Big Data Analytics

Iza Moise, Evangelos Pournaras

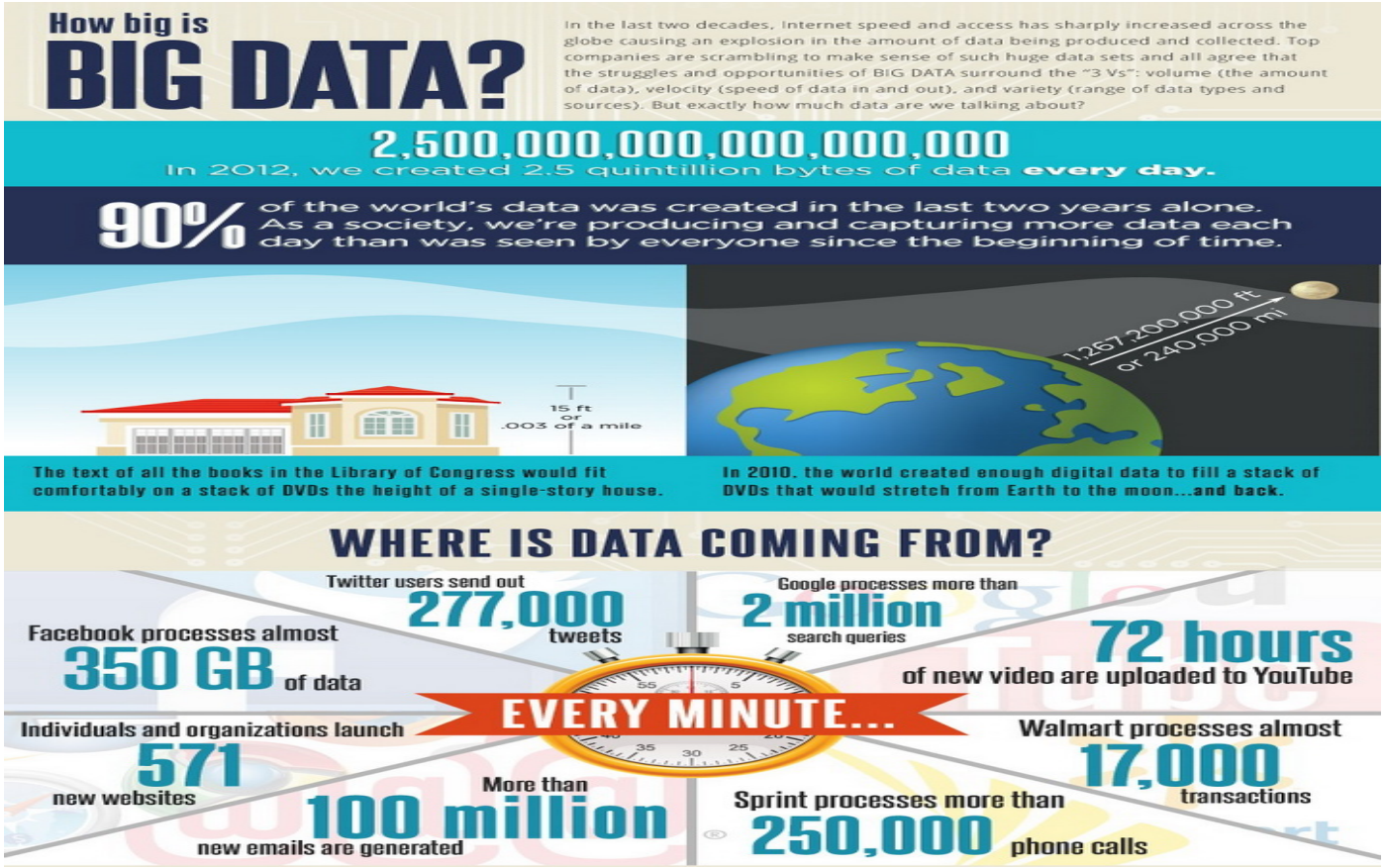
What is Big Data?

"Big Data refers to datasets whose size is beyond the ability of typical database software tools to capture, store, manage and analyze." (*McKinsey Global Institute*)

"Big Data is the term for a collection of datasets so large and complex that it becomes difficult to process using on-hand database management tools or traditional data processing applications." (*Wikipedia*)

Big Data Explosion

1.8 ZB of information world-wide (2011)



How *big* is Big Data?

Eric Schmidt: “Every 2 days we create as much information as we did up to 2003.” (2011)

We created 5 billion GigaBytes (5 ExaBytes) of data.

In 2014, the same amount of data is created every 7 minutes.

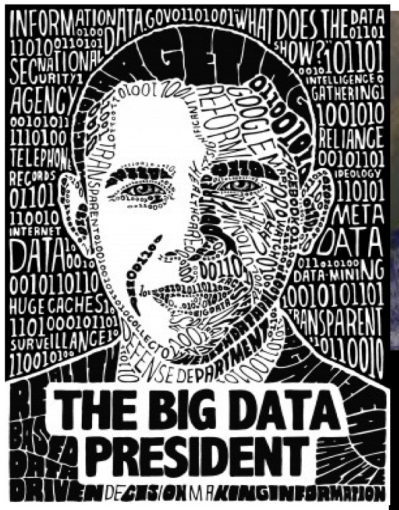
Total size of the Digital Universe in 2014: 4.4 ZetaBytes.

Every minute of every day

- More than 204 million email messages
- Over 2 million Google search queries
- 48 hours of new YouTube videos
- 684,000 bits of content shared on Facebook
- More than 100,000 tweets
- $272,000 spent on e-commerce

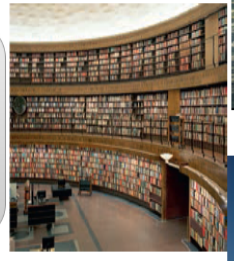
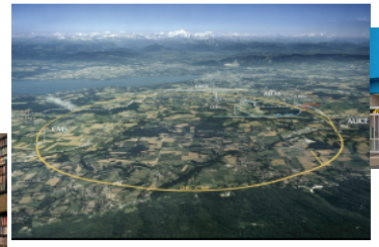
Big data becomes Real data

- Big Data became real in 2013
- Obama "the Big Data President"
- Oscar prediction
- Finding and telling data-driven stories in billions of Tweets

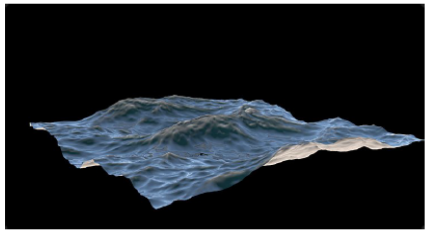


Sources



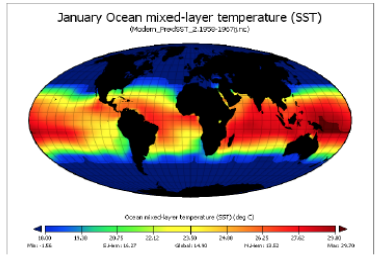





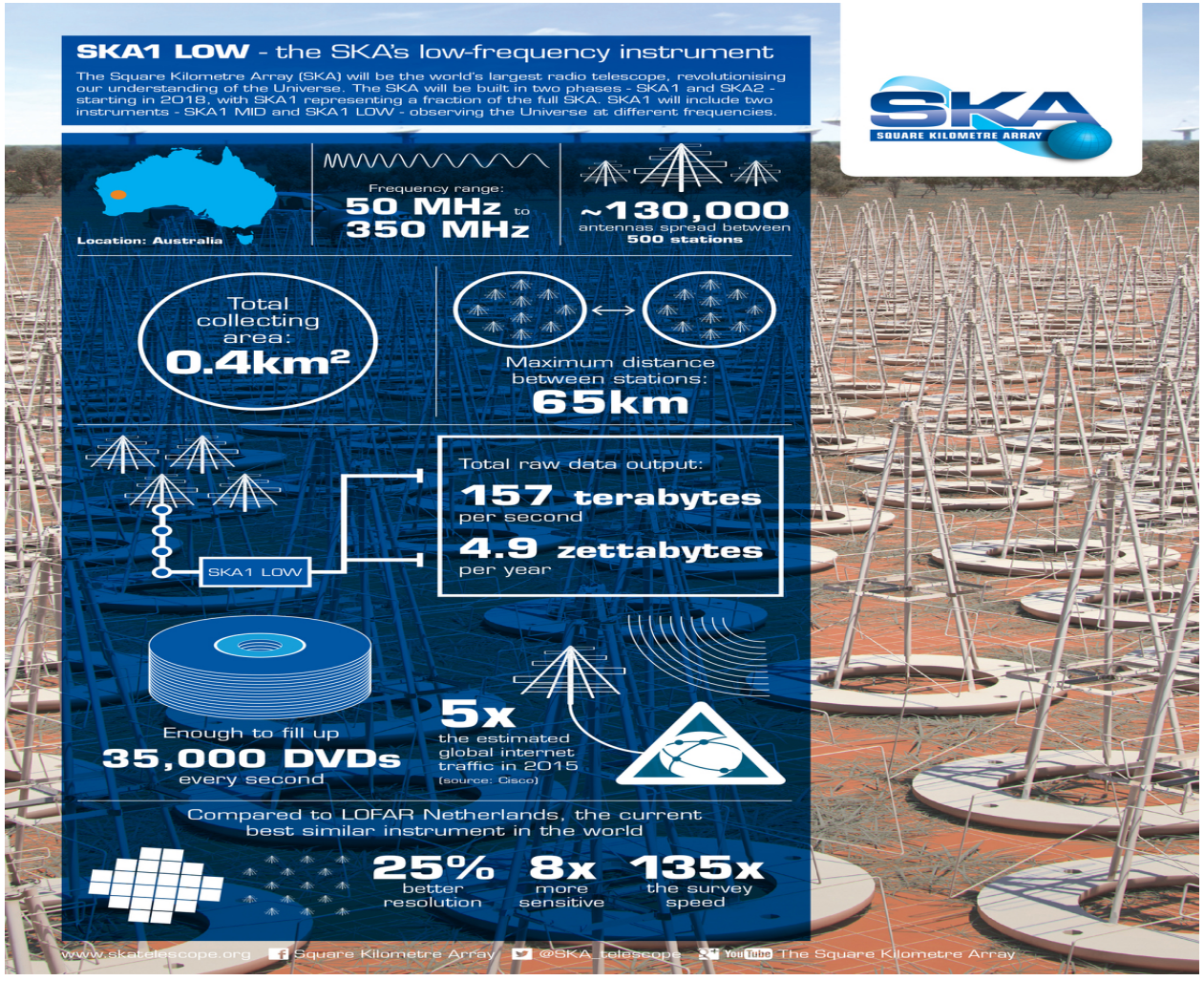




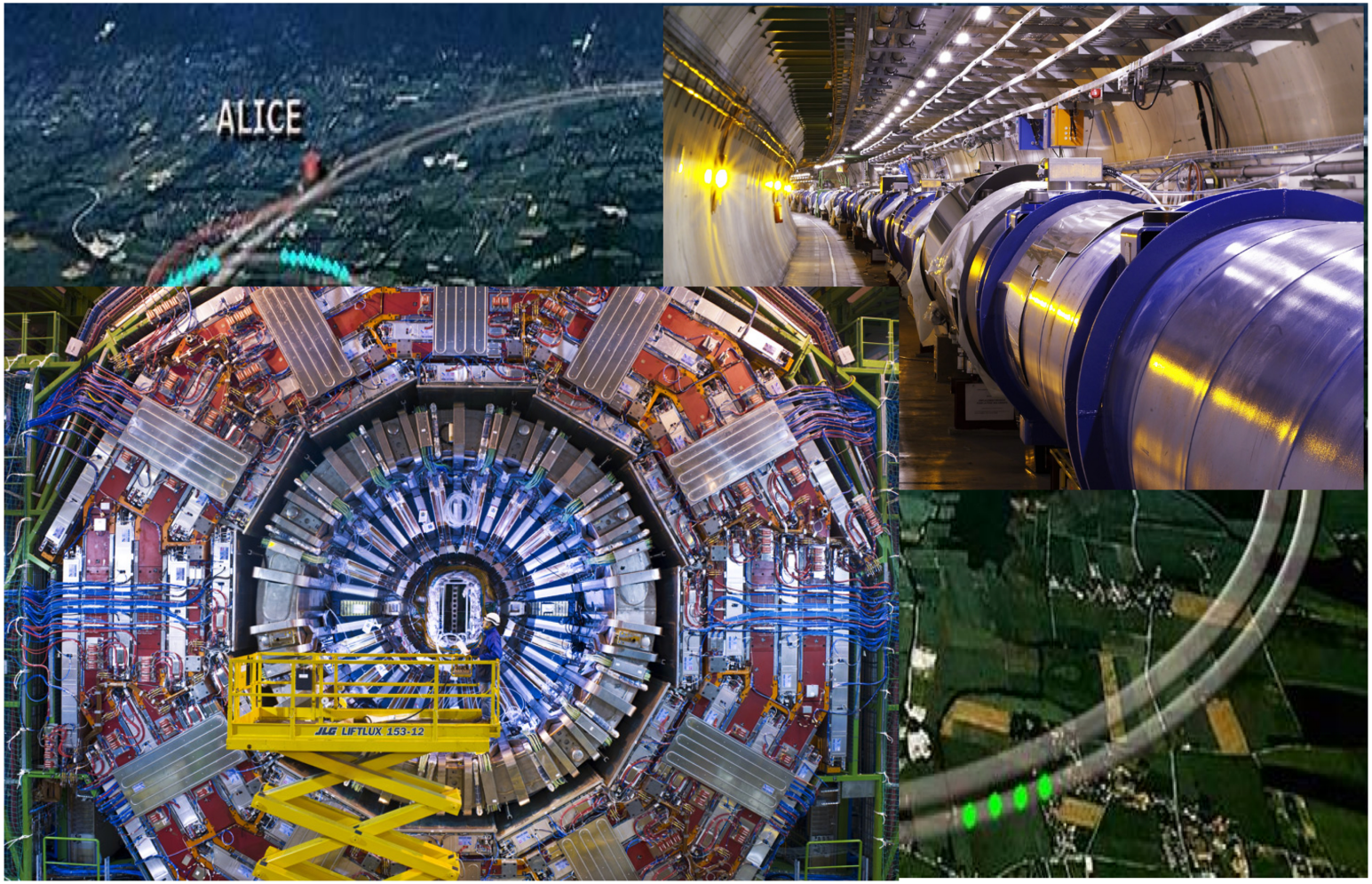




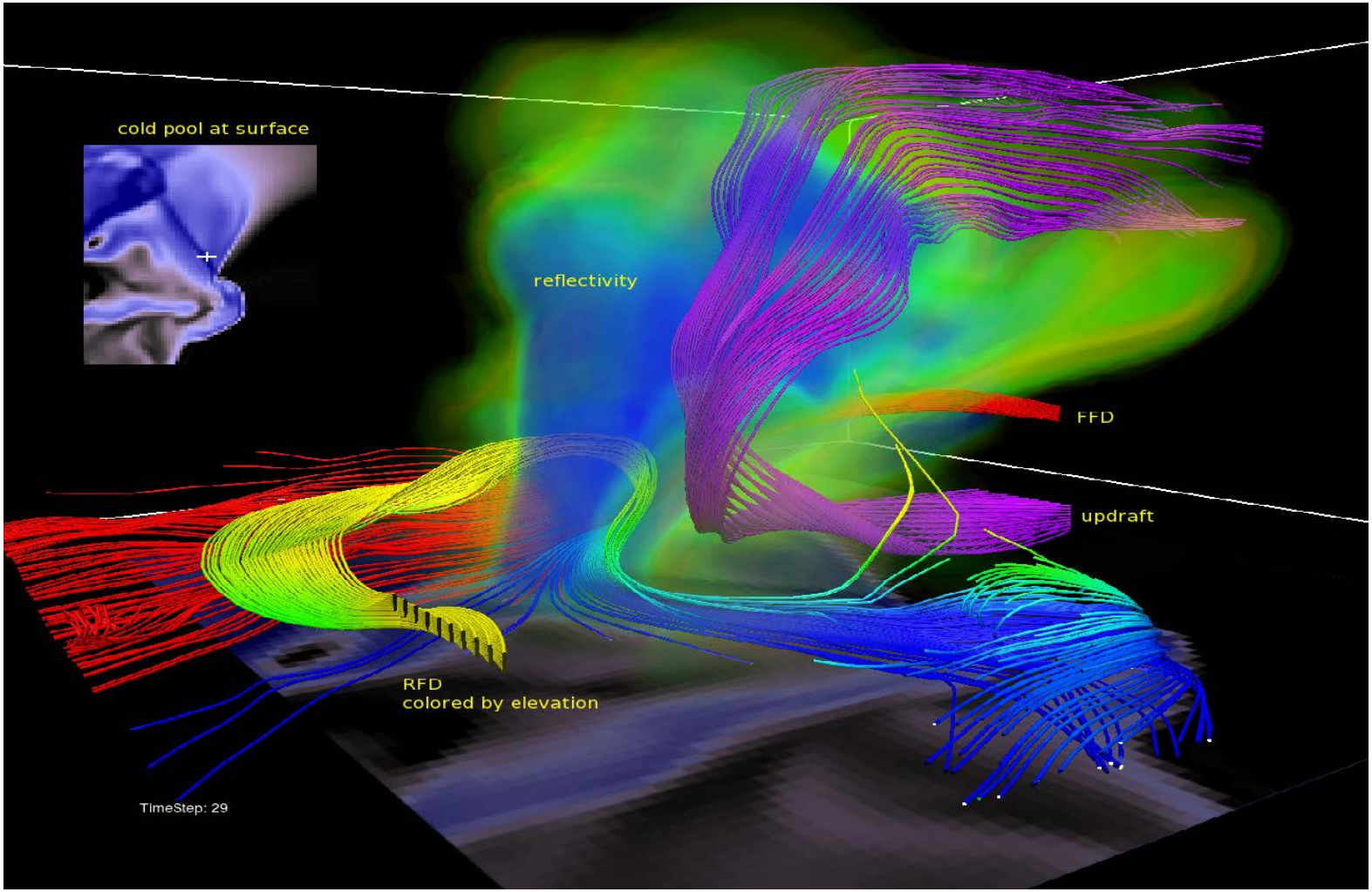

Square Kilometer Array



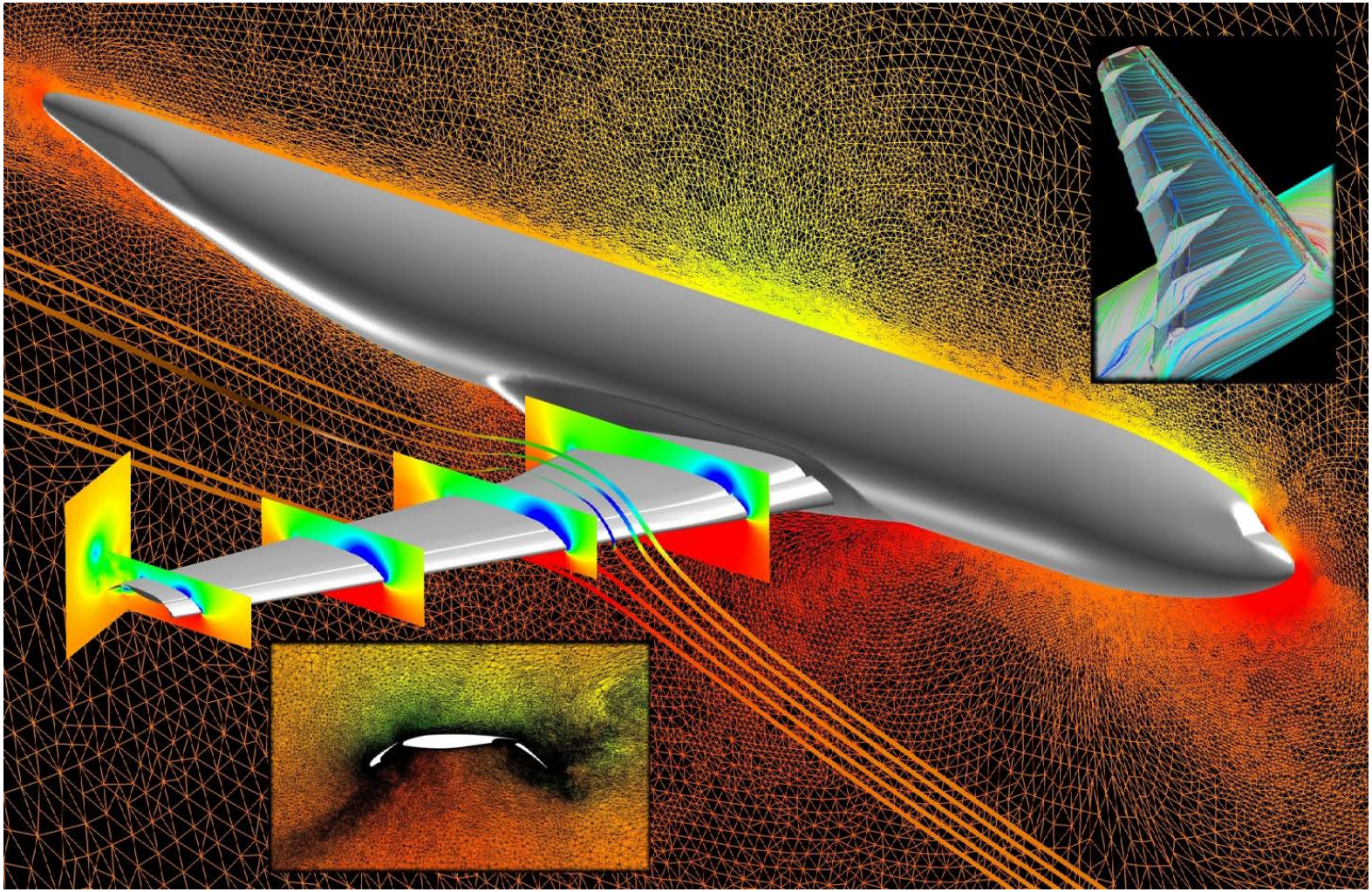
Large Hadron Collider



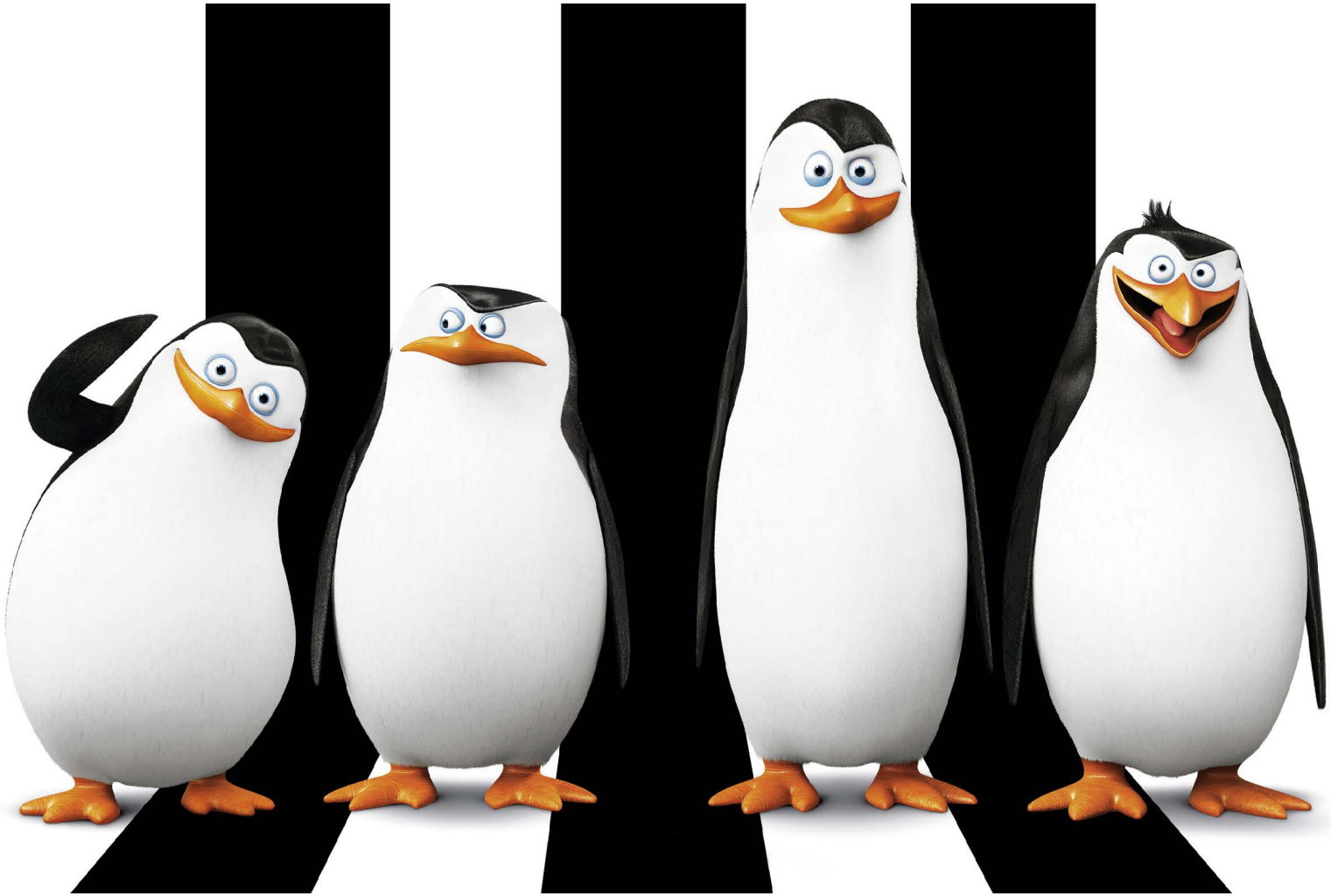
Climate Simulations



Aviation Simulations



3D Rendering for Animations



How to train your dragon

- 10,000 simultaneous computing cores
- 75 million computing hours
- 250 TB of active disk space for storage
- one movie = 250 billion pixels on screen

Internet data

- 1 PB of Facebook data per day
- 500 M tweets per day:
  - 200 B tweets / year
- Google processes 24 PB a day

Big picture

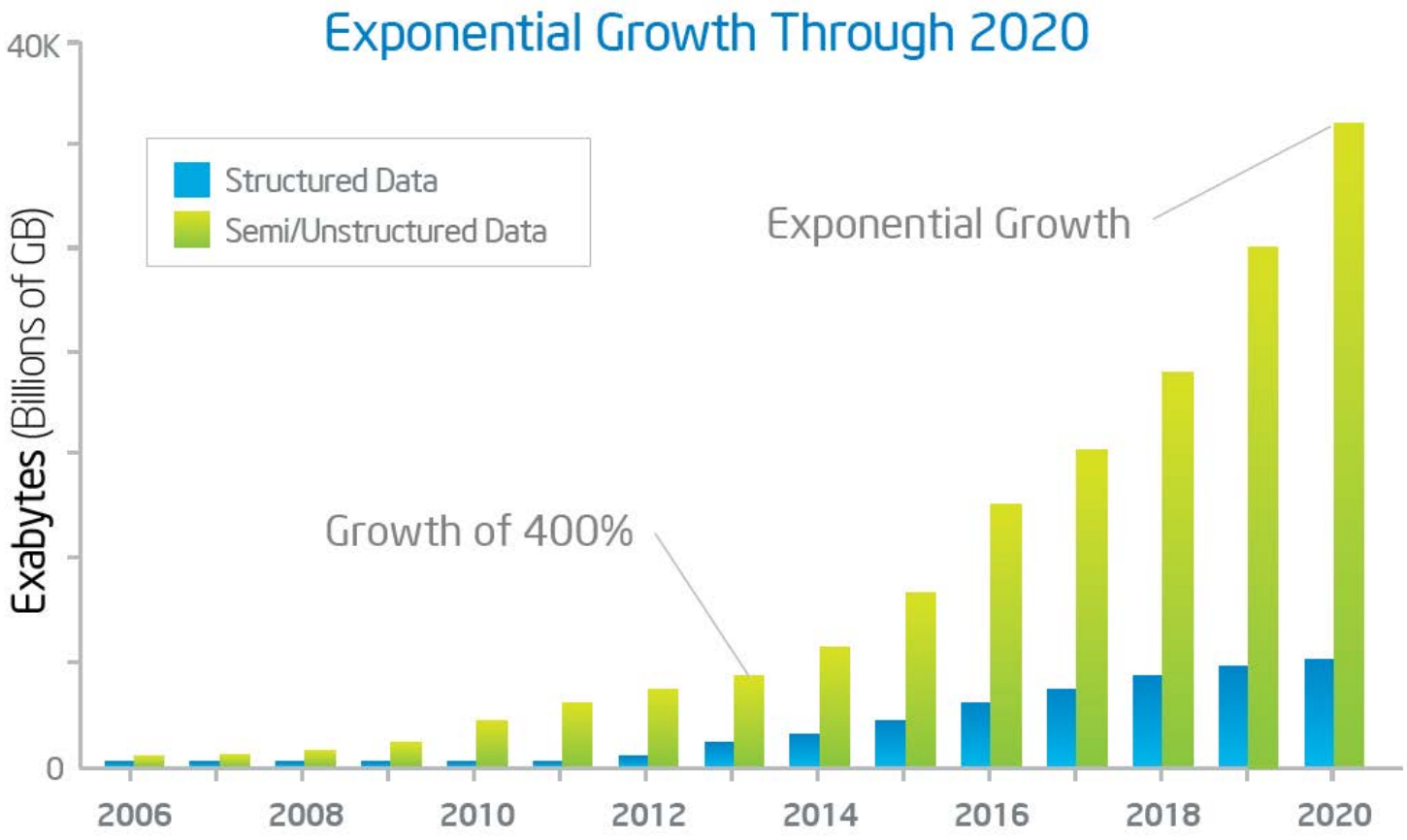
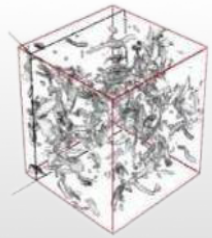
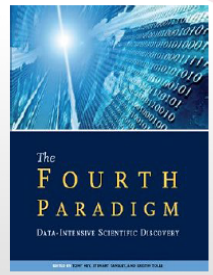


Figure 1. Current and forecasted growth of big data. Source: Philippe Botteri of Accel Partners, Feb. 2013.

The 4th Paradigm for Scientific Discovery

$$\left(\frac{\dot{a}}{a}\right)^2 = \frac{4\pi G\rho}{3} - K\frac{c^2}{a^2}$$





Experimental	Theoretical	Computational	The Fourth Paradigm
Thousand years ago	Last few hundred years	Last few decades	Today and the Future
Description of natural phenomena	*Newton's laws, Maxwell's equations…*	*Simulation of complex phenomena*	*Unify theory, experiment and simulation with large multidisciplinary Data* *Using data exploration and data mining (from instruments, sensors, humans…)* *Distributed Communities*

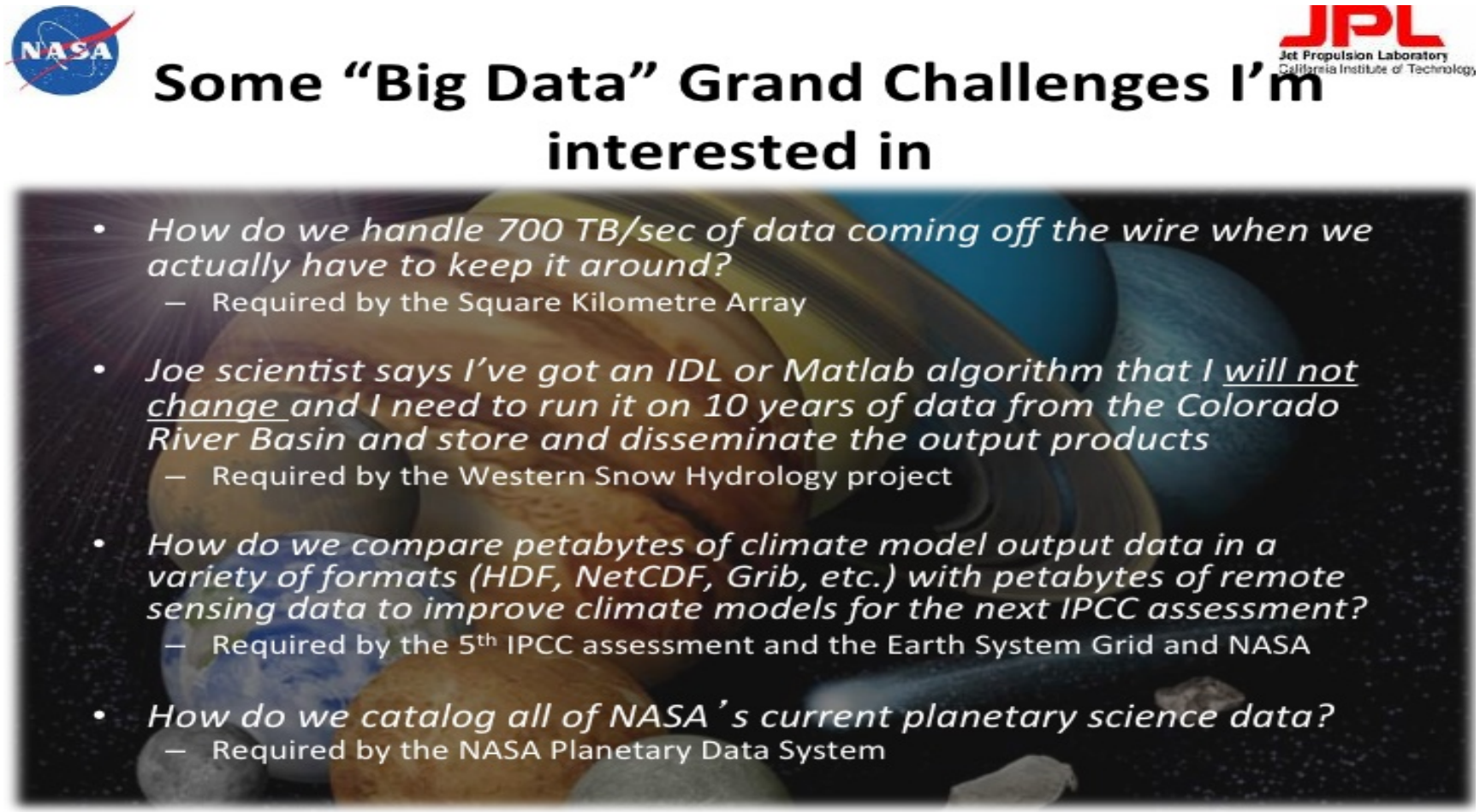
Crédits: Dennis Gannon

The First Big Data Problem

The Internet Search Engine:

In 2002, Google wanted to be able to crawl the web and index the content so that they could produce an internet search engine. The standard method to organize and store data in 2002 was by means of relational database management systems (RDBMS) which were accessed in a language called SQL. But almost all SQL and relational stores were not appropriate for internet search engine storage and retrieval because they were costly, not terribly scalable, not as tolerant to failure as required and not as performant as desired.

Big Data Problems nowadays



The 5 V's

- Volume
  - → large amounts of data generated every second (emails, twitter messages, videos, sensor data...)
- Velocity
  - → the speed of data moving in and out data management systems (videos going viral...)
  - → "on-the-fly"
- Variety
  - → different data formats in terms of structured or unstructured (80%) data
- Value
  - → insights we can reveal within the data
- Veracity
  - → trustworthiness of the data

Big data

Different Types of Data:

- → Structured data (Relational, Tables)
- → Semi Structured Data (XML, JSON, Logfiles)
- → Unstructured Data (Free Text, Webpages)
- → Graph Data (Social Network, Semantic Web)
- → Streaming Data

Typical Operations:

- Aggregation & Statistics
  - Data warehouse, OLAP
- Index, Searching, Querying
  - Keyword bases search
  - Pattern matching
- Knowledge discovery
  - Data Mining
  - Statistical Modeling

Big Data

Old model:

"Query the world": Data acquisition coupled to a specific hypothesis

New model:

"Download the world": Data acquisition supports many hypotheses

Big Data

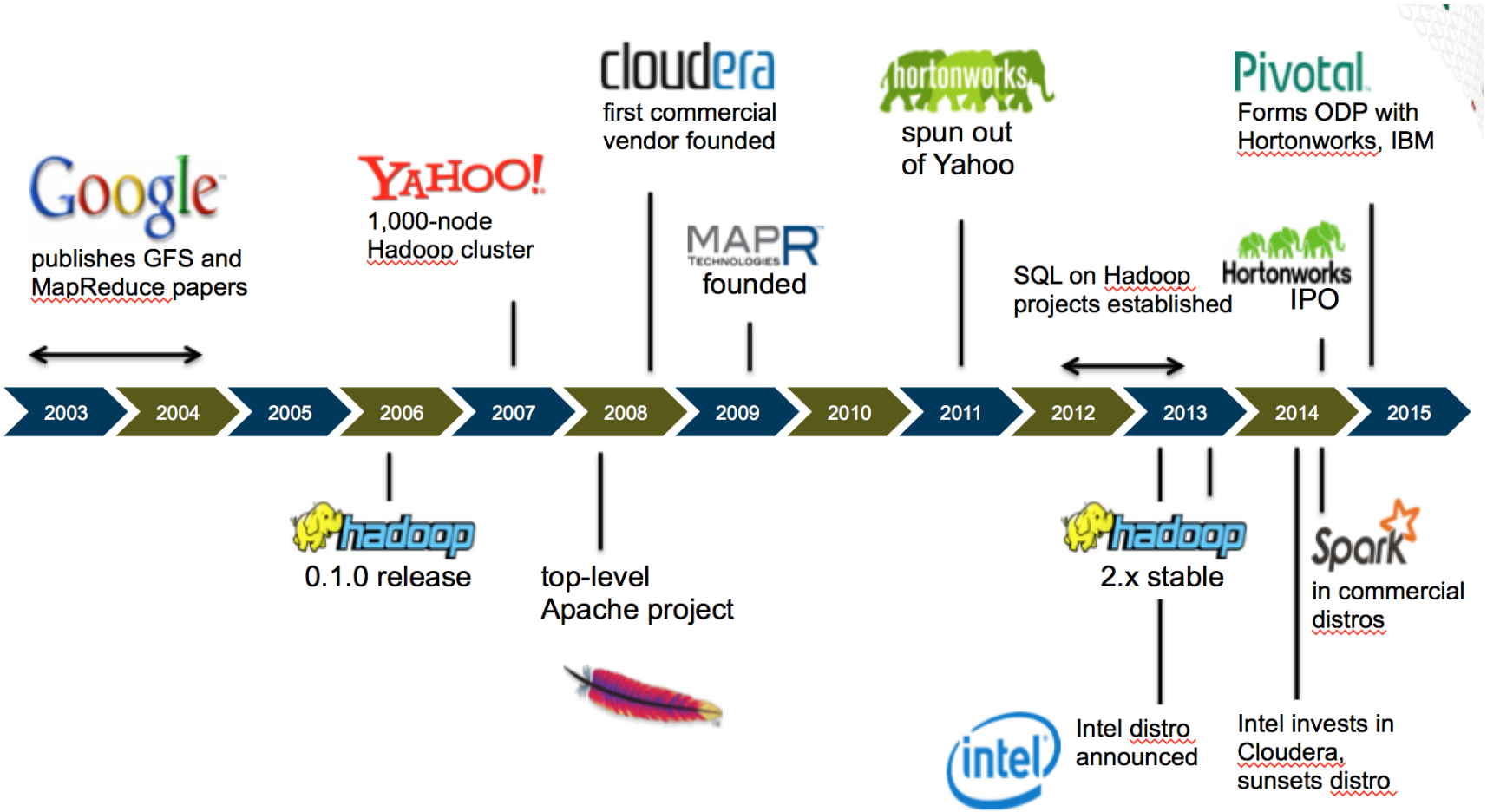
Old model:

"Query the world": Data acquisition coupled to a specific hypothesis

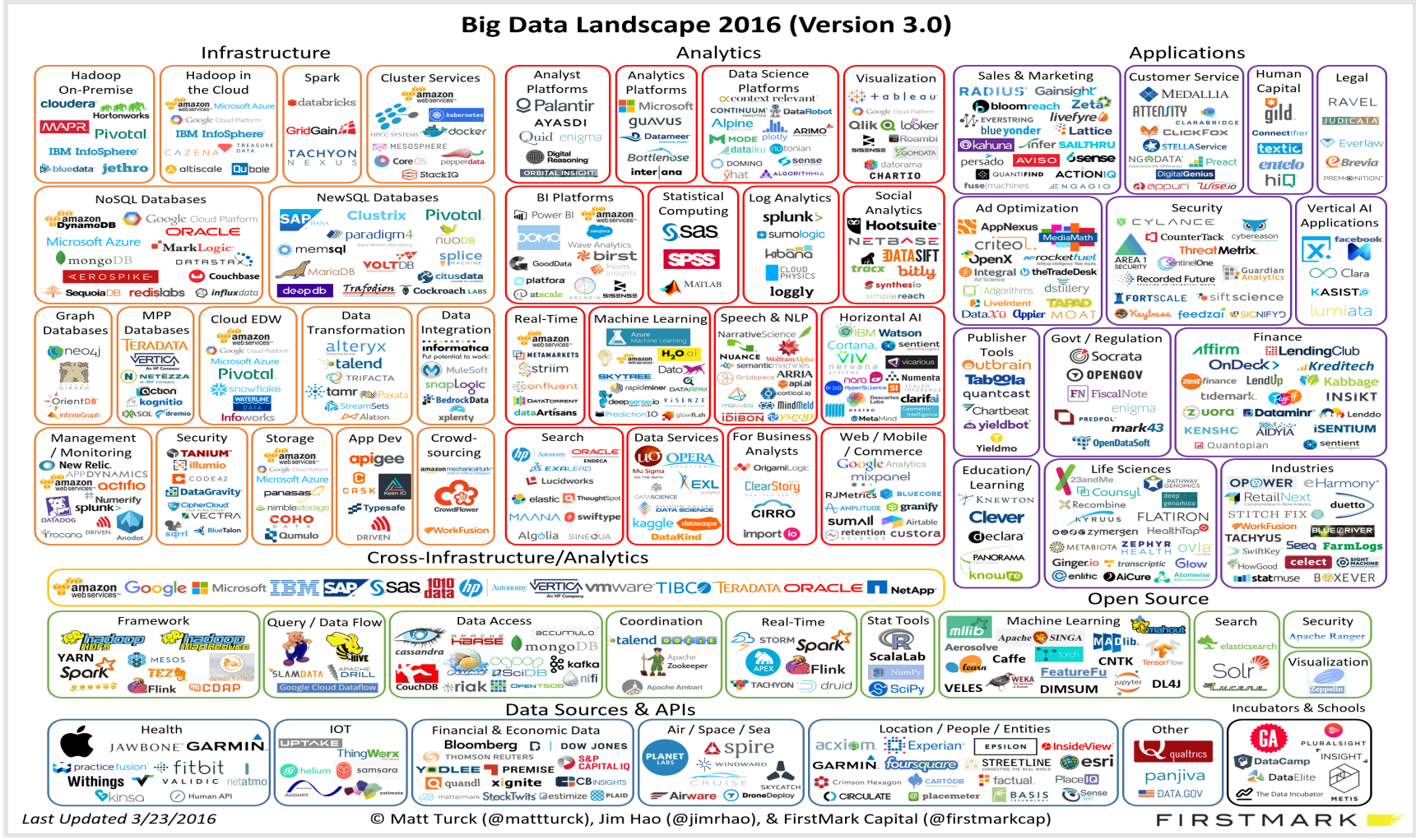
New model:

"Download the world": Data acquisition supports many hypotheses

Big Data tools over the years



Big Data tools over the years



Use Cases

- Web and Social content
  - ✓ Recommendation engines
  - ✓ Ad targeting
  - ✓ Search quality
- Government
  - ✓ Cyber security
  - ✓ Fraud detection
- Banking and Finance
  - ✓ Trade market (monitoring and predictions)
  - ✓ Risk modeling
- Healthcare and Life Sciences
  - ✓ Gene sequencing
  - ✓ Information sharing

Use Cases

- Yahoo!Inc.

  The biggest application of Hadoop on February 19, 2008 - 'The Yahoo! Search Webmap'

  - ✓ a Hadoop app running on more than 10,000 core Linux cluster
  - ✓ generates data that is used in each query of Yahoo! Web search.
  - ✓ more than 100,000 CPUs in $>$40,000 computers running Hadoop
  - ✓ biggest cluster: 4500 nodes ($2*4$cpu boxes w $4*1$TB disk & 16GB RAM)

Use Cases

- Facebook
  - ✓ a 1100 machine cluster with 8800 cores and about 12 PB raw storage.
  - ✓ a 300 machine cluster with 2400 cores and about 3 PB raw storage.
  - ✓ Hadoop helps Facebook in keeping track of all the profiles stored in it, along with the related data such as posts, comments, images, videos, and so on.
- CERN
  - ✓ Hadoop used to store metadata about Physics experiment data
    $\rightarrow$ 100 GB of Logs per day, > 108 TB of Logs in total
  - ✓ Event index Catalogue for experimental Data in the Grid

Use Cases

- NASA
  - ✓ SciSpark → extends Apache Spark for scaling scientific computations
  - ✓ Two scientific use cases:
    - ✓ K-means clustering to compute climate extremes over decades of climate model data
    - ✓ Graph-based automated method for identifying and tracking Mesoscale Convective Complexes (MCCs) that can be categorized as a severe weather event
  - ✓ Scientific data
    - ✓ Stored so as to preserve the logical representation of structured and dimensional data (netCDF, HDF formats)
    - ✓ Analyzed through high-level method that use the MapReduce model

MapReduce

Parallel and distributed programming paradigms

- **Partitioning**
  1. Computation
  2. Data
- **Mapping**

  $\rightarrow$ Assign computation and data parts to resources
- **Synchronisation**
- **Communication**

  $\rightarrow$ Send intermediate data between resources
- **Scheduling**

Parallel and distributed programming paradigms

- **Partitioning**
  1. Computation
  2. Data
- **Mapping**

  $\rightarrow$ Assign computation and data parts to resources
- **Synchronisation**
- **Communication**

  $\rightarrow$ Send intermediate data between resources
- **Scheduling**

A paradigm is an abstraction that hides the implementation of these issues from the users

MapReduce

- An abstraction for performing computations on data
  - ✓ automatic parallelization of computations
  - ✓ large-scale data distribution
  - ✓ simple, yet powerful interface
  - ✓ user-transparent fault tolerance
  - ✓ commodity hardware
- Introduced by Google in 2004: paradigm and implementation

Motivation: Common operations on data

- Iterate over a large number of records
- Extract something of interest from each
- Shuffle and sort intermediate results
- Aggregate intermediate results
- Generate final output

Motivation: Common operations on data

- Iterate over a large number of records
- Extract something of interest from each
- Shuffle and sort intermediate results
- Aggregate intermediate results
- Generate final output

Provide a functional abstraction for these two operations

Motivation: Common operations on data

- Iterate over a large number of records
- Extract something of interest from each
- Shuffle and sort intermediate results
- Aggregate intermediate results
- Generate final output

Map

Reduce

Provide a functional abstraction for these two operations

MapReduce Programming Model

- Inspired by functional programming (Lisp, Scheme, Haskell)
- Programmer specifies two functions: Map and Reduce
  - both work on a list of data elements
- Two phases:
  1. the *Map phase*: parallel execution of the Map function
  2. the *Reduce phase*: parallel execution of the Reduce function
- chained in a pipeline: Map output is Reduce input

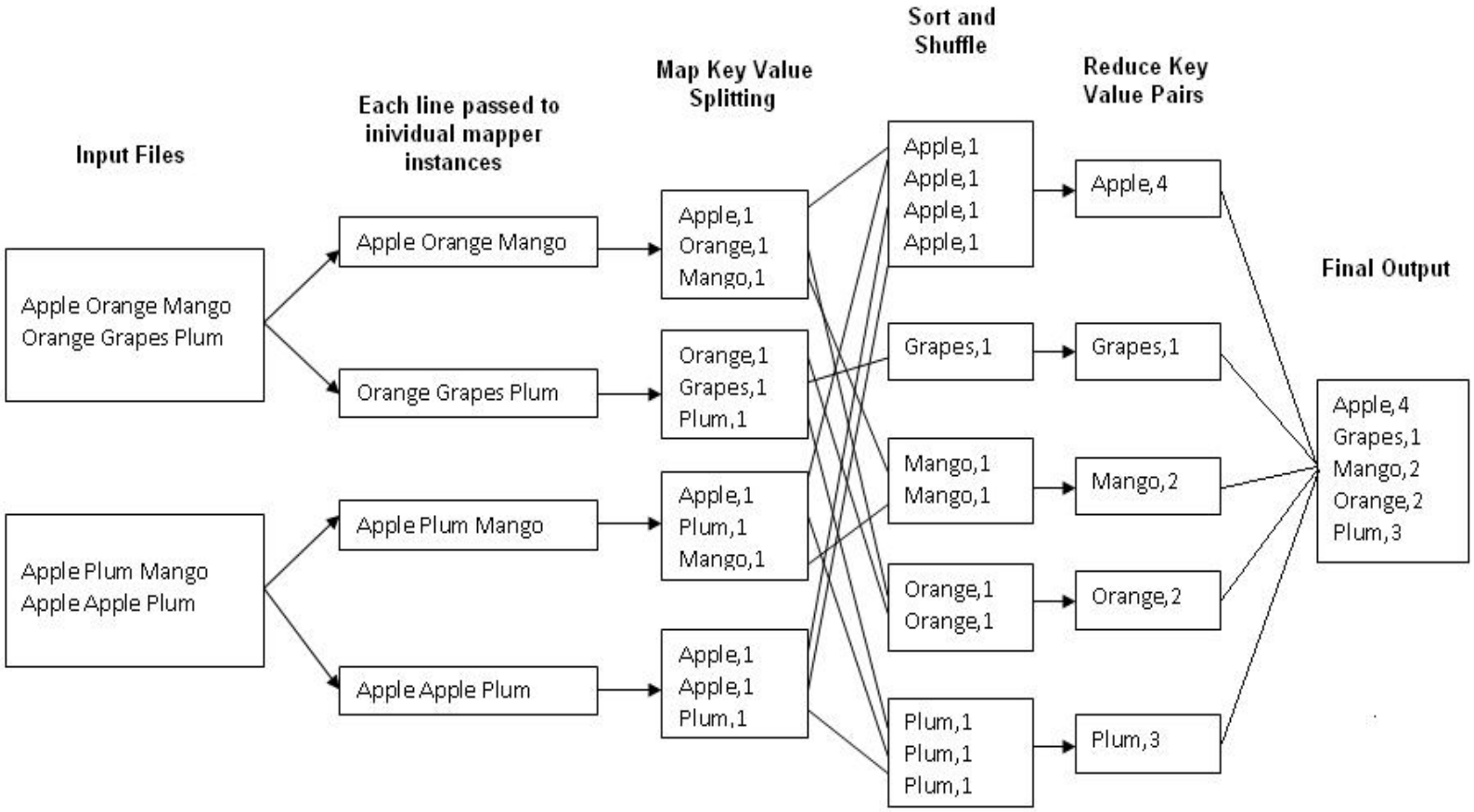
MapReduce model outline

Typical problem solved by MapReduce:

- Read a lot of data
- Map: extract something interesting from each record
- Shuffle and Sort
- Reduce: aggregate, summarize, filter or transform
- Write the results

Outline stays the same, map and reduce change to fit the problem

Example: Word Count

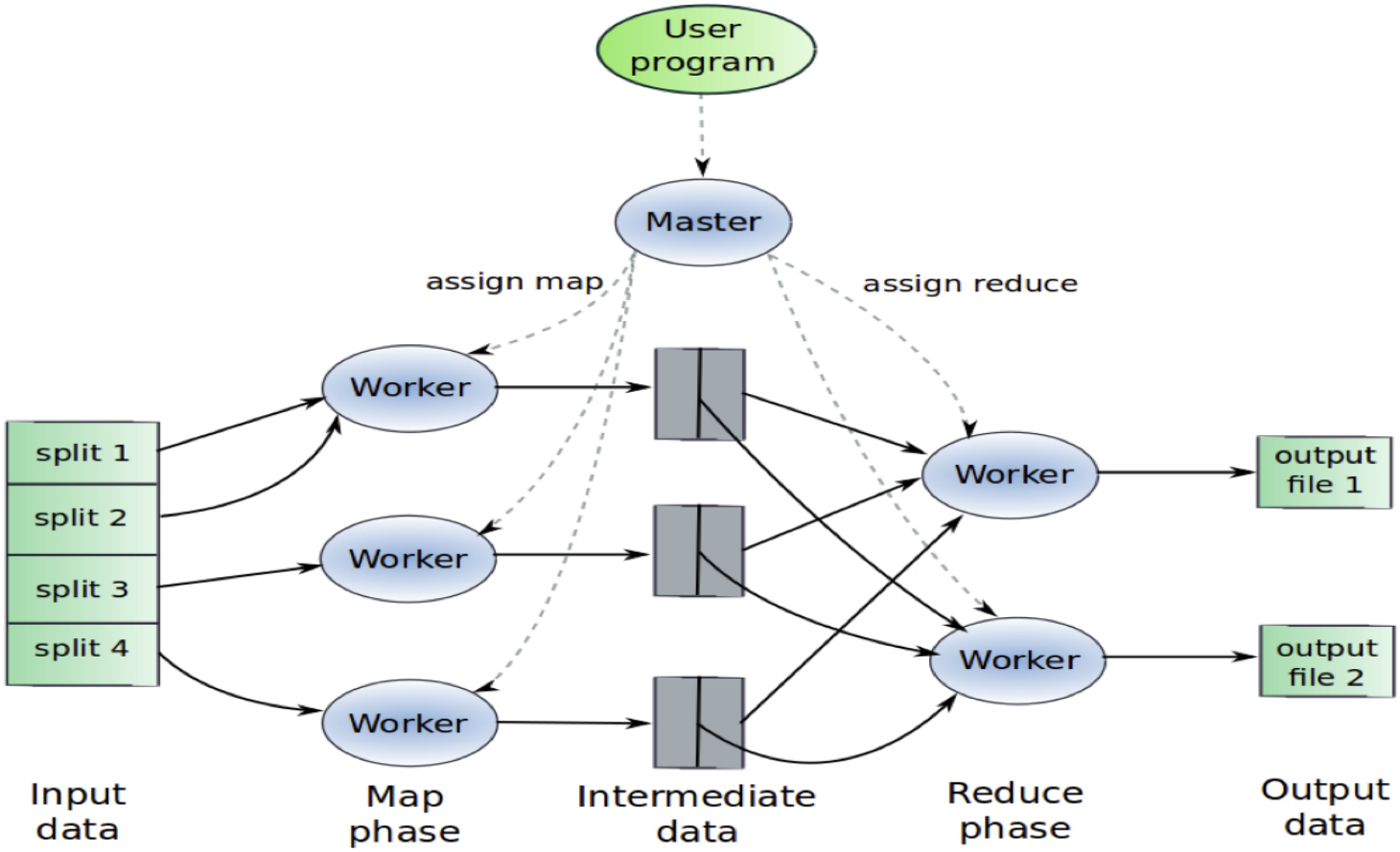


Reduce phase is optional: Jobs can be Map-only

MapReduce Concepts

- Data elements are (*key*, *value*) pairs
  - $map(in\_key, in\_value) -> list(intermediate\_key, intermediate\_value)$
  - $reduce(intermediate\_key, list(intermediate\_value)) -> list(out\_value)$
- Intermediate data
  - data produced by the Map phase and processed by the Reduce phase
- Parallelization
  - of the Map phase is achieved by splitting the input data into smaller blocks; the Map function is applied on each input pair in the block
  - of the Reduce phase is done through partitioning of the key space; the Reduce function is applied to a key and a list of values for each key belonging to a specific key range

How does it work?



Example: Word Count

Counting words in a large set of documents (pseudo code):

```
map (string key, string value) {
    // key: document name / value: document contents
    for each word in value
        EmitIntermediate(word,"1");
}

reduce (string key, iterator values) {
    // key: word  / values: list of counts
    for each v in values
        result += ParseInt(v);
    Emit(AsString(result));
}
```

Key Characteristics

- Parallelism
  - map() and reduce() functions run in parallel
  - each working on different data
  - reduce phase cannot start until map phase is completely finished
- Locality
  - master program assigns tasks based on location of data: tries to have map() tasks on the same machine as physical file data, or at least the same rack

Scheduling - Map

Master assigns each **map** task to a free worker:

- Considers locality of data to worker when assigning task
- Worker reads task input (often from local disk!)
- Worker applies the map function to each record in the split / task
- Worker produces R local files / partitions containing intermediate k/v pairs :
  - Using a partition function
  - E.g., hash(key) mod R

Scheduling - Reduce

- Master assigns each **reduce** task to a free worker
- The i^{th} reduce worker reads the i^{th} partition output by each map using remote procedure calls
- Data is sorted based on the keys so that all occurrences of the same key are close to each other
- Reducer iterates over the sorted data and passes all records from the same key to the user defined reduce function.

What is MapReduce used for?

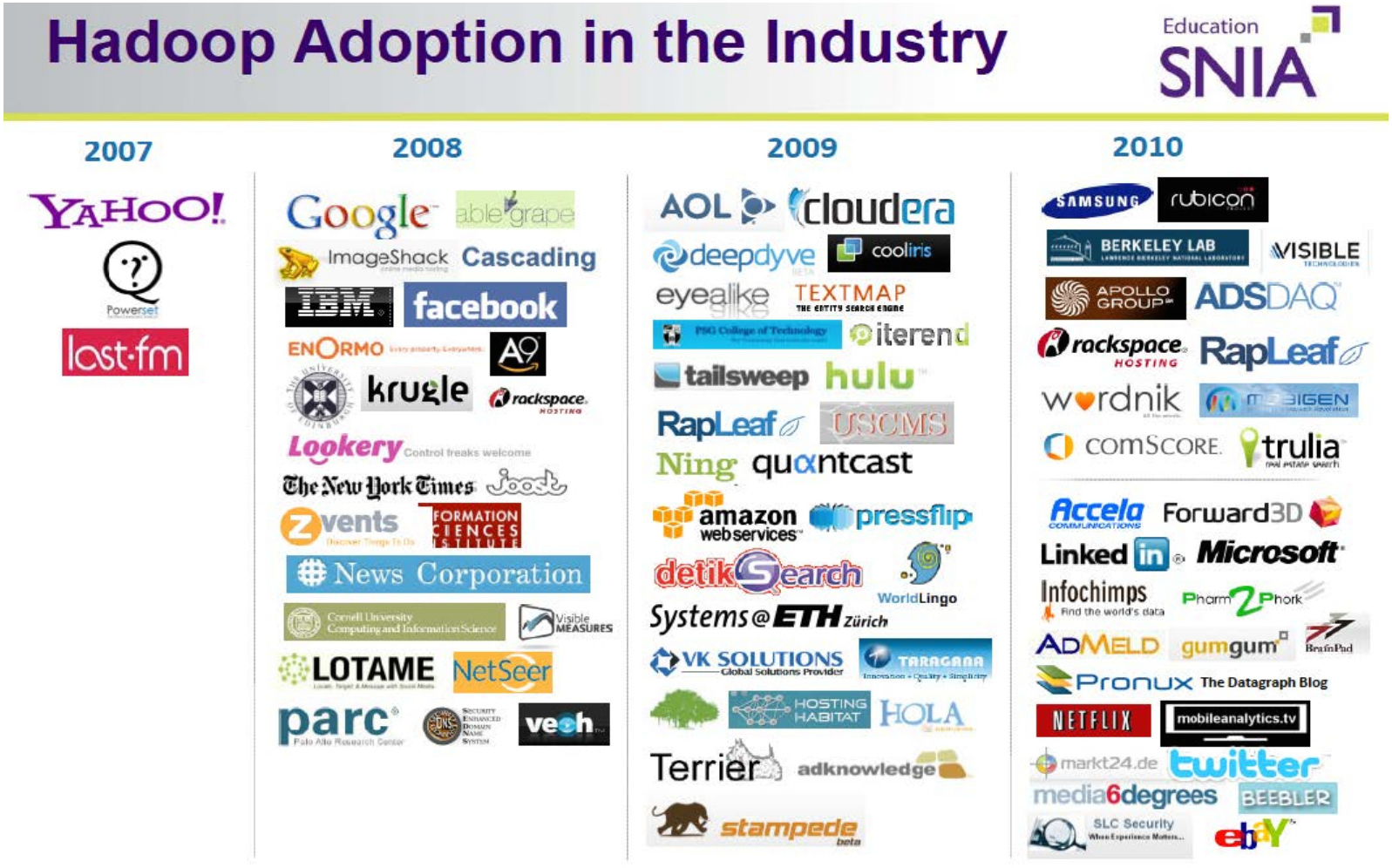
- At Google
  - Index construction for Google Search
  - Article clustering for Google News
  - Statistical machine translation
  - distributed grep, distributed sort
  - web access log stats
  - web link-graph reversal
  - inverted index construction
  - statistical machine translation
- At Yahoo!
  - "Web map" powering Yahoo! Search
  - Spam detection for Yahoo! Mail
- At Facebook
  - Data mining
  - Ad optimization
  - Spam detection


hadoop

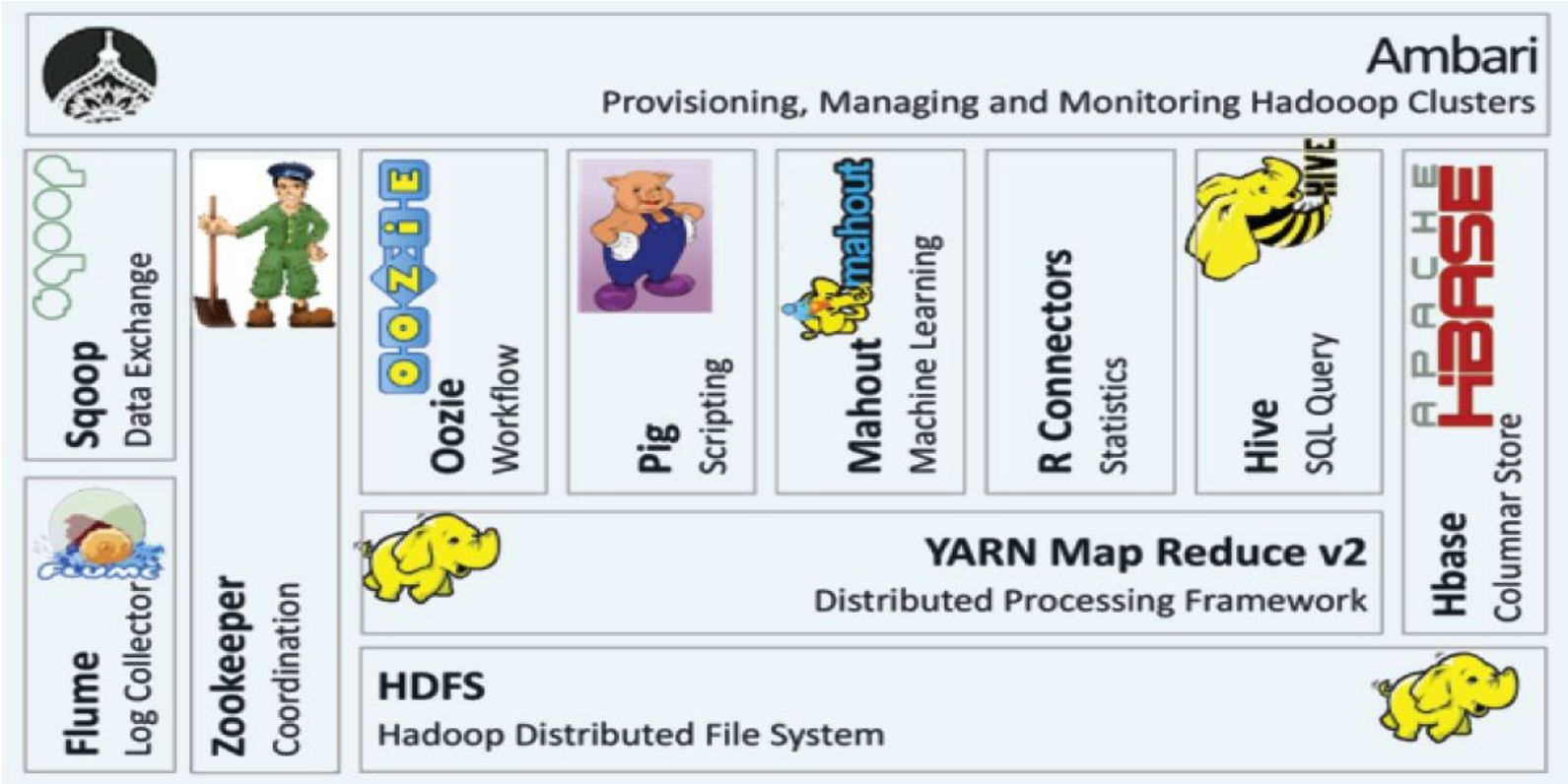
The Hadoop Project

- Open-source project started in 2006
- Developed in Java
- Two core components: HDFS and MapReduce implementation
- Founded by Apache
- Platform for data storage and processing
  - ✓ Scalable
  - ✓ Fault tolerant
  - ✓ Distributed
  - ✓ Any type of complex data

Who uses Hadoop?



Apache Hadoop Ecosystem



HDFS

Distributed File System

- Data split into large **chunks** of fixed (configurable) size – 64MB default
- Optimised for managing large files:
  - Write-once-read-many access pattern
  - Data I/O involving blocks of data, rather than random access
  - High throughput rather than low-latency
- Fault-tolerant
  - chunk-level replication
  - 3 replicas: local, same rack, different rack
- File System interface
  - Not POSIX compliant
  - Most FS commands
  - ls, mkdir, cat, rm, etc
  - put, get, distcp, etc

HDFS

Distributed File System

- Data split into large **chunks** of fixed (configurable) size – 64MB default
- Optimised for managing large files:
  - Write-once-read-many access pattern
  - Data I/O involving blocks of data, rather than random access
  - High throughput rather than low-latency
- Fault-tolerant
  - chunk-level replication
  - 3 replicas: local, same rack, different rack
- File System interface
  - Not POSIX compliant
  - Most FS commands
  - ls, mkdir, cat, rm, etc
  - put, get, distcp, etc

HDFS

Distributed File System

- Data split into large **chunks** of fixed (configurable) size – 64MB default
- Optimised for managing large files:
  - Write-once-read-many access pattern
  - Data I/O involving blocks of data, rather than random access
  - High throughput rather than low-latency
- Fault-tolerant
  - chunk-level replication
  - 3 replicas: local, same rack, different rack
- File System interface
  - Not POSIX compliant
  - Most FS commands
  - ls, mkdir, cat, rm, etc
  - put, get, distcp, etc

HDFS

Distributed File System

- Data split into large **chunks** of fixed (configurable) size – 64MB default
- Optimised for managing large files:
  - Write-once-read-many access pattern
  - Data I/O involving blocks of data, rather than random access
  - High throughput rather than low-latency
- Fault-tolerant
  - chunk-level replication
  - 3 replicas: local, same rack, different rack
- File System interface
  - Not POSIX compliant
  - Most FS commands
  - ls, mkdir, cat, rm, etc
  - put, get, distcp, etc

HDFS

Distributed File System

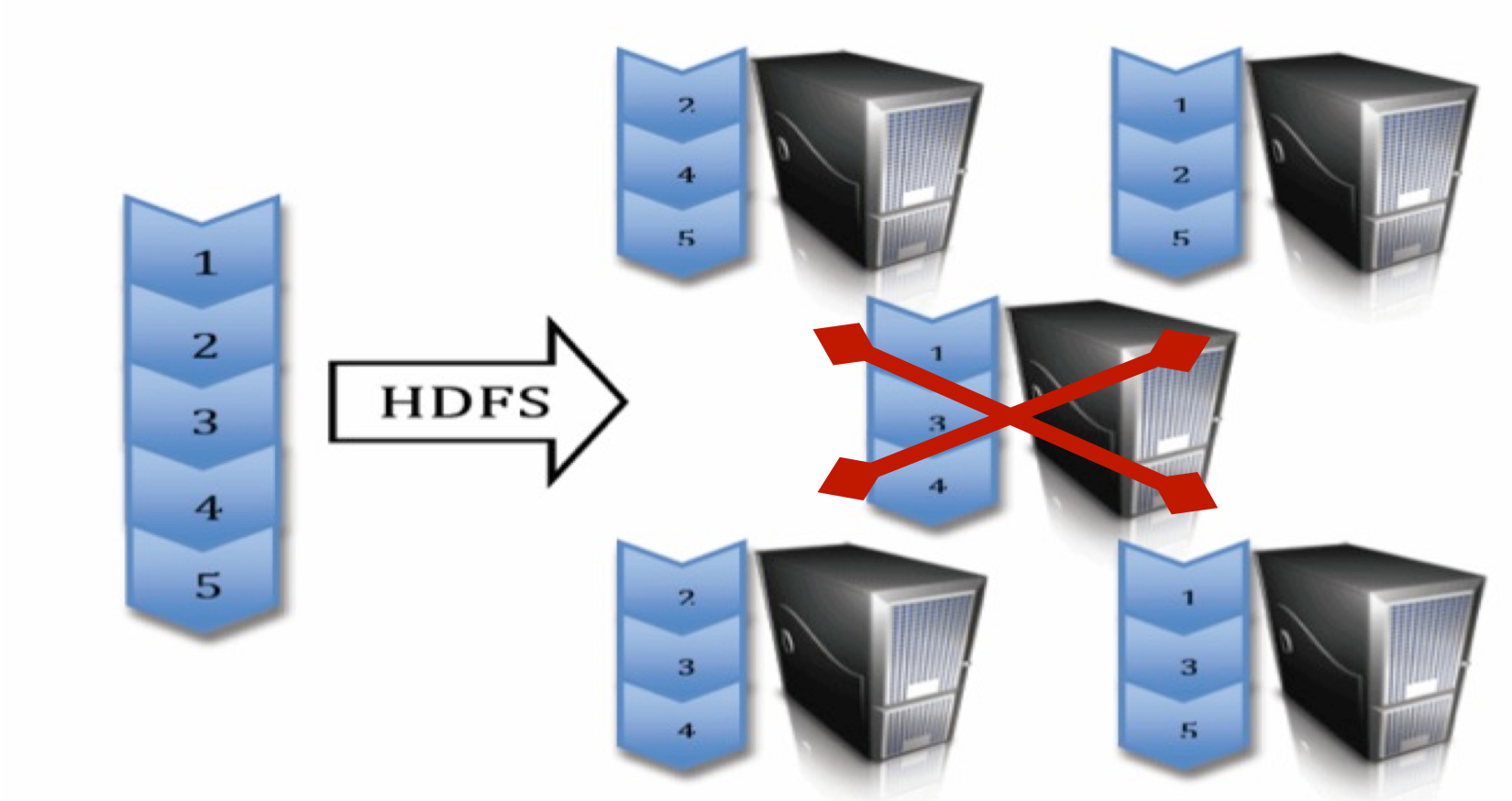
- Data split into large **chunks** of fixed (configurable) size – 64MB default
- Optimised for managing large files:
  - Write-once-read-many access pattern
  - Data I/O involving blocks of data, rather than random access
  - High throughput rather than low-latency
- Fault-tolerant
  - chunk-level replication
  - 3 replicas: local, same rack, different rack
- File System interface
  - Not POSIX compliant
  - Most FS commands
  - ls, mkdir, cat, rm, etc
  - put, get, distcp, etc

HDFS

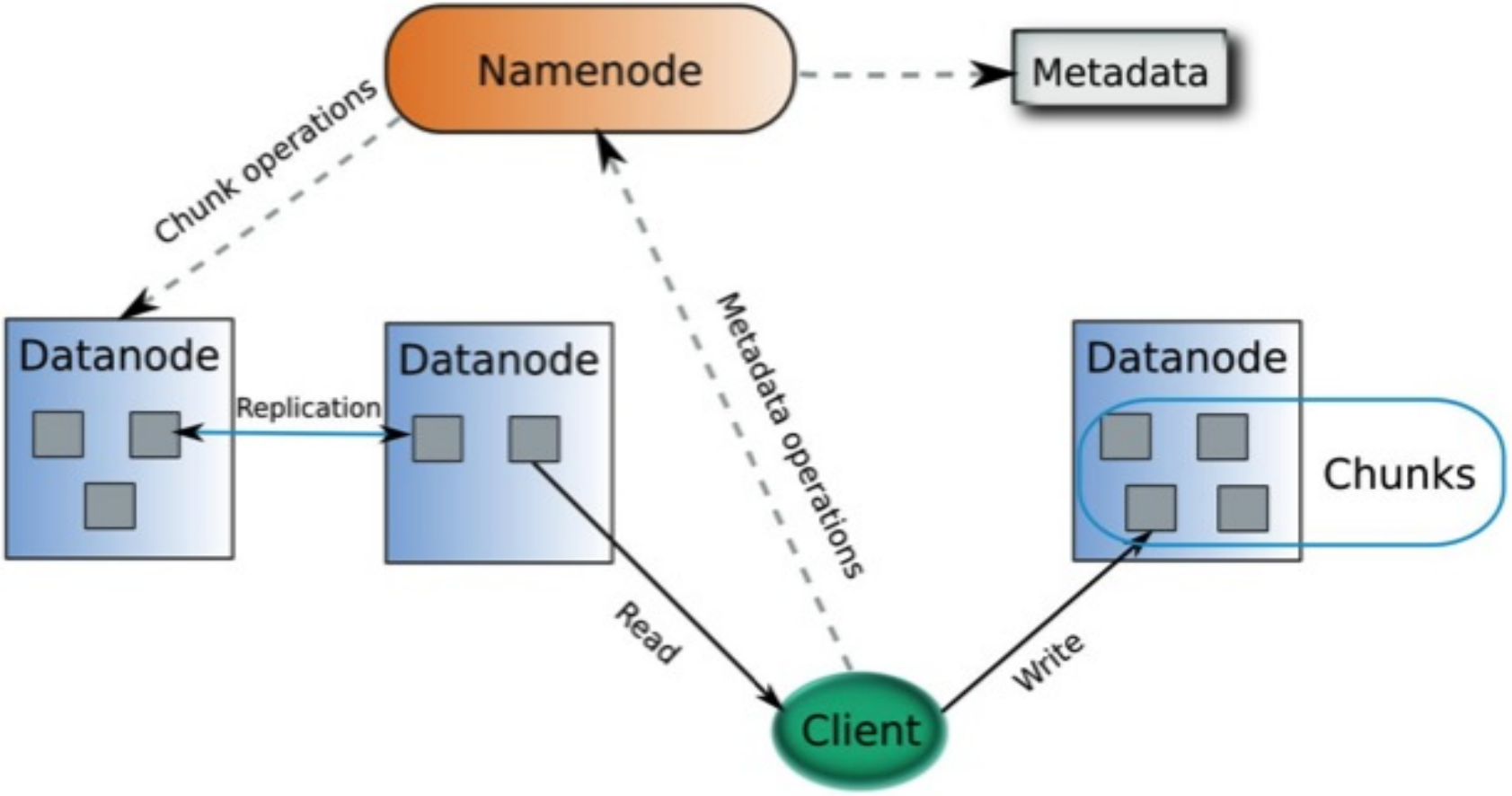
Distributed File System

- Data split into large **chunks** of fixed (configurable) size – 64MB default
- Optimised for managing large files:
  - Write-once-read-many access pattern
  - Data I/O involving blocks of data, rather than random access
  - High throughput rather than low-latency
- Fault-tolerant
  - chunk-level replication
  - 3 replicas: local, same rack, different rack
- File System interface
  - Not POSIX compliant
  - Most FS commands
  - ls, mkdir, cat, rm, etc
  - put, get, distcp, etc

HDFS-Hadoop Distributed File System



HDFS Design: Master - Slave



HDFS Architecture

HDFS Master: **Namenode**

- manages the filesystem namespace
  - list of files
  - for each file name: a set of blocks
  - for each block: a set of DataNodes
  - file attributes (creation time, replication factor)
- manages block replication
- single point of failure
  - if the Namenode fails, the filesystem is not usable anymore
  - the metadata can be stored on a remote disk so that the namespace can be reconstructed if the Namenode fails

HDFS Architecture

HDFS Slaves: **DataNodes**

- A DataNode is a block server
  - Stores data in the local file system (e.g. ext3)
  - Stores meta-data of a block (e.g. CRC)
  - Serves data and meta-data to Clients
- Block report
  - Periodically sends a report of all existing chunks to the NameNode
- Perform replication tasks upon instruction by NameNode

HDFS command line interface

- The user can access HDFS through various shell commands
  - `hadoop fs –put <localsrc> … <dst>`
  - `hadoop fs –get <src> <localdst>`
  - `hadoop fs –ls`
  - `hadoop fs –rm file`
  - `hadoop fs –help`
- Permissions model for files and directories similar to POSIX
  - read (r), write(w), execute(x)

Hadoop MapReduce

Concepts:

- MapReduce job: the implementation of a MapReduce application (java file, jar file, python, shell script, etc)
- Map Tasks / Reduce Tasks: computations that are executed in parallel
  - Each map task processes a single chunk of the input data stored in HDFS
    - number of map tasks equals the number of input splits
  - Reduce tasks process Map tasks output, by splitting the key range
    - the number of Reduce tasks is usually a lot smaller
    - each Reduce task produces its separate output file in HDFS
- Records are data elements: (*key*, *value*) pairs

Hadoop MapReduce

The MapReduce Framework:

- Sorts the map output and also the reduce output (the final output data)
- Handles transparently tasks scheduling, monitoring and re-execution in case of failure
- Starts map tasks on the same machines that store the assigned data chunks (data locality)

Hadoop 2 - YARN Architecture

ResourceManager (RM)

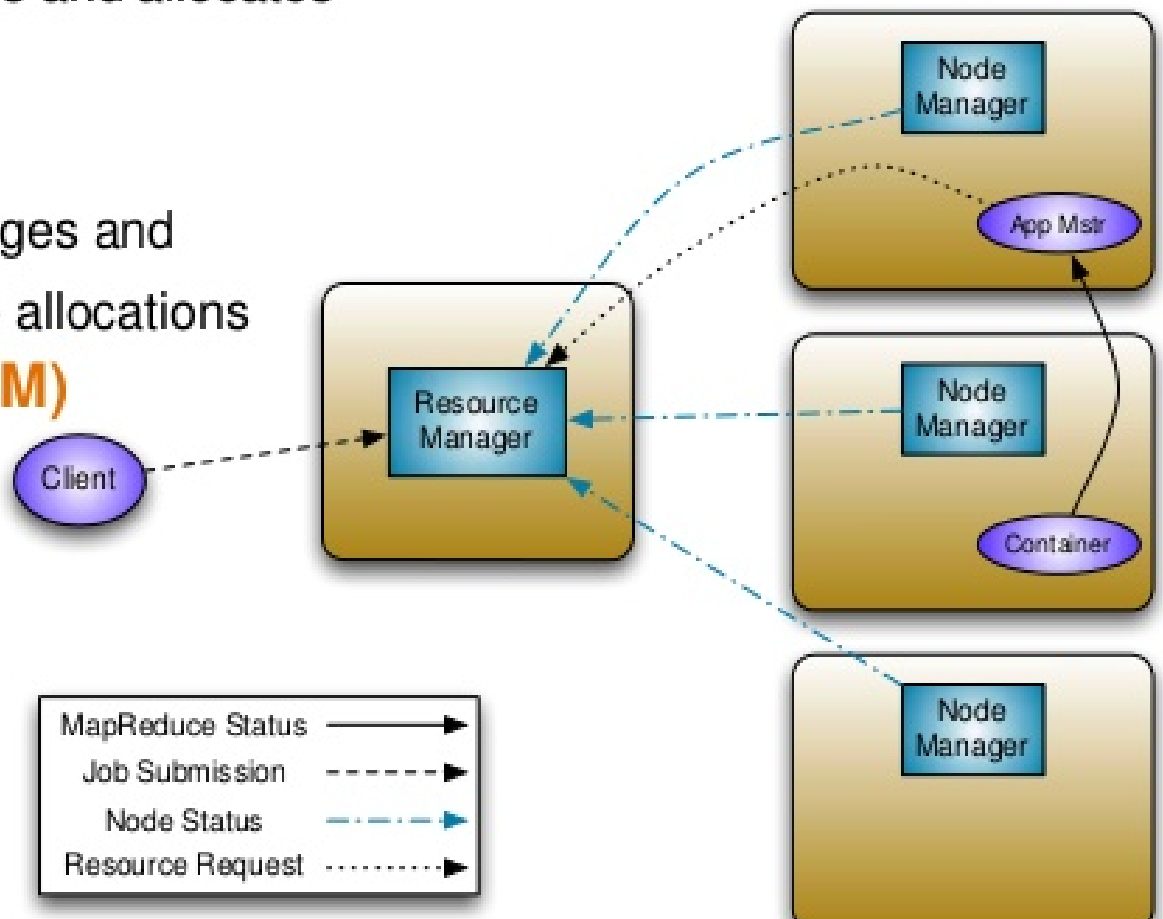
Central agent - Manages and allocates cluster resources

NodeManager (NM)

Per-Node agent - Manages and enforces node resource allocations

ApplicationMaster (AM)

Per-Application – Manages application lifecycle and task scheduling



Hadoop Terminology

HDFS	NameNode	Master process • FS metadata • Chunk layout
	DataNodes	Worker process • I/O access to chunks
	Chunk / Block	Block of data (64MB)
YARN	ResourceManager	Master process • Resource Scheduler • Responsible for communication with the client
	NodeManager	Worker process • Manages resources on a node • Monitors applications that run on the node
	ApplicationMaster	Per Application master process • Requests resources from the ResourceManager
	Container	Logical representation of a resource (cpu, memory) / Process spawned
MapReduce	Job	A running MapReduce application with one map phase followed by a reduce phase
	Task	Map or Reduce computation executed in parallel
	(Input) Split	Input data for a single Map task • Corresponds to a single HDFS chunk
	Shuffle & Sort	Phase in between Map and Reduce in which intermediate data is copied from the mappers to the reducers