core repeats cache-write operation. ⇒ 그후에 CPU core가 다시 캐시메모리에 write함.



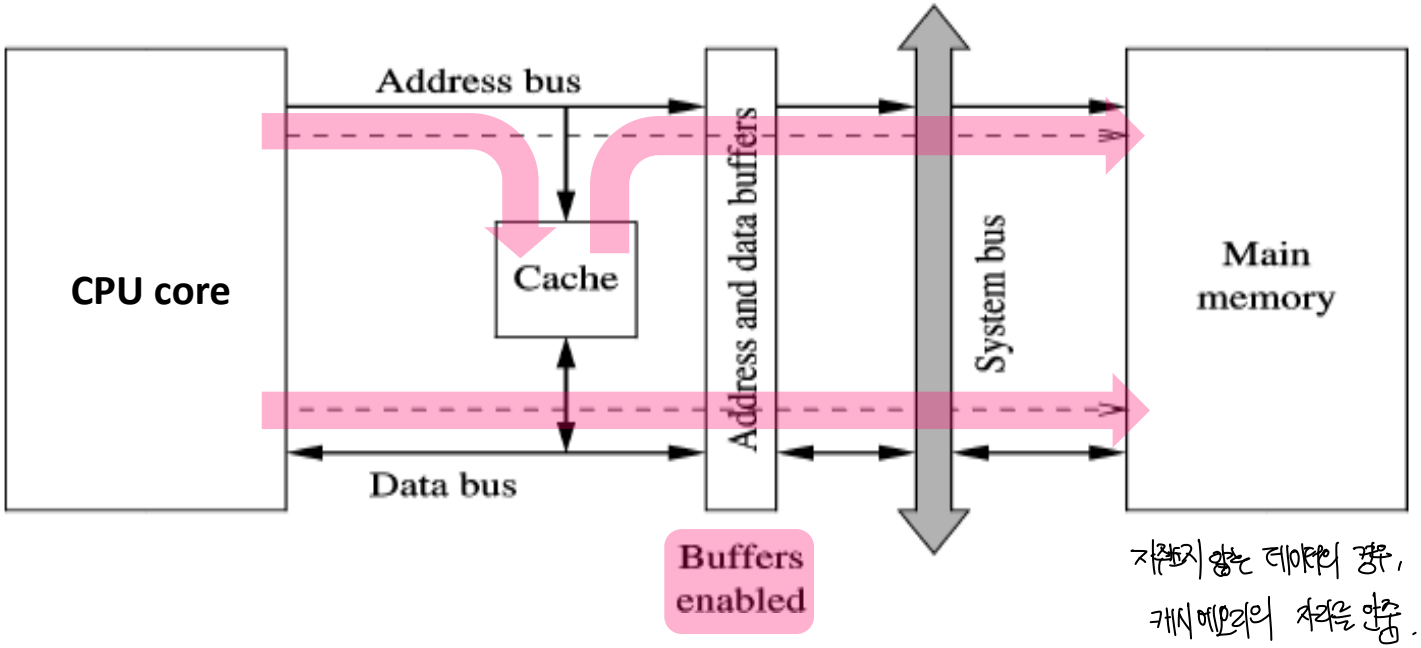
How does Cache Memory Work?

MISS

- Cache-Write Operation

- **Write miss** ②: no write-allocate ⇒ 캐시메모리에 해당 address의 공간을 내어주지 않는 것.
  - CPU core stalls and simply send write-data to main memory.

CPU 멈추고 메인메모리 address에 보내준 뒤 CPU에서 메인메모리로 데이터를 보내줄.



자주쓰지 않는 데이터의 경우, 캐시메모리의 자리를 안줌. ⇒ no write-allocate

⇓

프로그램에서는 100% 예측할 수 없으나 대부분의 경우에서 ① write-allocate or ② no write-allocate 인지는 판정할.

수행은 SW, HW로직이 결정함.

How does Cache Memory Work?

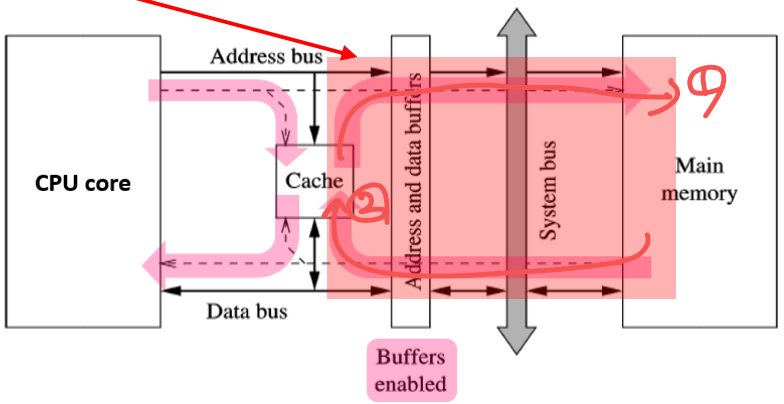
- Cache Memory Space Allocation

- **How to allocate cache memory space for data from main memory?** ⇒ 어떻게 캐시메모리에서 allocate 하는가?
  - When?

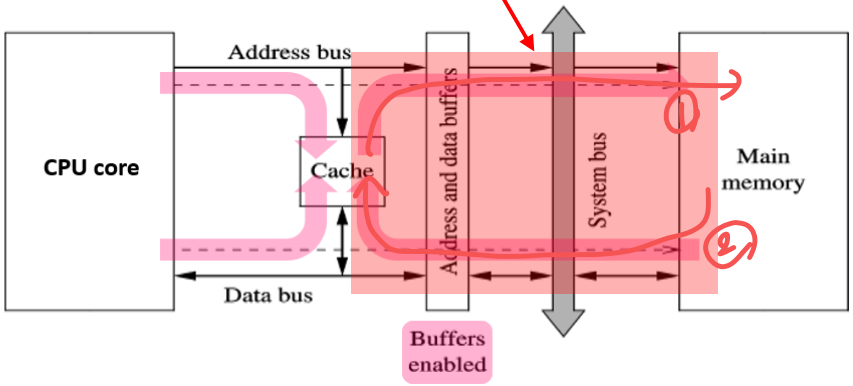
언제

과지상황

- **Read miss**
  - CPU core stalls and requests the data to main memory.
  - The data is copied in cache memory.
  - CPU core repeats cache-read operation.



- **Write miss** : write-allocate
  - CPU core stalls and requests the data to main memory.
  - The data is copied in cache memory.
  - CPU core repeats cache-write operation.



How does Cache Memory Work?

- Cache Memory Space Allocation

- **How to allocate cache memory space for data from main memory?**

  – In the aspect of data-placement & identification
    - Directed mapped
    - Fully associative
    - N-Way Set associative

  – In the aspect of data-replacement
    - LRU (Least Recently Used)
    - FIFO (First In First Out)
    - Random