ECE 2300
Digital Logic & Computer Organization

Spring 2025

More Caches

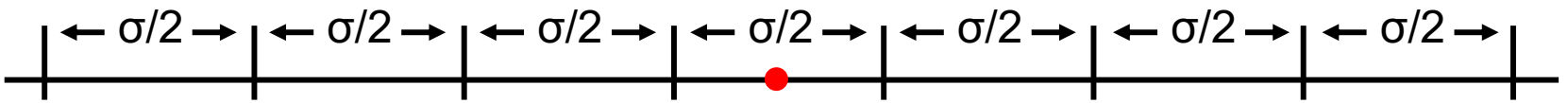
Cornell University

Announcements

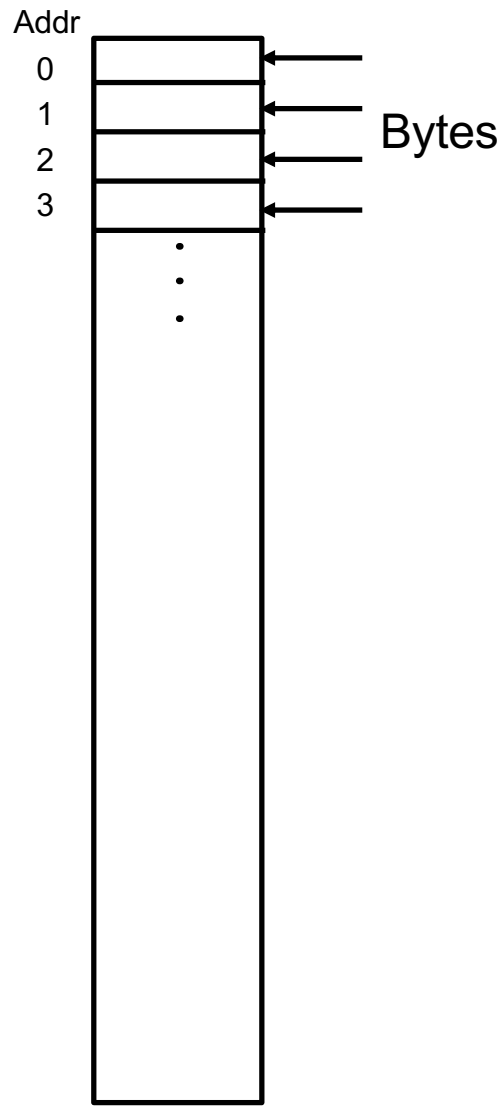
- **Lab4b due tomorrow**

- **Final exam is scheduled on May 10 at 9am, PHL 101**
  - **Cumulative, with an emphasis on material from Lecture 17 onward**
  - **If you have a schedule conflict, inform the instructor via email ASAP (by Friday 4/25 at the latest)**

- **Final grade determination**
  - **Median (●)**
  - **Deviation (σ)**

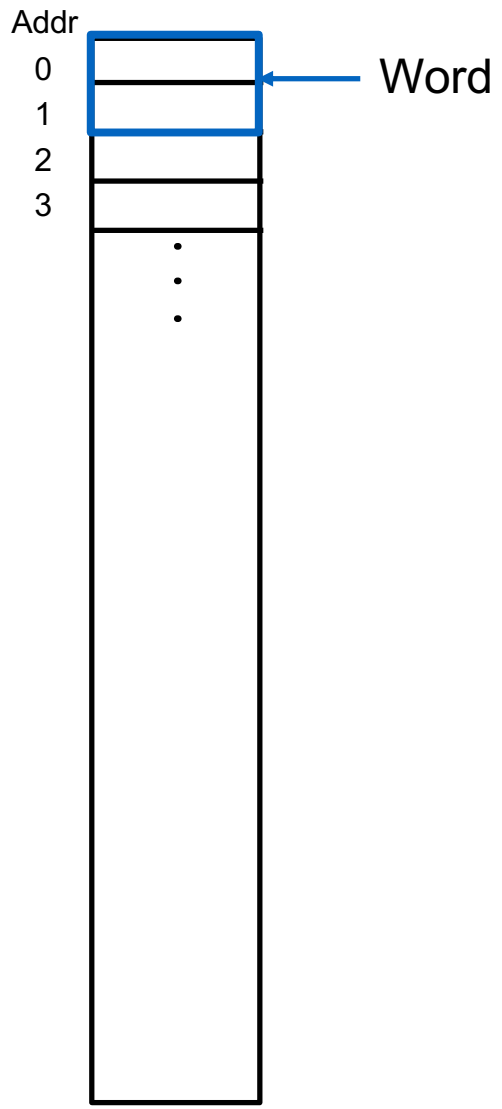


Review: Main Memory Organization



- **Assumptions**
  - **Byte addressable**

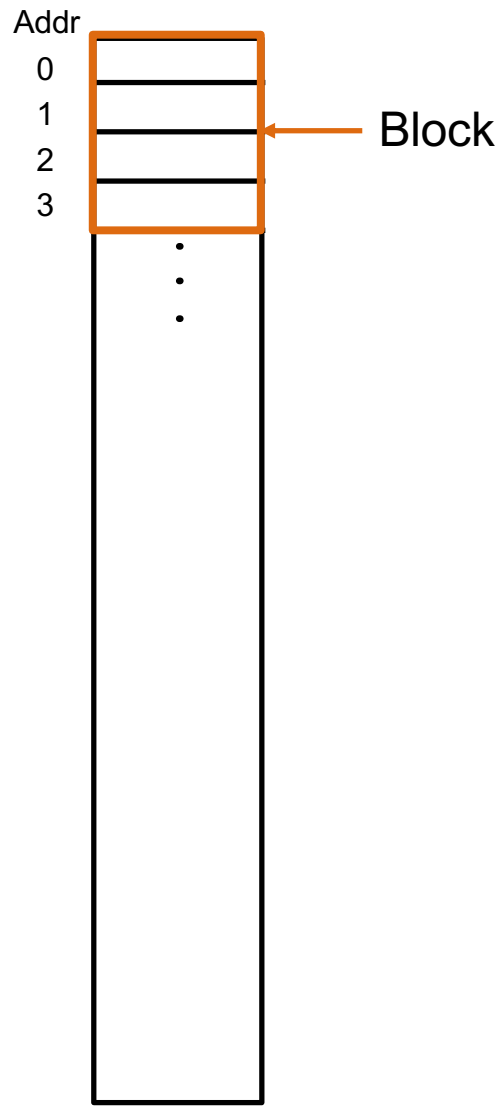
Review: Main Memory Organization



- **Assumptions**
  - **Byte addressable**
  - **2 bytes per word**

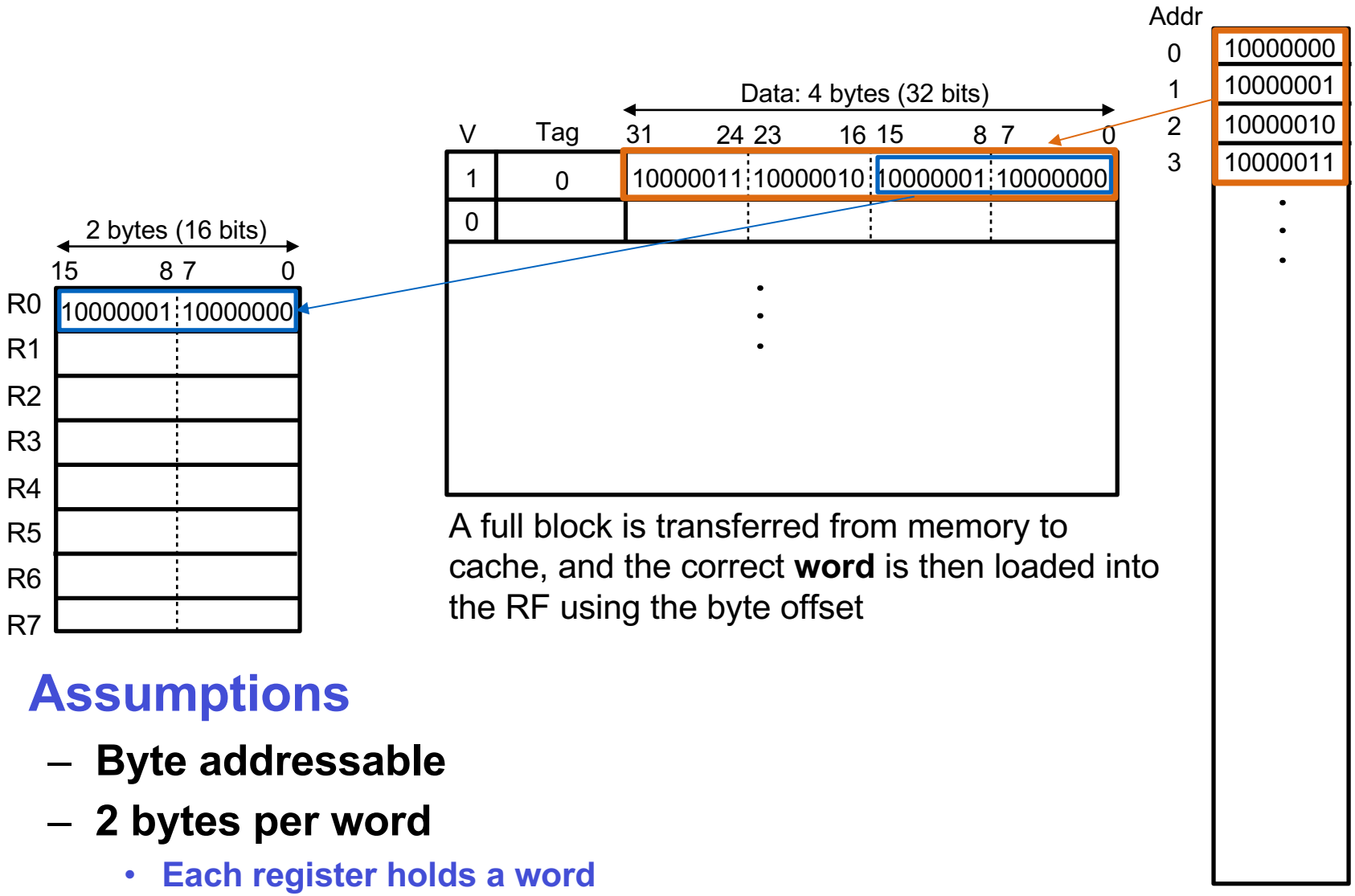
Review: Main Memory Organization

A full block is transferred between main memory and cache



- **Assumptions**
  - **Byte addressable**
  - **2 bytes per word**
  - **4 bytes per block**

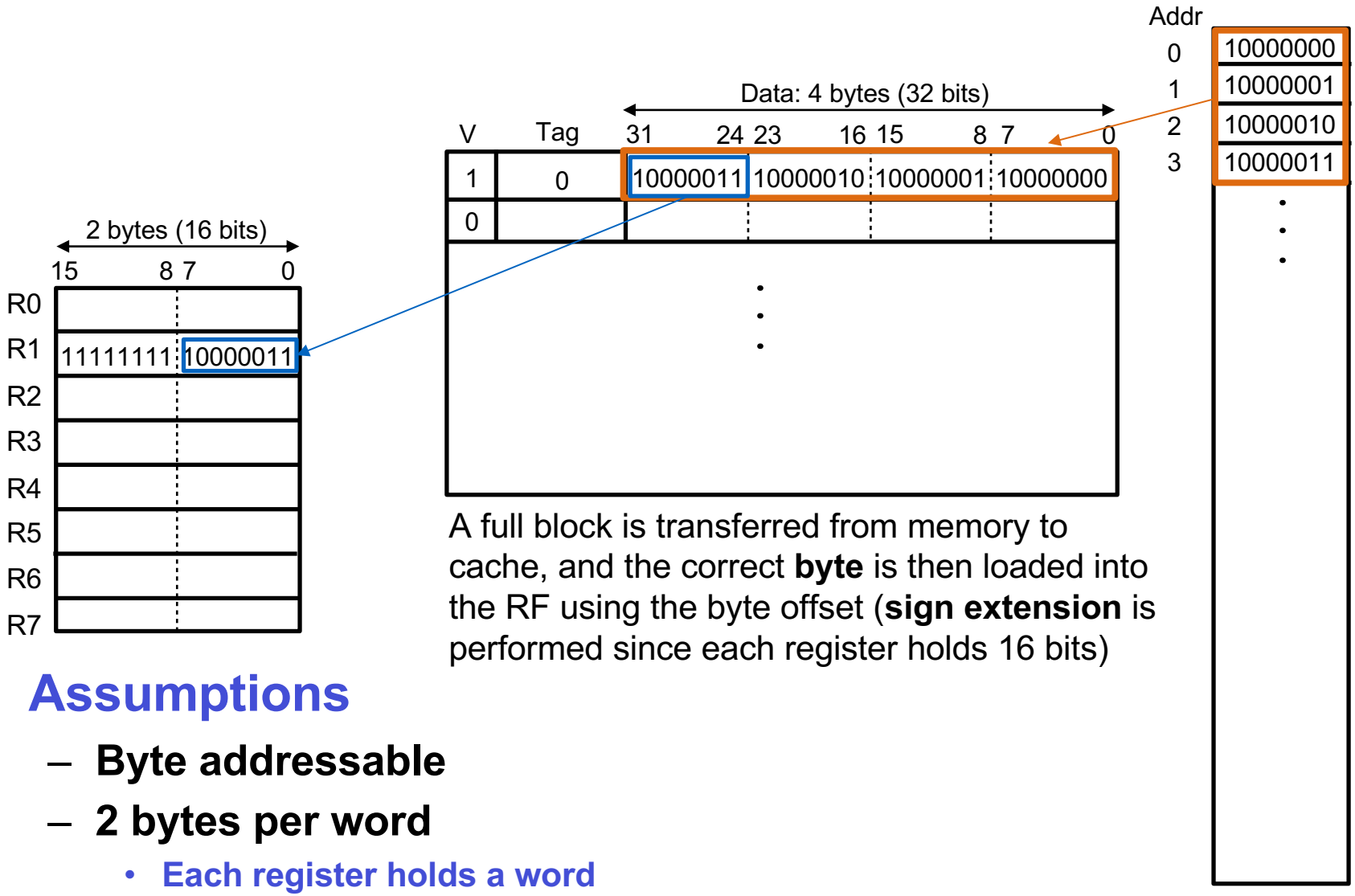
Load Word (LW) from Addr 0 to R0



A full block is transferred from memory to cache, and the correct **word** is then loaded into the RF using the byte offset

- **Assumptions**
  - **Byte addressable**
  - **2 bytes per word**
    - **Each register holds a word**
  - **4 bytes per block**

Load Byte (LB) from Addr 3 to R1



A full block is transferred from memory to cache, and the correct **byte** is then loaded into the RF using the byte offset (**sign extension** is performed since each register holds 16 bits)

- **Assumptions**
  - **Byte addressable**
  - **2 bytes per word**
    - **Each register holds a word**
  - **4 bytes per block**

Cache Basics (True or False)

- Together, the tag and index bits form the memory block address
- DM cache is a special case of set associative cache
- Accessing fully associative cache does not require tag bits

Review: Spectrum of Associativity

- A *K*-way set associative cache with *N* blocks
  - Number of cache sets $S = N / K$
    - Number of index bits = $\log_2(S)$
  - When $K = N$, fully associative cache
    - ONE cache set → zero index bits
  - When $K = 1$ (one-way), direct mapped cache
    - *N* cache sets

- Increasing the associativity
  - Typically improves the hit rate (fewer conflicts)
  - But increases the hit time (takes longer to search)

Example: Cache Address Breakdown

- Assuming 16-bit memory addresses, how many bits are associated with the tag, index, and offset of the following configurations?

- 16 blocks, 16B per block, 2-way set associative

Byte offset: 4 bits; Index: 3 bits; Tag: 9 bits

Understanding Cache Misses

- Reducing cache misses improves overall system performance
- Understanding miss types and causes aids in selecting suitable cache parameters
- Tailored hardware & software optimization techniques can be developed for each miss type

Miss Classification

- **Compulsory (Cold) misses**
  - Caused by the first access to a memory block
- **Capacity misses**
  - Occur because the cache is not large enough to hold the active set of memory blocks needed during program execution
- **Conflict misses**
  - Occur due to inflexibility of block placement, where multiple memory blocks contend for placement within the same cache set
  - *Never occur in a fully associative cache*

Misses vs. Associativity Example

- **Compare different caches**
  - **Capacity: 4 blocks**
  - **Direct mapped, 2-way set associative, fully associative**
  - **Block address sequence: 0, 8, 0, 6, 8 (in decimal)**
    - Note that a memory block address does not include byte offset bits
- **Direct mapped**

Time

Block address	Cache index	Hit/miss	Cache contents after access			
			Block 0	Block 1	Block 2	Block 3
0						
8						
0						
6						
8						

Four blocks

Misses vs. Associativity Example

- **Compare different caches**
  - **Capacity: 4 blocks**
  - **Direct mapped, 2-way set associative, fully associative**
  - **Block address sequence: 0, 8, 0, 6, 8 (in decimal)**
    - Note that a memory block address does not include byte offset bits

- **Direct mapped**

Time

Block address	Cache index	Hit/miss	Cache contents after access			
			Block 0	Block 1	Block 2	Block 3
0	0	miss	M[0]			
8	0	miss	M[8]			
0	0	miss	M[0]			
6	2	miss	M[0]		M[6]	
8	0	miss	M[8]		M[6]	

Color code: Cold miss Conflict miss

Misses vs. Associativity Example

- **2-way set associative**

Block address	Cache index	Hit/miss	Cache contents after access			
			Set 0		Set 1	
0						
8						
0						
6						
8						

Time

Two ways Two ways

- **Fully associative**

Block address		Hit/miss	Cache contents after access			
0						
8						
0						
6						
8						

Time

Misses vs. Associativity Example

- **2-way set associative**

Time ↓

Block address	Cache index	Hit/miss	Cache contents after access			
			Set 0		Set 1	
0	0	miss	M[0]			
8	0	miss	M[0]	M[8]		
0	0	hit	M[0]	M[8]		
6	0	miss	M[0]	M[6]		
8	0	miss	M[8]	M[6]		

- **Fully associative**

Time ↓

Block address	~~Cache index~~	Hit/miss	Cache contents after access			
0		miss	M[0]			
8		miss	M[0]	M[8]		
0		hit	M[0]	M[8]		
6		miss	M[0]	M[8]	M[6]	
8		hit	M[0]	M[8]	M[6]	

Color code: Cold miss Conflict miss

Block Replacement Policy

- Direct mapped: no choice
- Set associative and fully associative
  - Pick non-valid entry, if there is one
  - Otherwise, choose among entries in the set
- Least recently used (LRU)
  - Choose the one unused for the longest time
  - Requires extra bits to order the blocks
  - High overhead beyond 4-way set associative
- Random
  - Similar performance as LRU for high associativity

LRU Replacement Example

- **2-way set associative**

(*) = LRU block
1 bit per cache block here

Block address	Cache index	Hit/miss	Cache contents after access			
			Set 0		Set 1	
0	0	miss	M[0]			
4	0	miss	M[0] (*)	M[4]		
2	0	miss	M[2]	M[4] (*)		
6	0	miss	M[2] (*)	M[6]		
8	0	miss	M[8]	M[6] (*)		
0	0	miss	M[8] (*)	M[0]		
4	0					
2	0					
6	0					
8	0					
2	0					
6	0					
2	0					

Why is this considered a capacity miss rather than a conflict miss?

Color code: Cold miss Conflict miss Capacity miss

LRU Replacement Example (cont'd)

- **2-way set associative**

(*) = LRU block
1 bit per cache block here

Block address	Cache index	Hit/miss	Cache contents after access			
			Set 0		Set 1	
0	0	miss	M[0]			
4	0	miss	M[0] (*)	M[4]		
2	0	miss	M[2]	M[4] (*)		
6	0	miss	M[2] (*)	M[6]		
8	0	miss	M[8]	M[6] (*)		
0	0	miss	M[8] (*)	M[0]		
4	0	miss	M[4]	M[0] (*)		
2	0	miss	M[4] (*)	M[2]		
6	0	miss	M[6]	M[2] (*)		
8	0	miss	M[6] (*)	M[8]		
2	0	miss	M[2]	M[8] (*)		
6	0	miss	M[2] (*)	M[6]		
2	0	hit	M[2]	M[6] (*)		

Even employing *a fully associative cache with the same capacity* would not circumvent these misses; hence we classify them as capacity misses.

Color code: Cold miss Conflict miss Capacity miss

Another LRU Replacement Example

- **Fully associative**

(X) = LRU Age; 2 age bits per block in this case*
**Age saturates* at 3 (an approximation to reduce hardware complexity; multiple blocks can have the same age)

Block address	~~Cache index~~	Hit/miss	Cache contents after access			
0		miss	M[0] (0)			
4		miss	M[0] (1)	M[4] (0)		
2		miss	M[0] (2)	M[4] (1)	M[2] (0)	
6		miss	M[0] (3)	M[4] (2)	M[2] (1)	M[6] (0)
8		miss	M[8] (0)	M[4] (3)	M[2] (2)	M[6] (1)
0		miss	M[8] (1)	M[0] (0)	M[2] (3)	M[6] (2)
4		miss	M[8] (2)	M[0] (1)	M[4] (0)	M[6] (3)
2		miss	M[8] (3)	M[0] (2)	M[4] (1)	M[2] (0)
6		miss	M[6] (0)	M[0] (3)	M[4] (2)	M[2] (1)
8		miss	M[6] (1)	M[8] (0)	M[4] (3)	M[2] (2)
2		hit	M[6] (2)	M[8] (1)	M[4] (3)	M[2] (0)
6		hit	M[6] (0)	M[8] (2)	M[4] (3)	M[2] (1)
2		hit	M[6] (1)	M[8] (3)	M[4] (3)	M[2] (0)

Color code: Cold miss Conflict miss Capacity miss

What About Writes?

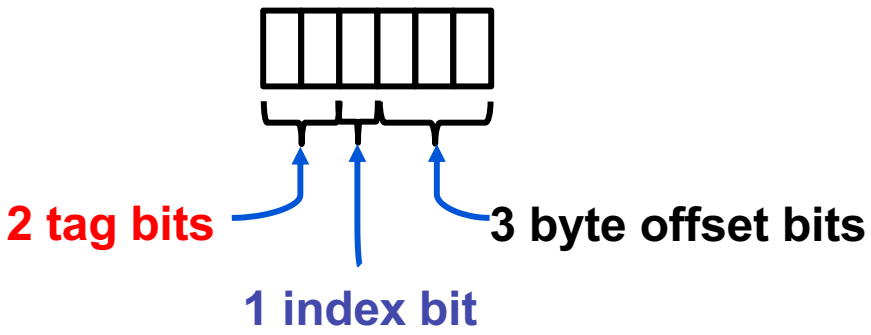
- Where do we put the result of a store?

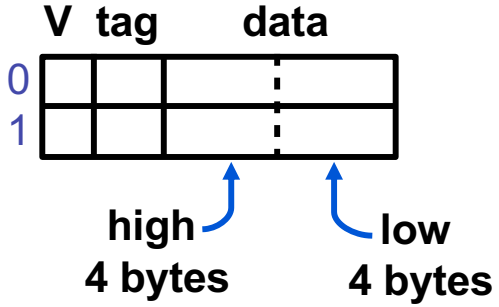
- Cache hit (block is in cache)
  - Write new data value to the cache
  - Also write to memory (write through)
  - Don't write to memory (write back)
    - Requires an additional *dirty bit* for each cache block
    - Writes back to memory when a dirty cache block is evicted

- Cache miss (block is not in cache)
  - Allocate the line (bring it into the cache) (write allocate)
  - Write to memory without allocation (no write allocate or write around)

Write Through Example

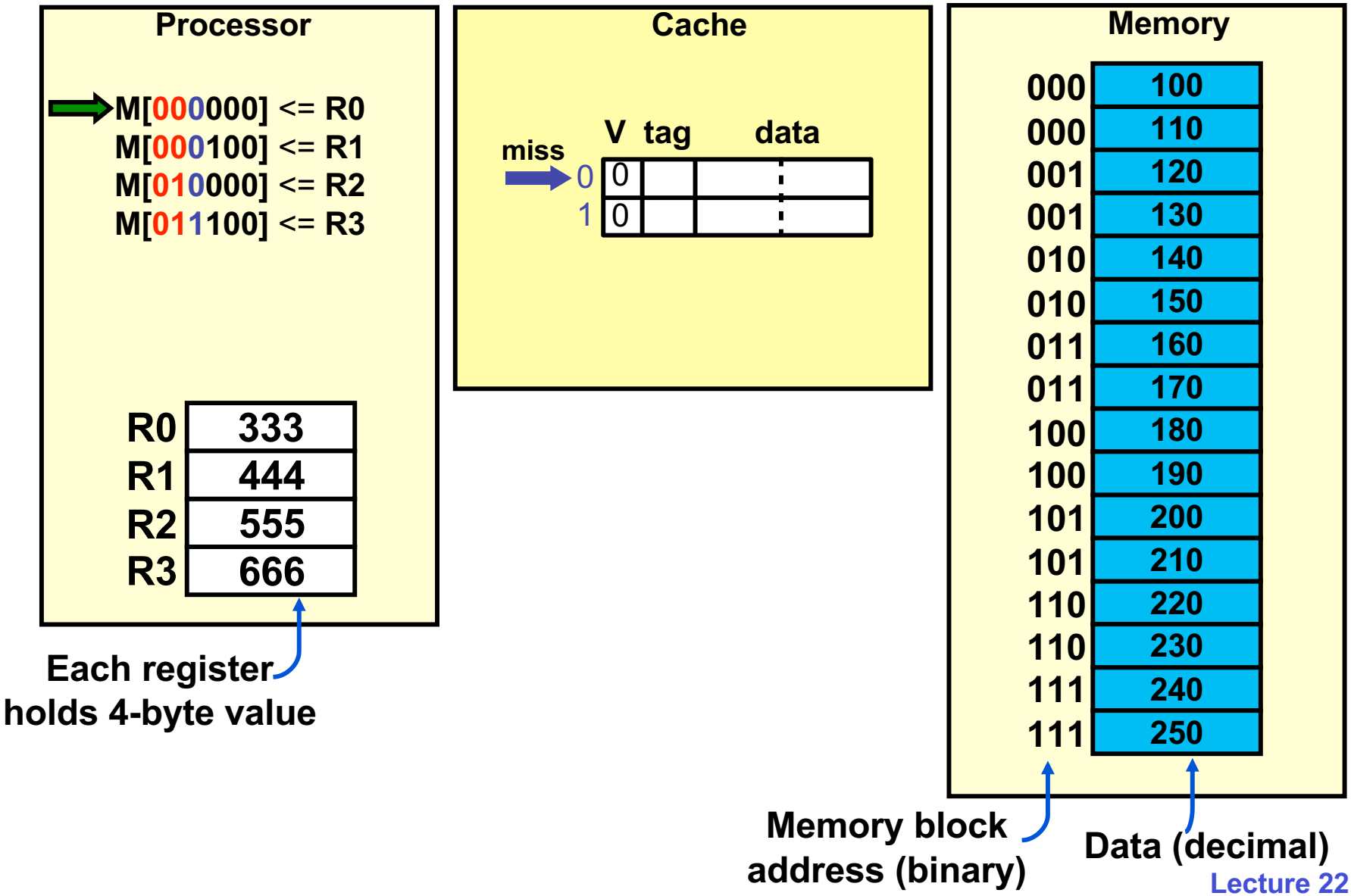
- Assume write allocate
- Size of each block is 8 bytes
- (Direct mapped) Cache holds 2 blocks
- Memory holds 8 blocks
- 6-bit memory address



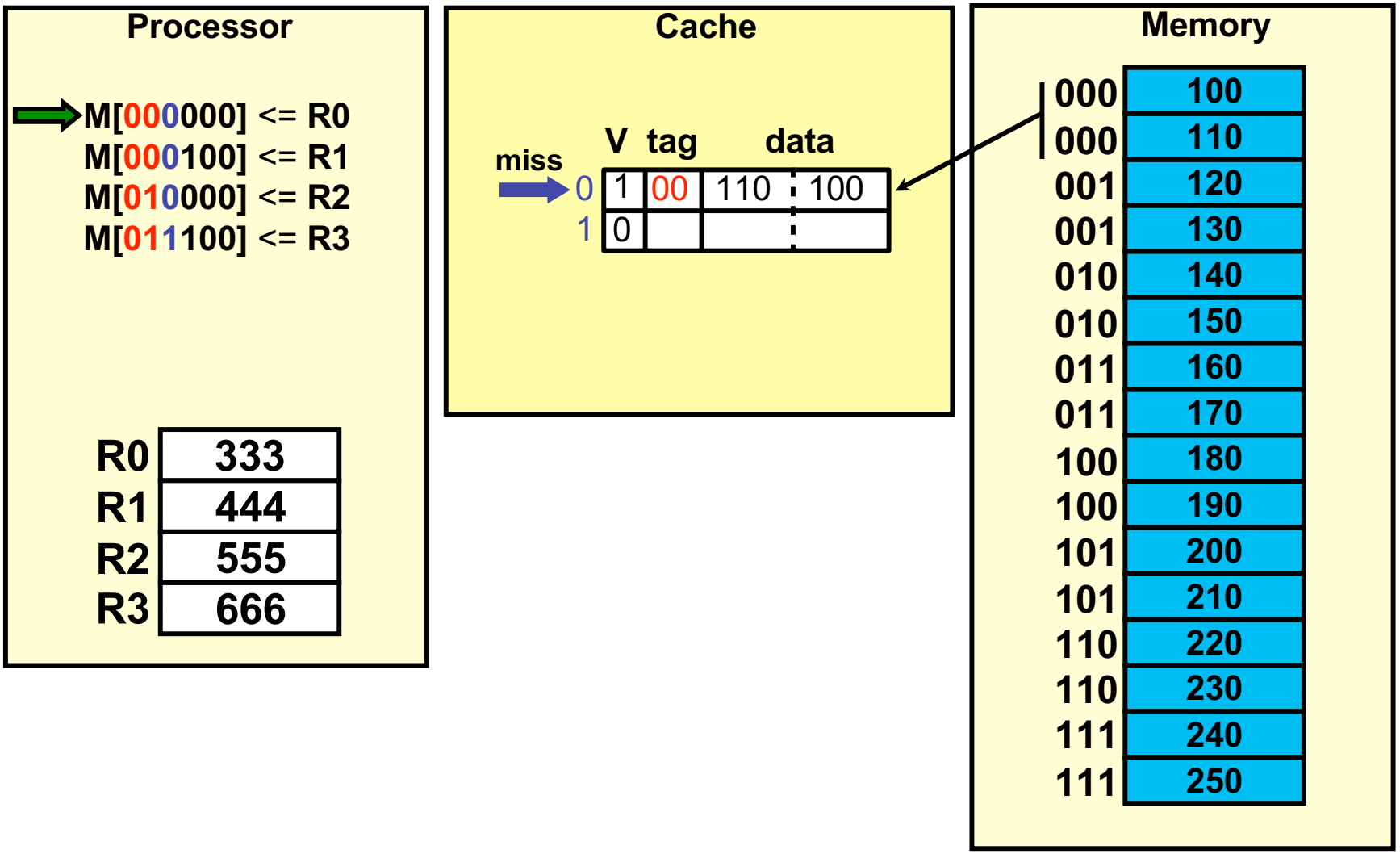


{ tag, index } = memory block address

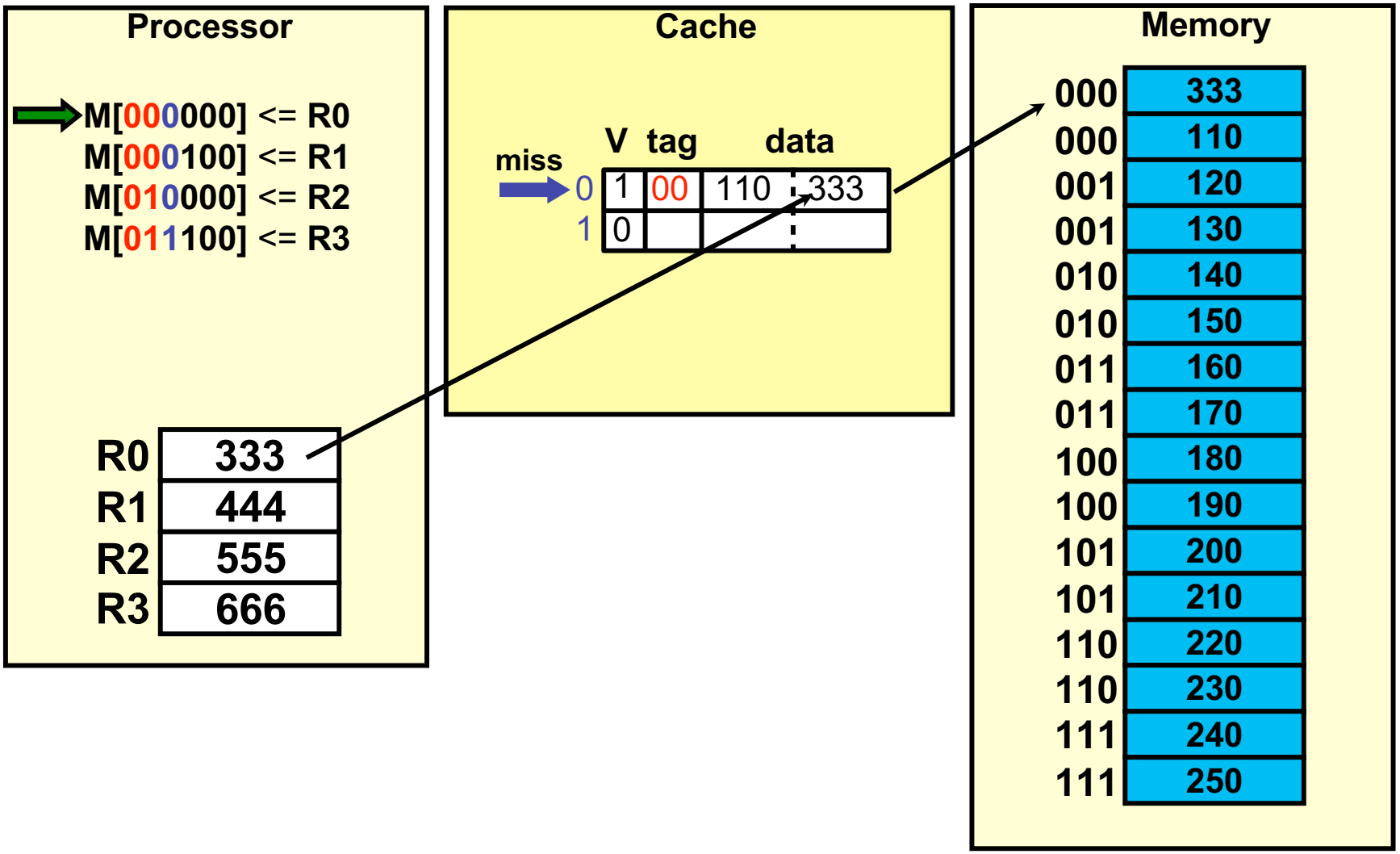
Write Through



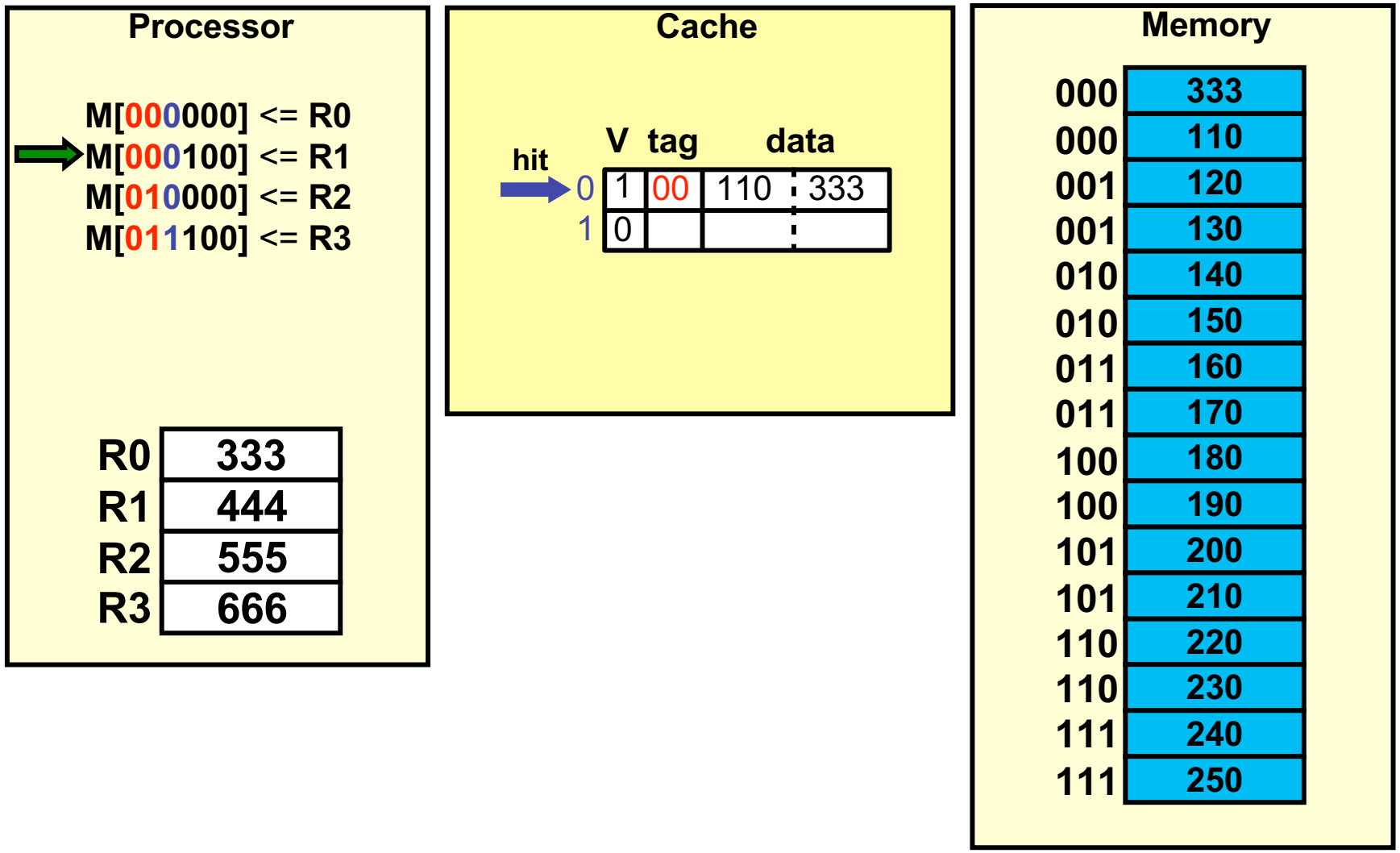
Write Through



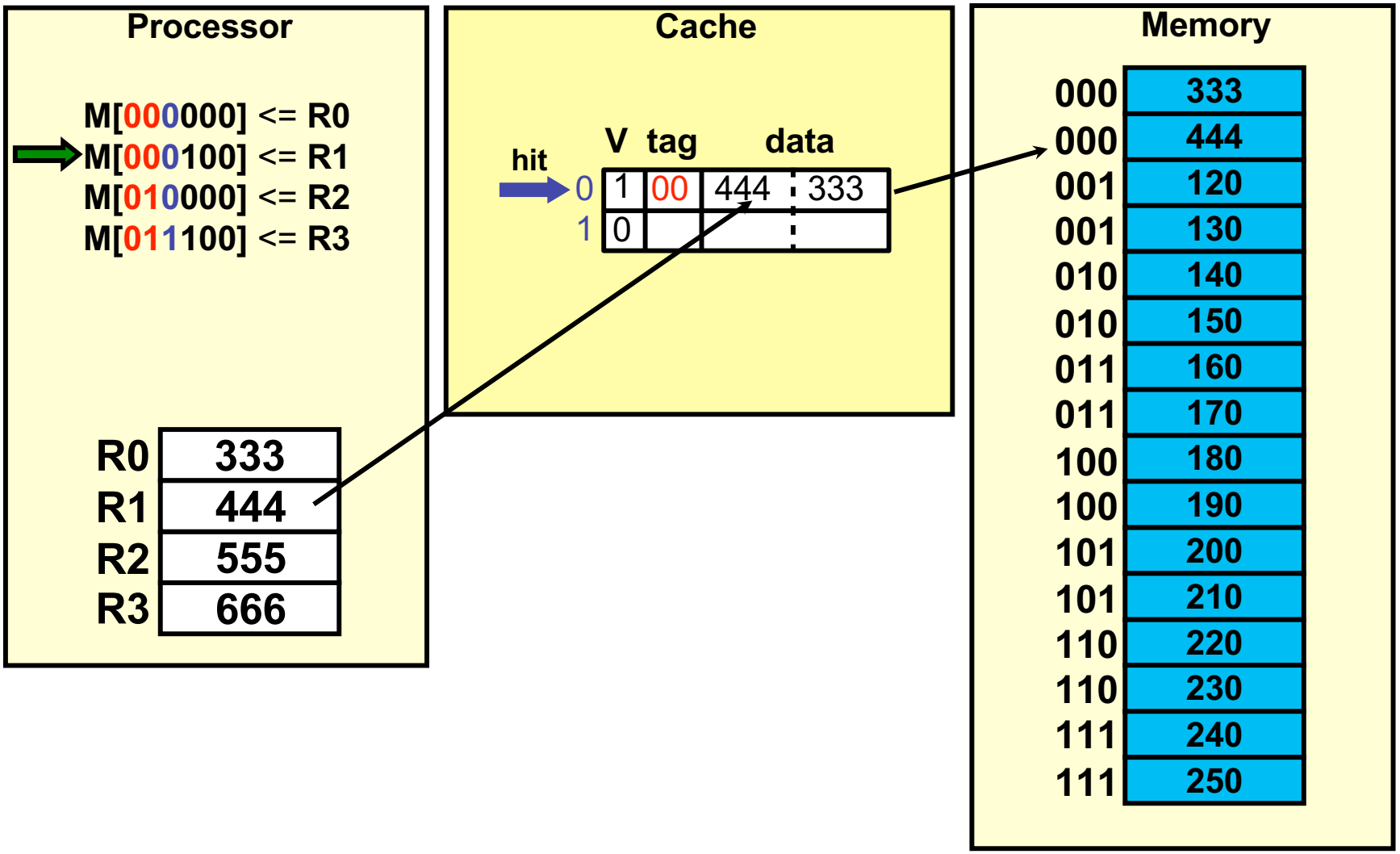
Write Through



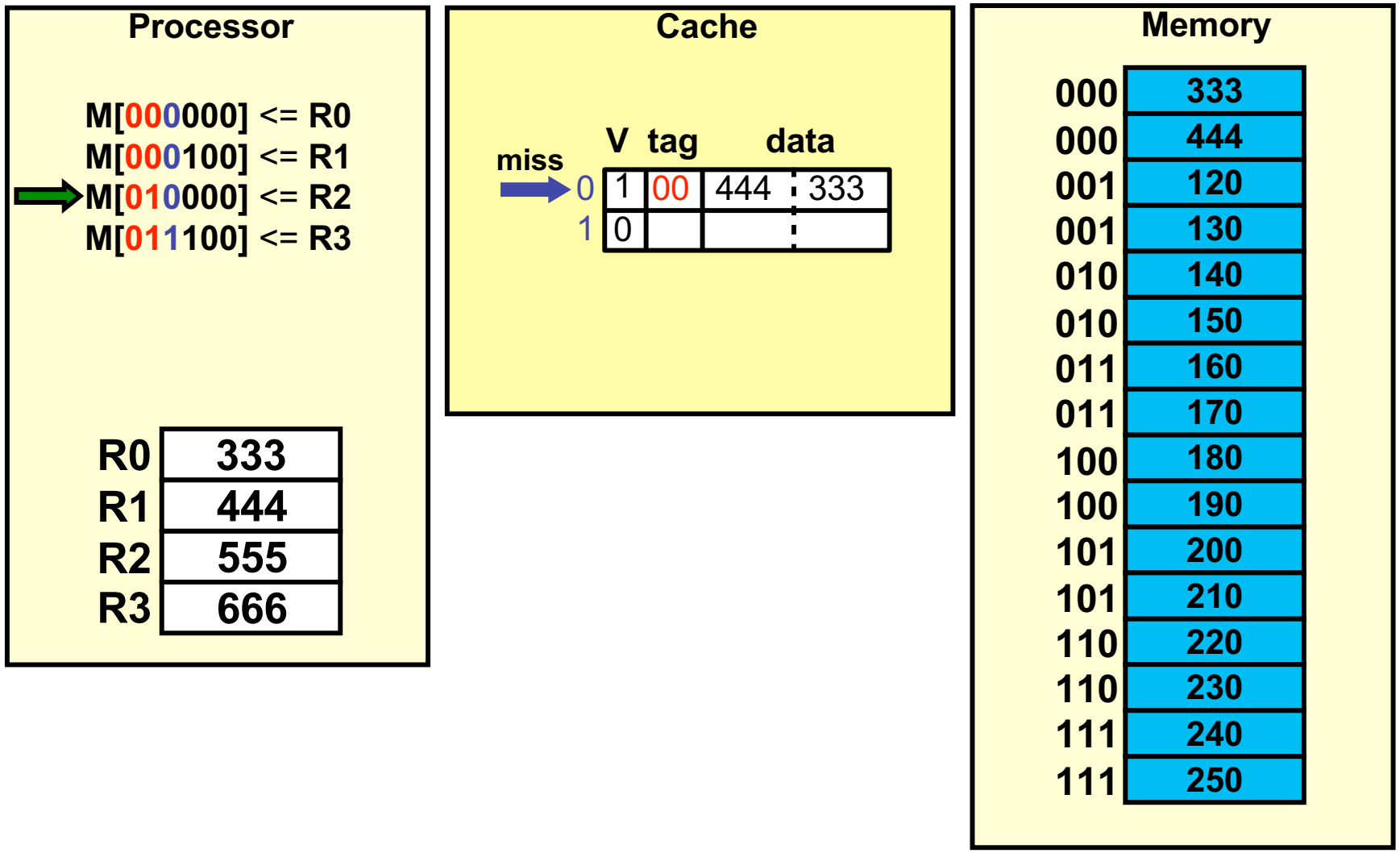
Write Through



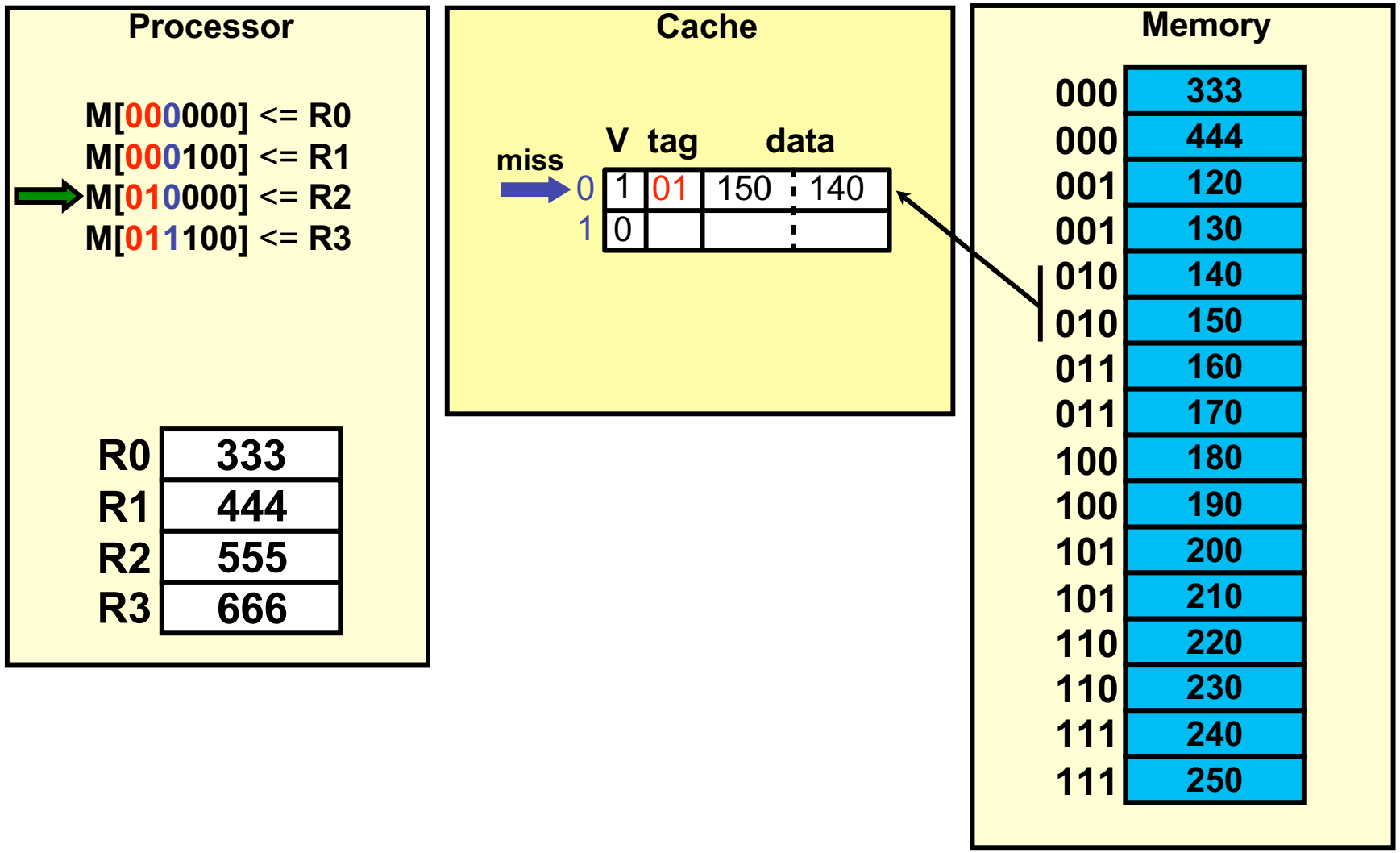
Write Through



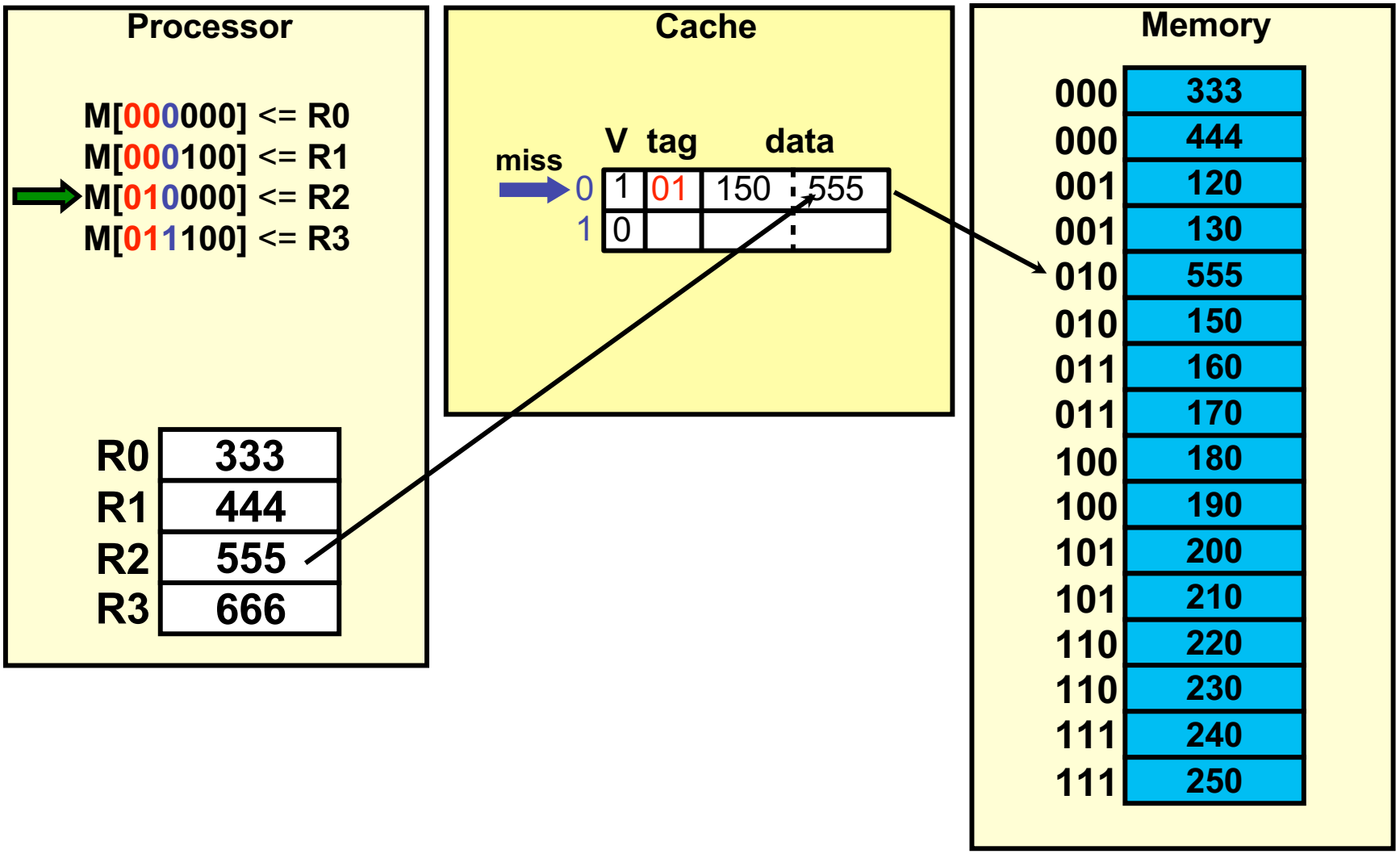
Write Through



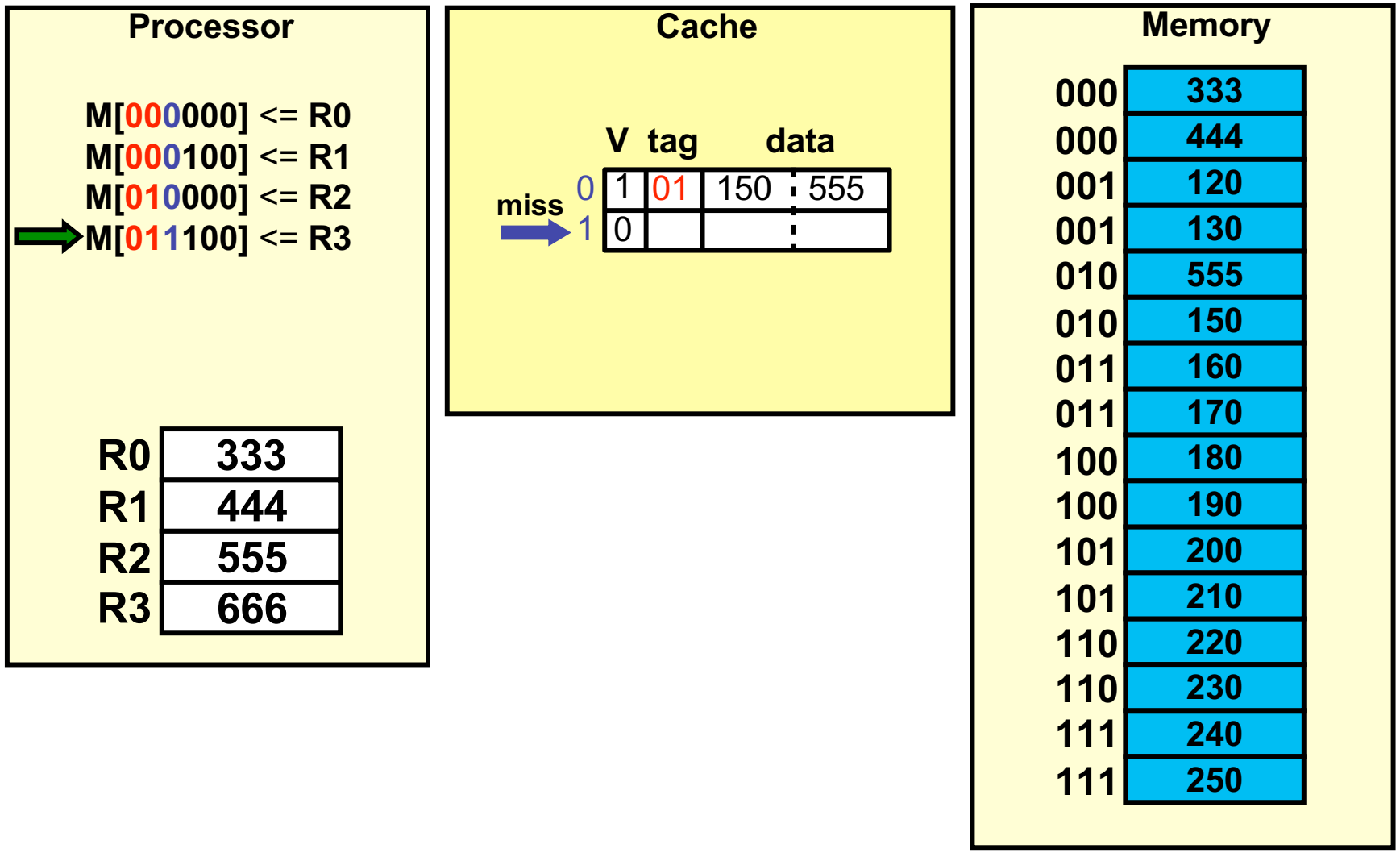
Write Through



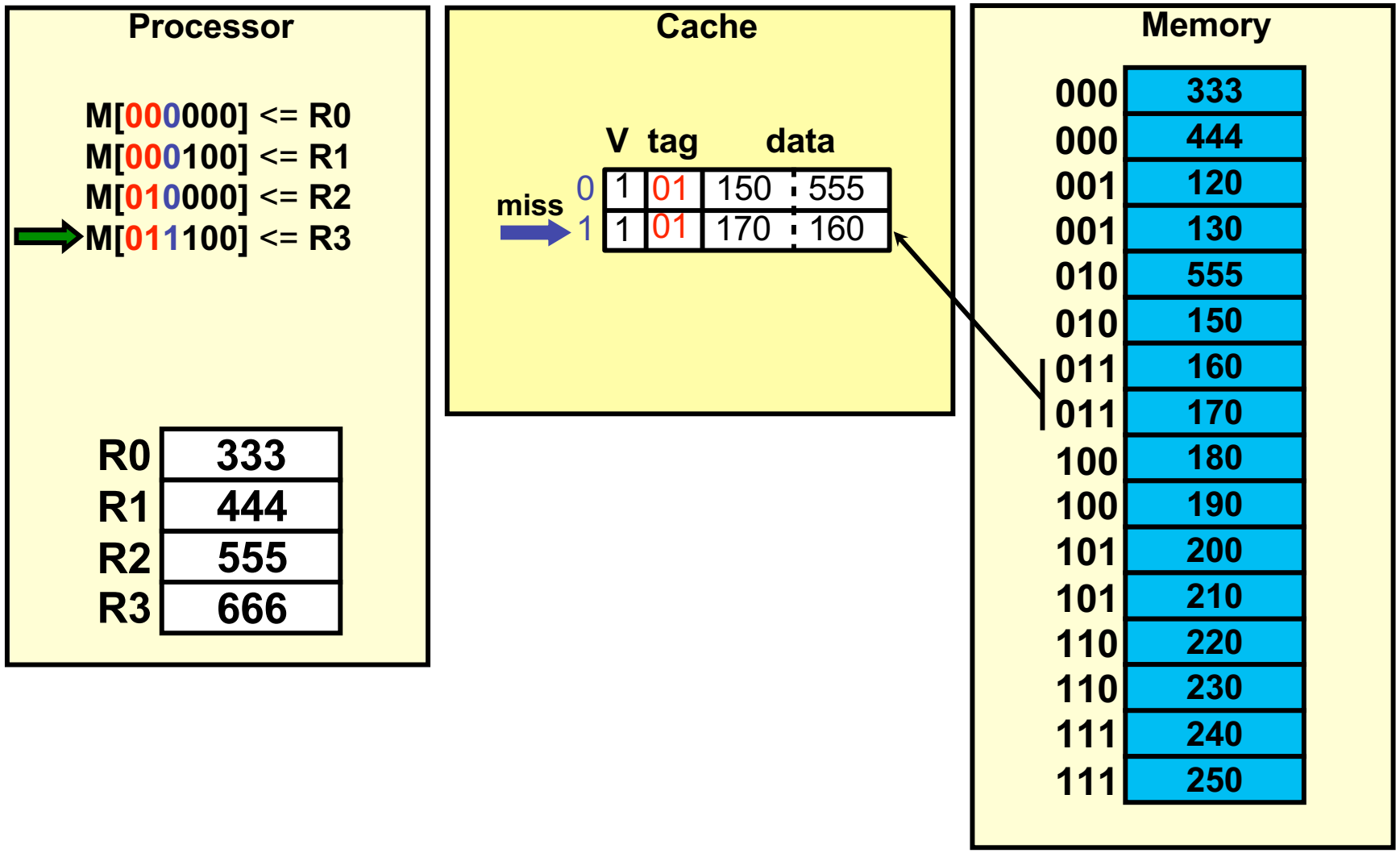
Write Through



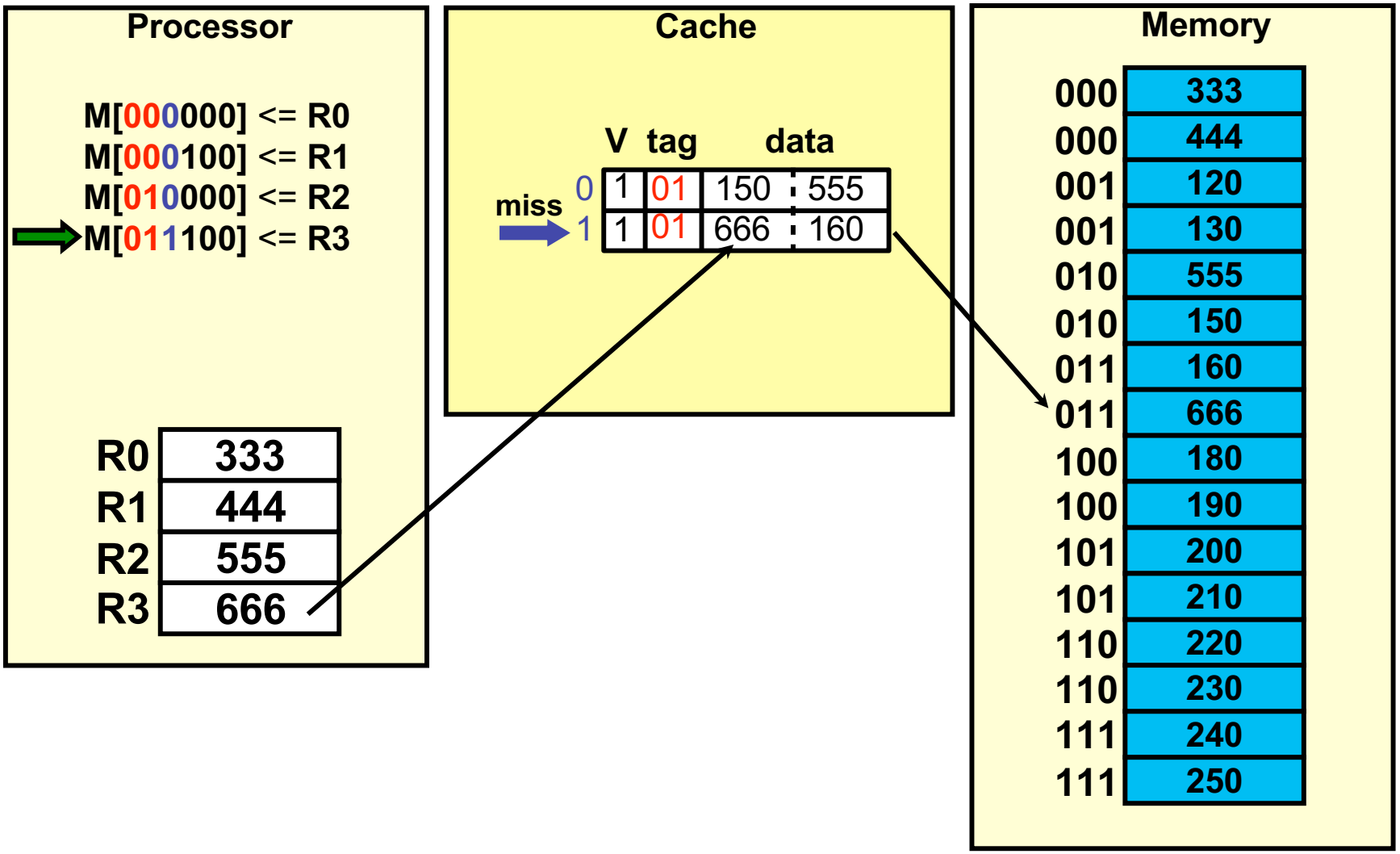
Write Through



Write Through

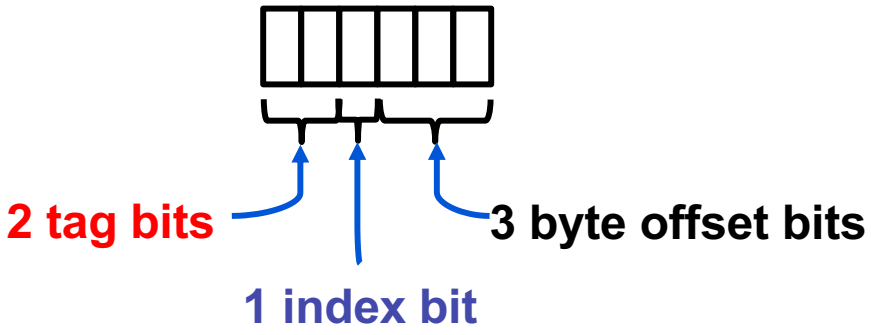


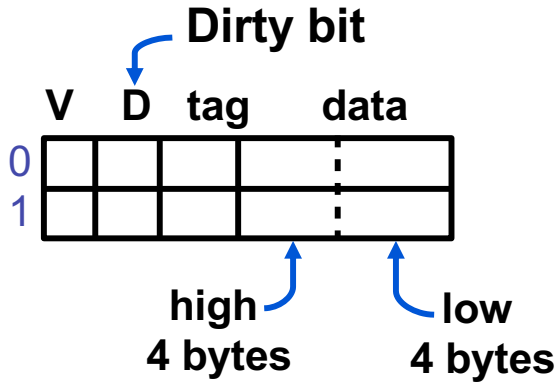
Write Through



Write Back Example

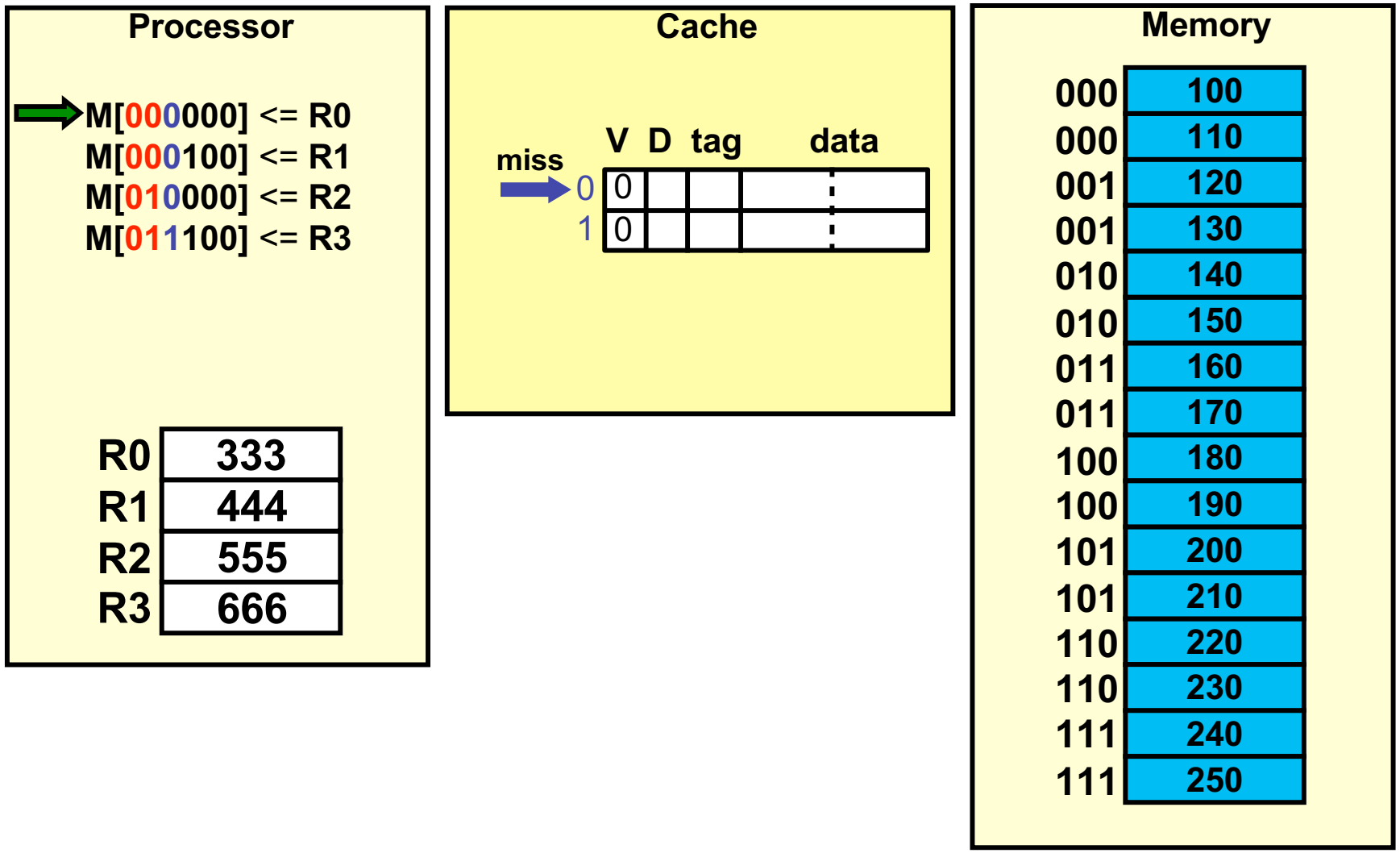
- Assume write allocate
- Size of each block is 8 bytes
- (Direct mapped) Cache holds 2 blocks
- Memory holds 8 blocks
- 6-bit memory address



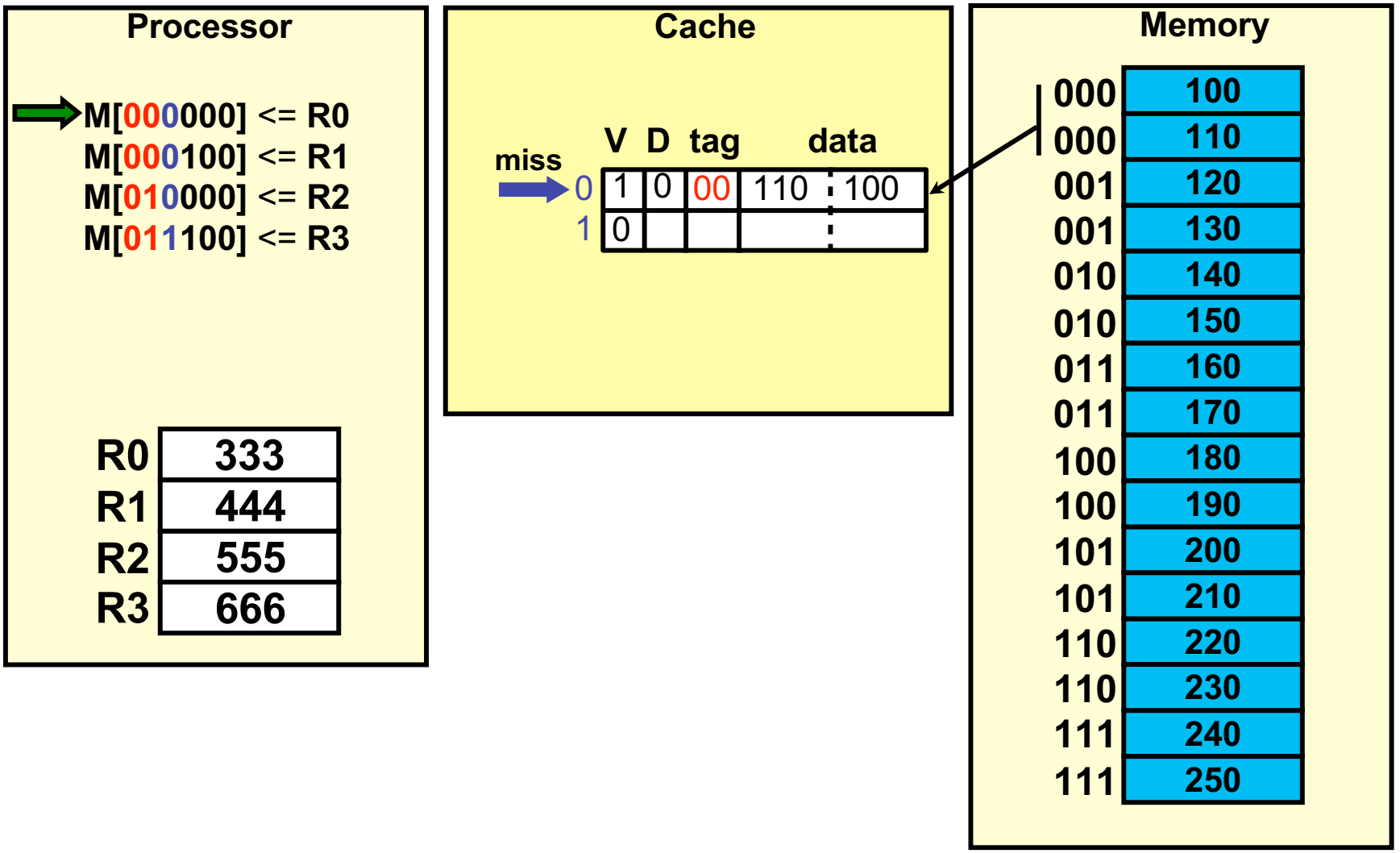


{ tag, index } = memory block address

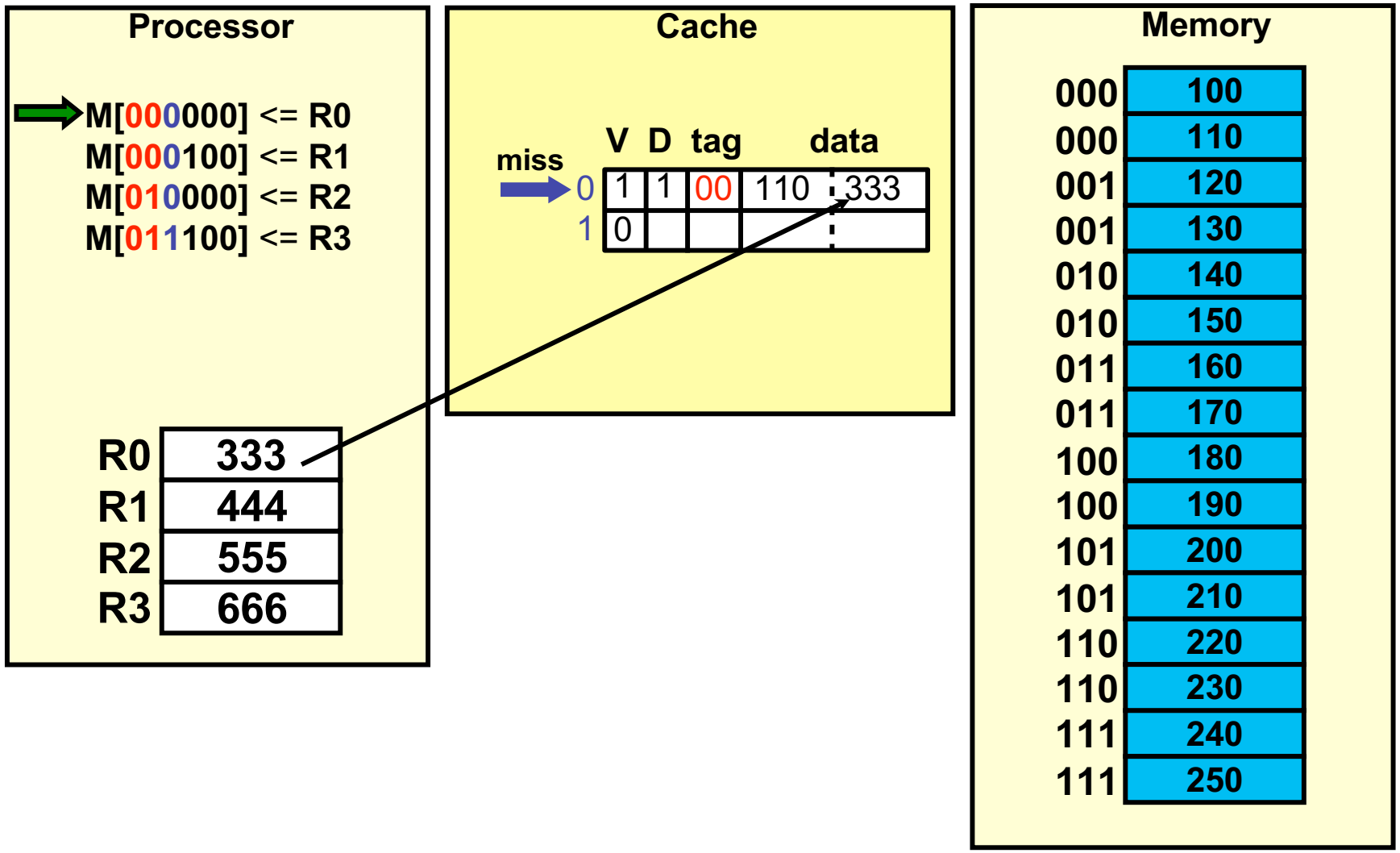
Write Back


Processor
M[000000] <= R0
M[000100] <= R1
M[010000] <= R2
M[011100] <= R3
R0 333
R1 444
R2 555
R3 666
Cache
miss
V D tag data
0 0
1 0
Memory
000 100
000 110
001 120
001 130
010 140
010 150
011 160
011 170
100 180
100 190
101 200
101 210
110 220
110 230
111 240
111 250

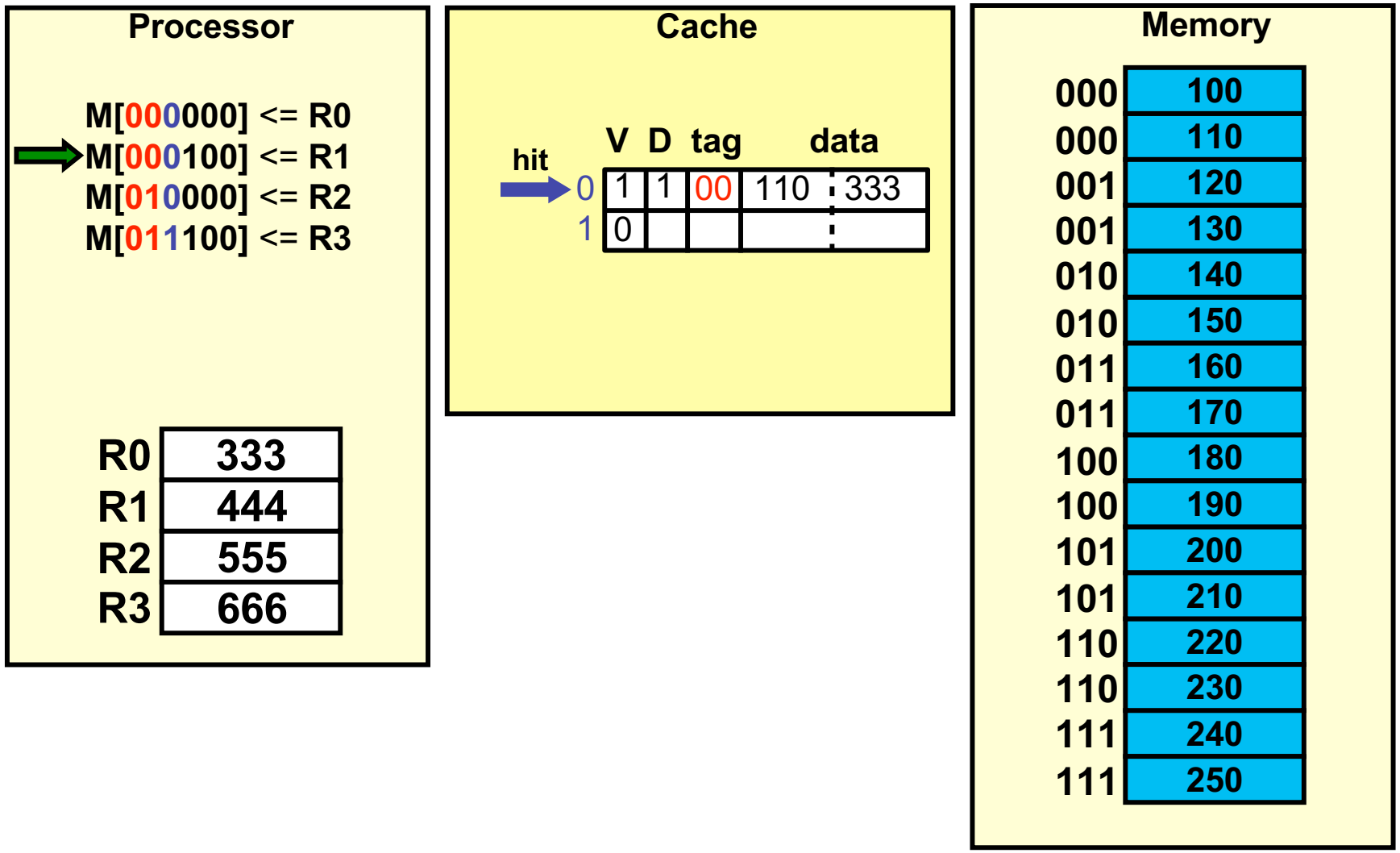
Write Back


Processor
M[000000] <= R0
M[000100] <= R1
M[010000] <= R2
M[011100] <= R3
R0 333
R1 444
R2 555
R3 666
Cache
miss
V D tag data
0 1 0 00 110 100
1 0
Memory
000 100
000 110
001 120
001 130
010 140
010 150
011 160
011 170
100 180
100 190
101 200
101 210
110 220
110 230
111 240
111 250

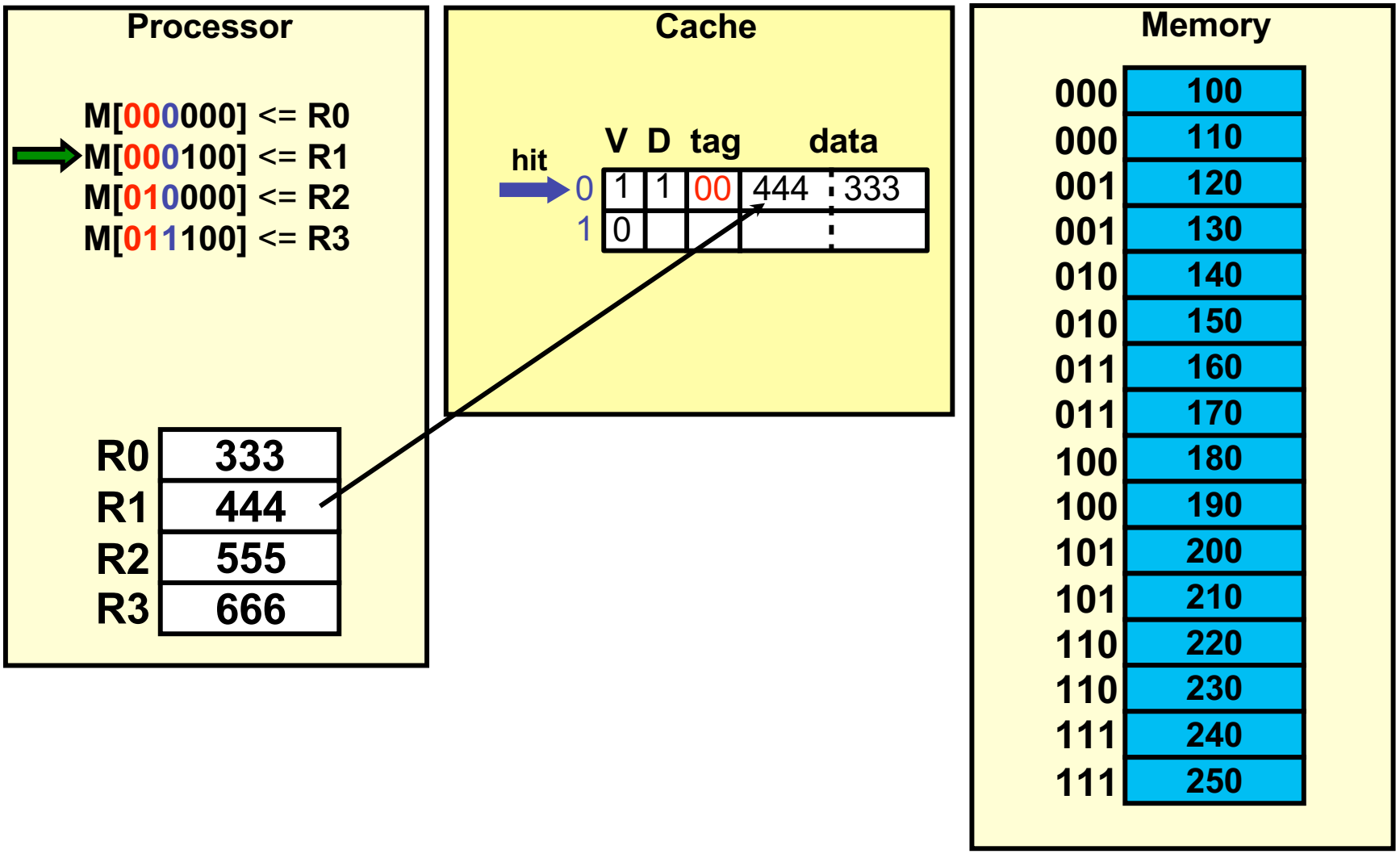
Write Back


Processor
M[000000] <= R0
M[000100] <= R1
M[010000] <= R2
M[011100] <= R3
R0 333
R1 444
R2 555
R3 666
Cache
miss
V D tag data
0 1 1 00 110 333
1 0
Memory
000 100
000 110
001 120
001 130
010 140
010 150
011 160
011 170
100 180
100 190
101 200
101 210
110 220
110 230
111 240
111 250

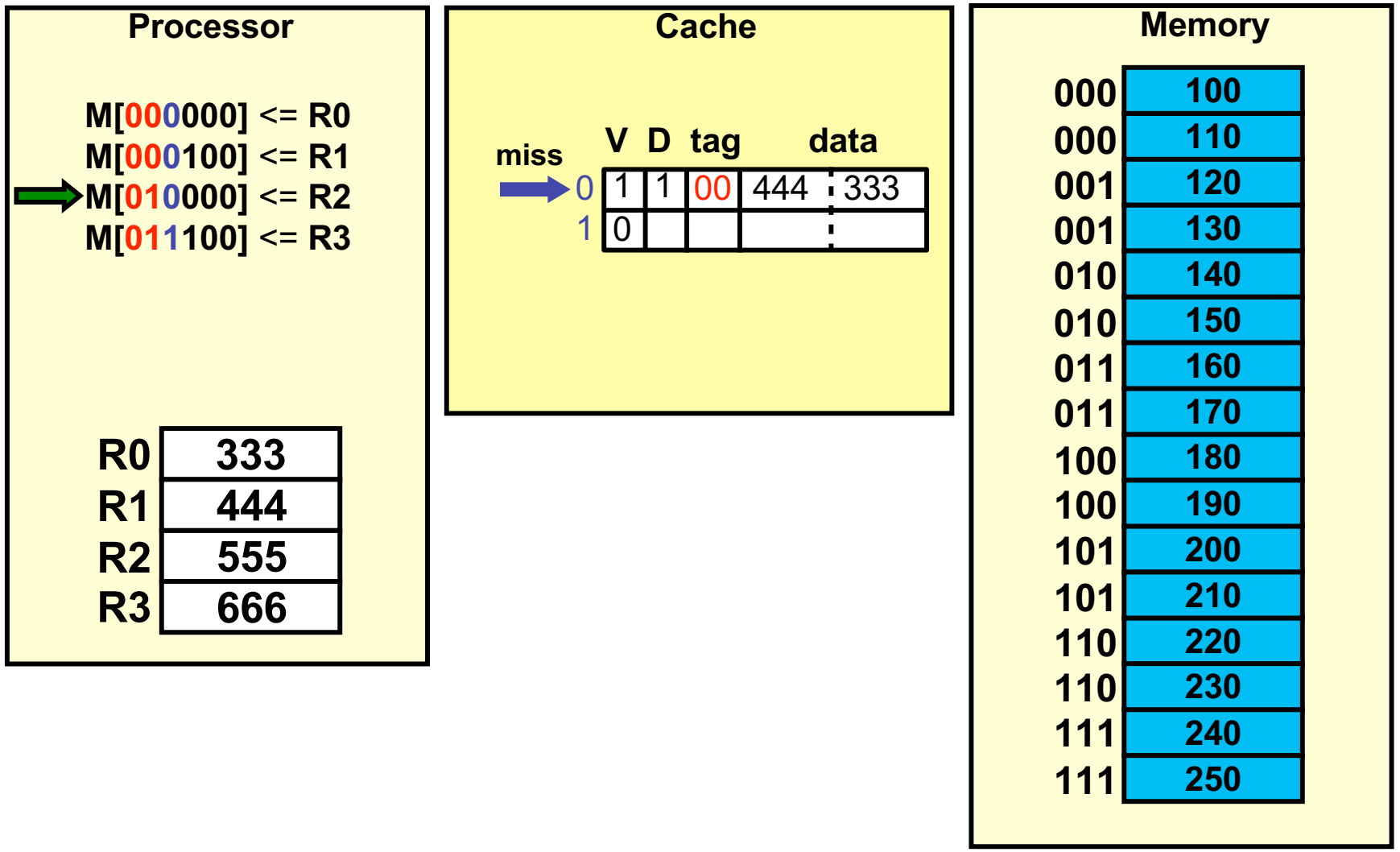
Write Back


Processor
M[000000] <= R0
M[000100] <= R1
M[010000] <= R2
M[011100] <= R3
R0 333
R1 444
R2 555
R3 666
Cache
hit
V D tag data
0 1 1 00 110 333
1 0
Memory
000 100
000 110
001 120
001 130
010 140
010 150
011 160
011 170
100 180
100 190
101 200
101 210
110 220
110 230
111 240
111 250

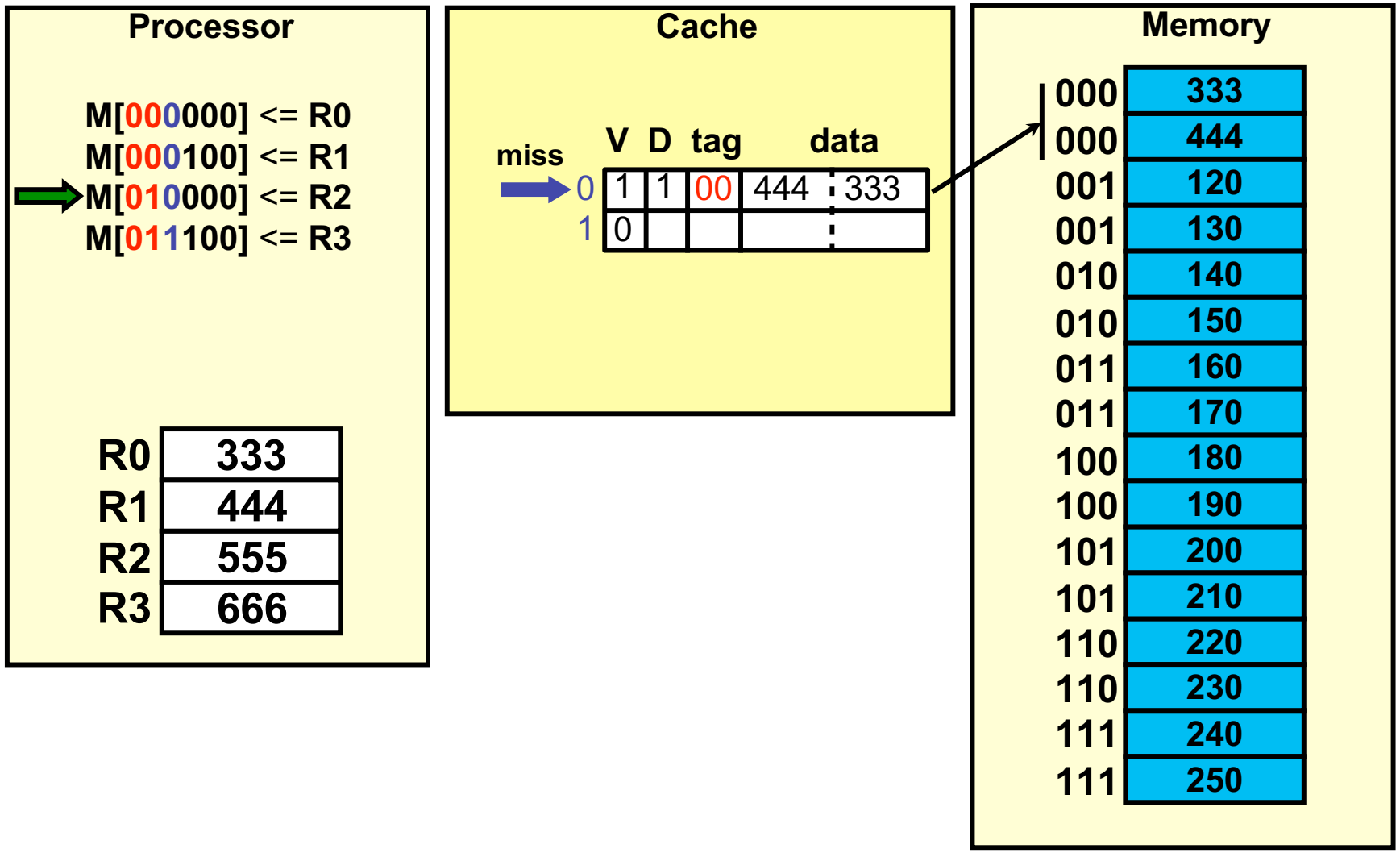
Write Back


Processor
M[000000] <= R0
M[000100] <= R1
M[010000] <= R2
M[011100] <= R3
R0 333
R1 444
R2 555
R3 666
Cache
hit
V D tag data
0 1 1 00 444 333
1 0
Memory
000 100
000 110
001 120
001 130
010 140
010 150
011 160
011 170
100 180
100 190
101 200
101 210
110 220
110 230
111 240
111 250

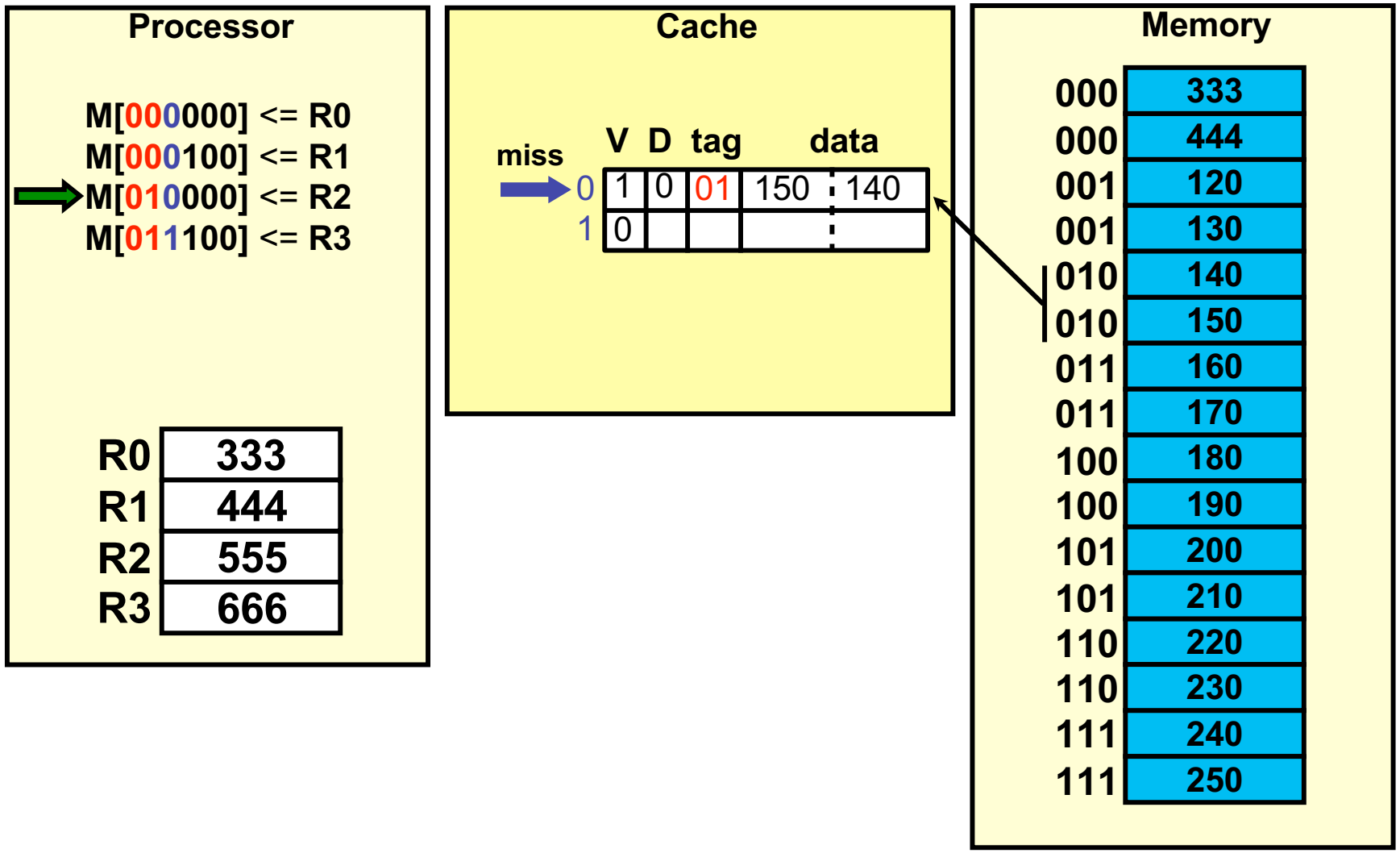
Write Back


Processor
M[000000] <= R0
M[000100] <= R1
M[010000] <= R2
M[011100] <= R3
R0 333
R1 444
R2 555
R3 666
Cache
miss
V D tag data
0 1 1 00 444 333
1 0
Memory
000 100
000 110
001 120
001 130
010 140
010 150
011 160
011 170
100 180
100 190
101 200
101 210
110 220
110 230
111 240
111 250

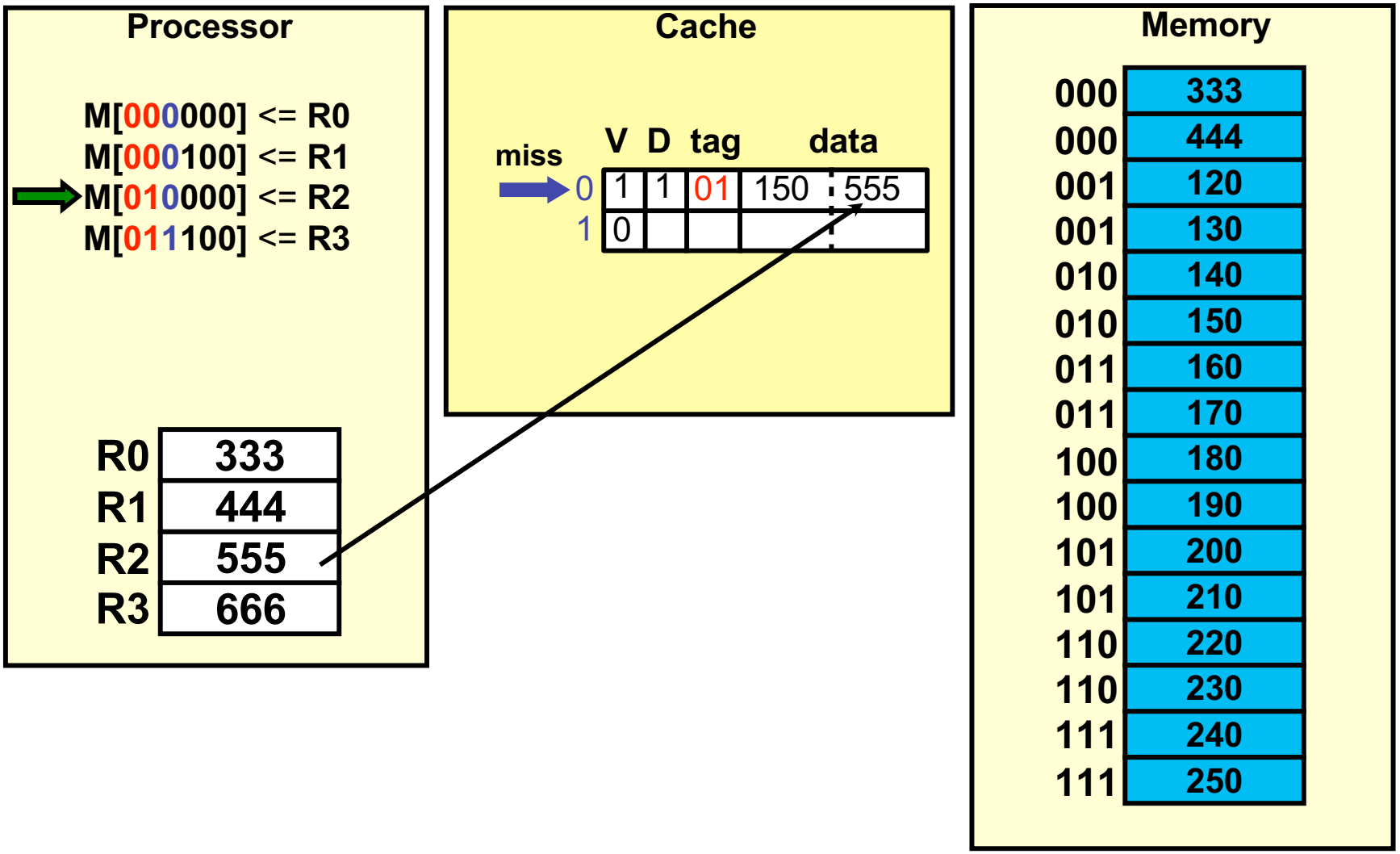
Write Back


Processor
M[000000] <= R0
M[000100] <= R1
M[010000] <= R2
M[011100] <= R3
R0 333
R1 444
R2 555
R3 666
Cache
miss
V D tag data
0 1 1 00 444 333
1 0
Memory
000 333
000 444
001 120
001 130
010 140
010 150
011 160
011 170
100 180
100 190
101 200
101 210
110 220
110 230
111 240
111 250

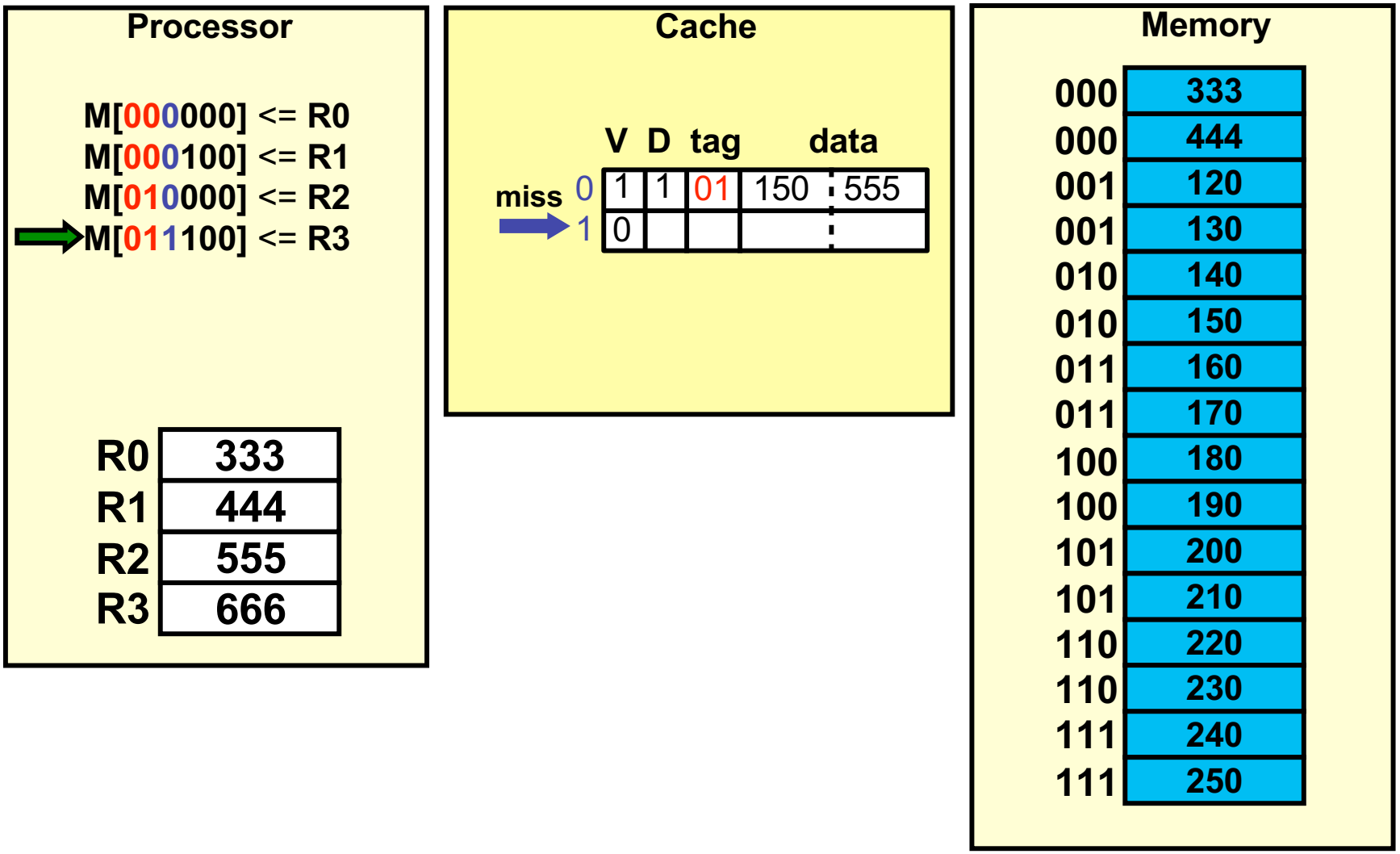
Write Back


Processor
M[000000] <= R0
M[000100] <= R1
M[010000] <= R2
M[011100] <= R3
R0 333
R1 444
R2 555
R3 666
Cache
miss
V D tag data
0 1 0 01 150 140
1 0
Memory
000 333
000 444
001 120
001 130
010 140
010 150
011 160
011 170
100 180
100 190
101 200
101 210
110 220
110 230
111 240
111 250

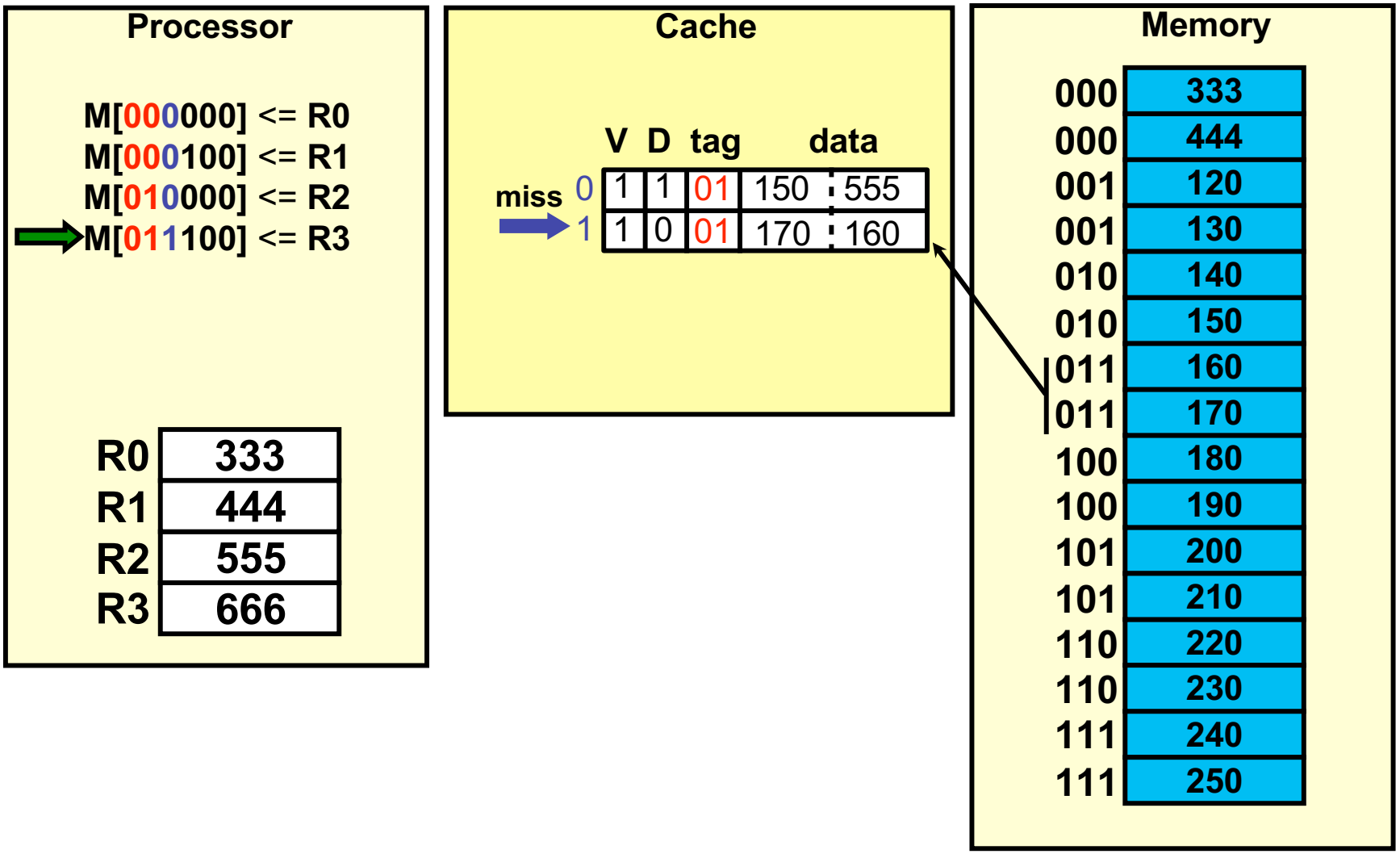
Write Back


Processor
M[000000] <= R0
M[000100] <= R1
M[010000] <= R2
M[011100] <= R3
R0 333
R1 444
R2 555
R3 666
Cache
miss
V D tag data
0 1 1 01 150 555
1 0
Memory
000 333
000 444
001 120
001 130
010 140
010 150
011 160
011 170
100 180
100 190
101 200
101 210
110 220
110 230
111 240
111 250

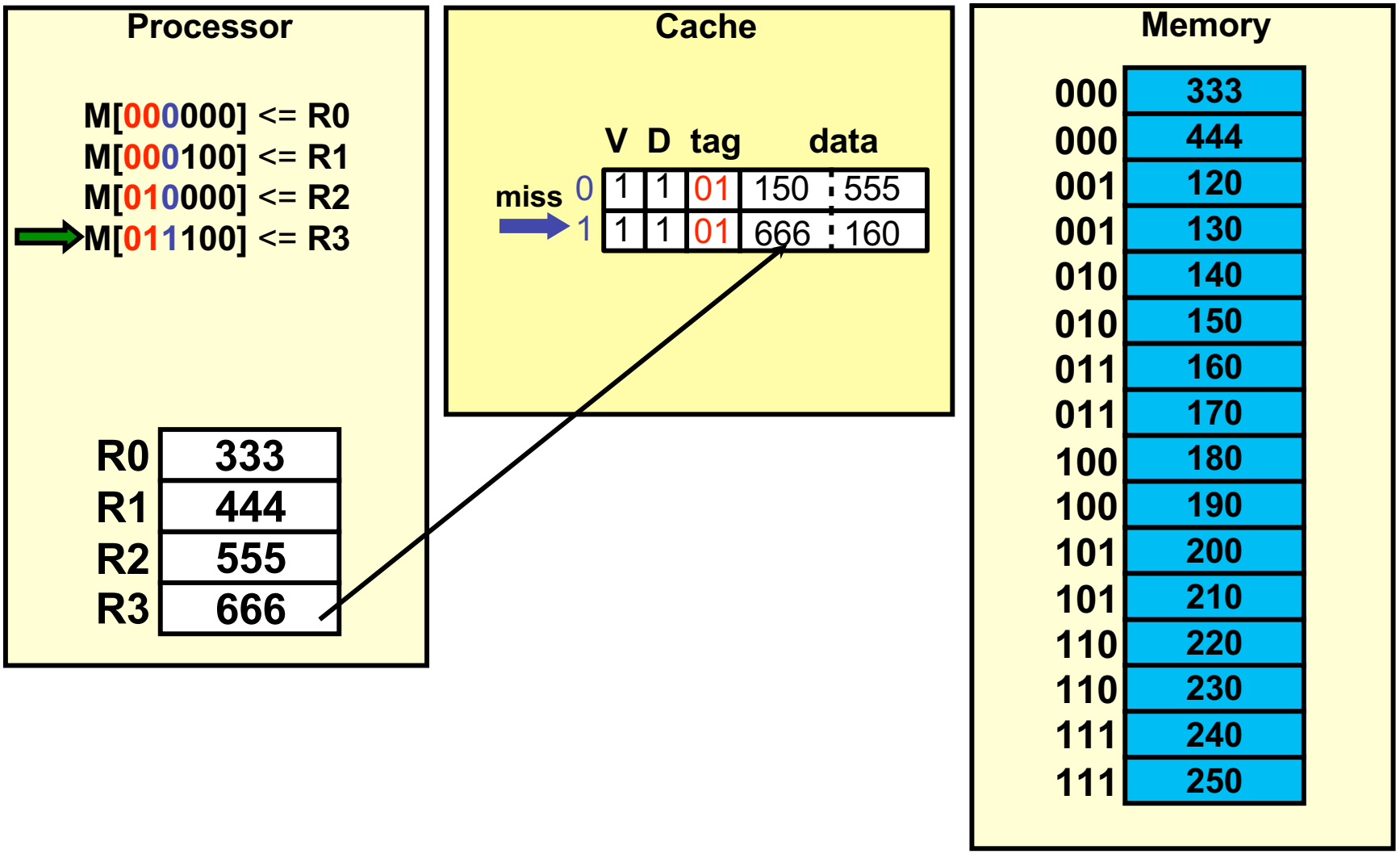
Write Back


Processor
M[000000] <= R0
M[000100] <= R1
M[010000] <= R2
M[011100] <= R3
R0 333
R1 444
R2 555
R3 666
Cache
V D tag data
miss
0 1 1 01 150 555
1 0
Memory
000 333
000 444
001 120
001 130
010 140
010 150
011 160
011 170
100 180
100 190
101 200
101 210
110 220
110 230
111 240
111 250

Write Back


Processor
M[000000] <= R0
M[000100] <= R1
M[010000] <= R2
M[011100] <= R3
R0 333
R1 444
R2 555
R3 666
Cache
V D tag data
miss 0 1 1 01 150 555
1 1 0 01 170 160
Memory
000 333
000 444
001 120
001 130
010 140
010 150
011 160
011 170
100 180
100 190
101 200
101 210
110 220
110 230
111 240
111 250

Write Back


Processor
M[000000] <= R0
M[000100] <= R1
M[010000] <= R2
M[011100] <= R3
R0 333
R1 444
R2 555
R3 666
Cache
V D tag data
miss 0 1 1 01 150 555
1 1 1 01 666 160
Memory
000 333
000 444
001 120
001 130
010 140
010 150
011 160
011 170
100 180
100 190
101 200
101 210
110 220
110 230
111 240
111 250

Next Class

Measuring Performance
Performance Tradeoffs
(H&H 7.5.4, 8.2)