Hadoop Overview for Managers

PART 1

Outline

PART 1

- Hadoop Overview
  - Traditional Computing Systems and Limitations
  - Why Big Data?
  - Why Hadoop?
  - Hadoop Basic Concepts
  - Where Hadoop fits in the Enterprise
- Hadoop Architecture
  - Building blocks
  - HDFS
  - Demo
- **Break**

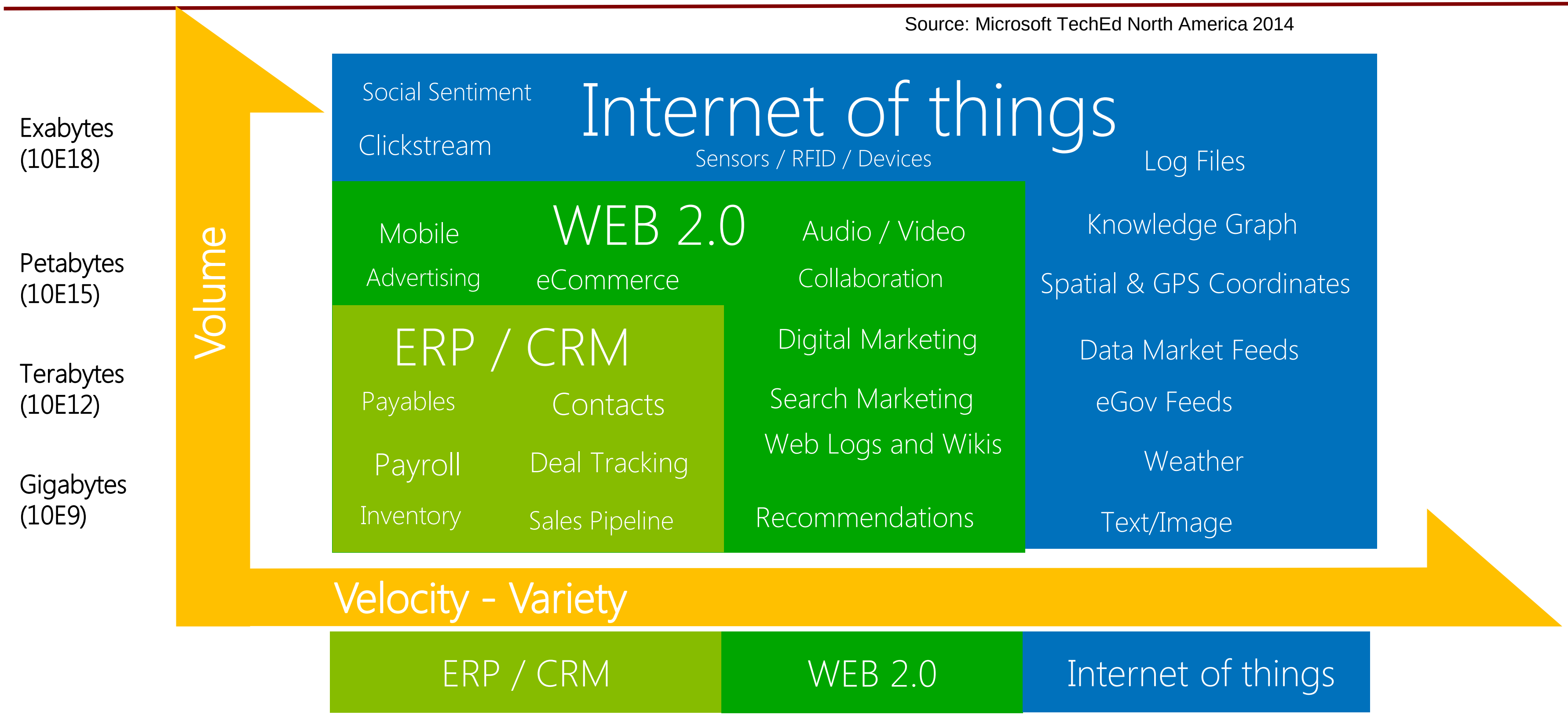
PART 2

- YARN Architecture
  - Yarn Overview
  - MapReduce
  - Demo
- Tools and technology for Hadoop ecosystem
- Hadoop Real Life Use Cases
- Establishing a Big Data Center of Excellence
  - Justifying business value for your organization
  - Challenges on building a production solution
  - Recommended organizational structure
  - Best Practices: Steps to effectively deploy Hadoop
- Recap and Q&A

ERA OF BIG DATA

Era of Big-Data

Source: Microsoft TechEd North America 2014



Exabytes (10E18)

Petabytes (10E15)

Terabytes (10E12)

Gigabytes (10E9)

Volume

Velocity - Variety

Internet of things

Social Sentiment

Clickstream

Sensors / RFID / Devices

Log Files

Knowledge Graph

Spatial & GPS Coordinates

Data Market Feeds

eGov Feeds

Weather

Text/Image

WEB 2.0

Mobile

Advertising

eCommerce

Audio / Video

Collaboration

Digital Marketing

Search Marketing

Web Logs and Wikis

Recommendations

ERP / CRM

Payables

Contacts

Payroll

Deal Tracking

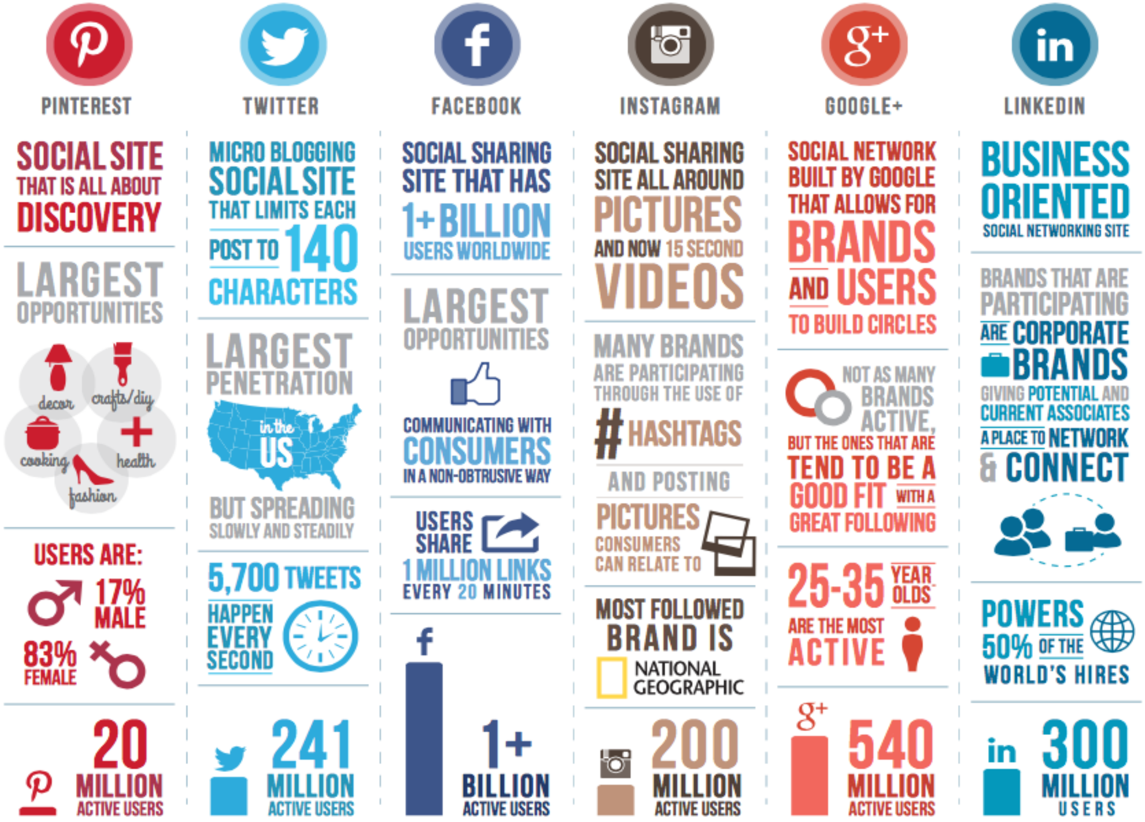
Inventory

Sales Pipeline

ERP / CRM | WEB 2.0 | Internet of things

Storage/GB	1980	1990	2000	2010
	190,000$	9,000$	15$	0.07$

Who is generating so much data?

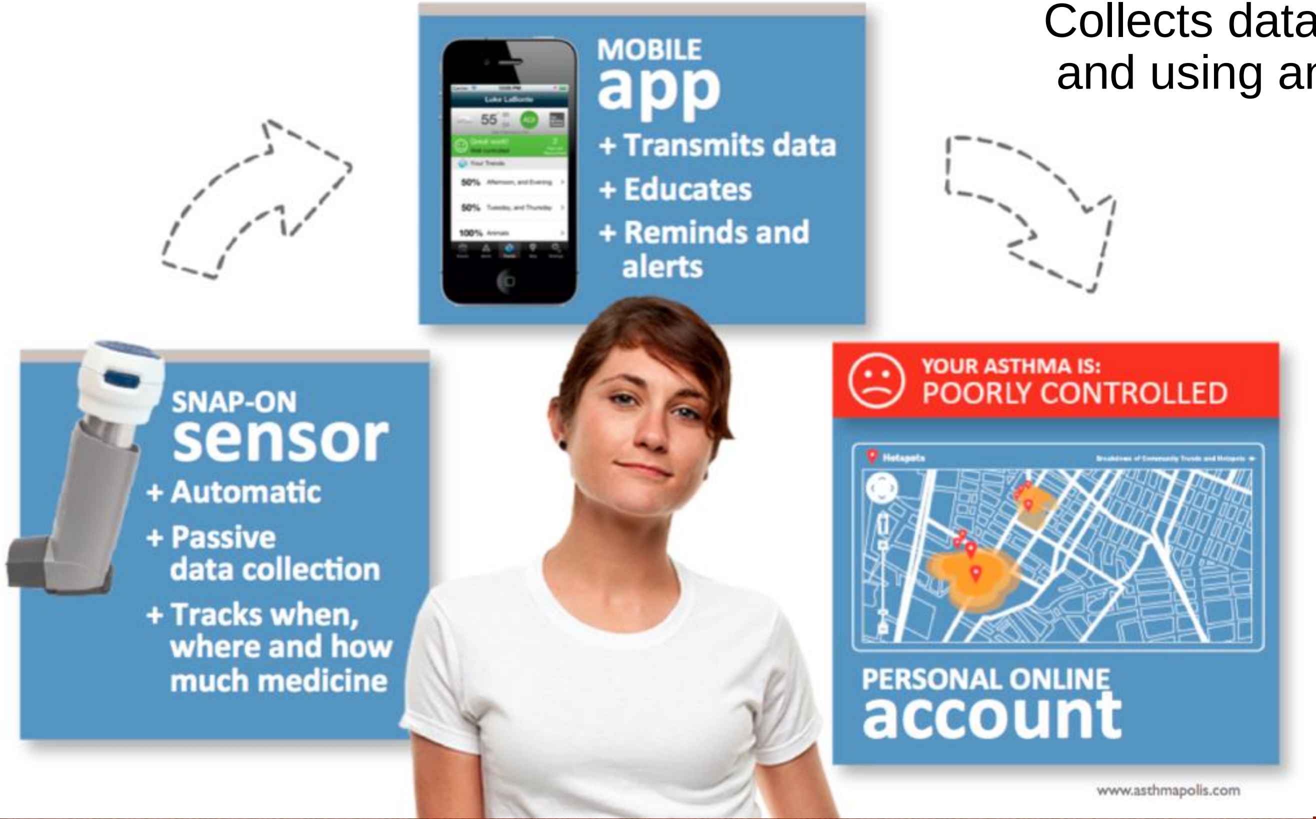


Statistics as of 4.25.2014 Designed by: Leverage - leveragenewagemedia.com

Source https://leveragenewagemedia.com/

Big Data in Healthcare - Asthmapolis

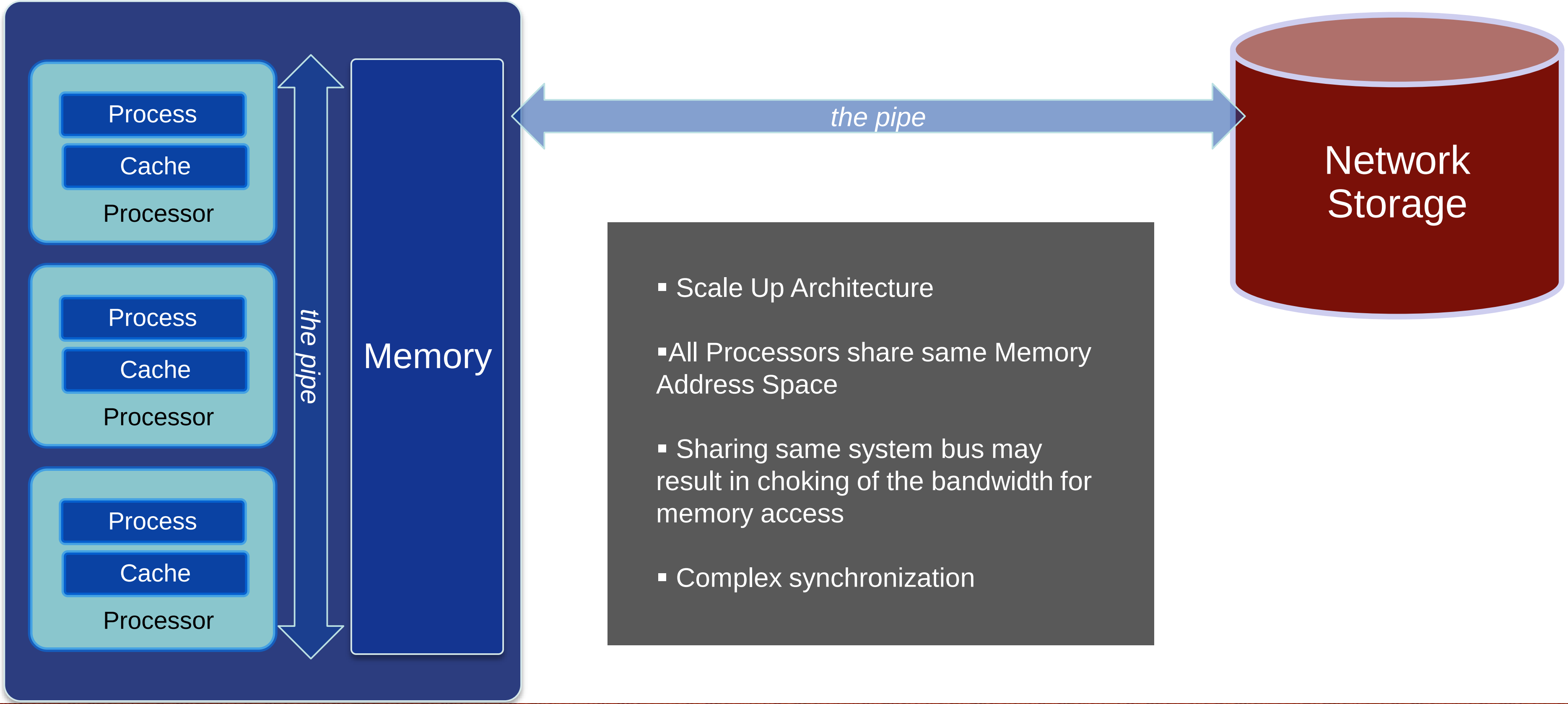
Collects data from patients (inhalers) and using analytics help them better manage their Asthma



Big Data Solution – key characteristics

- **Must Scale with increasing volume**
  - Performance
  - Availability
  - Cost
- **Support variety of data**
  - Structured
  - Unstructured
  - Semi Structured

“Shared Everything” Architecture



- Scale Up Architecture
- All Processors share same Memory Address Space
- Sharing same system bus may result in choking of the bandwidth for memory access
- Complex synchronization

(Is) Distributed Systems the solution

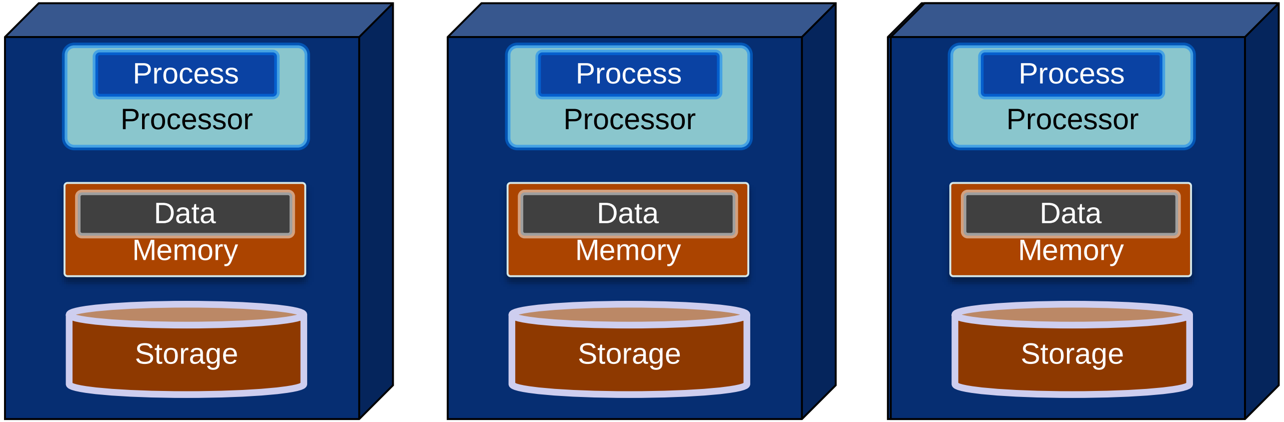
- Programming Model is complex
- Data exchange requires synchronization
- Failures are expensive and needs to be managed
- Does not scale for Large volumes of data – network interconnects in a datacenter are expensive!

New Approach to Distributed Systems

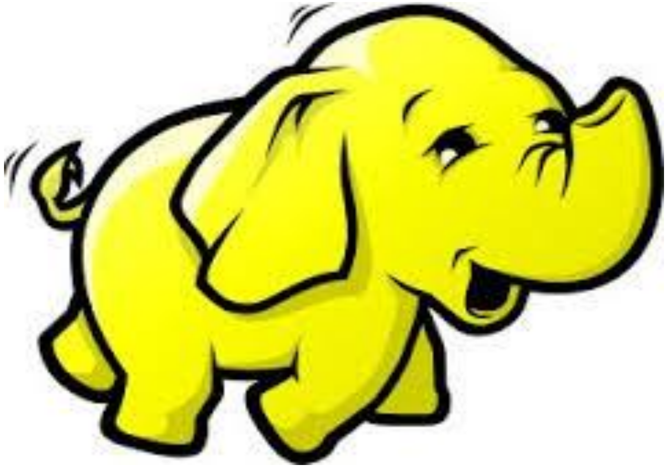
- Must Scale with increasing volume
  - Performance
  - Availability
  - Cost
- Support variety of data
  - Structured
  - Unstructured
  - Semi Structured

- ○ Shared Nothing Architecture
- ○ **Data Locality**
- ○ No synchronization requirement among the nodes
- ○ **Designed for failure** – Multiple copies of data
- ○ Consistent – individual failures does not fail the job
- ○ **Support "commodity"** hardware & heterogeneous

Shared Nothing Architecture

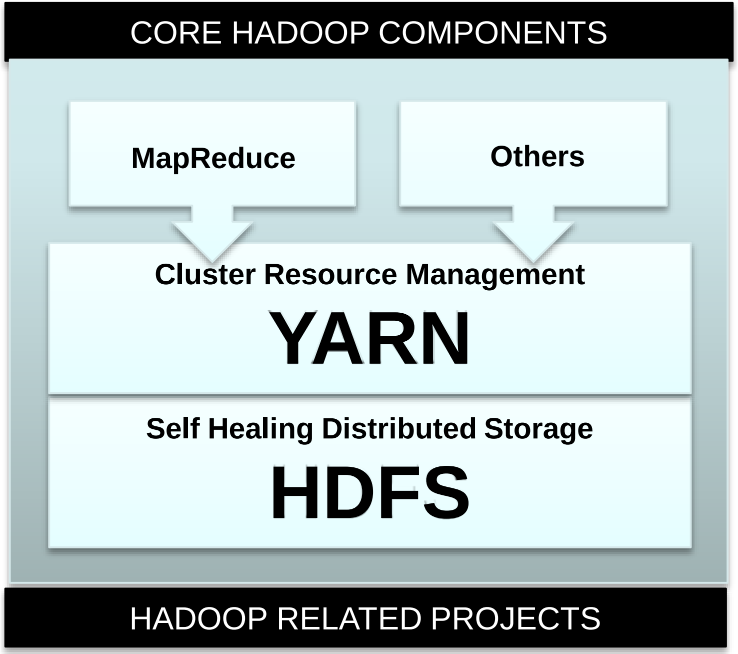


- **Data is sharded (partitioned) amongst the nodes**
- **Computation is local to the nodes – no need to get the data from elsewhere**
- **No synchronization, Simple implementation**



HADOOP OVERVIEW

Hadoop Core Components



- ○ **Hadoop Common:** A set of common libraries and utilities used by Hadoop modules.

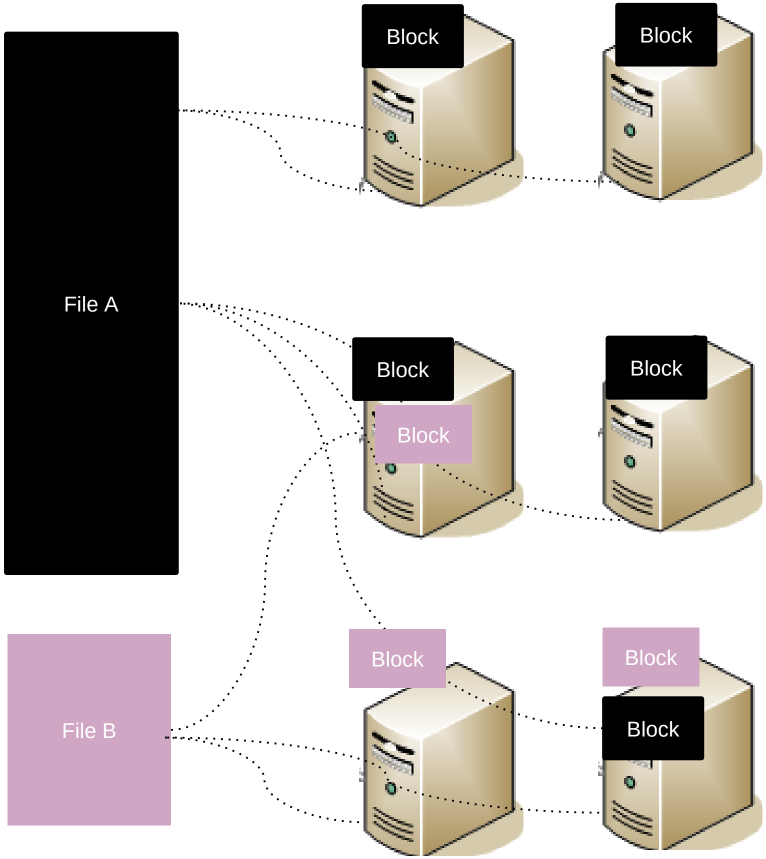
- ○ **Hadoop Distributed File System (HDFS)**: A scalable and fault tolerant distributed filesystem to data in any form.

- ○ **Yet Another Resource Negotiator (YARN)**: From Hadoop 2.0, YARN is the cluster management layer to handle various workloads on the cluster.

- ○ **MapReduce:** MapReduce is a framework that allows parallel processing of data in Hadoop.

Growing number of eco-system Projects

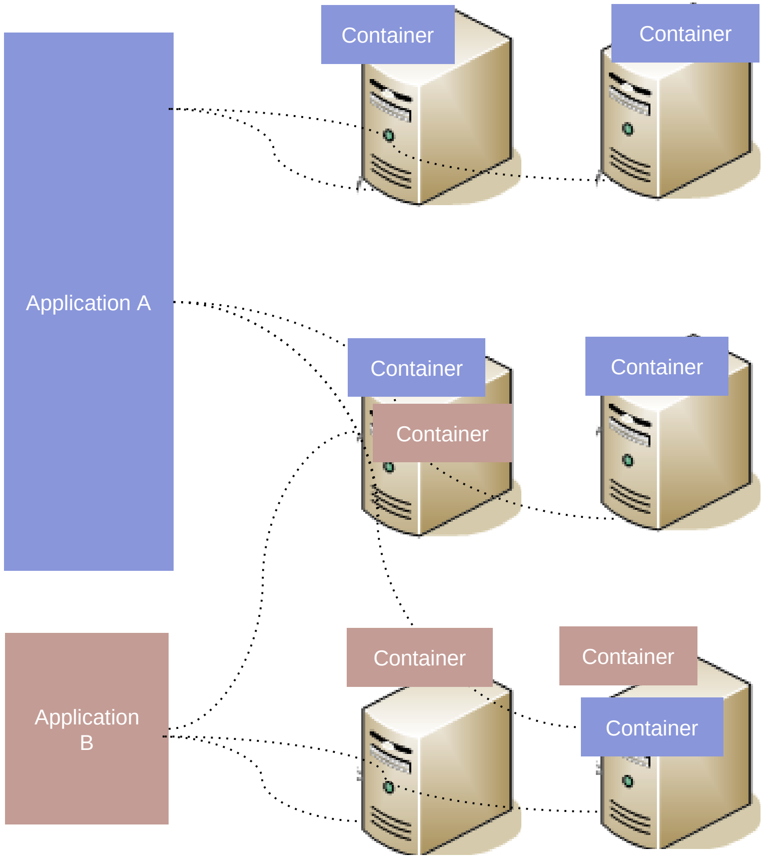
HDFS Overview



- Distributed Storage

- Designed to store very large files
- Sharded storage for high throughput
- Replicated storage for failure protection
- Self healing
- Add new disks or nodes to scale

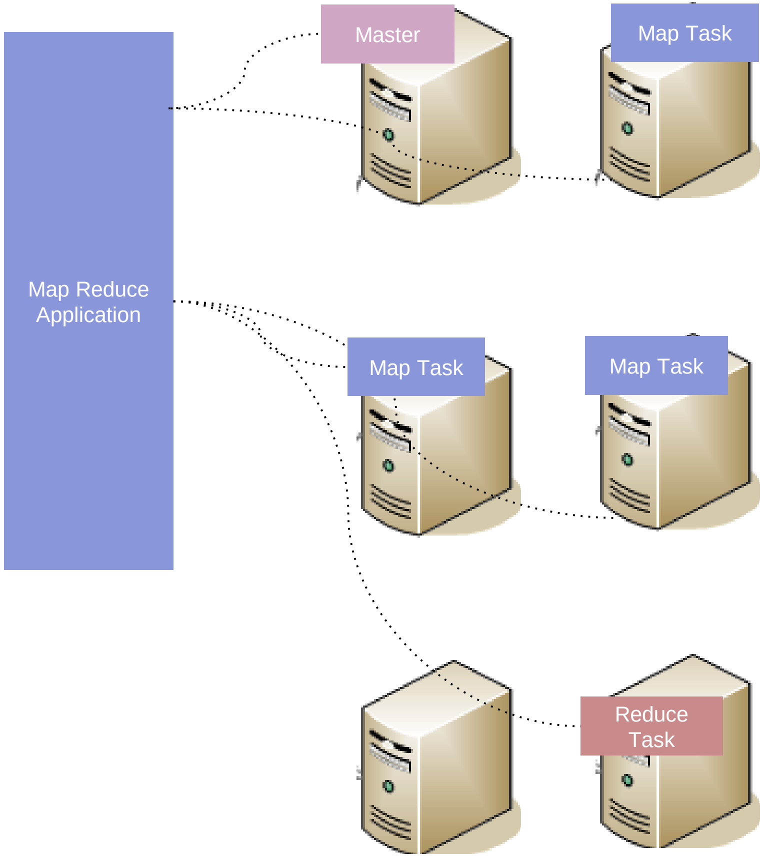
YARN Overview



- **<u>Distributed Compute</u>**

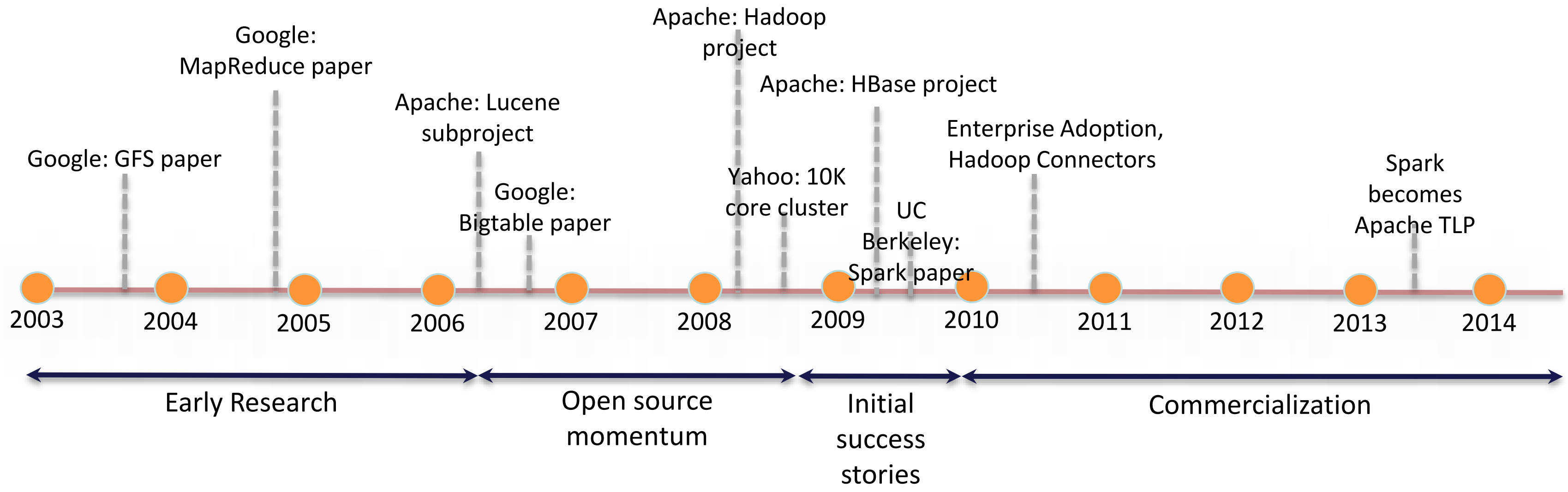
- <u>Y</u>et <u>A</u>nother <u>R</u>esource <u>N</u>egotiator
- Run Java code on arbitrary node(s) depending on availability
- “Cloud without the Virtualization”
- Applications can occupy resources as per need.
- Common model of distributing code and accessing Data
- Self healing

MapReduce Overview



- Model for processing large amount of data in parallel
- Oriented towards batch processing
- Programming Model derived from functional programming
- Built on top of YARN
- I/O on HDFS and others

Timeline


Apache: Hadoop project
Google: MapReduce paper
Apache: HBase project
Apache: Lucene subproject
Enterprise Adoption, Hadoop Connectors
Google: GFS paper
Spark becomes Apache TLP
Yahoo: 10K core cluster
Google: Bigtable paper
UC Berkeley: Spark paper
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
Early Research
Open source momentum
Initial success stories
Commercialization

Why Hadoop?

- **Runs on commodity hardware (and the cloud)**
  - Low cost
  - Ease of maintenance
- Scales well
- **Strong Ecosystem**
- **Open Source**
  - Apache 2.0 License
  - Strong Community
- Runs on the JVM

When to use

Relational Databases:

Use when:

- Interactive OLAP Analytics (<1sec)
- Multistep ACID Transactions
- 100% SQL Compliance

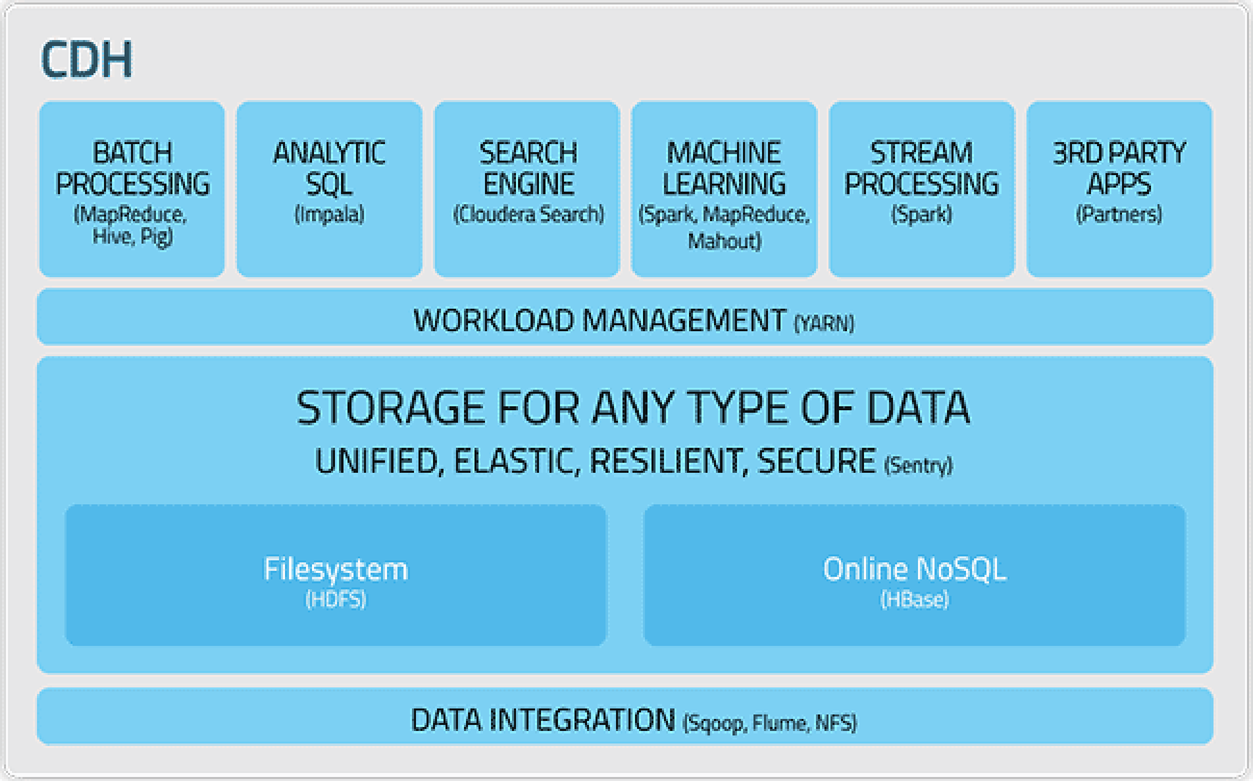
Hadoop:

Use when:

- Structured or Not (Flexibility)
- Scalability of Storage/Compute
- Complex Data Processing

HADOOP DISTRIBUTION AND FRAMEWORKS

Cloudera CDH



CDH

BATCH PROCESSING (MapReduce, Hive, Pig)

ANALYTIC SQL (Impala)

SEARCH ENGINE (Cloudera Search)

MACHINE LEARNING (Spark, MapReduce, Mahout)

STREAM PROCESSING (Spark)

3RD PARTY APPS (Partners)

WORKLOAD MANAGEMENT (YARN)

STORAGE FOR ANY TYPE OF DATA

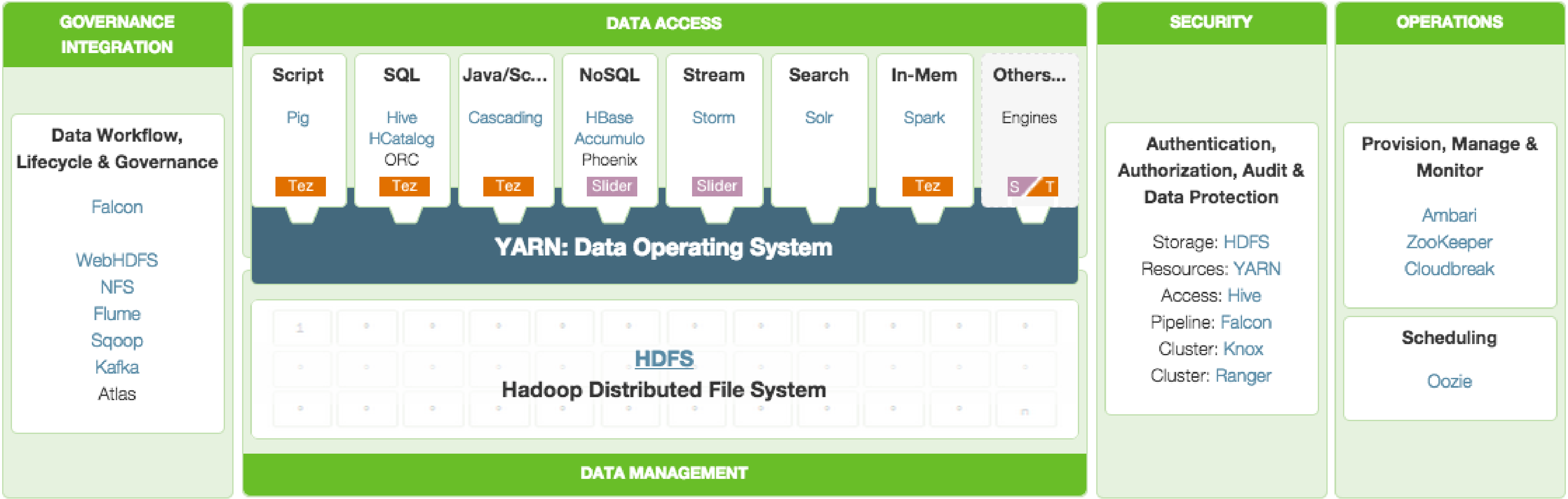
UNIFIED, ELASTIC, RESILIENT, SECURE (Sentry)

Filesystem (HDFS)

Online NoSQL (HBase)

DATA INTEGRATION (Sqoop, Flume, NFS)

Hortonworks HDP



GOVERNANCE INTEGRATION

Data Workflow, Lifecycle & Governance

Falcon

WebHDFS
NFS
Flume
Sqoop
Kafka
Atlas

DATA ACCESS

Script	SQL	Java/Sc...	NoSQL	Stream	Search	In-Mem	Others...
Pig	Hive HCatalog ORC	Cascading	HBase Accumulo Phoenix	Storm	Solr	Spark	Engines
Tez	Tez	Tez	Slider	Slider		Tez	S / T

YARN: Data Operating System

HDFS
Hadoop Distributed File System

DATA MANAGEMENT

SECURITY

Authentication, Authorization, Audit & Data Protection

Storage: HDFS
Resources: YARN
Access: Hive
Pipeline: Falcon
Cluster: Knox
Cluster: Ranger

OPERATIONS

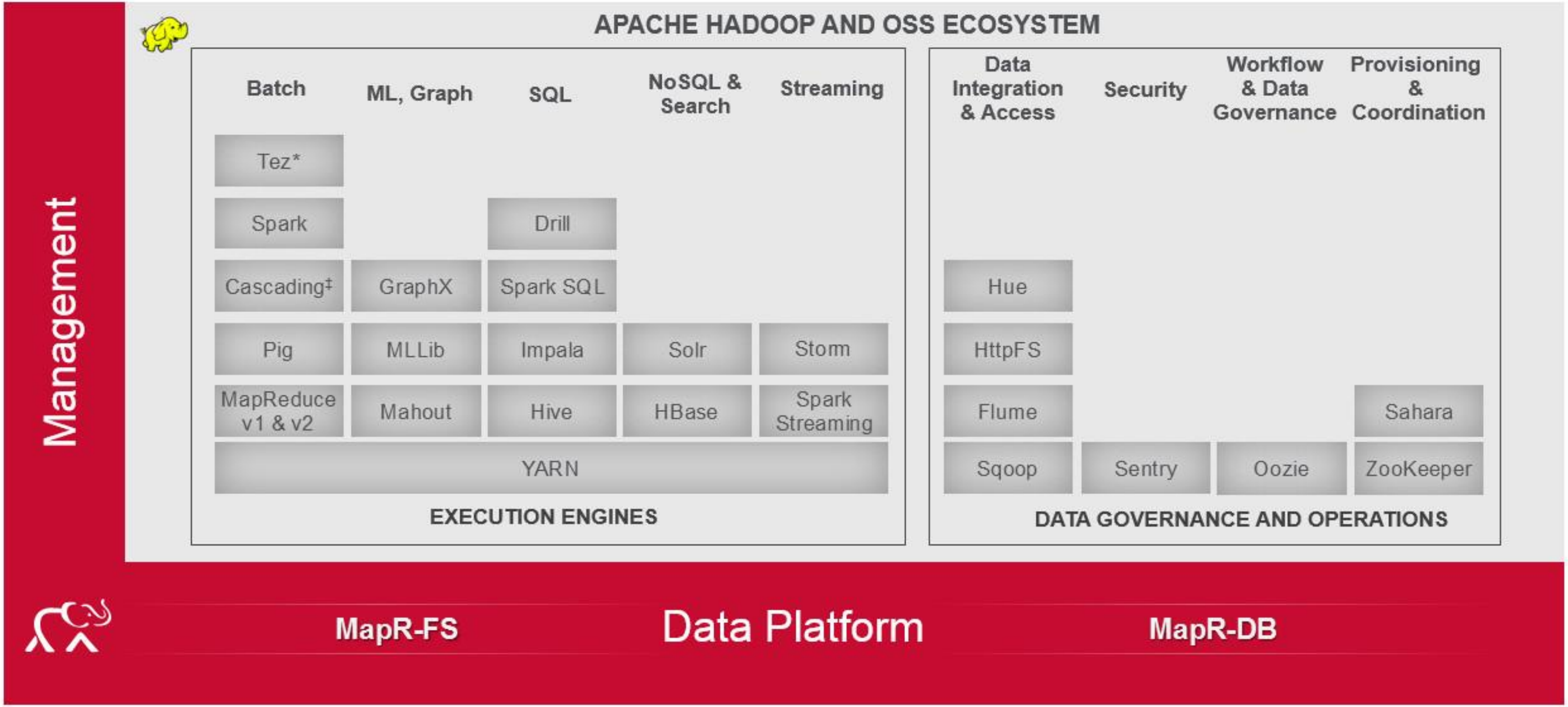
Provision, Manage & Monitor

Ambari
ZooKeeper
Cloudbreak

Scheduling

Oozie

MapR M5



APACHE HADOOP AND OSS ECOSYSTEM

Batch	ML, Graph	SQL	NoSQL & Search	Streaming
Tez*				
Spark		Drill		
Cascading‡	GraphX	Spark SQL		
Pig	MLLib	Impala	Solr	Storm
MapReduce v1 & v2	Mahout	Hive	HBase	Spark Streaming
YARN				

EXECUTION ENGINES

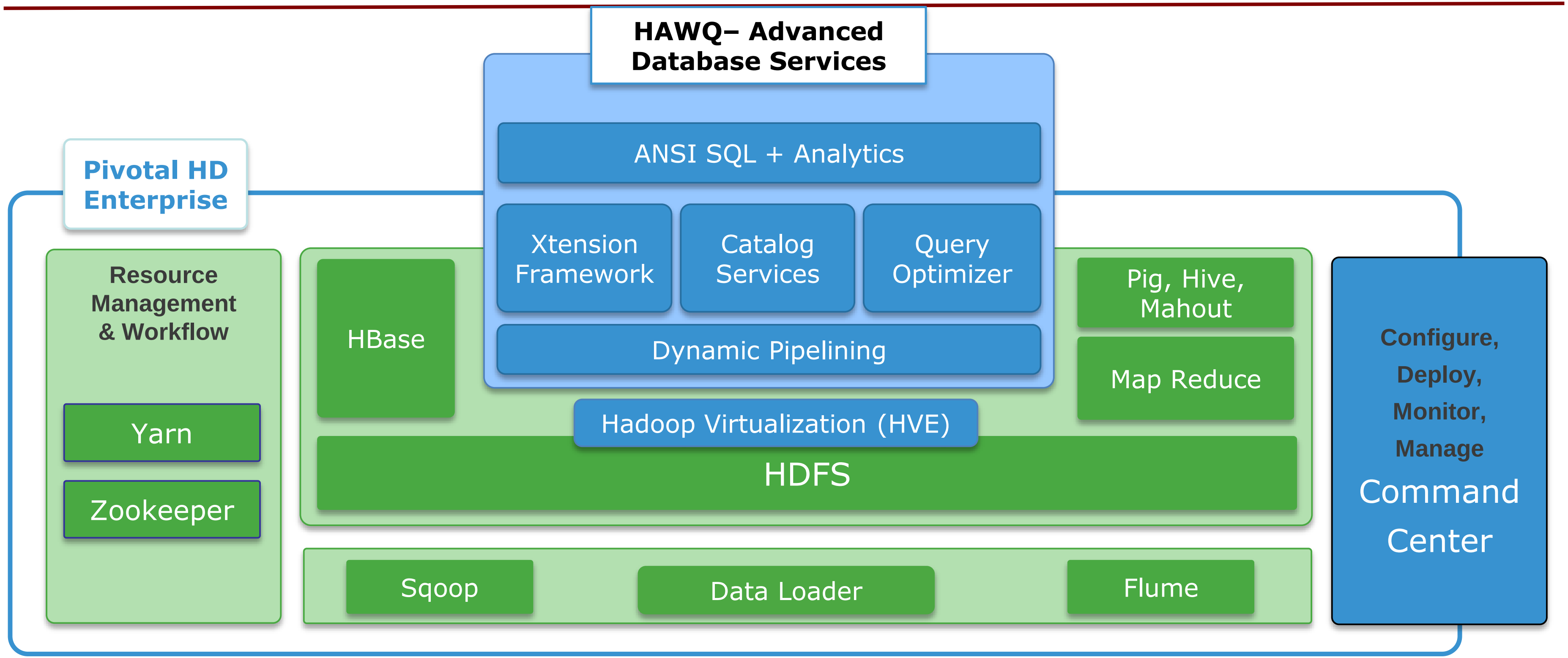
Data Integration & Access	Security	Workflow & Data Governance	Provisioning & Coordination
Hue			
HttpFS			
Flume			Sahara
Sqoop	Sentry	Oozie	ZooKeeper

DATA GOVERNANCE AND OPERATIONS

MapR-FS | Data Platform | MapR-DB

*Developer preview ‡ Certified on MapR

Pivotal HD



HAWQ– Advanced Database Services

ANSI SQL + Analytics

Xtension Framework

Catalog Services

Query Optimizer

Dynamic Pipelining

Pivotal HD Enterprise

Resource Management & Workflow

Yarn

Zookeeper

HBase

Pig, Hive, Mahout

Map Reduce

Hadoop Virtualization (HVE)

HDFS

Sqoop

Data Loader

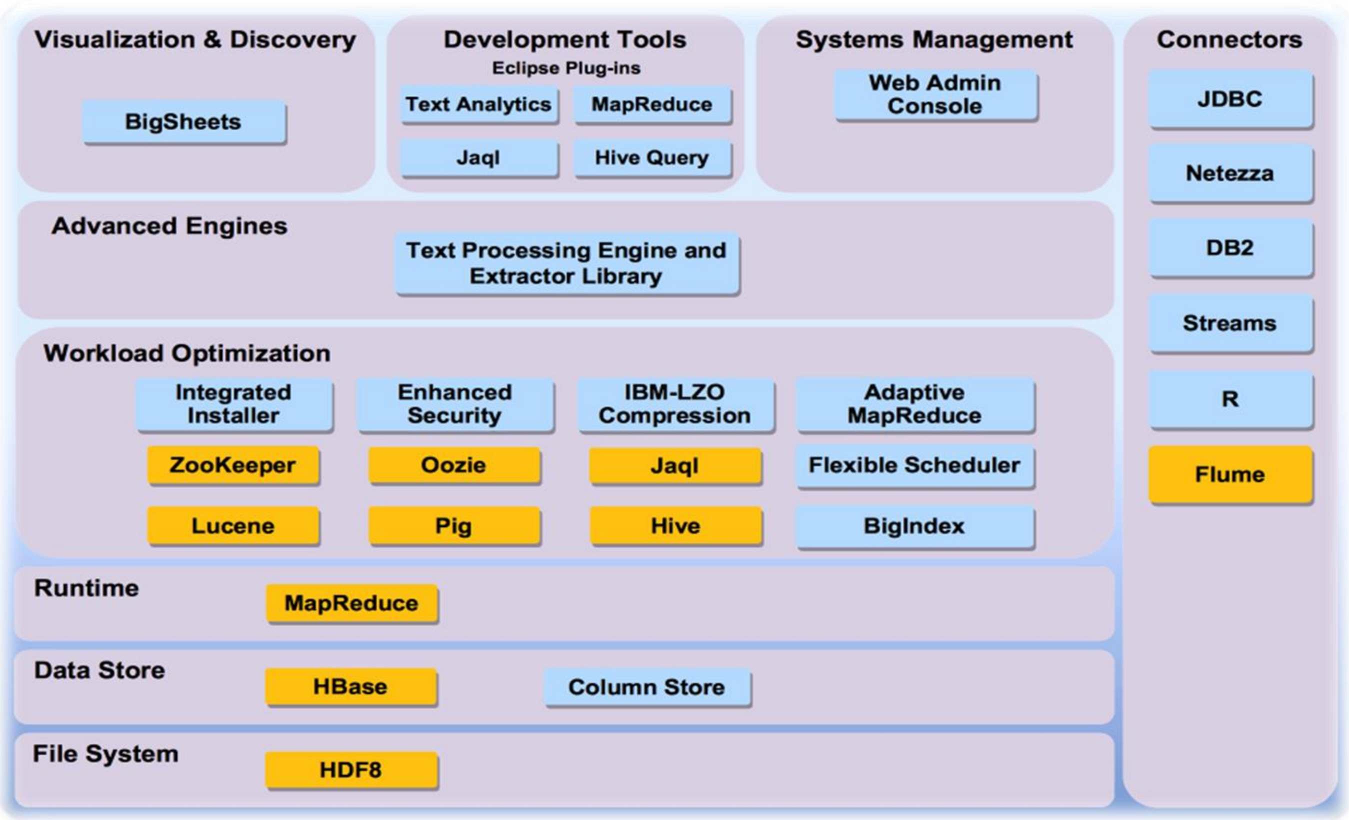
Flume

Configure, Deploy, Monitor, Manage

Command Center

🟩 Apache 🟦 Pivotal HD Added Value

IBM BigInsights



Visualization & Discovery
- BigSheets

Development Tools

Eclipse Plug-ins
- Text Analytics
- MapReduce
- Jaql
- Hive Query

Systems Management
- Web Admin Console

Connectors
- JDBC
- Netezza
- DB2
- Streams
- R
- Flume

Advanced Engines
- Text Processing Engine and Extractor Library

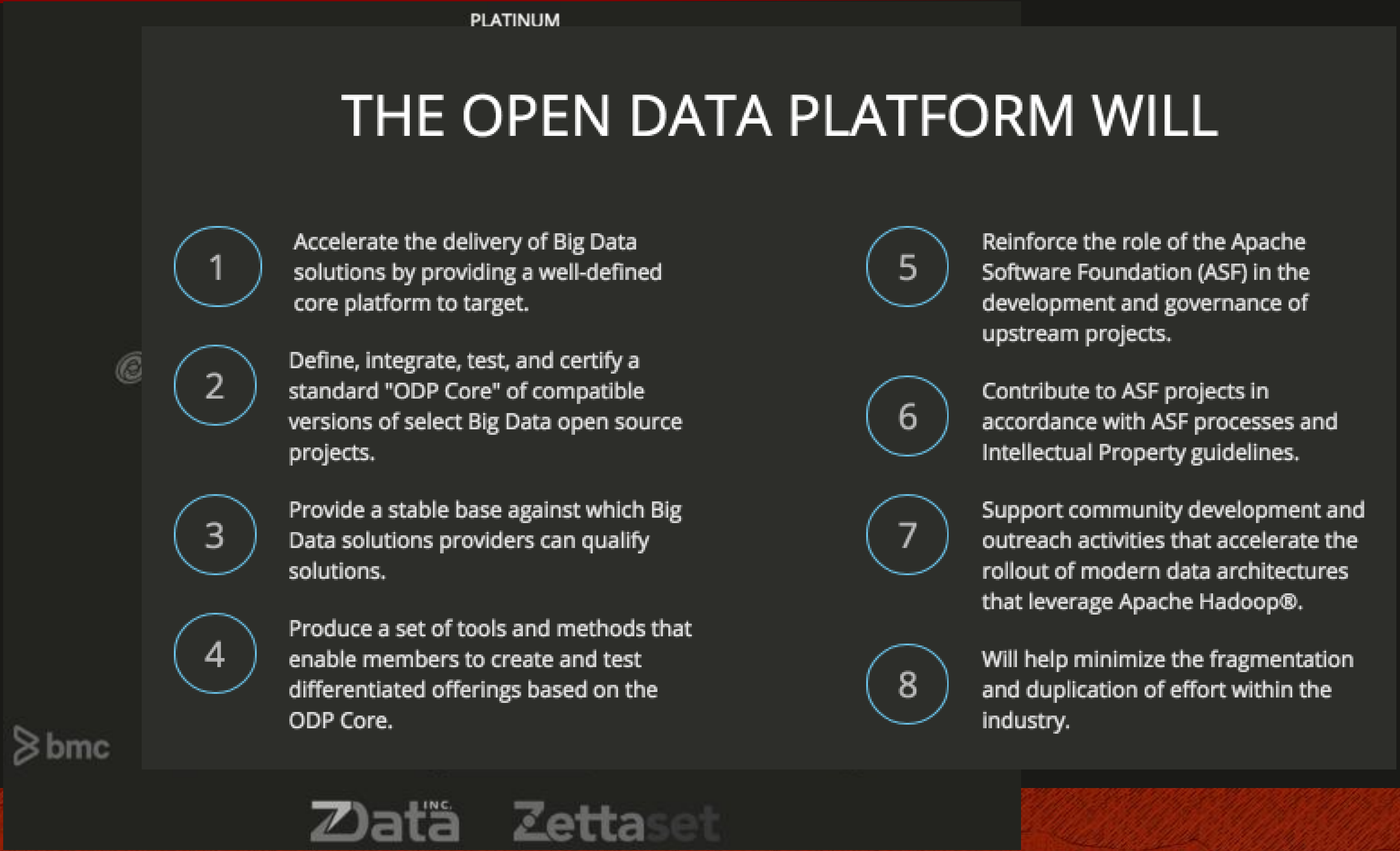
Workload Optimization
- Integrated Installer
- Enhanced Security
- IBM-LZO Compression
- Adaptive MapReduce
- ZooKeeper
- Oozie
- Jaql
- Flexible Scheduler
- Lucene
- Pig
- Hive
- BigIndex

Runtime
- MapReduce

Data Store
- HBase
- Column Store

File System
- HDF8

Open Data Platform



THE OPEN DATA PLATFORM WILL

1 Accelerate the delivery of Big Data solutions by providing a well-defined core platform to target.

2 Define, integrate, test, and certify a standard "ODP Core" of compatible versions of select Big Data open source projects.

3 Provide a stable base against which Big Data solutions providers can qualify solutions.

4 Produce a set of tools and methods that enable members to create and test differentiated offerings based on the ODP Core.

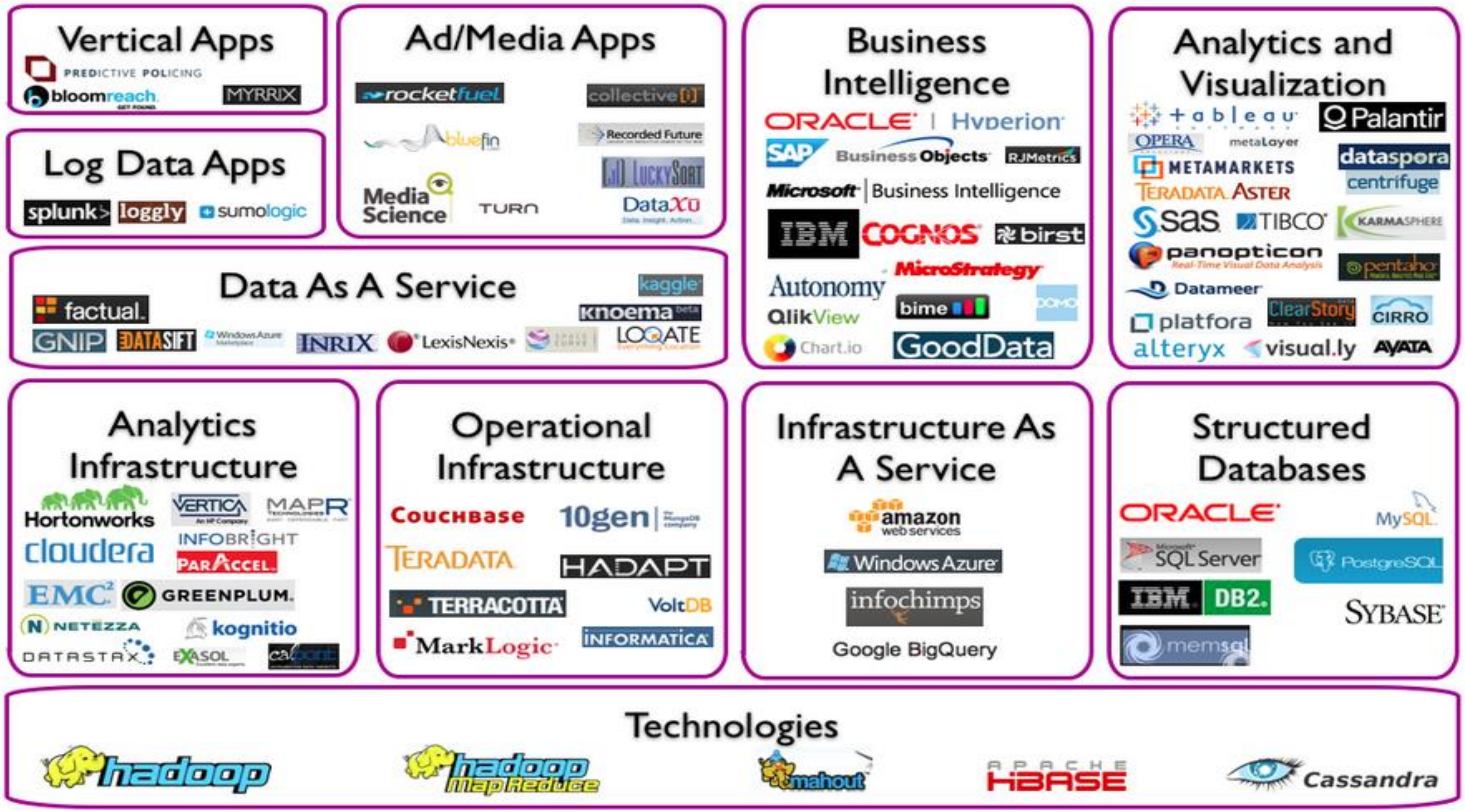
5 Reinforce the role of the Apache Software Foundation (ASF) in the development and governance of upstream projects.

6 Contribute to ASF projects in accordance with ASF processes and Intellectual Property guidelines.

7 Support community development and outreach activities that accelerate the rollout of modern data architectures that leverage Apache Hadoop®.

8 Will help minimize the fragmentation and duplication of effort within the industry.

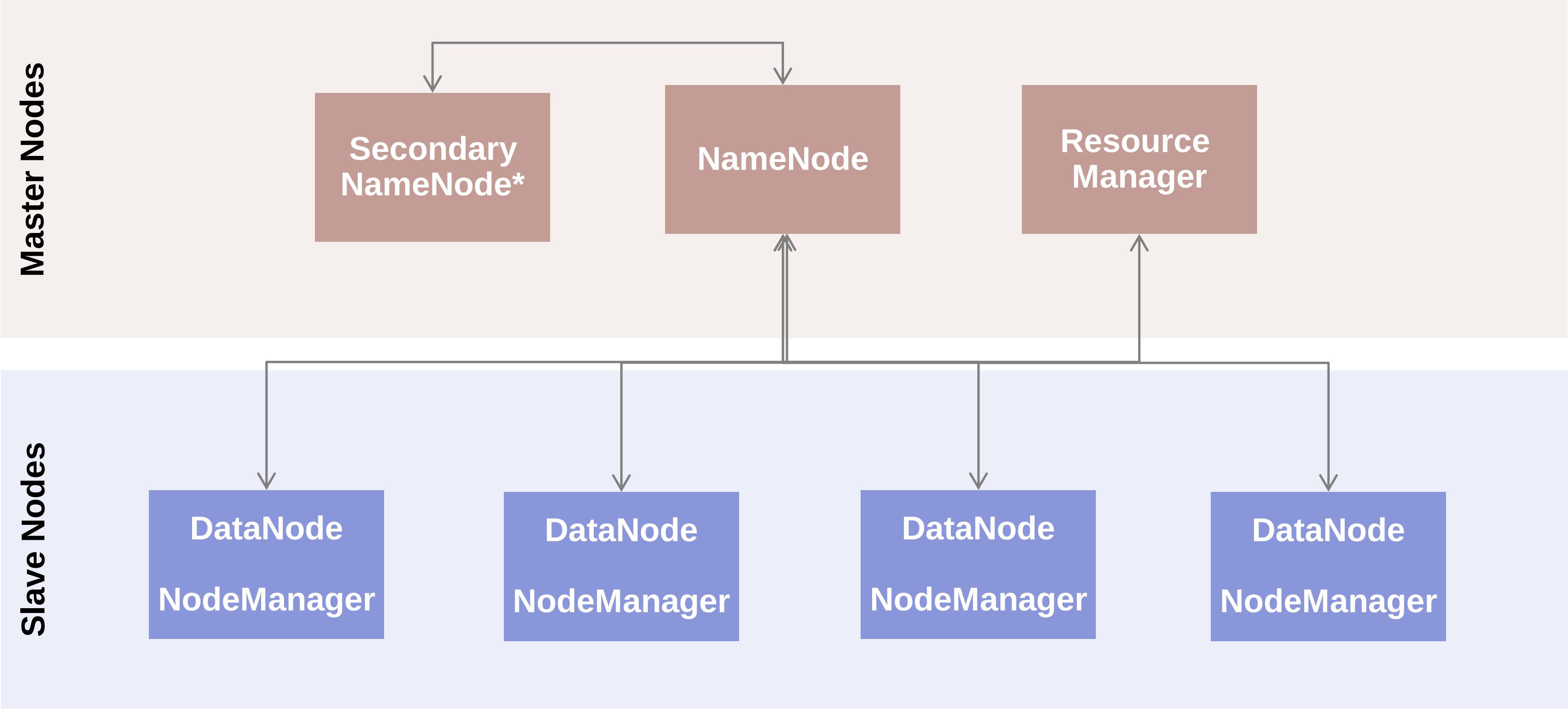
Ecosystem


Vertical Apps
PREDICTIVE POLICING
bloomreach
MYRRIX
Ad/Media Apps
rocketfuel
collective[i]
bluefin
Recorded Future
LuckySort
Media Science
TURN
DataXu
Log Data Apps
splunk>
loggly
sumologic
Business Intelligence
ORACLE | Hyperion
SAP Business Objects
RJMetrics
Microsoft | Business Intelligence
IBM COGNOS
birst
MicroStrategy
Autonomy
QlikView
bime
DOMO
Chart.io
GoodData
Analytics and Visualization
tableau
Palantir
OPERA
metaLayer
METAMARKETS
dataspora
centrifuge
TERADATA ASTER
sas
TIBCO
KARMASPHERE
panopticon
Real-Time Visual Data Analysis
pentaho
Datameer
platfora
ClearStory
CIRRO
alteryx
visual.ly
AYATA
Data As A Service
kaggle
factual.
knoema beta
GNIP
DATASIFT
Windows Azure Marketplace
INRIX
LexisNexis
LOQATE
Everything Location
Analytics Infrastructure
Hortonworks
VERTICA
An HP Company
MAPR
cloudera
INFOBRIGHT
PARACCEL
EMC²
GREENPLUM
NETEZZA
kognitio
DATASTAX
EXASOL
calpont
Operational Infrastructure
COUCHBASE
10gen | The MongoDB company
TERADATA
HADAPT
TERRACOTTA
VoltDB
MarkLogic
INFORMATICA
Infrastructure As A Service
amazon web services
Windows Azure
infochimps
Google BigQuery
Structured Databases
ORACLE
MySQL
Microsoft SQL Server
PostgreSQL
IBM DB2.
SYBASE
memsql
Technologies
hadoop
hadoop MapReduce
mahout
APACHE HBASE
Cassandra

HDFS

ARCHITECTURE DEEP-DIVE

Topology of a Hadoop Cluster



Master Nodes

Secondary NameNode*

NameNode

Resource Manager

Slave Nodes

DataNode

NodeManager

DataNode

NodeManager

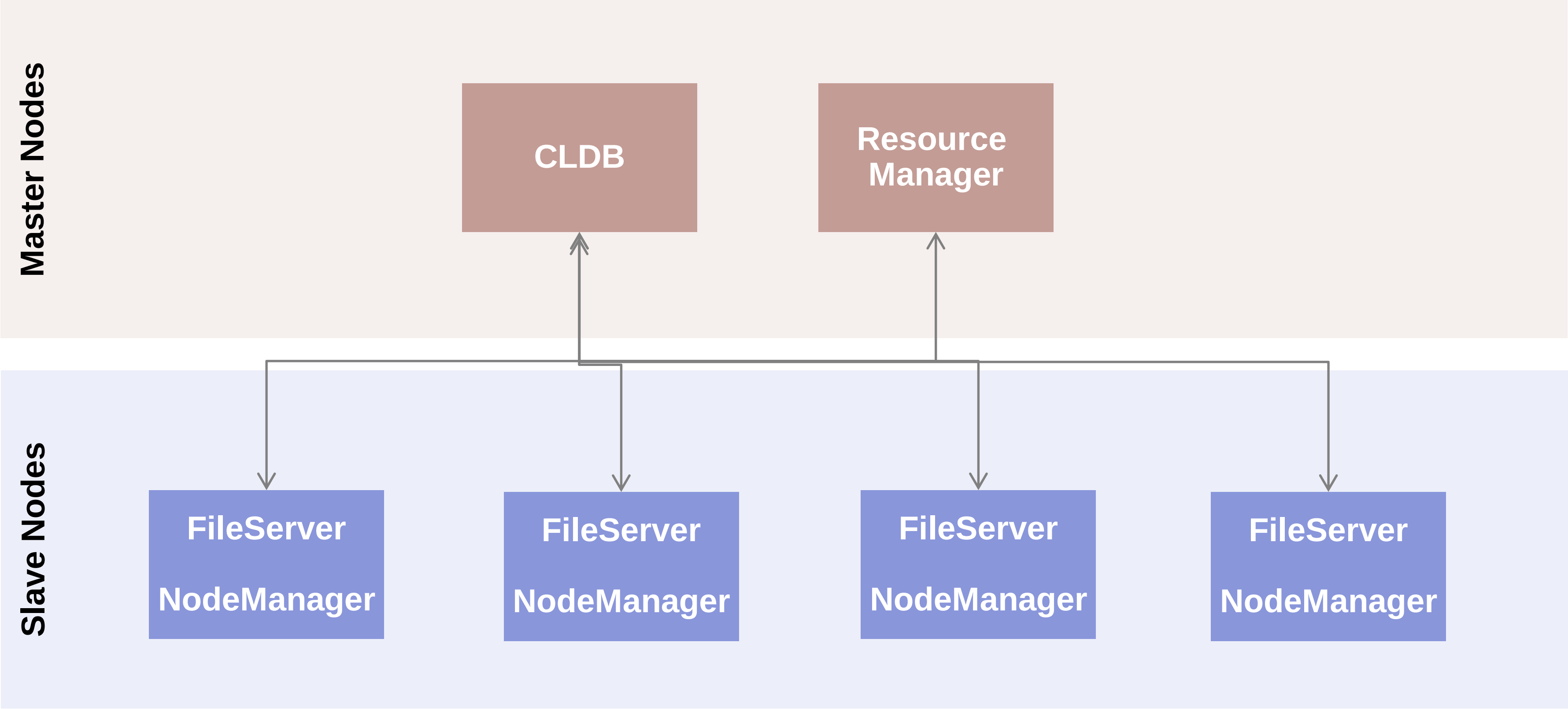
DataNode

NodeManager

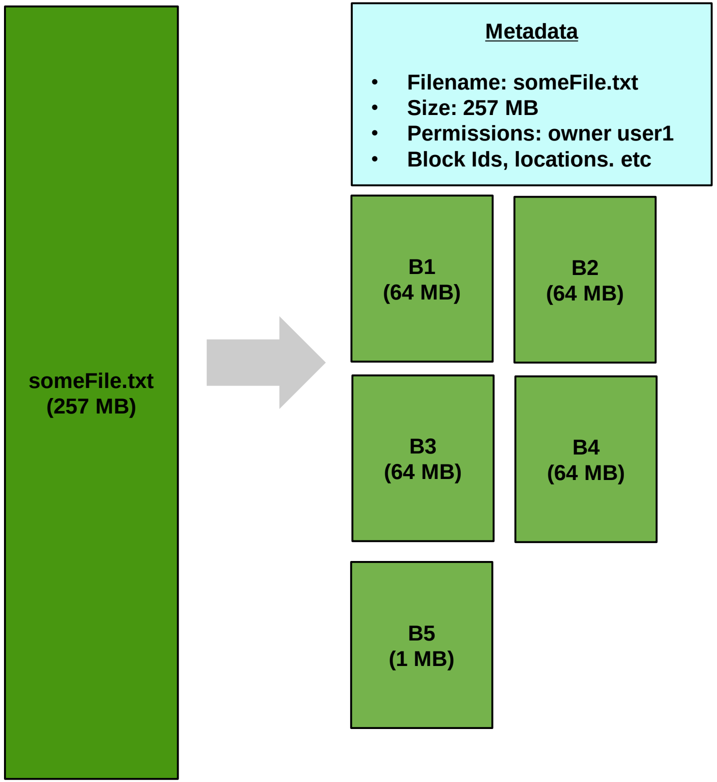
DataNode

NodeManager

Topology of a Hadoop (MapR) Cluster



HDFS: Architecture



someFile.txt
(257 MB)

Metadata

- Filename: someFile.txt
- Size: 257 MB
- Permissions: owner user1
- Block Ids, locations. etc

- File consists of Metadata and Data
- Data is broken into fixed sized **blocks**

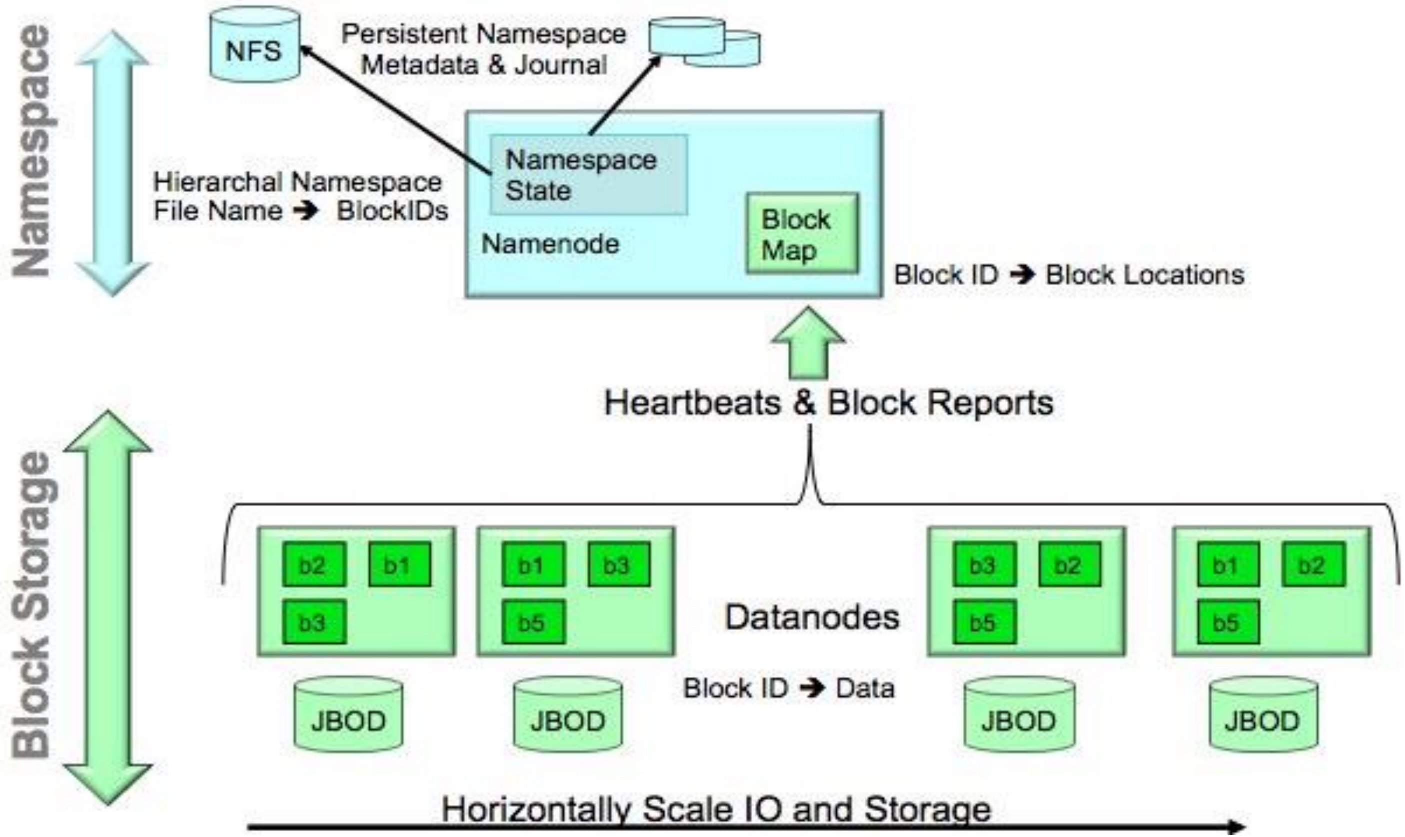
- **Blocks** stored by **DataNodes**

- DataNodes
  - Host and serve blocks
  - All operations go to NN
  - No idea of files / directory / metadata

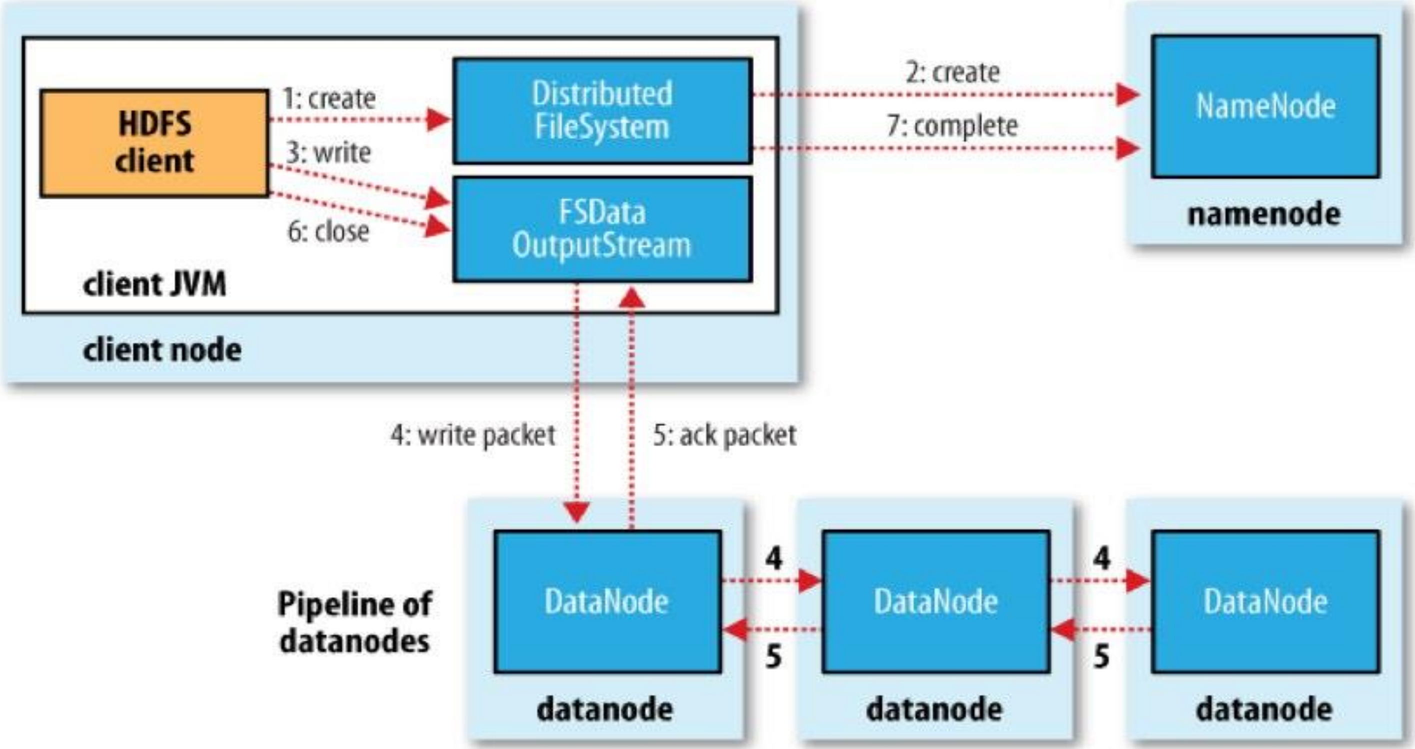
- **NameNode** holds (in memory)
  - Directory, Files Listing
  - Block replica locations

- Secondary NameNode
  - Assists NameNode

HDFS Storage Details

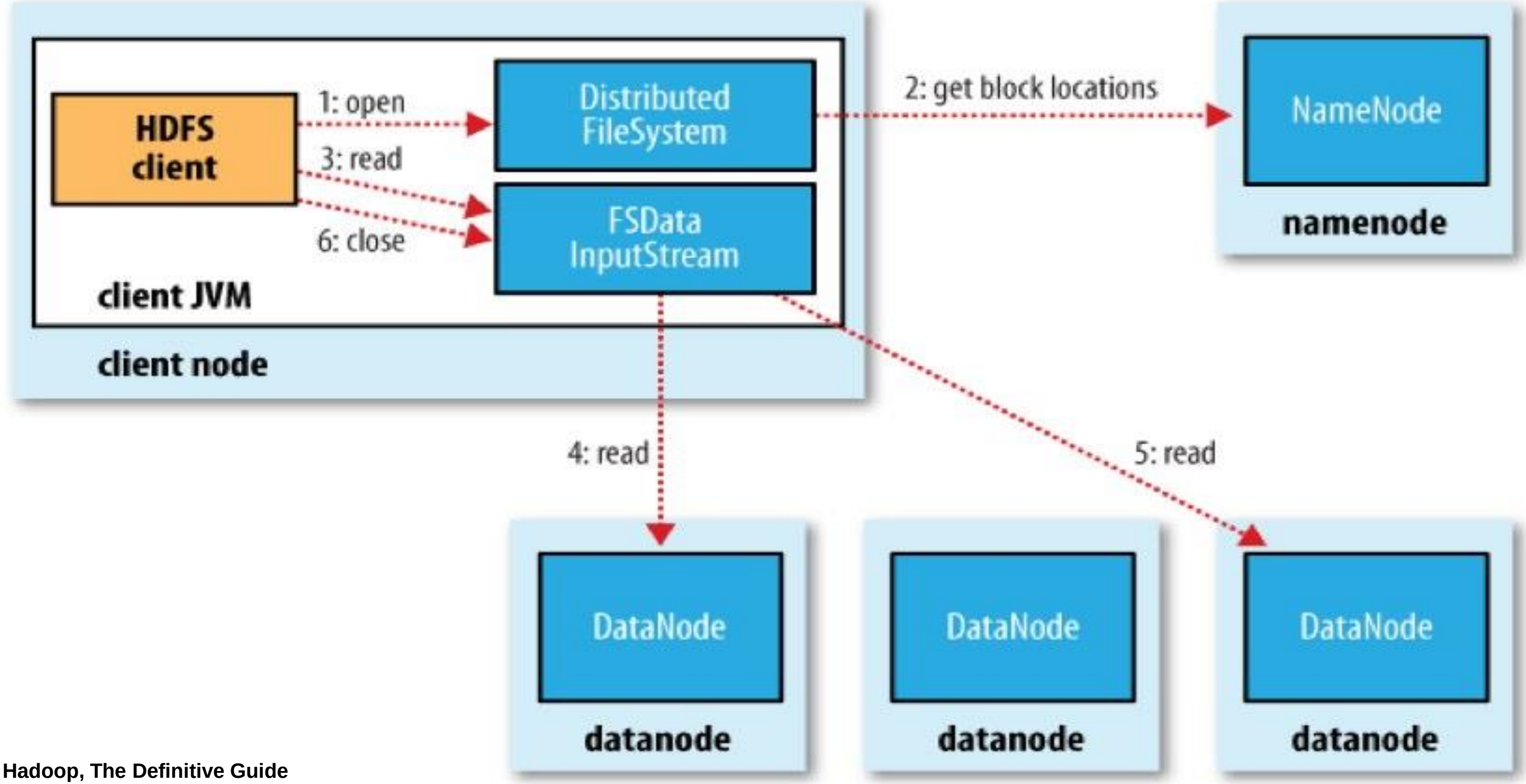


HDFS: Data Write



Source: Hadoop, The Definitive Guide

HDFS: Data Read

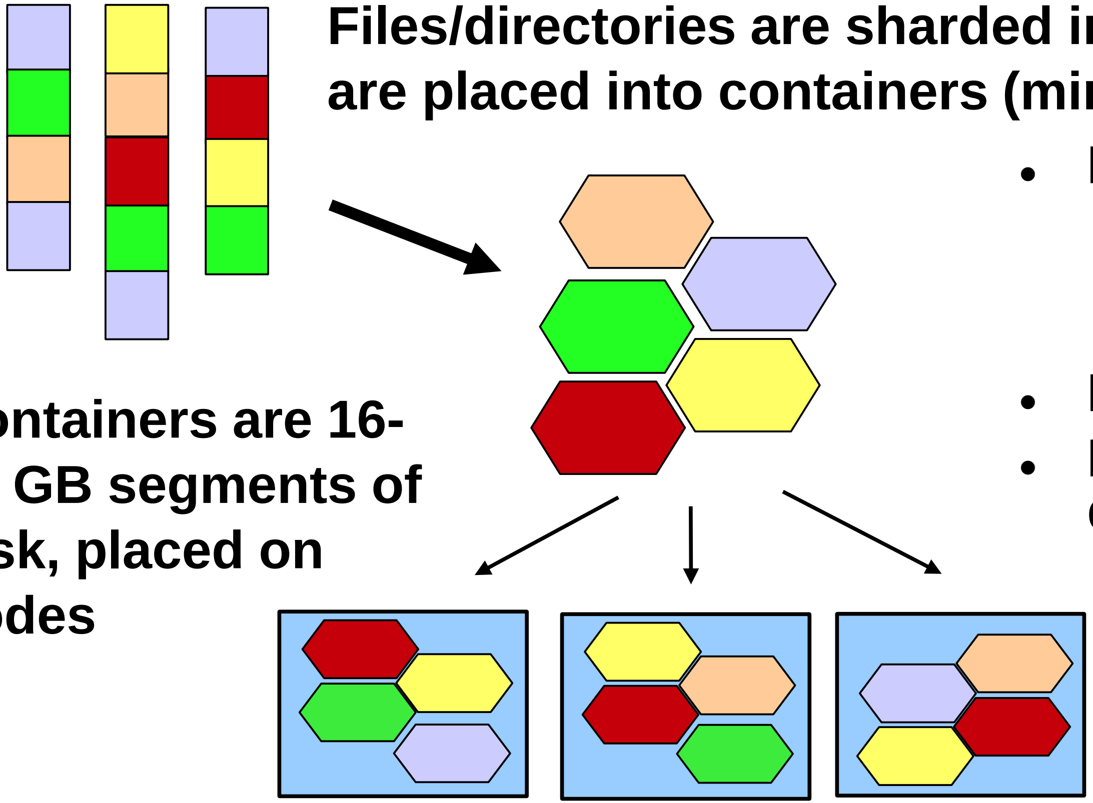


Source: Hadoop, The Definitive Guide

Alternatives to HDFS

- Any FileSystem that has implements the Apache Hadoop “FileSystem” API can interoperate with Hadoop
    - MapR File System (mapr-fs)
    - Amazon Simple Storage Service (S3)
    - Azure Blobstorage
    - Tachyon

MapR's Distributed NameNode



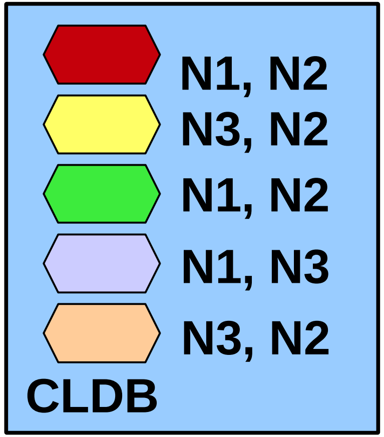
Files/directories are sharded into blocks, which are placed into containers (mini NNs) on disks

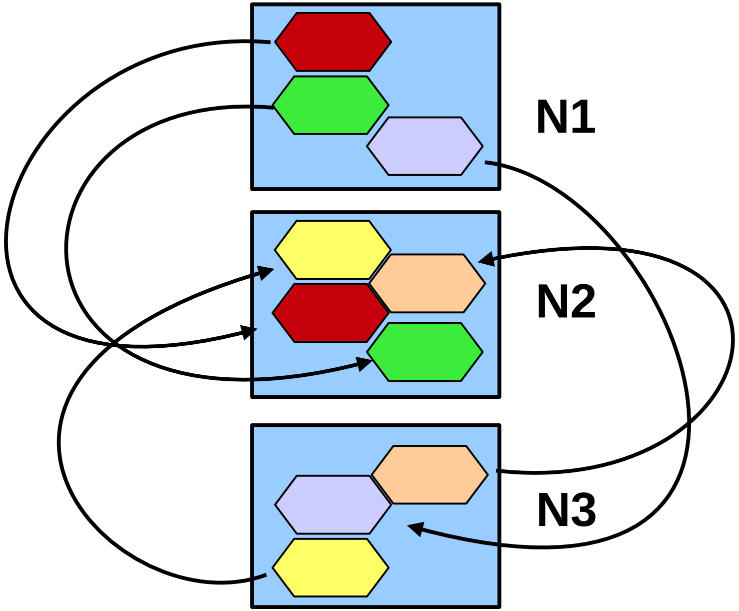
- Each container contains
  - Directories & files
  - Data blocks
- Replicated on servers
- No need to manage directly

Containers are 16-32 GB segments of disk, placed on nodes

http://www.slideshare.net/mcsrivas/design-scale-and-performance-of-maprs-distribution-for-hadoop

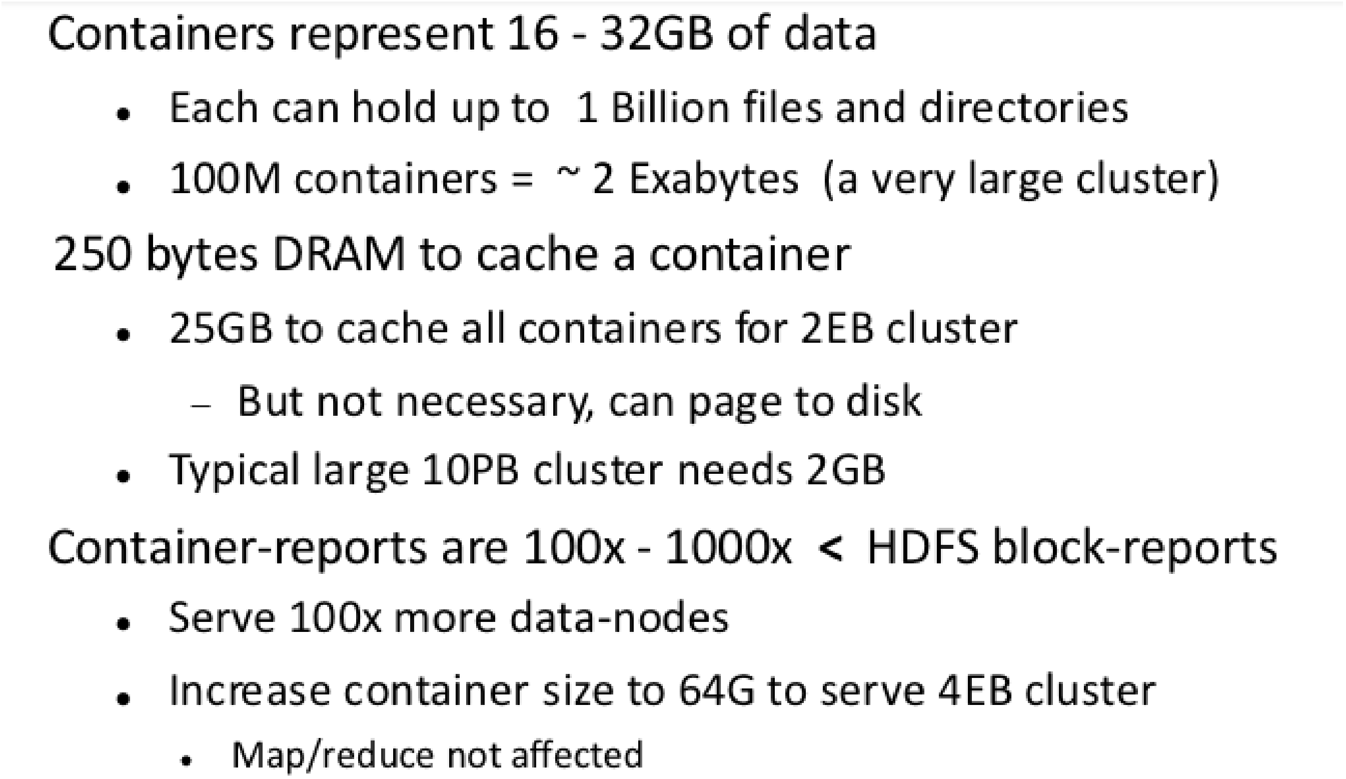
Container Location and Replication





Container location database (CLDB) keeps track of nodes hosting each container and replication chain order

MapR Distributed NameNode Scaling



Containers represent 16 - 32GB of data

- Each can hold up to 1 Billion files and directories
- 100M containers = ~ 2 Exabytes (a very large cluster)

250 bytes DRAM to cache a container

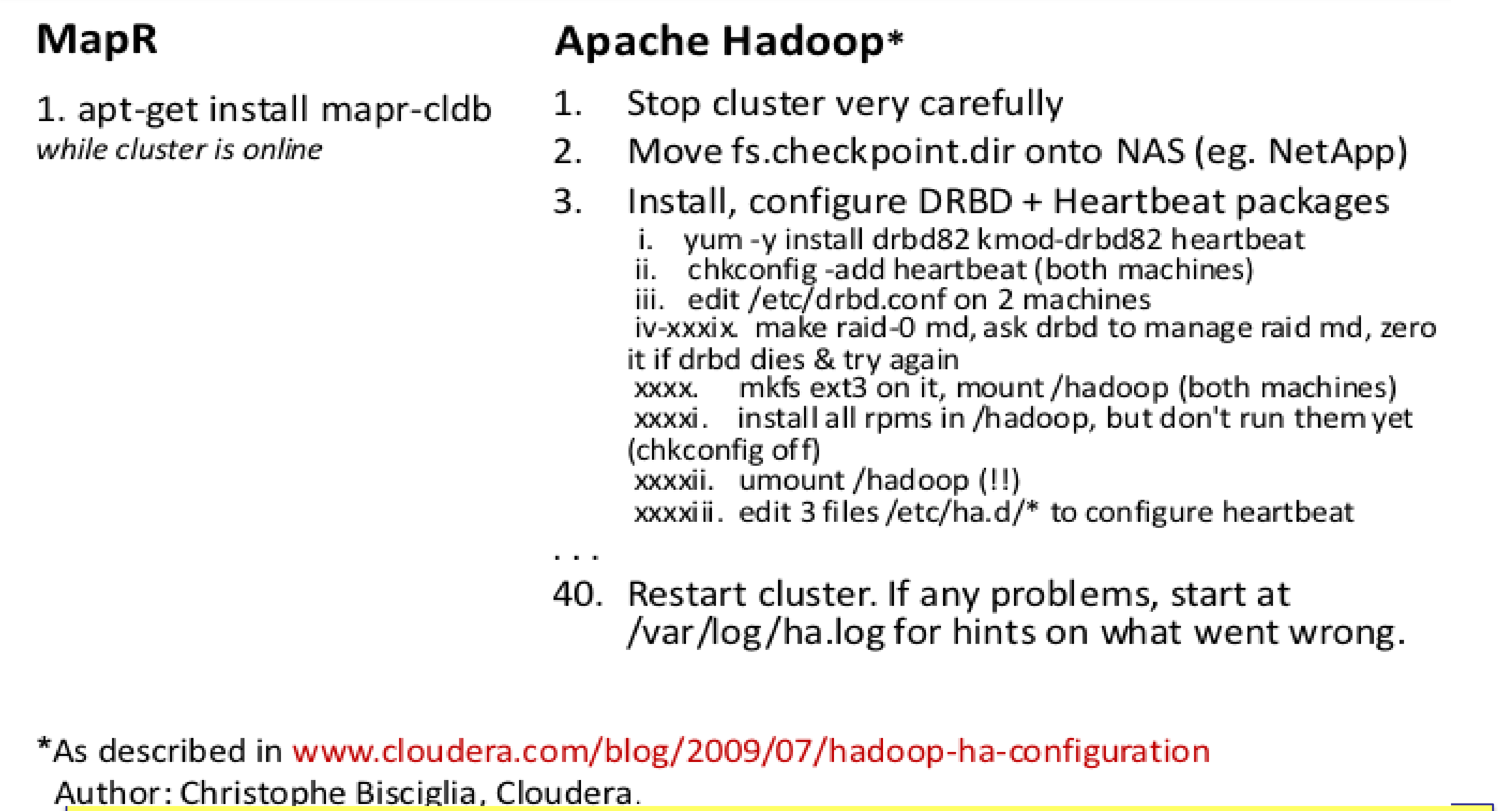
- 25GB to cache all containers for 2EB cluster
  - But not necessary, can page to disk
- Typical large 10PB cluster needs 2GB

Container-reports are 100x - 1000x < HDFS block-reports

- Serve 100x more data-nodes
- Increase container size to 64G to serve 4EB cluster
  - Map/reduce not affected

http://www.slideshare.net/mcsrivas/design-scale-and-performance-of-maprs-distribution-for-hadoop

MapR Distributed NameNode HA vs Hadoop



MapR

1. apt-get install mapr-cldb
while cluster is online

Apache Hadoop*

1. Stop cluster very carefully
2. Move fs.checkpoint.dir onto NAS (eg. NetApp)
3. Install, configure DRBD + Heartbeat packages
   i. yum -y install drbd82 kmod-drbd82 heartbeat
   ii. chkconfig -add heartbeat (both machines)
   iii. edit /etc/drbd.conf on 2 machines
   iv-xxxix. make raid-0 md, ask drbd to manage raid md, zero it if drbd dies & try again
   xxxx. mkfs ext3 on it, mount /hadoop (both machines)
   xxxxi. install all rpms in /hadoop, but don't run them yet (chkconfig off)
   xxxxii. umount /hadoop (!!)
   xxxxiii. edit 3 files /etc/ha.d/* to configure heartbeat

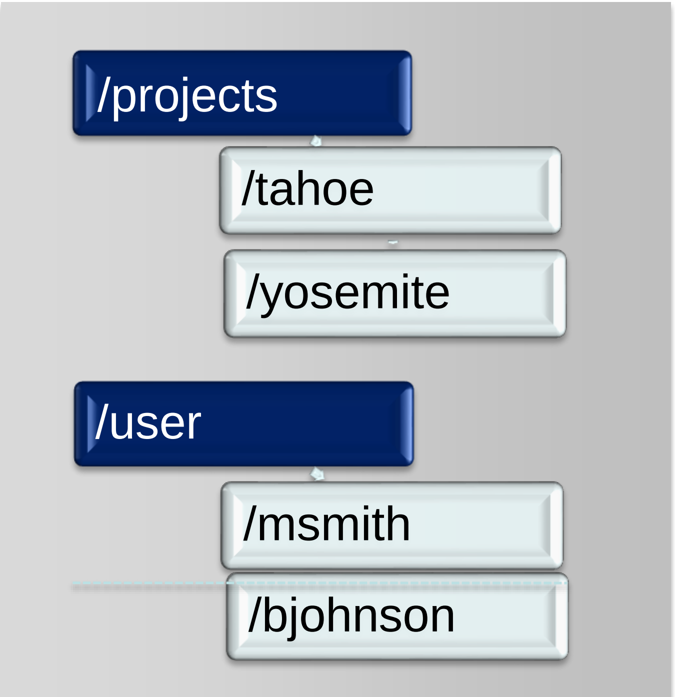
. . .

40. Restart cluster. If any problems, start at /var/log/ha.log for hints on what went wrong.

*As described in www.cloudera.com/blog/2009/07/hadoop-ha-configuration
Author: Christophe Bisciglia, Cloudera.

http://www.slideshare.net/mcsrivas/design-scale-and-performance-of-maprs-distribution-for-hadoop

MapR Volumes



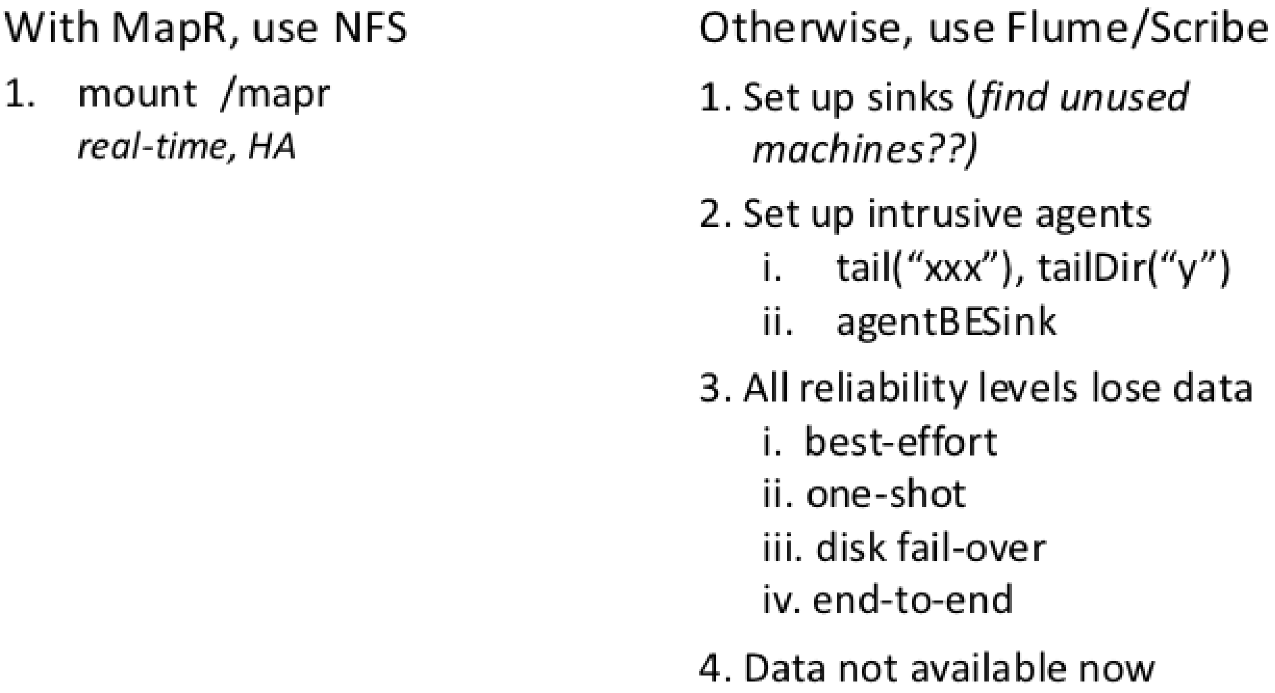
100K volumes are OK, create as many as desired!

Volumes allow management attributes to be applied in a scalable way at a very granular level and with flexibility

- **Replication factor**
- **Scheduled mirroring**
- **Scheduled snapshots**
- **Data placement control**
- **User access and tracking**
- **Administrative permissions**

http://www.slideshare.net/mcsrivas/design-scale-and-performance-of-maprs-distribution-for-hadoop

MapR NFS advantage for data import/export



With MapR, use NFS

1. mount /mapr

   real-time, HA

Otherwise, use Flume/Scribe

1. Set up sinks (*find unused machines??)*
2. Set up intrusive agents
   i. tail(“xxx”), tailDir(“y”)
   ii. agentBESink
3. All reliability levels lose data
   i. best-effort
   ii. one-shot
   iii. disk fail-over
   iv. end-to-end
4. Data not available now

http://www.slideshare.net/mcsrivas/design-scale-and-performance-of-maprs-distribution-for-hadoop

When to use HDFS

- Store large structured and unstructured files
  - Server logs
  - Relational Files
  - Data Feeds
  - Archive
  - Satellite images

When to not use HDFS

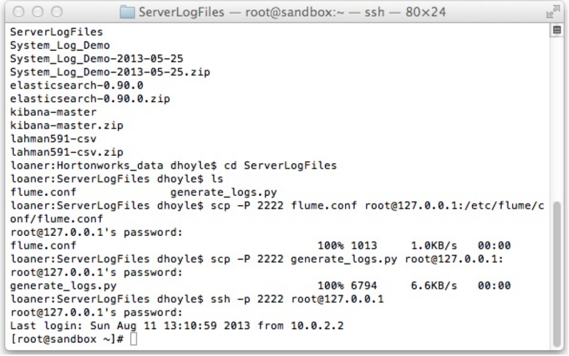
- HDFS is not suited for:
  - Large number of small files
    - ✓ Small files can always be concatenated
  - Files that are modified often
    - ✓ Create new files instead
  - Files that are randomly accessed
    - ✓ B-Trees, etc.

Accessing HDFS

- Command line tools
- Java API
- WebDFS API
- Third-party access
  - FUSE
  - Web UI
  - HDFS over FTP

HDFS Characteristics: Review

- Designed for **modest number of Large files** (millions instead of billions)
- **Sequential access** not Random access
- Write Once, Read Many
- Data **is split into chunks** and stored in multiple nodes as blocks
- **Namenode** maintains the block locations
- Blocks get **replicated** over the data nodes
- HDFS 2.x Features:
  - **High Availability** with Active and Standby NameNode
  - **Namespace Federation** for scalability
  - **Snapshots** to enable point-in-time recovery
  - **NFS** Gateway



```
ServerLogFiles
System_Log_Demo
System_Log_Demo-2013-05-25
System_Log_Demo-2013-05-25.zip
elasticsearch-0.90.0
elasticsearch-0.90.0.zip
kibana-master
kibana-master.zip
lahman591-csv
lahman591-csv.zip
loaner:Hortonworks_data dhoyle$ cd ServerLogFiles
loaner:ServerLogFiles dhoyle$ ls
flume.conf              generate_logs.py
loaner:ServerLogFiles dhoyle$ scp -P 2222 flume.conf root@127.0.0.1:/etc/flume/c
onf/flume.conf
root@127.0.0.1's password:
flume.conf                                    100% 1013     1.0KB/s   00:00
loaner:ServerLogFiles dhoyle$ scp -P 2222 generate_logs.py root@127.0.0.1:
root@127.0.0.1's password:
generate_logs.py                              100% 6794     6.6KB/s   00:00
loaner:ServerLogFiles dhoyle$ ssh -p 2222 root@127.0.0.1
root@127.0.0.1's password:
Last login: Sun Aug 11 13:10:59 2013 from 10.0.2.2
[root@sandbox ~]# 
```

HDFS

DEMO

END OF PART 1