# Chapter 5

## Large and Fast: Exploiting Memory Hierarchy

## (Review Sessions)

Computer Architecture and Organization

School of CSEE

HANDONG GLOBAL
UNIVERSITY

# Announcement!

- **Offline week!**
  - **If the last digit of your student ID is even, you should attend the offline lectures on Tuesday, 11/24 and 12/1.**
  - **Otherwise (odd case), you should attend the offline lectures on Friday, 11/27 and 12/4.**
  - **On the day not your turn, you should attend via Zoom.**
  - Example)
    - If the last digit of your student ID is 2: 11/24: offline → 11/27: online → 12/1: offline → 12/4: online
    - If the last digit of your student ID is 7: 11/24: online → 11/27: offline → 12/1: online → 12/4: offline
  - If you cannot attend the offline lectures due to the unavoidable circumstances, then please send an email to me.

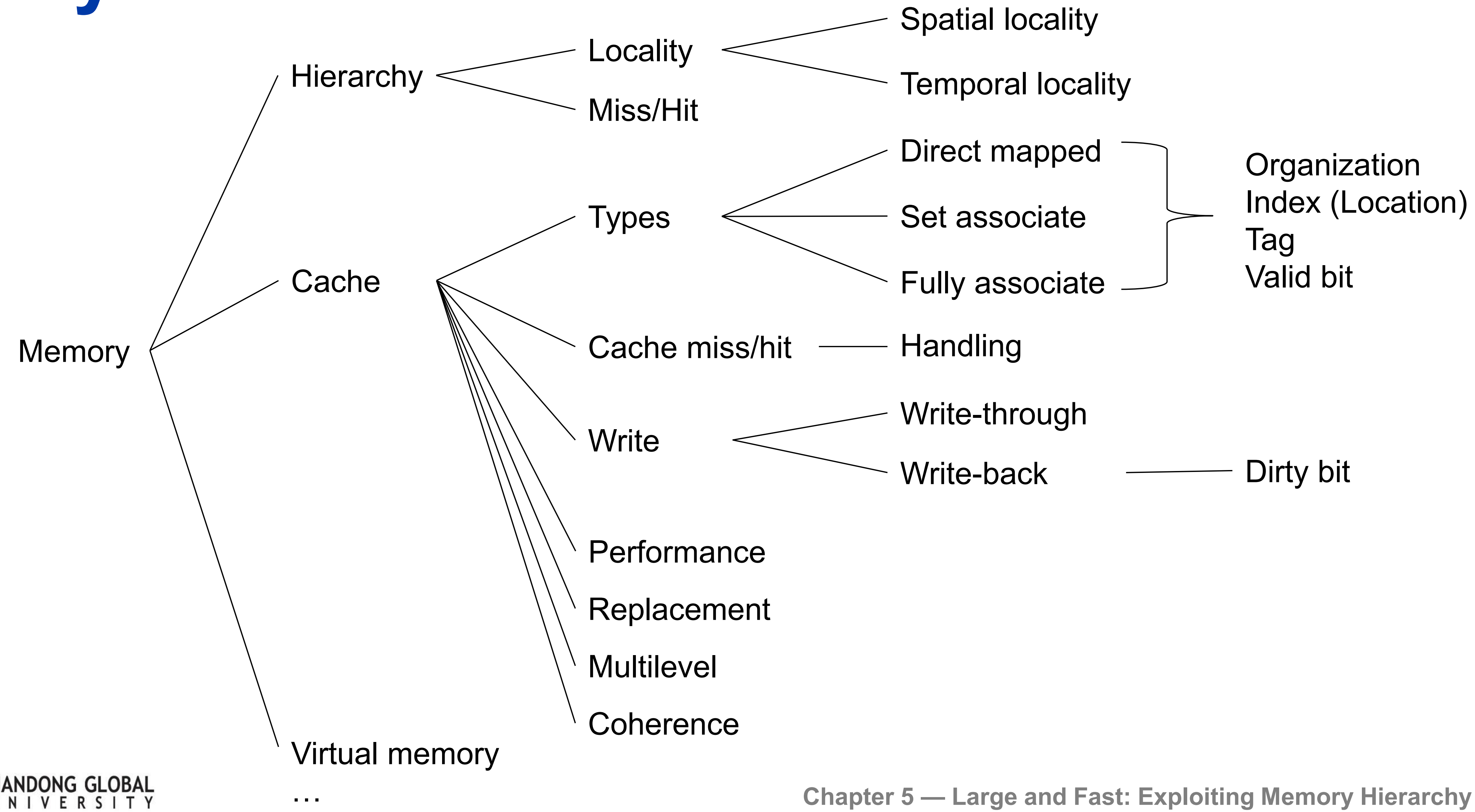
# Announcement!

- **Quiz #2**
  - 12/1 (Tuesday)
  - Scope: Memory (Chapter 4)
  - Open book test
  - LMS (online student) or Printed Paper (Offline student)

# Review (11/27)

## Introduction ~ Basics of Cache

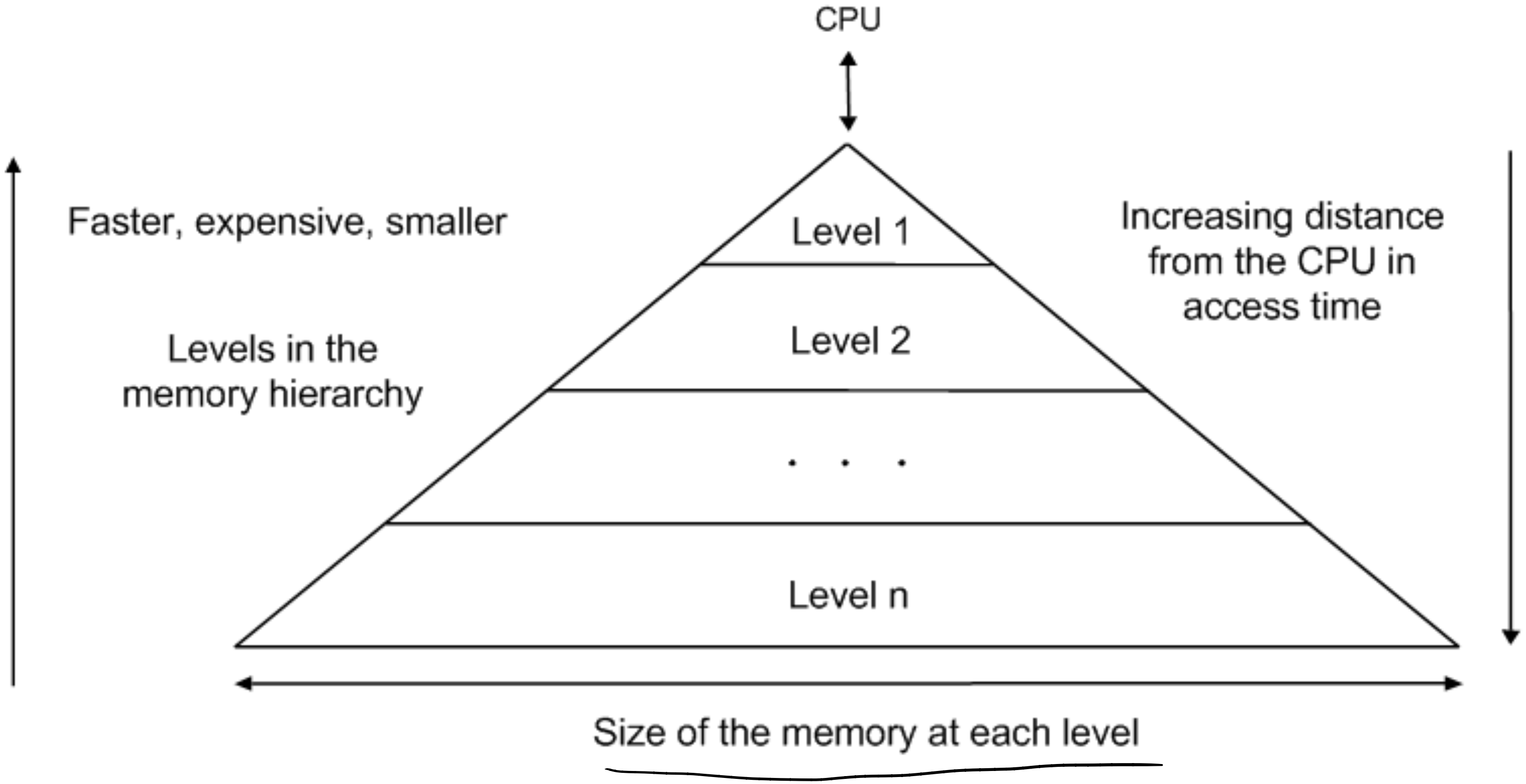
# Keywords



- Memory
  - Hierarchy
    - Locality
      - Spatial locality
      - Temporal locality
    - Miss/Hit
  - Cache
    - Types
      - Direct mapped
      - Set associate
      - Fully associate
        - Organization
        - Index (Location)
        - Tag
        - Valid bit
    - Cache miss/hit
      - Handling
    - Write
      - Write-through
      - Write-back
        - Dirty bit
    - Performance
    - Replacement
    - Multilevel
    - Coherence
  - Virtual memory
  - …

# Exploiting Memory Hierarchy

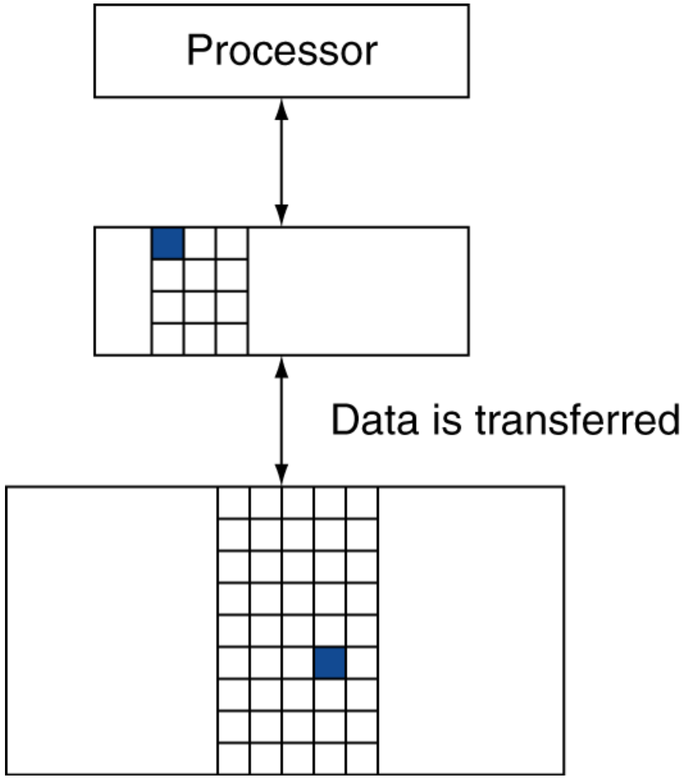
- Solution for ideal memory: Build a memory hierarchy


CPU
Faster, expensive, smaller
Levels in the memory hierarchy
Level 1
Level 2
. . .
Level n
Increasing distance from the CPU in access time
Size of the memory at each level

# Principle of Locality

- Programs access a small proportion of their address space at any time
- Temporal locality
  - Items accessed recently are likely to be accessed again soon
  - e.g., instructions in a loop, induction variables
- Spatial locality
  - Items near those accessed recently are likely to be accessed soon
  - e.g., sequential instruction access, array data

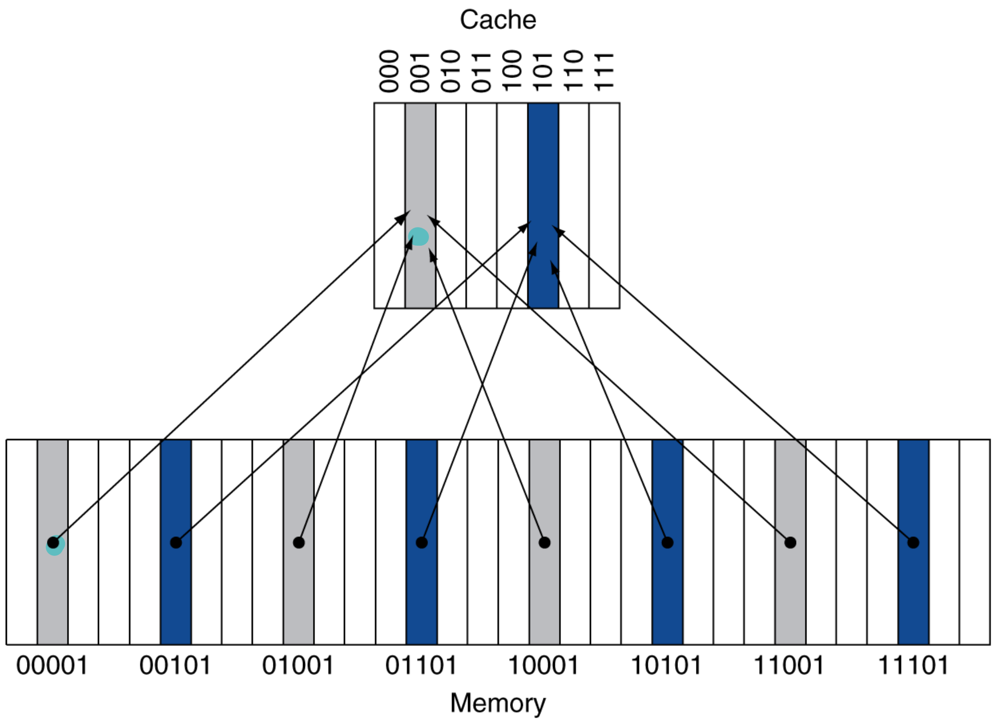
# Memory Hierarchy Levels



- Block (aka line): minimum unit of data
  - May be multiple words
- If accessed data is present in upper level
  - Hit: access satisfied by upper level
    - Hit ratio: hits/accesses
- If accessed data is absent
  - Miss: block copied from lower level
    - Time taken: miss penalty
      - Load block into upper memory → CPU stalls → very time consuming process → performance degrade
    - Miss ratio: misses/accesses
= 1 – hit ratio
  - Then accessed data supplied from upper level

# Direct Mapped Cache

- Location determined by address
- Direct mapped: only one choice
  - Location in the Cache = (Block address) modulo (#Blocks in cache)

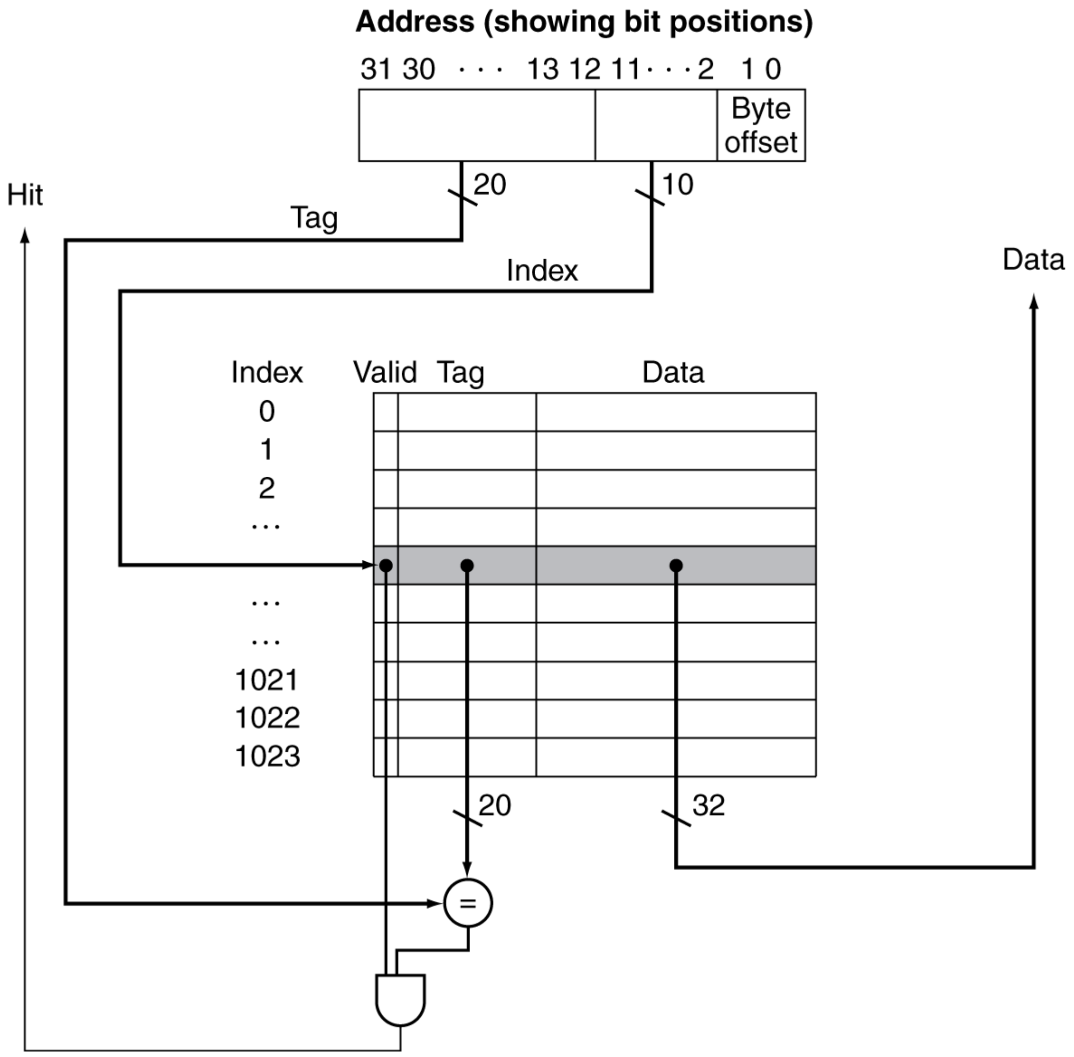


- #Blocks is a power of 2
- Use low-order address bits

# Tags and Valid Bits

- How do we know which particular block is stored in a cache location?
  - Store block address as well as the data
  - Actually, only need the high-order bits
  - Called the tag
- What if there is no data in a location?
  - Valid bit: 1 = present, 0 = not present
  - Initially 0

# Address Subdivision



- Cache has 1024 words
  - 10 bits are used as index (location in the cache)
- Each block is 1 word (4 bytes)
- Tag size is 32 – 10 – 2 = 20 bits
- 1 bit of valid bit
- Total number of bits in cache = 1024 * (32 + 20 + 1)
  - Or $2^n$ * (32 + (32 - n - 2) + 1) for cache with n bits of index and block size of 1 word (4 bytes)

# Cache Misses

- On cache hit, CPU proceeds normally
- On cache miss
  - Stall the CPU pipeline
  - Fetch block from next level of hierarchy
  - Instruction cache miss
    - Restart instruction fetch
  - Data cache miss
    - Complete data access

# Write-Through

- On data-write hit, could just update the block in cache
  - But then cache and memory would be inconsistent
- Write through: also update memory
  - The information is written to both the block in the cache and to the block in the main memory
  - In case of ‘miss’, the old block does not need to be saved
  - Faster processing in case of ‘miss’
  - Whenever data cache is updated, there should a memory write

    → More memory access

# Write-Back

- Write-Back: On data-write hit, just update the block in cache
  - The information is written only to the block in the cache
  - The modified cache block is written to main memory only when it is replaced
  - Keep track of whether each block is dirty → Is block clean or dirty?
  - Dirty: Dirty bit = 1
    - Cache block was updated after it is loaded into cache.
    - The old block should be written into memory when it is replaced.
  - Clean: Dirty Bit = 0
    - Cache block was not updated after it is loaded into cache.

# Questions

- Q1. In this exercise we look at memory locality properties of matrix computation. The following code is written in C, where elements within the same row are stored contiguously. Assume each word is 32-bit integer.

```
for (i = 0; i < 8; i++)
    for (j = 0; j < 8000; j++)
        A[i][j] = B[i][0] + A[j][i];
```

$\log_2 16$

- 1) How many 32-bit integers can be stored in a 16-byte cache block? 4
- 2) References to which variables exhibit temporal locality? i, j
- 3) References to which variables exhibit spatial locality? A[j][j]

# Questions

- Q2. Below is a list of 32-bit memory address references, given as word addresses.
  - 3, 180, 43, 2, 191, 88, 190, 14, 181, 44, 186, 253

- 1) For each of these references, identify the tag and the index given a direct-mapped cache with 16 one-word blocks. Also list if each reference is a hit or a miss, assuming the caches is initially empty.

| Word Address | Binary Address | Tag | Index | Hit/Miss |
|---|---|---|---|---|
| 3 | 0000 0011<br>number=index<br>modulo operation | 0 | 3 | M |
| 180 | 1011 0100 | 11 | 4 | M |
| 43 | 0010 1011 | 2 | 11 | M |
| 2 | 0000 0010 | 0 | 2 | M |
| 191 | 1011 1111 | 11 | 15 | M |
| 88 | 0101 1000 | 5 | 8 | M |
| 190 | 1011 1110 | 11 | 14 | M |
| 14 | 0000 1110 | 0 | 14 | M |
| 181 | 1011 0101 | 11 | 5 | M |
| 44 | 0010 1100 | 2 | 12 | M |
| 186 | 1011 1010 | 11 | 10 | M |
| 253 | 1111 1101 | 15 | 13 | M |

# Questions

- Q2. Below is a list of 32-bit memory address references, given as word addresses.
  - 3, 180, 43, 2, 191, 88, 190, 14, 181, 44, 186, 253

- 2) For each of these references, identify the tag and the index given a direct-mapped cache with two-word blocks and a total size of 8 blocks. Also list if each reference is a hit or a miss, assuming the caches is initially empty.

| Word Address | Binary Address | Tag | Index | Hit/Miss |
|---|---|---|---|---|
| 3 | 0000 0011 | 0 | 1 | M |
| 180 | 1011 0100 | 11 | 2 | M |
| 43 | 0010 1011 | 2 | 5 | M |
| 2 | 0000 0010 | 0 | 1 | H |
| 191 | 1011 1111 | 11 | 7 | M |
| 88 | 0101 1000 | 5 | 4 | M |
| 190 | 1011 1110 | 11 | 7 | H |
| 14 | 0000 1110 | 0 | 7 | M |
| 181 | 1011 0101 | 11 | 2 | H |
| 44 | 0010 1100 | 2 | 6 | M |
| 186 | 1011 1010 | 11 | 5 | M |
| 253 | 1111 1101 | 15 | 6 | M |

# Questions

- Q2. Below is a list of 32-bit memory address references, given as word addresses.
  - 3, 180, 43, 2, 191, 88, 190, 14, 181, 44, 186, 253

- 3) Calculate the total number of bits required for the cache listed above, assuming a 32-bit address.
  - Cache data size: 32KB
  - Cache block size: 2 words

32KB / (2 words * 4 bytes) = 4096 blocks → 12 bits for index
1 bit for word offset, 2 bits for byte offset
Tag bits = 32 bits (total) – 12 bits (index) – 1 bit (word offset) – 2 bits (byte offset) = 17 bits
1 bit for valid bit
17 bits (Tag) + 1 bits(valid) = 18 bits → 18 bits * 4096 blocks = 73728 bits = 9216 bytes

Total cache size = 9216 bytes + 32768 bytes = 41984 bytes