CS380L:
Advanced Operating Systems

Emmett Witchel

Agenda for Today

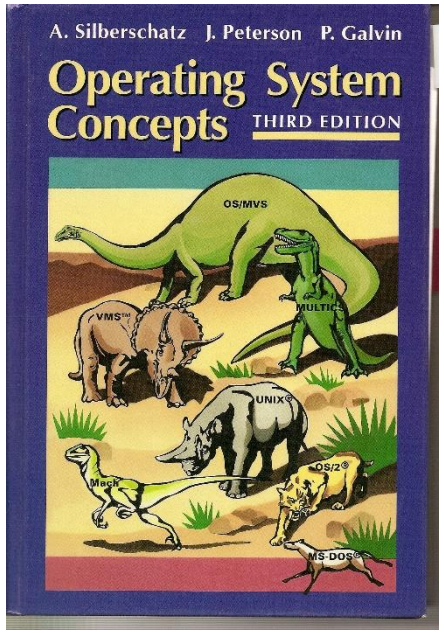
- Some important words…
- Intro/Overview
  - What is OS/systems research
  - Why care about OS/systems research?
  - What are the fundamental problems?
- Administrivia
  - Mechanical stuff
  - Course structure/Goals
  - ***There is a quiz coming up!***
- Today's Readings
  - Yes, there were readings! Did you read them? ☺
  - How [not] to write a good research paper, On being the Right size
- Questions (please ask throughout)

A climate conducive to learning and creating knowledge is the right of every person in our community. Bias, harassment and discrimination of any sort have no place here. If you notice an incident that causes concern, please contact the Campus Climate Response Team: **diversity.utexas.edu/ccrt**

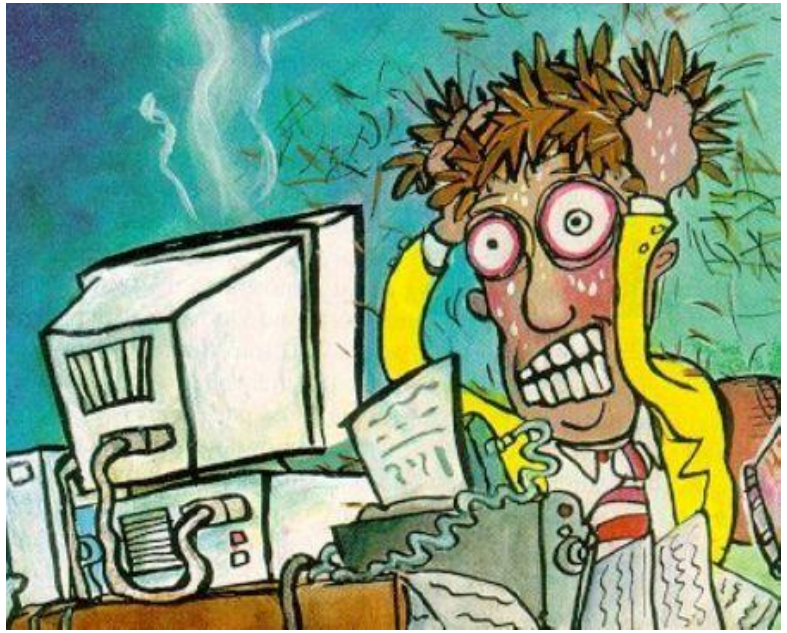
The University of Texas at Austin
College of Natural Sciences

The College of Natural Sciences is steadfastly committed to enriching and transformative educational and research experiences for every member of our community. Find more resources to support a diverse, equitable and welcoming community within Texas Science and share your experiences at ***cns.utexas.edu/diversity***

The public image of OS research




```
[3448015.307991]  [<ffffffffa0145c3b>] ? :ext3:ext3_ordered_write_end+0x73/0x110
[3448015.307991]  [<ffffffff80265486>] ? generic_file_buffered_write+0x1c0/0x63c
[3448015.307991]  [<ffffffff80231409>] ? current_fs_time+0x1e/0x24
[3448015.307991]  [<ffffffff80265c41>] ? __generic_file_aio_write_nolock+0x33f/0
x3a9
[3448015.307991]  [<ffffffff802419a1>] ? hrtimer_wakeup+0x0/0x22
[3448015.307991]  [<ffffffff80265d0c>] ? generic_file_aio_write+0x61/0xc1
[3448015.307991]  [<ffffffffa01422fe>] ? :ext3:ext3_file_write+0x16/0x94
[3448015.307991]  [<ffffffff8028a12f>] ? do_sync_write+0xc9/0x10c
[3448015.307991]  [<ffffffff8023f699>] ? autoremove_wake_function+0x0/0x2e
[3448015.307991]  [<ffffffff80242079>] ? ktime_get_ts+0x22/0x4b
[3448015.307991]  [<ffffffff8028a8d9>] ? vfs_write+0xad/0x156
[3448015.307991]  [<ffffffff8028af64>] ? sys_pwrite64+0x50/0x70
[3448015.307991]  [<ffffffff8020b528>] ? system_call+0x68/0x6d
[3448015.307991]  [<ffffffff8020b4c0>] ? system_call+0x0/0x6d
[3448015.307991]
[3448015.307991]
[3448015.307991] Code: 30 fa 58 80 4c 39 2c 08 75 04 0f 0b eb fe 48 c7 c0 40 fa
58 80 eb 1f 65 48 8b 04 25 10 00 00 00 66 f7 80 44 e0 ff ff 00 ff 75 04 <0f> 0b
eb fe 48 c7 c0 30 fa 58 80 48 8d 1c 08 48 83 3b 00 74 04
[3448015.307991] RIP  [<ffffffff8037fc7c>] xen_spin_wait+0x90/0x139
[3448015.307991]  RSP <ffffffff80595e28>
[3448015.307991] ---[ end trace 604fbc4ae1a5e660 ]---
[3448015.308075] Kernel panic - not syncing: Aiee, killing interrupt handler!
```



OS research *actually*

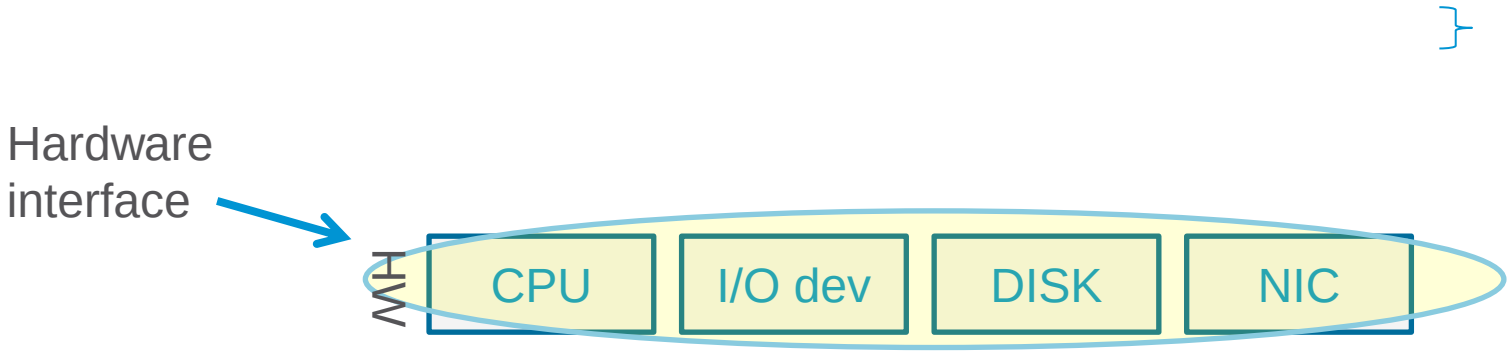
- Not really *just* about building Oses
- Any large code base converges on becoming an OS
  - Manages memory, programming for space/performance, hardware details
    - database, JVM, browser, parallel/distributed systems, internet services
- How to structure systems and deal with complexity
  - Modularize and encapsulate
  - Choose interfaces/abstractions carefully
- Themes in this class
  - Abstractions for managing/accessing resources: storage, replication, naming
  - Correctness: concurrency and sharing
  - Guarantees in *real* systems: security, fault tolerance, etc.

How to “structure” a system

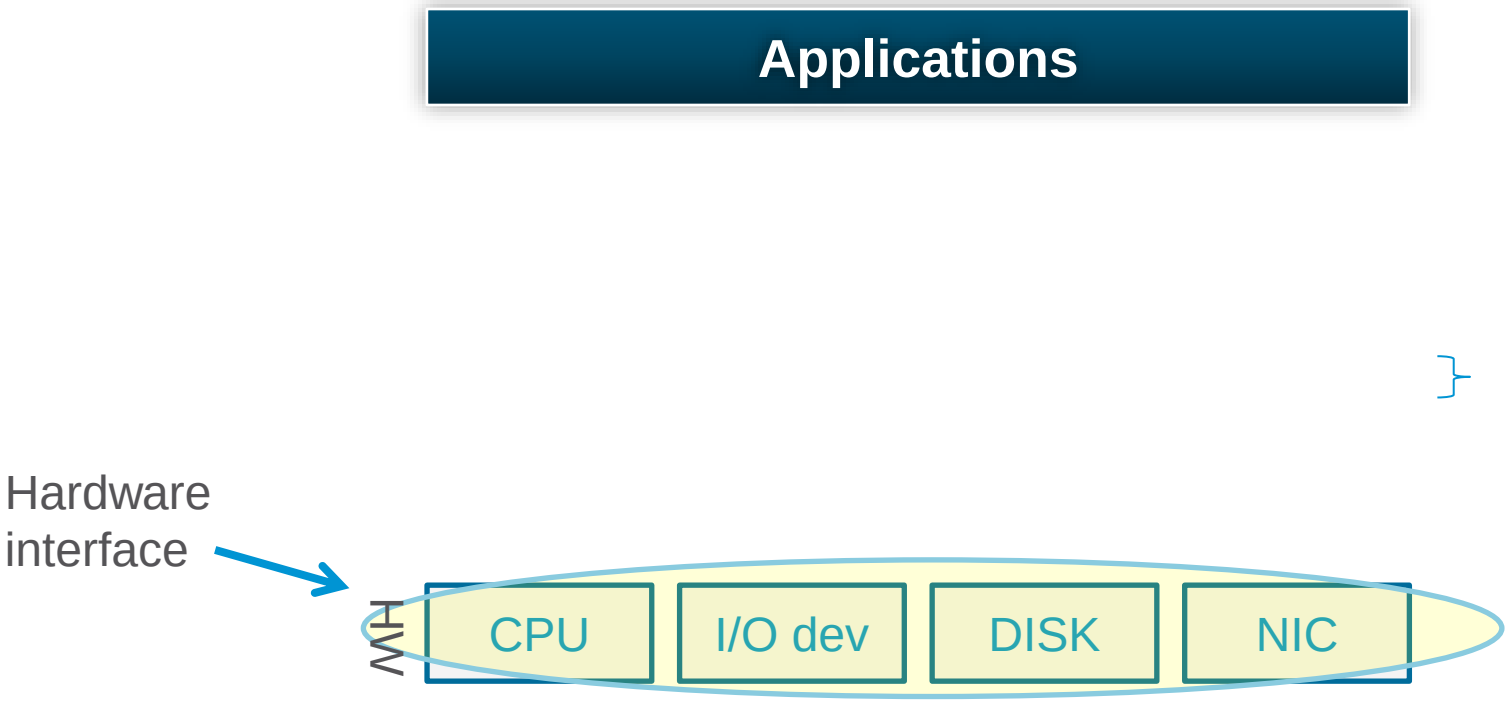
}

How to “structure” a system

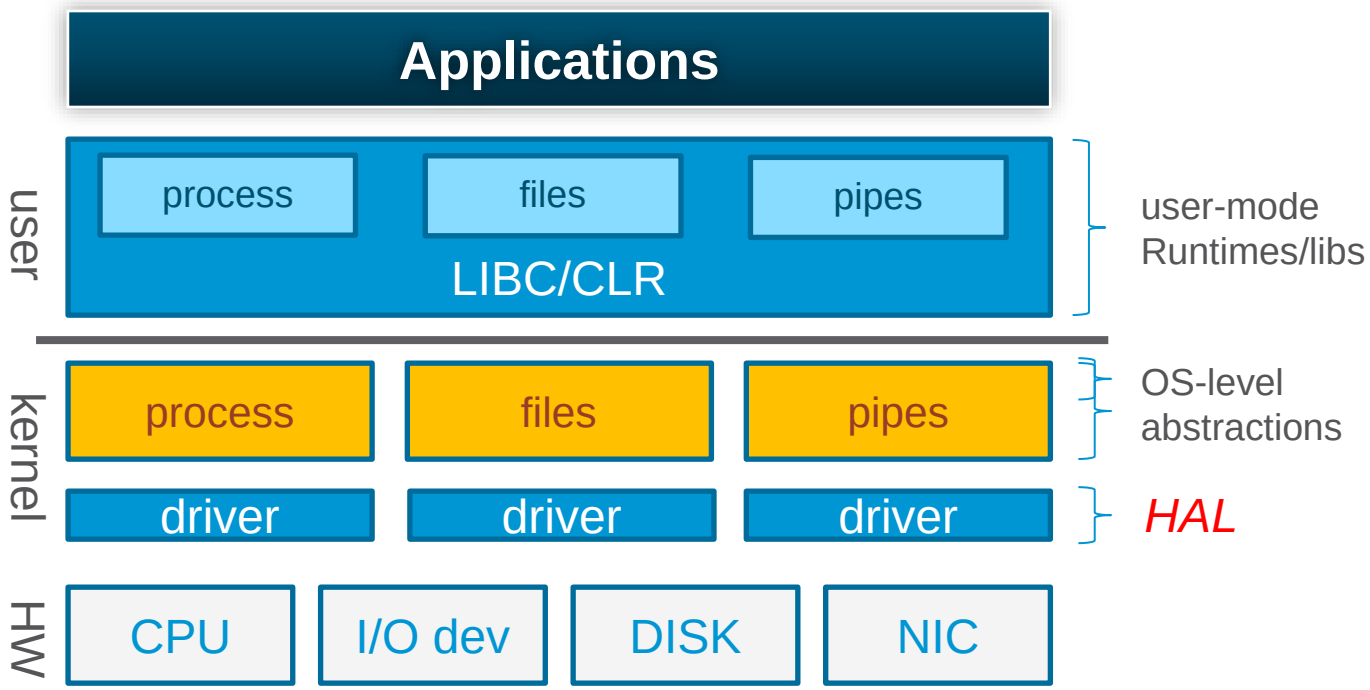
How to “structure” a system



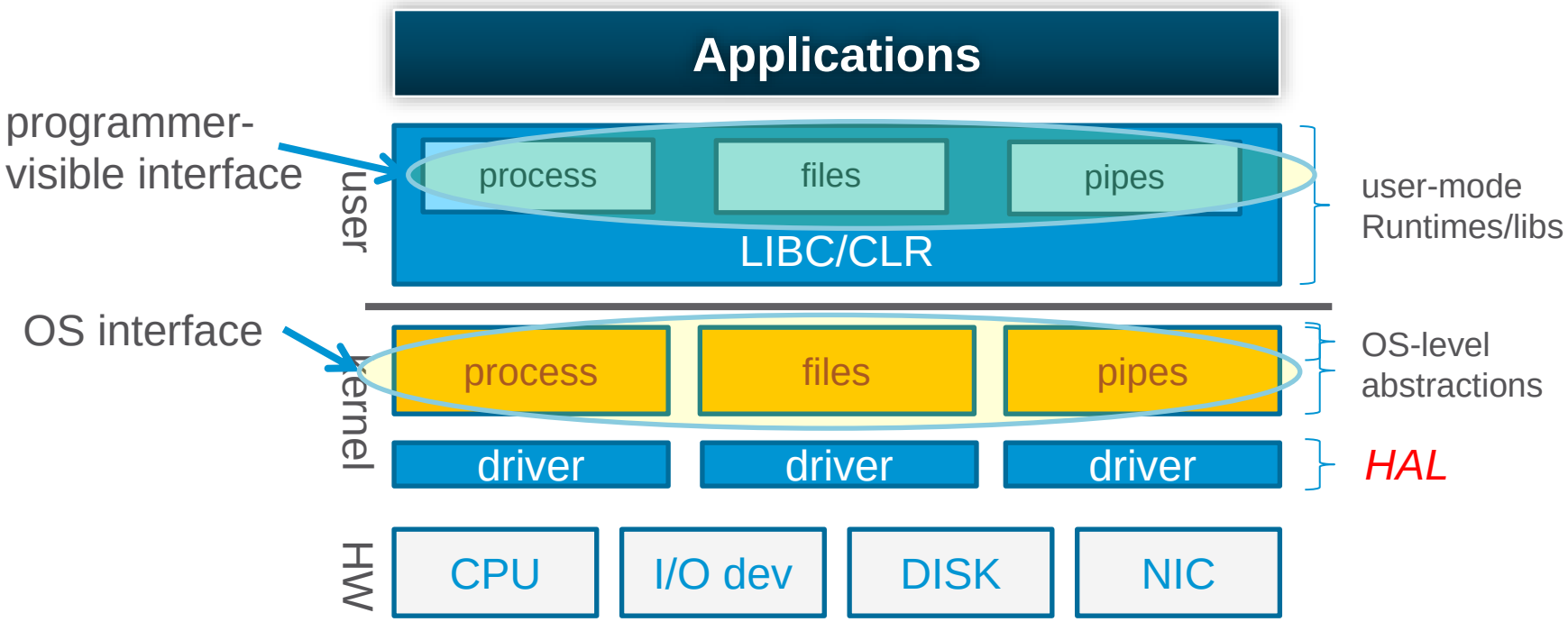
How to “structure” a system



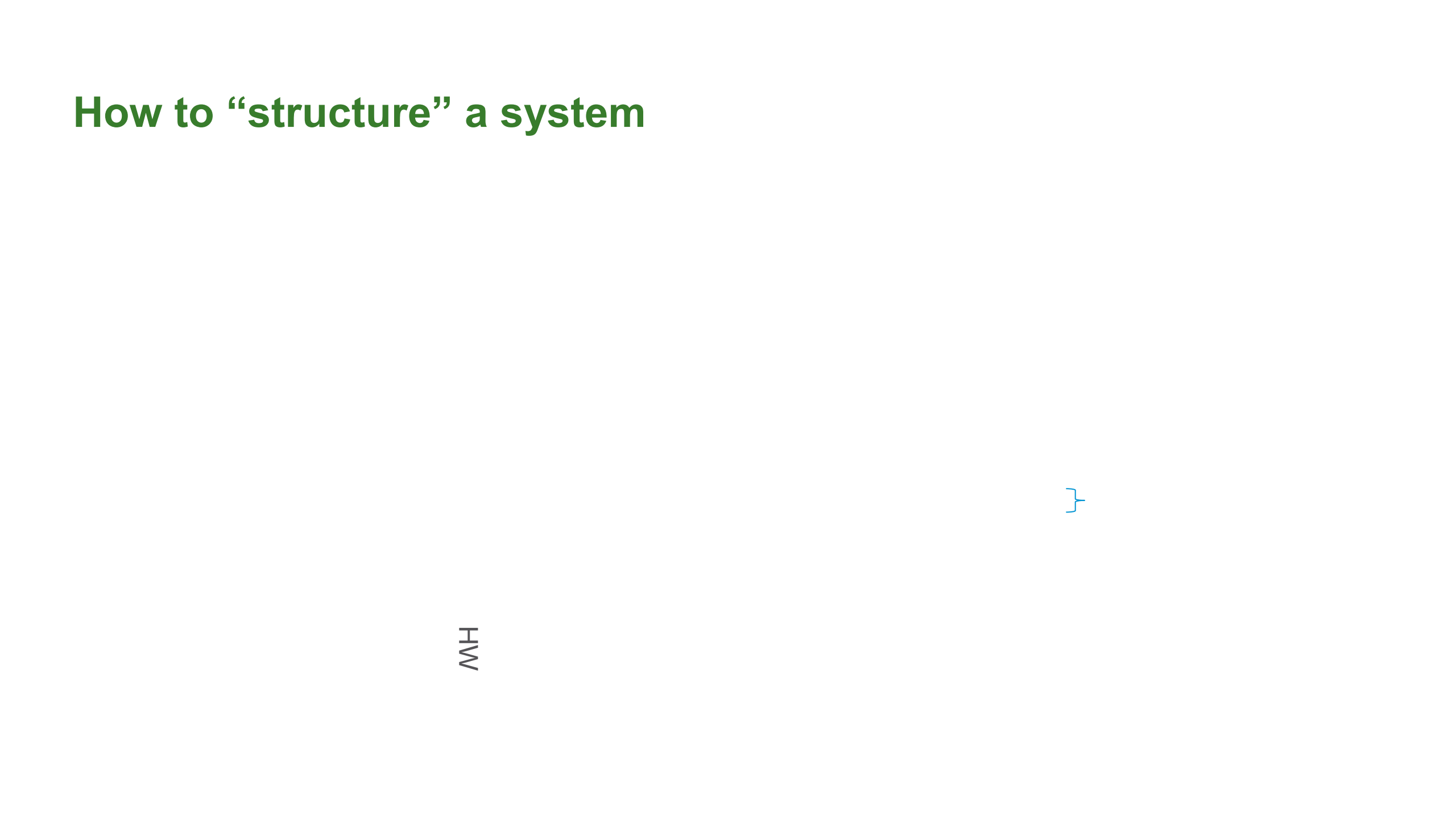
How to “structure” a system



How to “structure” a system



How to “structure” a system

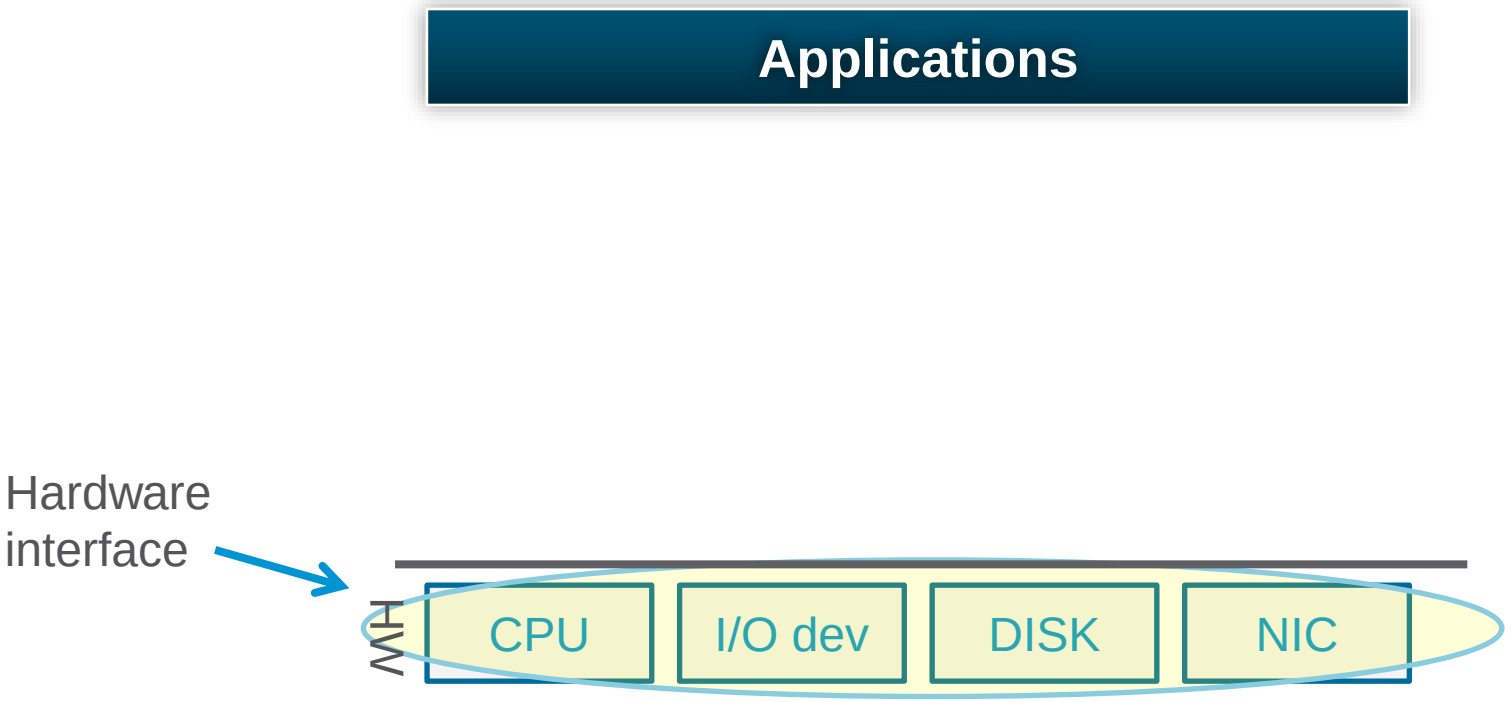


How to “structure” a system

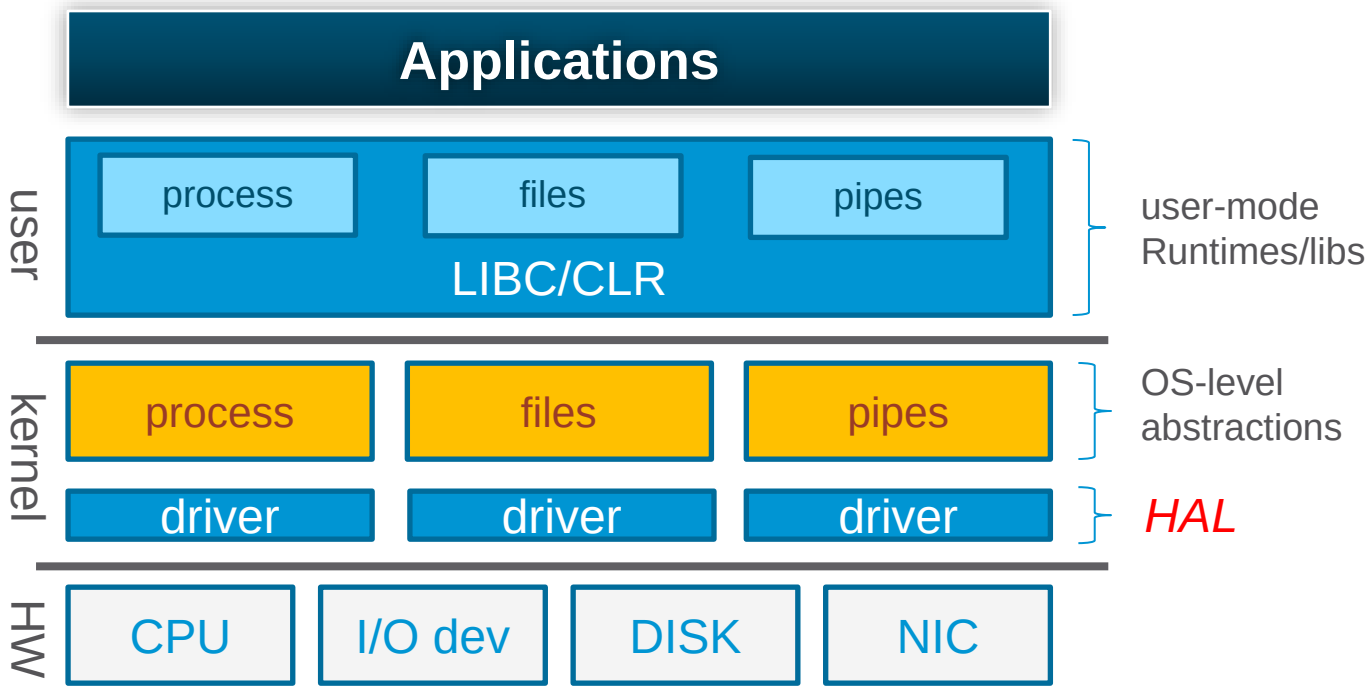
How to “structure” a system

How to “structure” a system

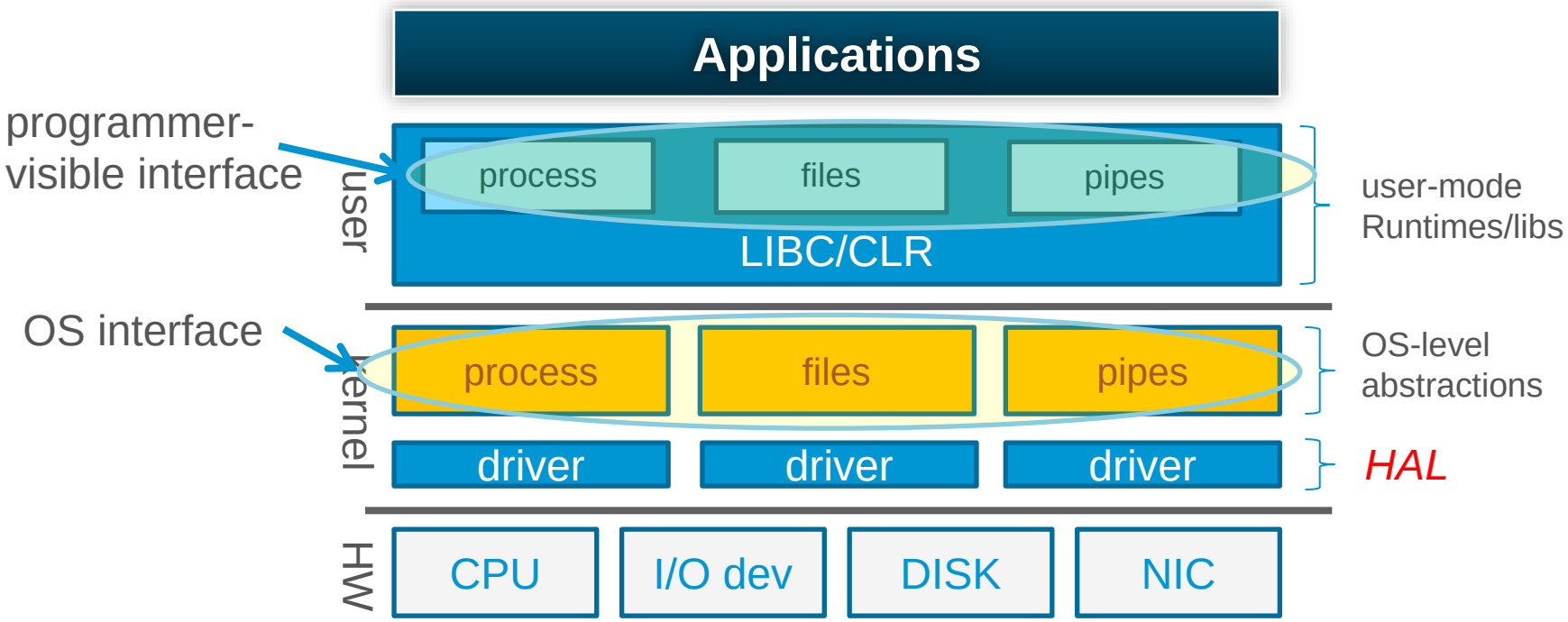
How to “structure” a system



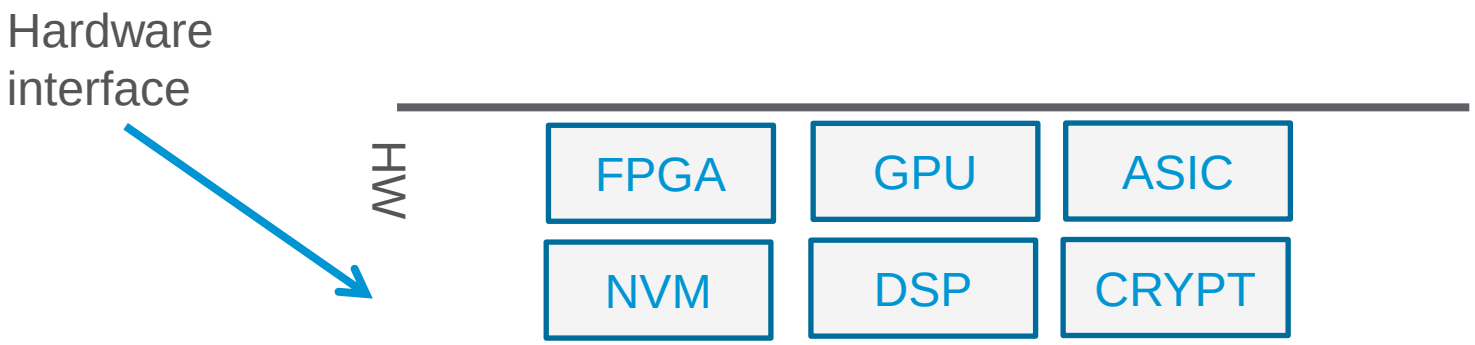
How to “structure” a system



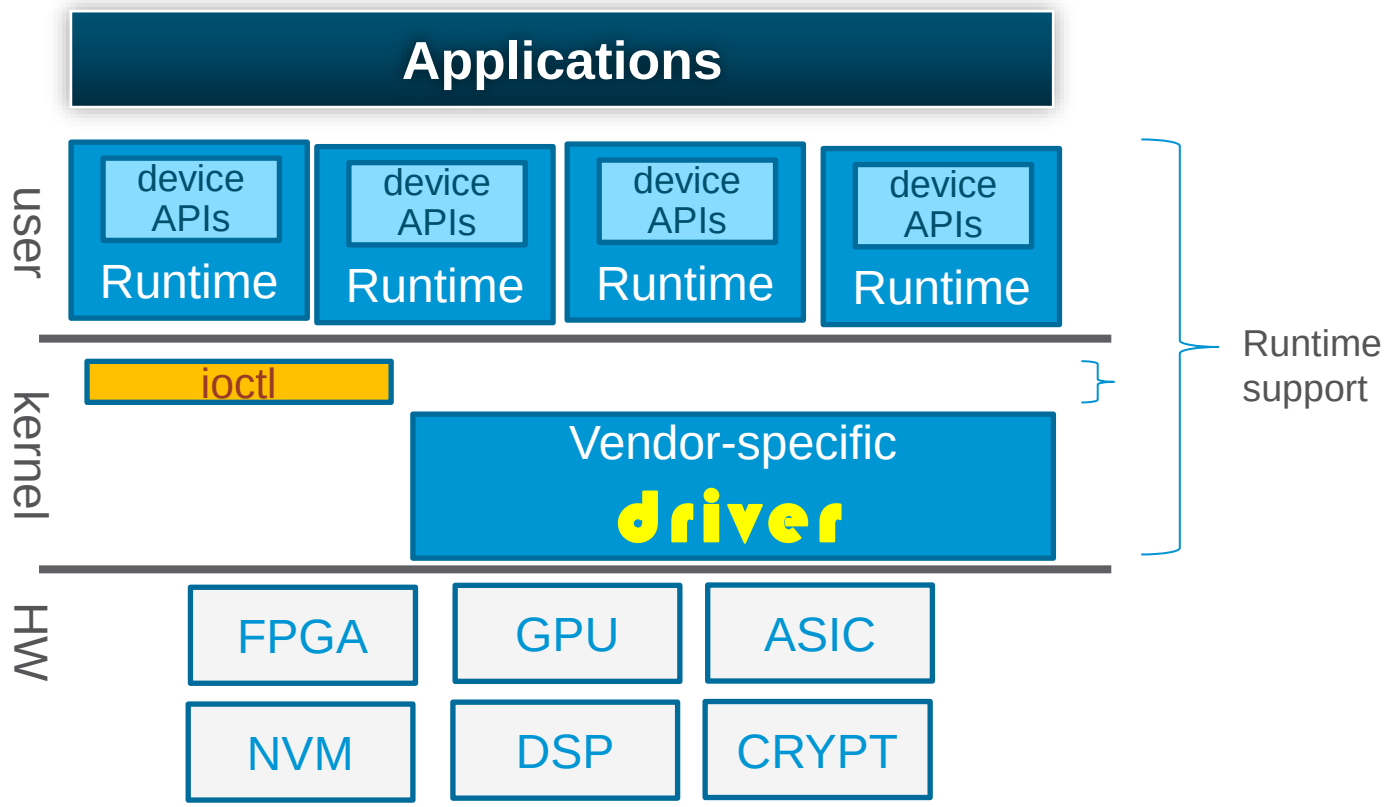
How to “structure” a system



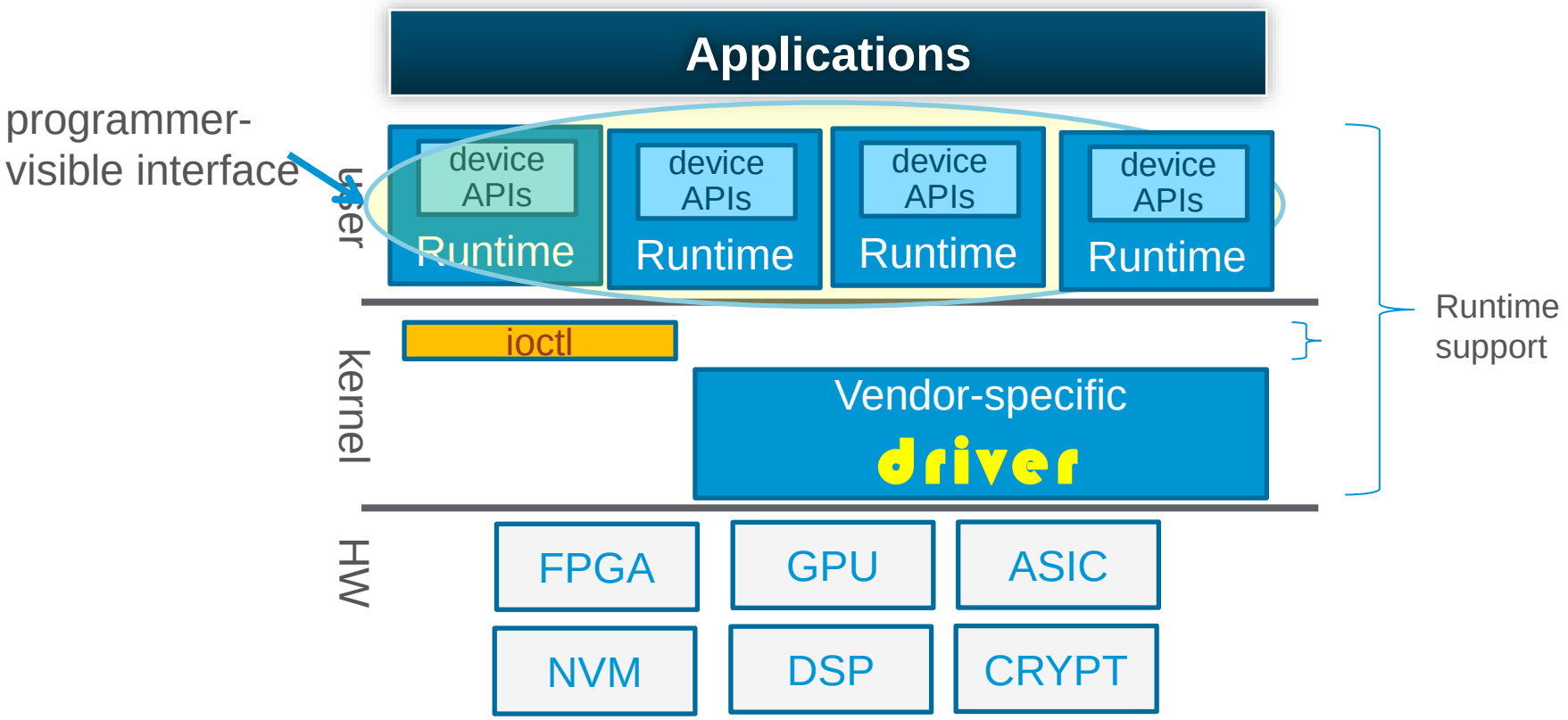
How to “structure” a system



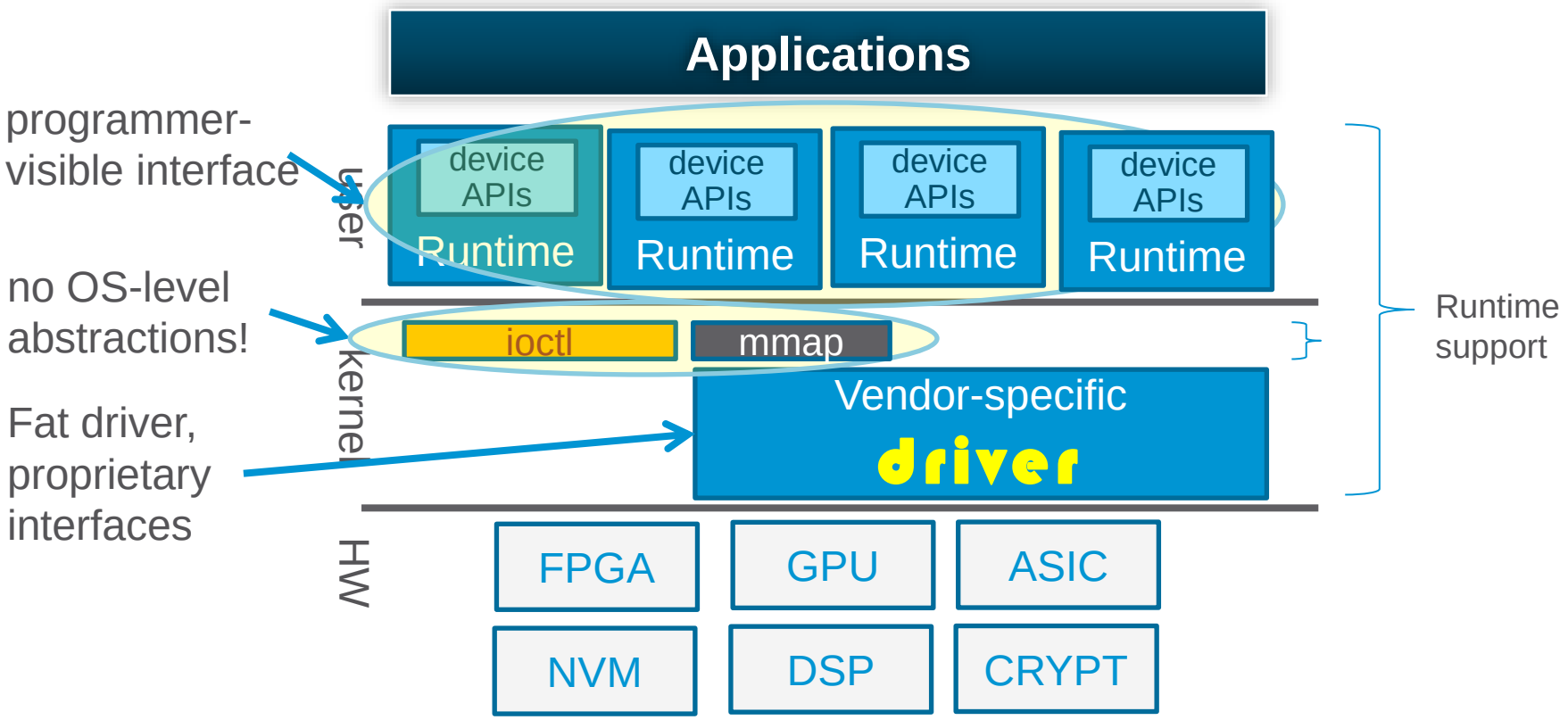
How to “structure” a system



How to “structure” a system

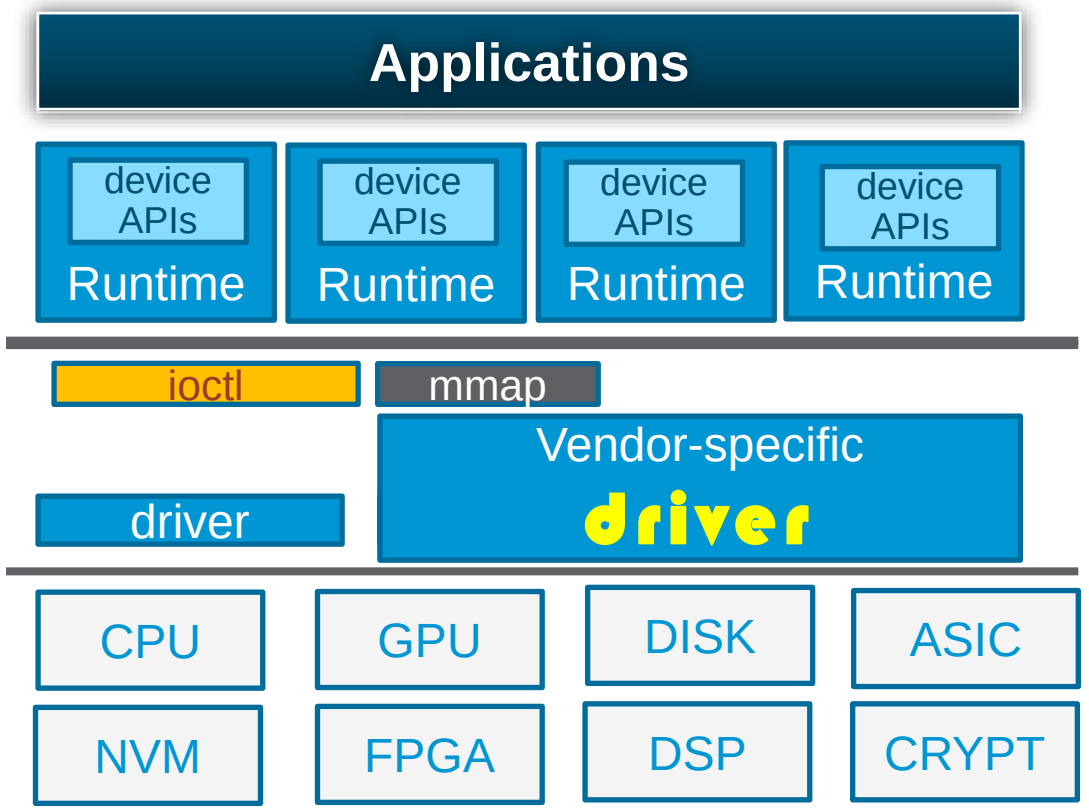


How to “structure” a system



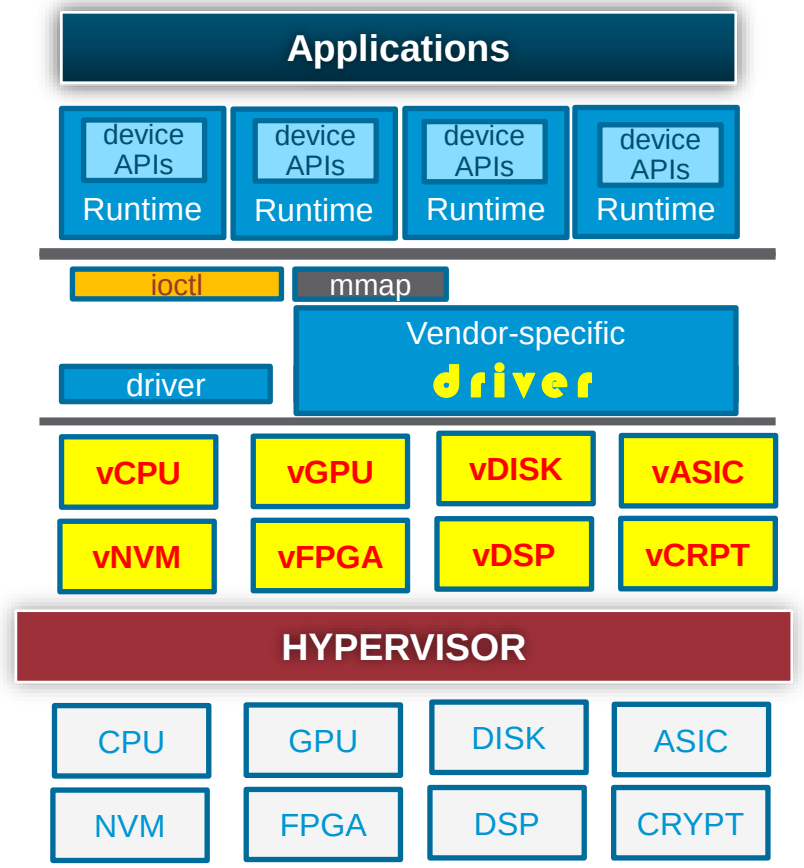
System Software is Living in the Past

- OSes designed for old hardware
- Old OS assumptions: CPUs, disks, networks, OS is the only resource manager



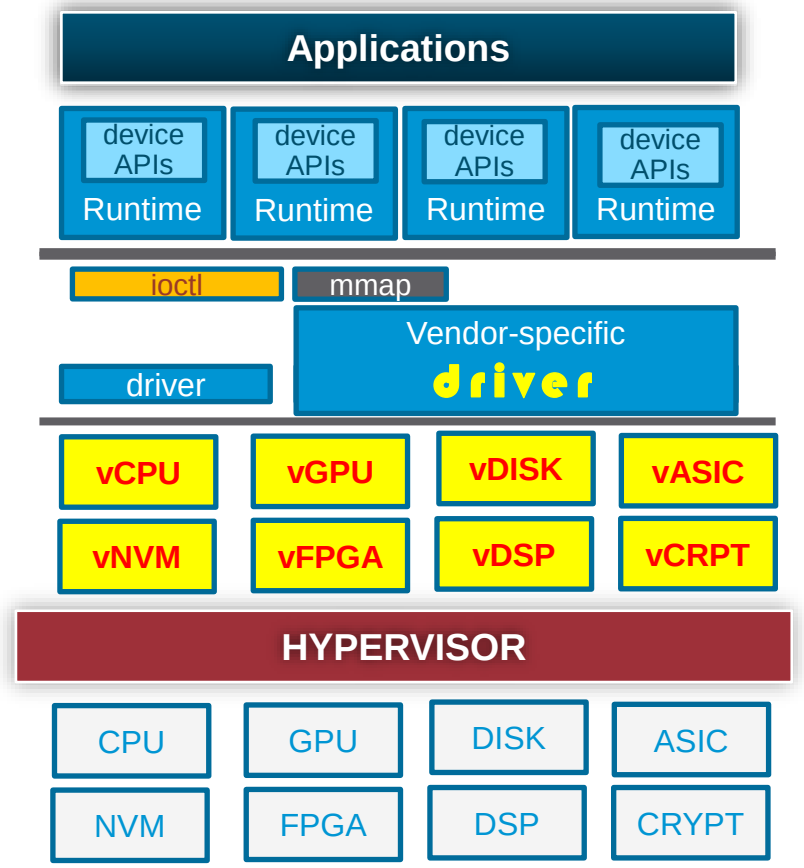
System Software is Living in the Past

- OSes designed for old hardware
- Old OS assumptions: CPUs, disks, networks, OS is the only resource manager



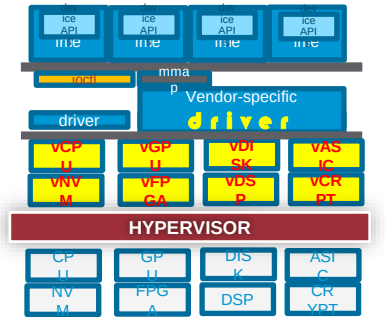
System Software is Living in the Past

- OSes designed for old hardware
- Old OS assumptions: CPUs, disks, networks, OS is the only resource manager



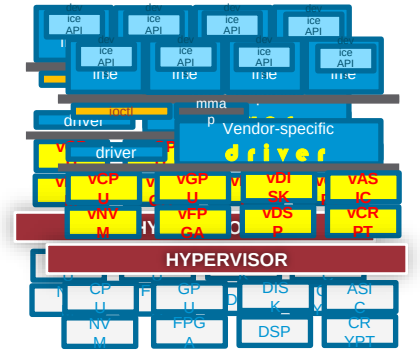
System Software is Living in the Past

- OSes designed for old hardware
- Old OS assumptions: CPUs, disks, networks, OS is the only resource manager



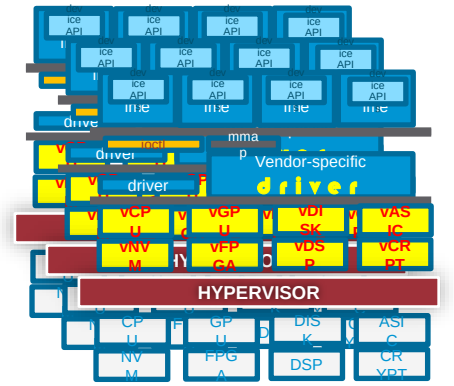
System Software is Living in the Past

- OSes designed for old hardware
- Old OS assumptions: CPUs, disks, networks, OS is the only resource manager



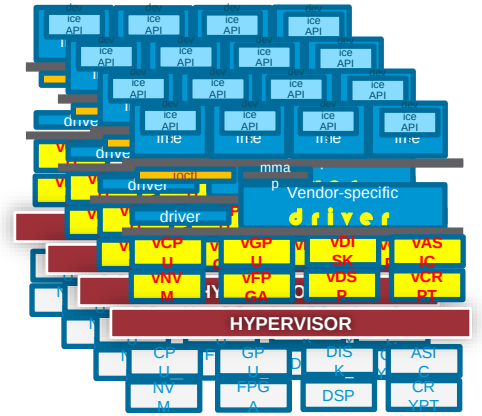
System Software is Living in the Past

- OSes designed for old hardware
- Old OS assumptions: CPUs, disks, networks, OS is the only resource manager



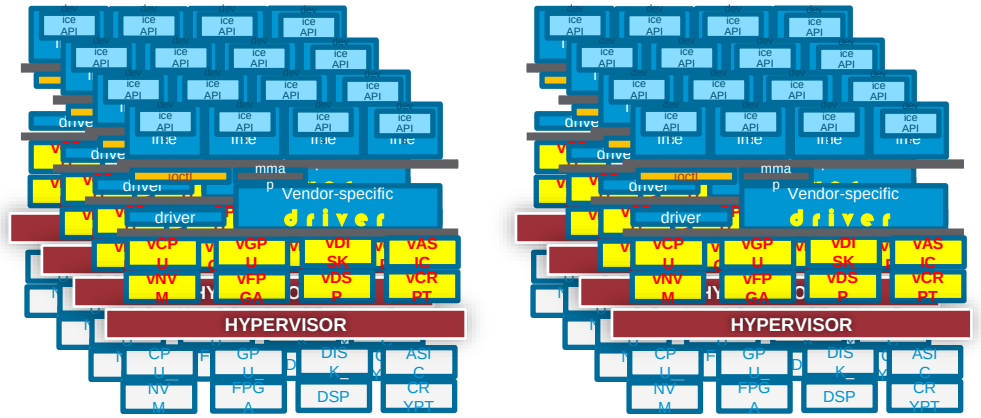
System Software is Living in the Past

- OSes designed for old hardware
- Old OS assumptions: CPUs, disks, networks, OS is the only resource manager



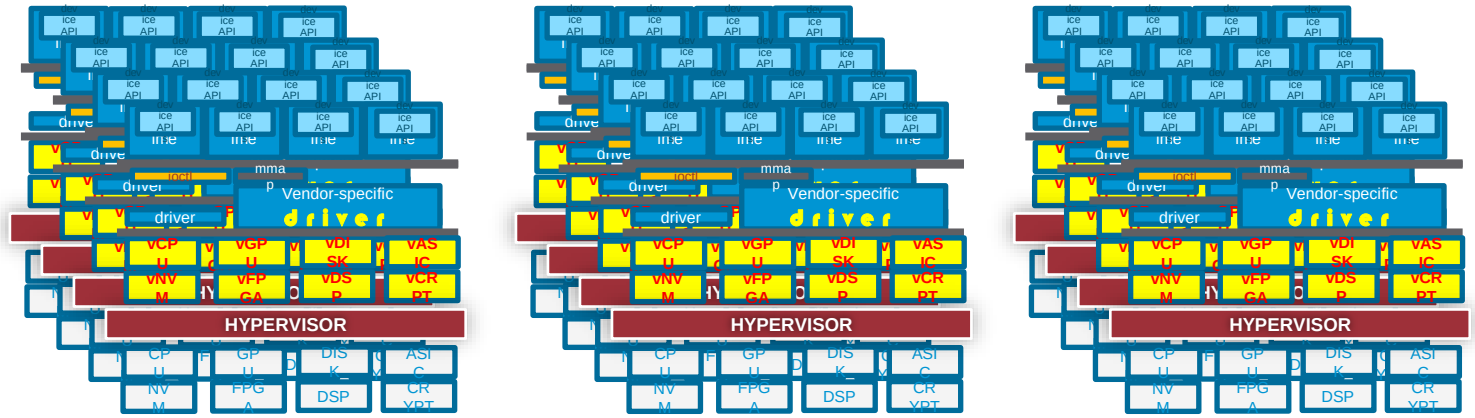
System Software is Living in the Past

- OSes designed for old hardware
- Old OS assumptions: CPUs, disks, networks, OS is the only resource manager



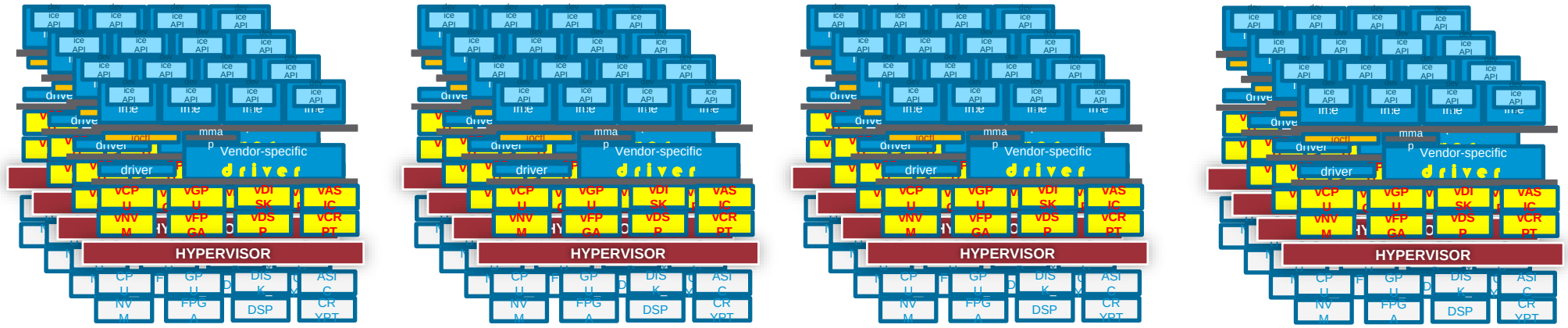
System Software is Living in the Past

- OSes designed for old hardware
- Old OS assumptions: CPUs, disks, networks, OS is the only resource manager



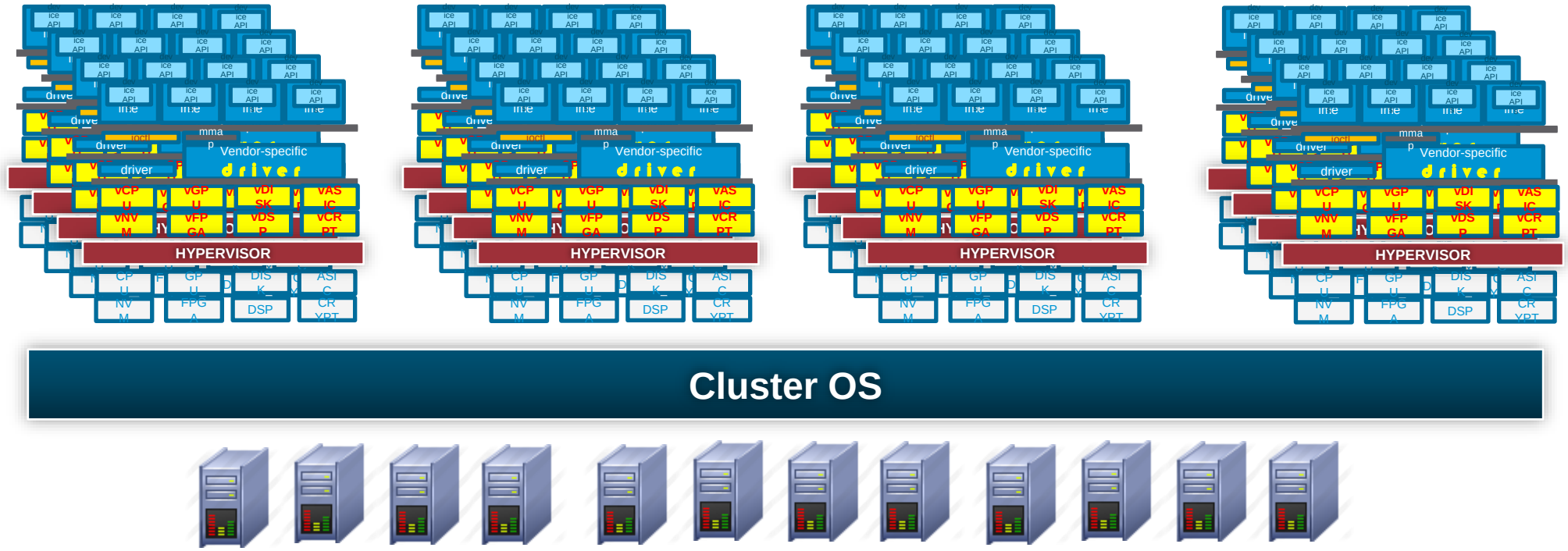
System Software is Living in the Past

- OSes designed for old hardware
- Old OS assumptions: CPUs, disks, networks, OS is the only resource manager



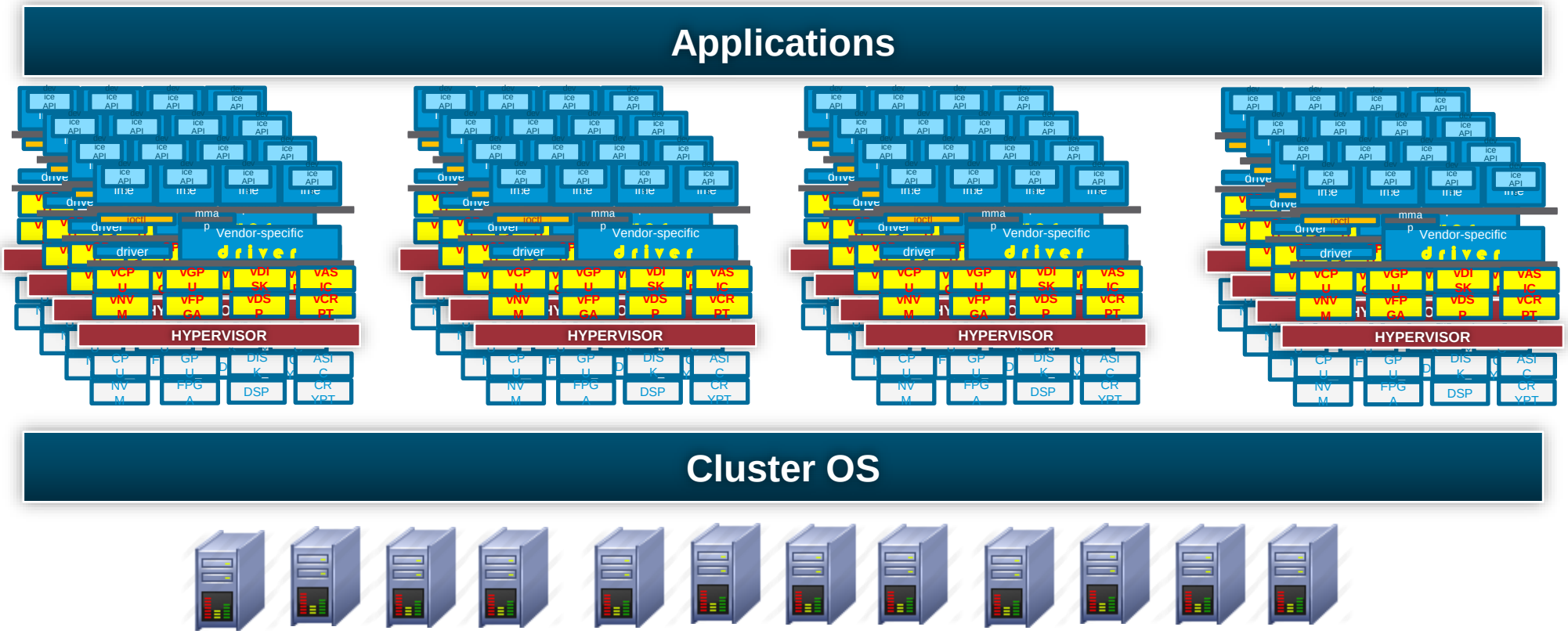
System Software is Living in the Past

- OSes designed for old hardware
- Old OS assumptions: CPUs, disks, networks, OS is the only resource manager



System Software is Living in the Past

- OSes designed for old hardware
- Old OS assumptions: CPUs, disks, networks, OS is the only resource manager



Modern Stacks: choose an interface and virtualize

- **Cool modern interfaces at the top**
- **Many *many* layers**
- **Each component or layer may compose and deal with**
  - Distribution (challenges: scheduling, fault tolerance, orchestration)
  - Heterogeneity (multiple programming models)
  - Concurrency (synchronization, consistency)
- **How do we structure systems with these properties?**
- **How do we know they are correct/safe/etc?**
- **We’re going to talk about a lot of systems but only a subset will actually be OSes**

What is virtualization?

- In computing, virtualization … is the act of creating a virtual (rather than actual) version of something at the same abstraction level. – wikipedia
- Virtualization uses software to create an abstraction layer over computer hardware that allows the hardware elements of a single computer—processors, memory, storage and more—to be divided into multiple virtual computers, commonly called virtual machines (VMs). – ibm.com
- Virtualization defines an interface that allows multiple implementations, possibly with a very different technology. For example, virtual machines are implemented with software, physical machines with hardware.

There are many levels. Start with $PATH

- I want to execute mpirun on odachi

```
odachi:~> which mpirun
mpirun is /usr/bin/mpirun
```

- I want to execute mpirun on epee

```
epee:~> which mpirun
mpirun is /opt/amazon/openmpi/bin/mpirun
```

- My PATH variable virtualizes the executable name

```
epee:~> echo $PATH
…:/opt/amazon/openmpi/bin:/usr/bin:…
```

conda: a virtual software environment

- Rather than installing software system-wide, wouldn't it be great if we could install software separately for each research project?
  - Conda allows you to create separate environments containing files, packages, and their dependencies that will not interact with other environments. (docs.conda.io)
- Conda uses environment variables (PATH, etc.) to achieve isolation
  - This is easy to do
  - Does not require root privilege
  - Integrates well with build systems
  - It is easy to break conda's isolation via bugs or malicious code or by design
  - Each program interprets "search paths" differently and system paths and non-conda paths leak through

docker: a container

- Docker provides the ability to package and run an application in a loosely isolated environment called a container. (docs.docker.com)
  - The isolation and security allows you to run many containers simultaneously on a given host.
- OS capabilities enable docker
  - Process ID namespaces (e.g., each container has an init process pid==1)
  - Network communication (e.g., iptables)
  - Control group (cgroups)
  - Layered/union file system (e.g., AuFS)
- Tradeoffs
  - More secure than conda
  - OS struggles to achieve performance isolation
  - Container isolation is inferior to virtual machines

kvm: a virtual machine

- Using KVM, one can run multiple virtual machines running unmodified Linux or Windows images.
  - Each virtual machine has private virtualized hardware: a network card, disk, graphics adapter, etc. (linux-kvm.org)
- x86 processors enable KVM
  - Processor: VT-x: instructions to enter/exit a virtual machine
  - Memory: extended page tables (EPT): page table for hypervisor
  - Devices: VT-d: Memory translation for devices
- Tradeoffs
  - More secure than containers
  - Slower than containers

Course Outline

- Operating System Design and Implementation
- Virtualization
- File Systems
- Concurrency
- Data Centers and Computing at Scale
- Security
- Verification of Large Scale Systems

Now that you're motivated: mechanical/administrivia

- Web site: www.cs.utexas.edu/~witchel/cs380L
  - The ground truth, ***please read it***
  - ***It will change,*** *so please keep up with it*
  - Piazza: for most interactivity, link on website
  - Canvas: for grading only, no link on website, maybe for quiz/exams
  - E-mail: is a great thing, I respond to it.
- Homework 1: http://www.cs.utexas.edu/~witchel/380L/hw/hw1.pdf
- We will have a test in Canvas next week

Course Structure

Readings, programming assignments, project

Goals:

- Good discussions
- Some exposure to kernel hacking
- Do a significant research project
- Familiarity with reasoning about computing systems, design tradeoff spaces
- Preparation to do your own research and development.

Non-goals:

- busy work
- create grade distribution
- cause stress

What do we hope to achieve?

- **Research** is key: research, research, research
- **Readings** → exposure to others' experimental research
  - Most of work we look at will be good.
  - Good ideas. Also, good implementation tricks. There is wisdom in both.
  - Choice of papers: key results/impact in important areas, but seldom perfect
  - Learn the literature (more importantly: critical thinking about the literature)
  - Cross-pollination for neighboring fields
- **Labs** → improve your own research methodology
  - Exposure to some low-level hacking
  - Learn to problem solve across stack layers
- **Project** → research practice, impact beyond the class
  - learn by doing.
  - Rapidly evolving area: a great grad student project can go to top conference (mine did)

Readings

- Reading will take a long time (especially at first).
  - Be active in validating your understanding.
  - Take notes. Draw a picture. Work out a small example.
  - Many are influential papers, but not all are well written.
  - Most are good examples of experimental technique, though almost all could be improved. (deadlines!)
- Were they successful?
  - Not all of them ever *really* worked, all reach EOL at some point
  - Most you can hope for: some ideas live on
- What to look for:
  - Primarily: ideas and approach
  - In most cases, not style
  - In many cases: no right or wrong answer
- Get good at pulling out the useful information.
- Operational understanding fundamentals is key.
  - Enumerate asserted guarantees, ask/understand how they are achieved
  - For example, in a security paper, how does the system provide privacy? integrity? freshness?

Paper Reviews/Critiques

- Suggested form on website: http://www.cs.utexas.edu/~witchel/cs380L/rev/review.html
- Critique != Summary
- Use it to organize your thoughts
  - Perspectives often change during review writing
- What to include:
  - Enumerate the strengths and weaknesses
  - Questions it left you with
  - Suggestions for improvement/extension, better evaluation/understanding
  - Assess impact from modern perspective
  - How does it fit with related work
- Website suggests you write a critique/review for each paper
  - I will enforce this or not depending on how class discussions go

Exams / Project

- **Exams: 2 mid-terms**
  - De-emphasize memorization / regurgitation
  - Mix multiple choice, short answer long answer
  - Focus on design decisions, comparing systems, understanding of systems and tradeoffs
  - Likely on canvas/zoom

- **Project**
  - Work alone or in pairs
  - I can suggest topics (see website), but ideally, *you* choose.
  - Can be related to your research but if so, should extend your RA work meaningfully
  - Deliverables: milestones, written report, end-of-semester presentation

Today's Readings: Motivation & Professional Development

- Levin & Redell: "How (and How Not) to write a Good Systems Paper"
  - Summary:
    - Rules of thumb for writing good systems papers
    - Most papers are limited
    - A few researchers consistently publish at top conferences
    - Learn from them or risk a second-rate career
    - *A good framework to use for thinking about the papers we read in this class*
- Haldane '28: On being the right size
  - Summary: scale and use case are determinants for the fitness of a system
- (Optional) Hamming: "You and Your Research"
  - Summary:
    - Stay open to people and ideas
    - Research is both time-consuming and exciting
    - Research inspiration comes from unexpected directions

Levin & Redell

- Categories
  - Real system
  - Unimplemented system
  - Theoretical topic
- Most SOSP/OSDI papers are about prototypes
  - Some differences by field
    - E.g. architecture: more simulation (why?)
- Prototype is not enough
- Why build a system?
  - "The purpose of (scientific) computing is insight, not numbers." – Richard Hamming
  - Adapt old techniques to new technology

		Effort	SOSP Accept	SOSP Reject
Real system	Used by others	Huge	Occasional	Occasional
Worked once	Benchmarks ran by deadline	Large	Common	Occasional
Simulated	Simulations run	Large/med	Occasional	Common
Paper design	*Sounds* good	minimal	Rare	Common

On being the right size

- Why did we read this paper?
- Bold statements that are unusual in scientific literature
- Intellectual pursuits require breadth and depth
- Can you draw out lessons relevant to computer systems?

- Incommensurate scaling: eye size, jumping height
- Hardware constraints influence system design: strength of femur limits height
- Distribution of complexity and even basic assumptions change as a function of scale
  - E.g. MapReduce *algorithm* is a handful of lines of code
  - Apache Hadoop: 2,422,127 SLoC (as of 9/5/17)

Questions?