Lec 14: Caches

Kavita Bala
CS 3410, Fall 2008
Computer Science
Cornell University

Announcements

- Prelim
  - Oct 23rd, 7:30-10:00
  - All content up to Lecture on Oct 16th
- Ask us questions if in doubt

Performance

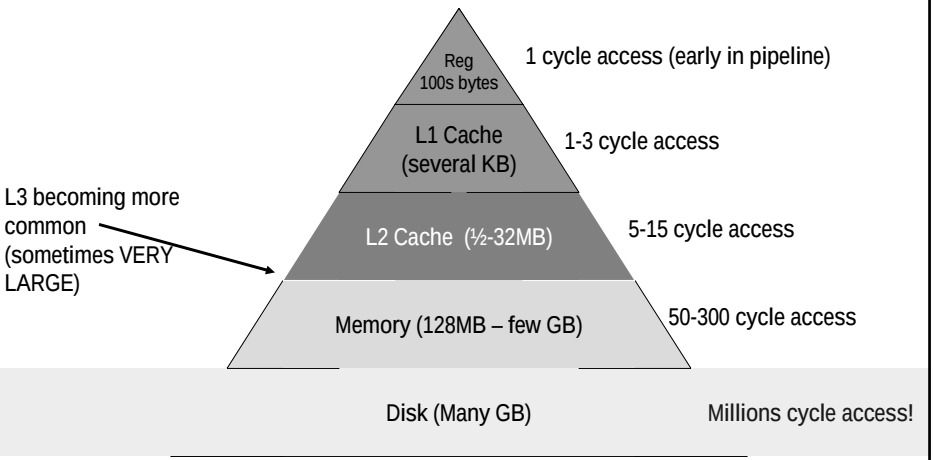
- CPU clock rates ~0.2ns-2ns (5GHz-500MHz)

Technology	Capacity	Cost/GB	Latency
Tape	1 TB	$.17	100s
Disk	300 GB	$.34	Millions cycles (ms)
DRAM	4GB	$100s	50-300 cycles (10s of ns)
SRAM off	512KB	$4-10'sk	5-15 cycles (few ns)
SRAM on	16 KB	???	1-3 cycles (ns)

- Capacity and latency are closely coupled
- Cost is inversely proportional

- How do we create the illusion of large and fast memory?

Cache Design 101



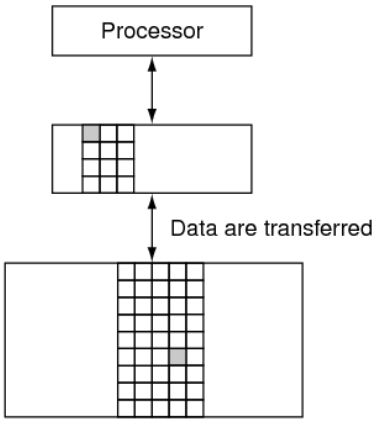
These are rough numbers: mileage may vary for latest/greatest
Caches USUALLY made of SRAM

Insight of Caches

- Exploit locality
  - Two types: temporal and spatial
- Temporal locality
  - If memory location X is accessed, then it is more likely to be accessed again in the near future than some random location Y
  - Caches exploit temporal locality by placing a memory element that has been referenced into the cache
- Spatial locality
  - If memory location X is accessed, then locations near X are more likely to be accessed in the near future than some random location Y
  - Caches exploit spatial locality by allocating a cache line of data (including data near the referenced location)

Memory Hierarchy

- Closer to processor
  - Subset of memory farther from processor
  - Faster and smaller
  - Transfer an entire block



Cache Lookups (Read)

- Look at address issued by processor
- Search cache to see if that block is in the cache
  - Hit: Block is in the cache
    - return requested data
  - Miss: Block is not in the cache
    - read line from memory
    - evict an existing line from the cache
    - place new line in cache
    - return requested data

Cache Organization

- Cache has to be fast and small
  - Gain speed by performing lookups in parallel, requires die real estate
  - Reduce hardware required by limiting where in the cache a block might be placed

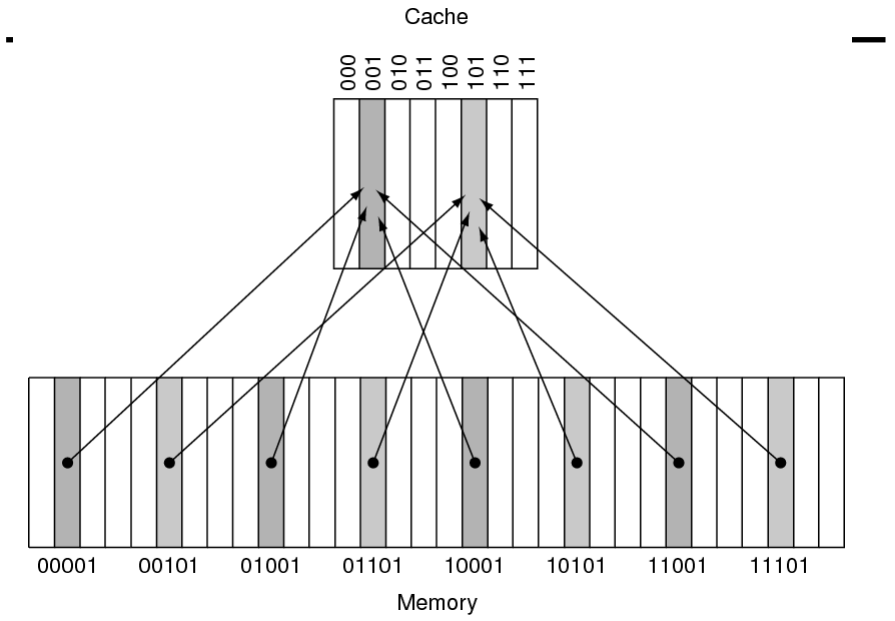
Cache Organization

- Three common designs
  - Fully associative: Block can be anywhere in the cache
  - Direct mapped: Block can only be in one line in the cache
  - Set-associative: Block can be in a few (2 to 8) places in the cache

Direct Mapped Cache

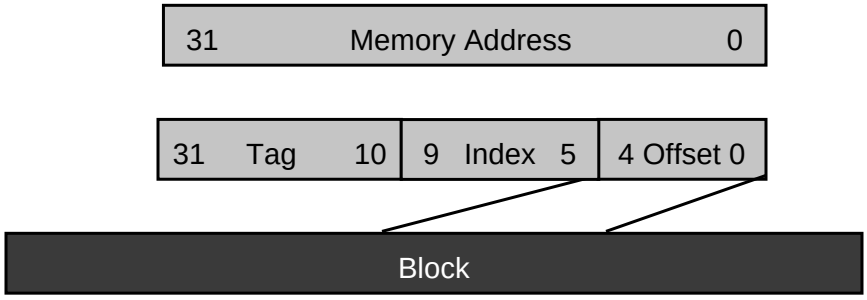
- Simplest
- Block can only be in one line in the cache
- Like an array

- How to determine this location?
  - Use modulo arithmetic
  - (address) modulo (# cache blocks)



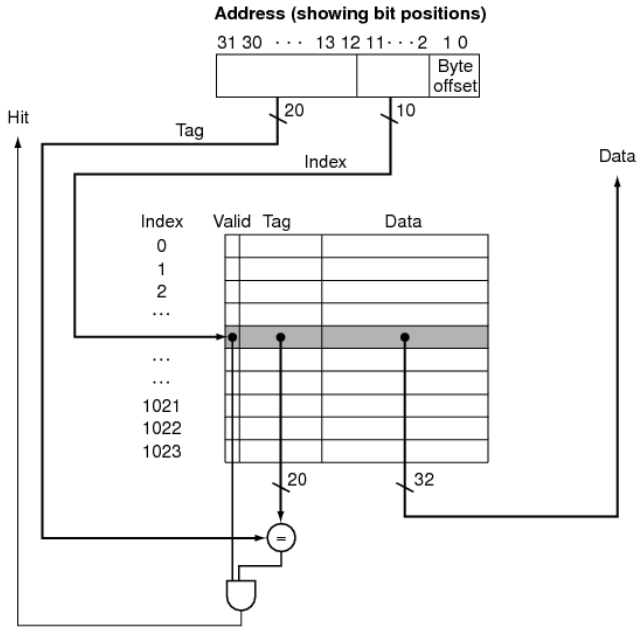
Tags and Offsets

- Offset: within block
- Valid bit: is the data valid?
- Tag: matching



Valid Bits

- Valid bits indicate whether cache line contains an up-to-date copy of the values in memory
  - Must be 1 for a hit
  - Reset to 0 on power up
- An item can be removed from the cache by setting its valid bit to 0

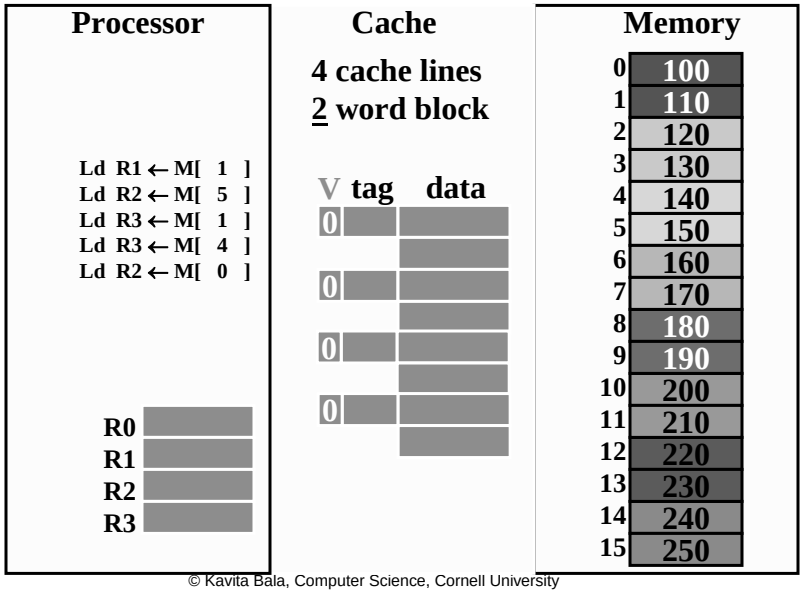


Cache Size

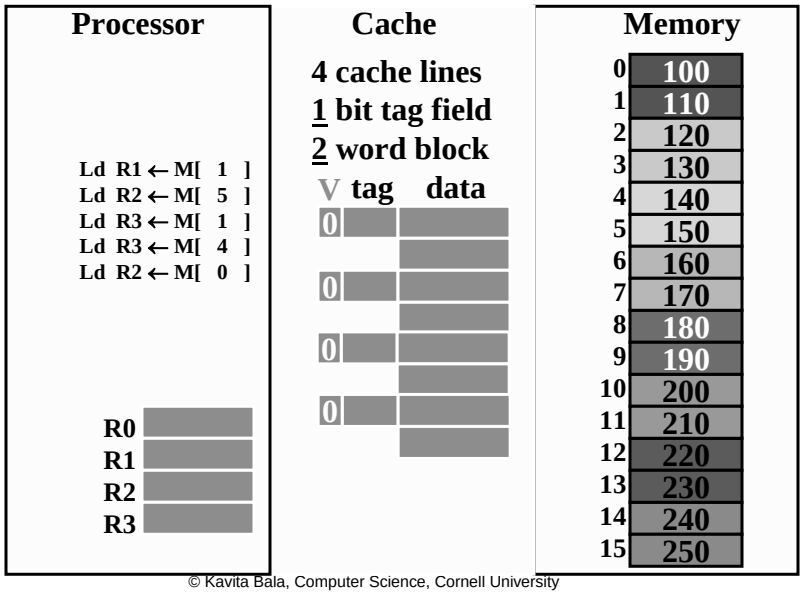
- Cache of size 2^n blocks (index: n bits)
- Block size of 2^m word (block index: m+2 bits)
- Tag field: 32 – (n + m + 2)
- Valid bit: 1

- Bits in cache: 2^n x (block size + tag size + valid bit size)
  - = 2^n (2^m x 32 + (32-n-m-2) + 1)

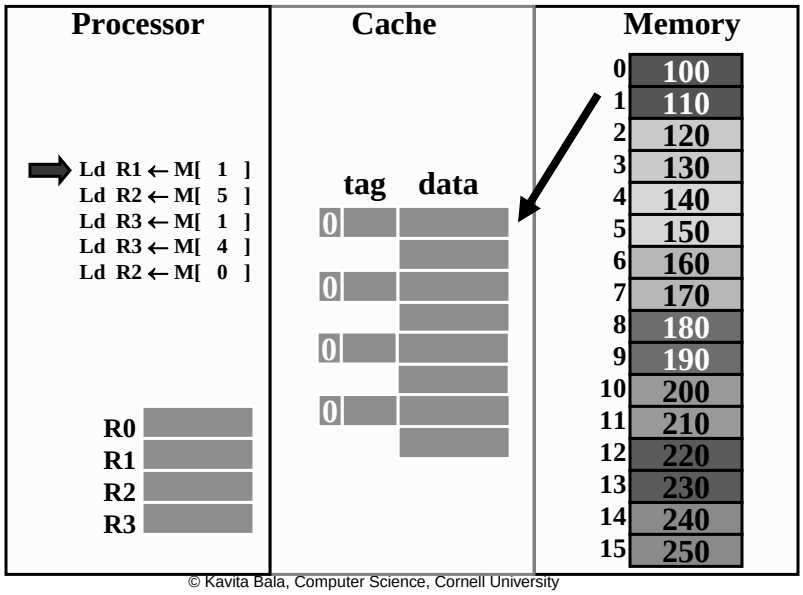
A Simple Direct Mapped Cache



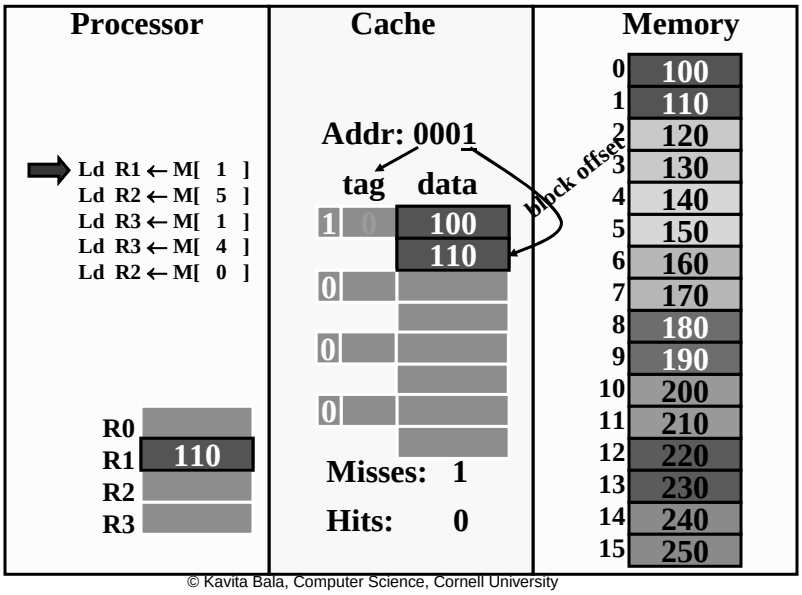
A Simple Direct Mapped Cache



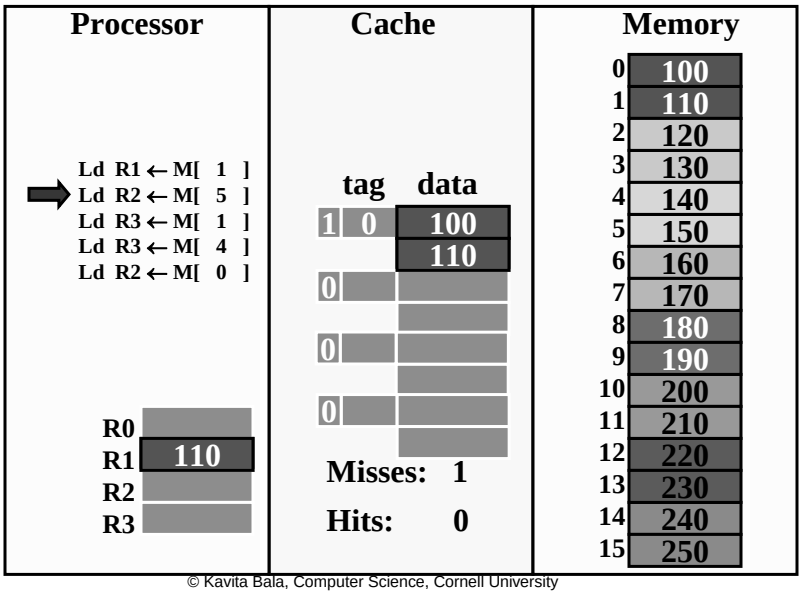
1st Access



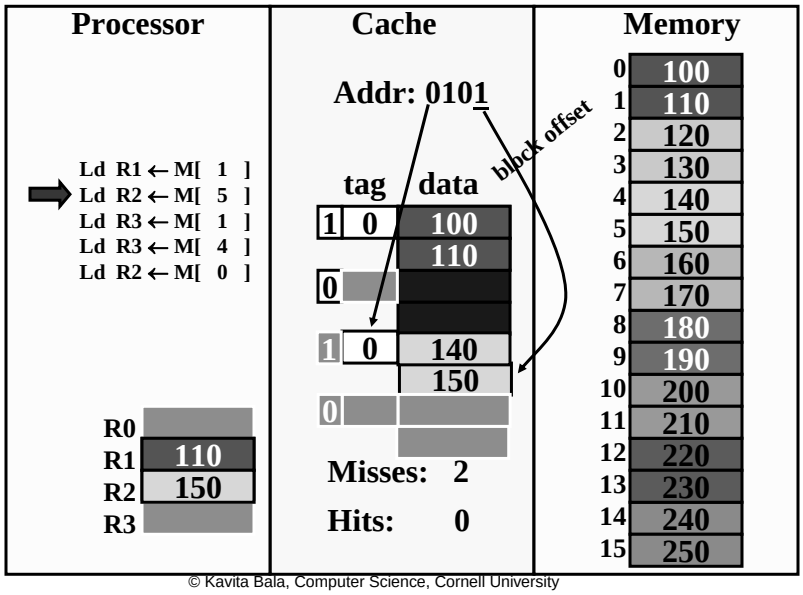
1st Access



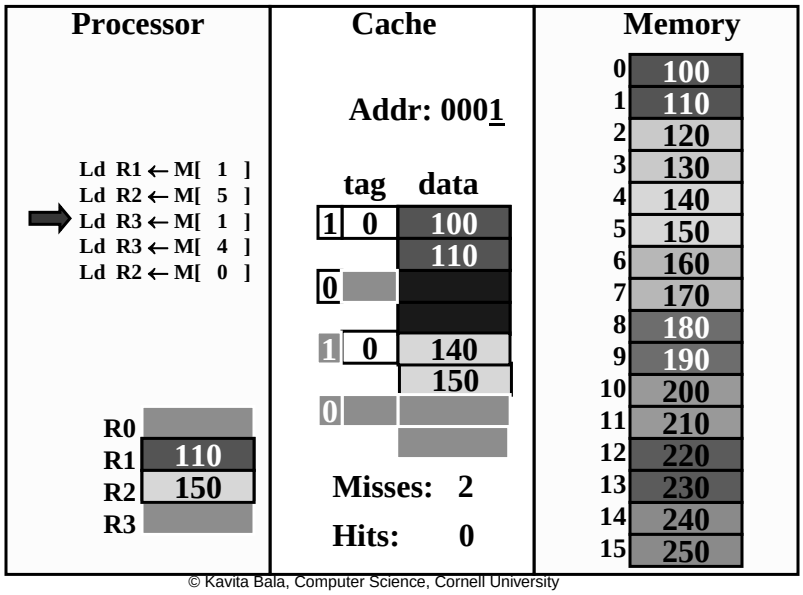
2nd Access



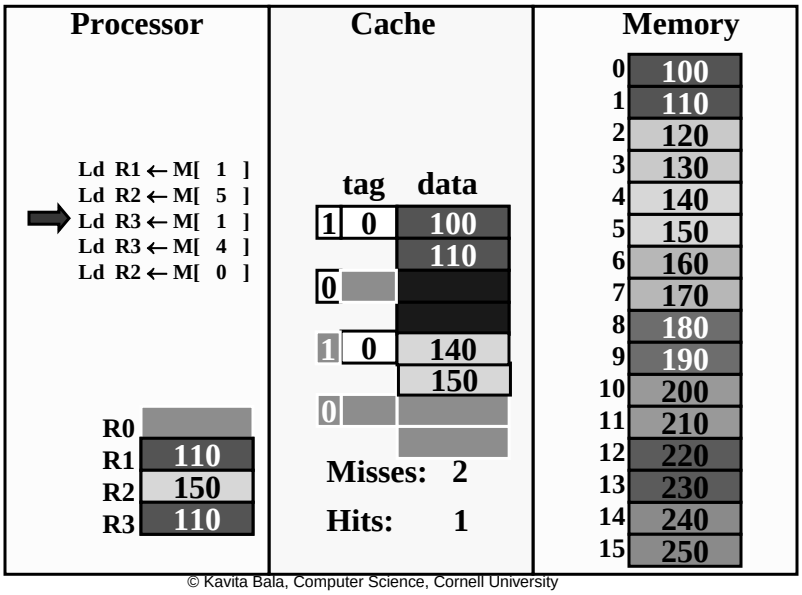
2nd Access



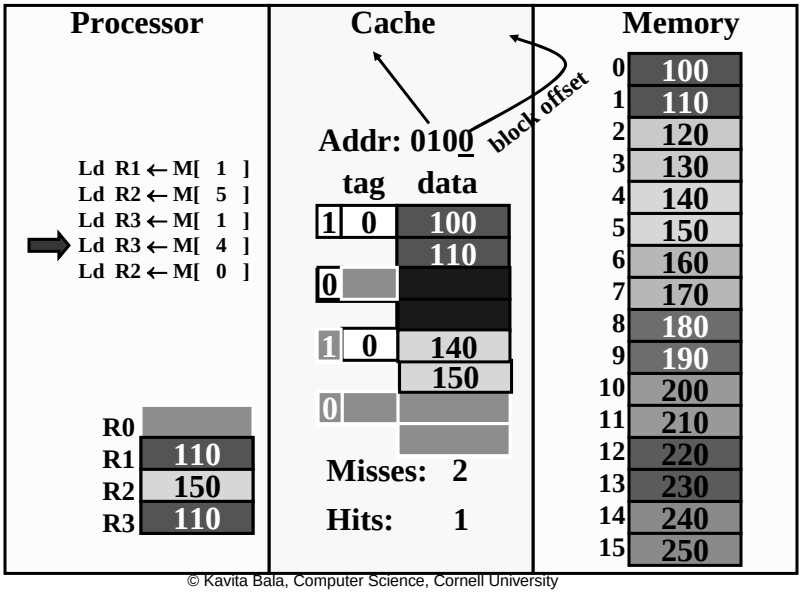
3rd Access



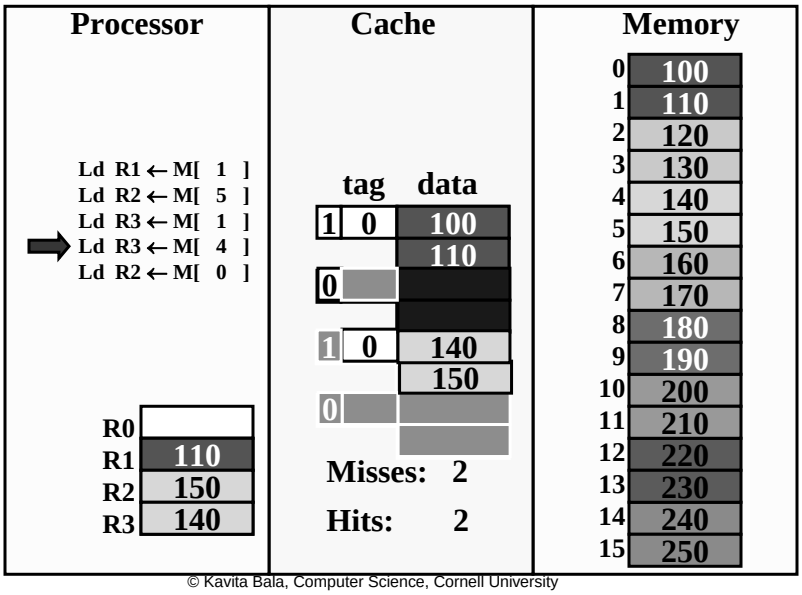
3rd Access



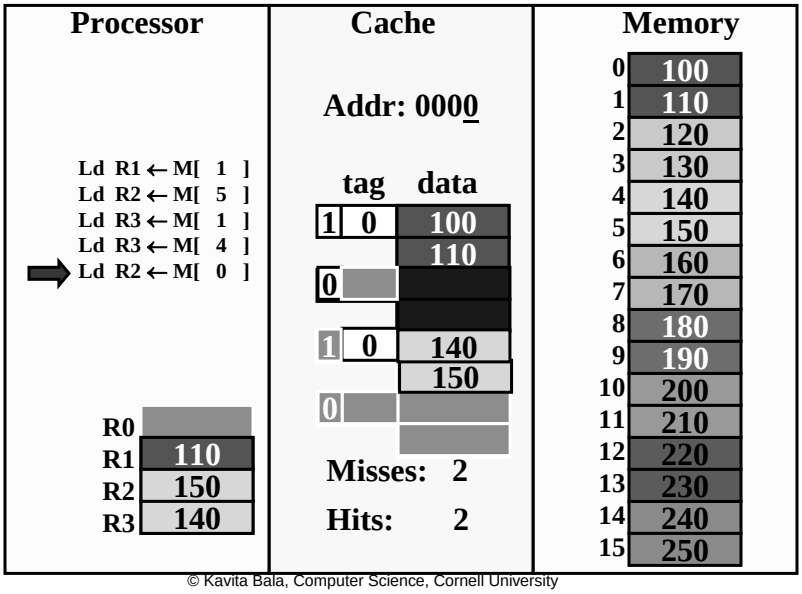
4th Access



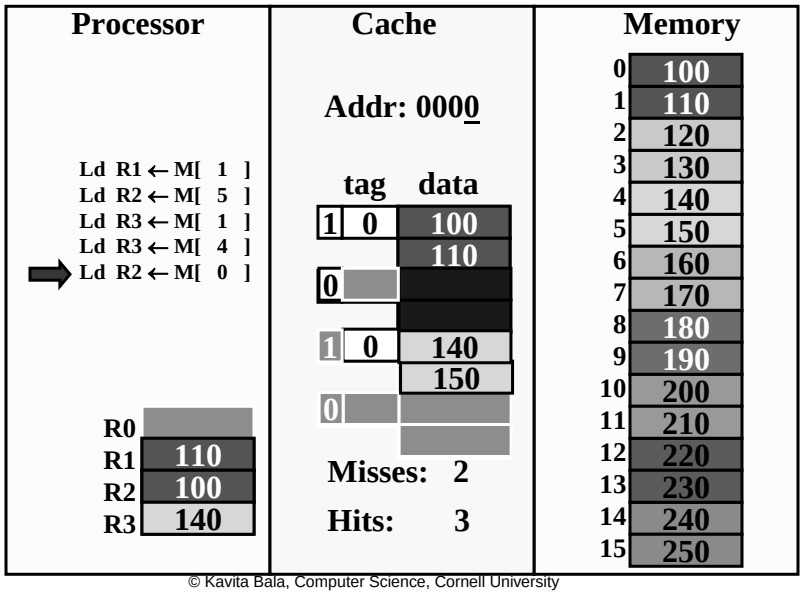
4th Access



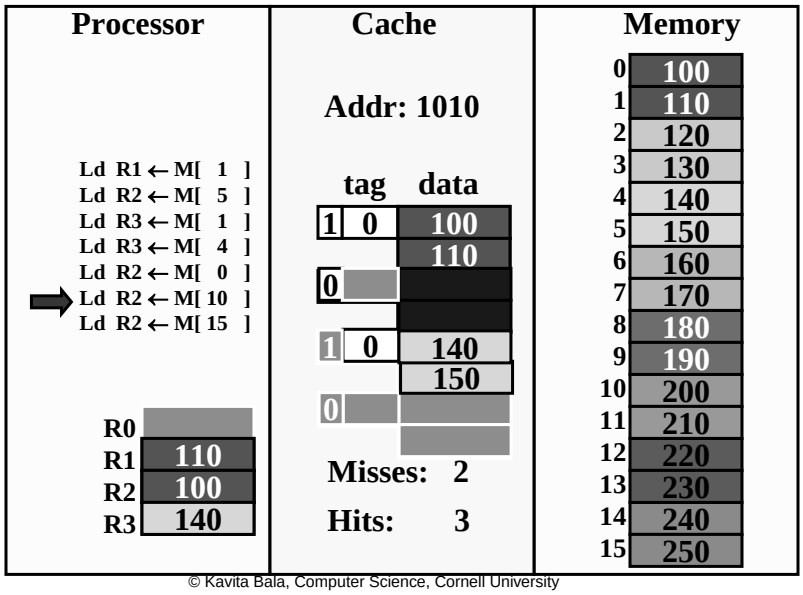
5th Access



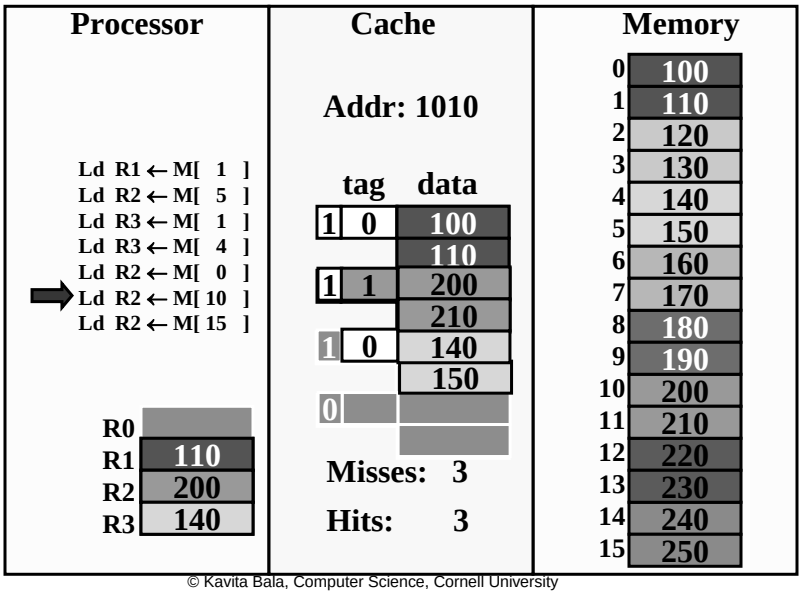
5th Access



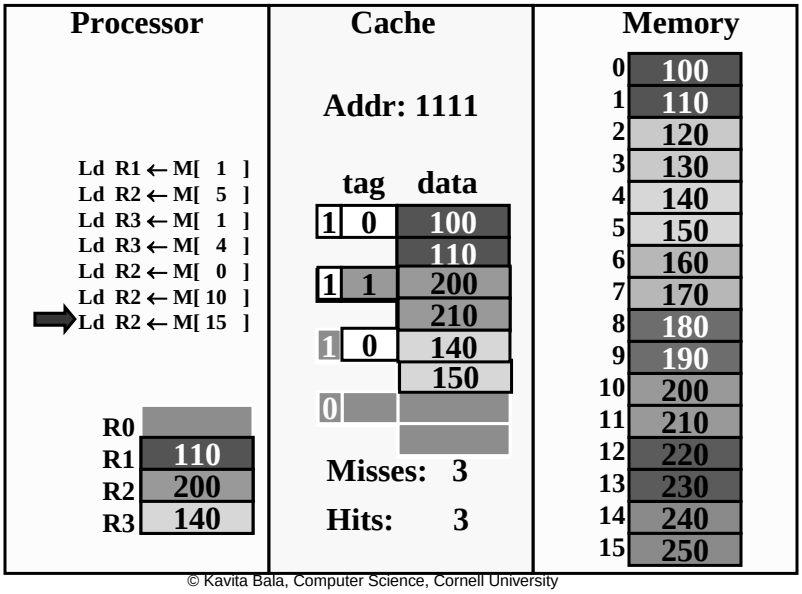
5th Access



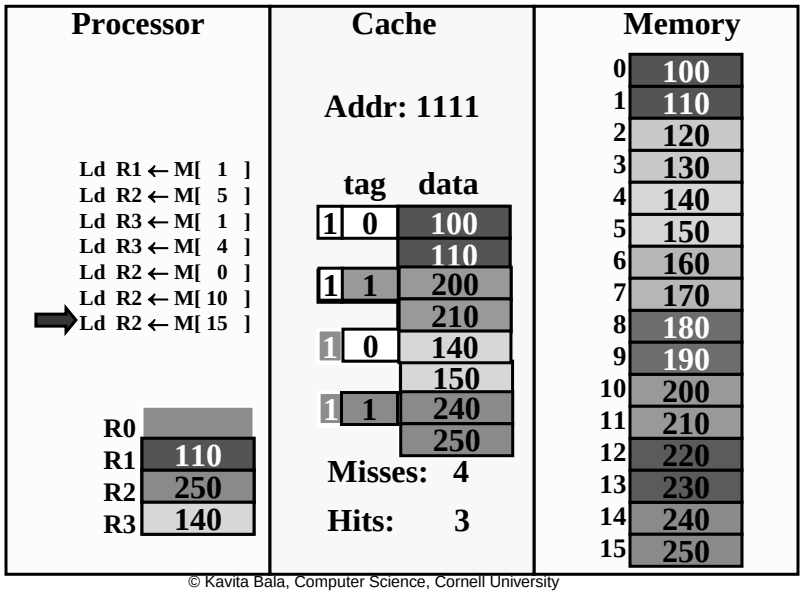
5th Access



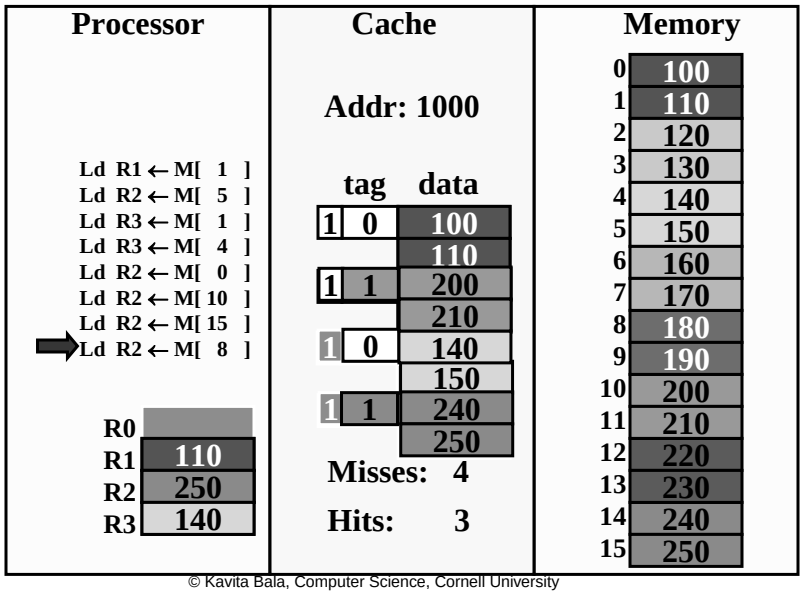
6th Access



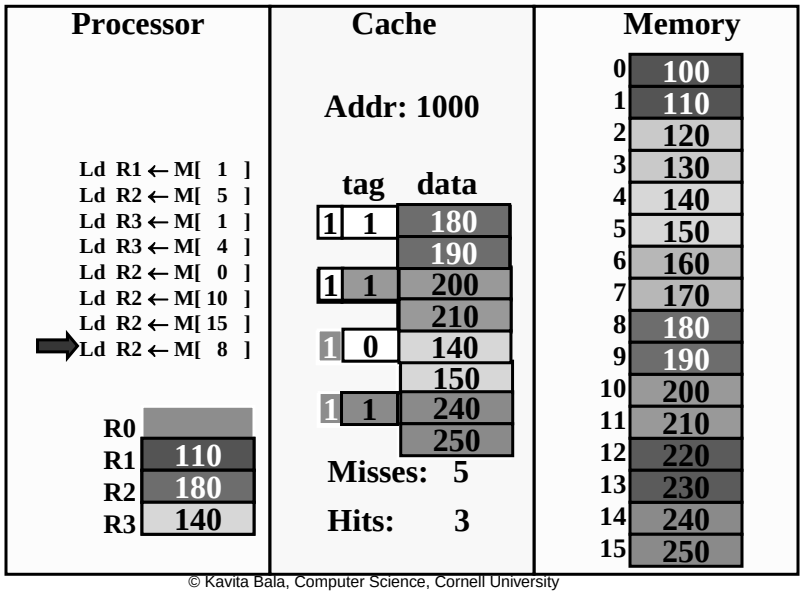
6th Access



7th Access



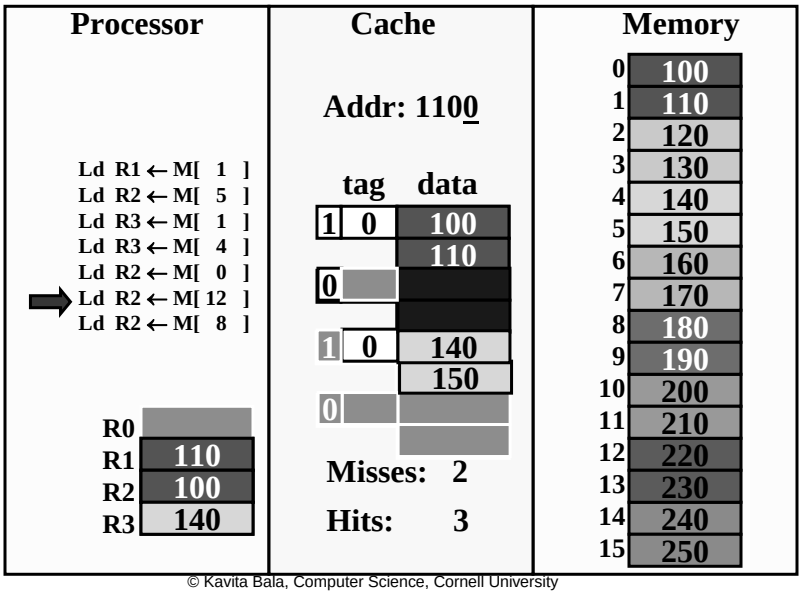
7th Access



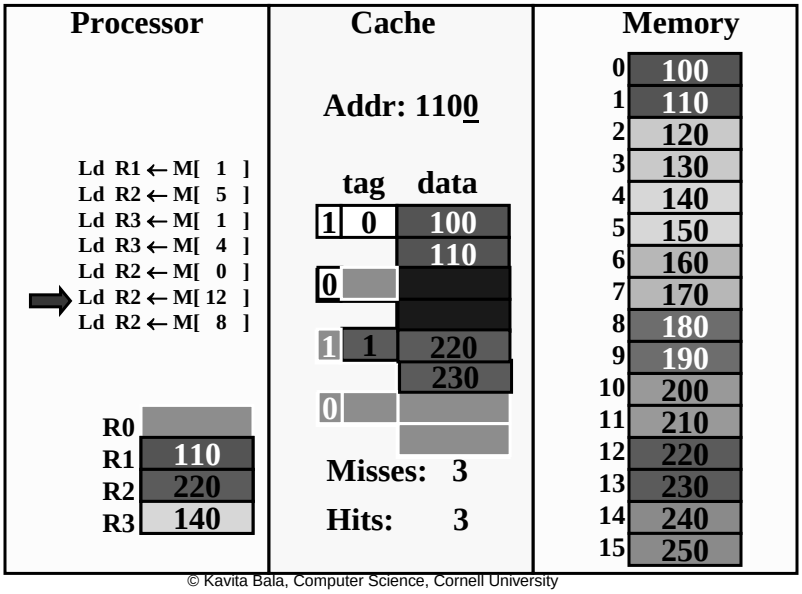
Misses

- Three types of misses
  - Cold
    - The line is being referenced for the first time
  - Capacity
    - The line was evicted because the cache was not large enough
  - Conflict
    - The line was evicted because of another access whose index conflicted

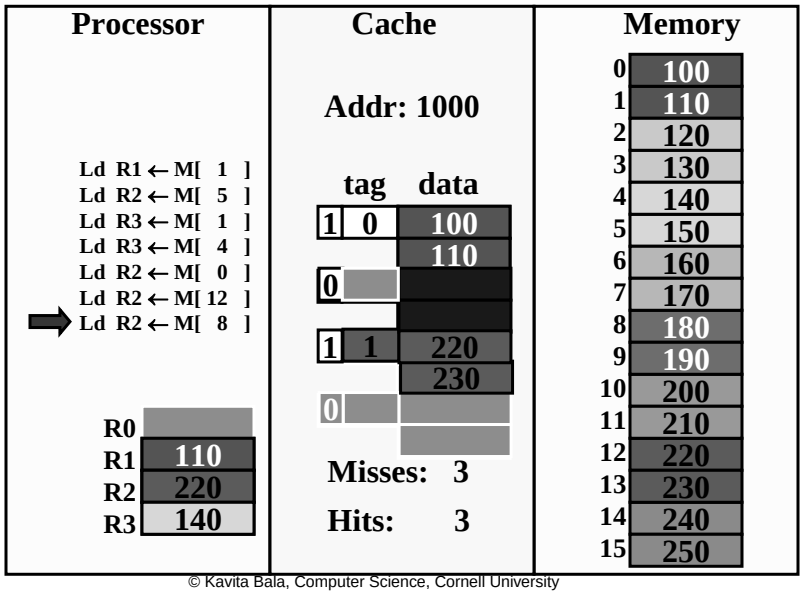
6th Access



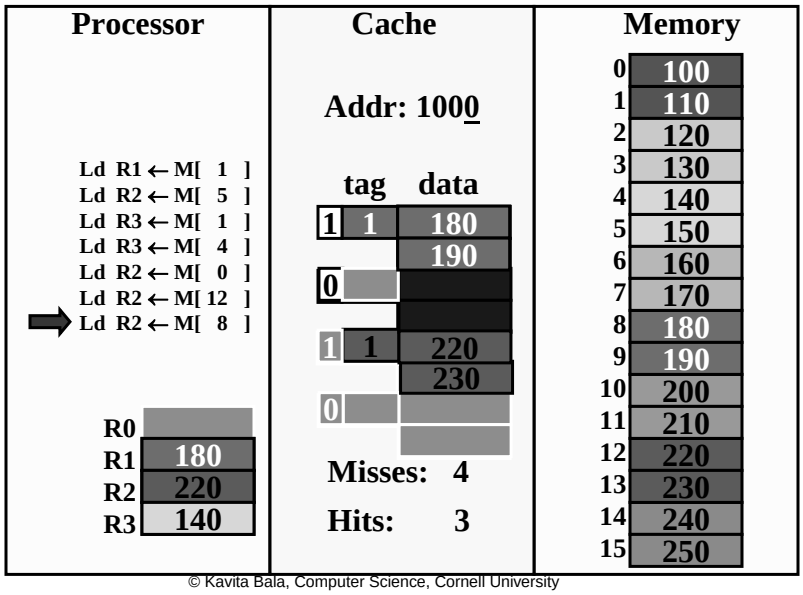
6th Access



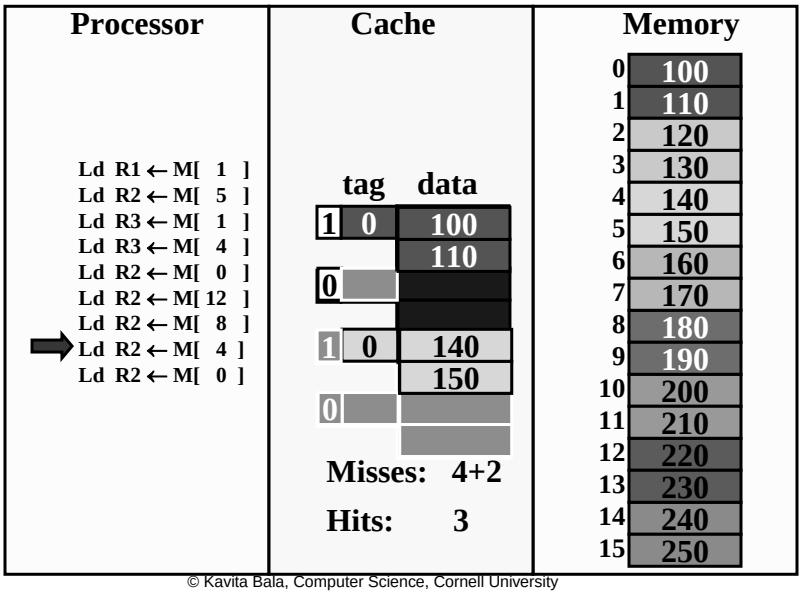
7th Access



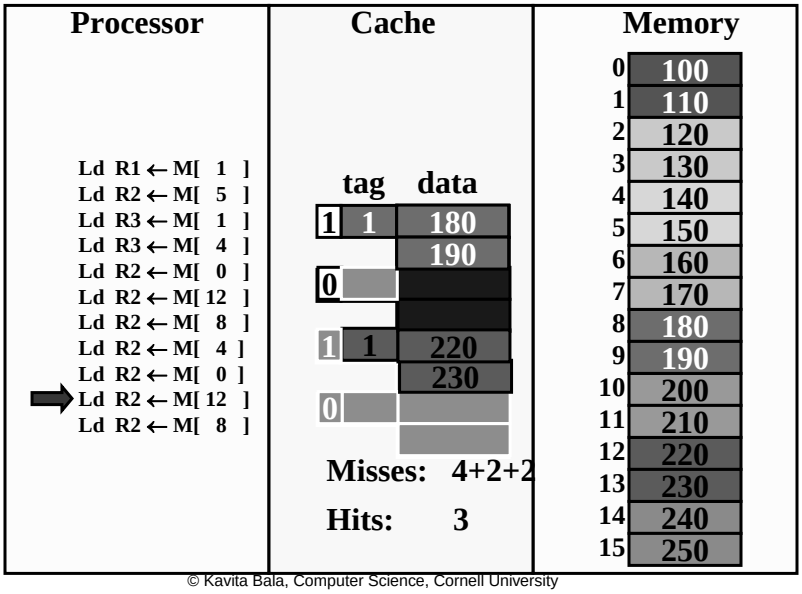
7th Access



8th and 9th Access



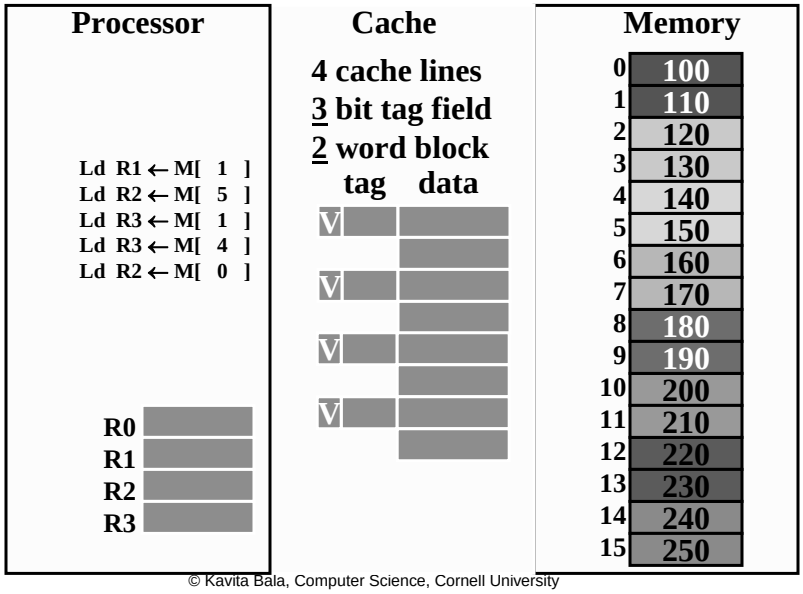
10th and 11th Access



Cache Organization

- Three common designs
  - Fully associative: Block can be anywhere in the cache
  - Direct mapped: Block can only be in one line in the cache
  - Set-associative: Block can be in a few (2 to 8) places in the cache

A Simple Fully Associative Cache

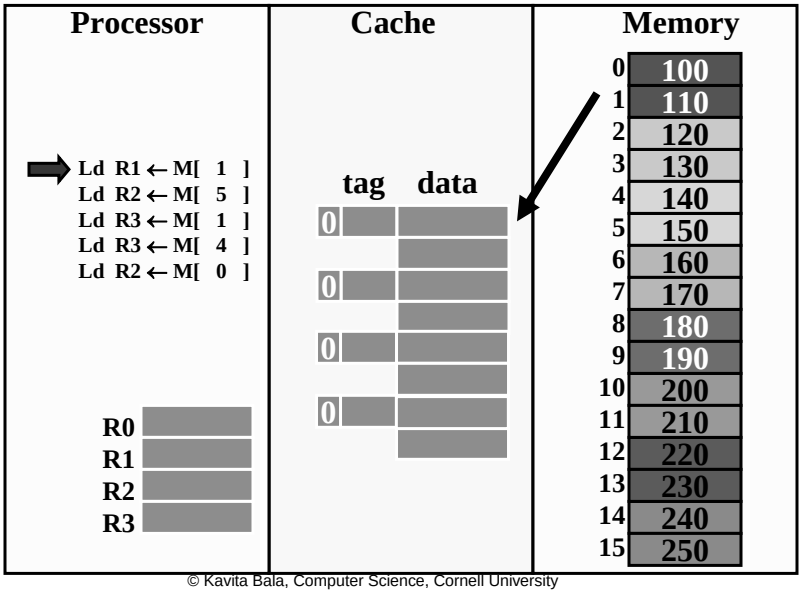


Cache Size

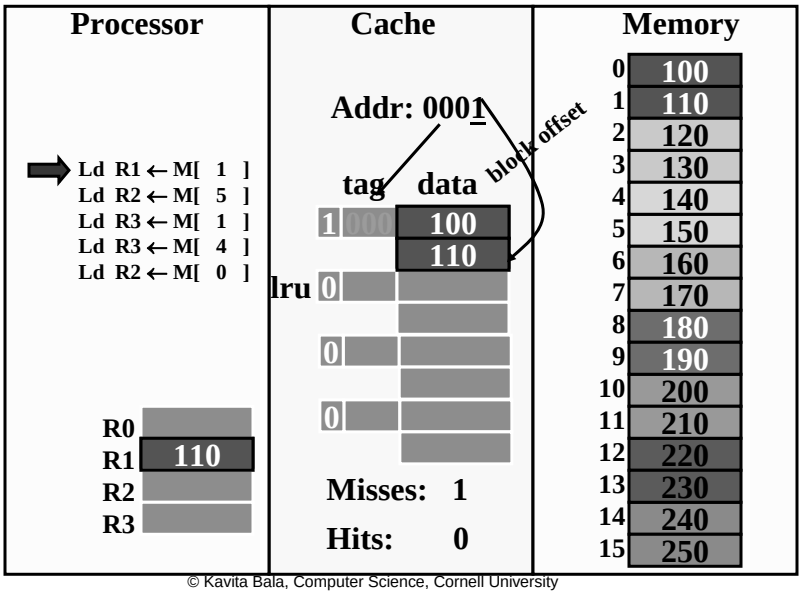
- Cache of size 2^n blocks
- Block size of 2^m word (block index: m+2 bits)
- Tag field: 32 – (m + 2)
- Valid bit: 1

- Bits in cache: 2^n x (block size + tag size + valid bit size)
  - = 2^n (2^m x 32 + (32-m-2) + 1)

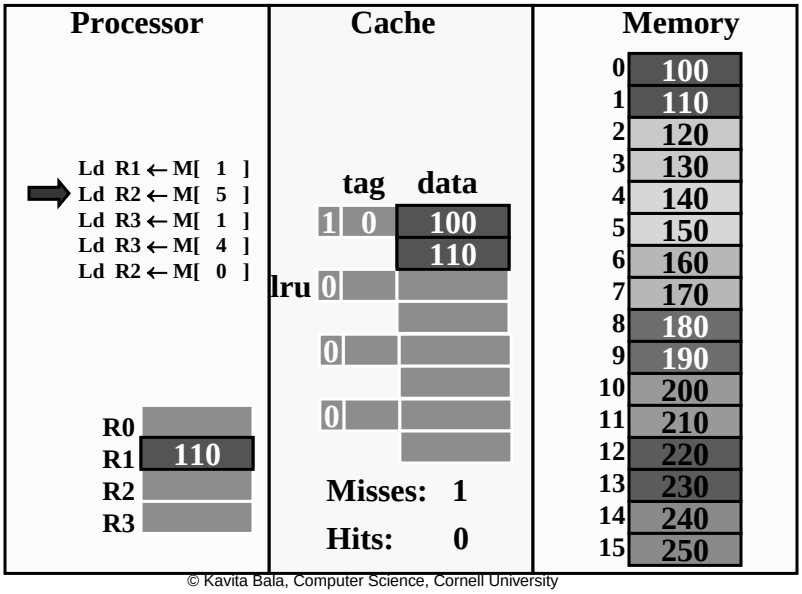
1st Access



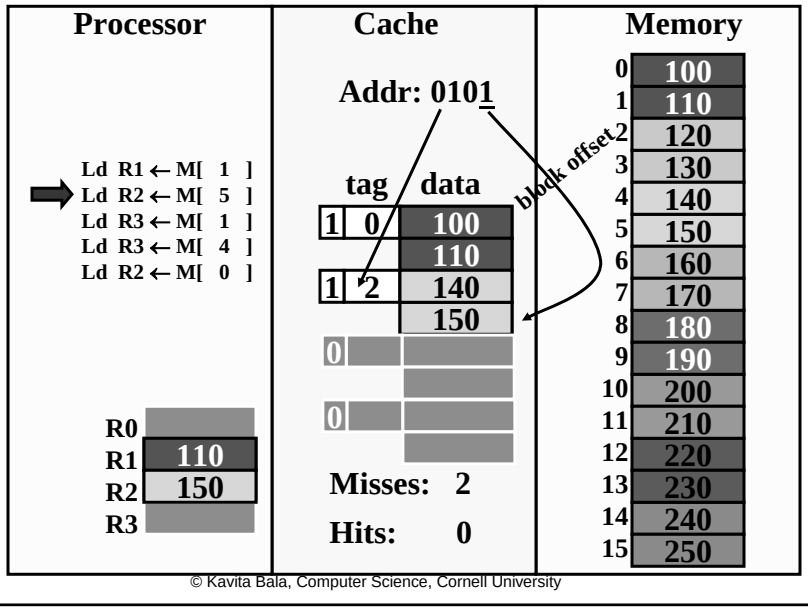
1st Access



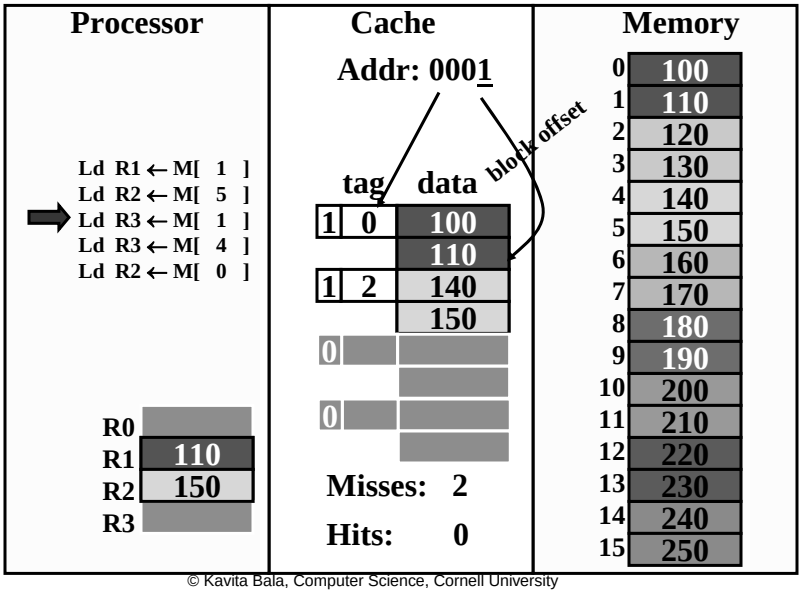
2nd Access



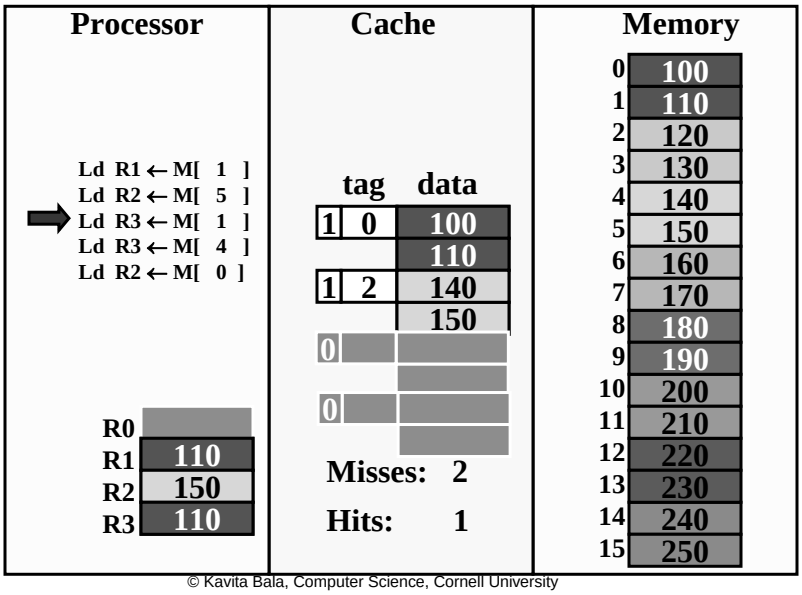
2nd Access



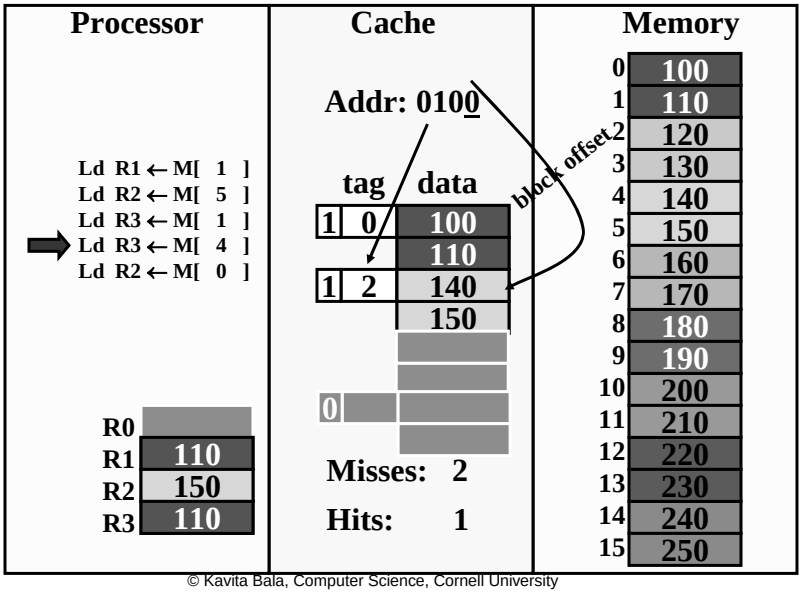
3rd Access



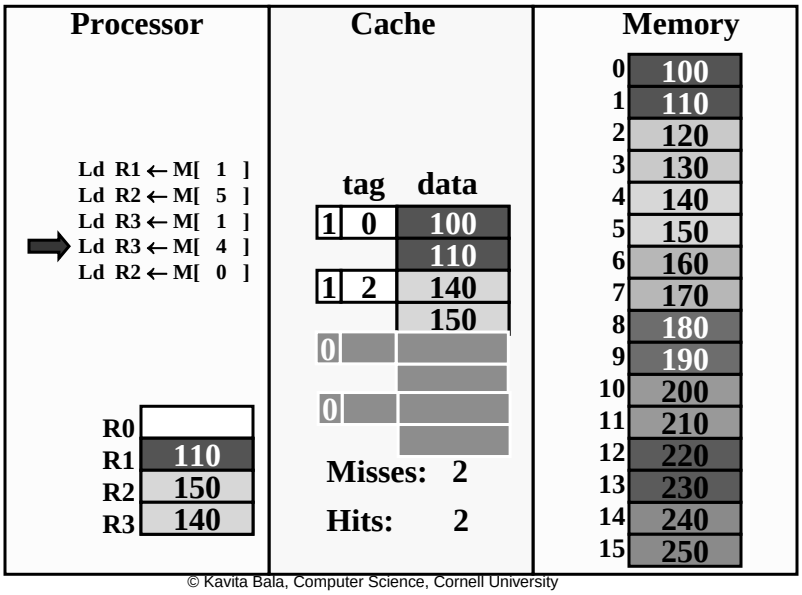
3rd Access



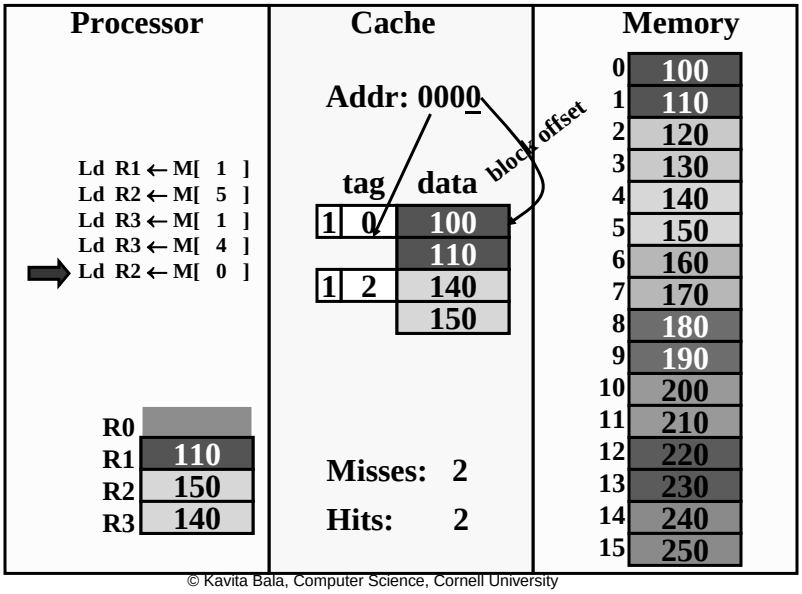
4th Access



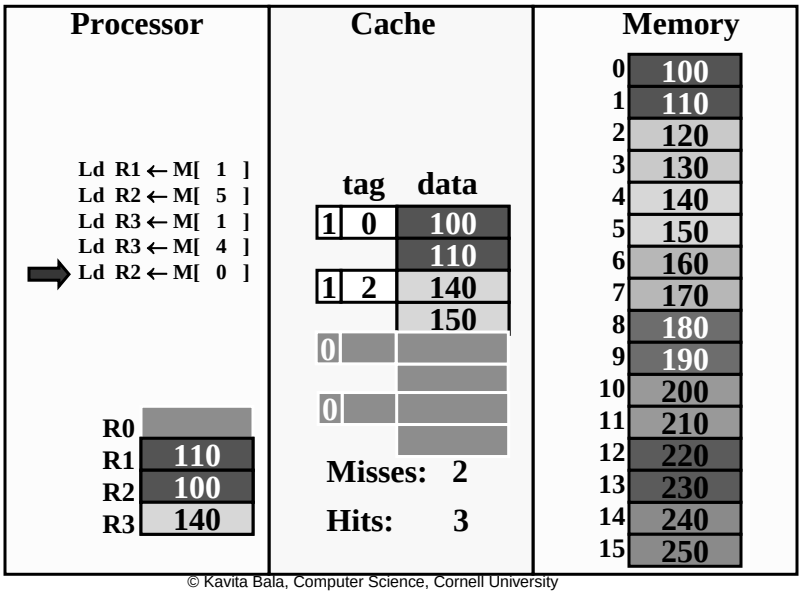
4th Access



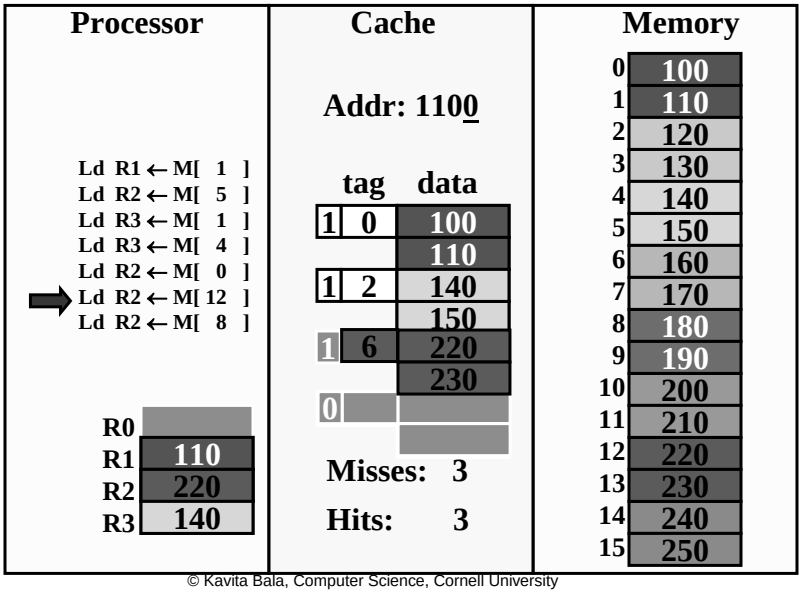
5th Access



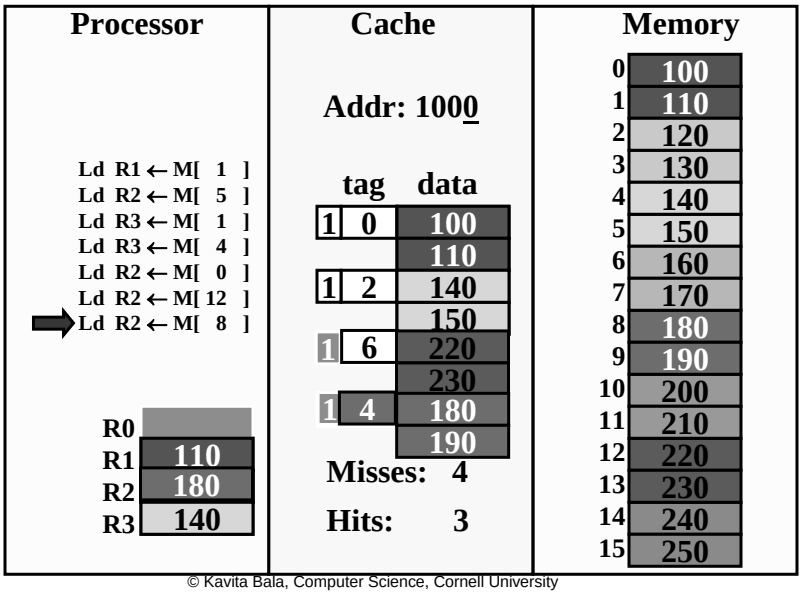
5th Access



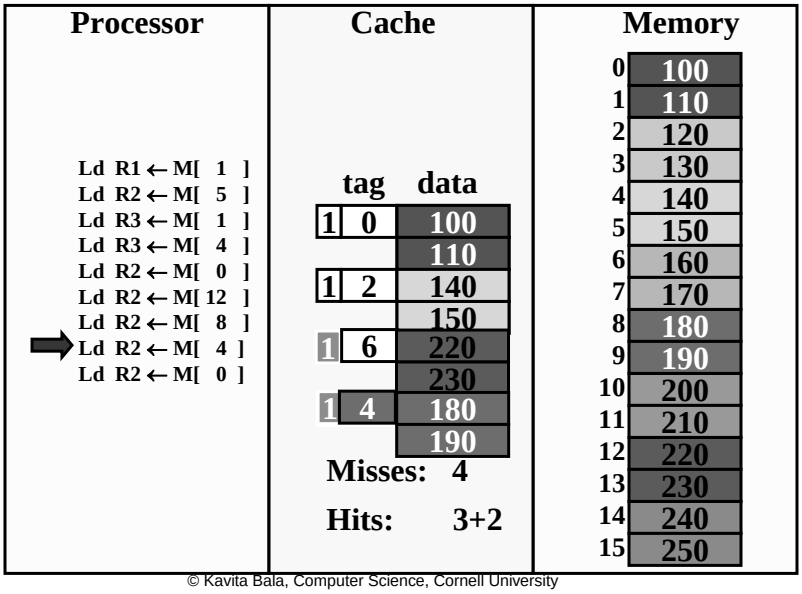
6th Access



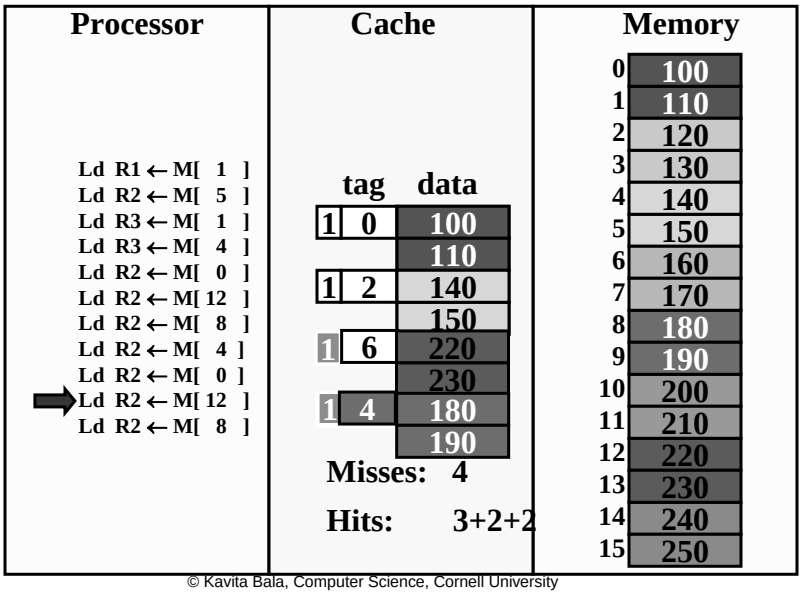
7th Access



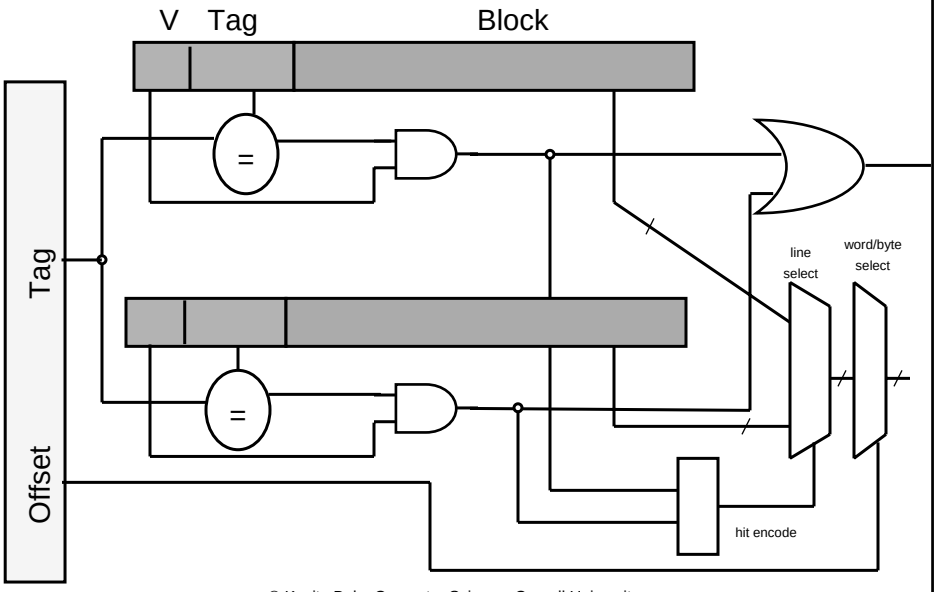
8th and 9th Access



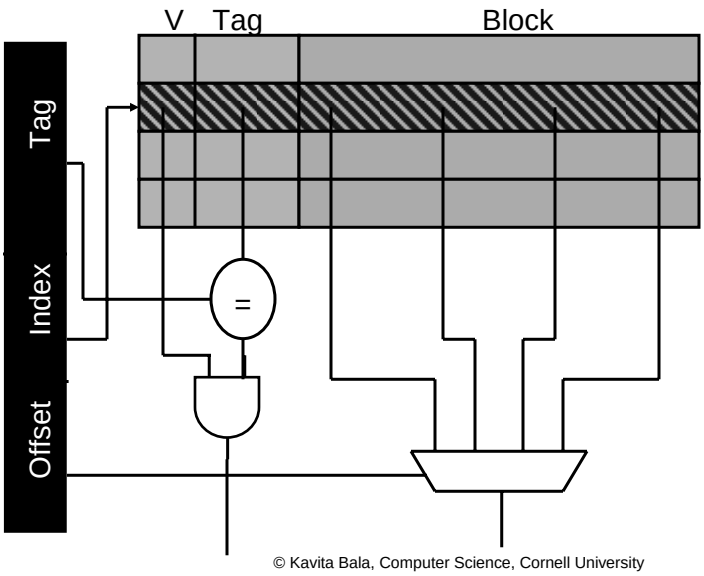
10th and 11th Access



Fully Associative Cache



Direct Mapped Cache



Eviction

- Which cache line should be evicted from the cache to make room for a new line?
  - Direct-mapped
    - no choice, must evict line selected by index
  - Associative caches
    - random: select one of the lines at random
    - round-robin: similar to random
    - FIFO: replace oldest line
    - LRU: replace line that has not been used in the longest time