CS107 Lecture 25
Optimization, Caching, Writing Cache-Friendly Code

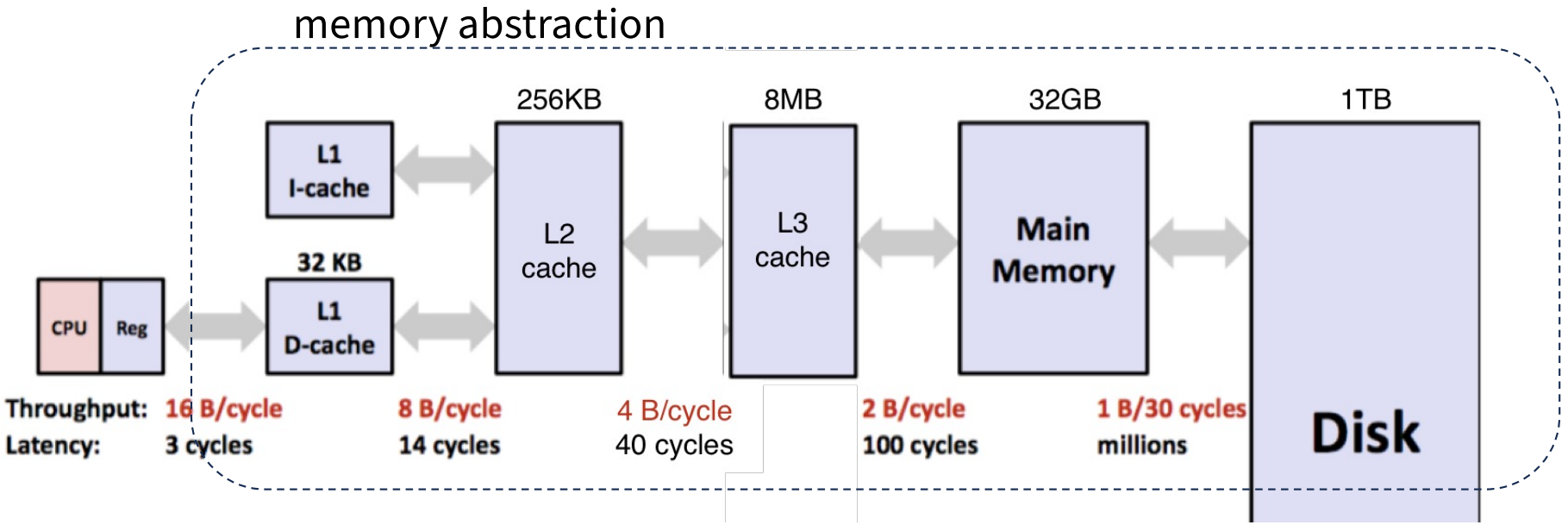
Reading: B&O 5

Caching

Processor speed is **not the only bottleneck** in program performance—memory access is perhaps even more of one!

Memory exists in levels and goes from **very fast** (registers) to **very slow** (disk).

Frequently accessed data tends to stay in faster memory.



Caching

Virtually all caching depends on **locality**.

Temporal locality

- Repeat access to the same data tends to be co-located in **time**
- **Intuition**: Memory I've accessed recently is likely to be accessed again very soon.

Spatial locality

- Related data tends to be collocated in **space**.
- **Intuition**: Memory I've recently accessed might be right next to memory I'll need very soon.

Realistic scenario:

97% cache hit rate, cache hits incur 1 clock cycle, whereas cache misses incur 100

Thought Question: How much time is spent on the 3% of accesses that are cache misses?

Caching

Virtually all caching depends on **locality**.

Credit: Kayvon Fatahalian and CS149

Array of 16 bytes in memory

	Address	Value
Line 0x0	0x0	16
	0x1	255
	0x2	14
	0x3	0
Line 0x4	0x4	0
	0x5	0
	0x6	6
	0x7	0
Line 0x8	0x8	32
	0x9	48
	0xA	255
	0xB	255
Line 0xC	0xC	255
	0xD	0
	0xE	0
	0xF	0

Assume:

Total cache capacity of 8 bytes

Cache with 4-byte cache lines
(So 2 lines fit in cache)

time

Address accessed	Cache action	Cache state (after load is complete)
0x0	"cold miss", load 0x0	0x0
0x1	hit	0x0
0x2	hit	0x0
0x3	hit	0x0
0x4	"cold miss", load 0x4	0x0, 0x4
0x5	hit	0x0, 0x4
0x6	hit	0x0, 0x4
0x7	hit	0x0, 0x4
0x8	"cold miss", load 0x8 (evict 0x0)	0x8, 0x4
0x9	hit	0x8, 0x4
0xA	hit	0x8, 0x4
0xB	hit	0x8, 0x4
0xC	"cold miss", load 0xC (evict 0x4)	0x8, 0xC
0xD	hit	0x8, 0xC
0xE	hit	0x8, 0xC
0xF	hit	0x8, 0xC
0x0	"capacity miss", load 0x0 (evict 0x8)	0x0, 0xC

Caching

The CPU can **execute instructions quickly**, but if it must wait for memory, it sits idle. Caches keep data the CPU needs **as close to it as possible**. Note: RAM officially holds the **data** and caches store **copies**.

Demo: `cache.c`