CS 316:
Memory and Caches

Kavita Bala
Fall 2007
Computer Science
Cornell University

Announcements

- Prelim graded
  - Average: 69.8
  - Standard Deviation: 16
  - Handed out after class
  - Discussed in recitation
- HW 2 is out
- Caches and Memory: Chapter 7 (H & P)

Memory So Far

Big array of storage with more complex indexing than registers

- Addressing modes help us access memory
- A[i]; use base (i) + displacement (A)
- Use less space in instruction than 32-bit immediate field

SRAM vs. DRAM

- SRAM (static random access memory)
  - Faster than DRAM
  - Each storage cell is larger (4-6 transistors)
  - So smaller capacity for same area
  - 2-10ns access time
- DRAM (dynamic random access memory)
  - Each storage cell tiny (capacitance on wire)
  - Can get 1-2GB chips
  - 50-150ns access time
  - Leaky–needs to refresh data periodically

Dynamic RAM: DRAM

- Dynamic-RAM
  - Data values require constant refresh
  - Internal circuitry keeps capacitor charges

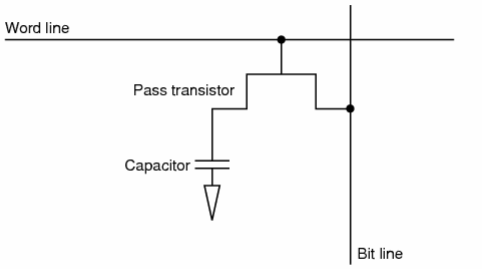


FIGURE B.9.5 A single-transistor DRAM cell contains a capacitor that stores the cell contents and a transistor used to access the cell.

Performance

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

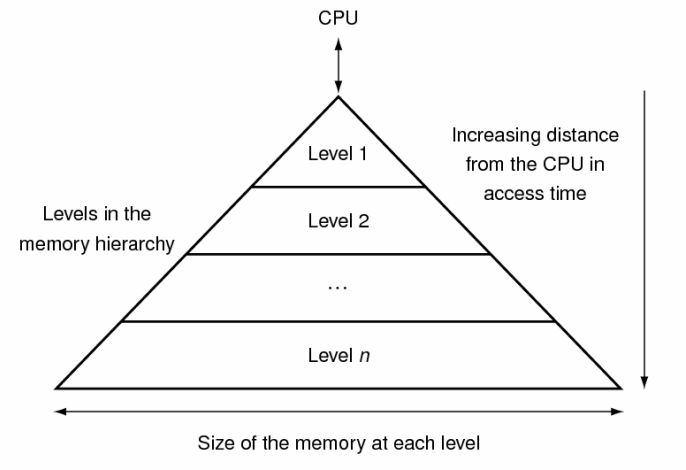
Technology	Capacity	Cost/GB	Latency
Tape	1 TB	$.17	100s
Disk	300 GB	$.34	Millions cycles (ms)
DRAM	4GB	$100s	50-300 cycles (10s of ns)
SRAM off	512KB	$4-10'sk	5-15 cycles (few ns)
SRAM on	16 KB	???	1-3 cycles (ns)

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

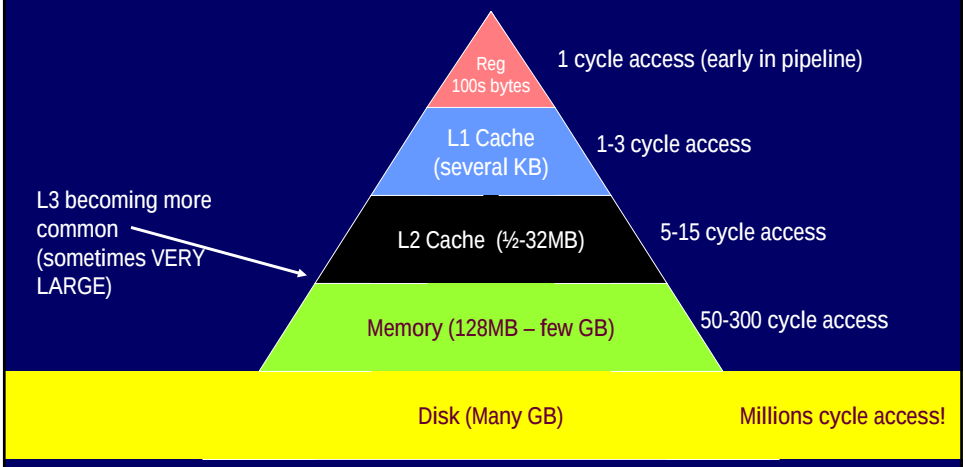
- How do we create the illusion of large and fast memory?

Memory Hierarchy

- Idea: Hide latency using small, fast memories called caches



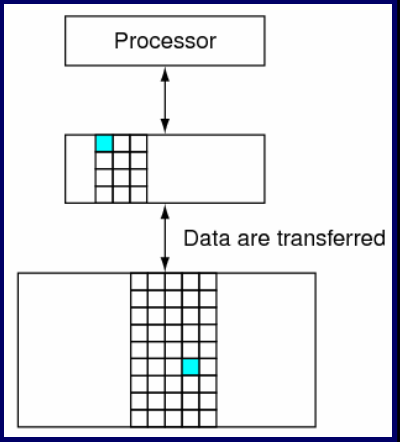
Cache Design 101



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

Memory Hierarchy

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

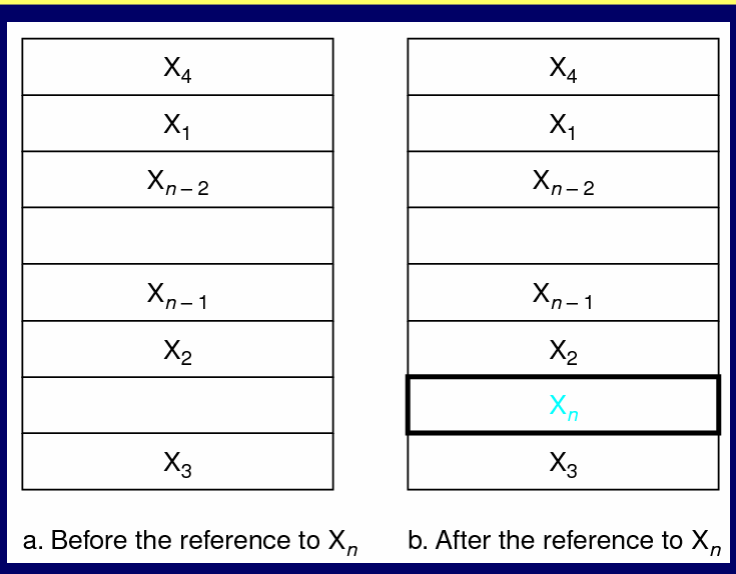


Insight of Caches

- 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



a. Before the reference to X_n b. After the reference to X_n

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

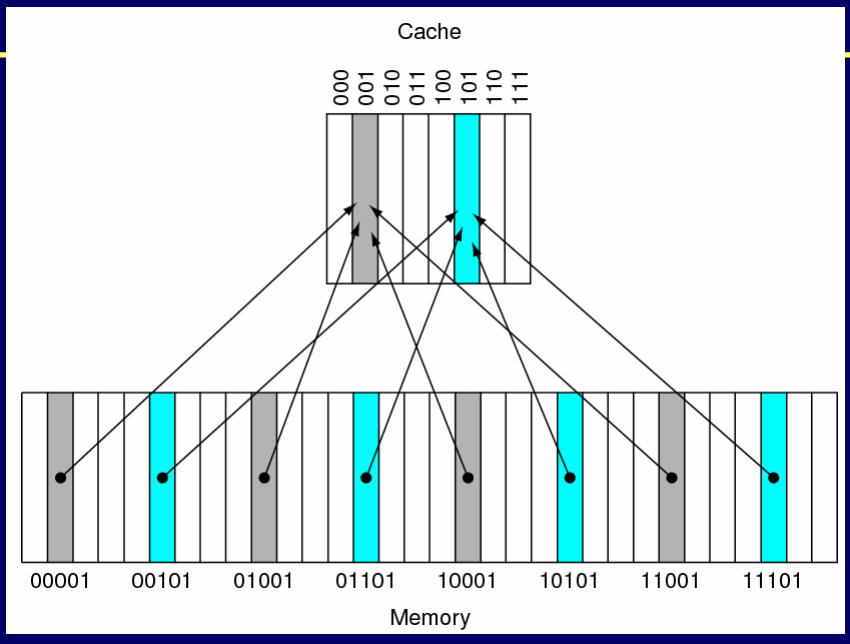
Cache Organization

- 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

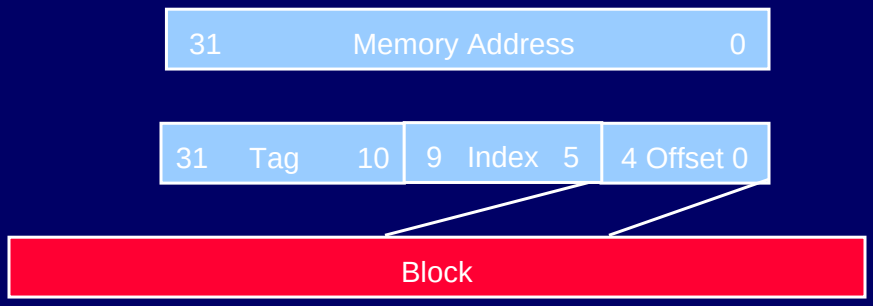
- Simplest
- Block can only be in one line in the cache

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



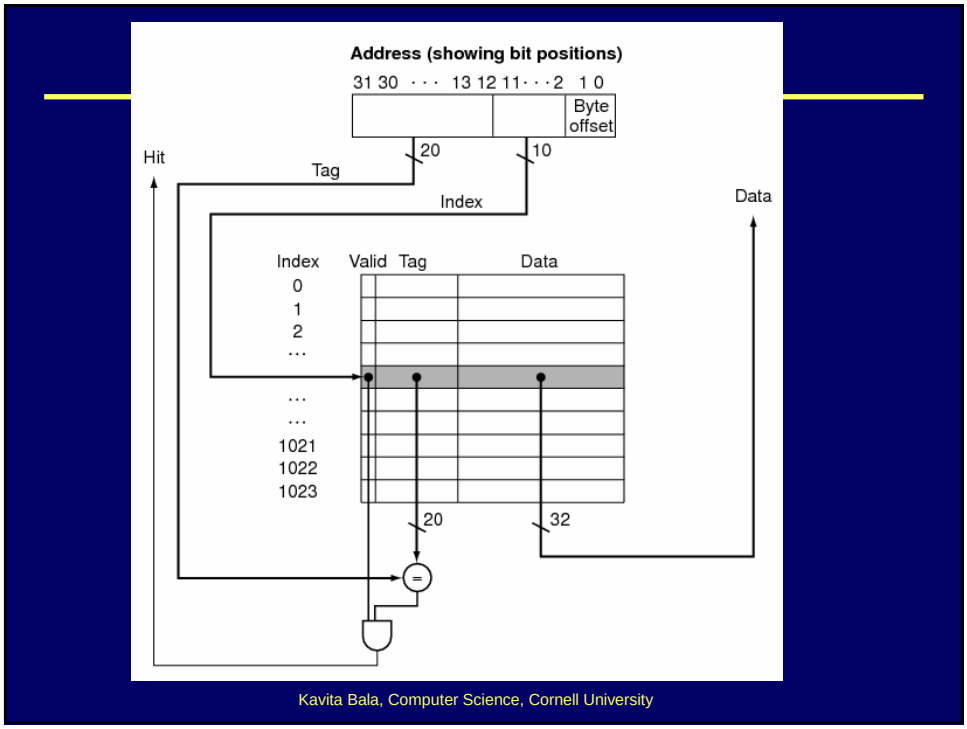
Tags and Offsets

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



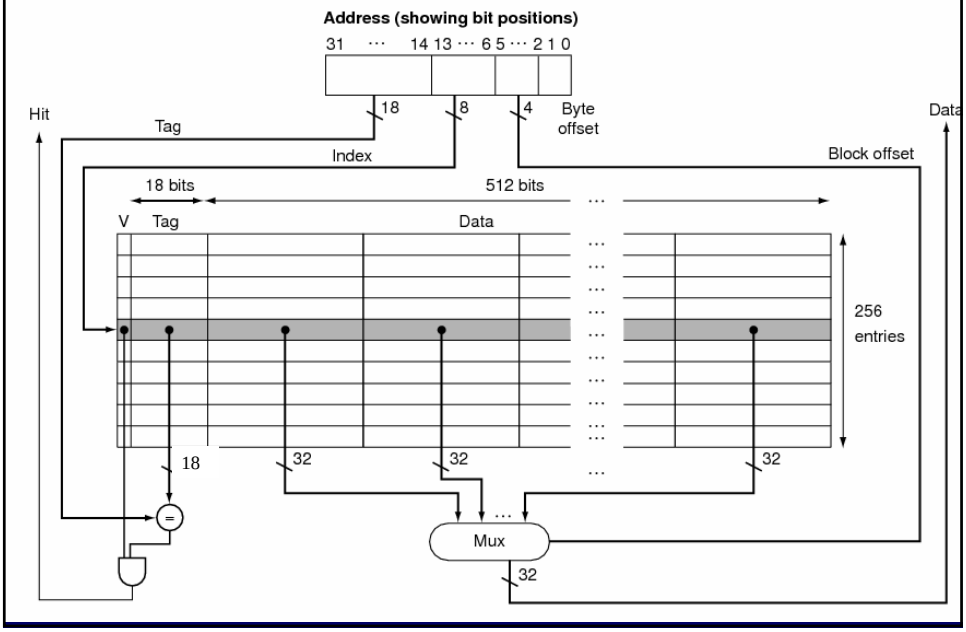
Valid Bits

- 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



Kavita Bala, Computer Science, Cornell University

Intrinsity FastMath

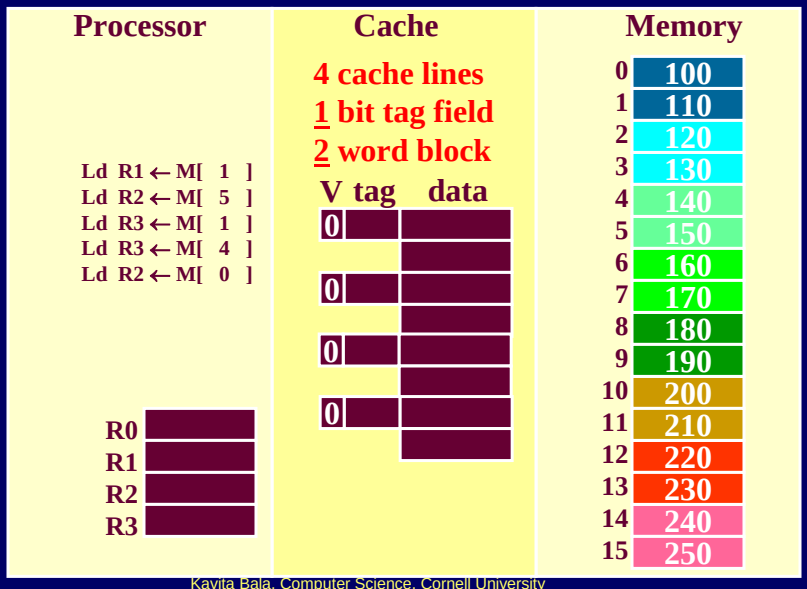


Cache Size

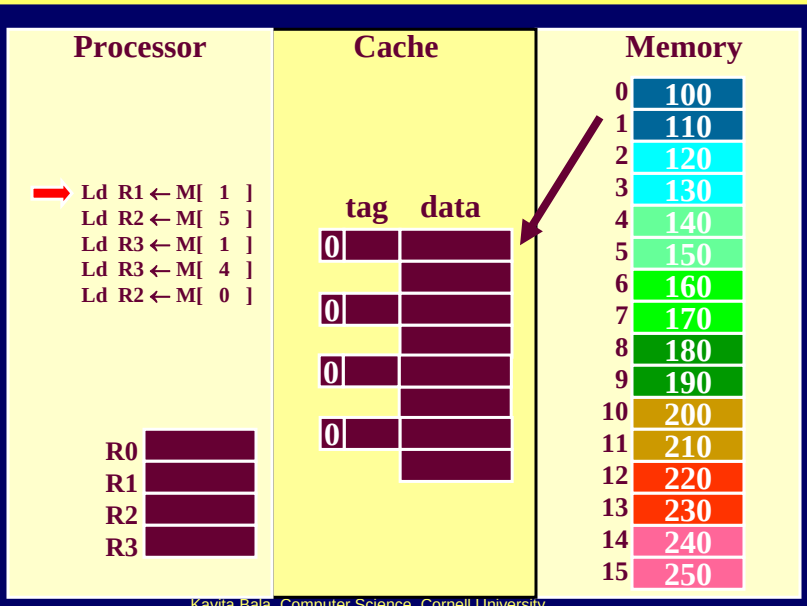
- Cache of size 2^n blocks (index: n bits)
- Block size of 2^m word (block index: m+2 bits)
- Tag field: 32 – (n + m + 2)
- Valid bit: 1

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

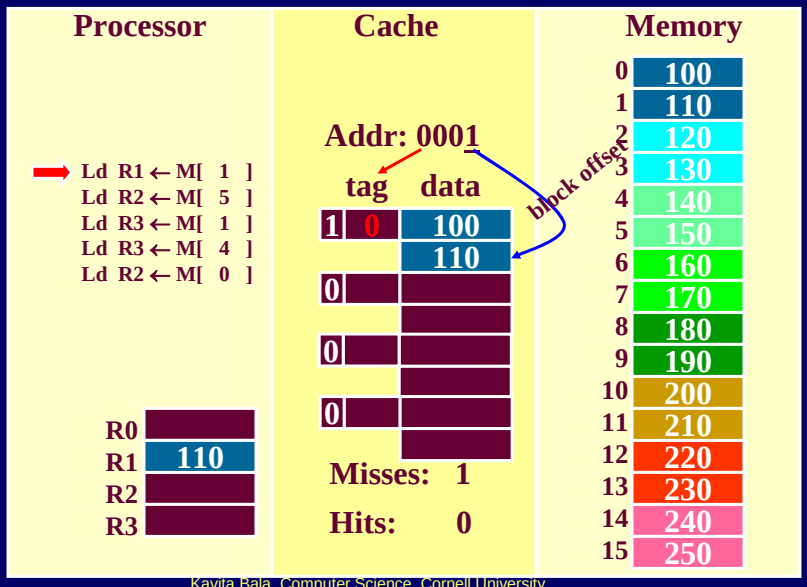
A Simple Direct Mapped Cache



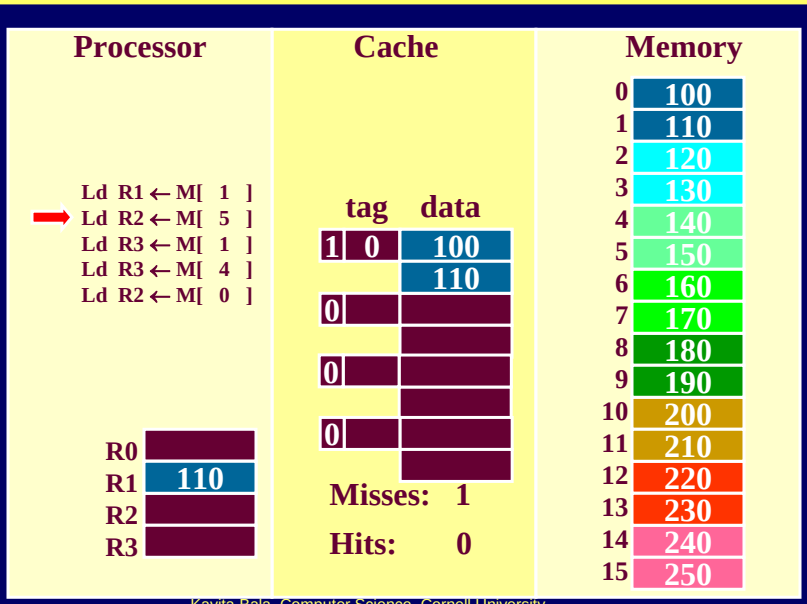
1st Access



1st Access

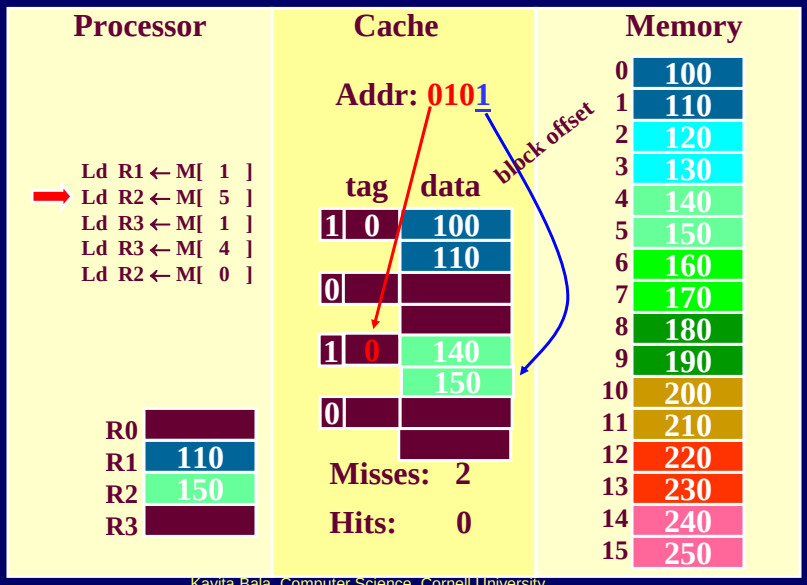


2nd Access



Kavita Bala, Computer Science, Cornell University

2nd Access



Kavita Bala, Computer Science, Cornell University

3rd Access

Processor

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

R0	
R1	110
R2	150
R3	

Cache

Addr: 0001

	tag	data
1	0	100
		110
0		
1	0	140
		150
0		

Misses: 2

Hits: 0

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

3rd Access

Processor

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

R0	
R1	110
R2	150
R3	110

Cache

	tag	data
1	0	100
		110
0		
1	0	140
		150
0		

Misses: 2

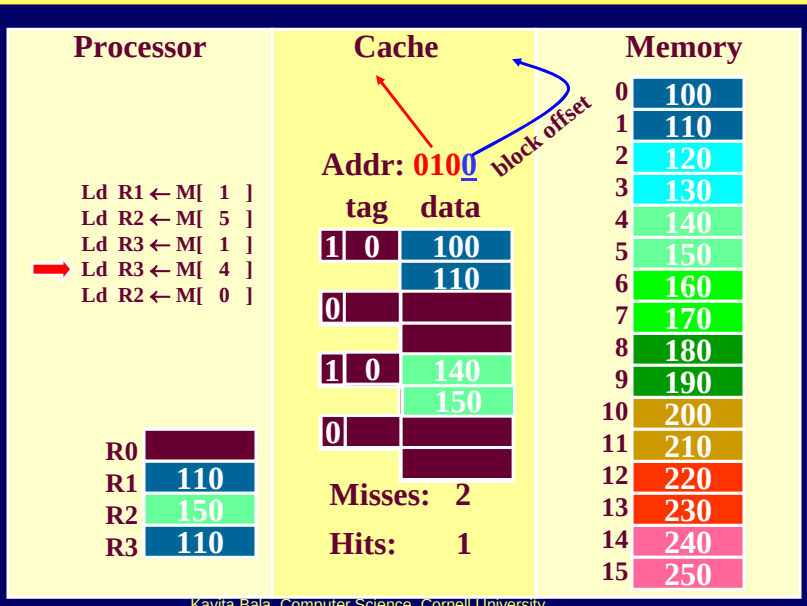
Hits: 1

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

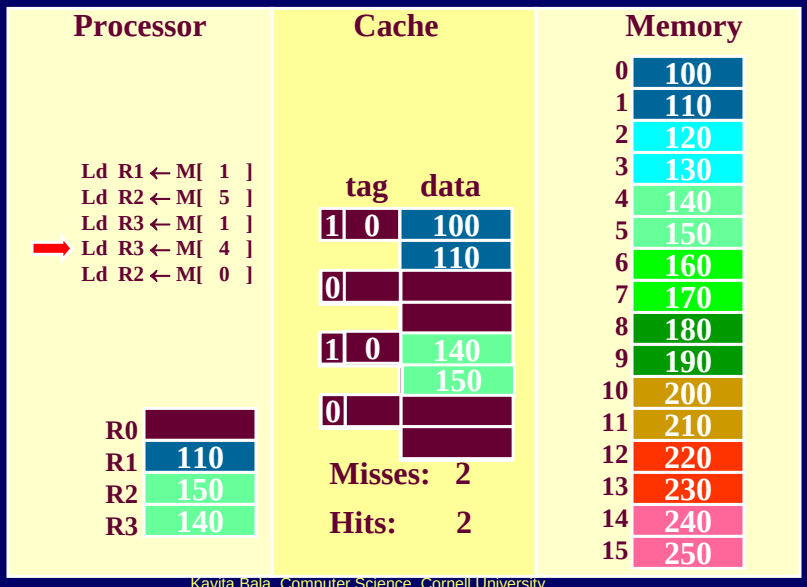
Kavita Bala, Computer Science, Cornell University

4th Access



Kavita Bala, Computer Science, Cornell University

4th Access



Kavita Bala, Computer Science, Cornell University

5th Access

Processor

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

R0	
R1	110
R2	150
R3	140

Cache

Addr: 0000

	tag	data
1	0	100
		110
0		
1	0	140
		150
0		

Misses: 2

Hits: 2

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

5th Access

Processor

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

R0	
R1	110
R2	100
R3	140

Cache

Addr: 0000

	tag	data
1	0	100
		110
0		
1	0	140
		150
0		

Misses: 2

Hits: 3

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

5th 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[10]
Ld R2 ← M[15]

R0	
R1	110
R2	100
R3	140

Cache

Addr: 1010

	tag	data
1	0	100
		110
0		
1	0	140
		150
0		

Misses: 2

Hits: 3

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

5th 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[10]
Ld R2 ← M[15]

R0	
R1	110
R2	200
R3	140

Cache

Addr: 1010

	tag	data
1	0	100
		110
1	1	200
		210
1	0	140
		150
0		

Misses: 3

Hits: 3

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

6th 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[10]
➡ Ld R2 ← M[15]

R0	
R1	110
R2	200
R3	140

Cache

Addr: 1111

	tag	data
1	0	100
		110
1	1	200
		210
1	0	140
		150
0		

Misses: 3

Hits: 3

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

6th 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[10]
➡ Ld R2 ← M[15]

R0	
R1	110
R2	250
R3	140

Cache

Addr: 1111

	tag	data
1	0	100
		110
1	1	200
		210
1	0	140
		150
1	1	240
		250

Misses: 4

Hits: 3

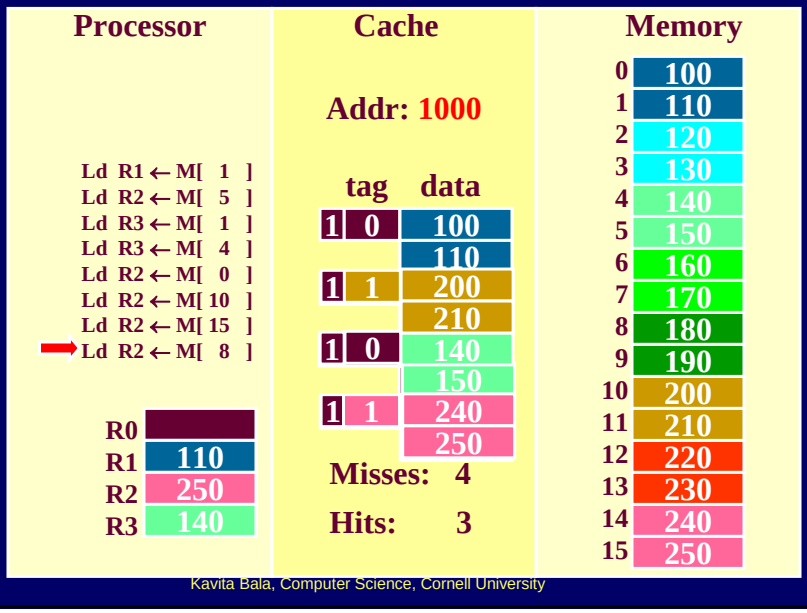
Memory

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

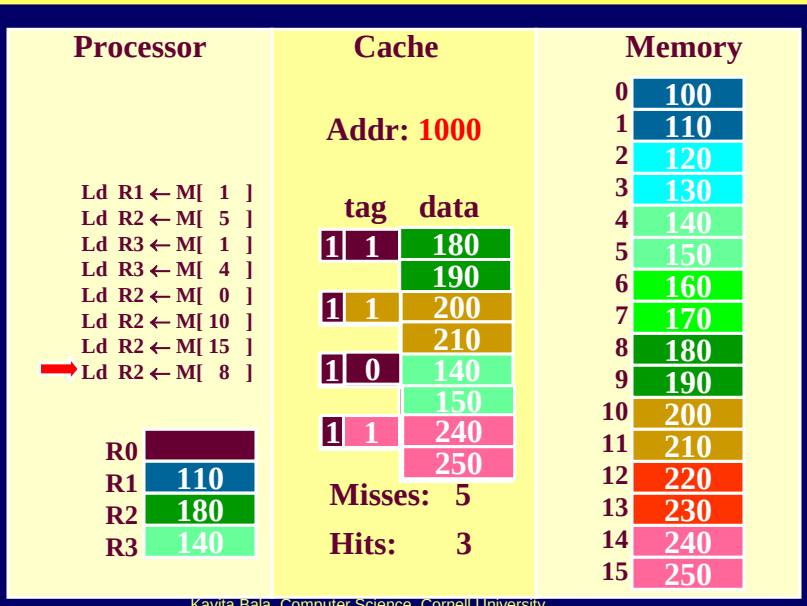
Cache

- Exploiting Spatial and Temporal Locality

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

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]

R0	
R1	110
R2	100
R3	140

Cache

Addr: 1100

	tag	data
1	0	100
		110
0		
1	0	140
		150
0		

Misses: 2

Hits: 3

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

6th 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]

R0	
R1	110
R2	220
R3	140

Cache

Addr: 1100

	tag	data
1	0	100
		110
0		
1	1	220
		230
0		

Misses: 3

Hits: 3

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

7th 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]

R0	
R1	110
R2	220
R3	140

Cache

Addr: 1000

	tag	data
1	0	100
		110
0		
1	1	220
		230
0		

Misses: 3

Hits: 3

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

7th 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]

R0	
R1	180
R2	220
R3	140

Cache

Addr: 1000

	tag	data
1	1	180
		190
0		
1	1	220
		230
0		

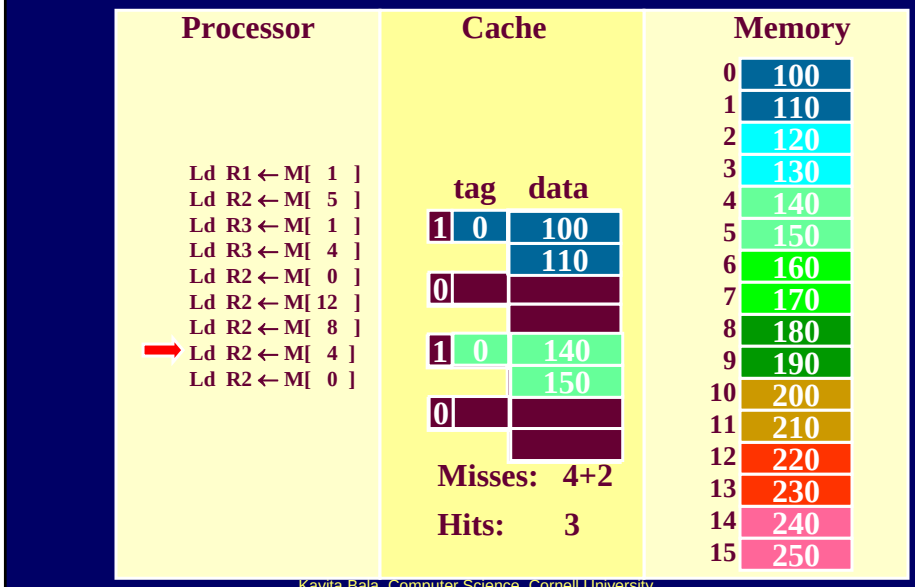
Misses: 4

Hits: 3

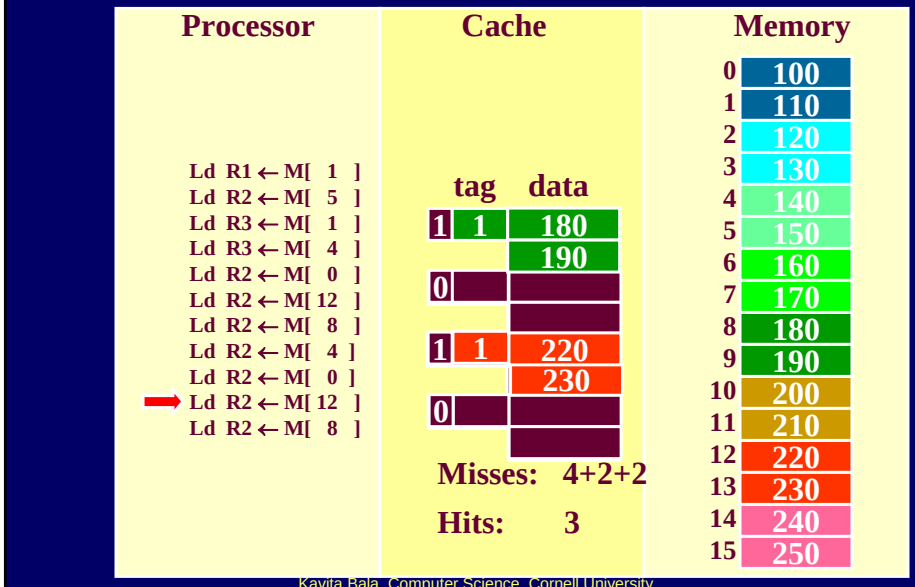
Memory

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

8th and 9th Access



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