Technische Universität Darmstadt

Telecooperation Lab
Prof. Dr. Max Mühlhäuser

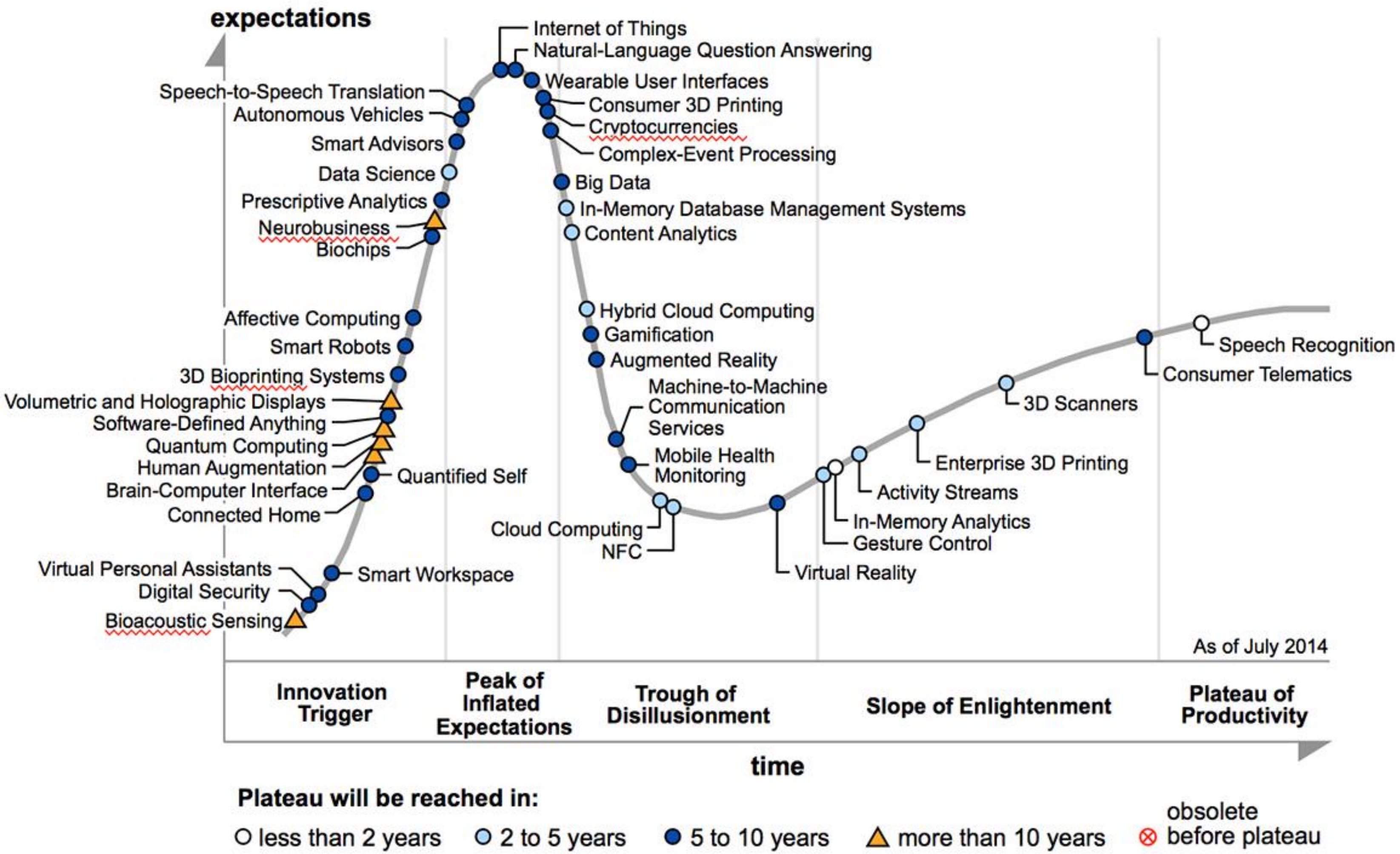
TK1: Distributed Systems - Programming & Algorithms

Chapter 2: Distributed Programming

Section 3: Cloud Computing

Lecturer: Prof. Dr. Max Mühlhäuser

Cloud Computing @ “Hype Cycle”



Gartner Hype Cycle 2014

Cloud Computing: Basic Idea

- **Provide resources or services** over the Internet to customers with the **reliability of a data center**
- Often **virtualized infrastructure** that can be **scaled on demand** regarding storage, computation and network capacity
- Service providers/users do not need to have knowledge of maintaining the technology infrastructure in the "cloud"

Cloud Computing: NIST Definition

NIST
National Institute of Standards and Technology
U.S. Department of Commerce

- **On-demand self-service:**
  - Unilateral provision of computing capabili-ties (**server time and network storage)**
  - Provision as needed automatically
  - Requiring no **human interaction** with each service provider
- **Broad Network Access**
  - Capabilities available over the network
  - Capabilities accessed through standard mechanisms
- **Resource Pooling**
  - Provider pools resources to serve multiple consumers using a multi-tenant model
  - Demand based reassignment of physical and virtual resources
- **Rapid Elasticity**
  - Capabilities can be elastically provisioned and released, in some cases automatically, to scale rapidly outward and inward commensurate with demand.
  - To the consumer, the capabilities available for provisioning often appear to be unlimited and can be appropriated in any quantity at any time.
- **Measured Service**
  - Cloud systems automatically control and optimize resource use by leveraging a metering capability at some level of abstraction appropriate to the type of service (e.g., storage, processing, bandwidth, and active user accounts).
  - Resource usage can be monitored, controlled, and reported, providing transparency for both the provider and consumer of the utilized service.

In-House vs. External Hosting: Analogy

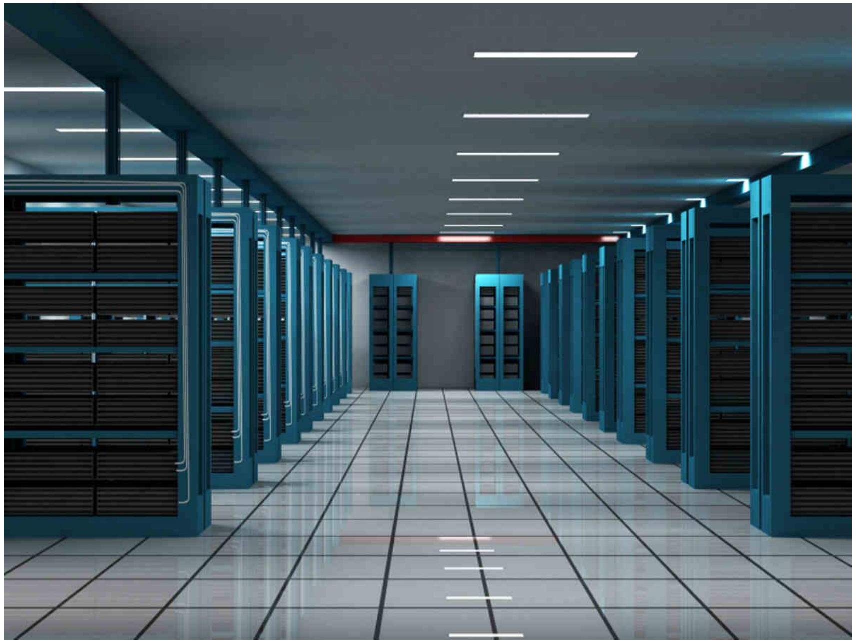
Cow	Supermarket: Bottle of Milk
Big upfront investment	Small continuous expense
Produces more than you need	Buy what you need
Uses up resources	Less resources needed
Maintenance needed	No maintenance
Unpleasant waste product	Waste is not my problem

Source: Donald Kossmann & Kraska 2009: What is new in the Cloud?

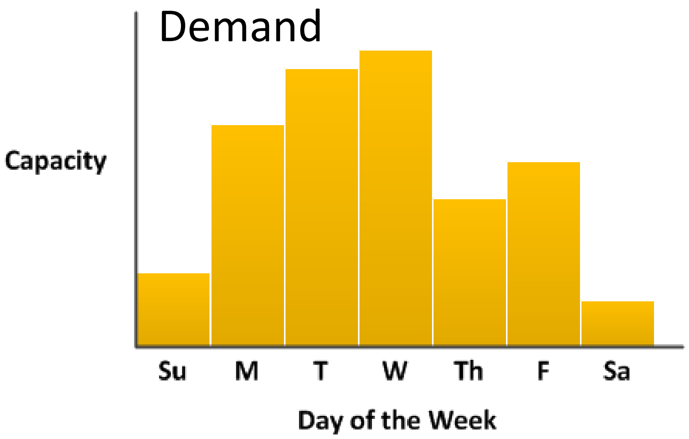
Analogy “translated”

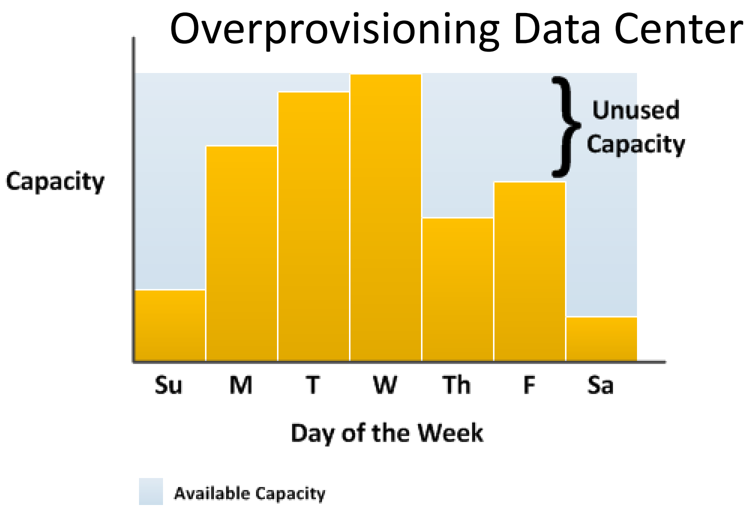
Cloud Computing (vs. traditional in-house hosting)

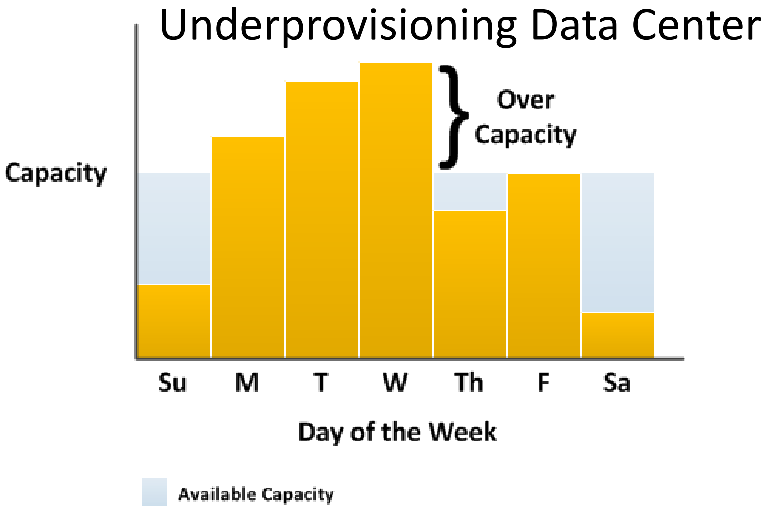
- Sold on demand (typically by minute or hour)
  - “pay as you go” – no need for long-standing contracts
- Elastic: user can determine amount of service (CPU, Storage, ...) he wants at any given time
- Service is fully managed

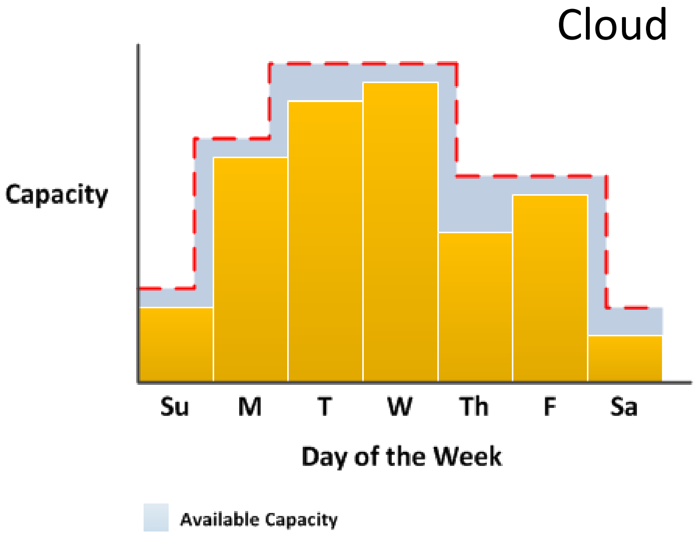


Elasticity

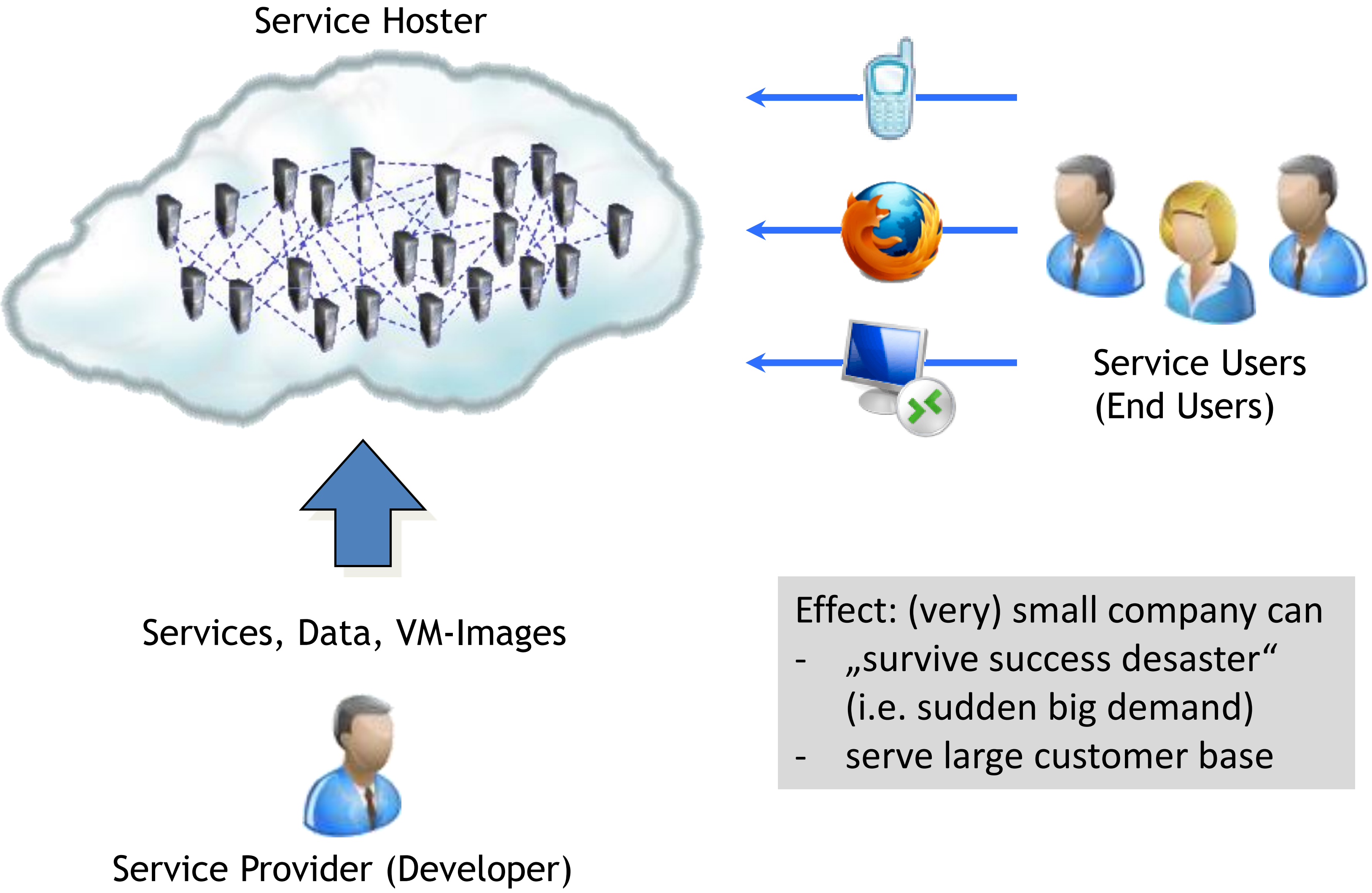

Demand


Overprovisioning Data Center


Underprovisioning Data Center


Cloud

Cloud Computing: Roles



Service Hoster

Service Users
(End Users)

Services, Data, VM-Images

Service Provider (Developer)

Effect: (very) small company can
- „survive success desaster“ (i.e. sudden big demand)
- serve large customer base

Cloud Types

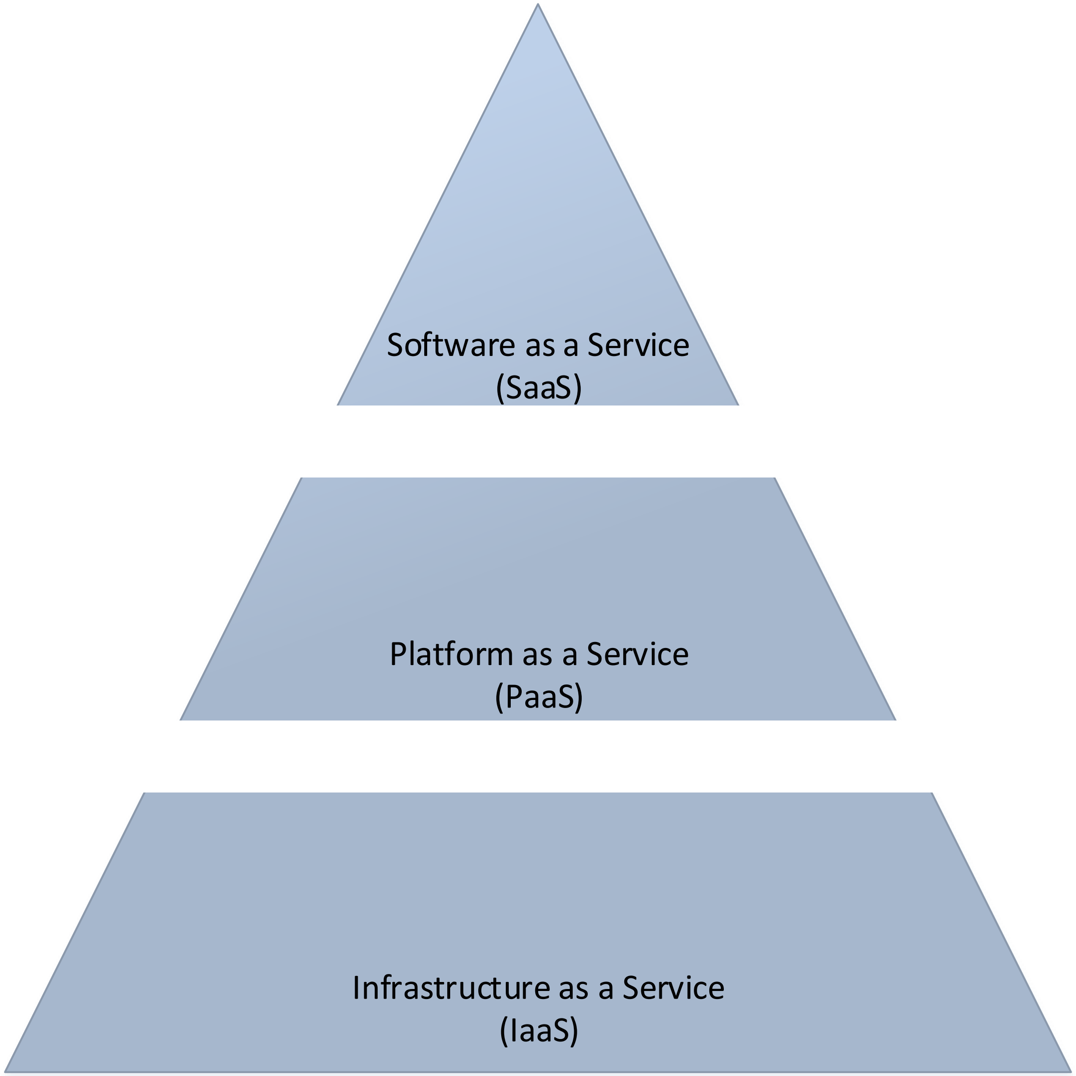
- **Public Cloud:** sells service to anyone on the Internet
  - Observation (by IBM): 70% of IT budget is spent on average for maintenance versus adding new capabilities
  - Goal: Replace fixed IT-costs with variable costs
  - Gartner says: “Organizations are switching from company-owned hardware and software assets to per-use service-based models”
  - Low entry barrier
  - For applications you have to do once
  - For applications you do not have the resources for
  - e.g., Amazon EC2
- **Private Cloud:** used inside a business (e.g., IBM Blue Cloud)
  - Observation: Most systems are oversized, run at 10-20% utilization most of the time
  - Goal: more efficient utilization of resources in a business
- **Community Cloud:** infrastructure shared between several organizations
  - Common concerns (security, compliance, jurisdiction, etc.)
- **Hybrid Cloud:** composition of $\geq$ 2 cloud infrastructures (private / community /public)
  - Clouds remain unique entities
  - Bound together to realize data or application portability
  - Usage: cloud bursting for load balancing

Cloud Computing

Key characteristics

- **„Unbounded scale“**
  - Rapid up- and downscaling possible
  - No more oversized systems that are mostly idle
  - Still, the peak load capacity can be provided
  - Assumption: data center hosts many different apps, only few need peak resources at a given time
- **Reliability**
  - Redundant computers and data centers
- More **location independence**
  - Hoster may operate data centers worldwide
- Often based on **cheap hardware**
- Compared to clusters & Grid: **Slow interconnection** network
- Subscription-based or **Pay-per-use**
- **Three basic (layers) variants, see below**

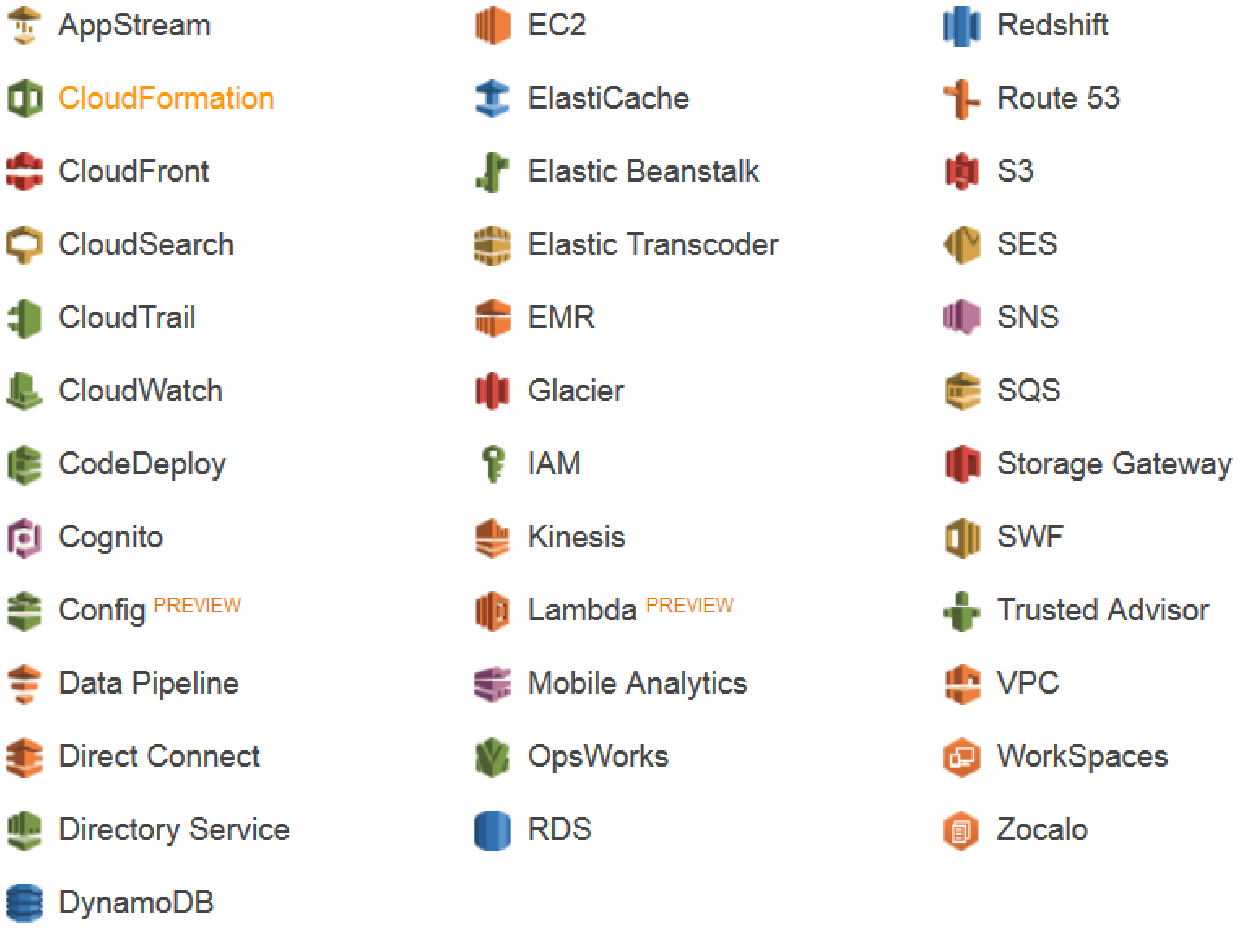
Service Models


Software as a Service (SaaS)

Platform as a Service (PaaS)

Infrastructure as a Service (IaaS)

Service Models: Selected „Products“



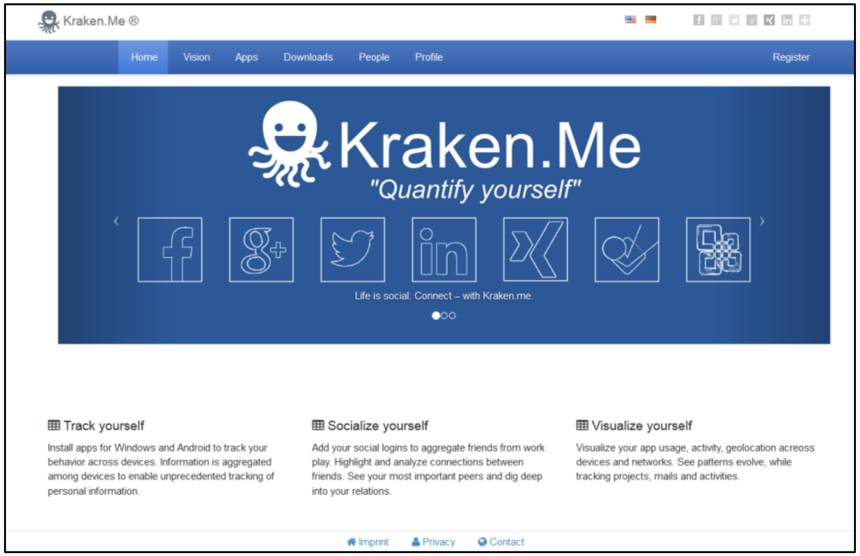
- AppStream
- CloudFormation
- CloudFront
- CloudSearch
- CloudTrail
- CloudWatch
- CodeDeploy
- Cognito
- Config PREVIEW
- Data Pipeline
- Direct Connect
- Directory Service
- DynamoDB
- EC2
- ElastiCache
- Elastic Beanstalk
- Elastic Transcoder
- EMR
- Glacier
- IAM
- Kinesis
- Lambda PREVIEW
- Mobile Analytics
- OpsWorks
- RDS
- Redshift
- Route 53
- S3
- SES
- SNS
- SQS
- Storage Gateway
- SWF
- Trusted Advisor
- VPC
- WorkSpaces
- Zocalo

Infrastructure as a Service (IaaS)

- **Computer Infrastructure** delivery model
  - A platform virtualization environment
    - „virtualization taken a step further“
  - Computing resources, such as processing capacity and storage
  - Pay-for-what-you-use model
    - resembles electricity, fuel, water … consumption → also called **utility computing**
  - Example: you want to run a batch job but you don’t have the infrastructure necessary to run it in a timely manner → use Amazon EC2
- **Providers**
  - **Amazon** EC2, S3 (Elastic Compute Cloud, Simple Storage Service)
    - Virtual server instances with unique IP addresses
    - Blocks of storage on demand
    - API to start, stop, access, configure virtual servers and storage
  - **Rackspace**: cloudServers
  - **FlexiScale**

Infrastructure as a Service (IaaS)

- Example (Kraken.me)
  - Web Application with Real-time User Interaction
  - Database for User Data
  - Processing for Generating Insights
  - API



Track yourself

Install apps for Windows and Android to track your behavior across devices. Information is aggregated among devices to enable unprecedented tracking of personal information.

Socialize yourself

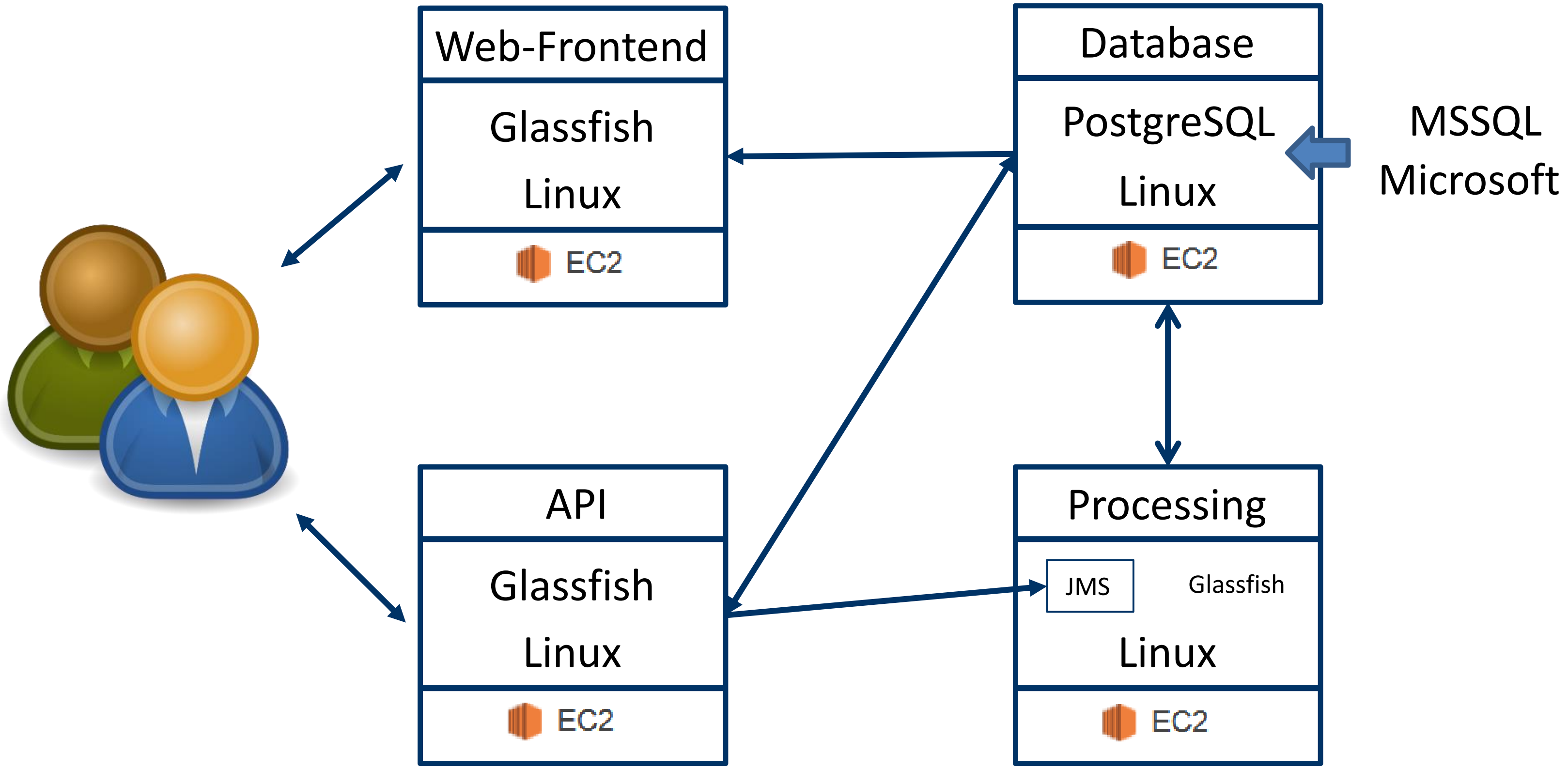
Add your social logins to aggregate friends from work play. Highlight and analyze connections between friends. See your most important peers and dig deep into your relations.

Visualize yourself

Visualize your app usage, activity, geolocation acreoss devices and networks. See patterns evolve, while tracking projects, mails and activities.

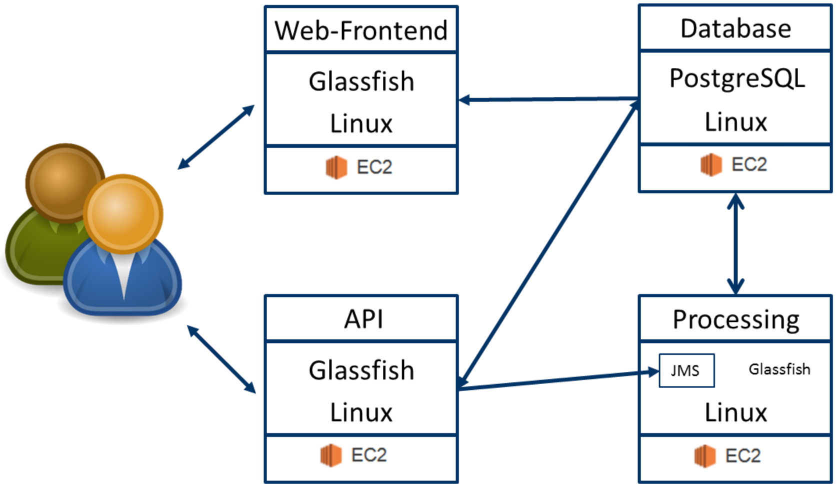
Infrastructure as a Service (IaaS)

User builds / modifies Platform & Apps, Cloud „just“ provides elastic VMs / StorageSpace



Elasticity (IaaS)

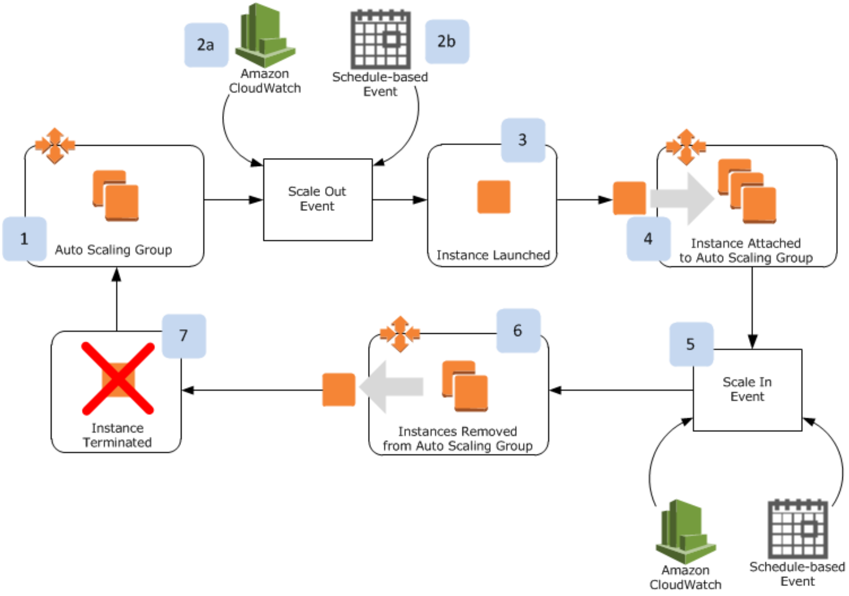
- Larger Instance per EC2
  37 instance types, e.g.:
  - T1.micro (1 vCPU, 0,613 GB)
  - I2.8xlarge (32 vCPU, 244GB)



- Limitations to resizing
  - Storage of Virtual Image
  - Format of Virtual Image
  - Downtime

Elasticity (IaaS)

- Add more instances
  - Scale In / Scale Out
  - Groups of instances with same functionality
  - Granularity
  - Load Balancer / Scheduler



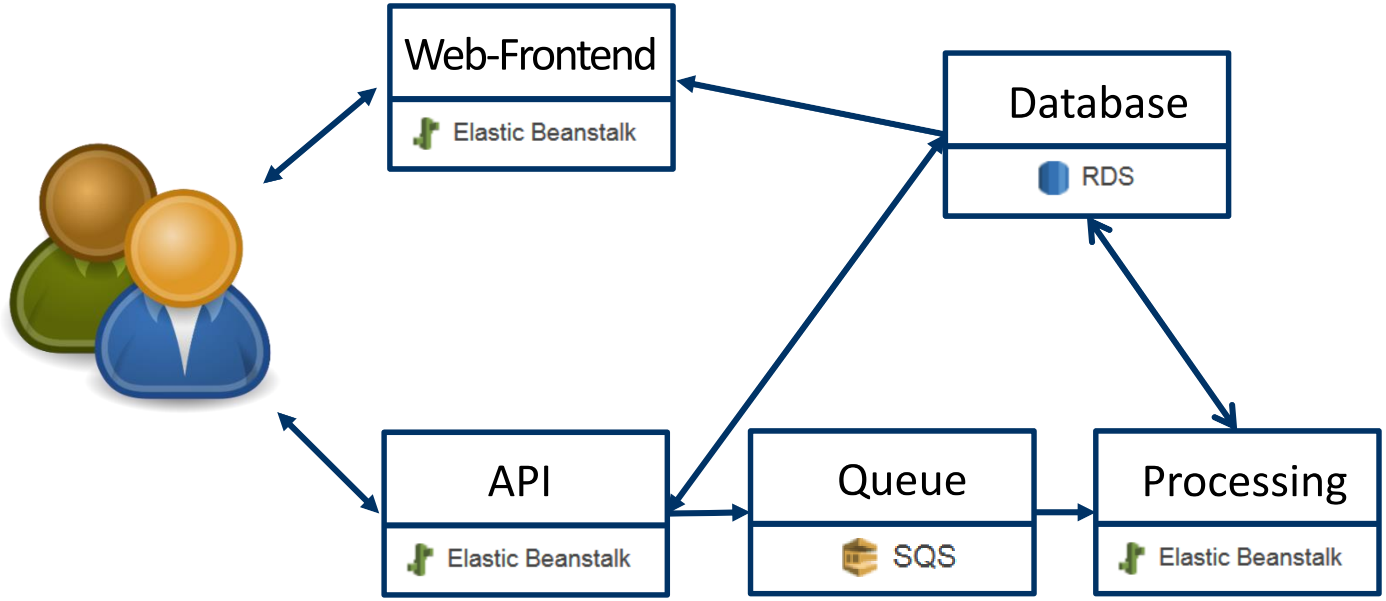
- Load Balance between regions etc.

Platform as a Service (PaaS)

- **Platform** delivery model
  - Platforms are built upon infrastructure
  - Provider is responsible for maintenance of OS & base frameworks
  - Problem: currently no standards for interoperability / data portability
  - Examples
    - You want to start storage services on your network for a large number of files and you do not have the storage capacity → use Amazon S3 (IaaS)
    - You want to host a website and expect many hits, but only for a few days → use Rackspace cloudSites (PaaS)

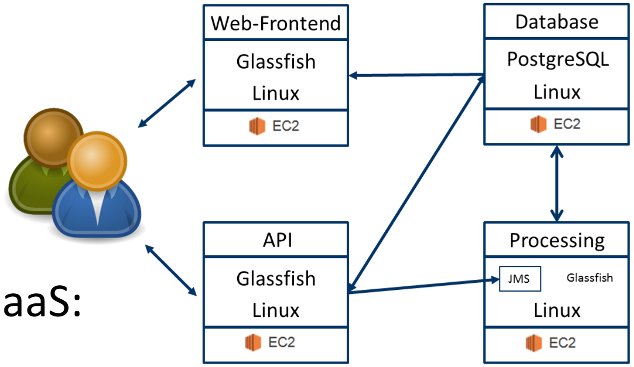
- **Providers:** Google App Engine, Force.com, Rackspace, ...
  - Rackspace: cloudSites, cloudFiles
    - cloudSites: scalable website w/o need to manage dedicated server

Platform as a Service (PaaS)



- Build upon elasticity of IaaS
  - more / larger Instances
  - but *automatically*
- "just" develop application
  - built to take advantage of PaaS

compare to IaaS:



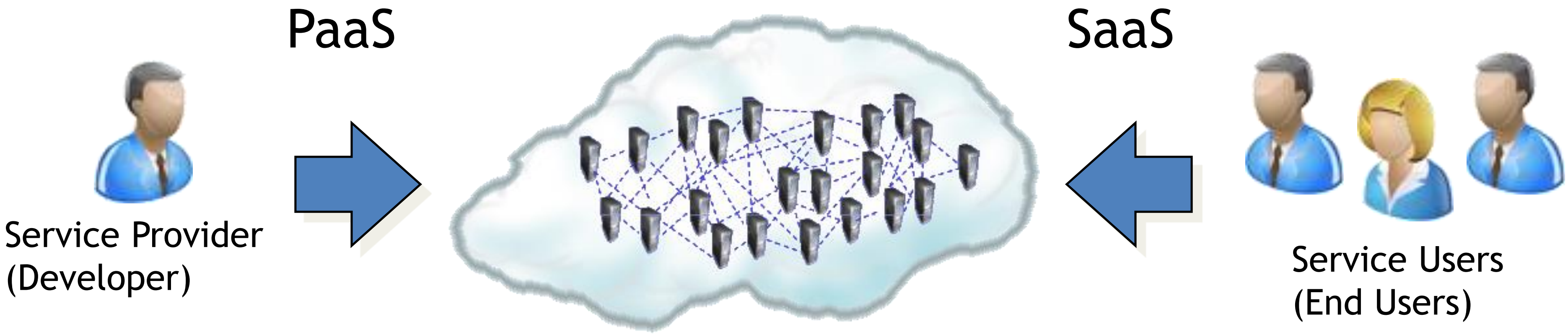
Software as a Service (SaaS)

- Software delivery model
  - No need to manage hardware or software
  - Service delivered through a browser (in most cases)
  - Provider hosts both application and data
  - End user is free to use the service from anywhere
  - Customers use the service on demand
  - Instant Scalability
    - Examples
    - Your email is hosted on an Exchange server in your office and it is very slow → outsource this using Hosted Exchange
    - Your current CRM package is not managing the load or you simply don't want to host it in-house → use a SaaS provider such as Salesforce.com
  - SaaS is a very broad market: many domain-specific applications

Other Services

- Human as a Service (HaaS)
  - Crowdsourcing applications consider humans as resource to execute tasks
  - Very useful for tasks, computers are not good at (e.g. prepare labeled data sets as test/training set for machine learning)
  - Example: Amazon Mechanical Turk

Cloud Computing: Paas vs. SaaS



PaaS

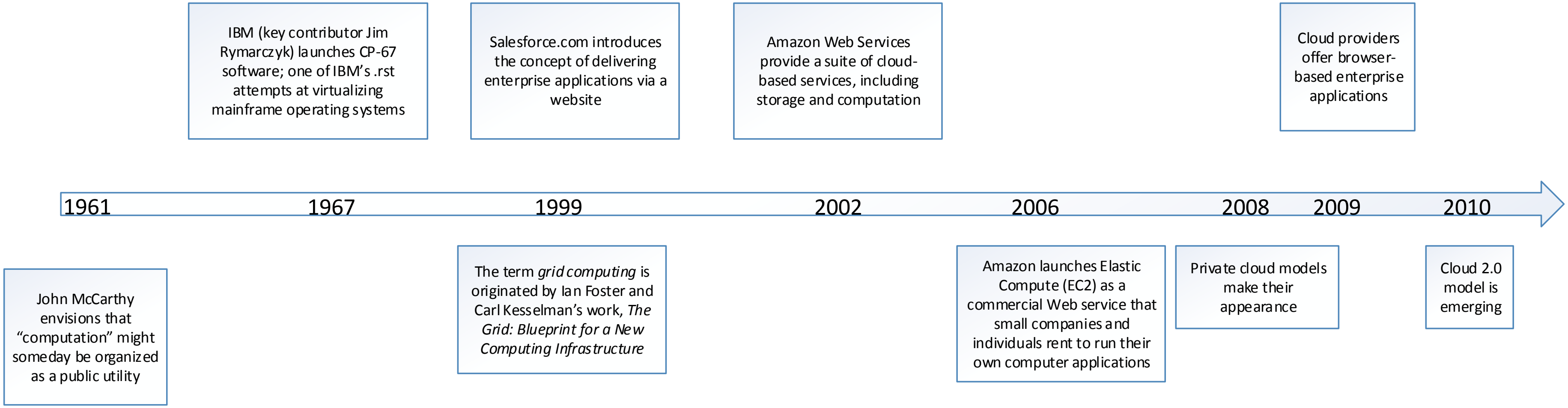
SaaS

Service Provider
(Developer)

Service Users
(End Users)

- PaaS for Developer
  - A set of software & product development tools hosted on the provider's infrastructure
  - Developers create applications on the provider’s platform over the Internet
- SaaS for End User

Short History of Cloud Computing



1961

John McCarthy envisions that "computation" might someday be organized as a public utility

1967

IBM (key contributor Jim Rymarczyk) launches CP-67 software; one of IBM's first attempts at virtualizing mainframe operating systems

1999

Salesforce.com introduces the concept of delivering enterprise applications via a website

The term *grid computing* is originated by Ian Foster and Carl Kesselman's work, *The Grid: Blueprint for a New Computing Infrastructure*

2002

Amazon Web Services provide a suite of cloud-based services, including storage and computation

2006

Amazon launches Elastic Compute (EC2) as a commercial Web service that small companies and individuals rent to run their own computer applications

2008

Private cloud models make their appearance

2009

Cloud providers offer browser-based enterprise applications

2010

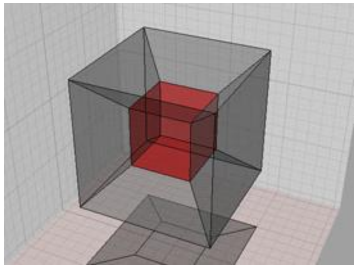
Cloud 2.0 model is emerging

G. Pallis
Cloud computing: the new frontier of internet computing
IEEE Internet Computing
(2010) [14:5: 5562494:70-73]

History: Technology Development

Grid Computing

- Solving large problems with parallel computing
- Made mainstream by Globus Alliance



Utility Computing

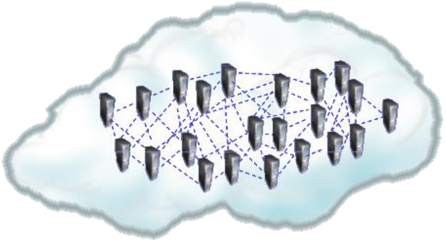
- Offering computing resources as a metered service
- Introduced in late 1990s

Software as a Service

- Network-based subscriptions to applications
- Gained momentum in 2001

Cloud Computing

- Next-generation Internet computing
- Next-generation Data Centers



Grid Computing

- Grid Definition: (Ian Foster)
  - **Coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations**
- Hence, grid is coordinated, aimed at solving problems
    - Motivation to join a grid: more resources available
  - Resources typically provided by everyone and usable by all
- Focus on large scale resource sharing, innovative applications, and high performance
  - This focus distinguishes grid from traditional distributed computing
- Main challenges: authentication, authorization, resource access, and resource discovery

Why ‘Grid’? From Grid to Cloud

Why ‘Grid’?

- Word has several meanings in English (map grid, starting grid, …)
- Relationship to grid computing? (are computers arranged in a grid? ☺)
- Best analogy is **power grid:** distribution of electricity
  - In power grid, there are producers and consumers
  - Grid connects them and lets any consumer use any producer seamlessly
  - Grid computing can be defined in a similar way, with similar goals:
    - Anytime, anywhere, just plug in and you are connected

From Grid to Cloud

- Both share a lot: intention, architecture & technology; problems
- Intentions:
  - Manage large, central facilities
  - Define methods by which consumers discover, request & use resources
  - Provide means for efficient parallel execution of software

Grid vs. Cloud Computing: Differences

- Business Model / **accessibility**
  - Grids are only accessible within pre-defined multi-organizations
    - Participants share resources, i.e., most participants also offer resources
    - Each individual organization maintains full control of their resources
  - User/Hoster-Roles in Cloud Computing
    - Cloud computing hoster offers its service to anyone on the Internet
- Computing Model / **application types**
  - Grids are mostly limited to **batch processing**
  - Clouds support **interactive applications** (and also batch)
    - Good support for “web-faced” applications met demand of many e-commerce applications -> great market success
- Programming Model / **job/application submission**
  - Grid: builds mostly on **source code distribution**
    - Send archive with source code to server in Grid
    - Job will be compiled and executed there
  - Cloud: builds on **virtualization**

Grid vs. Cloud Computing

- **Interoperability**
  - Grid
    - Wide adoption of the OGF (Open Grid Forum) architecture/protocol standards
    - Wide adoption of the Globus toolkit
  - Cloud
    - Currently no interoperability
    - Big players do not seem to be interested much in standards
      - Amazon, Microsoft, Salesforce, Google
    - Open Cloud Consortium
      - Members: Aerospace, Cisco, Infoblox, Nasa, Yahoo, ...
      - ... but the big players are not here

    → **Vendor lock-in!**
- **Elasticity**
  - Cloud: You pay for the resources you use (**pay-per-use**)
  - Grid: Accounting across organizations never worked

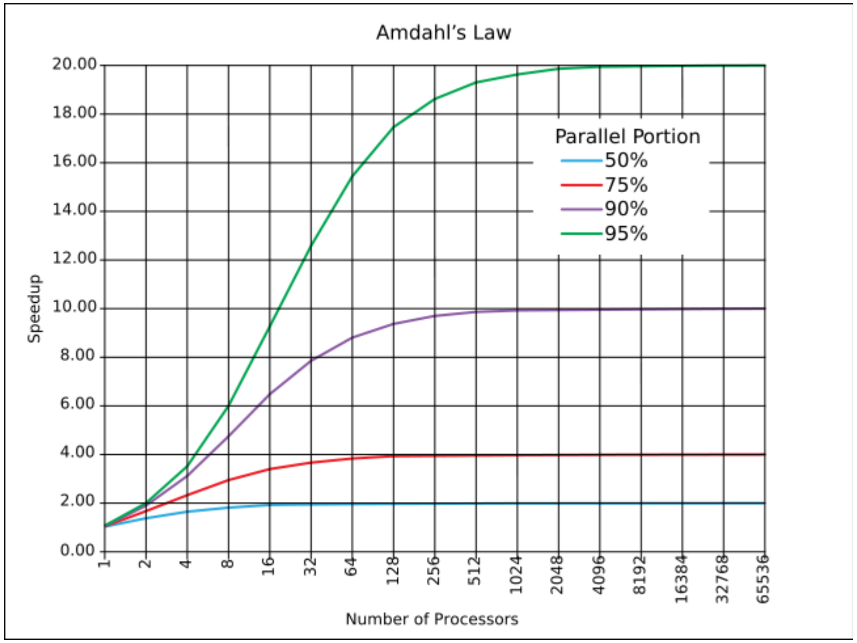
Grid vs. Cloud Computing

- Amdahl‘s Law:

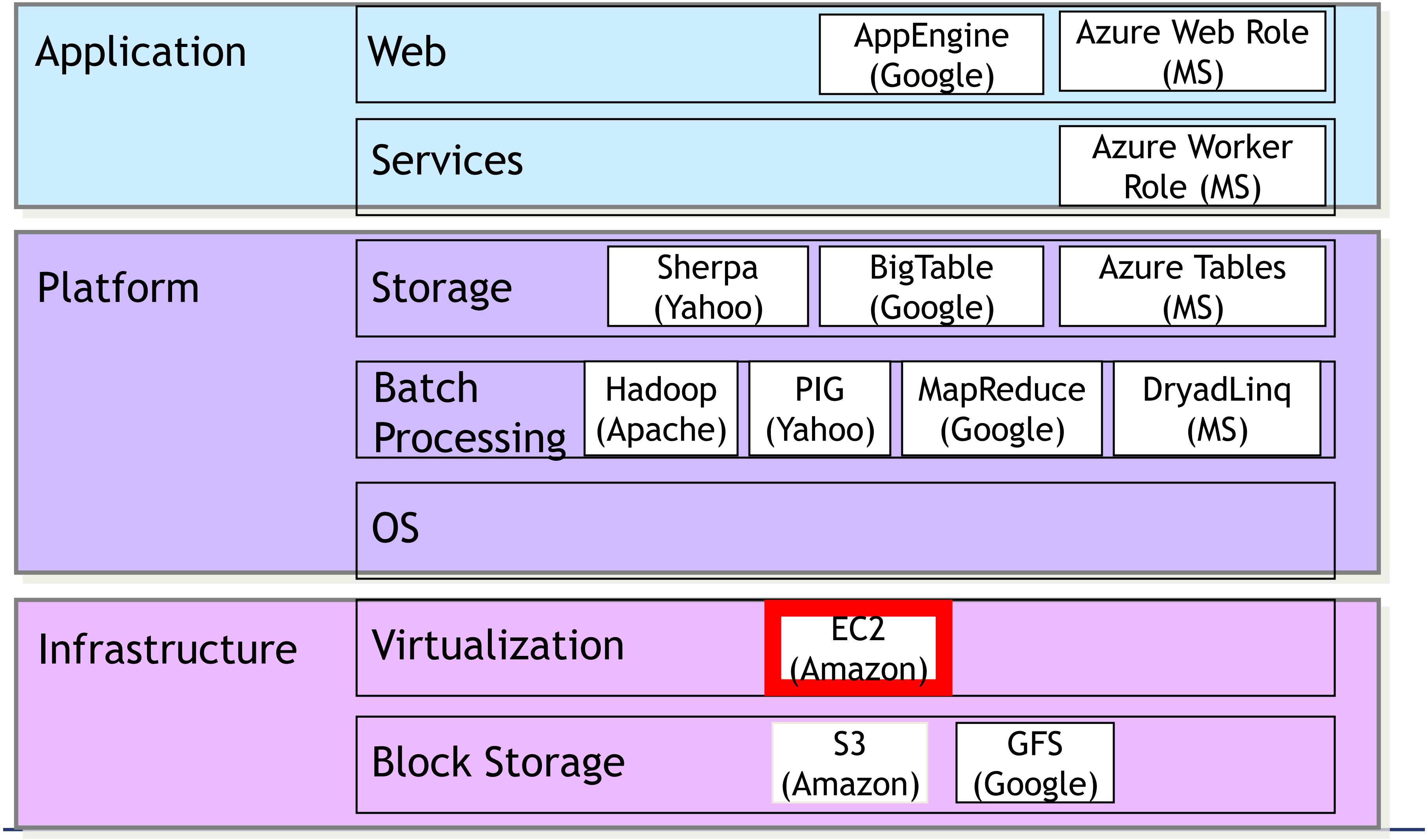
  s … speedup
  P … number of parallel processes
  o(P) … communication overhead
  a … sequential fraction of program

  - Max. expected speedup of overall system, assuming: *not everything* can be parallelized
  - Expresses relationship btw. sequential algorithm and parallelized version
  - The **sequential fraction** of a program (in terms of run time) determines the **upper bound for speedup**! - still true, but only if we look at solving <u>one</u> problem
- Grids
  - Parallel computers with fast interconnection networks -> low *o(P)*
  - Solve often **only 1 problem**
- Clouds
  - Inexpensive hardware, „slow“ interconnects -> high *o(P)*
  - Either *a* is really small (cf. MapReduce)
  - or mostly: **solve n problems** in parallel

$$s = \frac{1}{a + o(P) + \frac{1-a}{P}} \leq \frac{1}{a}$$



The „Cloud Stack“

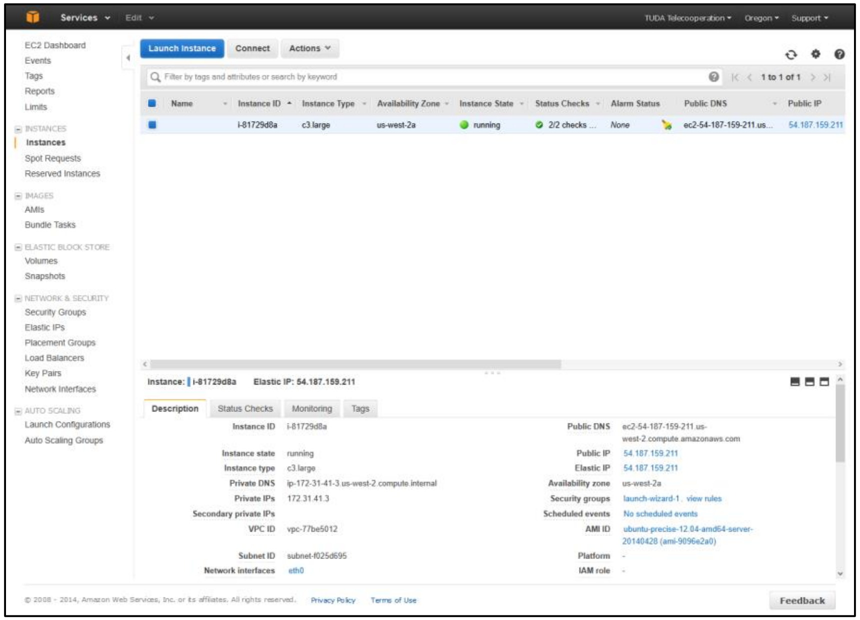


Application	Web	AppEngine (Google)	Azure Web Role (MS)		
	Services	Azure Worker Role (MS)			
Platform	Storage	Sherpa (Yahoo)	BigTable (Google)	Azure Tables (MS)	
	Batch Processing	Hadoop (Apache)	PIG (Yahoo)	MapReduce (Google)	DryadLinq (MS)
	OS				
Infrastructure	Virtualization	EC2 (Amazon)			
	Block Storage	S3 (Amazon)	GFS (Google)		

Amazon EC2

Amazon Elastic Cloud Computing

- Introduced 2006 based on internal project
  - >1.000.000 physical servers
- EC2 is not much more than “VM hosting”
  - Amazon Machine Image (AMI) = Encrypted VM image, stored in S3 repository
  - Pre-configured templates provided
- Instances: *controlled* using a Web Service interface, *accessed* via RDP or SSH



Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS	Public IP
	i-81729d8a	c3.large	us-west-2a	running	2/2 checks ...	None	ec2-54-187-159-211.us...	54.187.159.211

Instance: i-81729d8a Elastic IP: 54.187.159.211

Instance ID	i-81729d8a	Public DNS	ec2-54-187-159-211.us-west-2.compute.amazonaws.com
Instance state	running	Public IP	54.187.159.211
Instance type	c3.large	Elastic IP	54.187.159.211
Private DNS	ip-172-31-41-3.us-west-2.compute.internal	Availability zone	us-west-2a
Private IPs	172.31.41.3	Security groups	launch-wizard-1 . view rules
Secondary private IPs		Scheduled events	No scheduled events
VPC ID	vpc-77be5012	AMI ID	ubuntu-precise-12.04-amd64-server-20140428 (ami-9096e2a0)
Subnet ID	subnet-f025d695	Platform	-
Network interfaces	eth0	IAM role	-

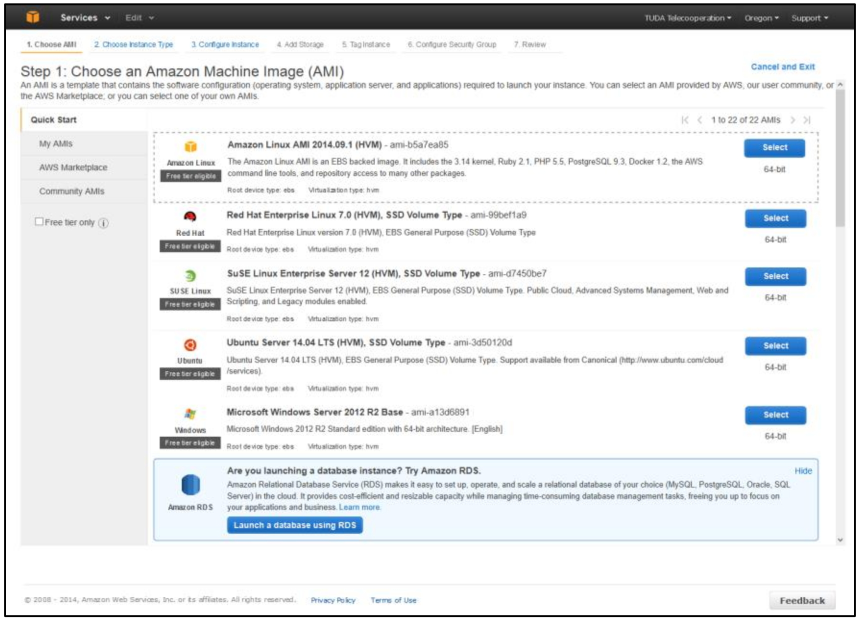
© 2008 - 2014, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

Amazon EC2: Block Store, API

Amazon EC2 API:

- Running
  - RegisterImage / DeregisterImage
  - RunInstances: launch the specified number of instances
  - TerminateInstances, RebootInstances
- Networking
  - AllocateAddress: acquire an elastic IP address; AssociateAddress: associate elastic IP address with instance
- Storage
  - CreateVolume / DeleteVolume
  - AttachVolume: mount an EBS volume
  - CreateSnapshot / DeleteSnapshot

Amazon EC2



Step 1: Choose an Amazon Machine Image (AMI)

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. You can select an AMI provided by AWS, our user community, or the AWS Marketplace; or you can select one of your own AMIs.

Amazon Linux AMI 2014.09.1 (HVM) - ami-b5a7ea85
The Amazon Linux AMI is an EBS backed image. It includes the 3.14 kernel, Ruby 2.1, PHP 5.5, PostgreSQL 9.3, Docker 1.2, the AWS command line tools, and repository access to many other packages.
Root device type: ebs   Virtualization type: hvm

Red Hat Enterprise Linux 7.0 (HVM), SSD Volume Type - ami-99bef1a9
Red Hat Enterprise Linux version 7.0 (HVM), EBS General Purpose (SSD) Volume Type
Root device type: ebs   Virtualization type: hvm

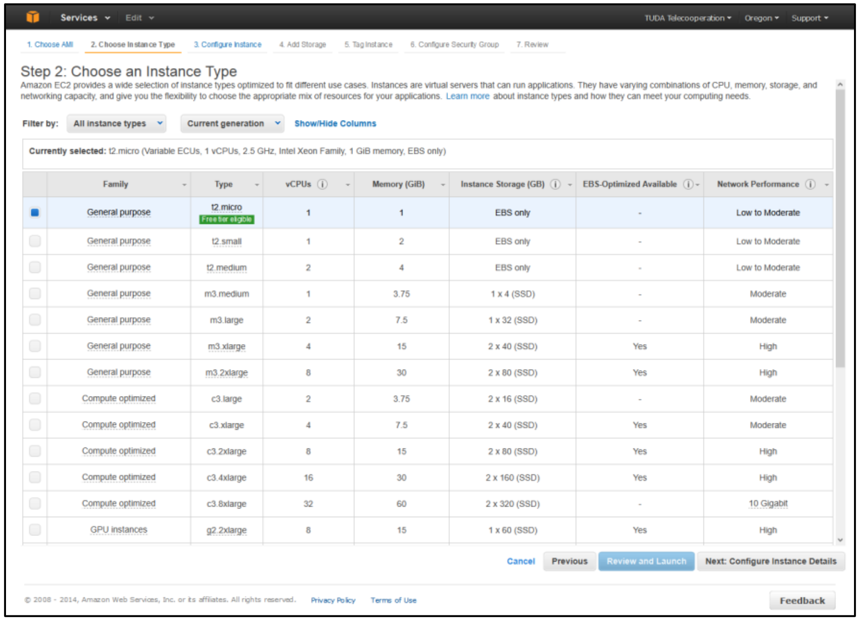
SuSE Linux Enterprise Server 12 (HVM), SSD Volume Type - ami-d7450be7
SuSE Linux Enterprise Server 12 (HVM), EBS General Purpose (SSD) Volume Type. Public Cloud, Advanced Systems Management, Web and Scripting, and Legacy modules enabled.
Root device type: ebs   Virtualization type: hvm

Ubuntu Server 14.04 LTS (HVM), SSD Volume Type - ami-3d50120d
Ubuntu Server 14.04 LTS (HVM), EBS General Purpose (SSD) Volume Type. Support available from Canonical (http://www.ubuntu.com/cloud/services).
Root device type: ebs   Virtualization type: hvm

Microsoft Windows Server 2012 R2 Base - ami-a13d6891
Microsoft Windows 2012 R2 Standard edition with 64-bit architecture. [English]
Root device type: ebs   Virtualization type: hvm

Are you launching a database instance? Try Amazon RDS.
Amazon Relational Database Service (RDS) makes it easy to set up, operate, and scale a relational database of your choice (MySQL, PostgreSQL, Oracle, SQL Server) in the cloud. It provides cost-efficient and resizable capacity while managing time-consuming database management tasks, freeing you up to focus on your applications and business. Learn more.

© 2008 - 2014, Amazon Web Services, Inc. or its affiliates. All rights reserved.



Step 2: Choose an Instance Type

Amazon EC2 provides a wide selection of instance types optimized to fit different use cases. Instances are virtual servers that can run applications. They have varying combinations of CPU, memory, storage, and networking capacity, and give you the flexibility to choose the appropriate mix of resources for your applications. Learn more about instance types and how they can meet your computing needs.

Filter by: All instance types   Current generation

Currently selected: t2.micro (Variable ECUs, 1 vCPUs, 2.5 GHz, Intel Xeon Family, 1 GiB memory, EBS only)

	Family	Type	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance
■	General purpose	t2.micro	1	1	EBS only	-	Low to Moderate
□	General purpose	t2.small	1	2	EBS only	-	Low to Moderate
□	General purpose	t2.medium	2	4	EBS only	-	Low to Moderate
□	General purpose	m3.medium	1	3.75	1 x 4 (SSD)	-	Moderate
□	General purpose	m3.large	2	7.5	1 x 32 (SSD)	-	Moderate
□	General purpose	m3.xlarge	4	15	2 x 40 (SSD)	Yes	High
□	General purpose	m3.2xlarge	8	30	2 x 80 (SSD)	Yes	High
□	Compute optimized	c3.large	2	3.75	2 x 16 (SSD)	-	Moderate
□	Compute optimized	c3.xlarge	4	7.5	2 x 40 (SSD)	Yes	Moderate
□	Compute optimized	c3.2xlarge	8	15	2 x 80 (SSD)	Yes	High
□	Compute optimized	c3.4xlarge	16	30	2 x 160 (SSD)	Yes	High
□	Compute optimized	c3.8xlarge	32	60	2 x 320 (SSD)	-	10 Gigabit
□	GPU instances	g2.2xlarge	8	15	1 x 60 (SSD)	Yes	High

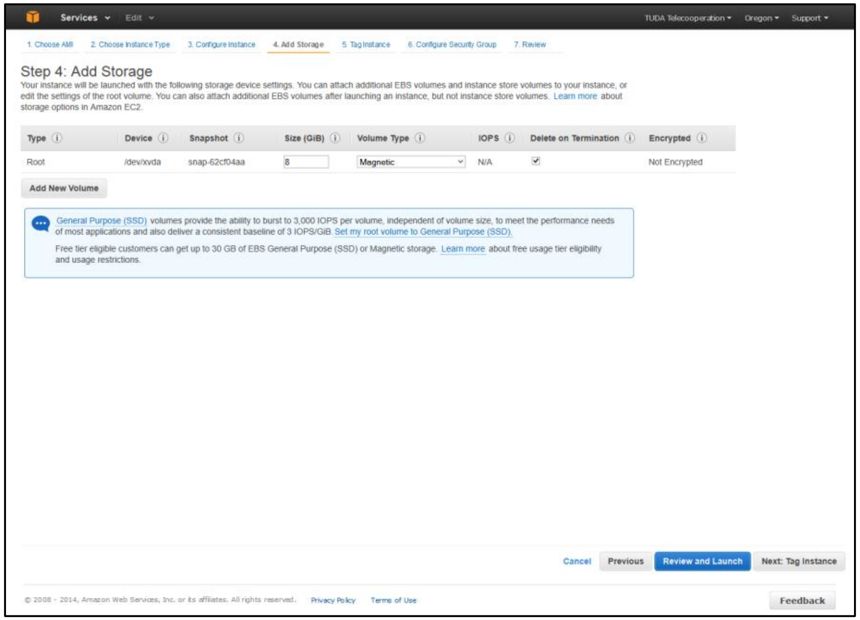
© 2008 - 2014, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Amazon EC2

- The local storage of instances is non-persistent
  - AMI Images are stored in Amazon S3

Amazon Elastic Block Store

- Raw block device that can be mounted (“attached”) by an instance (one instance at a time)
- Persistence beyond the lifetime of an instance, snapshots can be stored on S3
- Block store API functions (accessed via SOAP or REST):
  - CreateVolume / DeleteVolume; DescribeVolume: get info. about volume (size, source snapshot, creation time, status)
  - AttachVolume / DetachVolume
  - CreateSnapshot / DeleteShapshot; DescribeSnapshots: get information about snapshots



Step 4: Add Storage

Your instance will be launched with the following storage device settings. You can attach additional EBS volumes and instance store volumes to your instance, or edit the settings of the root volume. You can also attach additional EBS volumes after launching an instance, but not instance store volumes. Learn more about storage options in Amazon EC2.

Type	Device	Snapshot	Size (GiB)	Volume Type	IOPS	Delete on Termination	Encrypted
Root	/dev/xvda	snap-62cf04aa	8	Magnetic	N/A	☑	Not Encrypted

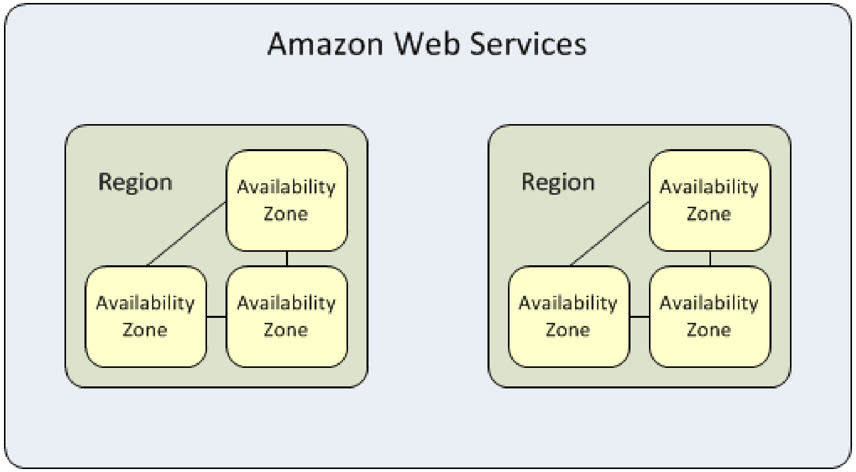
General Purpose (SSD) volumes provide the ability to burst to 3,000 IOPS per volume, independent of volume size, to meet the performance needs of most applications and also deliver a consistent baseline of 3 IOPS/GiB. Set my root volume to General Purpose (SSD).

Free tier eligible customers can get up to 30 GB of EBS General Purpose (SSD) or Magnetic storage. Learn more about free usage tier eligibility and usage restrictions.

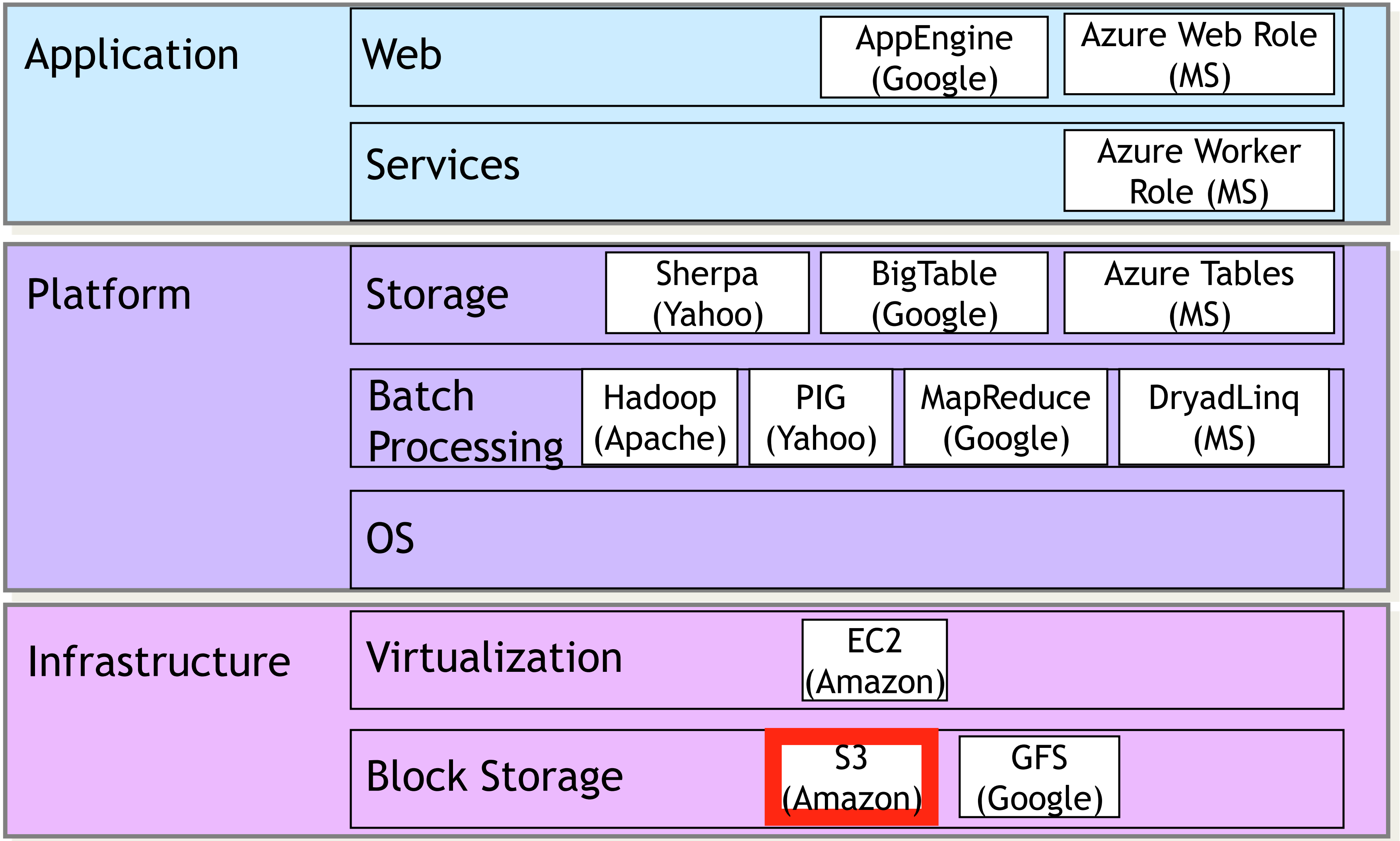
© 2008 - 2014, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Amazon EC2

- Instances are Availability Zones
  - Describe the location of a datacenter as geographic region (e.g., us-east-1a)
  - Placement of instances can be controlled
  - Distribution of instances can be used for fault tolerance



The „Cloud Stack“



- Application
  - Web: AppEngine (Google), Azure Web Role (MS)
  - Services: Azure Worker Role (MS)
- Platform
  - Storage: Sherpa (Yahoo), BigTable (Google), Azure Tables (MS)
  - Batch Processing: Hadoop (Apache), PIG (Yahoo), MapReduce (Google), DryadLinq (MS)
  - OS
- Infrastructure
  - Virtualization: EC2 (Amazon)
  - Block Storage: S3 (Amazon), GFS (Google)

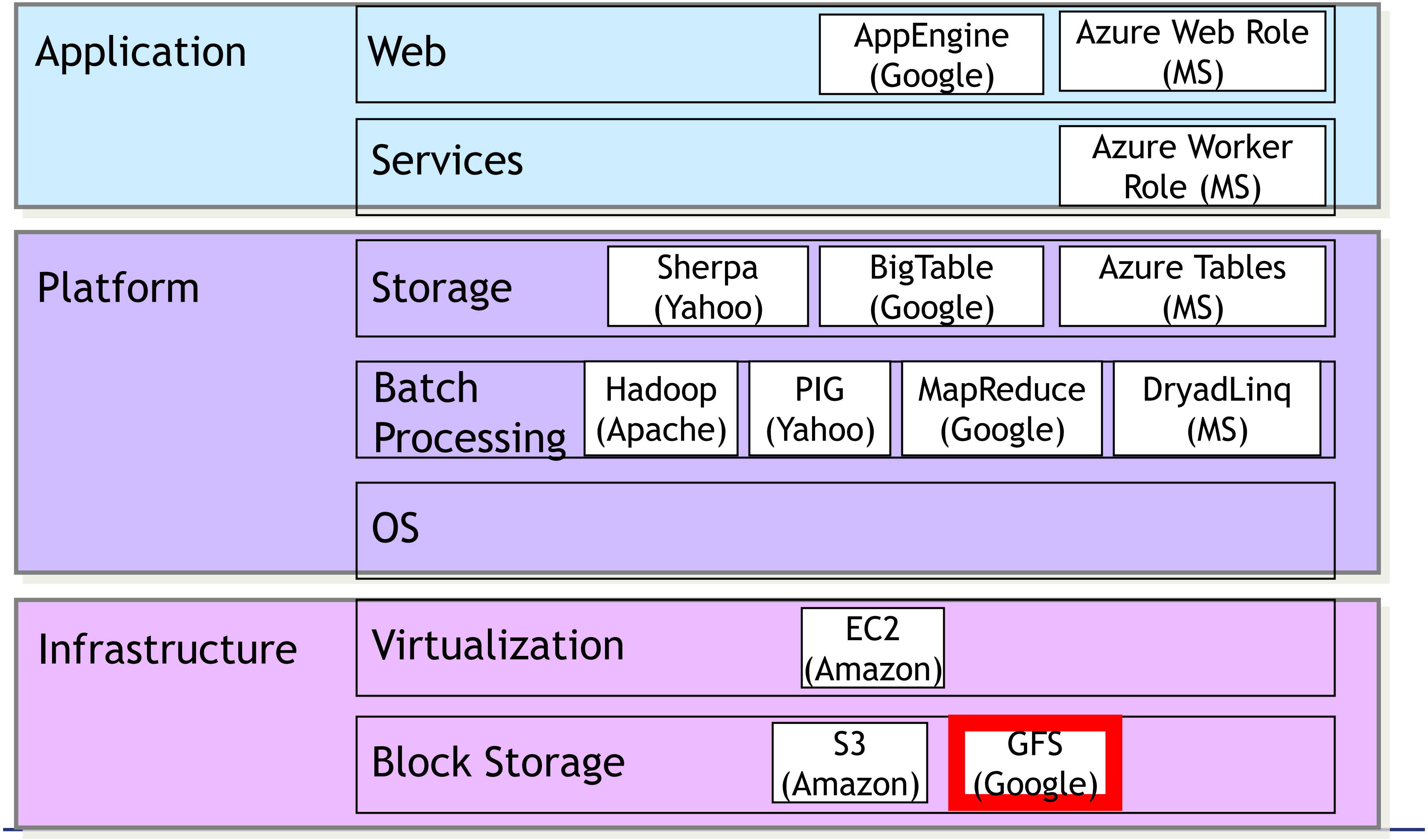
Amazon S3

- Amazon Simple Store Service (S3)
  - Based on REST or SOAP APIs, also browser-based management tools available
  - Organized in buckets
    - bucket = “drive” in which files can be stored
  - ACLs (access control lists) → share parts of the bucket (in group, in public)
  - Other services build on S3, such as EC2
- Example: Creating a Bucket

```
PUT /onjava HTTP/1.1
Content-Length: 0
User-Agent: jClientUpload
Host: s3.amazonaws.com
Date: Sun, 05 Aug 2007 15:33:59 GMT
Authorization: AWS 15B4D3461F177624206A:YFhSWKDg3qDnGbV7JCnkfdz/IHY=
```

- Authorization: AWS <AWSAccessKeyID>:<Signature>
  - AWSAccessKeyID: customer ID
  - Signature: SHA1 hash of message, encrypted with customer’s private key

The „Cloud Stack“



Application

Web

AppEngine (Google)

Azure Web Role (MS)

Services

Azure Worker Role (MS)

Platform

Storage

Sherpa (Yahoo)

BigTable (Google)

Azure Tables (MS)

Batch Processing

Hadoop (Apache)

PIG (Yahoo)

MapReduce (Google)

DryadLinq (MS)

OS

Infrastructure

Virtualization

EC2 (Amazon)

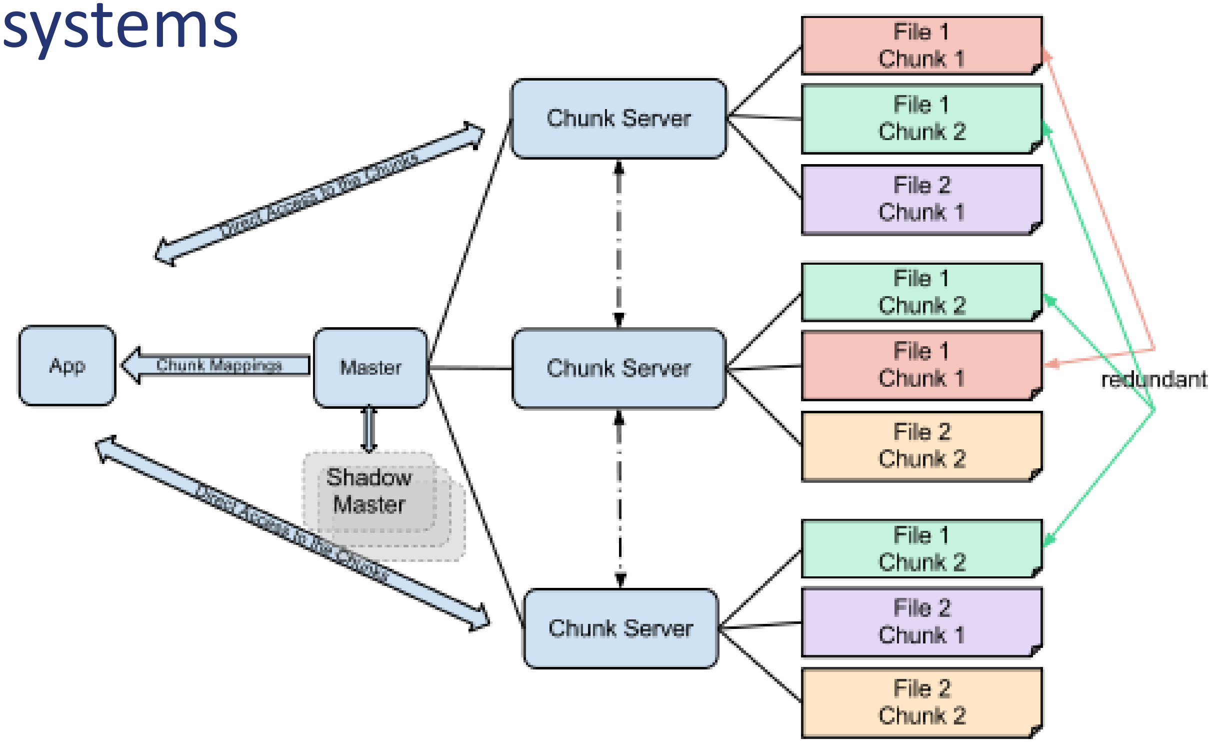
Block Storage

S3 (Amazon)

GFS (Google)

Google Filesystem (GFS)

- Shares many of the same goals as previous distributed file systems
  - Performance
  - Scalability
  - Reliability
  - Availability

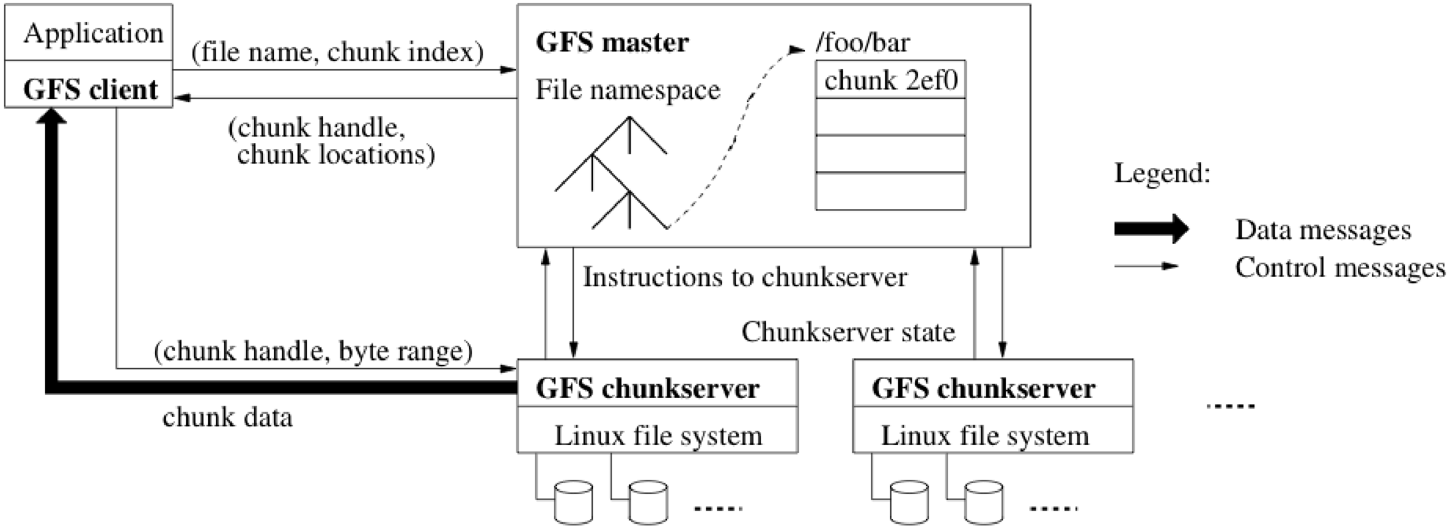


Google Filesystem (GFS)

1. Component failures are the norm rather than the exception
   - Quantity and quality of components virtually guarantees failures
   - Constant monitoring, error detection, fault tolerance, and automatic recovery are integral to the system
2. Huge files (by traditional standards)
   - Multi GB files are common
   - I/O operations and blocks sizes must be revisited
   - Number of files is limited
3. Most files are mutated by appending new data
   - Focus for performance optimization and atomicity guarantees
   - Concurrent Appends possible
   - Random file modifications are supported but inefficient
4. Co-designing the applications and APIs benefits overall system by increasing flexibility
5. Two main types of read: large streaming reads and small random reads
6. High sustained bandwidth more important than low latency

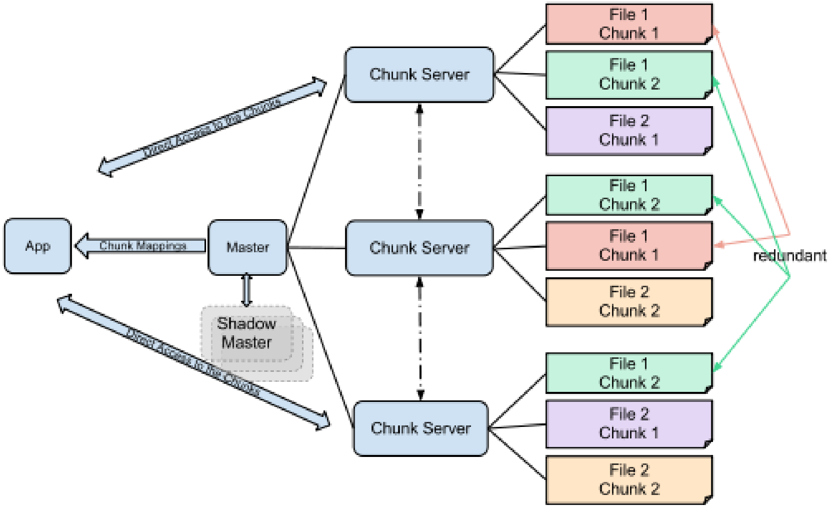
GFS: Architecture

- GFS cluster consists of
  - A single master
  - Multiple chunkservers
  - Is accessed by multiple clients
- Files organized hierarchically in directories & identified by pathnames



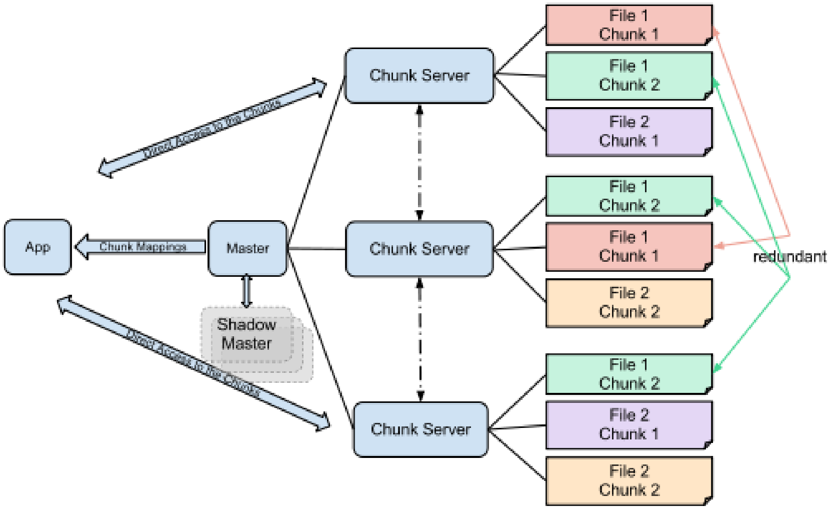
GFS: Master

- Maintains metadata
  - names space, access control info, file to chunk mappings, chunk locations, etc.
  - Kept in memory
- Client contacts Master to get chunk locations, then deals directly w/ chunkservers
  - Master is not a bottleneck for reads/writes
- Single master
  - Simplifies design
  - Helps make sophisticated chunk placement & replication decisions (global knowledge)
  - ‘Shadow masters’ keep replica of master state & log, details later



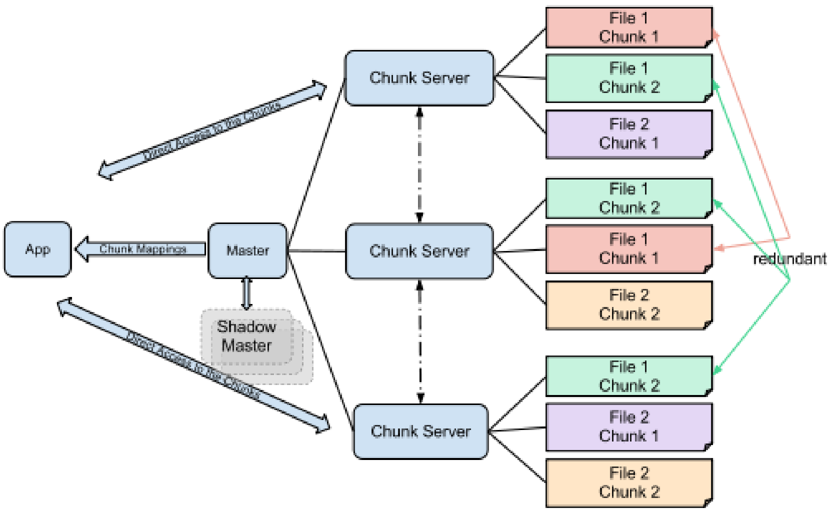
GFS: Chunks

- Files are divided into fixed-size chunks
  - 64 MB default
- Each chunk has a immutable globally unique 64-bit chunk-handle
  - Handle is assigned by the master at chunk creation
- Advantages
  - Reduced client / master interaction
  - Cache all chunk locations for mulit-TB working sets
  - Reduced size of metadata
- But: Hotspots possible



GFS: Chunkservers

- Chunks are stored as files in local Linux file system
- Each chunk is replicated on 3 (default) servers
- Heartbeat between master and chunk server
- Do not cache files (done by file system)



GFS: Clients

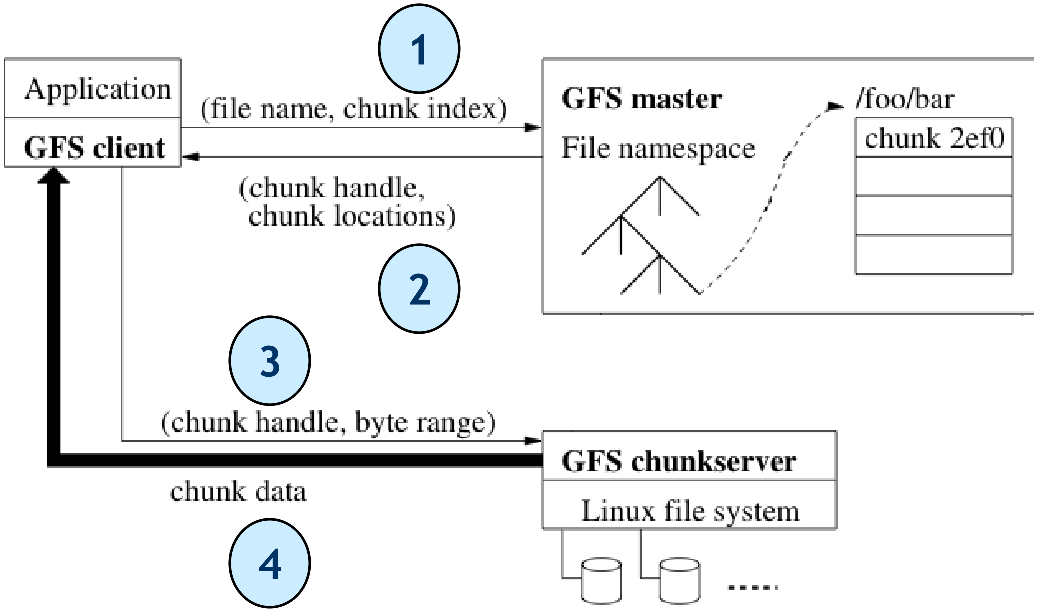
- GFS does not implement a standard API such as POSIX
- Linked as library into apps
- Communicates with master and chunkservers for reading and writing
  - Master interactions only for metadata
  - Chunkserver interactions for data
- Cache metadata
  - Do not cache file data (no coherence issues)

GFS: Operations

- Standard: create, delete, open, close, read, and write files
- Snapshot
  - Creates a copy of a file or a directory tree at low cost
- Record Append
  - Allows multiple clients to append data to the same file concurrently
  - Guaranteeing the atomicity of each individual client’s append
  - Useful for implementing multi-way merge results
  - Useful for producer-consumer queues that many clients can simultaneously append to without additional locking

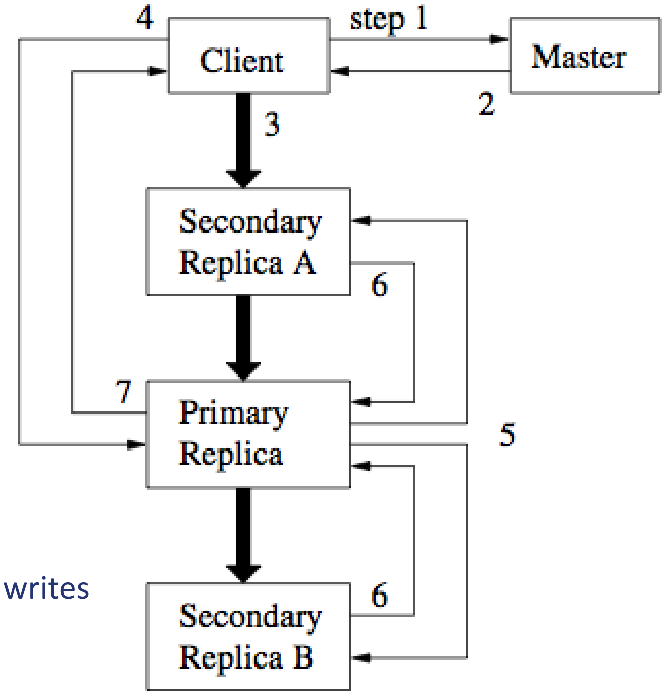
GFS: Read

1. Client sends Master a request containing file name and chunk index (= byte offset / 64MB)
2. Master replies with **chunk handle** and **locations of replicas**
3. Client sends request to one replica (closest one)
4. Client reads data from chunkserver



GFS: Write

1. *Client* asks *master* for *lease holder* (see below) of chunk to write to
2. Reply: identity of the *primary* and locations of *secondary* replicas
3. *Client* pushes data to all replicas on *chunkservers* (directly, not as depicted)
4. Once all the *chunkservers* have acknowledged receiving the data, client sends a write request to the *primary*
   - Request identifies the other replicas
   - Primary assigns consecutive serial numbers → serialization of concurrent writes (an atomic change to the file is called a **mutation**)
   - Data in local write buffer is ordered according to serial no. order
5. *Primary* forwards write request to all *secondary* replicas
   - Data is ordered according to serial number order
6. *Secondaries* all reply to *primary* indicating completion
7. *Primary* replies to *client*



GFS: Atomic Record Append

- Problem
  - In a traditional write, the client specifies offset at which data is to be written
  - Concurrent writes to the same region are not serializable: the region may end up containing data fragments from multiple clients

- Solution: **Atomic Record Append**
  - Client specifies only the data to write
  - GFS chooses and returns the offset it writes to
  - No need for a distributed lock manager
    - GFS chooses the offset, not the client
  - Follows similar control flow as Write
    - Primary tells secondary replicas to append at the same offset
    - Data must be written at the same offset for all chunk replicas for success to be reported

GFS Fault Tolerance: Chunkservers

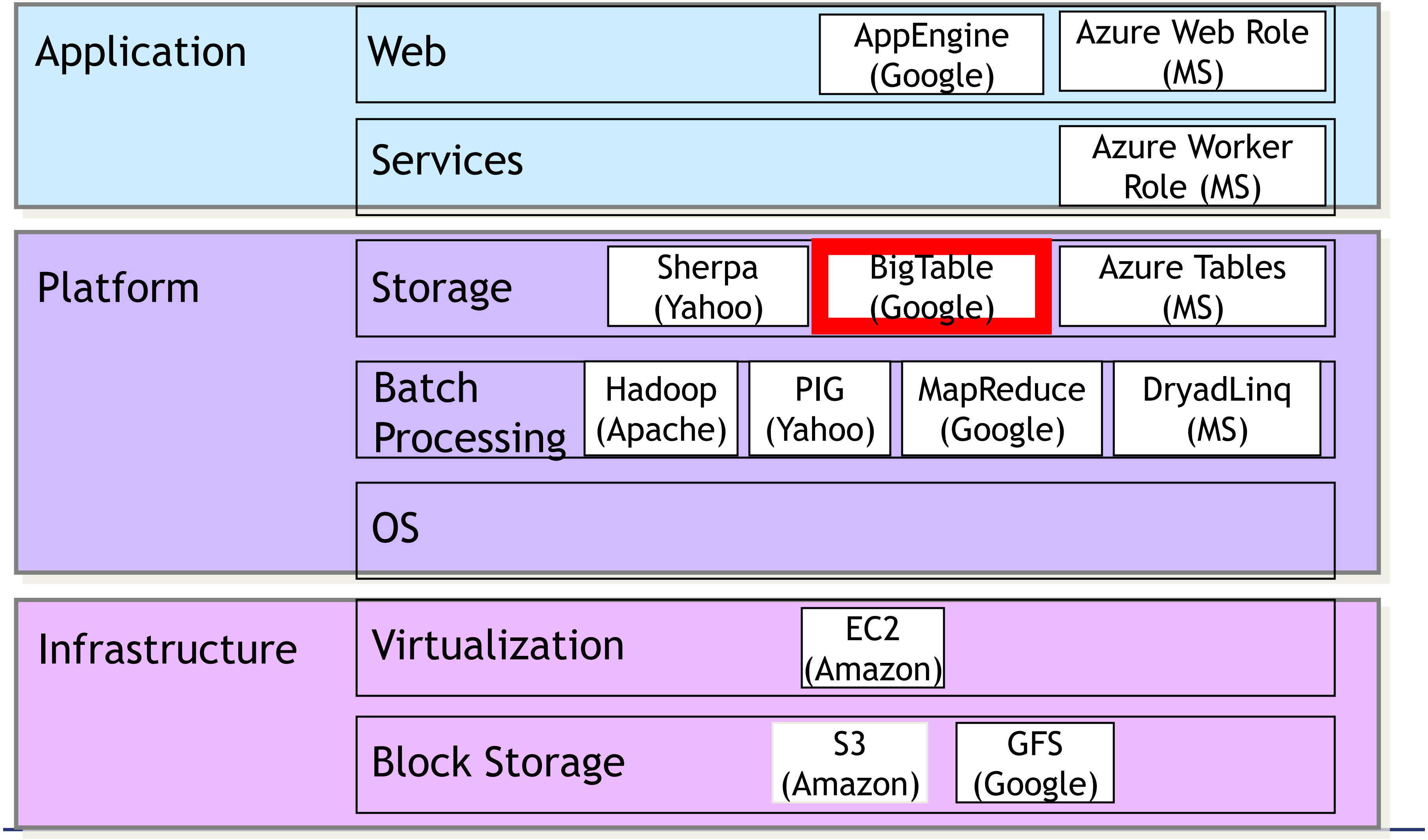
- Purpose
  - Maintain consistent *mutation order* across replicas by identifying primary copy
  - De-allocation of resources in case of a failure
- **Chunkserver: Failure Recovery**
  - Write, 1st step: Client asks Master for Lease Holder
    - If no primary assigned for specified chunk → Master issues new **Lease**
    - Lease has an initial timeout of 60 seconds
    - As long as mutation goes on, primary periodically requests extensions from Master
  - Now, if the master loses communication with a primary:
    - Master can safely grant a new lease to another replica after the old lease has expired
  - Now, if the faulty chunk server comes back:
    - Master maintains version number for each chunk
    - Allows detecting stale replicas

GFS Fault Tolerance: Master

- **Operation Log**
  - Record of all critical metadata changes
  - Stored on Master and replicated on other machines
  - Defines order of concurrent operations
  - Changes not visible to clients until metadata changes are persistent
  - Master checkpoints its state whenever the log grows beyond a certain size
    - Checkpoint is a compact B-tree (used for namespace lookup)
- **Master Recovery**
  - Read latest complete checkpoint
  - Replay operation log
- **Replication: Shadow Masters**
  - Help providing read-only access while primary master is up
  - … and even when primary master is down

http://static.googleusercontent.com/media/research.google.com/en//archive/gfs-sosp2003.pdf

The „Cloud Stack“



Application
- Web: AppEngine (Google), Azure Web Role (MS)
- Services: Azure Worker Role (MS)

Platform
- Storage: Sherpa (Yahoo), BigTable (Google), Azure Tables (MS)
- Batch Processing: Hadoop (Apache), PIG (Yahoo), MapReduce (Google), DryadLinq (MS)
- OS

Infrastructure
- Virtualization: EC2 (Amazon)
- Block Storage: S3 (Amazon), GFS (Google)

BigTable (Google)

BigTable: distributed storage system for managing **structured data**

- Lots of (semi-)structured data at Google
  - **URLs:** Contents, crawl metadata, links, anchors, pagerank, ...
  - **Per-user data:** User preference settings, recent queries/search results, ...
  - **Geo-locations:** places (shop, restaurant …), roads, satellite img. data, user annotations ...
- Designed to scale to a **very large size**
  - Petabytes of data across thousands of servers
  - Billions of URLs, many versions/page (~20K/version)
  - Hundreds of millions of users, thousands or q/sec
  - Many TB of satellite image data
- Scale is too large for most SQL databases
  - Even if it weren't, licensing costs would be very high
  - Low-level storage optimizations help performance significantly
  - Much harder to do when running on top of a database layer

http://www.cs.cornell.edu/courses/cs6464/2009sp/lectures/17-bigtable.pdf
http://static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf

BigTable: Goals

- Want asynchronous processes to be continuously updating different pieces of data
  - Want access to most current data at any time
- Need to support …:
  - Very high read/write rates (millions of op's per second)
  - Efficient scans over all or interesting subsets of data
  - Efficient joins of large one-to-one and one-to-many datasets
- Often want to examine data changes over time
  - E.g. Contents of a web page over multiple crawls
- Fault-tolerant, persistent
- Self-managing
  - Servers can be added/removed dynamically
  - Servers adjust to load imbalance

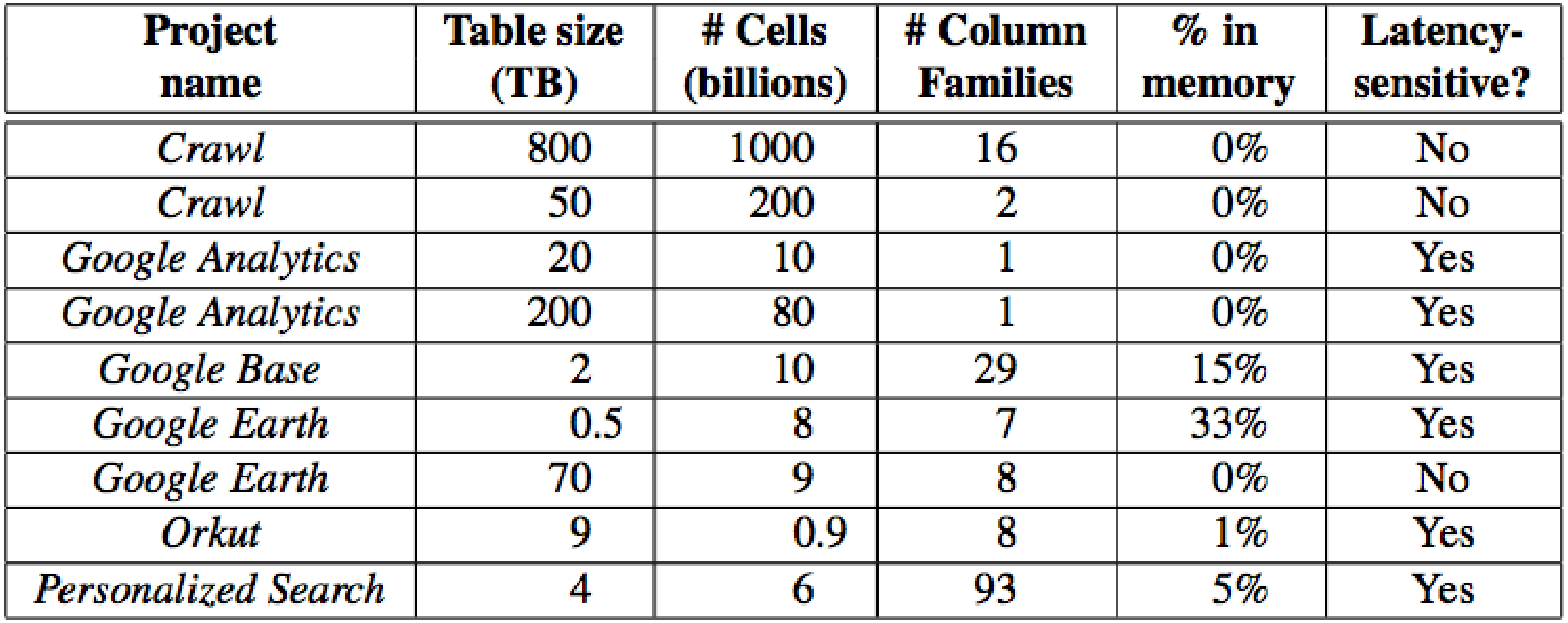
BigTable: Dat Model

(row, column, time) → string

- String: Array of uninterpreted bytes
  - Whatever data
  - No size limit
- Row keys: uninterpreted bytes
  - Up to 64KB
  - Data sorted by row key
  - Atomic changes to multiple columns with the same row key
- Column: uninterpreted bytes
  - Sorted into column families
    - Basic unit of access control / only a few hundred
    - Lone columns are trivial families
- Time
  - Used for versioning
  - 64bit Integers
  - Timestamps per column
  - Current time in milliseconds or application timestamp

BigTable: Applications

- Used in many Google projects (data from 2006)



Project name	Table size (TB)	# Cells (billions)	# Column Families	% in memory	Latency-sensitive?
Crawl	800	1000	16	0%	No
Crawl	50	200	2	0%	No
Google Analytics	20	10	1	0%	Yes
Google Analytics	200	80	1	0%	Yes
Google Base	2	10	29	15%	Yes
Google Earth	0.5	8	7	33%	Yes
Google Earth	70	9	8	0%	No
Orkut	9	0.9	8	1%	Yes
Personalized Search	4	6	93	5%	Yes

DBMS vs. Key/Value Store Database Structure

Relational Database

- Tables w/ columns and rows, rows contain column values, rows in table all have same schema
- Data model is well-defined in advance, schema is typed, has constraints & relationships to enforce integrity
- Data model based on „natural" representation of data, independent of app data structures
- Data model is normalized to remove duplication. Relationships associate between multiple tables
- No good support for large amounts of binary data

Key/Value Store

- Table is a bucket were key/value pairs are put into.
- Different rows can have different „schemas"
- Records are identified by keys. A dynamic set of attributes can be associated with a key
- Sometimes attributes are all of a string type only
- No relationships are explicitly defined. No normalization models
- Handles binary data well

DBMS vs. Key/Value Store Data Access

Relational Database

- Data is created, updated, and retrieved using SQL
- SQL queries can act on single tables, but also on multiple tables through table joins
- SQL provides functions for aggregation & complex filtering
- „Active Database“ with triggers, stored procedures, functions

Key/Value Store

- Data manipulated through proprietary APIs
- Sometimes SQL-like syntax without join or update functionality (e.g., GQL)
- Only basic filter predicates (=, !=, <, >). Often: processing not considered to be part of DB (→ MapReduce, Hadoop)
- App & data integrity logic all contained in app code

DBMS vs. Key/Value Store Application Interface

Relational Database

- De-facto standards for SQL database drivers (e.g., ODBC, JDBC)
- Data is stored in „natural“ representation. Often mapping between app data structures and relational structure used: Object-Relational Mapping (ORM)

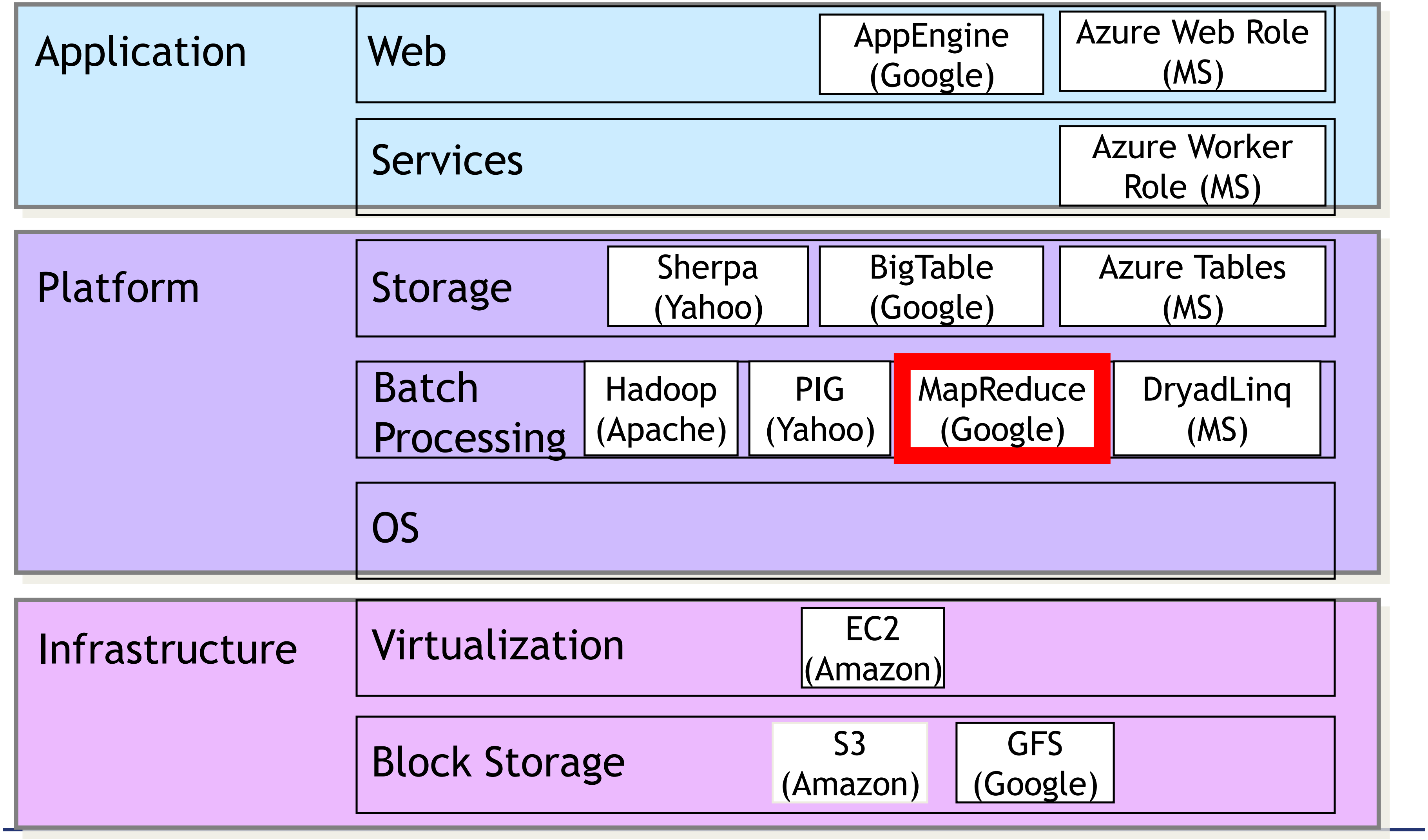
Key/Value Store

- Proprietary APIs, sometimes SOAP/REST
- Direct serialization of app objects into the value part of records

DBMS vs. Key/Value Store

- Applications for Key/Value Stores
  - Data is heavily document-oriented
    → more natural fit with the key/value model than the relational model
  - Large amounts of binary data
  - Application builds heavily on object-orientation → key/value store minimizes need for „plumbing“ code
  - Data store is cheap & integrates easily with cloud platform
  - On-demand, vast scalability is of vital importance
    - Thousands of servers
    - Terabytes of in-memory data
    - Petabyte of disk-based data
    - Millions of reads/writes per second, efficient scans

The „Cloud Stack“



Application	Web	AppEngine (Google) Azure Web Role (MS)
	Services	Azure Worker Role (MS)
Platform	Storage	Sherpa (Yahoo) BigTable (Google) Azure Tables (MS)
	Batch Processing	Hadoop (Apache) PIG (Yahoo) MapReduce (Google) DryadLinq (MS)
	OS	
Infrastructure	Virtualization	EC2 (Amazon)
	Block Storage	S3 (Amazon) GFS (Google)

MapReduce

Consider problem: count number of occurrences of each word in large document collection (in how many doc’s does it appear?)

- **MapReduce** Programming Model
  - Inspired from **map** and **reduce** operations commonly used in functional programming languages like Lisp

  1. map: (key1, val1) → **[** (key2, val2) **]**
  2. reduce: (key2, [val2]) → [val3]
- **Map**
  - is a pure function (always evals to same result; no side effects)
  - processes input key/value pair
  - produces *set of* key/value pairs (intermediate results)
  - e.g.: (document_name, document_content) -> [(word$_1$, 1),...,(word$_n$, 1)]
- **Reduce**
  - Combines all intermediate values for a particular key
  - Produces a set of merged output values (usually just one)
  - e.g.: (word, [1,...,1]) -> [n]

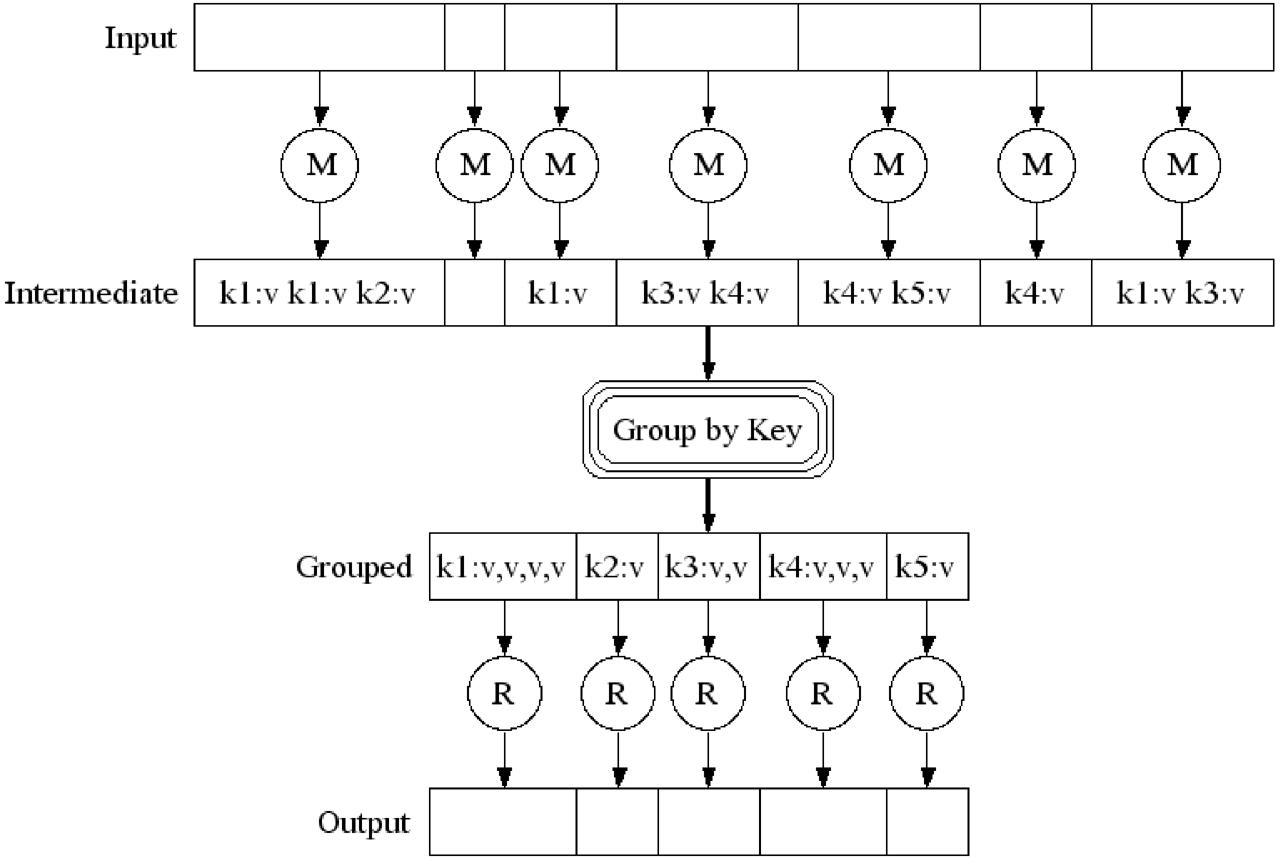
MapReduce

- Example: count word occurrences

```
map (String input_key, String input_value):
    // input_key: document name
    // input_value: document contents
    for each word w in input_value:
      EmitIntermediate(w, "1");


reduce(String output_key, Iterator intermediate_values):
    // output_key: a word
    // output_values: a list of counts
    int result = 0;
    for each v in intermediate_values:
      result += ParseInt(v);
    Emit(AsString(result));
```

MapReduce: Execution



MapReduce: Distributed Execution

Input Reader

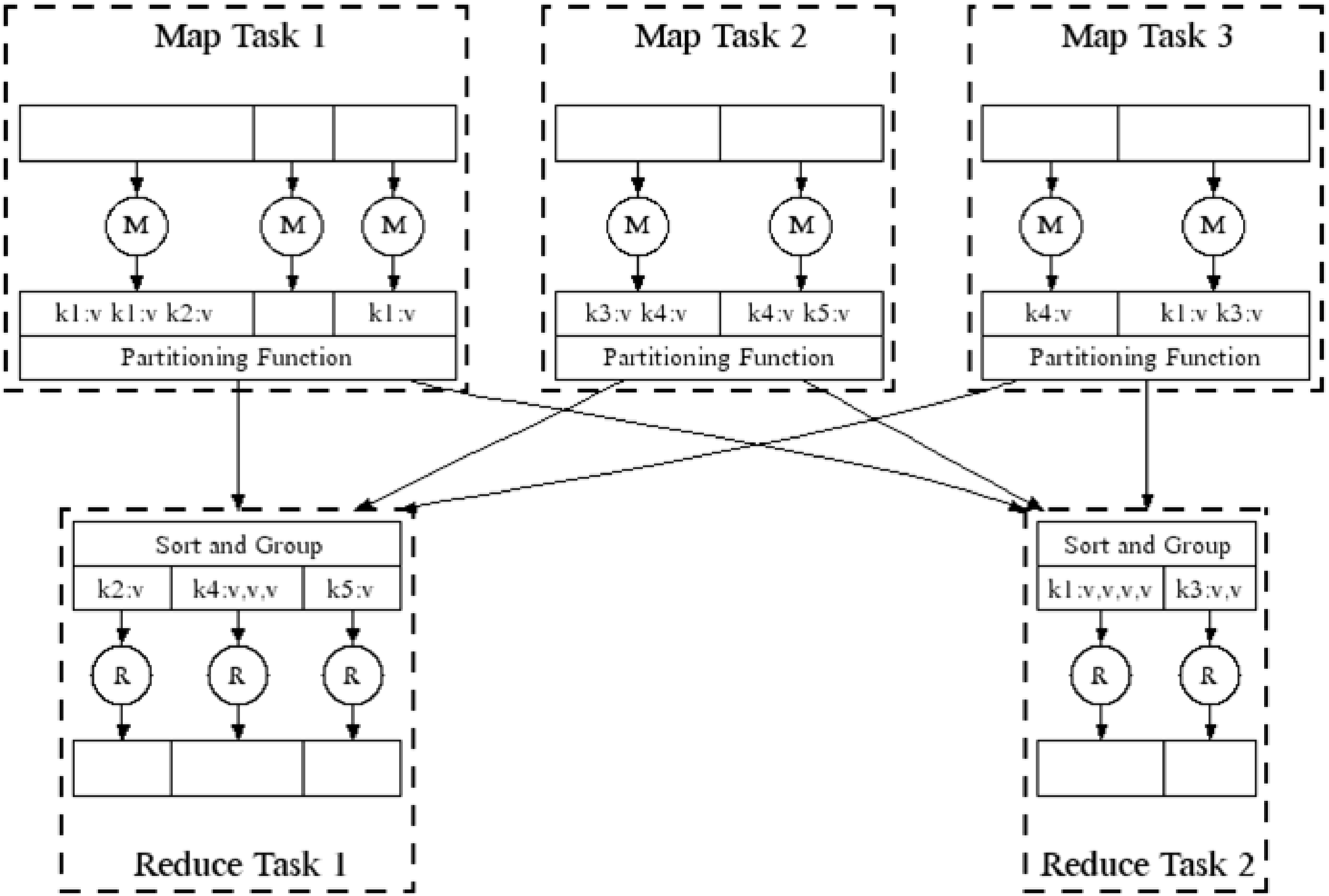
Map

Partition

Compare

Reduce

Output Writer



MapReduce: Data Flow

Static part of MapReduce: “large distributed sort”

Variable part, defined by applications:

1. Input Reader

- Reads data from (distributed) file system, generates key/value pairs
- E.g., read directory full of text files & return each line as record

2. Map Function

3. Partition Function

- Assign output of a map to a reducer (on all map nodes, for all map functions): reducerIndex = partitionFunction(key, numberOfReducers)
  - Note: all identical keys must be processed by same reducer
  - E.g., reducerIndex = hash(key) % numberOfReducers

4. Compare Function

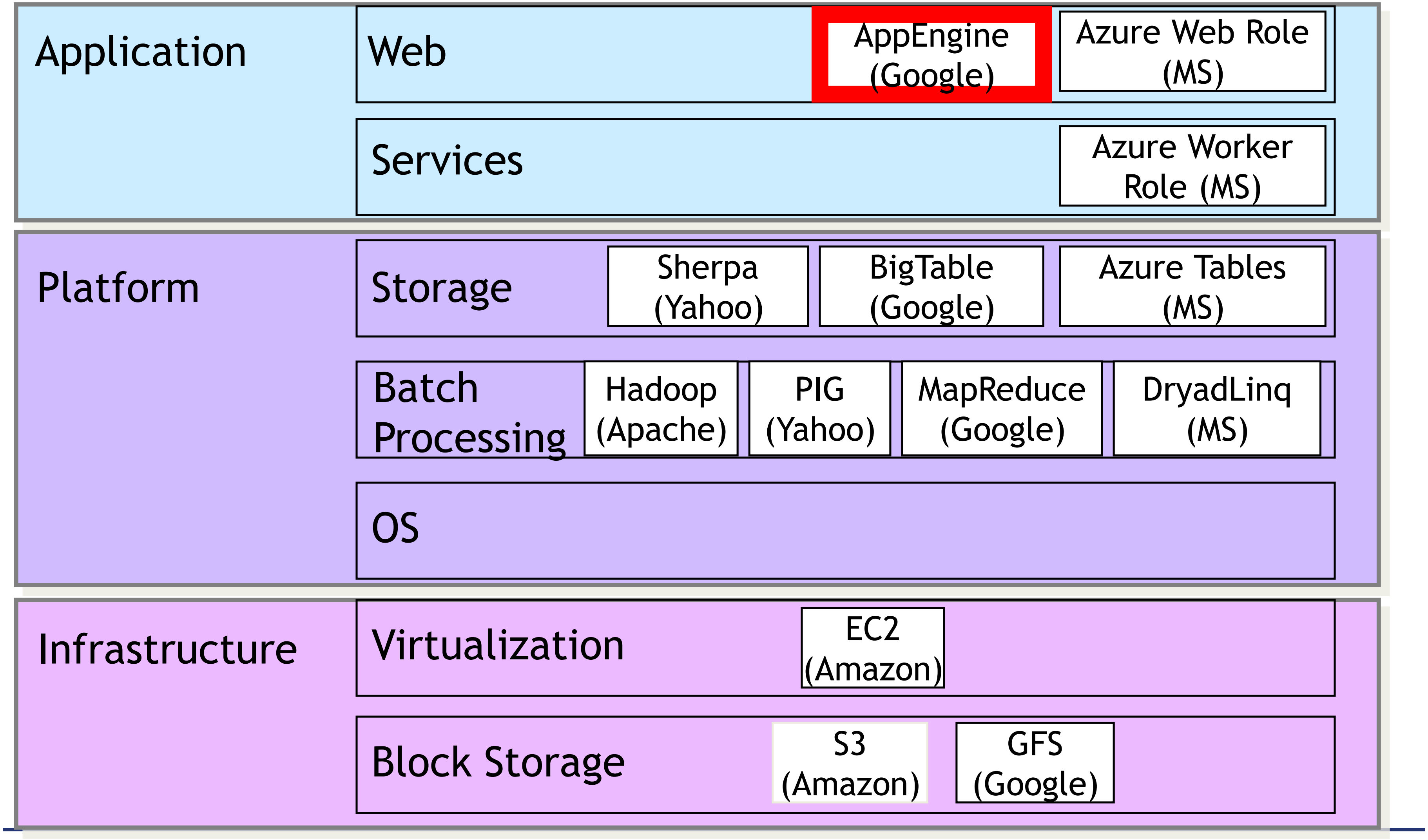
- Defines which keys to be considered identical, groups input for reduce step

5. Reduce Function

6. Output Writer

- Writes output to (distributed) filesystem

The „Cloud Stack“



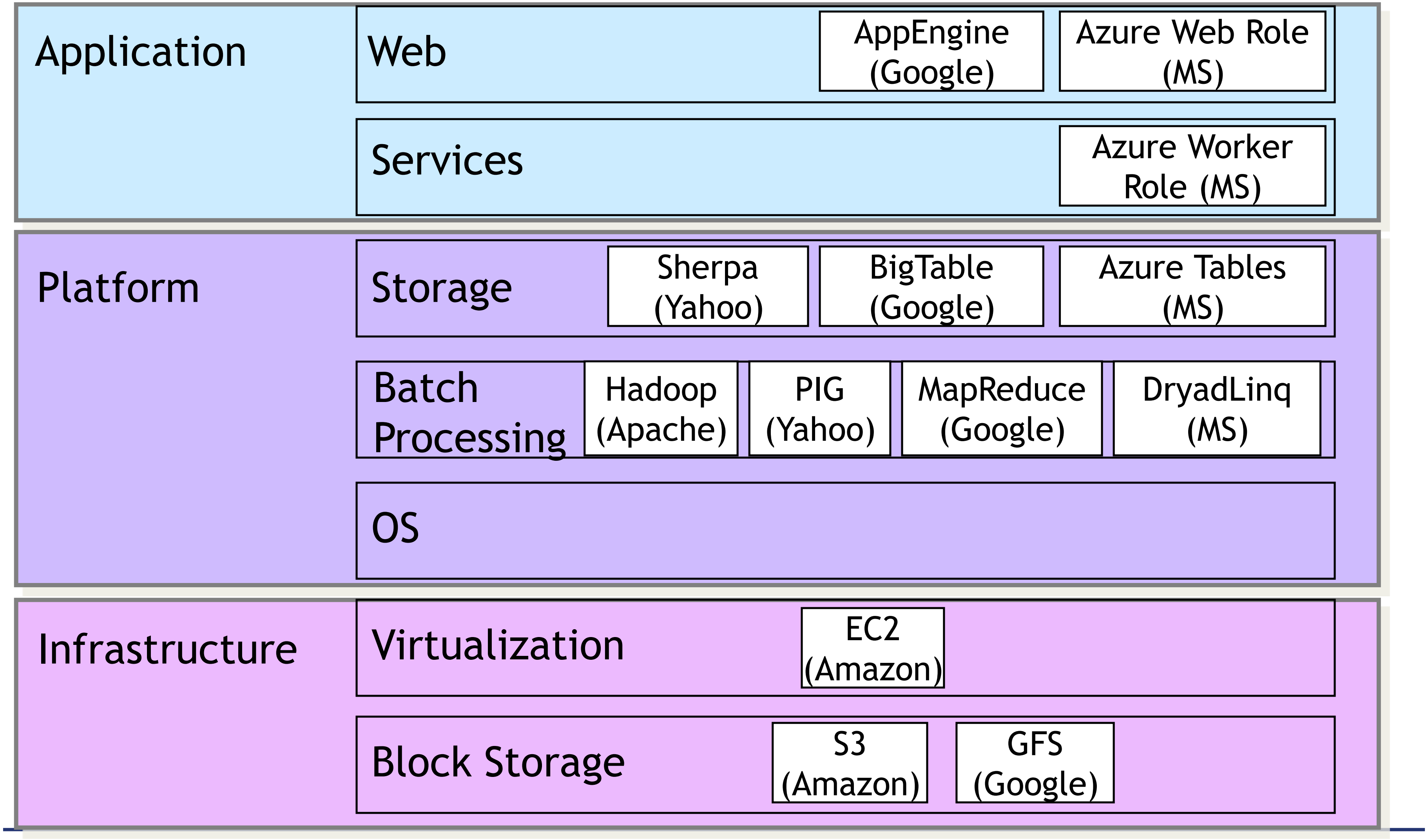
Application	Web	AppEngine (Google) Azure Web Role (MS)
	Services	Azure Worker Role (MS)
Platform	Storage	Sherpa (Yahoo) BigTable (Google) Azure Tables (MS)
	Batch Processing	Hadoop (Apache) PIG (Yahoo) MapReduce (Google) DryadLinq (MS)
	OS	
Infrastructure	Virtualization	EC2 (Amazon)
	Block Storage	S3 (Amazon) GFS (Google)

Google App Engine

Main purpose: run your web applications on Google's infrastructure

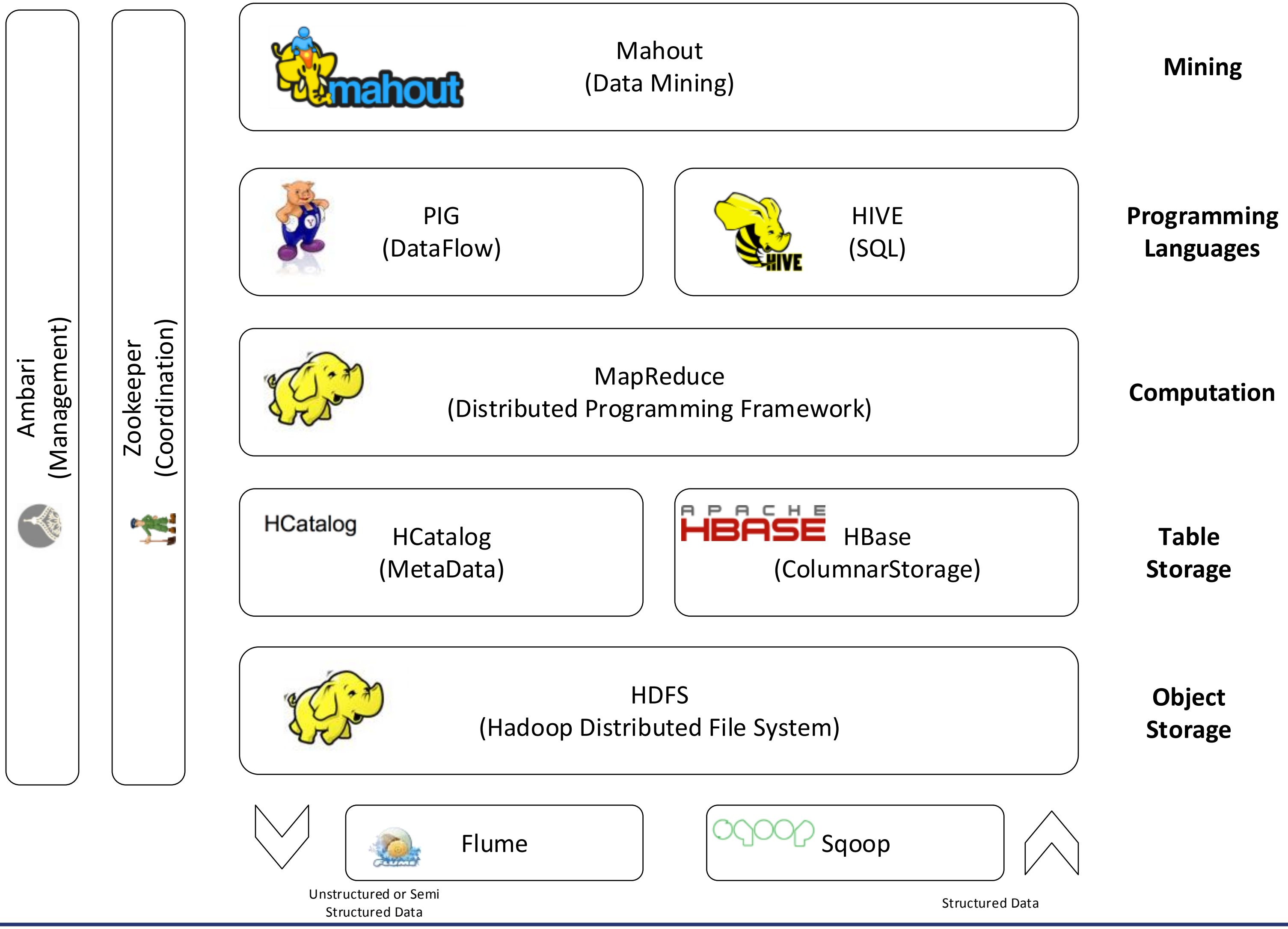
- **Python, Java, PHP, Go**
- **Sandbox** with several limitations
  - App code can only run as result of HTTP(S) request, may not spawn sub-processes
  - Cannot write to file system
  - App can only fetch URLs or use email services for communication
- **Datastore**
  - Distributed storage service with query engine and transactions
  - Stores data objects of mixed types, each object consists of typed key/value pairs
  - Query with GQL, a limited SQL variant
- **Google Cloud SQL**
  - MySQL in the cloud
- **Google accounts**
  - App can access information from Google accounts

The „Cloud Stack“



Application	Web	AppEngine (Google)	Azure Web Role (MS)		
	Services	Azure Worker Role (MS)			
Platform	Storage	Sherpa (Yahoo)	BigTable (Google)	Azure Tables (MS)	
	Batch Processing	Hadoop (Apache)	PIG (Yahoo)	MapReduce (Google)	DryadLinq (MS)
	OS				
Infrastructure	Virtualization	EC2 (Amazon)			
	Block Storage	S3 (Amazon)	GFS (Google)		

Hadoop Stack



Mahout
(Data Mining)

Mining

PIG
(DataFlow)

HIVE
(SQL)

Programming Languages

MapReduce
(Distributed Programming Framework)

Computation

HCatalog
(MetaData)

HBase
(ColumnarStorage)

Table Storage

HDFS
(Hadoop Distributed File System)

Object Storage

Ambari
(Management)

Zookeeper
(Coordination)

Flume

Unstructured or Semi Structured Data

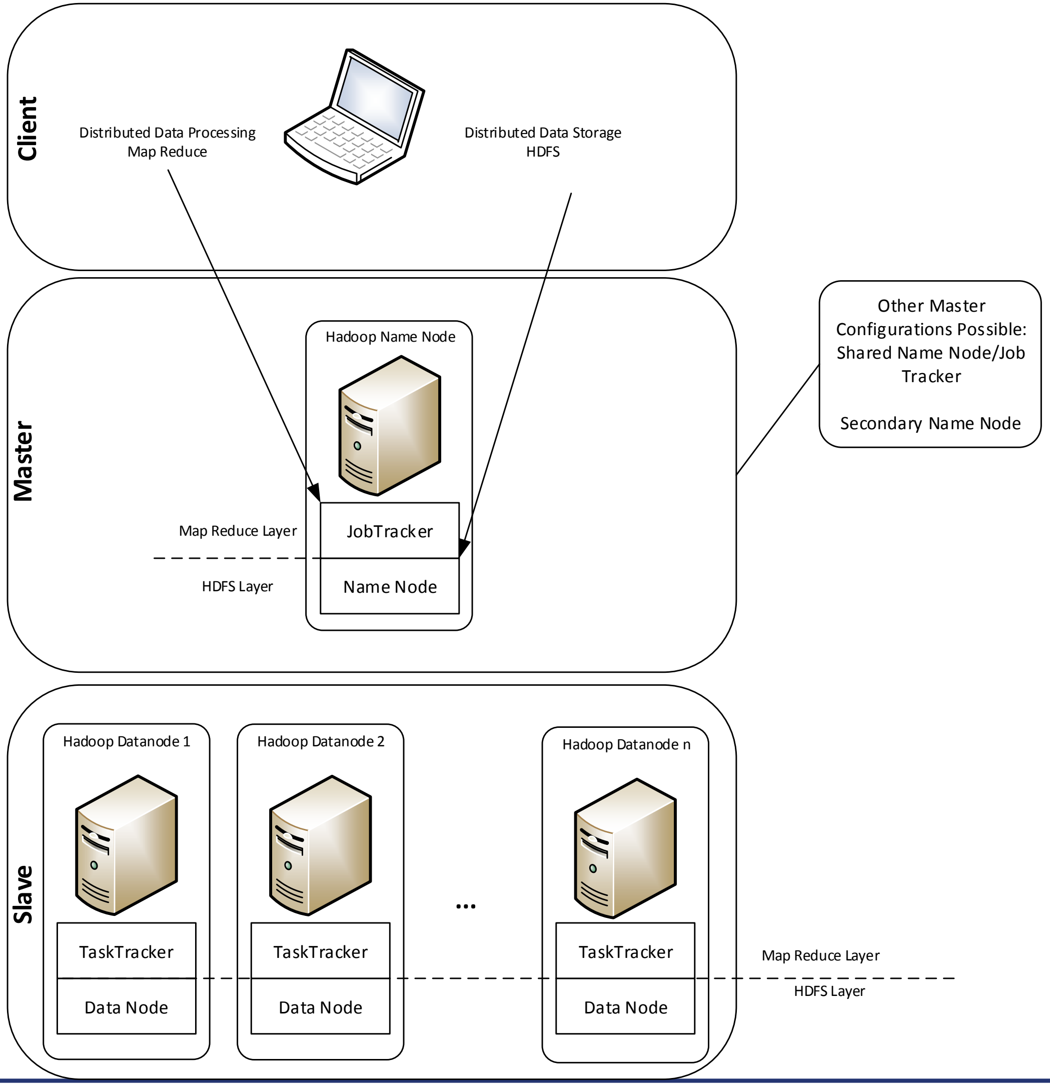
Sqoop

Structured Data

Hadoop Stack

- Mahout: data mining capabilities
- Pig / Hive: High level code transformed into MapReduce Tasks
- Hadoop: MapReduce and Distributed File System
- Hbase: Key-Value Store, a persistent hash-map
- HCatalog: centralized meta-data managemet for Hadoop
- Scoop: exchange between Hadoop and relational database
- Flume: efficiently collecting, aggregating, and moving large amounts of log data
- Ambari/Zookeeper: Manage cluster: configuration, sychronization, etc.

Hadoop Overview



Hadoop HDFS Main Components

- Name Node (Master Node)
  - Centerpiece of an HDFS file system
  - Maintains and manages the blocks which are present on the Data Nodes
  - Keeps the directory tree of all files in the file system, and tracks where across the cluster the file data is kept
  - It is responsible for maintaining namespace image and edit log files
  - Any Changes to the file system namespace or its properties is recorded by Name Node
  - Mapping of file blocks to Data Nodes (Physical location of Data)
  - Aware of the data nodes for a particular file
  - Important files for Name Node
    - Image: File consists of meta data information
    - Check Point: Persistent record of image stored in the native file system
    - Journal: Modification log of image stored in local file system

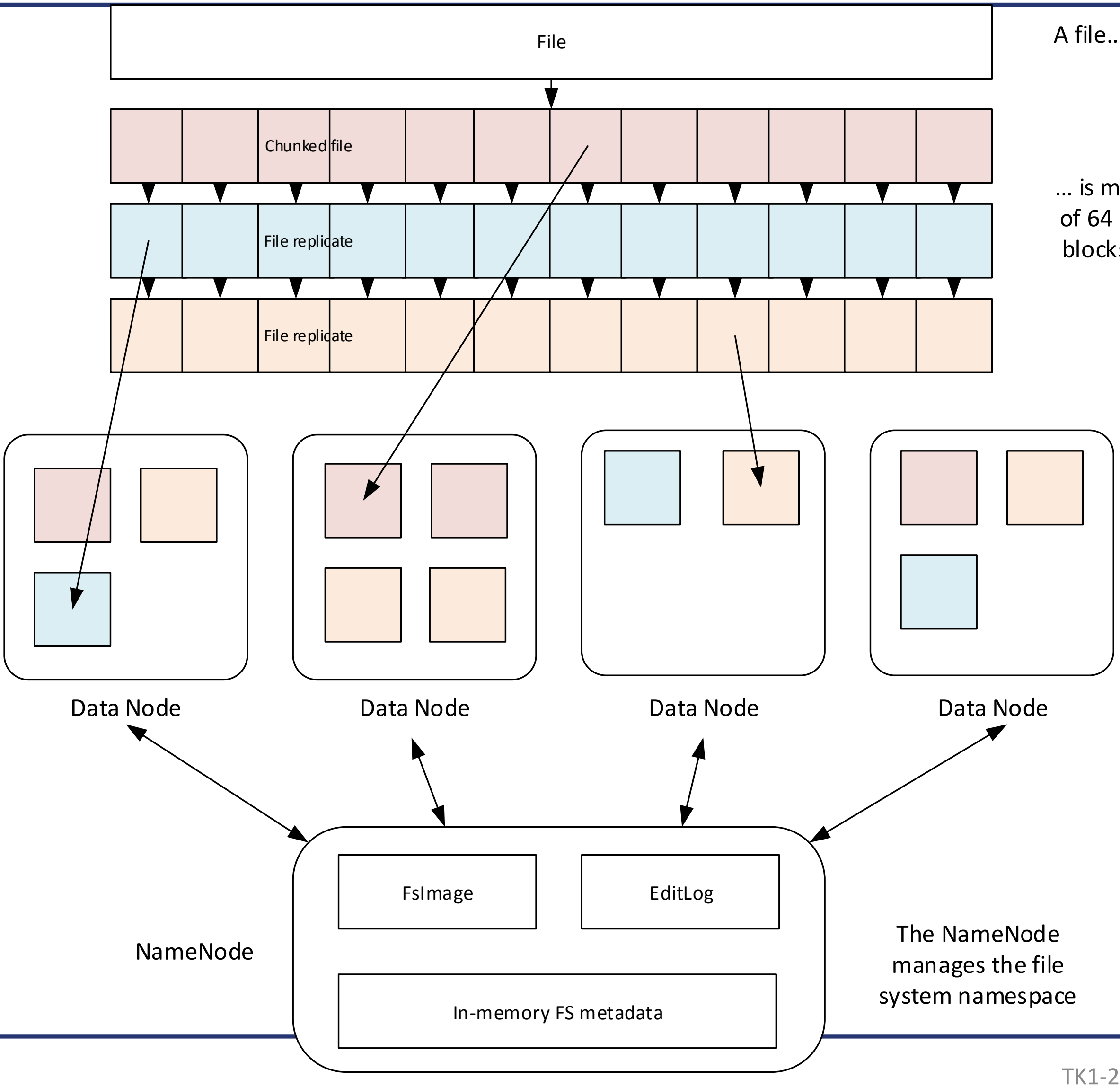
Hadoop HDFS Main Components

- Data Node (Slave Node)
  - Slaves which are deployed on each machine and provide the actual storage
  - Responsible for serving read and write requests for the clients
  - Store and retrieve blocks when they are told to
  - Report back to Name Node periodically with the list of blocks they have
  - Data Nodes are the work horses of the Hadoop file system
  - Block replica is represented by two files
    - File that stores the data itself
    - File that stores the block meta data which includes checksum for the block and block’s generation time stamp

Files in Hadoop

- Hadoop stores large files
- Files are broken into big chunks
  - Default 64MB
- These blocks are distributed among different machines
- Blocks are replicated for redundancy
  - Default: 3

Hadoop File Handlung

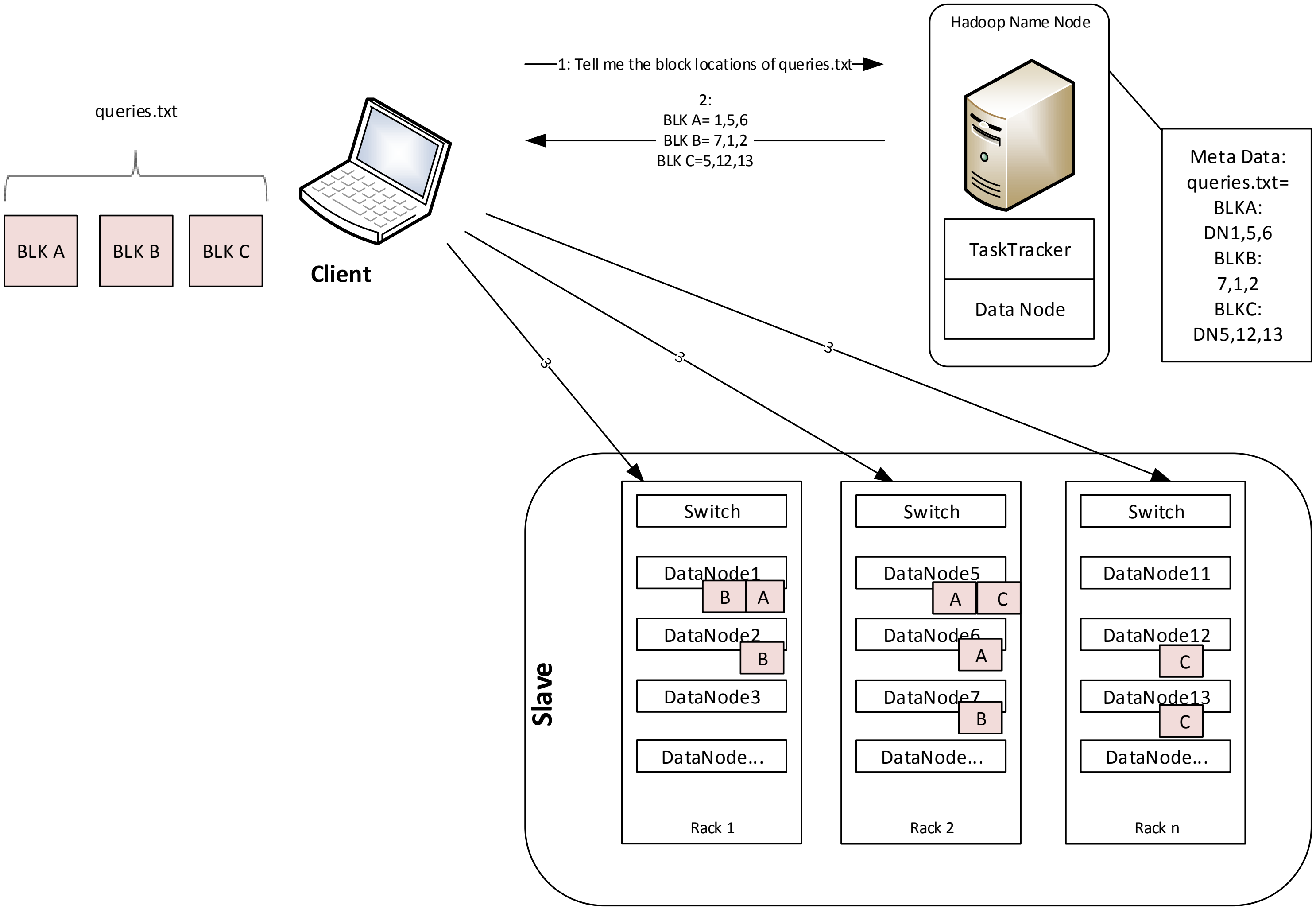


A file….

… is made of 64 MB blocks …

The NameNode manages the file system namespace

Read Process



queries.txt

BLK A

BLK B

BLK C

Client

1: Tell me the block locations of queries.txt

2:
BLK A= 1,5,6
BLK B= 7,1,2
BLK C=5,12,13

Hadoop Name Node

TaskTracker

Data Node

Meta Data:
queries.txt=
BLKA:
DN1,5,6
BLKB:
7,1,2
BLKC:
DN5,12,13

3

3

3

Slave

Switch

DataNode1

B

A

DataNode2

B

DataNode3

DataNode...

Rack 1

Switch

DataNode5

A

C

DataNode6

A

DataNode7

B

DataNode...

Rack 2

Switch

DataNode11

DataNode12

C

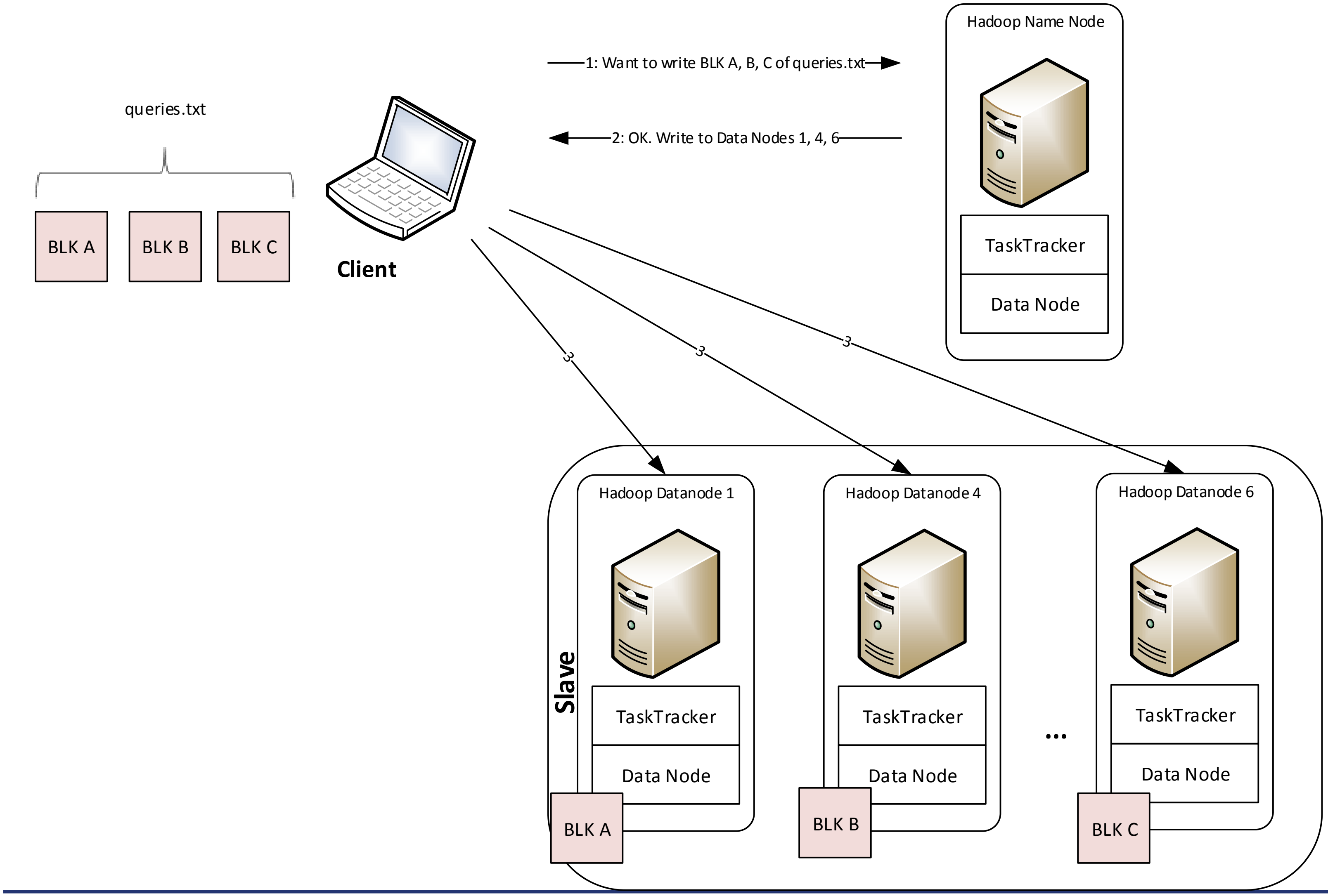
DataNode13

C

DataNode...

Rack n

Write Process



Node Replication

- Why replicate data?
  - Fault tolerance
  - Read latency improvement
  - Configure: Replication factor -> how many replicates?
- Replica placement policy
  - 1st replica on-rack: replicate on local rack
  - 2nd replica off-rack: random node in random, different rack
  - 3rd replica: different node, same rack as 2nd replica
  - Further replica: random, avoid too many on same rack

Hadoop and Map Reduce

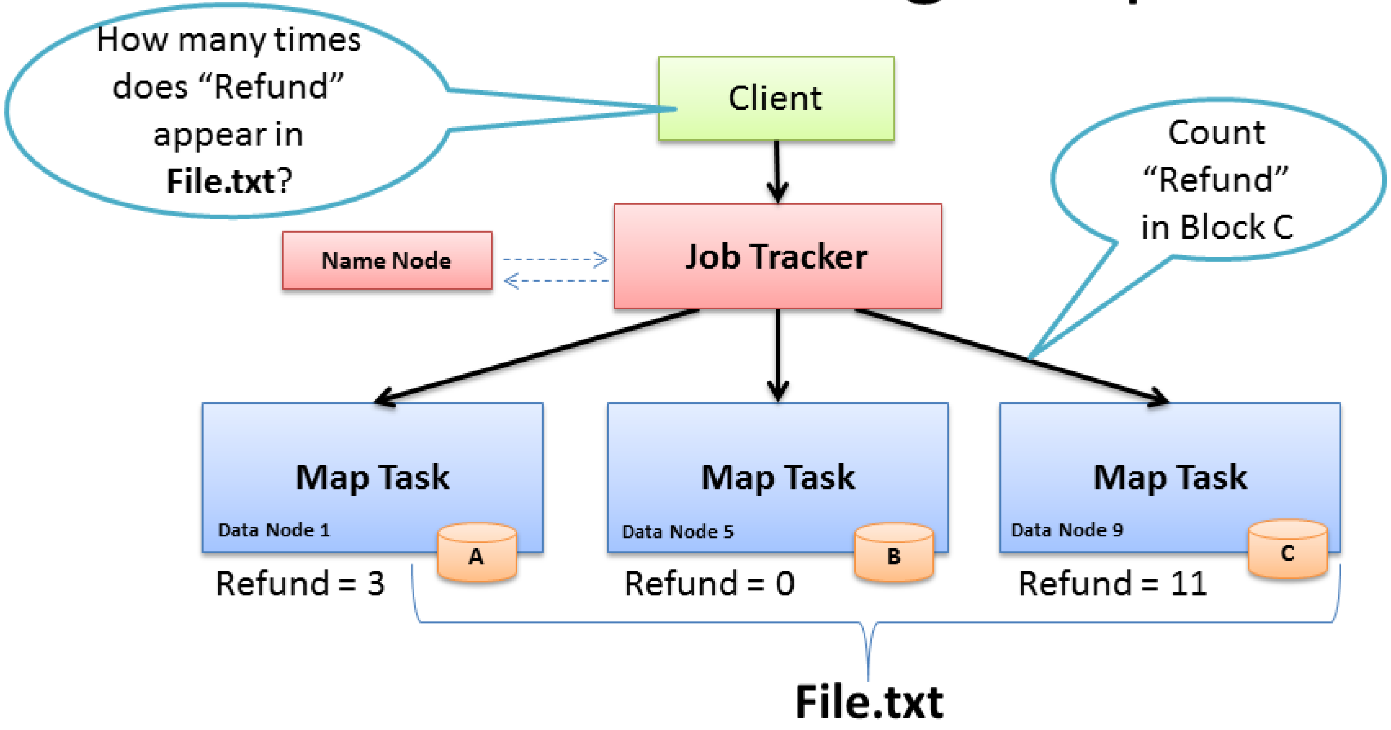
- Client applications submit map reduce tasks to job tracker
- Job Tracker
  - Runs on master node
  - Schedules tasks
  - Allocates tasks to slave task trackers
  - Manages data nodes
  - Keeps track of resources, already running tasks
  - Single point of failure
- Task Tracker
  - Executes and manages the tasks received from job tracker
  - Handles data motion between map and reduce phase
  - Communicates status of task to job tracker (send heartbeats)

Map Reduce Process 1/4

- Client submits job to job tracker
- Job tracker gets location of data from name node
- Job tracker identifies task tracker nodes w. available slots close to data
- Job tracker submits tasks to task tracker nodes
- Job tracker monitors task trackers based on their heartbeat. No heartbeat=assume crash
- If task tracker fails with a task the job tracker is notified. Job tracker decides what to do (resubmit, mark data as to-be-avoided)
- Job tracker offers status information to client systems

Map Reduce Process 2/4

Data Processing: Map



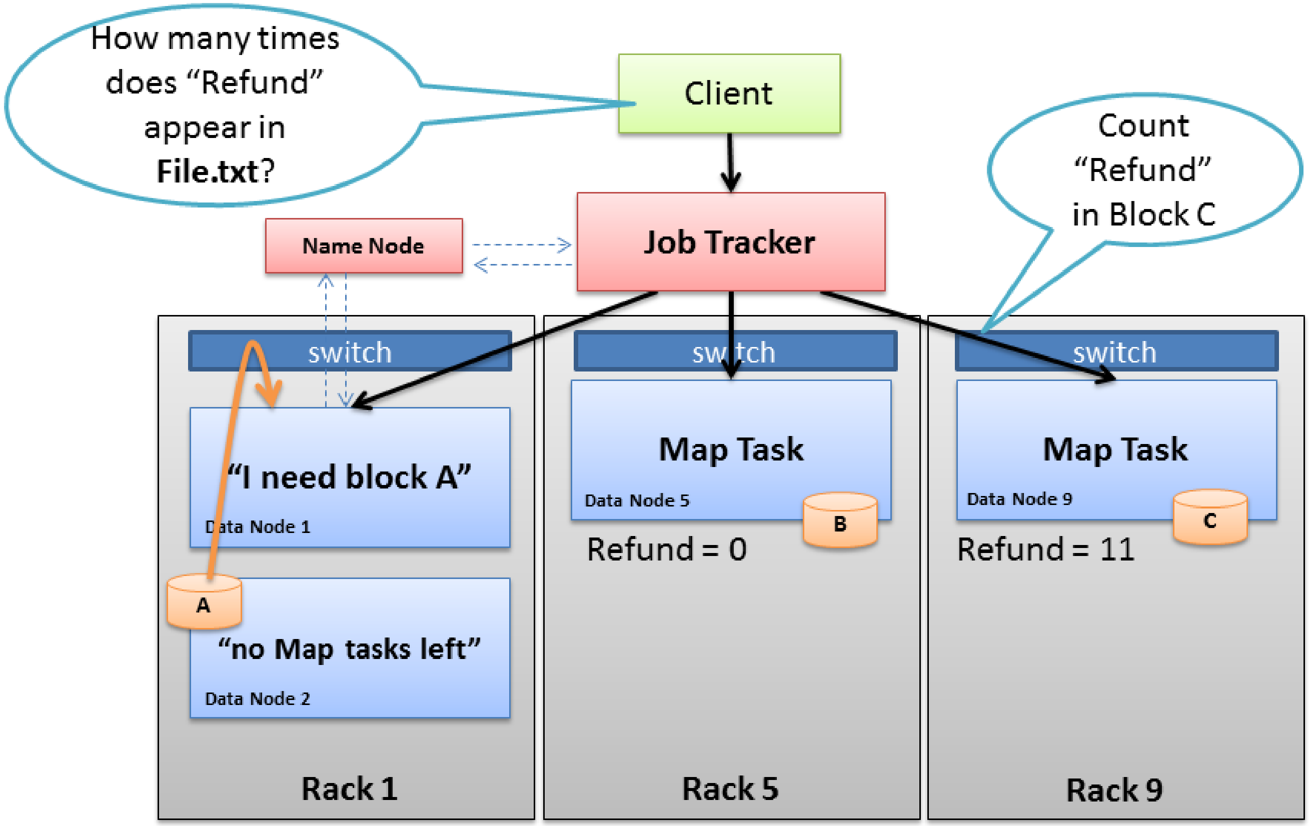
- **Map:** "Run this computation on your local data"
- Job Tracker delivers Java code to Nodes with local data

BRAD HEDLUND .com

http://bradhedlund.com/2011/09/10/understanding-hadoop-clusters-and-the-network/

Map Reduce Process 2/4

What if data isn't local?

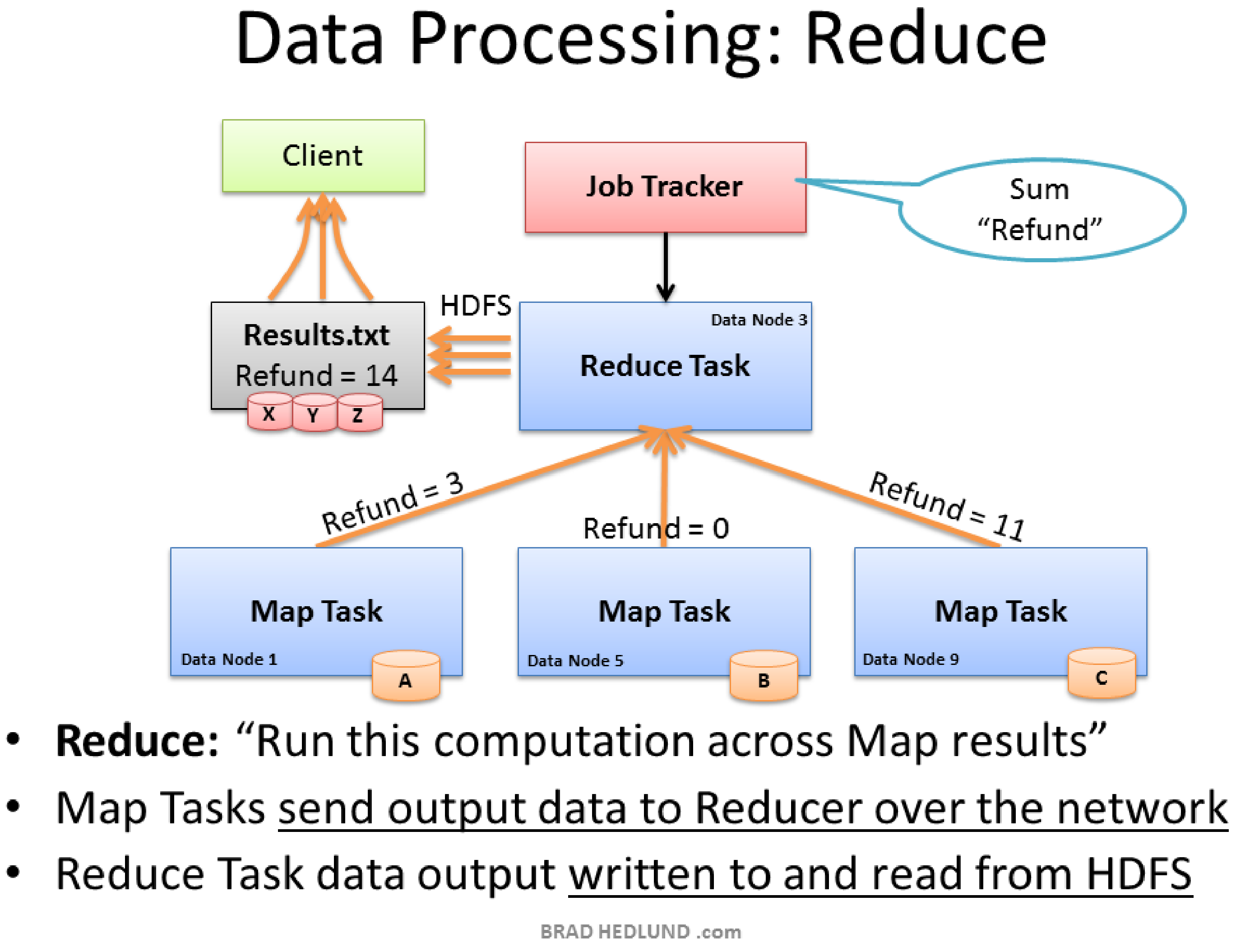


- Job Tracker tries to select Node in same rack as data
- Name Node rack awareness

BRAD HEDLUND .com

http://bradhedlund.com/2011/09/10/understanding-hadoop-clusters-and-the-network/

Map Reduce Process 2/4



Data Processing: Reduce

- **Reduce:** "Run this computation across Map results"
- Map Tasks send output data to Reducer over the network
- Reduce Task data output written to and read from HDFS

BRAD HEDLUND .com

http://bradhedlund.com/2011/09/10/understanding-hadoop-clusters-and-the-network/

Hadoop Modes

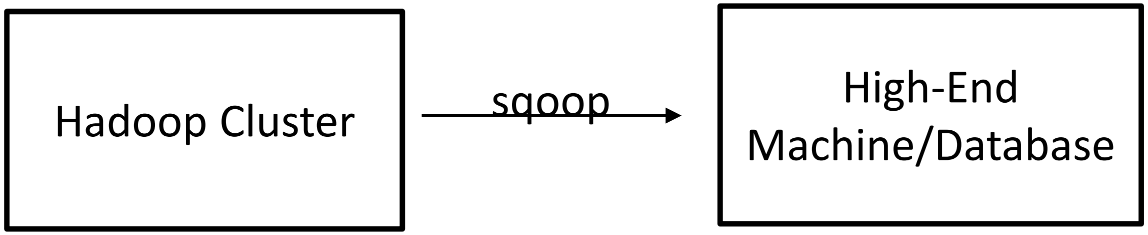
- **Single node setup (standalone setup) :** Hadoop as a single Java process.
  No daemons running and everything runs in a single JVM instance.
  HDFS is not used.
- **Pseudo-distributed mode :** Simulating a cluster on a small scale.
  Hadoop daemons run on a local machine. Different Hadoop daemons run in different JVM instances, but on a single machine.
  HDFS is used instead of local FS.
- **Fully Distributed Mode:** Hadoop on a cluster of machines

One usage of Clouds: Big Data

- There is big data
  - Airline Jet: 10 TB of data for 30 min flight
  - NYSE: 1 TB of trade data for one trading day
  - Facebook: 1 billion active users, 3.2 billion likes per day
  - Twitter: 6000 Tweets per minute

Perspectives

- Hadoop provides an infrastructure built from relatively cheap commodity hardware
- What about the really fast machines and the really fast databases? E.g., your in-memory database (e.g., Hana) and the high-end server?
- Use Hadoop for the „much data“ things… raw logs, backups, analytic tasks for huge data sets
- Use the fast machines for „selected“ tasks -> use sqoop to transfer selected data (might be results from a map reduce job)



Existing Open Source Cloud Frameworks

- Eucalyptus
- Open Nebula
- Nimbus
- Cloud Stack
- Open Stack
- App Scale
- Typhoon AE
- Apache Hadoop

Summary

- Cloud Computing objectives
- Types of Cloud Service: SaaS, PaaS, IaaS
- Cloud vs. Grid Computing, Scalability
- Architecture of a Cloud Platform, “The Cloud Stack”
  - Block Storage: Amazon S3, Google File System
  - Virtualization: Amazon EC2
  - Structured Distributed Storage: BigTable
  - Batch Processing: MapReduce, Hadoop
  - Google App Engine

Literature

- Sanjay Ghemawat, Howard Gobioff, Shun-Tak Leung. "The Google File System". in *Symposium on Operating Systems Principles (SOSP), 2003.*
- Jeffrey Dean, Sanjay Ghemawat. "MapReduce: Simplified Data Processing on Large Clusters". in *Operating Systems Design and Implementation (OSDI), 2004.*
- Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber. "Bigtable: A Distributed Storage System for Structured Data". in *Operating Systems Design and Implementation (OSDI), 2006.*

- Tony Bain. Is the Relational Database Doomed?. Retrieved Nov 25, 2015, from http://www.readwriteweb.com/enterprise/2009/02/is-the-relational-database-doomed.php, 2009.
- Donald Kossmann, Tim Kraska, and Simon Loesing: An Evaluation of Alternative Architectures for Transaction Processing in the Cloud, in: ACM SIGMOD’10, 2010.
- David Chappell: Introducing the Windows Azure Platform, retrieved Dec 07, 2010, from http://www.microsoft.com/windowsazure/
- Simon Munro: The Trouble With Sharding, retrieved Nov 25, 2015, from http://simonmunro.com/2009/09/10/the-trouble-with-sharding/
- Keith Brown, Sesha Mani: Microsoft Windows Identity Foundation (WIF) Whitepaper for Developers, 2009.