Cloud Infrastructure and Management

Prof Rajiv Misra IIT Patna

Date: 06-04-2022

TATA CONSULTANCY SERVICES

tcs iON

Contents of Lecture

- What is a Data Center?
- Servers
- Softwares
- Networking devices
- Storage solutions
- Managing the infrastructure

Data Centers

Cloud Data Center :

On-Premises Data Center

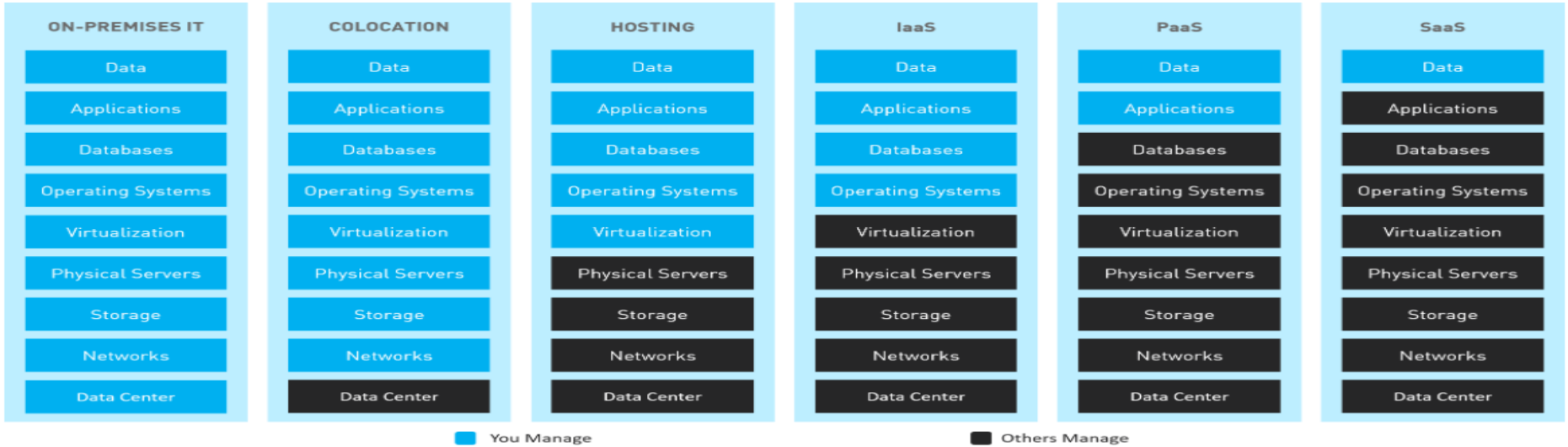
- An on-prem data center simply means that the organization maintains all of the IT infrastructure needed by the business on-site.
- An on-prem data center includes everything from the servers that support web and email to the networking hardware connecting them to support infrastructure equipment like uninterruptible power supplies (UPS).

Cloud Data Center?

- A cloud data center moves a traditional on-prem data center off-site. Instead of personally managing their own infrastructure, an organization leases infrastructure managed by a third-party partner and accesses data center resources over the Internet.
- Under this model, the cloud service provider is responsible for maintenance, updates, and meeting service level agreements (SLAs) for the parts of the infrastructure stack under their direct control.

Cloud Data Center: Hybrid Cloud:Shared Responsibility Model

- The migration from an on-premises data center to a cloud data center doesn't mean moving everything to the cloud. Many companies have hybrid cloud data centers which have a mix of on-premises data center components and virtual data centers components. In the figure below we see how as-a-service models are shifting ownership of data center and infrastructure components from a fully owned and operated on-premises facility towards a commodity service model. Depending on the model selected, an organization may be responsible for maintaining and securing more or less of their infrastructure stack. The breakdown of responsibilities is outlined by the cloud services provider in shared responsibilities models



On-Premises Data Centers vs Cloud Data Centers

- Nearly all organizations now have at least some of their infrastructure hosted in the cloud. The reason for this is that cloud data centers offer a number of advantages over maintaining an on-prem data center. Some of the pros and cons of cloud-based vs. on-premises data centers include:
- **Scalability:** In an on-premises data center, resource scalability is limited by the infrastructure that the company has purchased and deployed. In the cloud, additional resources can be quickly and easily spun up as needed.
- **Flexibility:** In an on-premises data center, resource flexibility is limited by the need to acquire, provision, or update appliances. In the cloud, a customer can spin up or take down resources quickly to meet business needs.
- **Cost:** Maintaining an on-prem data center is more expensive than a cloud-based one. On-prem, an organization pays full price for all of their infrastructure. In the cloud, resources can be shared, and cloud service providers can take advantage of economies of scale.
- **Availability:** In an on-premises data center, an organization has complete control over their infrastructure, which can be good or bad. In the cloud, availability is protected by service level agreements, which may provide better guarantees than an organization can in-house.
- **Security:** In the cloud, the cloud service provider is responsible for securing part of an organization's infrastructure stack and is likely more practiced at doing so. However, some customers may want additional security of their cloud-based data centers that are not natively provided by the cloud service provider.
- **Accessibility:** In an on-prem data center, the organization has complete control over the systems that it deploys and uses. In the cloud, the organization is limited to what is offered by the service provider.

Cloud Servers vs Dedicated Servers

- Cloud servers can be configured to provide levels of performance, security and control similar to those of a dedicated server. But instead of being hosted on physical hardware, they reside in a shared “virtualized” environment that’s managed by your cloud hosting provider.
- You benefit from the economies of scale of sharing hardware with other customers. And, you only pay for the exact amount of server space used. Cloud servers also allow you to scale resources up or down, depending on demand, so that you're not paying for idle infrastructure costs when demand is low.
- With cloud servers, you can optimize IT performance without the huge costs associated with purchasing and managing fully dedicated infrastructure. Businesses with variable demands and workloads often find that cloud servers are an ideal fit.
- A dedicated server is a physical server that is purchased or rented entirely for your own business needs. Dedicated servers are typically used by large businesses and organizations that require exceptionally high levels of data security, or organizations that have steady, high demands for server capacity.
- With dedicated servers, businesses still need the IT capacity and expertise to manage ongoing maintenance, patches and upgrades. Businesses using I/O-heavy applications, such as databases and big data platforms, find significant value in bare metal dedicated hardware.

open source software for cloud computing

- The question used to be, “What container orchestration platform are you using?” Now the question is, “So, how are you running Kubernetes?” The past year has seen Kubernetes continue its domination, with managed “K8s” clusters being offered by all three major cloud providers, and lots of innovation happening in the surrounding ecosystem. Our 2018 Bossie winners in cloud computing are ushering in the new era of cloud-native applications.

Kubernetes

- There was a time when other options were considered for container orchestration. But when it comes to running distributed containerized applications today, Kubernetes has consolidated its dominance. If you're going to deploy a new scalable service across AWS, Azure, GCP, or your private cloud, Kubernetes is likely going to enter the conversation.
- Building any container infrastructure means supporting a whole ecosystem of hardware, software, and network devices. With the Kubernetes 1.11 release, Kubernetes now supports IPVS in-cluster load balancing and CoreDNS (a pluggable DNS server). This follows major security, storage, and networking enhancements earlier in the year.
- Istio

Docker

- Docker allows you to package your software in “containers” and run them as an OS-level virtual machine. As opposed to VMware and other popular virtualization technologies, Docker doesn’t waste CPU or other resources virtualizing a whole computer, or require an additional OS for each “guest.” In just five short years Docker has totally changed the way virtualization works in computing.
- With the ascension of Kubernetes, Docker is no longer the way most people are looking to distribute containers. However, Docker is certainly still a viable alternative for managing container clusters, and it continues to be the way most people create and run individual containers.

Cloud Storage

- when we select a cloud storage service, we should consider some essential features such as service security, privacy, mobile app, pricing, complexity, and speed.
- Some of the most popular cloud storage providers are Apple (iCloud), **Amazon (Amazon Web Services), Dropbox, and Google**
- **How does cloud storage work?**
- Cloud storage works on a client-server model, in which a client sends the request to the subscribed cloud service storage and the server at the data center gives the appropriate response.
- The main objective of the cloud, instead of saving data at local storage, the data of the user can be collected at some data center so the user could retrieve his or her data from any device.
- What if the data center of the cloud service provider collapses or gets destroyed, would the user data also be destroyed? The answer is no.
- The cloud storage depends on hundreds of data centers, so even if one of the data centers collapses, there are hundreds of centers that would help you to retrieve and save your data.

What is a Data Center?

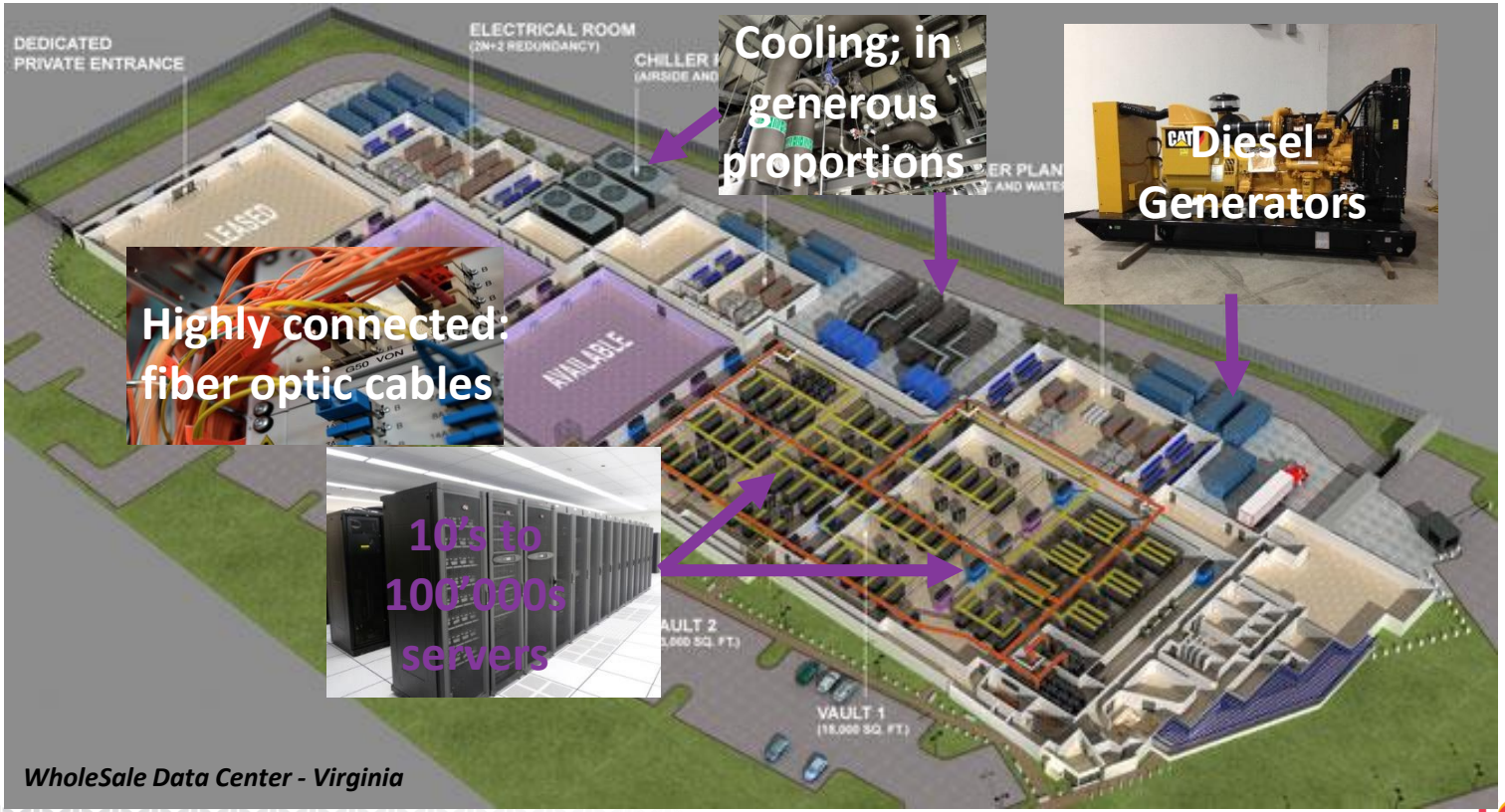
"A facility used to house computer systems and associated components." (Wikipedia)

"It is the brain of a company and the place where the most critical processes are run." (SAP)

"It is a factory that transforms and stores bits."
(Albert Greenberg - Microsoft)

Collection of physical compute, storage, and network resources.

The Data Center in a nutshell



Data Centers Challenges

Large-scale

- 10’s to 100’s of thousands of servers
- high bandwidth and low latency is critical

Availability and high performance

- 99.99X% availability
- redundancy of all critical components

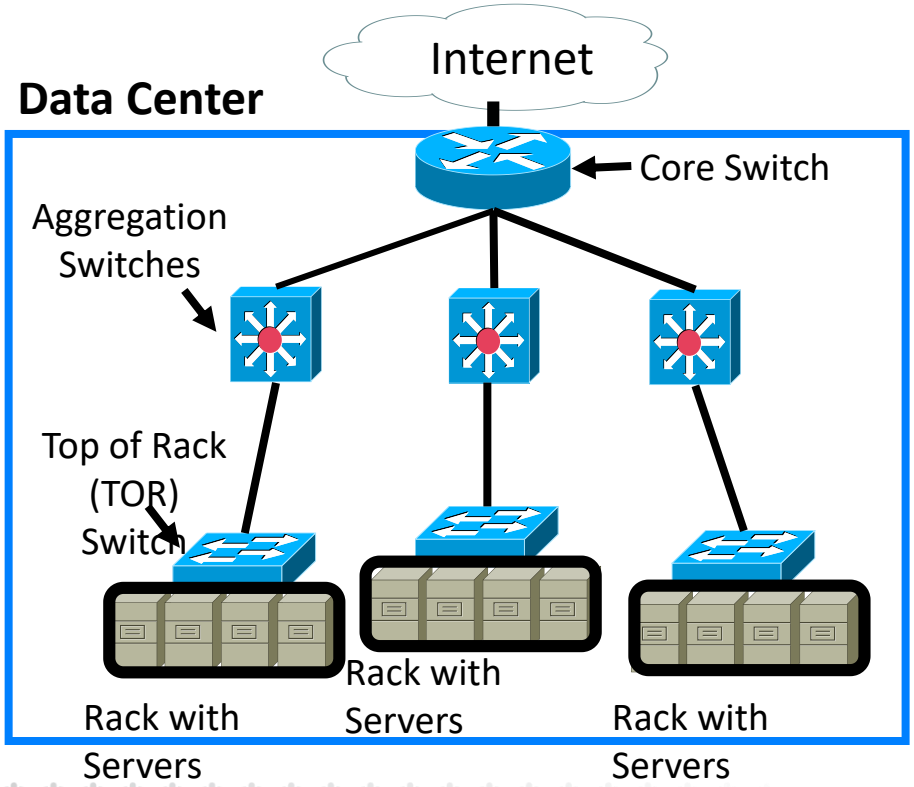
Security and performance isolation

- controlled access to infrastructure
- secure the data

Complexity

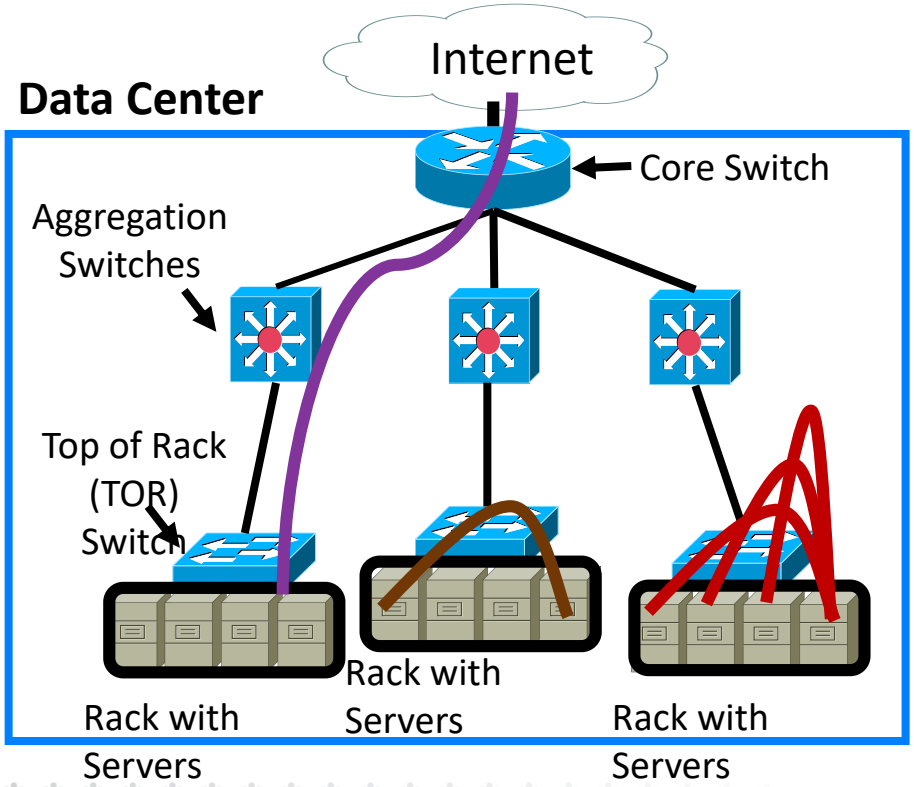
- plethora of components
- plethora of software and hardware failures

Anatomy of a Data Center



Typical physical network topology is a tree (3-tier Architecture)

Anatomy of a Data Center



Common traffic patterns

North-south

- common with web sites

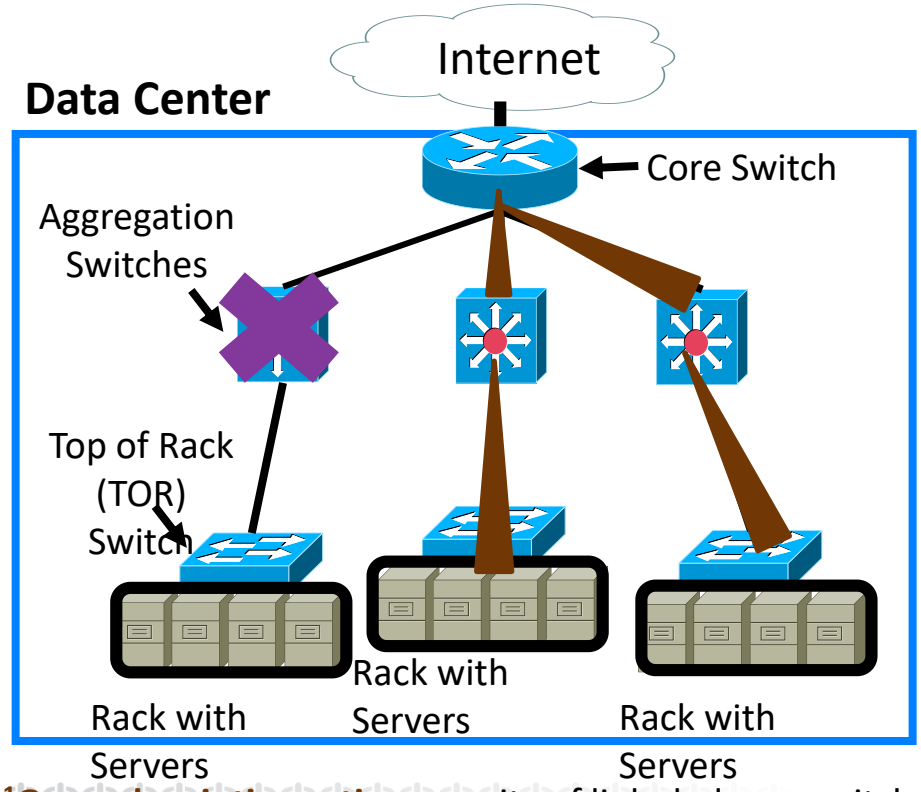
East-West

- common with back-end of web sites/ web services
- common in Big Data

Many-to-one

- causes TCP incast

Anatomy of a Data Center



What are the challenges with this design?

Single point of failure

- redundancy increase costs

Links higher in topology are oversubscribed[1]

- cannot handle all servers sending at maximum rate
- design tradeoff to scale

[1] **Oversubscription ratio:** capacity of links below a switch relative to capacity of links above

Anatomy of a Data Center

Can we achieve full bisection bandwidth?

(oversubscription ratio of 1:1)

Partially, because …

- requires enterprise-level switches

- even they become saturated at large scale

Can we do better?

Emerging data center topology – Fat Trees

Goals:

- use cheap **identical commodity switches**

- provides **redundancy**

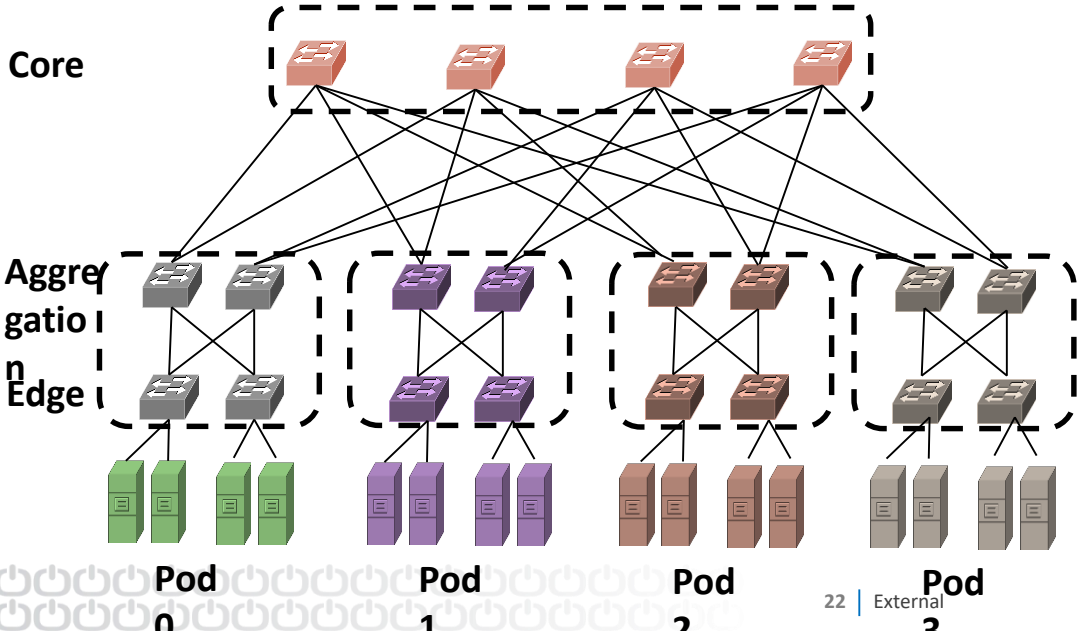
- help address large volumes of data

- provides **full bisection bandwidth**
  the bottleneck is the network interface,
  not the link in the network

Emerging data center topology – Fat Trees

Main idea: **inter-connect racks using a fat-tree topology**

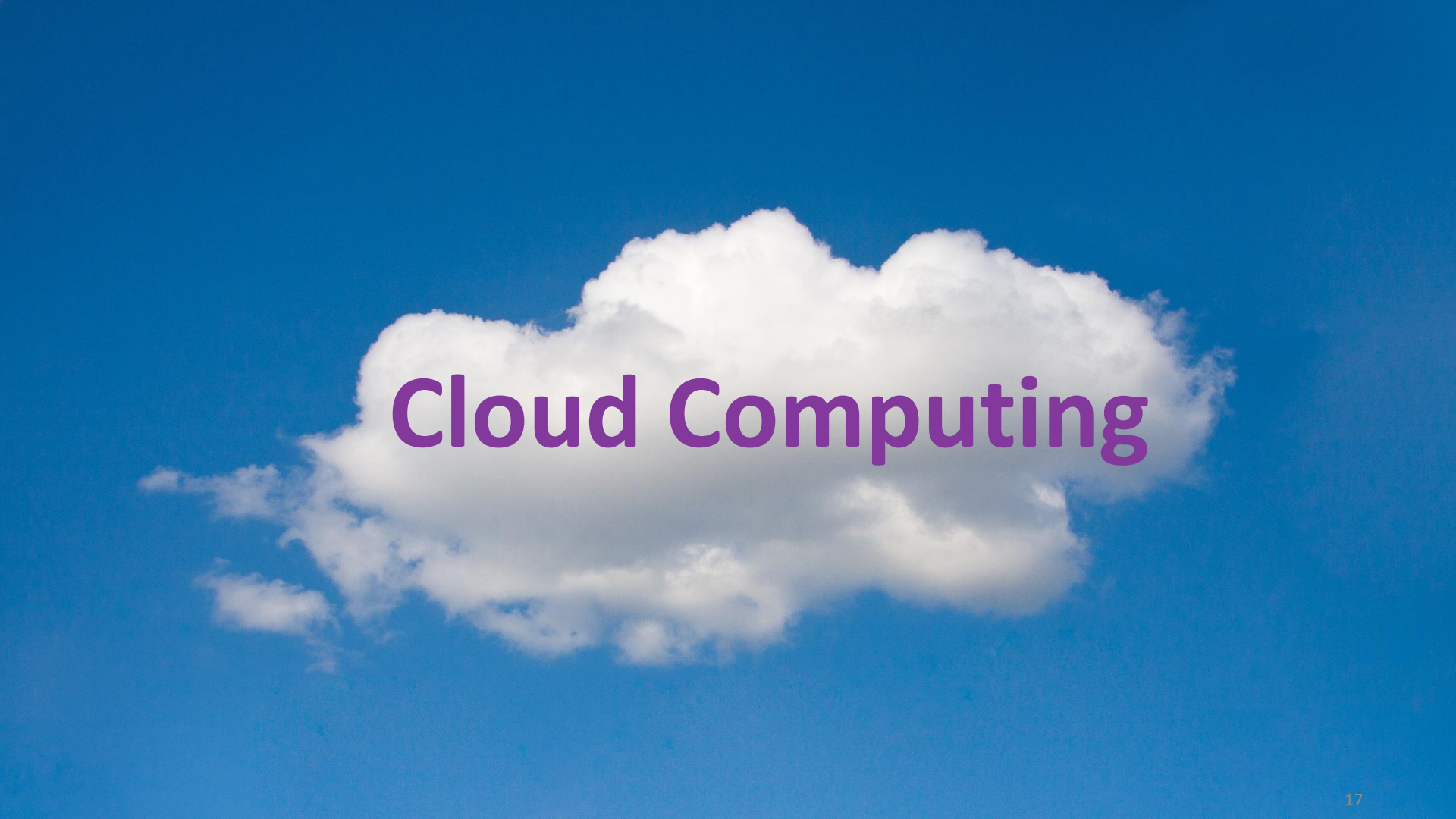
E.g.: given K-ports identical switches, where K = 4:



Challenges

- More switches than the tree topology
- Different routing approach
- Does not solve TCP incast

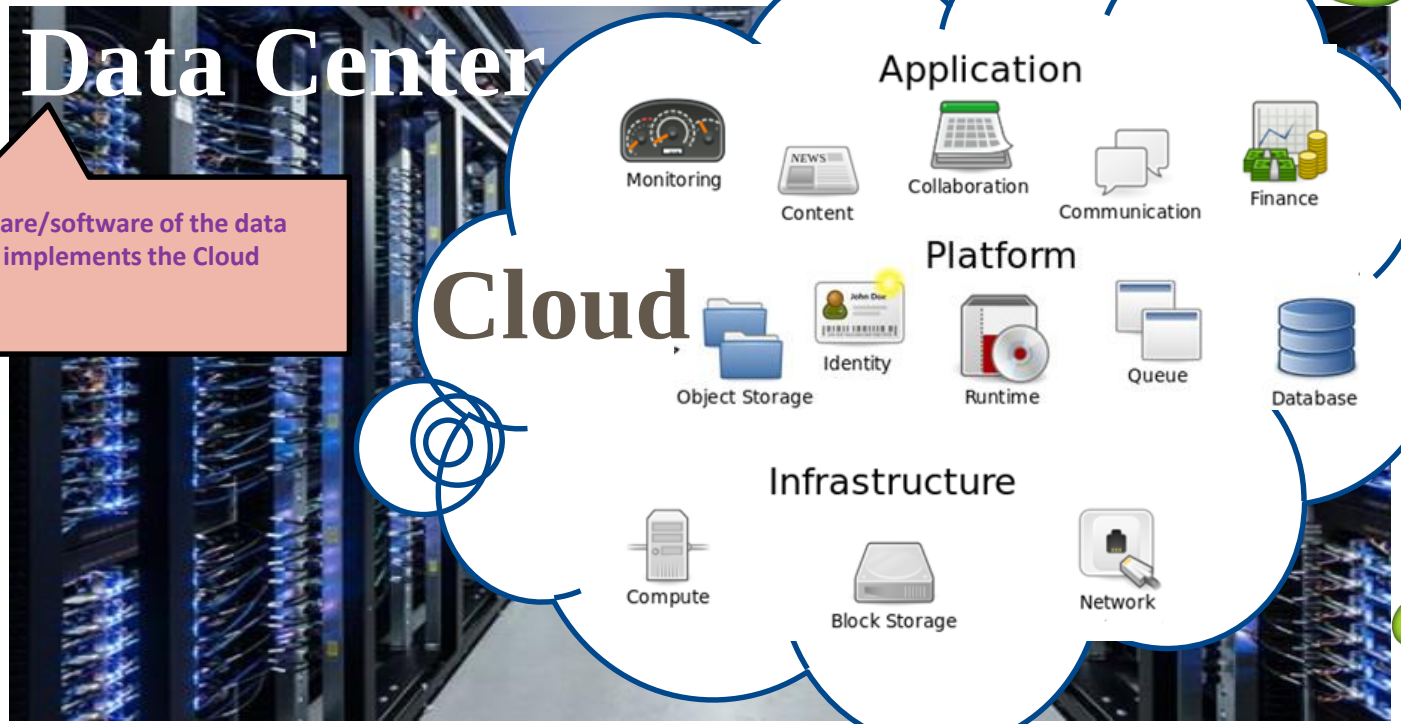
How do we make use of Data Centers?

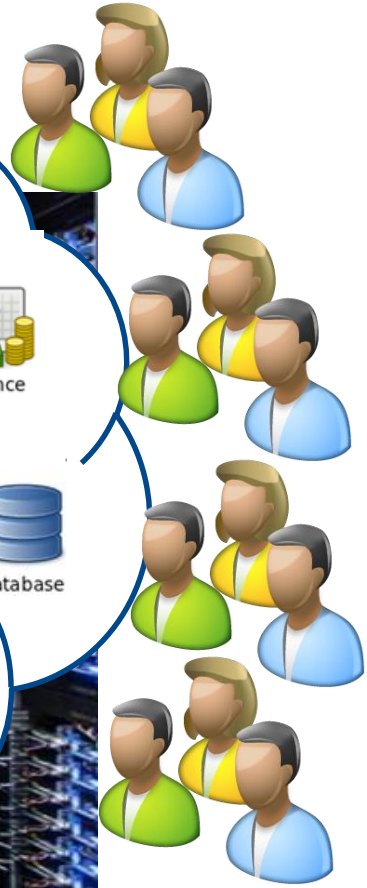

Cloud Computing

What is Cloud Computing?

Delivery of on-demand shared computing resources – everything from applications, services to compute, storage and network resources.

What is Cloud Computing?





Cloud Computing – Key Characteristics

Virtualized Resources

- physical resources divided into pieces
- each customer gets an isolated piece

Rapid elasticity

- capabilities can be elastically provisioned/released

Pay-per-use

- only pay for the resources you use

On-demand

- customers can request/release resources whenever they want

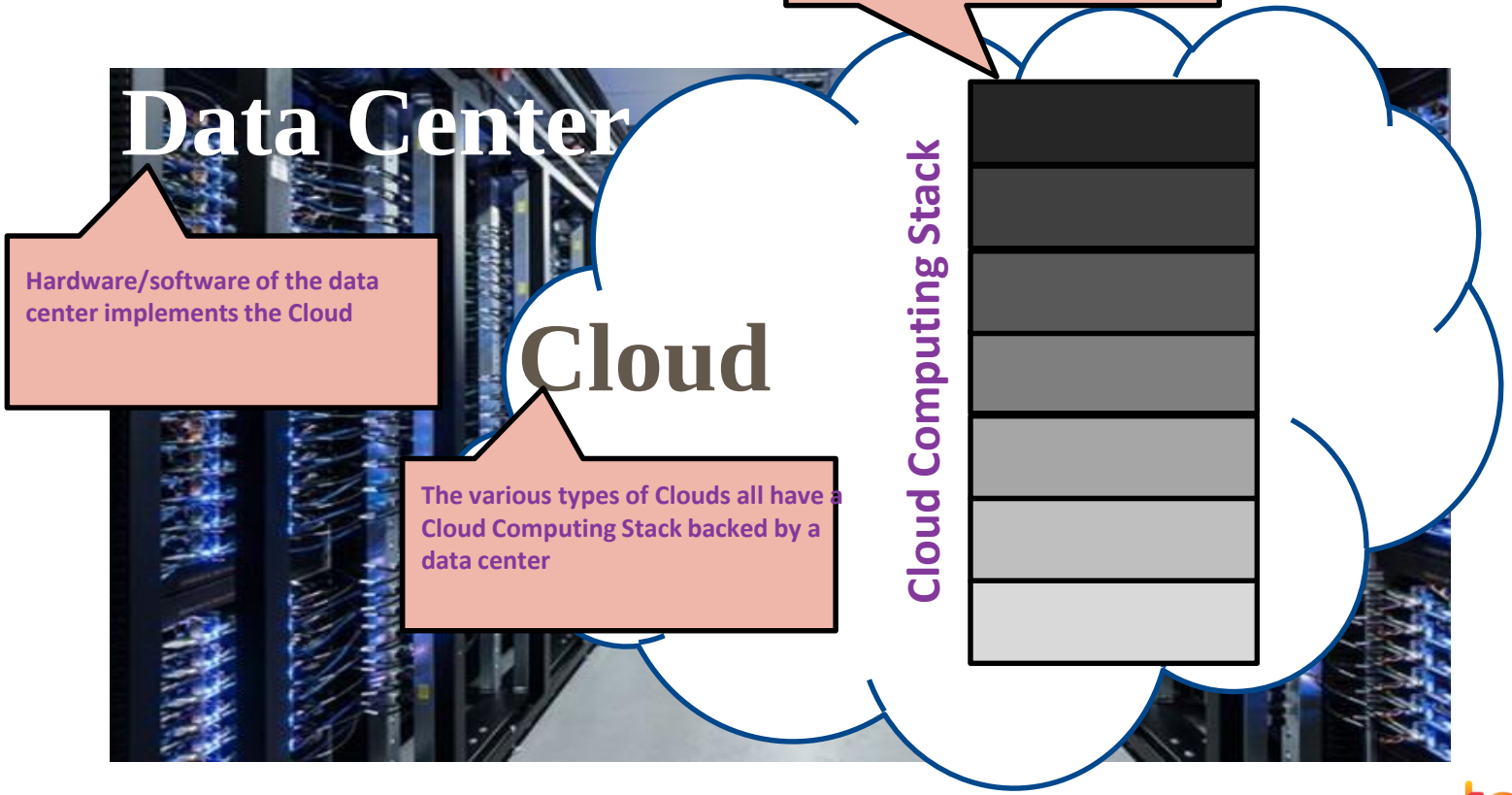
Resilient

- multiple pools of resources that are unlikely to fail simultaneously

Shared

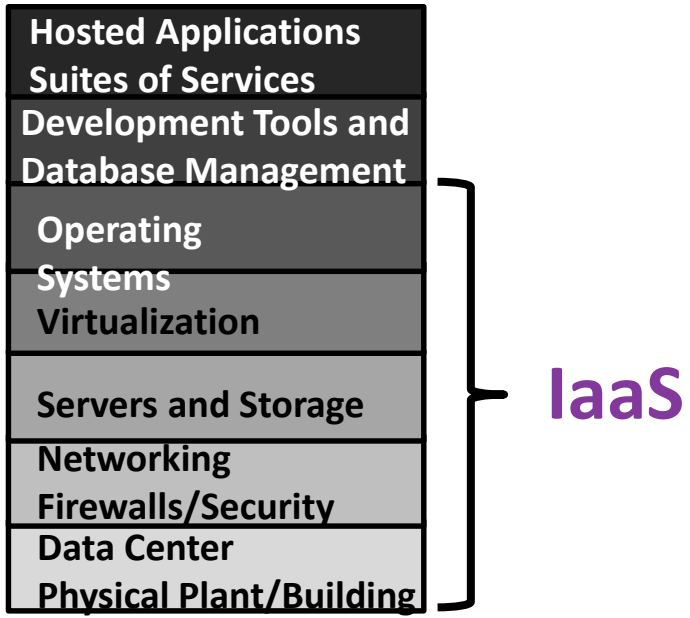
- multiple customers share the same physical resources

Cloud Computing



Cloud Computing – Categories

IaaS



Cloud Computing Stack

- customers lease virtual machines, virtual storage, virtual networks

- customers must manage operating system, file system, etc..

rackspace HOSTING

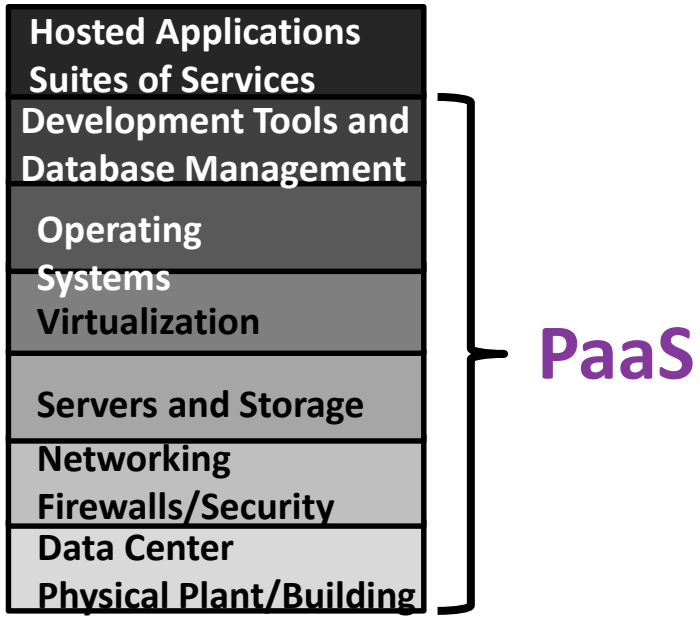
amazon web services | EC2

Google CLOUD STORAGE

tcs iON

Cloud Computing – Categories

PaaS



Cloud Computing Stack

- customers lease resources to run applications written in a specific language such as Python, Java, MapReduce

- cloud provider manages the operating system, file system, and network

Windows Azure

Google app engine

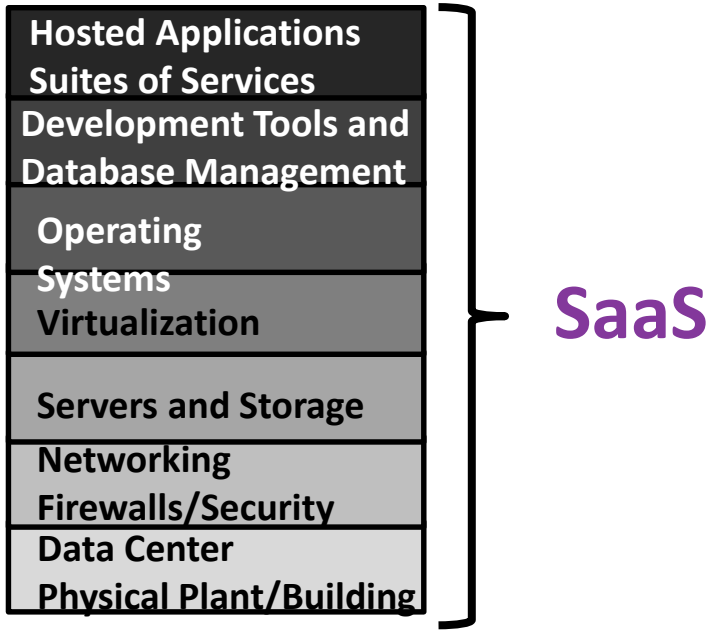
Microsoft SQL Azure

force.com platform as a service

tcs iON

Cloud Computing – Categories

SaaS



Cloud Computing Stack

- customers lease machines that run specific software

- it is what most people mean when they say the “Cloud”

Google Chrome OS

Google docs

Office 365

LinkedIn

tcs iON

Cloud Computing – Deployment Models

Private Cloud

- only available to users (e.g. departments) within a company or organization

Public Cloud

- anyone can request and use the cloud

Hybrid Cloud

- a composition of public and private cloud resources
- bounded by standardized or proprietary technology

Server Virtualization

Cloud Computing depends on Server Virtualization

- **In the context of server virtualization, how do we network the large number of VMs that might reside on a single physical server?** Cloud computing depends heavily on server virtualization for several reasons:

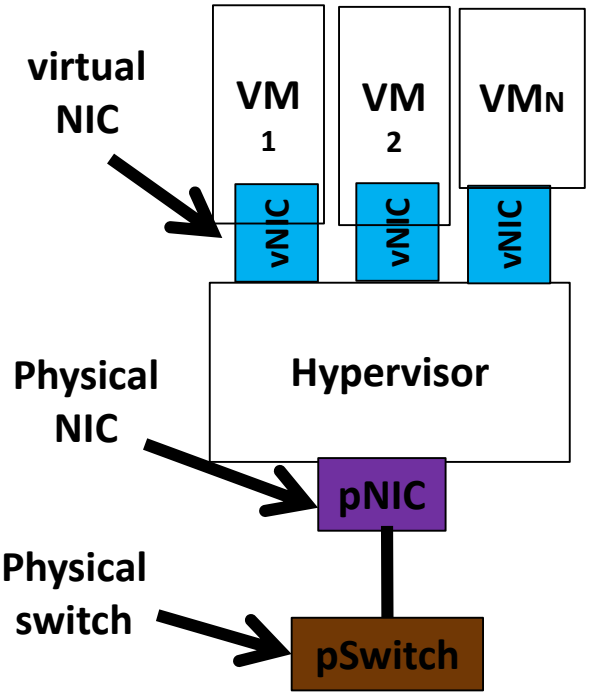
(i) Sharing of physical infrastructure: Virtual machines allow multiplexing of hardware with tens to 100s of VMs residing on the same physical server. Also, it allows rapid deployment of new services.

(ii) Spinning up a virtual machine in seconds: Spinning up a virtual machine might only need seconds compared to deploying an app on physical hardware, which can take much longer.

(ii) Live VM migration: Further, if a workload requires migration. For example, you do physical server requiring maintenance. This can be done quickly with virtual machines, which can be migrated to other servers without requiring interruption of service in many instances. Due to these advantages today, more endpoints on the network are virtual rather than physical.

Server Virtualization

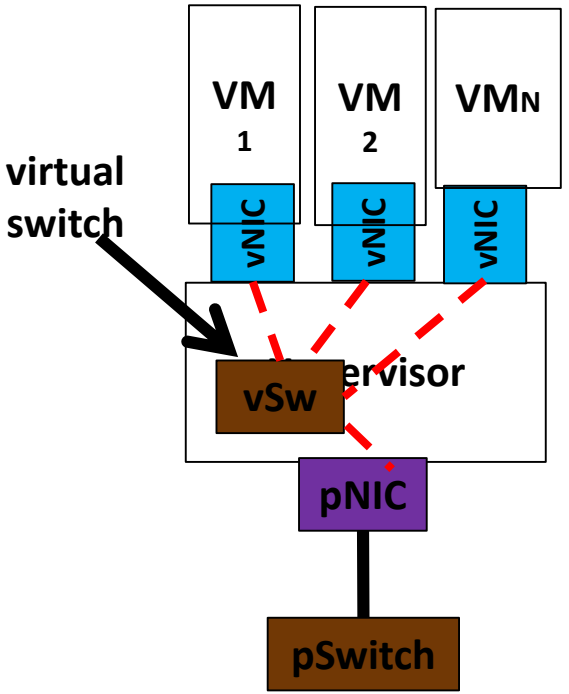
- The physical hardware is managed by a Hypervisor. This could be **Xen, KVM, or VMWare's ESXI,** or multiple such alternatives.
- On top of the Hypervisor runs several VMs in user space.
- The hypervisor provides an emulated view of the hardware to the VMs, which the VMs treat as their substrate to run a guest OS on.
- **Among other hardware resources the network interface card is also virtualized in this manner.** The hypervisor managing the physical NIC, is exposing virtual network interfaces to the VMs. **The physical NIC also connects the server to the rest of the network.**



Networking of VMs inside the Hypervisor

- **The hypervisor runs a virtual switch,** this can be a simple layer tool searching device, Operating in software inside the hypervisor.

- vSw is **connected to all the virtual NICs,** has them as the physical NIC, and moved packets between the VMs and the external network.



Alternate Methods of virtualization:

(i) Using Docker

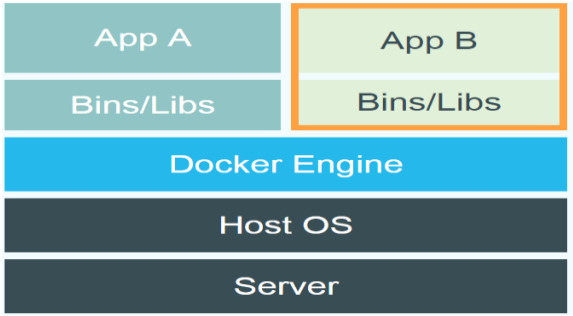
(ii) Using Linux containers

Docker

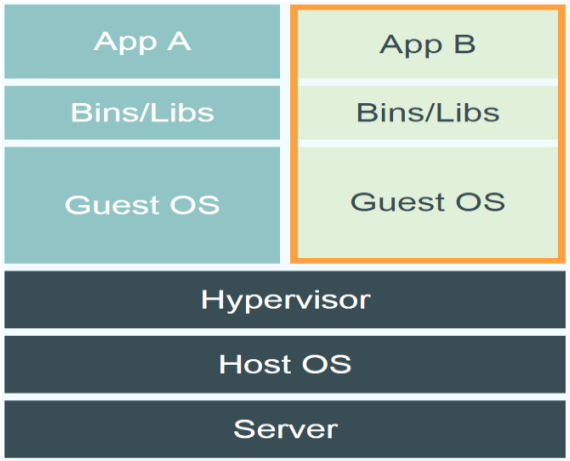
- **Docker is an open-source project that automates the deployment of applications inside software containers,** by providing an additional layer of abstraction and automation of OS–level virtualization on Linux.

- **The Docker Engine container comprises just the application and its dependencies.**

- **It runs as an isolated process in userspace on the host operating system, sharing the kernel with other containers.** Thus, it enjoys the resource isolation and allocation benefits of VMs but is much more portable and efficient.



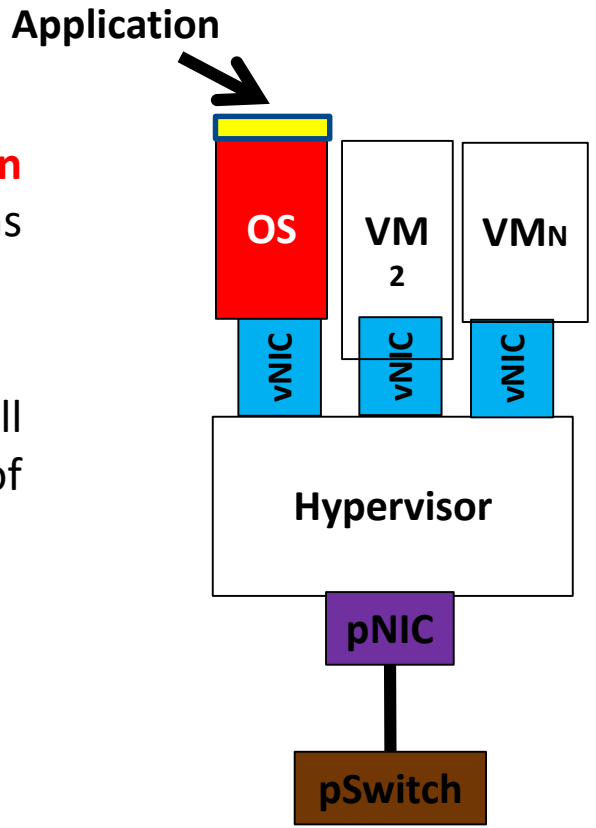
Docker



Virtual Machine

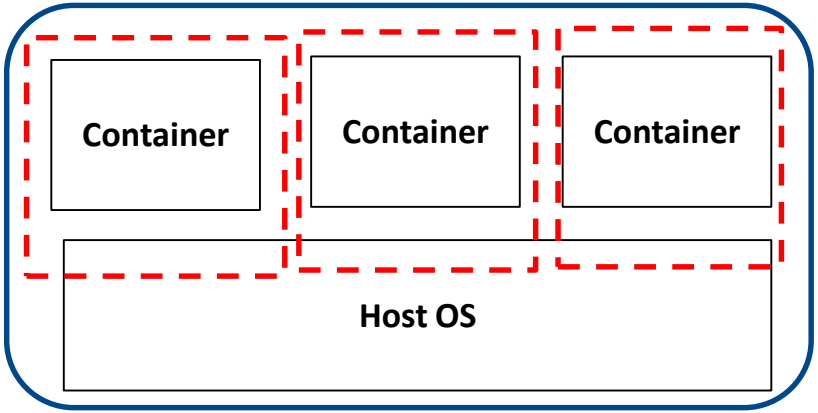
(i) Using VMs as Virtualization

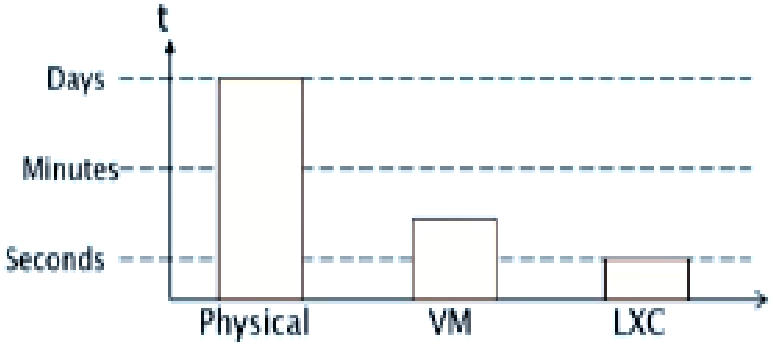
- In this model, **each VM runs its own entire guest OS.** The application runs as a process inside this guest OS.
- This means that even running a small application requires the overhead of running an entire guest OS.



Linux containers as Virtualization

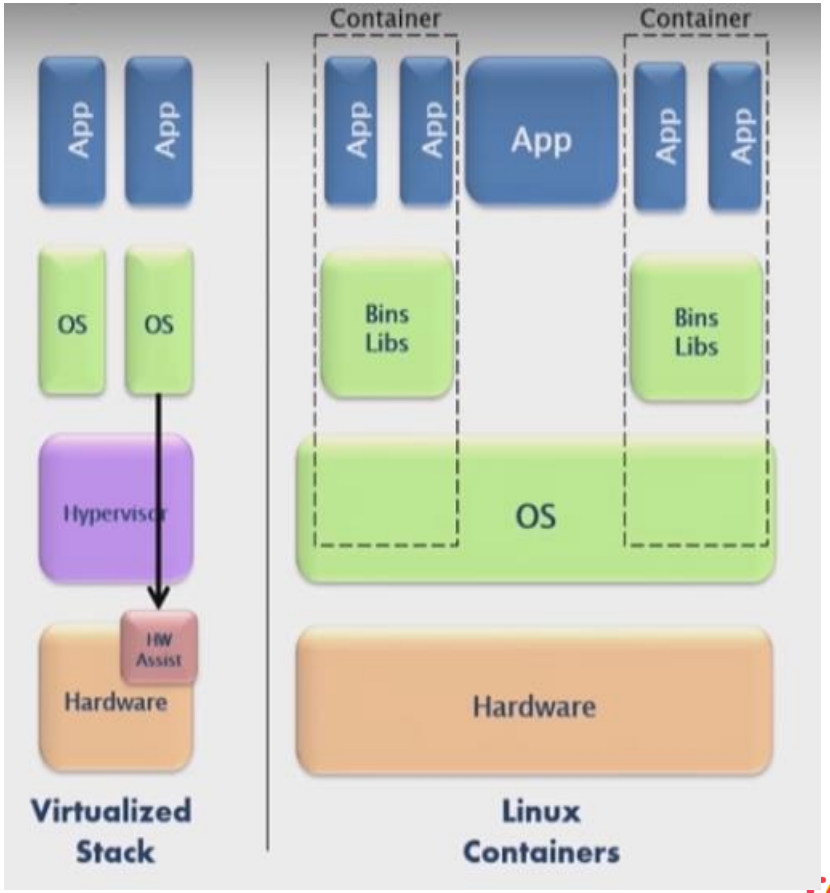
- Create an environment as close as possible to a standard Linux without separate kernel
- **Uses kernel features for separation**
- **Near bare metal (physical hardware) performance**
- **Fast provisioning times (Building up/start-up time)**





Linux containers

- Containers run in host systems kernel
- Separated with policies
- Can use apps in host system
- Can install additional libraries and apps
- Portable between OS variants supporting linux container
- No overhead with hypervisor and guest OS kernel

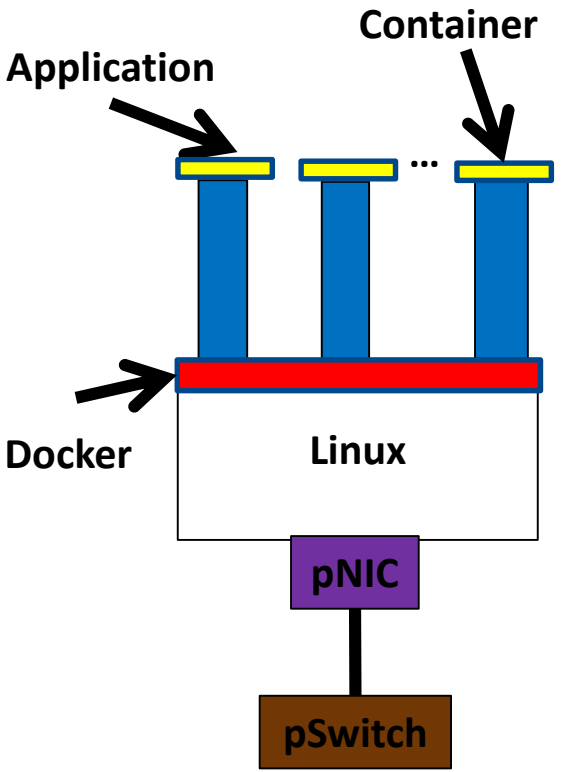


(ii) Using Linux containers

In this approach, an application together with its dependencies uses packages into a Linux container, which runs using the host machine's Linux stack and any shared resources.

Docker is simply a container manager for multiple such containers. Applications are isolated form each other by the use of separate namespaces. Resources in one application cannot be addressed by other applications. This yields isolation quite similar to VMs, but with a smaller footprint.

Further, containers can be brought up much faster than VMs. Hundreds of milliseconds as opposed to seconds or even tens of seconds with VMs.



Kubernetes: Definition

Kubernetes (K8S) is an open source software tool for managing containerized workloads.

- It operates at the container (not hardware) level to automate the deployment, scaling and management of applications.

- K8s works alongside a containerisation tool, like Docker. So if containers are the ‘ingredients’ of an application, then K8S would be the ‘chef’.

- As well as managing individual containers, K8s can also manage clusters.

  - A cluster is a series of servers connected to run containers.
  - K8s can scale up to 5,000 servers and 150,000 pods in a single cluster.
  - A pod is a group of containers that share resources, a network and can communicate with one another.

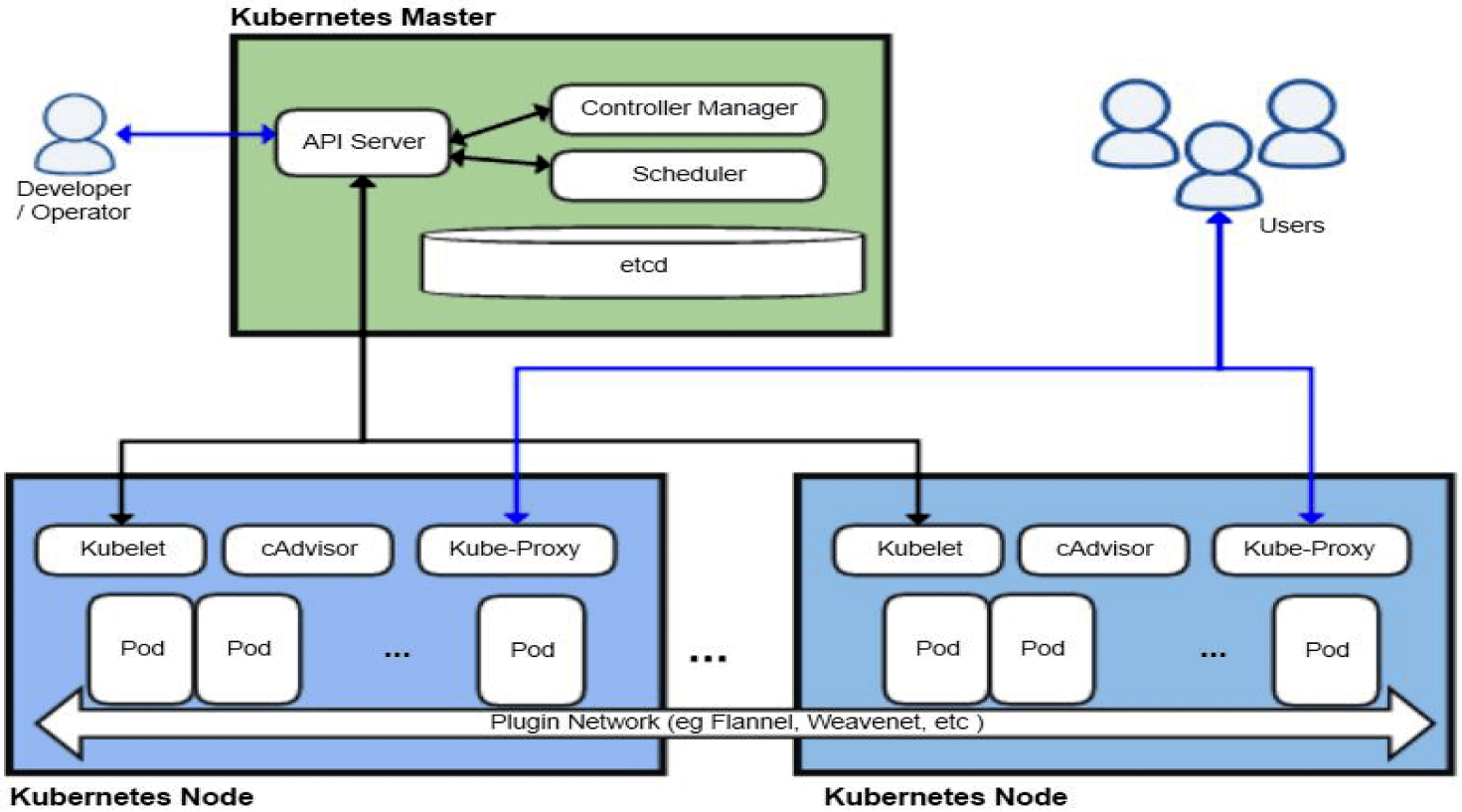
How Does Kubernetes Work?

Kubernetes has a master-slave architecture:

1. Worker Nodes (slave) - This is where containers are deployed. These nodes contain:
a. Multiple pods
b. Docker engine
c. Any add-ons e.g. DNS
d. Kubelet - this is an important component as it carries out the instructions from the master node.

2. Master Node (master) - This controls the deployment. This node contains:
a. API server - this receives inputs from the User Interface (UI) or Command Line Interface (CLI).
b. Controller - uses information from the API to drive the application from its current state towards the desired state provided.
c. Scheduler
d. Etcd - handles configuration management, service discovery etc.

Kubernetes Architecture



Kubernetes Master Node

Master Node is a collection of components like Storage, Controller, Scheduler, API-server that makes up the control plan of the Kubernetes. When you interact with Kubernetes by using CLI you are communicating with the Kubernetes cluster's master node. All the processes run on a single node in the cluster, and this node is also referred to as the master.

Master Node Components:
1) Kube API-server performs all the administrative tasks on the master node. A user sends the rest commands as YAML/JSON format to the API server, then it processes and executes them. The Kube API-server is the front end of the Kubernetes control plane.

2) etcd is a distributed key-value store that is used to store the cluster state. Kubernetes stores the file in a database called the etcd. Besides storing the cluster state, etcd is also used to store the configuration details such as the subnets and the config maps.

3) Kube-scheduler is used to schedule the work to different worker nodes. It also manages the new requests coming from the API Server and assigns them to healthy nodes.

4) Kube Controller Manager task is to obtain the desired state from the API Server. If the desired state does not meet the current state of the object, then the corrective steps are taken by the control loop to bring the current state the same as the desired state.

Kubernetes Worker Node

The worker nodes in a cluster are the machines or physical servers that run your applications. The Kubernetes master controls each node. there are multiple nodes connected to the master node. On the node, there are multiple pods running and there are multiple containers running in pods.

Worker Node Components

1) Kubelet is an agent that runs on each worker node and communicates with the master node. It also makes sure that the containers which are part of the pods are always healthy. It watches for tasks sent from the API Server, executes the task like deploy or destroy the container, and then reports back to the Master.

2) Kube-proxy is used to communicate between the multiple worker nodes. It maintains network rules on nodes and also make sure there are necessary rules define on the worker node so the container can communicate to each in different nodes.

3) Kubernetes pod is a group of one or more containers that are deployed together on the same host. Pod is deployed with a shared storage/network, and a specification for how to run the containers. Containers can easily communicate with other containers in the same pod as though they were on the same machine.

4) Container Runtime is the software that is responsible for running containers. Kubernetes supports several container runtimes: Docker, containers.

Benefits of Kubernetes

Self-healing

- Clusters can auto-restore from errors by rolling back to the last working version of software. This allows teams to ship quickly without the risk of breaking anything.

- High Availability
  - Clusters can be recreated on a working node to avoid downtime during server failure.
- Simplifys Maintenance
  - If a server needs to be rebooted, or the host OS needs updating, containers can be moved to another node whilst maintenance is carried out.
- Automatic Scaling
  - Uses information from user requests and CPU usage to increase or decrease the number of nodes running to match demand.
- Efficient
  - Automatically spins up any new containers on under-utilized nodes.

Things Kubernetes does not do...

There are sometimes misconceptions about what Kubernetes can do. Kubernetes does not…

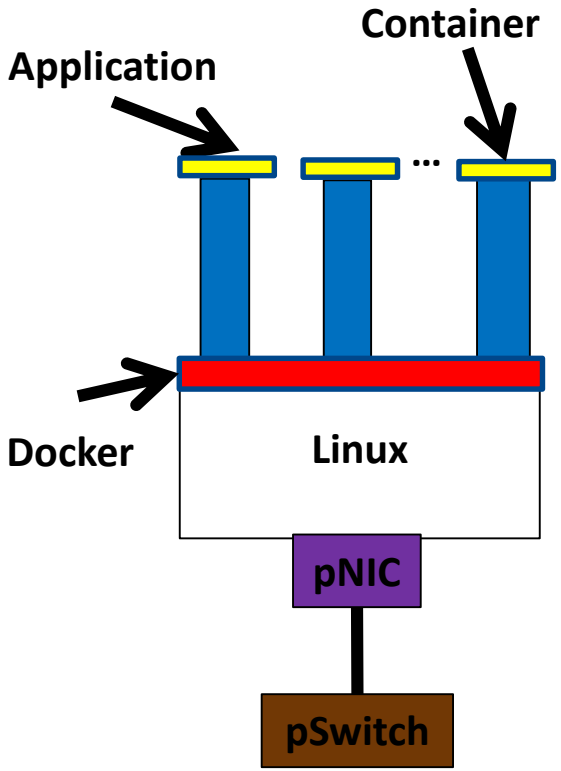
- Provide any comprehensive machine configuration.
- Provide a configuration language/system
- Dictate logging, monitoring or alerting solution
- Build your application
- Provide middleware, data-processing frameworks, databases, caches etc. BUT these components can run on Kubernetes

Networking with Docker

- **Each container is assigned a virtual interface.** Docker contains a virtual ethernet bridge connecting these multiple virtual interfaces and the physical NIC.

- **Configuring Docker and the environment variables decide what connectivity is provided.** Which machines can talk to each other, which machines can talk to the external network, and so on.

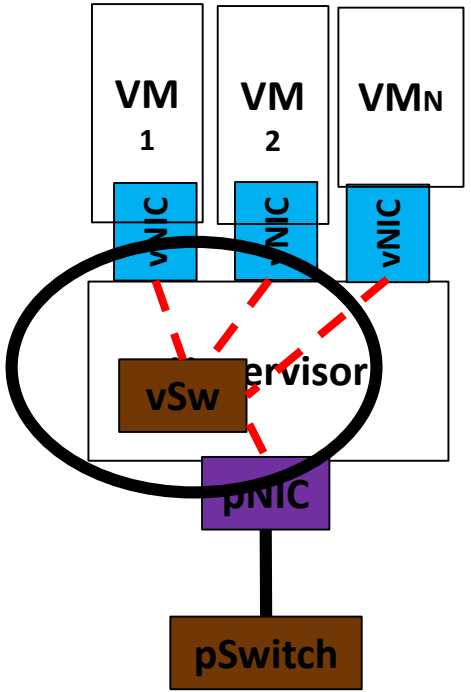
- **External network connectivity is provided through a NAT, that is a network address translator.**



Improving networking performance

- The **hypervisor runs a virtual switch to able to network the VM's** and **CPU is doing the work for moving the packets.**

CPU does the work!



Packet processing on CPUs

Flexible slow, CPU-expensive: Now packet processing on CPUs can be quite flexible because it can have general purpose forwarding logic. You can have packet filters on arbitrary fields run multiple packet filters if necessary, etc. But if done naively, this can also be **very CPU-expensive and slow.**

Packet forwarding: The packet forwarding entails: At 10Gbps line rates with the smallest packets, that's 84 Bytes. We only have an interval of 67ns before the next packet comes in on which we need to make a forwarding decision. Note that ethernet frames are 64 bytes, but together with the preamble which the tells the receiver that a packet is coming and the necessary gap between packets. The envelope becomes 84 bytes. For context a CPU to memory access takes tens of nanoseconds. So, 67 nanoseconds is really quite small.

Packet processing on CPUs

Need time for Packet I/O: Moving packets from the NIC buffers to the OS buffers, which requires CPU interrupts. Until recently a single X86 core couldn't even saturate a ten gigabits per second link. And this is without any switching required. This was just moving packets from the NIC to the OS. After significant engineering effort, packet I/O is now doable at those line rates. However for a software switch we need more.

Userspace overheads: If any of the switching logic is in userspace, one incurs the overhead for switching between userspace and kernel space.

Packet classification: Further, for switching we need to match rules for forwarding packets to a forwarding table. All of this takes CPU time. Also keep in mind that forwarding packets is not the main goal for the CPU. The CPU is there to be doing useful computation.

Approaches for Networking of VMs

There are two different approaches to address the problem of networking VMs:

(i) One, using specialized hardware:

- SR-IOV, single-root I/O virtualization

(ii) Other using an all software approach:

- Open vSwitch

(i) Hardware based approach

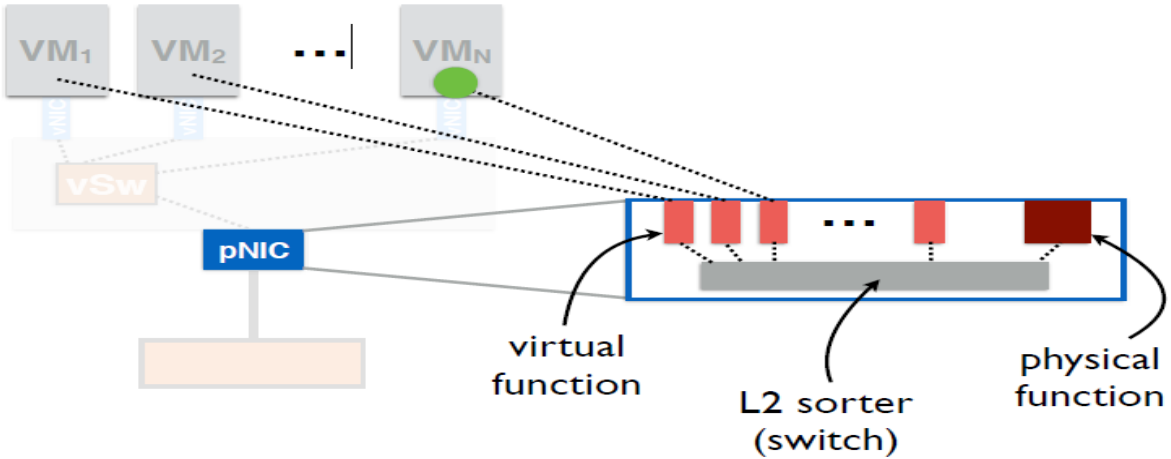
- **The main idea behind the hardware approach is that CPUs are not designed to forward packets, but the NIC is.**

- The naive solution would be to just give access to the VMs, to the NIC. But then problems arise. **How do you share the NIC's resources? How do you isolate various virtual machines?**

- **SR-IOV, single-root I/O virtualization**, based NIC provides one solution to this problem.

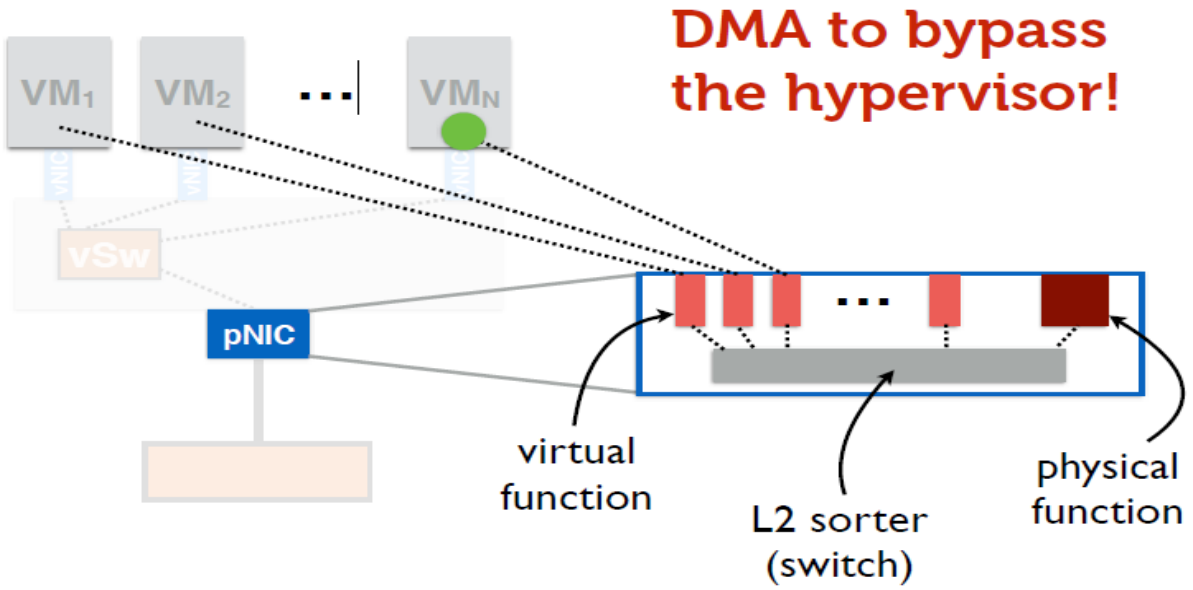
SR-IOV: Single-root I/O Virtualization

- **The SR-IOV, the physical link itself, supports virtualization in hardware.** So let's see inside this SR-IOV enabled network interface card.
- The NIC provides a physical function, which is just a standard ethernet port. In addition, it also provides several virtual functions which are simple queues that transmit and receive functionality.
- **Each VM is mapped to one of these virtual functions. So the VMs themselves get NIC hardware resources.**



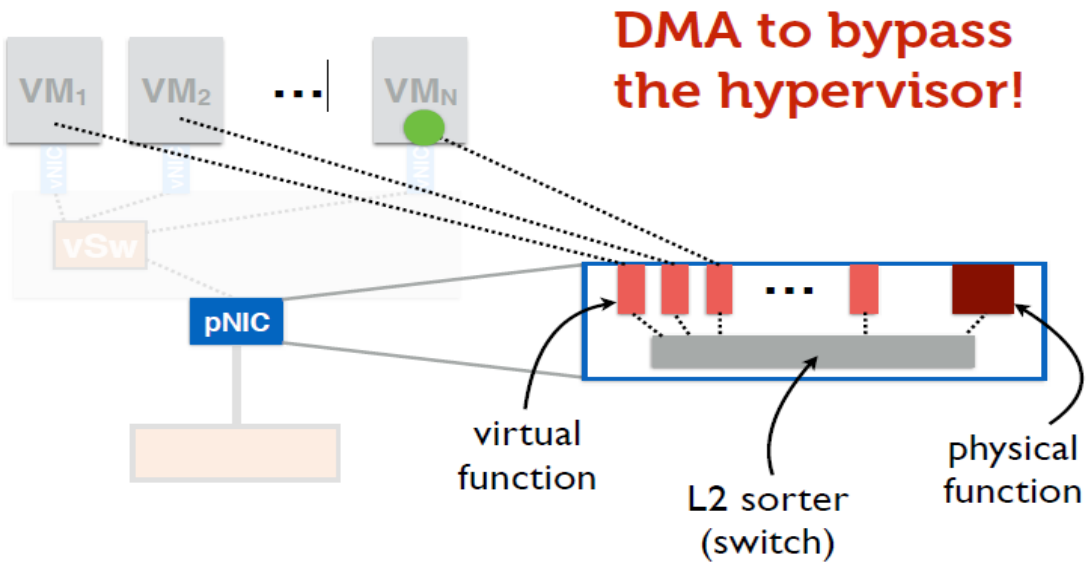
SR-IOV: Single-root I/O Virtualization

- **On the NIC there also resides a simple layer two, which classifies traffic into queues responding to these virtual functions**. Further, packets are moved directly from the net virtual function to the responding VM memory using **DMA (direct memory access).** This allows us to bypass the hypervisor entirely.



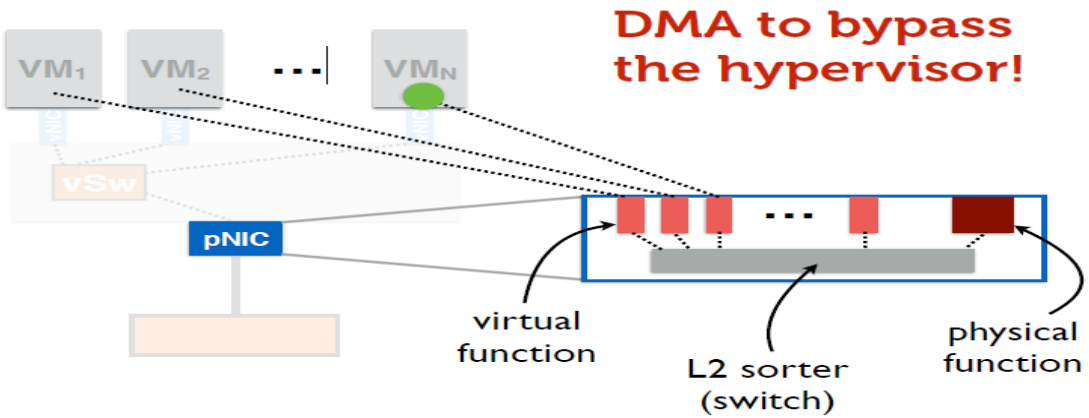
SR-IOV: Single-root I/O Virtualization

- **The hypervisor is only involved in the assignment of virtual functions to virtual machines.** And the management of the physical function, but not the data part for packets. The upshot to this is **higher through-put, lower latency and lower CPU utilization** and this give us close to native performance.



SR-IOV: Single-root I/O Virtualization

- There are downsides to this approach though. For one, **live VM migration becomes trickier**, because now you've tied the virtual machine to physical resources on that machine. The forwarding state for that virtual machine now resides in the layer two switch inside the NIC.
- **Second, forwarding is no longer as flexible.** We're relying on a layer two switch that is built into the hardware of the NIC. So we cannot have general purpose rules and we cannot be changing this logic very often. It's built into the hardware. **In contrast, software-defined networking (SDN) allows a much more flexible forwarding approach.**



(ii) Software based approach

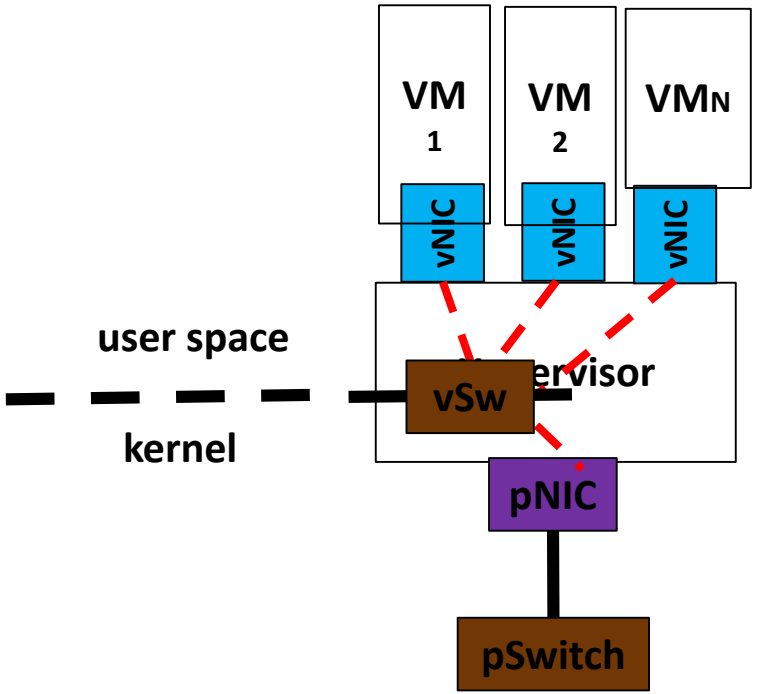
The software based approach that addresses:

(i) Much more flexible forwarding approach

(ii) Live VM migration

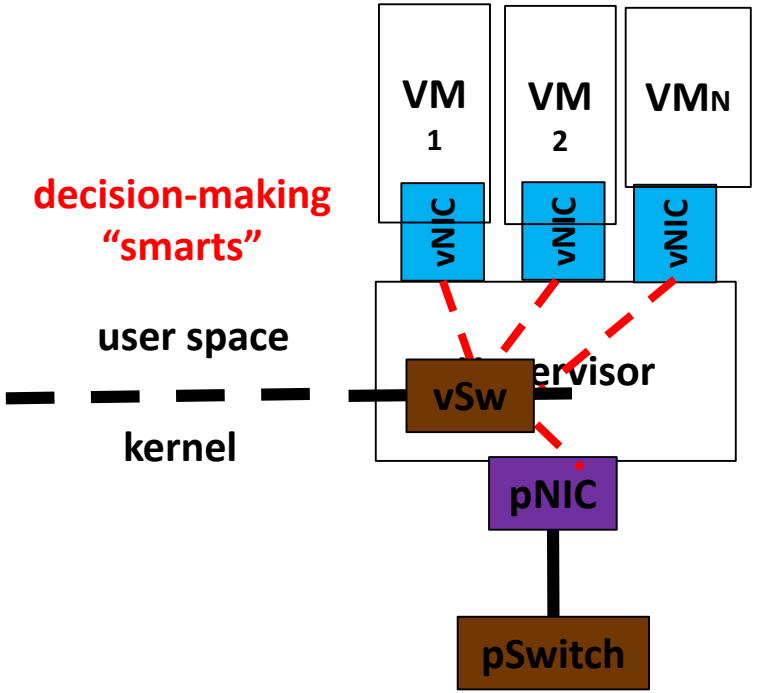
Open vSwitch

- Open vSwitch design goals are **flexible and fast-forwarding.**
- This necessitates a division between **user space** and **kernel space** task. One can not work entirely in the kernel, because of development difficulties. It's hard to push changes to kernel level code, and it's desirable to keep logic that resides in the kernel as simple as possible.



Open vSwitch

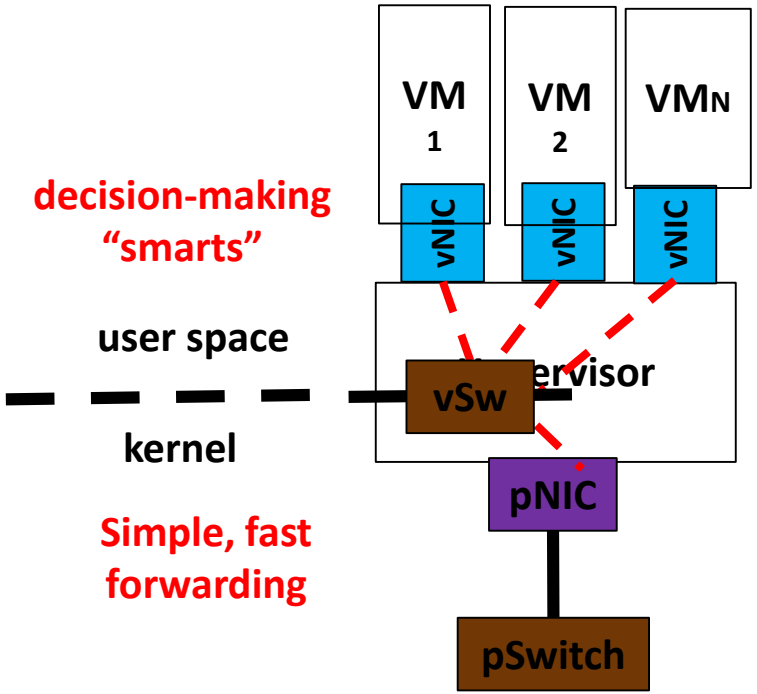
- So, the smarts of this approach, is the **switch routing decisions** lie in user space.
- This is where one **decides what rules or filters apply to packets of a certain type.** Perhaps based on network updates from other, possibly virtual, such as in the network. This behavior can also be programmed using **open flow.** **So, this part is optimized for processing network updates, and not necessarily for wire speed packet forwarding.**



Open vSwitch

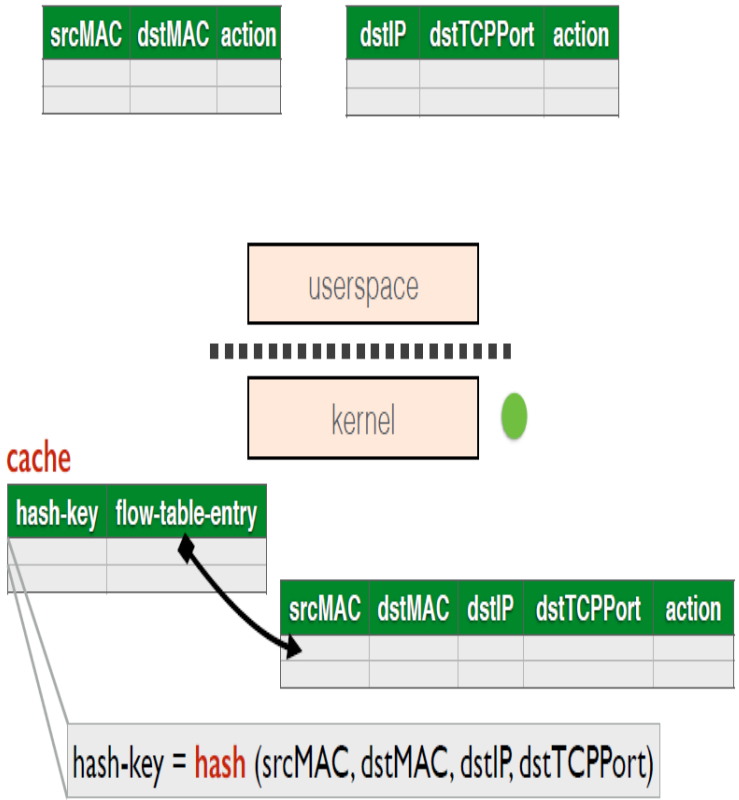
- **Packet forwarding**, on the other hand, is handled largely in the kernel, broadly.

- **Open vSwitch approach is to optimize the common case, as opposed to the worst case line rate requirements and caching will be the answer to that need.**



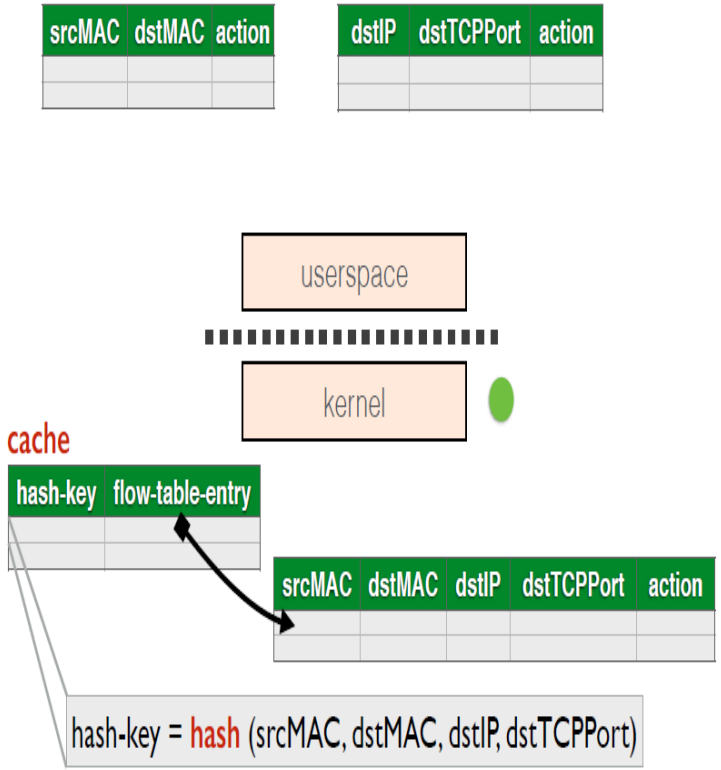
Inside Open vSwitch

- **The first packet of a flow goes to userspace here several different packet classifiers may be consulted.** Some actions may be based on MAC addresses and some others might depend on TCP PORTs, etc.
- The **highest priority matching action** across these different classifiers will be used to forward the packet.
- **Once a packet is forwarded, a collapsed rule used to forward that packet is installed in the kernel. This is a simple classifier with no priorities.** The following packets of this flow will never enter user space, seeing only the kernel level classifier.



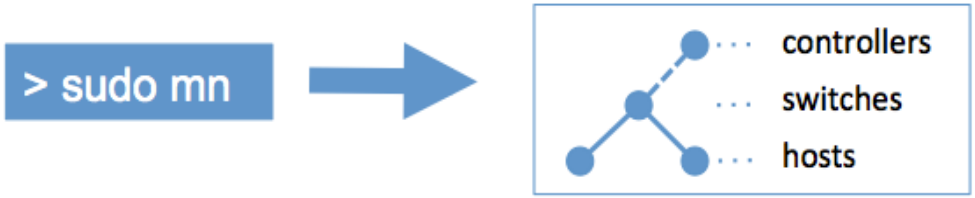
Inside Open vSwitch

- The problem though is when it's still running a packet classifier in the kernel in software. What this means is for **every packet that comes in, you are searching in this table for the right entry that matters and using that entry for forward the packet. This can be quite slow.**
- **Open vSwitch solves this problem is to create a simple hash table based cache into the classifier.** So instead of looking at this entire table and finding the right rule. You'll hash what fields are used to match packet, and the hash key is now your pointer to the action that needs to be taken. And, these hash keys and their actions can be cache.



Introduction to Mininet

- Mininet creates a **realistic virtual network, running real kernel, switch and application code,** on a single machine (VM, cloud or native), in seconds, with a single command:

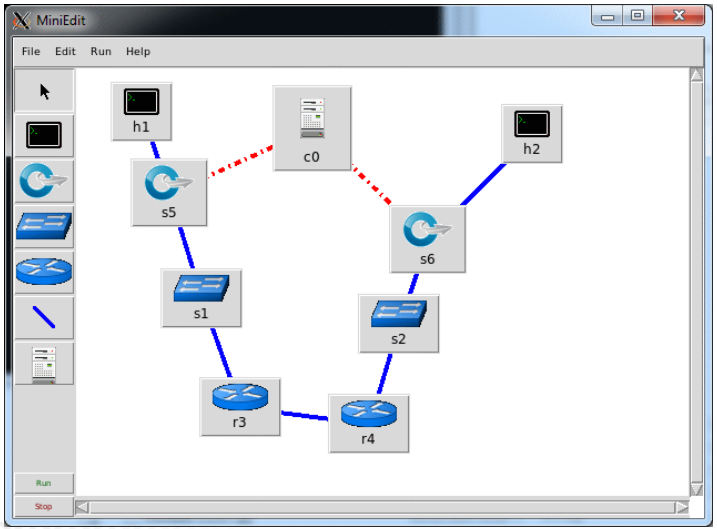


It is a **network emulator** which creates realistic virtual network

  - Runs real kernel, switch and application code on a single machine
  - Provides both Command Line Interface (CLI) and Application Programming Interface (API)
    - **CLI: interactive commanding**
    - **API: automation**
  - **Abstraction**
    - Host: emulated as an OS level process
    - Switch: emulated by using software-based switch
      - **E.g., Open vSwitch, SoftSwitch**

Mininet Applications

- **MiniEdit (Mininet GUI)**
  - A GUI application which eases the Mininet topology generation
  - Either save the topology or export as a Mininet python script
- **Visual Network Description (VND)**
  - A GUI tool which allows automate creation of Mininet and OpenFlow controller scripts



Important Links for Mininet

- mininet.org
- github.com/mininet
- github.com/mininet/mininet/wiki/Documentation
- reproducingnetworkresearch.wordpress.com

Comparison

- The **hardware based approach** the **SR-IOV** takes **sacrifices flexibility and forwarding logic** for line rate performance in all scenarios. Virtually hitting native performance.

- While the **software based approach** **Open vSwitch**, the compromise made is to avoid targeting worst case performance, and **focusing on forwarding flexibility**.

References

- W. Felter, A. Ferreira, R. Rajamony and J. Rubio, "**An updated performance comparison of virtual machines and Linux containers**," *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, Philadelphia, PA, 2015, pp. 171-172.

- Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan J. Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Jonathan Stringer, Pravin Shelar, Keith Amidon, and Martín Casado, "**The design and implementation of open vSwitch**" In Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation (NSDI'15). USENIX Association, Berkeley, CA, USA, 2015, pp. 117-130.

Conclusion

- In this lecture, we have discussed **server virtualization and also discuss the need of routing and switching for physical and virtual machines.**

- We have discussed two methods of virtualization:
  (i) Docker based and (ii) Linux container based.

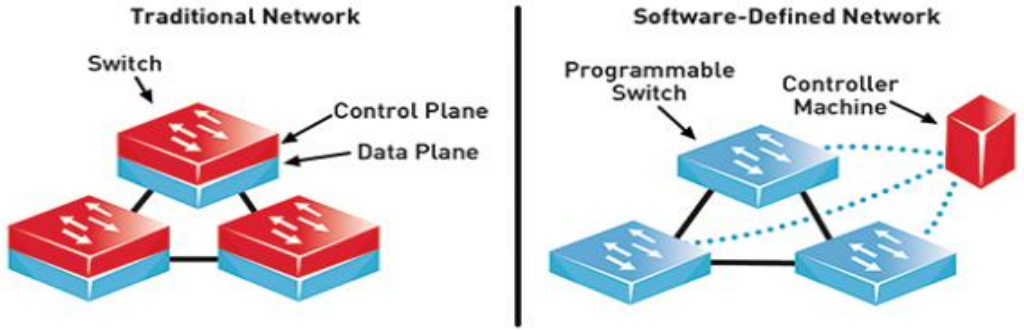
- To address the problem of networking VMs, two approaches are discussed: **(i) One, using specialized hardware: SR-IOV, single-root I/O virtualization and (ii) Other using an all software approach: Open vSwitch**

Software Defined Network Application to Networking in the Cloud

Preface

Content of this Lecture:

- In this lecture, we will discuss the architecture of software defined networking and its applications to networking in the cloud.
- We will also discuss the network Virtualization in multi-tenant data centers with case study of VL2 and NVP



Need of SDN

The traditional networking problem that SDN (software defined networking) is addressing as:

I) Complexity of existing networks

- Networks are complex just like computer system, having system with software.
- But worst than that it's a distributed system
- Even more worse: No clear programming APIs, only **“knobs and dials”** to control certain functions.

II) Network equipment traditionally is proprietary

- Integrated solutions (operating systems, software, configuration, protocol implementation , hardware) from major vendors

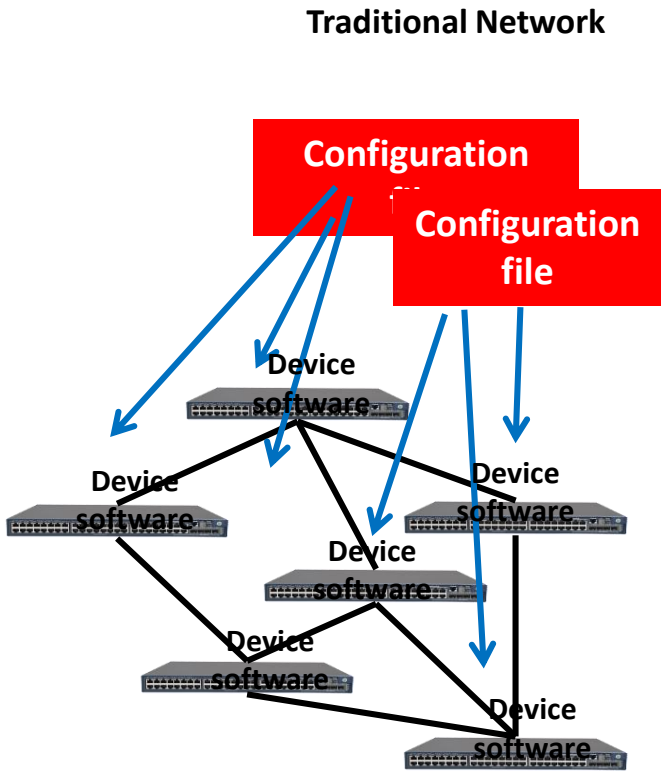
RESULT: - Hard and time intensive to innovate new kinds of networks and new services or modify the traditional networks more efficiently.

Traditional Network

In traditional network, **configuring a router**, for example, for BGP routing, in a configuration file.

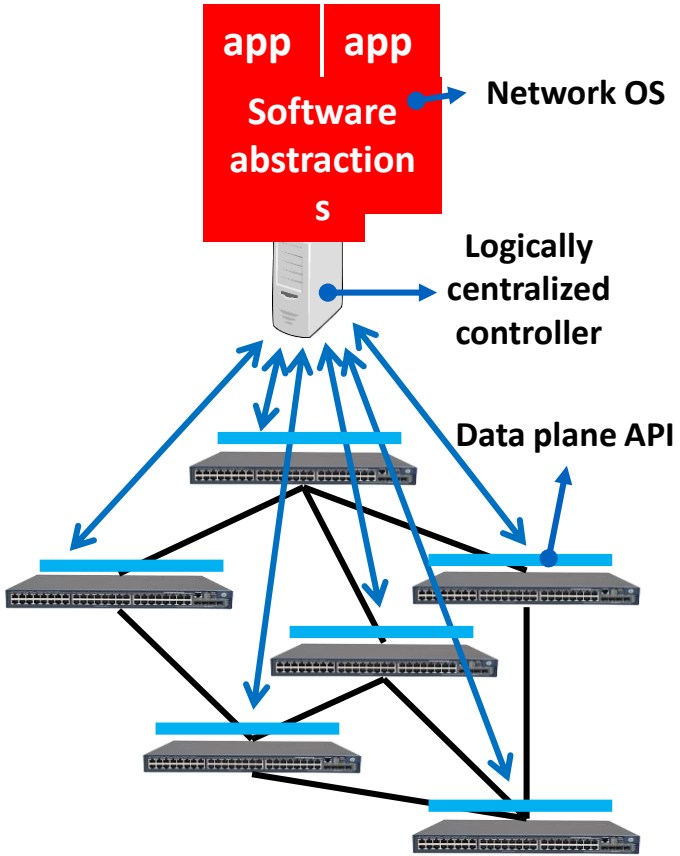
Hundreds, thousands, tens of thousands network devices, router switches and firewalls throughout the network that will **get pushed out configuration to various devices on the network.**

Once it reaches a device then these **configurations have to be interpreted by router software and implementation of protocols** that run distributed algorithms and arrive at routing solutions that produce the ultimate behavior that's installed in the network hardware.
So here the **policy is embedded into the mechanism.** It means the network routing to achieve low latency or high utilization of the network that control are baked into the distributive protocol that are in standardized implementations.



Software-defined network

- The traditional software and OSs are **built on layers and APIs.** So it begin close to the hardware build a low level interface that gives direct access to what the network switching hardware is doing.
- Then a **logically centralized controller** which communicates with distributed switches and other devices in the network.
- The goal of a logically centralized controller is to express our goal in one location and **keep the widely distributed switching gear as simple as possible.**
- **Put the intelligence in a logically centralized location.** On top of that, build software abstractions that help us to build different applications, a network operating system if you want.



Key Ideas of SDN

Key ideas software-defined networking architecture:

Division of Policy and Mechanisms-

- Low-level interface and programmatic interface **for the data plane**
- **Logically centralized controller that allows us to build software abstractions on top of it.**

Example: NOX

NOX is a very early SDN controller. **So to identify a user's traffic, a particular user or computer to send traffic through the network,** and that traffic through the network is going to be tagged with an identifier, a VLAN and that identifies that user.

So to instruct the network we **match a specific set of relevant packets and look at the location where this traffic comes in from as well as the MAC address.** And we're going to construct an action that should happen, in this case, tagging, adding that VLAN tag to the traffic.

And then we're going to **install that action on the specified set of packets in a particular switch.** So we're basically telling the switch, if you see this, do that.

In addition, commonly **SDN controllers have some kind of topology discovery, the ability to control traffic and monitor the behavior in the network.**

From NOX [Gude, Koponen, Pettit, Pfaff, Casado, McKeown, Shenker, CCR 2008]

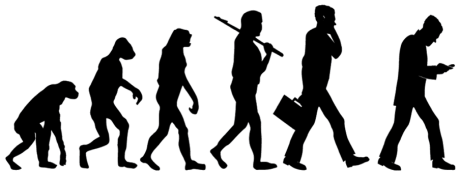
```
# On user authentication, statically setup VLAN tagging
# rules at the user's first hop switch
def setup_user_vlan(dp, user, port, host):
    vlanid = user_to_vlan_function(user)

    # For packets from the user, add a VLAN tag
    attr_out[IN_PORT] = port                                  Match specific
    attr_out[DL_SRC] = nox.reverse_resolve(host).mac          set of packets
    action_out = [(nox.OUTPUT, (0, nox.FLOOD)),               Construct action
        (nox.ADD_VLAN, (vlanid))]
    install_datapath_flow(dp, attr_out, action_out)           Install (match, action)
                                                              in a specific switch
    # For packets to the user with the VLAN tag, remove it
    attr_in[DL_DST] = nox.reverse_resolve(host).mac
    attr_in[DL_VLAN] = vlanid
    action_in = [(nox.OUTPUT, (0, nox.FLOOD)), (nox.DEL_VLAN)]
    install_datapath_flow(dp, attr_in, action_in)

nox.register_for_user_authentication(setup_user_vlan)
```

Key Ideas of SDN

- A programmatic low level interface with the data plane
- Centralized control
- Higher level abstractions that makes easier control.

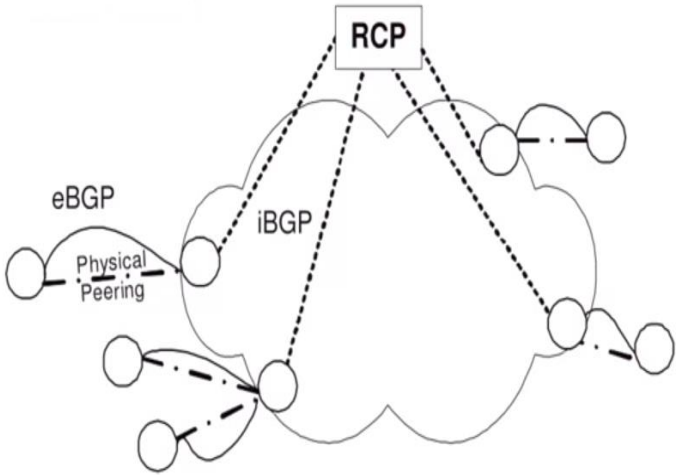
Evolution of SDN: Flexible Data Planes

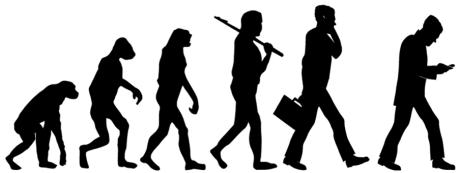
- Evolution of SDN is driving towards making the network flexible.
- **Label switching or MPLS (1997)** i.e. matching labels, executing actions based on those labels adding flexibility:-
- Lay down any path that we want in the network for certain classes of traffic.
- Go beyond simple shortest path forwarding,
- Good optimization of traffic flow to get high throughput for traffic engineering
- Setting up private connections between enterprises and distributed sites.



Evolution of SDN: Logically Centralized Control

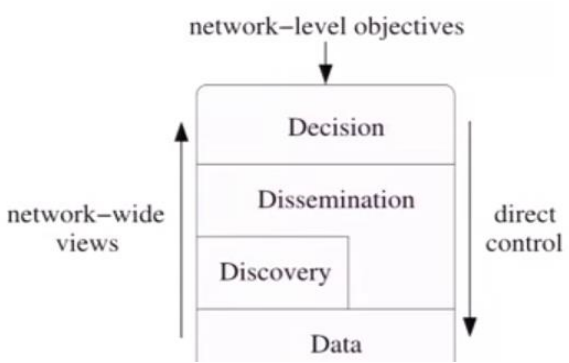
- **Routing Control Platform (2005) [Caesar et al. NSDI 2005]**
- Centralized computing to BGP routes, pushed to border routers via iBGP





Evolution of SDN: Logically Centralized Control

- **4D architecture (2005)**
  - A clean slate 4D Approach to Network control and management [Greensburg, Et. Al., CCR Oct 2005]
  - Logically centralized “decision plane” separated from data planes

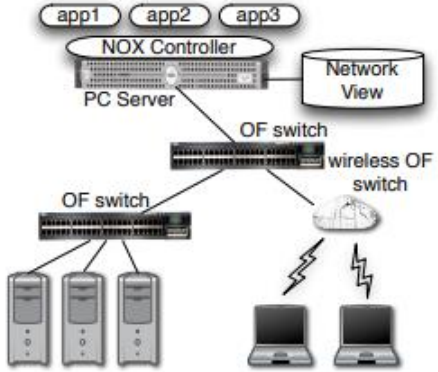


Evolution of SDN: Logically Centralized Control

- **Ethane (2007) [Casado et al., SIGCOMM 2007]**
  - Centralized controller enforces enterprise network Ethernet forwarding policy using existing hardware.

- **OpenFlow (2008) [McKeown et al. 2008]**
  - Thin standardized interface to data planes.
  - General purpose programmability at control.

- Routing Control Platform (2005)
- 4D architecture (2005)
- Ethane (2007)
- OpenFlow (2008)
- **NOX (2008) [Gude et al. CCR 2008]**
  - **First OpenFlow (OF) controller:** centralized network view provided to multiple control applications s as a database.
  - Handles state collection and distribution.
- **Industry explosion (~2010+)**



SDN Opportunities

- **Open data plane interface**
  - **Hardware :** with standardized API, easier for operators to change hardware, and for vendors to enter market
  - **Software :** can more directly access device behavior
- **Centralized controller:**
  - Direct programmatic control of network
- **Software abstraction of the controller**
  - Solves distributed systems problem only once, then just write algorithm.
  - Libraries/languages to help programmers write net apps
  - Systems to write high level policies instead of programming

Challenges of SDN

Performance and Scalability

- Controlling the network through devices to respond quickly with latency concerns i.e. capacity concerns.

Distributed systems challenges still present

- Network is fundamentally a distributed system
- Resilience of logically centralized controllers
- Imperfect knowledge of network state
- Consistency issues between controllers

Architectural Challenges of SDN

- **Protocol to program the data planes**
  - OpenFlow ? NFV function ? WhiteBox switching ?

- **Devising the right control abstraction ?**
  - Programming OpenFlow : far too low level
  - But what are the right level abstractions to cover important use cases ?

The First Cloud Apps for SDN

- **Virtualization of multi-tenant data centers**
  - Create separate virtual network for tenants
  - Allow flexible placement and movements of VMs
- **Inter-datacenter traffic engineering**
  - Trying to achieve maximum utilization near 100% if possible.
  - Protect critical traffic from congestion.
- **Key-characteristics for above use cases**
  - Special purpose deployments with less diverse hardware.
  - Existing solutions aren’t just inconvenient and don’t work.

Multi-tenant Data Centers : The challenges

Cloud is shared among multiple parties and gives economy of scale. To share the cloud among multiple tenants, there's bit more work to do. So the key needs for building a multi-tenant Cloud data center are:

(i) Agility

(ii) Location independent addressing

(iii) Performance uniformity

(iv) Security

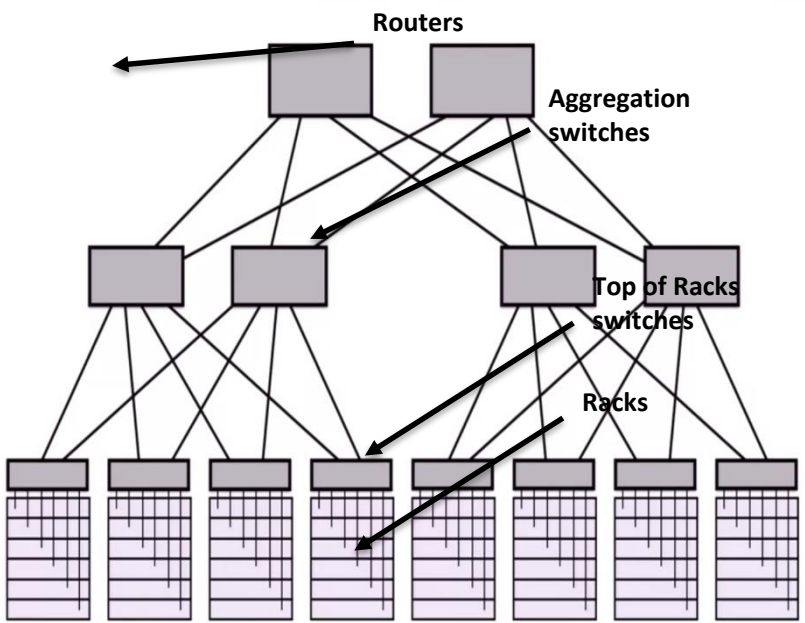
(v) Network semantics

(i) Agility

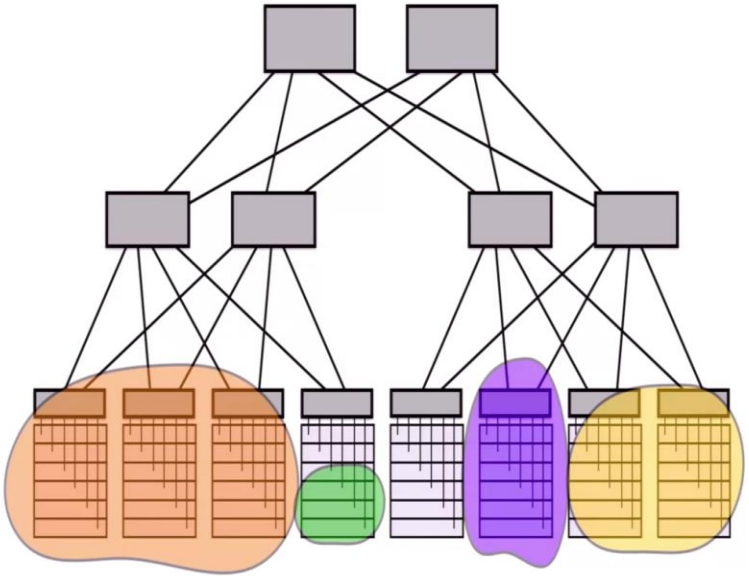
- **Use any server for any service at any time:**
  - **Better economy of scale through increased utilization:** Pack, compute as best we can for high utilization. If we ever have constraints then it's going to be a lot harder to make full use of resources.
  - **Improved reliability:** If there is a planned outage or an unexpected outage, move the services to keep running uninterrupted.

- **Service or tenant can means:**
  - A customer renting space in a public cloud
  - Application or service in a private cloud as an internal customer

Traditional Datacenters



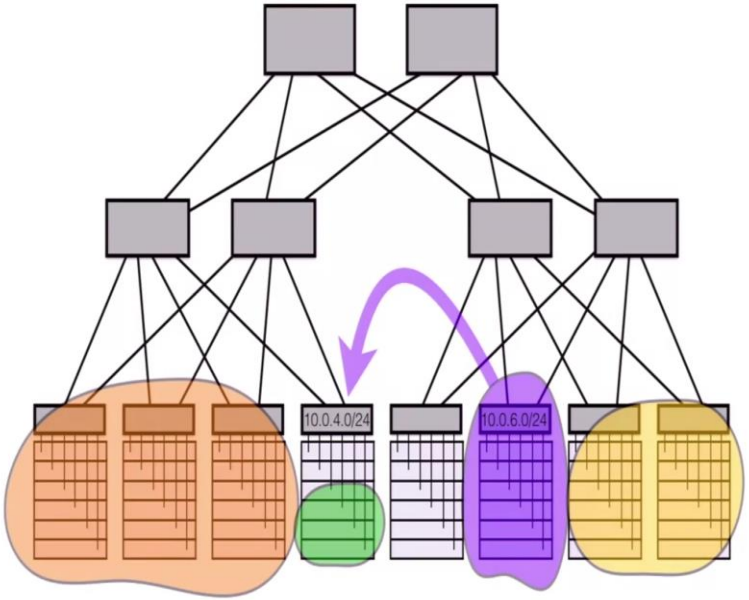
Lack of Agility in Traditional DCs



- **Tenant in “silos” –** Means one rack or a part of the cluster is devoted to a particular service

- **Poor Utilization**
- **Inability to expand**

Lack of Agility in Traditional DCs



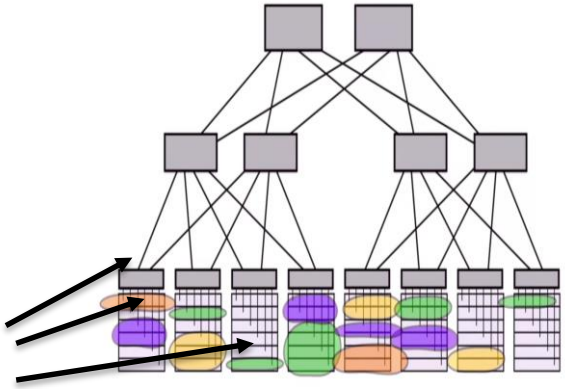
- **IP addresses locked to topological location!**

Key needs: Agility

- **Agility**
  - **Location independent addressing:** Racks are generally assigned different IP Subnets, because subnets are used as topological locators so that we can route. To move some service over there, we're going to have to change its IP address and it is hard to change the IP addresses of live running services.
  - **Tenant's IP address can be taken anywhere:** Tenant's IP address to be taken anywhere, independent of the location and the data center without notify tenants that it has changed location. Large over subscription ratio i.e. 100 % or greater if there's a lot of communication between both sides will be about a hundred times lower throughput than communicating within the rack.

Key needs: Performance Uniformity

- **Performance Uniformity**
  - Wherever the VMs are, they will see the **same performance and latency.**
  - **Smaller units of compute** that we've divided our services into, and put them anywhere in the data center, and may be on the same physical machine.

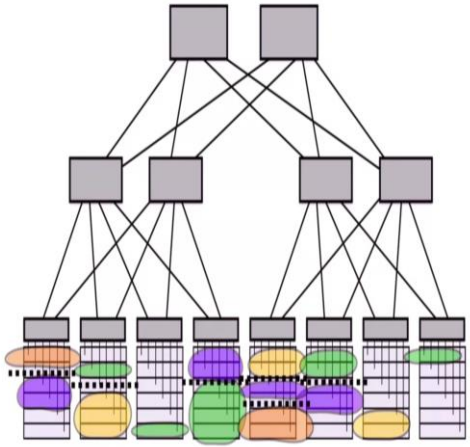


Smaller units of compute

Key needs: Security

- **Security:** Untrusting applications and users sitting right next to each other and can be inbound attacks. So to protect our tenants in the data center from each other in both the public data center as well as in the private cloud and you don't want them to have to trust each other.

  - **Micro-segmentation :** separation of different regions of a network.
  - Much finer grained division and control, of how data can flow.
  - Isolate or control just the data flow between pairs of applications, or tenants that should be actually allowed.



Key needs: Network semantics

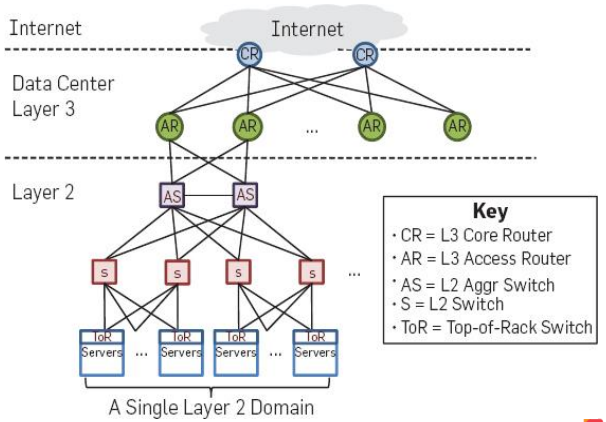
- **Network semantics:**
  - Match the functional service of a traditional data center
  - Not just Layer 3 routing services but also, Layer 2 services i.e. discovery services, multicast, broadcast etc. have to be supported.

Network Virtualization in Multi-tenant Data Centers Case Study: VL2

VL2: A Scalable and Flexible Data Center Network

Albert Greenberg, James R. Hamilton, Navendu Jain
Srikanth Kandula, Changhoon Kim, Parantap Lahiri
David A. Maltz, Parveen Patel, Sudipta Sengupta
Microsoft Research

[ACM SIGCOMM 2009]



Network Virtualization Case Study: VL2

Key Needs:

(i) Agility

(ii) Location independent addressing

(iii) Performance uniformity

(iv) Security

(v) Network Semantics

Motivating Environmental Characteristics

Increasing internal traffic is a bottleneck

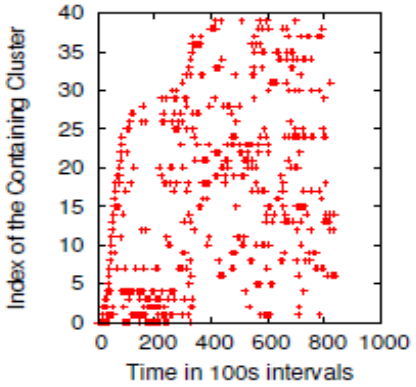
- Traffic volume between servers is 4 times larger than the external traffic

Rapidly-changing traffic matrices (TMs)

- i.e. Take traffic matrices in 100 second buckets and classify them into 40 categories of similar clusters of traffic matrices and see which of the clusters appear in the measurements
- So over time rapidly changing and no pattern to what the particular traffic matrix is.

Design result: Nonblocking fabric

- High throughput for any traffic matrices that respects server NIC rates.
- The fabric joining together all the servers, we don't want that to be a bottle neck.



[Greenberg et al.]

Motivating Environmental Characteristics

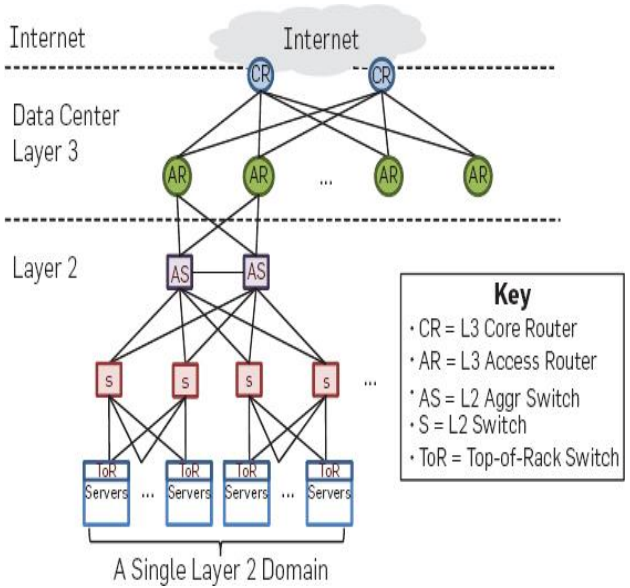
Failure characteristics:

- Analyzed 300K alarm tickets, 36 million error events from the cluster
- 0.4% of failures were resolved in over one day
- 0.3% of failures eliminated all redundancy in a device group (e.g. both uplinks)

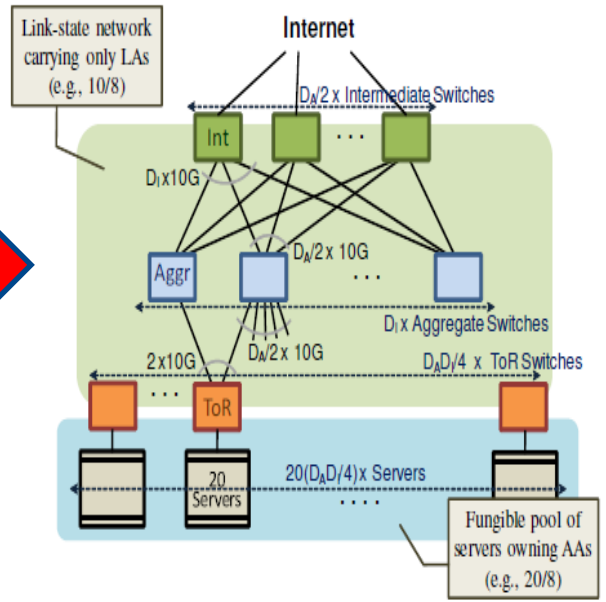
Design result: Clos topology:

- Particular kind of non blocking topology
- “Scale out” instead of “scale up”

VL2 physical topology



Traditional



VL2

An example Clos network between Aggregation and Intermediate switches provides a richly-connected backbone well suited for VLB. .e network is built with two separate address families—topologically significant Locator Addresses (LAs) and at Application Addresses (AAs).

Figures from Greenberg et al.

Routing in VL2

Unpredictable traffic

- Means it is difficult to adapt. So this leads us to a design that is what's called **oblivious routing**. It means that the path along which we send a particular flow does not depend on the current traffic matrix.

Design result: “Valiant Load Balancing”

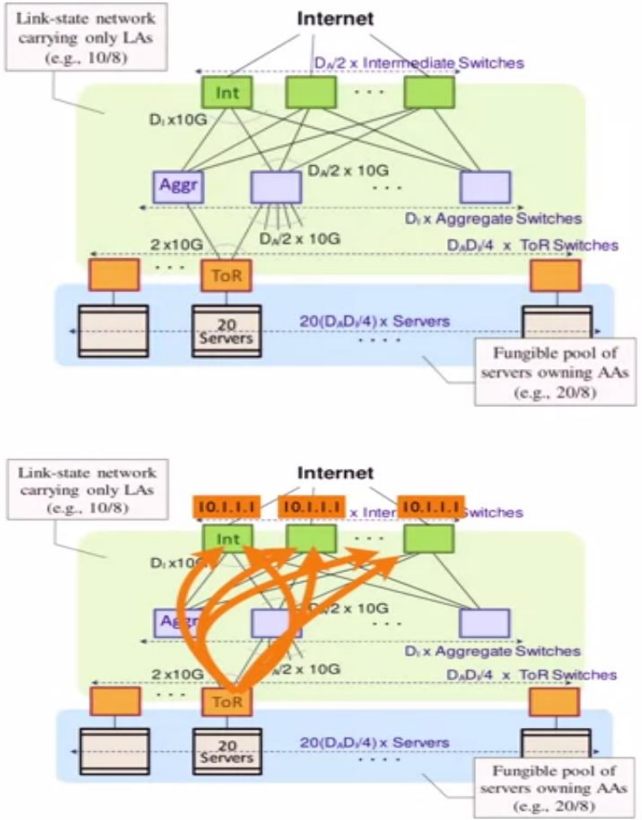
- For routing on hyper cubes take an arbitrary traffic matrix and make it look like a completely uniform even traffic matrix.
- Take flows and spreading evenly over all the available paths. Spread traffic as much as possible.
- Route traffic independent of current traffic matrix

Routing Implementation

Spreads arbitrary traffic pattern so it's uniform among top layer switches which are called intermediate switches.

Now to do that what VL2 does is **it assigns those intermediate switches and any cast address.** The same any cast address for all of the switches.

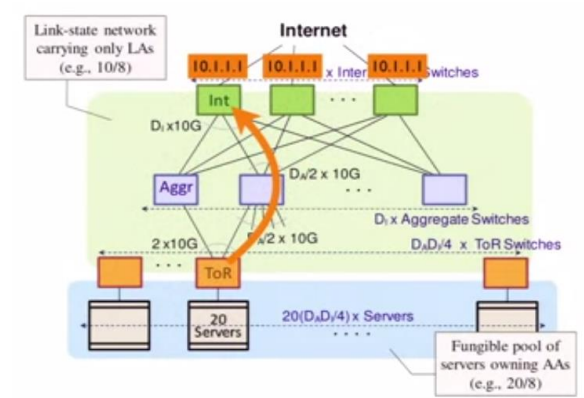
So, then a top of rack switch can send to a random one by just using that single address. And if we are using **ECMP we will use a random one of those paths that are shortest.**

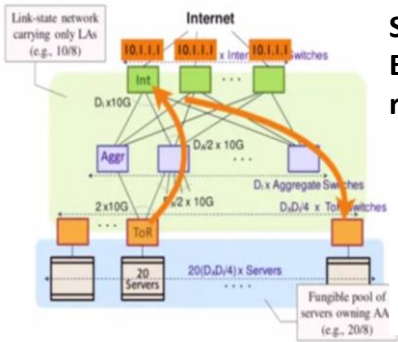


Routing Implementation

- As all of the paths are shortest because all of those intermediate switches are the same distance from the top of racks, So ECMP is going to give the full breadth of possible paths to any one of those switches, just by naming the single anycast address of all of the intermediates.

- **ECMP lets us select from one of those paths then one will be picked from any particular flow.** We send it to that intermediate switch. Now that outer anycast address is wrapping an inner header that actually has the destination address, in this design. So we'll forward it from there onto the destination.





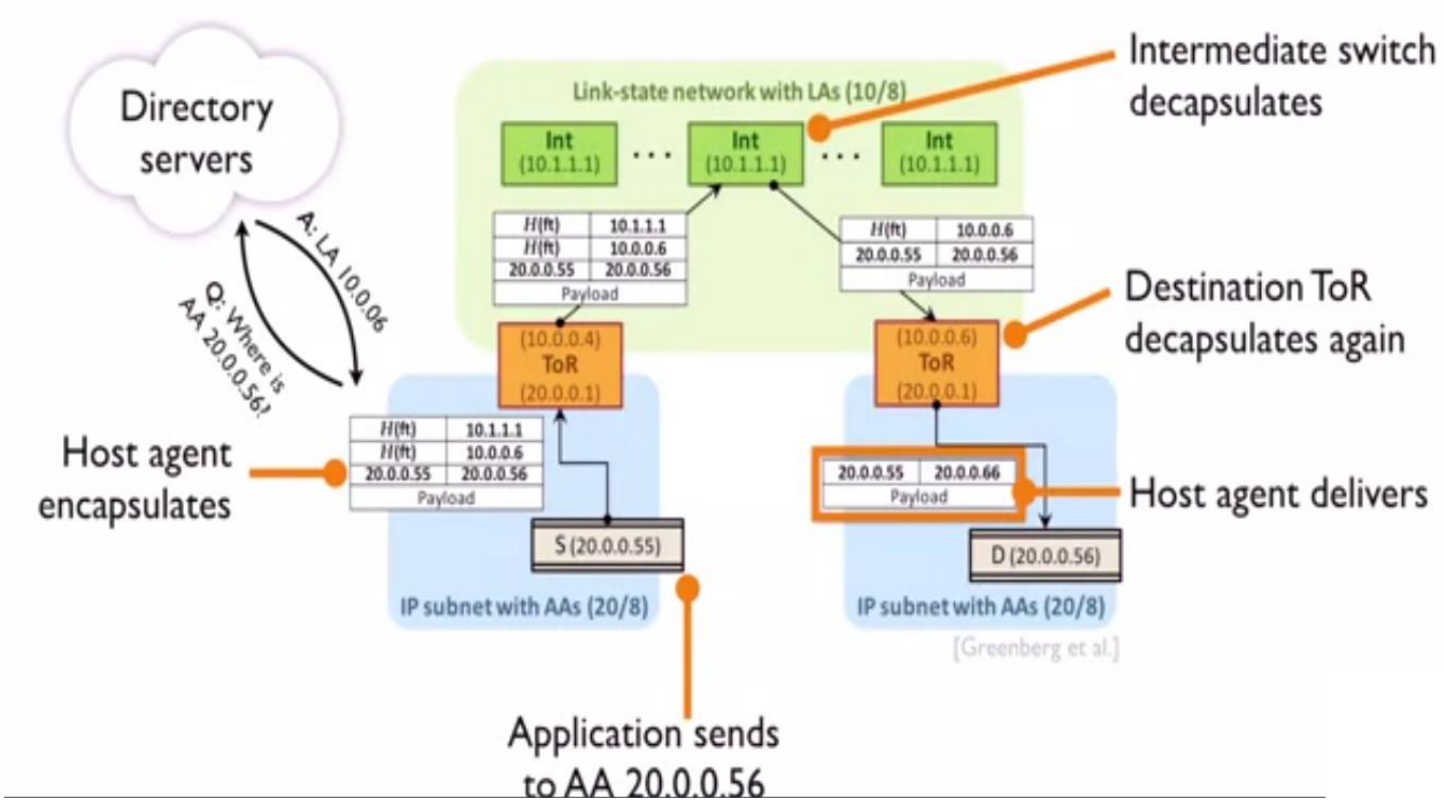
Similar effect to ECMP to each rack

Smaller forwarding table at most switches

Any service anywhere

App/Tenant layer	Application or tenant of the data center is going to see what's called application addresses. These are location independent, Same address no matter where the VM goes. And they're going to see the illusion of a single big Layer 2 switch connecting all of the application's VMs.
Indirection or Virtualization layer	Maintains a directory server that maps the application level addresses to their current locators. VL2 has agents that run on the server that will query the directory server and find that AA to LA mapping. And then when it sends a packet, it'll wrap the application address in the outer locator header
Physical network layer	Different set of IP addresses called locator addresses. Tied to topology used to route Layer 3 routing via OSPF

End-to-end Example



Did we achieve agility?

Location independent addressing

- AAs are location independent

L2 network semantics

- Agent intercepts and handles L2 broadcast, multicast

- Both of the above require “layer 2.5” shim agent running on host; but, concept transfers to hypervisor-based virtual switch

Did we achieve agility?

Performance uniformity:

- Clos network is nonblocking (non-oversubscribed)
- Uniform capacity everywhere.
- ECMP provides good (though not perfect) load balancing
- But performance isolation among tenants depends on TCP backing off to the rate that the destination can receive.
- Leaves open the possibility of fast load balancing

Security:

- Directory system can allow/deny connections by choosing whether to resolve an AA to a LA
- But, segmentation not explicitly enforced at hosts

Where's the SDN?

Directory servers: Logically centralized control

- Orchestrate application locations
- Control communication policy

Hosts agents: dynamic "programming" of data path

Network Virtualization Case Study: NVP

Network Virtualization in Multi-tenant Datacenters

Teemu Koponen, Keith Amidon, Peter Balland, Martín Casado, Anupam Chanda, Bryan Fulton, Igor Ganichev, Jesse Gross, Natasha Gude, Paul Ingram, Ethan Jackson, Andrew Lambeth, Romain Lenglet, Shih-Hao Li, Amar Padmanabhan, Justin Pettit, Ben Pfaff, and Rajiv Ramanathan, *VMware;* Scott Shenker, *International Computer Science Institute and the University of California, Berkeley;* Alan Shieh, Jeremy Stribling, Pankaj Thakkar, Dan Wendlandt, Alexander Yip, and Ronghua Zhang, *VMware*

https://www.usenix.org/conference/nsdi14/technical-sessions/presentation/koponen

This paper is included in the Proceedings of the
11th USENIX Symposium on Networked Systems
Design and Implementation (NSDI '14).

April 2–4, 2014 • Seattle, WA, USA

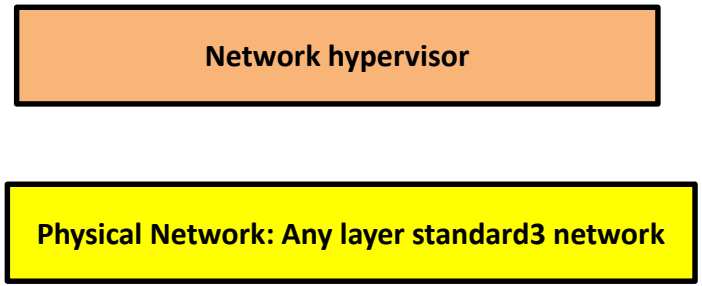
NVP Approach to Virtualization

- The network virtualization platform that was introduced in the paper **“Network virtualization in Multi-tenant Datacenters”** by Teemu Koponen et al. in NSDI 2014.

- And this comes out of a product developed by the Nicira startup that was acquired by VMware.

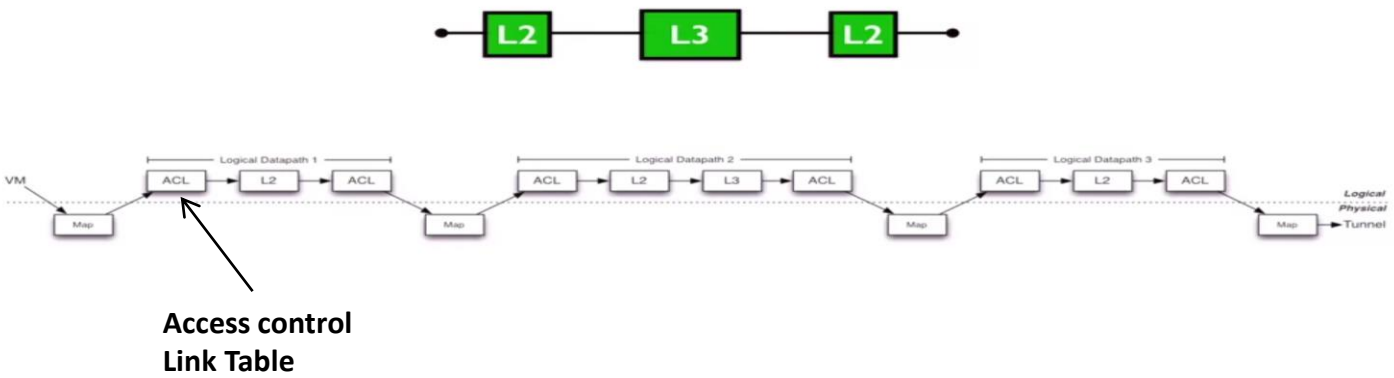
Service: Arbitrary network topology

- Replicate arbitrary topology
- Any standard layer 3 network
- Network hypervisor
- Virtual network tenants want to build



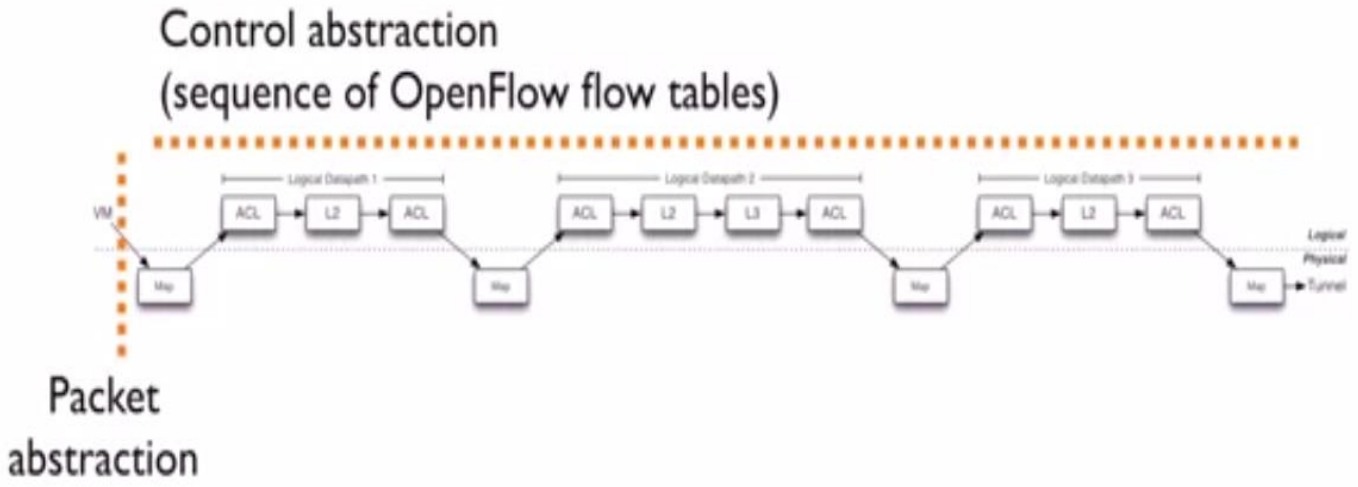
Virtual Network Service

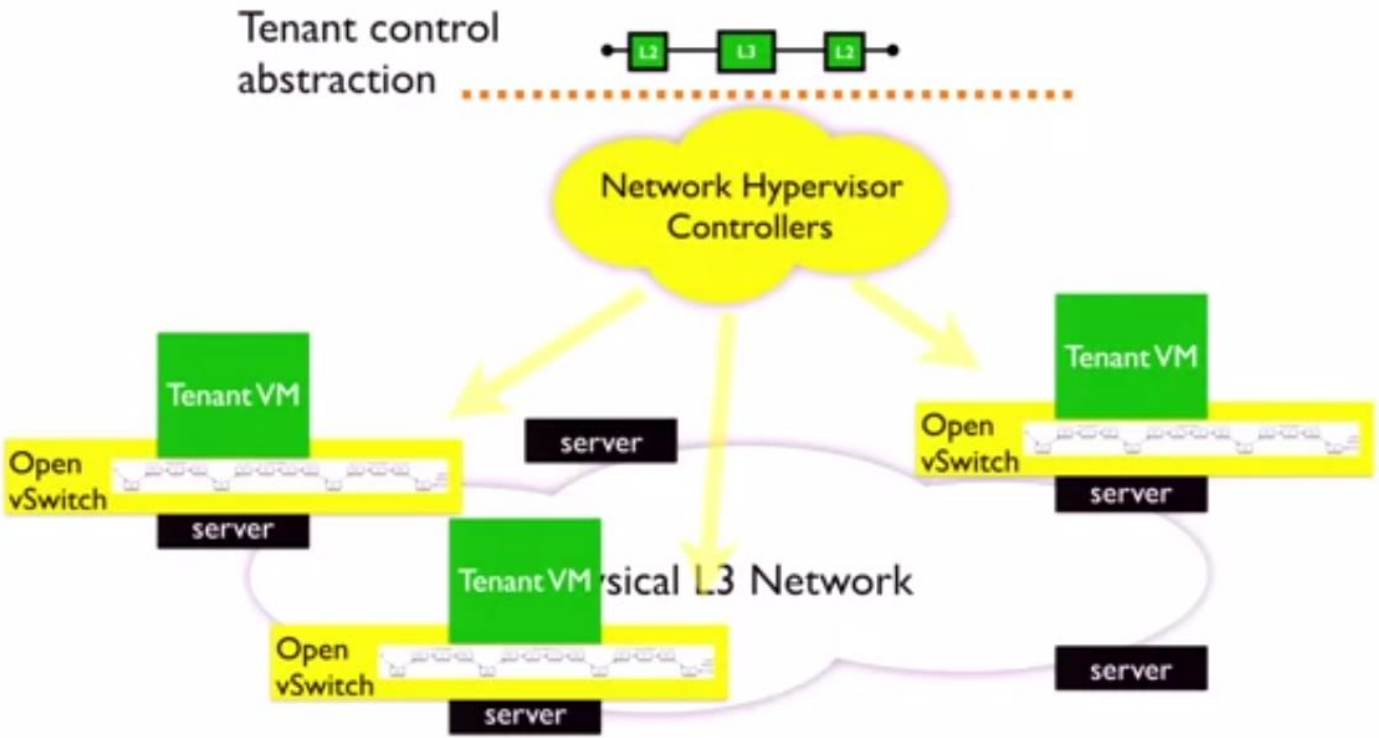
- Modeled as a sequence of data path elements that represent those switches.
- And these data path elements, each one of them is a OpenFlow forwarding table.
- It means the table will match on certain signature of packet headers and take certain resulting actions like dropping the packet, modifying certain fields in the packet, or forwarding the packet on.
- So the idea's that we can model the switching, and routing gear with the right sequences of the right OpenFlow tables that we set up.



Virtual Network Service

- There's a **packet abstraction** where virtual machines are able to inject traffic into the virtual network.
- And there's a **control abstraction** where the tenant was able to define this entire virtual network pipeline, that sequence of OpenFlow flow tables.
- That's the interface that the tenant is given, at least the lowest level interface, that the tenant is given to be able to program their virtual network.




Tenant control abstraction
L2
L3
L2
Network Hypervisor Controllers
Tenant VM
Open vSwitch
server
Tenant VM
Open vSwitch
server
Tenant VM
Open vSwitch
server
sical L3 Network
server

Challenge: Performance

Large amount of state to compute

- Full virtual network state at every host with a tenant VM!
- $O(n^2)$ tunnels for tenant with n VMs
- Solution 1: Automated incremental state computation with nlog declarative language
- Solution 2: Logical controller computes single set of universal flows for a tenant, translated more locally by “physical controllers”

Challenge: Performance

Pipeline processing in virtual switch can be slow

- **Solution:** Send first packet of a flow through the full pipeline: thereafter, put an exact-match packet entry in the kernel

Tunneling interfaces with TCP Segmentation Offload (TSO)

- NIC can’t see TCP outer header
- Solution: STT tunnels adds “fake” outer TCP header

Conclusion

- In this lecture, we have discussed the **need of software defined network, key ideas and challenges of software defined network.**

- We have also discussed the challenges in multi-tenant data centers i.e. **(i) Agility, (ii) Location independent addressing (iii) Performance uniformity, (iv) Security and (v) Network semantics.**

- Then we have discussed the **network Virtualization in multi-tenant data centers with a case study of VL2 and NVP**

TATA CONSULTANCY SERVICES

tcs iON

Design of Key-Value Stores

Preface

Content of this Lecture:

- In this lecture, we will discuss the design and insight of **Key-value/NoSQL stores** for today's cloud storage systems.

- We will also discuss one of the most popular cloud storage system i.e. **Apache Cassandra** and different consistency solutions.

The Key-value Abstraction

- (Business) Key → Value
- (flipkart.com) item number → information about it
- (easemytrip.com) Flight number → information about flight, e.g., availability
- (twitter.com) tweet id → information about tweet
- (mybank.com) Account number → information about it

The Key-value Abstraction (2)

- **It's a dictionary datastructure.**
  - Insert, lookup, and delete by key
  - Example: hash table, binary tree

- But distributed.

- Seems familiar? Remember **Distributed Hash tables (DHT) in P2P systems?**

- Key-value stores reuse many techniques from DHTs.

Is it a kind of database ?

- Yes, kind of
- **Relational Database Management Systems (RDBMSs)** have been around for ages
- **MySQL** is the most popular among them
- Data stored in tables
- Schema-based, i.e., structured tables
- Each row (data item) in a table has a primary key that is unique within that table
- Queried using **SQL (Structured Query Language)**
- Supports joins

Relational Database Example

users table

user_id	name	zipcode	blog_url	blog_id
110	Smith	98765	smith.com	11
331	Antony	54321	antony.in	12
767	John	75676	john.net	13

Primary keys

Foreign keys

blog table

Id	url	last_updated	num_posts
11	smith.com	9/7/17	991
13	john.net	4/2/18	57
12	antony.in	15/6/16	1090

Example SQL queries

1. SELECT zipcode
   FROM users
   WHERE name = "John"

2. SELECT url
   FROM blog
   WHERE id = 11

3. SELECT users.zipcode, blog.num_posts
   FROM users JOIN blog
   ON users.blog_url = blog.url

Mismatch with today's workloads

- **Data: Large and unstructured:** Difficult to come out with schemas where the data can fit
- **Lots of random reads and writes:** Coming from millions of clients.
- **Sometimes write-heavy:** Lot more writes compare to read
- **Foreign keys rarely needed**
- **Joins infrequent**

Needs of Today's Workloads

- Speed
- Avoid Single point of Failure (SPoF)
- Low TCO (Total cost of operation and Total cost of ownership)
- Fewer system administrators
- Incremental Scalability
- Scale out, not scale up

Scale out, not Scale up

- **Scale up = grow your cluster capacity by replacing with more powerful machines**
  - Traditional approach
  - Not cost-effective, as you're buying above the sweet spot on the price curve
  - And you need to replace machines often

- **Scale out = incrementally grow your cluster capacity by adding more COTS machines (Components Off the Shelf)**
  - Cheaper
  - Over a long duration, phase in a few newer (faster) machines as you phase out a few older machines
  - Used by most companies who run datacenters and clouds today

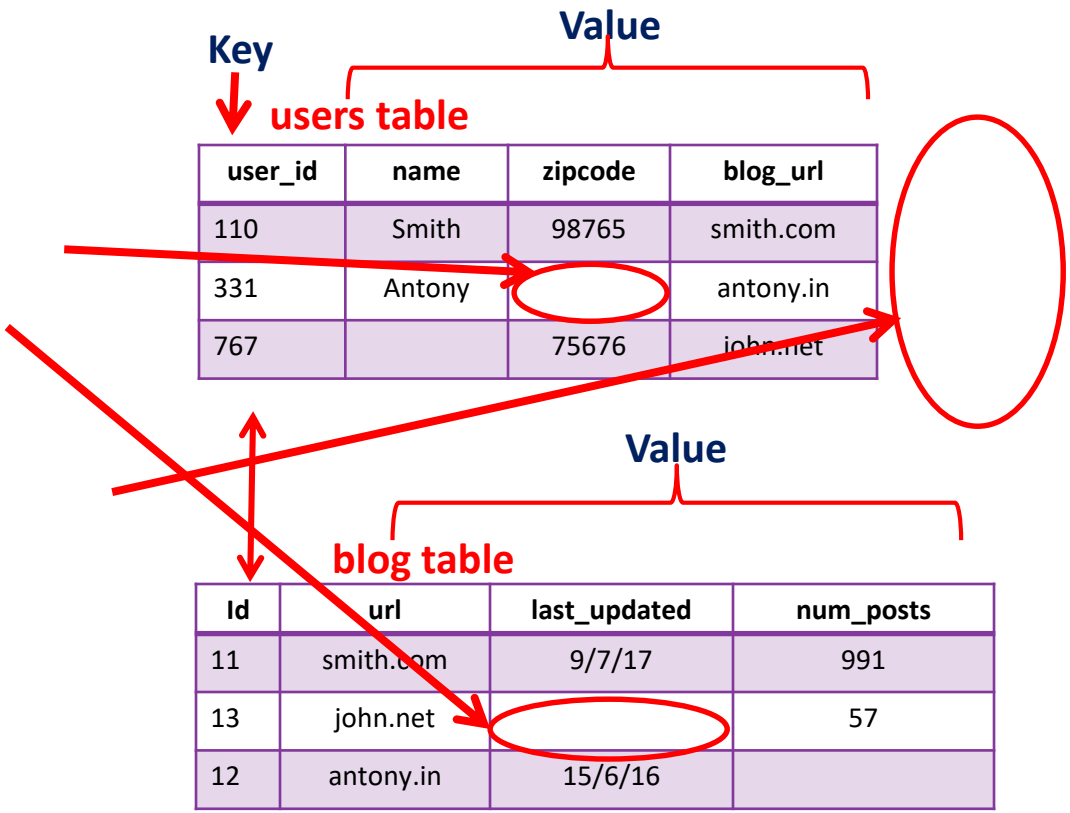
Key-value/NoSQL Data Model

- NoSQL = “Not Only SQL”
- Necessary API operations: **get(key) and put(key, value)**
  - And some extended operations, e.g., “CQL” in Cassandra key-value store

- **Tables**
  - “Column families” in Cassandra, “Table” in HBase, “Collection” in MongoDB
  - Like RDBMS tables, but …
  - May be unstructured: May not have schemas
    - Some columns may be missing from some rows
  - Don’t always support joins or have foreign keys
  - Can have index tables, just like RDBMSs

Key-value/NoSQL Data Model

- Unstructured
- No schema imposed
- Columns missing from some Rows
- No foreign keys, joins may not be supported



Column-Oriented Storage

NoSQL systems often use column-oriented storage

- RDBMSs store an entire row together (on disk or at a server)
- NoSQL systems typically store a column together (or a group of columns).
    - Entries within a column are indexed and easy to locate, given a key (and vice-versa)
- **Why useful?**
    - Range searches within a column are fast since you don't need to fetch the entire database
    - E.g., Get me all the blog_ids from the blog table that were updated within the past month
        - Search in the the last_updated column, fetch corresponding blog_id column
        - Don't need to fetch the other columns

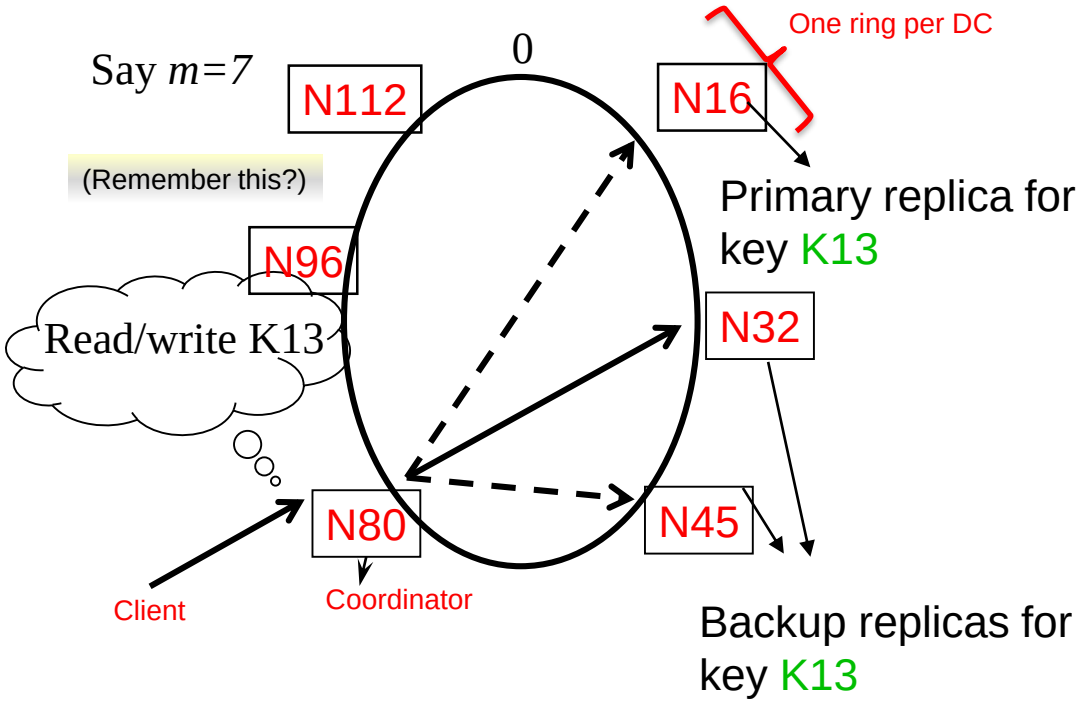
Design of Apache Cassandra

Cassandra

- **A distributed key-value store**
- Intended to run in a datacenter (and also across DCs)
- Originally designed at Facebook
- Open-sourced later, today an Apache project
- Some of the companies that use Cassandra in their production clusters
  - Blue chip companies: IBM, Adobe, HP, eBay, Ericsson
  - Newer companies: Twitter
  - Nonprofit companies: PBS Kids
  - Netflix: uses Cassandra to keep track of positions in the video.

Inside Cassandra: Key -> Server Mapping

- **How do you decide which server(s) a key-value resides on?**



Cassandra uses a Ring-based DHT but without finger tables or routing

Key→server mapping is the "Partitioner"

Data Placement Strategies

- **Replication Strategy:**
  1. *SimpleStrategy*
  2. *NetworkTopologyStrategy*

1. **SimpleStrategy:** uses the Partitioner, of which there are two kinds
   1. ***RandomPartitioner*:** Chord-like hash partitioning
   2. ***ByteOrderedPartitioner*:** Assigns ranges of keys to servers.
      - Easier for ***range queries*** (e.g., Get me all twitter users starting with [a-b])
2. **NetworkTopologyStrategy:** for multi-DC deployments
   - Two replicas per DC
   - Three replicas per DC
   - Per DC
     - First replica placed according to Partitioner
     - Then go clockwise around ring until you hit a different rack

Snitches

- **Maps:** IPs to racks and DCs. Configured in cassandra.yaml config file
- **Some options:**
  - **SimpleSnitch:** Unaware of Topology (Rack-unaware)
  - **RackInferring:** Assumes topology of network by octet of server's IP address
    - 101.102.103.104 = x.<DC octet>.<rack octet>.<node octet>
  - **PropertyFileSnitch:** uses a config file
  - **EC2Snitch:** uses EC2.
    - EC2 Region = DC
    - Availability zone = rack
- Other snitch options available

Writes

- Need to be lock-free and fast (no reads or disk seeks)
- Client sends write to one coordinator node in Cassandra cluster
  - Coordinator may be per-key, or per-client, or per-query
  - Per-key Coordinator ensures writes for the key are serialized
- Coordinator uses Partitioner to send query to all replica nodes responsible for key
- When X replicas respond, coordinator returns an acknowledgement to the client
  - X?

Writes (2)

- **Always writable: Hinted Handoff mechanism**
  - If any replica is down, the coordinator writes to all other replicas, and keeps the write locally until down replica comes back up.
  - When all replicas are down, the Coordinator (front end) buffers writes (for up to a few hours).

- **One ring per datacenter**
  - Per-DC coordinator elected to coordinate with other DCs
  - Election done via Zookeeper, which runs a Paxos (consensus) variant

Writes at a replica node

On receiving a write

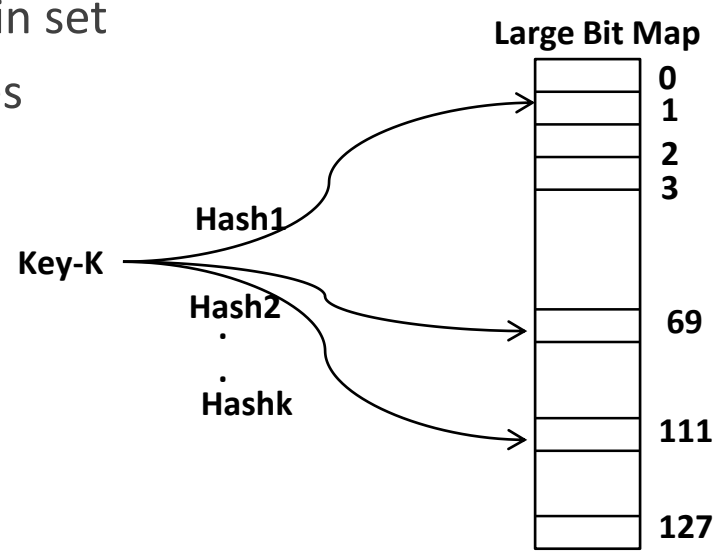
1. Log it in disk commit log (for failure recovery)
2. Make changes to appropriate memtables
   - **Memtable** = In-memory representation of multiple key-value pairs
   - *Typically append-only datastructure (fast)*
   - Cache that can be searched by key
   - Write-back aas opposed to write-through

Later, when memtable is full or old, flush to disk

- Data File: An **SSTable** (Sorted String Table) – list of key-value pairs, sorted by key
- *SSTables are immutable (once created, they don't change)*
- Index file: An SSTable of (key, position in data sstable) pairs
- And a Bloom filter (for efficient search)

Bloom Filter

- Compact way of representing a set of items
- Checking for existence in set is cheap
- Some probability of false positives: an item not in set may check true as being in set
- Never false negatives



On insert, set all hashed bits.

On check-if-present, return true if all hashed bits set.

- **False positives**

False positive rate low

- **m=4 hash functions**
- **100 items**
- **3200 bits**
- **FP rate = 0.02%**

Compaction

Data updates accumulate over time and SStables and logs need to be compacted

- The process of compaction merges SSTables, i.e., by merging updates for a key
- Run periodically and locally at each server

Deletes

Delete: don’t delete item right away

- Add a **tombstone** to the log
- Eventually, when compaction encounters tombstone it will delete item

Reads

Read: Similar to writes, except

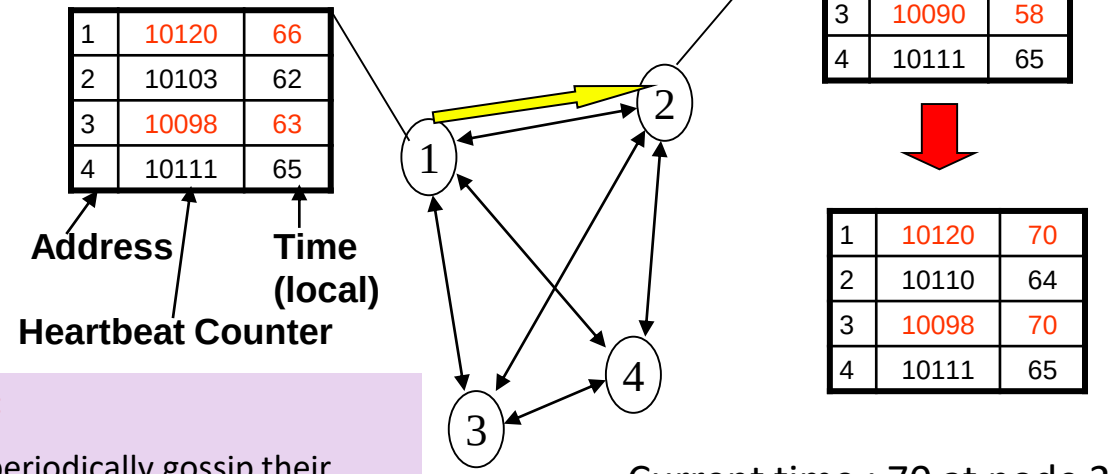
- **Coordinator can contact X replicas (e.g., in same rack)**
  - Coordinator sends read to replicas that have responded quickest in past
  - When X replicas respond, coordinator returns the latest-timestamped value from among those X
  - (X? We will check it later.)
- **Coordinator also fetches value from other replicas**
  - Checks consistency in the background, initiating a **read repair** if any two values are different
  - This mechanism seeks to eventually bring all replicas up to date
- **At a replica**
  - A row may be split across multiple SSTables => reads need to touch multiple SSTables => reads slower than writes (but still fast)

Membership

- Any server in cluster could be the coordinator
- So every server needs to maintain a list of all the other servers that are currently in the server
- List needs to be updated automatically as servers join, leave, and fail

Cluster Membership – Gossip-Style

Cassandra uses gossip-based cluster membership



Protocol:

- Nodes periodically gossip their membership list
- On receipt, the local membership list is updated, as shown
- If any heartbeat older than Tfail, node is marked as failed

Current time : 70 at node 2

(asynchronous clocks)

(Remember this?)

Suspicion Mechanisms in Cassandra

- Suspicion mechanisms to adaptively set the timeout based on underlying network and failure behavior
- **Accrual detector:** Failure Detector outputs a value (PHI) representing suspicion
- Applications set an appropriate threshold
- **PHI calculation for a member**
  - Inter-arrival times for gossip messages
  - PHI(t) =
    - log(CDF or Probability(t_now – t_last))/log 10
  - PHI basically determines the detection timeout, but takes into account historical inter-arrival time variations for gossiped heartbeats
- In practice, PHI = 5 => 10-15 sec detection time

Cassandra Vs. RDBMS

- MySQL is one of the most popular (and has been for a while)
- On > 50 GB data
- **MySQL**
  - Writes 300 ms avg
  - Reads 350 ms avg
- **Cassandra**
  - Writes 0.12 ms avg
  - Reads 15 ms avg
- Orders of magnitude faster
- What's the catch? What did we lose?

CAP Theorem

CAP Theorem

- **Proposed by Eric Brewer (Berkeley)**
- Subsequently proved by Gilbert and Lynch (NUS and MIT)
- In a distributed system you can satisfy atmost 2 out of the 3 guarantees:

1. **Consistency:** all nodes see same data at any time, or reads return latest written value by any client
2. **Availability:** the system allows operations all the time, and operations return quickly
3. **Partition-tolerance:** the system continues to work in spite of network partitions

Why is Availability Important?

- **Availability =** Reads/writes complete reliably and quickly.
- Measurements have shown that a 500 ms increase in latency for operations at Amazon.com or at Google.com can cause a 20% drop in revenue.
- At Amazon, each added millisecond of latency implies a $6M yearly loss.
- **User cognitive drift:** If more than a second elapses between clicking and material appearing, the user's mind is already somewhere else
- SLAs (Service Level Agreements) written by providers predominantly deal with latencies faced by clients.

Why is Consistency Important?

- **Consistency =** all nodes see same data at any time, or reads return latest written value by any client.

- When you access your bank or investment account via multiple clients (laptop, workstation, phone, tablet), you want the updates done from one client to be visible to other clients.

- When thousands of customers are looking to book a flight, all updates from any client (e.g., book a flight) should be accessible by other clients.

Why is Partition-Tolerance Important?

- **Partitions** can happen across datacenters when the Internet gets disconnected
  - Internet router outages
  - Under-sea cables cut
  - DNS not working

- Partitions can also occur within a datacenter, e.g., a rack switch outage

- Still desire system to continue functioning normally under this scenario

CAP Theorem Fallout

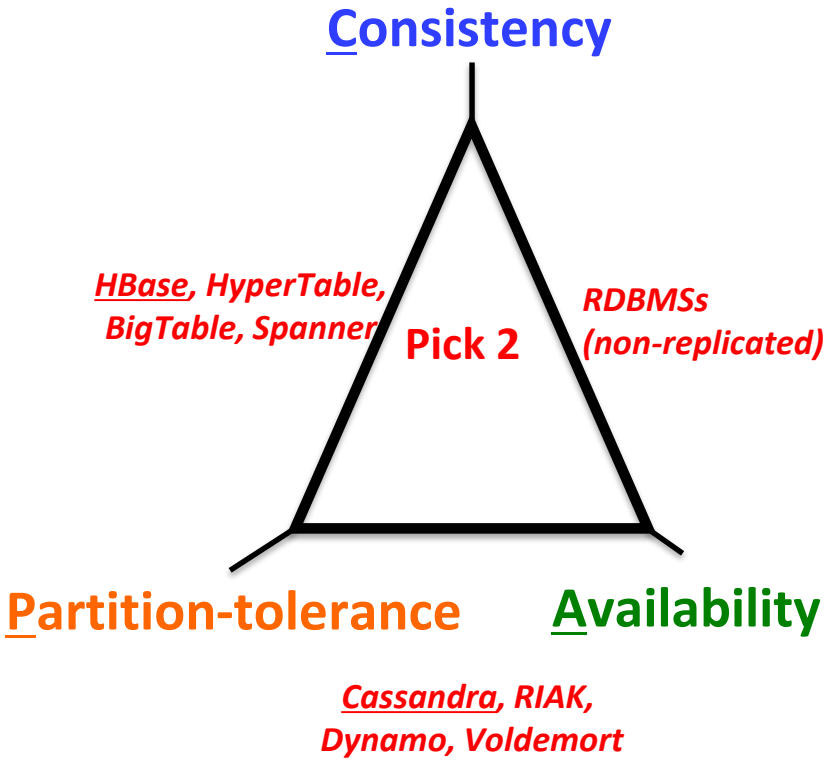
- Since partition-tolerance is essential in today's cloud computing systems, CAP theorem implies that a system has to choose between consistency and availability

- **Cassandra**
  - Eventual (weak) consistency, Availability, Partition-tolerance

- **Traditional RDBMSs**
  - Strong consistency over availability under a partition

CAP Tradeoff

- Starting point for NoSQL Revolution
- A distributed storage system can achieve **at most two of C, A, and P.**
- When partition-tolerance is important, you have to choose between consistency and availability



Eventual Consistency

- If all writes stop (to a key), then all its values (replicas) will converge eventually.
- If writes continue, then system always tries to keep converging.
  - **Moving “wave” of updated values lagging behind the latest values sent by clients, but always trying to catch up.**
- May still return stale values to clients (e.g., if many back-to-back writes).
- But works well when there a few periods of low writes – system converges quickly.

RDBMS vs. Key-value stores

- While RDBMS provide **ACID**
  - **A**tomicity
  - **C**onsistency
  - **I**solation
  - **D**urability

- Key-value stores like Cassandra provide **BASE**
  - **B**asically **A**vailable **S**oft-state **E**ventual Consistency
  - Prefers Availability over Consistency

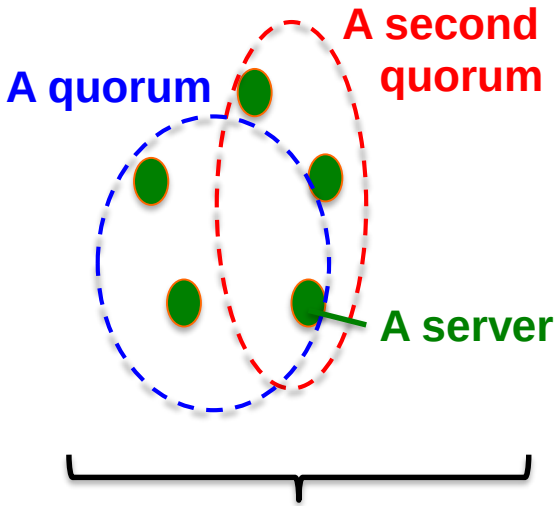
Consistency in Cassandra

- Cassandra has **consistency levels**
- Client is allowed to choose a consistency level for each operation (read/write)
  - **ANY:** any server (may not be replica)
    - Fastest: coordinator caches write and replies quickly to client
  - **ALL:** all replicas
    - Ensures strong consistency, but slowest
  - **ONE:** at least one replica
    - Faster than ALL, but cannot tolerate a failure
  - **QUORUM:** quorum across all replicas in all datacenters (DCs)
    - What?

Quorums for Consistency

In a nutshell:

- Quorum = majority
  - > 50%
- Any two quorums intersect
  - Client 1 does a write in red quorum
  - Then client 2 does read in blue quorum
- At least one server in blue quorum returns latest write
- Quorums faster than ALL, but still ensure strong consistency



Five replicas of a key-value pair

Quorums in Detail

- Several key-value/NoSQL stores (e.g., Riak and Cassandra) use quorums.
- **Reads**
  - Client specifies value of **R** (≤ N = total number of replicas of that key).
  - R = read consistency level.
  - Coordinator waits for R replicas to respond before sending result to client.
  - In background, coordinator checks for consistency of remaining (N-R) replicas, and initiates read repair if needed.

Quorums in Detail (Contd..)

- Writes come in two flavors
  - Client specifies W (≤ N)
  - W = write consistency level.
  - Client writes new value to W replicas and returns. Two flavors:
    - Coordinator blocks until quorum is reached.
    - Asynchronous: Just write and return.

Quorums in Detail (Contd.)

- R = read replica count, W = write replica count
- Two necessary conditions:
  1. **W+R > N**
  2. **W > N/2**
- Select values based on application
  - **(W=1, R=1):** very few writes and reads
  - **(W=N, R=1):** great for read-heavy workloads
  - **(W=N/2+1, R=N/2+1):** great for write-heavy workloads
  - **(W=1, R=N):** great for write-heavy workloads with mostly one client writing per key

Cassandra Consistency Levels (Contd.)

- Client is allowed to choose a consistency level for each operation (read/write)
  - ANY: any server (may not be replica)
    - Fastest: coordinator may cache write and reply quickly to client
  - ALL: all replicas
    - Slowest, but ensures strong consistency
  - ONE: at least one replica
    - Faster than ALL, and ensures durability without failures
  - **QUORUM:** quorum across all replicas in all datacenters (DCs)
    - Global consistency, but still fast
  - **LOCAL_QUORUM:** quorum in coordinator's DC
    - Faster: only waits for quorum in first DC client contacts
  - **EACH_QUORUM:** quorum in every DC
    - Lets each DC do its own quorum: supports hierarchical replies

Types of Consistency

- Cassandra offers **Eventual Consistency**

- Are there other types of weak consistency models?

Consistency Solutions

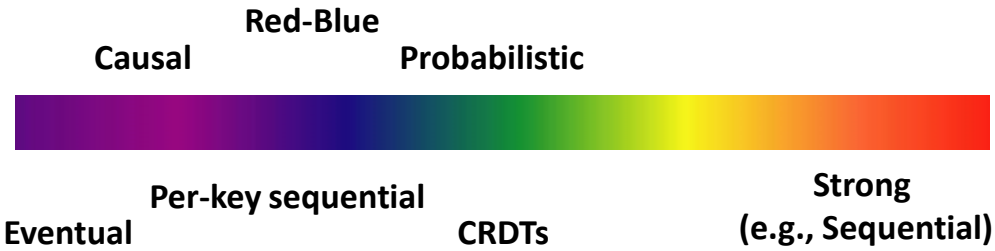
Consistency Solutions

Eventual Consistency

- Cassandra offers **Eventual Consistency**
  - If writes to a key stop, all replicas of key will converge
  - Originally from Amazon's Dynamo and LinkedIn's Voldemort systems

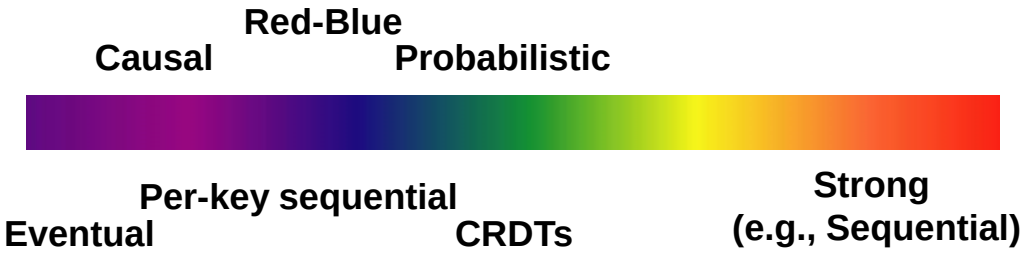
Newer Consistency Models

- Striving towards strong consistency
- While still trying to maintain high availability and partition-tolerance



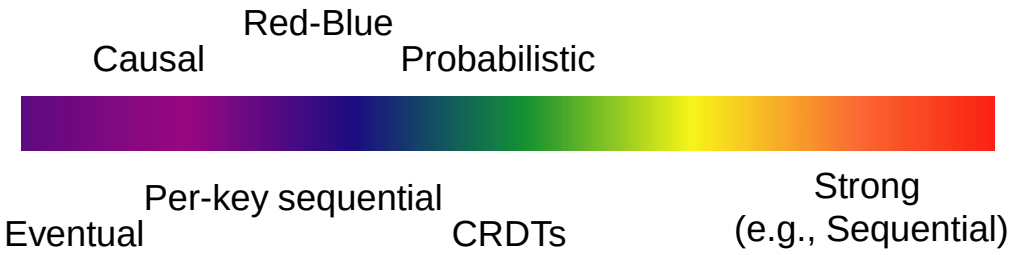
Newer Consistency Models (Contd.)

- **Per-key sequential**: Per key, all operations have a global order
- **CRDTs** (Commutative Replicated Data Types): Data structures for which commutated writes give same result [INRIA, France]
  - E.g., value == int, and only op allowed is +1
  - Effectively, servers don't need to worry about consistency



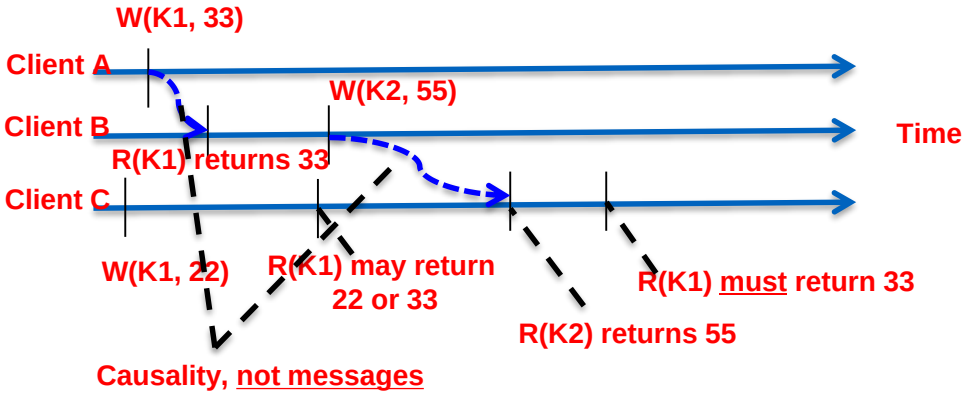
Newer Consistency Models (Contd.)

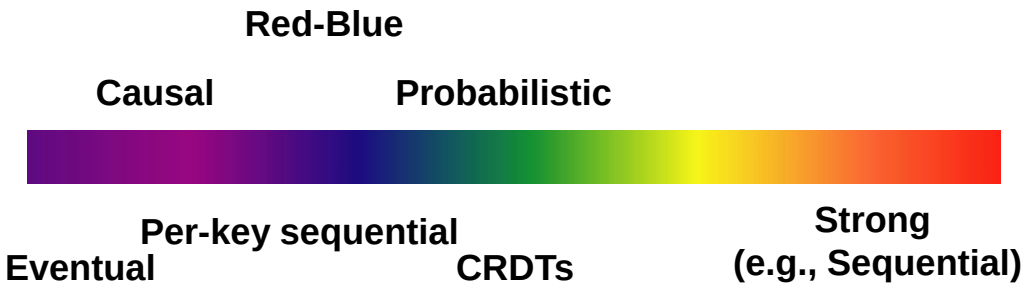
- **Red-blue Consistency**: Rewrite client transactions to separate operations into red operations vs. blue operations [MPI-SWS Germany]
  - Blue operations can be executed (commutated) in any order across DCs
  - Red operations need to be executed in the same order at each DC



Newer Consistency Models (Contd.)

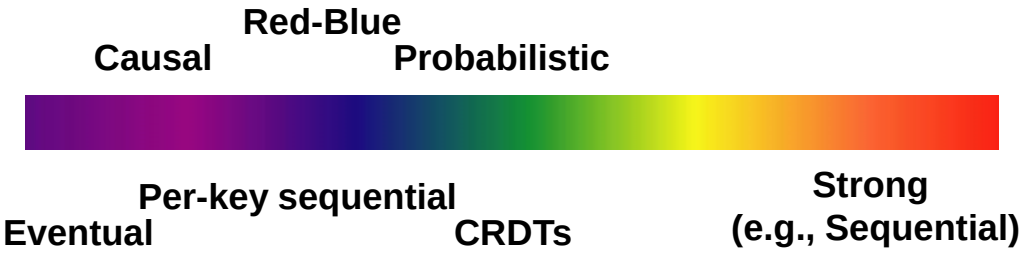
Causal Consistency: Reads must respect partial order based on information flow [Princeton, CMU]





Which Consistency Model should you use?

- Use the lowest consistency (to the left) consistency model that is “correct” for your application
  - Gets you fastest availability



Strong Consistency Models

- **Linearizability**: Each operation by a client is visible (or available) **instantaneously** to all other clients
  - Instantaneously in real time
- **Sequential Consistency** [Lamport]:
  - *"... the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program.*
  - After the fact, find a "reasonable" ordering of the operations (can re-order operations) that obeys sanity (consistency) at all clients, and across clients.
- Transaction ACID properties, example: newer key-value/NoSQL stores (sometimes called **"NewSQL")**
  - Hyperdex [Cornell]
  - Spanner [Google]
  - Transaction chains [Microsoft Research]

Conclusion

- Traditional Databases (RDBMSs) work with strong consistency, and offer ACID
- Modern workloads don't need such strong guarantees, but do need fast response times (availability)
- Unfortunately, CAP theorem
- **Key-value/NoSQL systems offer BASE** [**B**asically **A**vailable **S**oft-state **E**ventual Consistency]
  - Eventual consistency, and a variety of other consistency models striving towards strong consistency
- We have also discussed the design of Cassandra and different consistency solutions.

Thank You