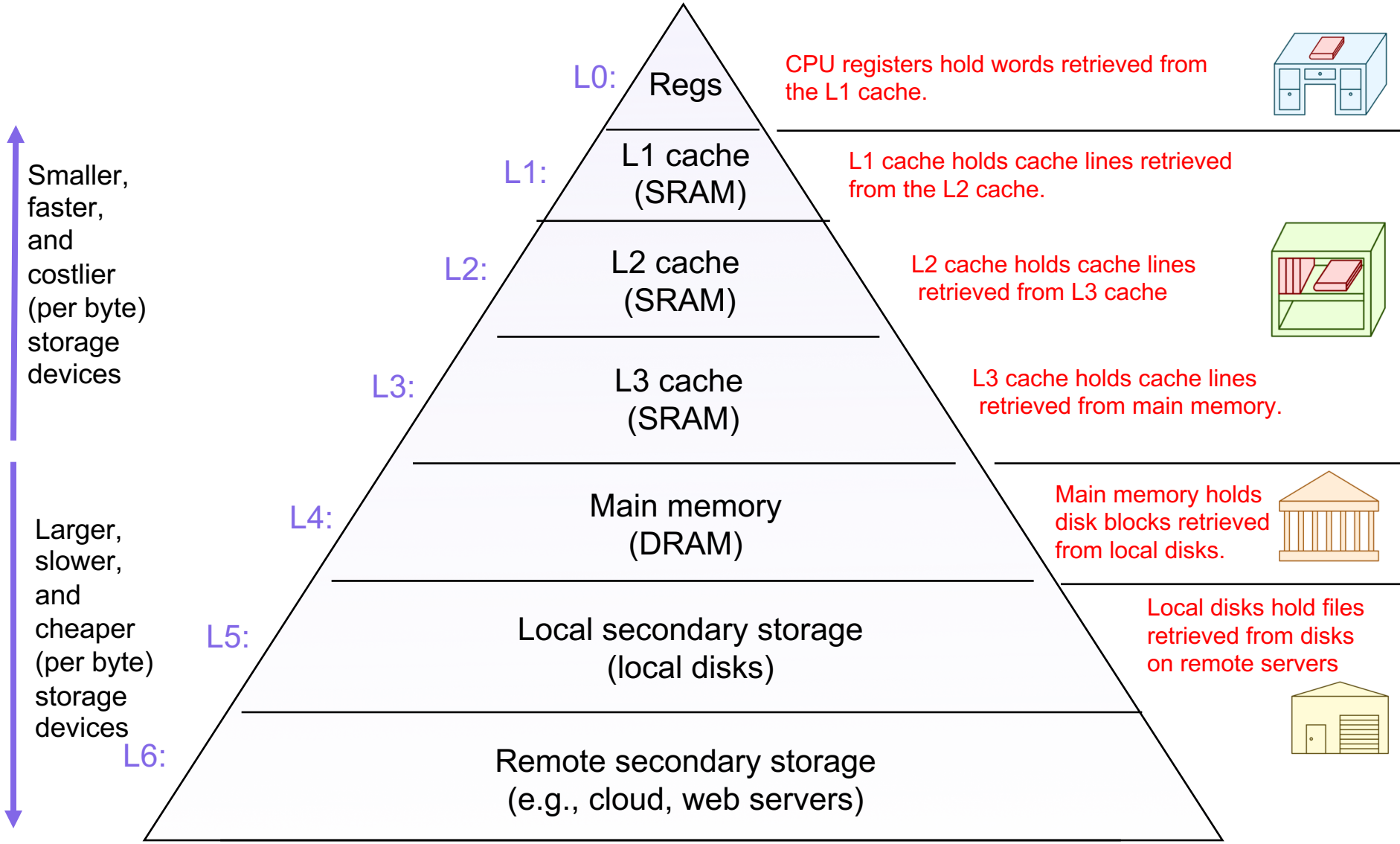
Lecture 11: Caches (cont'd)

CS 105

October 10, 2019

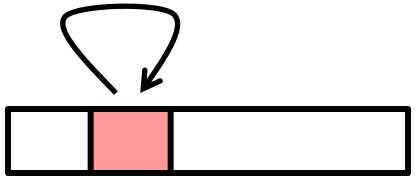
Memory Hierarchy



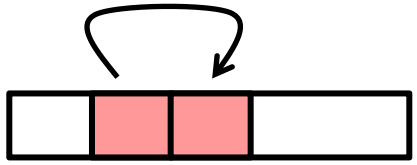
Principle of Locality

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

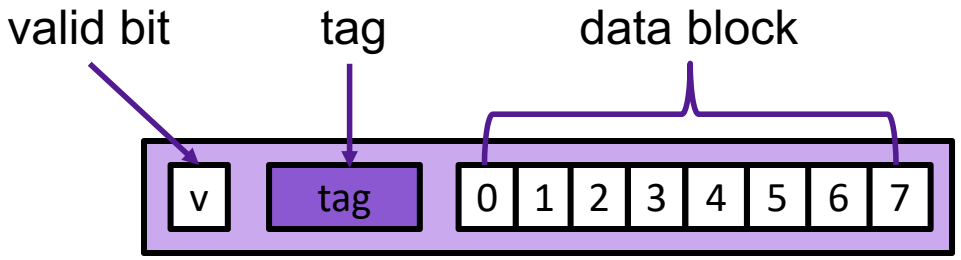
- Temporal locality:
  - Recently referenced items are likely to be referenced again in the near future



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



Cache Lines

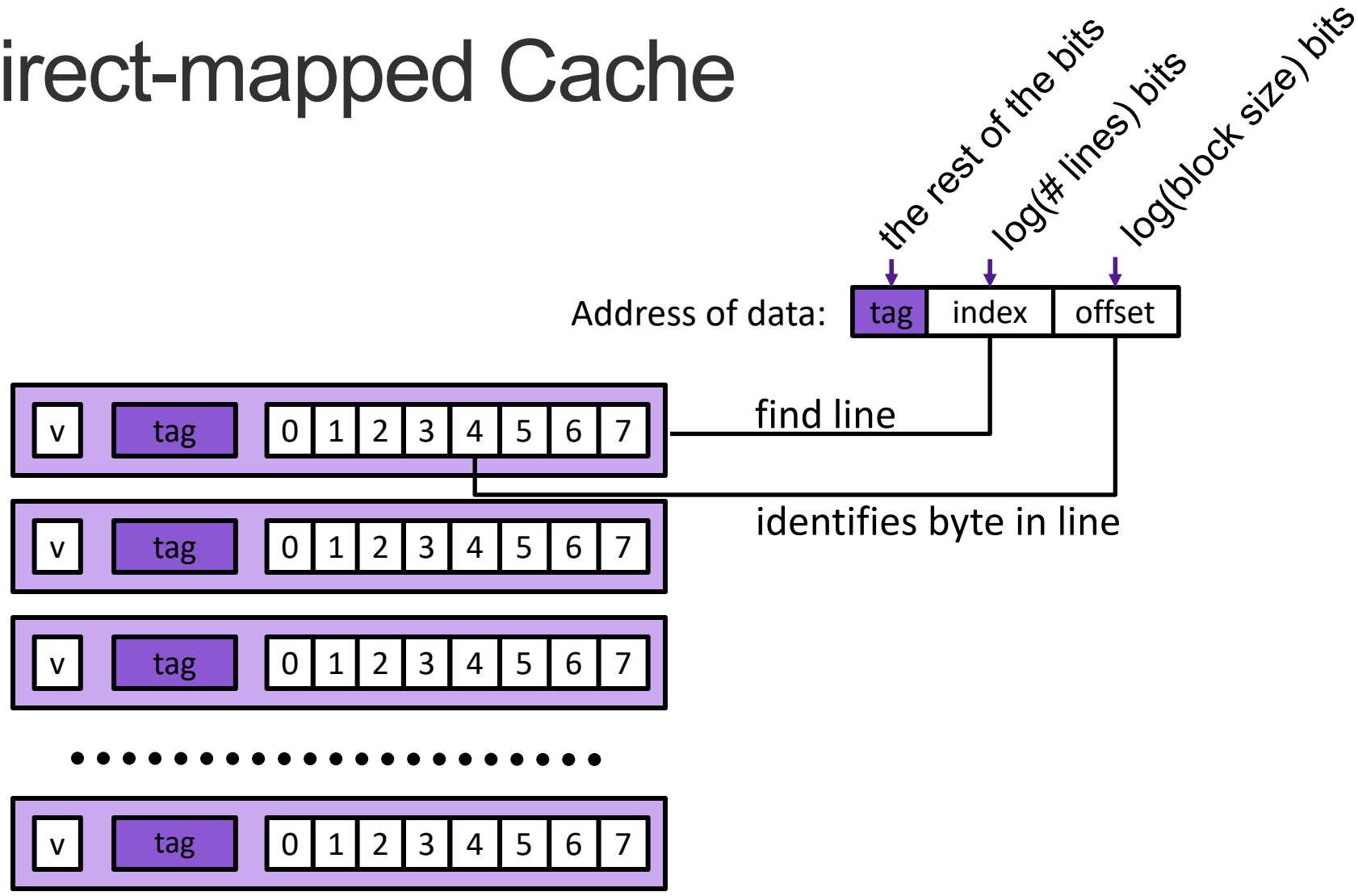


- **data block:** cached data
- **tag:** uniquely identifies which data is stored in the cache line
- **valid bit:** indicates whether or not the line contains meaningful information

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

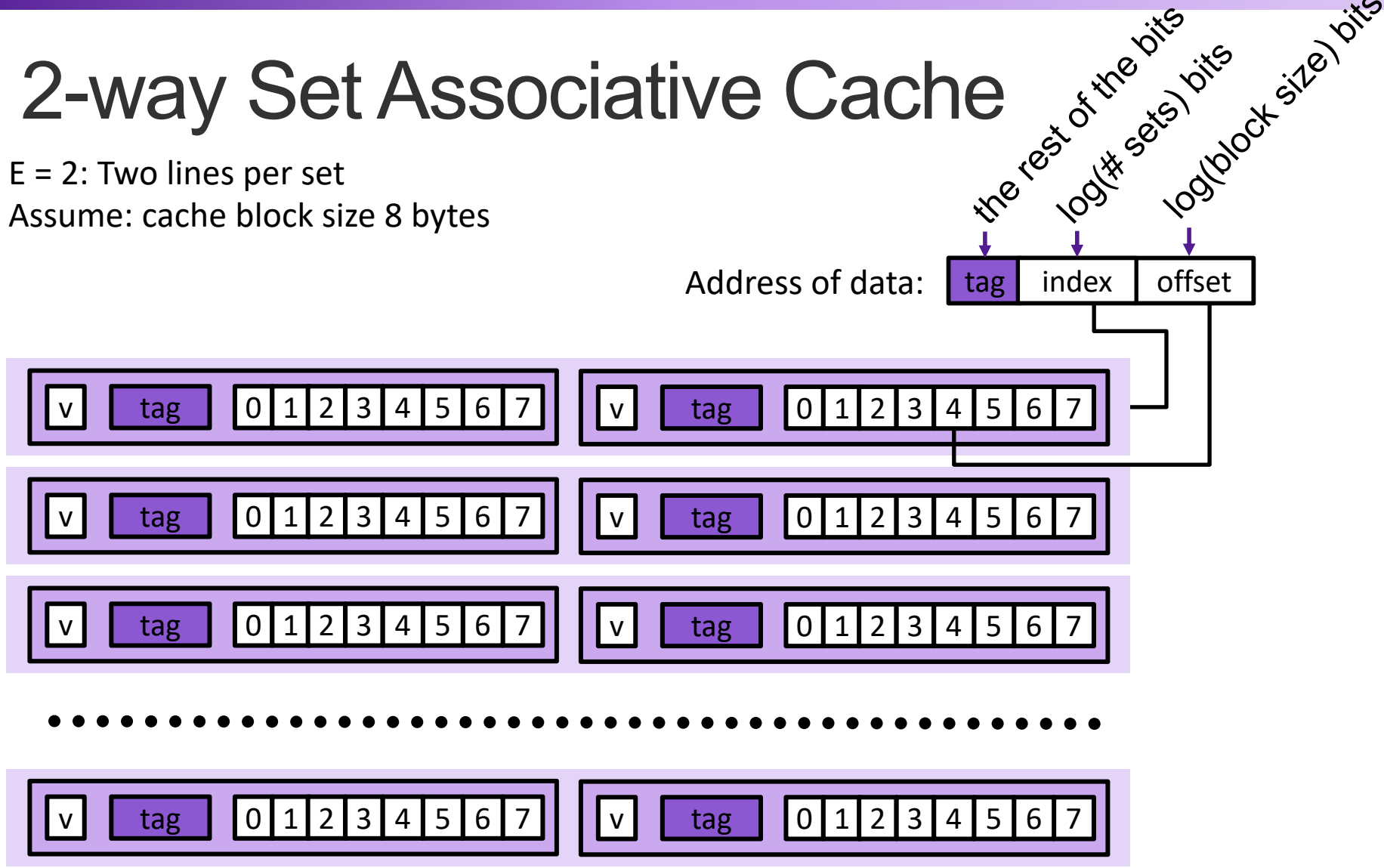


How well does this take advantage of spacial locality?
How well does this take advantage of temporal locality?

2-way Set Associative Cache

E = 2: Two lines per set

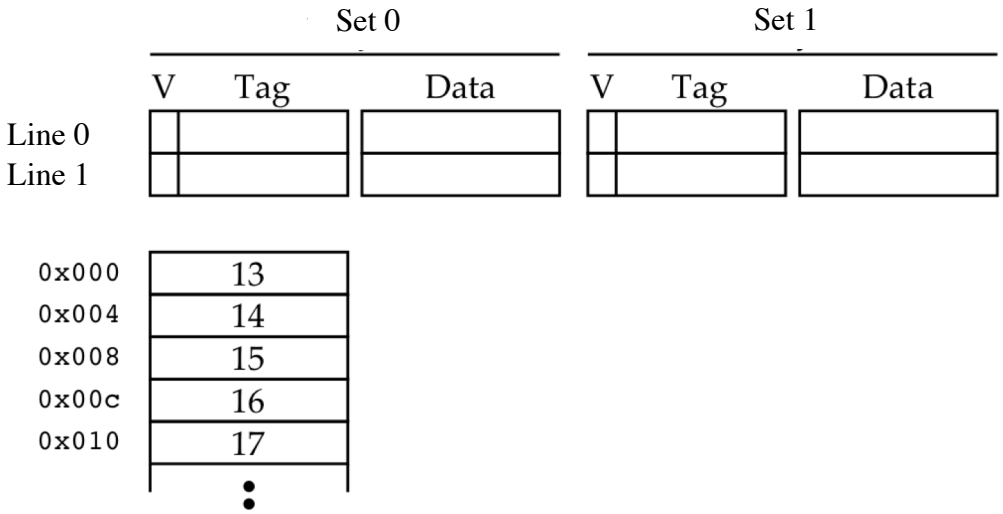
Assume: cache block size 8 bytes



Exercise: 2-way Set Associative Cache

Dynamic Transaction Stream

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



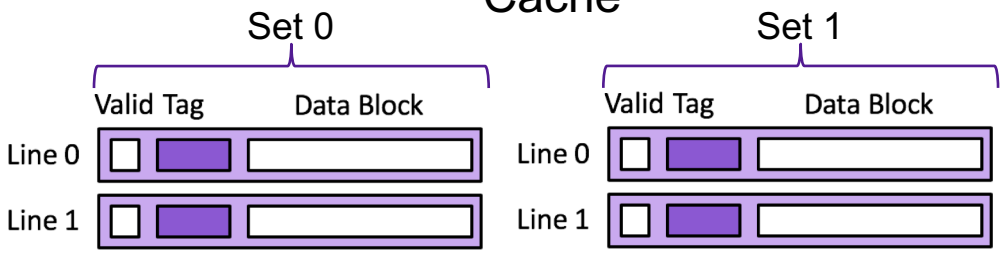
	tag	idx	h/m	Set 0		Set 1	
				Line 0	Line 1	Line 0	Line 1
`rd 0x000`							
`rd 0x004`							
`rd 0x010`							
`rd 0x000`							
`rd 0x004`							
`rd 0x020`							

Exercise: 2-way Set Associative Cache

Memory

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

Cache



Assume 8 byte data blocks

				Set 0		Set 1	
Transaction	**tag**	**index**	**h/m**	**Line 0**	**Line 1**	**Line 0**	**Line 1**
rd 0x00				?	?	?	?
rd 0x04							
rd 0x14							
rd 0x00							
rd 0x04							
rd 0x10							
rd 0x20							

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

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

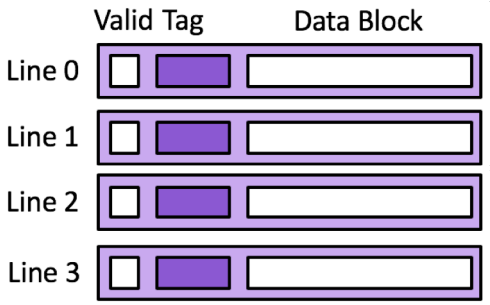
- What to do on a write-hit?
  - **Write-through:** write immediately to memory
  - **Write-back:** defer write to memory until replacement of line
    - Need a dirty bit (line different from memory or not)
- What to do on a write-miss?
  - **Write-allocate:** load into cache, update line in cache
    - 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**

Example: Write-through + No-write-allocate

Memory

`rd 0x24`	22
`rd 0x20`	21
`rd 0x1c`	20
`rd 0x18`	19
`rd 0x14`	18
`rd 0x10`	17

Cache



Assume 4 byte data blocks

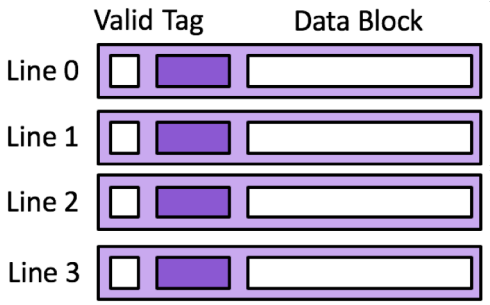
Transaction	tag	index	h/m	Line 0	Line 1	Line 2	Line 3	W
rd 0x10				?	?	?	?	?
wr 0x10								
wr 0x24								
rd 0x24								
rd 0x20								

Exercise: Write-back + Write-allocate

Memory

`rd 0x24`	22
`rd 0x20`	21
`rd 0x1c`	20
`rd 0x18`	19
`rd 0x14`	18
`rd 0x10`	17

Cache



Assume 4 byte data blocks

Transaction	tag	index	h/m	Line 0	Line 1	Line 2	Line 3	W
rd 0x10				?	?	?	?	?
wr 0x10								
wr 0x24								
rd 0x24								
rd 0x20								

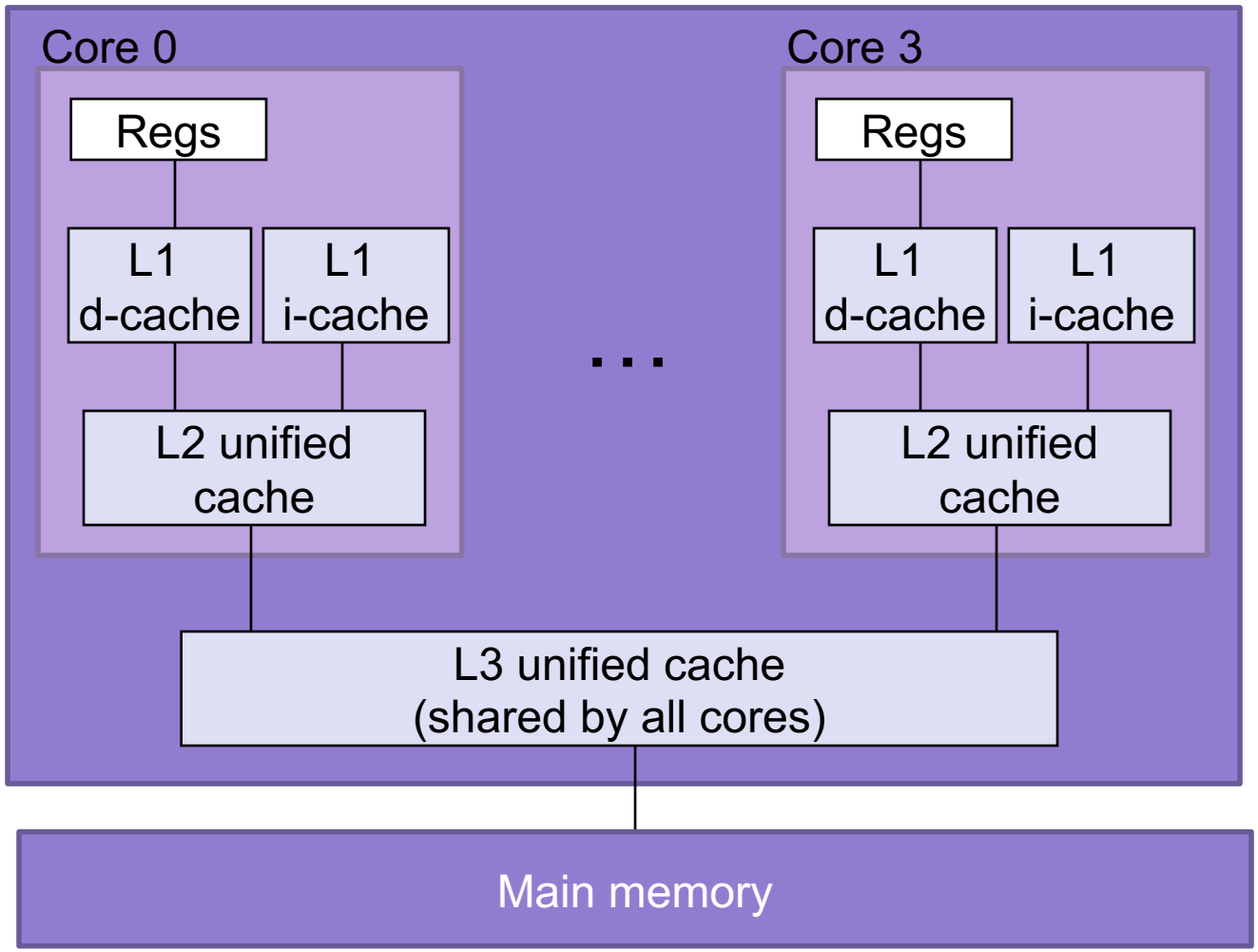
Categorizing Misses

- **Compulsory:** first-reference to a block
- **Capacity:** cache is too small to hold all of the data
- **Conflict:** collisions in a specific set

Classifying misses in a cache with a target capacity and associativity as a sequence of three questions:

1. Would this miss occur in a cache with infinite capacity? If the answer is yes, then this is a compulsory miss and we are done. If the answer is no, then consider question 2.
2. Would this miss occur in a *fully associative* cache with the desired capacity? If the answer is yes, then this is a capacity miss and we are done. If the answer is no, then consider question 3.
3. Would this miss occur in a cache with the desired capacity and associativity? If the answer is yes, then this is a conflict miss and we are done. If the answer is no, then this is not a miss – it is a hit!

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.

Example: Cache-Aware Optimization

```
for (i=0; i<n; i++) {
  for (j=0; j<n; j++) {
   sum = 0.0;
   for (k=0; k<n; k++)
     sum += a[i][k] * b[k][j];
   c[i][j] = sum;
 }
}
```

ijk (& jik):

- 2 loads, 0 stores
- misses/iter = 1.25

```
for (k=0; k<n; k++) {
 for (i=0; i<n; i++) {
  r = a[i][k];
  for (j=0; j<n; j++)
   c[i][j] += r * b[k][j];
 }
}
```

kij (& ikj):

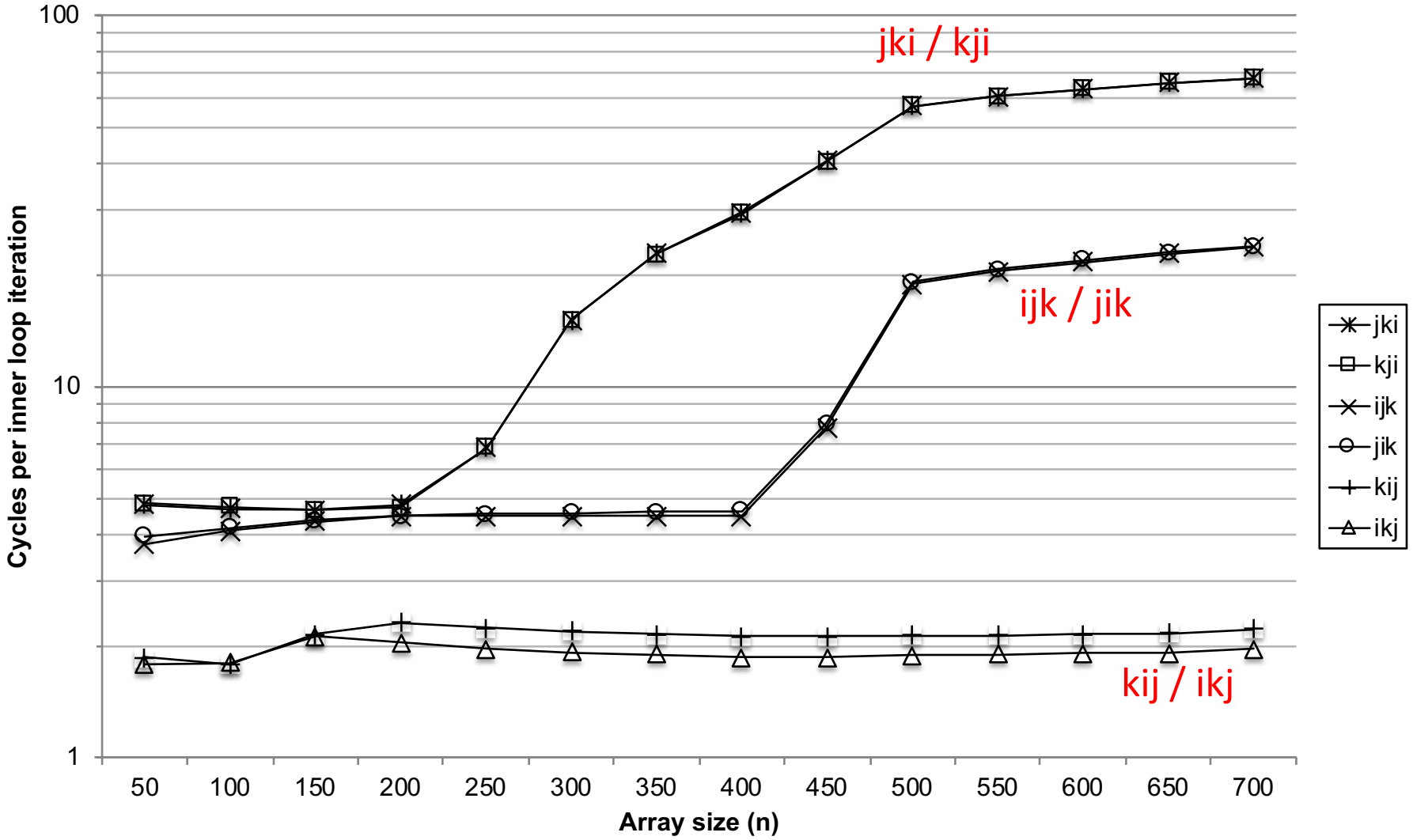
- 2 loads, 1 store
- misses/iter = 0.5

```
for (j=0; j<n; j++) {
 for (k=0; k<n; k++) {
   r = b[k][j];
   for (i=0; i<n; i++)
    c[i][j] += a[i][k] * r;
 }
}
```

jki (& kji):

- 2 loads, 1 store
- misses/iter = 2.0

Core i7 Matrix Multiply Performance



Pentium III Xeon Matrix Multiply Performance