Data management on the cloud for data at different scales

Genoveva Vargas-Solar

http://www.vargas-solar.com

French Council of Scientific Research, LIG & LAFMIA Labs

Montevideo, 23rd November – 4th December, 2015

The cloud

The cloud

Software as a service
PaaS

Platform as a service
PaaS

Infrastructure as a service
IaaS

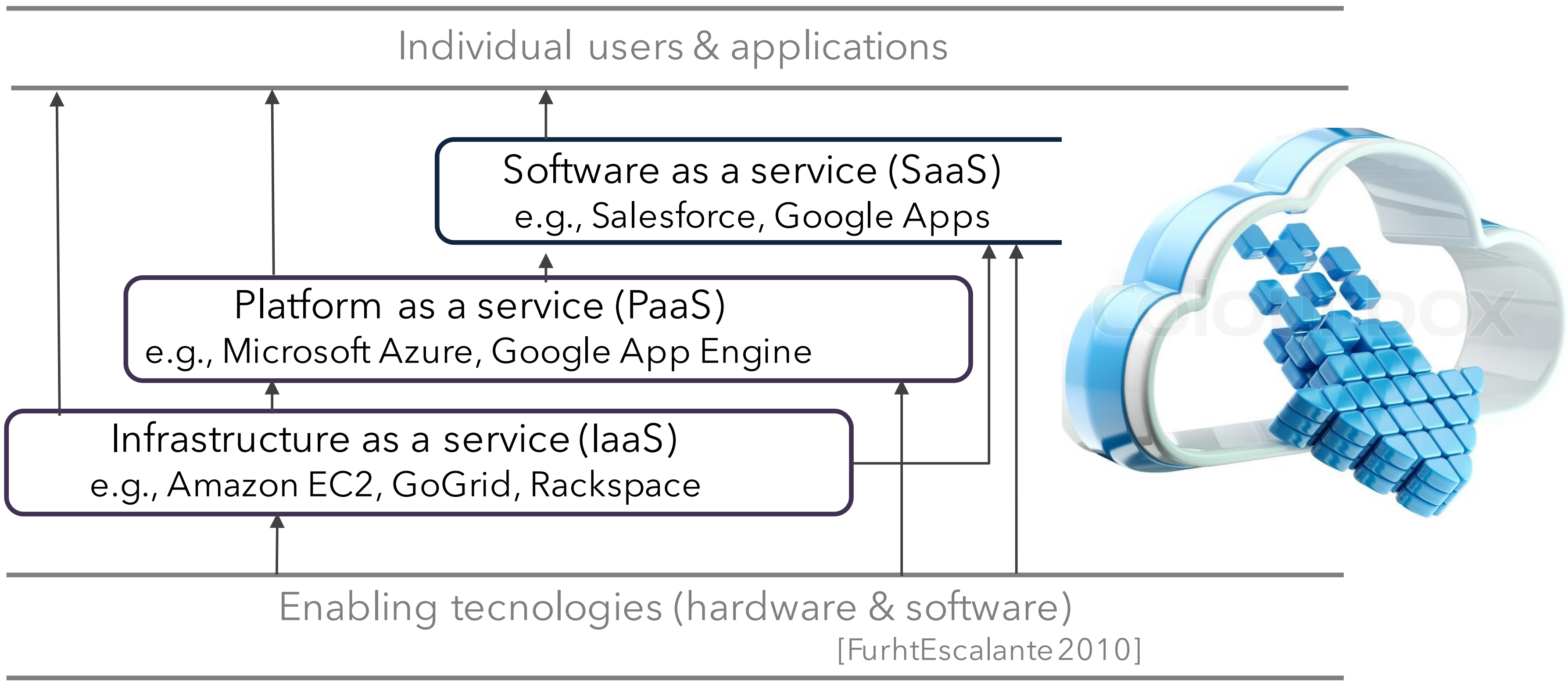
PAYG

- Illusion of infinite resources
- No up-front cost
- Fine-grained billing (e.g. hourly)

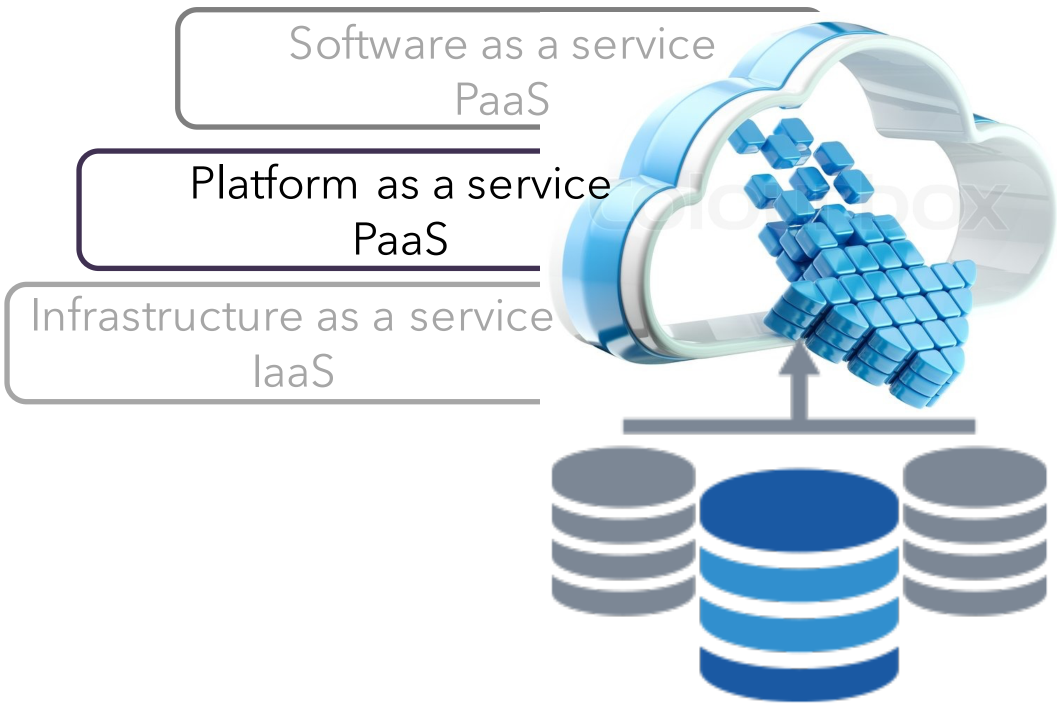
- Promotes a style of computing in which dynamically scalable and often virtualized resources are provided as a service over the Internet

- PaaS: allows customers to rent computers (virtual machines) on which to run their own computer applications.

The cloud


Individual users & applications
Software as a service (SaaS)
e.g., Salesforce, Google Apps
Platform as a service (PaaS)
e.g., Microsoft Azure, Google App Engine
Infrastructure as a service (IaaS)
e.g., Amazon EC2, GoGrid, Rackspace
Enabling tecnologies (hardware & software)
[FurhtEscalante 2010]

The cloud



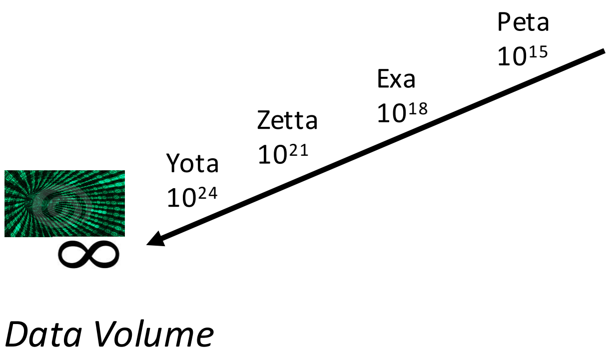
- Computing power is elastic, but noly if workload is parallelizable
  - Shared-nothing architecture
- Data is stored at un-trusted hosts
  - Solution: encrypting data
- Data is replicated, across large geographic distances
  - Availability and durability

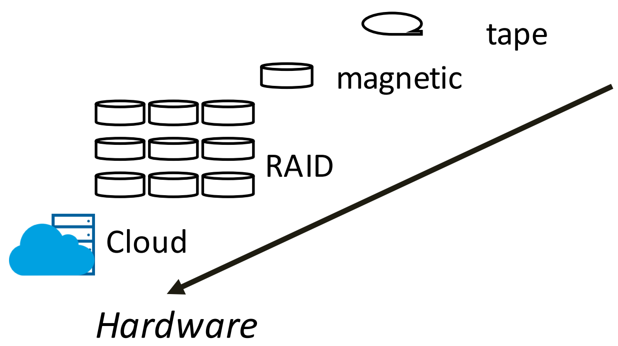
The cloud as
data management environment

Cloud data management: services views

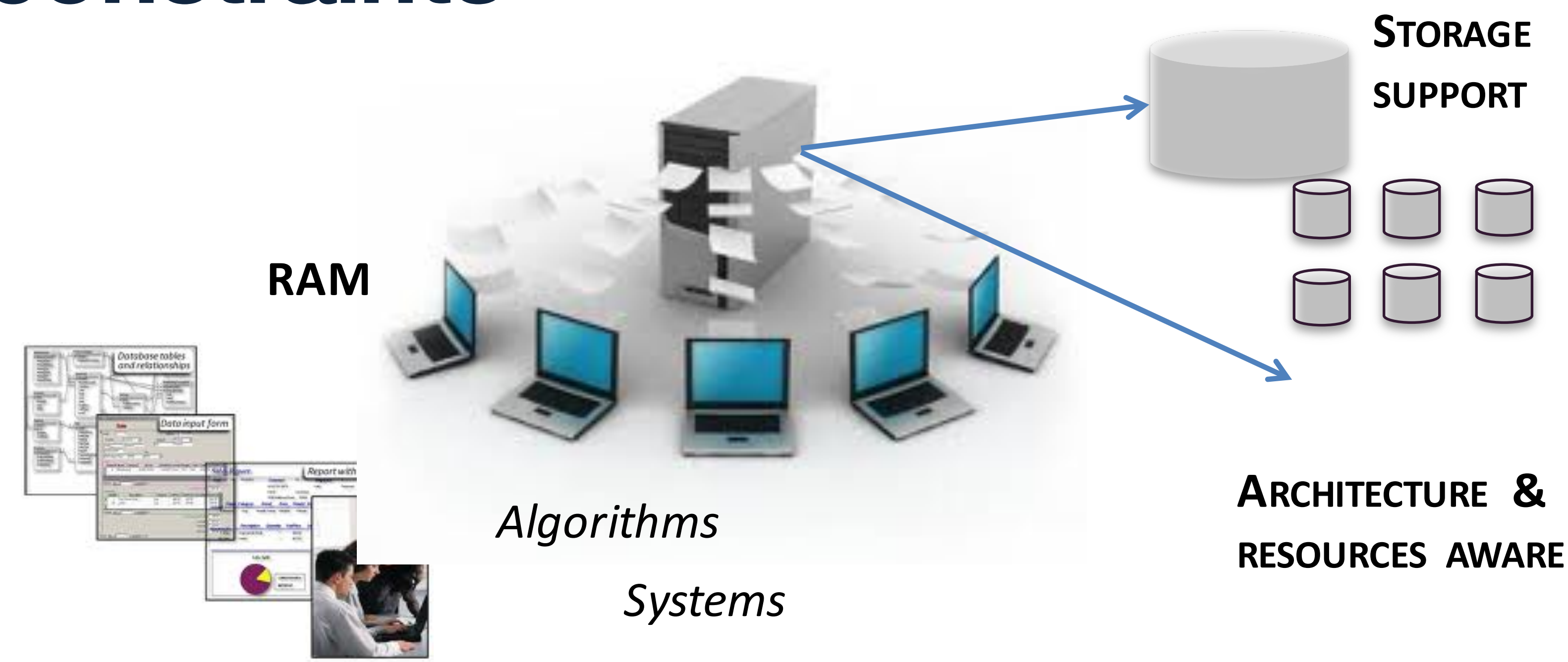
- Definition
- Querying and exploiting
- Manipulation

- Storage (persistency)
- Efficient retrieval (indexing, caching)
- Fault tolerance (recovery, replication)
- Maintenance





Data management with resources constraints

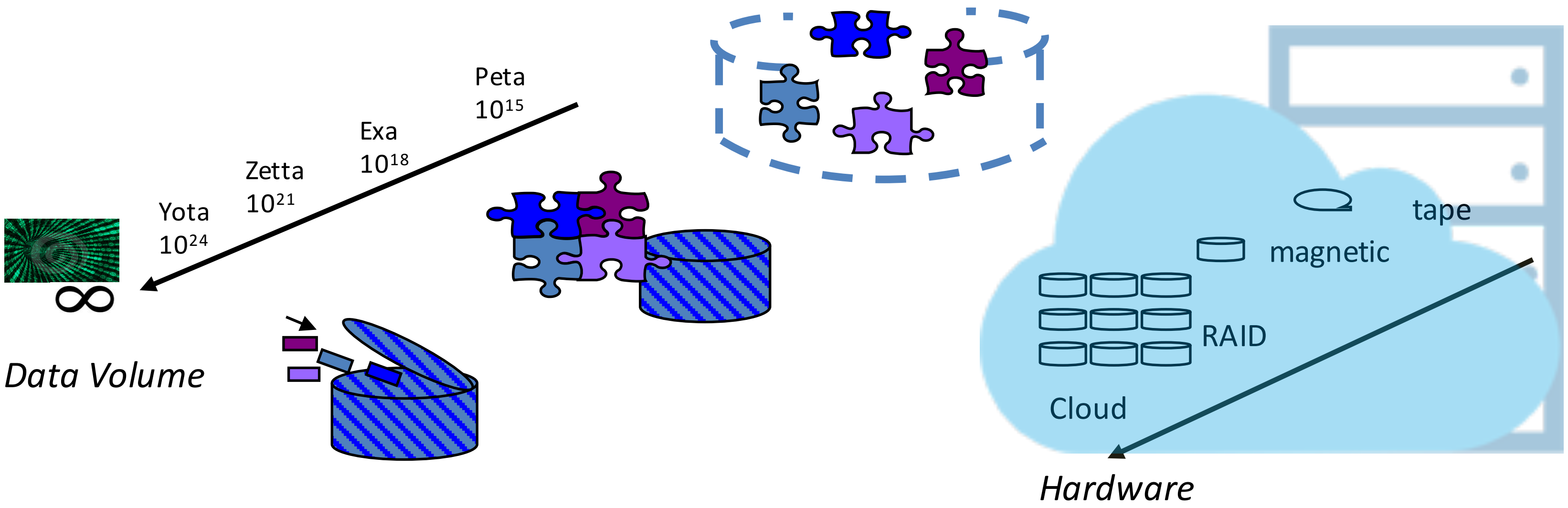


Efficiently manage and exploit data sets according to given specific storage, memory and computation resources

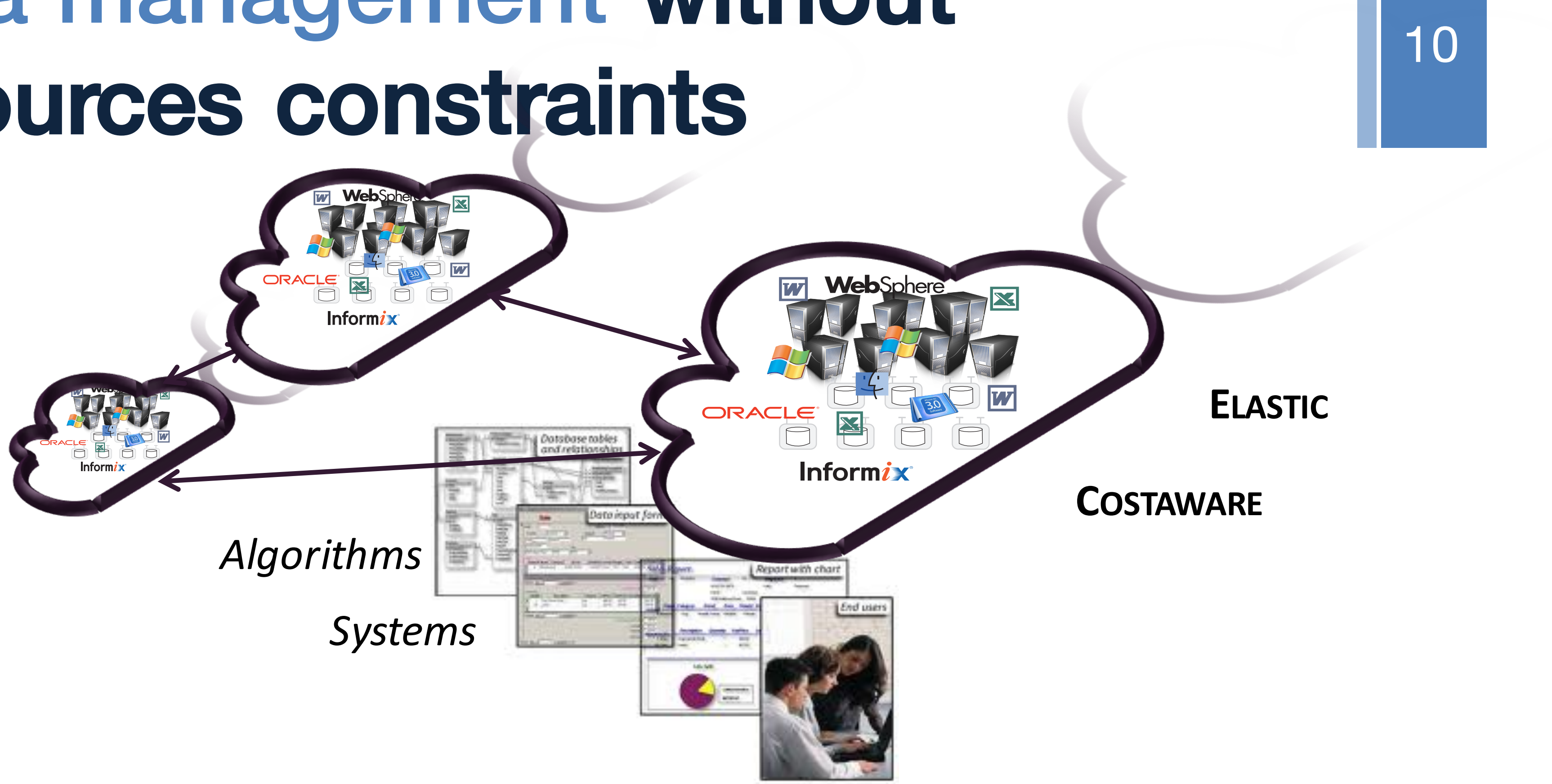
Cloud data management: services views

- Definition
- Querying and exploiting
- Manipulation

- Storage (persistency)
- Efficient retrieval (indexing, caching)
- Fault tolerance (recovery, replication)
- Maintenance



Data management **without resources constraints**



Reduce the cost to manage and exploit data sets according to unlimited storage, memory and computation resources

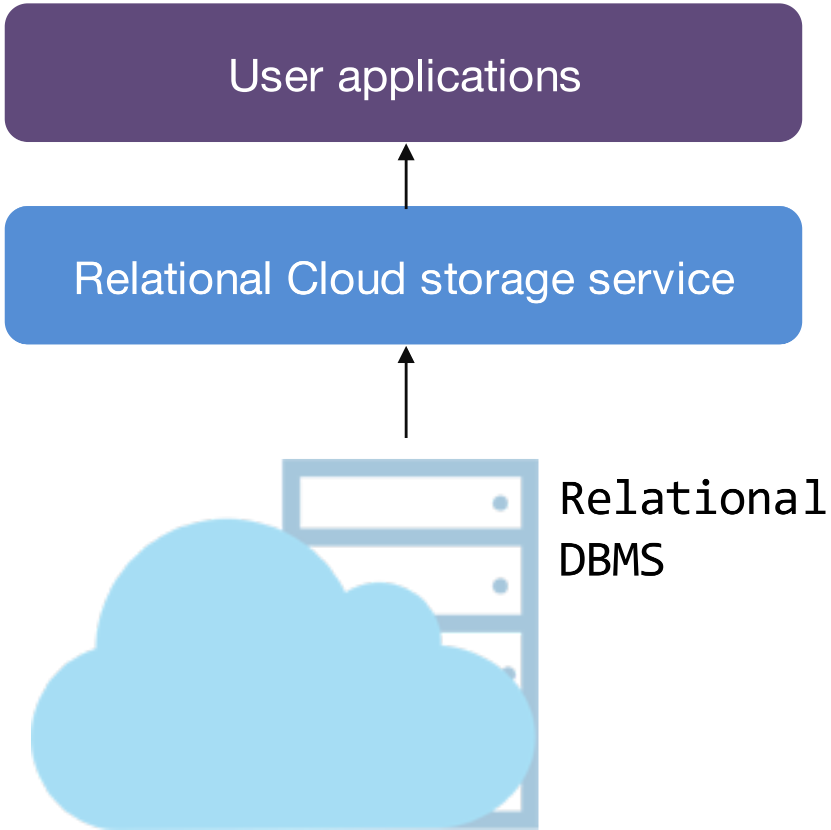
Cloud data management wish list

- **Scalability and elasticit**y are the keys in cloud data management
  - Quality: efficiency, economic cost, provenance, user preferences and constraints
  - Multi-tenancy: managing large number of small tenants
  - Consistency and replication
- Fault Tolerance
  - If a query must restart each time a node fails, then long, complex queries are difficult to complete
- Run in heterogeneous environments
  - Should prevent the slowest node from making a disproportionate affect on total query performance
- Operate on encrypted data
- Interface with **data analytics** and **exploitation services**

Cloud data management: aspects to consider

- Security [Agrawal2]
  - Confidentiality
  - Privacy
- Data Analytics
  - Large scale processing of complex queries
  - Machine learning and data mining at large scale
- Multi-tenancy
  - For OLTP [Agrawal1]
  - For OLAP [Wong 2013]
- Consistency, scalability and elasticity [Agrawal1]
  - Replication and consistency models
  - Elasticity

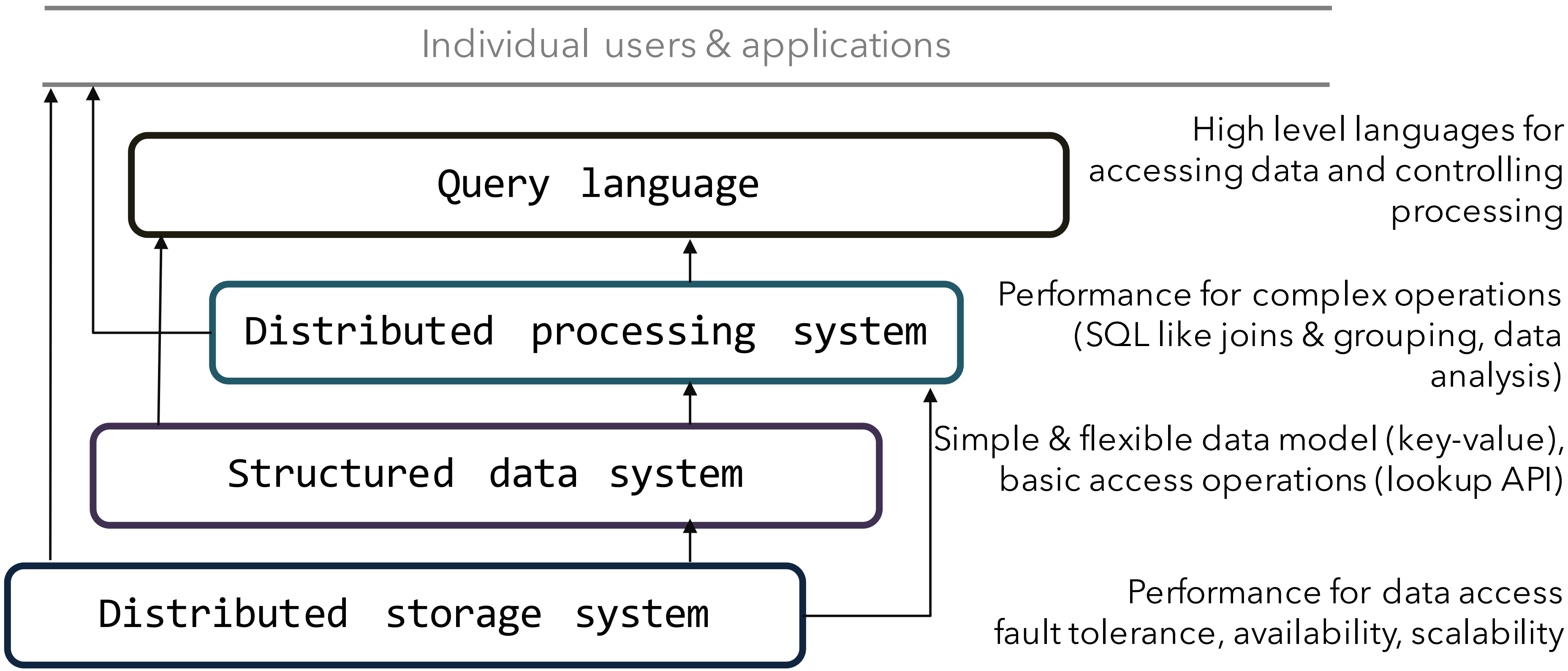
SQL as a Service



Relational model and SQL as a Service e.g. Amazon relational database service (RDS), MS SQL Azure

Implemented on top of parallel clusters of common DBMS servers e.g., MySQL MS SQL Server

Cloud data management: functions view



Cloud data management: functions view

Individual users & applications

Query language

HiveQL, JaQL, Pig on top of Hadoop Map-Reduce

Distributed processing system

Google/Hadoop MapReduce

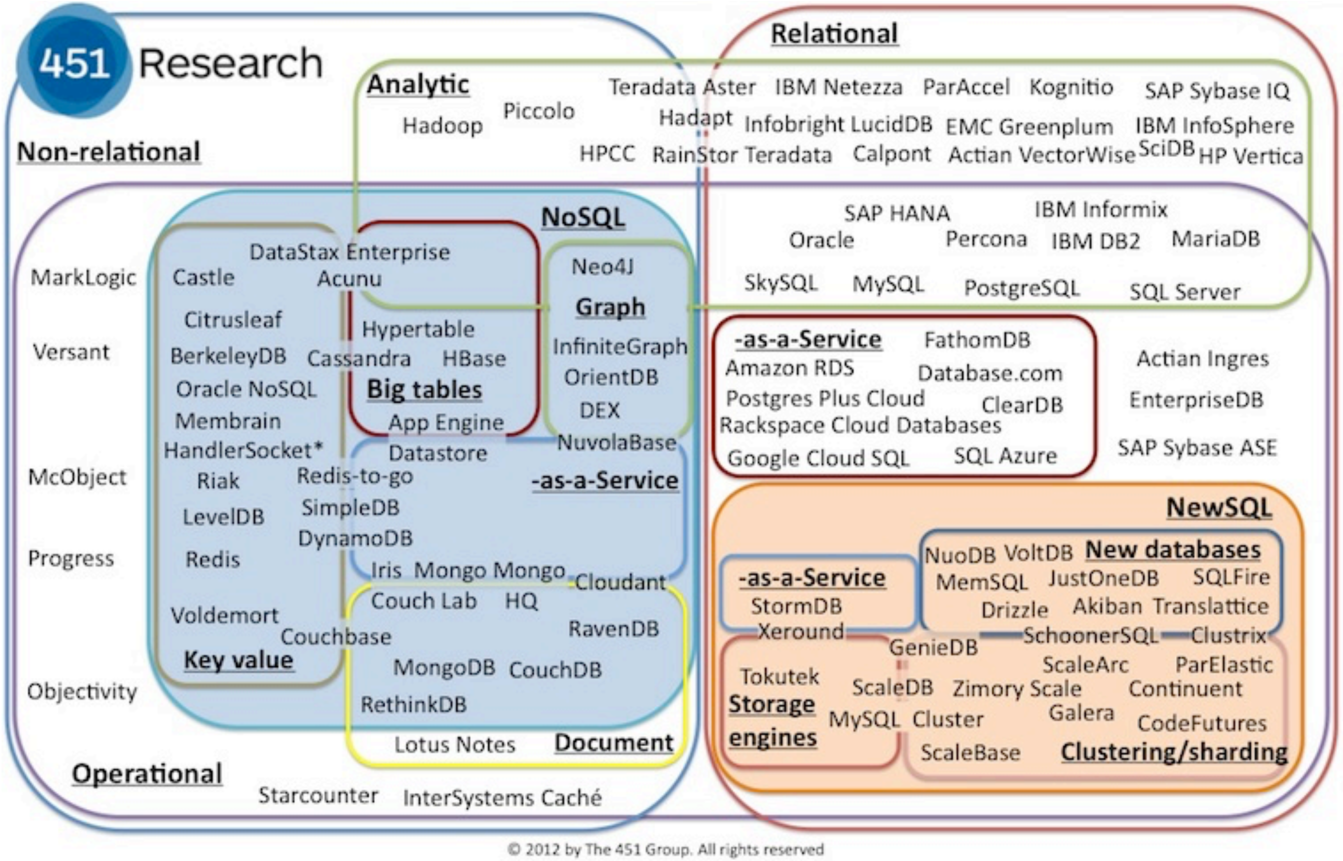
Structured data system

Google BigTable & other BigTable implementations like **Hbase**, **Cassandra**, **Amazon SimpleDB**

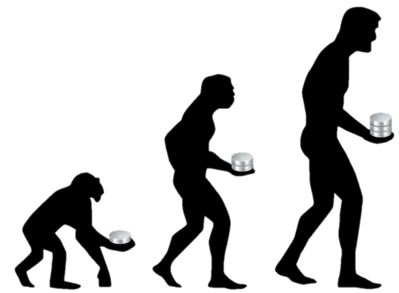
Distributed storage system

Distributed file systems:
Google file system, Hadoop Distributed File System, CloudStore
Cloud-based file Service: Amazon S3
P2P-like file service: Amazon Dynamo

Database landscape



Next generation of data management systems



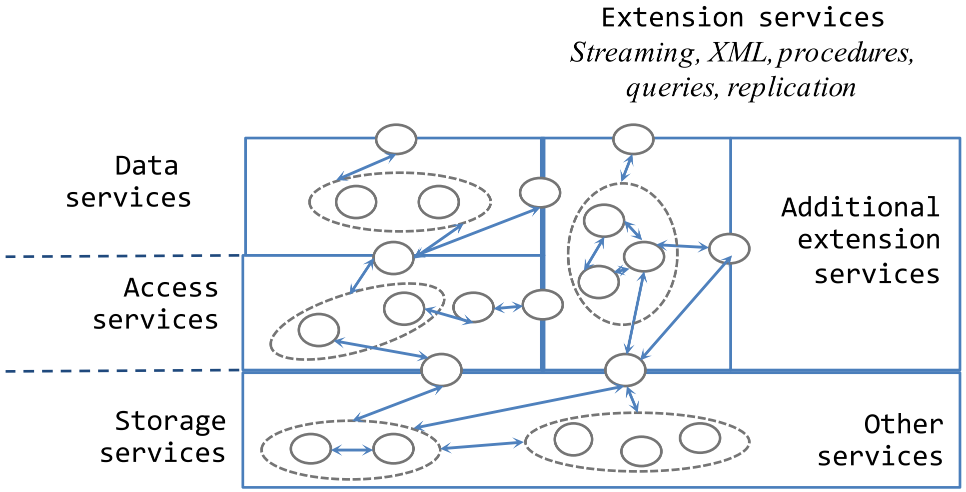
DBMS evolution

- No more monolithic DBMS
- Extensible, lightweight DBMS
- Unbundled technology*
- Component-based architectures* (thick-grain vs. fine-grain)
- OO Frameworks
- Components are providing Services
- Blur the boundaries between OS & DBMS
- Self-adaptive Systems
- Multi-tier architectures, Web, P2P, GRID, CLOUD,…

* See Dittrich, Geppert, Eds, “Component Database Systems”, MK 2000

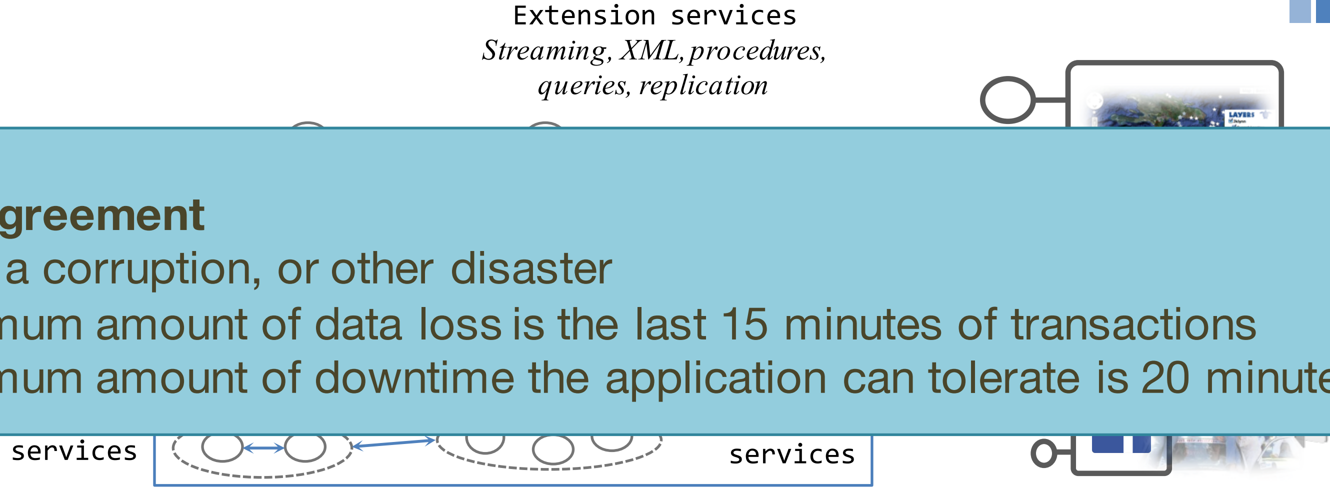
* Chaudhuri & Weikum, Rethinking Database System Architecture: Towards a Self-tuning RISC-style Database System, VLDB 2000

Service oriented DBMS[1]



[1] Ionut Subasu, Patrick Ziegler, and Klaus R Dittrich. *Towards service-based data management systems*. In Workshop Proceedings of Datenbanksysteme in Business, Technologie und Web (BTW 2007)
Klaus R Dittrich and Andreas Geppert. *Component database systems*. Morgan Kaufmann, 2000.

Service oriented DBMS[1]



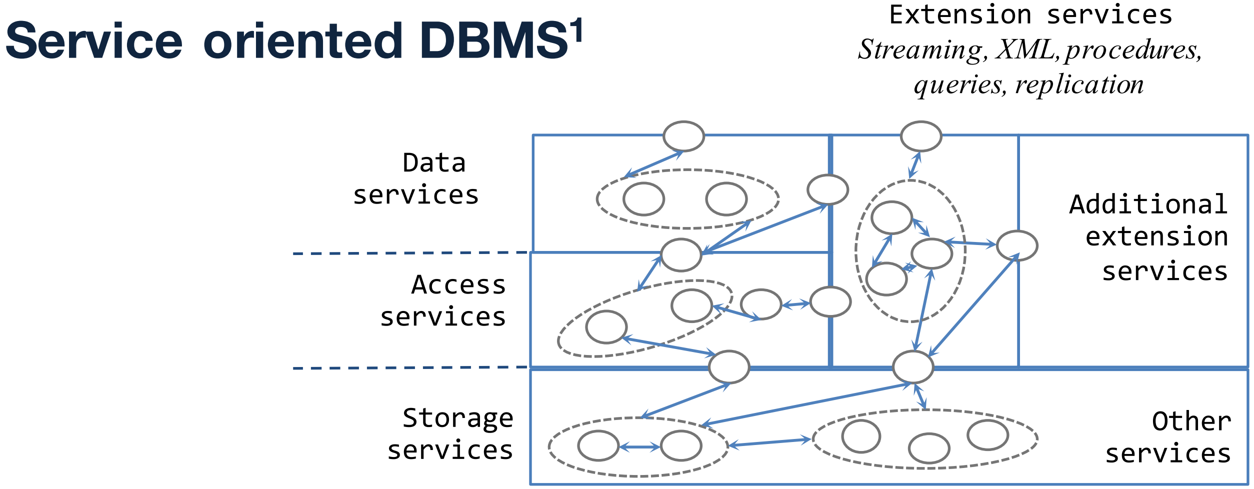
Service Level Agreement

- In the event of a corruption, or other disaster
  - the maximum amount of data loss is the last 15 minutes of transactions
  - the maximum amount of downtime the application can tolerate is 20 minutes

- *Service level agreement:* the contracted delivery time of the service or performance
- *Required SLA*: agreements between the user and SDBMS expressed as a combination of weighted measures associated to a query

[1] Ionut Subasu, Patrick Ziegler, and Klaus R Dittrich. *Towards service-based data management systems.* In Workshop Proceedings of Datenbanksysteme in Business, Technologie und Web (BTW 2007)
Klaus R Dittrich and Andreas Geppert. *Component database systems*. Morgan Kaufmann, 2000.

Service oriented DBMS[1]



- *Service level agreement:* the contracted delivery time of the service or performance
- *Required SLA*: agreements between the user and SDBMS expressed as a combination of weighted measures associated to a query

[1] Ionut Subasu, Patrick Ziegler, and Klaus R Dittrich. *Towards service-based data management systems*. In Workshop Proceedings of Datenbanksysteme in Business, Technologie und Web (BTW 2007)
Klaus R Dittrich and Andreas Geppert. *Component database systems*. Morgan Kaufmann, 2000.

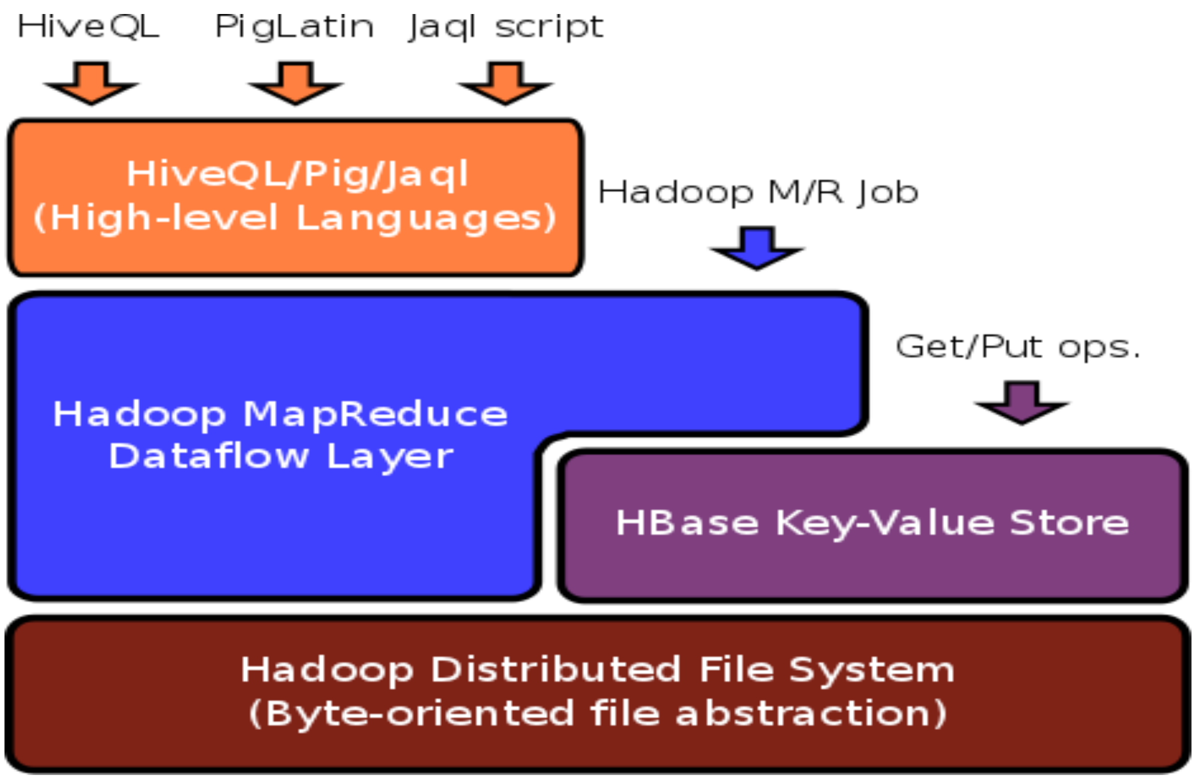
Challenges and objective

- How to combine, deploy, and deliver DBMS functionalities:
  - **Compliant** to application/user requirements
  - **Optimizing** the consumption of computing resources in the presence of **greedy** data processing tasks
  - Delivered according to **Service Level Agreement (SLA)** contracts
  - Deployed in **elastic** and distributed **platforms**

* See Dittrich, Geppert, Eds, "Component Database Systems", MK 2000

* Chaudhuri & Weikum, Rethinking Database System Architecture: Towards a Self-tuning RISC-style Database System, VLDB 2000

Open Source Big Data Stacks

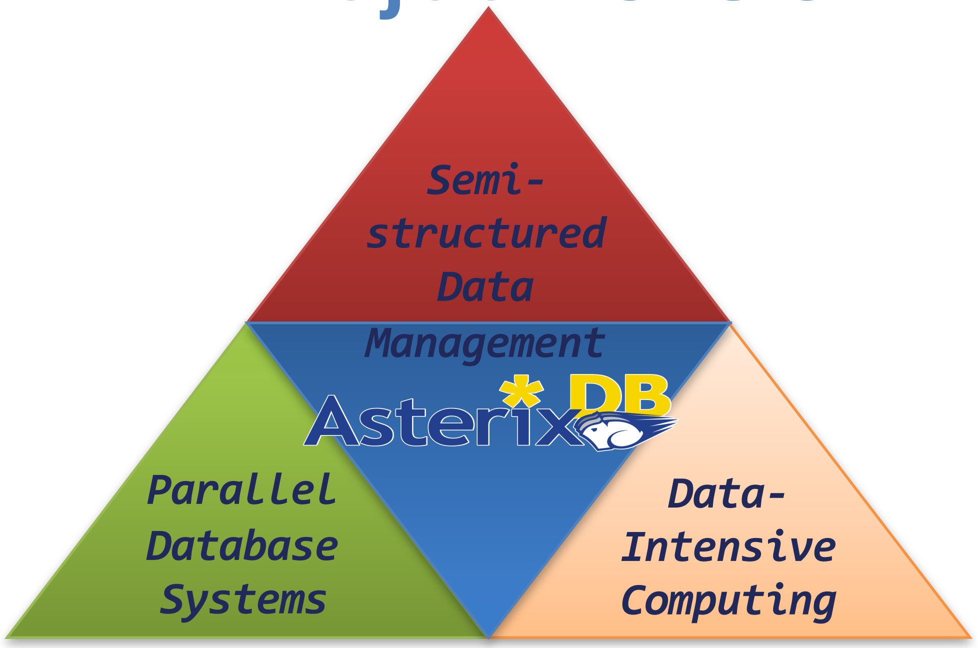


Notes:

- Giant byte sequence at the bottom
- Map, sort, shuffle, reduce layer in middle
- Possible storage layer in middle as well
- HLLs now at the top

From Mike Carey

ASTERIXDB Project @ UCI



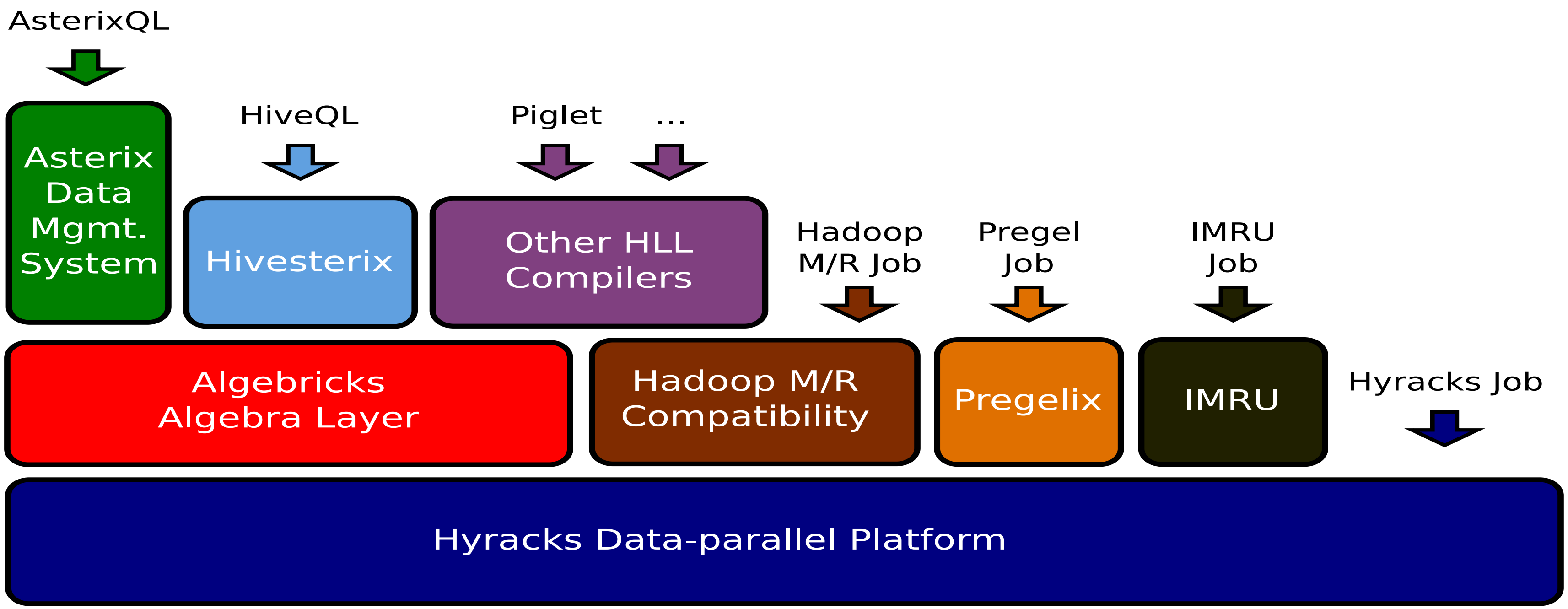
UCIrvine
http://isg.ics.uci.edu

"One Size Fits a Bunch"

http://asterixdb.ics.uci.edu

- Inside "Big Data Management": Ogres, Onions, or Parfaits?, Vinayak Borkar, Michael J. Carey, Chen Li, EDBT/ICDT 2012 Joint Conference Berlin
- Data Services, Michael J. Carey, Nicola Onose, Michalis Petropoulos
CACM June 2012, (Vol55, N.6)

The ASTERIX Software Stack



Google BigQuery

Key Differences	BigQuery	MapReduce
What is it?	Query service for large datasets	Programming model for processing large datasets
Common use cases	Ad hoc and trial-and- error interactive query of large dataset for quick analysis and troubleshooting	Batch processing of large dataset for time-consuming data conversion or aggregation
Sample use cases		
OLAP/BI use case	Yes	No
Data Mining use case	Partially (e.g. preflight data analysis for data mining)	Yes
Very fast response	Yes	No (takes minutes - days)
Easy to use for non-programmers (analysts, tech support, etc)	Yes	No (requires Hive/Tenzing)
Programming complex data processing logic	No	Yes
Processing unstructured data	Partially (regular expression matching on text)	Yes

Google BigQuery Pricing

BigQuery uses a columnar data structure, which means that for a given query, you are only charged for data processed in each column, not the entire table. The first 100GB of data processed per month is at no charge

Pricing Table

Resource	Pricing	Default Limits
Storage	$0.12 (per GB/month)	2TB
Interactive Queries	$0.035 (per GB Processed) **	20,000 Queries Per Day (QPD) 20TB of Data Processed Per Day
Batch Queries	$0.02 (per GB processed)	20,000 Total Queries Per Day (QPD)

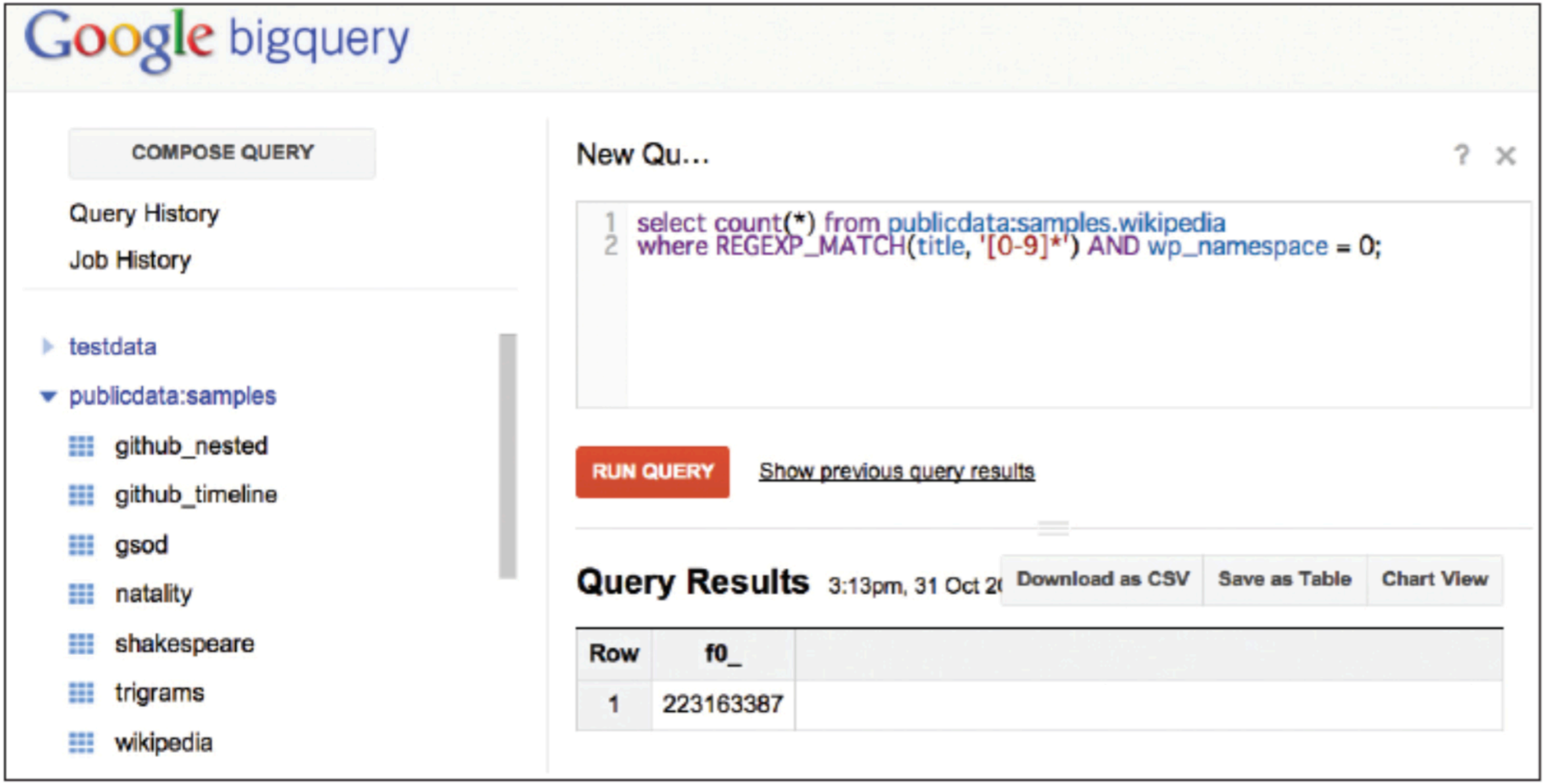
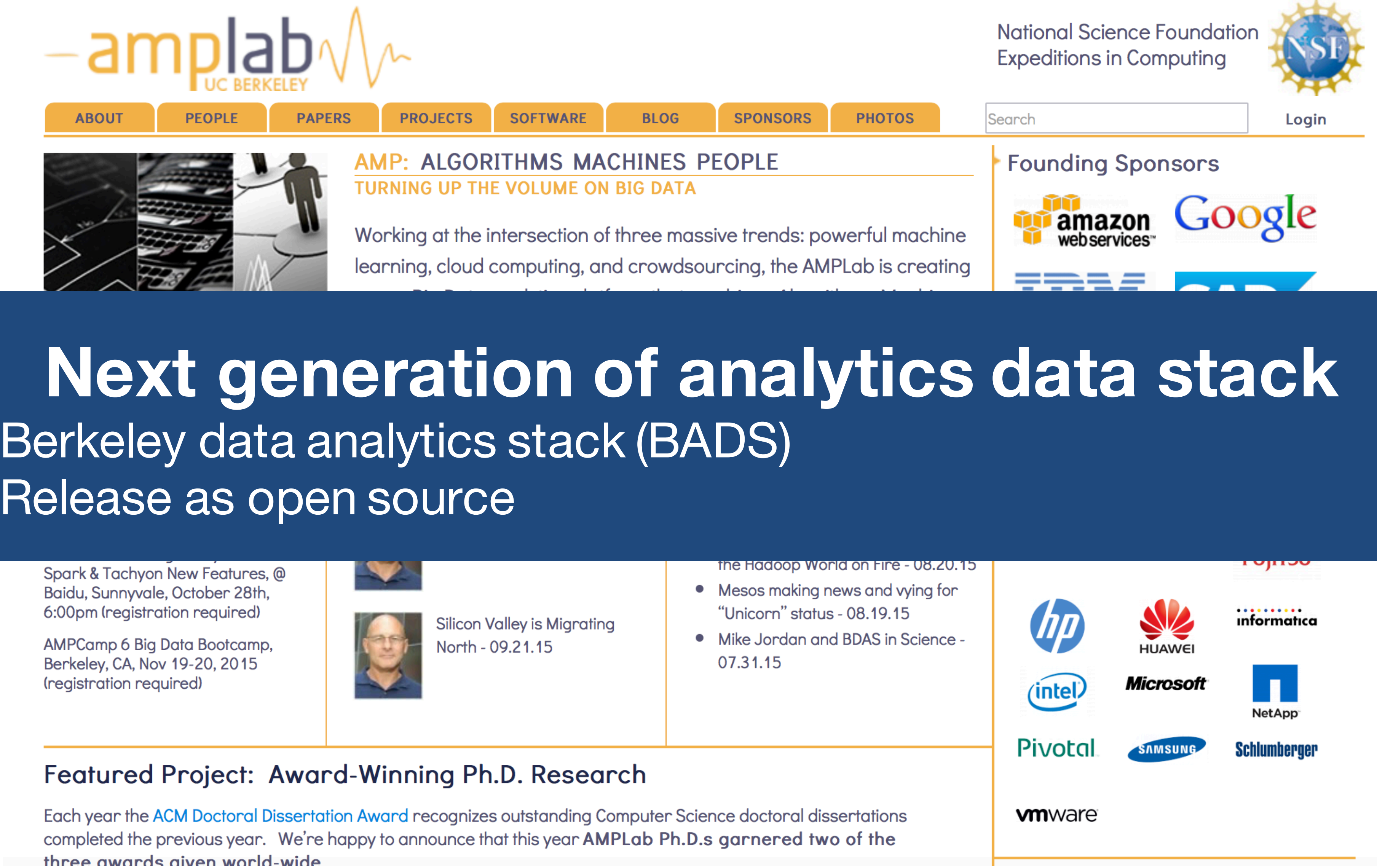


Figure 1 Querying Sample Wikipedia Table on BigQuery

(You can try out BigQuery by simply sign up for it.)



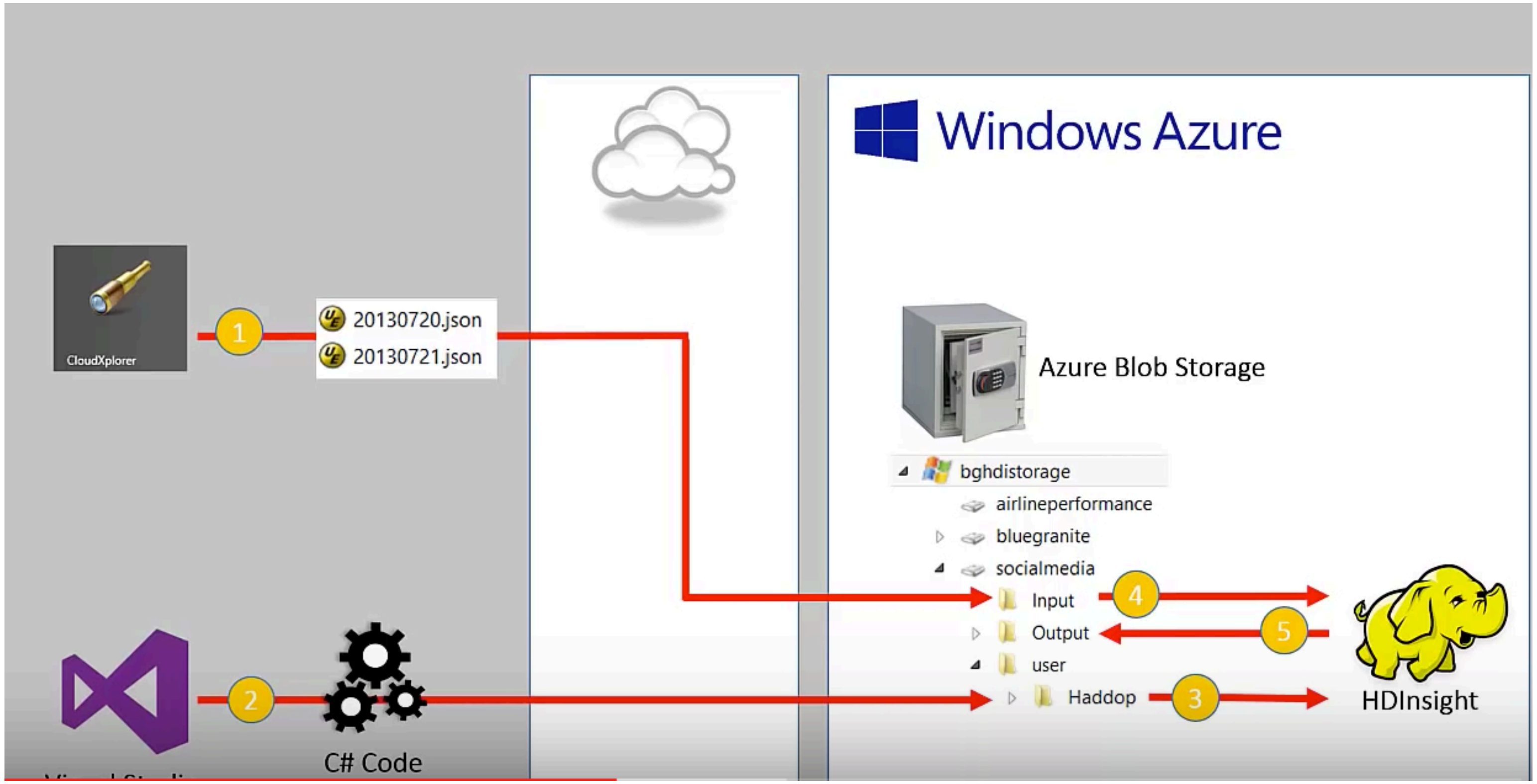
Next generation of analytics data stack

- Berkeley data analytics stack (BADS)
- Release as open source

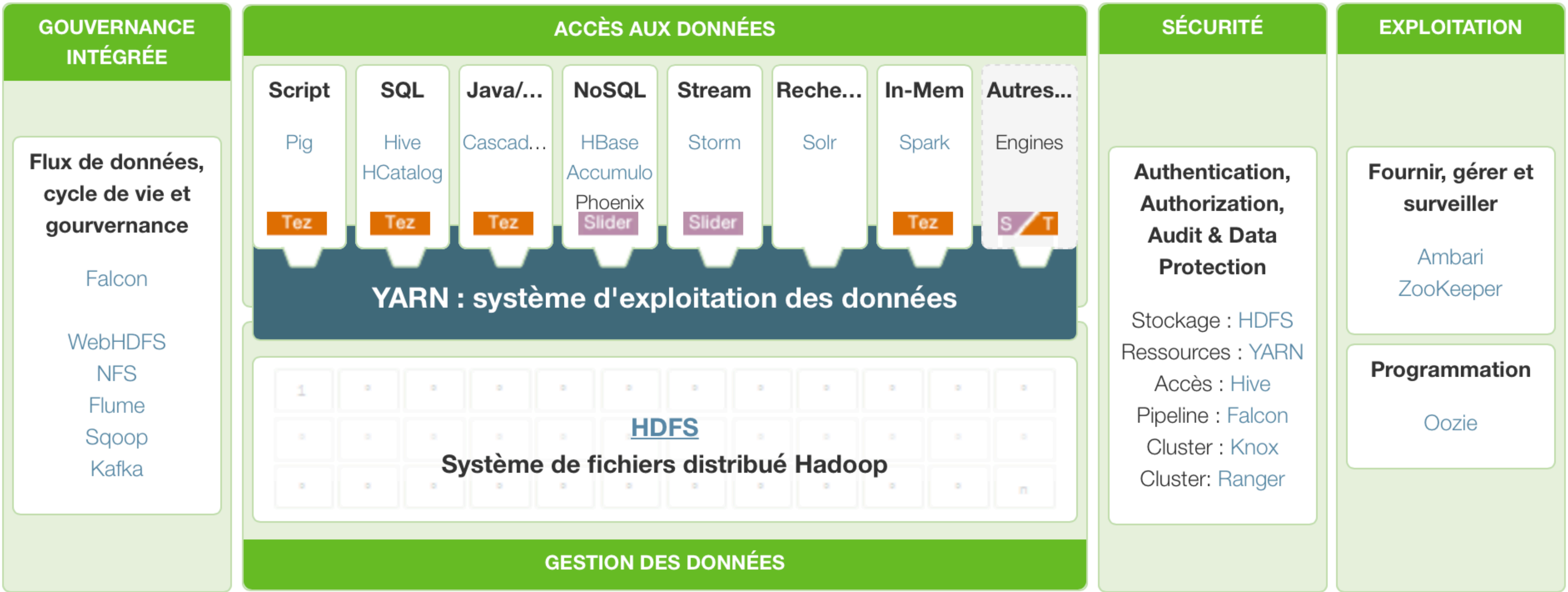

INSTITUT Mines-Télécom
GROUPE-GENES
TERALAB
Le projet
La communauté
Calendrier
Contact
EN
FR
ABOUT
Institut
Mines-Télécom
Grand établissement public à caractère scientifique, culturel et professionnel. L'institut regroupe un réseau de 13 écoles parmi les plus grandes de France.
ABOUT
Le GENES
Le groupe des Écoles Nationales d'Économie et Statistique est un établissement public d'enseignement supérieur et de recherche rattaché au ministère de l'économie et des finances, et dont l'INSEE assure ainsi la tutelle technique.
BIG DATA
Ambition TeraLab
TeraLab est un « Projet d'Investissement d'Avenir » (PIA) lauréat de l'appel à projet Big Data de 2012.

Where is the cloud?

Map Reduce on Azure



Hortonworks



http://fr.hortonworks.com

Conclusions & Perspectives

Conclusions

- **Data collections**
  - **New scales:** bronto scale due to emerging IoT
  - **New types**: thick, long hot, cold
  - New **quality measures**: QoS, QoE, SLA
- Data **processing** & **analytics**
  - Complex jobs, stream analytics are still open issues
  - Economic cost model & business models (Big Data value & pay-as-U-go)
- **Multi-cloud:** elasticity, quality, SLA

Genoveva Vargas-Solar

http://www.vargas-solar.com

French Council of Scientific Research, LIG & LAFMIA Labs

Distributed file system

- Abandons the separation of computation and storage as distinct components in a cluster
  - Google File System (GFS) supports Google's proprietary implementation of MapReduce;
  - In the open-source world, HDFS (Hadoop Distributed File System) is an open-source implementation of GFS that supports Hadoop

- The main idea is to divide user data into blocks and replicate those blocks across the local disks of nodes in the cluster

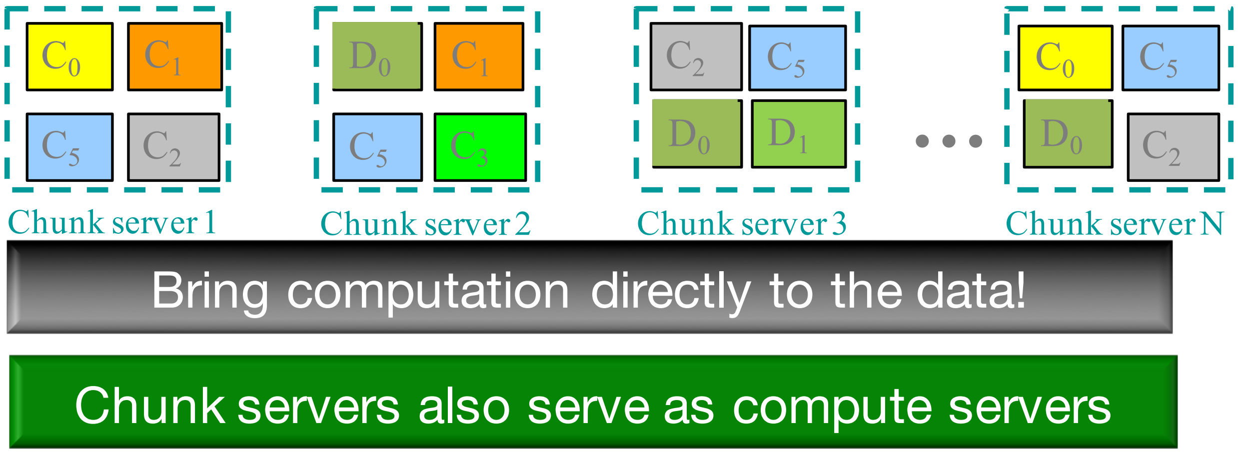
- Adopts a master–slave architecture
  - Master (`namenode` HDFS) maintains the file namespace (metadata, directory structure, file to block mapping, location of blocks, and access permissions)
  - Slaves (`datanode` HDFS) manage the actual data blocks

Distributed File System

- **Chunk servers**
  - File is split into contiguous chunks
  - Typically each chunk is 16-64MB
  - Each chunk replicated (usually 2x or 3x)
  - Try to keep replicas in different racks
- **Master node**
  - a.k.a. Name Node in Hadoop's HDFS
  - Stores metadata about where files are stored
  - Might be replicated
- **Client library for file access**
  - Talks to master to find chunk servers
  - Connects directly to chunk servers to access data

Distributed File System

- **Reliable distributed file system**
- Data kept in "chunks" spread across machines
- Each chunk replicated on different machines
  - Seamless recovery from disk or machine failure



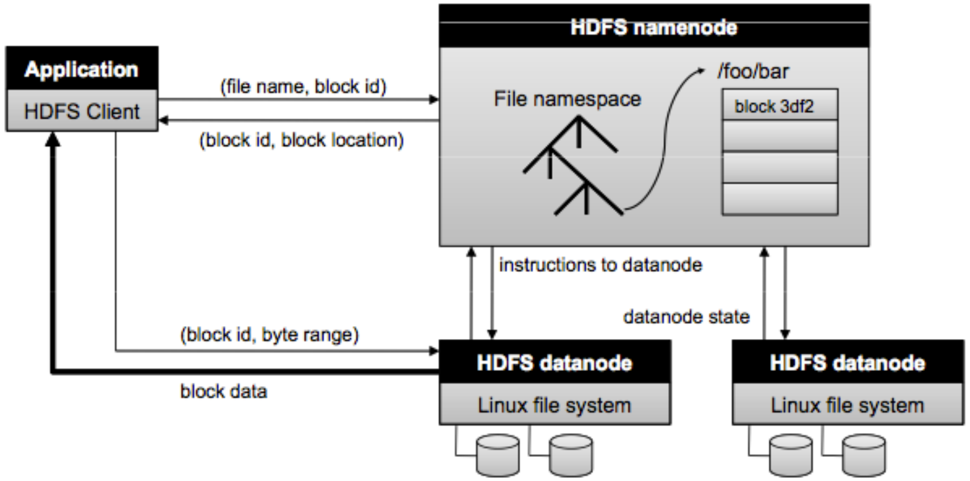
HFDS general architecture

- An application client wishing to read a file (or a portion thereof) must first contact the `namenode` to determine where the actual data is stored

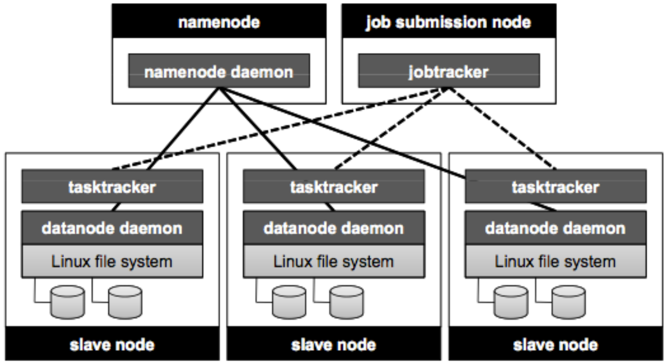
- The `namenode` returns the relevant `block id` and the `location` where the block is held (i.e., which `datanode`)

- The client then contacts the `datanode` to retrieve the data.

- HDFS lies on top of the standard OS stack (e.g., Linux): blocks are stored on standard single-machine file systems



Hadoop cluster architecture



- The HDFS **`namenode`** runs the `namenode daemon`
- The job submission node runs the **`jobtracker`**, which is the single point of contact for a client wishing to execute a MapReduce job
- The **`jobtracker`**
  - Monitors the progress of running MapReduce jobs
  - Is responsible for coordinating the execution of the `mappers` and `reducers`
  - Tries to take advantage of data locality in scheduling map tasks

Hadoop cluster architecture

- `Tasktracker`
  - It accepts tasks (Map, Reduce, Shuffle, etc.) from `JobTracker`
  - Each `TaskTracker` has a number of slots for the tasks: these are execution slots available on the machine or machines on the same rack
  - It spawns a separate JVM for execution of the tasks
  - It indicates the number of available slots through the hearbeat message to the `JobTracker`

HDFS properties

- HDFS stores **three separate** copies of each data block to ensure both reliability, availability, and performance
- In large clusters, the three replicas are spread across different physical racks,
  - HDFS is resilient towards two common failure scenarios individual `datanode` crashes and failures in networking equipment that bring an entire rack offline.
  - Replicating blocks across physical machines also increases opportunities to **co-locate data** and processing in the scheduling of MapReduce jobs, since multiple copies yield more opportunities to exploit locality
- To create a new file and write data to HDFS
  - The application client contacts the `namenode`
  - The `namenode`
    - updates the file namespace after checking permissions and making sure the file doesn't already exist
    - allocates a new block on a suitable datanode
  - The application is directed to stream data directly to it
  - From the initial `datanode`, data is further propagated to additional replicas

NoSQL stores characteristics

- **Simple operations**
  - Key lookups reads and writes of one record or a small number of records
  - No complex queries or joins
  - Ability to dynamically add new attributes to data records
- **Horizontal scalability**
  - Distribute data and operations over many servers
  - Replicate and distribute data over many servers
  - No shared memory or disk
- **High performance**
  - Efficient use of distributed indexes and RAM for data storage
  - Weak consistency model
  - Limited transactions

Next generation databases mostly addressing some of the points: being **non-relational, distributed, open-source** and **horizontally scalable** [`http://nosql-database.org`]

so now we have NoSQL databases

- Data model
- Consistency
- Storage
- Durability
- Availability
- Query support

Data stores designed to scale simple

OLTP-style application loads

Read/Write operations
by **thousands/millions** of users

examples include

mongoDB, CouchDB, riak, Cassandra, Neo4j, Apache HBase, redis

We should also remember Google's Bigtable and Amazon's SimpleDB. While these are tied to their host's cloud service, they certainly fit the general operating characteristics

Important design goals

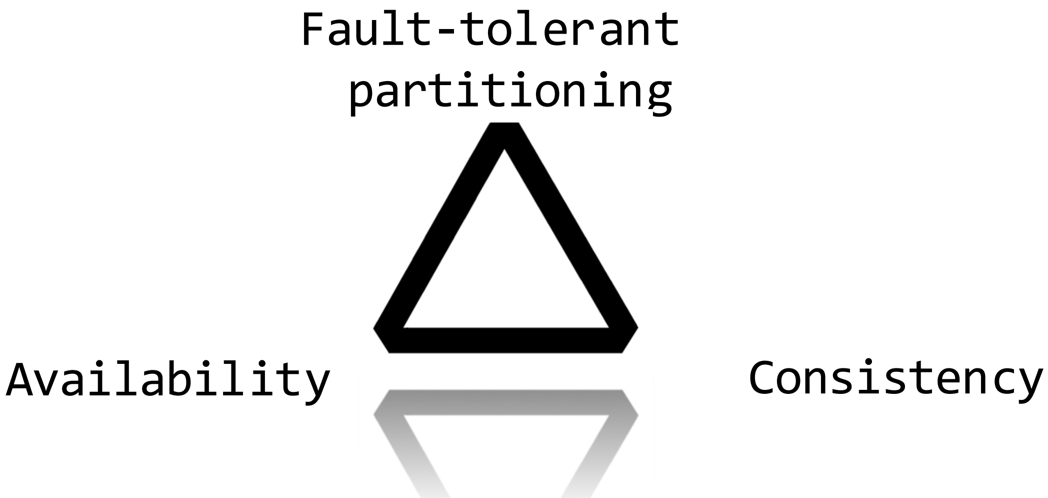
- Scale out: designed for scale
  - Commodity hardware
  - Low latency updates
  - Sustain high update/insert throughput
- Elasticity – scale up and down with load
- High availability – downtime implies lost revenue
  - Replication (with multi-mastering)
  - Geographic replication
  - Automated failure recovery

Lower priorities

- No Complex querying functionality
  - No support for SQL
  - CRUD operations through database specific API
- No support for joins
  - Materialize simple join results in the relevant row
  - Give up normalization of data?
- No support for transactions
  - Most data stores support single row transactions
  - Tunable consistency and availability (e.g., Dynamo)

→ **Achieve high scalability**

Non functional properties



- CAP theorem[1]: a system can have two of the three properties
- NoSQL systems sacrifice **consistency**

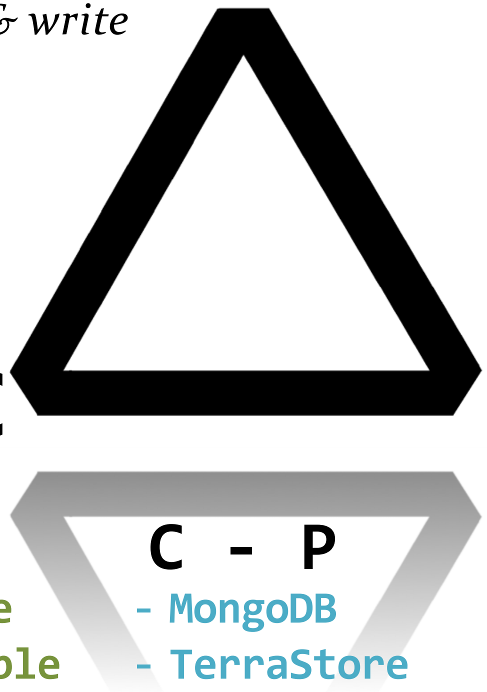
1 Eric Brewer, "Towards robust distributed systems." PODC. 2000 http://www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf

Visual guide to NoSQL systems

Data models
- Relational
- Key-Value
- Column oriented Tabular
- Document oriented

Availability:
each client can always read & write

A



C

Consistency:
all clients always have the same view of de data

P

Partition tolerance:
The system works well despite physical network partitions

C - A

- RDBM's
- MySQL
- Postgres
- etc
- Aster Data
- GreenPlum
- Vertica

A - P

- Dynamo
- Voldemort
- Tokyo Cabinet
- KAI
- Cassandra
- SimpleDB
- CouchDB
- Riak

C - P

- BigTable
- HyperTable
- Hbase
- MongoDB
- TerraStore
- Scalaris
- BerkeleyDB
- MemcacheDB
- Redis

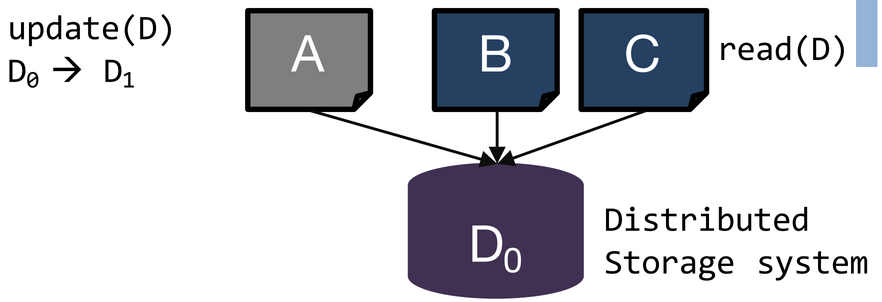
Why sacrifice consistency?

- It is a simple solution
  - nobody understands what sacrificing P means
  - sacrificing A is unacceptable in the Web
  - possible to push the problem to app developer
- C not needed in many applications
  - Banks do not implement ACID (classic example wrong)
  - Airline reservation only transacts reads (Huh?)
  - MySQL et al. ship by default in lower isolation level
- Data is noisy and inconsistent anyway
  - making it, say, 1% worse does not matter

Consistency model

- ACID semantics (transaction semantics in RDBMS)
  - **Atomicity**: either the operation (e.g., write) is performed on all replicas or is not performed on any of them
  - **Consistency**: after each operation all replicas reach the same state
  - **Isolation**: no operation (e.g., read) can see the data from another operation (e.g., write) in an intermediate state
  - **Durability**: once a write has been successful, that write will persist indefinitely
- BASE semantics (modern Internet systems)
  - **Basically Available**
  - **Soft-state** (or scalable)
  - **Eventually** consistent

Consistency models



- **Strong consistency**:
  - After the update completes, every subsequent access from A, B, C will return D_1
- **Weak consistency**:
  - Does not guaranty that any subsequent accesses return D_1 -> a number of conditions need to be met before D_1 is returned
- **Eventual consistency**: Special form of weak consistency
  - Guaranty that if no new updates are made, eventually all accesses will return D_1

Variations of eventual consistency

- Causal consistency:
  - If A notifies B about the update, B will read D1 (but not C!)
- Read your writes:
  - A will always read D1 after its own update
- Sessionconsistency:
  - Read your writes inside a session
- Monotonic reads:
  - If a process has seen Dk, any subsequent access will never return any Di with i < k
- Monotonic writes:
  - Guaranty to seiralize the writes of the same process

ACID vs BASE

ACID	BASE
Strong consistency for transactions highest priority	Availability and scaling highest priorities
Availability less important	Weak consistency
Pessimistic	Optimistic
Rigorous analysis	Best effort
Complex mechanisms	Simple and fast

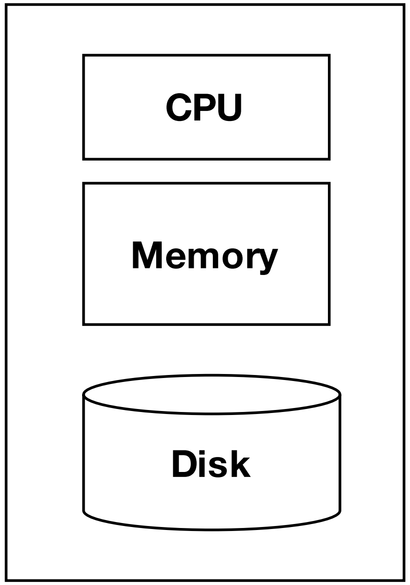
Map reduce

The new software stack

Map Reduce

- Much of the course will be devoted to **large scale computing** for **data mining**
- **Challenges:**
  - How to distribute computation?
  - Distributed/parallel programming is hard
- **Map-reduce** addresses all of the above
  - Google's computational/data manipulation model
  - Elegant way to work with big data

Single Node Architecture



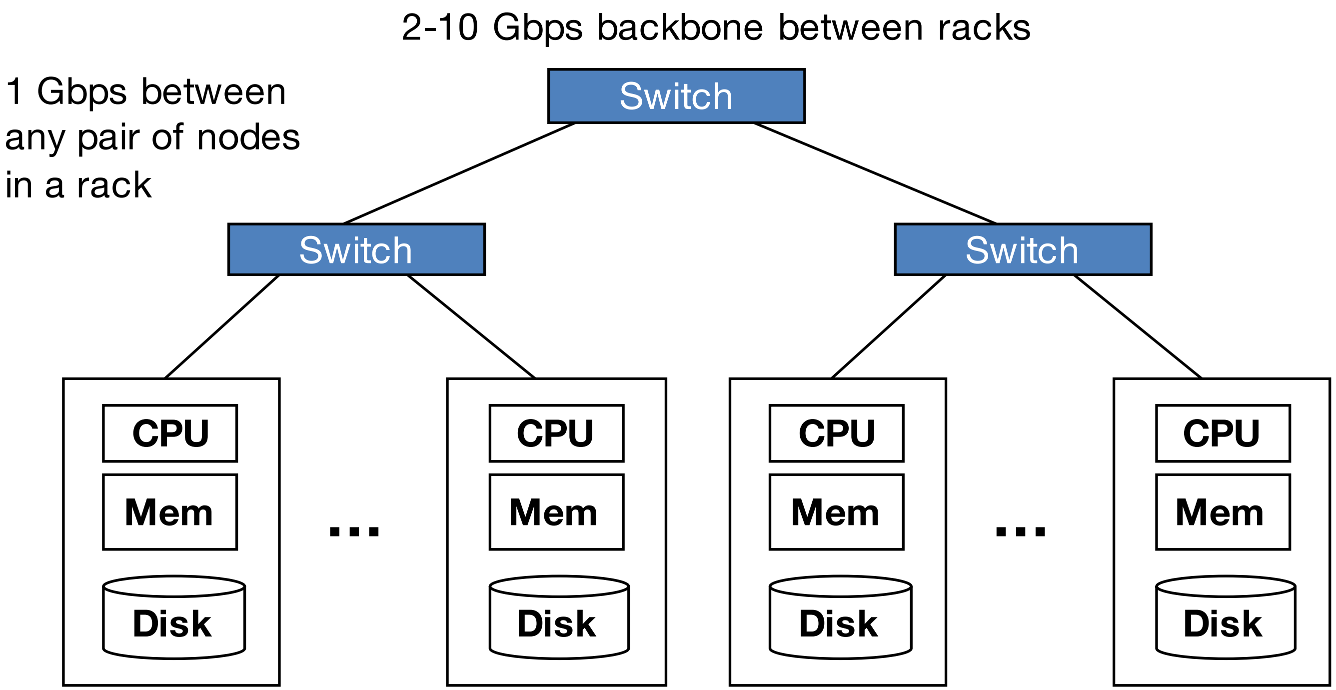
Machine Learning, Statistics

"Classical" Data Mining

Motivation: Google Example

- 20+ billion web pages x 20KB = 400+ TB
- 1 computer reads 30-35 MB/sec from disk
  - ~4 months to read the web
- ~1,000 hard drives to store the web
- Takes even more to **do** something useful with the data!
- **Today, a standard architecture for such problems is emerging:**
  - Cluster of commodity Linux nodes
  - Commodity network (ethernet) to connect them

Cluster Architecture



Each rack contains 16-64 nodes
In 2011 it was guestimated that Google had 1M machines, http://bit.ly/Shh0RO

Large-scale Computing

- **Large-scale computing** for **data mining** problems on **commodity hardware**
- **Challenges:**
  - **How do you distribute computation?**
  - **How can we make it easy to write distributed programs?**
  - **Machines fail:**
    - One server may stay up 3 years (1,000 days)
    - If you have 1,000 servers, expect to loose 1/day
    - People estimated Google had ~1M machines in 2011
      - 1,000 machines fail every day!

Idea and Solution

- **Issue: Copying data over a network takes time**
- **Idea:**
  - Bring computation close to the data
  - Store files multiple times for reliability
- **Map-reduce** addresses these problems
  - Google's computational/data manipulation model
  - Elegant way to work with big data
  - **Storage Infrastructure – File system**
    - Google: GFS. Hadoop: HDFS
  - **Programming model**
    - Map-Reduce

Storage Infrastructure

- **Problem:**
  - If nodes fail, how to store data persistently?
- **Answer:**
  - **Distributed File System:**
    - Provides global file namespace
    - Google GFS; Hadoop HDFS;
- **Typical usage pattern**
  - Huge files (100s of GB to TB)
  - Data is rarely updated in place
  - Reads and appends are common

Programming Model: Map Reduce

Warm-up task:

- We have a huge text document
- Count the number of times each distinct word appears in the file
- **Sample application:**
  - Analyze web server logs to find popular URLs

Task: Word Count

Case 1:

- File too large for memory, but all <word, count> pairs fit in memory

Case 2:

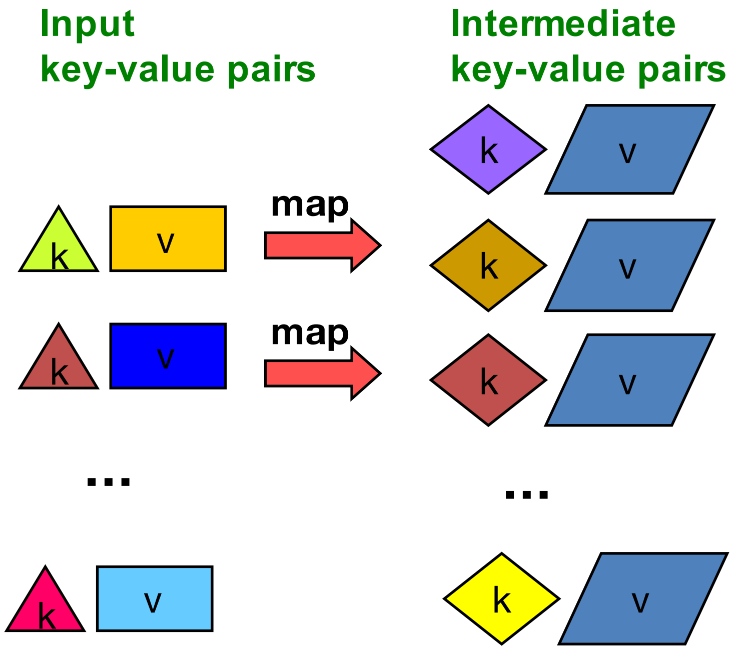
- Count occurrences of words:
  - `words(doc.txt) | sort | uniq -c`
    - where `words` takes a file and outputs the words in it, one per a line
- Case 2 captures the essence of **MapReduce**
  - Great thing is that it is naturally parallelizable

Map Reduce: Overview

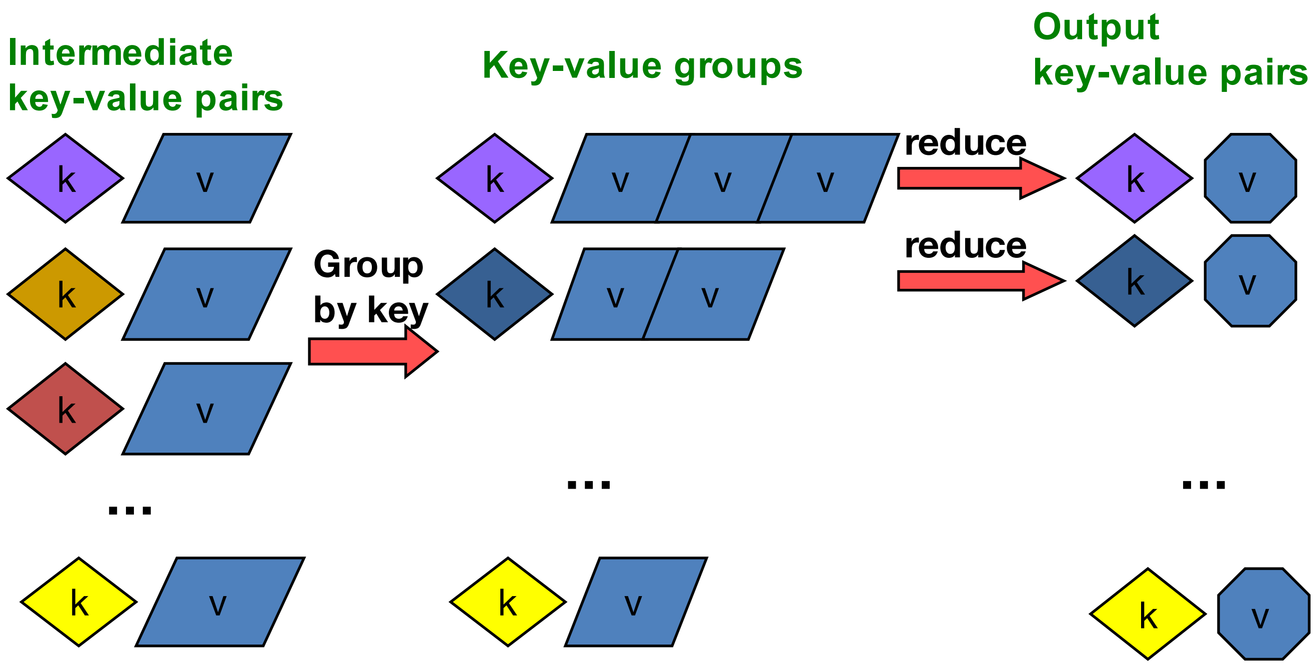
- Sequentially read a lot of data
- **Map:**
  - Extract something you care about
- **Group by key:** Sort and Shuffle
- **Reduce:**
  - Aggregate, summarize, filter or transform
- Write the result

Outline stays the same, **Map** and **Reduce** change to fit the problem

MapReduce: The Map Step



Map Reduce: The Reduce Step

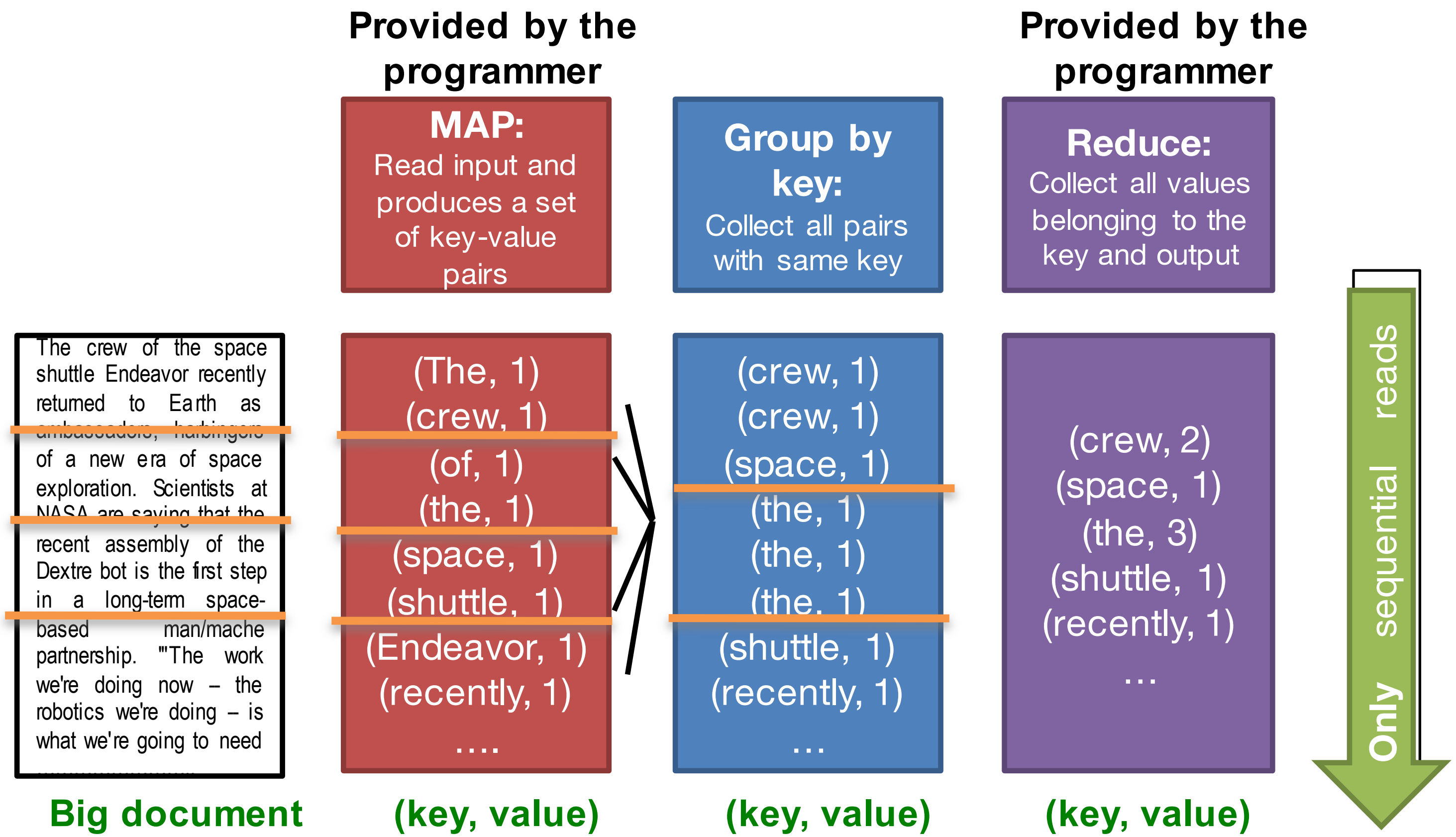


J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

More Specifically

- **Input:** a set of key-value pairs
- Programmer specifies two methods:
  - **Map(k, v)** → <k', v'>*
    - Takes a key-value pair and outputs a set of key-value pairs
      - E.g., key is the filename, value is a single line in the file
    - There is one Map call for every *(k,v)* pair
  - **Reduce(k', <v'>*)** → <k', v''>*
    - **All values *v'* with same key *k'* are reduced together and processed in *v'* order**
    - There is one Reduce function call per unique key *k'*

Map Reduce: Word Counting



J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Word Count Using Map Reduce

```
map(key, value):

// key: document name; value: text of the document

  for each word w in value:

        emit(w, 1)
reduce(key, values):
// key: a word; value: an iterator over counts
    result = 0
    for each count v in values:
        result += v
    emit(key, result)
```

J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Map-Reduce: Environment

Map-Reduce environment takes care of:

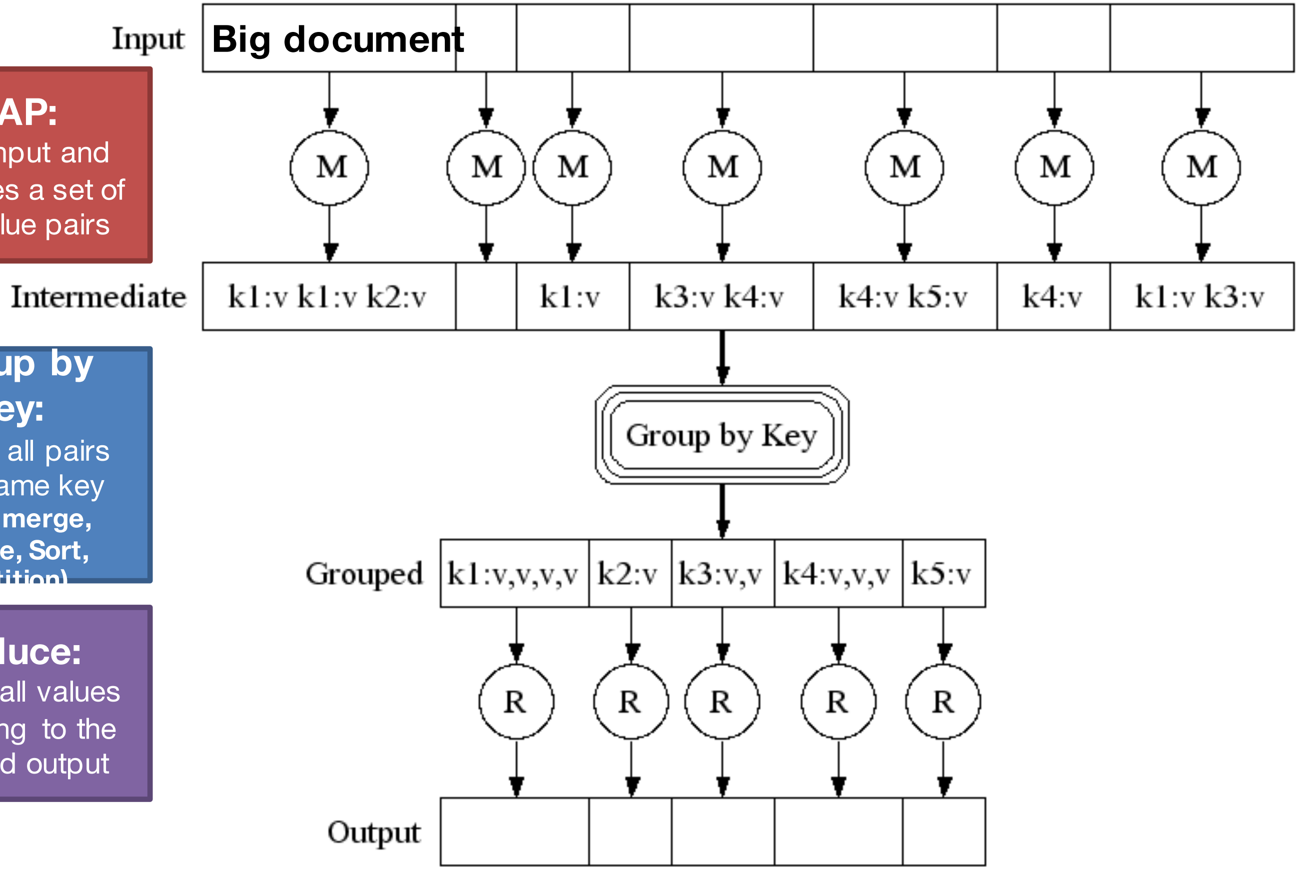
- Partitioning the input data
- Scheduling the program's execution across a set of machines
- Performing the **group by key** step
- Handling machine failures
- Managing required inter-machine communication

Map-Reduce: A diagram

MAP:
Read input and produces a set of key-value pairs

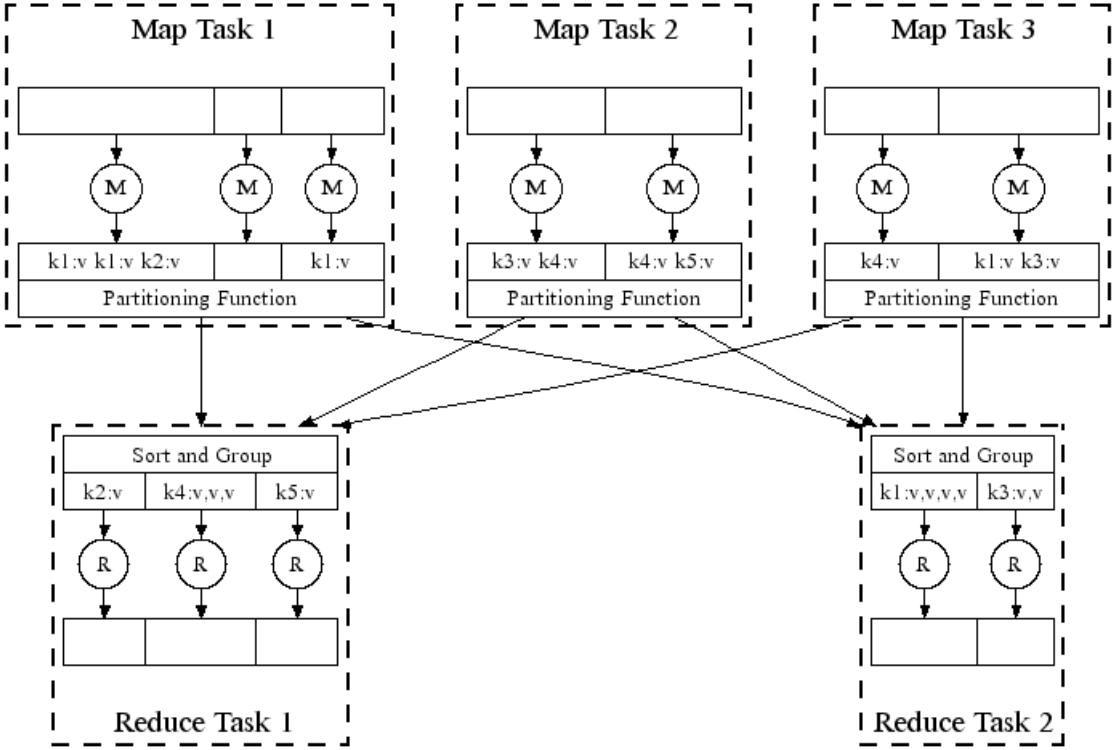
Group by key:
Collect all pairs with same key
(Hash merge, Shuffle, Sort, Partition)

Reduce:
Collect all values belonging to the key and output



J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

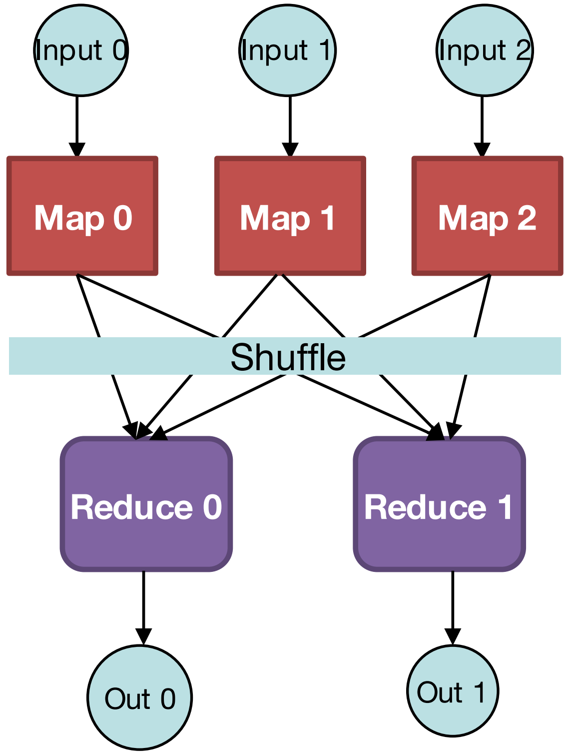
Map-Reduce: In Parallel



All phases are distributed with many tasks doing the work

Map-Reduce

- Programmer specifies:
  - Map and Reduce and input files
- **Workflow:**
  - Read inputs as a set of key-value-pairs
  - **Map** transforms input kv-pairs into a new set of k'v'-pairs
  - Sorts & Shuffles the k'v'-pairs to output nodes
  - All k'v'-pairs with a given k' are sent to the same **reduce**
  - **Reduce** processes all k'v'-pairs grouped by key into new k''v''-pairs
  - Write the resulting pairs to files
- All phases are distributed with many tasks doing the work



J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Data Flow

- **Input and final output are stored on a distributed file system (FS):**
  - Scheduler tries to schedule map tasks “close” to physical storage location of input data

- **Intermediate results are stored on local FS of Map and Reduce workers**

- **Output is often input to another MapReduce task**

Coordination: Master

- **Master node takes care of coordination:**
  - **Task status:** (idle, in-progress, completed)
  - **Idle tasks** get scheduled as workers become available
  - When a map task completes, it sends the master the location and sizes of its *R* intermediate files, one for each reducer
  - Master pushes this info to reducers
- Master pings workers periodically to detect failures

Dealing with Failures

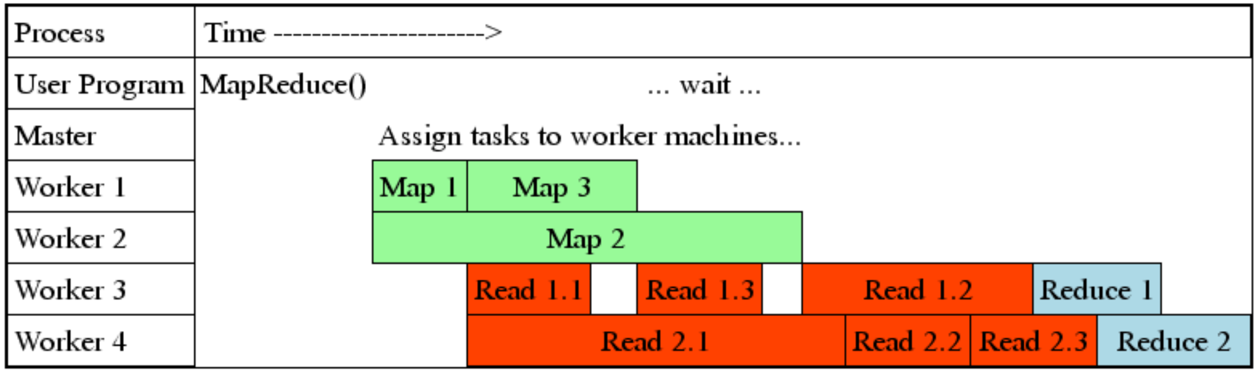
- **Map worker failure**
  - Map tasks completed or in-progress at worker are reset to idle
  - Reduce workers are notified when task is rescheduled on another worker
- **Reduce worker failure**
  - Only in-progress tasks are reset to idle
  - Reduce task is restarted
- **Master failure**
  - Map Reduce task is aborted and client is notified

How many Map and Reduce jobs?

- *M* map tasks, *R* reduce tasks
- **Rule of a thumb:**
  - Make *M* much larger than the number of nodes in the cluster
  - One DFS chunk per map is common
  - Improves dynamic load balancing and speeds up recovery from worker failures
- **Usually *R* is smaller than *M***
  - Because output is spread across *R* files

Task Granularity & Pipelining

- **Fine granularity tasks:** map tasks >> machines
  - Minimizes time for fault recovery
  - Can do pipeline shuffling with map execution
  - Better dynamic load balancing



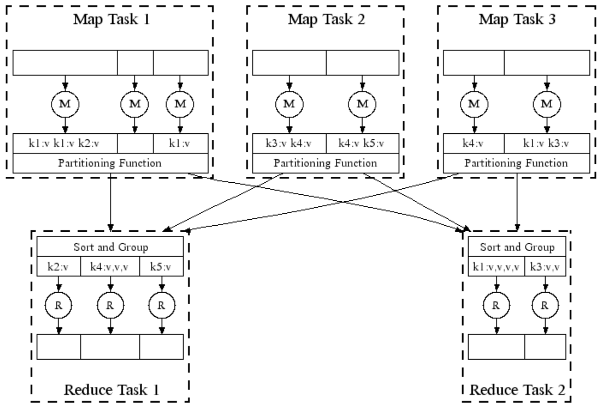
J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Refinements: Backup Tasks

- **Problem**
  - Slow workers significantly lengthen the job completion time:
    - Other jobs on the machine
    - Bad disks
    - Weird things
- **Solution**
  - Near end of phase, spawn backup copies of tasks
    - Whichever one finishes first "wins"
- **Effect**
  - Dramatically shortens job completion time

Refinement: Combiners

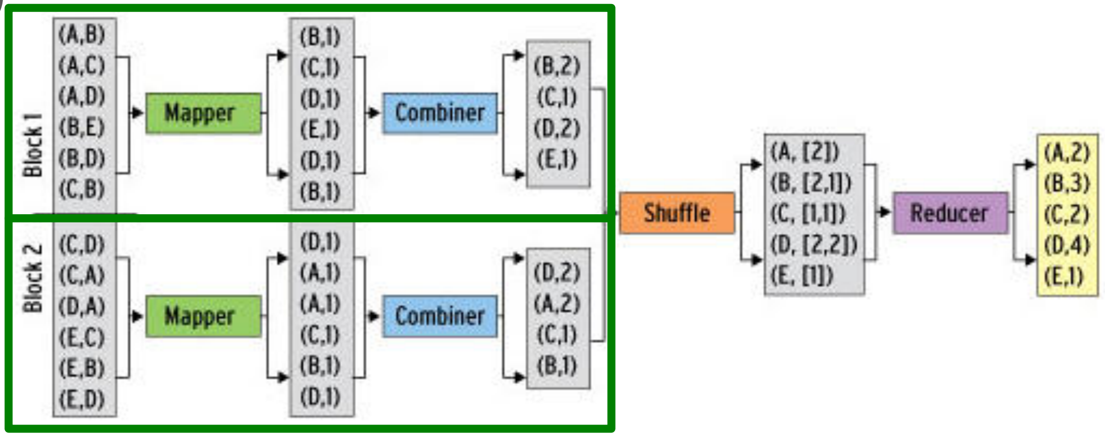
- Often a Map task will produce many pairs of the form $(k,v_1), (k,v_2), \ldots$ for the same key k
  - E.g., popular words in the word count example



- **Can save network time by pre-aggregating values in the mapper:**
  - combine(k, list(v_1)) → v_2
  - Combiner is usually same as the reduce function
- Works only if reduce function is commutative and associative

Refinement: Combiners

- **Back to our word counting example:**
  - Combiner combines the values of all keys of a single mapper (single machine):



  - Much less data needs to be copied and shuffled!

J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Refinement: Partition Function

- **Want to control how keys get partitioned**
  - Inputs to map tasks are created by contiguous splits of input file
  - Reduce needs to ensure that records with the same intermediate key end up at the same worker
- **System uses a default partition function:**
  - **hash(key) mod *R***
- **Sometimes useful to override the hash function:**
  - E.g., **hash(hostname(URL)) mod *R*** ensures URLs from a host end up in the same output file

hands on

Map reduce

Suited problems

Example: Host size

- **Suppose we have a large web corpus**
- Look at the metadata file
  - Lines of the form: (URL, size, date, …)
- **For each host, find the total number of bytes**
  - That is, the sum of the page sizes for all URLs from that particular host
- **Other examples:**
  - Link analysis and graph processing
  - Machine Learning algorithms

Example: Language Model

- **Statistical machine translation:**
  - Need to count number of times every 5-word sequence occurs in a large corpus of documents

- **Very easy with MapReduce:**
  - **Map:**
    - Extract (5-word sequence, count) from document
  - **Reduce:**
    - Combine the counts

Example: Join By Map-Reduce

- **Compute the natural join *R(A,B)* ⋈ *S(B,C)***
- *R* and *S* are each stored in files
- Tuples are pairs *(a,b)* or *(b,c)*

A	B
a_1	b_1
a_2	b_1
a_3	b_2
a_4	b_3

R

$\bowtie$

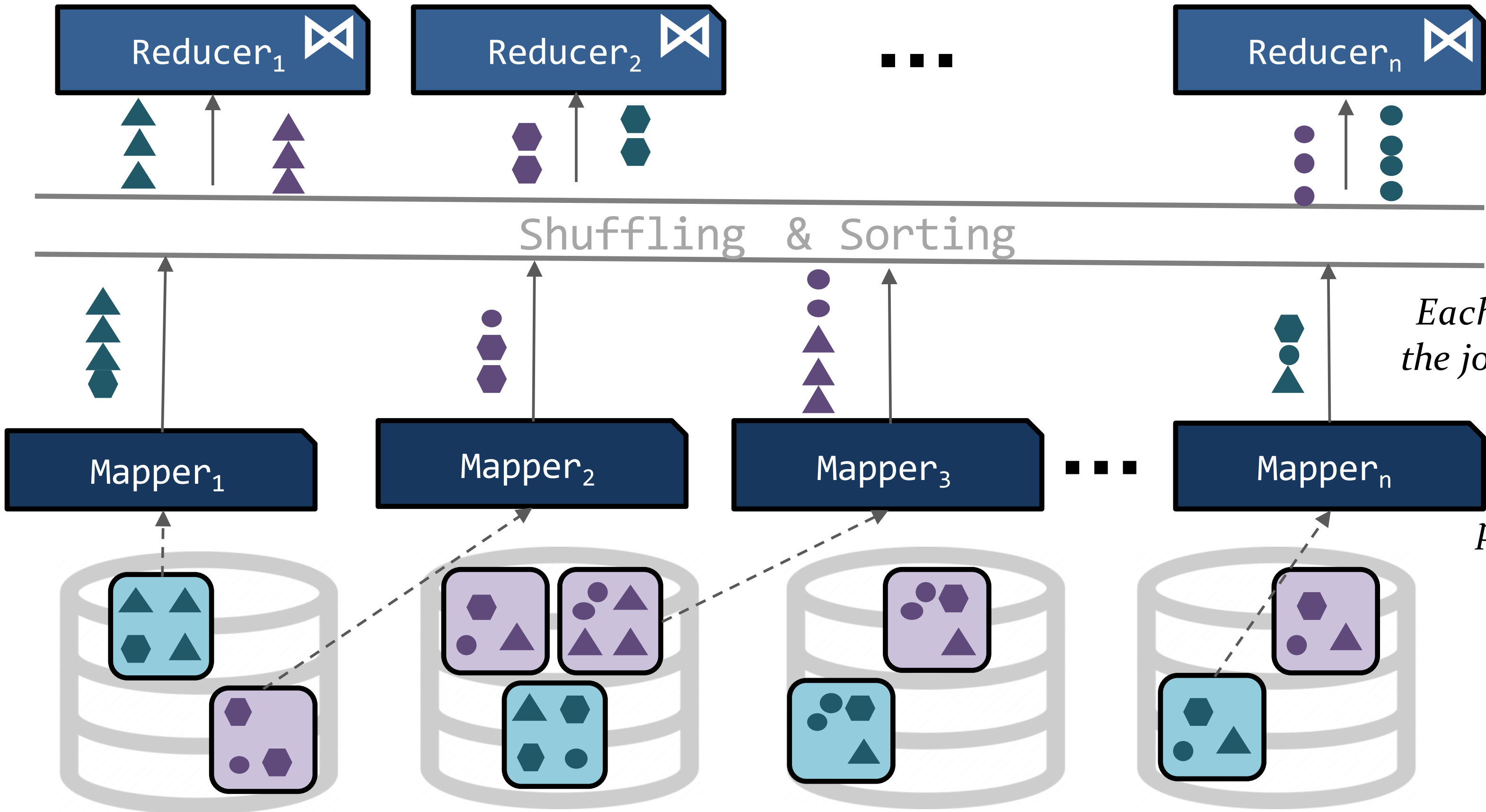
B	C
b_2	c_1
b_2	c_2
b_3	c_3

S

=

A	C
a_3	c_1
a_3	c_2
a_4	c_3

Map Reduce complex jobs



Reducers perform the actual join

Each mapper produces the join key & the record pairs

Each mapper processes one block

HDFS stores data blocks

Map-Reduce Join

- **Use a hash function *h* from B-values to *1...k***
- **A Map process turns:**
  - Each input tuple *R(a,b)* into key-value pair *(b,(a,R))*
  - Each input tuple *S(b,c)* into *(b,(c,S))*
- **Map processes** send each key-value pair with key *b* to Reduce process *h(b)*
  - Hadoop does this automatically; just tell it what *k* is.
- Each **Reduce process** matches all the pairs *(b,(a,R))* with all *(b,(c,S))* and outputs *(a,b,c)*.

hands on

Cost Measures for Algorithms

- **In MapReduce we quantify the cost of an algorithm using**

1. *Communication cost* = total I/O of all processes
2. *Elapsed communication cost* = max of I/O along any path
3. (*Elapsed*) *computation cost* analogous, but count only running time of processes

Note that here the big-O notation is not the most useful
(adding more machines is always an option)

Example: Cost Measures

- **For a map-reduce algorithm:**
  - **Communication cost =** input file size + 2 × (sum of the sizes of all files passed from Map processes to Reduce processes) + the sum of the output sizes of the Reduce processes.
  - **Elapsed communication cost** is the sum of the largest input + output for any map process, plus the same for any reduce process

What Cost Measures Mean

- Either the I/O (communication) or processing (computation) cost dominates
  - Ignore one or the other
- Total cost tells what you pay in rent from your friendly neighborhood cloud
- Elapsed cost is wall-clock time using parallelism

Cost of Map-Reduce Join

- **Total communication cost** = $O(|R|+|S|+|R \bowtie S|)$
- **Elapsed communication cost** = O(s)
  - We're going to pick ***k*** and the number of Map processes so that the I/O limit ***s*** is respected
  - We put a limit ***s*** on the amount of input or output that any one process can have. ***s* could be:**
    - What fits in main memory
    - What fits on local disk
- With proper indexes, computation cost is linear in the input + output size
  - So computation cost is like comm. cost

Map reduce summary

- Highly fault tolerant
- Relatively easy to write "arbitrary" distributed computations over very large amounts of data
- MR framework removes burden of dealing with failures from programmer

- Schema embedded in application code
- A lack of shared schema
- Makes sharing data between applications difficult
- Makes lots of DBMS "goodies" such as indices, integrity constraints, views, ... impossible
- No declarative query language

Pointers and further reading

Implementations

- Google
  - Not available outside Google
- **Hadoop**
  - An open-source implementation in Java
  - Uses HDFS for stable storage
  - Download: http://lucene.apache.org/hadoop/
- Aster Data
  - Cluster-optimized SQL Database that also implements MapReduce

Reading

- Jeffrey Dean and Sanjay Ghemawat: MapReduce: Simplified Data Processing on Large Clusters
  - http://labs.google.com/papers/mapreduce.html
- Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung: The Google File System
  - http://labs.google.com/papers/gfs.html

Resources

- Hadoop Wiki
  - Introduction
    - http://wiki.apache.org/lucene-hadoop/
  - Getting Started
    - http://wiki.apache.org/lucene-hadoop/GettingStartedWithHadoop
  - Map/Reduce Overview
    - http://wiki.apache.org/lucene-hadoop/HadoopMapReduce
    - http://wiki.apache.org/lucene-hadoop/HadoopMapRedClasses
  - Eclipse Environment
    - http://wiki.apache.org/lucene-hadoop/EclipseEnvironment
- Javadoc
  - http://lucene.apache.org/hadoop/docs/api/

Resources

- Releases from Apache download mirrors
  - http://www.apache.org/dyn/closer.cgi/lucene/hadoop/
- Nightly builds of source
  - http://people.apache.org/dist/lucene/hadoop/nightly/
- Source code from subversion
  - http://lucene.apache.org/hadoop/version_control.html

Further Reading

- Programming model inspired by functional language primitives
- Partitioning/shuffling similar to many large-scale sorting systems
  - NOW-Sort ['97]
- Re-execution for fault tolerance
  - BAD-FS ['04] and TACC ['97]
- Locality optimization has parallels with Active Disks/Diamond work
  - Active Disks ['01], Diamond ['04]
- Backup tasks similar to Eager Scheduling in Charlotte system
  - Charlotte ['96]
- Dynamic load balancing solves similar problem as River's distributed queues
  - River ['99]

Pig

"Pig Latin: A Not-So-Foreign Language for Data Processing"

- Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, Andrew Tomkins (Yahoo! Research)
- http://www.sigmod08.org/program_glance.shtml#sigmod_industrial_program
- http://infolab.stanford.edu/~usriv/papers/pig-latin.pdf

Pig

General description

- High level data flow language for exploring very large datasets
- Compiler that produces sequences of MapReduce programs
- Structure is amenable to substantial parallelization
- Operates on files in HDFS
- Metadata not required, but used when available
- Provides an engine for executing data flows in parallel on Hadoop

Key properties

- **Ease of programming**
  - Trivial to achieve parallel execution of simple and parallel data analysis tasks
- **Optimization opportunities**
  - Allows the user to focus on semantics rather than efficiency
- **Extensibility**
  - Users can create their own functions to do special-purpose processing

Example

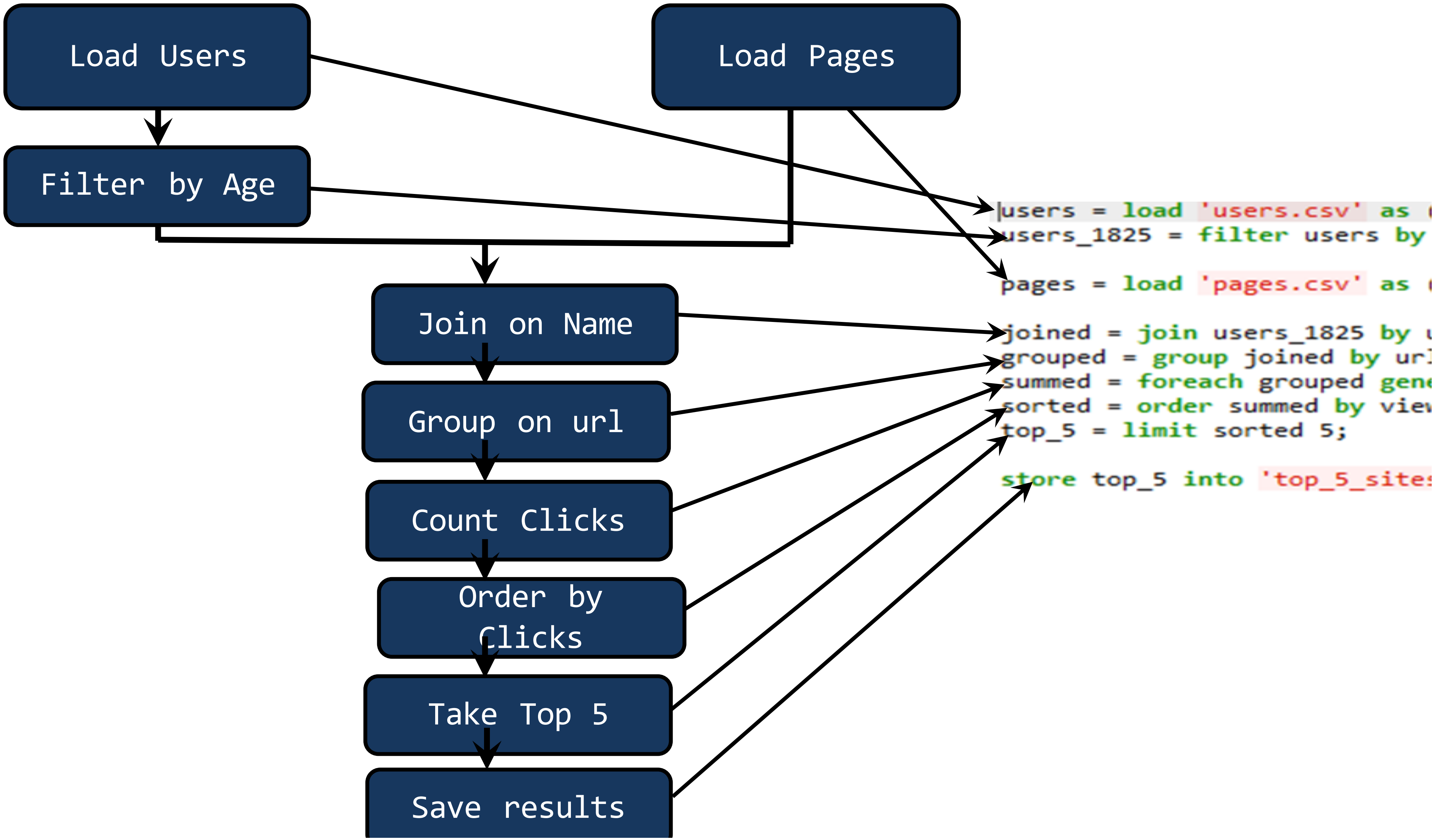
Top 5 pages accessed by users between 18 and 25 year

D:\1_TheFifthElephant_2012_Hands-on_Intro_to_Pig\top_5_sites.pig - Sublime Text 2

File Edit Selection Find View Goto Tools Project Preferences Help

top_5_sites.pig

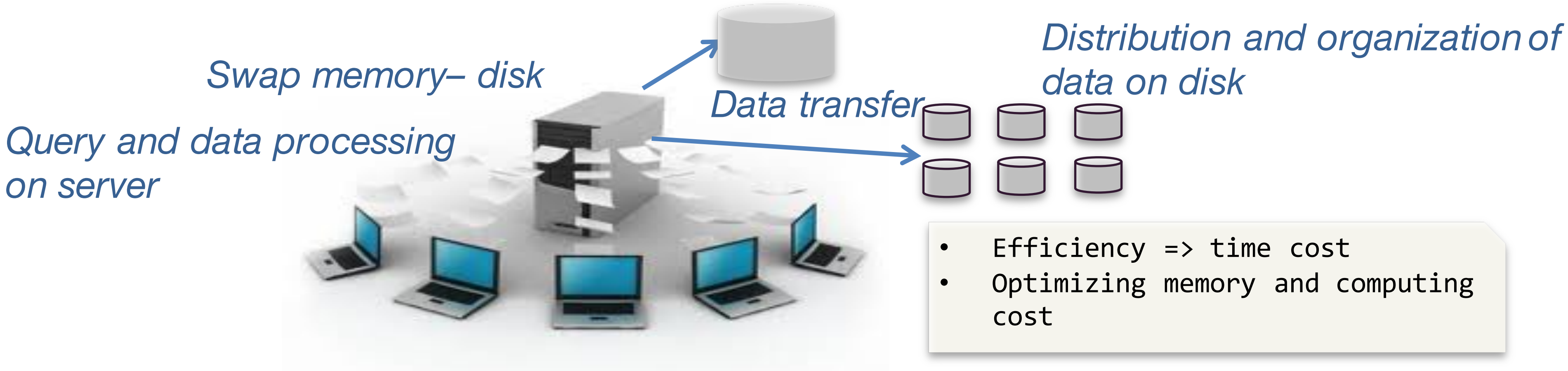
```
users = load 'users.csv' as (username:chararray, age:int);
users_1825 = filter users by age >= 18 and age <= 25;

pages = load 'pages.csv' as (username:chararray, url:chararray);

joined = join users_1825 by username, pages by username;
grouped = group joined by url;
summed = foreach grouped generate group as url, COUNT(joined) as views;
sorted = order summed by views desc;
top_5 = limit sorted 5;

store top_5 into 'top_5_sites.csv';

```


Load Users
Load Pages
Filter by Age
Join on Name
Group on url
Count Clicks
Order by Clicks
Take Top 5
Save results
users = load 'users.csv' as
users_1825 = filter users by
pages = load 'pages.csv' as
joined = join users_1825 by
grouped = group joined by url
summed = foreach grouped gen
sorted = order summed by view
top_5 = limit sorted 5;
store top_5 into 'top_5_site

Equivalent Java map reduce code

hands on

Querying **with** resources constraints



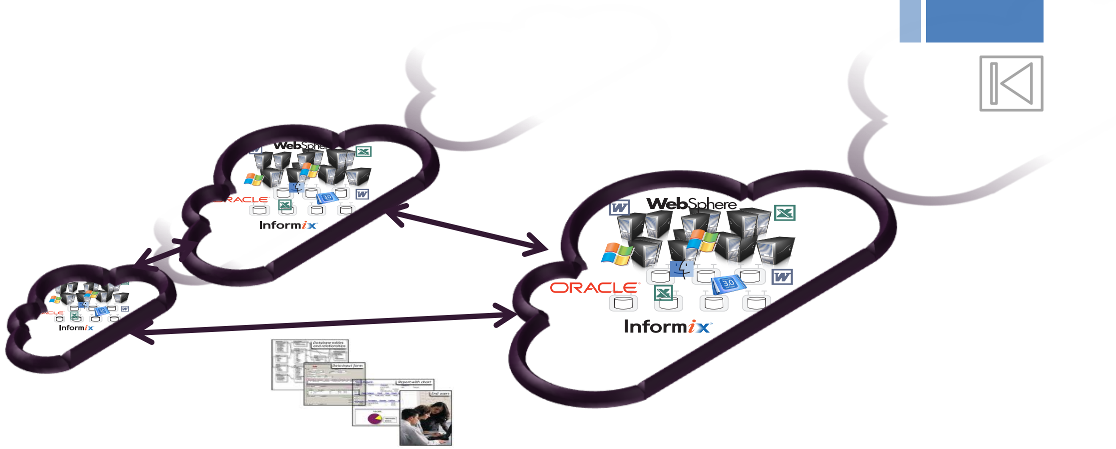
Q1: Which are the most popular products at Starbucks ?

Q2: Which are the consumption rules of Starbucks clients ?

Efficiently manage and exploit data sets according to given specific storage, memory and computation resources

Querying **without** resources constraints


```
Costly => minimizing cost, energy
consumption
```



- Query evaluation→ How and under which limits ?
  - Is not longer completely constraint by resources availability: computing, RAM, storage, network services
  - Decision making process determined by resources consumption and consumer requirements

- Data involved in the query, particularly in the result can have different costs: top 5 gratis and the rest available in return to a credit card number

- Results storage and exploitation demands more resources