CS61C: Great Ideas in Computer Architecture (aka Machine Structures)

Lecture 24: Caches Part 1

Instructors: Dan Garcia, Lisa Yan, (TA: Caroline Liu)

Last Time

- Datapath parallelism, AKA pipelining
    - Inserting registers between stages along wired components
    - Realising memory accesses are still slow
- Pipelining makes data dependencies very visible in the form of hazards
    - Structural (already solved with additional hardware)
    - Data
    - Control (found in branches)
- Fixed using hardware and logical additions!
    - Forwarding the previous instruction's output from EX into the next instruction's input to EX, based on need
        - Also called bypassing since we *bypass* the decoded register's contents passed out through ID
    - Stalls/NOPs: basically wasting instruction cycles until correct behaviour is obtained

Agenda

- Review
  - Binary Prefixes
  - Memory Hierarchy
- Intro to Caches
- Fully Associative (FA) Caches
- Intro to Direct-Mapped (DM) Caches

Binary Prefix

Kilo, Mega, Giga, Tera, Peta, Exa, Zetta, Yotta

- These are all common use prefixes ⇒ all SI (International System of Units), AKA what we've been using since middle school
  - NOTE: capital prefixes != lowercase prefixes in SI! Thus, mm != Mm
- However, most EE/CS world prefixes are <u>not</u> that!
  - Base 10 is hard to work with, base 2 is much easier
  - Confusing! Common usage of “kilobyte” means 1024 bytes, but the “correct” SI value is 1000 bytes
- Hard Disk manufacturers & Telecommunications are the only computing groups that use SI factors
  - What is advertised as a 1 TB drive actually holds about 90% of what you expect because of differences in units between OS standards and hard drive standards
    - Hard drive is actually 10^{12} bytes but OS says "what is it with a base 2?"
  - A 1 Mbit/s connection actually transfers 10^6 bps!

Chart

Name	Abbr	Factor	SI size
Kilo	K	2^{10} = 1,024	10^{3} = 1,000
Mega	M	2^{20} = 1,048,576	10^{6} = 1,000,000
Giga	G	2^{30} = 1,073,741,824	10^{9} = 1,000,000,000
Tera	T	2^{40} = 1,099,511,627,776	10^{12} = 1,000,000,000,000
Peta	P	2^{50} = 1,125,899,906,842,624	10^{15} = 1,000,000,000,000,000
Exa	E	2^{60} = 1,152,921,504,606,846,976	10^{18} = 1,000,000,000,000,000,000
Zetta	Z	2^{70} = 1,180,591,620,717,411,303,424	10^{21} = 1,000,000,000,000,000,000,000
Yotta	Y	2^{80} = 1,208,925,819,614,629,174,706,176	10^{24} = 1,000,000,000,000,000,000,000,000

kibi, mebi, gibi, tebi, pebi, exbi, zebi, yobi

- IEC Standard Prefixes

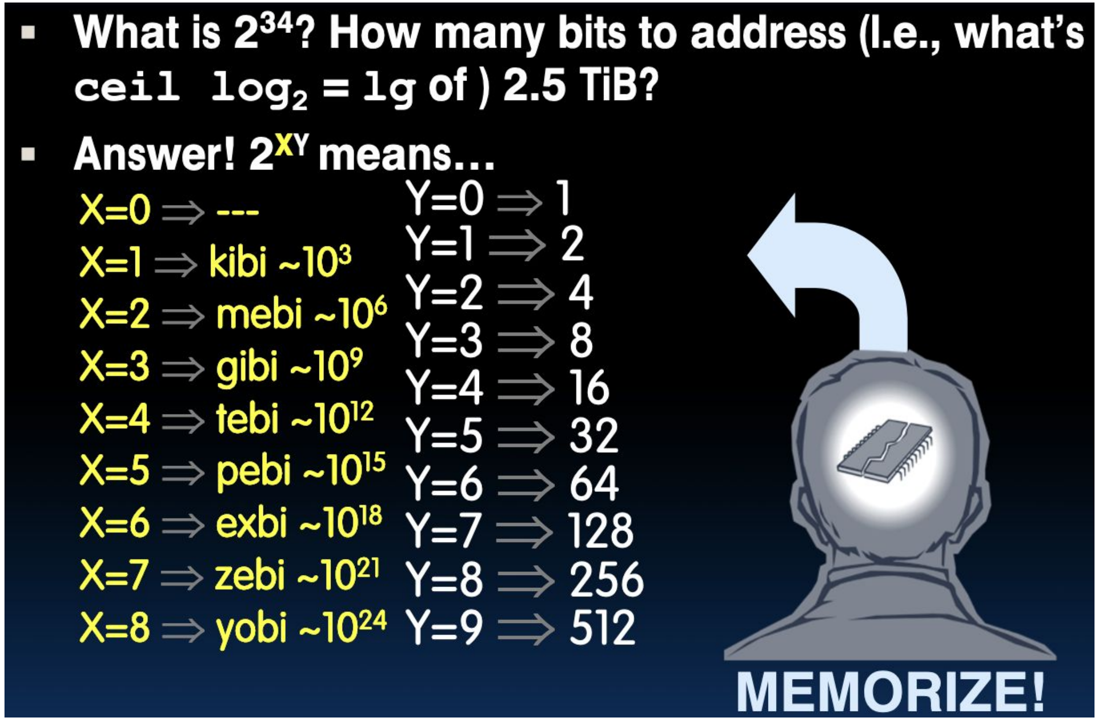
Name	Abbr	Factor
kibi	Ki	2^{10} = 1,024
mebi	Mi	2^{20} = 1,048,576
gibi	Gi	2^{30} = 1,073,741,824
tebi	Ti	2^{40} = 1,099,511,627,776
pebi	Pi	2^{50} = 1,125,899,906,842,624
exbi	Ei	2^{60} = 1,152,921,504,606,846,976
zebi	Zi	2^{70} = 1,180,591,620,717,411,303,424
yobi	Yi	2^{80} = 1,208,925,819,614,629,174,706,176

- International Electrotechnical Commission (IEC) in 1999 introduced these to specify binary quantities.
- Names come from shortened versions of the original SI prefixes (same pronunciation) and bi is short for “binary”, but pronounced “bee” :-(
- Now SI prefixes only have their base-10 meaning and never have a base-2 meaning.

Dan's Mnemonics

- **K**id **m**eets **g**iant **T**exas **p**eople **e**xercising **z**en-like **y**oga. – Rolf O
- **K**ind **m**en **g**ive **t**en **p**ercent **e**xtra, **z**estfully, **y**outhfully. – Hava E
- **K**issing **m**entors **g**ives **t**esty **p**ersistent **e**xtremists **z**ealous **y**outhfulness. – Gary M
- **K**indness **m**eans **g**iving, **t**eaching, **p**ermeating **e**xcess **z**eal **y**ourself. – Hava E
- **K**illing **m**essengers **g**ives **t**errible **p**eople **e**xactly **z**ero, yo
- **K**indergarten **m**eans **g**iving **t**eachers **p**erfect **e**xamples (of) **z**eal (&) **y**outh
- **K**issing **m**ediocre **g**iraffes **t**eaches **p**eople (to) **e**xpect **z**ero (from) **y**ou
- **K**inky **m**ean **g**irls **t**each **p**eople **e**xciting **z**en **y**oga
- **K**issing **M**el **G**ibson, **T**eddy **P**endergrass **e**xclaimed: “**z**esty, **y**o!” – Dan G
- **K**issing **m**e **g**ives **t**en **p**ercent **e**xtra **z**eal & **y**outh! – Dan G (borrowing parts)

The way to remember #s



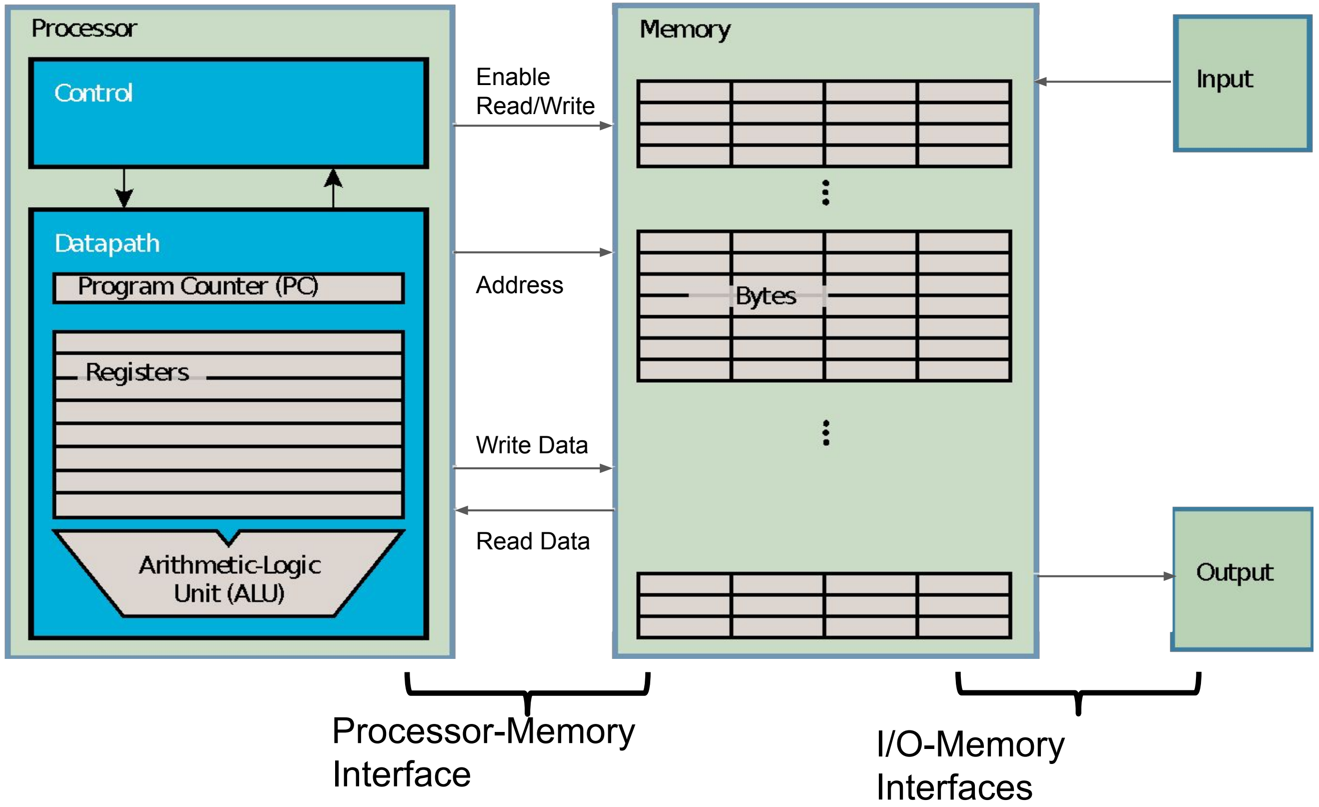
- What is 2^{34}? How many bits to address (I.e., what's `ceil` $\texttt{log}_2$ = `lg` of) 2.5 TiB?
- Answer! 2^{XY} means…

$X=0 \Rightarrow$ ---	$Y=0 \Rightarrow 1$
$X=1 \Rightarrow$ kibi ~10^3	$Y=1 \Rightarrow 2$
$X=2 \Rightarrow$ mebi ~10^6	$Y=2 \Rightarrow 4$
$X=3 \Rightarrow$ gibi ~10^9	$Y=3 \Rightarrow 8$
$X=4 \Rightarrow$ tebi ~10^{12}	$Y=4 \Rightarrow 16$
$X=5 \Rightarrow$ pebi ~10^{15}	$Y=5 \Rightarrow 32$
$X=6 \Rightarrow$ exbi ~10^{18}	$Y=6 \Rightarrow 64$
$X=7 \Rightarrow$ zebi ~10^{21}	$Y=7 \Rightarrow 128$
$X=8 \Rightarrow$ yobi ~10^{24}	$Y=8 \Rightarrow 256$
	$Y=9 \Rightarrow 512$

MEMORIZE!

Library Analogy

Components of a Computer



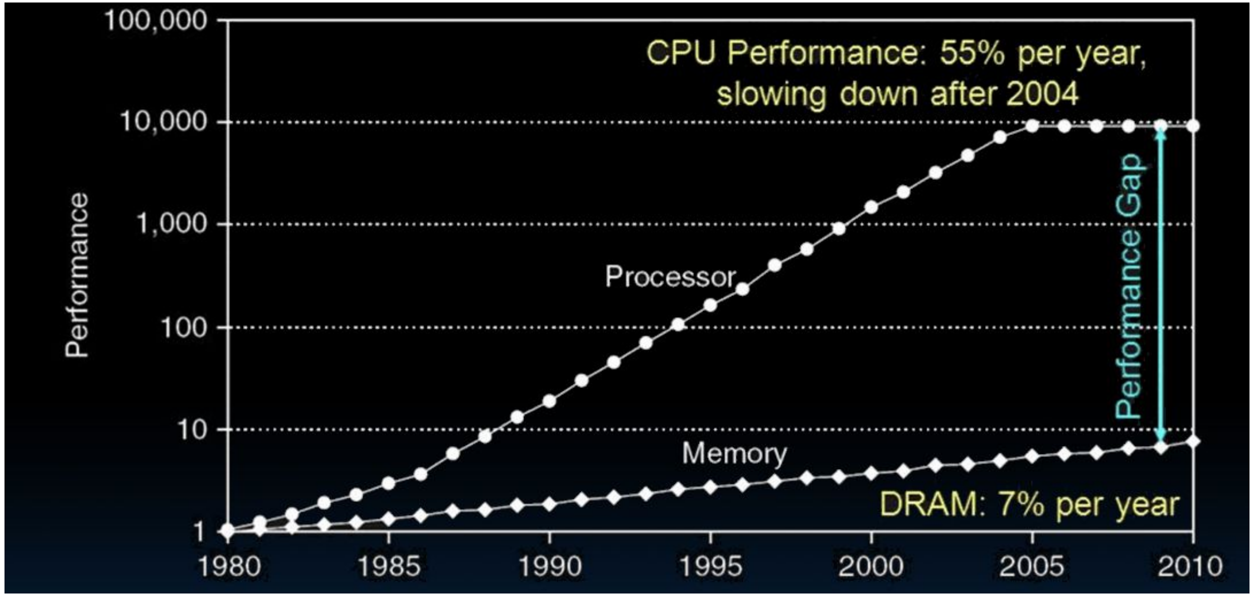
Why are large memories slow? Library Analogy

- You want to write a report using library books
  - E.g., works of J.D. Salinger
- Our current memory state is analogous to:
  - When we need a memory address, we go to the library
  - Then we search the library for the particular book we want, and the line within that book that we want to use.
  - We keep that in mind, put the book back, and return to our desk at home to continue writing our report.

Why are large memories slow? Library Analogy

- Time to find a book in a large library
  - Search a large card catalogue–mapping title/author to index number
  - Round-trip time to walk to the stacks and retrieve the desired book
- Larger libraries worsen both delays
  - Electronic memories have same issue, plus the technologies used to store a bit slow down as density increases (e.g., SRAM vs. DRAM vs. Disk)

Processor-DRAM Gap (Latency)



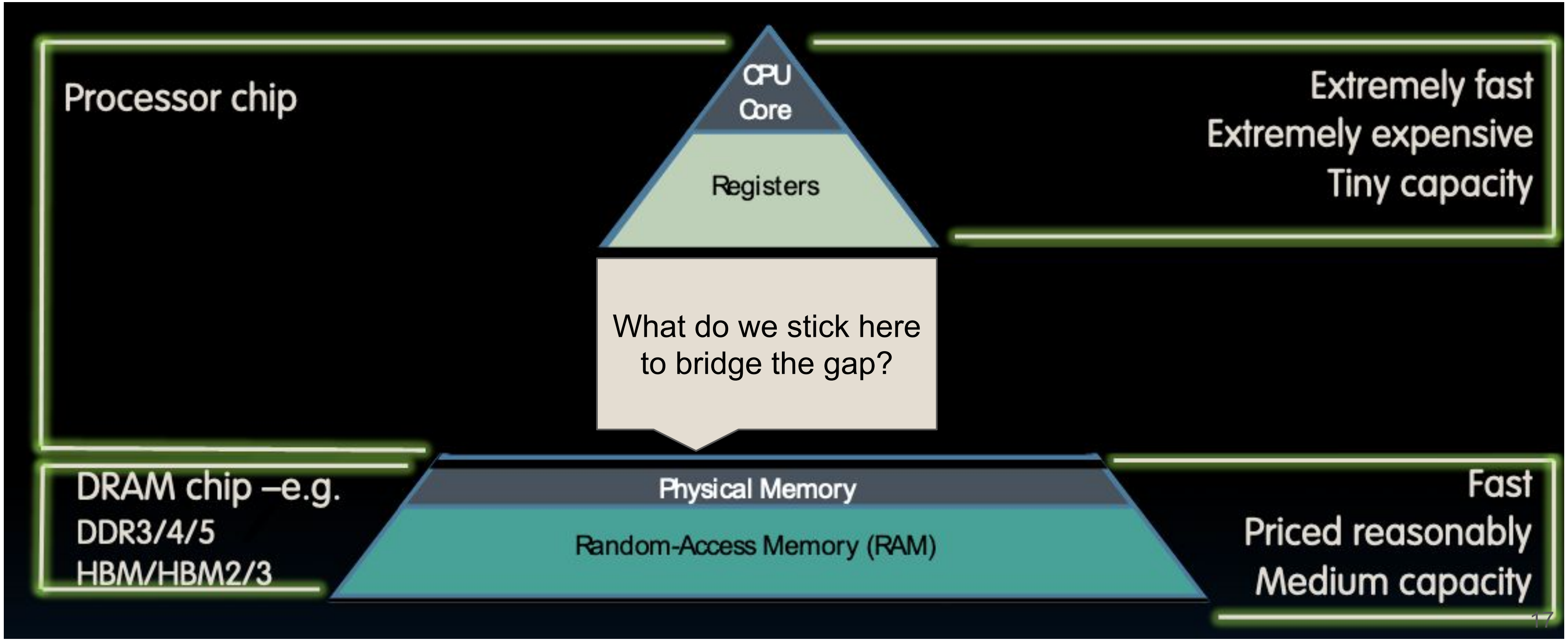
1980 microprocessor executes ~one instruction in same time as DRAM access
2020 microprocessor executes ~1000 instructions in same time as DRAM access
Slow DRAM access has disastrous impact on CPU performance!

Memory Hierarchy

What To Do: Library Analogy

- Main idea: If you're writing a report on Salinger, there's a good chance that you know what books you'll need
- If you use a book once, chances are you'll use it again.
  - Solution: Get a bookshelf at home that you can use to keep a few books, and "borrow" the book from the library when you read it.
- If we need another book, go back to the library and borrow a new one
  - But don’t return earlier books since you might need them, until your bookshelf is too full to carry more books
- You hope this collection of ~10 books on desk enough to write report, despite 10 being only 0.00001% of books in UC Berkeley libraries

Great Idea #3: Principle of Locality / Memory Hierarchy



Memory Caching

- Mismatch between processor and memory speeds leads us to add a new level ⇒ introducing a "memory cache"
- Implemented with same IC processing technology as the CPU (usually integrated on same chip)
    - Faster but more expensive than DRAM memory, and typically implemented with SRAM
    - NOTE: SRAM := Static Random Access memory, DRAM := Dynamic Random Access Memory
        - Latencies are inherent to how the memory is implemented in hardware (see notes for more details)
- Cache is a (usually) copy of a subset of main memory ⇒ when we bring something into the cache, we make a copy of the chunk from main memory
    - Sometimes we "move" the chunk from main memory to the cache (instead of copying)
    - Copying ⇒ inclusive caching; moving ⇒ exclusive caching; we typically consider inclusive
- Most processors have separate caches for instructions and data ⇒ I$, D$

Memory Hierarchy

- If level closer to Processor, it is:
  - Smaller
  - Faster
  - More expensive
  - subset of lower levels (contains most recently used data)
- Lowest Level (usually disk=HDD/SSD) contains all available data (does it go beyond the disk?)
- Memory Hierarchy presents the processor with the illusion of a very large & fast memory

Locality, Design, Management

Memory Hierarchy Basis

- Caches are an intermediate memory level between fastest and most expensive memory (registers) and the slower components (DRAM)
  - DRAM can be considered a "cache" for caches/registers an disk in a way!
- Cache contains copies of data in memory that are being used.
- Memory contains copies of data on disk that are being used.
- Caches work on the principles of temporal and spatial locality.
  - Temporal locality (locality in time): If we use it now, chances are we’ll want to use it again soon.
  - Spatial locality (locality in space): If we use a piece of memory, chances are we’ll use the neighboring pieces soon.

What to do about locality?

- Temporal Locality
  - If a memory location is referenced then it will tend to be referenced again soon
  - Keep most recently accessed data items closer to the processor
- Spatial Locality
  - If a memory location is referenced, the locations with nearby addresses will tend to be referenced soon
  - Move blocks consisting of contiguous words closer to the processor

Cache Design

- How do we organize cache?
  - Depends on how we want the cache to behave! There are 3 major categories in the upcoming lectures:
    - Direct-mapped (everything in memory can only map to one location in a cache)
    - Fully-associative (everything can go anywhere)
    - N-way set associative (somewhere in between)
- Where does each memory address map to?
  - Remember that cache is subset of memory, so multiple memory addresses map to the same cache location!
- How do we know which elements are in cache?
  - We attempt to perform memory operations by accessing cache first, and if needed, go to memory!
- How do we quickly locate them?
  - TIO (tag-index-offset) breakdown of the memory address!

How is the hierarchy managed?

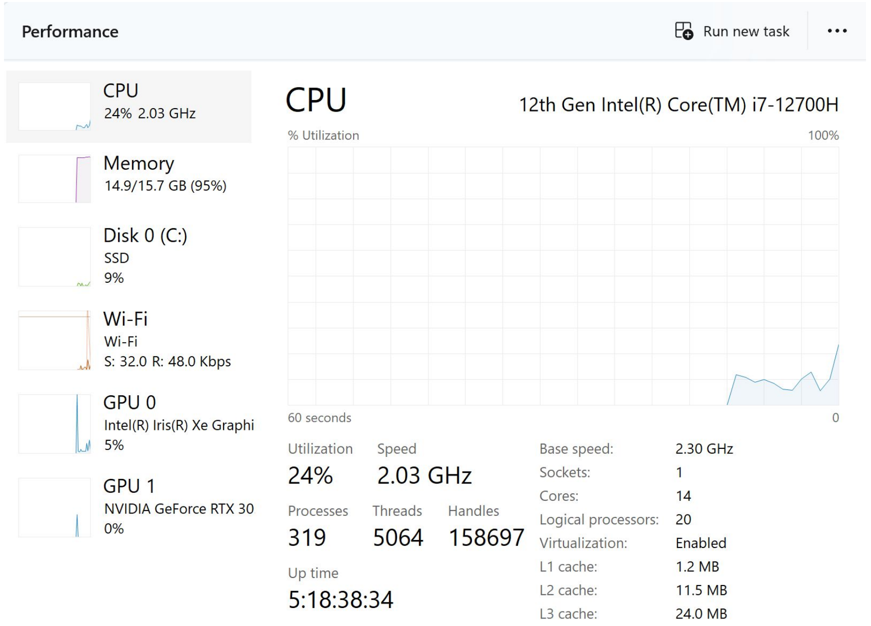
- registers « memory
  - By compiler (or assembly level programmer)
- cache « main memory
  - By the cache controller hardware
- main memory « disks (secondary storage)
  - By the operating system (virtual memory)
  - Virtual to physical address mapping assisted by the hardware (‘translation lookaside buffer’ or TLB)
  - By the programmer (files)
  - Also a type of cache

And in conclusion…

- Caches provide an illusion to the processor that the memory is infinitely large and infinitely fast

Can we do even better?

- If "caches" are a cache between registers (on-chip) and main memory, and main memory is a "cache" between the cache and the hard drive… why not have more levels bridging the gaps in between?
  - We can! And in reality, we will!
- Are caches always going to help us?
  - In theory, yes!
  - In reality? Depends on what we're trying to achieve!
  - We'll explore ways of measuring cache performance in the upcoming lectures



Utilization	24%
Speed	2.03 GHz
Processes	319
Threads	5064
Handles	158697
Up time	5:18:38:34
Base speed:	2.30 GHz
Sockets:	1
Cores:	14
Logical processors:	20
Virtualization:	Enabled
L1 cache:	1.2 MB
L2 cache:	11.5 MB
L3 cache:	24.0 MB

And in conclusion…

- We have learned how to bridge the gap between processor storage and main memory
  - Explored the potential effects of caching on access latencies
- We've explored some of the general big picture ideas for the next 3 lectures
  - Direct-mapped, fully associative, and N-way set associative caches
  - Multi-level caching
  - Measuring cache performance
- How is the memory system supported, and how can we attempt to utilise the cache to our advantage?
  - Temporal and spatial locality
  - Performance
  - Code behaviour