Memory hierarchy

Recap thus far

Architecture studies HW/SW interface (“how a computer works”)

ISAs: interface between high-level languages and hardware

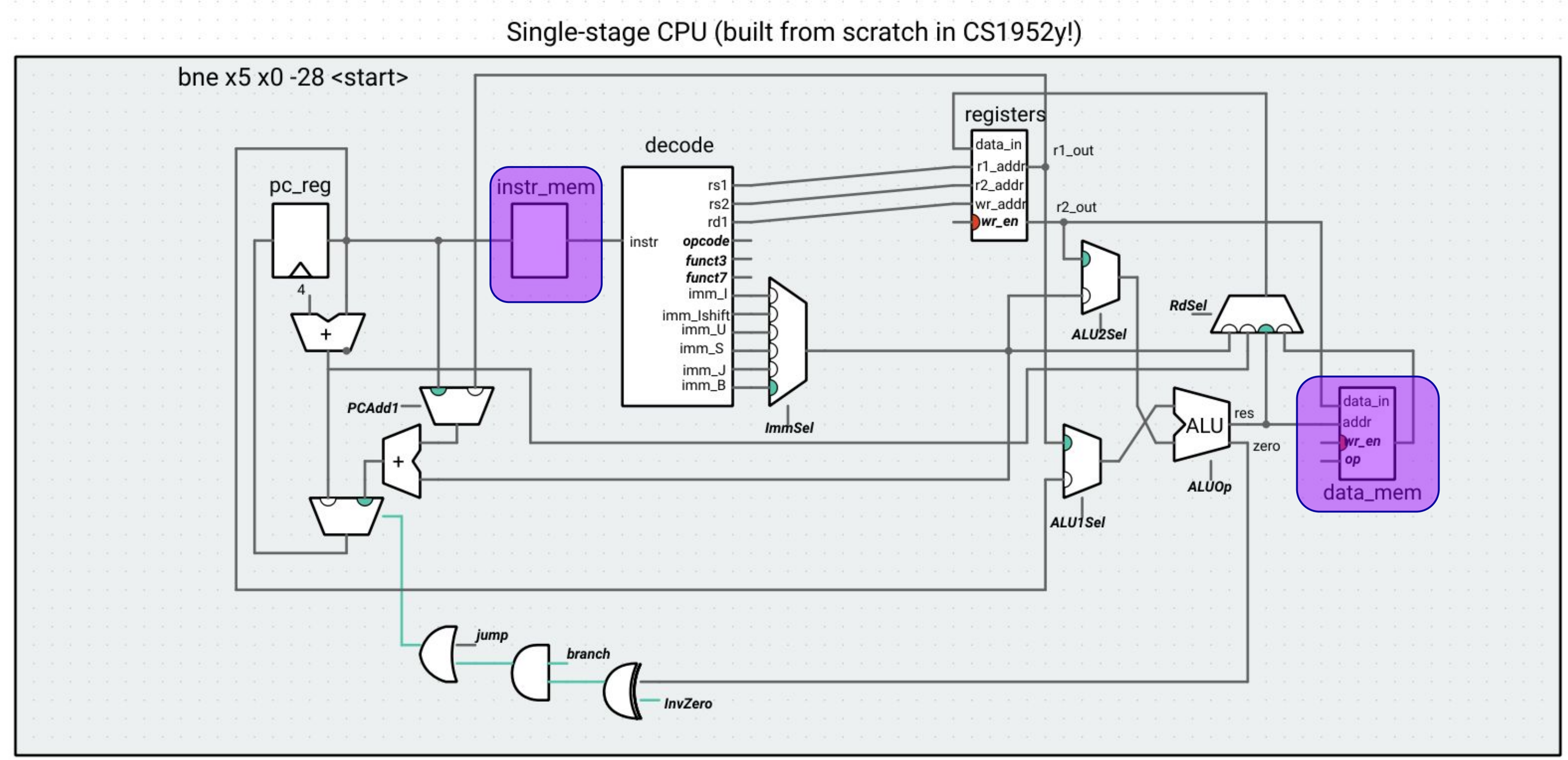
Microarchitectures: implementation of ISAs

Focused mostly on correctness, started talking about speed/throughput and *vaguely* power

We’ve built two CPUs (and two more in HW2), which can execute programs from memory…

…assuming the program exists in memory and memory works!

What do we know (so far) about memory?



There's some way that data persists on a computer
Data being accessed by addresses

- RISC-V: byte-addressing (each byte in memory has a unique address)
- Endianness allows us to interpret data that is larger than one byte in memory

Memory is way slower than registers
There is some sort of hardware that "takes care of" memory accesses for us

Persistent storage

HDD (Hard Disk Drive)

Magnet-based
10s of TBs of capacity
<$15/TB
100s of MB/s read speed (ms access time)
100s yrs lifetime

Cheaper
Slower
Bigger

SSD (Solid State Disk)

Flash-based
TBs of capacity
<$100/TB
10s of GB/s read speed (10s of us access time)
~10 yrs lifetime

Pricier
Faster
Smaller

Main memory

Persistent storage is *secondary* (CPU uses I/O buses to access)

Main memory is *primary*

Holds instructions + data while program is running

Volatile (does not persist when power is turned off)

DRAM (dynamic random access memory) technology

10s of GBs of capacity

$5-$10/GB

60 ns access time

This is still *really* slow compared with modern processor clock speeds

Processor-memory performance gap

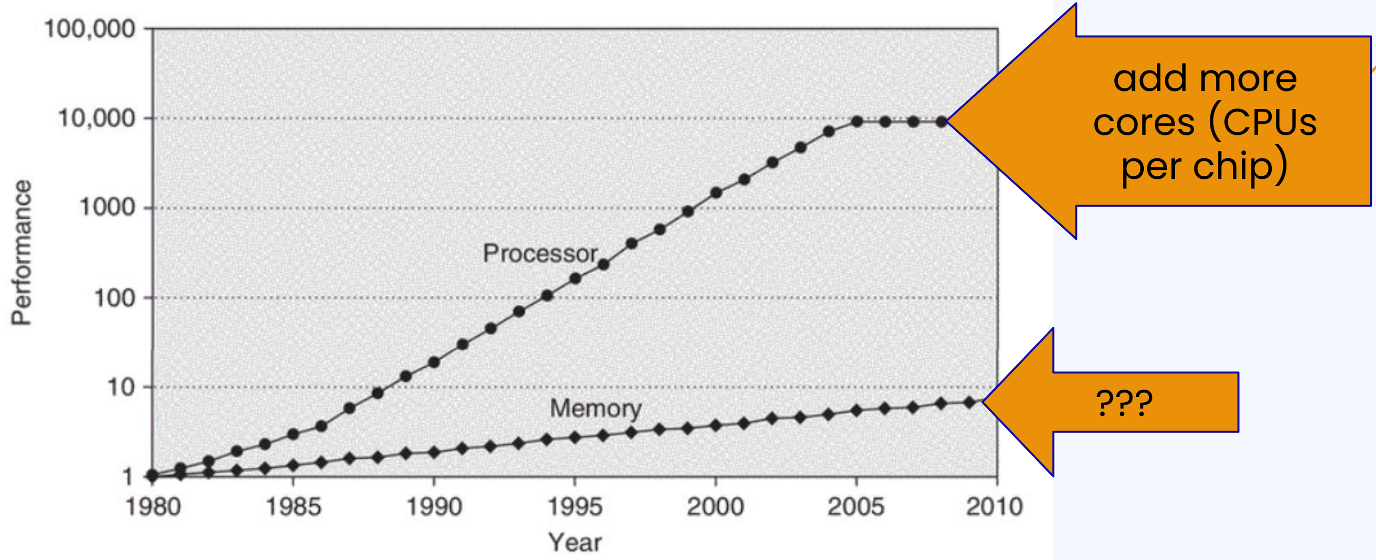


Figure 2.2 Starting with 1980 performance as a baseline, the gap in performance, measured as the difference in the time between processor memory requests (for a single processor or core) and the latency of a DRAM access, is plotted over time.

Computer Architecture: A Quantitative Approach, John L Hennessy and David A Patterson

SRAM

"Static Random Access Memory"

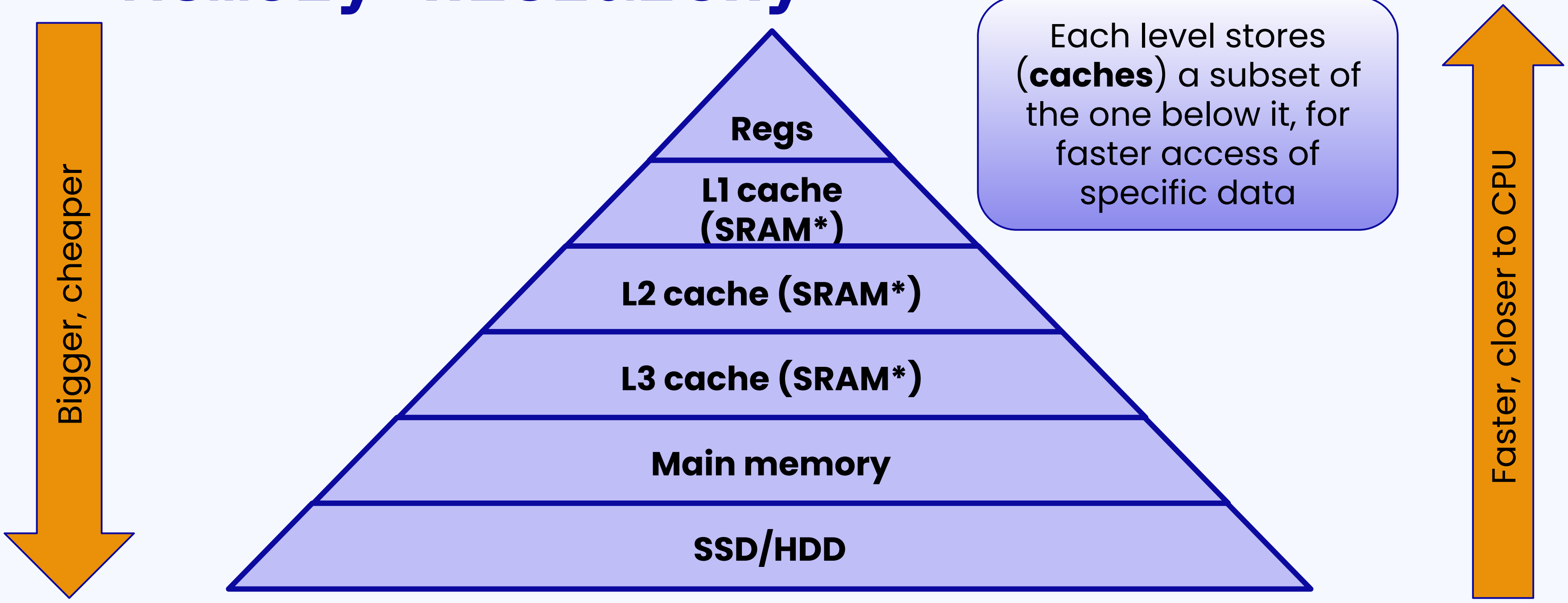
Less dense than DRAM (more $$$/area) but faster

1-100s of MB of capacity

0.5-10s of ns access time

Not practical to use for main memory, but want to take advantage of speed – what to do? **Caching**

Memory hierarchy



Each level stores (**caches**) a subset of the one below it, for faster access of specific data

? ? ?

L1-L3 caches use the same technology (SRAM).
Why do they have different access speeds?

Modern CPU layouts

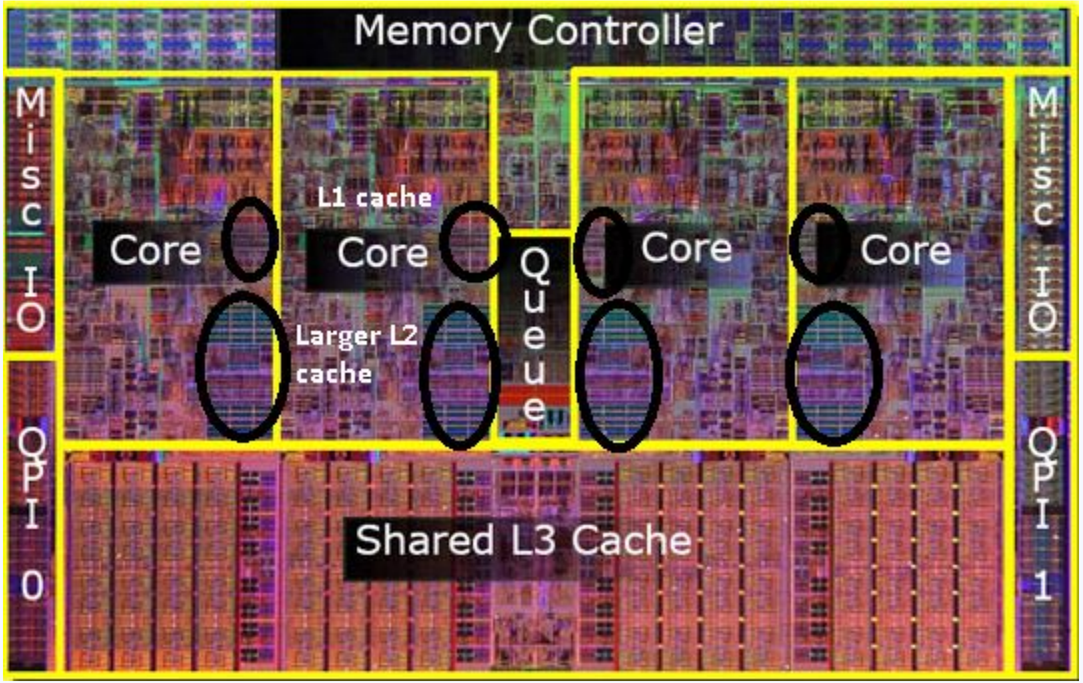


image source

Modern CPU specs

	AMD Ryzen 7 7800X3D	Apple M1	Intel Core i5-13600K
L1 cache	64 KB / core	192 KB / core* I-cache, 128 KB / core* D-cache	80 KB / core
L2 cache	1 MB / core	12 MB / core*	2 MB / core
L3 cache	96 MB	8MB*	24 MB

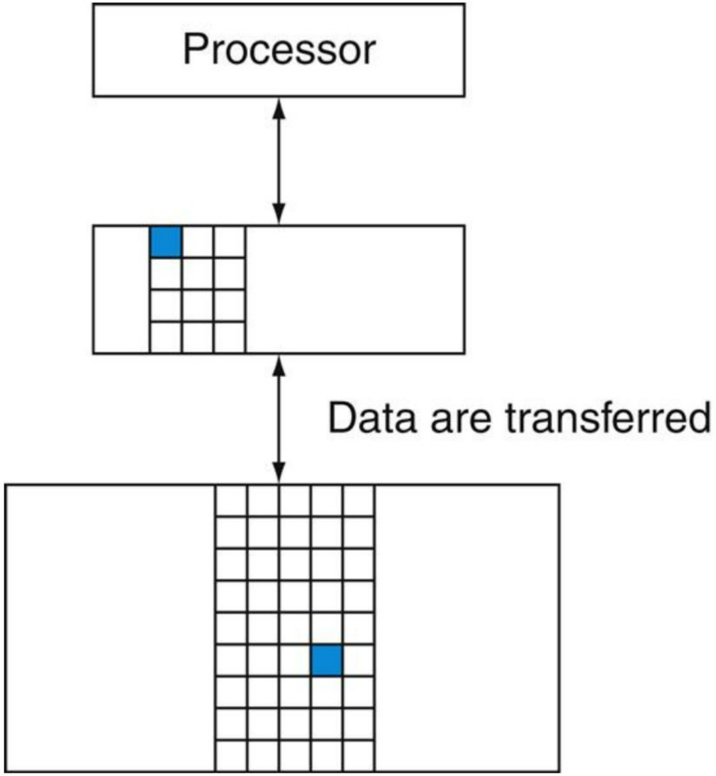
Blocks

Block (or line) - minimum unit of information that can be present/not present in a cache

Blocks are transferred between levels in the memory hierarchy

Modern CPUs: often 64 bytes (128 for M1)

Fundamental challenge of caching: smaller cache needs to be able to bring in, keep, and evict any data from larger cache



P&H 5.2

Principle of locality

Observations about the ways programs access data

Temporal locality: if data is referenced, it will tend to be referenced again soon

Spatial locality: if data is referenced, data whose addresses are close by will tend to be referenced soon

Is a for-loop that iterates through an array once an example of spatial or temporal locality?

In the context of caching, why is spatial locality useful? Why is temporal locality useful?

Spatial: Pulling in subsequent memory locations into the cache

Temporal: Keeping stuff around in cache after it’s been used (LRU eviction)

? ? ?

What questions do we need to ask when designing a cache?

- How do we decide what goes where in a cache?
- What is our scheme of maintaining consistent data?
- How do we decide what data to evict?
- How do we build an efficient memory hierarchy? (What is “efficient”)
- What control information do we need to keep around in order to implement our cache?