CS510 Computer Architecture

Lecture 13: Review: Cache Memory II

Soontae Kim

Spring 2017

School of Computing, KAIST

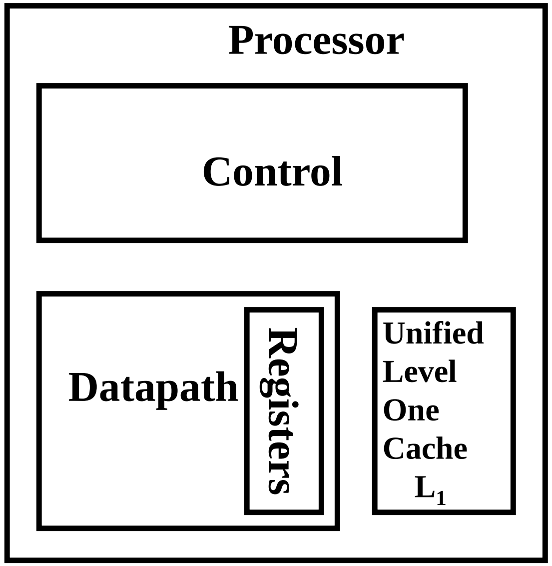
Notice

- **Term project proposal**
  - On May 9 (Wednesday)
  - Prepare less than 10 slides in 5 minutes.
    - **No proposal report required**
  - All team members must prepare parts of presentation.
  - Any topic related to computer architecture; if you are not sure that your topic is appropriate, you can discuss with me.
  - Practice your presentation before coming to class several times so that you can finish your presentation in time; we have only 75 minutes for all of your presentations.
  - You may need to change your topic or direction if your topic conflicts with other team's topic or you find a difficulty in performing your projects.

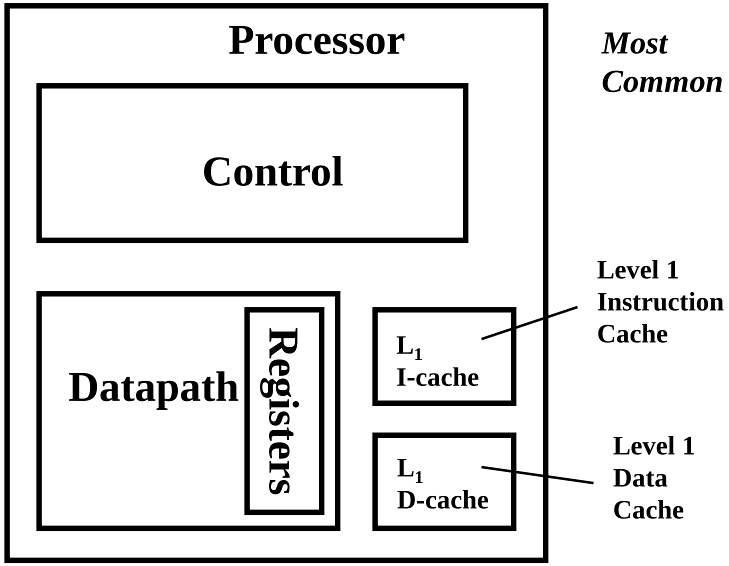
Unified vs. Separate Level 1 Cache

- **Unified Level 1 Cache (Princeton Memory Architecture).**
  A single level 1 (L_1) cache is used for both instructions and data.

- **Separate instruction/data Level 1 caches (Harvard Memory Architecture):**
  The level 1 (L_1) cache is split into two caches, one for instructions (instruction cache, L_1 I-cache) and the other for data (data cache, L_1 D-cache).



Unified Level 1 Cache
(Princeton Memory Architecture)



Separate (Split) Level 1 Caches
(Harvard Memory Architecture)

Memory Hierarchy Performance:
Average Memory Access Time (AMAT), Memory Stall cycles

- **<u>The Average Memory Access Time (AMAT):</u> The number of cycles required to complete an average memory access request by the CPU.**
- **<u>Memory stall cycles per memory access:</u> The number of stall cycles added to CPU execution cycles for one memory access.**
- $\boxed{\text{Memory stall cycles per average memory access} = (\text{AMAT} - 1)}$
- **For ideal memory: AMAT = 1 cycle, this results in zero memory stall cycles.**
- **Memory stall cycles per average instruction =**

Number of memory accesses per instruction

x Memory stall cycles per average memory access

Instruction Fetch from \$

= (1 + fraction of loads/stores) x (AMAT -1)

$$\text{Base CPI} = \text{CPI}_{\text{execution}} = \text{CPI with ideal memory}$$

$$\boxed{\text{CPI} = \text{CPI}_{\text{execution}} + \text{Mem Stall cycles per instruction}}$$

Cache Performance:

Single Level L1 Princeton (Unified) Memory Architecture

CPUtime = Instruction count x CPI x Clock cycle time

$\boxed{CPI_{execution} = \text{CPI with ideal memory}}$

CPI = $CPI_{execution}$ + Mem Stall cycles per instruction

Mem Stall cycles per instruction =

Memory accesses per instruction x Memory stall cycles per memory access

Assuming no stall cycle on a cache hit (cache access time = 1 cycle, stall = 0)

Cache Hit Rate = H1 Miss Rate = 1- H1

Memory stall cycles per memory access = Miss rate x Miss penalty

AMAT = 1 + Miss rate x Miss penalty

Memory accesses per instruction = (1 + fraction of loads/stores)

Miss Penalty = M = the number of stall cycles resulting from missing in cache

Thus for a unified L1 cache with no stall on a cache hit:

CPI = $CPI_{execution}$ + (1 + fraction of loads/stores) x (1 - H1) x M

AMAT = 1 + (1 - H1) x M

Memory Access Tree:
For Unified Level 1 Cache

Probability to be here

CPU Memory Access

H1 — 100% — (1-H1)

L_1

L1 Hit:
% = Hit Rate = H1
Hit Access Time = 1
Stall cycle per access = 0
Stall= H1 x 0 = 0
(No Stall)

L1 Miss:
% = (1- Hit rate) = (1-H1)
Access time = M + 1
Stall cycles per access = M
Stall = M x (1-H1)

AMAT = H1 x 1 + (1 -H1) x (M+ 1) = 1 + M x (1 -H1)

Stall Cycles Per Access = AMAT - 1 = M x (1 -H1)

CPI = $CPI_{execution}$ + (1 + fraction of loads/stores) x M x (1 -H1)

M = Miss Penalty = stall cycles per access resulting from missing in cache
H1 = Level 1 Hit Rate
1- H1 = Level 1 Miss Rate

Cache Performance Example

- Suppose a CPU executes at Clock Rate = 200 MHz (5 ns per cycle) with a single level of cache.
- $CPI_{execution}$ = 1.1
- Instruction mix: 50% arith/logic, 30% load/store, 20% control
- Assume a cache miss rate of 1.5% and a miss penalty of M= 50 cycles.

$$CPI = CPI_{execution} + \text{mem stalls per instruction}$$

Mem Stalls per instruction =

Mem accesses per instruction x Miss rate x Miss penalty

Mem accesses per instruction = 1 + .3 = 1.3

Instruction fetch — Load/store

Mem Stalls per memory access = (1- H1) x M = .015 x 50 = .75 cycles

AMAT = 1 +.75 = 1.75 cycles

Mem Stalls per instruction = 1.3 x .015 x 50 = 0.975

CPI = 1.1 + .975 = 2.075

The ideal memory CPU with no miss is 2.075/1.1 = 1.88 times faster

M = Miss Penalty = stall cycles per access resulting from missing in cache

Cache Performance Example

- **Suppose for the <u>previous example</u> we <u>double the clock rate</u> to 400 MHz, how much faster is this machine, assuming similar miss rate, instruction mix?**
- **Since memory speed is not changed, the miss penalty takes more CPU cycles:**

Miss penalty = M = 50 x 2 = 100 cycles.

CPI = 1.1 + 1.3 x .015 x 100 = 1.1 + 1.95 = 3.05

Speedup = $(CPI_{old} \times C_{old}) / (CPI_{new} \times C_{new})$

= 2.075 x 2 / 3.05 = 1.36

The new machine is only 1.36 times faster rather than 2 times faster due to the increased effect of cache misses.

→ ***CPUs with higher clock rate, have more cycles per cache miss and more memory impact on CPI.***

Cache Performance

Harvard Memory Architecture

For a CPU with separate or <u>split level one (L1)</u> caches for instructions and data (Harvard memory architecture) and no stalls for cache hits:

CPUtime = Instruction count x CPI x Clock cycle time

$$\boxed{\text{CPI} = \text{CPI}_{\text{execution}} + \text{Mem Stall cycles per instruction}}$$

Mem Stall cycles per instruction =

Instruction Fetch Miss rate x M +

Data Memory Accesses Per Instruction x Data Miss Rate x M

M = Miss Penalty = stall cycles per access to main memory resulting from missing in cache

Memory Access Tree
For Separate Level 1 Caches

CPU Memory Access

L_1

% Instructions — **100%** — **% data**

Instruction

%instructions x Instruction H1

%instructions x (1 - Instruction H1)

<u>Instruction L1 Hit:</u>
Hit Access Time = 1
Stalls = 0

<u>Instruction L1 Miss</u>:
Access Time = M + 1
Stalls Per access = M
%instructions x (1 - Instruction H1) x M

Data

% data x Data H1

% data x (1 - Data H1)

<u>Data L1 Hit:</u>
Hit Access Time: = 1
Stalls = 0

<u>Data L1 Miss:</u>
Access Time = M + 1
Stalls per access: M
Stalls = % data x (1 - Data H1) x M

Stall Cycles Per Access = % Instructions x (1 - Instruction H1) x M + % data x (1 - Data H1) x M

AMAT = 1 + Stall Cycles per access

CPI = $CPI_{execution}$ + (1 + fraction of loads/stores) x Stall Cycles per access

Cache Performance Example

- **Suppose a CPU uses separate level one (L1) caches for instructions and data (Harvard memory architecture) with different miss rates for instruction and data access:**
  - **A cache hit incurs no stall cycles while a cache miss incurs 200 stall cycles for both memory reads and writes.**
  - **$CPI_{execution}$ = 1.1**
  - **Instruction mix: 50% arith/logic, 30% load/store, 20% control**
  - **Assume a cache miss rate of 0.5% for instruction fetch and a cache data miss rate of 6%.**
  - **Find the resulting CPI using this cache? How much faster is the CPU with ideal memory?**

$$CPI = CPI_{execution} + \text{mem stalls per instruction}$$

Mem Stall cycles per instruction = Instruction Fetch Miss rate x Miss Penalty + Data Memory Accesses Per Instruction x Data Miss Rate x Miss Penalty

Mem Stall cycles per instruction = 0.5/100 x 200 + 0.3 x 6/100 x 200 = 1 + 3.6 = 4.6
Mem Stall cycles per access = 4.6 / 1.3 = 3.5 cycles AMAT = 1 + 3.5 = 4.5 cycles

$CPI = CPI_{execution}$ + mem stalls per instruction = 1.1 + 4.6 = 5.7

The CPU with ideal cache (no misses) is 5.7/1.1 = 5.18 times faster

With no cache the CPI would have been = 1.1 + 1.3 X 200 = 261.1 !!

Typical Cache Performance Data Using SPEC92

Size	Instruction cache	Data cache	Unified cache
1 KB	3.06%	24.61%	13.34%
2 KB	2.26%	20.57%	9.78%
4 KB	1.78%	15.94%	7.24%
8 KB	1.10%	10.19%	4.57%
16 KB	0.64%	6.47%	2.87%
32 KB	0.39%	4.82%	1.99%
64 KB	0.15%	3.77%	1.35%
128 KB	0.02%	2.88%	0.95%

Miss rates for instruction, data, and unified caches of different sizes.

Types of Cache Misses: *The Three C's*

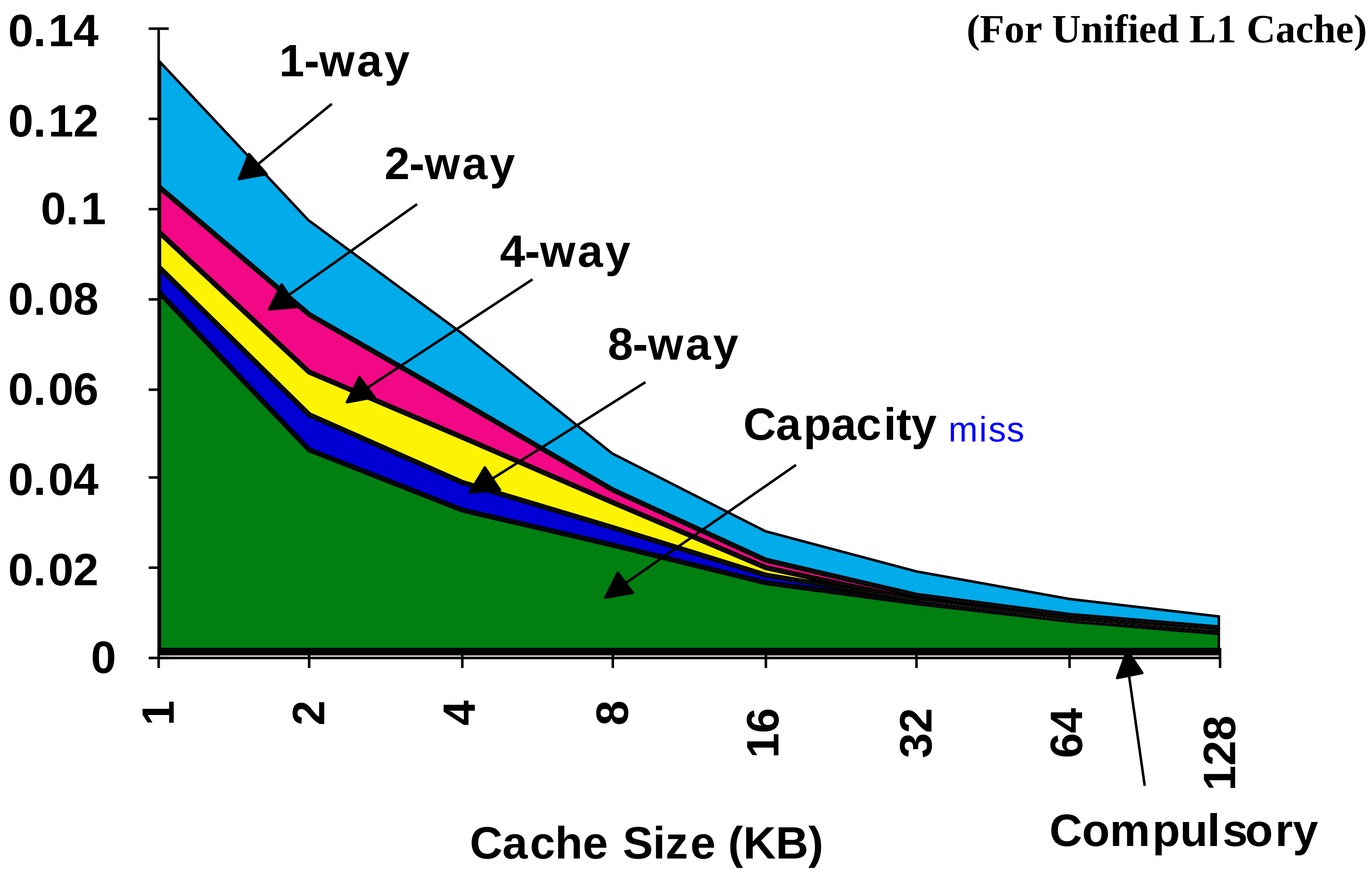
1 **Compulsory:** **On the first access to a block; the block must be brought into the cache; also called cold start misses, or first reference misses.**

- **Initially upon program startup: Miss rate ~ 100% All compulsory misses**

2 **Capacity:** **Occur because blocks are being discarded from cache because cache cannot contain all blocks needed for program execution (program working set is much larger than cache capacity).**

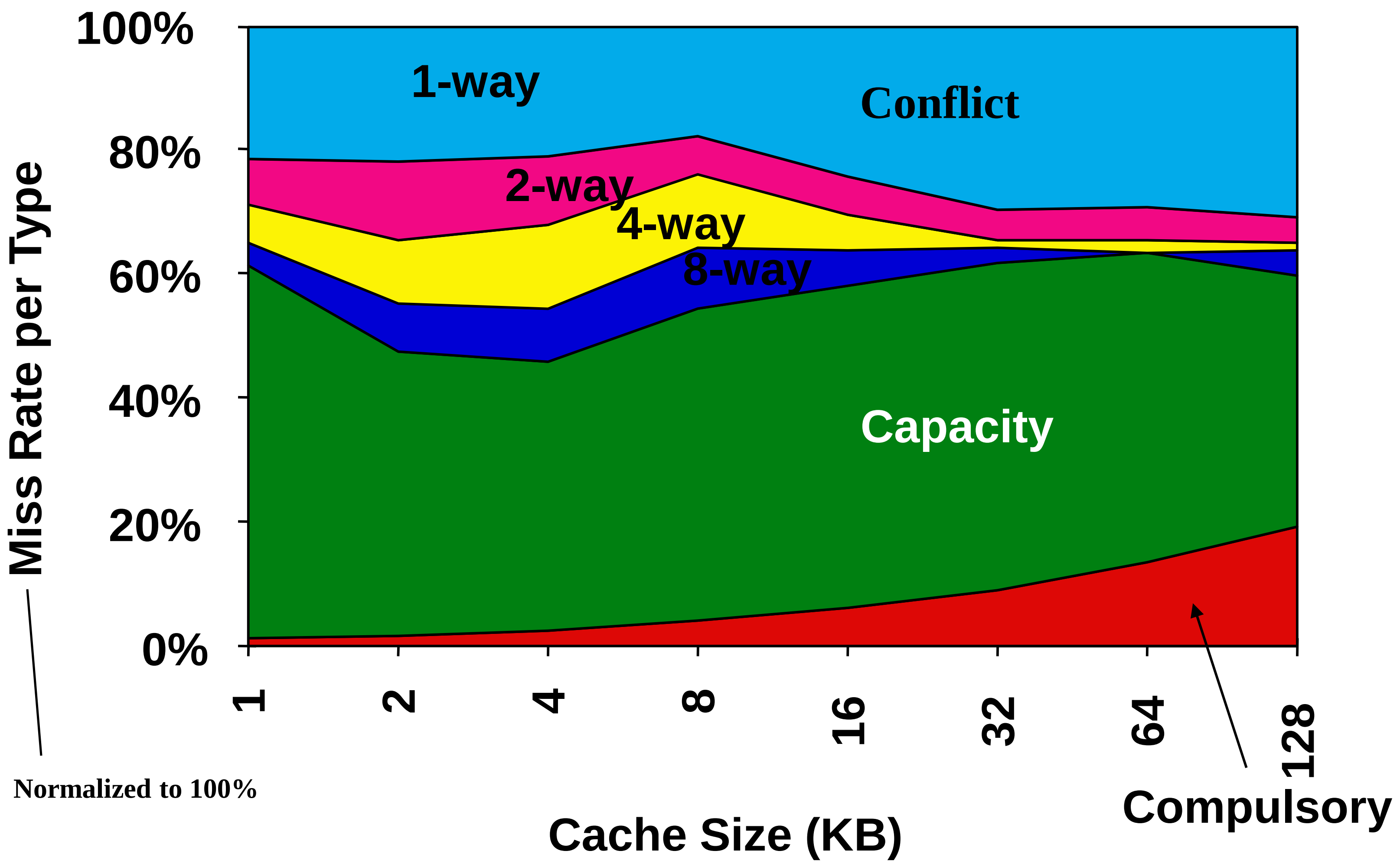
3 **Conflict:** **In the case of set associative or direct mapped block placement strategies, conflict misses occur when several blocks are mapped to the same set or block frame; also called collision misses or interference misses.**

The 3 Cs of Cache:
Steady State Miss Rates (SPEC92)



The 3 Cs of Cache:

Relative Steady State Miss Rates (SPEC92)



Cache Read/Write Operations

- **Statistical data suggest that reads (*including instruction fetches)* dominate processor cache accesses (writes account for ~ 30% of data cache traffic).**
- **In cache reads, a cache block is read at the same time while the tag is being compared with the block address. If the read is a hit the data is passed to the CPU, otherwise ignores it.**
- **In cache writes, modifying the block cannot begin until the tag is checked to see if it is a hit.**
- **Thus for cache writes, <u>tag checking cannot take place in parallel</u>, and only the specific data (between 1 and 8 bytes) requested by the CPU can be modified.**
- **Cache can be classified according to the write and memory update strategy: <u>write through</u>, or <u>write back</u>.**

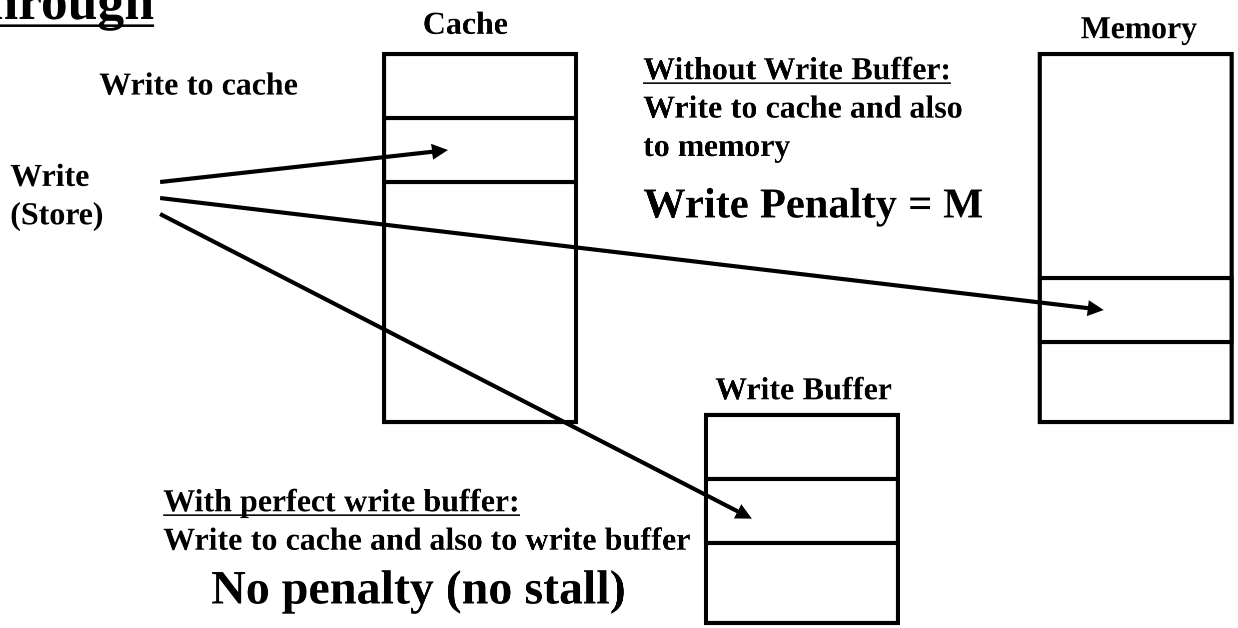
Cache Write Strategies

1. **Write Through: Data is written to both the cache block and to main memory block.**
   - **The lower level always has the most updated data; an important feature for I/O and multiprocessing.**
   - **Easier to implement than write back.**
   - **A write buffer is often used to reduce CPU write stall cycles while data are written to memory.**
2. **Write Back: Data are written or updated only to the cache block. The modified or dirty cache block is written to main memory when it's being replaced from cache.**
   - **Writes occur at the speed of cache**
   - **A status bit called a dirty or modified bit, is used to indicate whether the block was modified while in cache; if not the block is not written back to main memory when replaced.**
   - **Advantage: Uses less memory bandwidth than write through.**

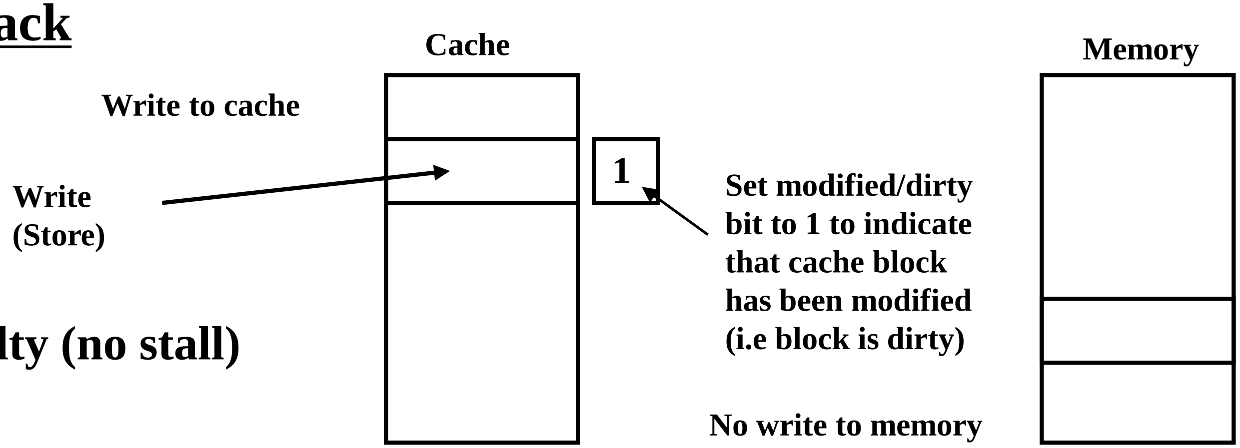
Cache Write Strategies:

Write Hit Operation (block to be written is in cache)

Write Through



Write Back



No penalty (no stall)

Cache Write Miss Policy

- **Since data are usually not needed immediately on a write miss, two options exist on a cache write miss:**

Write Allocate:

The cache block is loaded on a write miss followed by write hit actions.

No-Write Allocate:

The block is modified in the lower level (lower cache level, or main memory) and not loaded into cache.

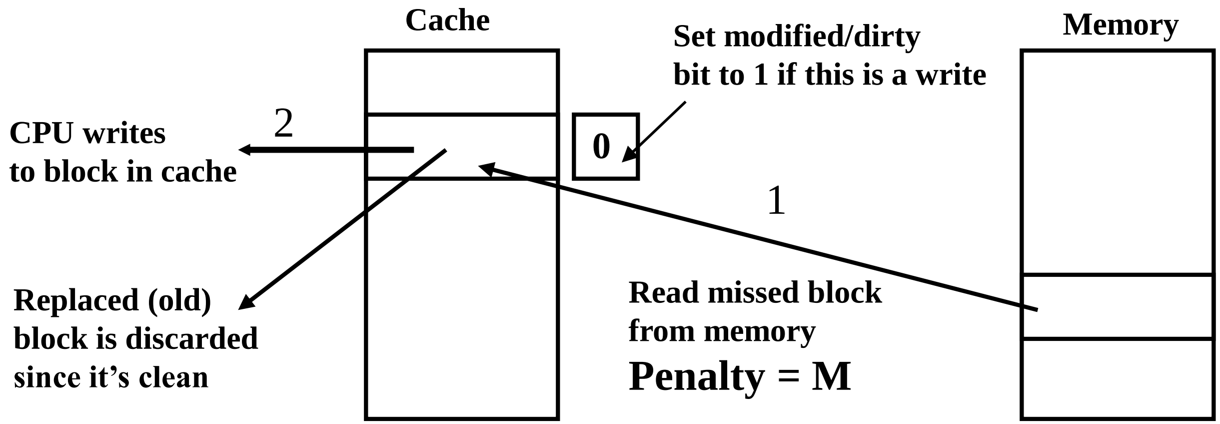
While any of the above two write miss policies can be used with either write back or write through:

- **Write back caches always use write allocate to capture subsequent writes to the block in cache.**
- **Write through caches usually use no-write allocate since subsequent writes still have to go to memory.**

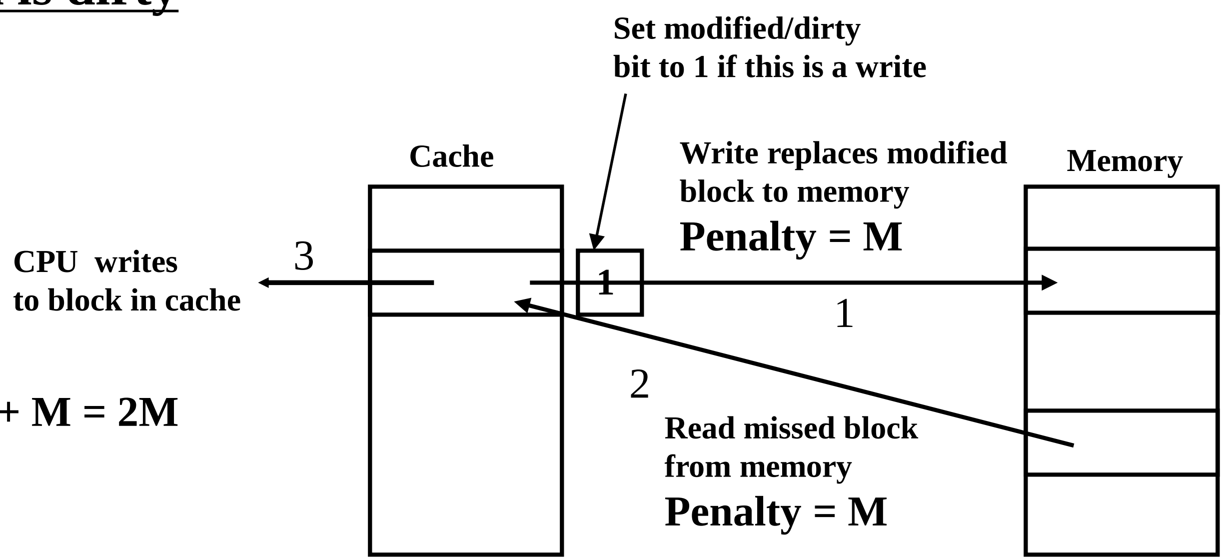
Write Back Cache With Write Allocate: Cache Miss Operation

Block to be replaced is clean

Miss Penalty = M



Block to be replaced is dirty



Total Miss Penalty = M + M = 2M

M = Miss Penalty = stall cycles per access resulting from missing in cache