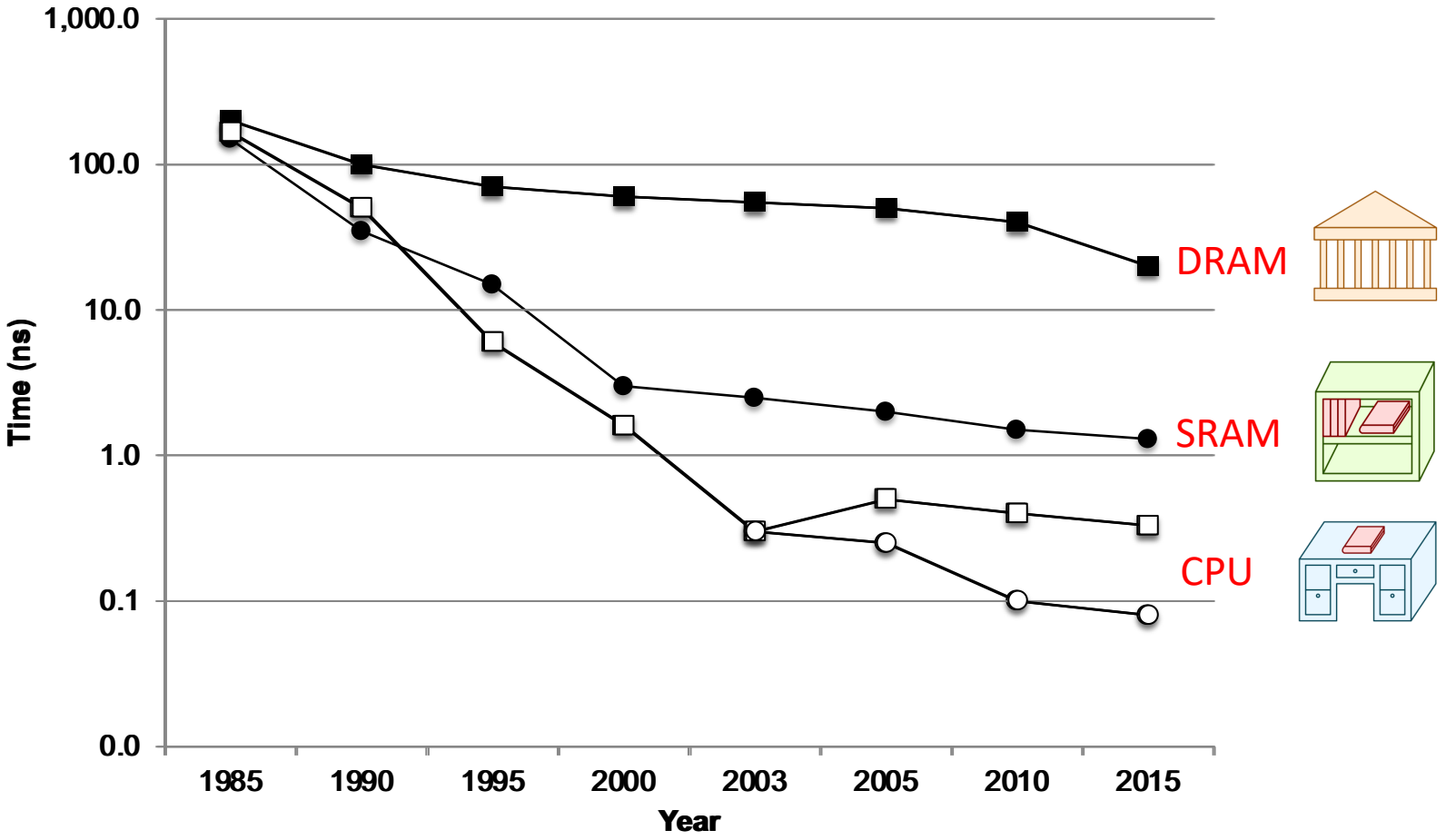
Lecture 12: Caches (cont'd)

CS 105

Spring 2024

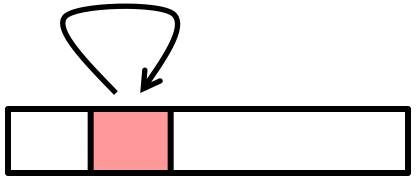
Review: The CPU-Memory Gap



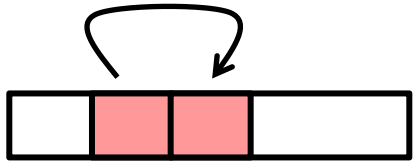
Review: Principle of Locality

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

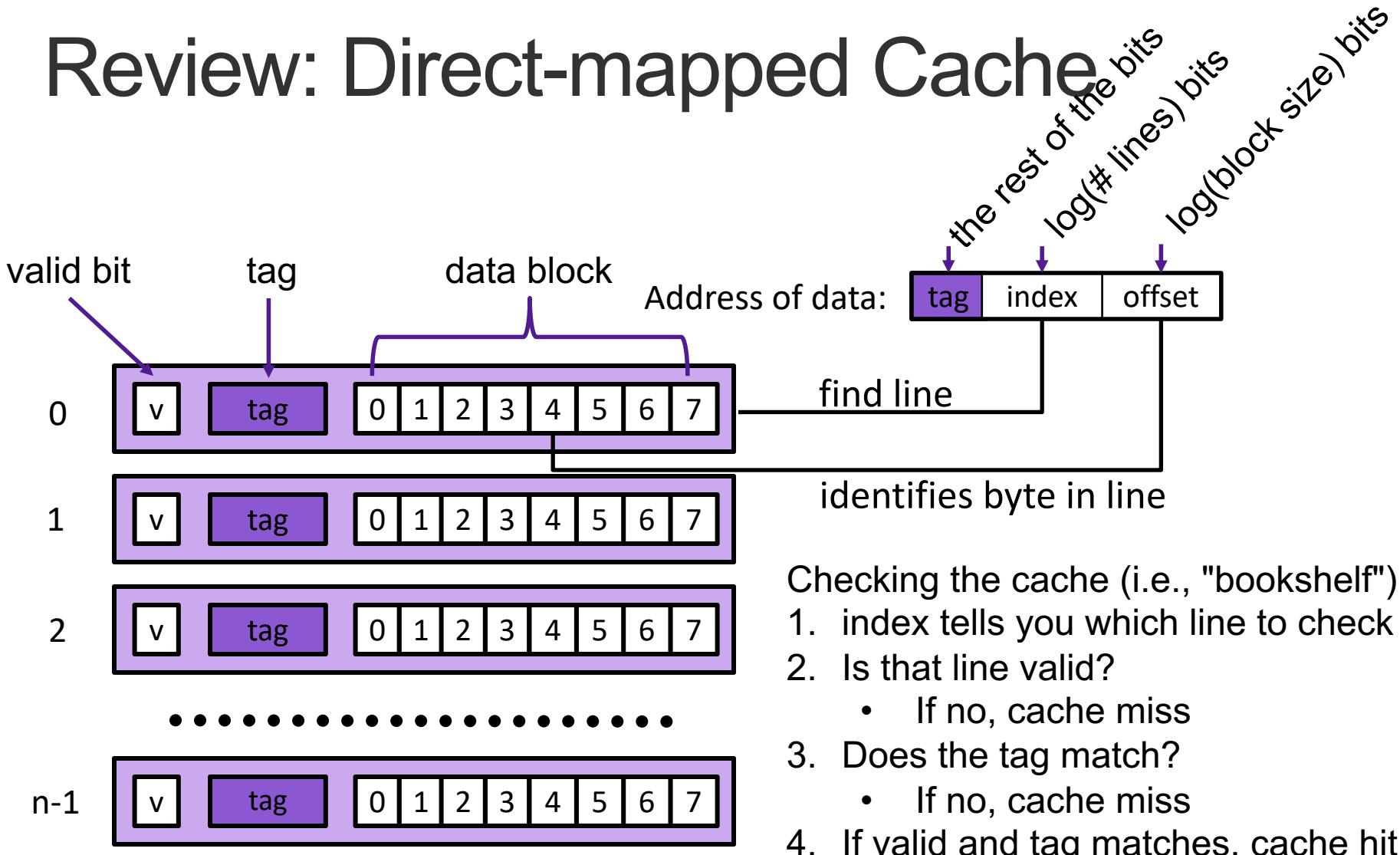
- Temporal locality:
  - Recently referenced items are likely to be referenced again in the near future



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



Review: Direct-mapped Cache



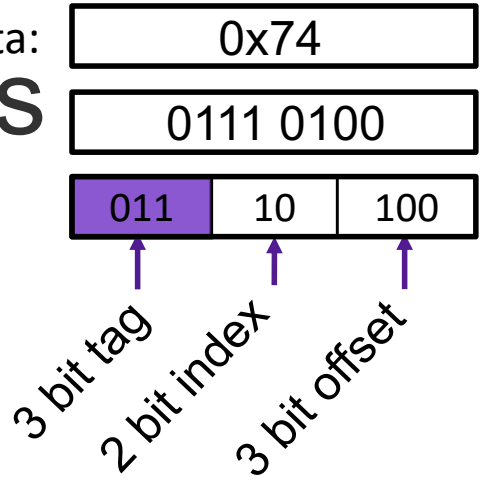
Checking the cache (i.e., "bookshelf"):

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. Set valid bit to 1
2. Update tag
3. Replace data block with bytes from memory



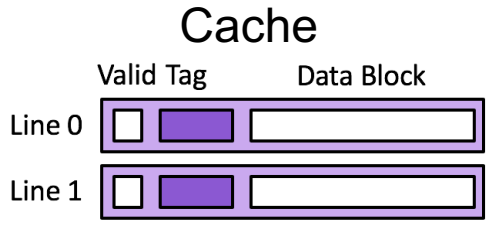
Line 0	1	110	0F	12	AB	34	FF	FF	EA	68
Line 1	1	001	00	00	00	00	00	40	06	1D
Line 2	1	011	2F	33	47	1A	AB	E0	15	64
Line 3	0	001	00	11	22	33	44	55	66	77

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

Review: Direct-mapped Cache

0x74	18
0x70	17
0x6c	16
0x68	15
0x64	14
0x60	13

Memory



Access	tag	idx	off	h/m
rd 0x60	0110	0	000	Miss
rd 0x64	0110	0	100	Hit
rd 0x70	0111	0	000	Miss
rd 0x64	0110	0	100	Miss
rd 0x64	0110	0	100	Hit
rd 0x60	0110	0	000	Hit
rd 0x70	0111	0	000	Miss

Time

Assume 8 byte data blocks

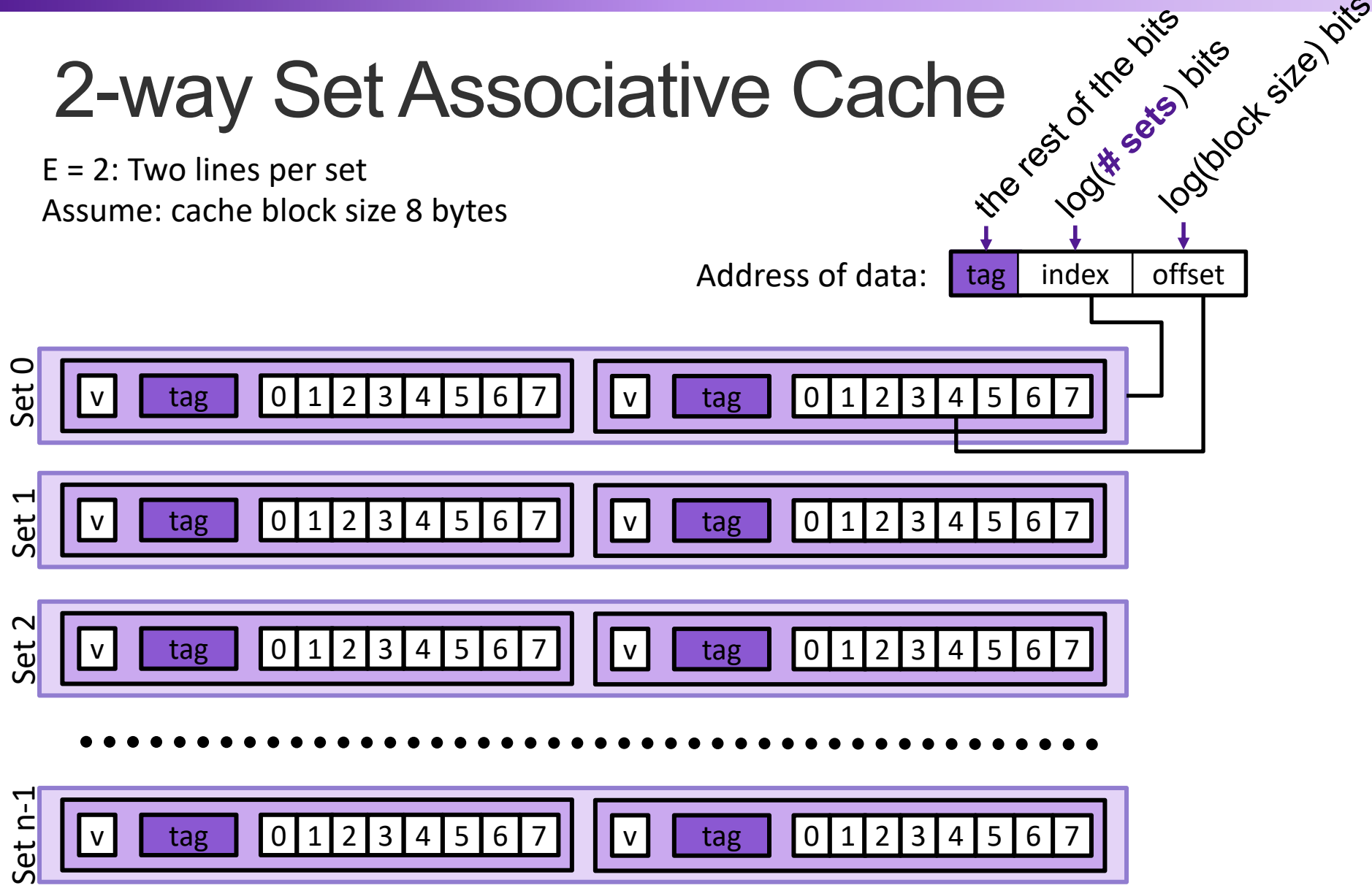
Line 0				Line 1			
0	0000	47	48	0	0000	47	48
1	0110	13	14				
1	0111	17	18				
1	0110	13	14				
1	0111	17	18				

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

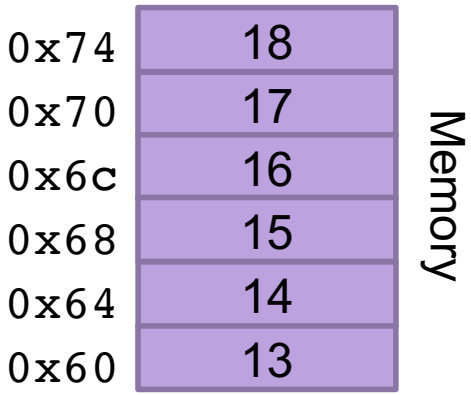
2-way Set Associative Cache

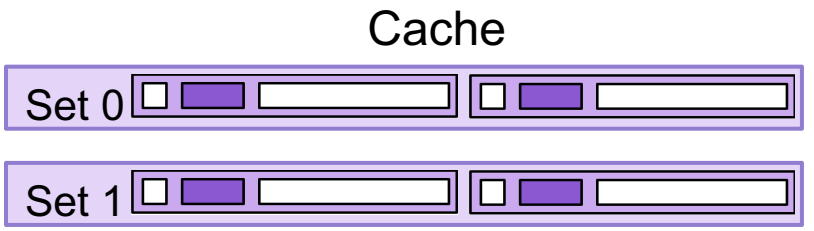
E = 2: Two lines per set

Assume: cache block size 8 bytes



Exercise: 2-way Set Associative Cache





Assume 8 byte data blocks

Access	tag	idx	off	h/m
rd 0x60				
rd 0x64				
rd 0x70				
rd 0x64				
rd 0x64				
rd 0x60				
rd 0x70				

Time

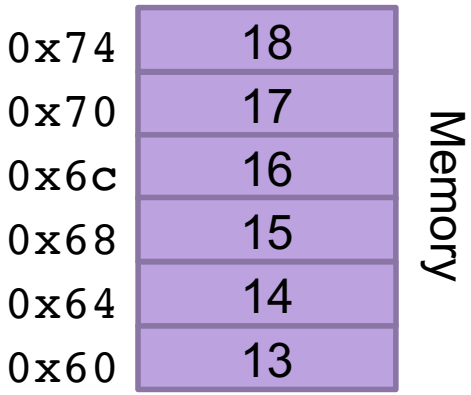
Set 0						Set 1					
Line 0			Line 1			Line 0			Line 1		
0	0	47 48	0	1	47 48	0	0	47 48	0	1	47 48

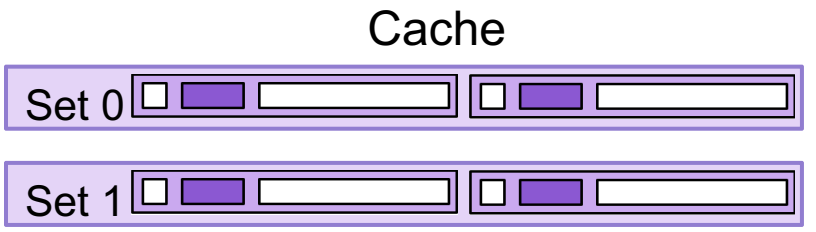
Eviction from the Cache

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

Exercise: 2-way Set Associative Cache





Assume 8 byte data blocks

Access	tag	idx	off	h/m
rd 0x60				
rd 0x64				
rd 0x70				
rd 0x64				
rd 0x64				
rd 0x60				
rd 0x70				
rd 0x80				

Time

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

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

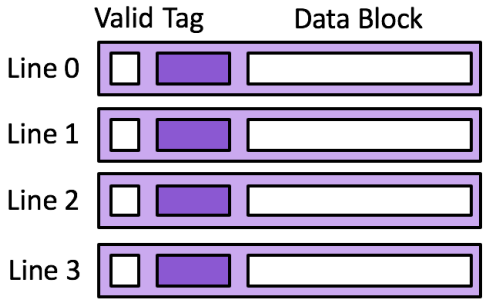
Exercise: Write-back + Write-allocate

Memory

Access	tag	idx	off	h/m
rd 0x10				
wr 8,0x10				
wr 9,0x24				
rd 0x24				
rd 0x20				

Access	tag	idx	off	h/m
rd 0x10	0001	00	00	m
wr 8,0x10	0001	00	00	h
wr 9,0x24	0010	01	00	m
rd 0x24	0010	01	00	h
rd 0x20	0010	00	00	m

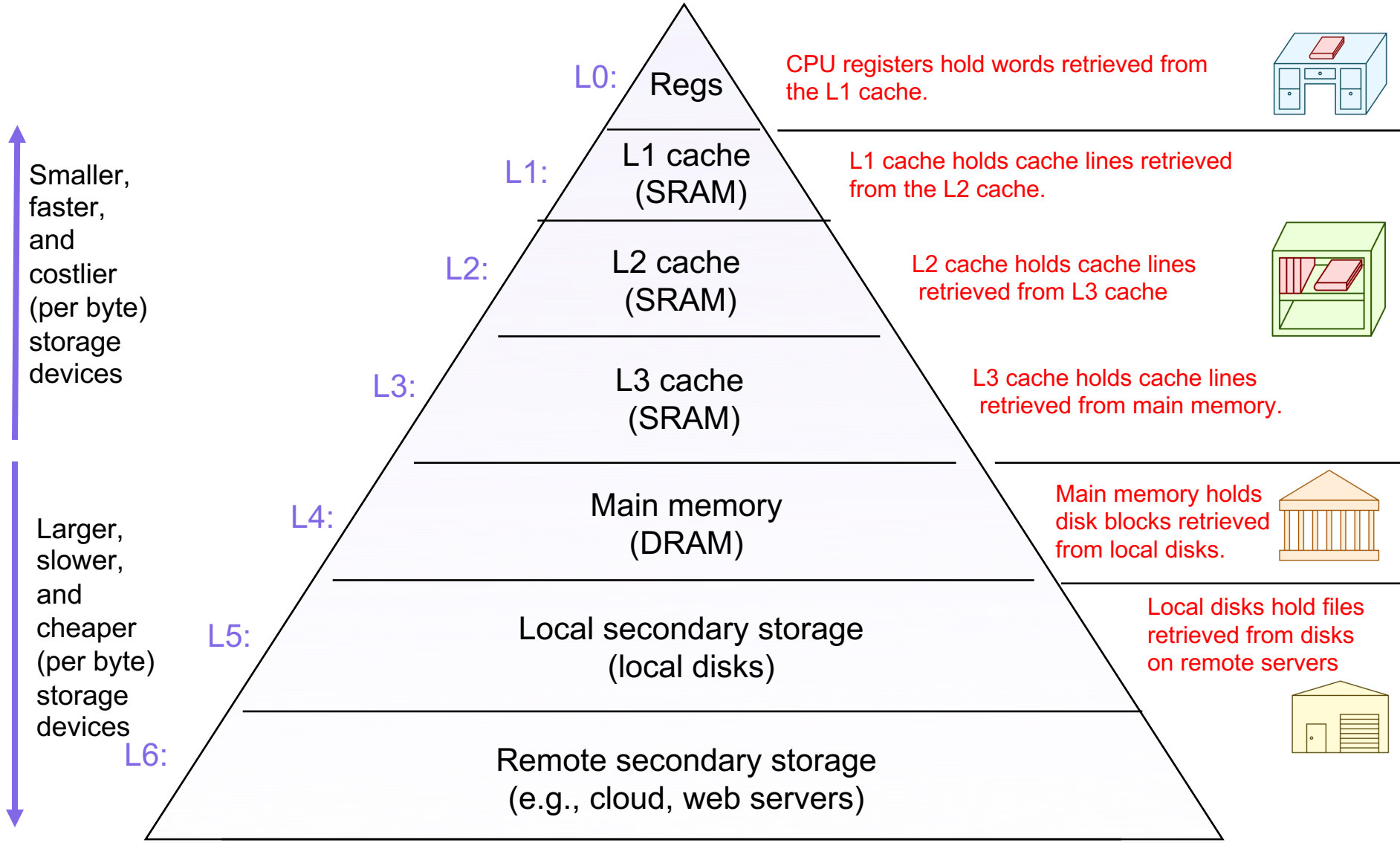
Cache



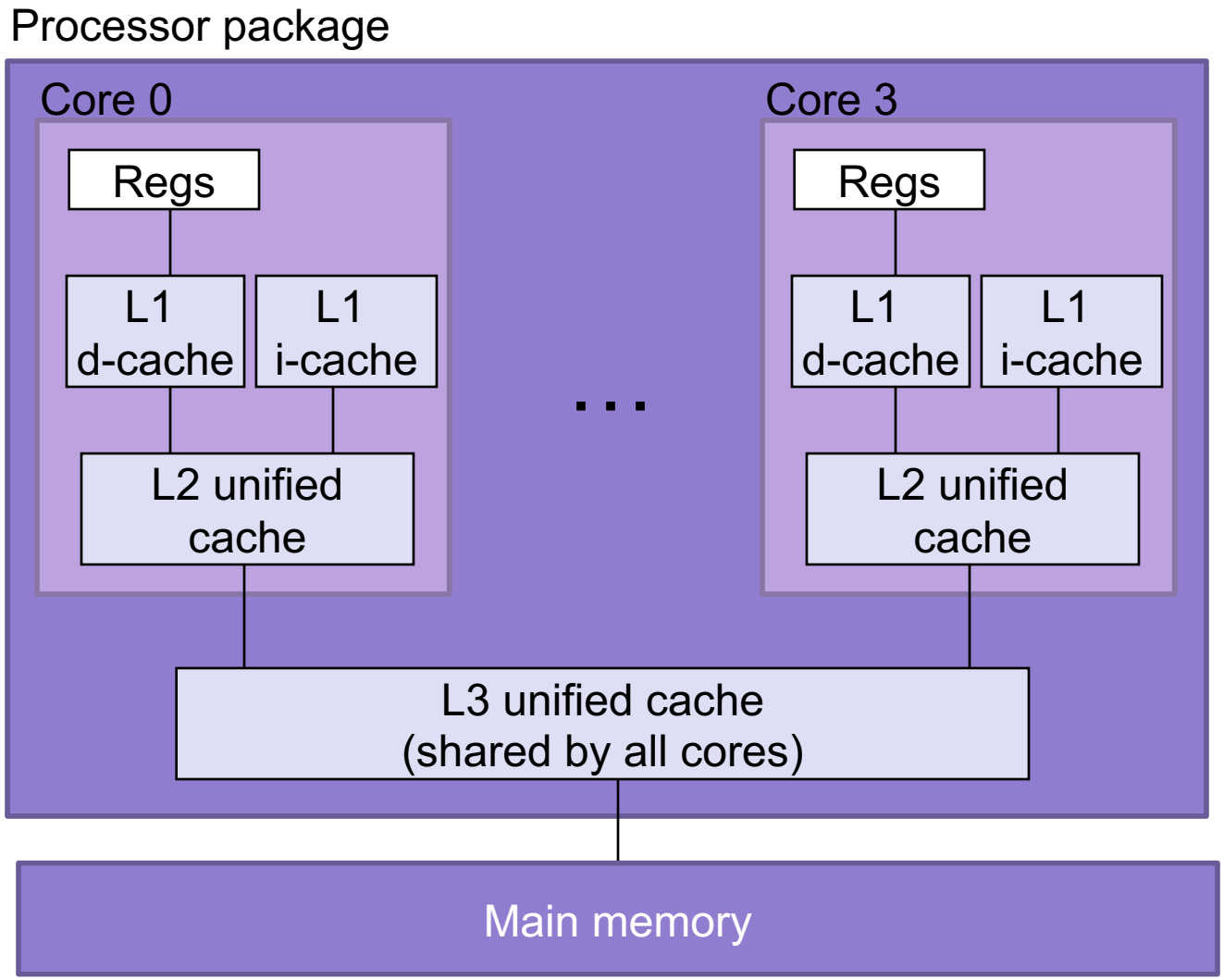
Assume 4 byte data blocks

Line 0			Line 1			Line 2			Line 3			W
0	0	47	0	1	47	0	2	47	0	3	47	

Memory Hierarchy



Typical Intel Core i7 Hierarchy



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.

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