

# Introduction to BigData – Spark – Processing

## Concrete Examples Experience at SG

Arnaud Nauwynck

Oct 2022

@Copyright All rights reserved

# Objectives

4 x 2 hours of lessons  
4 x 2 hours of Hands-On

## Objective Lesson 1 : Overview

What is BigData ?

What is Spark ?

What is Data processing ?

... concrete examples from experience in a company

## Objective Lesson 2,3,4 : Deep Dive in Spark

# Outline

- What is BigData ?
  - Order of Magnitudes for « Big »
  - Distributed System challenges
  - Scaling Storage Resources: Hdfs
  - Scaling Compute Resources: Yarn, Kubernetes
  - Using Storage – Compute apis
  - Evolution of Hardware – Softwares – Hadoop Ecosystem, Spark
- Description of a Datalake
  - Content
  - Usages
- Example of Data-Processing
  - RAW to LAKE, Reports
- Evolution to Cloud

# About Me

← **Arnaud Nauwynck**  
2 Tweets



NOT using Twitter/LinkedIn/FaceBook/

Only [arnaud.nauwynck@gmail.com](mailto:arnaud.nauwynck@gmail.com)

IT Passionate since 26 years

Github repos

<https://github.com/Arnaud-Nauwynck>

<http://arnaud-nauwynck.github.io/>

Working at Société Générale (La Defense)

In BigData team since 6 years

Expert Java + BigData + more





## DISCLAIMER

I do not know everything. I am biased with what I know.  
My opinions do not represent any company

# About You ... Quick Survey ?

Raise your Hands...

You are developper ?

In Java ? SQL ?

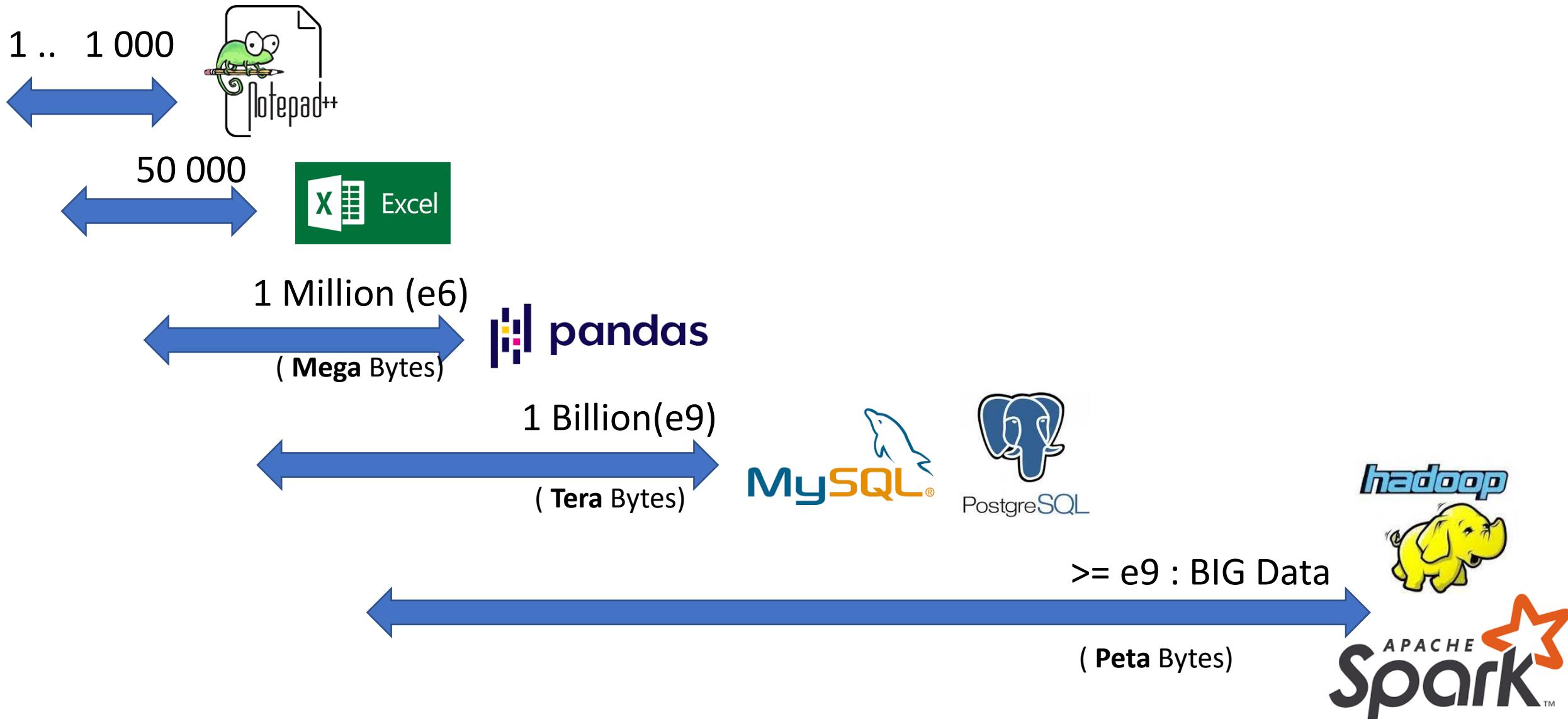
You know Hadoop / Hdfs / Spark ?

You want to be Data Engineer / Data Scientist ?

# Outline

- What is BigData ?
- **Order of Magnitudes for « Big »**
  - Distributed System challenges
  - Scaling Storage Resources: Hdfs
  - Scaling Compute Resources: Yarn, Kubernetes
  - Using Storage – Compute apis
  - Evolution of Hardware – Softwares – Hadoop Ecosystem, Spark
- Description of a Datalake
  - Content
  - Usages
- Example of Data-Processing
  - RAW to LAKE, Reports
- Evolution to Cloud

# How Big ? => Then Which Tool



# Compute Resources

## Vertical Scaling

Historically:

have Bigger needs => buy BIGGER Server

Very Frequent in Bank OLTP Databases

128 cores x 500Go RAM + SAN Bay of Disks

= Millions of Euros

.. BUT price is exponential to performance, and limited

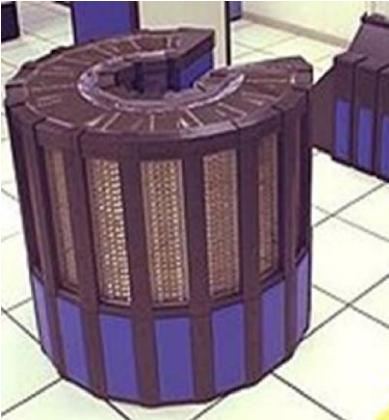
# BigData definition

Data does not fit on 1 server ... even Huge, because of

3 V = Volume / Velocity / Variety

# Bigger Hardware, different Software

## Vertical Scaling -> Horizontal



1 OS, 1 process, N Threads



Distributed computing  
... Softwares need  
Network, Fault Tolerant



# BigData At Scale

## Google Datacenter



# Not Only Gafas

Example of Smaller Companies, obviously doing BigData

Criteo (internet advertisement)

Cern (particule accelerators)

Gouvernment

Social Networks

Auchan, Carrefour, .. (E-Commerce)

MeteoFrance ( Computing, Satelite Images)

Uber, Taxis, Cars ( IOT )

Medical, Pharmaceutic, Genetics

Industries

Banks

..

# BigData In Banks ?



Corporate Investment Bank  
(for private / owned trading )



Example of Société Générale:  
At La Défense



Network Bank (for everybody customers)



Example of Société Générale:  
At Val de Fontenay

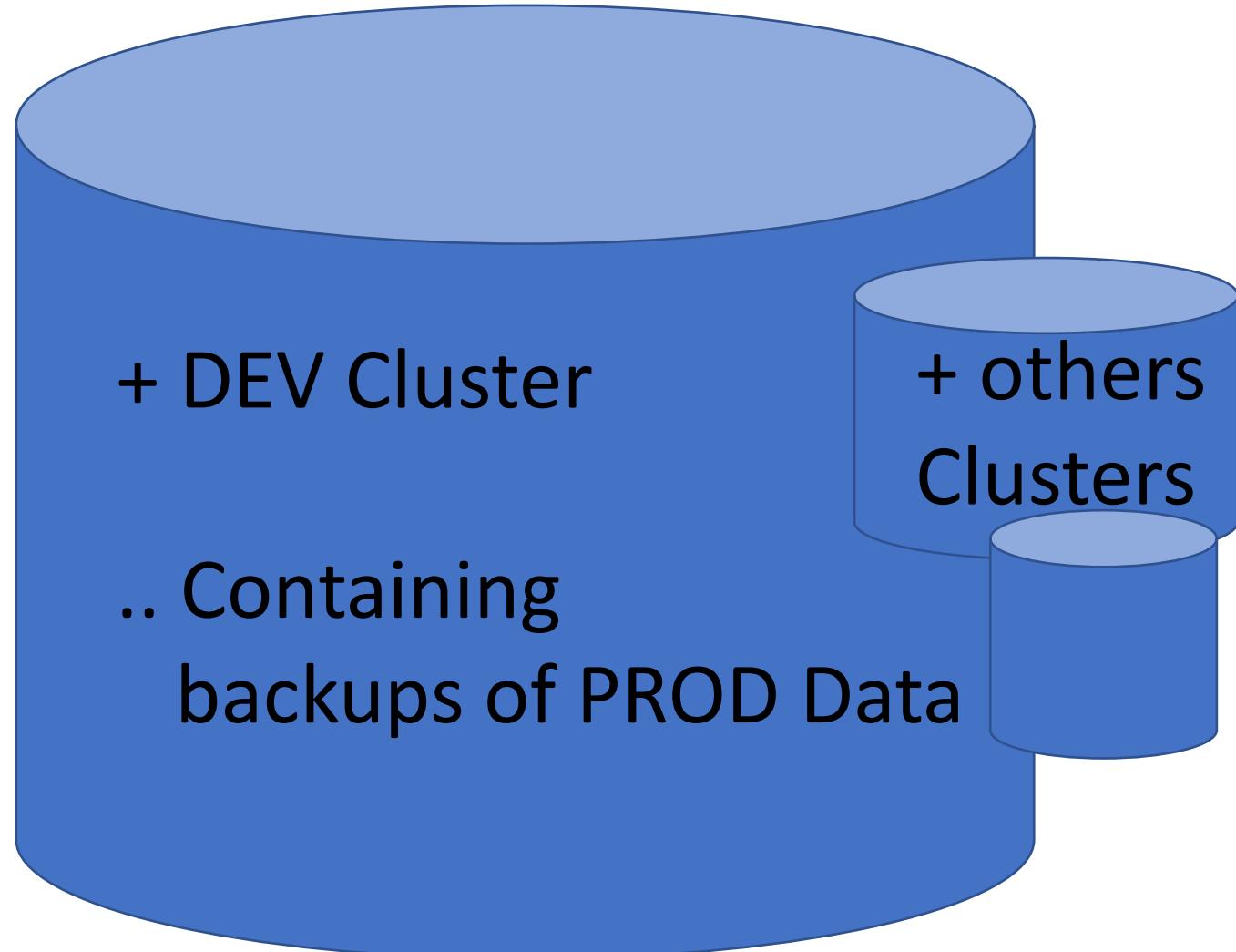
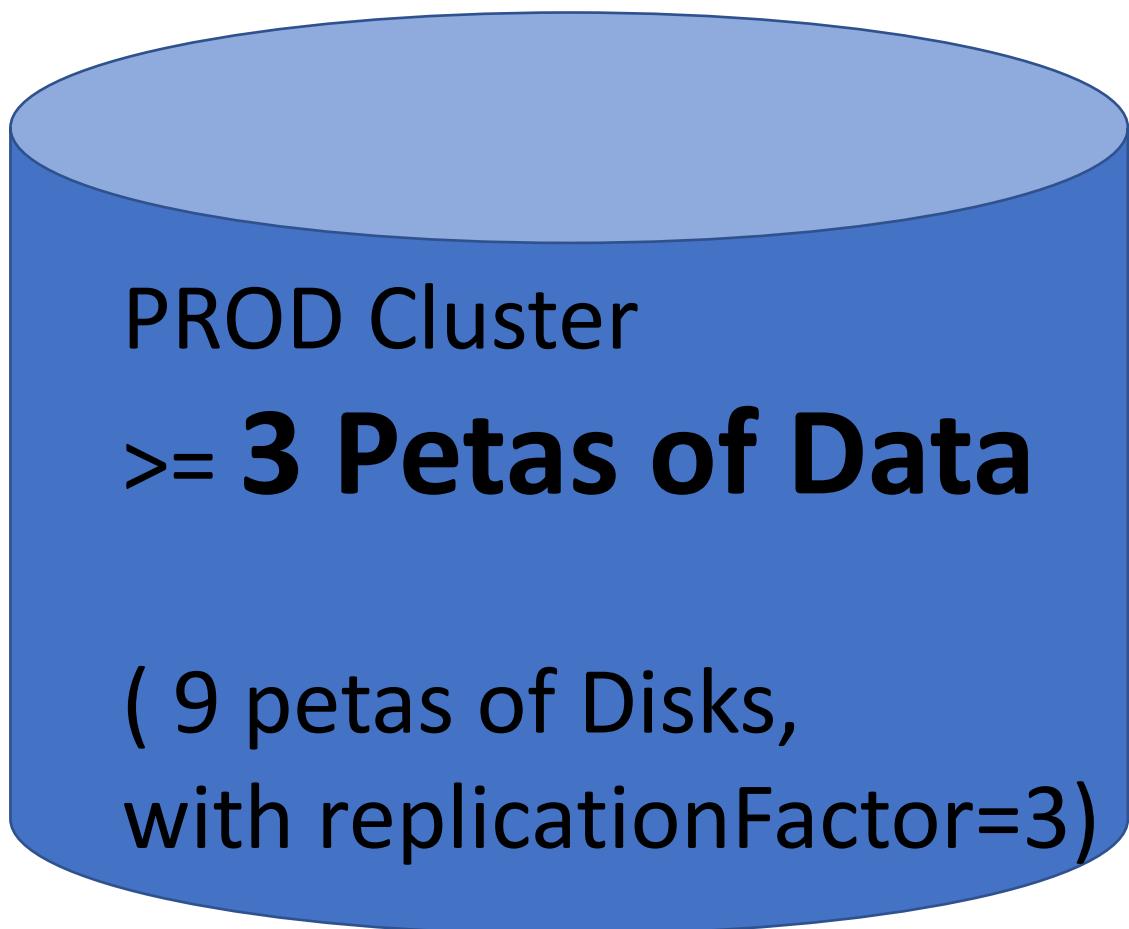
## Data for

- Historical Market data
- Risk / Var management
- Aggregation of all Trading departments
- Regulatory Reporting
- Business analysis
- ..

## Data for

- RGPD... warn personnal data!
- Anti Fraud detection,
- customers-oriented
- Business analysis
- ..

# BigData at SG (Investment Bank)



Typical Datacenter Hardware  
= N Racks x 10 blades (1U / 2U)



# 1U Server for Datalake



1 server = **42 cores** + **256 Go RAM** + **network 1Gb/s** + **8 disks**



# Trend.. Splitting Compute and Storage Resources Virtualizing (VMs,Kubernetes,Cloud) for Elasticity & Costs

For Hardware historical performance  
Compute = linked to Storage



1 Fixed « **Compute** » + local « **Storage** » Resource

**Compute ONLY** Virtual Resource

Provision **on-demand Cluster, VMs in cluster**  
Choose RAM (compromise perf/cost) per VM  
Dedicated for app workloads



**Storage ONLY** Virtual Resource

Elastic storage, pay as-you-go  
Dedicated hardwares for data density



# Data Density



48 \* HDD disks in 2U blade



32 \* SSD (E.1 EDSFF) in 1U blade  
... 500 Tera / 1 Peta !! ... 200 000 euros

# On-Premise « Small » Datalake ?

100 servers for PROD  
( + 120 servers for DEV )

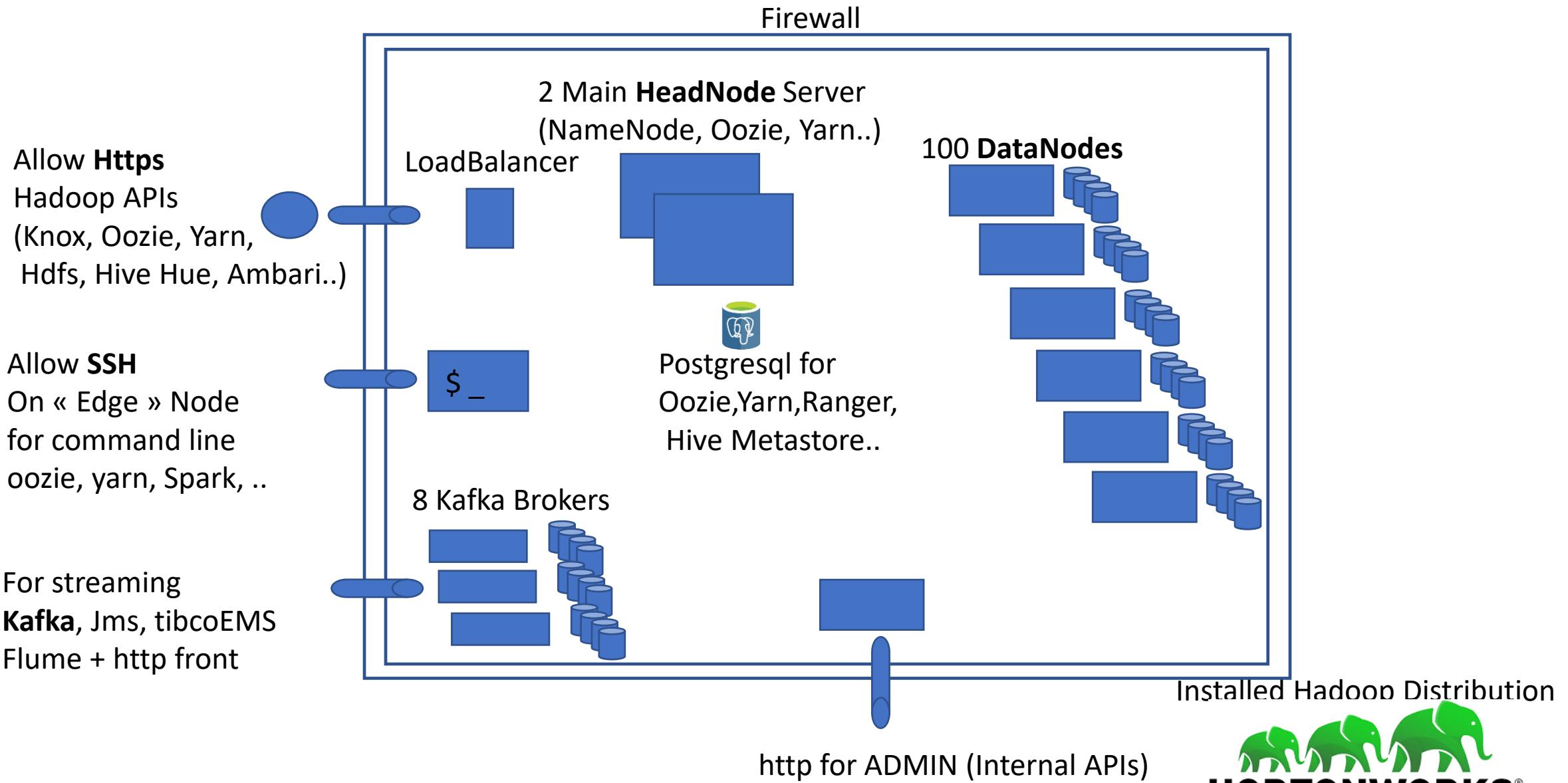
= 4200 cores

+ 25600 Go RAM : 25 Tera of RAMS

+ 800 disks : 2 Petas of Data ( 6 peta of Disks )

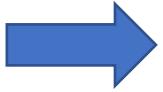
... now 3 Petas on Azure

# Hadoop Cluster Architecture



# Outline

- What is BigData ?
  - Order of Magnitudes for « Big »
- **Distributed System challenges**
  - Scaling Storage Resources: Hdfs
  - Scaling Compute Resources: Yarn, Kubernetes
  - Using Storage – Compute apis
  - Evolution of Hardware – Softwares – Hadoop Ecosystem, Spark
- Description of a Datalake
  - Content
  - Usages
- Example of Data-Processing
  - RAW to LAKE, Reports
- Evolution to Cloud

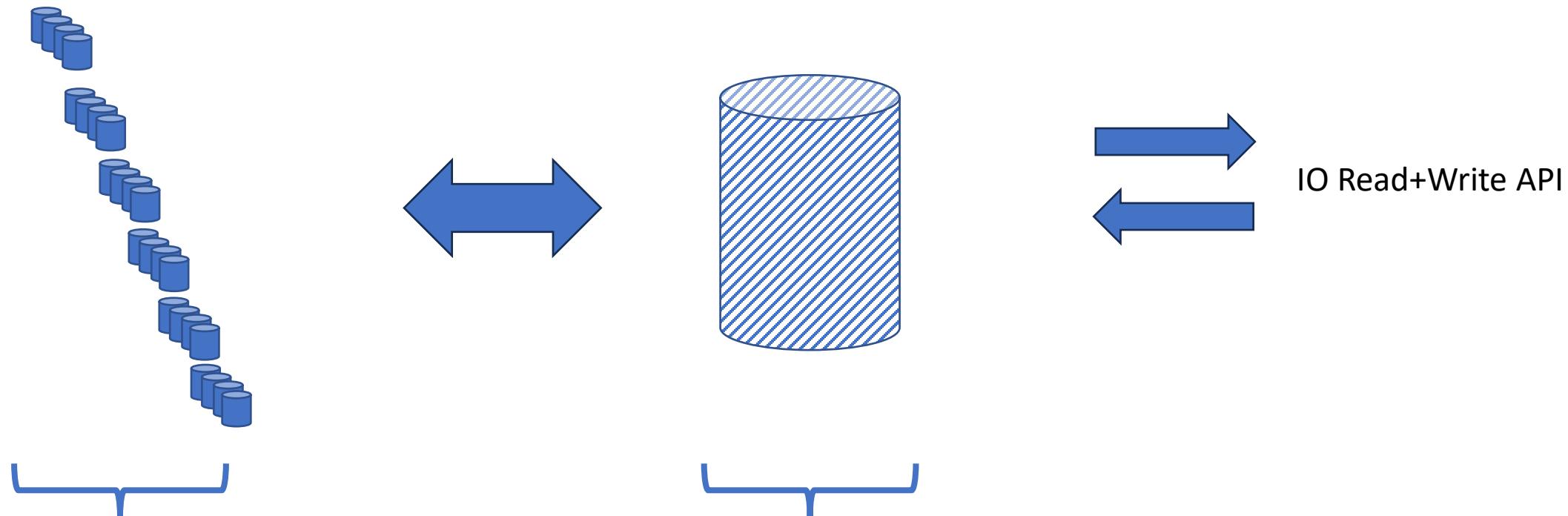


Handling 1 Server + 1 Disk  
... OK with SSH,Bash,Java...

How to handle 300 Servers ?  
1 Billions Files ? In 50 Millions Directories ?

... At Scale  
(performance, and way of working in IT teams)

# Challenge #1: Unify « Virtual » Storage Resources



Physical « **Storage** » Resources  
= 800 HDD Disks  
(on 100 Linux servers + network)

1 Virtual (Abstract) « **Storage** » Resource  
= HDFS  
Hadoop Distributed File System

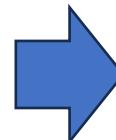
# Mind Shift .. Datalake FileSystem Abstractions

Instead of  
Handling remote SSH / SCP  
and naming server & dirs ...

Do this:  
Servers are invisible to users  
User see virtual HDFS Dirs & Files

```
# SSH for Remote command
$ ssh server-abc-for-dept-xyz
$ cd someDir
$ mkdir .. ls ... cp ... launch java
$ scp result.file another-server:
```

```
# SCP for Remote Copy to Local
$ scp server-abc-for-dept-xyz:result.file local.file
```



```
# execute commands from any client
$ hdfs dfs -ls /datalake/a/b
$ hdfs dfs mkdir /datalake/a/b/c
$ hdfs dfs rm /datalake/a/b/d

# Upload local file to Remote
$ hdfs dfs put ./local.file /datalake/a/file

# Download Remote File to local
$ hdfs dfs get /datalake/a/file ./local.file
```

# Challenges of Distributed System

How to Handle ?  
Failures,  
Resilience,  
Horizontal Scalling,  
Availability,  
Consistency,  
Replication,

...

Sh\*ts Happens (Hardware Failures, Software, Electric Power, Network, Data Corruption ..)

Example for **1 Disk**

MTBF = Mean Time Between Failure = **~25 years**

... FINE ! .... Wait ! Fine ????

for a big cluster 20 000 servers x 8 Disks => ??



Big Data = Handle « Normal » (recurrent) Failures !

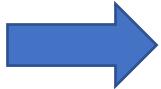
1 Disk:  $25 \text{ Y} = 25 * 365 * 24 \text{ hours} = 219k \text{ hours}$  ... 1 Cluster:  $20000 * 8 = 160k$   
=> In average : **1 failure every 1h30mn ... 17 per days !!**

Failure is **NOT exceptional**, it is the « normal » way

System must (try to) handle it **TRANSPARENTLY**

# Outline

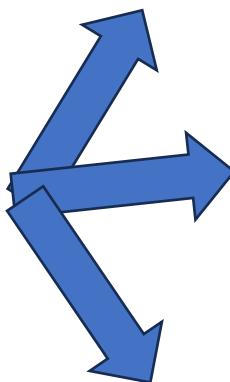
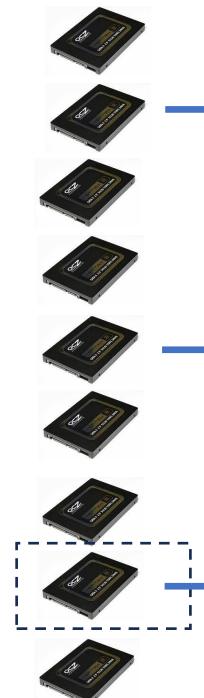
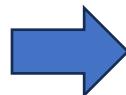
- What is BigData ?
  - Order of Magnitudes for « Big »
  - Distributed System challenges
- **Scaling Storage Resources: Hdfs**
  - Scaling Compute Resources: Yarn, Kubernetes
  - Using Storage – Compute apis
  - Evolution of Hardware – Softwares – Hadoop Ecosystem, Spark
- Description of a Datalake
  - Content
  - Usages
- Example of Data-Processing
  - RAW to LAKE, Reports
- Evolution to Cloud



# How to Handle Disk Crash: Obviously Replication Copies, But in XA & Real-Time

**RF = Replication Factor = 3 (or more!)**

End-user operation =  
Write to a « virtual File »



Then Dispatch to  
the current « Master »  
Server for this group



Then Immediate COPY  
... hopefully Transactionally  
To current « follower » servers

# The Devil hides in the Details ...

the « master » server can not reach other for copying ? Or takes too long

Files will become « Under-replicated », after a server crash (or switch Off)  
... « Over-replicated » when the server rejoin cluster (switch ON)

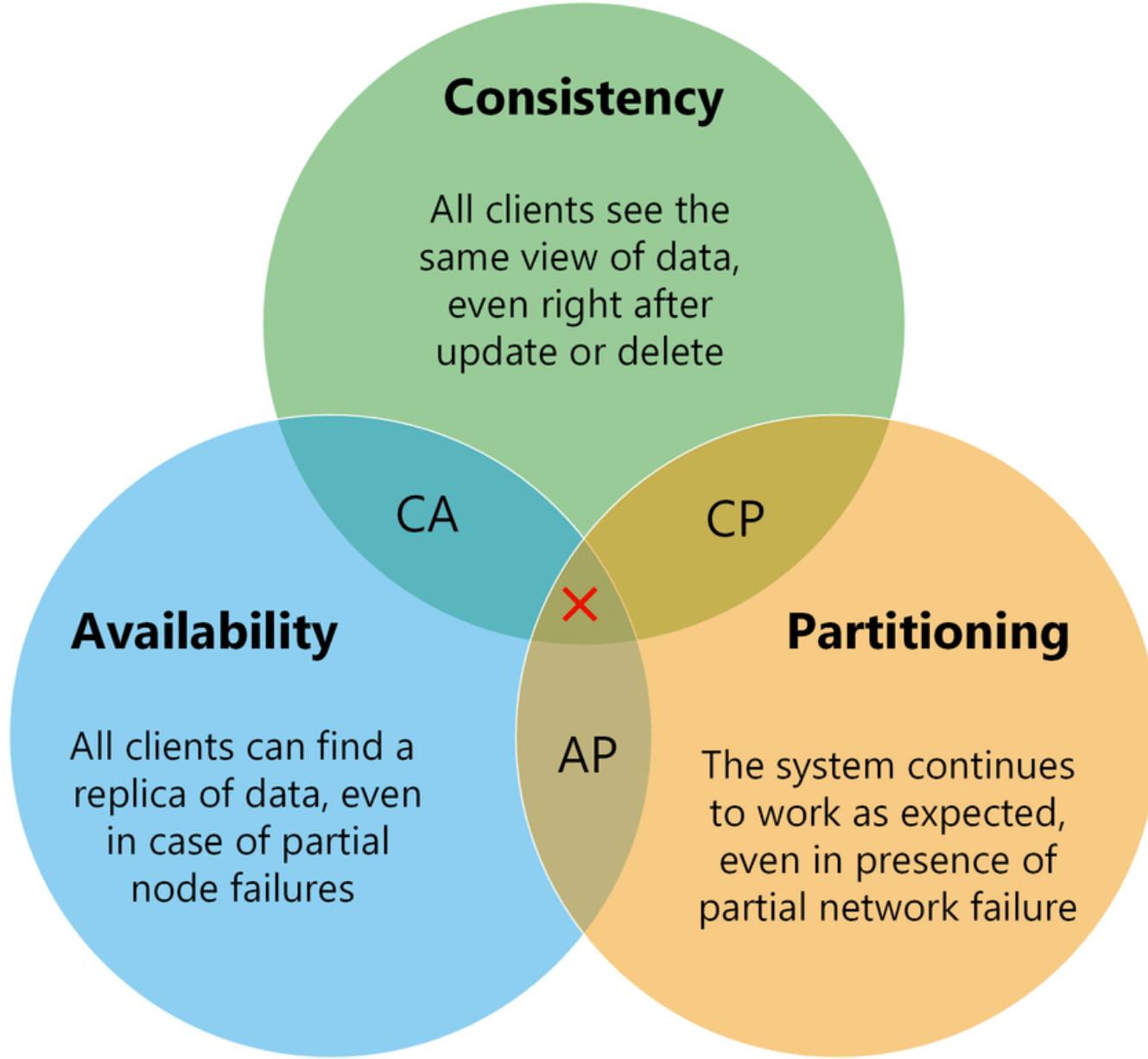
2 servers pretend to be « master » at same time (network partition) ?

Reconcile 2 conflicting (diverged) servers data ?

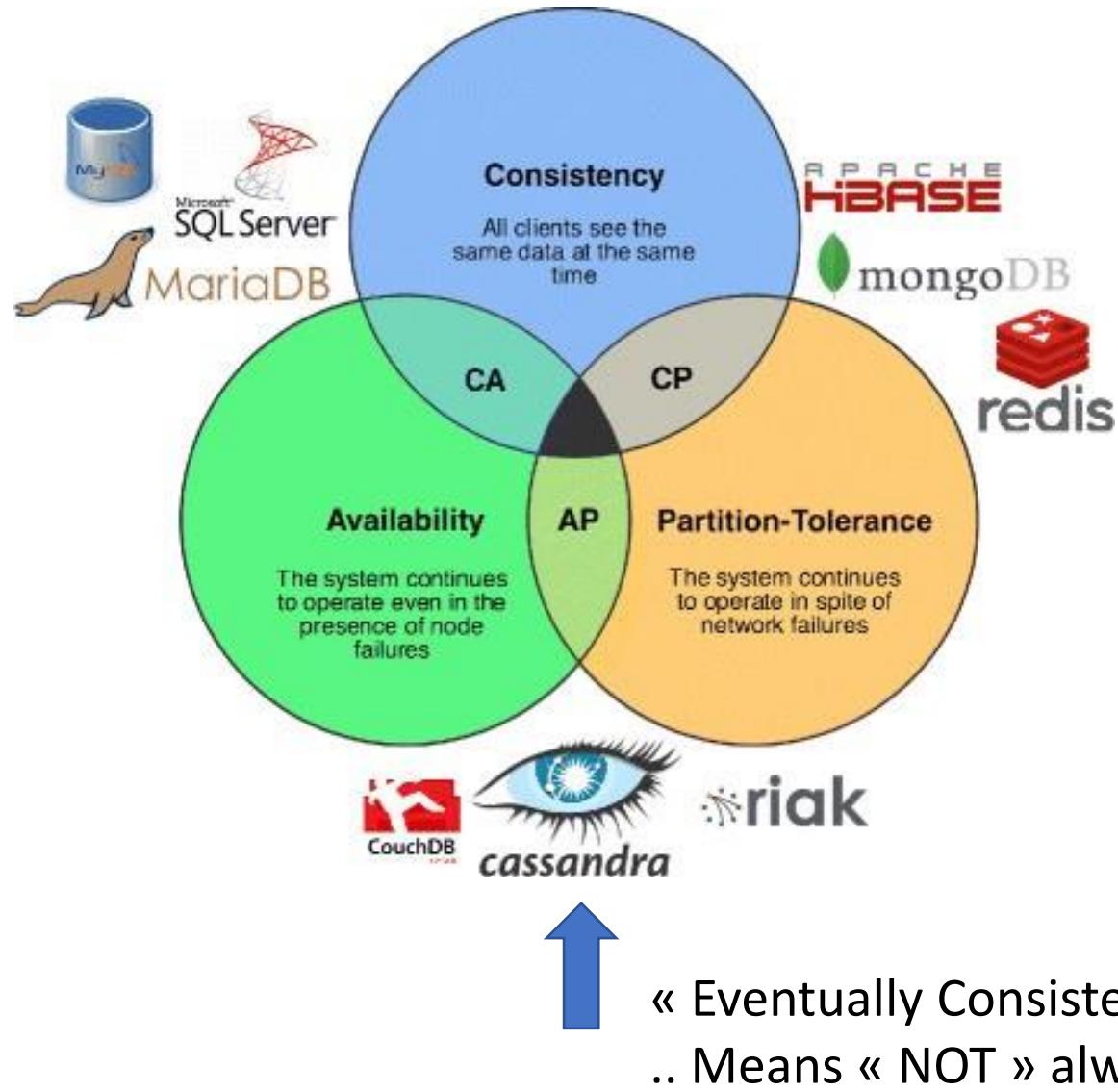
SPOF : Single Point Of Failures ... all servers depends of same Electric Power supply

Etc, Etc, Etc ...

# CAP Theorem .. Choose 2 out of 3



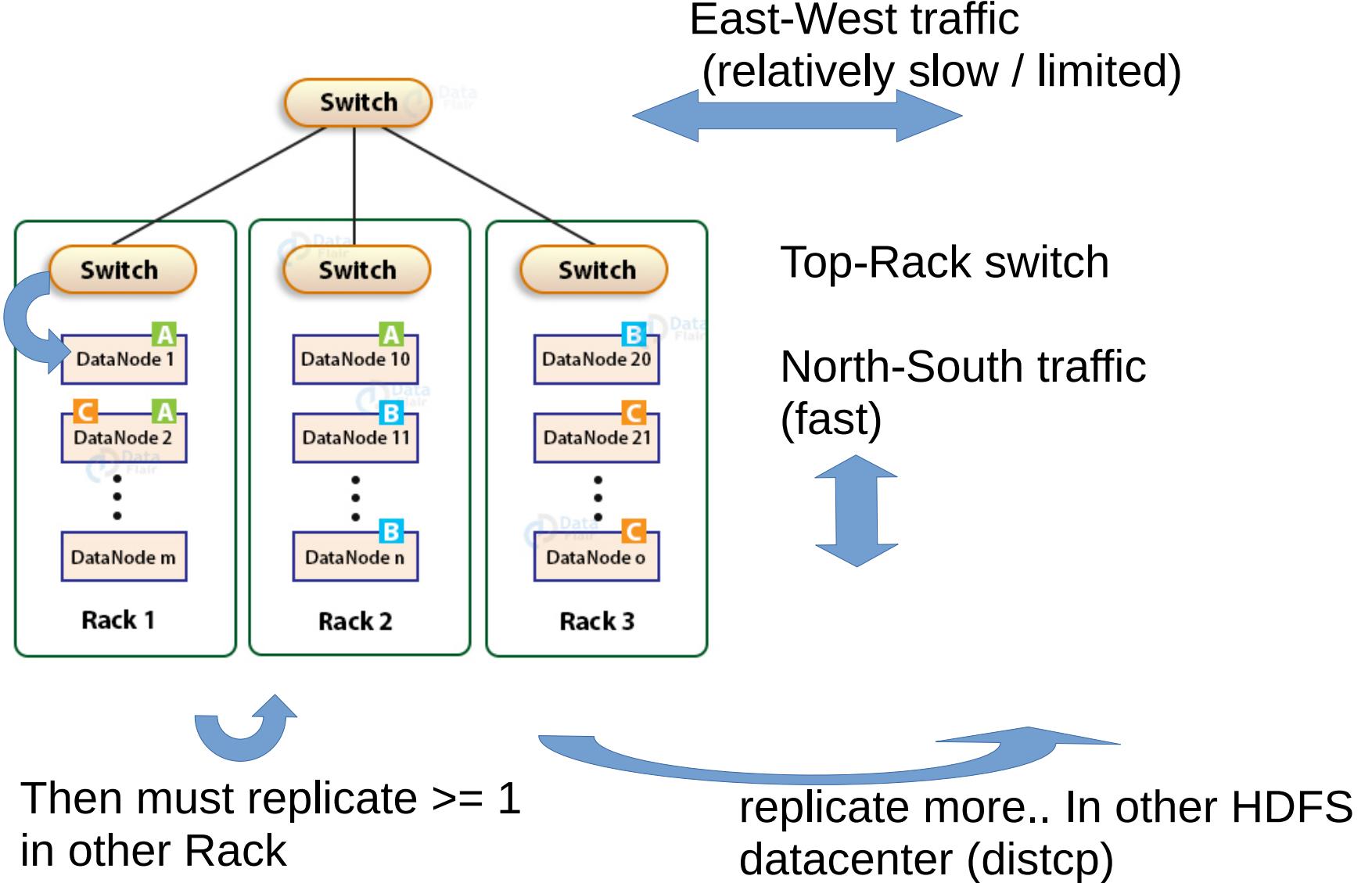
# Databases choices



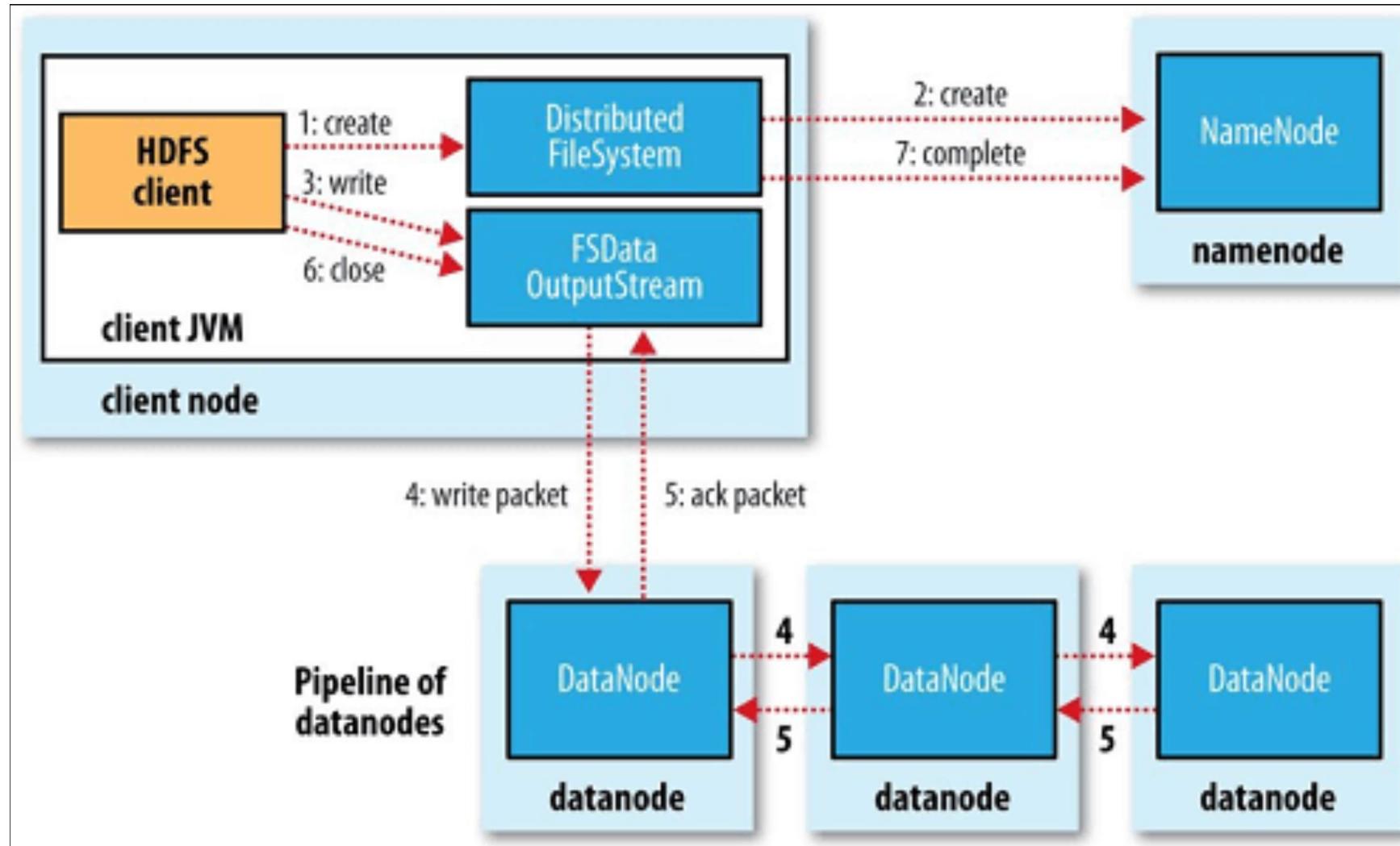
# Hadoop Legacy Replication Factor: Rack Aware



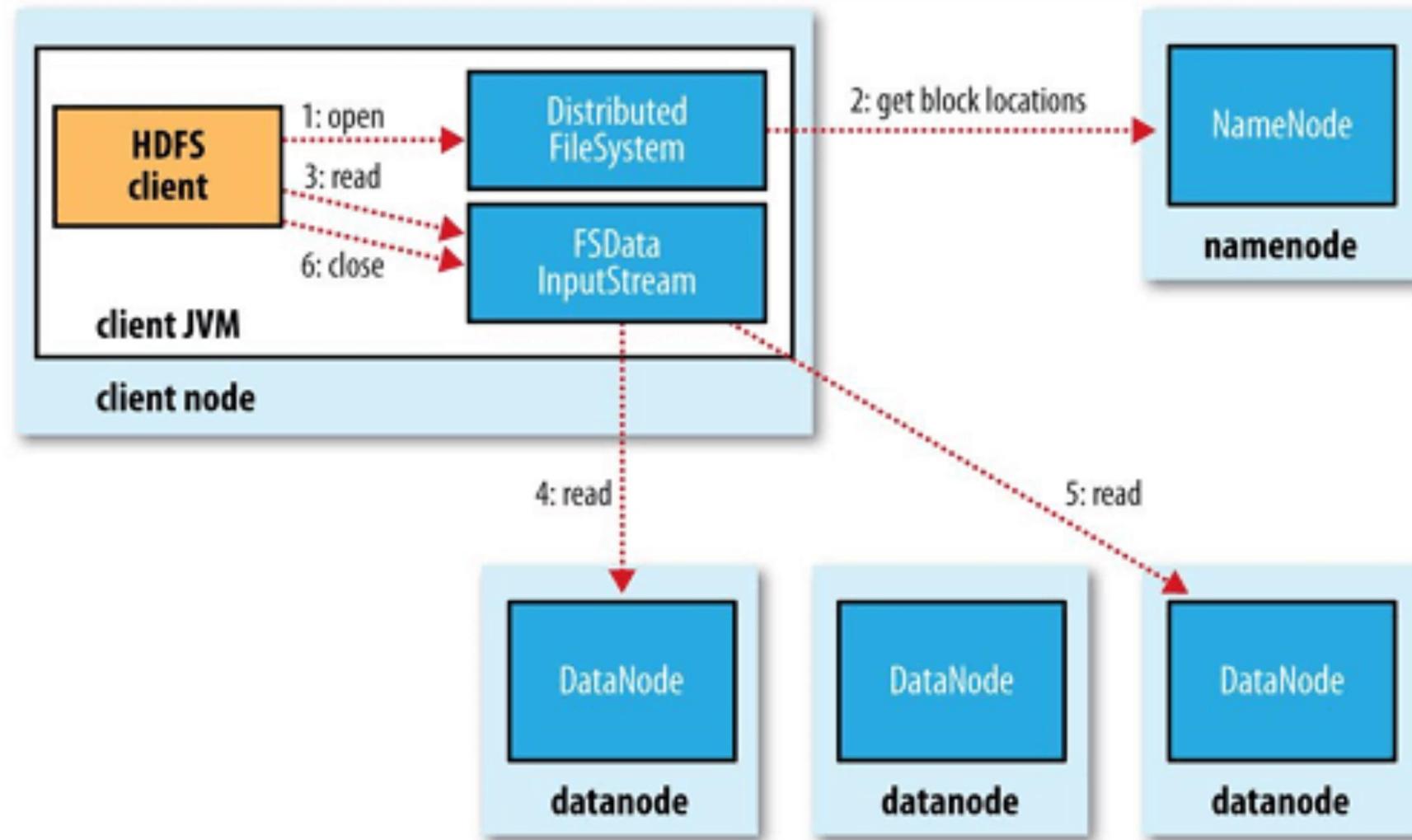
Allow replicate 1  
in same Rack  
(fast)



# Writing a HDFS File



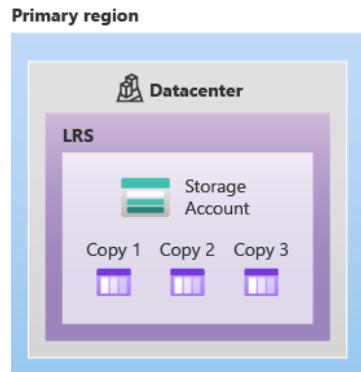
# Reading a HDFS File



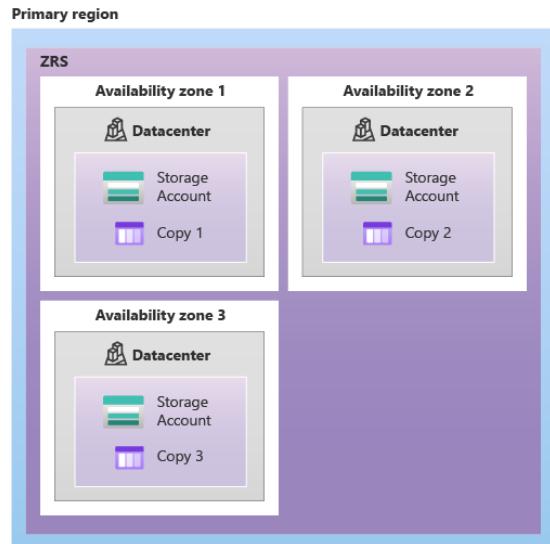
# Example on Azure Storage... LRS, ZRS, GRS, GZRS

## 1 Region = 3 AZ (Availability Zone) .. 3 Datacenters

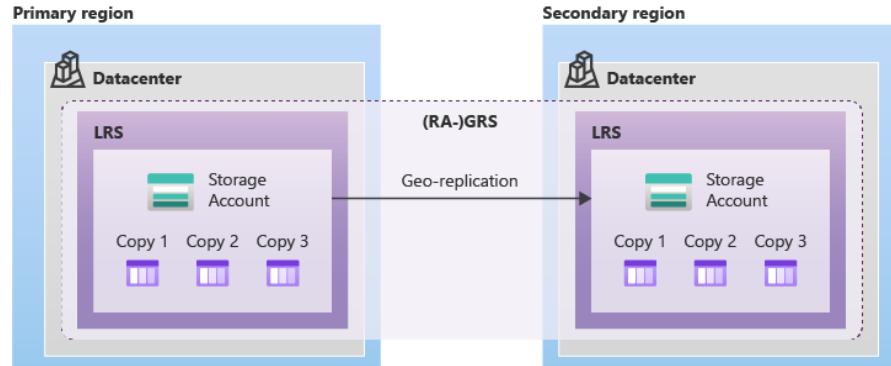
**LRS** (cheaper)  
... 1 Datacenter



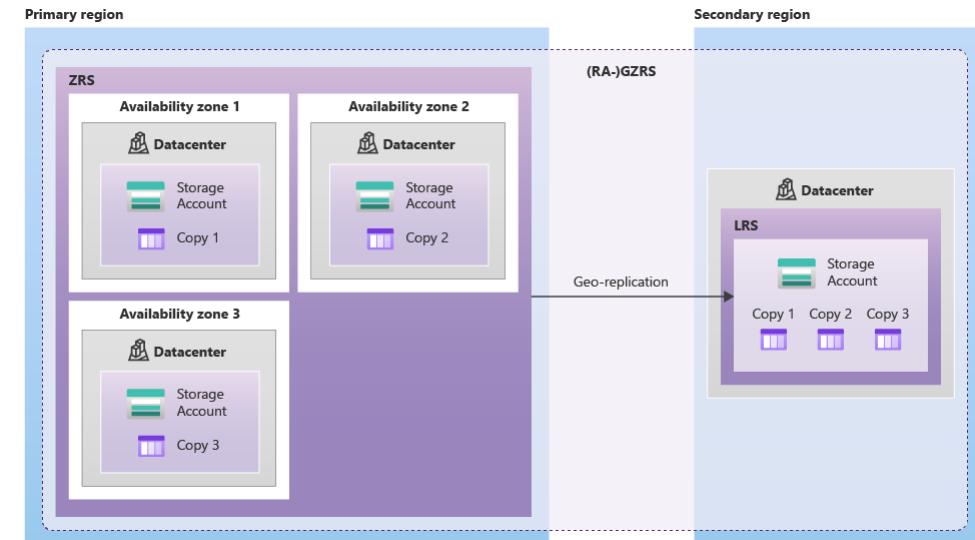
**ZRS ... 1 Region**



**(RA-) GRS** ... 1 LRS + async 1LRS, 2 zones



**(RA-) GZRS** ... 1 ZRS + async 1LRS

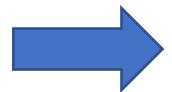


# Recap ... Storage Resources

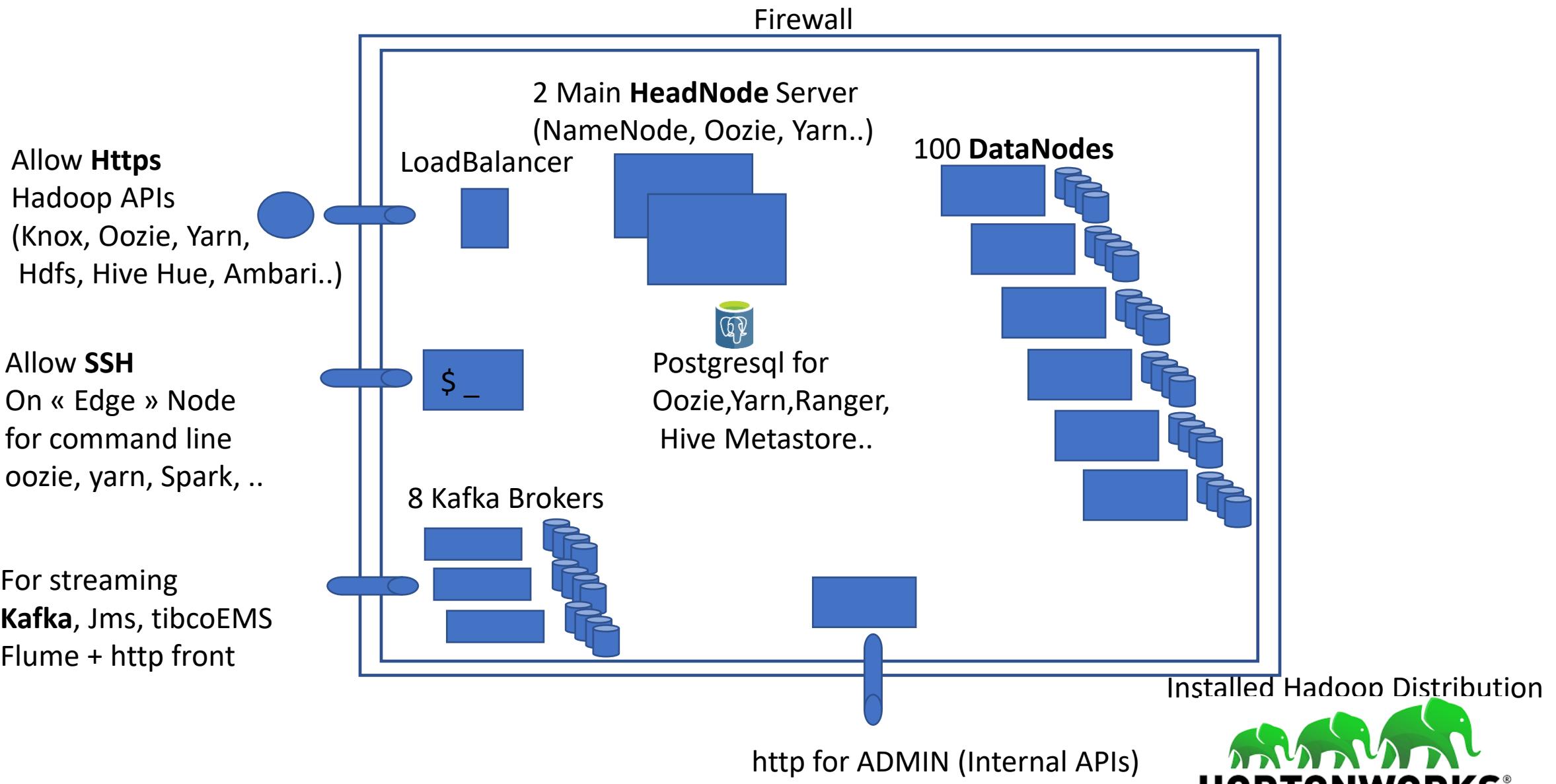
Coffe Break ?

# Outline

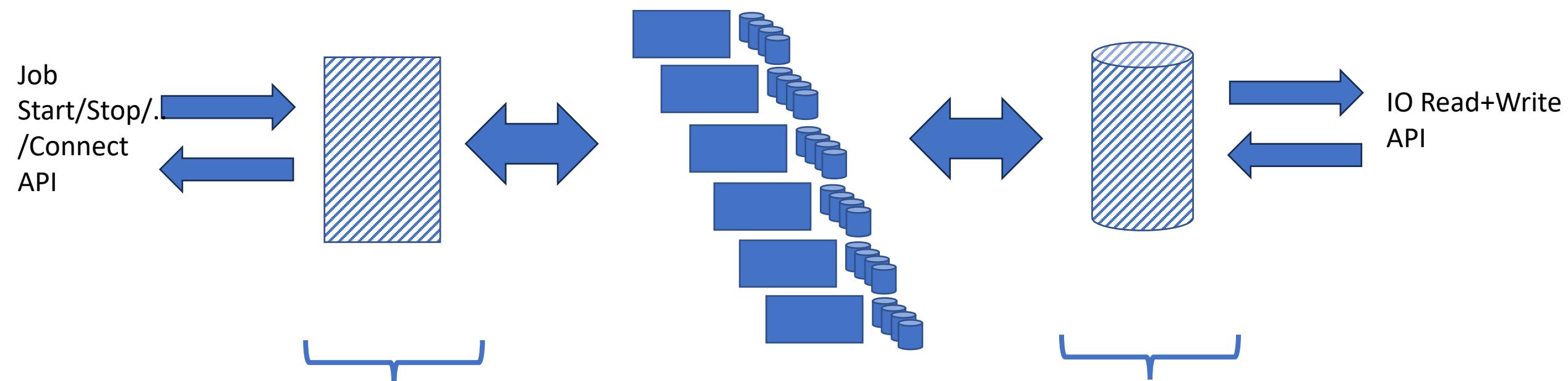
- What is BigData ?
  - Order of Magnitudes for « Big »
  - Distributed System challenges
  - Scaling Storage Resources: Hdfs
- **Scaling Compute Resources: Yarn, Kubernetes**
  - Using Storage – Compute apis
  - Evolution of Hardware – Softwares – Hadoop Ecosystem, Spark
- Description of a Datalake
  - Content
  - Usages
- Example of Data-Processing
  - RAW to LAKE, Reports
- Evolution to Cloud



# Reminder ... Hadoop Cluster Architecture



# 2 Distributed Challenges : 1=Storage + 1=Compute



1 Virtual « **Compute** » Resource  
= Yarn (or Kubernetes)  
Distributed Container(Process/Pod) API

1 Virtual « **Storage** » Resource  
= HDFS  
Hadoop Distributed File System

# Mind Shift: SSH -> Job Compute Resource

Instead of  
Handling remote SSH / exec - kill  
and naming server & processes ...

```
# SSH for Remote command
$ ssh server-abc-for-dept-xyz
$ cd someDir

# long running services
$ sudo systemctl start my-service
$ do; if [ -e/proc/$pid ] ./start.sh; sleep; done

# (short running) job application
$ ./start.sh; hdfs dfs –get yourapp.jar
$ java –jar yourapp.jar
$ ps, pid=..
$ kill -9 $pid
```



Do this:  
**Servers are invisible to users**  
**User see virtual Services and Jobs scheduling**

```
# execute commands from any client
$ curl http://ambari admin commands

$ oozie job start .. hdfs://.../workflow.xml
$ oozie job start .. hdfs://.../coordinator.xml
$ oozie job kill $oozield

$ yarn app -list
$ yarn app –kill $yarnId

# OR BETTER in Kubernetes
$ kubectl apply –f myservice.yml
$ kubectl apply –f my-spark-job.yml
$ kubectl sparkApp list|get|delete
```

# Pet vs Cattle

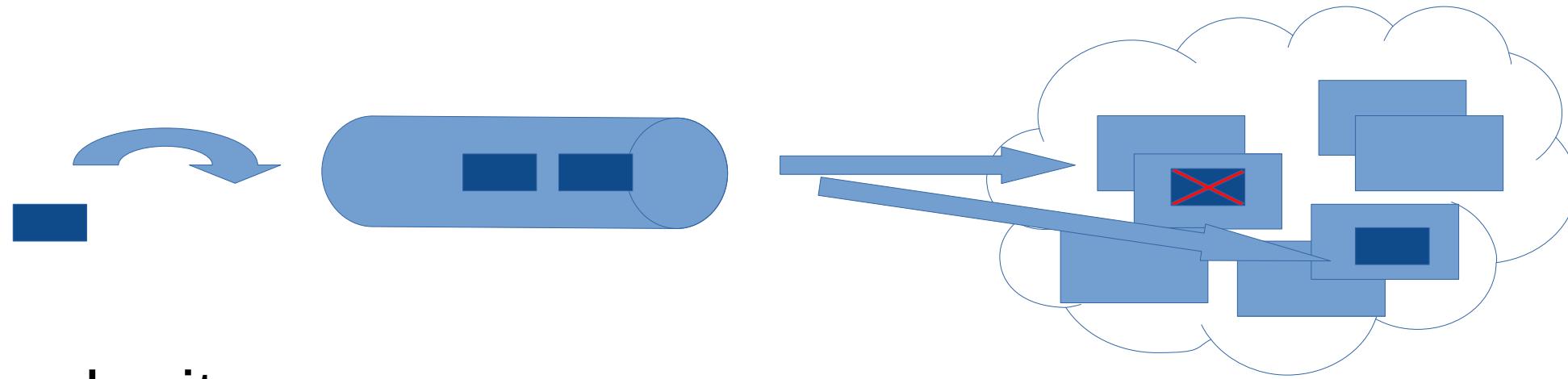


Pet have id=name  
You take extra care of it



Cattle  $\geq 1000$  : anonymous  
Id=Number, interchangeable

# Launching => Scheduling on Allocated Resource



You submit  
Something to run

You don't run yourself  
You don't choose server

Then wait..  
maybe allocate ++resources  
(=> app retries managed)  
You don't have access to servers

# Yarn Queue



## NEW, NEW\_SAVING, SUBMITTED, ACCEPTED, RUNNING Applications

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved	Active Nodes	Decommissioned Nodes	Lost Nodes
1	0	1	0	12	20.88 GB	102.38 GB	0 B	12	64	0	4	0	0 !

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Alloc
Capacity Scheduler	[MEMORY]	<memory:32, vCores:1>	<memory:26208, vCores:16>

Application Queues

Legend: Capacity (green bar), Used (blue bar), Used (over capacity) (orange bar), Max Capacity (grey bar)

```
graph TD; root --> q1; q1 --> q1q11; q1 --> q1q12; q1 --> q2; q1 --> default;
```

Queue	Capacity	Used	Over Capacity	Max Capacity
root	20.4%	0.0%	0.0%	20.4%
q1	0.0%	0.0%	0.0%	0.0%
q1.q11	0.0%	0.0%	0.0%	0.0%
q1.q12	0.0%	0.0%	0.0%	0.0%
q2	0.0%	0.0%	0.0%	0.0%
default	102.0%	102.0%	0.0%	102.0%

Show 20 entries

ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress	Tracking
application_1494408864466_0001	hadoop	random-text-writer	MAPREDUCE	default	Wed May 10 18:07:39 +0800 2017	N/A	RUNNING	UNDEFINED	0%	Application

Queue Hierarchy :

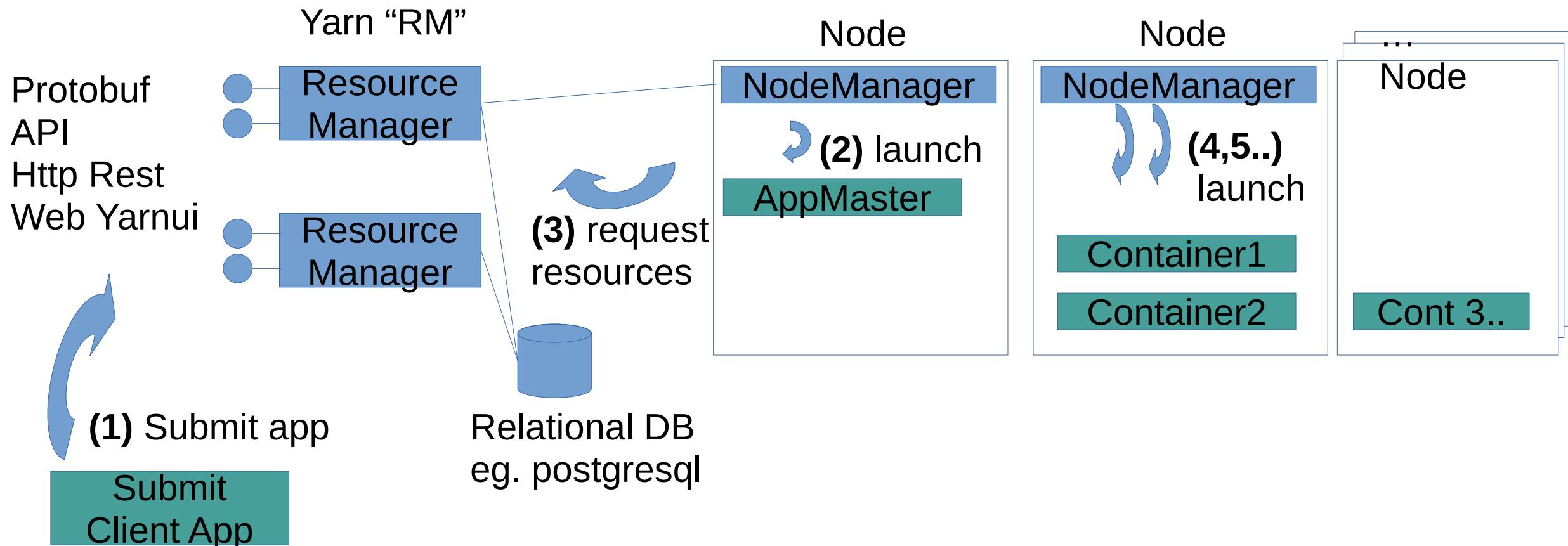
Child queues => sum to 100%

Capacity / Max Capacity  
... over-quota allowed

If Preemption enabled =>  
may kill app in over-quota

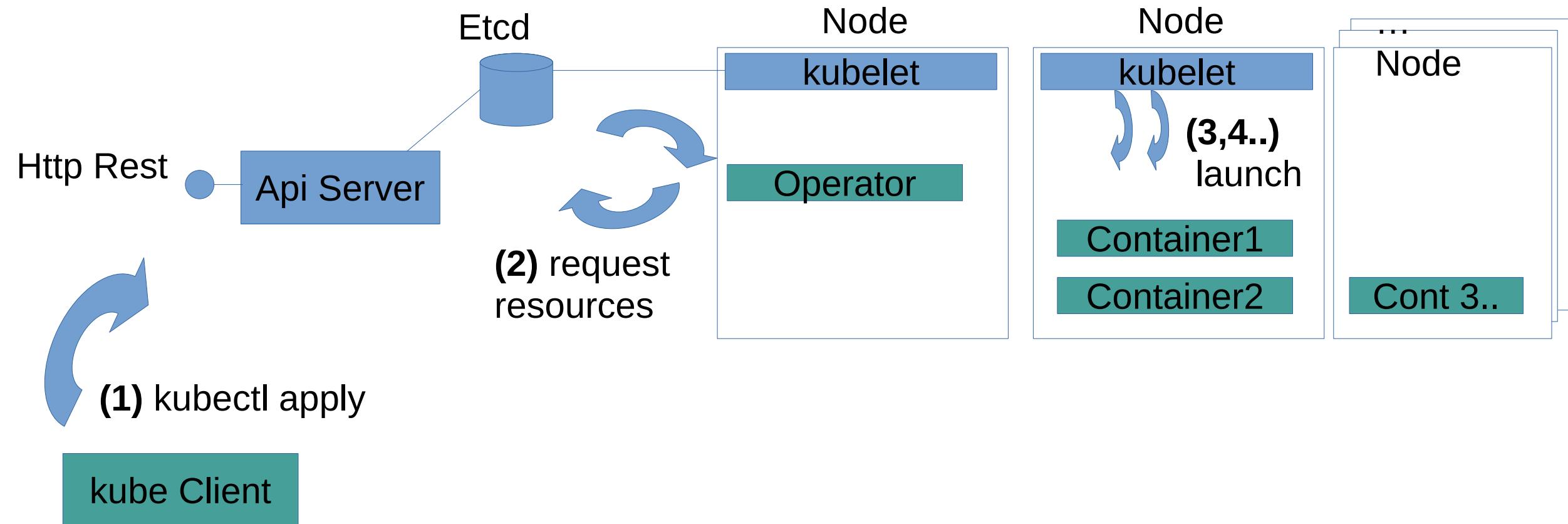


# Hadoop Yarn Architecture

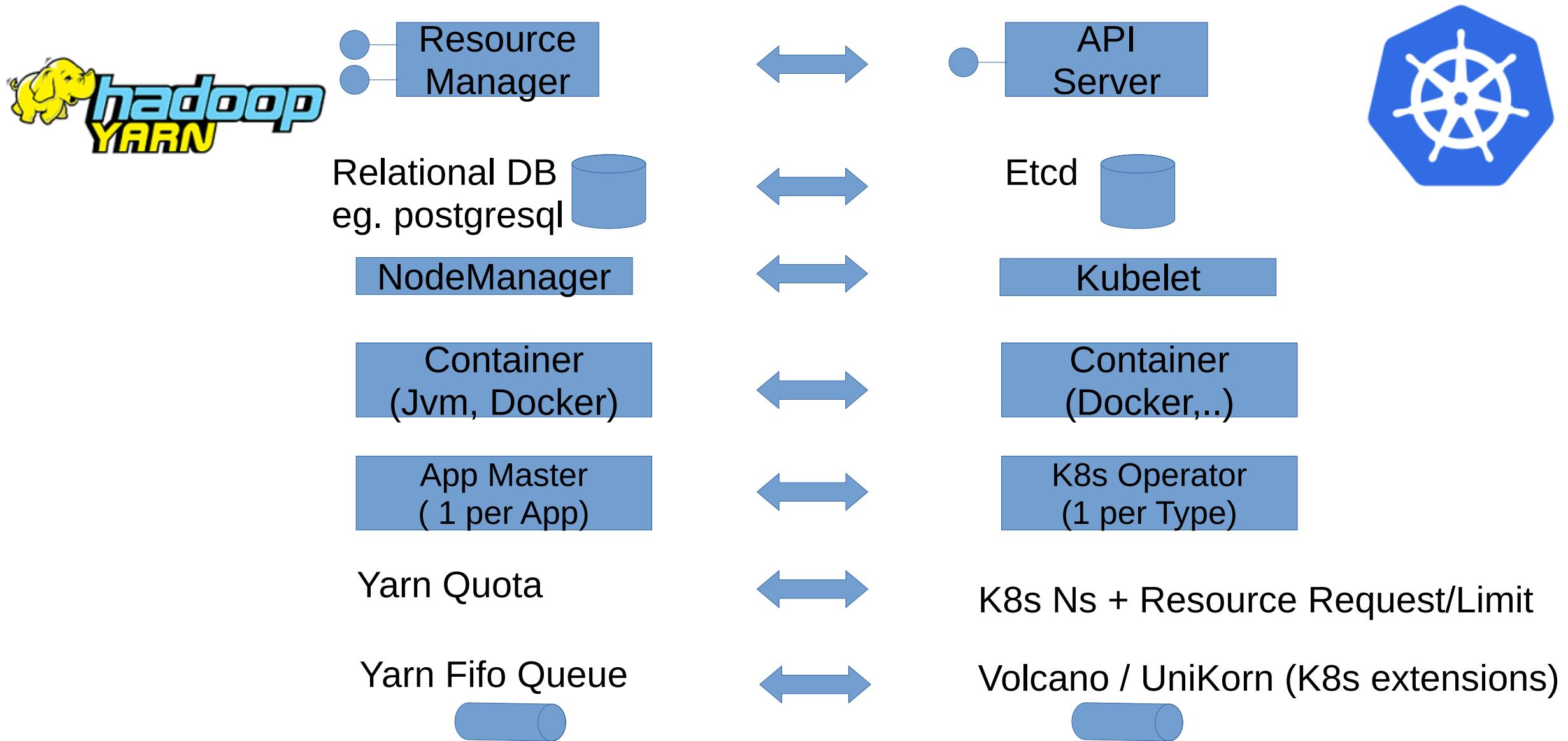




# Kubernetes Architecture



# Architecture Comparison Yarn <-> Kubernetes



# Oozie → Yarn UI → Yarn ... Nightmare, ex: Logs

## 1/ Find Oozie workflow Id

Job Name: My\_first\_Workflow/JobId: 0000007-150727083427440-oozie-oozi-W

Job Id: 0000007-150727083427440-oozie-oozi-W  
Name: My\_first\_Workflow  
App Path: hdfs://sandbox.hortonworks.com:8020/user/hue/oozie/worksheets/hue-o...  
Run: 0  
Status: SUCCEEDED  
User: hue  
Group:  
Parent Coord:  
Create Time: Mon, 27 Jul 2015 11:06:58 GMT  
Start Time: Mon, 27 Jul 2015 11:06:58 GMT  
Last Modified: Mon, 27 Jul 2015 11:06:59 GMT  
End Time: Mon, 27 Jul 2015 11:06:59 GMT

Action Id: 0000007-150727083427440-oozie-oozi-W@start  
Name: start  
Type: fs  
Status: OK  
Transition: fs->2178

Action Id: 0000007-150727083427440-oozie-oozi-W@End  
Name: End  
Type: fs  
Status: OK  
Transition: 2178->

Action Id: 0000007-150728104024980-oozie-oozi-W  
Name: end  
Type: fs  
Status: OK  
Transition: fs->

Action Id: 0000007-1507280913189-oozie-oozi-W  
Name: end  
Type: fs  
Status: OK  
Transition: fs->

Action Id: 0000007-15072809413189-oozie-oozi-W  
Name: end  
Type: fs  
Status: OK  
Transition: fs->

## 2/ oozie-action

Action Info | Action Configuration

Name: mr-node  
Type: map-reduce  
Transition:  
Start Time: Fri, 14 Oct 2011 07:56:16 GMT  
End Time:  
Status: RUNNING  
Error Code:  
Error Message:  
External ID: job\_201110101107\_0004  
External Status: RUNNING  
Console URL: http://localhost:50030/jobdetails.jsp?jobid=job\_201110101107\_0001  
Tracker URI: localhost:9001

## 4/ Console URL = url in Yarn (Proxy or History)

MapReduce Job job\_1381856870533\_0004

ApplicationMaster	Attempts	Start Time	Node	Logs
Map	1	Fri Oct 13 13:28:28 EDT 2017	10.0.2.15:50842	
Map	2	Fri Oct 13 13:28:28 EDT 2017	10.0.2.15:50842	
Map	3	Fri Oct 13 13:28:28 EDT 2017	10.0.2.15:50842	
Map	4	Fri Oct 13 13:28:28 EDT 2017	10.0.2.15:50842	

## 9/ application (example: Reduce for shell)

## Example Shell → spark-submit

- 7/ spark
- 6/ Yarn (spark-submit client)
- 5/ Mapreduce (SHELL)  
... call spark-submit --mode cluster

All Applications

ID	User	Name	Application Type	Queue	Start Time	Final Status	Running Containers	Allocated CPU Vcores
application_1455000259206_0001	dr.who	infaspark SPARK	root	root	Mon May 16 07:35:26 +0000 2016	FINISHED	0	0
attempt_1455000259206_0001_000001	dr.who	infaspark SPARK	root	root	Mon May 16 07:35:26 +0000 2016	FINISHED	0	0
attempt_1455000259206_0001_000002	dr.who	infaspark SPARK	root	root	Mon May 16 07:35:26 +0000 2016	FINISHED	0	0
attempt_1455000259206_0001_000003	dr.who	infaspark SPARK	root	root	Mon May 16 07:35:26 +0000 2016	FINISHED	0	0
attempt_1455000259206_0001_000004	dr.who	infaspark SPARK	root	root	Mon May 16 07:35:26 +0000 2016	FINISHED	0	0
attempt_1455000259206_0001_000005	dr.who	infaspark SPARK	root	root	Mon May 16 07:35:26 +0000 2016	FINISHED	0	0

Application application\_1455000259206\_0001

Kill Application

User: root  
Name: select count(\*) from server\_logs(Stage-1)  
Application Type: MAPREDUCE  
YarnApplicationState: ACCEPTED: waiting for AM container to be allocated, launched and register with RM.  
FinalStatus Reported by AM: Application has not completed yet.  
Started: Tue Feb 09 07:35:26 +0000 2016  
Elapsed: 50sec  
Tracking URL: ApplicationMaster  
Diagnostics:

Application Metrics

Total Resource Preempted:	<memory:0, vCores:0>
Total Number of Non-AM Containers Preempted:	0
Total Number of AM Containers Preempted:	0
Resource Preempted from Current Attempt:	<memory:0, vCores:0>
Number of Non-AM Containers Preempted from Current Attempt:	0
Aggregate Resource Allocation:	0 MB-seconds, 0 vcore-seconds

Logs

Attempt ID	Started	Node	Logs
attempt_1455000259206_0001_000001	Fri Feb 09 07:35:26 +0000 2016	http://	0

## 8/ appmaster application attempt 1

Logs for container\_1428386215281\_60485\_01\_000002

```
spark.log : Total file length is 25231972 bytes.
spark.log_1: Total file length is 52428862 bytes.
spark.log_2: Total file length is 52428888 bytes.
spark.log_3: Total file length is 52428821 bytes.
spark.log_4: Total file length is 52428942 bytes.
stderr : Total file length is 52428929 bytes.
stdout : Total file length is 1508202 bytes.
```

## 11/ stdout !!

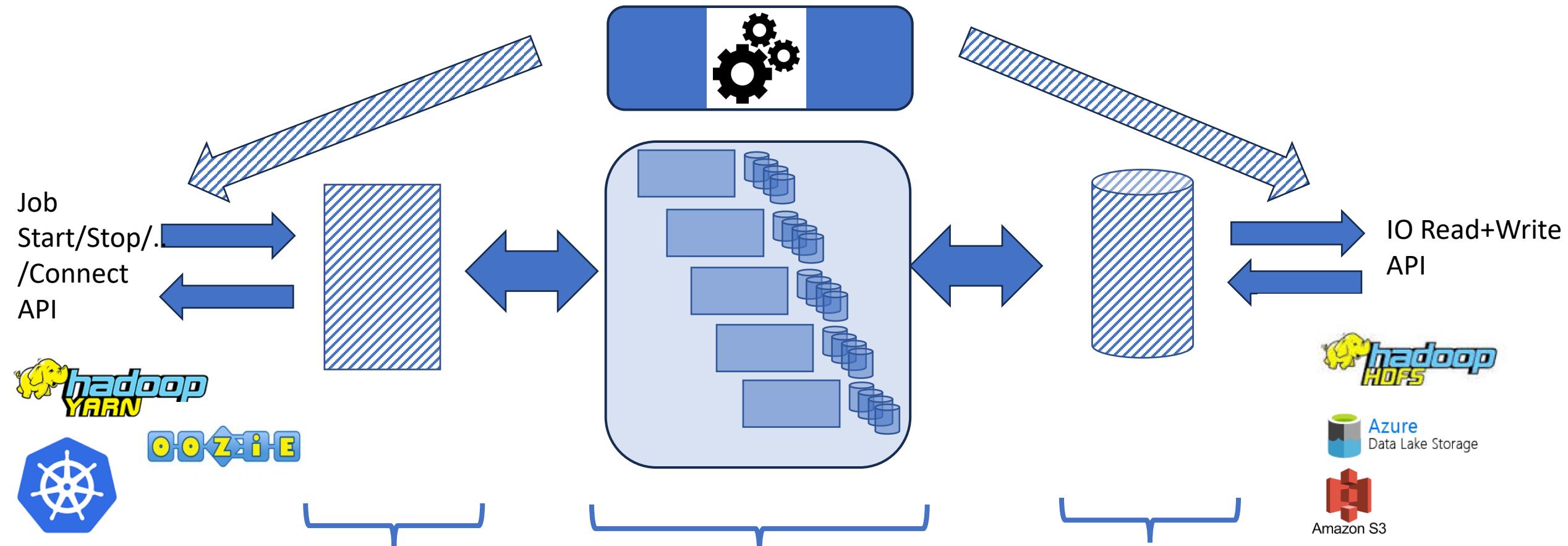
Logs for container\_1428386215281\_60485\_01\_000002

```
spark.log : Total file length is 25231972 bytes.
spark.log_1: Total file length is 52428862 bytes.
spark.log_2: Total file length is 52428888 bytes.
spark.log_3: Total file length is 52428821 bytes.
spark.log_4: Total file length is 52428942 bytes.
stderr : Total file length is 52428929 bytes.
stdout : Total file length is 1508202 bytes.
```

# Outline

- What is BigData ?
  - Order of Magnitudes for « Big »
  - Distributed System challenges
  - Scaling Storage Resources: Hdfs
  - Scaling Compute Resources: Yarn, Kubernetes
- **• Using Storage – Compute apis**
  - Evolution of Hardware – Softwares – Hadoop Ecosystem, Spark
- Description of a Datalake
  - Content
  - Usages
- Example of Data-Processing
  - RAW to LAKE, Reports
- Change Storage-Compute, Evolution to Cloud

# Recap .. Objective for Distributed Data Processing Solution: Use Compute + Storage APIs

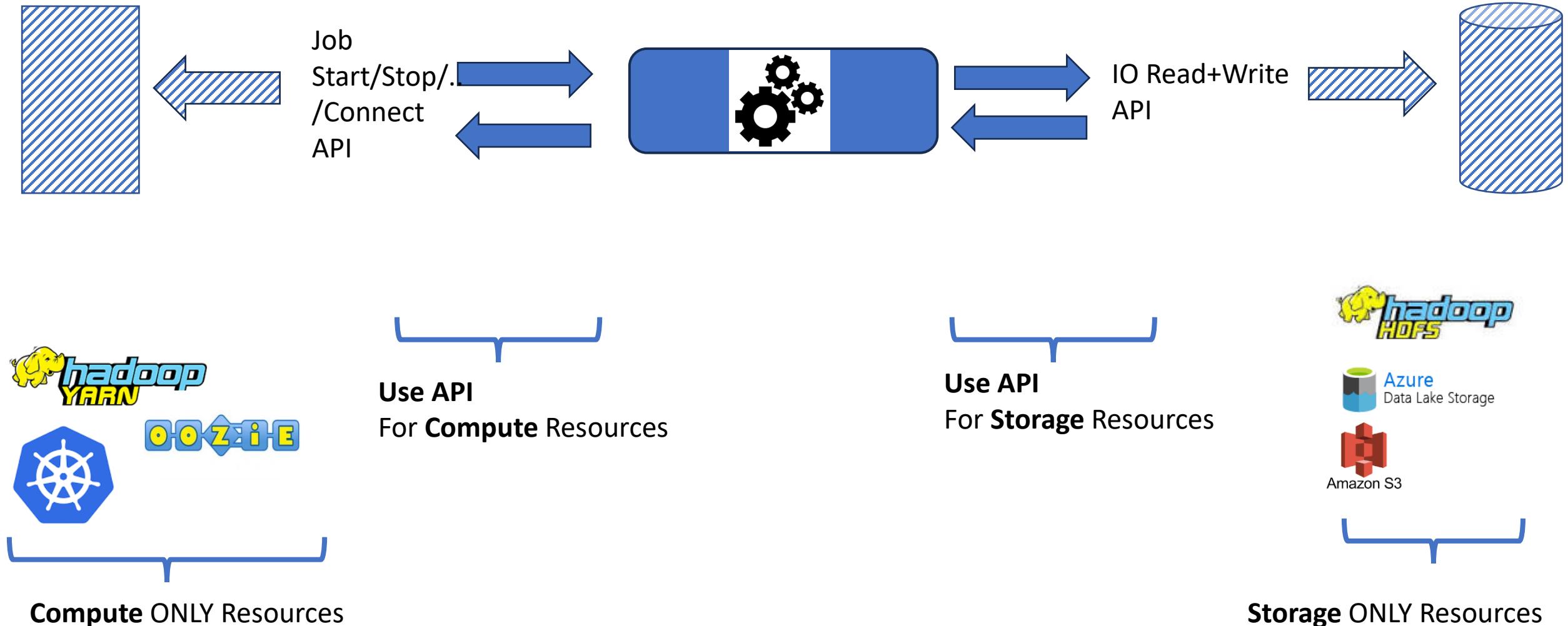


1 Virtual « **Compute** » Resource  
= Yarn+Oozie (or Kubernetes)  
Distributed Container(Process/Pod) API

Hidden physical  
Resources

1 Virtual « **Storage** » Resource  
= HDFS  
Hadoop Distributed File System

# Recap .. Objective for Distributed Data Processing Solution: Use Compute + Storage APIs

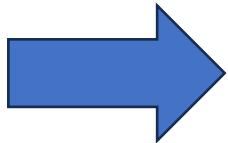


# Data Processing = using Compute + Storage APIs

**Compute**

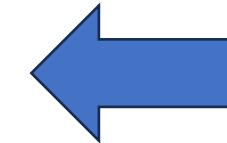
Cluster

$\geq 200$  servers



**Storage**

$\geq 3$  Peta bytes



Your Data Processing



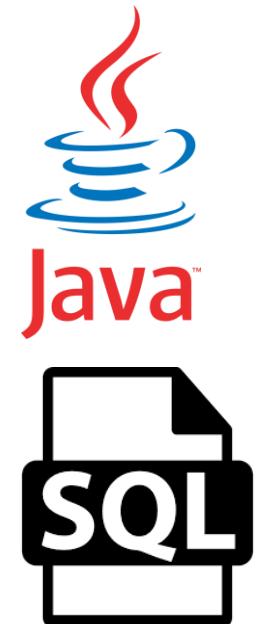
Amazon S3

<https://www.youtube.com/watch?v=YVAVugfnW6Y>



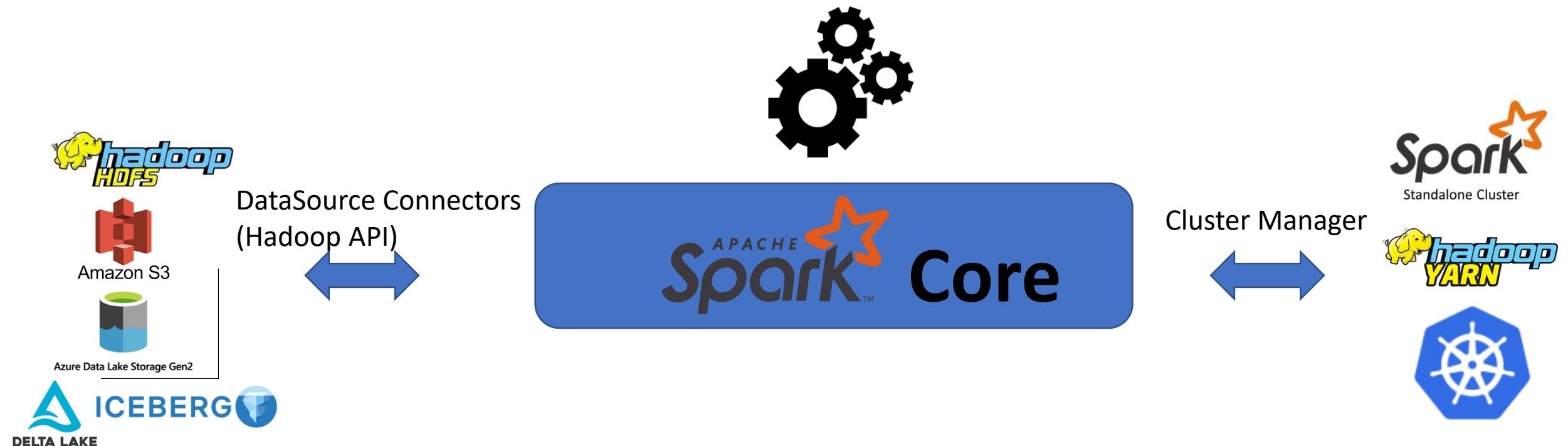
# Spoiler Alert ... Data Processing in Spark

Your Code:



# Spark Core

## Unified Engine... using Compute + Storage APIs



# Outline

- What is BigData ?
    - Order of Magnitudes for « Big »
    - Distributed System challenges
    - Scaling Storage Resources: Hdfs
    - Scaling Compute Resources: Yarn, Kubernetes
    - Using Storage – Compute apis
    - **Evolution of Hardware – Softwares – Hadoop Ecosystem, Spark**
  - Description of a Datalake
    - Content
    - Usages
  - Example of Data-Processing
    - RAW to LAKE, Reports
  - Change Storage-Compute, Evolution to Cloud
- 

# Hardware Evolutions... => Architecture Changes

HDD = Hard Disk (magnetic + disks rotation) => slow « seek » to position

SSD => no more « seek » problem, so NO need to locally cache data in RAM  
and huge read Go per second

Disk connectivity to motherboard => HDI < SCSI < SATA < ...PCI Nvme ...

Google « nvme vs sata ssd bandwidth »

SSD + NvMe => HUGE bandwith ... > 4Go per seconds  
( maybe more than network card support?)

Network cards ... 10Go/s ... fiber channel 100Go/s

So downloading HUGE file for Full scan is now OK !!

**No more need to cache in local RAM, no need to collocate program near data**

How Softwares evolve to handle BigData ?

Which ones to use today?

# Hadoop Ecosystem « Explosion »

# Spark (Recent) History & Ancestors

MapReduce @ 

2002 @Google    2004 Google  
Paper published

2014 Google  
No more used of MapReduce



2006 @Yahoo  
Hadoop implementation  
2008 Apache Open-Source

2012 Yarn (v2)

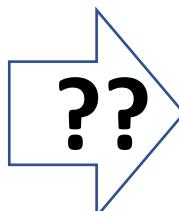
2021 MapReduce bashing  
... HDFS & Hadoop also  
HortonWork bought by Cloudera  
HDInsight @Azure...very bad choice  
.. To be abandonned



1995 Message Passing Interface



2010 Spark paper    2013 Apache top-level



2015 Kubernetes    2020  
 Spark on K8s

# Datalake History At SG

~8 years ago  
2014



~2 years ago (2020)  
Decision to migrate to Azure  
+ Dismantle On-Prem

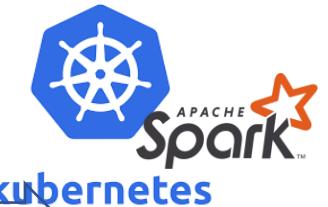
Dec 2022  
OFF



Init BigData  
cluster at SGCIB

2 major PROD projects:  
- MiFid reports (all trades)  
- Var Risk

> 150 projects



~ 1 year ago (2021)  
>= 100 projects migrate to Azure



Internal Re-organizations  
Another Cluster..

Point in common  
Despite all changes



# MapReduce @Yahoo = Hadoop .. 2006

Constraint  
=>  
Architecture  
Choice

Commodity Hardwares (datacenters):  
Only HDD + RAM  
**Data Locality** : co-host Storage near Compute  
use RAM to cache  
avoid network



Think different?



2006



2020

# MapReduce -> Spark + other changes

Constraint

=>

Architecture  
Choice



2006

Faster Disk (SSD)

Faster Network

Compute != Storage

Cloud



kubernetes



Google Cloud

Think different?



2022

# Simple => Many Specific Systems => Unified



« Simple » ecosystem  
( verbose inefficient &  
complex java code)

« **Bazard** » ecosystem  
**(Too MANY TOO SPECIFIC**  
redundant, complexes)

**“Unified” ecosystem**  
**Simple**  
+ extensible modules

At The end, Only 1 will remain  
( French TV Game: Koh-Lanta)





# Spark = « Unified Engine »

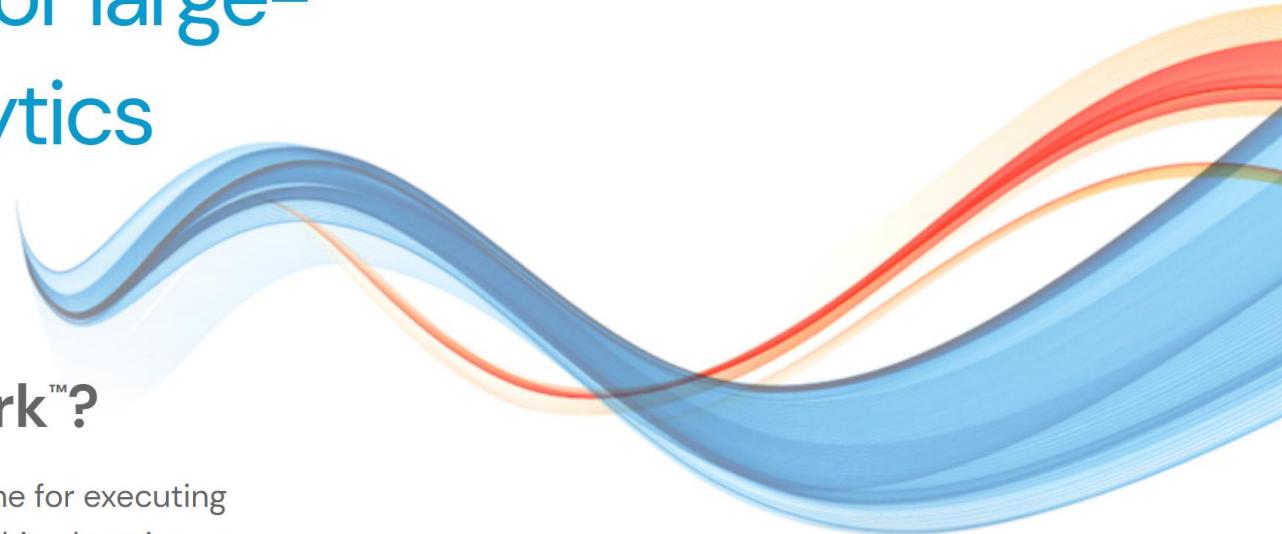


## Unified engine for large-scale data analytics

[GET STARTED](#)

### What is Apache Spark™?

Apache Spark™ is a multi-language engine for executing data engineering, data science, and machine learning on single-node machines or clusters.



# Multi Purposes – Multi Languages

**Simple.  
Fast.  
Scalable.  
Unified.**

## Key features



### Batch/streaming data

Unify the processing of your data in batches and real-time streaming, using your preferred language: Python, SQL, Scala, Java or R.



### SQL analytics

Execute fast, distributed ANSI SQL queries for dashboarding and ad-hoc reporting. Runs faster than most data warehouses.



### Data science at scale

Perform Exploratory Data Analysis (EDA) on petabyte-scale data without having to resort to downsampling



### Machine learning

Train machine learning algorithms on a laptop and use the same code to scale to fault-tolerant clusters of thousands of machines.

Python

SQL

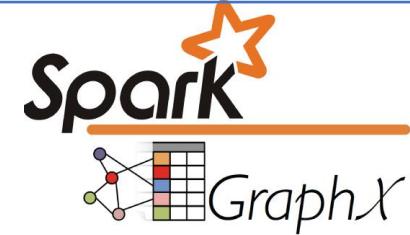
Scala

Java

R

# Spark-Core + ...

Structured  
Data



Modules



DataSource Connectors  
(Hadoop API)



Cluster Manager



Languages Support



# Take Away

BigData = VVV ...

Distributed Architectures (Horizontal Scaling) => Challenges

Compute vs Storage Resources

Compute Api => Yarn, Kubernetes / Storage Api => Hdfs, Cloud

Hardware evolves, Hadoop Hdfs/Yarn is deprecated

Spark = Unified Engine, using both Compute Api and Storage Api

Coffee Break

# Outline

- What is BigData ?
  - Order of Magnitudes for « Big »
  - Distributed System challenges
  - Scaling Storage Resources: Hdfs
  - Scaling Compute Resources: Yarn, Kubernetes
  - Using Storage – Compute apis
  - Evolution of Hardware – Softwares – Hadoop Ecosystem, Spark
- Description of a Datalake
- Content
  - usages
  - Example of Data-Processing
    - RAW to LAKE, Reports
  - Change Storage-Compute, Evolution to Cloud

# Not Only Machines

**Tower of Babel**  
מגדל בבל



*The Tower of Babel* by Pieter Bruegel the Elder (1563)

**General information**

Type	Tower
Location	Babylon
Height	See § Height

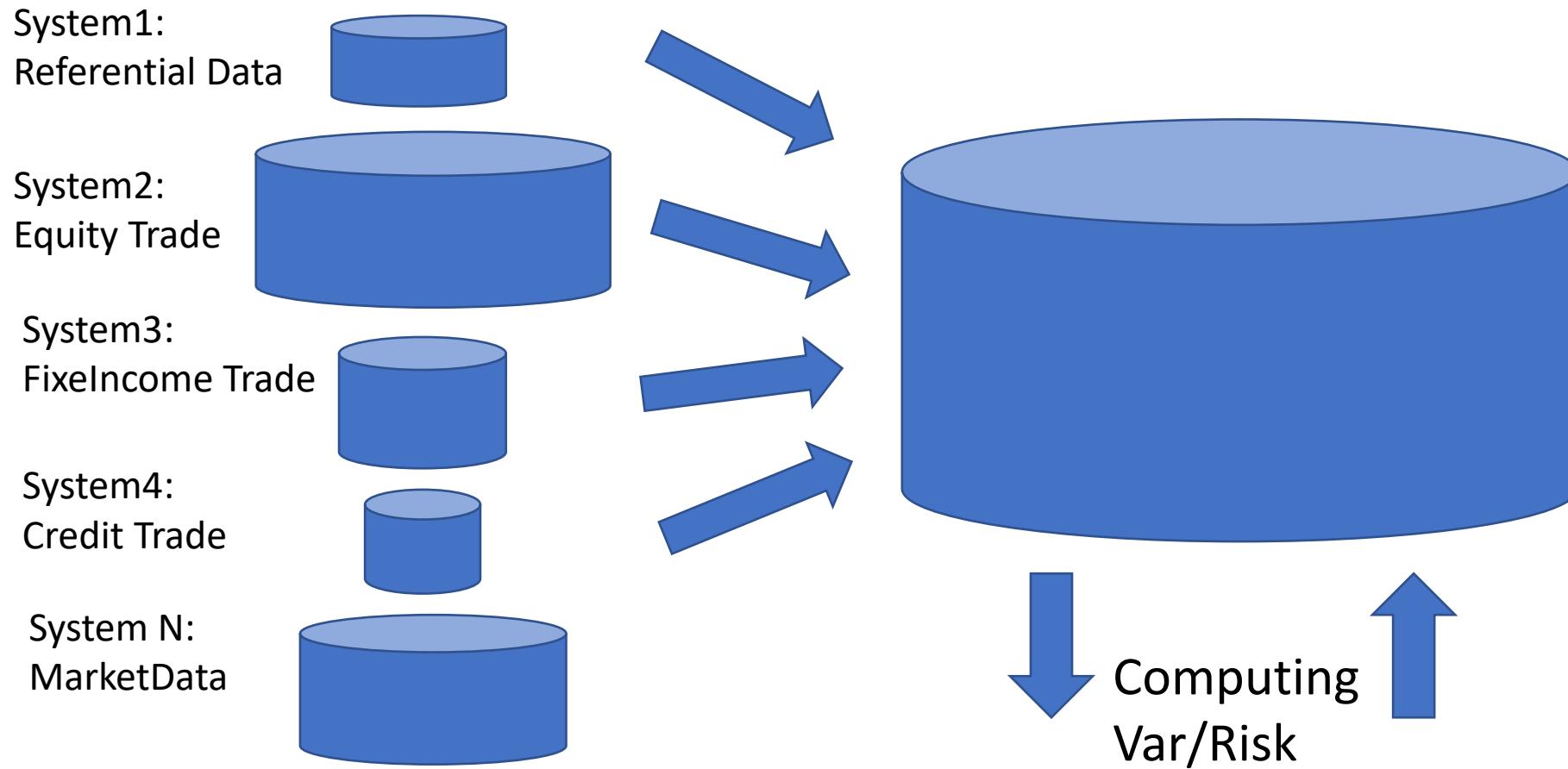
~ 200 different Teams / 10 departments  
... for 1000 developpers / data engineers / users

In Paris.. But not Only (Bangalore, Montreal, London, New-York, ..)

On ~6000 tables,  
described in 1500 DataSets

Lot of Financial / Functional aspects

# Feeding Data Inputs to Datalake: aggregating $\geq 150$ Systems



# Feeding Data / Streaming

BigData at SG is mostly « uploading daily files »  
then launching « daily batches »

Many Trade extractions are Real-Time Streaming  
... but aggregated in Files per Hours / per Day

Var Computing during night ... Huge streaming of events  
from dedicated pricing cluster (GPUs)

# Feeding Types & Volumetry

Daily batches: upload Files

Streaming then converted to Files

Streaming to Key-Values Databases



**Teras/day  
( millions of files)**



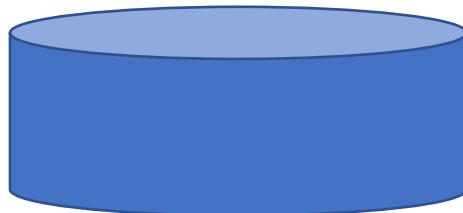
**Billions events/day**

Peeks to 4000 events/seconds  
on each 100 x servers  
During 4hours nightly Var batches

# Content of a Datalake



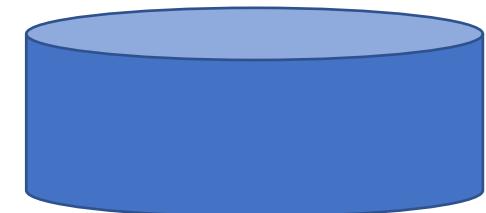
FileSystem  
= Directories + Files



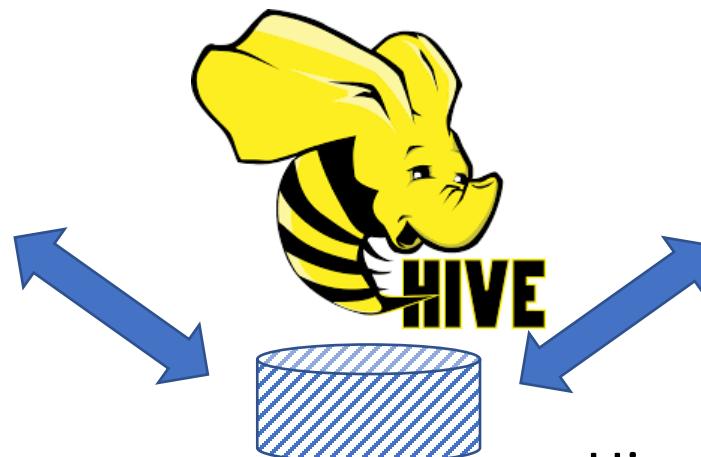
For batches



Key-Value database



FAST get/put access by single key



Hive Metastore = SQL View  
« Table » abstraction for HDFS / Hbase

# Storage = Files and Directories

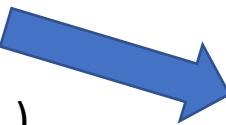
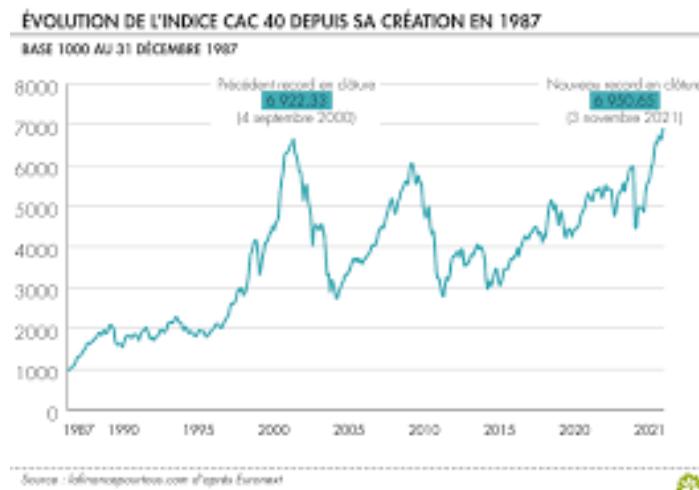
How would you organize  
**1 Billions Files**  
**in 50 Millions Directories ?**

How do you ensure data respect directories organization ?

Try Scanning Directories ?

# A Very « Simple » Specific Sub-System

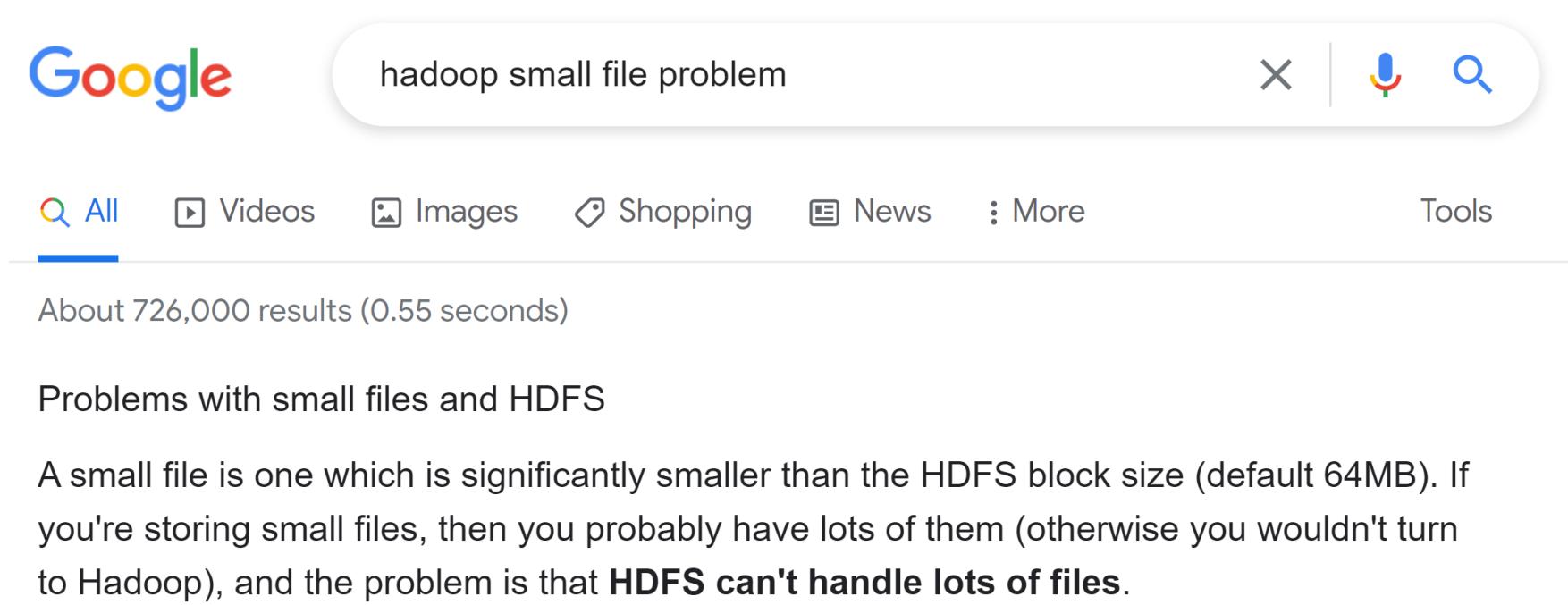
« Tick » per seconds  
From Market Exchange Places  
(Bourse Paris, London Stock Exchange, ...)



- 1 binary file per Day/Place/Instrument
- ... 2M files per day
- ... 40 Giga per day
- ... since 22 years ... 25 Billions Files / Petas

Volume, Velocity ... but not Variety  
( NO spark, No Hadoop )

# Hadoop hates too many Small Files



A screenshot of a Google search results page. The search bar at the top contains the query "hadoop small file problem". Below the search bar, there are several navigation links: "All" (which is underlined in blue), "Videos", "Images", "Shopping", "News", "More", and "Tools". A status message indicates "About 726,000 results (0.55 seconds)". The main content area starts with the heading "Problems with small files and HDFS". Below this, a paragraph explains that a small file is one significantly smaller than the HDFS block size (default 64MB), and that storing many small files leads to performance issues because HDFS cannot handle lots of files.

hadoop small file problem

All Videos Images Shopping News More Tools

About 726,000 results (0.55 seconds)

Problems with small files and HDFS

A small file is one which is significantly smaller than the HDFS block size (default 64MB). If you're storing small files, then you probably have lots of them (otherwise you wouldn't turn to Hadoop), and the problem is that **HDFS can't handle lots of files**.

BIG Files ? More than Giga ?  
YES ... BUT PARQUET Partitioned

Doing BigData = using



Files (correctly with APACHE **Spark**)

End Of Story.

# Outline

- What is BigData ?
  - Order of Magnitudes for « Big »
  - Distributed System challenges
  - Scaling Storage Resources: Hdfs
  - Scaling Compute Resources: Yarn, Kubernetes
  - Using Storage – Compute apis
  - Evolution of Hardware – Softwares – Hadoop Ecosystem, Spark
- Description of a Datalake
  - Content
- **Usages**
  - Example of Data-Processing
    - RAW to LAKE, Reports
  - Evolution to Cloud



# Governance

## Data Directories Organization

### Read-Write Permissions, Quota, RDPG ...

How would you organize Read-Write Permissions on  
6000 tables x 1000 jobs ?

Project ask access  
AND describe « Datasets »  
( data owner)  
=> Validation by DataOffice  
=> Create domain sub-dirs  
+ granted recursive  
WRITE permission

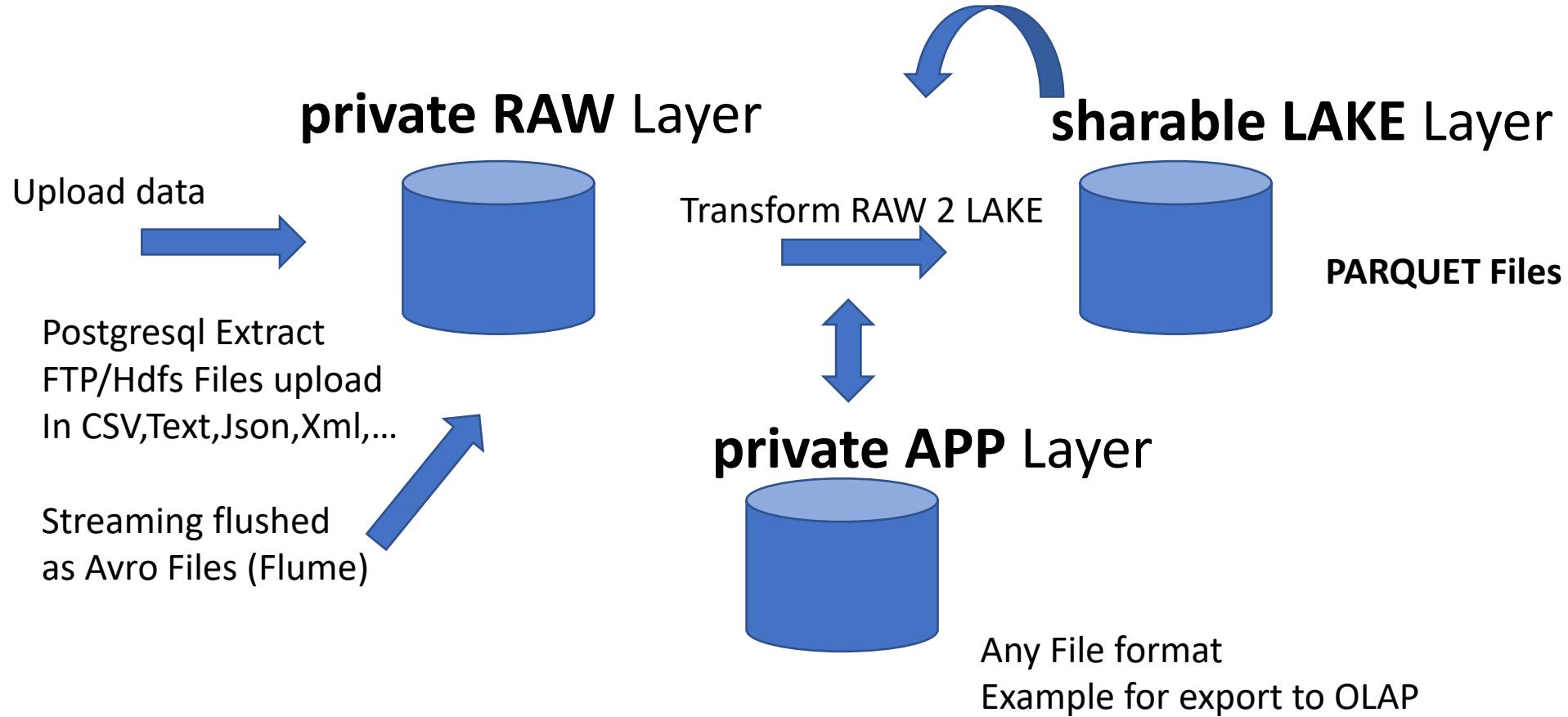
```
/ <root HDFS>
  /env
    / {raw|lake}
      /dir =by Team
        /subDir =by domain
          /sub-subDir=Hive table
            /hive partitions
              /*.parquet
```

Project ask for a « Usage » Job  
Explain Inputs/Output  
=> Validation by DataOffice  
=> Grant recursive  
READ permission for inputs

# LAKE = Parquet

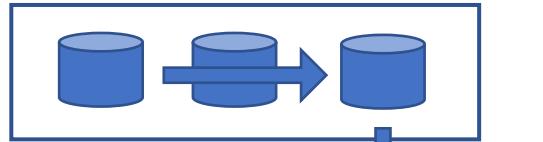
## Starting from RAW: Postgresql or {CSV|Json|Avro..}

### STAGING Dirs

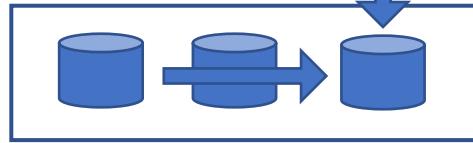


# Feeding – Transforming – Consuming Data

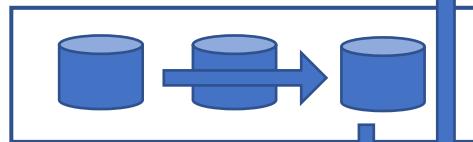
Project 1  
( Feeder only)



Project 2  
( Feed + Enrich)



Project 3



Project 4  
( Transform  
only)



Project 5  
( consume  
Only, for reports )



Every project is granted  
3 Read-Write Databases: RAW, APP, LAKE

LAKE are sharable  
... designed to be efficiently RE-read after (for analytical)

Every project can request Read access on others LAKE  
( after validation by Data-Office )

A Dataset is writable by only 1 Project (the owner )

# Big Processing

$\geq 1000$  periodic jobs running daily / or every hours

Some processing take  $\geq 15$  Hours

Some processing are distributed on more than 200 yarn containers

Some take more than 1000 Go of RAM

# Few Interactive Queries Processing screenshot Zeppelin

The screenshot shows a Zeppelin notebook titled "Untitled Note 1" with three completed cells. Each cell displays a SQL query, its execution time, and the resulting data.

- Cell 1:**

```
spark.sql("select max() from lake_fact_fpx_flat_usd_v2").show(10)
```

FINISHED | X | UI | ⚡

[database] table[isTemporary]  
[lake\_fact\_fpx\_flat\_usd\_v2] false  
[lake\_fact\_fpx] false

Spark Application ID: application\_1655792426681\_0013  
Spark Web UI: [http://mo-azt01.sqazureprod.netmicrosoft.com:8080/proxy/application\\_1655792426681\\_0013/](http://mo-azt01.sqazureprod.netmicrosoft.com:8080/proxy/application_1655792426681_0013/)  
Spark UI was updated by user at 2022-06-21 16:00:44 UTC (1 minute ago)
- Cell 2:**

```
spark.sql("select max(createdDate) from lake_fact_fpx_flat_usd_v2").show(100, False)
```

FINISHED | X | UI | ⚡

[max(createdDate)]  
[2022-06-21 22:54:48.295]

Spark Application ID: application\_1655792426681\_0013  
Spark Web UI: [http://mo-azt01.sqazureprod.netmicrosoft.com:8080/proxy/application\\_1655792426681\\_0013/](http://mo-azt01.sqazureprod.netmicrosoft.com:8080/proxy/application_1655792426681_0013/)  
Spark UI was updated by user at 2022-06-21 16:00:44 UTC (1 minute ago)
- Cell 3:**

```
spark.sql("select distinct product_id from lake_fact_product_orderDetails where (productCategory='Electronics')").show(100, False)
```

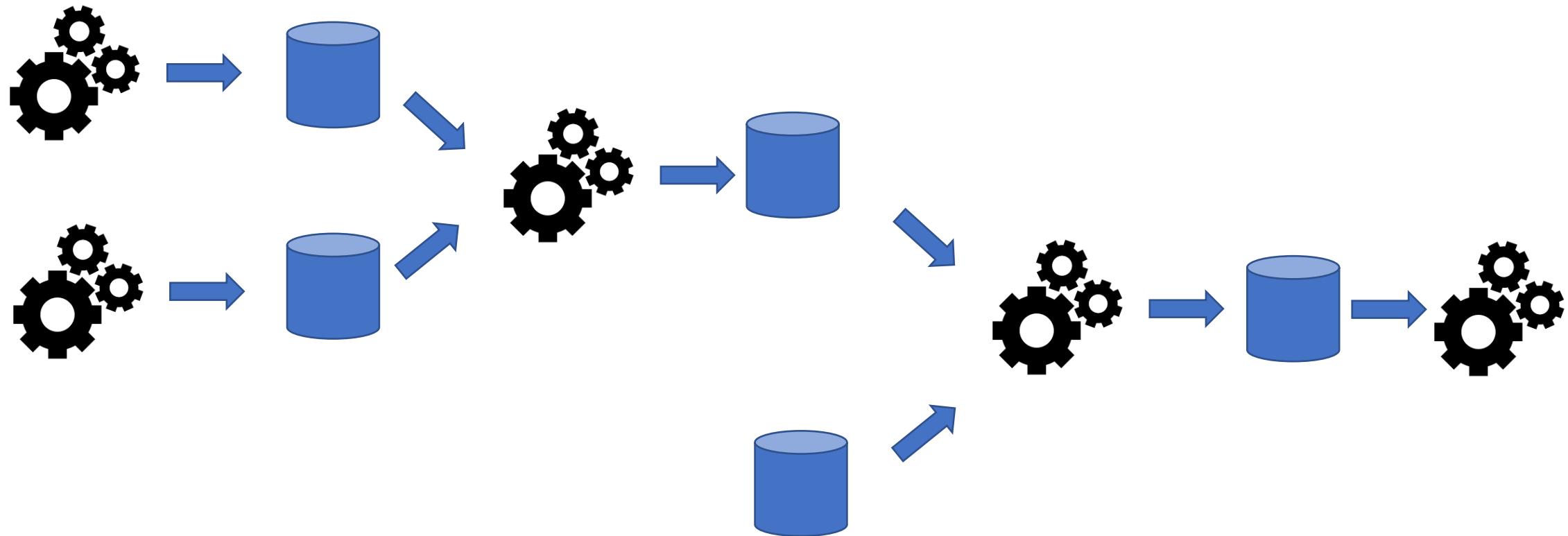
FINISHED | X | UI | ⚡

[distinct product\_id]  
[Electronics]

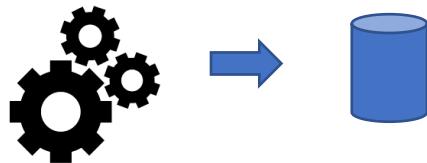
Spark Application ID: application\_1655792426681\_0017  
Spark Web UI: [http://mo-azt01.sqazureprod.netmicrosoft.com:8080/proxy/application\\_1655792426681\\_0017/](http://mo-azt01.sqazureprod.netmicrosoft.com:8080/proxy/application_1655792426681_0017/)  
Spark UI was updated by user at 2022-06-21 16:00:44 UTC (1 minute ago)

# Orchestration / DataLineage

## Data and Workflows are linked



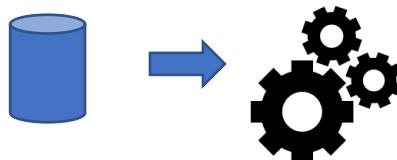
# Read-Write Permissions from DataOffice



Dataset « Feeding » declaration:

= exclusive Write Permission : Project -> canWrite -> DB

For validation by DataOffice, DataSet must be described :  
confidentiality, source, business owner, quality, RDPG legal



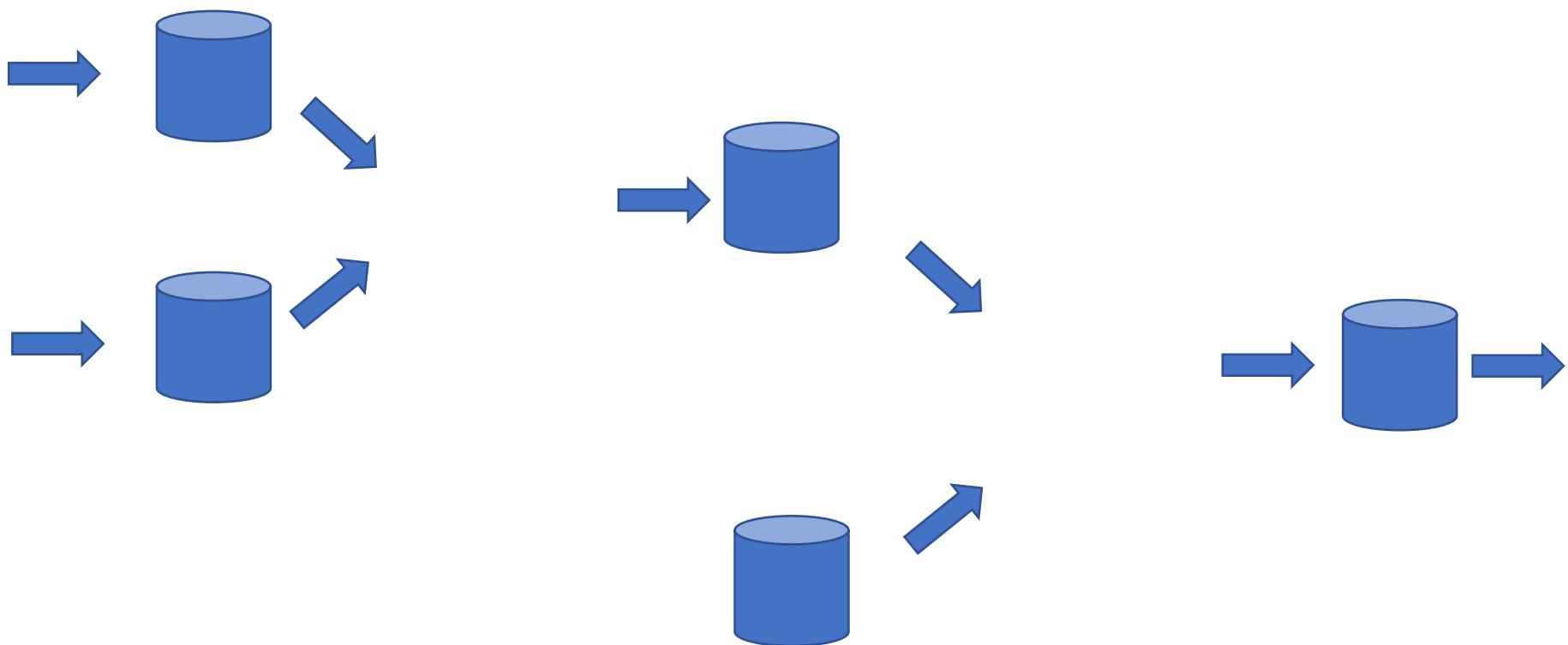
Consumer declaration:

= authorization to read : Project -> canRead -> DB  
... for a specific Usage

For validation by DataOffice, use-case must be described,  
usage must be « legitimate », RDI legal

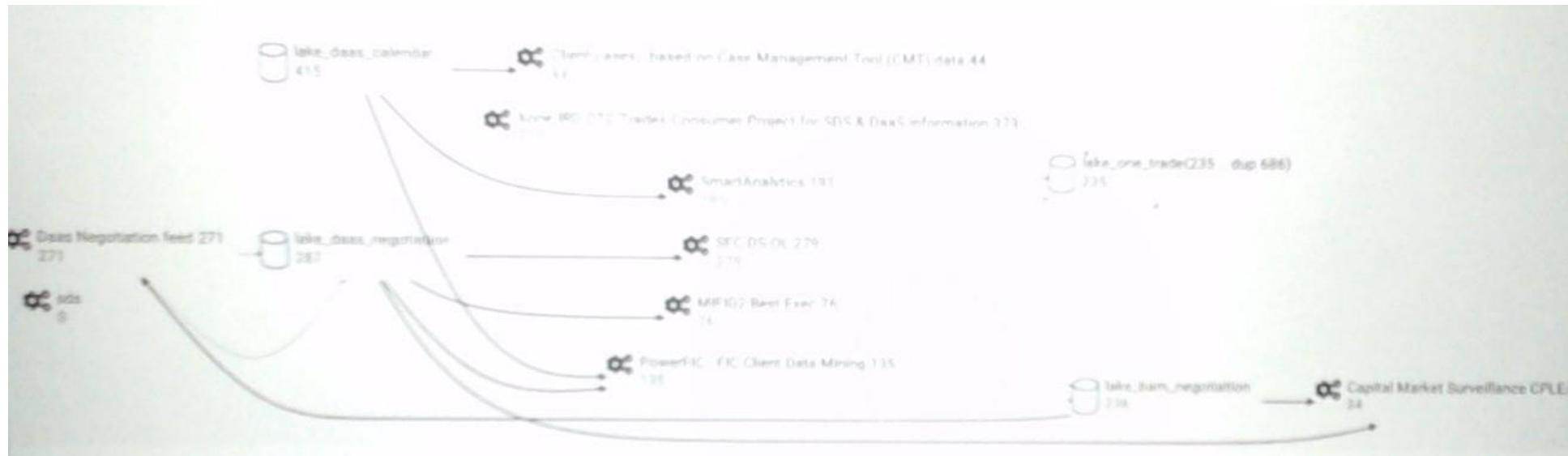
# DataLineage

## Data depends on other Sources Data



# Screenshots (Fuzzy.. On purpose) Read/Write Dependencies Hive Database / Job...

Zooming on 4 out of 250 Hive databases

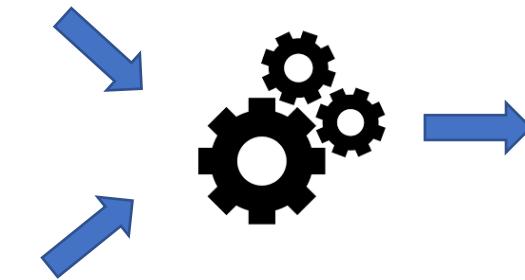
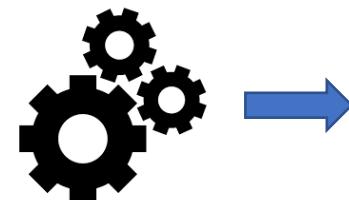
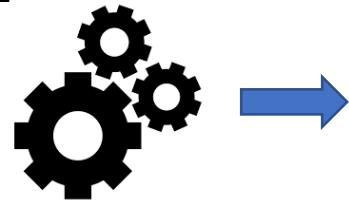


# Orchestration

## Start process2 after process1 finished

Start condition=

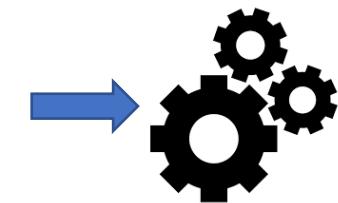
CRON 1 \* \* \* \* \*



Start condition=  
Wait for data...



Start condition=  
CRON 10 \* \* \* \* \*



# Prod Screenshot (Fuzzy.. on purpose)

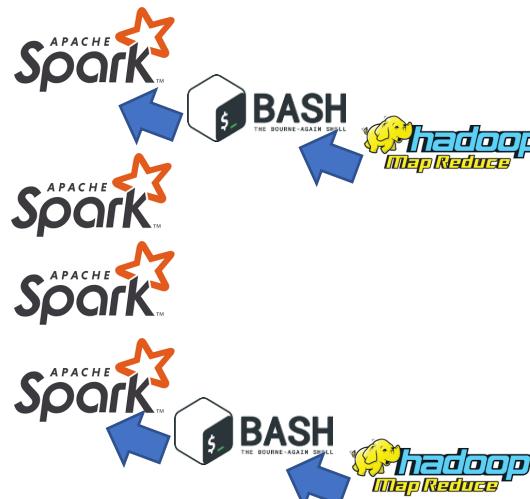
## Running Yarn Applications

The screenshot shows the Hadoop YARN web interface with the title "RUNNING Applications". The left sidebar includes links for Cluster Metrics, Scheduler Metrics, and a Scheduler section. The main table lists several applications:

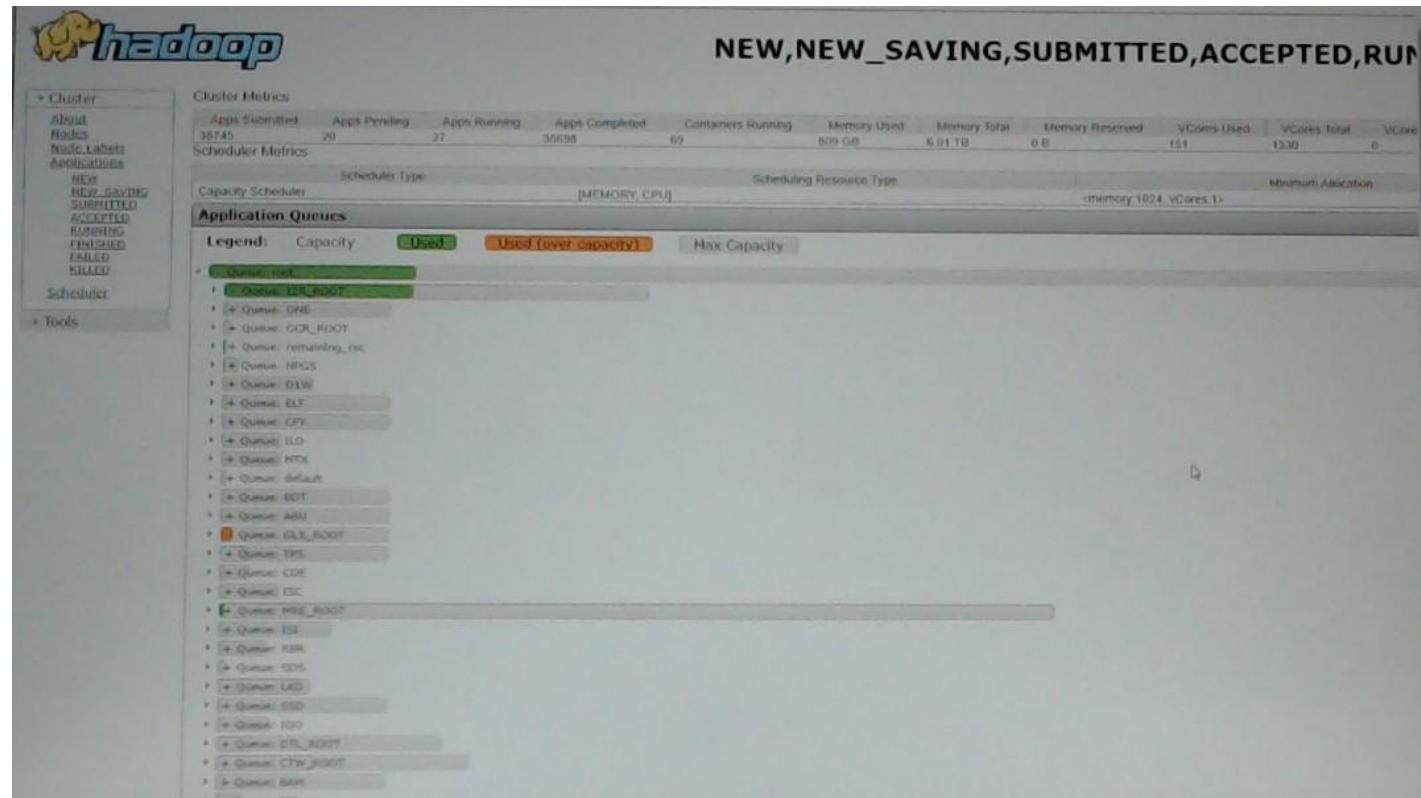
ID	User	Name	Application Type	Queue	Application Priority	StartTime	FinishTime	State
application_1564573480572_38713	hadoop01	ozone launcher T-shell W=prtd_llta_llre_cals_Reprocess.A=cals_llre_process ID=0007972	MAPREDUCE	FDA	0	Thu Oct 11 09:30:00 +0200 2022	N/A	RUNNING
application_1564573480572_38713	hadoop03	prtd_command-stats-high	SPARK	l2R_prtd_var	0	Thu Oct 13 05:28:16 +0200 2022	N/A	RUNNING
application_1564573480572_38729	hadoop03	ozone launcher T-shell W=prtd_command_stats-high A=shell ID=0007861	MAPREDUCE	l2R_prtd_var	0	Thu Oct 13 05:22:55 +0200 2022	N/A	RUNNING
application_1564573480572_38729	gtprtd09	SQMarketData branch origin MDL-481-Manu2-FDR at 1202163120 commit 32745db	SPARK	GLX_default	0	Thu Oct 13 04:01:37 +0200 2022	N/A	RUNNING
application_1564573480572_38746	gtprtd09	BIG Persec GetHistory RunJobPerSecHistCorrections	SPARK	GLX_default	0	Wed Oct 12 16:17:08 +0200 2022	N/A	RUNNING
application_1564573480572_38746	gtprtd09	com.socgem.staticdata.RunPersecStatic	SPARK	GLX_default	0	Wed Oct 12 16:16:09 +0200 2022	N/A	RUNNING
application_1564573480572_38772	gtprtd09	ozone launcher T-shell W=bulk-2.7-A=processed_bals_Bms ID=0007286	MAPREDUCE	GLX_HOLA	0	Wed Oct 12 13:42:31 +0200 2022	N/A	RUNNING

MAPREDUCE ?

... only internally to launch shell commands.. To spark  
ALL others = SPARK

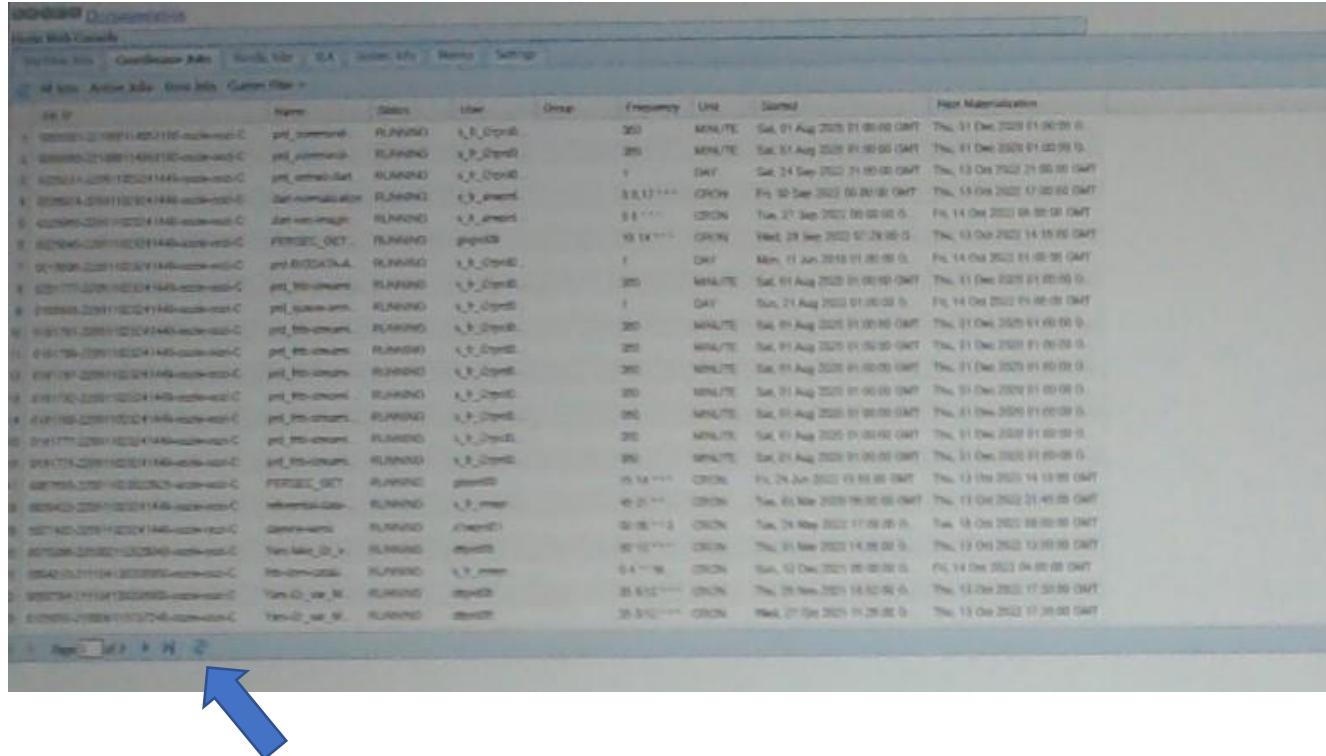


# Prod Screenshot : Yarn Queues (Scheduler, CPU+Mem usages)



You see scrollbar?...  
Hundreds or Yarn Queues  
+ dozen of sub-queues

# Screenshot Oozie (Coordinators / Workflows)



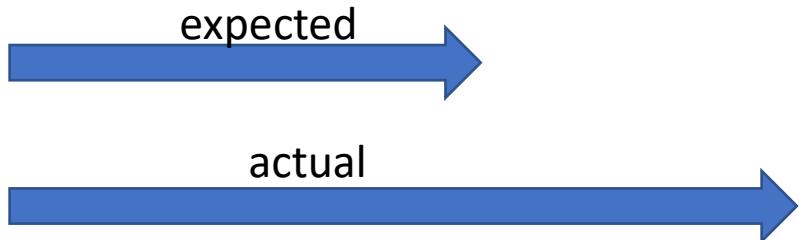
You see pagination?...  
Thousands of CRON jobs

Oozie display only last 50 000 workflows  
... so only 2 days of history

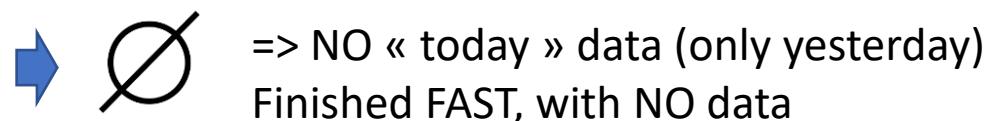
# Orchestration « By CRON: 10 \* \* \* \* »

## Example of Possible Failures

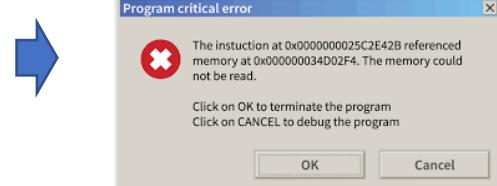
batch1 started at 1 \*\*\* ... but takes today (anormally) 15h instead of 10h



batch2 started at 12 \*\*\* .... data not yet present



batch2 started, during critical section of batch1 ... table dropped/recreated after



# Outline

- What is BigData ?
    - Order of Magnitudes for « Big »
    - Distributed System challenges
    - Scaling Storage Resources: Hdfs
    - Scaling Compute Resources: Yarn, Kubernetes
    - Using Storage – Compute apis
    - Evolution of Hardware – Softwares – Hadoop Ecosystem, Spark
  - Description of a Datalake
    - Content
    - Usage
  - Example of Data-Processing
    - **RAW to LAKE, Reports**
  - Evolution to Cloud
- 

# Typical Example of adding 1 daily partition

```
$ hdfs dfs –mkdir hdfs://raw/team/domain/table/date=2022-10-12
```

```
$ hdfs dfs –put localFile hdfs://raw/team/domain/table/date=2022-10-12/
```

```
$ spark-sql –master local[1] \  
  –e « ALTER TABLE raw_lake_domain.table  
    ADD PARTITION ( date='2022-10-12' ) »
```

# Typical RAW to LAKE processing with Spark as Java code

```
read {  
    spark.read  
        .format("csv")  
        .option("schema", "col1 type1, ... colN typeN")  
        .load("hdfs://raw/team/domain/table/date=2022-10-12")  
  
transform {  
    .map(row -> transformData(row))  
  
write {  
    .repartition(3, "col1")  
    .sortWithinPartition("col1, col2, col3")  
    .write  
        .format("parquet")  
        .save("hdfs://lake/team/domain/table/date=2022-10-22")  
    ;
```

# Typical RAW to LAKE processing with Spark as SQL code

```
write    { INSERT OVERWRITE
            lake_team_domain.table
            SELECT /* +REPARTITION(col1, 3) */
                    col1, col2,
            transform { udf_func1(col3, col4) as col3,
                        udf_func2(col4, col5) as col4,
                        ..
            read     { FROM
                        raw_team_domain.table
            transform { JOIN
                        lake_anotherTeam_domain.anotherTable x ON x.ID=id
            read     { WHERE date='2022-10-22' AND ..
            write   { SORT BY col1, col2, col3 -- idem sortWithinPartition
```

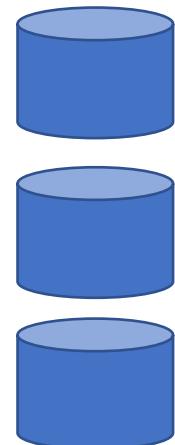
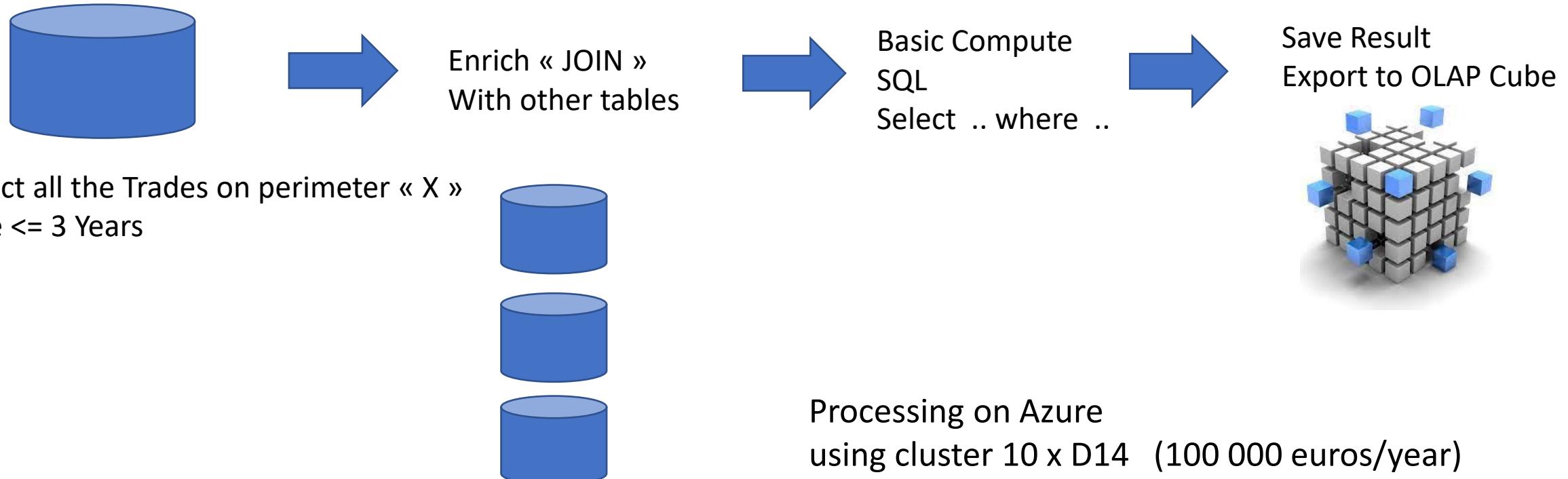
# Example of LAKE Aggregation

```
INSERT OVERWRITE
    lake_team_domain.table
SELECT * FROM (
    SELECT * FROM table1 WHERE ..
    UNION
    SELECT * FROM table2 WHERE ..
    UNION
    SELECT * FROM table3 WHERE ..
    UNION
    SELECT * FROM table4 WHERE ..
)
SORT BY col1, col2, col3 -- idem sortWithinPartition
```

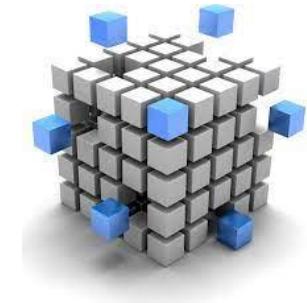
# Example of « latest value » cristalisation analytical query « over(partition by) »

```
INSERT OVERWRITE
    lake_team_domain.table
SELECT
    col1,col2,... colN    -- idem * EXCEPT rank (cf issue SPARK-33164)
FROM (
    SELECT *,
        RANK() OVER (PARTITION BY id ORDER BY update_time DESC) as rank
    FROM lake_team_domain.event_table
)
WHERE rank=1
SORT BY col1, col2, col3    -- idem sortWithinPartition
```

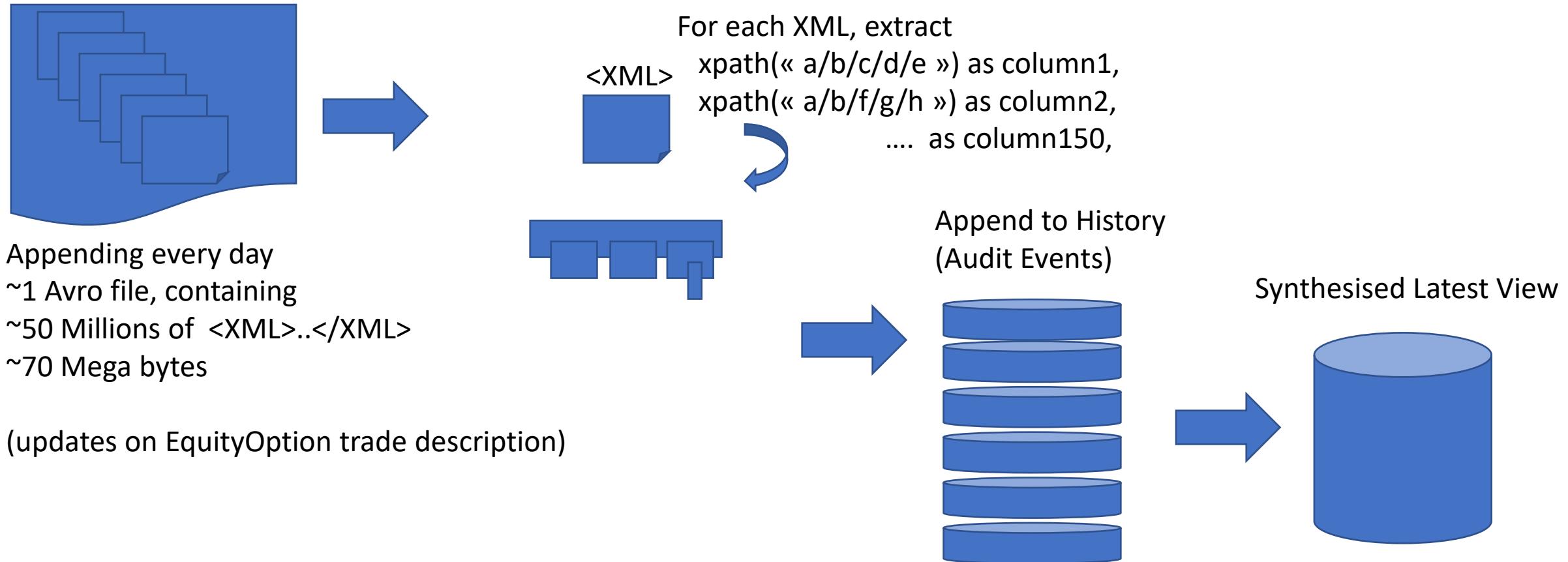
# Example of a « Small » Data Processing resource needed = RAM



Processing on Azure  
using cluster 10 x D14 (100 000 euros/year)  
ONLY 600 Giga of RAM data ...  
NOT Enough => Swapping => take 4 Hours



# Example of a Slow « Un-parallelized » Processing resource needed = CPU x Time x ..

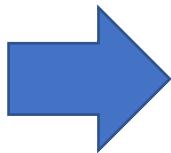


# A Slow Processing ?

Legacy « On-Prem »

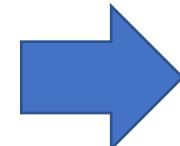
Daily Batch takes  $\geq$  15 Hours  
But is mostly SINGLE-THREADED !!!

Some Parallelized intermediate parts take  
 $\geq$  200 yarn containers ... 10% of Cluster  
... Result = ULTRA FAST !!



Backported code « as-is » to Azure HDInsight  
Used Cluster of 10 x D14 cluster (40 000 euros/month)

Run NEVER Finished  
... stopped in « echo count lines before insert \$(... ) »



Fully re-implemented in Spark + Java  
instead of Hive SQL + UDF xpath  
+ Optimized ...

Now run in 40 minutes on 2 x D13 cluster

## Take Away

At SG, most workflows are uploading daily files + processing by daily batches

Except 1 Huge stream feeding

Almost all use Spark

- |                |   |
|----------------|---|
| Spark Supports | <ul style="list-style-type: none"><li>- Interactive queries</li><li>- Batch mode</li><li>- Streaming mode</li></ul> |
|----------------|---|

Coffee Break

# Outline

- What is BigData ?
  - Order of Magnitudes for « Big »
  - Distributed System challenges
  - Scaling Storage Resources: Hdfs
  - Scaling Compute Resources: Yarn, Kubernetes
  - Using Storage – Compute apis
  - Evolution of Hardware – Softwares – Hadoop Ecosystem, Spark
- Description of a Datalake
  - Content
  - Usages
- Example of Data-Processing
  - RAW to LAKE, Reports
- • Evolution to Cloud

# Appendix: Cloud

... More Detailed on

« From OnPrem to Azure Cloud »

see Document « -cloud.pdf »

# Appendix: Cloud

# Fundamental Resources TradeOffs

Network (Bandwidth, Latency)



Horizontal Scale



CPU (HOT, Watts)



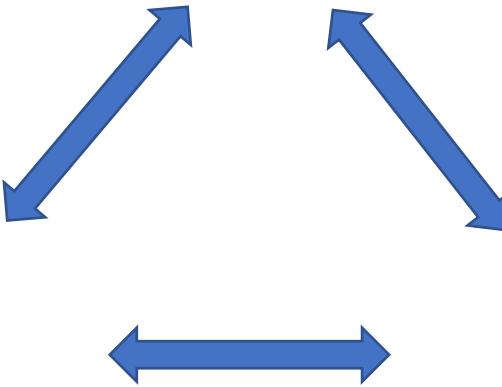
Development price / Run cost



Cold Storage  
(SLOW)



Storage (~Small)



RAM (FAST .. Expensive)



# Performance Changes -> Architecture Changes

**Historically:**

SLOW Disks

Disk Collocated with RAM+CPU

MapReduce « send program » collocated to data

**Now:**

FAST(ER) SSD Disks + Networks

Can « send» data to distributed programs



**Storage and Compute can be SPLIT ... OK !**

Higher Density for Huge Storage, lower prices



48 \* disks in 2U blade



1 Peta SSD ... 200 000 €

Cloud Provider : AWS, Azure, Google  
offer Storage solutions (S3, Azure Storage, ..)



- + Managed Services
- + Serverless
- + Kubernetes

# Migration to Cloud

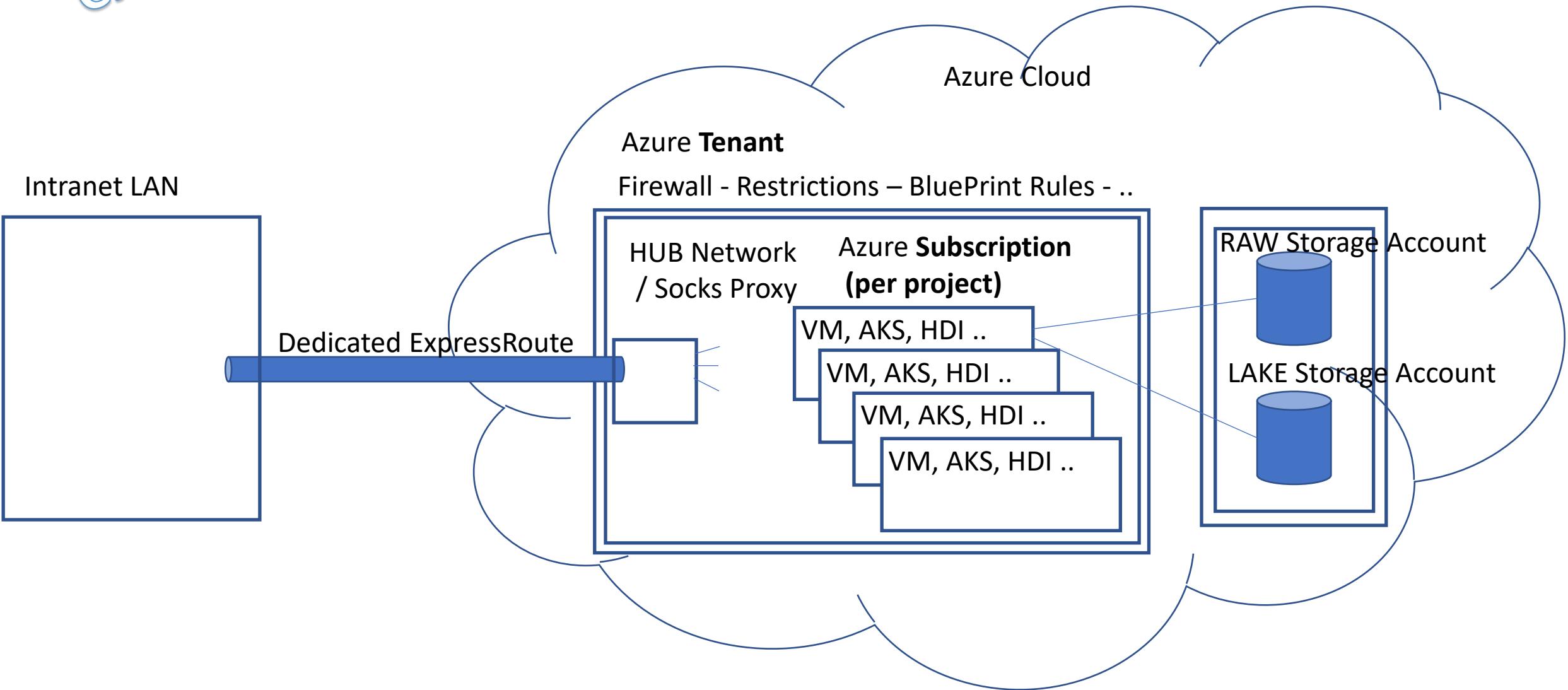
## Azure - AWS - Google

### Goals:

- Elasticity of Storage  
(No more fear of HDFS FileSystem Full)
- Elasticity of Compute  
(adapt CPU to workload... Pay only what you use)
- Clear visibility of cost per Projects / internal refacturation
- By-Pass internal IT department
- Easier (?) Self-Service API for Provisionning
- No More Multi-tenant (no risk of 1 project crashing/consuming whole cluster)

Since 2020 SGCIB is moving its Datalake to Azure

# Architecture of Datalake on Clouds



# BigData Engineering from OnPrem to Azure



Development of custom Yarn/Oozie/Ranger tools

1 Huge cluster, used as Multi-tenant (several users)

Used at 100%

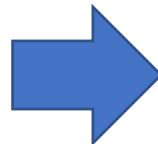
Almost Disk-Full  
Many report on disk usage / data purges

Clear view of all Running workloads  
(1 Ambari screen)

Lot of Network & Security  
Development of Provisionning « sudo » tools

NO more multi-tenant system... but LOT of small clusters

Hundred of clusters, each used at 5% !!!  
COST COST COST  
Necessity to Optimize Performance  
( batches too long... 100x slower / too expensive)



No more Disk worry  
... data is growing (no more purges?)

NO admin central view  
NO Orchestration of workloads

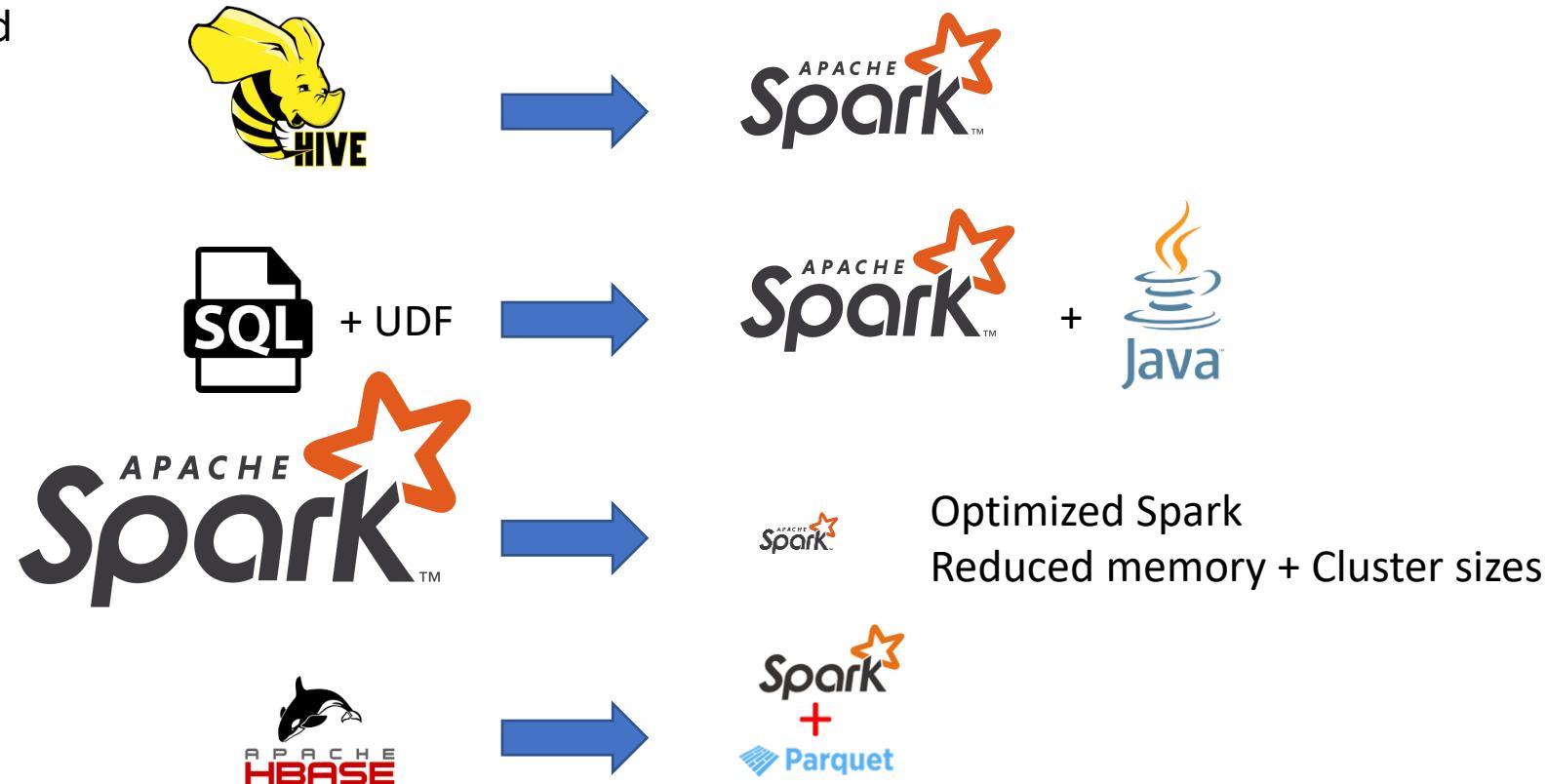
# What About Spark in Azure Migration?

COST COST COST

**Necessity to Optimize Performance**

( batches too long... 100x slower / too expensive)

=> Many projects have  
migrated + optimized



# Managed Spark ?

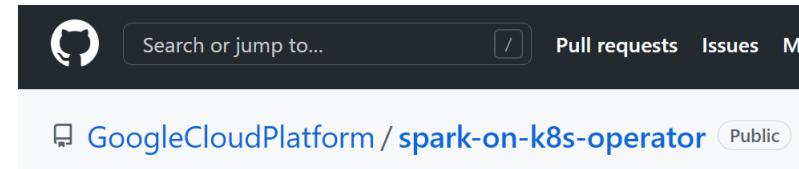
## Goals: Autoscaling / « serverless »

Managed by Azure :



Azure Synapse Analytics

Managed via Kubernetes



kubernetes

databricks



databricks

AWS : EMR

GCP : BigQuery

Questions?

# Take Away

What is BigData ? Horyzontal Scaling  
compute: cluster with Tera of RAM used by Spark apps  
storage: Petas of Files, in parquet

What is Spark ?  
Simple unified Sql/Java engine  
for distributed compute (Yarn/Kubernetes)  
distributed storage (HDFS/cloud)

What is Processing ?  
mostly spark batches  
Feeding RAW  
Transforming RAW to LAKE  
Consuming SQL analytics

Hadoop ecosystem is complex, Spark brings simplicity  
Ecosystem is evolving (Cloud, Kubernetes)