



CLOUD COMPUTING APPLICATIONS

CLOUD COMPUTING INTRODUCTION

Roy Campbell & Reza Farivar

Tremendous Buzz

“Not only is it faster and more flexible, it is cheaper. [...] the emergence of cloud models radically alters the cost-benefit decision”

(FT)

“Cloud computing achieves a quicker return on investment”

(Lindsay Armstrong of salesforce.com)

“In an economic downturn, the appeal of that cost advantage will be greatly magnified”

(IDC)

“Revolution, the biggest upheaval since the invention of the PC in the 1970s [...] IT departments will have little left to do once the bulk of business computing shifts [...] into the cloud”

(Nicholas Carr)

“No less influential than e-business”

(Gartner)

“The economics are compelling, with business applications made three to five times cheaper and consumer applications five to 10 times cheaper”

(Merrill Lynch)

“Domestic cloud computing estimated to grow at 53%”

(moneycontrol.com)

Perils of Corporate Computing

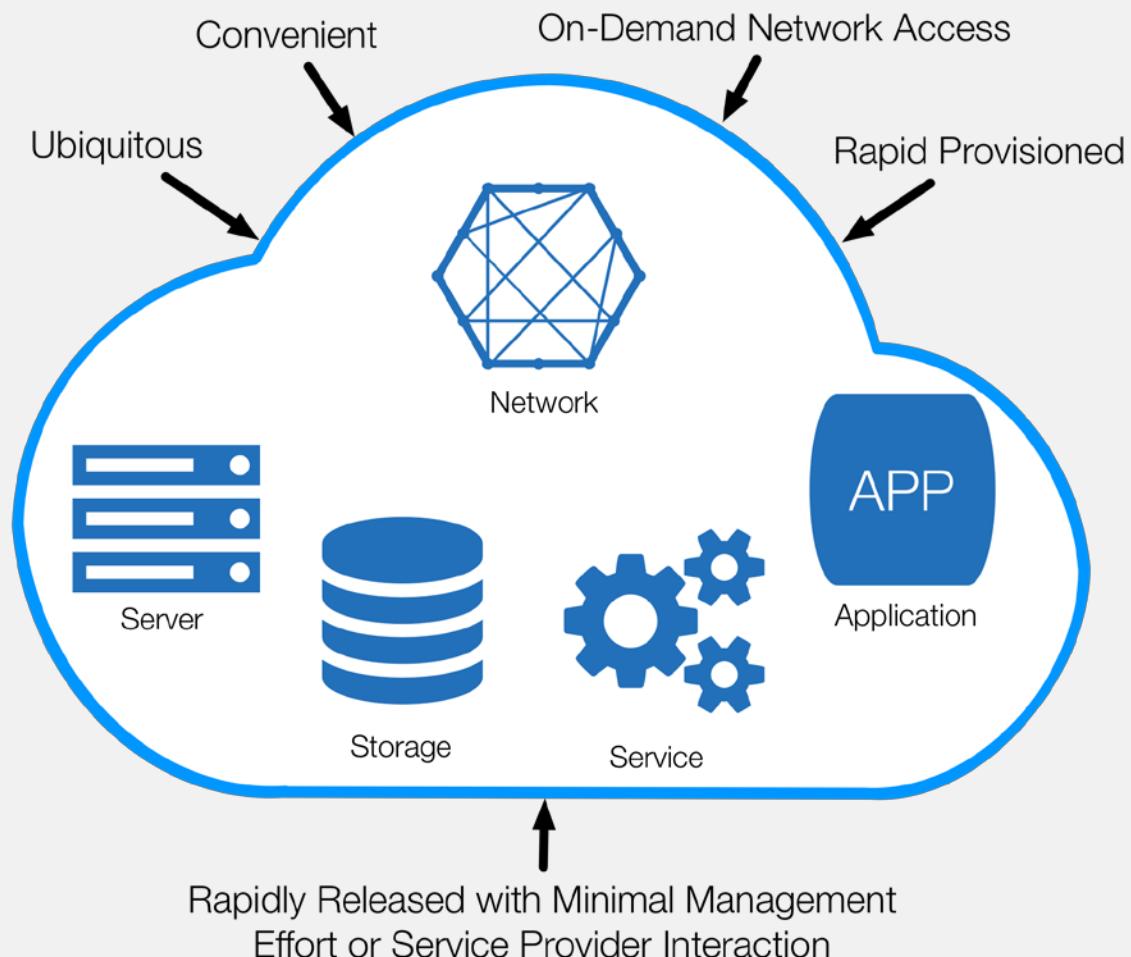
- Own information systems ☺
- However
 - Capital investment ☹
 - Heavy fixed costs ☹
 - Redundant expenditures ☹
 - High energy cost, low CPU utilization ☹
 - Dealing with unreliable hardware ☹
 - High levels of overcapacity (technology and labor) ☹
- NOT SUSTAINABLE

Back to the Future

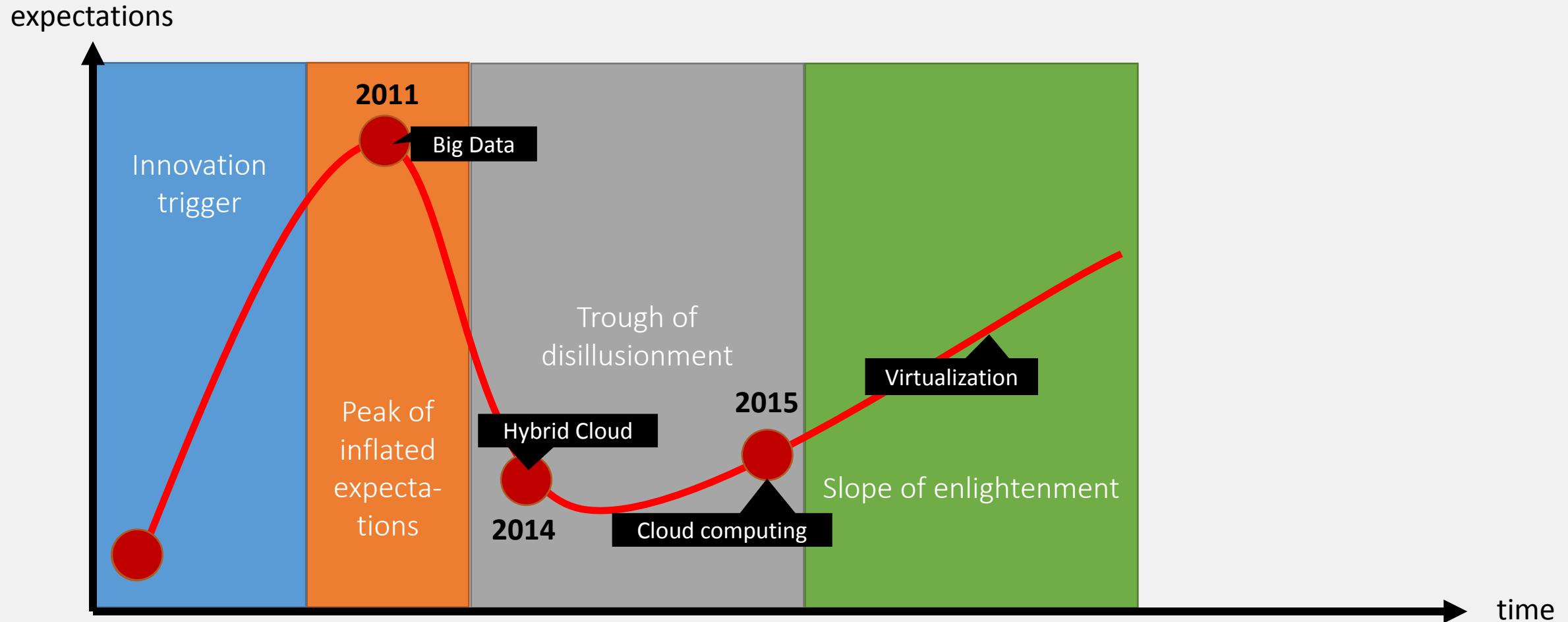
“Computing may someday be organized as a public utility, just as the telephone system is organized as a public utility”

(John McCarthy, 1961)

Cloud Computing



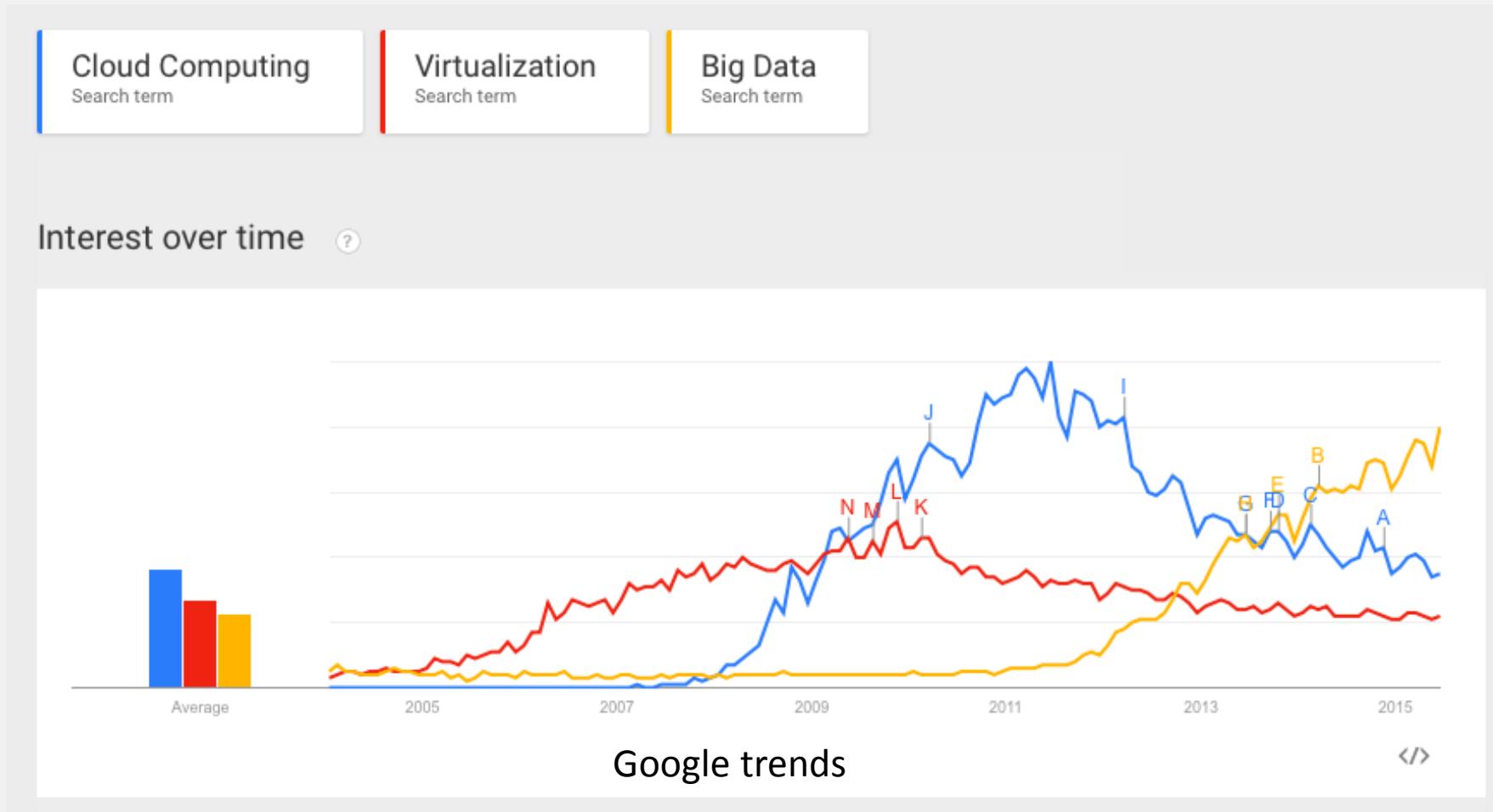
Cloud Adoption: Gartner's Hype Cycle



Delivery Models

- Software as a Service (SaaS)
 - Use provider's applications over a network
 - SalesForce.com
- Platform as a Service (PaaS)
 - Deploy customer-created applications to a cloud
 - AppEng
- Infrastructure as a Service (IaaS)
 - Rent processing, storage, network
 - Capacity and other fundamental computing resources
 - EC2, S3

Synergy: Cloud Computing, Virtualization, and Big Data



Big Data Data Revolution and Clouds

- Data collection too large to transmit economically over Internet
 - Petabyte data collections
- Computation is data intensive
 - Lots of disks, networks and CPUs
 - Overhead of maintaining cyber infrastructure is expensive
 - Users buy Big Data services from Clouds to share overhead
- Easy-to-write programs, fast turnaround
- MapReduce – Hadoop, PIG, HDFS, HBase



CLOUD COMPUTING APPLICATIONS

CLOUDONOMICS: PART 1

Roy Campbell & Reza Farivar

Cloudonomics: Part 1

Economics necessitates Cloud computing:

- Part 1: Utility Pricing
- Part 2: Benefits Common Infrastructure

See other details and benefits in

“Cloudonomics: A Rigorous Approach to Cloud Benefit Quantification,” Joe Weinman

https://www.csiac.org/sites/default/files/journal_files/stn14_4.pdf

Value of Utility Pricing

- Cloud services don't need to be cheaper to be economic!
- Consider a car
 - Buy or lease for \$10 per day
 - Rent a car for \$45 a day
 - If you need a car for 2 days in a trip, buying would be much more costly than renting
 - **It depends on the demand**

Utility Pricing in Detail

$D(t)$: demand for resources, $0 < t < T$

$P = \max(D(t))$: Peak Demand; $A = \text{Avg}(D(t))$: Average Demand

B = Baseline (owned) unit cost; $B_T = \text{Total Baseline Cost}$

C = Cloud unit cost; $C_T = \text{Total Cloud Cost}$

$U = C / B$: Utility Premium (for the rental car example, $U = 4.5$)

C_T

(because the Baseline should handle Peak Demand)

When is the Cloud cheaper than owning?

Substituting for C_T , B_T :

which implies

i.e., when Utility Premium is less than ratio of Peak Demand to Average Demand

Utility Pricing in Real World

- In practice, demands are often highly spiky
 - News stories, marketing promotions, product launches, Internet flash floods (Slashdot effect), tax season, Christmas shopping, etc.
- Often a hybrid model is the best
 - You own a car for daily commute, and rent a car when traveling or when you need a van to move
 - Key factor is again the ratio of Peak Demand to Average Demand
 - But we should also consider other costs
 - Network cost (both fixed costs and usage costs)
 - Interoperability overhead
 - Consider reliability, accessibility

Summary

- Utility Pricing is good when demand varies over time, as is the case of a start-up or a seasonal business
- When Utility Premium is less than ratio of Peak Demand to Average Demand, Cloud computing is beneficial
- Next, we look at the possible savings that Cloud providers can create using statistical multiplexing



CLOUD COMPUTING APPLICATIONS

CLOUDONOMICS: PART 2

Roy Campbell & Reza Farivar

Cloudonomics: Part 2

Economics necessitates Cloud Computing:

- Part 1: Utility Pricing
- Part 2: Benefits Common Infrastructure

See other details and benefits in

“Cloudonomics: A Rigorous Approach to Cloud Benefit Quantification,” Joe Weinman

https://www.csiac.org/sites/default/files/journal_files/stn14_4.pdf

The Value of Common Infrastructure

- For infrastructure built to peak requirements: Multiplexing demand → higher utilization
 - Lower cost per delivered resource than unconsolidated workloads
- For infrastructure built to less than peak: Multiplexing demand → reduce the unserved demand
 - Lower loss of revenue or a Service-Level agreement violation payout

A Useful Measure of “Smoothness”

The coefficient of variation:

$$C_v = \frac{\text{standard deviation } \sigma}{\text{mean } |\mu|}$$

C_v is a measure of smoothness

- small is smooth!
- large mean and/or smaller standard deviation

Implications of “Smoothness”

- A fixed-asset facility servicing highly variable jobs yields low utilization
- Same facility servicing smooth jobs yields high utilization
- **Multiplexing jobs with different distributions may reduce the coefficient of variation C_V**

Case Study of C_V for Independent Jobs

- X_1, X_n, \dots, X_n independent jobs with standard variation σ and mean μ
- Aggregated jobs
 - Mean \rightarrow sum of means: $n. \mu$
 - Variance \rightarrow sum of variances: $n.\sigma^2$
 - Aggregate $C_v \rightarrow \frac{\sqrt{n}.\sigma}{n.\mu} = \frac{\sigma}{\sqrt{n}.\mu} = \frac{1}{\sqrt{n}} C_v$

Case Study of C_V for Independent Jobs

Adding n independent jobs reduces C_V by $1/\sqrt{n}$

- Penalty of insufficient/excess resources grows smaller
- Aggregating 100 workloads brings the penalty to 10%

Case Study of C_V for Correlated Jobs

- Best Case: Negative correlation
 - Optimal packing of customer jobs
 - X and $1-X \rightarrow$ Sum is 1, $C_v = 0$
 - Optimally smooth, best CPU utilization
- Worst Case: Positive correlation
 - Mean: $n.\mu(X)$, standard deviation: $n.\sigma(X)$
 - Aggregate $C_v = C_v(X) = \frac{\sigma(X)}{\mu(X)}$
 - Which isn't smoother!

Results from Theory

- Negative-correlated jobs
 - Private, mid-size, and large-size providers can experience similar statistics of scale
- Independent jobs
 - Mid-size providers can achieve similar statistical economies to an infinitely large provider

Common Infrastructure in Real World

- Available data on economy of scale for large providers is mixed
 - Use the same COTS computers and components
 - Locate near cheap power supplies → everyone can do that
 - Early entrant automation tools → 3rd parties take care of it
- Takeaway lesson: you don't need to be as large as Amazon.com to compete! ☺
 - At least according to "Value of Common Infrastructure"



CLOUD COMPUTING APPLICATIONS

BIG DATA

Roy Campbell & Reza Farivar

Big Data (a Singular Phrase)!

- A collection of data sets so large and complex, it's impossible to process it on one computer with the usual databases and tools
- Because of its size and complexity, Big Data is hard to capture, store, copy, delete (privacy), search, share, analyze, and visualize

Big Data

Big Data represents the information assets characterized by such high

- Volume,
- Velocity, and
- Variety

as to require specific technology and analytical methods for its transformation into

- Value

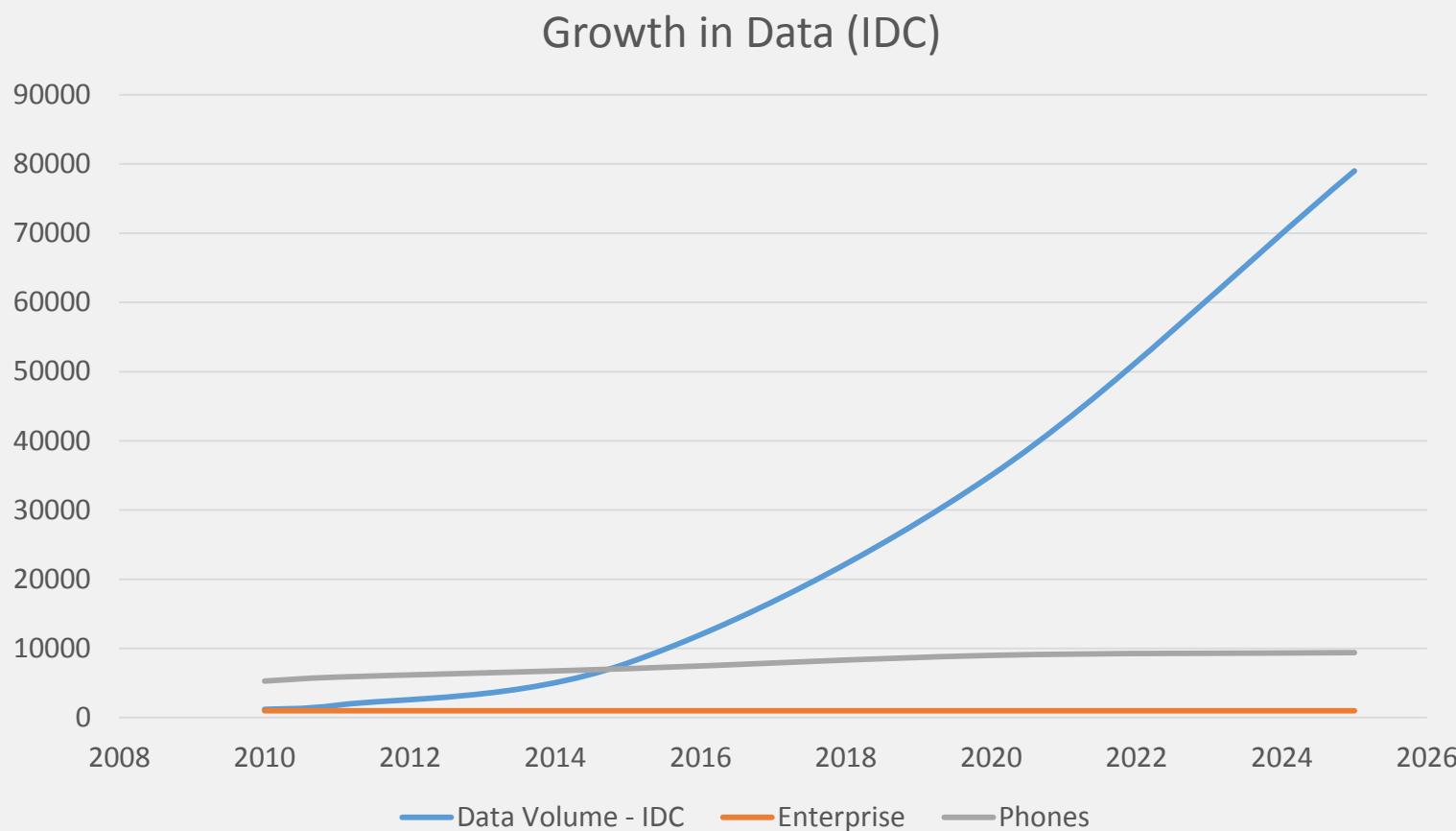
Challenges

Mobiles
Sensing
Logs
Cameras
RFID
Social Nets
Telescopes
Medical

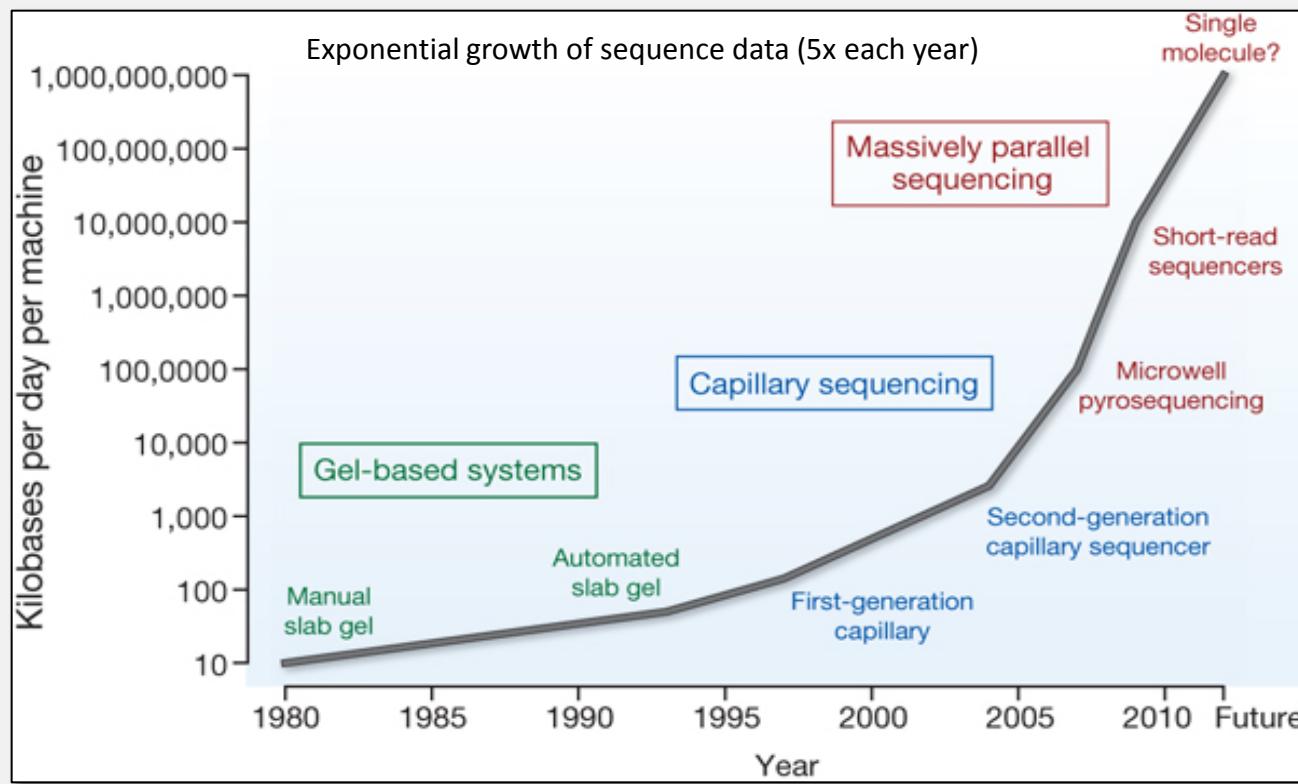
Analysis,
Capture,
Data curation,
Search,
Sharing,
Storage,
Transfer,
Visualization,
Information privacy

Predictive
analytics,
Better decision
making,
Discovery

Timeline

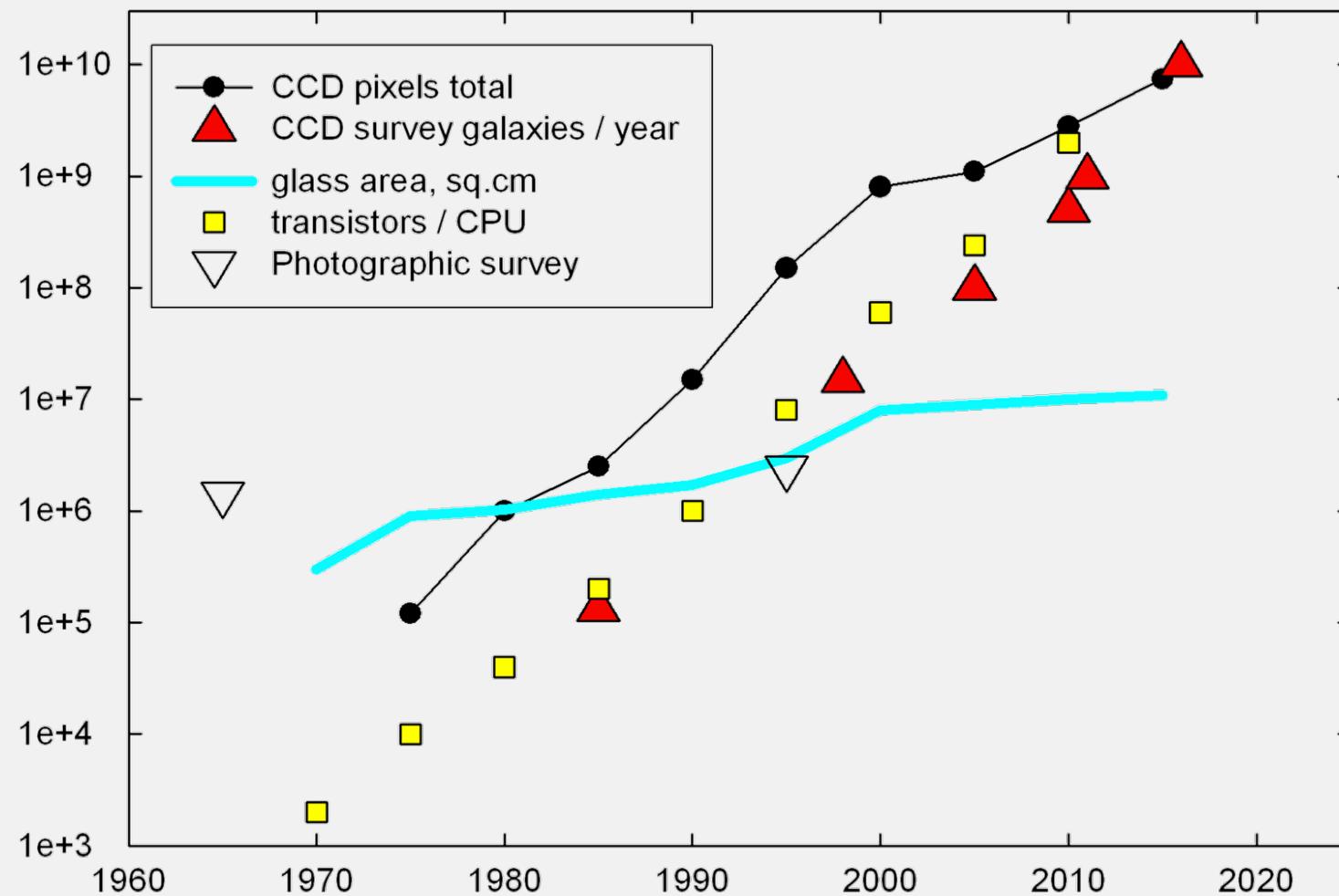


Example: Bioinformatics

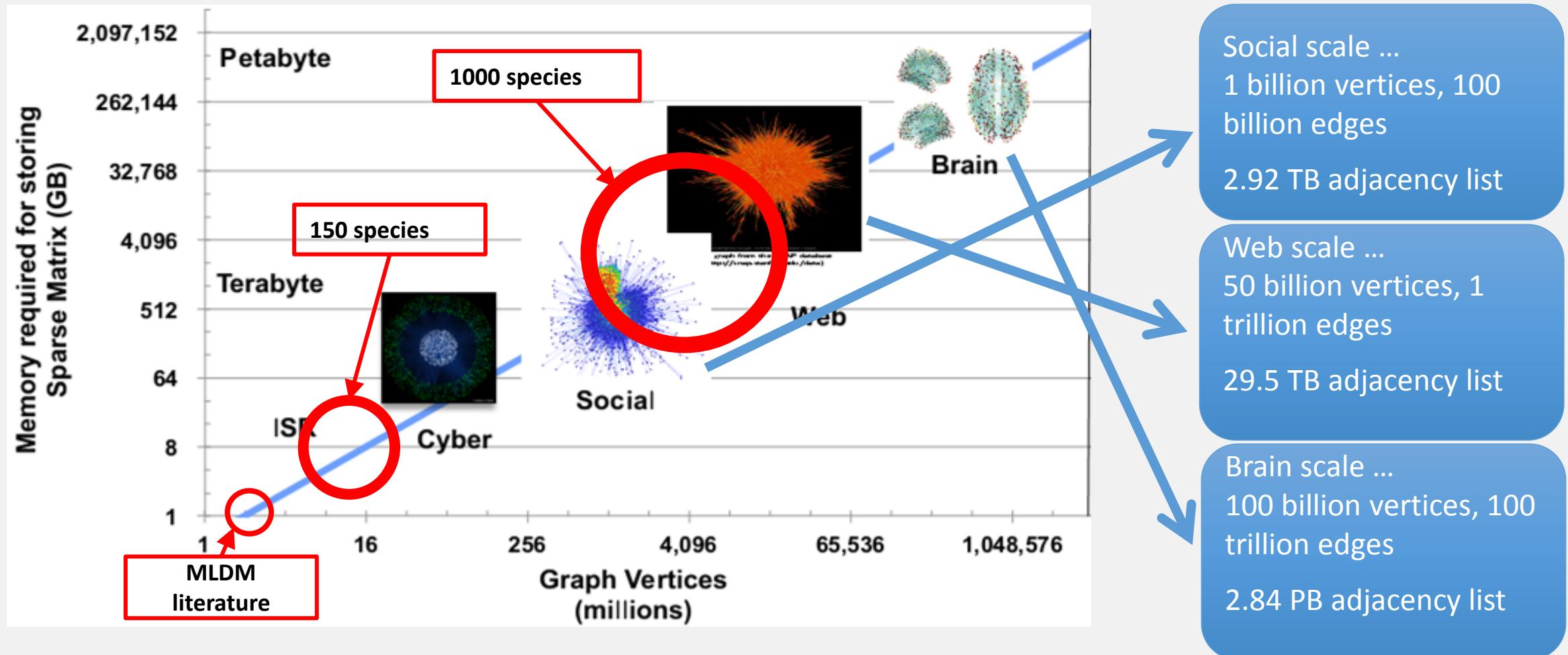


Michael R. Stratton, Peter J. Campbell & P. Andrew Futreal
Nature 458, 719-724(9 April 2009)

Example: Astronomy

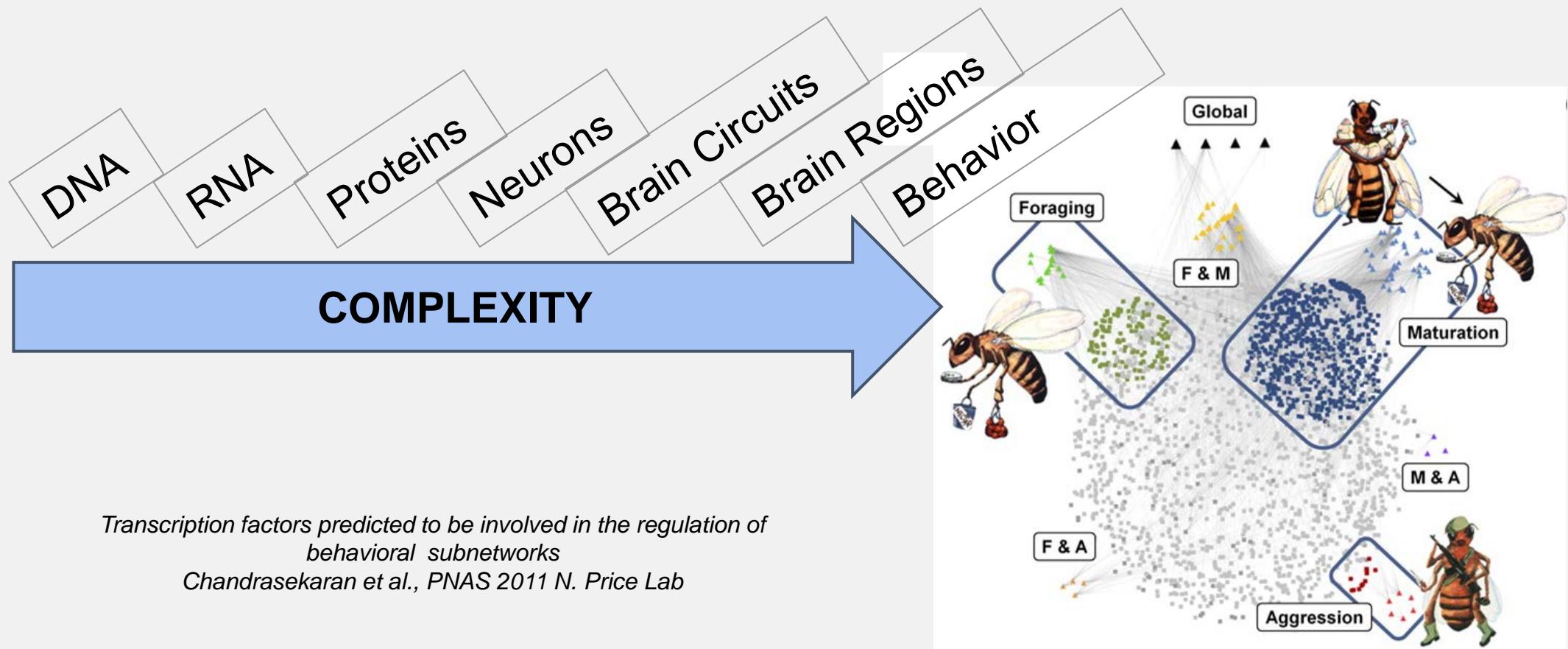


Example: Graphs

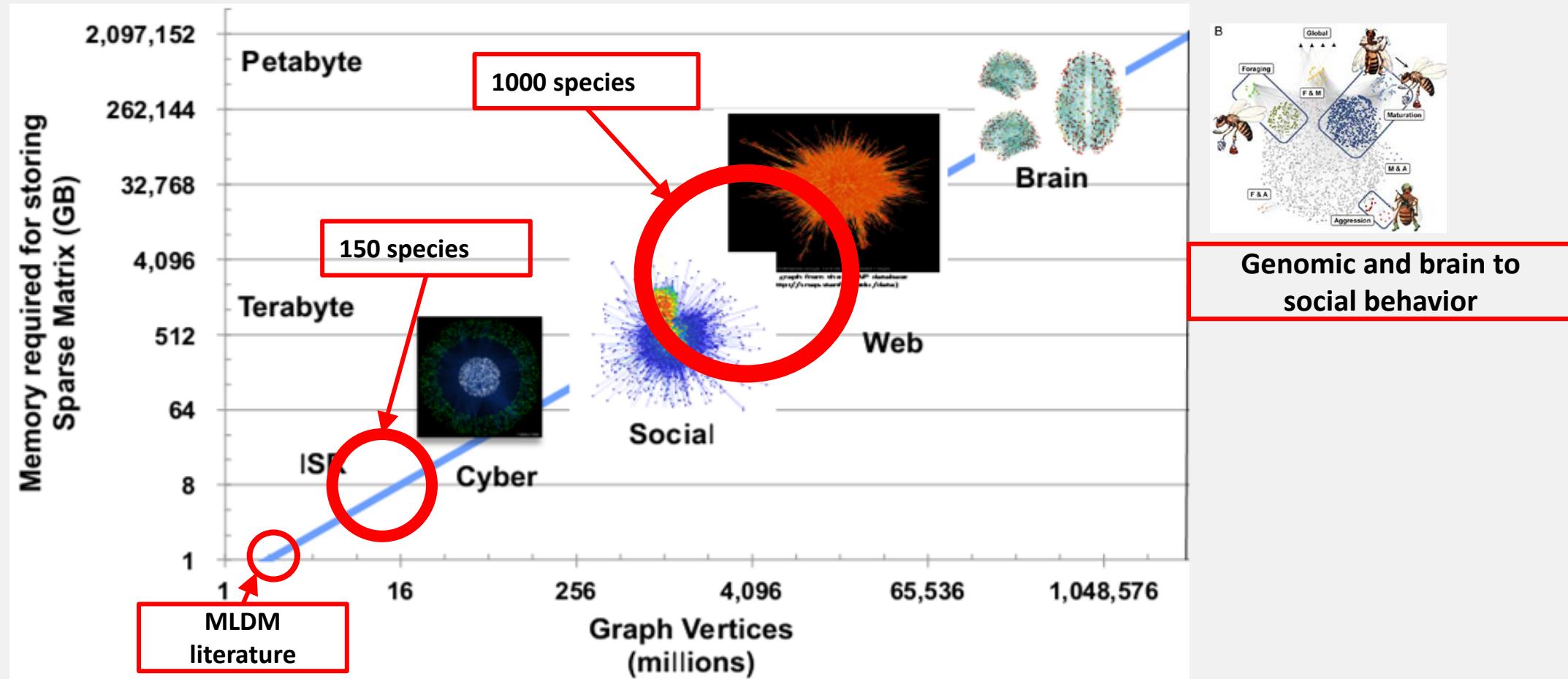


Future: Scale of Real-World Graphs

From genes to brains and social behavior...



Scale of Real-World Graphs: Today





CLOUD COMPUTING APPLICATIONS

SUMMARY OF CLOUD INTRODUCTION

Roy Campbell & Reza Farivar

Takeaways

- Multiple reasons for Cloud adoption
- Clouds allow economies
- Sharing for the Big Data revolution?
- Another round in innovation: everyone benefits from Clouds?

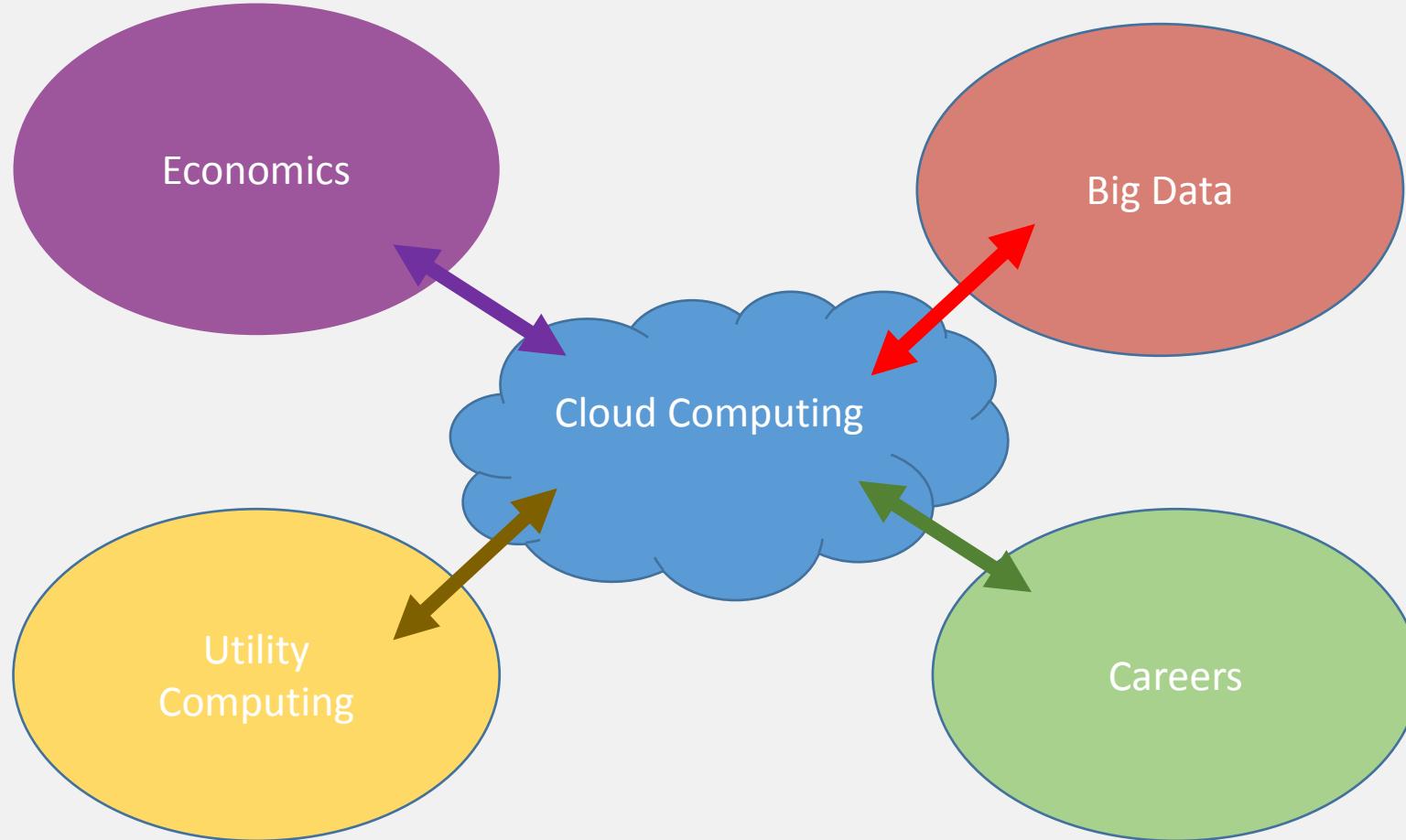
Careers

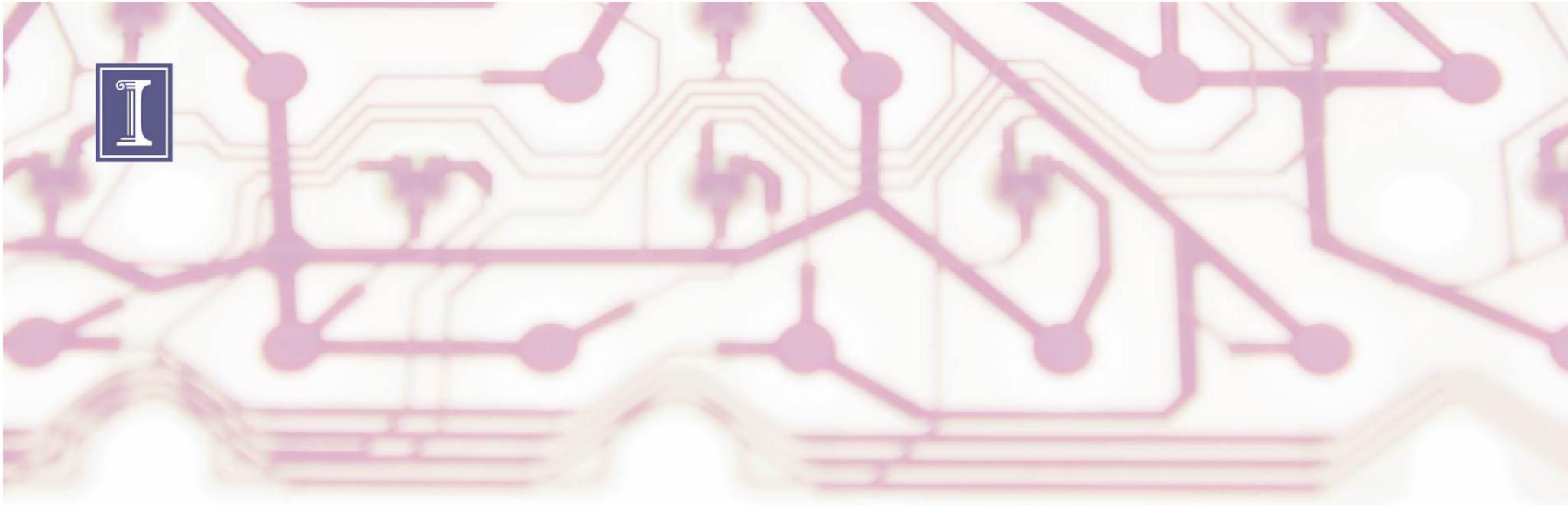
- Almost all people will use Cloud computing: search, video streaming, social networking
- IT specialists need to know Cloud
 - When to outsource, applications, uses
 - Architectures, sharing, privacy
 - Programming, efficiency, parallelism, scale
- General audience
 - Huge impact on society, business, government
 - New capabilities, policies, ways to communicate
 - Privacy and security

First Exercise

- Show scalability of web service
- Demonstrate user application
- Software-defined architecture
 - Management controls
 - Billing
 - Software modules available
- Distributed computing
 - Reliability, parallelism
- Provides insight to the power of sharing data

Summary





CLOUD COMPUTING APPLICATIONS

Infrastructure as a Service

Reza Farivar

Infrastructure as a Service

- The most fundamental of Cloud Computing Models
- Allows the user to “rent” computing resources
- The product is a “virtual” computer, that you can access remotely and do whatever you want
 - From the choice of the I/O specs for attached “hard drives” to the Operating System, middleware and applications
 - Network Connection
 - You are now responsible for managing everything running on the machine, including security of your server
- These resources are usually virtualized

Virtualized Resources

- Different customers have different needs
 - Ephemeral needs
- The Cloud Provider cannot operate a pool of many different sized computers
- Solution: Cloud provider operates a fleet of similar, powerful, hardware
- Carve out chunks of resources through virtualization
 - CPU
 - Memory
 - Storage
 - Network
 - Accelerators

Virtualized Resources

- Different customers have different needs
 - Ephemeral needs
- The Cloud Provider cannot operate a pool of many different sized computers
- Solution: Cloud provider operates a fleet of similar, powerful, hardware
- Carve out chunks of resources through virtualization: VM Instance
 - CPU
 - Memory
 - Storage
 - Network
 - Accelerators
- Metal as a Service (MaaS)

Dedicated Host SKUs (VM series and Host Type)	Available vCPUs	Available RAM	CPU
Dasv4_Type1	96	768 GiB	2.35 GHz AMD EPYC™ 7452
Ddsv4_Type1	80	504 GiB	Intel® Xeon® Platinum 8272CL (Cascade Lake)
Dsv4_Type1	80	504 GiB	Intel® Xeon® Platinum 8272CL (Cascade Lake)
Dsv3_Type1	64	256 GiB	2.3 GHz Intel® Xeon® E5-2673 v4 (Broadwell)
Dsv3_Type2	76	504 GiB	Intel® Xeon® Platinum 8171M (Skylake)
Esv3_Type2	76	504 GiB	Intel® Xeon® Platinum 8171M (Skylake)
Esv3_Type3	80	504 GiB	Intel® Xeon® Platinum 8272CL (Cascade Lake)
Fsv2_Type2	72	144 GiB	Intel® Xeon® Platinum 8168 (Skylake)
Fsv2_Type3	86	504 GiB	Intel® Xeon® Platinum 8272CL (Cascade Lake)
Lsv2_Type1	80	640 GiB	2.55 GHz AMD EPYC™ 7551
Ms_Type1	128	2,048 GiB	Intel® Xeon® Platinum 8280 (Cascade Lake)
Msm_Type1	128	3,892 GiB	Intel® Xeon® Platinum 8280 (Cascade Lake)
Msmv2_Type1	416	11,400 GiB	Intel® Xeon® Platinum 8180M (Skylake)

Virtualized Resources

- Different customers have different needs
 - Ephemeral needs
- The Cloud Provider cannot operate a pool of many different sized computers
- Solution: Cloud provider operates a fleet of similar, powerful, hardware
- Carve out chunks of resources through virtualization: VM Instance
 - CPU
 - Memory
 - Storage
 - Network
 - Accelerators
- Metal as a Service (MaaS)



Advantages of IaaS vs. On-Prem

- No need to run a data center
 - No worries about space, power supplies, physical building security, network, failing components, ...
- OpEx vs. CapEx
- Use different instances when needed
 - Rapid innovation
 - Quick response to shifting business conditions

IaaS Examples

- Microsoft Azure
- Amazon EC2 (Elastic Compute Cloud)
- Google Cloud Platform Compute Engine
- Oracle Cloud
- IBM Cloud
- Alibaba Cloud
- Rackspace
- Vultr
- ...

The screenshot shows the IBM Cloud interface for creating a Virtual server instance. The top navigation bar includes links for Catalog, Cost Estimator, Docs, and Sign up now. The main content area is titled "Virtual server instance" and describes it as "Delivers rapid scalability with pre-defined sizes that get you up and running quickly." A "Preview mode" section allows users to explore the offering. Below this, there are tabs for Type of virtual server: Public (Multi-tenant), Dedicated (Single-tenant), Transient (Multi-tenant Ephemeris), and Reserved (Multi-tenant Term commitment). The "Public instance" tab is selected, showing options for Quantity (1), Billing (Hourly), and Location (NA West, NA South, NA East, South America, Europe, Asia-Pacific). The "Profile" section shows a selected profile: "Balanced | B1.2x4" with 2 vCPU, 4 GB RAM, SAN storage type, and a price of \$0.085 per hour. The "Image" section lists several operating system options: CentOS 8.x Minimal (64 bit) - HVM, Debian 9.x Minimal Stable (64 bit) - HVM, Red Hat 8.x Minimal (64 bit) - HVM, and Microsoft 2019 Standard (64 bit) - HVM. The "Attached storage disks" section shows a single disk entry: "Boot disk" (SAN, 25 GB (\$0.000)). The "Network interface" section includes fields for Uplink port speed (100 Mbps rate-limited public & private network uplinks [\$0.000]) and Public egress - bandwidth** (0 GB [\$0.000]). The "Private security group" and "Public security group" sections allow users to search for security groups.

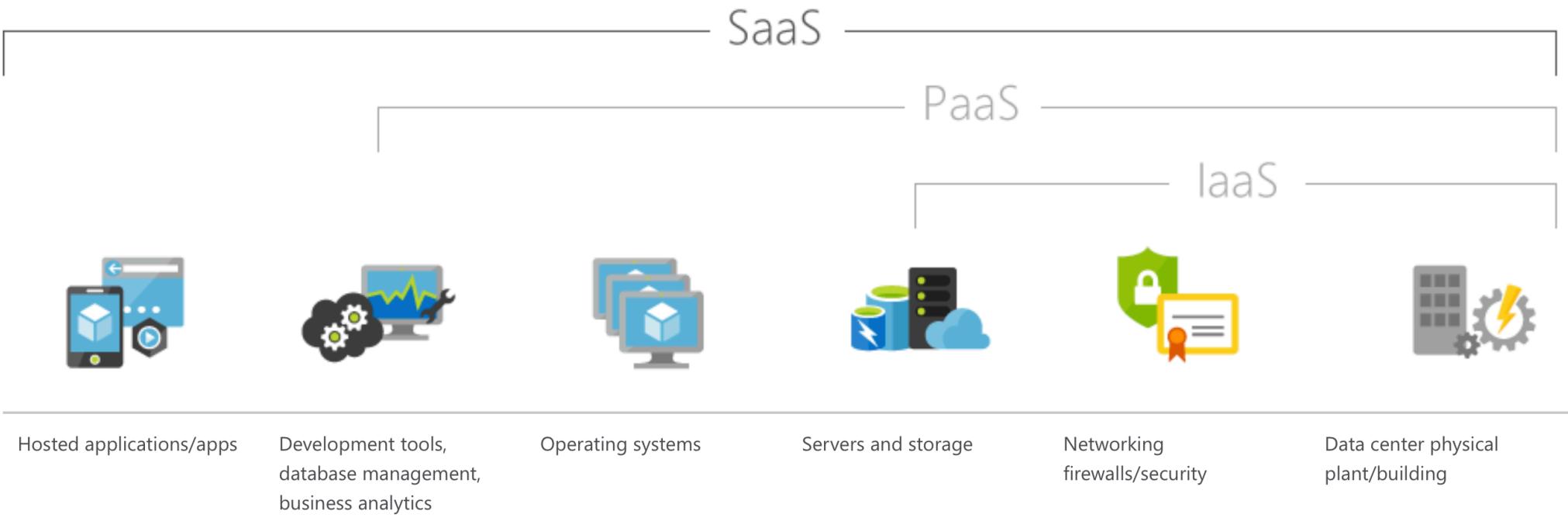
Instance Pricing

- On-Demand
- Reserved
- Spot Pricing

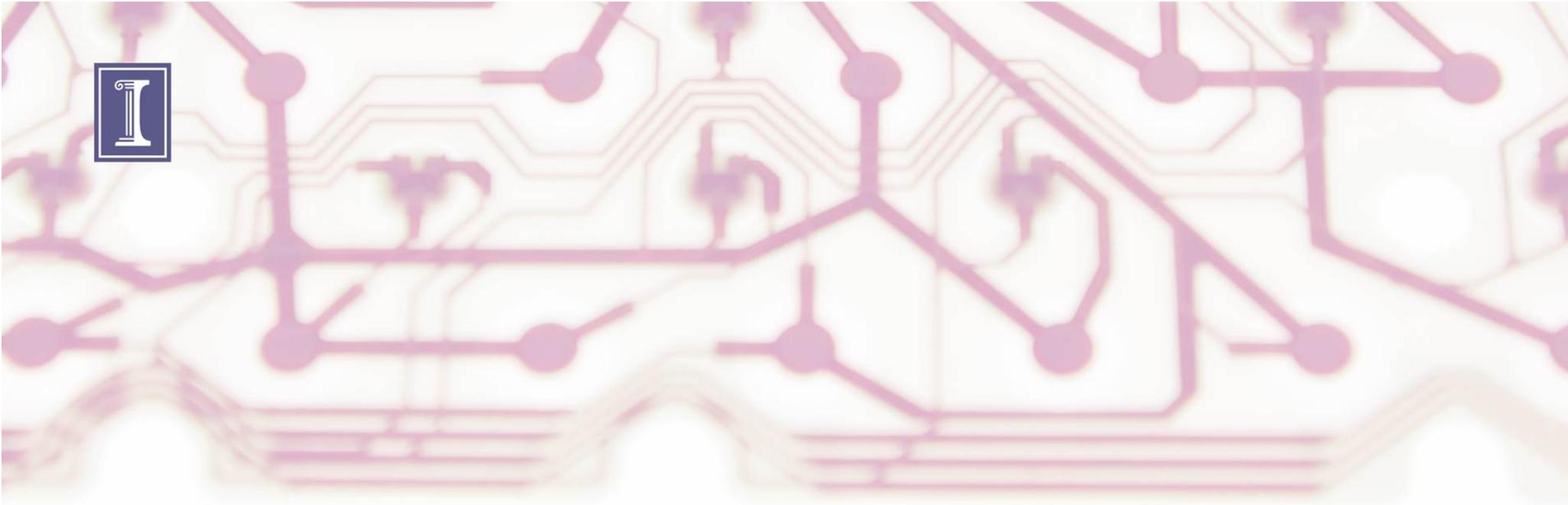
IaaS Sub-Category: Containers and Orchestration

- A subcategory of IaaS, or a place half-way between IaaS and PaaS (more towards the IaaS)
- You may think of a container as a light-weight Virtual Machine
 - Time to spin up a VM is tens of seconds to a few minutes
 - Time to start a container is fraction of a second to a few seconds
- Linux-Only

SaaS in Perspective



* Image courtesy of Microsoft Azure



CLOUD COMPUTING APPLICATIONS

Infrastructure as a Service: Regions and Zones

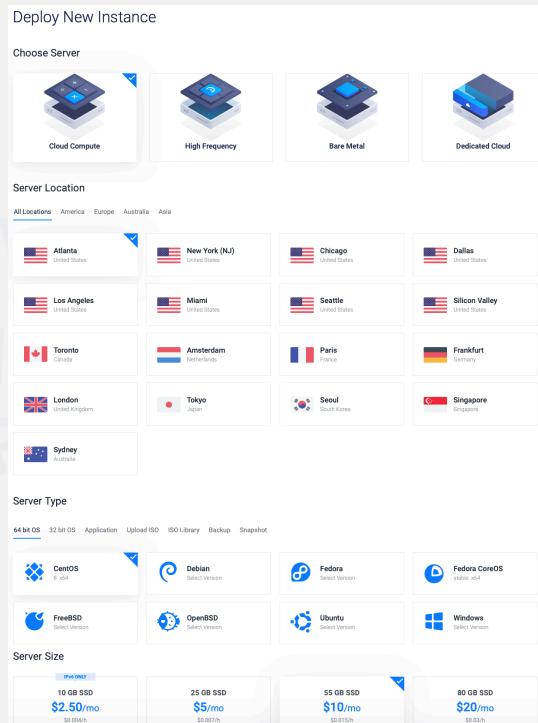
Reza Farivar

Virtual Machine Instance Location

- The cloud provider has multiple physical data centers, all over the globe
- Where does your virtual machine reside?
- Regions
- Availability Zones / Zones

Data Center Location

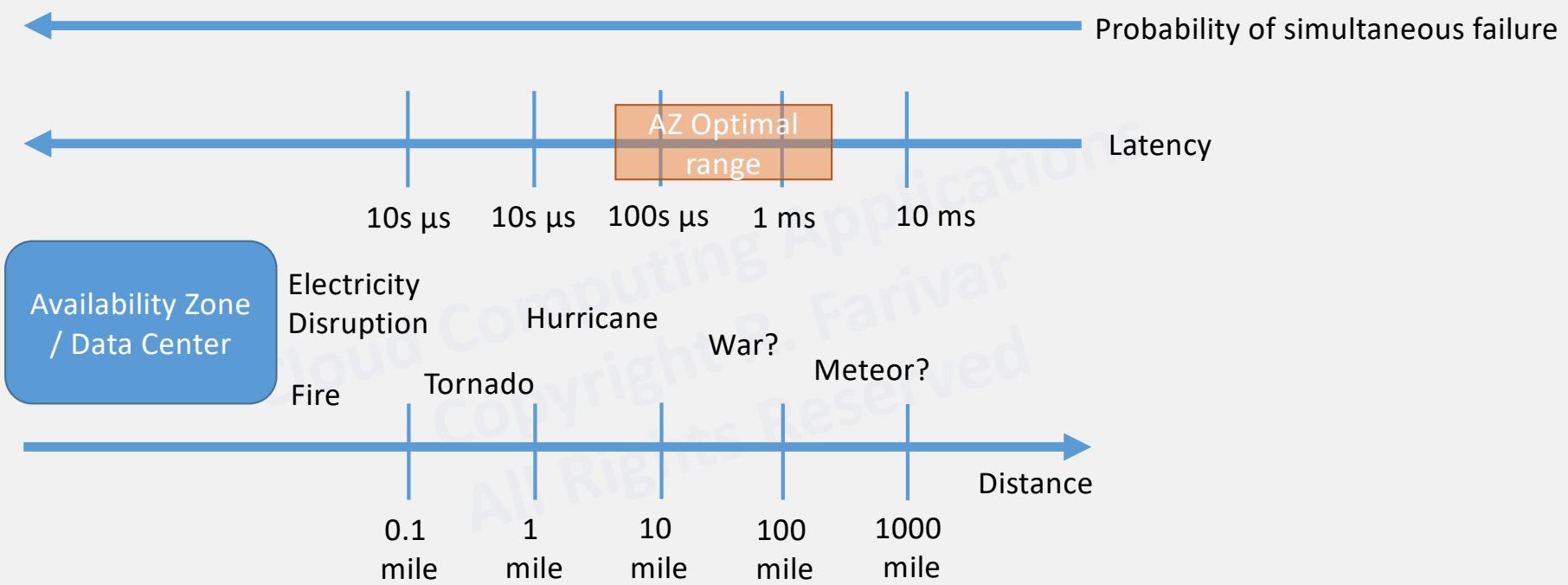
- Some Cloud Providers simply let you choose the Data Center
- E.g. Vultr:



Availability Zones and Regions



Availability Zones Locations



Availability Zones and Regions

- Typically each user ID gets access to a handful of availability zones to launch VM instances per each region
 - AWS: us-east-1 → us-east-1a, us-east-1b, ..., us-east-1f
 - Azure: US East → 1, 2, 3
 - GCP: us-east1 → us-east1-b, us-east1-c, us-east1-c
- The availability zone “a” for user1 is NOT the same as availability zone “a” for user2

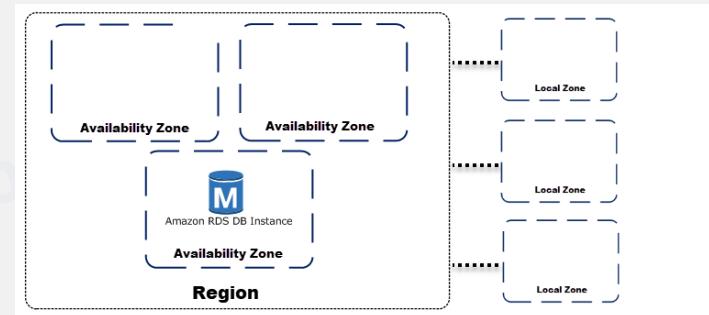
SLA for Virtual Machines

Last updated: July 2020

- For all Virtual Machines that have two or more instances deployed across two or more Availability Zones in the same Azure region, we guarantee you will have Virtual Machine Connectivity to at least one instance at least 99.99% of the time.
- For all Virtual Machines that have two or more instances deployed in the same Availability Set or in the same Dedicated Host Group, we guarantee you will have Virtual Machine Connectivity to at least one instance at least 99.95% of the time.
- For any Single Instance Virtual Machine using Premium SSD or Ultra Disk for all Operating System Disks and Data Disks, we guarantee you will have Virtual Machine Connectivity of at least 99.9%.
- For any Single Instance Virtual Machine using Standard SSD Managed Disks for Operating System Disk and Data Disks, we guarantee you will have Virtual Machine Connectivity of at least 99.5%.
- For any Single Instance Virtual Machine using Standard HDD Managed Disks for Operating System Disks and Data Disks, we guarantee you will have Virtual Machine Connectivity of at least 95%.

Availability Zones and Regions

- Each zone is made up of **one or more datacenters** equipped with independent power, cooling, and networking.
 - Chance of simultaneous failure in all the separate AZs in a region is extremely small
- *Local zones: A *Local Zone* is an extension of a Region that is geographically close to your users



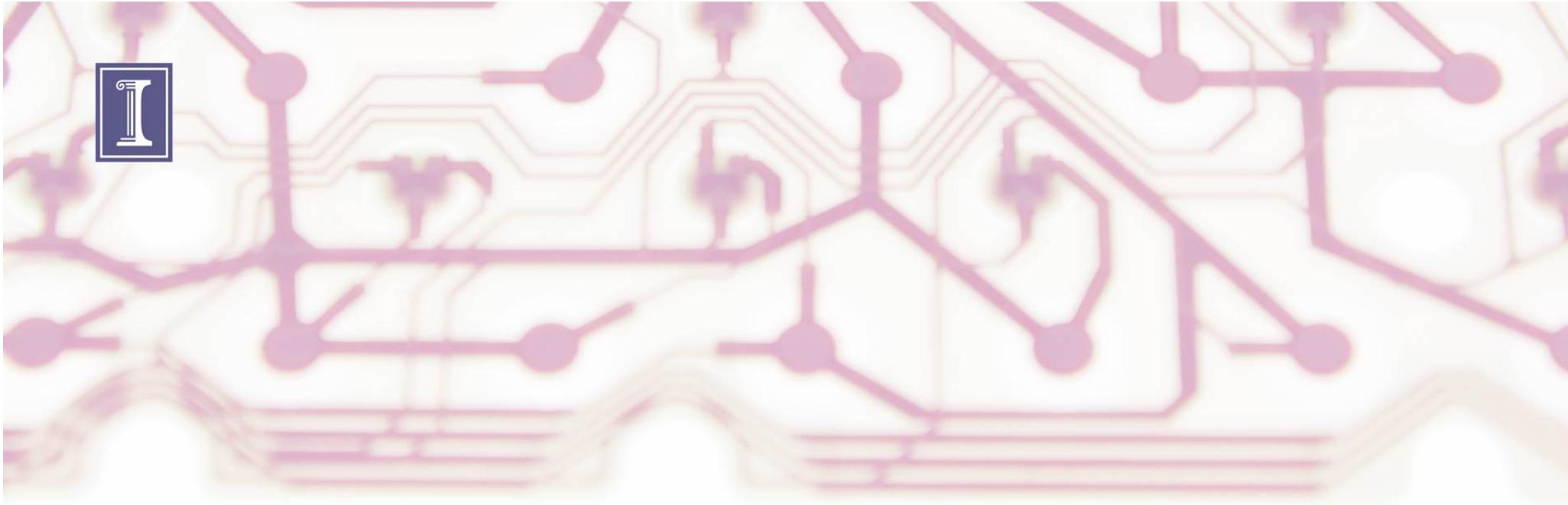
Availability Zones and Regions

- For high availability, design your system to have instances running in multiple AZ in a region, and maybe even have more than one region
- Data traffic costs most from one region to another, then from one AZ to another in the same region, and it is cheapest (or free) in the same AZ

SLA for Virtual Machines

Last updated: July 2020

- For all Virtual Machines that have two or more instances deployed across two or more Availability Zones in the same Azure region, we guarantee you will have Virtual Machine Connectivity to at least one instance at least 99.99% of the time.
- For all Virtual Machines that have two or more instances deployed in the same Availability Set or in the same Dedicated Host Group, we guarantee you will have Virtual Machine Connectivity to at least one instance at least 99.95% of the time.
- For any Single Instance Virtual Machine using Premium SSD or Ultra Disk for all Operating System Disks and Data Disks, we guarantee you will have Virtual Machine Connectivity of at least 99.9%.
- For any Single Instance Virtual Machine using Standard SSD Managed Disks for Operating System Disk and Data Disks, we guarantee you will have Virtual Machine Connectivity of at least 99.5%.
- For any Single Instance Virtual Machine using Standard HDD Managed Disks for Operating System Disks and Data Disks, we guarantee you will have Virtual Machine Connectivity of at least 95%.



CLOUD COMPUTING APPLICATIONS

Platform as a Service

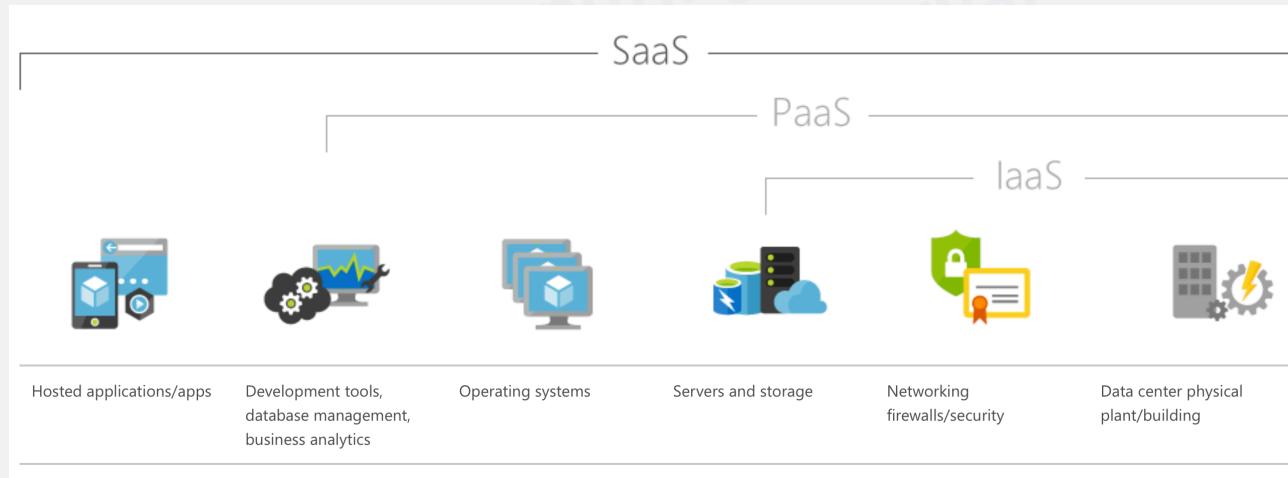
Reza Farivar

Platform as a Service

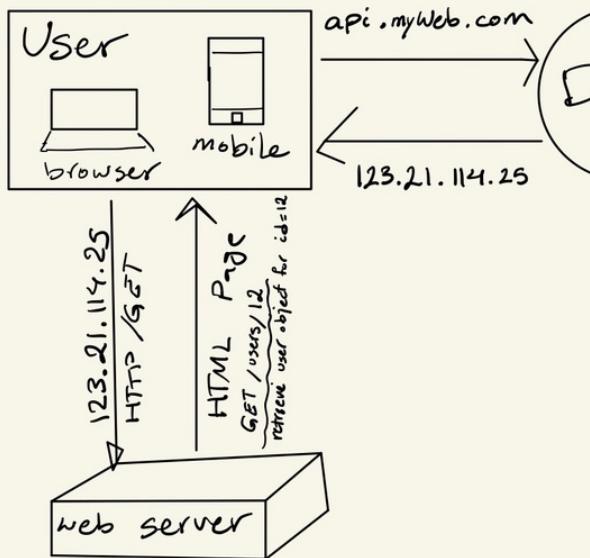
- The main goal is to run the user's distributed web service in a managed environment
 - Much more opinionated than IaaS

Platform as a Service

- The main goal is to run the user's distributed web service in a managed environment
 - Much more opinionated than IaaS

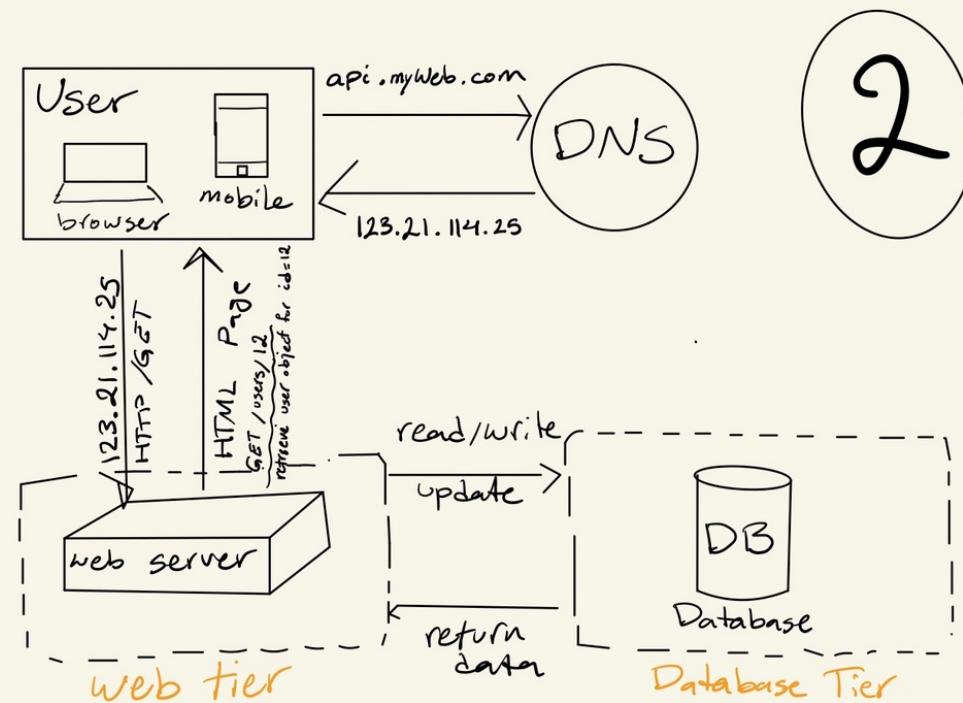


Anatomy of a Web Service Application

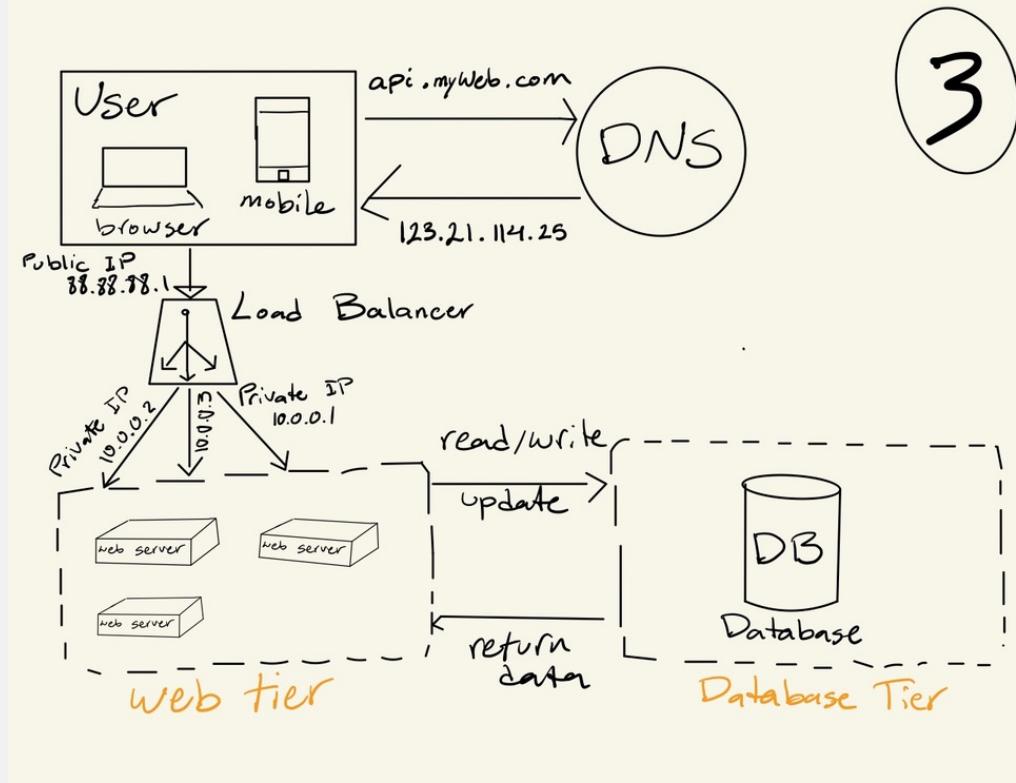


!
GET /users/12
{
 "id": 12,
 "firstName": "John",
 "lastName": "Doe",
 "address": {
 "streetAddress": "123 main st."
 },
 "phoneNumbers": [
 "217-123-1234",
 "800-123-1234"
]
}

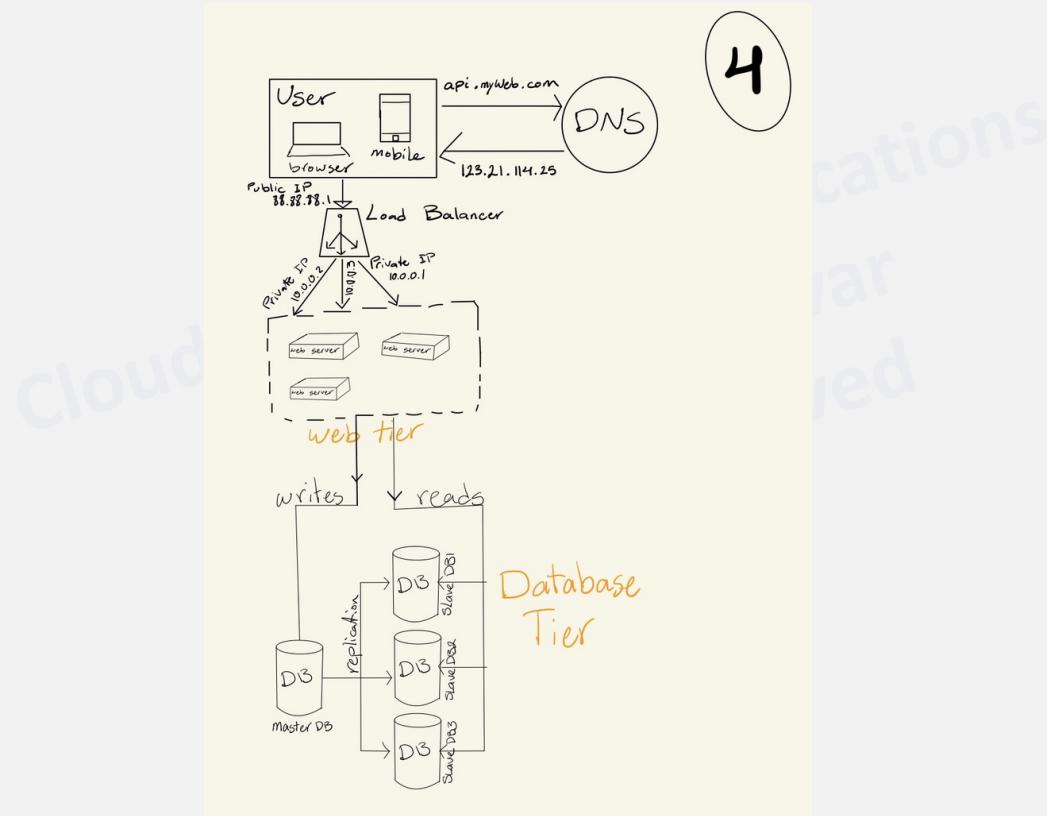
Anatomy of a Web Service Application



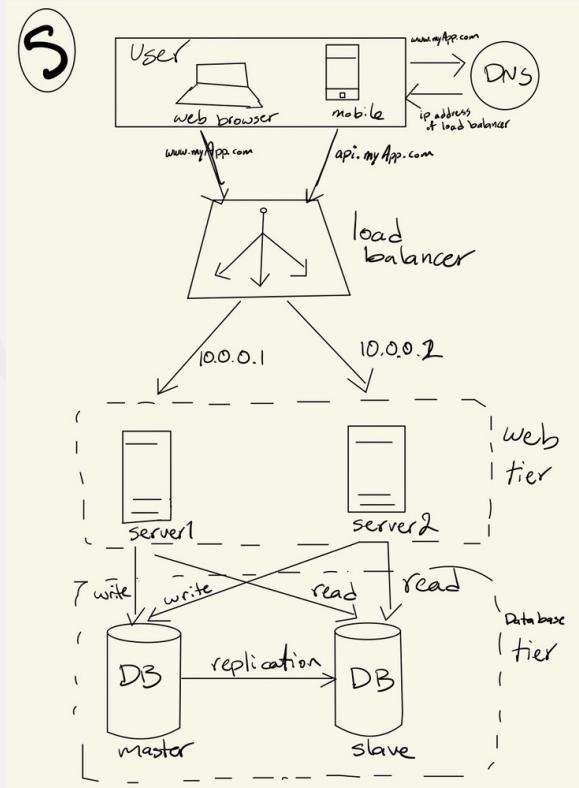
Anatomy of a Web Service Application



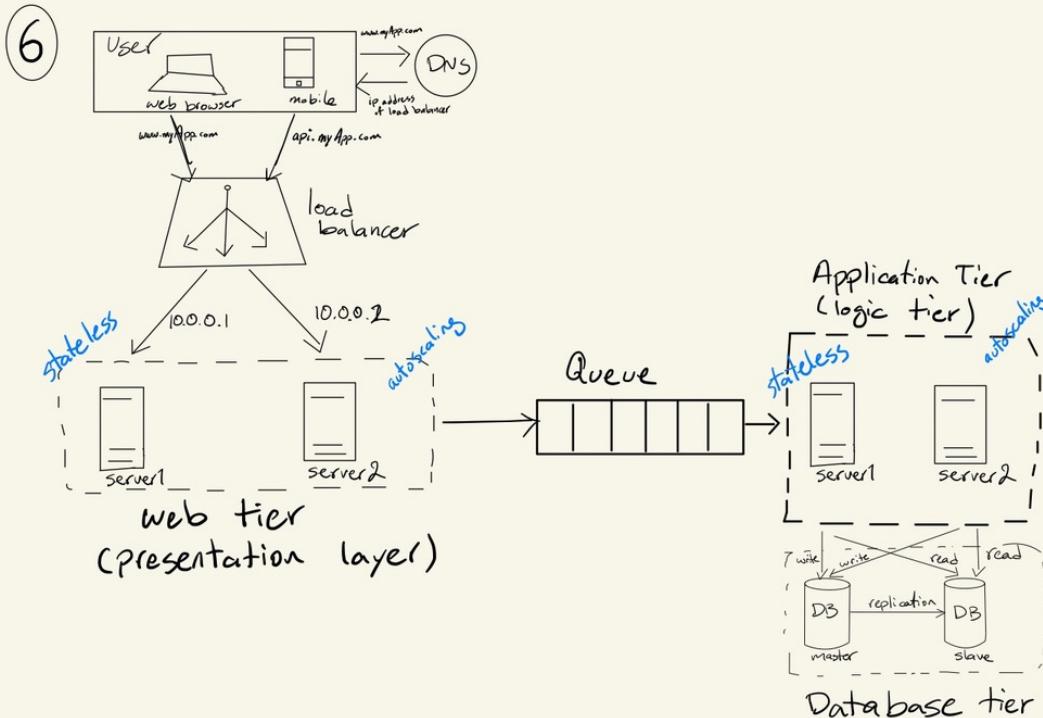
Anatomy of a Web Service Application



Anatomy of a Web Service Application



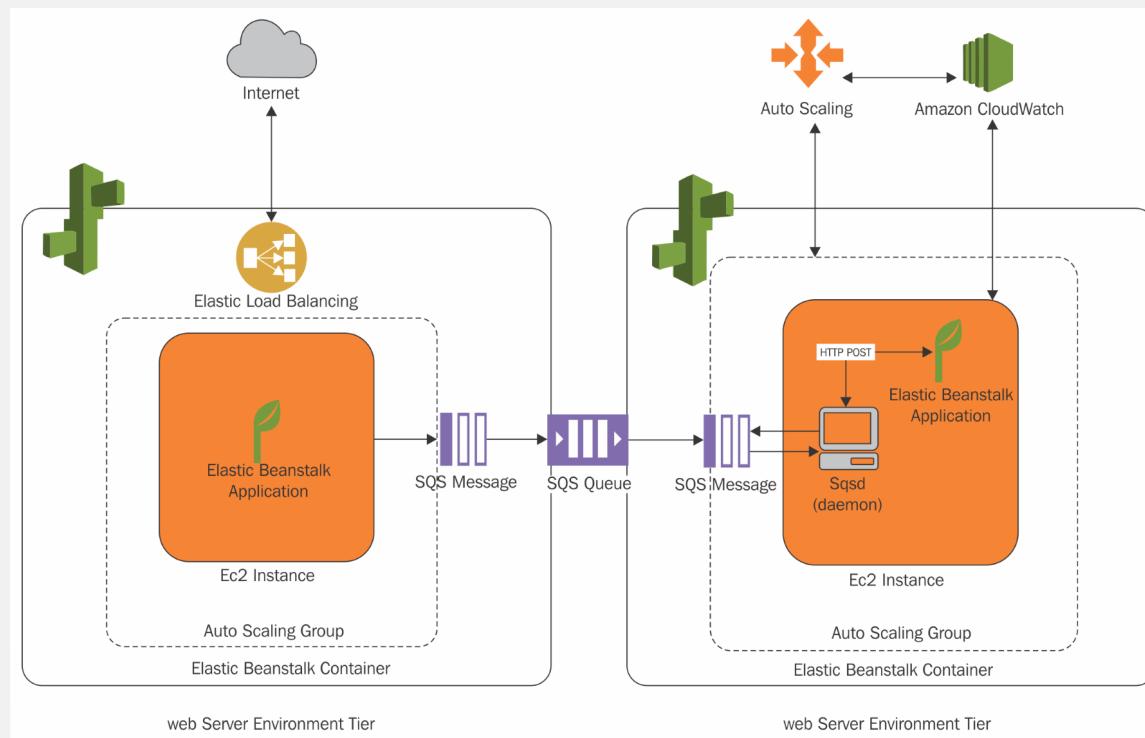
Anatomy of a Web Service Application



Platform as a Service

- Platform as a Service provides environments where all the machinery needed for the distributed application is provided by the cloud provider and managed by them
 - Autoscaling groups
 - Load balancers
 - Queues
 - Daemons managing the queues
 - Consistent data Storage solutions
 - SQL Databases
 - NoSQL solutions

Platform as a Service



Examples

- Microsoft Azure App Service
- Google App Engine
- Amazon Elastic BeanStalk
- Heroku
- IBM Cloud Foundry

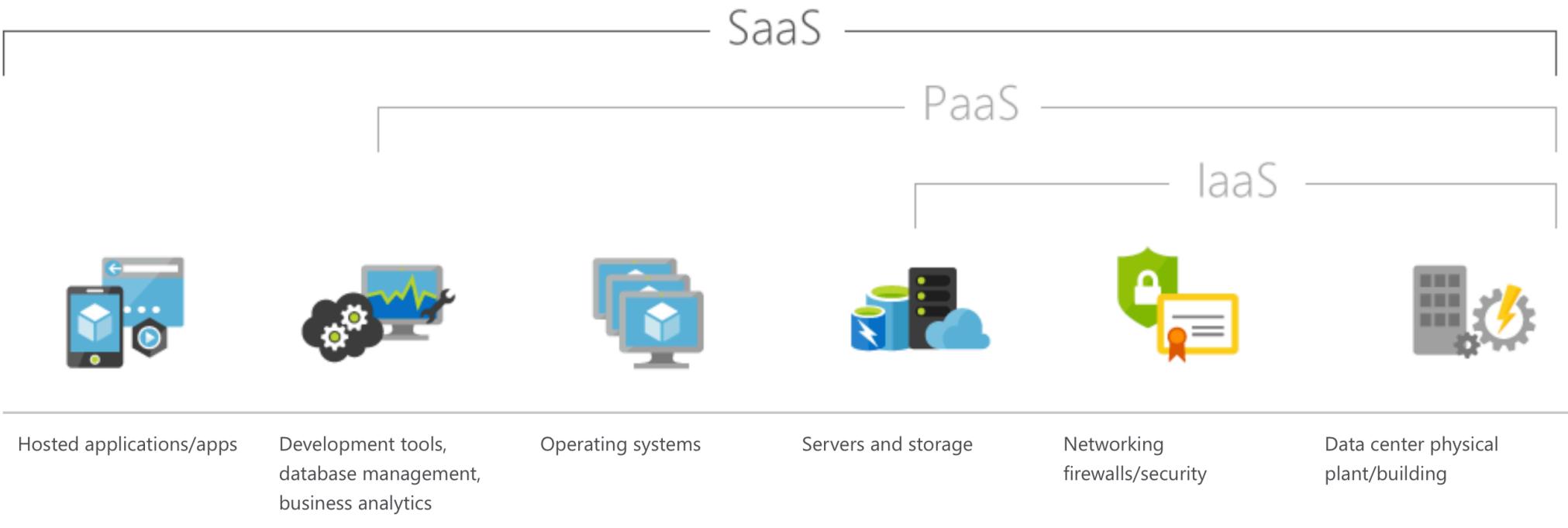
The screenshot shows a comparison chart with three columns. Each column contains a card for a different cloud service: AWS Elastic Beanstalk, Google App Engine, and Microsoft Azure. Each card includes a logo, a call-to-action button, and social metrics.

Service	Stacks	Followers	Votes
AWS Elastic Beanstalk	1.8K	1.5K	240
Google App Engine	6.3K	4.6K	611
Microsoft Azure	13.8K	7.8K	739

Below each card, there is a section titled "Pros of [Service Name]" followed by a list of pros with upvote counts.

Pros of Service	Upvotes	Detail
Pros of AWS Elastic Beanstalk	77	Integrates with other aws services
Pros of AWS Elastic Beanstalk	65	Simple deployment
Pros of AWS Elastic Beanstalk	44	Fast
Pros of Google App Engine	143	Easy to deploy
Pros of Google App Engine	108	Auto scaling
Pros of Google App Engine	80	Good free plan
Pros of Microsoft Azure	111	Scales well and quite easy
Pros of Microsoft Azure	93	Can use .Net or open source tools
Pros of Microsoft Azure	79	Startup friendly

SaaS in Perspective



* Image courtesy of Microsoft Azure



CLOUD COMPUTING APPLICATIONS

Platform as a Service

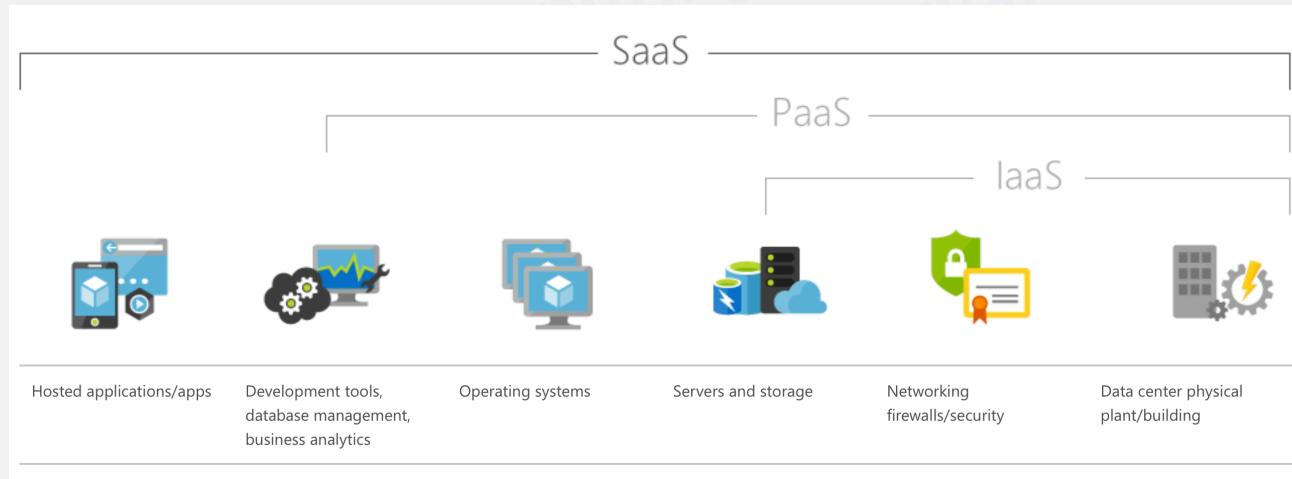
Reza Farivar

Platform as a Service

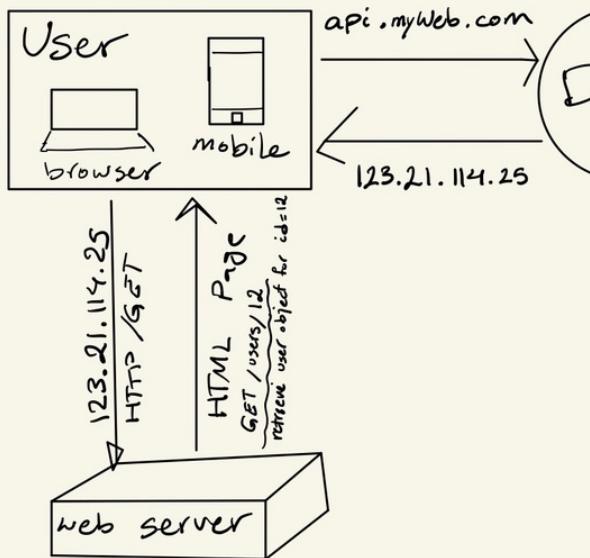
- The main goal is to run the user's distributed web service in a managed environment
 - Much more opinionated than IaaS

Platform as a Service

- The main goal is to run the user's distributed web service in a managed environment
 - Much more opinionated than IaaS

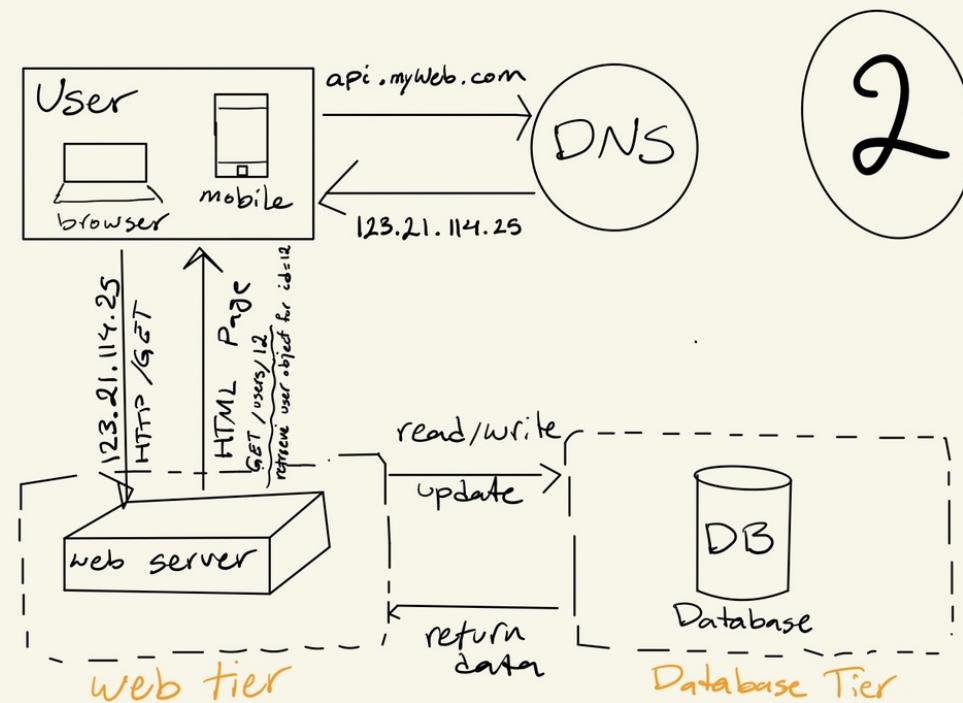


Anatomy of a Web Service Application

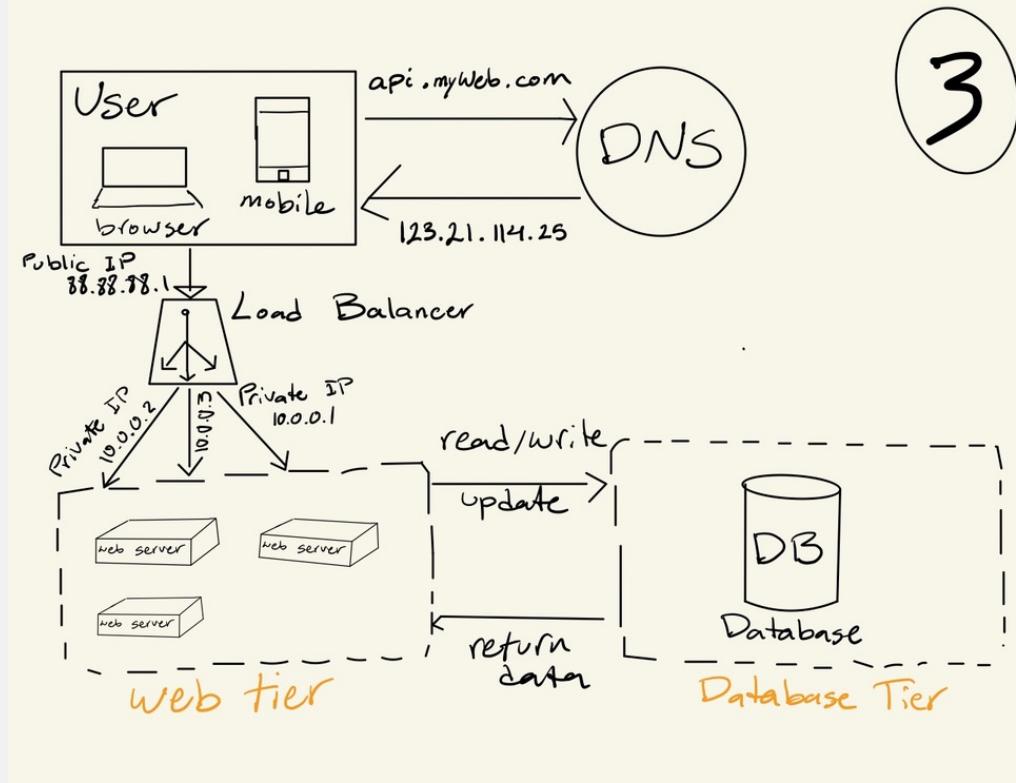


!
GET /users/12
{
 "id": 12,
 "firstName": "John",
 "lastName": "Doe",
 "address": {
 "streetAddress": "123 main st."
 },
 "phoneNumbers": [
 "217-123-1234",
 "800-123-1234"
]
}

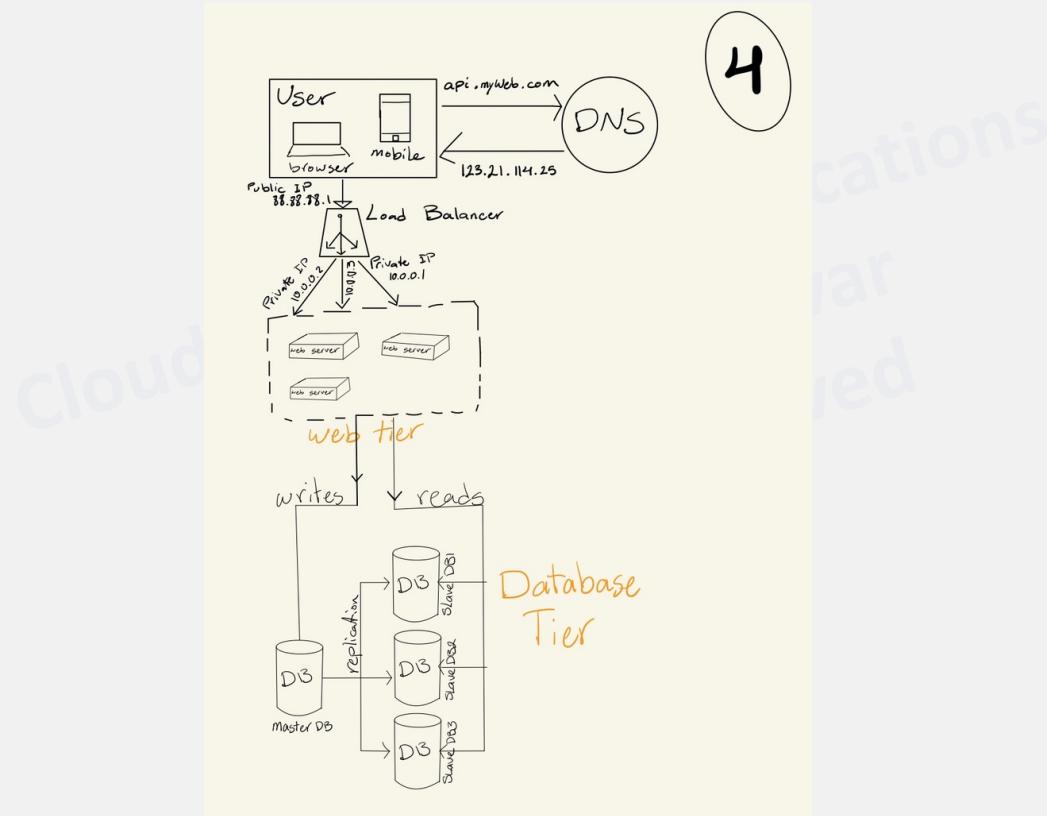
Anatomy of a Web Service Application



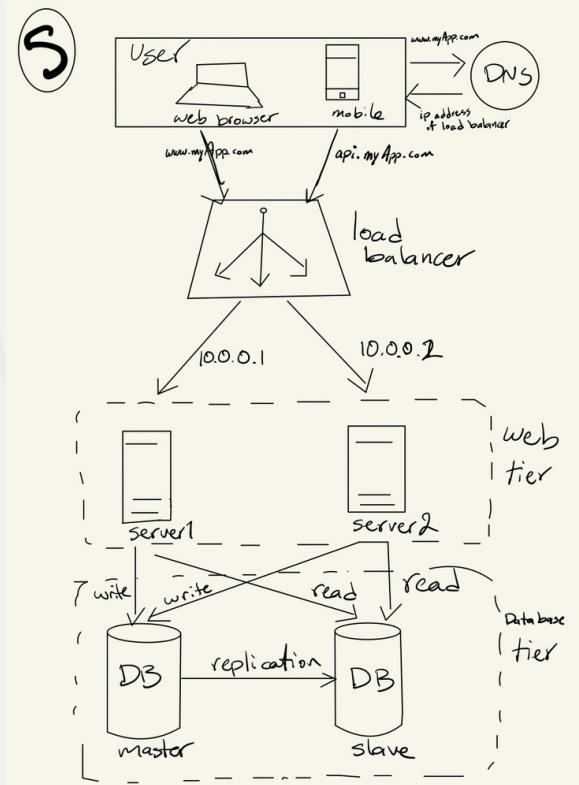
Anatomy of a Web Service Application



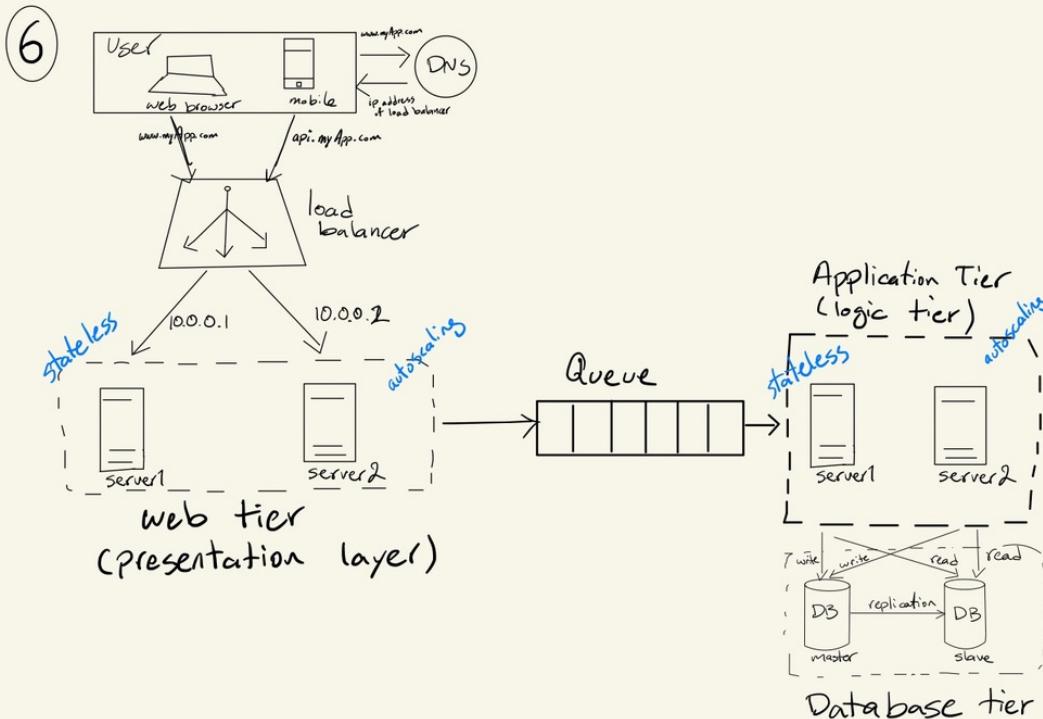
Anatomy of a Web Service Application



Anatomy of a Web Service Application



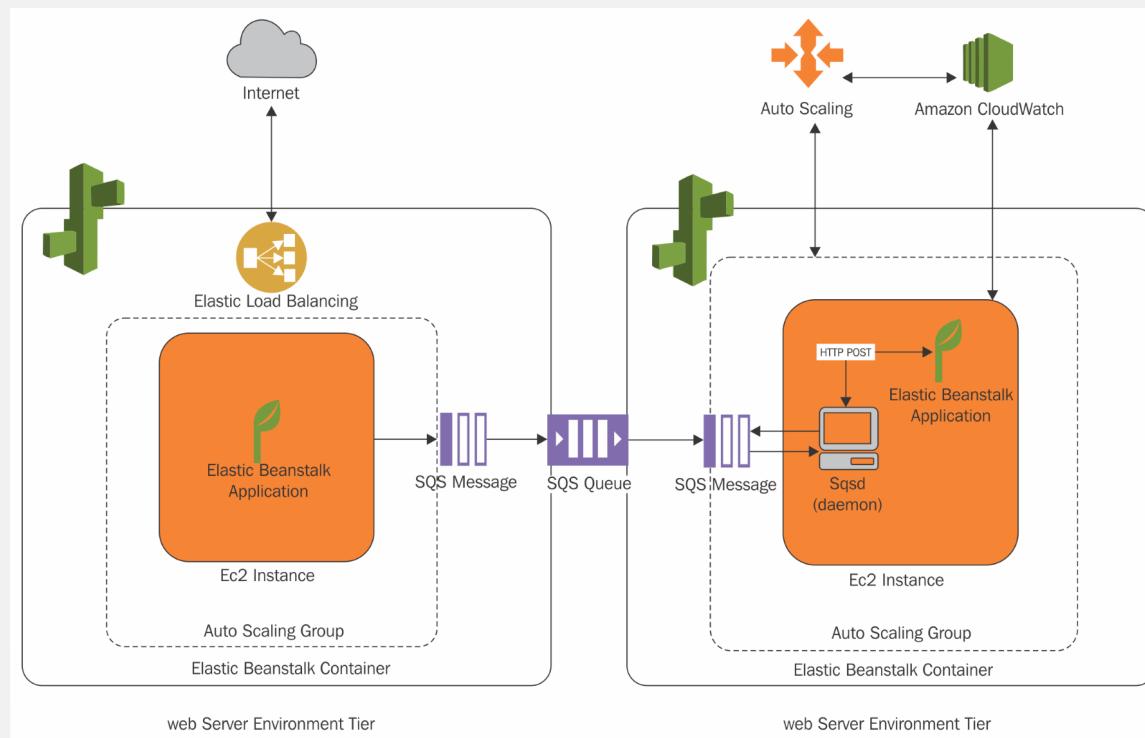
Anatomy of a Web Service Application



Platform as a Service

- Platform as a Service provides environments where all the machinery needed for the distributed application is provided by the cloud provider and managed by them
 - Autoscaling groups
 - Load balancers
 - Queues
 - Daemons managing the queues
 - Consistent data Storage solutions
 - SQL Databases
 - NoSQL solutions

Platform as a Service



Examples

- Microsoft Azure App Service
- Google App Engine
- Amazon Elastic BeanStalk
- Heroku
- IBM Cloud Foundry

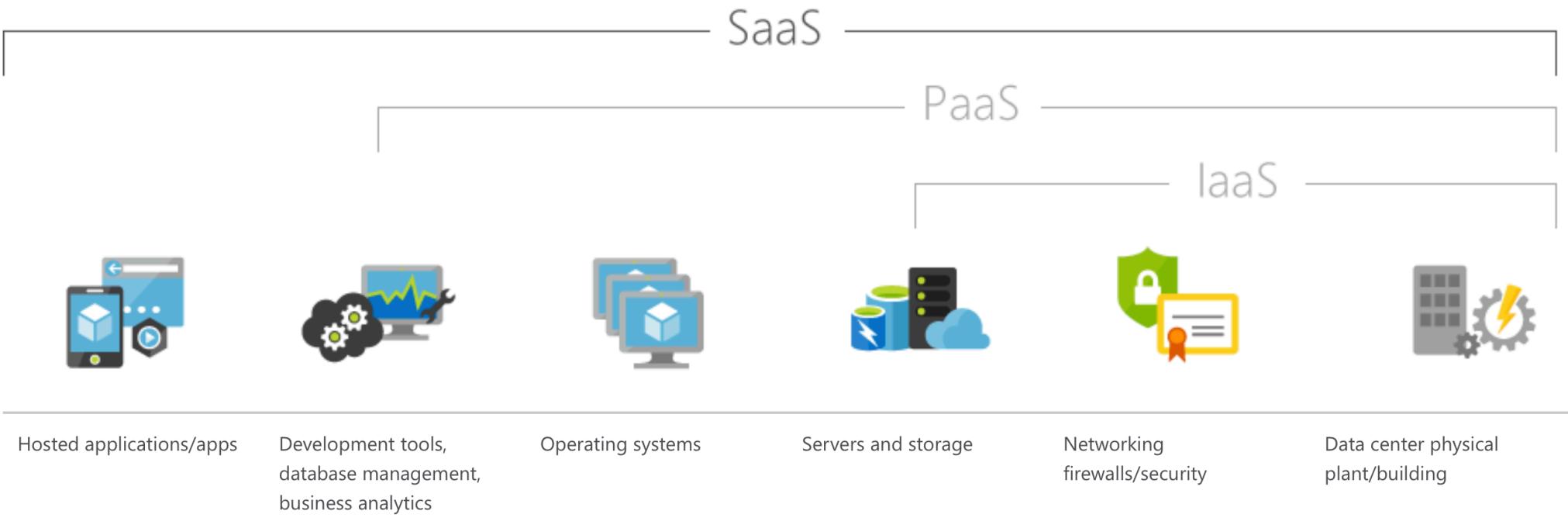
The screenshot shows a comparison chart with three columns. Each column contains a card for a different cloud service: AWS Elastic Beanstalk, Google App Engine, and Microsoft Azure. Each card includes a logo, a call-to-action button, and social metrics.

Service	Stacks	Followers	Votes
AWS Elastic Beanstalk	1.8K	1.5K	240
Google App Engine	6.3K	4.6K	611
Microsoft Azure	13.8K	7.8K	739

Below each card, there is a section titled "Pros of [Service Name]" followed by a list of pros with upvote counts.

Pros of Service	Upvotes	Detail
Pros of AWS Elastic Beanstalk	77	Integrates with other aws services
Pros of AWS Elastic Beanstalk	65	Simple deployment
Pros of AWS Elastic Beanstalk	44	Fast
Pros of Google App Engine	143	Easy to deploy
Pros of Google App Engine	108	Auto scaling
Pros of Google App Engine	80	Good free plan
Pros of Microsoft Azure	111	Scales well and quite easy
Pros of Microsoft Azure	93	Can use .Net or open source tools
Pros of Microsoft Azure	79	Startup friendly

SaaS in Perspective



* Image courtesy of Microsoft Azure



CLOUD COMPUTING APPLICATIONS

PaaS Providers: Google App Engine

Prof. Roy Campbell

Google App Engine (GAE)

- GAE was developed by Google in 2008 as a PaaS
- It supports multi-tenancy and offers automatic scaling for web applications
- It supports Python, Java, and Go

GAE Frameworks and Tools

- GAE supports Django web framework and the Grails web app framework
- GAE provides infrastructure tools that enable users to deploy code without worrying about infrastructure challenges such as deployment, failover, or scalability
- However, the GAE infrastructure limits the type of applications that can be run

GAE Security, Sandbox

- Applications run in a secure environment
- Isolates applications from hardware and operating system, and imposes security limitations
- For example, application code only runs in response to requests, and a request handler cannot spawn potentially malicious sub-processes after response has been sent

Storing GAE Data

- Users of GAE can use App Engine Datastore, Google Cloud SQL, and Google Cloud Storage
- Users can also harness Google's database technology, such as Bigtable

GAE's Use with Google Services

- Can take advantage of Google's single sign on feature when users want to access their Gmail or Google docs
- Build Chrome and Android games on GAE
- Google Cloud Endpoints to use / access mobile services

Other Services Supported

- App engine Map Reduce
- Search API
- SSL support
- Page speed
- XMPP API
- Memcache API

Case Studies of GAE

- BugSense - An application error-reporting service, it used GAE to maintain logs of bugs in software and analyze them
- Ubisoft - Used GAE to build its first web-based game, “From Dust,” on Chrome browser
- Claritics - A small social analytics company of 15 employees, used to analyze game data sets

GAE is Great for Mobile

- Many cell phone apps use GAE for their backend, e.g., Ruzzle and Tap Zoo
- GAE's purpose – being able to scale up for small teams of developers – fits well





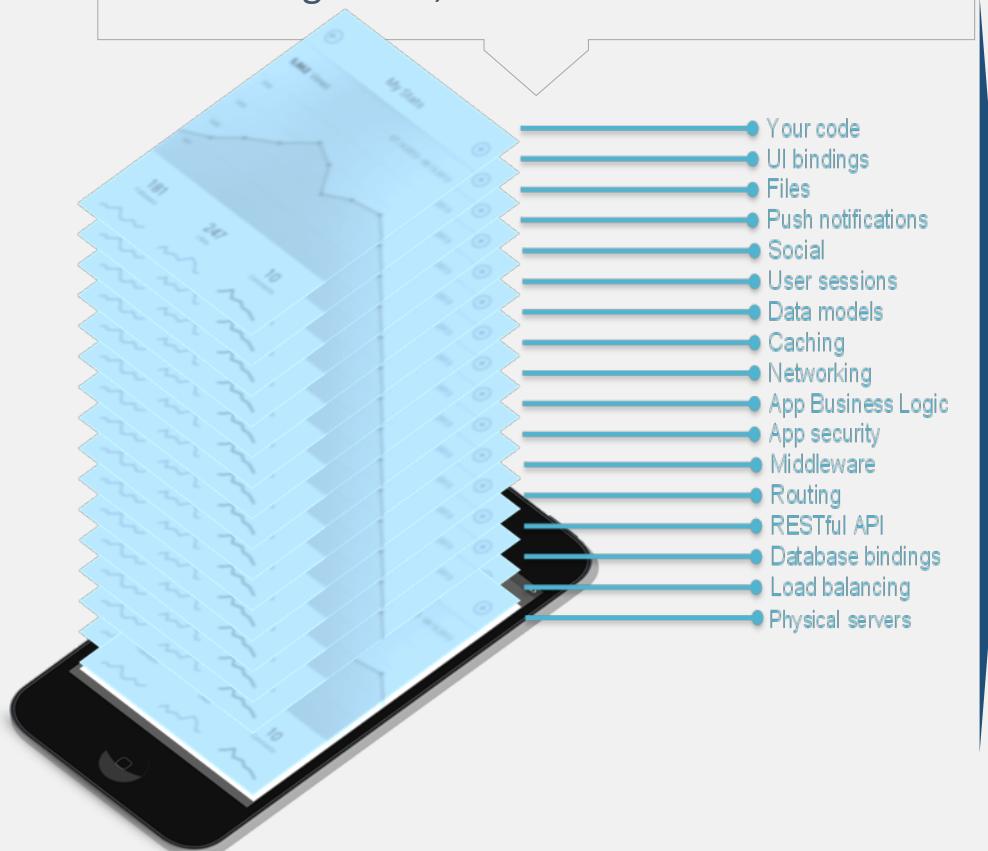
CLOUD COMPUTING APPLICATIONS

Mobile Backend as a Service

Roy Campbell & Reza Farivar

Why MBaaS?

When you are thinking to develop a new app, you have lot of steps like server setup, database creation, routing, social integration, UI binding, file management, etc. that need to be solved:



Just imagine, you focus only on your frontend code and the rest will bind together as a service as follows:

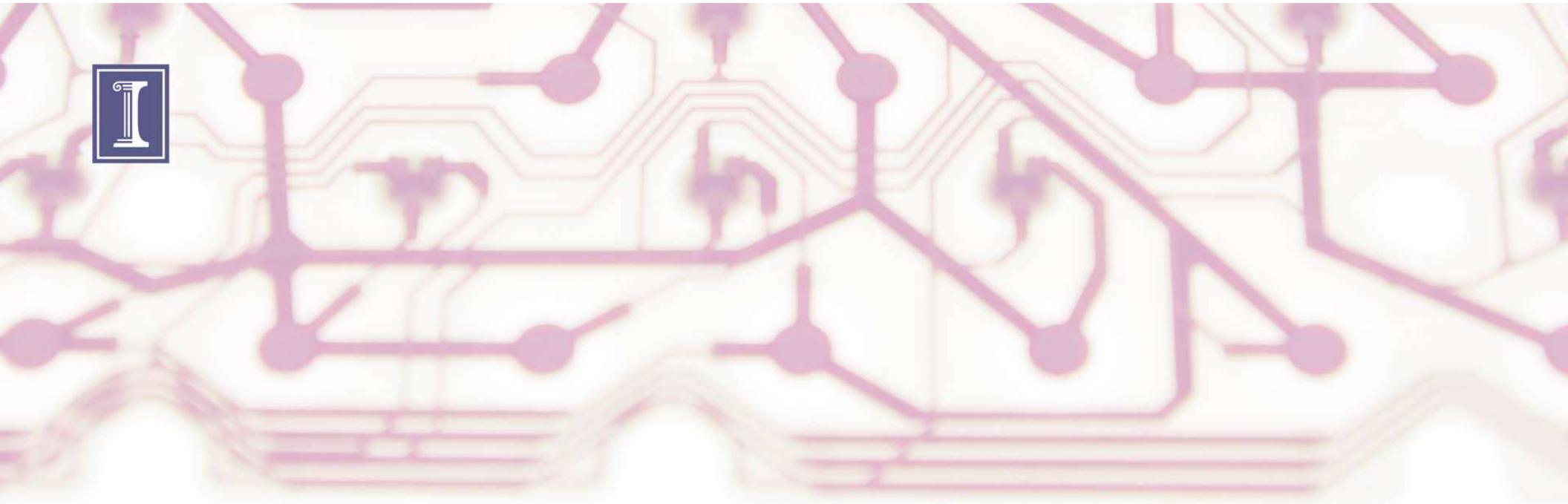


Introduction to MBaaS

- General idea: mobile apps need common services that can be shared among apps instead of being custom developed for each
- Enable web and mobile app developers to link their applications to backend cloud storage and backend APIs
 - Cloud storage
 - User management
 - Push notifications
 - Integration with social networking services
- Provide all of these in a one-shop model

MBaaS Examples

- Appcelerator, AnyPresence, Appery, Built.io, FeedHenry, Kinvey, TruMobi, Apple CloudKit (iCloud), etc.
- Many commonalities
 - E.g. many use MongoDB to serve JSON objects
 - REST API common
 - MicroServices
 - DevOps
 - Frontend design framework
- Different levels of enterprise integration
- Either on-premise or in private clouds
- Some support compliance with HIPAA, PCI, FIPS, and EU data security standards



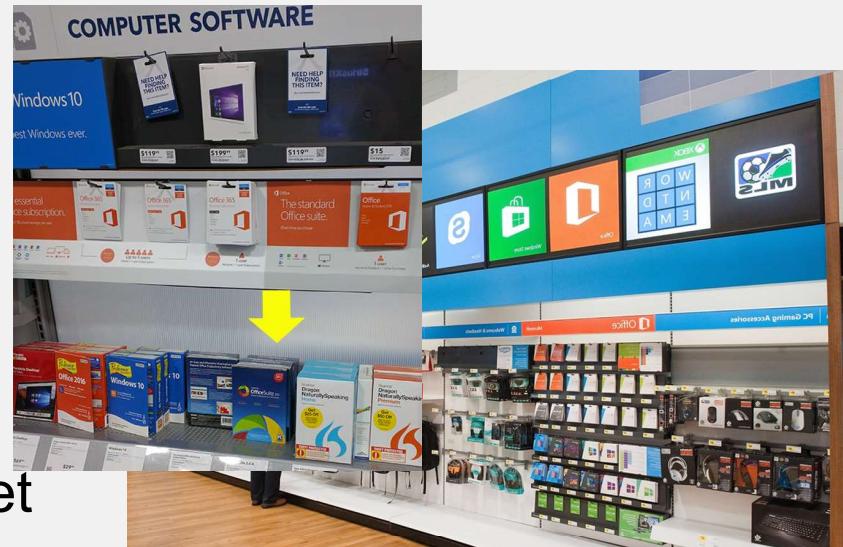
CLOUD COMPUTING APPLICATIONS

Software as a Service

Reza Farivar

Software Distribution

- Once upon a time, retail was the main method of getting access to software
- With the advent of Cloud Computing, that model is now dated
- Then came software purchase over internet as downloads
 - Mobile App Store
 - Steam
 - Etc.
- Third wave is SaaS, which mainly rely on Browser capability and broadband internet
 - Browser is the new Operating System
- Marc Andresen, 1995



Software as a Service

- Bring a complete software solution to the consumer with zero setup in a browser
- Examples:
 - Web-based email (gmail, Yahoo! Mail)
 - Internet Search Portals (Google, Yahoo!, Bing)
 - Project Management (Atlassian JIRA)
 - Office Productivity (MS Office 365, Google Docs)
 - Document Signing (DocuSign)
 - Customer Relations Management (Salesforce)
 - Tax Services (TurboTax, H&R Block, TaxAct)
 - Remote education (Coursera ☺)

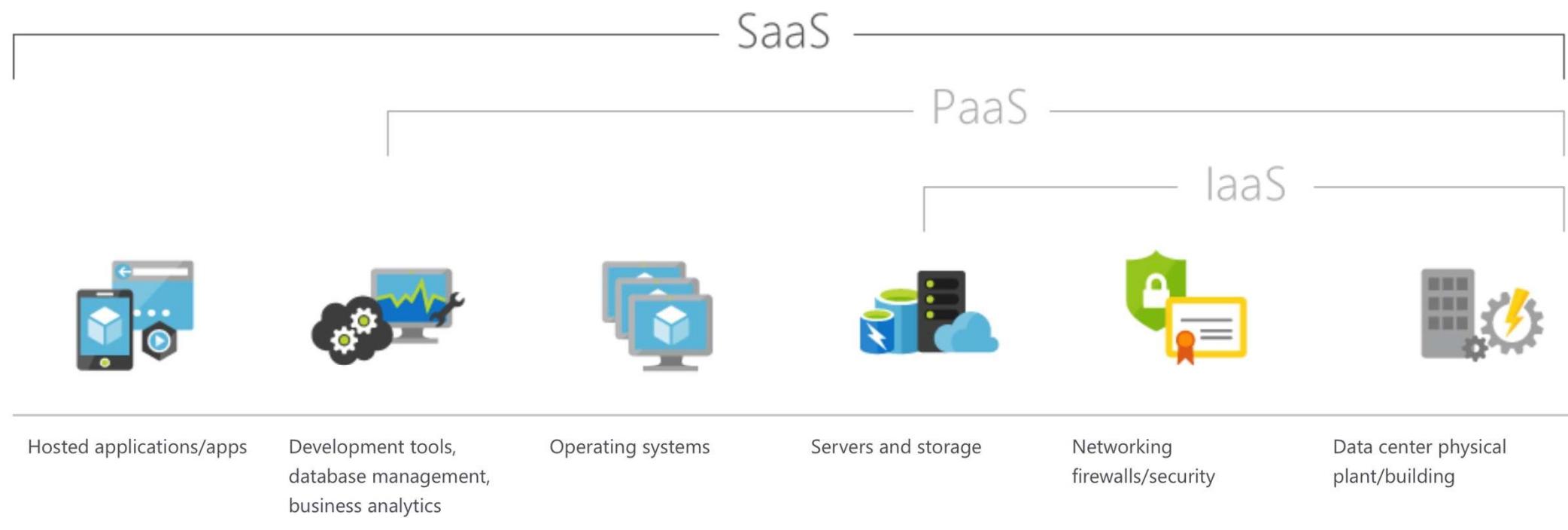
Software as a Service

- Multitenant Architecture
 - Same Software for all customers
 - Data is specific for individual user
- Client-side software runs in users' browser
- Server-side software runs in the cloud
 - Many SaaS solutions themselves build on IaaS and PaaS services
- Customer's data may be stored locally, in the cloud or both locally and in the cloud
- Because SaaS applications cannot access a company's internal systems, many SaaS solutions allow API access to further customize their offerings
 - HTTP, REST, SOAP

Advantages and Disadvantages

- Advantages:
 - No need to install software, licensing, maintenance and support
 - Flexible Payments
 - Scalable Usage
 - Automatic Updates
 - Access anywhere
- Disadvantages
 - You lose control
 - Might not be such a bad thing, see above advantages
 - No access to source code (as opposed to Open Source)
 - Provider service disruptions will impact you

SaaS in Perspective



* Image courtesy of Microsoft Azure



CLOUD COMPUTING APPLICATIONS

Prof. Roy Campbell

PAAS Providers: SalesForce

Main CRM Services (Mainly SaaS, PaaS)

- Sales Cloud, Service Cloud, Marketing Cloud, Analytics Cloud, Community Cloud, Lightening
- Social Software in the Workplace (Microsoft, IBVM, Jive, Salesforce)
- Find out about each other personally or professionally
- Mine their networks of contacts and acquaintances for advice, references and referrals
- Form teams, communities or informal groups
- Collaborate on the same work objects

Main CRM Services (Mainly SaaS, PaaS)

- Discuss and comment on their work
- Organize work from an individual or group perspective
- Identify relevant information
- Discover other people with common interests
- Alert users to information or events that might be relevant to them
- Learn from others' expertise
- Wave, Salesforce Analytics Cloud



CLOUD COMPUTING APPLICATIONS

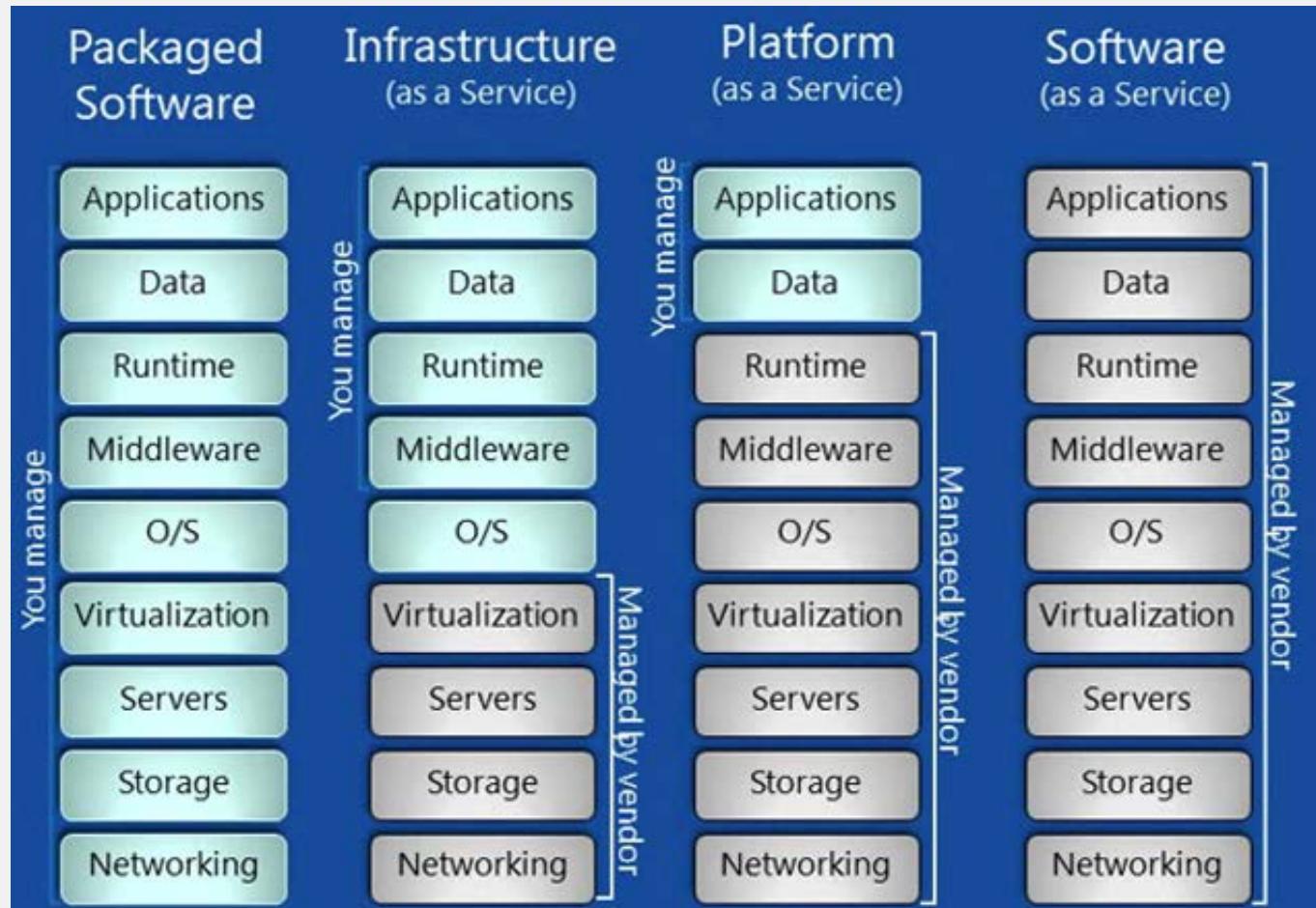
Roy Campbell & Reza Farivar

CLOUD SERVICES

Objective

- Compare IaaS, PaaS, and SaaS
- Look at what services major Cloud companies provide and how they provide them

IaaS, PaaS, and SaaS Comparison



Cloud Fundamentals

- Infrastructure as a Service (IaaS): basic compute and storage resources
 - On-demand servers
 - Amazon EC2, VMWare, vCloud
- Platform as a Service (PaaS): Cloud application infrastructure
 - On-demand application-hosting environment
 - For example, Google AppEngine, Salesforce.com, Windows Azure, Amazon
- Software as a Service (SaaS): Cloud applications
 - On-demand applications
 - For example, GMail, Microsoft Office Web Companions

Platform as a Service (PaaS)

- PaaS is a Cloud computing service that offers a platform for users to run applications on the Cloud
- PaaS is a level above IaaS because unlike IaaS, PaaS does not require users to develop their own operating system environment

Platform as a Service (PaaS)

- Middle ground between SaaS and IaaS
- Development platform
 - Customers use it to develop applications that benefit from the scalability of the Cloud without fully developing their own solution using an IaaS provider
- Offers an application development platform that will automatically scale with demand

The Benefits of the Cloud

The Cloud is about cheap, on-demand capacity

 = Managed for You	Standalone Servers	IaaS	PaaS	SaaS
Applications	✗	✗	✗	✓
Runtimes	✗	✗	✓	✓
Database	✗	✗	✓	✓
Operating system	✗	✗	✓	✓
Virtualization	✗	✓	✓	✓
Server	✗	✓	✓	✓
Storage	✗	✓	✓	✓
Networking	✗	✓	✓	✓

Platform as a Service (PaaS)

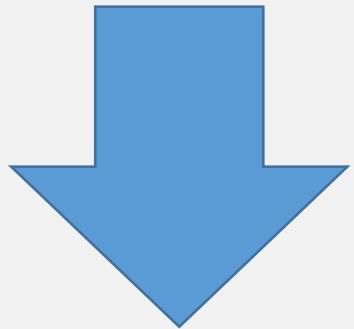
Official definition of PaaS from NIST standard

“The capability provided to the consumer is to deploy, onto the Cloud infrastructure, consumer-created or acquired applications created using programming languages, libraries, services, and tools supported by the provider. The consumer does not manage or control the underlying Cloud infrastructure, including network, servers, operating systems, or storage, but has control over the deployed applications and possibly configuration settings for the application-hosting environment.”

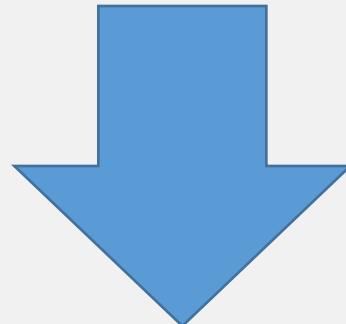
Example: Google

PaaS

Runtime environment,
database, development



Google
App
Engine



Amazon
AWS, EC2

Example: Windows Azure

- PaaS
 - Application platform in the Cloud
- Provides
 - *Compute*
 - Web, worker, and VM role
 - *Storage*
 - Blob, Table, Queue, and Azure SQL server
 - *Application fabric*
 - Service bus, access control
 - Future: cache, integration, and composite

More Cost Effective

- PaaS can be better for costs than IaaS, because systems are optimized to run applications efficiently
- IaaS may only provide hardware, and thus, clients must be in charge of load balancing and networking

Multi-tenancy

- PaaS is better suited for **multi-tenancy** because the PaaS provider optimizes its infrastructure for use by many providers
- Multi-tenancy means that many users may share the same physical computer and database

Multi-tenancy

- PaaS is better suited for multi-tenancy than an IaaS because an IaaS may (1) provide each user with his own virtual machine and (2) create a clear separation of resources
- However, in a PaaS, users may share the same machine, database, etc.

Vendor Lock-in

- PaaS may lock in applications by requiring users to develop apps using proprietary interfaces and languages
- This means that it may be difficult for users to go to another vendor to host their app
- Businesses may risk their future on the dependability of the PaaS

Development Tools

- Often, a PaaS will offer browser-based development tools
- In this way, developers can create their own applications online
- Ease of deployment: the platform takes care of the scaling for you

Principles of Software Development

- As a developer, your objective is to create an application in the quickest, most effective way possible
- You should not create applications using convoluted methods that may take a long time to complete
- The user only sees the end product, not the development process

PaaS vs. IaaS

- When you use the Cloud, remember that your decisions have long-term consequences
- If you choose to use a PaaS and get your application vendor locked in, then your business may fail if the PaaS greatly increases the vendor's prices
- You will not be able to move to another Cloud since your app cannot be easily migrated to somewhere else

PaaS vs. IaaS

- An app that is used to fulfill a temporary need may be handled by a PaaS solution
- An app that needs to be deployed quickly may be faster developed by a PaaS
- If your software team is small, it may be better to develop a PaaS and let the PaaS provider handle the OS and networking for your team



CLOUD COMPUTING APPLICATIONS

Cloud Computing Glue: Introduction

Prof. Reza Farivar

Cloud Computing Glue

- Cloud Computing is all about running your compute tasks and storing your data on other computers
- At the core, it's about communication between computing sources
 - Your client + some server on the cloud, through internet
- Have we seen this problem before?
 - Inter-process communication
 - Communication on a local network
 - Communication on internet

Overview

1. Intro
2. Communication
3. Internet Protocol, HTTP and RTC on HTTP
4. Service Oriented Architecture and SOAP
5. RESTful Architecture
6. Asynchronous RPC, WebSocket
7. HTTP2 Push, Streaming Video
8. JSON, comparison with XML
9. RPC semantics and implementation
10. Protocol Buffers and Thrift

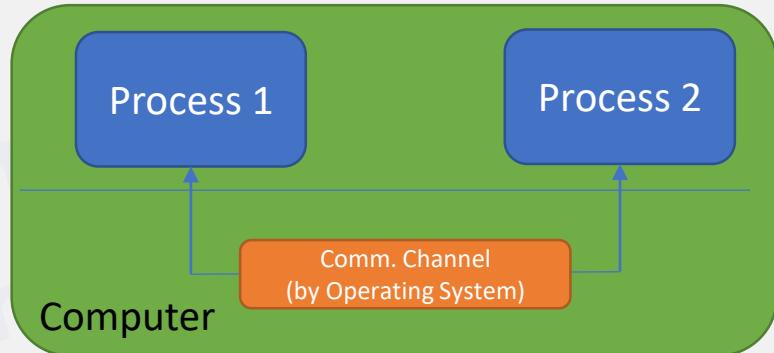


CLOUD COMPUTING APPLICATIONS

Cloud Computing Glue:
Communication
Prof. Reza Farivar

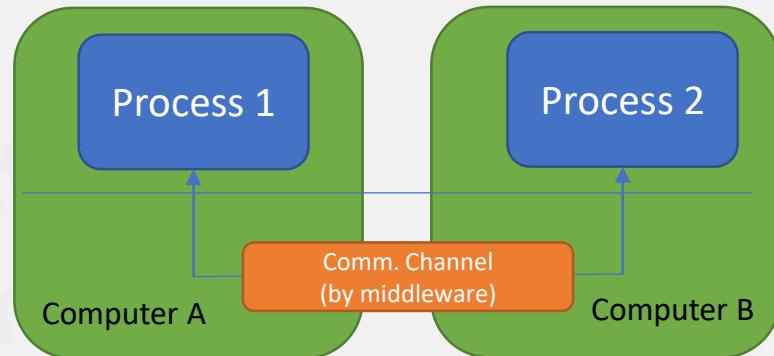
Communication in a single machine

- Communication Channel provided by the Operating System
- Shared memory block
- Shared File System
- Signal
- POSIX Socket, aka. Berkeley Socket
 - Port numbers
 - *SOCK_STREAM (compare to TCP)*
 - *SOCK_DGRAM (compare to UDP)*
- Remote Method Invocation (RMI)
 - Method invocations between objects *in different processes* (*processes may be on the same or different host*)
 - *From one JVM to another*
- Message Queue
- Message Passing
 - Unix Pipe
 - *Actor Model*
 - *Pi Calculus*

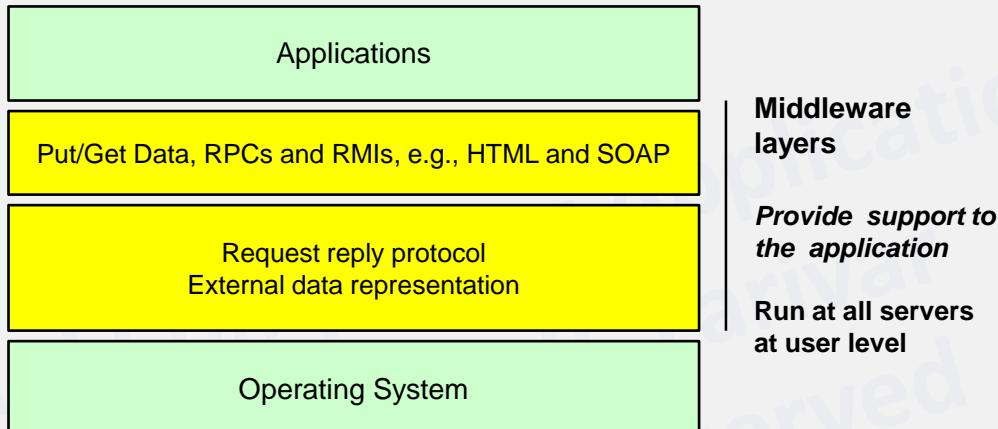


Middleware Layer Definition

- *Software that provides services to applications beyond those generally available at the operating system*
- Middleware implements functionalities that are common across many different applications
 - No need to reinvent the wheel (e.g., message parsing) every time you need to do something
- A Middleware Layer can provide the same abstractions to distributed applications!
 - Building distributed systems while maintaining our code is not very different from a single-node program



Middleware Layers



RMI = Remote Method Invocation

CORBA = Common Object Request Brokerage Architecture

SOAP = Simple Object Access Protocol

Communication in a local network

- Scientific Computing
 - Message Passing Interface (MPI)
 - Simple model: Send() and Receive()
 - No native support for fault tolerance
 - Programming interface is complicated
 - Race, deadlock, etc.
- Business Sector
 - Remote Procedure Calls
 - RPC Semantics (behavior in presence of network failures)
 - RPC Implementation
 - Remote Method Invocation (RMI)
 - Between two JVMs on a network

Communication in Big Data Deployments

- Need scaling from the start
 - Sometimes many 10s of thousands of nodes on the network
 - Ad hoc solutions do not work
- RPC Frameworks
 - Google Protocol Buffer
 - You define the functions that will be called remotely
 - Then Compile
 - The system automatically generates interfaces functioning as communication stubs in your choice of programming language
 - Many languages are supported: C++, C#, Objective C, Java, Python, etc.
 - You import the generated header / code files into your project
 - At run-time, your program just calls the function locally. The auto-generated code takes care of serialization and marshalling of the function parameters, making the network calls (handling any network errors), and transferring the function call in the target system.
 - Apache Thrift
 - Apache HDFS, Hadoop, Spark, Storm, etc. extensively use Thrift
- Consistency
 - Paxos
 - Zookeeper

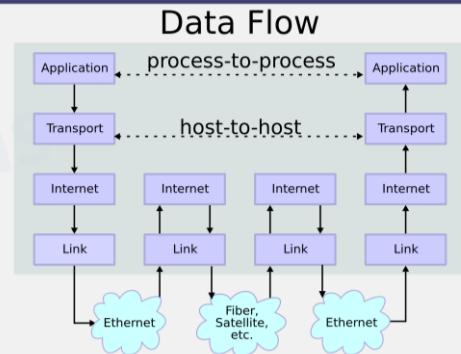


CLOUD COMPUTING APPLICATIONS

Cloud Computing Glue: Internet
Protocol, HTTP and RPC on HTTP
Prof. Reza Farivar

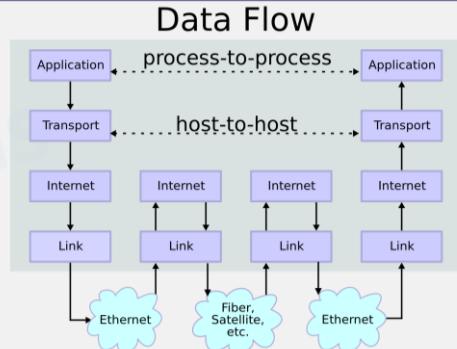
Internet Protocol Stack: Link and Internet Layers

- The second level, “internet level”, is where internet switches and routers operate
 - Gets your message from one machine to another*
 - Responsible for addressing **host interfaces**
 - Encapsulating data into datagrams (including fragmentation and reassembly)
 - Routing datagrams across one or more IP networks
 - IP address, IPV4, IPV6*
- EE and Network engineers live at the bottom level, the Link level
 - Out of scope our discussion



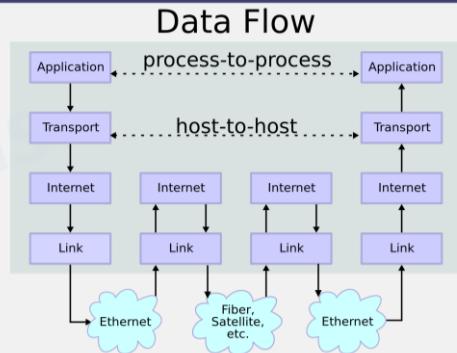
Internet Protocol Stack: Transport Layer

- The transport level handles packetizing of data
- Gets your message from a process in one machine to another process in another machine
- Typically implemented in the Operating System
- Provide port numbers (similar to Unix sockets)
- TCP
 - Keeps track of data segments, retransmission, acknowledgement
 - Handle network congestion
 - Traffic load balancing
 - Unpredictable network behavior
 - Lost, duplicated, or delivered out of order IP packets
 - Guarantees that all bytes received will be identical and in the same order as those sent
- UDP
 - “User Datagram Protocol”, aka. “Unreliable Datagram Protocol”
 - Very simple



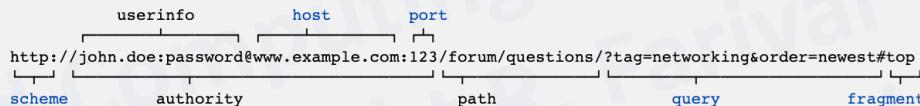
Internet Protocol Stack: Application Layer

- Application level handles “what to send”
 - HTTP, HTTPS
 - RESTful APIs
 - FTP
 - WebSocket
 - SMTP
 - IMAP
 - SSH
 - DHCP
 - DNS
 - Bit Torrent



HTTP Protocol

- Originally targeted static web pages: *Client Requests, Server Responds, Connection is closed*
- Works on top of TCP for reliable transport
- Client (user agent) can be a web browser, or any other software
- HTTP resources are identified and located on the network by Uniform Resource Locators (URLs), using the Uniform Resource Identifiers (URI's) schemes http and https



- In HTTP/1.0 a separate connection to the same server is made for every resource request
 - Establishment of TCP connections has overhead
- HTTP/1.1 can reuse a connection multiple times to download images, scripts, stylesheets, etc. after the page has been delivered
 - Persistent Sessions
 - Lower latency

HTTP Message Format

- Request “verbs”
 - GET: retrieve data
 - POST: server should accept the call parameter as a new value for the resource specified in the URL
 - PUT: server should store the enclose entity under the supplied URL
 - DELETE: server should delete the specified resource by the URL
 - PATCH: server should apply partial modification to the resource
 - ...
- Request message
 - a request line (e.g., GET /dataset/inventory.htm HTTP/1.1, which requests /dataset/inventory.htm resource from the server)
 - request header fields
 - an empty line
 - an optional message body
- Response message
 - a status line which includes the status code and reason message (e.g., HTTP/1.1 200 OK, which indicates that the client's request succeeded)
 - *Informational 1XX | Successful 2XX | Redirection 3XX | Client Error 4XX | Server Error 5XX*
 - response header fields (e.g., Content-Type: text/html)
 - an empty line
 - an optional message body

Client Request

```
GET / HTTP/1.1  
Host: www.example.com
```

Session

Server Response

```
HTTP/1.1 200 OK  
Date: Mon, 23 May 2005 22:38:34 GMT  
Content-Type: text/html; charset=UTF-8  
Content-Length: 138  
Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT  
Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)  
ETag: "3f80f-1b6-3elcb03b"  
Accept-Ranges: bytes  
Connection: close  
  
<html>  
 <head>  
   <title>An Example Page</title>  
 </head>  
 <body>  
   <p>Hello World, this is a very simple HTML document.</p>  
 </body>  
</html>
```

RPC on HTTP

- Remote Procedure Calls built on HTTP
 - For many types of RPC, the client/server conversation model of HTTP works just fine
 - Just replace the HTML markup with an XML or JSON representation of the data
 - XML-RPC
 - JSON-RPC
 - E.g. Bitcoin server
 - Commands encoded as JSON, sent over HTTP

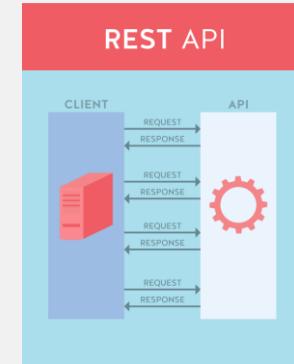


CLOUD COMPUTING APPLICATIONS

Cloud Computing Glue: RESTful
Architecture
Prof. Reza Farivar

Representational State Transfer (REST)

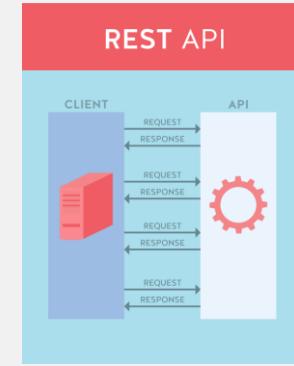
- A style of software architecture for distributed hypermedia systems such as the World Wide Web
- Introduced in the doctoral dissertation of Roy Fielding
 - One of the principal authors of the HTTP specification
- The motivation for REST was to capture those characteristics of the Web that made the Web successful
 - URI-addressable resources
 - HTTP
 - Make a request – receive response – display response
- A collection of network architecture principles that outline how resources are defined and addressed
 - Based on HTTP methods to access resources via URL-encoded parameters and the use of JSON or XML to transmit data
 - Request/response between client and server, like a conversation
 - Something is requested, something is done, and then something is sent in return



RESTful API

- Uses HTTP verbs: GET, POST, PUT, PATCH, DELETE
 - Exploits the use of the HTTP beyond HTTP POST and HTTP GET
 - HTTP PUT and DELETE are not even supported in HTML
 - GET is safe (does not change state)
 - GET, PUT and DELETE are idempotent (you can execute them more than once and get the same state change result)
 - Example request:
 - curl -X POST <https://api.github.com/user/repos>
 - Response:

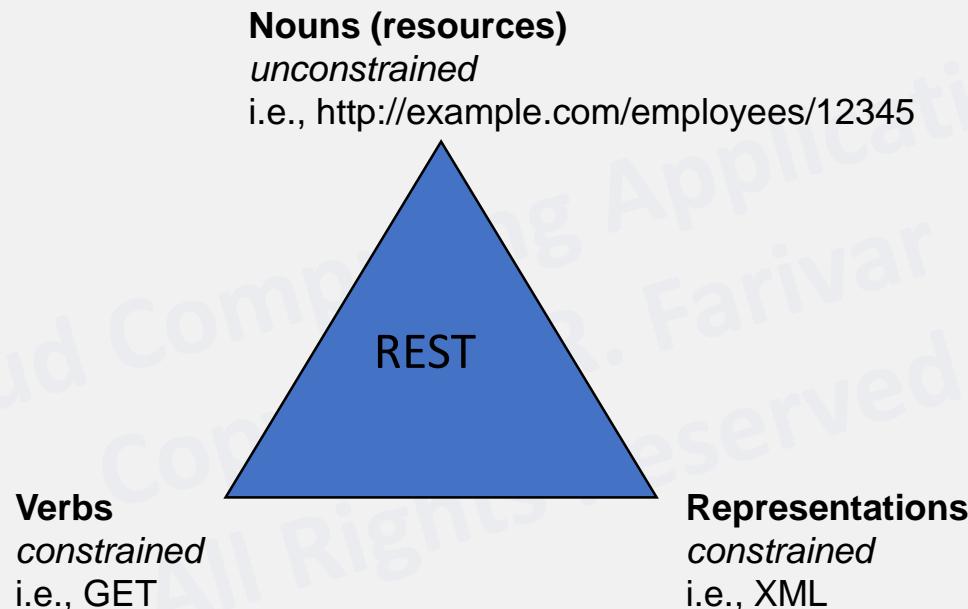
```
{  
  "message": "Requires authentication",  
  "documentation_url": "https://developer.github.com/v3/repos/#create"  
}
```



REST – Not a Standard

- There is no “official standard”, REST is an architectural style
 - JSR 311: JAX-RS: The Java™ API for RESTful Web Services
- But it uses several standards:
 - HTTP
 - URL
 - XML/HTML/GIF/JPEG/etc. (resource representations)
 - Text/xml, text/html, image/gif, image/jpeg, etc. (resource types, MIME types)
- Huge adoption for “Web mashup” applications, operations on When entities
- Many cloud SaaS and PaaS services
- LinkedIn, Twitter,

Main Concepts

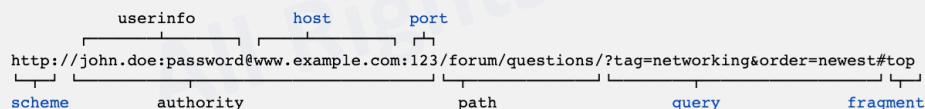


Resources

- The key abstraction of information in REST is a resource
- A resource is a conceptual mapping to a set of entities
 - Any information that can be named can be a resource: a document or image, a temporal service (e.g., "today's weather in Los Angeles"), a collection of other resources, a non-virtual object (e.g., a person), and so on
- Represented with a global identifier (URI in HTTP)
 - <http://www.boeing.com/aircraft/747>

Naming Resources

- REST uses URI to identify resources
 - <http://localhost/books/>
 - <http://localhost/books/ISBN-0011>
 - <http://localhost/books/ISBN-0011/authors>
 - <http://localhost/classes>
 - <http://localhost/classes/cs2650>
 - <http://localhost/classes/cs2650/students>
- As you traverse the path from more generic to more specific, you are navigating the data



Verbs

- Represent the actions to be performed on resources
- HTTP GET
- HTTP POST
- HTTP PUT
- HTTP DELETE
- HTTP PATCH

HTTP GET

- How clients ask for the information they seek
- Issuing a GET request transfers the data from the server to the client in some representation
- GET <http://localhost/books>
 - Retrieve all books
- GET <http://localhost/books/ISBN-0011021>
 - Retrieve book identified with ISBN-0011021
- GET <http://localhost/books/ISBN-0011021/authors>
 - Retrieve authors for book identified with ISBN-0011021

HTTP POST, HTTP PUT

- HTTP POST creates a resource
- HTTP PUT updates a resource
- POST <http://localhost/books/>
 - Content: {title, authors[], ...}
 - Creates a new book with given properties
- PUT <http://localhost/books/isbn-111>
 - Content: {isbn, title, authors[], ...}
 - Updates book identified by isbn-111 with submitted properties

HTTP DELETE

- Removes the resource identified by the URI
- DELETE <http://localhost/books/ISBN-0011>
 - Delete book identified by ISBN-0011

Representations

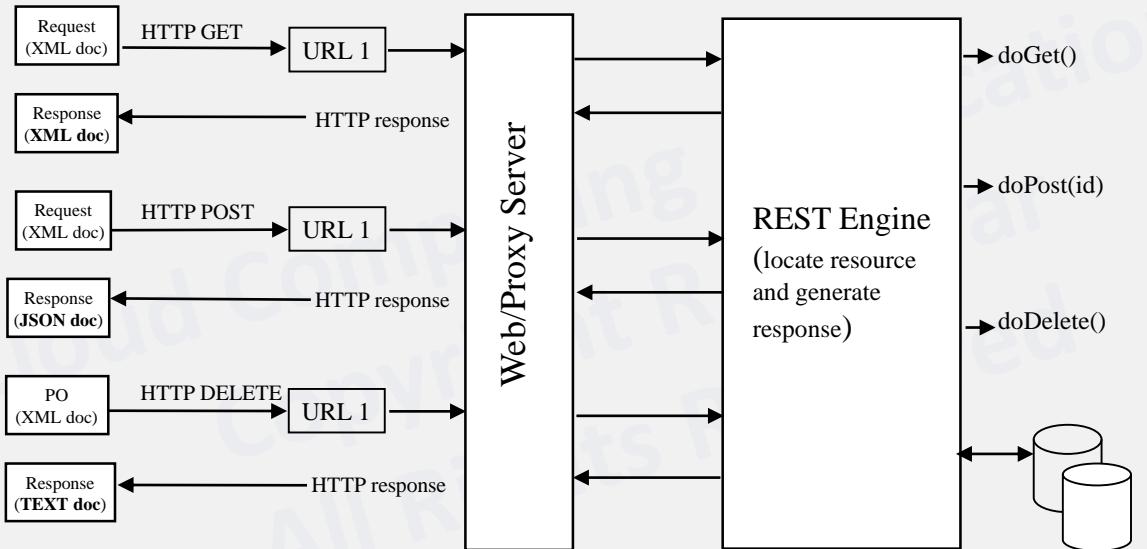
- How data is represented or returned to the client for presentation
- Two main formats:
 - JavaScript Object Notation (JSON)
 - XML
- It is common to have multiple representations of the same data
- XML

```
<COURSE>
    <ID>CS2650</ID>
    <NAME>Distributed Multimedia Software</NAME>
</COURSE>
```

- JSON

```
{course
    {id: CS2650}
    {name: Distributed Multimedia Software}
}
```

Architecture Style



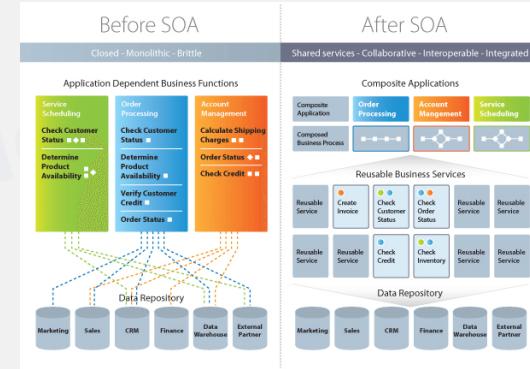


CLOUD COMPUTING APPLICATIONS

Cloud Computing Glue: Service
Oriented Architecture and SOAP
Prof. Reza Farivar

Service Oriented Architecture

- Came out of the needs of the business sector, enterprise and B2B applications
 - “*SOA is the philosophy of encapsulating application logic in services with a uniformly defined interface and making these publicly available via discovery mechanisms.*”
- Benefits of SOA
 - Reusable Code
 - Interaction
 - Scalability
 - Reduce Costs
- The term “Web Services” typically relates to this type of communication
 - Web Services are one option to implement SOA
 - Other options include: Java Business Integration (JBI), Windows Communication Foundation (WCF) and data distribution service (DDS)
- An example Technology was / is SOAP



Simple Object Access Protocol (SOAP)

- SOAP-based Web APIs use XML validation to ensure structural message integrity

- XML schemas provisioned with WSDL documents

- Evolved as the successor to XML-RPC

- Characteristics:

- Extensible
 - Neutrality (from transport layer)
 - Can run on HTTP, WebSocket, even SMTP
 - Independence

- Use XML validation to ensure structural message integrity

- XML schemas provisioned with WSDL documents

- Evolved as the successor to XML-RPC

- Characteristics:

- Extensible
 - Neutrality (from transport layer)
 - Can run on HTTP, WebSocket, even SMTP
 - Independence

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: 299
SOAPAction: "http://www.w3.org/2003/05/soap-envelope"

<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope" xmlns:m="http://www.example.org">
  <soap:Header>
  </soap:Header>
  <soap:Body>
    <m:GetStockPrice>
      <m:StockName>T</m:StockName>
    </m:GetStockPrice>
  </soap:Body>
</soap:Envelope>
```

Example Message

* From Wikipedia

Simple Object Access Protocol (SOAP)

- SOAP-based Web APIs use XML validation to ensure structural message integrity
 - XML schemas provisioned with WSDL documents
- Evolved as the successor to XML-RPC
- Characteristics:
 - Extensible
 - Neutrality (from transport layer)
 - Can run on HTTP, WebSocket, even SMTP
 - Independence
- Use XML validation to ensure structural message integrity
 - XML schemas provisioned with WSDL documents
- Evolved as the successor to XML-RPC
- Characteristics:
 - Extensible
 - Neutrality (from transport layer)
 - Can run on HTTP, WebSocket, even SMTP
 - Independence

```
<!-- Abstract interfaces -->
<interface name="Interface1">
  <fault name="Error1" element="tns:response"/>
  <operation name="Get" pattern="http://www.w3.org/ns/wsdl/in-out">
    <input messageLabel="In" element="tns:request"/>
    <output messageLabel="Out" element="tns:response"/>
  </operation>
</interface>

<!-- Concrete Binding Over HTTP -->
<binding name="HttpBinding" interface="tns:Interface1"
  type="http://www.w3.org/ns/wsdl/http">
  <operation ref="tns:Get" whttp:method="GET"/>
</binding>

<!-- Concrete Binding with SOAP-->
<binding name="SoapBinding" interface="tns:Interface1"
  type="http://www.w3.org/ns/wsdl/soap"
  wssoap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP/"
  wssoap:mepDefault="http://www.w3.org/2003/05/soap/mep/request-response">
  <operation ref="tns:Get" />
</binding>
```

Example WSDL

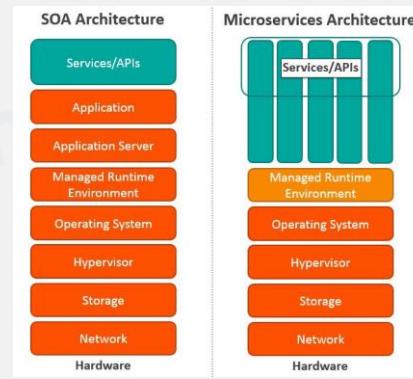
* From Wikipedia

Simple Object Access Protocol (SOAP)

- Its popularity has somewhat diminished, but still very relevant in enterprise applications
 - SOAP is still used most often in the enterprise world, where communication between different services needs to *conform to a set of rules and contracts* (*)
 - Because it follows objects, rules, and constraints, SOAP is a more strict (*) protocol than REST
 - * *But do enterprises really conform to strict rules? Agile seems to work best in practice*
 - It might have been too rigid for its own good
- E.g. Salesforce SOAP API to create, retrieve, update or delete records, such as accounts, leads, and custom objects
- E.g. The PayPal SOAP API is based on open standards known collectively as web services, which include the Simple Object Access Protocol (SOAP), Web Services Definition Language (WSDL), and the XML Schema Definition language (XSD)

SOA and MicroServices

- MicroService Architecture is very similar, modern reincarnation of SOA
 - SOA mainly 2000~2010
 - MicroServices 2015~...
 - “*Microservices are the kind of SOA we have been talking about for the last decade. Microservices must be independently deployable, whereas SOA services are often implemented in deployment monoliths.*” - Torsten Winterberg
- What has changed? Adoption of:
 - Containerization
 - Asynchronous Programming
 - Distributed Computing mindset
 - CICD and Agile workflows



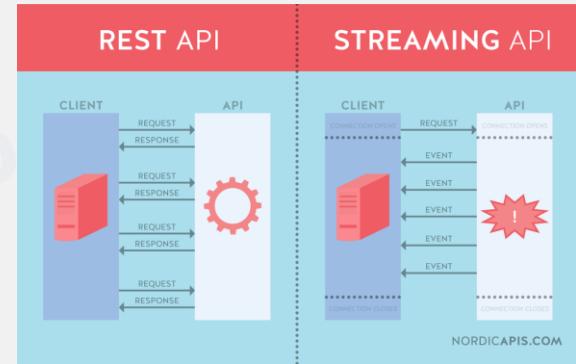


CLOUD COMPUTING APPLICATIONS

Cloud Computing Glue:
Asynchronous RPC, WebSocket
Prof. Reza Farivar

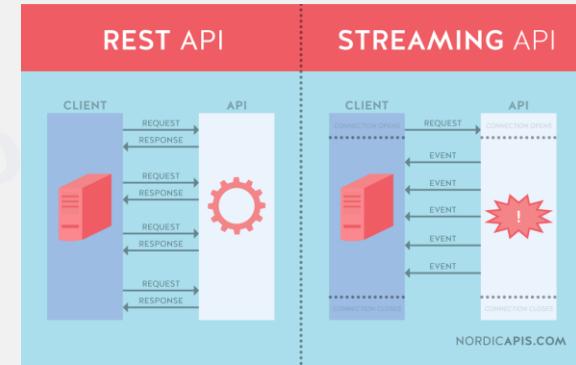
Asynchronous RPC, aka. Streaming API

- Using old HTTP/0.9 and 1.0, what if the remote server takes a long time?
 - The client can wait, blocked, keeping the HTTP connection open (long polling)
 - It can keep polling the server periodically
 - Ultimately, these are hacks
 - HTTP/1.1 not ready for real-time web
- Web 2.0: Bidirectional client / server communication
 - AJAX
 - Comet
 - Umbrella term: Ajax Push, Reverse Ajax, Two-way-web, HTTP Streaming, and HTTP server push, ...
 - Push notifications
 - XMLHttpRequest
 - XMLHttpRequest (XHR) is an API in the form of an object whose methods transfer data between a web browser and a web server.
 - The object is provided by the browser's JavaScript environment.
 - jQuery provides a nice wrapper (it also encapsulates WebSockets and server push)



WebSocket

- RPC libraries can issue asynchronous methods to the server, and the server can inform them of the response later
- Streaming Architecture
 - Minimizing latency
- WebSocket
 - Part of HTML 5 standard
 - Can handle interactive sessions better than RESTful architecture
- Use cases:
 - Chat
 - Stock price update
 - Collaborative Document Editing
 - Location update (I am here now)
 - Multiplayer games



WebSocket Protocol

- WebSocket is an application protocol, running on top of TCP
 - It uses URIs, but not http://
 - Instead, uses ws://
- It utilizes an initial HTTP session and HTTP port numbers to process handshake phase
- The protocol has 3 phases
 - Opening Handshake
 - Data Transfer
 - Closing Handshake
- Protocol “upgrade”
- Purely event driven
 - Application code listens for events on WebSocket objects to handle incoming data and changes in connection status
- Asynchronous programming
 - Client does not need to do anything (e.g. poll) to receive data

WebSocket 3 phases

1) Opening Handshake

- HTTP request/response to open WebSocket connection
- Example client request

```
GET /chat HTTP/1.1
Host: example.com
Upgrade: websocket
Connection: Upgrade
Origin: http://example.com
Sec-WebSocket-Key: dGhlIHNhbXBsZSub25jZQ==
Sec-WebSocket-Version: 13
```

- Example server response

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzhZRbK+xOo=
```

- HTTP protocol is switched (aka. upgraded) to WebSocket

2) Data Transfer

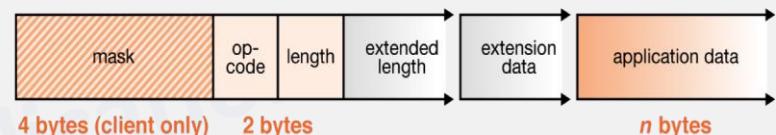
- Bidirectional communication
- WebSocket frame
- Description of fields:
 - Op-code: Continuation, Text, Binary, Close, Ping, Pong

To keep the connection alive

3) Closing handshake

- A WebSocket frame with opcode 0x8 is sent

WebSocket Frame



WebSocket API

- Simple Javascript W3C WebSocket API
- four different events:
 - Open - fires to establish a connection
 - Message - contains the data from the server
 - Error - fires in response to an unexpected event (failure)
 - Close - fires when the WebSocket connection is closed
- Primary methods + events
 - **WebSocket (URL, [protocols]) – Create a connection**
 - `var ws = new WebSocket("ws://www.websocket.org", "SOAP"); // "SOAP" is optional`
 - **onOpen() – WebSocket opened**
 - `ws.onOpen = function(e){ console.log(ws.protocol); }`
 - **Send (data) – Send data (string, Blob or ArrayBuffer)**
 - `ws.send("Hello WebSocket!");`
 - **onMessage () – Message received**
 - `ws.onMessage = function(e){ log("Message received: " + e.data); ws.close(); }`
 - **onClose () – Close message received**
 - `ws.onOpen = function(e){ console.log ("Disconnected: " + e.reason); }`
 - **onError () - Error**

WebSocket

- Note that once you allow asynchronous communication in a networked environment, you should handle faults
- There are wrapper packages, encapsulating WebSockets functionality with additional features
 - Node.js supports WebSockets through plugins
 - Example: Socket.io.
 - Consisting of a Node.js server and a Javascript client library, socket.io provides reliability for handling proxies and load balancers as well as personal firewall and antivirus software and even supports binary streaming.
- Java 11 supports both HTTP as well as WebSocket protocols

WebSocket in Cloud Computing

- Many Cloud providers support WebSocket as the API of choice for interactive sessions with the Cloud service and the client
 - Amazon AWS API Gateway
 - Salesforce
 - Many cloud video vendors
 - E.g. easylive.io
 - Basis of Slack and its “Real Time Messaging API”
 - Slack has a simpler Event API based on HTTP/2 Push, but only the RTM guarantees real time delivery of messages
 - Google App Engine Channel API used WebSocket to allow server to client messaging
 - *Now deprecated, replaced by Firebase (a MBaaS solution)*
 - We can always run a Socket.io instance on Computer Engine (or EC2) instance
 - <https://cloud.google.com/solutions/real-time-gaming-with-node-js-websocket>
 - Or use AWS API Gateway to handle the deployment of WebSocket servers for us



CLOUD COMPUTING APPLICATIONS

Cloud Computing Glue: HTTP2
Push, Streaming Video
Prof. Reza Farivar

HTTP/2

- Published in 2015, most browsers supported it by the end of 2015
 - By 2020, about ~42% of top 10 million websites supported HTTP/2
- One of the main features of HTTP/2 is server push
 - Closely related to another feature of HTTP/2: data streaming
 - HTTP/2 Server push is being progressively implemented, for example Nginx web server implemented it in February 2018.
 - By now, all major servers and browsers support it
- It “proposed” new data in a new stream to the browser, to be stored in a Cache
- It does not send the pushed data directly to the client application itself
- To make the application aware, HTTP/2 utilizes Server-Side Events (SSE)

Amazon AWS API Gateway

- REST API
- WebSocket API
- HTTP API
 - Based on HTTP/2 push and notification

Video Streaming over the Internet

- Streaming Video Content
 - RTMP
 - Still very prevalent, but slowly being phased out
 - RTP (over UDP)
 - HLS (over HTTP)
 - MPEG-DASH
 - WebRTC



CLOUD COMPUTING APPLICATIONS

VPC: Virtual Private Cloud

Prof. Reza Farivar

Virtual Private Clouds

- Most Cloud Providers have some sort of network virtualization solution
 - Arguably the most fundamental building block
 - Amazon Virtual Private Cloud (VPC)
 - Microsoft Azure Virtual Network (VNet)
 - Google Virtual Private Cloud (VPC)
 - Oracle Virtual Cloud Networks
- They somewhat differ in detail, but the concepts are general
- In this lesson we will focus on Amazon VPC
 - Geared towards Cloud Architecture

Solving a Fundamental Problem

- Allow many different users have their own private network in the cloud
- Isolate different customers' network packets from each other
- Solution: VPC

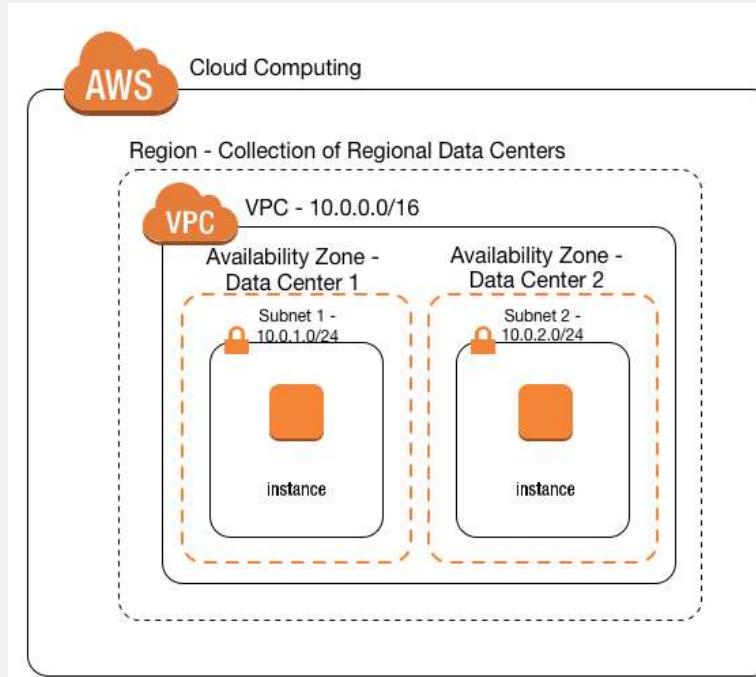
VPCs and Subnets

- You have your “own” network in the cloud
 - VPC
 - In AWS, a VPC is associated with a region, e.g. us-east-1
 - You can have more than one VPC, even in the same region
 - Default 5 quota
 - The IP address range of all the nodes in this VPC can be defined with a CIDR
- Your VPC is subdivided into logically separate segments
 - Subnet
 - Each subnet get a smaller CIDR range
 - Each subnet is associated with one Availability Zone
- You can then launch instances (EC2, RDS, etc.) in a subnet

VPCs, Subnets and Availability Zones

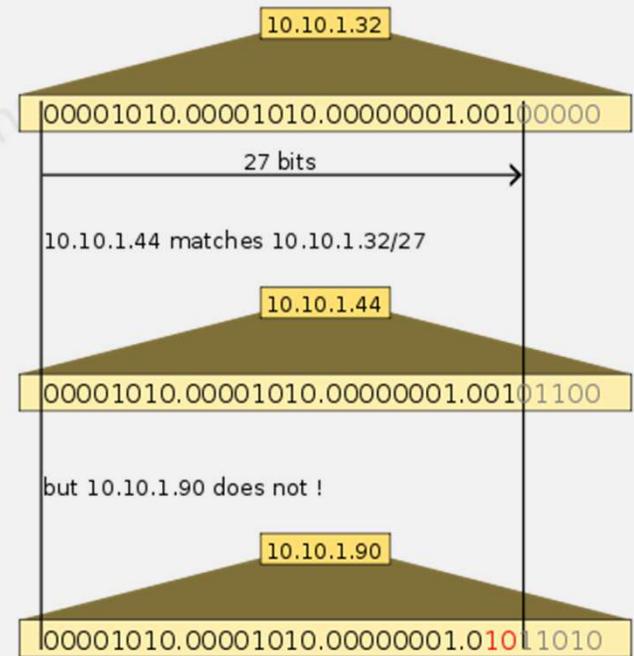
The main routing table, associated with the VPC, has the following route:

Destination	Target
10.0.0.0/16	local



Background Knowledge: CIDR

- CIDR: Classless Inter-Domain Routing
- A method of allocating IP address ranges
- In IPV4, each IP address is a 32 bit value
 - 4 bytes
 - 192.168.0.1
- CIDR notation:
 - 100.101.102.103/24
 - Take the mask (24 bits)
 - Keep the upper 24 bits the same
 - The lower bits can change → range
 - 100.101.102.0 ... 100.101.102.255
- IPV6, each address is 128 bits:
 - the IPv6 block $2001:db8::/48$ represents the block of IPv6 addresses from $2001:db8:0:0:0:0:0:0$ to $2001:db8:0:ffff:ffff:ffff:ffff:ffff$.
 - Note that in IPV6 notation, each segment is written in hex



RFC 1918: Address Allocation for Private Internets

- The Internet Assigned Numbers Authority (IANA) has reserved the following three blocks of the IP address space for private internets
- 10.0.0.0/8
 - 10.0.0.0 - 10.255.255.255
 - Number of addresses: 16,777,216
- 172.16.0.0/12
 - 172.16.0.0 - 172.31.255.255
 - Number of addresses: 1,048,576
- 192.168.0.0/16
 - 192.168.0.0 - 192.168.255.255
 - Number of addresses: 16,777,216
- Why? Because it is guaranteed no other server on the public internet has an IP address in these ranges
- Routing rules do not conflict

RFC 1918 and AWS VPC

- When you create a VPC, you must specify an IPv4 CIDR block for the VPC
- The allowed block size is between a /16 netmask (65,536 IP addresses) and /28 netmask (16 IP addresses)
 - Note that RFC 1918 would allow for 16 million distinct IP addresses in the 10.0.0.0/8, but Amazon would at most accept a /16 netmask in a VPC or subnet

RFC 1918 range	Example AWS VPC CIDR block
10.0.0.0 - 10.255.255.255 (10/8 prefix)	Your VPC must be /16 or smaller, for example, 10.0.0.0/16.
172.16.0.0 - 172.31.255.255 (172.16/12 prefix)	Your VPC must be /16 or smaller, for example, 172.31.0.0/16.
192.168.0.0 - 192.168.255.255 (192.168/16 prefix)	Your VPC can be smaller, for example 192.168.0.0/20.

Reserved IP Addresses

- The first four IP addresses and the last IP address in each subnet CIDR block are not available for you to use, and cannot be assigned to an instance; E.g. for 10.0.0.0/24:
 - 10.0.0.0: Network address
 - 10.0.0.1: The VPC router
 - 10.0.0.2: The IP address of the DNS server is the base of the VPC network range plus two
 - 10.0.0.3: Reserved for future use
 - 10.0.0.255: Network broadcast address
 - AWS does NOT support broadcast in a VPC, therefore this address is reserved

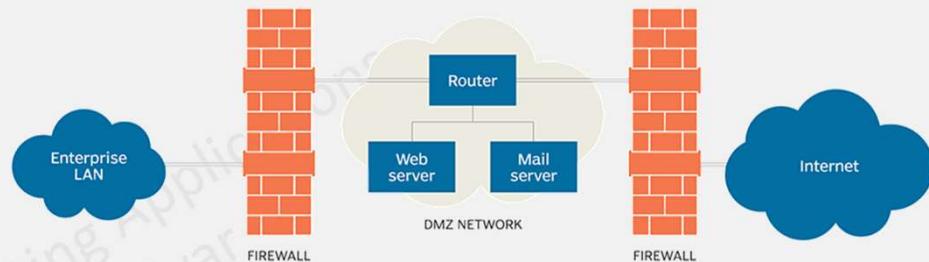


CLOUD COMPUTING APPLICATIONS

VPC: Subnets
Prof. Reza Farivar

Subnets

- Create subnets to isolate resources per the project requirement
 - DMZ/Proxy
 - Load balancer
 - web applications
 - Mail servers
 - Databases
- E.g. have a public subnet to host internet-facing resources and a private subnet for databases that accept web requests
- Create multiple subnets (public or private) in multiple AZs to host a high availability multi-AZ infrastructure and avoid a single point of failure
 - each subnet can communicate with every other subnet in the same VPC



Private Subnets

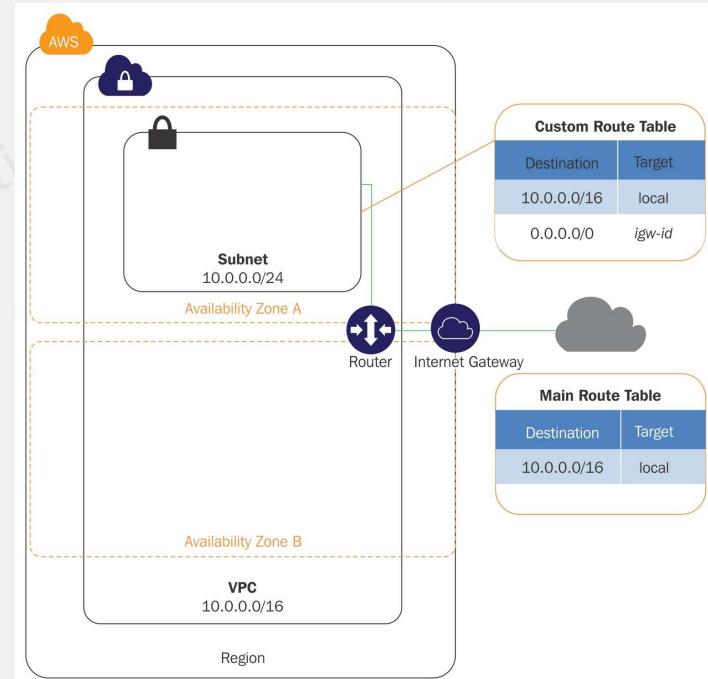
- Any incoming traffic from the internet cannot directly access the resources within a private subnet
- Outgoing traffic from a private subnet cannot directly access the internet
 - Restricted; or
 - Routed through a NAT
- Each resource (instance) gets a private IP
 - From the CIDR range associated with the subnet
- Technically, a subnet is private if there is no route in the routing table to an internet gateway

Public Subnet

- A subnet that has access to an internet gateway defined in the routing table
- Each resource in a public subnet gets a private IP within the CIDR range, AND a public IP accessible from the internet
 - the public IP can be dynamic (only remains valid while the instance is alive, and then AWS reclaims it), or
 - It can be an elastic IP, where you pay for it, and it will remain yours even if the instance shuts down
- Outgoing traffic can directly access internet
 - Unlike Private, which needs a NAT to access internet

Route Tables

- Each VPC has an associated “Main Route Table”
 - The Default VPC has a route in its “Main Route Table” to an internet gateway
 - Custom VPCs usually only have the local route in the MRT
- Subnets can have their own custom route table
 - If no custom route table is explicitly associated with a subnet, then it is associated with the VPC's main route table
- Possible route table targets
 - Local, IGW, a NAT device, A VGW, a peering connection, or a VPC endpoint (e.g. S3)
- Selection of the optimum route for network traffic is done based on the longest prefix match
 - the most specific routes that match the network traffic





CLOUD COMPUTING APPLICATIONS

VPC: Gateways

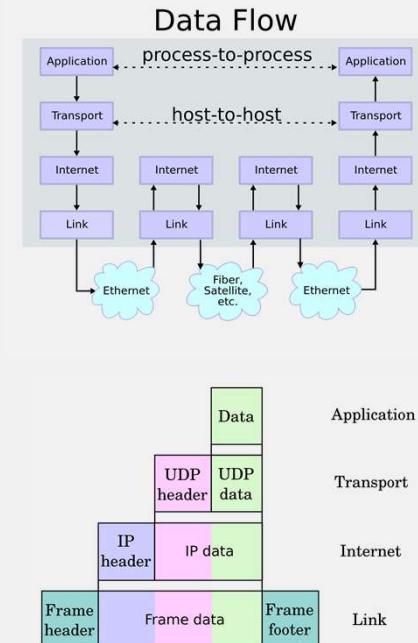
Prof. Reza Farivar

Internet Gateways

- Internet Gateway is a logical construct, not a specific instance or resource
- AWS does quite a bit of behind the scene work to allow highly available internet to all the required Availability Zones in the VPC
- Is attached to a VPC
- Highly available, redundant, and horizontally scaled
- → in the route tables, it is referred to by its name (e.g. [igw-05ae7f551a8154d1a](#)), not an IP address

NAT Gateways

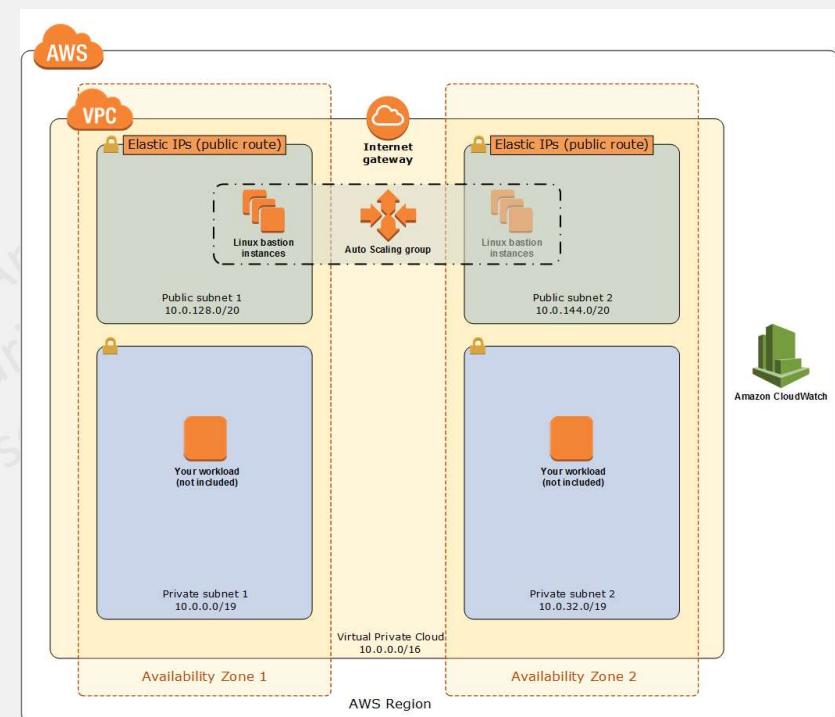
- Network Address Translation
 - Just like your home wireless router
- Virtual router or a gateway in a public subnet that enables instances in a private subnet to interact with the internet
 - IPv4 only
- Modifies the network address information in the IP header
 - It receives traffic from an EC2 instance residing in a private subnet
 - before forwarding the traffic to the internet, replaces the reply-to IPv4 address with its own public or Elastic IP address
 - When a reply is received from internet, it changes the reply-to address from its IP address to the EC2 instance private IP address
- Two types of NAT
 - NAT Instance → Runs as an EC2 instance
 - NAT Gateway → fully managed by AWS, requires elastic IP
 - Better availability and higher bandwidth



Bastion Host

- Use a bastion host to access private machines hosted in a private network in a VPC
- Bastion host: “a server whose purpose is to provide access to a private network from an external network, such as the Internet. Because of its exposure to potential attack, a bastion host must minimize the chances of penetration”

Further reading: <https://cloudacademy.com/blog/aws-bastion-host-nat-instances-vpc-peering-security/>





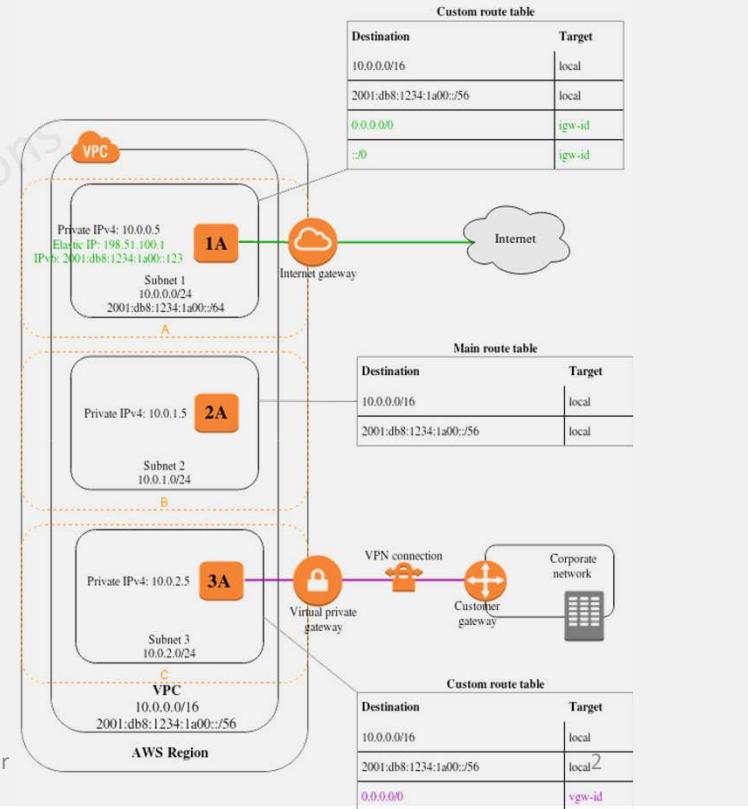
CLOUD COMPUTING APPLICATIONS

VPC: Advanced VPC

Prof. Reza Farivar

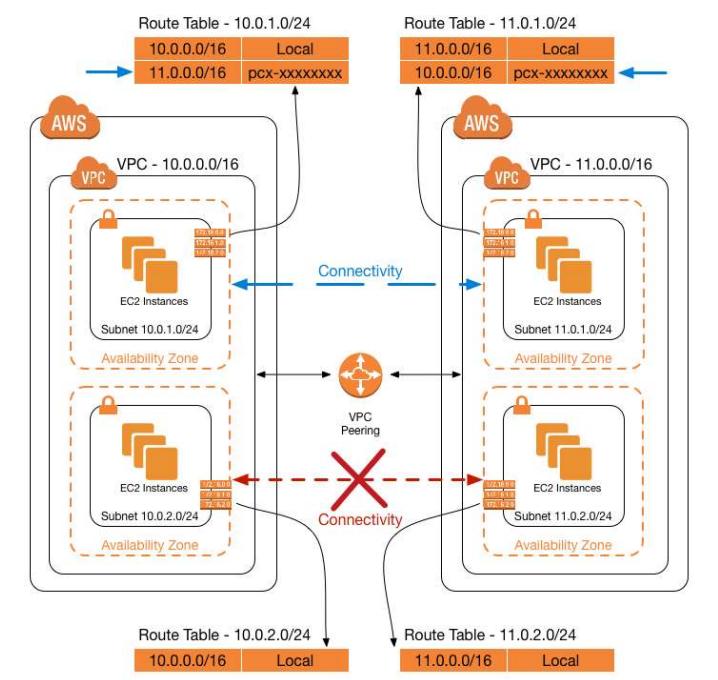
Virtual Private Gateway

- If a subnet doesn't have a route to the internet gateway, but has its traffic routed to a virtual private gateway for a Site-to-Site VPN connection, the subnet is known as a *VPN-only subnet*
- In this diagram, subnet 3 is a VPN-only subnet
- Concepts
 - VPN connection:** A secure connection between your on-premises equipment and your VPCs.
 - VPN tunnel:** An encrypted link where data can pass from the customer network to or from AWS.
 - Each VPN connection includes two VPN tunnels which you can simultaneously use for high availability.
 - Customer gateway:** An AWS resource which provides information to AWS about your customer gateway device.
 - Customer gateway device:** A physical device or software application on your side of the Site-to-Site VPN connection.
 - Virtual private gateway:** The VPN concentrator on the Amazon side of the Site-to-Site VPN connection. You use a virtual private gateway or a transit gateway as the gateway for the Amazon side of the Site-to-Site VPN connection.
 - Transit gateway:** A transit hub that can be used to interconnect your VPCs and on-premises networks. You use a transit gateway or virtual private gateway as the gateway for the Amazon side of the Site-to-Site VPN connection.
- As of 2020, VPN connections into AWS are IPv4 only
- It is recommended that you use non-overlapping CIDR blocks for your networks



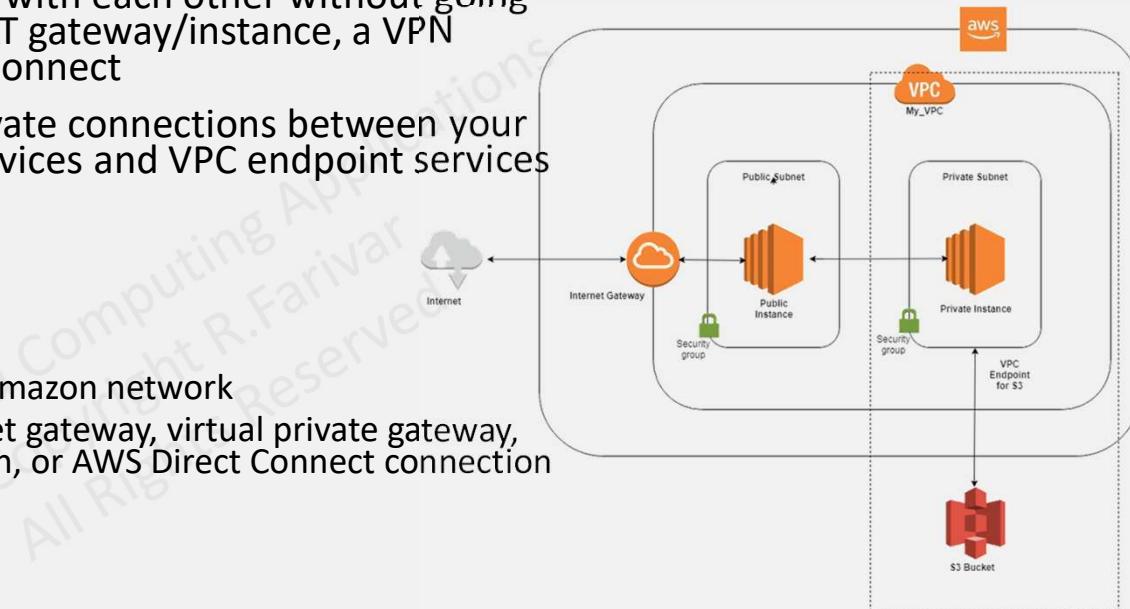
VPC Peering

- VPC peering can be used to make communication between VPCs within the same account, different AWS accounts, or any two VPCs within the same region or different regions
- Initially, VPC peering was supported only within the same region, but later AWS added support for VPC peering across regions
- The two VPCs cannot have CIDR blocks that overlap with each other.



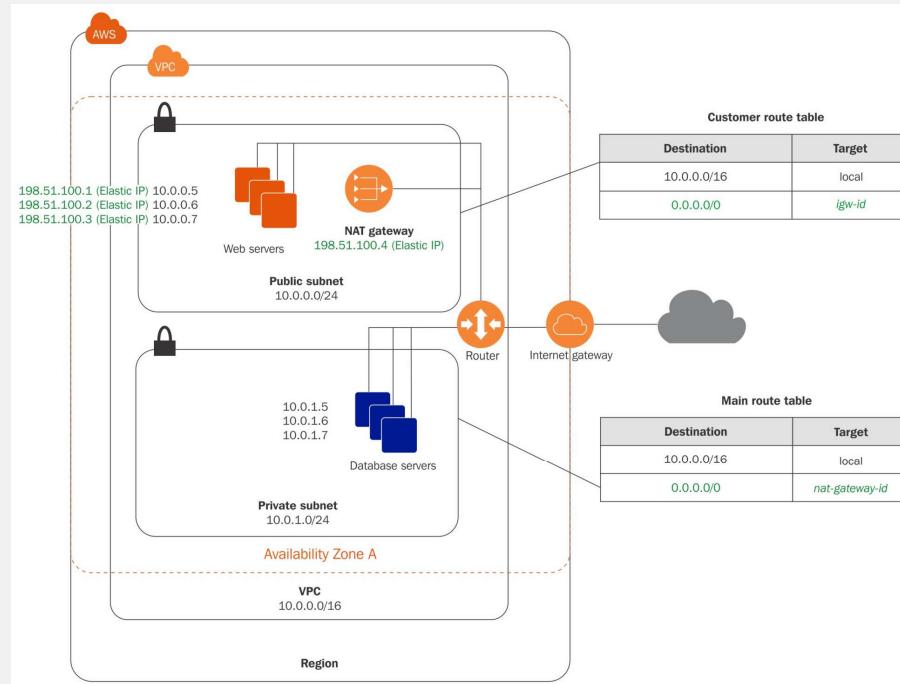
VPC Endpoints

- Generally, AWS services are different entities and do not allow direct communication with each other without going through either an IGW, a NAT gateway/instance, a VPN connection, or AWS Direct Connect
- A VPC endpoint enables private connections between your VPC and supported AWS services and VPC endpoint services
 - S3
 - DynamoDb
- AWS PrivateLink
 - Private IP addresses
 - Traffic does not leave the Amazon network
 - Does not require an internet gateway, virtual private gateway, NAT device, VPN connection, or AWS Direct Connect connection



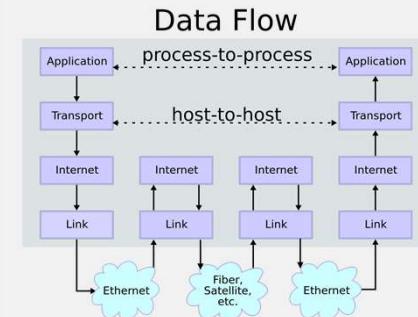
* Interesting reading: <https://www.bluematador.com/blog/s3-endpoint-connectivity-in-aws-vpc>

VPC with Private and Public Subnets



Routing in VPC vs. Physical Network

- Physical Ethernet Network
 - Link Layer
 - Lowest layer in the Internet Protocol
 - Layer 2 in OSI model
 - In a physical traditional network, this layer uses MAC address and ARP messaging (to discover unknown MAC addresses)
 - VPC Network
 - Amazon backend intercepts any MAC ARP request
 - Looks up routing tables, and returns the destination without implementing ARP





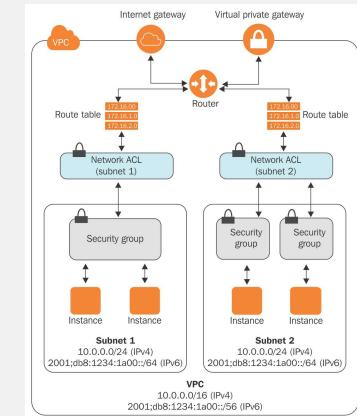
CLOUD COMPUTING APPLICATIONS

VPC: Security and Firewalls

Prof. Reza Farivar

Security and Firewalls

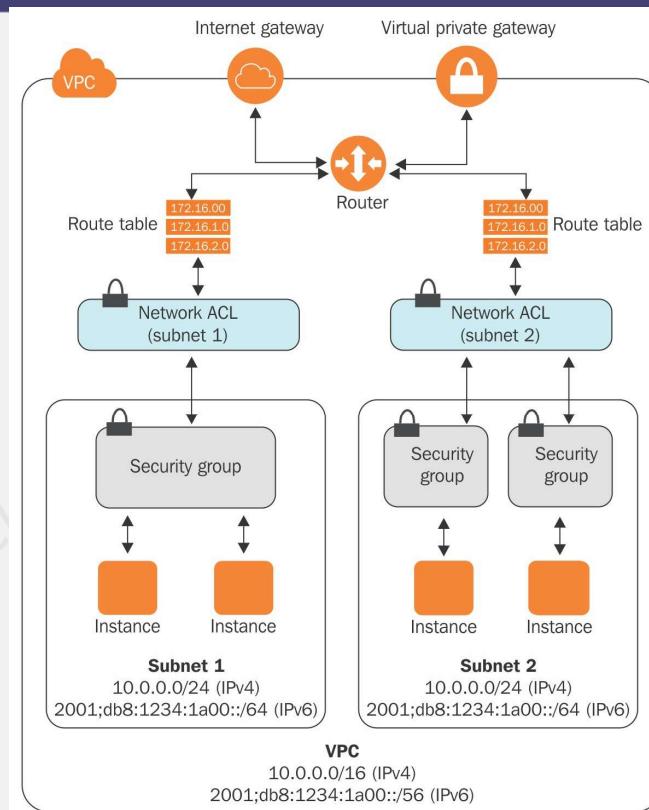
- Security
 - Security Groups
 - EC2 instance-level firewall
 - Network Access Control Lists (NACL)
 - Subnet firewall
- Monitoring
 - Flow Logs
 - Enable VPC flow logs for audit purposes
 - Study flow logs from time to time
 - highlights unauthorized attempts to access the resources



Security

- Make sure that only required ports and protocols from trusted sources can access AWS resources using security groups and NACLs
- Make sure that unwanted outgoing ports are not open in security groups
 - A security group for a web application does not need to open incoming mail server ports

Security and Firewalls



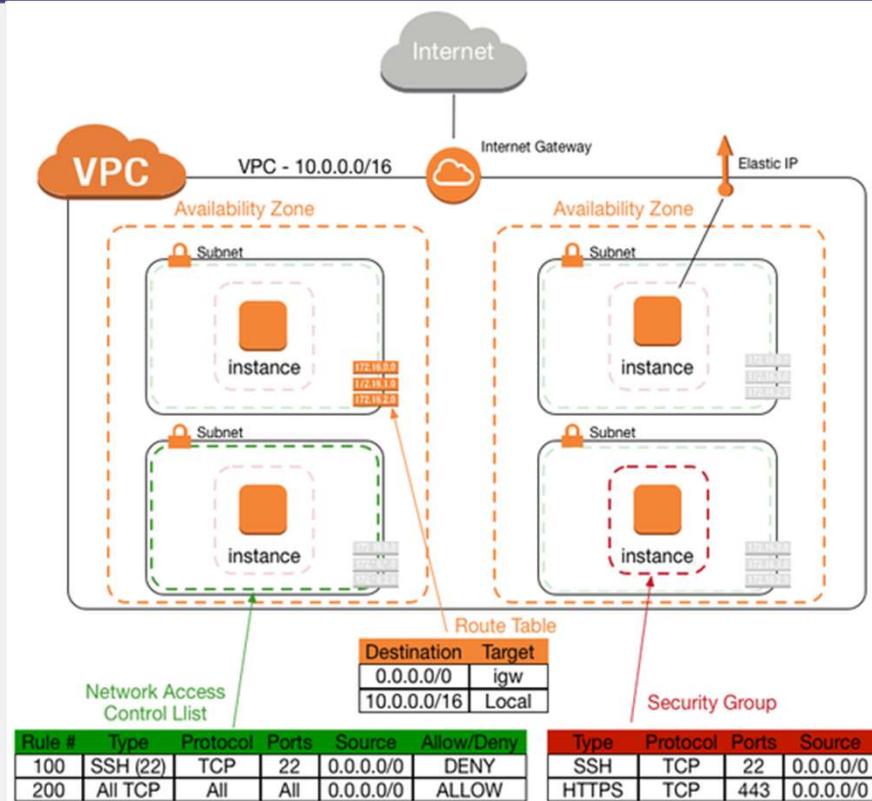
Network Access Control List

Inbound rules (2)							Edit inbound rules	
Rule number	Type	Protocol	Port range	Source	Allow/Deny			
100	All traffic	All	All	0.0.0.0/0	<input checked="" type="checkbox"/> Allow			
*	All traffic	All	All	0.0.0.0/0	<input checked="" type="checkbox"/> Deny			

Outbound rules (2)							Edit outbound rules	
Rule number	Type	Protocol	Port range	Destination	Allow/Deny			
100	All traffic	All	All	0.0.0.0/0	<input checked="" type="checkbox"/> Allow			
*	All traffic	All	All	0.0.0.0/0	<input checked="" type="checkbox"/> Deny			

- NACL acts as a virtual firewall at the subnet level
- Every VPC has a default NACL
- Every subnet, whether it is private or public in a VPC, must be associated to one NACL
- One NACL can be associated with one or more subnets; but each subnet can have ONE NACL associated with it
- NACL rules are evaluated based on its rule numbers. It evaluates the rule starting from the lowest number to the highest number
- NACL is stateless:
 - Separate rules to allow or deny can be created for inbound and outbound traffic
 - If a port is open for allowing inbound traffic, it does not automatically allow outbound traffic
- The default NACL for any VPC contains a rule numbered as * in both inbound and outbound rules
 - This rule appears and executes last

Anatomy of a VPC with Route Table, Network ACL and Security Group



Security Group

Inbound rules					Edit inbound rules
Type	Protocol	Port range	Source	Description - optional	
HTTP	TCP	80	0.0.0.0/0	-	
HTTP	TCP	80	::/0	-	
Custom TCP	TCP	8080	0.0.0.0/0	-	
Custom TCP	TCP	8080	::/0	-	
SSH	TCP	22	0.0.0.0/0	-	
SSH	TCP	22	::/0	-	
HTTPS	TCP	443	0.0.0.0/0	-	
HTTPS	TCP	443	::/0	-	

Outbound rules					Edit outbound rules
Type	Protocol	Port range	Destination	Description - optional	
All traffic	All	All	0.0.0.0/0	-	

- Firewall at the instance level
- One or more security groups can be associated with each EC2 instance
- A security group can be attached to many EC2 instances
- Each SG contains rules allowing inbound and outbound traffic
- Using CIDR notation, a source IP can be fixed to a particular IP, such as 10.108.20.107/32
- Any source IP can be allowed by a 0.0.0.0/0

Security Group as Source IP

Inbound rules					Edit inbound rules
Type	Protocol	Port range	Source	Description - optional	
Custom TCP	TCP	4003 - 65535	sg-00fbbeba74e062d16 (nodes.dev.k8s.mp3-k8.in)	-	
Custom TCP	TCP	2382 - 4000	sg-00fbbeba74e062d16 (nodes.dev.k8s.mp3-k8.in)	-	
All traffic	All	All	sg-003e7c9121913dca3 (masters.dev.k8s.mp3-k8.in)	-	
SSH	TCP	22	0.0.0.0/0	-	
Custom UDP	UDP	1 - 65535	sg-00fbbeba74e062d16 (nodes.dev.k8s.mp3-k8.in)	-	
Custom TCP	TCP	1 - 2379	sg-00fbbeba74e062d16 (nodes.dev.k8s.mp3-k8.in)	-	
HTTPS	TCP	443	0.0.0.0/0	-	

- A security group ID can be specified as a source IP to allow communication from all the instances that are attached to that security group
- For example, in the case of autoscaling, the number of EC2 instances and their IP addresses keeps changing.
- In such situations, it is best practice to attach a security group to such EC2 instances with the help of an autoscaling template and place a security group ID as a source IP in another security group.



CLOUD COMPUTING APPLICATIONS

SOFTWARE DEFINED ARCHITECTURE

Roy Campbell & Reza Farivar

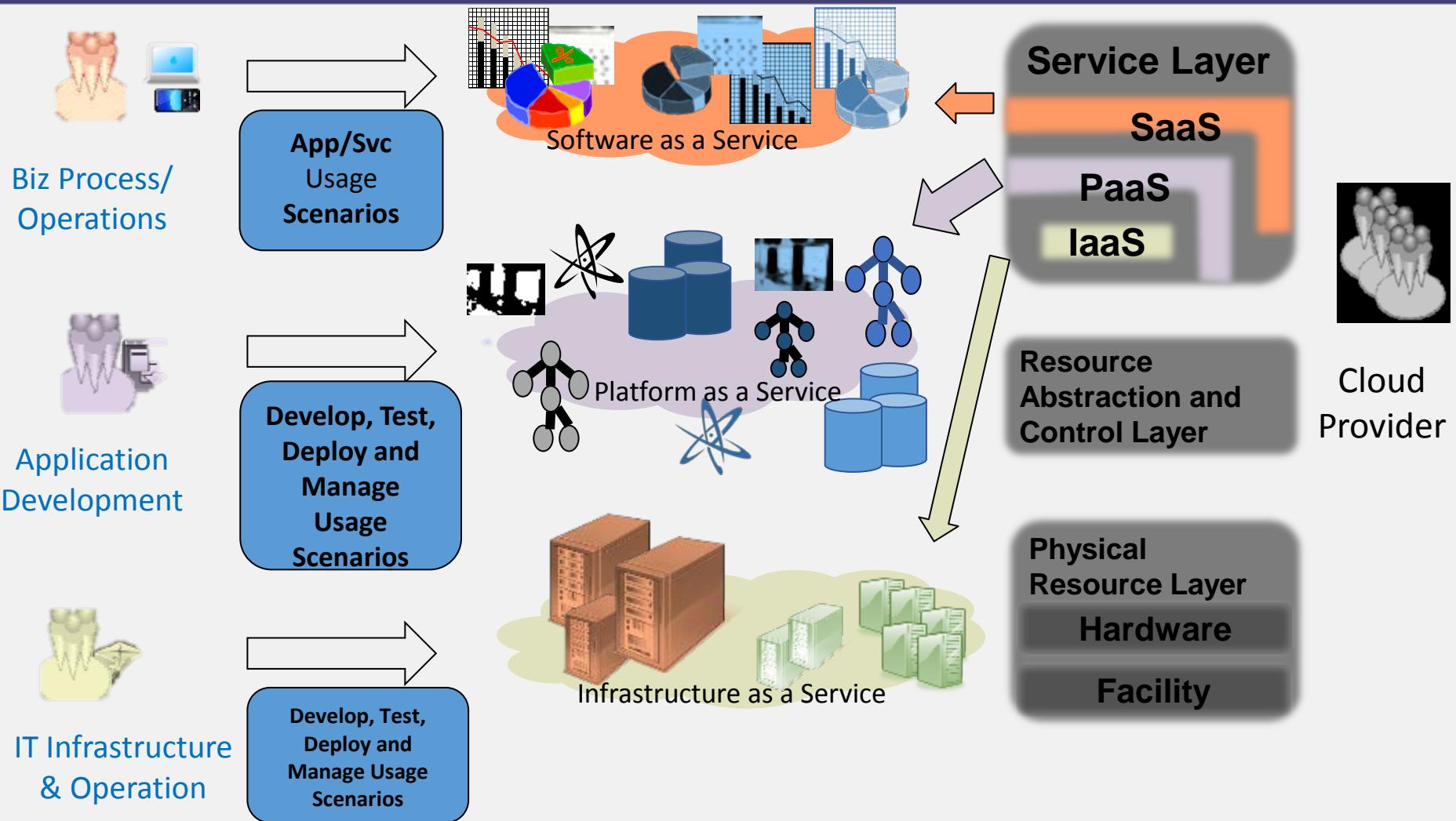
Learning Objectives

- How services are created
- How services can control other services
- The principal architectural components of a cloud and their organization
- How services and orchestration play a role in each layer of a production cloud: IAAS, PAAS, SAAS

Software Defined Architecture

- Cloud provides services, service orchestration, and provisioning
- A Cloud may provide IAAS, PAAS, SAAS and have both internal and external Application Programming Interfaces
- The mechanisms and concept of providing services, orchestration, and provisioning is called a Software Defined Architecture
- A Cloud may contain other software defined entities:
 - Software Defined Network
 - Software Defined Storage
 - Software Defined Compute

Cloud Provider: Service Orchestration

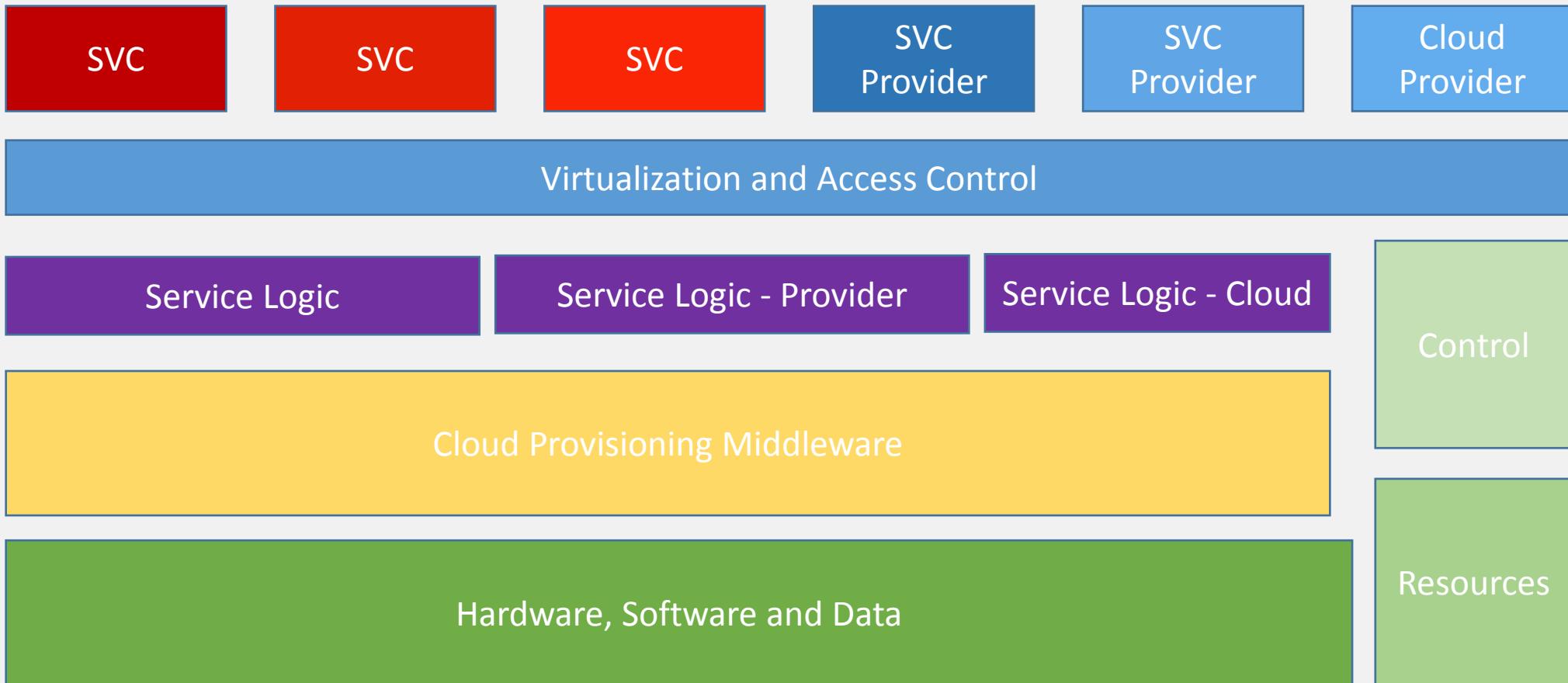


Orchestration

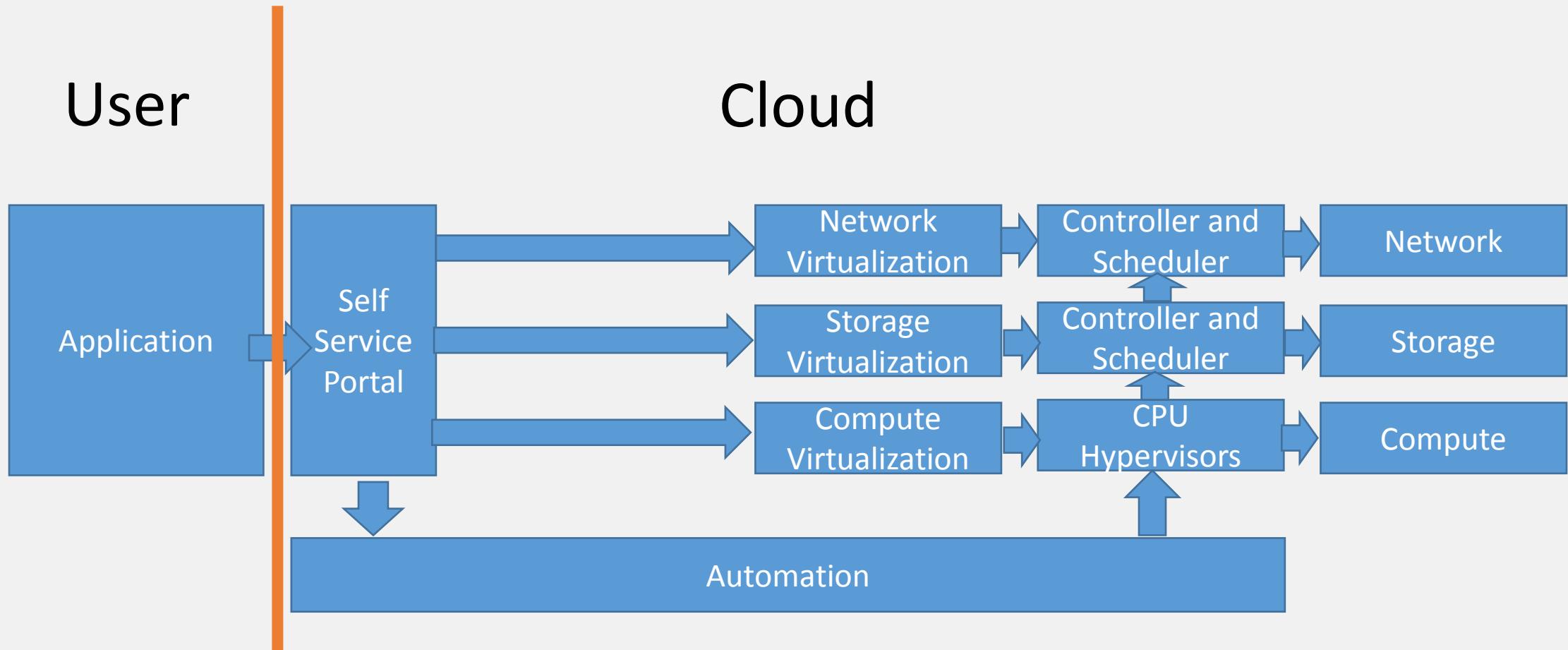
Cloud service orchestration is the:

- *Composing* of architecture, tools and processes used by humans to deliver a defined Service.
- *Stitching* of software and hardware components together to deliver a defined Service.
- *Connecting* and *Automating* of work flows when applicable to deliver a defined Service.
- Provides: up and down scaling, assurance, billing, workflows

Software Defined Architecture



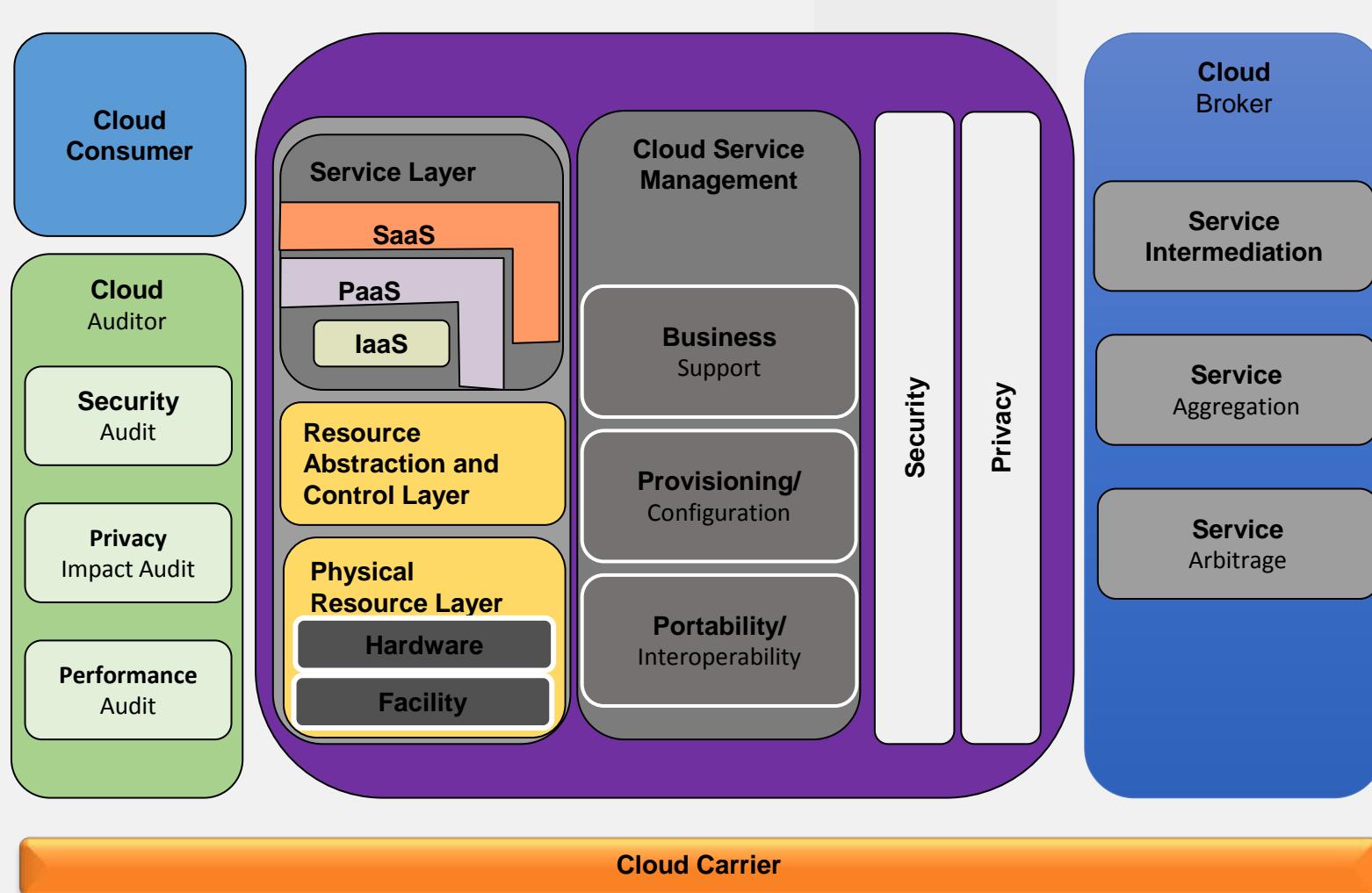
Software Defined Data Center



Content and Learning Objectives

- 1) Virtualization is a key abstraction in building software defined architectures:
 - 1) Software Defined Networks
 - 2) Software Defined Storage
 - 3) Software Defined Compute
- 2) Web Service: A Simple Application built on a Data Center
- 3) Load Balancing: A simple scheme to distribute the load of multiple servers
- 4) Infrastructure as a Service
- 5) Mirantis and OpenStack
- 6) How systems are structured with orchestration

The Combined Conceptual Reference Diagram





CLOUD COMPUTING APPLICATIONS

Cloud Services

Prof. Roy Campbell

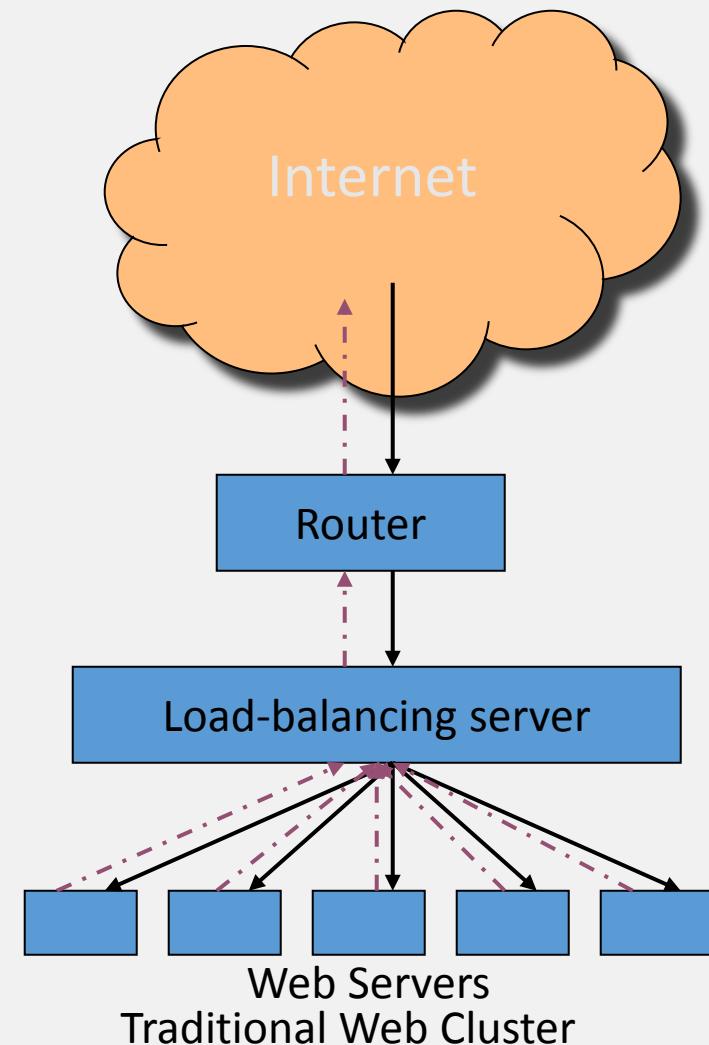
Contents

- Web Services
- Remote Procedure Calls
 - RMI, SOAP
- HTTP, REST
- JSON, XML
- Load Balancing

Introduction to Web Servers and Load Balancing

- Request enters a router
- Load balancing server determines which web server should serve the request
- Sends the request to the appropriate web server

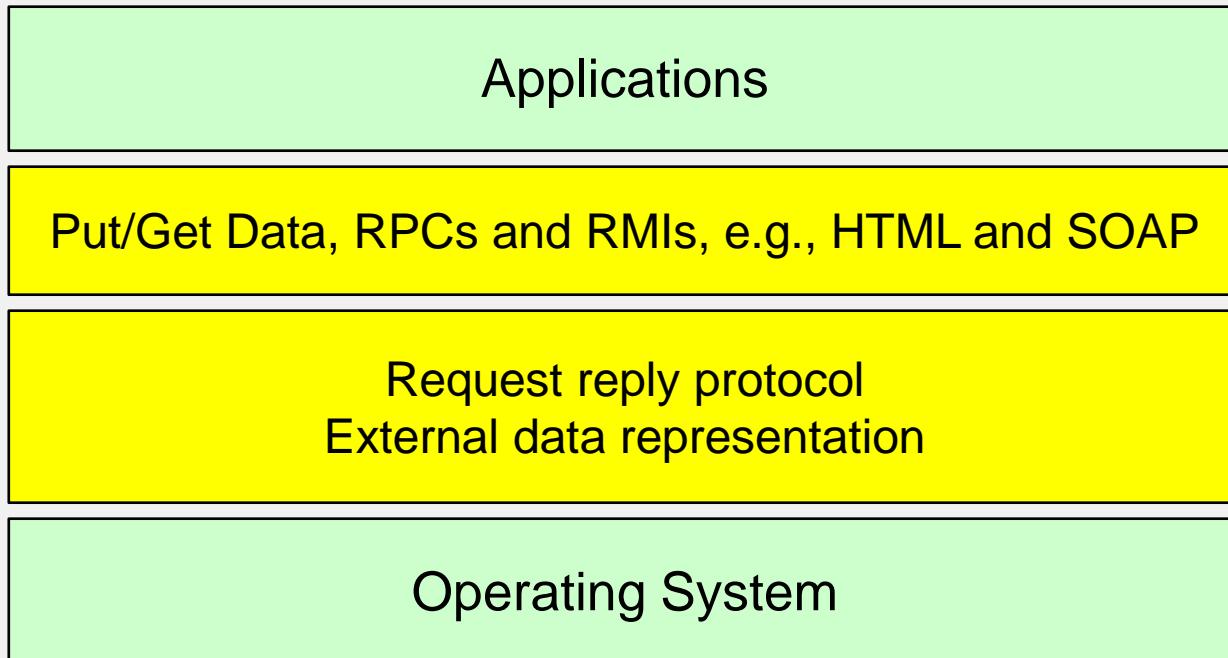
→ Request
- - - → Response



Middleware Layer Definition

- *Software that provides services to applications beyond those generally available at the operating system*
- Middleware implements functionalities that are common across many different applications
 - No need to reinvent the wheel (e.g., message parsing) every time you need to do something
- Building distributed systems while maintaining our code is not very different from a single-node program

Middleware Layers



**Middleware
layers**

***Provide support to
the application***

**Run at all servers
at user level**

RMI = Remote Method Invocation

CORBA = Common Object Request Brokerage Architecture

SOAP = Simple Object Access Protocol

SOAP — Simple Object Access Protocol

- Transmitted by HTTP or SMTP (or many others)
- Coded in XML (Can be decoded on any machine)
- Return value: Any XML document
- Underlies Web Services Description Language (WSDL)

Common Abstractions

- In single-node programs, we often rely on concepts such as
 - Procedure calls
 - Objects
 - Shared memory
 - ...
- A Middleware Layer can provide the same abstractions to distributed applications!
- We'll start with **objects** and **procedure calls**

Local Objects

- Within one process' address space
- Object
 - Consists of a set of data and a set of methods
 - For example, C++ object, Chord object at a client (Chord data structures + functions at a node)
- Object reference
 - An identifier by which objects can be accessed, i.e., a *pointer*
- Interface
 - Provides a definition of the signatures of a set of methods (i.e., the types of their arguments, return values, and exceptions) without specifying their implementation
 - For example, {put(objectname), get(objectname)} interface for Chord object. Same interface also applies to other P2P objects such as Gnutella, Kazaa, etc.

Remote Objects

- May cross multiple process' address spaces
- Remote method invocation
 - Method invocations between objects *in different processes (processes may be on the same or different host)*
 - Remote Procedure Call (RPC): procedure call between functions on different processes in non-object-based system
- Remote objects
 - Objects that can receive remote invocations
- Remote object reference
 - An identifier that can be used globally *throughout a distributed system* to refer to a particular unique remote object
- Remote interface
 - Every remote object has a remote interface that specifies which of its methods can be invoked remotely, e.g., CORBA interface definition language (IDL)



CLOUD COMPUTING APPLICATIONS

RPC IMPLEMENTATION

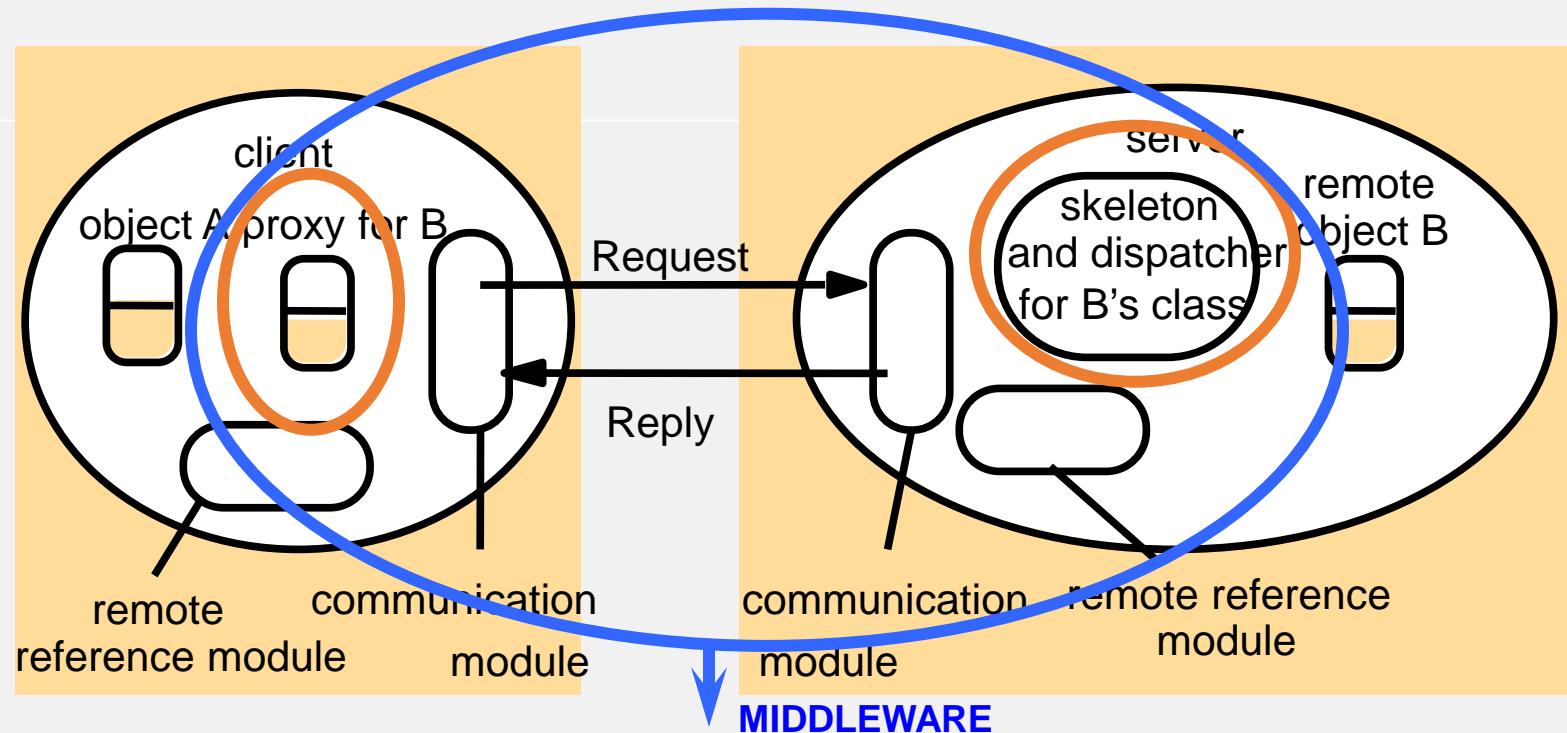
Prof. Roy Campbell

Contents

- Remote Procedure Calls
 - RMI, SOAP

How Do We Implement the Abstractions?

What should the middleware do to make the call appear similar to a local call?



- **Proxy on the “client” (process P1) side**
- **Skeleton and dispatcher on the “server” (process P2) side**

Proxy

- Is responsible for making RMI transparent to clients by behaving like a local object to the invoker
 - The proxy *implements* (Java term, not literally) the methods in the interface of the remote object that it represents. But...
- Instead of executing an invocation, the proxy forwards it to a remote object
 - On invocation, a method of the proxy *marshals* the following into a request message: (i) a reference to the target object, (ii) its own method id and (iii) the argument values. Request message is sent to the target, then proxy awaits the reply message, *unmarshals* it, and returns the results to the invoker
 - Invoked object unmarshals arguments from request message, and when done, marshals return values into reply message

Marshalling / Unmarshalling

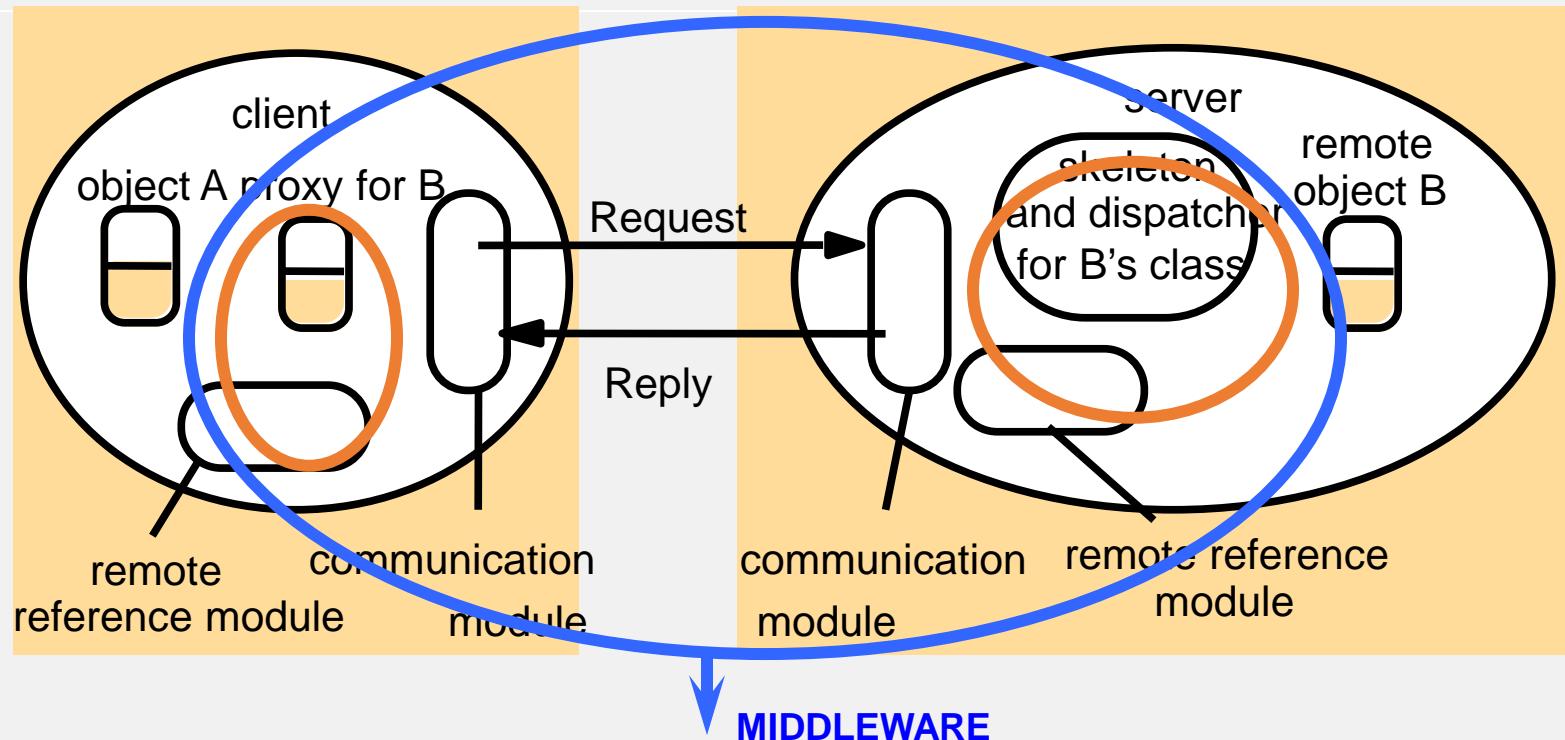
- **External data representation:** an agreed, platform-independent, standard for the representation of data structures and primitive values
 - CORBA Common Data Representation (CDR)
 - Allows an ARM client (possibly big endian) to interact with a x86 Unix server (little endian).
- **Marshalling:** the act of taking a collection of data items (platform dependent) and assembling them into the external data representation (platform independent)
- **Unmarshalling:** the process of disassembling data that is in external data representation form into a locally interpretable form

Remote Reference Module

- Is responsible for translating between local and remote object references and for creating remote object references
- Has a *remote object table*
 - An entry for each remote object held by any process (e.g., B at P2)
 - An entry for each local proxy (e.g., proxy-B at P1)
- When the remote reference module sees a new remote object, it creates a remote object reference and adds it to the table
- When a remote object reference arrives in a request or reply message, the remote reference module is asked for the corresponding local object reference, which may refer to either a proxy or to a remote object
- In case the remote object reference is not in the table, the RMI software creates a new proxy and asks the remote reference module to add it to the table

How Do We Implement the Abstractions?

What should the middleware do to make the call appear similar to a local call?

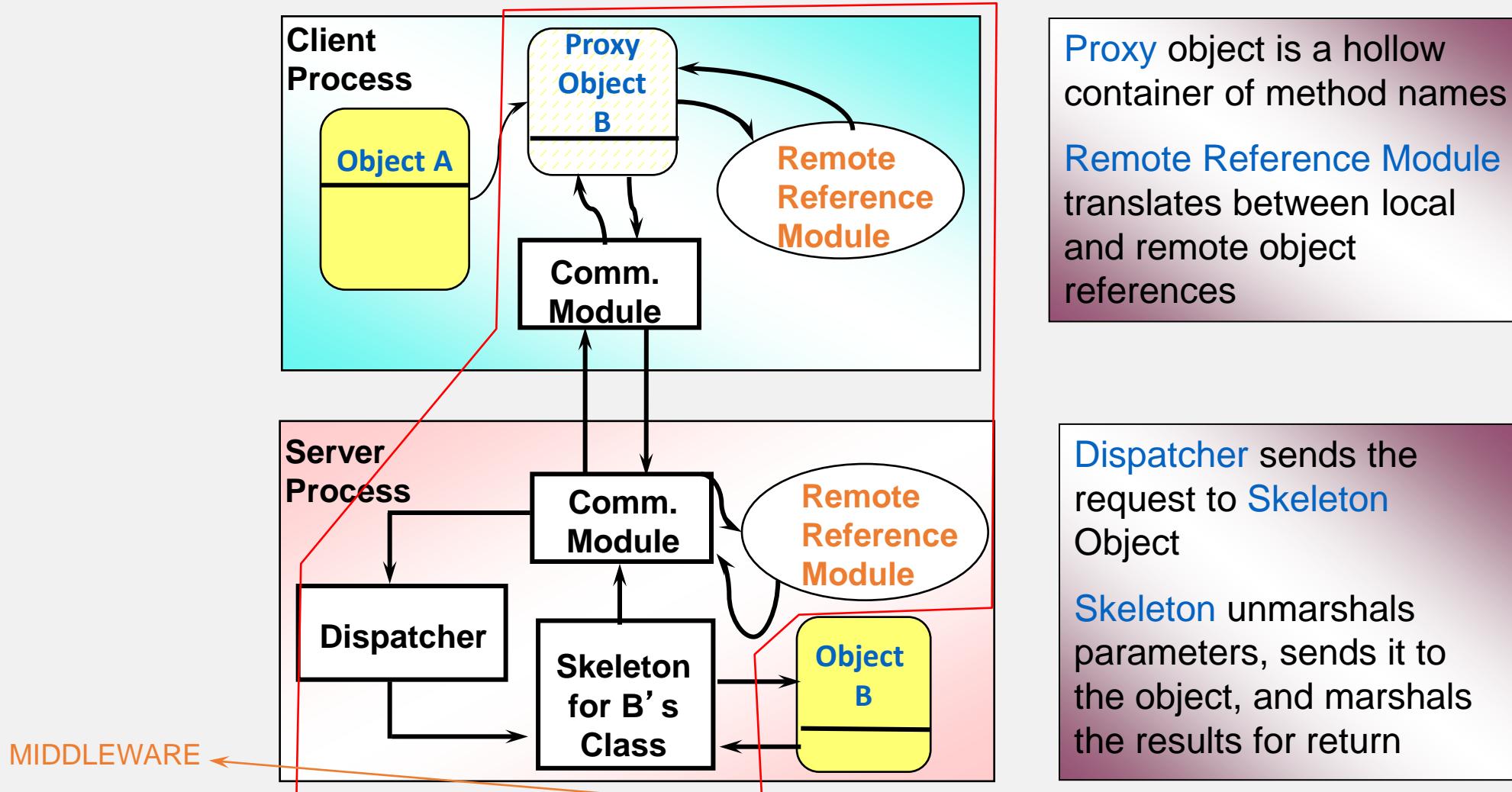


- **Proxy on the “client” (process P1) side**
- **Skeleton and dispatcher on the “server” (process P2) side**

What About Server Side? Dispatcher and Skeleton

- Each process has one dispatcher and a skeleton for each local object (actually, for the class)
- The dispatcher receives all request messages from the communication module
 - For the request message, it uses the method id to select the appropriate method in the appropriate skeleton, passing on the request message
- Skeleton “implements” the methods in the remote interface
 - A skeleton method unmarshals the arguments in the request message and invokes the corresponding method in the remote object (the actual object)
 - It waits for the invocation to complete and marshals the result, together with any exceptions, into a reply message

Summary of Remote Method Invocation (RMI)





CLOUD COMPUTING APPLICATIONS

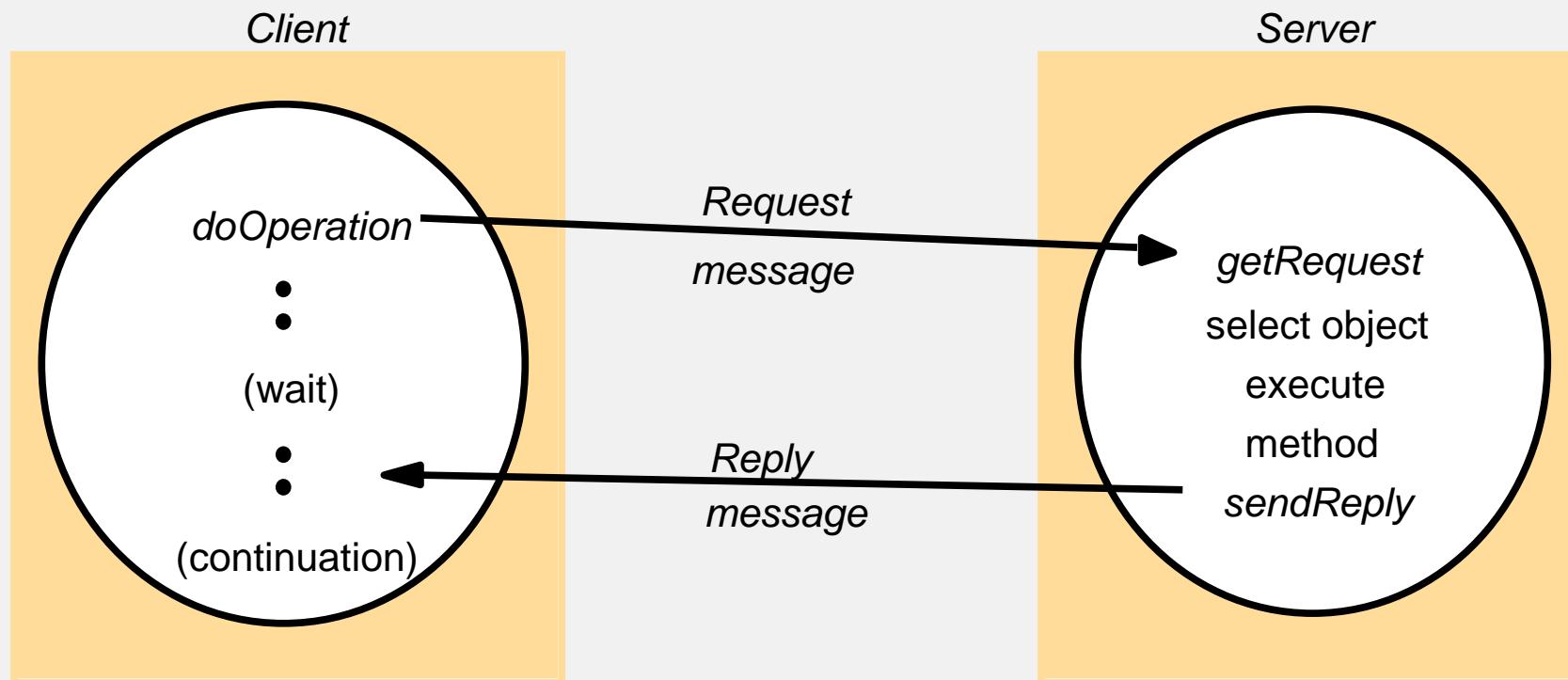
Prof. Roy Campbell

RPC Semantics

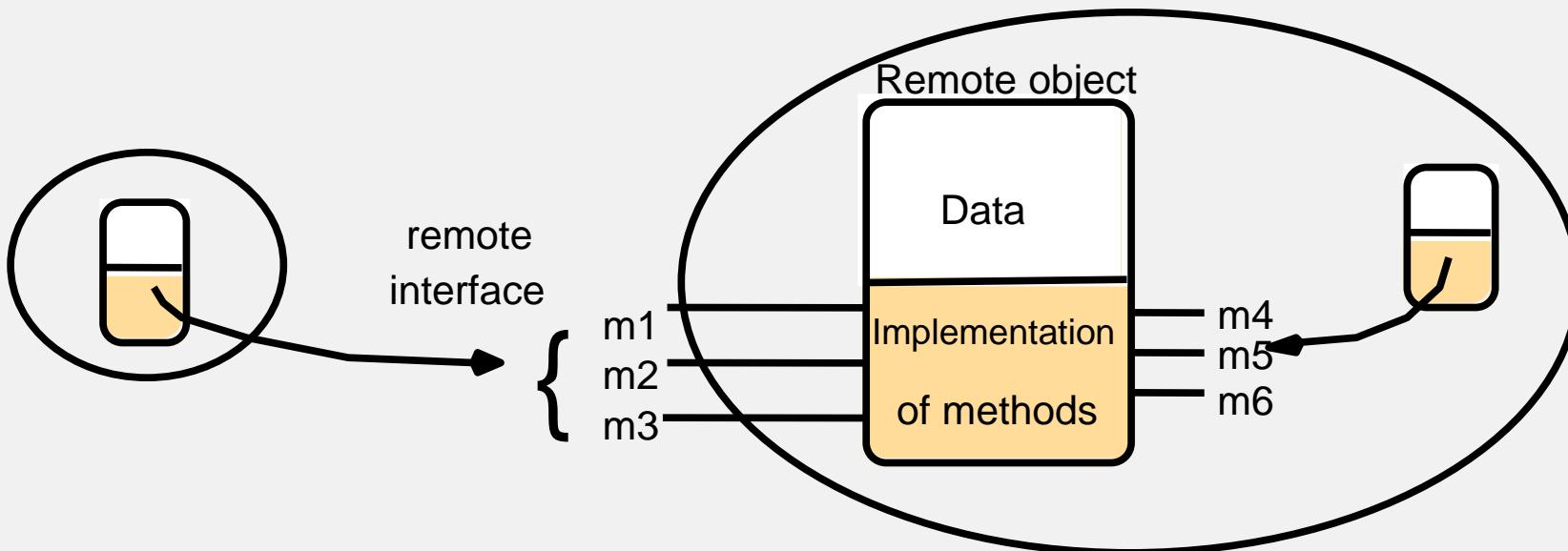
Contents

- Remote Procedure Call Semantics
 - RMI, SOAP

Request-Reply Communication

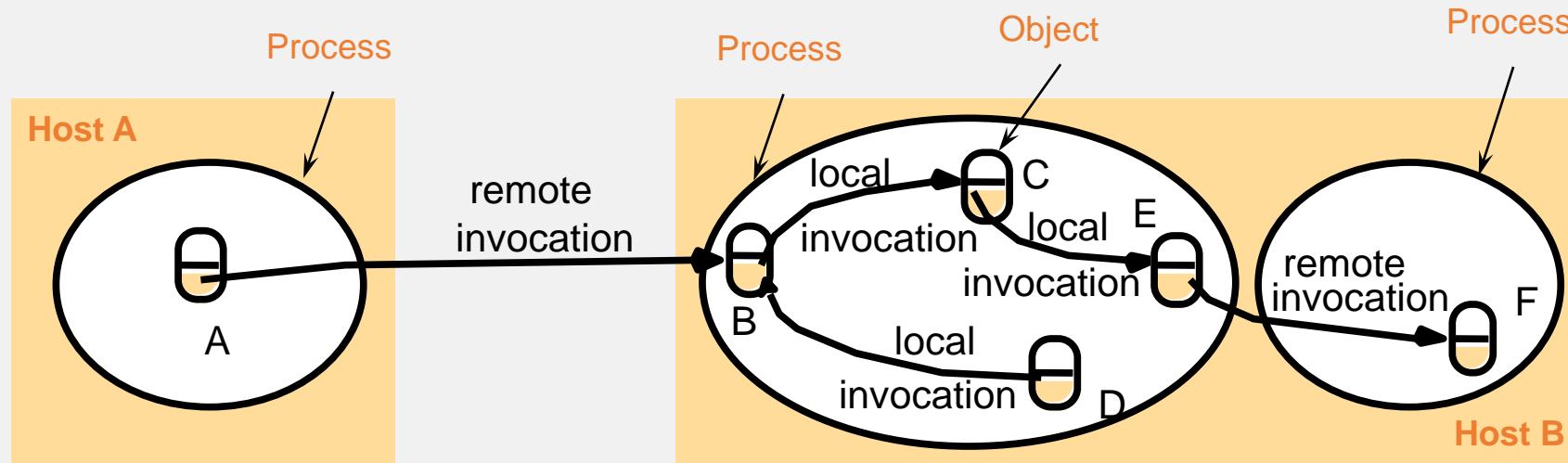


A Remote Object and Its Remote Interface



Example remote object reference = (IP, port, object number, signature, time)

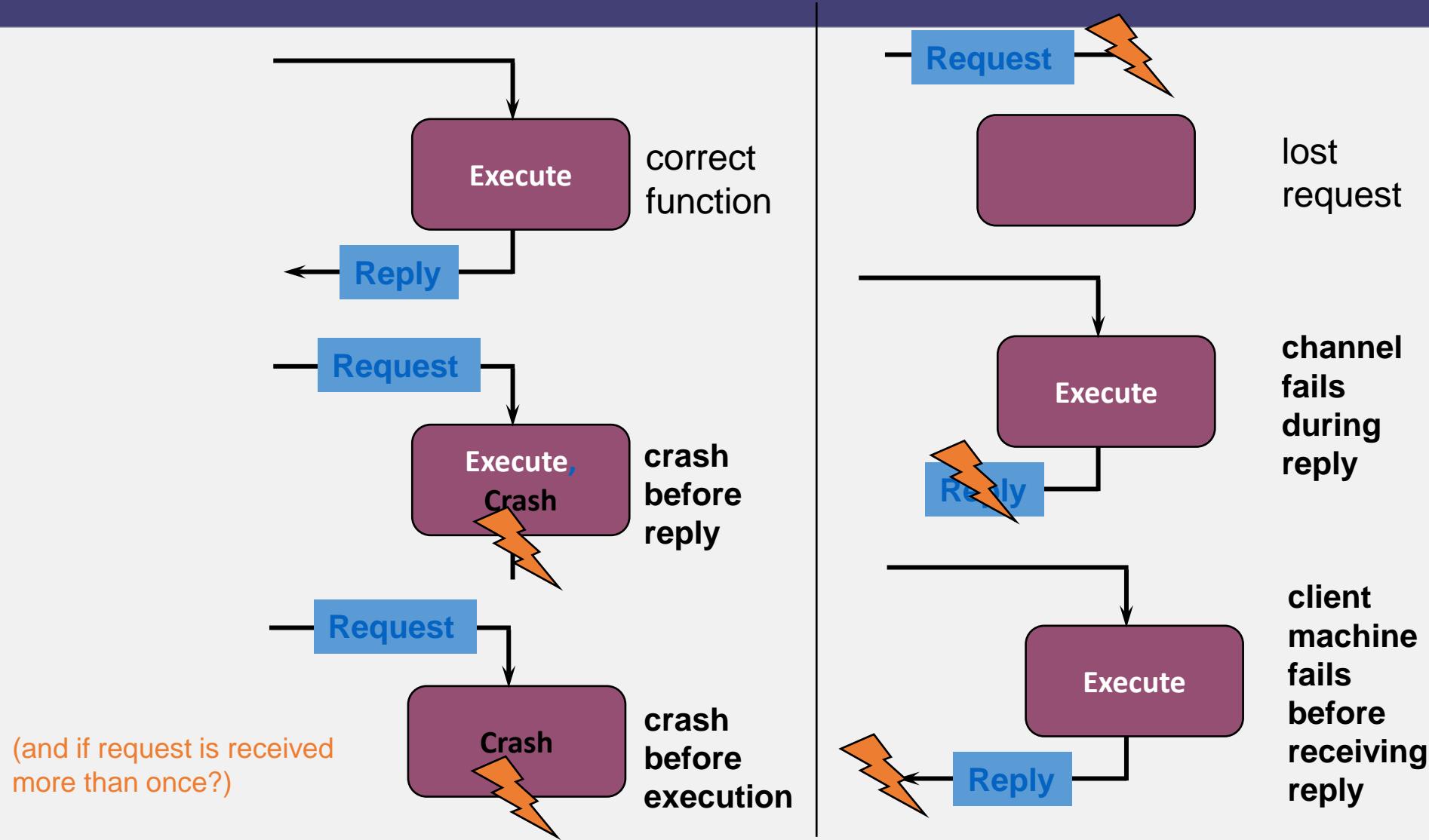
Remote and Local Method Invocations



Local invocation = between objects on same process; has exactly once semantics

Remote invocation = between objects on different processes. Ideally, you'd also want exactly once semantics for remote invocations, but it's difficult in distributed systems (why?)

Failure Modes of RMI RPC



Invocation Semantics

- Middleware aims at providing **transparency**
 - Sometimes obtaining transparency is expensive and unnecessary
 - Because of faults, obtaining “exactly once” semantics is expensive
- Other invocation semantics are possible
 - **Maybe:** Caller cannot determine whether the remote method has been executed
 - **At-least-once:** Caller either receives a result (in which case the user knows the method was executed at least once) or an exception
 - **At-most-once:** Caller either receives a result (in which case the user knows the method was executed at most once) or an exception
- The right invocation semantic depends on the application
 - For example, at-least-once semantic + **idempotent operations** can provide a semantic that would be equivalent to exactly-once applications

Idempotent = same result if applied repeatedly, w/o side effects

Invocation Semantics

Retransmit request message	Duplicate filtering	Re-execute procedure or retransmit reply	Invocation semantic	Used in ...
No	N/A	N/A	Maybe	CORBA
Yes	No	Re-execute procedure	At-least-once	Sun-RPC
Yes	Yes	Retransmit old reply	At-most-once	RMI;CORBA

Retransmit request message: Whether to retransmit the request message until either a reply is received or the server is assumed to be failed

Duplicate filtering: When retransmissions are used, whether to filter out duplicate requests at the server

Re-execute procedure or retransmit reply: Whether to keep a history of result messages to enable lost results to be retransmitted without re-executing the operations



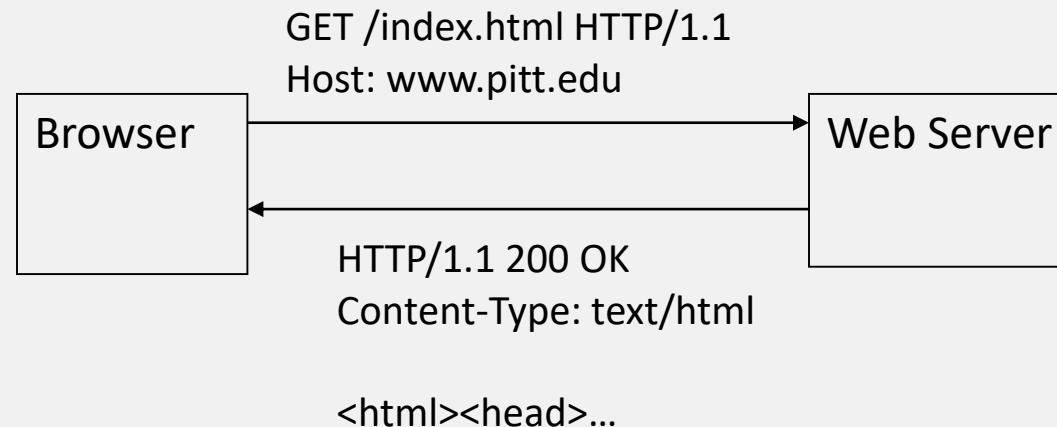
CLOUD COMPUTING APPLICATIONS

Prof. Roy Campbell

HTTP SOAP REST

Hypertext Transfer Protocol (HTTP)

- A communications protocol
- Allows retrieving inter-linked text documents (hypertext)
 - World Wide Web
- HTTP verbs
 - HEAD
 - **GET**
 - **POST**
 - PUT
 - DELETE
 - TRACE
 - OPTIONS
 - CONNECT



SOAP – Simple Object Access Protocol

- Transmitted by HTTP or SMTP (or many others)
- Coded in XML (can be decoded on any machine)
- Return value: any XML document
- Underlies Web Services Description Language (WSDL)

Representational State Transfer (REST)

- A style of software architecture for distributed hypermedia systems such as the World Wide Web
- Introduced in the doctoral dissertation of Roy Fielding
 - One of the principal authors of the HTTP specification
- A collection of network architecture principles that outline how resources are defined and addressed

REST and HTTP

- The motivation for REST was to capture those characteristics of the Web that made the Web successful
 - URI-addressable resources
 - HTTP
 - Make a request – receive response – display response
- Exploits the use of the HTTP beyond HTTP POST and HTTP GET
 - HTTP PUT, HTTP DELETE

REST – Not a Standard

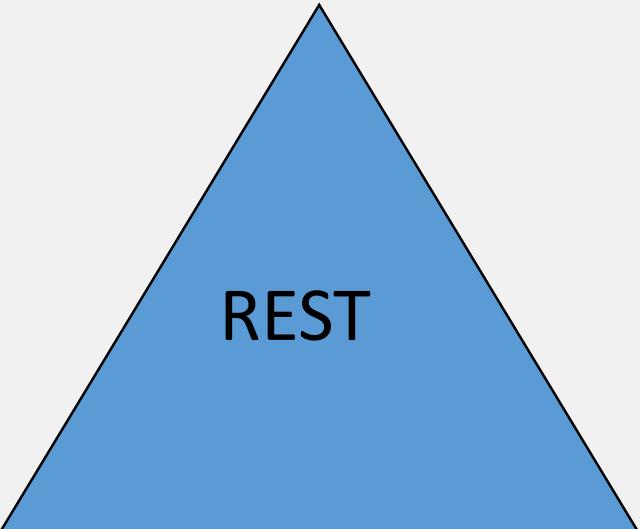
- REST is not a standard
 - JSR 311: JAX-RS: The Java™ API for RESTful Web Services
- But it uses several standards:
 - HTTP
 - URL
 - XML/HTML/GIF/JPEG/etc. (resource representations)
 - Text/xml, text/html, image/gif, image/jpeg, etc. (resource types, MIME types)

Main Concepts

Nouns (resources)

unconstrained

i.e., <http://example.com/employees/12345>



REST

Verbs

constrained

i.e., GET

Representations

constrained

i.e., XML

Resources

- The key abstraction of information in REST is a resource
- A resource is a conceptual mapping to a set of entities
 - Any information that can be named can be a resource: a document or image, a temporal service (e.g., "today's weather in Los Angeles"), a collection of other resources, a non-virtual object (e.g., a person), and so on
- Represented with a global identifier (URI in HTTP)
 - <http://www.boeing.com/aircraft/747>

Naming Resources

- REST uses URI to identify resources
 - <http://localhost/books/>
 - <http://localhost/books/ISBN-0011>
 - <http://localhost/books/ISBN-0011/authors>
 - <http://localhost/classes>
 - <http://localhost/classes/cs2650>
 - <http://localhost/classes/cs2650/students>
- As you traverse the path from more generic to more specific, you are navigating the data

Verbs

- Represent the actions to be performed on resources
- HTTP GET
- HTTP POST
- HTTP PUT
- HTTP DELETE

HTTP GET

- How clients ask for the information they seek
- Issuing a GET request transfers the data from the server to the client in some representation
 - GET <http://localhost/books>
 - Retrieve all books
 - GET <http://localhost/books/ISBN-0011021>
 - Retrieve book identified with ISBN-0011021
 - GET <http://localhost/books/ISBN-0011021/authors>
 - Retrieve authors for book identified with ISBN-0011021

HTTP POST, HTTP PUT

- HTTP POST creates a resource
- HTTP PUT updates a resource
- POST <http://localhost/books/>
 - Content: {title, authors[], ...}
 - Creates a new book with given properties
- PUT <http://localhost/books/isbn-111>
 - Content: {isbn, title, authors[], ...}
 - Updates book identified by isbn-111 with submitted properties

HTTP DELETE

- Removes the resource identified by the URI
- DELETE <http://localhost/books/ISBN-0011>
 - Delete book identified by ISBN-0011

Representations

- How data is represented or returned to the client for presentation
- Two main formats:
 - JavaScript Object Notation (JSON)
 - XML
- It is common to have multiple representations of the same data

Representations

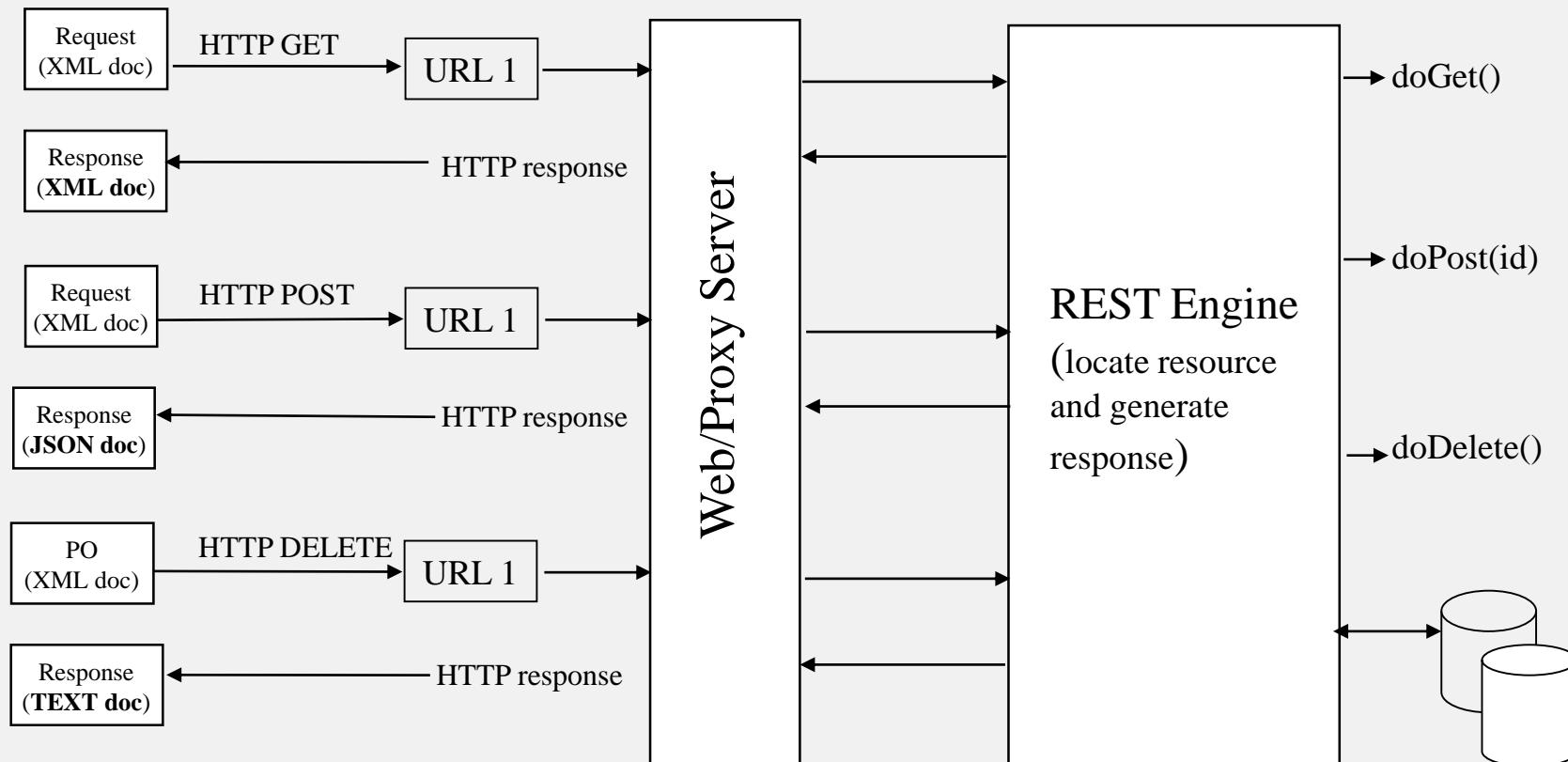
- XML

```
<COURSE>
  <ID>CS2650</ID>
  <NAME>Distributed Multimedia Software</NAME>
</COURSE>
```

- JSON

```
{
  "course":{
    "id":"CS2650",
    "name":"Distributed Multimedia Software"
  }
}
```

Architecture Style



Real Life Examples

- Google Maps
- Google AJAX Search API
- Yahoo Search API
- Amazon WebServices



CLOUD COMPUTING APPLICATIONS

JSON

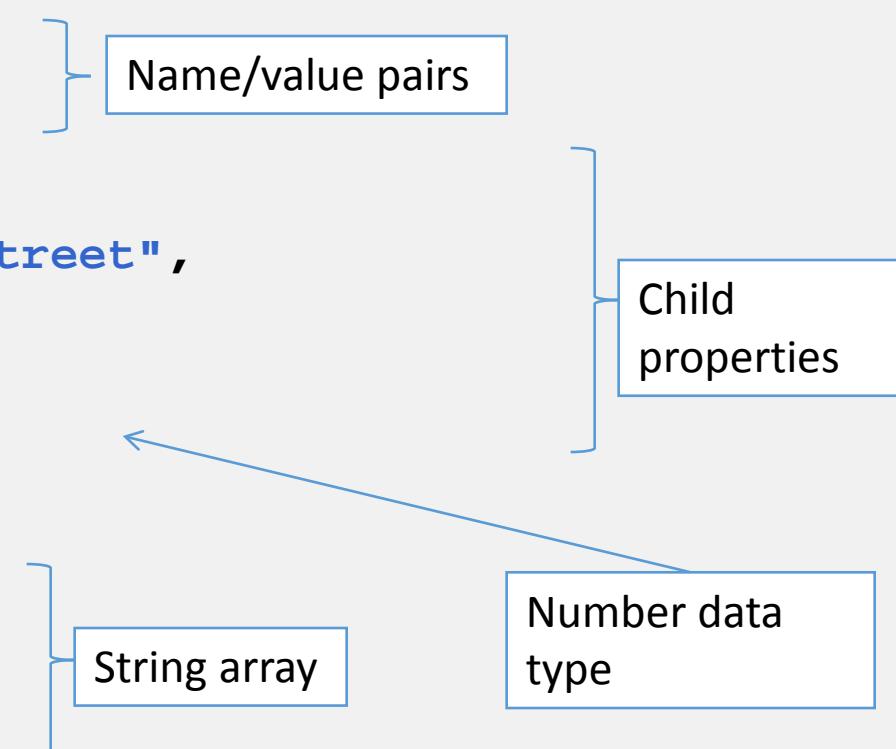
Prof. Roy Campbell

JSON – What Is It?

- “*JSON (JavaScript Object Notation) is a **lightweight data interchange format**. It is easy for humans to read and write. It is easy for machines to parse and generate.*”
– JSON.org
- Importantly: JSON is a subset of JavaScript

JSON – What Does It Look Like?

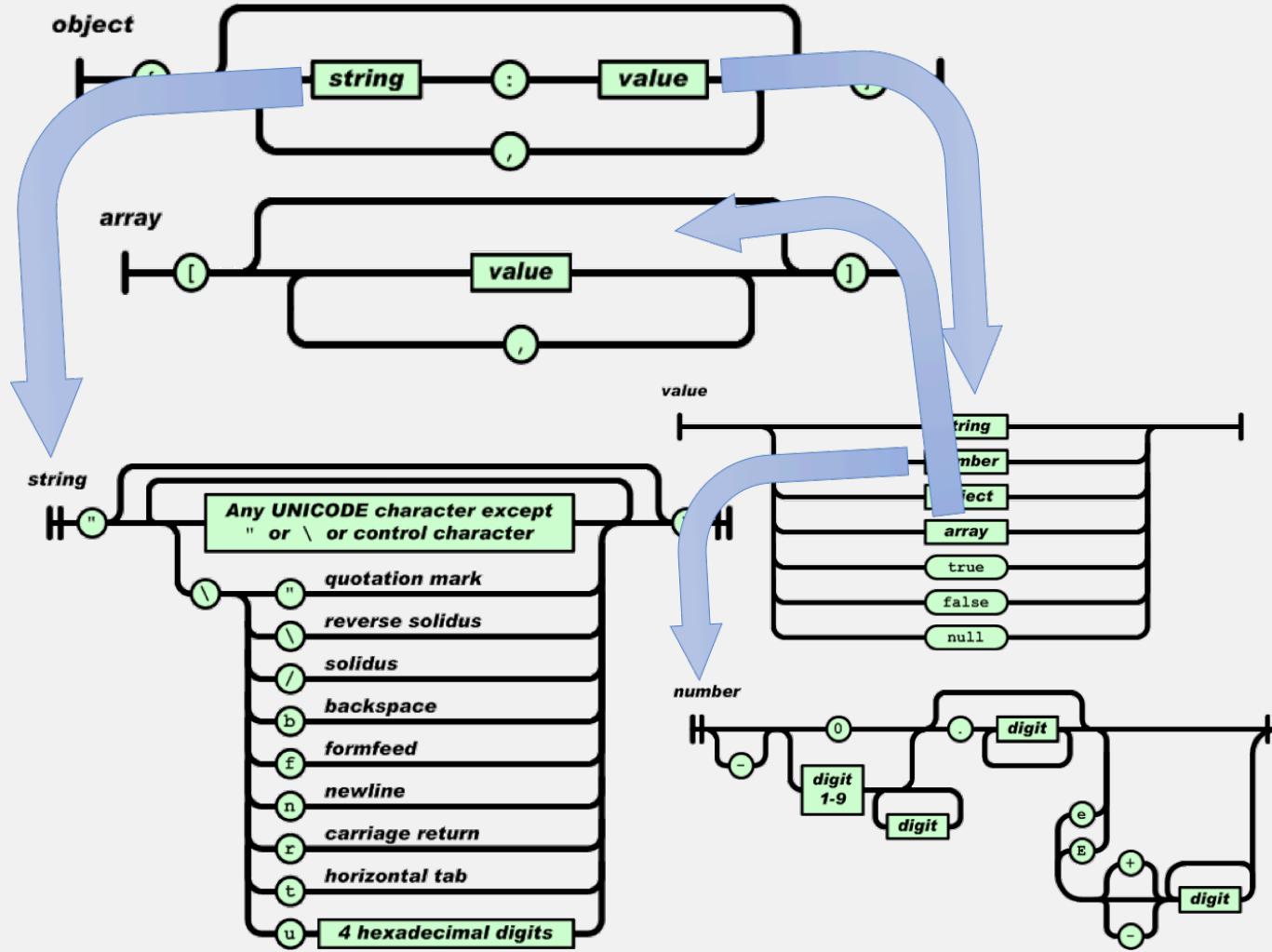
```
{  
    "firstName": "John",  
    "lastName": "Smith",  
    "address": {  
        "streetAddress": "21 2nd Street",  
        "city": "New York",  
        "state": "NY",  
        "postalCode": 10021  
    },  
    "phoneNumbers": [  
        "212 555-1234",  
        "646 555-4567"  
    ]  
}
```



The diagram illustrates the structure of the provided JSON object with the following annotations:

- A blue bracket labeled "Name/value pairs" groups the top-level properties: "firstName", "lastName", "address", "phoneNumbers", and the final closing brace.
- A blue bracket labeled "Child properties" groups the properties within the "address" object: "streetAddress", "city", "state", and "postalCode".
- A blue bracket labeled "String array" groups the elements within the "phoneNumbers" array: "212 555-1234" and "646 555-4567".
- A blue bracket labeled "Number data type" groups the value of the "postalCode" property, which is the number 10021.

JSON Data Structures



JSON vs. XML

JSON	XML
Data Structure	Data Structure
No validation system	XSD
No namespaces	Has namespaces (can use multiples)
Parsing is just an eval •Fast •Security issues	Parsing requires XML document parsing using things like XPath
In JavaScript you can work with objects – runtime evaluation of types	In JavaScript you can work with strings – may require additional parsing
Security: Eval() means that if the source is not trusted anything could be put into it. Libraries exist to make parsing safe(r)	Security: XML is text/parsing – not code execution



CLOUD COMPUTING APPLICATIONS

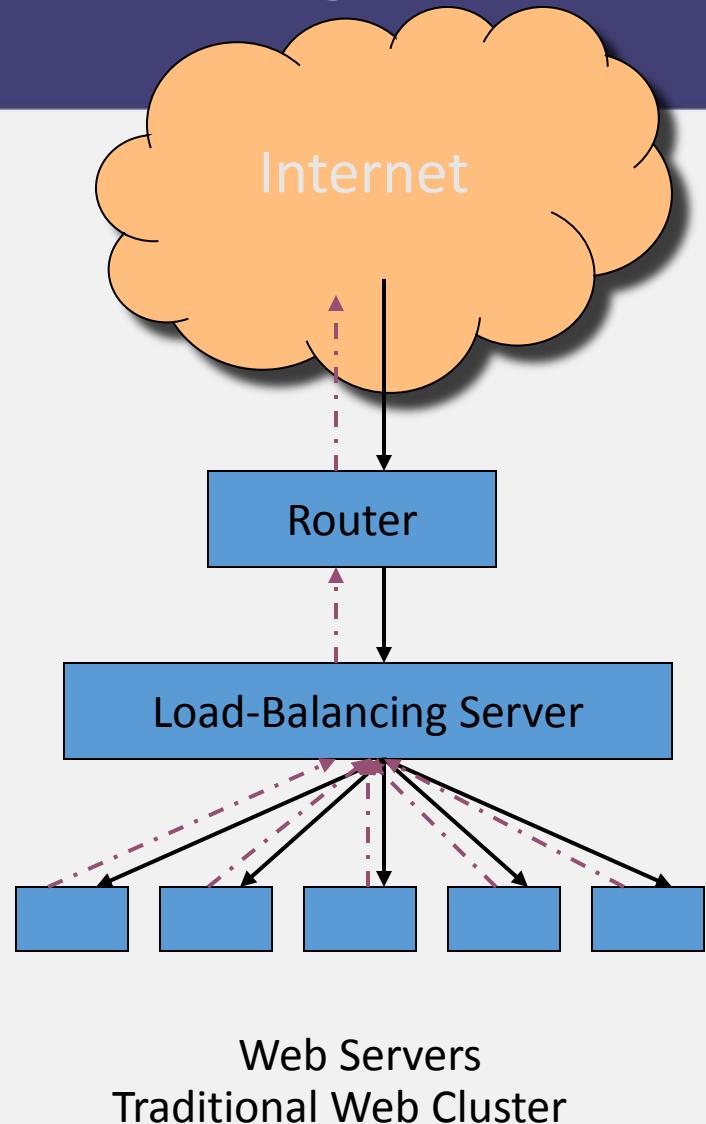
LOAD BALANCER INTRO

Prof. Roy Campbell

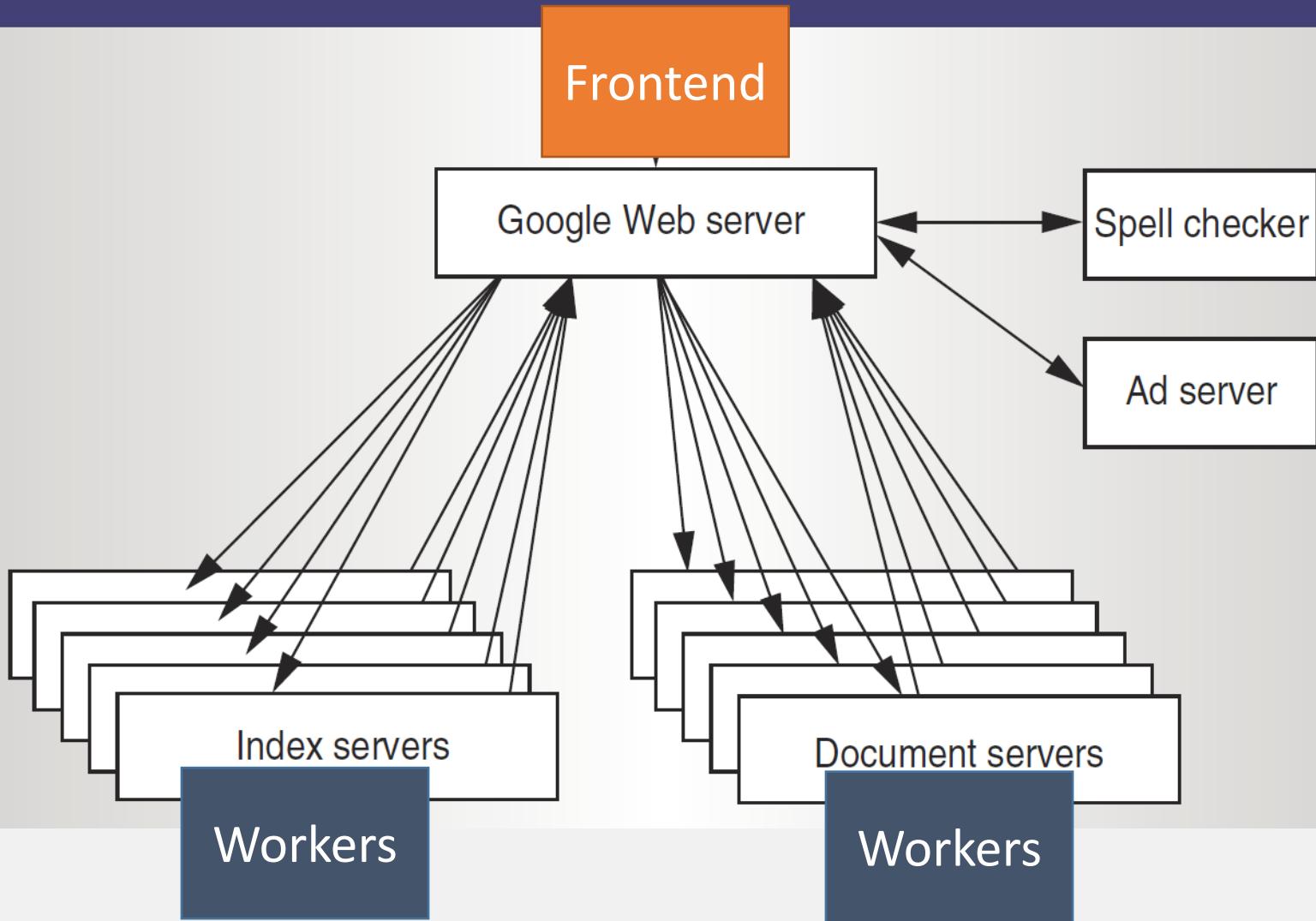
Introduction to Load Balancing

- Request enters a router
- Load balancing server determines which web server should serve the request
- Sends the request to the appropriate web server

→ Request
- - - → Response

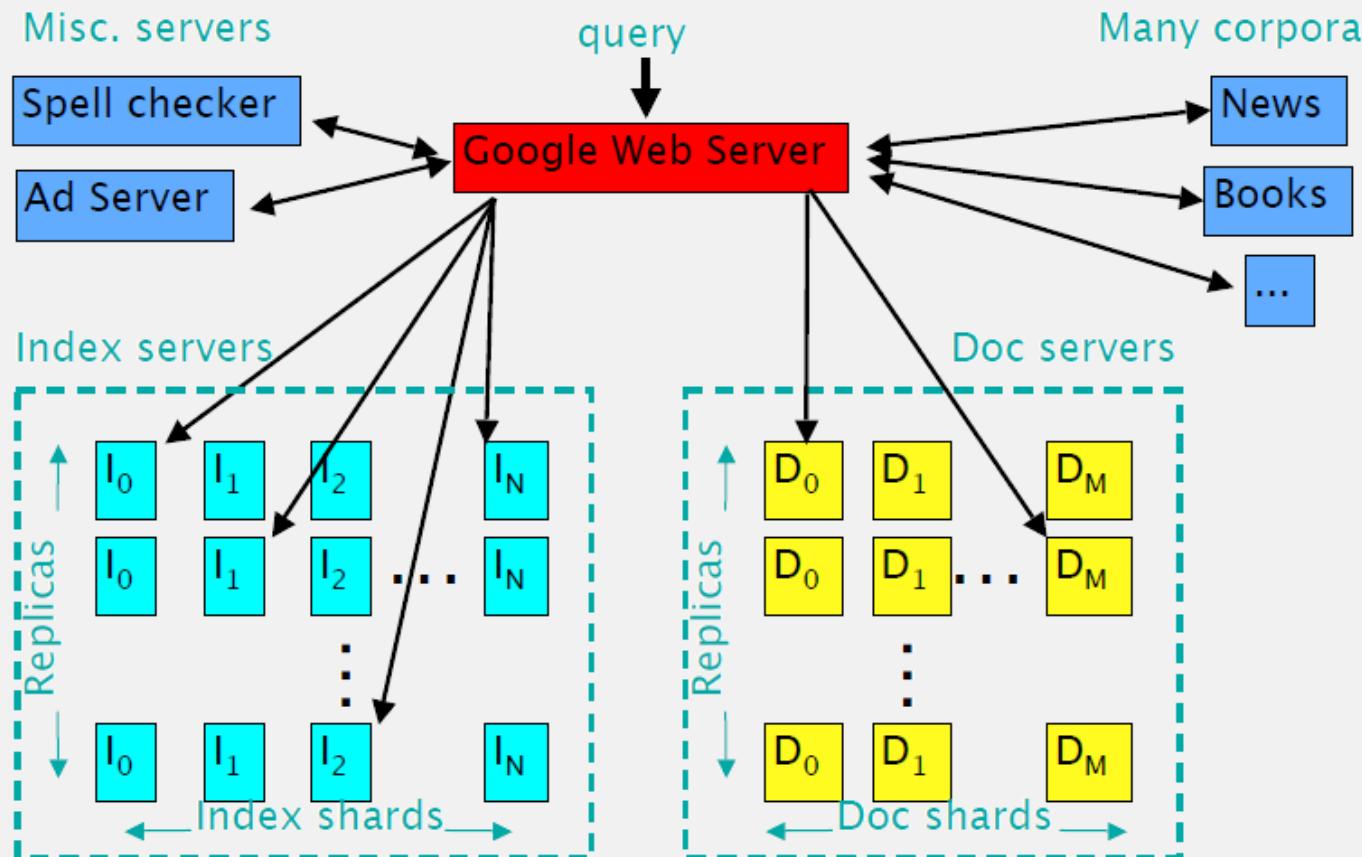


Web Search for a planet: The Google Cluster Architecture (2003)



Google: A Behind-the-Scenes Tour

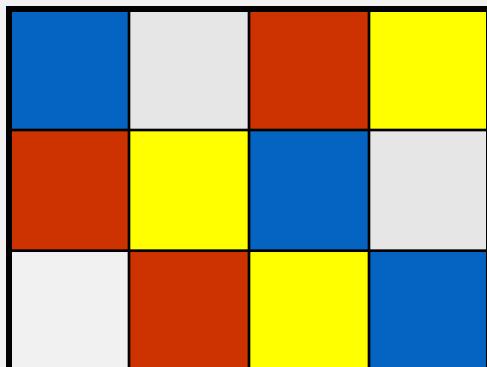
Google Query Serving Infrastructure



Elapsed time: 0.25s, machines involved: 1000s+

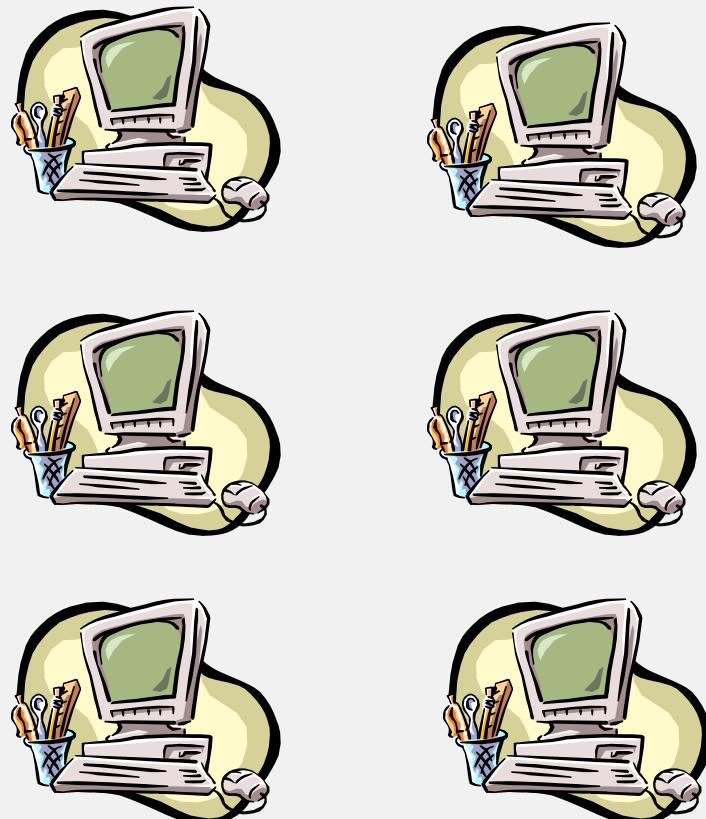
How do we split up information?

Content



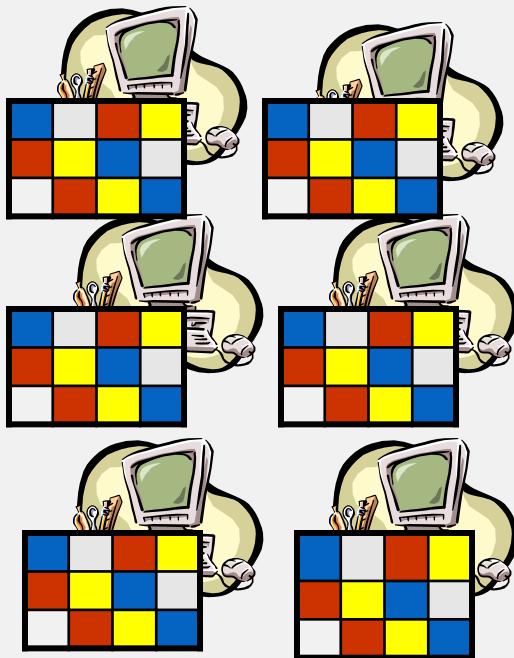
?

Server Farm

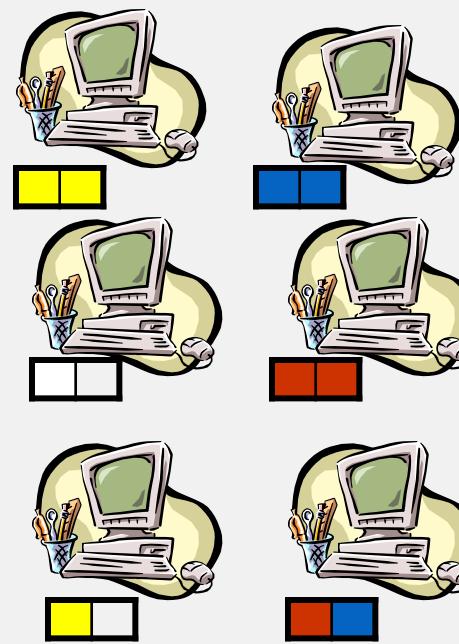


Information Strategies

Replication



Partition



Load Balancing Approaches

File Distribution	Routing
Content/Locality Aware	DNS Server
Size Aware	Centralized Router
Workload Aware	Distributed Dispatcher

Issues

- Efficiently processing requests with optimizations for load balancing
 - Send and process requests to a web server that has files in cache
 - Send and process requests to a web server with the least amount of requests
 - Send and process requests to a web server determined by the size of the request



CLOUD COMPUTING APPLICATIONS

LOAD BALANCER SCHEMES

Prof. Roy Campbell

What Does a Server Load Balancer (SLB) Do?

- Gets user to needed resource
 - Server must be available
 - User's "session" must not be broken
 - If user must get to the same resource over and over, the SLB device must ensure that happens (i.e., session persistence)
- In order to do work, SLB must
 - Know servers – IP / port, availability
 - Understand details of some protocols (e.g., FTP, SIP)
- Network Address Translation (NAT)
 - Packets are rewritten as they pass through the SLB device

Reasons to Load-Balance

- Scale applications / services
- Ease of administration / maintenance
 - Easily and transparently remove physical servers from rotation in order to perform any type of maintenance on that server
- Resource sharing
 - Can run multiple instances of an application / service on a server; could be running on a different port for each instance; can load-balance to different port based on data analyzed

Load-Balancing Algorithms

- Most predominant
 - **Least connections:** Server with fewest number of flows gets the new flow request
 - **Weighted least connections:** Associate a weight / strength for each server and distribute load across server farm based on the weights of all servers in the farm
 - **Round robin:** Round robin through the servers in server farm
 - **Weighted round robin:** Give each server “weight” number of flows in a row; weight is set just like it is in weighted least flows
- There are other algorithms that look at or try to predict server load in determining the load of the real server

How SLB Devices Make Decisions

- The SLB device can make its load-balancing decisions based on several factors
 - Some of these factors can be obtained from the packet headers (i.e., IP address, port numbers)
 - Other factors are obtained by looking at the data beyond the network headers. Examples:
 - HTTP cookies
 - HTTP URLs
 - SSL client certificates
- The decisions can be based strictly on flow counts, or they can be based on knowledge of application
- For some protocols, like FTP, you must have knowledge of protocol to correctly load-balance (i.e., control and data connection must go to same physical server)

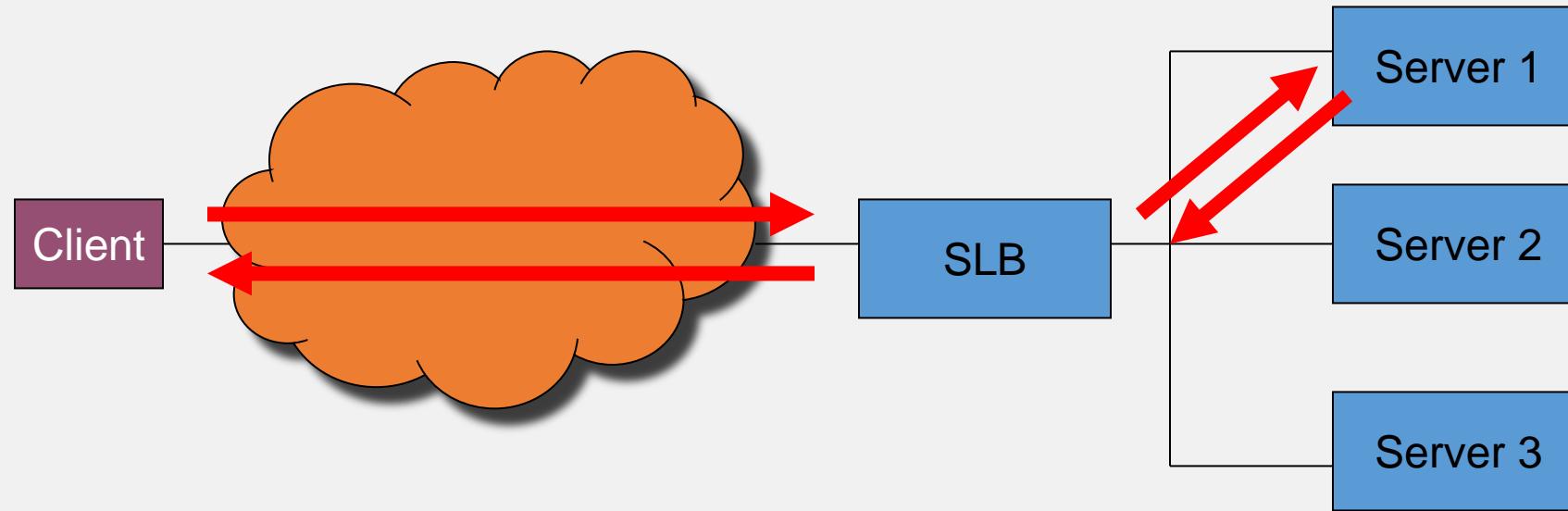
When a New Flow Arrives

- Determine whether virtual server exists
 - If so, make sure virtual server has available resources
 - If so, then determine level of service needed by that client to that virtual server
 - If virtual machine is configured with particular type of protocol support of session persistence, then do that work
 - Pick a real server for that client
 - The determination of real server is based on flow counts and information about the flow
 - In order to do this, the SLB may need to proxy the flow to get all necessary information for determining the real server; this will be based on the services configured for that virtual server
- If not, the packet is bridged to the correct interface based on Layer 2

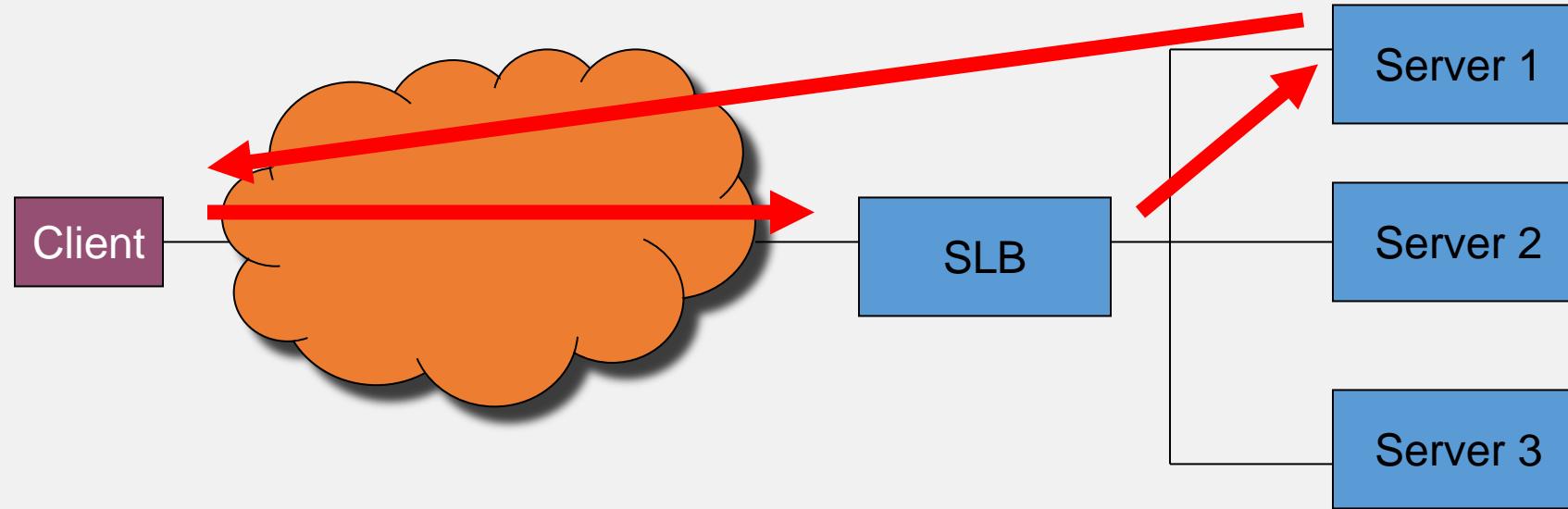
SLB: Architectures

- Traditional
 - SLB device sits between the Clients and the Servers being load-balanced
- Distributed
 - SLB device sits off to the side and only receives the packets it needs to, based on flow setup and teardown

SLB: Traditional View with NAT



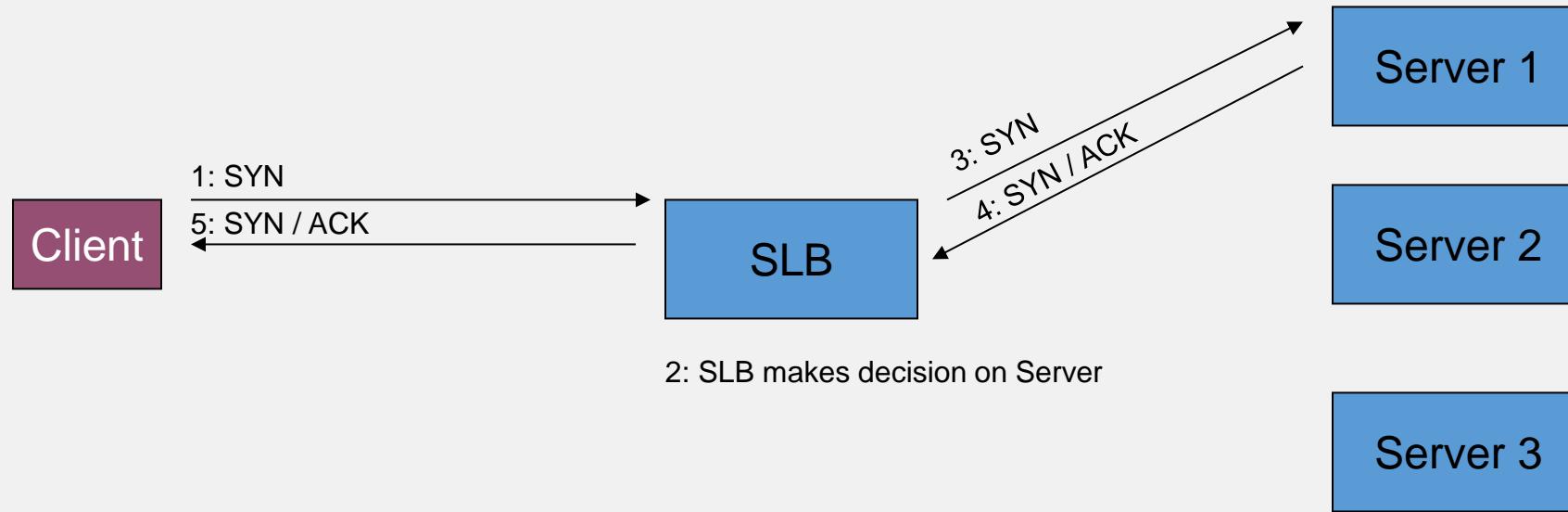
SLB: Traditional View without NAT



Load-Balance: Layer 3 / 4

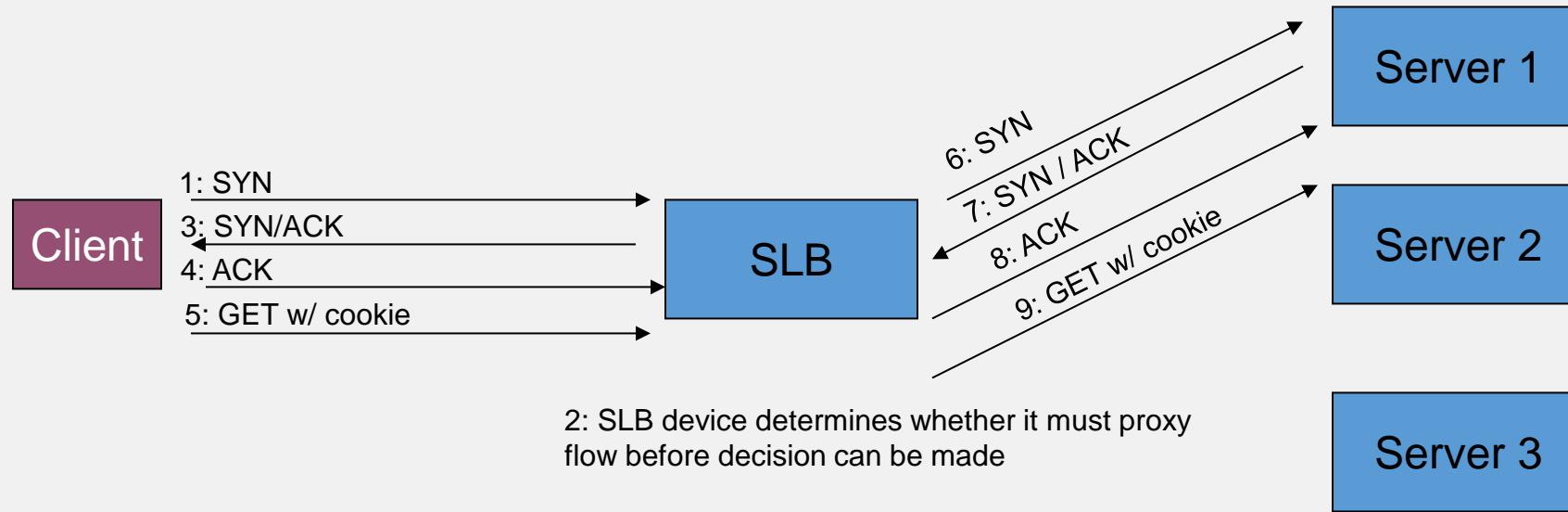
- Look at the destination IP address and port to make a load-balancing decision
- In order to do that, you can determine a real server based on the first packet that arrives

Layer 3 / 4: Sample Flow



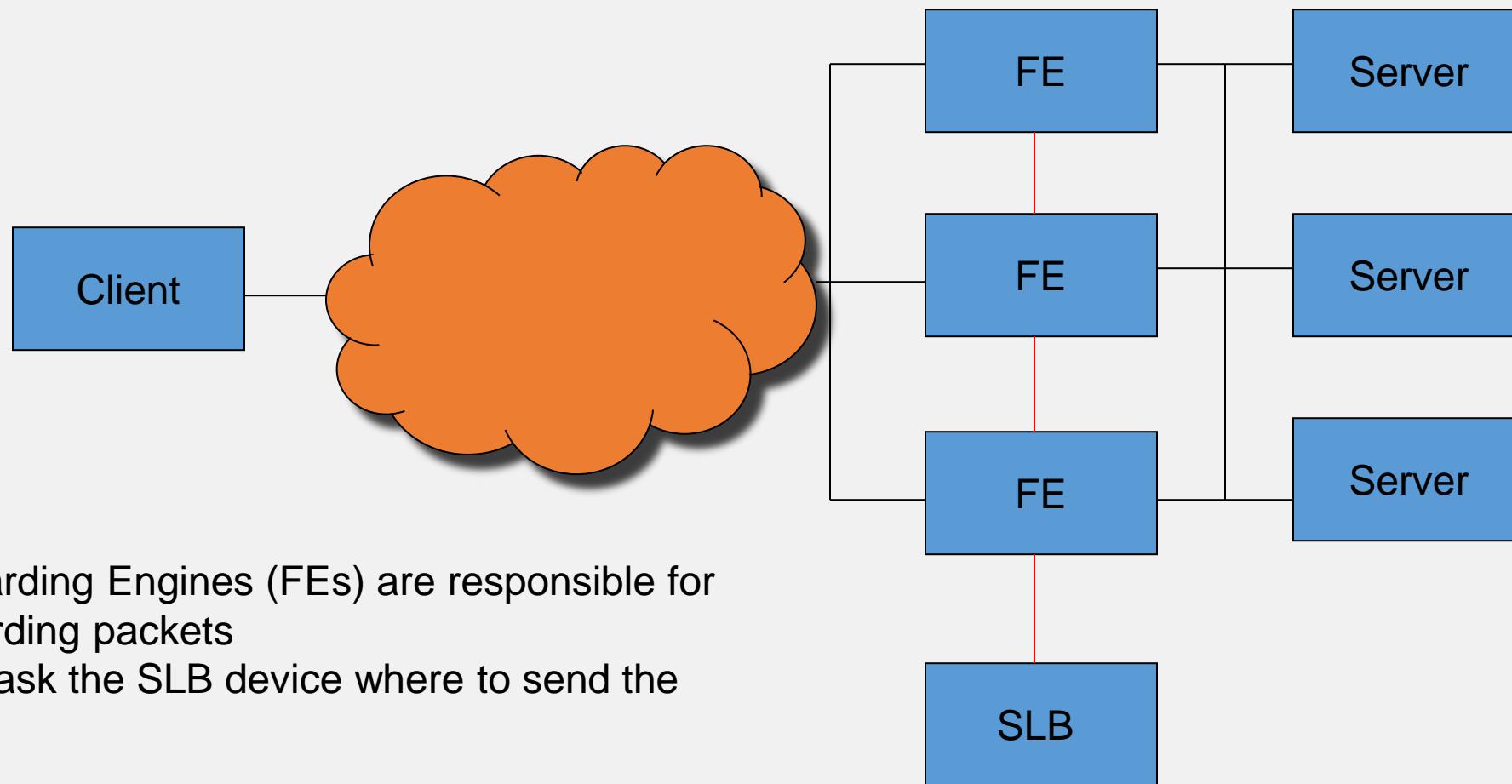
Rest of flow continues through HTTP GET and Server response

Layer 5+: Sample Flow

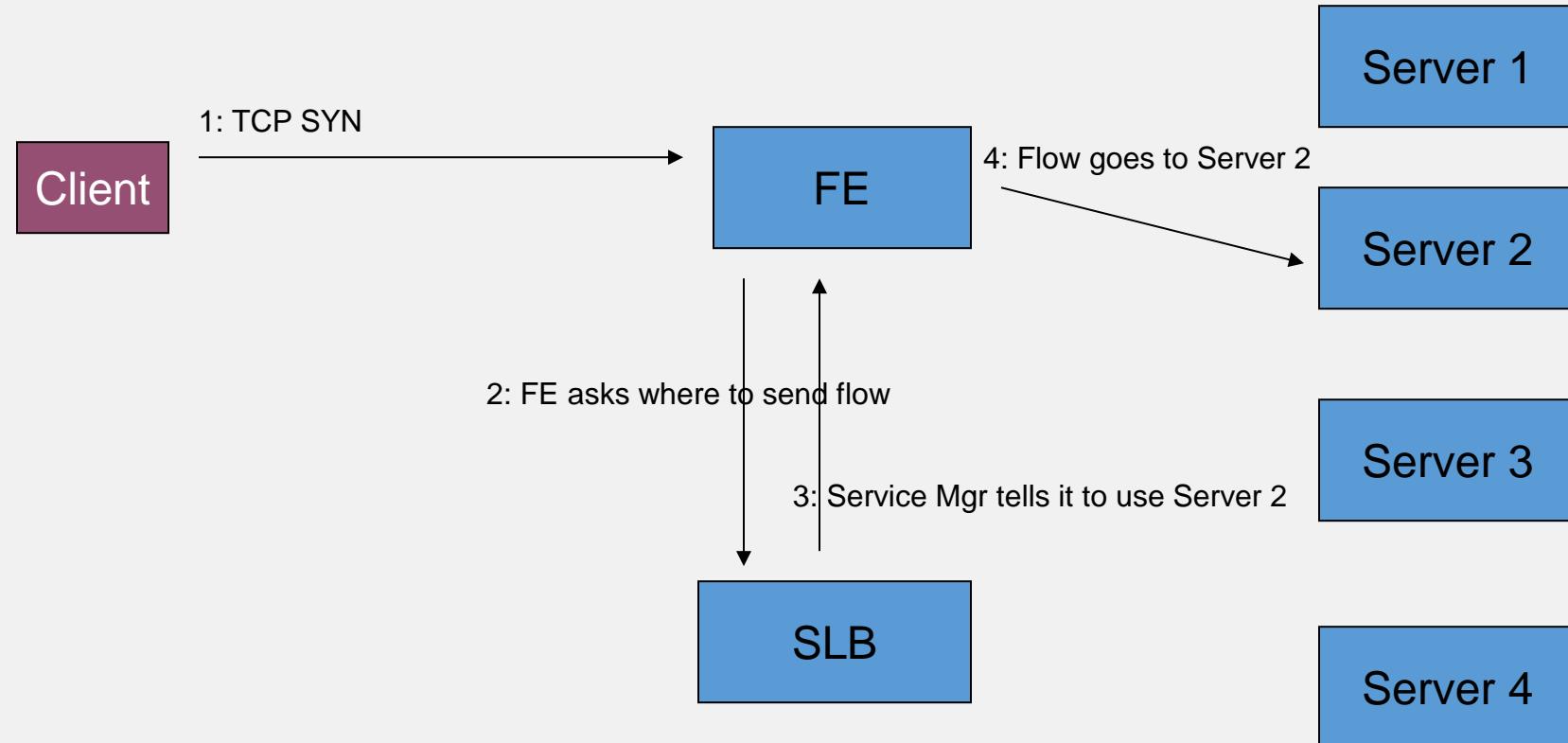


- Rest of flow continues with Server response
- Note that the flow can be unproxied at this point for efficiency

SLB: Distributed Architecture



Distributed Architecture: Sample Flow



- Subsequent packets flow directly from Client to Server 2 through the FE
- The FE must notify the SLB device when the flow ends



CLOUD COMPUTING APPLICATIONS

Prof. Roy Campbell

Protocol Buffers and Thrift

Protocol Buffers

- Invented by Google
- Build on RPCs
- Language-neutral, platform-neutral
- Extensible mechanism
- Serializing structured data
- For distributed services

Example Schema

```
1. Message Person{  
2.     required int32 id = 1;  
3.     required string name = 2;  
4.     optional string email = 3;  
5. }
```

Example Code (Ruby for example)

```
1. #!/usr/bin/env ruby
2. # Generated by the protocol buffer compiler. DO NOT EDIT!
3. require "protocol_buffers"

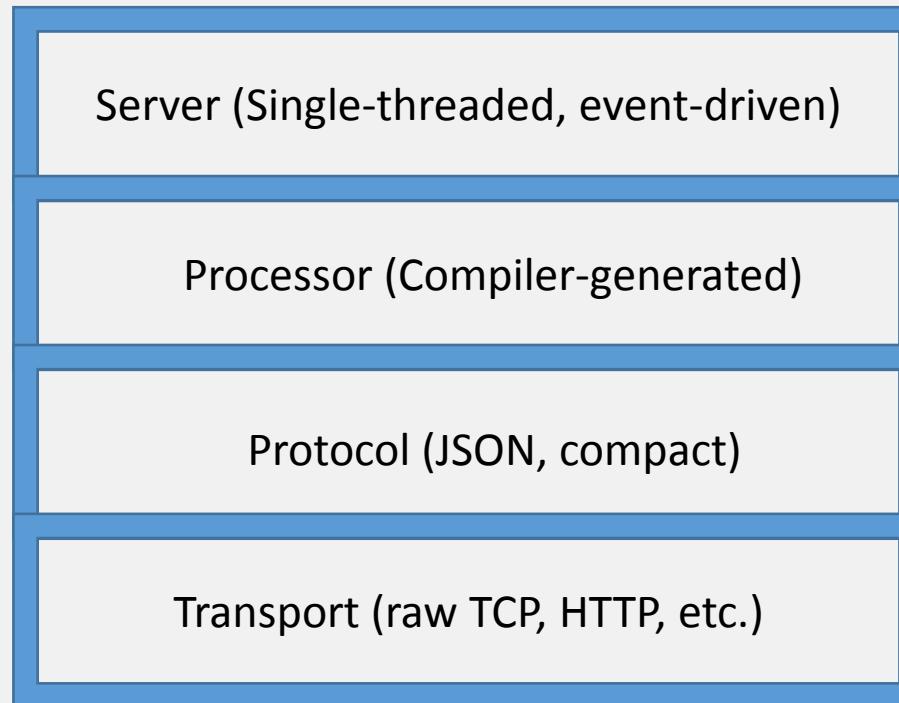
4. # forward declarations
5. class Person < ::ProtocolBuffers::Message; end

6. class Person < ::ProtocolBuffers::Message set_fully_qualified_name "Person"

7. required :int32, :id, 1
8. required :string, :name, 2
9. optional :string, :email, 3
10. end
```

Thrift Network Stack

Interface Definition Language. Creates files for clients and servers from needs to serialize structured data (Facebook)



Transport Methods

- Transport
 - open
 - close
 - read
 - write
 - flush
- Server transport also has open, listen, accept, and close allowing
 - **Read / write to / from a file on disk**
 - **http**

Example File Transports

- `TFileTransport` – This transport writes to a file
- `TFramedTransport` – Transport for non-blocking server
- `TMemoryTransport` – Uses memory for I/O
- `TSocket` – Uses blocking socket I/O for transport
- `TZlibTransport` – Performs compression using [zlib](#)

Example server codes

- `TNonblockingServer` – A multi-threaded server using [non-blocking I/O](#)
- `TSimpleServer` – A single-threaded server using standard blocking I/O. Useful for testing.
- `TThreadPoolServer` – A multi-threaded server using standard blocking I/O.

Example Schema

```
1. struct Person{  
2.     int32 id = 0,  
3.     string name,  
4.     optional string email  
5. }
```



CLOUD COMPUTING APPLICATIONS

Serverless Cloud Computing:
Landscape

Prof. Reza Farivar

Serverless Cloud Computing Landscape

- Different Categories of Serverless Computing
 - Compute
 - Platform as a Service (Apps)
 - Function as a Service (Functions)
 - Container as a Service (Containers)
 - Storage
 - Blobs (Binary Large Objects)
 - Key/Value Datastores
 - Analytics
 - AI and ML

Compute: PaaS

- The unit of compute is a full App
- You still select which server / platform you need, only that you don't need to manage it anymore
- Amazon Elastic BeanStalk
- Google AppEngine
- Microsoft Azure App Service
- IBM CloudFoundry
 - Based on Open Source CloudFoundry
 - Originally developed by VMware, transferred to Pivotal Software (a joint venture by EMC, VMware and General Electric), brought back into VMware at the end of 2019
- Oracle Java Cloud Service

Function as a Service

- This is what most people think of as “Serverless”
- Unit of compute is a function
 - Functions running when “events” are triggered
- Amazon AWS Lambda
- Microsoft Azure Functions
- Google Cloud Functions
- IBM Cloud Functions
 - Based on Apache Open Whisk
- Oracle Functions
 - Apache Fn
- Open Source Open Lambda

Container as a Service

- Function as a Service on HEAVY steroids!
 - Containers have to be Stateless
 - Unit of compute is a whole container
 - Container running when “events” are triggered
- Amazon Elastic Container Service (ECS), Elastic Kubernetes Service (EKS), Fargate
- Microsoft Azure Kubernetes Service (AKS)
 - Built on top of Open Source KEDA
- Google Cloud Run, Anthos
 - Built on top of Open Source Knative
 - Kubernetes-based platform to deploy and manage modern serverless workloads.
- IBM Cloud Kubernetes Service
- Oracle Container Engine for Kubernetes
- Open Source Kubernetes
 - Google Borg

Serverless Storage: Blobs

- Blob Storage
- Amazon Simple Storage Service (S3)
- Microsoft Azure Blob Storage
- IBM Object Storage
- Google Cloud Storage
- Oracle Object Storage

Serverless Storage: Key/Value Database

- Distributed NOSQL key/value storage service
- Amazon DynamoDB
- Microsoft Azure Cosmos DB
- Google Cloud Datastore, Firestore, Cloud BigTable
- IBM Cloudant
- Open Source: Cassandra



CLOUD COMPUTING APPLICATIONS

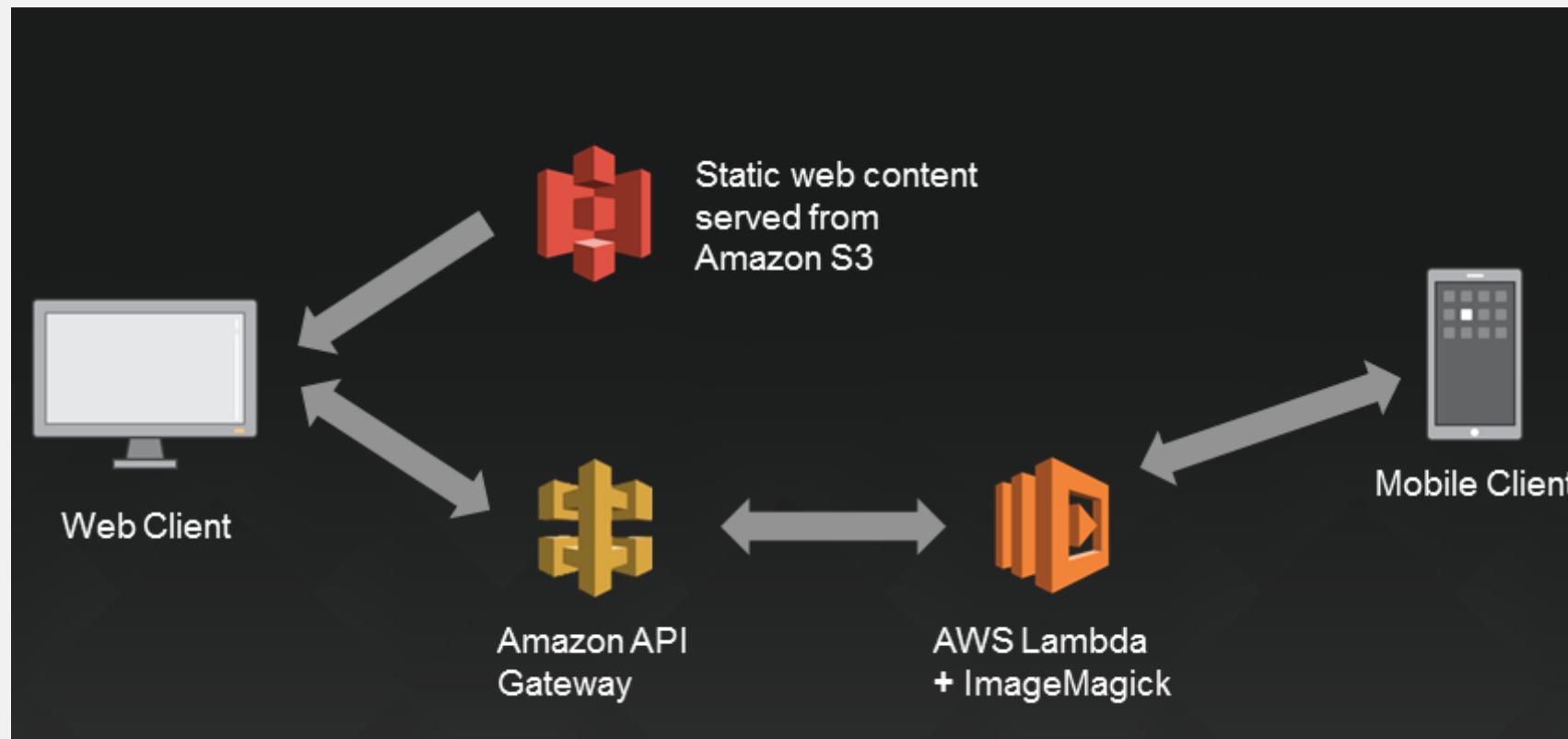
Serverless Architecture

Roy Campbell & Reza Farivar

Introduction to Serverless Architecture

- “Applications where some amount of server-side logic is still written by the application developer but unlike traditional architectures is run in stateless compute containers that are event-triggered, ephemeral (may only last for one invocation), and fully managed by a 3rd party”
- ‘Functions as a service / FaaS’
- AWS Lambda is one of the most popular implementations of FaaS at present, but there are others

Introduction to Serverless Architecture



Desktop Platform

Microsoft

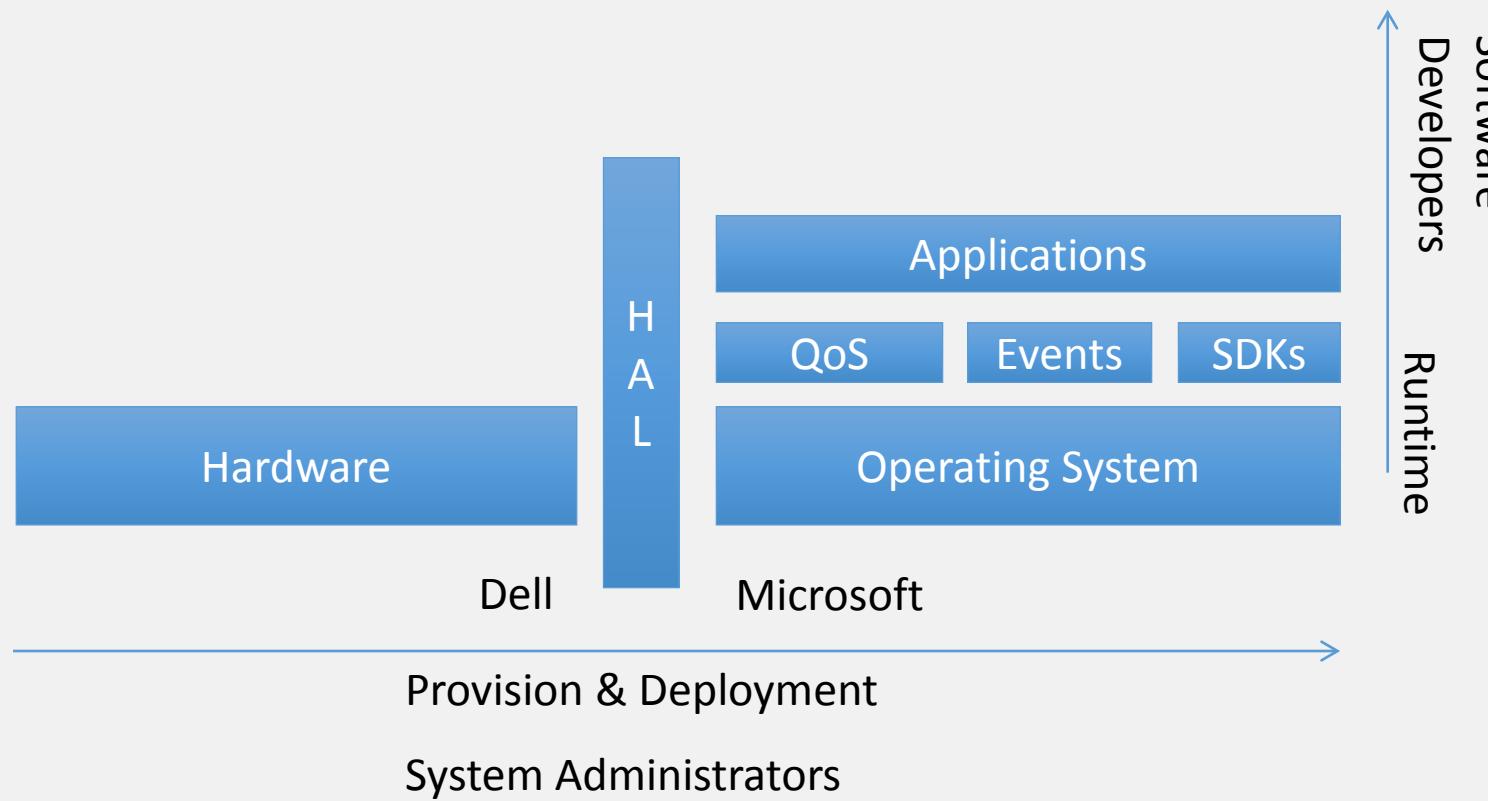
Operating System

HAL

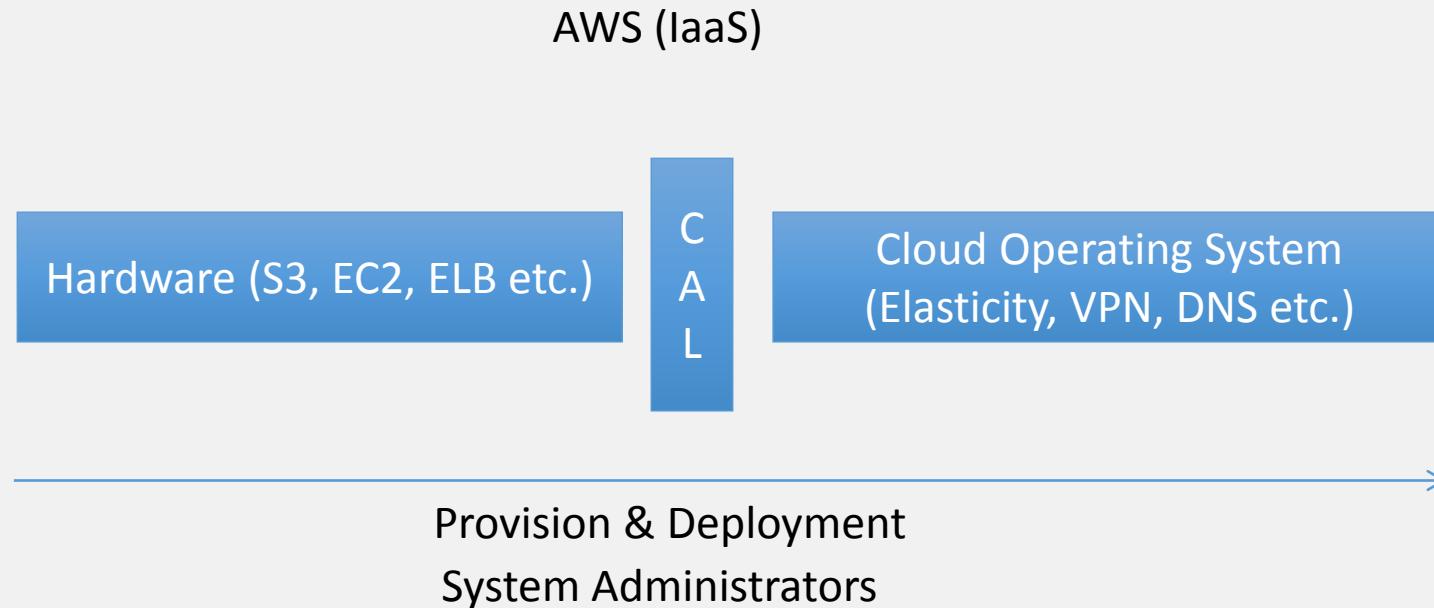
Dell

Hardware

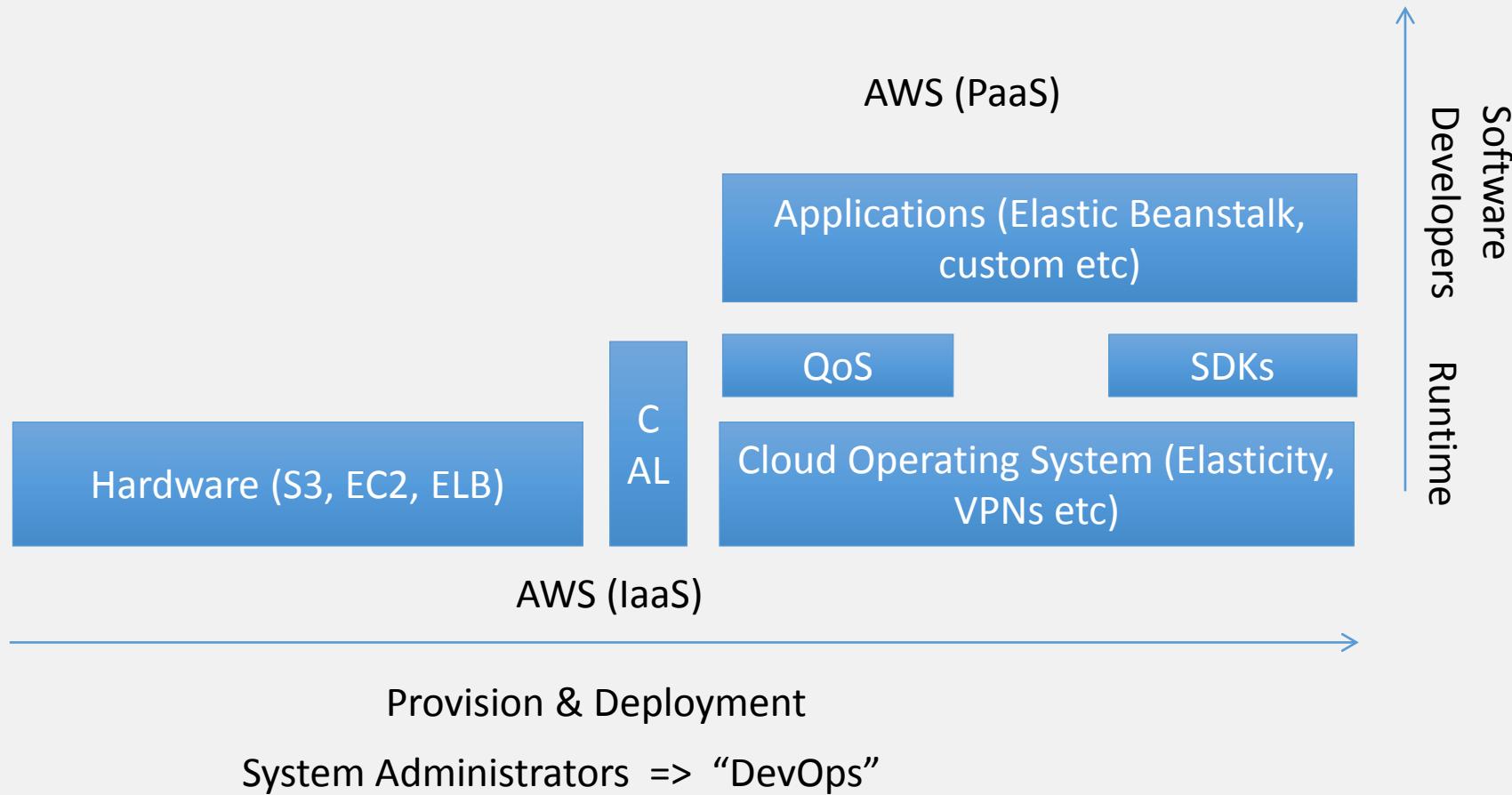
Desktop Platform



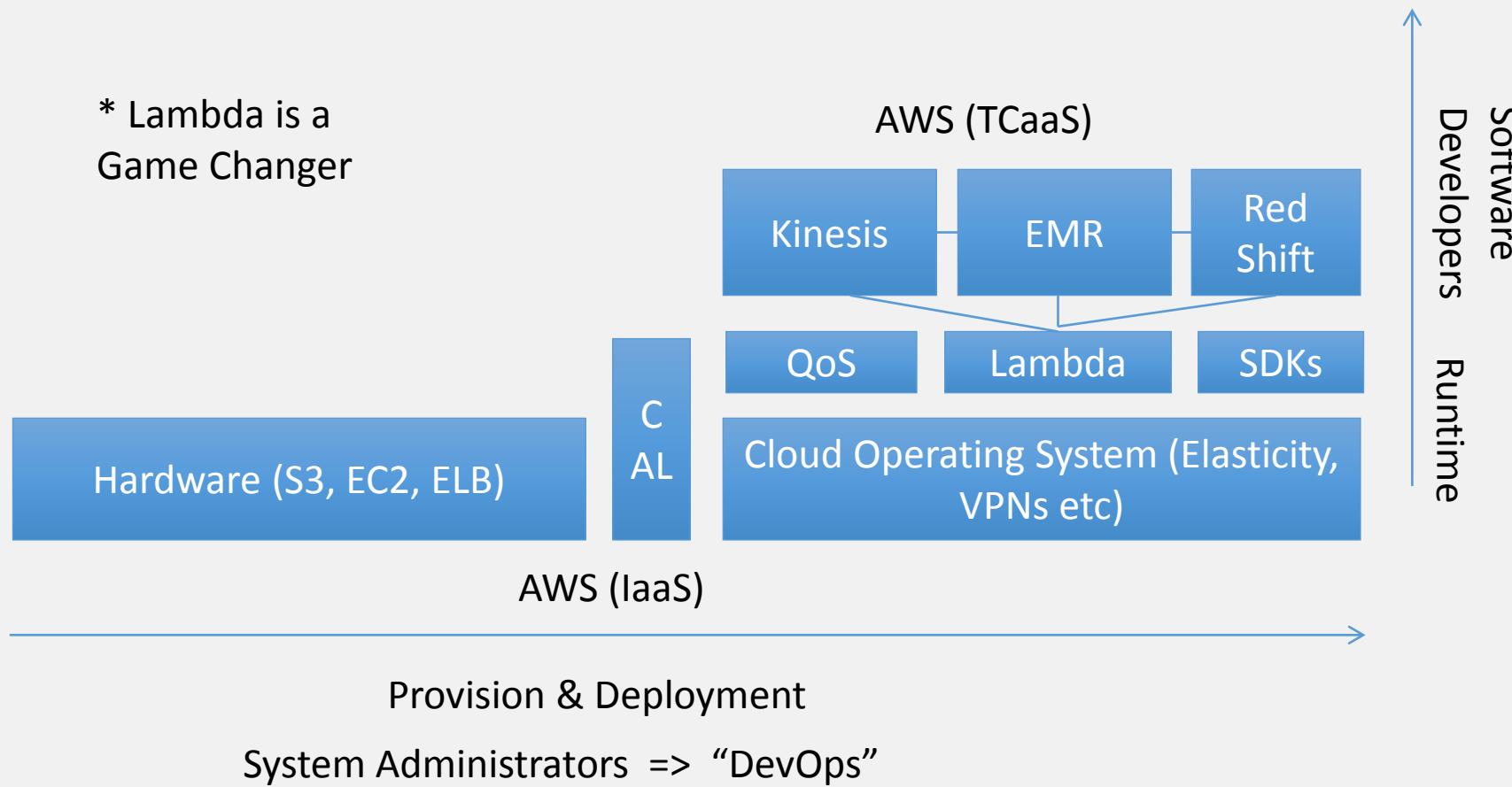
AWS Cloud Platform, 2010



AWS Cloud platform, 2014



AWS Cloud Platform 2016



AWS Elastic Beanstalk

- Deploy and scale web applications easily
- Languages: Java, .NET, PHP, Node.js, Python, Ruby, Docker
- Servers: Apache, Nginx, Phusion Passenger, IIS
- Simply upload your code; AWS handles:

Deployment

Auto scaling

Capacity Provisioning

Health Monitoring

Load balancing

AWS Lambda Event-driven Compute

- Runs stateless, request-driven code called **Lambda functions in Java, NodeJS & Python**
- Triggered by events (state transitions) in other AWS services
- Pay only for the requests served and the compute time
- Focus on business logic, not infrastructure.
- Just upload your code; AWS Lambda handles:

Capacity

Monitoring

Fault Tolerance

Scaling

Logging

Security Patching

Deployment

Web service front end

AWS Lambda Event Sources

- Amazon S3
- Amazon DynamoDB
- Amazon Kinesis Streams
- Amazon Simple Notification Service
- Amazon Simple Email Service
- Amazon Cognito
- AWS CloudFormation
- Amazon CloudWatch Logs
- Amazon CloudWatch Events
- Scheduled Events (powered by Amazon CloudWatch Events)
- AWS Config
- Amazon Echo
- Amazon API Gateway
- Other Event Sources: Invoking a Lambda Function On Demand
- Sample Events Published by Event Sources

AWS Lambda Execution Environment

- State-less functions
- You can use multi-threading, etc.
- 500 MB of /tmp storage space
- You set how much memory you need:
 - From 128 MB to 1.5GB
 - 64GB increments
 - CPU scales accordingly
- Function should finish in a certain time
 - Default 3 seconds, up to 300 seconds

AWS Lambda Pricing

- You pay per use of your function
- \$0.20 per 1 million function call
- Also, \$0.00001667 for every GB-second used



CLOUD COMPUTING APPLICATIONS

Amazon S3 BLOB Storage

Roy Campbell & Reza Farivar

Definition

Online file storage web service offered by Amazon Web Services.
Amazon S3 provides storage through web service interfaces
REST, SOAP, BitTorrent. (wikipedia)

<https://aws.amazon.com/s3/>

Use case

- Scalability, high availability, low latency – 99.99% availability
- Files up to 5 terabytes
- Objects stored in buckets owned by users
- User assigned keys refer to objects
- Amazon Machine Images (exported as a bundle of objects)
- SmugMug, Hadoop file store, Netflix, reddit, Dropbox, Tumbler

Simple Storage Service (S3)

- A **bucket** is a container for objects and describes location, logging, accounting, and access control.
 - A bucket has a name that must be **globally unique**.
 - `http://bucket.s3.amazonaws.com`
 - `http://bucket.s3-aws-region.amazonaws.com.`
- A bucket can hold any number of **objects**, which are files of up to 5TB.
 - `http://bucket.s3.amazonaws.com/object`
 - `http://johnsmith.s3.amazonaws.com/photos/puppy.jpg`

Fundamental operations corresponding to HTTP actions:

`http://bucket.s3.amazonaws.com/object`

- POST a new object or update an existing object.
- GET an existing object from a bucket.
- DELETE an object from the bucket
- LIST keys present in a bucket, with a filter.

A bucket has a **flat directory structure**

S3 Weak Consistency Model

“Updates to a single key are **atomic**....”

“Amazon S3 achieves high availability by replicating data across multiple servers within Amazon's data centers. If a PUT request is successful, your data is safely stored. However:

- A process writes a new object to Amazon S3 and immediately attempts to read it. Until the change is fully propagated, Amazon S3 might report "key does not exist."
- A process writes a new object to Amazon S3 and immediately lists keys within its bucket. Until the change is fully propagated, the object might not appear in the list.
- A process replaces an existing object and immediately attempts to read it. Until the change is fully propagated, Amazon S3 might return the prior data.
- A process deletes an existing object and immediately attempts to read it. Until the deletion is fully propagated, Amazon S3 might return the deleted data.”

S3 Command Line Interface

```
aws s3 mb s3://bucket
...
      cp localfile s3://bucket/key
      mv s3://bucket/key s3://bucket/newname
      ls s3://bucket
      rm s3://bucket/key
      rb s3://bucket
```

```
aws s3 help
aws s3 ls help
```



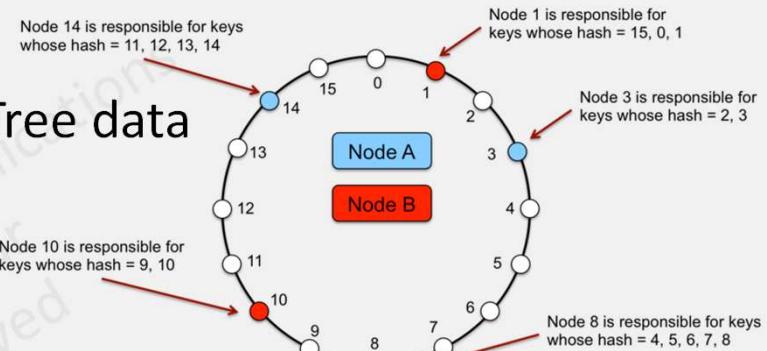
CLOUD COMPUTING APPLICATIONS

Serverless Storage: DynamoDB

Prof. Reza Farivar

DynamoDB

- DynamoDB is a fully managed NoSQL database provided by Amazon AWS
- Think of it as a massive distributed B-Tree data structure in the cloud
 - Accessing specific items is blazingly fast
- Distributed system
 - Using the consistent hashing algorithm in a ring



Usage model

- First create a table
 - Remember that it's a managed service, so just create a table using the console (or CLI, API)
 - We will use the Python Boto3 package in this lesson
 - While creating the table, define the primary key
 - This key will be used by DynamoDB to distribute key/values in different partitions
 - Optionally, identify a sort key
 - The sort key is used to keep the items in a partition sorted
 - Will be useful for query and scan later on



Using the table: Put

- Having defined the table, we can now put values into it.
 - Note: DynamoDB items are limited to 400KB size

```
import boto3
dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('users')

table.put_item(
    Item={
        'username': 'janedoe',
        'first_name': 'Jane',
        'last_name': 'Doe',
        'age': 25,
        'account_type': 'standard_user',
    }
)
```

Using the table: Get

- Retrieve an item

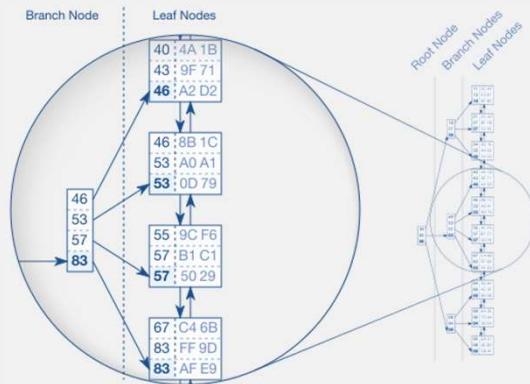
```
response = table.get_item(  
    Key={  
        'username': 'janedoe',  
        'last_name': 'Doe'  
    }  
)  
item = response['Item']  
print(item)
```

#Expected output:

```
{u'username': u'janedoe',  
 u'first_name': u'Jane',  
 u'last_name': u'Doe',  
 u'account_type': u'standard_user',  
 u'age': Decimal('25')}
```

Query and Scanning

- In RDBMS world*, query is usually defined as an operation where there is a usable index available, and we can quickly retrieve the item in $\text{Log}(n)$ time
- In comparison, scan happens where there is no usable index, and the engine has to read every record and test for a condition
- An RDBMS engine parses a SELECT statement, and performs query optimization, all behind the curtain
- DynamoDB just allows you to be your own database engine!



*See <https://use-the-index-luke.com/sql/where-clause/searching-for-ranges/greater-less-between-tuning-sql-access-filter-predicates>

Query

- Query only works on the primary key already defined for the table
- Or any other attribute that we have explicitly made a secondary index for it
- If a composite primary key was used (hash key + sort key), we can ask query to return a conditional range of value

```
response = table.query(  
    KeyConditionExpression=Key('username').eq('johndoe')  
)  
items = response['Items']  
print(items)  
  
#Expected output:  
  
[{u'username': u'johndoe',  
 u'first_name': u'John',  
 u'last_name': u'Doe',  
 u'account_type': u'standard_user',  
 u'age': Decimal('25'),  
 u'address': {u'city': u'Los Angeles',  
             u'state': u'CA',  
             u'zipcode': Decimal('90001'),  
             u'road': u'1 Jefferson Street'}}]
```

Scan

- What if we want to perform a query conditioned on attributes that there is no index for them?
- Scan will return everything!
 - It allows to filter based on any arbitrary condition

```
response = table.scan(  
    FilterExpression=Attr('age').lt(27)  
)  
items = response['Items']  
print(items)
```

Secondary Index

- Similar to the main index, requires a partition key and a sort key
- Local (LSI)
 - First released by Amazon in 2013
 - Immediately consistent
 - Once created, the table cannot grow any more
 - All the records that share the same partition key need to fit in 10GB
 - Once the allocation is full, writes fail ☹
- Global (GSI)
 - Came a few months after local indexes
 - Eventual consistency model
 - Do not constrain table size ☺



CLOUD COMPUTING APPLICATIONS

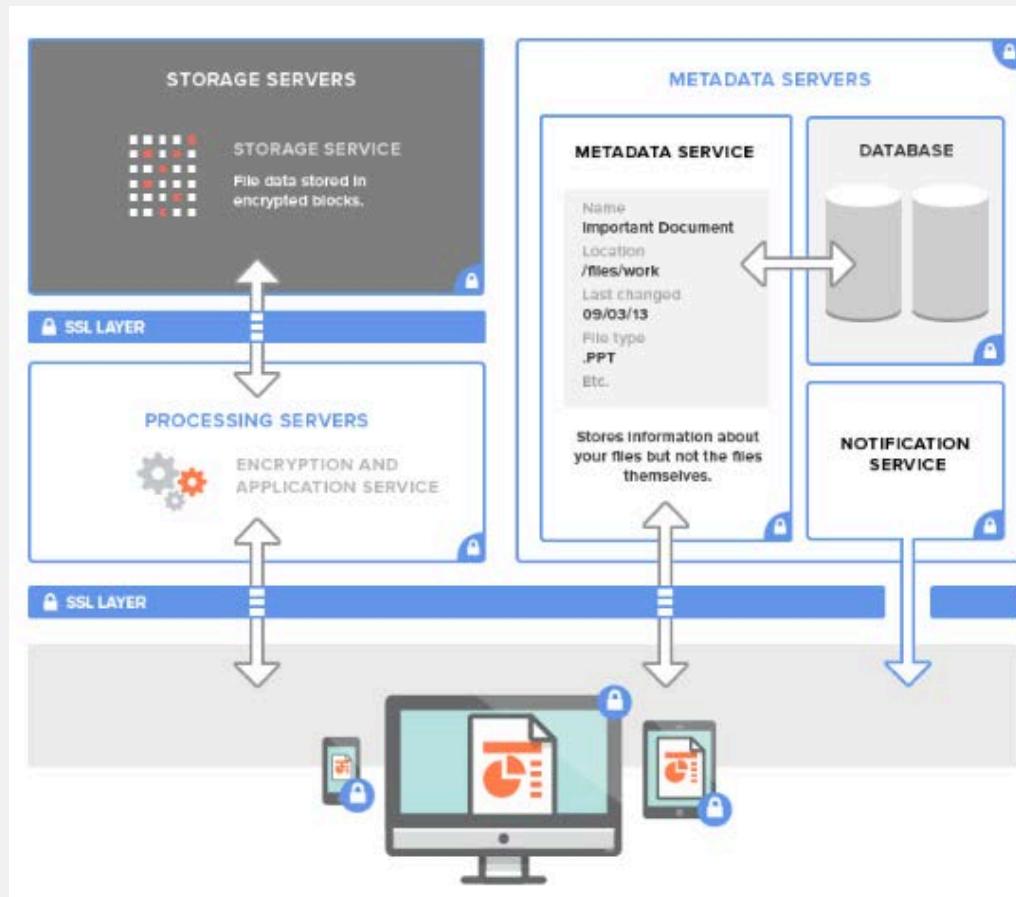
Roy Campbell & Reza Farivar

Dropbox Cloud API

Cloud Storage

- One interesting case study is Dropbox
- Dropbox offers cloud file storage
 - Easily synced across multiple devices
 - Accessible through web interface, mobile apps, and directly integrated with the file system on PCs
- Dropbox itself uses clouds!
 - Metadata stored in Dropbox servers
 - Actual files stored in Amazon S3
 - Amazon EC2 instances run the logic

Dropbox Architecture

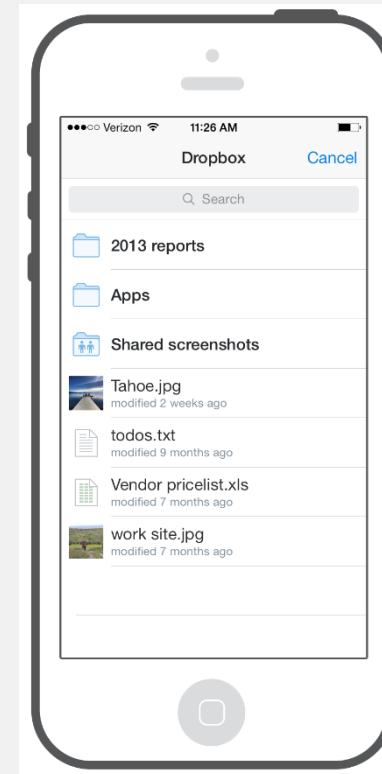


Dropbox API

- Two levels of API access to Dropbox
 - Drop-ins
 - Cross-platform UI components that can be integrated in minutes
 - *Chooser* allows instant access to files in Dropbox
 - *Saver* makes saving files to Dropbox easy
 - Core API
 - Support for advanced functionality like search, revisions, and restoring file
 - Better fit for deeper integration

Drop-In API

- Simple objects
 - Chooser available for JavaScript, Android and iOS
 - Saver on web and mobile web
- Handles all the authentication (OAuth), file browsing
- Chooser object returns the following:
 - Link: URL to access the file
 - File name
 - File Size
 - Icon
 - Thumbnails
- Saver
 - Pass in URL, filename and options



Core API

- Many languages and environments
 - Python, Ruby, PHP, Java, Android, iOS, OS X, HTTP
- Based on HTTP and OAuth
 - OAuth v1, OAuth v2
- Low-level calls to access and manipulate a user's Dropbox account
 - Create URL schemes
 - Upload files
 - Download files
 - List files and folders
 - Delta
 - Metadata access
 - Create and manage file sharing



CLOUD COMPUTING APPLICATIONS

MapReduce Introduction

Roy Campbell & Reza Farivar

Motivation: Large Scale Data Processing

- Many tasks composed of processing lots of data to produce lots of other data
- Large-Scale Data Processing
 - Want to use 1000s of CPUs
 - But don't want hassle of managing things

Motivation: Large Scale Data Processing

- MapReduce provides
 - User-defined functions
 - Automatic parallelization and distribution
 - Fault tolerance
 - I/O scheduling
 - Status and monitoring



CLOUD COMPUTING APPLICATIONS

MapReduce Motivation

Roy Campbell & Reza Farivar

Lesson Outline

- Motivation: Why MapReduce model
- Programming Model
- Examples
 - Word Count
 - Pi Estimation
 - Image Smoothing
 - PageRank
- MapReduce execution

Challenges with Traditional Programming Models (MPI)

- MPI gives you MPI_Send, MPI_Recieve
- Deadlock is possible...
 - Blocking communication can cause deadlock
 - "crossed" calls when trading information
 - example:
 - Proc1: MPI_Receive(Proc2, A); MPI_Send(Proc2, B);
 - Proc2: MPI_Receive(Proc1, B); MPI_Send(Proc1, A);
 - There are some solutions - MPI_SendRecv()
- Large overhead from comm. mismanagement
 - Time spent blocking is wasted cycles
 - Can overlap computation with non-blocking comm.
- Load imbalance is possible! Dead machines?
- Things are starting to look hard to code!

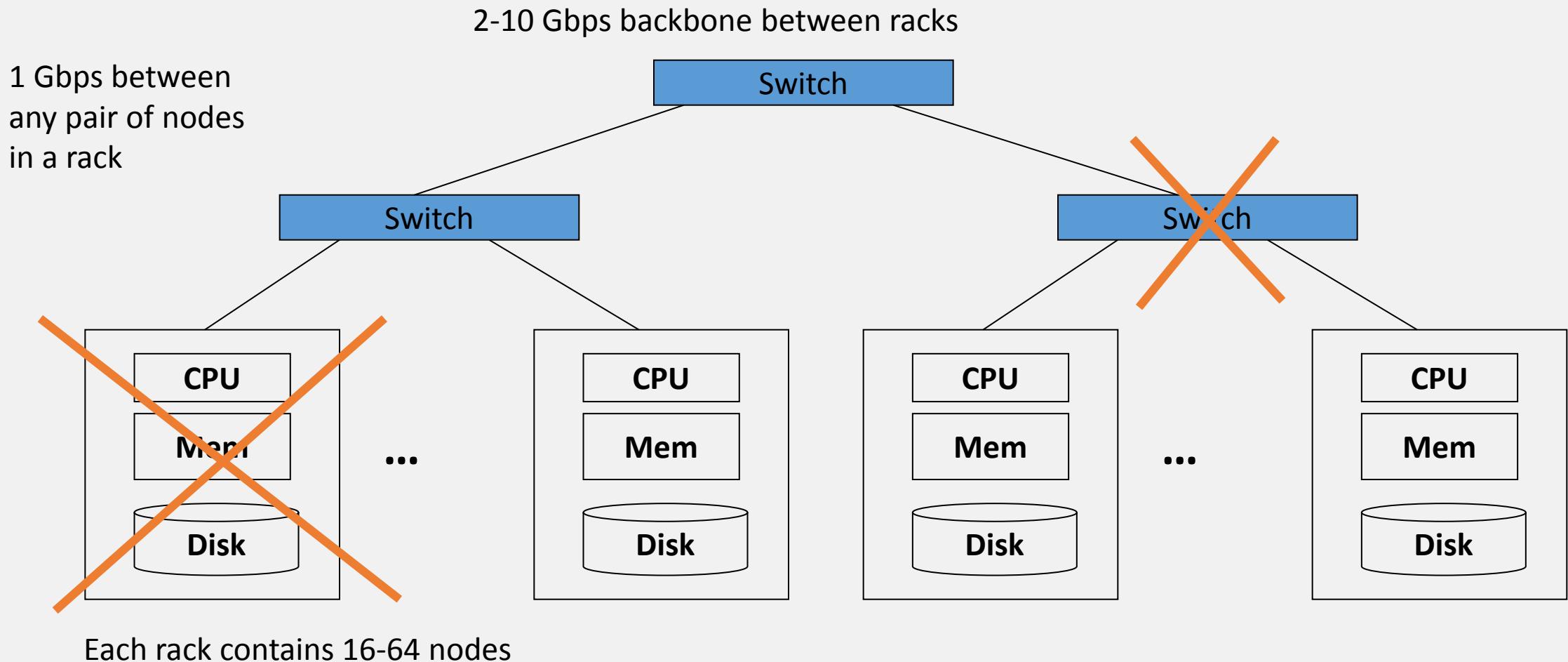
Commodity Clusters

- Web data sets can be very large
 - Tens to hundreds of terabytes
 - Cannot mine on a single server
- Standard architecture emerging:
 - Cluster of commodity Linux nodes
 - Gigabit Ethernet interconnect
- How to organize computations on this architecture?
 - Mask issues such as hardware failure

Solution

- Use distributed storage
 - 6-24 disks attached to a blade
 - 32-64 blades in a rack connected by Ethernet
- Push computations down to storage
 - Computations process contents of disks
 - Data on disks read sequentially from beginning to end
 - Rate limited by speed of disks (speed can get at data)

Cluster Architecture



Stable Storage

- First-order problem: if nodes can fail, how can we store data persistently?
- Answer: Distributed File System
 - Provides global file namespace
 - Google GFS / Hadoop HDFS
- Typical usage pattern
 - Huge files (100s of GB to TB)
 - Data is rarely updated in place
 - Reads and appends are common



CLOUD COMPUTING APPLICATIONS

MapReduce Programming Model

Roy Campbell & Reza Farivar

What is MapReduce?

- MapReduce
 - Programming model from LISP
- Many problems can be phrased this way
- **Easy** to distribute
 - Hides difficulty of writing parallel code
 - System takes care of load balancing, dead machines, etc.
- Nice retry & failure semantics

Programming Concept

- Map
 - **Perform** a function on **individual values** in a data set to create a **new list** of values
- Reduce
 - **Combine** values in a data set to create a new **value**

SQUARE X = X * X

MAP SQUARE [1,2,3,4,5]

RETURNS [1,4,8,16,25]

SUM= [All Elements in Array, Total+=]

REDUCE [1,2,3,4,5]

RETURNS 15

MapReduce Programming Model

Input & Output: Each a set of key/value pairs

Programmer specifies two functions:

MAP (IN_KEY, IN_VALUE)

LIST(OUT_KEY, INTERMEDIATE_VALUE)

- Processes input key/value pair
- Produces intermediate pairs

MapReduce Programming Model

Input & Output: Each a set of key/value pairs

Programmer specifies two functions:

REDUCE (IN_KEY, IN_VALUE)

LIST(OUT_KEY, INTERMEDIATE_VALUE)

- Combines all intermediate values for a particular key
- Produces a set of merged output values (usually just one)



CLOUD COMPUTING APPLICATIONS

MapReduce Example: Word Count

Roy Campbell & Reza Farivar

Word Count

- Have a large file of words, many words in each line
- Count the number of times each distinct word appears in the file

Word Count Program

MAP(KEY = LINE, VALUE = CONTENTS):

FOR EACH WORD W IN VALUE:

EMIT INTERMEDIATE(w,1)

REDUCE(KEY, VALUES):

//key: a word; values: an iterator

//over counts

RESULT = 0

FOR EACH V IN INTERMEDIATE VALUES:

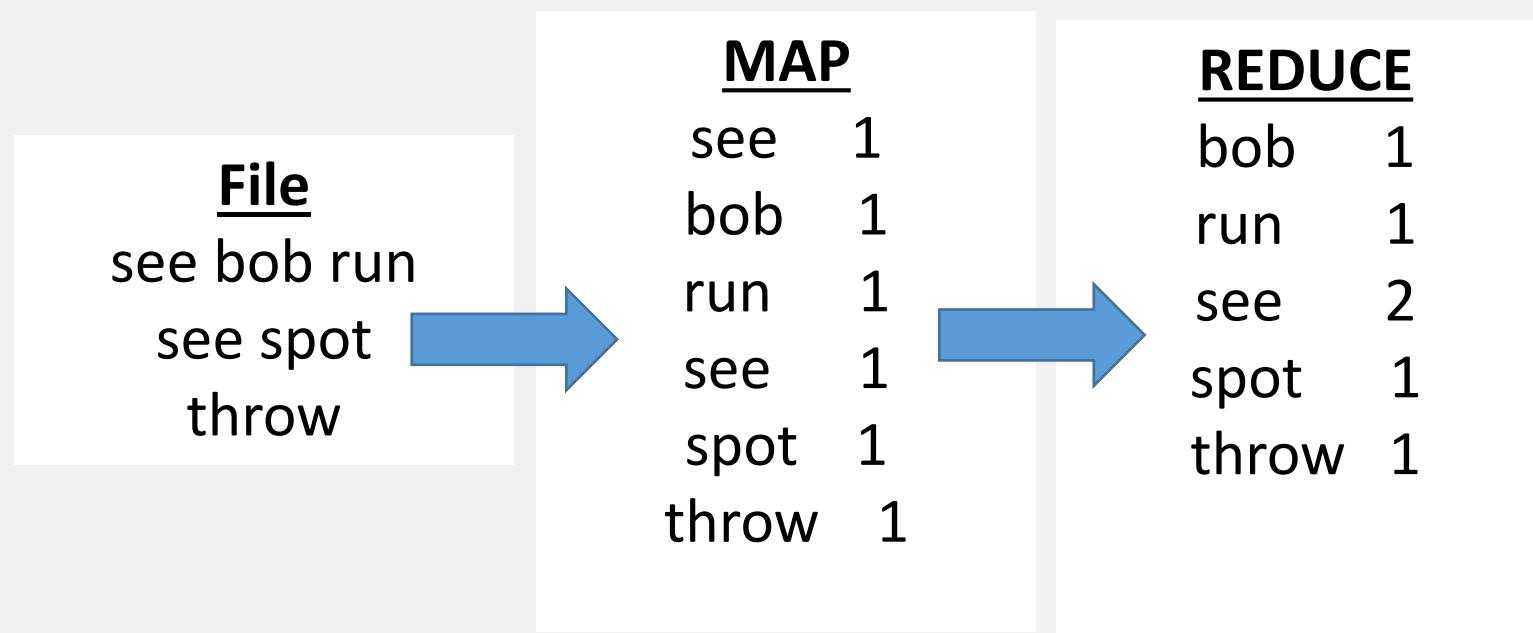
RESULT += V

EMIT(KEY, RESULT)

Word Count Illustrated

map(key=line, values=contents):
for each **word** w in contents:
emit(w,"1")

reduce(key=line, values=uniq_counts):
sum all 1's in **values** list
emit result (word, sum)





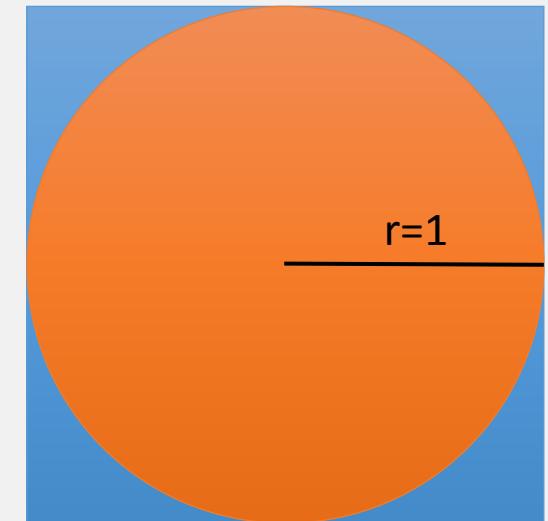
CLOUD COMPUTING APPLICATIONS

MapReduce Example:
Pi Estimation & Image Smoothing

Roy Campbell & Reza Farivar

Pi Estimation

- Using Monte Carlo simulation, estimate the value of π
- Throw darts
- Compute the ratio of the darts landed within the square vs. the darts landed within the circle
- Evaluating whether a particular dart landed within the circle is easy



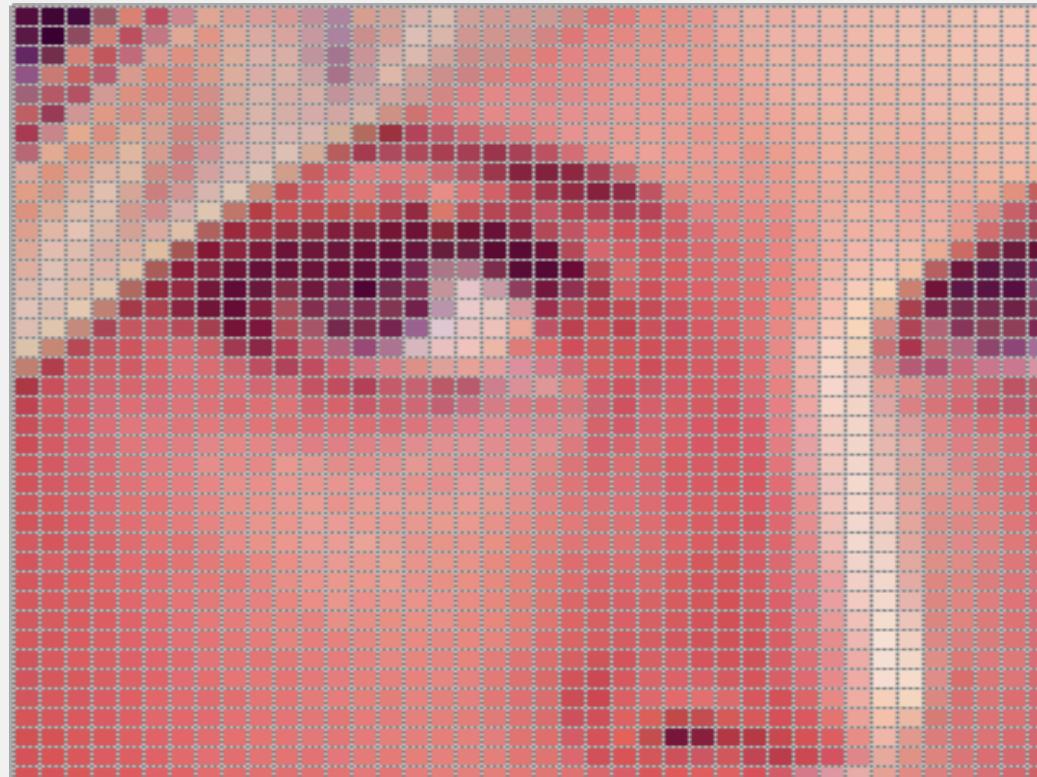
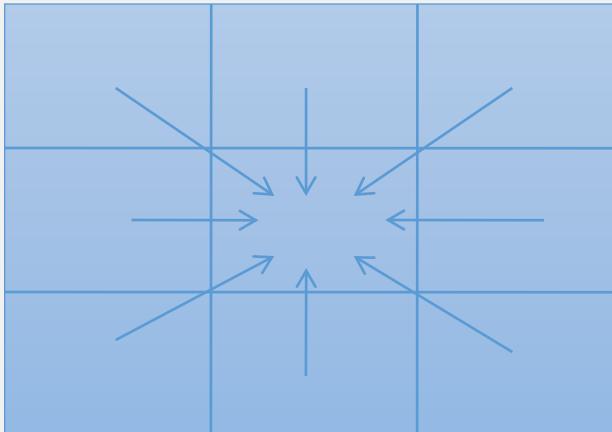
$$\begin{aligned}\text{Circle Area} &= \pi \\ \text{Square Area} &= 4 \\ \pi &= 4.C/S\end{aligned}$$

Pi Estimation

- **Mapper:** Generate points in a unit square and then count points inside/outside of the inscribed circle of the square
- **Reducer:** Accumulate points inside/outside results from the mappers
- After the MapReduce job, estimate Pi
 - The fraction **NumInside/NumTotal** is an approximation of the value
 - $(\text{Area of the circle}) / (\text{Area of the square})$
 - Then, Pi estimated value to be **(NumInside/NumTotal)**

Exercise 2: Image Smoothing

- To smooth an image, use a sliding mask and replace the value of each pixel



Exercise 2: Image Smoothing

- Map: input key = x, y input value = R, G, B
 - Emit 9 points
 - $(x-1, y-1, R, G, B)$
 - $(x, y-1, R, G, B)$
 - $(x+1, y-1, R, G, B)$
 - Etc.
- Reduce: input key = x, y input value: list of R, G, B
 - Compute average R, G, B
 - Emit key = x, y value = average R, G, B



CLOUD COMPUTING APPLICATIONS

MapReduce Example: Page Rank

Roy Campbell & Reza Farivar

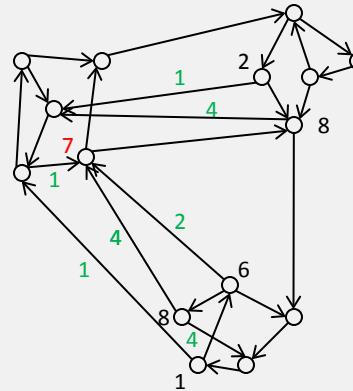
PageRank Algorithm

- Program implemented by Google to rank any type of recursive “documents” using MapReduce
 - Initially developed at Stanford University by Google founders, Larry Page and Sergey Brin, in 1995
 - Led to a functional prototype named Google in 1998
-
- PageRank value for a page u is dependent on the PageRank values for each page v out of the set B_u (all pages linking to page u), divided by the number $L(v)$ of links from page v

$$PR(u) = \sum_{v \in B_u} \frac{PR(v)}{L(v)}$$

PageRank

- Phase 1: Propagation
 - Phase 2: Aggregation
-
- Input: A pool of objects, including both vertices and edges



PageRank: Propagation

- Map: for each object
 - If object is vertex, emit key=URL, value=object
 - If object is edge, emit key=source URL, value=object
- Reduce: (input is a web page and all the outgoing links)
 - Find the number of edge objects → outgoing links
 - Read the PageRank value from the vertex object
 - Assign $PR(\text{edges}) = PR(\text{vertex}) / \text{num_outgoing}$

PageRank: Aggregation

- Map: for each object
 - If object is vertex, emit key=URL, value=object
 - If object is edge, emit key=destination URL, value=object
- Reduce: (input is a web page and all the incoming links)
 - Add the PR value of all incoming links
 - Assign $\text{PR}(\text{vertex}) = \sum \text{PR}(\text{incoming links})$



CLOUD COMPUTING APPLICATIONS

Summary

Roy Campbell & Reza Farivar

MapReduce Advantages/Disadvantages

- Now it's easy to program for many CPUs
 - Communication management effectively gone
 - I/O scheduling done for us
 - Fault tolerance, monitoring
 - Machine failures, suddenly slow machines, etc., are handled
 - Can be much easier to design and program
 - Can cascade several MapReduce tasks
- But ... it further restricts solvable problems
 - Might be hard to express problem in MapReduce
 - Data parallelism is key
 - Need to be able to break up a problem by data chunks

MapReduce Conclusions

- MapReduce has proven to be a useful abstraction
- Greatly simplifies large-scale computations
- Functional programming paradigm can be applied to large-scale applications
- Fun to use: focus on problem, let the middleware deal with messy details

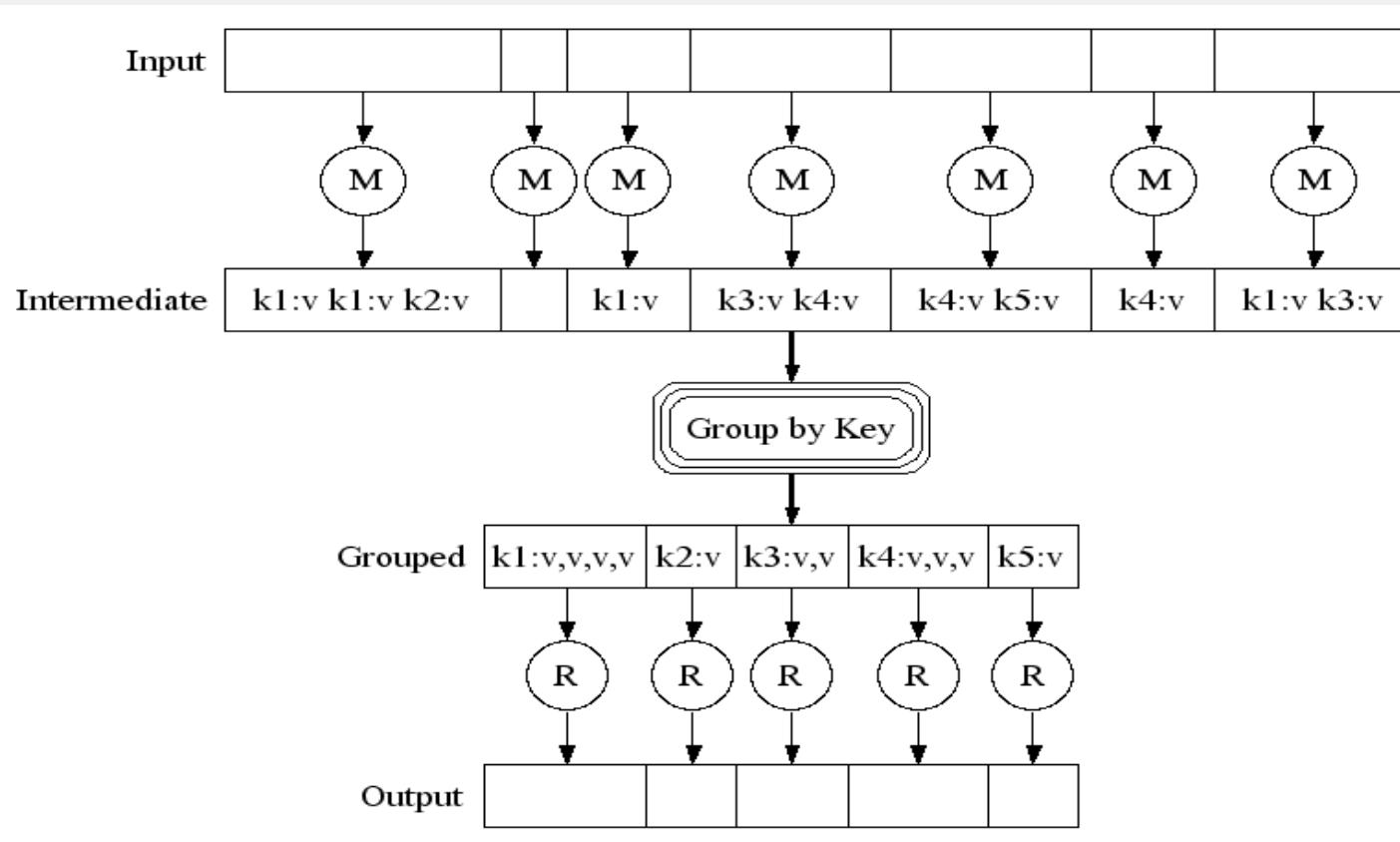


CLOUD COMPUTING APPLICATIONS

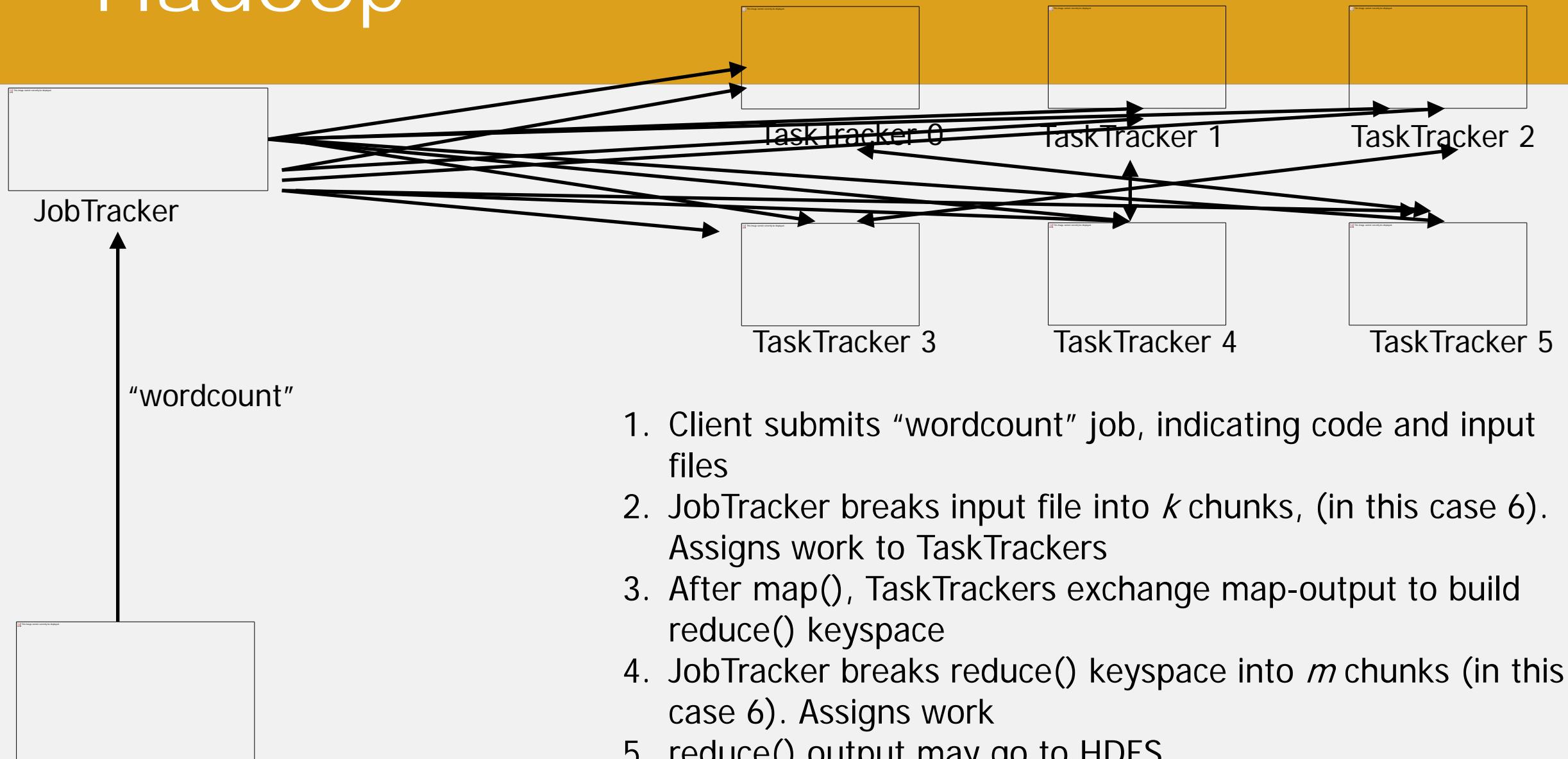
Roy Campbell & Reza Farivar

Hadoop Introduction

Execution



Hadoop



Execution Initialization

- Split input file into 64MB sections (GFS)
 - Read in parallel by multiple machines
- Fork off program onto multiple machines
- One machine is Master
- Master assigns idle machines to either Map or Reduce tasks
- Master coordinates data communication between map and reduce machines

Partition Function

- Inputs to map tasks are created by contiguous splits of input file
- For reduce, we need to ensure that records with the same intermediate key end up at the same worker
- System uses a default partition function e.g., $\text{hash}(\text{key}) \bmod R$
- Sometimes useful to override
e.g., $\text{hash}(\text{hostname}(\text{URL})) \bmod R$ ensures URLs from a host end up in the same output file

Map-Machine

- Reads contents of assigned portion of input file
- Parses and prepares data for input to map function (e.g., read `<a />` from HTML)
 - Classes implementing `InputFormat`
- Passes data into map function and saves result in memory (e.g., `<target, source>`)
- Periodically writes completed work to local disk
- Notifies Master of this partially completed work (intermediate data)

Reduce-Machine

- Receives notification from Master of partially completed work
- Retrieves intermediate data from Map-Machine via remote-read
- Sorts intermediate data by key (e.g., by target page)
- Iterates over intermediate data
 - For each unique key, sends corresponding set through reduce function
- Appends result of reduce function to final output file (GFS)

Data Flow

- Input, final output are stored on a distributed file system
 - Scheduler tries to schedule map tasks “close” to physical storage location of input data
- Intermediate results are stored on local FS of map and reduce workers
- Output is often input to another map reduce task



CLOUD COMPUTING APPLICATIONS

BIG DATA PIPELINES:
THE MOVE TO HADOOP

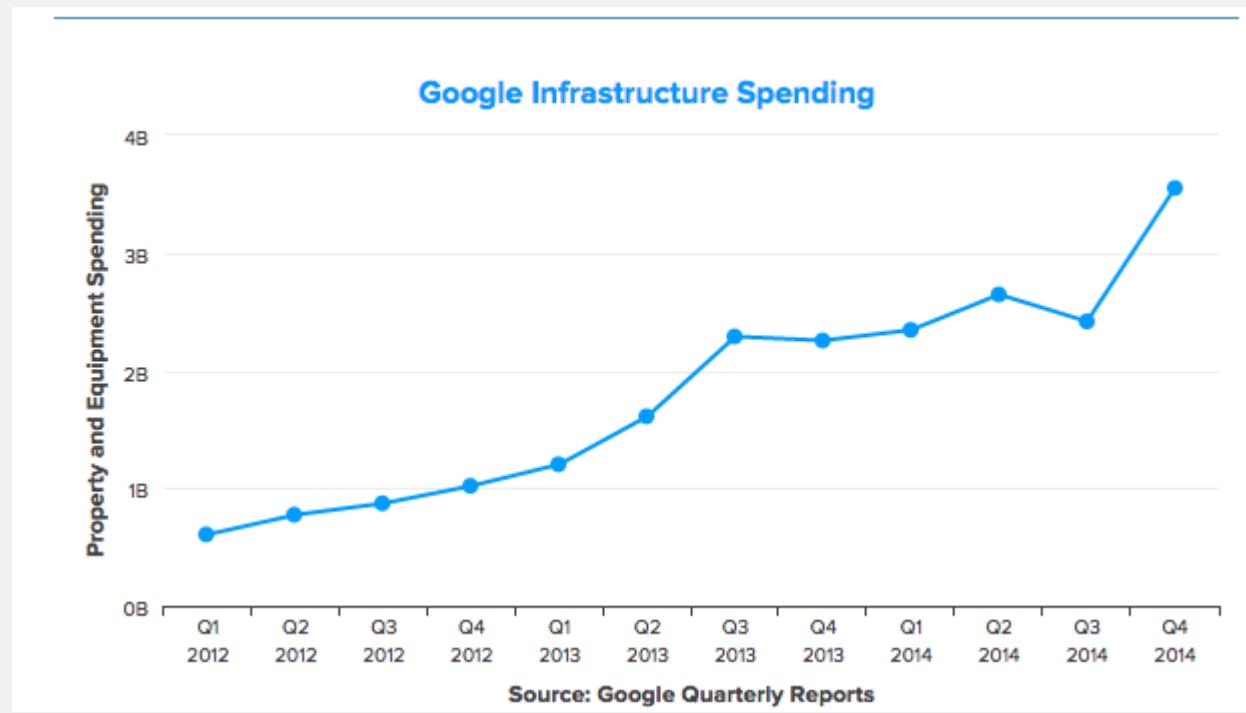
Matt Ahrens – Yahoo

Why Pipelines Are Behind Everything

- With the rise of large data sets, there needs to be a system that can reliably and quickly organize the data
- “Big data” is the trend in the industry, but how do you actually obtain data that is useful on a regular basis?
- Use cases
 - Relevant content tailored to users
 - Programmatic digital advertising
 - Data analytics for research and sciences

Why Pipelines Are Behind Everything

- Data keeps growing



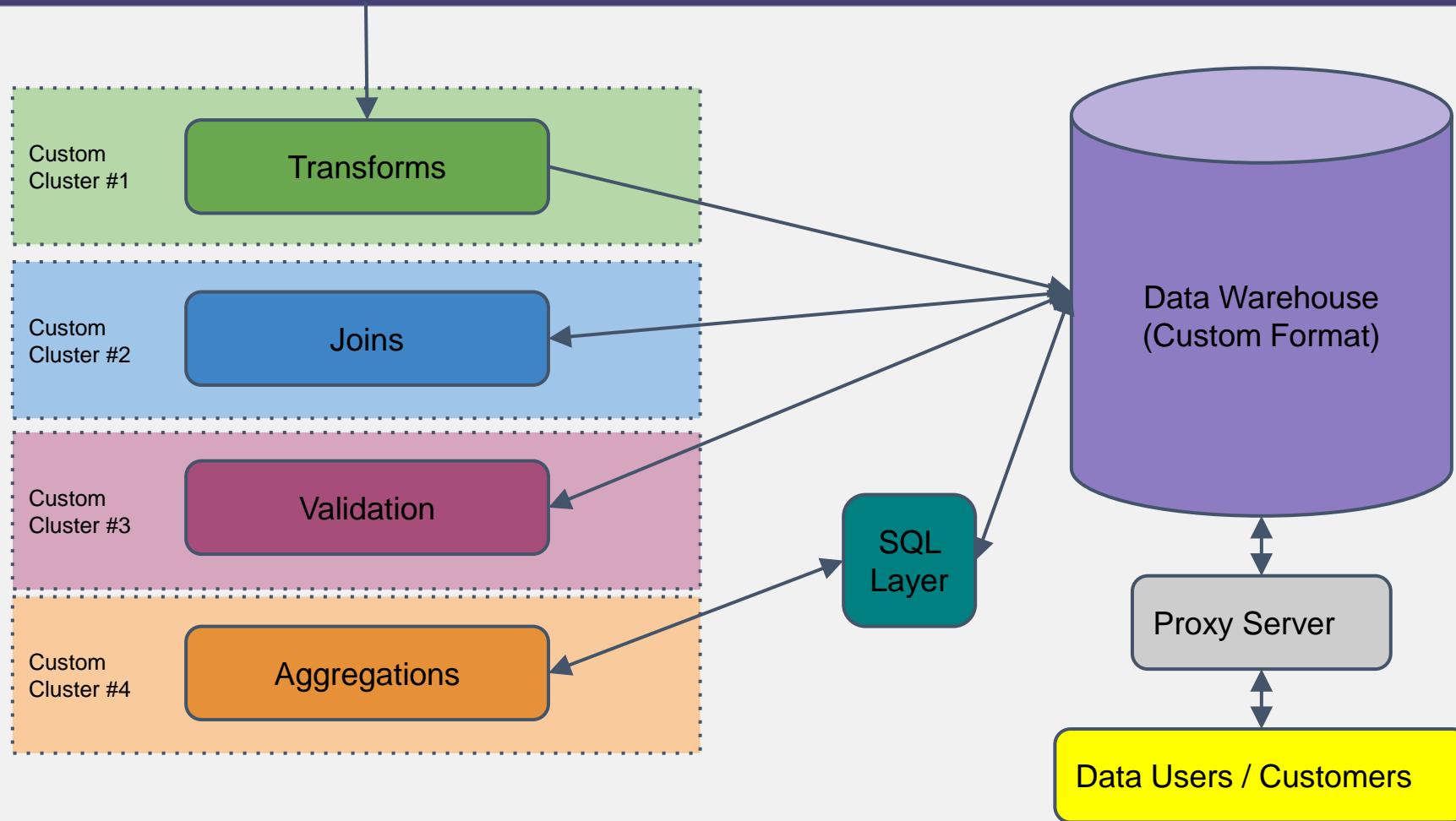
What Is a Data Pipeline

- Simple definition: system that transforms events into a usable format
- Input: raw logs, interactions, activities
- Output: data sets for specific users (filtered, aggregated, joined, etc.)
- Data size scale
 - Billions of transactions per day (millions / minute)
 - TBs of data per day (GBs / minute)

Where We Came From

- Customized mini-clusters of hardware
 - Tailored to specific type of job: transformation, joins, aggregation
 - Pro: mix of memory/cpu config specific for job type
 - Con: scale issues, overhead of HW setup/maintenance
- Lack of well-defined interfaces and APIs
 - No standard schema format or data model
- Data access limitations
 - Access was limited to core developers with advanced data and programming knowledge

The Past Architecture



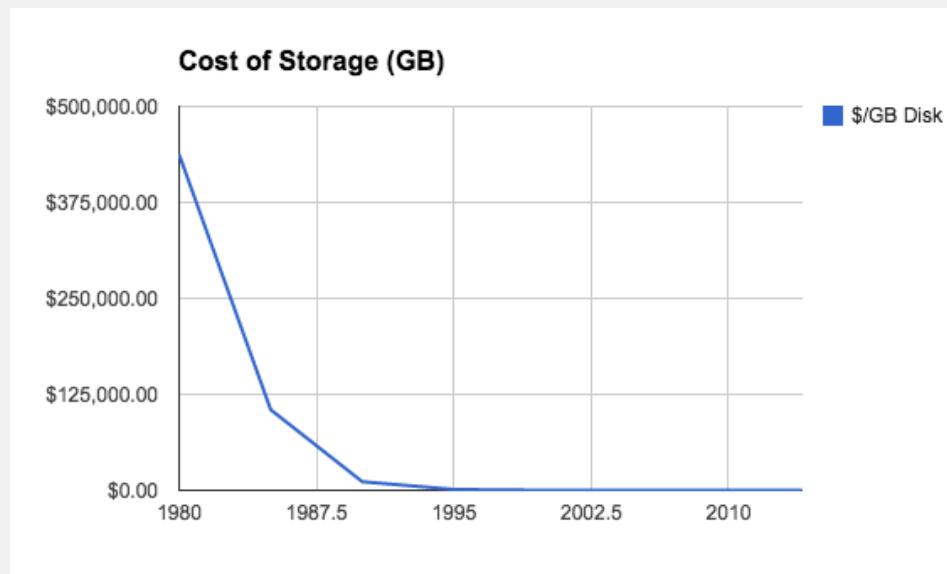
The Elephant Comes Into The Room



Why Move To Hadoop?

- Legacy systems were not performing well (< 1 TB / day)
- We had customers who wanted access to raw feeds (TB / day per customer)
- The advertising roadmap called for a 3-5x increase in traffic (new features, new customers onboarding)

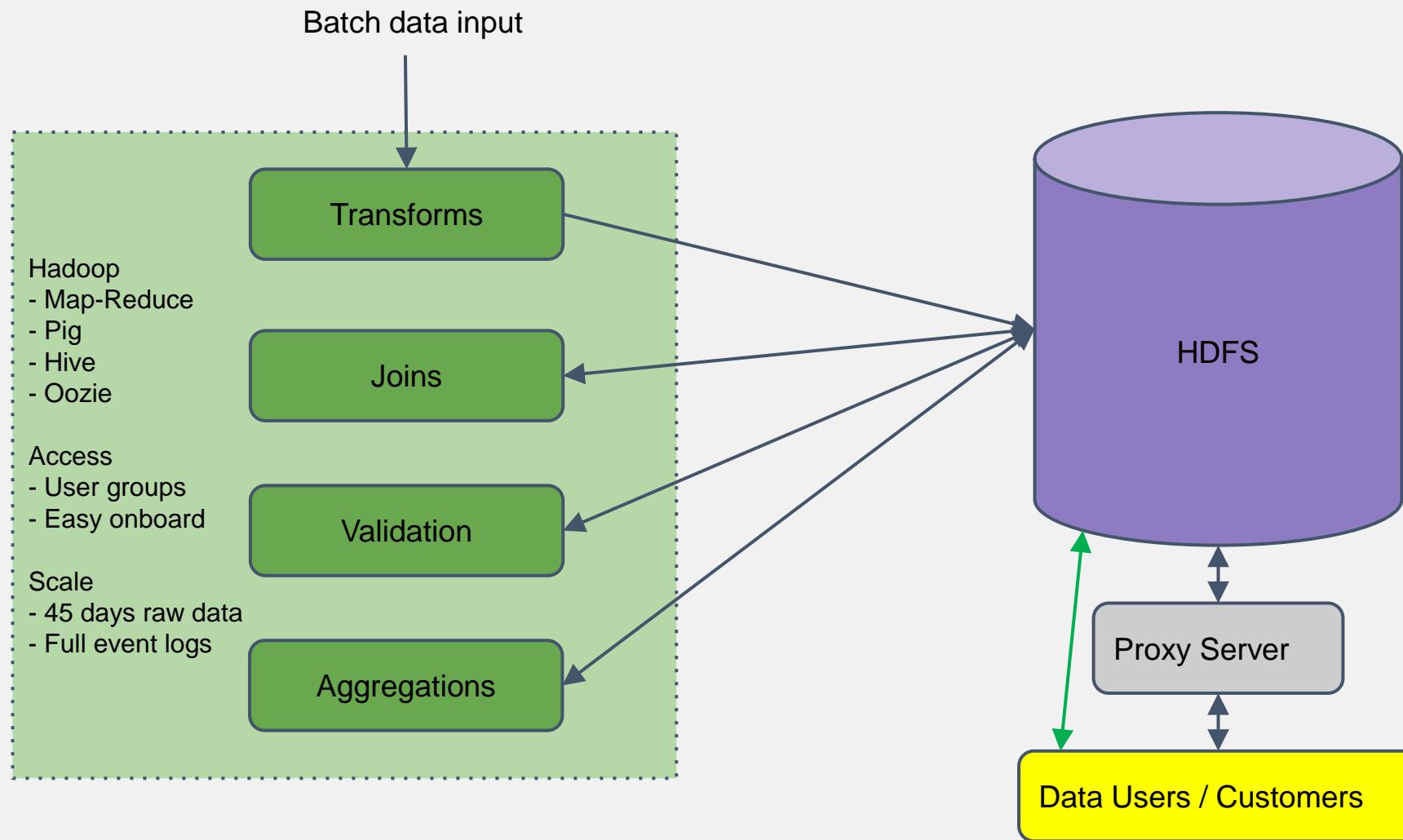
Year	\$/GB Disk
1980	\$437,500.00
1985	\$105,000.00
1990	\$11,200.00
1995	\$1,120.00
2000	\$11.00
2005	\$1.24
2010	\$0.09
2013	\$0.05
2014	\$0.03



The Promise of Hadoop

- PB+ storage capabilities
 - Multi-tenant internal clusters made up of 1000s of nodes could handle TB/day easily
 - Storage was fault-tolerant with default 3x replication
 - Easy to scale up as new growth occurred
- Hosted service for job execution and data storage
 - No more need for separate clusters as Map/Reduce could handle all types of jobs
 - ETL operations easily handled using Pig Latin interface
 - New innovative frameworks were starting up (HBase, Hive, Oozie) promising more platform adoption

The Architecture on Hadoop



Life on Hadoop

- Platform hardening had its consequences
 - Migrating data users and customers to the new system took longer than expected
 - Running large-scale data pipelines on multi-tenant clusters caused customer issues
- Data for everyone (who is permitted)
 - Number of data users increased dramatically on Hadoop
- Scaled better than expected (over past 5 years)
 - As data size has continued to grow, job runtime and data latency has continued to shrink

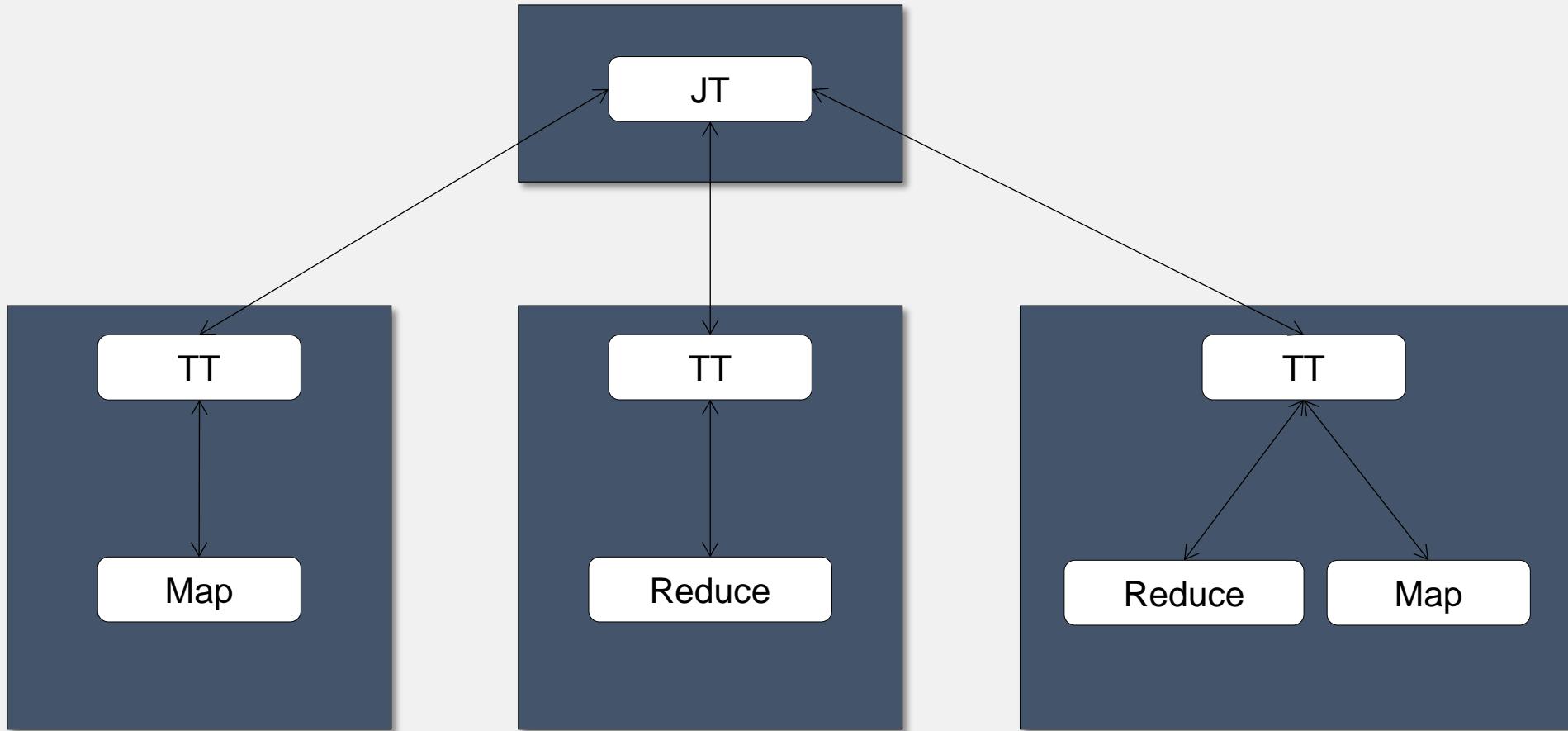


CLOUD COMPUTING APPLICATIONS

YARN Introduction

Roy Campbell & Reza Farivar

Hadoop 1.x



Issues with Hadoop

- Hadoop JobTracker was a barrier for scaling
 - Primary reason Hadoop 1.x is recommended for clusters no larger than 4000 nodes
 - Thousands of applications each running tens of thousands of tasks
 - JobTracker not able to schedule resources as fast as they became available
 - Distinct map and reduce slots led to artificial bottlenecks and low cluster utilization

Issues with Hadoop

- MapReduce was being abused by other application frameworks
 - Frameworks trying to work around sort and shuffle
 - Iterative algorithms were suboptimal
- YARN strives to be application framework agnostic
- Different application types can share the same cluster
- Runs MapReduce “out of the box” as part of Apache Hadoop

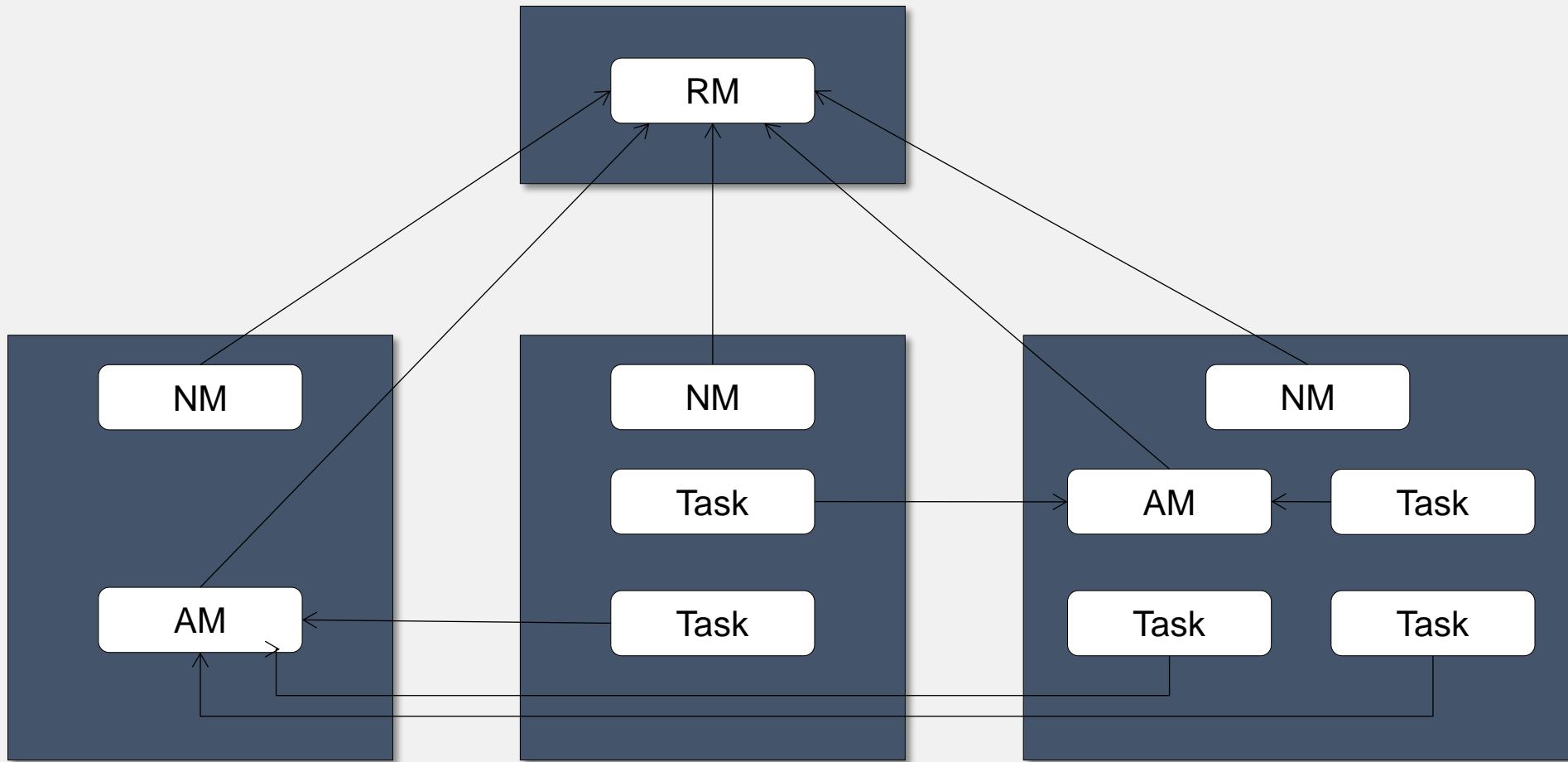
What is YARN?

- Yet Another Resource Negotiator
- Provides resource management services
 - Scheduling
 - Monitoring
 - Control
- Replaces the resource management services of the JobTracker
- Bundled with Hadoop 0.23 and Hadoop 2.x

YARN High-Level Architecture

- ResourceManager
 - Single, centralized daemon for scheduling containers
 - Monitors nodes and applications
- NodeManager
 - Daemon running on each worker node in the cluster
 - Launches, monitors, and controls containers
- ApplicationMaster
 - Provides scheduling, monitor, control for an application instance
 - RM launches an AM for each application submitted to the cluster
 - AM requests containers via RM; launches containers via NM
- Containers
 - Unit of allocation and control for YARN
 - ApplicationMaster and application-specific tasks run within containers

YARN High-Level Architecture



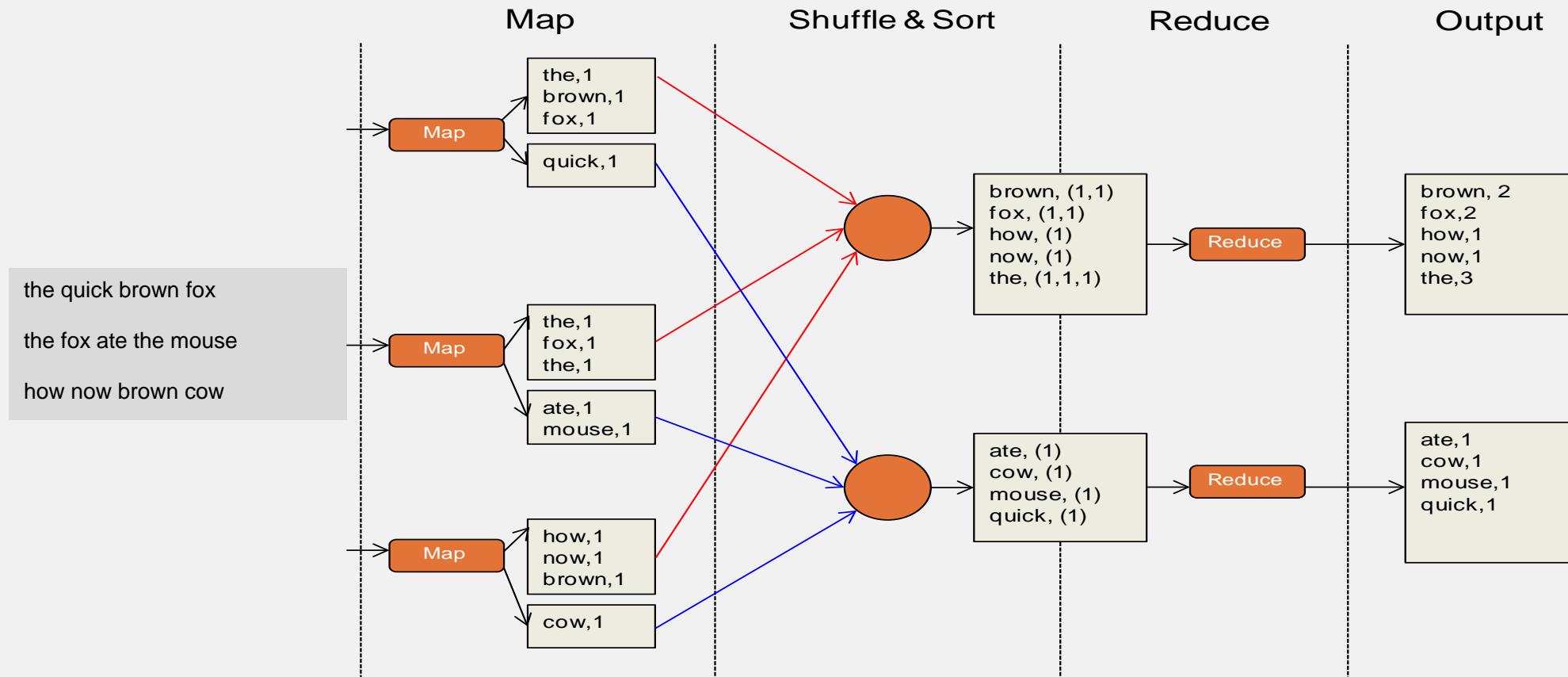


CLOUD COMPUTING APPLICATIONS

Roy Campbell & Reza Farivar

YARN: MapReduce on YARN

MapReduce



MapReduce on YARN

- MapReduce AM determines number of map and reduce tasks
 - Split metainfo file indicates number of map tasks based on number of splits
 - Job config determines number of reducers
- AM schedules when to request containers for map and reduce tasks
 - Split metainfo file has data locality for each map task
 - Reducers have no locality
 - Uses headroom provided by RM to avoid livelocks where reducers consume all available resources but more maps need to run

MapReduce on YARN

- Tasks connect back to AM upon startup via TaskUmbilicalProtocol
 - Report progress, liveness
 - AM kills tasks that do not report progress in a timely manner
 - AM provides reducers with shuffle data locations
 - Reducers notify AM of shuffle fetch failures; AM relaunches map tasks if necessary

MapReduce on YARN

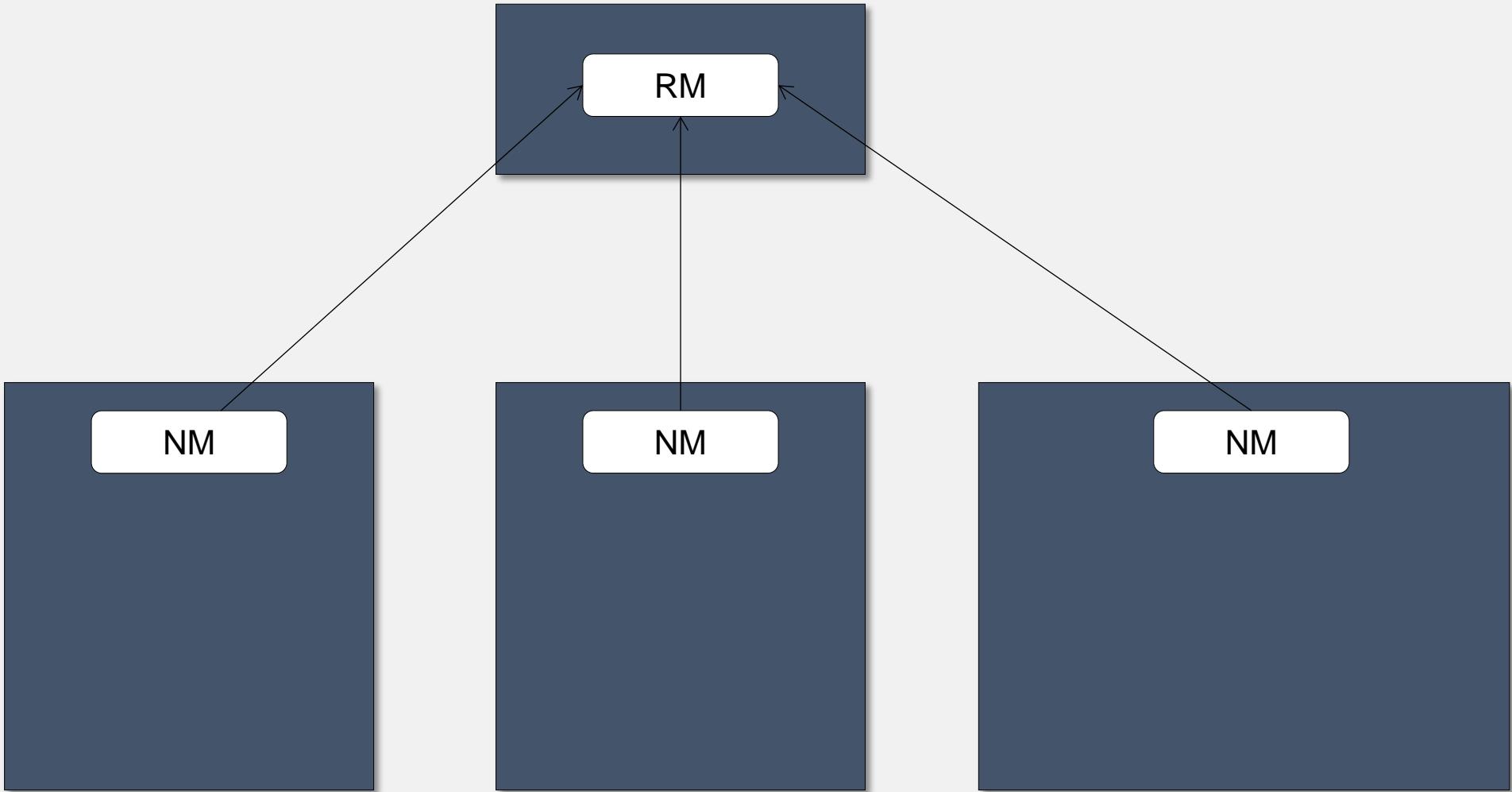
- Shuffle provided as a plugin service to NodeManagers
 - Shuffle port configurable, passed to reducers via AM
- AM responsible for job history
 - Job history events written to a file as job progresses
 - Copied to a drop location in HDFS when job completes
 - Used to provide recovery when AM crashes and is retried by RM
- MapReduce AM provides client interface
 - Report job and tasks status
 - Kill job or task attempts
 - Web app and services
 - Client can redirect to job history server if application has completed



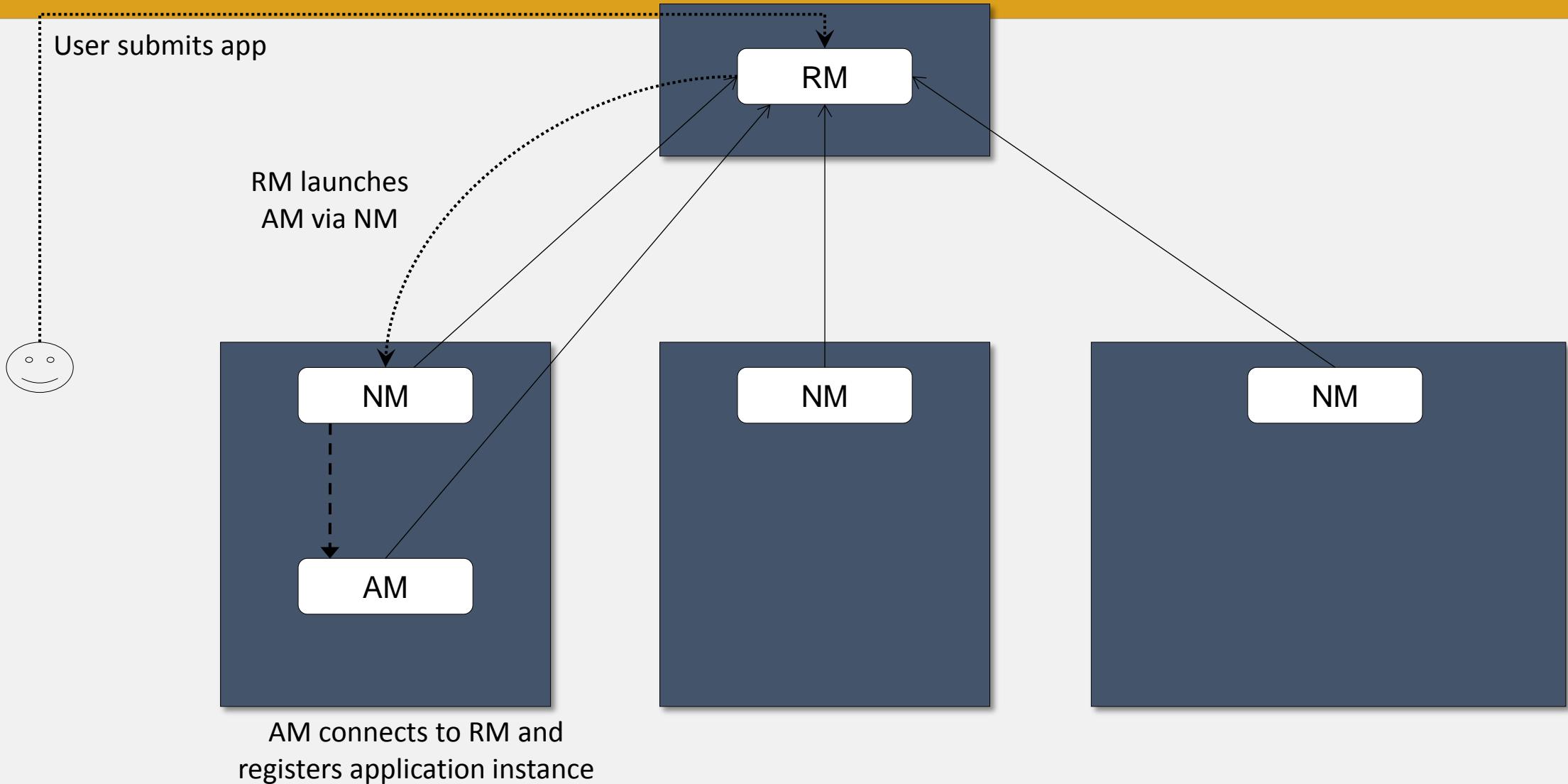
CLOUD COMPUTING APPLICATIONS

YARN: MapReduce on YARN in Diagram

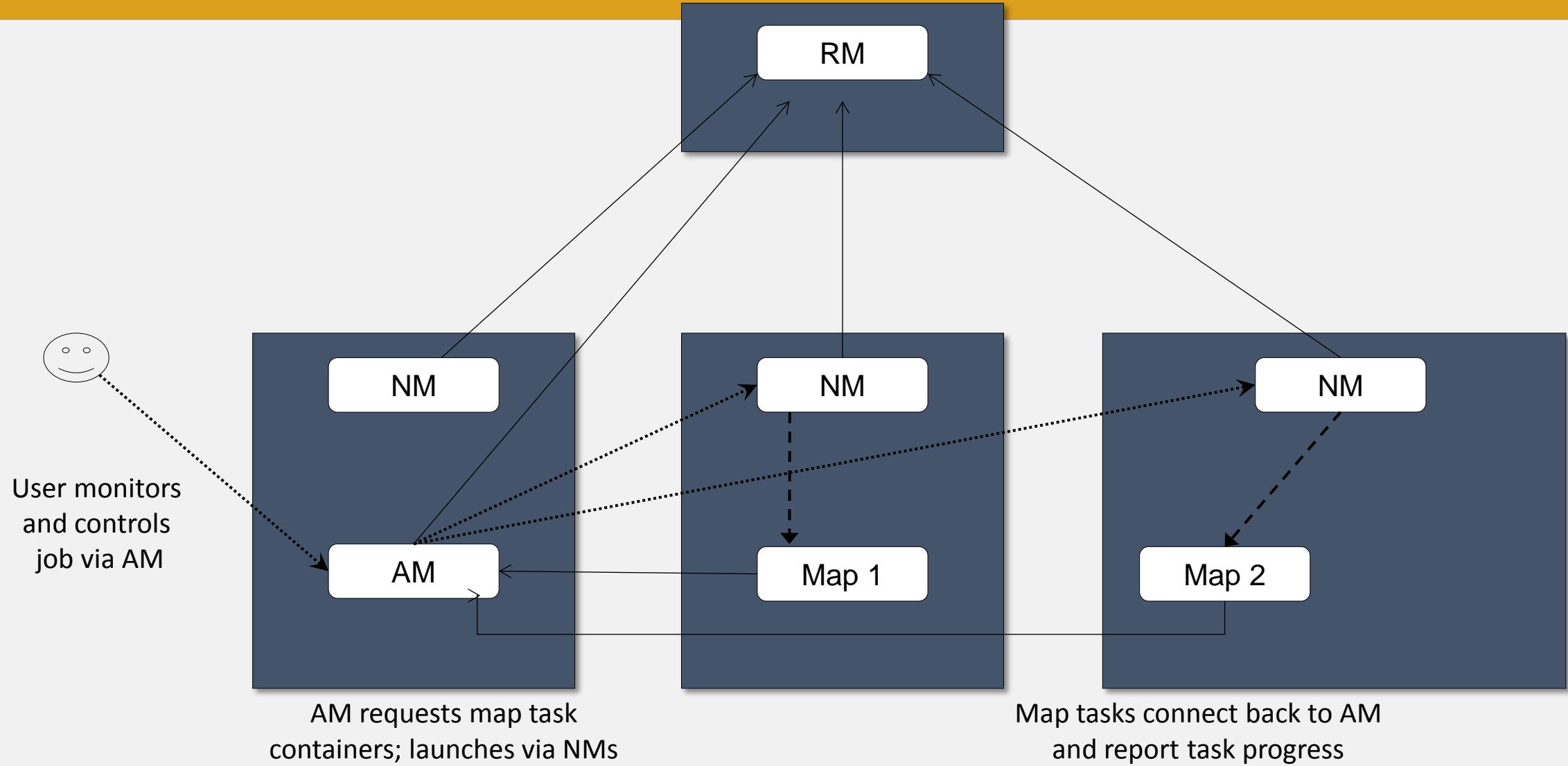
MapReduce on YARN



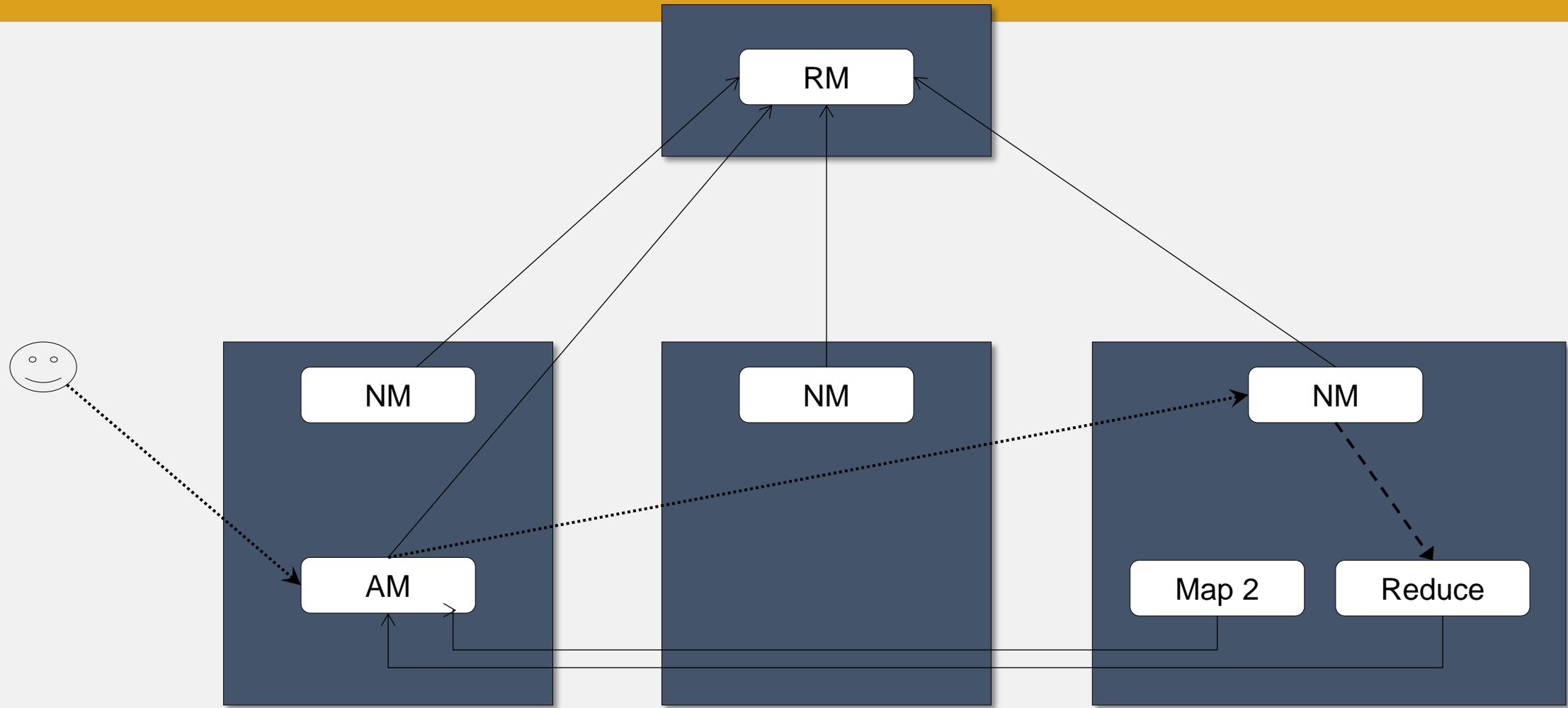
MapReduce on YARN



MapReduce on YARN

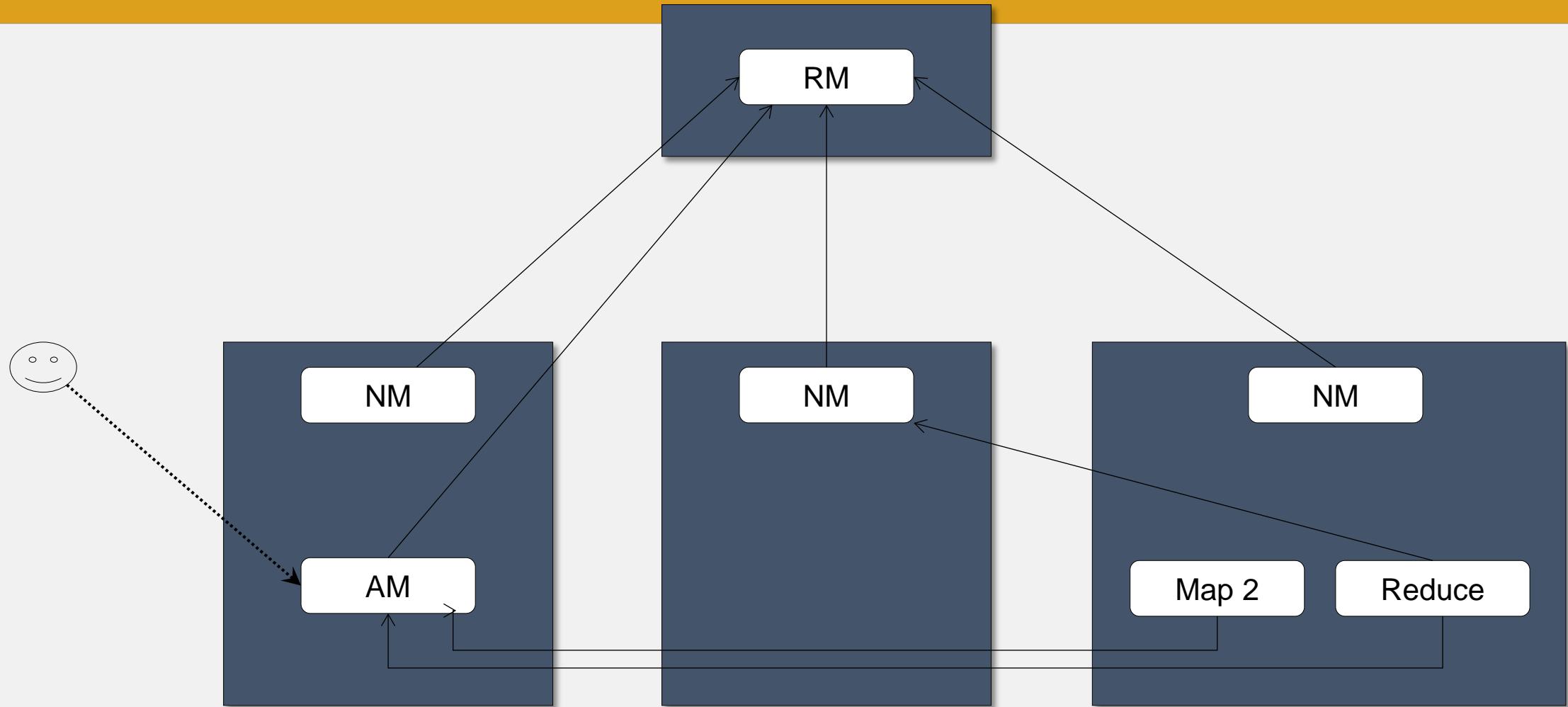


MapReduce on YARN



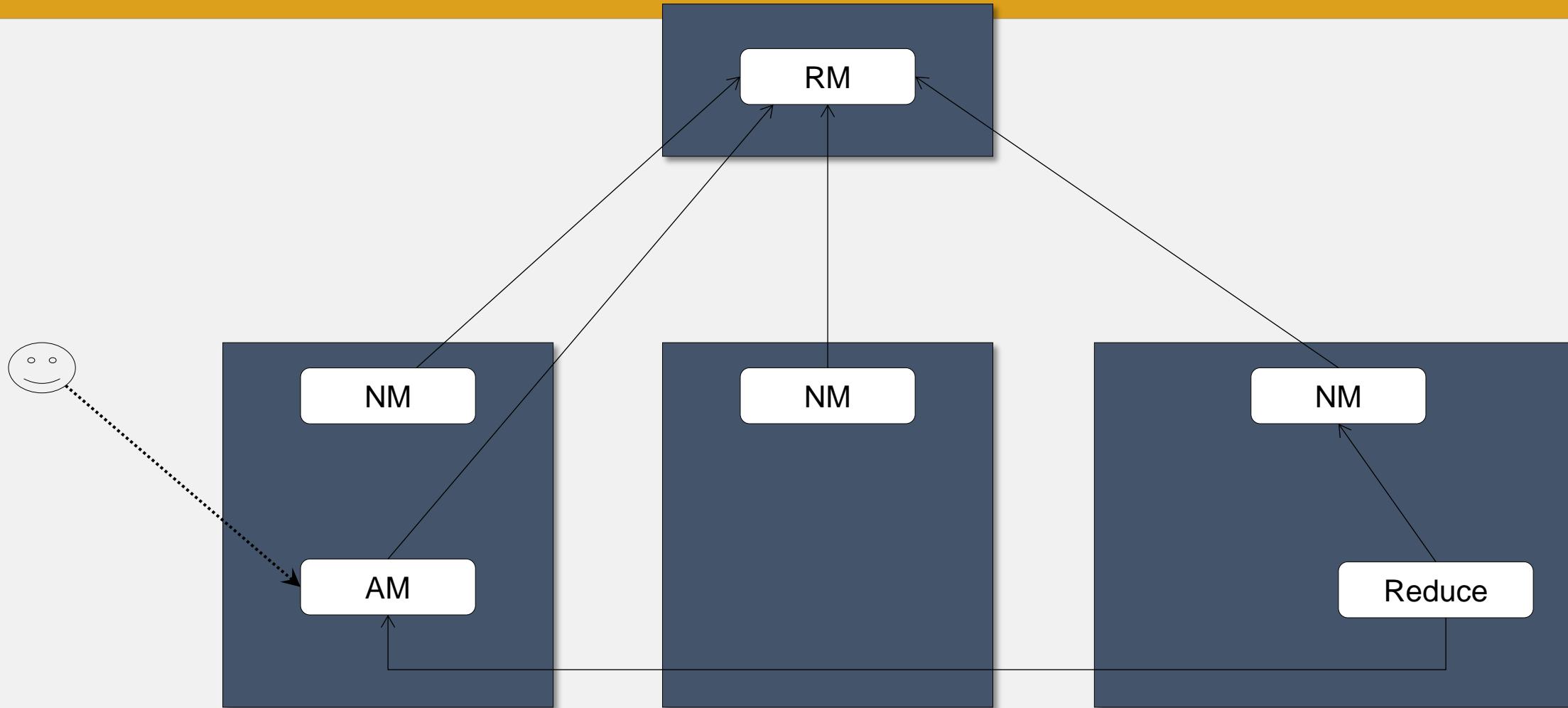
Map task 1 completes; AM requests container for reduce task and launches via NM

MapReduce on YARN



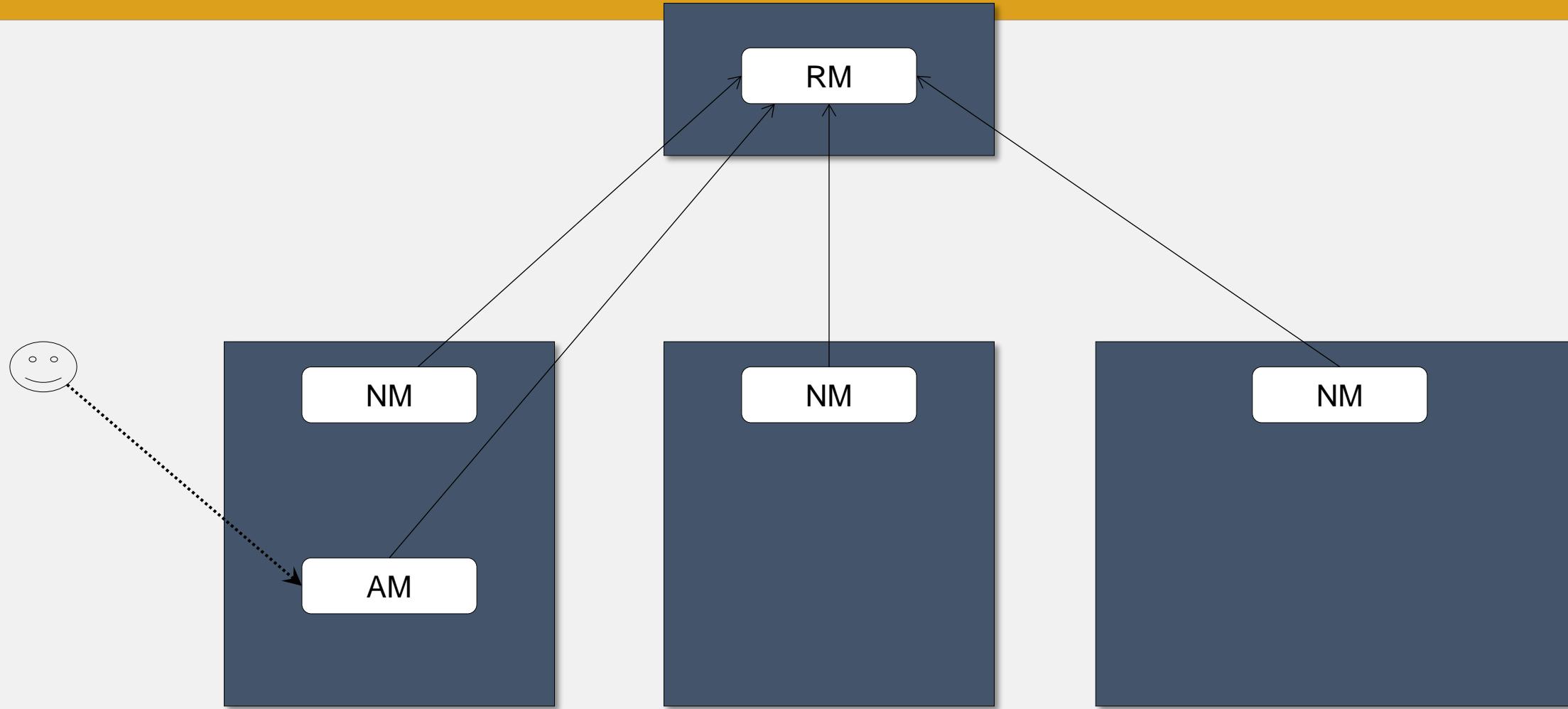
AM informs reducer of shuffle data location for map 1;
reduce task connects to NM and retrieves

MapReduce on YARN



Map 2 completes. AM informs reducer of shuffle data
location for map 2 and reducer retrieves

MapReduce on YARN



Reducer completes. AM commits final output, copies job history,
cleans staging area, and unregisters from RM before exiting



CLOUD COMPUTING APPLICATIONS

YARN: Node Manager

Roy Campbell & Reza Farivar

Node Manager Responsibilities

- Launch containers
- Localize files for containers
- Report container status to RM
- Kill containers and clean up local storage
- Monitor container resource usage and kill misbehaving containers
- Run health check and notify RM if unhealthy
- Log aggregation
- Web app for reporting node status and applications/containers
- Web services to avoid screen-scraping
- Runs auxiliary services for application frameworks as plugins
 - MapReduce ShuffleHandler serves map outputs to reducers

Node Manager Container Localization

- Download files/directories to local filesystem for the container
 - .jar file or executable for container
 - Supporting files, libraries
 - Known as the Distributed Cache in MapReduce
- Container launch context lists entries needing to be localized
- Localization requests have three privilege modes
 - Public: all users can access
 - Private: only the user can access
 - Application: only the application can access
- Downloaded entries are symlinked into the container working directory

Node Manager Log Aggregation

- NM collects logs from all containers for an application
- Collated into a file per node per application and uploaded to HDFS
- Benefits:
 - Allows long-term application log retention
 - Increases log availability
 - Easier to post-process logs via MapReduce jobs

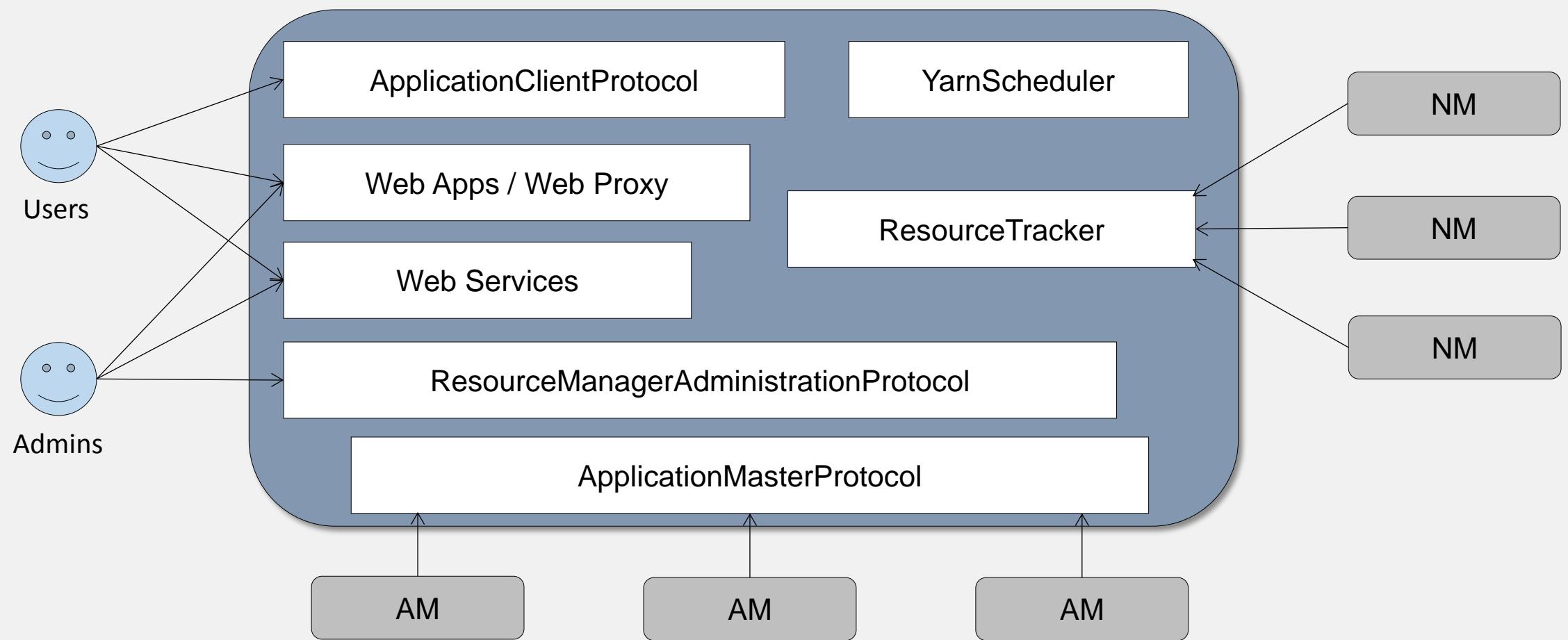


CLOUD COMPUTING APPLICATIONS

YARN: Resource Manager

Roy Campbell & Reza Farivar

ResourceManager Interfaces



Resource Manager Interfaces

- ApplicationClientProtocol
 - Submit an application
 - Kill an application
 - Query application, scheduler, cluster status
- Web Apps
 - UI for users to track status of cluster, scheduler, and applications

User-Facing Interfaces

- Web Proxy
 - Users provided with a proxy URL when application submitted
 - AM can update tracking URL to an application-specific UI
 - Proxy redirects users to tracking URL
- Web Services
 - Avoid programmatic screen-scraping of web apps

User-Facing Interfaces

- ResourceManagerAdministrationProtocol
 - Control access of users
 - Decommission or restrict nodes
 - Reconfigure scheduler

Node Manager-Facing Interface

- ResourceTracker
 - Node registration and liveliness monitoring
 - Process container status updates and notify scheduler

Application Master-Facing Interface

- ApplicationMasterProtocol
 - AM registration and liveness monitoring
 - Notify scheduler of container request updates from AMs
 - Notify AMs of granted containers and completed containers
 - Notify AMs of capacity remaining in their queue
 - Tracking URL management

Scheduling Interfaces

- YarnScheduler
 - Pluggable interface
 - FIFO Scheduler, Capacity Scheduler, and Fair Scheduler bundled with YARN
 - Active area of research in YARN



CLOUD COMPUTING APPLICATIONS

YARN: Application Master

Roy Campbell & Reza Farivar

Application Master Responsibilities

- Task scheduling and monitoring for an application instance
 - Only AM knows if a task has hung as RM and NM only see an opaque container

Application Master Responsibilities

- Heartbeats to RM
 - Indicates liveliness
 - Request new containers
 - Receive allocated containers

Application Master Responsibilities

- Provides optional tracking URL when registering with RM
 - Proxy URL provided by RM on app submission is an easy way to find app instance
 - URL can be updated when unregistering to point to a history server



CLOUD COMPUTING APPLICATIONS

Roy Campbell & Reza Farivar

Motivation for Spark



CLOUD COMPUTING APPLICATIONS

Roy Campbell & Reza Farivar

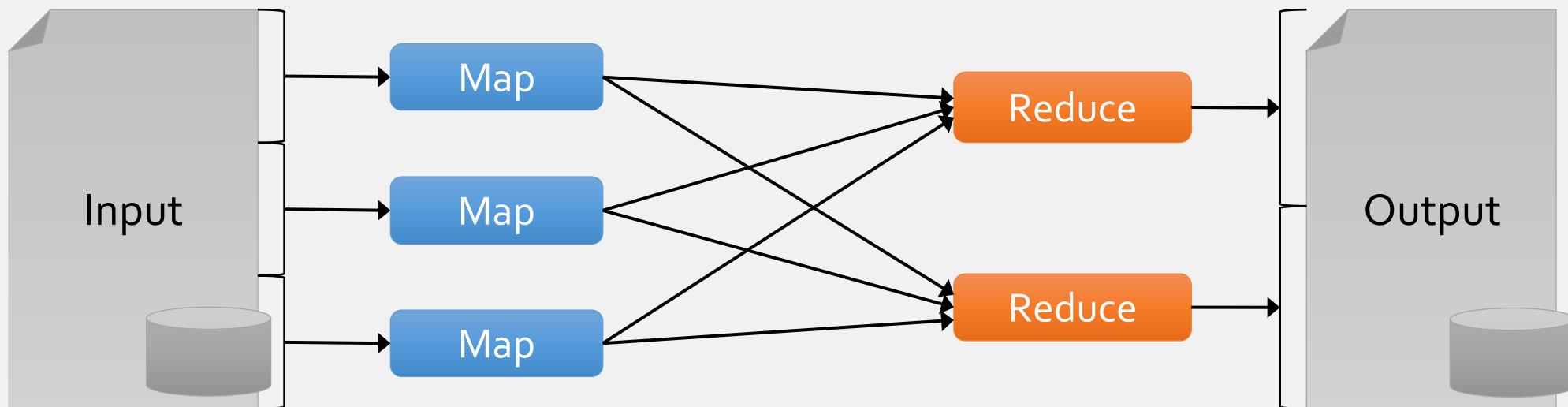
Apache Spark

Apache Spark

- Extend the MapReduce model to better support two common classes of analytics apps:
 - **Iterative** algorithms (machine learning, graphs)
 - **Interactive** data mining
- Enhance programmability:
 - Integrate into Scala programming language
 - Allow interactive use from Scala interpreter

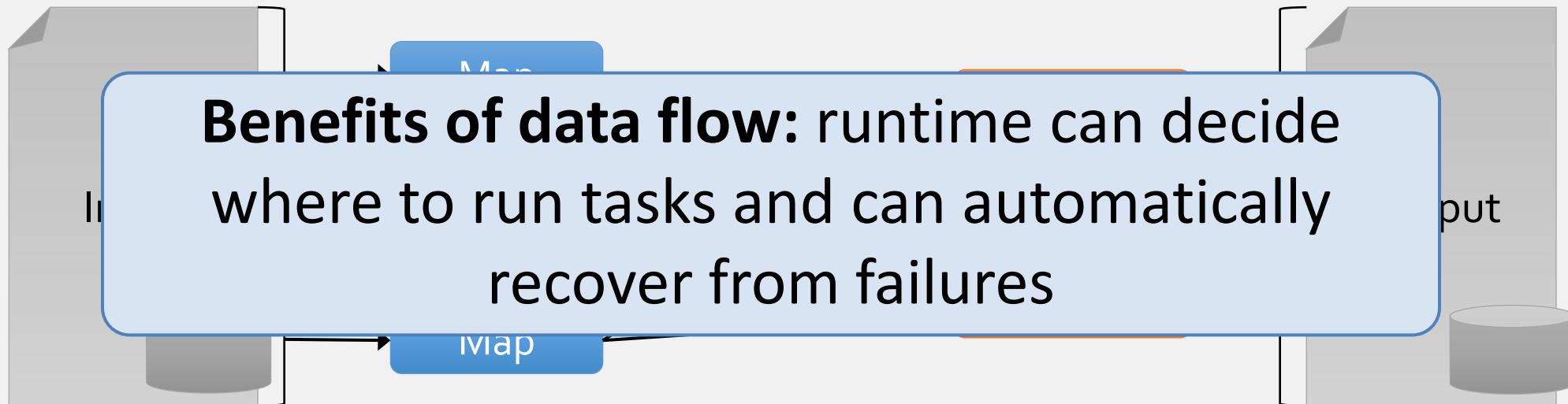
Motivation

Most current cluster programming models are based on *acyclic data flow* from stable storage to stable storage



Motivation

Most current cluster programming models are based on *acyclic data flow* from stable storage to stable storage



Motivation

- Acyclic data flow is inefficient for applications that repeatedly reuse a *working set* of data:
 - **Iterative** algorithms (machine learning, graphs)
 - **Interactive** data mining tools (R, Excel, Python)
- With current frameworks, apps reload data from stable storage on each query

Solution: Resilient Distributed Datasets (RDDs)

- Allow apps to keep working sets in memory for efficient reuse
- Retain the attractive properties of MapReduce
 - Fault tolerance, data locality, scalability
- Support a wide range of applications

Programming Model

- Resilient distributed datasets (RDDs)
 - Immutable, partitioned collections of objects
 - Created through parallel *transformations* (map, filter, groupBy, join, ...) on data in stable storage
 - Can be *cached* for efficient reuse
- Actions on RDDs
 - Count, reduce, collect, save, ...



CLOUD COMPUTING APPLICATIONS

Spark Example: Log Mining

Roy Campbell & Reza Farivar



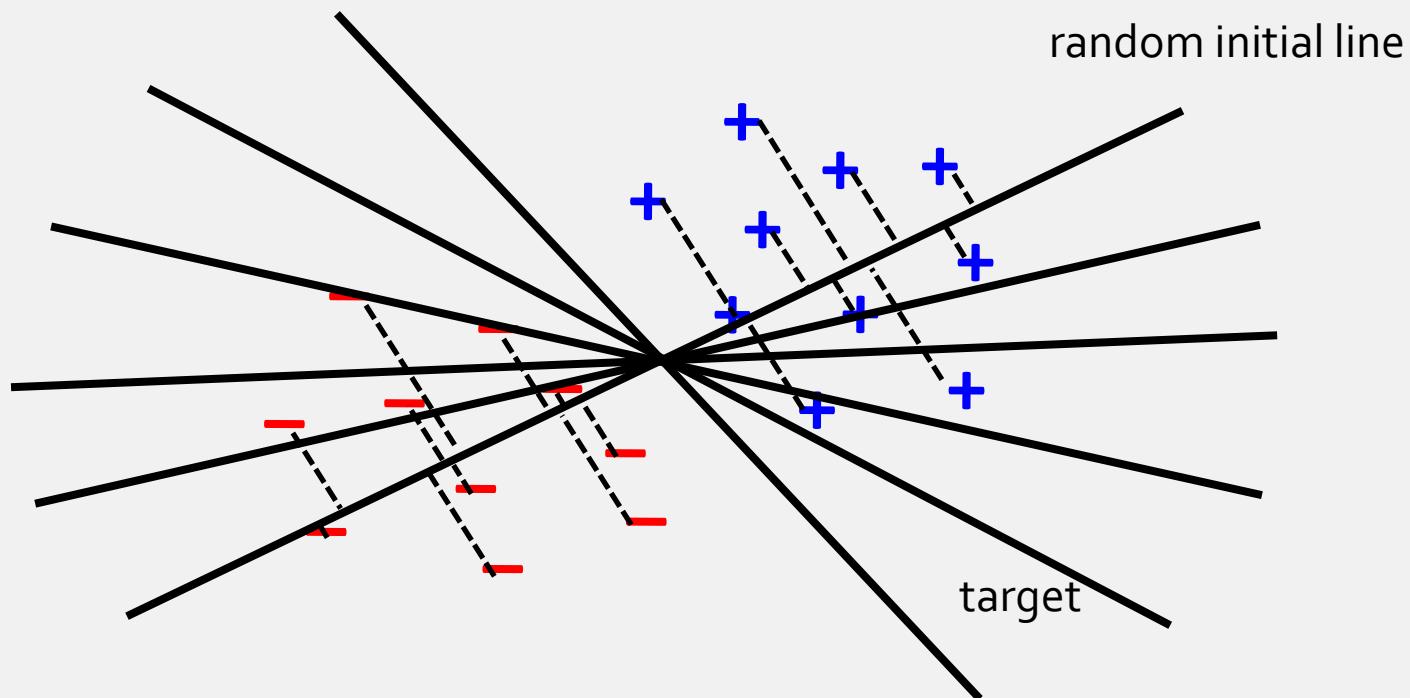
CLOUD COMPUTING APPLICATIONS

Spark Example: Logistic Regression

Roy Campbell & Reza Farivar

Example: Logistic Regression

Goal: find best line separating two sets of points



Example: Logistic Regression

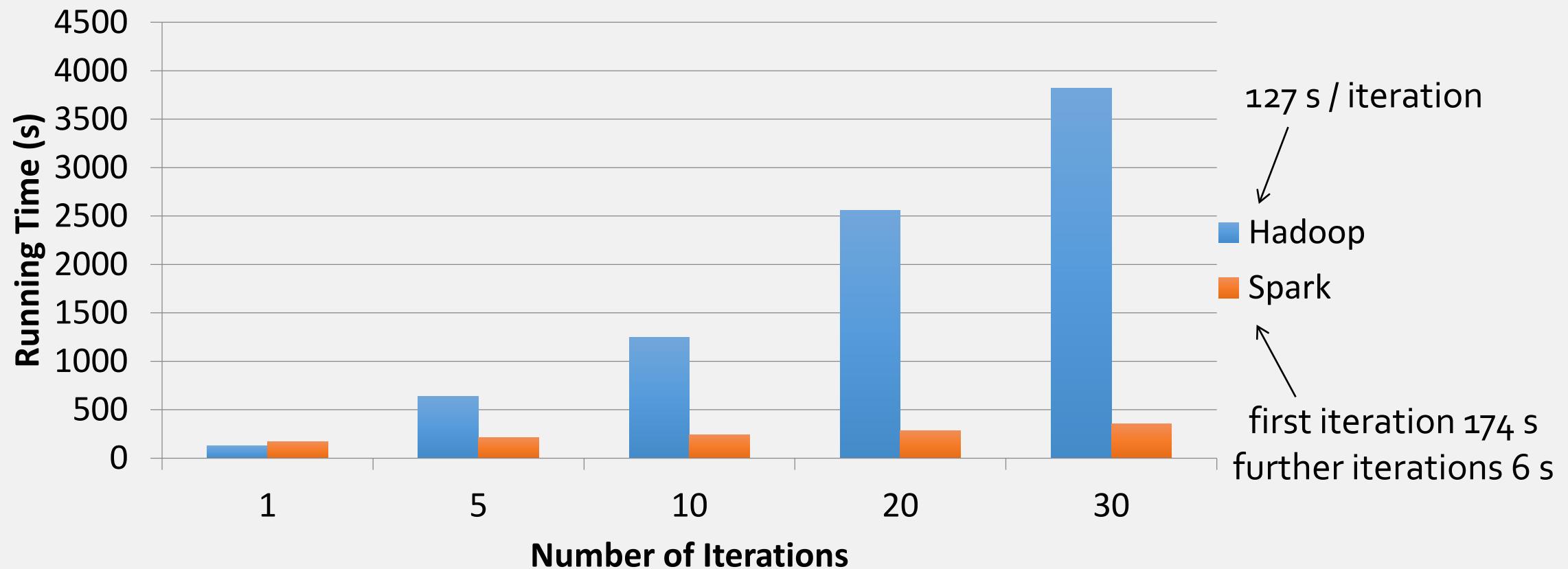
```
val data = spark.textFile(...).map(readPoint).cache()

var w = Vector.random(D)

for (i <- 1 to ITERATIONS) {
    val gradient = data.map(p =>
        (1 / (1 + exp(-p.y * (w dot p.x))) - 1) * p.y * p.x
    ).reduce(_ + _)
    w -= gradient
}

println("Final w: " + w)
```

Logistic Regression Performance





CLOUD COMPUTING APPLICATIONS

Roy Campbell & Reza Farivar

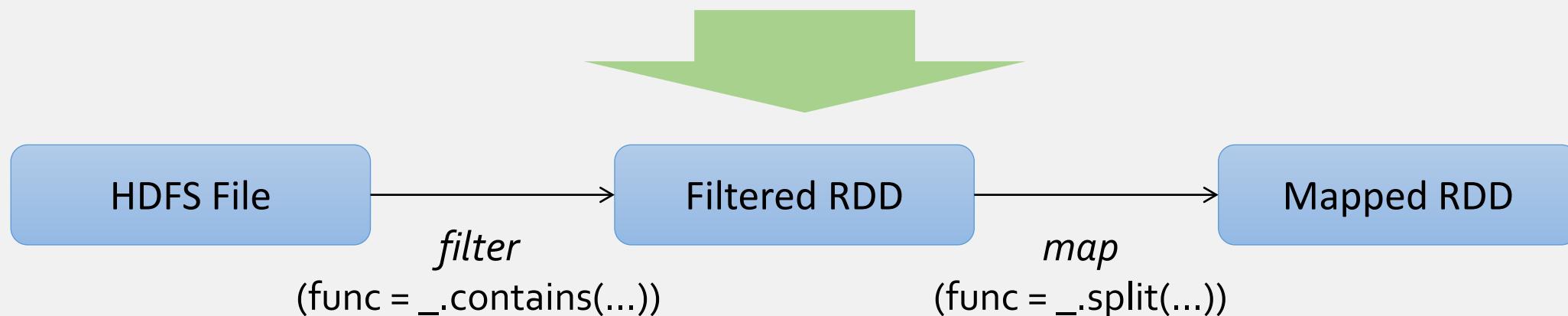
RDD Fault Tolerance

RDD Fault Tolerance

RDDs maintain *lineage* information that can be used to reconstruct lost partitions

Ex:

```
messages = textFile(...).filter(_.startsWith("ERROR"))
           .map(_.split('\t')(2))
```





CLOUD COMPUTING APPLICATIONS

Roy Campbell & Reza Farivar

Interactive Spark

Interactive Spark

- Modified Scala interpreter to allow Spark to be used interactively from the command line
- Required two changes:
 - Modified wrapper code generation so that each line typed has references to objects for its dependencies
 - Distribute generated classes over the network

Frameworks Built on Spark

- Pregel on Spark (GraphX)
 - Google message passing model for graph computation
- Hive on Spark (SparkSQL)
 - Compatible with Apache Hive
 - ML operators in Scala
- Mllib
 - Scalable Machine Learning Library



CLOUD COMPUTING APPLICATIONS

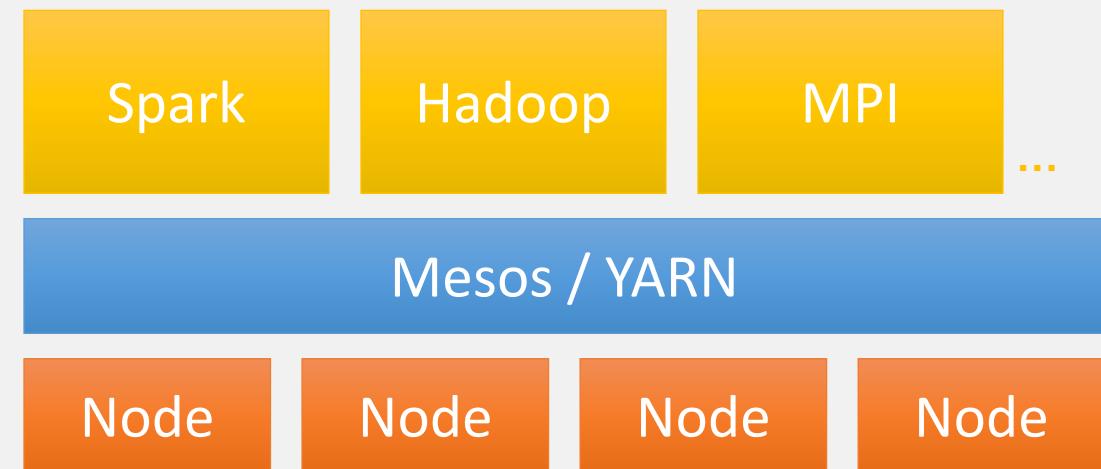
Roy Campbell & Reza Farivar

Spark Implementation

Implementation

Runs on Apache Mesos or YARN
to share resources with Hadoop &
other apps

Can read from any Hadoop input
source (e.g. HDFS)



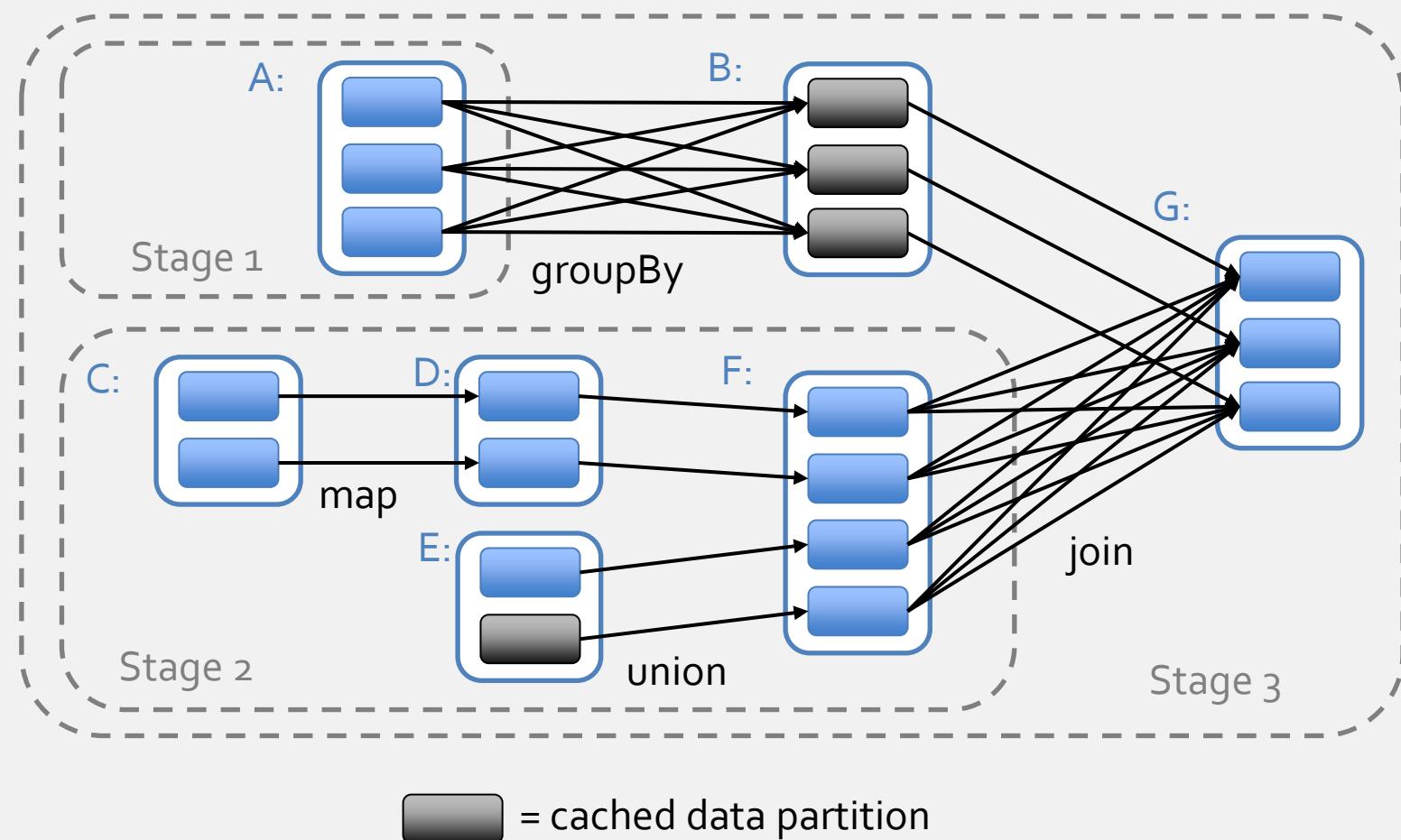
Spark Scheduler

Dryad-like DAGs

Pipelines functions
within a stage

Cache-aware work
reuse & locality

Partitioning-aware
to avoid shuffles





CLOUD COMPUTING APPLICATIONS

HDFS Introduction

Roy Campbell & Reza Farivar

Large-Scale Data Processing

- Many tasks
 - Processing lots of data in parallel to produce lots of other data
- Large-scale data processing
 - Want to use 1000s of CPUs
 - But don't want overhead of **managing** storage
 - Storage devices fail 1.7% year 1 – 8.6% year 3 (Google, 2007)
 - 10,000 nodes, 7 disks per node, year 1, 1190 failures/yr or 3.3 failures/day
- MapReduce provides:
 - User-defined functions
 - Automatic parallelization and distribution
 - Fault tolerance
 - I/O scheduling
 - Status and monitoring

HDFS

- Synergistic with Hadoop
 - Apache Project inspired by Google Map/Reduce and GFS
- Massive throughput
- Throughput scales with attached HDs
- Have seen VERY LARGE production clusters
 - Facebook, Yahoo... Nebraska
- Doesn't even pretend to be POSIX compliant
- Optimized for reads, sequential writes and appends

References

- The Google File System
 - SOSP 2003
- The Hadoop Distributed File System
 - 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)

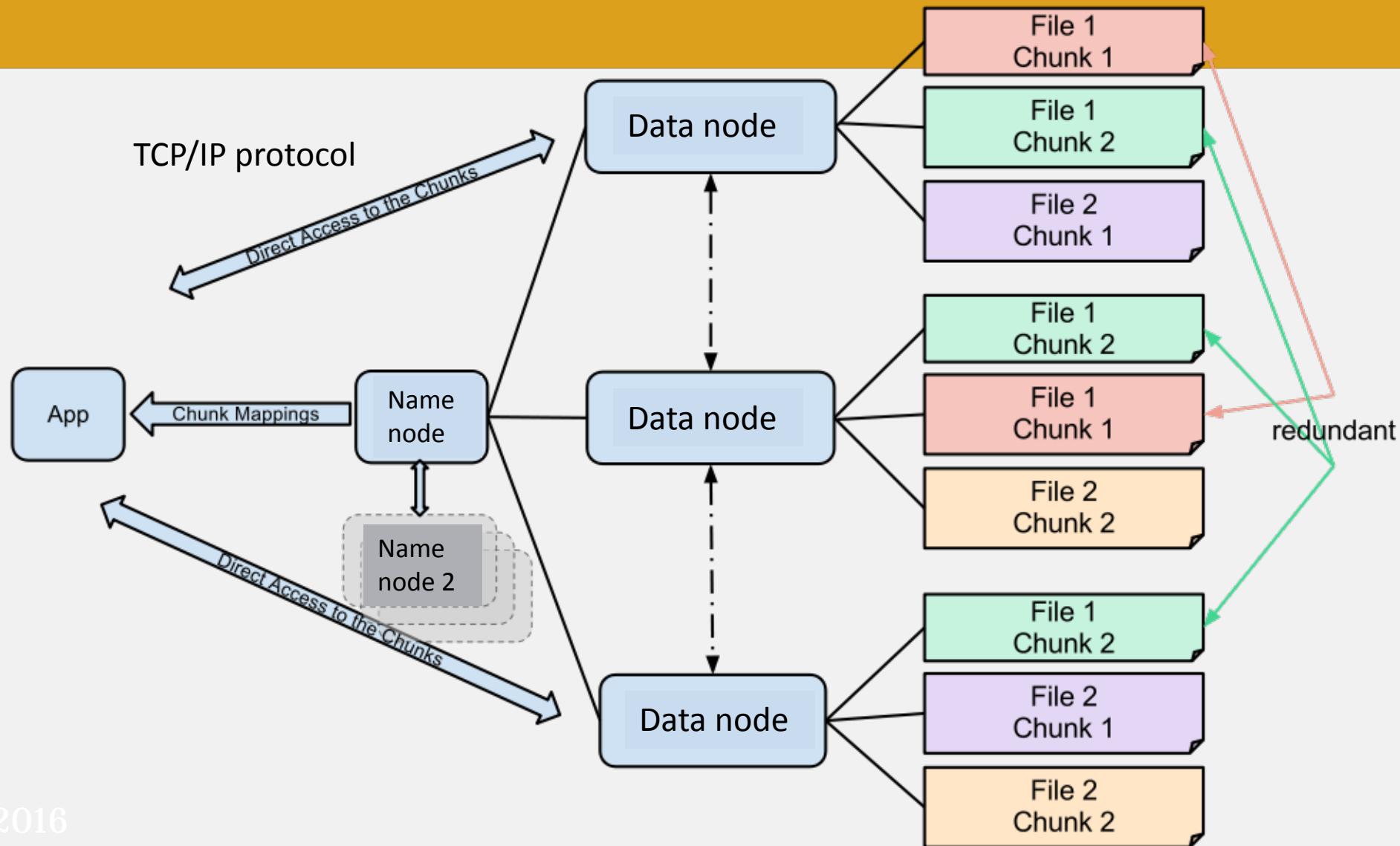
Stable Storage

- If nodes failure is the norm and not the exception, how can we store data persistently?
- **Answer:** Distributed File System replicates files
 - Provides global file namespace
 - Google GFS, Hadoop HDFS
 - Kosmix KFS
- Typical usage pattern
 - Huge files (100s of GB to TB)
 - Data is rarely updated in place
 - Multiple copies improves availability
 - Reads and appends are common

Distributed File System

- Datanode Servers
 - A file is split into contiguous chunks
 - Typically each chunk is 16-64MB
 - Each chunk replicated (usually 2x or 3x)
 - Sends heartbeat and BlockReport to namenode
- Replicas are placed: one on a node in a local rack, one on a different node in the local rack, and one on a node in a different rack

HDFS Architecture



Distributed File System

- Master node
 - a.k.a. NameNode in HDFS
 - Stores metadata
 - Might be replicated
- Client library for file access
 - Talks to master to find data node chunk
 - Connects directly to data node servers to access data

Replication Pipelining

- When the client receives response from NameNode, it flushes its block in small pieces (4K) to the first replica, that in turn copies it to the next replica and so on
- Thus data is pipelined from data node to the next

Staging

- A client request to create a file does not reach NameNode immediately
- HDFS client caches the data into a temporary file. When the data reaches an HDFS block size, the client contacts the NameNode
- NameNode inserts the filename into its hierarchy and allocates a data block for it
- The NameNode responds to the client with the identity of the data node and the destination of the replicas (data nodes) for the block
- Then the client flushes it from its local memory

Application Programming Interface

- HDFS provides Java API for application to use
- Python access is also used in many applications
- A C language wrapper for Java API is also available
- A command line interface provided in Hadoop
- The syntax of the commands is similar to bash
- Example: to create a directory /foodir
- /bin/hadoop dfs –mkdir /foodir
- An HTTP browser can be used to browse the files of an HDFS instance



CLOUD COMPUTING APPLICATIONS

YARN, MESOS

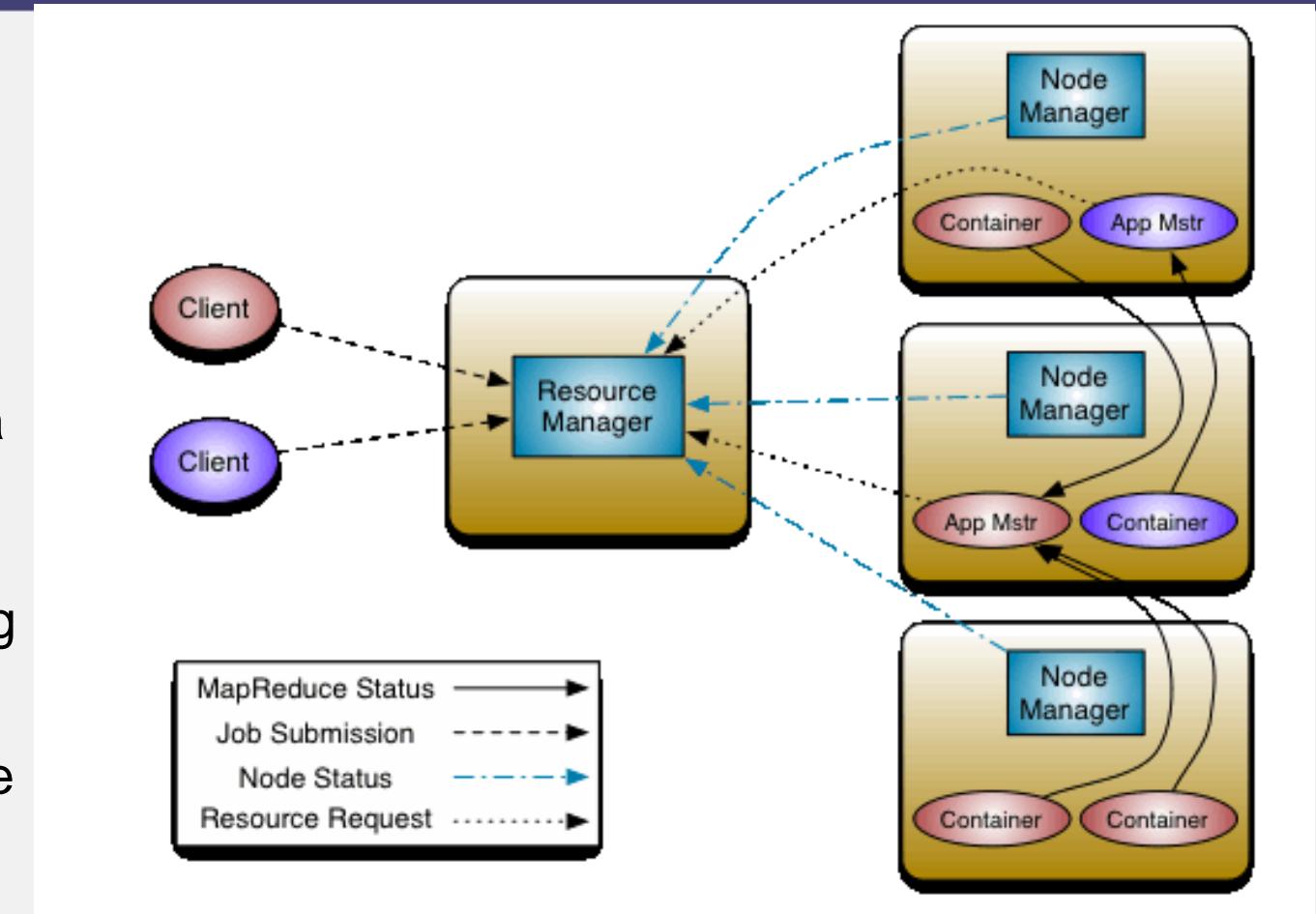
Roy Campbell & Reza Farivar

Overview

- Mesos was built to be a scalable global resource manager for the entire data center (Berkeley 2007)
 - Hadoop, MPI,
- YARN was created out of the necessity to scale Hadoop.

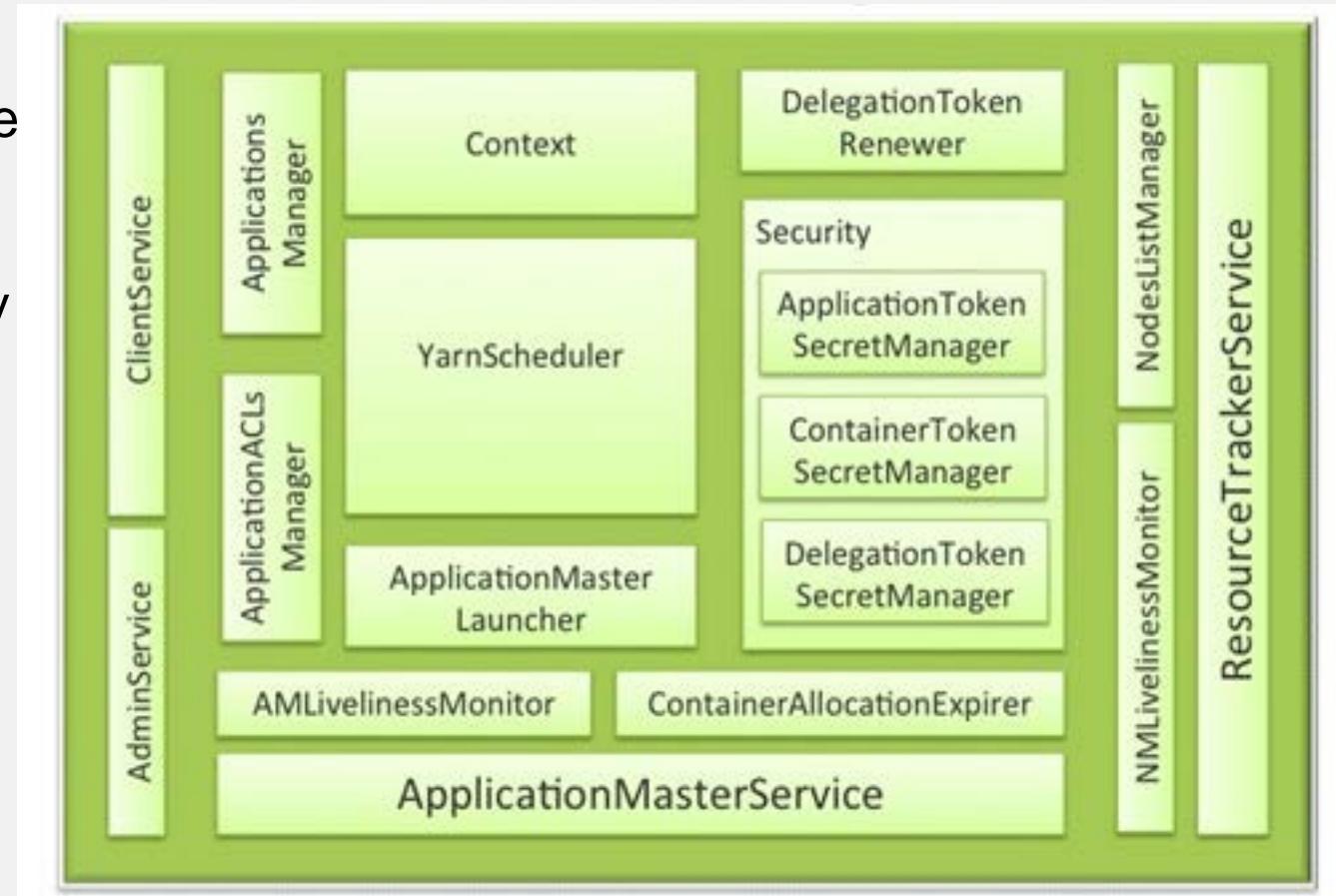
YARN ResourceManager

- Scheduler
 - pluggable policy partitions cluster resources among the various queues, and applications
 - CapacityScheduler and the FairScheduler
- NodeManagers take instructions from the ResourceManager and manage resources available on a single node.
- ApplicationsManager negotiates resources with the resourcemanager and for working with the nodemanagers to start the containers.
- Compatible API with Map/Reduce



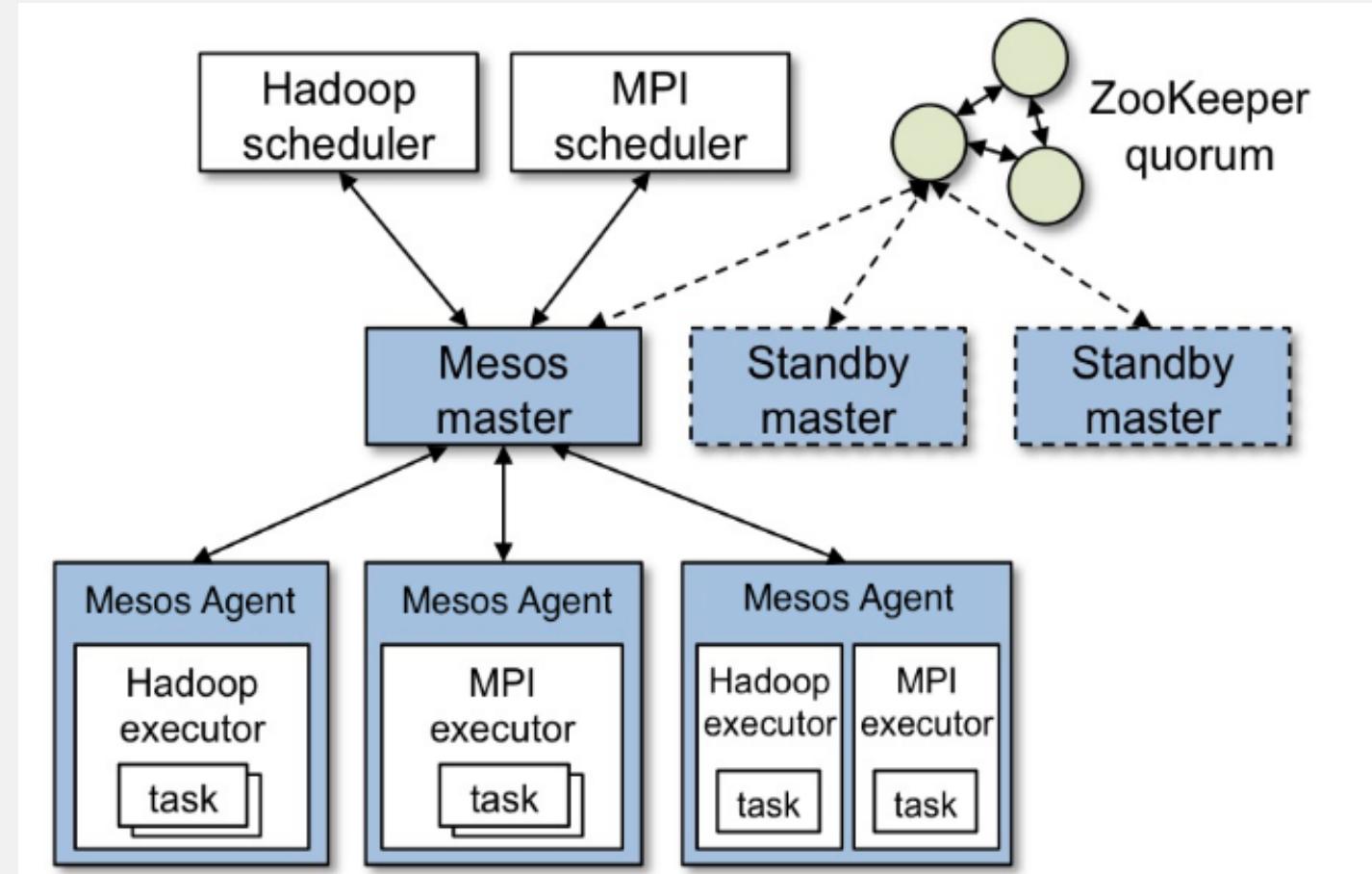
Insides of Yarn

- ResourceTrackerService
 - responds to RPCs from all the nodes.
 - Registration of new nodes, rejection of requests from any invalid/decommissioned nodes,
 - obtaining node-heartbeats and forward them over to the YarnScheduler.



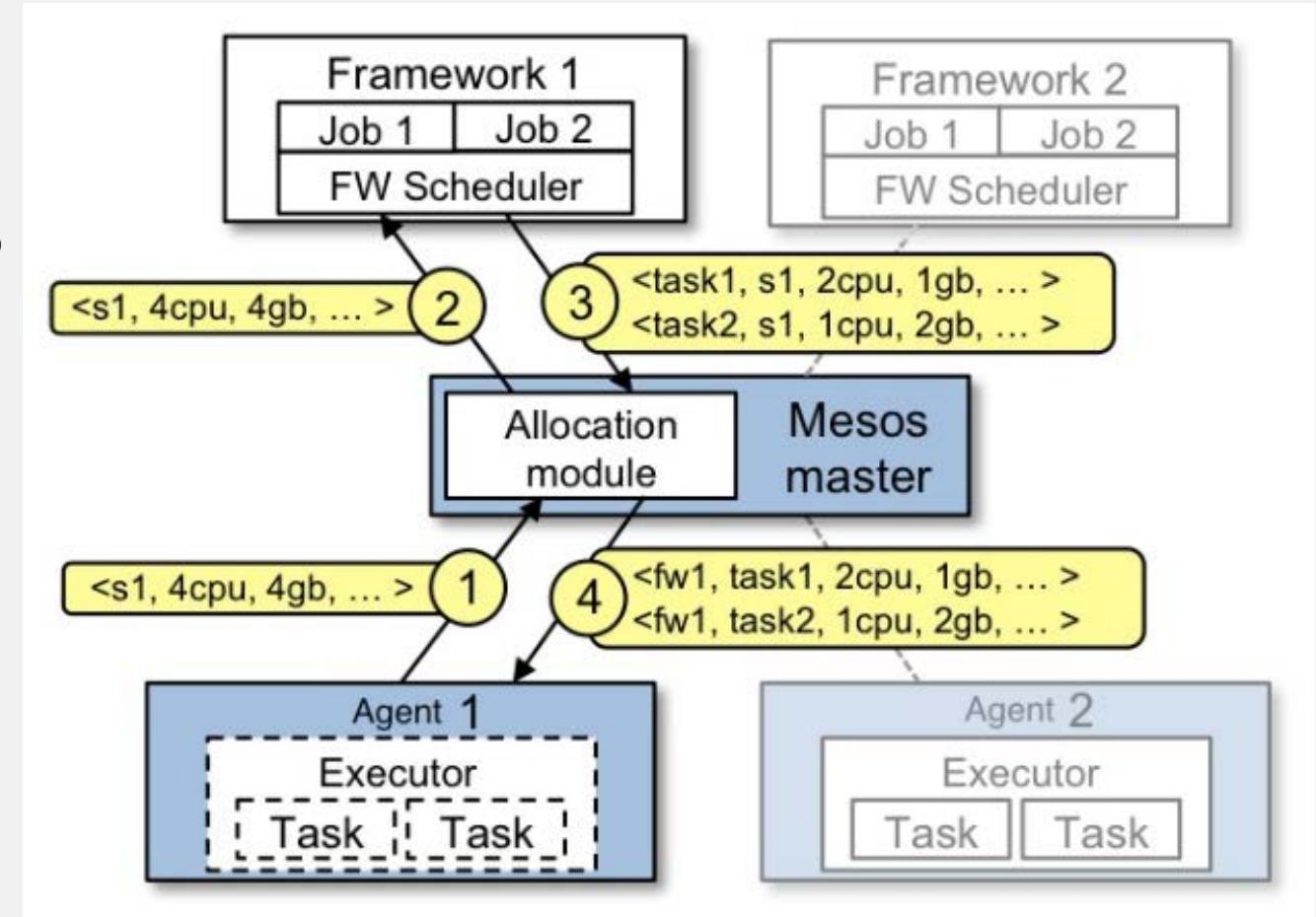
Mesos Architecture

- The master decides *how many* resources to offer to each framework according to a given organizational policy, such as fair sharing or strict priority.



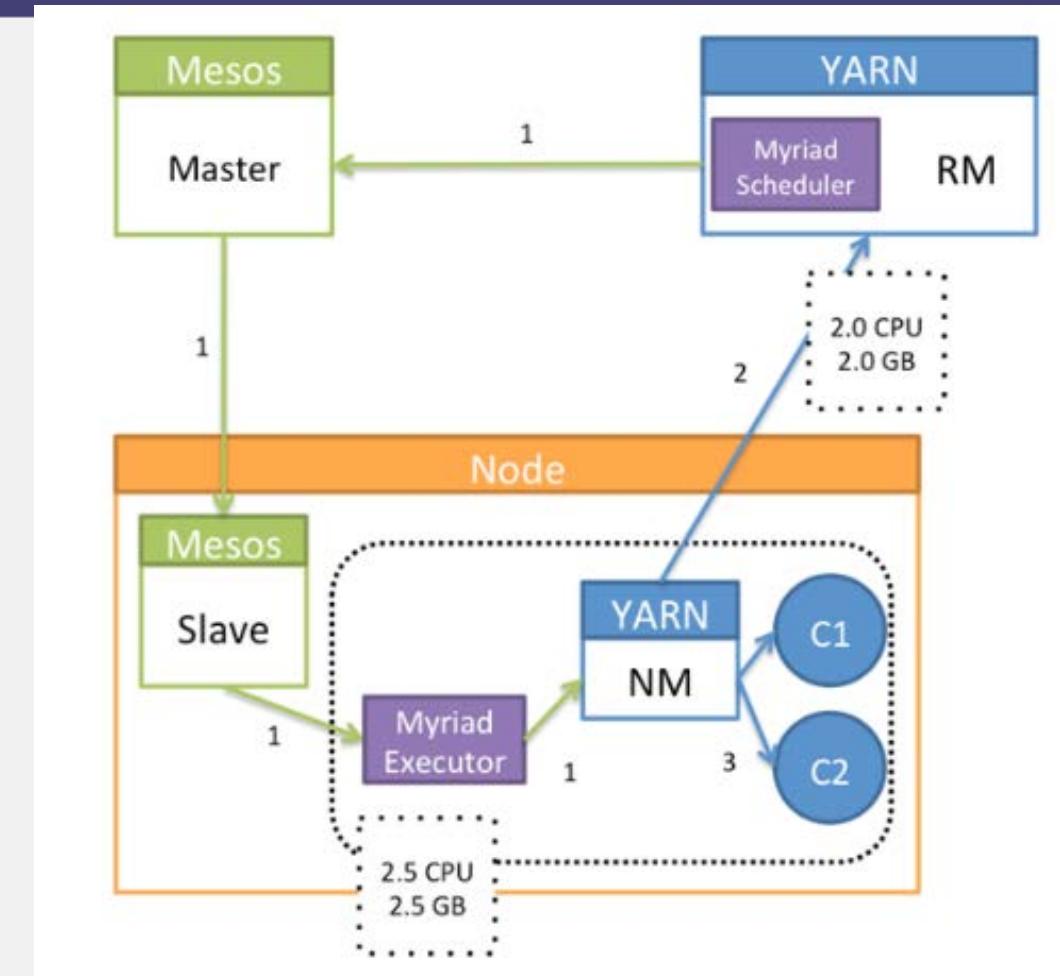
Mesos Resource Offer

1. Agent 1 reports to the master that it has 4 CPUs and 4 GB of memory free. The master then invokes the allocation policy module, which tells it that framework 1 should be offered all available resources.
2. The master sends a resource offer describing what is available on agent 1 to framework 1.
3. The framework's scheduler replies to the master with information about two tasks to run on the agent, using <2 CPUs, 1 GB RAM> for the first task, and <1 CPUs, 2 GB RAM> for the second task.
4. Finally, the master sends the tasks to the agent, which allocates appropriate resources to the framework's executor, which in turn launches the two tasks (depicted with dotted-line borders in the figure). Because 1 CPU and 1 GB of RAM are still unallocated, the allocation module may now offer them to framework 2.



Mesos and YARN Convergence

- **Project Myriad**
- Mesos framework and a YARN scheduler that enables Mesos to manage YARN resource requests.





CLOUD COMPUTING APPLICATIONS

Roy Campbell & Reza Farivar

Introduction to Distros

Distribution of Data Analytics Tools

- Hortonworks
- Cloudera
- MapR



CLOUD COMPUTING APPLICATIONS

Roy Campbell & Reza Farivar

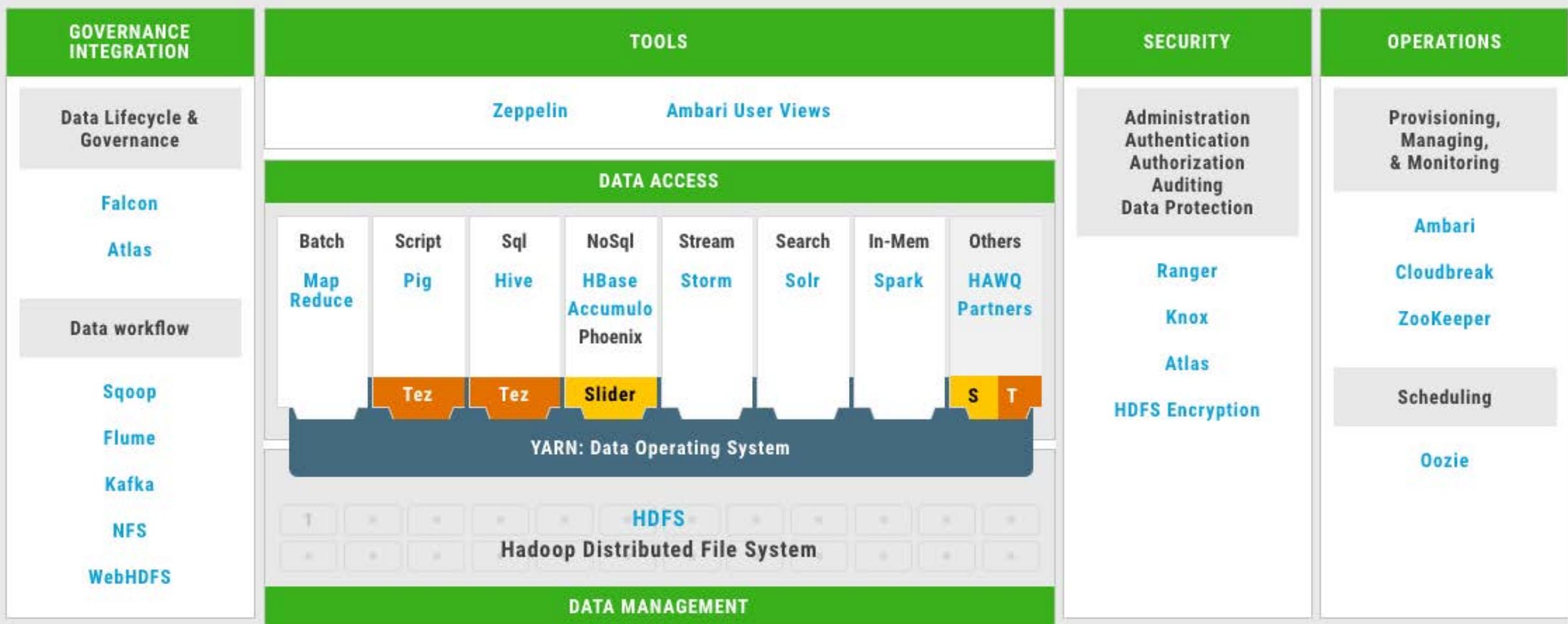
Hortonworks

Pieces of the Puzzle

Connected Data Strategy

- HDP: Apache Hadoop® is an open source framework for distributed storage and processing of large sets of data on commodity hardware. Hadoop enables businesses to quickly gain insight from massive amounts of structured and unstructured data.
- HDF enables real-time data collection, curation, analysis and delivery of data to and from any device, source or system, either on-premise and in the cloud

Hortonworks Data Platform (HDP)



HDP Tools

- Apache Zeppelin: open web-based notebook that enables interactive data analytics
- Apache Ambari: source management platform for provisioning, managing, monitoring and securing Apache Hadoop clusters.

Apache Zeppelin (Python, Spark, Scala...)

Zeppelin Notebook - Interpreter

Connected

```
import sys,process..  
// sc is an existing SparkContext,  
val sc = new org.apache.spark.sql.SQLContext(sc)  
val health_dataset = sc.textFile("/Users/mshrawe/Downloads/health_data_expenses.csv")  
case class Health(year: String, state: String, category: String, funding_src1: String, funding_src2: String, spending: Integer)  
val health = health_dataset.map(_.split(",")).map(  
    h => Health(  
        h(0),  
        h(1),  
        h(2),  
        h(3),  
        h(4),  
        h(5).toInt  
    )  
)  
health.registerTempTable("health_table")  
  
import sys,process..  
sqlContext: org.apache.spark.sql.SQLContext = org.apache.spark.sql.SQLContext@87e76cc70  
health_dataset: org.apache.spark.rdd.RDD[String] = /Users/mshrawe/Downloads/health_data_expenses.csv MapPartitionsRDD[546] at textFile at <console>:182  
defined class Health  
health: org.apache.spark.sql.DataFrame = [year: string, state: string, category: string, funding_src1: string, funding_src2: string, spending: int]  
Task 4 seconds.
```

Sql.
select state,sum(spending)/1000 SpendinginBillions from health_table group by state order by SpendinginBillions desc

SETTINGS +



Sql.
select year,sum(spending)/1000 SpendinginBillions from health_table group by year order by SpendinginBillions

SETTINGS +



Sql.
select category,sum(spending)/1000 SpendinginBillions from health_table group by category order by SpendinginBillions desc

SETTINGS +

category	SpendinginBillions
Public hospitals	445.845
Medical services	272.507
Private hospitals	121.022
Benefit-paid pharmaceuticals	104.221
Dental services	90.766
Community health	75.765
Capital expenditure	72.698
All other medications	70.506
Other health practitioners	51.382

Apache Zeppelin (Python, Spark, Scala...)

User View	Description
Tez	The Tez View helps you understand and optimize your cluster resource usage. Using the view, you can optimize and accelerate individual SQL queries or Pig jobs to get the best performance in a multi-tenant Hadoop environment.
Hive	Hive View allows the user to write & execute SQL queries on the cluster. It shows the history of all Hive queries executed on the cluster whether run from Hive view or another source such as JDBC/ODBC or CLI. It also provides graphical view of the query execution plan. This helps the user debug the query for correctness and for tuning the performance. It integrates Tez View that allows the user to debug any Tez job, including monitoring the progress of a job (whether from Hive or Pig) while it is running. This view contribution can be found here .
Pig	Pig View is similar to the Hive View. It allows writing and running a Pig script. It has support for saving scripts, and loading and using existing UDFs in scripts. This view contribution can be found here .
Capacity Scheduler	Capacity Scheduler View helps a Hadoop operator setup YARN workload management easily to enable multi-tenant and multi-workload processing. This view provisions cluster resources by creating and managing YARN queues. This view contribution can be found here .
Files	Files View allows the user to manage, browse and upload files and folders in HDFS. This view contribution can be found here .

Data Access

- **YARN:** Data Operating System
 - **MapReduce.** Batch application framework for structured and unstructured data
 - **Pig.** Script Extract-transform-load (ETL) data pipelines, Research on raw data, and Iterative data processing.
 - SQL. **Hive** interactive SQL queries over petabytes of data in Hadoop
 - NoSql (**Hbase Accumulo**) non-relational (NoSQL) database on top of HDFS
 - **Storm** (Stream) distributed real-time --large volumes of high-velocity data.
 - **Solr** (Search) -- full-text search and near real-time indexing.
 - **Spark** (In-Mem)
- Data Management
 - Hadoop Distributed File System (**HDFS**)

MapReduce

Benefit	Description
Simplicity	Developers can write applications in their language of choice, such as Java, C++ or Python, and MapReduce jobs are easy to run
Scalability	MapReduce can process petabytes of data, stored in HDFS on one cluster
Speed	Parallel processing means that MapReduce can take problems that used to take days to solve and solve them in hours or minutes
Recovery	MapReduce takes care of failures. If a machine with one copy of the data is unavailable, another machine has a copy of the same key/value pair, which can be used to solve the same sub-task. The JobTracker keeps track of it all.
Minimal data motion	MapReduce moves compute processes to the data on HDFS and not the other way around. Processing tasks can occur on the physical node where the data resides. This significantly reduces the network I/O patterns and contributes to Hadoop's processing speed.

Pig

Characteristic	Benefit
Extensible	Pig users can create custom functions to meet their particular processing requirements
Easily programmed	Complex tasks involving interrelated data transformations can be simplified and encoded as data flow sequences. Pig programs accomplish huge tasks, but they are easy to write and maintain.
Self-optimizing	Because the system automatically optimizes execution of Pig jobs, the user can focus on semantics.

Hive

Feature	Description
Familiar	Query data with a SQL-based language
Fast	Interactive response times, even over huge datasets
Scalable and Extensible	As data variety and volume grows, more commodity machines can be added, without a corresponding reduction in performance
Compatible	Works with traditional data integration and data analytics tools.

Nosql HBase

Characteristic	Benefit
Fault tolerant	<ul style="list-style-type: none">• Replication across the data center• Atomic and strongly consistent row-level operations• High availability through automatic failover• Automatic sharding and load balancings of tables
Fast	<ul style="list-style-type: none">• Near real time lookups• In-memory caching via block cache and bloom filters• Server side processing via filters and co-processors
Usable	<ul style="list-style-type: none">• Data model accommodates wide range of use cases• Metrics exports via File and Ganglia plugins• Easy Java API as well as Thrift and REST gateway APIs

Nosql Accumulo

Feature	Benefit
Table design and configuration	<ul style="list-style-type: none">Includes cell tables for cell-level access controlLarge rows need not fit into memory
Integrity and availability	<ul style="list-style-type: none">Master fail-over with ZooKeeper locksWrite-ahead logs for recoveryScalable master metadata storeFault tolerant executor (FATE)
Performance	<ul style="list-style-type: none">Relative encoding to compress similar consecutive keysSpeed long scans with parallel server threadsCache recently scanned data
Data Management	<ul style="list-style-type: none">Group columns within a single fileAutomatic tablet splitting and rebalancingMerge tablets and clone tables

Storm

Feature	Description
Fast	– benchmarked as processing one million 100 byte messages per second per node
Scalable	– with parallel calculations that run across a cluster of machines
Fault-tolerant	– when workers die, Storm will automatically restart them. If a node dies, the worker will be restarted on another node.
Reliable	– Storm guarantees that each unit of data (tuple) will be processed at least once or exactly once. Messages are only replayed when there are failures.
Easy to operate	– standard configurations are suitable for production on day one. Once deployed, Storm is easy to operate.

Solr

Feature	Description
Advanced full-text search	
Near real-time indexing	
Standards-based open interfaces like XML, JSON and HTTP	
Comprehensive HTML administration interfaces	
Server statistics exposed over JMX for monitoring	
Linearly scalable, auto index replication, auto failover and recovery	
Flexible and adaptable, with XML configuration	

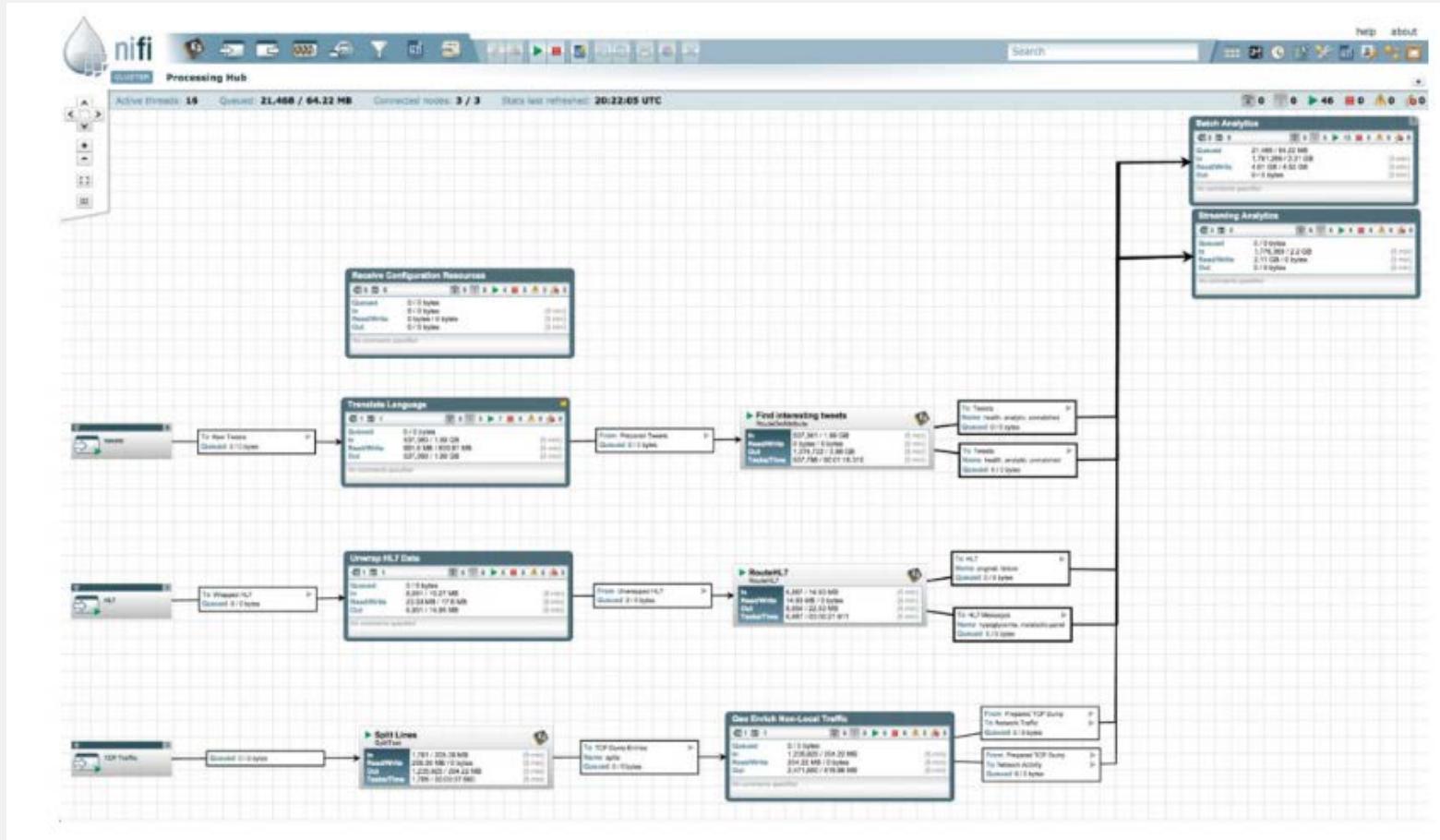
Spark

- | Feature | Description |
|---|-------------|
| <ul style="list-style-type: none">• In memory Spark Core<ul style="list-style-type: none">• Programming Scala, Java, Python, R, APIs• Libraries<ul style="list-style-type: none">• Mlib• Spark SQL• Spark Streaming• Graphx• ML Pipeline for data science workflow | |

Hortonworks Data Flow (HDF)

- Apache NiFi, Kafka and Storm provide real-time dataflow management and streaming analytics.
- HDF enables real-time data collection, curation, analysis and delivery of data to and from any device, source or system, either on-premise and in the cloud.
- Tools
 - KAFKA: fast, scalable, durable, and fault-tolerant publish-subscribe messaging
 - NIFI, integrated data logistics and simple event processing
 - STORM

NiFi



NiFi

NiFi Term	FBP Term	Description
FlowFile	Information Packet	A FlowFile represents each object moving through the system and for each one, NiFi keeps track of a map of key/value pair attribute strings and its associated content of zero or more bytes.
FlowFile Processor	Black Box	Processors actually perform the work. In [eip] terms a processor is doing some combination of data Routing, Transformation, or Mediation between systems. Processors have access to attributes of a given FlowFile and its content stream. Processors can operate on zero or more FlowFiles in a given unit of work and either commit that work or rollback.
Connection	Bounded Buffer	Connections provide the actual linkage between processors. These act as queues and allow various processes to interact at differing rates. These queues then can be prioritized dynamically and can have upper bounds on load, which enable back pressure.
Flow Controller	Scheduler	The Flow Controller maintains the knowledge of how processes actually connect and manages the threads and allocations thereof which all processes use. The Flow Controller acts as the broker facilitating the exchange of FlowFiles between processors.
Process Group	Subnet	A Process Group is a specific set of processes and their connections, which can receive data via input ports and send data out via output ports. In this manner process groups allow creation of entirely new components simply by composition of other components.

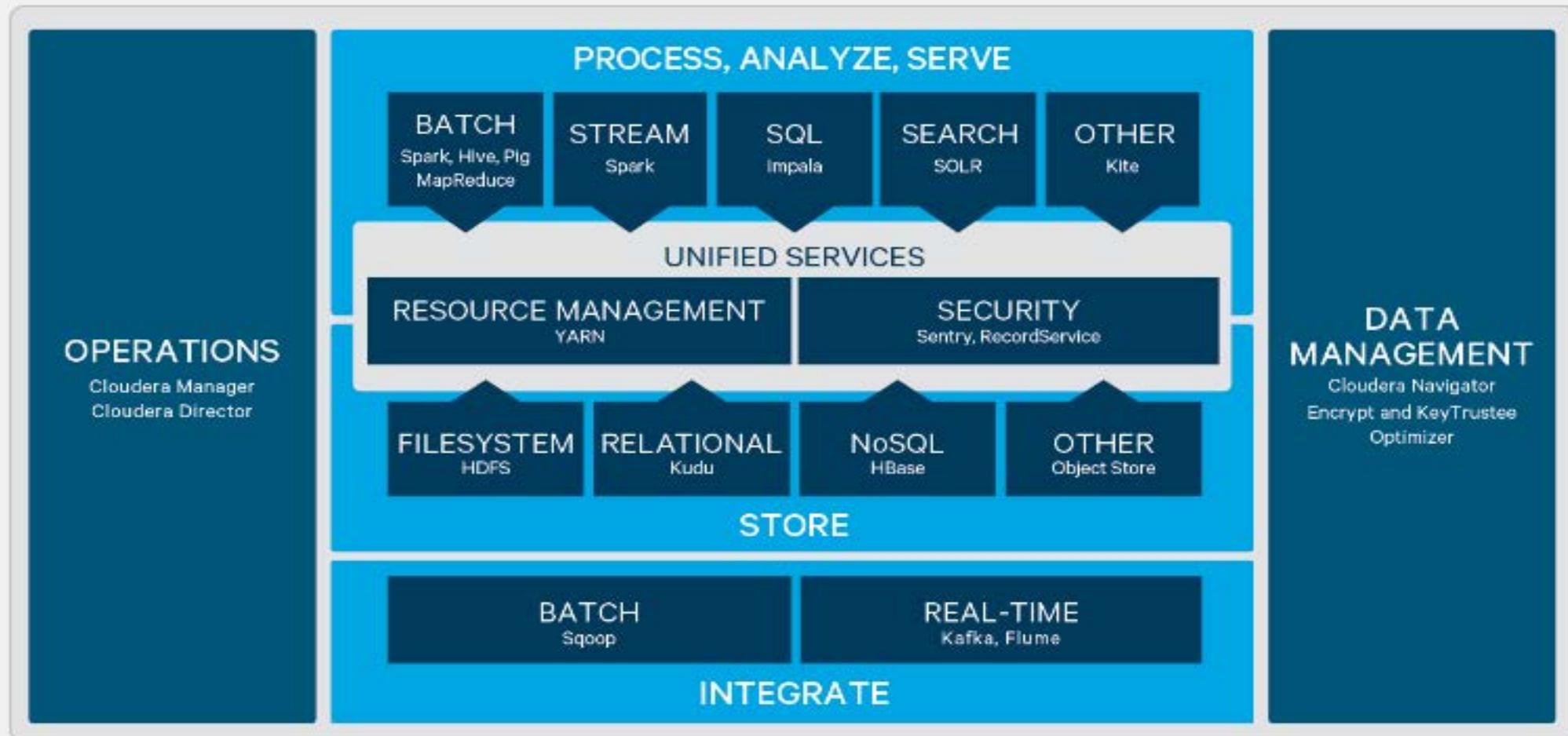


CLOUD COMPUTING APPLICATIONS

Cloudera CDH

Roy Campbell & Reza Farivar

Cloudera Enterprise Data Hub (EDH)





CLOUD COMPUTING APPLICATIONS

MapR Distro

Roy Campbell & Reza Farivar

Platform for Big Data

- MapReduce (Hadoop written in C/C++)
- NFS
- Interactive SQL (Drill, Hive Spark SQL, Impala)
- MapR-DB
- Search (Apache Solr)
- Stream Processing (MapR Streams)

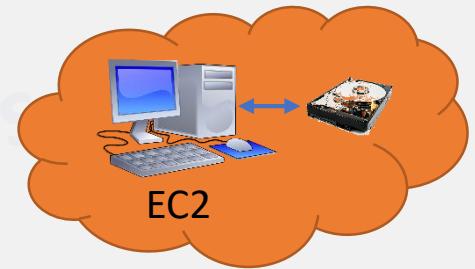


CLOUD COMPUTING APPLICATIONS

Cloud Storage: Block Storage
Prof. Reza Farivar

Cloud Storage Category 1: Block Storage – Instance Stores

- The physical machine running the virtual machine is physically connected to the storage device
 - virtual devices whose underlying hardware is physically attached to the host computer for the instance
 - Data transfer is limited by SATA / NVMe bandwidth
 - I3en.metal: 8 x 7,500 GB (60 TB)
- Since the machine may be rented to someone else in the next minute, the data stored on the drive does not persist, it's **ephemeral**.
- Example:
 - Amazon AWS: Instance Store
 - Google: Local SSD



Cloud Storage Category 1: Block Storage – Instance Stores

- The physical machine running the virtual machine is physically connected to the storage device
 - virtual devices whose underlying hardware is physically attached to the host computer for the instance
 - Data transfer is limited by SATA / NVMe bandwidth
 - I3en.metal: 8 x 7,500 GB (60 TB)
- Since the machine may be rented to someone else in the next minute, the data stored on the drive does not persist, it's **ephemeral**.
- Example:
 - Amazon AWS: Instance Store
 - Google: Local SSD

Instance Size	Local Storage (GB)	Read MBps	Write GBps
i3.1large *	1 x 475 NVMe SSD	391	137
i3.xlarge *	1 x 950 NVMe SSD	806	273
i3.2xlarge	1 x 1,900 NVMe SSD	1,611	703
i3.4xlarge	2 x 1,900 NVMe SSD	3,223	1,406
i3.8xlarge	4 x 1,900 NVMe SSD	6,445	2,813
i3.16xlarge	8 x 1,900 NVMe SSD	12,891	5,469
i3.metal	8 x 1,900 NVMe SSD	12,891	5,469

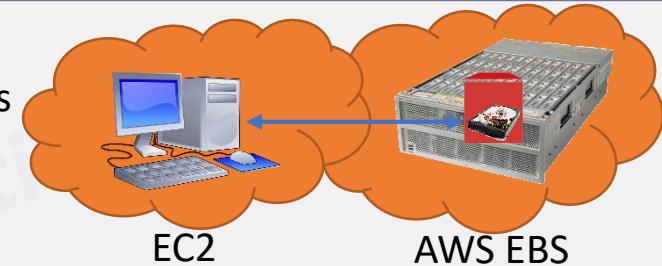
* Throughputs are a snapshot in time, might be different now

Storage Virtualization

- The process of presenting a logical view of the physical storage resources to a host computer system
 - Block Virtualization
 - File Virtualization

Cloud Block Storage – Virtual Block Stores

- Simulate a hard drive or SSD
- The physical machine running the virtual machine is separate from the physical machine hosting the data
 - NVMe over Fabric
 - NVMe or NVMe-oF
 - Data transfer is limited by network bandwidth
- The Hypervisor on the host machine has middleware that intercepts the network comm. and presents the network stream of bytes to the virtual machine as a “Block Storage Device”
 - E.g. in AWS, EBS as an NVMe device
 - E.g. in Google Cloud: Persistent Disk



Cloud Block Storage – Virtual Block Stores

- Typically less storage bandwidth than the previous option (EBS)
- You can also select how much bandwidth you are willing to pay for
 - Selecting IOPS for io1 type
- gp2 types are preselected and fixed
 - 3 IOPS/ gigabyte, 16KBps / IOPS, min 100, max 16,000
 - Bursting support
- Note: A single EC2 instance can be attached to more than 1 EBS volume
- Some instance types are EBS optimized
 - Bandwidth for EBS access is separate from network bandwidth
 - Other (micro, small, older generations) share network bandwidth

Instance Size	Instance Storage	EBS Bandwidth (Mbps)
m5.large	EBS-Only	Up to 594
m5.xlarge	EBS-Only	Up to 594
m5.2xlarge	EBS-Only	Up to 594
m5.4xlarge	EBS-Only	594
m5.8xlarge	EBS Only	850
m5.12xlarge	EBS-Only	1,188
m5.16xlarge	EBS Only	1,700
m5.24xlarge	EBS-Only	2,375
m5.metal	EBS-Only	2,375

* Throughputs are a snapshot in time, might be different now

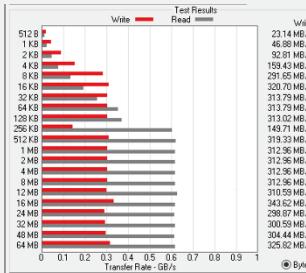
AWS Elastic Block Storage

- Depending on IO requirements, different offerings and prices
 - gp2 up to 250 MB/s @ \$0.10 per GB-month
 - io1 up to 1,000 MB/s @ \$0.125 per GB-month
* Prices and bandwidths are a snapshot in time, might be different now
AND \$0.065 per IOPS-month
 - ...
- Note: A single EC2 instance can be attached to more than 1 EBS volume
 - Max Throughput/Instance for T3 class: 2,375 MB/s

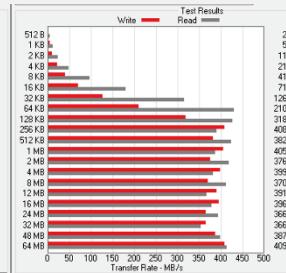
Instance Store vs. EBS Throughput

Experiment on an AWS m5ad.2xlarge instance (General Purpose, 8 vCPU, 32 GB RAM, 10GB network, EBS optimized)

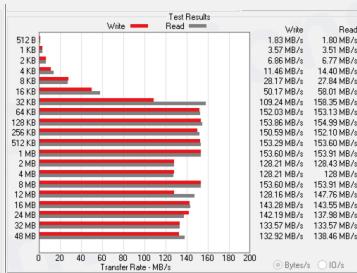
Instance Store 300GB



Io1 @ 200GB with 10000 IOPS



gp2 @ 30GB with 100/3000 IOPS

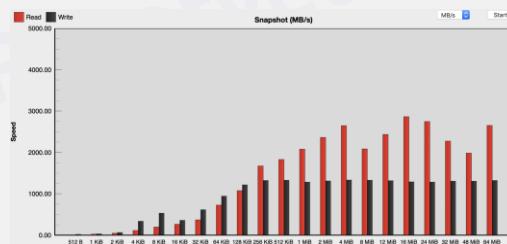
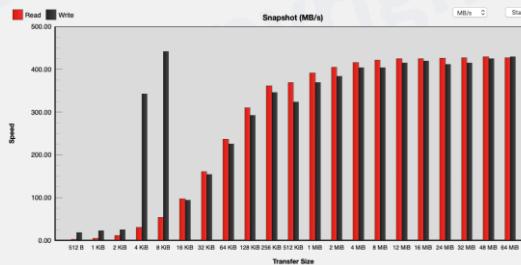


$$100*16KB = 1.56 \text{ MBps}$$

$$3000*16KB = 45 \text{ MBps}$$

- Bursting
- IO Consolidation at EBS Backend

Experiment on laptop (Macbook Pro)
SSD via USB 3.0 (5 Gbit/s) NVMe SSD (PCIe 3.0 × 4 8.0 GT/s (31.5 Gbit/s))



Summary

- Block Storage
- Instance Store
- Virtual Block Store
- Performance Comparison



CLOUD COMPUTING APPLICATIONS

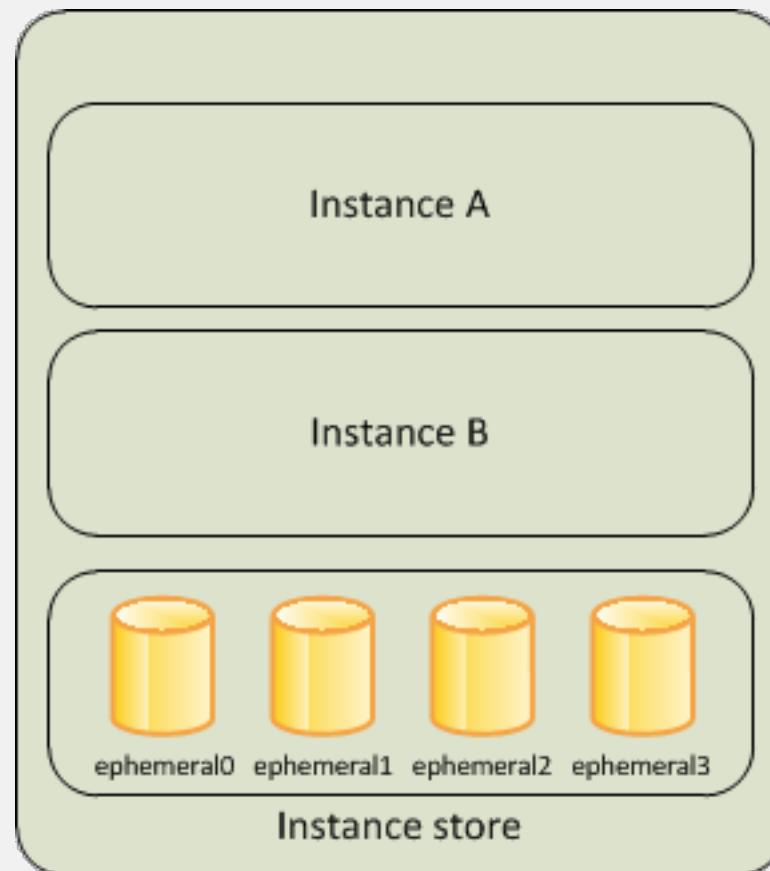
Amazon AWS Block Stores

Roy Campbell & Reza Farivar

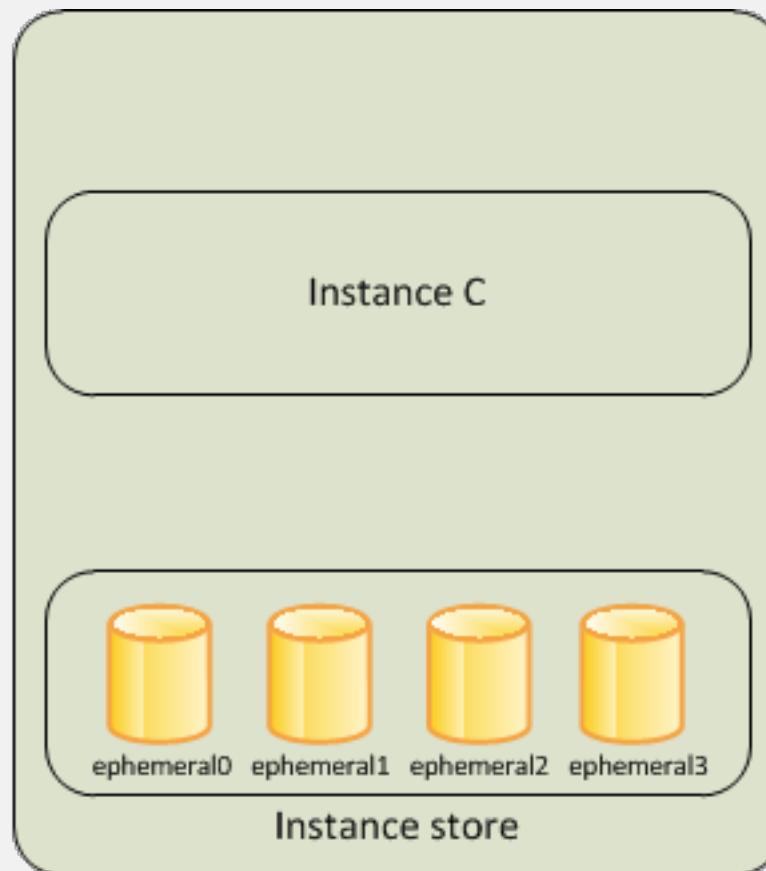
Amazon AWS Instance Store

- Instance stores are another form of cloud-hosted **temporary** block-level storage
 - These are provided as part of an 'instance', such as an Amazon EC2
- Their contents will be lost if the cloud instance is stopped.
 - But offer higher performance and bandwidth to the instance.
 - Located on disks that are physically attached to the host computer
- They are best used for temporary storage such as caching or temporary files, with persistent storage held on a different type of server.

Amazon AWS Instance Store



Host Computer 1



Host Computer 2

Instance Store Lifetime

- Data persists only during the lifetime of its associated instance
 - It persists a reboot
- Data lost if
 - The underlying disk drive fails
 - The instance stops
 - The instance terminates
- You can get reliability by
 - A distributed file system (e.g. HDFS)
 - Backup to S3 or EBS

Instance Store Size

- A typical instance store is small
 - SSD: can be anywhere from around 80 GB to 320 GB SSD, up to 3,840 GB on x1.32xlarge
 - HDD: when available (on older generation instances), up to 1,680 GB

Amazon AWS EBS

- EBS Volumes are highly available and reliable
- Can be attached to running instances in the same availability zones
 - Persist independently of the life of an instance
- Use when data must be quickly accessible and requires long-term persistence
- Support encryption
- Up to 16TB in size

Amazon AWS EBS

- Different types
 - General Purpose SSD (gp2)
 - 100 IOPS/GiB, burst up to 10,000, 160 MB/s throughput
 - Provisioned IOPS SSD (io1)
 - Provision a specific level of performance
 - up to 20,000 IOPS and 320 MB/s of throughput
 - Throughput Optimized HDD (st1)
 - Low cost magnetic storage
 - Throughput of up to 500 MB/s
 - Large, sequential workloads such as Amazon EMR, ETL, data warehouses, and log processing
 - Cold HDD (sc1)
 - Inexpensive magnetic
 - Throughput of up to 250 MB/s



CLOUD COMPUTING APPLICATIONS

Cloud Storage: Object Storage
Prof. Reza Farivar

Cloud Object Storage

- As we have seen, Distributed File Systems are not easy
 - Considerable overhead, complexity, cost
 - Main reason: maintaining consistency while providing transparency
- CAP: Consistency, Availability, Partition Tolerance
- Transparency: "invisible" to client programs, which "see" a system which is similar to a local file system.
 - Behind the scenes, the distributed file system handles locating files, transporting data, and potentially providing other features listed below
 - While transparency may seem trivial, these semantics can incur a significant performance penalty at scale despite not being strictly necessary



Internet Scale Storage: Breaking the Chains of Transparency and Consistency

- What if we want to scale to “unlimited Storage”?
- Solution: Stop trying to have all 3 components of CAP
 - Availability: Important to keep, otherwise customer data may be unavailable
 - Partition Tolerance: Networks do fail, cloud providers have to be resilient
 - Consistency: Can be scarified
- The typical BLOB storage by a cloud provider can scale “infinitely” by being “eventually consistent”
- In addition, they typically are not POSIX compliant
- The access model is through REST APIs
 - GET, PUT, DELETE
- Examples:
 - AWS S3
 - OpenStack Swift

AWS S3 Consistency Model

- Objects have a URI, and are accessible by REST API calls
 - <https://cloudApplications.s3.us-west-2.amazonaws.com/photos/picture1.jpg>
- If you PUT to an existing key, a subsequent read might return the old data or the updated data, but it never returns corrupted or partial data.
- For Availability, data will be replicated across AWS datacenters (Availability Zones)
 - If a PUT request is successful, your data is safely stored. But temporarily:
 - A process writes a new object to Amazon S3 and immediately lists keys within its bucket. Until the change is fully propagated, the object might not appear in the list.
 - A process replaces an existing object and immediately tries to read it. Until the change is fully propagated, Amazon S3 might return the previous data.
 - A process deletes an existing object and immediately tries to read it. Until the deletion is fully propagated, Amazon S3 might return the deleted data.
 - A process deletes an existing object and immediately lists keys within its bucket. Until the deletion is fully propagated, Amazon S3 might list the deleted object.
 - Amazon S3 does not currently support object locking. If two PUT requests are simultaneously made to the same key, the request with the latest timestamp wins
 - Updates are key-based. There is no way to make atomic updates across keys.

Cloud Object Storage

- By relaxing the consistency model, building the distributed storage system becomes much simpler
- The Storage costs are significantly cheaper
- Bandwidth can be quite high
 - Up to 25 GB/s
- Unlike the previous models (Block Storage, Managed File System), data can be accessible from outside the cloud
 - Your mobile app customers all over the world
 - The small number of computers you personally own

AWS S3 tiers comparison

	S3 Standard	S3 Intelligent-Tiering*	S3 Standard-IA	S3 One Zone-IA†	S3 Glacier	S3 Glacier Deep Archive	Service	Cost per 1TB / month
Designed for durability	99.999999999% (11 9's)	AWS EFS	\$300					
Designed for availability	99.99%	99.9%	99.9%	99.5%	99.99%	99.99%	AWS FSx Lustre	\$290
Availability SLA	99.9%	99%	99%	99%	99.9%	99.9%	EBS gp2	\$100
Availability Zones	≥3	≥3	≥3	1	≥3	≥3	AWS EFS infrequent access	\$25
Minimum capacity charge per object	N/A	N/A	128KB	128KB	40KB	40KB	S3 standard	\$23
Minimum storage duration charge	N/A	30 days	30 days	30 days	90 days	180 days	S3 infrequent	\$13
Retrieval fee	N/A	N/A	per GB retrieved	per GB retrieved	per GB retrieved	per GB retrieved	S3 Glacier	\$4
First byte latency	milliseconds	milliseconds	milliseconds	milliseconds	select minutes or hours	select hours	S3 Glacier deep archive	\$1
Storage type	Object	Object	Object	Object	Object	Object		
Lifecycle transitions	Yes	Yes	Yes	Yes	Yes	Yes		

* Prices and bandwidths are a snapshot in time, might be different now

Summary

- Cloud Object Storage
- Consistency Model
- API
- Tiers



CLOUD COMPUTING APPLICATIONS

SWIFT – An Object Store

Roy Campbell & Reza Farivar

Definition

A **Binary Large OBject (BLOB)** is a collection of binary data stored as a single entity in a database management system.

(The blob was a science fiction movie featuring Steve McQueen.)

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

Use case

- Store unstructured object data like text or binary data
- Images
- Movies
- Audio, Signal data
- Large queue of messages
- Example is LinkedIn data in a user page (Uses Ambry)
- Usually accessible over the web

Examples

- Windows Azure Blob Storage
- Ambry – LinkedIn
- Facebook's Warm BLOB Storage System
- Amazon Simple Storage Service (S3)*
- Apache Open Stack Blob Service (SWIFT)

Goals

- Data growth ~ 50% a year
- 50%-70% data is unstructured or archival
- RESTful API (HTTP)
- High availability (no single point of failure)
- Agile data centers
- Open Source
- Multi-region, geographic distribution of data
- Storage policies
- Erasure Coding

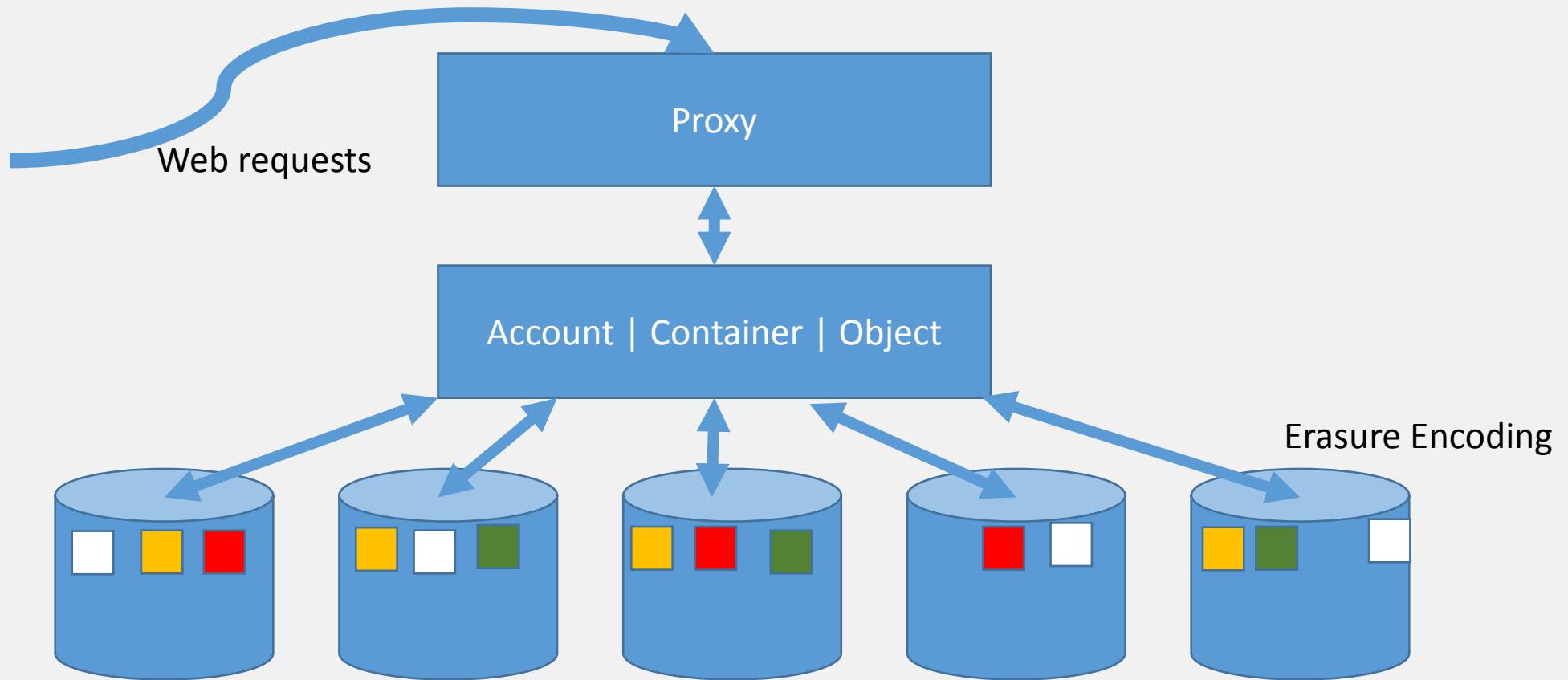
Swift API

https://swift.illinois.edu/version2/auth_account/container/object

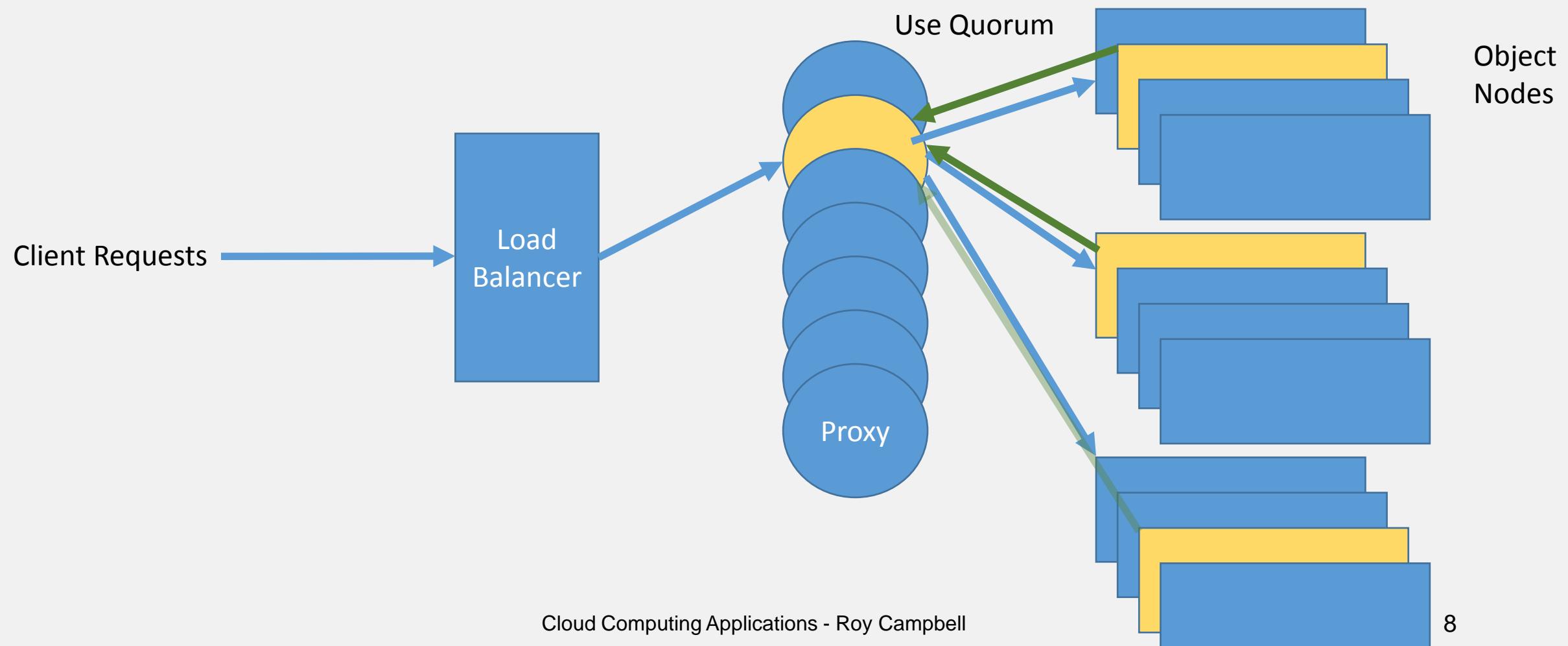
PUT /version2/roy/myblobs/classvideo1

GET /version2/reza/hisblobs/yesterdaysdataforhadoop

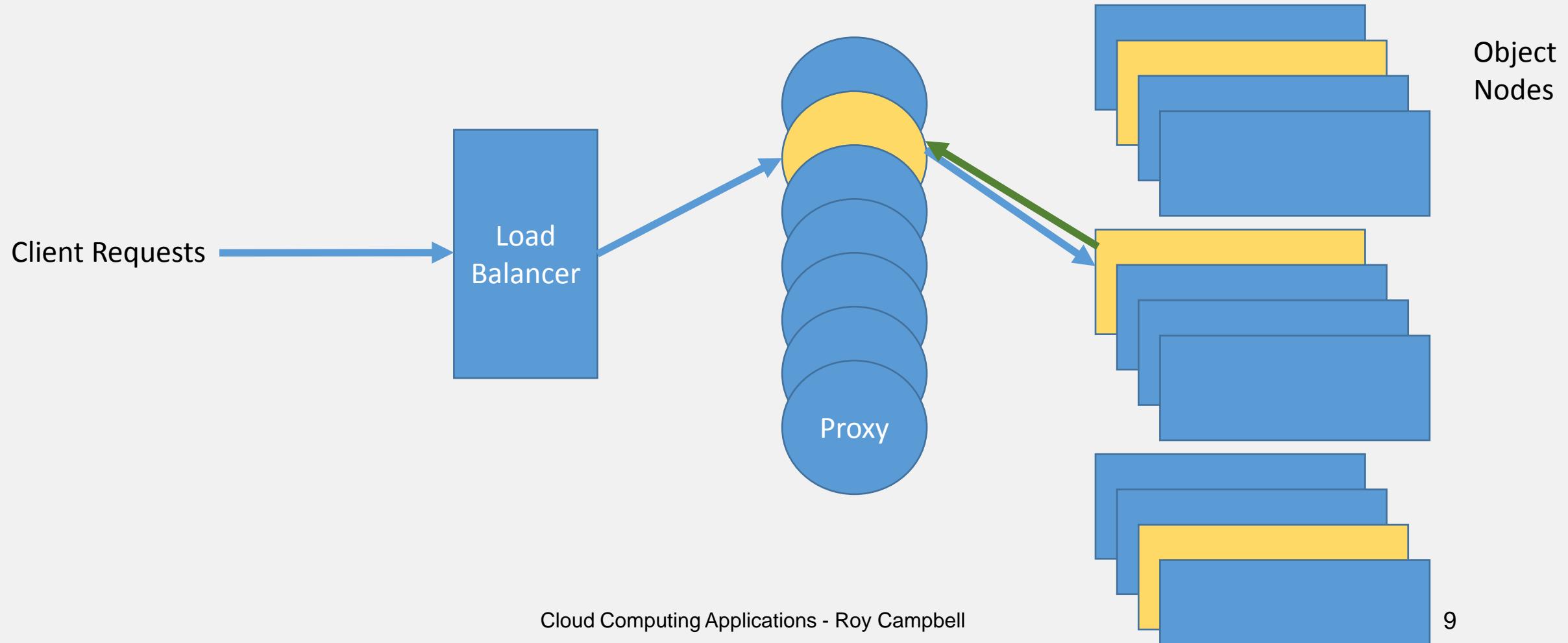
Swift Components



Write Requests - Load Balancer and Proxy



Read Requests - Load Balancer and Proxy

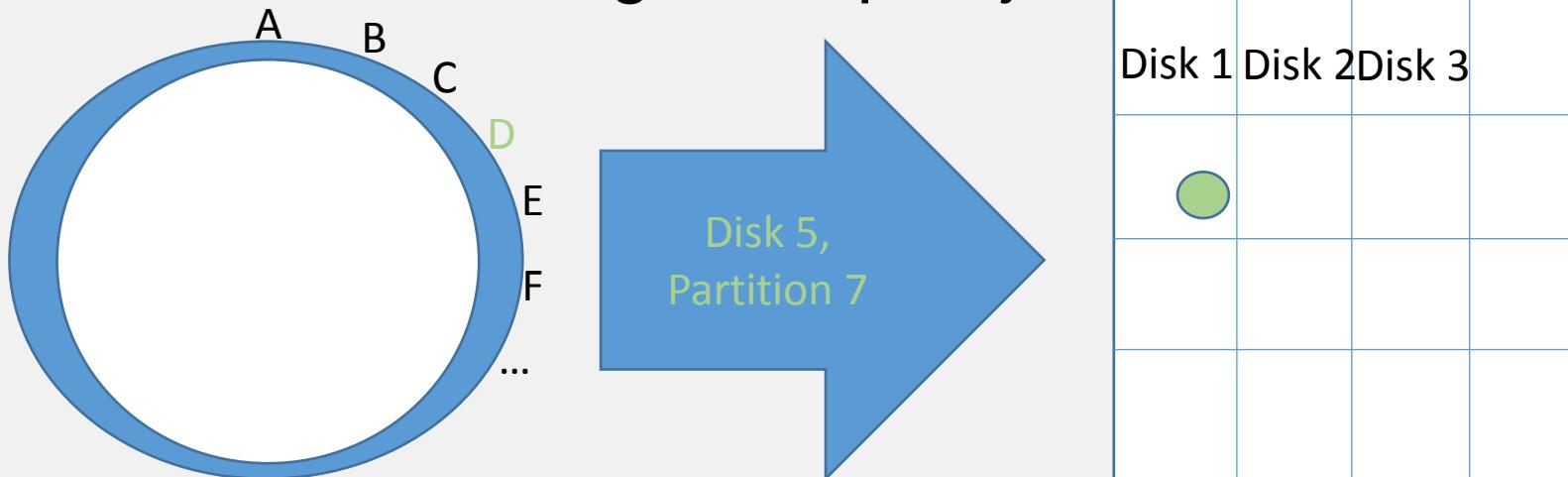


Details

- MD5 Checksums with each object
- Auditing and active replication
- Any sized disks

Swift Partitions

- 1 Node, 8 Disks, 16 Partitions per disk, $8*16 = 128$ partitions
- 2 Nodes, 8 Disks each, 8 Partitions per disk, $8*16 = 128$ partitions
- Use Hash into Ring to map objects into storage partitions





CLOUD COMPUTING APPLICATIONS

Amazon S3 BLOB Storage

Roy Campbell & Reza Farivar

Definition

Online file storage web service offered by Amazon Web Services.
Amazon S3 provides storage through web service interfaces
REST, SOAP, BitTorrent. (wikipedia)

<https://aws.amazon.com/s3/>

Use case

- Scalability, high availability, low latency – 99.99% availability
- Files up to 5 terabytes
- Objects stored in buckets owned by users
- User assigned keys refer to objects
- Amazon Machine Images (exported as a bundle of objects)
- SmugMug, Hadoop file store, Netflix, reddit, Dropbox, Tumbler

Simple Storage Service (S3)

- A **bucket** is a container for objects and describes location, logging, accounting, and access control.
 - A bucket has a name that must be **globally unique**.
 - `http://bucket.s3.amazonaws.com`
 - `http://bucket.s3-aws-region.amazonaws.com.`
- A bucket can hold any number of **objects**, which are files of up to 5TB.
 - `http://bucket.s3.amazonaws.com/object`
 - `http://johnsmith.s3.amazonaws.com/photos/puppy.jpg`

Fundamental operations corresponding to HTTP actions:

`http://bucket.s3.amazonaws.com/object`

- POST a new object or update an existing object.
- GET an existing object from a bucket.
- DELETE an object from the bucket
- LIST keys present in a bucket, with a filter.

A bucket has a **flat directory structure**

S3 Weak Consistency Model

“Updates to a single key are **atomic**....”

“Amazon S3 achieves high availability by replicating data across multiple servers within Amazon's data centers. If a PUT request is successful, your data is safely stored. However:

- A process writes a new object to Amazon S3 and immediately attempts to read it. Until the change is fully propagated, Amazon S3 might report "key does not exist."
- A process writes a new object to Amazon S3 and immediately lists keys within its bucket. Until the change is fully propagated, the object might not appear in the list.
- A process replaces an existing object and immediately attempts to read it. Until the change is fully propagated, Amazon S3 might return the prior data.
- A process deletes an existing object and immediately attempts to read it. Until the deletion is fully propagated, Amazon S3 might return the deleted data.”

S3 Command Line Interface

```
aws s3 mb s3://bucket
...
      cp localfile s3://bucket/key
      mv s3://bucket/key s3://bucket/newname
      ls s3://bucket
      rm s3://bucket/key
      rb s3://bucket
```

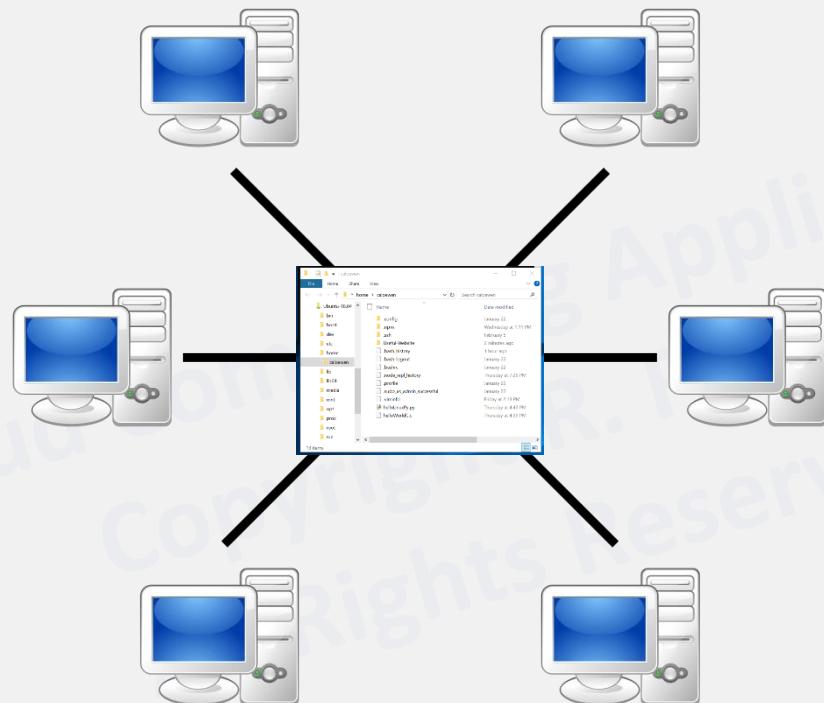
```
aws s3 help
aws s3 ls help
```



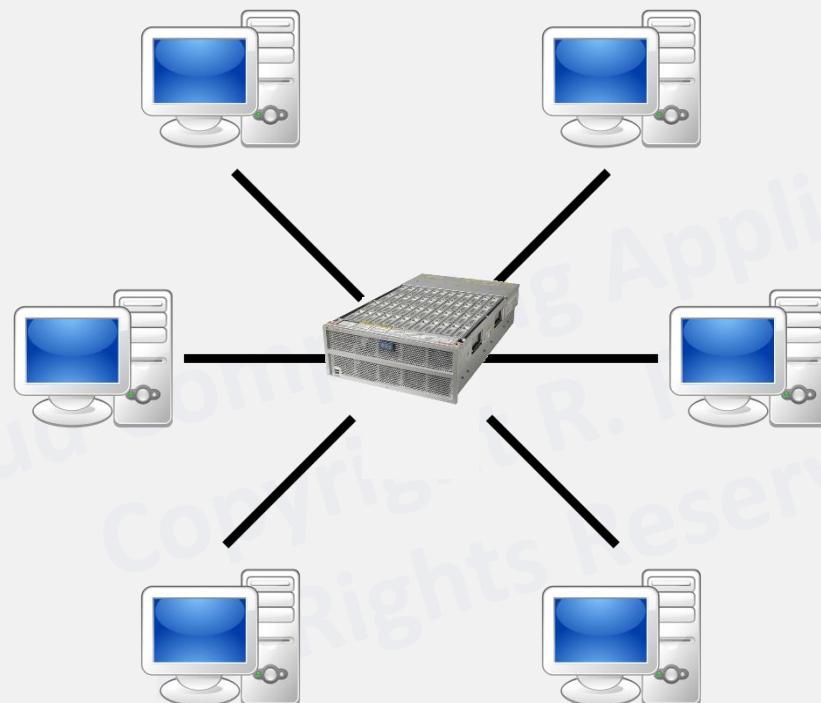
CLOUD COMPUTING APPLICATIONS

Cloud Storage: Managed File Systems
Prof. Reza Farivar

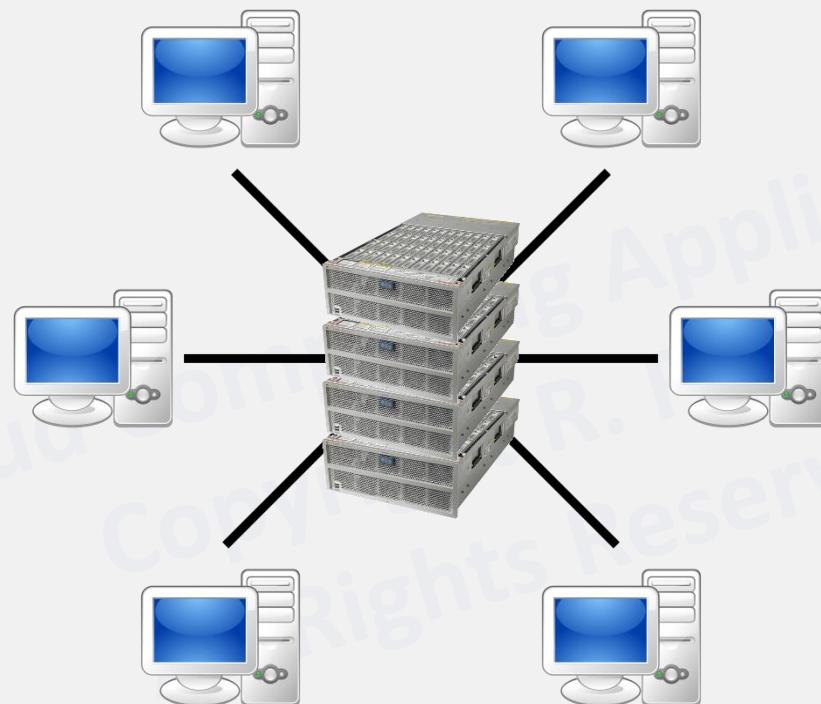
Logical View Networked File System



Network Attached Storage (NAS)



Network Attached Storage (NAS)



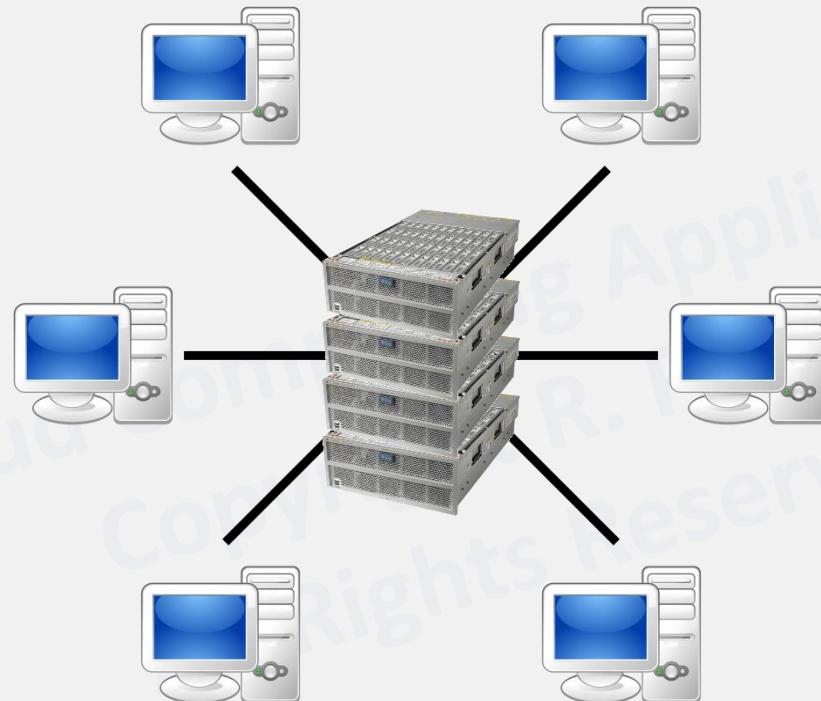
Clustered File Systems

- A clustered file system allows files to be accessed using the same interfaces and semantics as local files – for example, mounting/unmounting, listing directories, read/write at byte boundaries, system's native permission model.
 - Fencing
 - Concurrency
 - Consistency
- Examples
 - NFS
 - Unix
 - V4 , V4.1 (pNFS extension)
 - SMB
 - Windows
 - Lustre
 - HPC
 - Ceph
 - Many OpenStack implementations use Ceph as the storage substrate
 - Gluster
 - Classic file serving, second-tier storage, and deep archiving

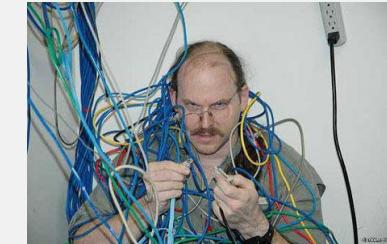
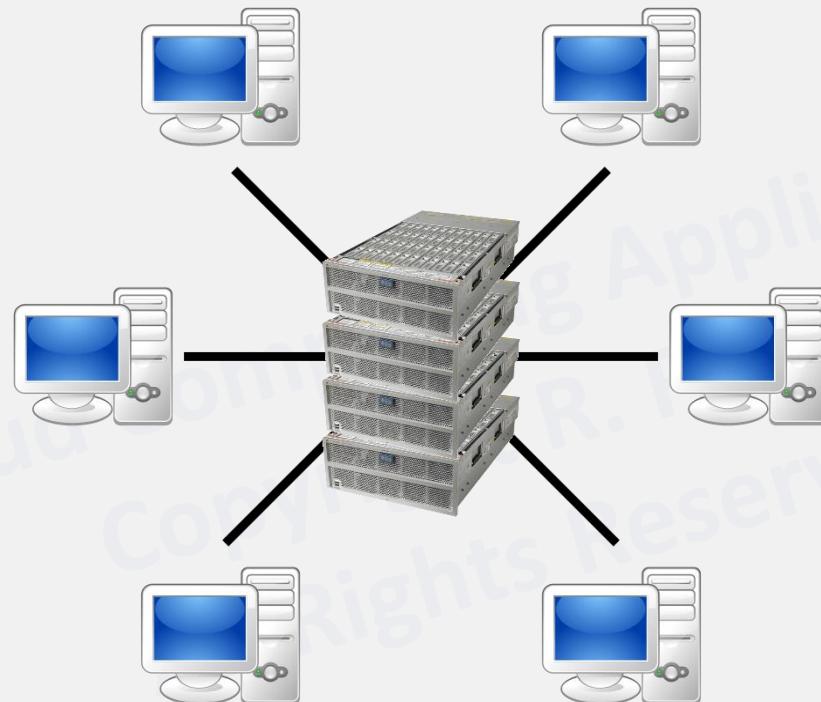
Clustered File System Consistency

- Ideally, a parallel file system would completely hide the complexity of distributed storage and exhibit a behavior as it is specified by POSIX
 - Fencing
- Relatively easy on one Machine
- Much harder on a cluster of servers
- Maintaining CAP is extremely hard
 - **Consistency:** Every read receives the most recent write or an error
 - **Availability:** Every request receives a (non-error) response, without the guarantee that it contains the most recent write
 - **Partition tolerance:** The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes

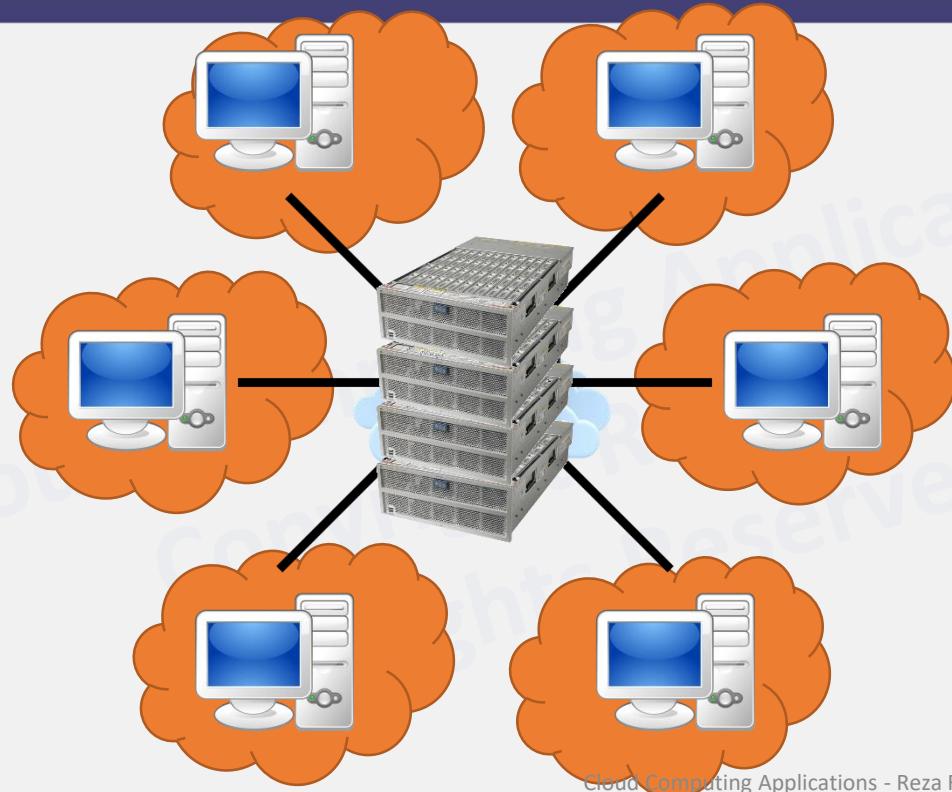
Network Attached Storage (NAS)



Network Attached Storage (NAS)

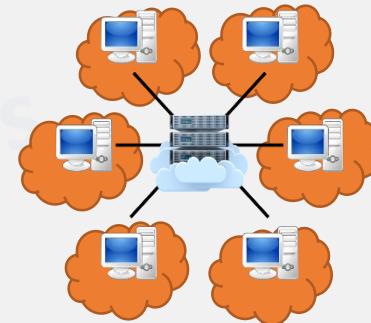


Network Attached Storage (NAS)



Cloud-Based File Systems

- Roll your own Clustered Distributed File System
 - Grab a number of “storage optimized” virtual machines, or metal machines
 - Each with large and fast instance block stores
 - A.k.a. SSDs / HDDs attached to the instance
 - Keep those instances on 24/7
 - Install a distributed file system on them
 - Lustre, MS DFS, NFS, Gluster, Ceph, Hadoop, etc.
- Managed filesystem deployed by the Cloud Provider



Cloud-managed file system

- Amazon
 - AWS FSx for Lustre
 - AWS FSx for Windows File Server
 - AWS EFS
- Azure
 - Azure Files
 - SMB access protocol
 - REST API
 - Azure DataLake Storage
 - Hadoop compatible file system
- Google
 - Cloud Filestore
 - NFS v3
 - Up to 64 TB

Google Filestore, IBM Cloud File Storage, AWS EFS

- Google Filestore

Simple commands to create a Filestore instance with gcloud.

```
gcloud filestore instances create nfs-server \
--project=[PROJECT_ID] \
--zone=us-central1-c \
--tier=STANDARD \
--file-share=name="vol1",capacity=1TB \
--network=name="default"
```

Simple commands to install NFS, mount your file share, and set access permissions.

```
sudo apt-get -y update
sudo apt-get -y install nfs-common
sudo mkdir /mnt/test
sudo mount 10.0.0.2:/vol1 /mnt/test
sudo chmod go+rW /mnt/test
```

	Standard	Premium
Max read throughput	100 MB/s (1 TB), 180 MB/s (10+ TB)	1.2 GB/s
Max write throughput	100 MB/s (1 TB), 120 MB/s (10+ TB)	350 MB/s
Max IOPS	5,000	60,000
Max capacity per share	63.9 TB	63.9 TB
Typical customer availability	99.9%	99.9%
Protocol	NFSv3	NFSv3
Price	See Cloud Filestore pricing for more information	See Cloud Filestore pricing for more information

- IBM Cloud File Storage

- Up to 12 TB
- Up to 48,000 IOPS

- One beefy machine can handle them

- AWS i3en.24xlarge: 8 x 7,500 NVMe SSD, 100 Gbps network
 - \$60,405 / year per reserved instance
 - Cost of 64 TB standard storage EFS: $\$0.3 * 12 * 1024 * 64 = \$235,929$
 - Provisioned IO up to 1 Gbps ≈ 10 Gbps

Are Managed File Systems Distributed?

- FSx for Windows File Server
 - Max size: 64 TB
 - Can Utilize Microsoft DFS to unify data from many file servers for hundreds of petabytes
 - Shared namespace: Location transparency
 - Replication: Redundancy
- FSx for Lustre
 - Max size: 100 TB
 - Throughput: Read 50-200 MB/s per TB, can burst to 3,000 MB/s per TB
 - E.g. 50.4 TB runs on 22 file servers
 - Hundreds of GB/s of throughput
- AWS EFS
 - Maximum size: “Petabytes”
 - The throughput available to a file system scales as a file system grows
 - 50 MB/s per TB, can burst to 100 MB/s per TB
 - Supports NFS v4.1
- Azure
 - Azure Files
 - 100 TB
 - Data Lake Storage Gen 2
 - HDFS semantics
 - Built on top of Azure Blob Storage
 - Distributed file system
 - Can serve “many exabytes”
 - Throughput measured in gigabits per second (Gbps)

Distributed file Systems Design Goals

- *Access transparency*: clients are unaware that files are distributed and can access them in the same way as local files are accessed.
- *Location transparency*: a consistent namespace exists encompassing local as well as remote files. The name of a file does not give its location.
- *Concurrency transparency*: all clients have the same view of the state of the file system. This means that if one process is modifying a file, any other processes on the same system or remote systems that are accessing the files will see the modifications in a coherent manner.
- *Failure transparency*: the client and client programs should operate correctly after a server failure.
- *Heterogeneity*: file service should be provided across different hardware and operating system platforms.
- *Scalability*: the file system should work well in small environments (1 machine, a dozen machines) and also scale gracefully to bigger ones (hundreds through tens of thousands of systems).
- *Replication transparency*: Clients should be unaware of the file replication performed across multiple servers to support scalability.
- *Migration transparency*: files should be able to move between different servers without the client's knowledge.

Summary

- Clustered File Systems
- File Systems in the Cloud



CLOUD COMPUTING APPLICATIONS

Roy Campbell & Reza Farivar

Amazon AWS EFS

Amazon AWS EFS

- Elastic File System
- Motivation: enterprise customers need a large distributed file system
 - S3 is large and distributed, but it is an object store without performance guarantees and eventual consistency model
 - Block storage (EBS, instance store) are small
 - Enterprise can build a distributed file system on top of these, but it requires operational expertise
 - Glacier: Good for only archival storage
- EFS provides a fully NFSv4 compliant network file system

Amazon AWS EFS

- SSD backed
- Highly available and highly durable
 - files, directories, and links are stored redundantly across multiple Availability Zones within an AWS region
- Grow or shrink as needed
 - No need to pre-provision capacity

Amazon AWS EFS

		Amazon EFS	Amazon EBS PIOPS
Performance	Per-operation latency	Low, consistent	Lowest, consistent
	Throughput scale	Multiple GBs per second	Single GB per second
Characteristics	Data Availability/Durability	Stored redundantly across multiple AZs	Stored redundantly in a single AZ
	Access	1 to 1000s of EC2 instances, from multiple AZs, concurrently	Single EC2 instance in a single AZ
	Use Cases	Big Data and analytics, media processing workflows, content management, web serving, home directories	Boot volumes, transactional and NoSQL databases, data warehousing & ETL



CLOUD COMPUTING APPLICATIONS

Cloud Storage: Archive Storage and Backup
Prof. Reza Farivar

Cloud Archive Storage

- The availability of the storage has a direct impact on costs
- If you know file requests are very infrequent, you can store them in cold storage
- Cloud providers offer archive storage at much reduced costs
 - However, access to them is relatively expensive
- Different tiers of archive storage

Amazon Archive Storage: S3 Glacier

- AWS S3 Glacier
 - \$0.004 per GB / mo
 - Compare with S3
 - Frequent access tier: \$0.023 per GB / mo
 - Infrequent access tier: \$0.0125 per GB / mo
- Glacier Retrieval
 - retrieval option from 1 minute to 12 hours

* Prices and bandwidths are a snapshot in time, might be different now

Retrieval Time	Data Retrievals
Expedited (typically available within 1 – 5 minutes)	\$0.03 per GB / mo
Standard (typically accessible within 3 – 5 hours)	\$0.01 per GB / mo
Bulk	\$0.0025 per GB / mo

Retrieval request	Retrieval Time	Retrieval Requests
Expedited		\$10.00 per 1,000 requests
Standard		\$0.05 per 1,000 requests
Bulk		\$0.025 per 1,000 requests

- S3 Glacier Deep Archive
 - For long-term data archiving that is accessed once or twice in a year and can be restored within 12 hours
 - \$0.00099 per GB / mo

AWS S3 tiers comparison

	S3 Standard	S3 Intelligent-Tiering*	S3 Standard-IA	S3 One Zone-IA†	S3 Glacier	S3 Glacier Deep Archive	Service	Cost per 1TB / month
Designed for durability	99.999999999% (11 9's)	AWS EFS	\$300					
Designed for availability	99.99%	99.9%	99.9%	99.5%	99.99%	99.99%	AWS FSx Lustre	\$290
Availability SLA	99.9%	99%	99%	99%	99.9%	99.9%	EBS gp2	\$100
Availability Zones	≥3	≥3	≥3	1	≥3	≥3	AWS EFS infrequent access	\$25
Minimum capacity charge per object	N/A	N/A	128KB	128KB	40KB	40KB	S3 standard	\$23
Minimum storage duration charge	N/A	30 days	30 days	30 days	90 days	180 days	S3 infrequent	\$13
Retrieval fee	N/A	N/A	per GB retrieved	per GB retrieved	per GB retrieved	per GB retrieved	S3 Glacier	\$4
First byte latency	milliseconds	milliseconds	milliseconds	milliseconds	select minutes or hours	select hours	S3 Glacier deep archive	\$1
Storage type	Object	Object	Object	Object	Object	Object		
Lifecycle transitions	Yes	Yes	Yes	Yes	Yes	Yes		

* Prices and bandwidths are a snapshot in time, might be different now

Cloud-managed Backups

- Some cloud providers allow managed backups from their block stores or managed file systems or other data stores
 - Distributed Backup is not a simple task
- Example: Amazon backup

Resource Type	Warm Storage	Cold Storage
Amazon EFS File System Backup	\$0.05 per GB-Month	\$0.01 per GB-Month
Amazon EBS File System Backup	\$0.05 per GB-Month	n/a†
Amazon RDS Database Snapshot	\$0.095 per GB-Month	n/a†
Amazon DynamoDB Table Backup	\$0.10 per GB-Month	n/a†
AWS Storage Gateway Volume Backup	\$0.05 per GB-Month	n/a†

- Amazon restore

Resource Type	Warm Storage	Cold Storage	Item-level Restore
Amazon EFS File System Backup	\$0.02 per GB	\$0.03 per GB	\$0.5 per request
Amazon EBS Volume Snapshot	Free	n/a†	n/a**
Amazon RDS Database Snapshot	Free	n/a†	n/a**
Amazon DynamoDB Table Backup	\$0.15 per GB	n/a†	n/a**
AWS Storage Gateway Volume Backup	Free	n/a†	n/a**

- Example: Azure

- Only supports backups of VMs (block stores) and SQL workloads





CLOUD COMPUTING APPLICATIONS

Ceph

Roy Campbell & Reza Farivar

Motivation

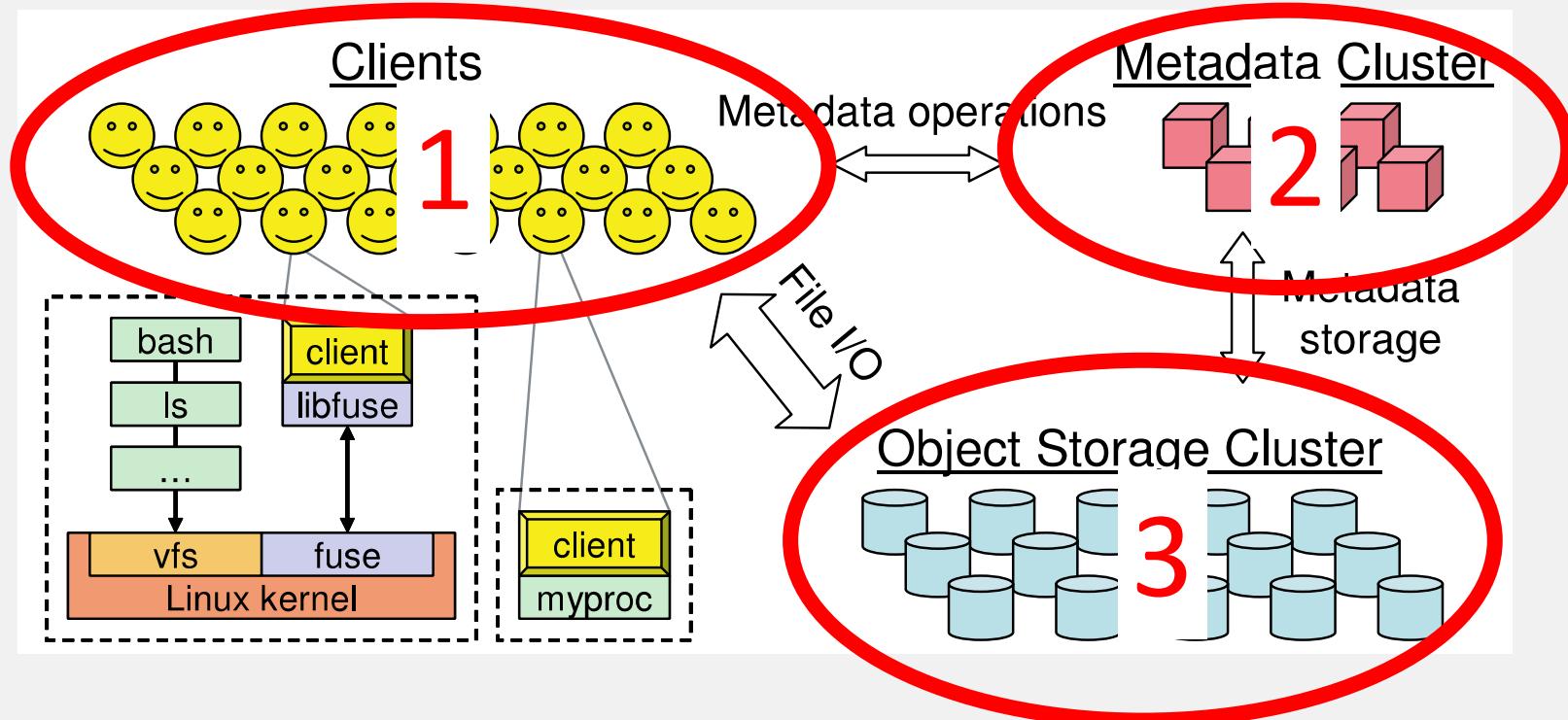
- Ceph is an emerging technology in the production-clustered environment
- Designed for:
 - Performance – Striped data over data servers
 - Reliability – No single point of failure
 - Scalability – Adaptable metadata cluster
 - More general than HDFS
 - Smaller files
- GlusterFS has more or less similar ideas
 - Uses ring-based hashing; Ceph uses CRUSH

Ceph Overview

- MDS – Meta Data Server
- ODS – Object Data Server
- MON – Monitor (now fully implemented)
- **Decoupled data and metadata**
 - I/O directly with object servers
- **Dynamic distributed metadata management**
 - Multiple metadata servers handling different directories (subtrees)
- **Reliable autonomic distributed storage**
 - ODS's manage themselves by replicating and monitoring

Ceph Components

- Ordered: Clients, Metadata, Object Storage



Decoupled Data and Metadata

- Increases performance by limiting interaction between clients and servers
- Decoupling is common in distributed filesystems: HDFS, Lustre, Panasas...
- In contrast to other file systems, Ceph uses a function to calculate the block locations

Dynamic Distributed Metadata Management

- Metadata is split among cluster of servers
- Distribution of metadata changes with the number of requests to even load among metadata servers
- Metadata servers also can quickly recover from failures by taking over neighbors' data
- Improves performance by leveling metadata load

Reliable Autonomic Distributed Storage

- Data storage servers act on events by themselves
- Initiates replication and
- Improves performance by offloading decision making to the many data servers
- Improves reliability by removing central control of the cluster (single point of failure)



CLOUD COMPUTING APPLICATIONS

Amazon AWS Glacier

Roy Campbell & Reza Farivar

Amazon AWS Glacier

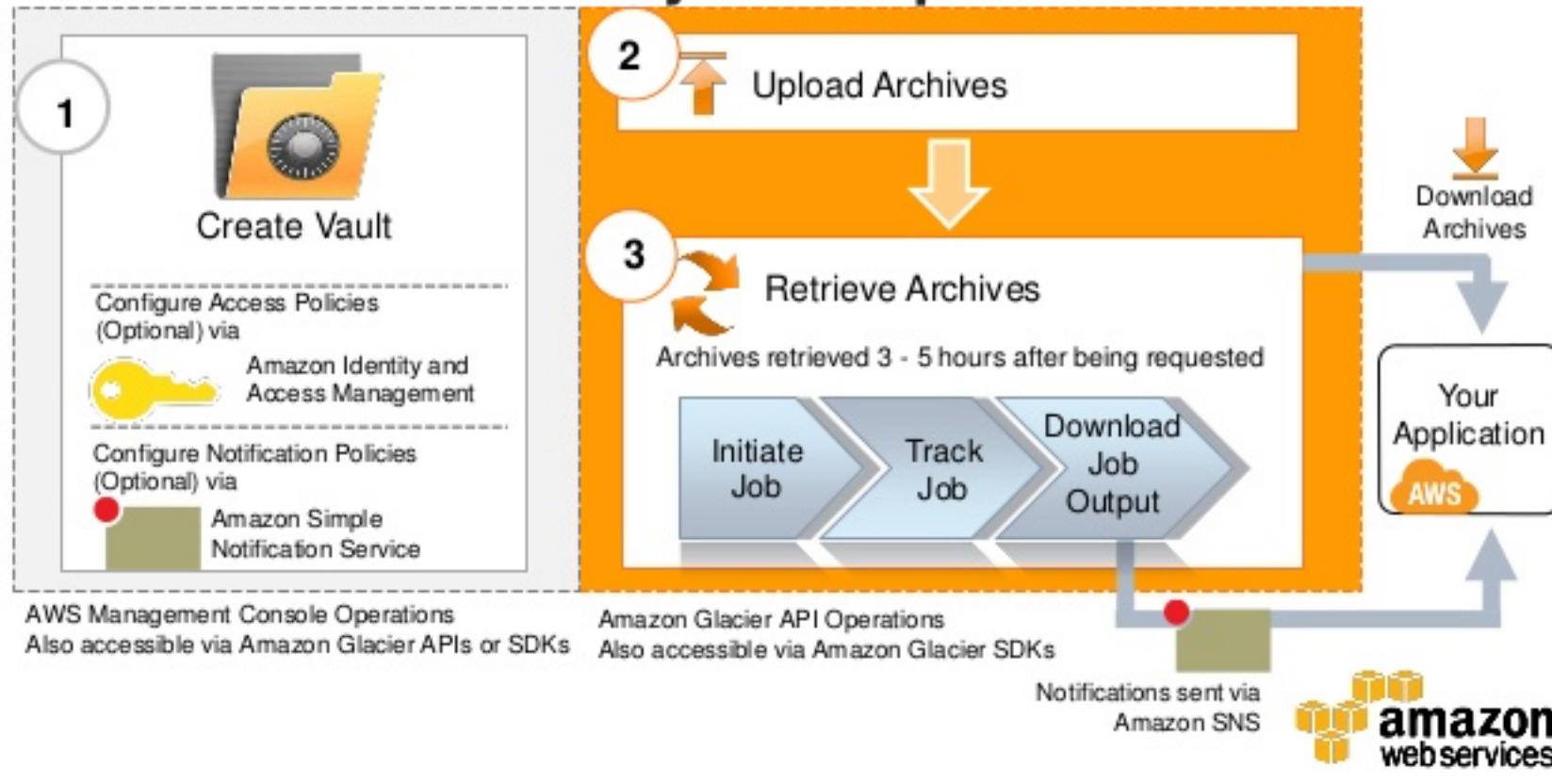
- Allows you to archive your data
- Very low cost, \$0.007 per GB per month
- Very durable
 - Average annual durability of 99.99999999%
- Each single archive up to 40 TB

Amazon AWS Glacier

- Archives stored in vaults
- The main access point to glacier is S3
- Typically takes between **3 to 5 hours** to prepare a download request
 - After that you have 24 hours to download from the staging location

Amazon AWS Glacier

Amazon Glacier Key Concepts





CLOUD COMPUTING APPLICATIONS

Cloud Storage: Storage Gateway
and Mass Data Transfer
Prof. Reza Farivar

Cloud Storage Gateways

- Hybrid Cloud
 - Some VMs on your infrastructure, some on public cloud
 - Need a way to connect storage locations
- AWS Storage Gateway
 - File Gateway
 - NFS, SMB
 - S3 REST API access
 - POSIX-style metadata, including ownership, permissions, and timestamps stored as S3 user metadata
 - Tape Gateway
 - Volume Gateway
 - Presented as iSCSI protocol
 - Stores in AWS as EBS
- Azure
 - Data Box Gateway
 - SMB, NFS
 - Azure Blob Storage
 - Azure Files
 - File Sync
 - StorSimple

Cloud Mass Data Transfer

- Once a company decides to move to the cloud, there is a mass of data that needs to be transferred to the cloud
 - Others may need to transfer data into and out of the cloud
 - Problems:
 - The cost of mass data transfer could get astronomical
 - Network reliability could make it very hard
 - Typical Solution: Cloud provider sends you a physical device
 - AWS
 - Snowball: \$250 / 80 TB
 - Moving data into Amazon is free, same as S3
 - In comparison, transferring 80TB into/out of S3 can take 7-8 days on a 1Gbps internet connection, if there is no data transmission error
 - $80*8*1024 / (60*60*24) = 7.58 \text{ days}$
 - Moving 80TB out of S3 can cost
 - $80*1024*0.08 = \$6,553$
 - Moving 80TB out through Snowball
 - $80*1024*0.03 = \$2,457$
 - Snowmobile
 - Azure Data Box
 - Data Box: \$250 / 100 TB
 - Data Box Disk: \$50 / 8 TB
 - Data Box Heavy: \$4,000 / 1 PB
- Shipping fees also apply*



AWS Snowball: 50-80 TB



Cloud Mass Data Transfer

- Once a company decides to move to the cloud, there is a mass of data that needs to be transferred to the cloud
 - Others may need to transfer data into and out of the cloud
 - Problems:
 - The cost of mass data transfer could get astronomical
 - Network reliability could make it very hard
 - Typical Solution: Cloud provider sends you a physical device
 - AWS
 - Snowball: \$250 / 80 TB
 - Moving data into Amazon is free, same as S3
 - In comparison, transferring 80TB into/out of S3 can take 7-8 days on a 1Gbps internet connection, if there is no data transmission error
 - $80*8*1024 / (60*60*24) = 7.58$ days
 - Moving 80TB out of S3 can cost
 - $80*1024*0.08 = \$6,553$
 - Moving 80TB out through Snowball
 - $80*1024*0.03 = \$2,457$
 - Snowmobile
 - Azure Data Box
 - Data Box: \$250 / 100 TB
 - Data Box Disk: \$50 / 8 TB
 - Data Box Heavy: \$4,000 / 1 PB
- * Shipping fees also apply*



AWS Snowmobile: 100 PB

Assuming a 7-day transfer time, bandwidth = 824 Gbps



Summary

- Storage Gateway
- Mass Data Transfer



CLOUD COMPUTING APPLICATIONS

Cloud Storage: Internet-level Filesystem Storage
Prof. Reza Farivar

Internet-level Filesystem Storage

- Object Storage
 - AWS S3, Azure Blob Storage, etc.
- DropBox
- Google Cloud
- Box
- Apple iCloud Drive

Internet-level Filesystem Storage

- Synch and access your file system across a limited number of owned devices
- Block-level file copying
- Synch throttling
- Shared folders and links
- Data recovery (e.g. 30 days, 180 days, etc.)
- File Sync with local file system
- Web-based access
- Shared folders with other users
- Analytics services
 - Text search in documents and images
 - etc.

Data access frequency

- Unlike Object Stores (AWS S3, Azure Blobs, etc.) this market segment does not accept access pattern limits
- Cloud providers use statistics to extract average access patterns and set pricing
 - They may lose money on some customers that access / change their data all the time
 - In average, they have positive profit
- Case study: DropBox
 - Dropbox started as a startup by storing their customer's data on Amazon S3
 - IN 2016 Dropbox moved off of Amazon and onto their own datacenters for storage
 - The economy of scale had grown so far that their “cloudonomics” dictated they should build vs. rent
 - Think of Amazon AWS as an incubator for startups
 - Netflix moved off its CDN from third parties to its own network once it grew large enough
 - Netflix still uses AWS & Google Cloud for everything else: compute, analytics, etc.
 - No Datacenter

Service	Cost per 2TB / month
AWS EFS	\$600
AWS EFS infrequent access	\$50
S3 standard	\$46
S3 infrequent	\$25
S3 Glacier	\$8
S3 Glacier deep archive	\$2
Do not include transfer costs	
Box	\$15
Dropbox	\$10
Google Cloud	\$10
Amazon Drive	\$10
Apple iCould Drive	\$10
pCloud	\$10
Microsoft OneDrive	\$7
Sync.com	\$7

* Prices and bandwidths are a snapshot in time, might be different now

Integration

- Custom integration in company ecosystem
 - Microsoft OneDrive: Comes with Office 365
 - Google Cloud: Integration with Google Docs, Gmail
 - Apple iCould Drive: Integration with iPhones, Macs
 - Dropbox: first to market, product typically more robust, integration with both MS Office and Google Docs
 - Box: Enterprise and government features

Summary

- End-user facing Cloud Storage
- Many competing offerings
- Utilize low frequency access pattern for low-cost offerings



CLOUD COMPUTING APPLICATIONS

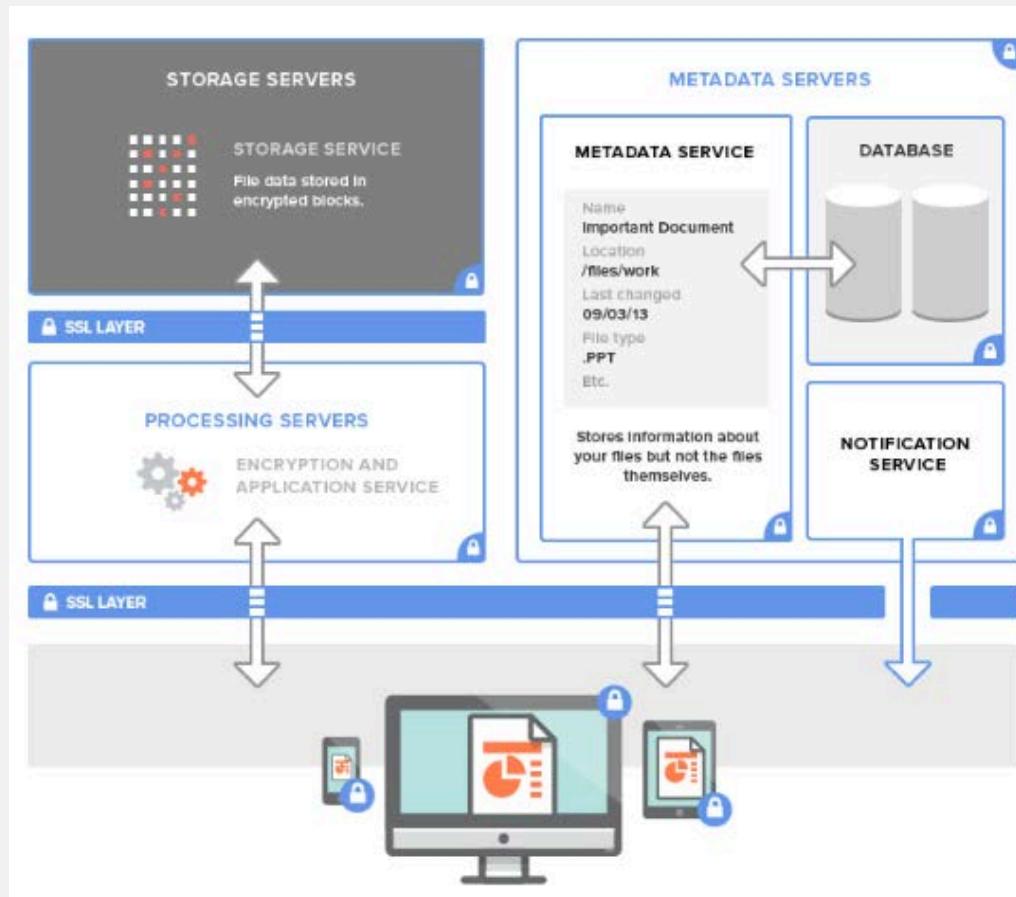
Roy Campbell & Reza Farivar

Dropbox Cloud API

Cloud Storage

- One interesting case study is Dropbox
- Dropbox offers cloud file storage
 - Easily synced across multiple devices
 - Accessible through web interface, mobile apps, and directly integrated with the file system on PCs
- Dropbox itself uses clouds!
 - Metadata stored in Dropbox servers
 - Actual files stored in Amazon S3
 - Amazon EC2 instances run the logic

Dropbox Architecture

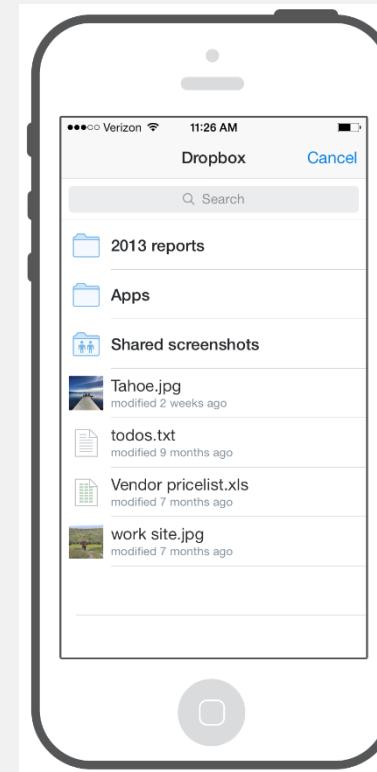


Dropbox API

- Two levels of API access to Dropbox
 - Drop-ins
 - Cross-platform UI components that can be integrated in minutes
 - *Chooser* allows instant access to files in Dropbox
 - *Saver* makes saving files to Dropbox easy
 - Core API
 - Support for advanced functionality like search, revisions, and restoring file
 - Better fit for deeper integration

Drop-In API

- Simple objects
 - Chooser available for JavaScript, Android and iOS
 - Saver on web and mobile web
- Handles all the authentication (OAuth), file browsing
- Chooser object returns the following:
 - Link: URL to access the file
 - File name
 - File Size
 - Icon
 - Thumbnails
- Saver
 - Pass in URL, filename and options



Core API

- Many languages and environments
 - Python, Ruby, PHP, Java, Android, iOS, OS X, HTTP
- Based on HTTP and OAuth
 - OAuth v1, OAuth v2
- Low-level calls to access and manipulate a user's Dropbox account
 - Create URL schemes
 - Upload files
 - Download files
 - List files and folders
 - Delta
 - Metadata access
 - Create and manage file sharing



CLOUD COMPUTING APPLICATIONS

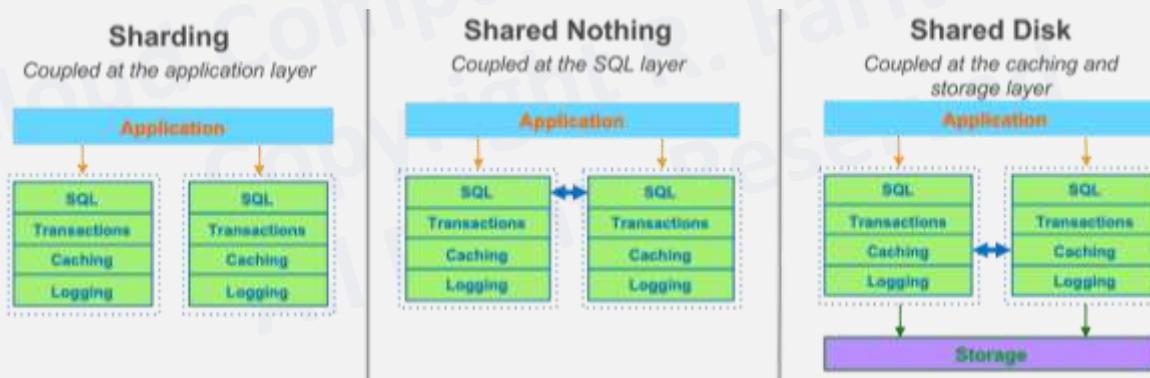
Cloud Databases – Managed RDBMS
Prof. Reza Farivar

Relational Cloud Databases

- For decades, managing a relational database has been a high-skill, labor-intensive task
- Relational databases store data with predefined schemas and relationships between them
- These databases are designed to support ACID transactions, maintain referential integrity and strong data consistency.
 - Atomicity
 - Consistency
 - Isolation
 - Durability
- OLTP workloads
 - On-Line Transactional Processing (OLTP)

Relational Databases

- In large systems, failures are a norm, not an exception
- Users want to start with a small footprint and then grow massively without infrastructure limiting their velocity
- Replication
 - Storage (SAN, NAS, Aurora)
 - Database
 - Application



Sharding

- Sharding: Split data set by certain criteria and store such “shards” on separate “clusters”
 - Sharding can be considered an embodiment of the "share-nothing" architecture and essentially involves breaking a large database into several smaller databases
- One common way to split a database is splitting tables that are not joined in the same query onto different hosts
- Another method is duplicating a table across multiple hosts and then using a hashing algorithm to determine which host receives a given update

Managed Relational Databases

- Traditional single server databases running on a virtual machine
 - AWS RDS: MySQL, PostgreSQL, MariaDB, Oracle, MS SQL server
 - Azure SQL Database, Database for MySQL, PostgreSQL, MariaDB
 - Google Cloud SQL
 - IBM Cloud Databases for PostgreSQL, DB2 on cloud
- Instances are fully managed, relational MySQL, PostgreSQL, and SQL Server databases
- Cloud provider handles replication, patch management, and database management to ensure availability and performance
- Availability through failover
- Horizontal Scalability through read replicas
 - Vertical scalability by using larger machines (64 processors, 400GB RAM)

Managed Relational Databases

- Typically the database instance is accessible by most compute resources in the cloud provider's network
 - Virtual Machines (AWS EC2, Azure VMs, Google Compute Engine)
 - PaaS (Aws Elastic beanstalk, Google App Engine)
 - Serverless (AWS lambda, Google Cloud Functions, Azure functions, etc.)
- Over the internet
 - SQL Proxy
 - Google Cloud SQL Proxy for public interfacing
 - `./cloud_sql_proxy -instances=INSTANCE_CONNECTION_NAME=tcp:3306 &`
 - `import pymysql`
`connection = pymysql.connect(host='127.0.0.1',`
`user='DATABASE_USER',`
`password='PASSWORD'`
`db='DATABASE_NAME')`
- Encryption at rest and in transport



CLOUD COMPUTING APPLICATIONS

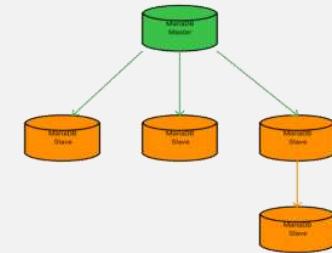
Cloud Databases – Multinode RDBMS
Prof. Reza Farivar

Beyond a Single Node

- Replication Options in MySQL
 - Classical MySQL Replication
 - One master, multiple slaves
 - Support for Many masters to many slaves
 - asynchronous
 - No Conflict Resolution or “Protection”
 - MySQL Group Replication
 - distributed state machine replication with strong coordination between servers
 - Built on Paxos
 - Majority vote for transaction commit
 - Network partition can stop the system
 - Multi Master - Galera
 - Multi-master
 - MySQL (NDB) Cluster
 - Synchronous

Replication in Databases

- Replication is a feature allowing the contents of one or more servers (called masters) to be mirrored on one or more servers (called slaves)
- **Scalability:** By having one or more slave servers, reads can be spread over multiple servers, reducing the load on the master.
 - The most common scenario for a high-read, low-write environment is to have one master, where all the writes occur, replicating to multiple slaves, which handle most of the reads.
- **Backup assistance:** Backups can more easily be run if a server is not actively changing the data.
 - A common scenario is to replicate the data to slave, which is then disconnected from the master with the data in a stable state. Backup is then performed from this server.



Replication and Binary Log

- The main mechanism used in replication is the binary log.
 - All updates to the database (data manipulation and data definition) are written into the binary log as binlog events.
 - The binary log contains a record of all changes to the databases, both data and structure, as well as how long each statement took to execute
 - It consists of a set of binary log files and an index
 - This means that statements such as CREATE, ALTER, INSERT, UPDATE and DELETE will be logged, but statements that have no effect on the data, such as SELECT and SHOW, will not be logged
- Slaves read the binary log from each master in order to access the data to replicate.

Database Replication

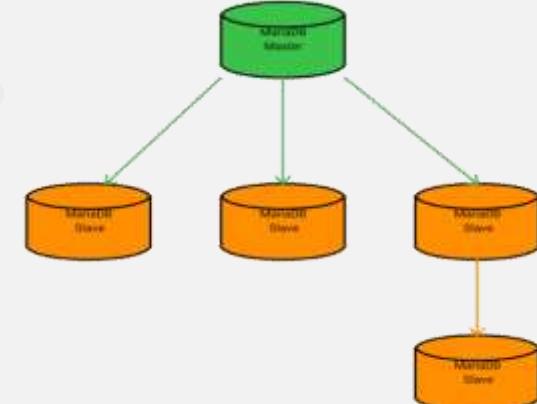
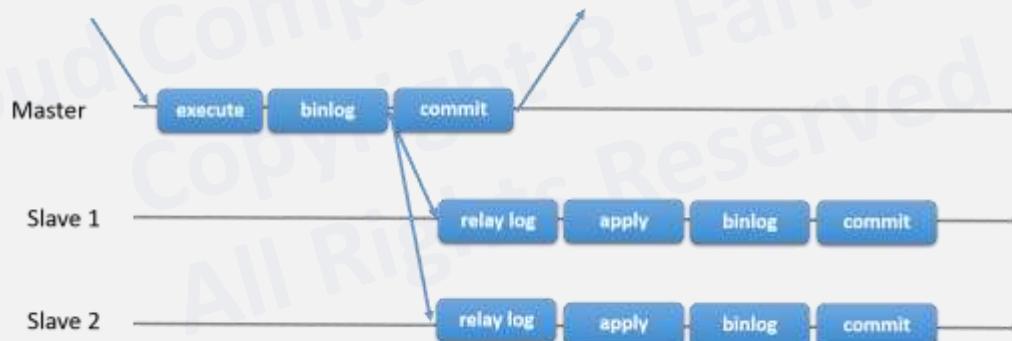
- A relay log is created on the slave server, using the same format as the binary log, and this is used to perform the replication
 - Old relay log files are removed when no longer needed
- A slave server keeps track of the position in the master's binlog of the last event applied on the slave
- This allows the slave server to re-connect and resume from where it left off after replication has been temporarily stopped
- It also allows a slave to disconnect, be cloned and then have the new slave resume replication from the same master
- There will be a measurable delay between the master and the replica. The data on the replica eventually becomes consistent with the data on the master
 - Use this feature for workloads that can accommodate this delay

Replication Steps

1. Replication events are read from the master by the IO thread and queued in the relay log
 2. Replication events are fetched one at a time by the SQL thread from the relay log
 3. Each event is applied on the slave to replicate all changes done on the master
-
- Replication is essentially asynchronous
 - The third step can optionally be performed by a pool of separate replication worker threads
 - **In-order** executes transactions in parallel, but orders the commit step of the transactions to happen in the exact same order as on the master
 - Transactions are only executed in parallel to the extent that this can be automatically verified
 - **Out-of-order** can execute and commit transactions in parallel
 - The application must be tolerant to seeing updates occur in different
 - Only when explicitly enabled by the application

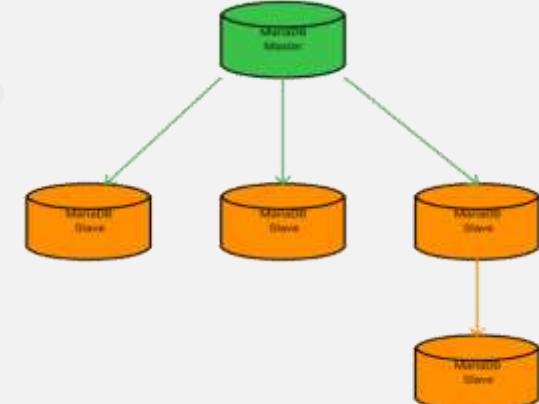
Asynchronous Replication

- A.k.a Standard replication
- Provides infinite read scale out
 - Most websites fit into this category, where users are browsing the website, reading articles, posts, or viewing products.
 - Updates only occur during session management, or when making a purchase or adding a comment/message to a forum.
- Provides high-availability by upgrading slave to master
- slaves read-only to ensure that no one accidentally updates them
- Eventual Consistency



Semi-synchronous Replication

- Semi Synchronous Replication
- Better than eventual consistency
- The master waits for at least one ACK
 - Slower commits
- If no ACK received and timeout, master reverts to asynchronous
 - When at least one ACK is finally received, master goes back to semi-synchronous

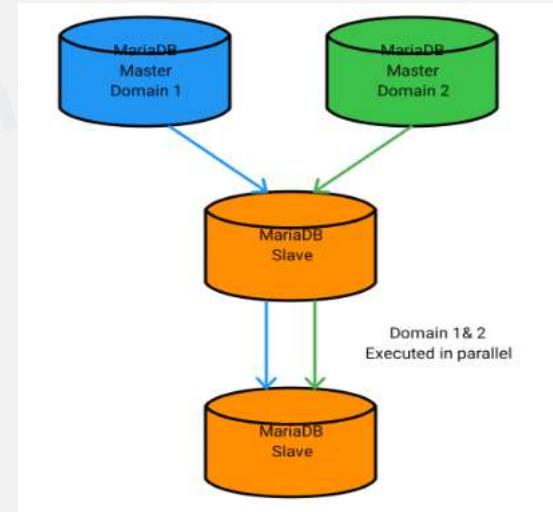


GTIDs-based Replication

- MySQL newer method based on global transaction identifiers (GTIDs)
- Transactional and therefore does not require working with log files or positions within these files
 - May simplify common replication tasks
- Replication using GTIDs guarantees consistency between master and slave as long as all transactions committed on the master have also been applied on the slave

Multi-Source Replication

- Multi-source replication means that one server has many masters from which it replicates
- Allows you to combine data from different sources
- Different domains executed independently in parallel on all slaves



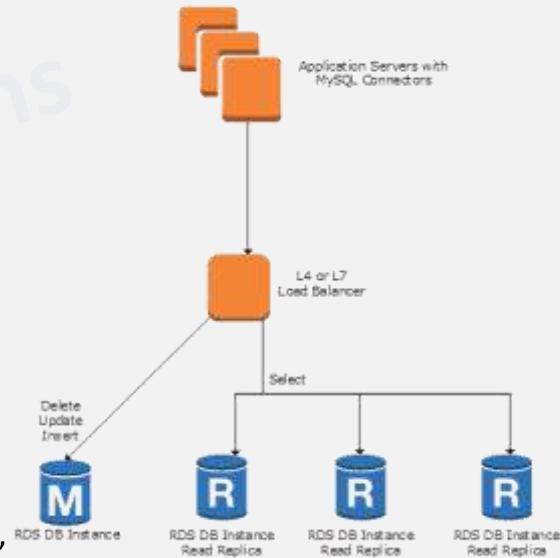
AWS RDS Horizontal Scaling

- Scaling beyond the compute or I/O capacity of a single DB instance for read-heavy database workloads
- Amazon RDS uses the MariaDB, MySQL, Oracle, PostgreSQL, and Microsoft SQL Server DB engines' built-in replication functionality to create a special type of DB instance called a read replica from a source DB instance
 - Up to five read replicas from one DB instance for MariaDB, MySQL
 - Similar limit of 5 replicas in Azure
 - Aurora allows 15 read replicas
 - specify an existing DB instance as the source
 - Amazon RDS takes a snapshot of the source instance and creates a read-only instance from the snapshot
 - Amazon RDS then uses the asynchronous replication method for the DB engine to update the read replica whenever there is a change to the source DB instance
- Updates to the source DB instance are **asynchronously** copied to the read replica
- If the read replica resides in a different AWS Region than its source DB instance, Amazon RDS sets up a secure communications channel between the source and the read replica
 - Amazon RDS establishes any AWS security configurations needed to enable the secure channel, such as adding security group entries
- Replicas can be “promoted” to full databases
 - They will reboot first



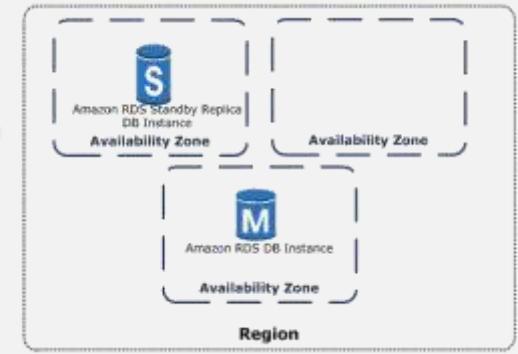
Load Balancing between RDS replicas

- Application-level load balancing
 - Different DNS record-sets using Route 53
- MySQL Connectors
 - If using the native MySQL driver, there are MySQL Connectors that allow read/write splitting and read-only endpoint load balancing without a major change in the application
- ELB does not support multiple RDS instances
- Level 4 proxy solutions
 - HAProxy: configure HAProxy to listen on one port for read queries and another port for write queries
- Level 7 proxy solutions
 - more sophisticated capability of understanding how to properly perform the read/write splits on multi-statements than a MySQL Connector does
 - This solution handles the scaling issues in a distributed database environment, so you don't have to handle scaling on the application layer, resulting in little or no change to the application itself
 - Several open-source solutions (such as MaxScale, ProxySQL, and MySQL Proxy) and also commercial solutions, some of which can be found in the AWS Marketplace



High Availability (Multi-AZ) for Amazon RDS

- Amazon RDS uses several different technologies to provide failover support
 - Multi-AZ deployments for MariaDB, MySQL, Oracle, and PostgreSQL DB instances use Amazon's failover technology
 - SQL Server DB instances use SQL Server Database Mirroring (DBM) or Always On Availability Groups (AGs)
- The high-availability feature is not a scaling solution for read-only scenarios

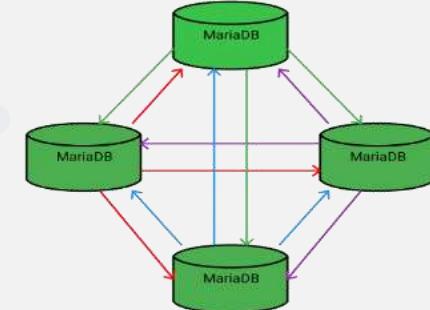


Read replicas, Multi-AZ deployments, and multi-region deployments (Amazon AWS)

Multi-AZ deploymentss	Multi-Region deployments	Read replicas
Main purpose is high availability	Main purpose is disaster recovery and local performance	Main purpose is scalability
Non-Aurora: synchronous replication; Aurora: asynchronous replication	Asynchronous replication	Asynchronous replication
Non-Aurora: only the primary instance is active; Aurora: all instances are active	All regions are accessible and can be used for reads	All read replicas are accessible and can be used for readscaling
Non-Aurora: automated backups are taken from standby; Aurora: automated backups are taken from shared storage layer	Automated backups can be taken in each region	No backups configured by default
Always span at least two Availability Zones within a single region	Each region can have a Multi-AZ deployment	Can be within an Availability Zone, Cross-AZ, or Cross-Region
Non-Aurora: database engine version upgrades happen on primary; Aurora: all instances are updated together	Non-Aurora: database engine version upgrade is independent in each region; Aurora: all instances are updated together	Non-Aurora: database engine version upgrade is independent from source instance; Aurora: all instances are updated together
Automatic failover to standby (non-Aurora) or read replica (Aurora) when a problem is detected	Aurora allows promotion of a secondary region to be the master	Can be manually promoted to a standalone database instance (non-Aurora) or to be the primary instance (Aurora)

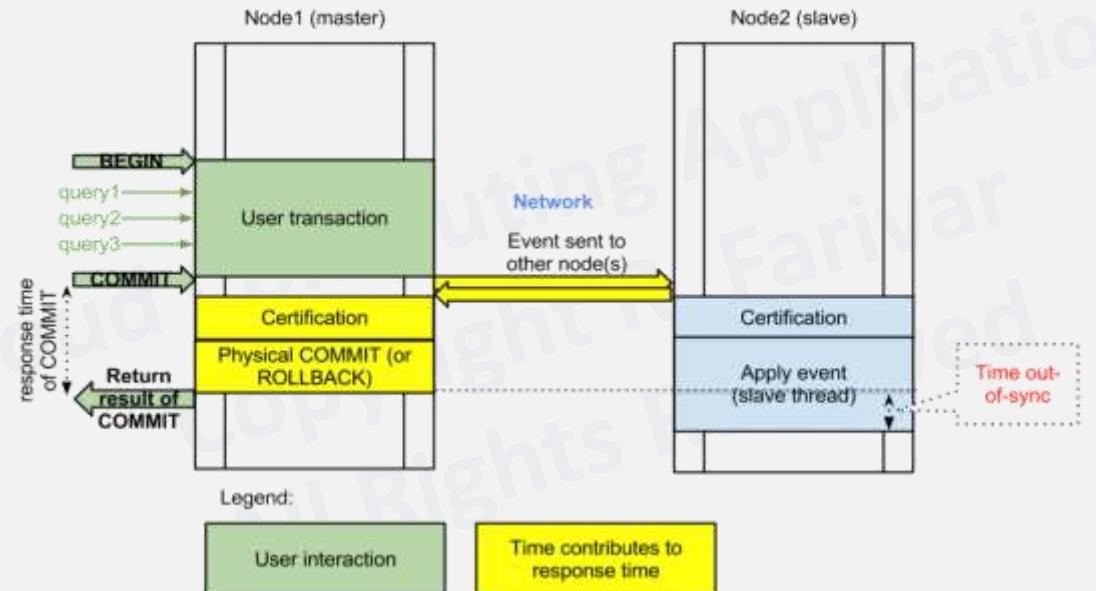
Multi-Master Cluster

- Galera (MySQL, MariaDB)
- Synchronous replication
- Active-active multi-master topology
- Read and write to any cluster node
- Automatic membership control, failed nodes drop from the cluster
- Automatic node joining
- True parallel replication, on row level
- Direct client connections, native MariaDB look & feel



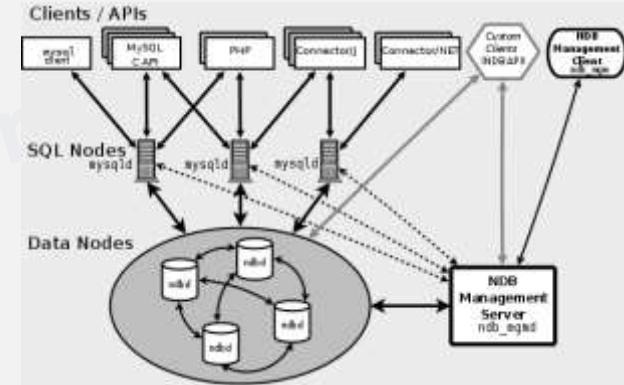
Galera Transaction Commit Flow

- Certification Based Replication
- Virtually Synchronous



MySQL NDB Cluster

- Network DataBase Engine
 - Replaces InnoDB
 - Separation of Compute and Data
 - SQL Nodes
 - Data Nodes
 - Shared Nothing Architecture





CLOUD COMPUTING APPLICATIONS

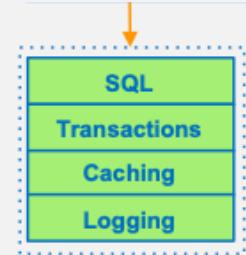
Cloud Databases – Amazon Aurora
Prof. Reza Farivar

Relational Database on the Cloud

- Traditional RDBMS rely on B+Trees, replication, etc. to optimize usage on one or a few servers
- Cloud brings many new things to the table
 - Backend storage
 - Network
 - Worldwide Scalability
- How can we optimize RDBMS for the cloud?
- First step: separate storage layer from the transactional logic
 - Decoupling storage from compute
- Deuteronomy
 - Transaction Component (TC) provides concurrency control and recovery
 - Data Component (DC) provides access methods on top of LLAMA, a latch-free log-structured cache and storage manager.
- Aurora, CosmosDB both inspired by Deuteronomy

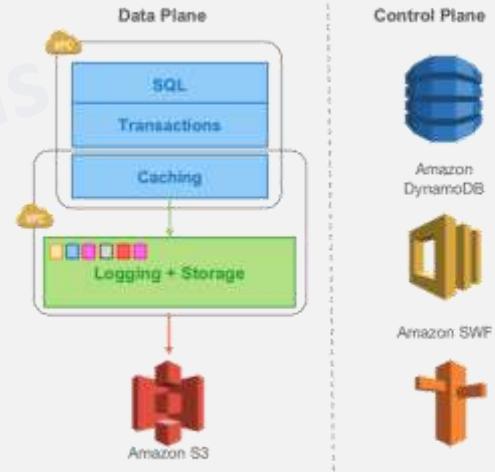
Amazon AWS Aurora

- Optimized DB engine, built from MySQL (and later PostgreSQL), with a distributed storage layer
- API compatible with MySQL or Postgres
 - i.e. you can use an existing application
- Separate storage and compute
 - Query processing, transactions, concurrency, buffer cache, and access
 - Logging, storage, and recovery that are implemented as a scale out service.
- Move caching and logging layers into a purpose-built, scale-out, self-healing, multitenant, database-optimized storage service



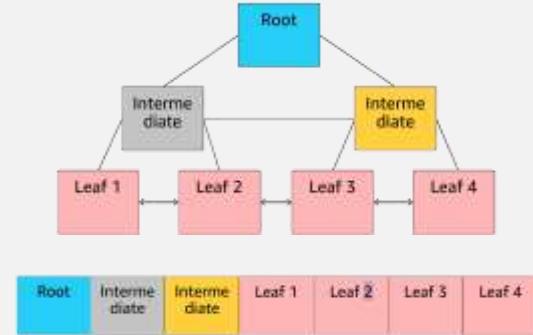
Amazon AWS Aurora

- Each instance still includes most of the components of a traditional kernel (query processor, transactions, locking, buffer cache, access methods and undo management)
- Several functions (redo logging, durable storage, crash recovery, and backup/restore) are off-loaded to the storage service



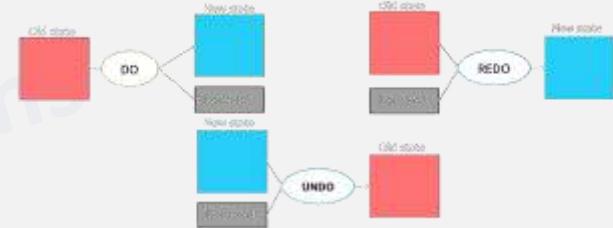
Redo Logging

- Traditional relational databases organize data in *pages* (e.g. 16KB), and as pages are modified, they must be periodically flushed to disk
 - B+ Tree
- For resilience against failures and maintenance of ACID semantics, page modifications are also recorded in *do-redo-undo log records*, which are written to disk in a continuous stream.
- rife with inefficiencies.
 - E.g. a single logical database write turns into multiple (up to five) physical disk writes, resulting in performance problems.
 - Write Amplification
 - combat the write amplification problem by reducing the frequency of page flushes
 - This in turn worsens the problem of crash recovery duration



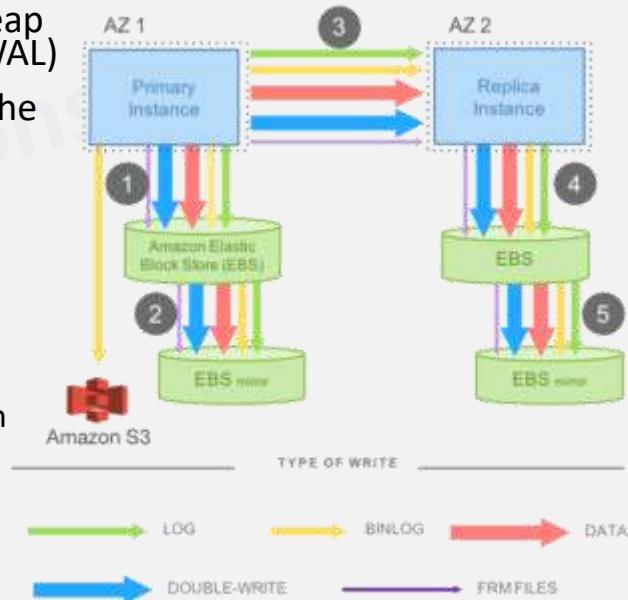
Redo Logging

- Traditional relational databases organize data in *pages* (e.g. 16KB), and as pages are modified, they must be periodically flushed to disk
 - B+ Tree
- For resilience against failures and maintenance of ACID semantics, page modifications are also recorded in *do-redo-undo log records*, which are written to disk in a continuous stream
 - redo log record : difference between the after and the before-image of a page
- rife with inefficiencies.
 - E.g. a single logical database write turns into multiple (up to five) physical disk writes, resulting in performance problems.
 - Write Amplification
 - Combat the write amplification problem by reducing the frequency of page flushes
 - This in turn worsens the problem of crash recovery duration



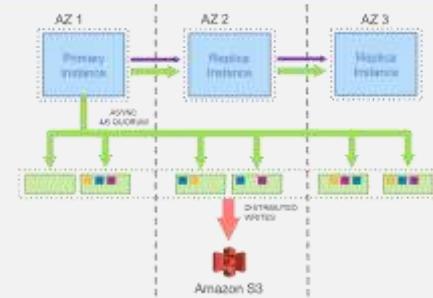
Burden of Amplified Writes

- A system like MySQL writes data pages to objects it exposes (e.g., heap files, b-trees etc.) as well as redo log records to a write-ahead log (WAL)
- The writes made to the primary EBS volume are synchronized with the standby EBS volume using software mirroring
- Data needed to be written in step 1:
 - redo log (typically a few bytes, transaction commit requires the log to be written)
 - binary (statement) log that is archived to Amazon S3 to support point-in-time restores
 - modified data pages (the data page write may be deferred, e.g. 16KB)
 - a second temporary write of the data page (double-write) to prevent torn pages (e.g. 16KB)
 - metadata (FRM) files
- Steps 1, 3, and 5 are sequential and synchronous
 - Latency is additive because many writes are sequential
 - 4/4 write quorum requirement and is vulnerable to failures and outlier performance
- Different writes representing the same information in multiple ways



Log is the database

- In Amazon Aurora, the log is the database
- Database instances write redo log records to the distributed storage layer, and the storage takes care of constructing page images from log records on demands from the database
 - Write performance is improved due to the elimination of write amplification and the use of a scale-out storage fleet
 - 5x write IOPS on the SysBench benchmark compared to Amazon RDS for MySQL running on similar hardware
 - Database crash recovery time is cut down, since a database instance no longer has to perform a redo log stream replay
 - From 27 seconds down to 7 seconds according to one benchmark



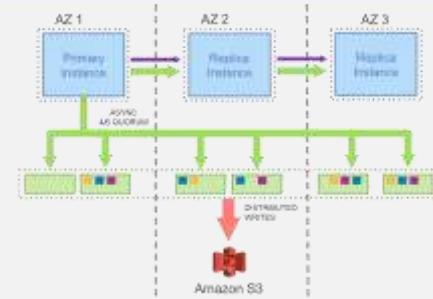
Aurora Replication and Quorum

- Everything fails all the time
 - The traditional approaches of blocking I/O processing until a failover can be carried out—and operating in "degraded mode" until recovery —are problematic at scale
 - in a large system, the probability of operating in degraded mode approaches 1
- Aurora uses quorums to combat the problems of component failures and performance degradation
 - Write to as many replicas as appropriate to ensure that a quorum read always finds the latest data
- Goal is Availability Zone+1: tolerate a loss of a zone plus one more failure without any data durability loss, and with a minimal impact on data availability
 - 4/6 quorum
 - For each logical log write, issue six physical replica writes
 - Write operation successful when four of those writes complete
 - Instances only write redo log records to storage
 - Typically 10s to 100s of bytes, makes a 4/6 write quorum possible without overloading the network
- If a zone goes down and an additional failure occurs, can still achieve read quorum (3/6), and then quickly regain the ability to write by doing a *fast repair*



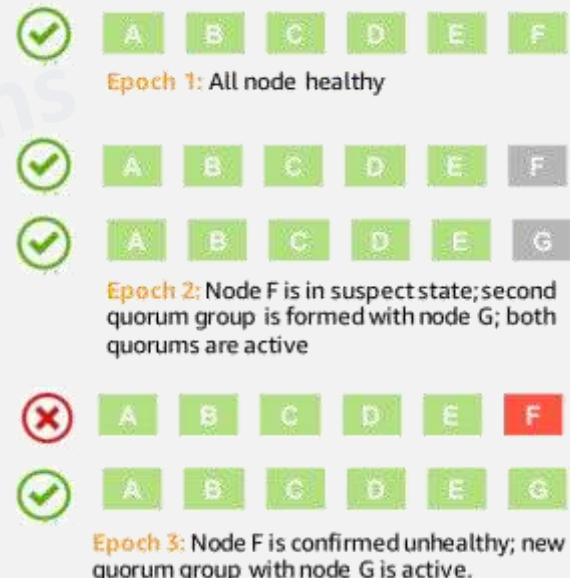
Offloading Redo Processing to Storage

- In Aurora, the only writes that cross the network are redo log records
 - No pages are ever written from the database tier, not for background writes, not for checkpointing, and not for cache eviction
- log applicator is pushed to the storage tier to generate database pages in background or on demand
 - Generating each page from the complete chain of its modifications from the beginning of time is prohibitively expensive
 - Continually materialize database pages in the background to avoid regenerating them from scratch on demand every time
- The storage service can scale out I/Os in an embarrassingly parallel fashion without impacting write throughput of the database engine
- primary only writes log records to the storage service and streams those log records as well as metadata updates to the replica instances
- database engine waits for acknowledgements from 4 out of 6 replicas in order to satisfy the write quorum



Aurora Fast Repair

- Amazon Aurora approach to replication: based on sharding and scale-out architecture
- An Aurora database volume is logically divided into 10-GiB logical units (*protection groups*), and each protection group is replicated six ways into physical units (*segments*)
- When a failure takes out a segment, the repair of a single protection group only requires moving ~10 GiB of data, which is done in seconds.
- When multiple protection groups must be repaired, the entire storage fleet participates in the repair process.
 - Massive bandwidth to complete the entire batch of repairs
- A zone loss followed by another component failure → Aurora may lose write quorum for a few seconds for a given protection group
 - Recovery is quick

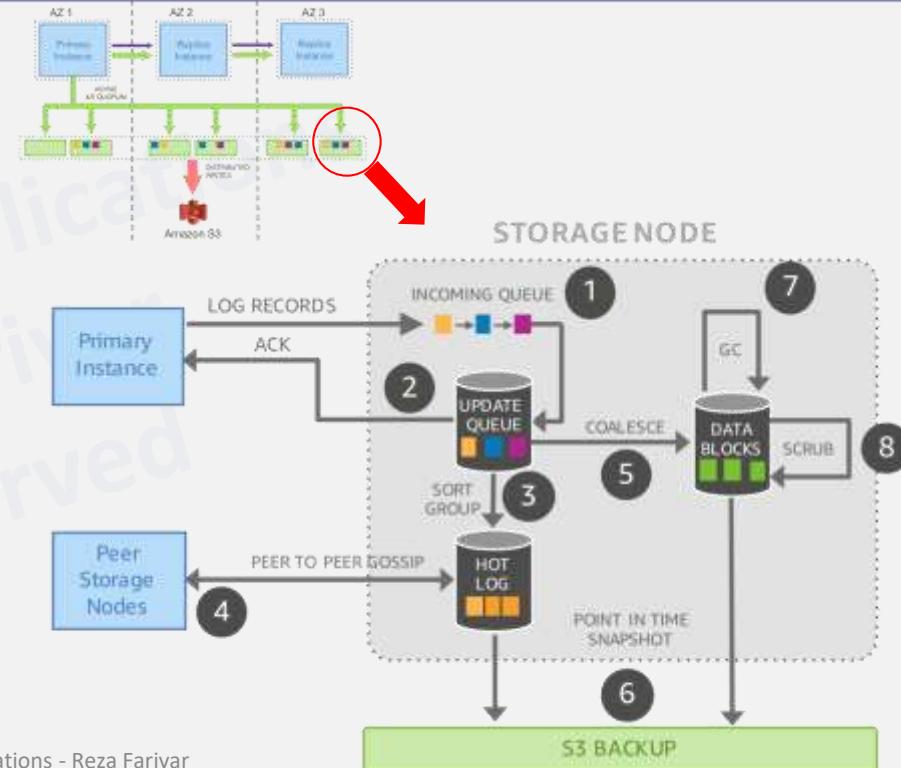


Quorum Reads

- A quorum read is expensive, and is best avoided
- We do not need to perform a quorum read on routine page reads
 - It always knows where to obtain an up-to-date copy of a page
 - The client-side Aurora storage driver tracks which writes were successful for which segments
 - The driver tracks read latencies, and always tries to read from the storage node that has demonstrated the lowest latency in the past
- The only scenario when a quorum read is needed is during recovery on a database instance restart

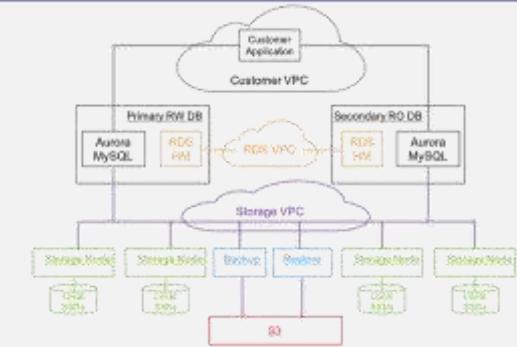
Amazon Aurora Storage Nodes

1. Receive log records and add to in-memory queue
 2. persist record on disk and acknowledge, ACK to the database
 3. Organize records and identify gaps in log since some batches may be lost
 4. Gossip with peers to fill in holes
 5. Coalesce log records into new page versions
 6. Periodically stage log and new page versions to S3
 7. Periodically garbage-collect old versions
 8. Periodically validate CRC codes on blocks
- Each of the steps above are asynchronous
 - Only steps (1) and (2) are in the foreground path potentially impacting latency



Database Engine Implementation

- The database engine is a fork of “community” MySQL/InnoDB and diverges primarily in how InnoDB reads and writes data to disk
 - In community InnoDB, a write operation results in data being modified in buffer pages, and the associated redo log records written to buffers of the WAL in LSN order
 - On transaction commit, the WAL protocol requires only that the redo log records of the transaction are durably written to disk
 - The actual modified buffer pages are also written to disk eventually through a double-write technique to avoid partial page writes
 - These page writes take place in the background, or during eviction from the cache, or while taking a checkpoint
- In addition to the IO Subsystem, InnoDB also includes the transaction subsystem, the lock manager, a B+-Tree implementation and the associated notion of a “mini transaction” (MTR).
 - An MTR is a construct only used inside InnoDB and models groups of operations that must be executed atomically (e.g., split/merge of B+-Tree pages).
- Concurrency control is implemented entirely in the database engine without impacting the storage service

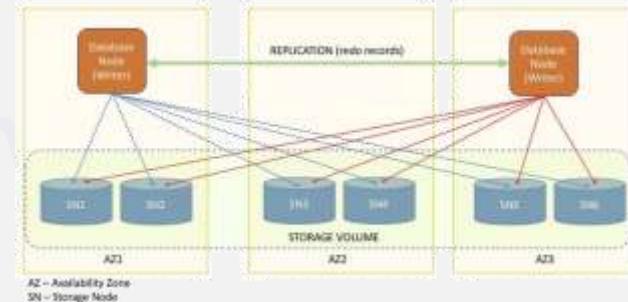


Aurora and Consensus

- Aurora leverages only quorum I/Os, locally observable state, and monotonically increasing log ordering to provide high performance, non-blocking, fault-tolerant I/O, commits, and membership changes
- Aurora is able to avoid much of the work of consensus by recognizing that, during normal forward processing of a system, there are local oases of consistency
- Using backward chaining of redo records, a storage node can tell if it is missing data and gossip with its peers to fill in gaps
- Using the advancement of segment chains, a database instance can determine whether it can advance durable points and reply to clients requesting commits
- The use of monotonically increasing consistency points – SCLs, PGCLs, PGMRPLs, VCLs, and VDLs – ensures the representation of consistency points is compact and comparable
 - These may seem like complex concepts but are just the extension of familiar database notions of LSNs and SCNs.
- The key invariant is that the log only ever marches forward.

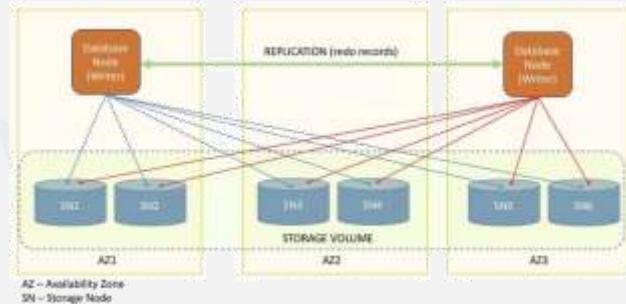
Aurora Multi-Master

- For high availability and ACID transactions across a cluster of database nodes with configurable read after write consistency
- With single-master Aurora, a failure of the single writer node requires the promotion of a read replica to be the new writer
- In the case of Aurora Multi-Master, the failure of a writer node merely requires the application using the writer to open connections to another writer
- When designing for high availability, make sure that you are not overloading writers
- Conflicts arise when concurrent transactions or writes executing on different writer nodes attempt to modify the same set of pages



Aurora Multi-Master Replication and Quorum

1. The application layer starts a write transaction
2. For cross-cluster consistency: The writer node proposes the change to all six storage nodes
3. Each storage node checks if the proposed change conflicts with a change in flight or a previously committed change and either confirms the change or rejects it
 - Each storage node compares the LSN (think of this as a page version) of the page submitted by the writer node with the LSN of the page on the node
 - It approves the change if they are the same and rejects the change with a conflict if the storage node contains a more recent version of the page
4. If the writer node that proposed the change receives a positive confirmation from a quorum of storage nodes:
 1. First, it commits the change in the storage layer, causing each storage node to commit the change
 2. It then replicates the change records to every other writer node in the cluster using a low latency, peer-to-peer replication protocol
5. The peer writer nodes, upon receiving the change, apply the change to their in-memory cache (buffer pool)
6. If the writer node that proposed the change does not receive a positive confirmation from a quorum of storage nodes, it cancels the entire transaction and raises an error to the application layer (The application can then retry the transaction)
7. Upon successfully committing changes to the storage layer, writer nodes replicate the redo change records to peer writer nodes for buffer pool refresh in the peer node



Aurora vs. RDS

- RDS offers a greater range of database engines and versions than Aurora RDS
- Aurora RDS offers superior performance to RDS due to the unique storage subsystem
- Aurora RDS offers superior scalability to RDS due to the unique storage subsystem
- The pricing models differ slightly between RDS and Aurora RDS, but Aurora RDS is generally a bit more expensive to implement for the same database workload
- Aurora RDS offers superior high availability to RDS due to the unique storage subsystem
- According to AWS, Aurora offers five times the throughput of standard MySQL, performance on-par with commercial databases, but at one-tenth the cost
 - It should be noted that these numbers are AWS marketing claims.
 - House of Brick has found the cost claims to be roughly correct when compared with Oracle Enterprise Edition deployments, but the performance advantage of Aurora in real-world scenarios is closer to 30%

<http://houseofbrick.com/aws-rds-mysql-vs-aurora-mysql/>



CLOUD COMPUTING APPLICATIONS

Cloud Databases – Google Cloud Spanner
Prof. Reza Farivar

Google Cloud Spanner

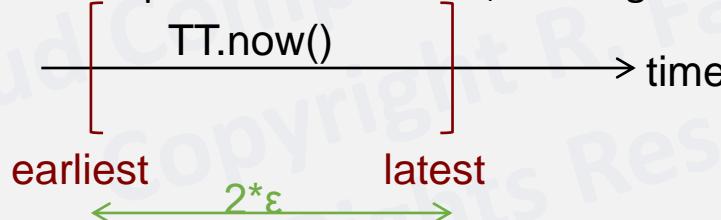
- Spanner is a distributed data layer that uses optimized sharded Paxos to guarantee consistency even in a system that spans multiple geographic regions
 - Query API is SQL (i.e. SELECT)
 - Insert and updates are done through a specialized GRPC interface
- Two-phase commit to achieve serializability
- TrueTime for external consistency, consistent reads without locking, and consistent snapshots
 - *External > Strong > Weak*

Spanner and CAP

- Is Spanner C+A+P?
 - No, It is CP
 - during (some) partitions, Spanner chooses C and forfeits A
- Availability is in the 5 nines range
 - Is this acceptable to your application?
 - Effectively CA
- Spanner uses the Paxos algorithm as part of its operation to shard (partition) data across hundreds of servers
- 2PC known as the anti-availability protocol
 - because all members must be up for it to work
 - In Spanner, each member is a Paxos group
 - ensures each 2PC “member” is highly available even if some of its Paxos participants are down
- Cloud Spanner provides stale reads, which offer similar performance benefits as eventual consistency but with much stronger consistency guarantees
 - A stale read returns data from an “old” timestamp, which cannot block writes because old versions of data are immutable

TrueTime

- Heavy use of hardware-assisted clock synchronization using GPS clocks and atomic clocks to ensure global consistency
 - avoid communication in a distributed system
 - GPS and Atomic clock have different failure modes
- “Global wall-clock time” with bounded uncertainty
 - ϵ is worst-case clock divergence
 - Timestamps become intervals, not single values



- Consider event e_{now} which invoked $tt = TT.now()$:
 - Guarantee: $tt.earliest \leq t_{abs}(e_{now}) \leq tt.latest$

Method	Returns
$TT.now()$	$TTinterval: [earliest, latest]$
$TT.after(t)$	true if t has definitely passed
$TT.before(t)$	true if t has definitely not arrived

Spanner

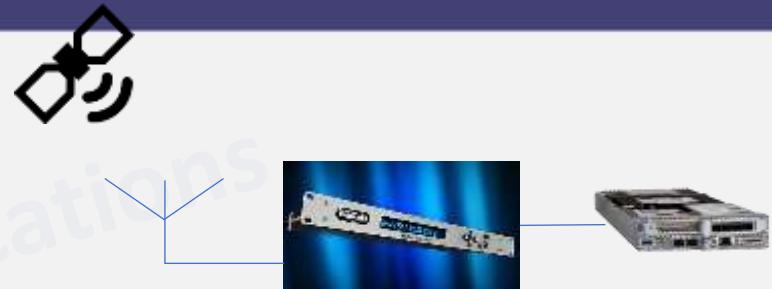
- TrueTime exposes clock uncertainty
 - Commit wait ensures transactions end after their commit time
 - Read at TT.now.latest()
- Reads dominant, make them lock-free
 - Read-Only Transaction
 - A replica can satisfy a read at a timestamp t if $t \leq t_{safe}$.
 - Snapshot Read, client-provided timestamp
 - read at a particular time in the past
 - Snapshot Read, client-provided bound
- Read-Write Transaction less common
 - Pessimistic, use 2 phase locking
- Globally-distributed database
 - 2PL w/ 2PC over Paxos!

$$t_{safe} = \min(t_{safe}^{Paxos}, t_{safe}^{TM})$$

Paxos State Machine Transaction Manager
Safe Time Safe Time

HW-based Time Synchronization

- NTP
 - Software time synchronization with one server
 - Stratum 2
 - Network delays
 - 1/2 to 100 ms accuracy
- Google Spanner
- Microsoft Azure now offers GPS clock synchronization
 - VMICTimeSync provider
 - Precision Time Protocol
 - Stratum 1 devices
 - I.e. direct connection, not through shared network, to a reference time server (Stratum 0)
 - 10 microseconds accuracy
 - For reference, GPS accuracy < 1 us, 95% of the time ≤40 nanoseconds
- Amazon Time Sync Service
 - Chrony vs. NTP





CLOUD COMPUTING APPLICATIONS

Cloud Databases – Microsoft Azure CosmosDB
Prof. Reza Farivar

Azure CosmosDB

- Globally distributed, multi-model database
 - Wire compatible with Cassandra
 - Table API
 - 5 types of consistency levels
 - MongoDB API
 - Etcd API
 - etcd is a consistent distributed key value storage
 - Compare with Zookeeper
 - Backend of Kubernetes
 - Gremlin API
- Replicated State Machine (RSM) for concurrency
 - Specific algorithm not published
 - *RAFT is also a state machine consensus algorithm*

Azure CosmosDB

- Write-optimized, resource-governed and schema-agnostic database engine
 - It automatically indexes everything it ingests
 - Internally based on a document store model
 - All JSON documents are a tree
 - Make an index for each path of the tree
 - **Synchronously** makes the index durable and highly available before acknowledging the client's updates while maintaining low latency guarantees
- BW-Tree indexing
 - Log Structure Record
 - Latch free updates
 - Atom-record-sequence (ARS) system
- Uses Lamport's TLA+ specification language to describe SLAs at each consistency level

CosmosDB Consistency Models

- **Strong:** With strong consistency, you are guaranteed to always read the latest version of an item similar to read committed isolation in SQL Server. You can only ever see data which is durably committed. Strong consistency is scoped to a single region.
- **Bounded-staleness:** In bounded-staleness consistency read will lag behind writes and guarantees global order and not scoped to a single region. When configuring bounded-staleness consistency you need to specify the maximum lag by:
 - Operations: For a single region the maximum operations lag must be between 10 and 1,000,000, and for the multi region, it will be between 100,000 and 1,000,000.
 - Time: The maximum lag must be between 5 seconds and 1 day for either single or multi-regions.
- **Session:** This is the most popular consistency level, since it provides consistency guarantees but also has better throughput.
- **Consistent Prefix:** A global order is preserved, and prefix order is guaranteed. A user will never see writes in a different order than that in which it was written.
- **Eventual:** Basically, this is like asynchronous synchronization. It guarantees that all changes will be replicated eventually, and as such, it also has the lowest latency because it does not need to wait on any commits.



Figure 4. Multiple well-defined consistency choices along the spectrum.

Consistency Level	Guarantees
Strong	Linearizability
Bounded Staleness	Consistent Prefix. Reads lag behind writes by k prefixes or t interval
Session	Consistent Prefix. Monotonic reads, monotonic writes, read-your-writes, write-follows-reads
Consistent Prefix	Updates returned are some prefix of all the updates, with no gaps
Eventual	Out of order reads

Azure CosmosDB Implementation

- Cosmos DB service is deployed on several replicated shared-nothing nodes across geographical regions for high-availability, low-latency, and high throughput.
- Some or all of these distributed nodes form a replica set for serving requests on a data shard that contains documents.
- Among the replicas, one of them is elected as a master to perform totally-ordered writes on the data shard.
- Writes are done on the write-quorum (W), a subset of the replica nodes, to ensure that the data is durable.
- Reads are performed on read-quorum (R), a subset of replica nodes, to get the desired consistency levels (Strong, Bounded-staleness, Session, Consistent Prefix, Eventual) as configured by users.
- Data is partitioned at logic level and is replicated at storage layer in terms of physical partitions to achieve desired availability and throughput.
- storage
 - transactional storage engine
 - analytical storage engine
 - storage engines are log-structured and write-optimized



CLOUD COMPUTING APPLICATIONS

Cloud Databases - NoSQL
Prof. Reza Farivar

Key/value Databases

- Key-value databases are optimized for common access patterns, typically to store and retrieve large volumes of data
- These databases deliver quick response times, even in extreme volumes of concurrent requests
- High-traffic web apps, ecommerce systems, and gaming apps
- AWS DynamoDb
- Azure CosmosDB

Wide Column Databases

- Google BigTable
 - Cloud Bigtable is a fully managed, wide-column NoSQL database that offers low latency and replication for high availability
 - This is what HBase was modeled after
- Managed Casandra
 - *Cassandra was modeled after Dynamo (paper)*
 - *DynamoDB was modeled after Casandra*
 - AWS managed Casandra
- Cassandra AMI for any cloud provider

In Memory (Cache) Databases

- In-memory databases are used for applications that require real-time access to data
- By storing data directly in memory, these databases deliver microsecond latency to applications for whom millisecond latency is not enough
- Caching, gaming leaderboards, and real-time analytics
- Common usage pattern: Cache RDS or document databases
- AWS ElastiCashe (Redis / MemCached)
- Azure Cache for Redis
- Google Memorystore (Redis / MemCached)
- IBM Redis

Document Databases

- A document database is designed to store semistructured data as JSON-like documents
 - Makes it easy to store, ***query***, and index JSON data
 - Content management, catalogs, and user profiles
 - Non-relational database service
-
- AWS DocumentDB + (MongoDB compatibility)
 - Azure CosmosDB
 - Google Firestore
 - Targeted for mobile App support
 - IBM Cloudant / IBM MongoDB

AWS DocumentDB

- Managed instance
- Implements MongoDB 3.6 API
- Storage and compute are decoupled, allowing each to scale independently
- Automatically grows the size of storage volume as the database storage needs grow
 - Grows in increments of 10 GB, up to a maximum of 64 TB
- Up to 15 low latency read replicas to increase read capacity
- Replicates six copies of data across three AWS Availability Zones (AZs)
- Access to Amazon DocumentDB clusters must be done through the mongo shell or with MongoDB drivers

Other Types of Cloud Databases

- Graph Databases
 - Covered in a different module
- Time Series Databases
 - AWS Timestream
- Blockchain / Ledgers
 - Immutable and cryptographically verifiable transactions
 - AWS QLDB
- Data Warehouses
 - Covered in a different module
 - Columnar storage
 - AWS Redshift
 - Google BigQuery
 - Azure Synapse (formerly Azure SQL Data Warehouse)



CLOUD COMPUTING APPLICATIONS

Caching as a Universal Concept:
Overview

Prof. Reza Farivar

The need for caching

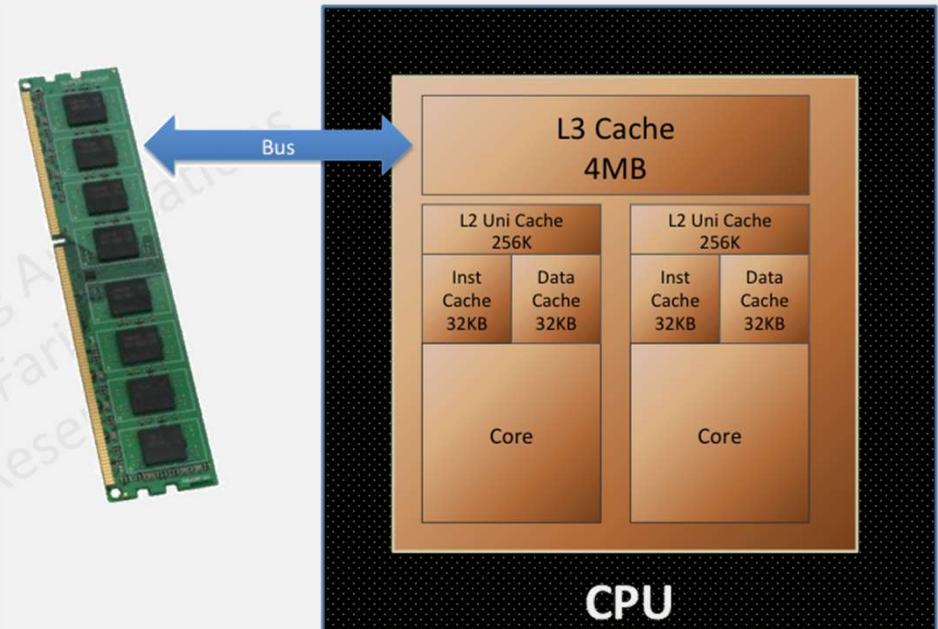
- Success for many websites and web applications relies on speed
 - Users can register a 250-millisecond (1/4 second) difference between competing sites
 - “[*For Impatient Web Users, an Eye Blink Is Just Too Long to Wait*](#)” NYT, 2012
 - For every 100-ms (1/10 second) increase in load time, [sales decrease 1 percent](#)
 - Data that is cached can be delivered much faster
- In-memory Key-value stores can provide sub-millisecond latency
 - querying a database is always slower and more expensive than locating a key in a key-value pair cache

Caching

- Caching is a universal concept
- Based on the principle of locality (aka. locality of reference)
 - Tendency of the “processor” to access the same set of memory locations repetitively over a short period of time
 - Temporal locality vs spatial locality
- Whenever you have “large + slow” source of information and “small + fast” storage technology, you can use the latter to cache the former
- You can see this concept anywhere from CPUs and processors, to operating systems, to large web applications on the cloud

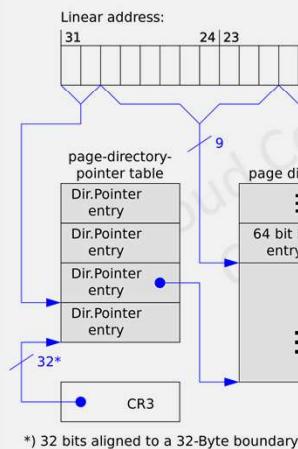
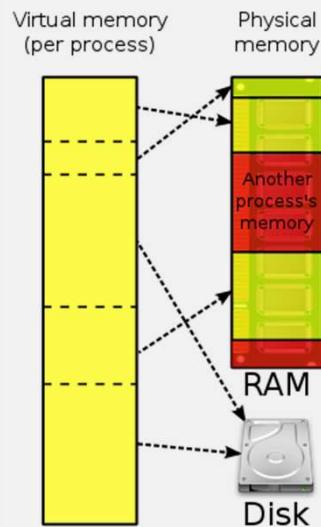
Caching in Processors: Data & Instructions

- Big/Slow RAM memory
- Multiple layers of caching
- Access to data exhibits temporal and spatial locality
 - L1 Data Cache, L2 and L3 Caches
- The program instructions have spatial and temporal locality
 - One instruction after another
 - Loops
- Also branch locality
 - If ... else

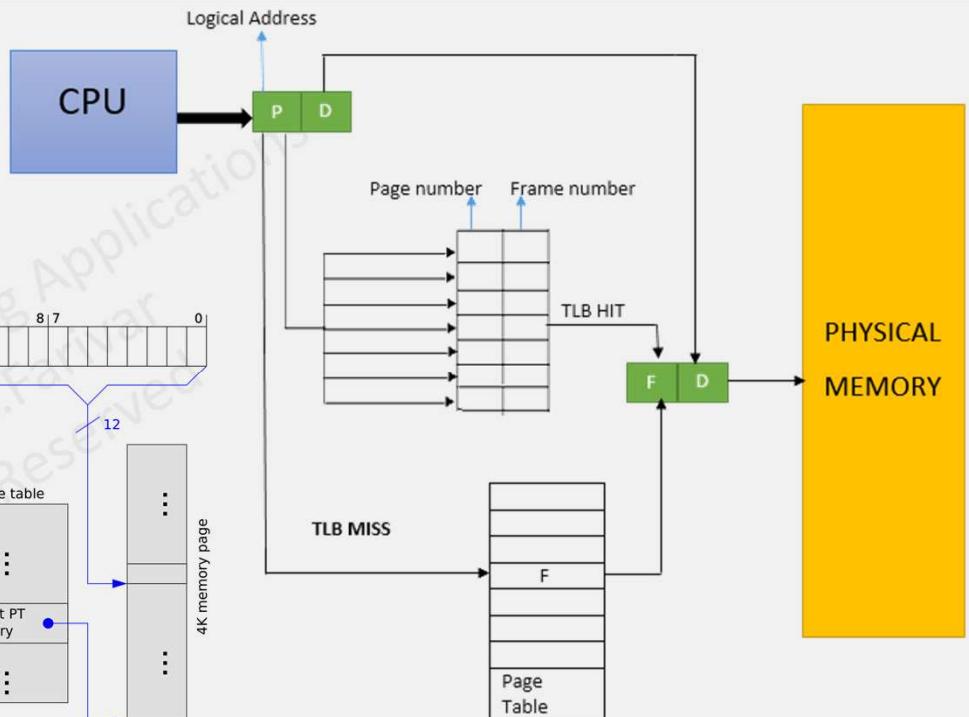


Caching in Processors: Virtual Memory

- Virtual memory address translation
- Every memory access needs a translation
 - Needs a page walk
 - Slow/big source of data
- Translation Lookaside Buffer (TLB)
 - iTLB and dTLB



Cloud Computing Applications - Reza Farivar



Virtual memory and OS-level Page Caching

- Virtual memory:
 - Each process thinks it has $2^{48} = 256$ TB of memory
- Paging
 - computer stores and retrieves data from secondary storage (HDD/SSD) for use in main memory
 - RAM acts as the “cache” for the SSD
 - When a process tries to reference a page not currently present in RAM, the processor treats this invalid memory reference as a **page fault** and transfers control from the program to the operating system
- OS does the following
 - Determine the location of the data on disk
 - Obtain an empty page frame in RAM to use as a container for the data
 - Load the requested data into the available page frame
 - Update the page table to refer to the new page frame
 - Return control to the program, transparently retrying the instruction that caused the page fault

Linux Page Cache

- Linux kernels up to version 2.2 had both a Page Cache as well as a Buffer Cache. As of the 2.4 kernel, these two caches have been combined. Today, there is only one cache, the Page Cache.
- This mechanism also caches files.
- Usually, all physical memory not directly allocated to applications is used by the operating system for the page cache
- So the OS keeps other pages that it may think may be needed in the page cache.
- If Linux needs more memory for normal applications than is currently available, areas of the Page Cache that are no longer in use will be automatically deleted.

```
wfischer@pc:~$ dd if=/dev/zero of=testfile.txt bs=1M count=10
10+0 records in
10+0 records out
10485760 bytes (10 MB) copied, 0.0121043 s, 866 MB/s
wfischer@pc:~$ cat /proc/meminfo | grep Dirty
Dirty:           10260 kB
wfischer@pc:~$ sync
wfischer@pc:~$ cat /proc/meminfo | grep Dirty
Dirty:           0 kB
https://www.thomas-krenn.com/en/wiki/Linux\_Page\_Cache\_Basics
```

Linux VFS Cache

- Dentry Cache
 - A "dentry" in the Linux kernel is the in-memory representation of a directory entry
 - A way of remembering the resolution of a given file or directory name without having to search through the filesystem to find it
 - The dentry cache speeds lookups considerably; keeping dentries for frequently accessed names like /tmp, /dev/null, or /usr/bin/tetris saves a lot of filesystem I/O.
- Inode Cache
 - As the mounted file systems are navigated, their VFS inodes are being continually read and, in some cases, written
 - the Virtual File System maintains an inode cache to speed up accesses to all of the mounted file systems
 - Every time a VFS inode is read from the inode cache the system saves an access to a physical device.

Caching in Distributed Systems

- CDN Caching
- Web Server Caching
 - Reverse Proxies
 - Varnish
 - Web servers can also cache requests, returning responses without having to contact application servers
 - NGINX
- Database Caching
- Application Caching
 - In-memory caches such as Memcached and Redis are key-value stores between your application and your data storage

What to Cache

- There are multiple levels you can cache that fall into two general categories: **database queries and objects**:
 - Row level
 - Query-level
 - Fully-formed serializable objects
 - Fully-rendered HTML

Caching at the database query level

- Whenever you query the database, hash the query as a key and store the result to the cache
- Suffers from expiration issues:
 - Hard to delete a cached result with complex queries
 - If one piece of data changes such as a table cell, you need to delete all cached queries that might include the changed cell

Caching at the object level

- See your data as an object, similar to what you do with your application code. Have your application assemble the dataset from the database into a class instance or a data structure(s):
 - Remove the object from cache if its underlying data has changed
 - Allows for asynchronous processing: workers assemble objects by consuming the latest cached object
- Suggestions of what to cache:
 - User sessions
 - Fully rendered web pages
 - Activity streams
 - User graph data



CLOUD COMPUTING APPLICATIONS

Caching Technical Concepts

Prof. Reza Farivar

Topics

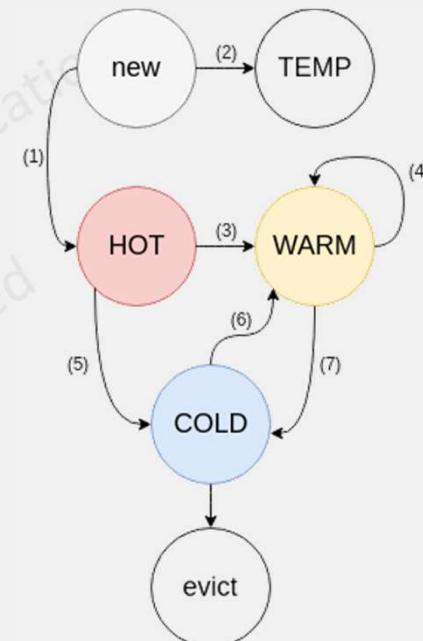
- Cache Replacement Policy
 - LRU, FIFO, etc.
- Cache Writing / Updating Policy
 - Cache Aside, Write-through, Write-back
- Cache Coherence

Reading from Cache

- A client first checks to see if a data piece is available in a cache
- Cache hit: the desired data is in the cache
- Cache miss: the desired data is not in the cache
 - In this case, the client needs to go to the “slow/big” source of data
 - Once data is retrieved, it is also copied in the cache, ready for next accesses (principle of locality)
 - But **where** in the cache should we put this new data?

Cache Replacement Policy

- Also known as:
 - Cache Eviction Policy
 - Cache Invalidation Algorithm
- Least Recently Used (LRU)
 - Replaces the oldest entry in the cache
 - Memcached uses segmented LRU
- Time-aware Least Recently Used (TLRU)
 - TTU: Time To Use
- Least Frequently Used (LFU)
- First in First Out (FIFO)
- Many others
 - LIFO, FILO, MRU, PLRU,



Cache Replacement

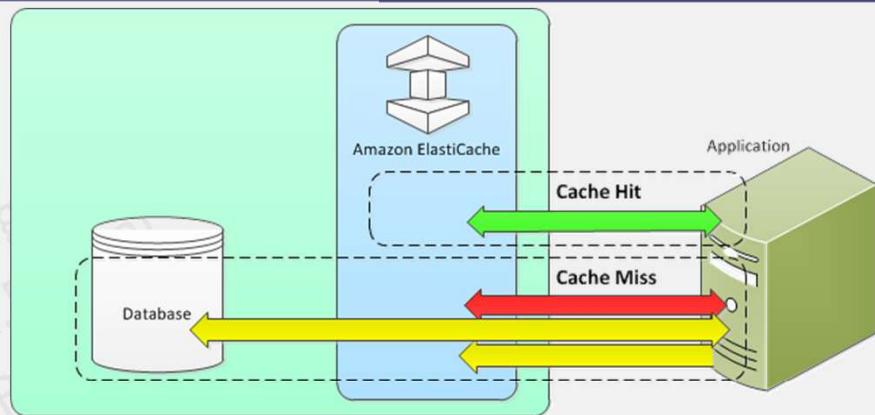
- maintain consistency between caches and the source of truth such as the database through cache invalidation
- Cache invalidation is a difficult problem, there is additional complexity associated with when to update the cache.
- Time to Live (TTL): Expiration time for records in the cache
 - Eventual consistency for coherence problem

Cache Writing/Updating Policies

- Whenever a new piece of data is created and needs to be stored, we also have a question regarding updating the caches
 - Cache Aside
 - Lazy Loading
 - Write-Through
 - Write is done synchronously both to the cache and to the backing store.
 - Write – Back
 - Write-Behind
 - Lazy Writing

Cache Aside (aka lazy loading)

- The **application** is responsible for reading and writing from storage
 - The cache does not interact with storage directly
- The **application** does the following:
 - Look for entry in cache, resulting in a cache miss
 - Load entry from the database
 - Add entry to cache
 - Return entry
- Memcached is usually used in this manner



```
def get_user(self, user_id):
    user = cache.get("user.{0}", user_id)
    if user is None:
        user = db.query("SELECT * FROM users WHERE user_id = {0}", user_id)
        if user is not None:
            key = "user.{0}".format(user_id)
            cache.set(key, json.dumps(user))
    return user
```

Disadvantages of Cache-aside

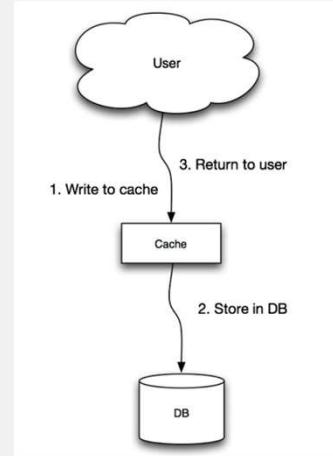
- Each cache miss results in three trips, which can cause a noticeable delay.
- Data can become stale if it is updated in the database. This issue is mitigated by setting a time-to-live (TTL) which forces an update of the cache entry, or by using write-through.
- When a node fails, it is replaced by a new, empty node, increasing latency.

Write-through writing/updating policy

- The application uses the cache as the main data store, reading and writing data to it, while the cache is responsible for reading and writing to the database:
 - Application adds/updates entry in cache
 - Cache synchronously writes entry to data store
 - Return

Application code:

```
set_user(12345, {"foo":"bar"})
```



Cache code:

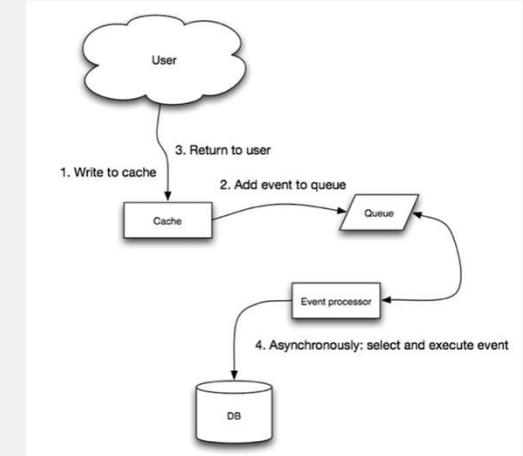
```
def set_user(user_id, values):
    user = db.query("UPDATE Users WHERE id = {0}", user_id, values)
    cache.set(user_id, user)
```

Disadvantages of Write-Through

- Write-through is a slow overall operation due to the write operation, but subsequent reads of just written data are fast
 - Users are generally more tolerant of latency when updating data than reading data.
- Data in the cache is never stale ☺
- When a new node is created due to failure or scaling, the new node will not cache entries until the entry is updated in the database
 - Cache-aside in conjunction with write through can mitigate this issue
- ***Cache Churn:*** Most data written might never be read
→ waste of resource
 - Can be minimized with a TTL

Write-back (Write-Behind) writing/updating policy

- Initially, writing is done only to the cache.
- The write to the backing store is postponed until the modified content is about to be replaced by another cache block.
 - Asynchronously write entry to the data store, improving write performance
- Write-back cache is more complex to implement, since it needs to track which of its locations have been written over, and mark them as *dirty* for later writing to the backing store
 - **Lazy Write:** Data in these dirty locations are written back to the backing store only when they are evicted from the cache
- a read miss in a write-back cache (which requires a block to be replaced by another) will often require two memory accesses to service: one to write the replaced data from the cache back to the store, and then one to retrieve the needed data.



Img source: [Scalability, Availability & Stability Patterns](#)
Jonas Bonér

Cloud Computing Applications - Reza Farivar

11

Write-around

- Writing is only done to the underlying data source
- **Advantage:** Good for not flooding the cache with data that may not subsequently be re-read
- **Disadvantage:** Reading recently written data will result in a cache miss (and so a higher latency) because the data can only be read from the slower backing store
- The write-around policy is good for applications that don't frequently re-read recently written data
 - This will result in lower write latency but higher read latency which is an acceptable trade-off for these scenarios

Write Allocation Policies

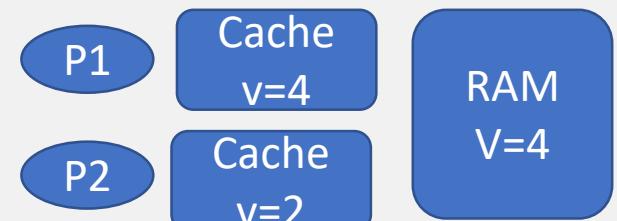
- Since no data is returned to the requester on write operations, a decision needs to be made on write misses, whether or not data would be loaded into the cache. This is defined by these two approaches:
 - *Write allocate* (also called *fetch on write*): data at the missed-write location is loaded to cache, followed by a write-hit operation. In this approach, write misses are similar to read misses.
 - *No-write allocate* (also called *write-no-allocate* or *write around*): data at the missed-write location is not loaded to cache, and is written directly to the backing store. In this approach, data is loaded into the cache on read misses only.

Write Allocation Policies

- A write-back cache uses write allocate, hoping for subsequent writes (or even reads) to the same location, which is now cached.
- A write-through cache uses no-write allocate. Here, subsequent writes have no advantage, since they still need to be written directly to the backing store.

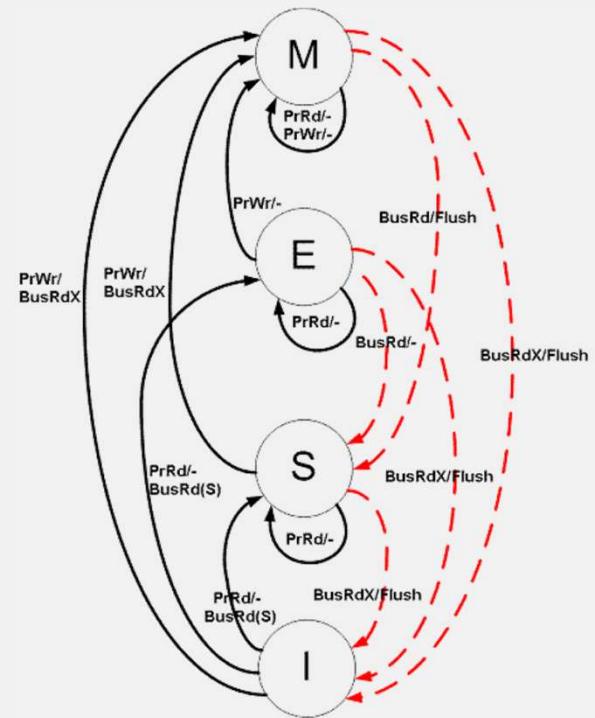
Cache Coherency

- Note: do not confuse with Consistency!
- Whenever cache is distributed, **with the same data in more than one place**, we have the coherency problem
 - Write Propagation: Changes to the data in any cache must be propagated to other copies (of that value) in the peer caches
 - Transaction Serialization: Reads/Writes to a single memory location must be seen by all processors in the same order
- Coherence Protocols
 - **Write-invalidate:** When a write operation is observed to a location that a cache has a copy of, the cache controller **invalidates** its own copy
 - MESI protocol
 - **Write-Update:** When a write operation is observed to a location that a cache has a copy of, the cache controller **updates** its own copy



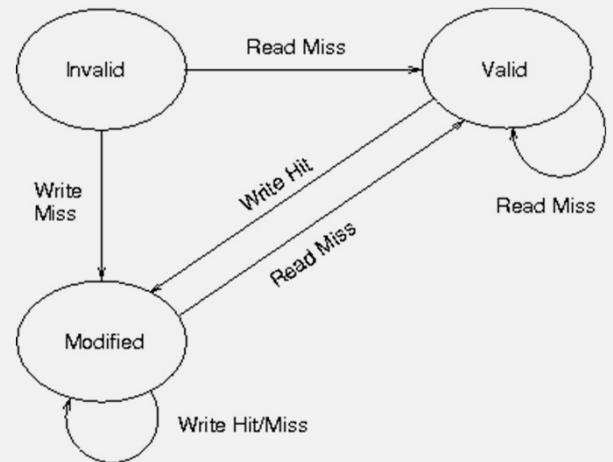
MESI protocol

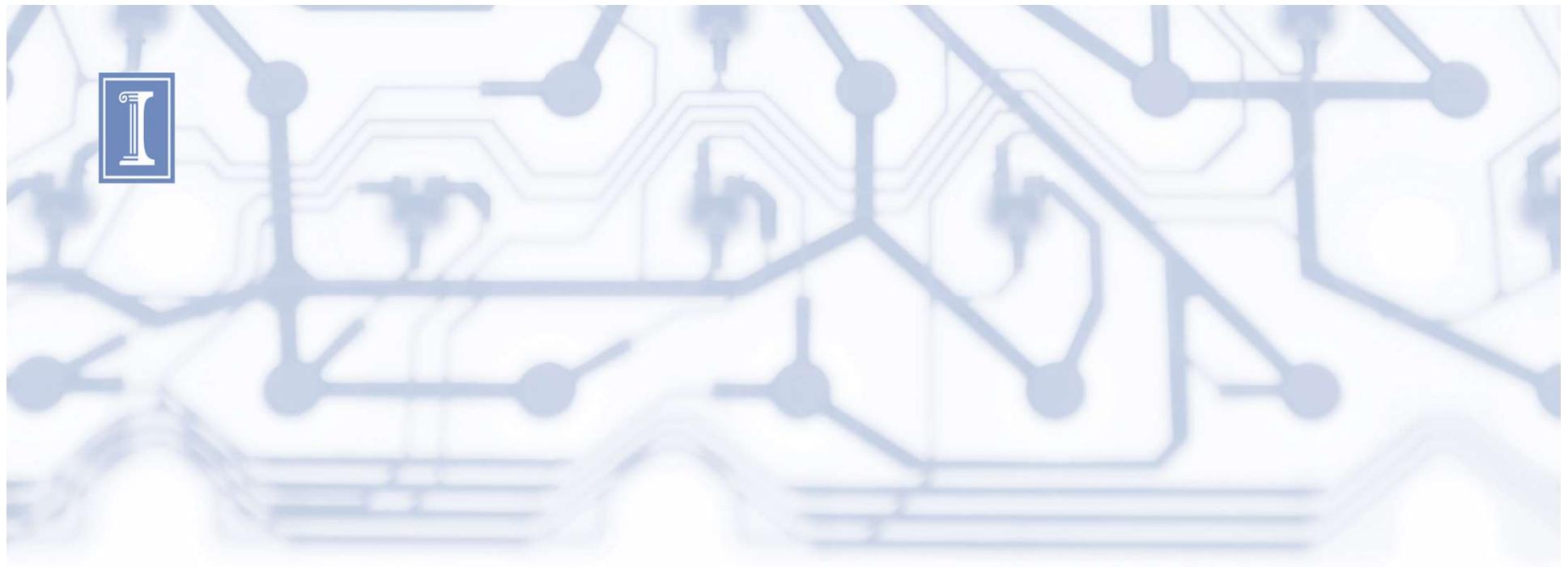
- Aka the Illinois Protocol
- Supports “write-back-caches”
- Uses Cache-to-cache transfer
- States:
 - Modified
 - Exclusive
 - Shared
 - Invalid



Coherency Mechanisms

- Snooping
 - Send all requests for data to all processors
 - Processors “snoop” to see if they have a local copy
 - Requires broadcast
 - Works well with a “bus” → CPUs
 - Usually implemented with state machines
- Directory-based
 - Keep track of what is being share in a centralized location
 - In reality, every block in every cache
 - Send point-to-point requests
 - Scales better than snooping
 - Lcache, WP-Lcache
- TTL: Eventually consistent caches
 - Main mechanism for DNS caching records





CLOUD COMPUTING APPLICATIONS

AWS ElastiCache for Memcached
Prof. Reza Farivar

MemCached

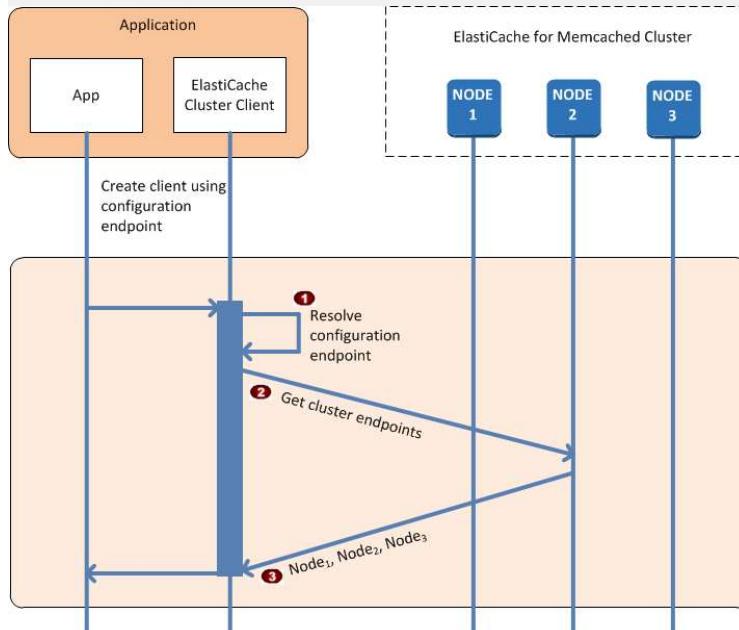
- Simple key/value storage model
 - E.g. in a Lambda Python function, just import pymemcache and set / get data
 - Other libraries: python-Memcached, pylibmc, twisted-Memcached, etc.
- Simple data model:
 - Strings, Objects
 - From the results of database calls, API calls, even HTML page renderings (e.g. PHP to HTML)
- No data storage
 - If a node goes down, you lose all the data in that node's memory
- Multi-threaded engine
- Multi-cluster using consistent hashing
 - Default limit 20 nodes
- Auto Discovery

ElastiCache Memcached Auto Discovery

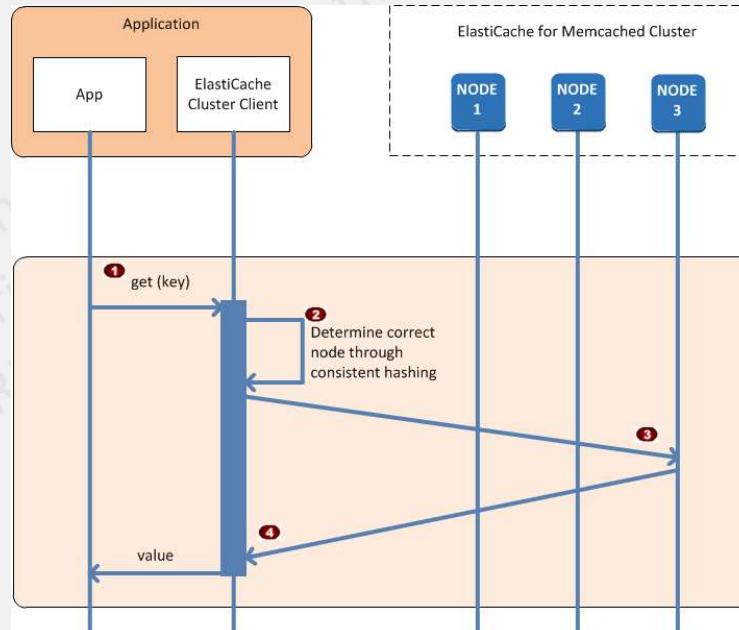
- Configuration endpoint
 - When you increase the number of nodes in a cache cluster, the new nodes register themselves with the configuration endpoint and with all of the other nodes
- When you remove nodes from the cache cluster, the departing nodes deregister themselves.
- In both cases, all the other nodes in the cluster are updated with the latest cache node metadata.
- Cache node failures are automatically detected; failed nodes are automatically replaced.
- A client program only needs to connect to the configuration endpoint.
- After that, the Auto Discovery library connects to all of the other nodes in the cluster.
- Client programs poll the cluster once per minute (this interval can be adjusted if necessary). If there are any changes to the cluster configuration, such as new or deleted nodes, the client receives an updated list of metadata. Then the client connects to, or disconnects from, these nodes as needed.

ElastiCache for Memcached Auto Discovery

Connecting to Cache nodes



Normal operation



Cluster Client available for Java, .Net and PHP

Example use of Elasticache for Memcached

```
from __future__ import print_function
import time
import uuid
import sys
import socket
import elasticache_auto_discovery
from pymemcache.client.hash import HashClient

#elasticache settings
elasticache_config_endpoint = "your-elasticache-cluster-endpoint:port"
nodes = elasticache_auto_discovery.discover(elasticache_config_endpoint)
nodes = map(lambda x: (x[1], int(x[2])), nodes)
memcache_client = HashClient(nodes)

def handler(event, context):
    """
    This function puts into memcache and get from it.
    Memcache is hosted using elasticache
    """

    #Create a random UUID... this will be the sample element we add to the cache.
    uuid_inserted = uuid.uuid4().hex
    #Put the UUID to the cache.
    memcache_client.set('uuid', uuid_inserted)
    #Get item (UUID) from the cache.
    uuid_obtained = memcache_client.get('uuid')
    if uuid_obtained.decode("utf-8") == uuid_inserted:
        # this print should go to the CloudWatch Logs and Lambda console.
        print ("Success: Fetched value %s from memcache" %(uuid_inserted))
    else:
        raise Exception("Value is not the same as we put :(. Expected %s got %s" %(uuid_inserted, uuid_obtained))

    return "Fetched value from memcache: " + uuid_obtained.decode("utf-8")
```



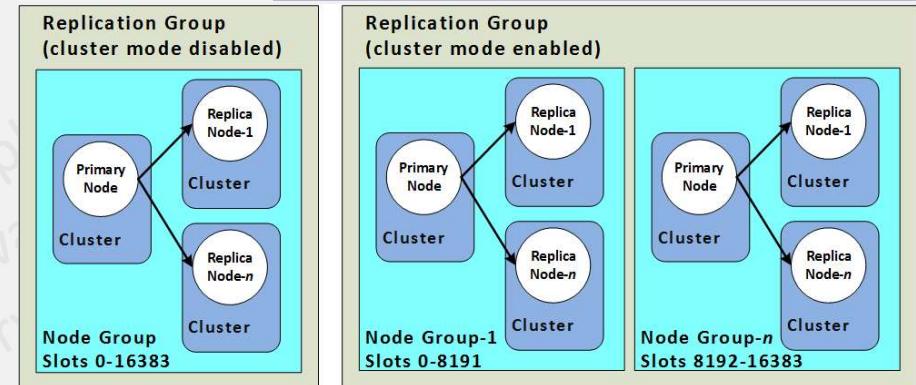
CLOUD COMPUTING APPLICATIONS

AWS ElastiCache for Redis

Prof. Reza Farivar

ElastiCache for Redis

- Up to 250 shards
- Each shard can be on a node-group
- Each node group can have one master (Write + read) and 5 other read replicas
 - If any primary has no replicas and the primary fails, you lose all that primary's data
- Node or shard limit of 500 in Redis 5.0.6+
 - 83 shards (one primary and 5 replicas per shard)
 - 500 shards (single primary and no replicas)



Designing the Right Cache

- At the highest level, Memcached is generally used to store small and static data, such as HTML code pieces
 - Memory management is efficient and simple
 - No data persistence
 - If any node/cluster fails Memcached data is lost
 - Use Memcached with easily recoverable data
- Redis supports more complex data structures
 - Fast performance
 - Persistent storage
 - Read replicas

Comparing Memcached and Redis

- Memcahced
 - Simple model
 - Strings, Objects
 - Large nodes, multithreading
 - Ability to scale out, and scale in
- Redis
 - Complex Data Types
 - Strings, Hashes, lists, sets, sorted sets, bitmaps
 - Sort in-memory datasets
 - Persistence of the key store
 - Replicate data for read-access to up to 5 read replicas per shard
 - Automatic failover if the primary node fail
 - Authenticate users with role-based access control
 - Redis streams: log data structure, producers append new data, consumers consume messages
 - Encryption
 - HIPAA eligible, PCI DSS, FedRAMP
 - Dynamically adding / removing shards from cluster mode Redis
 - Online resharding



CLOUD COMPUTING APPLICATIONS

Cloud Caching Strategies

Prof. Reza Farivar

Caching Strategies

- strategies to implement for populating and maintaining your cache depend upon what data you cache and the access patterns to that data
 - Cache Aside (Lazy Loading)
 - Write-Through
 - Adding TTL
- Deploying nodes to multiple Availability Zones (Elasticache supports this) can avoid single point of failure and provides high availability

The Right Caching Strategy

- Cache-Aside (Lazy Loading)
 - Application data is written only into the source
 - Only loads data to the cache when it is required on a “read”
 - Typically most data is never requested
 - Suitable for read-heavy Applications
 - Allows stale data
 - In case of cache node failure, just read from source
- Write-Through
 - Application data is written into the cache and source at the same time
 - Suitable for write-heavy Applications, where data loss is not acceptable
 - But every write is expensive
 - Cache never gets stale
 - In case of cache node failure, just read from source
- Write-Back (Lazy Writing)
 - Application data is written only to the Cache
 - More complex to implement

Cache Sharding

- There is only one machine that contains each piece of data
- Memcached:
 - Consistent Hashing ring algorithm
- Redis:
 - ElastiCache for Redis can have up to 250 shards
 - Each shard can consist of a master Redis node, and up to 5 Redis Read replicas
- Sharding is a great technique but has its own problems
 - Resharding data when adding/removing nodes
 - Celebrity problem
 - Join and de-normalization:
 - Not as big a problem in Caches, but can be serious for databases
 - Once data is sharded, it is hard to perform join operations across all shards.

Time to Live (TTL)

- Lazy loading allows for stale data but doesn't fail with empty nodes
- Write-through ensures that data is always fresh, but can fail with empty nodes and can populate the cache with superfluous data
- By adding a time to live (TTL) value to each write, you can have the advantages of each strategy. At the same time, you can and largely avoid cluttering up the cache with extra data.
- *Time to live (TTL)* is an integer value that specifies the number of seconds until the key expires
 - Redis can specify seconds or milliseconds for this value.
 - For Memcached, it is seconds.
- When an application attempts to read an expired key, it is treated as though the key is not found. The database is queried for the key and the cache is updated.
- This approach doesn't guarantee that a value isn't stale. However, it keeps data from getting too stale and requires that values in the cache are occasionally refreshed from the database.



CLOUD COMPUTING APPLICATIONS

Roy Campbell & Reza Farivar

Redis

Redis: REmote DIctionary Server

- Open Source
- Written in C
- Data model is a dictionary which maps keys to values
- Supports not only strings, but also abstract data types:
 - Lists of strings
 - Sets of strings (collections of non-repeating unsorted elements)
 - Sorted sets of strings (collections of non-repeating elements ordered by a floating-point number called score)
 - Hashes where keys and values are strings

Redis

- Ultrafast response time
 - Everything is in memory
 - Non-blocking I/O, single threaded
 - 100,000+ read / writes per second
- Periodically checkpoints in-memory values to disk every few seconds
- In case of failure, only the very last seconds' worth of key/values are lost

Potential Uses

- Session store
 - One (or more) sessions per user
 - Many reads, few writes
 - Throw-away data
 - Timeouts
- Logging
 - Rapid, low latency writes
 - Data you don't care that much about
 - Not that much data (must be in-memory)

General Cases

- Data that you don't mind losing
- Records that can be accessed by a single primary key
- Schema that is either a single value or is a serialized object

Simple Programming Model

- GET, SET, INCR, DECR, EXISTS, DEL
- HGET, HSET, KEYS, HDEL
- SADD, SMEMBERS, SRLEM
- PUBLISH, SUBSCRIBE

Summary

- Redis is not a database
 - It complements your existing data storage layer
 - E.g. stackoverflow uses Redis for data caching
- Publish/Subscribe support



CLOUD COMPUTING APPLICATIONS

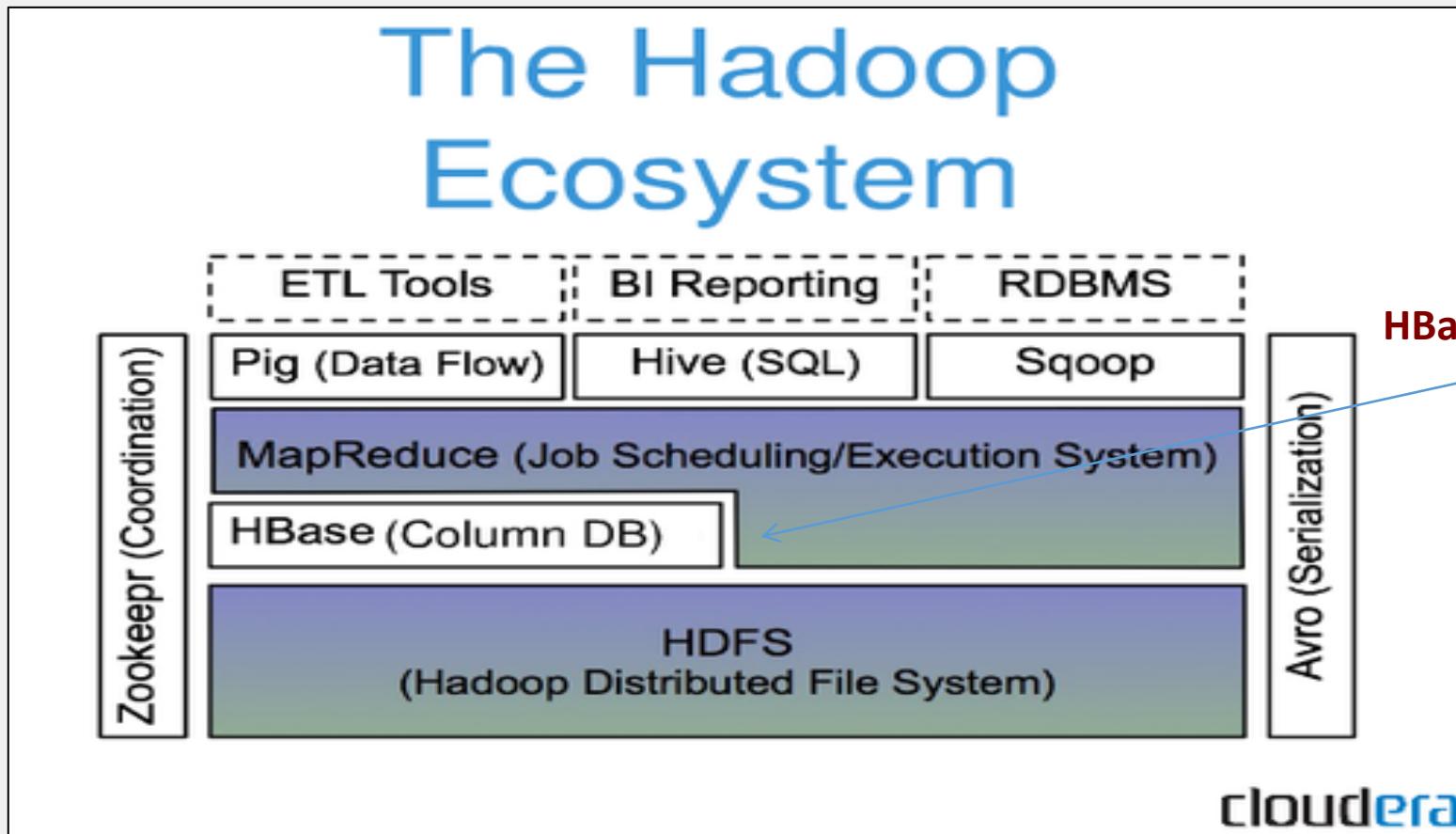
Prof. Roy Campbell

HBase Usage API

HBase: Overview

- HBase is a distributed column-oriented data store built on top of HDFS
- HBase is an Apache open source project whose goal is to provide storage for Hadoop Distributed Computing
- Data is logically organized into tables, rows, and columns

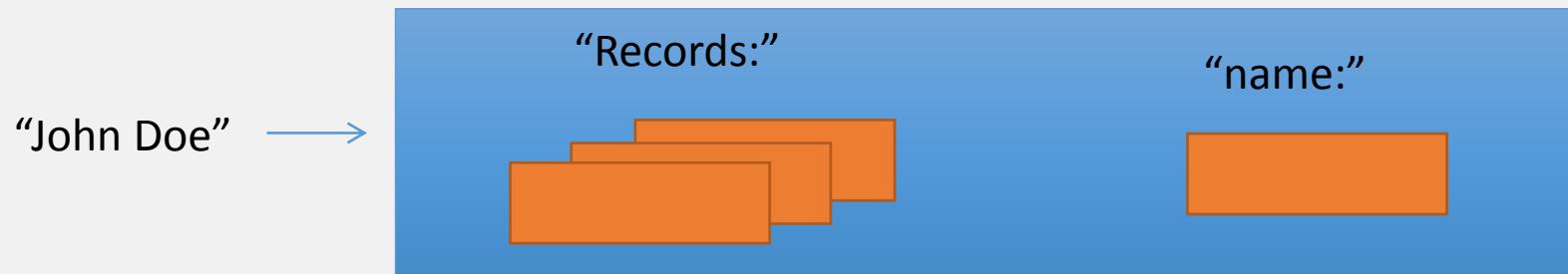
HBase: Part of Hadoop's Ecosystem



HBase vs. HDFS

- **HDFS** is good for batch processing (scans over big files)
 - Not good for record lookup
 - Not good for incremental addition of small batches
 - Not good for updates
- **HBase** addresses the above points
 - Fast record lookup
 - Support for record-level insertion
 - Support for updates

Data Model



- A table in Bigtable is a sparse, distributed, persistent multidimensional sorted map
- Map indexed by a row key, column key, and a timestamp
 - (row:string, column:string, time:int64) → uninterpreted byte array
- Supports lookups, inserts, deletes
 - Single row transactions only

Notes on Data Model

- HBase schema consists of several **Tables**
- Each table consists of a set of **Column Families**
 - Columns are not part of the schema
- HBase has **Dynamic Columns**
 - Because column names are encoded inside the cells
 - Different cells can have different columns

“Roles” column family has
different columns in different
cells



Row key	Data
cutting	info: { 'height': '9ft', 'state': 'CA' } roles: { 'ASF': 'Director', 'Hadoop': 'Founder' }
tlipcon	info: { 'height': '5ft7', 'state': 'CA' } roles: { 'Hadoop': 'Committer' @ts=2010, 'Hadoop': 'PMC' @ts=2011, 'Hive': 'Contributor' } ↑↑

Notes on Data Model

- The **version number** can be user-supplied
 - Does not have to be inserted in increasing order
 - Version numbers are unique within each key
- Table can be very sparse
 - Many cells are empty
- **Keys** are indexed as the primary key

Has two columns
[cnnsi.com & my.look.ca]



Row Key	Time Stamp	ColumnFamily contents	ColumnFamily anchor
"com.cnn.www"	t9		anchor:cnnsi.com = "CNN"
"com.cnn.www"	t8		anchor:my.look.ca = "CNN.com"
"com.cnn.www"	t6	contents:html = "<html>..."	
"com.cnn.www"	t5	contents:html = "<html>..."	
"com.cnn.www"	t3	contents:html = "<html>..."	

Rows and Columns

- Rows maintained in sorted lexicographic order
 - Applications can exploit this property for efficient row scans
 - Row ranges dynamically partitioned into tablets
- Columns grouped into column families
 - Column key = *family:qualifier*
 - Column families provide locality hints
 - Unbounded number of columns

HBase lookup example

```
Scanner scanner (T);
ScanStream *stream;
stream = scanner.FetchColumnFamily("name");
stream -> SetReturnAllVersions();
//Filter return sets using regex
scanner.lookup ("John Doe");
for (; !stream -> Done(); stream -> Next()) {
    printf ("%s %s %s \n",
           scanner.RowName(),
           stream -> ColumnName(),
           stream – Value());
}
```

HBase lookup example

```
Table *T = OpenOrDie (“/hbase/myTable”);
```

```
RowMutation rowMut (T, “John Doe”);  
rowMut.Set (“name:Jane Roe”, “NAMES”);  
rowMut.Delete(“name:Jack Public”);  
Operation op;  
Apply (&op, &rowMut);
```

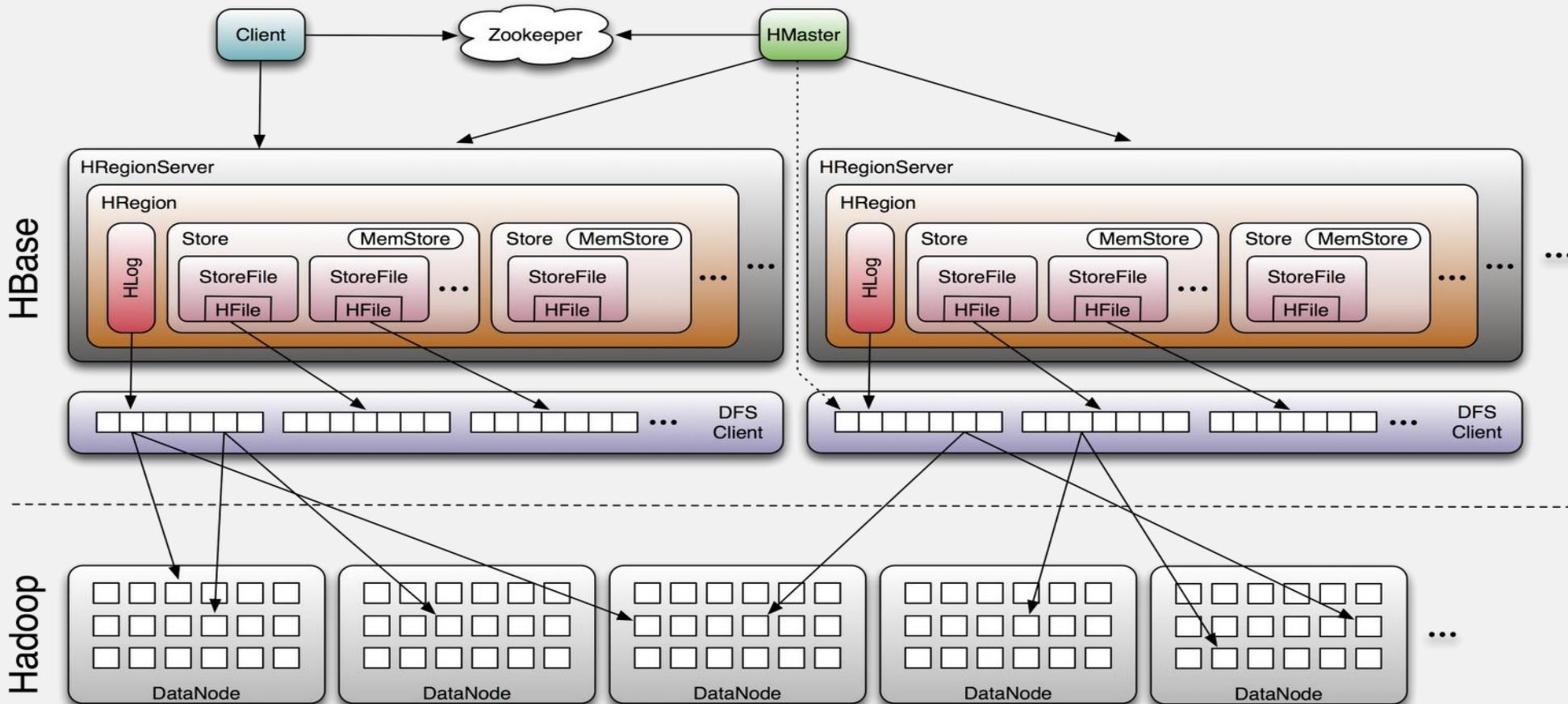


CLOUD COMPUTING APPLICATIONS

Roy Campbell & Reza Farivar

HBase Internals - Part 1

HBase



HBase Building Blocks

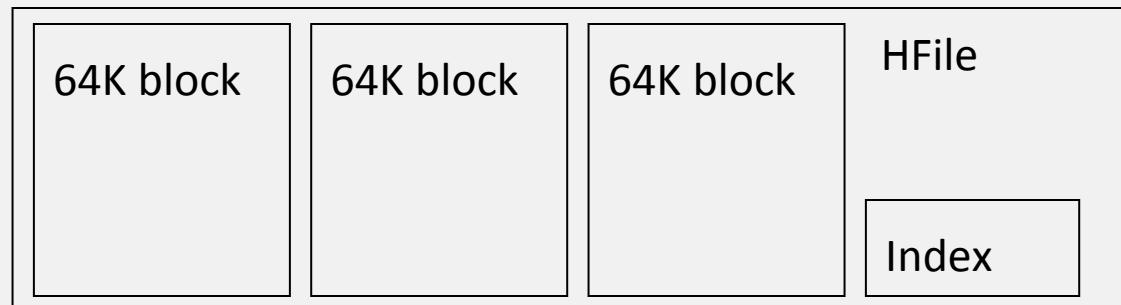
- HDFS
- Apache ZooKeeper
 - ZooKeeper uses ZAB (ZooKeeper's Atomic Broadcast)
- HFile

HFile

- Basic building block of HBase
- On-disk file format representing a map from string to string
- Persistent, ordered immutable map from keys to values
 - Stored in HDFS

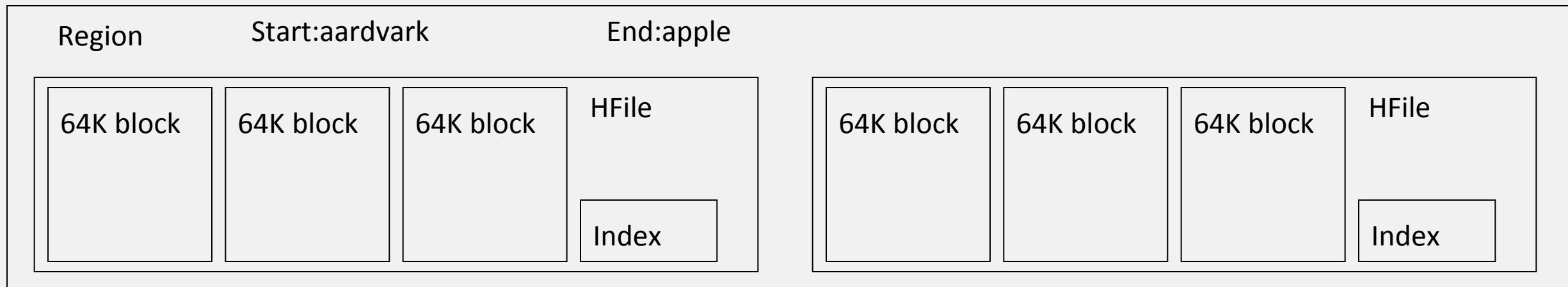
HFile

- Sequence of blocks on disk plus an index for block lookup
 - Can be completely mapped into memory
 - MemStore
- Supported operations:
 - Lookup value associated with key
 - Iterate key/value pairs within a key range



HRegion

- Dynamically partitioned range of rows
- Built from multiple HFiles



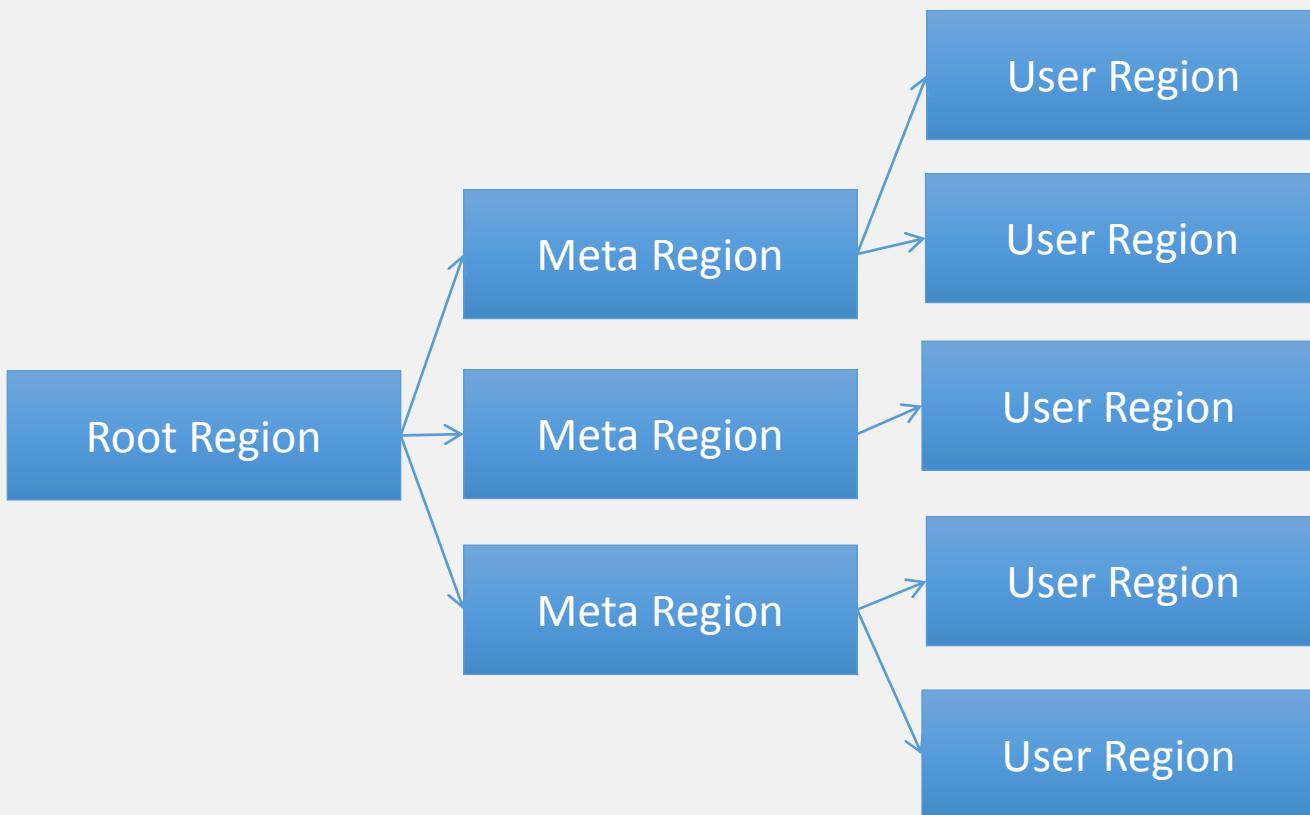
HBase Master

- Assigns HFiles to HRegion servers
- Detects addition and expiration of HRegion servers
- Balances HRegion server load. HRegions are distributed randomly on nodes of the cluster for load balancing.
- Handles garbage collection
- Handles schema changes

HRegion Servers

- Each HRegion server manages a set of regions
 - Typically between 10 to 1,000 regions
 - Each 100-200 MB by default
- Handles read and write requests to the regions
- Splits regions that have grown too large

HRegion Location





CLOUD COMPUTING APPLICATIONS

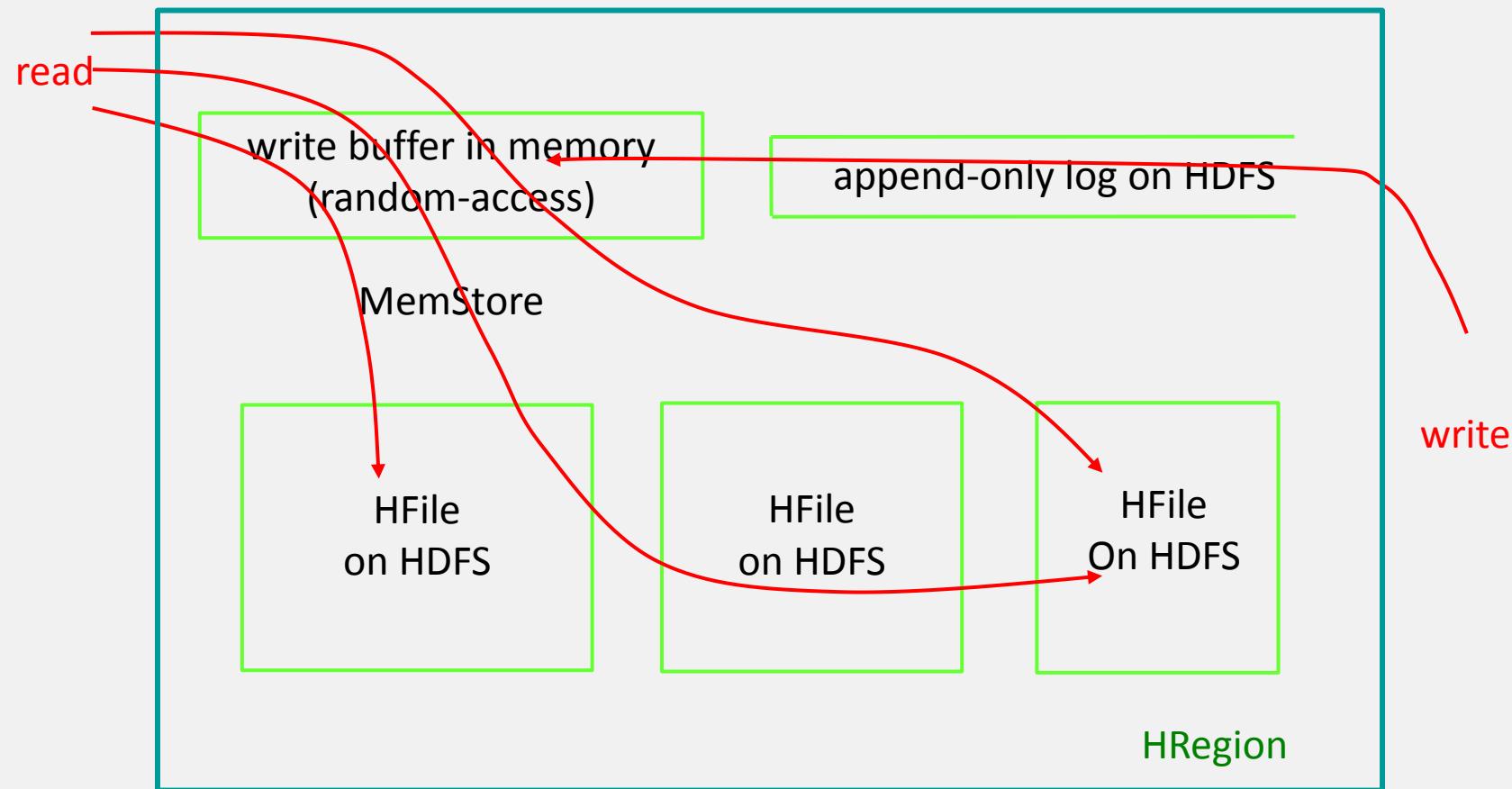
Roy Campbell & Reza Farivar

HBase Internals - Part 2

HRegion Assignment

- HBase Master keeps track of:
 - Set of live HRegion servers
 - Assignment of HRegions to HRegion servers
 - Unassigned HRegions
- Each HRegion is assigned to one HRegion server at a time
 - HRegion server maintains an exclusive lock on a file in ZooKeeper
 - HBase Master monitors HRegion servers and handles assignment
- Changes to HRegion structure
 - Table creation/deletion (HBase Master initiated)
 - HRegion merging (HBase Master initiated)
 - HRegion splitting (HRegion server initiated)

HRegion in Memory Representation





CLOUD COMPUTING APPLICATIONS

Spark SQL

Roy Campbell & Reza Farivar

Spark SQL

- Structured Data Processing in Apache Spark
- Built on top of RDD data abstraction
- Need more information about the columns (Schema)
 - Can use this for optimizations
- Spark can read data from HDFS, Hive tables, JSON, etc.
- Can use SQL to query the data
- When needed, switch between SQL and python/java/scala
- Strong query engine

DataSet

- A Dataset is a distributed collection of data
- Dataset provides the benefits of RDDs (strong typing, ability to use powerful lambda functions) with the benefits of Spark SQL's optimized execution engine
- A Dataset can be constructed from JVM objects and then manipulated using functional transformations (map, flatMap, filter, etc.)
- The Dataset API is available in Scala and Java
 - Python does not currently (as of 2.0.0) have the support for the Dataset API.
 - But due to Python's dynamic nature, many of the benefits of the Dataset API are already available (row.columnName).

DataFrame

- A DataFrame is a *Dataset* organized into named columns
- Equivalent to a table in a relational database or a data frame in R/Python
- DataFrames can be constructed from:
 - structured data files
 - tables in Hive
 - external databases
 - existing RDDs
- The DataFrame API is available in Scala, Java, Python, and R



CLOUD COMPUTING APPLICATIONS

HIVE

Roy Campbell & Reza Farivar

Hive: Background

- Started at Facebook
- Data was collected by nightly cron jobs into Oracle DB
- “ETL” via hand-coded Python
- HQL, a variant of SQL
- Translates queries into map/reduce jobs, Hadoop YARN, Tez, or Spark
- Note
 - No UPDATE or DELETE support, for example
 - Focuses primarily on the query part of SQL

Hive: Example

- Hive looks similar to an SQL database
- Relational join on two tables:
 - Table of word counts from Shakespeare collection
 - Table of word counts from Homer

Hive: Example

```
SELECT s.word, s.freq, k.freq FROM shakespeare s  
JOIN homer k ON (s.word = k.word) WHERE s.freq >= 1 AND k.freq >= 1  
ORDER BY s.freq DESC LIMIT 10;
```

the	25848	62394
I	23031	8854
and	19671	38985
to	18038	13526
of	16700	34654
a	14170	8057
you	12702	2720
my	11297	4135
in	10797	12445
is	8882	6884

Hive: Behind the Scenes

```
SELECT s.word, s.freq, k.freq FROM shakespeare s  
JOIN homer k ON (s.word = k.word) WHERE s.freq >= 1 AND k.freq >= 1  
ORDER BY s.freq DESC LIMIT 10;
```



((Abstract Syntax Tree)

```
(TOK_QUERY (TOK_FROM (TOK_JOIN (TOK_TABREF shakespeare s) (TOK_TABREF homer k) (= (.  
(TOK_TABLE_OR_COL s) word) (. (TOK_TABLE_OR_COL k) word)))) (TOK_INSERT (TOK_DESTINATION (TOK_DIR  
TOK_TMP_FILE)) (TOK_SELECT (TOK_SELEXPR (. (TOK_TABLE_OR_COL s) word)) (TOK_SELEXPR (.  
(TOK_TABLE_OR_COL s) freq)) (TOK_SELEXPR (. (TOK_TABLE_OR_COL k) freq))) (TOK_WHERE (AND (>= (.  
(TOK_TABLE_OR_COL s) freq) 1) (>= (. (TOK_TABLE_OR_COL k) freq) 1))) (TOK_ORDERBY  
(TOK_TABSORTCOLNAMEDESC (. (TOK_TABLE_OR_COL s) freq))) (TOK_LIMIT 10)))
```



(one or more of MapReduce jobs)

Hive Components

- Shell: allows interactive queries
- Driver: session handles, fetch, execute
- Compiler: parse, plan, optimize
- Execution engine: DAG of stages (MR, HDFS, metadata)
- Metastore: schema, location in HDFS, etc.

Metastore

- Hive uses a traditional database to store its metadata
 - A namespace containing a set of tables
 - Holds table definitions (column types, physical layout)
 - Holds partitioning information
- Can be stored in MySQL, Oracle, and many other relational databases

Physical Layout

- Warehouse directory in HDFS
 - E.g., /user/hive/warehouse
- Tables stored in subdirectories of warehouse
 - Partitions form subdirectories of tables
- Actual data stored in flat files
 - Control char-delimited text, or SequenceFiles
 - Can be customized to use arbitrary format



CLOUD COMPUTING APPLICATIONS

Decoupling in Cloud Architectures

Prof. Reza Farivar

Multi-tier Distributed Architecture

- Enterprise architectures require elasticity and scalability
 - Scalability: respond to increasing demand
 - Elasticity: respond to decreasing demand
- Fault tolerance
 - Component failure is the rule in cloud environments
- Changing demand patterns
 - Hard to predict how many resources we will need in the future
- Complexity
 - Multiple platforms and development teams

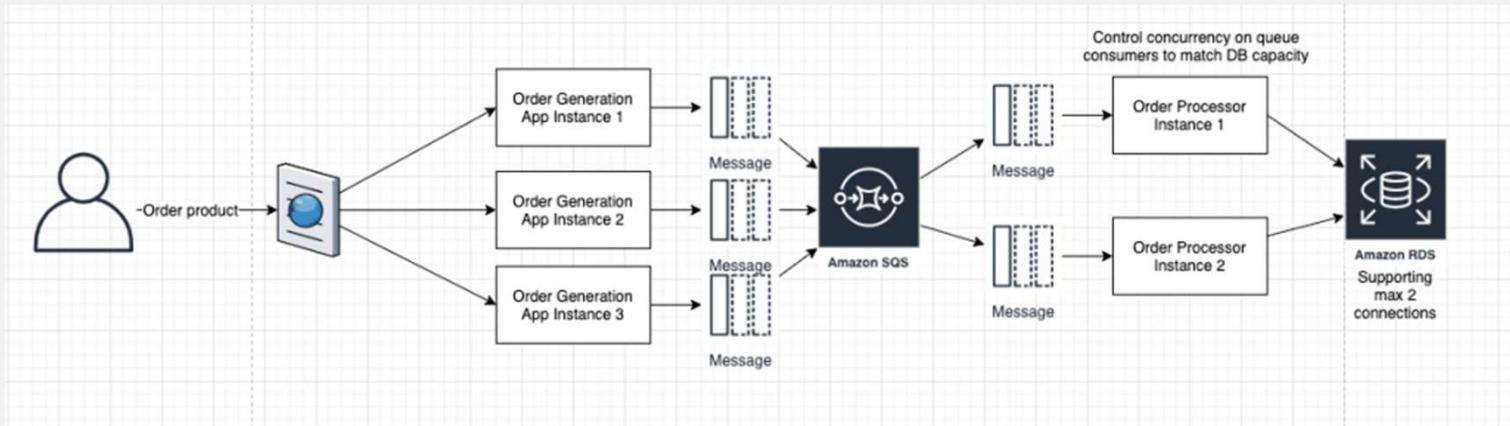
Decoupling

- The key to achieving reliable, scalable, elastic architectures is decoupling
- Building applications from individual components that each perform a discrete function
- A reliable queue between components
 - Allows many integration patterns for connecting services
- Loose coupling → increases an architecture's resiliency to failure and ability to handle traffic spikes
 - producer and consumer operate independently
- Asynchronous Communication

Message Queues

- Message queues → decouple and scale microservices, distributed systems, and serverless applications
 - Send, store, and receive messages between software components
 - Any volume
 - Without losing messages
 - No need to rely on other services be always available
- Can easily handle momentary spikes in demands
 - Up spikes and down spikes
- Guaranteed message delivery
 - At least Once
 - Exactly Once

Example



<https://aws.amazon.com/blogs/architecture/application-integration-using-queues-and-messages/>

Message Queue Platforms

- Open Source
 - Apache ActiveMQ
 - Apache RabbitMQ
 - Apache Kafka
- Proprietary / cloud services
 - Amazon AWS Simple Queue Service
 - Amazon MQ
 - Apache ActiveMQ
 - Apache RabbitMQ
 - Amazon Kinesis
 - Amazon Managed Kafka

Publish-Subscribe Model

- A sibling of the message queue systems
- Producers publish messages to the queue
- Several consumers, having subscribed to a specific producer (or topic, etc.), all receive the message
- Publishers and subscribers are decoupled
- Example:
 - Kafka
 - AWS Simple Notification Service



CLOUD COMPUTING APPLICATIONS

Amazon Simple Queue Service

Prof. Reza Farivar

AWS SQS

- Simple Queue Service
- **put-get-delete** paradigm
 - It requires a consumer to explicitly state that its data has finished processing the message it pulled from the queue
 - The message data is kept safe with the queue and gets deleted from the queue only after confirmation that it has been processed
- Multiple Producers and Consumers
- Pull model
 - The consumers must explicitly pull the queue
- Redundant infrastructure

Guaranteed message delivery: At Least Once

- when a process retrieves the message from the queue, it temporarily makes this message invisible
- When the client informs the queue that it has finished processing the message, SQS deletes this message from the queue
- If the client does not respond back to the queue in a specific amount of time, SQS makes it visible again
- Generally in order message delivery

Guaranteed message delivery: Exactly Once*

- * only true under limited conditions
- FIFO
 - queues work in conjunction with the publisher APIs to avoid introducing duplicate messages
 - Content-based deduplication:
 - SQS generates an SHA-256 hash of the body of the message to generate the content-based message deduplication ID
 - When an application sends a message to the FIFO queue with a message deduplication ID, it is used to deduplicate
 - Message order guarantee
- Bandwidth limits

Recommended reading: <https://sookocheff.com/post/messaging/dissecting-sqs-fifo-queues/>

Short Pull vs Long pull

- Short Pull: Returns empty if there is nothing in the queue
- What then? Pull again
 - AWS charges based on number of requests
- Solution: Long pull
 - `ReceiveMessageWaitTime`
 - The maximum amount of time that a long-polling receive call waits for a message to become available before returning an empty response

Handling Failures

- Visibility of messages in the Queue
 - A consumer has to explicitly “delete” a message after processing is done
 - Timeout period → message becomes visible again
- What if there is something wrong with a particular message?
 - Infinite loop of visible-invisible?
 - Dead Letter Queue (DLQ)
 - Store failed messages that are not successfully consumed by consumer processes
 - Isolate unconsumed or failed messages and subsequently troubleshoot these messages to determine the reason for their failures
 - Redrive Policy
 - If a queue fails to process a message a predefined number of times, that message is moved to the DLQ



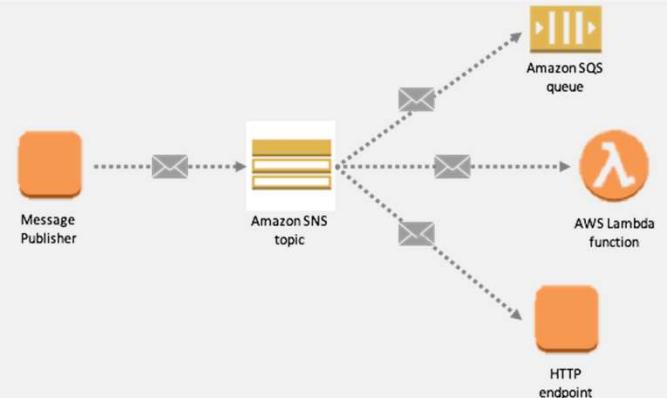
CLOUD COMPUTING APPLICATIONS

Amazon Simple Notification Service

Prof. Reza Farivar

Amazon SNS

- SNS: Simple Notification Service
- PubSub model
 - push
- One to many
- Fan-out architecture
- Endpoint Protocols
 - SQS
 - Lambda
 - HTTP, HTTPS endpoint
 - Email
 - SMS
 - Mobile Device Push Notification



SNS Topic

- A publisher sends a message to an SNS topic
- Subscribers subscribe to SNS topics
- Each SNS topic can have multiple subscribers
 - Each subscriber may use the same protocols or different protocols
- SNS will “push” messages to all subscribers

SNS Messages

- Subject
- Time to live (TTL)
 - It specifies the time to expire for a message
 - Within a specified time, **Apple Push Notification Service (APNS)** or **Google Cloud Messaging (GCM)** must deliver messages to the endpoint.
 - If the message is not delivered within the specified time frame, the message will be dropped with no further attempts to deliver the message
- Payload
 - Can be the same, or different for each endpoint protocol type

Managing Access

- AWS uses a mechanism based on policies
 - Policy: JSON document, consisting of statements
 - Statement: describes a single permission written in an access policy language
 - In JSON
- E.g. the user with AWS account 1111-2222-3333 can publish messages to the topic action (for example, Publish)

```
{  
  "Statement": [  
    {"Sid": "grant-1234-publish",  
     "Effect": "Allow",  
     "Principal": {  
       "AWS": "111122223333"  
     },  
     "Action": ["sns:Publish"],  
     "Resource": "arn:aws:sns:us-east-2:444455556666:MyTopic"  
   ]  
}
```



CLOUD COMPUTING APPLICATIONS

Roy Campbell & Reza Farivar

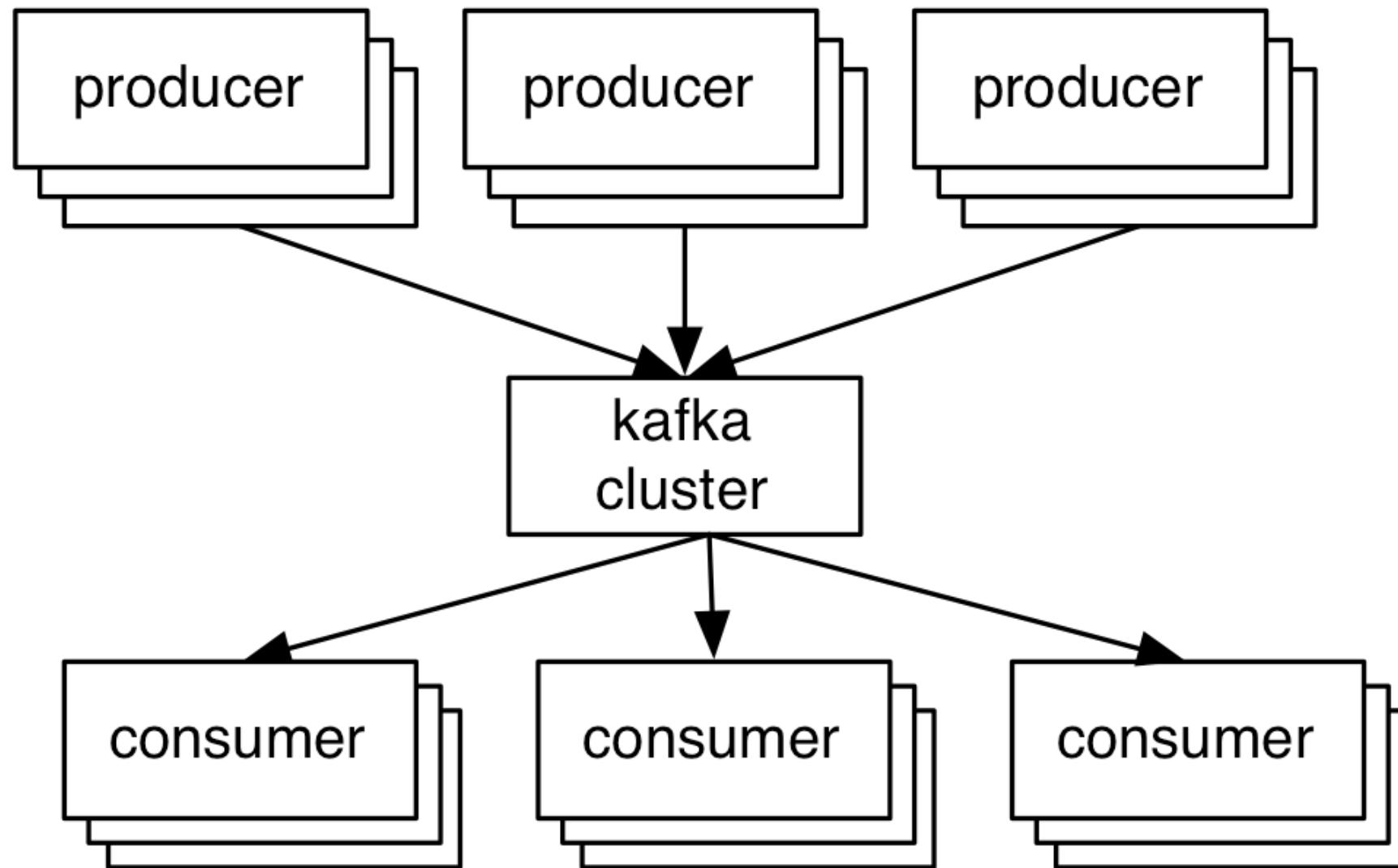
Kafka

Thanks to public domain slides Jiangjie (Becket)
Qin

Contents

- What is Kafka
- Key concepts
- Kafka clients

Kafka: a distributed, partitioned, replicated publish subscribe system providing commit log service



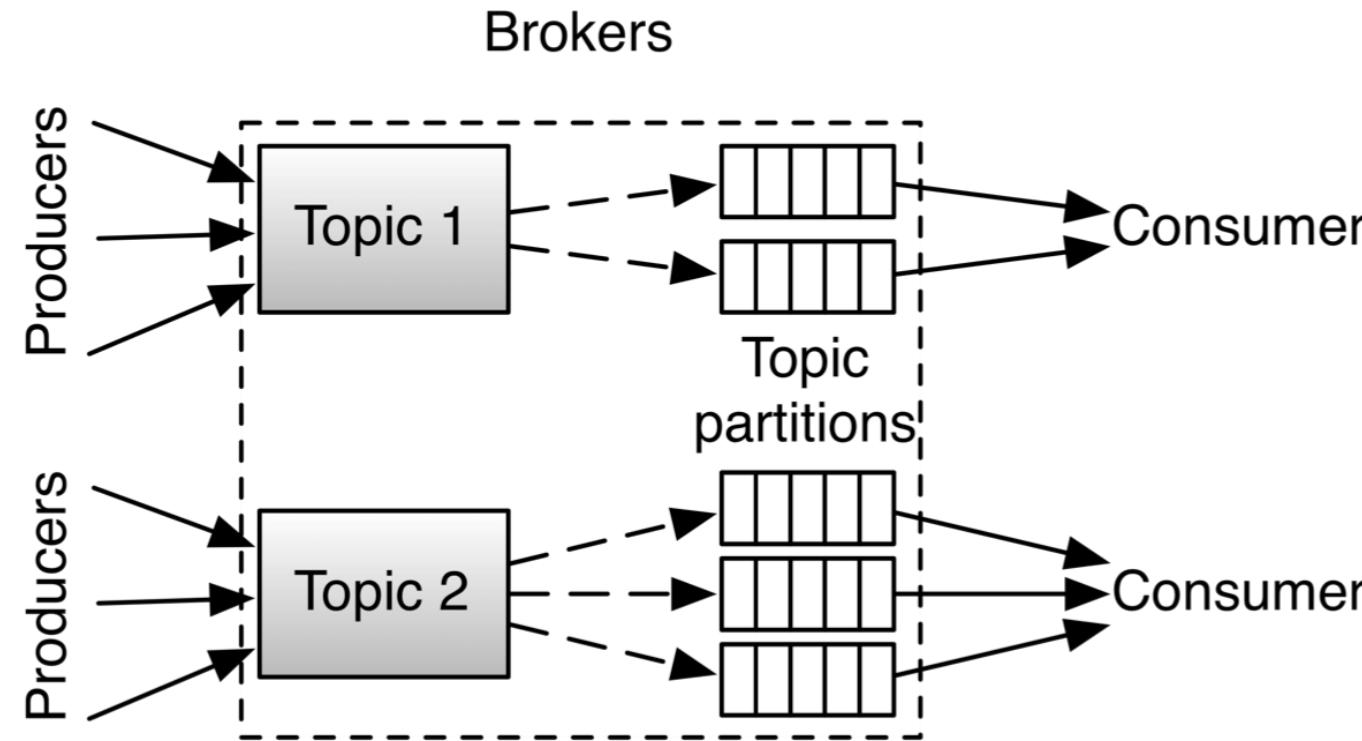
Description

- Kafka maintains feeds of messages in categories called *topics*.
- Processes that publish messages to a Kafka topic are *producers*.
- Processes that subscribe to topics and process the feed of published messages are *consumers*.
- Kafka is run as a cluster comprised of one or more servers each of which is called a *broker*.
- Communication uses TCP, Clients include Java

Characteristics

- Scalability (Kafka is backed by file system)
 - Hundreds of MB/sec/server throughput
 - Many TB per server
- Strong guarantees about messages
 - Strictly ordered (within partitions)
 - All data persistent
- Distributed
 - Replication
 - Partitioning model

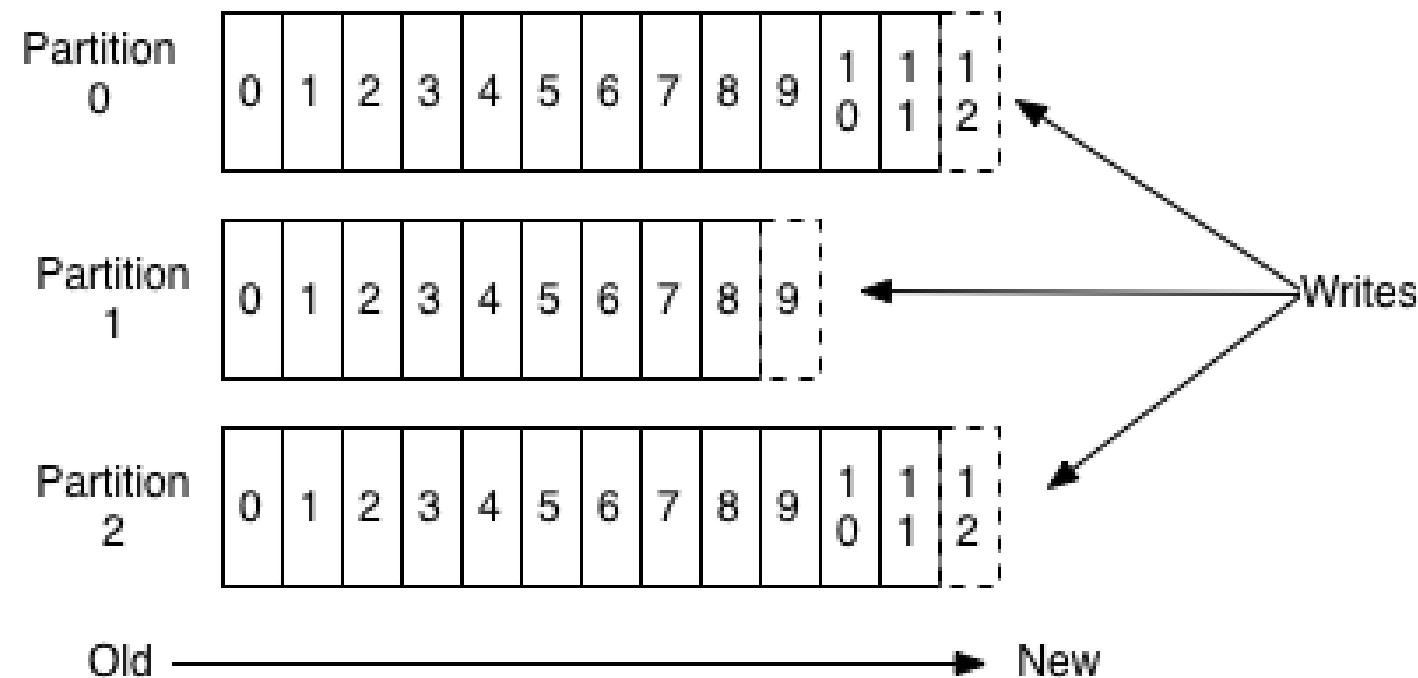
Topics



- A **Topic** has several **Partitions**
- **Partitions** of a **Topic** are distributed across **Brokers**

Topics and Logs

- Kafka stores messages about a topic in a partition as an append only log.

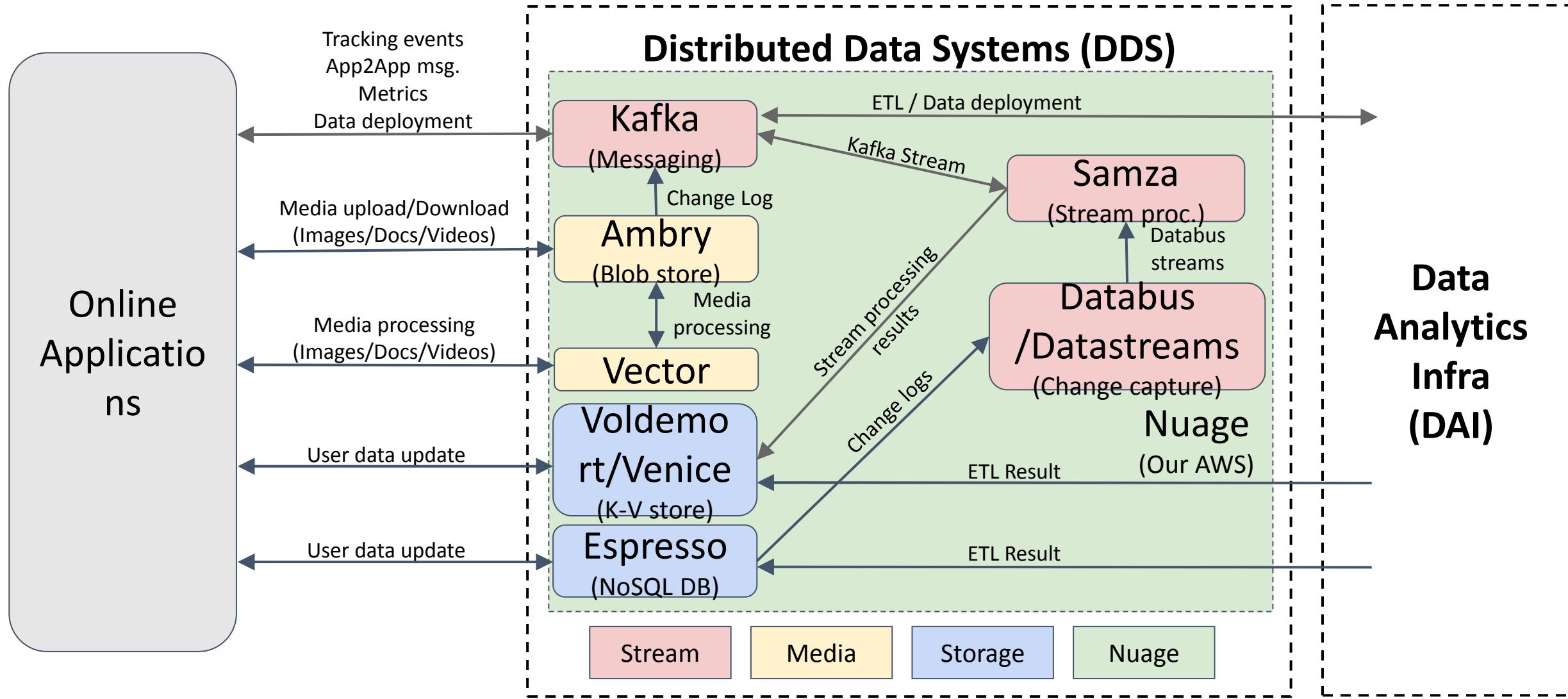


Each partition is an ordered, numbered, immutable append only sequence of messages--- like a commit log.

Kafka Server Cluster Implementation

- Each partition is replicated across a configurable number of servers.
- Each partition has one “leader” server and 0 or more followers.
- A leader handles read and write requests
- A follower replicates the leader and acts as backup.
- Each server is a leader for some of its partitions and a follower for others to load balance
- Zookeeper is used to keep the servers consistent

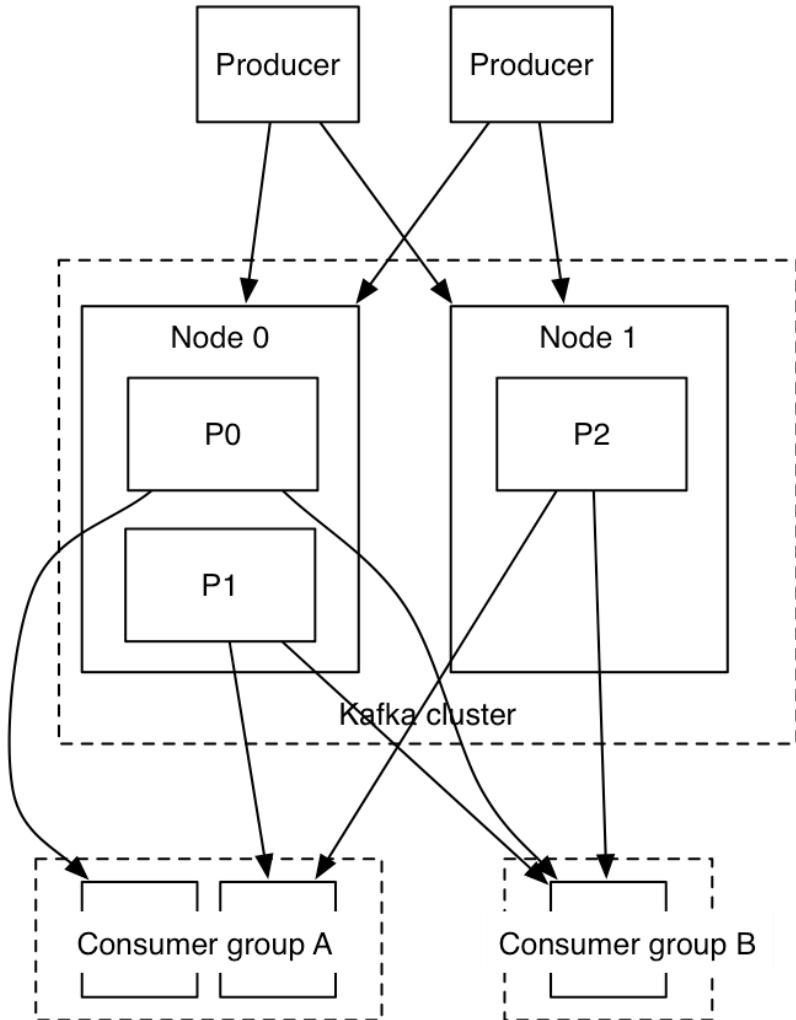
Kafka in a big picture (Linked In)



Producer in Kafka

- Send messages to Kafka **Brokers**
- Messages are sent to a **Topic**
 - Messages with same **Key** go to same partition (so they are in order)
 - Messages without a key go to a random partition (no order guarantee here)
 - Number of partitions changed? - Sorry...Same key might go to another partition...

Consumer in Kafka



- A consumer **can** belong to a **Consumer Group (CG)**
- Consumers in the same **CG**
 - Coordinate with each other to determine which consumer will consume from which partition
 - Share the **Consumer Offsets**

Offset

From Brokers' View

- The Index of a message in a log
- **Message Offset** does not change

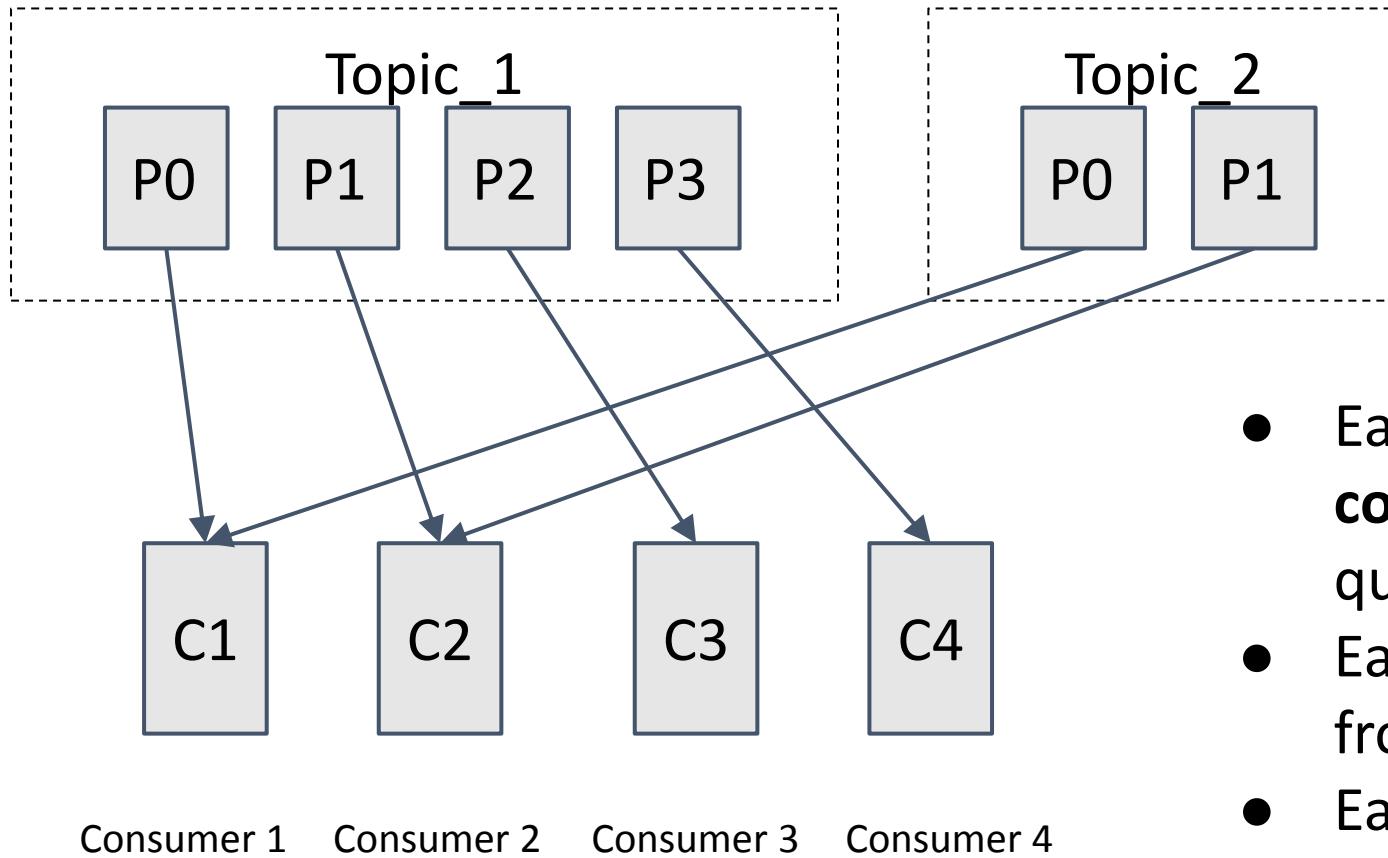
From Consumers' View

- **Consumer Offset**
- The position from where I am consuming
- Consumer Offset can change

More about Consumer Offsets

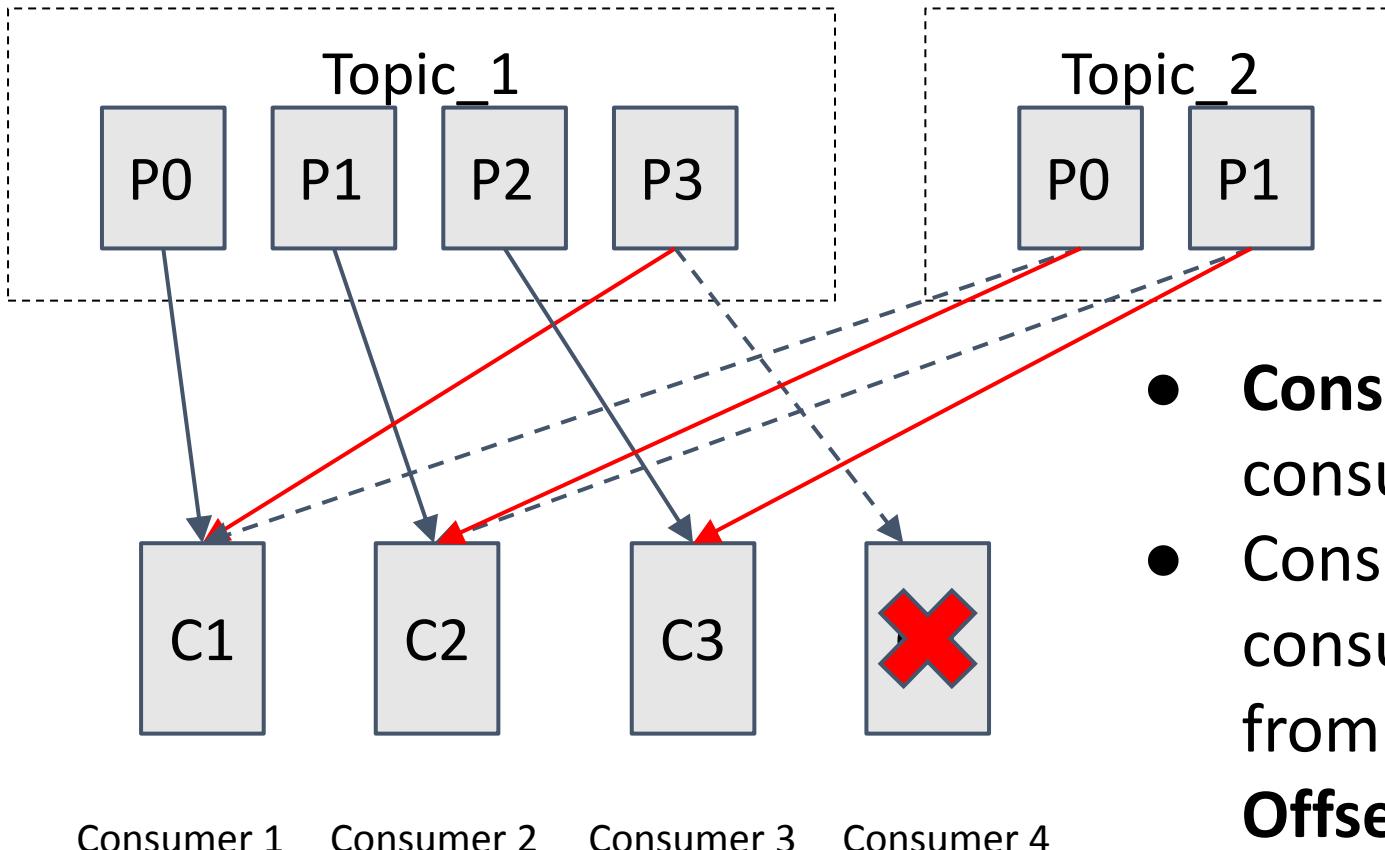
- Consumer Offsets are per
Topic/Partition/ConsumerGroup
(For a given group, look up the last consumed position in a topic/partition)
- Consumer Offsets can be **committed** as a checkpoint of consumption so it can be used when
 - Another consumer in the same CG takes over the partition
 - Resuming consumption later from committed offsets

Consumer Rebalance



- Each consumer can have several **consumer threads** (essentially one queue per thread)
- Each consumer thread can consume from multiple partitions
- Each partition will be consumed by **exactly one consumer in the entire group**

Consumer Rebalance



- **Consumer rebalance** occurs when consumer 4 is down
- Consumer 1, 2, 3 takes over consumer 4's partitions and **resume** from the last committed **Consumer Offsets of the CG**
- **Transparent to user**