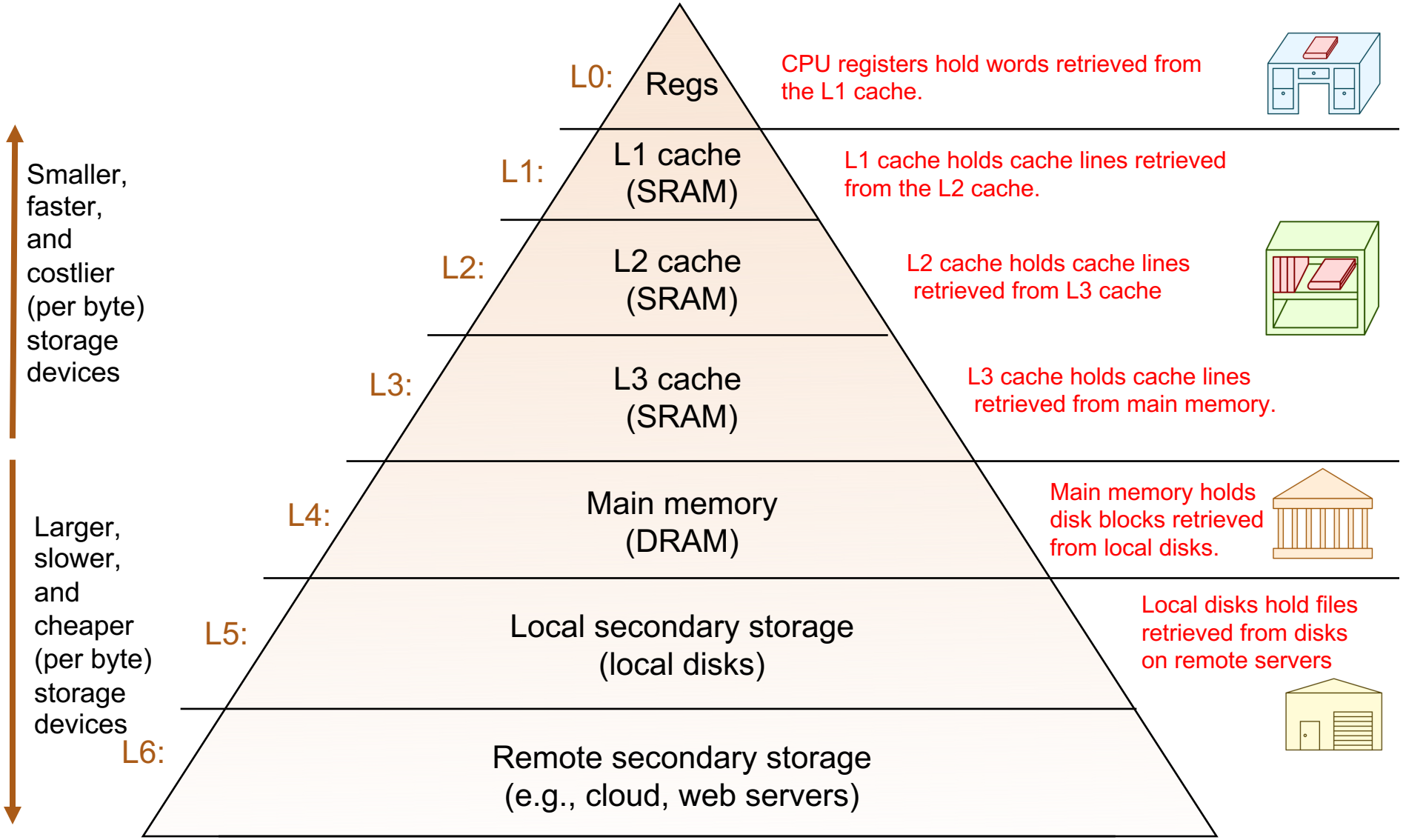
Cache

Set-Associativity and Eviction

Memory Hierarchy

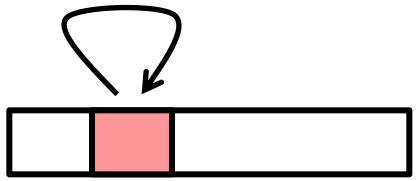


Principle of Locality

Programs tend to use data and instructions with addresses near or equal to those they have used recently

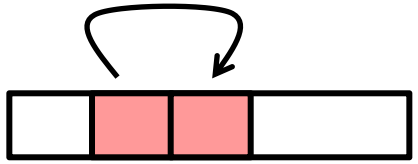
Temporal locality:

- Recently referenced items are likely to be referenced again soon

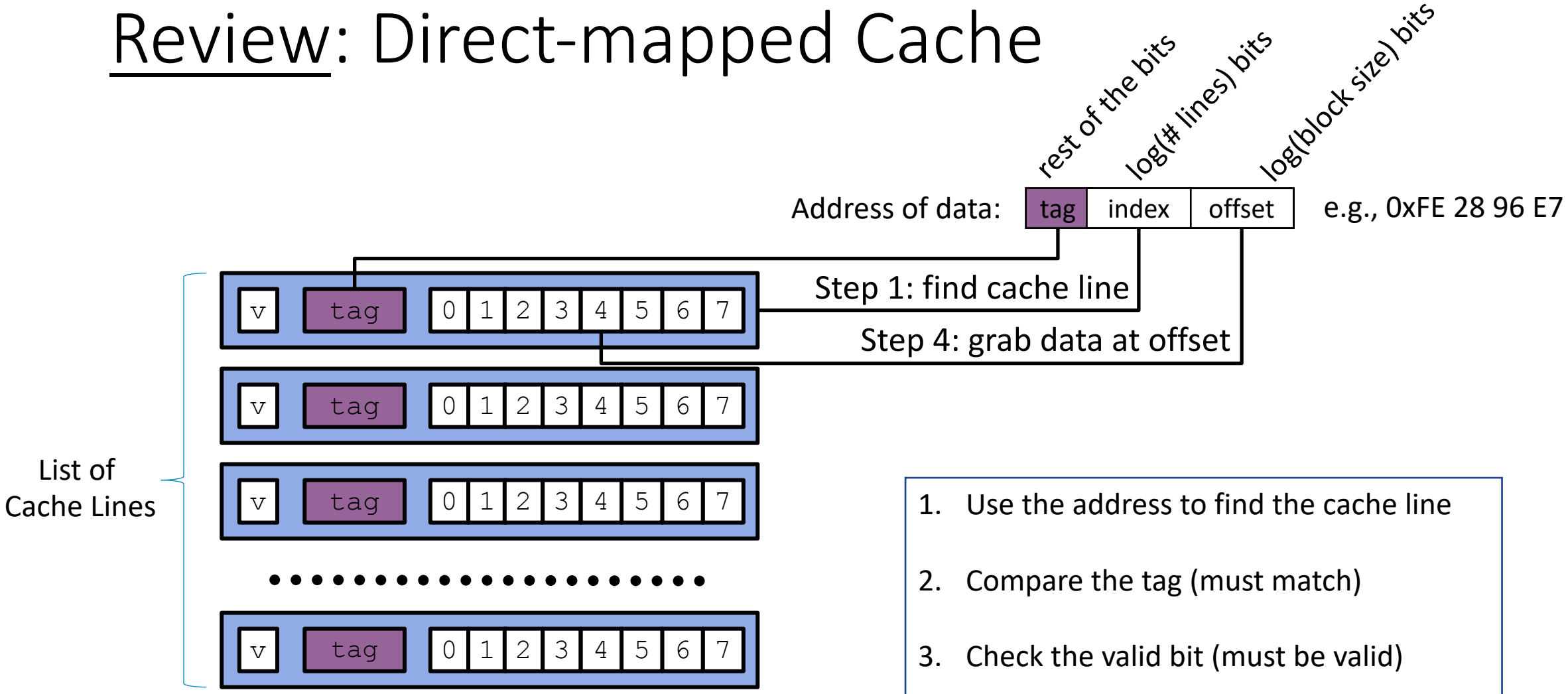


Spatial locality:

- Items with nearby addresses tend to be referenced close together in time



Review: Direct-mapped Cache

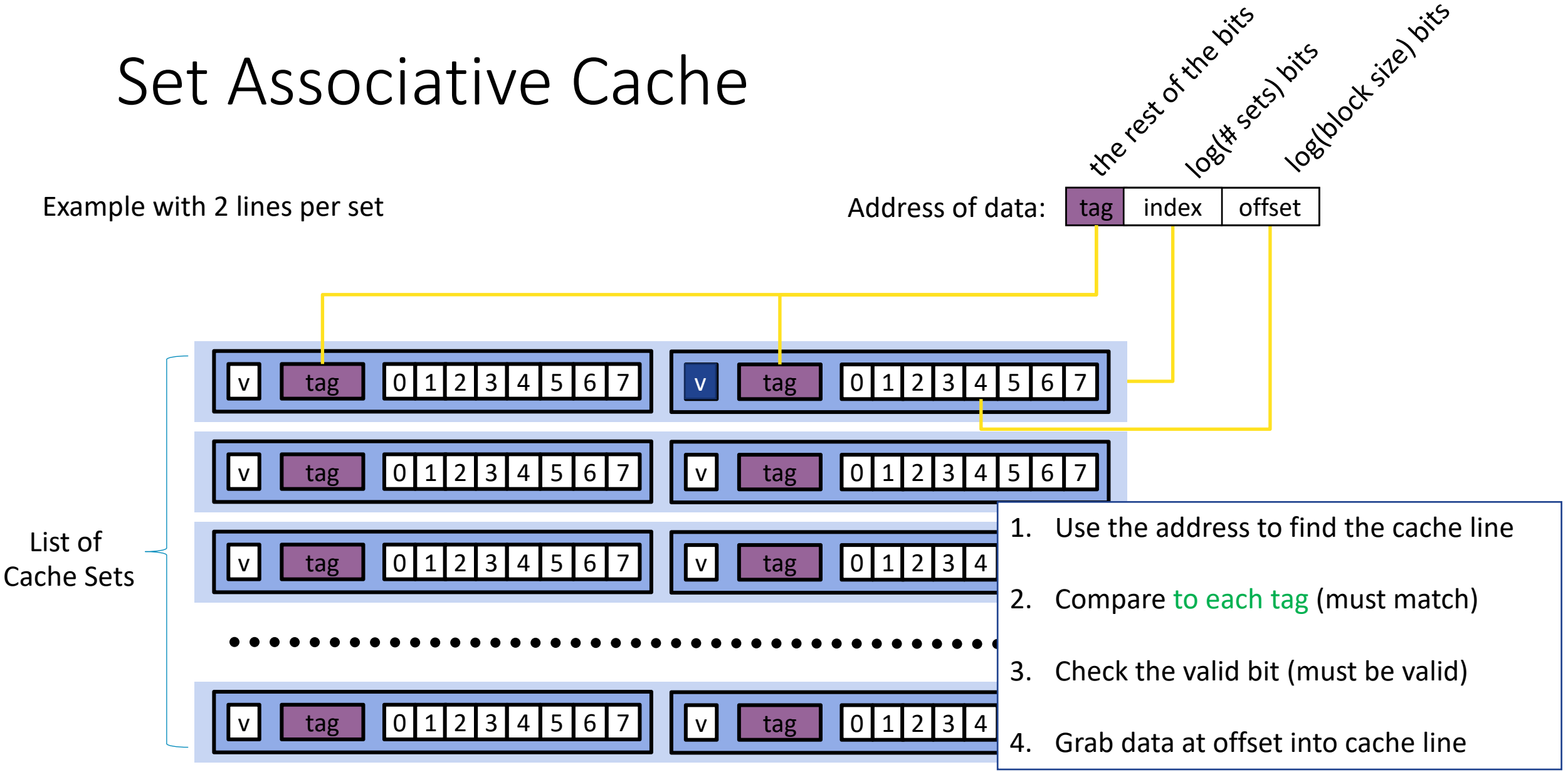


1. Use the address to find the cache line
2. Compare the tag (must match)
3. Check the valid bit (must be valid)
4. Grab data at offset into cache line

Do the first two steps sound familiar?

Set Associative Cache

Example with 2 lines per set

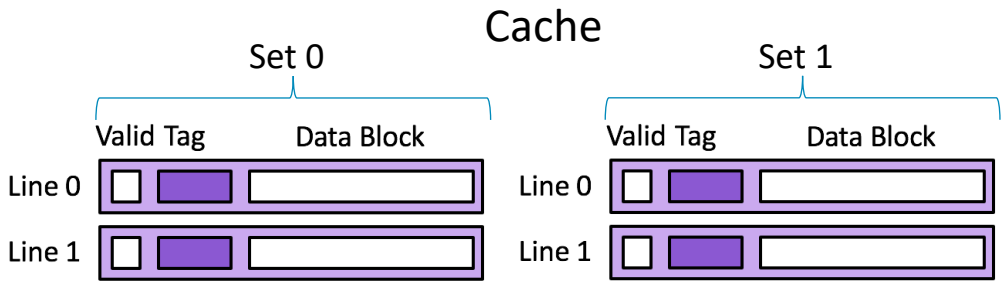


Practice with Set Associative Cache

Memory

0x14	18
0x10	17
0x0c	16
0x08	15
0x04	14
0x00	13

Cache



Assume 8-byte data blocks

Binary	Access	tag	idx	off	h/m
0000 0000	rd 0x00				
0000 0100	rd 0x04				
0001 0100	rd 0x14				
0000 0000	rd 0x00				
0000 0100	rd 0x04				
0001 0100	rd 0x14				
0010 0100	rd 0x20				

Set 0				Set 1			
Line 0		Line 1		Line 0		Line 1	
0 0	47 48	0 1	47 48	0 0	47 48	0 1	47 48

Time

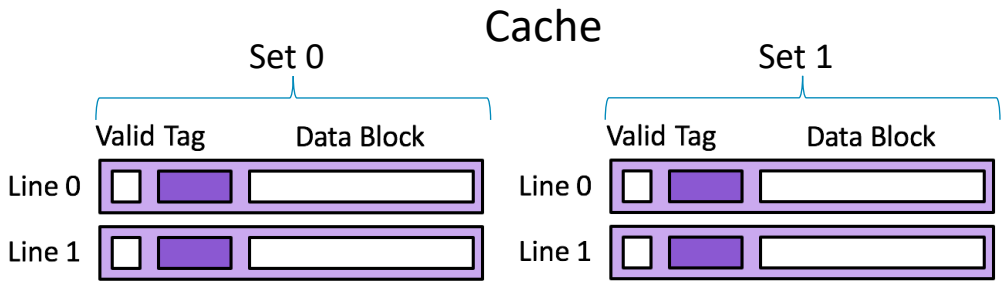
Practice with Set Associative Cache

What kind of hit?

Memory

0x14	18
0x10	17
0x0c	16
0x08	15
0x04	14
0x00	13

Cache



Assume 8-byte data blocks

Binary	Access	tag	idx	off	h/m
0000 0000	rd 0x00	0000	0	000	m
0000 0100	rd 0x04	0000	0	100	h
0001 0100	rd 0x14	0001	0	100	m
0000 0000	rd 0x00	0000	0	000	h
0000 0100	rd 0x04	0000	0	100	h
0001 0100	rd 0x14	0001	0	100	h
0010 0100	rd 0x20	0010	0	000	m

Set 0				Set 1			
Line 0				Line 1			
0	0	47	48	0	1	47	48
1	0	13	14				
				1	1	17	18

Now what?

Time

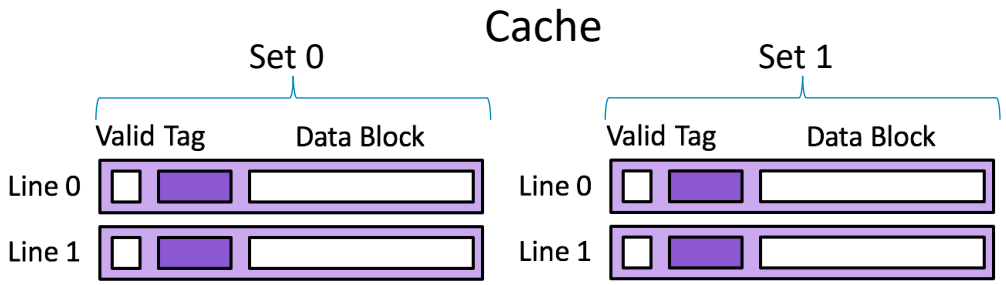
Eviction from the Cache

- On a cache miss, a new block is loaded into the cache
- In direct-mapped cache a valid block at the same location must be evicted (there is no other no choice)
- In set-associative cache (assuming all lines are valid), one line must be evicted
  - Random policy
  - FIFO
  - LIFO
  - Least-recently used; requires extra data in each set
  - Most-recently used; requires extra data in each set
  - Most-frequently used; requires extra data in each set

Practice with Set Associative Cache

Memory

0x14	18
0x10	17
0x0c	16
0x08	15
0x04	14
0x00	13



Assume 8-byte data blocks

Binary	Access	tag	idx	off	h/m
0000 0000	rd 0x00	0000	0	000	m
0000 0100	rd 0x04	0000	0	100	h
0001 0100	rd 0x14	0001	0	100	m
0000 0000	rd 0x00	0000	0	000	h
0000 0100	rd 0x04	0000	0	100	h
0001 0100	rd 0x14	0001	0	100	h
0010 0100	rd 0x20	0010	0	000	m

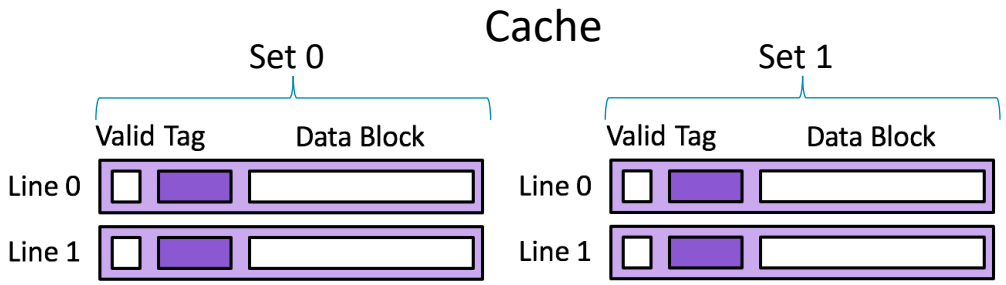
Set 0								Set 1							
Line 0				Line 1				Line 0				Line 1			
0	0	47	48	0	1	47	48	0	0	47	48	0	1	47	48
1	0	13	14												
				1	1	17	18								

Time

Practice with Set Associative Cache

Memory

0x14	18
0x10	17
0x0c	16
0x08	15
0x04	14
0x00	13



Assume 8-byte data blocks

Binary	Access	tag	idx	off	h/m
0000 0000	rd 0x00	0000	0	000	m
0000 0100	rd 0x04	0000	0	100	h
0001 0100	rd 0x14	0001	0	100	m
0000 0000	rd 0x00	0000	0	000	h
0000 0100	rd 0x04	0000	0	100	h
0001 0100	rd 0x14	0001	0	100	h
0010 0100	rd 0x20	0010	0	000	m

Set 0								Set 1							
Line 0				Line 1				Line 0				Line 1			
0	0	47	48	0	1	47	48	0	0	47	48	0	1	47	48
1	0	13	14												
				1	1	17	18								
1	2	21	22												

Time

Caching Organization Summarized

- A cache consists of lines
- A line contains
  - A block of bytes, the data values from memory
  - A tag, indicating where in memory the values are from
  - A valid bit, indicating if the data are valid
- Lines are organized into sets
  - One set per line in direct-mapped cache
  - K lines per set in k-way associative cache
  - N lines per set in fully associative cache

Caching Vocabulary

- Size: the total number of bytes that can be stored in the cache

- Cache Hit: the desired value is in the cache and quickly returned
- Hite rate: the fraction of accesses that are hits
- Hit time: the time to process a hit

- Cache Miss: the desired value is not in the cache and must be fetched elsewhere
- Miss rate: the fraction of accesses that are misses
- Miss penalty: the additional time to process a miss

- Average access time: hit-time + miss-rate * miss-penalty

Categorizing Misses

- Compulsory: first-reference to a block (no way to hit)
- Capacity: cache is too small to hold all data (no way to fit)
- Conflict: collisions in a specific set (bad luck or poor planning)

- Average access time: hit-time + miss-rate * miss-penalty

Practice

Categorize each miss as one of
1. Compulsory
2. Capacity
3. Conflict

Based on your categorizations, would you recommend
1. increasing the block size
2. increasing the associativity
3. increasing the total cache size

0x14
0x10
0x0c
0x08
0x04
0x00

Assume 8-byte data blocks

Binary	Access	tag	idx	off	h/m
0000 0000	rd 0x00	0000	0	000	m
0000 0100	rd 0x04	0000	0	100	h
0001 0100	rd 0x14	0001	0	100	m
0000 0000	rd 0x00	0000	0	000	h
0000 0100	rd 0x04	0000	0	100	h
0001 0100	rd 0x14	0001	0	100	h
0010 0100	rd 0x20	0010	0	000	m

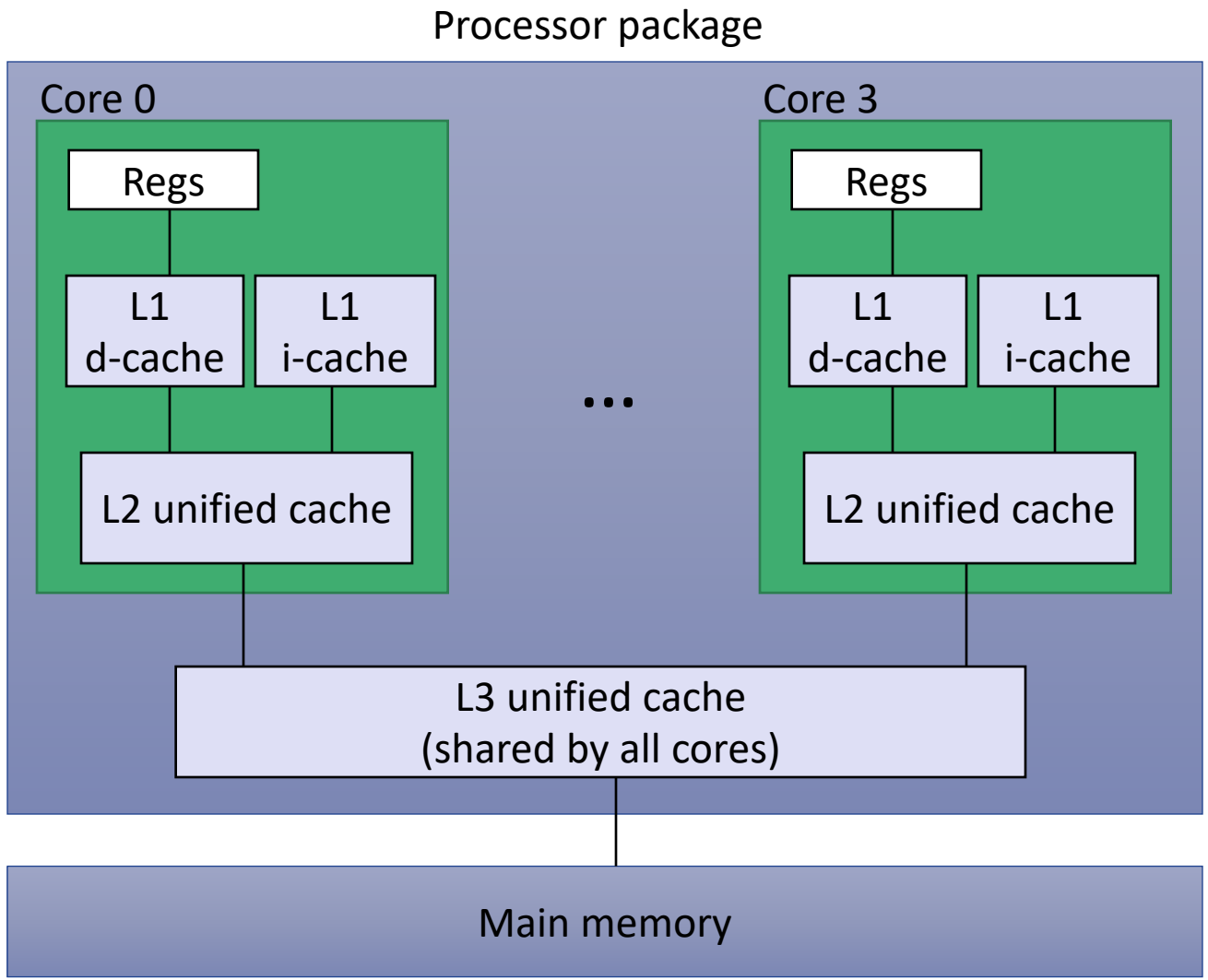
Set 1

Line 0				Line 1				Line 0				Line 1			
0	0	47	48	0	1	47	48	0	0	47	48	0	1	47	48
1	0	13	14												
				1	1	17	18								

Time

Now what?

Typical Intel Core i7 Hierarchy



Register reference:
Access: 1 cycle

L1 d-cache and i-cache:
32 KB, 8-way,
Access: 4 cycles

L2 unified cache:
256 KB, 8-way,
Access: 10 cycles

L3 unified cache:
8 MB, 16-way,
Access: 40-75 cycles

Block size: 64 bytes for all caches.