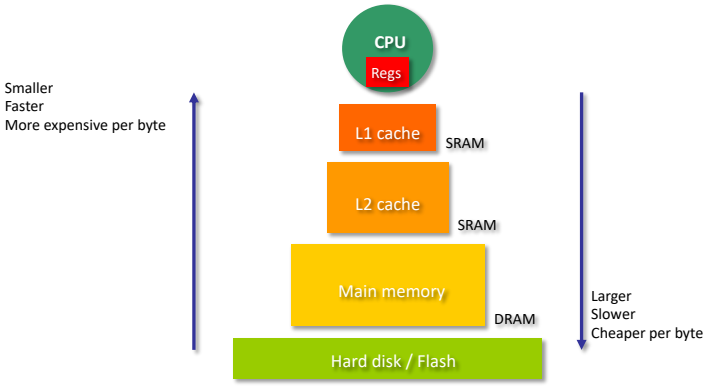
# Review

- Last Class:
  - Multi-threading and SMT
- Today's class:
  - Review of memory hierarchy
- Announcement and reminder

1

# Memory Hierarchy Introduction - §2.1 (summary of Appendix B)




2

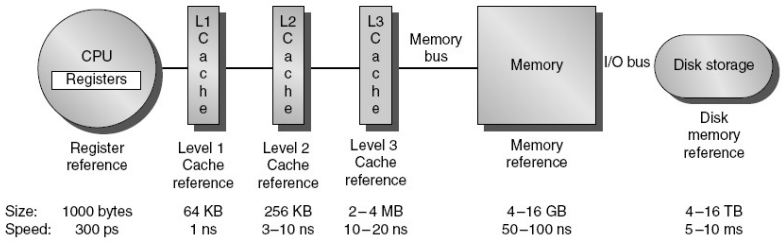
# The Principle of Locality

- Two Different Types of Locality:
  - **Temporal Locality** (Locality in Time): If an item is referenced, it will tend to be referenced again soon.
  - **Spatial Locality** (Locality in Space): If an item is referenced, items whose addresses are close by tend to be referenced soon.

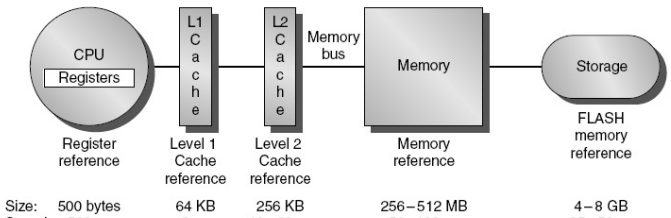
- **Memory** Hierarchy Terminology:
  - block: minimum unit of data transferred between levels (e.g., cacheline)
  - hit: data requested is in the upper level
  - miss: data requested is not in the upper level
  - hit rate: percentage of hits (sometimes called hit ratio)
  - miss rate: percentage of miss (sometimes called miss ratio)
  - hit time: the time to satisfy request in case of a hit
  - miss penalty: the time to satisfy a request in case of a miss

3

# Memory Hierarchy




(a) Memory hierarchy for server




(b) Memory hierarchy for a personal mobile device

4

## Cache performance

Example: Consider a 5-stage (in order) pipeline and assume that

- With perfect cache, CPI = C
- 40% of instructions are load/store (access cache for data)
- Miss penalty = 30 clock cycles
- Miss rate = 2% (for each of the I and D caches)

Number of memory accesses per instruction = 1.4

Actual CPI = C + av. memory stalls per instruction due to cache miss
= C + av. cache misses per instruction * miss penalty
= C + (1.4 * 0.02) * 30 = C + 0.84

NOTE: miss penalty of L1 cache is calculated based on the hit time, miss rate and miss penalty of the L2 cache.

5

## 4 Questions for Memory Hierarchy:

- Q1: Where can a block be placed in the upper level? *(Block placement)*
- Q2: How is a block found if it is in the upper level? *(Block identification)*
- Q3: Which block should be replaced on a miss? *(Block replacement)*
- Q4: What happens on a write? *(Write strategy)*

Note: a block (cache line) is the unit of traffic between memory and cache. A block is usually from 4 to 8 words.

6

# Simplest cache: Direct Mapped

For each item (block) of data in memory, there is exactly one location in the cache where it might be.

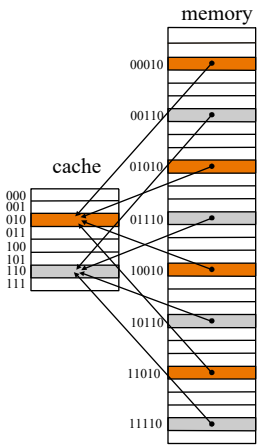
Index = (memory address) **mod** (cache size)

Cache location (block address)

**Note:** lots of items in memory share locations in cache

- Cache can be accessed using the low-order address bits
- Need to *tag* data using the high order address bits
- A valid bit is used for each block

Memory address (byte) = < word address , word offset >
Word address = < block address , block offset >
Block address = < tag , index >




7

# Example

## Cache

Index Valid Tag Data

00
01
10
11

## Main Memory

One word blocks
Two low order bits define the byte in the word (32b words)

0000xx
0001xx
0010xx
0011xx
0100xx
0101xx
0110xx
0111xx
1000xx
1001xx
1010xx
1011xx
1100xx
1101xx
1110xx
1111xx

**Q1: Is it there?**

Compare the cache tag to the high order 2 memory address bits to tell if the memory block is in the cache

**Q2: How do we find it?**

Use next 2 low order memory address bits – the index – to determine which cache block (i.e., modulo the number of blocks in the cache)

**(block address) modulo (cache size)**

8

# Direct Mapped Cache

- Consider the main memory word reference string

Start with an empty cache - all blocks initially marked as not valid

0 1 2 3 4 3 4 15

**0** miss

| | |
|---|---|
| 00 | Mem(0) |
| | |
| | |
| | |

**1** miss

| | |
|---|---|
| 00 | Mem(0) |
| 00 | Mem(1) |
| | |
| | |

**2** miss

| | |
|---|---|
| 00 | Mem(0) |
| 00 | Mem(1) |
| 00 | Mem(2) |
| | |

**3** miss

| | |
|---|---|
| 00 | Mem(0) |
| 00 | Mem(1) |
| 00 | Mem(2) |
| 00 | Mem(3) |

**4** miss

| | |
|---|---|
| ~~00~~ 01 | ~~Mem(0)~~ Mem(4) |
| 00 | Mem(1) |
| 00 | Mem(2) |
| 00 | Mem(3) |

**3** hit

| | |
|---|---|
| 01 | Mem(4) |
| 00 | Mem(1) |
| 00 | Mem(2) |
| 00 | Mem(3) |

**4** hit

| | |
|---|---|
| 01 | Mem(4) |
| 00 | Mem(1) |
| 00 | Mem(2) |
| 00 | Mem(3) |

**15** miss

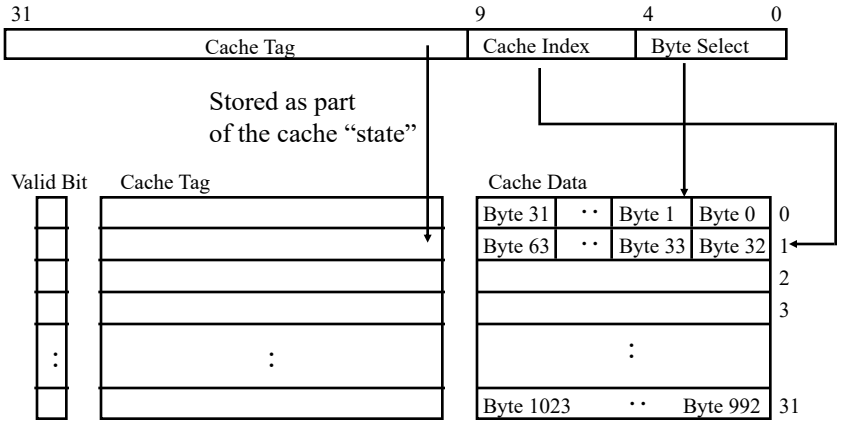
| | |
|---|---|
| 01 | Mem(4) |
| 00 | Mem(1) |
| 00 | Mem(2) |
| ~~00~~ 11 | ~~Mem(3)~~ Mem(15) |

- 8 requests, 6 misses

10

# 1 KB Direct Mapped Cache, 32B blocks

- For a $2^N = 2^{10}$ byte cache and 32-bit memory address:
  - The upper most $(32 - N) = 22$ bits form the Cache Tag
  - The lowest $M = 5$ bits are the Byte Select (Block Size = $2^M$)



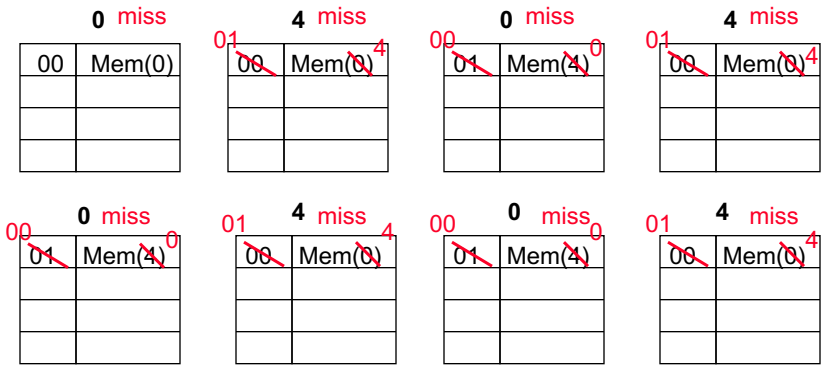

11

## Another Reference String Mapping

- Consider the main memory word reference string

Start with an empty cache - all blocks initially marked as not valid

0 4 0 4 0 4 0 4

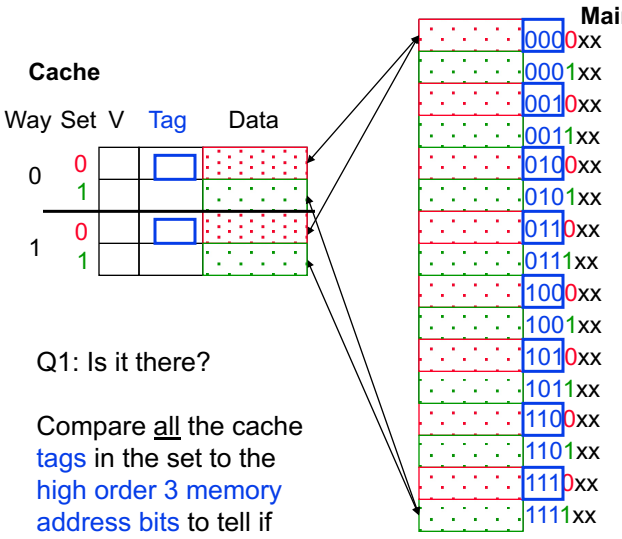



- 8 requests, 8 misses

- Ping pong effect due to conflict misses - two memory locations that map into the same cache block

12

## Set Associative Cache Example




One word blocks
Two low order bits define the byte in the word (32b words)

Q1: Is it there?

Compare all the cache tags in the set to the high order 3 memory address bits to tell if the memory block is in the cache

Q2: How do we find it?

Use next 1 low order memory address bit to determine which cache set (i.e., modulo the number of sets in the cache)

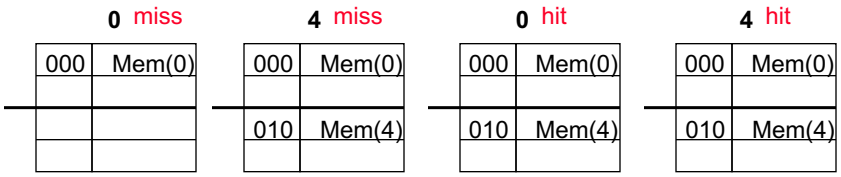
14

## Another Reference String Mapping

- Consider the main memory word reference string

Start with an empty cache - all blocks initially marked as not valid

0 4 0 4 0 4 0 4



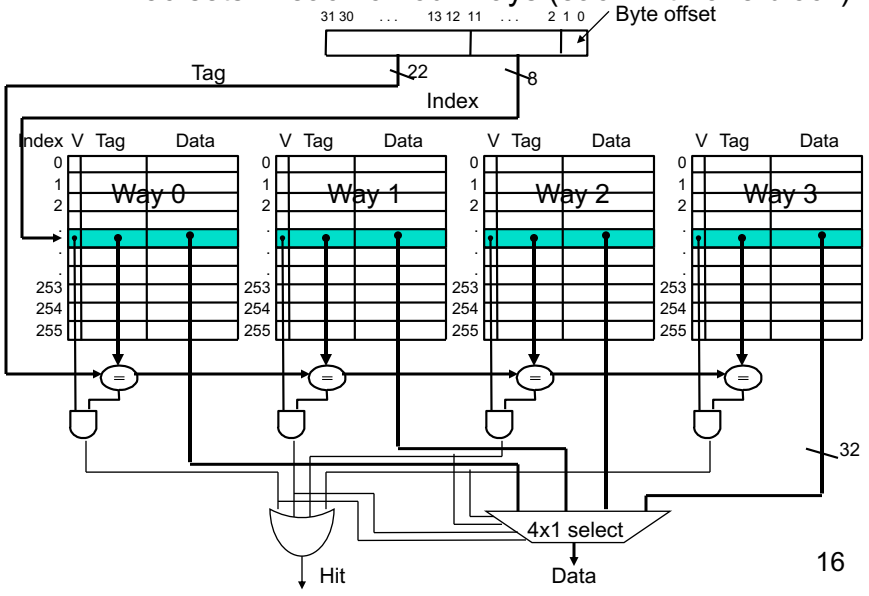

- 8 requests, 2 misses

- Solves the ping pong effect in a direct mapped cache due to conflict misses since now two memory locations that map into the same cache set can co-exist!

15

## Four-Way Set Associative 4KB Cache

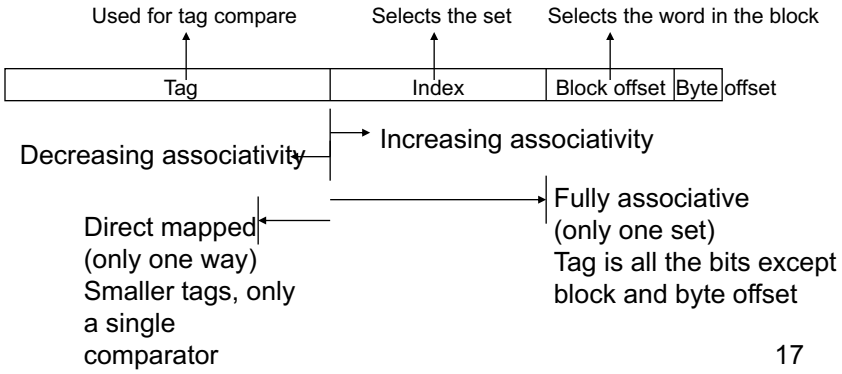
- $2^8$ = 256 sets in each of four ways (each with one block)




16

## Range of Set Associative Caches

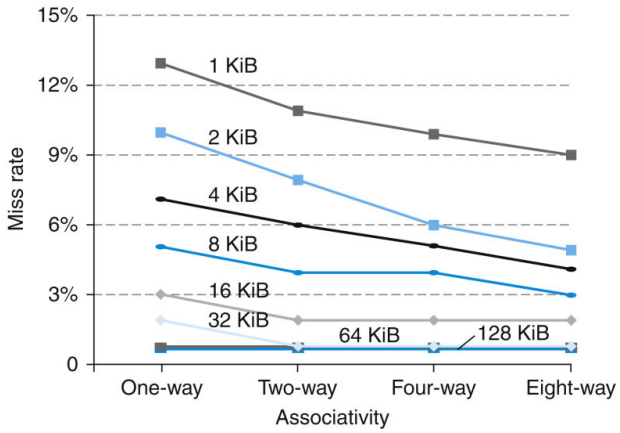
- For a fixed size cache, each increase by a factor of two in associativity doubles the number of blocks per set (i.e., the number or ways) and halves the number of sets – decreases the size of the index by 1 bit and increases the size of the tag by 1 bit




17

## Benefits of Set Associative Caches

- The choice of direct mapped or set associative depends on the cost of a miss versus the cost of implementation




- Largest gains are in going from direct mapped to 2-way (20%+ reduction in miss rate)

18

## Costs of Set Associative Caches

- When a miss occurs, which way's block do we pick for replacement?
  - **Least Recently Used (LRU):** the block replaced is the one that has been unused for the longest time
    - Must have hardware to keep track of when each way's block was used relative to the other blocks in the set
    - For 2-way set associative, takes one bit per way → set the bit when a block is referenced (and reset the other way's bit)
- N-way set associative cache additional costs
  - N comparators (delay and area) – one per way
  - MUX delay (way selection) before data is available
  - Data available after way selection (and Hit/Miss decision). In a direct mapped cache, the cache block is available before the Hit/Miss decision
    - So its not possible to just assume a hit and continue and recover later if it was a miss

19

## Disadvantage of Set Associative Cache:

- N-way Set Associative Cache vs. Direct Mapped Cache:
  - N comparators vs. 1
  - Extra MUX delay for the data
  - Data comes AFTER Hit/Miss
- In a direct mapped cache, Cache Block is available BEFORE Hit/Miss:
  - Possible to assume a hit and continue. Recover later if miss.
- A fully associative cache is one with N=size of the cache (slow ???)

## Block replacement:

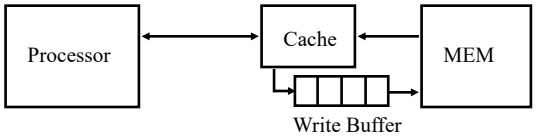
- Not an issue for Direct Mapped
- Set Associative or Fully Associative:
  - Random
  - LRU (Least Recently Used)
  - FIFO
  - Belady (ideal but not practical)
  - Others? (ML-based approach)

Designing a Cost-Effective Cache Replacement Policy using Machine Learning
Subhash Sethumurugan, Jieming Yin, John Sartori
27th IEEE International Symposium on High-Performance Computer Architecture (HPCA), 2021.

20

## Write policies

- Write through or write back,
- Write back can take advantage of a dirty bit
- A Write Buffer (FIFO) is needed between the Cache and Memory
  - Typical number of entries: 4
  - Write buffer saturation??

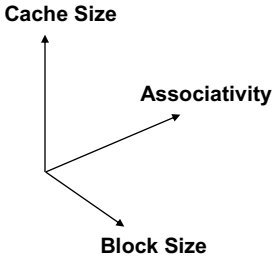



## Two options on a write miss :

- Write allocate: read block first to cache (as in a read miss), then write the word
- No-write allocate: write only to memory and not to cache

21

- Several interacting dimensions
  - cache size
  - block size
  - associativity
  - replacement policy
  - write-through vs write-back
  - write allocate
- The optimal choice is a compromise
  - depends on access characteristics
    - » workload
    - » use (I-cache, D-cache, TLB)
  - depends on technology / cost
- Simplicity often wins




22

## Average memory access time

Example: Assume that

- Processor speed = 1 GHz (1 n.sec. clock cycle)
- Cache access time = 1 clock cycle
- Miss penalty = 100 n.sec (100 clock cycles)
- I-cache miss rate = 1%, and D-cache miss rate = 3%
- 74% of memory references are for instructions and 26% for data

Effective cache miss rate = 0.01 * 0.74 + 0.03 * 0.26 = 0.0152
Av. (effective) memory access time = 1 + 0.0152 * 100 = 2.52 cycles
= 2.52 n.sec

- Now, assume that you increase the processor speed to 1.5 GHz

Miss penalty = 150 cycles (0.66 n.sec cycles)
Av. (effective) memory access time = 1 + 0.0152 * 150 = 3.28 cycles
= 2.18 n.sec

23

## Cache Performance (§B.2)

*CPUtime = IC x ($CPI_{execution}$ + Misses per instruction x Miss penalty) x Clock cycle time*

*IC* = Instruction Count

Misses per instruction = Memory accesses per instruction x Miss rate

$$CPUtime = IC \times \left( CPI_{Execution} + \frac{Memory\ \ accesses}{Instruction} \times \textbf{Miss rate} \times Miss\ \ penalty \right) \times Clock\ \ cycle\ \ time$$

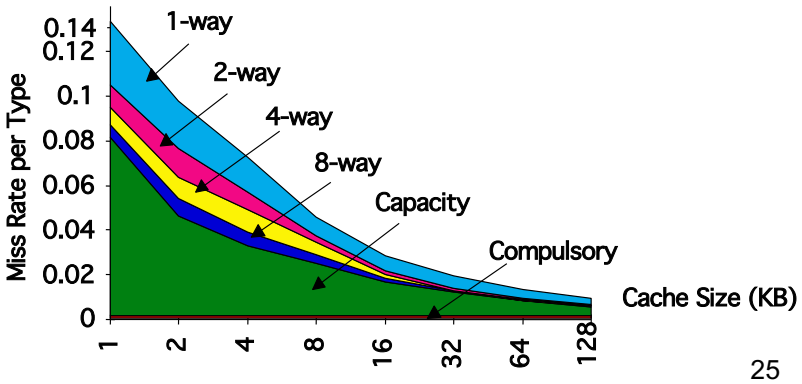
## Improving Cache Performance:

1. Reduce the miss rate,
2. Reduce the miss penalty,
3. Reduce the time to hit in the cache.

24

# Classification of cache misses

- Compulsory Misses: Sad facts of life. Example: cold start misses.
- Capacity Misses: Increase cache size
- Conflict Misses: Increase cache size and/or associativity.
  Nightmare Scenario: ping pong effect!




25