Looking back and looking ahead

Ryan Eberhardt
August 23, 2021

Looking back: Principles of systems design

- Abstraction
  - How do we separate behavior from implementation?
  - Processes, threads, filesystems, terminals… cars, door locks, light bulbs
- Modularity and Layering
  - How do we separate complexity into separate modules and layers, where each layer builds on the layers beneath it?
  - Filesystems, compilers, networks, MapReduce
- Client-server request-and-response
  - When we have functionality organized into different pieces, how do we execute that functionality?
  - Ordinary function calls, system calls, HTTP requests, DNS requests, etc
- Naming and Name Resolution
  - Sometimes it’s helpful to introduce different ways of talking about some resource
  - File names, domain names: make it easier for humans to talk about
  - Virtual memory addresses, file descriptors, PID namespaces: provide extra security properties

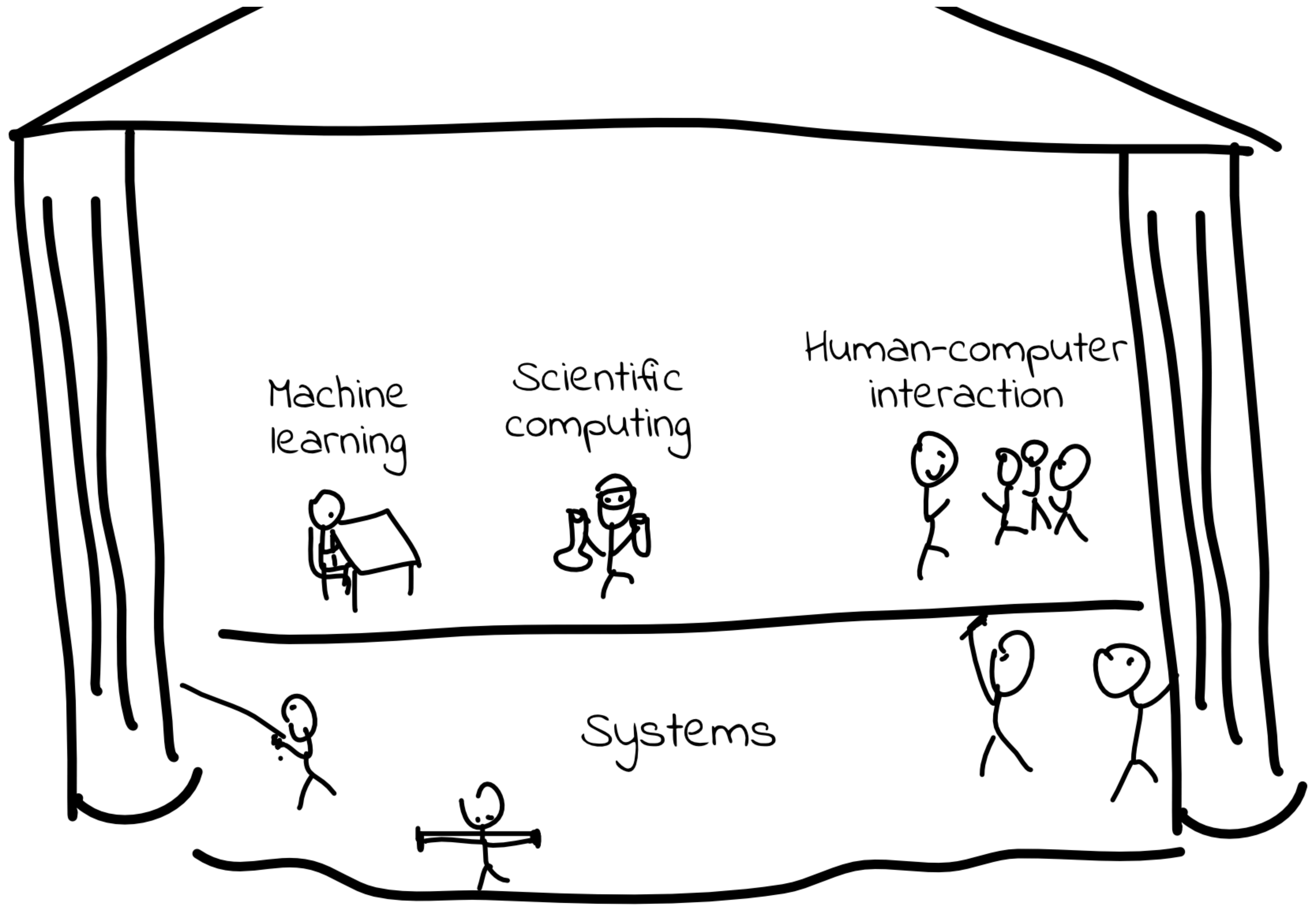
Looking back: Principles of systems design

- Caching
  - How do we efficiently keep data around that may be needed later?
  - CPU L1, L2, L3 caches; web browser, proxy caches; DNS lookup caches
- Virtualization
  - Making many resources look like one: e.g. AFS, load balancers
  - Making one resource look like many: e.g. virtual memory, threads, virtual machines
- Concurrency
  - How do we make it possible to run multiple things at the same time?

Looking ahead

What is systems?

- "Computer systems" is so broad and vague that it seems it could refer to anything involving computers
- I view systems as being about building the platform that all application software stands on



What is systems?

- “Computer systems” is so broad and vague that it seems it could refer to anything involving computers
- I view systems as being about building the platform that all application software stands on
- Systems people are the carpenters, masons, electricians, *and also* the architects and civil engineers
    - You’re designing a solution around high-level goals and tradeoffs
    - You’re also getting your hands dirty building it, fixing problems, etc

What is systems?

- Systems is about:
  - Given some building blocks, how do we put them together to achieve some goal?
  - How do we analyze the implications of different ways of solving the same problem?
  - How do we take complicated things and create abstractions that are easy to understand and easy to use?
  - How do we debug a malfunctioning system?
- Systems is *not* about C or C++ or manual memory management or what not
  - These are crucial building blocks in one limited domain, but there are plenty of domains in which they are not so relevant

Computer networking

- How do we facilitate communication between devices?
  - Want fast, efficient, inexpensive communication
- Also has many subfields:
  - Hardware design (routers, low-energy mesh networks, radios, signals processing, etc.)
  - Software (many layers involving implementation of various protocols)
  - There are even people that specialize in protocol design!
- Classes: CS 144, 244, 344
- Possibly also EE 179, EE 382C, EE 384C, EE 384S, EE 384X if you’re more interested in hardware

Databases and information systems

- How do we store and analyze data efficiently?
- Many different database designs depending on whether you are storing structured vs unstructured data, want batch processing vs streaming processing, etc
- Interesting reads related to my favorite general-purpose database:
    - https://www.theguardian.com/info/2018/nov/30/bye-bye-mongo-hello-postgres
    - https://pganalyze.com/blog/postgres-14-performance-monitoring
- Classes: CS 145, 245, 246

Distributed systems

- How do you distribute computation and/or the storage of information over many (as few as 2, as many as 100k+) different computers?
- How can we detect and handle unreliability? Computers may crash, networks may go down, data can be silently corrupted
- Need to build redundancy into the system, and build logic to detect / switch over when something has gone wrong
- Need to design *consensus protocols* so that different computers can come to an agreement when different servers are doing different things
- Classes: CS 244, 244B

High performance computing

- How do we take some hardware with certain characteristics and do math really, really fast?
- Classes: CS 149, ME 344?

Operating systems

- How do we share resources across programs?
- How do we provide properties such as fairness and isolation?
- How do we design basic building blocks that are useful, usable, and secure?
- I used to see operating systems as a mature, static area with not a lot of growth
- But many new OSs are emerging, driven by new pressures / incentives / perspectives
  - Mobile: Android, iOS, ChromeOS, Fuchsia
  - Embedded systems: Tock, others
- The “old” OSs are changing too!
  - io_uring: a better way to do async I/O (introduced late 2019)
    - https://lwn.net/Articles/810414/
    - https://despairlabs.com/posts/2021-06-16-io-uring-is-not-an-event-system/
  - eBPF: run user code *inside* the kernel. incredible for debugging, logging, performance analysis
    - https://filipnikolovski.com/posts/ebpf/
    - https://netflixtechblog.com/how-netflix-uses-ebpf-flow-logs-at-scale-for-network-insight-e3ea997dca96

- Classes: CS 140(E), 240

Compilers, programming languages, program analysis

- How do we help people express their intent as code? (How do we help them say “this is what I want to do” and translate that into code, and make that as easy as possible for humans?)
- How do we provide good error messages, good safety properties, and help people find bugs?
- Program analysis: can we find ways to predict what a program will do? can we alert someone if it does something bad?
    - clang-tidy, ASan, TSan are all different kinds of program analyzers, developed relatively recently!
    - New program analyzers: Ethereum is getting popular, and Ethereum smart contracts are programs too! Can you automatically identify bugs in smart contracts that would allow someone to “steal” millions of dollars?
- Classes: CS 144, 242, 244

Applications

- How do we design and build and reason about a complicated system involving 10k-10M lines of code written by countless different people, many libraries, and code running on different machines all talking to each other?
- What abstractions and mental models make all this complexity tractable to reason about?
- Systems doesn’t have to be about living in some super specific technical niche! Modern web applications may be all higher-level programming, but involve just as much complexity (if not more)
- Classes: CS 142, 190

Security

- Given all these areas… how do you secure all this stuff?
- How can you anticipate how a malicious person might take well-intentioned code and use it against its intended purpose?
- Classes: CS 155, 356, 255

General advice

- Try lots of things out!
- Don’t be a “systems” person. Systems is inherently applied. Take a wide breadth of classes. See what domains you like working in the most
- Don’t stress about picking one area to focus in for the next 20 years of your life. Work in areas you find interesting, and you can always move to other areas in the future
  - The most interesting problems of 2040 most likely don’t exist yet!
- Read engineering blogs
- Get your hands dirty. Talk to professors, see if you can do research or an internship