

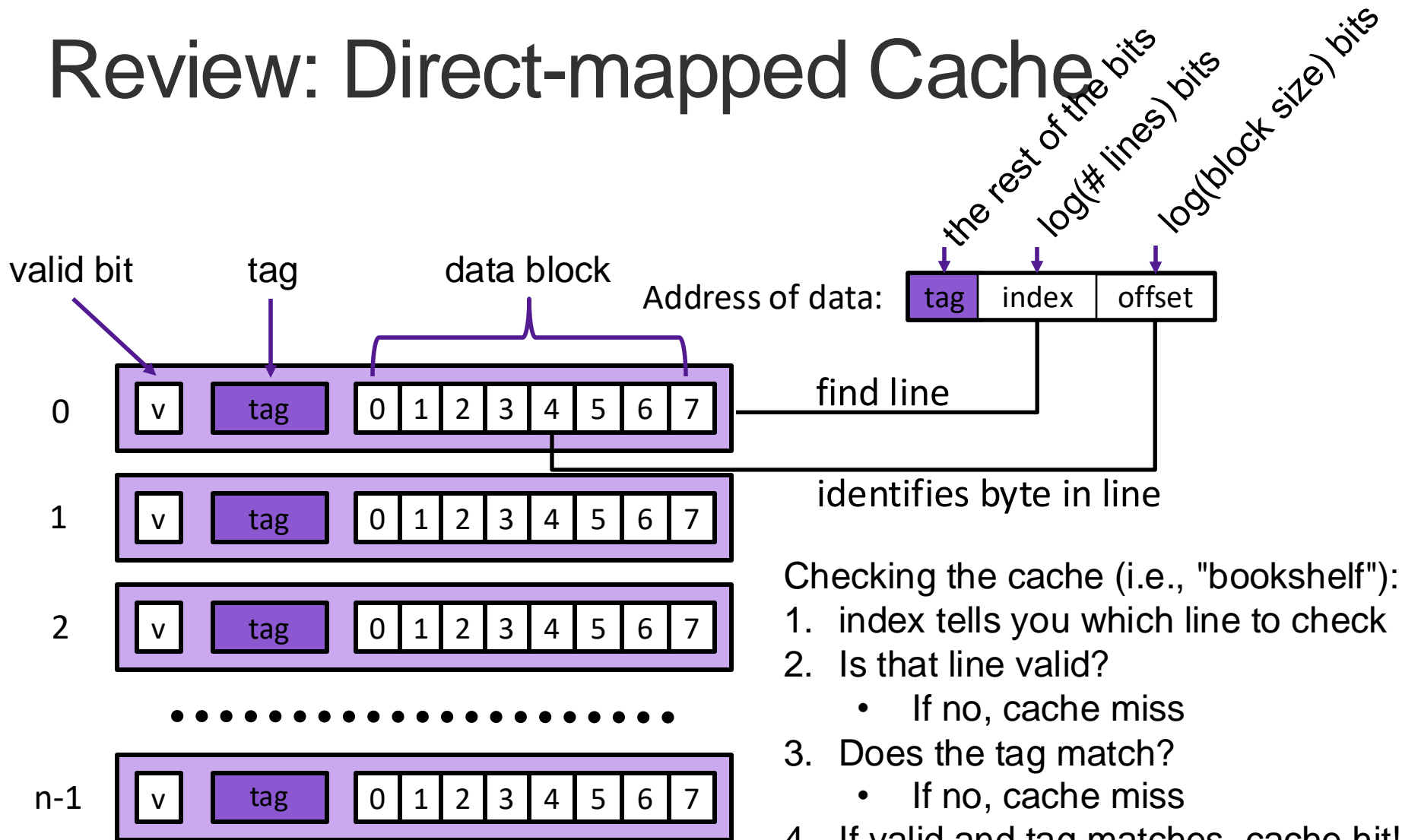
# Lecture 11: Caches (cont'd)

---

CS 105

Fall 2024

# Review: Direct-mapped Cache



Checking the cache (i.e., "bookshelf"):

1. index tells you which line to check
2. Is that line valid?
  - If no, cache miss
3. Does the tag match?
  - If no, cache miss
4. If valid and tag matches, cache hit!
  - Read from datablock at offset

# Review: Handling Cache Miss

When a cache miss occurs update cache line at that index:

1. Replace data block with bytes from memory
  - Copies all bytes with same tag + index
2. Update tag
3. Set valid bit to 1 (if not already)

Address of data:

0x74

0111 0100

011 | 10 | 100

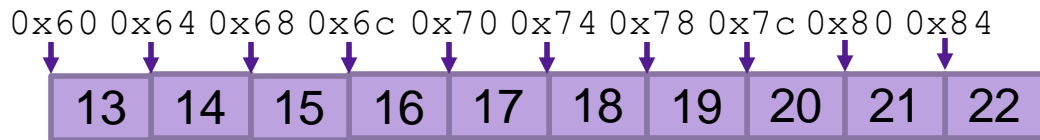
3 bit tag  
2 bit index  
3 bit offset

0x79	23
0x78	B7
0x77	64
0x76	15
0x75	E0
0x74	AB
0x73	1A
0x72	47
0x71	33
0x70	2F
0x6F	0A
0x6E	00

Line 0	1	110	0F	12	AB	34	FF	FF	EA	68
Line 1	1	001	00	00	00	00	00	40	06	1D
Line 2	1	011	2F	33	47	1A	AB	E0	15	64
Line 3	0	001	00	11	22	33	44	55	66	77

# Review: Direct-mapped Cache Example

Memory



Cache



Assume 8 byte data blocks (fit 2 ints)

Access	tag	idx	off	h/m	val
rd 0x60	011=0x3	00	000	Miss	13
rd 0x64	011=0x3	00	100	Hit	14
rd 0x80	100=0x4	00	000	Miss	21
rd 0x64	011=0x3	00	100	Miss	14
rd 0x64	011=0x3	00	100	Hit	14
rd 0x60	011=0x3	00	000	Hit	13
rd 0x80	100=0x4	00	000	Miss	21

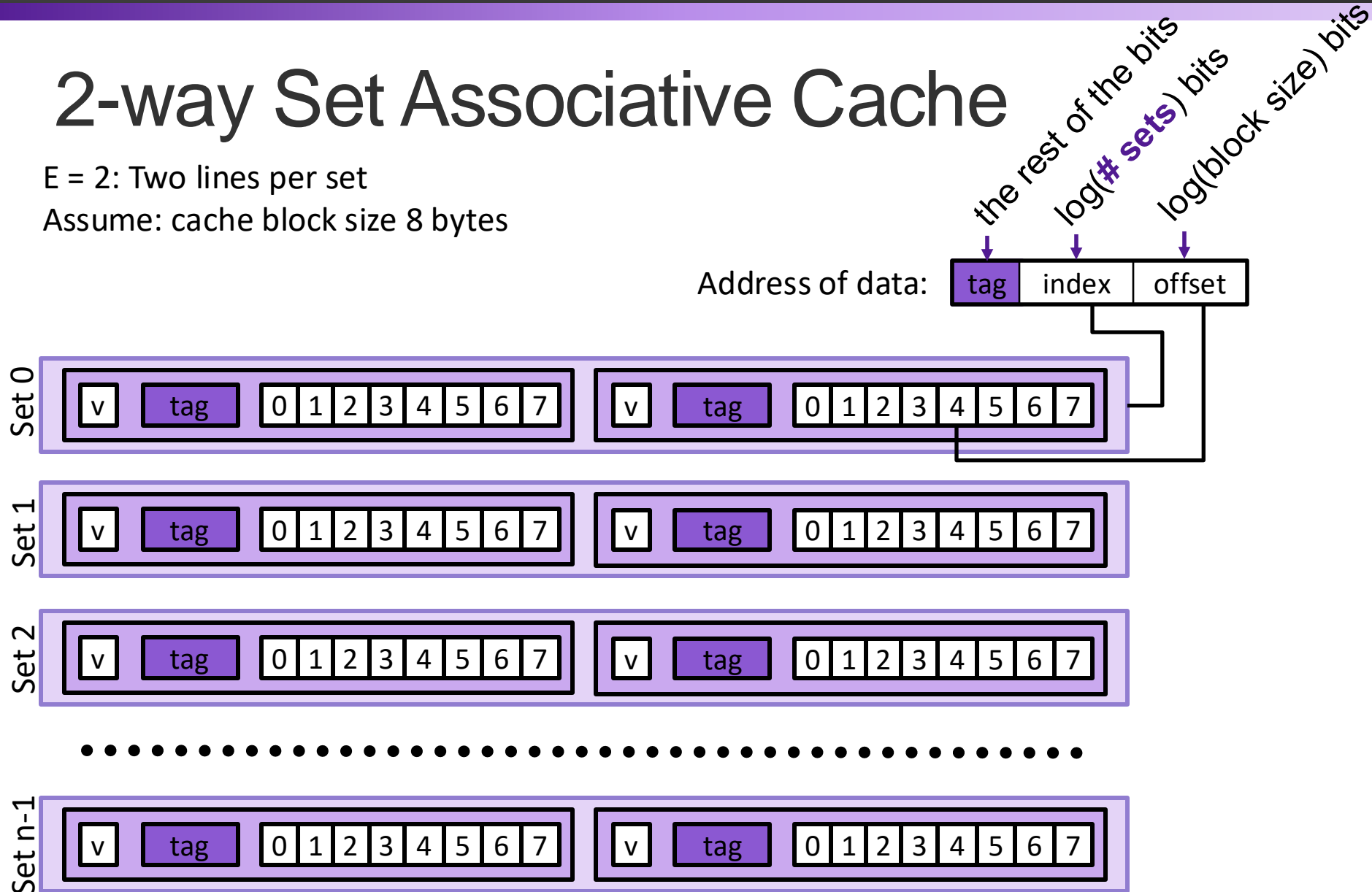
		Line 0		Line 1		Line 2		Line 3				
Time ↓	0	0	47 48	0	8	49 50	0	a	51 52	0	1	53 54
	1	3	13 14	0	8	49 50	0	a	51 52	0	1	53 54
	1	3	13 14	0	8	49 50	0	a	51 52	0	1	53 54
	1	4	21 22	0	8	49 50	0	a	51 52	0	1	53 54
	1	3	13 14	0	8	49 50	0	a	51 52	0	1	53 54
	1	3	13 14	0	8	49 50	0	a	51 52	0	1	53 54
	1	3	13 14	0	8	49 50	0	a	51 52	0	1	53 54
	1	3	13 14	0	8	49 50	0	a	51 52	0	1	53 54
	1	3	13 14	0	8	49 50	0	a	51 52	0	1	53 54
	1	4	21 22	0	8	49 50	0	a	51 52	0	1	53 54

How well does this take advantage of spacial locality?  
 How well does this take advantage of temporal locality?

# 2-way Set Associative Cache

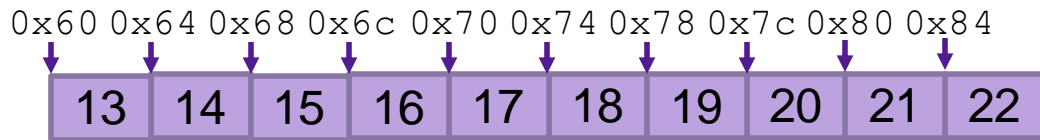
E = 2: Two lines per set

Assume: cache block size 8 bytes



# Review: Direct-mapped Cache Example

Memory



Cache



Assume 8 byte data blocks (fit 2 ints)

Access	tag	idx	off	h/m	val
rd 0x60	011=0x3	00	000	Miss	13
rd 0x64	011=0x3	00	100	Hit	14
rd 0x80	100=0x4	00	000	Miss	21
rd 0x64	011=0x3	00	100	Miss	14
rd 0x64	011=0x3	00	100	Hit	14
rd 0x60	011=0x3	00	000	Hit	13
rd 0x80	100=0x4	00	000	Miss	21

		Line 0		Line 1		Line 2		Line 3				
Time ↓	0	0	47 48	0	8	49 50	0	a	51 52	0	1	53 54
	1	3	13 14	0	8	49 50	0	a	51 52	0	1	53 54
	1	3	13 14	0	8	49 50	0	a	51 52	0	1	53 54
	1	4	21 22	0	8	49 50	0	a	51 52	0	1	53 54
	1	3	13 14	0	8	49 50	0	a	51 52	0	1	53 54
	1	3	13 14	0	8	49 50	0	a	51 52	0	1	53 54
	1	3	13 14	0	8	49 50	0	a	51 52	0	1	53 54
	1	3	13 14	0	8	49 50	0	a	51 52	0	1	53 54
	1	3	13 14	0	8	49 50	0	a	51 52	0	1	53 54
	1	4	21 22	0	8	49 50	0	a	51 52	0	1	53 54

How well does this take advantage of spacial locality?  
 How well does this take advantage of temporal locality?



# Eviction from the Cache

On a cache miss, a new block is loaded into the cache

- Direct-mapped cache: A valid block at the same location must be evicted—no choice
- Associative cache: If all blocks in the set are valid, one 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





# Caching and Writes

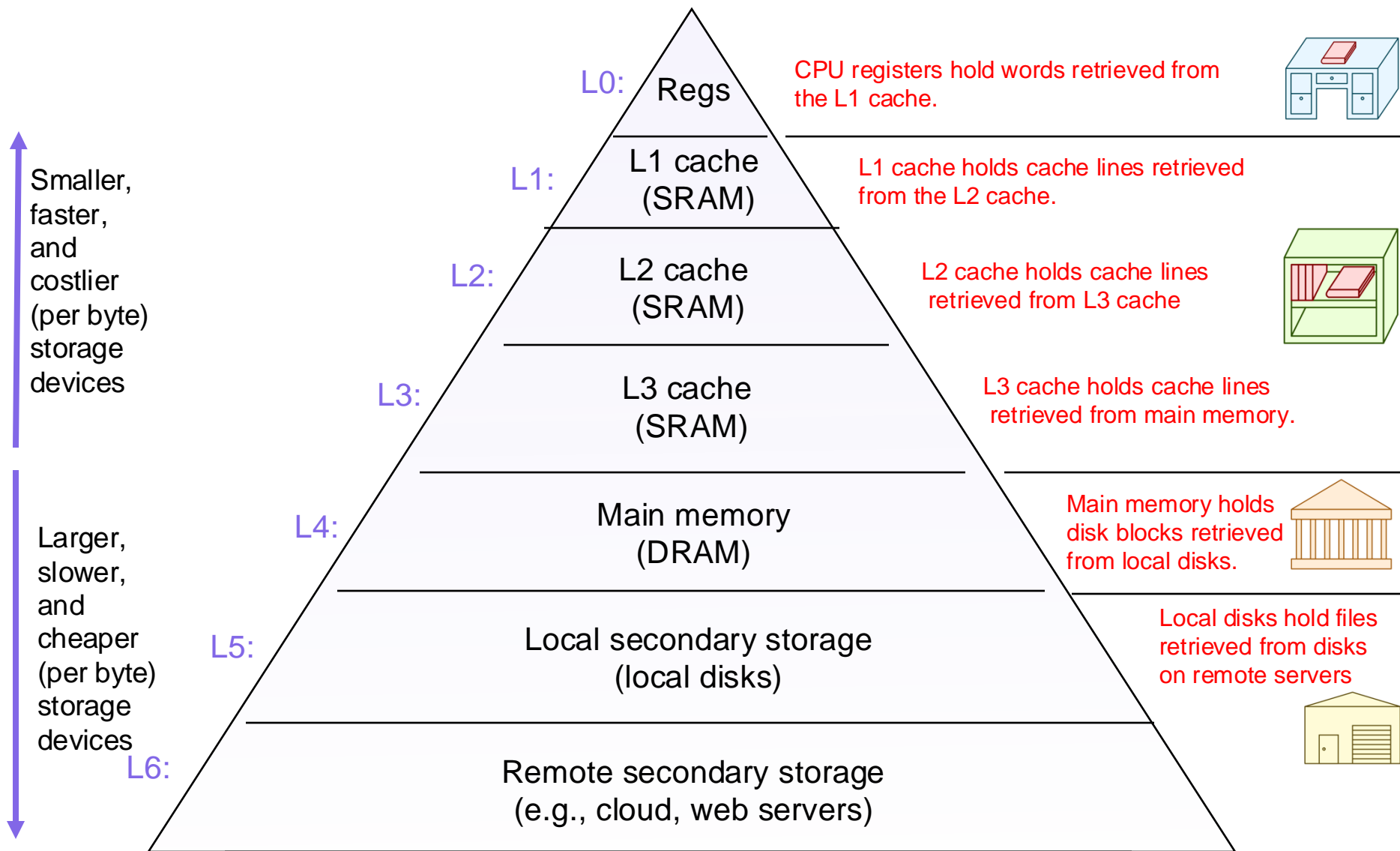
- What to do on a write-hit?
  - **Write-through:** write immediately to memory
  - **Write-back:** defer write to memory until replacement of line
    - Need a dirty bit (line different from memory or not)
- What to do on a write-miss?
  - **Write-allocate:** load into cache, update line in cache
    - Good if more writes to the location follow
  - **No-write-allocate:** writes straight to memory, does not load into cache
- Typical
  - Write-through + No-write-allocate
  - Write-back + Write-allocate



# Caching Organization Summarized

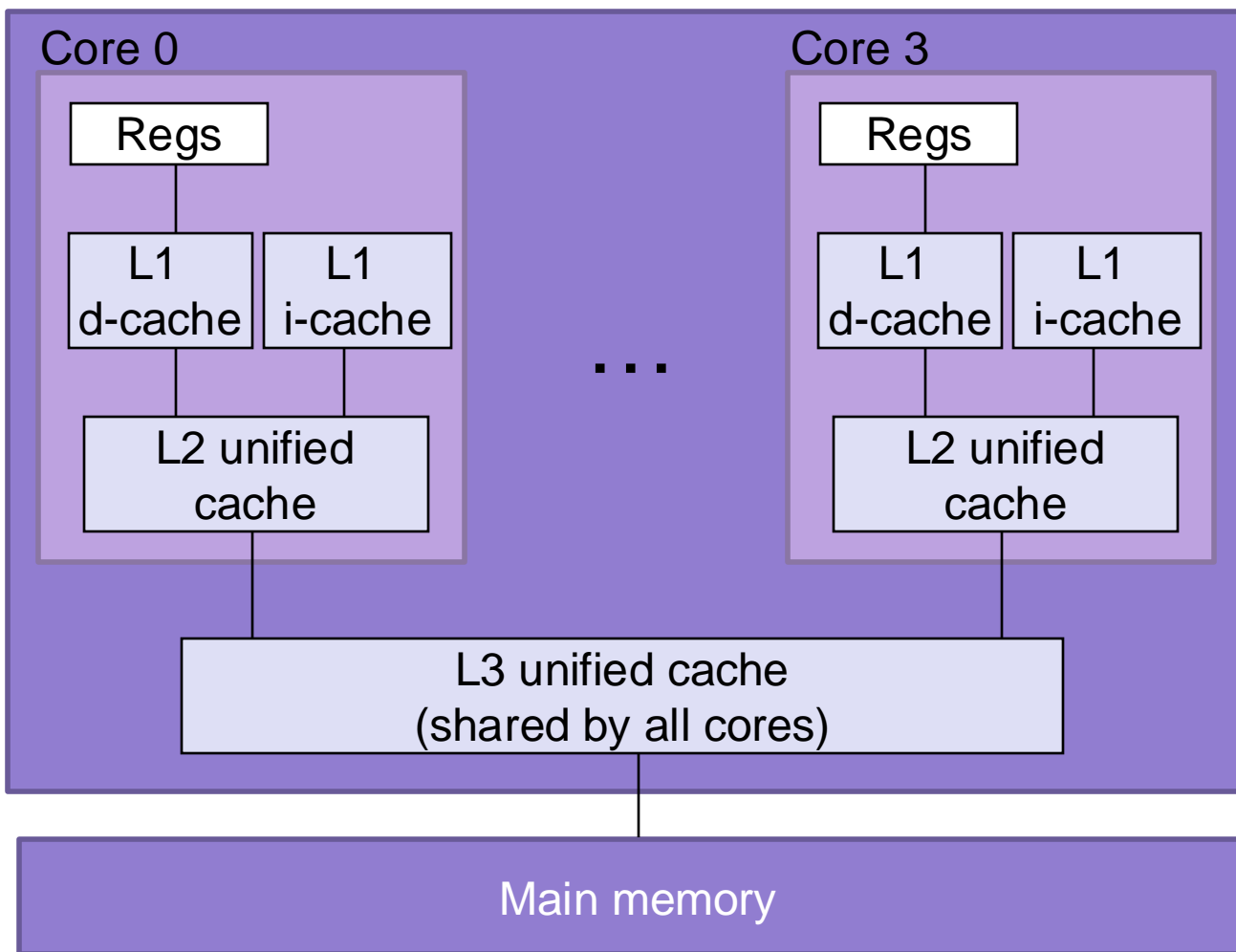
- A cache consists of lines
- A **line** contains
  - A **block** of bytes, the data values from memory
  - A **tag**, indicating where in memory the values are from
  - A **valid bit**, indicating if the data are valid
- Lines are organized into sets
  - **Direct-mapped cache**: one line per set
  - **k-way associative cache**: k lines per set
  - **Fully associative cache**: all lines in one set
- Caches handle both reads and writes
  - **write-through**: write to both cache and memory
  - **write-back**: write only to cache, write to memory on evict,
  - **write-allocate**: alloc on any miss
  - **no-write allocate**: alloc only on read miss

# Memory Hierarchy



# Typical Intel Core i7 Hierarchy

Processor package



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.