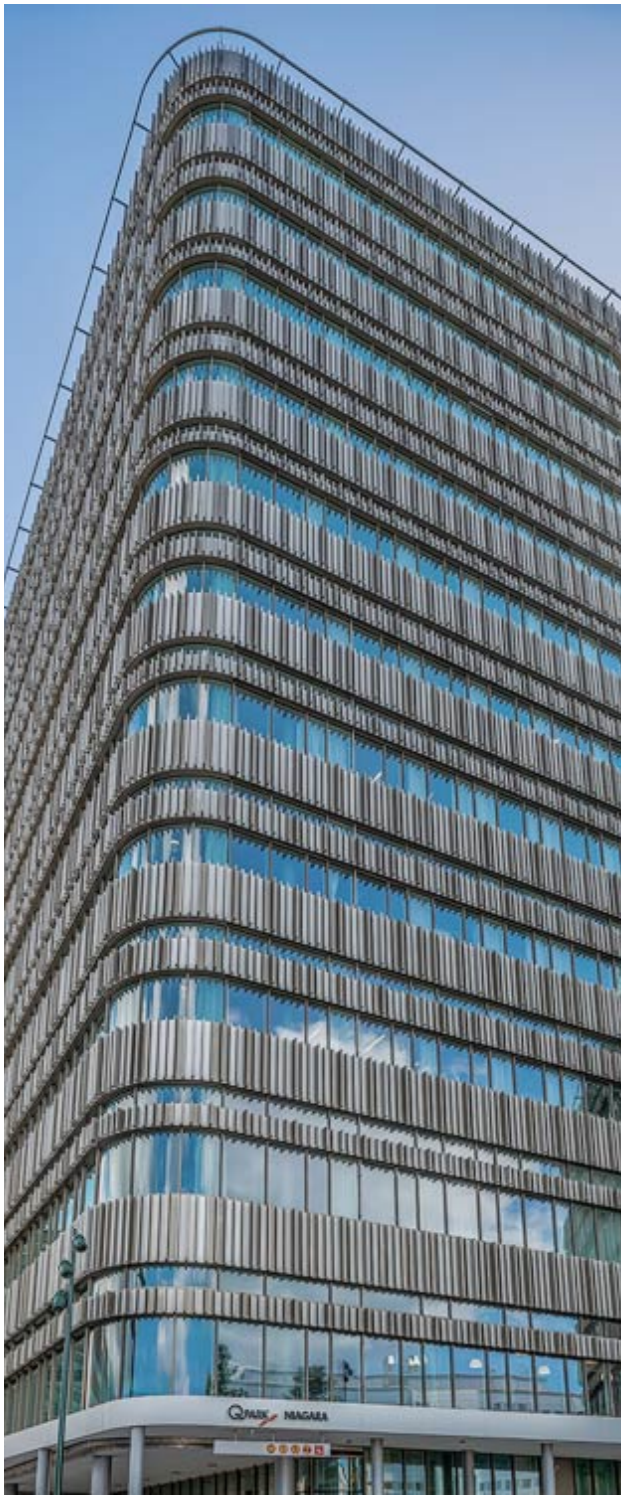

Cloud
Computing

Agenda

1. Brief History to Cloud
2. Characteristics of Cloud
3. Type of Clouds
4. Economics of Clouds
5. Virtualization
6. New Cloud Programming Paradigms
7. Batch Processing
8. Stream Processing



Who am I ?

Radu Mihailescu

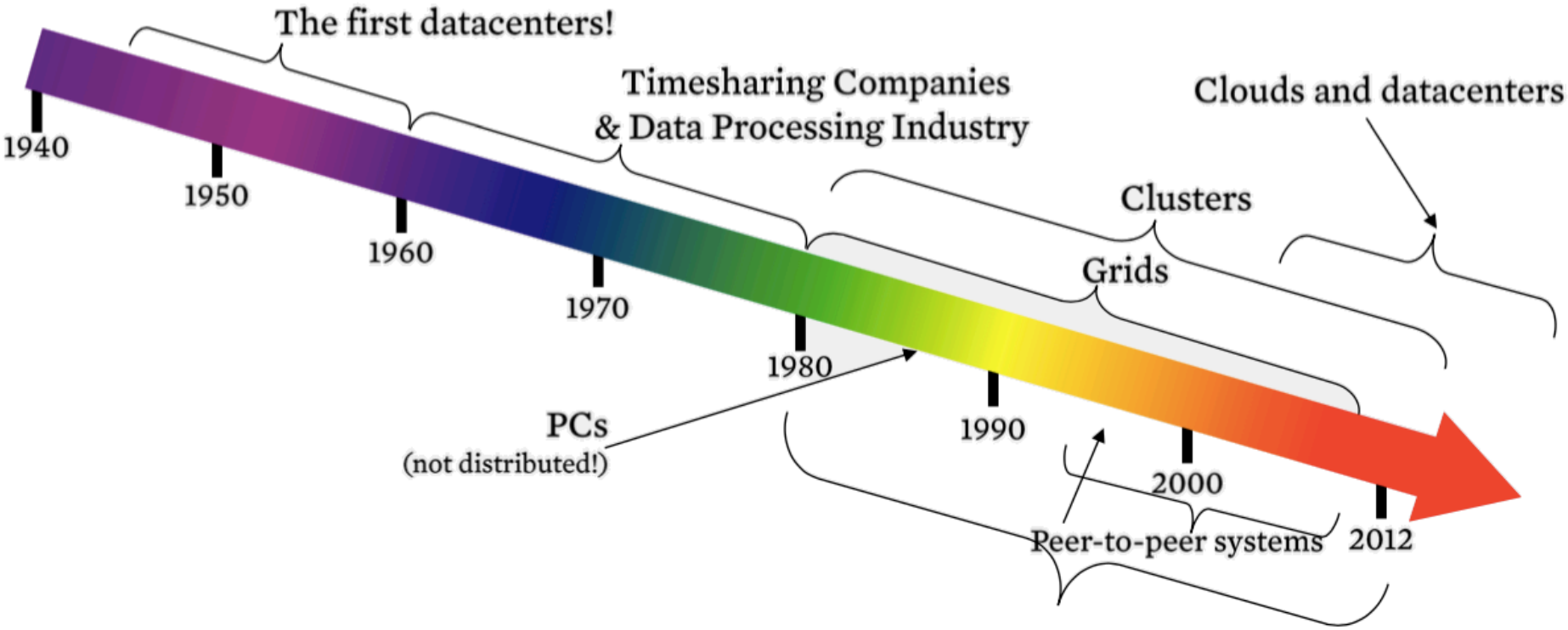
Senior Lecturer

PhD in Computer Science/AI/MAS

Internet of Things and People Research Center (IOTAP)

Email:
radu.c.mihailescu@mah.se

Historical progression to Clouds



Vision: Computing will be available as a utility, just like electricity is available as a utility – MIT's F. Corbato (1965)

Characteristics of Cloud Computing

Massive Scale

Need for datacenter management.

On-demand access

Pay and scale as you go pricing schemes.

Data Intensive

TBs, PBs and XBs of data.

New Programming Paradigms

Solve distributed computing problems in the cloud.

How massive is massive-scale?

#servers

900K

[2011]

1.3M

[2016]

180K

[2012]

1M

[2013]

380K

[2012]

Datacenter quick Virtual Tour

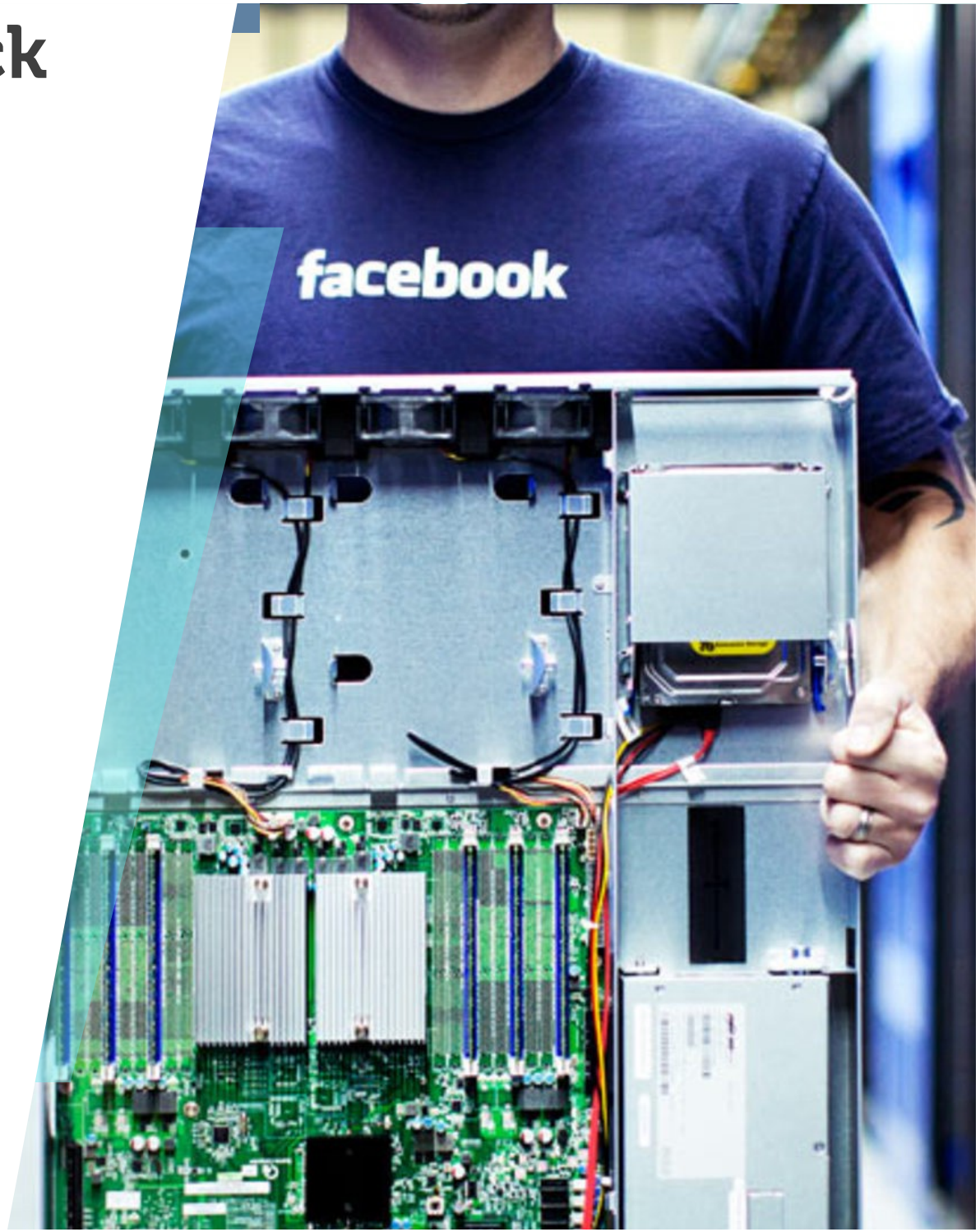
Front-side

Datacenter quick Virtual Tour

Back-side

Datacenter quick Virtual Tour

In-side



On-demand access (*aaS Classification)

1

HaaS

Hardware as a Service:
access to barebones hardware machines; may be remote or on site, depending on the hardware setup; involves a contract for the maintenance and administration of hardware systems.

2

IaaS

Infrastructure as a Service:
provides virtualized computing resources over the internet i.e. access to flexible computing and storage infrastructure.

3

PaaS

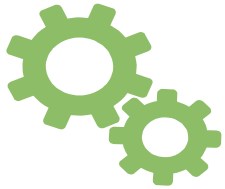
Platform as a Service:
provider delivers hardware and software tools ready for application development to users over the internet; provider hosts the hardware and software on its own infrastructure.

4

SaaS

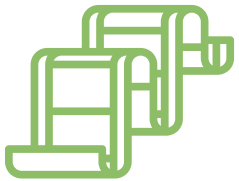
Software as a Service:
provider hosts applications and makes them available to customers over the internet; removes the need for users to install and run the applications themselves.

Data-intensive Computing



Computing-intensive processing

- High performance computing
- Running on supercomputers
- Small amounts of data
- Focus on CPU utilization
- E.g. weather modeling



Data-intensive processing

- Large amounts of data stored nearby in datacenters
- Brings compute cycles closer to the data
- Disk and network I/O is maximized

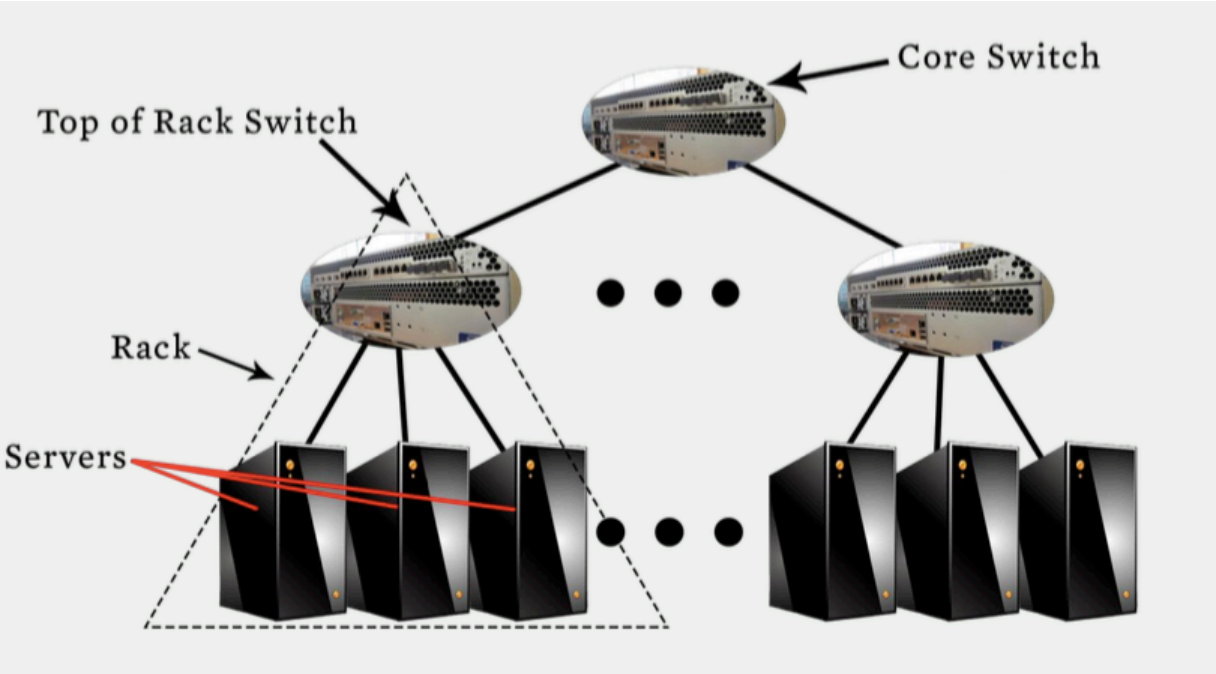
Technology Trends (capacity/$):

- CPU compute capacity X 2 about every 18 months (Moore's law)
- Storage capacity X 2 every 12 months
- Bandwidth capacity X 2 every 9 months

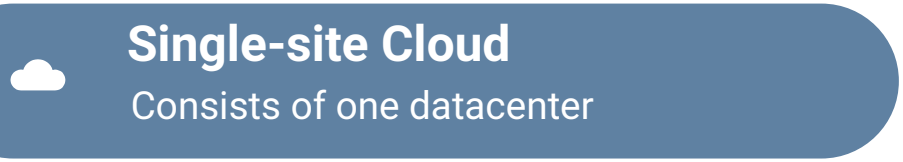
So, what is a Cloud? & Cloud Topology Example

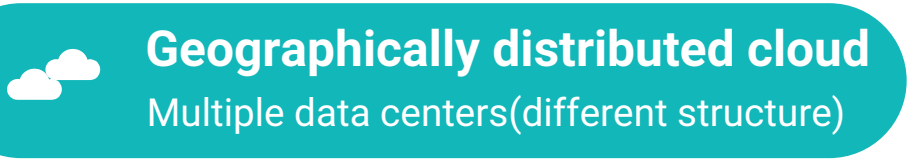
IT infrastructure with lots of storage and computing cycles nearby and/or applications on rent over the Internet

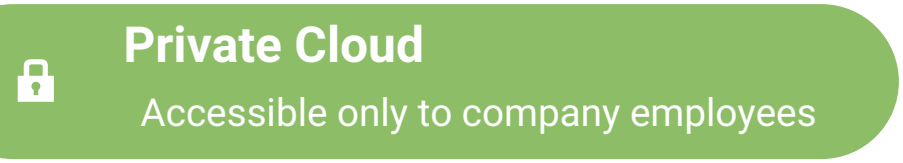
- Compute nodes
- Switches
- Network topology
- Storage nodes
- Front-end
- Software services



Different type of Clouds







Economics of Clouds

Build your own infrastructure vs. use a public cloud?

Own

- Build hardware: storage, cpu, networking, upgrades, etc.
- Over/Under estimation of hardware needs
- Complex software stacks: installation, maintenance, etc.
- High capital investment

Rent

- Solves resource estimation
- Solve resource management
- Instant access to different resources: cpu, gpu, etc.
- Pay-as-you-go (scalability)
- Quick prototyping
- Deploy closer to your client

Economics of Clouds

Build your own infrastructure vs. use a public cloud?

Example: suppose a company with the following requirements:

- Computing power of 128 servers (1024 cores)
- Storage 524 TB
- Time horizon of *N* months

AWS monthly costs:

- S3 costs $0.023 per GB per month
- EC2 costs $0.026 per CPU per hour
- Total = $0.023 X 524 X 1000 + $0.026 X 1024 X 24 X 30
- Total = $12052 + $19169
- Total =$31 221

Own infrastructure cost monthly:

- 3 yrs lifetime of hardware
- Total = ($4000 X 128 + $5000) / N + 1sysadmin
- Total aprox. $512 000 /N + $10000

-> Breakeven: N >24 (preferable to own infrastructure?)

Economics of Clouds

Power consumption

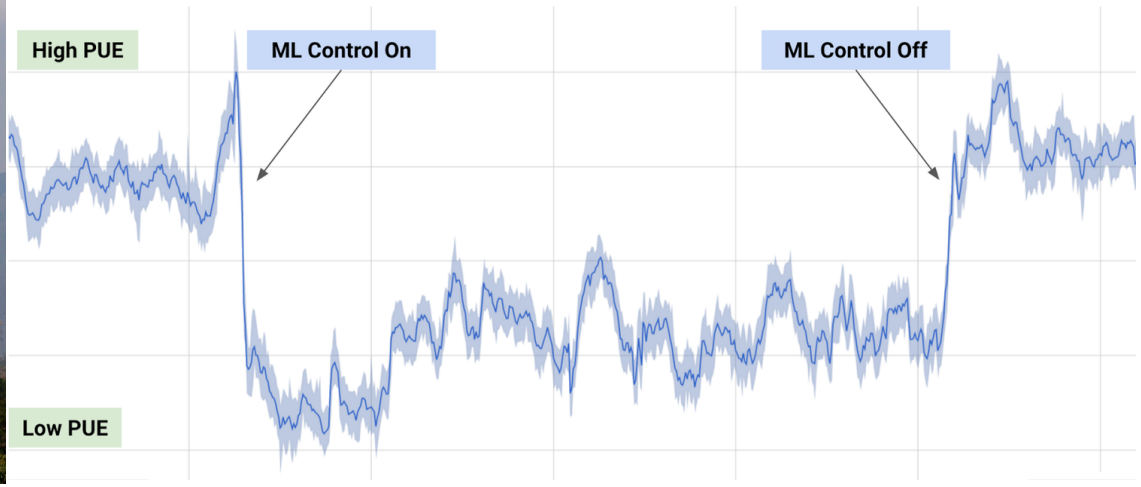
Metrics:

WUE = Annual Water Usage / IT Equipment Energy (L/kWh)

PUE = Total Facility Power / IT Equipment Power

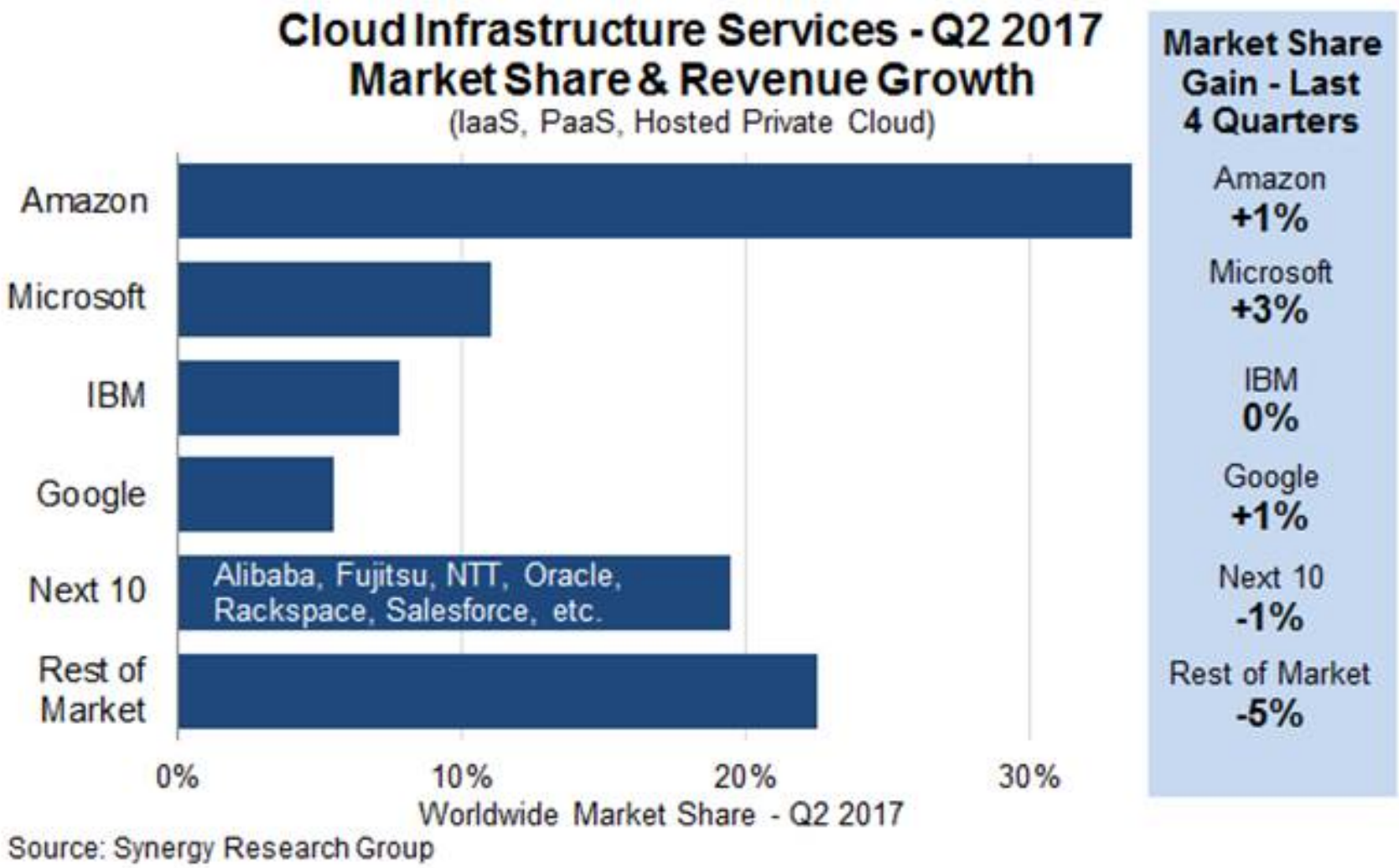
- 40% Energy for cooling
- 15% PUE

Google DeepMind



Some Cloud Computing Providers

Market share 2017



Virtualization

Clouds are based on Virtualization

Problem: How do we share a physical computer among multiple users?

Solution: Introduce an abstract model of what a generic computing resource should look like, then provide this abstract model to many users

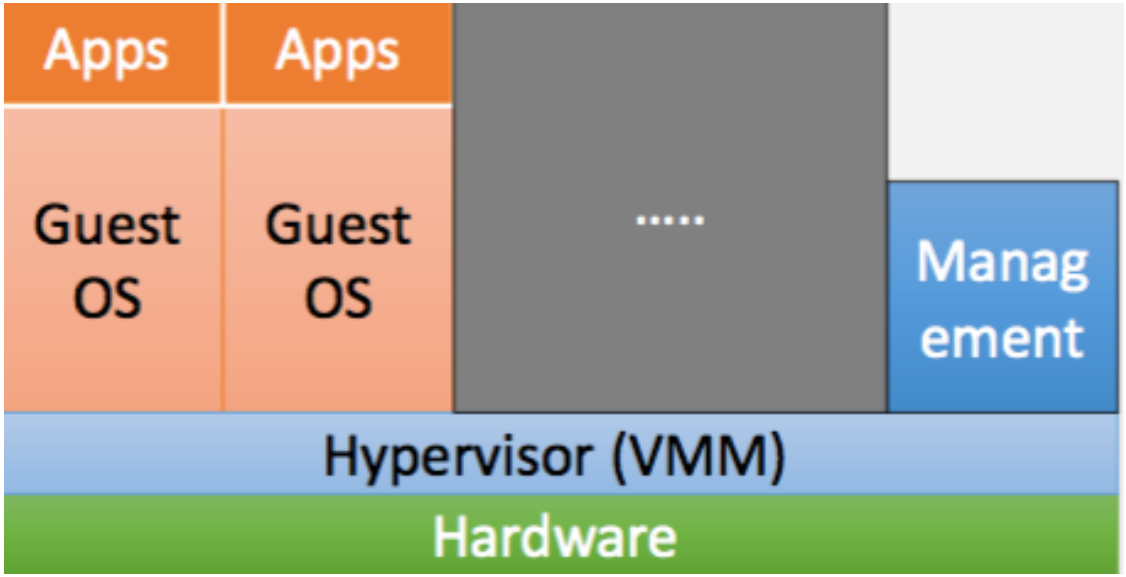
Each instance “thinks” it has a simple model of a computer underlying its computation

Types of Virtualization:

- Native/full virtualization
- Hardware assisted virtualization
- Para-virtualization
- OS level virtualization

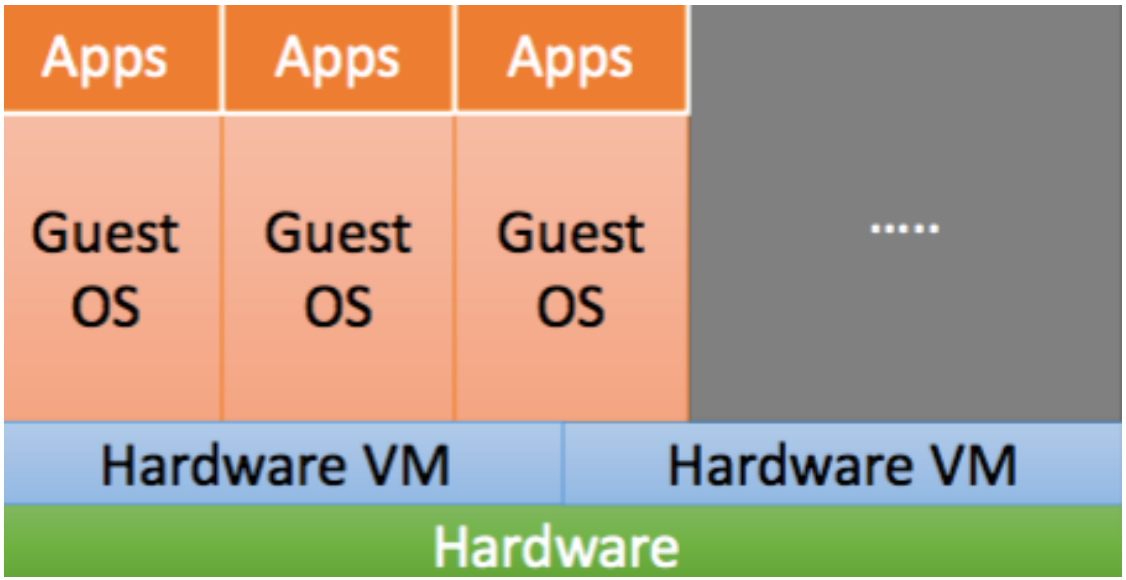
Native/Full Virtualization

- the virtual machine simulates enough hardware to allow an unmodified "guest" OS (one designed for the same CPU) to be run in isolation.
- e.g. VirtualBox, VMware, etc.



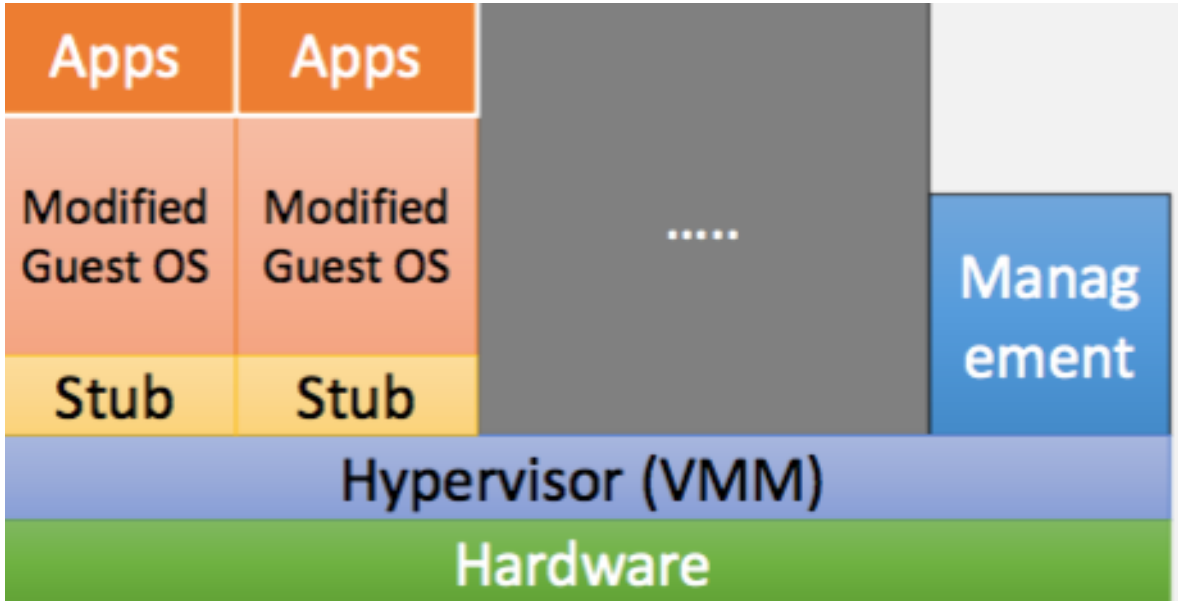
Hardware enabled virtualization

- the virtual machine has its own hardware and allows a guest OS to be run in isolation (e.g. Intel VT (IVT), AMD virtualization (AMD-V))
- e.g. VMware Fusion



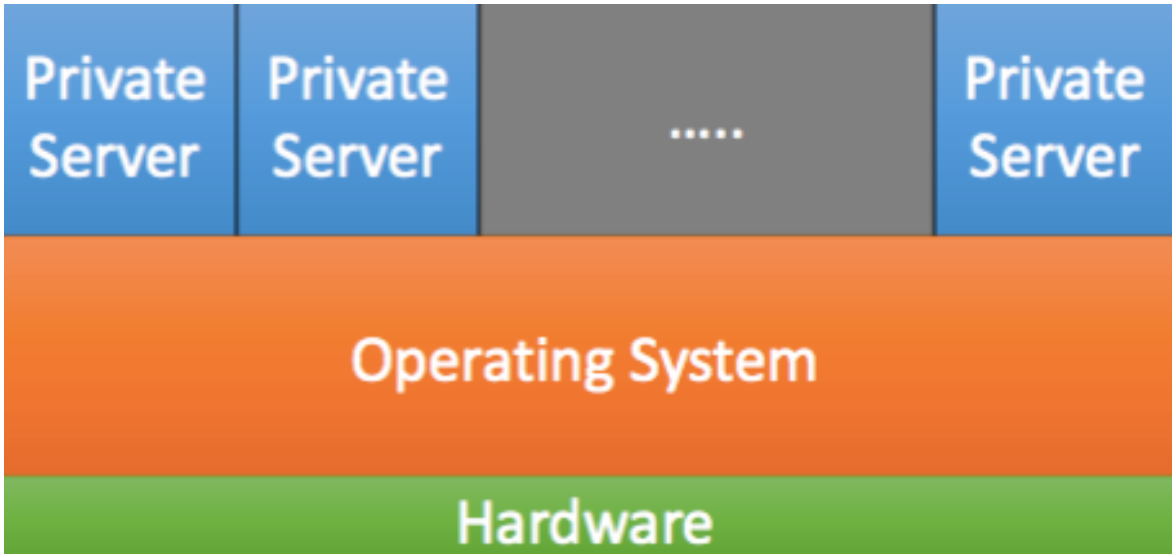
Para-virtualization

- the virtual machine does not necessarily simulate hardware, but instead (or in addition) offers a special API that can only be used by **modifying** the "guest" OS.
- e.g. XEN



OS-level Virtualization

- virtualizing a physical server at the operating system level, enabling multiple isolated and secure virtualized servers to run on a single physical server.
- e.g. Linux-Vserver, Solaris Containers, FreeBSD Jails



New Cloud Programming Paradigms

Advent of new cloud programming paradigms for:

- Computing data
- Storing/Querying the data

[2004] Google publishes a paper on their processing framework MapReduce
[2005] Yahoo releases an open source implementation of it called Hadoop

Google: introduced MapReduce
- back in 2006 google search engine indexing a chain of 24 MapReduce jobs
- aprox. ~ 200.000 jobs processing 50PB every month

Yahoo: Hadoop + Pig
- a chain of 100 MapReduce jobs
- 280 TB of data, 2500 nodes, 73 hours

Facebook: Hadoop + Hive
- ~300TB total, adding 2TB/day (in 2008)
- 3.000 jobs processing 55TB/day

Amazon: Elastic MapReduce service

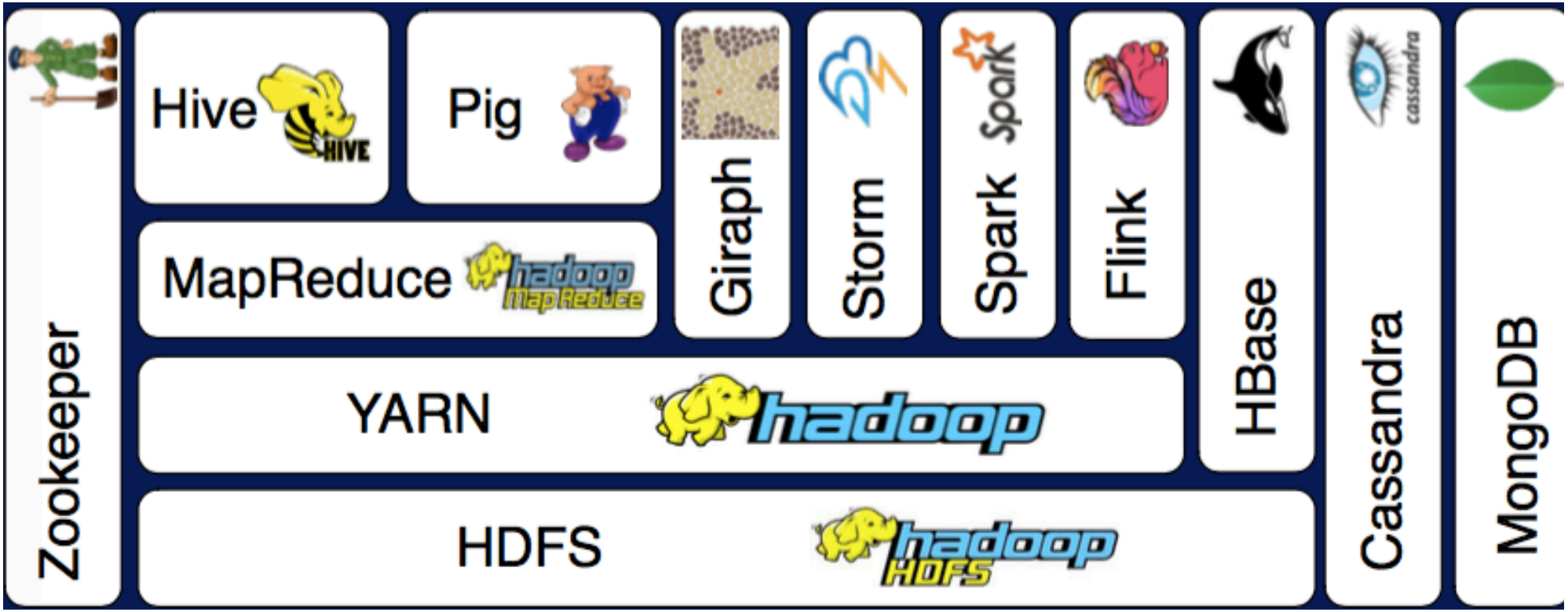
NoSQL: MySQL is an industry standard, but Cassandra is 2400 times faster!

Hadoop Ecosystem

Goals:

1. Enable Scalability on commodity hardware
2. Handle Fault tolerance and graceful recovery
3. Optimized for integrating a variety of data types
4. Facilitate a shared environment
5. Open source projects backed by a large active community

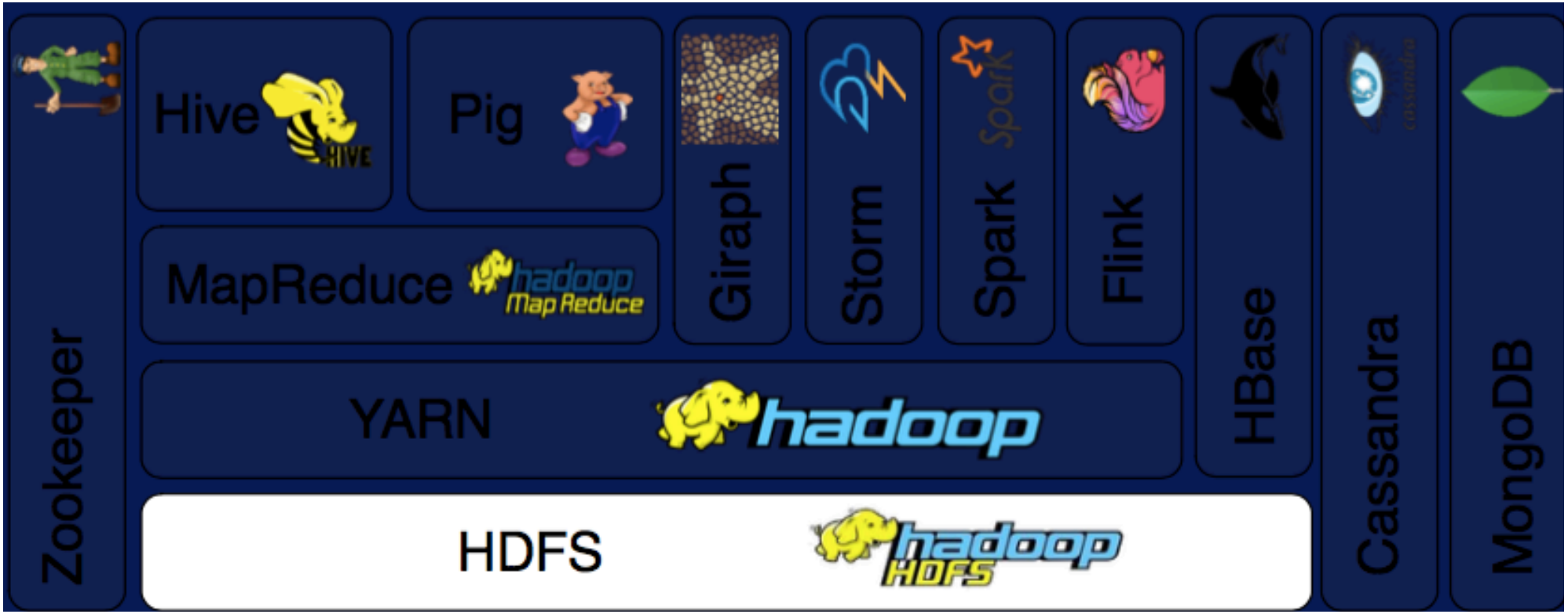
Today there are more that 100 open source projects for cloud (growing number)!



Hadoop Ecosystem

HDFS = Hadoop Distributed File System

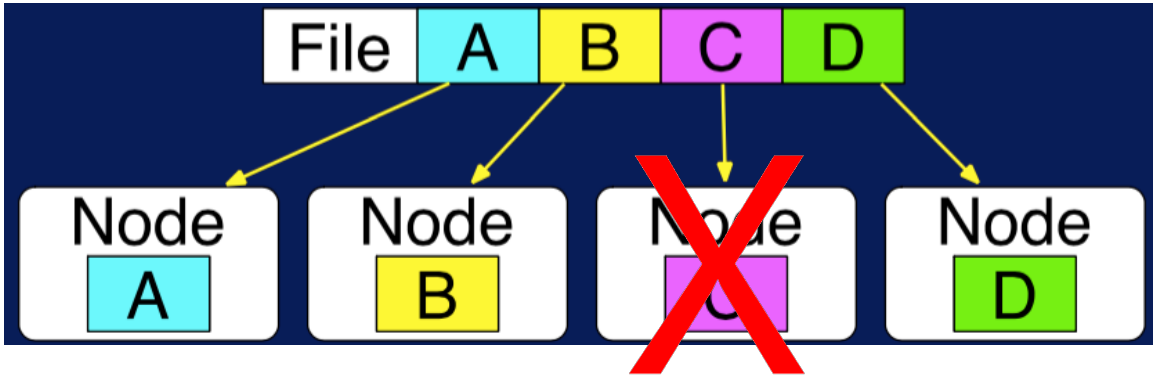
- Scalable storage
- Fault tolerance/Reliability
- Stores massively large datasets (PB)



HDFS

Cluster comprising up to 4500 nodes/servers in production

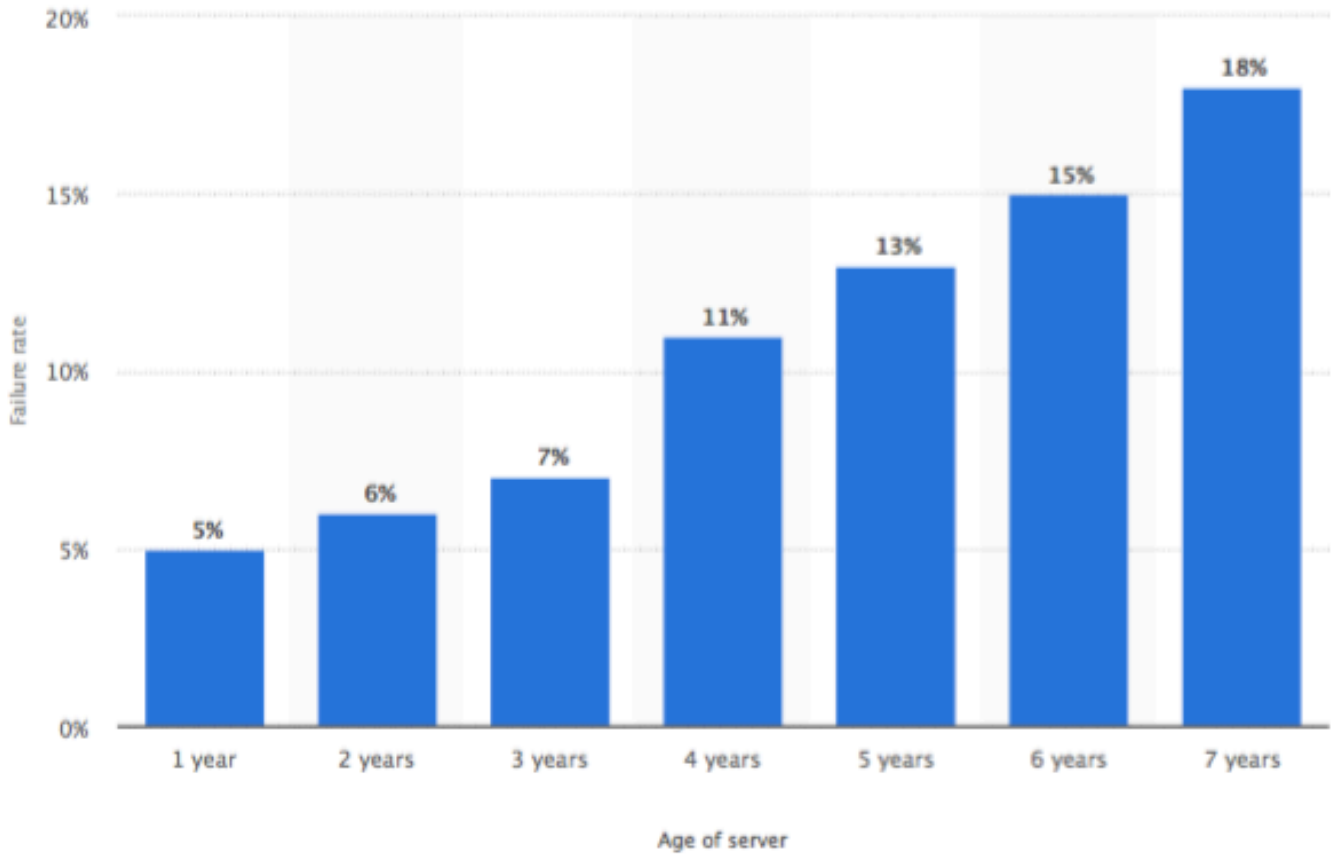
Splitting data files across multiple nodes -> parallel data processing
(scales up to 1 billion files and blocks)



- typical file size: GB - TB
- default chunk size of a file: 64 MB (configurable to any size)

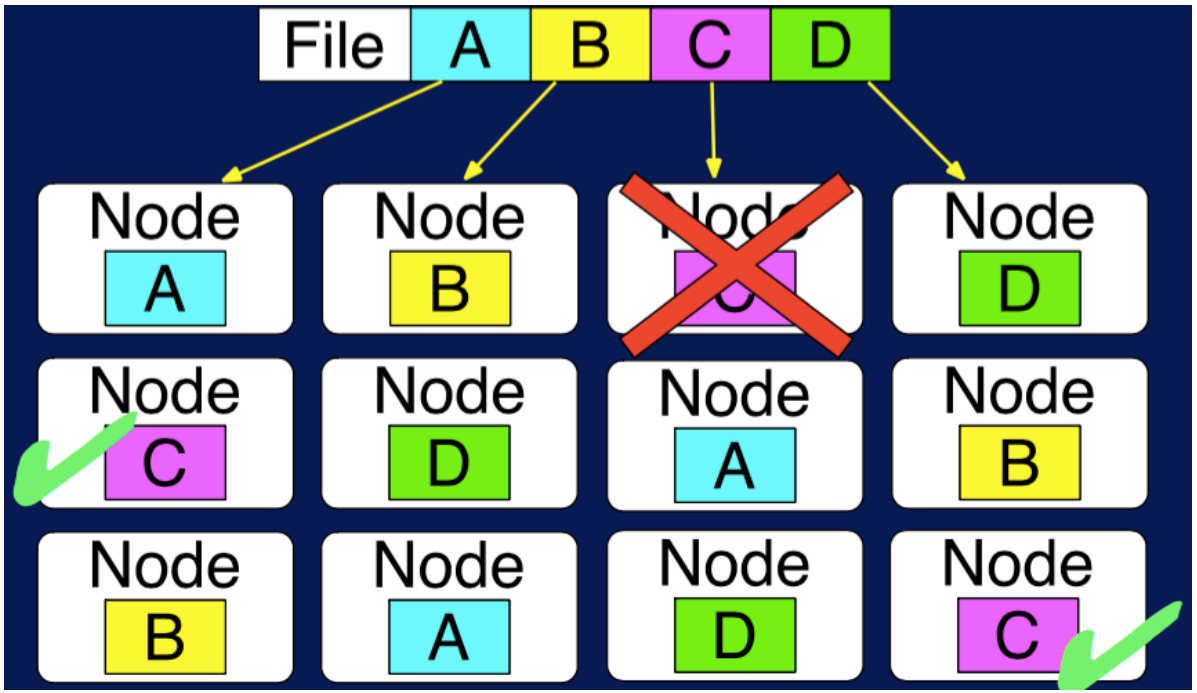
Infrastructure Failure

What is the average frequency of server failure based on the age of the server?



Infrastructure Failure

- Replication for fault tolerance
- HDFS default replication factor = 3 (configurable)

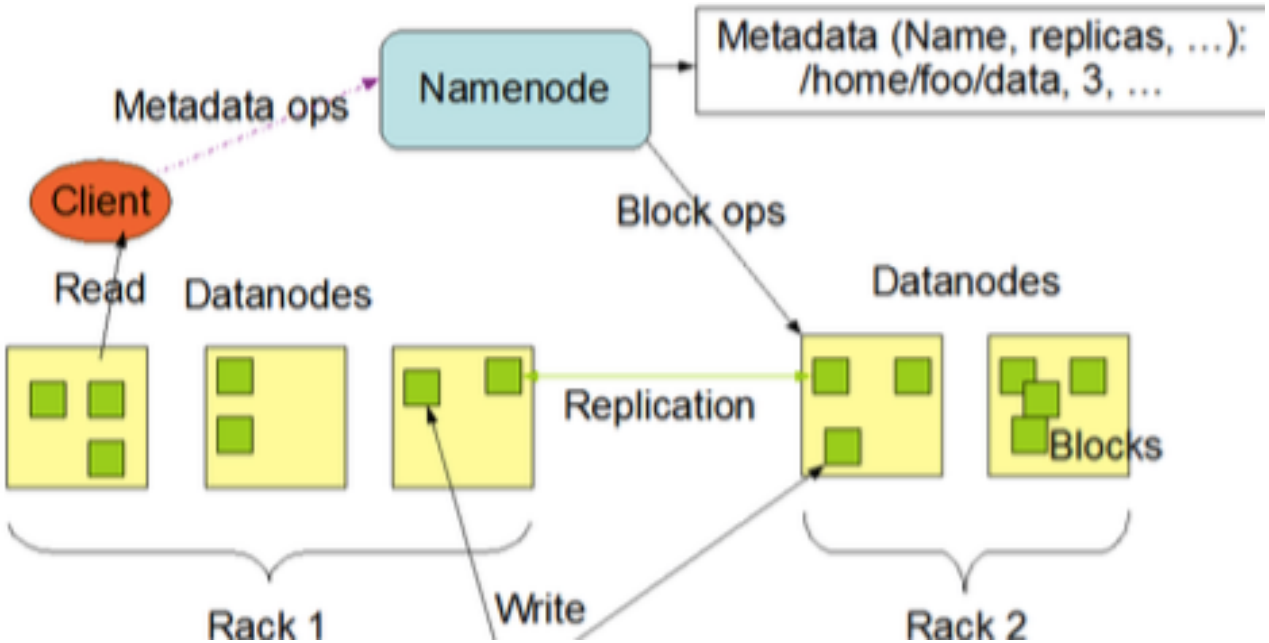


HDFS

HDFS has 2 components (master-slave)

- *Namenode* – keeps track of files name, location in directory hierarchy, stores *metadata*, issues commands, coordinates operations (block creation, deletion, replication);
  usually 1 namenode for each cluster;

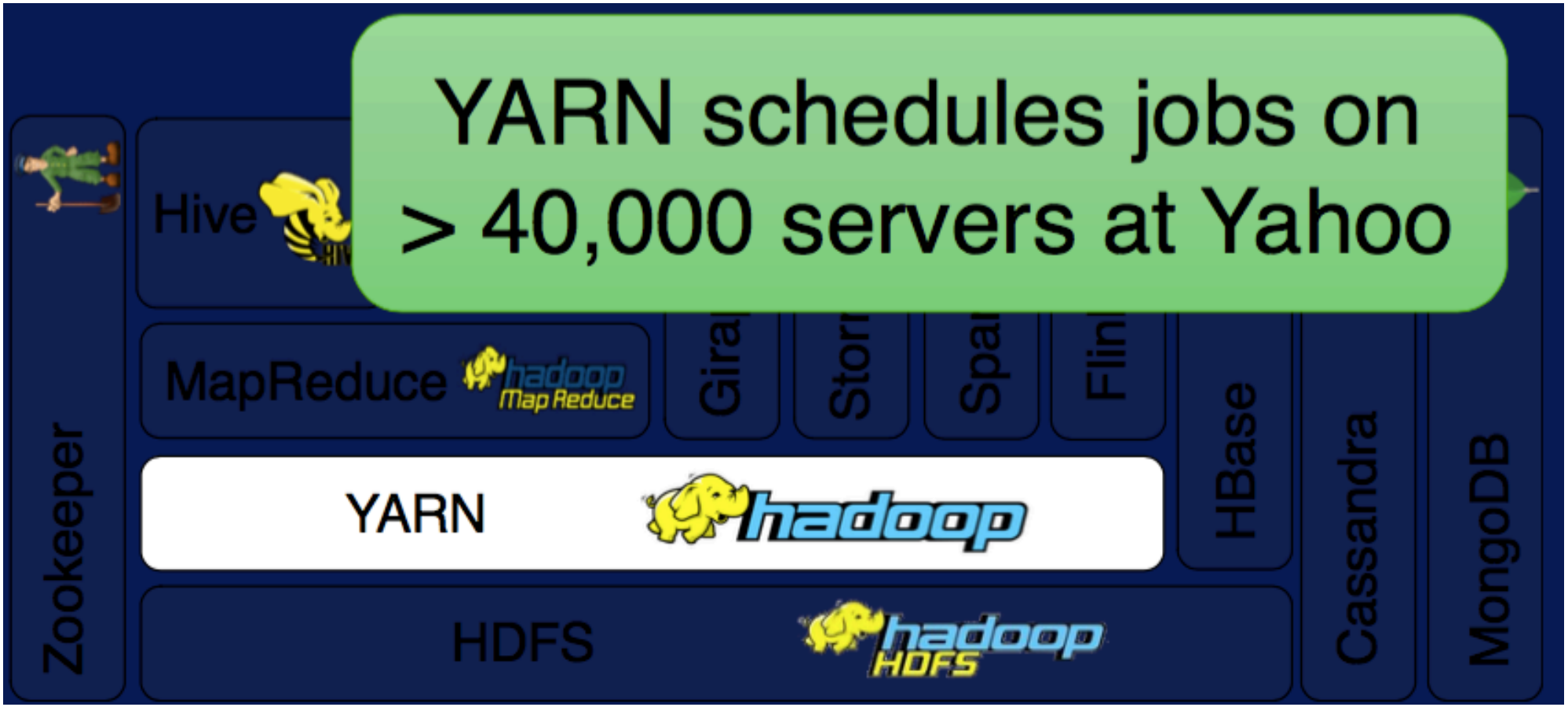
- *Datanodes* – provides *block storage*; 1 for each node;



Hadoop Ecosystem

YARN (Yet Another Resource Negotiator)

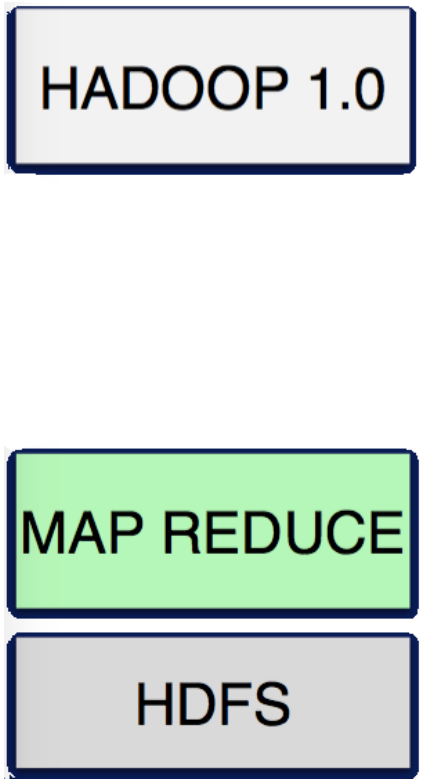
-> Flexible scheduling and resource management
-> Interacts with Applications and schedules resources



YARN

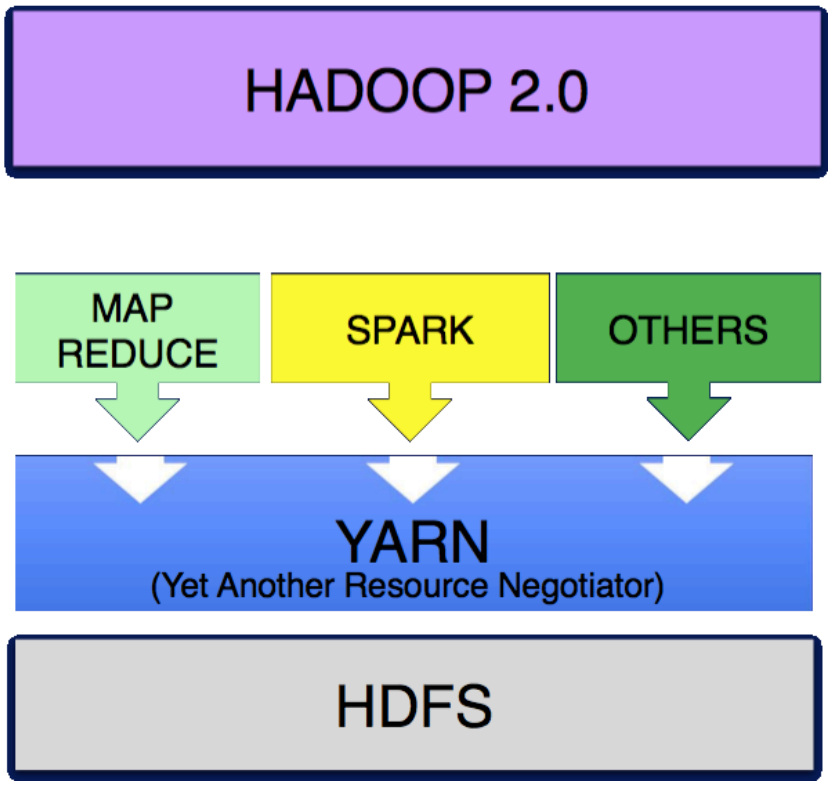
Hadoop 1.0

- no resource manager
- unable to support non-MapReduce applications
- poor resource utilization



Hadoop 2.0

- YARN extends Hadoop to enable to share HDFS cluster across multiple applications



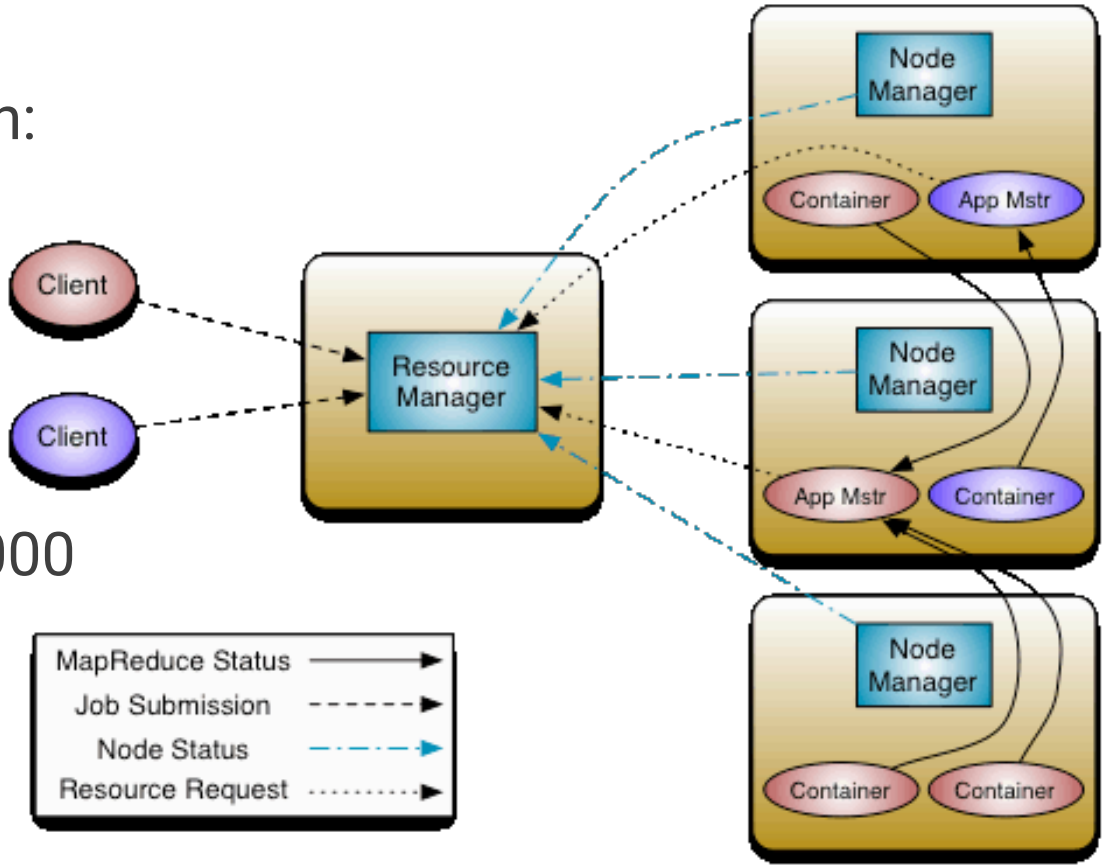
YARN Architecture Overview

- Resource Manager control all the resources
- Each application gets an Application Master
- Node Manager is in charge of a single machine
- App Master “negotiates” resources with Resource Manager and “talks” to Node Master to get its tasks completed

Higher resource utilization:

- 2X CPU utilization
- 2X jobs per day
- Equivalent to adding 1000 machines to a 2500 machine cluster

-> Lower cost



Hadoop Ecosystem

Simplified parallel programming model based on 2 functions:

- **Map**
- **Reduce**

MapReduce vs. Parallel programming

Traditional Parallel Programming Paradigm deals with:

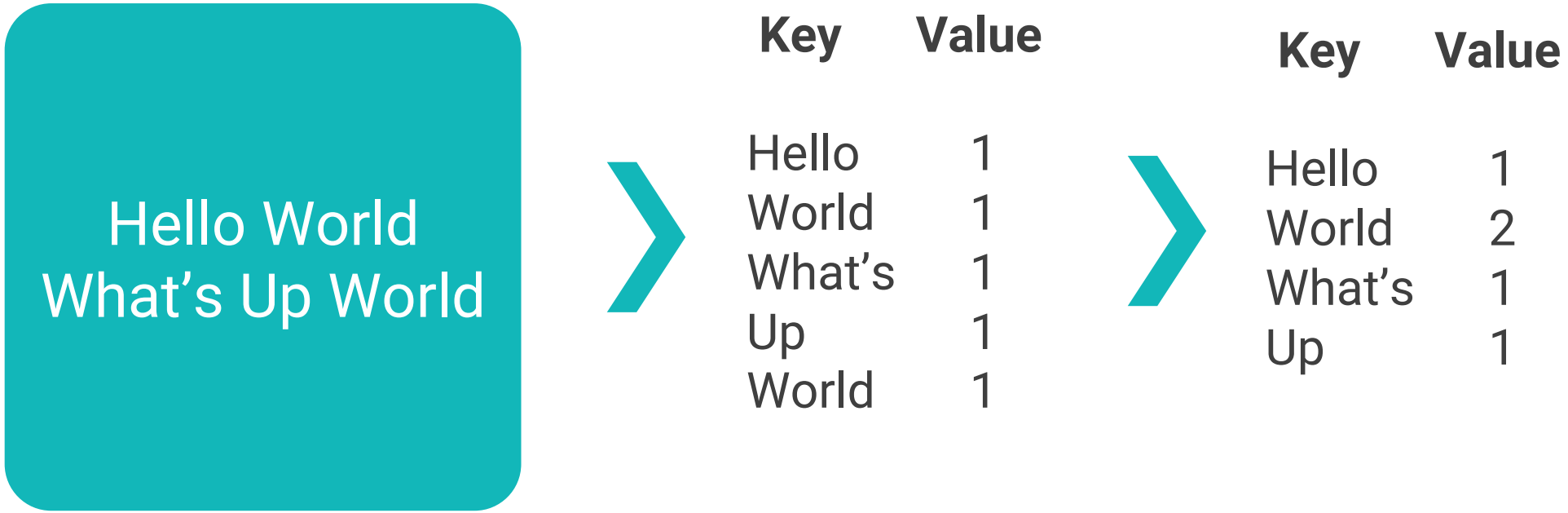
- synchronization mechanisms (locks, semaphores, monitors)
- parallel processes (threads, message passing, etc.)

MapReduce

- Only requires to define *Map* and *Reduce* tasks
- Don’t have to deal with synchronization or concurrency issues
- Based on functional programming

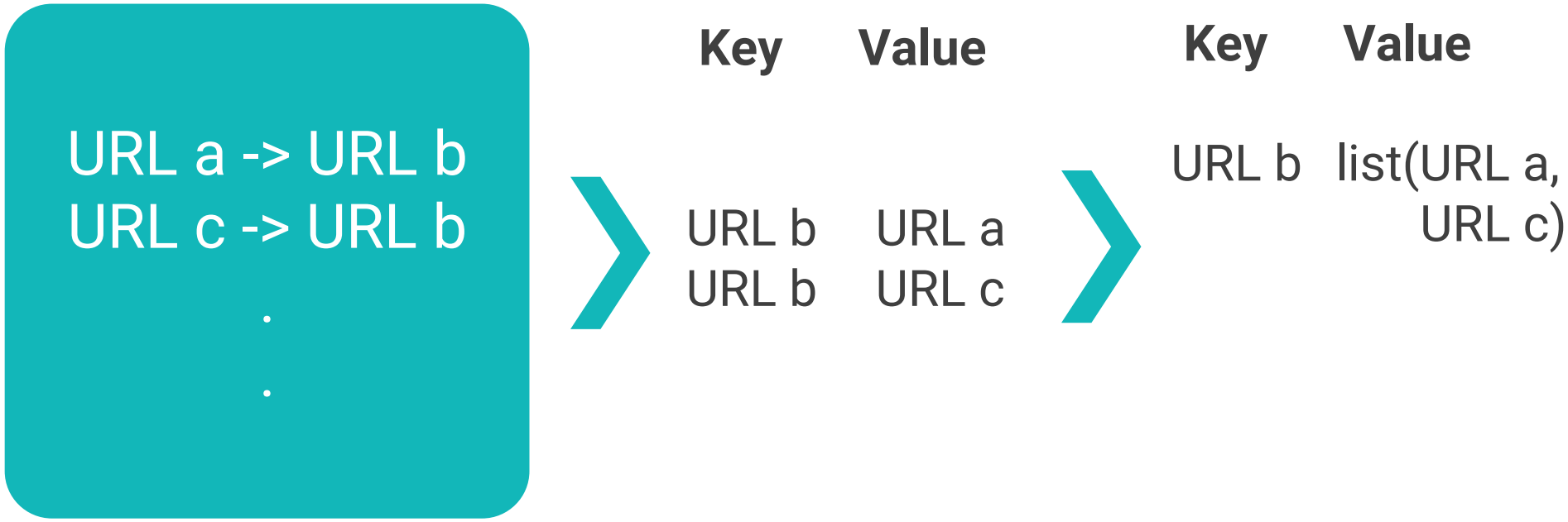
"Word Count" the "Hello World" for MapReduce

- *Map* applies operation to each individual record to generate key/value pairs *(parallel processing)*
- *Map* "goes" to each Node containing a block of the file
- *Reduce* summarizes operation by merging all values associated per key
- Each key corresponds to one *Reduce* task (add values for same key)



"Word Count" the "Hello World" for MapReduce

- Input file: Web-link Graph log file
- Goal1: For each page list the pages that link to it
- Goal2: For each page count the the # of pages that link to it
- Chain MapReduce jobs one after another



Some code.. (1)

```
public static class MapClass extends MapReduceBase
            implements Mapper<LongWritable, Text, Text,
              IntWritable> {
  private final static IntWritable one =
    new IntWritable(1);
  private Text word = new Text();

  public void map( LongWritable key, Text value,
            OutputCollector<Text, IntWritable> output,
            Reporter reporter)
    throws IOException {
    String line = value.toString();
    StringTokenizer itr = new StringTokenizer(line);
    while (itr.hasMoreTokens()) {
            word.set(itr.nextToken());
            output.collect(word, one);
    }
  }
} // Source: http://developer.yahoo.com/hadoop/tutorial/module4.html#wordcount
```

Some more code.. (2)

```
public static class ReduceClass extends MapReduceBase
            implements Reducer<Text, IntWritable, Text,
            IntWritable> {
  public void reduce(
            Text key,
            Iterator<IntWritable> values,
            OutputCollector<Text, IntWritable> output,
            Reporter reporter)
     throws IOException {
            int sum = 0;
            while (values.hasNext()) {
              sum += values.next().get();
            }
            output.collect(key, new IntWritable(sum));
  }
} // Source: http://developer.yahoo.com/hadoop/tutorial/module4.html#wordcount
```

Some more code.. (3)

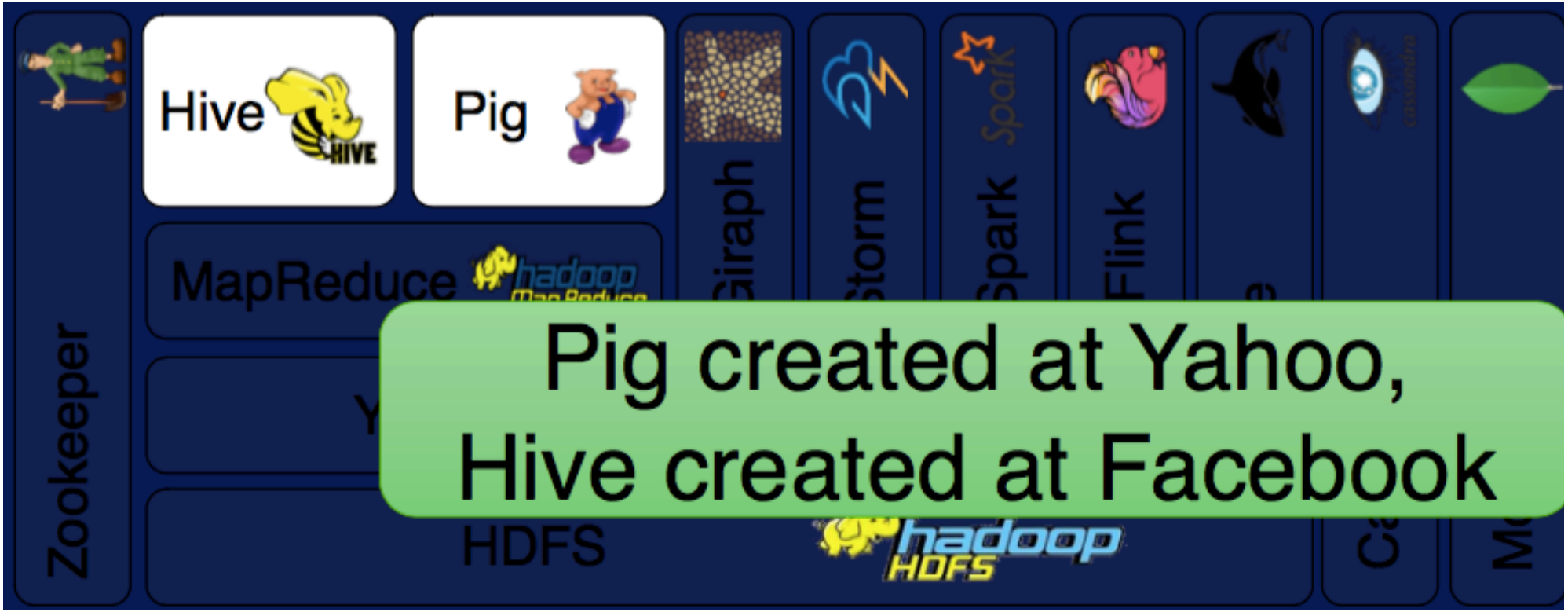
```
// Tells Hadoop how to run your Map-Reduce job
public void run (String inputPath, String outputPath)
            throws Exception {
  // The job. WordCount contains MapClass and Reduce.
  JobConf conf = new JobConf(WordCount.class);
  conf.setJobName("mywordcount");
  // The keys are words
  (strings) conf.setOutputKeyClass(Text.class);
  // The values are counts (ints)
  conf.setOutputValueClass(IntWritable.class);
  conf.setMapperClass(MapClass.class);
  conf.setReducerClass(ReduceClass.class);
  FileInputFormat.addInputPath(
            conf, newPath(inputPath));
  FileOutputFormat.setOutputPath(
            conf, new Path(outputPath));
  JobClient.runJob(conf);
} // Source: http://developer.yahoo.com/hadoop/tutorial/module4.html#wordcount
```

When NOT to use MapReduce

- Small datasets
- Online processing/Low latency
- Frequently changing data
  - MapReduce is slow – needs to process entire dataset
- Dependent tasks
  - MapReduce is best for independent batch tasks
  - Map and Reduce execute independent of each other
- Interactive analysis
  - MapReduce does not return results until the entire processing is finished

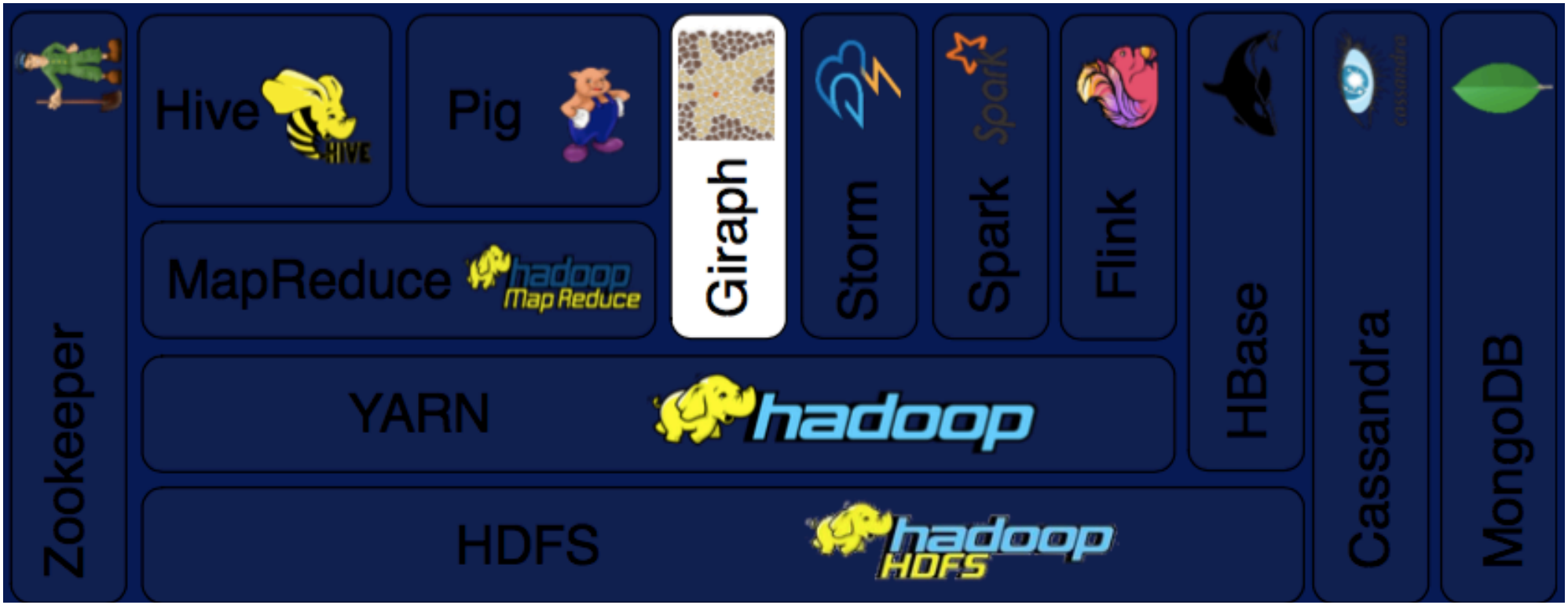
Hadoop Ecosystem

- ***Pig*** enables dataflow scripting
- ***Hive*** enables SQL-like queries/relational algebra



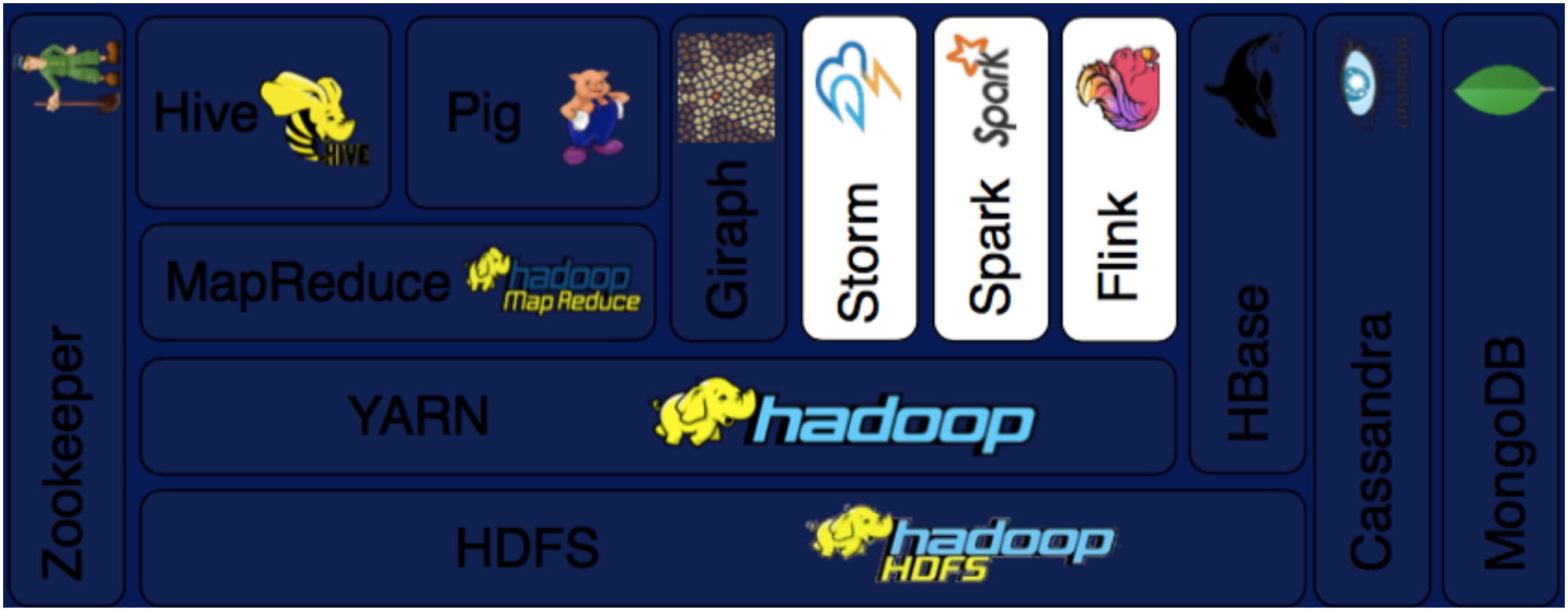
Hadoop Ecosystem

- ***Giraph*** used by Facebook to analyze social graphs
- Specialized models for graph processing



Hadoop Ecosystem

- Performs real-time in-memory processing
- Up to X 100 faster on certain tasks
- ***Spark*** – hybrid batch & stream processing; ML libraries
- ***Flink*** – stream processing with low latency
- ***Storm*** – stream processing in real-time; very low latency



Stream Processing

Why?

Large amounts of data that needs to be processed in *real-time*
(i.e. process events as they are generated/real-time views of data)

- Low latency requirements
- High throughput requirements

e.g. Social network trends, Twitter real-time search by hashtag, Google analytics, website click statistics, network intrusion detection, etc.

MapReduce is a batch processing framework:

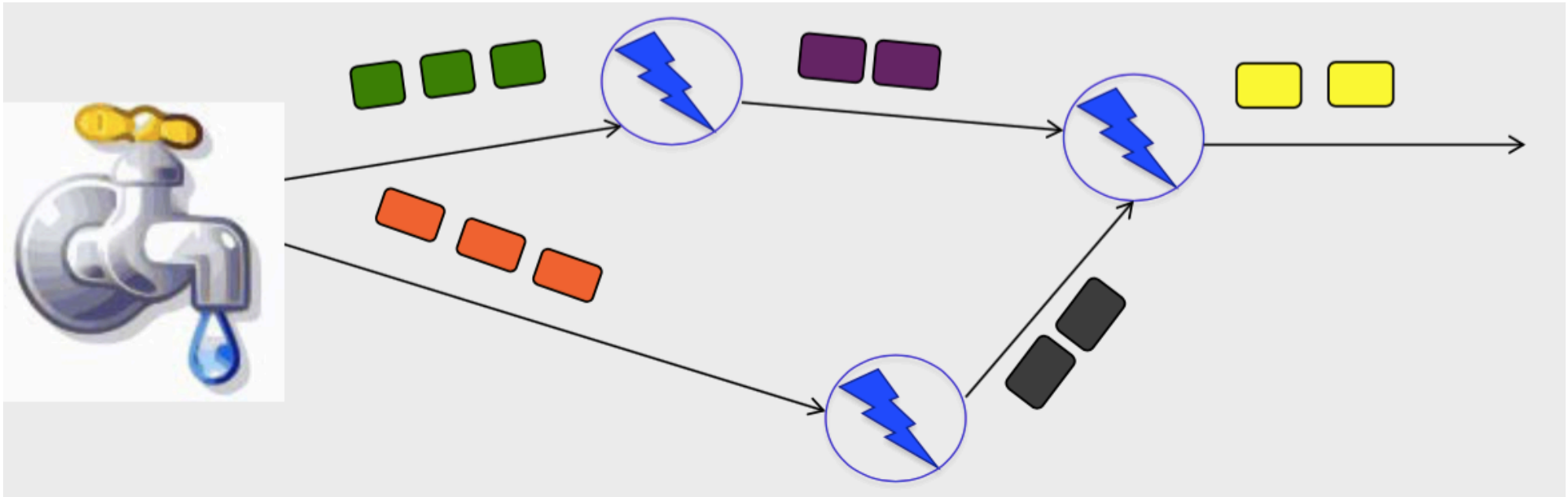
- Need to wait for entire computation on large dataset to complete
- Not intended for long-running stream processing
- No notion of partial results
- Each data transformation: Disk->Memory->Computation->Disk
  which makes it not ideal for complex ML algorithms

Storm - Stream Processing

Used by companies such as Twitter for personalized search, Flipboard, etc.

Storm Components:

- **Tuples**: ordered list of elements: <tweeter, tweet>
- **Streams**: sequence of tuples
- **Spouts**: process that generates a stream
- **Bolts**: entity that processes input stream(s)
- **Topologies**: directed graph of spouts and bolts – corresponds to a Storm Application, can contain cycles if it is required (caution to infinite loops!)



Storm - Stream Processing

Bolts

Bolts typically perform the following type of operations:

- **Filter**: forward only tuples which satisfy a condition
- **Joins**: When receiving two streams A and B, output all pairs (A,B) (cross product) which satisfy a condition
- **Apply/transform**: Modify each tuple according to a function
- **Aggregation**
- **Sinks**: special type of bolts that have an output interface

A Bolt can execute multiple tasks

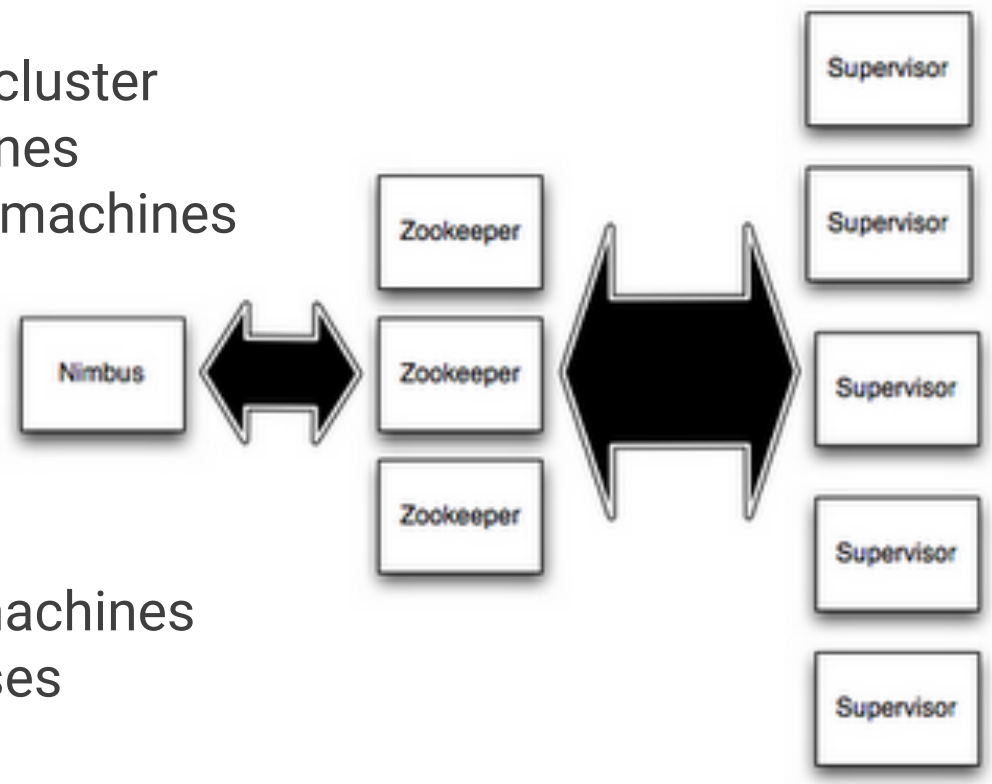
Each tuple goes to one task in the bolt according to *grouping* criteria

Storm - Stream Processing

Running a Storm cluster

Master node

- Runs a daemon called *Nimbus*
- Responsible for
    - Distributing code around cluster
    - Assigning tasks to machines
    - Monitoring for failures of machines



Worker node

- Runs on a machine (server)
- Runs a daemon called *Supervisor*
- Listens for work assigned to its machines
- Monitors its computation processes

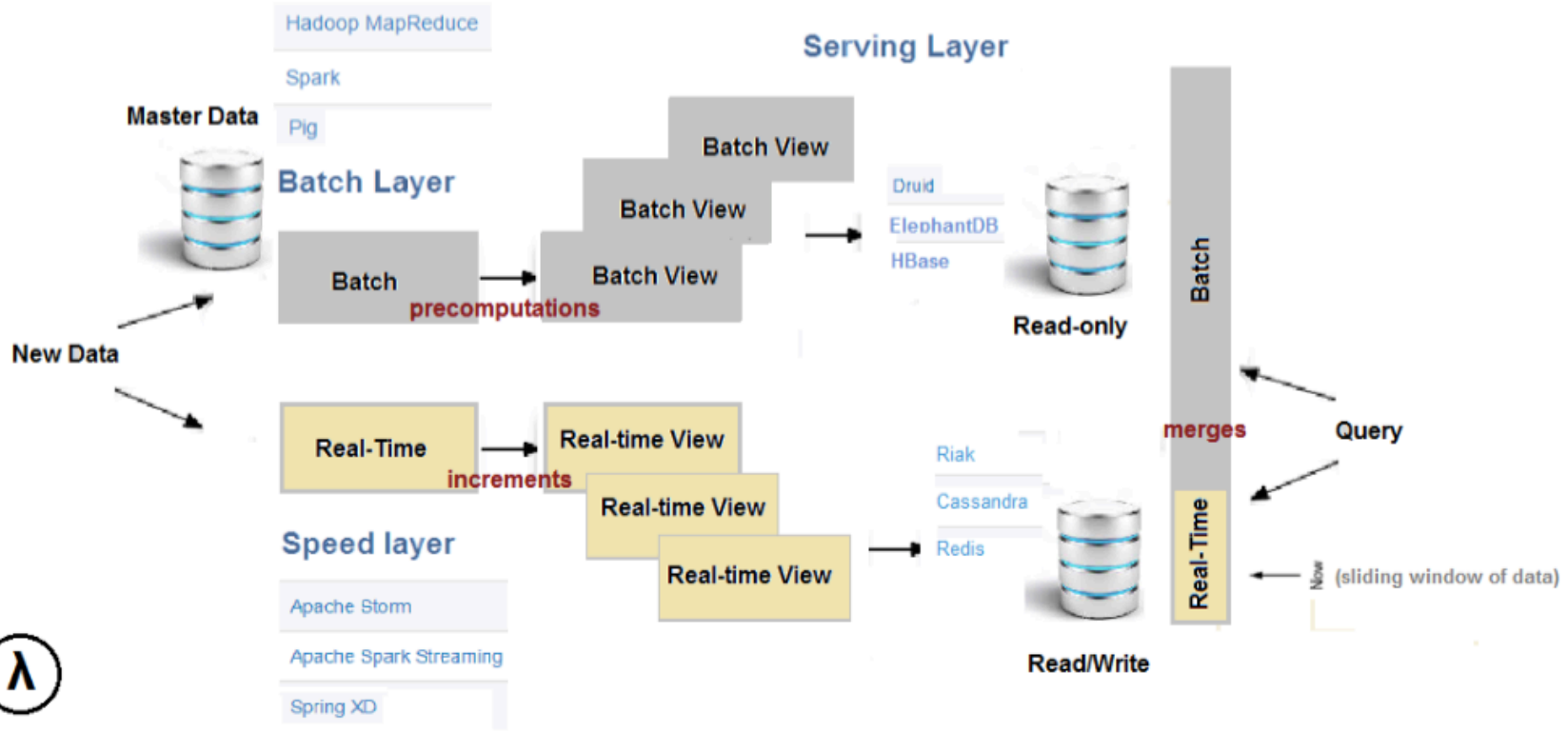
Zookeeper

- Coordinates Nimbus and Supervisors communication
- All state of Supervisor and Nimbus is kept here

Spark - Batch & Stream Processing

(Lambda Architecture)

- UC Berkley 2009/Apache 2013
- In-memory processing (up to 100 times faster)
- Discretized stream processing (e.g. batch length 1s - RDD)
- Interactive shell/Scripting languages (e.g. PySpark)
- Spark Stack includes: SparkSQL, Spark MLlib, Spark GraphX



Spark - Batch & Stream Processing

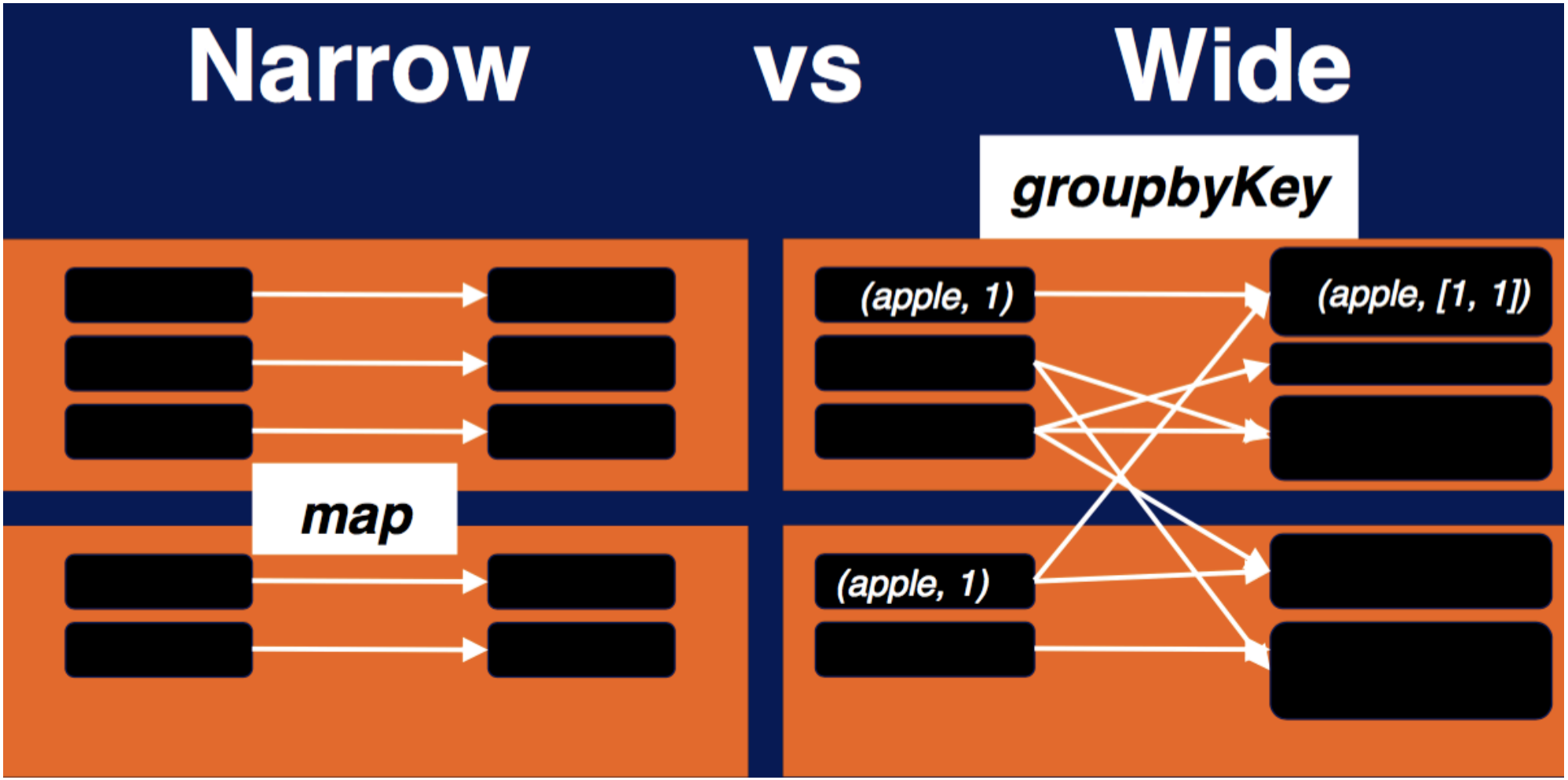
Programming in Spark Pipeline:

- Data streams converted to discrete RDDs via sliding window
  - Resilient Distributed Datasets(RDDs) can ingest data from batch data storage (e.g. HDFS, NoSQL) or streaming data (e.g. JSON)
  - Partitions of the data can be changed dynamically
- Transformations: e.g. filter transf., map transf.
  - Linear chain of RDD
- Action:
  - Last step in a Spark pipeline
  - Returns result of the transformations pipeline or stores result
  - Transformations are evaluated after an action is performed

Spark - Batch & Stream Processing

Narrow Transformation -> each node applies the RDD transformation locally
Wide Transformation -> requires shuffling the data across worker nodes

- processing depends on data residing on multiple partitions distributed across worker nodes



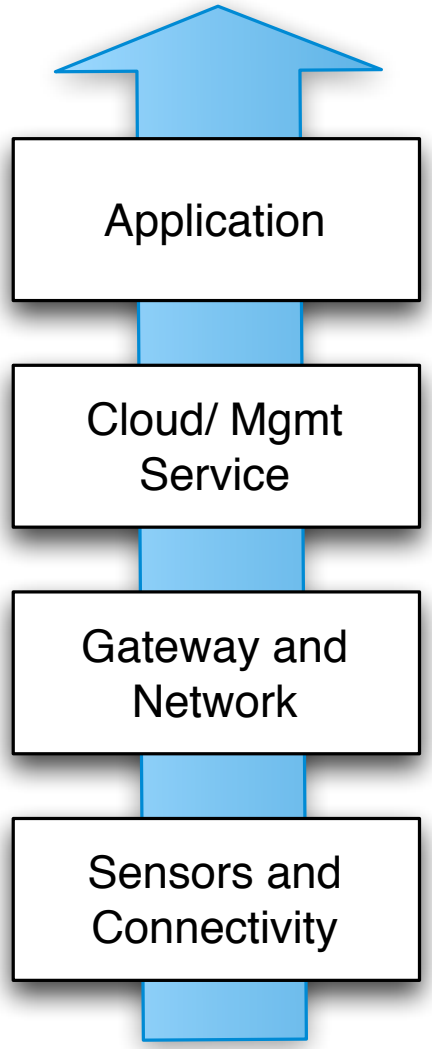
Hadoop Ecosystem

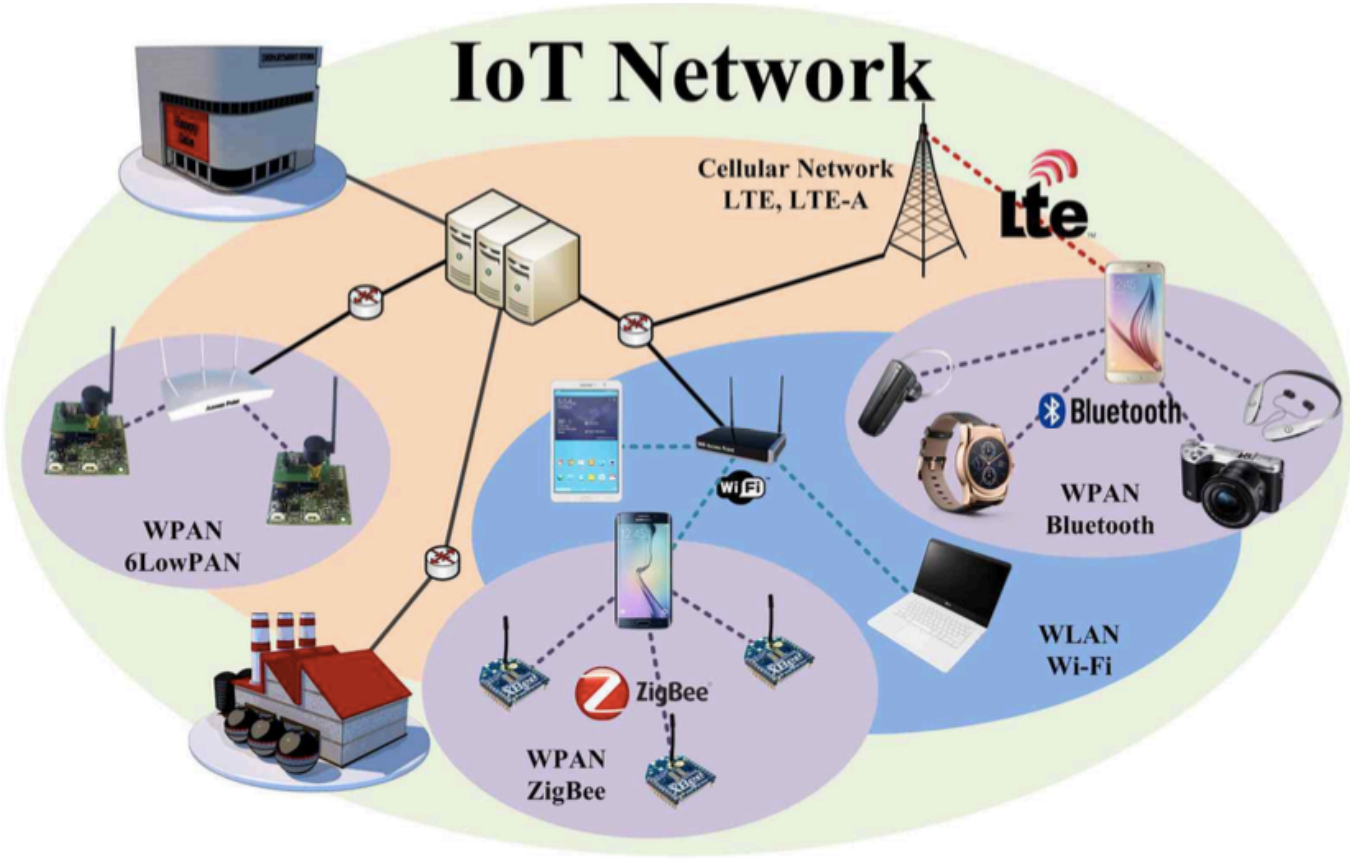
- **NoSQL** key-value stores
- **Cassandra** nodes communicating over a *gossip protocol,* used to discover location and state information about nodes inside the cluster.

Hadoop Ecosystem

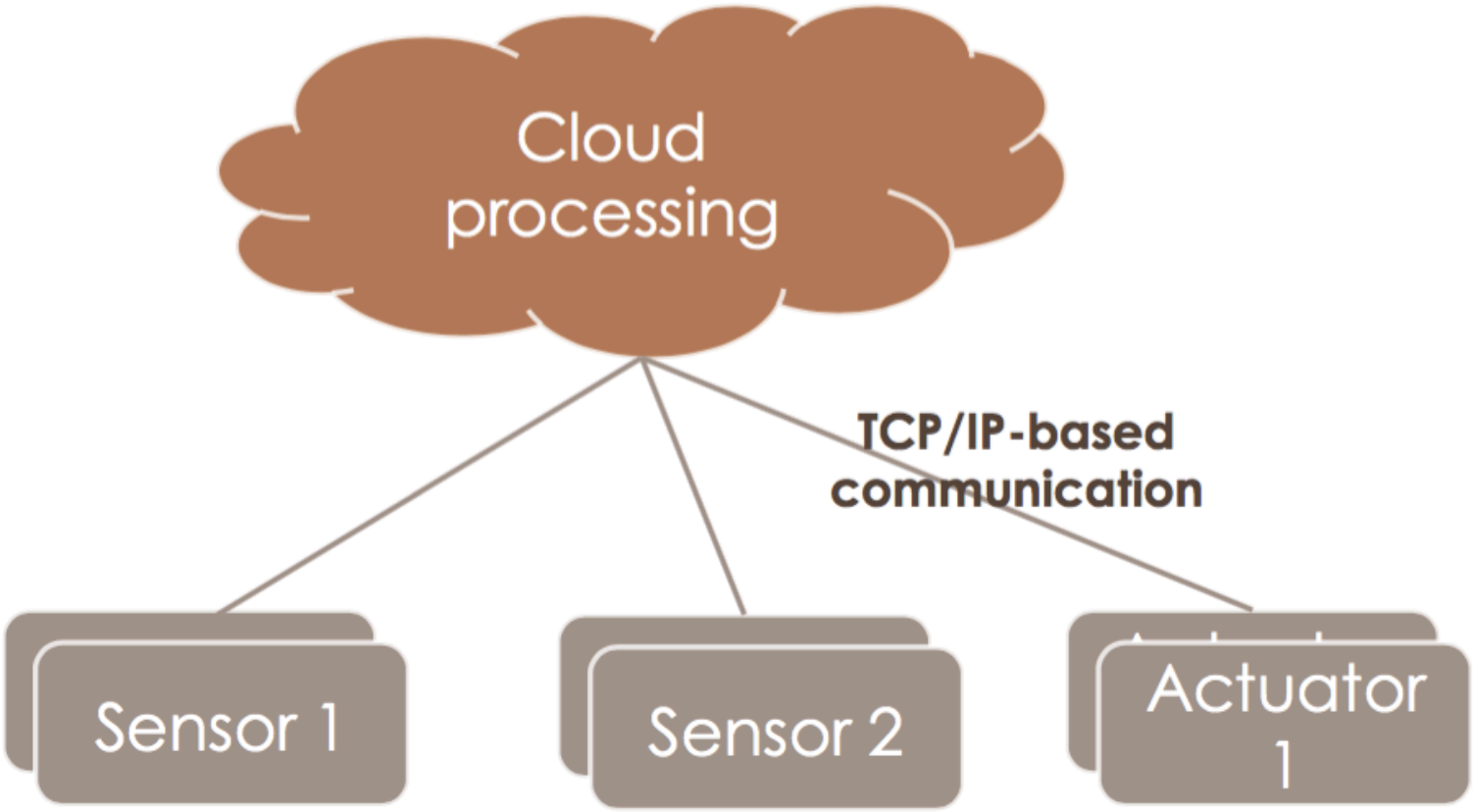
Zookeeper for resource management, synchronization, configuration, monitoring, etc.

IoT Architecture Layers

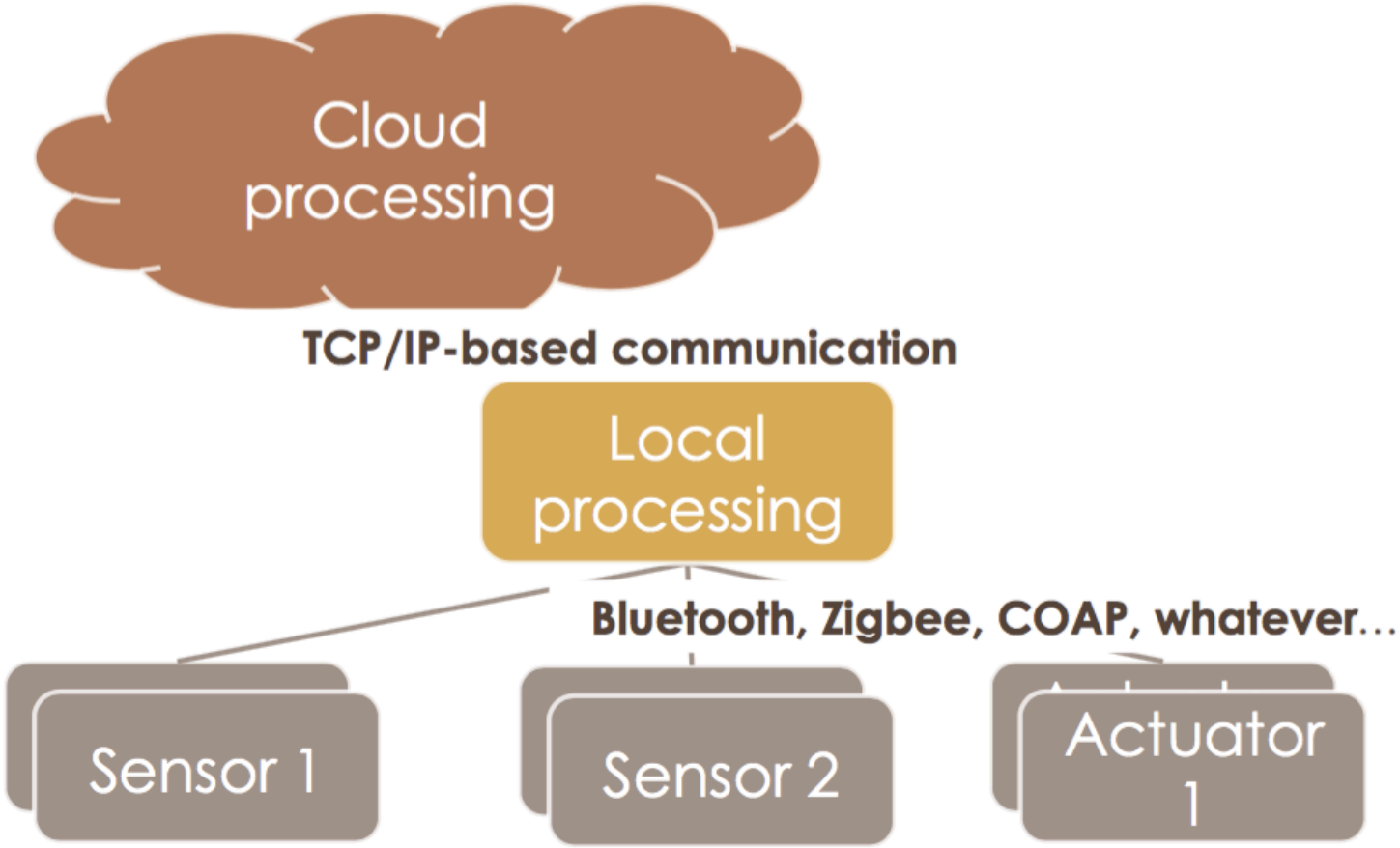




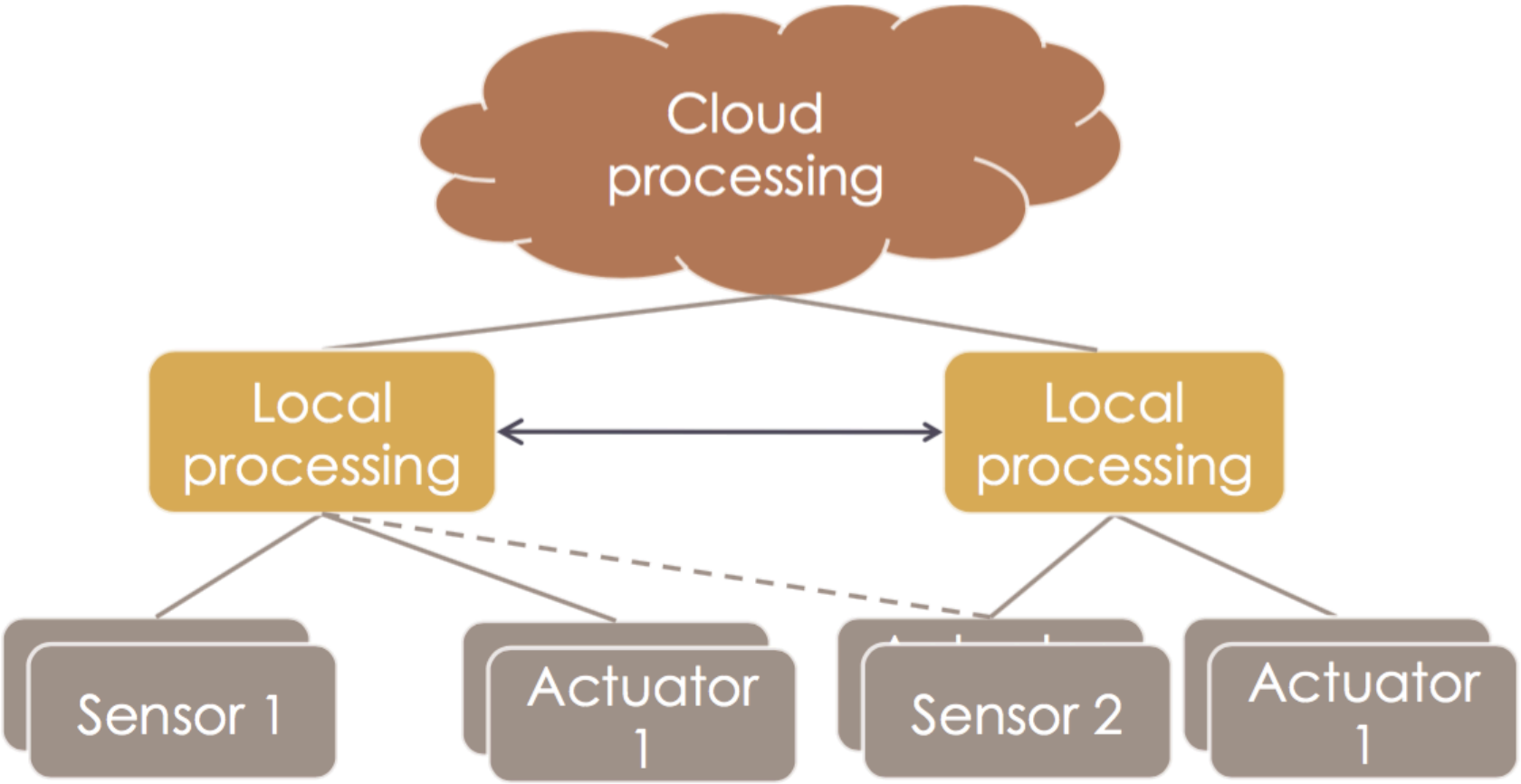
Just Cloud Data Processing



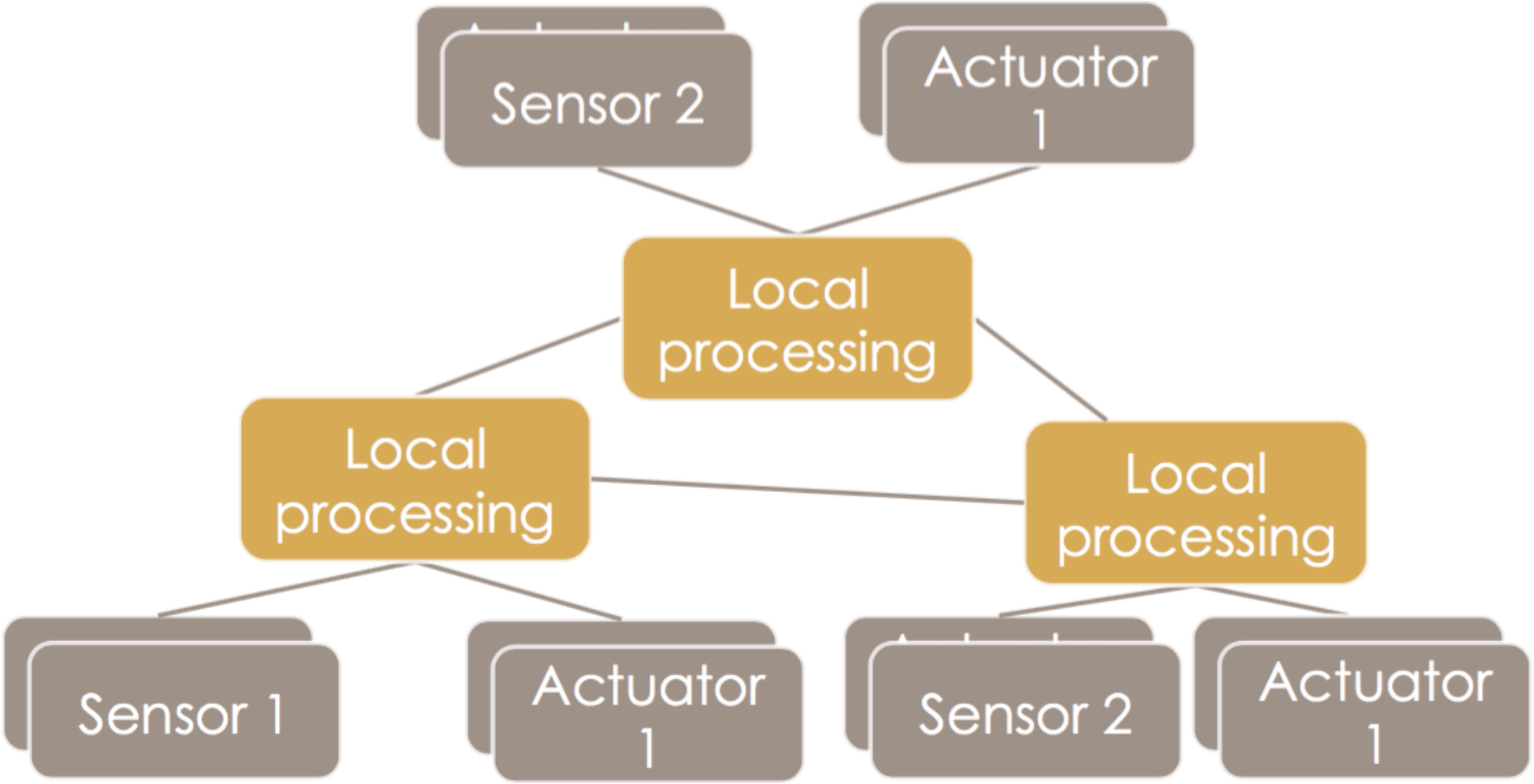
Hybrid Cloud + Local Data Processing



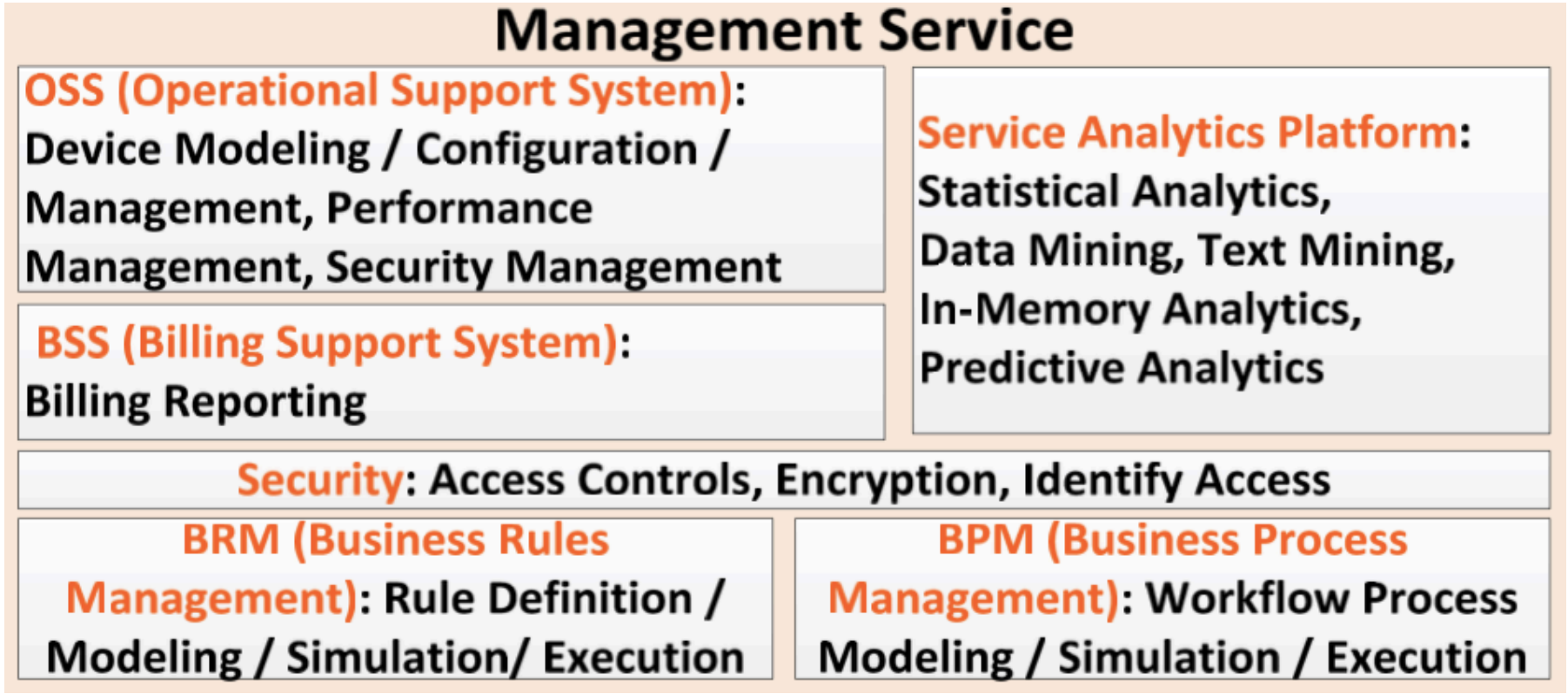
Hybrid with Local Collaboration



Purely Peer-to-peer



Cloud/Management Service Layer



- Management Service Layer is in charge of 1) Device Management 2) Information Analytics 3) Security Control 4) Process Modeling

Application Layer

Service Domain	Services
Smart Home	Entertainment, Internet Access
Smart Office	Secure File Exchange, Internet Access, VPN, B2B
Smart Retail	Customer Privacy, Business Transactions, Business Security, B2B, Sales & Logistics Management
Smart City	City Management, Resource Management, Police Network, Fire Department Network Transportation Management, Disaster Management
Smart Agriculture	Area Monitoring, Condition Sensing, Fire Alarm, Trespassing
Smart Energy & Fuel	Pipeline Monitoring, Tank Monitoring, Power Line Monitoring, Trespassing & Damage Management
Smart Transportation	Road Condition Monitoring, Traffic Status Monitoring, Traffic Light Control, Navigation Support, Smart Car Support, Traffic Information Support, ITS (Intelligent Transportation System)
Smart Military	Command & Control, Communications, Sensor Network, Situational Awareness, Security Information, Military Networking

References

* R. Khan, S. U. Khan, R. Zaheer, and S. Khan, ŒFuture Internet: The Internet of Things Architecture, Possible Applications and Key Challenges¹, in 2012 10th International Conference on Frontiers of Information Technology (FIT), 2012, pp. 257-260.

* Jeffrey Dean and Sanjay Ghemawat. 2004. MapReduce: simplified data processing on large clusters. In Proceedings of the 6th conference on Symposium on Operating Systems Design & Implementation, Vol. 6., Berkeley, CA, USA.

The End.