Cloud Computing

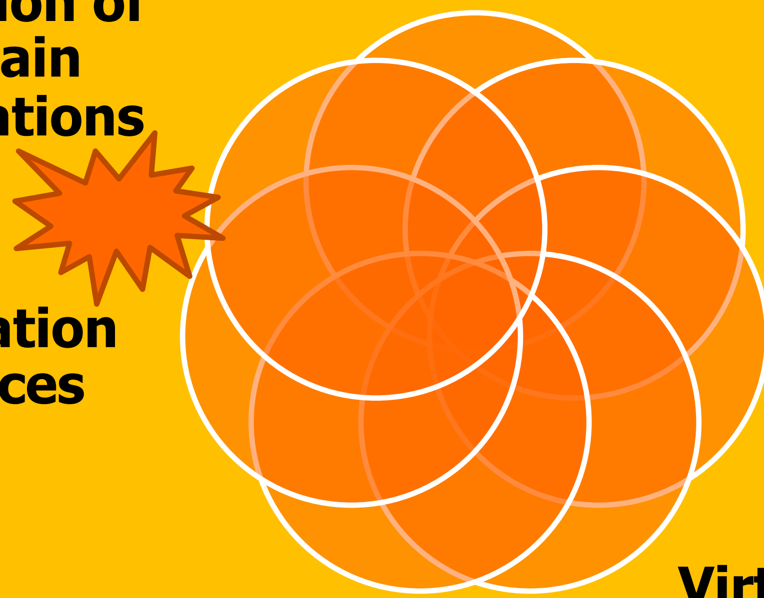
- Technology Context
- Cloud Models
- Cloud Capabilities
- MapReduce Application

A Golden Era in Computing

Powerful multi-core processors

Explosion of domain applications

General purpose graphic processors



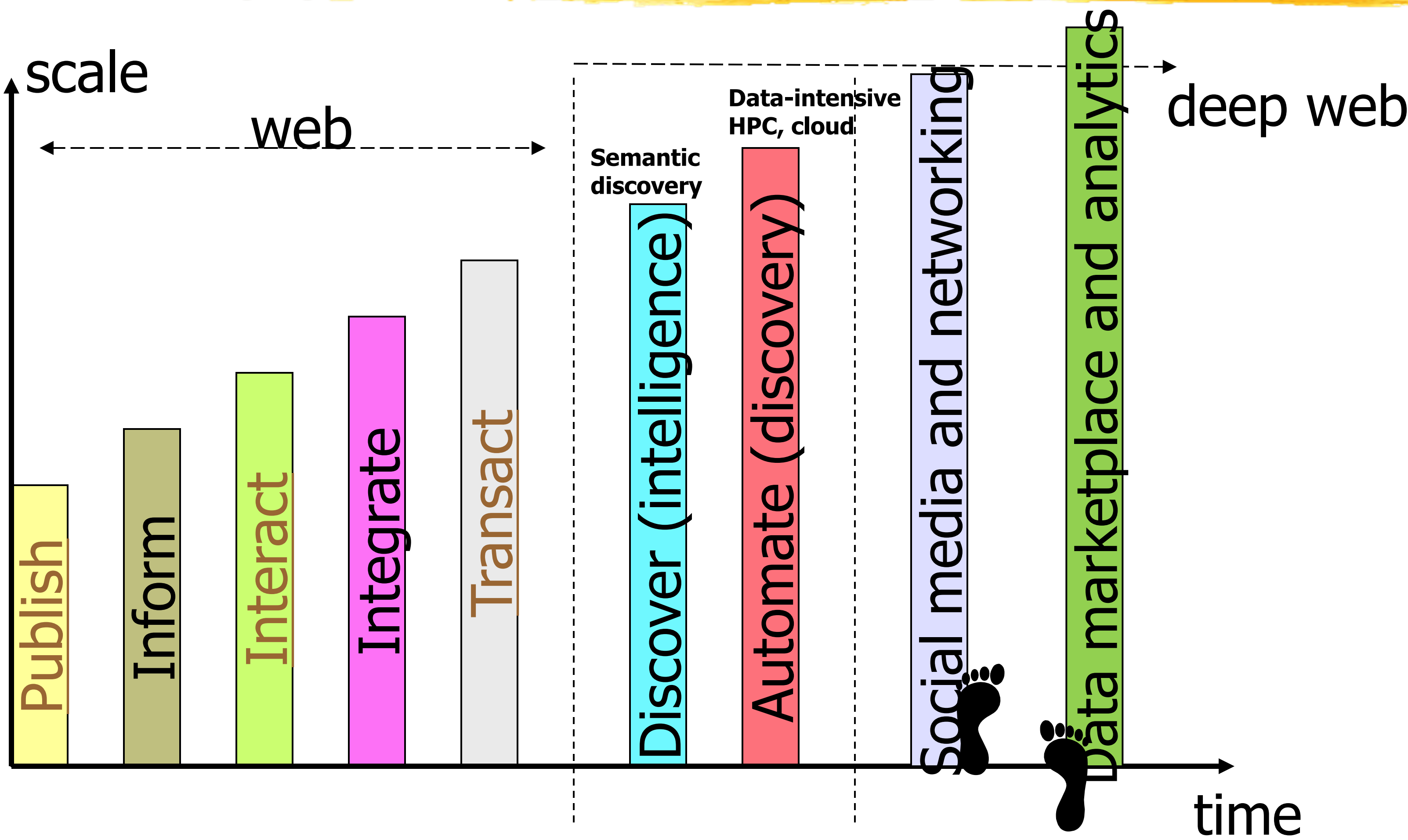
Proliferation of devices

Superior software methodologies

Wider bandwidth for communication

Virtualization leveraging the powerful hardware

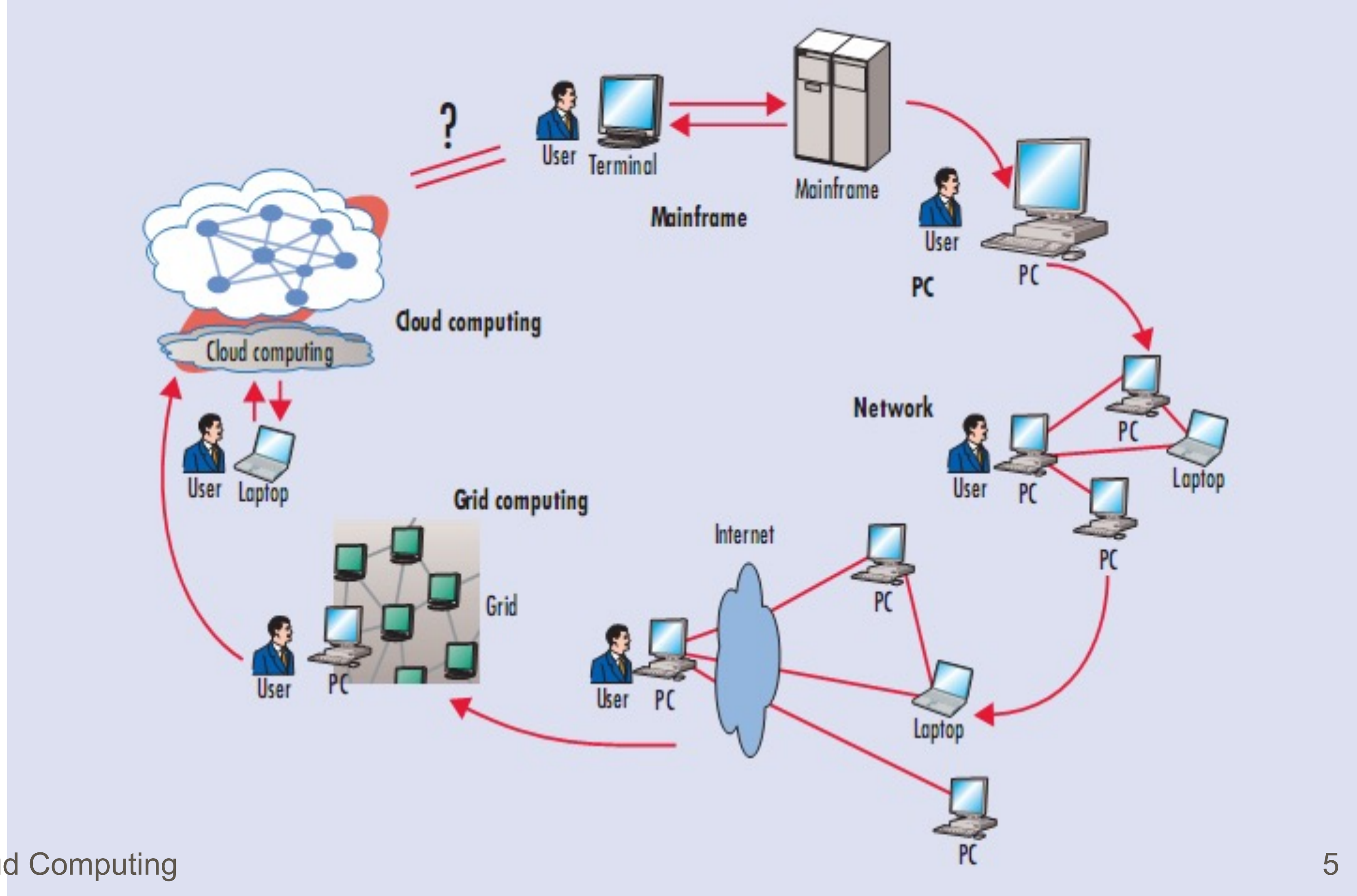
Evolution of Internet Computing



The world is changing...

- Explosive growth in applications: biomedical informatics, space exploration, business analytics, web 2.0 social networking: YouTube, Facebook
- Deluge of data / extraordinary rate of digital content consumption
- Exponential growth in compute capabilities: multi-core, storage, bandwidth, virtual machines (virtualization)
- Very short cycle of obsolescence in technologies: Windows 7→ Windows 11; Java versions; C→C#; Python
- Newer architectures: web services, persistence models, distributed file systems/repositories (Google, Hadoop), multi-core, wireless and mobile
- Diverse knowledge and skill levels of the workforce
- Very difficult to manage this complex situation with your traditional IT infrastructure

Computing Paradigm Shift



What is Cloud Computing?

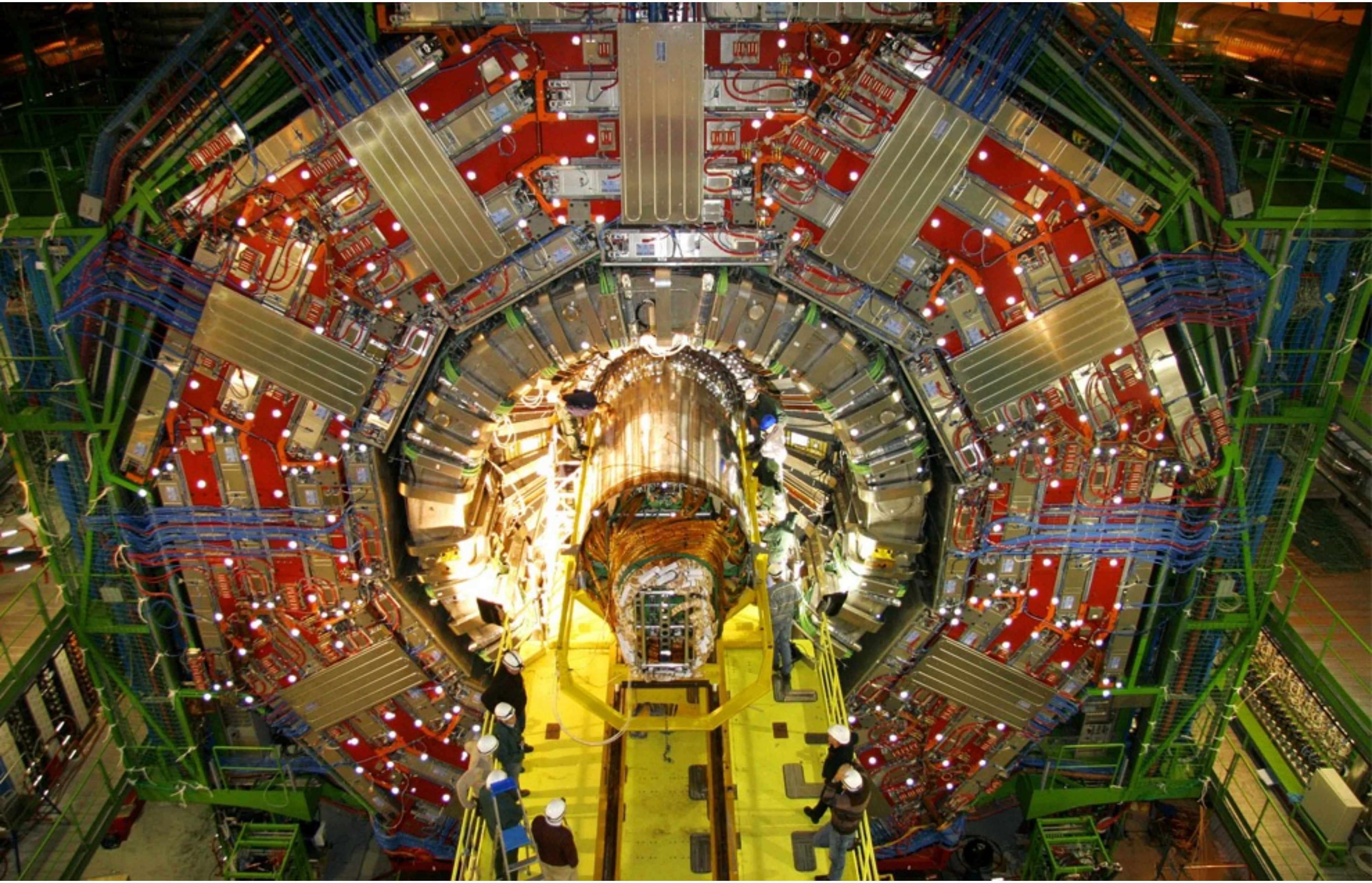
- Some Characteristics
  - Solves web-scale problems
  - Uses large data centers
  - Typically uses different models of computing
  - Highly-interactive Web applications

Web-Scale Problems

- Typically, highly data-intensive
  - May also be processing intensive
- Examples:
  - Crawling, indexing, searching, mining the Web
  - "Post-genomics" life sciences research
  - Other scientific data (physics, astronomy)
  - Sensor networks and Web 2.0 applications

How much is a lot of data?

- Google processes ~99,000 searches / sec
- "all words ever spoken by human beings" ~ 5 EB
- NOAA has multiple PB of climate data
- CERN's LHC generates about 25 PB / year



How more data helps?

- Answer factoid questions
  - What is the capital of Mali?
  - Pattern matching on the Web
- Learning relations from data
  - Start with seed instances
  - Search for patterns on the Web
  - Using patterns to find more instances

Large Data Centers

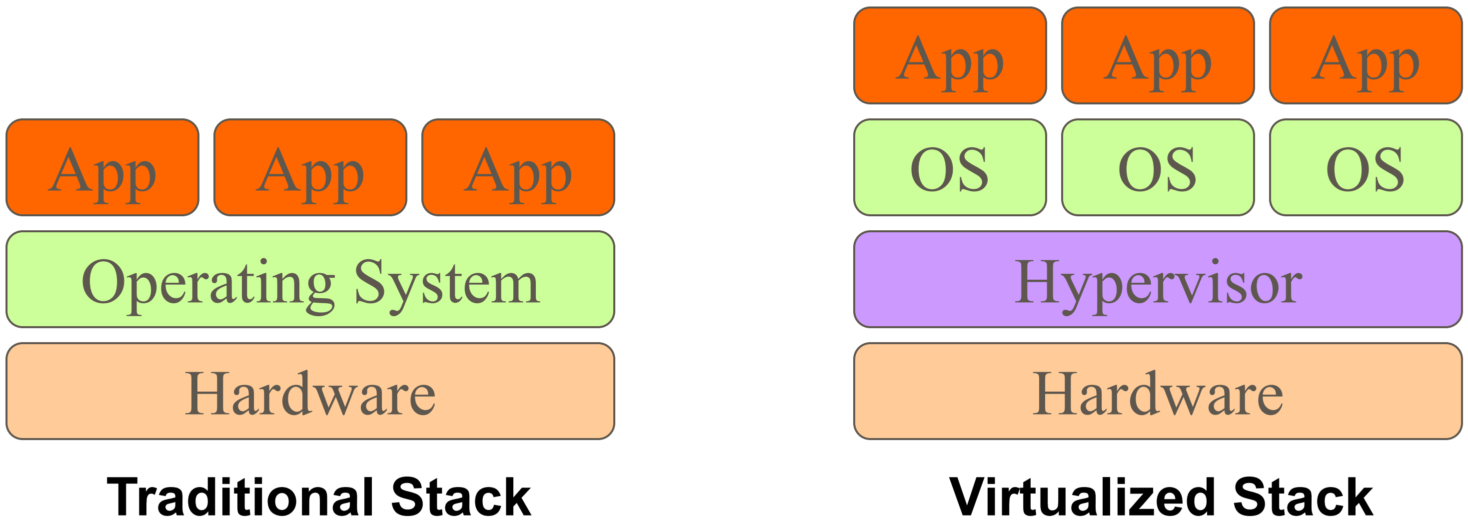
- How can we solve web-scale problems? Throw more machines at it!
- Centralization of computing resources in large data centers
  - Needs: good connectivity, cheap energy and space
- Important Issues are Redundancy, Efficiency, Utilization and Management

Maximilien Brice, © CERN

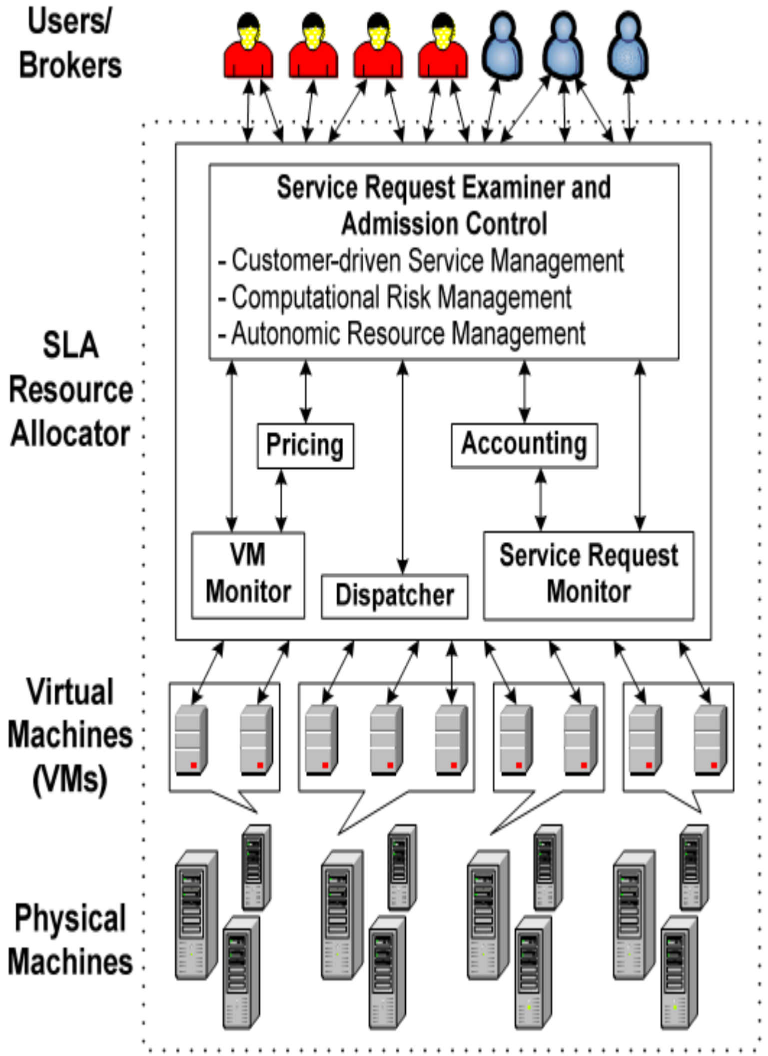
Cloud Computing Model

Acquisition Model **Service based**	"I only care about results, not how IT capabilities are implemented"
Business Model **Usage based**	"I want to pay for what I use, like a utility"
Access Model **Internet, Intranet**	"I can access services from anywhere, from any device"
Technical Model **Dynamic, flexible**	"I can scale up or down capacity, as needed"

Virtualization is the key

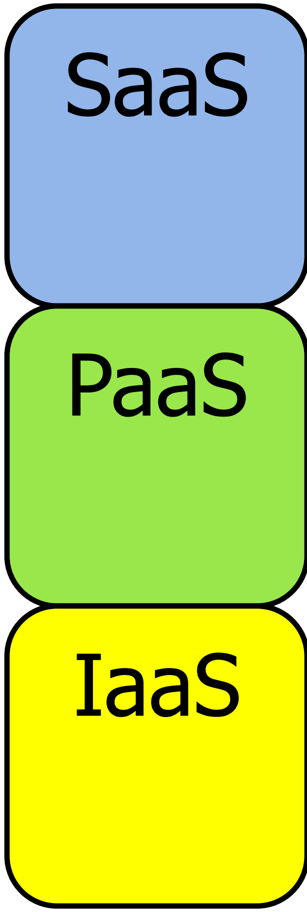


Cloud Architecture



Cloud Computing Services

"Why do it yourself if you can pay someone to do it for you?"



- Software as a Service (SaaS)
  - Just run it for me!
  - Example: Gmail
- Platform as a Service (PaaS)
  - Give me nice API and take care of the implementation
  - Example: Google App Engine
- Infrastructure as a Service (IaaS)
  - Why buy machines when you can rent cycles?
  - Examples: Amazon's EC2, GoGrid, AppNexus

Software Delivery Model

SaaS

- Increasingly popular with SMEs
- No hardware or software to manage
- Service delivered through a browser

Examples of SaaS

- CRM (e.g. salesforce.com)
- Financial Planning
- Human Resources
- Word processing
- gmail

Platform Delivery Model

PaaS

- Platforms are built upon expensive infrastructure
- Estimating demand isn’t easy
- Platform management can also be difficult and expensive

Typical PaaS Services

PaaS

- Storage
- Database
- Scalability

Some PaaS Providers

PaaS

- Google App Engine
- Rackspace.com
- AWS: S3

IaaS Delivery Model

IaaS

Access to infrastructure stack

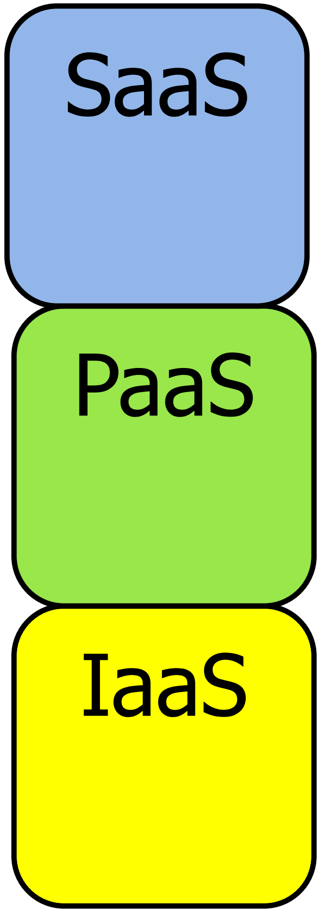
- Full OS access
- Firewalls
- Routers
- Load balancing

Some IaaS Providers

IaaS

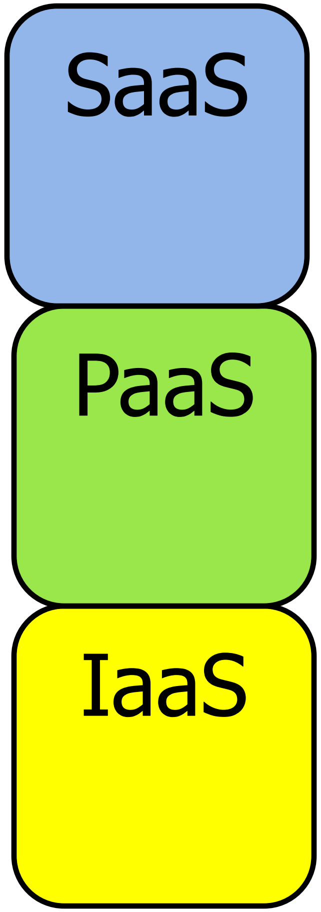
- Windows Azure
- AWS: EC2
- Flexiscale

Cloud Features



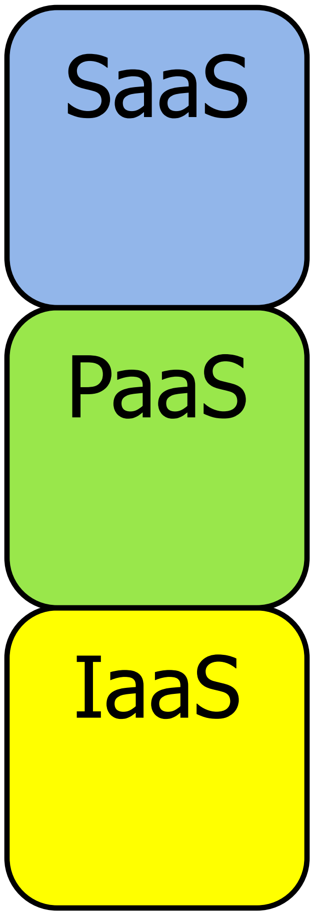
- **Pay per use**
- **Instant Scalability**
- **Security**
- **Reliability**
- **APIs**

Cloud Advantages



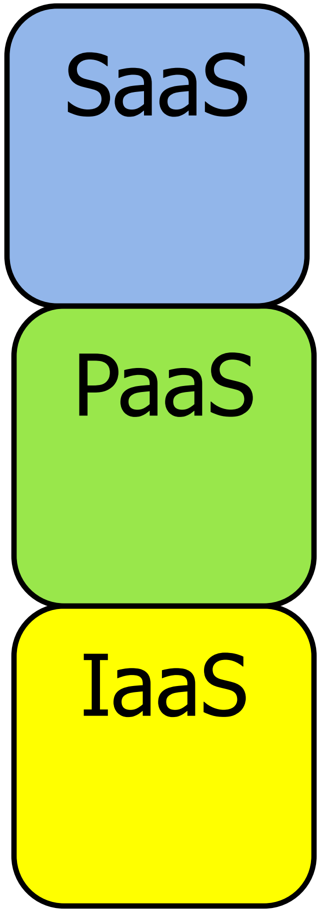
- **Lower cost of ownership**
- **Reduce infrastructure management responsibility**
- **Allow for unexpected resource loads**
- **Faster application rollout**

Economics of the Cloud



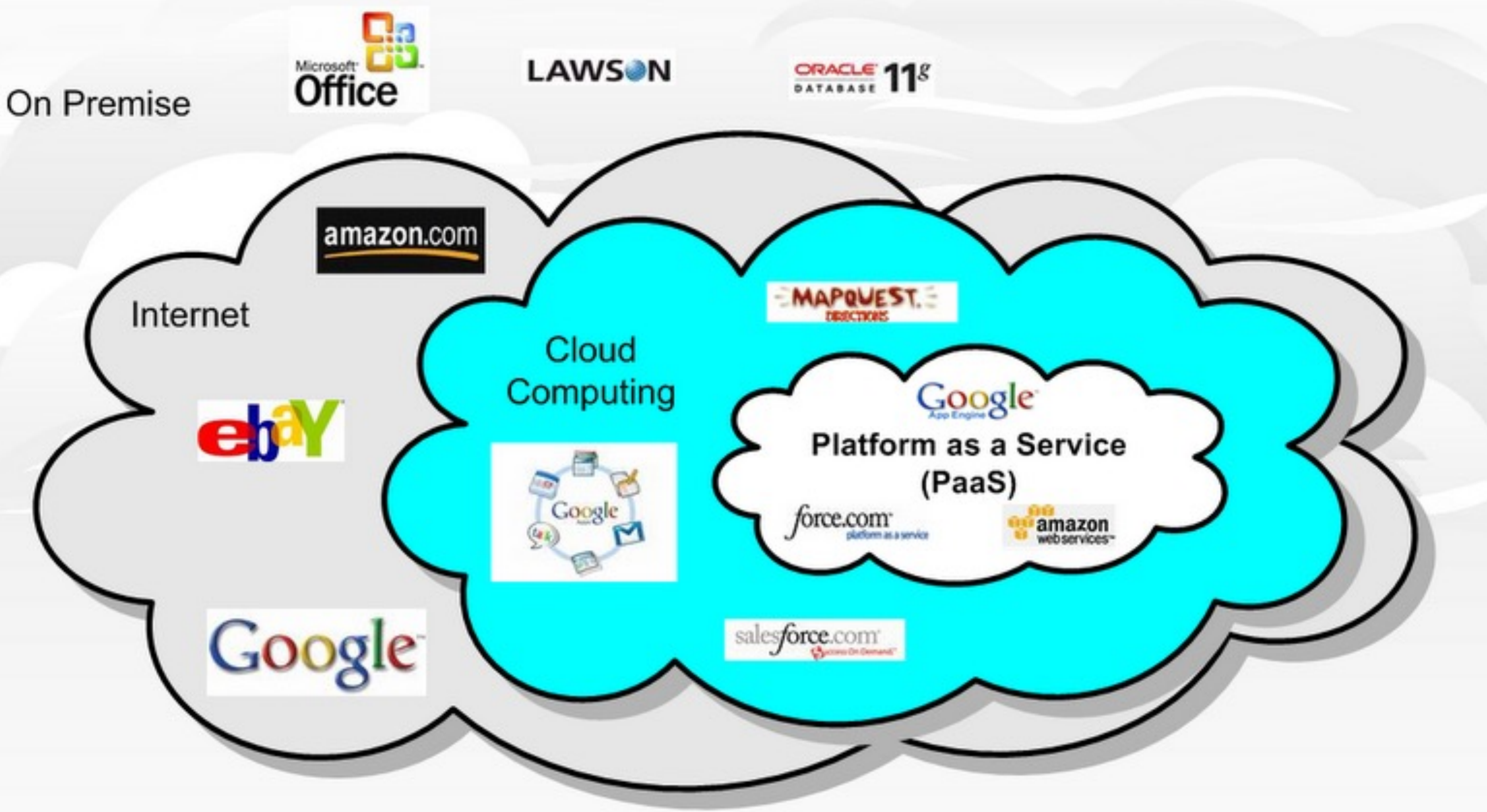
- Multi-tenented
- Virtualisation lowers costs by increasing utilisation
- Economies of scale afforded by technology
- Automated update policy

Risks of Cloud Computing



- **Security**
- **Downtime**
- **Access**
- **Dependency**
- **Interoperability**

Commercial clouds


On Premise
Microsoft Office
LAWSON
ORACLE DATABASE 11g
Internet
amazon.com
ebay
Google
Cloud Computing
MAPQUEST
Google App Engine
Platform as a Service (PaaS)
force.com platform as a service
amazon web services
salesforce.com

AWS

- Signature features: EC2, S3, Cloud Management Console, MapReduce Cloud, Amazon Machine Image (AMI)
- Excellent distribution, load balancing, cloud monitoring tools
- Good online videos and tutorials

AWS

- Elastic Compute Cloud (EC2)
  - Rent computing resources by the hour with additional costs for bandwidth
  - Facilitate computations through Amazon Machine Images (AMIs) with multiple OS images supported
- Simple Storage Service (S3)
  - Persistent storage charged by the GB/month with additional costs for bandwidth

Google App Engine

- This is more a web interface for a development environment that offers a one stop facility for design, development and deployment for Java and Python-based applications.
- Google offers the same reliability, availability and scalability as with Google’s own applications

Google App Engine

- Interface is software programming based
- Comprehensive programming platform irrespective of the size (small or large)
- Signature features: templates and appspot, excellent monitoring and management console
- Plugin for Eclipse is available

Windows Azure

- Enterprise-level on-demand capacity builder
- Fabric of cycles and storage available on-request for a cost
- You have to use Azure API to work with the infrastructure offered by Microsoft
- Significant features: web role, worker role, blob storage, table and drive-storage

Choosing a Commercial Platform

- Revise cost model to utility-based computing: CPU/hour, GB/day etc.
- Include hidden costs for management, training
- Different cloud models for different applications - evaluate
- Use for prototyping applications and learn
- Link it to current strategic plans for Services-Oriented Architecture, Disaster Recovery, etc.

Cloud Programming Models

- Map Reduce: the “back-end” of cloud computing
  - Batch-oriented processing of large datasets
- Ajax: this often provides the “front-end” of cloud computing
  - Highly-interactive Web-based applications
  - Asynchronous JavaScript and XML

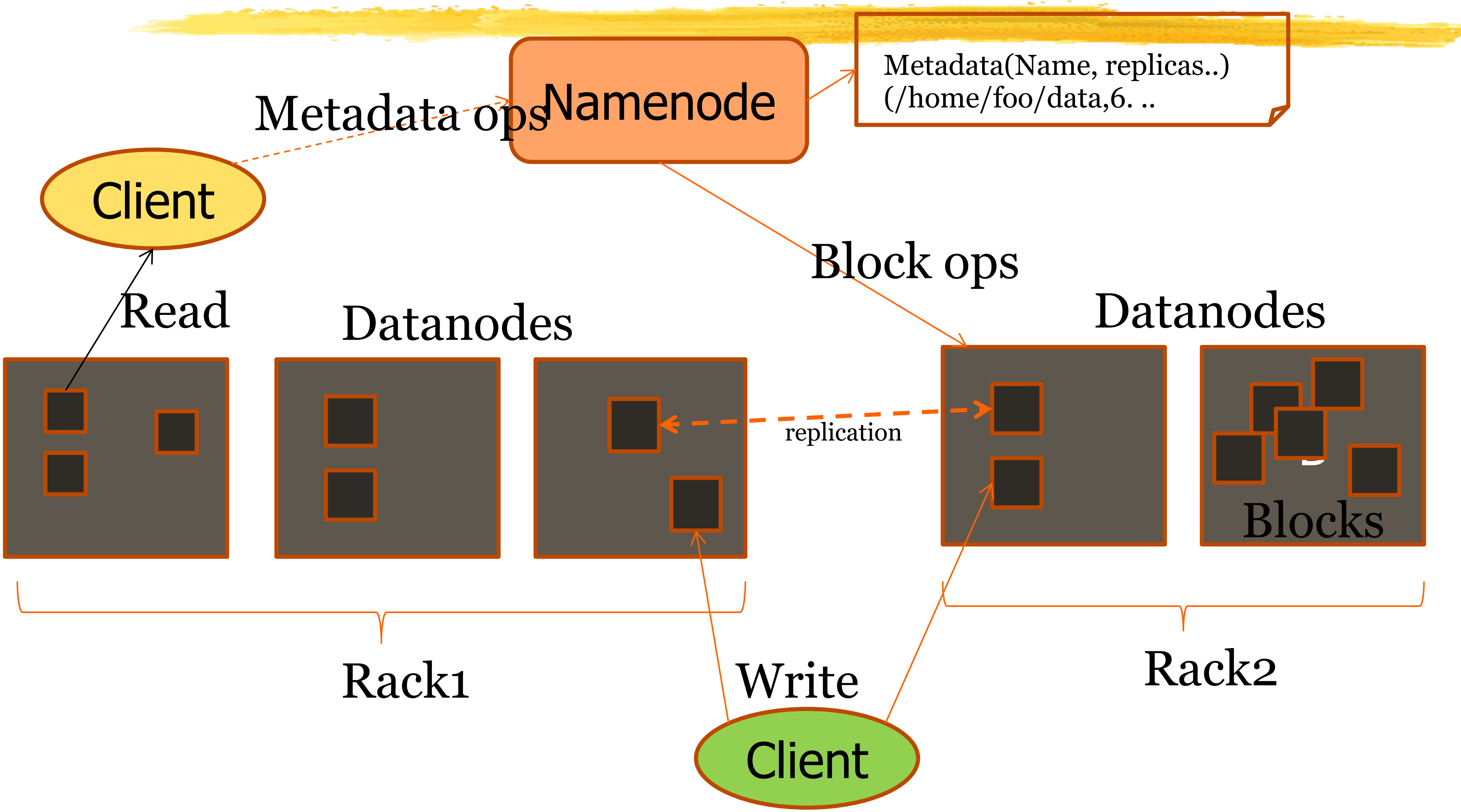
Google File System

- Internet introduced a new challenge in the form of huge amounts of web logs and web crawler's data - "peta scale" data problem
- This type of data has a uniquely different characteristic than your transactional or "customer order" type data
- Google exploited this characteristics in its Google file system (GFS)

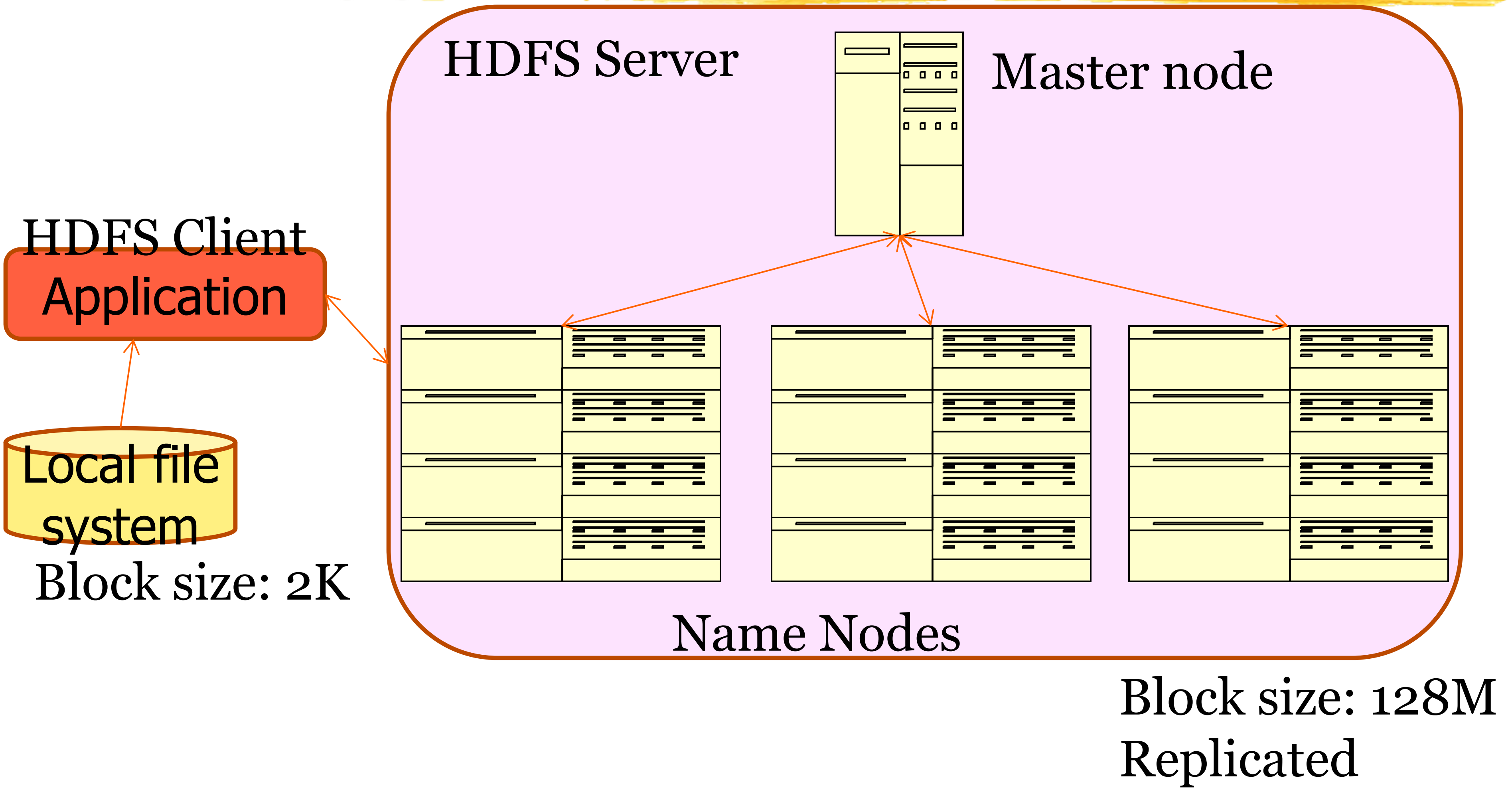
Apache Hadoop

- Google run MapReduce operations on a special file system called Google File System (GFS) that is highly optimized for this purpose but GFS is not open source
- Doug Cutting and others at Yahoo! reverse engineered the GFS and called it Hadoop Distributed File System (HDFS)
- The software framework that supports HDFS, MapReduce and other related entities is called Hadoop and is distributed by Apache

HDFS Architecture



HDFS Deployment



HDFS Fault Tolerance

- Failure is the norm rather than exception
- A HDFS instance may consist of thousands of server machines, each storing part of the file system’s data.
- Since we have huge number of components and that each component has non-trivial probability of failure means that there is always some component that is non-functional.
- Detection of faults and quick, automatic recovery from them is a core architectural goal of HDFS.

Whats is MapReduce?

- MapReduce is a programming model Google has used successfully for processing its “big-data” sets (> 20000 peta bytes per day)
  - A map function extracts some intelligence from raw data.
  - A reduce function aggregates according to some guides the data output by the map.
  - Users specify the computation in terms of a *map* and a *reduce* function,
  - Underlying runtime system automatically parallelizes the computation across large-scale clusters of machines, and
  - Underlying system also handles machine failures, efficient communications, and performance issues

Example of MapReduce

- Google uses it for wordcount, adwords, pagerank, indexing data.
- Simple algorithms such as grep, text-indexing, reverse indexing
- Bayesian classification: data mining domain
- Facebook uses it for various operations: demographics
- Financial services use it for analytics
- Astronomy: Gaussian analysis for locating extra-terrestrial objects.
- Expected to play a critical role in semantic web and in web 3.0

HDFS and MapReduce Engine

- MapReduce requires a distributed file system and an engine that can distribute, coordinate, monitor and gather the results
- Hadoop provides that engine through (the file system we discussed earlier) and the JobTracker + TaskTracker system
  - JobTracker is simply a scheduler
  - TaskTracker is assigned a Map or Reduce (or other operations); Map or Reduce run on node and so is the TaskTracker; each task is run on its own JVM on a node

MapReduce Application

- Word count application using Java that indexes some text files by word
- This is similar in functionality to what a web search engine does…
  - MapReduce is the basis for how the Google search works
  - Performs well for very large datasets

MapReduce Application

- Below are some text files and their contents, which to keep it simple are just a bunch of words:

file1.txt => "foo foo bar cat dog dog"

file2.txt => "foo house cat cat dog"

file3.txt => "foo foo foo bird"

MapReduce Application

- The final result is a map indexing each word to the files it is present, with an occurrence counter for each file:

"foo" => { "file1.txt" => 2, "file3.txt" => 3,
"file2.txt" => 1 }

"bar" => { "file1.txt" => 1 }

"cat" => { "file2.txt" => 2, "file1.txt" => 1 }

MapReduce Application

"dog" => { "file2.txt" => 1, "file1.txt" => 2 }
"house" => { "file2.txt" => 1 }
"bird" => { "file3.txt" => 1 }

- With the hash map above it becomes trivial to quickly search the files by word.

Approach One

- Done Without MapReduce
  - Count the words one by one and place the results in a hash map, which is actually a map of maps
  - The code is relatively simple but this solution would not be very scalable for a huge data set
  - Imagine indexing a massive amount of web page data using this approach…

Input and Output Data

```
Map<String, String> input = new  HashMap<String, String>();

input.put("file1.txt", "foo foo bar cat dog dog");
input.put("file2.txt", "foo house cat cat dog");
input.put("file3.txt", "foo foo foo bird");

Map<String, Map<String, Integer>> output = new
        HashMap<String, Map<String, Integer>>();
```

Code uses a while loop to iterate over the input map and then update the output map as required

Approach Two

- With MapReduce using a single thread
  - MapReduce takes the problem and breaks it down in two independent phases: the map phase and the reduce phase
  - In practice there is a third pre-reduce phase called *grouping*, but the only phases that can get distributed in the cluster are the map and the reduce phases

Map Phase

- It is important to understand that the Map phase returns a list of key/value pairs
  - In our example the key is a word and the value is the file where this word was found
- The resulting list will have duplicates
  - For example the item ["foo", "file3.txt"] appears three times in the list because the word “foo” appears in the file three times

Map Phase Output

A List<MappedItem> is built during the map phase and it would look like this if printed out:

[["foo","file2.txt"], ["house","file2.txt"], ["cat","file2.txt"], ["cat","file2.txt"], ["dog","file2.txt"], ["foo","file1.txt"], ["foo","file1.txt"], ["bar","file1.txt"], ["cat","file1.txt"], ["dog","file1.txt"], ["dog","file1.txt"], ["foo","file3.txt"], ["foo","file3.txt"], ["foo","file3.txt"], ["bird","file3.txt"]]

This is then used as the input for the Grouping phase...

Grouping Phase

- The intermediate phase of MapReduce is the grouping phase where the map results are grouped and prepared for the reduce phase.
- The output of the grouping phase is the following data structure:

  Map<String, List<String>> groupedItems =
  new HashMap<String, List<String>>();

Grouping Phase Output

- Output here is like the mapping phase but without the duplicates:

  {bird=[file3.txt], cat=[file2.txt, file2.txt, file1.txt], foo=[file2.txt, file1.txt, file1.txt, file3.txt, file3.txt, file3.txt], house=[file2.txt], bar=[file1.txt], dog=[file2.txt, file1.txt, file1.txt]}

- Final step is to reduce this data to the final format we want

Reduce Phase

- Iterate over the grouped items and call the reduce method for each entry
- This method then simply builds the final map based on the values passed
- This method can be run in parallel for each entry in the in the grouped items

Approach Three

- Using a MapReduce Cluster
  - The map and reduce work can be easily broken down in independent jobs and distributed across a cluster of machines that can perform the work in parallel
  - In this case the cluster is simulated using multiple java threads to process the map and reduce work independently and in parallel
  - When you have a large data set, the ability to use a cluster and scale horizontally becomes crucial

MapReduce Cluster

- Map and Reduce methods are modified to use callback interfaces that allow the worker threads callback to update the final shared map structure as required
- Main application thread uses the join() method to wait for the threads running each phase to complete

Callback Interfaces

```
public static interface MapCallback<E, V> {
    public void mapDone(E key, List<V> values);
}
public static interface ReduceCallback<E, K, V> {
    public void reduceDone(E e, Map<K,V> results);
}
```

Cluster Node Failure

- Each job sent to a thread has a unique identifier. For the map phase it is the file and for the reduce phase it is the word
- It would not be hard to simulate a cluster node failure by timing out a thread that is taking too long and then re-send the job to another thread
  - That's precisely what frameworks like Hadoop do

Framework Support

- Frameworks like Hadoop and MongoDB can manage the execution of a MapReduce operation in a cluster of computers with support for fault-tolerance
  - The complexity becomes hidden from the developer who only has to worry about implementing the map and reduce functions to transform the data set in any way they want to

Cloud Summary

- Cloud concepts and the cloud capabilities
- Business issues in adoption of the cloud
- Hadoop File System and using MapReduce paradigm to handle big-data sets
- This is a key technology that is sure to transform computing in business