Course Wrap-Up

CSE 351 Spring 2024

Instructor:

Elba Garza

Teaching Assistants:

Ellis Haker
Adithi Raghavan
Aman Mohammed
Brenden Page
Celestine Buendia
Chloe Fong
Claire Wang
Hamsa Shankar
Maggie Jiang
Malak Zaki
Naama Amiel
Nikolas McNamee
Shananda Dokka
Stephen Ying
Will Robertson

Playlist: CSE 351 24Sp Lecture Tunes!

Announcements, Reminders

- Lab 5 due Friday at 11:59 PM!
- **Final Exam: June 3rd through June 5th** (released June 3rd at 00:01)
  - Same rules as midterm; details on Exams page
  - Stand-alone Final Exam review session next class
- End-of-Year Announcements:
  - Bob Bandes TA Award!
  - Please fill out course evaluations!
  - Great follow-up textbooks and reads!
  - Come meet Gigi on Monday!

Supporting Yourself While Debugging

- CS cultural norms **actively encourage** prolonged periods of mental concentration
  - Easy to tune everything else out when you remain immobile just a few feet from your (increasingly large) screen
  - Programmers describe sometimes being "in the zone"
  - Long coding sessions and late nights are socially and culturally encouraged
  - Hackathons are designed this way: ignore your bodily needs for 24 hours?!
  - Tech companies entice you to stay at work with free food and amenities. It's not because they're just being nice!

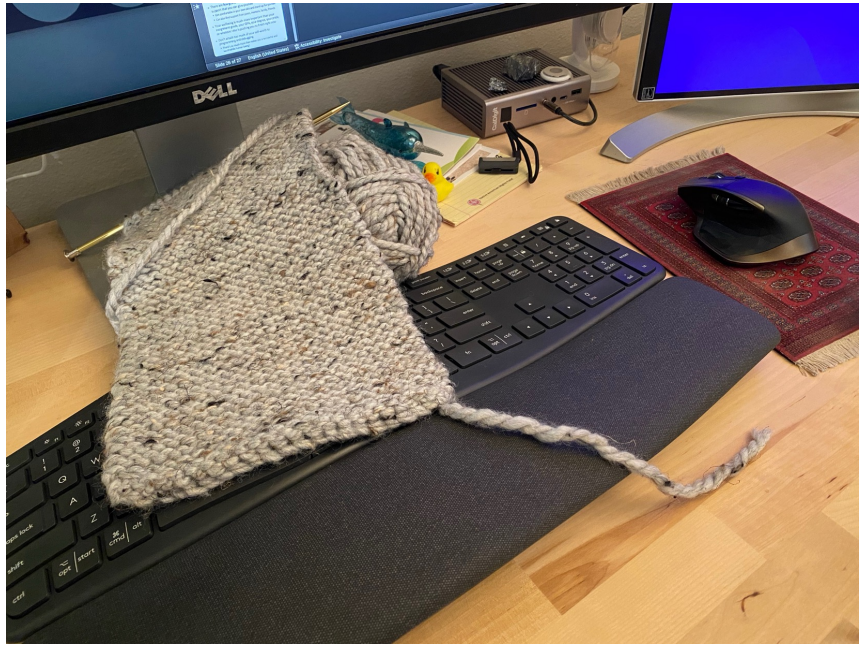
Mindfulness and Debugging

- Mindfulness: "The practice of bringing one's attention in the present moment"
  - Lots of different definitions and nuance, but we'll stick with this broad definition

- While debugging, try to be mindful of your emotional and physical state as well as your current approach
  - Am I focused on the task at hand or distracted?
  - Am I calm and/or rested enough to be thinking "clearly?"
  - How is my posture, breathing, and tenseness?
  - Do I have any physical needs that I should address?
  - What approach am I trying and **why**? Are there alternatives?

Supporting Yourself While Debugging

- Try: set a timer for some interval (e.g., 15 minutes) to evaluate your state and approach
  - Like the system timer your OS uses for context switching!
- If you're distracted, feeling negative emotions, tense, or need to address something, **take a break!**
  - You might even think of a new approach!
  - Take a shower, go for a walk, take a nap, have a snack, cook dinner, bake cookies, learn to sew…



Supporting Yourself

- There are few guarantees for support, besides the support that you can give yourself
  - Get comfortable in your own skin and stand up for yourself
  - Look to your peers, mentors, family, friends! Make use of the support systems that you do have!
- Your well-being is much more important than your assignment grade, your GPA, your degree, your pride, or whatever else is pushing you to finish right now.

Computer Science and You

- Being mindful of how you work best is important!
- CS culture tends to perpetuate certain images of what being a programmer "should" look like
- But it might not, which is also completely valid
  - **Regardless, you still belong in computer science**
  - Some personal examples: I don't do side projects, my GitHub is dead, my website is (currently) dead, I prefer to unwind away from a computer.

- The best approaches are the ones that work for you. And they don't change your value as a computer scientist or as a person.

Today

- End-to-end Review
  - What happens after you write your source code?
    - How code becomes a program
    - How your computer executes your code
- Victory lap and high-level concepts (🔑 points)
  - More useful for “5 years from now” than for the final exam
- Question time

C: The Low-Level High-Level Language

- C is a "hands-off" language that "exposes" more of hardware (especially memory)
  - Weakly-typed language that stresses data as bits
    - Anything can be represented with a number!
  - Unconstrained pointers can hold address of anything
    - And no bounds checking – buffer overflow possible!
  - Efficient by leaving everything up to the programmer
  - "C is good for two things: being beautiful and creating catastrophic 0days in memory management." (https://medium.com/message/everything-is-broken-81e5f33a24e1)

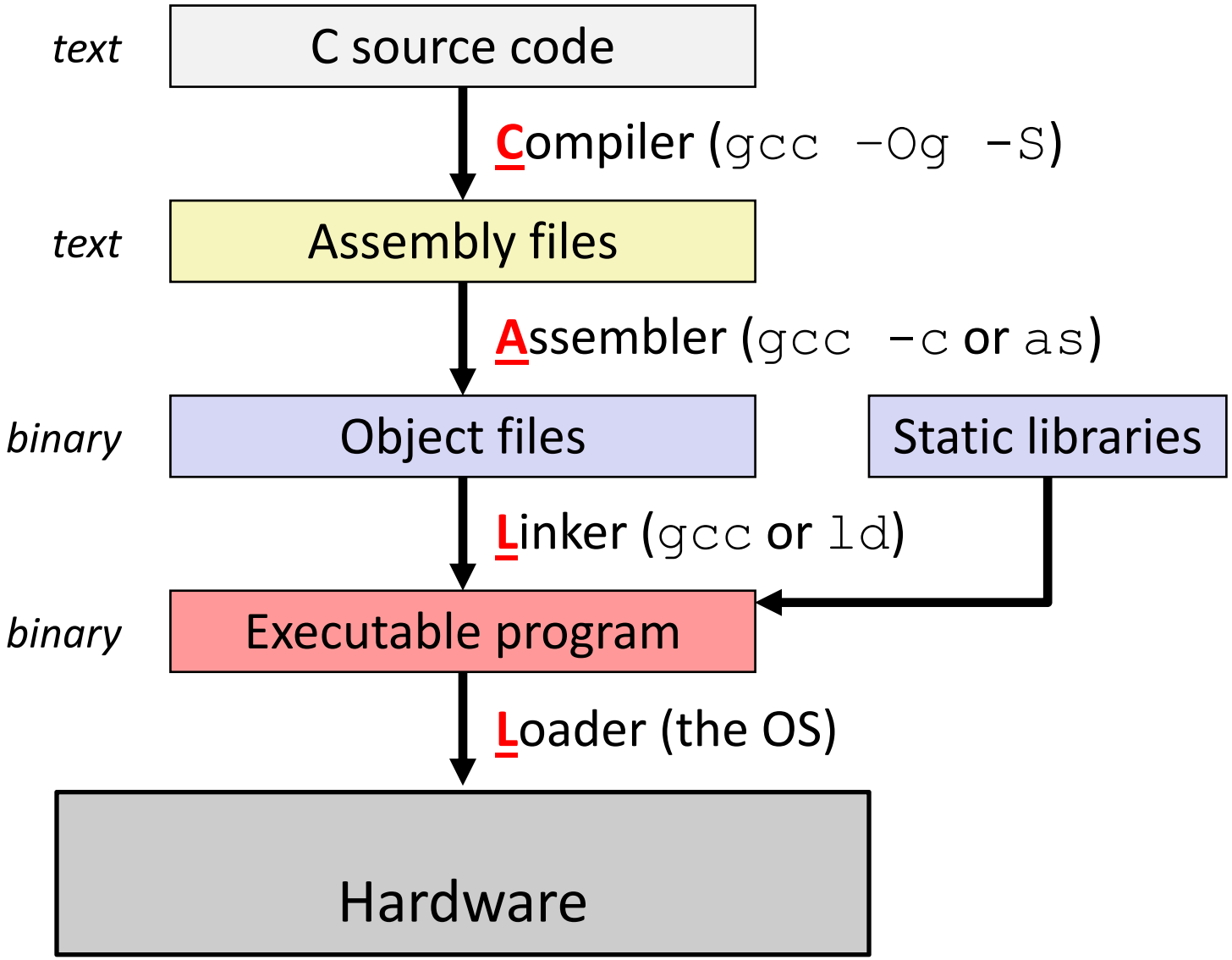
C Data Types

- C Primitive types
  - Fixed sizes and alignments
  - Characters (`char`), Integers (`short`, `int`, `long`), Floating Point (`float`, `double`)
- C Data Structures
  - Arrays – contiguous chunks of memory
    - Multidimensional arrays = still one continuous chunk, but row-major
    - Multi-level arrays = array of pointers to other arrays
  - Structs – structured group of variables
    - Struct fields are ordered according to declaration order
    - ***Internal* fragmentation:** space between members to satisfy member alignment requirements (aligned for each primitive element)
    - ***External* fragmentation:** space after last member to satisfy overall struct alignment requirement (largest primitive member)

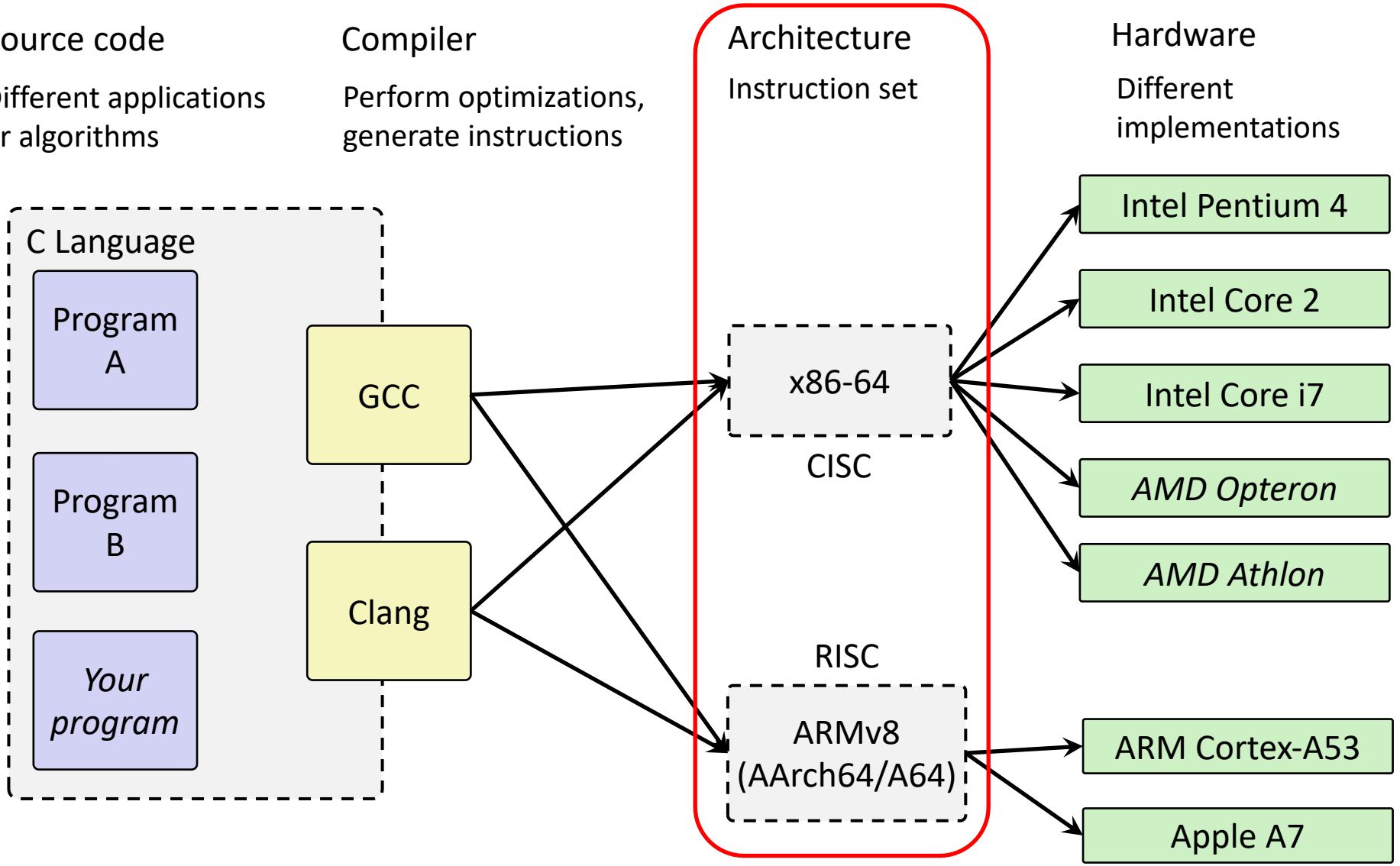
C and Memory

- Using C allowed us to examine how we store and access data in memory
  - Endianness (**only applies to memory**)
    - Is the first byte (lowest address) the least significant (little endian) or most significant (big endian) of your data?
  - Array indices and struct fields result in calculating proper addresses to access
- Consequences of your code:
  - Affects performance (locality)
  - Affects security
- But to understand these effects better, we had to dive deeper…

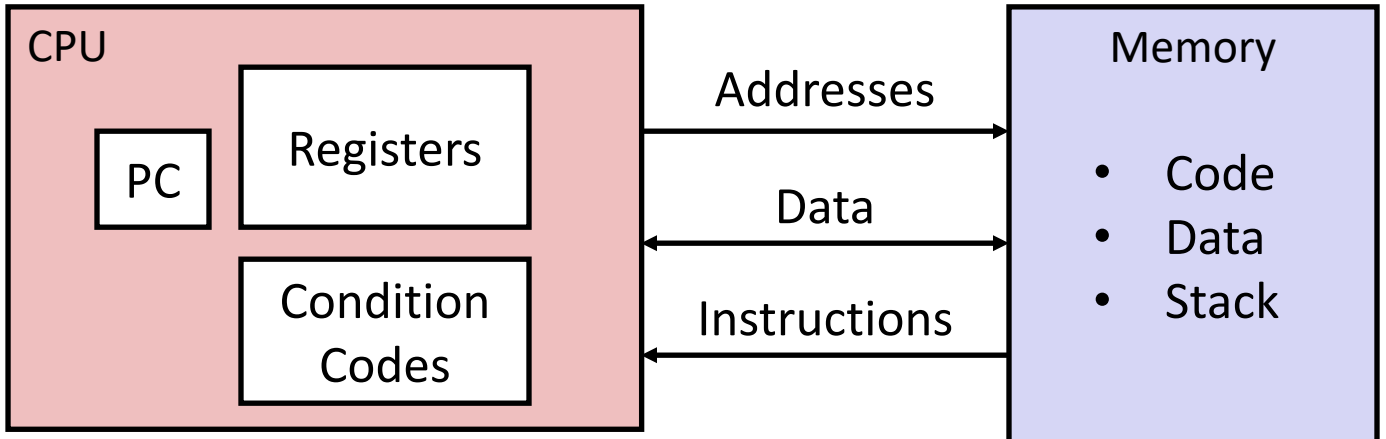
How Code Becomes a Program



Instruction Set Architecture



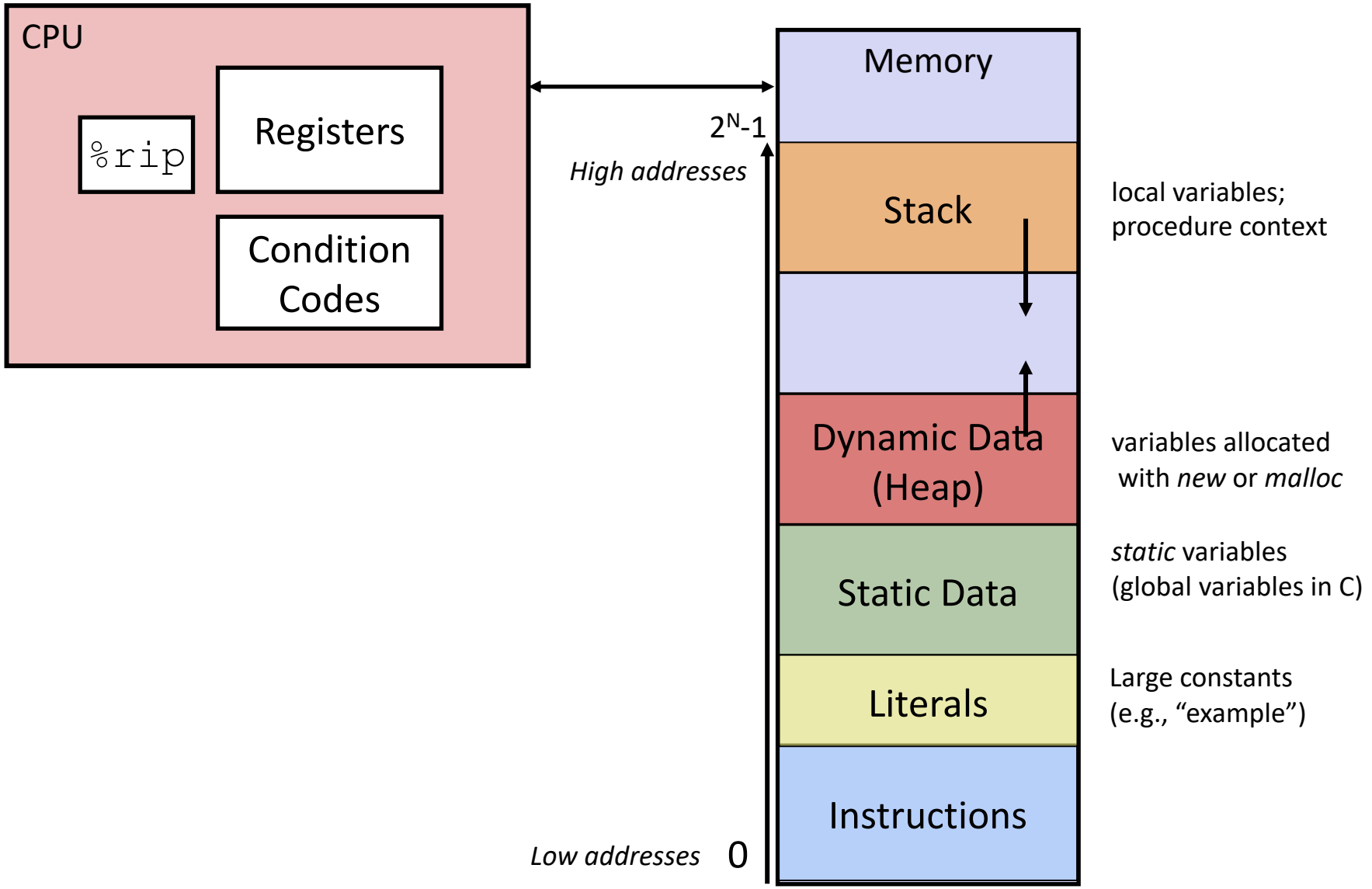
Assembly Programmer's View



- Programmer-visible state
  - PC: the Program Counter (`%rip` in x86-64)
    - Address of next instruction
  - Named registers
    - Together in "register file"
    - Heavily used program data
  - Condition codes
    - Store status information about most recent arithmetic operation
    - Used for conditional branching

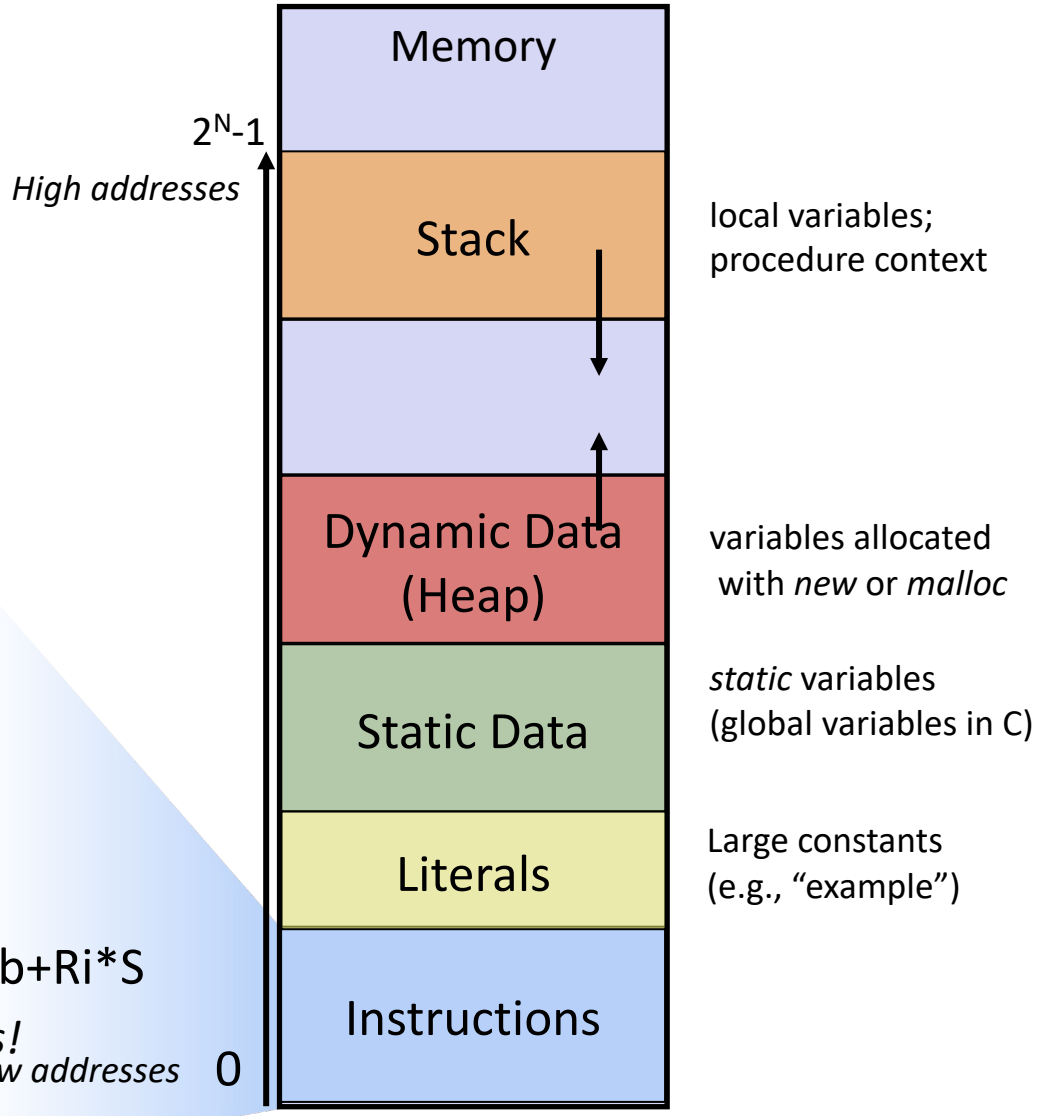
- Memory
  - Byte-addressable array
  - Huge *virtual* address space
  - *Private, all to yourself…*

Program's View: Memory Space



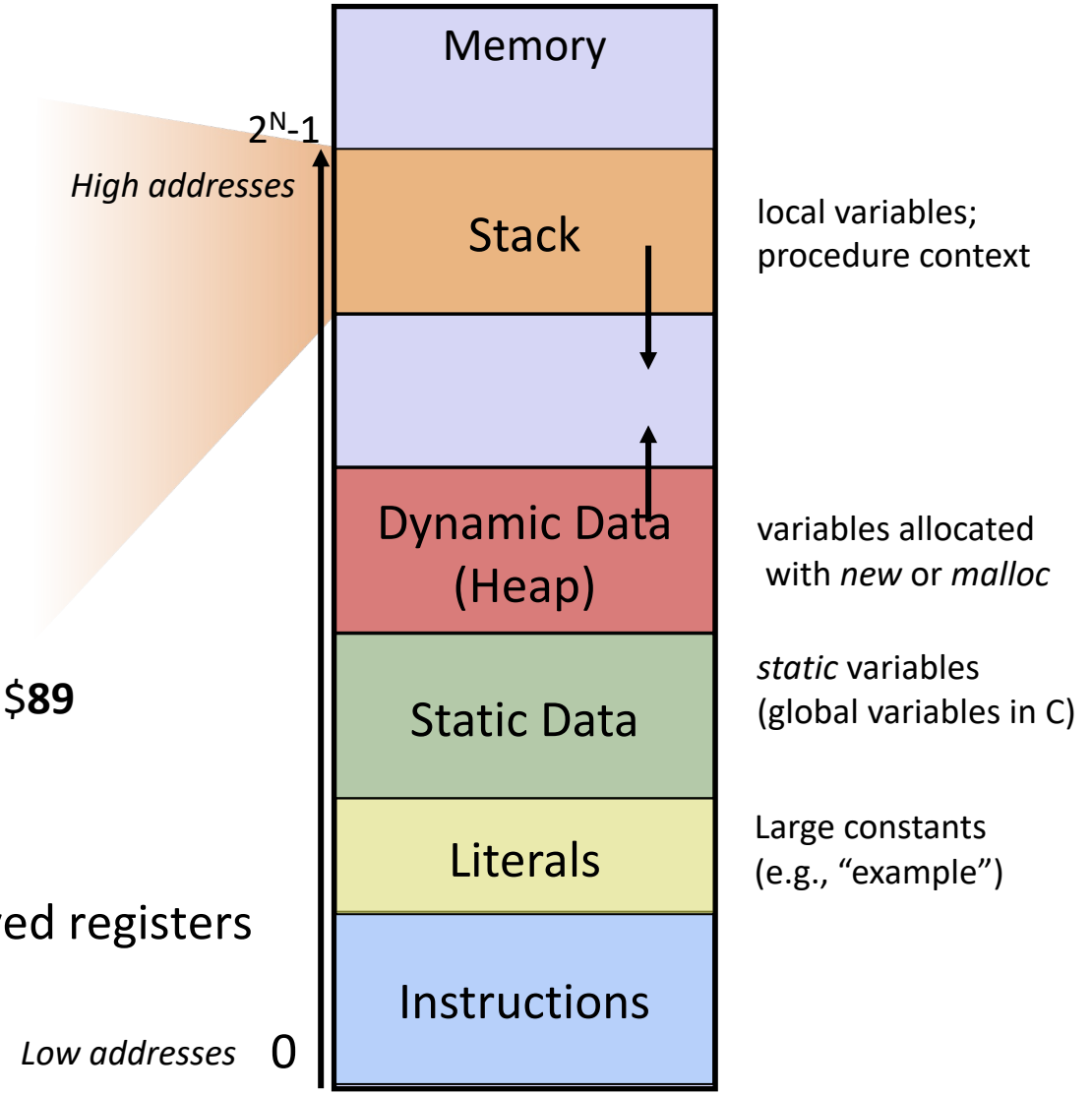
Program's View

- Instructions
  - Data movement
    - `mov, movz, movz`
    - `push, pop`
  - Arithmetic
    - `add, sub, imul`
  - Control flow
    - `cmp, test`
    - `jmp, je, jgt, ...`
    - `call, ret`
- Operand types
  - Literal: `$8`
  - Register: `%rdi, %al`
  - Memory: D(Rb,Ri,S) = D+Rb+Ri*S
    - `lea`: *not a memory access!*



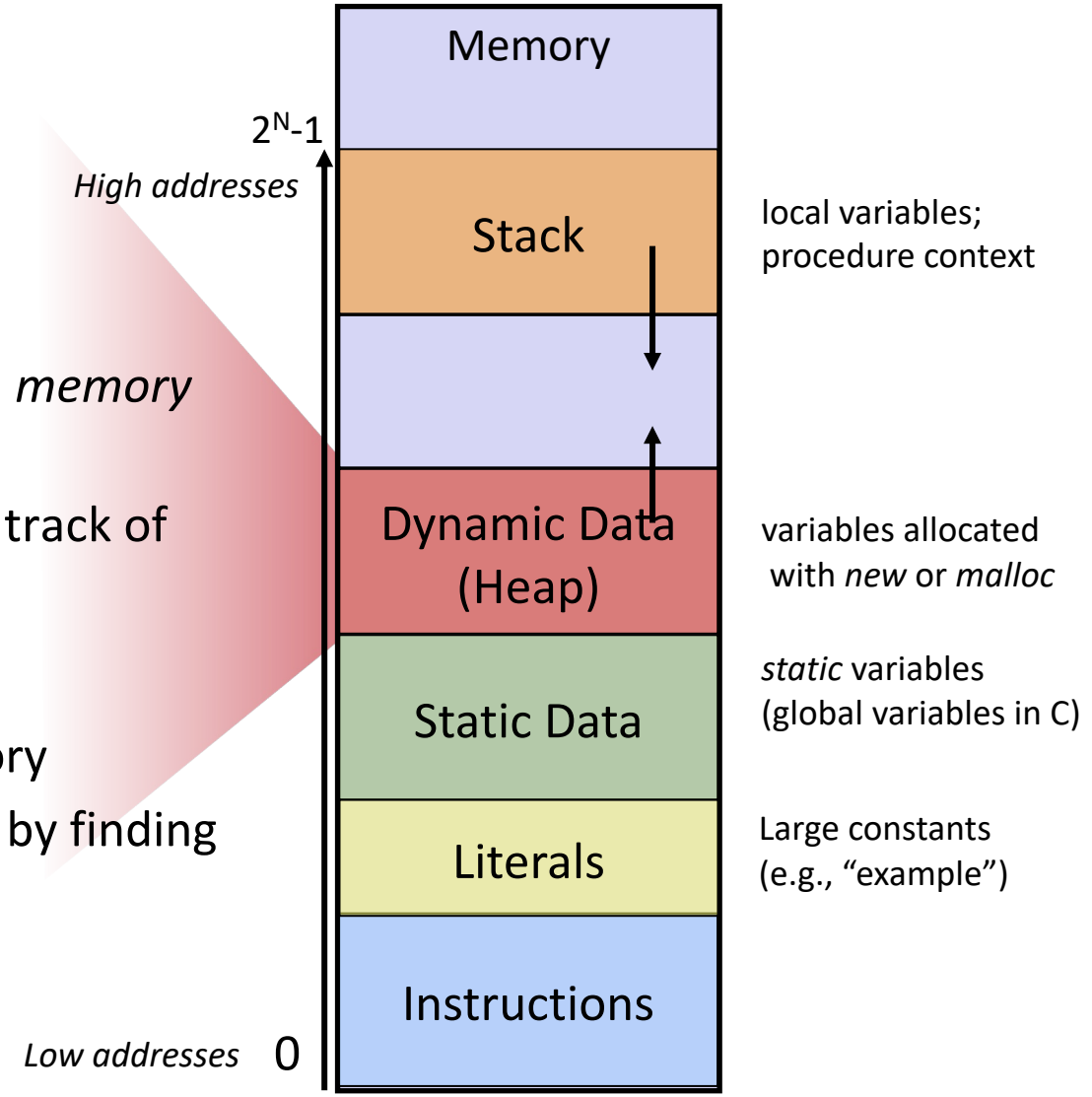
Program's View

- Procedures
  - Essential abstraction
  - Recursion…
- Stack discipline
  - Stack frame per call
  - Local variables
- Calling convention
  - How to pass arguments
    - **Di**ane's **Si**lk **D**ress **C**osts $**89**
  - How to return data
  - Return address
  - Caller-saved / callee-saved registers

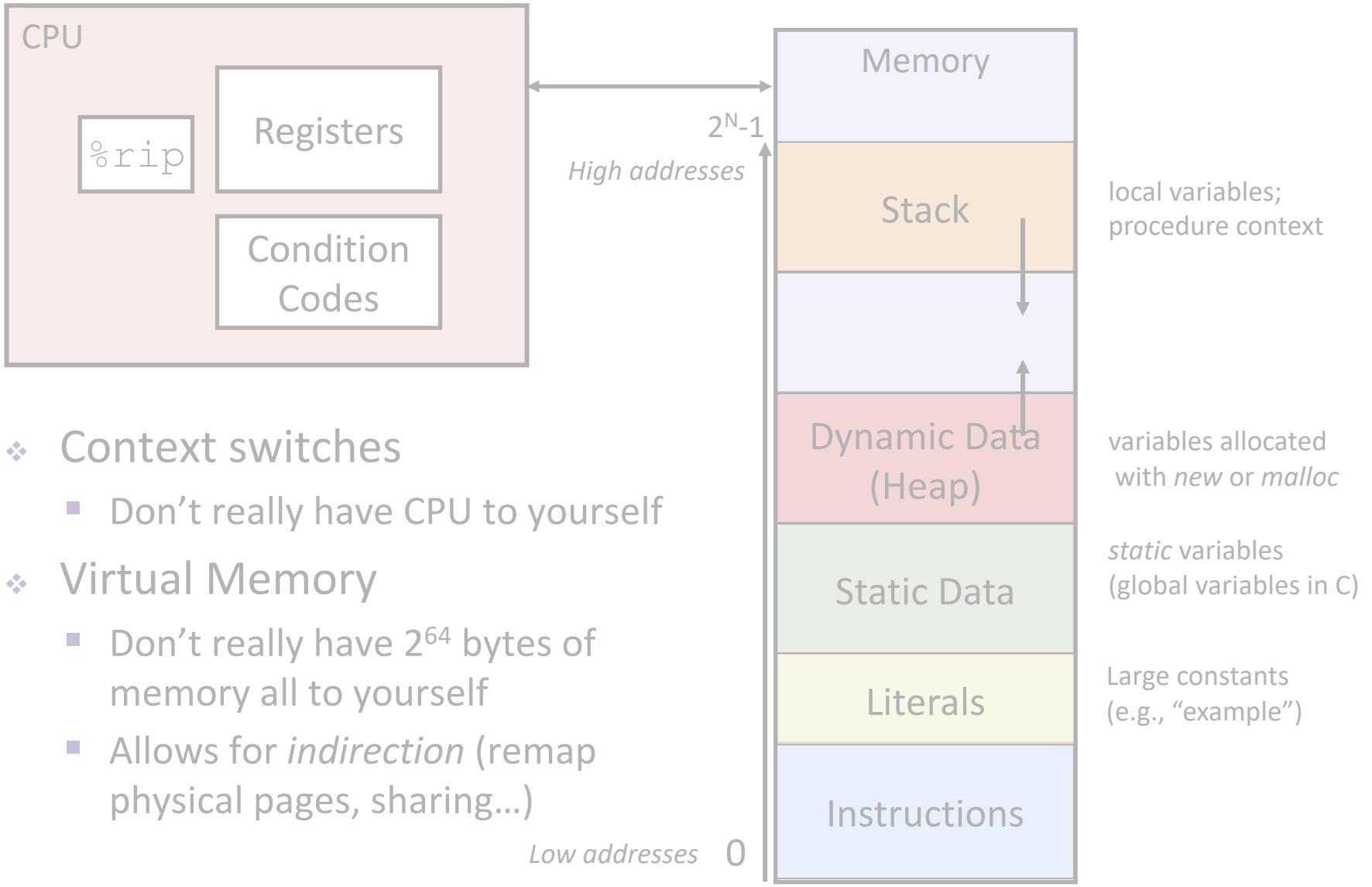


Program's View

- Heap data
  - Variable size
  - Variable lifetime
- Allocator
  - Balance *throughput* and *memory utilization*
  - Data structures to keep track of free blocks
- Garbage collection
  - Must always free memory
  - Garbage collectors help by finding anything *reachable*
  - Failing to free results in *memory leaks*

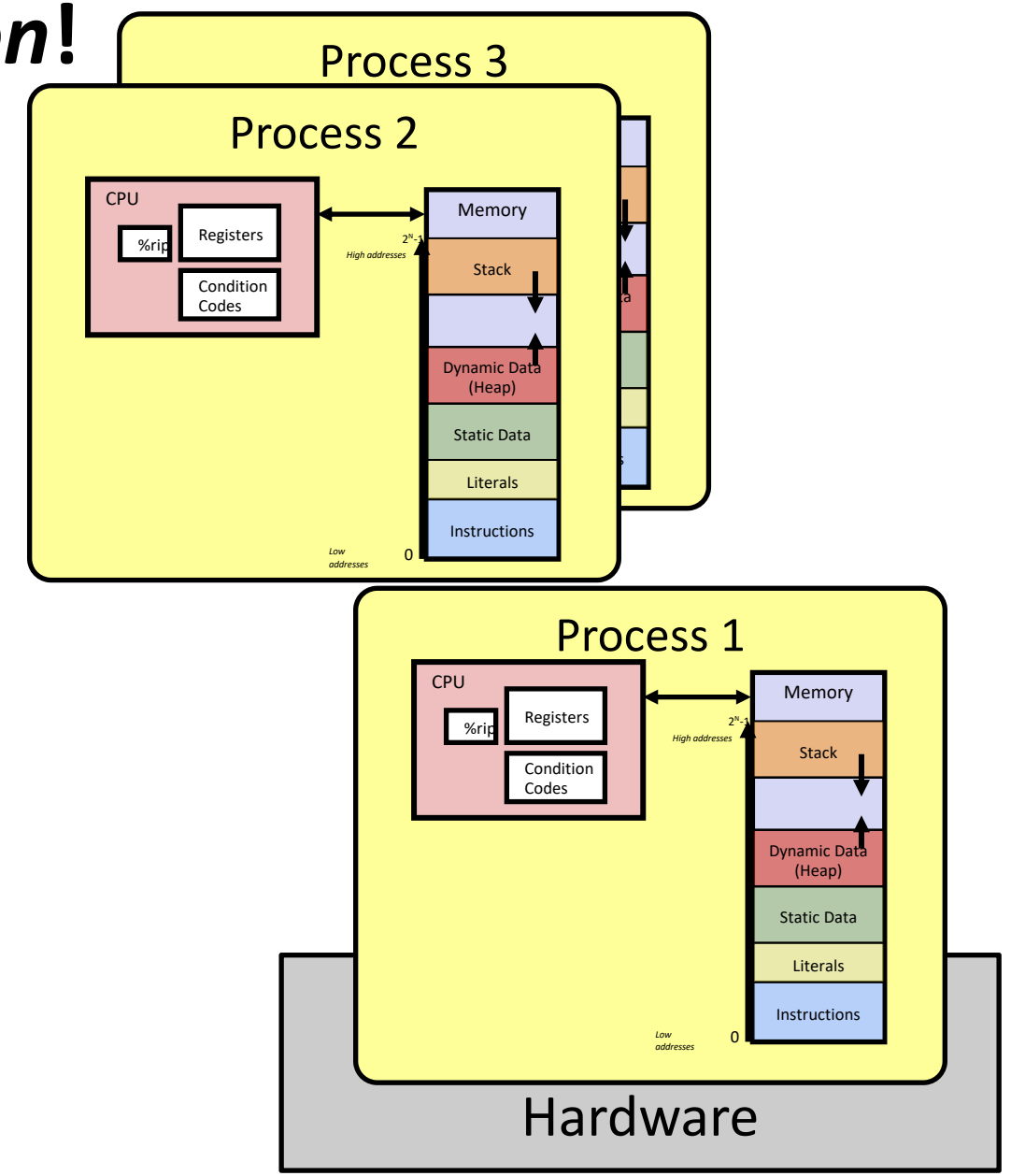


But remember… it's all an *illusion*!



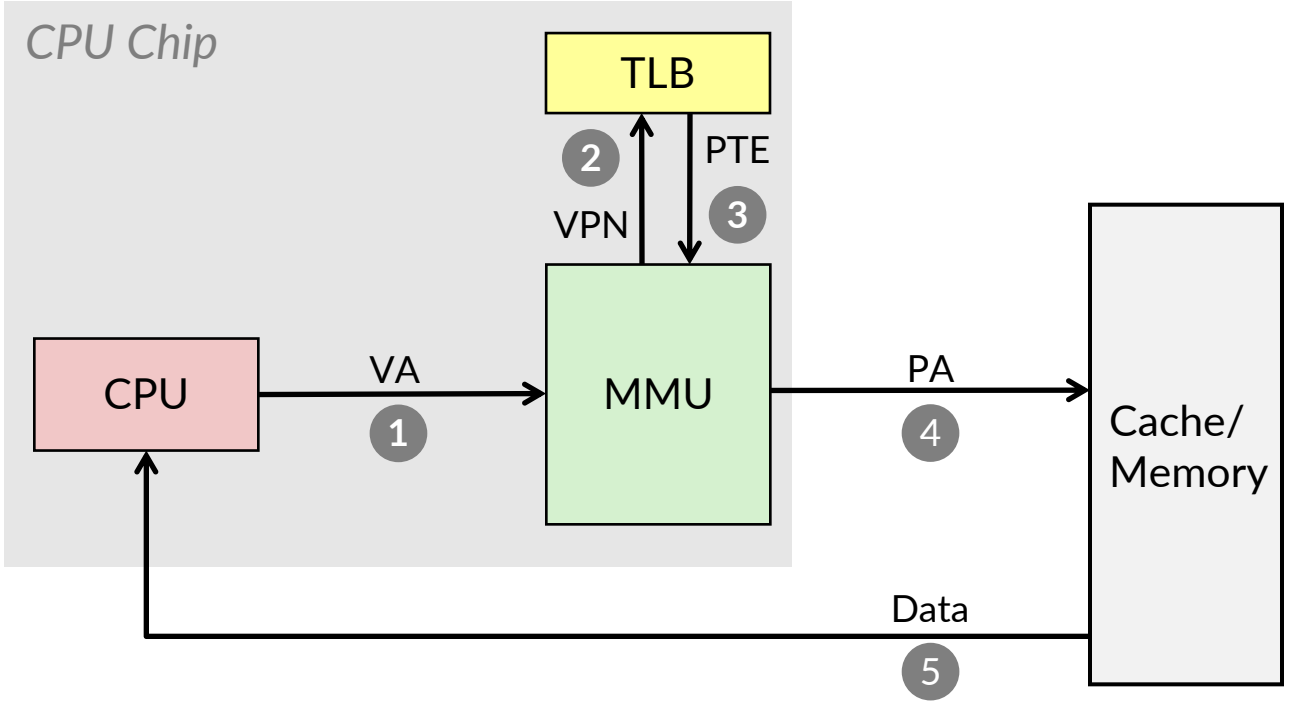
- Context switches
  - Don't really have CPU to yourself
- Virtual Memory
  - Don't really have 2^{64} bytes of memory all to yourself
  - Allows for *indirection* (remap physical pages, sharing…)

But remember… it's all an *illusion*!



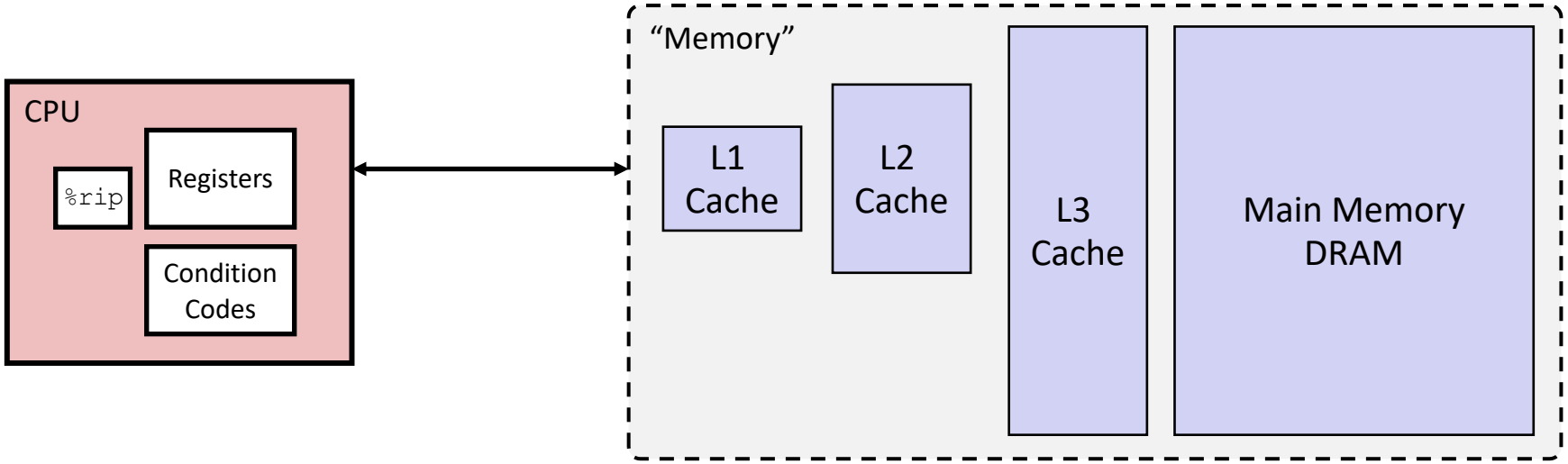
- `fork`
  - Creates copy of the process
- `execv`
  - Replace with new program
- `wait`
  - Wait for child to die (to *reap* it and prevent *zombies*)

Virtual Memory



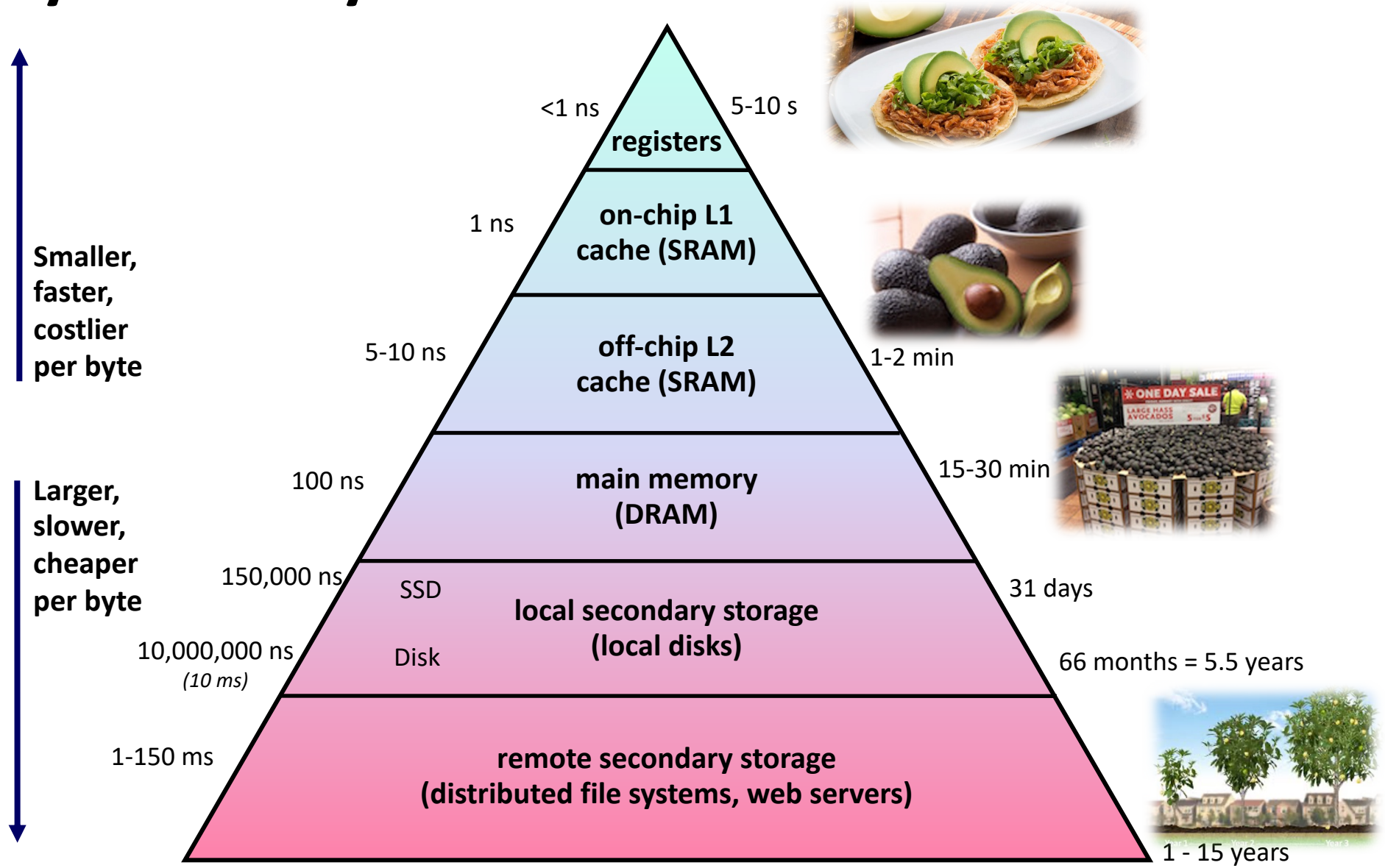
- Address Translation
  - Every memory access must first be converted from virtual to physical
  - **Indirection***:* just change the address mapping when switching processes
  - Luckily, TLB (and page size) makes it pretty fast

But Memory Itself is Also a Lie!



- *Illusion* of one flat array of bytes
  - But *caches* invisibly make accesses to physical addresses faster!
- Caches
  - **Associativity** tradeoff with miss rate and access time
  - **Block size** tradeoff with spatial and temporal locality
  - **Cache size** tradeoff with miss rate and cost

Memory Hierarchy



Victory Lap

- A victory lap is an extra trip around the track
  - By the exhausted victors (that's us) ☺

- Review course goals
  - Put everything in perspective

Big Theme 1: Abstractions and Interfaces

- Computing is about abstractions
  - (but we can't forget reality)
- What are the abstractions that we use?
- What do you need to know about them?
  - When do they break down and you have to peek under the hood?
  - What bugs can they cause and how do you find them?
- How does the hardware relate to the software?
  - Become a better programmer and begin to understand the important concepts that have evolved in building ever more complex computer systems

Little Theme 1: Representation/Encoding

- All digital systems represent everything as 0s and 1s
  - The 0 and 1 are really two different voltage ranges in the wires
  - Or magnetic positions on a disc, or hole depths on a DVD, or even *DNA*…
- "Everything" includes:
  - Numbers – integers and floating point
  - Characters – the building blocks of strings
  - Instructions – the directives to the CPU that make up a program
  - Pointers – addresses of data objects stored away in memory
- Encodings are stored throughout a computer system
  - In registers, caches, memories, disks, etc.
- They all need addresses (a way to locate)
  - Find a new place to put a new item
  - Reclaim the place in memory when data no longer needed

Little Theme 2: Translation

- There is a big gap between how we think about programs and data and the 0s and 1s of computers
  - Need languages to describe what we mean
  - These languages need to be translated one level at a time

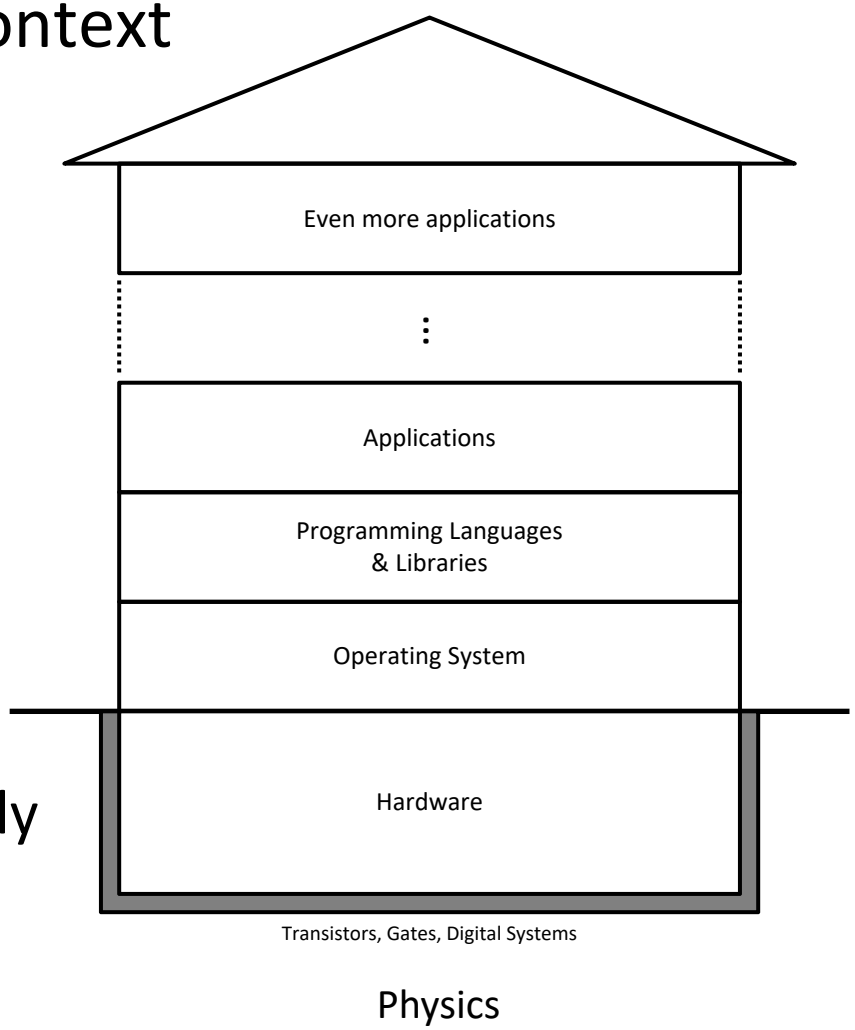
- We know Java as a programming language
  - Need to work our way down to the 0s and 1s of computers
  - Try not to lose anything in translation!
  - We encountered C language, assembly language, and machine code (for the x86 family of CPU architectures)

Little Theme 3: Control Flow

- How do computers orchestrate everything they are doing?
- Within one program:
  - How do we implement if/else, loops, switches?
  - What do we have to keep track of when we call a procedure, and then another, and then another, and so on?
  - How do we know what to do upon "return"?
- Across programs and operating systems:
  - Multiple user programs
  - Operating system has to orchestrate them all
    - Each gets a share of computing cycles
    - They may need to share system resources (memory, I/O, disks)
  - Yielding and taking control of the processor
    - Voluntary or "by force"?

Big Theme 2: Design Values

- Design choices are a combination of goals and context
  - Based on history and the society of the times
    - Usually assumptions about normativity or "common case"
  - Imbued with the values of the creators
    - Think critically about what you are told & sold!
- Nothing is future-proof
  - The House of Computing needs remodeling!
    - Built on the values of efficiency, profit, and scaling
  - Need to reexamine your heading and vision periodically
    - Check your metrics and definition of success



Thinking Critically About Ideologies: Efficiency

- C is close to the hardware: lack of abstraction makes it **fast** but also **difficult to use**
  - Buffer overflows, segmentation faults, memory bugs, etc.
  - "Minimalistic" and "rugged" or "bloated" and "spoiled"
    - We need people in systems with both perspectives!
- Automation of "boring, repetitive work"
  - Augmentation is highly valued and exclusive
  - "Boring, repetitive work" is more available but tenuous; want to be rid of it!

Thinking Critically About Ideologies: Scaling

- Large tech companies operate on a global scale.
- But scaling is really, really difficult…
  …and when you do it wrong, there are global consequences.
- Co-design required: when designing a solution for the world, need to **involve** lots of people all over the world
  - What does "organizing the world" (Google) mean? Organizing it for whom? To what end?
- Computers make it technically easier to scale globally, but that doesn't make us otherwise uniquely suited
  - "Move fast and break things" doesn't scale well...

Course Perspective

- CSE351 will make you a more informed programmer
  - Purpose is to show how software really works
  - Understanding the underlying system makes you more effective:
    - Better debugging
    - Better basis for evaluating performance
    - How multiple activities work in concert (*e.g.*, OS and user programs)
  - Not just a course for hardware enthusiasts!
    - What **every** CSE major needs to know (plus many more details)
    - See many **patterns** that come up over and over in computing (like caching)
- CSE351 presents a world-view that will empower you
  - The intellectual and software tools to understand the trillions+ of 1s and 0s that are "flying around" when your program runs

Course Perspective

- CSE 351 provides you the tools you need to continue your journey into computer science
- You may have learned that you love low-level stuff!
  Here is what to take next:
  - Systems Programming (333), Operating Systems (451), Compilers (401), Embedded Systems (474)
  - Pick up an Arduino, and hack away on your own projects!
- You might never want to touch C again…but maybe something else struck your fancy!
  - Programming Languages (341), Distributed Systems (452), Machine Learning (446)
  - Create a website! Pick up a Raspberry Pi, hook up some LEDs, program it in Python!

Course Perspective

- Wherever your next steps take you…
  - You belong in computer science **because** of who you are, not despite it. **Your** perspective is unique and valuable.
  - The areas of computer science that you like to spend time in don't make you any "less" of a programmer.
  - You get to decide what works best for you.

- But also, take some time away from the computer 😁
  - The world is huge, and your laptop's screen is only, like, 14"
  - Take classes outside of CSE – you might even make connections!

Can You Now Explain These to a Friend?

- Which of the following did you actually find the most interesting to learn about?

a) What is a GFLOP and why is it used in computer benchmarks?
b) How and why does running many programs for a long time eat into your memory (RAM)?
c) What is stack overflow and how does it happen?
d) Why does your computer slow down when you run out of *disk* space?
e) What was the flaw behind the original Internet worm and the Heartbleed bug?
f) What is the meaning behind the different CPU specifications? (*e.g.*, # of cores, # and size of cache, supported memory types)

Courses: What's Next?

- Staying near the hardware/software interface:
  - **CSE 369/EE 271:** Digital Design – basic hardware design using FPGAs
  - **CSE 474/EE 474:** Embedded Systems – software design for microcontrollers
- Systems software:
  - **CSE 341/CSE 413:** Programming Languages
  - **CSE 332/CSE 373:** Data Structures and Parallelism
  - **CSE 333/CSE 374:** Systems Programming – building well-structured systems in C/C++
- Looking ahead:
  - **CSE 401:** Compilers (pre-reqs: 332)
  - **CSE 451:** Operating Systems (pre-reqs: 332, 333)
  - **CSE 461:** Networks (pre-reqs: 332, 333)
  - **CSE 484:** Computer Security (pre-reqs: 332, 351)
  - **CSE 590E**: Computer Science Education Seminar

Courses: What's Next?

- Something completely different!
- From past & present TAs
  - MUSIC 116: Elementary Music Theory
  - GWSS 374: Transgender Studies
  - PHIL 118: Persuasion Or Manipulation? The Ethics And Psychology Of Influence
  - LING 269: Swearing and Taboo Language
  - DIS ST 230: Introduction to Disability Studies
  - ESRM 190: Environmental Science Special Topics
  - LING 200: Intro to Linguistics
  - DESIGN 150: What Is Design: Practices, Principles, And Perspectives
  - EDUC 251: Seeking Educational Equity and Diversity
  - MUSIC 162: American Popular Song

Thanks for a great quarter!

- Huge thanks to your awesome TAs!

- Don’t be a stranger!
  - Next year my teaching schedule is Au: CSE 121, Wi: CSE 122, Sp: CSE 391
  - I’m always looking for TAs, so be sure to apply for any of these courses!


Adithi Raghavan
(she/her)
adithira@uw
Section BD/DD


Aman Mohammed
(he/him)
amanm4@cs
Section BD/DD, BE


Brenden Page
(he/him)
bpage1@cs
Section AC/CC


Celestine Buendia
(they/them)
cbuendia@uw
Section BA/DA


Chloe Fong
(she/her)
cf2024@uw
Section AD/CD


Claire Wang
(she/her)
wangclai@cs
Section AB/CB


Ellis Haker
(he/him)
ehaker1@uw
Section BA/DA


Hamsa Shankar
(she/her)
hamsas@cs
Section AE/CE


Maggie Jiang
(she/her)
mjiang@cs
Section AE/CE


Malak Zaki
(she/her)
mzaki@uw
Section AC/CC


Naama Amiel
(she/her)
namiel@uw
Section BC/DC


Nikolas McNamee
(he/him)
mcnanik@uw
Section AB/CB


Shananda Dokka
(she/her)
sdokka@uw
Section AD/CD


Stephen Ying
(he/him)
glying@cs
Section BB/DB


Will Robertson
(he/him)
willruw@cs
Section BB/DB, BE

Ask Me Anything