



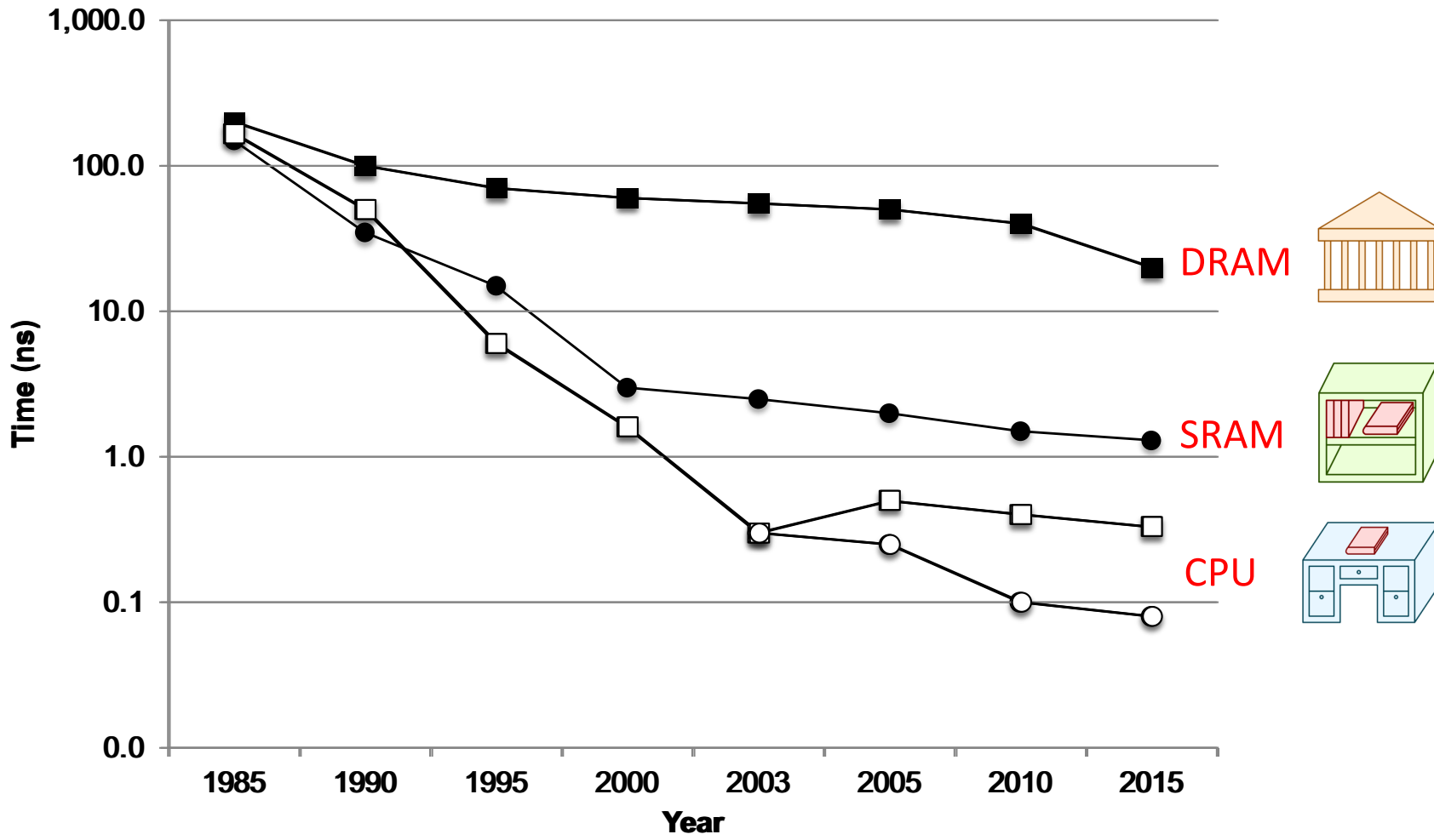
# Lecture 12: Caches (cont'd)

---

CS 105

Fall 2023

# Review: The CPU-Memory Gap

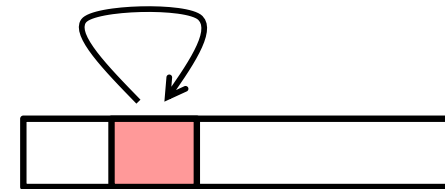


# Review: Principle of Locality

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

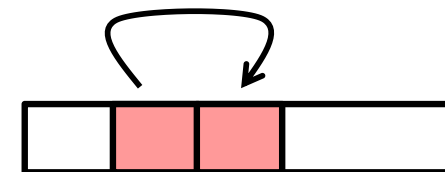
- ▶ **Temporal locality:**

- ▶ Recently referenced items are likely to be referenced again in the near future

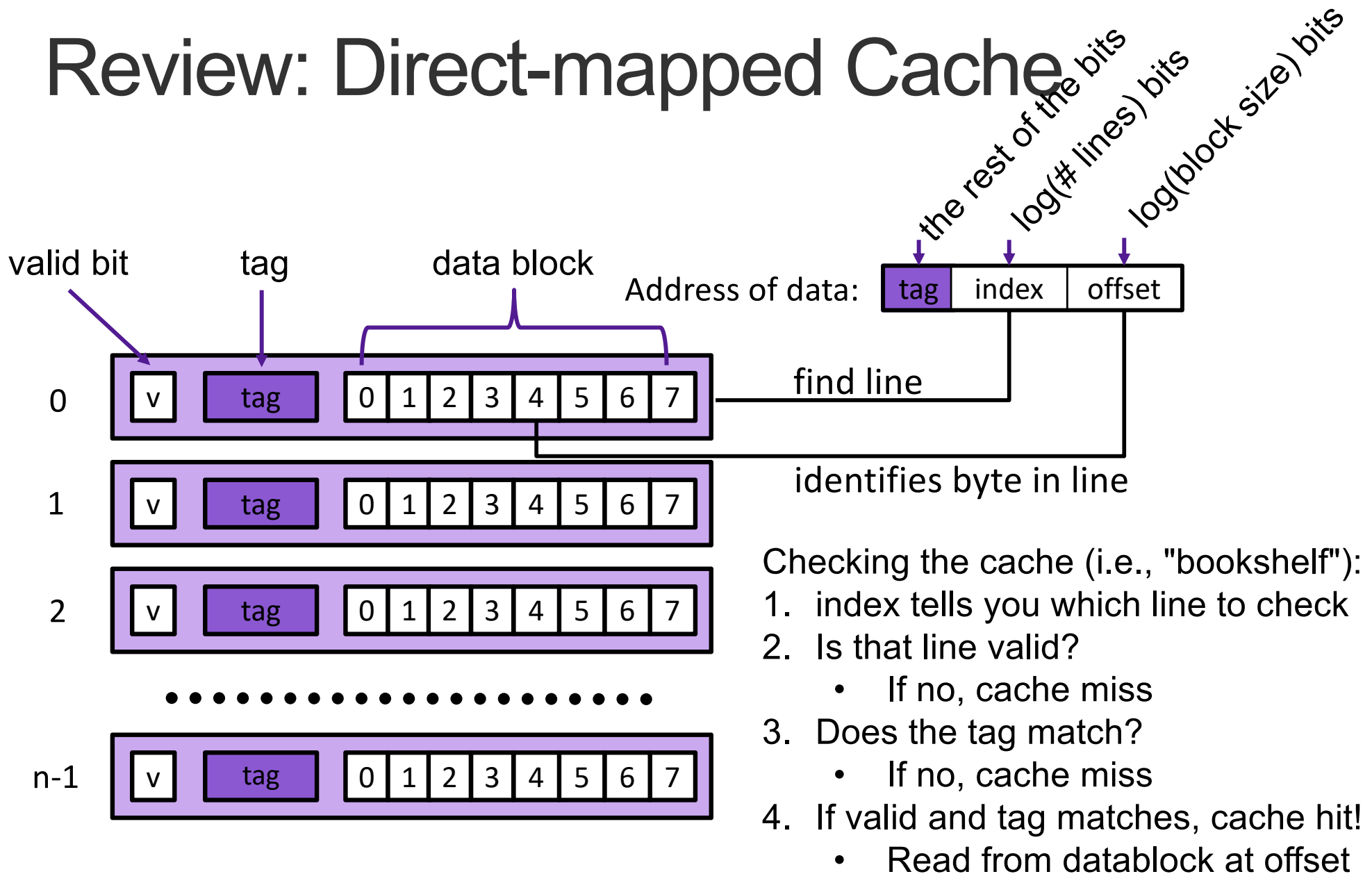


- ▶ **Spatial locality:**

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



# Review: Direct-mapped Cache



# Review: Handling Cache Miss

Address of data:

0x74

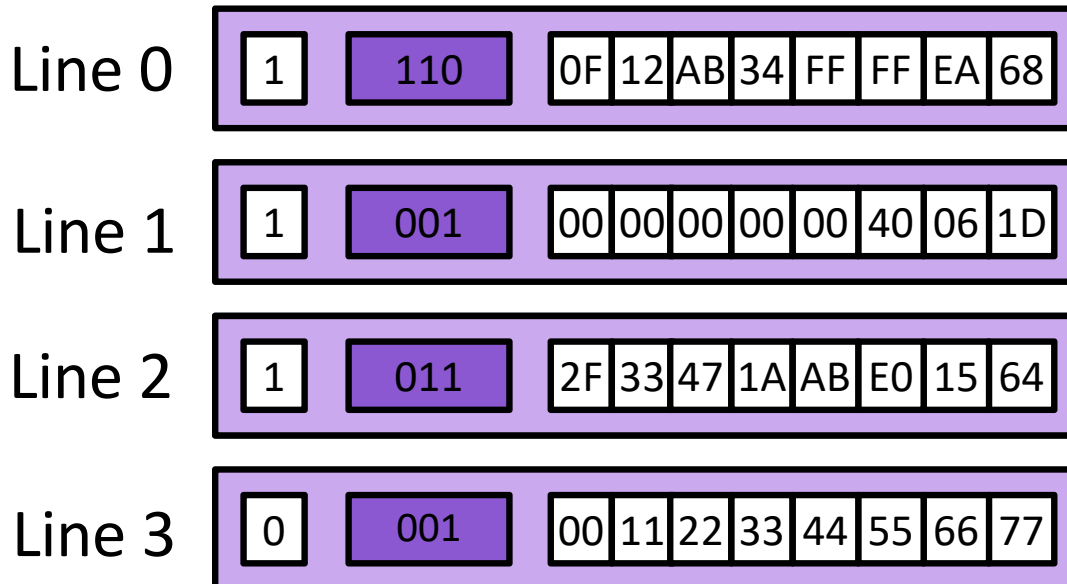
0111 0100

011 10 100

3 bit tag  
2 bit index  
3 bit offset

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



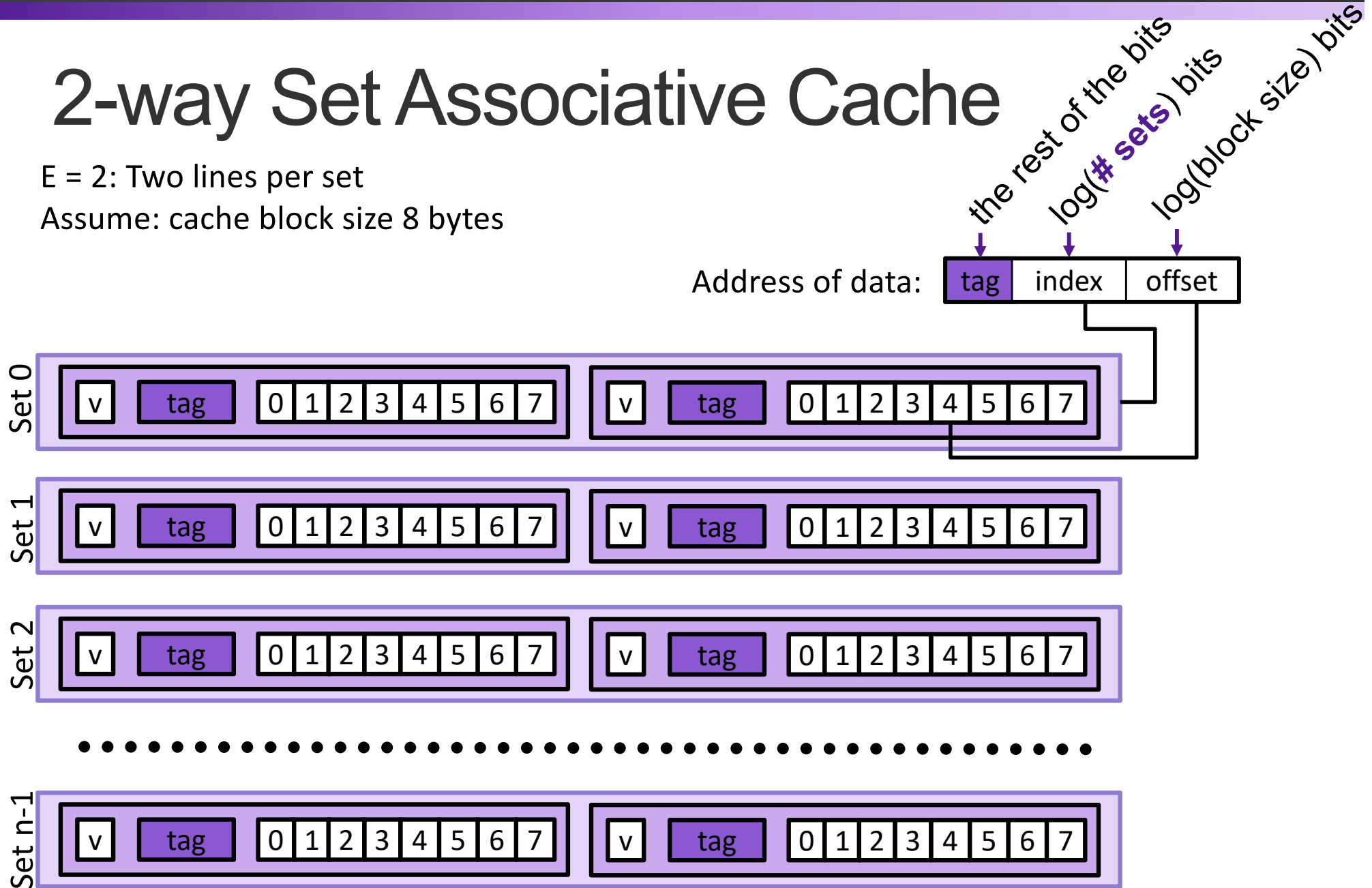
0x79	23
0x78	B7
0x77	64
0x76	15
0x75	E0
0x74	AB
0x73	1A
0x72	47
0x71	33
0x70	2F
0x6F	0A
0x6E	00



# 2-way Set Associative Cache

E = 2: Two lines per set

Assume: cache block size 8 bytes







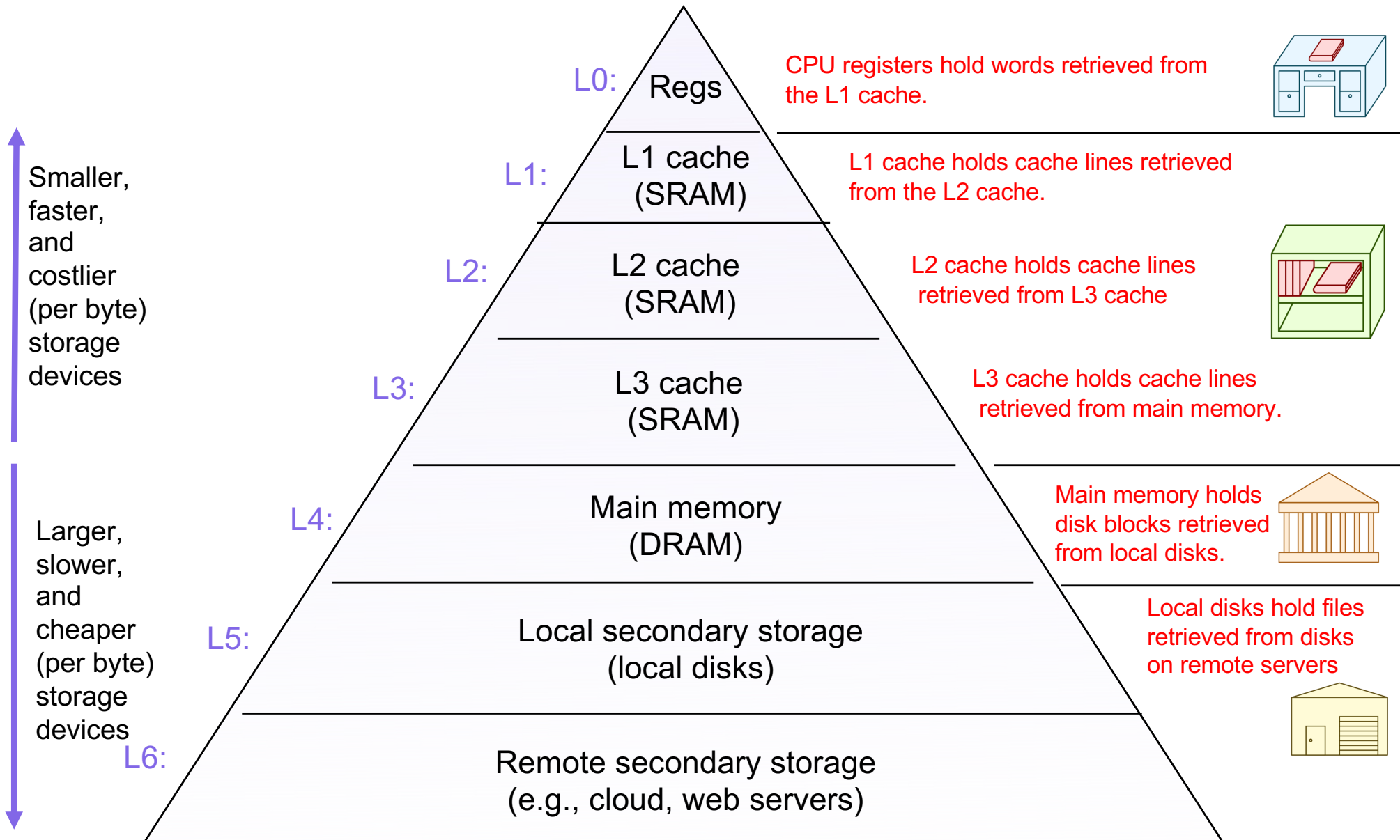
# Eviction from the Cache

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

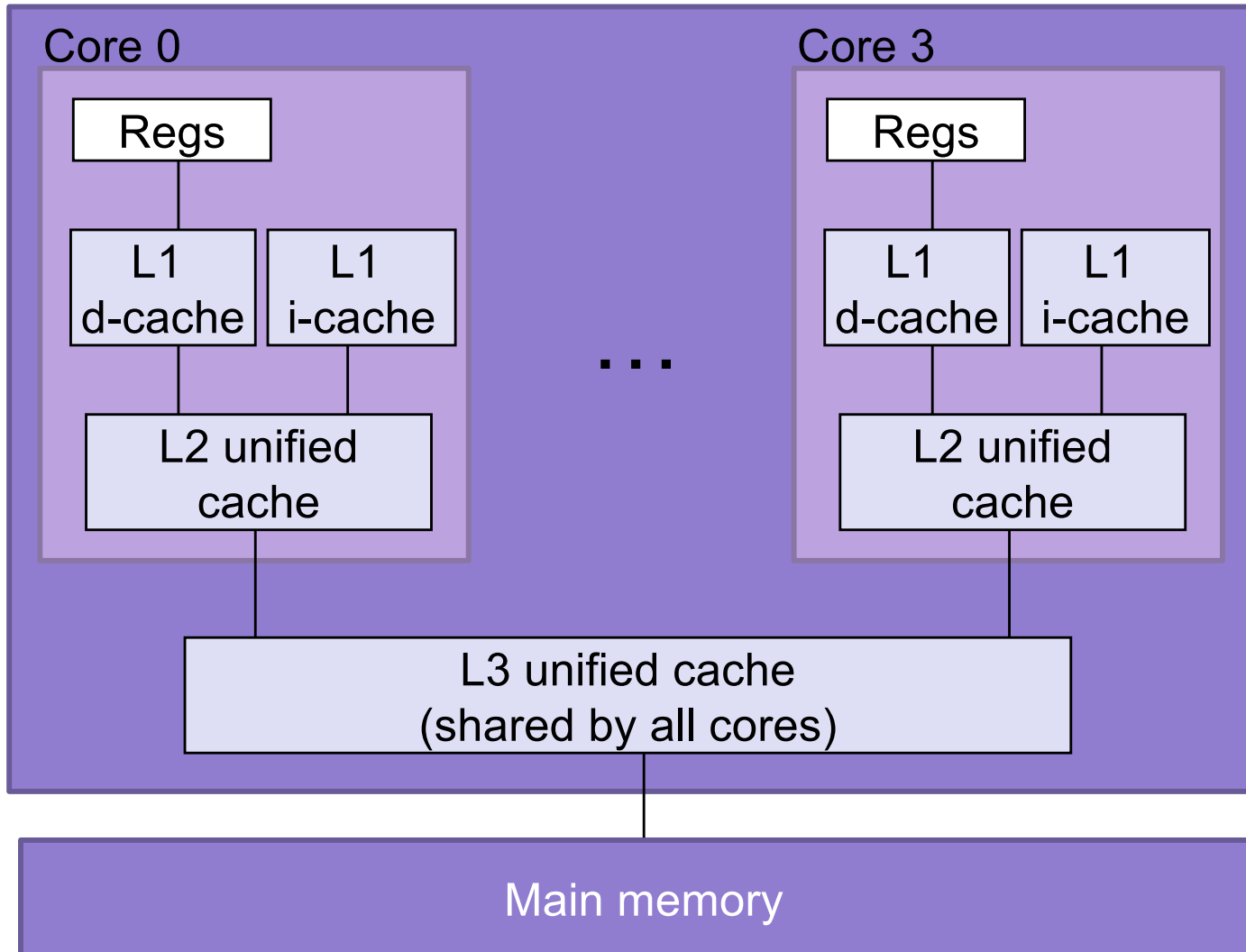


# Memory Hierarchy



# 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.

# 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