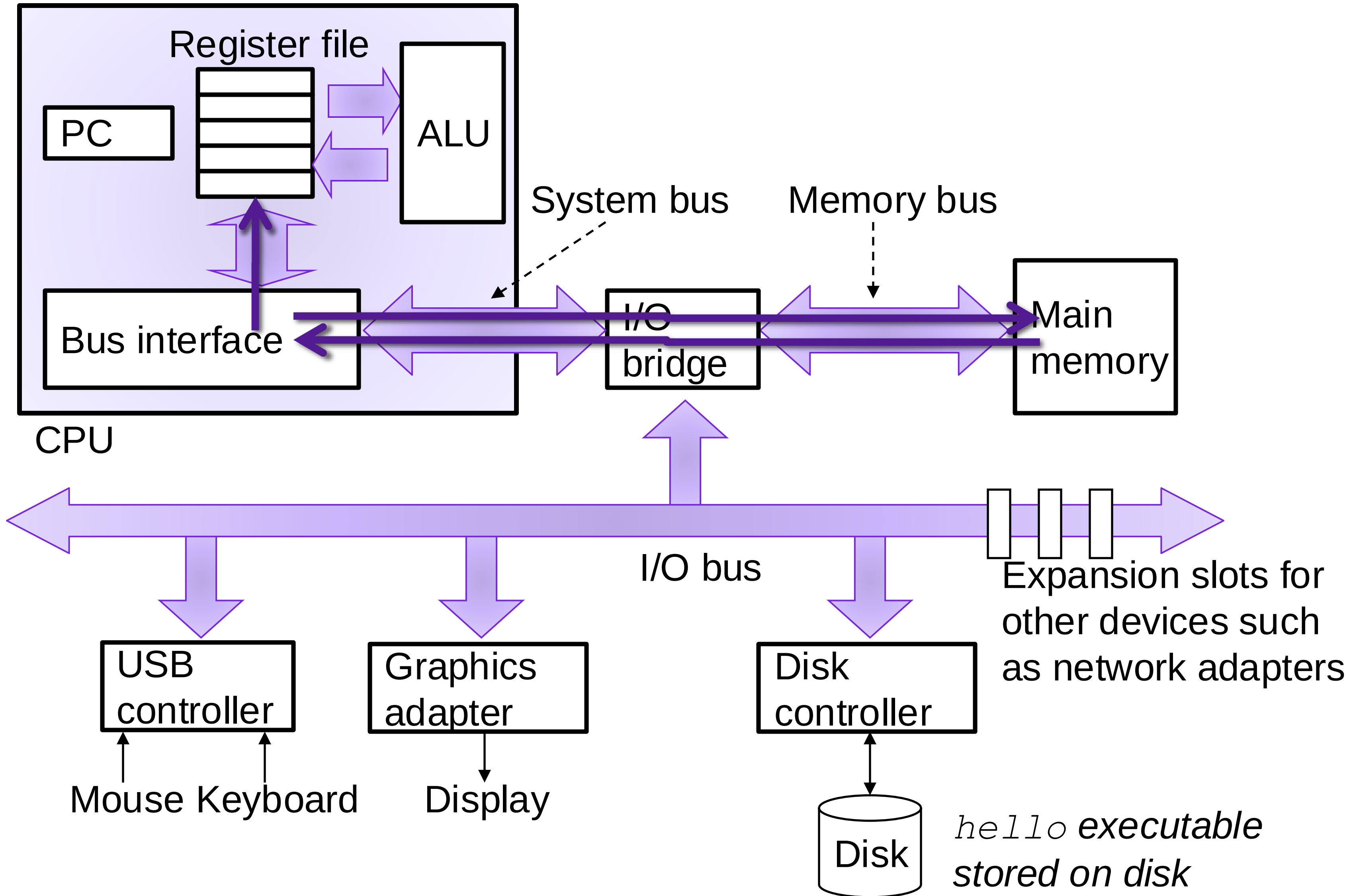
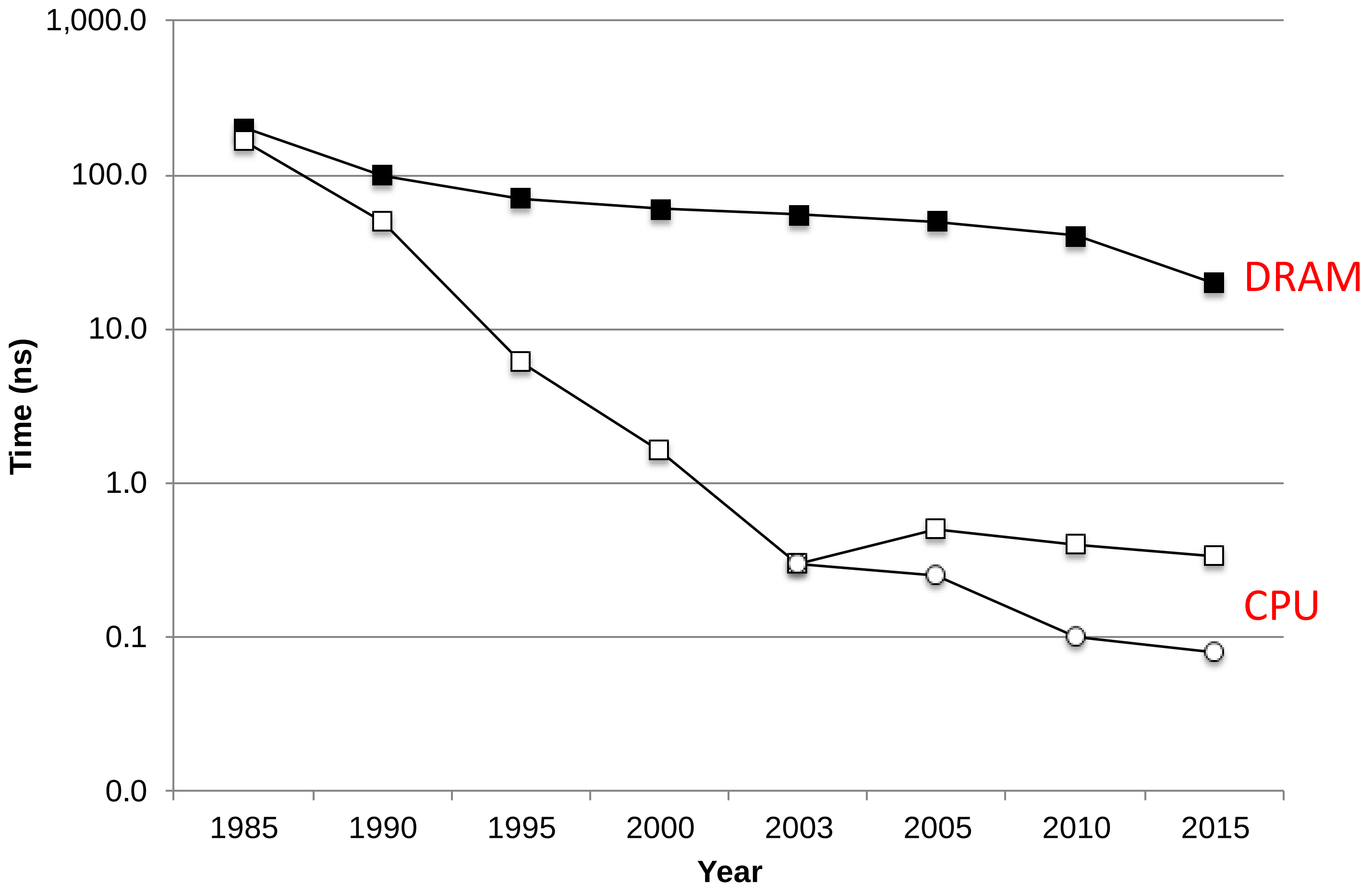
Lecture 10: Caches

CS 105 Fall 2024

A Computer System

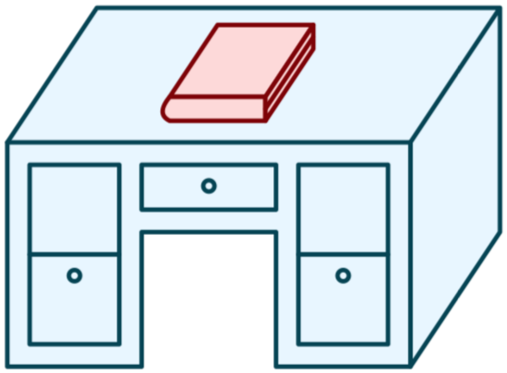


The CPU-Memory Gap

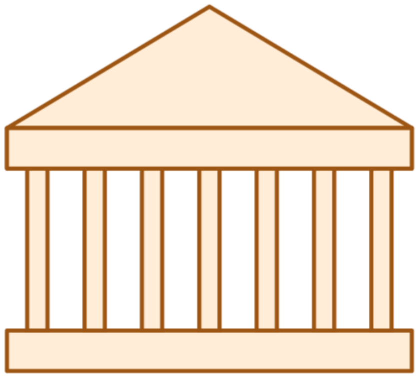


Life without caches

- You decide that you want to learn more about computer systems than is covered in this course
- The library contains all the books you could possibly want, but you don't like to study in libraries, you prefer to study in your dorm room.
- You have the following constraints:

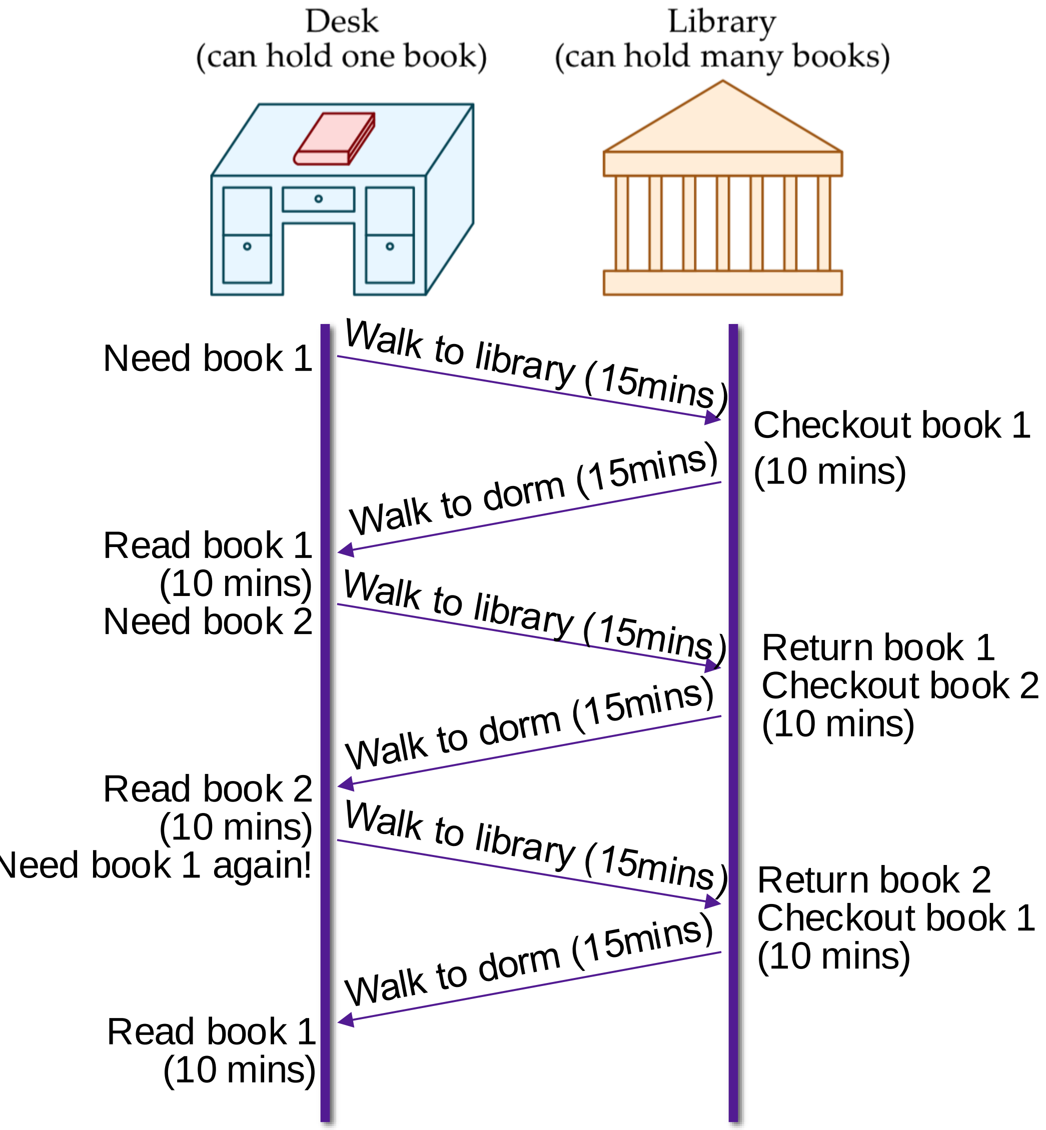


Desk
(can hold one book)



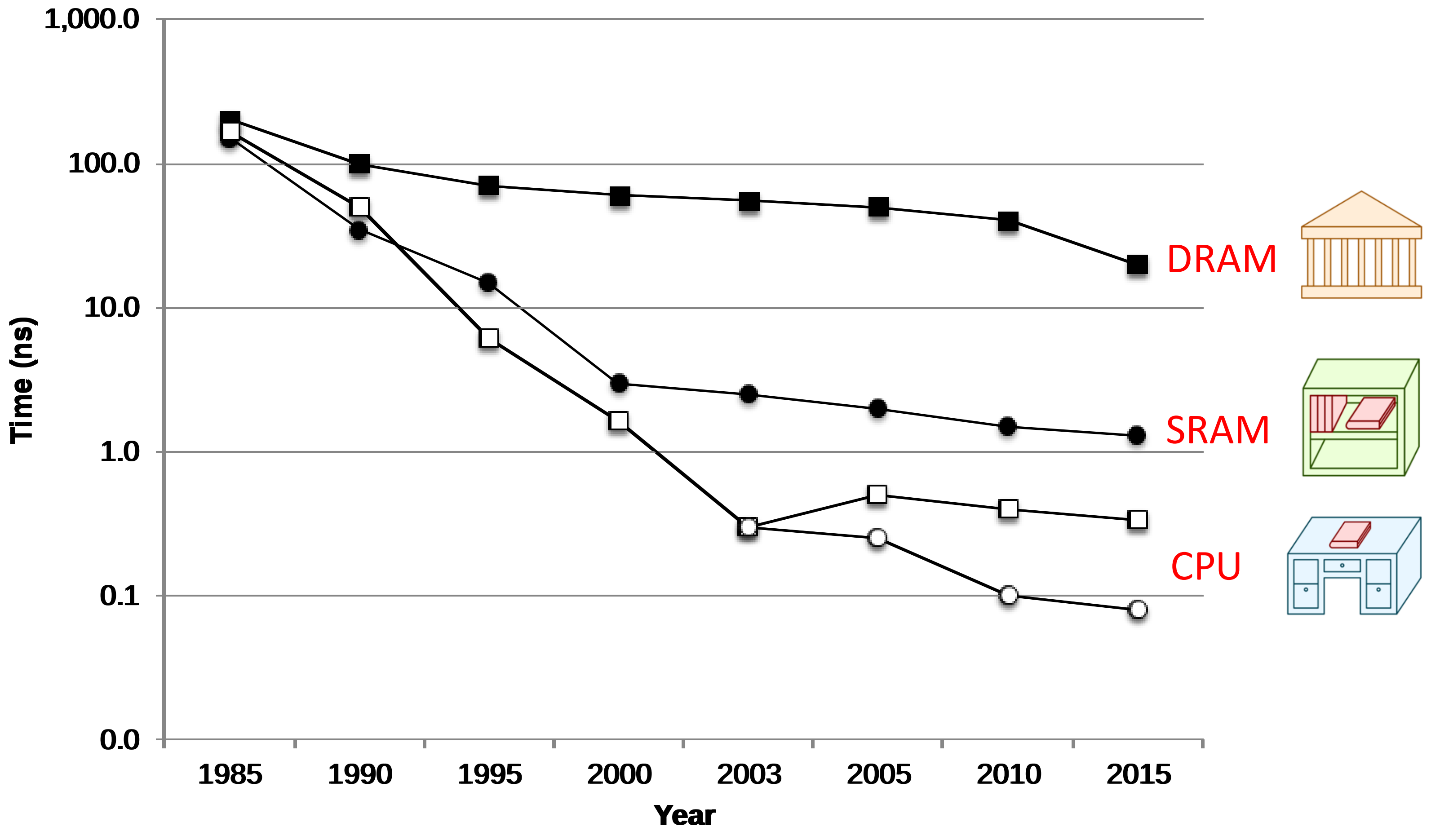
Library
(can hold many books)

Quantifying Speed (without caches)

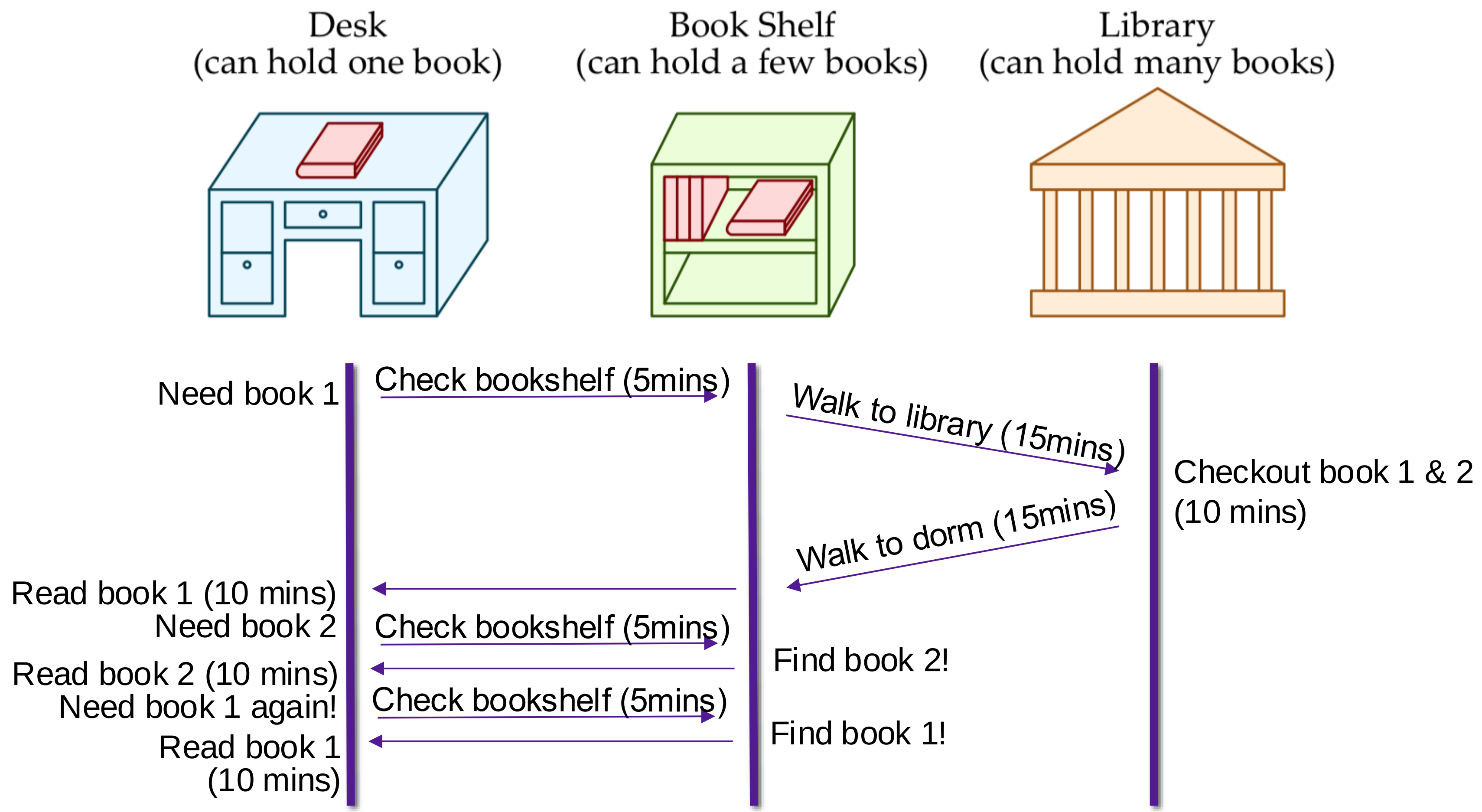


- Average latency to access a book: 40mins
- Average throughput (incl. reading time): 1.2 books/hr

The CPU-Memory Gap



Life with caching



- Average latency to access a book: <20mins
- Average throughput (incl. reading time): ~2 books/hr

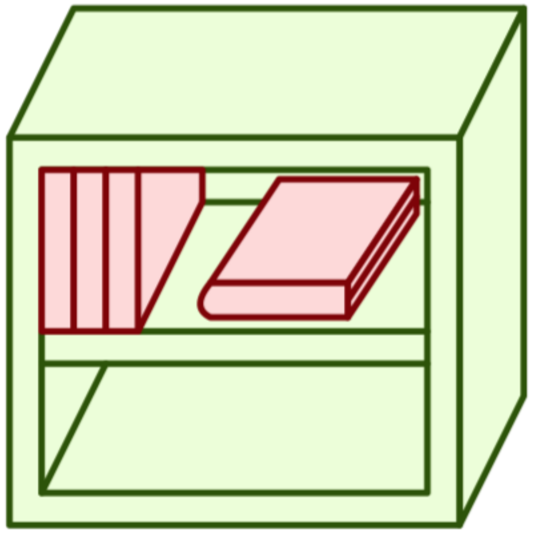
Caching—The Vocabulary

- **Size:** the total number of bytes that can be stored in the cache

- **Cache Hit:** the desired value is in the cache and returned quickly
- **Cache Miss:** the desired value is not in the cache and must be fetched from a more distant cache (or ultimately from main memory)

Exercise 1: Caching Strategies

How should we decide which books to keep in the bookshelf?



Example Access Patterns

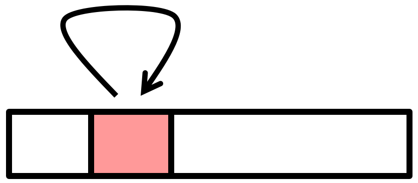
```
int sum = 0;
for (int i = 0; i < n; i++){
    sum += a[i];
}
return sum;
```

- Data references
  - Reference array elements in succession.
  - Reference variable **`sum`** each iteration.
- Instruction references
  - Reference instructions in sequence.
  - Cycle through loop repeatedly.

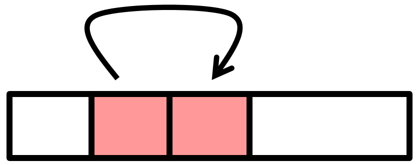
Principle of Locality

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

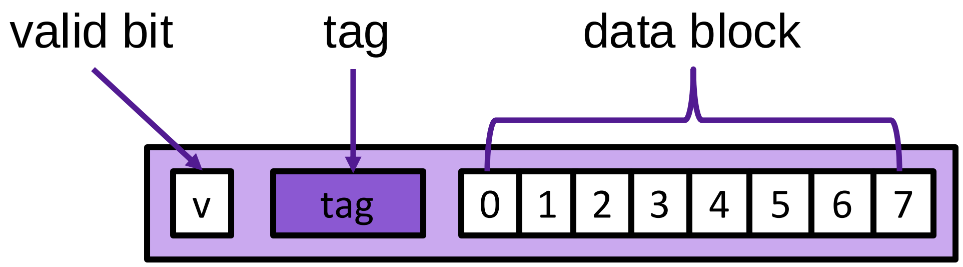


- Spatial locality:
  - Items with nearby addresses tend to be referenced close together in time



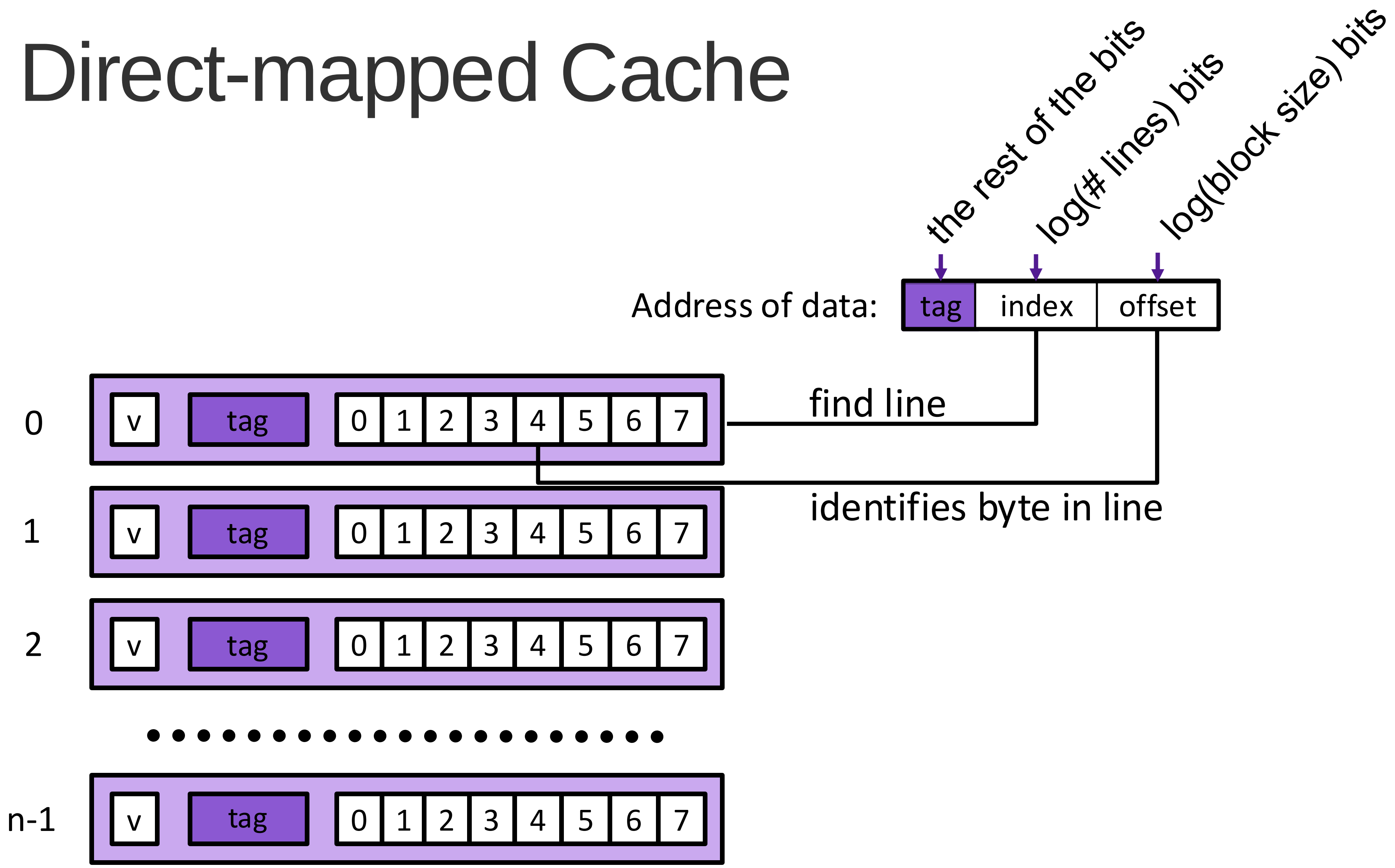
CACHE ORGANIZATION

Cache Lines



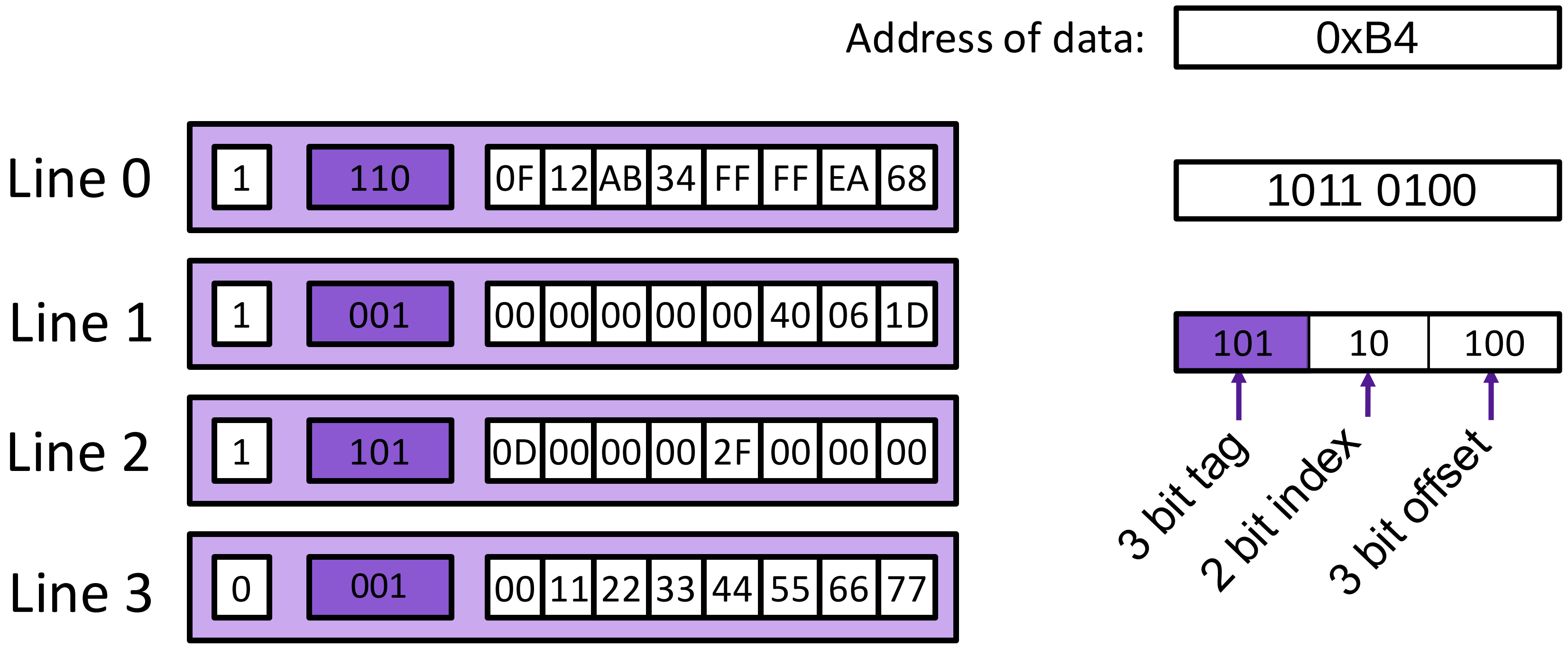
- **data block:** cached data (i.e., copy of bytes from memory)
- **tag:** uniquely identifies which data is stored in the cache line
- **valid bit:** indicates whether or not the line contains meaningful information

Direct-mapped Cache



Example: Direct-mapped Cache

Assume: cache block size 8 bytes, total cache size 32 bytes
Assume: assume 8-bit machine



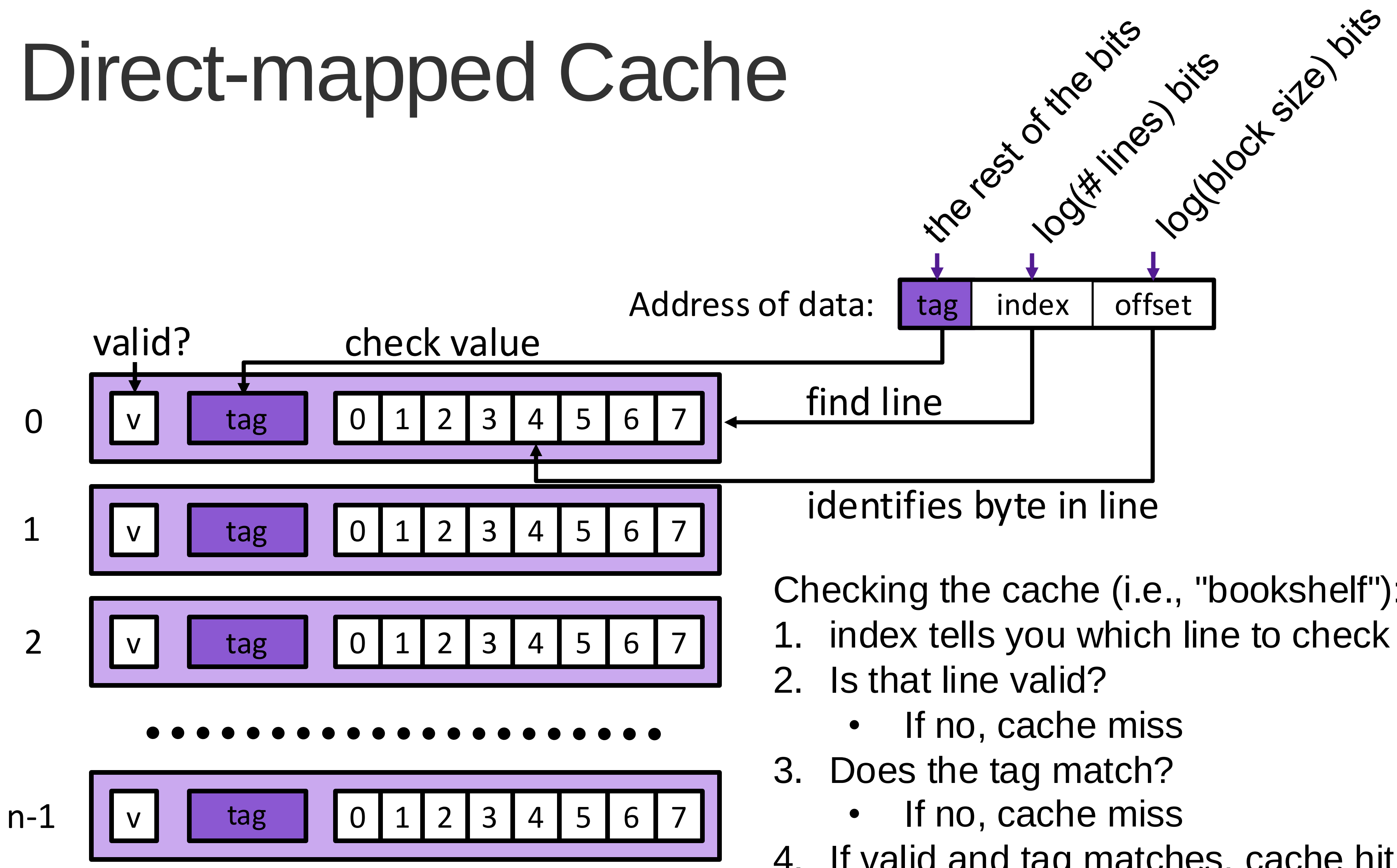
	Valid	Tag	Data							
Line 0	1	110	0F	12	AB	34	FF	FF	EA	68
Line 1	1	001	00	00	00	00	00	40	06	1D
Line 2	1	101	0D	00	00	00	2F	00	00	00
Line 3	0	001	00	11	22	33	44	55	66	77

Exercise 2: Interpreting Addresses

Consider the 12-bit address 0xA59. What would be the tag, index, and offset for this address with each of the following cache configurations?

1. A direct-mapped cache with 8 cache lines and 8-byte data blocks
2. A direct-mapped cache with 16 cache lines and 4-byte data blocks
3. A direct-mapped cache with 16 cache lines and 8-byte data blocks

Direct-mapped Cache

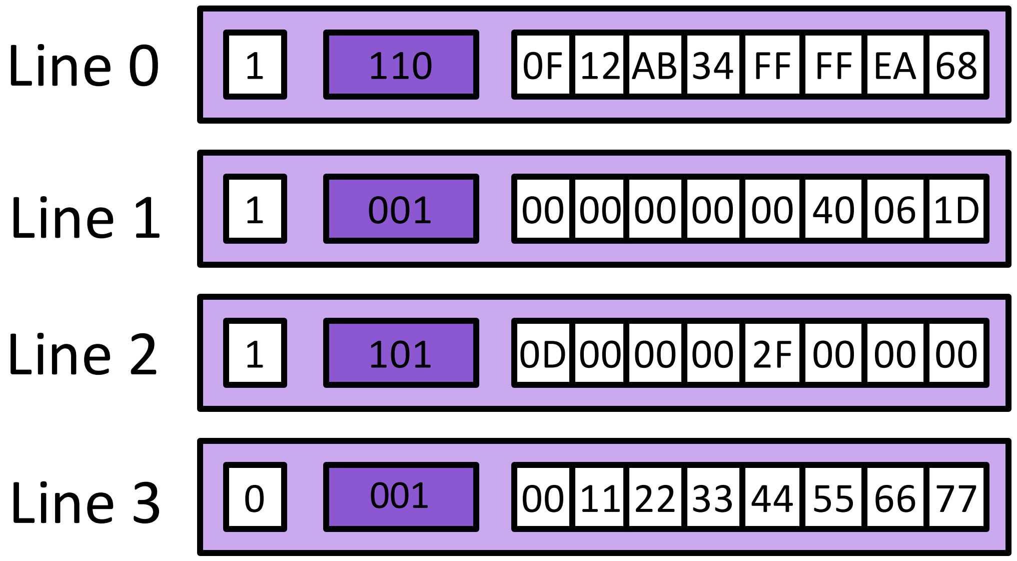


Checking the cache (i.e., "bookshelf"):

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

Exercise 3: Cache Hits and Misses

Assume: cache block size 8 bytes, total cache size 32 bytes
Assume: assume 8-bit machine


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 1 101 0D 00 00 00 2F 00 00 00
Line 3 0 001 00 11 22 33 44 55 66 77

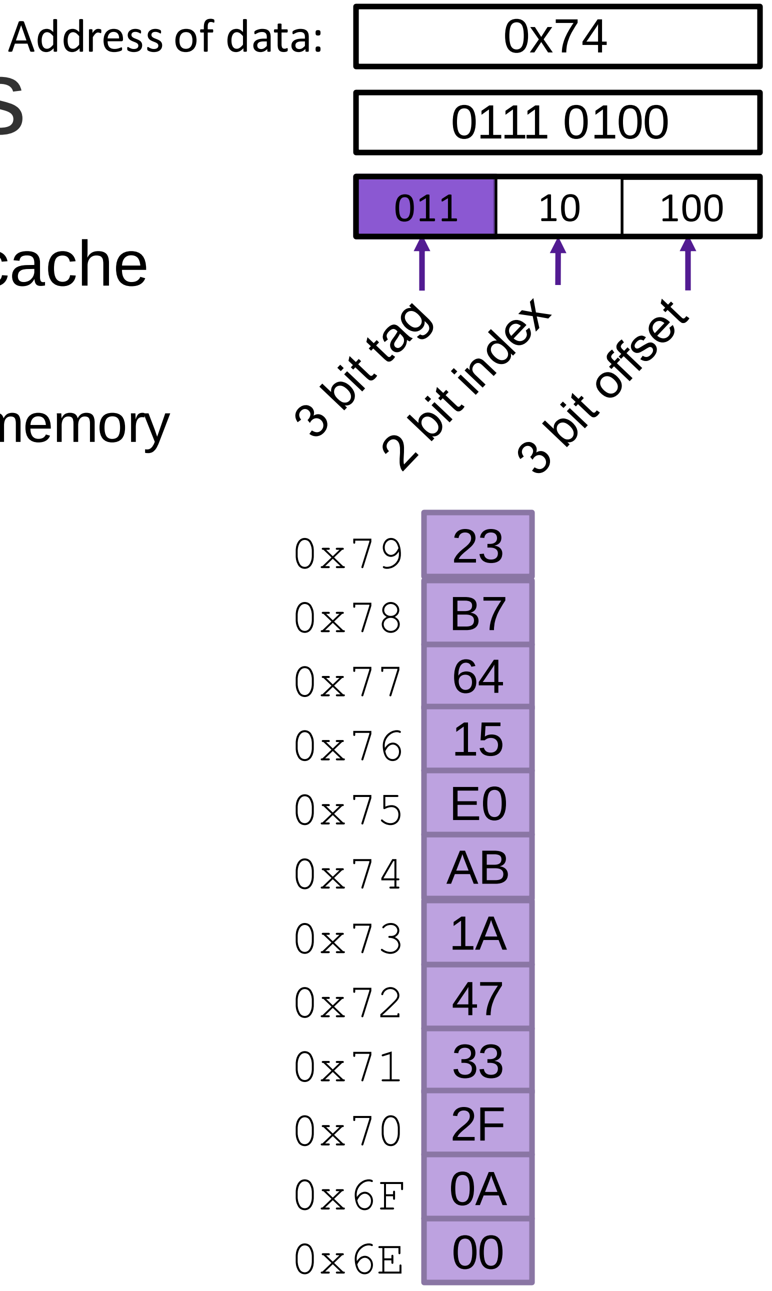
For each address, is it a hit or a miss? For hits, what data is at that address in memory?

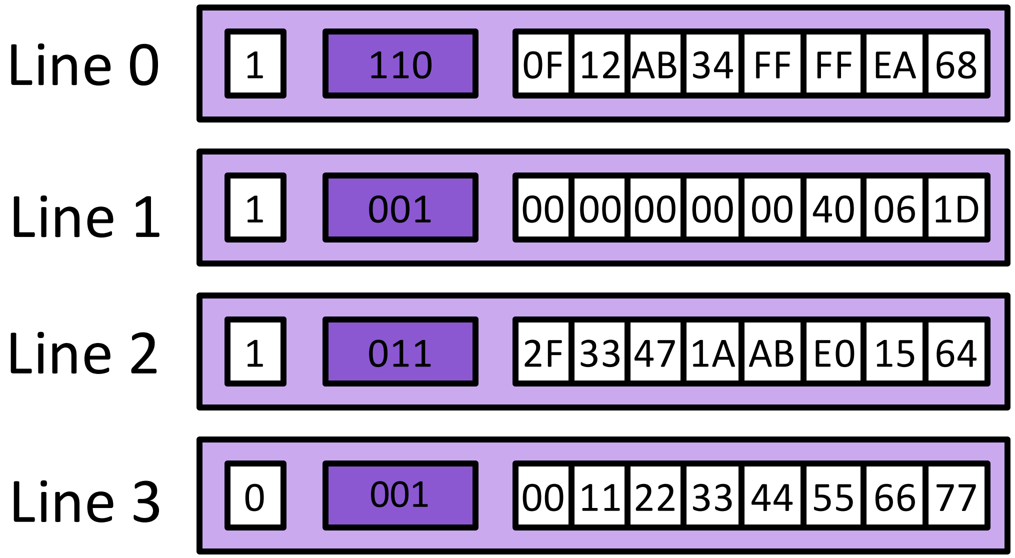
- 0x2D
- 0x2E
- 0x74
- 0x3A

Handling a 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)





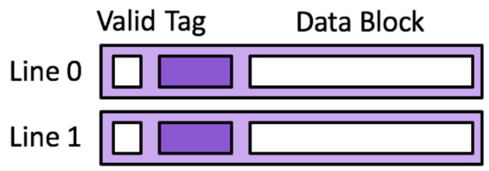
	Valid	Tag	Data							
Line 0	1	110	0F	12	AB	34	FF	FF	EA	68
Line 1	1	001	00	00	00	00	00	40	06	1D
Line 2	1	011	2F	33	47	1A	AB	E0	15	64
Line 3	0	001	00	11	22	33	44	55	66	77

Exercise 4: Direct-mapped Cache

Memory

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

Cache



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

Assume 8 byte data blocks

Time

Line 0			Line 1		
0	0000	47 48	0	0000	49 50

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