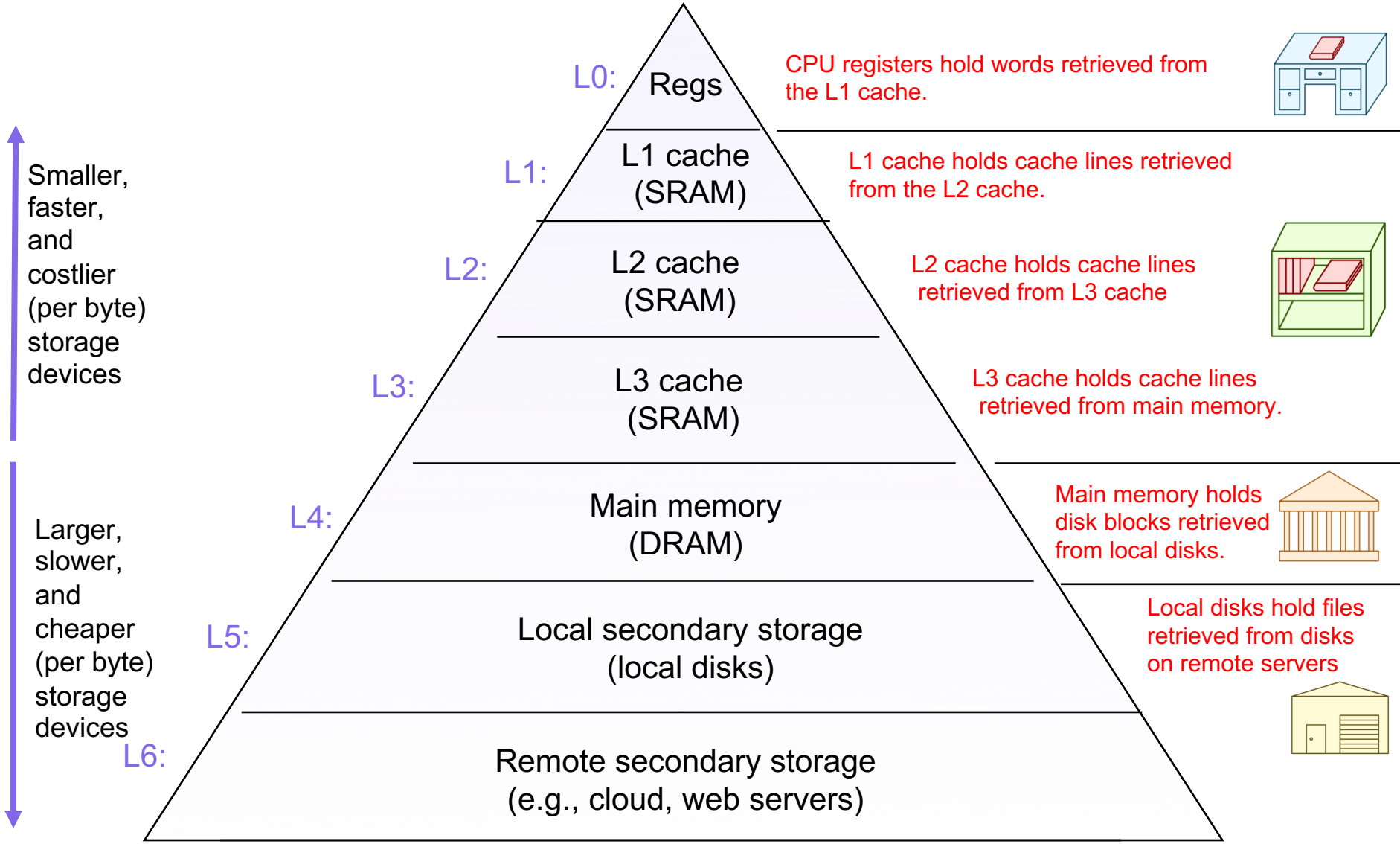
Lecture 13: Caches (cont'd)

CS 105

March 6, 2019

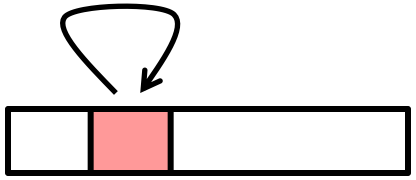
Memory Hierarchy



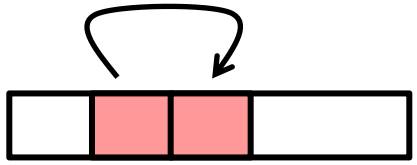
Principle of Locality

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

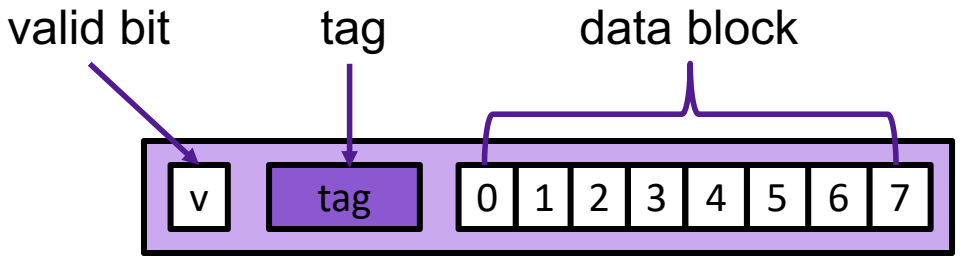
- Temporal locality:
  - Recently referenced items are likely to be referenced again in the near future



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

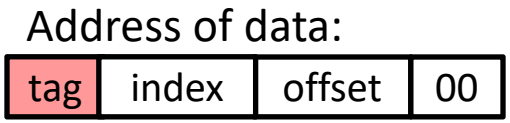


Cache Lines



- **data block:** cached data
- **tag:** uniquely identifies which data is stored in the cache line
- **valid bit:** indicates whether or not the line contains meaningful information

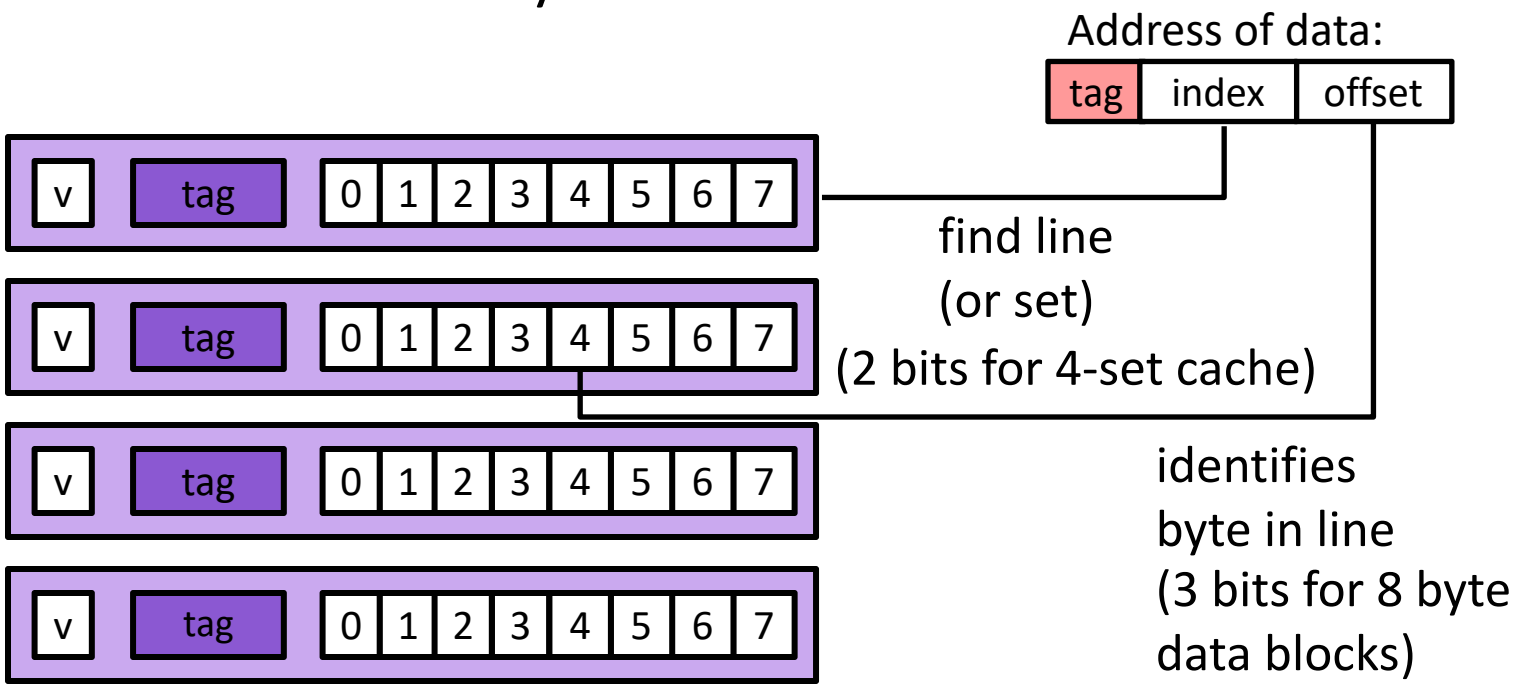
Caching—The Organization



- An address is decomposed into three parts
  - Low-order b bits, providing an offset into a block (2^b is the data block size)
  - Middle s bits, indicating which set in the cache to search (2^s is the number of sets)
  - Upper remaining bits, the tag to be matched

Direct-mapped Cache

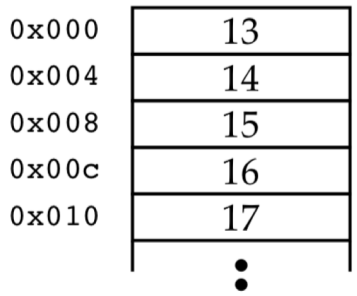
Assume: cache block size 8 bytes



Example: Direct-Mapped Cache

Dynamic Transaction Stream

```
rd 0x000
rd 0x004
rd 0x010
rd 0x000
rd 0x004
```



Assume 4-byte data block

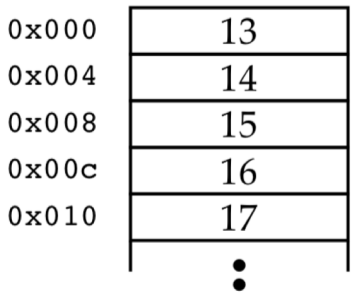
	tag	idx	h/m	Set 0	Set 1	Set 2	Set 3
rd 0x000							
rd 0x004							
rd 0x010							
rd 0x000							
rd 0x004							
rd 0x020							

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

Exercise: Direct-Mapped Cache

Dynamic Transaction Stream

```
rd 0x000
rd 0x004
rd 0x010
rd 0x000
rd 0x004
```



Assume 8-byte data block

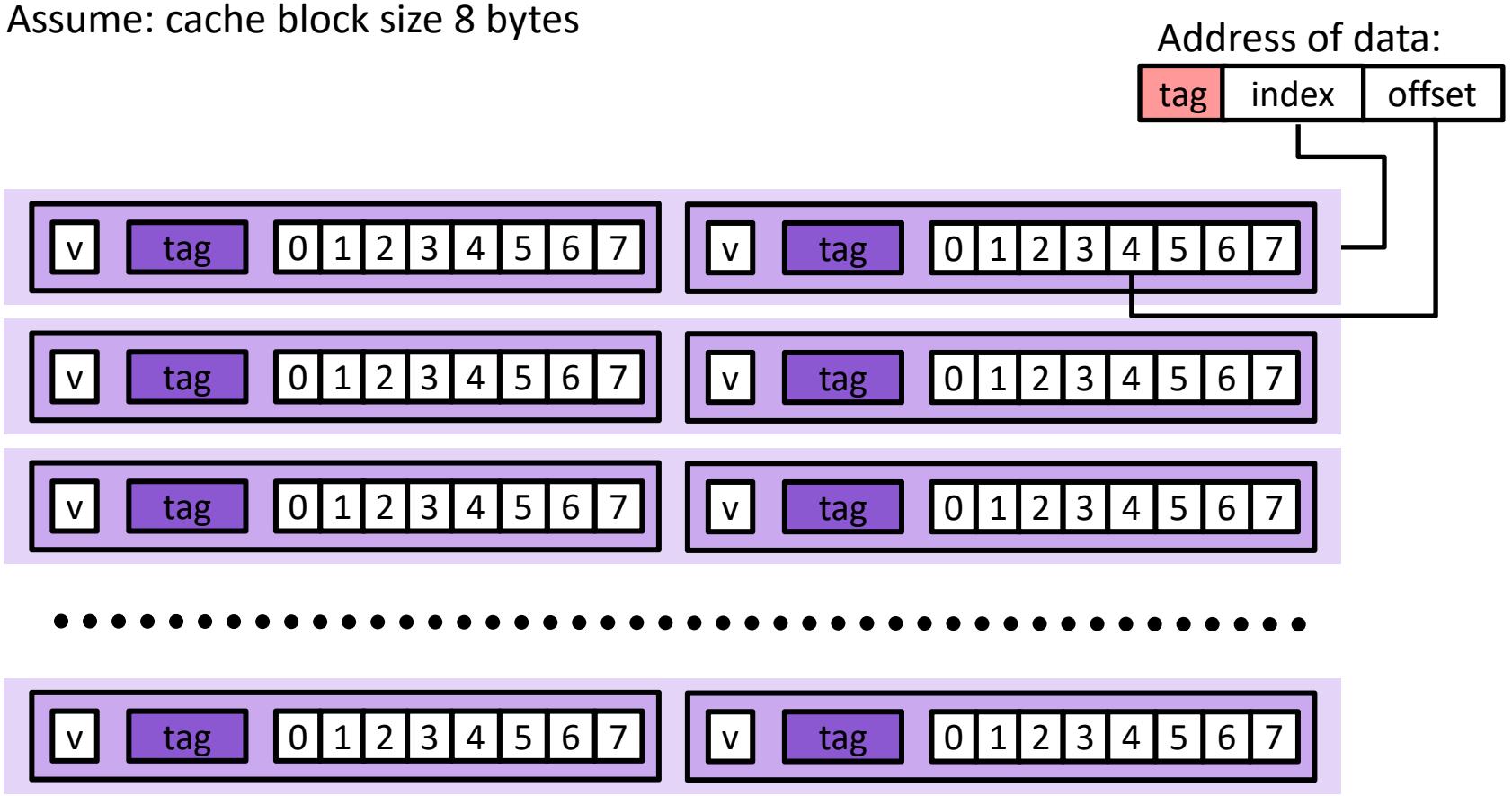
	tag	idx	h/m	Set 0	Set 1
rd 0x000					
rd 0x004					
rd 0x010					
rd 0x000					
rd 0x004					
rd 0x020					

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

2-way Set Associative Cache

E = 2: Two lines per set

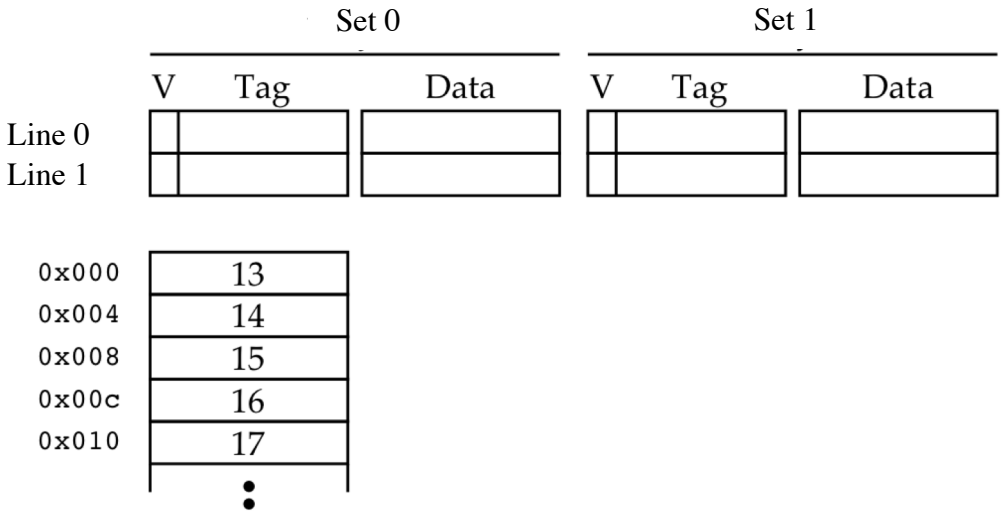
Assume: cache block size 8 bytes



Exercise: 2-way Set Associative Cache

Dynamic Transaction Stream

```
rd 0x000
rd 0x004
rd 0x010
rd 0x000
rd 0x004
```



	tag	idx	h/m	Set 0		Set 1	
				Line 0	Line 1	Line 0	Line 1
`rd 0x000`							
`rd 0x004`							
`rd 0x010`							
`rd 0x000`							
`rd 0x004`							
`rd 0x020`							

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
  - Least-recently used policy; requires extra data in each set
  - Random policy

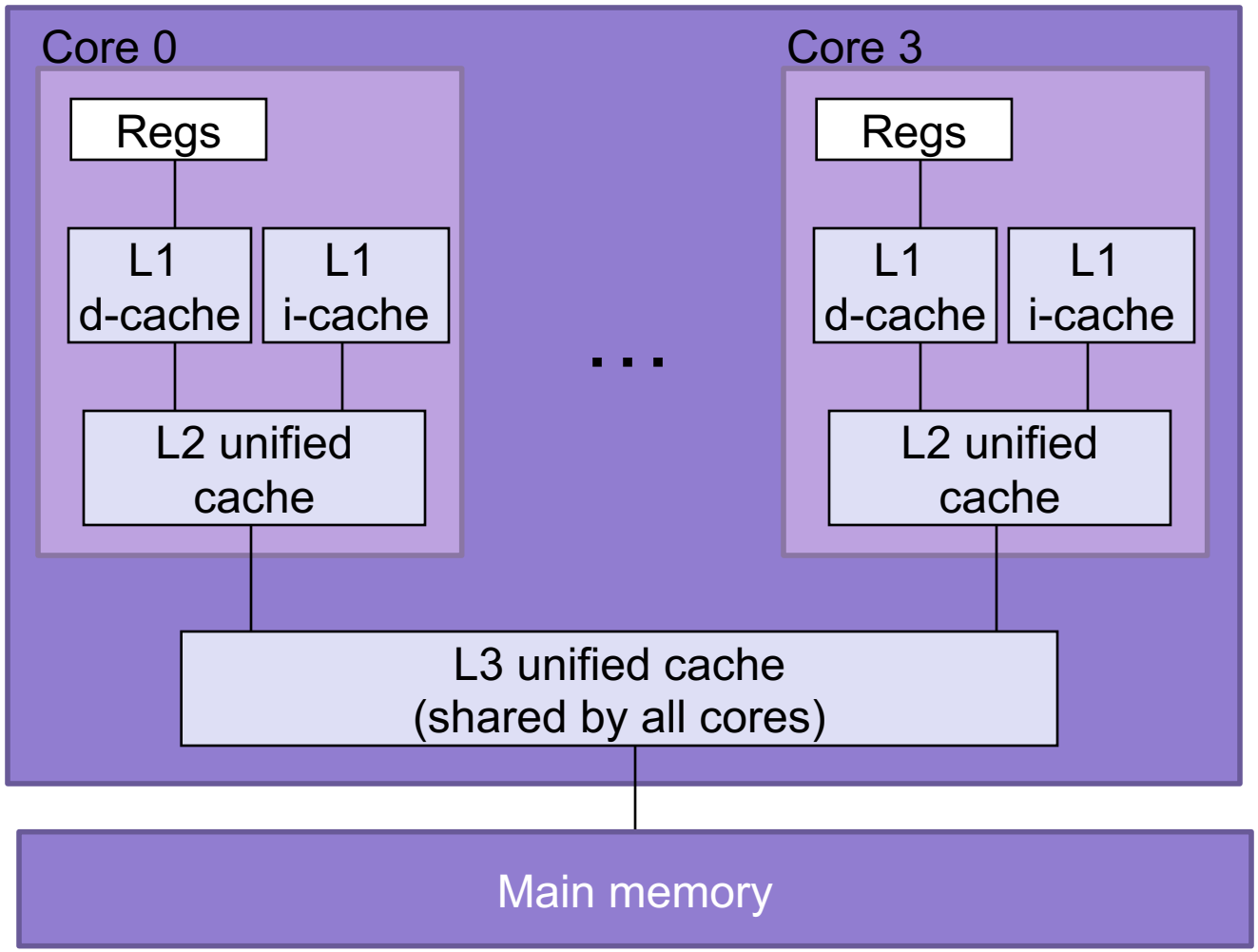
Caching Organization Summarized

- 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

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**

Typical Intel Core i7 Hierarchy



L1 i-cache and d-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.