

WEB-401 - Cloud Server Administration

Lecture 1 - Course Introduction

About Me!

- My name is: Andrew Raymer
- I've been in the industry over 10 years
- Went from Systems Administrator to Developer to Lead DevOps and Project Manager
- I enjoy collecting and selling old video games, playing said games, Star Wars Lego, and listening to smooth jazz.



Course Outline

Syllabus

Course Details

Course Details

- 14 Weeks this semester
- 12 Module Assignments (Think Labs)
- 2 Projects
 - IaaS Deployment Project
 - PaaS Deployment Project

- 2 Case Studies
- 12 Quizzes
- 2 Tests
- Midterm Exam -> March 1st, 2022 @ 6:00PM
- Final Exam -> April 26th, 2022 @ 6:00PM
- A deduction of 20% of the item's total per day will be given for late assignments, projects, and case studies

Assignment Details

Assignment Details

- Assignments are given out each class via blackboard
- Assignments are to be submitted over blackboard
- Assignments are due the night before before the next weeks class (11:59:59 PM EST)

Project Details

Project Details

- IaaS Deployment Project will be given out on February 1st, 2022
- IaaS Deployment Project is due February 22nd, 2022
- PaaS Deployment Project will be given out on March 22nd, 2022
- PaaS Deployment Project is due April 12, 2022

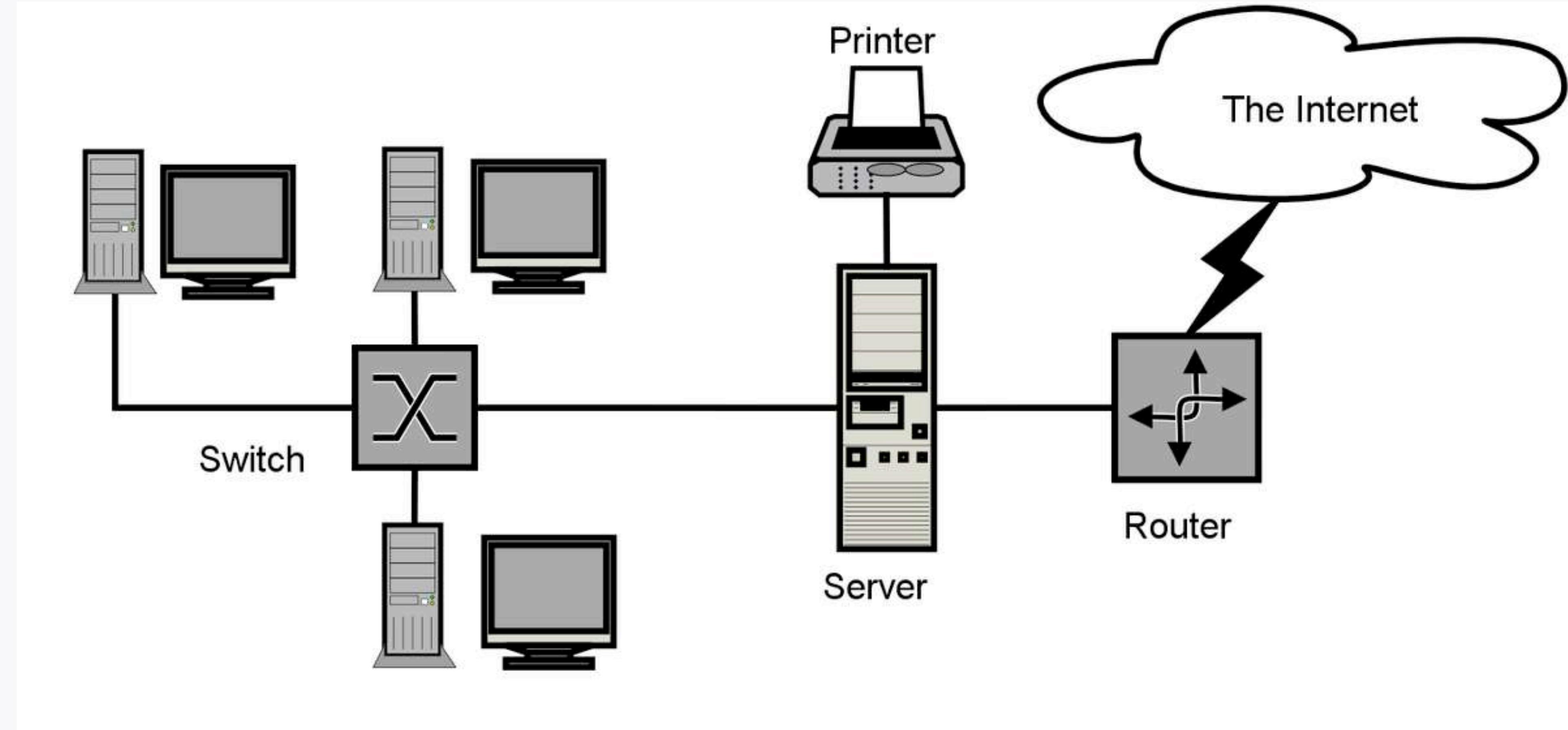
Important Dates

Week Number	Class Date	Lecture Topic	Assignment	Quiz	Projects
1	January 18th, 2022	Course Introduction	Assignment #1	Quiz #1	
2	January 25th, 2022	Introduction to Cloud Providers	Assignment #2	Quiz #2	
3	February 1st, 2022	Identify Management	Assignment #3	Quiz #3	IaaS Deployment Project Given
4	February 8th, 2022	Cloud Virtualization	Assignment #4	Quiz #4	Case Study #1 Given
5	February 15th, 2022	Linux Management	Assignment #5	Quiz #5	
6	February 22nd, 2022	Linux Automation	Assignment #6	Quiz #6	IaaS Deployment Project Due
7	March 1st, 2022	Midterm			
8	March 7th, 2022	Cloud Storage	Assignment #7	Quiz #7	Case Study #1 Due
March Break	March 15th, 2022	March Break	March Break		
9	March 22nd, 2022	Cloud Networking	Assignment #8	Quiz #8	PaaS Deployment Project Given
10	March 29th, 2022	Database Management	Assignment #9	Quiz #9	Case Study #2 Given
11	April 5th, 2022	Deployment	Assignment #10	Quiz #10	
12	April 12th, 2022	Autoscaling	Assignment #11	Quiz #11	PaaS Deployment Project Due
13	April 19th, 2022	Serverless Servers	Assignment #12	Quiz #12	Case Study #2 Due
14	April 26th, 2022	Final Exam			

What is the “Cloud”?

What is the “Cloud”?

- The cloud can be a nebulous term to nail down.
- Originally it was used on Network Diagrams to Symbolize the Wide Area Network (WAN)
- The term rose to popularity in the mid-2000s as a way to describe people accessing software hosted elsewhere from their web browsers



- But the long and short of it is, the cloud are just servers being hosted by a 3rd party in a datacenter
- It's a way of cutting out the cost of having to own and manage the “Iron” and instead you pay a small monthly premium for someone else to do it



**There is no cloud
it's just someone else's computer**

Dot Com Bubble

Dot Com Bubble

- In 1993 we saw the release of the “Web Browser” which popularized the use of the internet
- Between 1990 and 1997 the amount of American households with a computer increased from 15% to ~35% as computers moved from a luxury to a necessity
- This marked the shift to the “Information Age”

- At the same time of the rise of the Computer we saw a decline in interest rates which increased the availability of Capital
- This, along with some law changes, allowed investors to make more speculate investments
- From all these factors, and some luck, we saw investors throw large sums of money to anyone who had a “dot-com” company
- Basically any company with a url ending in .com

- Since word of these investments got out it fuelled even more speculation which encouraged even more investing
- Between 1995 and 2000 the Nasdaq Composite stock market index rose 400%
- Most of these “dot-com” companies operated at a net operating loss as they spent heavily in customer acquisition to build market share / mind share
- A common motto was **“get big fast”** and **“get large or get lost”**

Amazon

Amazon

- From this “dot-com” bubble, in July 1994, Jeff Bezos founded a little book selling business called Amazon
- He chose Amazon because search engines back then displayed results Alphabetically and it put it near the top of the list
- In 1997 Amazon went public to try and gather capital for expansion

- In 1998 they expanded from just books to selling music and videos
- In the early 2000s Amazon started building **merchant.com** as an e-commerce-as-a-service platform for third-party retailers to build their own web-stores
- This pushed them to engineer a **Service-Oriented Architecture** (micro services, essentially)
- They also forced all engineering efforts to be standardized, removed bureaucracy, and created a “shared IT platform” for the entire company to use

- Since Amazon started building this platform they started purchasing more and more servers
- Benjamin Black and Chris Pinkham wrote a short paper describing a vision for Amazon infrastructure that, in Black's words, "**was completely standardized, completely automated, and relied extensively on web services for things like storage.**"
- Amazon then launched "**Amazon Web Services**" which at the time was just a collection of API's and tools to access amazon.com
- In 2005 Amazon launched a private precursor to AWS which starts their plans for public releases of storage, compute, and database offerings

Virtualization

AUGUST 2014

Virtualization

- When you have a Datacenter filled with Racks you could rent them out one by one but that's highly cost ineffective and prone to a **Single Point Of Failure (SPOF)**
- Instead we leverage a technology called virtualization where the host computer (**hypervisor**) can run a virtual version of another computer and share its resources with it
- This allows a single Server's resources to be allocated to multiple smaller virtual computers which increases the efficiency ratio greatly

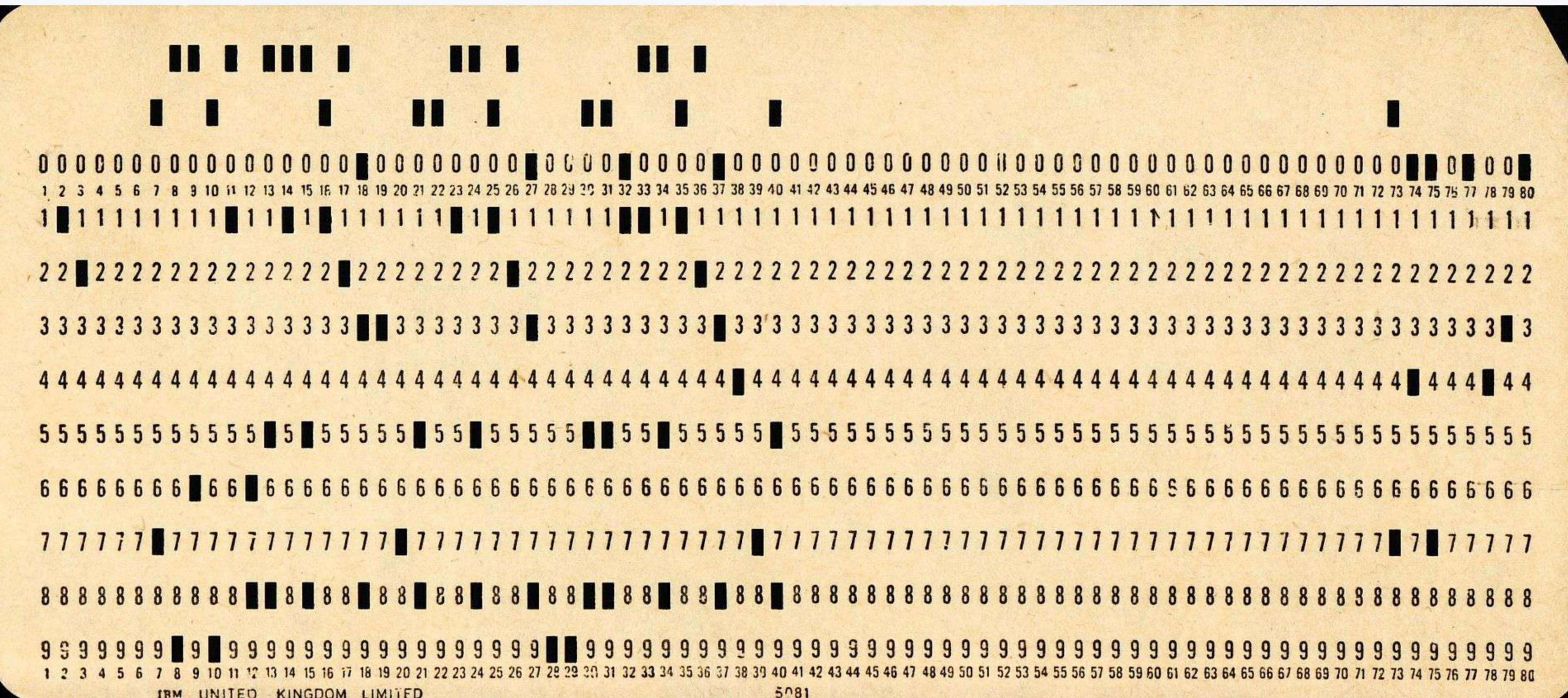
- We can even further abstract the Virtualization concept to where the virtual hard drive isn't even located on the same Server that is sharing it's resources
- With virtualization you can place the virtual hard drive on a **Storage Area Network** server (SAN) and have it mounted over the network to the server to present the virtual hard drive as if it was local
 - A common method of doing this is with using something called **iSCSI** targets

- The point of all this talk about Virtualization and SANs is that Amazon leverages its large volume of Servers and data storage to sell virtual machines (**EC2 Instances**) to its customers for an hourly fee
- They also expose an API that allows you to create via software any number of EC2 instances of any size and have AWS do all the provisioning and load balancing for you
- You just have to pay them a nice and tidy sum at the end of the month

Y2K

Y2K

- At the end of the 90's a new fear arose in the minds of people, the Year 2000 Problem or better known as "**Y2K**"
- The issue was that back in the early days of computing it was common practice to use only 2 digits to record the date
- Since most programs were written using punch cards, and memory was at a premium it made sense to save the extra few bits
- It was reported that the lowest cost was \$10 USD a kilobyte on a mainframe during the 70s but typically was \$100 USD a kilobyte



- As we neared the end of the millennium programmers started to notice that there would be an issue
- When December 31st, 1999 rolled into January 1st, 2000, some computers will incorrectly think the year is now 1900
- People panicked and falsely predicted that the world was going to end since we've computerized so much of our infrastructure at that point
- People started panic buying, and hoarding as much as they could

The Bubble Popped

The Bubble Popped

- Since some investments are based off of speculation, investors started getting nervous by all the spending and the Y2K Bug
- In 2000, Alan Greenspan, then Chair of the Federal Reserve, raised interest rates several times; these actions were believed by many to have caused the bursting of the dot-com bubble
- On Friday March 10, 2000, the NASDAQ Composite stock market index peaked at 5,048.62

- During early 2000 multiple big tech companies started merging such as AOL with Time Warner, and YaHoo! With eBay
- On March 20th, 2000, Barron's featured a cover article titled "Burning Up; Warning: Internet companies are running out of cash—fast", which predicted the imminent bankruptcy of many Internet companies which led many people to rethink their investments
- That same day, MicroStrategy announced a revenue restatement due to aggressive accounting practices
- Its stock price, which had risen from \$7 per share to as high as \$333 per share in a year, fell \$140 per share, or 62%, in a day

- On April 3rd, 2000 Microsoft was ruled guilty of Monopolization which led to a one-day 15% decline of the value of shares in Microsoft, which was an 8% drop in Nasdaq
- On November 9th, 2000 the much hyped pets.com went out of business only 9 months after IPO which marked the final nail in the dot-com bubble
- Luckily that little plucky startup Amazon survived the burst and still continued building up their market share

Total Cost Of Ownership

Total Cost Of Ownership

- Last thing I want to mention in the introduction is the **Total Cost Of Ownership (TCO)**
- TCO is the sum of the purchase cost of something plus the costs of operation
- When you're building a product for your company and you need to figure out the monthly costs, you need to account for all the hosting as well as development costs
- Lowering your TCO could mean life or death of a company / project

- Think of your car when you're thinking of TCO:
 - \$25K initial price, which is ~\$331/month
 - \$40/week for gas, average 4.3 weeks in a month makes \$172/month
 - \$150/month insurance
 - \$100/month preventative maintenance
 - All adds up to a TCO of about ~\$753

- A majority of modern applications are built using a Linux subsystem instead of Windows
- Windows is just too costly for most applications to be successful
- For example on AWS a t3a.xlarge (4 vCPUs, 16 GBs of RAM) has a On Demand Linux Rate of \$0.15/hour vs \$0.22/hour for Windows (not including MS SQL Server)
- That is roughly \$109.79/month for Linux vs \$163.52/month for Windows or 33% more per month

- If you went for a Web On Demand Windows MS SQL Database instance it would cost you for a t3a.xlarge \$0.29/hour or \$212.86
- That's for the basic version of MS SQL Server
- PSQL or MySQL are free to use... so that would net you the same rate of \$109.79/month
- So if you had to run say a dozen servers a month, which would you pick?
- Example: https://instances.vantage.sh/?cost_duration=monthly&selected=t3a.xlarge

- Another reason we use Linux over Windows is purely automation
- Windows has come along way over the years with **PowerShell** but it still doesn't meet the decades of tooling built for the default environment of **Bash**
- Through **CronJobs** and **Bash Scripting**, we can automate almost any task need doing on a server
- This is a tool you will be learning this semester

Questions?

Lecture 2 - Introduction to Cloud Providers

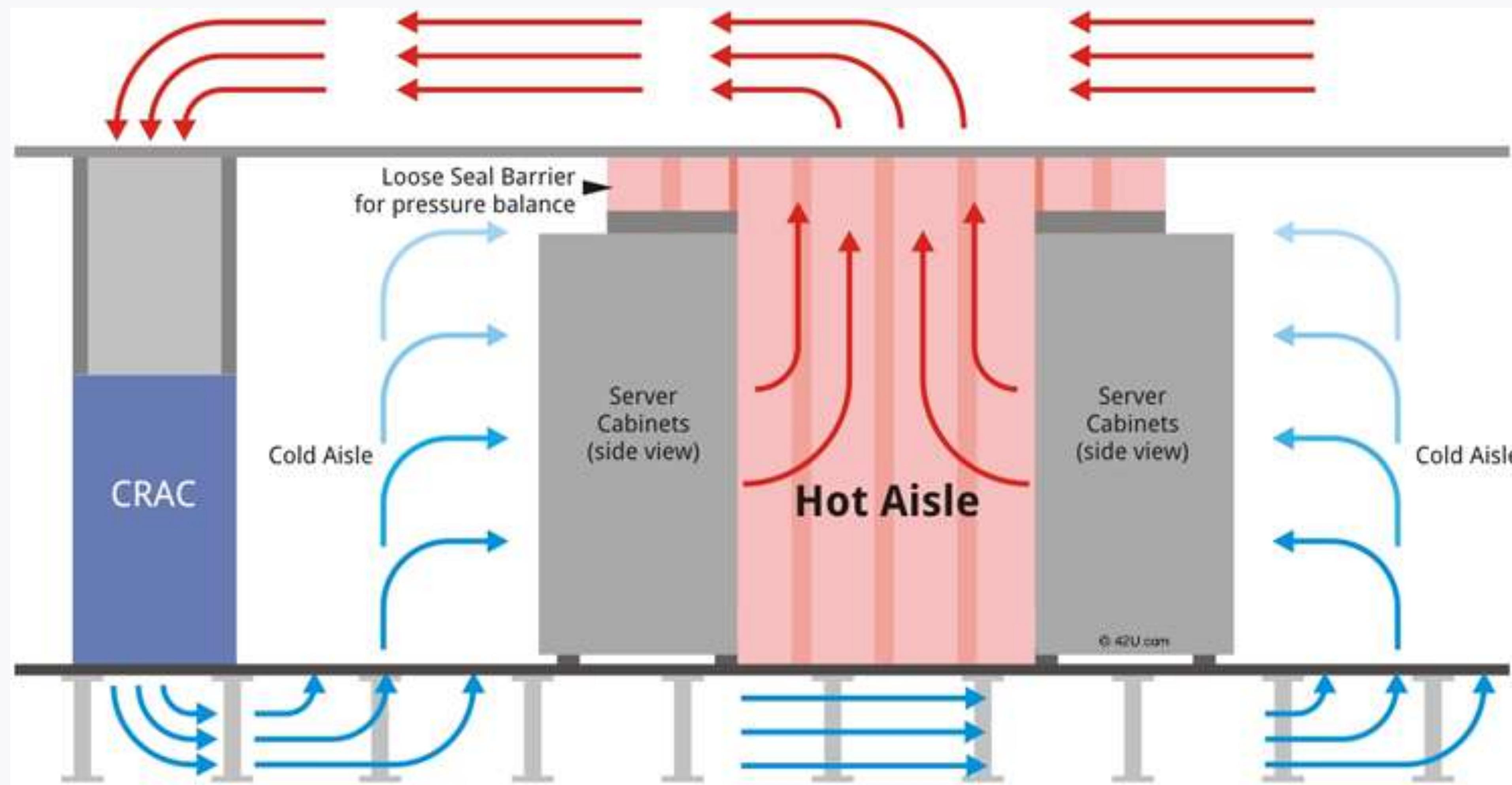
What is a Data Center?

What is a Data Center?

- A **Data Center** is a facility that centralizes one or many organizations shared IT Operations
- This secure facility has multiple incoming data connections, with multiple internet service providers (Tier 1 through Tier 3)
- It also typically has redundant power with on-site diesel generators and large Uninterrupted Power Supplies (UPS) which can keep the site running on batteries for a little while

- Due to the amount of electrical equipment, Data Centers typically employ non water based fire suppression systems
- For example it could be Inert Gas based, or CO₂ based or even some more exotic agents
- Fun fact; if you're in a DC and hear the fire alarm go off you need to run and get out of there ASAP or you could die due to fire suppression system
- See <https://www.youtube.com/watch?v=zb-5fU-ILgo>

- Data Centers also have very strong Air Conditioning requirements for server cooling
- Most Data Centers have a raised floor in which cold air is pumped by a **Computer Room Air Conditioner (CRAC)**
- Servers are in rows of Server Cabinets called Racks which all servers are suppose to face a single direction
- Cold air comes up through alternating aisles, which is then pulled through computers and blow into the other aisle
- This is called Cold-Aisle Hot-Aisle cooling



- In theory you could implement all of these requirements into a room inside your office, it tends to be costly at a low scale
- A lot of people opt to just rent space, a rack, from one of these Data Centers and allow them to manage all that underlying infrastructure

Racks

Racks

- As mentioned before Servers are typically slotted into standardized Server Cabinets called “Racks”
- Equipment designed to be placed in a rack is typically described as rack-mount, rack-mount instrument, a rack-mounted system, a rack-mount chassis, subrack, rack cabinet, rack-mountable, or occasionally simply shelf
- The height of the electronic modules is also standardized as multiples of 1.75 inches (44.45 mm) high by 19 inches (482.6 mm) wide or one rack unit or U
- The industry-standard rack cabinet is 42U tall



- Equipment attaches to the Rack via either rails that fit in the bolt cutouts along the sides or by Rack Nuts that also fit in the cutouts along the sides
- Again these are industry standard measurements for width and height
- Sometimes the depth of a Rack can be an issue so always make sure before you buy equipment







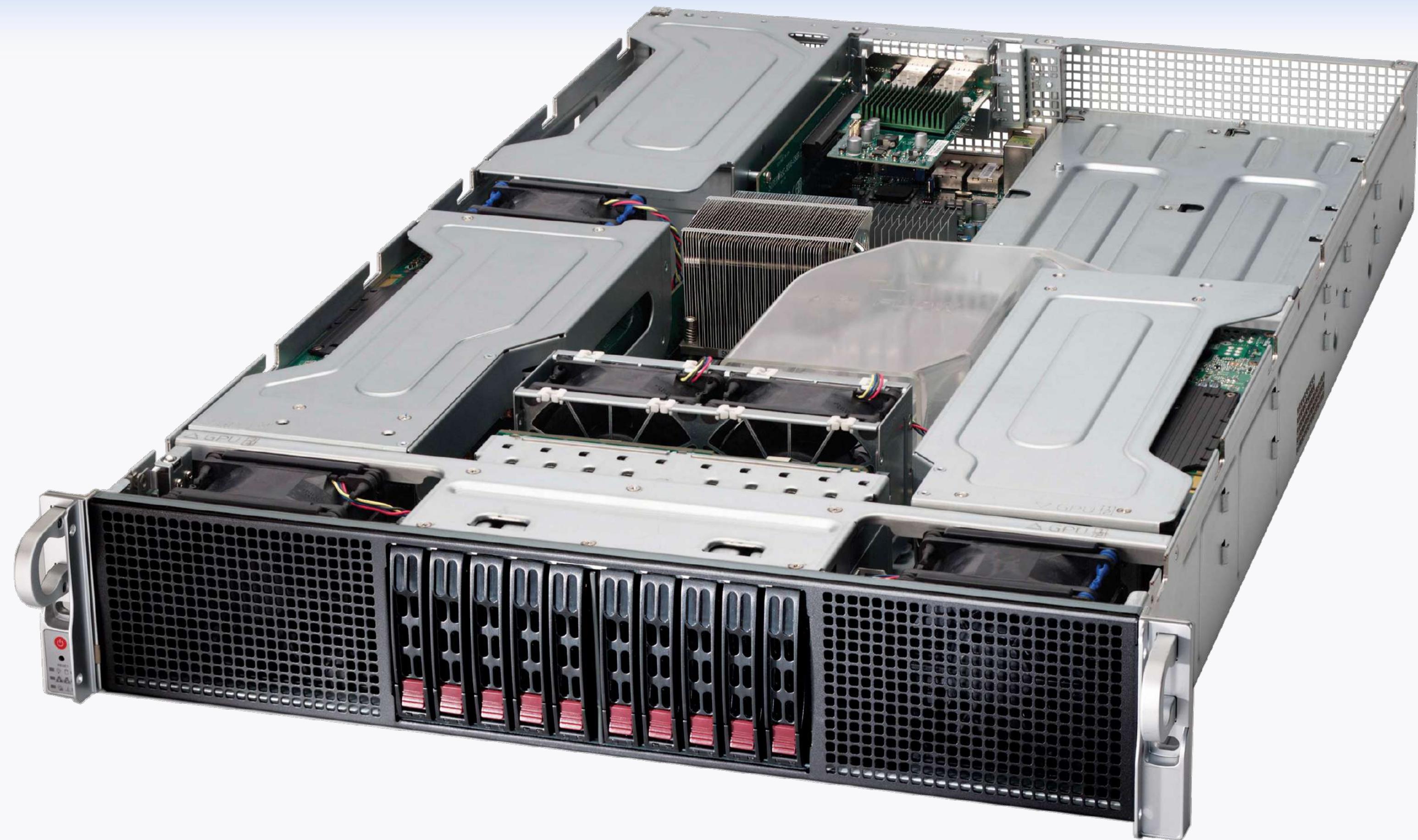
1U

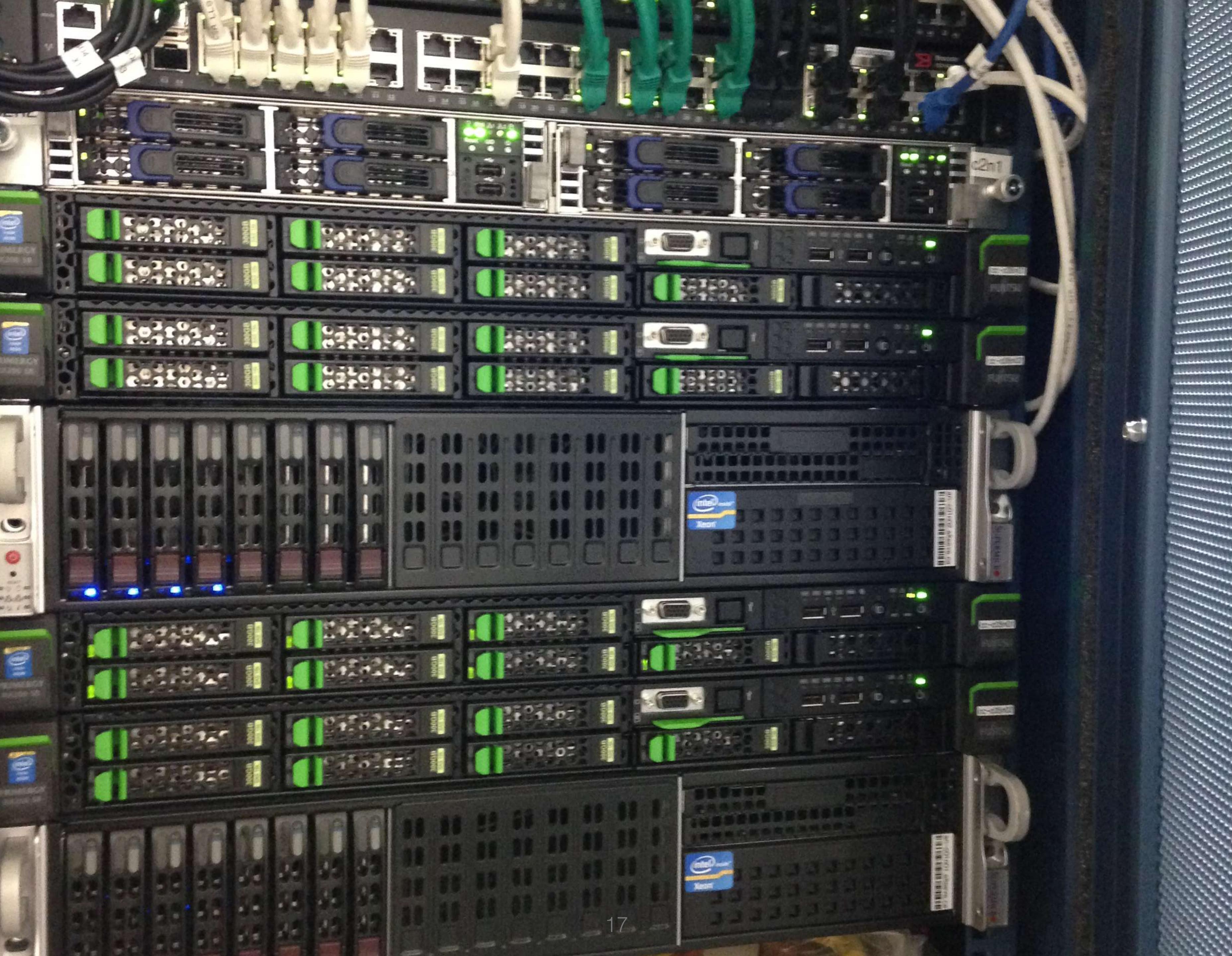
1U

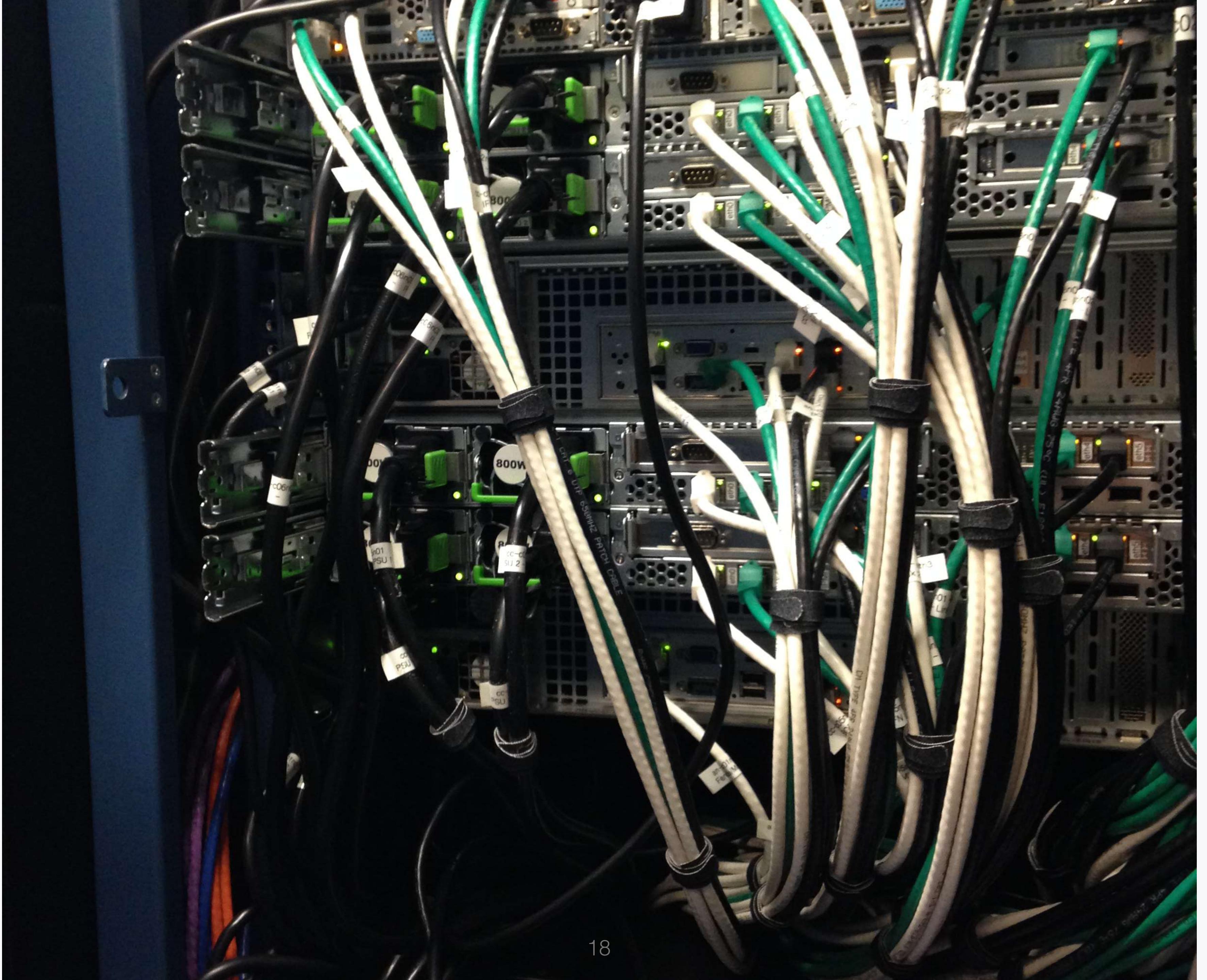
- When we describe the size of a server we always refer to how many U's it takes up in the Rack
- Typical server ranges between 1 - 4 Us



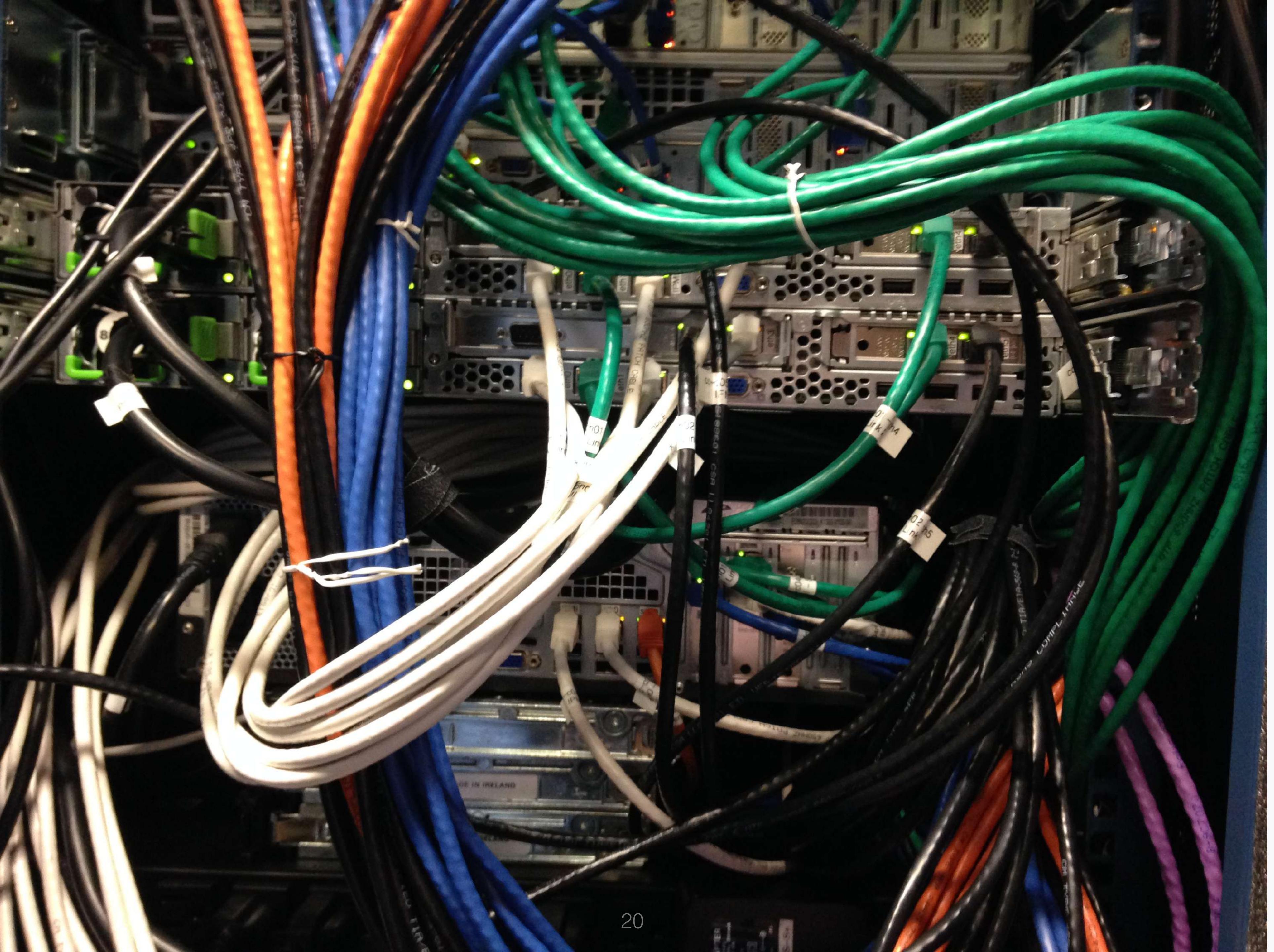
2U

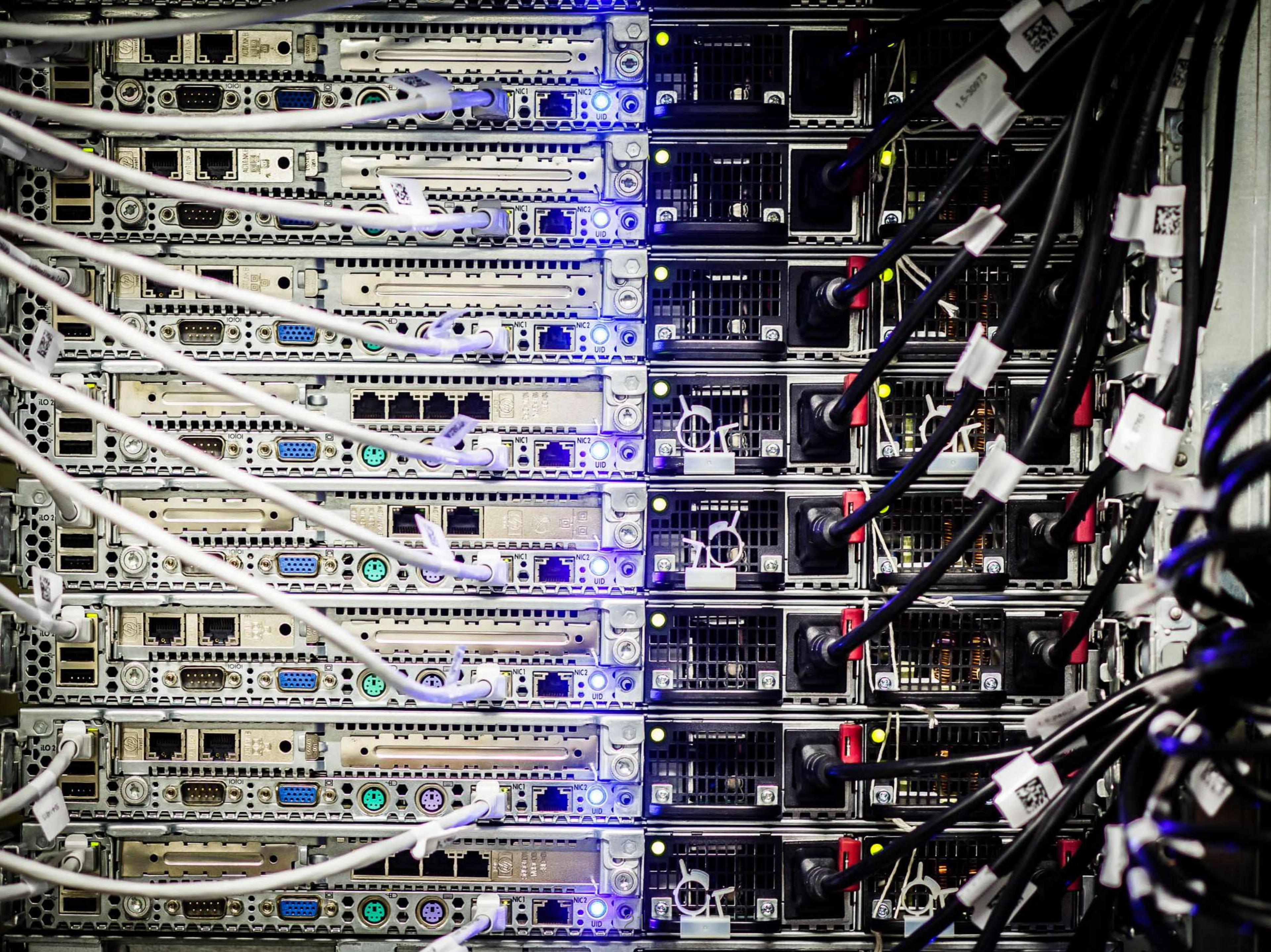












Why do you need to know all of this?

Why do you need to know all of this?

- Some employers don't trust clouds, for various reasons
- Sometimes a company will opt to purchase their own hardware and host it in a Data Center Colocation
- This is where DC's will opt to rent you out Rack space (whole or half) for so much a month with a 95th Percentile connection

- What a 95th Percentile connection means is that you're allowed to use as much bandwidth as your equipment can handle, for only 5% of the time
- If your 95th is rated at 100mbps then as long as 95% of the billing cycle you're < 100mbps you won't be charged extra
- If 95% of the time billing you're over that 100mbps threshold they will charge you an arm and a leg for data

- You may also need to know some of this information if your employer wishes to house the equipment in house
- You'll need to order Servers, Racks, UPS' and other equipment as well

Types of Cloud Providers

Types of Cloud Providers

- In the world of Cloud Computing there are many different types of Cloud Computing paradigms depending on the requirements
- For example, you wouldn't want the CRA's data anywhere near everyone else's data inside a cloud provider
- You'd want an isolated network that has no way to bridge to the public side of things

- For this reason they are typically 4 types of Cloud Computing considered:
 - **Public Cloud**
 - **Private Cloud**
 - **Hybrid Cloud**
 - **On Premises**

Public Cloud

• Basic concepts

Public Cloud

- **Public Cloud** is what we mainly consider Cloud Computing
- This is your standard offering that you will get if you sign up for an AWS or Azure account
- It doesn't mean that your servers are accessible by anyone, it just means that the hardware hosting them are not privately reserved

- For most applications, this is absolutely fine
- The only times you start getting into issues are when you're dealing with Medical Data, Financial Data, Military Intelligence, or Classified Data
- I don't think anyone here is working with this kind of data currently so public cloud is absolutely fine

Private Cloud

• Infrastructure as code

Private Cloud

- **Private Cloud** is Cloud Computing resources used exclusively by one business or organization
- The networking equipment is physically separated from the Public Cloud portion of the datacenter
- Much much more costly than public clouds since you're going to have to reserve the specific hardware you think you may need

- This also means that if you need to rapidly expand your computing needs there is a lead time for the **Cloud Provider** to provision the hardware and deploy it to your Private Cloud
- That means the rapid scaling can't be done
- But since it's all isolated you can meet regulatory compliance requirements

Hybrid Cloud

Hybrid Cloud

- **Hybrid Cloud** is a mixture of your local infrastructure or a private cloud and the public cloud
- This means that you have your most valuable assets stored separately off the public cloud and you treat your public cloud instances as disposable

- A lot of companies use this approach when it comes to one off operations
- For example; movie studios will design all the effects locally inside the studio but when it comes down to doing the final rendering of the scene they will spin up a few hundred heavy GPU servers on a public cloud and render out the frames
- It saves the studio from having to invest in all that hardware for non-common tasks

On Premises

Outcomes

On Premises

- **On Premises** is a much more unique and rare type of cloud computer
- Essentially you have a contract with a cloud provider to have their technicians come onsite and run the equipment for you
- The local IT department gets some of the benefits of all the R&D that the cloud services but at the server cost of having them do it at your location

- I've only ever heard of On Prem happening for some Fortune 100 Companies or Governments
- The costs of On Prem must outweigh the potential impact of a data exposure due to misconfigured virtualization platforms

IaaS

IaaS

- When we talk about hosting our own Server Instances in the Public Cloud we typically are talking about IaaS
- **Infrastructure As A Service** is the term for online services that provide a high-level API to manage low-level details of underlying network infrastructure
- Through most Cloud Provider APIs we can provision server instances, firewall management, load balancing, DNS, and much more

- Before IaaS when you needed to add more computing power to an application or to launch a new server you'd have to:
 - Call up your Server Vendor
 - Give specifics to them about the server
 - Expend the capital and order it
 - Wait a period of time for the manufacture to configure and ship
 - Configure the hardware locally with your requirements
 - Schedule time to move it to the Data Center
 - Have technicians install it

- Now all I have to do is click a button to add a new instance, give it a few details and within 5 minutes I have a new server instance running
- This allows you to do fun things like scale up or scale down an application to meet customer demands
- During peak periods for Netflix, they spin up hundreds if not thousands of application servers to handle demand
- During off hours they reduce the amount of idle servers by destroying servers until they meet a minimum threshold

PaaS

Platform as a Service

PaaS

- **Platform as a Service (PaaS)** is a category of cloud computing services that provides a platform allowing customers to develop, run, and manage applications without the complexity of building and maintaining the infrastructure typically associated with developing and launching an app
- PaaS providers will sometimes have libraries you can include in your project to allow a seamless interface when you deploy

- Like many things, the trade off with using this level of automation is cost
- The more things a provider does for you, the more they charge you for these services
- Since the PaaS Service is just an abstraction, in the background they're still typically using a Public Cloud solution for the actual hardware
- A few examples of this are: **Heroku, Engine Yard, Amazon Web Services (AWS)**

SaaS

Software as a Service

SaaS

- **Software as a Service (SaaS)** is a software licensing and delivery model in which software is licensed on a subscription basis and is centrally hosted
- This is typically End User software, which you fine folks hope to be making
- The business model is generally a small monthly payment for a license to use the software
- Most companies strive for this since, at scale, the revenue is unbelievably high

Some Public Clouds

AMAZON WEB SERVICES

AMAZON WEB SERVICES

- **Amazon Web Services** or colloquially known as **AWS** is a SaaS, Paas, IaaS Cloud Computing Provider owned and operated by Amazon in Amazon Operated Data Centres across the world
- As of 2022, AWS is the current market leader in the Public Cloud sector of the market
- We've talked at length the origins of AWS in Lecture 1- Introduction To WEB401
- The estimated Revenue in 2020 for AWS was \$46 Billion USD

MICROSOFT AZURE

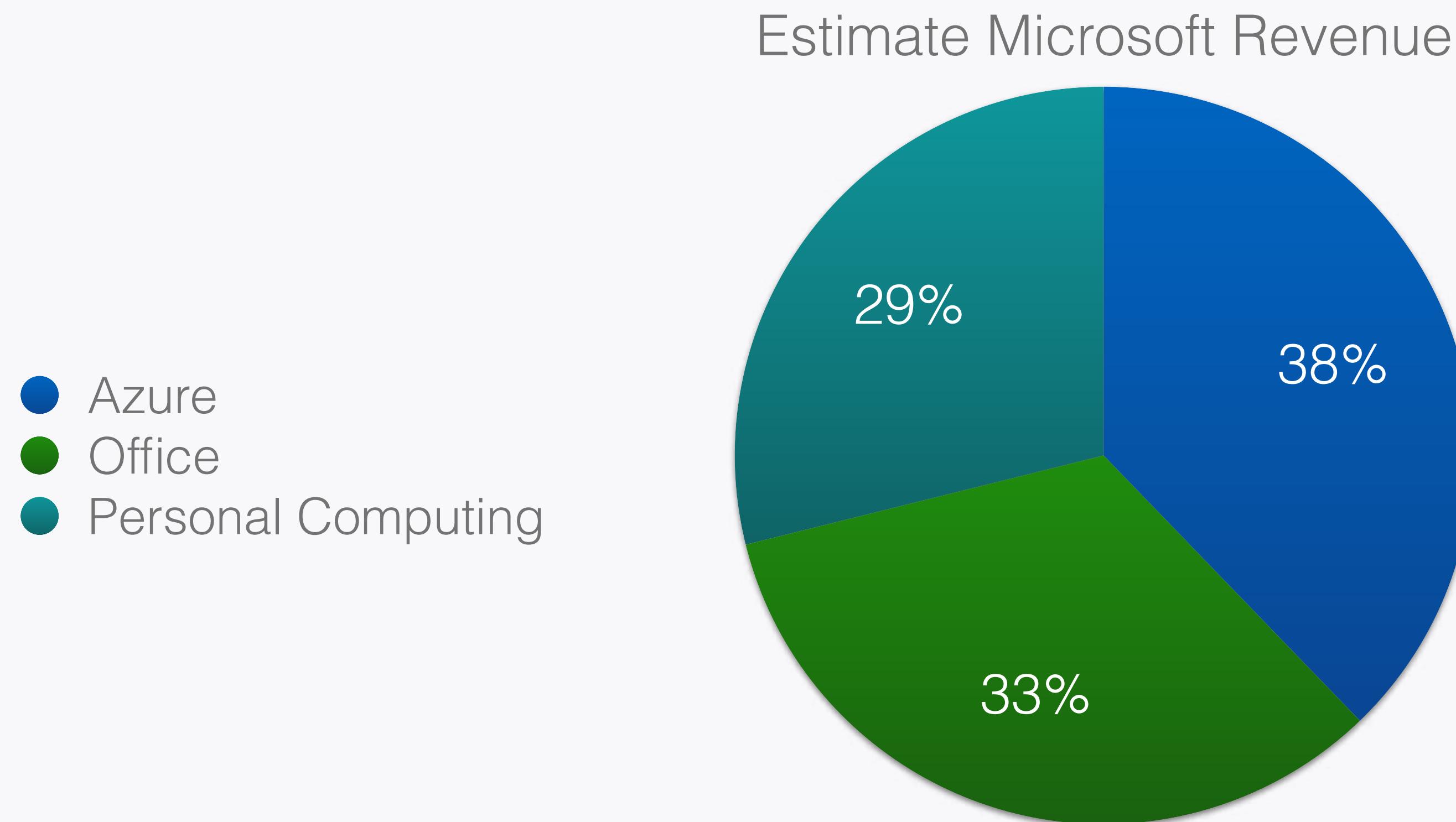
MICROSOFT AZURE

- **Microsoft Azure** is a SaaS, Paas, IaaS Cloud Computing Provider owned and operated by Microsoft in Microsoft-Managed Data Centers
- Originally announced in October 2008 and formally released in February 2010 as Windows Azure before being renamed to Microsoft Azure on March 25th, 2014
- Originally ran onto of Windows Server 2008 with a custom Hyper-V Hypervisor that only allowed for Windows Guest Machines

- During Steve Ballmer's tenure as Microsoft's CEO he was adamantly against the use of Linux or any **Free Libre Open Source Software (FLOSS)**
- Since Satya Nadella became Microsoft's CEO on February 4th, 2014, he has revitalized Microsoft's approach for the better
- Almost begrudgingly, support for non Windows based guest instances began roughly in June 2012

- Since 2012 Microsoft started implementing more and more support and adoption of non Windows operating systems into its cloud architecture
- In 2017 Microsoft reported that 40% of Azure virtual machines were Linux
- In 2018 it was reported that over 50% of Azure virtual machines were Linux
- As of June, 2019 the majority of infrastructure on Azure is now running on a Linux Operating System

- One last thing I wanted to mention was how much money Azure makes per year
- It's estimated that Azure made estimated \$17 Billion in 2021



GOOGLE CLOUD PLATFORM

GOOGLE CLOUD PLATFORM

- **Google Cloud Platform** is a SaaS, Paas, IaaS Cloud Computing Provider owned and operated by Google in Google-Managed Data Centers using the same hardware to power their own services
- Google announced App Engine (Paas) in April 2008 and became generally available in November 2011
- Since then they've added more features to compete with other cloud Providers

- In 2012 Google Announced **Google Compute Engine** which was an IaaS Product that allowed you to host virtual machines on demand
- Google is known to acquire companies that have complimentary software products and fold those products into the Google Cloud Platform
- They're the smallest of the “Big 3” Cloud Providers
- It's estimated in 2020 Google made \$13 Billion USD

LINODE

LINODE

- **Linode** is an American privately-owned Cloud Hosting company based in Philadelphia, Pennsylvania
- Linode mainly has IaaS products for offer, they don't have the same suite as AWS or Azure
- Linode is an inexpensive option for hosting and managing your own Virtual Machines
- Estimated revenue is around \$100 Million USD

DIGITALOCEAN

DIGITALOCEAN

- DigitalOcean is an American owned Cloud Infrastructure Provider with headquarters in New York City and includes DigitalOcean operated Data Centers worldwide
- DigitalOcean mainly offers Virtual Private Servers (VPS) that they call “droplets” which are one-click deployments
- In 2017 they started to offer load balancing solutions to their customers

- DigitalOcean's business strategy is to focus on less options with simplicity in mind
- Determining the estimated monthly bill for AWS, Azure, or GCP can be a nightmare and DigitalOcean tries to streamline it to something easy to digest
- In 2020 their estimated revenue was \$320 Million USD

which is the best?

AUSTRALIAN SURVEY

Which is the best?

- This is a hard question because of the caveat of what it depends on you doing
- AWS is currently the market leader with over 227 different Cloud Products in 26 data centres across the globe
- Azure is typically second but specializes more in Windows deployments
- Google Cloud Platform has great offerings if you're willing to use their technology, for example their great support for TensorFlow AI

- If you just want a simple, easy to understand, and cheap VPS then Linode or DigitalOcean will probably be better
- If you want to forego management of the underlying infrastructure and don't mind paying more and potentially being locked in to a Vendor then pick a PaaS provider
- At the end of the day you need to sit down and do a cost analysis on what platform will best suit your needs and what your budget is

In Summary...

In Summary

- You should have at least a cursory knowledge of Server Racks and what a **U** is
- 4 main types of clouds: public, private, hybrid, on prem
- As of 2022, AWS Is currently the king of Cloud Providers
- PaaS removes the stress of infrastructure management with added costs

Questions?

Lecture 3 - Identity Management

Authentication And Authorization

Authentication And Authorization

- Most people think Authentication and Authorization are one and the same so we should go over some quick definitions
- **Authentication** is the function of granting access into a system via a shared secret
 - Like a password
- **Authorization** is the function of specifying access rights / privileges in a computer system

Linux Local Login

Linux Local Login

- Ever since we've transitioned from InitV to SystemD on linux we've been using LoginD service to authenticate users
- Basically it's what does the checking of your username and password when you attempt to login
- For our purposes today we aren't going to go too deep into login systems but know that it's very robust

- Once you've logged in you are now in the Bash prompt
 - Note: most tutorials will have a \$ before a command which is suppose to inform you that it's for a non root bash shell
 - This is a fully programmable interface in which we will be doing scripting later this semester
 - Through scripting of Bash we can automate most tasks and through Crontab we can schedule those automations to take place at a specific time

Linux User Accounts

Linux User Accounts

- In most Linux systems there is a special reserved account that has absolute power over the system
- That account is called **root**
- With Root you can do anything, including accidentally deleting the whole server while running
- Besides root, all other accounts do not have the privilege to do most administrations tasks

- This is a great security feature unlike other operating systems that grant all users the highest permissions by default
- When you want to execute a command you need to preface the command with **sudo**
- When you run a command with **sudo** it will execute the command as **Root** temporarily
- That being said you shouldn't run most commands with **sudo**, instead you should attempt least privilege first and work your way up

Sudoers and Sudo

Sudoers and Sudo

- As mentioned before **sudo** will execute a command as if it was Root doing it
- We only need to do this for commands that effect the system as a whole and not generally a single user
- For example:
 - **\$ sudo apt-get install apache2**
 - **\$ sudo adduser bob**
 - **\$ sudo deluser bob**

- Now the only users allowed to issue **sudo** commands are users that in the **Sudoers file**
 - Sudoers file can be found **/etc/sudoers**
 - In ubuntu 20.04 you can add a user to Sudoers using the following command:
 - **\$ sudo usermod -a -G sudo <user>**
 - Replace the **<user>** with the correct username of the user you wish to add

How we store passwords?

Storing A Password

- Should they be stored as plain text in a database?
- Should they be stored as scrambled data in a database?
- Should you be able to decrypt them?
- Should you store passwords at all?

- What happens when a database gets stolen?
- Your users end up here if you've failed to properly protect them: <https://haveibeenpwned.com>
- Canada passed a law requiring companies to disclose when you have been breached (<http://www.gazette.gc.ca/rp-pr/p1/2017/2017-09-02/html/reg1-eng.html>)

- We store passwords on Linux salted and encrypted in the **/etc/shadow** using **crypt**
- Crypt is a one way hashing algorithm which takes the password + the salt and hashes
- The output of that has it taken and hashed again, repeated 5, 000 times
- This requires any attacker to do the same process with the guessed password + salt 5, 000 times

Cryptography

What is Cryptography?

- The ability to write what looks like a jumbled message but to the right person reveals a hidden meaning
- This could be done using pen and paper, symbols that hold meaning to a select few, or even electronically

Where was it developed?

- Since the beginning of our civilization we've been hiding messages from other people
- Everything from, "Oh shouldn't tell wife that I ate before dinner" to "Messages delivered from the general to the troops about strategies"
- The earliest known use of cryptography is found in non-standard hieroglyphs in the old kingdom of Egypt circa 1900 BC

Why Cryptography

- The use of cryptography stems from the need to hide something in plain sight
- If messages intercepted during a time of war were decrypted then the enemy would have an advantage
- Nowadays we encrypt messages to and from servers in order to hide the actions
 - The best example is online banking.
 - If an attacker could see your bank account information then he would have the ability to steal from you

Classic Cryptography

Classic Cryptography

- There are two main types of classic cryptography
 - **Substitution ciphers**
 - Systematically replace letters or groups of letters with other letters or groups of letters
 - **Transposition ciphers**
 - Rearranges the order of letters in a message

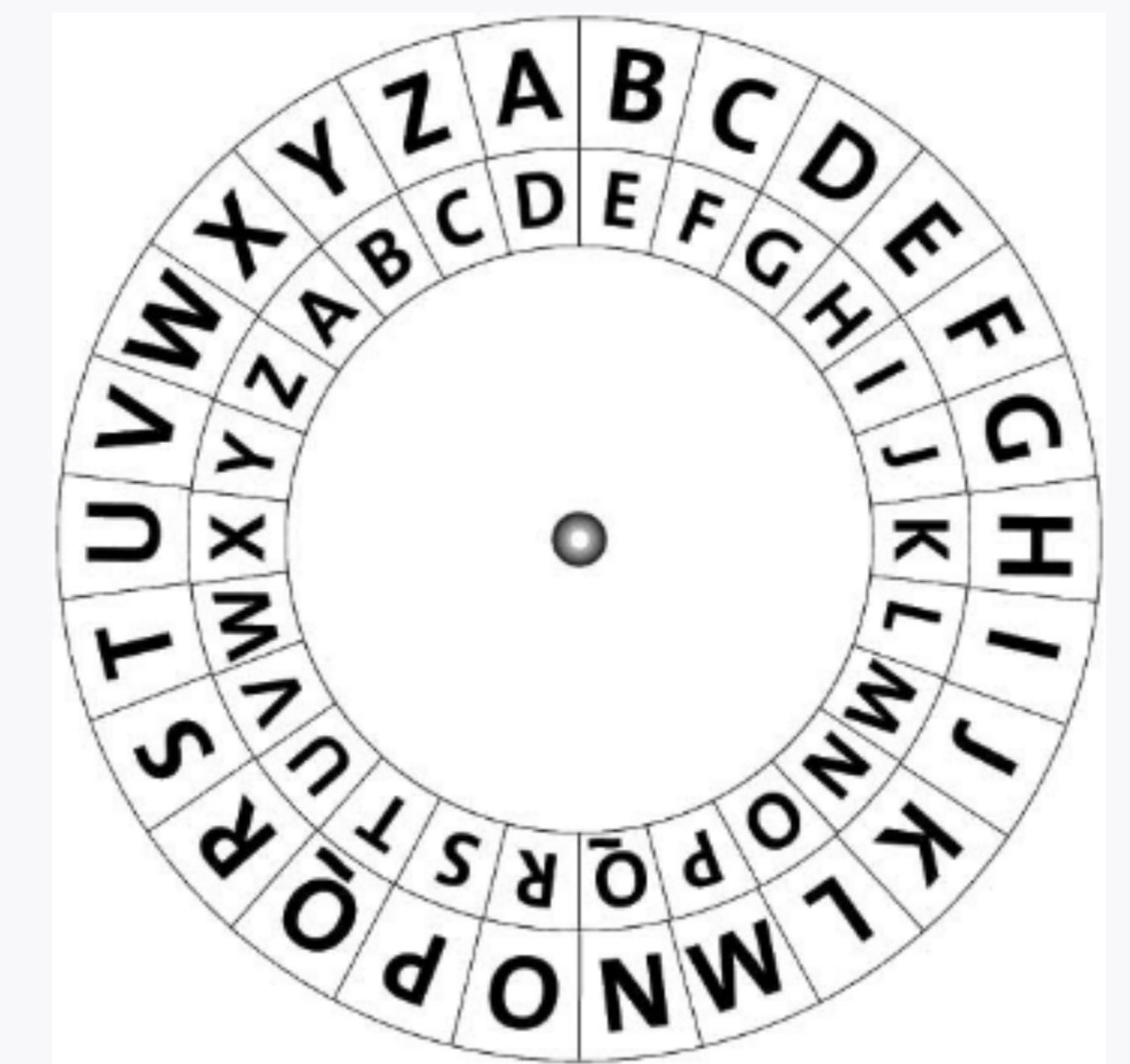
Historical Context

- Let's use Julius Caesar as our prime example of an early substitution cipher
- Julius Caesar developed his own cipher which was dubbed the Caesar cipher
- This cipher was where each letter in plain text was replaced by a letter some fixed number of positions further down the alphabet

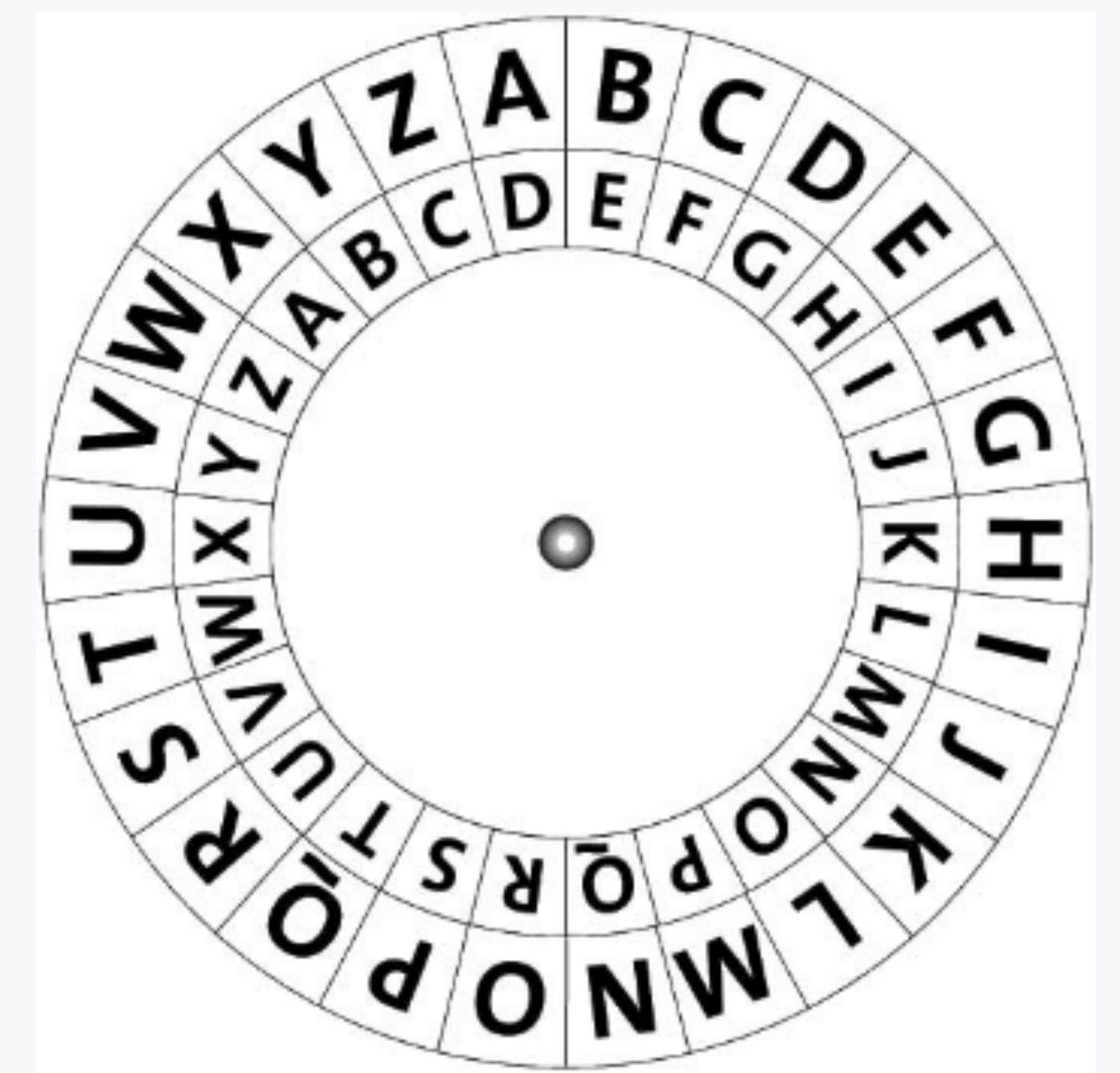
Caesar cipher

- Plain text is a message before it's encrypted / enciphered
- In this case it was a shift of 3 letters
 - Plain: ABC|DEF|GHI|JKL|MNO|PQR|STU|VWX|YZ
 - Cipher: DEF|GHI|JKL|MNO|PQR|STU|VWX|YZA|BC
- An example of this is "Khoor Zruog"
 - What does this spell? (<https://cryptii.com/pipes/caesar-cipher>)

- “ wkh txlfn eurzq ira mxpshg ryhu wkh odcb grj ”
 - *The Quick Brown Fox Jumped Over The Lazy Dog*
- “ Qr, L dp brxu idwkhu ”
 - *No, I am your father*
- “ lw'v gdqjhurxv wr jr dorqh! wdnh wklv ”
 - *It's dangerous to go alone! Take this*



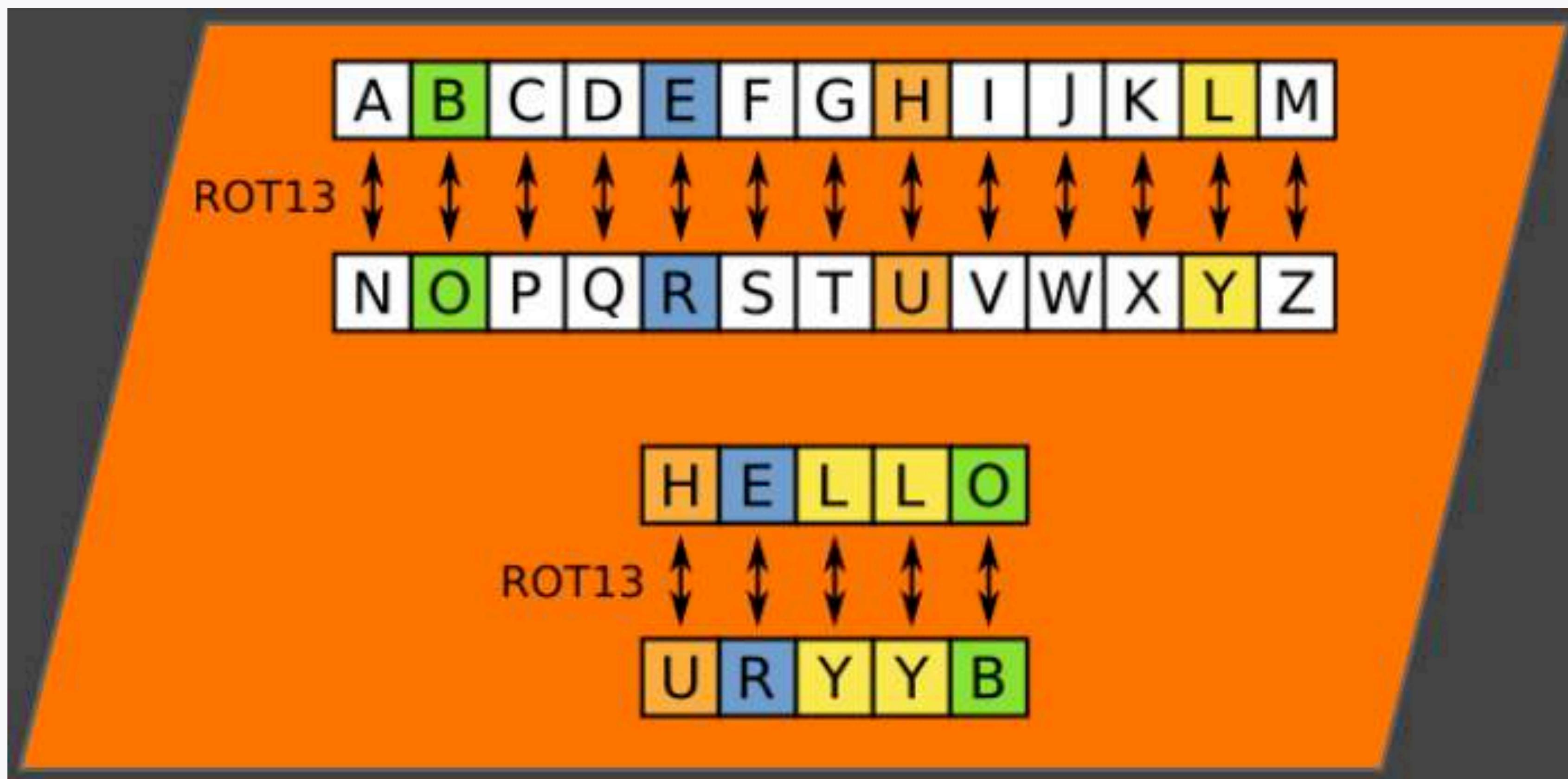
- “ Exw rxu sulqfhvv lv lq dqrwkhu fdvwoh! ”
 - *But our princess is in another castle!*
- “ Khoor, pb qdph lv Lqljr Prqwrbd, brx nloohg pb idwkhu, suhsduh wr glh. ”
 - *Hello, my name is Inigo Montoya, you killed my father, prepare to die.*



ROT13

- ROT13 is a cipher that was used in the early days of the internet
- It is based off of the Caesar Cipher
- ROT13 stands for rotate by 13 places
 - What does this mean?

- Solve this puzzle "gur pnxr vf n yvr"



Transposition Ciphers

- The example I give you is the Rail Fence Cipher
- In this cipher you read diagonally as if the letters were on a fence
- What does the following say?

W . . . E . . . C . . . R . . . L . . . T . . . E
· E . R . D . S . O . E . E . F . E . A . O . C .
· . A . . . I . . . V . . . D . . . E . . . N . .

Why are they no longer used?

- What would you see as a possible problem for these ciphers?
- What do you think is wrong with the Substitution cipher?
- The transposition cipher?

Breaking Simple Ciphers

- If you think about it, there are a few letters that are more commonly used in the English language than others
 - What do you think these letters are?
 - Vowels?
 - Pluralizations?

- Would people write in sentence structure or abbreviations?
- Based on the length of the censored text you can discern meaning just by breaking the commonly used words and inferring context
 - the ***** to life, the *****, and *****
 - the answer to life, the universe, and everything

- If you look at the ciphered text you can see repeats of the same ciphered letters
- This is one way we broke the encryption used by the Germans in World War 2
 - Almost all German messages ended with the same phrase
- These methods can be applied to transposition ciphers as well

Letter	Relative frequency in the English language
a	8.167%
b	1.492%
c	2.782%
d	4.253%
e	12.702%
f	2.228%
g	2.015%
h	6.094%
i	6.966%
j	0.153%
k	0.772%
l	4.025%
m	2.406%
n	6.749%
o	7.507%
p	1.929%
q	0.095%
r	5.987%
s	6.327%
t	9.056%
u	2.758%
v	0.978%
w	2.360%
x	0.150%
y	1.974%
z	0.074%

Rank	Word	Rank	Word	Rank	Word	Rank	Word	Rank	Word
1	the	21	this	41	so	61	people	81	back
2	be	22	but	42	up	62	into	82	after
3	to	23	his	43	out	63	year	83	use
4	of	24	by	44	if	64	your	84	two
5	and	25	from	45	about	65	good	85	how
6	a	26	they	46	who	66	some	86	our
7	in	27	we	47	get	67	could	87	work
8	that	28	say	48	which	68	them	88	first
9	have	29	her	49	go	69	see	89	well
10	I	30	she	50	me	70	other	90	way
11	it	31	or	51	when	71	than	91	even
12	for	32	an	52	make	72	then	92	new
13	not	33	will	53	can	73	now	93	want
14	on	34	my	54	like	74	look	94	because
15	with	35	one	55	time	75	only	95	any
16	he	36	all	56	no	76	come	96	these
17	as	37	would	57	just	77	its	97	give
18	you	38	there	58	him	78	over	98	day
19	do	39	their	59	know	79	think	99	most
20	at	40	what	60	take	80	also	100	us

Modern Day Encryption

- Since we saw how we use to do ciphering how do you think we do in the modern era of computers?
- Do you know of any encryption algorithms?
- Some common types of Encryption are:
 - Asymmetric Key Encryption
 - Symmetric Key Encryption
 - Cryptographic Hashing

Cryptographic Hash Function

Cryptographic Hash Function

- A cryptographic hash function is a special class of hash function that has certain properties which make it suitable for use in cryptography.
- It is a mathematical algorithm that maps data of arbitrary size to a bit string of a fixed size (a hash) and is designed to be a one-way function, that is, a function which is infeasible to invert

- The ideal cryptographic hash function has five main properties:
 1. it is deterministic so the same message always results in the same hash
 2. it is quick to compute the hash value for any given message
 3. it is infeasible to generate a message from its hash value except by trying all possible messages
 4. a small change to a message should change the hash value so extensively that the new hash value appears uncorrelated with the old hash value
 5. it is infeasible to find two different messages with the same hash value
- If it does happen it's called a Hash Collision

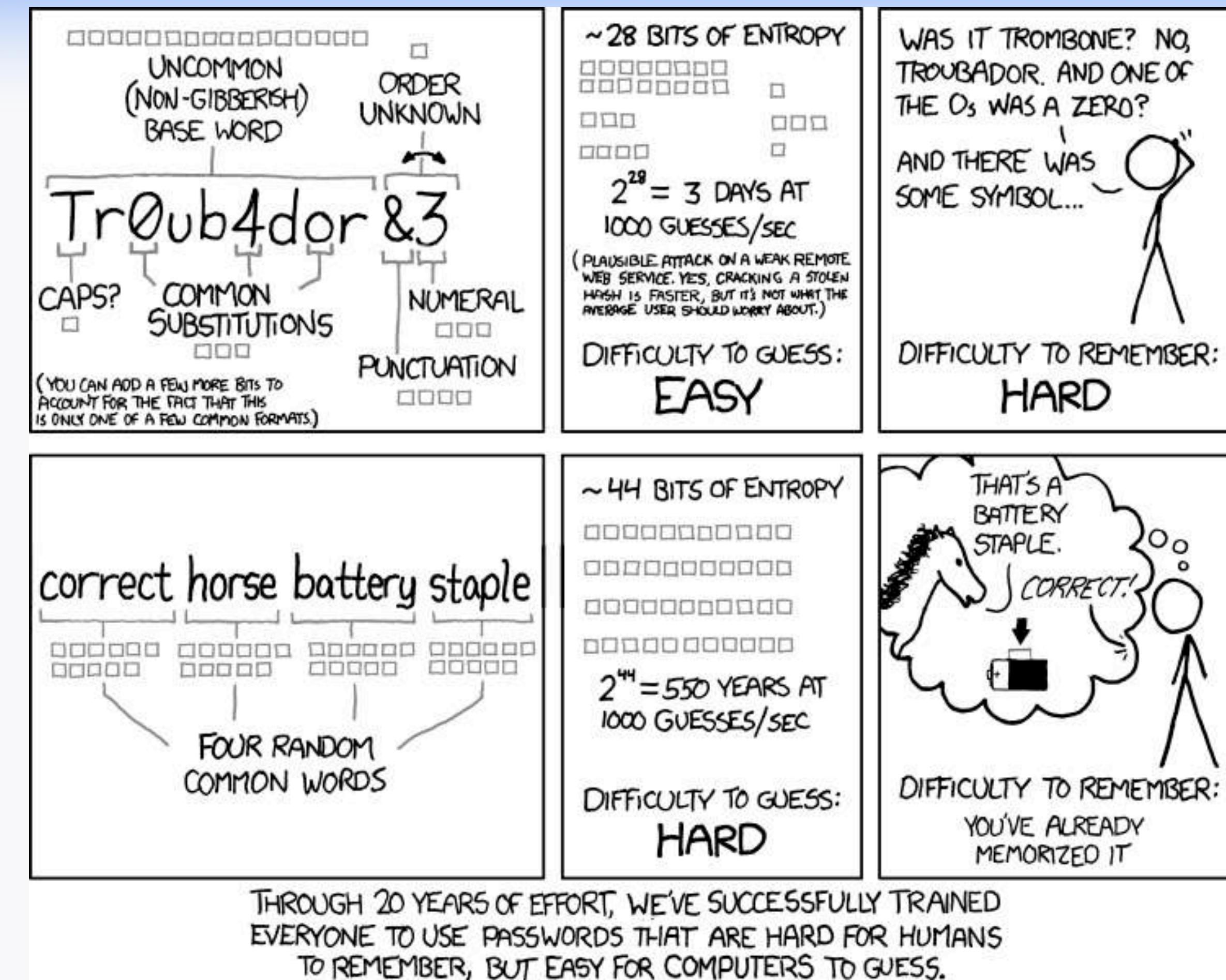
Storing Passwords

- Never ever EVER store passwords as plaintext
- Also hash your passwords using a known hashing algorithm, never your own
- Some known hashing functions are:
 - **MD5**
 - **BCrypt** or **Crypt**
 - **SHA-1**
 - **SHA-256**
 - **SHA-512**

- When a user signs up for your service they submit a password to you
- You take that password and hash it and store the hash into your database as an “encrypted password”
- When a users attempts to log into your application with their email, do a lookup of their email and hash the password they’ve provided and attempt to match the provided hashed password to the known good one
- If they don’t match, fail to log them in
- If they do match, welcome back Mr Smith

Password Length

- Despite what a lot of people think, length is more important than complexity
- The key space entropy, length of password with possible characters in, will drastically change how easy it is to break your password
- A great older tool to give some kind of example is: <https://www.grc.com/haystack.htm>



Rainbow Tables

- One issue is if you use a non salted / peppered hash you are subjected to a Rainbow Table attack which will break every password in seconds
- A rainbow table is a precomputed table for reversing cryptographic hash functions, usually for cracking password hashes
- It is an example of a space–time tradeoff, using less computer processing time and more storage than a brute-force attack which calculates a hash on every attempt, but more processing time and less storage than a simple lookup table with one entry per hash
- Use of a key derivation function that employs a salt makes this attack infeasible

Salt + Pepper + Password

- A salt is random data that is used as an additional input to a one-way function that "hashes" data, a password or passphrase
- Each user is given their own salt at the time of account creation and is the first 29 characters of their encrypted password after the third \$
- A pepper is a secret added to an input such as a password prior to being hashed with a cryptographic hash function.
- A pepper performs a similar role to a salt, but while a salt is stored alongside the hashed output, a pepper is not.

Example of Salt + Peppered Passwords

1	1	liza_schmidt@e...	\$2a\$11\$lvw1zCukO132fcssCzuGpeba/FKga8DXZ6Qur6OT9PKIJ1qBjYsKa
2	2	carlo@example....	\$2a\$11\$ojryBCoHPI6SnS2Jc/ceduBaw80Z/MlyhGrDOK4kYFQ6Dt368aYia
3	3	virgie.ondricka@...	\$2a\$11\$X6Lyf4oMoKbXklNwsEdiEuSH0J0G8e7ddU.6gvsW3FtcY4oK6Hlka
4	4	cara@example.org	\$2a\$11\$Wx3rvHqgSi4WbwAtEu9sYu.dNy94q2dQxzhZ6TxiiDNOsx8FZRp92
5	5	isaac_mclaughli...	\$2a\$11\$Wwachp2Oq9Fr9/0/MgQalesWfWYxboW3KlDhG.zoxUJ8Lc2eC5ZXW
6	6	elvis@example.c...	\$2a\$11\$XgKPzHoY0lwyzQQEyJIIZO7ulRfw1f5xa128LkTQ5xhn7v5LiXbqu
7	7	aliya_ratke@exa...	\$2a\$11\$58gk0nLCvA40.AZsVuFdrOgnYBhn14YusOr8VNxcmsGksJ2BcsZ/u
8	8	garrison@example...	\$2a\$11\$Vlq3k0XLDT2q.7nyfu0xC.TW9aFUoU9NyFhix7hRHb471ctNFpfDO
9	9	corine@example...	\$2a\$11\$76GqZ84ans3QWGKYJqja/.wEn6WosS106MFY0gk0KedQvxM2hohpO
10	10	liana_gorczany...	\$2a\$11\$rVlx9AYhavIVRbGfOgox7O2bCCLJnBKkjP1jUHFVLYhhoh4PHjHqS

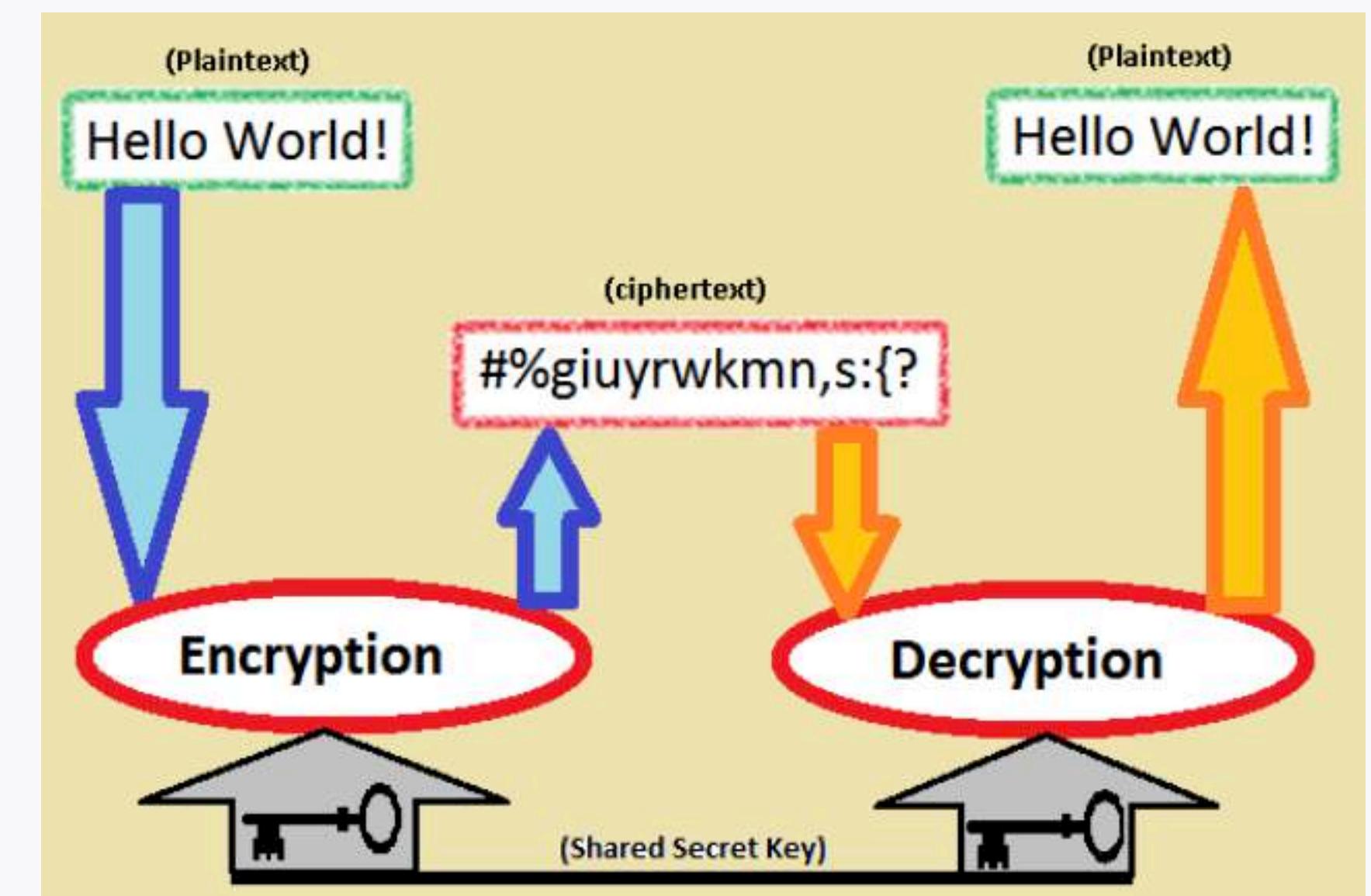
- Note that all these passwords are test1234

Encrypting Transmissions

Symmetric Key Encryption

Symmetric Key

- This is a form of cryptography in which both parties have a shared secret
- Meaning that they each have the same key which allows encryption and decryption



- How do we exchange the key to our recipient?
 - The way in which these keys are exchanged have changed over the years
 - From simple **face to face meetings**
 - **Trusted couriers**
 - Now these are transferred using the **asymmetric-key cryptography**
 - This is called **hybrid-cryptosystems**

- Symmetric-key cryptography is known as computationally cheap compared to asymmetric
- This means it takes the CPU fewer cycles to encrypt and decrypt information when using Symmetric-key Algorithms

Asymmetric Key Encryption

Asymmetric Key Encryption

- A team of scientist, **James H. Ellis**, **Clifford Cocks**, and **Malcolm Williamson** at the Government Communications Headquarters, developed the **asymmetric key algorithms** in secret in 1973
 - They did not disclose this until 1997
 - The more well known paper on asymmetric-key cryptosystem was published in 1976 by **Whitfield Diffie**, a mathematician, and **Martin Hellman**, an electrical engineer.
 - This paper is known as the Diffie-Hellman Key Exchange (more on that later)

- In asymmetric-key cryptography two "keys" are generated that are used to encrypt and decrypt the data
- The **private key** is the key that allows information to be decrypted
- The **private key** should never leave your hands
 - A private key is called a secret, something you want to hide from onlookers

- The **public key** is the key that allows information to be encrypted
 - This key is free to be given away to whomever you want to encrypt data
 - The key can only encrypt; it can never decrypt the data

Deriving The Keys

- They are created by factoring two large unique pseudo-random integers
- So because these are "random" numbers they increase the difficulty by a massive amount
- But since it's all math driven and a random number is never truly random it can be figured out
 - This is just highly unlikely.
- **Asymmetric Key Cryptography** is computationally expensive to perform

SSH Keys

SSH Keys

- Another way to log into a server is via SSH keys
- **Secure Shell (SSH)** is a cryptographic network protocol that allows you to interact with a server remotely as if you were at a mouse and keyboard locally
- It uses complex ephemeral asymmetric and symmetric keys in order to encrypt all data in transit between the client and the server

- For each user you can generate a Public / Private key pair that can be used to log in without a password securely
- The client needs to transfer their public key to the server via some means and saved as a line in the `~/.ssh/authorized_keys` file
- When your client reaches out to the server the server will first ask if there are any SSH keys to transmit, if the client answers yes it will attempt to use those before asking for a password
- Most cloud servers do not use any passwords but SSH keys exclusively, they're much more secure

Types of Authentication

Types of Authentication: IAM

- Since AWS is the biggest Cloud Provider it's important to go over IAM
- **AWS Identity and Access Management (IAM)** is the service which authenticates users in the AWS Console
- This service will define a user, their group, and their permissions allowed in the AWS system
- <https://console.aws.amazon.com/iamv2/home?#/home>

- AWS defines most things as JSON objects in their system
- Not too long ago they also launched a **Wizard**
 - Fun fact it seems that the term **Wizard** is from a magazine MacUser in 1992: '*We'd like you to meet Wizards, step-by-step guides that are designed to walk you through complex tasks.*'
 - It also seems like it's derived from Clarke's three laws: [https://en.wikipedia.org/wiki/Clarke%27s three laws#The laws](https://en.wikipedia.org/wiki/Clarke%27s_three_laws#The_laws)

Types of Authentication: LDAP

- The **Lightweight Directory Access Protocol (LDAP)** is an open, vendor-neutral, industry standard application protocol for accessing and maintaining distributed directory information services over an Internet Protocol (IP) network.
- This allows a centralized server to become the **Source Of Authority** over logins and allows the centralization of Usernames and Passwords

- LDAP can be used for System Level logins or even App level logins
- A great use is that you have your Users accounts in LDAP for both System and App so that way you have one single place to update your password and for all systems and apps to be updated at once
- It also means if you're fired it's a singular place to lock you out of

Types of Authentication: Kerberos

- Kerberos is a computer-network authentication protocol that works on the basis of tickets to allow nodes communicating over a non-secure network to prove their identity to one another in a secure manner
- Basically it works in conjunction with another service, usually LDAP, to provide secure Logins
- Super confusing I know but basically we almost always pair LDAP and Kerberos together and that's all you need to know for now

Types of Authentication: Active Directory

- Microsoft Active Directory (AD) is a directory service for a windows domain
- Basically it's a centralized place to house all Usernames, passwords, and permissions for an entire group of Windows Computers
- Sounds familiar to something else?
- Well it should since it uses the LDAP protocol and Microsoft's fork of Kerberos along with DNS

- Almost all fortune 500 companies use Active Directory to manage their workforce
- Just like LDAP / Kerberos before you have a single point of lock out for people and password management
- You can tie AD into web apps as an authentication source
- In the past few years Azure introduced online cloud based Active Directory

In Summary...

In Summary

- Private / Public SSH keys are an important and safe way to log into your cloud servers
- Most major systems have a unified login of some sort: IAM, LDAP, Kerberos, AD, etc...
- Only users in the Sudoers file can sudo
- Root is the most important account and should never be logged into

Questions?

Lecture 4 - Cloud Virtualization

Hypervisor

Hypervisor

- A hypervisor is the software that runs **Virtual Machines (VMs)**
- It sits distributes resources to each virtual machine as specified by their configuration
- It simulates the virtual hardware that virtual machine is real “**bare metal**”
- It will terminate the virtual machine if it goes rogue

- Since all the hardware is virtualized you can do things with a VM that you couldn't easily do with real hardware
- For example you can take a live snapshot of a VM, including the **current working memory**, and experiment with something
 - This can also be known as **Checkpointing**
 - If that something doesn't work out, you can roll back to that snapshot in seconds and be exactly where you were with all the changes reverted

- This means you can do daily, semi daily, or even hourly snapshots of the virtual machines and in case of a major event you can rollback with minimal data loss
- It also means that you could live copy the VM file to another server, copy the memory while it's running, and flip over to the new server with 0 interruption
- In the clustering world we call that a **Live-Migration**

- This is how Cloud Providers provide near 100% uptime for their customers
- They're constantly live migrating VMs to other machines that they're about to service, that way to the customer they're none the wiser that they're on a different server
- There is a caveat about snapshotting, I've had things go sour when rolling back to a running snapshot, so like many things it's not perfect

- It also allows you to also simulate other CPU Architectures
- For example an **X86 or AMD64 bit Complex Instruction Set Computer (CISC) CPU** can simulate a **Reduced Instruction Set Computer (RISC) CPU** ARM or something even more exotic
- It can also simulate older hardware, to a point, to support legacy operating systems

Types of Hypervisors

Types of Hypervisors

- There are a great many types of Hypervisors that have been developed
- They range from **Free/Libre Open Source Software (FLOSS)** to paid Commercial Software
- They also each have their own strengths and weaknesses

Name	License	Supported Architectures	Hypervisor Type	Notes
Hyper-V	Proprietary	x86-64, (up to 64 physical CPUs), ARMv8	Type 2	Comes with Windows Server and some installations of windows, Azure uses heavily
QEMU	GPL / LGPL	x86, x86-64, Alpha, ARM, CRIS, LM32, M68k, MicroBlaze, MIPS, OpenRisc32, PowerPC, S/390, SH4, SPARC 32/64, Unicore32, Xtensa	Type 2	Supports a vast amount of configurations and guest operating systems
Virtual Box	GPL version 2; full version with extra enterprise features is proprietary	x86, x86-64 (with Intel VT-x or AMD-V, and VirtualBox 2 or later)	Type 2	Free open-ish sourced virtualization for your desktop computer
Proxmox VE	AGPLv3	x86, x86-64	Type 1	Debian based bare metal hypervisor, FLOSS so it's a great starting point for messing around
VMware ESX Server	Proprietary	x86, x86-64	Type 1	Only bare metal server configuration, very powerful, can be very expensive
XEN	GNU GPLv2 +	Same as host So X86 can only do X86	Type 2	Used heavily by AWS

```
graph TD; HW1[HARDWARE] --> HV1[HYPER VISOR]; HV1 --> OSes1[OS OS OS OS OS]; HW2[HARDWARE] --> HV2[HYPER VISOR]; HV2 --> OSes2[OS OS OS]
```

HYPER VISOR

TYPE 1

*native
(bare metal)*

TYPE 2

hosted

Type 1 - Bare Metal Hypervisor

Type 1 - Bare Metal Hypervisor

- A Type 1 Hypervisor installs directly onto the **Server** without going through an operating system
- Since it's not relying on Windows to host it, it's very light weight and fault tolerant
- Allows for joining into large clusters for added features such as **Live Migrations** and **Fault Tolerance**

Type 2 - Hosted Hypervisor

Type 2 - Hosted Hypervisor

- A Type 2 Hypervisor is a Hypervisor that runs on top of a Host Operating System
- It runs just like any other application on the guest
- Allows for point and click configuration
- Gives the benefit of testing out real server operating systems and configurations from your desktop without the risk of damaging hardware or the expense of it

Emulation Vs Virtualization

Emulation Vs Virtualization

- Before we continue let's talk quickly about the difference between virtualization and emulation
- **Emulation** is where you virtualize a very specific set of hardware completely
- **Virtualization** is where you virtualize a generic set of hardware completely

- Emulation can perform better since it's targeting specific hardware and can be optimized better but is much more costly to develop and narrow in function
- Emulation also has to translate the binaries compiled for that specific hardware and convert them into something that the host can actually use
- And to muddy the waters even further you can talk about **WINE** (**Wine is not an emulator**) which just works at an ABI level to capture and translate API calls for the Host operating system

Cloud Hypervisor

Cloud Hypervisor

- Cloud Hypervisors are proprietary hypervisors provided by organizations such as: Amazon, Microsoft, Google, Digital Ocean
- They use other virtualization technologies or modified versions of other hypervisors (**XEN** by **AWS** for example)
- They typically start from an FLOSS project and then modify them and add their proprietary “*special sauce*” on top
- These Hypervisors are controlled by: Web Portal, Command Line, APIs

Alternative Resource Sharing

Alternative Resource Sharing

- There are other ways you can share the host operating system resources with applications
- Other ways to bundle them into smaller containers and ship those containers off to a Cloud Computer host...

Containerization





Containerization

- The idea behind the containerization movement is to put software into standard size containers that can run semi-independent of host operating system
- It comes from the shipping world where everything must fit into a standard container OR you will pay out the nose to ship it
- When you eliminate the variance of the environment of your application you can, in theory, deploy to anything rapidly and reliably

- There's a whole host of technologies to do this, each with their own Pros and Cons
- Some require special kernel patches, some require special cut down kernels
- Some container technologies allow you to move the running application BETWEEN servers with 0 downtime. This is called a live-migration
- This idea has been around since the late 70's and used to be called Jails

Mechanism	Operating system	License	Actively developed since or between	Features									
				File system	Copy on	Disk quotas	I/O rate limiting	Memory limits	CPU quotas	Network isolation	Nested virtualization	Partition checkpointing	Root privilege
chroot	Most UNIX-like operating systems	Varies by operating	1982	Partial	No	No	No	No	No	No	Yes	No	No
Docker	Linux, FreeBSD, Windows x64, macOS	Apache License 2.0	2013	Yes	Yes	Not directly	Yes (since 1.10)	Yes	Yes	Yes	Yes	Only in Experimental Mode with CRIU [1]	Yes (since 1.10)
Linux-VServer	Linux, Windows Server 2016	GNU GPLv2	2001	Yes	Yes	Yes	Yes	Yes	Yes	Partial	?	No	Partial
lmbtfd	Linux	Apache License 2.0	2013	Yes	Yes	Yes	Yes	Yes	Yes	Partial	?	No	Partial
LXC	Linux	GNU GPLv2	2008	Yes	Yes	Partial	Partial	Yes	Yes	Yes	Yes	Yes	Yes
Singularity	Linux	BSD Licence	2015	Yes	Yes	Yes	No	No	No	No	No	No	Yes
OpenVZ	Linux	GNU GPLv2	2005	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Partial	Yes	Yes
Virtuozzo	Linux, Windows	Trialware	2000	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Partial	Yes	Yes
Solaris Containers	illumos (OpenSolaris), Solaris	CDDL, Proprietary	2004	Yes	Yes (ZFS)	Yes	Partial	Yes	Yes	Yes	Partial	Partial	Yes
FreeBSD jail	FreeBSD, DragonFly BSD	BSD License	2000	Yes	Yes (ZFS)	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Partial
vkernel	DragonFly BSD	BSD Licence	2006	Yes	Yes	N/A	?	Yes	Yes	Yes	?	?	Yes
sysjail	OpenBSD, NetBSD	BSD License	2006–2009	Yes	No	No	No	No	No	Yes	No	No	?
WPARs	AIX	Commercial proprietary	2007	Yes	No	Yes	Yes	Yes	Yes	Yes	No	Yes	?
iCore Virtual Accounts	Windows XP	Freeware	2008	Yes	No	Yes	No	No	No	No	?	No	?
Sandboxie	Windows	Trialware	2004	Yes	Yes	Partial	No	No	No	Partial	No	No	Yes
systemd-nspawn	Linux	GNU LGPLv2.1+	2010	Yes	Yes	Yes	Yes	Yes	Yes	Yes	?	?	Yes
Turbo	Windows	Freemium	2012	Yes	No	No	No	No	No	Yes	No	No	Yes
RKT	Linux	Apache License 2.0	2014	?	?	?	?	?	?	?	?	?	?

- The general idea is to run an application and all its dependancies in a pseudo virtual machine
- This pseudo virtual machine will share the same Kernel with the Host OS, but its entire filesystem is isolated and same with network connections
- These containers will run as processes under the Host OS so if they go rogue (eat all resources) the OS can still kill them to save itself
- It also means we can run many containers parallel as long as they don't bind the same ports

Chroot Jail

Chroot Jail

- A chroot is an operation that changes the apparent root directory for a currently running process and its children
- A program that is Chrooted cannot modify files outside of its designated directory tree
- First developed in 1979 for Version 7 Unix and became the first “Container” type technology
- This has become the basis of LXC which Docker was developed from

LXC

LXC

- Linux Containers (LXC) is the technology used for running multiple isolated Linux systems on a control host using a single Linux Kernel
- It allows the most computationally expensive part of an Operating System, the Kernel, to be shared with multiple instances of Linux
- LXC specifically works with a “Vanilla Kernel” so no extra Kernel specific patches are required, or specialized kernels with reduced functionality like OpenVZ

Docker

Docker

Docker

- Docker was first released in 2013 by **Docker Inc** and was built around an existing technology, LXC
- They were founded by **Y Combinator** (Startup incubator) in 2010 and “launched” in 2011
- In 2014 they rewrote the container technology and stopped supporting LXC as the base
- The goal of docker is to be able to build consistent environments for applications and to be able to “ship” those containers to production or other developers without the issue of minor variances between environments

- Docker releases super cut down versions of linux-based operating systems for users to build upon called base images
- These are all available in the Docker Hub (<https://hub.docker.com/>)
- One of the most common base images for docker containers is called **Alpine linux**
- Alpine linux starts off at only 5MBs of disk space
- In order to specify the environment you need to have a Dockerfile in each project which describes the requirements and actions needed to get your application running

Dockerfile

```
# This tells Docker which image to use, this is an official ruby image from docker
FROM ruby:2.6

# Update the repo listings in the container
RUN apt-get update
# Install the following packages into the container
RUN apt-get install --assume-yes --no-install-recommends build-essential postgresql-client ca-certificates nodejs

# Specify the environment variable APP with the path
ENV APP /usr/src/app

# This makes the directory for the app
RUN mkdir -p $APP

# This tells the container where we will be working from
WORKDIR $APP

# This copies the gemfile and gemfile.lock to the app folder
COPY Gemfile* $APP/

# This tells it to run bundle install on as many threads as it has available
RUN bundle install --jobs=$(nproc)

# This tells it to copy our code into the app folder
COPY . $APP/

# This tells the container to run your migrations
CMD ["bin/rails", "db:create"]

# This tells the container to run your migrations
CMD ["bin/rails", "db:migrate"]

# This tells the container to start the rails server on port 3000 and bind to wildcard IP
CMD ["bin/rails", "server", "-p", "3000", "-b", "0.0.0.0"]
```

- You can find many pre-built application templates in Dockerhub for all sorts of services such as MySQL, PSQL, Redis, Ruby, PHP, etc...
- When you run your container you need to expose Ports from the container to the outside world just like if you wanted to Port forward on your router at home
- Once the docker is started, you can PAUSE the docker at any time
- STOPPING a docker image will actually destroy that instance and any data that's inside like your SQLITE3 database or uploaded files

```
1 # This builds the docker image based off the Dockerfile in current directory
2 docker image build -t lab3:1.0 .
3
4 # This starts the container called lab3:1.0 detached, name it lab 3, and bind port 3000 to it
5 docker container run --publish 3000:3000 --detach --name lab3 lab3:1.0
6
7 # This shows all currently running docker processes
8 docker ps
9
10 # If the docker conainter is running, this pauses it
11 docker pause lab3
12
13 # If the docker contianer is paused, this runs it
14 docker run lab3
15
16 # This logs you into the container to run commands like you would in terminal
17 docker run -it lab3:1.0 sh
18
19 # In docker stop doesn't mean stop. It means destroy all currently running instances with all data inside
20 docker stop lab3
21
22 # This will delete the container image from your computer
23 docker container rm --force lab3
```

- Now, that's one heck of a drawback if all data is destroyed if you accidentally run the wrong command
- How would YOU solve this issue but still keep your entire app in a docker container consuming no outside resources?
- The real answer is to use a Docker Compose instance and save the database file to your local computer instead of keeping it in docker

- Before we move on to Docker Compose we need to talk about the problems with docker
- For example; if you're not running a Linux based machine then Docker needs to run a special Linux based Virtual Machine in order to work
- This EATS up space on your machine, at least 8GBs but in reality much more (I have 80GBs dedicated to it)
- And since it has to do some special filesystem hackery in Non-Linux machines the Disk IO is much slower

Docker Compose

Docker Compose

- Docker Compose is a tool that allows you to specify many docker containers to be run in a group (stack)
- It allows each container to have its own Dockerfile but gives inter-container communication easily
- For example in your docker-compose.yml file if you specify a database technology you can give it the name of DB and call from the other containers the “DB” service and it will route the traffic to the correct place for you

- You can also expose Ports externally so you can view the application just like you can in any given Docker Container
- Reminder: the left port is what your computer sees and the right port is inside the docker container (54320:5432)
- You can also use other pre-made container solutions so you can, for instance, test if email is working and what it could look like to a client (mailcatcher)
- This also keeps all instances of services at the same version for all people working on the project which solves a lot of headaches

- You can also map the data directory of a PSQL Container's database to a file in your project just so it will survive if you rebuild the containers
 - PSQL example: `./tmp/db:/var/lib/postgresql/`
 - Up to the first colon is where in your project the DB files will reside, after the second colon is where in the container the PSQL data is
 - You can also specify startup scripts to execute so you can have a little more control over what happens

```
version: '3.4'
services:
  # These are backend services that the app depends on
  db:
    image: postgres:9.6.13-alpine
    volumes:
      - ./tmp/db:/var/lib/postgresql/
    ports:
      - "54320:5432"
  redis:
    image: redis:5.0-alpine
    command: redis-server
    logging:
      driver: none
    volumes:
      - './tmp/redis:/data'
  mail:
    image: jeanberu/mailcatcher
    ports:
      - "1025:1025"
      - "1080:1080"
  # These are the components of the app
  app:
    build:
      context: .
      args:
        UID: ${UID:-1001}
    volumes:
      - ./:/usr/src/app
    ports:
      - "3000:3000"
    depends_on:
      - db
    environment:
      - RAILS_ENV=development
    entrypoint: ["bin/docker_start.sh"]
```

```
1  #!/bin/bash
2
3  # Written by: Andrew Raymer
4  # Last Modified By: Andrew Raymer
5  readonly VERSION="1.0.0"
6  # Date Created: Dec 17th, 2019
7  # Date Last Modified: Dec 17th, 2019
8  set +e
9  echo "Attempting to Migrate the DB"
10 bin/rails db:migrate 2>/dev/null
11 RET=$?
12 set -e
13 if [ $RET -gt 0 ]; then
14     echo "Migration failed; creating the database"
15     bin/rails db:create
16     echo "Migrating the database"
17     bin/rails db:migrate
18     bin/rails db:test:prepare
19     echo "Seeding the database"
20     bin/rails db:seed
21 fi
22 echo "Removing the old server PID"
23 rm -f tmp/pids/server.pid
24 echo "Starting the webserver"
25 bin/rails server -p 3000 -b '0.0.0.0'
```

- In my professional opinion Docker Compose is the ideal way to hand over a project to a coworker and know they can get it running in under 5 minutes
- Compare trying to get projects working with each other and how long and horrible that is to having them clone the project, run a command and BAM it's all there working
- This is one of the MAJOR benefits of Docker, let alone the production shipping aspect

Availability Zones

Availability Zones

- Now we've talked about how Cloud Providers can have a many guest VMs to a single host, let's talk about how these VMs can be geographically shared
- An **Availability Zone (AZ)** is a physically separated location within a **Region**
- This location is at least another datacenter that has it's own independent Power, Networking, and Cooling

- Most Cloud Providers allow you, for a fee, to have a **Hot Spare** of your virtual machines in another AZ
- A **Hot Spare** is basically a VM that is kept in relatively lock step with the **Live** VM so it can be turned on quickly to take over
- That way if there is a critical failure of an entire AZ, the Cloud Provider will automatically migrate to the AZ and reroute traffic for you

Region

Region

- **Regions** are a group of Availability Zones that are hosted in entirely different geographical locations
- They do this in case of a major disaster such as flooding, nuclear attack, or major outages
- AWS has the following: https://aws.amazon.com/about-aws/global-infrastructure/regions_az/
- Someone created an interactive Azure map here: <https://build5nines.com/map-azure-regions/>

Resource Credits

Resource Credits

- So we have a VM in multiple AZs across a few regions, what do we have to worry about next?
- **Noisy Neighbours**
- Now an issue you can run into is that one of the Guest VMs on the Hypervisor is going full tilt all the time which is consuming too much resources, they're a noisy neighbour
- Just like the 95th Percentile networking concept we talked about in Lecture 2, Cloud Providers over subscribe resources on Servers

- If everyone used 100% CPU and RAM then every server would immediately buckle
 - Just like your local ISP
 - Instead they expect you to use only a fraction of the total output most of the time
 - The way they balance this is with **Resource Credits**
 - When you're below a certain threshold, you gain credits but when you exceed it you consume them

CPU Credits

CPU Credits

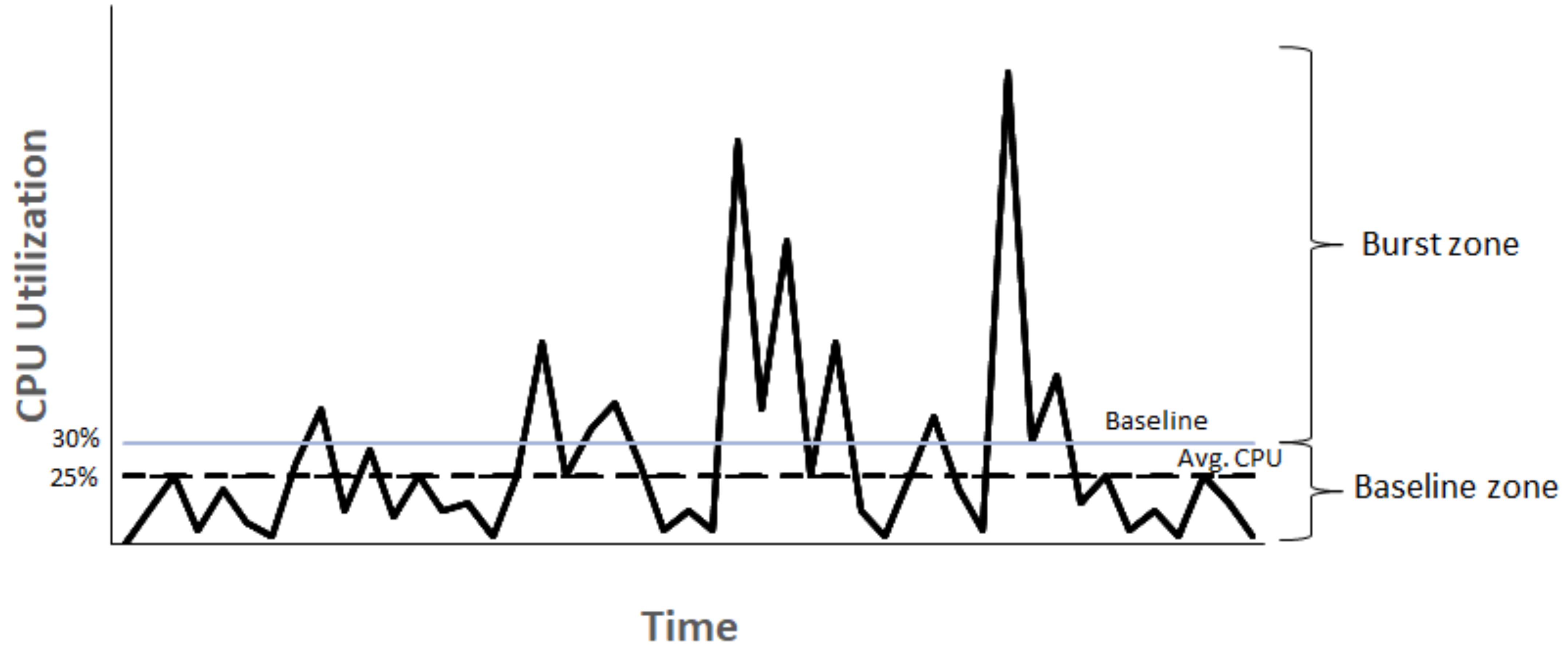
- Some instance types come with “**Bursty**” performance
- As stated before each burstable performance instance continuously earns credits when it stays below the CPU baseline, and continuously spends credits when it bursts above the baseline.

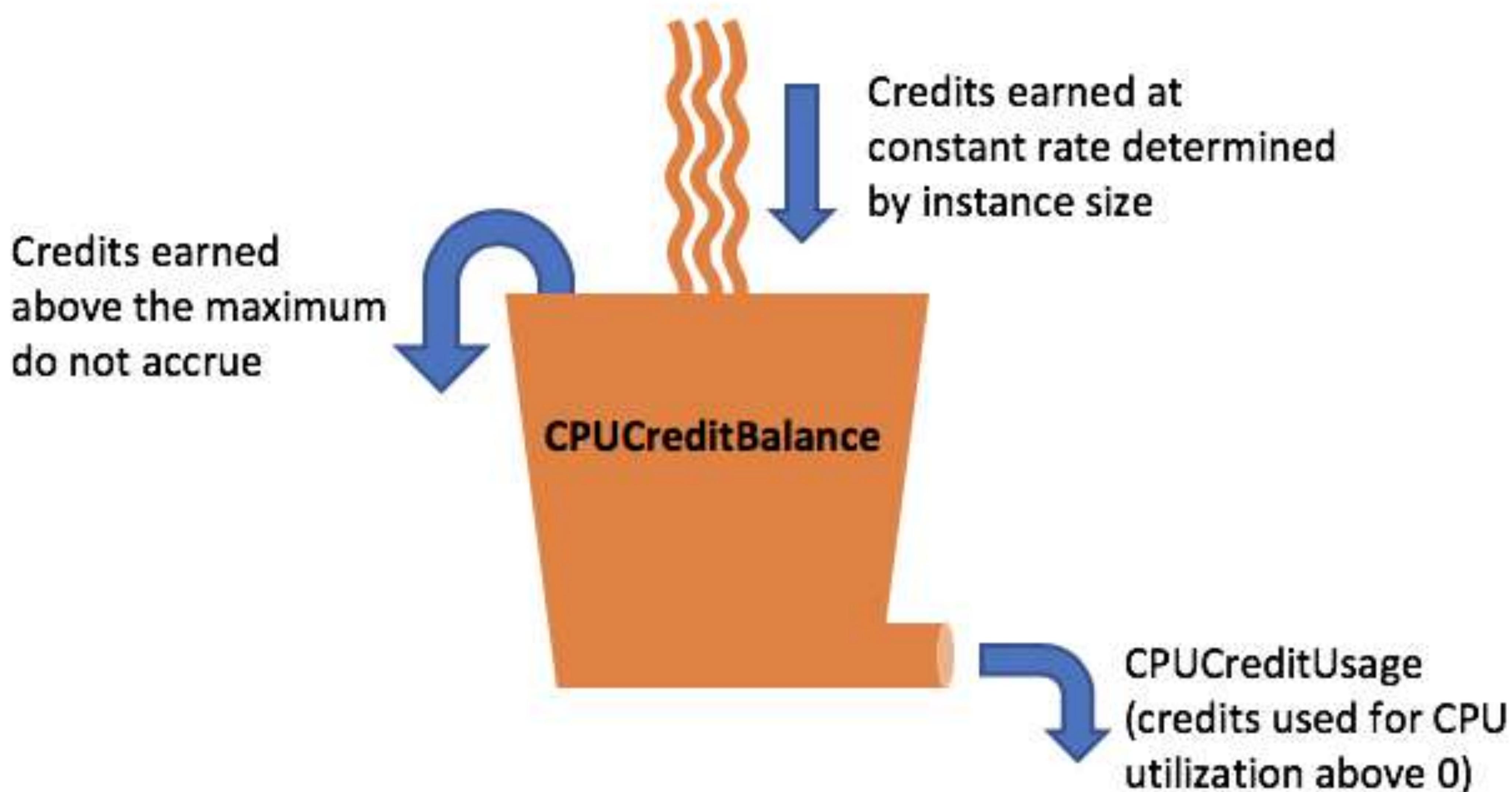
- The amount of credits earned or spent depends on the CPU utilization of the instance:
 - If the CPU utilization is below baseline, then credits earned are greater than credits spent.
 - If the CPU utilization is equal to baseline, then credits earned are equal to credits spent.
 - If the CPU utilization is higher than baseline, then credits spent are higher than credits earned.

- If you have earned more credits than spent then you can accrue credits which can be used at a later point
- Example calculations for CPU Credit expenditure rate:
 - $1 \text{ CPU credit} = 1 \text{ vCPU} * 100\% \text{ utilization} * 1 \text{ minute.}$
 - $1 \text{ CPU credit} = 1 \text{ vCPU} * 50\% \text{ utilization} * 2 \text{ minutes}$
 - $1 \text{ CPU credit} = 2 \text{ vCPU} * 25\% \text{ utilization} * 2 \text{ minutes}$

- For AWS the **Baseline** is calculated as follows:
 - $(\text{number of credits earned}/\text{number of vCPUs})/60 \text{ minutes} = \% \text{ baseline utilization}$
 - For example, a **t3.nano** instance, with **2 vCPUs**, earns 6 credits per hour, resulting in a baseline utilization of 5% , which is calculated as follows:
 - $(6 \text{ credits earned}/2 \text{ vCPUs})/60 \text{ minutes} = 5\%$ *baseline utilization*

Example of t3.large





- When you launch an instance you are normally given a certain amount of credits to use for initial setup
- You can also purchase CPU Credits for an additional fee
- But what happens when you run out of CPU Credits?

Credit Exhaustion

Credit Exhaustion

- When you exhaust your CPU Credits it will cap your performance at the baseline CPU value
- Since the baseline is based on the instance sometimes this isn't that bad but other time it's awful
- For example a t2.micro (1 vCPU, 0.5GBs RAM) has a baseline of 5% which is just knee capping
- A t2.large (2 vCPU, 8GBs RAM) has a baseline of 30% which is still not great but palatable

- The issue lies with when the exhaustion takes place
- Typically the reason why you're running out of CPU Credits is because you're under high load or traffic
- So when you need there performance the most you could run out of credits and cripple your application
- This is one reason why you may consider switching to a more expensive but non burstable instance type

How much time do I have before I run out of CPU credits?

... then you have this number of minutes left,

If your per instance type:

CPU consumption is...	t2.nano	t2.micro	t2.small	t2.medium	t2.large	t2.xlarge	t2.2xlarge
5%	unlimited						
10%	600	unlimited	unlimited	unlimited	unlimited	unlimited	unlimited
25%	150	200	600	600	unlimited	1200	369
50%	66	75	100	100	150	109	90
75%	42	46	54	54	54	57	51
100%	31	33	37	37	37	38	36

*Assumes the instance has just been started and is assigned the following CPU credits at launch: for t2.nano, t2.micro and t2.small: 30; for t2.medium, t2.large: 60, t2.xlarge:120, t2.2xlarge:240



Cloud VM Storage

Cloud VM Storage

- We all have heard of S3 for file storage but this isn't what we're talking about now
- What we're talking about when I mention cloud VM storage is the "**hard drive**" on the VM that the guest operating system relies on
- We call this **Backing Storage**
- As we mentioned before datacenter typically use **Storage Area Network (SANS)** to host the bulk of the backing storage for virtual machines

- On AWS the backing storage is referred to as **Elastic Block Storage (EBS)**
- Azure refers to this as **Azure Virtual Disk**
- This is an additional fee based on the type of Backing Storage you want such as: **Hard Disk Drives (Spinning Rust aka HDDs)**, **General Purpose Solid State Drives (SSDs)**, **Provisioned IOPS Solid State Drives**

- Each tier of Backing Storage is faster but most costly:
 - Provisioned SSDs (up to 500MBs/sec) are the fastest
 - General Purpose SSDs (128MBs/sec) are not as fast
 - HDDs (40-60MBs/sec) are the slowest but super cheap

- This brings into question **IOPS (input/output operations per second)** and what that is
- An IOPS is the unit of measurement for the amount of input/output (write/read) operations per second
- Basically it's how many things you can do and unless you're hosting your own high volume Databases on an instance, just get the default and move on with your life
- Here is a handy chart! [https://aws.amazon.com/ebs/
provisioned-iops/#Product Details](https://aws.amazon.com/ebs/provisioned-iops/#Product Details)

In Summary...

In Summary

- Hypervisor is a program that virtualizes computers
- Containers are small shippable files that host the entire application but not a guest operating system
- CPU Credits are confusing and counter intuitive
- Acronyms you need to know: HDD, SSD, IOPS, EBS, VM, Guest OS, Host OS, Region, AZ

Questions?

Lecture 5 - Linux Management

Kernel

Kernel

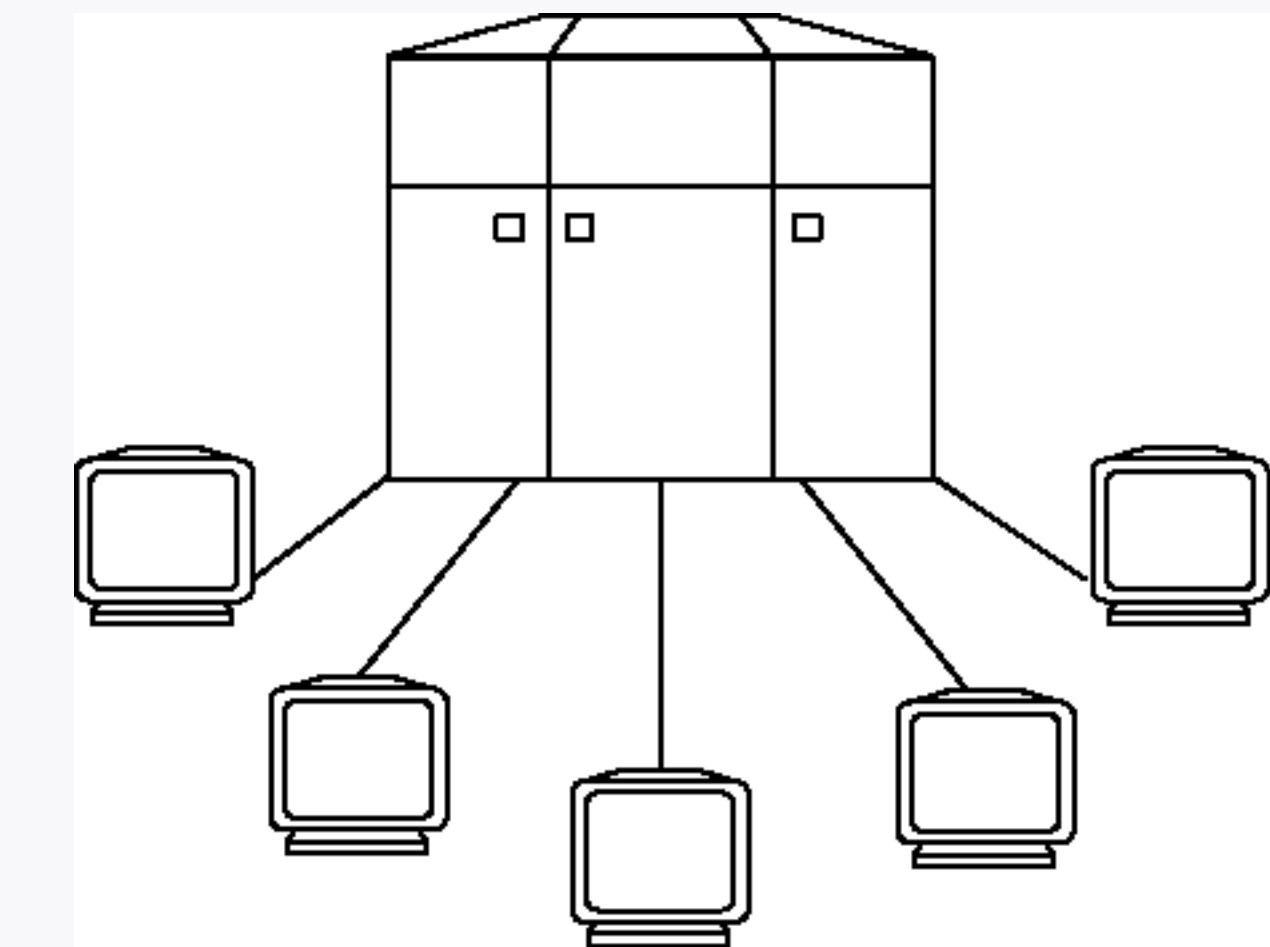
Unix

OS

Unix

- A multitasking, Multi-user, Mainframe Operating System
- Developed in 1969 by AT&T's Bell Labs division
- Originally written in Assembly Language but was rewritten in C to ease portability to other architectures

- The Usage model was users connected via Dumb Terminals to the Server
 - Much like active directory of today
 - This existed before passwords were used / invented
 - People just assumed you were who you said you were



- The two most important philosophies behind Unix are:
 - **Everything is a file**
 - **Each program should be for a single purpose and should perform that purpose well.**
- This means core configurations are in files instead of locked behind a program, like the Windows Registry

- Unix's Source code was given away with a purchased license in the 1970's
- The license cost for a Non-Educational copy of Unix was \$20,000
- In the mid 1980's Unix stopped sharing its source with a purchase of the license

Richard Stallman

Richard Stallman

- **Richard Stallman** was a researcher at the MIT Artificial Intelligence Labs in 1971
- Was considered among the first "hackers"
- Was heavily against the use of Passwords
- He started the GNU project because he believes all software should be free and open to “hack”



GNU

GLIBC

GNU

- GNU stands for "GNU's not Unix"
- Recursive humour is very popular in the open source community
- In order to understand recursion, you must understand recursion
- Richard Stallman released the GNU Manifesto in 1985

- Richard Stallman also started the NPO "Free Software Foundation" (FSF)
- Richard also popularized the term "Copyleft"
- It's the antithesis of "Copyrighting" something
- He also wrote the first version of the "GNU General Public License" (GPL)

- Stallman announced when founding the "FSF" he was going to build a UNIX compatible Operating System
- In order to create the OS he need to create a few "free" applications
- All of these applications were licensed under the GNU GPL
- You've used most, if not all of these applications already without even realizing it

- **Emacs**
 - A command line Text Editor
- **GCC**
 - A C language compiler
- **gdb**
 - A debugger
- **Gmake**
 - A build automator

- With these tools he started his work on the GNU OS kernel
- The Kernel's name is "GNU Hurd"
- The GNU Hurd is a microkernel
- Three types of Kernels
 - Monolithic
 - Micro
 - Hybrid

Linux

www.linux.org

Linux History

- When we think of Linux we are actually thinking of 2 things at the same time.
- Linux is just the Kernel (engine) and the user space is built on the GNU Toolkit
- The Linux kernel was started in October 1991 by **Linus Torvalds** because he didn't want to pay the licensing cost of Unix on a 80386 processor
- The name Linux is a portmanteau of "Linus" and "UNIX"
- It became his Masters Thesis at University of Helsinki.





Linus Benedict Torvalds



Hello everybody out there using minix -

I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones. This has been brewing since april, and is starting to get ready. I'd like any feedback on things people like/dislike in minix, as my OS resembles it somewhat (same physical layout of the file-system (due to practical reasons) among other things).

I've currently ported bash(1.08) and gcc(1.40), and things seem to work. This implies that I'll get something practical within a few months, and I'd like to know what features most people would want. Any suggestions are welcome, but I won't promise I'll implement them :-)

Linus (torv...@kruuna.helsinki.fi)

PS. Yes - it's free of any minix code, and it has a multi-threaded fs. It is NOT protable (uses 386 task switching etc), and it probably never will support anything other than AT-harddisks, as that's all I have :-(.

A Brief Survey

A Brief Survey

- Who here has an Android Phone?
- Anyone who has an Android Phone uses the Linux Kernel
- Android is open source-sorta
 - The google applications are not though

- Who here has an iPhone?
 - iPhone, just like Macs, use the Darwin Kernel
 - The Darwin Kernel is based off of BSD
 - BSD is a UNIX like OS, just like GNU/Linux

- Ever use a Land Line Telephone?
 - A large portion of the POTS network is based off of UNIX
- Have a modern TV? (not even smart)
 - A large majority of new TVs use the Linux Kernel
- Who has been to Facebook or Amazon or Twitter or Google?
 - All use Linux Servers



PRESENTS...

File System

What is a File System?

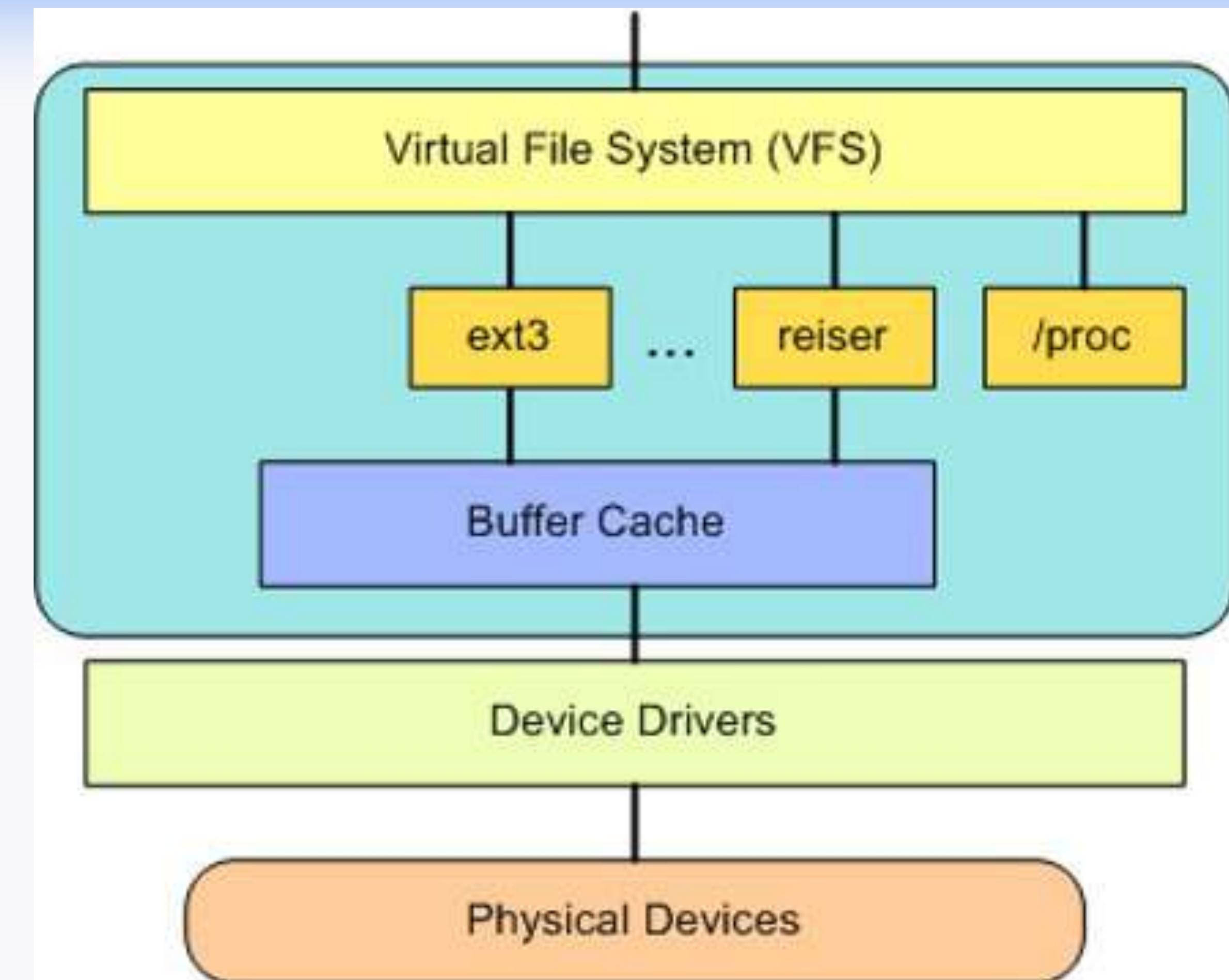
What is a File System?

- A file system is an abstract system to organize data
- The file system has multiple layers stemming from the kernel
- Inside the Kernel there is a Virtual File System (VFS)
- VFS provides a common interface abstraction for file systems

Virtual File System

Virtual File System

- At the top is a common API abstraction of functions such as open, close, read and write
- At the bottom of the VFS are the file system abstractions that define how the upper-layer functions are implemented



- Each filesystem within the VFS are just plugins
- **Fourth Extended Filesystem (ext4)**
 - Older reliable file system
 - Supports up to 1 Exabyte in volume, 16 Terabytes File
- **Global File System (GFS)**
 - Used for server farms
- **B-tree FS (Btrfs)**
 - Very common filesystem used
 - Supports snapshots, pooling, checksums, and multi-device spanning
 - Supports up to 16 Exabytes in volume, 16 Exabytes File

- **LustreFS**
 - used mostly in supercomputing
- **ZFS**
 - used for high storage capacities for data resiliency
 - RAID-Z
 - supports up to 16 Exabytes (16 million terabytes)

- Many many more File systems for Linux
- Each was built for a purpose and has its own disadvantages and advantages
 - Research which would fit your needs then implement it
- Since each of these sub-file systems are plugins you can add and remove them dynamically

Fourth Extended Filesystem (ext4)

Fourth Extended Filesystem (ext4)

- A journaled File System
- built with backwards compatibility of ext3
- Included with the 2.6.18 Linux Kernel (Sept 20th, 2006)
 - marked as stable in the 2.6.28 Linux kernel release (Dec 24th, 2008)
- Supports up to 1 Exabytes of storage volumes (1 million terabytes)
- Supports file sizes of 16 Terabytes

- Journal checksumming
- A journaled filesystem is a file system which keeps tracks of changes in a journal before committing them to the main filesystem
- When a power failure happens or a system crash the data is less likely to have data corruption
- Persistent pre-allocation

- During a write to the medium, ext4 reserves the space beforehand in order to increase the likelihood of the data being contiguous on the medium
- This also guarantees the correct amount of space for said write operation
- Improved timestamps
- ext4 also measures timestamps on files to the nanosecond
- This is done for mission-critical applications not to mistakenly overwrite objects

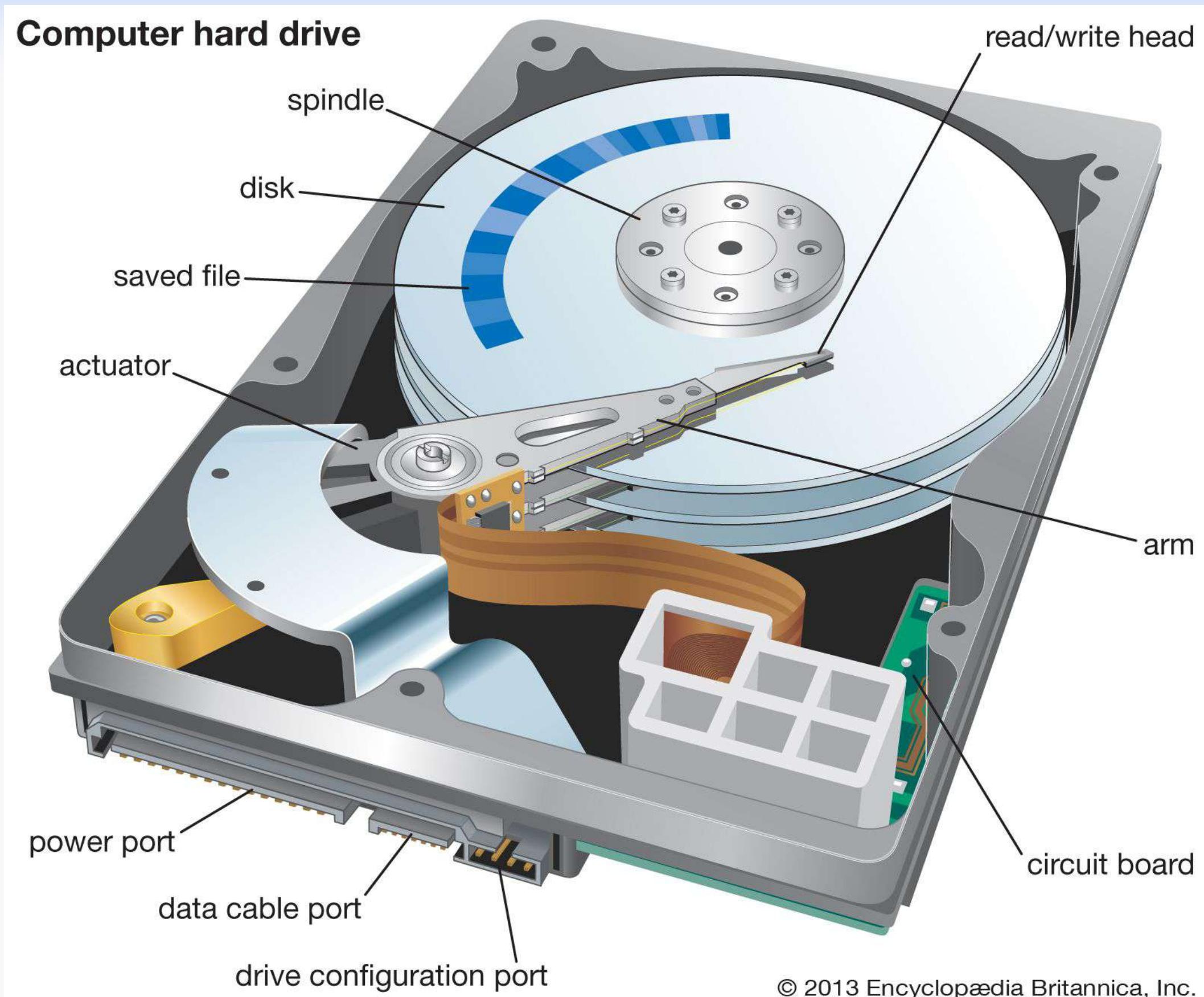
Lower Level Concepts

Buffer Cache

- This element **keeps track of read and write requests** from the individual file system implementations and the physical devices (through the device drivers).
- For efficiency, Linux maintains a cache of the requests to **avoid having to go back out to the physical device for all requests**
- Instead, the most-recently used buffers (pages) are cached here and can be **quickly provided back to the individual file systems**

Hard Disk Drive

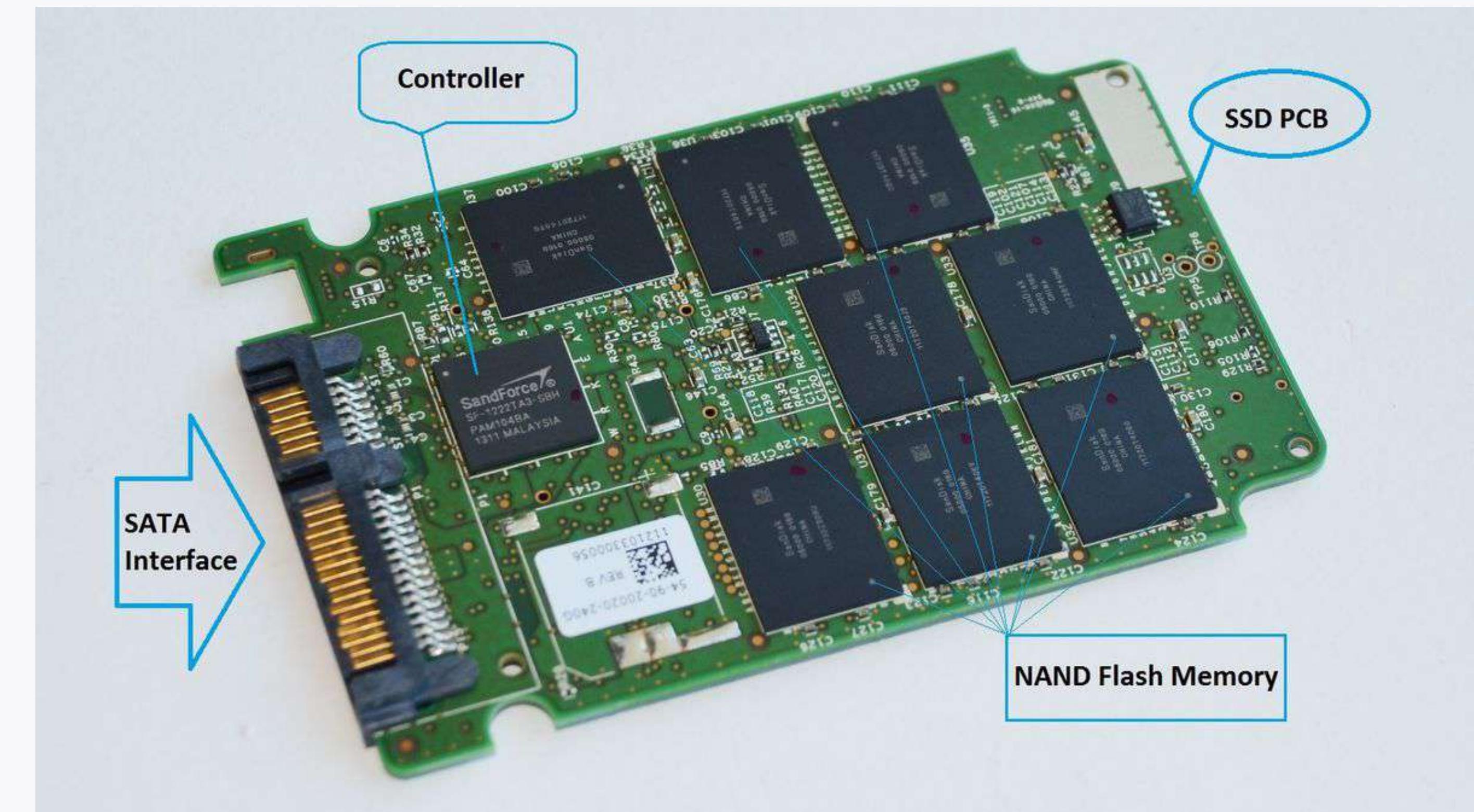
- **Hard Disk Drive** (aka spinning rust) is an electro-mechanical data storage device that stores and retrieves digital data using magnetic storage and one or more rigid rapidly rotating platters coated with magnetic material
- When data is being recalled, a small read / write head is rapidly moved around the platters looking for the non-contiguous data
- Due to there being something physically moving, they're prone to failure



© 2013 Encyclopædia Britannica, Inc.

Solid State Drive

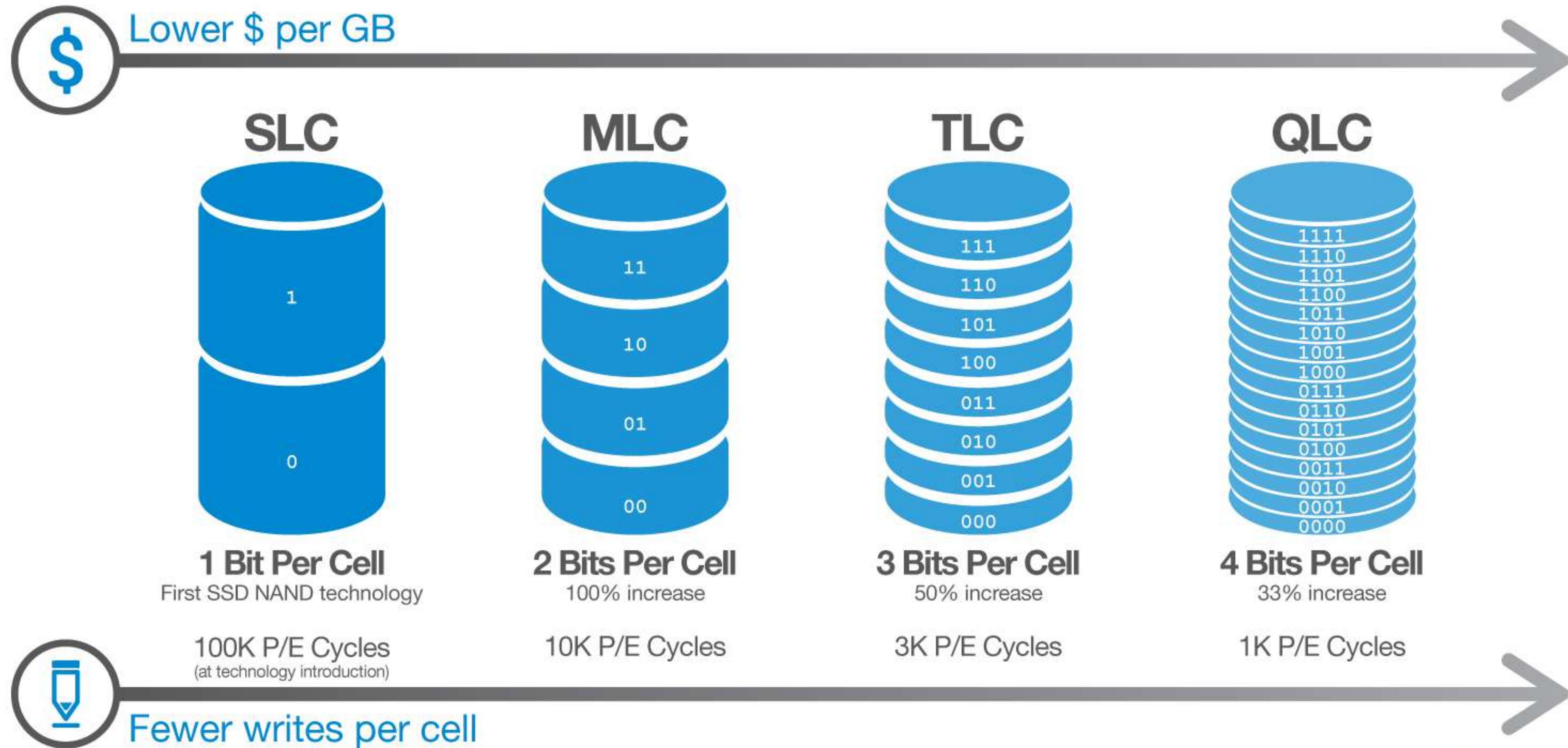
- A **solid-state drive (SSD)** is a solid-state storage device that uses integrated circuit assemblies to store data persistently, typically using flash memory
- Data is stored in blocks of **cells** in which bits can be stored inside
- Due to the nature of how data is stored, SSDs have a limited lifetime number of writes, and also slow down as they reach their full storage capacity



- When we talk about SSDs we talk about the type of Flash memory they're built with and how many bits can fit into a single cell
- The less data per cell, the lower density and therefore a higher cost per GB

Cell Type	Full Name	How Many Bits Per Cell	Write Capacity
SLC	Single Level Cell	1	100, 000 Writes
MLC	Multi-level Cell	2	10, 000 - Planar 35, 000 - 3D NAND
TLC	Triple-level Cell	3	3, 000 Writes
QLC	Quad-level Cell	4	1, 000 Writes

QLC = More Density Per NAND Cell



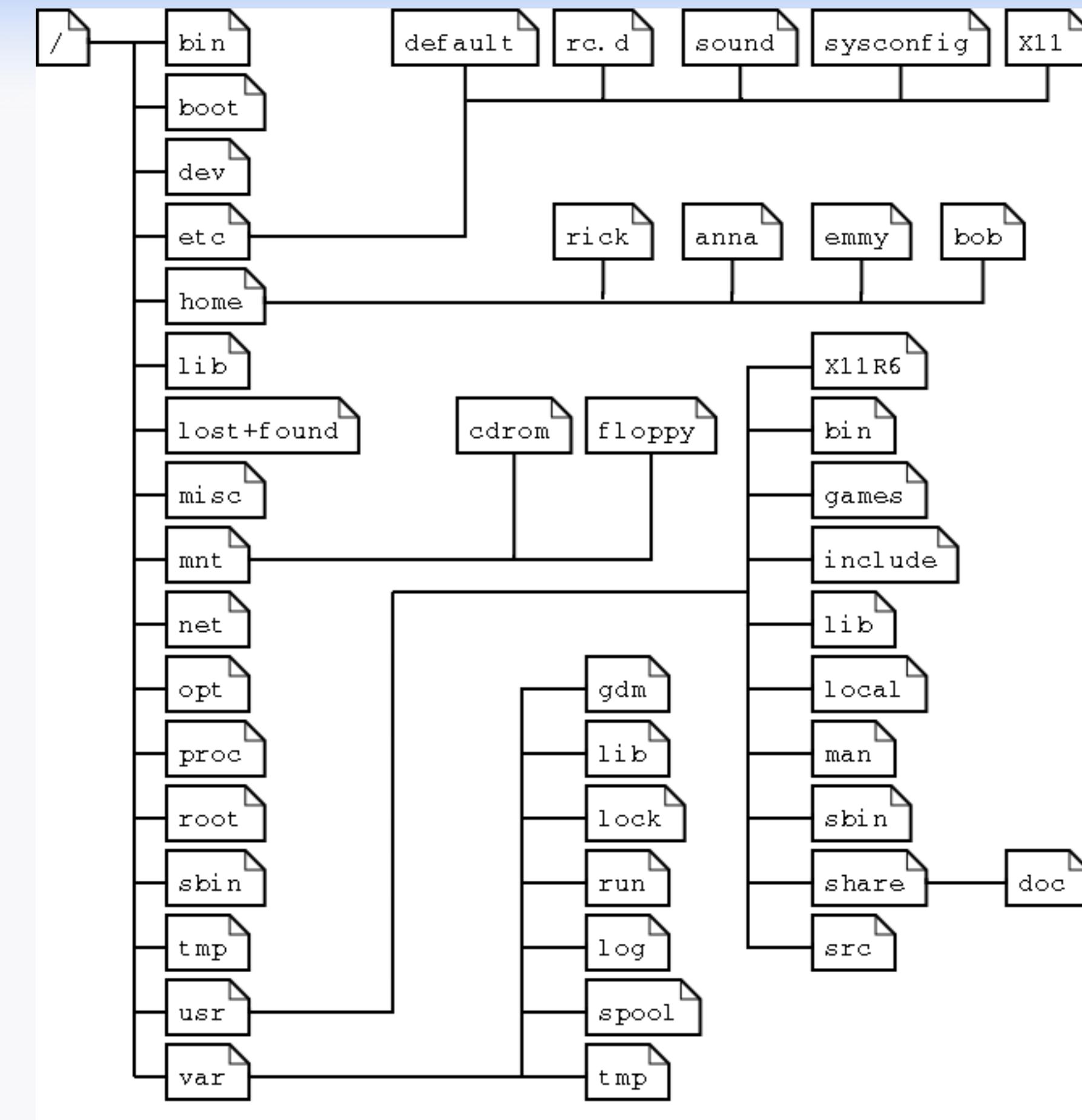
Micron®

SSD Write Amplification

- One thing I want to mention is how data is stored on SSDs
- Data is stored in blocks of typically 4KBs
- When data is written to the device all of that block has to be written at once, regardless if it's 4KBs worth of data
- The issue is when the device gets full it needs to read multiple blocks (and write) before it can find enough storage to write the data being requested
- This amplifies the amount of writes to the SSD which wears it out faster and makes it slower since it has to do multiple operations to disk

Linux File Structure

Linux File Structure



/ - Root

- The root directory.
- The starting point of your directory structure.
 - Think the **C: drive in Windows**
 - This is where the Linux system begins.
 - Every other file and directory on your system is under the root directory.
 - Usually the root directory contains only subdirectories, so it's a **bad idea to store single files directly under root**.

/boot - Boot Loader Files

- As the name suggests, this is the place where Linux **keeps information that it needs when booting up.**
- The **Grub Bootloader** is under this directory
- For example, this is where the Linux kernel is kept. If you list the contents of /boot, you'll see a file called **vmlinuz** - that's the **kernel**.

/etc - Configuration Files

- This is where, normally, all the configuration files for Linux applications are held
- A couple notable sub-directories or files are:
 - **/etc/passwd**
 - A file that contains various pieces of information for each user account
 - **/etc/fstab**
 - This file contains all the file system mount points
 - So /dev/sda1 becomes mounted at /

- **/etc/shadow**
 - stores actual password in encrypted format for user's account with additional properties related to user password
- **/etc/init.d/**
 - This is where all the startup scripts for applications go
 - Will start an application on boot if given the correct run level

/bin; /usr/bin - User binaries

- These two directories contain a lot of programs (binaries, hence the directory's name) for the system.
- The /bin directory contains the **most important programs that the system needs to operate**, such as the **shells, ls, grep, and other essential things**.
- /usr/bin are binaries for desktop type applications, usually

/sbin; /usr/sbin - Admin binaries

- Most **system administration programs** are stored in these directories.
- Example programs are:
 - reboot
 - poweroff
 - fdisk (disk formatter)
- In many cases you must **run these programs as the root user.**

/lib - Libraries

- The shared libraries for programs that are dynamically linked.
- The shared libraries are similar to Windows Dynamically Linked Libraries (DLL's)

/home - User Directories

- This is where **users keep their personal files**.
- Every user has their own directory under /home, and usually it's the **only place where normal users are allowed to write files**.
- You can configure a Linux system so that normal users can't even list the contents of other users' home directories.

/root/- Root User Directory

- The **superuser's** (root's) home directory.
- Don't confuse this with the root directory (/) of a Linux system.

/var - Variable Files

- This directory contains variable data that changes constantly when the system is running.
- Some interesting subdirectories:
 - **/var/log**
 - This is where Linux dumps all its log files
 - If something goes wrong, check here first
 - **/var/mail**
 - Incoming and outgoing mail is stored in this directory.
 - **/var/spool**
 - This directory holds files that are queued for some process, like printing.

/dev - Device Files

- The devices that are available to a Linux system.
- Remember that in Linux, **devices are treated like files** and you can read and write devices like they were files.
- For example, **/dev/fd0** is your first floppy drive, **/dev/cdrom** is your CD drive, **/dev/sda** is the first SATA hard drive, and so on.
- **All the devices that a Linux kernel can understand are located under /dev**, and that's why it contains hundreds of entries.

/proc - Process Information

- This is a special directory.
- Well, actually **/proc is just a virtual directory**, because it doesn't exist at all!
- **It contains some info about the kernel itself.**
- There's a bunch of numbered entries that correspond to all processes running on the system, and there are also named entries that permit access to the current configuration of the system.
- Many of these entries can be viewed.
 - Example: **\$ cat /proc/cpuinfo** gives you information about your current CPU.

/lost+found - Data Recovery

- Here Linux keeps the files that it restores after a system crash or when a partition hasn't been unmounted before a system shutdown.
- This way you can recover files that would otherwise have been lost.

Types of Links

Pointers

- In computer science, a **pointer** is a programming language data type whose value refers directly to (or "points to") another value stored elsewhere in the computer memory using its address
- Example while looking at the contents of a directory notice two files
- `../` is a file that **points to the previous directory**
- `./` is a file that points to the **current working directory**

Symbolic Link

- This is a pointer to a file or folder from another file or folder
 - An example is a folder pointing to the another folder on another hard drive
 - Think short cuts
- This symlink can cross file systems and storage mediums
- The issue is if the object the symlink is point to is moved then the symlink will break
- Example: **\$ ln -s ~/Documents /media/Storage**

Hard Link

- This is a pointer that points a single file to another single file on the SAME hard drive
- The binds the two files together so that if the target file were to move, the link would not break
- Example: **\$ ln SystemLog.log /var/log/syslog**

Drive Naming Scheme

Drive Naming Scheme

- In Linux hard drives are named in order they appear from the mother board
- Each motherboard has a **sata 0 port** (that's the first) then a sata 1 port and so on
- Any hard drive that's found on the sata 0 port is named **/dev/sda** (sata device a)
- Any hard drive that's found on the sata 1 port is named **/dev/sdb** (sata device b) and so on and so forth

SDA1

- The number that follows the `/dev/sda` is **the partition number on the drive**
 - For example if the hard drive has 3 partitions
 - `/dev/sda1`
 - `/dev/sda2`
 - `/dev/sda3`
 - Sometimes there can be a gap between the partition numbers, this is due to how the drives were partitioned in the first place
 - Even if there is a gap from say 2 to 5 the 5th partition is still considered the 3rd partition on the disk

Swap Space

Swap Space

- In Linux a swap space in a hard drive is where excess memory is stored if the hosts memory is running low or unused in a while
 - Ex. If you have 8GBs of memory and are currently using 9GBs, **1GB worth of memory will be saved on the hard drive**
 - In windows this is called a page file
 - This brings the entire system **down to the speed of the disk drive**, which compared to ram is considerably slower
 - Up to 25.6GB/s for DDR4 to ~100MB/s for HDD or ~550MB/s for SSD or 7.6GB/s M.1 SSDs

Data Redundancy

The Problem

- If you were to design a system that held data critical to the company how would you put in safeguards to establish data integrity?
- Would you span the data across multiple files?
 - Folders?
 - Hard Drives?
 - Servers?

Raid and mdadm

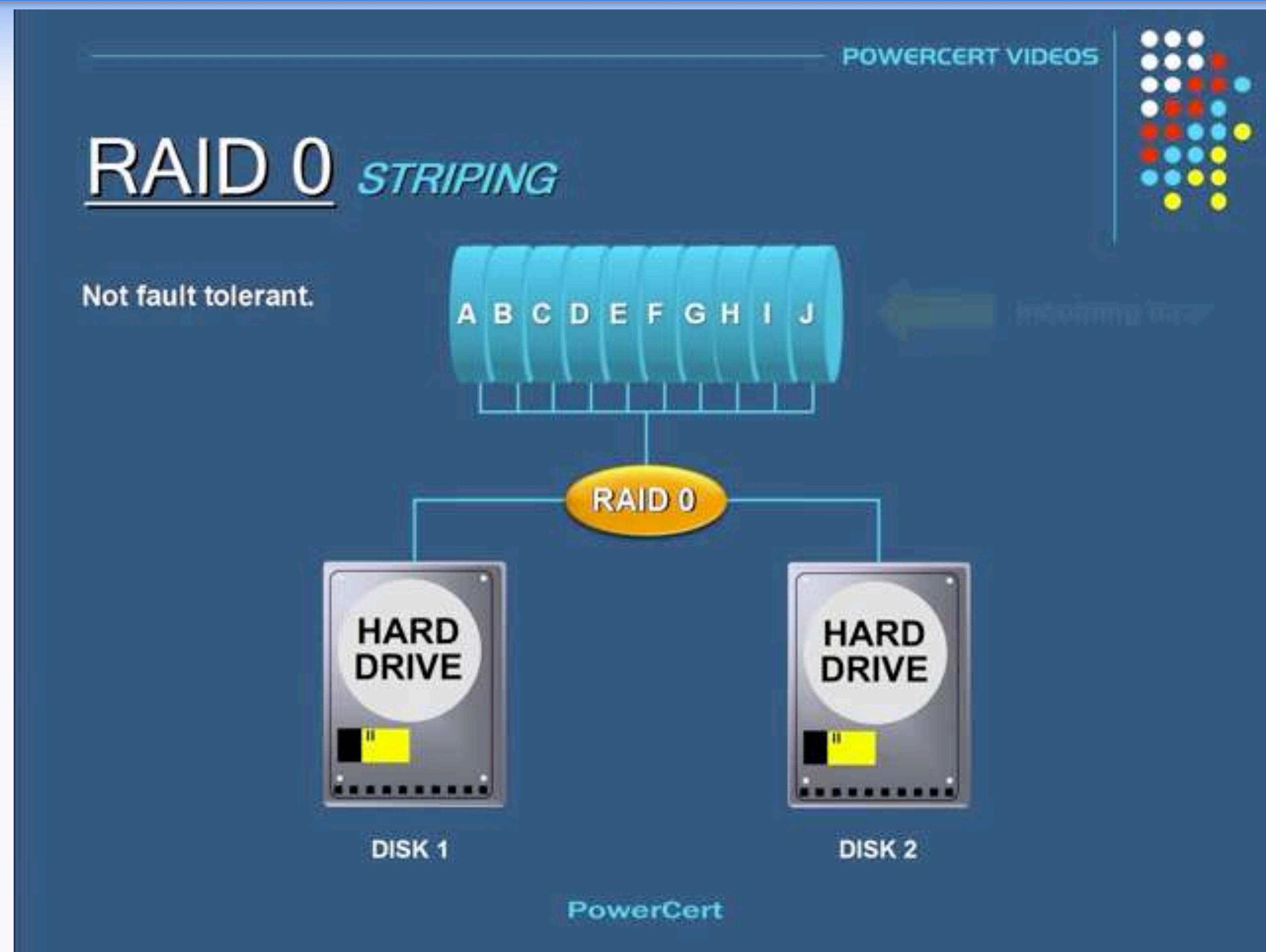
What is raid?

- **RAID** (A redundant array of independent disks) is a way to link multiple hard drives together to achieve a common goal
- That goal may be for speed, redundancy, or integrity
- There are generally 4 types of common raid implementations in order to achieve this

Raid 0

- **Raid 0** is defined by a block-level striping without parity or mirroring and has no redundancy
 - Think it as **0 redundancy** or death wish raid
- This form of raid combines 2 or more drives then writes alternatively to each
 - Write 1 goes on drive 1
 - Write 2 goes on drive 2
 - Write 3 goes on drive 1
- This allows all the space on both drives to be accessible
- If any of the disks fail then the entire data set is lost

Raid 0



Raid 1

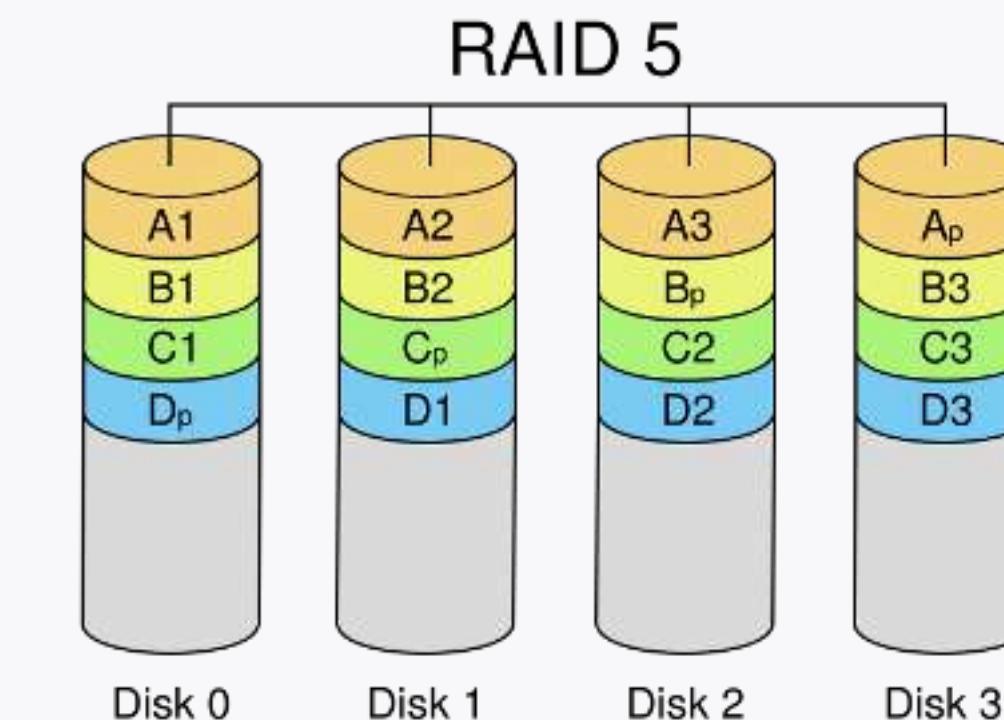
- Raid 1 is defined by Mirroring without parity or striping.
- This means that 2 or more drives are **mirrored together**
- This means that you only get **1 hard drives worth of space** but if one of the set fails, the other will replace it
- The fastest drive will serve read operations
- The data is **written to all drives at the same time**
- The slowest drive determines write speed

Raid 1

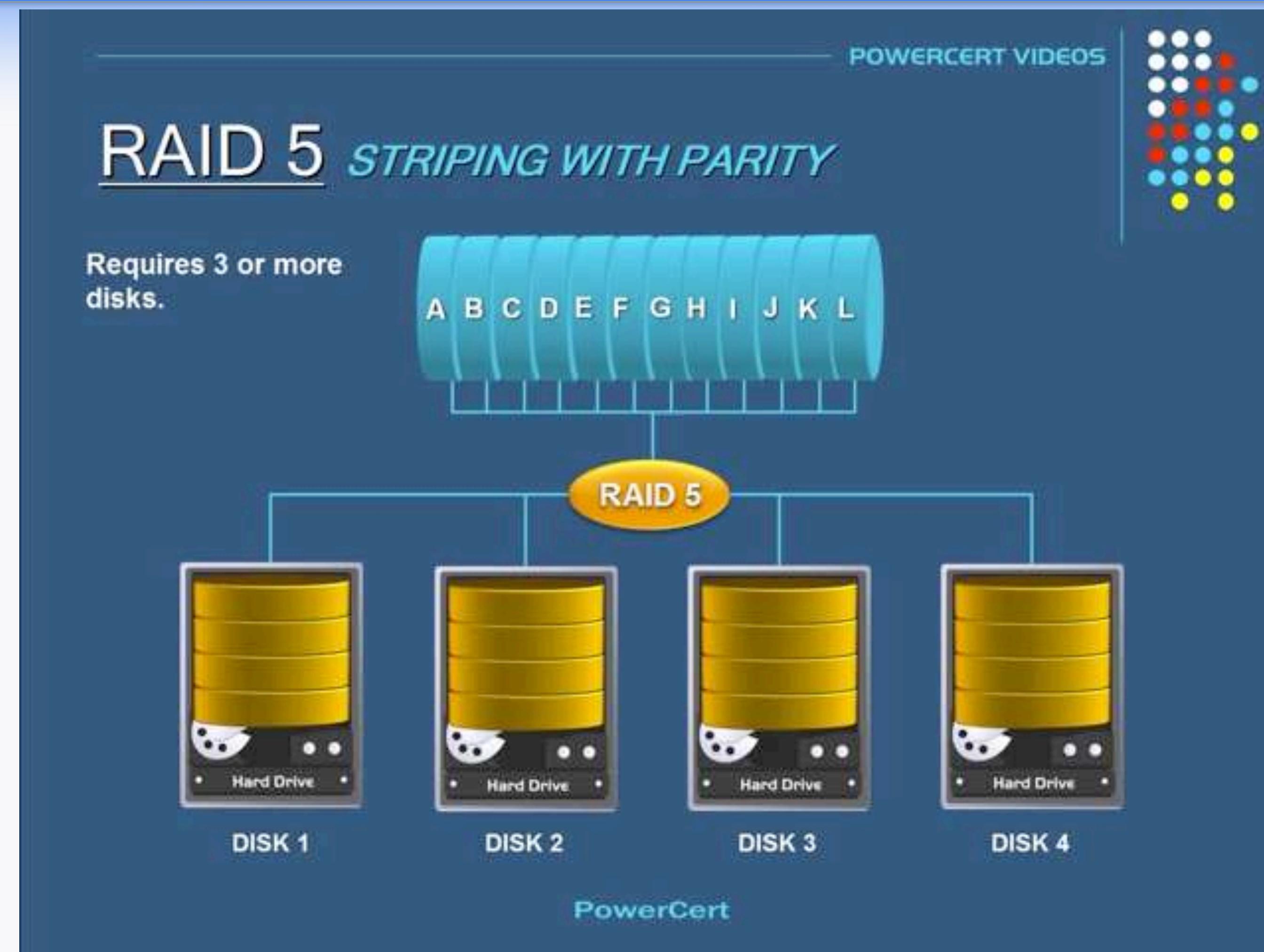


Raid 5

- Raid 5 is defined by block-level striping with distributed parity amongst the drives
- This requires at least 3 drives and will give you the equivalent of 2 drives worth of space
- The third drive will act as a parity device recovering data from any lost drive in the array
 - This is computationally expensive to do
 - This can suffer a **single drive failing**

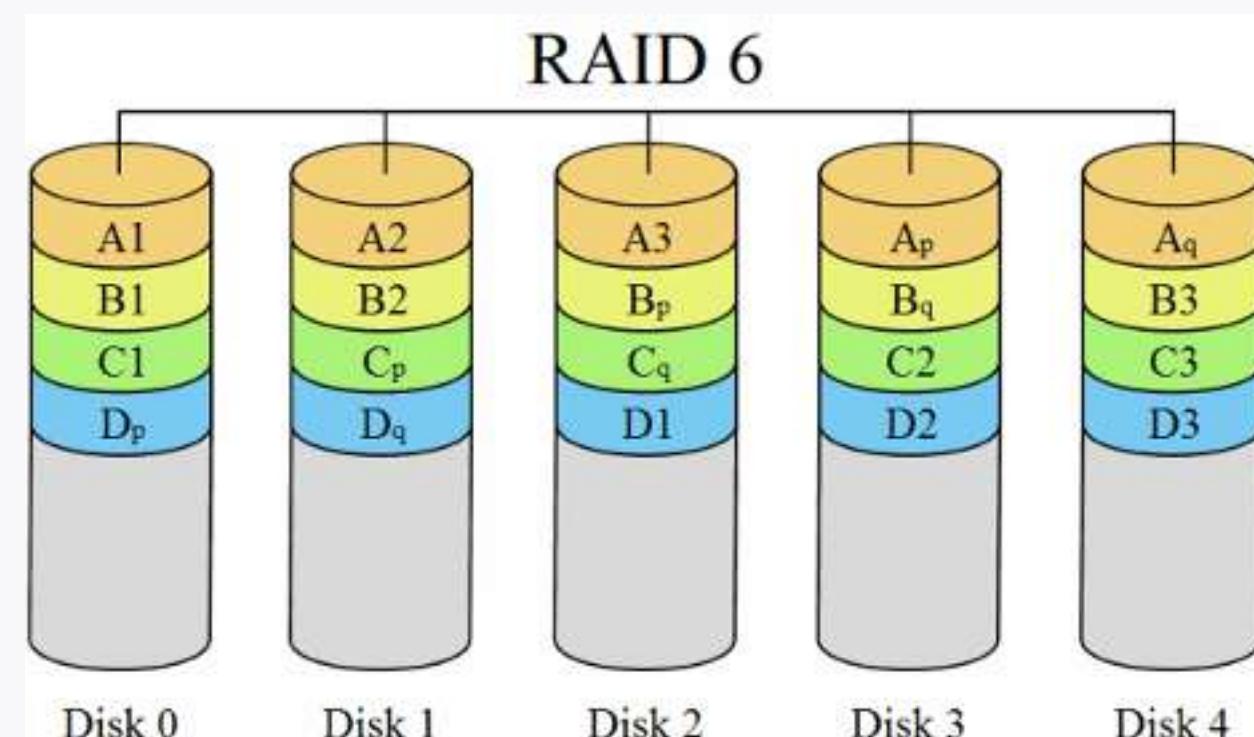


Raid 5



Raid 6

- **Raid 6** is defined by block-level striping with double distributed parity amongst the drives
- This type of raid requires at **least 4 drives** and gives you the equivalent data storage of **$N-2$ drives worth of space**
 - $4 * 2\text{TB}$ in raid 6 gives you 4TBs of space
 - Since the parity data is striped against all the drives it can suffer from **2 drives failing** before it fails itself
 - If a single drive fails the entire array will have reduced performance
 - In my opinion this is the safest bet



mdadm - Software Raid

- **mdadm** (multiple device administration) is a software only implementation of raid.
- It is a tool in which you can create raid 0, raid 1, raid 5, raid 6, and a few other raids without using a dedicated hardware controller
- It allows you to raid partitions or entire disks, depending on the need of the administrator

Hardware Raid

- Hardware raid controllers have **batteries to let them finish the current write** in a power failure
- Hardware raid controllers have **onboard ram in order to cache large writes** before sending the operation to the drives
- Hardware raid controllers have the ability to service **more hard drive connections than the motherboard has sata ports**
- Ex a hardware raid breakout controller can have 24 ports instead of the 6 on the motherboard



Network Based Raid

- There is also the ability to do network based raid
- This is usually done for servers who wish to have both an onboard raid and a backup of it offsite
- So far **DRBD (Distributed Replicated Block Device)** is the defacto standard for network based raid 1

Package Management

Package Management

- A package manager is a **collection of software tools that automate the process of installing, upgrading, configuring, and remove computer programs in a consistent manner**
- The packages are archived files stored remotely on a server which can be found from your sources file (/etc/apt/sources.list)
- The package manager will:
 - Download the file
 - Read the package metadata
 - Install dependency Packages if needed
 - Move the files inside to the correct location on disk

- With Ubuntu we typically use a package manager called **Advanced package tool** or typically known as **APT**
- **APT** is typically distributed with Debian based distros but can be found on non-debian distros as well
- A couple of other common package managers are:
 - **Yellowdog Updated, Modified (YUM)** typically found on **Red Hat / Fedora** based distros
 - **Pacman** typically found on **Arch Linux**

Package Manager	How To Update Sources	How to Install a package	How to Upgrade a Package	How to Remove a Package
APT	sudo apt-get update	sudo apt-get install vim	sudo apt-get upgrade vim	sudo apt-get remove vim
YUM	sudo yum check-update	sudo yum install vim	sudo yum update vim	sudo yum remove vim
PACMAN	sudo pacman -Syy	sudo pacman -S vim	sudo pacman -S vim	sudo pacman -R vim

- Since everything on Linux is a package we can update or install anything we need to
- If we wanted to say update everything on our Ubuntu system we'd run the following three commands:
 - **\$ sudo apt-get update**
 - **\$ sudo apt-get upgrade -y**
 - **\$ sudo apt-get dist-upgrade -y**

CLI Editor

CLI Editor

- For example how often do you find yourself in this situation
 - You download the website files via Filezilla (FTP)
 - You modify the files on your computer
 - Upload the modified files to the web server
 - Test
 - Find there's an error
 - Modify the files again
 - Upload them again
- Why not just edit them on the server and watch the changes in real time?
- **Never ever ever** do this on production, use GIT.....

Vi and VIM

Vi and VIM

- Vi is a screen-oriented text editor originally created for the Unix Operating System
- It was originally created as the visual mode for the line editor called ex
- The name vi is derived from the shortest unambiguous abbreviation for the command visual in ex
- We use **VIM (Vi Improved)** which is just VI but better

Different Modes

Visual Mode

- This is the mode in which you can issue commands to manipulate the file that is currently open
 - An example of that is replacing all words matching "foo" with the word "bar"
 - **:% s/foo/bar**
 - Another example is deleting from the cursor to the end of the line
 - **<shift> + <d>** or Capital D
 - Need a new line above the one you're currently on?
 - **<shift> + <o>** or Capital O

A brief note

- Whenever you see a letter or word surrounded by <> that means key on the keyboard
- Don't actually type < or > unless otherwise specified

Macros

- In order to start a macro you must first hit "q" then any letter to associate the macro with a register
 - Ex: qa
- Then you will type out the commands you wish to macro
 - Ex: you want to make 20 lines with incrementing numbers
 - test 1
 - qa -> yyp -> ctrl+a -> q
 - 18@a
 - That will take the line "test 1" copy it, increment the number 2 to then do that 18 more times.

Line Editor

- In this mode you are actually typing in information
- In order to enter this mode, you must hit "<i>" in visual mode
- In order to exit this mode, you must hit "<esc>"

Save and Exit

- In order to save and exit you first need to enter command mode
- Command mode is entered using the colon key (:)
 - to save use: <: > + <w> or w
 - to force save use: <: > + <w> + <shift><1> or w!
 - to exit use: <: > + <q> or q
 - to force exit use: <: > + <q> + <shift><1> or q!

Where to use VIM?

- We use VIM a lot when we are editing files remotely on a server
- Normally you'll be editing config files such as an apache virtual host (/etc/apache2/sites-enabled/default.conf) or writing short scripts remotely
- VIM is powerful and useful but hard to pick up
- Once you've picked it up, you'll be glad you did

In Summary...

In Summary

- “Linux” is just a kernel, the whole stack is called GNU/Linux
- The VFS abstracts so we can slot in any File System we need depending on the situation
- RAID is not a backup
- VIM is an important skill to learn

Questions?

Lecture 6 - Linux Automation

Terminology

Terminology

- **!** is called a Bang
- **>** is called a wakka or a carrot
 - Comes from Pacman
- **#** is called a crunch
- **~** is called a tilde
- **|** is called a pipe
- **-** is called a tack

Exit Codes

Exit Codes

- Every Linux or Unix command executed by the shell script or user, has an exit status.
- The exit status is an **integer number**.
- For the bash shell's purposes, a command which **exits with a zero (0)** exit status has **succeeded**.
- A **non-zero (1-255)** exit status indicates **failure**.
- If a command is not found, the child process created to execute it returns a status of 127. If a command is found but is not executable, the return status is 126.

- To catch the previously ran command you use the special operator: \$?
- When scripting we use these exit codes to determine if we should continue with the program or abort due to an error

Exit Code	Description
0	Success
1	Operation not permitted
2	No such file or directory
3	No such process
4	Interrupted system call
5	Input/output error
6	No such device or address
7	Argument list too long
8	Exec format error
9	Bad file descriptor
10	No child processes
11	Resource temporarily
12	Cannot allocate memory
13	Permission denied
14	Bad address
15	Block device required
16	Device or resource busy
17	File exists
18	Invalid cross-device link
19	No such device
20	Not a directory
21	Is a directory
22	Invalid argument
23	Too many open files in system
24	Too many open files

Commonly Used Commands

MAN

MAN

MAN

- The UNIX Programmer's Manual was first published on November 3, 1971
- A man page (short for manual page) is a form of online software documentation usually found on a Unix or Unix-like operating system
- Usage:
 - **\$ man <command_name>**

LS

- The ls command, which lists files, is one of the most essential utilities for Unix and Linux users and, not surprisingly, one of the oldest.
- In its earliest form it was called listf and was available on the Massachusetts Institute of Technology's Compatible Time Sharing System (CTSS) by July, 1961

LS Usage

- **\$ ls**
 - this will list all contents of the folder in alphabetical order
- **\$ ls -l**
 - This will list all the contents in a list format, also in alphabetical order
- **\$ ls -la**
 - This will list all the contents in a list format,
 - showing hidden files
 - in alphabetical order

Less

less

Less

- More came first
 - More is a filter for paging through text one screenful at a time. This version is especially primitive
 - **\$ more <file>**
- Less is more
 - opposite of more
 - Less is a program similar to more, but which allows backward movement in the file as well as forward movement.
 - Also, less does not have to read the entire input file before starting, so with large input files it starts up faster than text editors like vi.
 - More is less than less, but less is more.

Other Commonly Used Commands

- **\$ cat <file>**
 - concatenate files and print on the standard output
- **\$ cp <file> <file_destination>**
 - Copies files and directories
- **\$ rm <file>**
 - Removes Files or directories

- **\$ grep PATTERN <file>**
 - Prints lines with a matching pattern
 - Ex: **\$ grep john students.txt**
- **\$ head <file>**
 - Output the first part of a file
- **\$ mv <file> <file_destination>**
 - move (rename) files
 - Ex: **\$ mv students.txt former_students.txt**

Standard Streams

Standard Streams

- Standard streams are interconnected input and output communication channels between a computer program and its environment
- It was at one point a physical connection but has since been abstracted to the virtual
- The standard three I/O connections are:
 - **Standard Input (stdin)**
 - **Standard Output (stdout)**
 - **Standard Error (stderr)**

Standard Streams - stdin

- **Standard input (stdin)** of data (Usually text) into a program
- The Program requests data by use of the READ function
 - A command in linux wants some kind of input
 - Ex: **\$ cat <file>**; file being the standard input
- Not all commands require input
 - Ex: **\$ ls**

Standard Streams - stdout

- **Standard output (stdout)** is the stream where a program writes its output data.
- The program requests data transfer with the write operation
- Not all programs give you an output based on successful completion
 - Ex: **\$ mv <file> <file_destination>**
 - A majority of programs can have the switch for verboseness turned on
 - **\$ mv --verbose <file> <file_destination>**

Standard Streams - stderr

- **Standard error (stderr)** is another output stream typically used by programs to output error messages or diagnostics.
- It is a stream independent of standard output and can be redirected separately.
- This comes in handy when chaining commands together

Redirection

Discussion

Redirecting Output

- A Linux shell, such as Bash, receives input and sends output as sequences or streams of characters.
- One can take the stdout and redirect it to stdin on the next program

Redirecting Examples

- **\$ echo hello > I_Love_Linux.txt**
 - redirects output from echo to a file.
 - You must have write authority to the file.
 - If the file does not exist, it is created.
 - If it does exist, the existing contents are usually lost without any warning.
- **\$ echo world >> I_Love_Linux.txt**
 - also redirects output from echo to a file.
 - You must have write authority to the file.
 - If the file does not exist, it is created.
 - If it does exist, the output is appended to the existing file.

Piping Commands

- Piping is an ability to take the stdout from one command and use it as stdin on another.
- This allows for you to, for example, issue a command than parse its output
- Ex: **\$ ls -la | grep students.txt**
 - **-rw-rw-r-- 1 raymer raymer 99 Dec 32 25:49 students.txt**
 - It will first list all the files and directories in the folder
 - then will pipe that list into grep to look for the students.txt file

Chaining Commands

- What if you wanted to run a command and when it finished, run another?
- Linux provides three ways to do this with a slightly different end result.
 - **semicolon (;**)
 - **ampersand (&)**
 - **vertical pipe (|)**

The semicolon (;

- Works as a line break in a command chain
 - Ex: **\$ mkdir tmp; cd tmp; ls**
 - This will make the folder named tmp
 - Change directory to the folder tmp
 - and list all the files within the folder tmp
 - Executes the next command regardless of any errors the first command encounters
 - Ex: **\$ cd /root/; mkdir tmp; cd tmp; rm -r tmp;**
 - Assuming this is a user account it will try and enter the folder /root/ which is off limits to users
 - Everything else will execute

Ampersand (&)

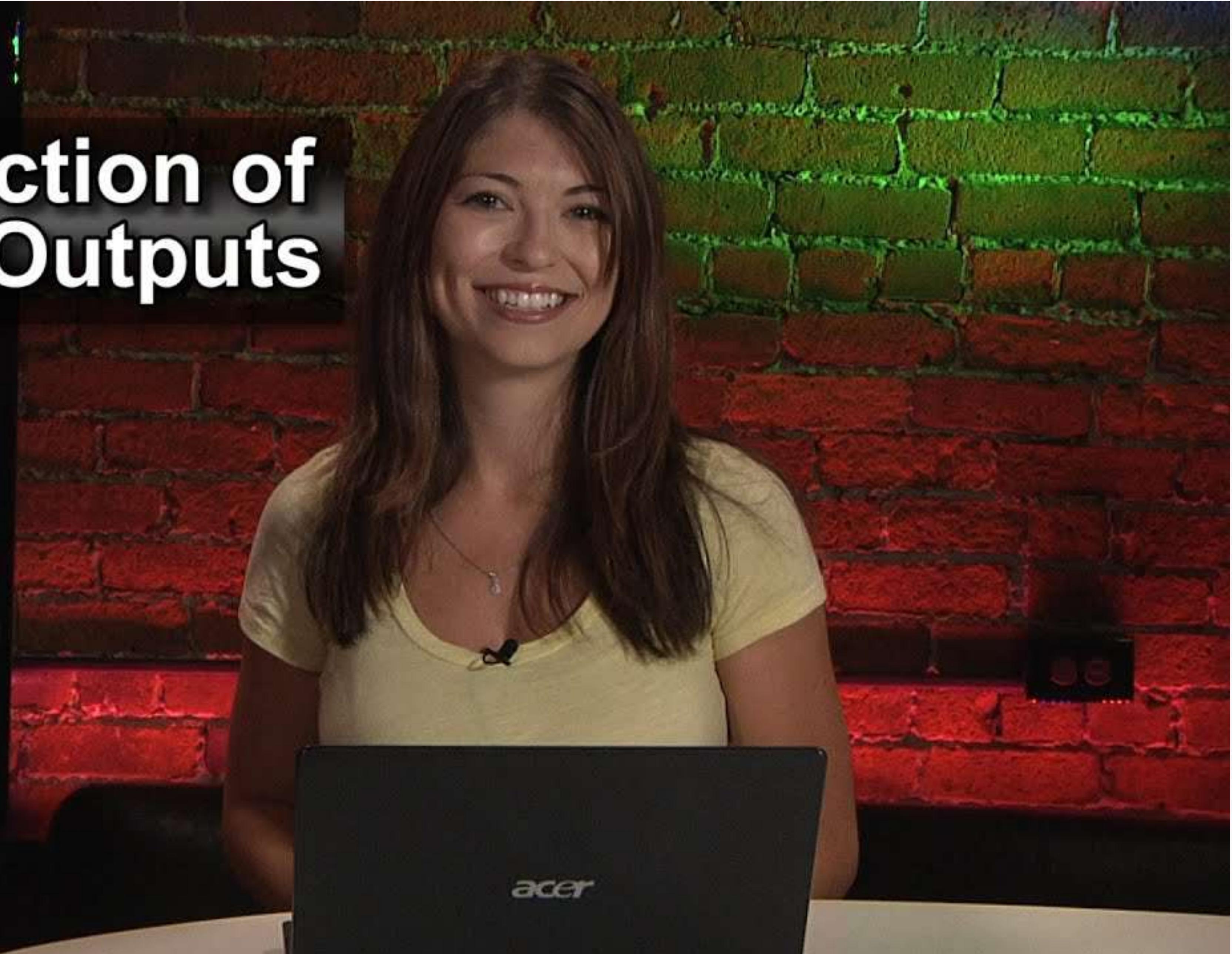
- This will execute commands in order ONLY if the previous hasn't exited with 0
- Ex: **\$ mkdir tmp && cd tmp && ls**
 - That will execute in order
- Ex: **\$ cd /root/ && mkdir tmp && cd tmp && rm -r tmp**
 - This will stop the moment the previous commands exits with a non 0 exit code

Vertical Pipe (|)

- As mentioned before this takes the stdout and redirects it as stdin for the next command
- Ex: **\$ ls | grep john.txt**
 - All the information from ls is now stdin for the grep function
 - Only works if the previous command doesn't return a non 0 exit code

I/O Redirection of Standard Outputs

TAKE IT



File Permissions

File Permissions

```
[raymer@ns1 ~]$ ls -la
total 32
drwx----- 4 raymer raymer 4096 Nov 23 17:13 .
drwxr-xr-x  4 root   root   4096 Nov 22 22:58 ..
-rw----- 1 raymer raymer    49 Nov 22 23:31 .bash_history
-rw-r--r-- 1 raymer raymer   18 May 10 2012 .bash_logout
-rw-r--r-- 1 raymer raymer  176 May 10 2012 .bash_profile
-rw-r--r-- 1 raymer raymer  124 May 10 2012 .bashrc
drwxrwxr-x  2 raymer raymer 4096 Nov 23 17:13 course
-rw-r--r-- 1 raymer raymer     0 Nov 23 17:13 Linux.doc
-rw-r--r-- 1 raymer raymer     0 Nov 23 17:13 linux_II.txt
drwx----- 2 raymer raymer 4096 Nov 22 23:28 .ssh
```

- Each of the columns has a meaning and are read left to right as normal
- First digit is the **File Type**
- The next 9 digits are the permission bits

```
drwx-----.
drwxr-xr-x.
-rw-----
-rw-r--r--.
-rw-r--r--.
-rw-r--r--.
drwxrwxr-x.
-rw-rw-r--.
-rw-rw-r--.
drwx-----.
```

File Type

- **d** = directory
 - Just a typical folder
 - **-** = regular file
 - Just a typical file
 - **l** = symbolic link
 - A link to another directory
 - **s** = Unix domain socket
 - A data communications endpoint for exchanging data between processes executing

```
drwx-----.
drwxr-xr-x..
-rw-----.
-rw-r--r--.
-rw-r--r--.
-rw-r--r--.
drwxrwxr-x..
-rw-rw-r--.
-rw-rw-r--.
drwx-----.
```

- **p** = named pipe
 - A file in which information is piped to
- **c** = character device file
 - A device driver that appears in a file system as if it were an ordinary file
- **b** = block device file
 - A block device is an array or disk

```
drwx-----.
drwxr-xr-x.
-rw-----
-rw-r--r-- .
-rw-r--r-- ,
-rw-r--r-- ,
drwxrwxr-x.
-rw-rw-r-- .
-rw-rw-r-- .
drwx-----.
```

Permission Bits

- You read them in groups of 3
- Ex. **rwx r-- r--**
- Starting with the first R
- The first set of 3 letters represent **User permissions**
- The second set of 3 represent **group permissions**
- The third set of 3 represent **other permissions**
- Other is basically **global permissions**

```
drwx-----.
drwxr-xr-x.
-rw-----.
-rw-r--r--.
-rw-r--r--.
-rw-r--r--.
drwxrwxr-x.
-rw-rw-r--.
-rw-rw-r--.
drwx-----.
```

RWX

- **r** = read permission
 - Allows it to be read
- **w** = write permission
 - Allows it to be written to
- **x** = execute permission
 - Allows the script to be executed
- **-** = no permission
- Note: Directories must have the executable bit enabled for them to work

```
drwx-----.
dRWXr-xr-x.
-rw-----.
-rw-r--r--.
-rw-r--r--.
-rw-r--r--.
drwxrwxr-x.
-rw-rw-r--.
-rw-rw-r--.
drwx-----.
```

Modifying Bits

Modifying Bits

- The application we use to modify permissions is **chmod**
 - It's been around since the first version of unix
 - To change permissions you first tell it which set of 3 it will be modifying and if you're removing or adding permissions
 - Ex: **\$ chmod u+rwx test.txt**

- Ex: **\$ chmod o+rwx example_script.sh**
- Ex: **\$ chmod o-rwx example_script.sh**
- Ex: **\$ chmod u+rwx,o-rwx example_script.sh**
- You can string together permission changes using a comma to separate
- Like anything else if you wish to apply permissions against an entire directory you'll have to assign the recursion flag to do so
 - Ex: **\$ chmod -r o+rwx /opt/share/**

Which user?	
u	user/owner
g	group
o	other
a	all
What to do?	
+	add this permission
-	remove this permission
=	set exactly this permission
Which permissions?	
r	read
w	write
x	execute

Binary

Binary Overview

- Let's take the Binary 10010011:
 - We count right to left instead of left to right
 - Use the chart below to find out what the answer is?

Bit #	8	7	6	5	4	3	2	1	0
Power	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Decimal Value	256	128	64	32	16	8	4	2	1
Question	0	1	0	0	1	0	0	1	1

- The answer: 147
- In linux permissions are stored in binary in the header of the file
- since we have **9 characters for permissions** (not including special bits like directory) that makes up for **3 bits per grouping**
- What is the highest number 3 bits can add up to?

- Since the highest it can be is 7 that means our permissions are **a number out of 7**
- 4 = **read (r)**
- 2 = **write (w)**
- 1 = **execute (x)**
- 0 = **no permission (-)**

- You can assign permissions just like the letter representations but using numerical value
- The catch is you **must assign all 9 bits at once**
- Ex: **\$ chmod 755 file**
 - 7 = User read, write, execute
 - 5 = Group read, execute
 - 5 = Other read, execute

- How would I write these out in numeric value?
 - -rw-r----- 1 ender admin 8187 2099-12-32 25:35 file
 - Convert to binary format: chmod u+rwx,g+rw,o+r file
 - **\$ chmod 764 file**
 - Convert to binary format: chmod u-rw,g-r file
 - **\$ chmod 000 file**
 - Convert to binary format: chmod a=rwx file
 - **\$ chmod 777 file**

```
[raymer@ns1 ~]$ ls -la
```

```
total 32
```

```
drwx----- 4 raymer raymer 4096 Nov 23 17:13 .
drwxr-xr-x. 4 root root 4096 Nov 22 22:58 ..
-rw----- 1 raymer raymer 49 Nov 22 23:31 .bash_history
-rw-r--r--. 1 raymer raymer 18 May 10 2012 .bash_logout
-rw-r--r--. 1 raymer raymer 176 May 10 2012 .bash_profile
-rw-r--r--. 1 raymer raymer 124 May 10 2012 .bashrc
drwxrwxr-x. 2 raymer raymer 4096 Nov 23 17:13 course
```

Type	User	Group	Global	Number Of Links	Owner	Group	Size	Last Modified Date	File Name
d	rwx	r-x	r-x	2	raymer	raymer	4096	Nov 22 22:58	course
-	rw-	r--	r--	1	raymer	raymer	176	Nov 22 22:58	.bash_profile

Ownerships

Ownerships

- File belong to someone, hence the first set of bits
- Just like chmod you can change the ownership of a file to another person
- The command is:
 - **\$ chown ender file**
 - This is for a single file
 - **\$ chown -R ender project/**
 - All the files within a directory
- This will give ownership (the U series of bits) to the ender in this case

Group Ownerships

- Say you were working on a project and you wish to assign the project files to have the group ownership; how would you do it?
- In linux it's:
 - **\$ chgrp admin file**
 - In this case it changes only a single file
 - **\$ chgrp -R admin project/**
 - This will give every object within the directory group ownership of admin

Remaining file fields

- **Size** is the actual size of the file in bytes
- **Last modified** is the a timestamp of when it was last altered (not created)
- **Filename** is the name of the reference

Type	User	Group	Global	Number Of Links	Owner	Group	Size	Last Modified Date	File Name
d	rwx	r-x	r-x	2	raymer	raymer	4096	Nov 22 22:58	course
-	rw-	r--	r--	1	raymer	raymer	176	Nov 22 22:58	.bash_profile

Basic Bash Scripting

Basic Bash Scripting

- **Bash** is a command language interpreter that was used to interact with the GNU Linux operating system
- **Bash** stands for ‘Bourne-Again SHell’
- Since when we interact with our **shell** we are actually issuing programming statements into the machine we can extrapolate and make small programs to do routine tasks for us

- In the world of Cloud Computer, it is critical to be able to read and write scripts to do repetitive tasks
- Bash scripts typically have the suffix of **.sh** but aren't required to
 - Ex: **/bin/nightly_backup.sh**
 - They also need to have the **execution bit** enabled for them to be able to run without an interpreter prefix
 - Ex: **\$ sudo chmod +x /bin/nightly_backup.sh**

- At the top of the script we need to add a **shebang** which directs the operating system to use the correct interpreter
 - **#!/bin/bash**
- If a script is named with the path `path/to/script`, and it starts with the following line, **#!/bin/bash**, then the program loader is instructed to run the program `/bin/bash`, passing `path/to/script` as the first argument

- I like to put a small commented out header at the top of all my scripts giving some basic information like:
 - what the script is suppose to do
 - original author
 - who last updated it
 - the version of the script
 - when it was first authored
 - and when it was last updated

- From here we just write commands as if you were issuing them directly into the shell but the commands have to be non-interactive (requiring user input)
- For example since it's a programming interface you have access to:
 - Variables
 - String interpolation
 - Conditional Statements (if, when)
 - Loops

- I've written an example script to use as a guideline, albeit using some more advanced techniques such as scoping, constants, and **whiptail**
- https://github.com/araymer-stclair/web401-lecture6-example-script/blob/main/setup_script.sh

Environment Variables

Environment Variables

- Every time a shell session spawns, a process takes place to gather and compile information that should be available to the shell process and its child processes
- It obtains the data for these settings from a variety of different files and settings on the system
- **Environment variables (ENV)** contain information about your login session, stored for the system shell to use when executing commands

- Environment variables are no different, technically, than variables
 - Typically they're all uppercase but don't have to be
 - In cloud computer we use Environment Variables heavily to pass information into applications to give them some context on how we want them to run
 - For example we normally pass a **software environment** to our application to tell them what configurations to run
 - Ex: **\$ NODE_ENV=production npm run start**
 - Ex: **\$ RAILS_ENV=staging bundle exec puma**

Path Variable

- The Path variable is a variable that each user account has on a linux system that tells the shell where to look for programs
- If you were to type:
 - **\$ echo \$PATH**
 - It's contents would be something like
 - **/home/raymer/bin:/usr/lib/lightdm/lightdm:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games**
 - Each of these is a directory path with executable binaries
 - If the program isn't found in one of these paths then it will not execute

CRON

CRON

CRON

- A **cronjob** is a way to schedule a script or task to run at a predetermined interval
- The file which handles it is:
 - **/etc/crontab**
- Within this file you can dictate the **minute, hour, day of the month, month, or even the day of the week** when the task is to run

```
student@web401-student:/var/www/html$ cat /etc/crontab
# /etc/crontab: system-wide crontab
# Unlike any other crontab you don't have to run the `crontab'
# command to install the new version when you edit this file
# and files in /etc/cron.d. These files also have username fields,
# that none of the other crontabs do.

SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# Example of job definition:
# .----- minute (0 - 59)
# | .----- hour (0 - 23)
# | | .----- day of month (1 - 31)
# | | | .---- month (1 - 12) OR jan,feb,mar,apr ...
# | | | | .--- day of week (0 - 6) (Sunday=0 or 7) OR sun,mon,tue,wed,thu,fri,sat
# | | | |
# * * * * user-name command to be executed
17 * * * * root    cd / && run-parts --report /etc/cron.hourly
25 6 * * * root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.daily )
47 6 * * 7 root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.weekly )
52 6 1 * * root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.monthly )
#
student@web401-student:/var/www/html$
```

- * indicates that it runs X amount of times for that time slot
 - minute hour day month weekday
 - * * * *
- How many times would this task run?
 - 60 minutes * 24 hours * every day of the month
* every month of the year * every week day
- After defining when it runs you define by **whom and the task**

- Examples

- **0 4 * * *** root **/bin/Backup.sh**

- This will run, as root, the backup script at 4AM, server time
 - What would the following do?

- **15 * * * *** root **traceroute google.ca**

- Traceroute google.ca every 15 minutes

- **0 4 14 * * *** root **/bin/backup.sh**

- at 4:00 AM on the 14th of each month it will backup your server
 - A great place for us to validate our crontab syntax is: <https://crontab.guru/>

Lastly Rsync

Rsync

- Sometimes we need to backup files remotely or even locally
- We use a program that was explicitly designed to synchronize a folder to another folder
- It is called "**Rsync**" and it's most commonly used to Backup entire servers or just single directories
- It can be used **over SSH**
- It can be used **locally**
 - This could be to a spare drive or a tape drive or something of this matter

- The syntax for Rsync is as follows:

- **\$ rsync -azvv -e ssh /etc/ username@127.0.0.1:/media/Backup**
 - **-a** is to archive the data
 - This means it preserves the timestamps and link files
 - **-z** enables compression while transferring
 - Lowers bandwidth consumption
 - **-vv** enables very verboseness
 - **-e ssh** this is the transport method
 - **/etc/** is the “**from**” folder path
 - **username@127.0.0.1:** username at the ip address of the remote target
 - **/media/ Backup** is the “**to**” folder path

- One of the major benefits from Rsync is the fact it does **differential backups**
 - What is a **differential backup**?
 - After the first backup, AKA **initial commit**, it will only back up the delta
 - This means it will only back up what has **changed since the last backup**
 - This means you only need **one copy of the data**, not multiple

In Summary...

In Summary

- Non-Zero Exit Codes indicates a failure, zero indicates a success
- File Permissions are a 3 set of 3 bits to determine **R**ead, **W**rite, or **e**Xecute
- Bash scripts end with .sh and require the execution bit to run
- Crontab is a scheduler that will run periodically

Questions?