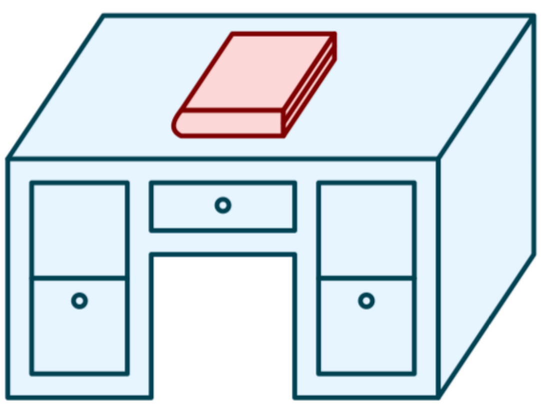
Lecture 12: Caches

CS 105

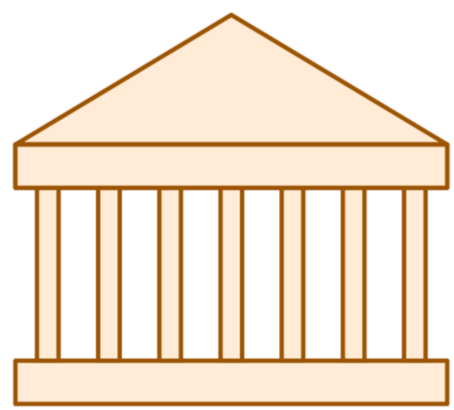
March 2, 2020

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 at home.
- You have the following constraints:

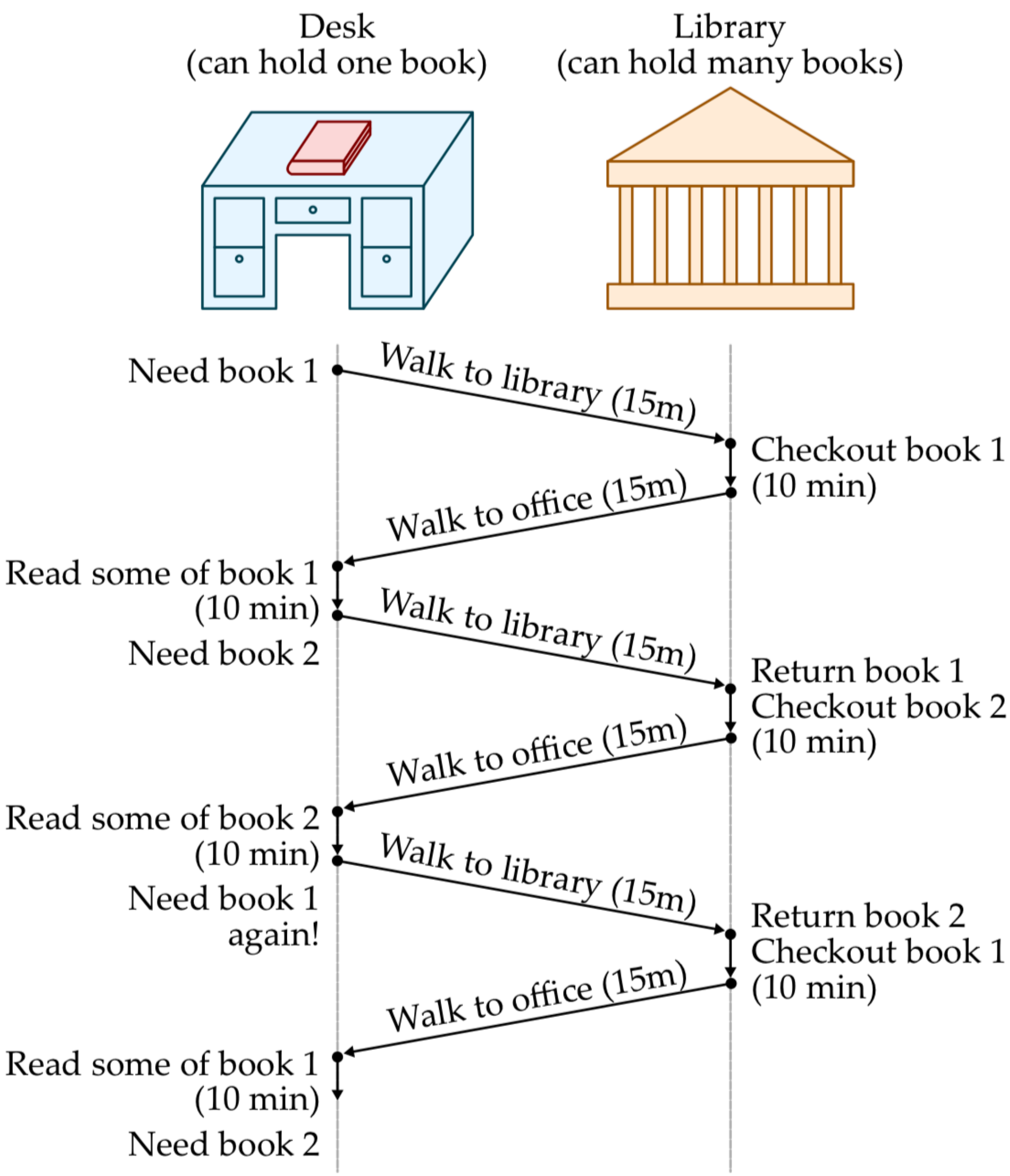


Desk
(can hold one book)



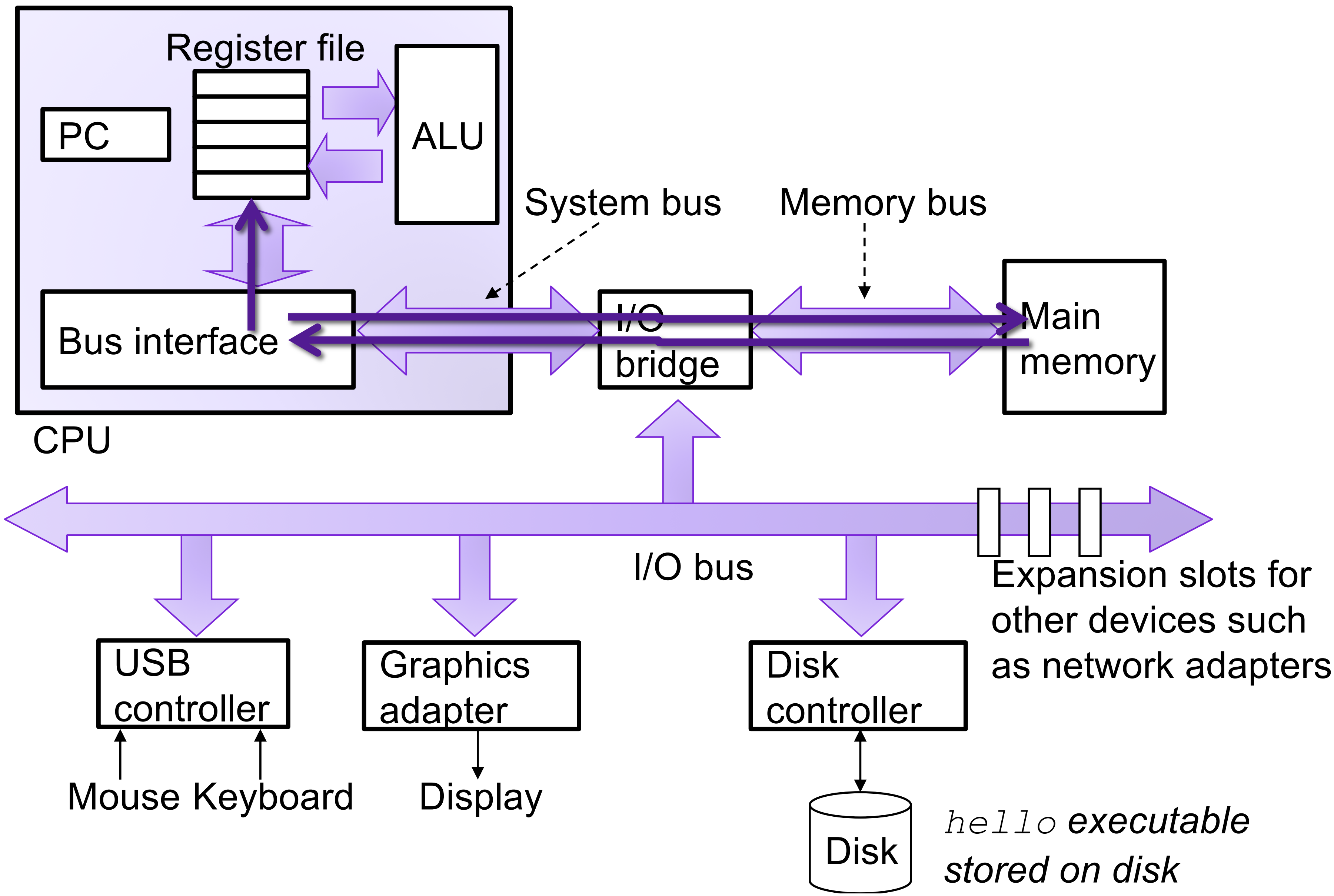
Library
(can hold many books)

Life without caches

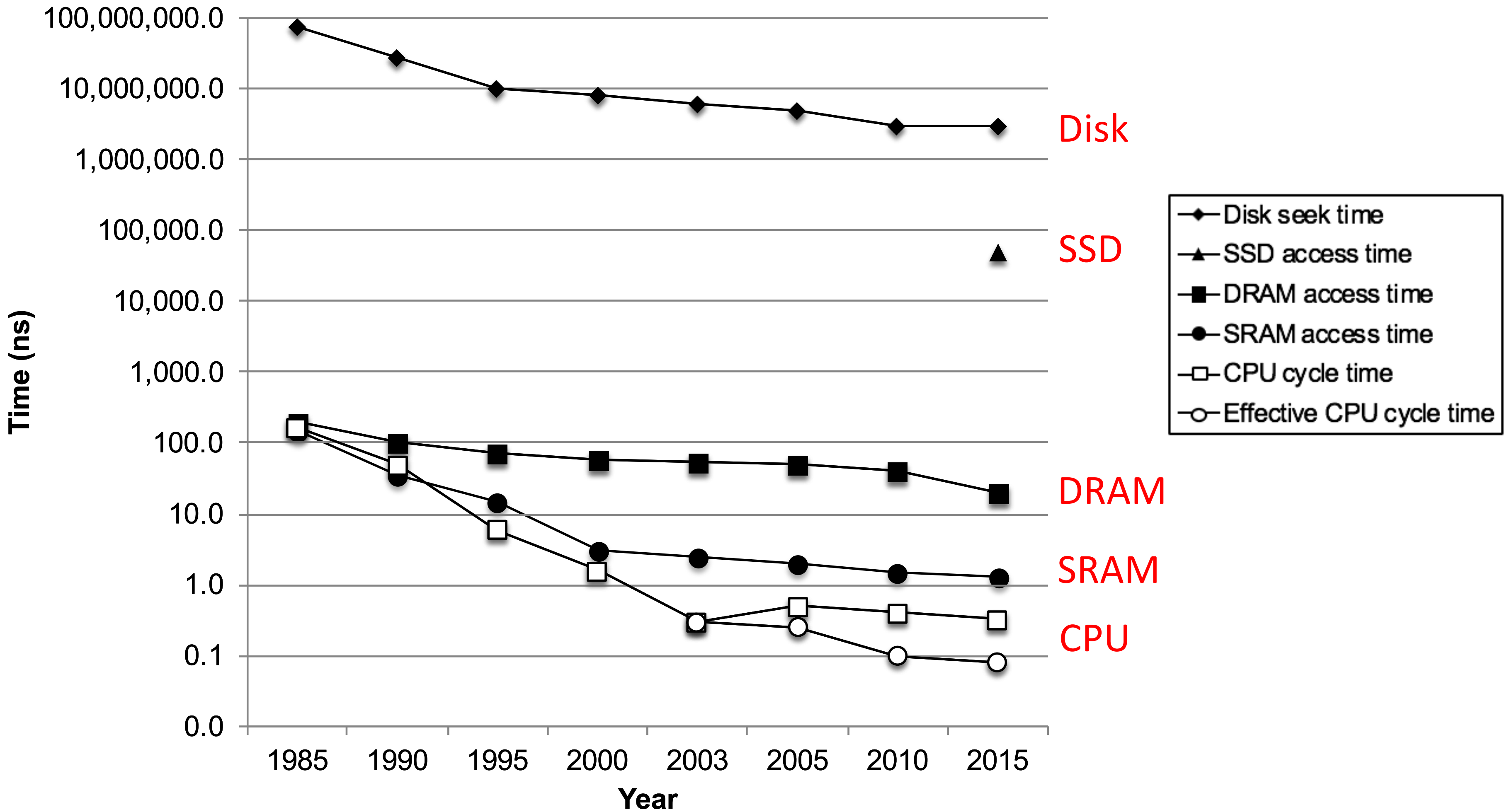


- Average latency to access a book: 40mins
- Average throughput (incl. reading time): 1.2 books/hr

A Computer System



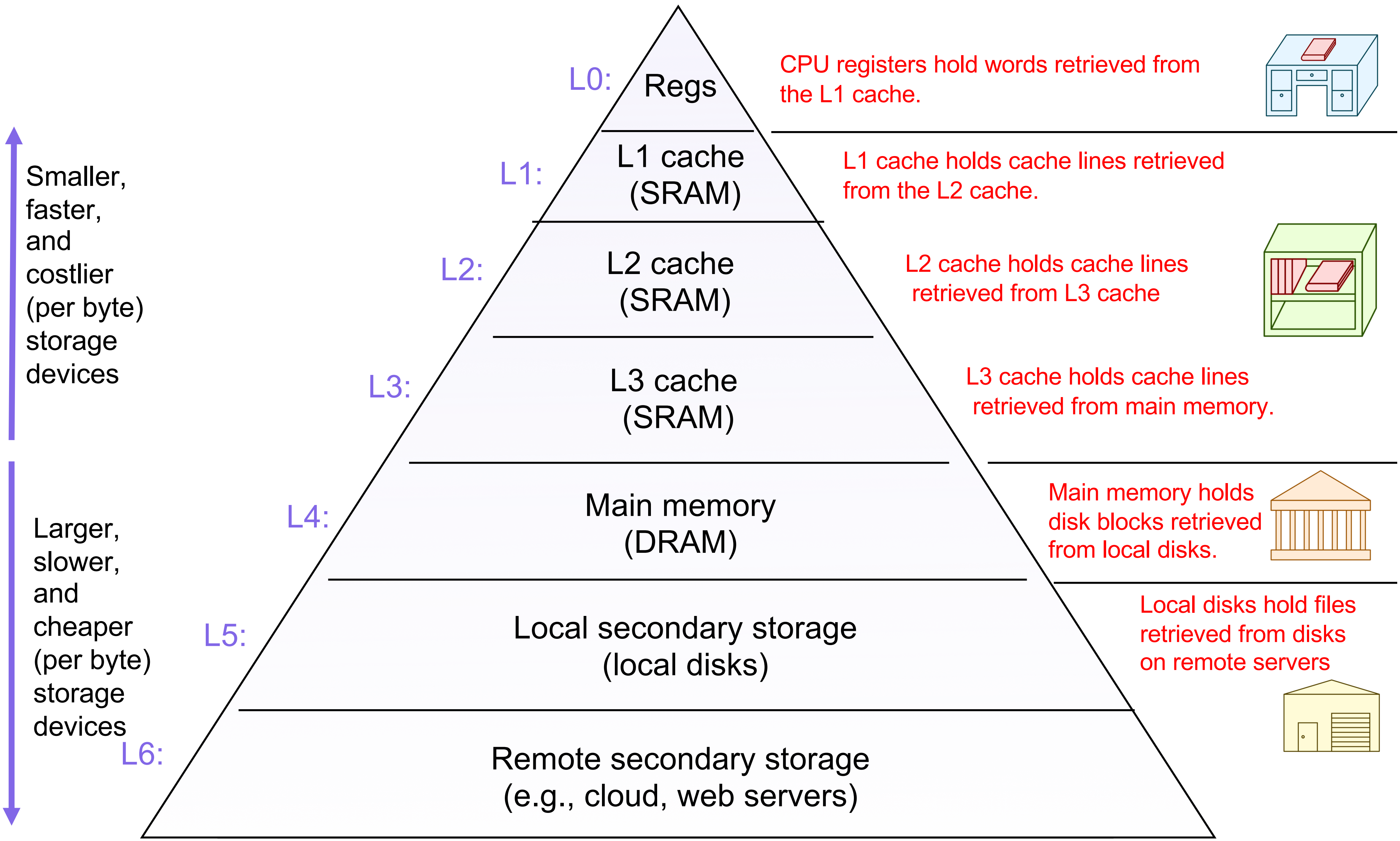
The CPU-Memory Gap



Caching—The Very Idea

- Keep some memory values nearby in fast memory
- Modern systems have 3 or even 4 levels of caches
- Cache idea is widely used:
  - Disk controllers
  - Web
  - (Virtual memory: main memory is a “cache” for the disk)

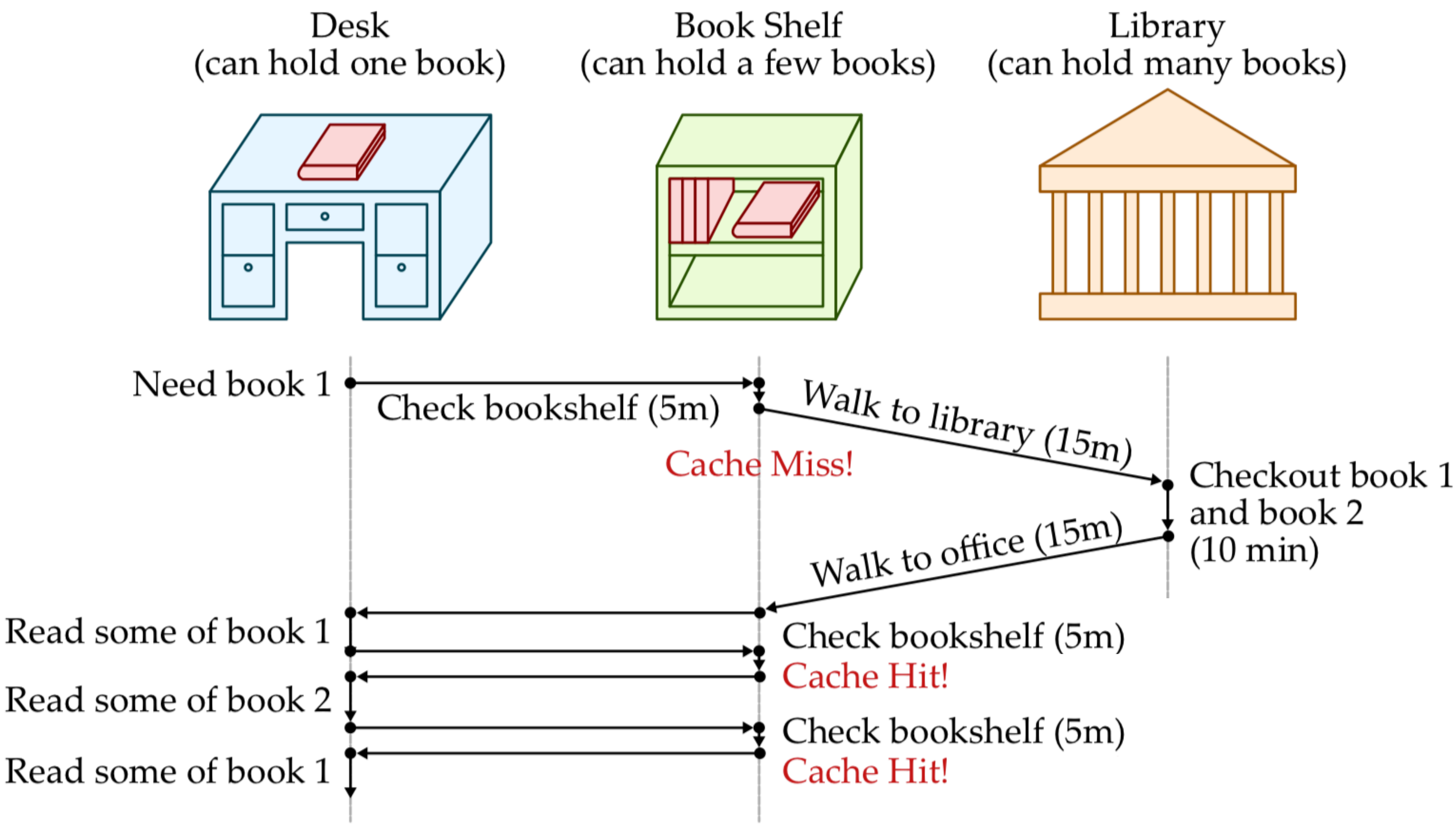
Memory Hierarchy



Latency numbers every programmer should know (2020)

L1 cache reference	1 ns	
Branch mispredict	3 ns	
L2 cache reference	4 ns	
Main memory reference	100 ns	
memory 1MB sequential read	3,000 ns	3 μs
SSD random read	16,000 ns	16 μs
SSD 1MB sequential read	49,000 ns	49 μs
Magnetic Disk seek	2,000,000 ns	2 ms
Magnetic Disk 1MB sequential read	825,000 ns	825 μs
Round trip in Datacenter	500,000 ns	500 μs
Round trip CA<->Europe	150,000,000 ns	150 ms

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

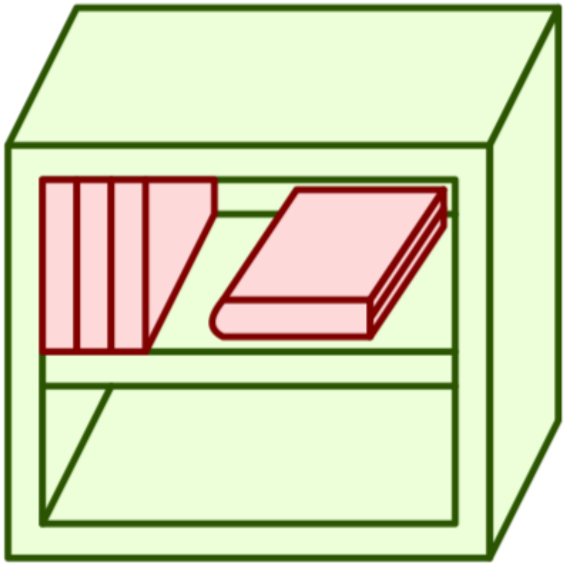
- **Cache Hit:** the desired value is in the cache and returned quickly
- **Cache Miss:** the desired value is not in the cache and must be fetched from a more distant cache (or ultimately from main memory)

- **Miss rate:** the fraction of accesses that are misses

- **Hit time:** the time to process a hit
- **Miss penalty:** the *additional* time to process a miss

- **Average access time:** hit-time + miss-rate * miss-penalty

Question: how do we decide which books to put on the bookshelf?

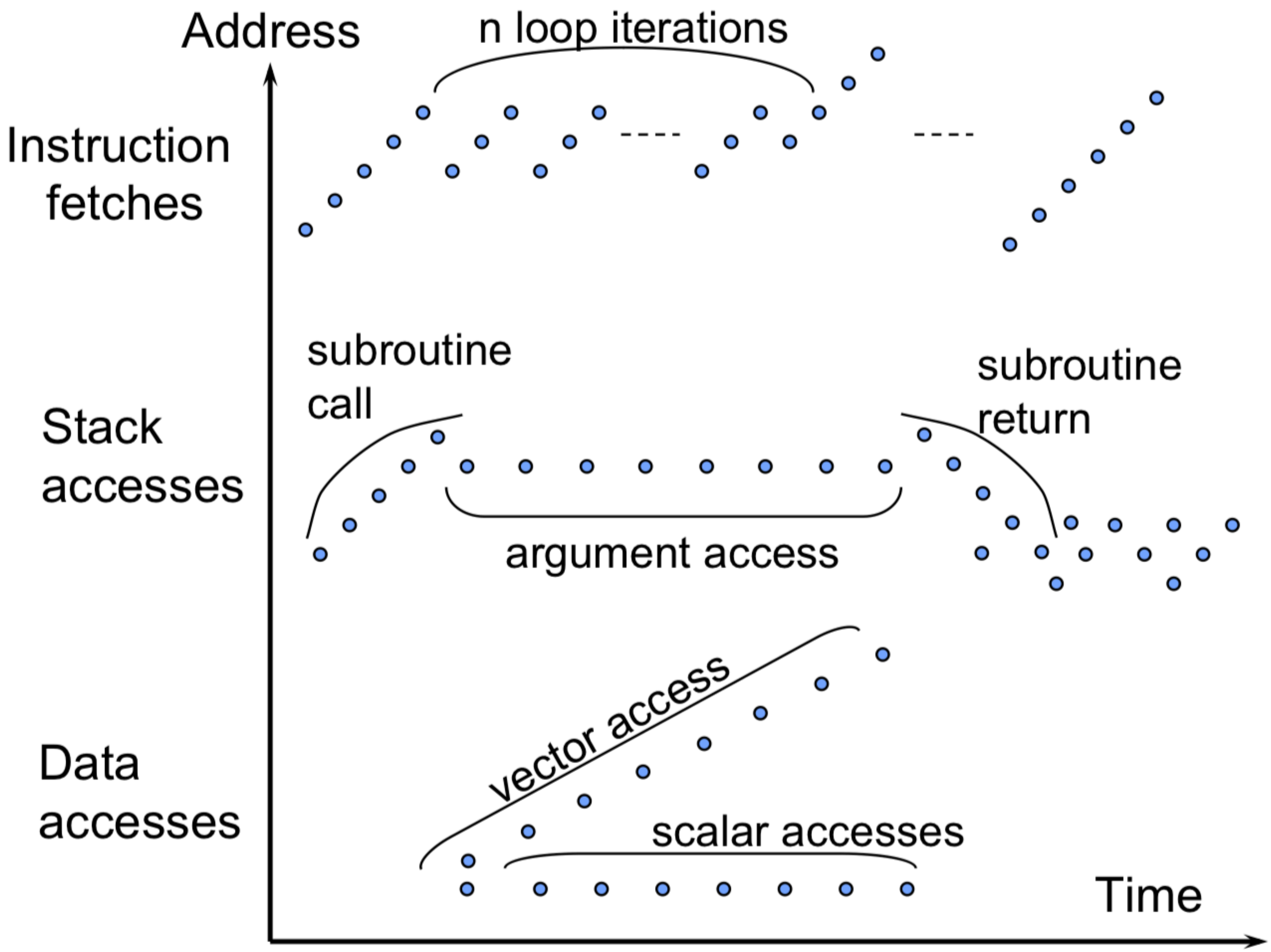


Example Access Patterns

```
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.

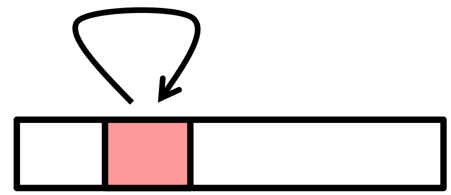
Example Access Patterns



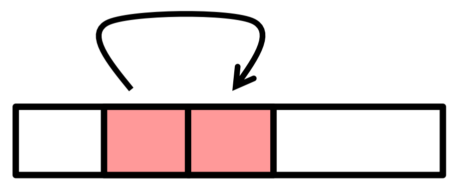
Principle of Locality

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

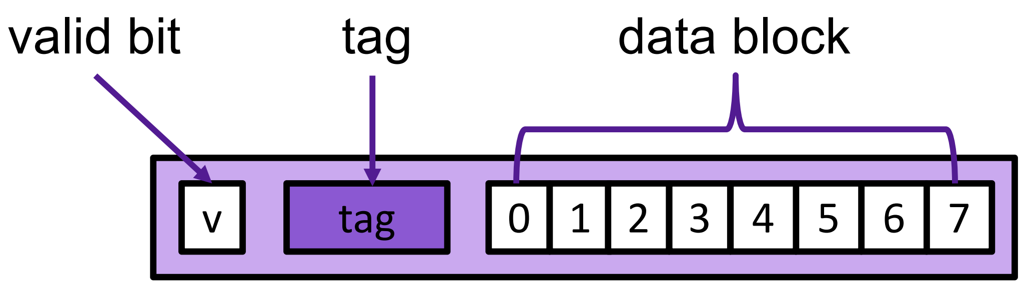


- Spatial locality:
  - Items with nearby addresses tend to be referenced close together in time



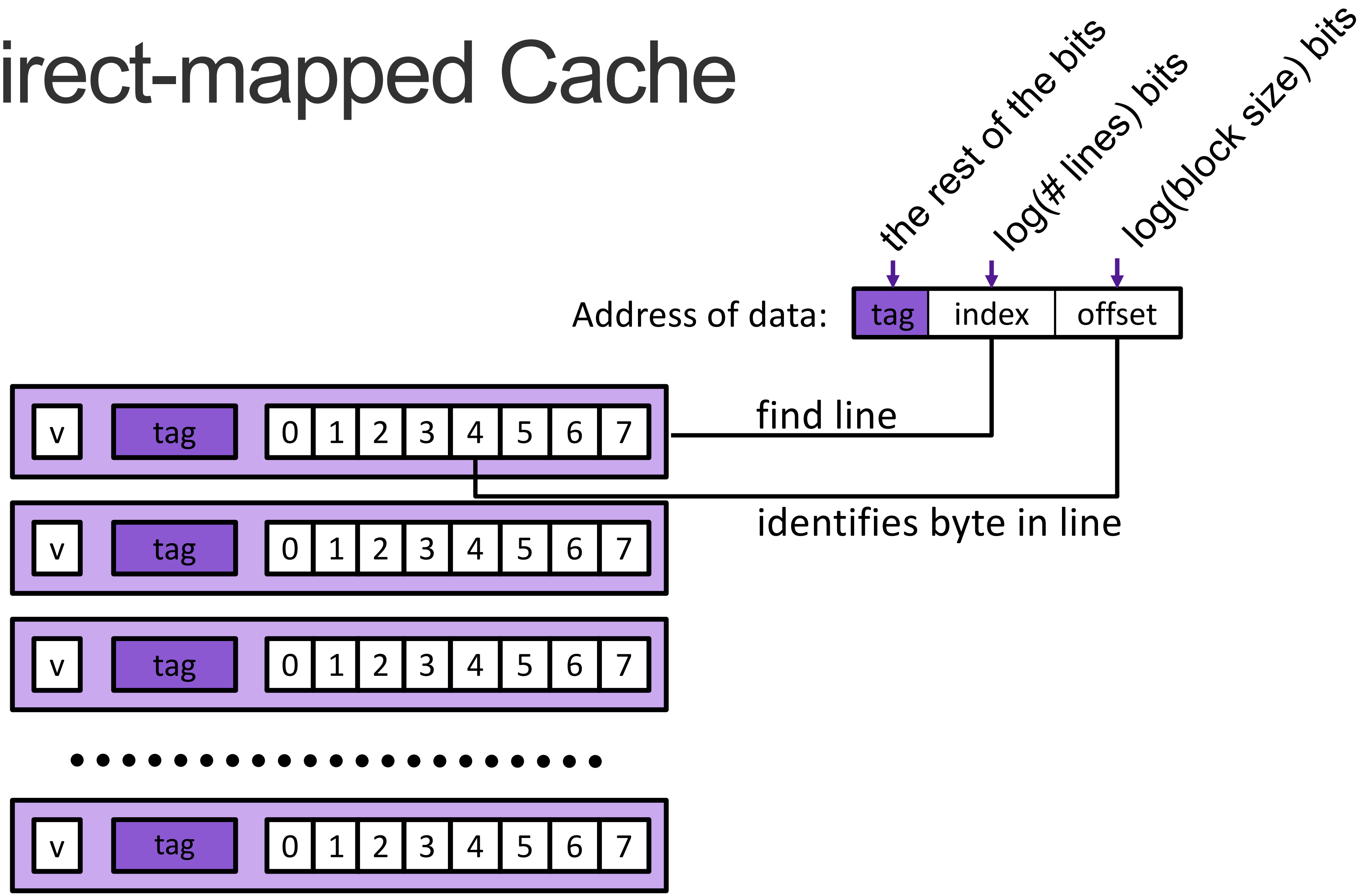
CACHE ORGANIZATION

Cache Lines



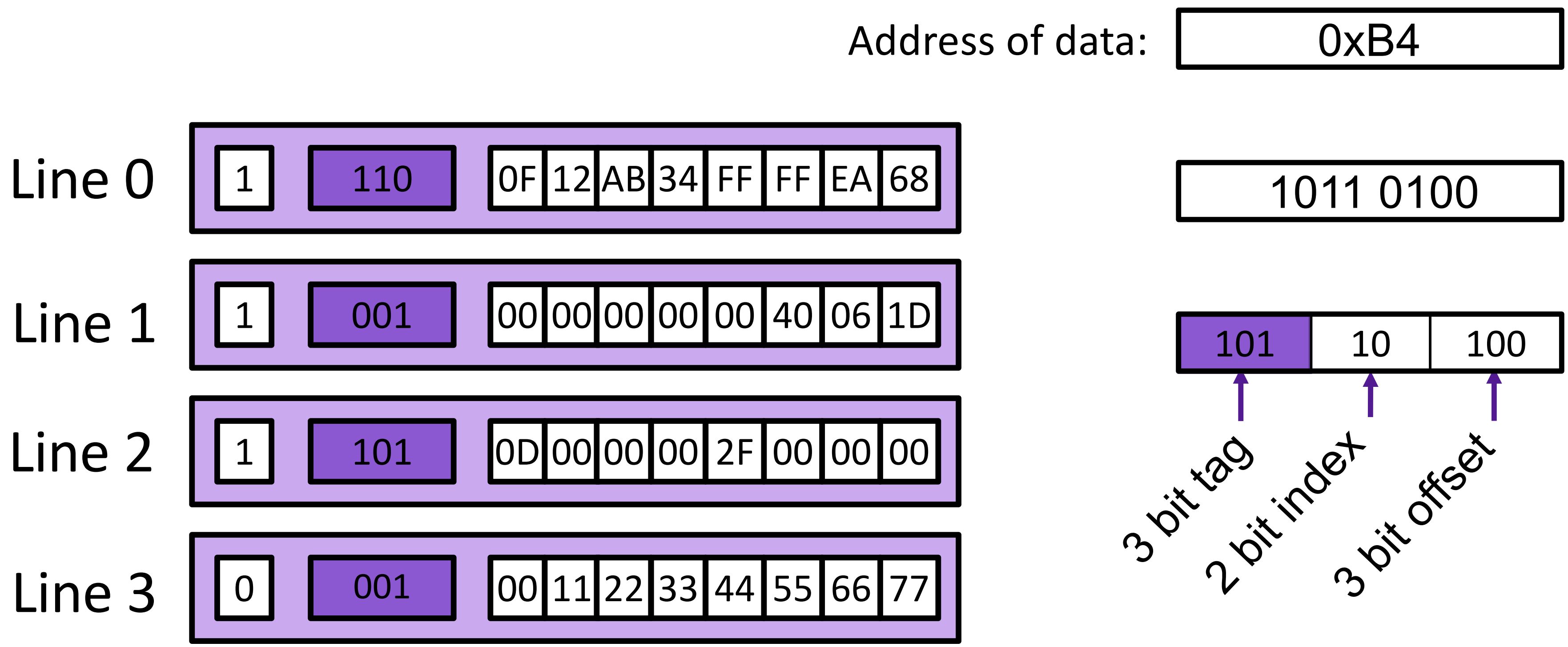
- **data block:** cached data
- **tag:** uniquely identifies which data is stored in the cache line
- **valid bit:** indicates whether or not the line contains meaningful information

Direct-mapped Cache



Example: Direct-mapped Cache

Assume: cache block size 8 bytes
Assume: assume 8-bit machine

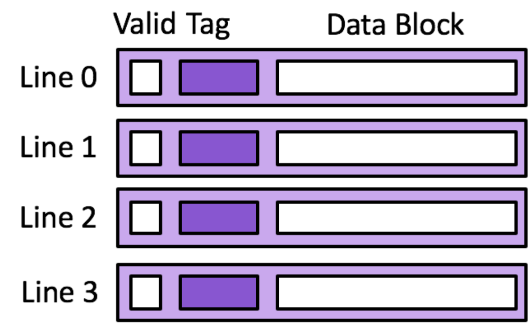


Exercise: Direct-mapped Cache

Memory

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

Cache



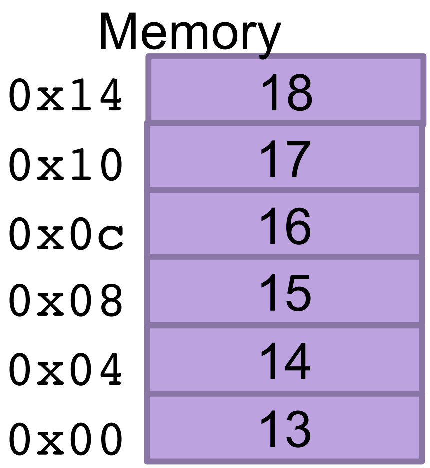
Assume 4 byte data blocks

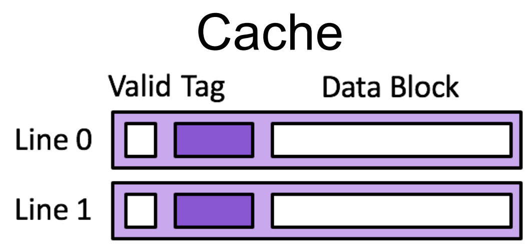
Access	tag	idx	off	h/m
rd 0x00				
rd 0x04				
rd 0x14				
rd 0x00				
rd 0x04				
rd 0x14				

Line 0			Line 1			Line 2			Line 3		
0	00	47	0	01	47	0	10	47	0	11	47

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

Exercise: Direct-mapped Cache





Access	tag	idx	off	h/m
rd 0x00				
rd 0x04				
rd 0x14				
rd 0x00				
rd 0x04				
rd 0x14				

Assume 8 byte data blocks

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

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