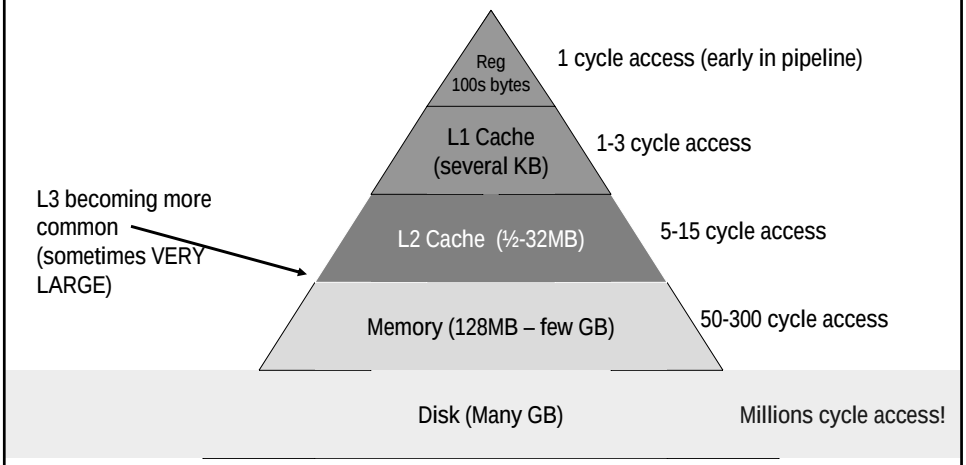
CS 316: Caches-II

Kavita Bala
Fall 2007
Computer Science
Cornell University

Announcements

- HW 2 is out
- Caches and Memory: Chapter 7 (H & P)

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)

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

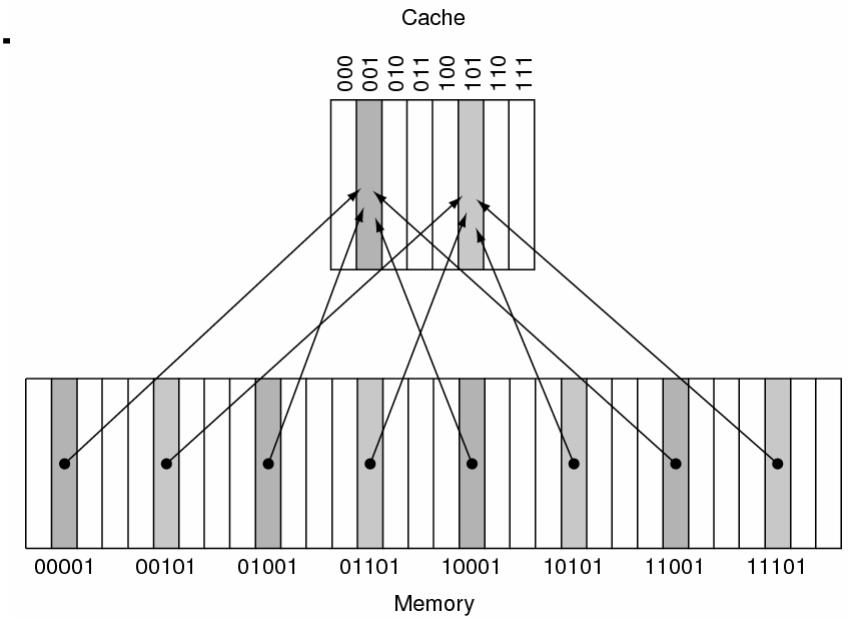
Cache Organization

- 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

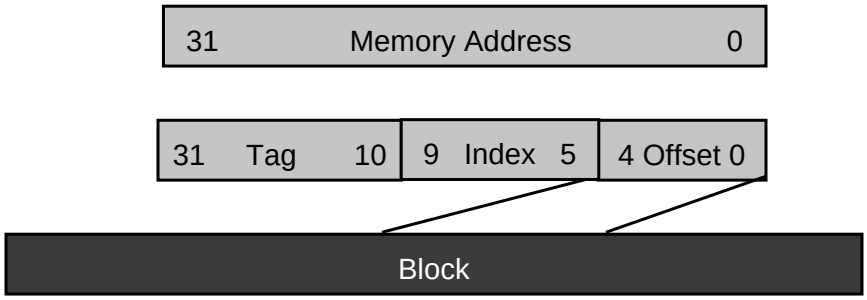
- Simplest
- Block can only be in one line in the cache

- How to determine this location?
  - Use modulo arithmetic
  - (Block address) modulo (# cache blocks)
  - For power of 2, use log (cache size in blocks)



Tags and Offsets

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



Valid Bits

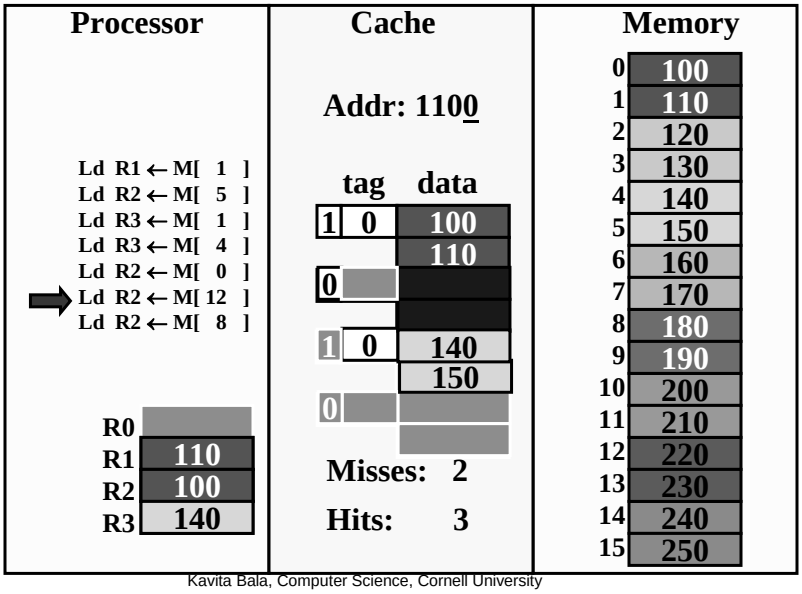
- Valid bits indicate whether cache line contains an up-to-date copy of the values in memory
  - Must be 1 for a hit
  - Reset to 0 on power up
- An item can be removed from the cache by setting its valid bit to 0

Cache Size

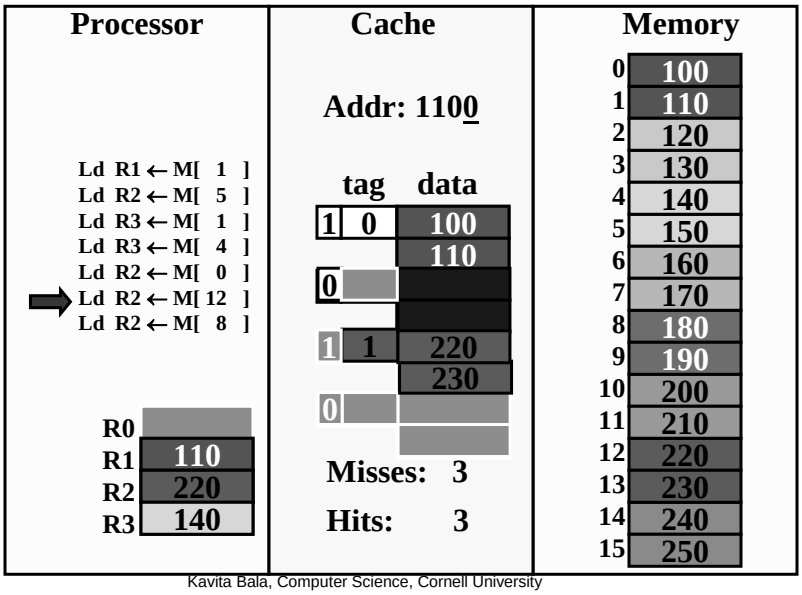
- Cache of size 2^n blocks (index: n bits)
- Block size of 2^m word (block index: m+2 bits)
- Tag field: 32 – (n + m + 2)
- Valid bit: 1

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

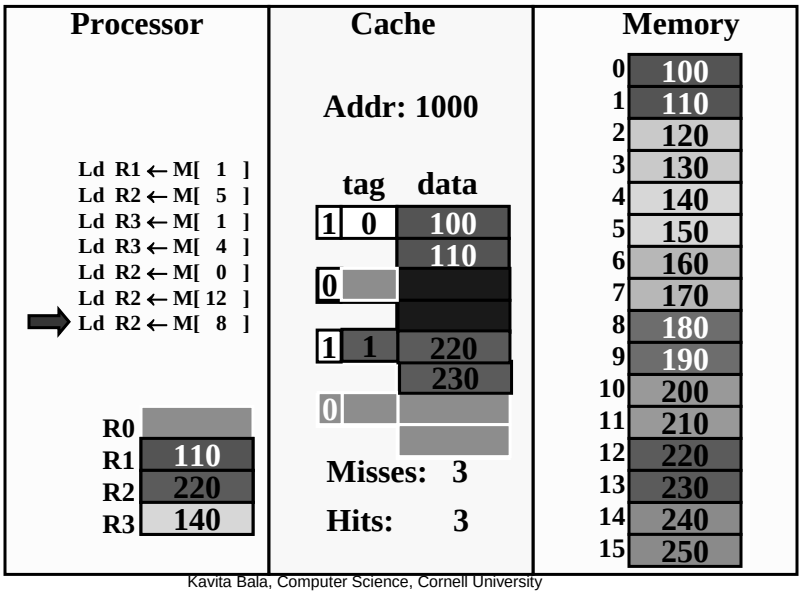
6th Access



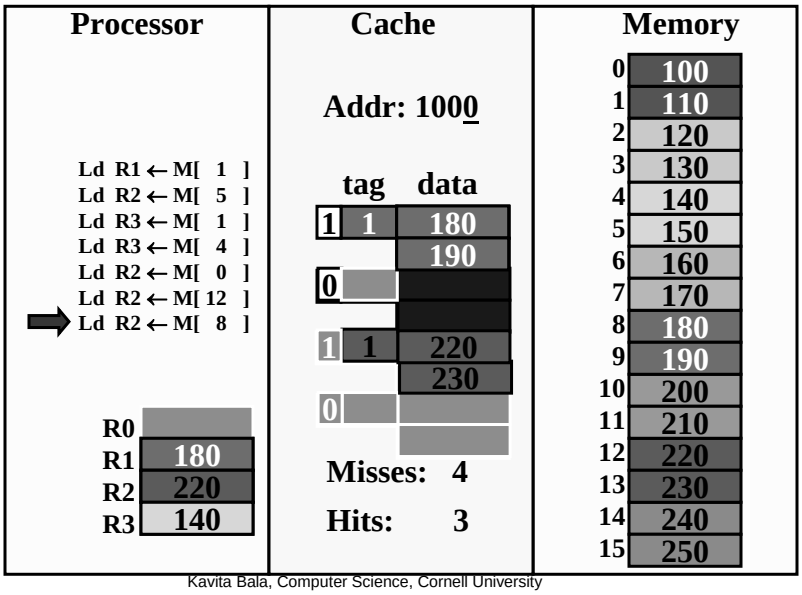
6th Access



7th Access

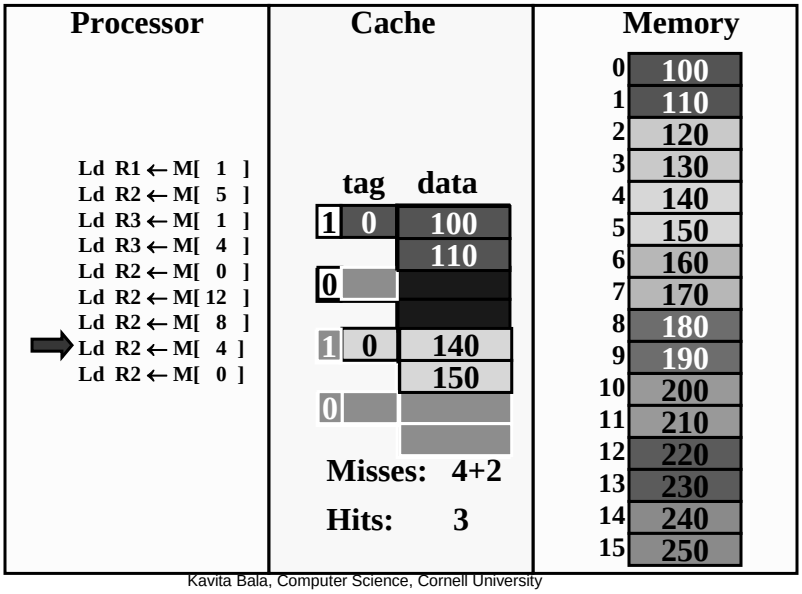


7th Access



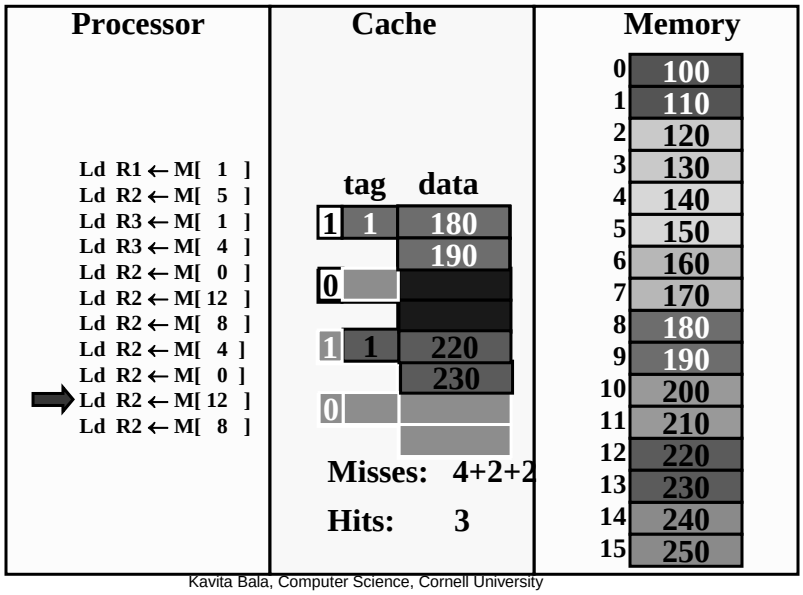
Kavita Bala, Computer Science, Cornell University

8th and 9th Access



Kavita Bala, Computer Science, Cornell University

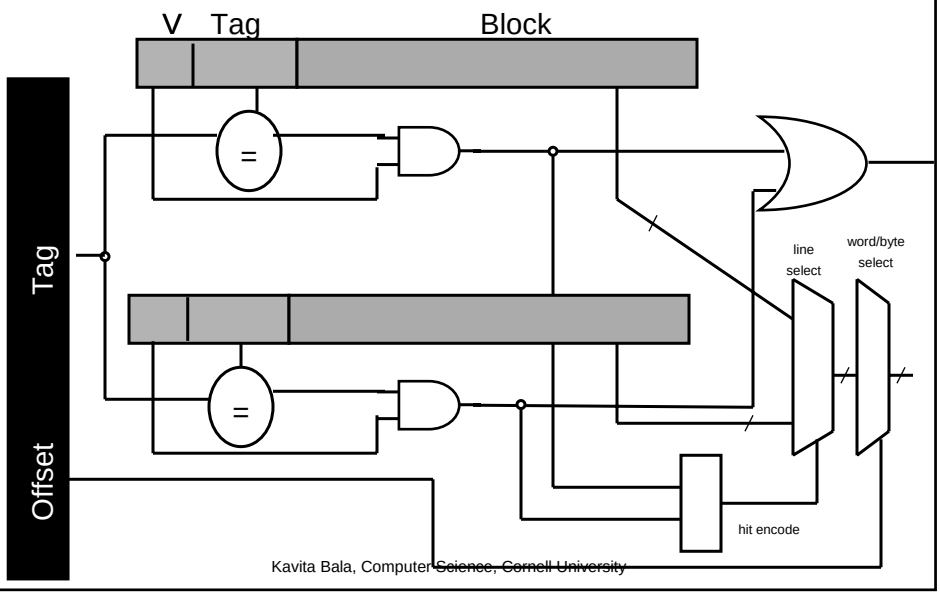
DM: 10th and 11th Access



Misses

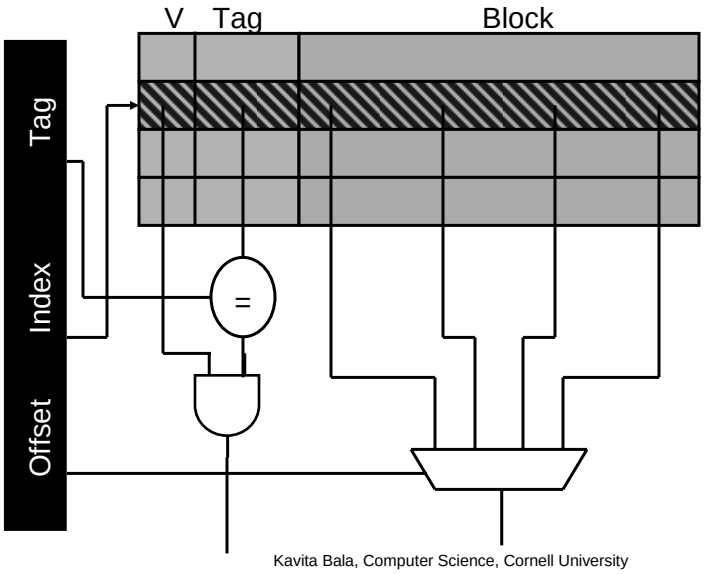
- Three types of misses
  - Cold
    - The line is being referenced for the first time
  - Capacity
    - The line was evicted because the cache was not large enough
  - Conflict
    - The line was evicted because of another access whose index conflicted

Fully Associative Cache



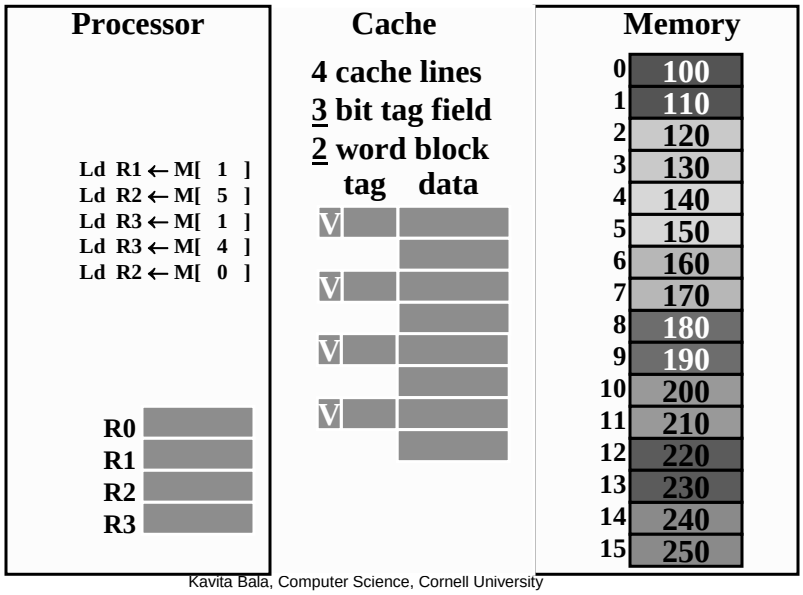
Kavita Bala, Computer Science, Cornell University

Comparison: Direct Mapped Cache

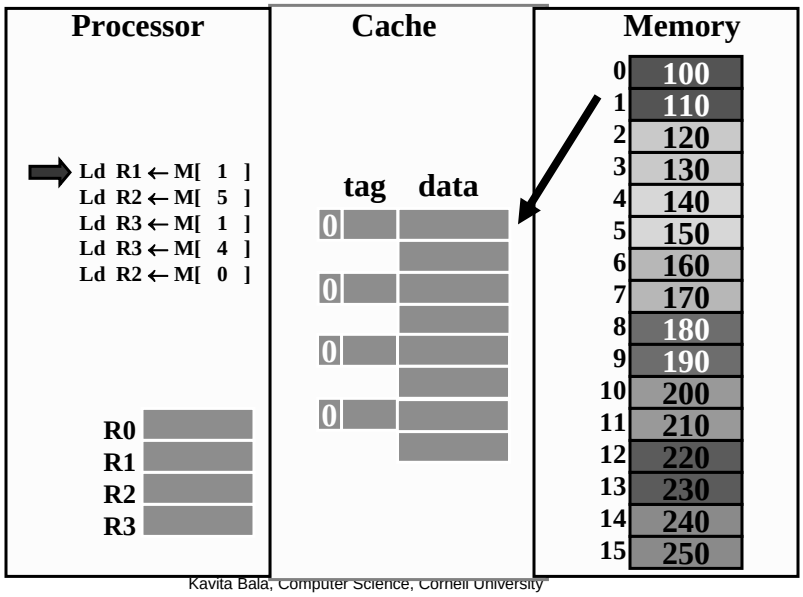


Kavita Bala, Computer Science, Cornell University

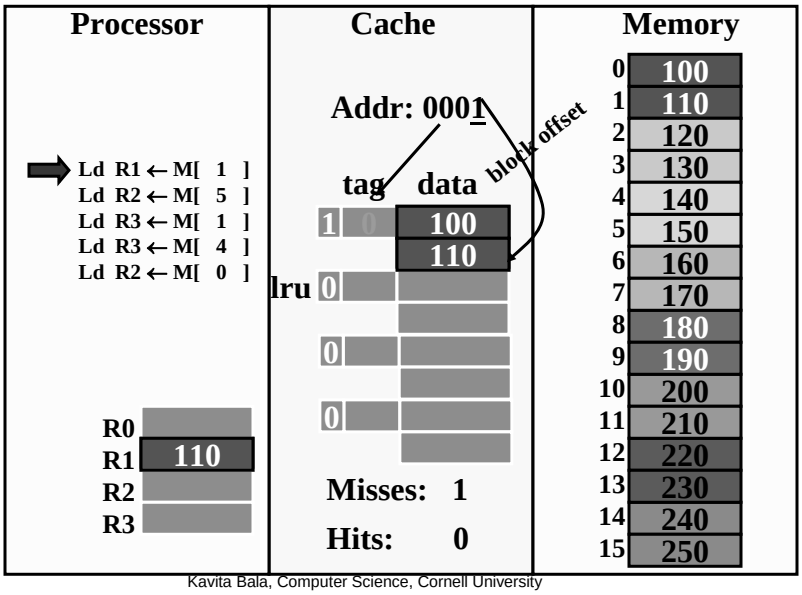
A Simple Fully Associative Cache



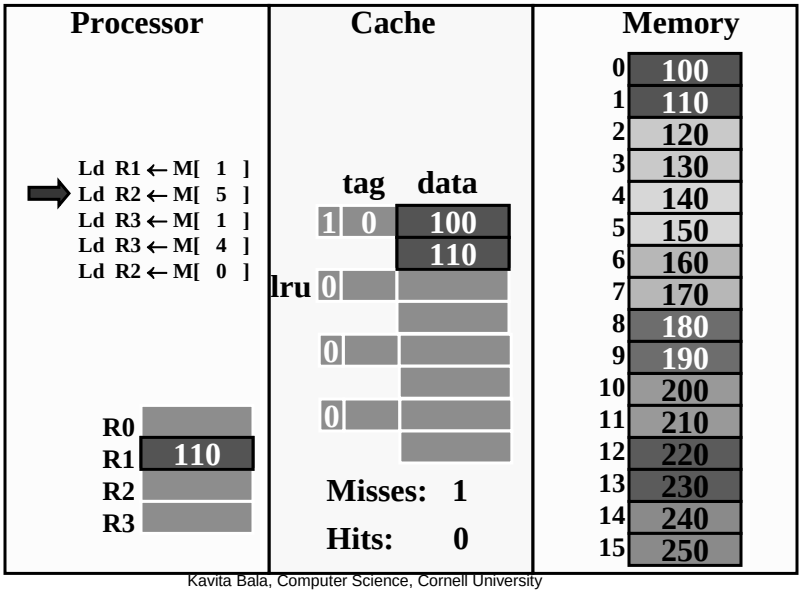
1st Access



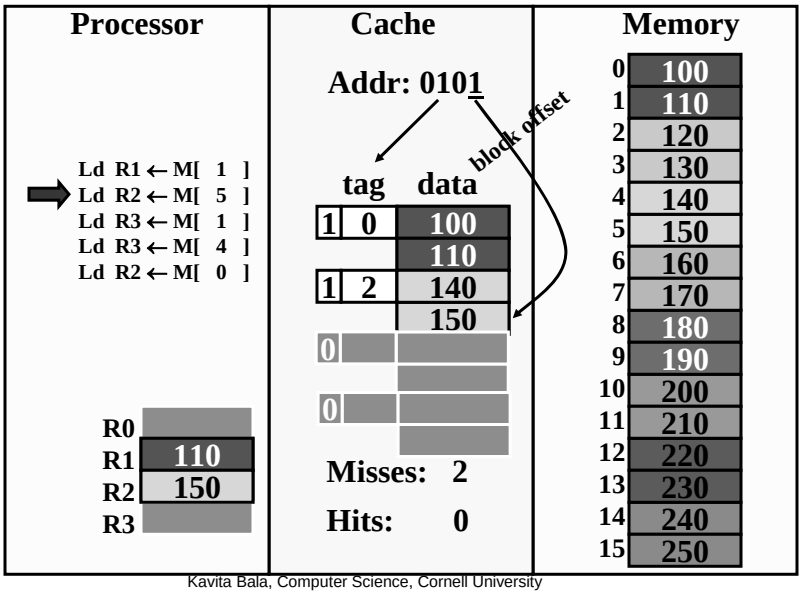
1st Access



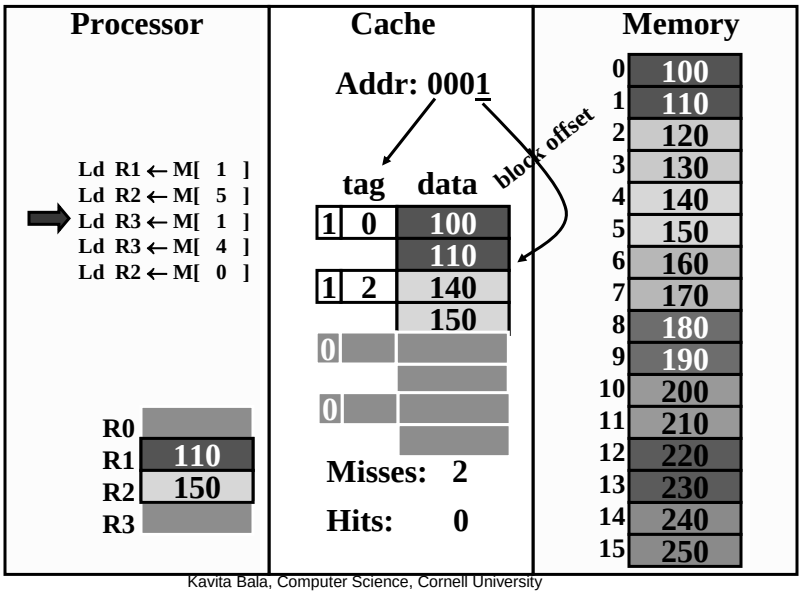
2nd Access



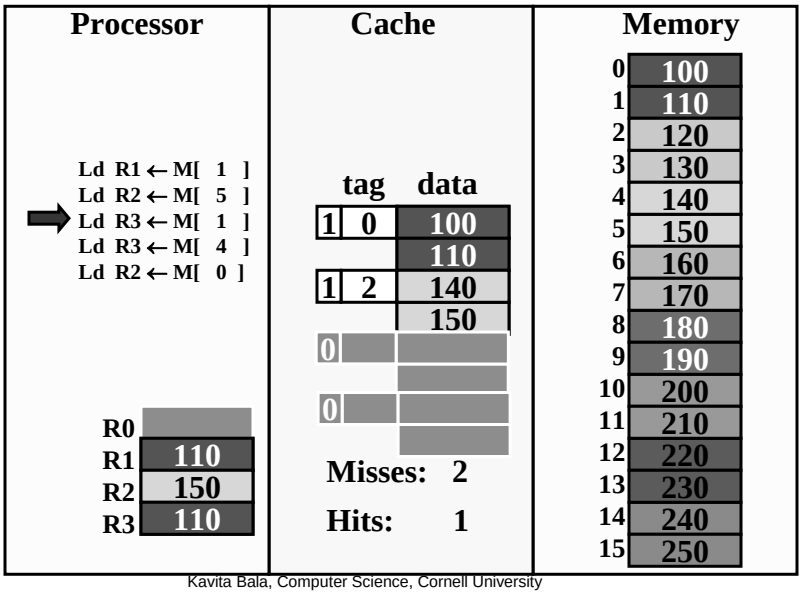
2nd Access



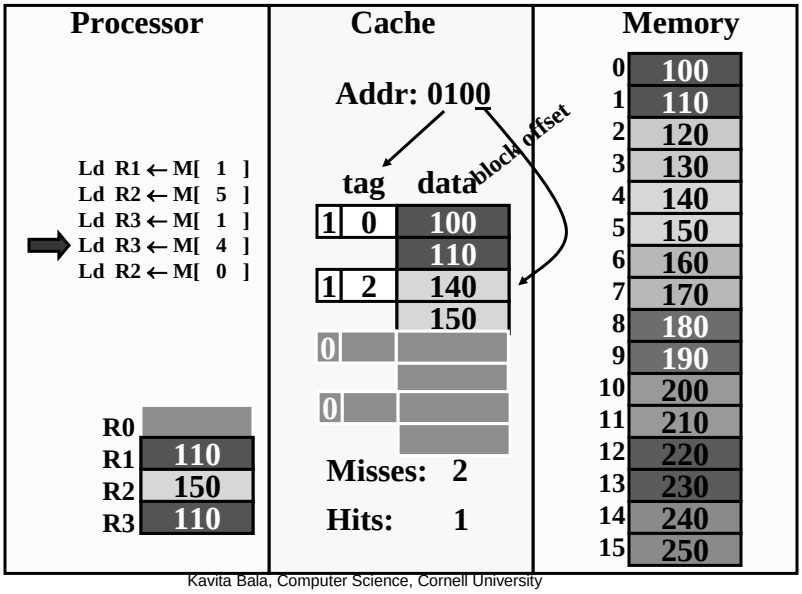
3rd Access



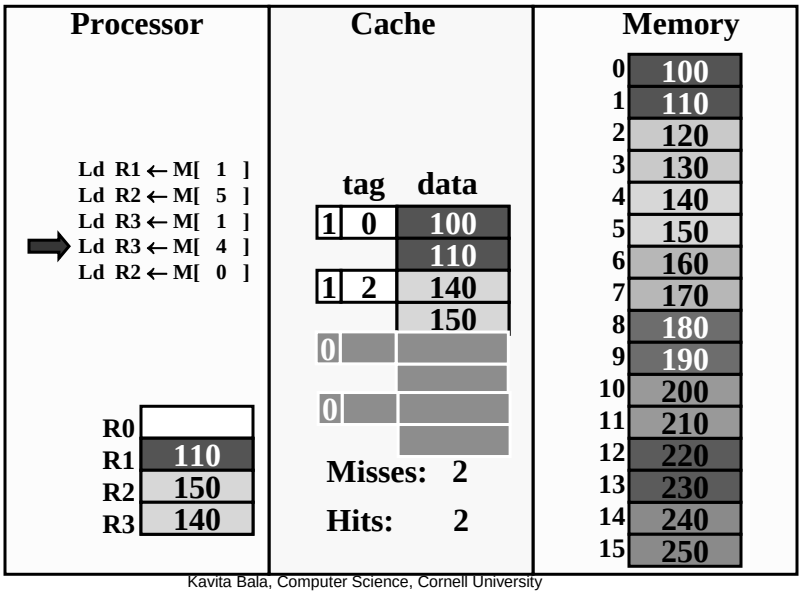
3rd Access



4th Access

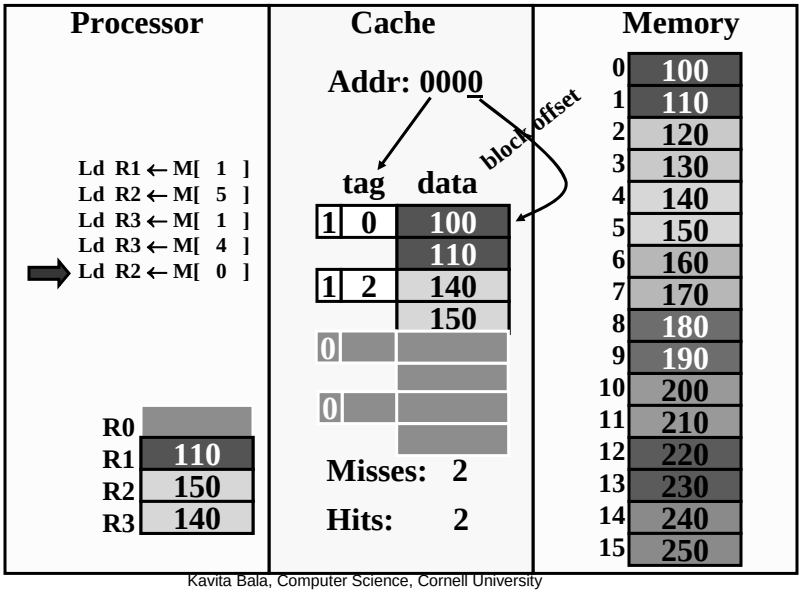


4th Access



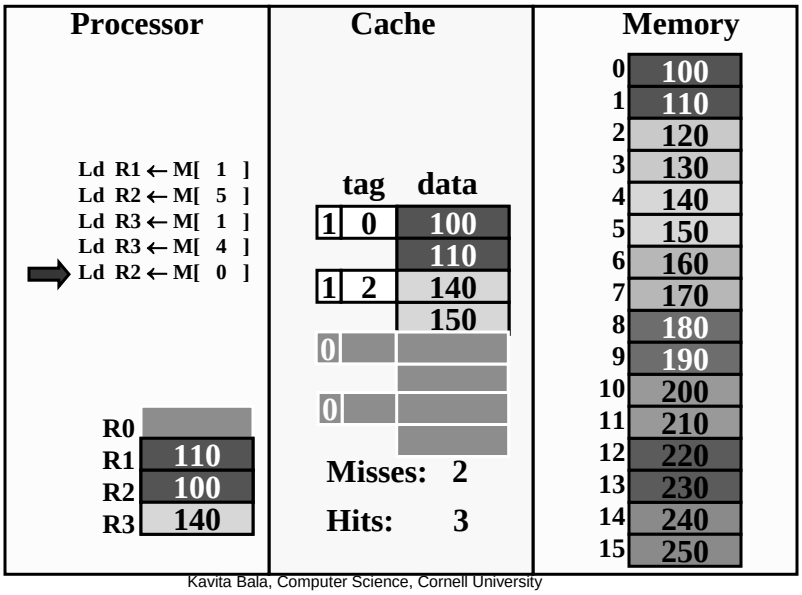
Kavita Bala, Computer Science, Cornell University

5th Access



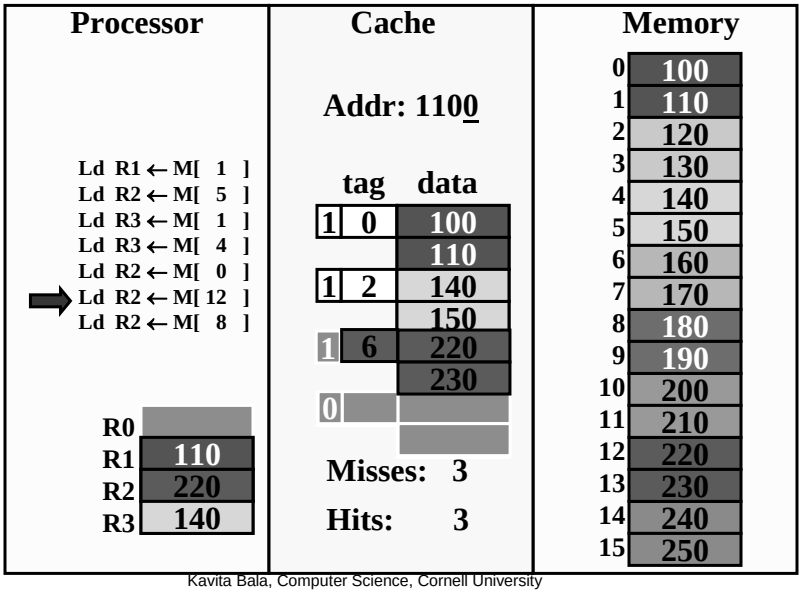
Kavita Bala, Computer Science, Cornell University

5th Access



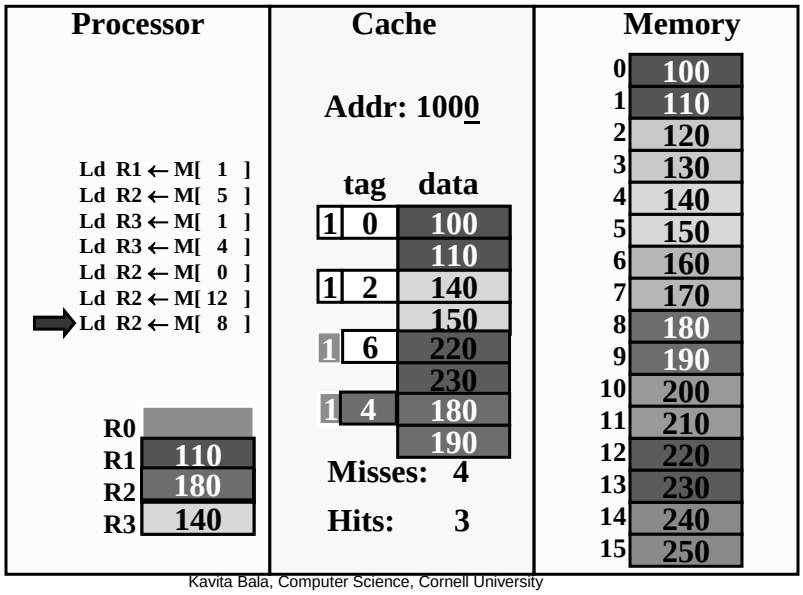
Kavita Bala, Computer Science, Cornell University

6th Access

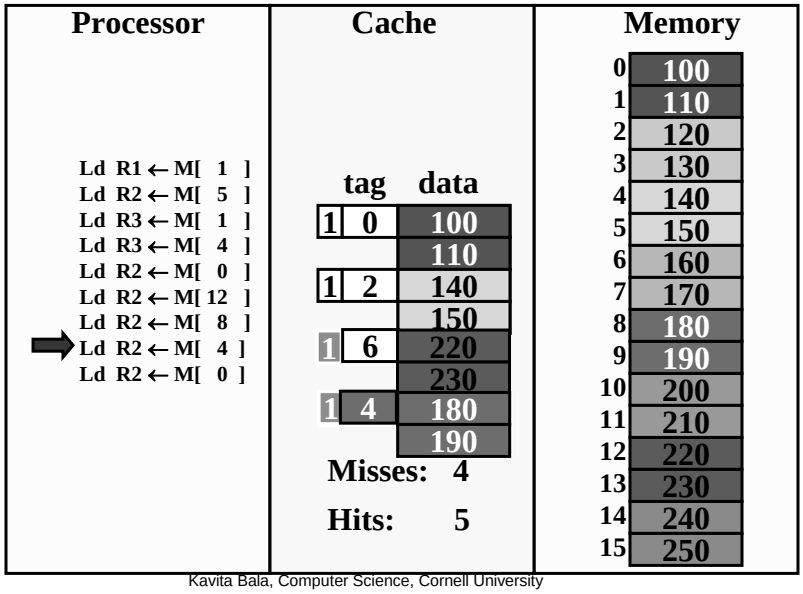


Kavita Bala, Computer Science, Cornell University

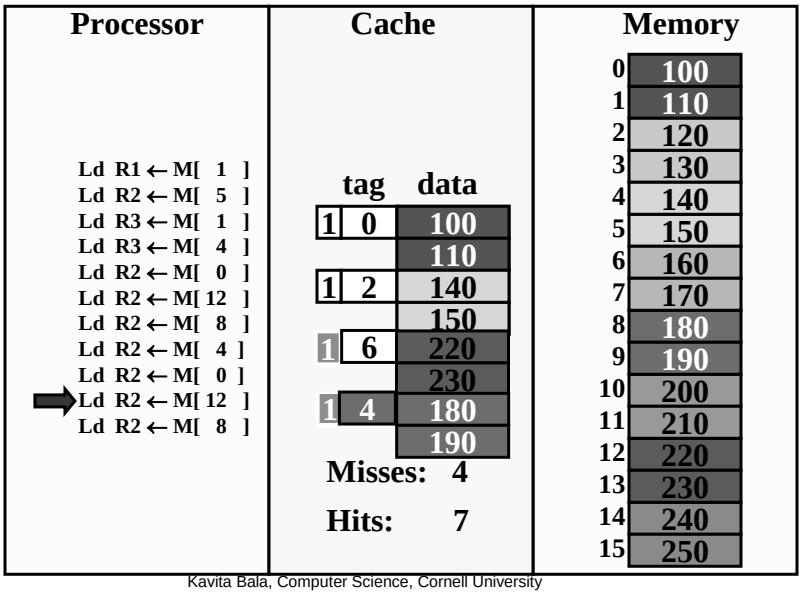
7th Access



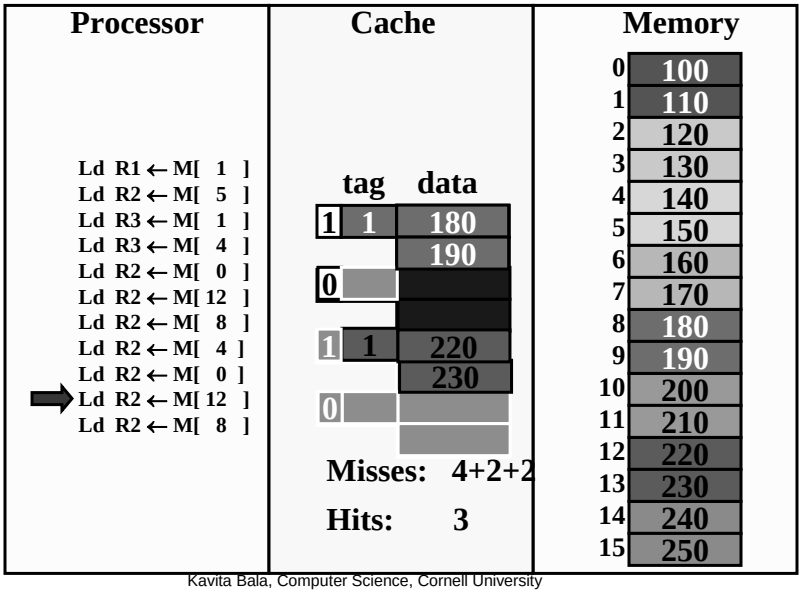
8th and 9th Access



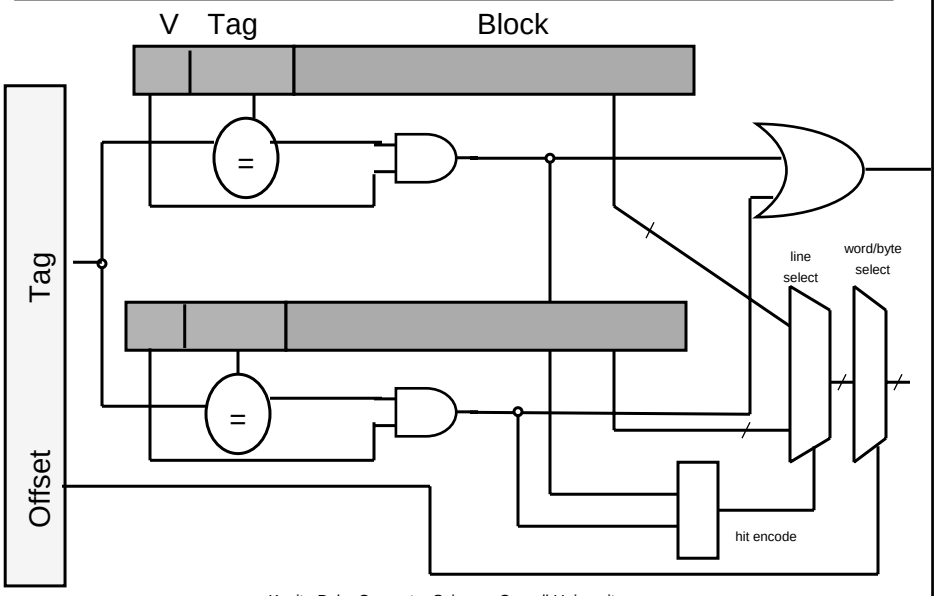
10th and 11th Access



DM: 10th and 11th Access



Fully Associative Cache

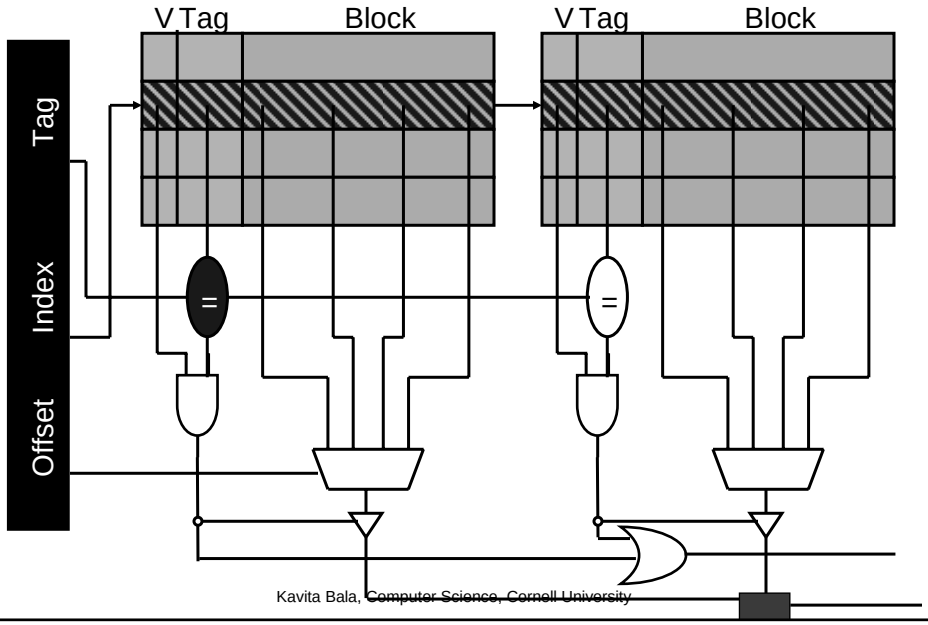


Cache Size

- 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)

2-Way Set-Associative Cache



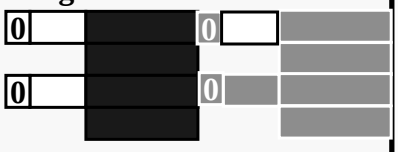
Kavita Bala, Computer Science, Cornell University

2 Way Set Assoc: 10th and 11th Access

Processor

Ld R1 ← M[1]
Ld R2 ← M[5]
Ld R3 ← M[1]
Ld R3 ← M[4]
Ld R2 ← M[0]
Ld R2 ← M[12]
Ld R2 ← M[8]
Ld R2 ← M[4]
Ld R2 ← M[0]
Ld R2 ← M[12]
Ld R2 ← M[8]

Cache



Misses: 7

Hits: 4

Memory

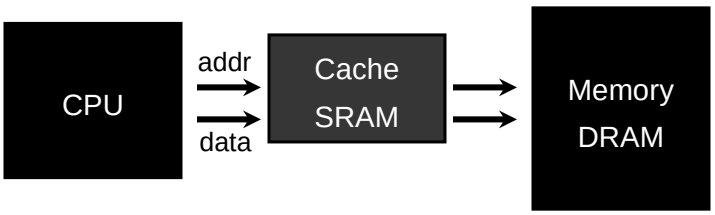
0	100
1	110
2	120
3	130
4	140
5	150
6	160
7	170
8	180
9	190
10	200
11	210
12	220
13	230
14	240
15	250

Kavita Bala, Computer Science, Cornell University

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

Cache Writes

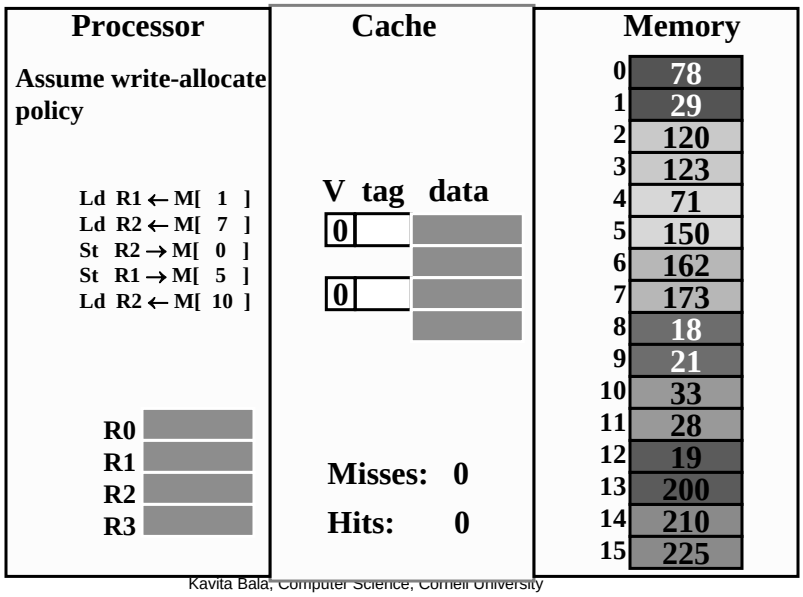


- No-Write
  - writes invalidate the cache and go to memory
- Write-Through
  - writes go to cache and to main memory
- Write-Back
  - write cache, write main memory only when block is evicted

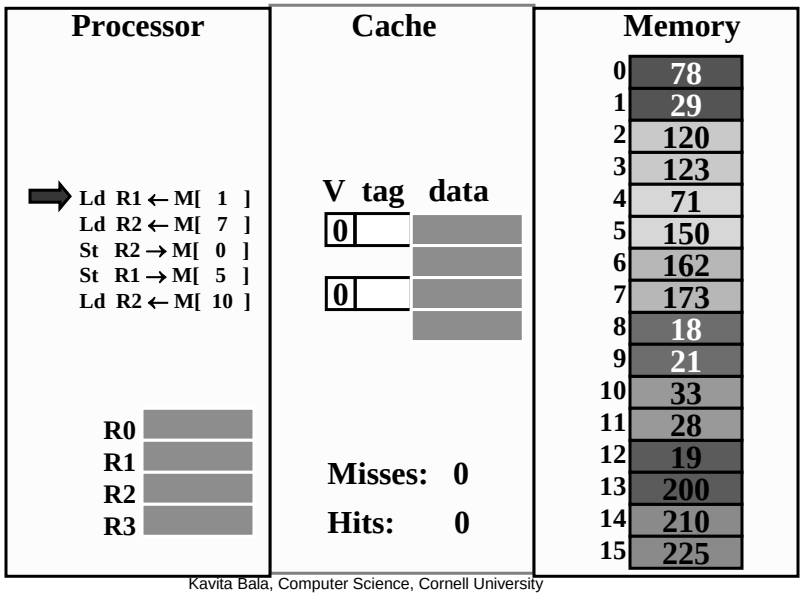
What about Stores?

- Where should you write the result of a store?
  - If that memory location is in the cache?
    - Send it to the cache
    - Should we also send it to memory right away? (write-through policy)
    - Wait until we kick the block out (write-back policy)
  - If it is not in the cache?
    - Allocate the line (put it in the cache)? (write allocate policy)
    - Write it directly to memory without allocation? (no write allocate policy)

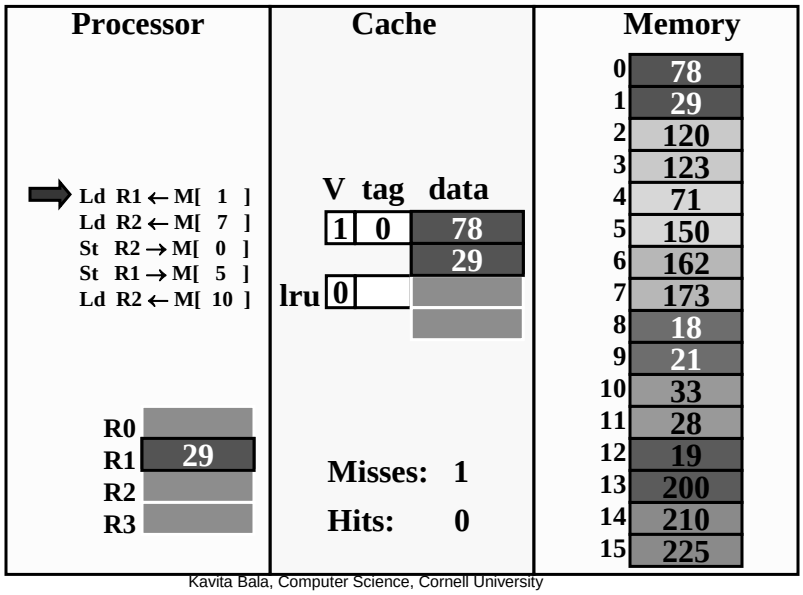
Handling Stores (Write-Through)



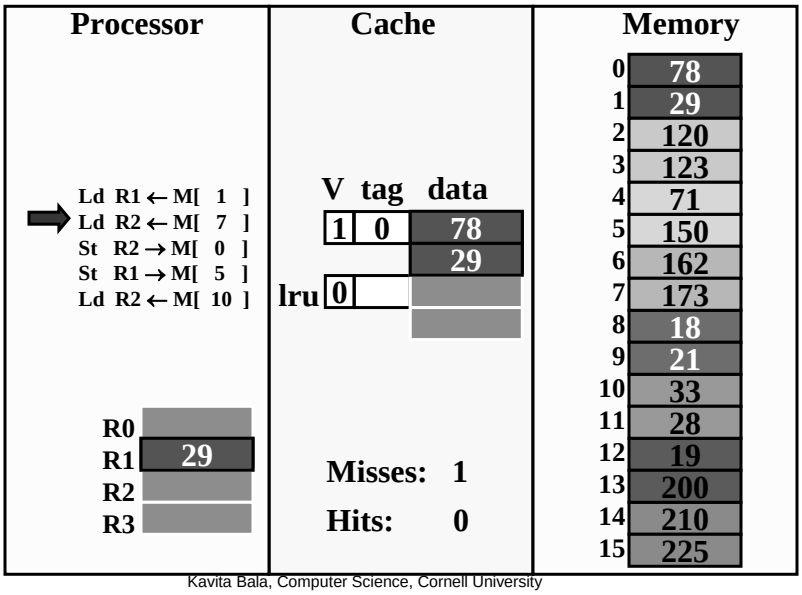
Write-Through (REF 1)



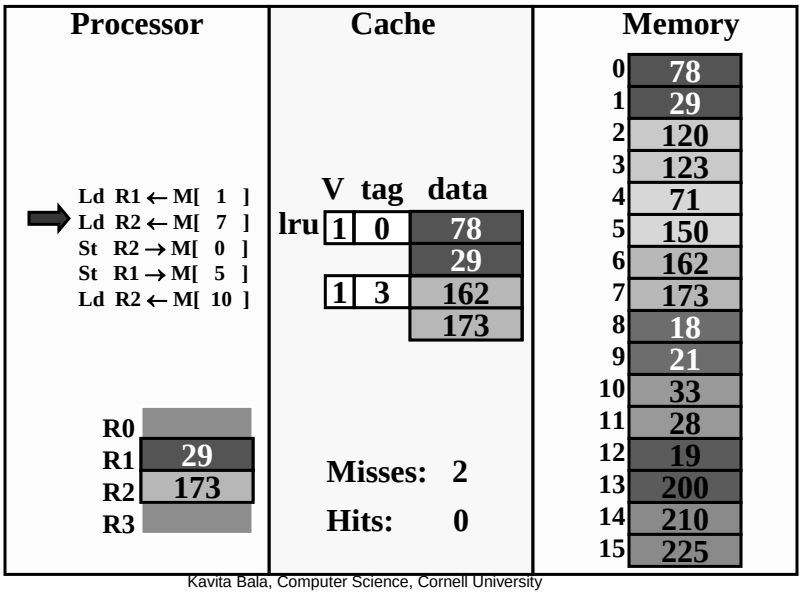
Write-Through (REF 1)



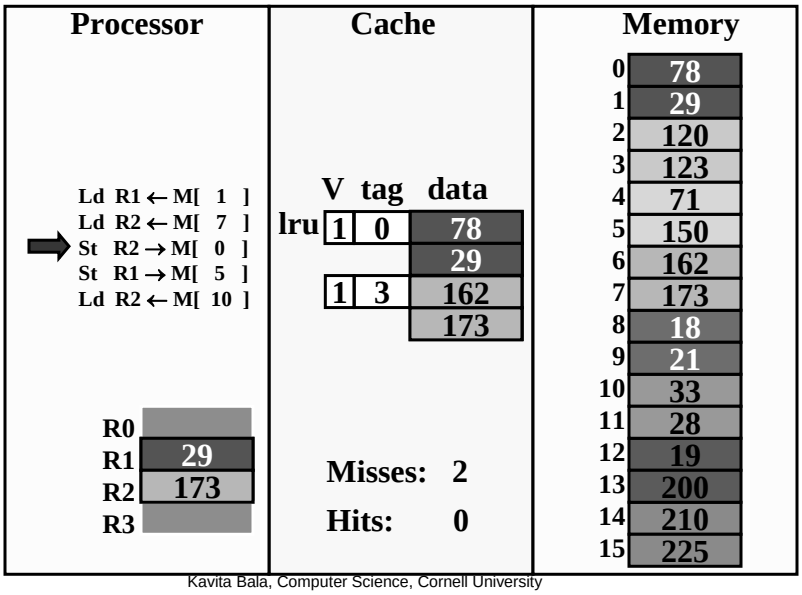
Write-Through (REF 2)



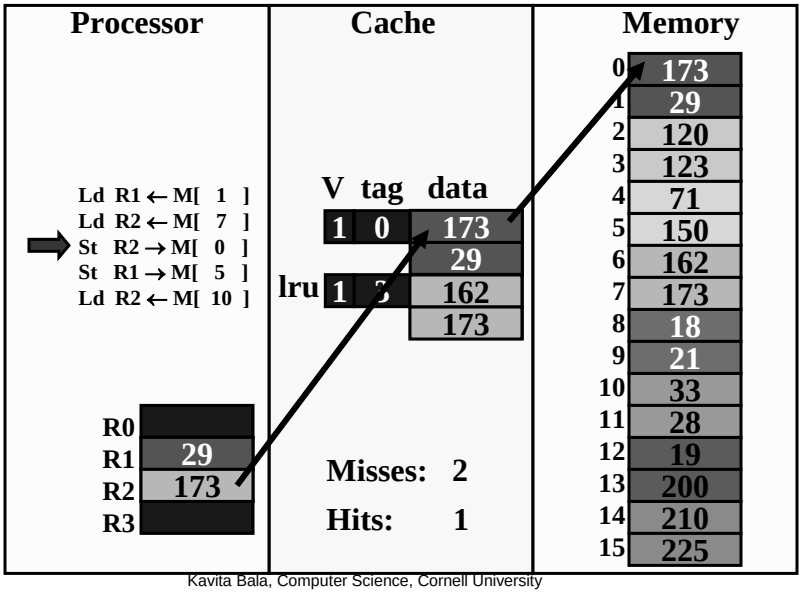
Write-Through (REF 2)



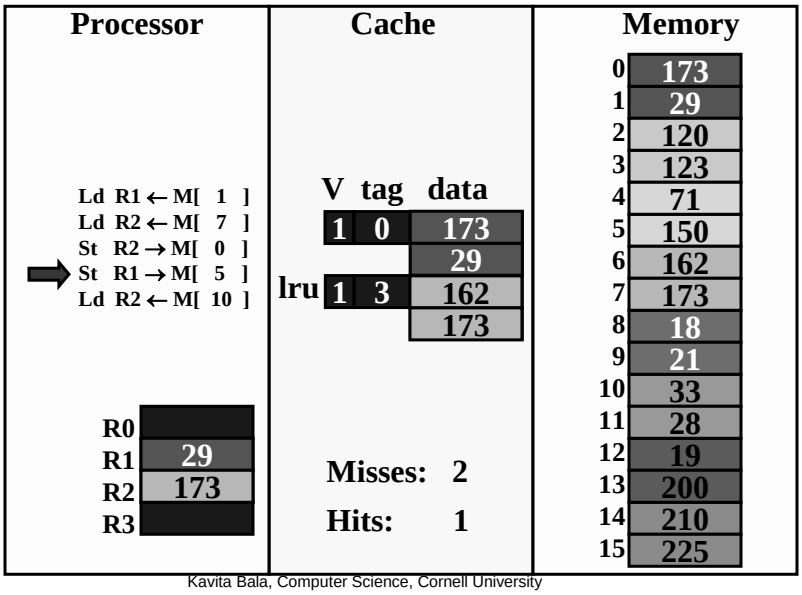
Write-Through (REF 3)



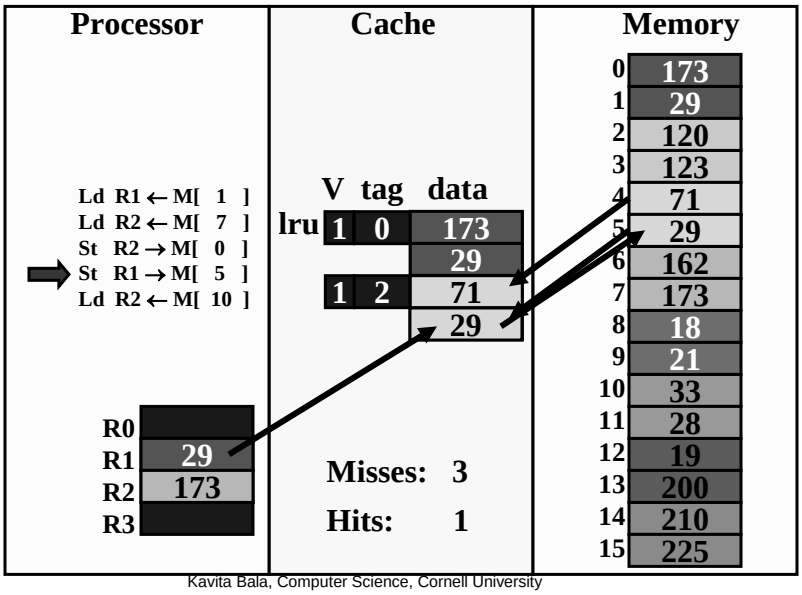
Write-Through (REF 3)



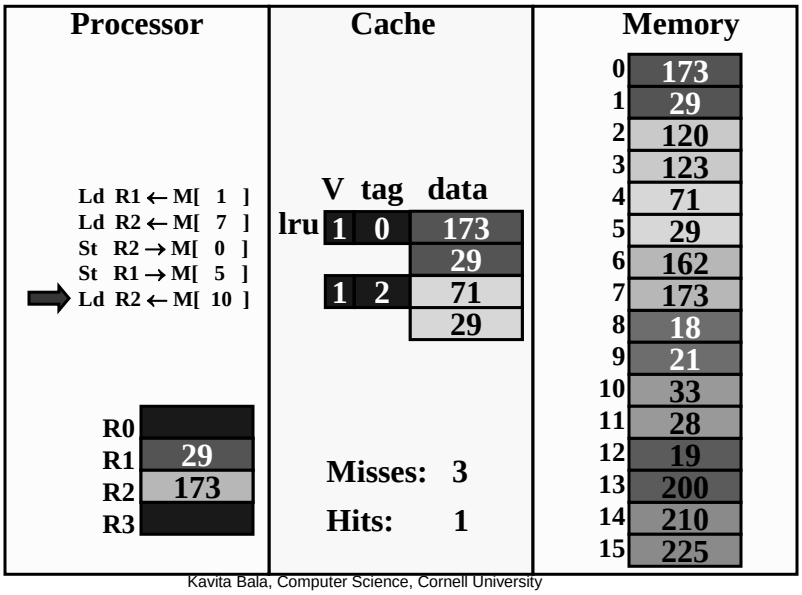
Write-Through (REF 4)



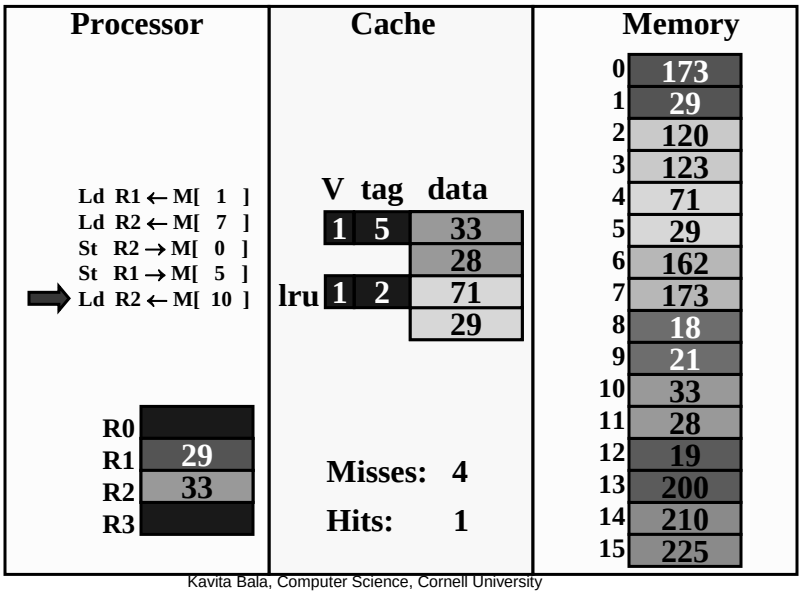
Write-Through (REF 4)



Write-Through (REF 5)



Write-Through (REF 5)



How Many Memory References?

- Each miss reads a block (only two words in this cache)
- Each store writes a word
- Total reads: eight words
- Total writes: four words

but caches generally miss < 20%
usually much lower miss rates . . . but depends on both cache and application!