# CS2100 - L13 - Caches (Direct Mapped) Week 8

13.1 - Memory Hierarchy

13.2 - The Principle of Locality

13.3 - The Cache Principle

13.4 - Direct-Mapped Cache

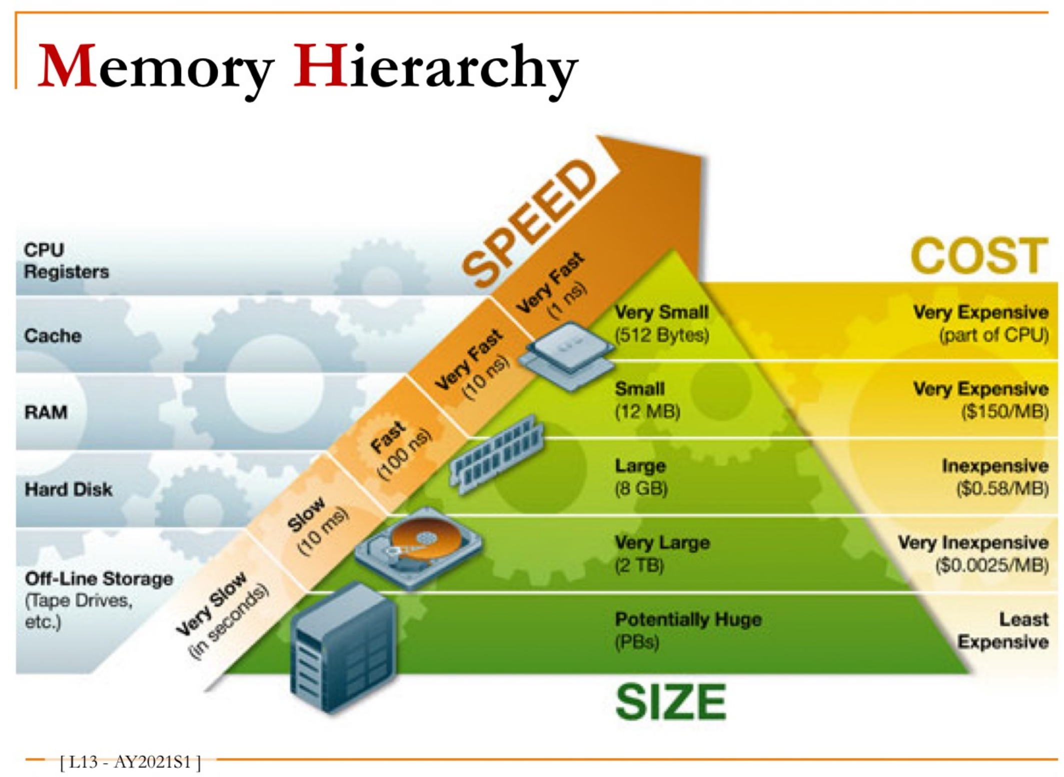
13.5 - Cache Structure

13.6 - Memory Load Instructions

13.7 - Memory Store Instructions

- Write Policy
- Write Miss Policy

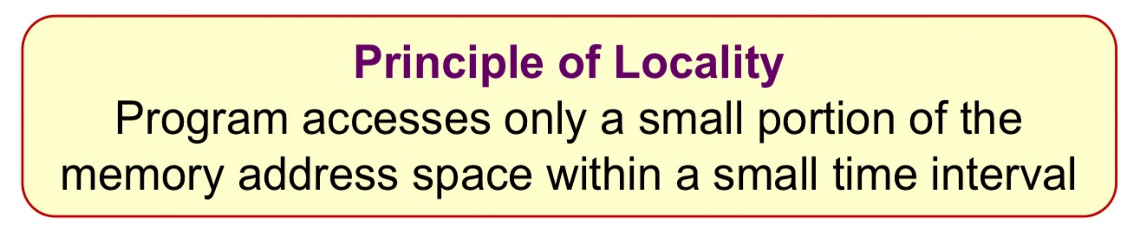
# 13.1 - Memory Hierarchy



## Memory Hierarchy

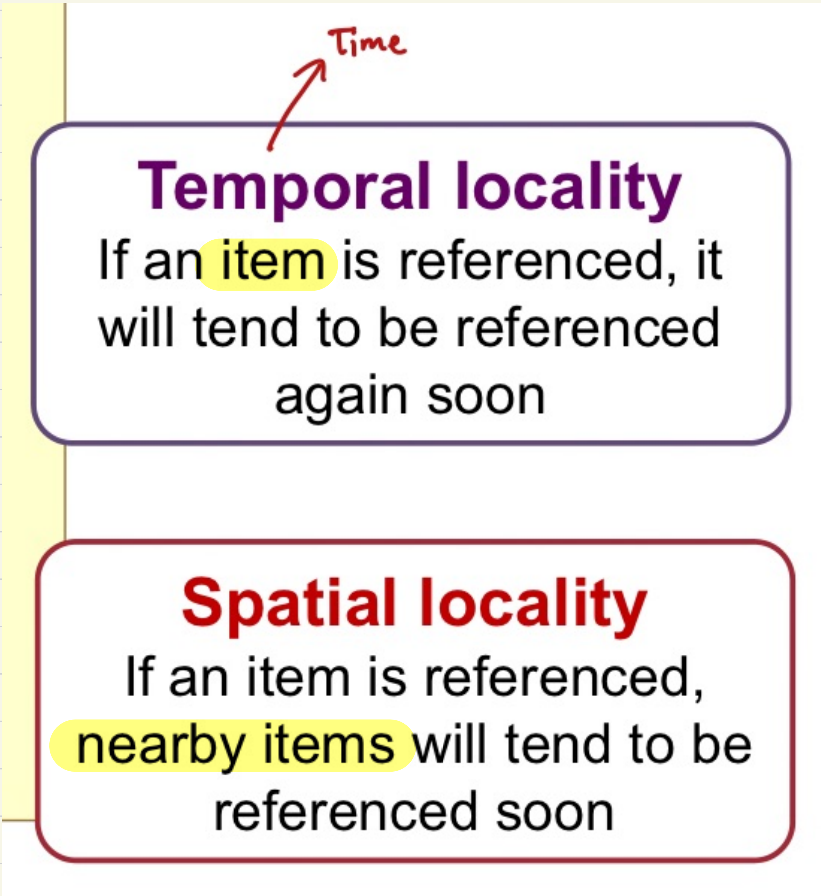
|  | SPEED | SIZE | COST |
|---|---|---|---|
| CPU Registers | Very Fast<br>(1 ns) | Very Small<br>(512 Bytes) | Very Expensive<br>(part of CPU) |
| Cache | Very Fast<br>(10 ns) | Small<br>(12 MB) | Very Expensive<br>($150/MB) |
| RAM | Fast<br>(100 ns) | Large<br>(8 GB) | Inexpensive<br>($0.58/MB) |
| Hard Disk | Slow<br>(10 ms) | Very Large<br>(2 TB) | Very Inexpensive<br>($0.0025/MB) |
| Off-Line Storage<br>(Tape Drives, etc.) | Very Slow<br>(in seconds) | Potentially Huge<br>(PBs) | Least Expensive |

## 13.2 - The Principle of Locality



**Principle of Locality**

Program accesses only a small portion of the memory address space within a small time interval



Time

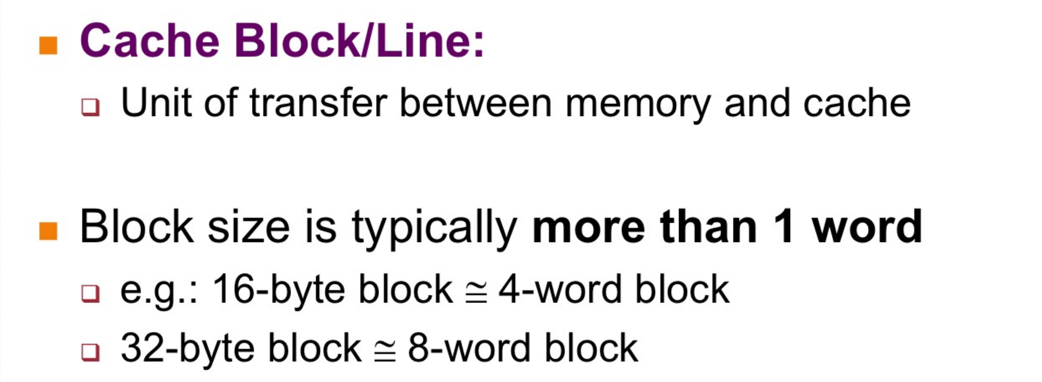
**Temporal locality**

If an item is referenced, it will tend to be referenced again soon

**Spatial locality**

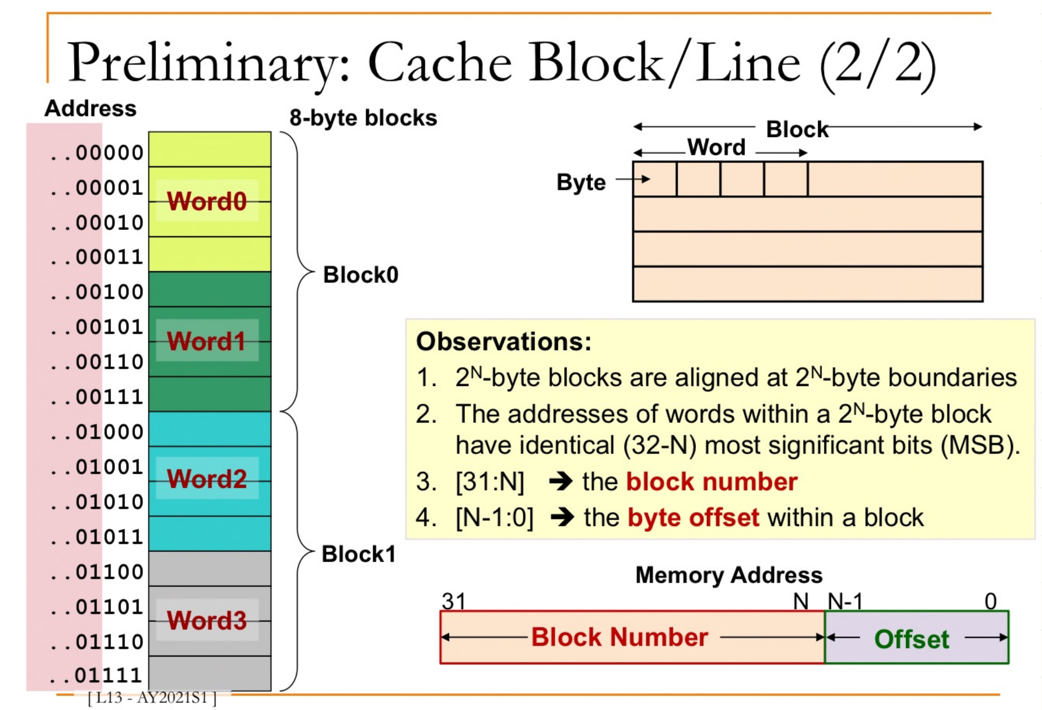
If an item is referenced, nearby items will tend to be referenced soon

## 13.3 - The Cache Principle



- **Cache Block/Line:**
  - Unit of transfer between memory and cache

- Block size is typically **more than 1 word**
  - e.g.: 16-byte block ≅ 4-word block
  - 32-byte block ≅ 8-word block

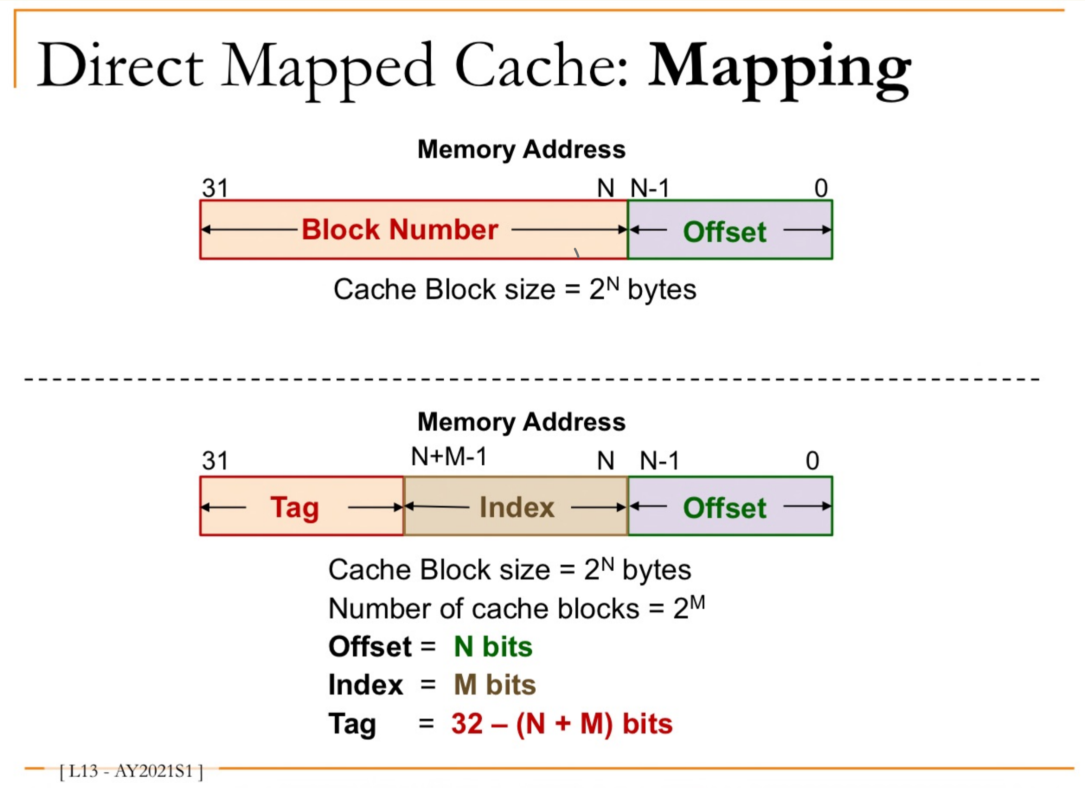


### Preliminary: Cache Block/Line (2/2)

**Observations:**

1. $2^N$-byte blocks are aligned at $2^N$-byte boundaries
2. The addresses of words within a $2^N$-byte block have identical (32-N) most significant bits (MSB).
3. [31:N] ➔ the **block number**
4. [N-1:0] ➔ the **byte offset** within a block

# 13.4 – Direct-Mapped Cache



## Direct Mapped Cache: **Mapping**

**Memory Address**

| 31 | N | N-1 | 0 |
|---|---|---|---|
| ← **Block Number** → | | ← **Offset** → | |

Cache Block size = $2^N$ bytes

---

**Memory Address**

| 31 | N+M-1 | N | N-1 | 0 |
|---|---|---|---|---|
| ← **Tag** → | ← **Index** → | | ← **Offset** → | |

Cache Block size = $2^N$ bytes

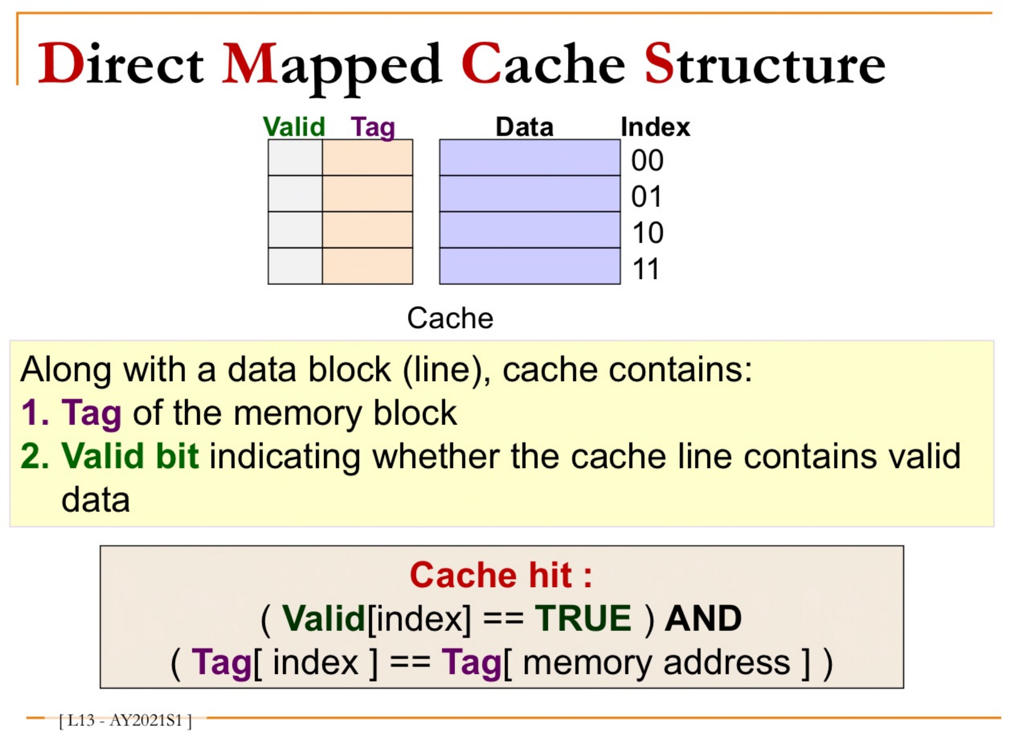
Number of cache blocks = $2^M$

**Offset** = **N bits**

**Index** = **M bits**

**Tag** = **32 – (N + M) bits**

# 13.5 - Cache Structure



## Direct Mapped Cache Structure

| Valid | Tag | Data | Index |
|---|---|---|---|
| | | | 00 |
| | | | 01 |
| | | | 10 |
| | | | 11 |

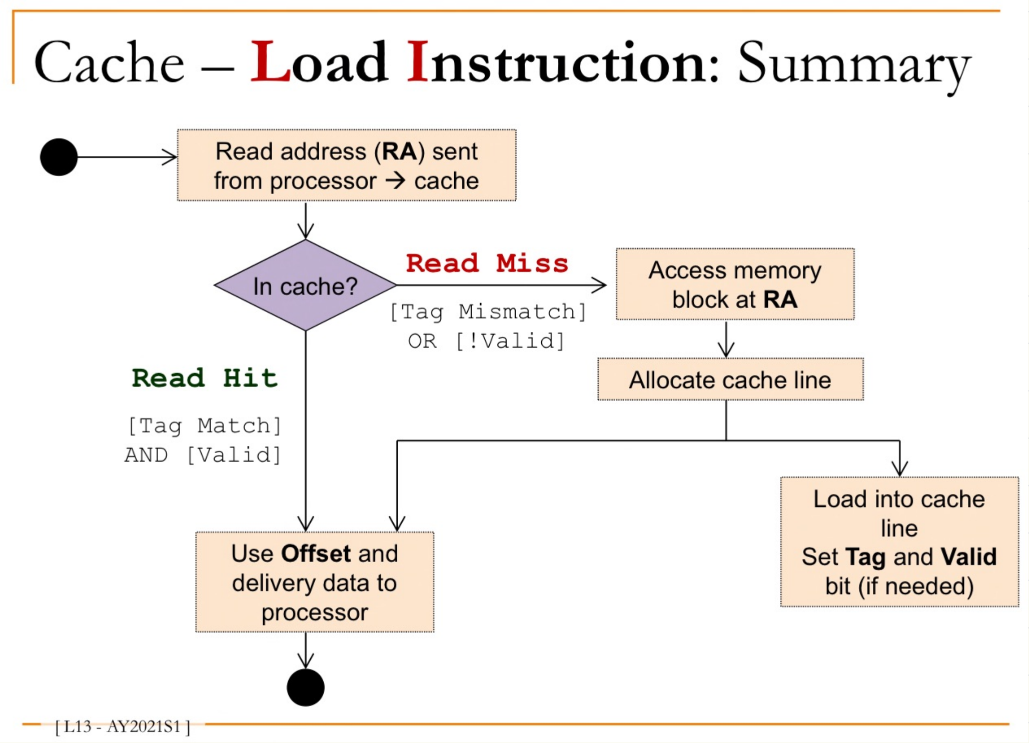
Cache

Along with a data block (line), cache contains:

1. **Tag** of the memory block
2. **Valid bit** indicating whether the cache line contains valid data

**Cache hit :**
( **Valid**[index] == **TRUE** ) **AND**
( **Tag**[ index ] == **Tag**[ memory address ] )

# 13.6 – Memory Load Instructions



## Cache – Load Instruction: Summary

Read address (**RA**) sent from processor → cache

In cache?

**Read Miss**

[Tag Mismatch] OR [!Valid]

Access memory block at **RA**

Allocate cache line

Load into cache line
Set **Tag** and **Valid** bit (if needed)

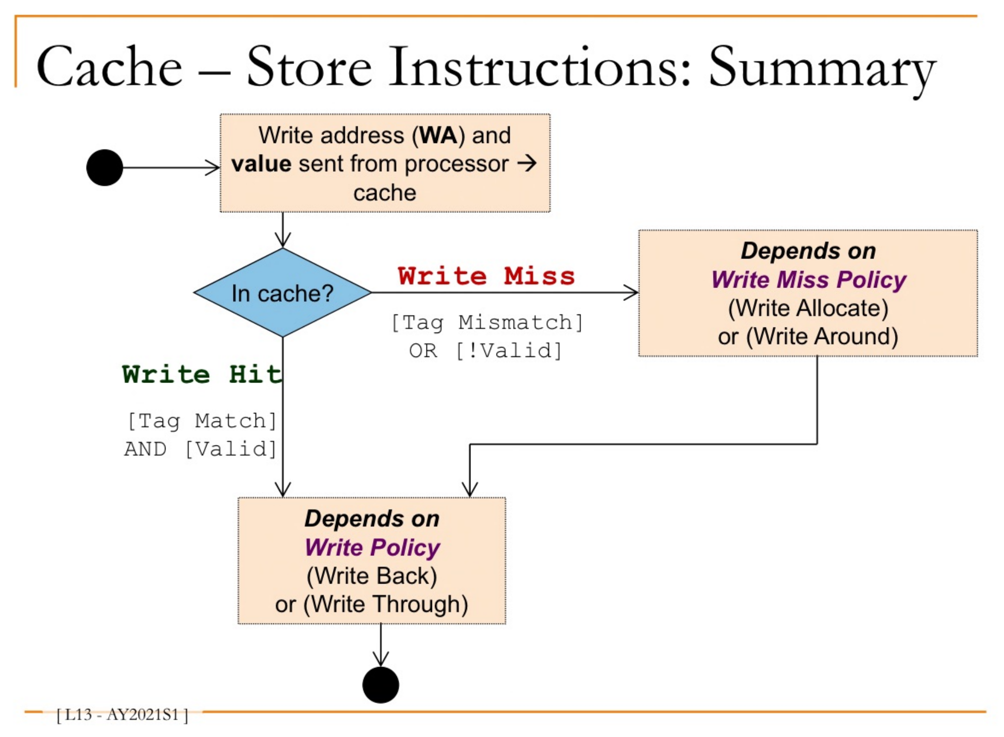
**Read Hit**

[Tag Match] AND [Valid]

Use **Offset** and delivery data to processor

[ L13 - AY2021S1 ]

## 13.7 - Memory Store Instructions



# Cache – Store Instructions: Summary

Write address (**WA**) and **value** sent from processor → cache

In cache?

**Write Miss**

[Tag Mismatch] OR [!Valid]

***Depends on***

***Write Miss Policy***

(Write Allocate) or (Write Around)

**Write Hit**

[Tag Match] AND [Valid]

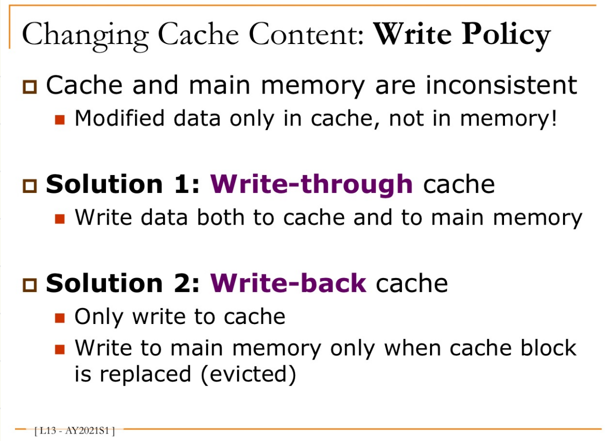
***Depends on***

***Write Policy***

(Write Back) or (Write Through)

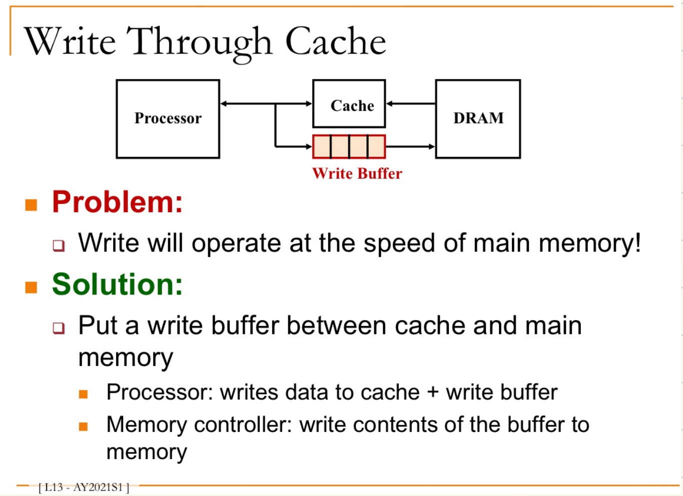
[ L13 - AY2021S1 ]

- Write Policy



## Changing Cache Content: **Write Policy**

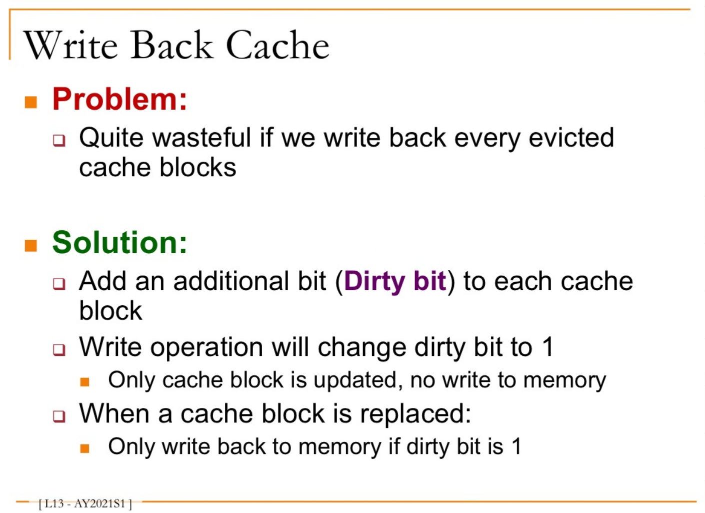
- Cache and main memory are inconsistent
  - Modified data only in cache, not in memory!
- **Solution 1: Write-through** cache
  - Write data both to cache and to main memory
- **Solution 2: Write-back** cache
  - Only write to cache
  - Write to main memory only when cache block is replaced (evicted)



## Write Through Cache

Processor
Cache
DRAM
Write Buffer

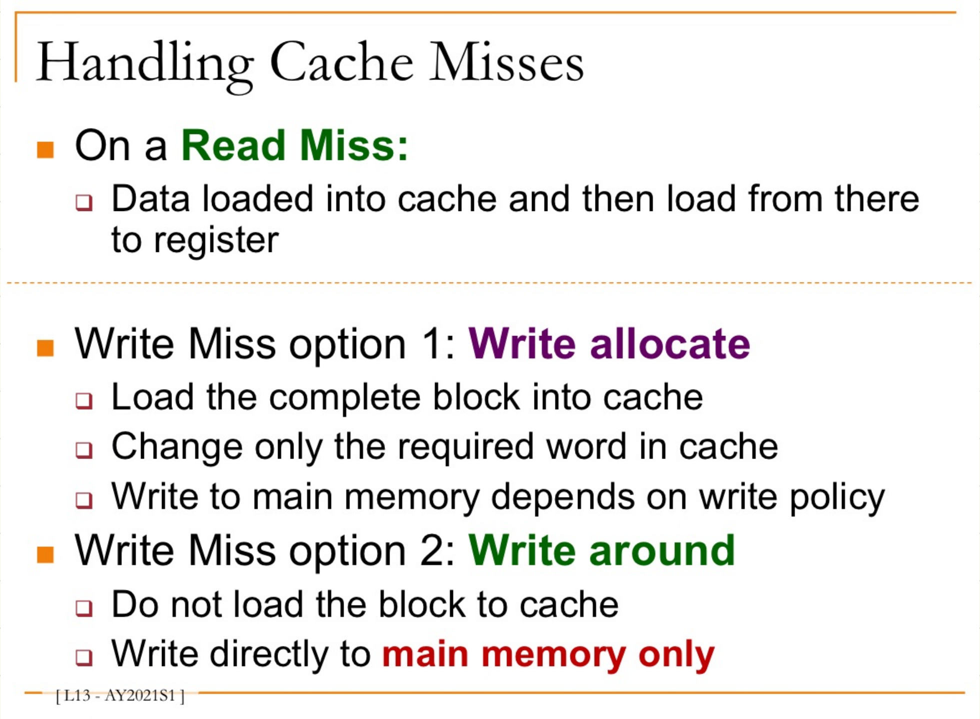
- **Problem:**
  - Write will operate at the speed of main memory!
- **Solution:**
  - Put a write buffer between cache and main memory
    - Processor: writes data to cache + write buffer
    - Memory controller: write contents of the buffer to memory



## Write Back Cache

- **Problem:**
  - Quite wasteful if we write back every evicted cache blocks
- **Solution:**
  - Add an additional bit (**Dirty bit**) to each cache block
  - Write operation will change dirty bit to 1
    - Only cache block is updated, no write to memory
  - When a cache block is replaced:
    - Only write back to memory if dirty bit is 1

- Write Miss Policy



# Handling Cache Misses

- On a **Read Miss:**
  - Data loaded into cache and then load from there to register

- Write Miss option 1: **Write allocate**
  - Load the complete block into cache
  - Change only the required word in cache
  - Write to main memory depends on write policy
- Write Miss option 2: **Write around**
  - Do not load the block to cache
  - Write directly to **main memory only**