LA-UR-19-29023
Approved for public release; distribution is unlimited.

Title: All Lectures, CSCNSI 2019 Curriculum

Author(s): Wofford, John Lowell
Lueninghoener, Cory Donald

Intended for: Educational materials

Issued: 2019-09-06 (Draft)

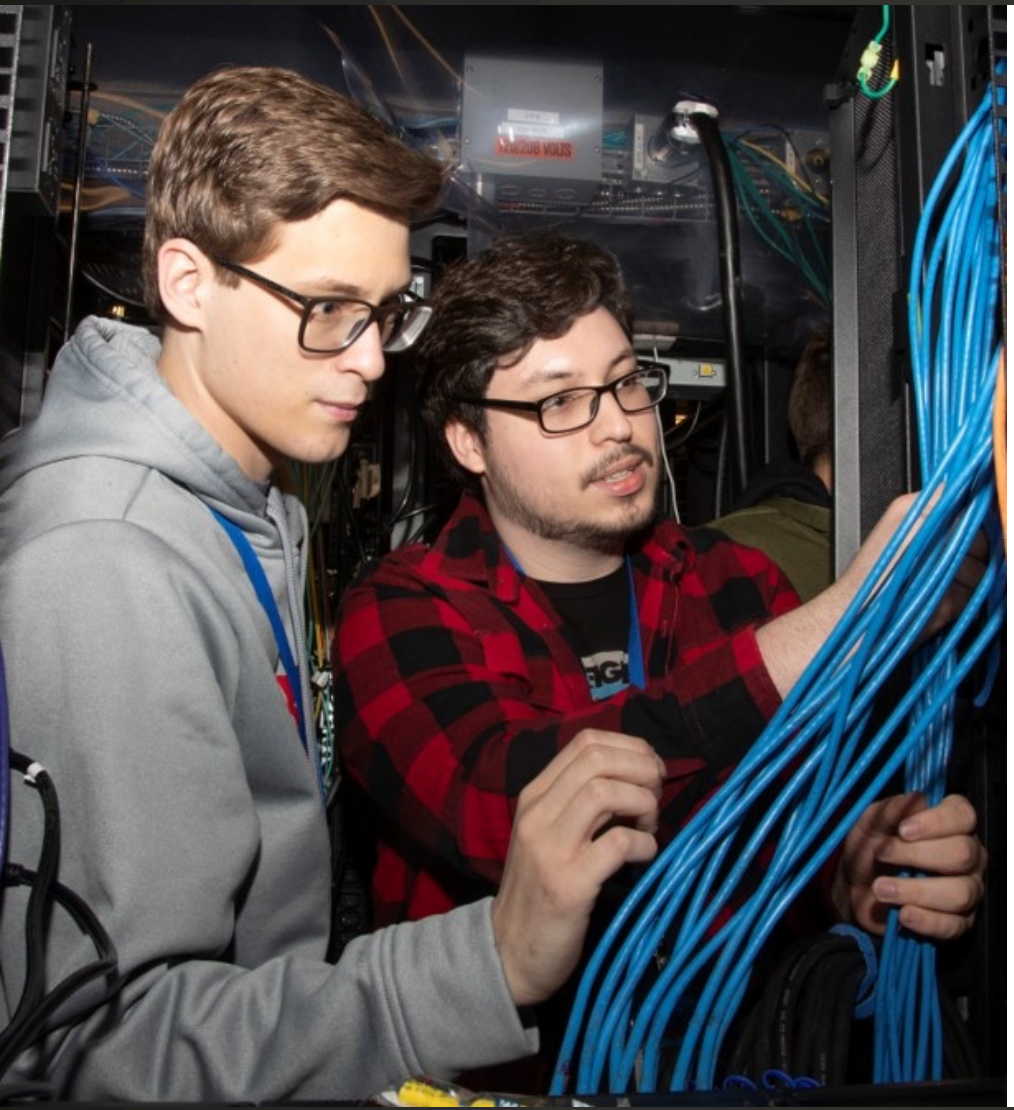
Los Alamos
NATIONAL LABORATORY
EST.1943

Delivering science and technology
to protect our nation
and promote world stability

Cluster Bootcamp

Course Introduction

Presented by CSCNSI



Agenda

- Introductions
- Course Goals
- Course Layout
- Expectations

Introductions

Instructors		
Lead Instructor	Lowell Wofford	HPC-DES, Futures/Integration
Instructor	Thomas Areba	HPC-SYS, Networking
Instructor	Kierstyn Brandt	HPC-SYS, Platforms
Instructor	Travis Cotton	HPC-SYS, Platforms
Instructor	Francine Lapid	HPC-ENV/CSCNSI Alumnus

Administrative		
Program Lead	Catherine Hinton	HPC-DES, ISSO
Deputy Program Lead	Reid Priedhorsky	HPC-ENV, Simulation & Analysis
Student Coordinator	Julie Wiens	HPC-DO, Intern Recruiter/Liason

About *you*

Tell us:

- Your name
- Where you're from
- A sentence about what brought you here
- A fun thing about yourself (e.g. hobbies, interests, experiences)

Course Layout

1. **Orientation** – *you should be mostly done with this already.*
2. **Bootcamp** – *our project for the next two weeks.*
3. **Research projects** – *work in teams with mentors on exciting research projects.*
4. **Posters & presentations** – *present your work! You should start working on these early.*

Week 1	Orientation
Week 2	Bootcamp
Week 3	
Week 4	Research projects
Week 5	
Week 6	
Week 7	
Week 8	
Week 9	Working on posters
Week 10	Posters & presentations

Course Agenda

1. Intro to HPC
2. HPC Facilities
3. Linux Essentials
4. Networks & Services
5. Stateless Netboot
6. HPC Tools & Provisioning
7. VCS & Configuration Management
8. Monitoring & Benchmarking
9. Using Compute Clusters
10. Intro to Parallel Programming & Visualization
11. Intro to Cluster Programming
12. HPC Futures

Goals: Bootcamp

- Learn the basics of cluster computing including:
  - Linux system administration & installation
  - TCP/IP & high-speed networking concepts
  - Common system services used for HPC
  - Cluster booting & provisioning
  - Monitoring & performance analysis
  - HPC job scheduling and workflows
  - Basics of parallel programming for clusters

Goals: Bootcamp (cont.)

- And, build your team cluster!
  - Physically cable and connect nodes
  - Install your “master” node
  - Learn to provision in 4 phases:
    - By manually installing nodes
    - By manually network booting nodes
    - By using a cluster provisioning tool
    - By using a cluster provisioning tool & configuration management
  - Learn to *use* your cluster
- At the end you should have a fully functional cluster
  - Many of you will use this cluster for your research projects, so this part is *important!*

Goals: Research project

- You'll be working on novel research projects to:
  - Learn HPC through detailed projects in specialty areas
  - Build skills in a specialty area
  - Develop general research skills
  - Learn to work with a team on highly technical projects
  - Learn to communicate and present your work
  - *And... advance an active HPC research question!*
- Attend talks for breadth of knowledge

Other course goals

- Enjoy yourselves; have fun!
- Meet new people with similar interests
- Play with a bunch of cool stuff
- Get to know the area

Course Layout: Bootcamp (weeks 1-2)

Timing may vary based on the day's content, e.g. for guest talks

- Day is ordered in 5 parts:
  1. **Morning lecture** – covering overviews and theory for the day
  2. **Practicum** – a guided lab to learn new skills
  3. **Lunch** – the most important part! (may be with guests)
  4. **Afternoon lecture** – will cover practical skills needed for lab
  5. **Lab** – dedicated time to work on your clusters
  6. *We will have a wrap-up meeting at 4:45 PM*

8 AM	Morning lecture
9 AM	Practicum
10 AM	
11 AM	
12 PM	Lunch
1 PM	Afternoon lecture
2 PM	Lab
3 PM	
4 PM	

Course Layout: Research projects (weeks 3-9)

Timing may vary based on the day's content, e.g. for guest talks

- During the research projects portion, you will spend most of the day working on your projects.
- You will be expected to regularly meet with your mentors and report on project status.
- You will be expected to attend the on-going lecture series (*see calendar*).

8 AM	Project status
9 AM	Research work
10 AM	
11 AM	
12 PM	Lunch
1 PM	Research work
2 PM	
3 PM	
4 PM	

What we expect from you

- Professionalism with
  - Each other
  - NMC/LANL teams
  - The instructors
- Work ethic
  - We have a lot to cover; this will be hard work
  - Show up on time every day
  - Pull your weight in your team
  - Help other teams when you can; this is a collaboration, not a competition

A note about research

- Research can be messy business, with no guarantees
  - Don’t be discouraged if your projects don’t go as planned
    - Look for what you aren’t expecting, that might be your most interesting result.
- These are *real* research projects; it may be that no one knows what the results of your project will be
  - Listen to your mentors
    - They want your project to succeed too.
- These are *genuine* problems that we care about.

Evaluations

- You will be evaluated by:
  - Your instructors
  - Your mentors
  - And, your teammates
- Your posters will give you a chance to showcase your efforts to the entire HPC/LANL/NMC community

Logistics

- Please be here *on time*:
  - Your day begins at **8:00 AM**
  - You are required to vacate the offices by **5:30 PM**
    - *Lock your laptops before you leave!*
- Your time here is short, so days off are strongly discouraged.
- Any time off *must* be coordinated with:
  - Your mentors
  - ***And***, Program leads (Catherine or Reid)
- If you have any issues (including personnel conflicts of any kind), please report them to an *Instructor*, *Mentor* or *Program Lead*.

Safety guidelines

Server rooms have real dangers: high voltage/amperage, heavy equipment & sharp objects, to name a few

- When working in the server room(s), always wear:
  - Closed toed shoes
  - Long pants
  - Earplugs (provided) are a good idea, please wear them if you'll be in the room for more than a few minutes
- If you don't meet this dress code, you won't be allowed in the server room. No exceptions.
- No heavy lifting
  - Server racking should be accompanied by instructors or mentors
- No electrical work
  - Aside from individual server plugs, leave electrical work to instructors or mentors
- Never touch equipment that is not in your rack (including network connections)
- **Stop any activity that appears dangerous!**
- **When in doubt, ask questions!**

Please give us feedback!

- Help us make this a great experience
- We will be providing ways to give both open and anonymous feedback – *please use them*
- If there's something we can do to make your experience better, let us know!

Questions?

Los Alamos

NATIONAL LABORATORY

EST. 1943

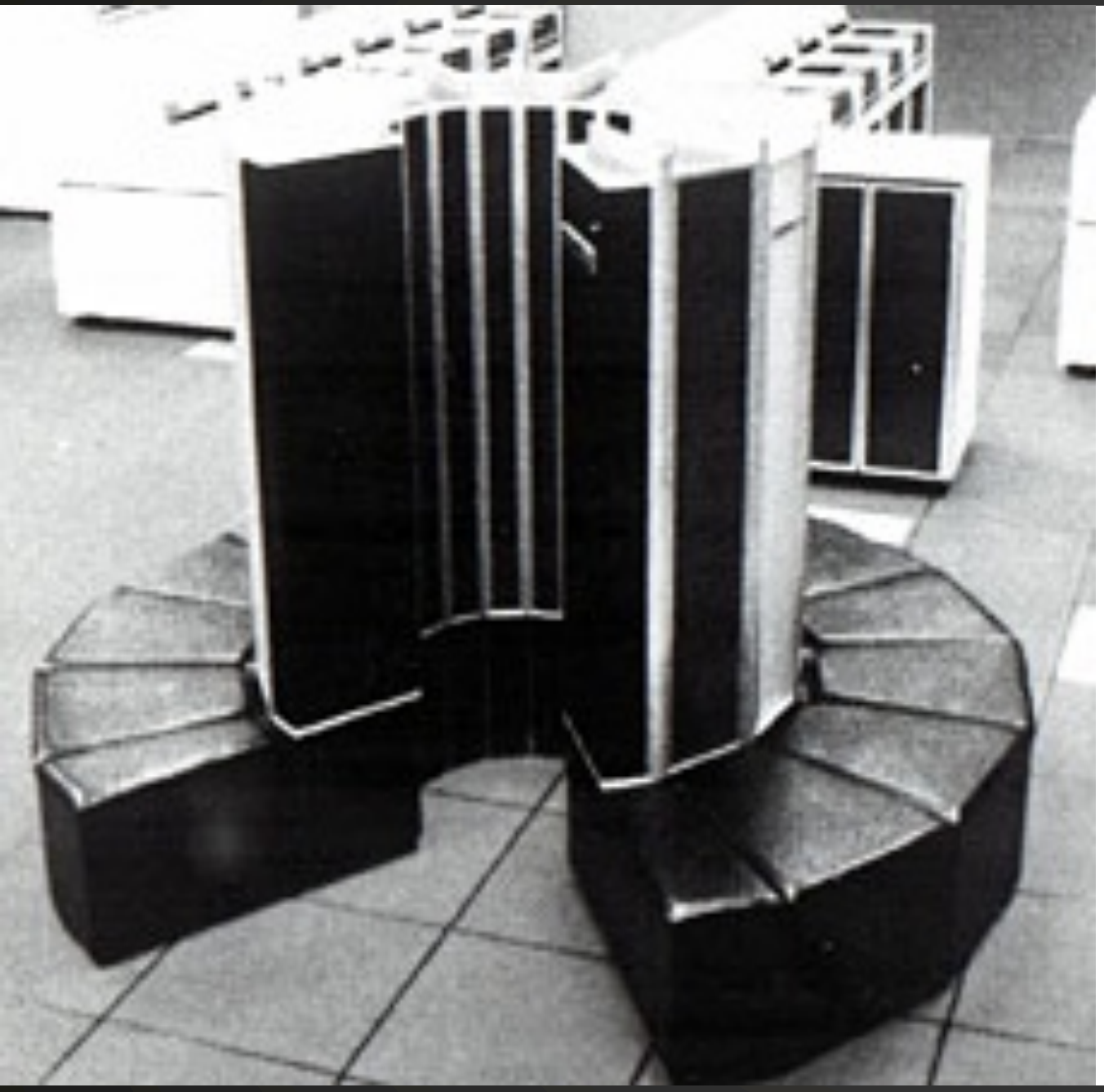
Los Alamos
NATIONAL LABORATORY
EST.1943

Delivering science and technology
to protect our nation
and promote world stability

Intro to High Performance Computing

Everything you need to know in 60 minutes

Presented by CSCNSI



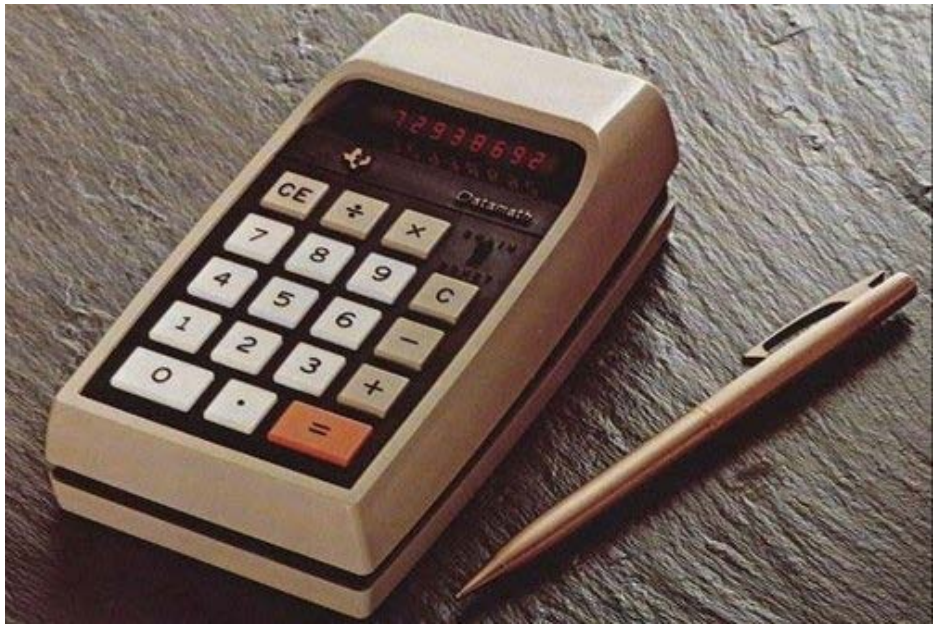
Agenda

- An toy problem
- An example real(ish) problem
- The Underlying Computer Science
- Cluster Computing
- HPC at Los Alamos

A Toy Problem: Summing Numbers

Serial Computing

- The problem: Add all of the numbers between 1 and 100
- The constraint: It takes you two seconds to add two numbers on your calculator
- How long does it take to add up 1 to 100?
  - Approach: Accumulate the sum in a single variable
    - n = 1+2; n = n+3; n = n+4; …, n = n+100
  - Time: (99 additions) * 2s = 198s = 3.3 minutes
  - Plus, your finger hurts



Serial Computing

- How could you make this faster?
  - Make the calculator's processor faster
    - But this isn't really the limiting factor
  - Make your finger work faster
    - This is the slow part, but there's a limit
  - Leverage the associative property of addition to recompose the problem as independent sets of addition, and then distribute those to more people

The women who operated the wartime Laboratory's desktop calculators and punched-card machines were themselves were called "computers." They often were the wives of Los Alamos scientists.

Parallel Computing

- The same problem: Add all of the numbers between 1 and 100
- The same constraint: It takes you two seconds to add two numbers on your calculator
- But you have three friends with calculators too
- How long does it take to add up 1 to 100?
  - Approach: Each person adds 25 numbers; one person then adds the four subtotals together
  - Parallel Time: (24 additions) *2s + (3 additions) *2s = 54s
  - Serial Time: 198 seconds
  - Speedup: 198s / 54s = 3.67x
- Note: 4x resources did not result in 4x speedup. Why not?

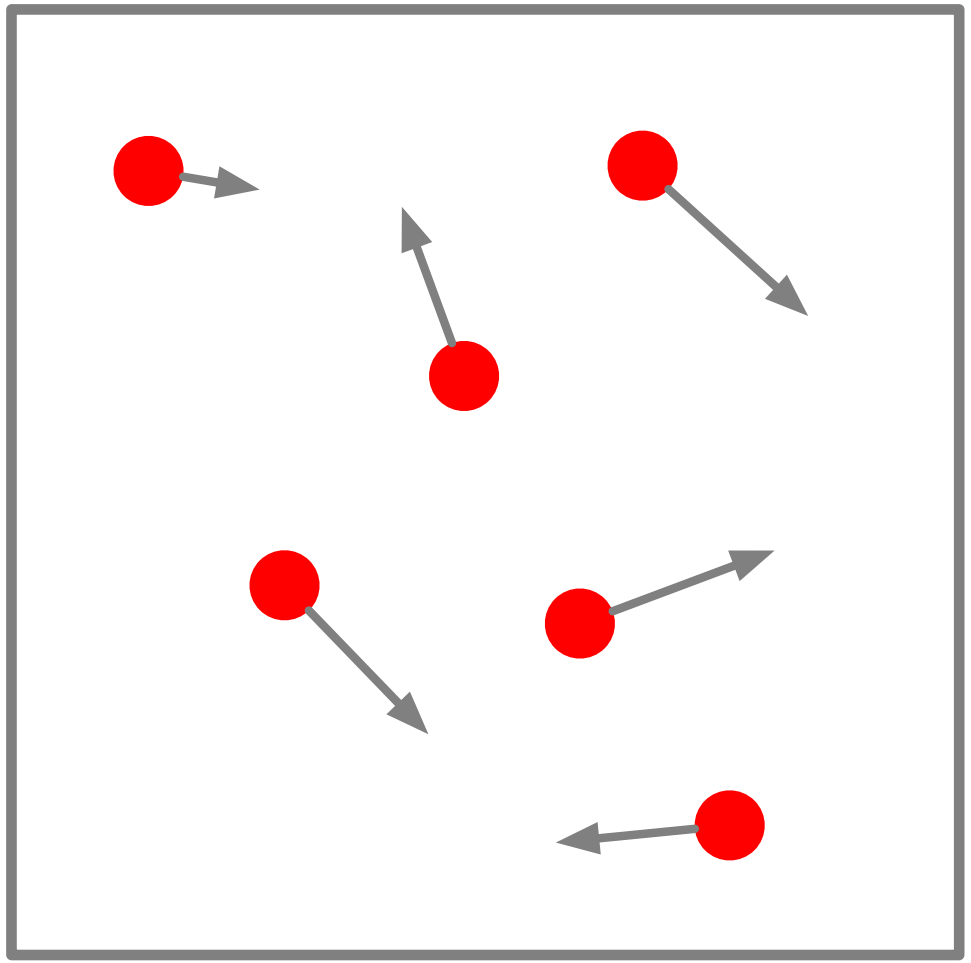
Limits of Parallel Computing

- This problem has two parts: one part that can be parallelized (the first stage additions), and one that can't (the final reduction)
- If you try to parallelize this too much, you start losing efficiency
  - 10 people
    - (9 additions)*2s + (9 additions)*2s = 36s
    - Speedup: 198s/36s = 5.5x
  - 25 people
    - (3 additions)*2s + (24 additions)*2s = 54s
    - Speedup: 198s/54s = 3.67x
  - 50 people
    - (1 addition)*2s + (49 additions)*2s = 100s
    - Speedup: 198s / 100s = 1.98x
- Eventually, the serial part of the computation dominates the parallel part

A Real(ish) Problem: Particles in a Box

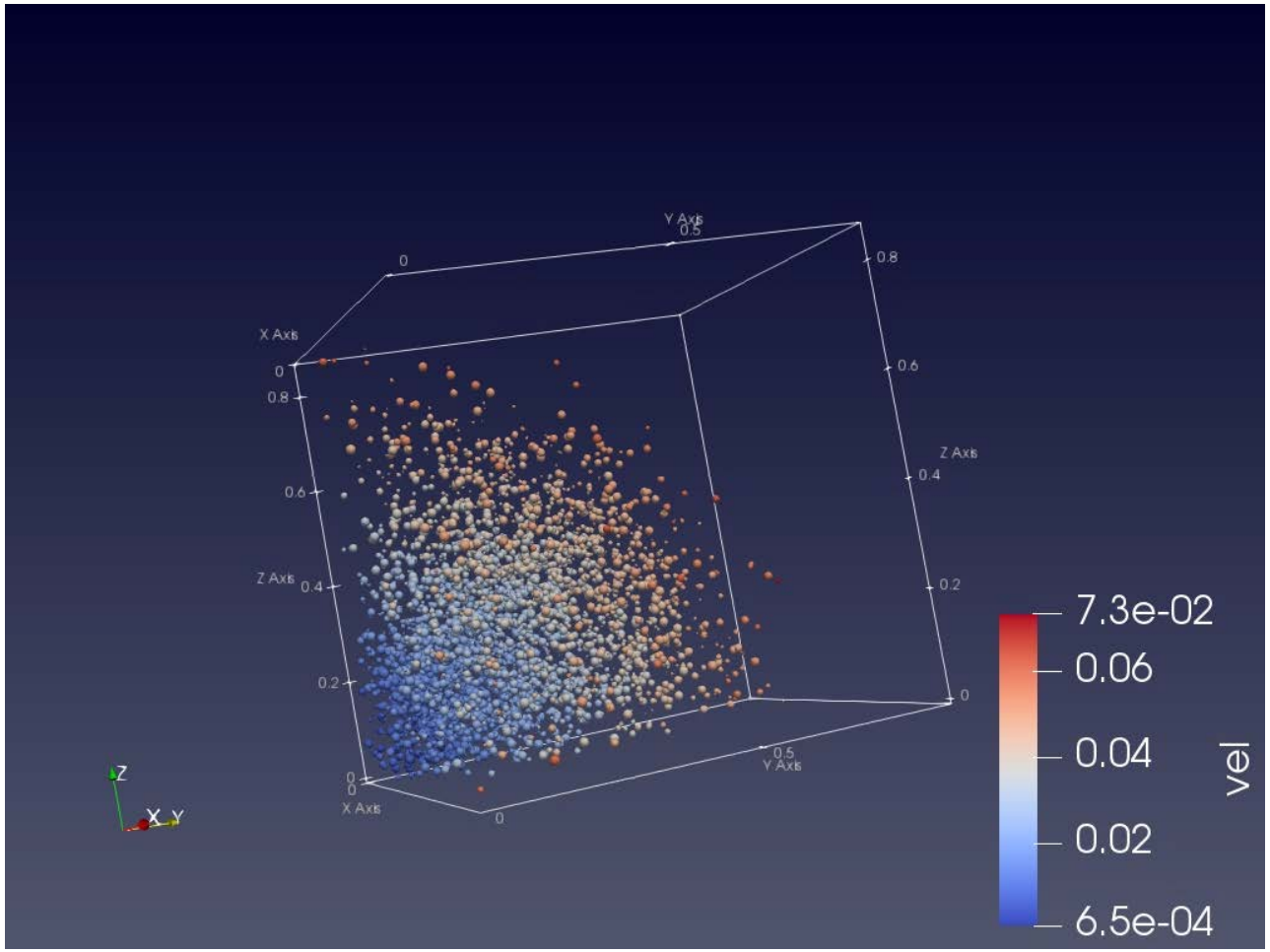
Real Parallel Computing

- The problem: Simulate the interactions between gas particles in a cube-shaped room
- The approach:
  - Model each particle as a set of coordinates in a three dimensional space and a three dimensional motion vector
  - Give each particle a set of initial conditions
  - Start a loop that performs one simulation timestep with each pass through the loop

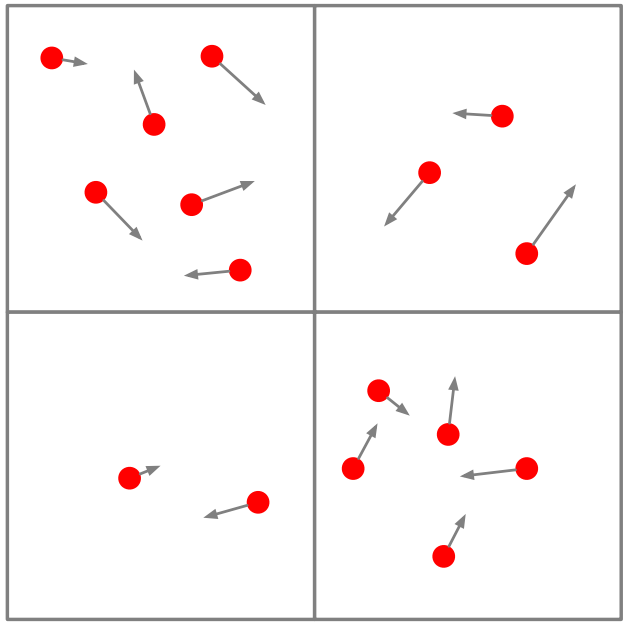


Serial Particles in a Box

- Running on one single processor system
- Store the particles in an array
- For each timestep of the simulation:
  - For each particle in the array:
    - Given the position and motion of this particle, calculate which other particles it will collide with in this timestep
    - Update motion vectors accordingly
- Some Advantages:
  - Simple to code
- Some Disadvantages:
  - Slower than parallel approaches
  - Size of simulation limited by system's memory size



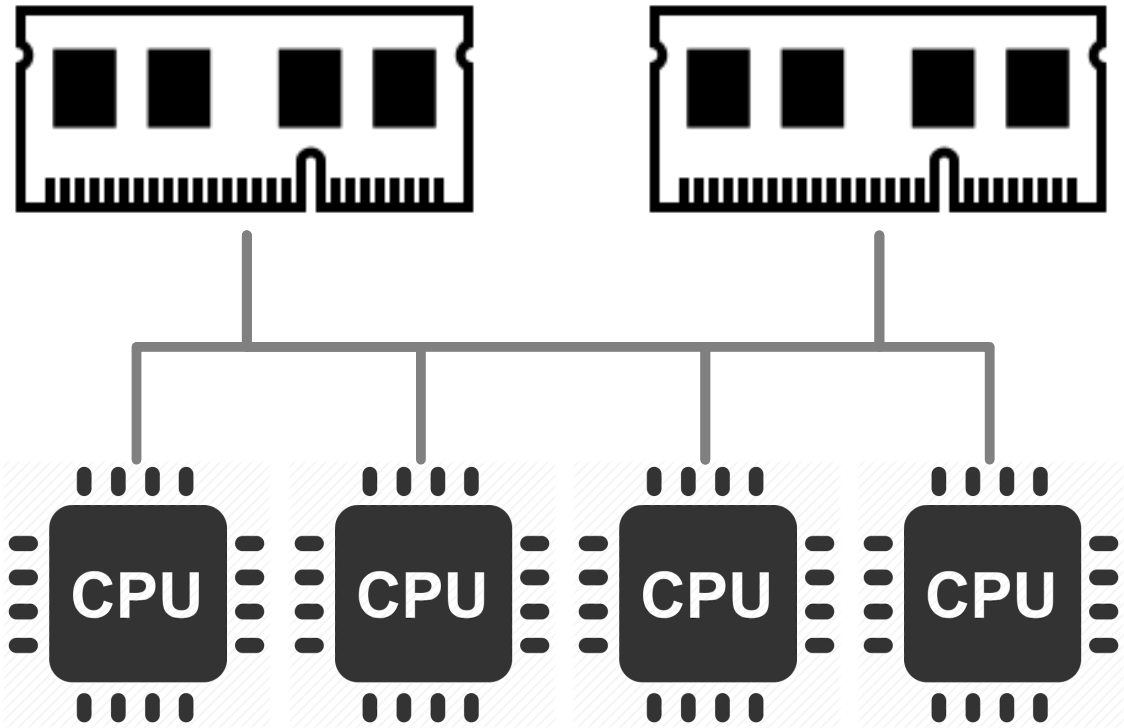
Parallel, Shared Memory Particles in a Box



- Running on one multi-core system
- Divide room into *n* equally-sized cells
- Store particles in one shared array
- Assign each cell to one of the processor cores:
  - Each core is responsible for running calculations on particles in its cell
- For each timestep in the calculation, each core runs:
  - For each particle currently located in the core's cell:
    - Given the position and motion of this particle, calculate which other particles it will collide with in this timestep
    - Update motion vectors accordingly
- Since this is a shared-memory system, all cores can see and update all particles

Parallel, Shared Memory Particles in a Box

- Some advantages
  - Can run many times faster than the serial version
- Some disadvantages
  - More complex than the serial version
  - Scaling larger requires adding higher-core processors to the system
  - Size of simulation still limited by system's memory size



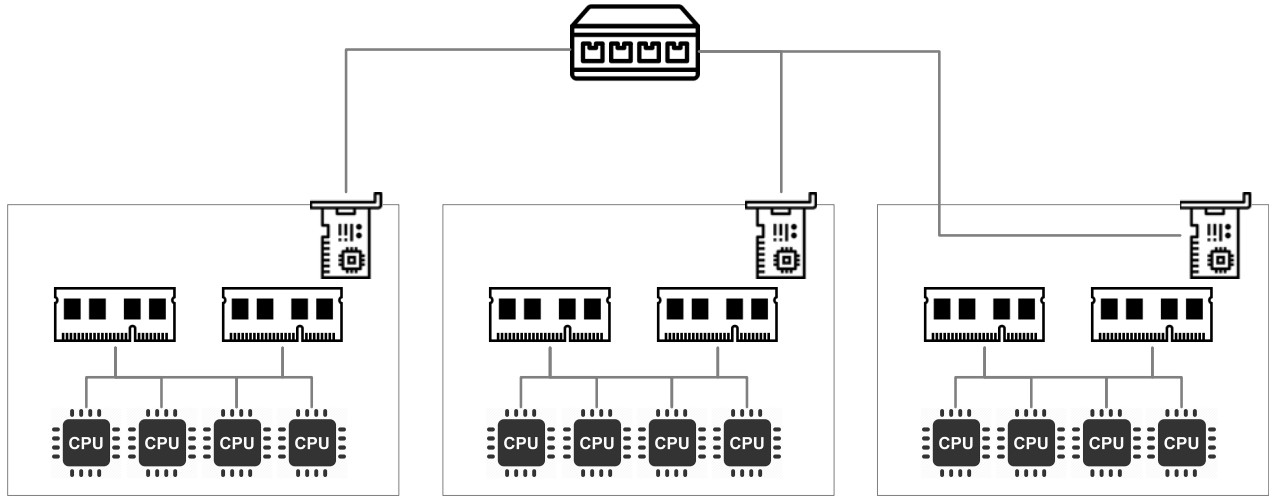
Parallel, Distributed Memory Particles in a Box

- Running on many single-core systems
- Divide room into *n* equally-sized cells
- Assign each cell to one of the systems:
  - Each system is responsible for running calculations on particles in its cell
- Store a cell's particles in an array on the system that owns it
- For each timestep in the calculation, each system runs:
  - For each particle currently located in the cell:
    - Given the position and motion of this particle, calculate which other particles it will collide with in this timestep
    - Update motion vectors accordingly
- Since this is a distributed memory system, information about a particle that leaves a cell must be explicitly sent to the system that owns the particle's new cell

Parallel, Distributed Memory Particles in a Box

- Some advantages
  - System memory only limits the size of a cell, not the size of the whole simulation
  - More processors can be added by adding new compute nodes
- Some disadvantages
  - Much more complex than the serial version
  - Communication can quickly become the bottleneck

- *One final version*: Run on many multi-core systems
  - Local cells can share memory, remote cells need to pass information
  - This is how modern parallel codes actually work



The Underlying Computer Science

Some Computer Science

- Processors can handle instructions and data in different ways:
  - SISD: Single Instruction, Single Data
  - MISD: Multiple Instruction, Single Data
  - SIMD: Single Instruction, Multiple Data
  - MIMD: Multiple Instruction, Multiple Data

Some Computer Science: The Processors

- Single-core CPU: SISD
  - One instruction run on one set of registers during each clock cycle
- Multi-core CPU: MIMD
  - Independent instructions run on independent sets of registers during each clock cycle
- Single-core CPU with Vector Additions: SISD + SIMD
  - Can do both: one instruction run on one set of registers, or one instruction run against a vector (or “list” or “array”) of registers
  - Examples: SSE, AVX
- GPU: SIMD
  - Load up lots of registers, run instructions across all of them *very* quickly

Some Computer Science: Sharing Data

- Shared memory:
  - On a single system, all processors can see all of the memory
  - This makes it easy to share memory between processes: it's all accessible
  - But! It might not all be accessible at the same speed
  - Non-uniform Memory Access (NUMA) designs place parts of memory "closer" to some processors than others – this results in longer access times for cross-NUMA domain reads
  - Scales to as many processors and as much memory as you can fit in a single system
- Parallelization techniques that make use of shared memory:
  - Threads: The most basic way to share memory between multiple process-like objects
  - OpenMP: Specification for adding parallelism to C, C++, and Fortran

Some Computer Science: Sharing Data

- Message passing:
  - Distributed systems cannot see each others' memory
  - This makes it harder to share memory between processors, but not impossible
  - Software libraries exist for explicitly sending information between systems
    - This is referred to as Message Passing
  - This information is normally sent over a high speed, low latency network
  - But! Again, it may not all be accessible at the same speed
    - Network topologies make some nodes "closer" to and "farther" from other nodes
  - Scales as far as your network can grow
- Parallelization techniques that make use of message passing:
  - RPC: A standard concept with many different implementations for moving data between systems
  - MPI: The most common high performance computing API for message passing

Cluster Computing

Designing a compute cluster

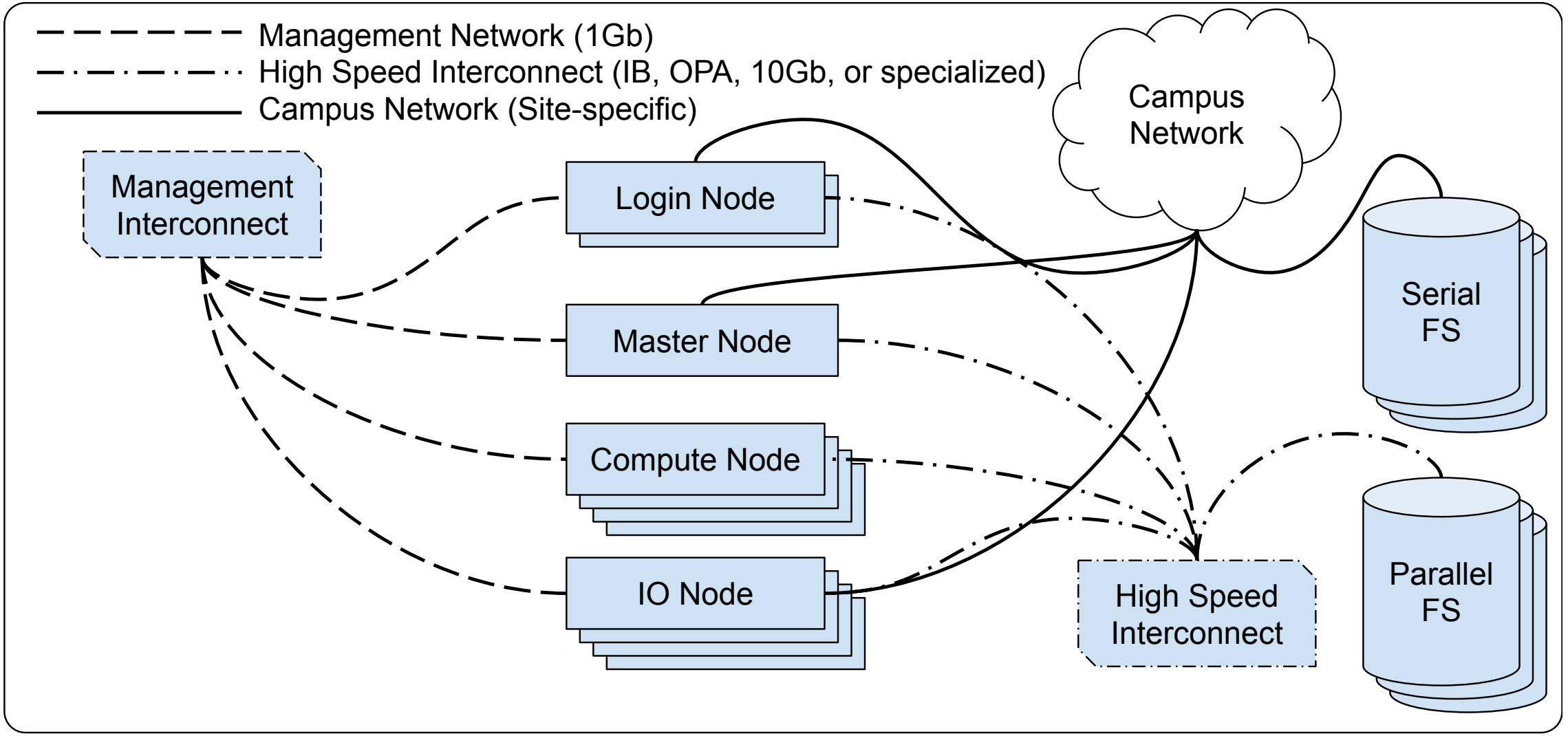
- Using what we now know about parallelization and memory, how do we build a system that can actually simulate the gas particles in a room?

- Today, HPC systems are most commonly implemented as clusters of independent computers
  - So they use a combination of shared memory & message passing.
- In this model, applications have to balance the available processing cores, memory space, network bandwidth, and other factors to get the best performance
- When designing a cluster, we try to find the best combination of hardware, software, and physical layout to optimize one or more of these factors

Typical cluster hardware

- HPC Clusters consist of…
  - Compute nodes
    - Tens, hundreds, thousands, or tens of thousands of individual computers
  - Infrastructure nodes
    - Tens or hundreds of computers that provide login gateways, manage compute nodes, interface with external filesystems, and perform other duties
  - Networks
    - Low-speed (1Gbps, frequently) management network
    - High speed (hundreds of Gbps, frequently) computation network
  - Filesystems
    - Frequently separated from clusters, both physically and logically
    - Standard serial network filesystems (NFS, normally) for home directories, specialized parallel filesystems (Lustre, GPFS) for computational use

A Typical HPC Cluster Layout



HPC at Los Alamos

So, what is HPC?

- There is no single definition of what high performance computing is
- But there are several indicators:
  - Often scientific in nature
  - Often highly parallel
  - Often tightly coupled
  - Often involves a dedicated high-speed network
  - Often involves a dedicated parallel filesystem
- … but sometimes doesn't involve many of those at all

HPC at LANL

- LANL HPC is generally…
  - **Scientific**: physics, chemistry, biology, astrophysics, mechanical engineering, climatology, computational entomology, and more
  - **Parallel**: sometimes small parallelization (O(10^1) cores on one node), sometimes enormous parallelization (O(10^4) cores on thousands of nodes)
  - **Tightly coupled**: frequent communication between many or all nodes
  - **Run on clusters**: systems with dedicated high-speed networks and parallel filesystems using message passing

HPC in the Cluster Institute

- Over the next two weeks, you will build a cluster that is very similar to larger clusters at the lab
- These clusters will exhibit many of the traits we've talked about in this lecture
- Specs:
  - 10 individual compute nodes
  - Each compute node having multiple cores
  - All compute nodes tied together by an Infiniband high speed network
  - Capable of running parallel scientific applications
- We'll start building this system later today!

Questions?

Los Alamos
NATIONAL LABORATORY
EST.1943

Delivering science and technology
to protect our nation
and promote world stability

Facilities Challenges for HPC

Space, power & cooling

Presented by CSCNSI

Agenda

- Facilities challenges for HPC
  - What does it take to run Trinity?
  - Some ways we handle facilities issues

HPC creates serious challenges for facilities

Trinity Supercomputer Facts	
Nodes	19,208
Compute Cores	967,456
Peak Perf.	43.9 PFlops
Floor space	5200 sq.ft.
Power	9.4 MW
Weight	~ 250,000 lbs
Liquid Vol.	18,000 ga

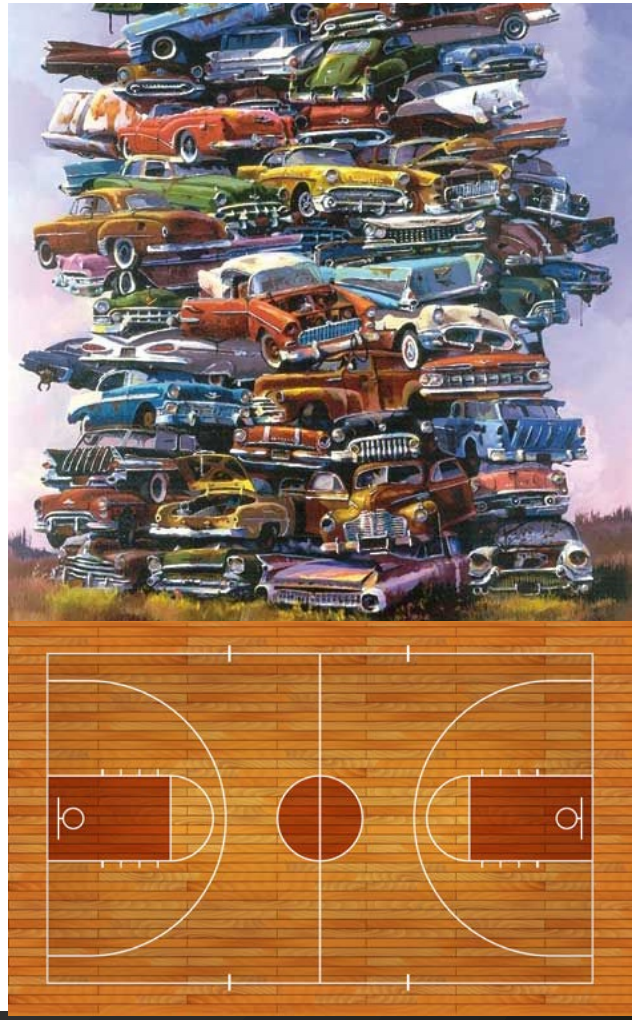
So, what does it take to run Trinity?

How do we put 20,000 computers in a room?

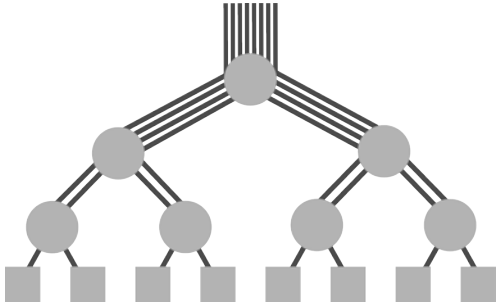
- …and fit them all?
- …and connect them all?
- …and power them all?
- …and cool them all?

How do we put 20,000 computers in a room…
…and fit them all?

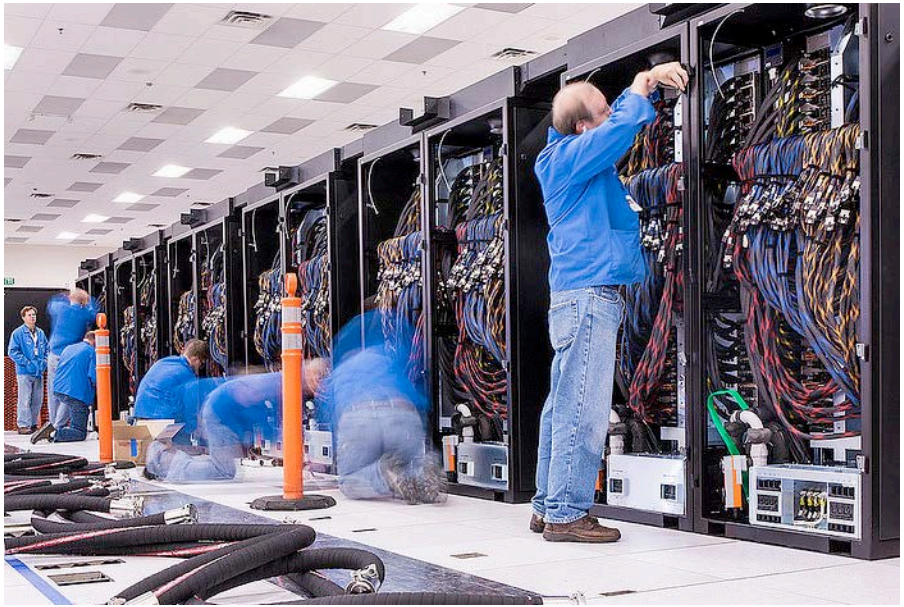
- Trinity's racks weigh > 3500 lbs each
  - That's more than an average car
- Trinity sits on roughly 5,200 sq. ft.
  - A bit bigger than a basketball court
- So… we have to fit about 80 cars on a basketball court
  - We'll have to stack them 2 deep!
- *We have to carefully engineer for weight and density, or we could literally fall through the floor!*



How do we put 20,000 computers in a room…
…and connect them all?



- Just naïvely connecting 20,000 nodes would take
- Using calculator at: https://clusterdesign.org/cgi-bin/network/network
  - > 40,000 cables
  - 95 (very large) switches
- We're going to need to stay very, very organized!
- (this isn't actually how Trinity is connected)



How do we put 20,000 computers in a room…
…and power them all?

- Trinity has a peak draw of 9.4 MW
- That's as much power as the rest of the county uses, combined
- At draws like this, we have to be especially careful about powering on or off everything at once
  - Can even blow power substations!

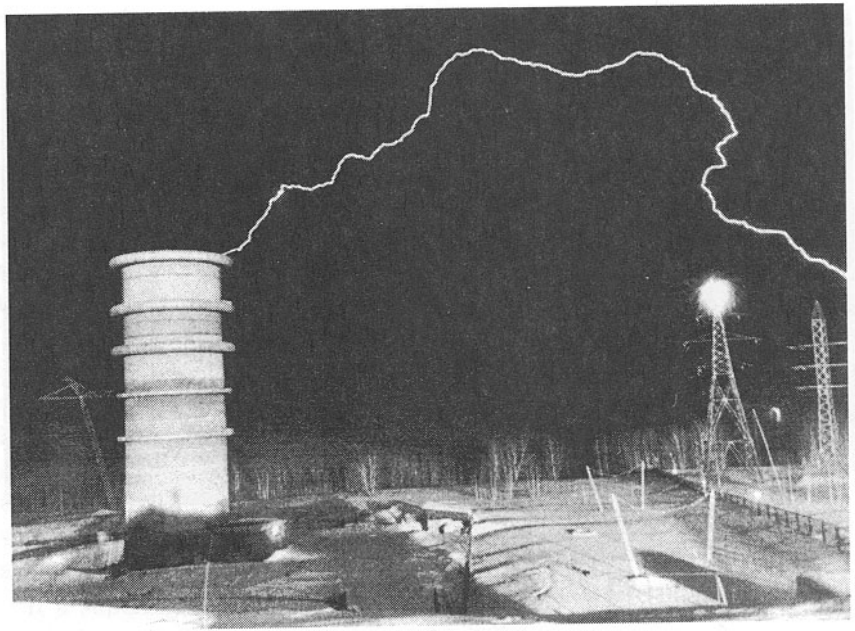
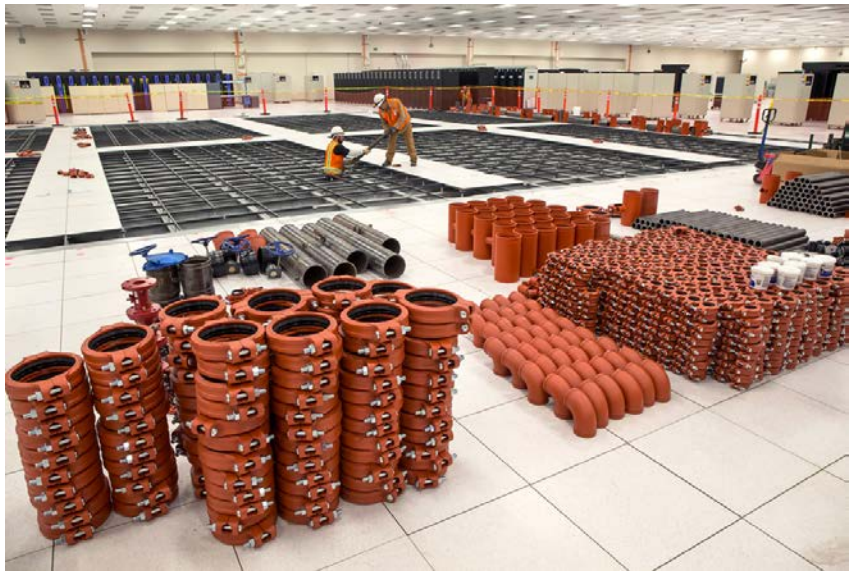


FIGURE 1.2
Superlong negative discharge to a 110 kV transmission line wire. Voltage pulse amplitude, 5 MV. Courtesy of A. Gaivoronsky and A. Ovsyannikov, the Siberian Institute for Power Engineering.

How do we put 20,000 computers in a room…
…and cool them all?

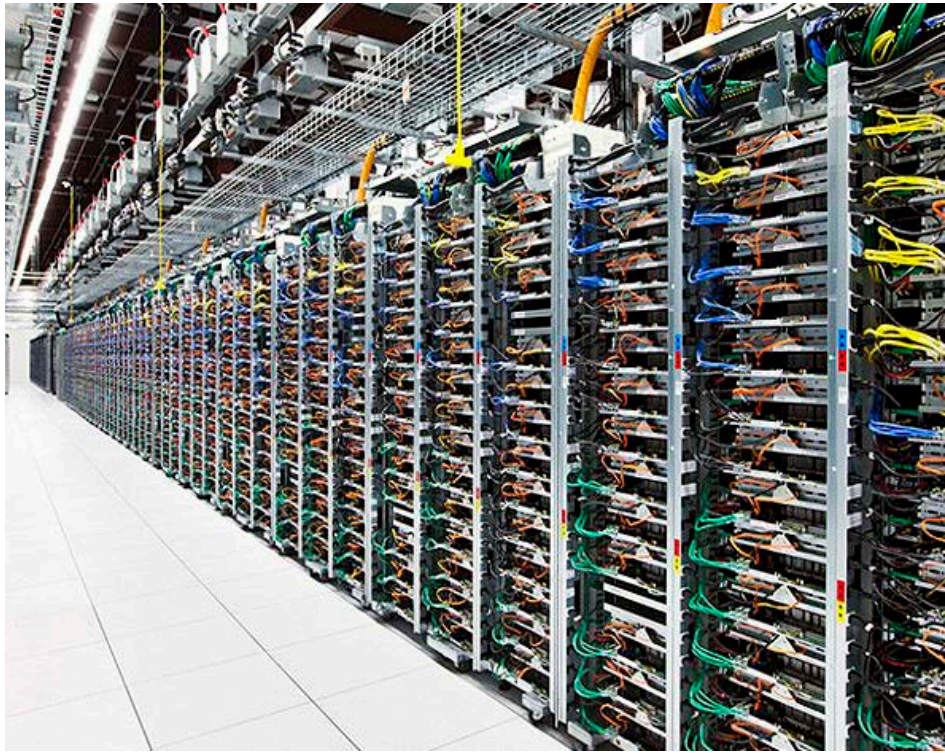
- For every bit of power we use, that is heat created
- 9.4 MW ~= 3000 tons of refrigeration
- There are 18,000 gallons of coolant in Trinity's main loop
  - …and much more in the total loop.
- That's 150,000 lbs of coolant
- And an additional 300 tons of air cooling



Some ways we handle Facilities issues

What we mean by an HPC Facility

- Any physical environment that provides:
  - Power
  - Cooling
  - Physical space
  - Physical security
- For HPC equipment



Space planning

- Arrange rooms in well-ordered rows
- Use raised floor systems for things like cables, power, plumbing
- Carefully calculate weight distribution to avoid overloading floors

Cable management

- For smaller systems, organized & orderly cabling is good enough
- For larger systems, we usually build more than one system into a chassis/rack that don’t have to use external cables
- We often put overhead trays and/or use a raised floor for cable management

Power management

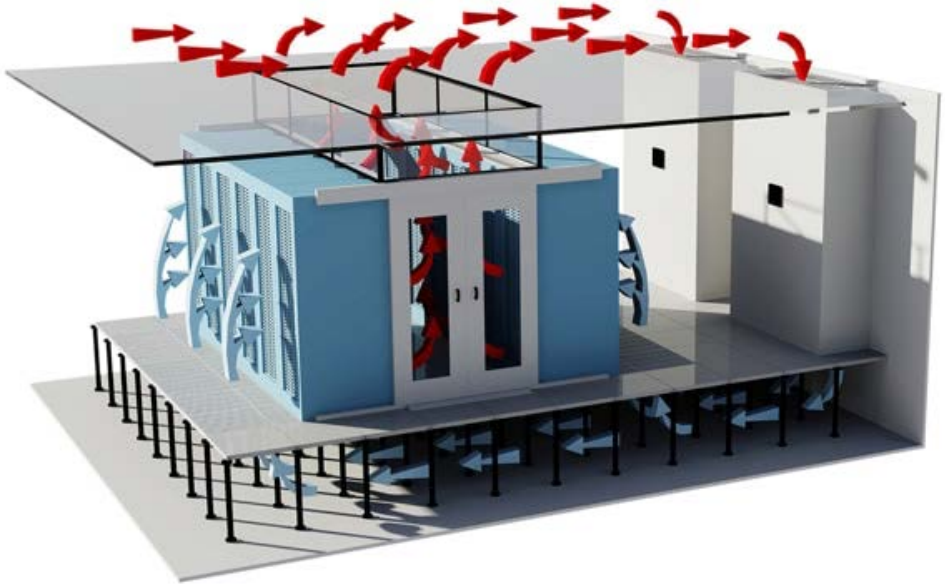
- We use *a lot* of power
  - And typically
- We use smart startup/shutdown procedures to mange the ramp up/ramp down problem
- We often run equipment at high voltage (e.g. 480 V)
- We typically tree the power through different tiers
  - Building distribution -> Floor PDU -> Rack PDU

Floor PDU

Rack PDUs

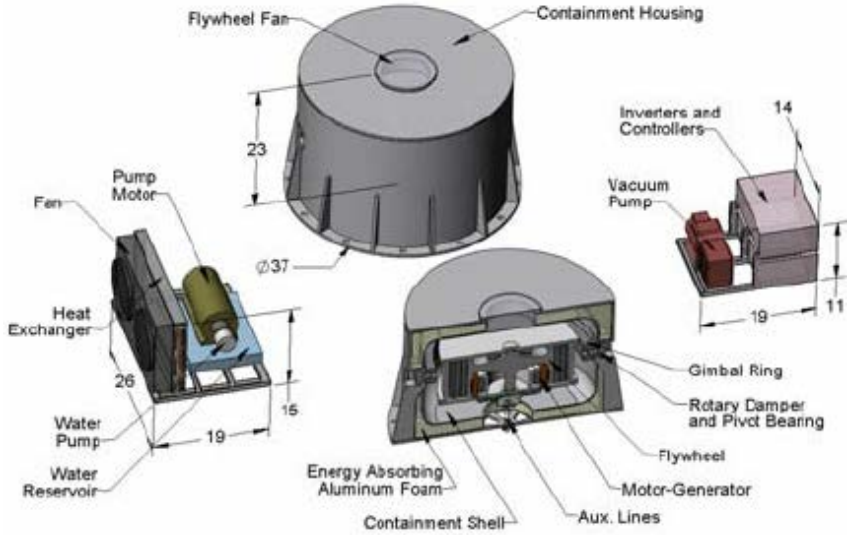
Cooling systems

- Air cooling can be managed with careful arrangement and planning
  - Keep hot air isolated
  - Inject cold air where needed
- Integrated high voltages racks are often liquid cooled
- The liquid can be kept cool through
  - Refrigeration
  - Evaporative cooling...



This is just a taste…

- We also have to think about things like:
  - Can the room withstand an earthquake?
  - …a flood?
  - …a fire?
  - …a power outage?
  - …

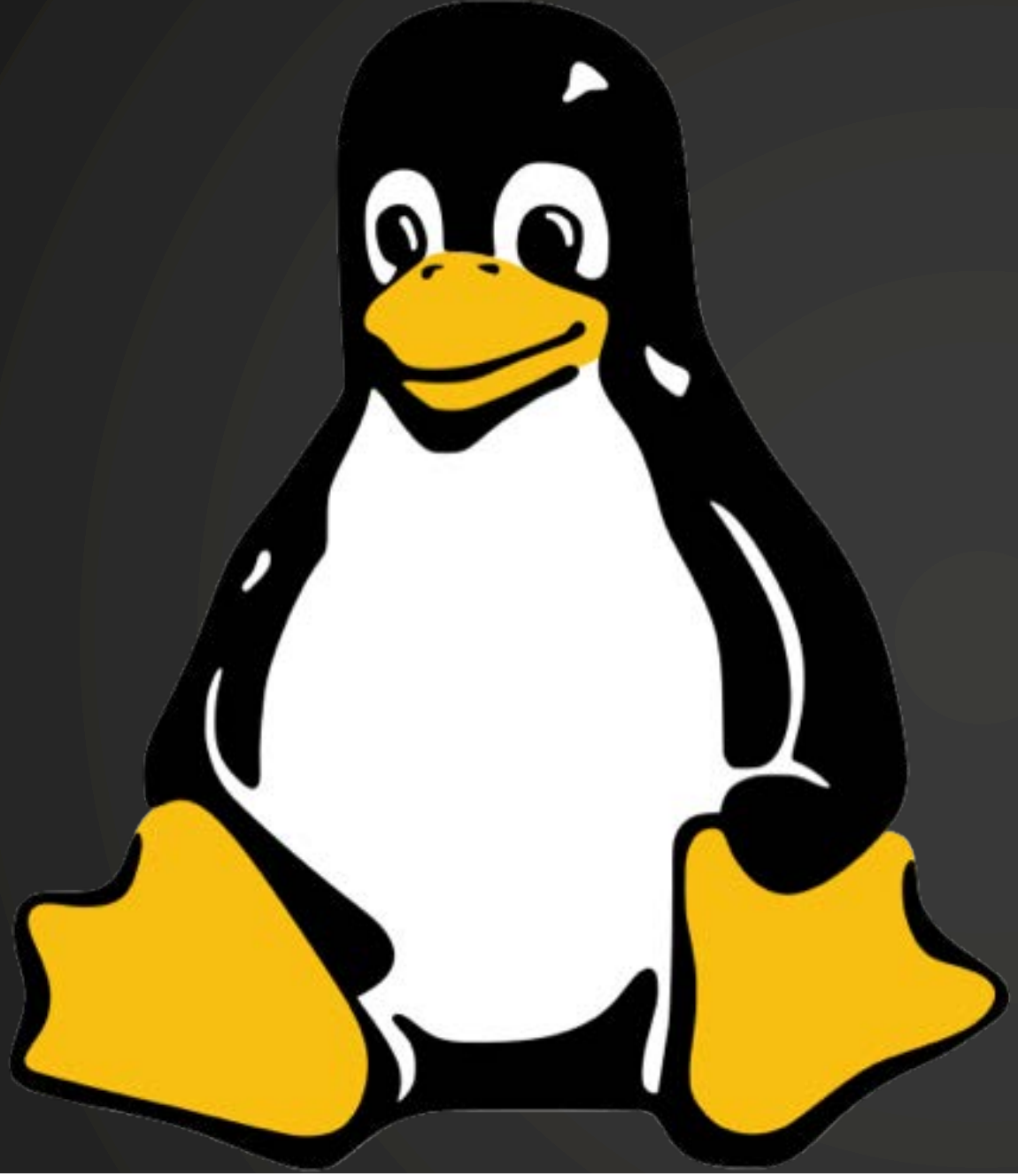


Questions?

Los Alamos

NATIONAL LABORATORY

EST. 1943



Los Alamos
NATIONAL LABORATORY
EST. 1943

Delivering science and technology
to protect our nation
and promote world stability

Overview of Linux

Theory of Linux with a focus on HPC

Presented by CSCNSI

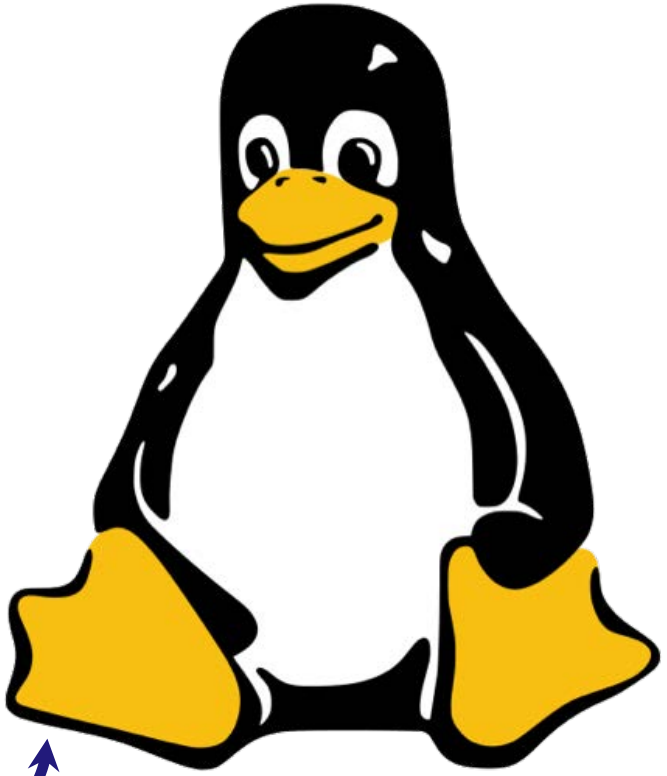
Outline

1. What is Linux?
2. The Linux Kernel
3. Linux distributions

What is Linux?

What's in a name?

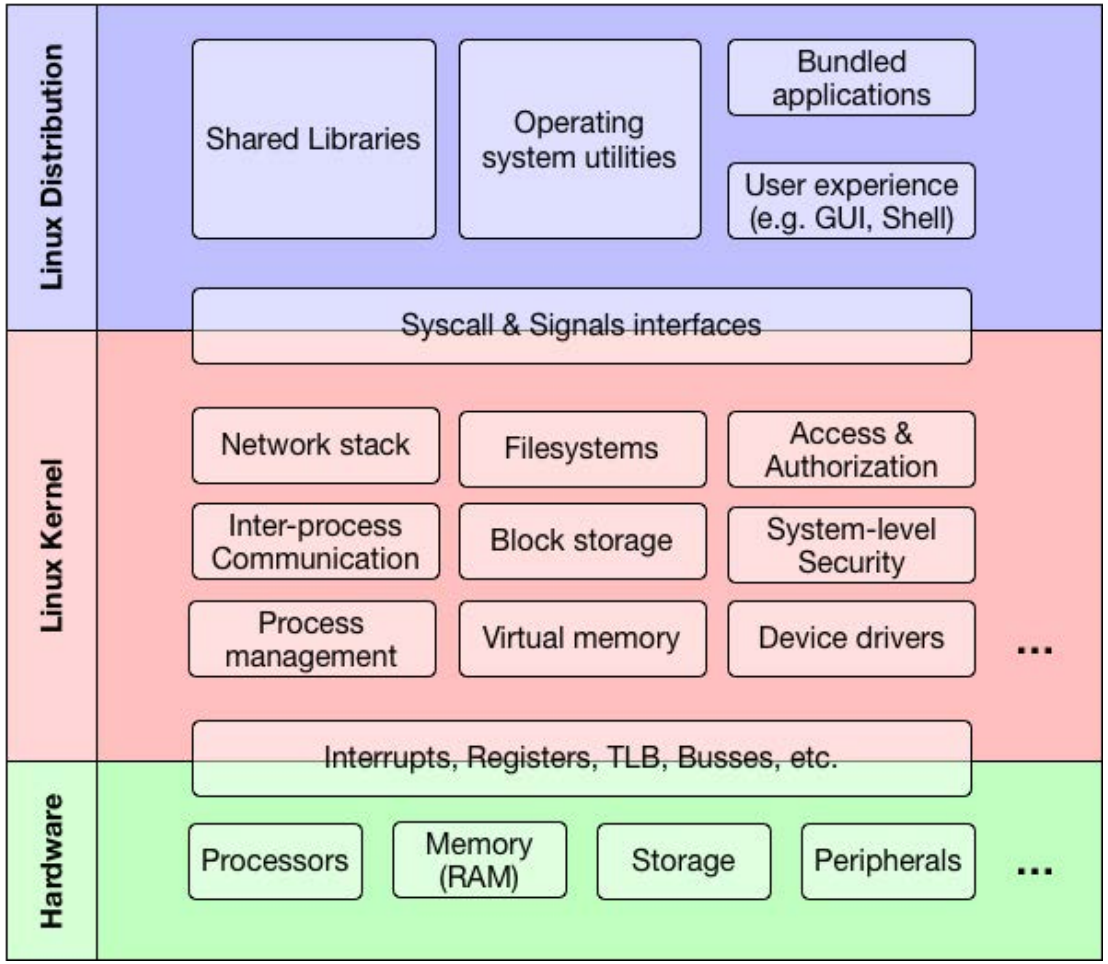
- The term "**Linux**" is overloaded. People often mean one of three things:
  - The Linux **kernel**–the core of the operating system
  - A Linux **distribution**—a bundle of software centered around the Linux kernel
  - The Linux software **community**—a vast and vibrant grouping of (mostly) open source development groups
  - *…and probably other things too.*



This is "Tux" the penguin. Tux is the official Linux mascot.

Linux vs. Linux distribution

- The Linux Kernel is the core of Linux
  - Provides interface between software and hardware
  - Provides process scheduling, memory access, hardware drivers, filesystems, and much more
- A Linux distribution is a bundle of software centered around the Linux kernel
  - To have a usable system we need more than the kernel, so various Linux Distributions exist that bundle needed tools
  - Red Hat, CentOS, Scientific, Debian, Ubuntu, Slackware, Gentoo, Arch, Mint, Raspbian, and so many more…



The Linux community

The Open Source community drives the development of the Linux

Special mention: **GNU** (pronounced *g'noo*)

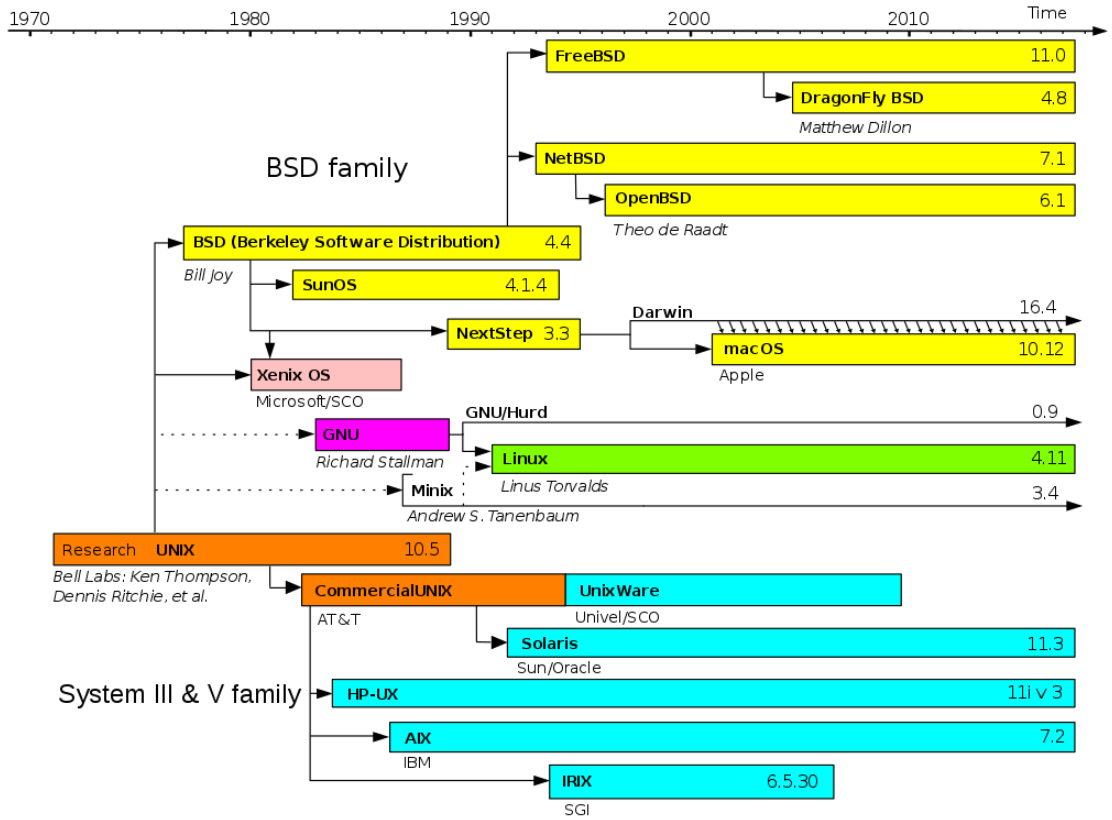
- Much of the system software typically used with the Linux kernel is part of the GNU project. All of the GNU project is free, open source software. This combination is often called **GNU/Linux**.

- Many other communities:
  - Linux Foundation
  - Free Software Foundation (FSF)
  - Apache Software Foundation
  - GNOME Foundation
  - KDE e.V.
  - Mozilla Foundation
  - Open Source for America (OSFA)
  - OpenStack Foundation
  - ...many more

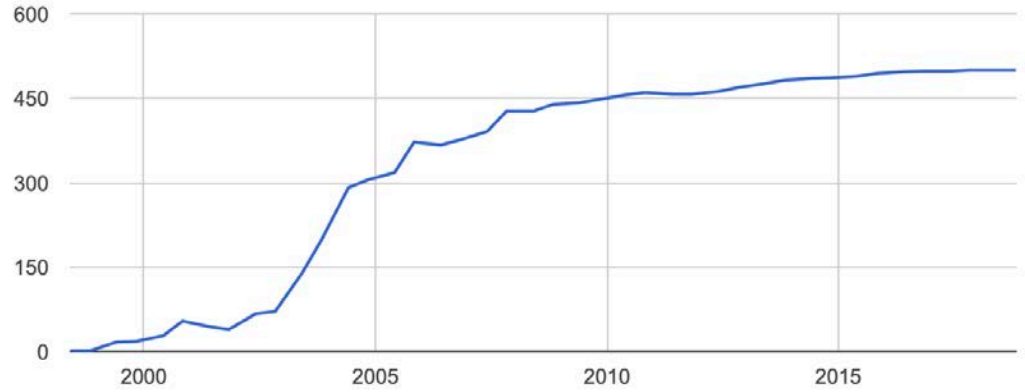
Plus lots of communities and individual developers at places like github, SourceForge,...

History of Linux in HPC

- 1991—Linux kernel is written by Linus Torvaldes (a 21yr old student).
- 1994—First compute cluster, “Beowulf,” created at NASA.
- 1996—Linux kernel v2 released, supports SMP (symmetric multiprocessing).
- 1998—First Linux cluster on the top500 list.
- 2004—Cray (the supercomputer company) releases its first Linux-based systems.
- 2012—Linux runs on all of the top10 of the top500.



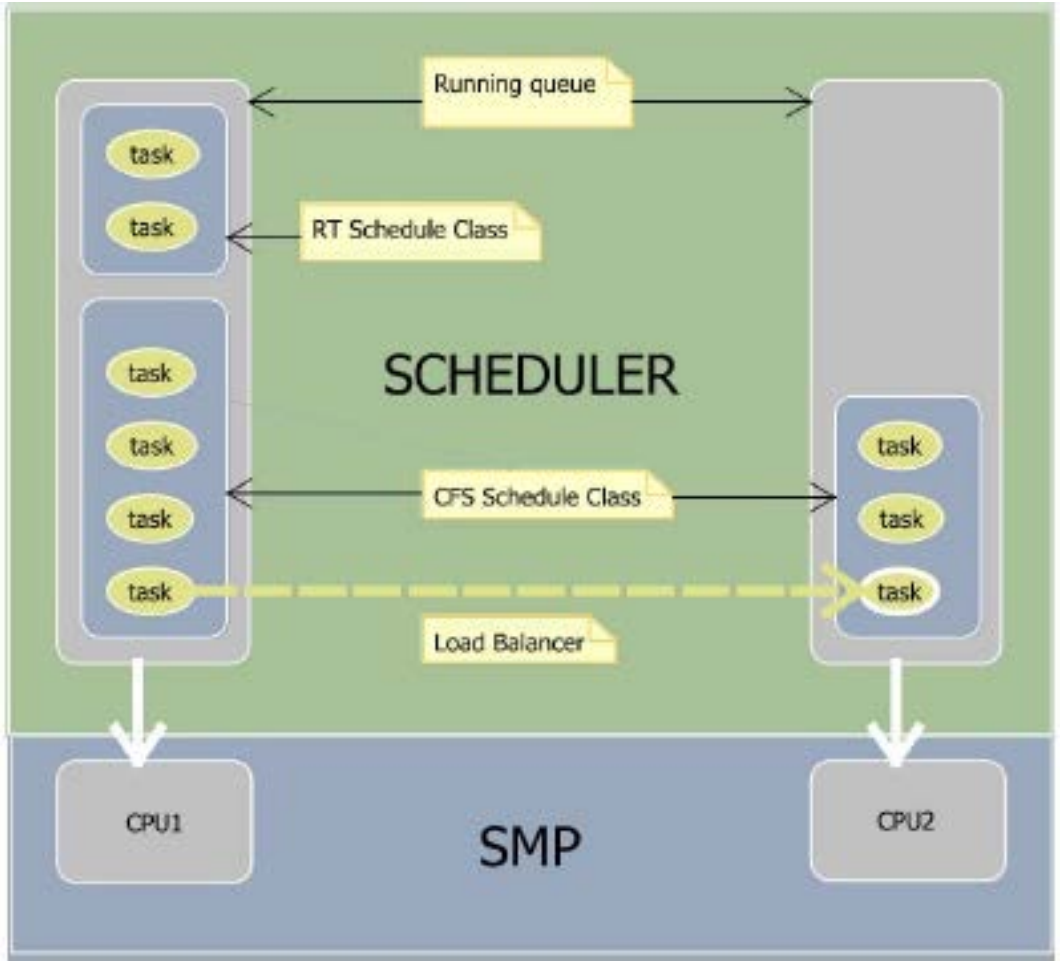
(above) History of Linux & UNIX. (below) Linux systems on the top500.



What the Linux Kernel Does

Processes: Process Management & Scheduling

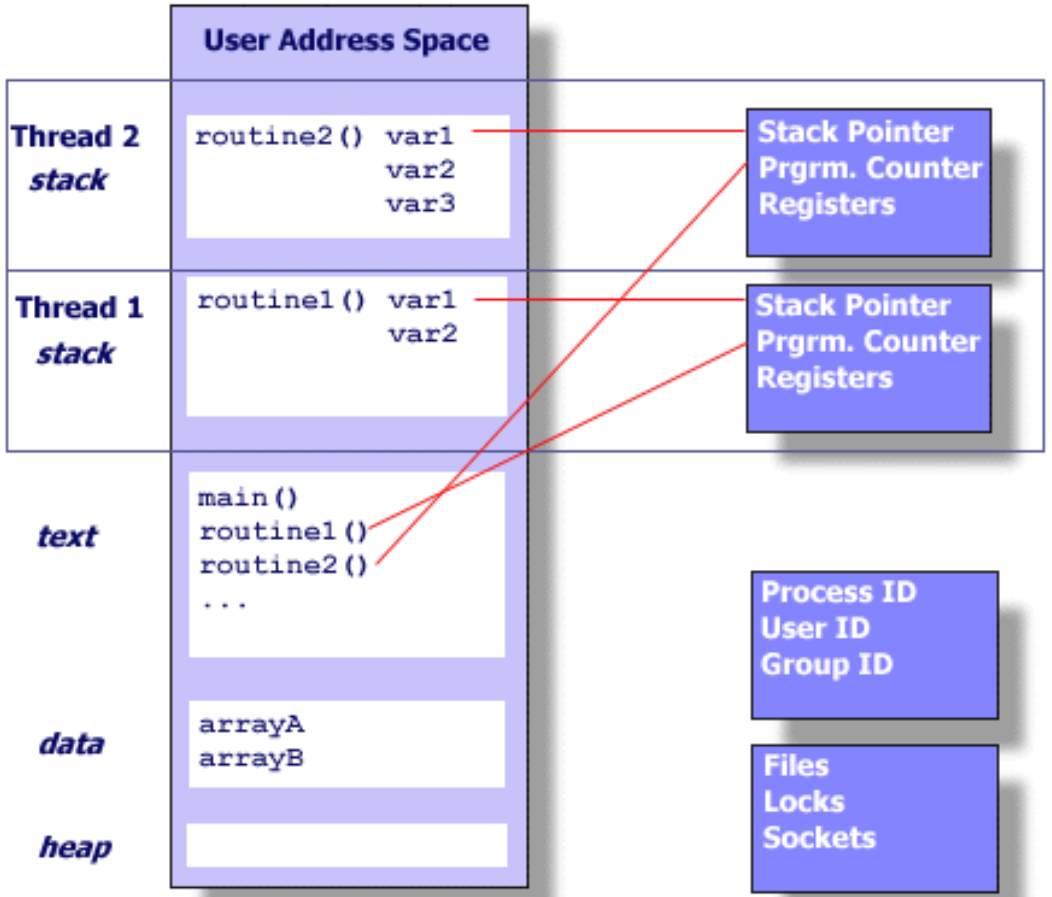
- The Linux kernel is responsible for process scheduling: what, when and how executable code runs on the CPU(s)
- It does this by allocating a process to a program. Processes have:
  - Their allocated memory (stack + heap)
  - Various process metadata (process ID, cmd name, usage statistics, ...)



https://www.cs.montana.edu/~chandrima.sarkar/AdvancedOS

Processes: Processes & Threads

- A process is further divided into threads. A process may have one or many threads active at any time.
- Threads can run concurrently with each other
- Threads can see each others' memory areas
- Threads are lighter weight than a full process
- Threads can be used to implement parallel algorithms



https://computing.llnl.gov/tutorials/pthreads/

API: What is a syscall?

- A system call (syscall) is a special software-triggered system interrupt
- It allows a program to make an API request to the kernel
- It transitions the system from user-space to kernel-space
- Lots of things are syscalls, e.g. file operations (open, close, read, write)

```
;example inspired by:
;   https://taufanlubis.wordpress.com
section .text
gobal _start
_start:

    ;display a message
    mov eax, 4      ;syscall for write is 4
    mov ebx, 1      ;set ebx to a file descriptor
                    ;   stdin=0, stdout=1, stderr=2
    mov ecx,msg     ;set ecx to a pointer to our string
    mov edx,len     ;set edx to the length of the string
    int 0x80        ;call interupt 0x80 (syscall)
                    ;note: modern systems use "syscall"
                    ;       instead of int 0x80
    mov eax, 1      ;syscall for exit prgram is 1
    int 0x80        ;call interupt 0x80 (syscall)

section .data
    msg db 'This is a test.',0xA,0xD
    len equ $ - msg
```

API: Signals

- Signals are in some ways like syscalls from the kernel to the program
- The kernel can send signals to applications
  - but some signals just, e.g. kill the process
- These can be done by the kernel itself (e.g. for program faults)
- …or can be requested through a syscall (like the kill command)
- Signals can be used to do a lot of things:
  - Abruptly terminate a program (SIGKILL)
  - Communicate between threads (“real-time” signals)
  - Or other program defined functions…

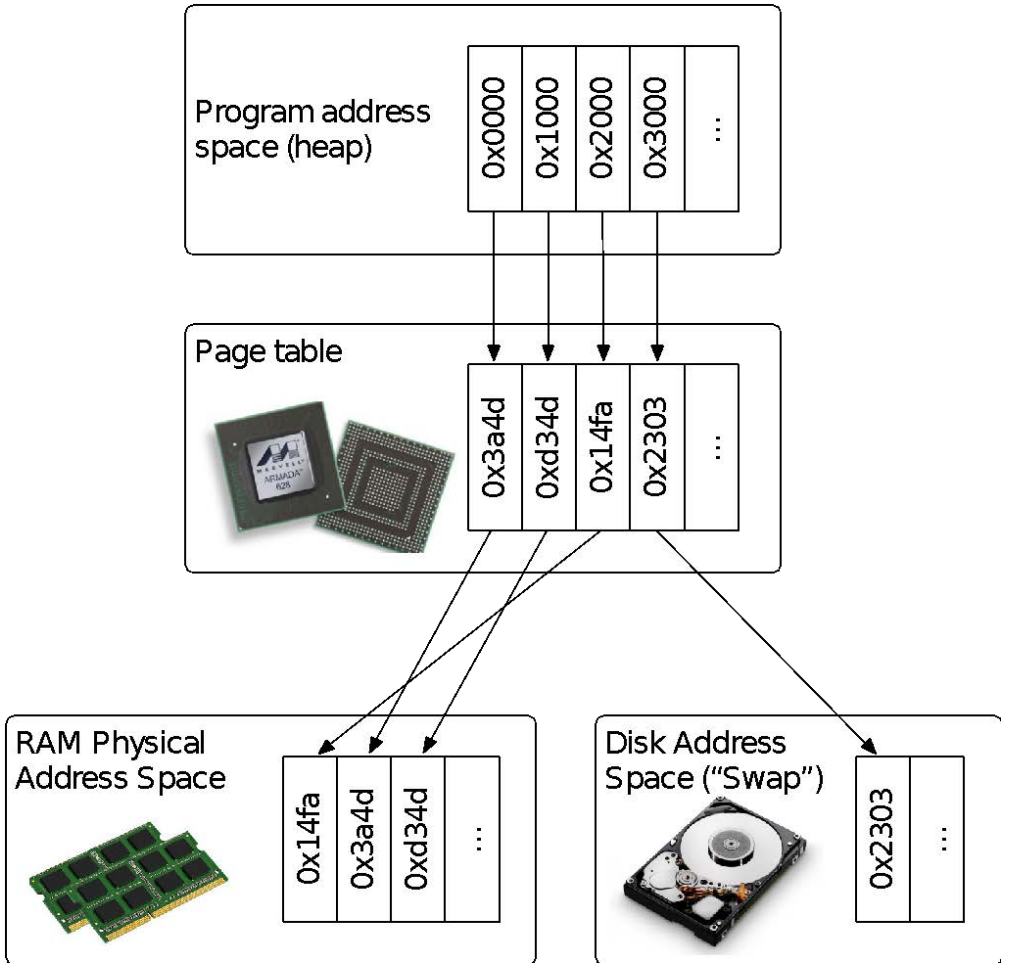
Signal	Value	Action	Comment
SIGHUP	1	Term	Hangup detected on controlling terminal or death of controlling process
SIGINT	2	Term	Interrupt from keyboard
SIGQUIT	3	Core	Quit from keyboard
SIGILL	4	Core	Illegal Instruction
SIGABRT	6	Core	Abort signal from **abort**(3)
SIGFPE	8	Core	Floating point exception
SIGKILL	9	Term	Kill signal
SIGSEGV	11	Core	Invalid memory reference
SIGPIPE	13	Term	Broken pipe: write to pipe with no readers
SIGALRM	14	Term	Timer signal from **alarm**(2)
SIGTERM	15	Term	Termination signal
SIGUSR1	30,10,16	Term	User-defined signal 1
SIGUSR2	31,12,17	Term	User-defined signal 2
SIGCHLD	20,17,18	Ign	Child stopped or terminated
SIGCONT	19,18,25	Cont	Continue if stopped
SIGSTOP	17,19,23	Stop	Stop process
SIGTSTP	18,20,24	Stop	Stop typed at terminal
SIGTTIN	21,21,26	Stop	Terminal input for background process
SIGTTOU	22,22,27	Stop	Terminal output for background process

The signals **SIGKILL** and **SIGSTOP** cannot be caught, blocked, or ignored.

Taken from `man 7 signal`, showing POSIX signals.

Memory: Virtual Memory Architecture

- Memory (RAM) divided into fixed-size “pages”, and given numeric addresses
- The kernel maintains a mapping between virtual memory addresses and real, physical memory addresses
- The CPU helps with this process by using a Memory Management Unit (MMU), and maintaining a hardware-accelerated mapping (TLB)

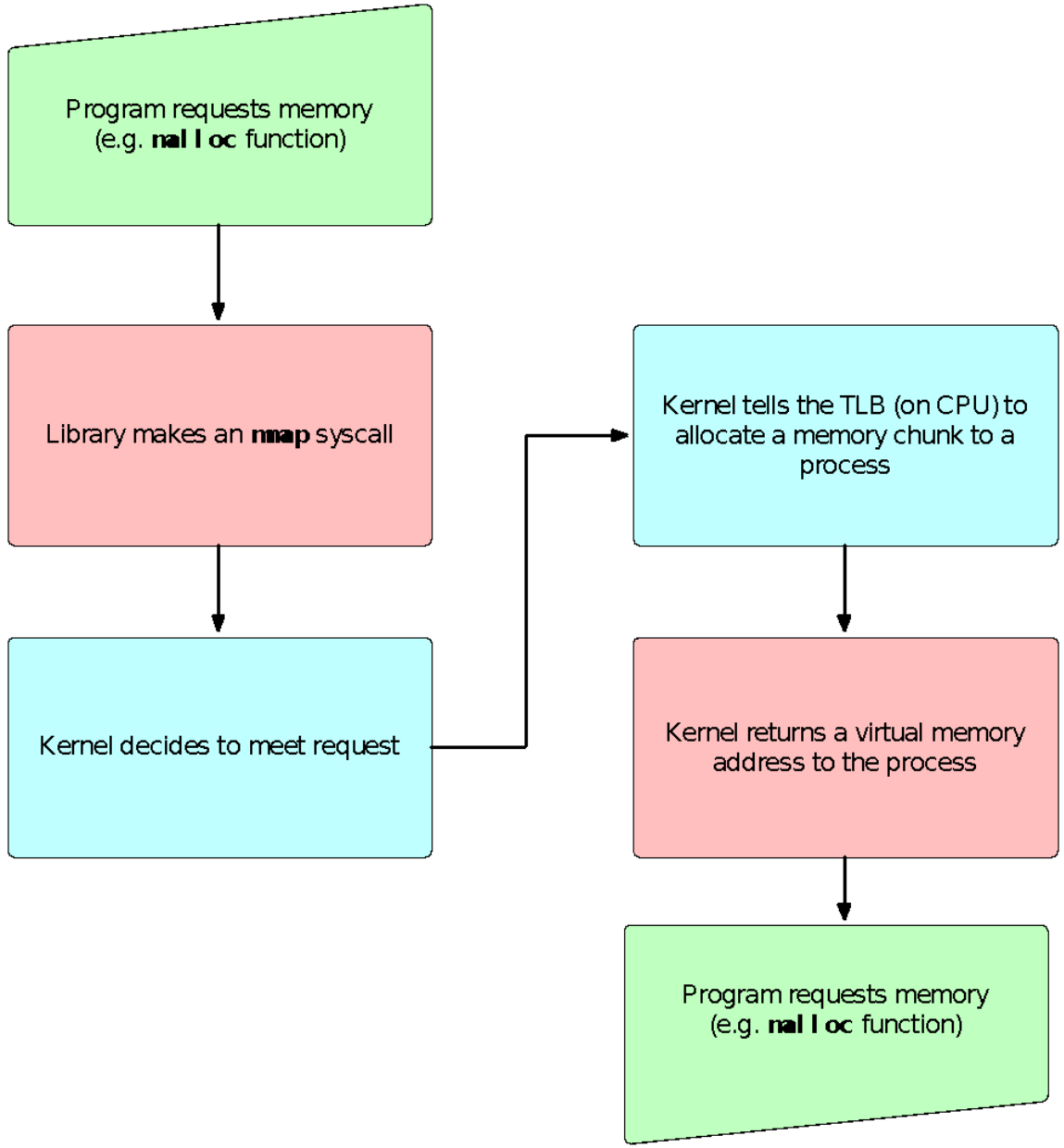


Memory: Why use virtual memory?

- The kernel controls which programs see which memory regions
- One program can’t alter another program’s memory (very important for security)
- Can create non-RAM memory areas: “swap” space on disk, memory areas on other devices
- Control the amount of memory a process can use
- User programs don’t need to know about memory hardware

Memory: Allocation process

- Program tells the kernel it needs new memory through a syscall (typically mmap)
- Kernel decides if it will allocate memory
- Kernel tells the CPU's MMU it wants to allocate a chunk of memory
- MMU creates a TLB entry mapping physical to virtual memory
- Kernel gives the program a virtual address it can use



Authorization: Users & Groups

- The Linux kernel provides an *authorization* layer for processes and system objects (e.g. files)
- The basic mechanism is provided through users/groups
  - A ***user*** is an authorization domain that can be tied to a person or processes.
  - A ***group*** is an authorization domain consisting of a collection of users.
  - A special user, ***root***, is (mostly) unconstrained.
- A processes runs in a user/group domain
- System objects have access permissions based on one user/group that decided if a process can perform an action.

Character: `-wrswrx-rx`

Octal: `4775`

	Special			User			Group			Other		
	su	sg	sv	w	r	x	w	r	x	w	r	x
Bitmask:	1	0	0	1	1	1	1	1	1	0	1	1

These are representations of object ACLs

- **su** – setuid (run as specific user)
- **sg** – setgid (run as specific group)
- **sv** – sticky (means different things)
- **w** = **W**rite, **r** = **R**ead, **x** = e**X**ecute
  - For directories: execute means “can enter”

Authorization: SELinux

- SELinux (“Security-Enhanced Linux”) was developed by the NSA
  - Open source release: 12/22/2000
- Extends the user/group model:
  - All processes and system objects have a *context*
    - E.g. files have SELinux context
  - A (possibly very complicated) *policy* determines what can happen in what contexts
    - Compiles to binary
    - Can be extended with modues
  - Enforces that both context and policy are specified, and policy is obeyed
- *While SELinux is great, we’ll be disabling it since it makes development harder.*

(right) Example of an selinux policy module.
(below) Inspecting the status of current selinux contexts.

```
module mysemanage 1.0;

require {
        class fd use;
        type init_t;
        type semanage_t;
        role system_r;
};

allow semanage_t init_t:fd use;
```

```
~]# sestatus -v
SELinux status:                 enabled
SELinuxfs mount:                /selinux
Current mode:                   enforcing
Mode from config file:          enforcing
Policy version:                 21
Policy from config file:        targeted

Process contexts:
Current context:                user_u:system_r:unconfined_t
Init context:                   system_u:system_r:init_t
/sbin/mingetty                  system_u:system_r:getty_t
/usr/sbin/sshd                  system_u:system_r:unconfined_t:s0-s0:c0.c1023

File contexts:
Controlling term:               user_u:object_r:devpts_t
/etc/passwd                     system_u:object_r:etc_t
/etc/shadow                     system_u:object_r:shadow_t
/bin/bash                       system_u:object_r:shell_exec_t
/bin/login                      system_u:object_r:login_exec_t
/bin/sh                         system_u:object_r:bin_t ->
system_u:object_r:shell_exec_t
/sbin/agetty                    system_u:object_r:getty_exec_t
/sbin/init                      system_u:object_r:init_exec_t
/sbin/mingetty                  system_u:object_r:getty_exec_t
/usr/sbin/sshd                  system_u:object_r:sshd_exec_t
/lib/libc.so.6                  system_u:object_r:lib_t ->
system_u:object_r:lib_t
/lib/ld-linux.so.2              system_u:object_r:lib_t ->
system_u:object_r:ld_so_t
```

Monolithic vs. Modular

- Since version 1.2 (1995), Linux supports “modules”
- Modules can add functionality to a running Linux kernel
  - Can implement new kinds of syscalls
  - Can support new hardware
  - Can add new kinds of functionality (e.g. firewall features)
- Modern Linux Distributions tend to heavily use modules, moving even essential functionality into modules. These modules must be loaded before the system is fully functional. Common examples:
  - Network card drivers (including high-speed networks like InfiniBand and OPA)
  - Filesystem, storage and software RAID drivers
  - Extra functionality for the IP Firewall system (iptables/nftables), cryptography
  - Video, sound, USB devices, …

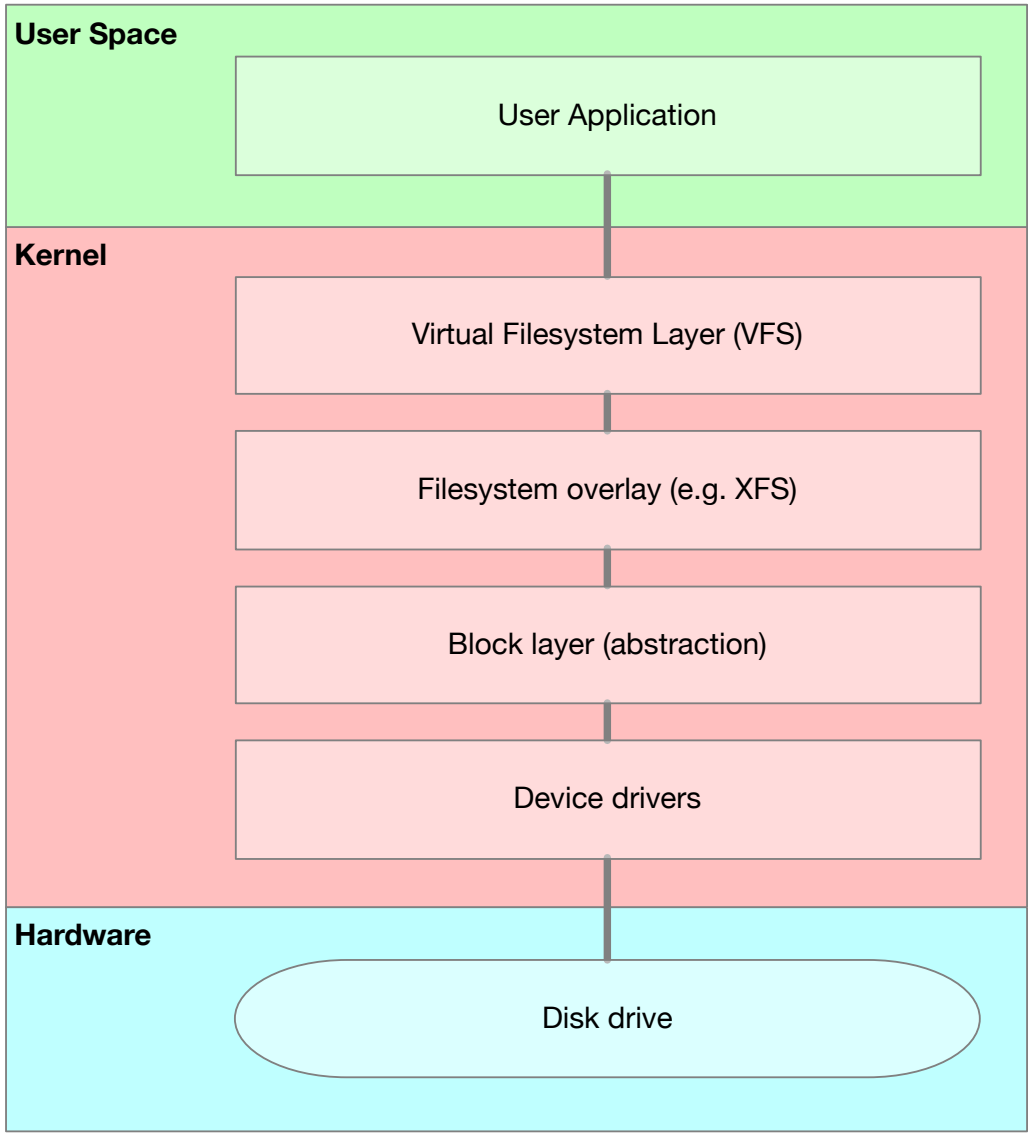
Device Drivers

- The kernel can provide interfaces for working with hardware in the system (drivers)
- Generally, this will provide mechanisms for talking to the device through the kernel APIs
  - Through syscalls
  - Through ioctl calls on a file (e.g. in /dev)
- Will provide handlers for dealing with hardware events
- Often, the device will be presented to the user through a common abstraction layer
  - E.g. storage block layer, network socket layer, …
- In modern systems, device drivers are usually loaded as kernel modules

A couple of examples

Example: Accessing storage

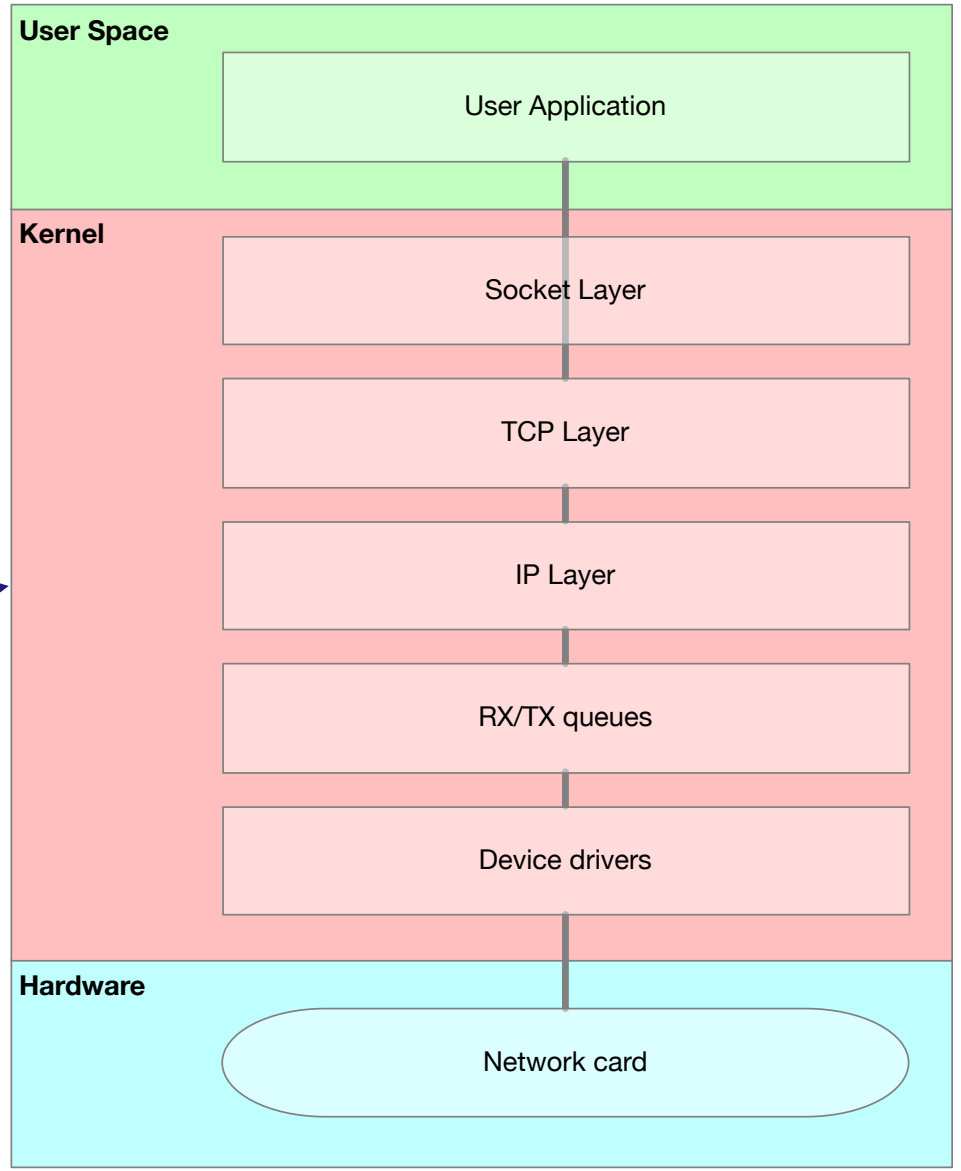
1. The user application accesses a file through a syscall (say *open* or *write*).
2. This syscall is caught by the VFS layer that abstracts filesystem access.
3. The VFS layer passes to the Filesystem layer that maps the operation to storage blocks in the specific filesystem.
4. The block layer maps these blocks to particular hardware addresses.
5. The device drivers determine how to make these changes happen on a physical disk.



Example: reading an HTTP GET request

1. The network card receives a string of high/low voltages it interprets as data
2. The network card sends a hardware interrupt (IRQ)
3. The IRQ is caught by a kernel handler for IRQs that maps to a handler in the network card device driver
4. The device driver reads in the data
5. The driver hands the data to the rx/tx queue
6. The IP layer reads the IP header; the TCP layer reads the TCP header
7. The TCP layer identifies (by port and address) and identifies a listening socket
8. The socket layer provides the data to the socket
9. The listening web server reads (syscall) the data
10. It reads “GET” and sends some data back down the channel to the client

Lots of filtering (e.g. iptables/nft) happens here!



Linux Distributions

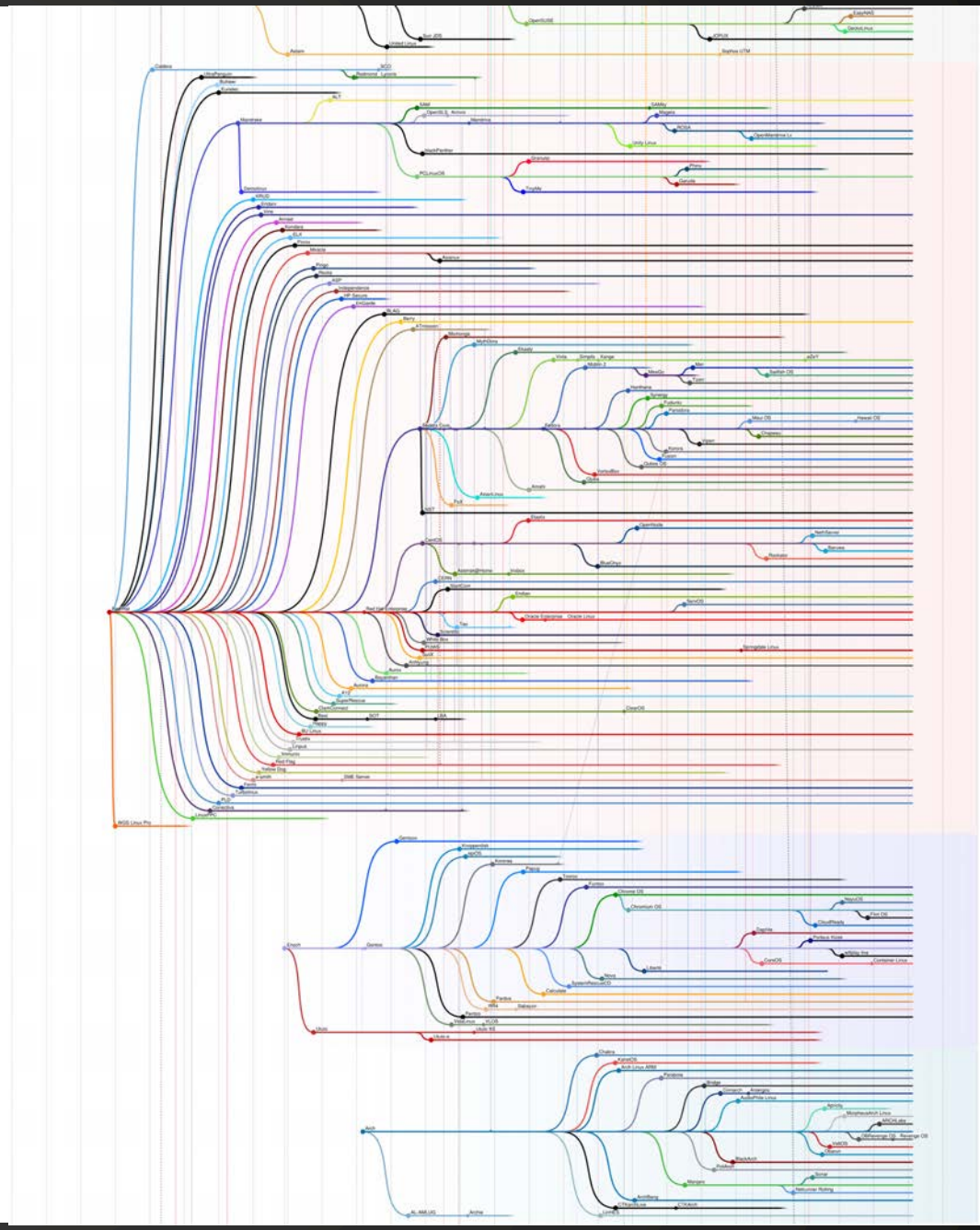
What makes a Linux Distribution?

There are many combinations of software that can build a usable Linux system.
A Linux distribution is a particular usable arrangement of these tools.

Must Have	Typically Has
• One (or more) provided Linux Kernel(s) • One (or more) system initialization system (systemd, sysV, supervisord, ...) • A collection of userspace software for using/managing the system	• Easy installation system • Some form of software installation management (e.g. rpm, yum, apt, dpkg, yast, portage) • Collections of useful software for users (e.g. firefox, libreoffice, GNOME, ...) • Software development tools

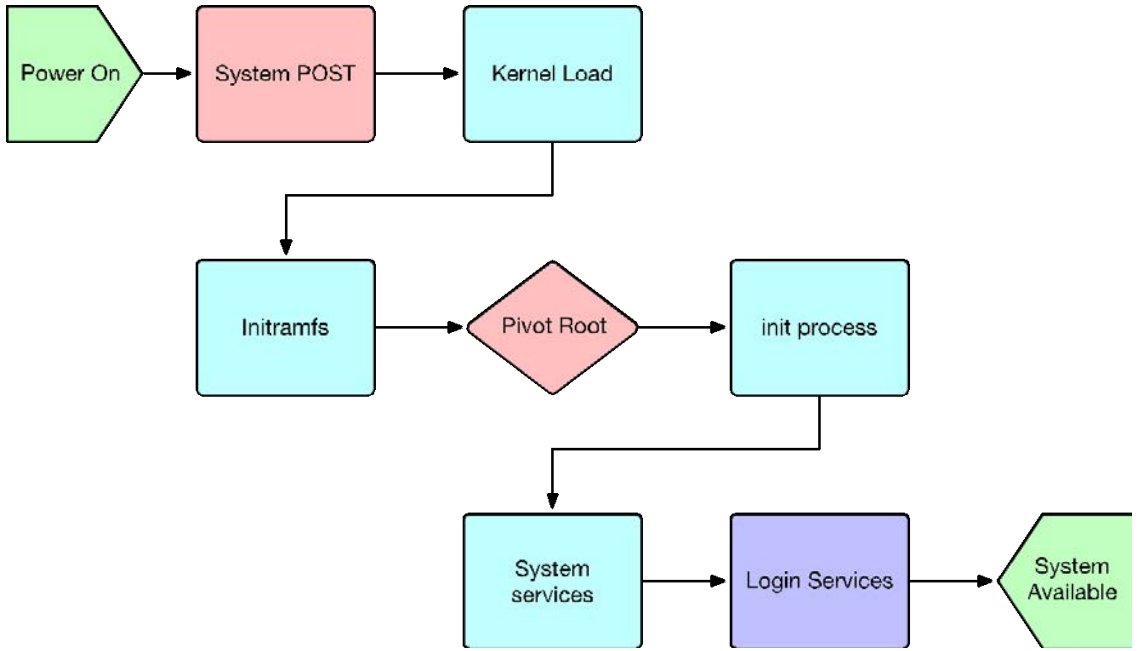
There are many...

- A site that tracks distributions claims there have been as many as 323 distributions at a given time.
- Many distributions are branches from one of the original major distributions:
  - Red Hat (CentOS, Fedora,...)
  - Debian (Ubuntu, Mint,...)
  - SUSE
  - SLS/Slackware
- And some major software like Android (non-GNU) and Chromium (based on Gentoo)



The Linux system boot process

What happens when you hit the power button?



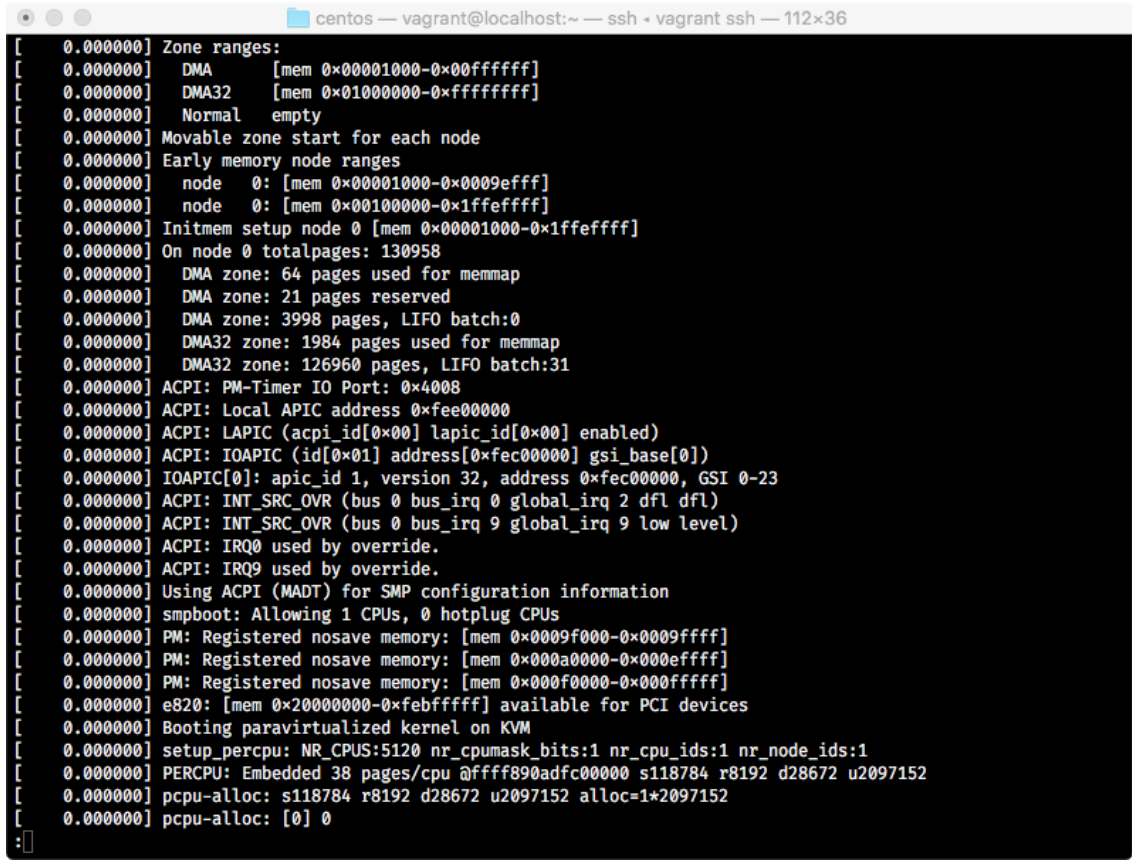
- System Power-on Self Test (POST)
- Kernel entry point
- Kernel loads initramfs (optional)
- Initramfs sets up sytem (e.g. loads modules)
- Initramfs calls pivot_root (or switch_root) to switch to real root filesystem
- pivot_root calls the new init process (e.g. systemd)
- RC system (e.g. systemd) starts services
- Login services (TUI and/or GUI) loaded

System POST

- POST = Power-on Self Test
- With UEFI (Unified Extensible Firmware Interface), this stage does a lot more than basic tests, and can take a while
  - Loads pluggable DXE (“dixie”) modules to do various system init tasks
- Does some really important stuff, like figure out timings needed to efficiently use RAM
- Loads the requested operating system (in our case, the Linux Kernel)

Kernel init

- The kernel provides an executable entry point for the system
- It takes control of things like:
  - Switching to “protected mode” (memory protection)
  - Registering memory
  - Registering interrupt handlers
  - …and lots of other things
- When it’s done getting everything ready, it extracts the initramfs filesystem and treats at is the filesystem “root” (/)
- Then, it attempts execute the “/sbin/init” command (or other if specified)



Initramfs

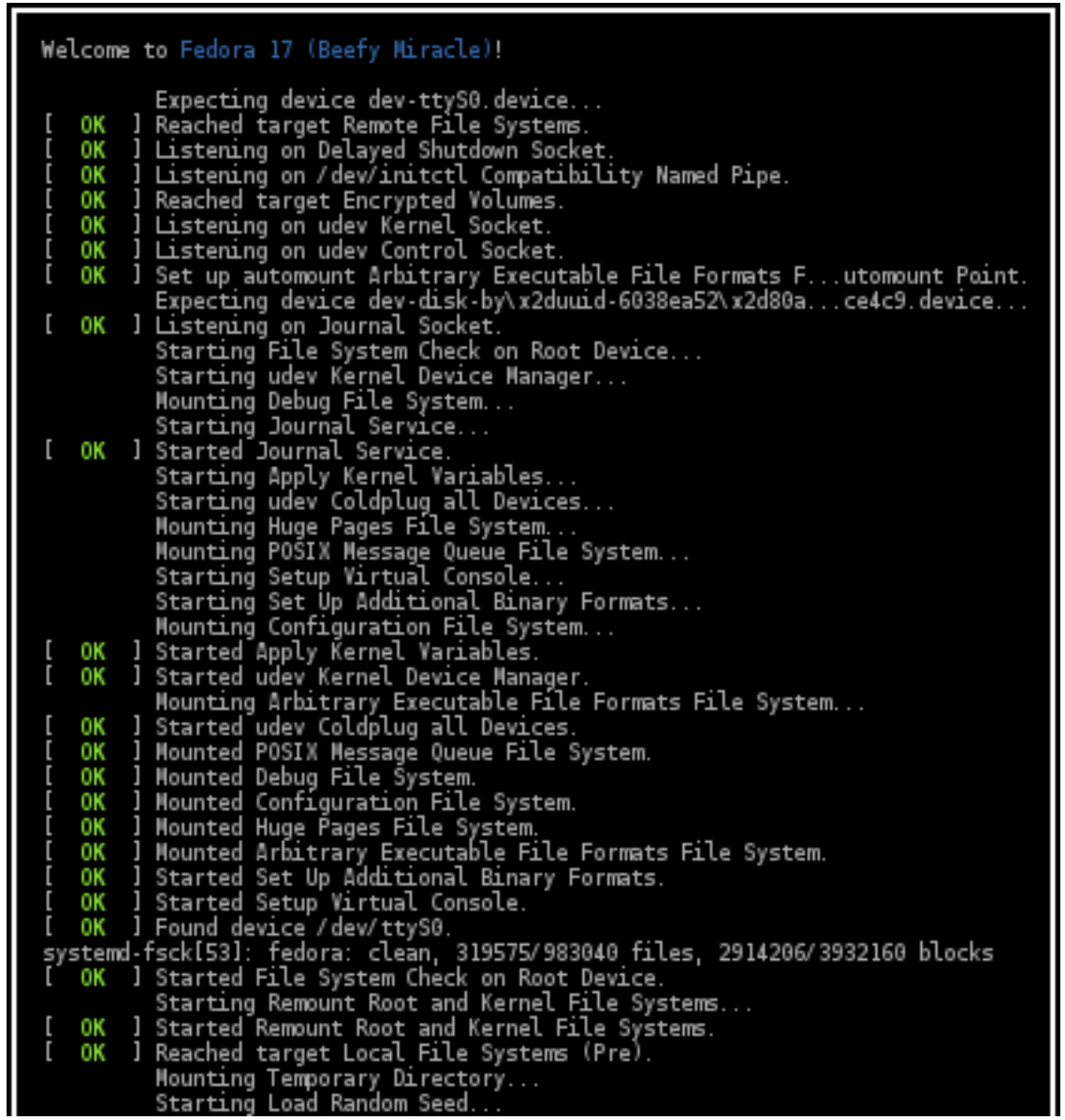
- The intramfs is a specially file archive that gets to temporarily act as the root filesystem.
- It's in the "cpio" archive format, possibly compressed.
- It can come from many places, like the "/boot" directory, or over the network.
- It gives us a chance to do any pre-setup we need (like getting the real root filesystem).
- When it's ready, it (typically) will "switch_root" into the new root filesystem and execute the real /sbin/init process.

initramfs — vim init — 77×33

```
#!/bin/sh

# mount our basic filesystems
mount -t devtmpfs devtmpfs /dev
mount -t proc proc /proc
mount -t sysfs sysfs /sys
mkdir -p /dev/pts
mount -t devpts devpts /dev/pts

# load modules
for m in $(cat modules.txt); do
        insmod lib/modules/${m}.ko.xz
done

# bring up network
ip link set up eth0
udhcpc -s /net.sh -i eth0

# load the system image into ramfs
sh load_img.sh

# shift mounts to the new image
mount --move /sys /mnt/root/sys
mount --move /dev /mnt/root/dev
mount --move /proc /mnt/root/proc

# switch to the full image
cd /mnt/root
exec switch_root /mnt/root /init

# fallback to a shell (shouldn't ever happen)
sh shell.sh
```

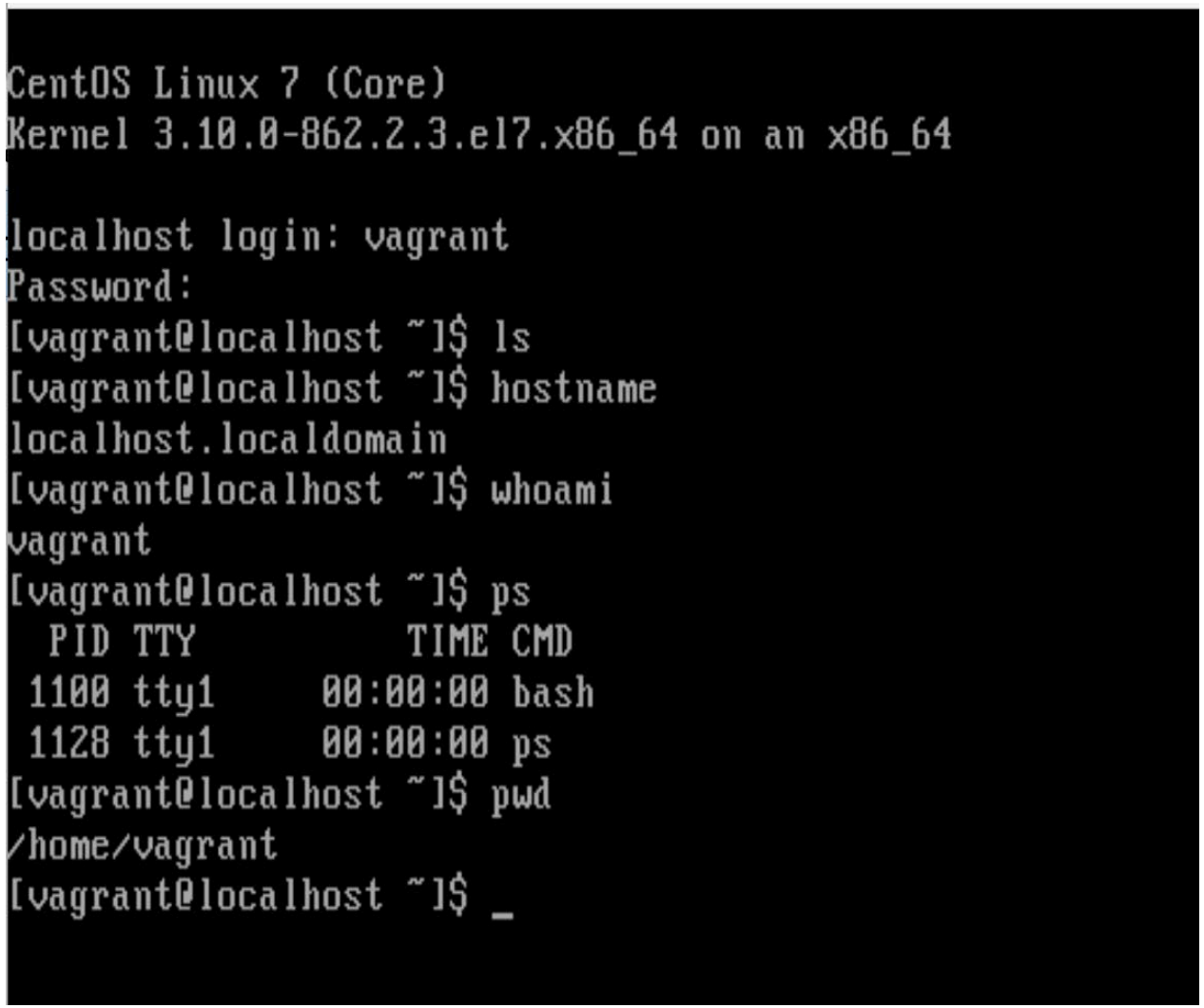
The init process

- The init process is responsible for doing any non-kernel level initialization, like starting system services
- The most common init process these days is “systemd”, a complete initialization and service management system (and more…)
  - We’ll be learning more about systemd and system services later on.
- One of the last services started is a login service that provides a login prompt to the user



Login & shell

- Once the user logs in, a new process is started (“the shell”) with that user’s running context
- The shell can technically be anything, but typically it’s an interactive tool for using the system (like BASH)
- A shell allows you to do things like:
  - Run more processes to do useful things
    - …and see their output
  - Keep track of certain session variables (“environment variables”)
  - Redirect input and output in useful ways
  - Use a simple scripting language to string these tasks together (“shell script”)
  - *…and generally a lot more.*

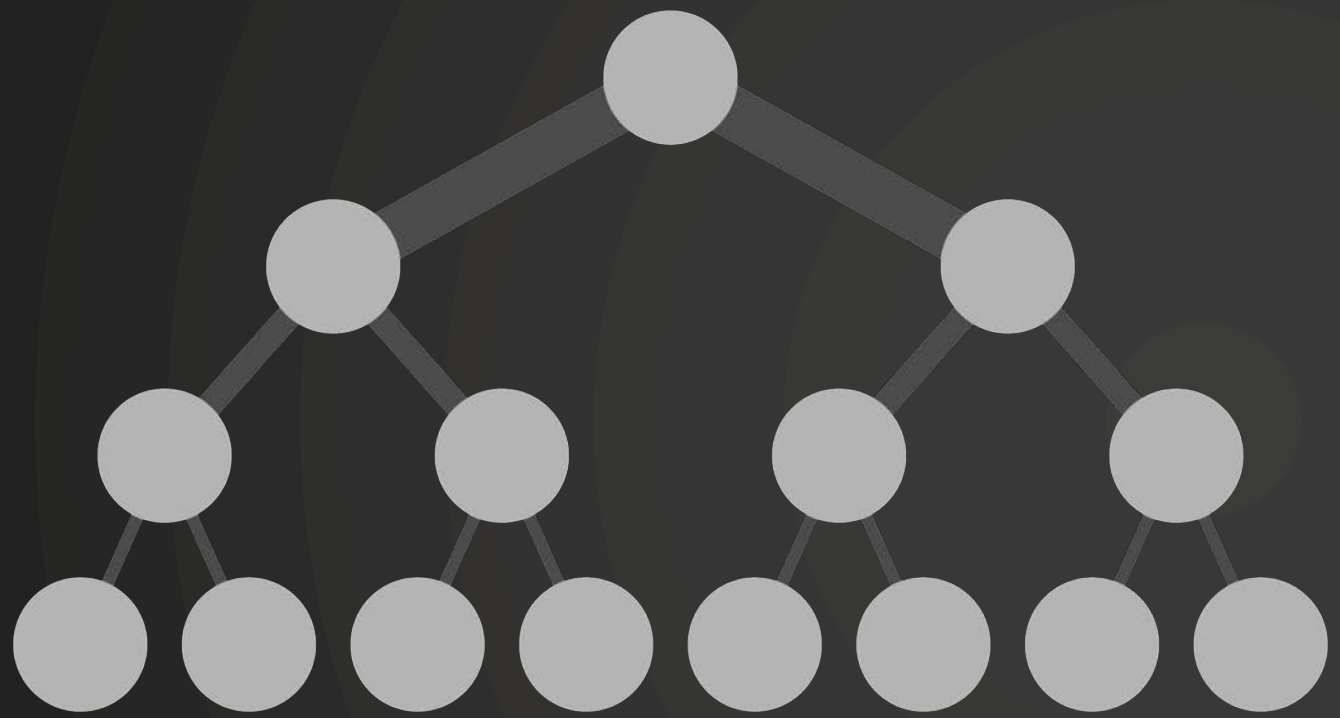


Questions?

Los Alamos

NATIONAL LABORATORY

EST. 1943

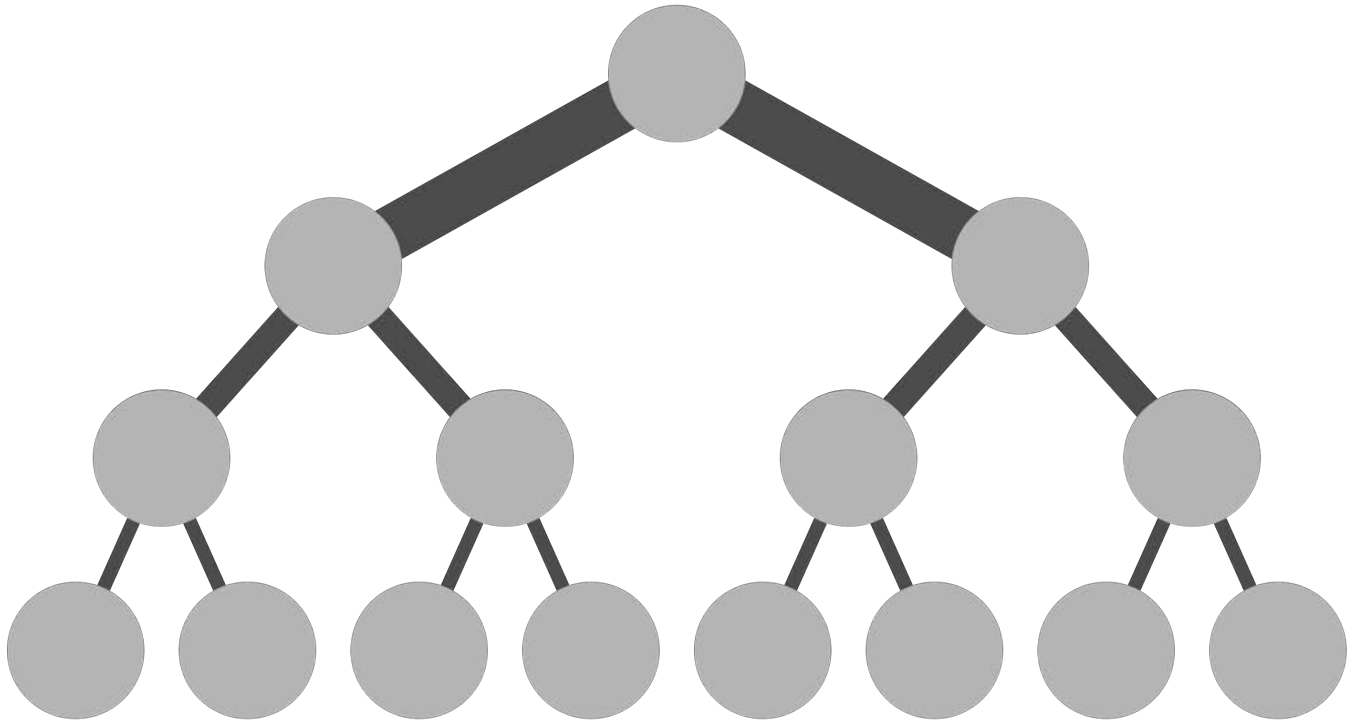


Los Alamos
NATIONAL LABORATORY
EST. 1943

Delivering science and technology
to protect our nation
and promote world stability

HPC Networking and Services

Presented by CSCNSI



Agenda

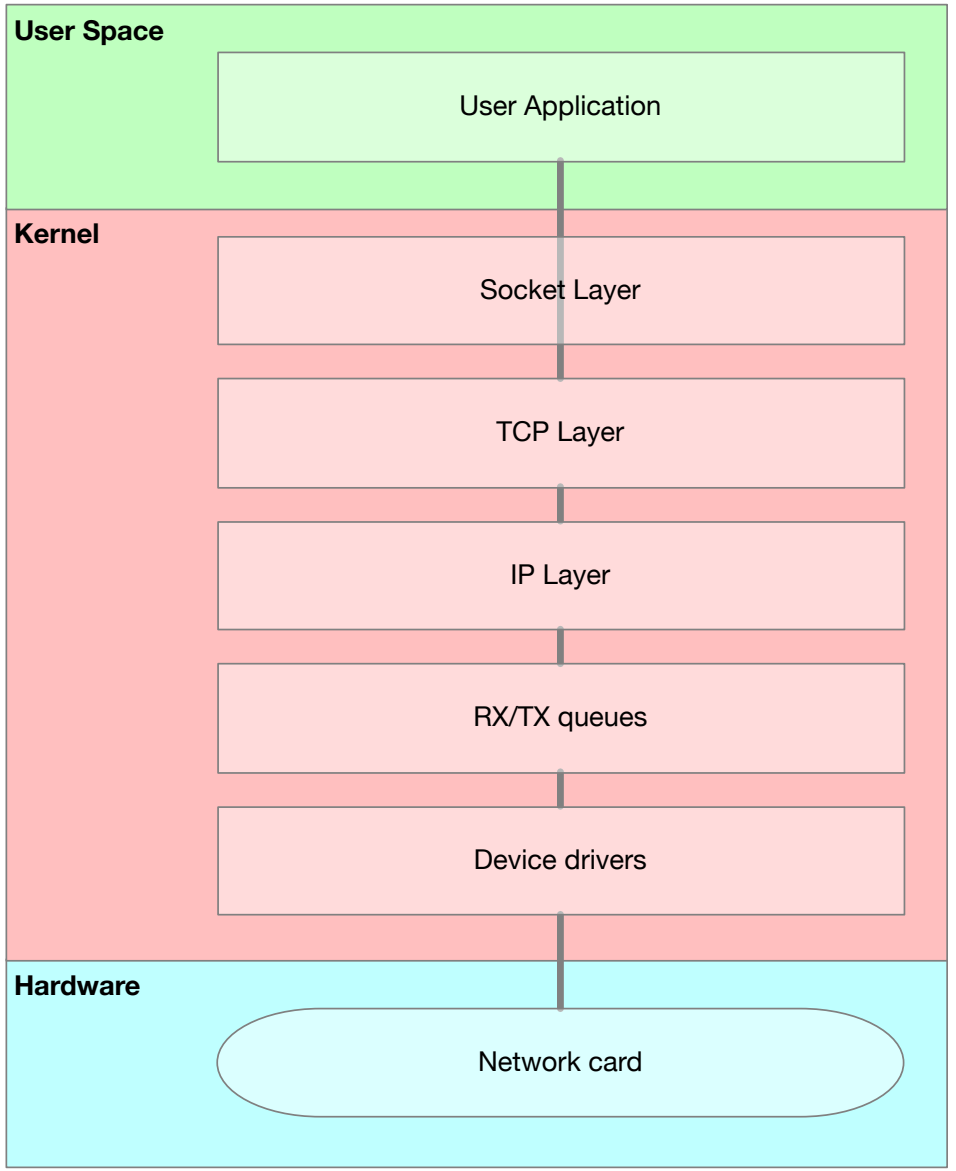
- Networking Overview
- Services

Networking Overview

- Computers need to talk to each other. They do this over networks made up of communication hardware
- HPC clusters frequently have multiple networks
  - High Speed Network (HSN): special low-latency, high speed network for job communication
  - BMC network: Standard management network for baseboard management controllers (BMCs)
  - Cluster network: Standard network used for general node-to-node communication (ssh between nodes; communication with scheduler; etc.)
- This section looks at the standard networks; HSNs will be covered later
- Your cluster uses 1000Mbit ethernet as its cluster network

HPC Networking

- Networks are complex
- They may consist of…
  - Multiple hardware types
    - (wireless, wired, optical, …)
  - Multiple vendors
    - (Linksys, Arista, Broadcom, Unifi, …)
  - Multiple protocols
    - (HTTP, Bittorrent, SSH, …)
  - Multiple speeds
    - (54Mbit, 100Mbit, 1000Mbit, …)
- These details are hidden beneath libraries, standards, and layers of abstraction



Networking Theory

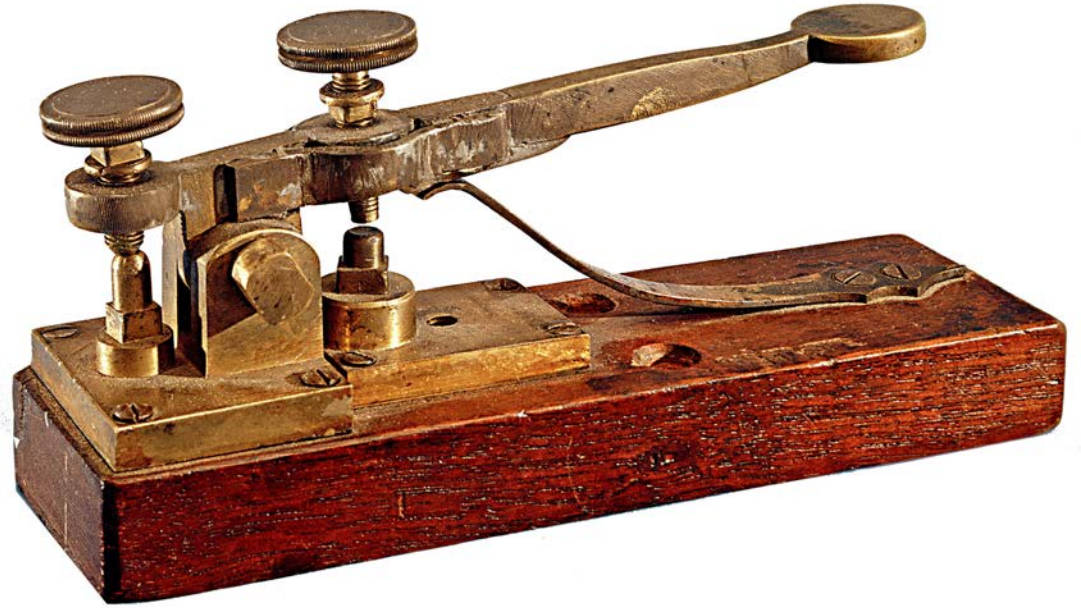
- The seven-layer Open Systems Interconnection (OSI) model
  - Logically splits an application's view of a network into interoperable pieces
  - In theory, any layer can be replaced without affecting the others
  - In reality, some layers are combined and some have complex interactions depending on the network implementation

7	Application
6	Presentation
5	Session
4	Transport
3	Network
2	Data Link
1	Physical

- *For now, we're concerned about layers 1 through 4.*

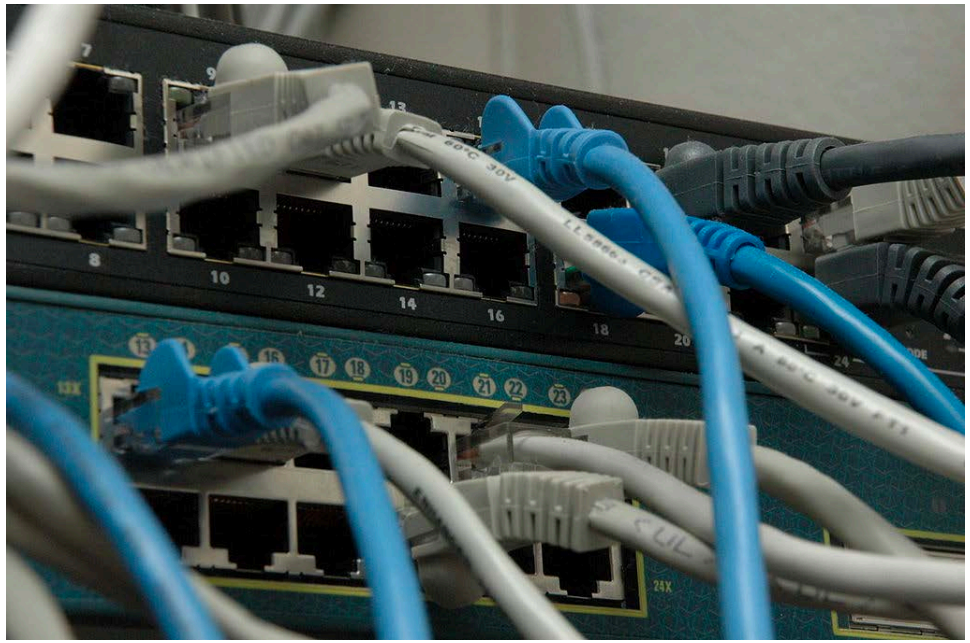
Layer 1: The Physical Layer

- The most basic communication layer
- Enough for one device to send a signal to another
  - Electrical signals flowing over copper wires
  - Light signals transmitted via fiberoptic cables
  - Wireless signals transmitted via radio waves
- Just a series of ones and zeros
- Bits are merely passed from one device to the next. No interpretation of the bits is performed at this level.

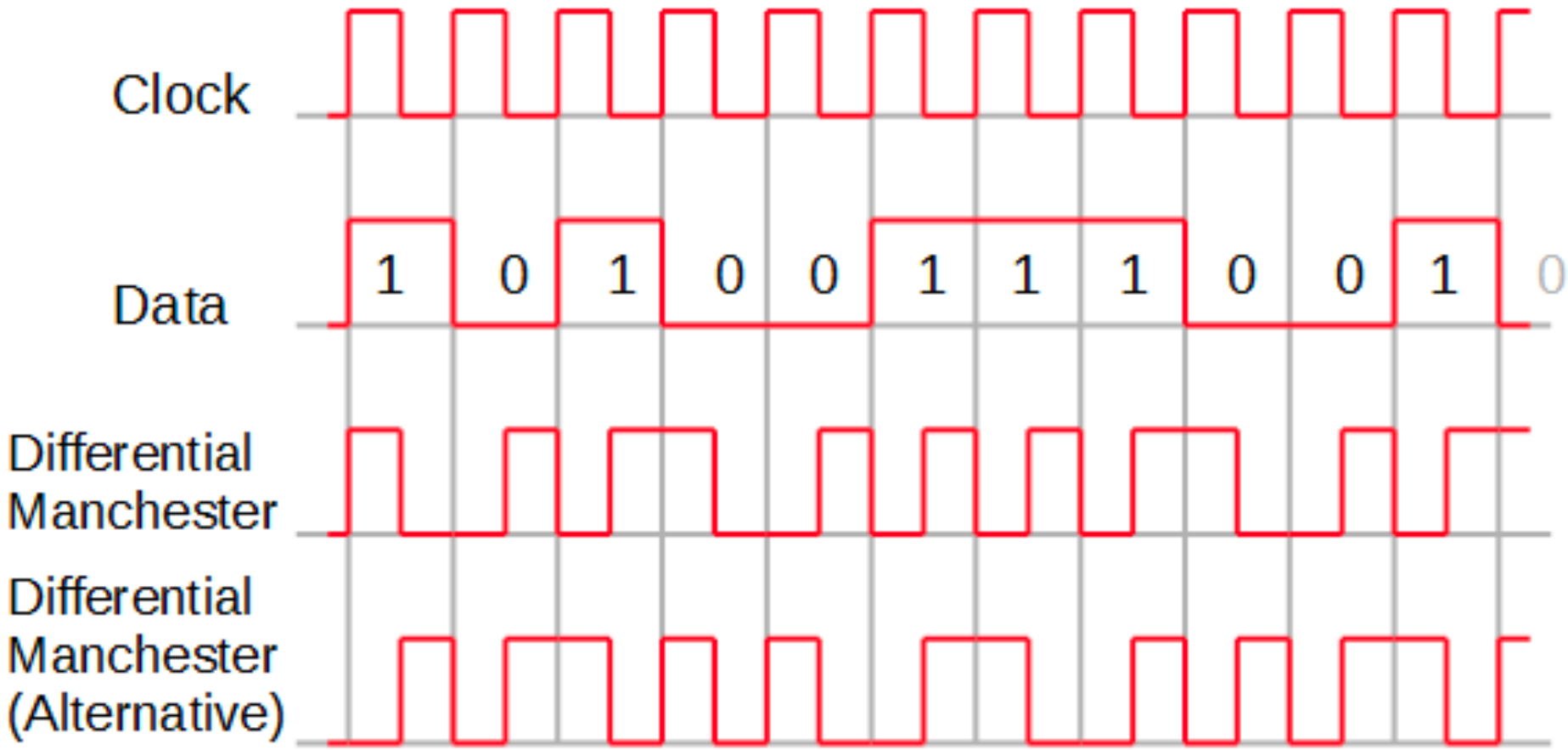


A Layer 1 Interface: Twisted Pair

- An extremely common physical network
- Cables consist of four pairs of individual copper wires that are twisted together
- Ends consist of RJ45 connectors, which look like big phone connectors
- Cables plug into hardware ports on each end: one on the computer, one on the switch
- Basic unit of transfer is a single bit, indicated by high or low voltage on the wire

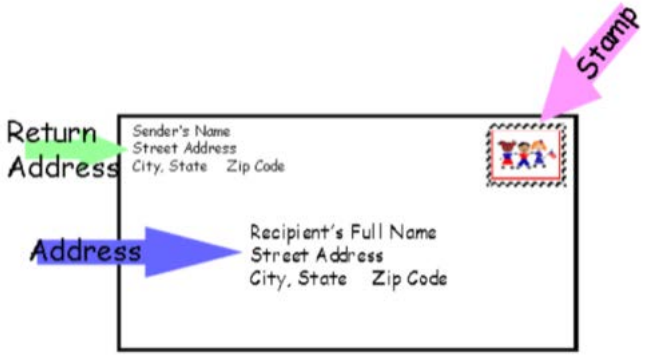


Layer 1 Details: Encoding



https://commons.wikimedia.org/wiki/File:Differential_Manchester_encoding_alternatives.png

Layer 2: The Data Link Layer



- The physical layer is too low-level to use directly
  - It's just high and low voltage on a wire
  - It only works on one wire
- The Data Link layer builds a protocol on top of the Physical Layer for moving information between endpoints
- At this level, the bits can be examined by intermediary hardware to do things like check for errors, look at destination addresses, and provide quality of service (i.e. choose who gets priority when things are busy)

A Layer 2 Protocol: Ethernet

- An extremely common Layer 2 local-area network standard
- Allows one computer to communicate to another, potentially passing through one or more switches on the way
- Basic unit of transfer is a *frame*, which consists of a set of bits
- Every endpoint has a globally unique Medium Access Control (MAC) address
  - 48-bit address in the firmware of every ethernet card
  - Example: EC:F4:BB:C6:23:E4
- Communication involves broadcasting to find what switch hardware port a particular MAC address is on, and then forwarding frames that way

- Efficient for local connections; very inefficient for wide-area networks
  - You can’t broadcast to the entire internet to find a particular MAC address

Layer 2: The Ethernet Frame

	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Preamble																															
4	32	Preamble, continued																								Start of Frame Delimiter							
8	64	Destination MAC Address																															
12	96	Destination MAC Address, continued																Source MAC Address															
16	128	Source MAC Address, continued																															
20	160	802.1Q Tag																															
24	192	Length																Payload															
…	…	Payload, continued																															
X	X	Frame Sequence Check																															

Layer 3: The Network Layer

- The Data Link layer is still too low-level to use directly
  - It would be essentially impossible to find a MAC address on the other side of the world (or even on the other side of town)
- The Network layer introduces network segments that can be routed between
  - MAC addresses can still be found on small local broadcast domains
  - Anything that can't be reached locally gets sent out of the local network

A Layer 3 Protocol: IPv4

- Internet Protocol: A “best effort” protocol built on top of the Data Link layer
  - No guaranteed data delivery; no guaranteed data ordering; possible duplicate data delivery
- Basic unit of transfer is a *packet*, which is encapsulated within a frame
- Packets are sent between *IP addresses*
  - Four 8-bit octets: 0.0.0.0 to 255.255.255.255
  - Unlike MAC addresses, IP addresses can be used to route packets over long distances
- IP Addresses are hierarchical, which helps the routing process
  - 192.x.x.x
  - 192.168.x.x
  - 192.168.1.x
- For scalability, addresses are grouped into *subnets*

Layer 3 Details: IPv4 Subnets

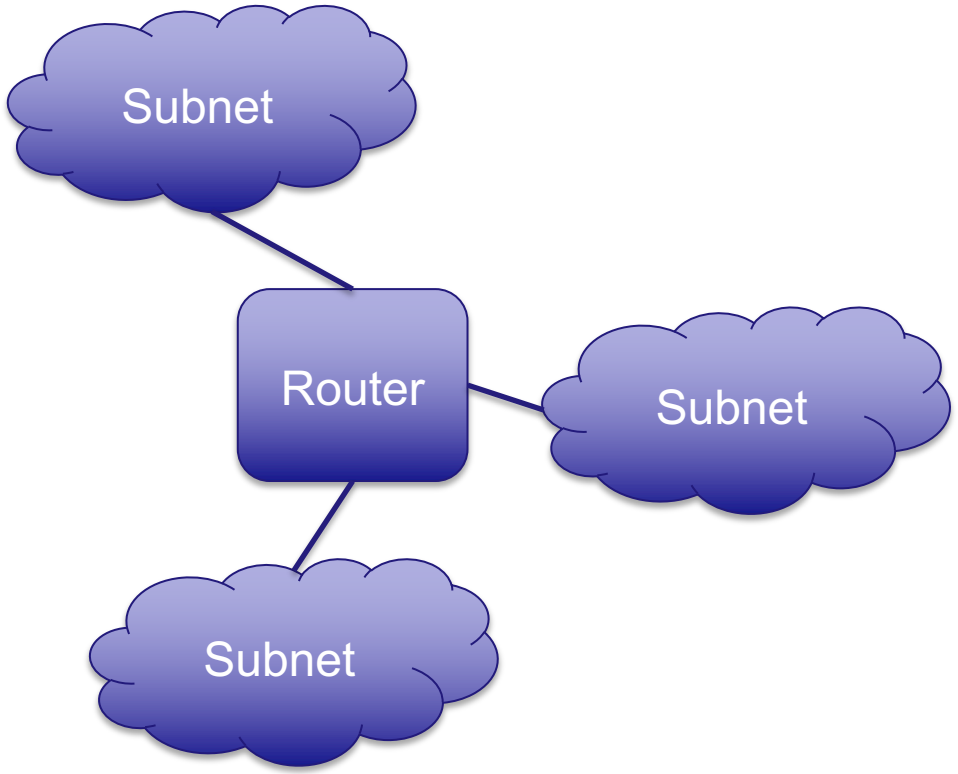
- Logical groupings of addresses that make networking more efficient
  - Within a subnet, local hardware and software handles packet delivery
  - Between subnets, routing needs to happen
- Defined by an address (192.168.0.0) and a mask (255.255.0.0)
  - Alternate format: 192.168.0.0/16
- All addresses on a subnet have the same routing prefix
  - 192.168.1.1, 192.168.1.2, 192.168.2.10 are all on 192.168.0.0/16
  - 10.0.0.1, 192.168.1.1, 172.16.2.1 are all on different subnets
- All subnets have a *broadcast address*
  - Packets sent to this special address get sent to all other addresses on the subnet
  - Implemented as the last address in a subnet
    - Examples: 10.255.255.255, 192.168.255.255

Layer 3: The IPv4 Header

	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Version				IHL				DSCP						ECN		Total Length															
4	32	Identification																Flags				Fragment Offset											
8	64	Time to Live								Protocol								Header Checksum															
12	96	Source IP Address																															
16	128	Destination IP Address																															
20	160	Options																															
24	192																																
28	226																																
32	256																																

IPv4 Routing

- IPv4 traffic can address the whole world by introducing *routers*
- Routers keep track tables of how to get to different subnets
  - A special entry in the table, the *default,* or *0.0.0.0/0*, entry becomes a bucket for any address that isn't specified by the table
- Building and maintaining routing tables can be complicated
  - Another example of a Layer 3 protocol is OSPF, a protocol used to build these tables dynamically



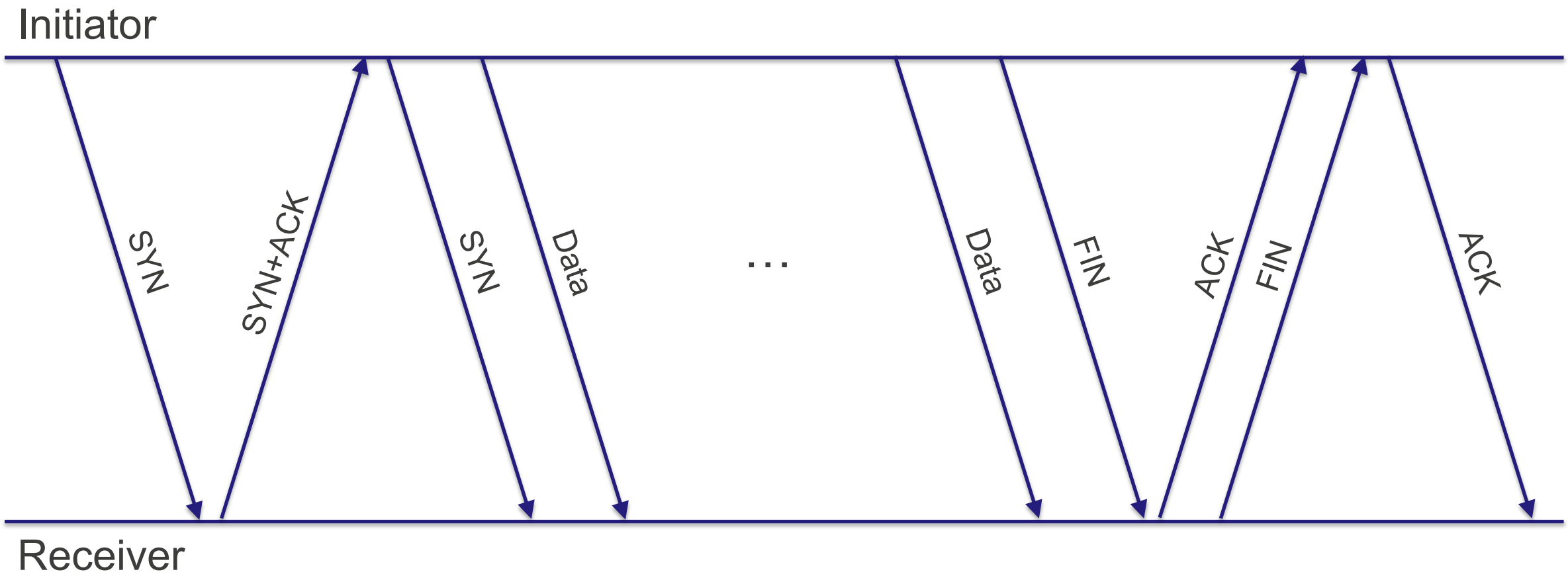
Layer 4: The Transport Layer

- The Network layer is *still* too low-level to use directly
  - Only one application can use it at a time
    - …how would we keep track of more than one address on one machine?
  - It is only best-effort
    - …we have no guarantee that data is delivered.
- The Transport layer adds the remaining functionality that applications need
  - Persistent connections
  - Multiplexing
  - … and others

A Layer 4 Protocol: TCP

- Transmission Control Protocol: a robust protocol built on top of the Network layer
  - Guarantees packet delivery, guarantees packets are received in order, guarantees at-most-once delivery of packets
  - This adds overhead, but also provides very useful functionality
- Basic unit of transfer is a *segment*, which is encapsulated within a packet
- TCP introduces *software ports*
  - Numbered between 1 and 65535
  - A port can be open by one application at a time
- TCP is connection oriented:
  - One computer connects to another via a unique IP address and port combination
  - This connection stays open until explicitly closed down (or until it times out)

Layer 4 Details: TCP Connection Lifecycle



Layer 4: The TCP Header

	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Source Port																Destination Port															
4	32	Sequence Number																															
8	64	Acknowledgement Number																															
12	96	Data Offset				Reserved				Flags								Window Size															
16	128	Checksum																Urgent Pointer															
20	160	Options																															
…	…																																

Another Layer 4 Protocol: UDP

- User Datagram Protocol: a less robust protocol built on top of the Network layer
  - No guaranteed data delivery; no guaranteed data ordering; possible duplicate data delivery
- But! Provides ports and checksums, just like TCP
- Provides some of the benefits of TCP without all of the overhead
- Good for:
  - Applications that can tolerate missing packets
  - Communication that would be dramatically slowed down by the full TCP connection process
  - Protocols that use simple, singleton messages

Layer 4: The UDP Header

	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Source Port																Destination Port															
4	32	Length																Checksum															

Layers 1 Through 4 Working Together

- Each of these protocols encapsulates the data above it
  - An IP packet's data section is an entire TCP segment: header plus data
  - An Ethernet frame's data section is an entire IP packet
- As the data moves through the stack, a layer either adds its header and passes the data up

<table>
<tr><td>TCP Segment</td><td colspan="2"></td><td>Header</td><td>Data</td><td></td></tr>
<tr><td>IP Packet</td><td></td><td>Header</td><td colspan="2">Data</td><td></td></tr>
<tr><td>Ethernet Frame</td><td>Header</td><td colspan="3">Data</td><td>Footer</td></tr>
</table>

Practical Networking

- On a real computer, the kernel maps physical *hardware ports* to virtual *interfaces*
  - Example: The leftmost hardware port is named `em1`
- Interfaces are associated with IP addresses
  - Example: Interface `em1` has address 192.168.1.1

- Interfaces can be manipulated using the `ip` command
  - `ip addr show`
- Interface connectivity can be tested using the `ping` command
  - `ping 192.168.1.2`

What is a cluster network useful for?

- Network booting: transferring an initial image from a master node to a compute node
- Remote login: using SSH to open a command line on another system
- Scheduling: submitting a compute job to a queue that will be run on part or all of the cluster
- Remote logging: sending logs from compute nodes to a master node
- And many other tasks

- All of these tasks are enabled by *services*

Services

Services

- Services are long-running processes that (usually) listen on a TCP or UDP port and perform actions based on incoming connections from other systems
  - These processes are referred to as “daemons”
  - Unlike most processes that you start in your shell, daemons are run in the background and are not associated with a login process
- Examples
  - `sshd`: provides remote login capabilities for a system
  - `httpd`: serves up web pages from a system
  - `mysqld`: provides a database on a system

Services

- Service processes listen for connections on an IP address and a well-defined software port
  - HTTP: port 80
  - HTTPS: port 443
  - SSH: port 22
- Ports assigned to standard services are listed in /etc/services
- Service protocols are generally built on top of TCP or UDP
  - The choice often depends on and latency and resiliency requirements
- Services on networks are sometimes referred to by their *address:port* combination
  - 192.168.1.1:80

Services

- Services you will likely see on an HPC cluster include:
  - SSH – Port 22, TCP; enables remote access
  - DNS – Port 53, UDP; Provides host name lookup
  - DHCP – Port 67, UDP; hands out IP addresses
  - NTP – Port 123, UDP; synchronizes clocks
  - Syslog – Port 514, UDP; collects system logs
  - Slurm – Port 6817, TCP; provides scheduling services

systemd

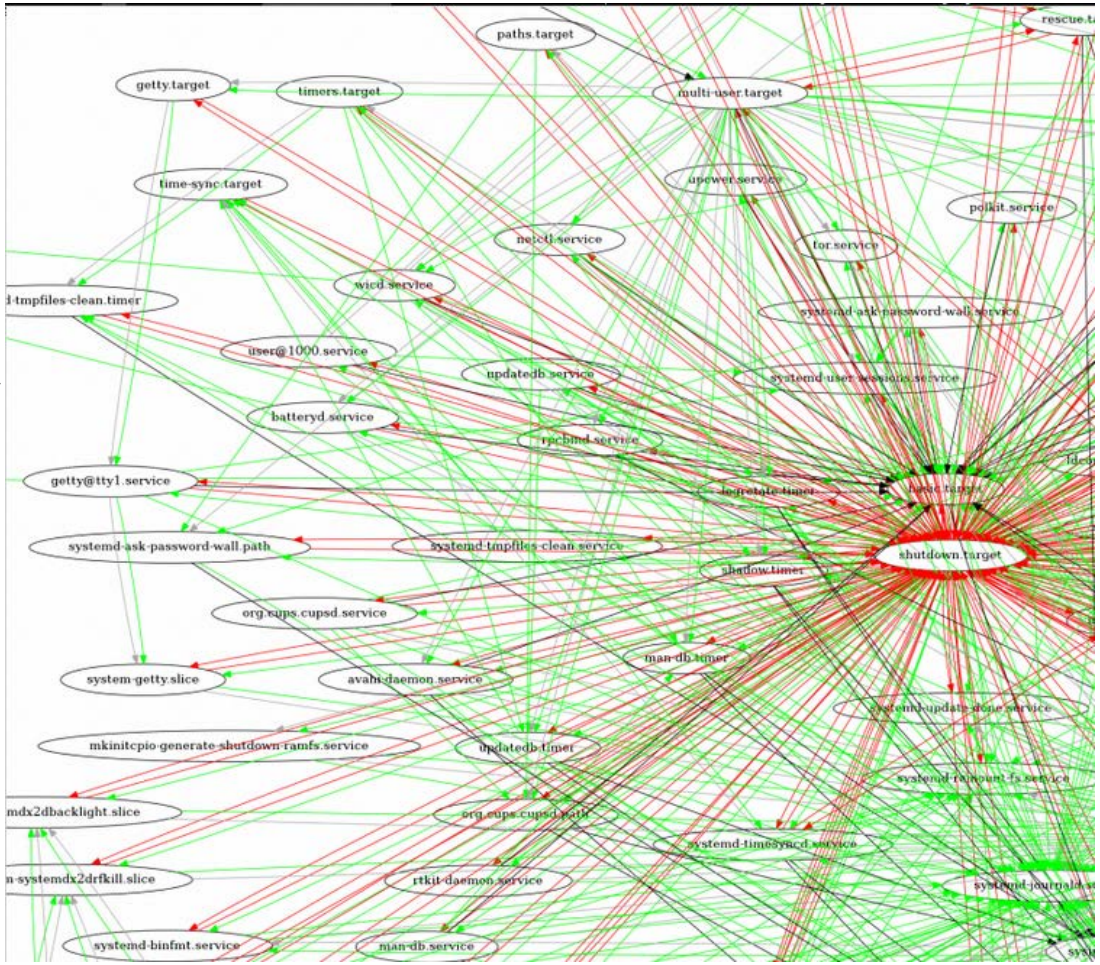
- systemd is the init system used on most modern Linux distributions
- On OS bootup, systemd performs initial service startup
  - Starts all of the service processes and puts them in the background
- After boot, systemd manages running services
  - Can monitor and restart services when they die
  - Provides tools for interacting with running or stopped services
- The base systemd object is the *unit*
- Units can be…
  - services
  - target levels
  - mountpoints
  - … and a variety of other things

systemd Unit File

```
# /usr/lib/systemd/system/ntpd.service

[Unit]
Description=Network Time Service
After=syslog.target ntpdate.service sntp.service

[Service]
Type=forking
EnvironmentFile=-/etc/sysconfig/ntpd
ExecStart=/usr/sbin/ntpd -u ntp:ntp $OPTIONS
PrivateTmp=true

[Install]
WantedBy=multi-user.target
```

Systemd Dependencies

- Most traditional service managers start services in a prescribed sequence
- Systemd, instead, can start/stop services in parallel
  - It does this by collecting all of its *units* together and making a directed graph of dependencies
  - Any process for which the dependencies have been satisfied can start/stop
    - …no prescribed sequence required
- But, this can lead to headaches like race conditions that are hard to diagnose



systemctl – The Main systemd Interface

- Start a service
  - `> systemctl start <servicename>`
- Stop a running service
  - `> systemctl stop <servicename>`
- Restart a running service
  - `> systemctl restart <servicename>`
- Check up on a service (running or not)
  - `> systemctl status <servicename>`
- List all units that systemd knows about
  - `> systemctl list-units`

Questions?

Los Alamos

NATIONAL LABORATORY

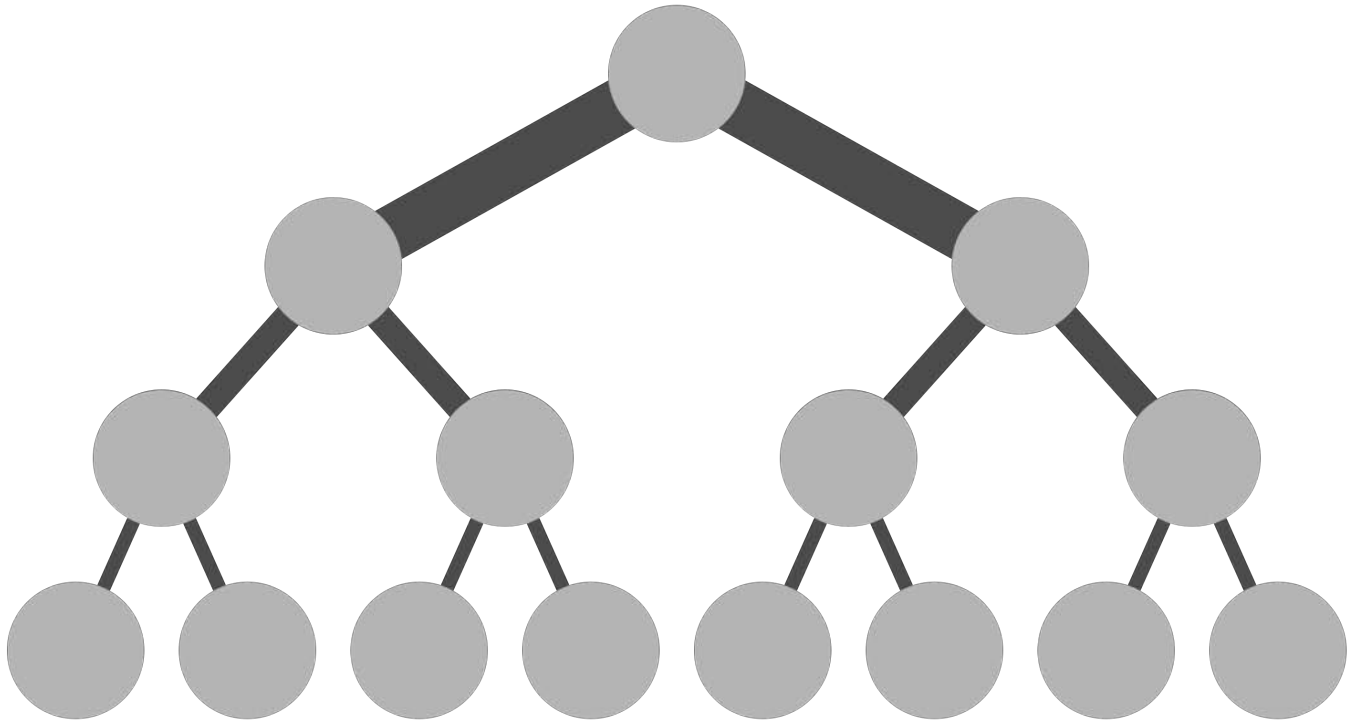
EST. 1943

Los Alamos
NATIONAL LABORATORY
EST. 1943

Delivering science and technology
to protect our nation
and promote world stability

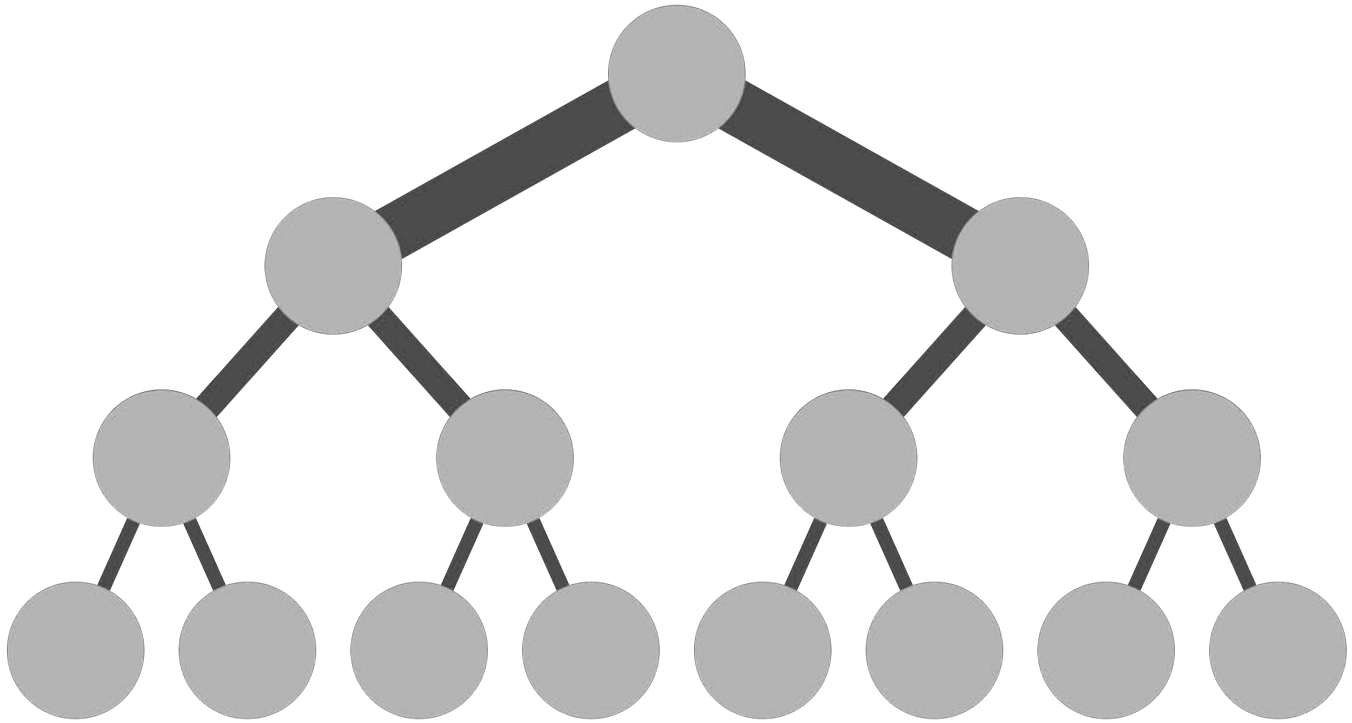
HPC Cluster Provisioning

Presented by CSCNSI



Agenda

- Overview of Cluster Provisioning Systems
- Warewulf



Agenda

- Overview of Cluster Provisioning Systems
- Warewulf

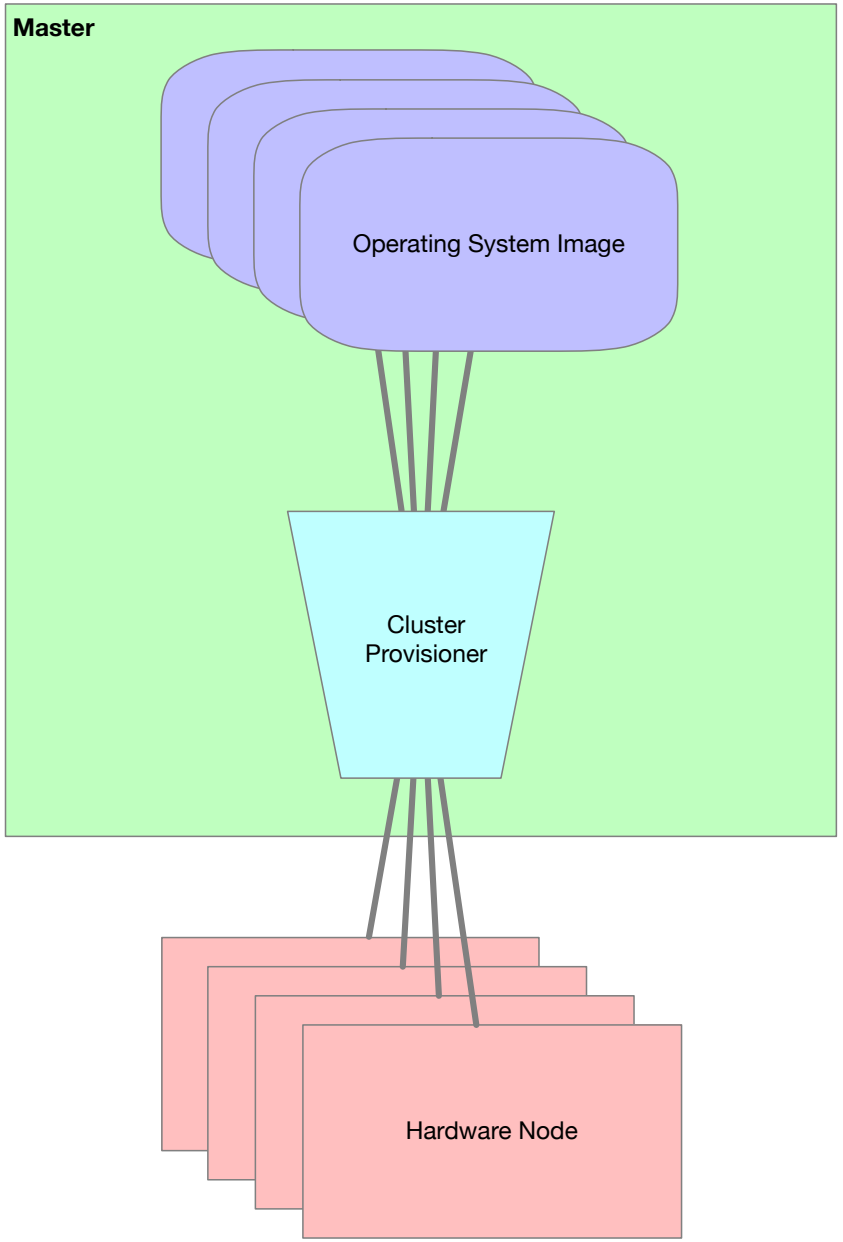
Agenda

- HPC Provisioning systems (overview)
- Warewulf

Overview of Cluster Provisioning Systems

What is HPC Provisioning?

- HPC Provisioning can be accomplished with a number of different provisioning tools
- Provisioning a cluster consists of two primary things
  - Providing a mechanism to boot/deploy a cluster node
  - Providing tools for maintaining operating system images for cluster nodes
- In other words, an HPC Provisioning tool streamlines and manages the netboot process we went through last time
- Often HPC Provisioning tools will do other work as well, such as maintain configuration files for nodes and provide some level of node monitoring



Some HPC Provisioning Systems

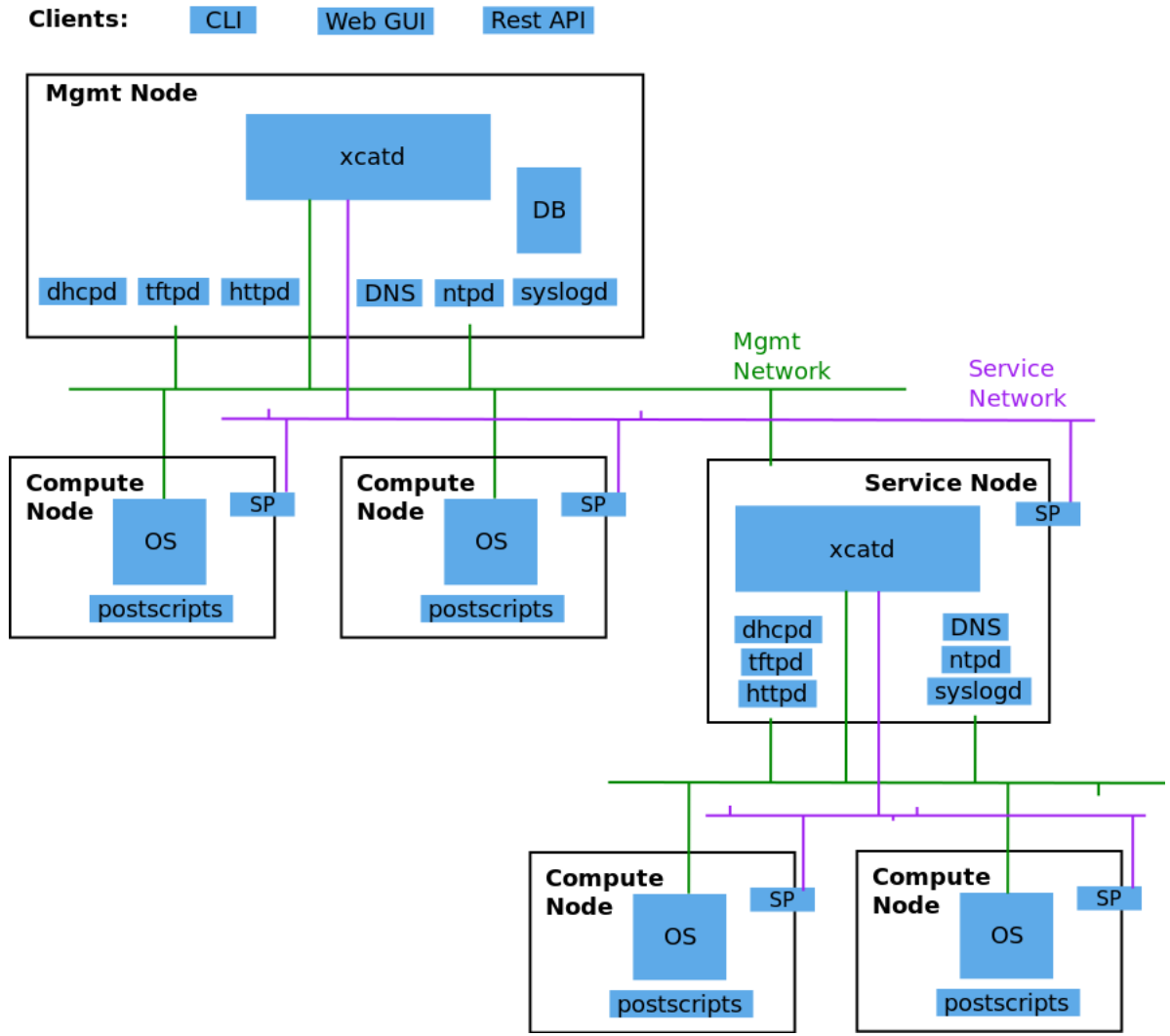
- There are a lot of node provisioning tools out there:
- Some open source tools:
  - Warewulf/Perceus
  - xCAT (IBM supported)
  - ROCKS (mostly deprecated)
- Some commercial tools:
  - Cray (various software)
  - IBM Platform HPC
  - Bright Cluster Manager

Architecture of an HPC Provisioning system

- Almost every existing HPC Provisioner has a similar architecture
- A front-end for managing cluster config information
  - Sometimes GUI (e.g. Bright)
  - Sometimes Text (e.g. Warewulf)
- A backend database store for cluster information
- Tools that use this information to generate config for
  - The PXE boot process (or iPXE)
  - DHCPD
  - Sometimes DNS, others
- Some integrate with other hardware like switches (e.g. xCAT)

xCAT

- Primarily maintained by IBM
- IBM offers commercial support
- VERY big code base (~500k lines of code)
- Monolithic design
- Supports hierarchical booting
- Has run on some very big systems
  - Including LANL's own Roadrunner



https://xcat-docs.readthedocs.io/en/stable/overview/architecture.html

Bright, ROCKS & More

- Commercial solutions are often bundled with hardware
  - Dell currently bundles Bright
  - Cray bundles… Cray
  - HPE has a mix of tools they call Performance Cluster Manager
  - IBM bundles either Platform HPC or xCAT
  - Penguin bundles Scyld
- Many older tools are largely defunct, but still come up:
  - ROCKS is still maintained but rarely run
    - And almost never on large systems

Warewulf/Perceus

- Warewulf v2 = Perceus
- Perceus was briefly a commercial project
- Warewulf 3 has some ongoing community maintenance
- Based on Perl and MySQL (MariaDB)
- Simple command-line interface
- *We will be using Warewulf 3*

All of these solutions do things just about how we did netboot!

Warewulf

Why we're using Warewulf

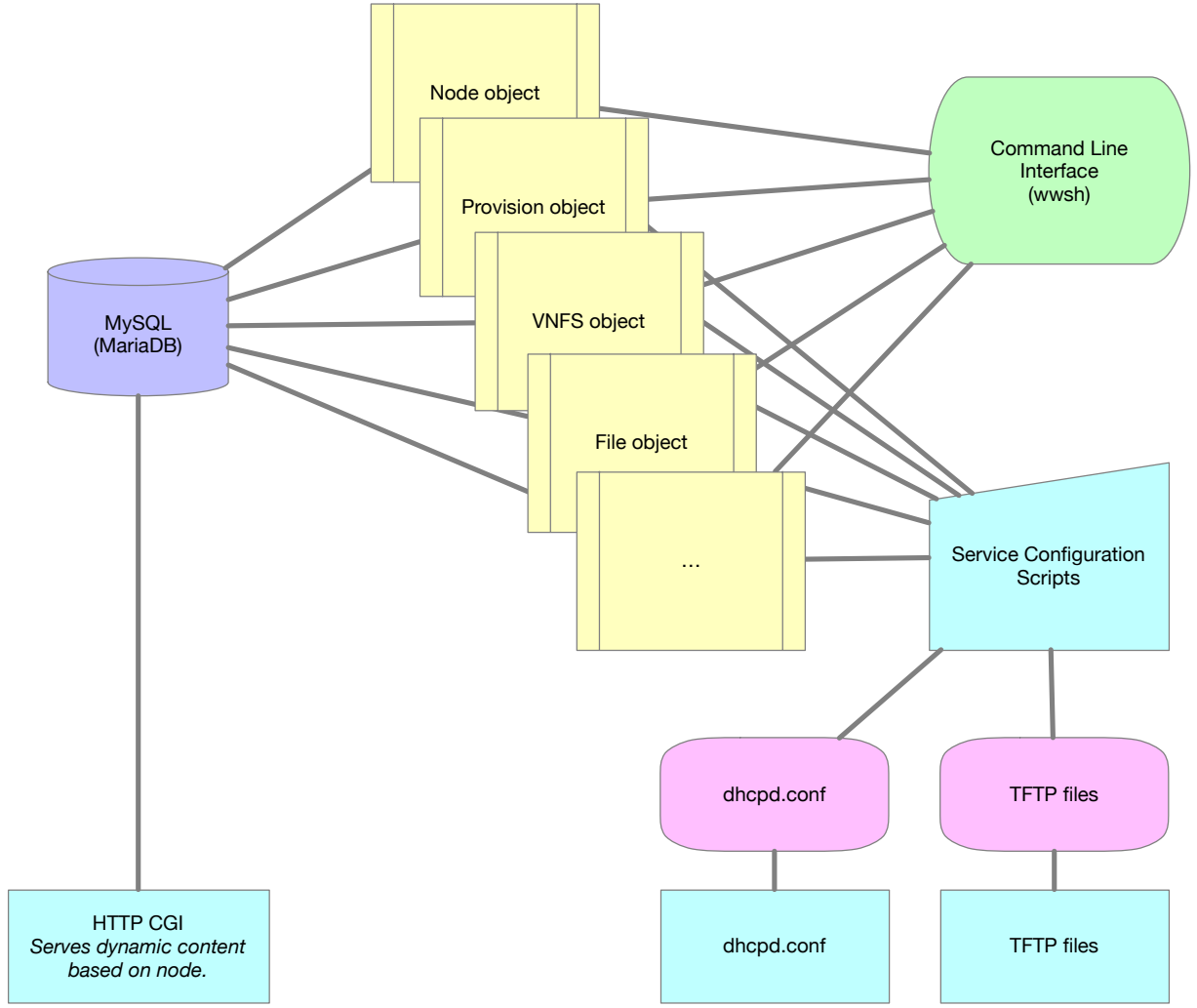
- Warewulf 3 is still (lightly) maintained
  - Mostly by Argonne National Labs
- Warewulf 3 is simple, compared to others
- Warewulf 3 is text-based
- We have some configuration management integrations with Warewulf (more on this later)
- Of the open source solutions, Warewulf and xCAT are the most used
  - But xCAT is much more complex

History

- Begun in 2000 by Gregory Kurtzer of LBNL
- “Ware” from “software” plus “wulf” from “Beowulf.”
- Used as part of Scientific Cluster Support pilot program
- Original version used floppy disks
- Pioneered fully stateless node provisioning over network
- Pioneered VNFS format/mechanism
- Provided the underlying code for xCAT 1.x stateless
- On “hiatus” from 2006-2010 during PERCEUS development
- Revived in 2010 with a complete redesign/rewrite

Architecture

- Perl objects provide an interface to a backend database
- CLI can interact with Perl objects to, e.g. add a node
  - Has a shell-like interface, `wwsh`
- Service Configuration Scripts write service configurations files
  - Event triggered
  - Or manually triggered
    - (e.g. `wwsh dhcp update`)
- HTTP CGI (Apache) provides information to nodes (like their Images, Kernel, etc.)



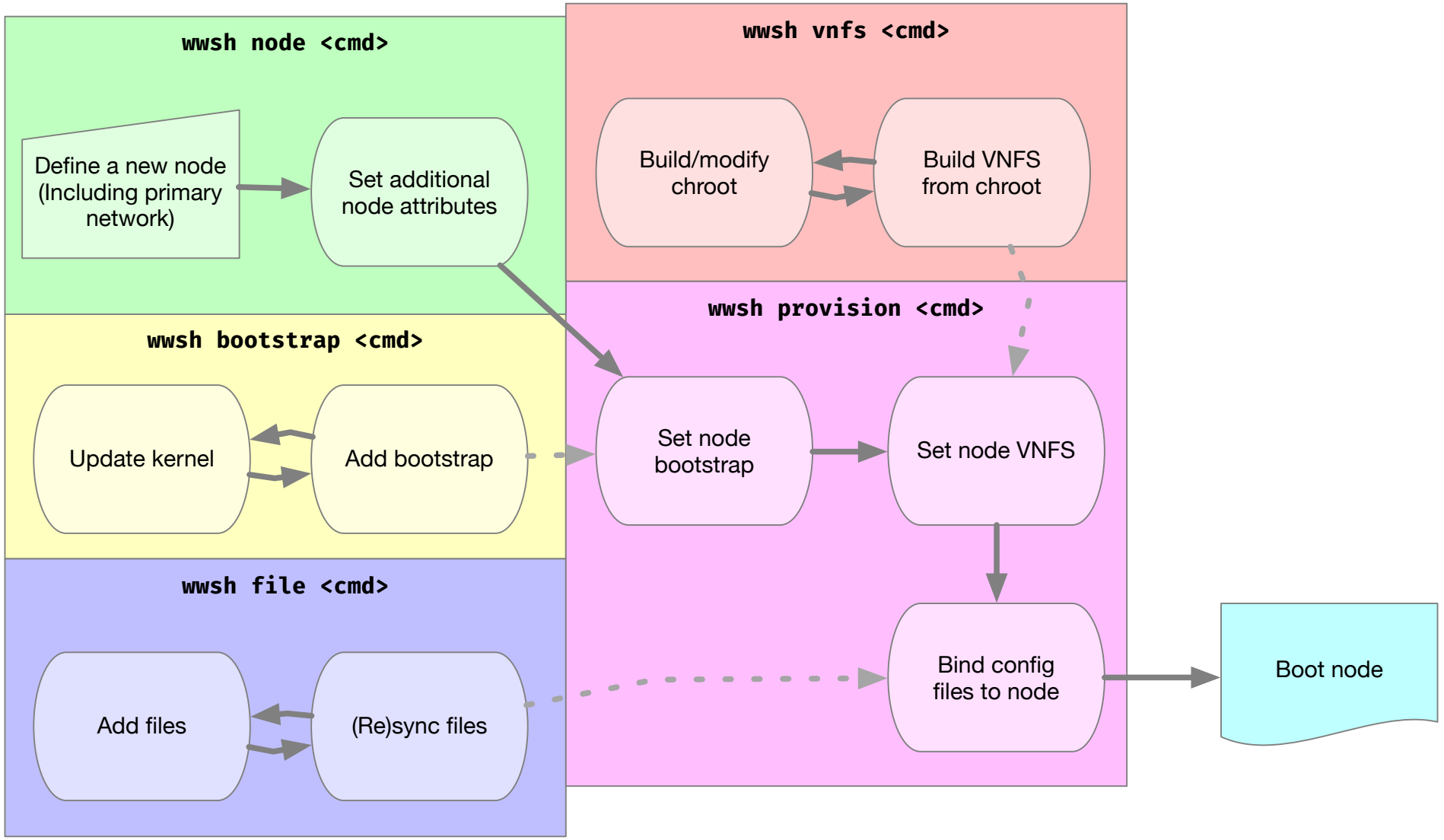
Terminology

- **Bootstrap**: Kernel+modules+firmware bundle assigned to nodes
  - Imported into data store from VNFS template via wwbootstrap
  - Must contain all kernel files required to boot nodes
- **VNFS image**: Virtual Node FileSystem; nodes' root FS image
  - Imported into data store from VNFS template via wwvnfs
  - Downloaded via HTTP by initramfs; used to populate tmpfs
- **initramfs**: Initial root FS image downloaded via PXE/TFTP
  - Combination of a bootstrap object and a shared base image
  - Contains Warewulf init, provisioning scripts, and capabilities
- **Capabilities**: Modular boot-time functionality in cpio format
  - Grouped into categories (e.g., provision, setup, transport)
  - provision-files, setup-filesystems, transport-http

Provisioning Step-by-Step

- Node's NIC PXE boots (DHCPs); TFTPs kernel+initramfs
- Kernel boots and runs WW /init script from initramfs
- Warewulf initializes network using DHCP or kernel cmdline
- wwgetnodeconfig in default (http) transport queries Node
- provisionhandler runs series of numbered scripts in initramfs
- If prescript property defined on Node, run named File script
- Create all partitions and filesystems (default is tmpfs on /)
- Download VNFS. Update network config, fstab, runtime.
- Copy over /dev and kernel files. Make bootable if needed.
- Pull provisioned files and "unmount" filesystems.
- If postscript property defined on Node, run it
- Chroot into new rootfs and run /sbin/init there. Node boots.

Basic Workflows



Useful commands

`wwinit <feature>` - initializes warewulf configuration
`wwsh` – interactive shell where commands can be run
`wwbootstrap <kernel_version>` – build/import a bootstrap (kernel & initramfs)
`wwmkchroot` – builds a minimal OS image chroot
`wwvnfs --chroot <dir>` – create a VNFS image from a chroot & import
`wwsh node new […]` – define a node
`wwsh provision set <node> –vnfs=<vnfs> --bootstrap=<bootstrap> […]` – set the VNFS, bootstrap and files associated with a node
`wwsh pxe update` – update pxe config files
`wwsh dhcp update` – update dhcpd.conf file
`wwsh file import` – add a file (config) to the image (can livesync)
`wwsh file sync` – (re)sync files

Questions?

Los Alamos

NATIONAL LABORATORY

EST. 1943

Los Alamos
NATIONAL LABORATORY
EST. 1943

Delivering science and technology
to protect our nation
and promote world stability

Workload Management & Slurm

Presented by CSCNSI

What is Workload Management?

- We need a way to easily put our many compute nodes to use
- We need a tool that can match work that needs to be done with resources that are available
- It needs to be smart enough to
  - Start multi-node jobs
  - Schedule jobs from a long list of waiting tasks according to policy
  - Know what resources are available
  - Know what jobs to send to what hardware
  - Know how to start a job on our hardware
- And generally a lot of other things, like keeping track of accounting information, etc.

Scheduling vs. Resource Management

Scheduler

- The “Brain” of the system
- Single, centralized daemon
- Obtains job and node data from Resource Manager
- Prioritizes jobs
- Decides:
  - What jobs will run
  - Where they will run
  - When they will run

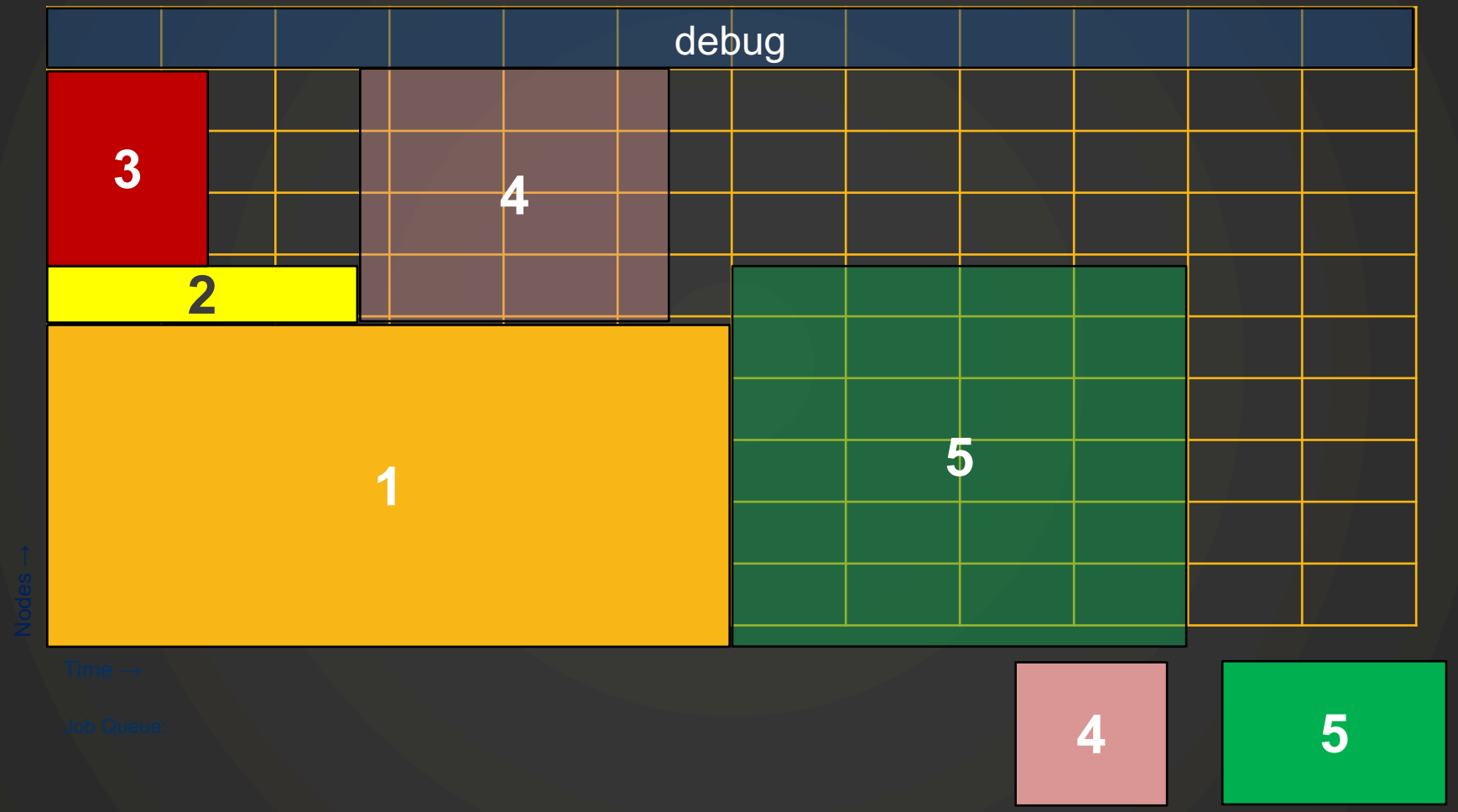
Resource Manager

- The “Worker” of the system
- Server daemon plus per-node client daemons
- Gathers node information (CPUs, memory, disk, load)
- Answers queries for this information
- Spawns jobs when told to
- Can add/remove resources

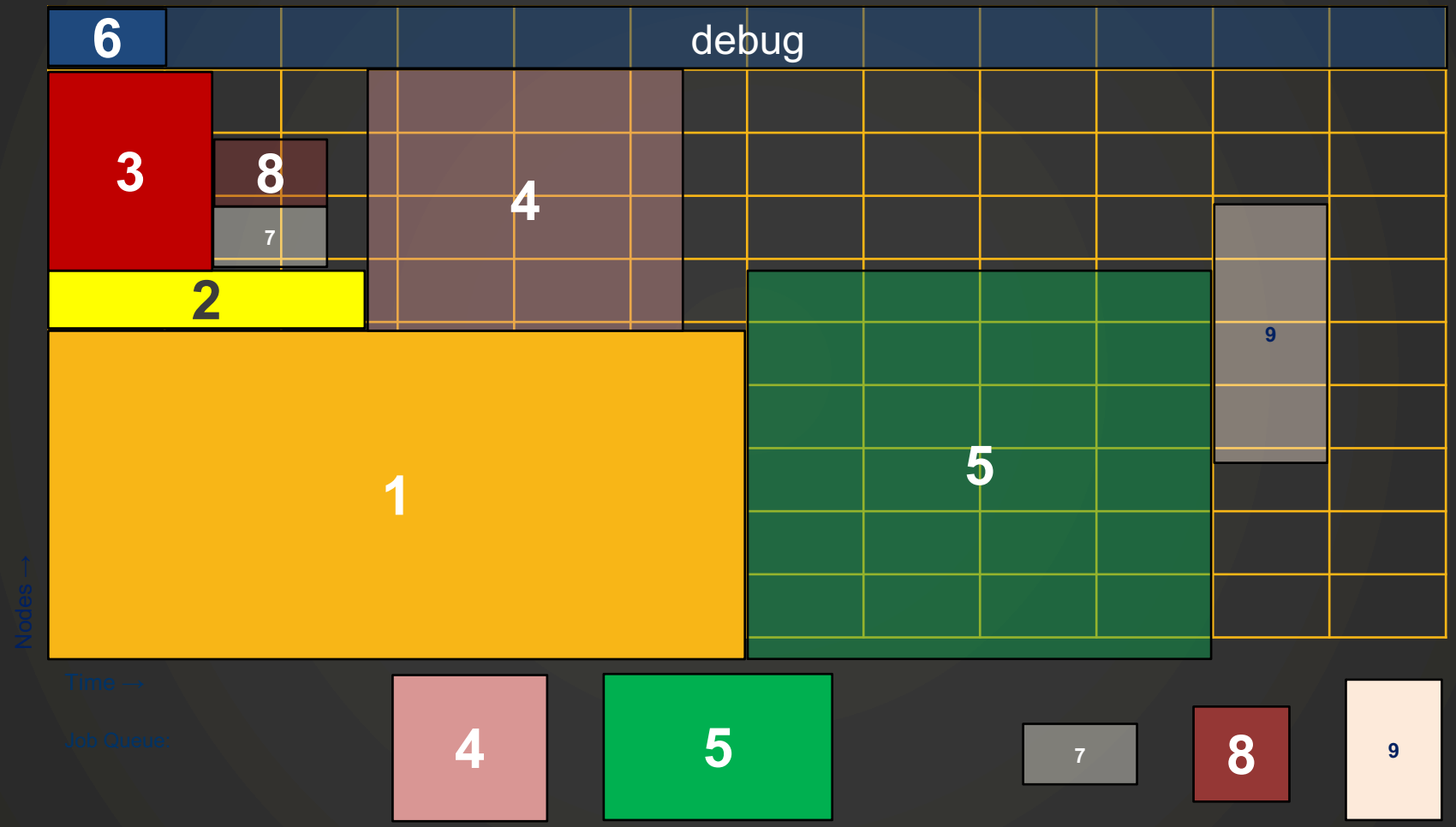
Scheduling Cycle

- Job Ordering
  - Calculate job priorities and sort job list by priority
  - Start all jobs which can start
  - Make reservations for next 1024 jobs
  - Use backfill policy to sort remaining jobs (1024)
- Job Placement (for each job being started)
  - Filter list of all nodes based on requirements
  - Filter list of nodes based on eligibility
  - Sort final node list based on node allocation policy
  - Start job on top n nodes

Scheduling Example – Iteration #1



Scheduling Example – Iteration #2 (backfill)

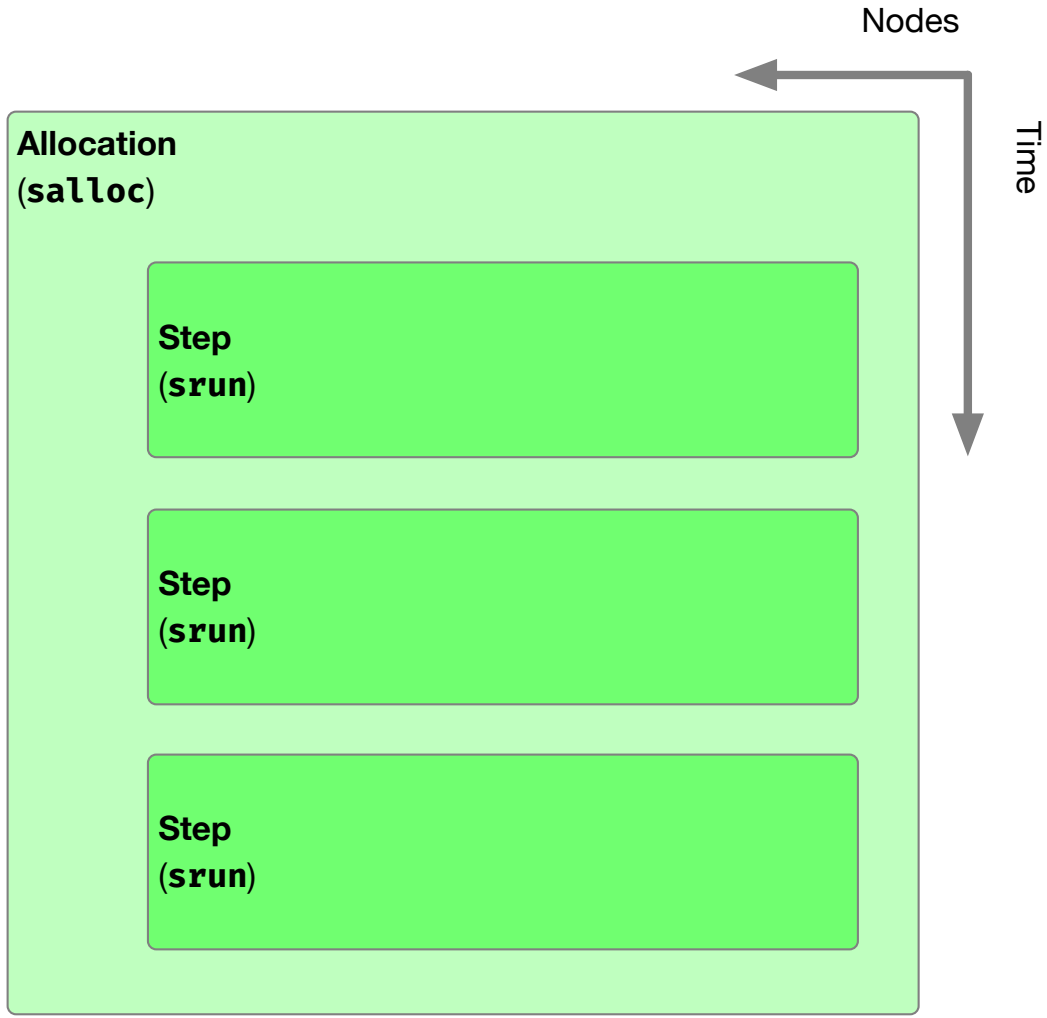


What is Slurm?

- Slurm is a Workload Manager for HPC workloads
- Originally "*Simple Linux Utility for Resource Management*"
  - Was spelled SLURM
- Since it now is also a scheduler, was rebranded *Slurm* (not an acronym)
- Originally written at LLNL
- Now commercially developed & supported by SchedMD
- Used on more than half of the top 10 of the top500

Structure of a Slurm job

- A job needs an “allocation”
  - Specifies needed resources
    - Memory
    - Nodes/CPUs
    - …
  - Specifies a max time
- Within an allocation multiple steps can execute
  - Steps, in total, must fit within allocation



Slurm interactive job

- An allocation is made with the **`salloc`** command
- An allocation is bound to a process
- **`srun`** started within that process create a step
- The job ends when the allocation process ends

```
[fe] $ salloc –n1 /bin/bash
salloc: Granted job allocation 423
[fe] $ srun --pty /bin/bash
[n01] $ ./do_work
[n01] $ exit
[fe] $ srun –pty ./do_more_work
(runs on n01)
[fe] $ exit
salloc: Relinquishing job allocation
423
[fe] $
```

Slurm batch job

- Jobs can be specified as a pre-defined script
- Special *#SBATCH* lines provide Slurm parameters
- Can submit multiples of the same task
  - Or can submit arrays of the same job, keyed by the *$SLURM_ARRAY_TASK_ID*
- Submitted with:
  - *sbatch <script>*

```
#!/bin/bash
#
#SBATCH --job-name=test_emb_arr
#SBATCH --output=res_emb_arr.txt
#SBATCH --ntasks=1
#SBATCH --time=10:00
#SBATCH --mem-per-cpu=100 #
#SBATCH --array=1-8
srun ./my_program.exe \
    $SLURM_ARRAY_TASK_ID
```

Common Slurm commands

Start an allocation	**`salloc`**
Run a step in an allocation	**`srun`**
Submit a batch script	**`sbatch`**
View the queue	**`squeue`**
Stop a running job	**`scancel`**
Attach to I/O of current task	**`sattach`**
View the cluster status	**`sinfo`**
View job accounting data	**`sacct`**
Control Slurm	**`scontrol`**

All of the commands have extensive options. See `man <slurm_command>`

Questions?

Los Alamos

NATIONAL LABORATORY

EST. 1943

NNSA
National Nuclear Security Administration
Managed by Triad National Security, LLC for the U.S. Department of Energy's NNSA

Los Alamos
NATIONAL LABORATORY
EST. 1943

Delivering science and technology
to protect our nation
and promote world stability

National Nuclear Security Administration
Managed by Triad National Security, LLC for the U.S. Department of Energy's NNSA

Getting  git.

An introduction to git version control & methodologies

Presented by CSCNSI

`git` --the-basics

What is git ?

"Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency." (git-scm.com)

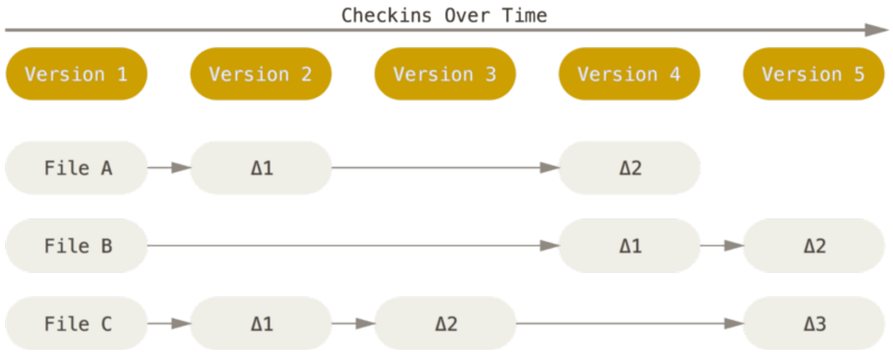
- **`git`** is *free* and *open sourced*
- **`git`** is *fast* and *efficient*
- **`git`** is *decentralized* version control
  - there is no "main" or "master" repo
- **`git`** is designed to handle *non-linear* development
  - allow many developers to work in parallel

What `git` does
(extremely incomplete)

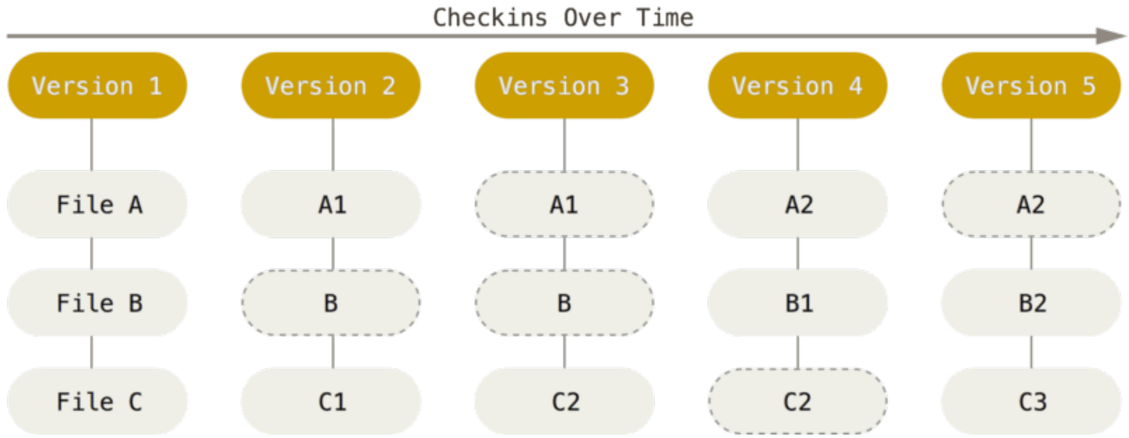
- Manages a revision history of a directory full of stuff (*commit, checkout*)
- Can sync that directory with other directories (*cloning, pushing, pulling*)
- Can manage multiple histories of one directory (*branching, tagging*)
- Provides tools for collaboratively working on directories (*merging, rebasing*)
- Provides assurances that the stuff is intact as advertised (*hashes, signing*)

How `git` does what it does

- Stores local repository information in a special directory, **`.git`**, in the base directory of the repo
  - Including config info, like who else we might want to synchronize with, called *remotes*
- Keeps snapshots-in-time of the contents of the directory, a *commit*
  - Note: this is *different* than how most version control works
- Keeps a pointer to the current snapshot (the *HEAD*), and populates the directory with its contents
- Tries to be smart about not storing the same data twice (copy-on-write-ish), keeps hashes, etc.



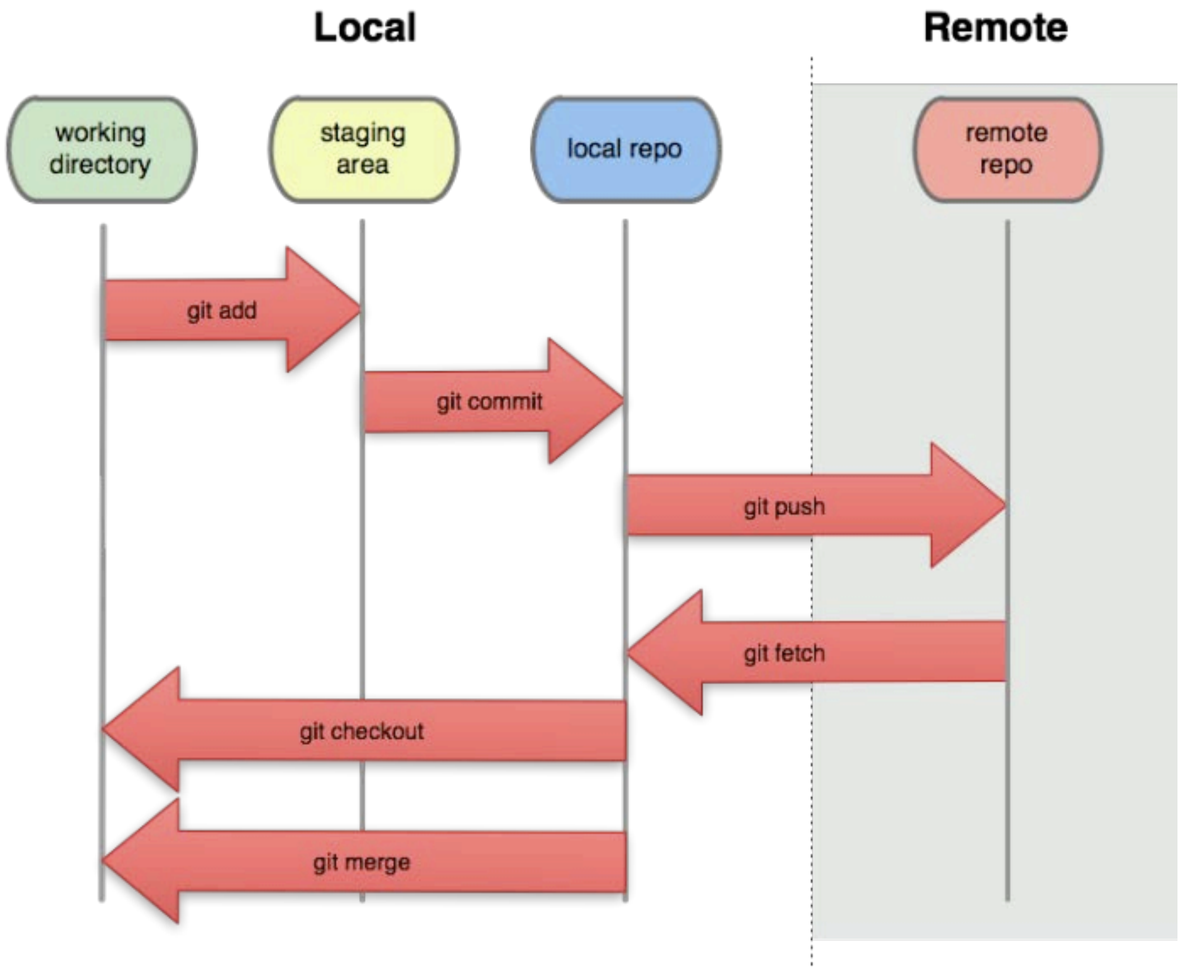
Other VCS



Git

`git` works locally

- Git does not use a central repository
- Instead, you work in your own local copy
- But if you make a new commit you can *push* it elsewhere (to a *remote*)
  - ...or if they make a new commit you can *pull* it to your copy
- Multiple people can work on individual histories of commits, or *branches*
  - You can *checkout* a remote's branch
  - You can *merge* changes from that branch

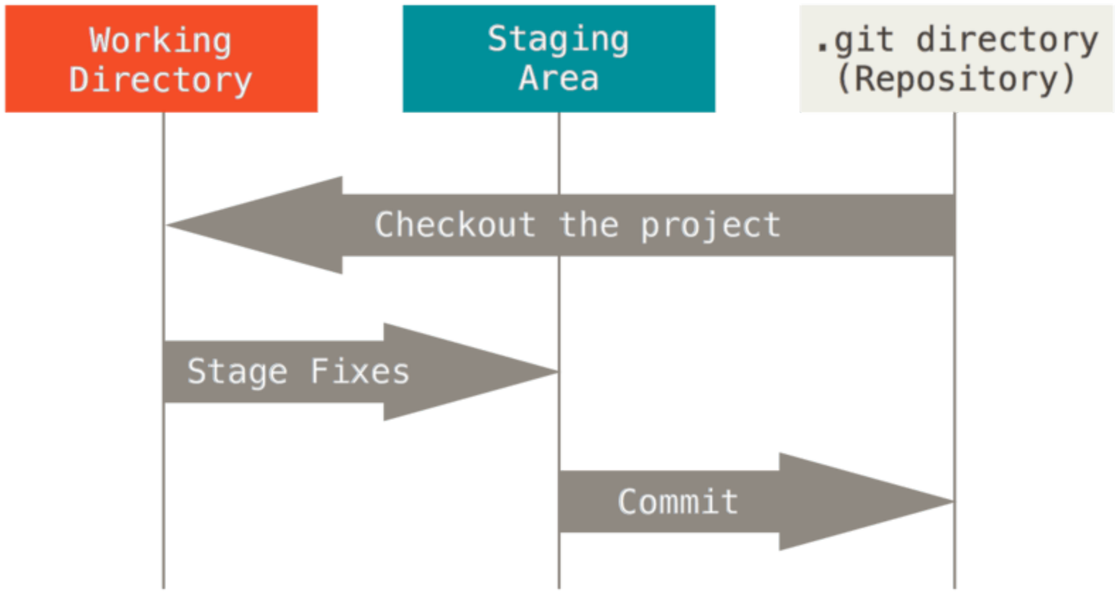


https://greenido.files.wordpress.com/2013/07/git-local-remote.png?w=696&h=570

The `git` repo state

A working copy of a repo can be in one of three states:

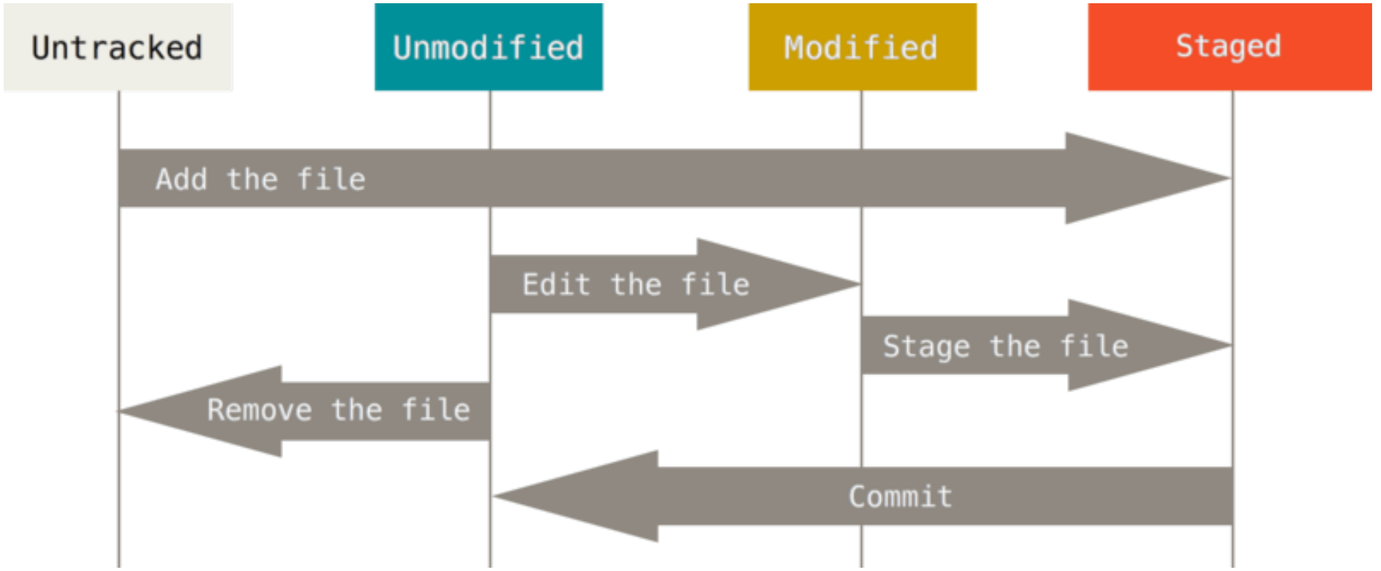
- ***Committed*** means that the data is safely stored in your local database, i.e. completely matches an existing commit.
- ***Modified*** means that you have changed the file but have not committed it to your database yet.
- ***Staged*** means that you have marked a modified file in its current version to go into your next commit.



https://git-scm.com/book/en/v2

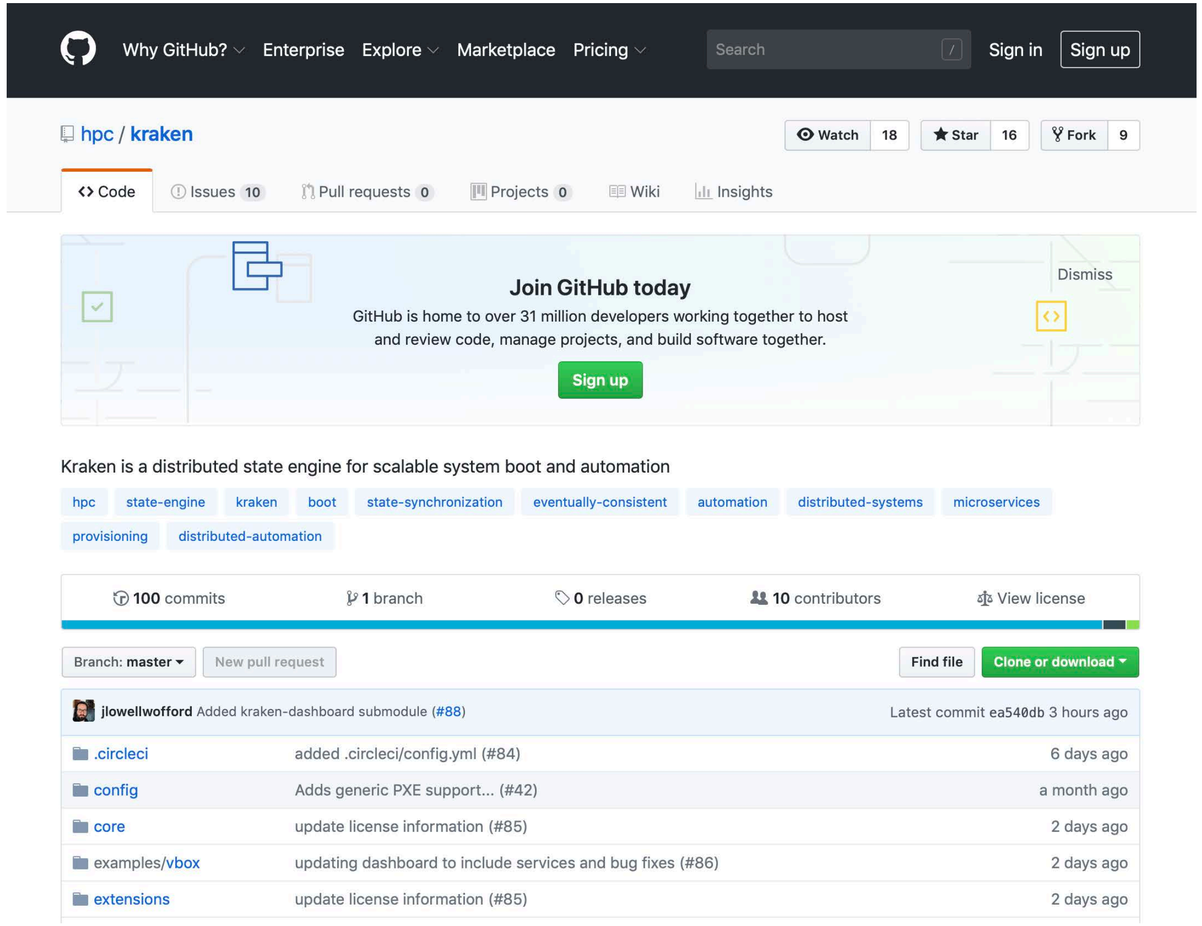
Basic `git` workflow

1. *init* a new repo, or *clone* one from somewhere else
2. create/delete/modify files in the repo
3. *add* files to the staged commit
4. *commit* the changes
5. GOTO 2, or...
   - *push* those commits to elsewhere
   - *checkout* an old commit
   - *pull* someone else's changes from elsewhere



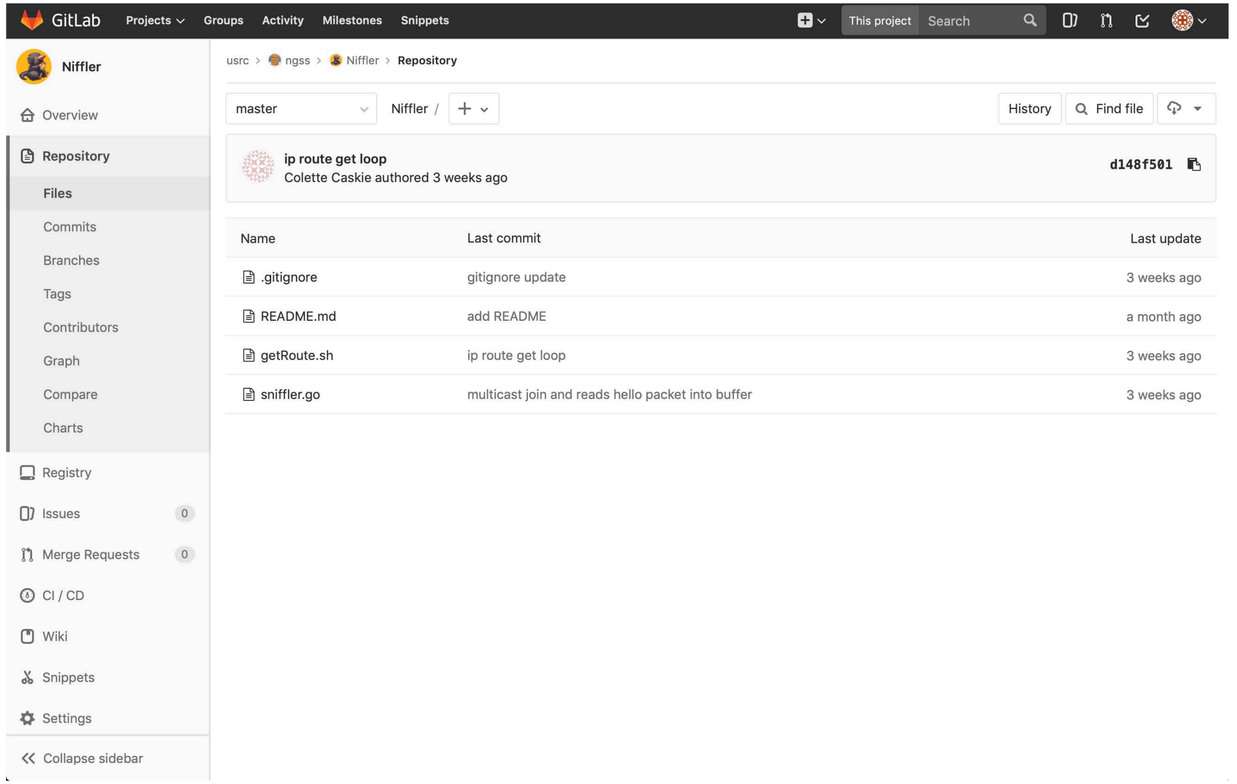
What is GitHub?

- *Github should not be confused with Git*
- Github (https://github.com) is a website dedicated to hosting git repositories
- Github offers some great add-on features:
  - A vast community of open source developers
  - An add-on workflows for collaboration, like “Pull Requests”
  - Loads of add-ons like Continuous Integration (CI) and automated testing tools
  - …and much more.



What is GitLab?

- *Gitlab should not be confused with Git or Github*
- Gitlab is a project for hosting Github-like sites
- LANL has multiple Gitlab instances:
  - https://git.lanl.gov
  - https://gitlab.newmexicoconsortium.org
- Gitlab has many of the features of Github
  - Pull request workflows (called “Merge” requests)
  - Continuous Integration (CI)
  - …



Questions?


Los Alamos
NATIONAL LABORATORY
EST. 1943

Los Alamos
NATIONAL LABORATORY
EST.1943

Delivering science and technology
to protect our nation
and promote world stability

NNSA
National Nuclear Security Administration
Managed by Triad National Security, LLC for the U.S. Department of Energy's NNSA

Configuration Management

…with Ansible

CSCNSI

Configuration Management

Configuration Management

"The discipline of applying technical and administrative direction and surveillance to identify and document the functional and physical characteristics of a *configuration item,* control changes to those characteristics, record and report change processing and implementation status, and verify compliance with specified requirements." --IEEE-Std-610

Configuration Management

- *Ad-Hoc – control what is convenient*: "by hand" modification of a host's software stack to achieve a desired result

```
#  vi /etc/dhcp/dhcpd.conf
```

Configuration Management

- *Ad-Hoc – control what is convenient*: "by hand" modification of a host's software stack to achieve a desired result

  ```
  #  vi /etc/dhcp/dhcpd.conf
  ```

- *Partial – control a few things*: use some gizmo as a means to the same end, where ...
  - Source of the configuration change is external to the host software stack
  - Desired result can be verified and re-achieved

Configuration Management

- *Complete – full prescription*: use a gizmo (or gizmos) to somehow specify everything about the configuration of a host

Configuration Management

- *Complete – full prescription*: use a gizmo (or gizmos) to somehow specify everything about the configuration of a host
  - Deterministic: know what your host is running

Configuration Management

- *Complete – full prescription*: use a gizmo (or gizmos) to somehow specify everything about the configuration of a host
  - Deterministic: know what your host is running
  - Reproducible: bare-metal {re}install

Configuration Management

- *Complete – full prescription*: use a gizmo (or gizmos) to somehow specify everything about the configuration of a host
  - Deterministic: know what your host is running
  - Reproducible: bare-metal {re}install
  - Convergent: recovery from unforeseen events

Configuration Management

- *Complete – full prescription*: use a gizmo (or gizmos) to somehow specify everything about the configuration of a host
  - Deterministic: know what your host is running
  - Reproducible: bare-metal {re}install
  - Convergent: recovery from unforeseen events
    - Implies the gizmo can validate the configuration state and take corrective action
    - Implies the gizmo can be run on a regular basis

Why we do CM

Why we do CM

Because we hate system administration!

Why we do CM

Because we hate system administration!

Also, scale.

Why we do CM

- The CMbot never tires
- The CMbot never forgets what it has been taught
- The CMbot's intelligence is the sum of *everyone's* expertise
- The CMbot is scalable to many nodes and architectures
- The CMbot increases human performance/reliability, freeing us up to do the things we are paid to do *and* enjoy

Intro to Ansible

Ansible is…

- … a configuration management tool
- … an automation tool
- … written in python
- … open-source
- … led out of Red Hat

Ansible's Philosophy

- Ansible performs tasks
  - install a package, copy a file, enable a service, etc.
- Tasks are run on hosts
  - ba-master.lanl.gov
- Related tasks are grouped into roles
  - "slurm server": install slurm package, copy slurm config files, enable slurm service
- Plays assign roles to hosts and run the corresponding tasks
  - ba-master.lanl.gov is a slurm server
- Playbooks orchestrate groups of plays
  - Update master node, then update compute node images, then copy images to service nodes

Tasks

- A base unit of work that needs to be done
  - copy a file
  - install an RPM
  - enable a service
  - create a cron job
  - … and many more
- Tasks are performed by Ansible **modules**
  - copy
  - yum
  - systemd
  - cron
  - … and many more

Example Task: Install and Enable ntp

- Goal:
  - Install the ntp package
  - Enable and start the ntpd service

- On a RHEL7 system:
  - `yum install ntp`
  - `systemctl enable ntpd`
  - `systemctl start ntpd`

Example Task: Install and Enable ntp

- Two Ansible tasks: install the package, enable the service
- Ansible invokes a module to do each task
  - No Linux commands in the tasks
- Tasks generally define the desired state, the module takes care of enforcing it
- Tasks should be idempotent: running multiple times will not change the result

roles/ntp/tasks/main.yaml

```
# Use the ‘package’ module to
# ensure the ntp package is
# installed
- name: "install ntp package"
  package:
    name: ntp
    state: present

# Use the ‘service’ module to
# ensure the service is enabled and
# started
- name: "enable ntpd service"
  service:
    name: ntpd
    state: started
    enabled: yes
```

Example Task: Drop /etc/ntp.conf in place

- Goal:
  - Copy our customize ntp.conf file to /etc/ntp.conf
  - Make the file owned by root:root
  - Set the file's permissions to 0444

ntp.conf

```
# Cluster NTP servers
server 204.121.3.1 prefer
server 204.121.6.1
server 127.127.1.0
fudge  127.127.1.0 stratum 10
```

- On a RHEL7 system

```
$ cp ntp.conf /etc/ntp.conf
$ chown root:root /etc/ntp.conf
$ chmod 0444 /etc/ntp.conf
```

Task: Drop a config file in place

- Uses the copy module
- Copies a file from the repository to a location on the filesystem
- Sets specified permissions and ownership in the process
- Default source: roles/rolename/files/src

roles/ntp/tasks/main.yaml

```
- name: "install ntp config file"
  copy:
    src: ntp.conf
    dest: /etc/ntp.conf
    owner: root
    group: root
    mode: 0444
```

Templates

- Static files aren’t always sufficient
- Templates embed variables inside files
- Ansible uses the Jinja2 templating engine to process these files
- Templates can do things like…
  - simple variable substitution
  - loops over lists of variables
  - include other files
  - much more complex actions

Task: Drop a templated config file in place

- Example one:
  - “ntp_server” is a variable
    - ntp_server = 204.121.3.1
  - Jinja2 replaces the variable between {{ and }} with its value
  - Everything else in the file is left alone

Static: roles/ntp/files/ntp.conf

```
# Cluster NTP servers
server 204.121.3.1
server 127.127.1.0
fudge  127.127.1.0 stratum 10
```

Templated: roles/ntp/templates/ntp.conf.j2

```
# Cluster NTP servers
server {{ ntp_server }}
server 127.127.1.0
fudge  127.127.1.0 stratum 10
```

Task: Drop a templated config file in place

- Example two:
  - “ntp_servers” is a list of ntp servers
  - Jinja2 interprets the expression between {% and %} as a control structure
  - Control structure syntax is very similar to python syntax

Static: roles/ntp/files/ntp.conf

```
# Cluster NTP servers
server 204.121.3.1
server 204.121.6.1
server 127.127.1.0
fudge  127.127.1.0 stratum 10
```

Templated: roles/ntp/templates/ntp.conf.j2

```
# Cluster NTP servers
{% for ip in ntp_servers %}
server  {{ ip }}
{% endfor %}
server 127.127.1.0
fudge  127.127.1.0 stratum 10
```

Task: Drop a templated config file in place

- The template module:
  - Reads the template file
  - Runs the content through the template engine
  - Writes the result to the specified destination
- Default source: roles/rolename/templates/

main.yaml

```
- name: "install ntp config file"
  template:
    src: ntp.conf.j2
    dest: /etc/ntp.conf
    owner: root
    group: root
    mode: 0444
```

Variables

- Can be set…
  - At the host level
  - At the group level
  - In roles, tasks, playbooks, and several other places
- Precedence (greatest to least):
  - host-specific
  - other groups
  - “all” group
  - role defaults

inventory/host_vars/kit-master.lanl.gov/main.yaml

```
cluster_master_hostname: 'kit-master'
```

inventory/group_vars/ccstar/main.yaml

```
ntp_servers:
 - '128.165.4.4'
 - '128.165.4.33'
```

Hosts

- **Hosts** are individual systems that Ansible knows about
- Examples
  - ba-master.lanl.gov
  - kit-master.ccstar.lanl.gov
- The Ansible client …
  - can be run locally on a host
  - can be run on a central systems that connects to the host via ssh

Groups

- **Groups** are names that can be used to target a set of hosts
- Examples
  - turquoise, ccstar, yellow
  - masters, logins, computes
  - badger, kit
  - room341, room205, room270
- Hosts can be members of multiple groups
- All hosts are members of an implicit “all” group

The Inventory

- Defines:
  - Hosts
    - In this example: kit-master, ba-master
  - Groups
    - In this example: ‘masters’, ‘turquoise’, and ‘ccstar’
- Hosts can be members of many groups
  - kit-master is a member of ‘masters’, ‘ccstar’, and ‘all’
  - ba-master is a member of ‘masters’, ‘turquoise’, and ‘all’

inventory/hosts

```
[masters]
kit-master.ccstar.lanl.gov
ba-master.lanl.gov

[turquoise]
ba-master.lanl.gov

[ccstar]
kit-master.ccstar.lanl.gov
```

Roles

- **Roles** combine related tasks into reusable building blocks
- Examples:
  - webserver
  - slurm-master
  - slurm-client
  - mysql
  - ssh
- Assigned to hosts or groups that need a role's functionality
  - ba-master needs "slurm-master" and "ssh"
  - ba-fe1 needs "slurm-client" and "ssh"

Example Role: NTP

- tasks/
  - Contains yaml files that define the tasks for this role
- templates/
  - Contains templates for this role
- handlers/
  - Contains callbacks that affect components managed by this role
- defaults/
  - Defines default values for variables used in this role

roles/ntp/

```
tasks/
  main.yaml
templates/
  ntp.conf.j2
handlers/
  main.yaml
defaults/
  main.yaml
```

Plays

- Assign individual tasks to hosts (or groups)
- Assign roles to hosts (or groups)

master-roles.yaml

```
- hosts: master
  roles:
  - common
  - nfs
  - ntp
  - slurm
```

master-tasks.yaml

```
- hosts: master
  tasks:
  - name: "install ntp package"
    package:
      name: ntp
      state: present
  - name: "enable ntpd service"
    service:
      name: ntpd
      state: started
      enabled: yes
```

Playbooks

- Sequences of plays to be run in order
- Can support orchestration workflows
  - Build a master
  - Next, build frontend images
  - Finally, build compute images
- Can support more complex orchestration workflows
  - Build a web server, then build a database server, then configure the webserver to use the database server
- Our environment probably isn’t complex enough to need very complex playbooks
  - Our dependencies are linear
  - Very few “work on host A, then host B, and then host A again” workflows

Anatomy of a Repository

```
ansible.cfg                          # Config file for ansible commands
cluster-masters.yaml                 # Playbook file that covers all master nodes
inventory/                           # Inventory directory.  Contains system information.
    group_vars/                      # Group-specific variable definitions
        all                          # Variables applied to all systems
        ccstar                       # Variables applied to systems in the ‘ccstar’ group
        turquoise                    # Same, for the ‘turquoise’ group
    hosts.ccstar                     # Inventory of hosts on the ccstar network
    hosts.turquoise                  # Inventory of hosts on the turquoise network
    host_vars/                       # Host-specific variable definitions
        ba-master.lanl.gov/          # Variables that apply only to ba-master
        kit-master.ccstar.lanl.gov/  # Variables that apply only to kit-master
roles/                               # Definitions of role building blocks
    ntp/                             # The ‘ntp’ role.  Provides everything needed for ntp
        defaults/                    # Files that define default values of template variables
            main.yaml                #
        handlers/                    # Files that define handlers for callbacks used in tasks
            main.yaml                #   Handlers do things like restart services when needed
        tasks/                       # Files that define tasks
            main.yaml                #   Tasks do things like copy files and install packages
        templates/                   # Files that define templates
            ntp.conf.j2              #   Templates look like config files, but with variables
    slurm/                           # The ‘slurm’ role.  Currently empty, but similar in
                                     #   concept to the ‘ntp’ role
```

How Ansible Runs

- Run targeting kit-master
  - `ansible-playbook -l kit-master.ccstar.lanl.gov cluster-masters.yaml`
- Inventory file gets read in
  - `kit-master` is a member of masters, ccstar, and all groups
- `cluster-masters.yaml` is read in
  - the masters group is assigned the ntp role
- Tasks in the ntp role are read in
- Each task is run
  - Tasks are performed by the module specified in the task
- Handlers are run
- Done!

Running Ansible - The Commands

- Commands
  - `ansible-playbook` – Run one or more tasks via a playbook
  - `ansible` – Run a single task via the command line
  - `ansible-inventory` – Displays the compiled inventory
  - `ansible-vault` – Encrypts/decrypts files in the Ansible repository
  - `ansible-pull` – Perform a git (or svn) pull (or update) before running locally
  - `ansible-galaxy` – Interacts with Ansible Galaxy, an online role repository
- Plus some extras: ansible-config, ansible-console, ansible-doc

Running Ansible - *Safely*

- Ansible has safety options
  - `--check`
    - Do a dry run. Don't make any changes, but reports what tasks would have made changes
  - `--diff`
    - When changing a file, display a diff between the existing file and the new file
  - `--step`
    - Run interactively, asking for confirmation before running each task
  - `--syntax-check`
    - Do a sanity check of a playbook without running it

Running Ansible - Example Command Lines

- Run a playbook
  - `ansible-playbook -i inventory/ -l localhost all.yaml`
- Run a playbook in check mode
  - `ansible-playbook --check -i inventory/ -l localhost all.yaml`
- Run a playbook in interactive mode
  - `ansible-playbook --step -i inventory/ -l localhost all.yaml`
- Display a list of all variables Ansible knows about for a host
  - `ansible-inventory -i inventory/ --host kit-master.ccstar.lanl.gov`

Exercises: Homework

- Write tasks to manage some common packages
  - Install `bind-utils`, `git`, and `zsh`
  - It probably makes sense to do this in the `common` role
- Write a task to manage `/etc/motd`
  - Create a role that manages a static `/etc/motd` file
- Write a role to manage rsyslog
  - Install `rsyslog` package
  - Enable `rsyslog` service
  - Install `/etc/rsyslog.conf`

Questions?

Los Alamos

NATIONAL LABORATORY

EST. 1943

Los Alamos
NATIONAL LABORATORY
EST. 1943

Delivering science and technology
to protect our nation
and promote world stability

Cluster Monitoring Basics

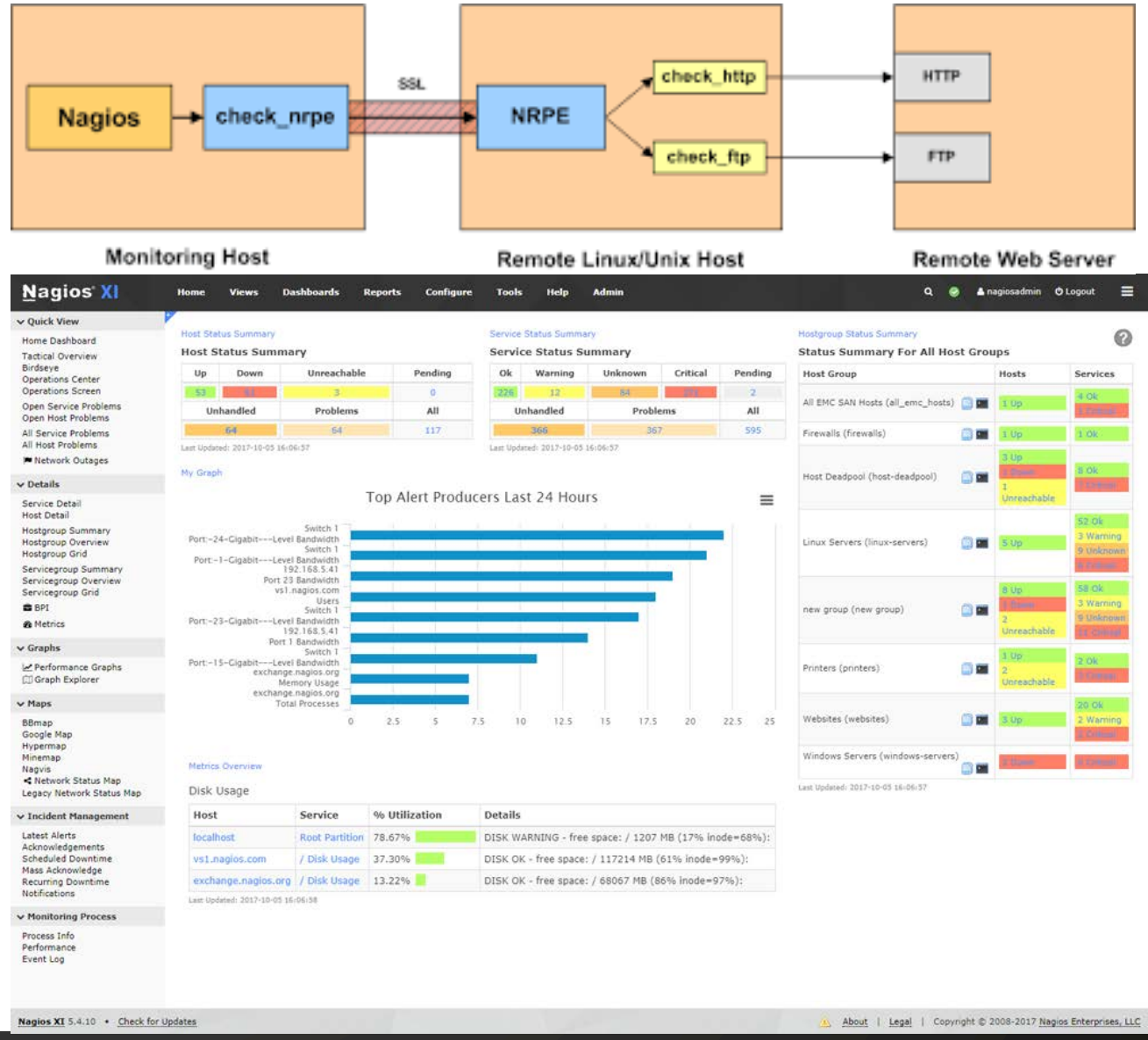
Presented by CSCNSI

Why monitor clusters?

- Detect failures in the system:
  - Did we get a transient failure that might have affected a job?
  - Did a service fail on our master?
  - Did we get a hard failure, like a hardware issue?
  - Are we getting consistent failures, like a design flaw?
- Learn about load and utilization:
  - How hard is our cluster working?
  - How balanced is our workload?
  - Who is using the most of what?
  - Are we consistently under CPU pressure? Memory pressure?...
    - Help to inform future design choices.

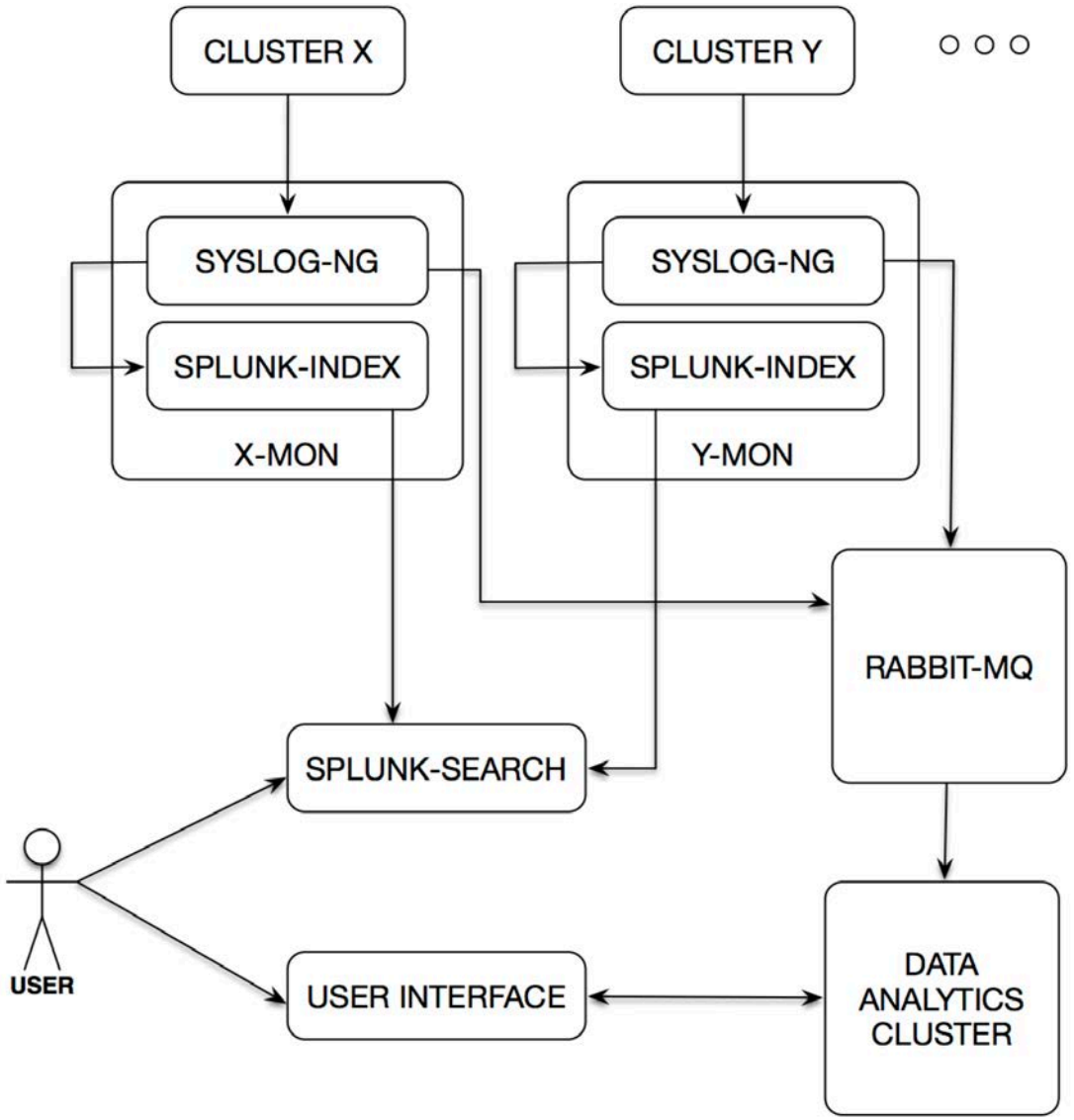
Active monitoring

- Used for:
  - Active polling for service states
  - Functionality checks
    - E.g. try to grab a VNFS image; does it succeed?
- Good for:
  - Getting early indication of failure
  - Things that naturally fit a “pull” model
- Tools that do this:
  - Nagios
  - Zabbix



Passive monitoring

- Used for:
  - Collecting data that already exists
    - System logs
    - Telemetry data, e.g. power utilization, system load…
  - Collecting system usage info
  - Collecting system errors
- Good for:
  - Passive alerts of system failure
  - Tree-like scaling
- Tools that do this:
  - LDMS
  - Splunk
  - rsyslog



Monitoring Infrastructure: The Challenges of Moving Beyond Petascale. – A. Bonnie, LANL

Scaling & performance concerns

- Latency
  - The bigger we grow and more we “fan-out” our monitoring, the more delay before we get a message
  - If we want to actively respond to events, we need to keep latency low
- Overhead
  - Any monitoring task will create a burden on the CPU &
- Jitter
  - Work on the network, CPU, etc. can cause brief interruptions for research applications
  - Routine interruptions can lead to significant over-all loss in performance

- Moving “out-of-band” to reduce Overhead & Jitter
  - We can keep monitoring network traffic to dedicated networks
  - We can even get some data through the BMC instead of the CPU

Active vs. Passive Monitoring

- Pro
  - Active detection of failure
  - Arbitrary scripted checks
- Con
  - Does not “fan-out” well
  - Can cause a lot of network and CPU traffic (high overhead & jitter)

- Pro
  - Great for collecting telemetry data
  - Tools like RabbitMQ/LDMS support fanning out to very large scale
  - Can be coordinated out-of-band to reduce jitter
- Con
  - Can produce huge amounts of uninteresting data
  - Alerts and analytics produced post-collection

Questions?

Los Alamos

NATIONAL LABORATORY

EST. 1943

Los Alamos
NATIONAL LABORATORY
EST. 1943

Delivering science and technology
to protect our nation
and promote world stability

Cluster Benchmarking

Presented by CSCNSI

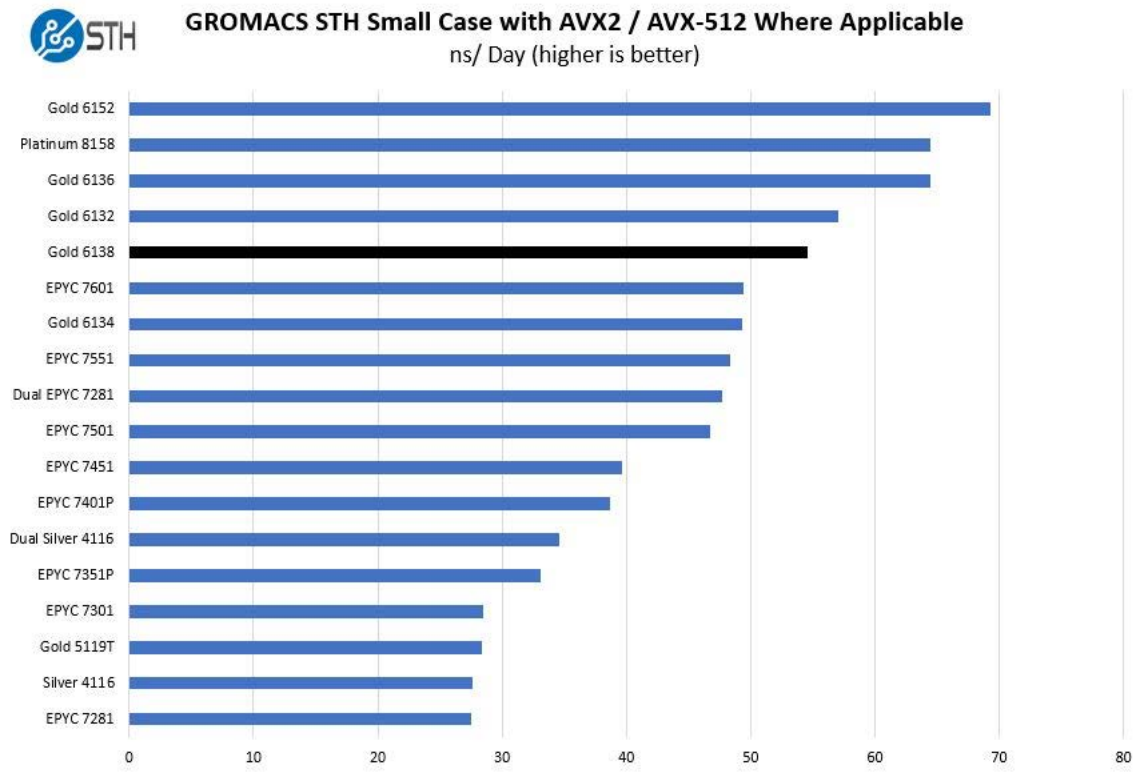
Why Benchmark?

Benchmarking is the processes of collecting quantifiable data on system performance

- Benchmarks can help us:
  - Evaluate comparative system performance
  - Identify bottlenecks for application performance
  - Estimate performance of an application on a given system
- Benchmarks are often a required as part of the “acceptance” process
  - Newly purchased systems are often contracted to reach a certain SSI
    - SSI = “Scalable System Improvement”, i.e. how much faster is this system than the old one?
- Benchmarks can sometimes be an early warning sign that something is wrong with a system, either in design or in function

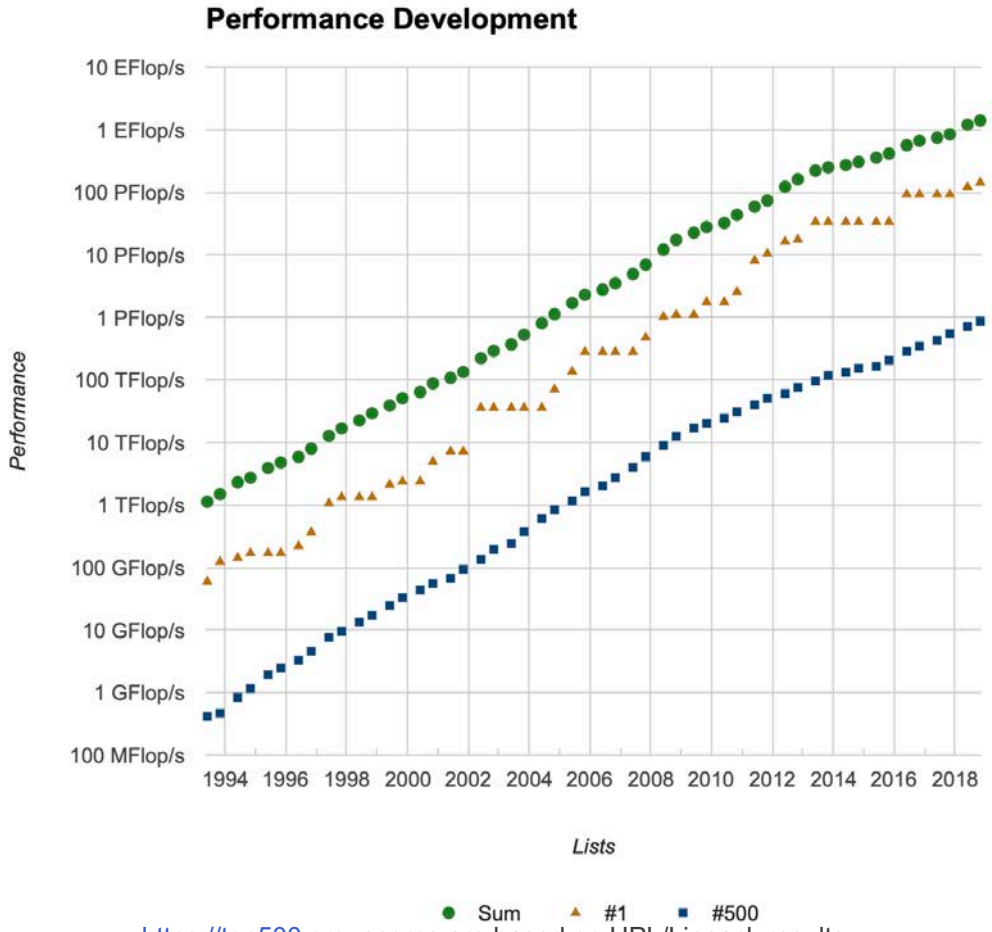
“Real” benchmarks

- The best way to know how a system will perform with an application is to run the application!
- Often, the data we collect with a real benchmark is simply the time it takes to run an application
- There can be problems with real benchmarks:
  - We may not have tuned or ported our application for the new platform
  - Results with one specifically build or data set may differ from another
  - We may not have the software available
  - They don’t always tell us why an application performance better or worse



Synthetic Benchmarks

- Synthetic benchmarks tend to focus on a class of algorithm, or particular hardware function
- Synthetic benchmarks can help understand which components perform better and worse
- Synthetic benchmarks have problems too:
  - It can be hard to predict how a specific complex application will perform
  - Systems can be tuned for synthetic benchmarks and ignore performance that really matters



https://top500.org: scores are based on HPL/Linpack results

Hybrid benchmarking

- Generally we want a combination of real and synthetic benchmarks
- Using a hybrid approach can give us:
  - High confidence that specific applications will perform
  - Understanding of which components help/hurt performance

Synthetic “micro” benchmarking

- Test a specific component or metric
- Examples:
  - Test network fabric latency/bandwidth
  - Test memory bandwidth
  - Test CPU vector operations
  - Test disk seek time
  - GPU performance
  - …

- Common Tools:
  - Netperf (TCP networks point-to-point)
  - ib_{send/recv}_{bw/lat} (Infiniband point-to-point)
  - Bonnie++ (filesystem performance)
  - Stream (memory performance)
  - Intel MPI Benchmarks (“IMB”, MPI operations bencharking)

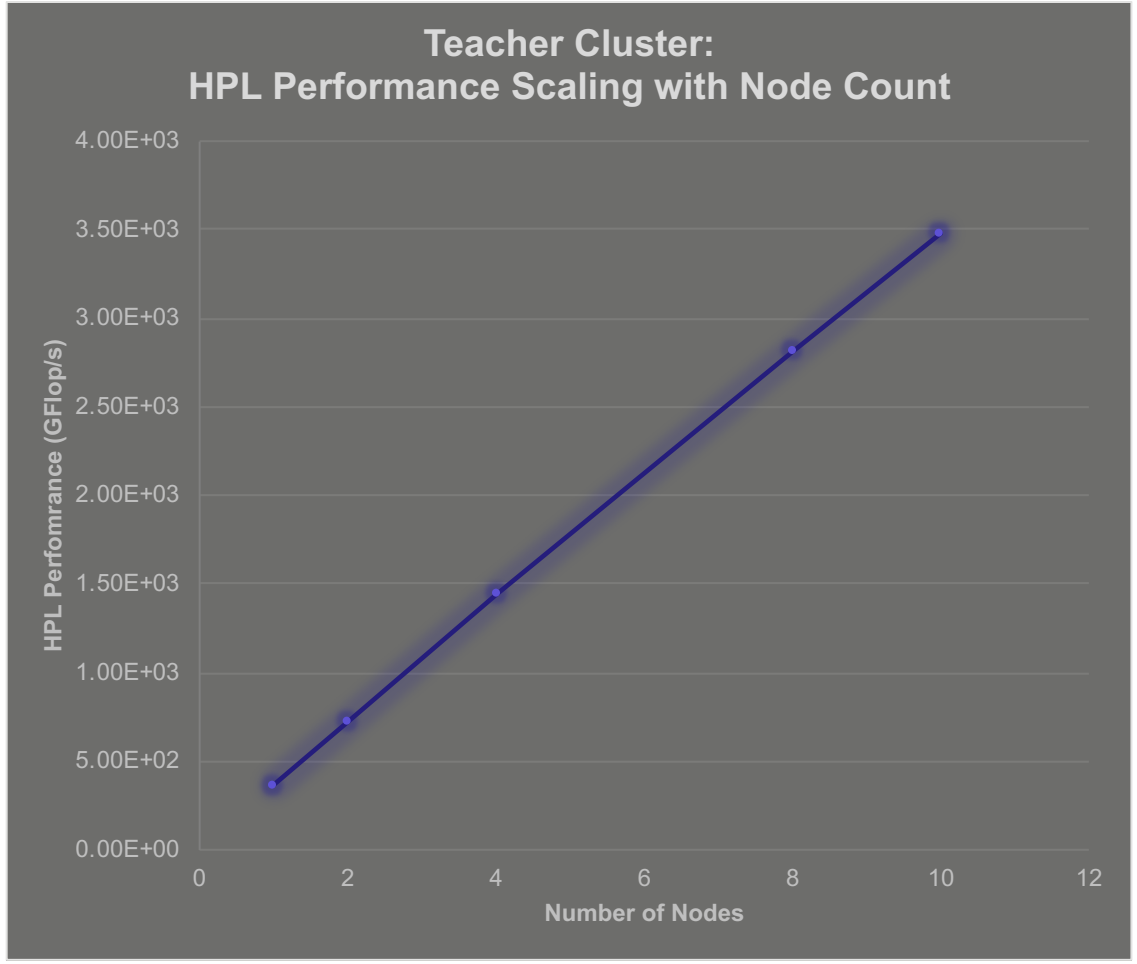
Synthetic “macro” benchmarking

- Macro benchmarks tend to try to solve a typical *kind* of problem that can tell about overall performance
- Often solving mathematical problems:
  - Linear algebra
  - Differential Equations
  - Graph theory
- High scores on these when you places on the big lists like:
  - Top500.org
  - Green500.org
  - Graph500.org
  - …

- Common Tools:
  - HPL (“High Performance Linpack”, distributed matrix operations, standard for top500.org)
  - HPCG (“High Performance Conjugate Gradients”, tests various linear algebra applications)
  - DGEMM (dense matrix, low communications overhead)
  - SMG2000 (Linear PDE solver)
  - …

Benchmarking scalability

- A common technique in both real and synthetic (mostly macro) benchmarking
- Focuses on how linear the quantity scales while increasing parallelism
- Is often critical to knowing how big to build a system
- Often coupled with profiling tools (when feasible) to understand why scaling breaks down



Some things to keep in mind…

- You probably want to build benchmarks for the hardware you're on
- Understand what the tool does, and how to tune it
- Keep your results well organized
- Try building against different libraries/tools:
  - Linear algebra: OpenBLAS, ATLAS…
  - Compilers: GCC, Intel, PGI…
  - MPI: OpenMPI, IntelMPI, MPICH…

Questions?

Los Alamos

NATIONAL LABORATORY

EST. 1943

Los Alamos
NATIONAL LABORATORY
EST. 1943

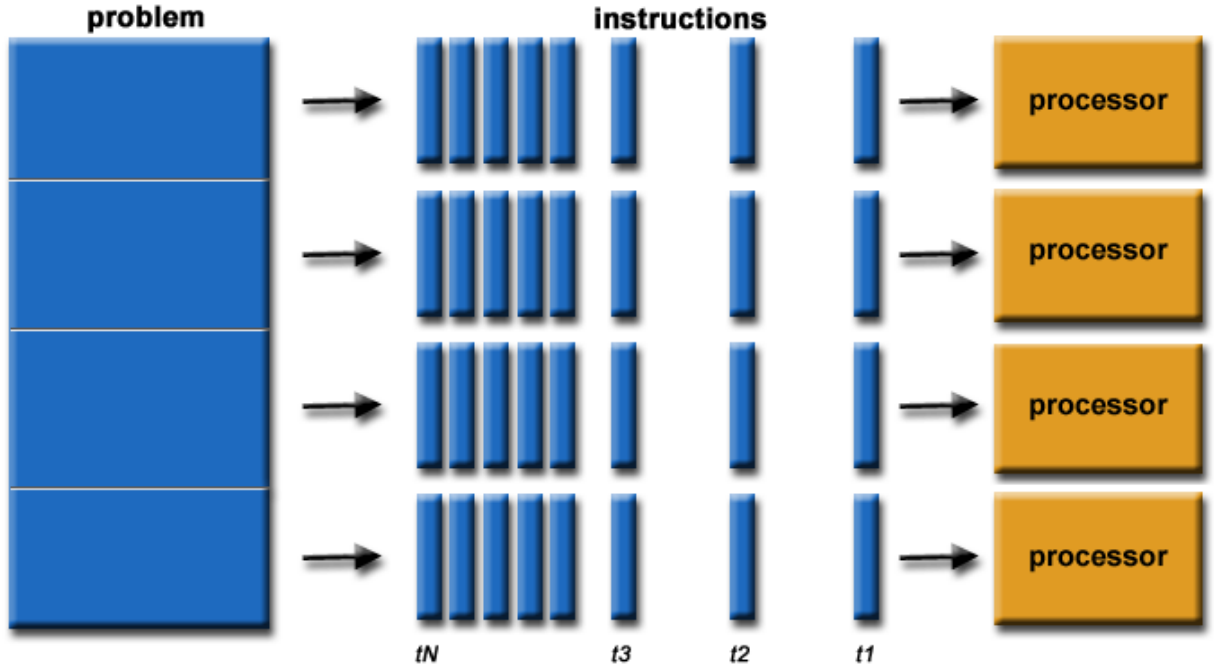
Delivering science and technology
to protect our nation
and promote world stability

Parallel Programming

Presented by CSCNSI

Working in parallel

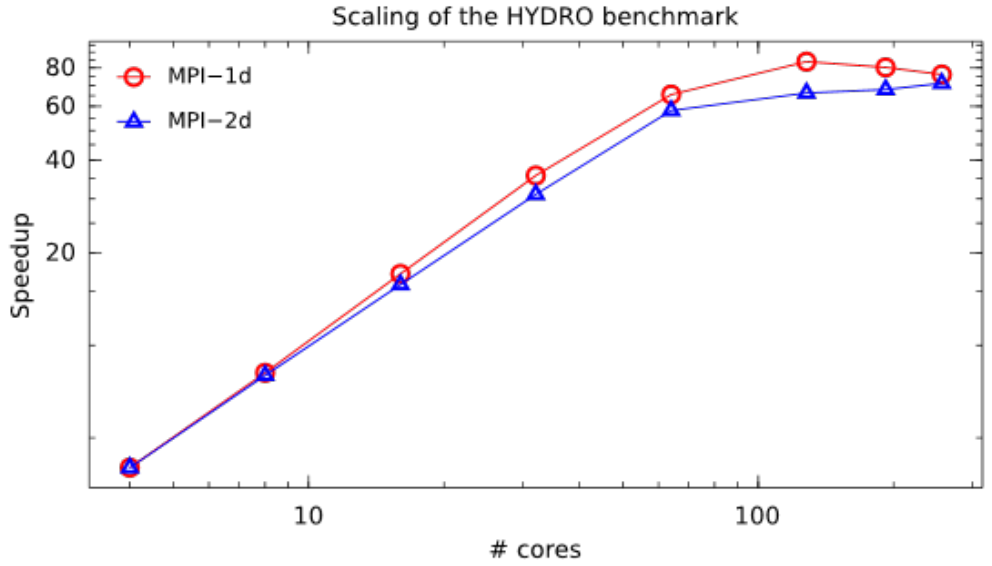
- Processors aren't getting faster
- But we are getting more of them
- We have to find ways to work in parallel
  - Performance tuning = parallelization
- Modern optimization is largely the process of making algorithms more parallel

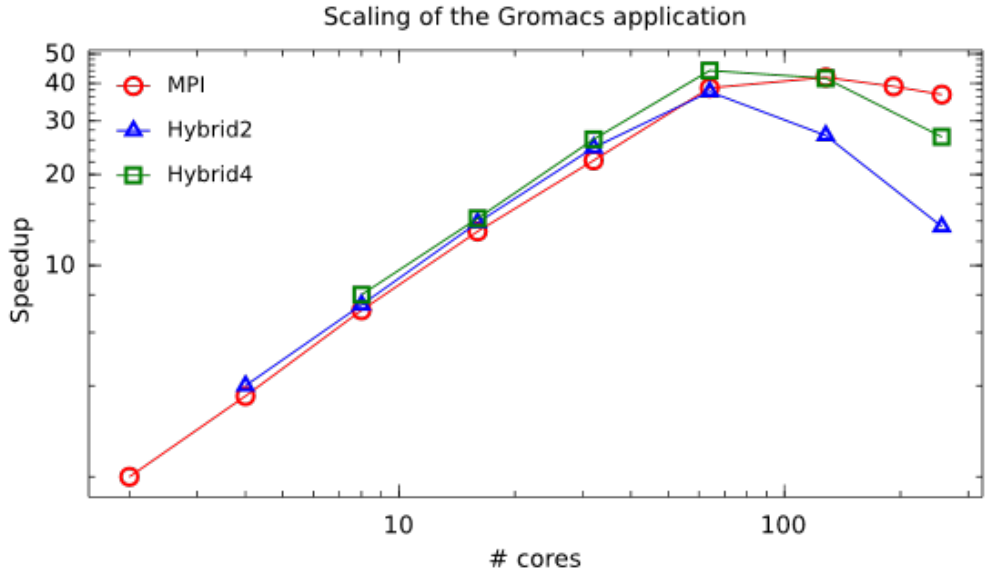


https://computing.llnl.gov/tutorials/parallel_comp/

Parallel Scaling

- Scaling is hard
- Usually we reach a plateau
- Things that can get us over the plateau (roughly in order of efficacy)
  - Rethinking our algorithms
    - Sometimes requires rethinking scientific approximations
  - Writing better parallelism
  - Improving hardware level paralellism

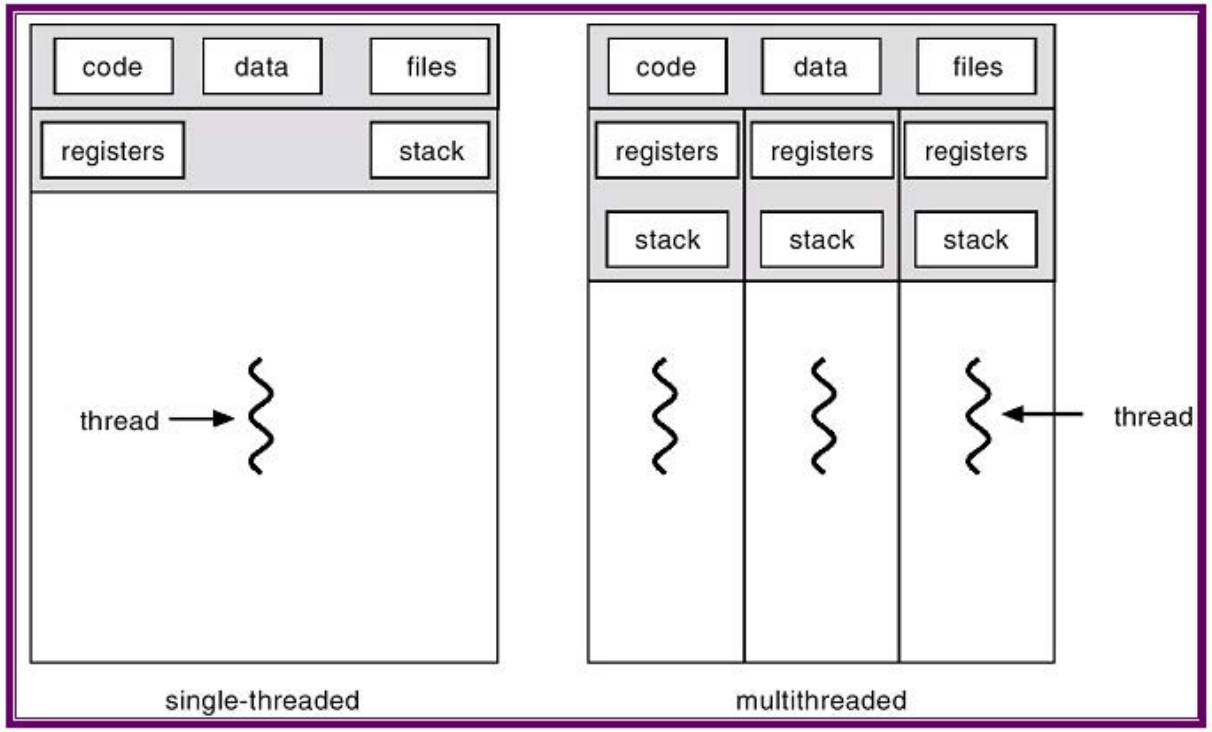




http://www.prace-ri.eu/best-practice-guide-knights-landing-january-2017/#scaling-and-speedup-NPB

Threads

- Threads are lightweight, possibly parallel processes
- They ***share*** their memory (except stack)
- If done well, threads lead to huge improvements on multi-processor hardware
- If done poorly, threads can cost more than they save
  - ...and cause huge headaches.



http://www.csc.villanova.edu/~mdamian/threads/posixthreads.html

Structure of a typical threaded program

1. Create a thread that is set to run a defined function at start (“create”)
2. Start the treads (“start”)
   1. The threads to work
   2. The main thread can do work too
3. Wait for threads to finish (“join”)

- The threads often call the same function
  - but do different work based on ID

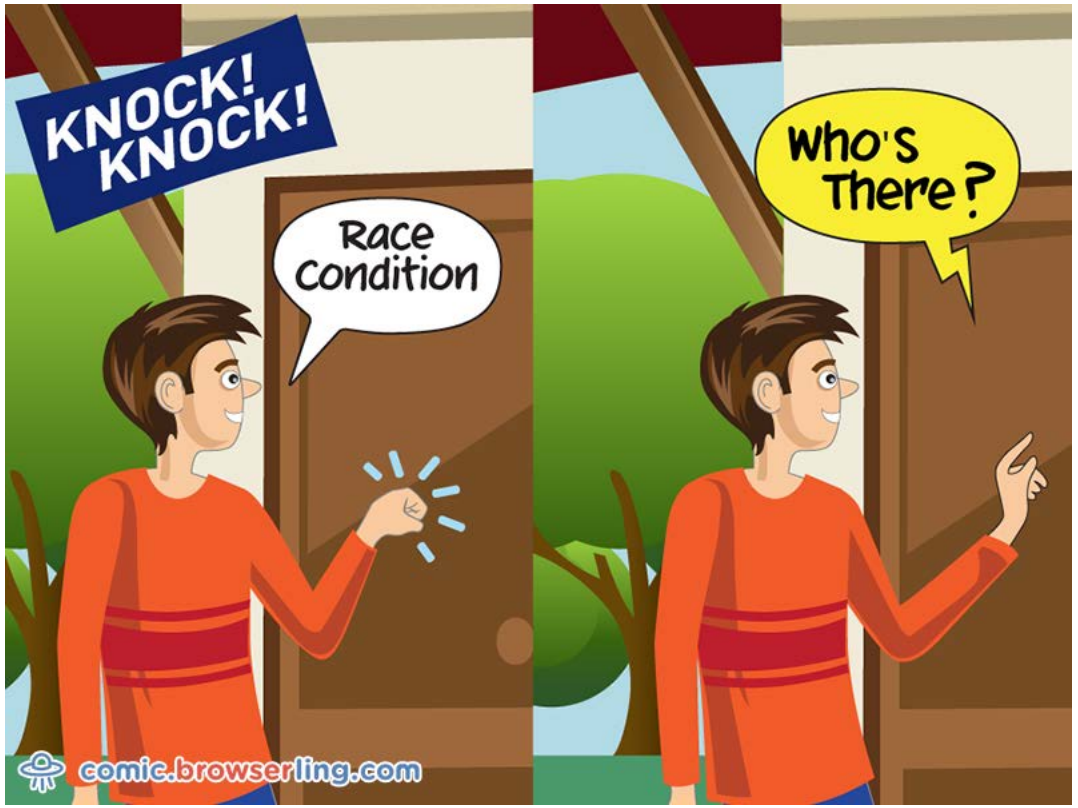
Psuedo-code for threading structure

```
function runner() {
    id = get_thread_id()
    do_some_work(id)
}

function main() {
    t1 = pthread_create(runner)
    t2 = pthread_create(runner)
    t1.start()
    t2.start()
    /* do something */
    t1.join()
    t2.join()
}
```

Race conditions

- Suppose t1's job involved writing data to a the file *data*
- While t1 is doing this, t2 is computing some data to add to the file
- After computing t2 reads the file and uses that data to finish its work
- *Usually* t2 takes long enough that *data* is ready
  - *...but what if it's not!*

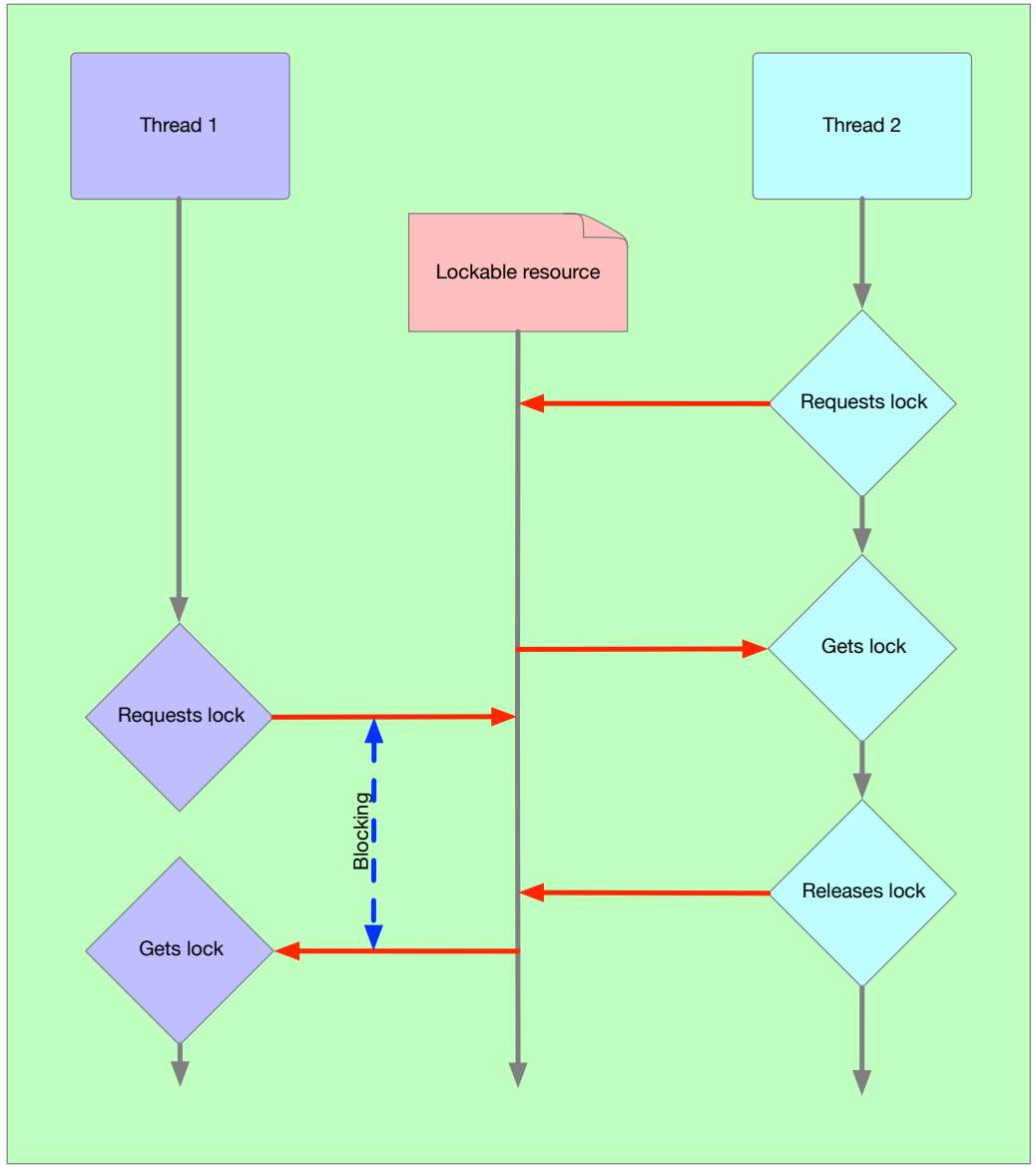
- **A race condition exists when success of an algorithm depends on one thread/process "winning the race" against another.**



Locking: Overview

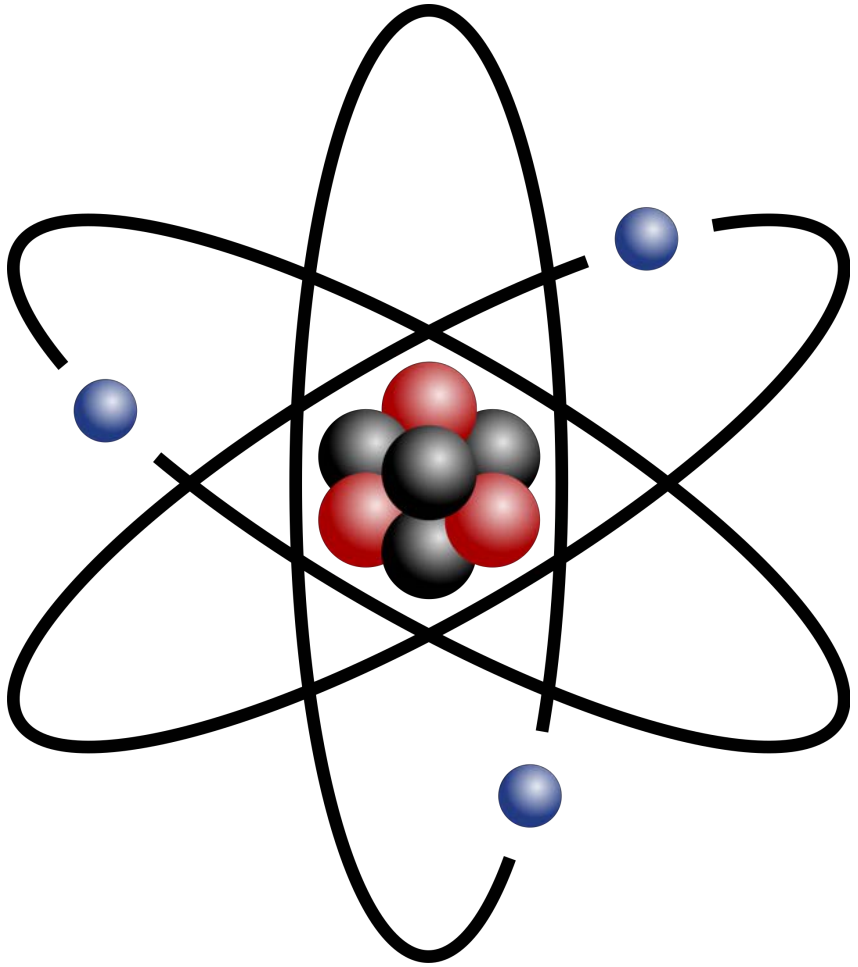
- How do we deal with things like race conditions?
- In our example, if:
  - t1 had a way to maintain exclusive access to *data*
  - ...and t2 would wait until that access was dropped.
    - Then there wouldn't be a race.

- This is an example of Locking.
- Locking can be used to synchronize concurrent computing processes/threads.



Atomic operations

- An “atomic” operation is guaranteed to complete without interference from any concurrent process.
- Typically, this means they take exactly one instruction cycle on the CPU, making them uninterruptable.
- Atomic operations are the necessary basis for locking & synchronization
  - If we want to lock something, we need to know the the operation of locking isn’t itself a race
    - ...what if two things call lock at the same time?



https://en.wikipedia.org/wiki/Atomic_mass

Locking: Mutex

- A Mutex has two operations:
  - Lock()
  - Unlock()
- Stores a reference in shared memory
- Uses atomic operations to read/set whether the lock is held
- If you try to Lock() an already locked mutex
  - ...you'll block until it gets Unlock()ed by someone else

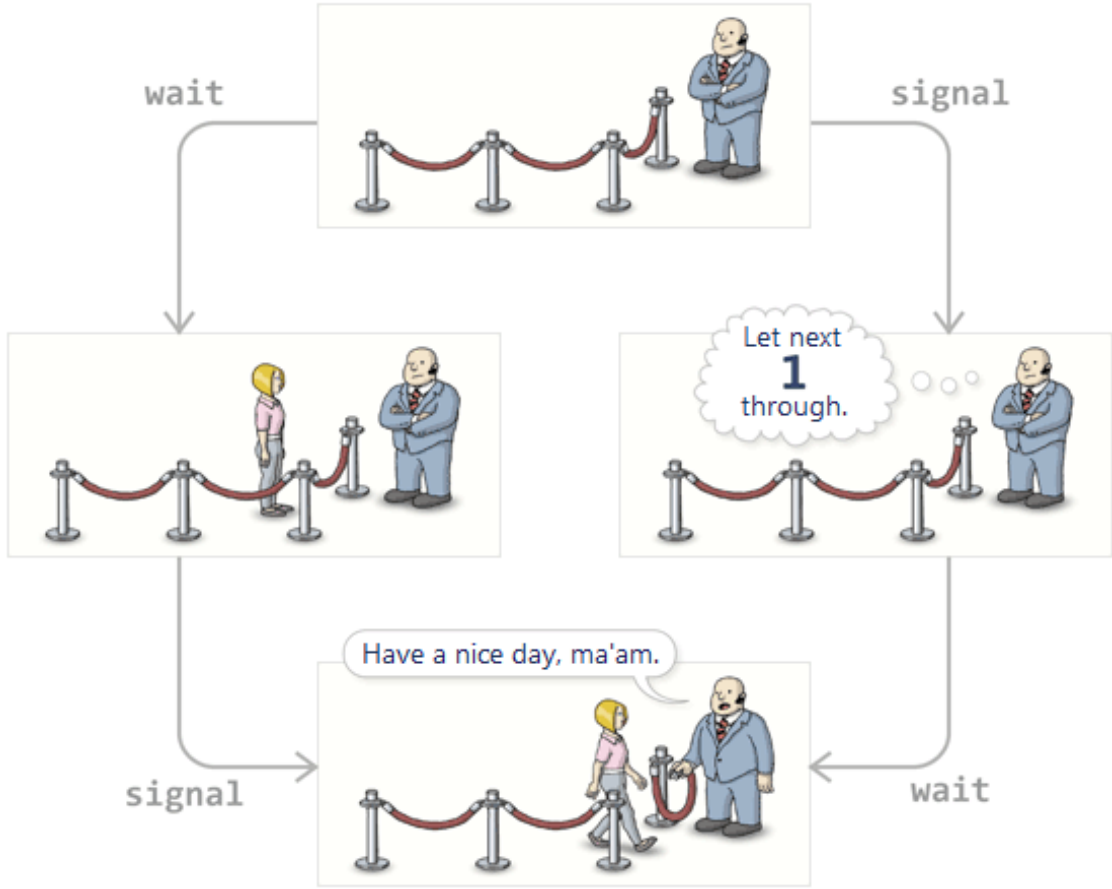
https://medium.com/@the1mills/a-better-mutex-for-node-js-4b4897fd9f11

Locking: RWMutex

- An RWMutex works like Mutex
- But:
  - There can be any number of RLocks() (read)
  - There can be only one WLock() (write)
  - WLock() and RLock() are mutually exclusive
  - Once WLock() is requested, new RLocks() block
    - ...once all RLock()s RUnlock(), WLock() is granted.

Locking: Semaphore

- Two operations, and an integer:
  - Wait() decrements the integer by 1
    - And waits to be signaled
  - Signal() incriments the integer by 1
    - If the integer goes from < 0 to >= 0, waiting processes are signaled
- Like Mutex, based on simple atomic add/subtract atomic operations.
- Can be used to implement things like RWLocks



https://preshing.com/20150316/semaphores-are-surprisingly-versatile/

Locking: ...but wait! There's more...

- Barriers: everybody must pause until we all call Barrier (synchronize).
- Conditional locks: lock until a condition is met.
- SpinLock: Like locking, but instead of blocking, pole for the lock.
- Event Locks: Block until this event is raised.

Deadlocks

- Example:
  1. run1() locks t and calls run2()
  2. run2() tries to lock t, and blocks
     - run2 never exits
     - run1 never continues to t.Unlock()

- This is a *deadlock*.
- A deadlock is a locking condition that has circular dependencies.
- Mostly avoided by careful locking symantics.

Psuedo-code for deadlock

```
function run1() {
    t.Lock()
    run2()
    t.Unlock()
}

function run2() {
    t.Lock()
    do_stuff()
    t.Unlock()
}
```

Threads aren't the only way

- Multiple processes with inter-process communication
  - Resource expensive, but this is mattering less and less
  - Often use UNIX domain sockets for communication (a special kind of file)
- Thread abstraction layers:
  - OpenMP—C/C++ extensions that give simple parallelism
  - Goroutines—Specific to Golang, message passing between lightweight concurrent routines
  - Event-driven design—register handlers for events, raise events. Run handlers concurrently. This is actually how the kernel works.
  - Higher-level abstractions, e.g. Python work queues
- No shared memory at all, but message passing:
  - E.g. MPI (next time)

OpenMP: Overview

- Standard implemented as “pragmas” (compiler directives)
  - Can be turned on/off for flexible coding & debugging
  - Commonly included within the compiler itself
  - Can vary the number of threads without recoding
- Typically wraps an existing loop, and makes it run in parallel

Psuedo-code for OpenMP structure

```
#include <omp.h>
int main() {
    int cpu_id = -1;
    cpu_id = sched_getcpu( );
...
    #pragma omp parallel for
    for (int i = 0; i < 4; i++){
        printf(“value:%d,
                thread:%d\n”, i,
                omp_get_thread_num());
    }
...
}
```

MPI

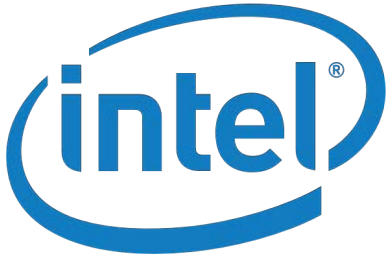
What is MPI?

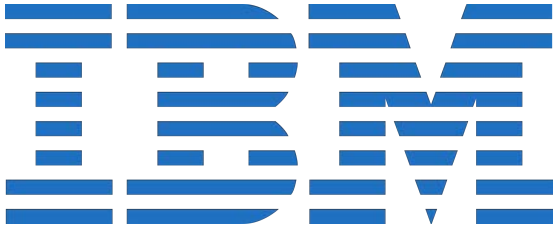
- “Message Passing Interface”
- A pluggable transport layer for sending bits of data to different processes
- Supports optimized transports for high speed fabrics (e.g. Infiniband)
- Allows efficient, fast inter-process communication across a cluster
- Allows easy porting between clustered platforms

MPI: History

- 1993 - Draft for standard submitted at supercomputing
- 1994 – MPI 1.0
- 1998 – MPI-2
- 2012 – MPI-3
- MPI-4 – Work in progress...
  - https://www.mpi-forum.org/mpi-40/

- Some implementations
  - OpenMPI
  - MVAPICH/MPICH
  - Intel MPI
  - IBM BG/Q MPI
  - IBM Spectrum MPI
  - Cray MPI

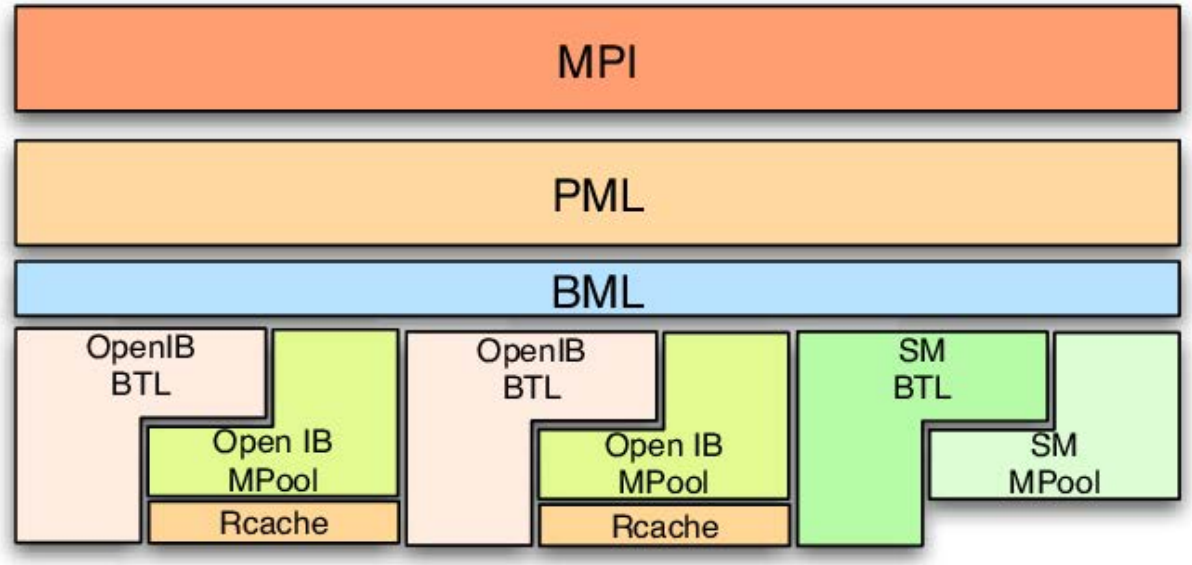




OpenMPI stack

- Layered model that abstracts hardware
- Supports various interconnects
- Full support for RDMA
  - ...maps remote memory to local system
- Also supports standard networks like Ethernet

OpenMPI Component Stack



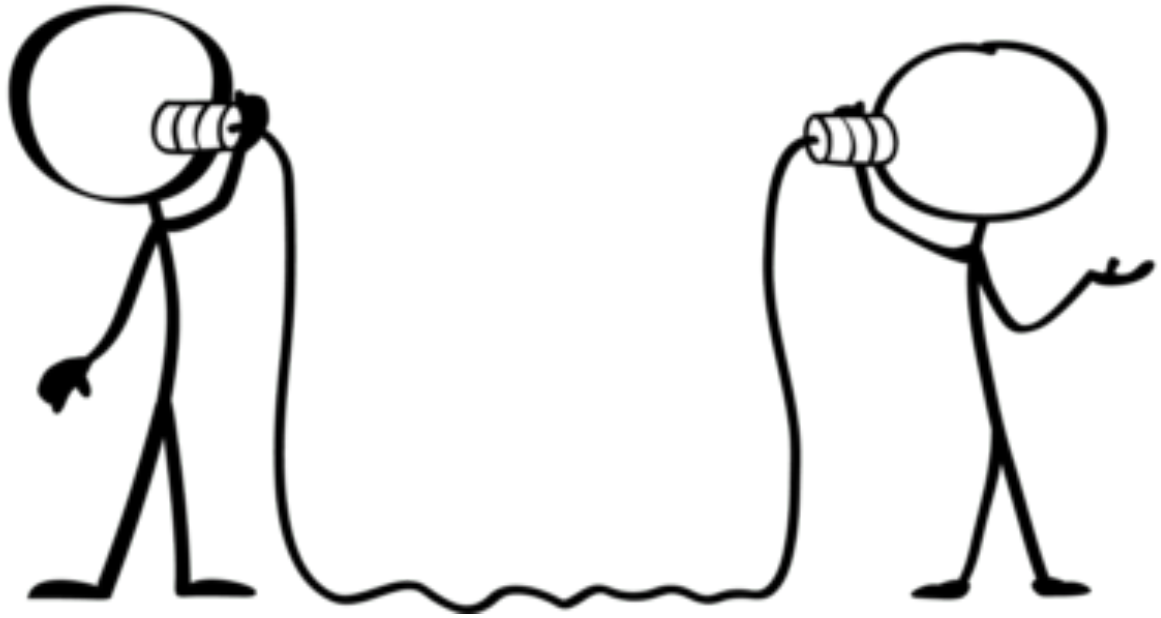
https://open-mpi.org

Terminology

- **Groups**—multiple processes form a group
- **Context**—a message sent in a context must be received in that context
- **Communicator**—the combination of a group and a context
- **MPI_COMM_WORD**—the default Communicator that talks to all processes
- **Rank**—one instance in an MPI process

MPI: Point-to-point operations

- **Send/Recv**: send & receive from specific rank to specific rank
  - Several variants:
    - isend/irecv: non-blocking
    - ssend/srecv: syncrhonous mode
- **Sendrecv**: send/recv, but both ends use the same function call
- Common use case:
  - Send information to immediate neighbors in grid



https://www.weshigbee.com/direct-communication-isnt-what-it-seems/

MPI: Collective operations

- Collective operations are some of the most useful parts of MPI
- Allow for coordination of a large number of ranks trivially
- Common uses:
  - Collect divide-and-conquer pieces of data into central location
  - Distribute work to workers
  - Update global state information

- **Scatter** – split a list and send a piece to each rank
- **Gather** – collect data from all ranks to one rank
- **Allgather** – collect data from all ranks to all ranks
- **Bcast** – send data from one rank to all ranks
- **Reduce** – reduce many values to one with a basic operation
- **Allreduce** – reduce many values to one with a basic operation, send to all ranks
- ...

Questions?