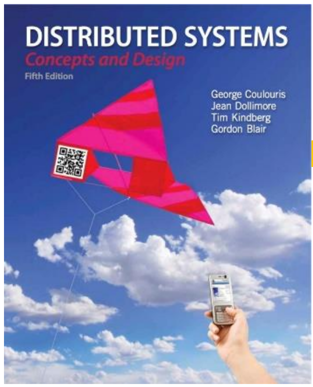
Case Study of Distributed Systems: Cloud Computing Systems



Concepts are drawn from Chapter 21

Slides are revised from Prof. Michael R. Lyu at CUHK and Prof. Richard O. Sinnott at UniMelb

Dr. Minxian Xu

Associate Professor
Research Center for Cloud Computing
Shenzhen Institute of Advanced Technology, CAS
http://www.minxianxu.info/dcp

松下问童子，言师采药去。只在此山中，云深不知处。
——（唐）贾岛

Review

- Q1: What are the relationships of Distributed Computing, Mobile Computing, Ubiquitous Computing?

Review

Review

- Q2: What is a volatile system? List the main types of changes that occur in a ubiquitous system.

Review

Outline

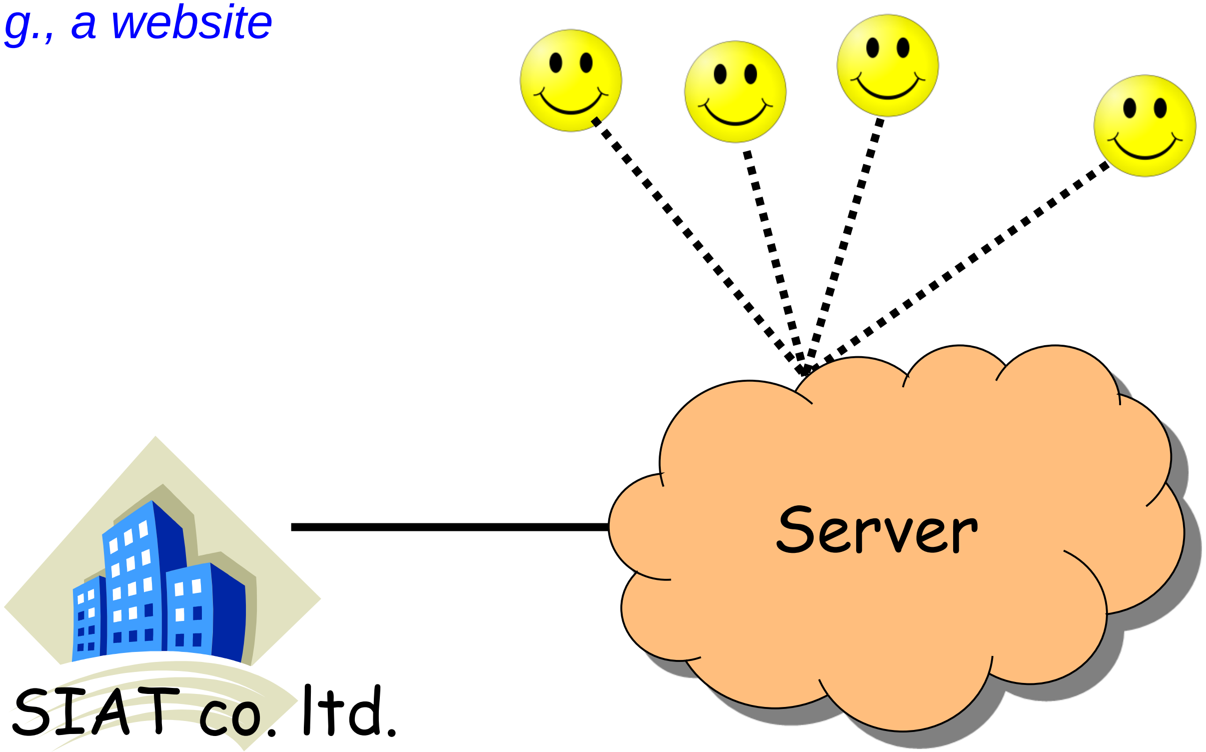
1. Cloud Computing Introduction
2. Virtualization and Containerization Concept
3. The Hadoop Distributed File System (HDFS)
4. MapReduce, Spark, OpenStack Computing Paradigm
5. Summary

1 What Is Cloud Computing?

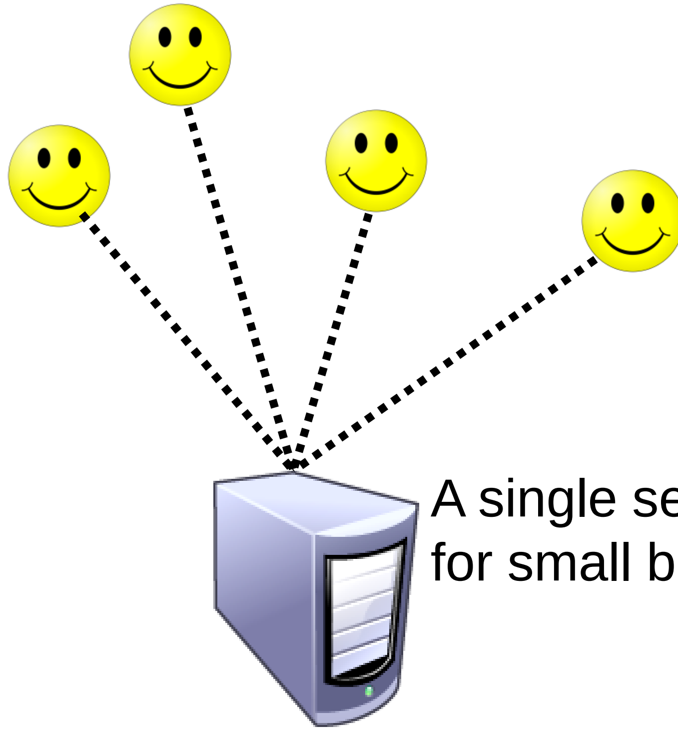
- Cloud computing is a hot IT topic in recent years
  - More and more companies have claimed their products are cloud computing based.
  - Gartner, Inc. forecasts worldwide public cloud revenue to grow 6.3% in 2020, reaching $257.9 billion. Forecast for 2021 is $306.9 billion and 2022 is $364.1 billion.
- So, what is cloud computing
  - A new name of an old technology created by marketing people?
  - A new technology focusing on higher-speed computing?
  - A new programming paradigm?

1 What Is Cloud Computing?

- Let us see what cloud computing is by its economic drive
  - See an example on hosting an Internet service
    - *e.g., a website*



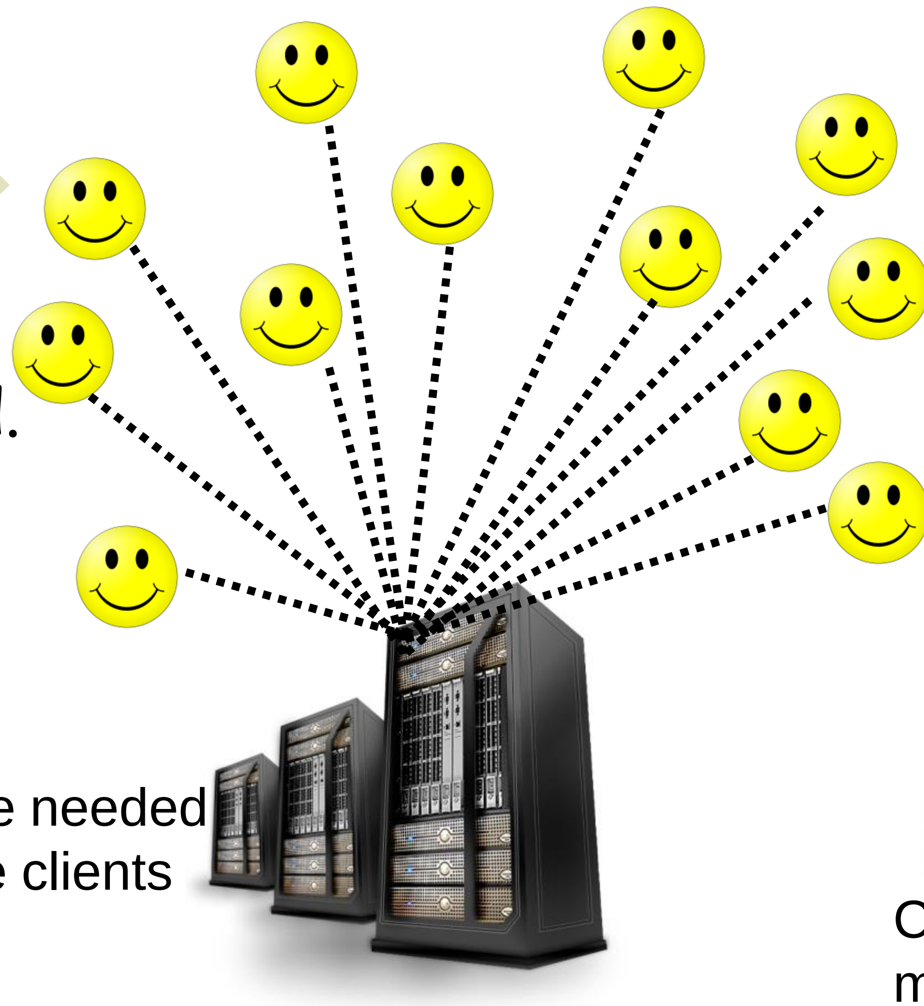
1.1 Hosting Computing Systems



A single server is enough
for small business

A small amount of budget

1.1 Hosting Computing Systems



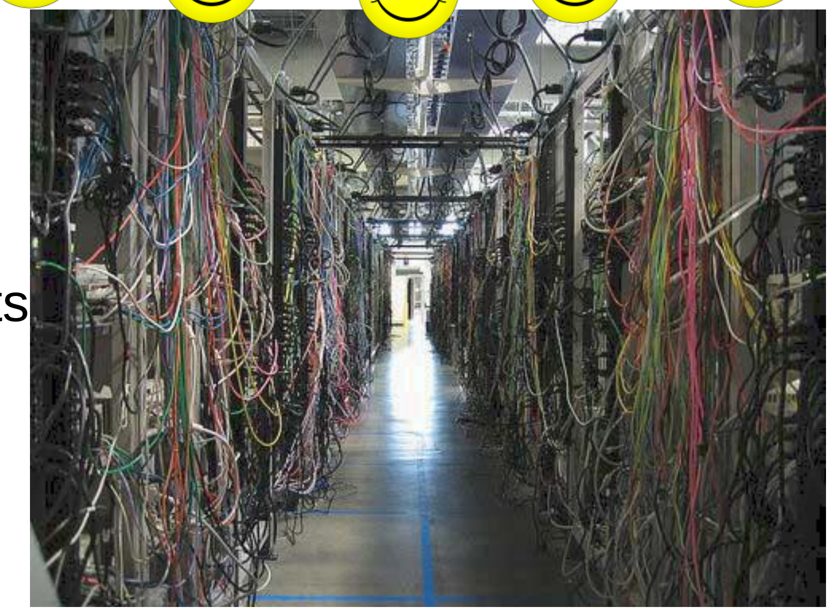
More servers are needed
if you have more clients

Cost more manpower and
money to provide the service

1.1 Hosting Computing Systems

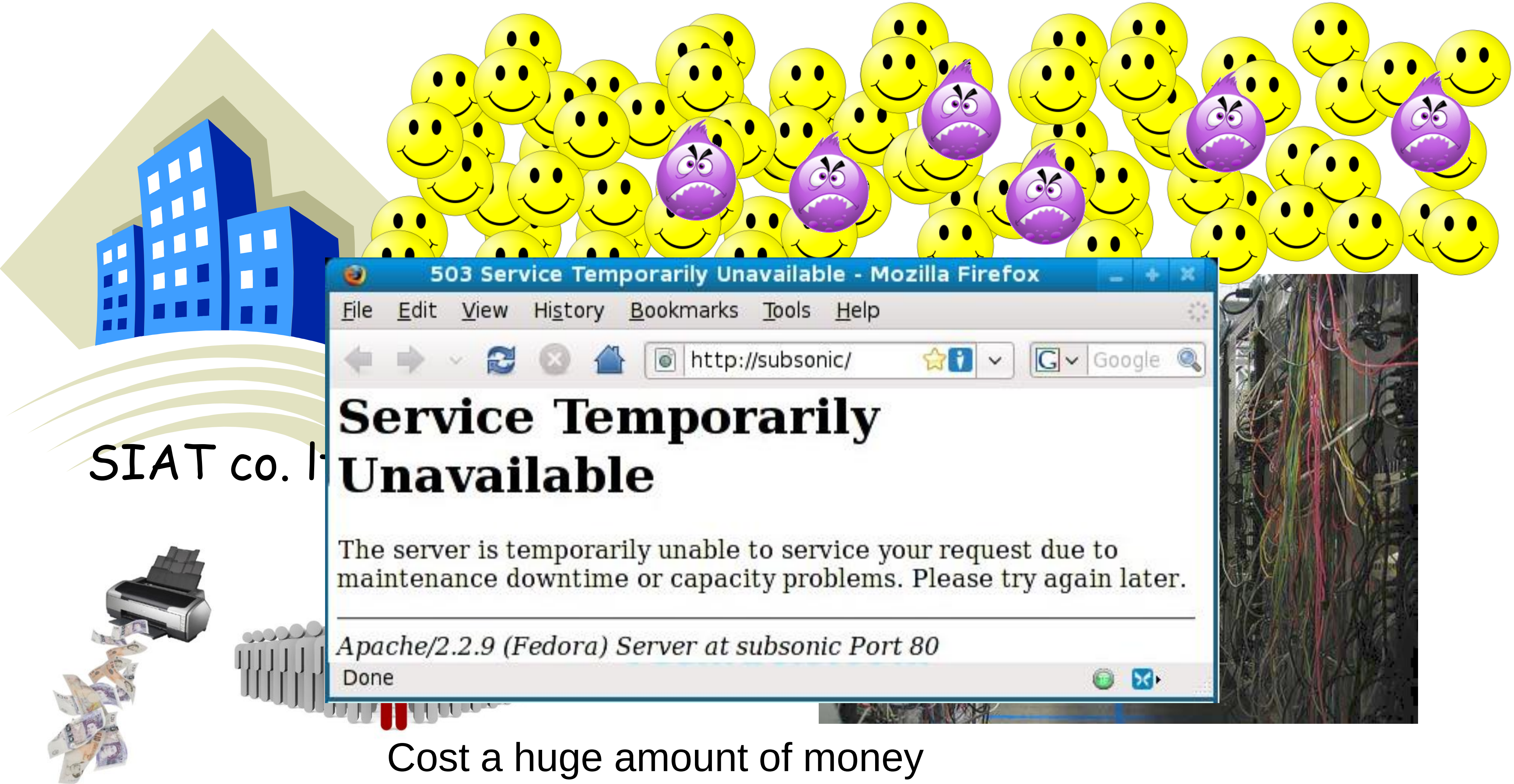
SIAT co. ltd.

A huge data center is needed if you have a huge number of clients



Cost a huge amount of money to provide the service

1.1 Hosting Computing Systems



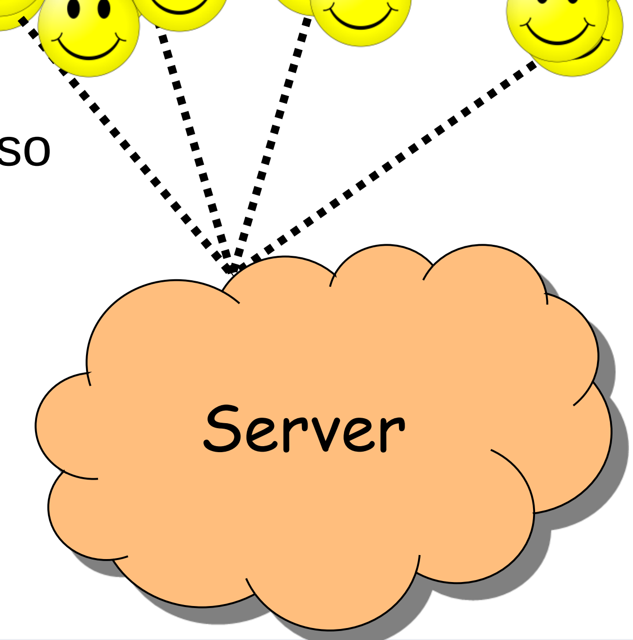
Cost a huge amount of money
to provide the service

1.1 Hosting Computing Systems

- The number of concurrent users is dynamic
  - You need to prepare for the peak number
    - *Otherwise, bad user experience!*
  - So you have to pay for the servers no matter whether you are using them or not
    - *The money you spend to buy the hardware*
    - *The salary for the maintainers*
    - *The electricity (including that for air conditioning)*
    - *Other maintenance costs, e.g., repairing, upgrading, renting a room for the servers*

1.2 Cloud Computing Motivations

- How to make it more economic when hosting computing systems



I wish this cloud can really host my system so that I can provide a scalable, on-demand service

1.2 Cloud Computing Introduction

- Hosting computing systems in a scalable, on-demand way
  - To be economic!
- Cloud computing
  - An emerging computing concept that approaches such a goal
  - Based on a lot of distributed system techniques
    - *Service-oriented system architecture*
    - *Virtualization technology*
    - *Distributed computing paradigm*

1.3 Pay As You Go

- The core of cloud computing is computing/ storage outsourcing
  - To another company
  - To a dedicating unit in the same organization
  - To be like using utilities, e.g. electricity, water and gas

vs

Pay as you go!!

Pay for exactly what you’ve used
Cost Down!!!

1.3 Outsourcing

Electronics industry

Brand vendor

Manufacturer

Manufacturing outsourcing

Different manufacturing requirements in low season and high season

# of product lines
# of workers

Cloud computing

Service vendor

Services for end users

Cloud computing provider

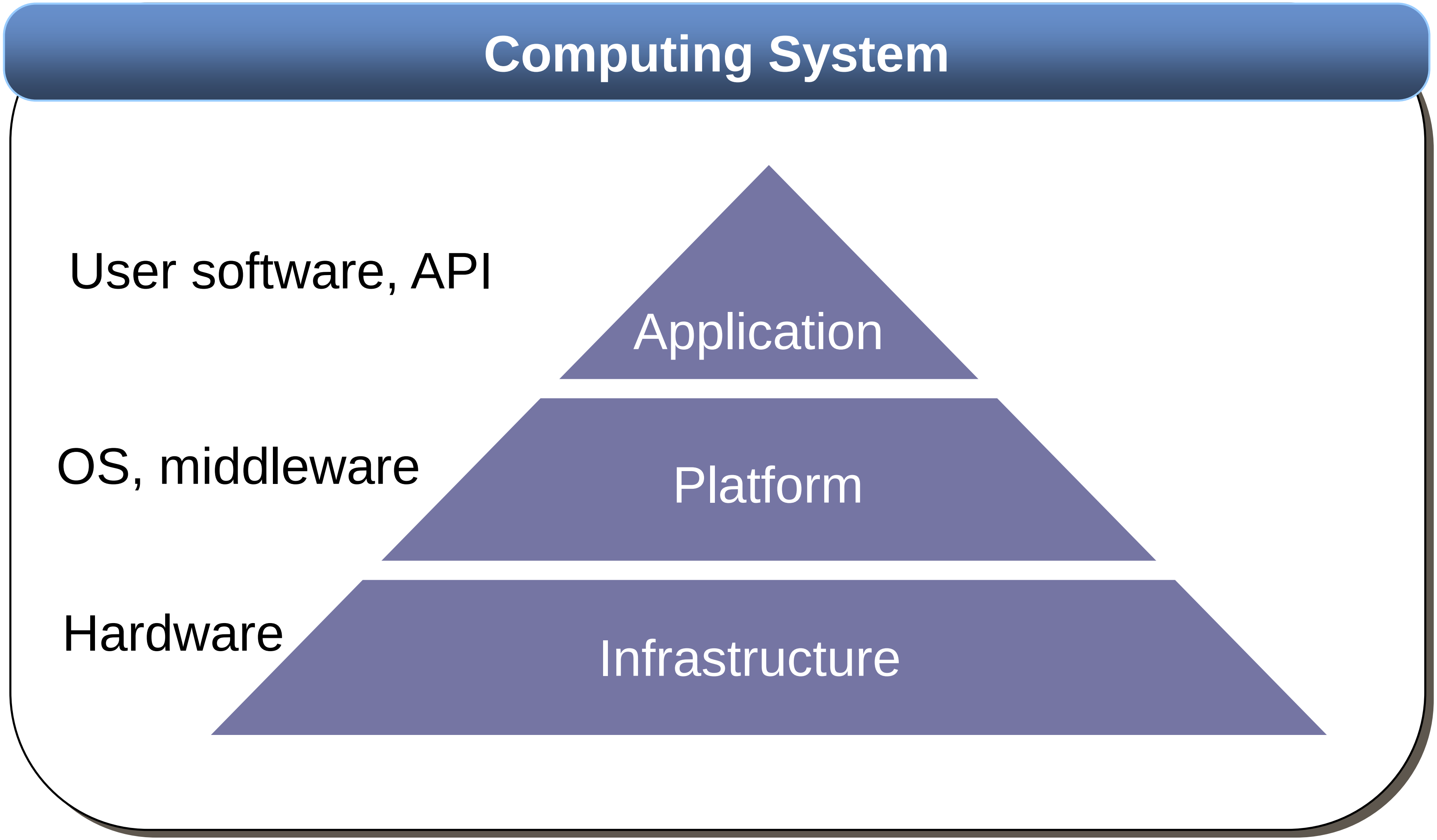
Computing and storage outsourcing

Different service requirements in different times

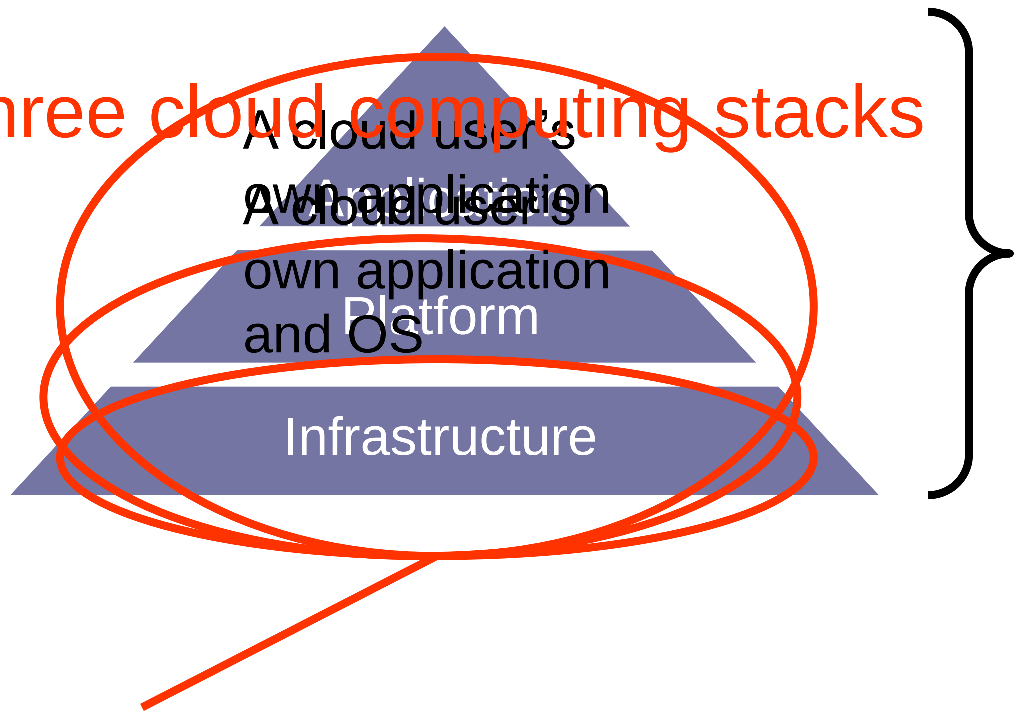
# of hardware machines
# of maintainers

Cost down!!!

1.4 Cloud Computing System Architecture



1.4 Cloud Computing System Architecture



SaaS: Software as a service

PaaS: Platform as a service

IaaS: Infrastructure as a service

Provided by the cloud computing providers

1.5 Cloud Computing Industry

- Amazon Web Services (AWS)
- Microsoft's Windows Azure Platform
- Google Cloud: Cloud Computing Services
- Alibaba Cloud
- Tencent Cloud
- Other small startups: Heroku & Engine Yard
- And more (and more)…

1.5 Cloud Computing Industry

- Amazon Web Services (AWS) https://aws.amazon.com/
  - Xen-based virtual computing environment
    - *A user can run Linux-based applications*
  - IaaS: A user can control nearly the entire software stack, from the kernel upwards.
  - Provides low level of virtualization
    - *Raw CPU cycles, block-device storage, IP-level connectivity*
- Microsoft's Windows Azure Platform
- Google Cloud: Cloud Computing Services
- Other small startups: Heroku & Engine Yard
- And more (and more)…

1.5 Cloud Computing Industry

- Amazon Web Services (AWS)
- Microsoft's Windows Azure Platform https://azure.microsoft.com/
  - PaaS: Use Windows Azure Hypervisor as the infrastructure, and use the .NET framework as the application container
  - Support general-purpose computing
  - Users can choose language, but cannot control the underlying operating system or runtime
- Google Cloud: Cloud Computing Services
- Other small startups: Heroku & Engine Yard
- And more (and more)…

1.5 Cloud Computing Industry

- Amazon Web Services (AWS)
- Microsoft's Windows Azure Platform
- Google Cloud: Cloud Computing Services https://cloud.google.com/
  - Support APIs for the Google Datastore, Google Accounts, URL fetch, image manipulation, email services, smart business and AI, etc.
  - SaaS: Application domain-specific platforms
    - *Not suitable for general-purpose computing*
- Other small startups: Heroku & Engine Yard
- And more (and more)…

1.5 Cloud Computing Industry

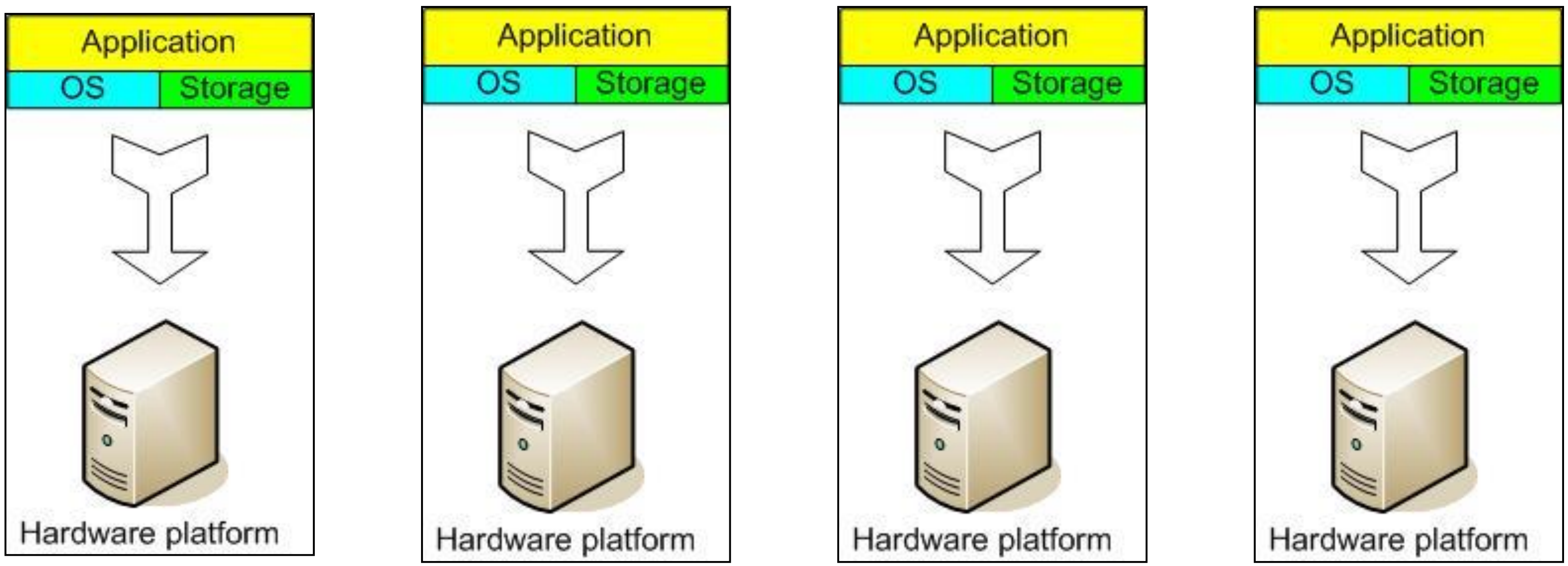
- Amazon Web Services (AWS)
- Microsoft's Windows Azure Platform
- Google App Engine
- Other small startups: Heroku & Engine Yard
  - Based on Ruby on Rails
  - PaaS
- And more (and more)…

  Alibaba Cloud, Huawei Cloud, Tencent Cloud, Baidu Cloud

2 Virtualization Concept

- Virtualization refers to the act of creating a virtual (rather than actual) version of
  - computer hardware platform
  - operating system (OS)
  - storage device
  - database server
  - computer network resources
- We can virtualize physical servers to support all applications running over them
- Virtualization of the key to Cloud Computing

2.1 The Traditional Server Concept

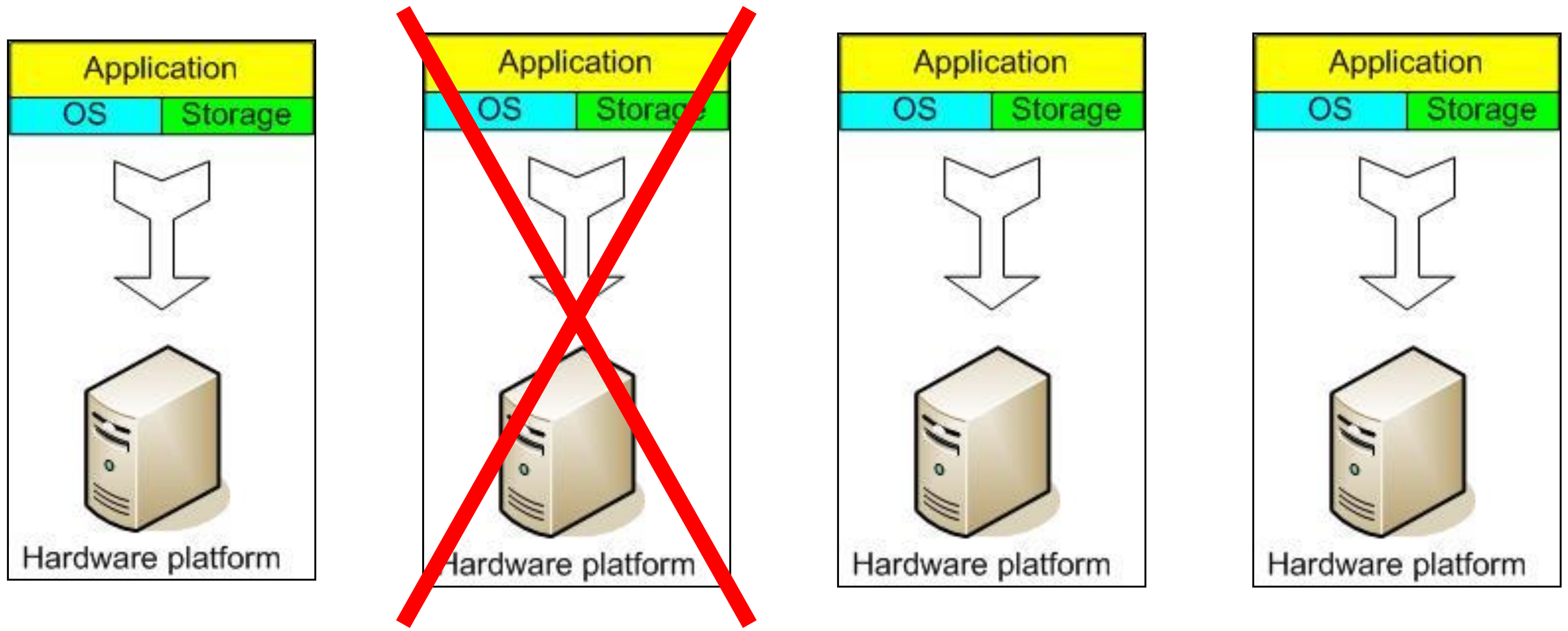


Web Server	**App Server**	**DB Server**	**EMail**
Windows	**Linux**	**Linux**	**Windows**
IIS	**Glassfish**	**MySQL**	**Exchange**

2.1 The Traditional Server Concept

- System administrators often talk about servers as a whole unit
  - Including the hardware, the OS, the storage, and the applications
- Servers are often referred to by their functions
  - e.g., the Web servers, the SQL servers
  - If the SQL servers are overloaded, the administrator must add in a new one

2.1 If Something Going Wrong…

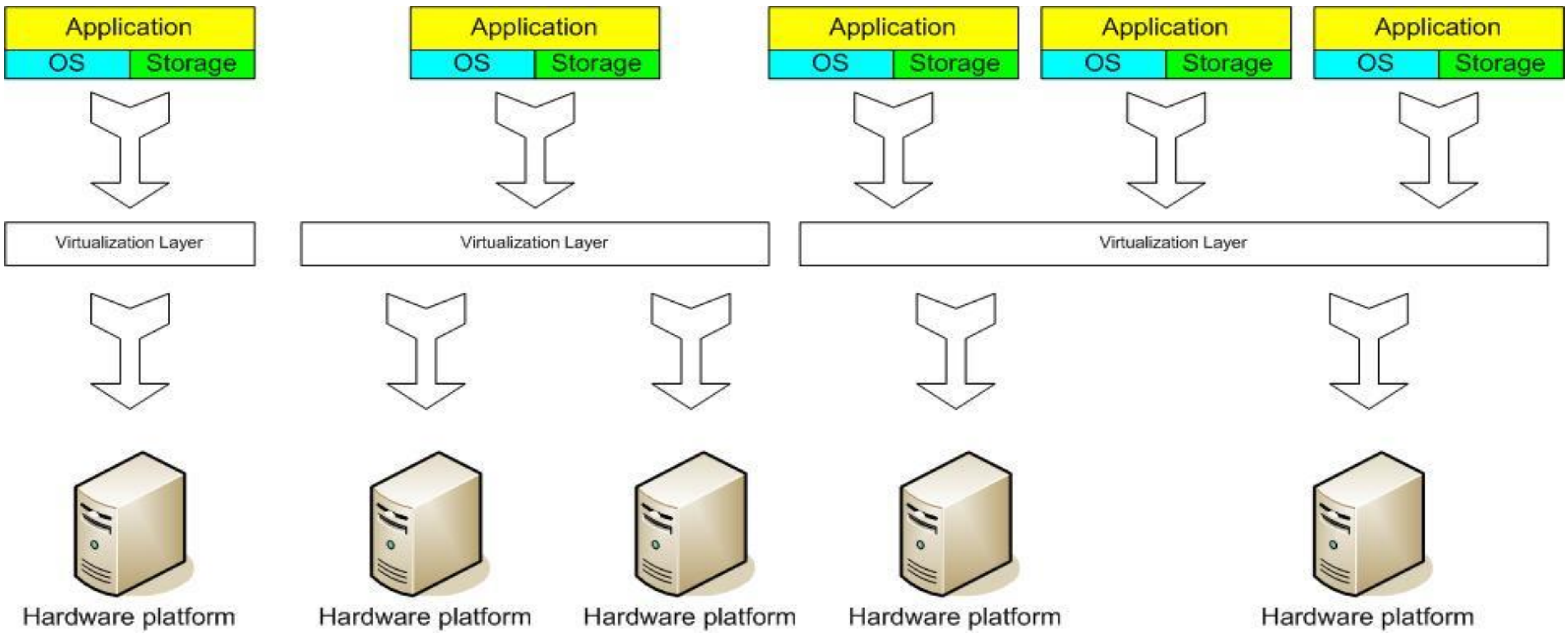


Web Server	App Server	DB Server	EMail
Windows	DOWN!	Linux	Windows
IIS		MySQL	Exchange

2.1 The Traditional Server Concept

- If server failure is experienced
  - Unless there are redundant servers, the whole service is down
- The administrators can implement clusters of servers
  - Make the service more fault tolerant
  - However, even clusters are not scalable
  - And, resource utilization is typically low

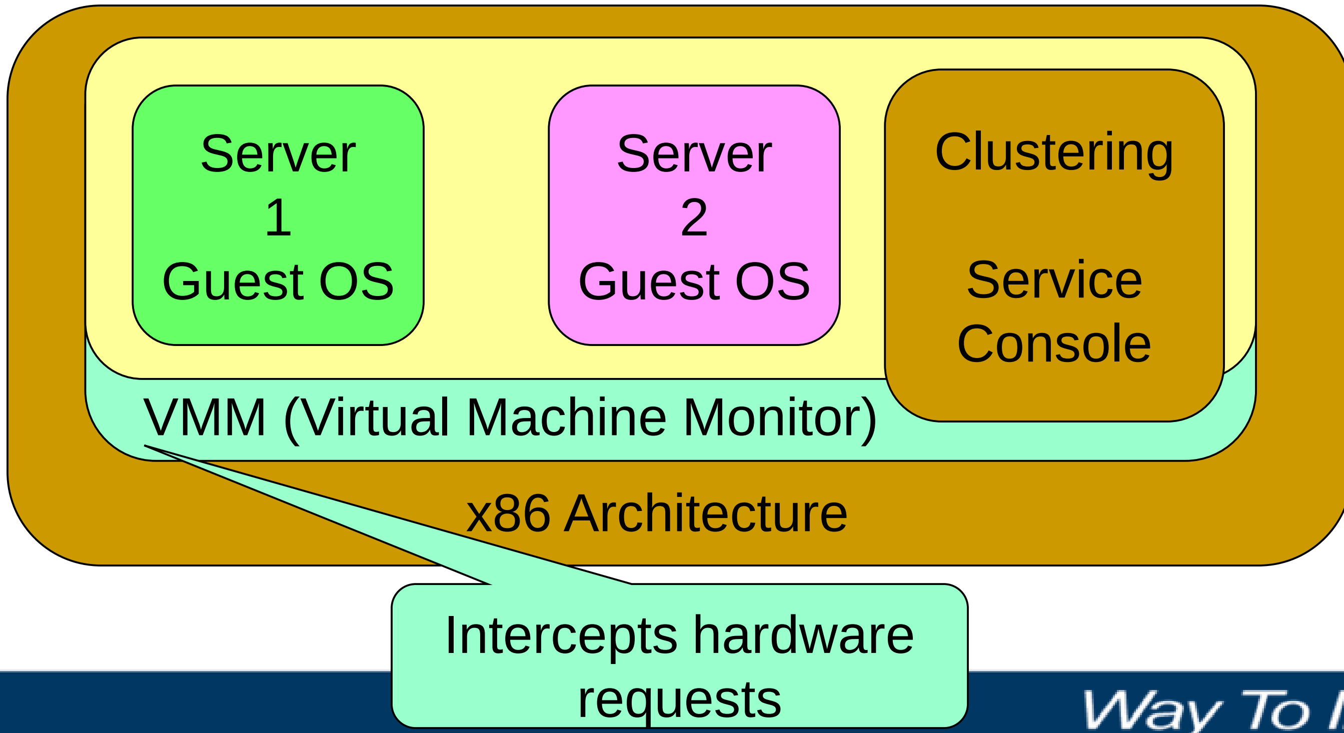
2.2 The Virtual Server Concept



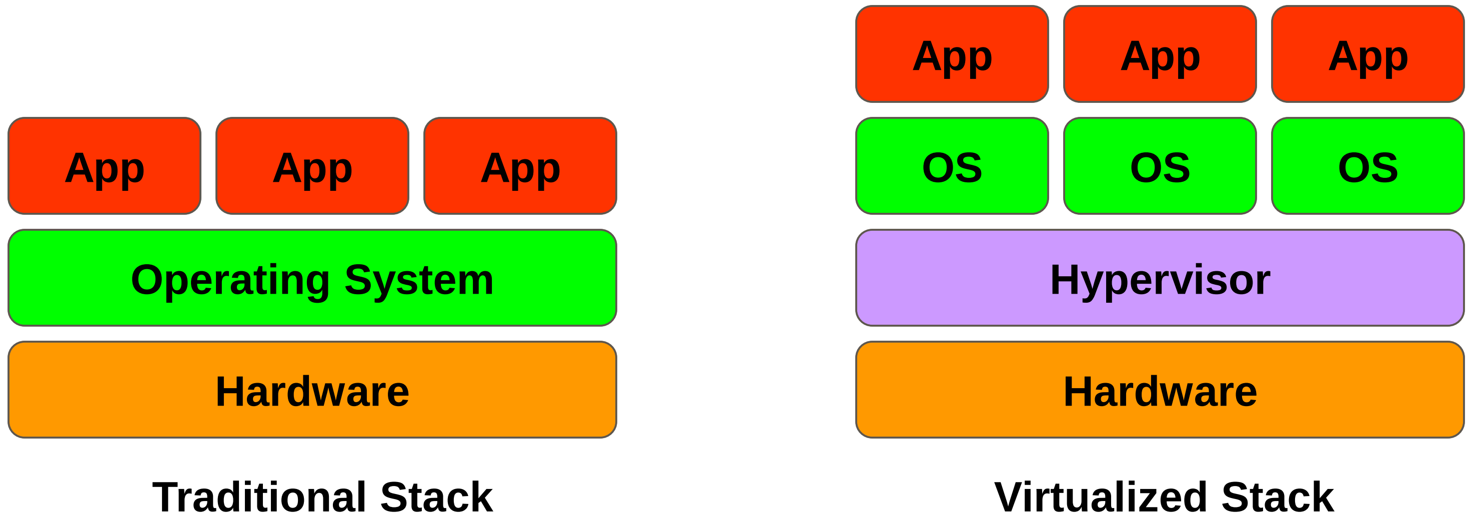
Virtual Machine Monitor (VMM) layer between *Guest OS* and hardware

2.2 The Virtual Server Concept

- Virtual servers encapsulate the service software away from the hardware
  - A virtual server can be hosted in one or more hardware machines
  - One hardware machine can host more than one virtual server



2.2 Virtualized Vs. Traditional



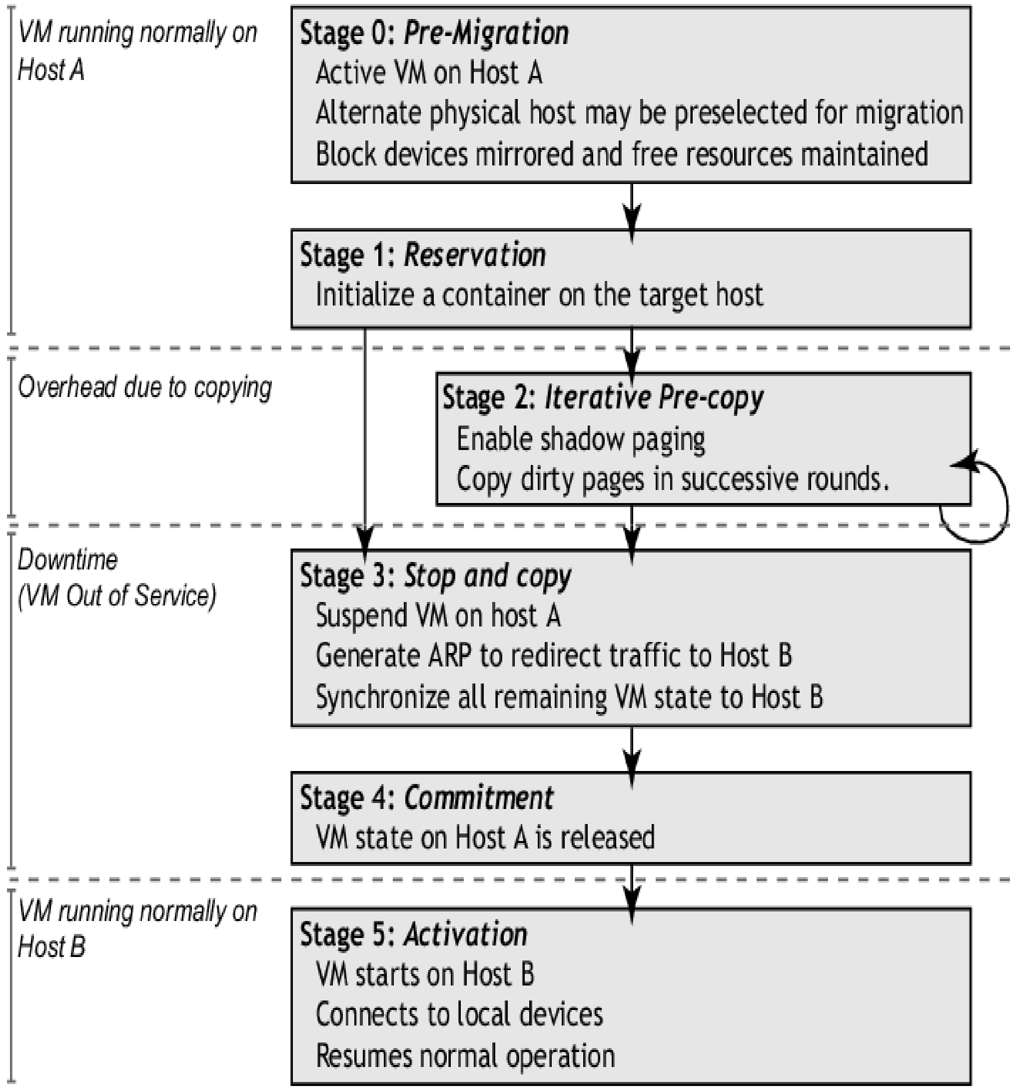
2.2 The Virtual Server Concept

- A flexible way to host computing systems
  - Virtual servers are just files in the storage
    - *They can be migrated from one machine to another easily*
  - Hardware machines can be removed and introduced conveniently
  - Administrators can adjust the amount of resources allocated to each virtual server
    - *Accommodate to the service requirements*

2.3 Virtualization Technique Supports

- Software implementations
  - Commercial software
    - *e.g., VMWare*
  - Open-source software
    - *e.g., XEN, VirtualBox*
- Hardware support
  - Intel VT (Intel Virtualization Technology)
  - AMD-V (AMD-Virtualization)
- Virtualization is a well-established technique

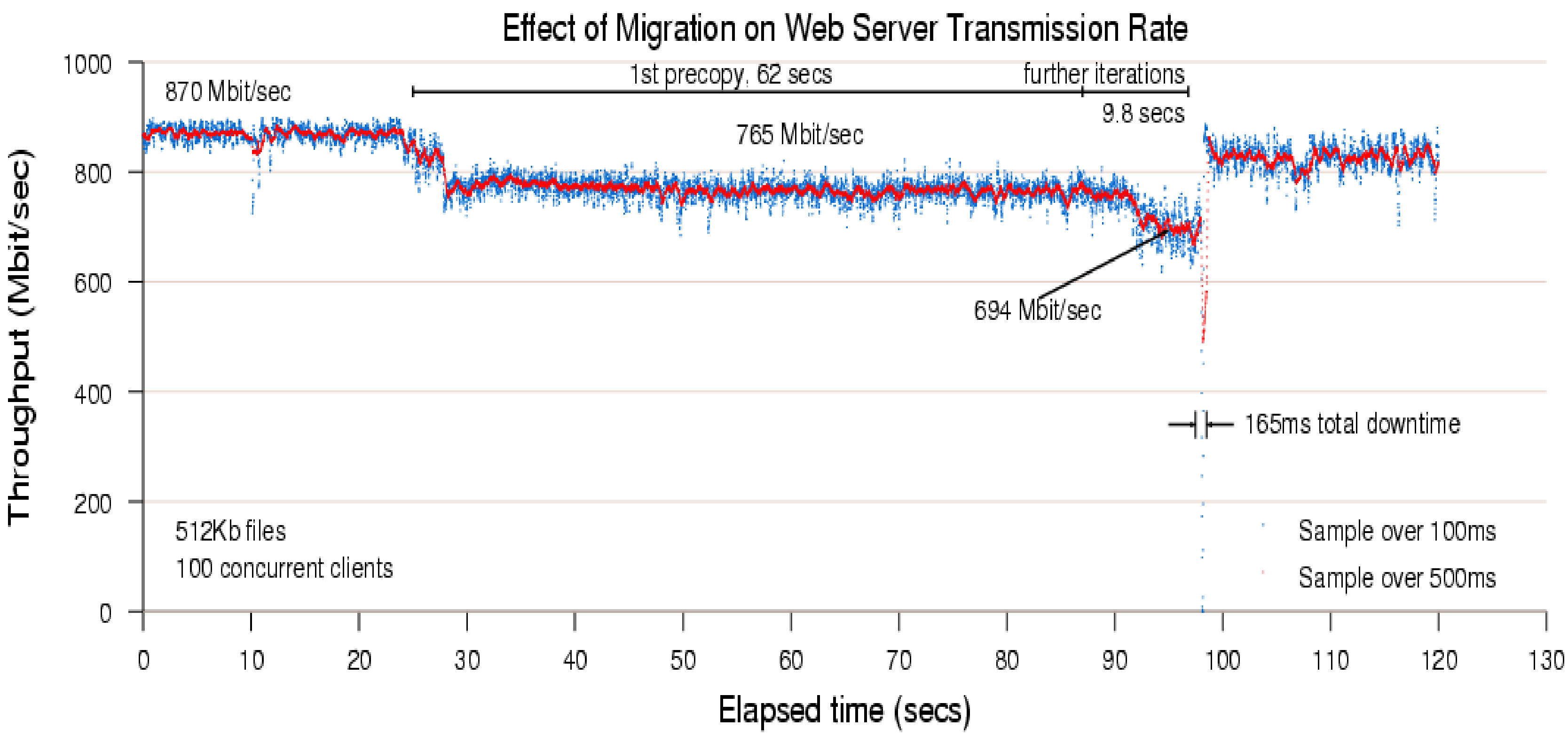
2.4 Live Migration from Virtualisation Perspective



Source: Clark et al, Live migration of virtual machines, NSDI 2005

Clark et al. Live migration of virtual machines, NSDI 2005.

2.4 Effects of Live Migration



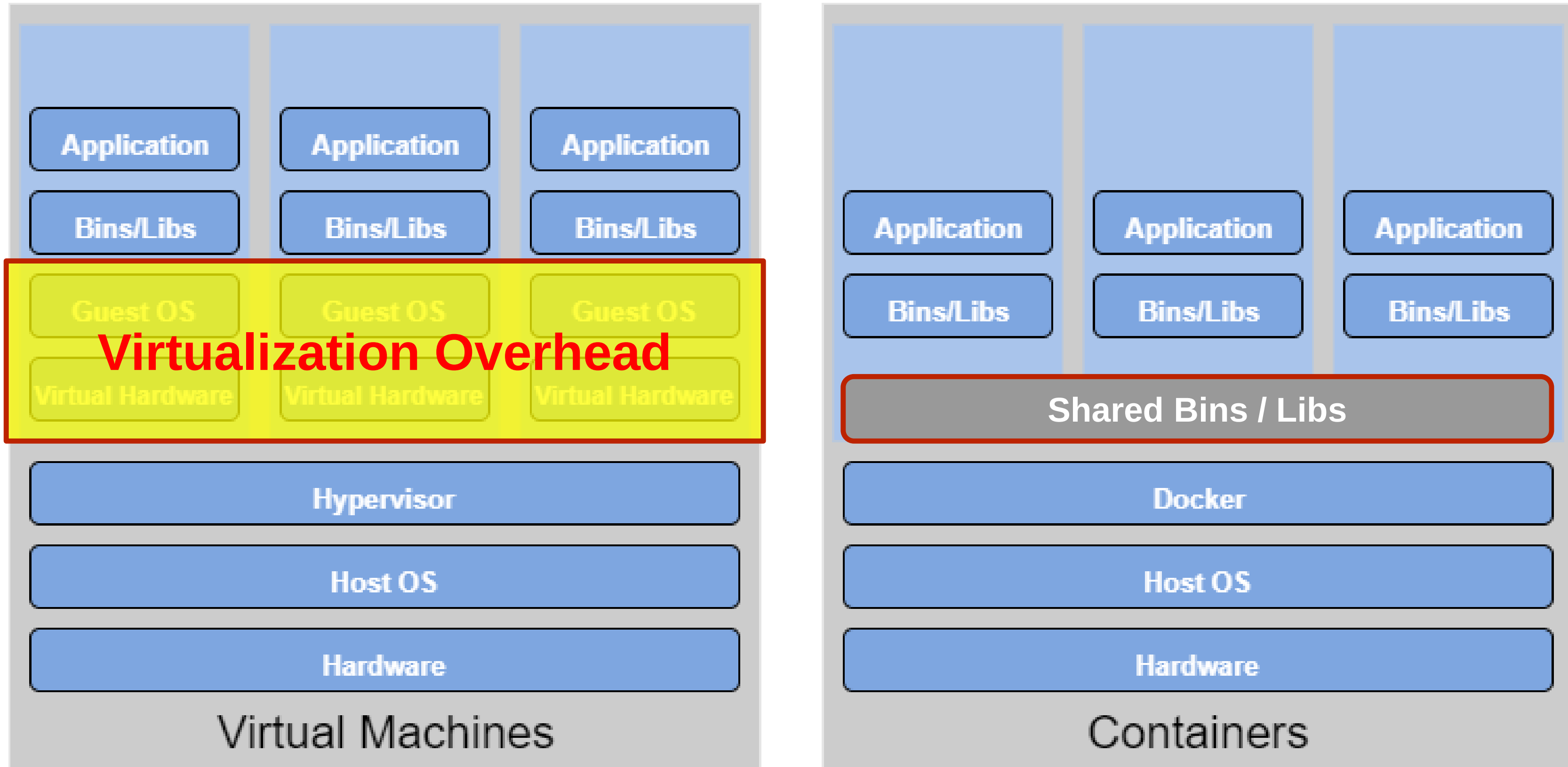
Source: Clark et al, Live migration of virtual machines, NSDI 2005

2.5 Virtualization vs Containerization

- The many advantages of virtualization, such as application containment and horizontal scalability, come at a cost: resources. The guest OS and binaries can give rise to duplications between VMs wasting server processors, memory and disk space and limiting the number of VMs each server can support.

- Containerization allows ***virtual instances*** to share a single host OS (and associated drivers, binaries, libraries) to reduce these wasted resources since each container only holds the application and related binaries. The rest are shared among the containers.

2.5 Virtualization vs Containerization

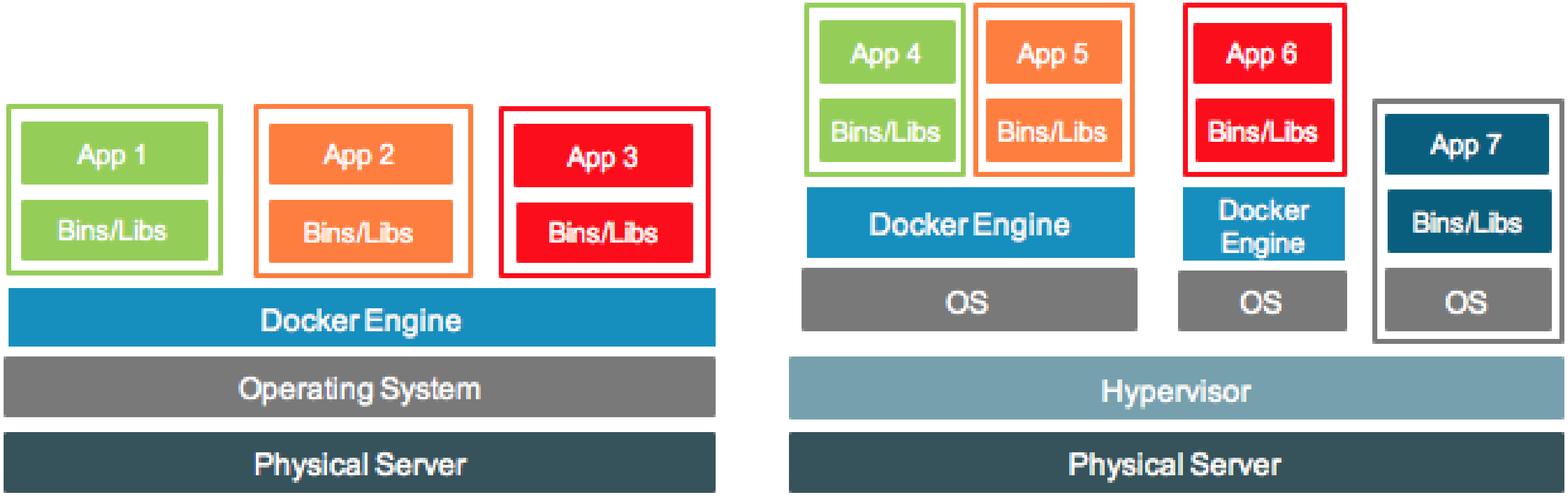


2.5 Virtualization vs Containerization

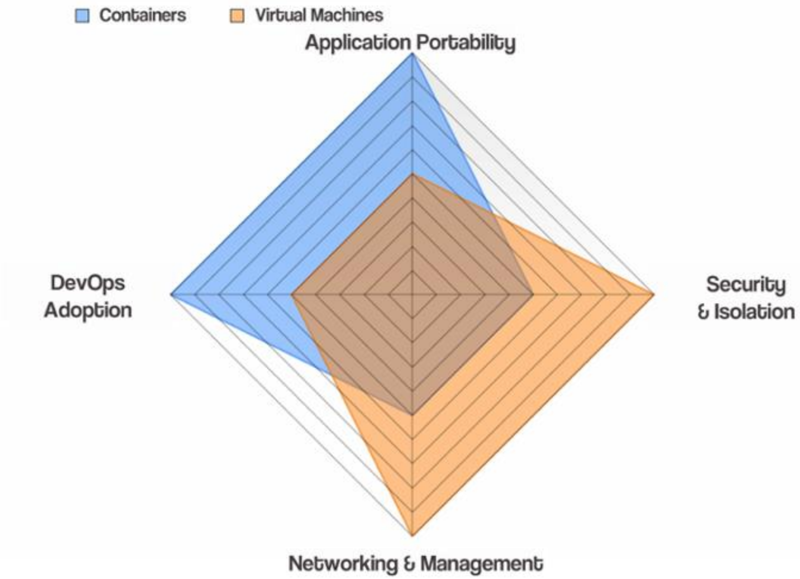
Parameter	Virtual Machines	Containers
Guest OS	Run on virtual HW, have their own OS kernels	Share same OS kernel
Communication	Through Ethernet devices	IPC mechanisms (pipes, sockets)
Security	Depends on the Hypervisor	Requires close scrutiny
Performance	Small overhead incurs when instructions are translated from guest to host OS	Near native performance
Isolation	File systems and libraries are not shared between guest and host OS	File systems can be shared, and libraries are
Startup time	Slow (minutes)	Fast (a few seconds)
Storage	Large size	Small size (most is re-use)

2.5 In the Real World, VMs and Containers can/do Co-exist

When deploying applications on the cloud, the base computation unit is a Virtual Machine. Usually Docker containers are deployed on top of VMs.



2.5 Are Containers Better than VMs?



It depends …

- The size of the task on hand
- The life span of the application
- Security concerns
- Host operation system, e.g. running Docker on Windows servers

Containers vs Virtual Machines: Which is a better fit for you?

Containers vs Virtual Machines: How to tell which is the right choice for your enterprise?

2.5 What is a Container?

- Similar concept of resource isolation and allocation as a virtual machine.

- Without bundling the entire hardware environment and full OS.

2.5 What container runtimes are in use?

- Container technologies: Docker, Rocket (rkt), Mesos, LxD ...

- **Docker** is currently the leading software container platform

2.5 What are Container Orchestration Tools?

Container orchestration technologies provides a framework for integrating and managing containers ***at scale***

Features:

•Networking
•Scaling
•Service discovery and load balancing
•Health check and self-healing
•Security
•Rolling updates

Goals:

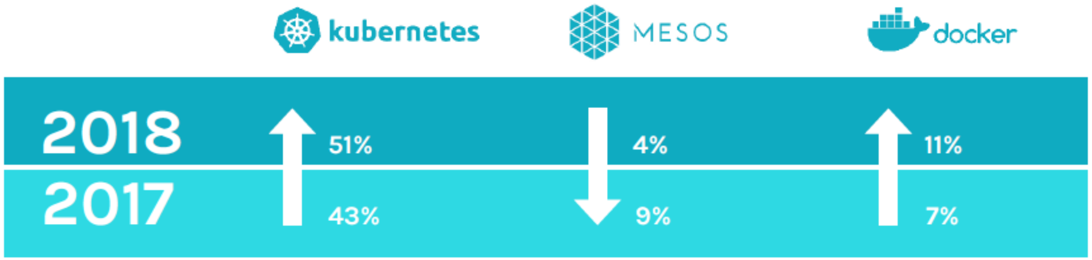
•Simplify container management processes
•Help to manage availability and scaling of containers

2.5 Container Orchestration Tools

- **Kubernetes and Hosted Kubernetes:**
  - Self-hosted Kubernetes
  - Amazon's Elastic Kubernetes Service (EKS)
  - Google Kubernetes Engine (GKE)
  - Azure Kubernetes Service (AKS)

- **Docker SWARM**

- **Others:**
  - Amazon's Elastic Container Service (ECS)
  - Azure Container Service (ACS, RIP in Jan 2020, replaced by AKS)
  - Mesos / Marathon



3 Large-scale Computing in Cloud

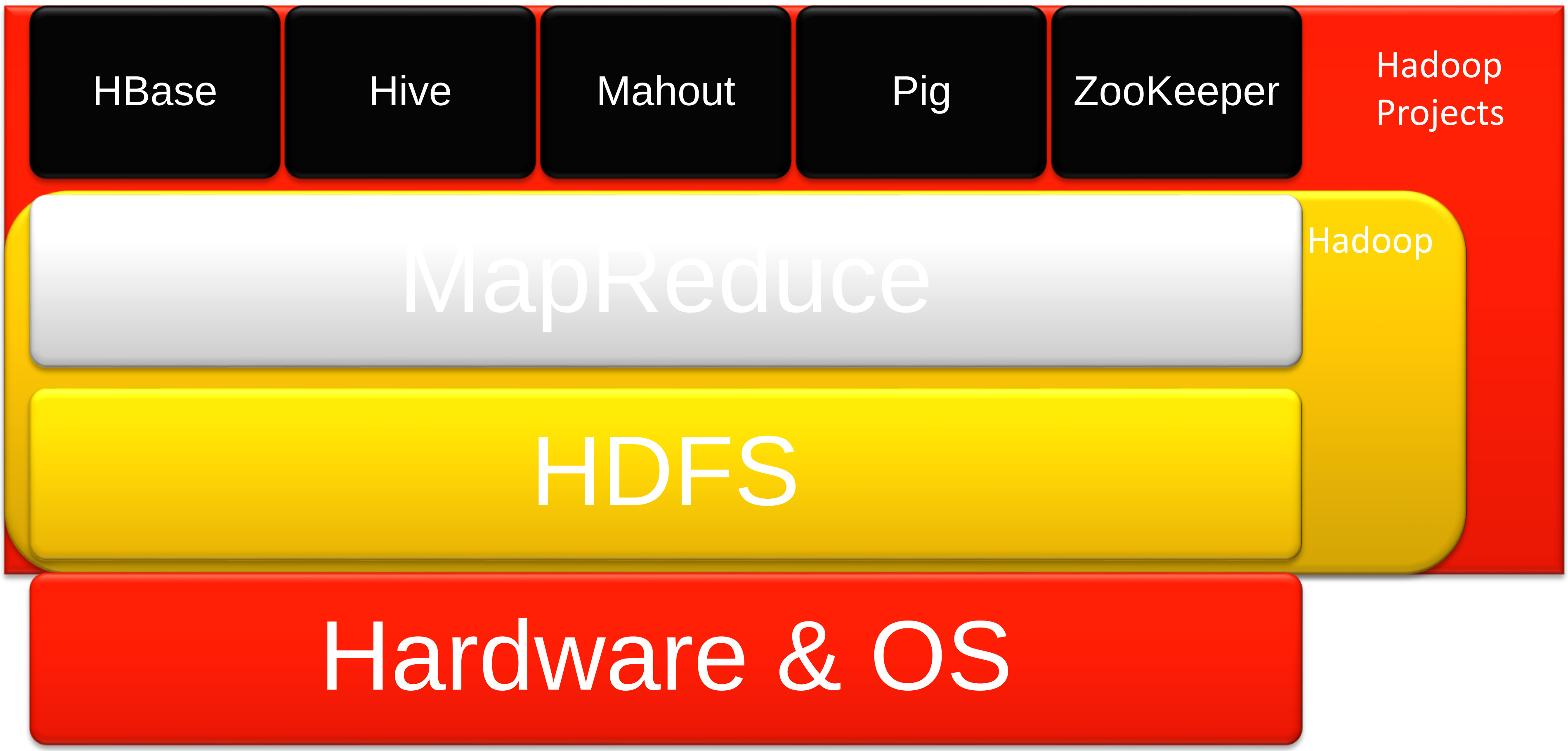
- Large-scale computing for data mining problems on commodity hardware

- Challenges:
  - How do you distribute computation?
  - How can we make it easy to write distributed programs?
  - How can you handle machine failures?
    - *One server may stay up 3 years (1,000 days)*
    - *If you have 1,000 servers, expect to lose 1/day*
    - *It is estimated that Google had 2.5M machines in 2016*
      - 2,500 machine fails every day!

3.1 Idea and Solution

- Issue: Copying data over a network takes time
- Idea:
  - Bring computation close to the data
  - Store files multiple times for reliability
- MapReduce addresses these problems
  - Google's computational/data manipulation model
  - Elegant way to work with big data
  - Storage Infrastructure – File system
    - *Google: GFS; Hadoop: HDFS*
  - Programming model
    - *Map-reduce*

Relationship



3.3 The Hadoop Distributed File System (HDFS)

- With hundreds of machines at hand, failure is the norm rather than exception
- Traditional file storage system cannot cope with the scale and failure faced by large clusters
- The Hadoop Distributed File System (HDFS) is a natural solution to this problem
  - Distributed File System
  - Provides global file namespace
  - Replica to ensure data recovery

3.3 The Hadoop Distributed File System (HDFS)

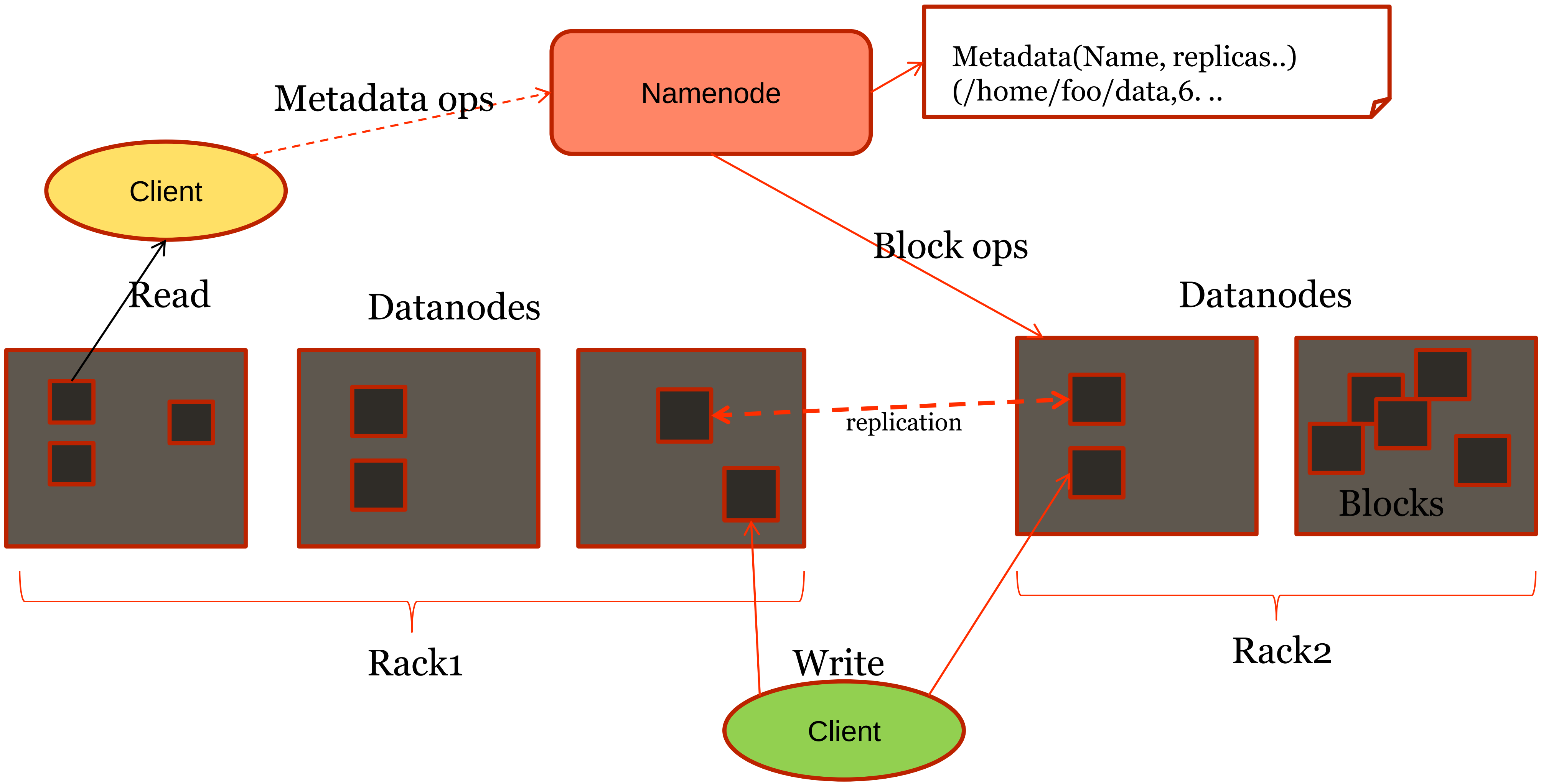
- A HDFS instance may consist of thousands of server machines, each storing part of the file system's data.
- Since we have huge number of components, and each component has non-trivial probability of failure, it means that there is always some component that is non-functional.
- Detection of faults and quick, automatic recovery from them are a core architectural goal of HDFS.

3.3 Data Characteristics

- Streaming data access
- Batch processing rather than interactive user access
- Write-once-read-many: a file, once created, written and closed, need not be changed – this assumption simplifies coherency

- Master/slave architecture
  - Master: NameNode
  - Slave: DataNode
- HDFS exposes a file system namespace (NameNode) and allows user data to be stored in files.
- A file is split into one or more blocks and set of blocks are stored in DataNodes.

HDFS Architecture



3.5 File System Namespace

- Namenode maintains the file system.
  - Hierarchical file system with directories and files.
  - Create, remove, move, rename, etc.
  - Any meta information changes to the file system is recorded by the Namenode.
- An application can specify the number of replicas of the file needed: replication factor of the file. This information is stored in the Namenode.

3.5 Data Replication

- HDFS is designed to store very large files across machines in a large cluster.
  - Each file is a sequence of blocks.
  - All blocks in the file except the last are of the same size.
- Blocks are replicated for fault tolerance.
- Block size and replicas are configurable per file.
- The NameNode receives a Heartbeat and a BlockReport from each DataNode in the cluster.
- BlockReport contains all the blocks on a DataNode.

3.6 Replica Selection

- Replica selection for read operation: HDFS tries to minimize the bandwidth consumption and latency.
- If there is a replica on the Reader node then that is preferred.
- HDFS cluster may span multiple data centers: replica in the local data center is preferred over the remote one.

3.6 Safemode Startup

- Each DataNode checks in with Heartbeat and BlockReport.
- NameNode verifies that each block has acceptable number of replicas
- After a configurable percentage of safely replicated blocks check in with the NameNode, NameNode exits Safemode.
- It then makes the list of blocks that need to be replicated.
- NameNode then proceeds to replicate these blocks to other DataNodes.

3.7 Filesystem Metadata

- The HDFS namespace is stored by NameNode.
- NameNode uses a transaction log called the EditLog to record every change that occurs to the filesystem meta data.
  - For example, creating a new file
  - Change replication factor of a file
  - EditLog is stored in the NameNode's local filesystem

3.7 NameNode

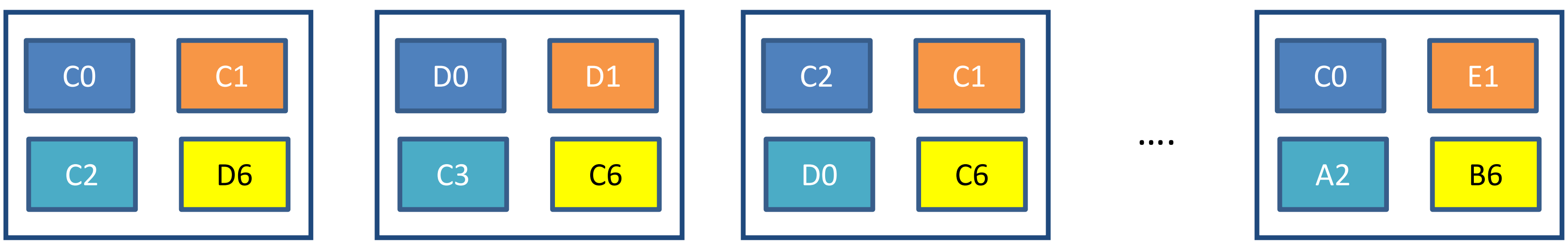
- Keeps image of entire file system namespace.
- When the Namenode starts up
  - Gets the FsImage and Editlog.
  - Update FsImage with EditLog information.
  - Stores a copy of the FsImage as a checkpoint.
- In case of crash
  - Last checkpoint is recovered.

3.7 DataNode

- A DataNode stores data in files in its local file system.
  - Each block of HDFS is a separate file.
  - These files are placed in different directories.
  - Creation of new directory is determined by heuristics.
- When the filesystem starts up:
  - Generates Blockreport.
  - Sends this report to NameNode.

3.7 DataNode

- Reliable distributed file system
- Data kept in “chunks” spread across machines
- Each chunk replicated on different machine and racks
  - Seamless recovery from disk or machine failure.



4 MapReduce

- Warm-up task
  - We have a huge text document
  - Count the number of times each distinct word appears in the file
- Sample application
  - Analyze web server logs to find popular URLs

4.1 Task: Word Count

- Using Unix tool chain, we can count the occurrences of words:
  - `Words (doc.txt) | sort | uniq -c`
    - *Where `words` takes a file and outputs the words in it, one per line*
- This way of counting captures the essence of MapReduce
  - Mapper (done by words)
  - Group by keys and sort (done by sort)
  - Reducer (done by uniq)
  - Hadoop handles the partition and parallelization

4.1 MapReduce: Overview

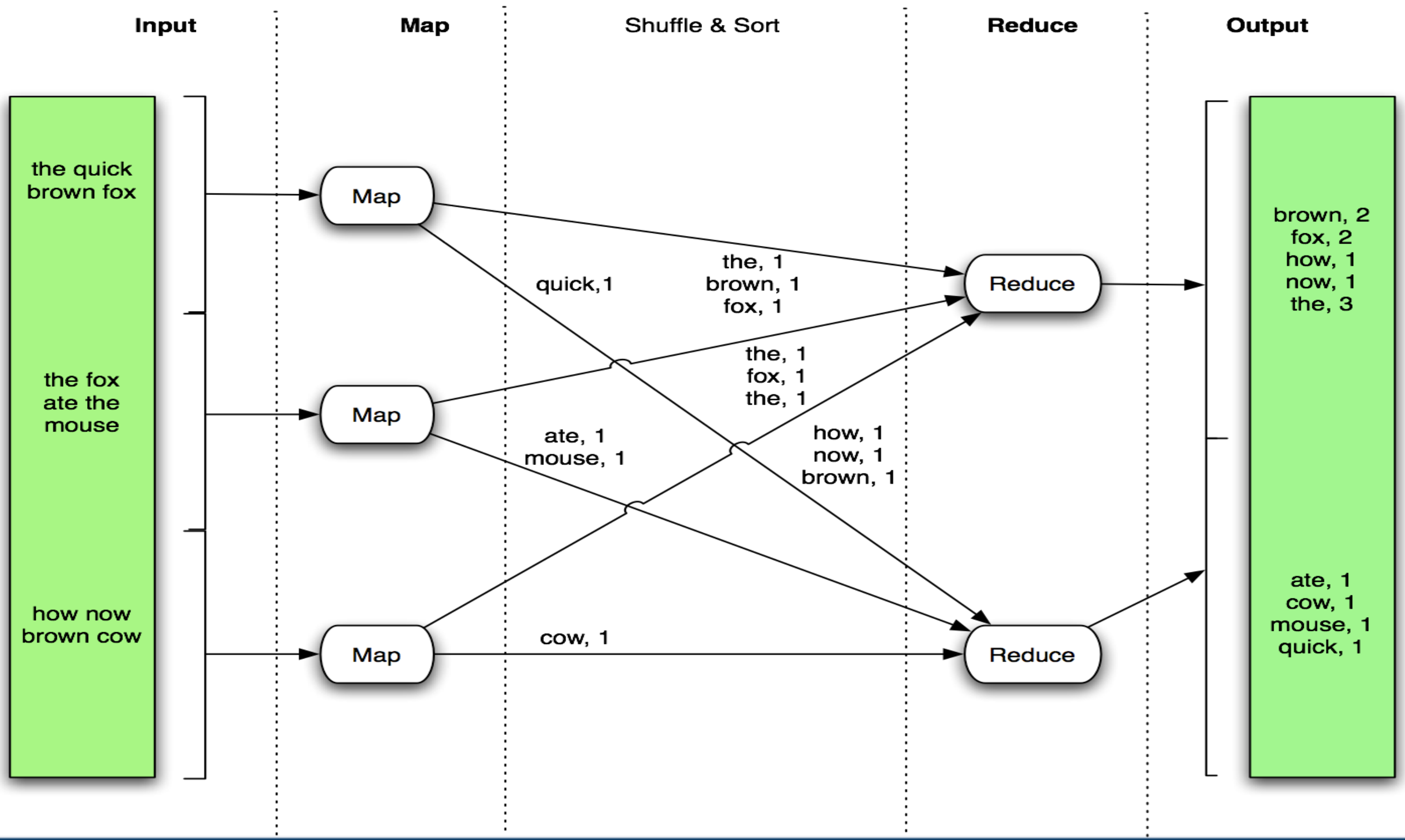
- Sequentially read a lot of data
- Map:
  - Extract something you care about
- Group by key: Sort and Shuffle
- Reduce:
  - Aggregate, summarize, filter or transform
- Write the result

4.1 MapReduce

- Input: a set of key-value pairs
- Programmer must specifies two methods:
  - Map(k,v) -> <k', v'>
    - *Takes a key-value pair and outputs a set of key-value pairs*
    - *There is one Map call for every (k,v) pair*
  - Reduce (k', <v'>) -> <k', v''>
    - *All values v' with the same key k' are reduced together and processed in v' order*
    - *There is one Reduce function call per unique key k'*

4.1 MapReduce: Word Count Example

- Now that one document changes to a large corpus of documents



4.2 MapReduce: Word Count Example

```
// key: document name; value: text of the document
Map(key, value):
  for each word w in value:
    emit(w, 1)

// key: a word; value: an iterator over counts
Reduce(key, values):
  result = 0
  for each count v in values:
    results += v
  emit(key, result)
```

4.3 When To Use MapReduce?

- You have a cluster of computers
- You are working with large dataset
- You are working with independent data (or assumed)
- The data processing can be casted into Map and Reduce operations

4.3 MapReduce Implementations

- MapReduce by Google Inc.
  - Centric to Google's searching engine
    - *Generate Google's index of the World Wide Web*
- Open-source implementations
  - Hadoop
    - *Yahoo! Search Webmap is a Hadoop application to produce data used in every Yahoo! Web search query*
  - And many others
    - *Disco, Skynet, …*

4.4 MapReduce: Environment

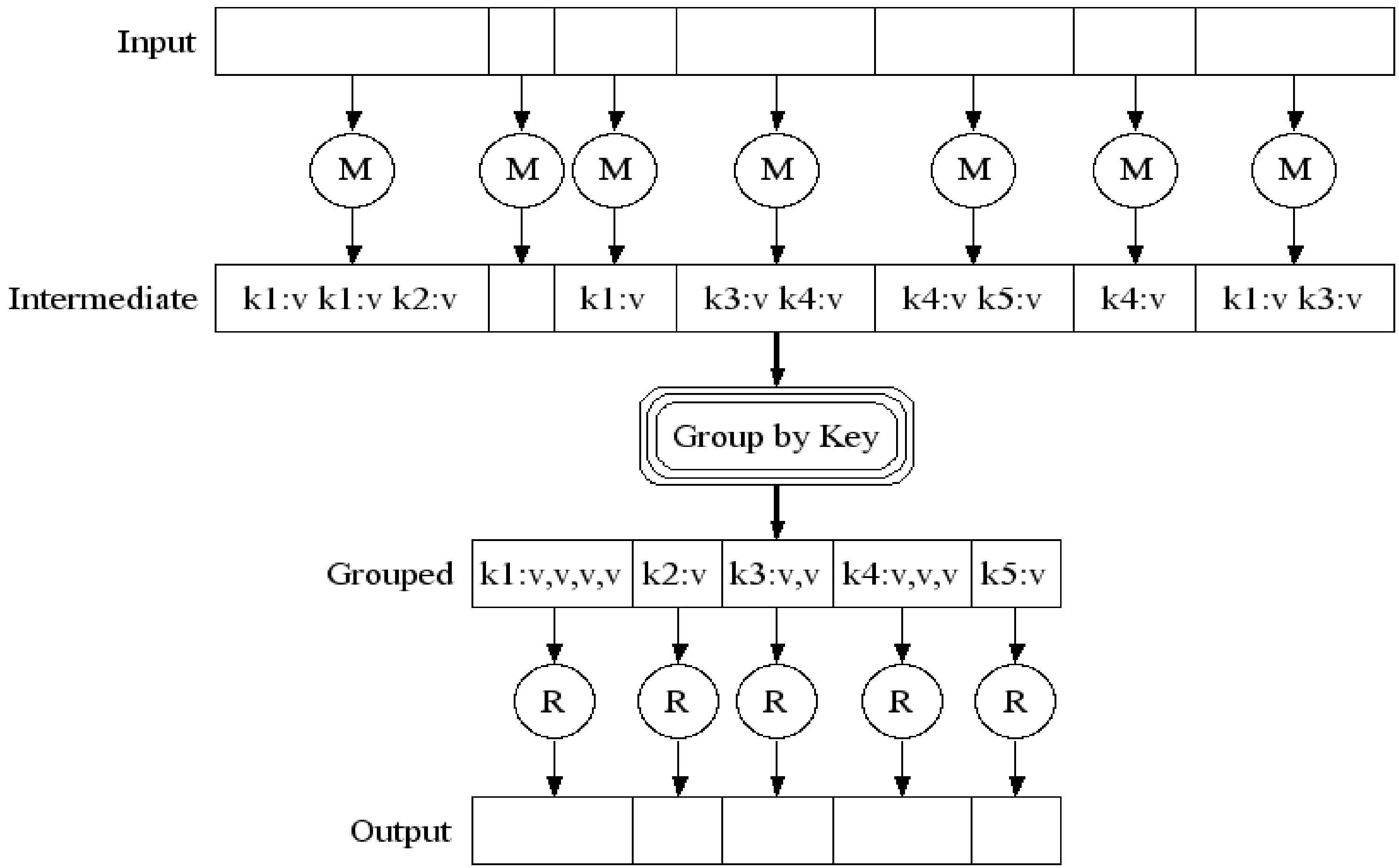
- MapReduce environment takes care of:
  - Partitioning the input data
  - Scheduling the program's execution across a set of machines
  - Performing the group by key step
  - Handling machine failures
  - Managing required inter-machine communication

4.4 MapReduce: Example

Map:
Read input and produces a set of key-value pairs

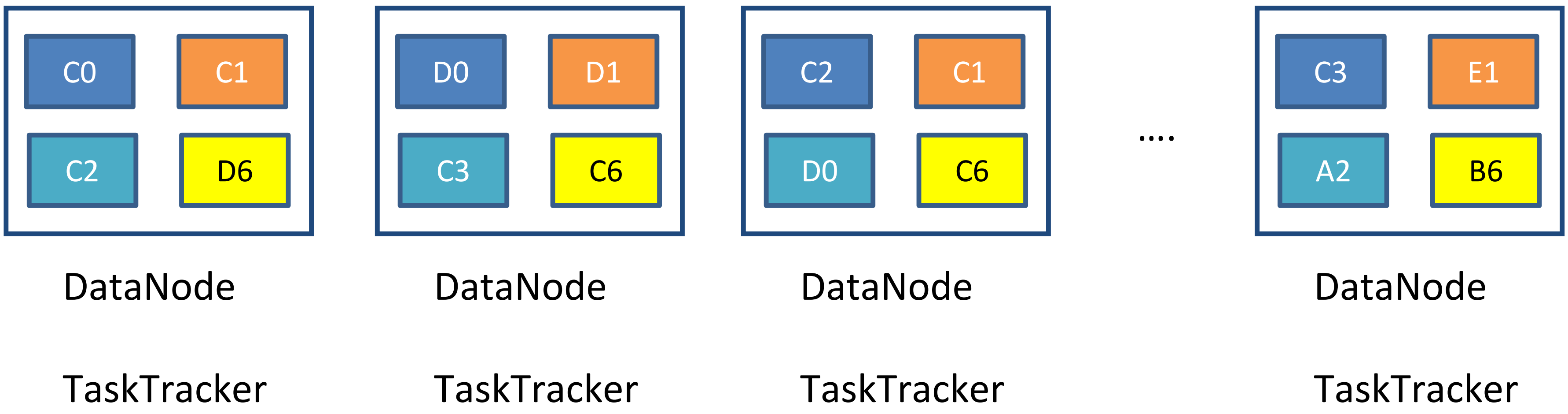
Group by key:
Collect all pairs with the same key

Reduce:
Collect all values belonging to the key and output



4.4 MapReduce on Hardoop

- Move computation to the data



Bring computation directly to the data!

DataNode also serve as compute servers

4.5 Data Flow

- Input and final output are stored on a distributed file system (FS):
  - 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 MapReduce task

4.5 Coordination: Master

- Master node takes care of coordination:
  - Task status: (idle, in-progress, completed)
  - Idle tasks get scheduled as workers become available
  - When a map task completes, it sends the master the location and sizes of its *R* intermediate files, one for each reducer
  - Master pushes this info to reducers
- Master pings workers periodically to detect failures

4.6 Dealing with Failures

- Map worker failure
  - Map tasks completed or in-progress at worker are reset to idle
  - Reduce workers are notified when task is rescheduled on another worker
- Reduce worker failure
  - Only in-progress tasks are reset to idle
  - Reduce task is restarted
- Master failure
  - MapReduce task is aborted and client is notified.

4.6 How Many Map and Reduce Jobs?

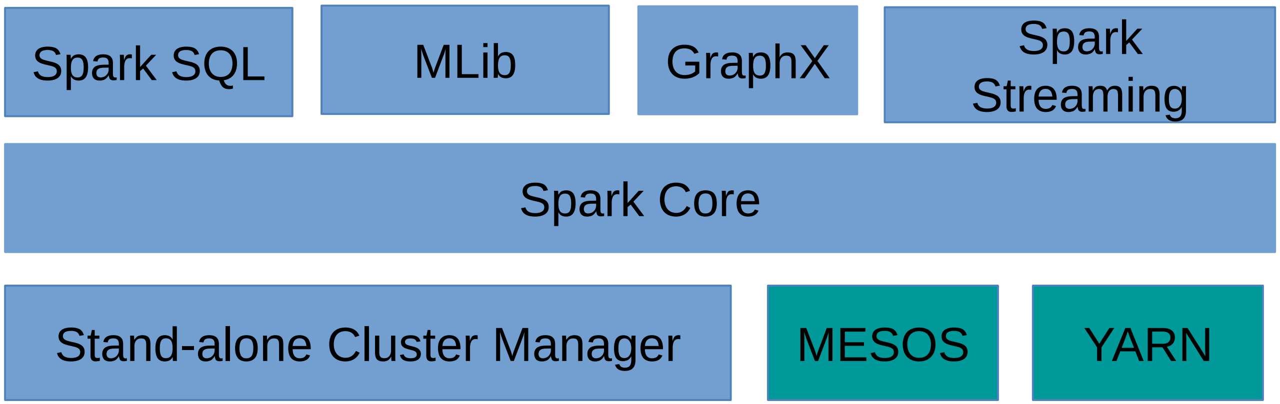
- M map tasks, R reduce tasks
- Rule of a thumb:
  - Make M much larger than the number of nodes in the cluster
  - One chunk per map is common
  - Improves dynamic load balancing and speeds up recovery from worker failures
- Usually R is smaller than M
  - Output is spread across R files

4.7 Spark

- While Hadoop MapReduce works well, it is geared towards performing relatively simple jobs on large datasets.
- However, when complex jobs are performed (say, machine learning or graph-based algorithms), there is a strong incentive for caching data in memory and in having finer-grained control on the execution of jobs.
- Apache Spark was designed to reduce the latency inherent in the Hadoop approach for the execution of MapReduce jobs.
- Spark can operate within the Hadoop architecture, using YARN and Zookeeper to manage computing resources, and storing data on HDFS.

4.7 Spark Architecture

- One of the strong points of Spark is the tightly-coupled nature of its main components:



- Spark ships with a cluster manager of its own, but it can work with other cluster managers, such as YARN or MESOS.

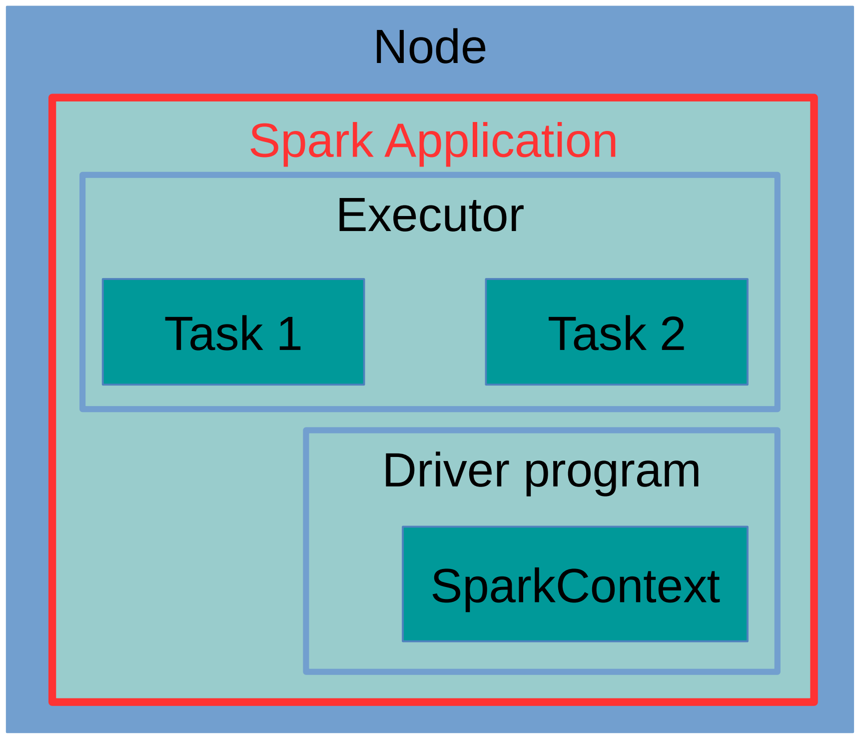
4.7 Spark Runtime Architecture

- Applications in Spark are composed of different components including:
  - ***Job***: the data processing that has to be performed on a dataset
  - ***Task***: a single operation on a dataset
  - ***Executors***: the processes in which tasks are executed
  - ***Cluster Manager***: the process assigning tasks to executors
  - ***Driver program***: the main logic of the application
  - ***Spark application***: Driver program + Executors
  - ***Spark Context***: the general configuration of the job

- These different components can be arranged in three different deployment modes across the cluster.

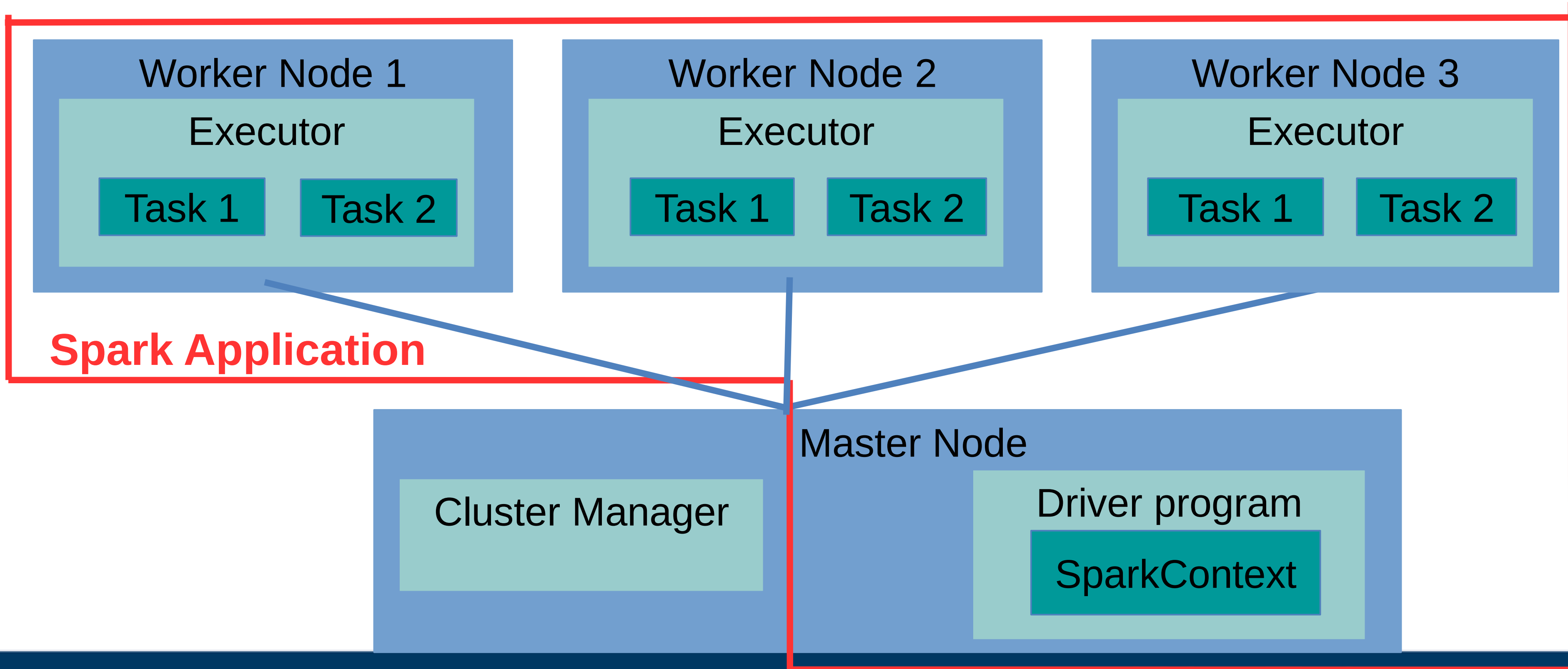
4.7 Spark Runtime Architecture: Local Mode

In local mode, every Spark component runs within the same JVM. However, the Spark application can still run in parallel, as there may be more than one executor active. (Local mode is good when developing/debugging)



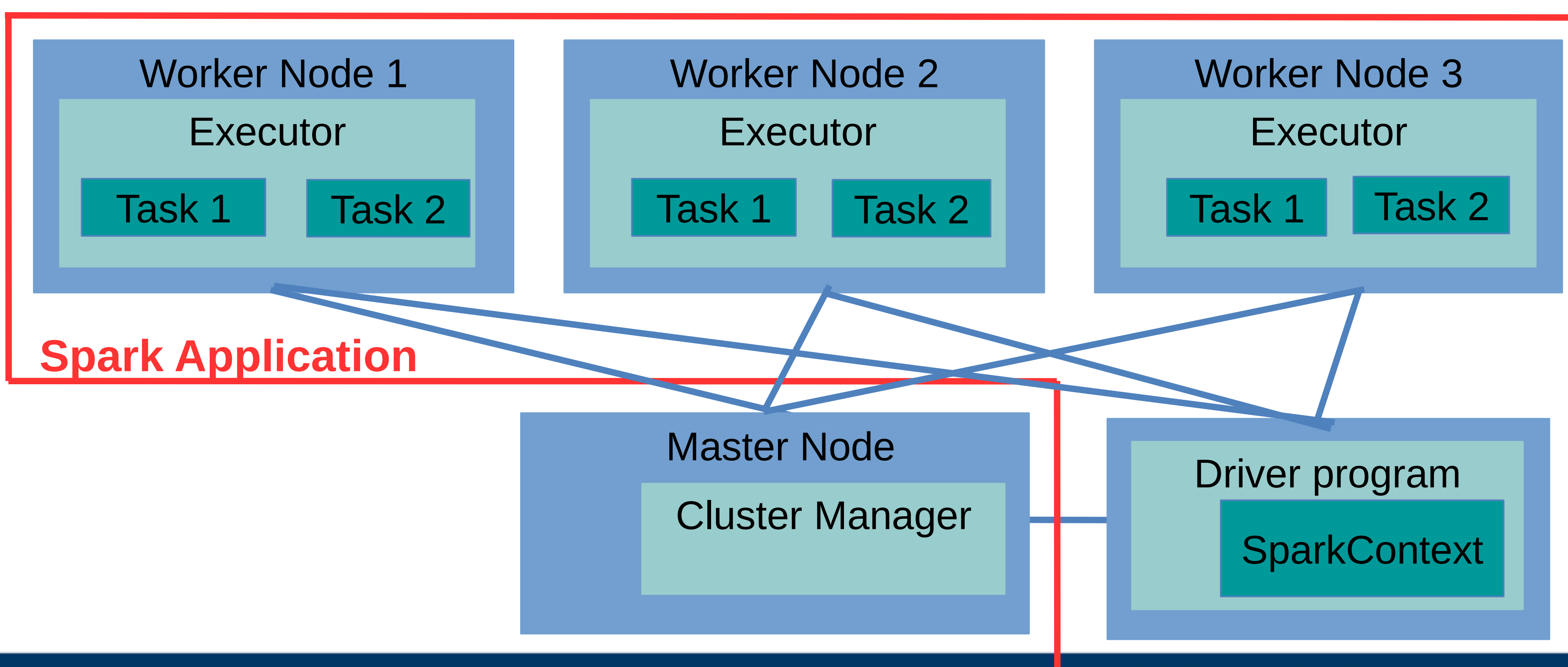
4.7 Spark Runtime Architecture: Cluster Mode

In cluster mode, every component, including the driver program, is executed on the cluster; hence, upon launching, the job can run autonomously. This is the common way of running non-interactive Spark jobs.

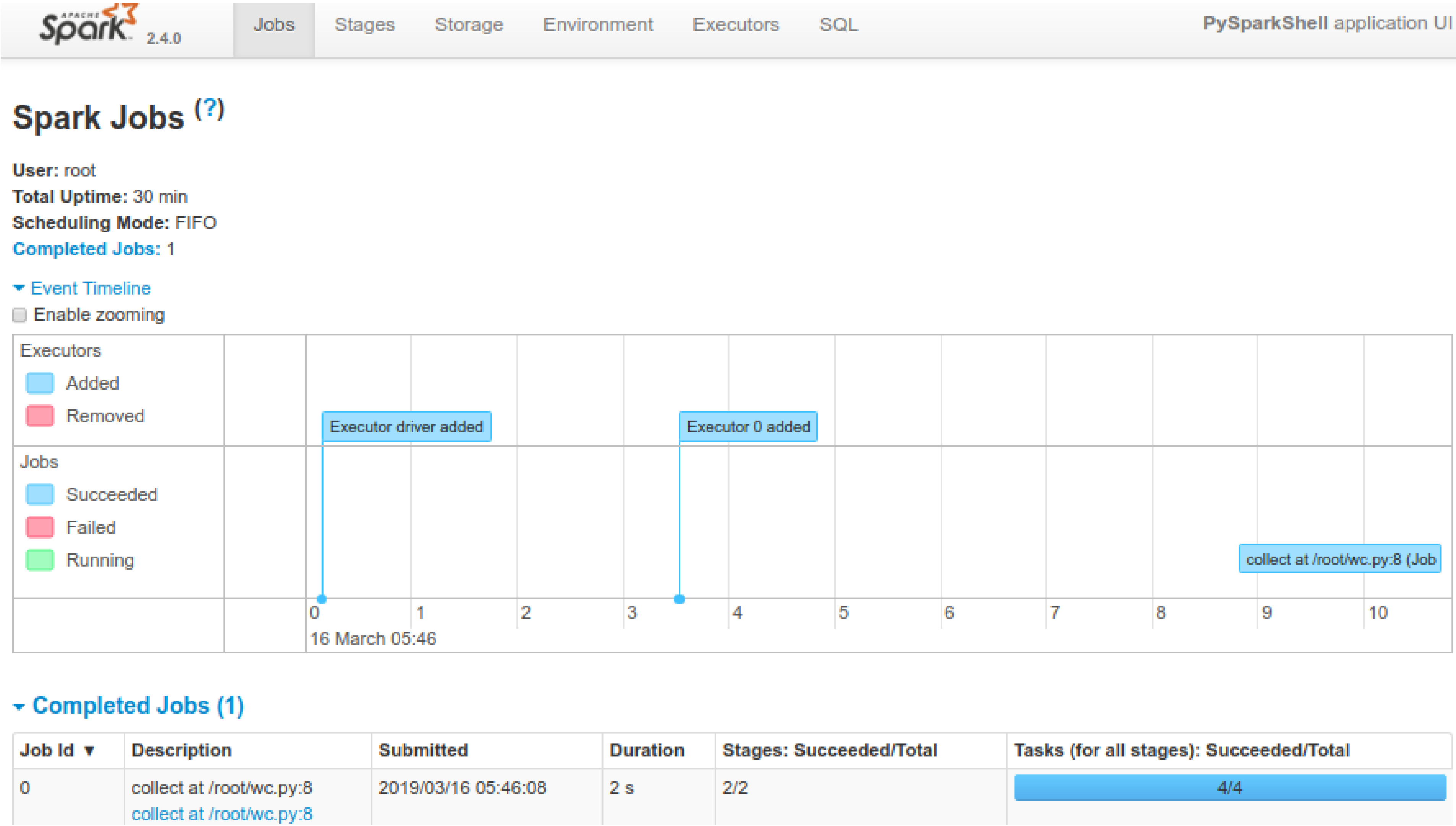


4.7 Spark Runtime Architecture: Client Mode

In client mode, the driver program talks directly to the executors on the worker nodes. Therefore, the machine hosting the driver program has to be connected to the cluster until job completion. Client mode must be used when the applications are interactive, as happens in the R, Python or Scala Spark shells.



4.7 Visualizing Spark Jobs: Jobs



Spark 2.4.0 Jobs Stages Storage Environment Executors SQL **PySparkShell** application UI

Spark Jobs (?)

User: root
Total Uptime: 30 min
Scheduling Mode: FIFO
Completed Jobs: 1

▾ Event Timeline
☐ Enable zooming

Executors
Added
Removed
Executor driver added
Executor 0 added
Jobs
Succeeded
Failed
Running
collect at /root/wc.py:8 (Job
0 1 2 3 4 5 6 7 8 9 10
16 March 05:46

▾ Completed Jobs (1)

Job Id ▾	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
0	collect at /root/wc.py:8 collect at /root/wc.py:8	2019/03/16 05:46:08	2 s	2/2	4/4

4.8 OpenStack

- Began in 2010 as a joint project between Rackspace and NASA
- Offers free and open-source software platform for cloud computing for (mostly) IaaS
- Consists of interrelated components (services) that control / support compute, storage, and networking resources
- Often used through web-based dashboards, through command-line tools, or programmatically through ReSTful APIs
- Released under the terms of the Apache License
- Managed/coordinated by the OpenStack Foundation
  - non-profit corporate entity established in 2012 to promote OpenStack software and its community
  - Over 500 companies have since joined the project

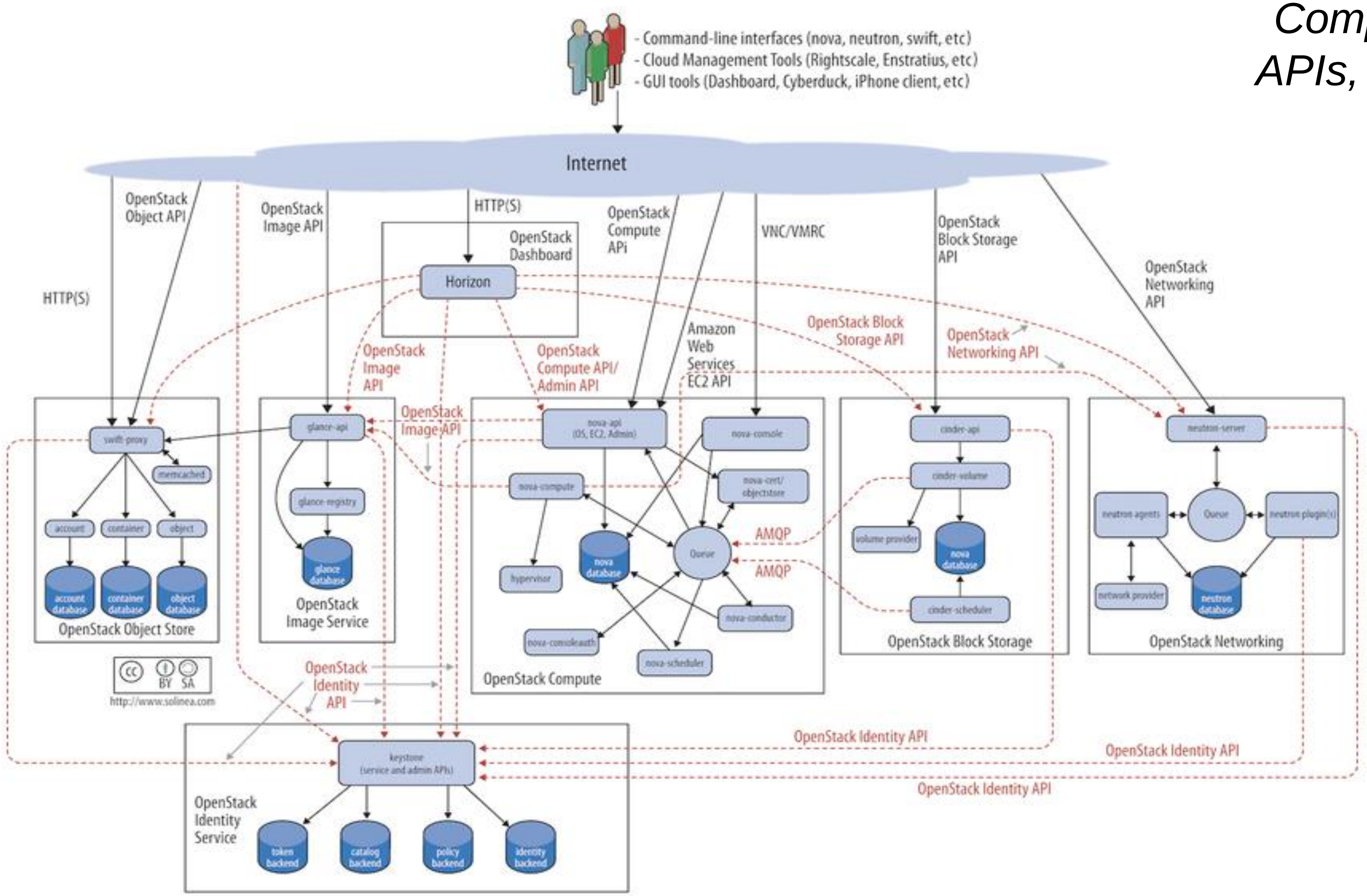
4.8 OpenStack Components

- Many associated/underpinning services
  - Compute Service (code-named Nova)
  - Image Service (code-named Glance)
  - Block Storage Service (code named Cinder)
  - Object Storage Service (code-named Swift)
  - Security Management (code-named Keystone)
  - Orchestration Service (code-named Heat)
  - Network Service (code-named Neutron)
  - Container Service (code-named Zun)
  - Database service (code-named Trove)
  - Dashboard service (code-named Horizon)
  - Search service (code-named Searchlight)
  - …

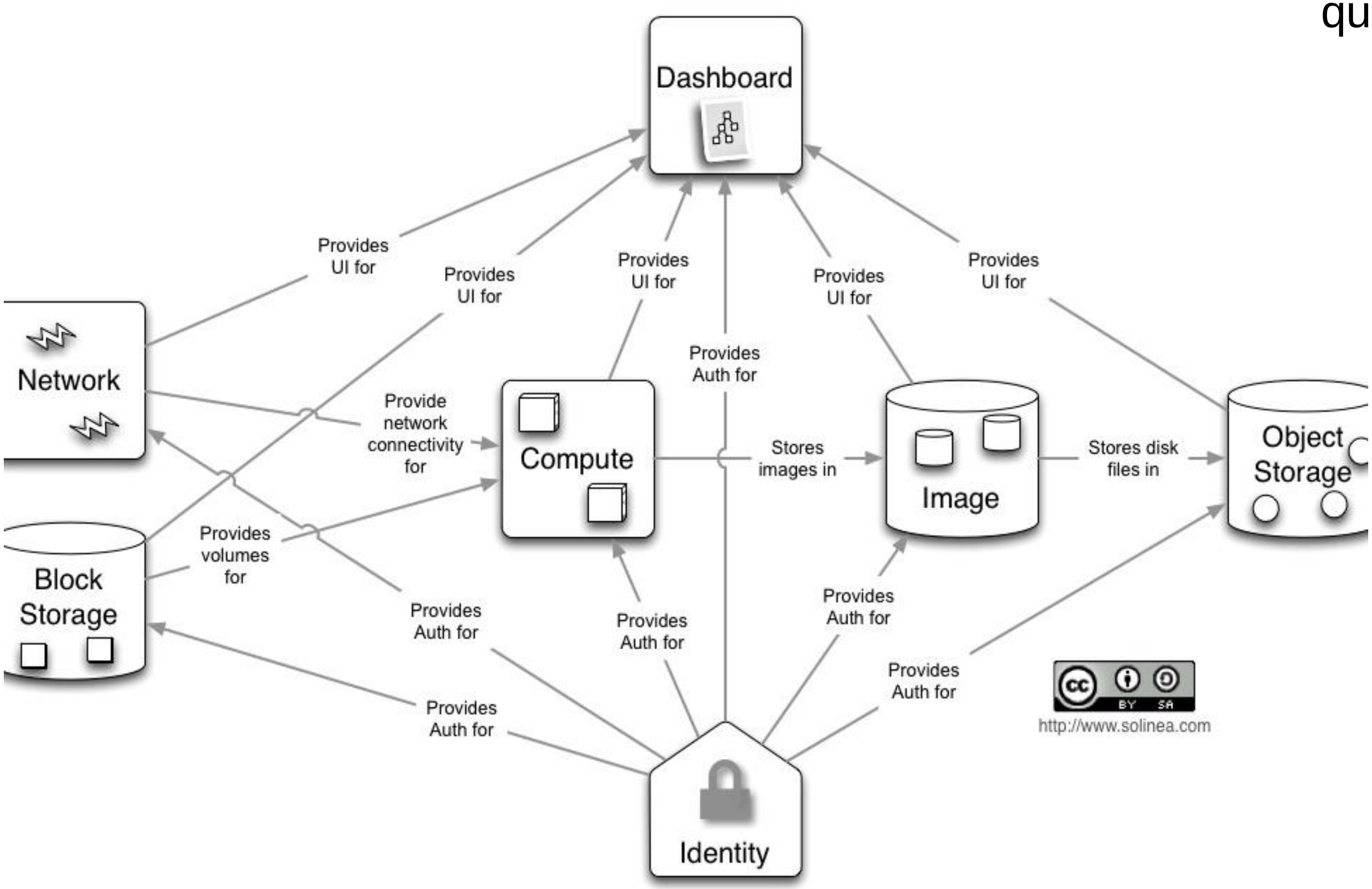
https://www.openstack.org/software/project-navigator/openstack-components#openstack-services

4.8 (Simplified) OpenStack Architecture

Components realised as APIs, Services, Daemons, Clients, …



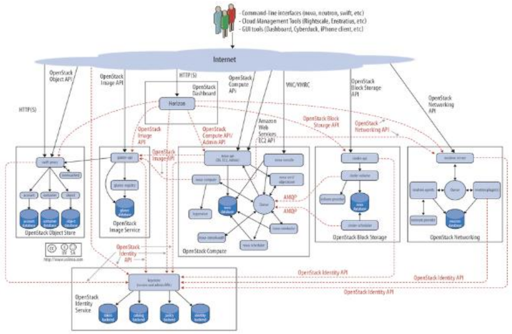
4.8 (Simplified) User Perspective



Typically asynchronous queuing systems used (AMQP)

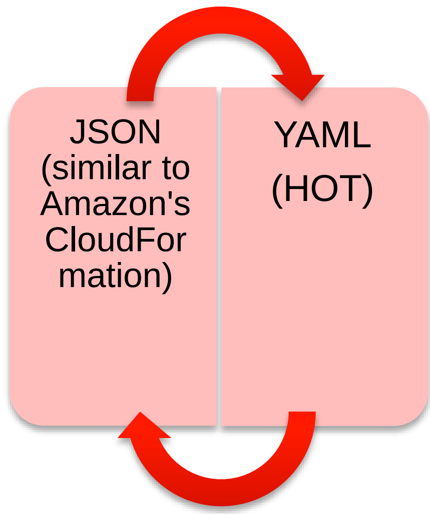
4.8 Key Services::Compute

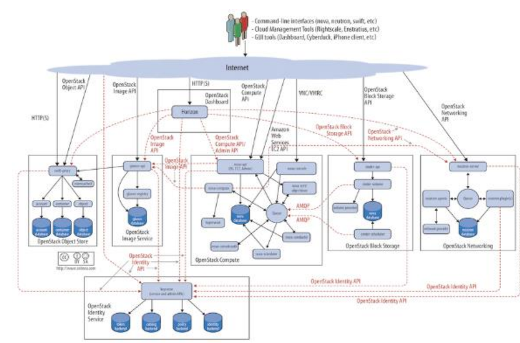
- Nova
  - Manages the lifecycle of compute instances in an OpenStack environment
  - Responsibilities include spawning, scheduling and decommissioning of virtual machines on demand
  - Virtualisation agnostic
    - *Libvirt*
      - open source API, daemon and tools for managing platform virtualisation including support for Kernel based virtual machine (KVM), Quick Emulator (QEMU), Xen, Lightweight Linux Container System (LXC)
    - *XenAPI, Hyper-V, VMWare ESX,*
    - *Docker (more later from Luca)*
    - *…*



4.8 Key Services::Orchestration Service

- Heat
  - Template-driven service to manage lifecycle of applications deployed on Openstack
  - *Stack*: Another name for the template and procedure behind creating infrastructure and the required resources from the template file
  - Can be integrated with automation tools such as Chef, Puppet, Ansible, etc.
  - Template format:





5 Summary

- Cloud computing can provide elastic Internet-based services
  - Host computing systems in an on-demand, scalable manner
  - For cost-down purpose
- Cloud computing systems are typically implemented as service-based systems: IaaS, PaaS, SaaS.
- Virtualization technology is key to implement cloud computing systems
- Hadoop Distributed File System handles failures systematically
- MapReduce is an emerging computing paradigm that can be implemented with cloud computing systems