# Big Data Analysis Platforms

SHYI-CHYI CHENG

# Outline

- Review of Virtual Machine (虛擬機器回顧)
- Hadoop Platform (運算分析系統架構)
- MapReduce
- Introduction to Python (Python入門簡介)
- Python Spark Platform (Python Spark運算分析架構)
- Parallel Programming With Spark

# Review of Virtual Machine

*This slide is from Intel® Corporation

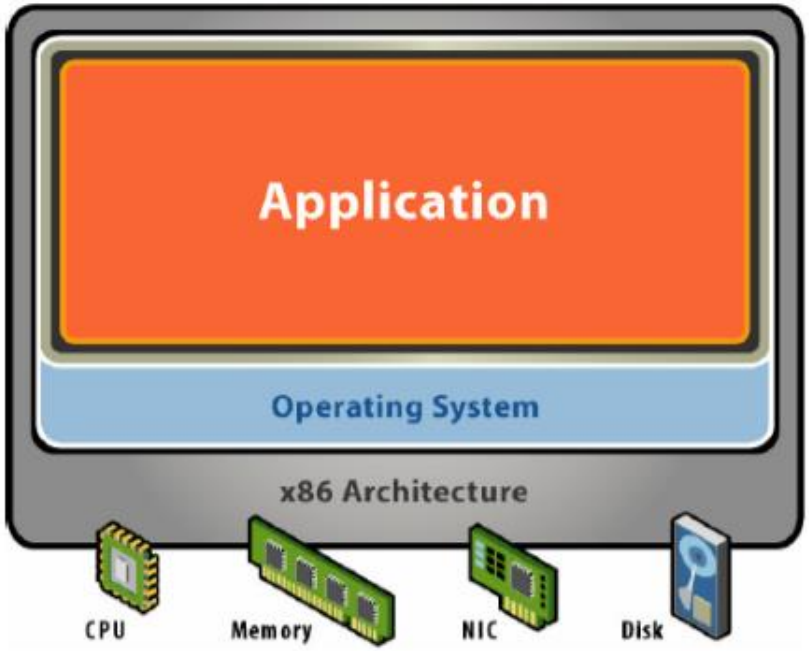
# Virtualization

- Virtualization deals with “extending or replacing an existing interface so as to mimic the behavior of another system”

- Virtual system examples: virtual private network, virtual memory, virtual machine

# Starting Point: A Physical Machine

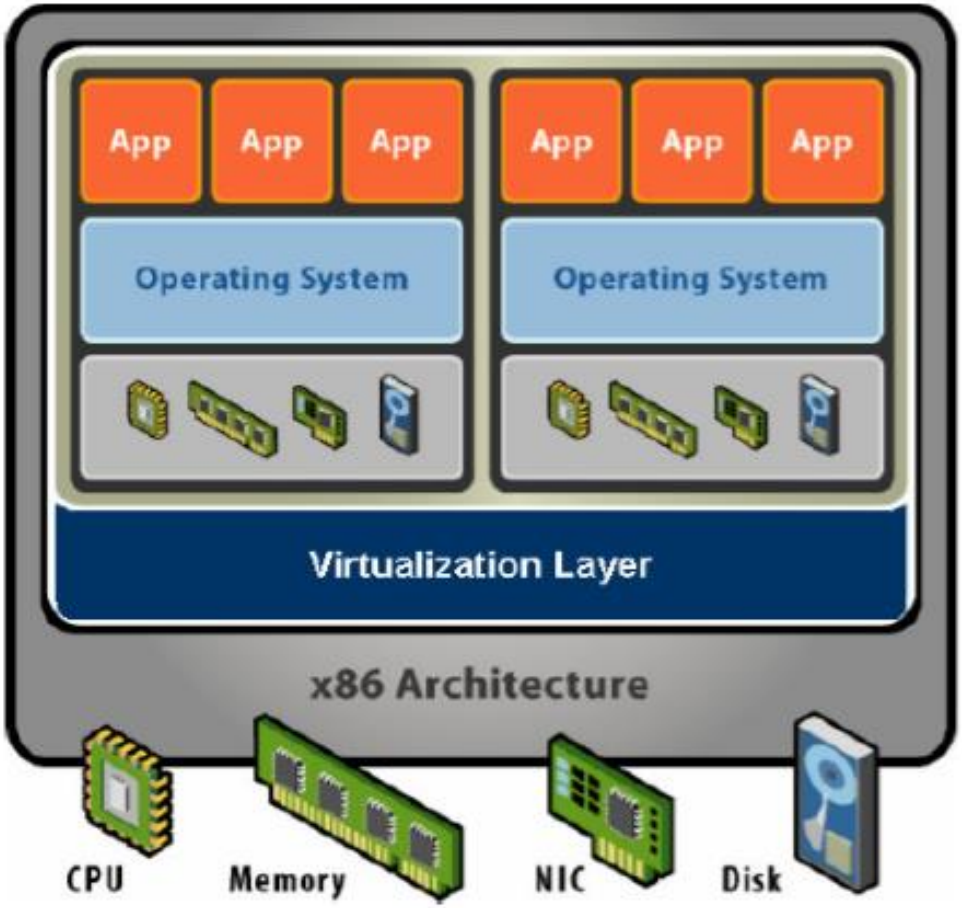
- Physical hardware
  - Processors, memory, chipset, I/O buses and devices, etc.
  - Physical resources often underutilized
- Software
  - Tightly coupled to hardware
  - Single active OS image
  - OS controls hardware



# What is a virtual machine?

- Hardware-level abstraction
  - Virtual hardware: processors, memory, chipset, I/O devices, etc.
  - Encapsulates all OS and application states
- Virtualization software
  - Extra level indirection decouples hardware and OS
  - Multiplexes physical hardware across multiple “guest” VMs
  - Strong isolation between VMs
  - Manage physical resources, improves utilization

# What is a virtual machine?

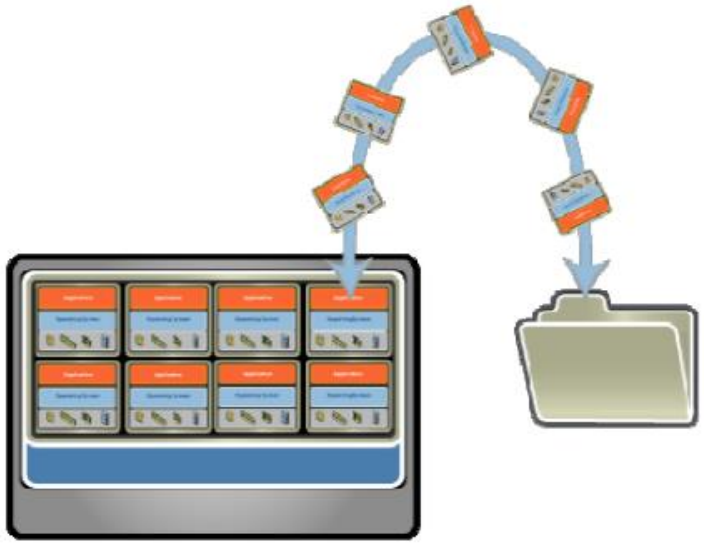


# VM Isolation

- Secure multiplexing
  - Run multiple VMs on a single physical host
  - Processor hardware isolates VMs, e.g., MMU
- Strong guarantees
  - Software bugs, crashes, viruses within one VM cannot affect other VMs
- Performance isolation
  - Partition system resources
  - Example: VMware controls for reservation, limits, and shares

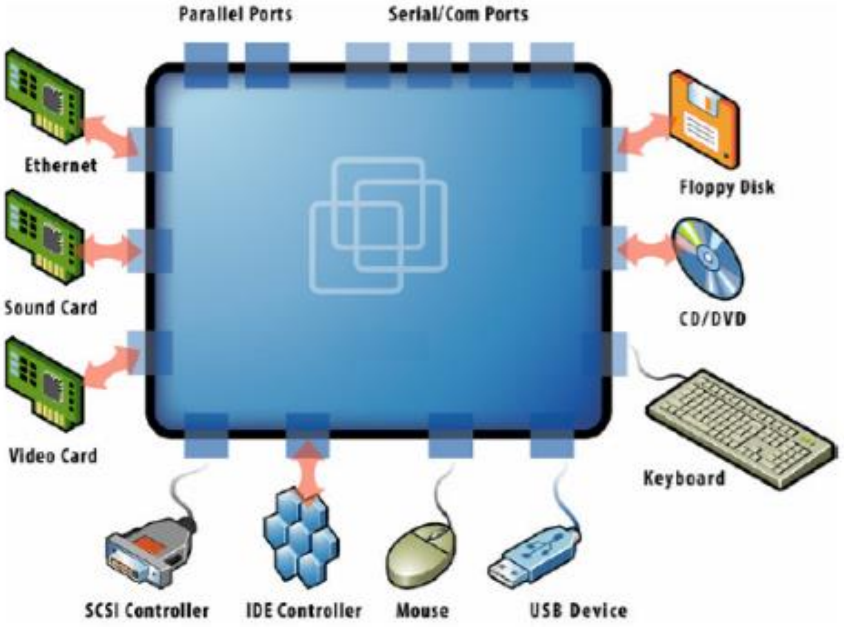
# VM Encapsulation

- Entire VM is a file
  - OS, applications, data
  - Memory and device state
- Snapshots and Clones
  - Capture VM state on the fly and restore to point-in-time
  - Rapid system provisioning, backup, remote mirroring
- Easy content distribution
  - Pre-configured apps, demos
  - Virtual appliances



# VM Compatibility

- Hardware-Independence
  - Physical hardware hidden by virtualization layer
  - Standard virtual hardware exposed to VM
- Create Once, Run Anywhere
  - No configuration issues
  - Migrate VMs between hosts
- Legacy VMs
  - Run ancient OS on new platform
  - E.g., DOS VM drivers virtual IDE and vLance devices, mapped to modern SAN and GigE hardware



# Common Virtualization Uses Today

- Test and development
  - Rapidly provision test and development servers

- Business Continuity
  - Reduce cost and complexity by encapsulating entire systems into single files that can be replicated and restored onto any target server

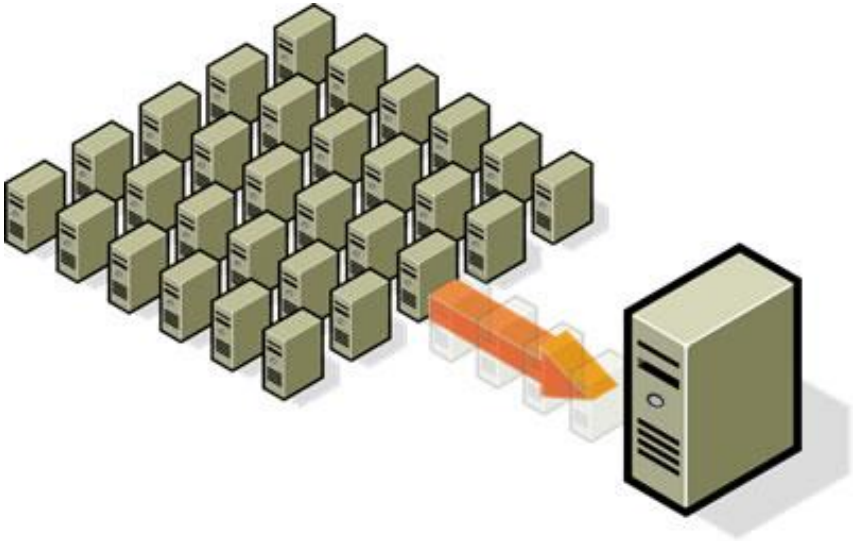
- Enterprise Desktop
  - Secured unmanaged PCs without compromising end-user autonomy by layering a security policy in software around desktop VMs

# Common Virtualization Uses Today

- Run legacy software on non-legacy hardware
- Run multiple operating systems on the same hardware
- Create a manageable upgrade path
- Manage outages (expected and unexpected) dynamically

# Common Virtualization Uses Today

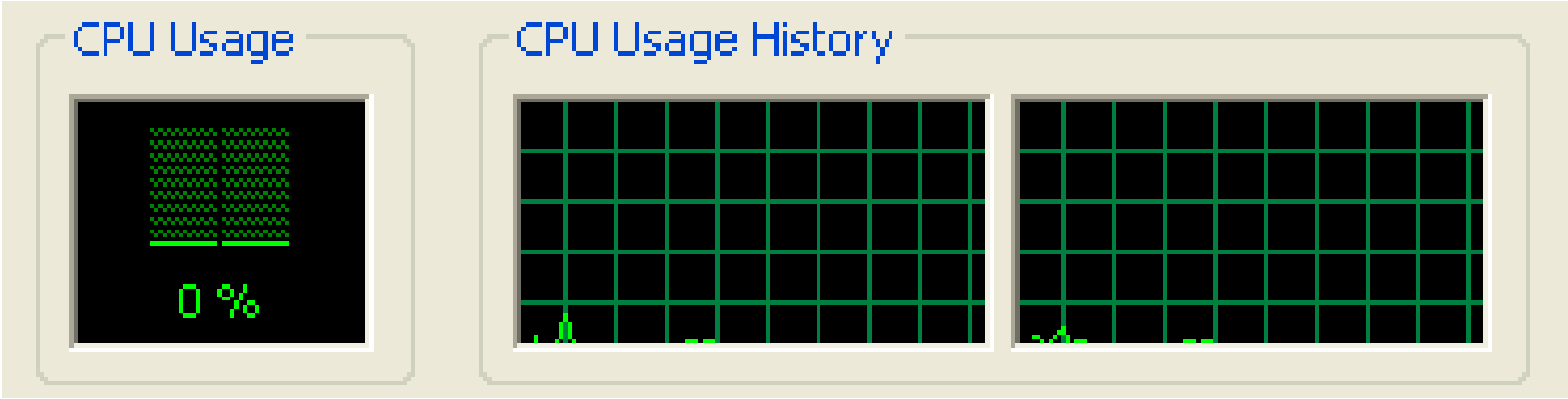
- Reduce costs by consolidating services onto the fewest number of physical machines



http://www.vmware.com/img/serverconsolidation.jpg

# Non-virtualized Data Centers

- Too many servers for too little work



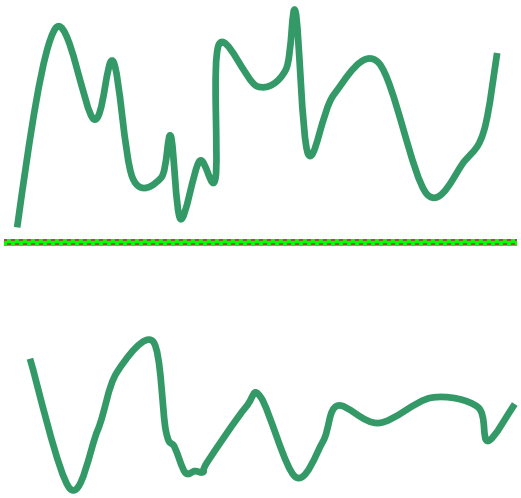

- High costs and infrastructure needs
  - Maintenance
  - Networking
  - Floor space
  - Cooling
  - Power
  - Disaster Recovery

# Dynamic Data Center

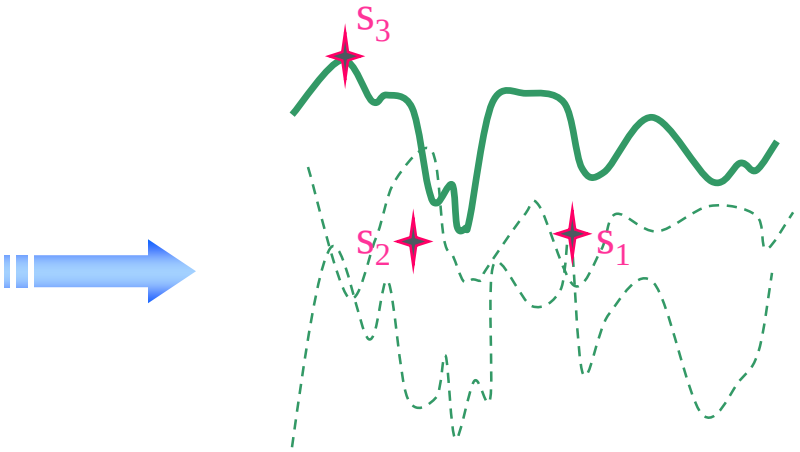
- Virtualization helps us break the "one service per server" model
- Consolidate many services into a fewer number of machines when workload is low, reducing costs
- Conversely, as demand for a particular service increases, we can shift more virtual machines to run that service
- We can build a data center with fewer total resources, since resources are used as needed instead of being dedicated to single services

# VM workload multiplexing

Separate VM sizing



VM multiplexing




***We expect $s_3 < s_1 + s_2$. Benefit of multiplexing !***

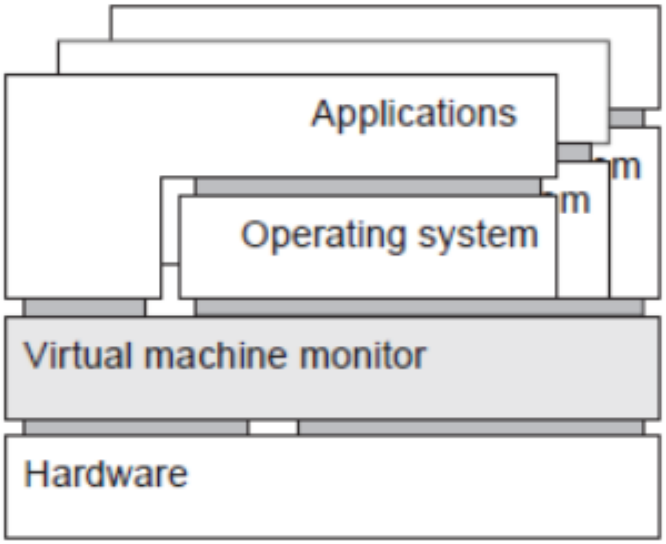
- Multiplex VMs' workload on same physical server
  - Aggregate multiple workload. Estimate total capacity need based on aggregated workload
  - Performance level of each VM be preserved

# What is a Virtual Machine Monitor

A virtual machine is taken to be an *efficient, isolated duplicate* of the real machine. We explain these notions through the idea of a *virtual machine monitor* (VMM). See Figure 1. As a piece of software a VMM has three essential characteristics. First, the VMM provides an environment for programs which is essentially identical with the original machine; second, programs run in this environment show at worst only minor decreases in speed; and last, the VMM is in complete control of system resources.

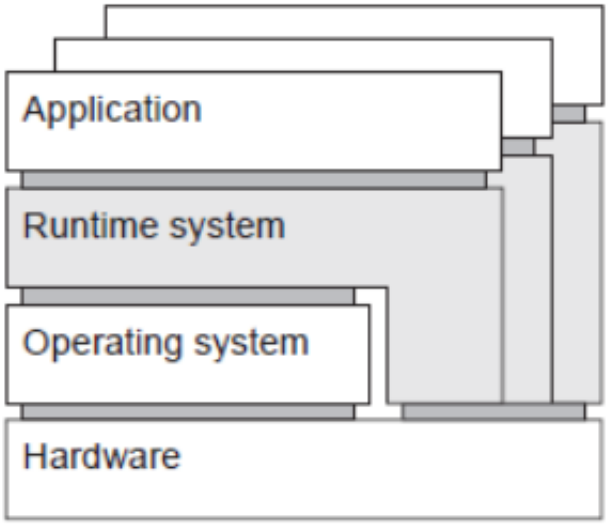
# VM Monitor

- A separate software layer mimics the instruction set of hardware: a complete OS + applications
  - Ex. VMware, VirtualBox



# Process VM

- A program is compiled to intermediate (portable) code, which is then executed by a runtime system
  - Ex. Java VM

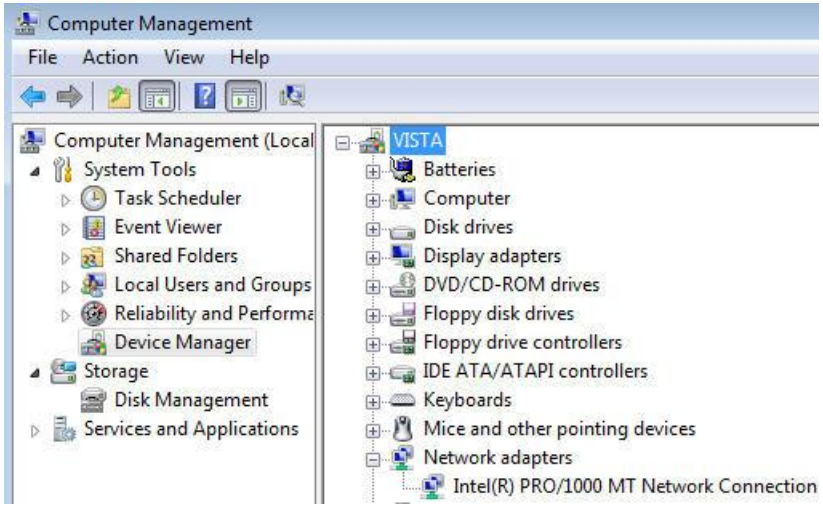


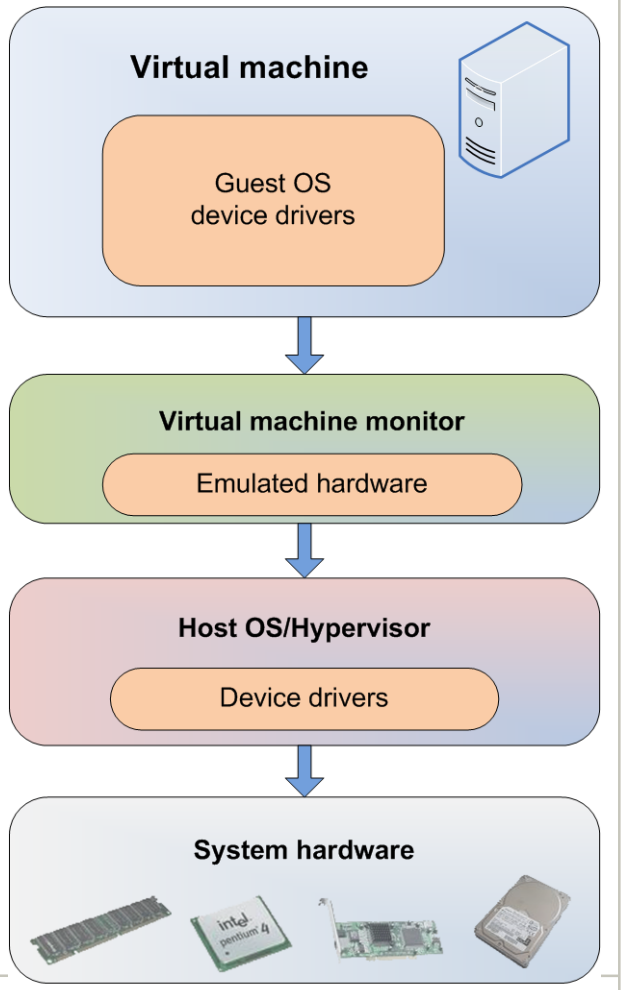
# Thee Virtualization Approaches

- Full Virtualization
- Paravirtualization
- Hardware-assisted Virtualization

# Full Virtualization

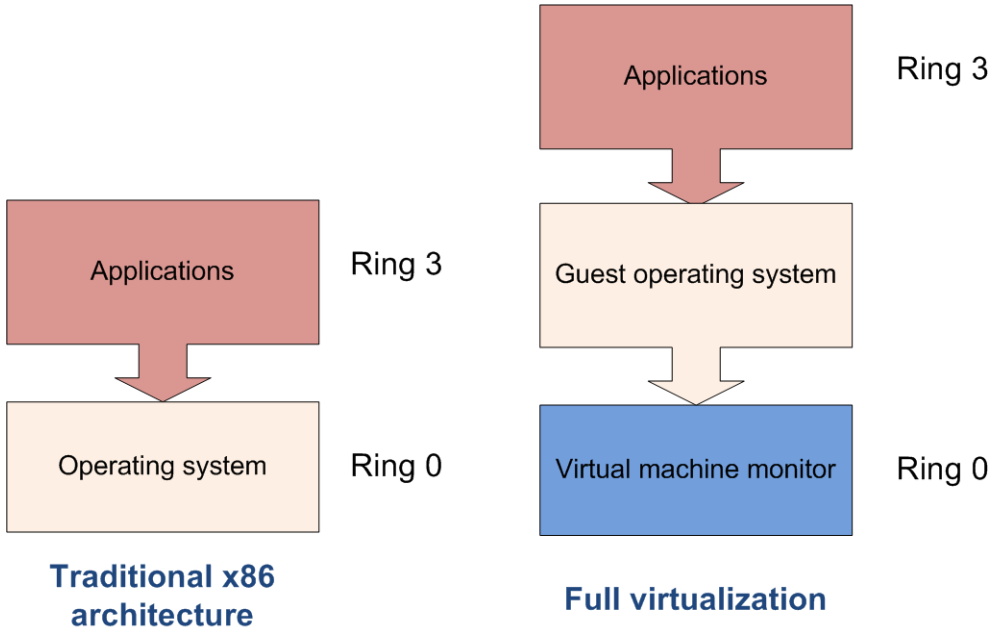
- Everything is virtualized
- Full hardware emulation
- Emulation = latency



# Privileged Instructions

- Privileged instructions: OS kernel and device driver access to system hardware
- Trapped and emulated by VMM



# Pros and Cons – Full Virtualization

- **Pros**
  - Disaster recovery, failover
  - Virtual appliance deployment
  - Legacy code on non-legacy hardware
- **Cons** – LATENCY of core four resources
  - RAM performance reduced 25% to 75%
  - Disk I/O degraded from 5% to 20%
  - Network performance decreased up to 10%
  - CPU privileged instruction dings nearing 1% to 7%

# Paravirtualization

- OS or system devices are virtualization aware

**Requirements:**

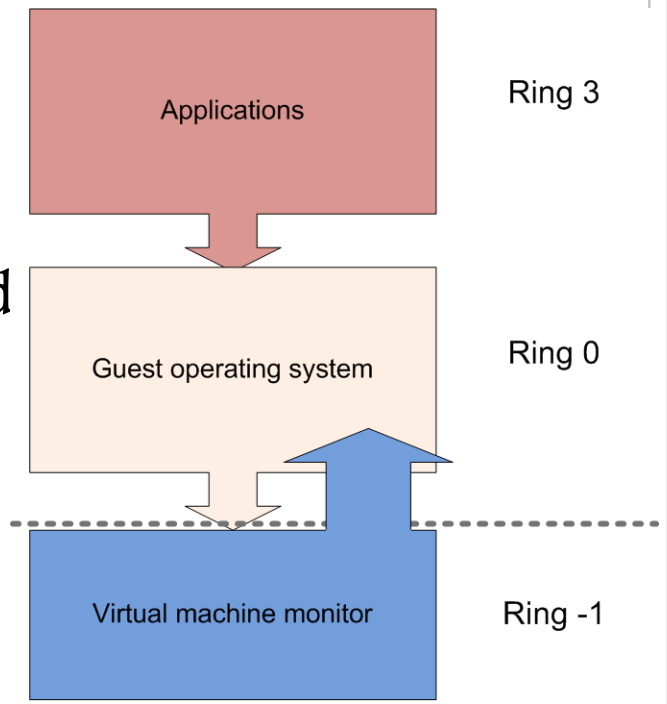
- OS level – recompiled kernel
- Device level – paravirtualized or "enlightened" device drivers

# Paravirtualization

- **Pro:** fast
- **Con:** requires a specially modified guest OS, thus precludes the ability to run off-the-shelf and legacy OS in paravirtual environments

# Hardware-assisted Virtualization

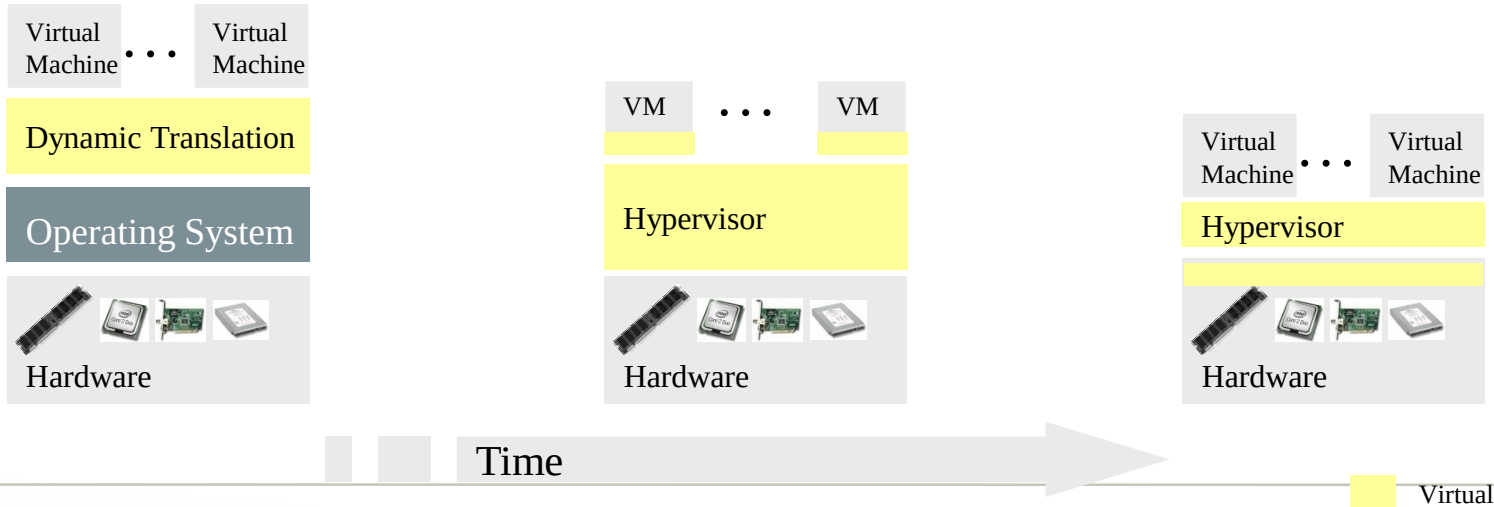
- Server hardware is virtualization aware
- Hypervisor and VMM load at privilege Ring -1 (firmware)
- Removes CPU emulation bottleneck
- Memory virtualization coming in quad core AMD and Intel CPUs




Hardware-assisted virtualization

# Evolution of Software solutions

- 1st Generation: Full virtualization (Binary rewriting)
  - Software Based
  - VMware and Microsoft
- 2nd Generation: Paravirtualization
  - Cooperative virtualization
  - Modified guest
  - VMware, Xen
- 3rd Generation: Silicon-based (Hardware-assisted) virtualization
  - Unmodified guest
  - VMware and Xen on virtualization-aware hardware platforms

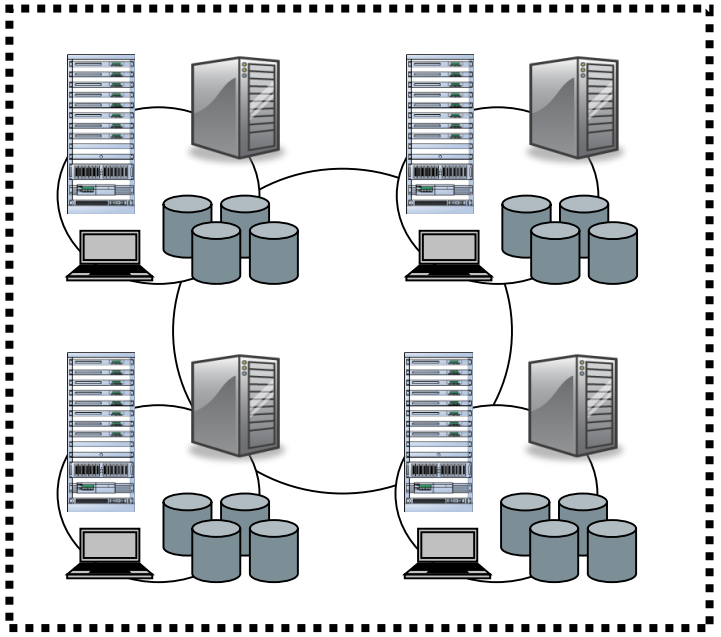


# Issues in VMs for Cloud Computing

- Aspects and expectation from
  - End-user
  - Operator/Manager

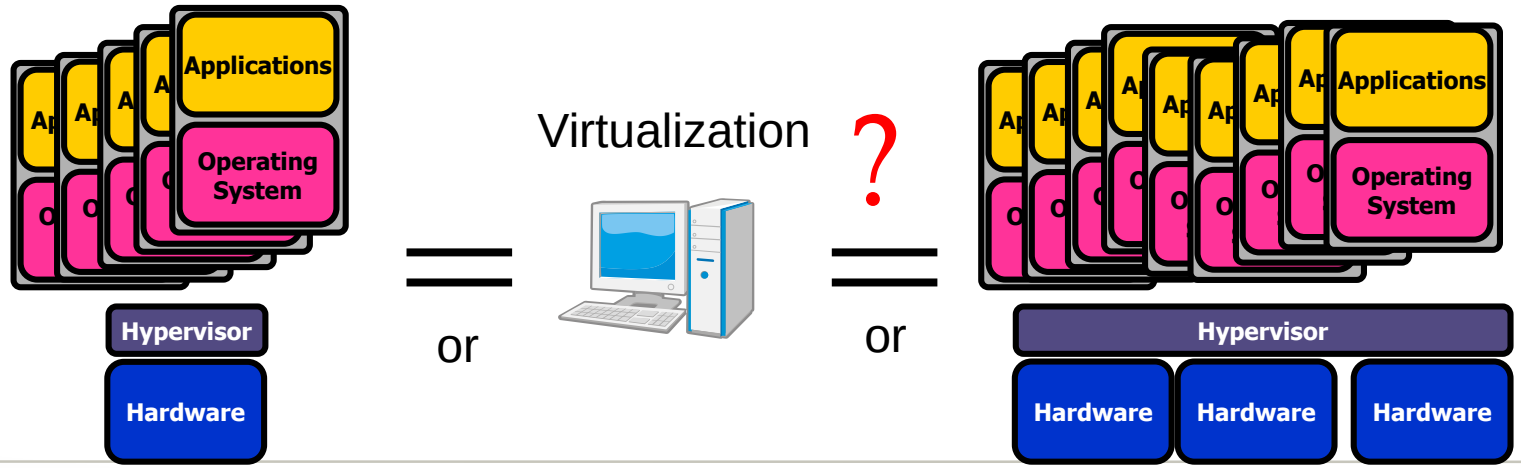
Virtualization

=



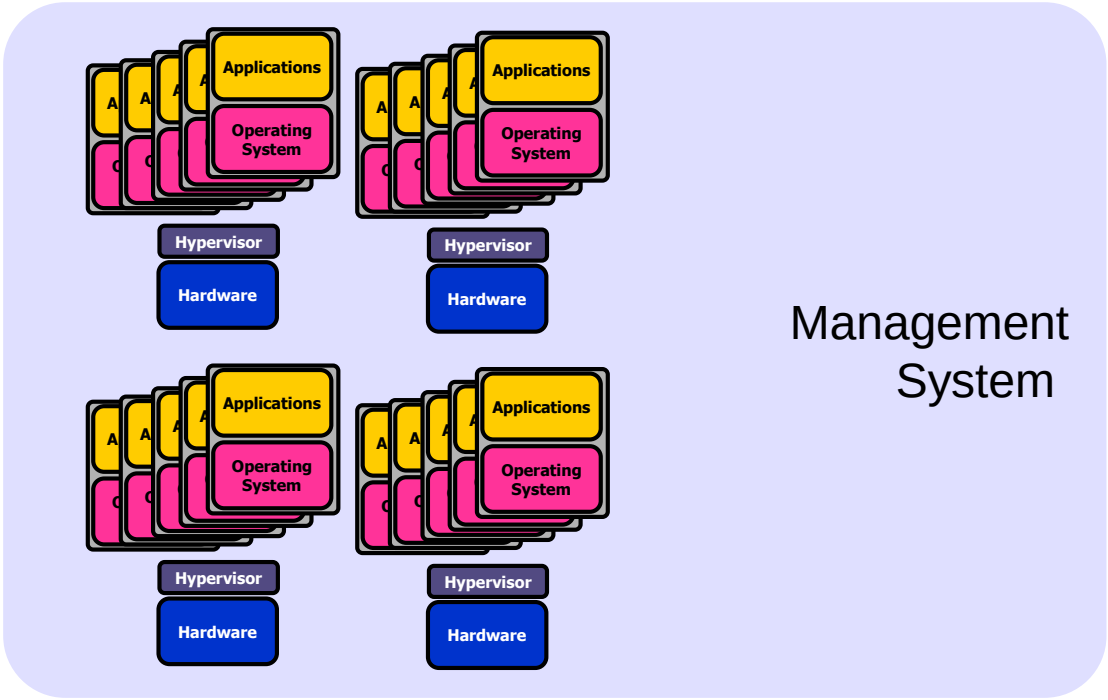
# Issues in VMs for Cloud Computing

- Virtualization implemented on
  - a single machine (with multi-core CPUs)
  - a cluster of machines (with multi-core CPUs)
- The state-of-the-art
  - Running a Xen or a cluster of Xens



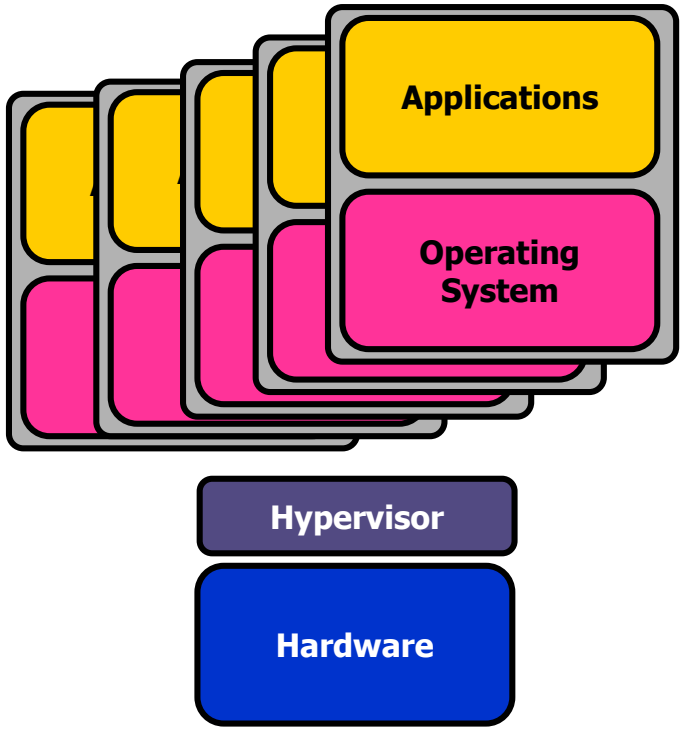
# Issues in Virtualization for Cloud-Computing

- Abiquo/abicloud may provide partial solutions



# Running multiple OS and applications

- Virtualization: One physical hardware can run multiple OS and applications through a hypervisor.
- A hypervisor is the virtualization manager on a physical hardware.



# Popular hypervisors

- Xen
- KVM
- QEMU
- virtualBox
- VMWare

# Homework 1

- 安裝 Virtual Box 虛擬機器軟體
- 在虛擬機器軟體上安裝 Ubuntu Linux 作業系統
- 練習操作Virtual Box及Ubuntu
- 撰寫並繳交實習報告(Hand in the report on next week)

# 安裝VirtualBox與Ubuntu

- **VirtualBox下載安裝**
  - 下載網址:https://www.virtualbox.org/wiki/Downloads
  - 請研讀安裝手冊並進行安裝
- **Ubuntu Linux的作業系統安裝**
  - 下載網址:http://www.ubuntu-tw.org/modules/tinyd0/
  - 請研讀安裝手冊並進行安裝

# Outline

- Review of Virtual Machine (虛擬機器回顧)
- Hadoop Platform (運算分析系統架構)
- MapReduce
- Introduction to Python (Python入門簡介)
- Python Spark Platform (Python Spark運算分析架構)
- Python Spark Mllib (Python Spark機器學習程式庫)

# Hadoop Platform

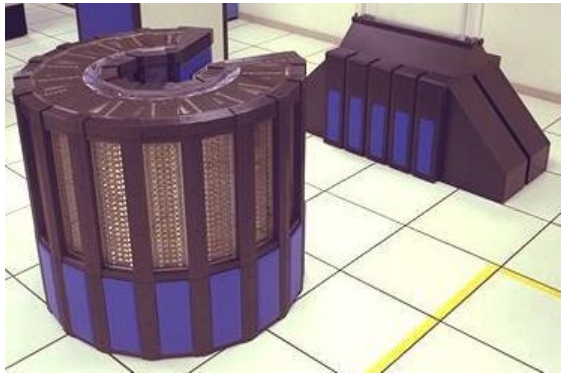
A Software Framework for Data Intensive Computing Applications

# Hadoop - Why ?

- Need to process huge datasets on large clusters of computers
- Very expensive to build reliability into each application
- Nodes fail every day
  - Failure is expected, rather than exceptional
  - The number of nodes in a cluster is not constant
- Need a common infrastructure
  - Efficient, reliable, easy to use
  - Open Source, Apache License

# Traditional Large-Scale Computing

- Traditionally, computation has been processor-bound
  - Relatively small amounts of data
  - Lots of complex processing
- The early solution: super computers
  - Faster processor, more memory
  - Fail in processing big data

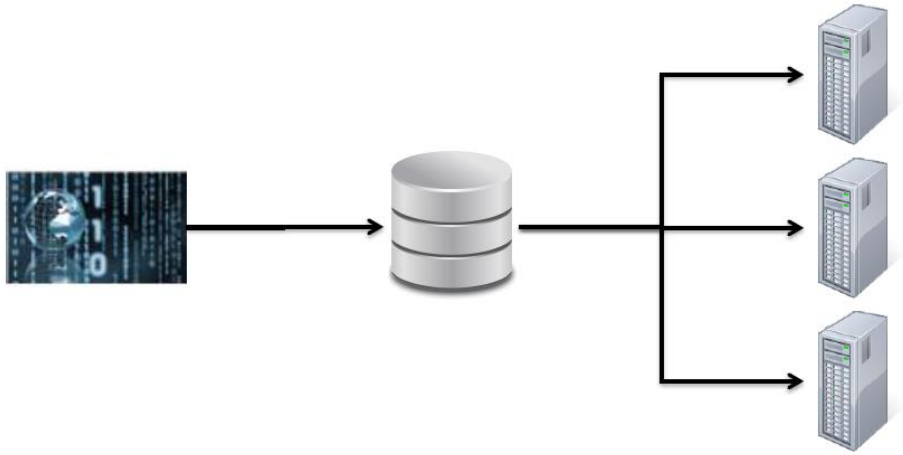


# Distributed Systems

- The better solution: more computers
  - Distributed systems use multiple machines for a single job

# Distributed Systems: The Data Bottleneck (1)

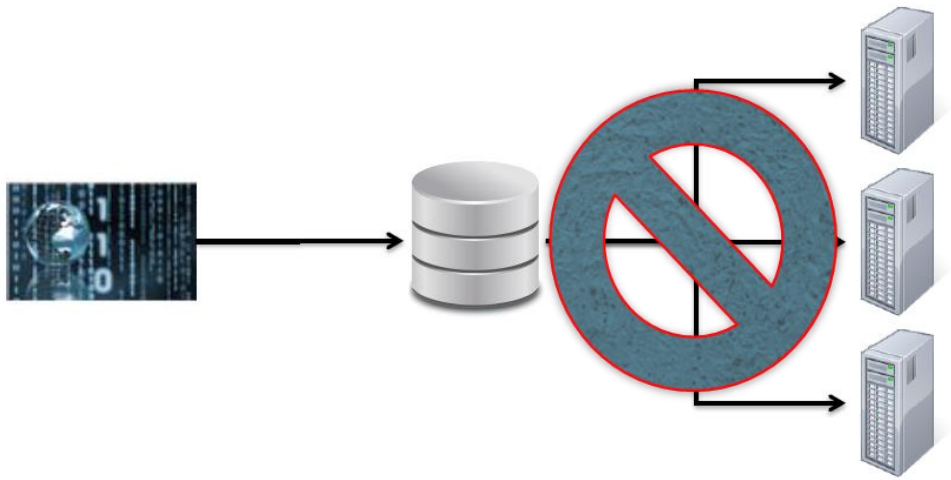
- Data is stored in a central location →Hot spot
- Data is copied to processors at runtime
- Fine for limited amount of data



# Big Data Analytics Needs New Solutions

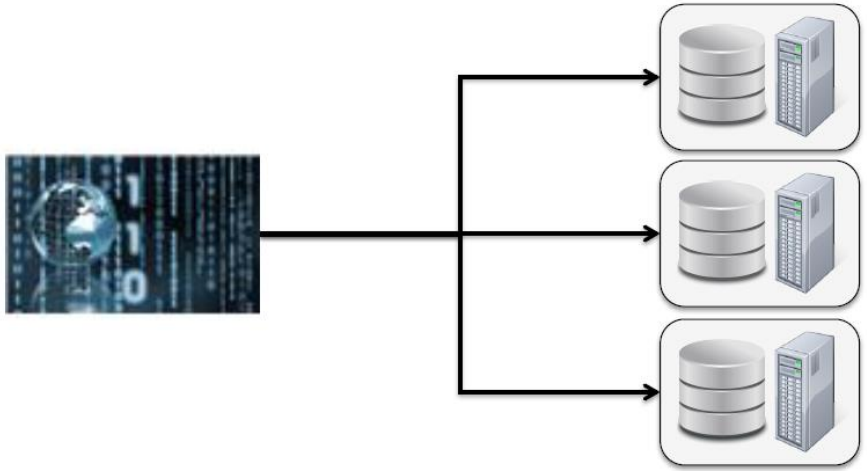
- Internet of Things (IoT) bring big amounts of data to be processed
  - terabytes a day
  - petabytes a weak
  - zetabytes in total
- New approach

required



# Hapdoop

- **A radical new approach to distributed computing**
  - Distribute data when the data is stored
  - Run computation where the data is
- **Originally based on the work at Google**
- **Open source project overseen by the Apache Software Foundation**



# Core Hapdoop Concepts

- Applications are written in high-level code
- Node talk to each other as little as possible
- Data is distributed in advance
  - Bring the computation to the data
- Data is replicated for increased availability and reliability
- Hapdoop is scalable and fault-tolerant

# Scalability

- Adding nodes adds capacity proportionally
- Increasing load results in a graceful decline in performance
  - Not failure of the system

# Fault Tolerance

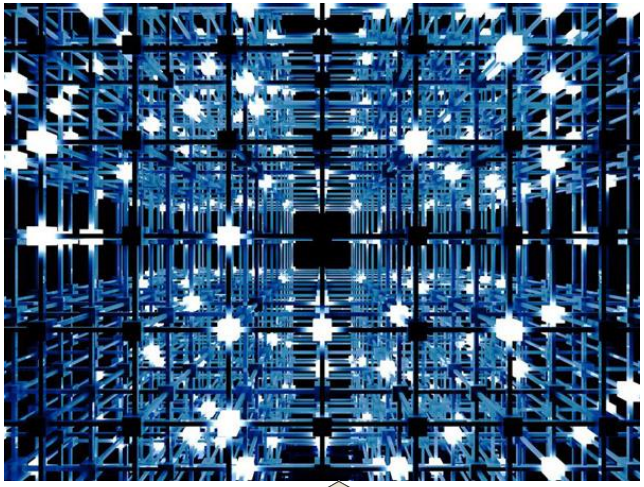
- Node failure is inevitable
- What happens?
  - System continues to function
  - Master re-assigns tasks to a different node
  - Data replication → no loss of data
  - Nodes which recover and rejoin the cluster automatically

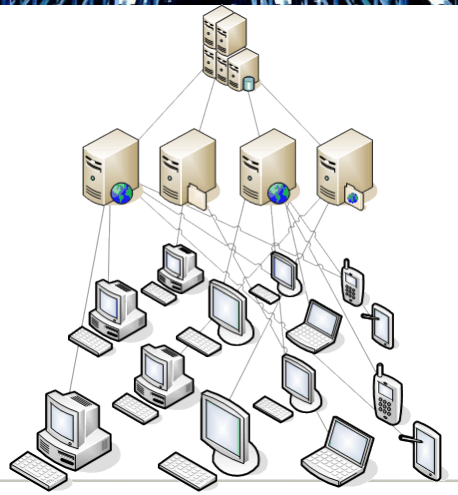
# Hadoop-able Problems

- Text mining
- Index building
- Graph creation and analysis
- Machine learning
- Collaborative filtering
- Prediction models
- Sentiment analysis
- Risk assessment


# What is Common Across Hadoop-able Problems?

- Nature of the data
  - Volume
  - Velocity
  - Variety
- Nature of the analysis
  - Batch processing
  - Parallel execution
  - Distributed data





# Advantages of Analyzing with Hapdoop

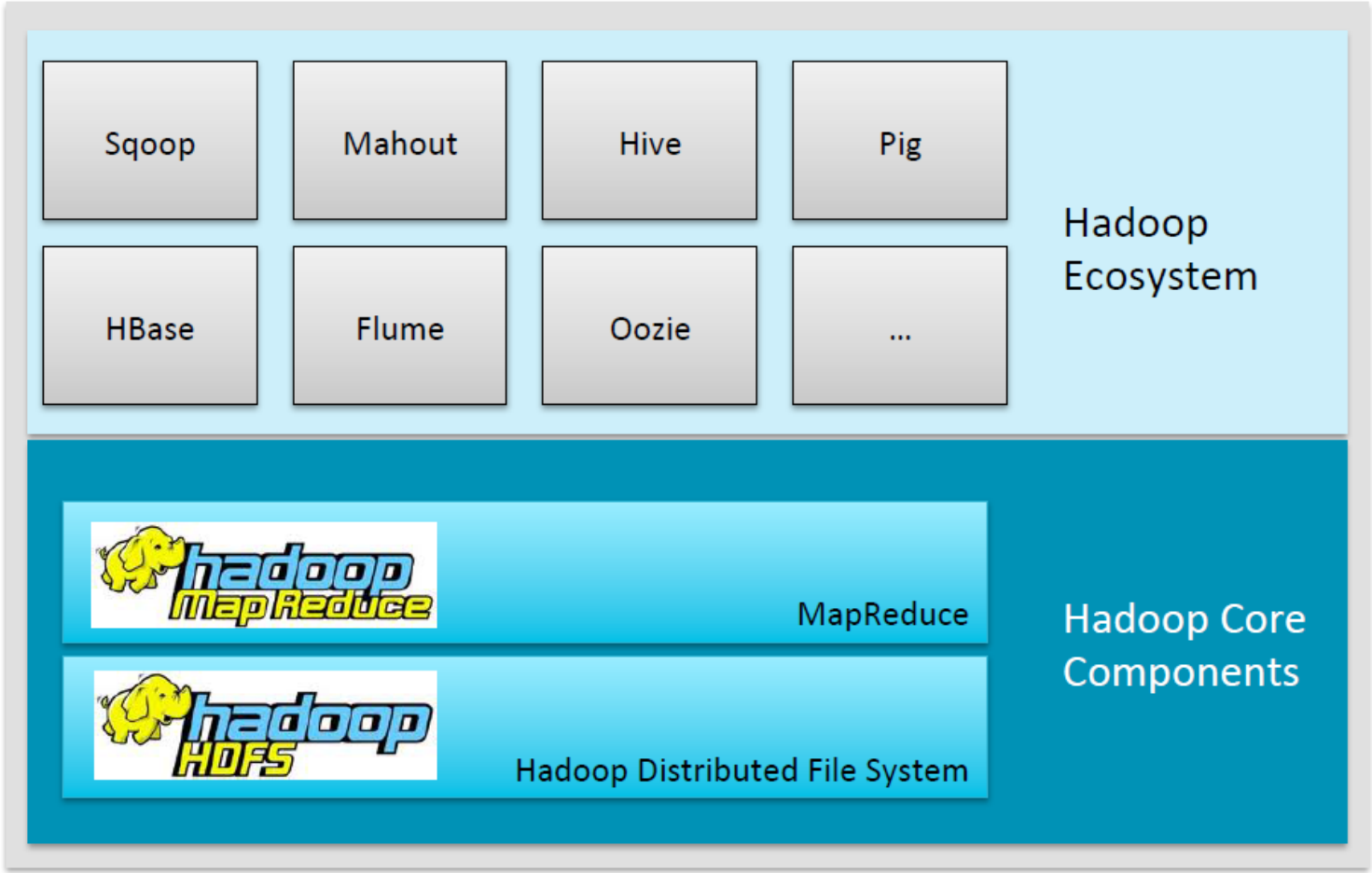
- Previous impossible or impractical analysis
- Lower lost
- Less time
- Greater flexibility
- Near-linear scalability
- Ask Bigger Questions

# Who uses Hadoop?

- Facebook
- Google
- New York Times
- Veoh
- Yahoo!
- …. many more

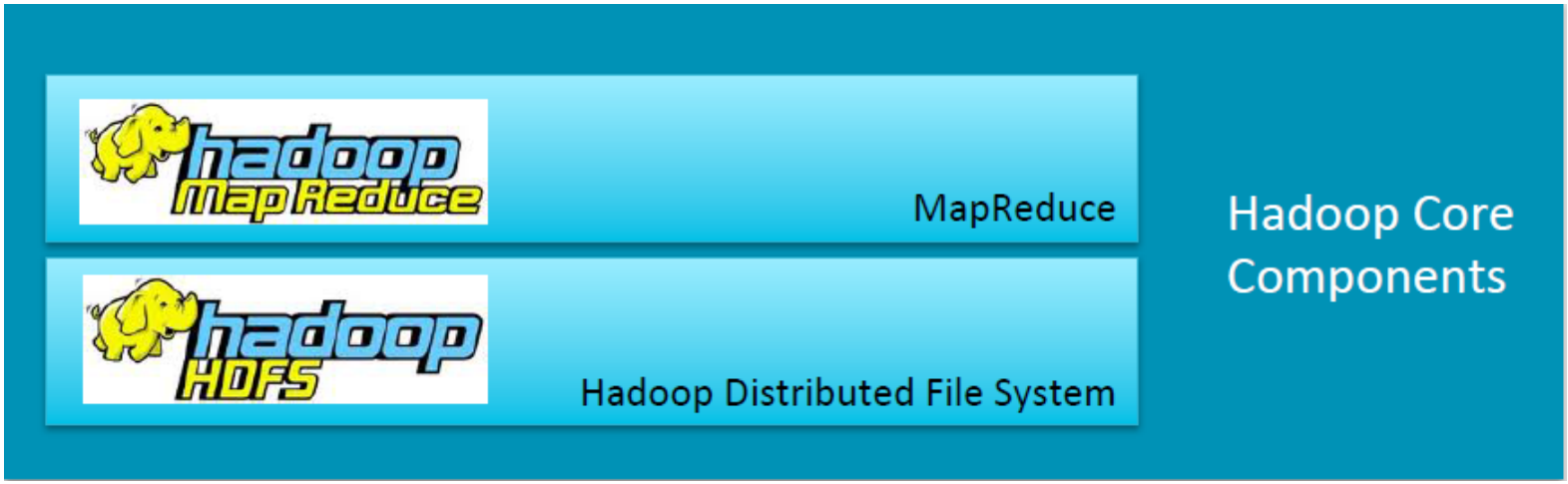
# Hadoop Basic Concepts and HDFS

# Software Architecture of Apache Hadoop

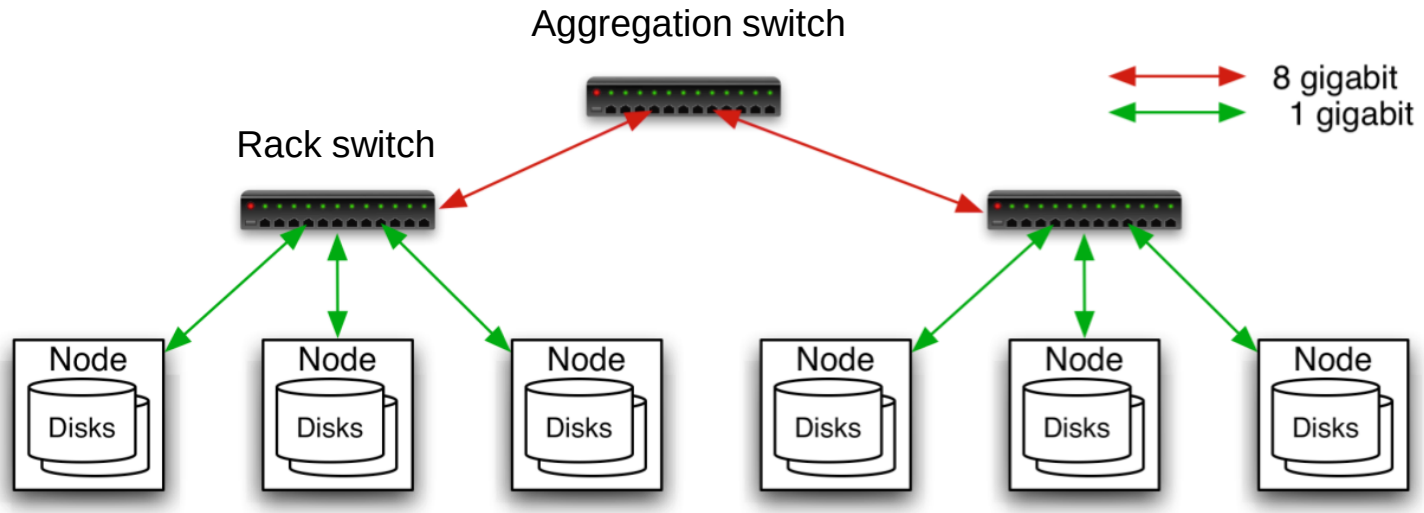


# Core Components: HDFS and MapReduce

- HDFS (Hapdoop Distributed File System)
  - Stores data on the cluster
- MapReduce
  - Process data on the cluster



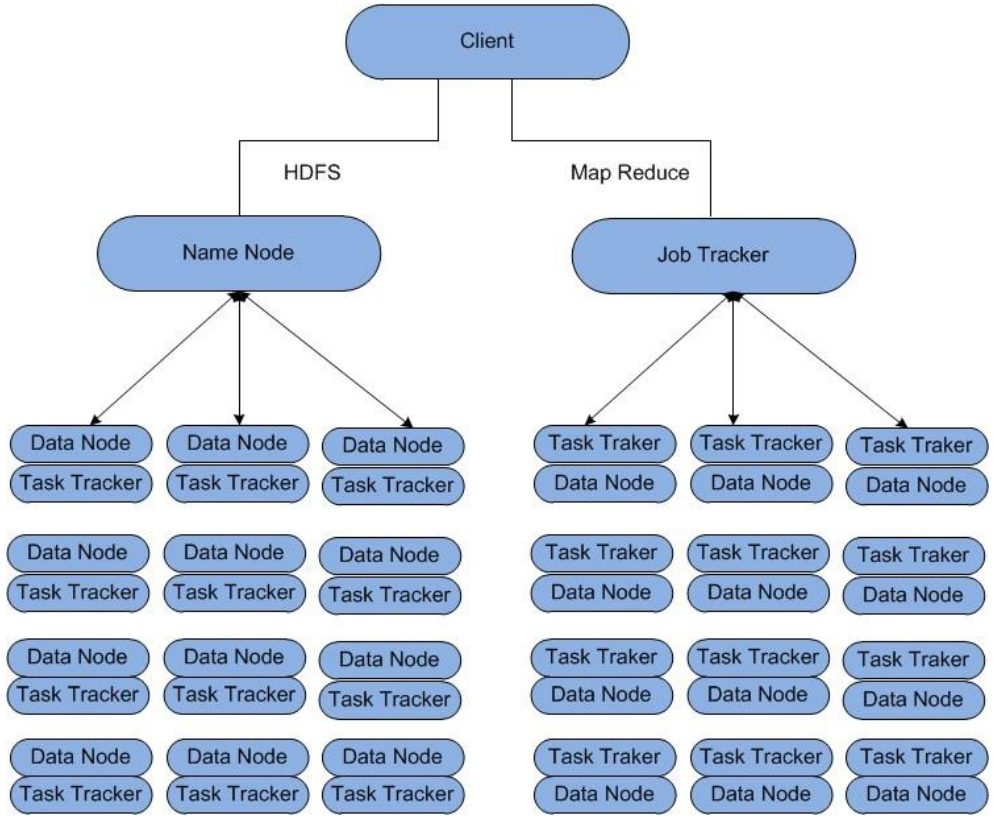
# Commodity Hardware




- Typically in 2 level architecture
  - Nodes are commodity PCs
  - 30-40 nodes/rack
  - Uplink from rack is 3-4 gigabit
  - Rack-internal is 1 gigabit

# A Simple Hapdoop Cluster

- Node types
  - Slave or worker nodes
    - HDFS to store data
    - MapReduce to process data
  - Master nodes
    - Name node to manage HDFS
    - Job Tracker to manage MapReduce



# Hadoop Distributed File System (HDFS)

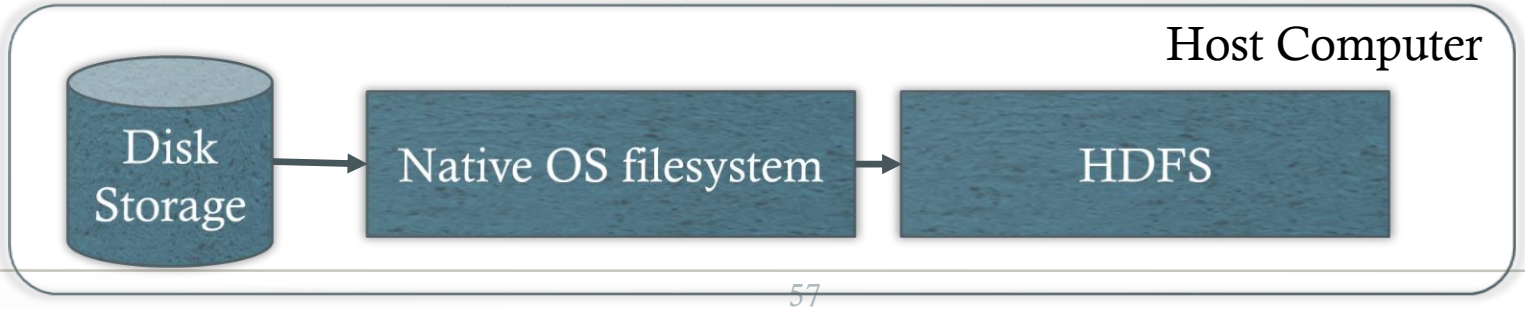
Original Slides by
Dhruba Borthakur
Apache Hadoop Project Management Committee

# Goals of HDFS

- Very Large Distributed File System
  - 10K nodes, 100 million files, 10PB
- Assumes Commodity Hardware
  - Files are replicated to handle hardware failure
  - Detect failures and recover from them
- Optimized for Batch Processing
  - Data locations exposed so that computations can move to where data resides
  - Provides very high aggregate bandwidth

# HDFS Basic Concepts (1)

- HDFS is a filesystem written in Java
  - Originally proposed by Google's GFS
- Sits on top of a native file system
  - Such as Linux's filesystems: etx3, etx4 or xfs
- Provides redundant storage for massive amounts of data
  - Using readily-available, industry-standard computers

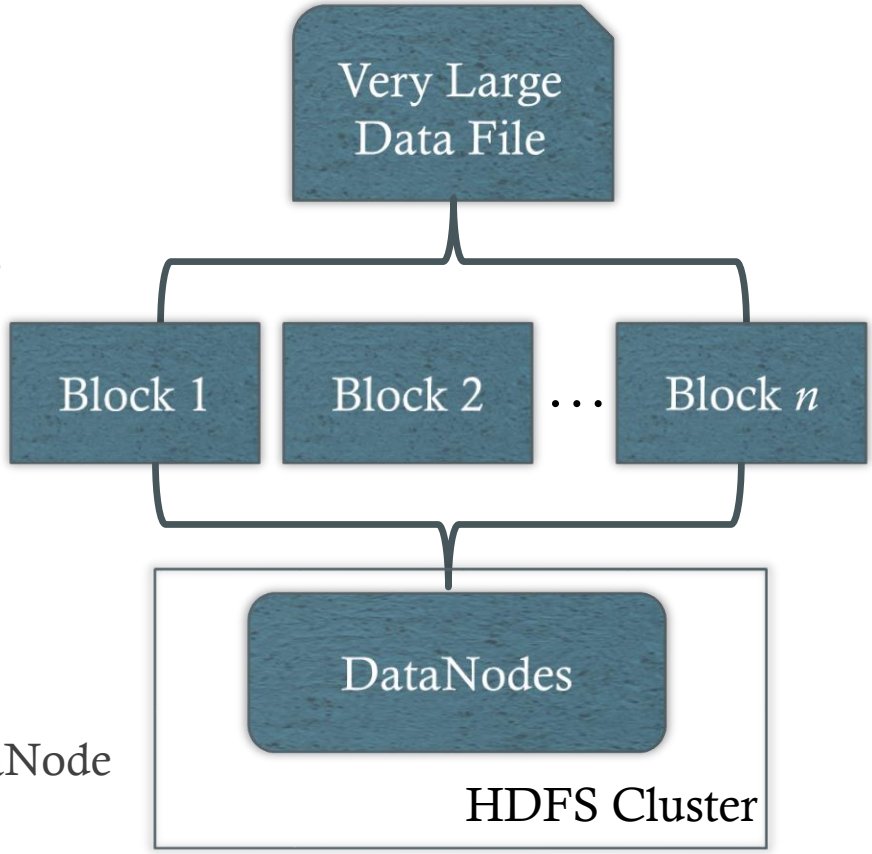


# HDFS Basic Concepts (2)

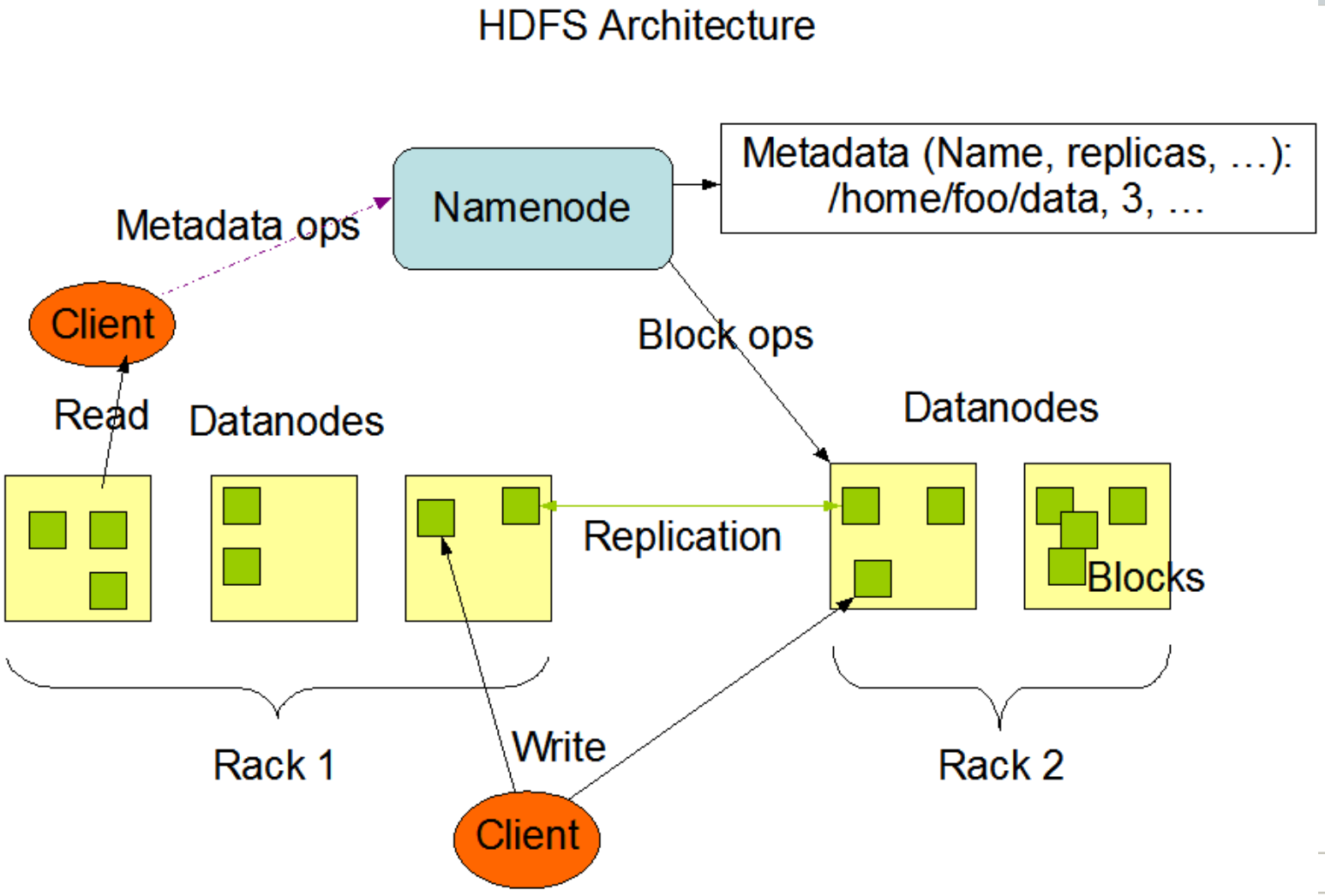
- HDFS performs best with a 'modest' number of large files
  - Millions, rather than billions, of files
  - Each file typically 100MB or more
- Files in HDFS are 'write once'
  - No random writes to files are allowed
- HDFS is optimized for large, streaming reads of files
  - Rather than random reads

# HDFS Basic Concepts (3)

- Single Namespace for entire cluster
- Data Coherency
  - Write-once-read-many access model
  - Client can only append to existing files
- Files are broken up into blocks
  - Typically 64MB block size
  - Each block replicated on multiple DataNodes (Default x3)
- Intelligent Client
  - Client can find location of blocks
  - Client accesses data directly from DataNode



# HDFS Architecture



# Functions of a NameNode

- Manages File System Namespace
  - Maps a file name to a set of blocks
  - Maps a block to the DataNodes where it resides
- Cluster Configuration Management
- Replication Engine for Blocks

# NameNode Metadata

- Metadata in Memory
  - The entire metadata is in main memory
  - No demand paging of metadata
- Types of metadata
  - List of files
  - List of Blocks for each file
  - List of DataNodes for each block
  - File attributes, e.g. creation time, replication factor
- A Transaction Log
  - Records file creations, file deletions etc

# DataNode

- A Block Server
  - Stores data in the local file system (e.g. ext3)
  - Stores metadata of a block (e.g. CRC)
  - Serves data and metadata to Clients
- Block Report
  - Periodically sends a report of all existing blocks to the NameNode
- Facilitates Pipelining of Data
  - Forwards data to other specified DataNodes

# Block Placement

- Current Strategy
  - One replica on local node
  - Second replica on a remote rack
  - Third replica on same remote rack
  - Additional replicas are randomly placed
- Clients read from nearest replicas
- Would like to make this policy pluggable

# Heartbeats

- DataNodes send hearbeat to the NameNode
  - Once every 3 seconds
- NameNode uses heartbeats to detect DataNode failure

# Replication Engine

- NameNode detects DataNode failures
  - Chooses new DataNodes for new replicas
  - Balances disk usage
  - Balances communication traffic to DataNodes

# Data Correctness

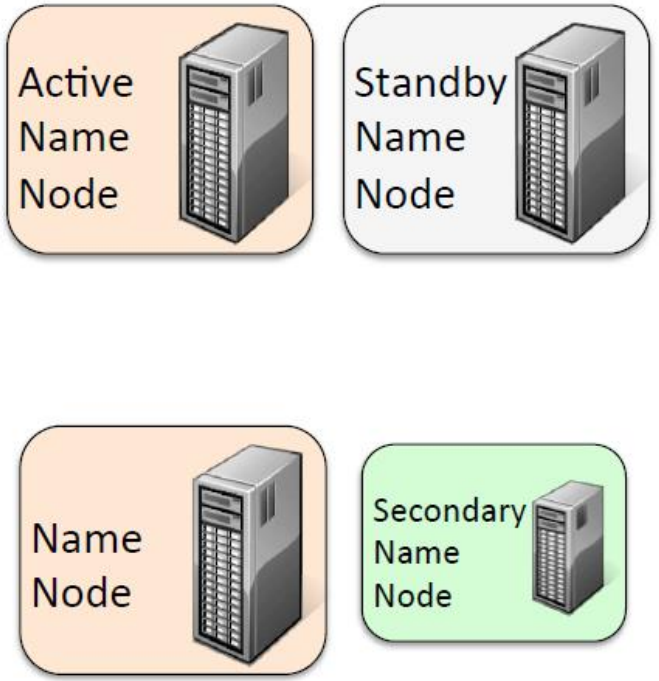
- Use Checksums to validate data
  - Use CRC32
- File Creation
  - Client computes checksum per 512 bytes
  - DataNode stores the checksum
- File access
  - Client retrieves the data and checksum from DataNode
  - If Validation fails, Client tries other replicas

# NameNode Failure

- A single point of failure
- Transaction Log stored in

  multiple directories
  - A directory on the local file system
  - A directory on a remote file system (NFS/CIFS)
- Need to develop a real

  HA solution



# Data Pipelining

- Client retrieves a list of DataNodes on which to place replicas of a block
- Client writes block to the first DataNode
- The first DataNode forwards the data to the next node in the Pipeline
- When all replicas are written, the Client moves on to write the next block in file

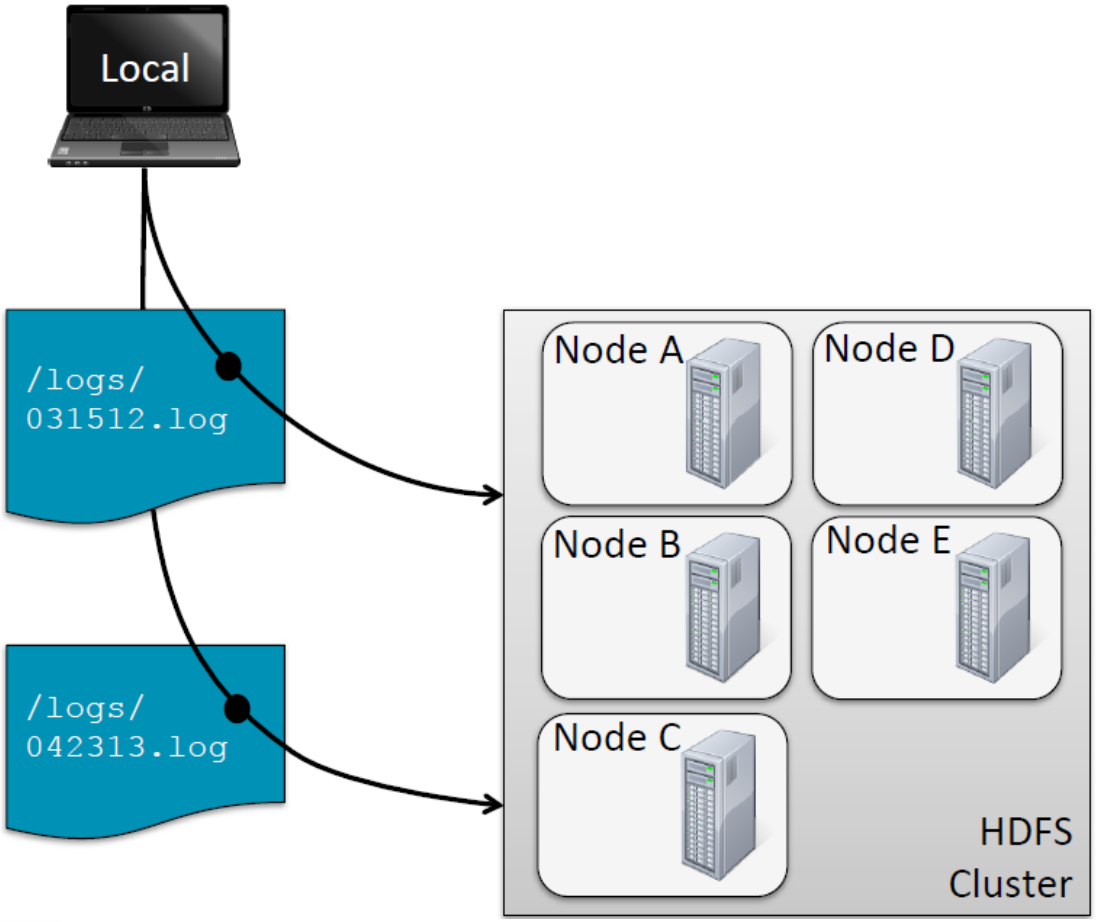
# Rebalancer

- Goal: % disk full on DataNodes should be similar
  - Usually run when new DataNodes are added
  - Cluster is online when Rebalancer is active
  - Rebalancer is throttled to avoid network congestion
  - Command line tool

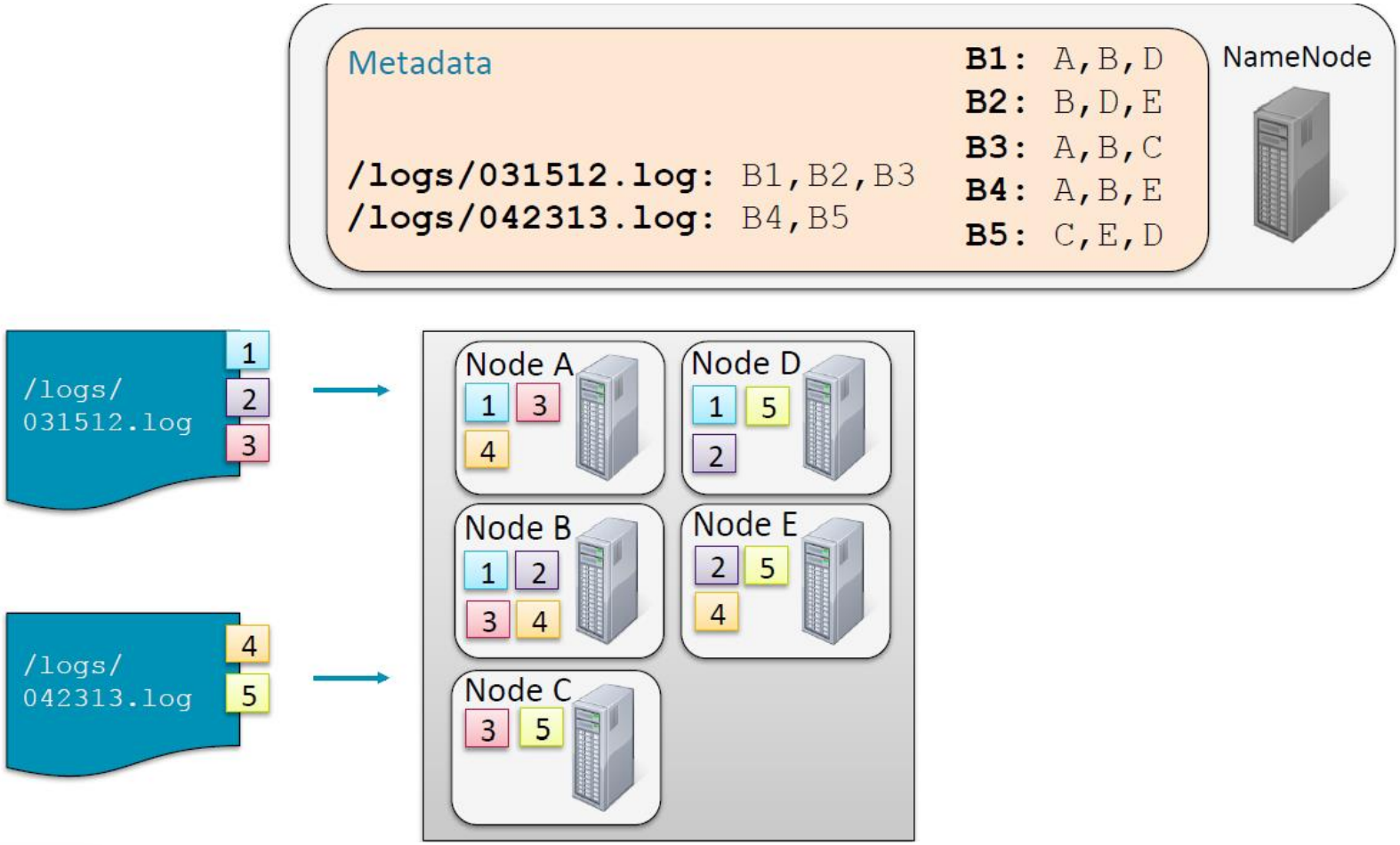
# Secondary NameNode

- Copies FsImage and Transaction Log from Namenode to a temporary directory
- Merges FSImage and Transaction Log into a new FSImage in temporary directory
- Uploads new FSImage to the NameNode
  - Transaction Log on NameNode is purged

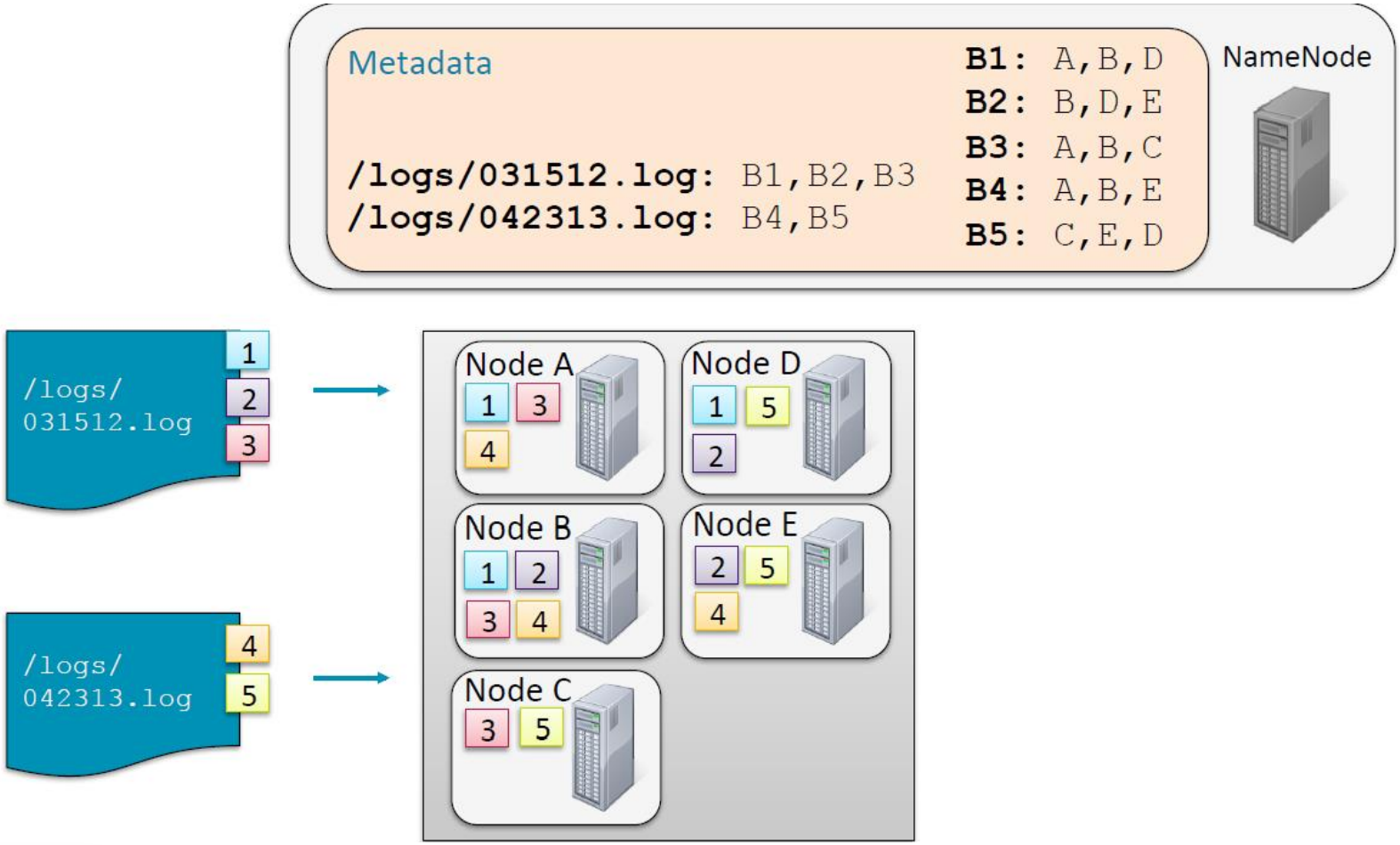
# Example: Storing and Retrieving Files (1)



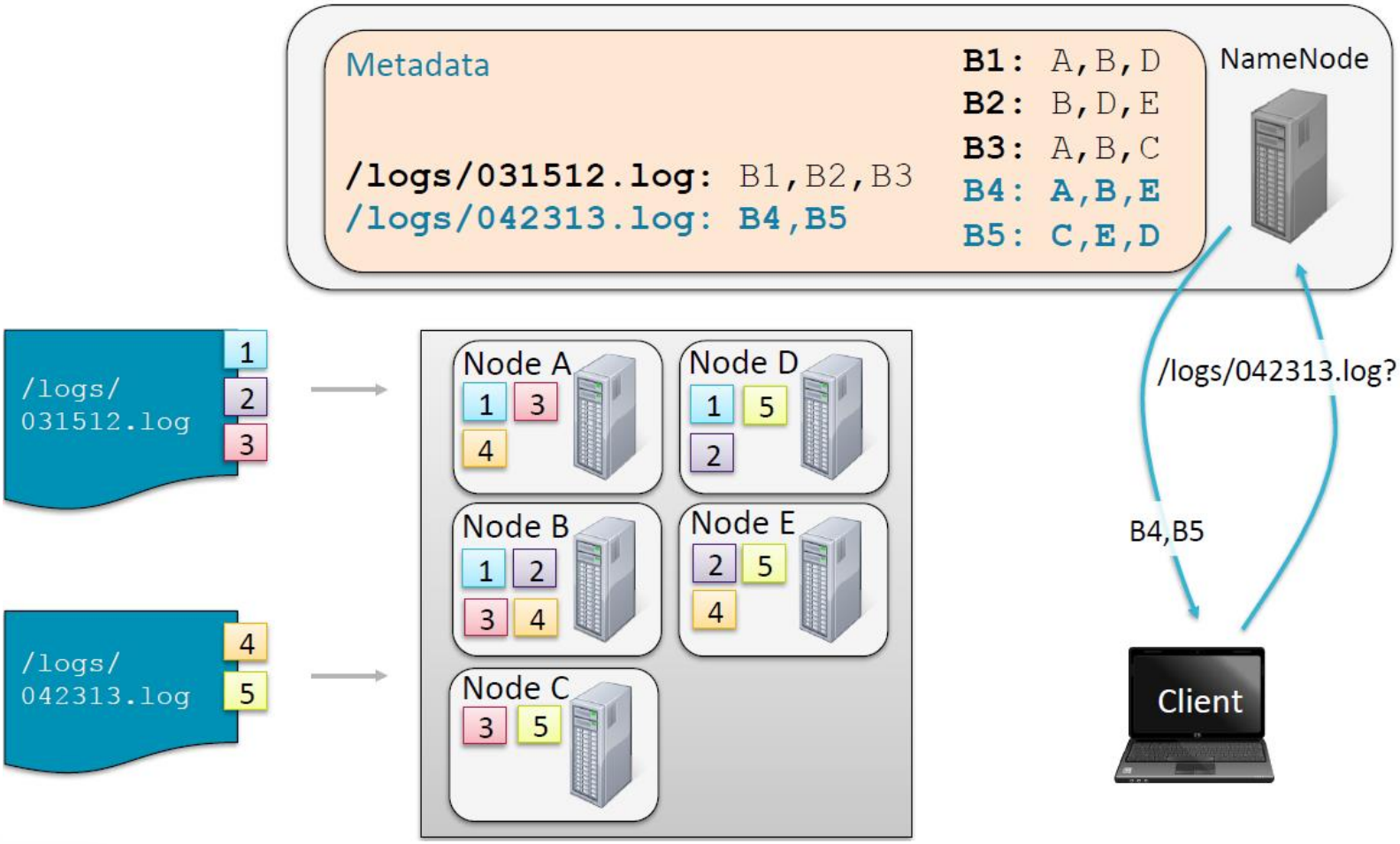
# Example: Storing and Retrieving Files (2)



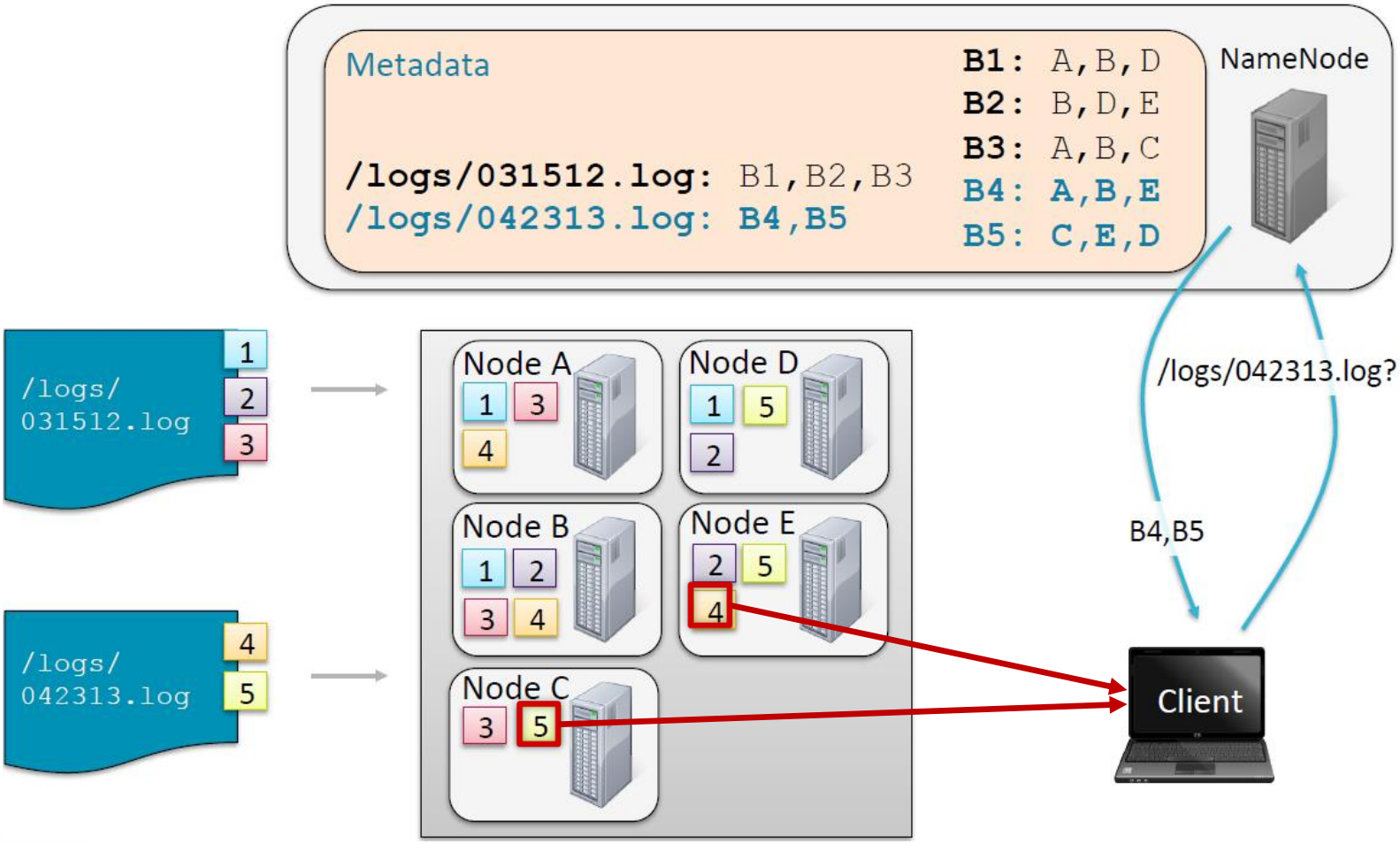
# Example: Storing and Retrieving Files (3)



# Example: Storing and Retrieving Files (4)

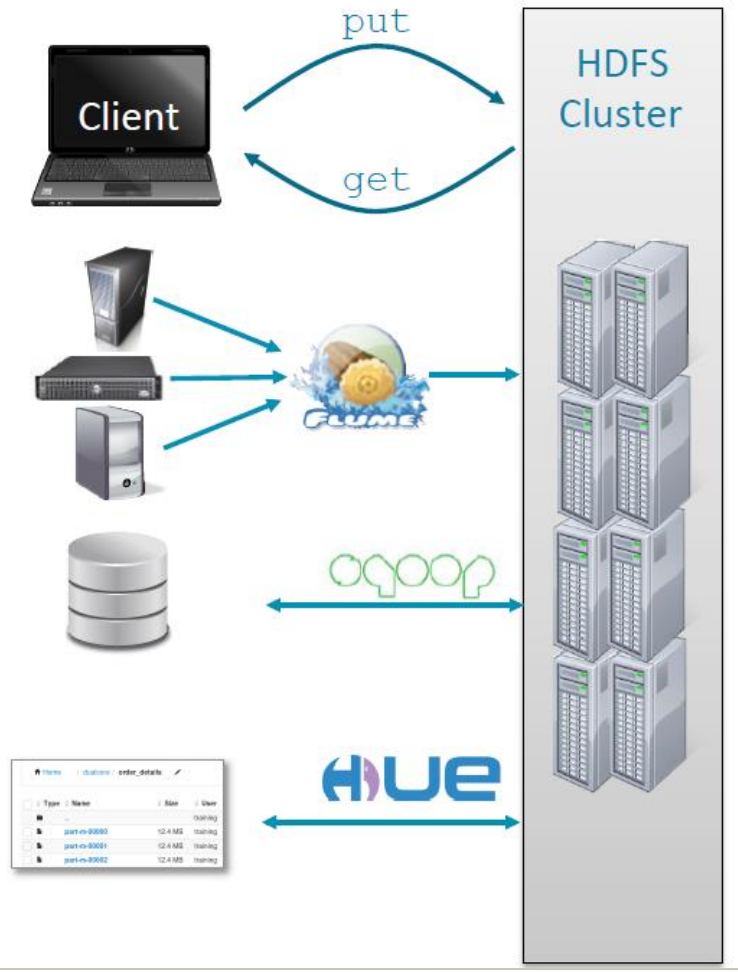


# Example: Storing and Retrieving Files (5)



# Options for Accessing HDFS

- FsShell Commnad line: hadoop fs
- Java API
- Ecosystem Projects
  - Flume
    - Collects data from network sources (e.g., system logs)
  - Sqoop
    - Transfers data between HDFS and RDBMS
  - Hue
    - Web-based interactive UI to browse, update, download, and view files



# User Interface

- Commands for HDFS User:
  - hadoop dfs -mkdir /foodir
  - hadoop dfs -cat /foodir/myfile.txt
  - hadoop dfs -rm /foodir/myfile.txt
- Commands for HDFS Administrator
  - hadoop dfsadmin -report
  - hadoop dfsadmin -decommision datanodename
- Web Interface
  - http://host:port/dfshealth.jsp

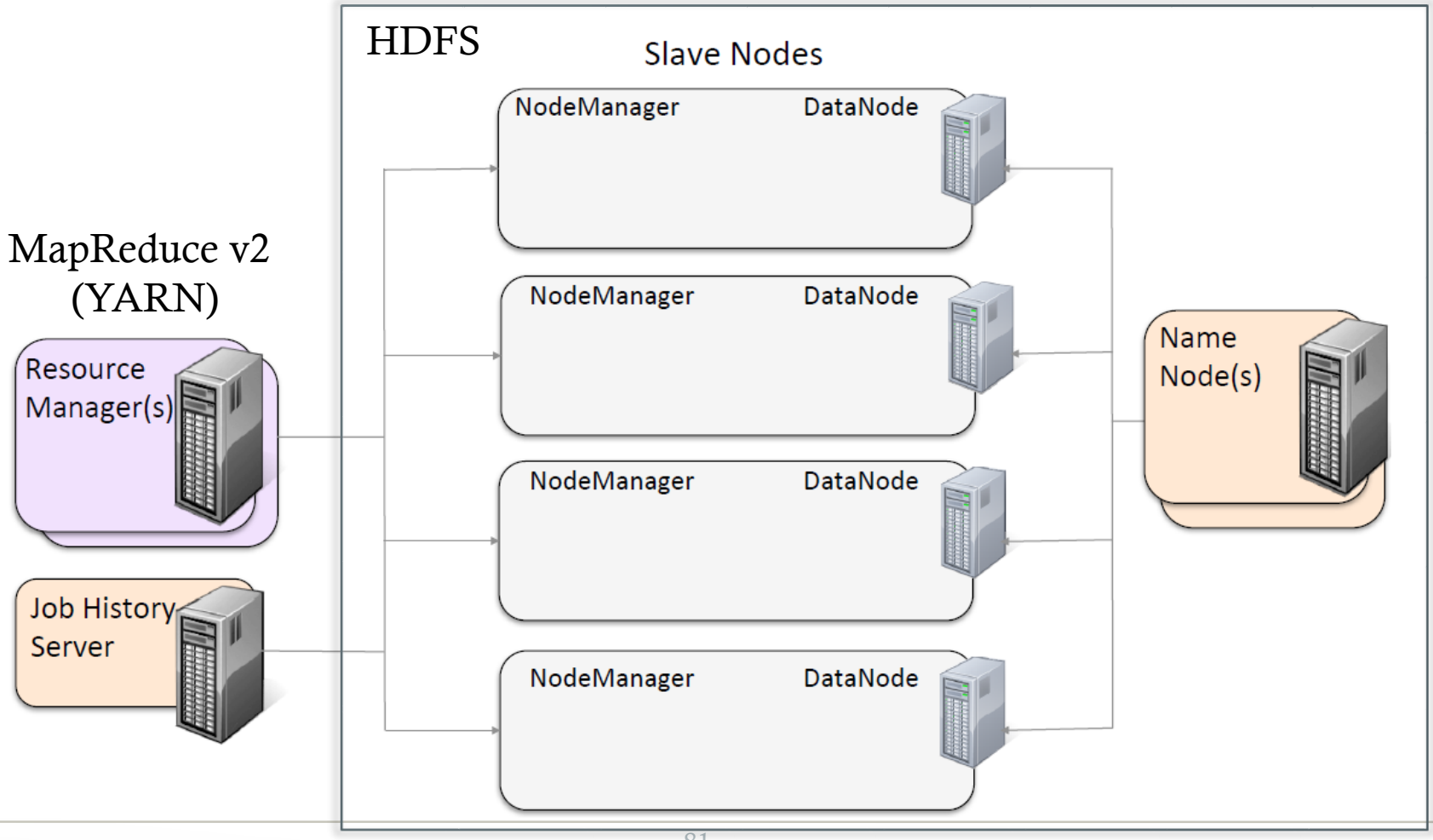
# Hadoop fs Examples (1)

- Copy file foo.txt from local disk to the user's directory in HDFS
  - hadoop fs –put foo.txt foo.txt
  - This will copy the file to /user/username/foo.txt
- Get a directory listing of the user's home directory in HDFS
  - hadoop fs –ls
- Get a directory listing of the HDFS root directory
  - hadoop fs –ls /

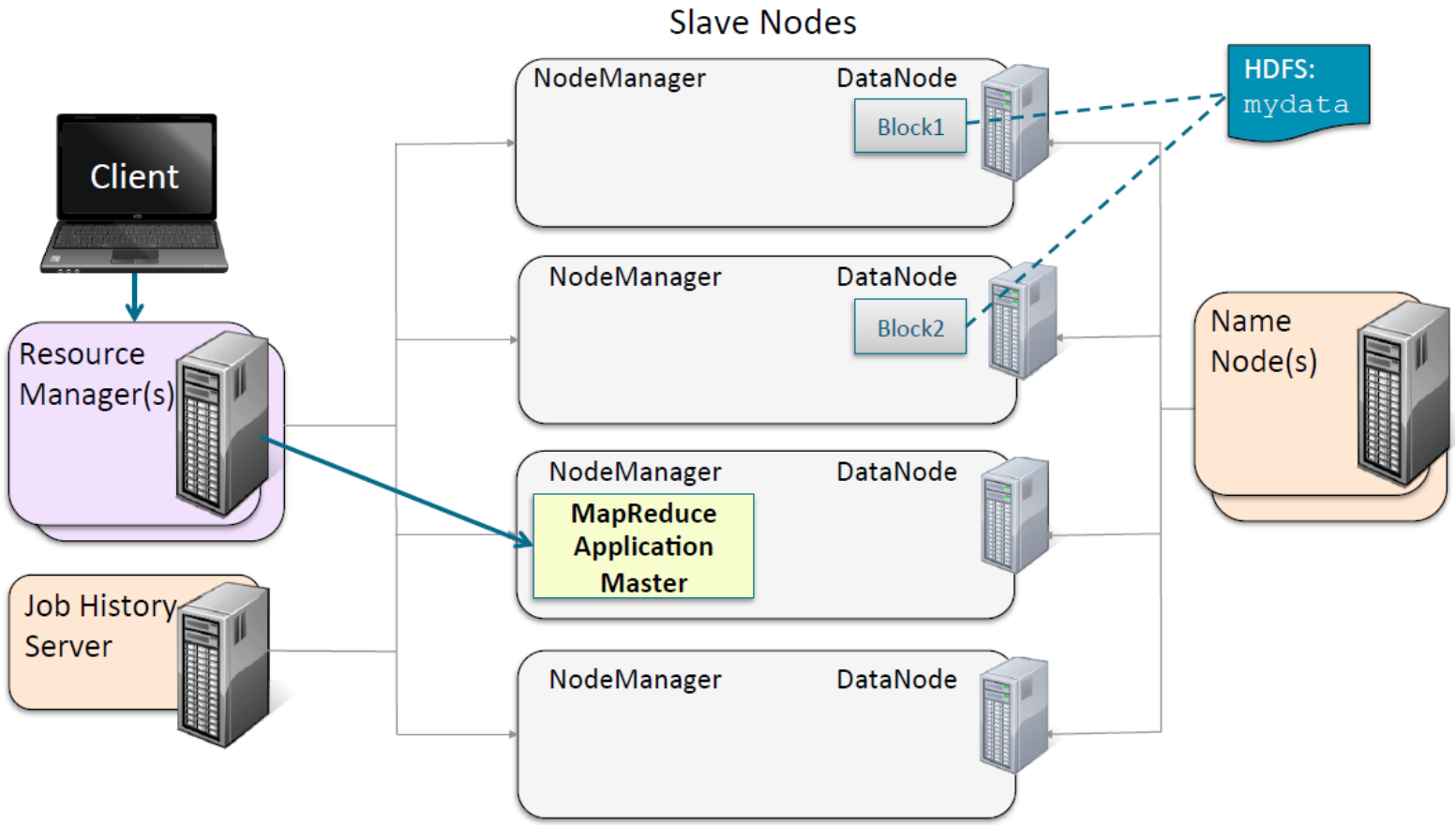
# Hadoop fs Examples (2)

- Display the contents of the HDFS file /user/csc/bar.txt
  - hadoop fs –cat /user/csc/bar.txt
- Copy that file to the local disk, named as baz.txt
  - hadoop fs –get /user/csc/bar.txt baz.txt
- Create a directory called input under the user's home directory
  - hadoop fs –mkdir input
- Delete the directory input_old and all its contents
  - hadoop fs –rm –r input_old

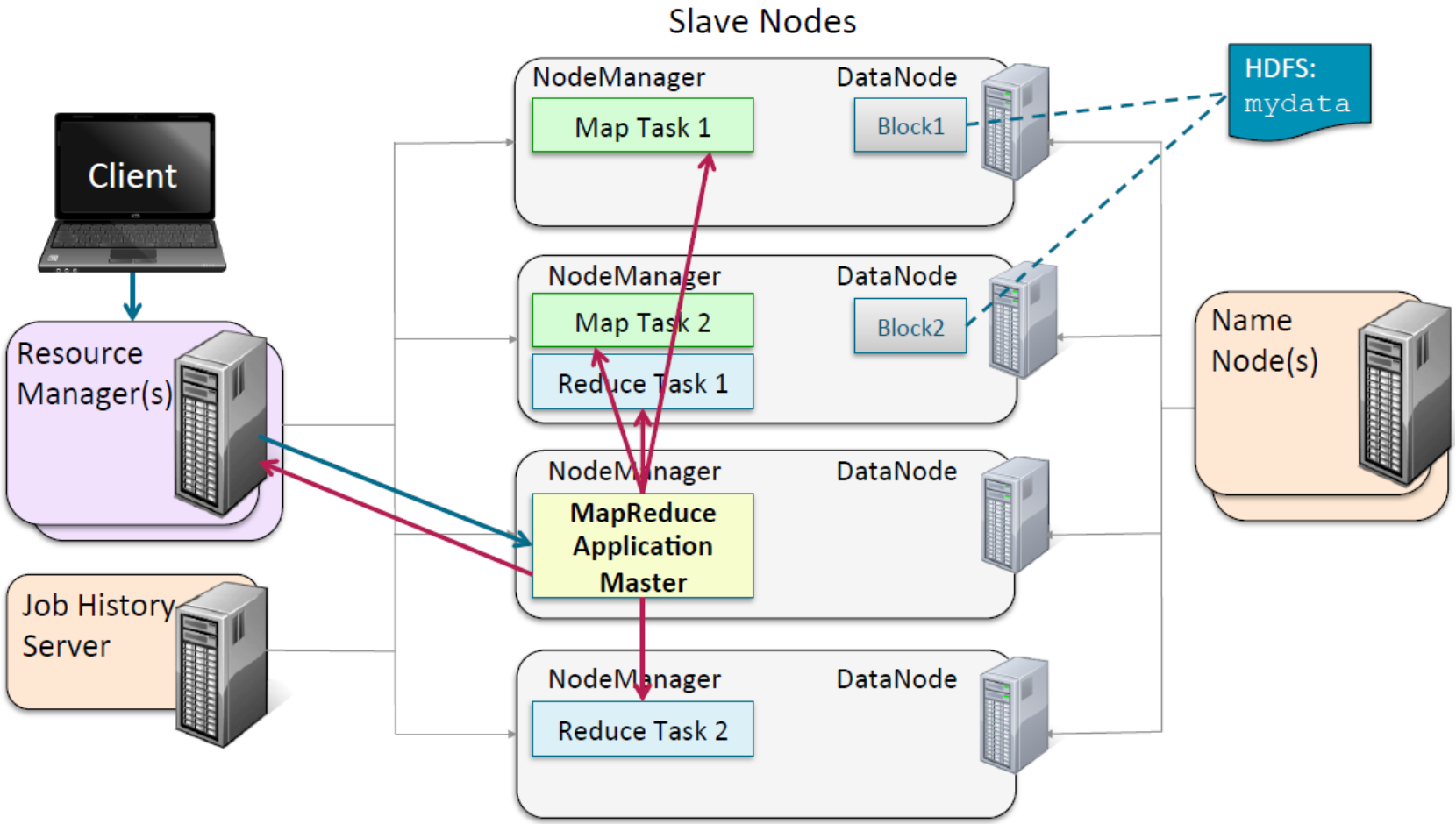
# A MapReduce v2 Cluster



# Running a Job on a MapReduce v2 Cluster (1)



# Running a Job on a MapReduce v2 Cluster (2)



# Hadoop 在Ubuntu安裝方式

- Hadoop Single Node Cluster 是只以一台機器建立Hadoop環境。
- Hadoop Multi Node Cluster 則是以多台機器建立Hadoop環境。

# Hadoop Single Node Cluster 安裝

1. 安裝JDK: Hadoop的核心為JAVA，所以必須先安裝JAVA
2. 設定SSH無密碼登入: Hadoop透過SSH連線與本機及其他主機連線，因此必須先設定SSH連線。
3. 下載安裝Hadoop:到Hadoop官網下載Hadoop，並且安裝至Ubuntu中。
4. 設定Hadoop環境變數:Hadoop的組態設定檔為/usr/local/hadoop/etc/hadoop

# Hadoop Single Node Cluster 安裝

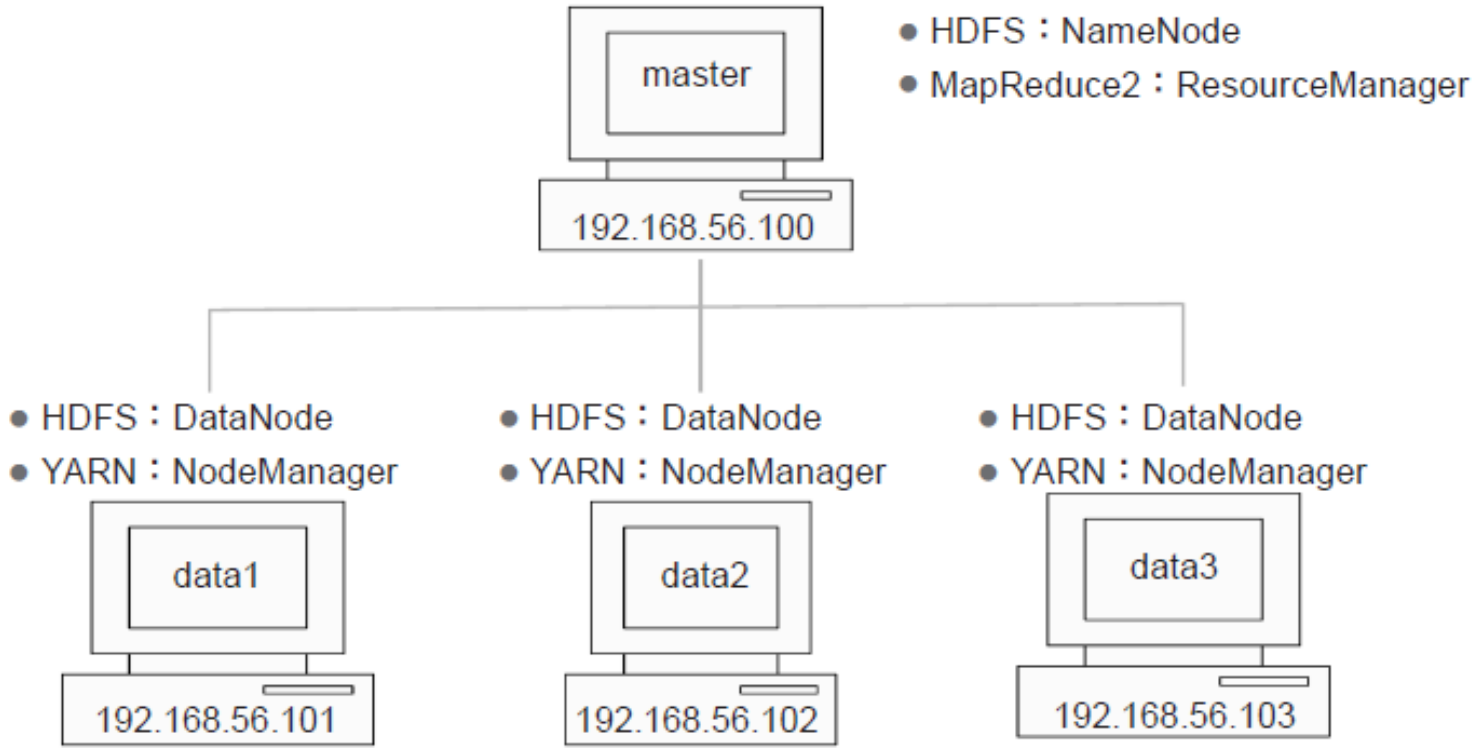
6. 建立與格式化HDFS目錄: HDF目錄是儲存HDFS檔案的地方，在啟動hadoop之前，必須建立與格式化HDFS目錄。

7. 啟動Hadoop

8. 開啟Hadoop Web介面

詳細設定方式請參考
http://pythonsparkhadoop.blogspot.tw/2017/01/hadoop-ubuntu.html#more

# Hadoop Multi Node Cluster安裝



# Hadoop Multi Node Cluster安裝

1. 複製Single Node Cluster 到data1
2. VirtualBox介面卡設定
   - 設定兩張介面卡
     - 介面卡1設定為'NAT介面卡'，可以透過Host主機連結到外部網路。
     - 介面卡2設定為'內部網路介面卡'，用以建立內部網路。
3. 設定data1伺服器
   - 設定Multi Node Cluster伺服器:組態設定檔共用部分
     - 固定IP、hostname、core-site.xml、yarn-site.xml、mapred-site.xml、hdfs-site.xml

# Hadoop Multi Node Cluster安裝

4. 複製data1伺服器到data2、data3、master

5. 設定data2伺服器:設定固定IP、hostname

6. 設定data3伺服器:設定固定IP、hostname

7.設定master伺服器:設定固定IP、hostname、hdfs-site.xml、masters、slaves

8. Master透過SSH連線到data1、data2、data3、建立HDFS目錄

9. 建立與格式化namenode HDF目錄

# Hadoop Multi Node Cluster安裝

10. 起動Hadoop Multi Node Cluster
11. 開啟Hadoop ResourceManager Web介面
12. 開啟NameNode Web介面

詳細設定方式請參考
http://pythonsparkhadoop.blogspot.tw/2017/01/hadoop-ubuntu.html#more

# Homework 2: Using HDFS

- In this hands-on exercise, you are required to set your Hadoop Single Node Cluster in order to get acquainted with the Hadoop tools.

1. Prepare your dataset to update into your Hadoop system
2. Set up your environment
3. Run the commands in a terminal window to view the contents of the /user directory: $ hadoop fs –ls /user
4. Update your files into HDFS with FsShell
5. Viewing and manipulating your files
6. Finally, write a homework report to record everything you did in the exercise

# Any Questions?