BIG DATA ANALYTICS ENVIRONMENTS
Architectures, systems and properties

Genoveva Vargas-Solar

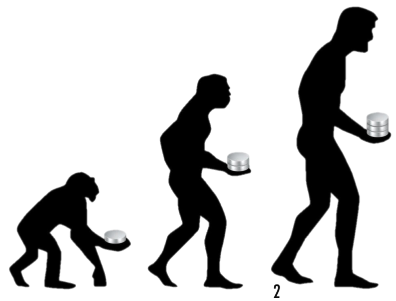
Senior Scientist, French Council of Scientific Research

Javier A. Espinosa Oviedo

Postdoctoral fellow, Barcelona Super Computing Centre

UFRN, Natal, October – November 2017
http://vargas-solar.com/datacentric-sciences/

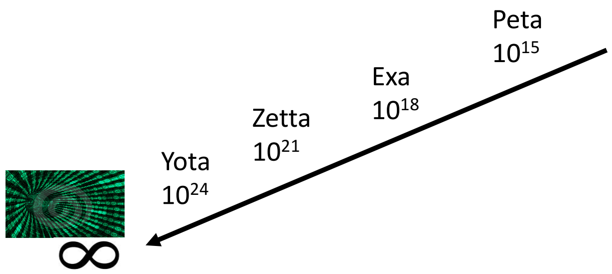
"*Design the* ***next generation*** *of* ***data processing systems & architectures guided by scientific requirements***"



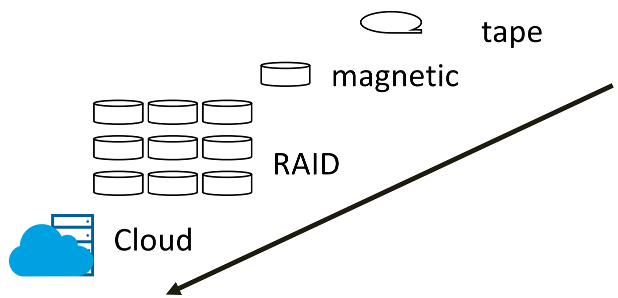
CLOUD DATA MANAGEMENT: SERVICES VIEWS

- Definition
- Querying and exploiting
- Manipulation

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



Data Volume



Hardware

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

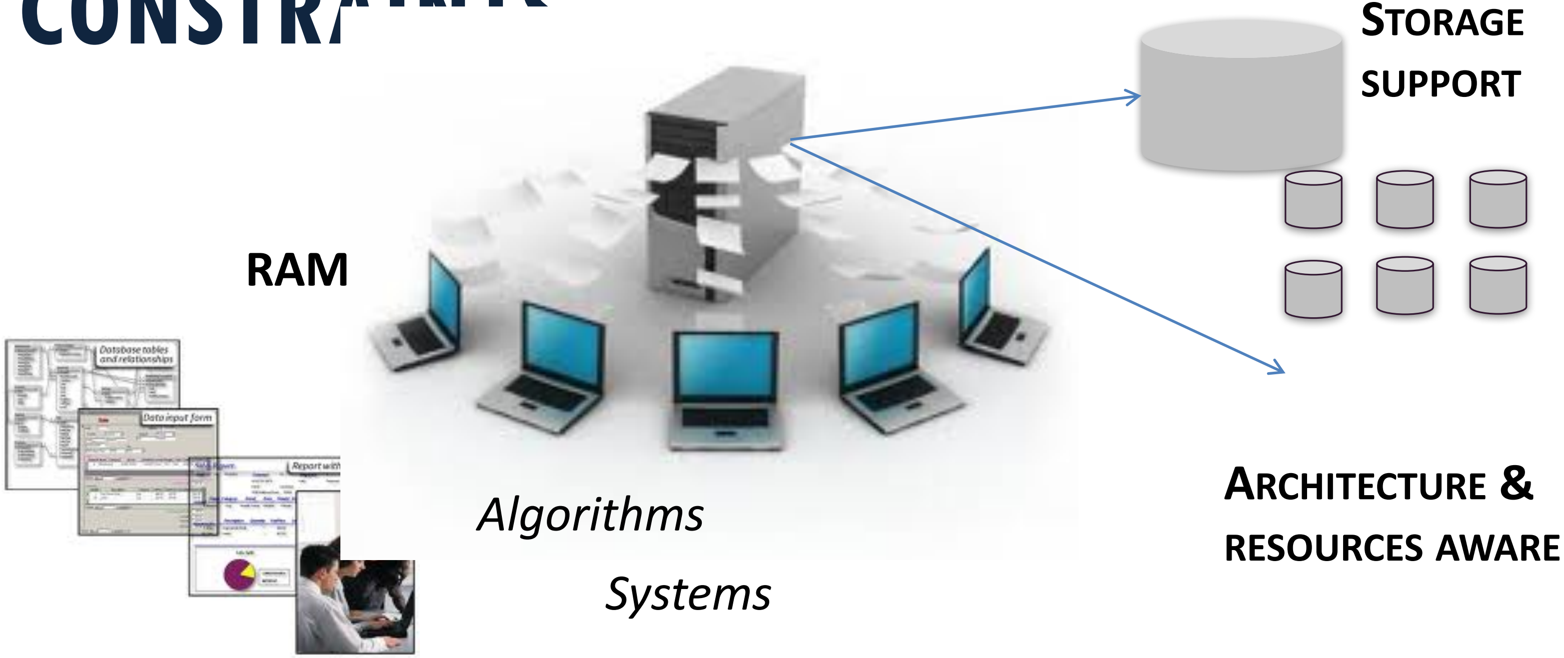
Self-adaptive Systems

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

DATA MANAGEMENT **WITH RESOURCES CONSTRAINTS**

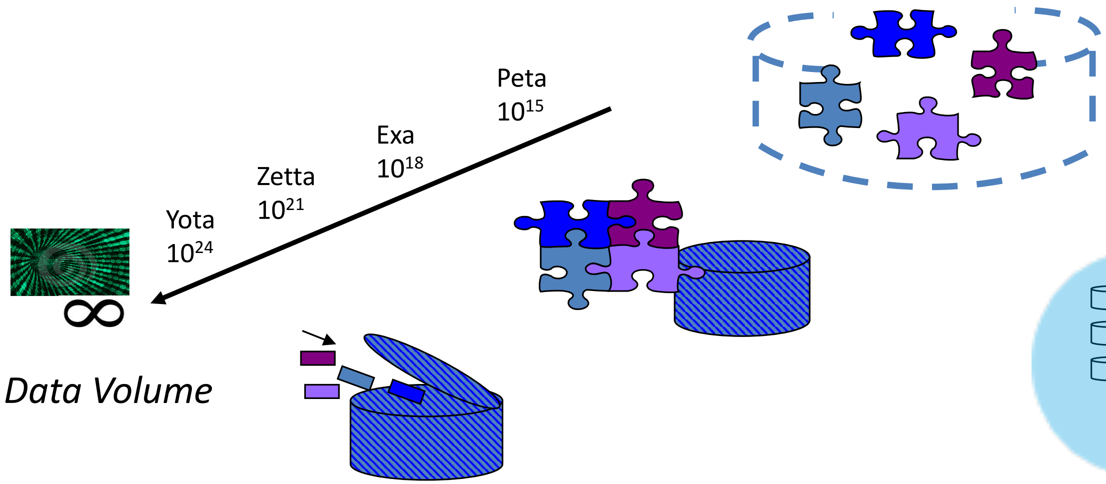


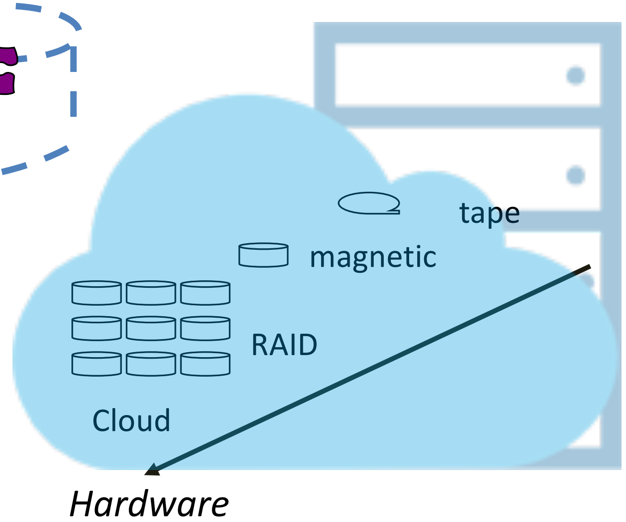
Efficiently manage and exploit data sets according to given specific storage, memory and computation resources

CLOUD DATA MANAGEMENT: SERVICES VIEWS

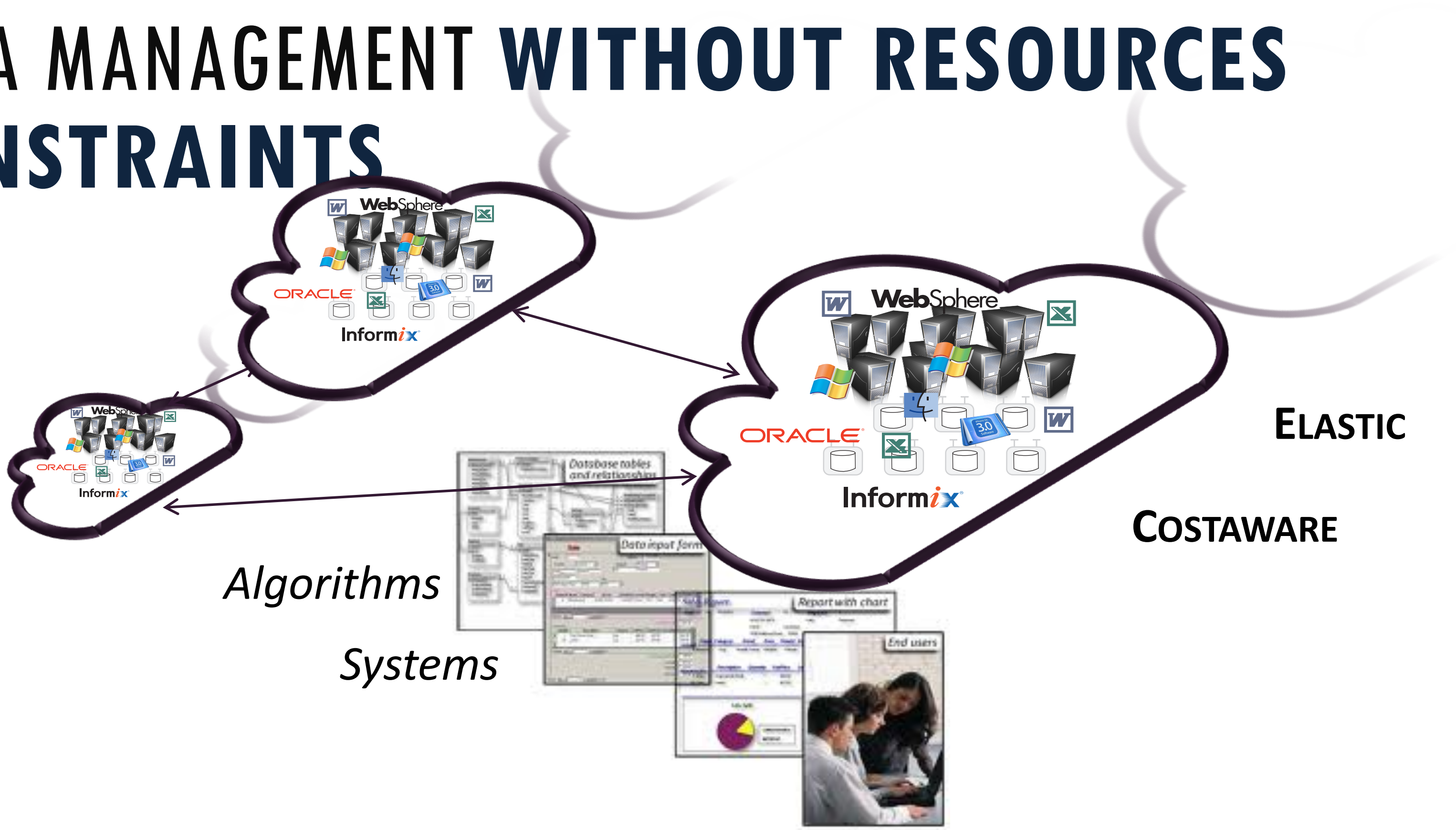
- Definition
- Querying and exploiting
- Manipulation

- 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 elasticity 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

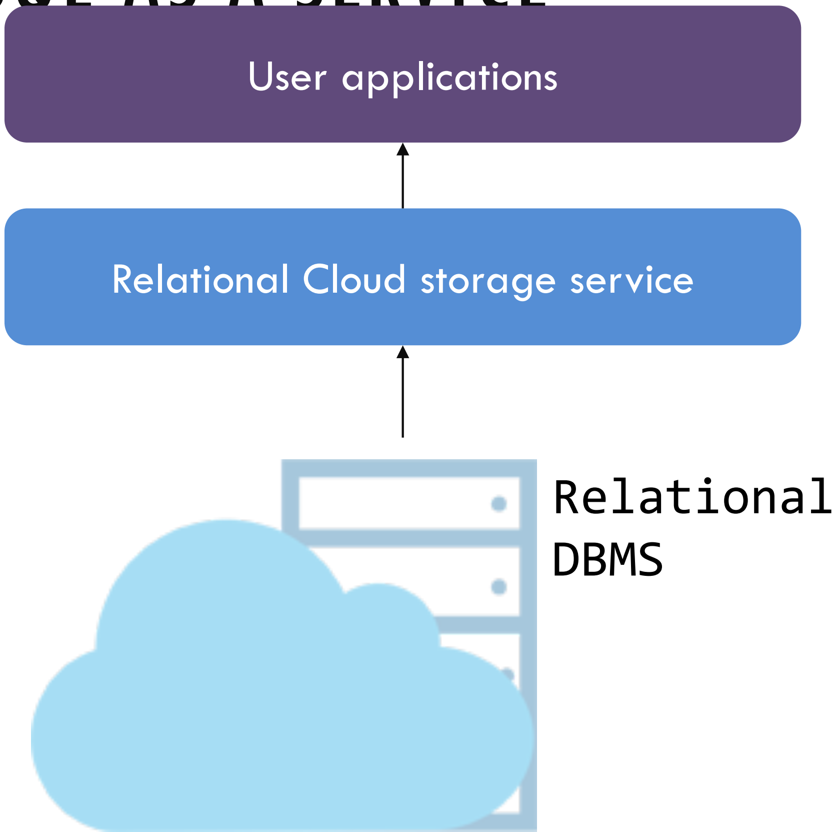
Multi-tenancy

- For OLTP [Agrawal1]
- For OLAP [Wong 2013]

Consistency, scalability and elasticity [Agrawal1]

- Replication and consistency models
- Elasticity

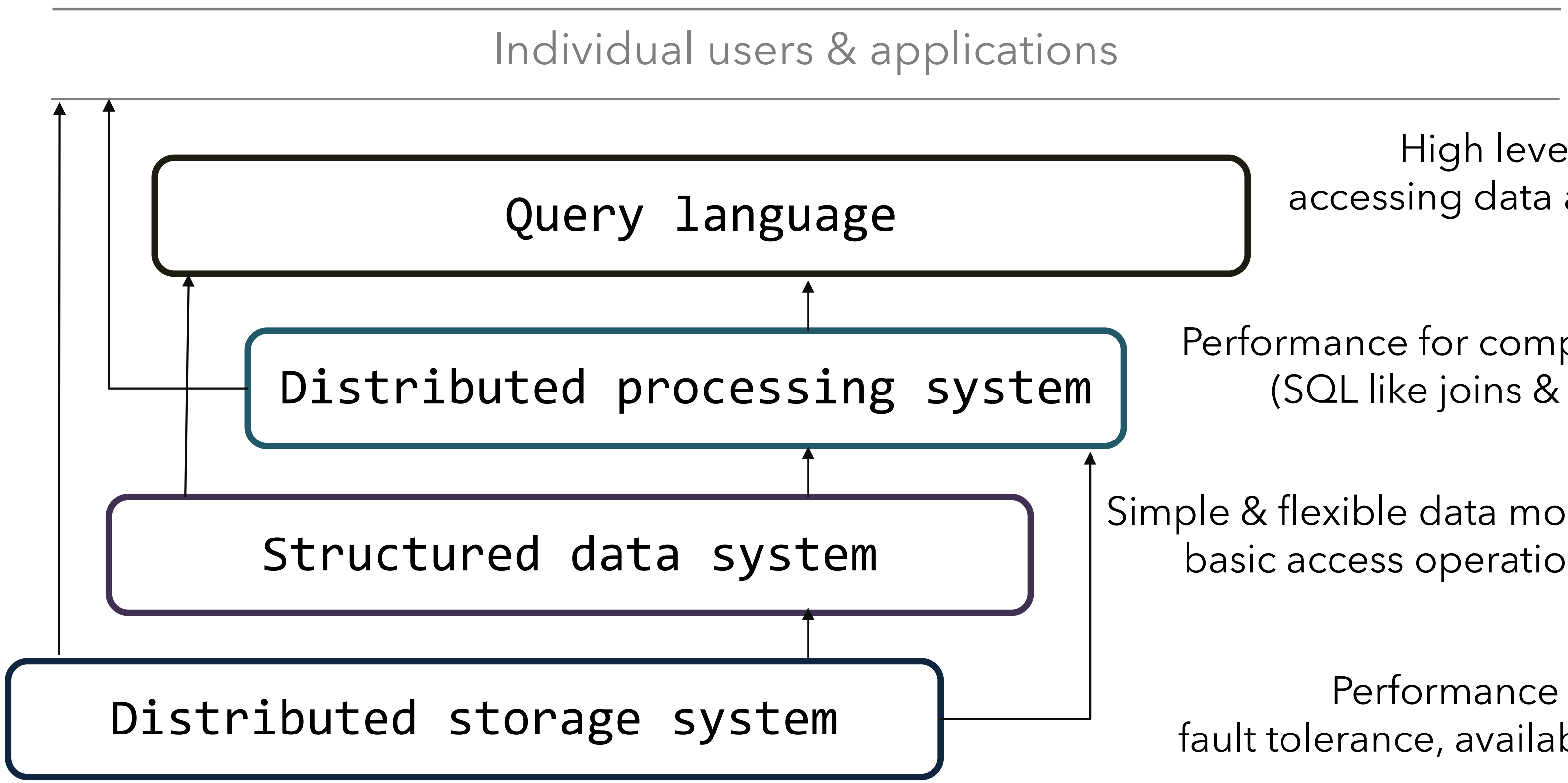
SQL AS A SERVICE



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



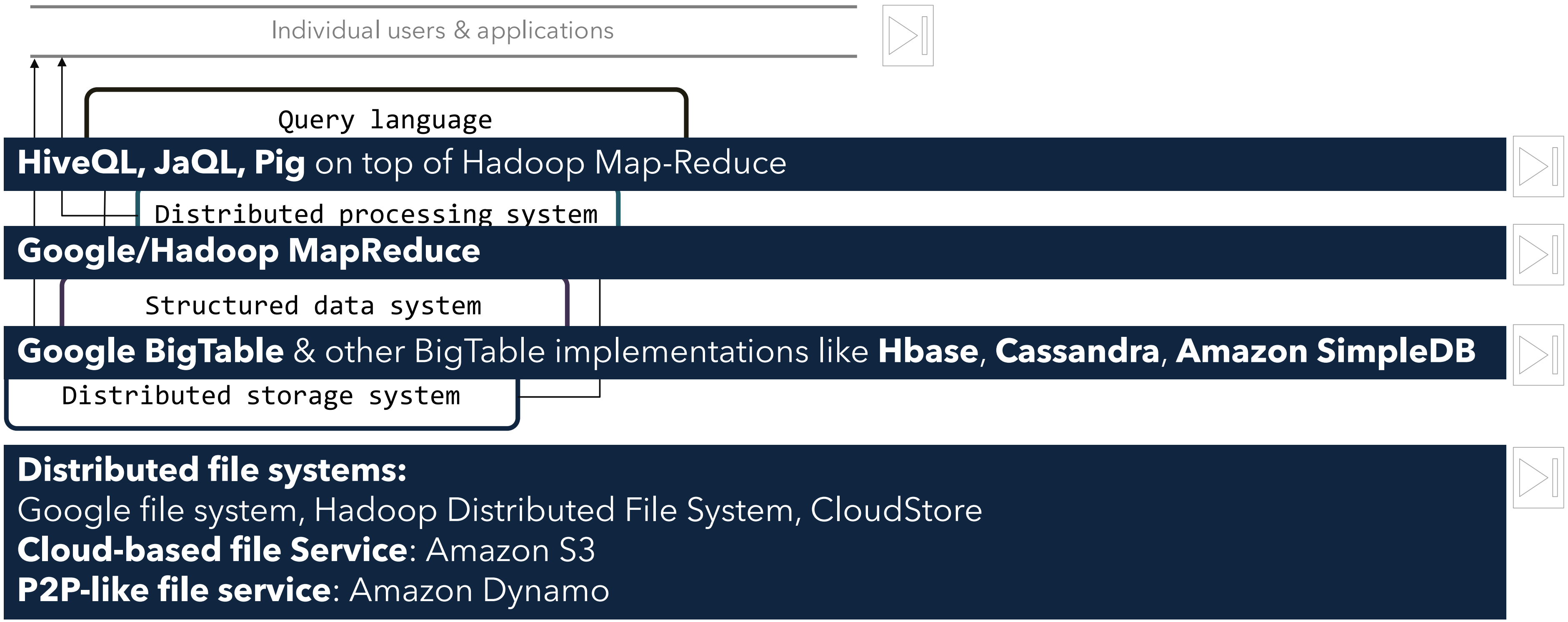
High level languages for accessing data and controlling processing

Performance for complex operations (SQL like joins & grouping, data analysis)

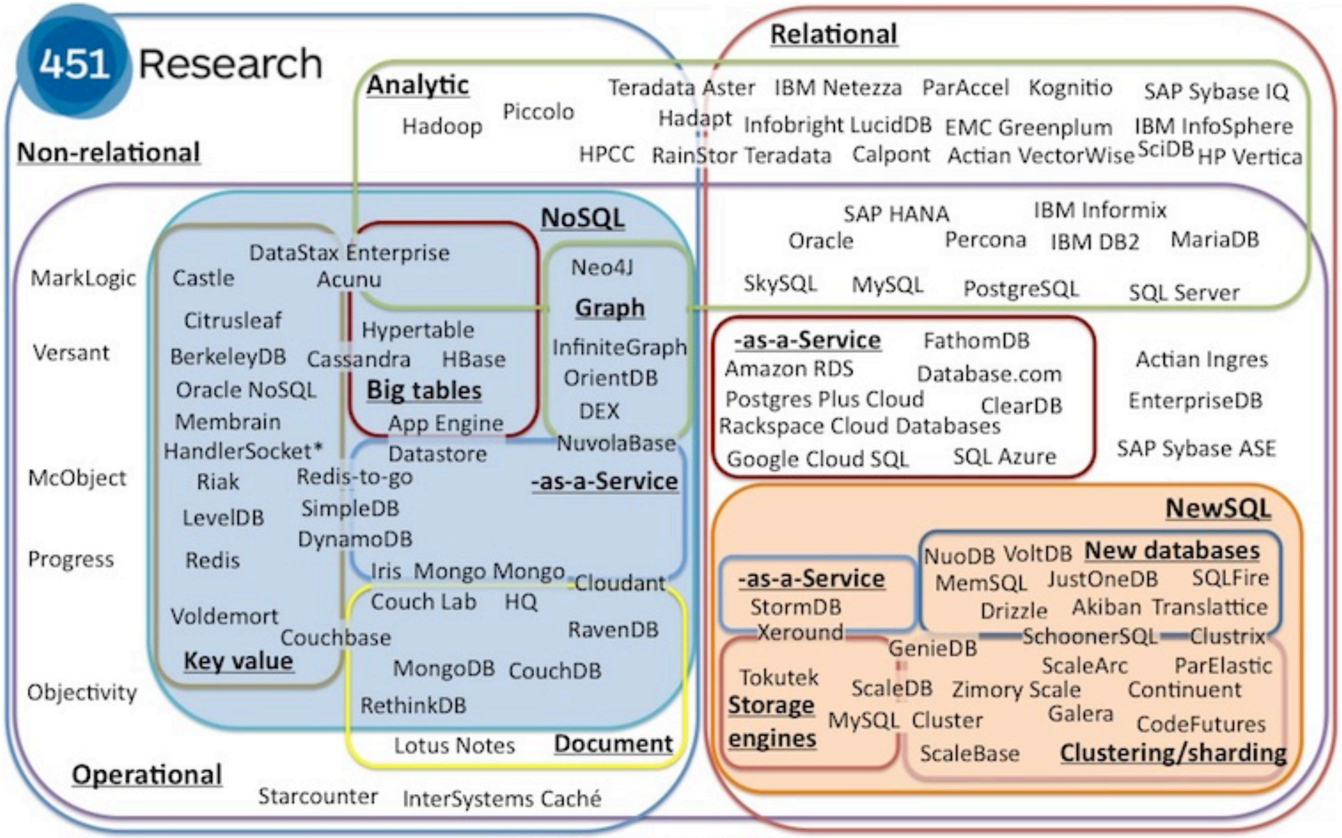
Simple & flexible data model (key-value), basic access operations (lookup API)

Performance for data access fault tolerance, availability, scalability

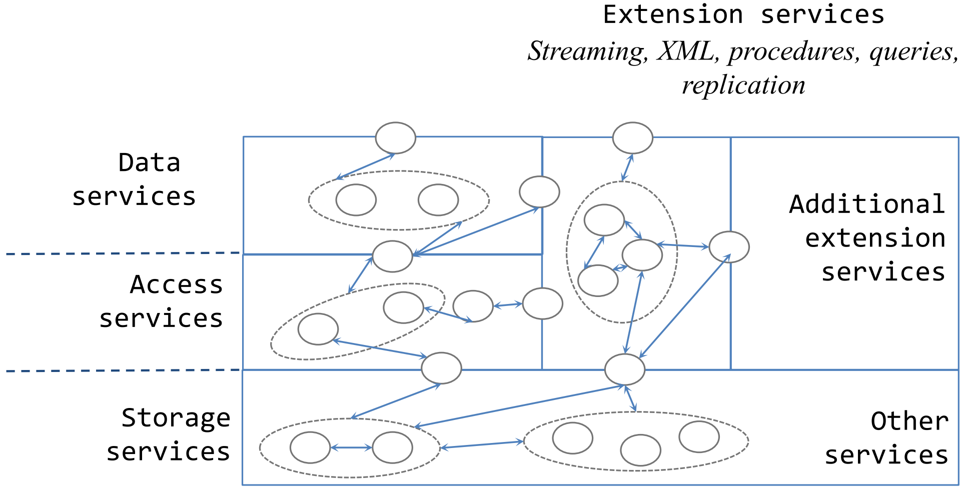
CLOUD DATA MANAGEMENT: FUNCTIONS VIEW

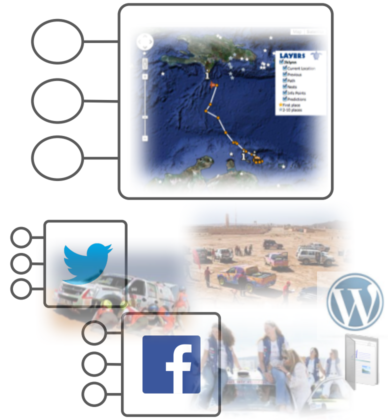


DATABASE LANDSCAPE



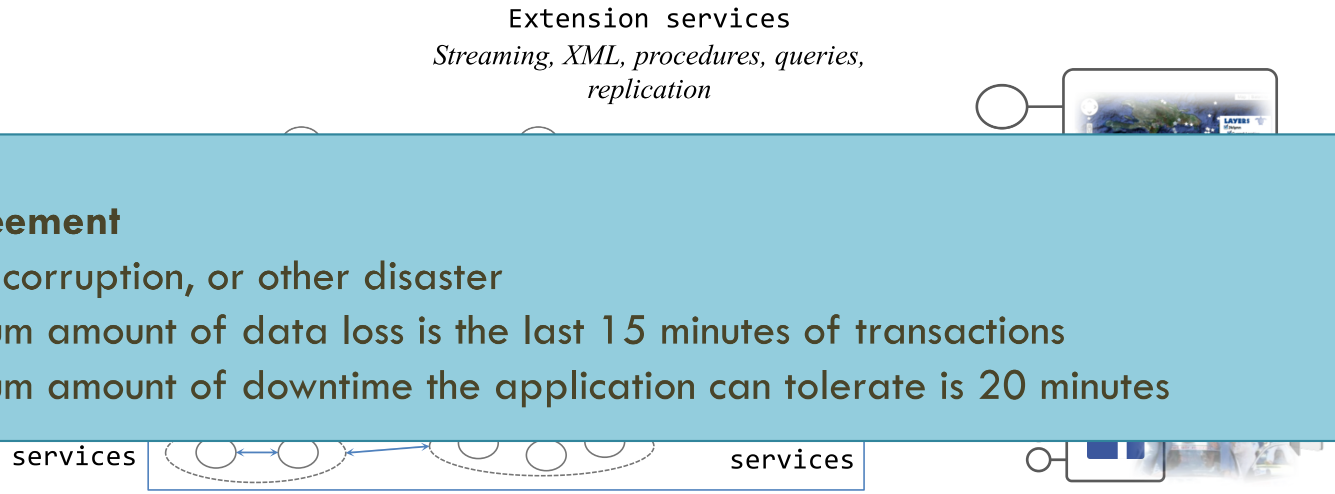
SERVICE ORIENTED DBMS





[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



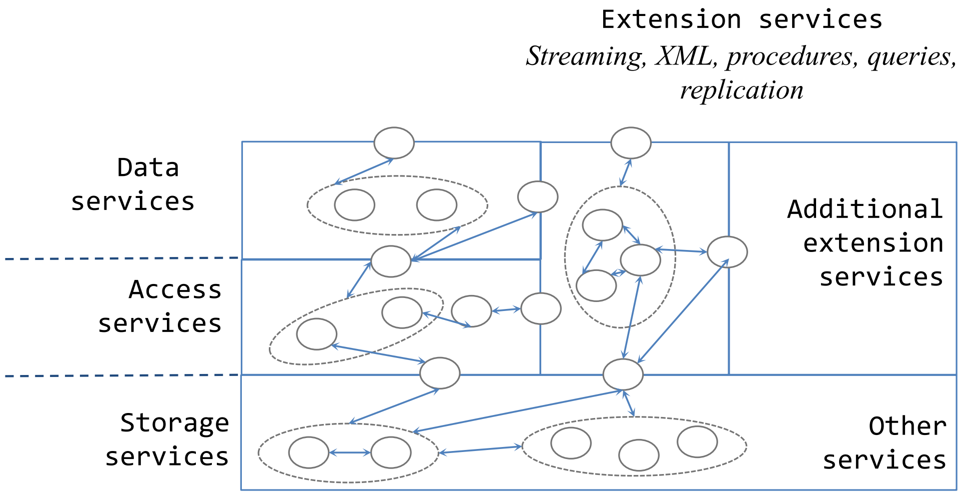
Service Level Agreement
- In the event of a corruption, or other disaster
  - the maximum amount of data loss is the last 15 minutes of transactions
  - the maximum amount of downtime the application can tolerate is 20 minutes

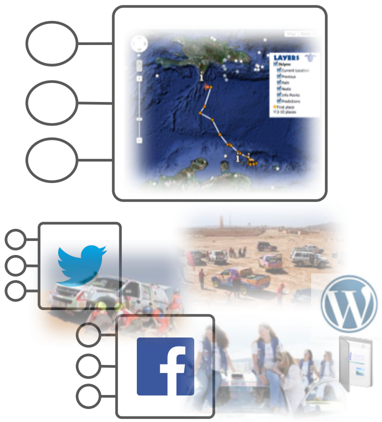
Service level agreement: the contracted delivery time of the service or performance

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





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

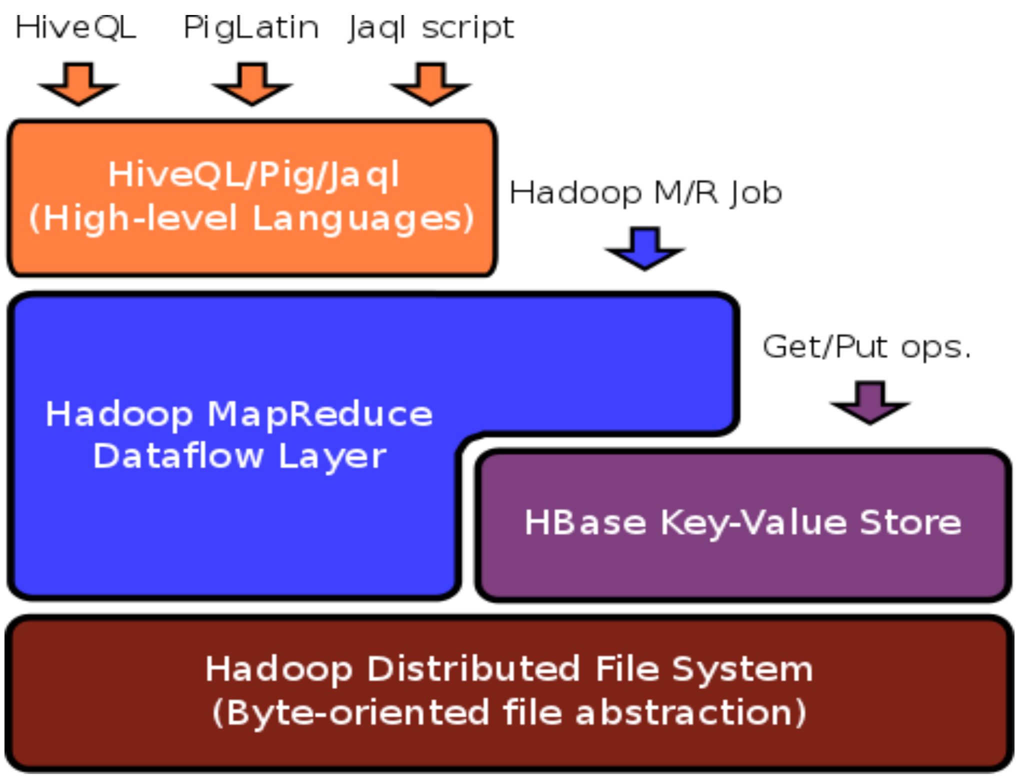
How to combine, deploy, and deliver DBMS functionalities:

- **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

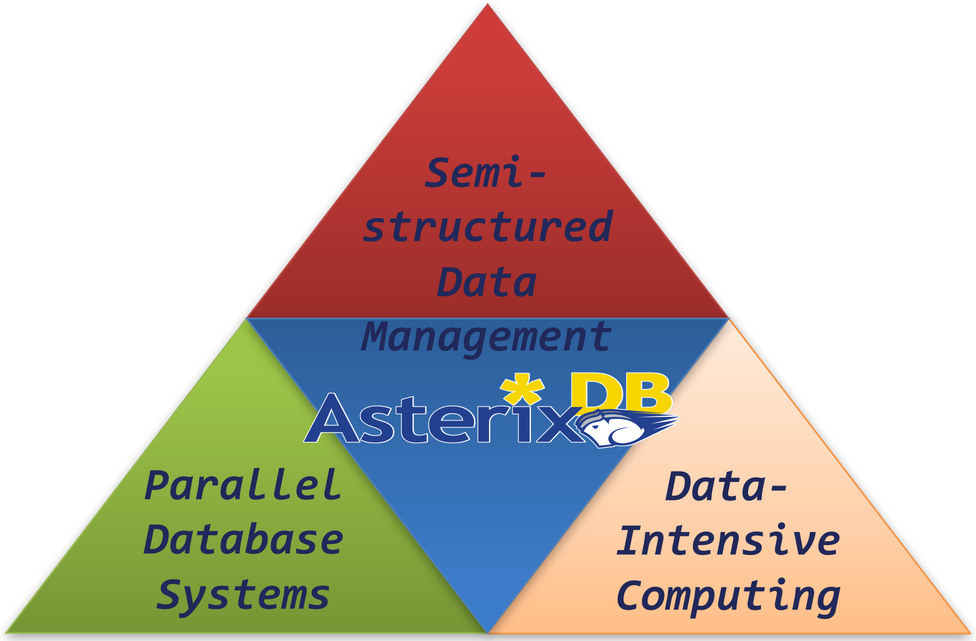


From Mike Carey

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

ASTERIX DB @ UCI

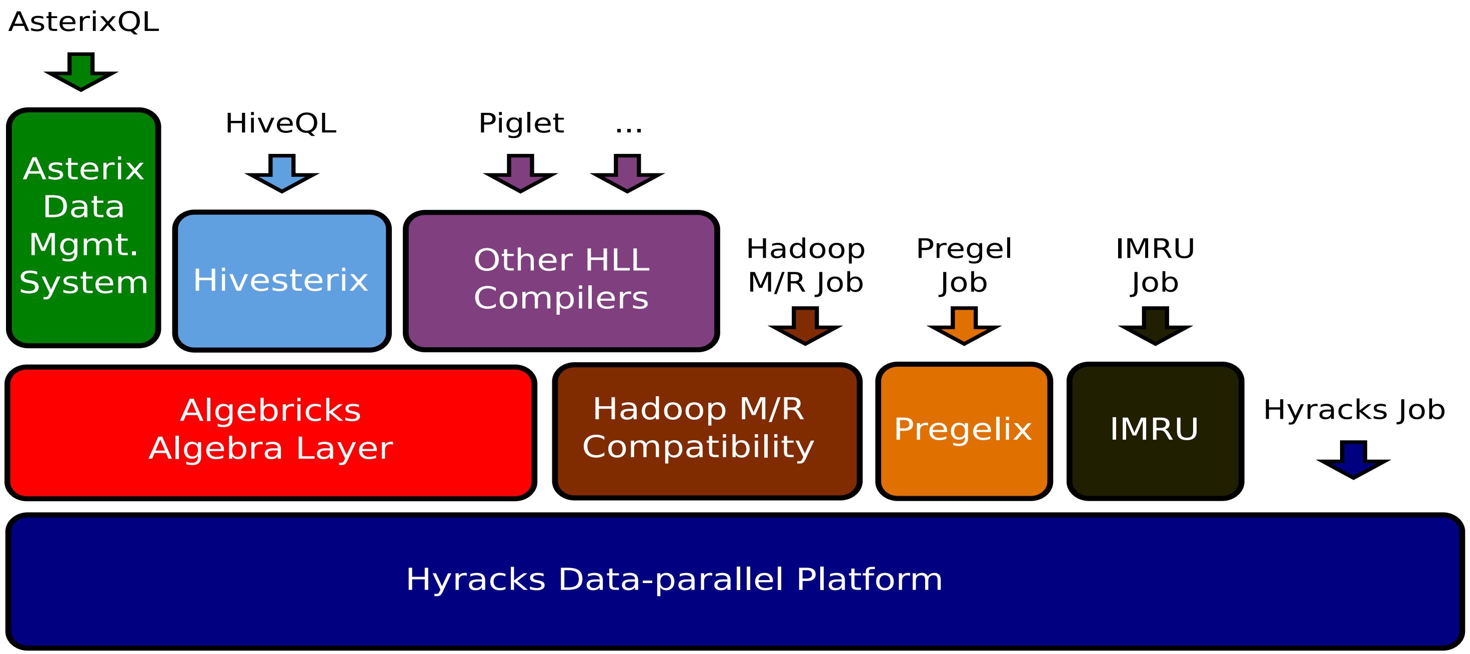


"One Size Fits a Bunch"

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)

ASTERIX SOFTWARE STACK



#ASTERIXDB

GOOGLE BIG QUERY

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)

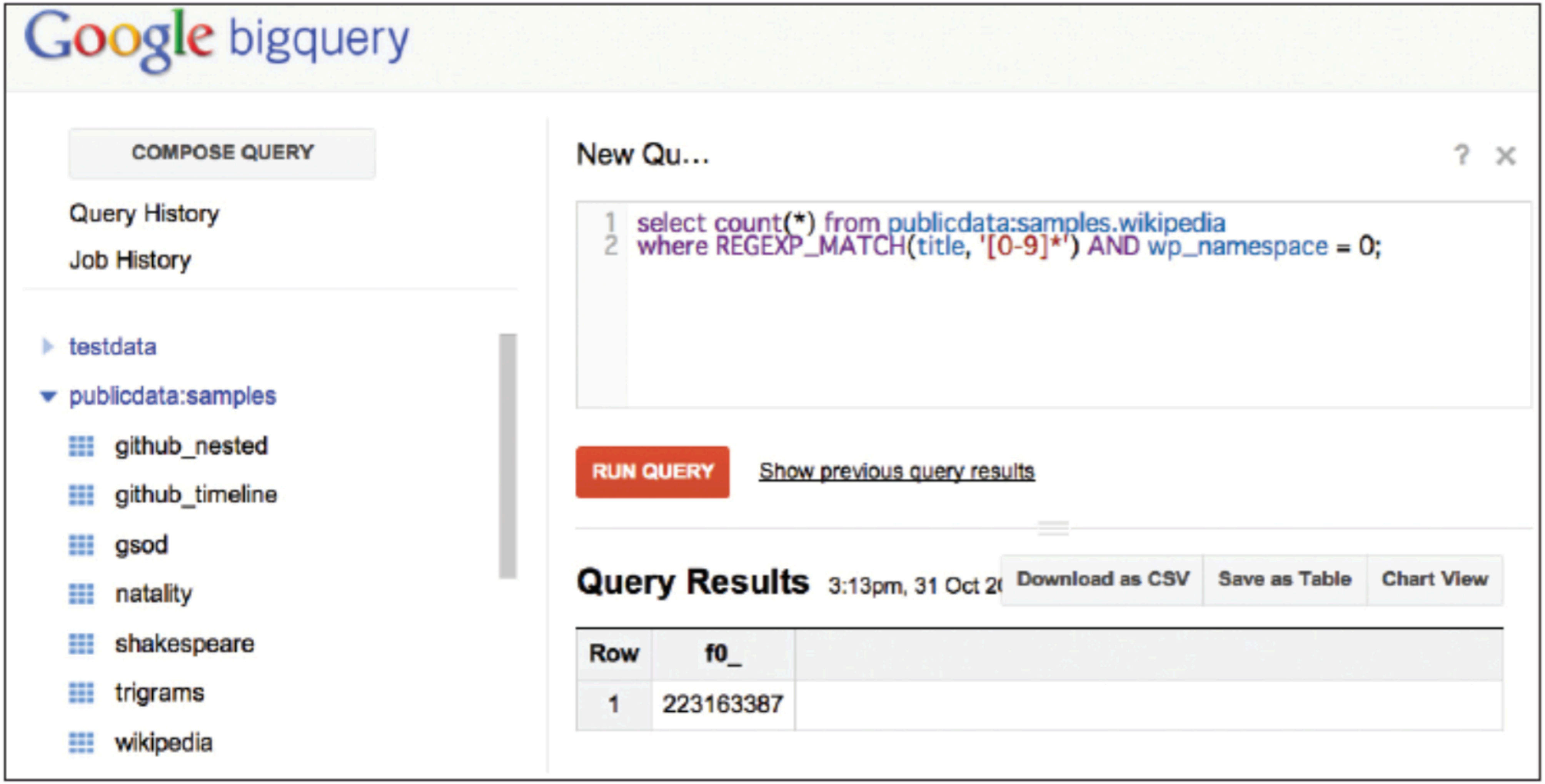
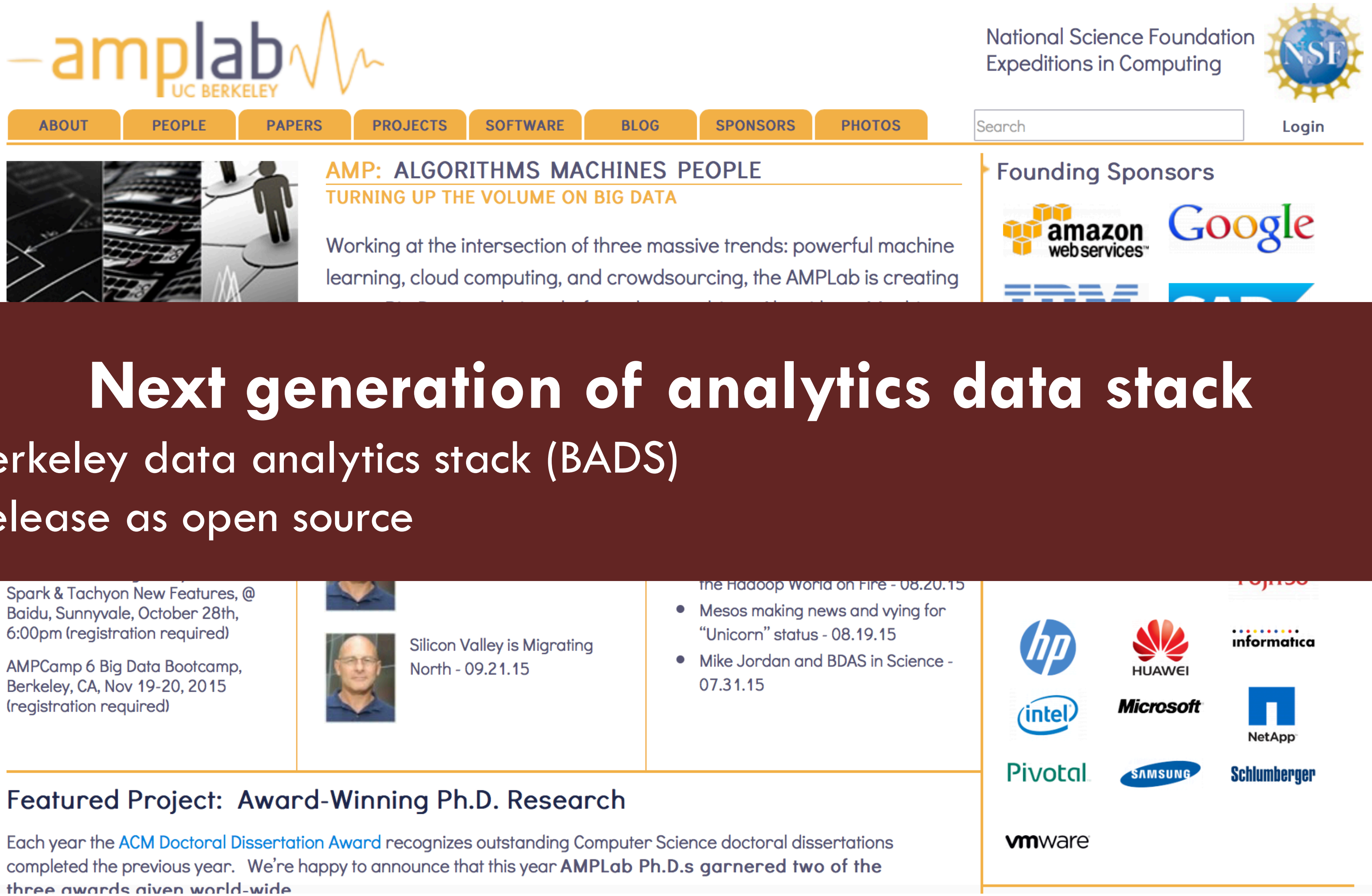


Figure 1 Querying Sample Wikipedia Table on BigQuery

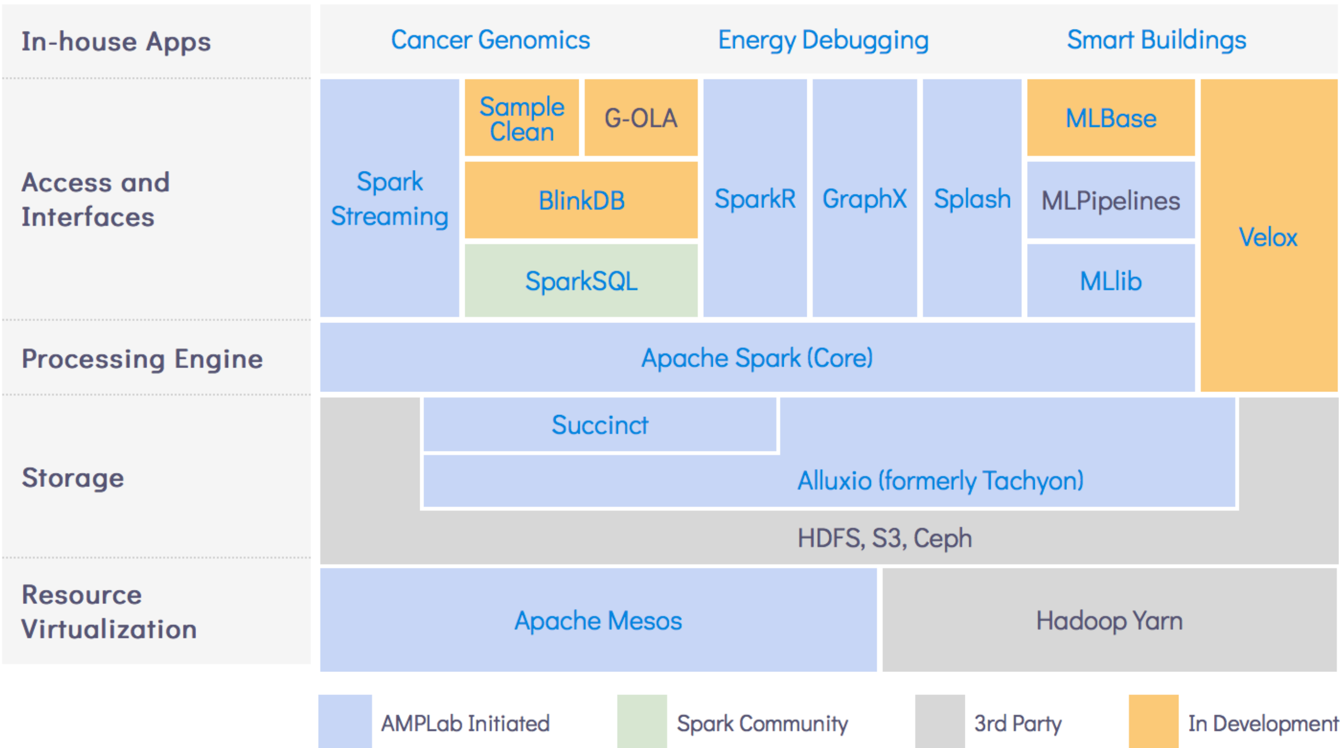
(You can try out BigQuery by simply sign up for it.)

Next generation of analytics data stack

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

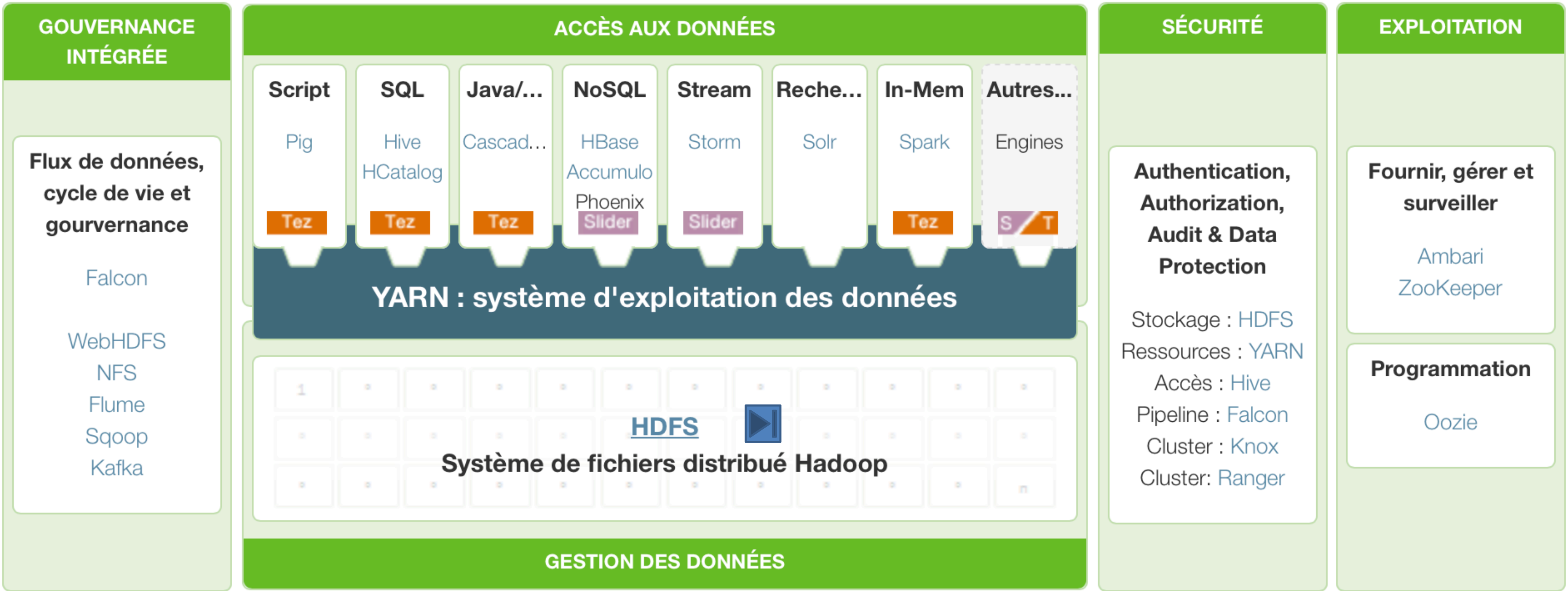


BERKELEY DATA ANALYTICS STACKS




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.

HORTONWORKS



http://fr.hortonworks.com

PRINCIPLE

$$\text{map: } (k_1, v_1) \longrightarrow [(k_2, v_2)]$$
$$\text{reduce: } (k_2, [v_2]) \longrightarrow [(k_3, v_3)]$$

Stage 1: Apply a user-specified computation over all input records in a dataset.

- These operations occur in parallel and yield intermediate output (**key-value** pairs)

Stage 2: Aggregate intermediate output by another user-specified computation

- Recursively applies a function on every pair of the list

COUNTING WORDS

(URI, document) → (term, count)

```
see bob throw
see spot run
```

```
see    1
bob    1
throw  1
see    1
spot   1
run    1
```

Map

```
bob   <1>
run   <1>
see   <1,1>
spot  <1>
throw <1>
```

Shuffle/Sort

```
bob    1
run    1
see    2
spot   1
throw  1
```

Reduce

KEY VALUE PAIRS

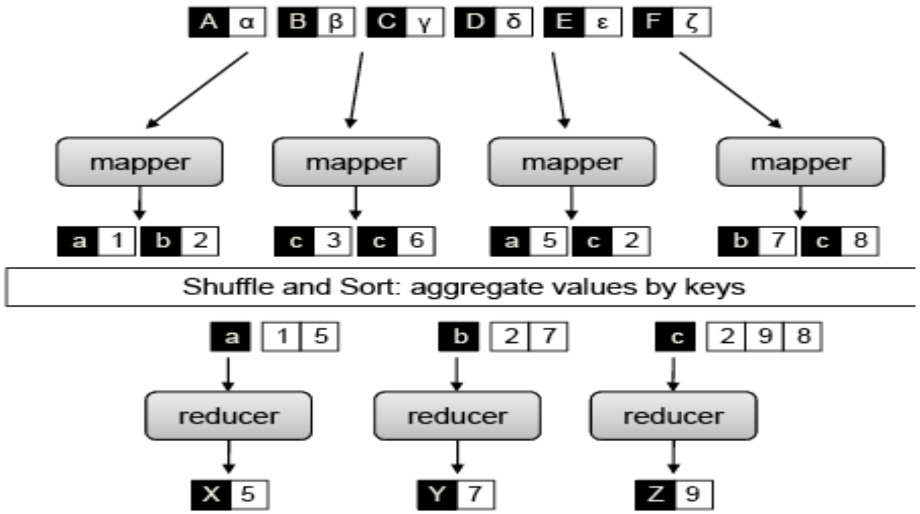
Basic data structure in MapReduce, keys and values may be

- primitive such as integers, floating point values, strings, and raw bytes
- arbitrarily complex structures (lists, tuples, associative arrays, etc.)

Part of the design of MapReduce algorithms involves imposing the key-value structure on arbitrary datasets

- For a collection of web pages, keys may be URLs and values may be the actual HTML content.
- For a graph, keys may represent node ids and values may contain the adjacency lists of those nodes

MAP REDUCE EXAMPLE



class MAPPER
  method MAP(docid a, doc d)
    for all term $t \in$ doc d **do**
      EMIT(term t, count 1)

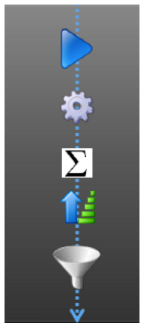
class REDUCER
  method REDUCE(term t, counts $[c_1, c_2, \ldots]$)
    $sum \leftarrow 0$
    for all count $c \in$ counts $[c_1, c_2, \ldots]$ **do**
      $sum \leftarrow sum + c$
    EMIT(term t, count sum)

MAP REDUCE PHASES

Initialisation

Map: *record reader, mapper, combiner, and partitioner*

Reduce: *shuffle, sort, reducer, and output format*

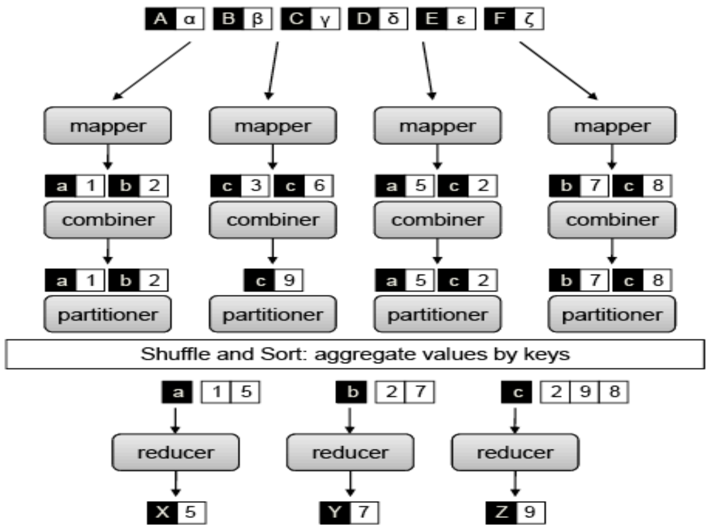


Partition input (key, value) pairs into chunks run map() tasks in parallel

After all map()'s have been completed consolidate the values for each unique emitted key

Partition space of output map keys, and run reduce() in parallel

MAP REDUCE ADDITIONAL ELEMENTS



Partitioners are responsible for dividing up the intermediate key space and assigning intermediate key-value pairs to reducers

- the partitioner species the task to which an intermediate key-value pair must be copied

Combiners are an optimization in MapReduce that allow for local aggregation before the shuffle and sort phase

COUNTING WORDS: BASIC ALGORITHM

```
class Mapper
    method Map(docid a, doc d)
        for all term t ∈ doc d do
            Emit(term t, count 1)
```

```
class Reducer
    method Reduce(term t, counts [c1, c2, ...])
        sum ← 0
        for all count c ∈ counts [c1, c2, ...] do
            sum ← sum + c
        Emit(term t, count sum)
```

the mapper emits an intermediate key-value pair for each term observed, with the term itself as the key and a value of one

reducers sum up the partial counts to arrive at the final count

LOCAL AGGREGATION

Combiner technique

- Aggregate term counts across the documents processed by each map task
- Provide a general mechanism within the MapReduce framework to reduce the amount of intermediate data generated by the mappers
- Reduction in the number of intermediate key-value pairs that need to be shuffled across the network
  - from the order of total number of terms in the collection to the order of the number of ***unique*** terms in the collection

```
class Mapper
    method Map(docid a, doc d)
        H ← new AssociativeArray
        for all term t ∈ doc d do
            H{t} ← H{t} + 1
        for all term t ∈ H do
            Emit(term t, count H{t})
```

IN-MAPPER COMBINING PATTERN: ONE STEP FURTHER

The workings of this algorithm critically depends on the details of how map and reduce tasks in Hadoop are executed

Prior to processing any input key-value pairs, the mapper's `Initialize` method is called

- which is an API hook for user-specified code
- We initialize an associative array for holding term counts
- Since it is possible to preserve state across multiple calls of the Map method (for each input key-value pair), we can
  - continue to accumulate partial term counts in the associative array across multiple documents,
  - emit key-value pairs only when the mapper has processed all documents

Transmission of intermediate data is deferred until the `Close` method in the pseudo-code

```
class Mapper
    method Initialize
        H ← new AssociativeArray
    method Map(docid a, doc d)
        for all term t ∈ doc d do
            H{t} ← H{t} + 1
    method Close
        for all term t ∈ H do
            Emit(term t, count H{t})
```

IN-MAPPER COMBINING PATTERN: ADVANTAGES

Provides control over when local aggregation occurs and how it exactly takes place

- Hadoop makes no guarantees on how many times the combiner is applied, or that it is even applied at all
- The execution framework has the option of using it, perhaps multiple times, or not at all
- Such indeterminism is unacceptable, which is exactly why programmers often choose to perform their own local aggregation in the mappers

In-mapper combining will typically be more efficient than using actual combiners.

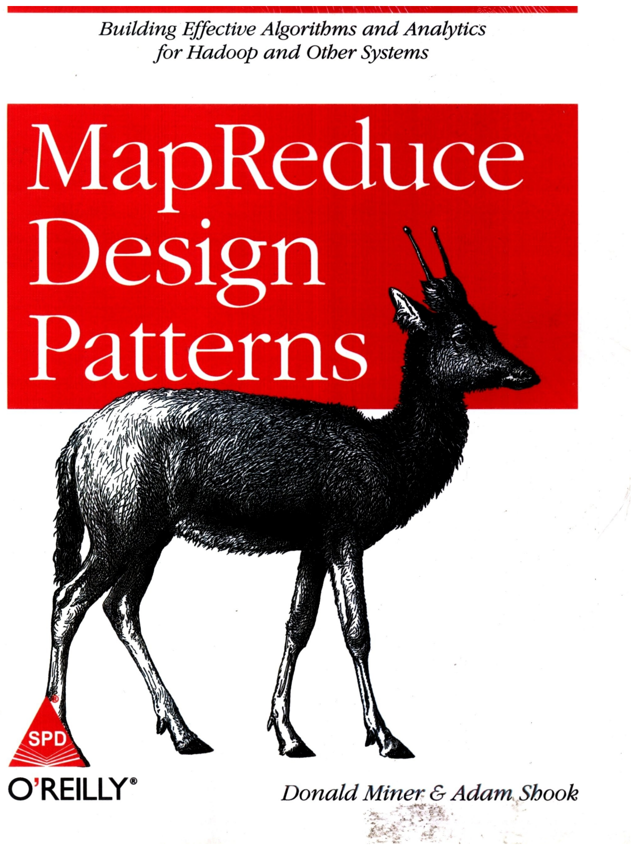
- One reason for this is the additional overhead associated with actually materializing the key-value pairs
  - Combiners reduce the amount of intermediate data that is shuffled across the network, but don't actually reduce the number of key-value pairs that are emitted by the mappers in the first place
  - The mappers will generate only those key-value pairs that need to be shuffled across the network to the reducers
  - Avoid unnecessary object creation and destruction (garbage collection takes time), and, object serialization and deserialization (when intermediate key-value pairs fill the in-memory buffer holding map outputs and need to be temporarily spilled to disk)

IN-MAPPER COMBINING PATTERN: LIMITATIONS

Breaks the functional programming underpinnings of MapReduce, since state is being preserved across multiple input key-value pairs

There is a fundamental scalability bottleneck associated with the in-mapper combining pattern

- It critically depends on having sufficient memory to store intermediate results until the mapper has completely processed all key-value pairs in an input split
- One common solution to limiting memory usage is to “block” input key-value pairs and “flush” in-memory data structures periodically
- Instead of emitting intermediate data only after every key-value pair has been processed, emit partial results after processing every n key-value pairs
  - Implemented with a counter variable that keeps track of the number of input key-value pairs that have been processed
  - The mapper could keep track of its own memory footprint and flush intermediate key-value pairs once memory usage has crossed a certain threshold
  - Memory size empirically determined: difficult due to concurrent access to memory



MAP REDUCE PATTERNS

MapReduce design patterns, O'Relly

MAP – REDUCE DESIGN PATTERNS

Summarization

Numerical
- Minimum, maximum, count, average, median-standard deviation

?

Inverted index
- Wikipedia inverted index

?

Counting with counters
- Count number of records, a small number of unique instances, summations
- Number of users per state

?

Filtering

Filtering
- Closer view of data, tracking event threads, distributed grep, data cleansing, simple random sampling, remove low scoring data

Bloom
- Remove most of nonwatched values, prefiltering data for a set membership check
- Hot list, Hbase query

Top ten
- Outlier analysis, select interesting data, catchy dashbords
- Top ten users by reputation

Distinct
- Deduplicate data, getting distinct values, protecting from inner join explosion
- Distinct user ids

Data organization

Structured to hierarchical
- Prejoining data, preparing data for Hbase or MongoDB
- Post/comment building for StackOverflow, Question/Answer building

Partitioning
- Partitioning users by last access date

Binning
- Binning by Hadoop-related tags

Total order sorting
- Sort users by last visit

Shuffling
- Anonymizing StackOverflow comments

Join

Reduce side join
- Multiple large data sets joined by foreign key
- User – comment join

Reduce side join with bloom filter
- Reputable user – comment join

Replicated join
- Replicated user – comment join

Composite join
- Composite user – comment join

Cartesian product
- Comment comparison

NUMERICAL SUMMARIZATION PATTERN

The numerical summarizations pattern is a general pattern for calculating aggregate statistical values over a data collection

Intent

- Group records together by a key field and calculate a numerical aggregate per group to get a top-level view of the larger data set
- θbe a generic numerical summarization function we wish to execute over some list of values $(v_1, v_2, v_3, \ldots, v_n)$ to find a value λ, i.e. $\lambda = \theta(v_1, v_2, v_3, \ldots, v_n)$. Examples of θ include a minimum, maximum, average, median, and standard deviation

Motivation and applicability

- Group logins by the hour of the day and perform a count of the number of records in each group, group advertisements by types to determine how affective ads are for better targeting
- Dealing with numerical data or counting
- The data can be grouped by specific fields

STRUCTURE

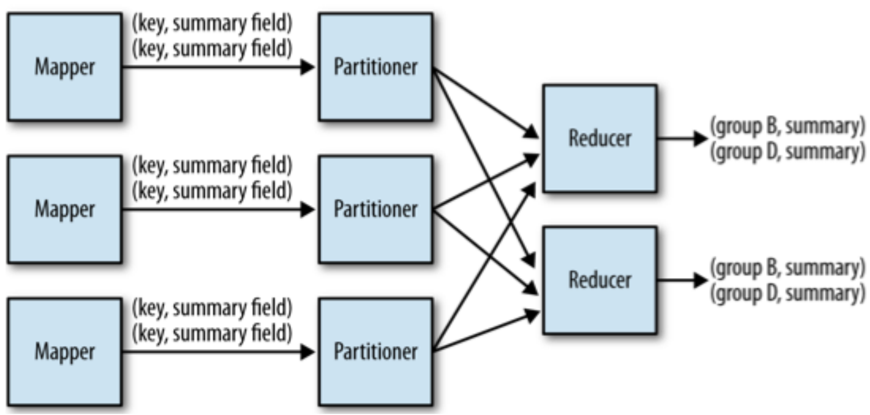
The mapper outputs keys that consist of each field to group by, and values consisting of any pertinent numerical items

The combiner can greatly reduce the number of intermediate key/value pairs to be sent across the network to the reducers for some numerical summarization functions

- If the function θ is an associative and commutative operation, it can be used for this purpose
- If you can arbitrarily change the order of the values and you can group the computation arbitrarily

The reducer

- receives a set of numerical values $(v_1, v_2, v_3, \ldots, v_n)$ associated with a group-by key records to perform the function $\lambda = \theta(v_1, v_2, v_3, \ldots, v_n)$
- The value of λ is output with the given input key



RESEMBLANCES AND PERFORMANCE ANALYSIS

Resemblances

SQL

The Numerical Aggregation pattern is analogous to using aggregates after a GROUP BY in SQL:

```
SELECT MIN(numericalcol1), MAX(numericalcol1),
        COUNT(*) FROM table GROUP BY groupcol2;
```

Pig

The GROUP … BY expression, followed by a FOREACH … GENERATE:

```
b = GROUP a BY groupcol2;
c = FOREACH b GENERATE group, MIN(a.numericalcol1),
        MAX(a.numericalcol1), COUNT_STAR(a);
```

Performance analysis

Aggregations performed by jobs using this pattern typically perform well when the combiner is properly used

These types of operations are what MapReduce was built for

Source: http://indoos.wordpress.com/2010/08/16/hadoop-ecosystem-world-map/

hands
on

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

TODO LIST

QoS based event flow composition
Economy based data delivery
SLA guided data integration
Coordination based parallel data processing
Optimization of different types of queries

NoSQL

Cloud *services*

Data science

Big *data*

Map reduce

Autonomous DaaS

No off the shelf DBMS

Pivot NoSQL data model
Distributed polyglot (big) database engineering
Extended YSCB NoSQL stores benchmark

2009 2011 2013 2014 …

Genoveva Vargas-Solar

CR1, CNRS, LIG-LAFMIA

Genoveva.Vargas@imag.fr

http://vargas-solar.com/datascience

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

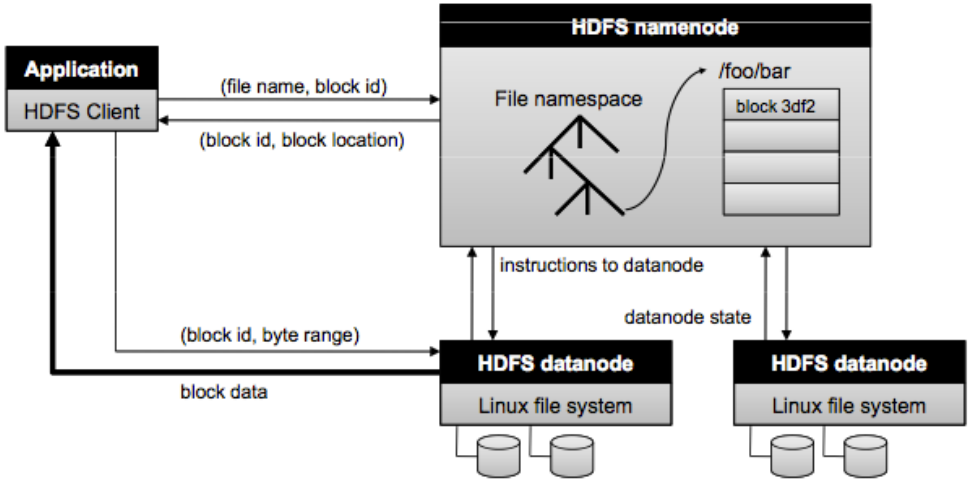
HDFS GENERAL ARCHITECTURE

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

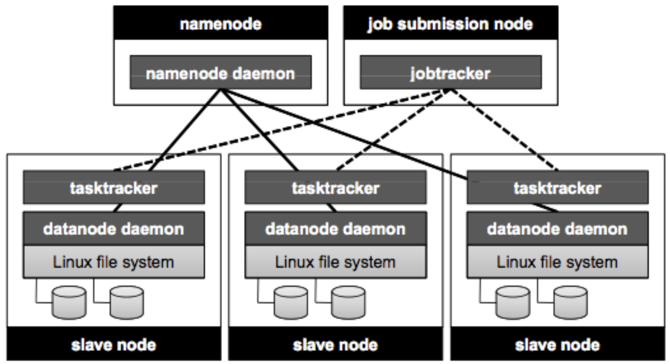
The `namenode` returns the relevant `block id` and the `location` where the block is held (i.e., which `datanode`)

The client then contacts the `datanode` t 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
{name: "mongo", type:"DB"}
apache CouchDB relax
riak
Cassandra
APACHE HBASE
Neo4j NOSQL for the Enterprise
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

Fault-tolerant partitioning

Availability

Consistency

CAP theorem[1]: a system can have two of the three properties

NoSQL systems sacrifice **consistency**

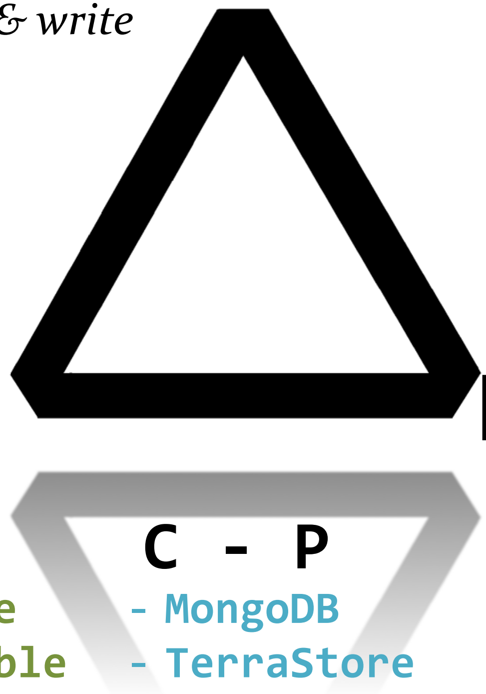
1 Eric Brewer, "Towards robust distributed systems." PODC. 2000 http://www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf

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

DATA MODELS

Tuple

- Row in a relational table, where attributes are pre-defined in a schema, and the values are scalar

Document

- Allows values to be nested documents or lists, as well as scalar values.
- Attributes are not defined in a global schema

Extensible record

- Hybrid between tuple and document, where families of attributes are defined in a schema, but new attributes can be added on a per-record basis

DATA STORES

Key-value
- Systems that store values and an index to find them, based on a key

Document
- Systems that store documents, providing index and simple query mechanisms

Extensible record
- Systems that store extensible records that can be partitioned vertically and horizontally across nodes

Graph
- Systems that store model data as graphs where nodes can represent content modelled as document or key-value structures and arcs represent a relation between the data modelled by the node

Relational
- Systems that store, index and query tuples

KEY-VALUE STORES

"Simplest data stores" use a data model similar to the memcached distributed in-memory cache

Single key-value index for all data

Provide a persistence mechanism

Replication, versioning, locking, transactions, sorting

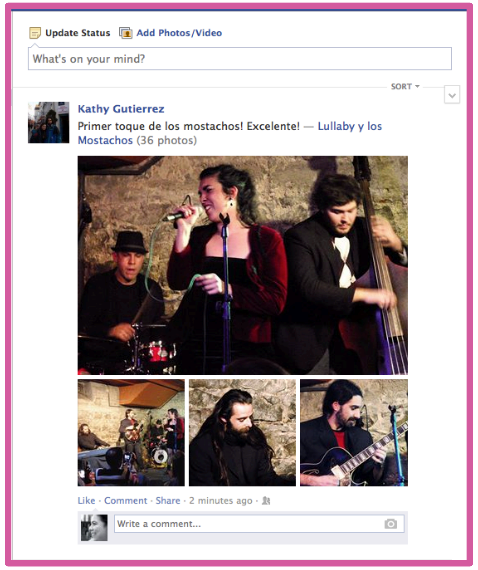
API: inserts, deletes, index lookups

No secondary indices or keys

System	Address
Redis	code.google.com/p/redis
Scalaris	code.google.com/p/scalaris
Tokyo	tokyocabinet.sourceforge.net
Voldemort	project-voldemort.com
Riak	riak.basho.com
Membrain	schoonerinfotech.com/products
Membase	membase.com

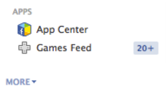

```
SELECT  name, pic, profile_url
FROM    user
WHERE   uid = me()
```

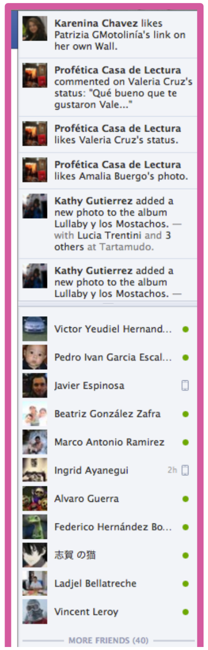

```
SELECT  message, attachment
FROM    stream
WHERE   source_id = me() AND type = 80
```




```
SELECT  name
FROM    friendlist
WHERE   owner = me()
```


```
SELECT  name
FROM    group
WHERE   gid IN ( SELECT  gid
                 FROM    group_member
                 WHERE   uid = me() )
```

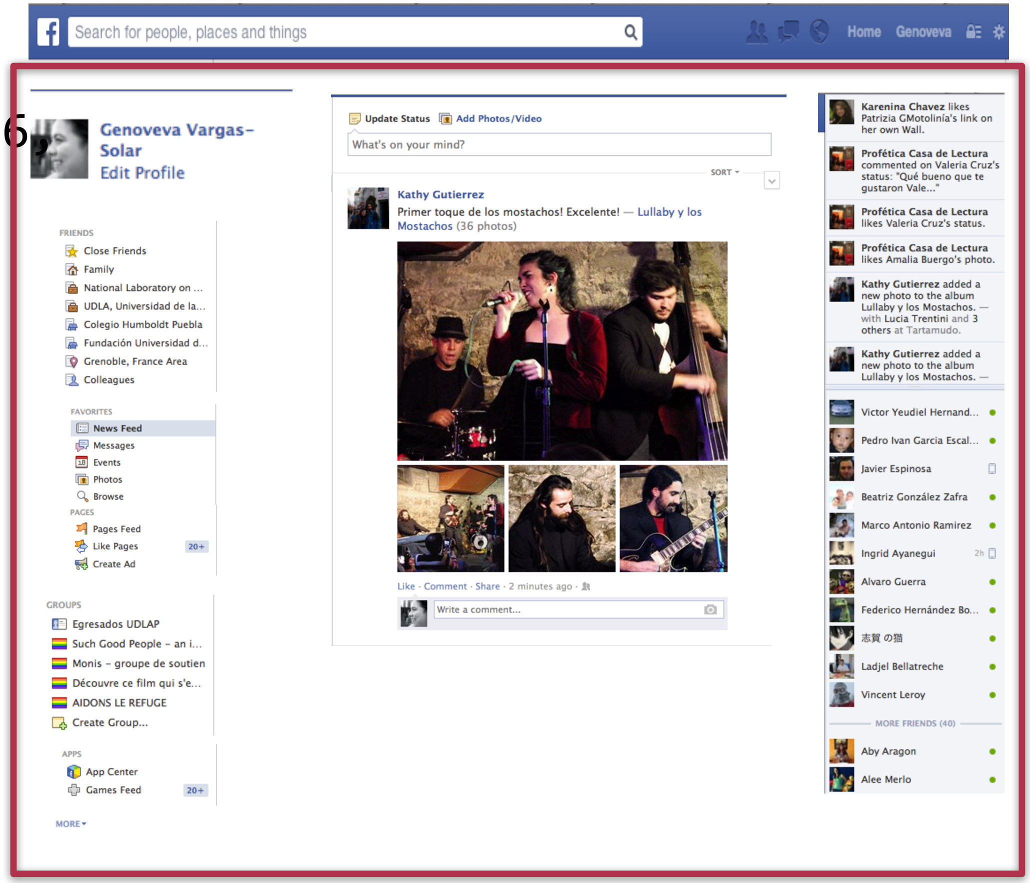





```
SELECT  name, pic
FROM    user
WHERE   online_presence = "active"
        AND
        uid IN ( SELECT  uid2
                 FROM    friend
                 WHERE   uid1 = me() )
```

https://developers.facebook.com/docs/reference/fql/

<805114856,



Search for people, places and things
Home
Genoveva
Genoveva Vargas-Solar
Edit Profile
FRIENDS
Close Friends
Family
National Laboratory on ...
UDLA, Universidad de la...
Colegio Humboldt Puebla
Fundación Universidad d...
Grenoble, France Area
Colleagues
FAVORITES
News Feed
Messages
Events
Photos
Browse
PAGES
Pages Feed
Like Pages
Create Ad
GROUPS
Egresados UDLAP
Such Good People - an i...
Monis - groupe de soutien
Découvre ce film qui s'e...
AIDONS LE REFUGE
Create Group...
APPS
App Center
Games Feed
MORE
Update Status
Add Photos/Video
What's on your mind?
SORT
Kathy Gutierrez
Primer toque de los mostachos! Excelente! — Lullaby y los Mostachos (36 photos)
Like · Comment · Share · 2 minutes ago
Write a comment...
Karenina Chavez likes Patrizia GMotolinía's link on her own Wall.
Profética Casa de Lectura commented on Valeria Cruz's status: "Qué bueno que te gustaron Vale..."
Profética Casa de Lectura likes Valeria Cruz's status.
Profética Casa de Lectura likes Amalia Buergo's photo.
Kathy Gutierrez added a new photo to the album Lullaby y los Mostachos. — with Lucia Trentini and 3 others at Tartamudo.
Kathy Gutierrez added a new photo to the album Lullaby y los Mostachos. —
Victor Yeudiel Hernand...
Pedro Ivan Garcia Escal...
Javier Espinosa
Beatriz González Zafra
Marco Antonio Ramirez
Ingrid Ayanegui
Alvaro Guerra
Federico Hernández Bo...
Ladjel Bellatreche
Vincent Leroy
MORE FRIENDS (40)
Aby Aragon
Alee Merlo

DOCUMENT STORES

Support more complex data: pointerless objects, i.e., documents

Secondary indexes, multiple types of documents (objects) per database, nested documents and lists, e.g. B-trees

Automatic sharding (scale writes), no explicit locks, weaker concurrency (eventual for scaling reads) and atomicity properties

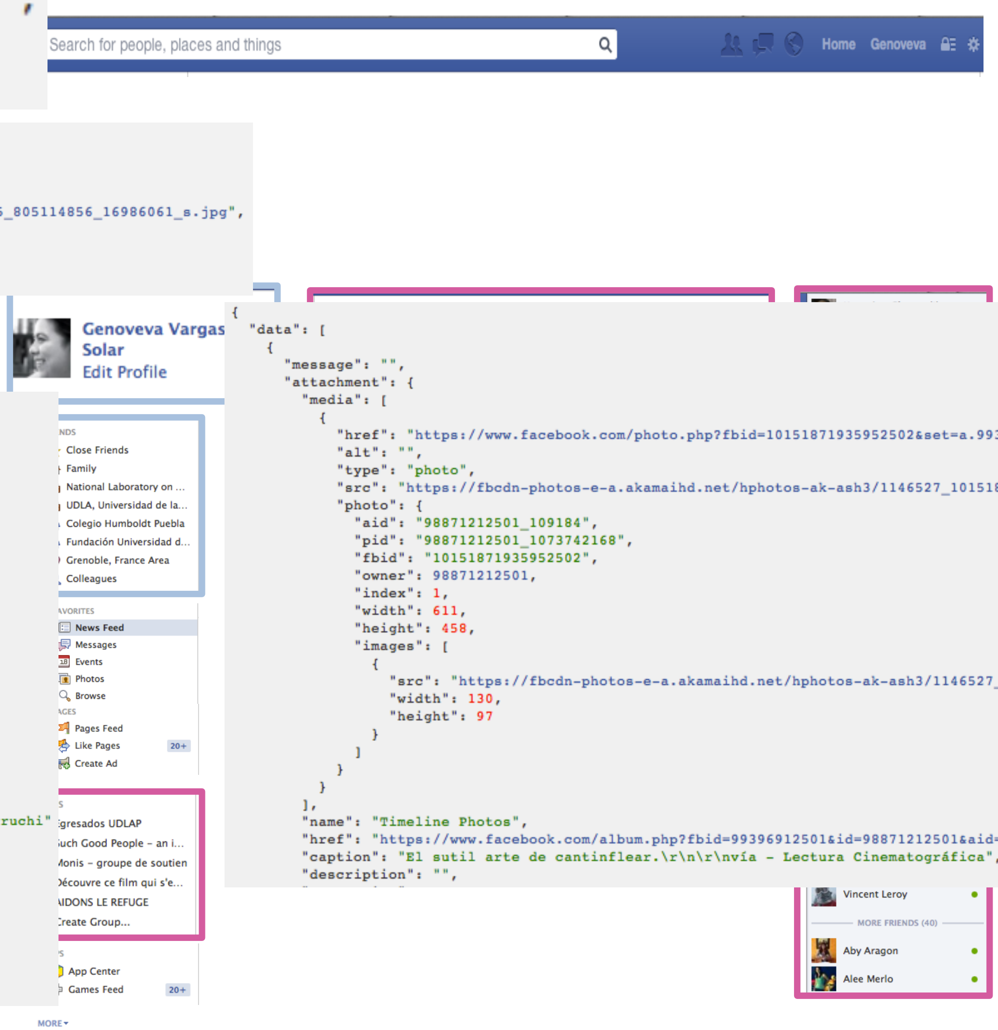
API: `select`, `delete`, `getAttributes`, `putAttributes` on documents

Queries can be distributed in parallel over multiple nodes using a map-reduce mechanism

System	Address
SimpleDB	`amazon.com/simpledb`
Couch DB	`couchdb.apache.org`
Mongo DB	`mongodb.org`
Terrastore	`code.google.com/terrastore`

```
{
  "name": "Genoveva Vargas-Solar",
  "id": "805114856"
}
```

```
{
  "data": [
    {
      "name": "Genoveva Vargas-Solar",
      "pic": "https://fbcdn-profile-a.akamaihd.net/hprofile-ak-ash4/275915_805114856_16986061_s.jpg",
      "profile_url": "https://www.facebook.com/genoveva.vargas"
    }
  ]
}
```


```
{
  "data": [
    {
      "name": "$$$ Se Vende Jeep Compass 2008 - 60,000kms. $$$"
    },
    {
      "name": "Découvre ce film qui s'engage pour le mariage pour tous"
    },
    {
      "name": "emepink"
    },
    {
      "name": "Such Good People - an indie screwball comedy"
    },
    {
      "name": "Comunidad Mexicana de Tecnologías Semánticas"
    },
    {
      "name": "TI-502 Administración de Datos"
    },
    {
      "name": "exaUDLAP Sistemas Computacionales"
    },
    {
      "name": "\"Hombre Nuevo\" artículos de valores humanos del P. Otaolaurruchi"
    },
    {
      "name": "LACCIR"
    },
    {
      "name": "Monis - groupe de soutien"
    },
    {
      "name": "Red Temática de las TIC"
    },
```

```
{
  "data": [
    {
      "message": "",
      "attachment": {
        "media": [
          {
            "href": "https://www.facebook.com/photo.php?fbid=10151871935952502&set=a.99396912501.109184.98871212501&type=1",
            "alt": "",
            "type": "photo",
            "src": "https://fbcdn-photos-e-a.akamaihd.net/hphotos-ak-ash3/1146527_10151871935952502_258686255_s.jpg",
            "photo": {
              "aid": "98871212501_109184",
              "pid": "98871212501_1073742168",
              "fbid": "10151871935952502",
              "owner": 98871212501,
              "index": 1,
              "width": 611,
              "height": 458,
              "images": [
                {
                  "src": "https://fbcdn-photos-e-a.akamaihd.net/hphotos-ak-ash3/1146527_10151871935952502_258686255_s.jpg",
                  "width": 130,
                  "height": 97
                }
              ]
            }
          }
        ],
        "name": "Timeline Photos",
        "href": "https://www.facebook.com/album.php?fbid=99396912501&id=98871212501&aid=109184",
        "caption": "El sutil arte de cantinflear.\r\n\r\nvía - Lectura Cinematográfica",
        "description": "",
```

EXTENSIBLE RECORD STORES

Basic data model is rows and columns

Basic scalability model is splitting rows and columns over multiple nodes

- Rows split across nodes through sharding on the primary key
  - Split by range rather than hash function
  - Rows analogous to documents: variable number of attributes, attribute names must be unique
  - Grouped into collections (tables)
  - Queries on ranges of values do not go to every node

Columns are distributed over multiple nodes using "column groups"

- Which columns are best stored together
- Column groups must be pre-defined with the extensible record stores

SYSTEM	ADDRESS
HBase	`hbase.apache.com`
HyperTable	`hypertable.org`
Cassandra	`incubator.apache.org/cassandra`

SCALABLE RELATIONAL SYSTEMS

SQL: rich declarative query language

Databases reinforce referential integrity

ACID semantics

Well understood operations:

- Configuration, Care and feeding, Backups, Tuning, Failure and recovery, Performance characteristics

Use small-scope operations

- Challenge: joins that do not scale with sharding

Use small-scope transactions

- ACID transactions inefficient with communication and 2PC overhead

Shared nothing architecture for scalability

Avoid cross-node operations

System	Address
MySQL C	`mysql.com/cluster`
Volt DB	`voltdb.com`
Clustrix	`clustrix.com`
ScaleDB	`scaledb.com`
Scale Base	`scalebase.com`
Nimbus DB	`nimbusdb.com`

NOSQL STORES CHARACTERISTICS

There is no standard definition of what NoSQL means. The term began with a workshop organized in 2009, but there is much argument about what databases can truly be called NoSQL.

But while there is no formal definition, there are some common characteristics of NoSQL databases

- they don't use the relational data model, and thus don't use the SQL language
- they tend to be designed to run on a cluster
- they tend to be Open Source
- they don't have a fixed schema, allowing you to store any data in any record

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

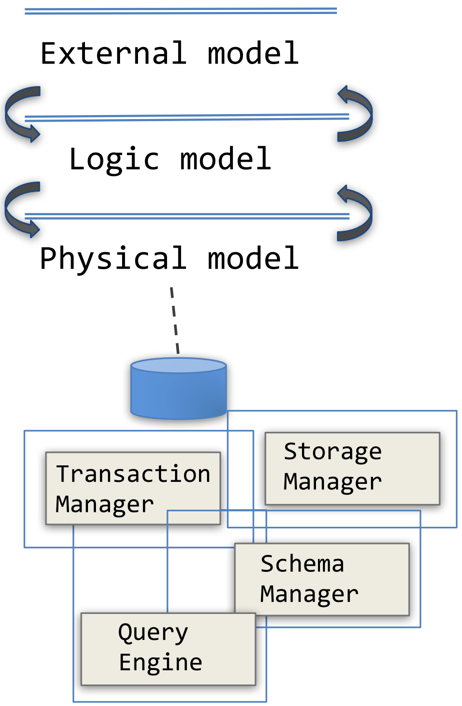
Horizontal scalability
- Distribute data and operations over many servers
- Replicate and distribute data over many servers
- No shared memory or disk

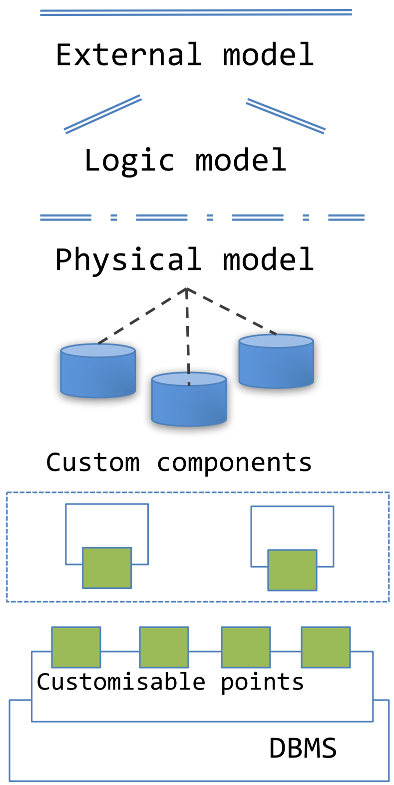
High performance
- Efficient use of distributed indexes and RAM for data storage
- Weak consistency model
- Limited transactions

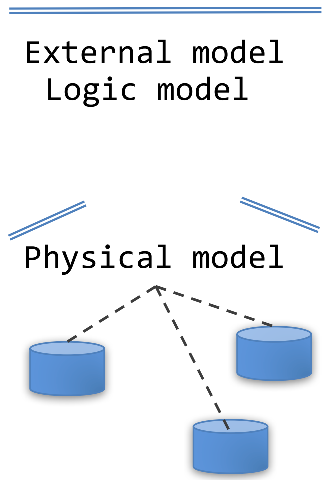
Next generation databases mostly addressing some of the points: being **non-relational, distributed, open-source** and **horizontally scalable** [http://nosql-database.org]

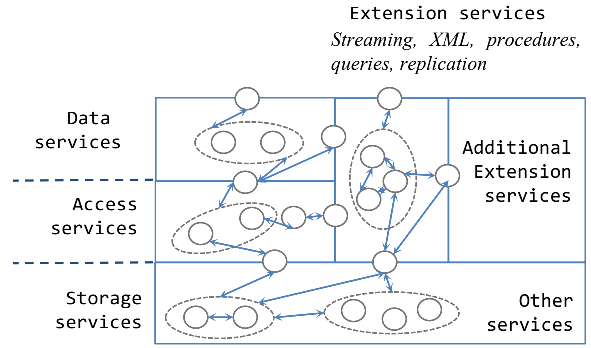
DATA MANAGEMENT SYSTEMS ARCHITECTURES

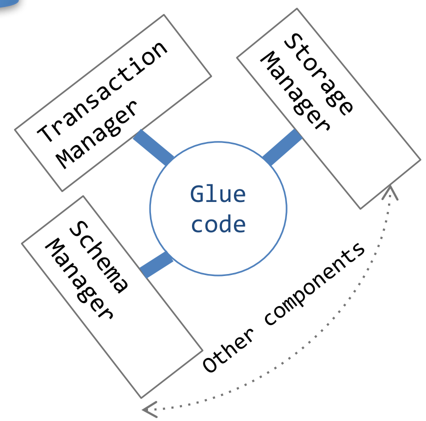
ANSI/SPARC











COMPARING NOSQL & NEWSQL SYSTEMS

Key-Value

System	Concurrency control	Data storage	Replication	Transaction
Redis	Locks	RAM	Asynchronous	No
Scalaris	Locks	RAM	Synchronous	Local
Tokyo	Locks	RAM/Disk	Asynchronous	Local
Voldemort	MVCC	RAM/BDB	Asynchronous	No
Riak	MVCC	Plug in	Asynchronous	No
Membrain	Locks	Flash+Disk	Synchronous	Local
Membase	Locks	Disk	Synchronous	Local

Document

System	Concurrency control	Data storage	Replication	Transaction
Dynamo	MVCC	Plug in	Asynchronous	No
SimpleDB	Non	S3	Asynchronous	No
MongoDB	Locks	Disk	Asynchronous	No
CouchDB	MVCC	Disk	Asynchronous	No

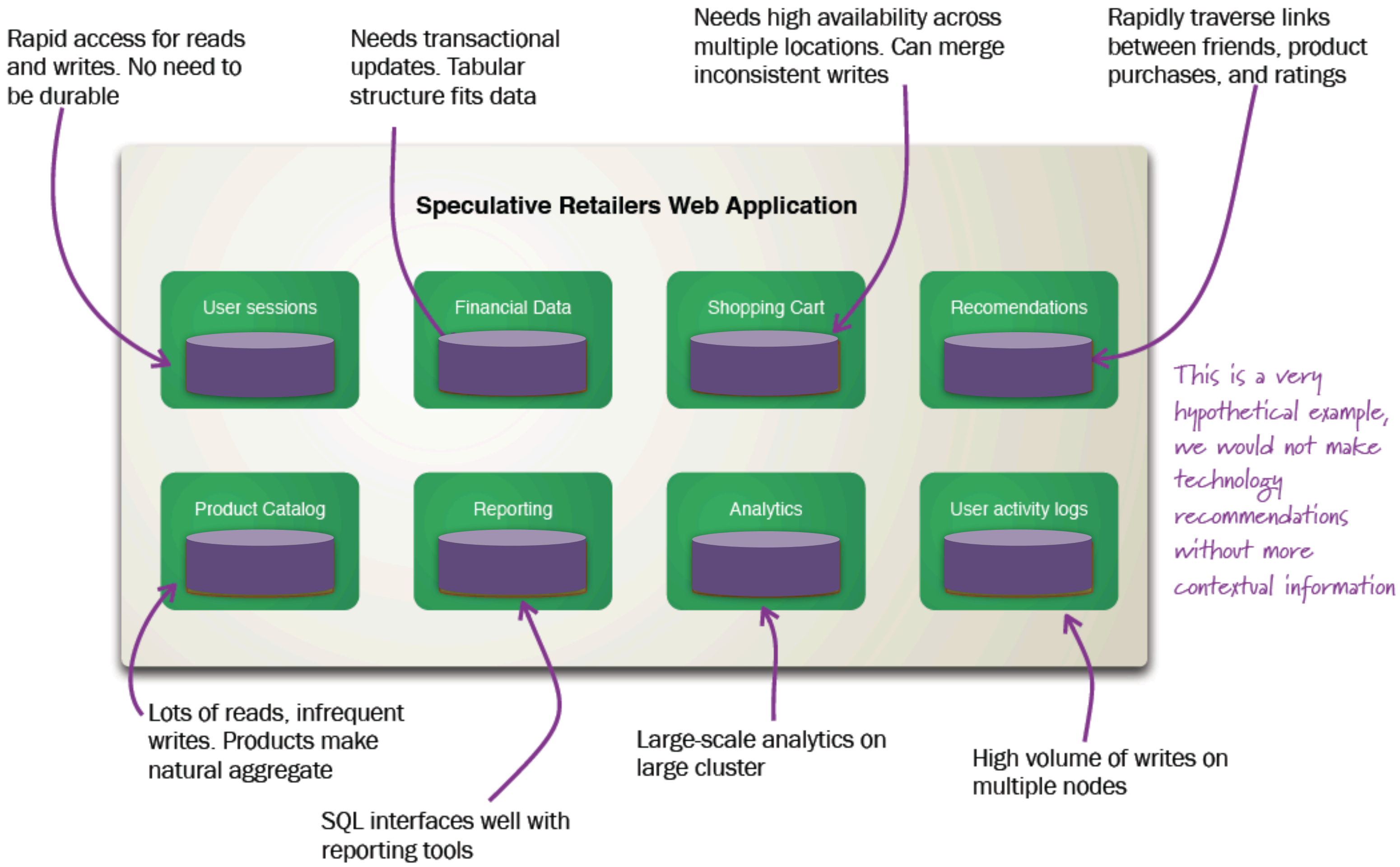
Extended records

System	Concurrency control	Data storage	Replication	Transaction
Terrastore	Locks	RAM+	Synchronous	L
Hbase	Locks	HADOOP	Asynchronous	L
HyperTable	Locks	Files	Synchronous	L
Cassandra	MVCC	Disk	Asynchronous	L
BigTable	Locs+stamps	GFS	Both	L

Relational

System	Concurrency control	Data storage	Replication	Transaction
PNuts	MVCC	Disk	Asynchronous	L
MySQL-C	ACID	Disk	Synchronous	Y
VoltDB	ACID/no Lock	RAM	Synchronous	Y
Clustrix	ACID/no Lock	Disk	Synchronous	Y
ScaleDB	ACID	Disk	Synchronous	Y
ScaleBase	ACID	Disk	Asynchronous	Y
NimbusDB	ACID/no Lock	Disk	Synchronous	Y

Cattell, Rick. "Scalable SQL and NoSQL data stores." ACM SIGMOD Record 39.4 (2011): 12-27


Rapid access for reads and writes. No need to be durable
Needs transactional updates. Tabular structure fits data
Needs high availability across multiple locations. Can merge inconsistent writes
Rapidly traverse links between friends, product purchases, and ratings
Speculative Retailers Web Application
User sessions
Financial Data
Shopping Cart
Recomendations
Product Catalog
Reporting
Analytics
User activity logs
This is a very hypothetical example, we would not make technology recommendations without more contextual information
Lots of reads, infrequent writes. Products make natural aggregate
SQL interfaces well with reporting tools
Large-scale analytics on large cluster
High volume of writes on multiple nodes

THIS TALK IS **NOT** ABOUT

NOSQL

Your Ultimate Guide to the Non-Relational Universe!

[including a **historic** Archive 2009-2011] **News Feed** covering some changes here **!**

http://nosql-database.org

Debate on whether NoSQL stores and relational systems are better or worse …

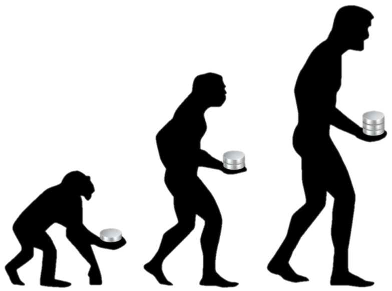
that is not the point

Of course we can surf on these waves at the end of the talk and during EDBT School!

THIS TALK **IS** ABOUT

Polyglot Persistence

alternative for managing multiform and multimedia data collections
according to different properties and requirements



Polyglot Persistence

using multiple data storage technologies, chosen based upon the way data is being used by individual applications. Why store binary images in relational database, when there are better storage systems?

Polyglot persistence will occur over the enterprise as different applications use different data storage technologies. It will also occur within a single application as different parts of an application's data store have different access characteristics.

http://martinfowler.com/bliki/PolyglotPersistence.html

POLYGLOT PERSISTENCE

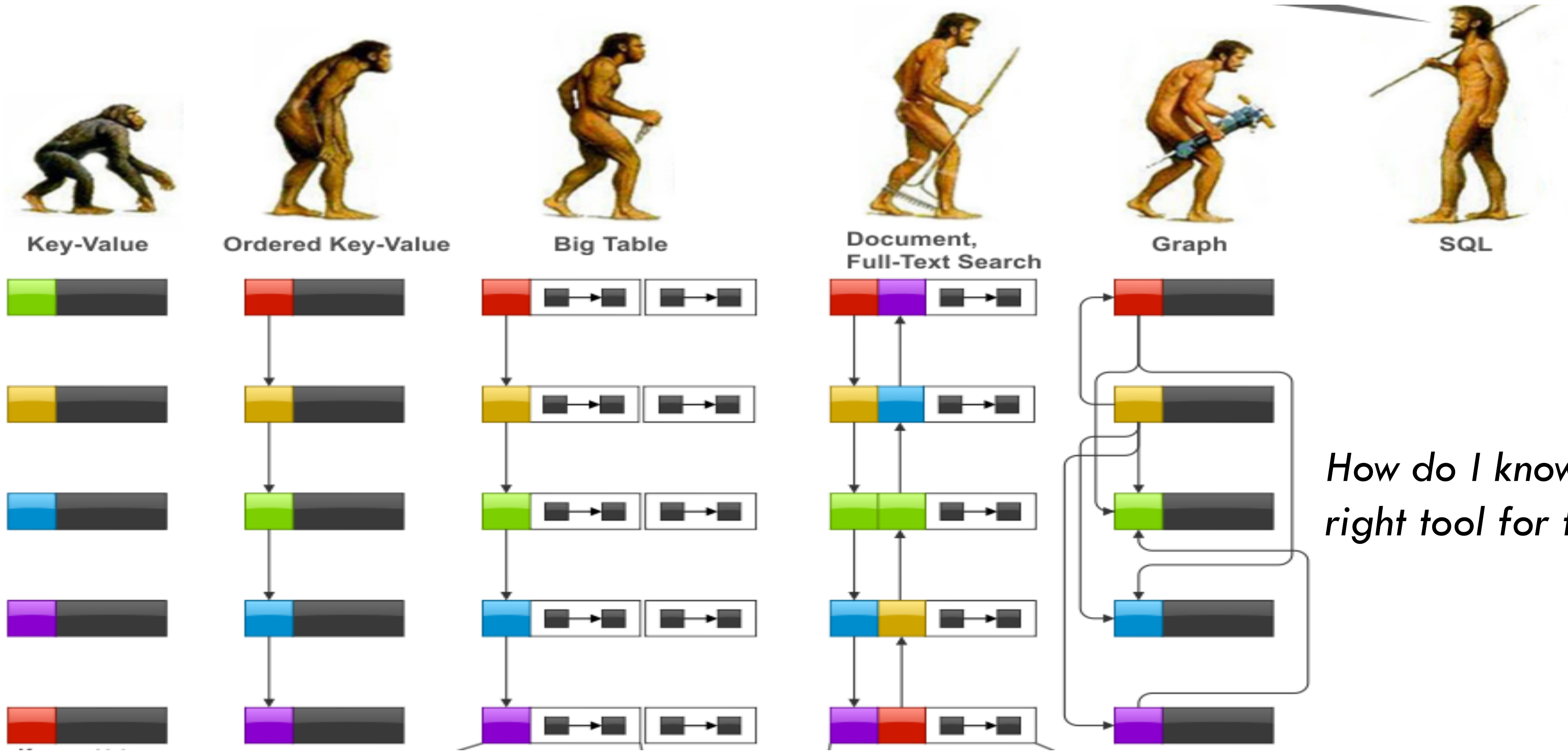
Polyglot Programming: applications should be written in a mix of languages to take advantage of different languages are suitable for tackling different problems

Polyglot persistence: any decent sized enterprise will have a variety of different data storage technologies for different kinds of data

- a new strategic enterprise application should no longer be built assuming a relational persistence support
- the relational option might be the right one - but you should seriously look at other alternatives

M. Fowler and P. Sadalage. *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*. Pearson Education, Limited, 2012

Use the right tool for the right job…



How do I know which is the right tool for the right job?

(Katsov-2012)

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

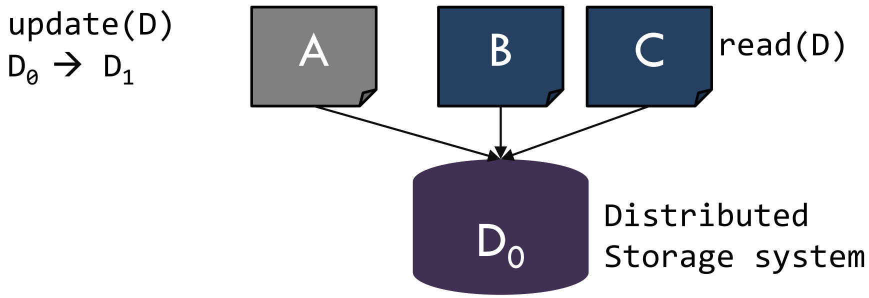
ACID semantics (transaction semantics in RDBMS)

- **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

Strong consistency for transactions highest priority

Availability less important

Pessimistic

Rigorous analysis

Complex mechanisms

BASE

Availability and scaling highest priorities

Weak consistency

Optimistic

Best effort

Simple and fast

MAP-REDUCE

Programming model for expressing distributed computations on massive amounts of data

Execution framework for large-scale data processing on clusters of commodity servers

Market: any organization built around gathering, analyzing, monitoring, filtering, searching, or organizing content must tackle large-data problems

- data- intensive processing is beyond the capability of any individual machine and requires clusters
- large-data problems are fundamentally about ***organizing computations on dozens, hundreds, or even thousands of machines***

MAP REDUCE JOB

map() → reduce()

sub-divide & conquer — *combine & reduce cardinality*

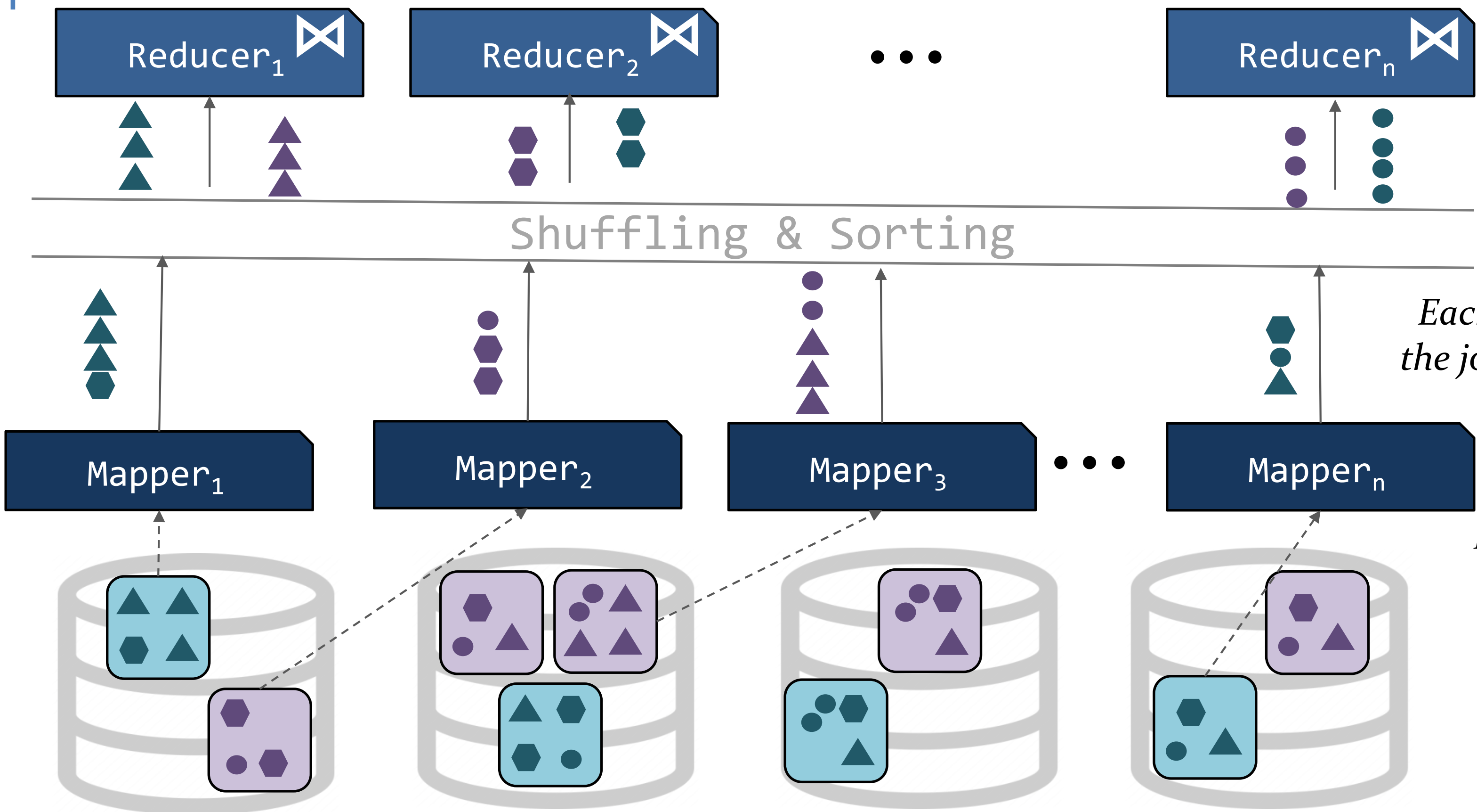
Stage 1: Apply a user-specified computation over all input records in a dataset.

- These operations occur in parallel and yield intermediate output (**key-value** pairs)

Stage 2: Aggregate intermediate output by another user-specified computation

- Recursively applies a function on every pair of the list

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 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

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

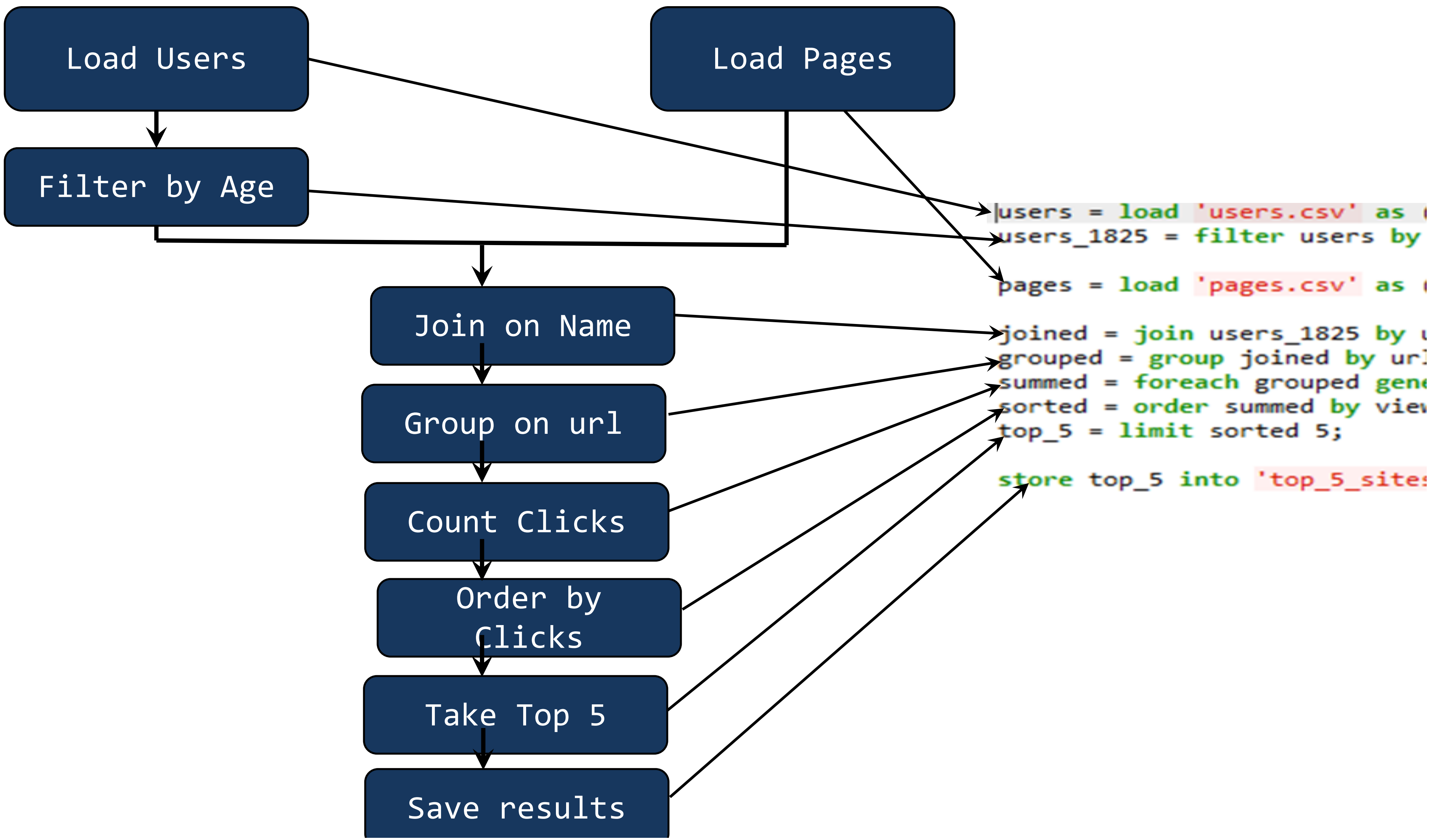
Top 5 pages accessed by users between 18 and 25 year

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

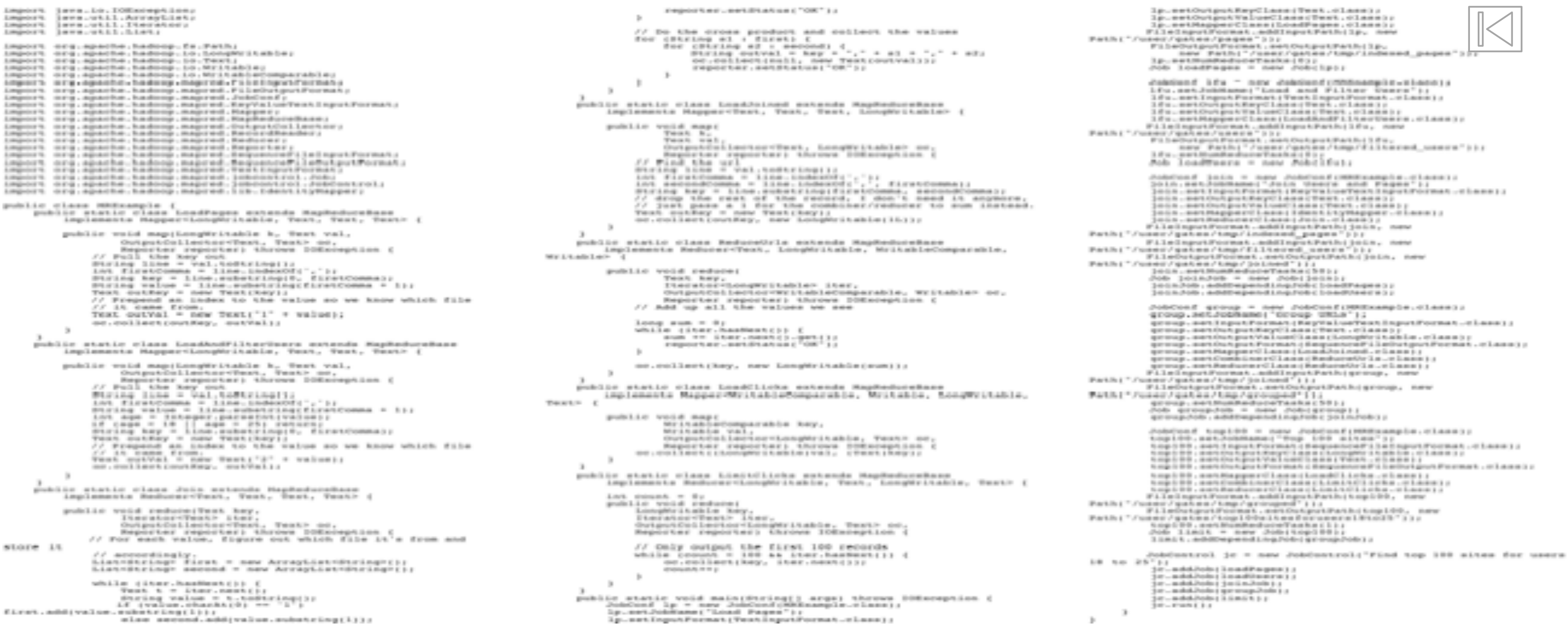
File Edit Selection Find View Goto Tools Project Preferences Help

top_5_sites.pig

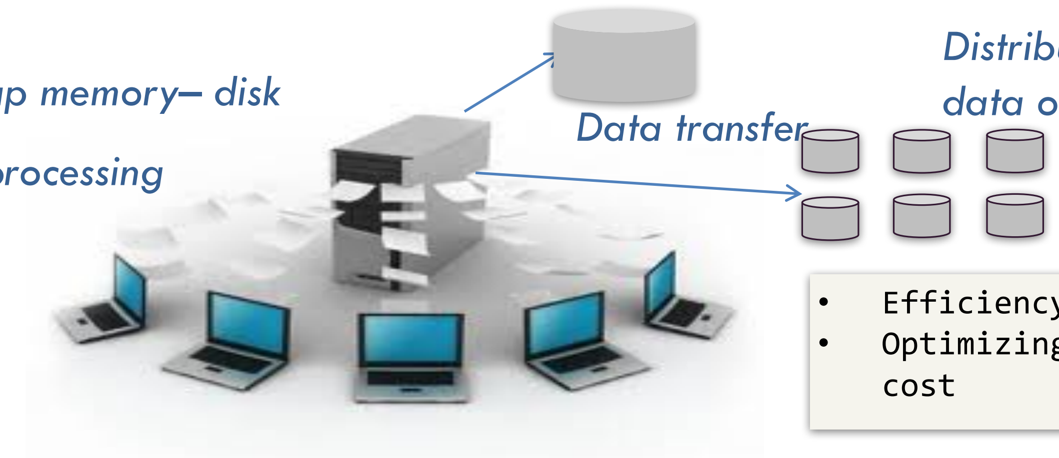
```
users = load 'users.csv' as (username:chararray, age:int);
users_1825 = filter users by age >= 18 and age <= 25;

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_sites

EQUIVALENT JAVA MAP REDUCE CODE



QUERYING **WITH** RESOURCES CONSTRAINTS



- Efficiency => time cost
- Optimizing memory and computing cost

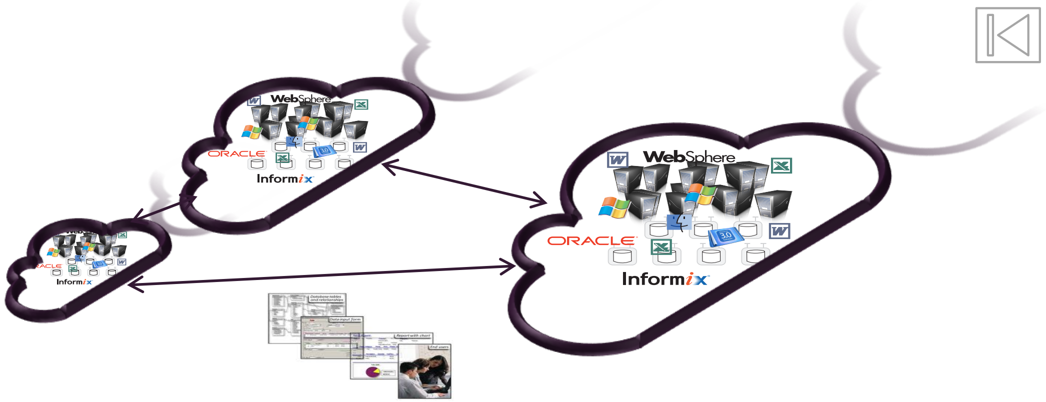
Q1: Which are the most popular products at Starbucks ?

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

Genoveva Vargas-Solar

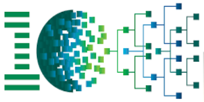
CR1, CNRS, LIG-LAFMIA

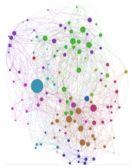
Genoveva.Vargas@imag.fr

http://vargas-solar.com/datascience

MULTIMODEL DATA MANAGEMENT

Greedy data processing

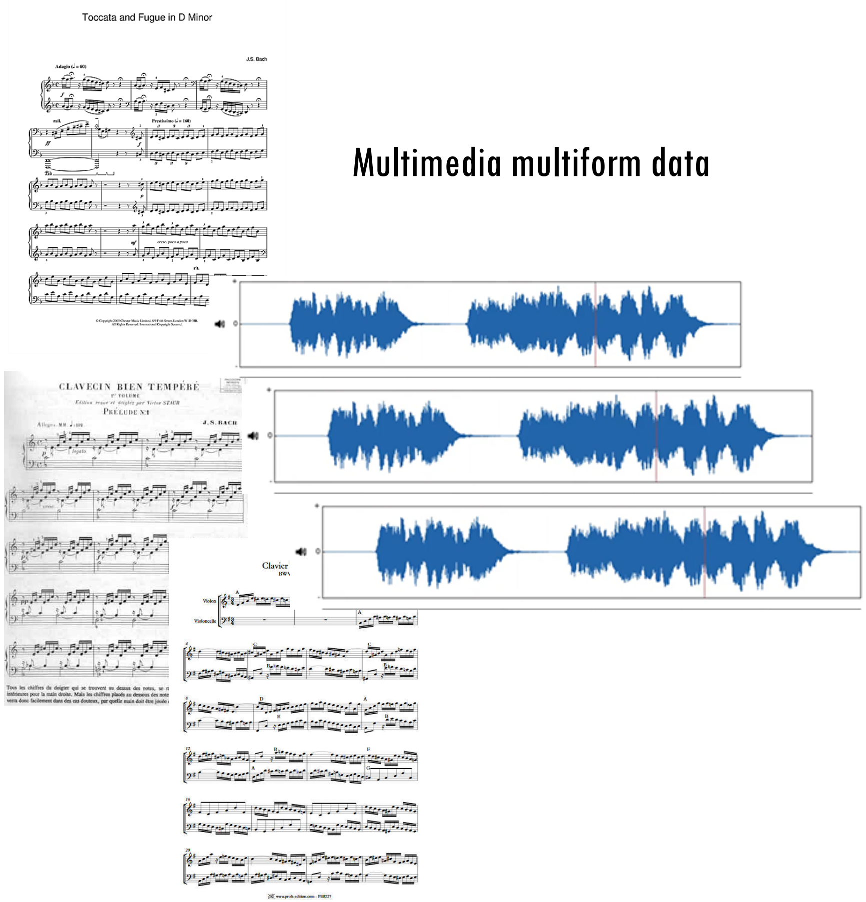




Provide data storage, fetching and delivery strategies

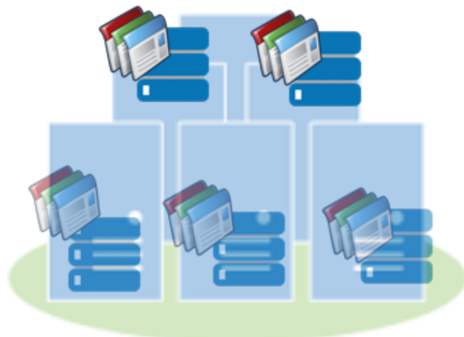
- Architecture: distributed file system across nodes
- Data sharding and replication: on storage and memory
- Fetch to fulfil multi-facetig application requirements
  - Prefetching
  - Memory indexing
  - Reduce impedance mismatch

DATA SHARDING



Multimedia multiform data

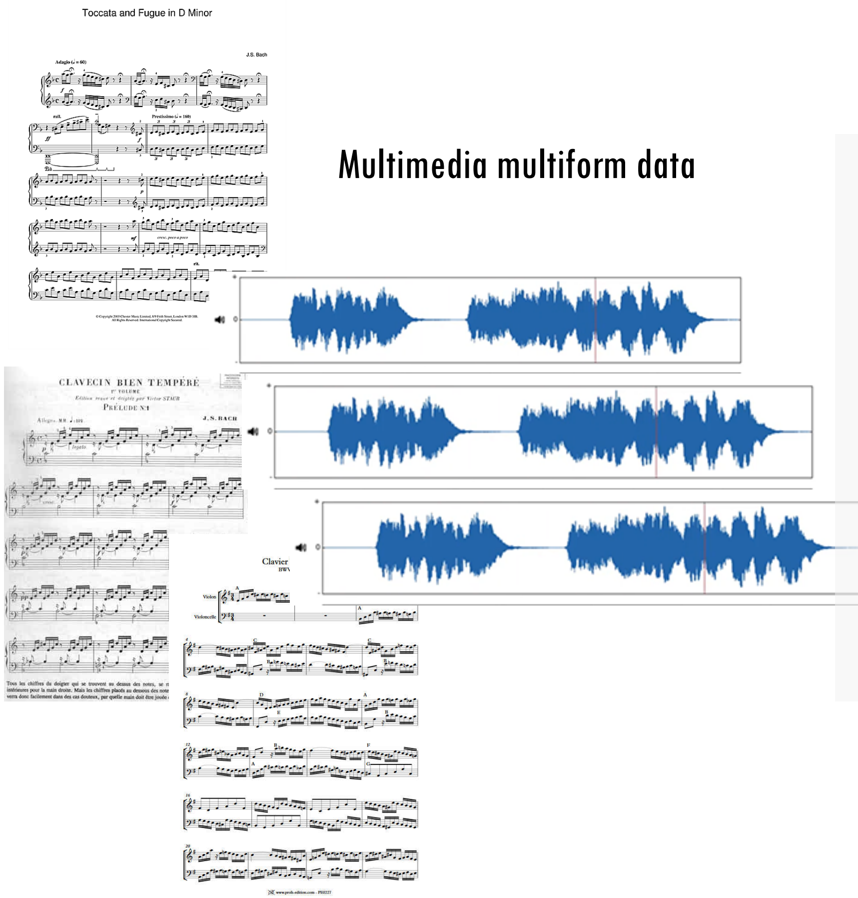
Distributed File System



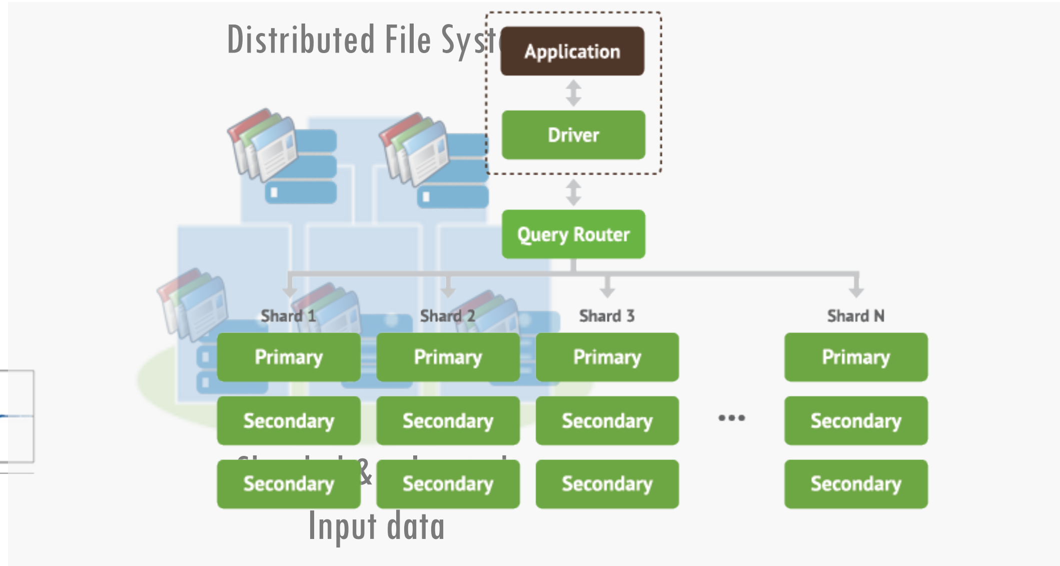
Sharded & colocated
Input data

DATA SHARDING

Multimedia multiform data



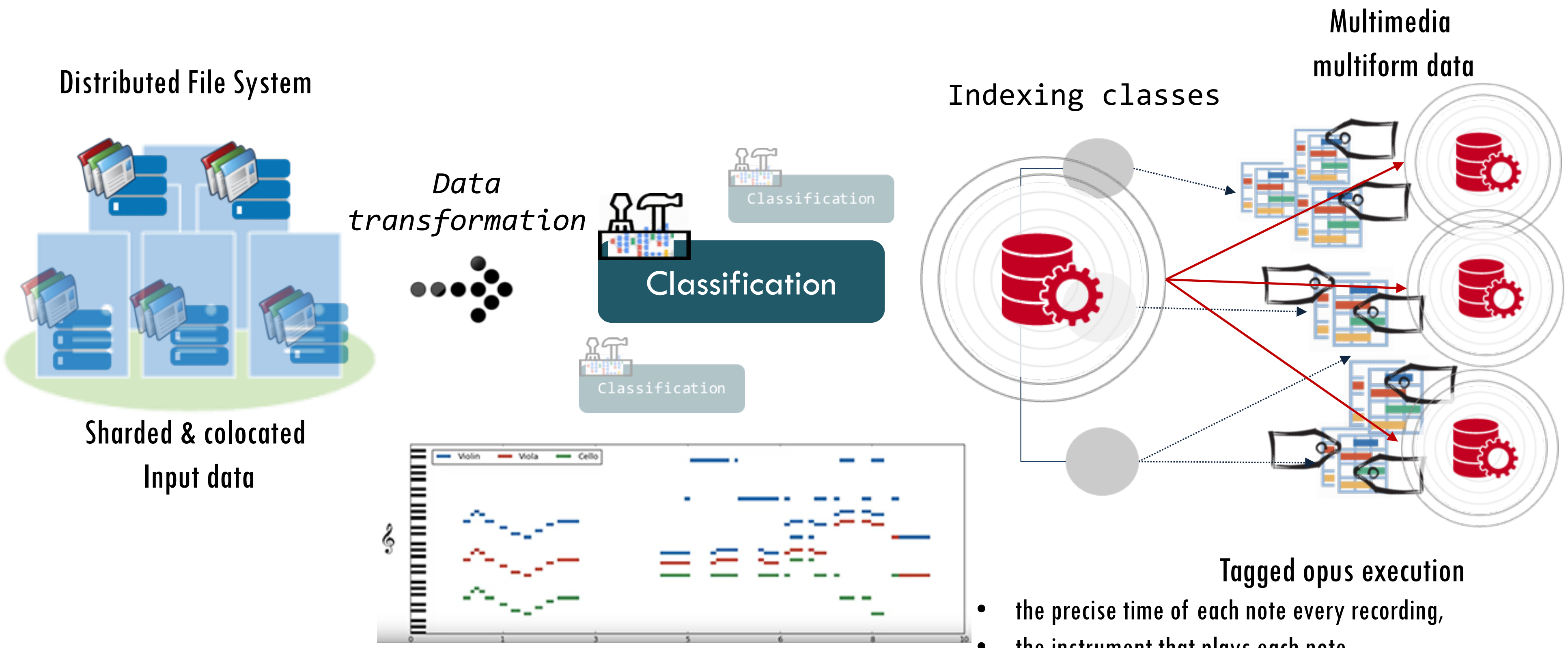
Sharded data architecture



Factors:

- RAM
- CPU
- Disk
- Network

INDEXING & STORING

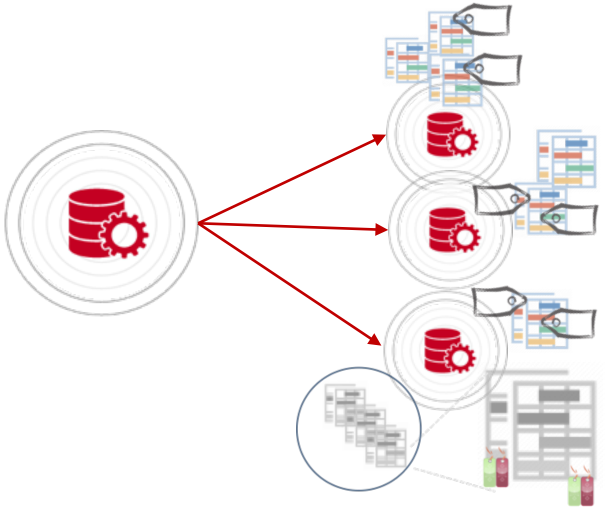


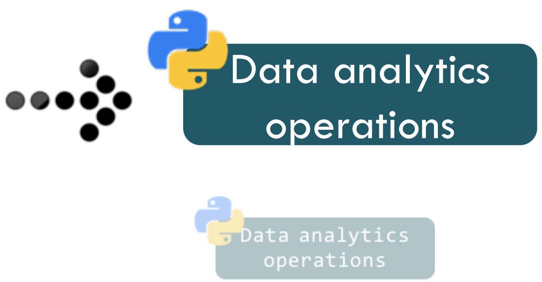
Tagged opus execution

- the precise time of each note every recording,
- the instrument that plays each note,
- the note's position in the metrical structure of the composition

MusicNet: 330 classical music recordings, 1 million annotated labels indicating http://homes.cs.washington.edu/~thickstn/musicnet.html

LOADING





Music information retrieval

- Automatic music transcription
- Inferring a musical score from a recording

Generative models that can fabricate performances under various constraints

- Can we learn to synthesize a performance given a score?
- Can we generate a fugue in the style of Bach using a melody by Brahms?

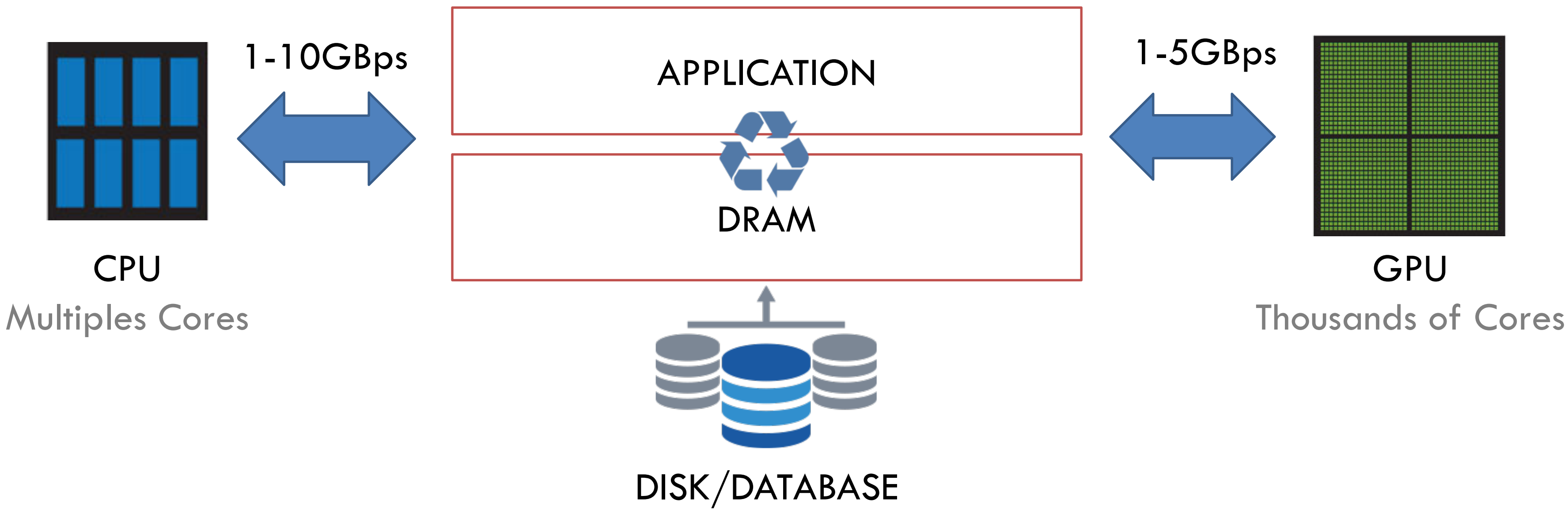
- Identify the *notes* performed at specific times in a recording
- Classify the *instruments* that perform in a recording
- Classify the *composer* of a recording
- Identify precise *onset* times of the notes in a recording
- Predict the *next note* in a recording, conditioned on history

GREEDY DATA PROCESSING

"Multi-view computational problem"

Iterative data processing and visualization tasks need to share CPU cycles

Data is a bottleneck



what can go wrong?

not enough space to index all data

not enough idle time to finish proper tuning

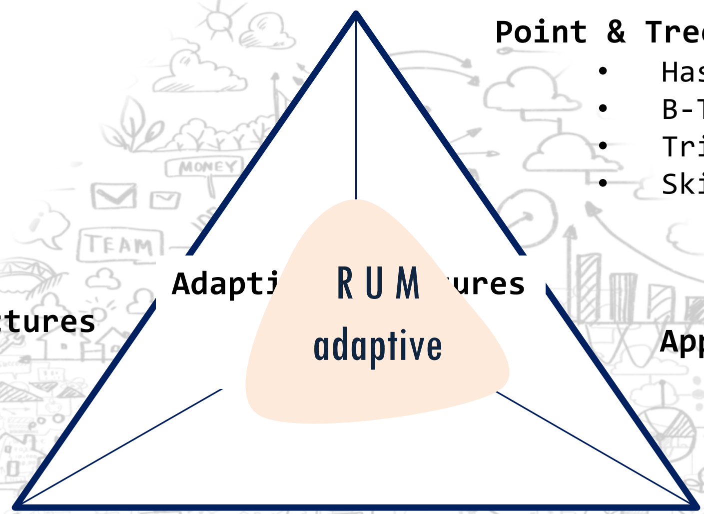
by the time we finish tuning, the workload changes

not enough money - energy - resources

ACCESS METHODS

Read Optimized

Point & Tree indexes
- Hash
- B-Tree
- Trie
- Skiplist



Differential structures
- PDT[1]
- LSM
- PBT
- MaSM

Approximate indexes
- Sparse index
- Bloom filter
- Bitmap

Update Optimized (write)

Memory Optimized (space)

DATA

Requirements

Operations

Hardware

Predefined Data Types, Log-structured Merge Tree, the Partitioned B-tree, the Materialized Sort-Merge algorithm

CHALLENGES & OBJECTIVE

How to combine, deploy, and deliver data management 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**