CE/CZ 3001: Advanced Computer Architecture

Module 5: Memory Systems

- review

Asst Prof Liu Weichen
School of Computer Science and Engineering
Nanyang Technological University, Singapore

Summary: Cache System

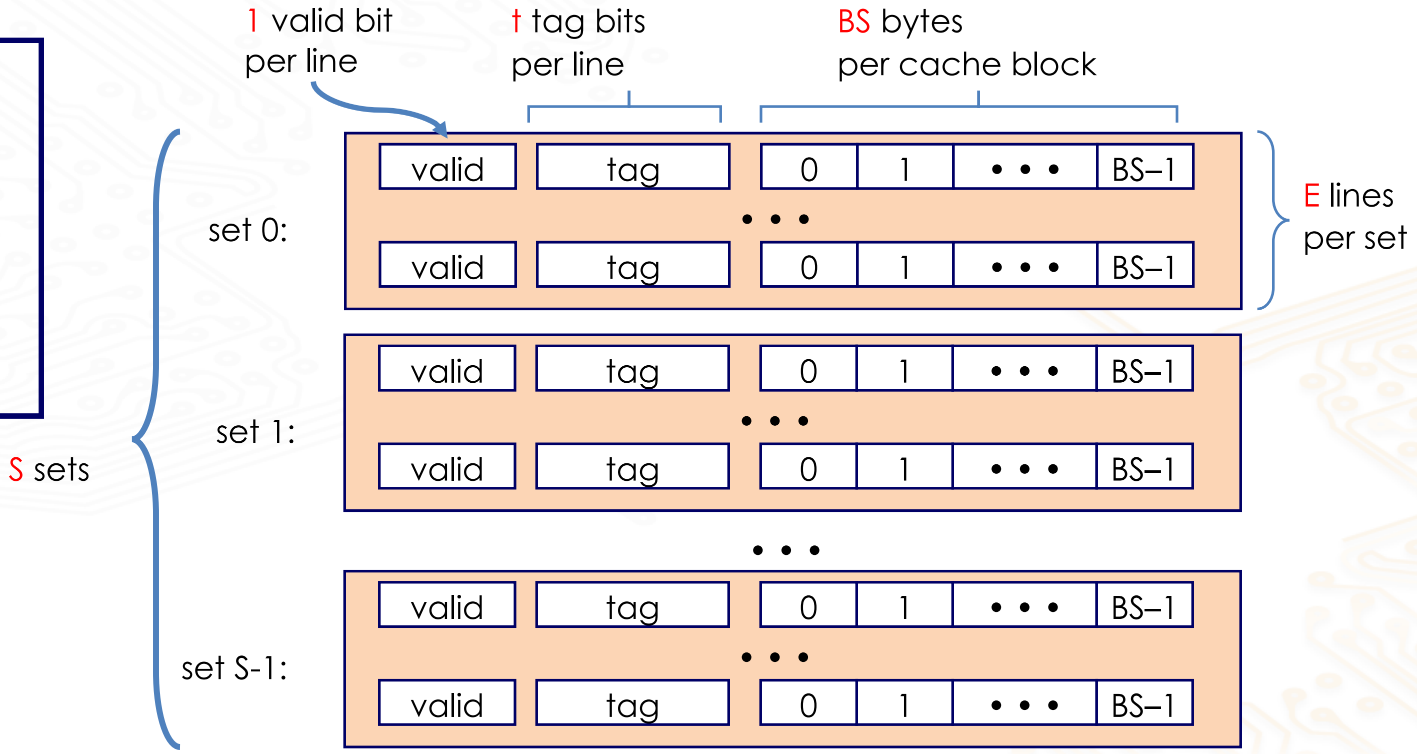
- Cache design
  - Organization schemes: direct mapped, set associative, fully associative
  - Write policies: write through, write back
  - Replacement policies: random, FIFO, LRU, etc.
  - Cache miss reasons and optimization
- Cache performance
  - Cache/memory address and size related calculations
  - CPI, Average Memory Access Time (AMAT)

General Organization of a Cache (Part 1/3)

Cache is an array of sets

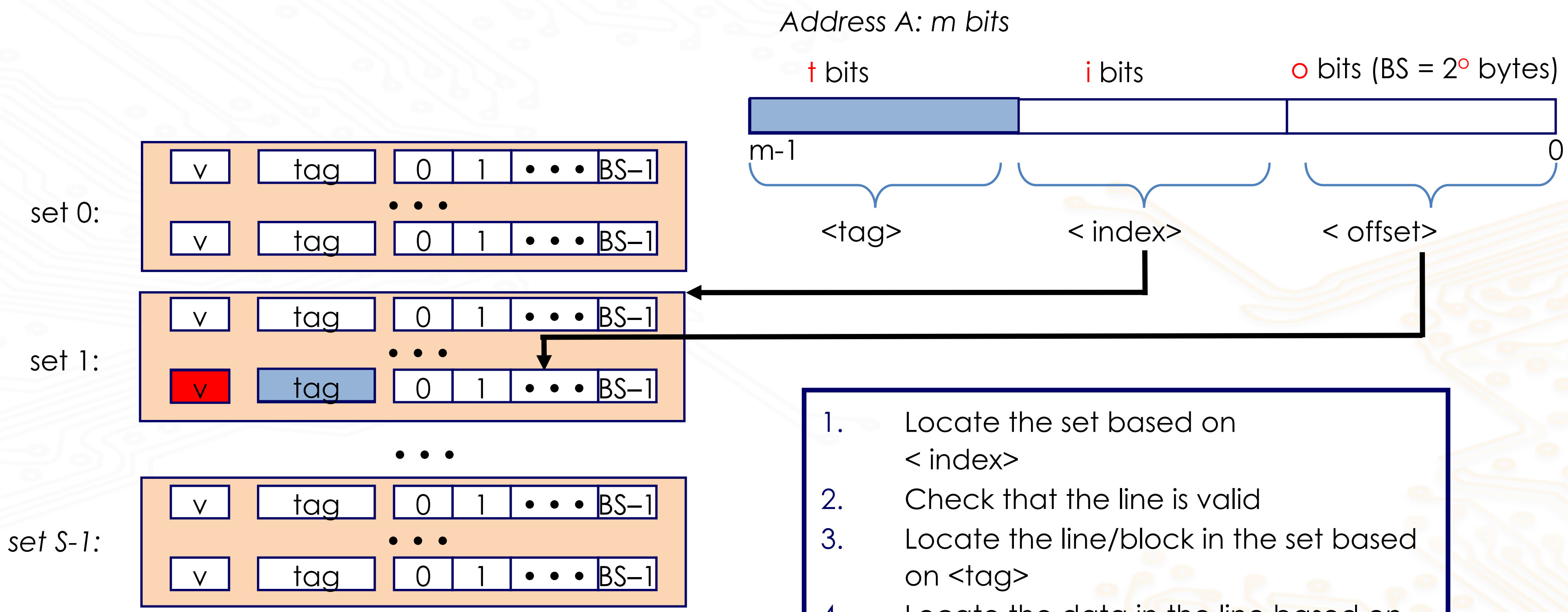
Each set contains one or more lines

Each line holds a block of data



Cache size: $C = (1\text{ bit} + t\text{ bits} + BS\text{ bytes}) \times E \times S$

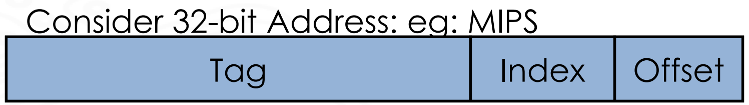
General Organization of a Cache (Part 2/3)



1. Locate the set based on < index>
2. Check that the line is valid
3. Locate the line/block in the set based on <tag>
4. Locate the data in the line based on < offset>

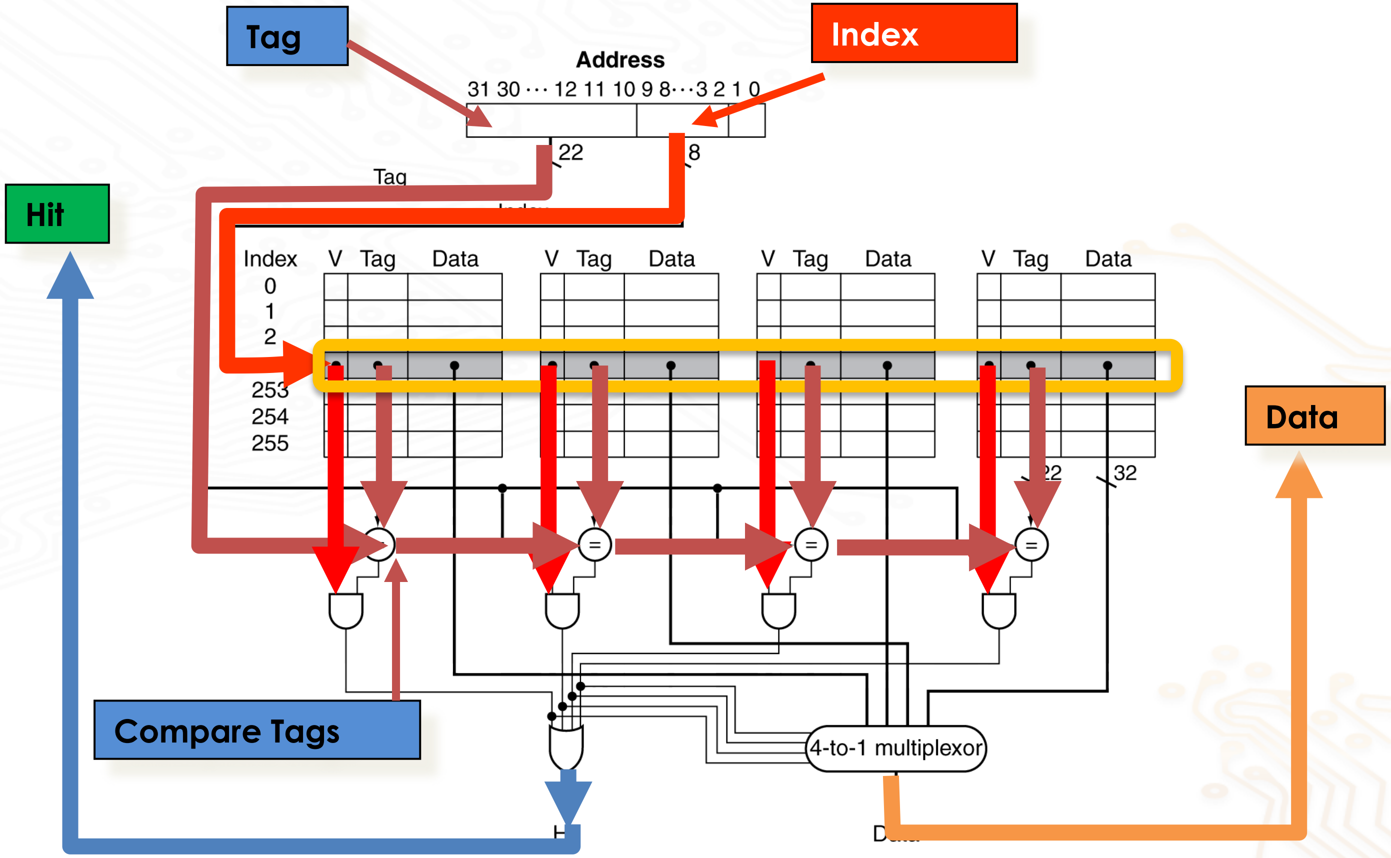
General Organization of a Cache (Part 3/3)

- How to determine the tag, index and offset bits in main memory address?
- Given
  - address size of main memory
  - block size (BS), number of sets(S) and number of lines(E)



Portion	Length	Purpose
Offset	o=log$_2$(block size)	Select word within block/line
Index	i=log$_2$(number of sets)	Select the set
Tag	t=32 - o - i	ID block/line within set

Set associative cache: MIPS Example



Replacement Policy

- How do we choose *victim cache line*?
  - Verbs: *Victimize, evict, replace, cast out*

- Several policies are possible
  - FIFO (first-in-first-out)
  - LRU (least recently used)
    - Takes care of temporal locality
    - Needs to keep history of access and hence slows down the system
  - NMRU (not most recently used)
  - Pseudo-random (yes, really!)

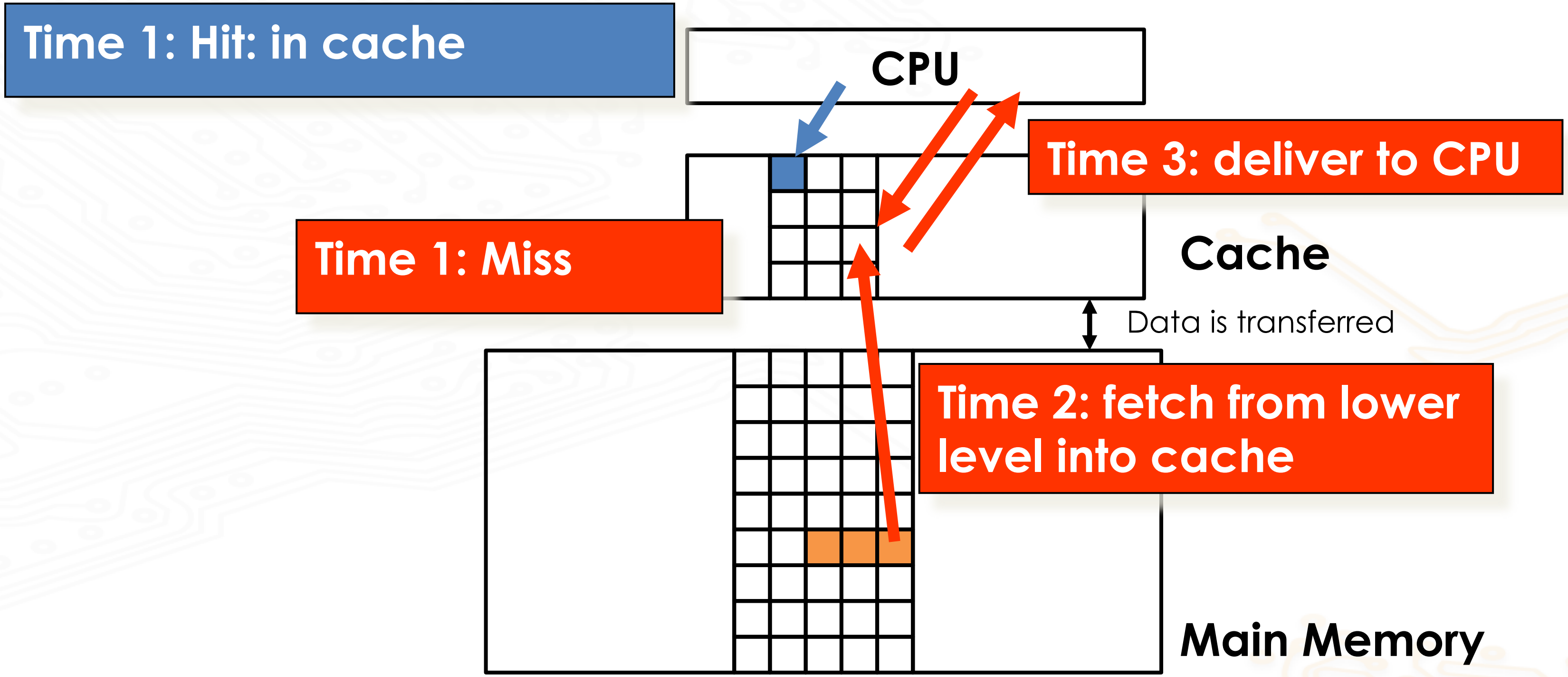
- Pick victim within *set* where a = *associativity*
  - If a <= 2, LRU is cheap and easy (1 bit)
  - If a > 2, it gets harder
  - Pseudo-random works pretty well for caches

Write Policies (Part 2/2)

- Easiest policy: *write-through*
- Every write propagates directly through hierarchy
  - Write in L1, L2, memory, disk
- Why is this a bad idea?
  - Very high bandwidth requirement
  - Memory becomes slow when its size increases
- Popular in real systems only for writing to the L2
  - Every write updates L1 and L2
  - Beyond L2, use *write-back policy*

- Most widely used : *write-back*
- Maintain *state* of each line in a cache
  - Invalid – not present in the cache
  - Clean – present, but not written (unmodified)
  - Dirty – present and written (modified)
- Store state in tag array, next to address tag
  - Mark dirty bit on a write
- On eviction, check dirty bit
  - If set, write back dirty line to next level
  - Called a *writeback* or *castout*

Cache Example



Hit time = Time 1

Miss penalty = Time 2 + Time 3

Cache Miss (Part 2/2)

- Reasons for cache miss
  - Compulsory miss
    - First-ever reference to a given block of memory
  - Capacity miss
    - Working set exceeds cache capacity
    - Useful blocks (with future references) displaced
  - Conflict miss
    - Placement restrictions (not fully-assoc.) cause useful blocks to be displaced
    - Think of as *capacity within set*

Cache Miss and Performance (Part 2/2)

- How does this affect performance?
- Execution Time

$$= \underset{\text{(IC)}}{\boxed{\text{Instruction count}}} \times \underset{\text{(CPI)}}{\boxed{\frac{\text{Cycles}}{\text{Instruction}}}} \times \underset{\text{(cycle time)}}{\boxed{\frac{\text{Time}}{\text{Cycle}}}}$$

- Assuming cache hit costs are included as part of the normal CPU execution cycle, then

$$\text{CPU time} = \text{IC} \times \text{CPI} \times \text{cycle time}$$
$$= \text{IC} \times \underbrace{(\text{CPI}_{\text{ideal}} + \text{Memory-stall cycles})}_{\text{CPI}_{\text{stall}}} \times \text{cycle time}$$

- Memory-stall cycles = memory accesses/program × miss rate × miss penalty

Average memory access time (AMAT) (Part 1/2)

- Average Memory Access Time (AMAT) is the average to access memory considering both hits and misses

$$\text{AMAT} = \text{Time for a hit} + \text{Miss rate} \times \text{Miss penalty}$$