Intro

Bigdata/SQL

Questions: https://app.sli.do/event/rak9o5wd

Andrei Arion, LesFurets.com, tp-bigdata@lesfurets.com

Plan

1. **Course info**
2. From SQL to NoSQL
3. Scaling MySQL @LesFurets.com
4. TP: PostgreSQL

Andrei ARION

- XML databases research, INRIA & UPS
- software engineer/consultant
- engineer, data team, ***LesFurets.com***

tp-bigdata@lesfurets.com,
andrei.arion@gmail.com

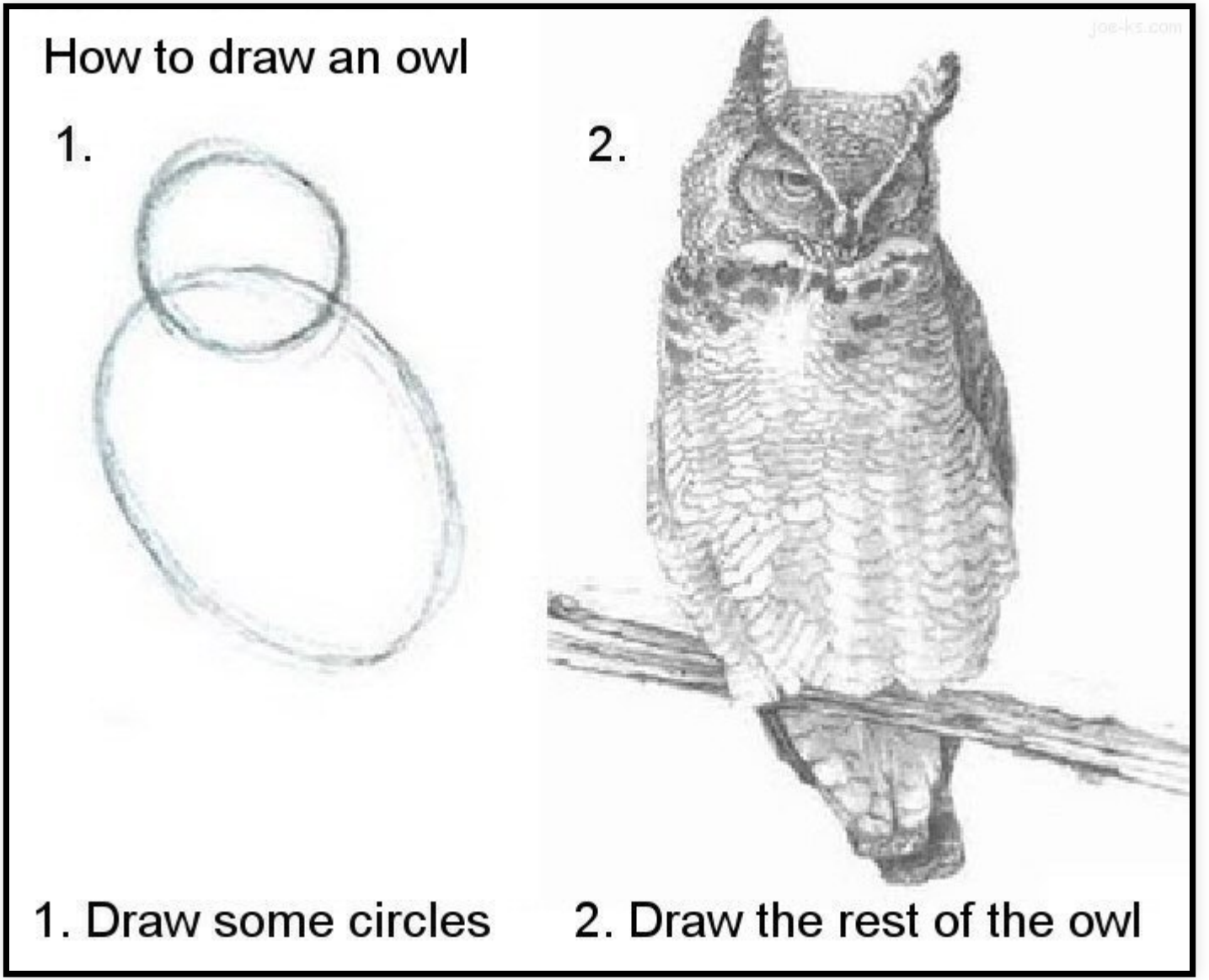
Planning

```
07/11 : Intro NoSQL +  PostgreSQL

14/11 : Cassandra Intro + Replication
21/11 : Cassandra modeling (timeseries)

28/11 : Apache Spark 1: Intro + RDD operations
05/12 : Apache Spark 2: exploratory data analysis using Dataframes

12/12 : AWS

19/12 : BD Graph, Neo4J

09/01 : MongoDB Intro + Data Modelling
16/01 : MongoDB Applications

17/01 : Projet 1
23/01 : Projet 2
24/01 : Soutenances
```

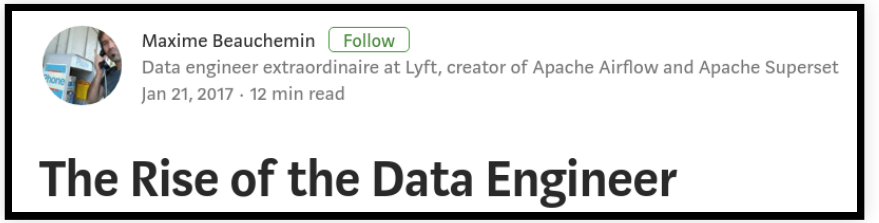
How

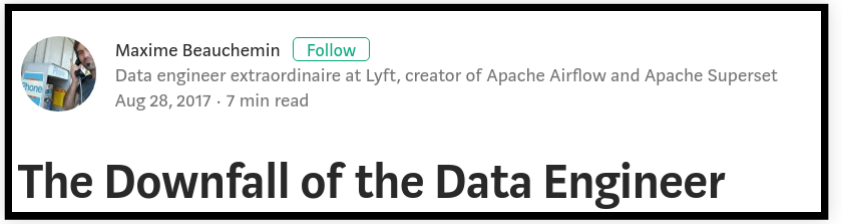
- Course: 1h30
- TP: 1h30 (small groups)
- Small surveys / home readings
- Projet
- Ressources ⇒ bit.ly/bigdata-telecom

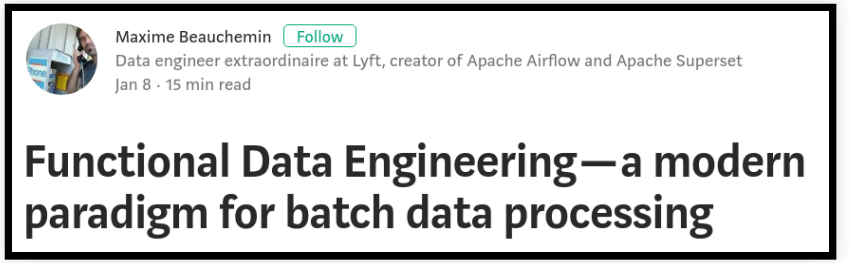
How it may seem



Home readings





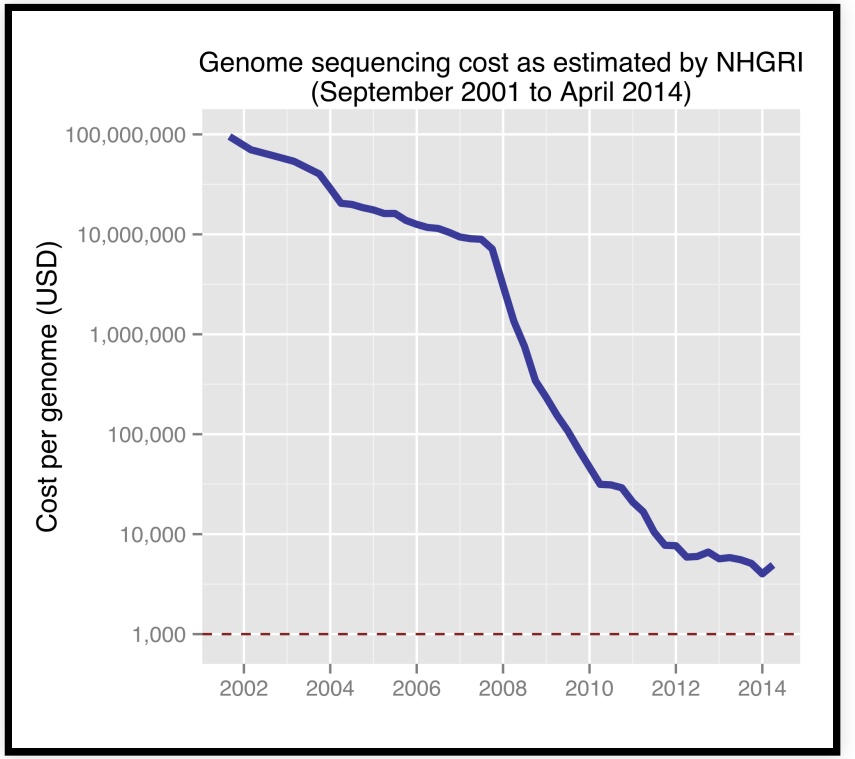


Plan

1. Course info
2. **From SQL to NoSQL**
    1. Scaling a simple application
    2. From relational to non-relational databases
3. Scaling MySQL @LesFurets.com
4. TP PostgreSQL

Why !

Data: the new hope

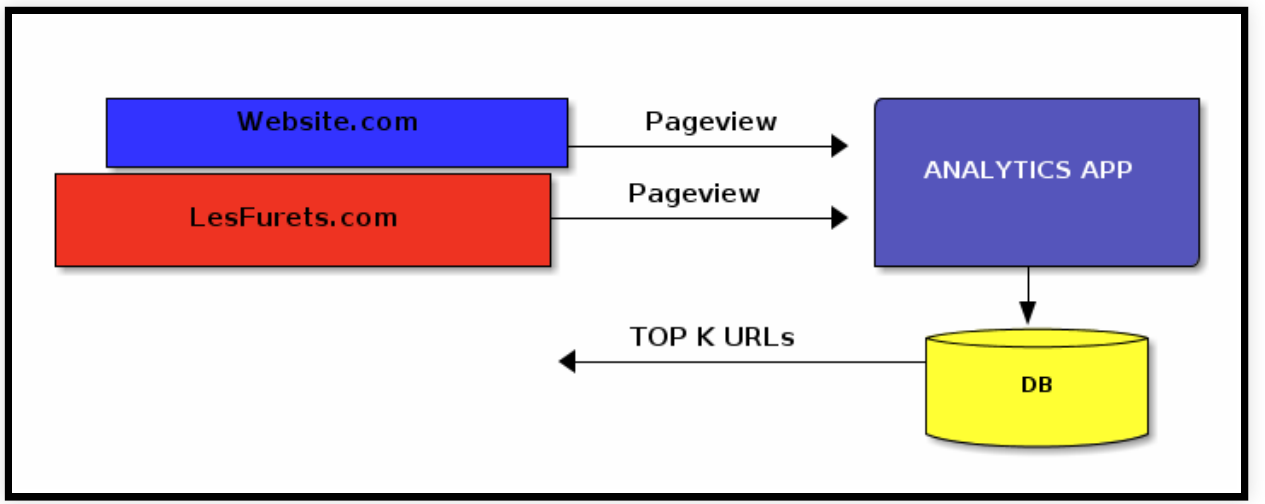


Processing the data has become the bottleneck!

Data: the new oil

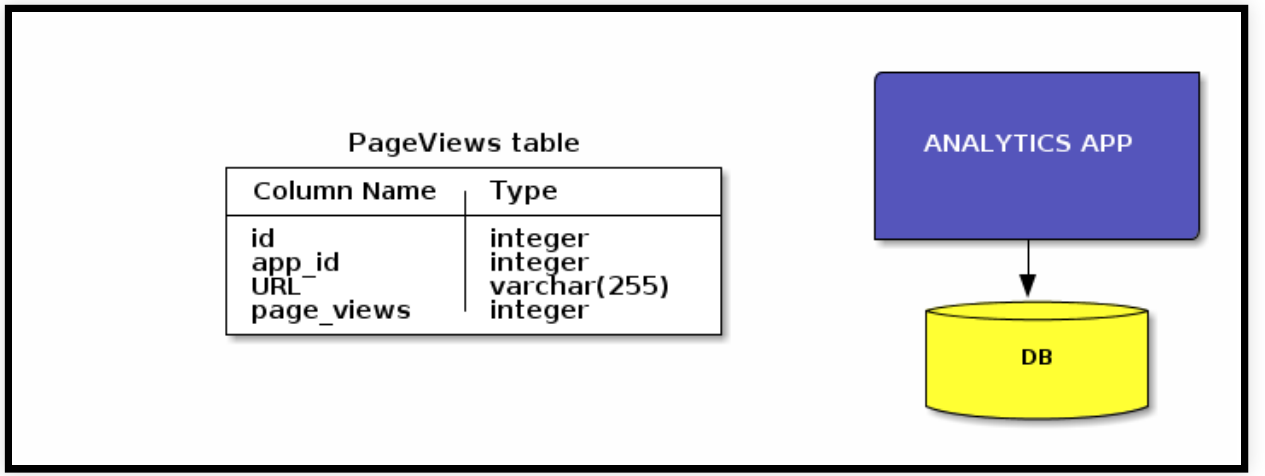
Data: most common use

- Web analytics applications
  - track the number of pageviews for each URL
  - what are the top 100 URLs



Simplest architecture

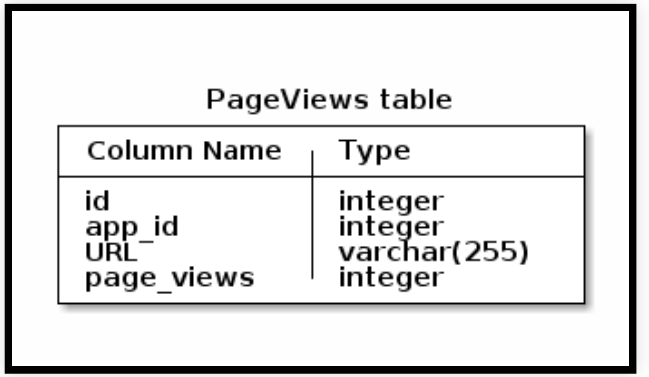
- track the number of pageviews for each URL
- what are the top 100 URLs



Queries

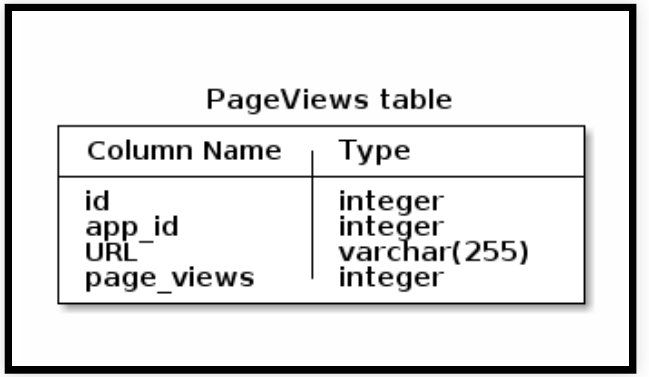
- insert a pageview
- update pageviews
- top 100 URLs for a client

Insert pageviews



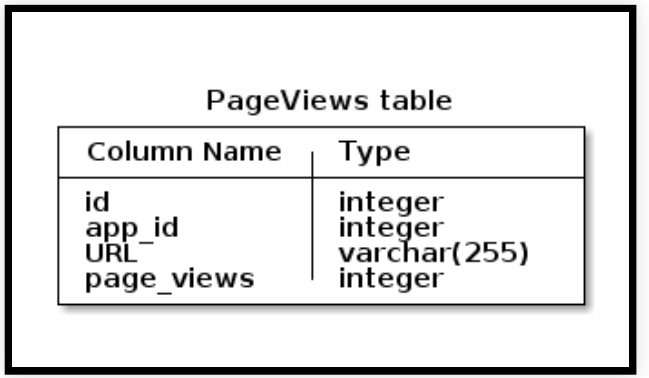

```
INSERT INTO PageViews(app_id, URL, page_views) VALUES
    (1,"http://www.lesfurets.com/index.html",1);
INSERT INTO PageViews(app_id, URL, page_views) VALUES
    (2,"http://Website.com/base.html",1);
INSERT INTO PageViews(app_id, URL, page_views) VALUES
    (1,"http://www.lesfurets.com/assurance-auto",1);
```

Update pageviews



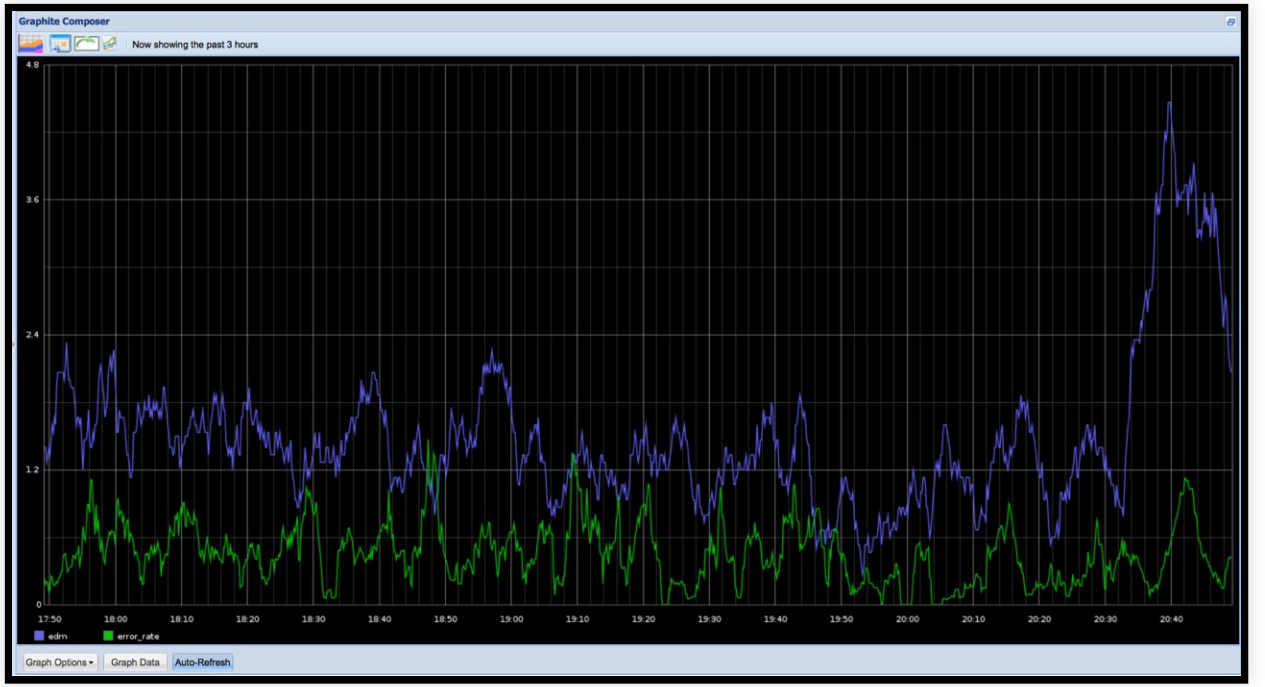

```
UPDATE PageViews SET page_views = page_views + 1
WHERE app_id="1" AND URL="http://www.lesfurets.com/index.html";
```

Top 100 URLs for a client

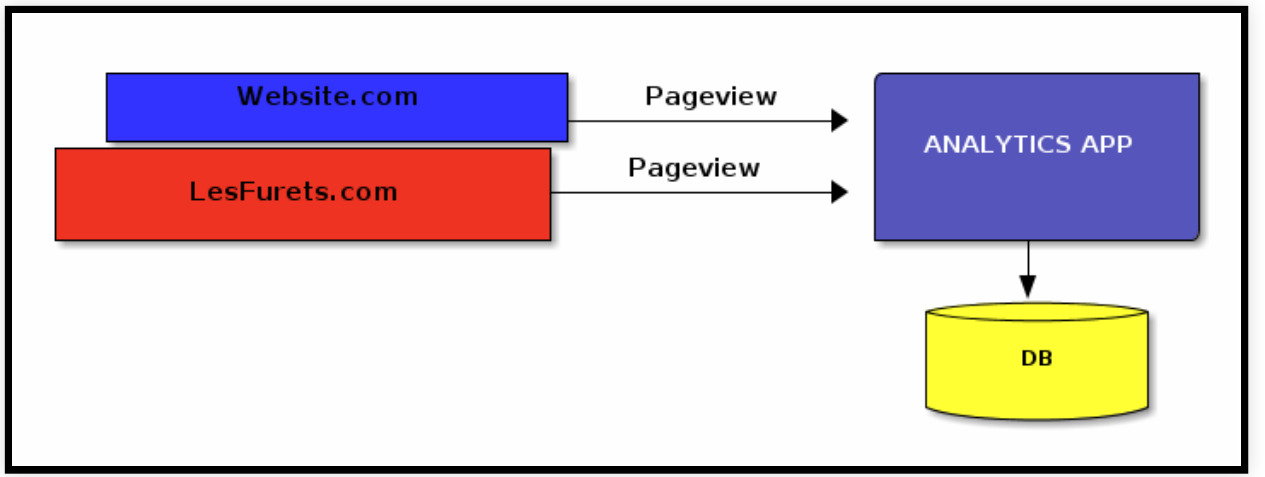



```
SELECT URL,page_views FROM PageViews
WHERE app_id = '1'
ORDER BY page_views DESC LIMIT 100
```

Production load

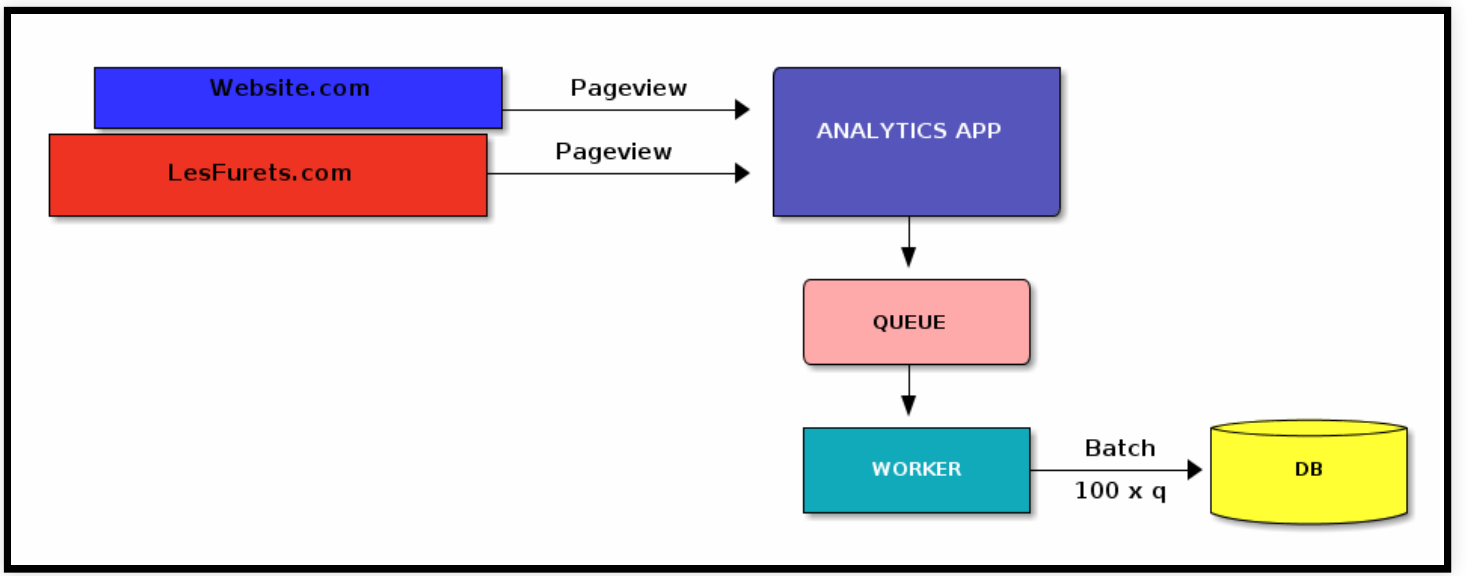


Timeouts



Timeout error on updating the database

Fix#1: queuing + batching



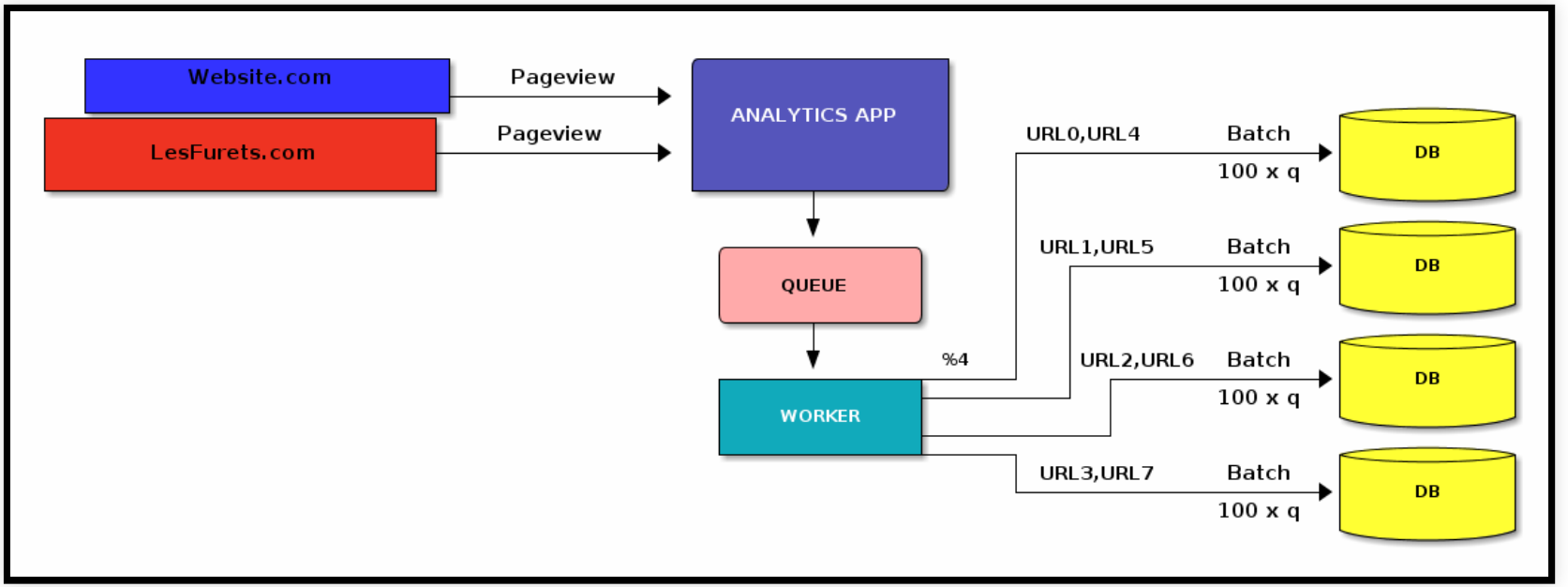
Fix#1: implications

- Modify the application ⇒ batch 100 queries
- Latency / queue size
- handle DB/queue failures ⇒ persistent queuing with event logging
- **cannot accommodate high load**

Fix#2: Sharding (vertical table partitioning)

- Spread the load
  - use multiple database servers
  - spread the PageView table across the servers
  - mapping keys to shards using a hash function

Fix#2 Sharding



Fix#2: Sharding implications

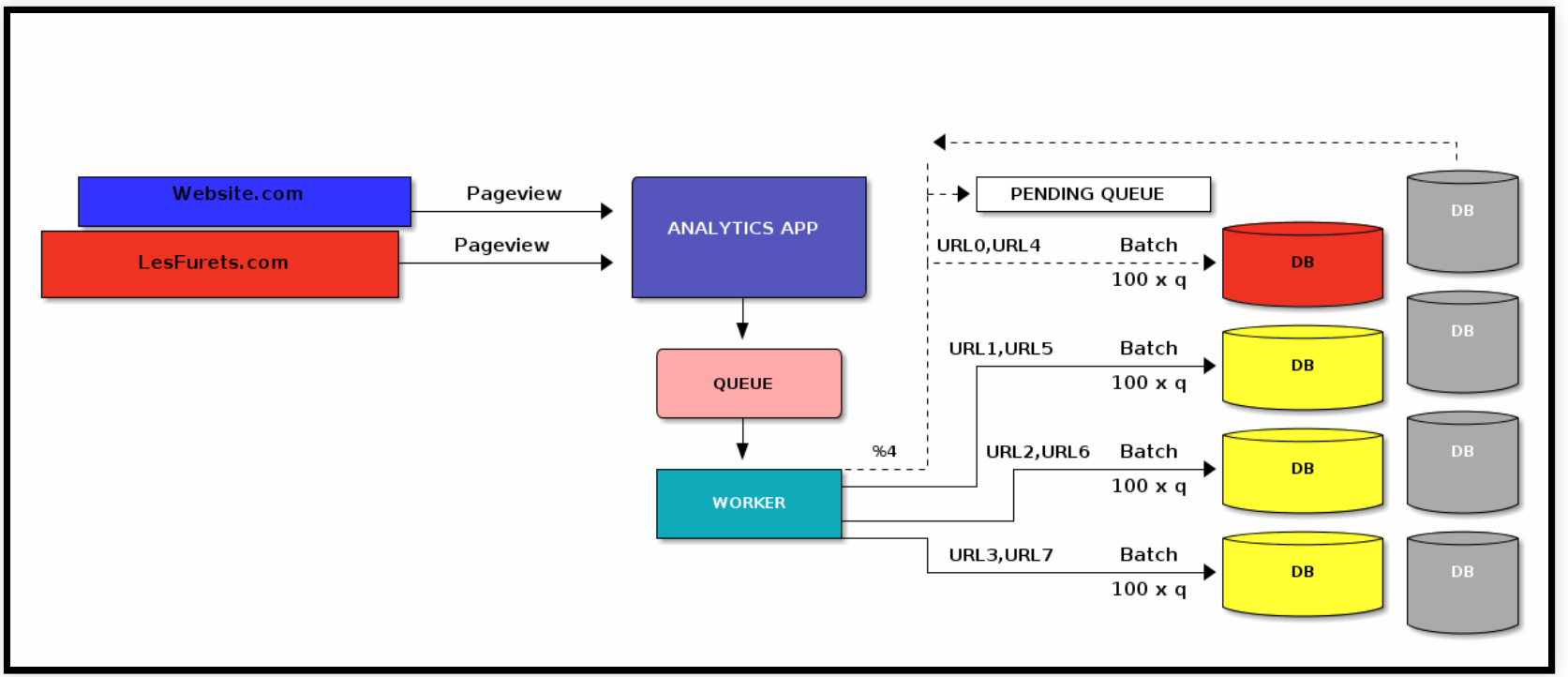
- distribute the keys to the new servers
- write to the "right" DB instance
- aggregate data from all the shards !
  - **Sharding more and more**
    - new shards to follow the load
    - repeat the last steps

Fix#2 Sharding

- **Server failures** are more likely
  - **WRITES**: use a pending queue flushed less frequently
  - **READS**: a portion of the data is unavailable

- ⇒ **replication**

Fix#3 Replication



Human failures

Distribution hash function = %3

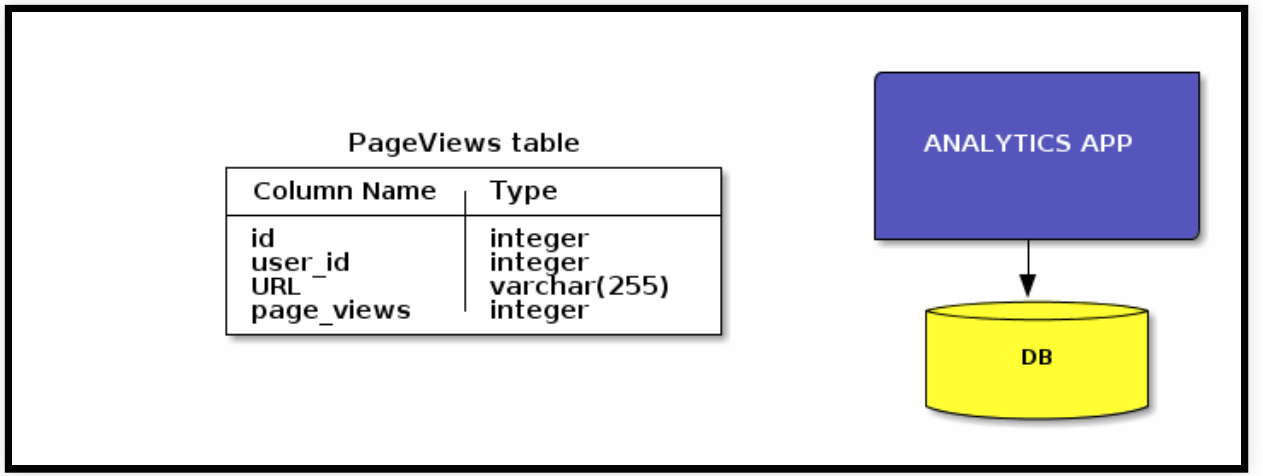
- **Data written to the wrong shards**
  - redistribute data to the right shard
  - while still accepting queries ?!

Human failures

```
UPDATE PageView SET page_views = page_views + 2
WHERE user_id='42' AND URL='myurl';
```

Increments the number of pageviews + 2

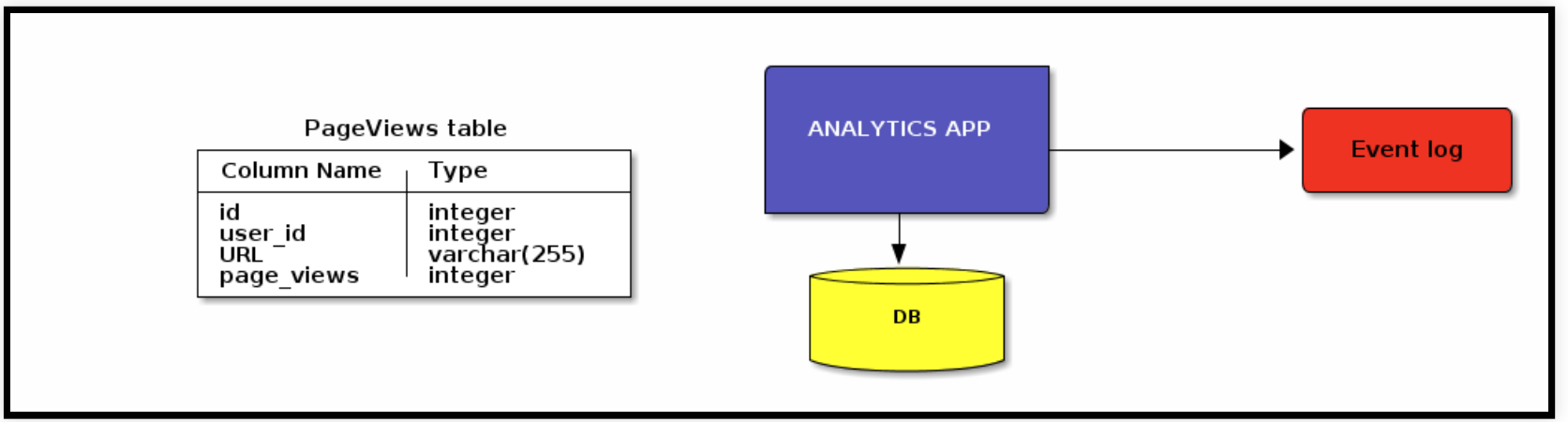
Human failures




```
UPDATE PageView SET page_views = page_views + 2
WHERE user_id='42' AND URL='myurl';
```

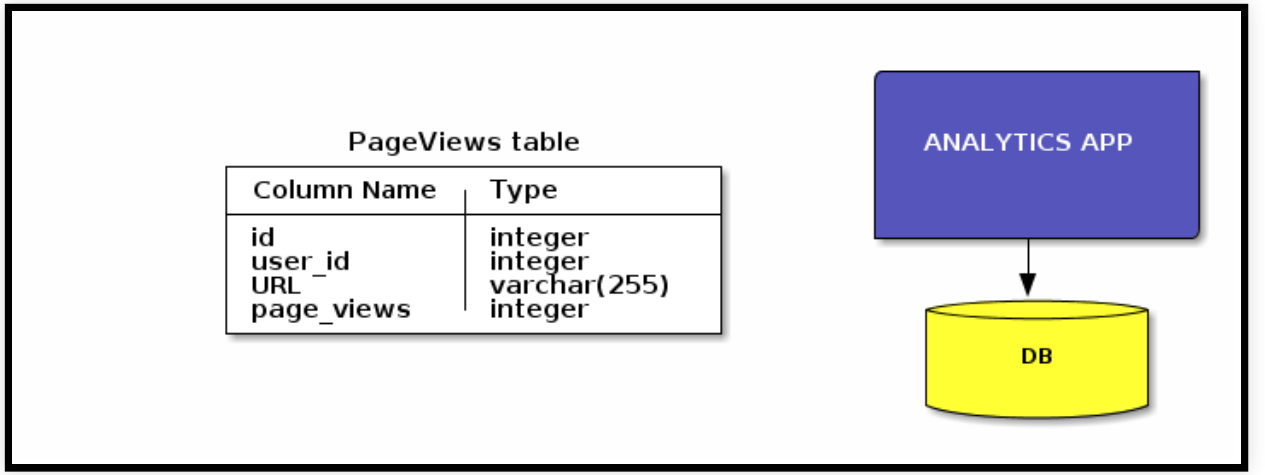
Human failures

- **event logging**



Human failures

- Incremental data model

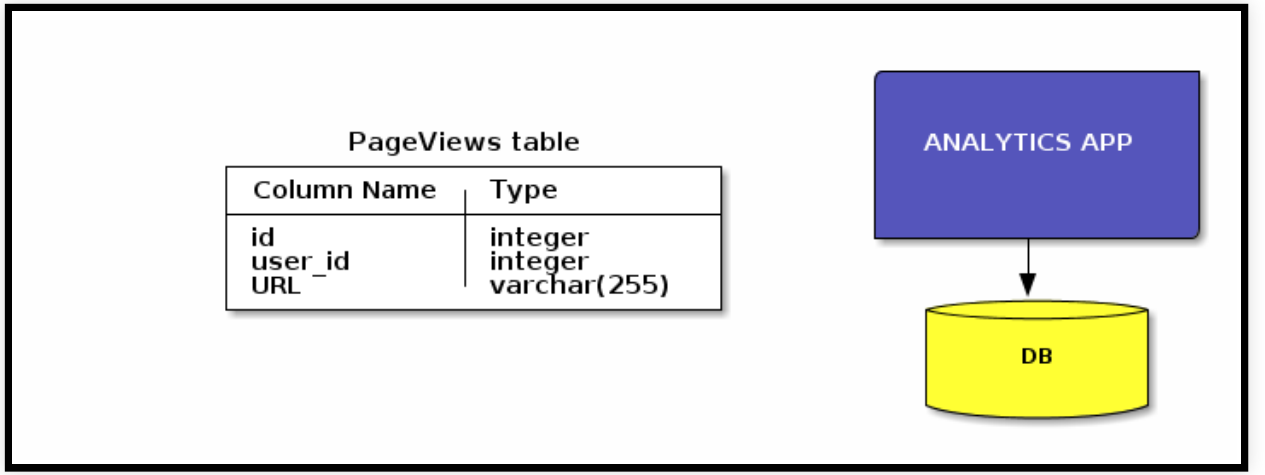


Incremental data model

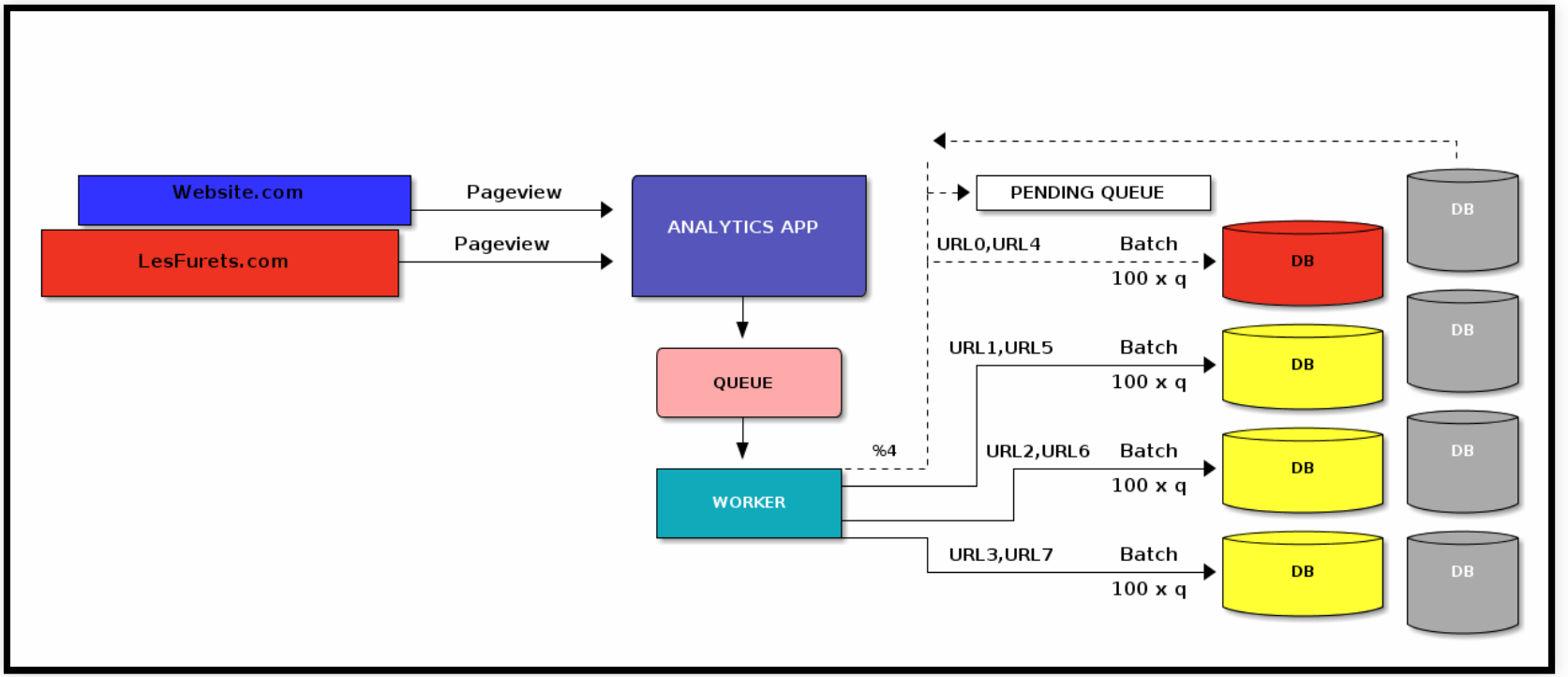
- **data corruption** ⇒ hard to correct (!)
- **contention** ⇒ locking ⇒ bad performance

Human failures

- Incremental data model ⇒ **immutable data model**



What went wrong?



- do I build new features for customers?
- or just dealing with reading/writing the data?

What went wrong?

- **A single server cannot take the load ⇒** solution / **complexity**

    1. *distributed storage*
    2. querying distributed data
    3. built a data model that is not resilient

Wishlist 1 : Storage

- ***Better storage***:
  - easy to add/remove nodes (**scaling**)
  - transparent data distribution (**auto-sharding**)
  - handle failures (**auto-replication**)

⇒ **Distributed databases**: *Redis*, ***Cassandra***, *HBase*, ***MongoDB***, *CouchDB*, ...

Wishlist 2 : Queries

- *General purpose (distributed) computing*:
  - distributed queries + **parallel processing**

⇒ **Distributed data processing engines : *MapReduce*, *Spark***

Wishlist 3 : Data model

- *We want a resilient data model*:
  - human error is **unavoidable**
  - an **incremental data model** is not resilient

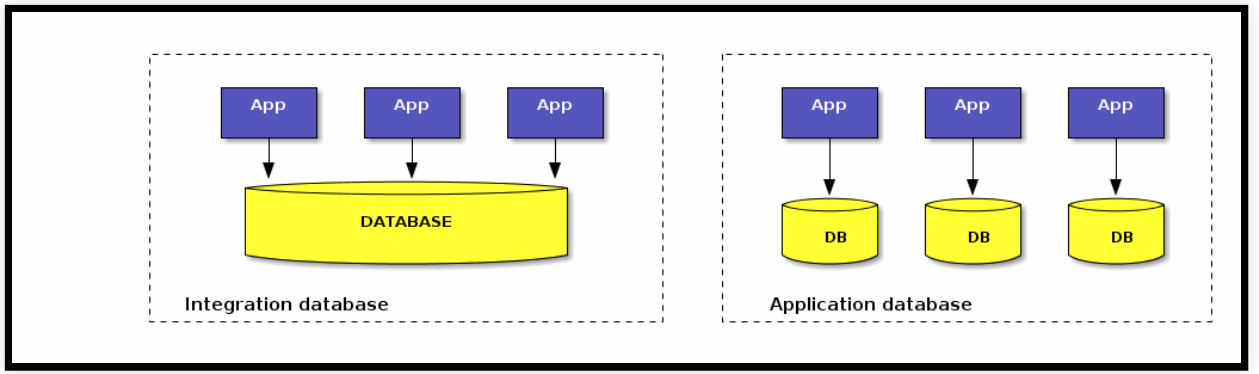
⇒ **Immutability**

Plan

1. Course info
2. From SQL to NoSQL
    1. Scaling a simple application
    2. **From relational to non-relational databases**
3. Scaling MySQL @LesFurets.com
4. TP PostgreSQL

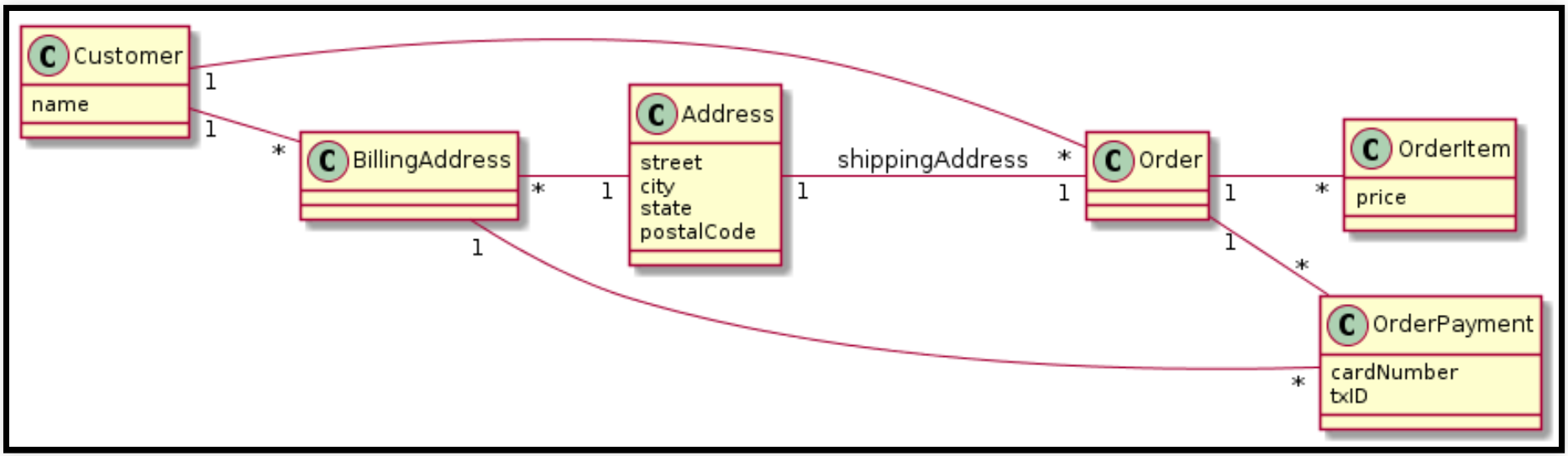
RDBMS: the good parts

- simple model with sound mathematical properties (ACID)

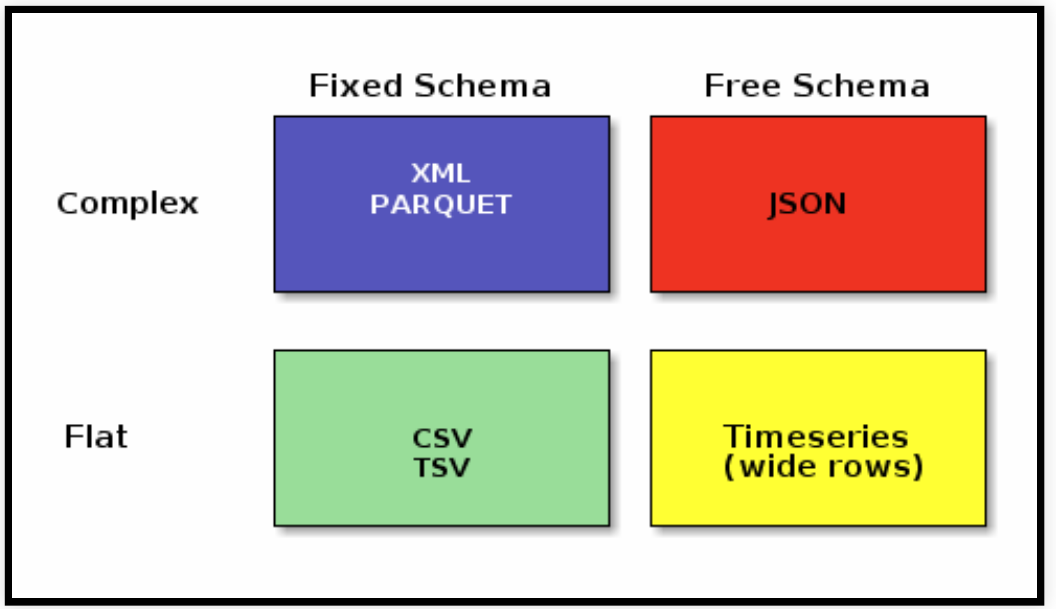


Integration database	Application database
consistent data set	easier to maintain/evolve/**scale**
changes need to be coordinated → side effects	standard interfaces between systems (SOA)

SQL models tuples and joins

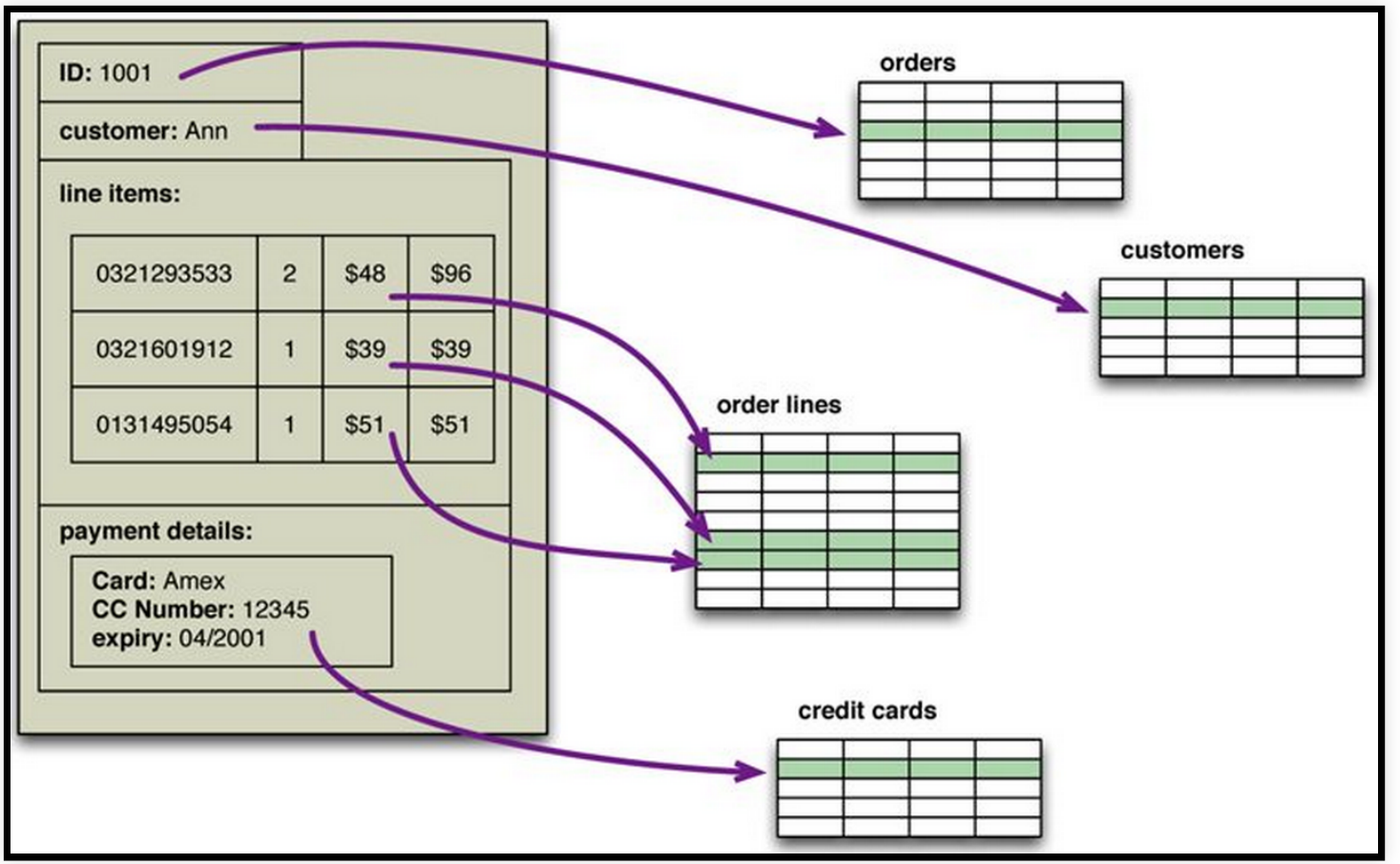


Modeling complex data



Relational vs Document/Objects

- **Relational model**: relations / tuples + normalization
- **Memory**: rich data structures !



Relational vs Document/Objects

- **Memory** - object graphs
  - complex mapping from object to relations → ORMs
  - leads to performance issues
    - many rows/tables/**JOINS**
- Schema evolution:
  - adding an attribute → adding a whole column
  - expensive locks → application downtime

RDBMs: scaling

- **Scaling** is expensive:
  - scaling writes → locking + partitioning (sharding)
  - scaling reads → replication (latency, error recovery)

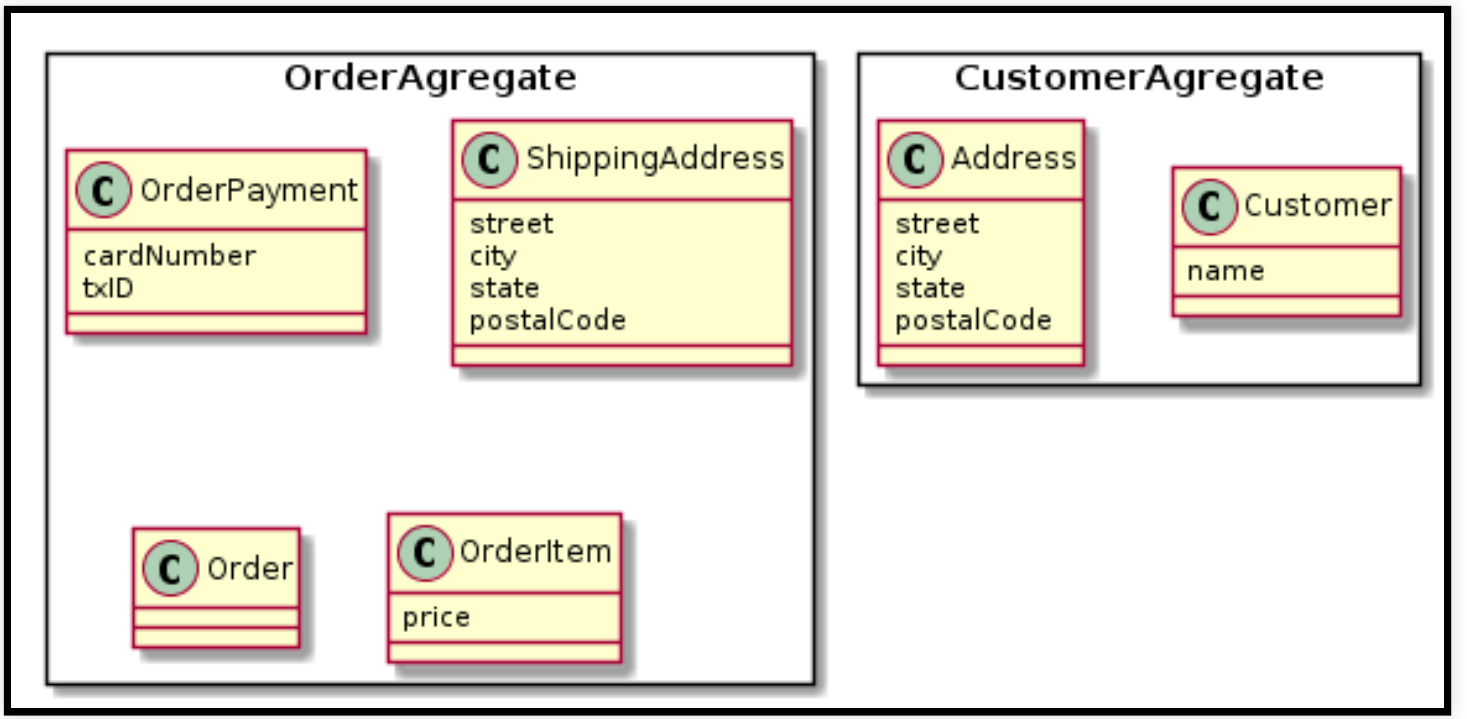
- **Not a good fit for RDBMs**
  - clustered databases ⇒ shared disk paradigm / cluster aware filesystem
  - application level sharding and replication

Not only SQL

- 2009, Johan Oskarson, #NoSQL meetup for distributed, open-source, non-relational databases
  - not using relational model SQL
  - run on clusters
  - ACID ⇒ tunable consistency
  - fixed schema ⇒ flexible schema
  - **polyglot persistance**

NoSQL models agregates

- collection of related objects that should be treated as a unit (consistency / data management)



Modelling SQL vs NoSQL

- *SQL*: **model first**
  - 1 model used for all queries
- *NoSQL*: **query first**
  - 1 data access pattern for each aggregate

NoSQL aggregate types

1. Key-value databases
2. Document-oriented databases
3. Column-oriented databases
4. Graph databases

Key-value databases

- Store and retrieve Blobs based on a primary Key
- Simplest API that matches REST : PUT/GET/DELETE

Map (K → V)

```
Use case: Session information, user profiles, ...
```

Document oriented databases

- Stores and retrieves **documents/fragments** (XML, JSON…):
  - self-describing, hierarchical tree data structures
  - maps, collections and scalar values

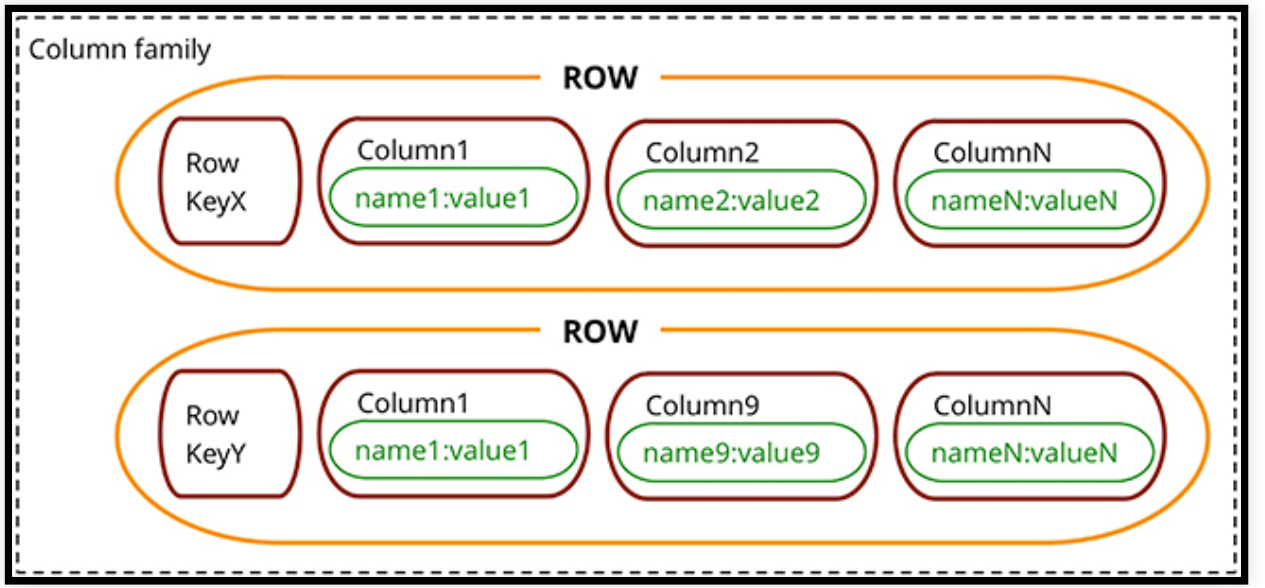
Key-value stores where the value is queryable

```
Use case:  CMS, Product catalog, ...
```

Column oriented databases

- Stores data in **tables/column families** as rows that have many columns associated to a row key

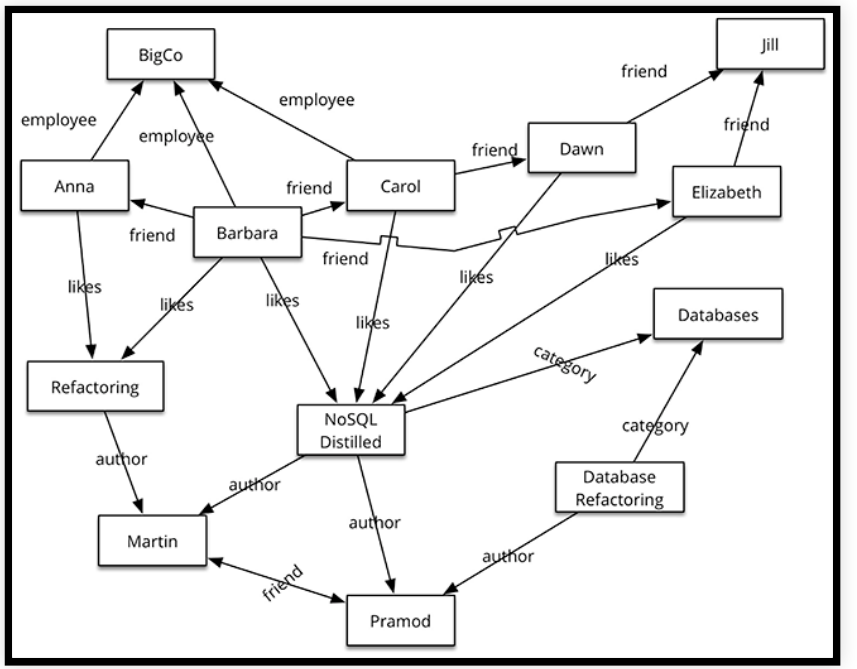
Map of maps (rowId → (columnName → columnValue))




```
Use cases: Time series, event logging, ...
```

Graph databases

- Stores **entities** and **relations** between entities
- **Query:** traversal of the graph



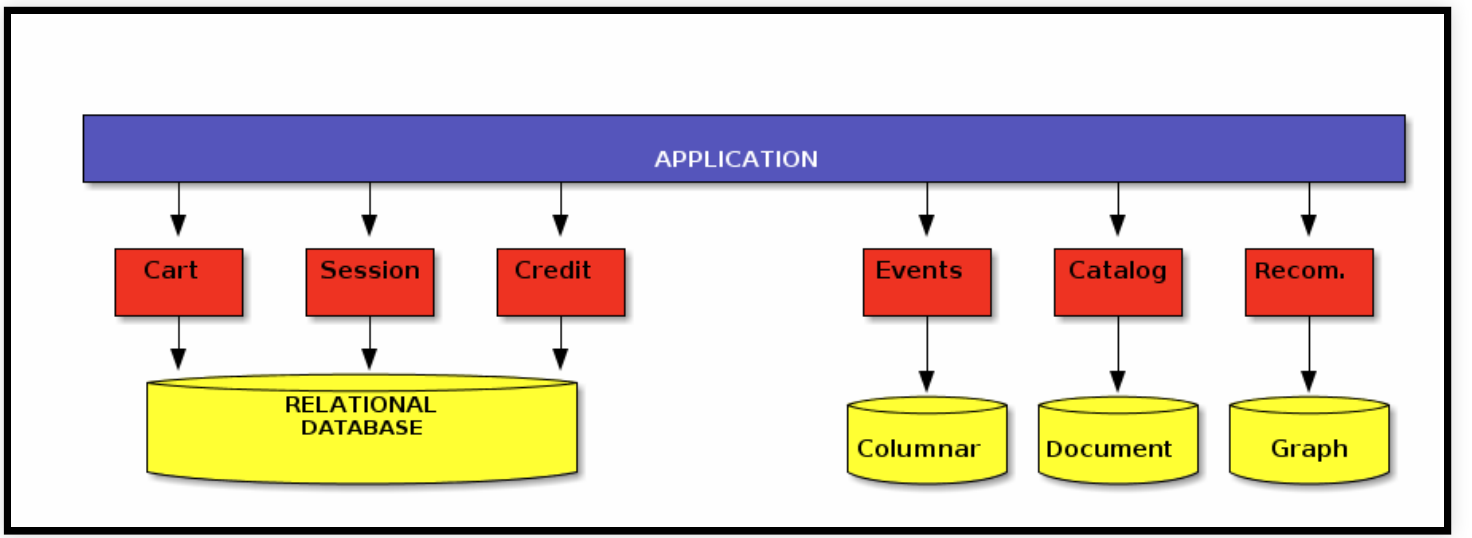
Use cases: Social networks, recommendation engines

NoSQL tradeoffs

- Performance:
  - Hadoop: large scale batch computation but **high latency**
  - Cassandra: low latency, fin grain storage but **limited data model**
- Consistency: **ACID** ⇒ tunable/eventual consistency (**CAP**)
- Model:
  - Incremental architectures ⇒ human failures

Polyglot persistance

- **aggregates** have different requirements (availability/consistance/backup)
- **Mix and match relational and non-relational storage**



Plan

1. Course info
2. From SQL to NoSQL
3. **Scaling MySQL @LesFurets.com**
4. PostgreSQL + TP

Scaling MySQL

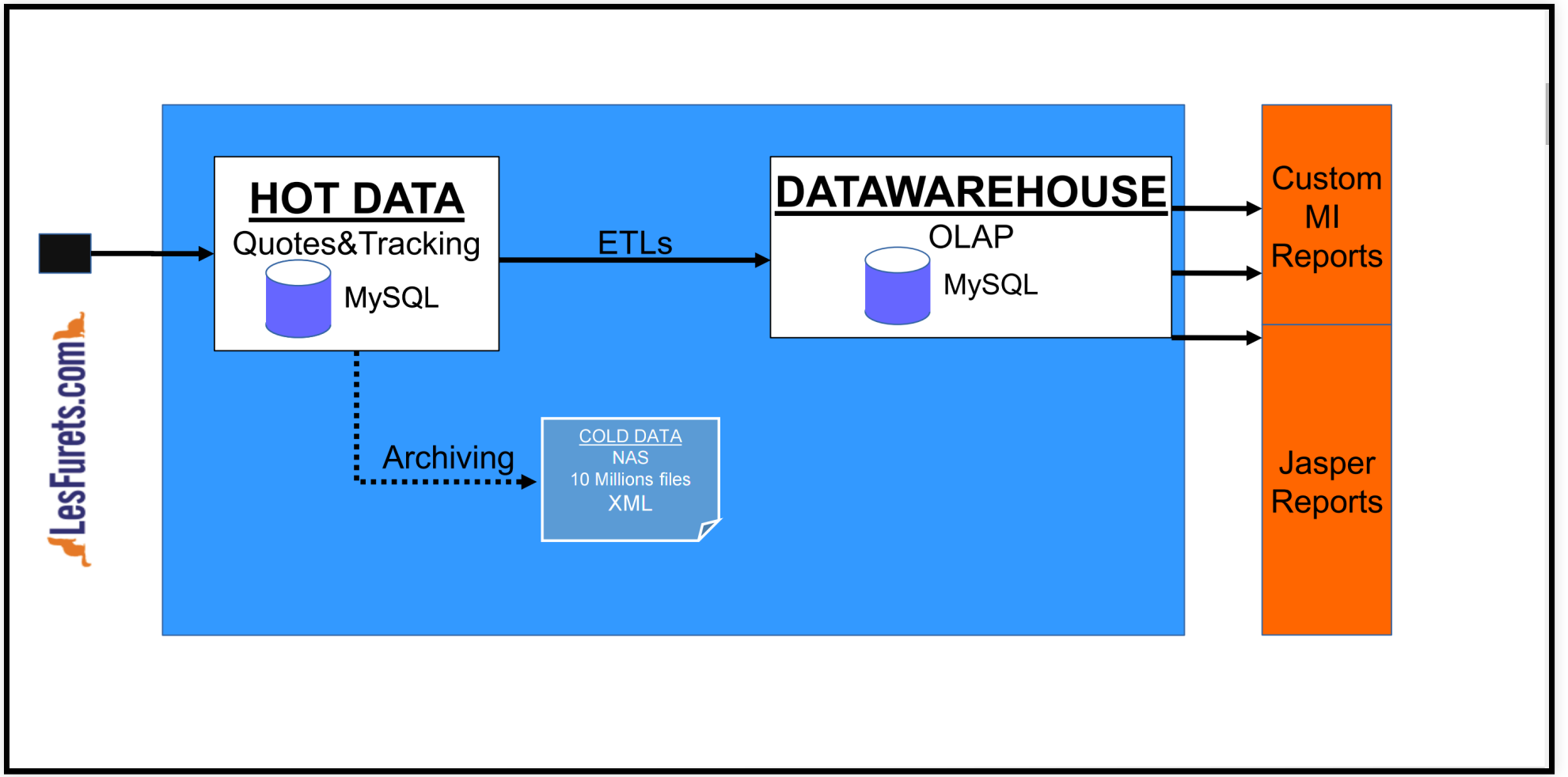
LesFurets.com

- Independent insurance aggregator
- OLTP (Runtime DB):
  - replicated MariaDB ⇒ DRBD ⇒ Galera Cluster
  - Cassandra / Spark (migration ongoing)
- OLAP (Analytics/BI DB):
  - MariaDB (snowflake Schema)
  - QlikView dashboard
  - Spark/Zeppelin over MySQL/Cassandra: ad-hoc analyses & ETLs

Data characteristics

- Specific workload
  - few updates
  - few synchronous queries
  - applicative caches
- Polyglot persistance
  - SQL
  - XML - blob (MySQL) and file (NFS/AWS S3)
  - ElasticSearch - centralized logging
  - Cassandra

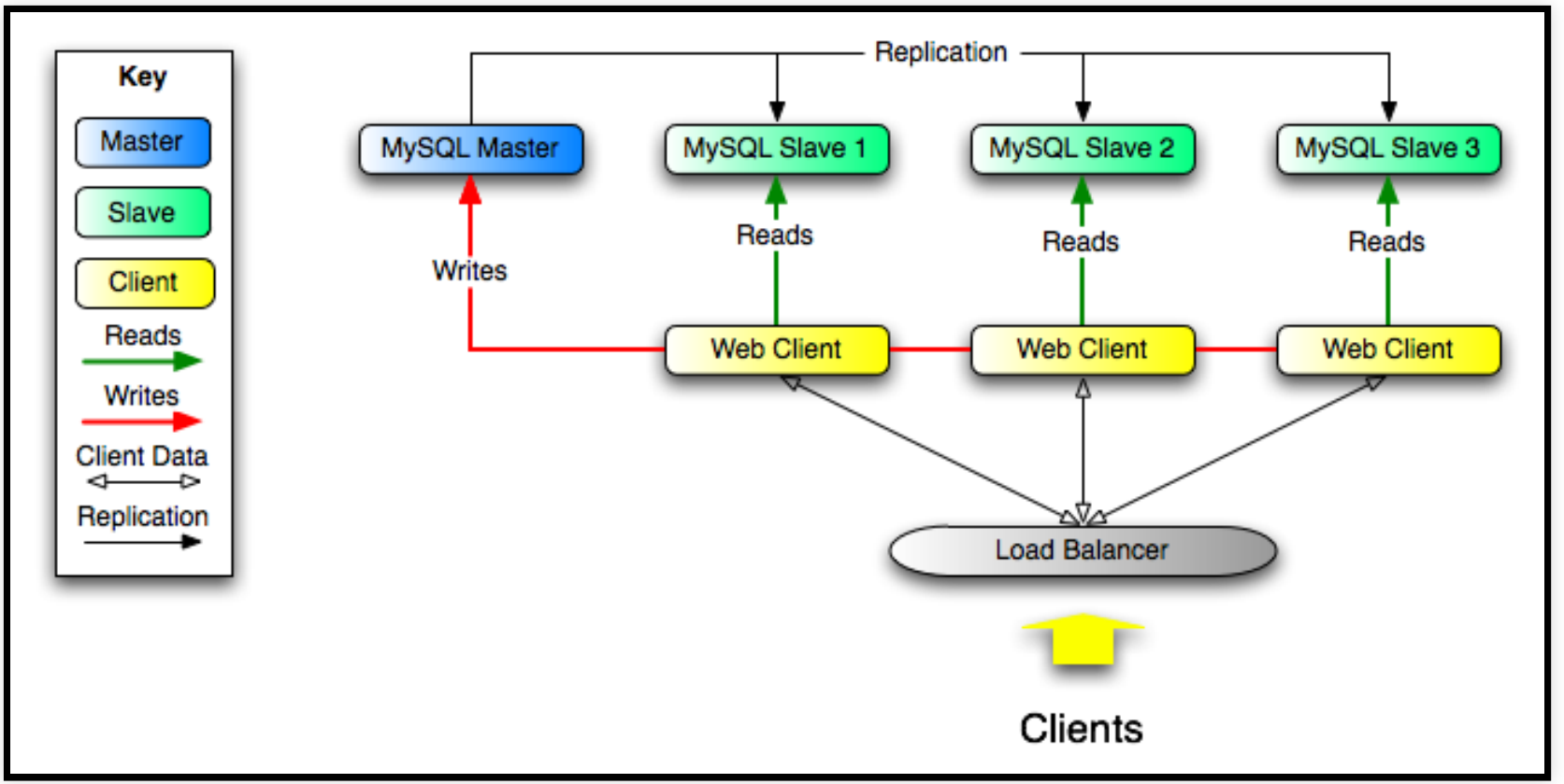
Data workflow



Why replicate?

- scale out
- **data security**
- **segregate workload** (writes on master/ analytics on slaves)
- data distribution
- **backups**
- restore in a point in time (delayed replication)

MySQL scale-out



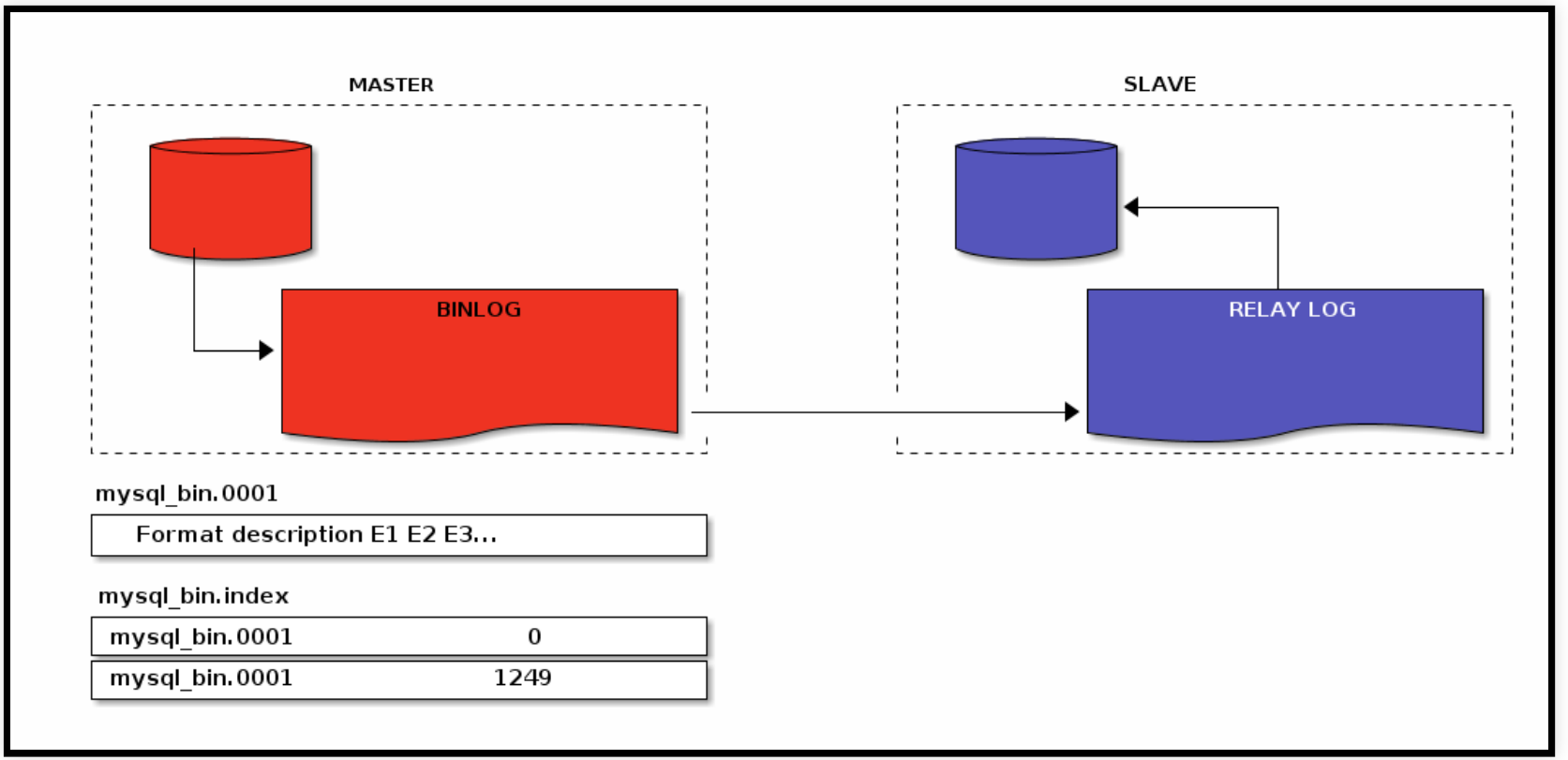
MySQL replication properties

- **1 Master ⇒ n Slaves**
- **Application transparent**
- **Full replication**: the full dataset is replicated on every node

MySQL basic replication

- **Master**
  - writes/updates only on master
  - log changes (events) on a bin-log
- **Slave(s)**
  - retrieve events from the master
  - replay the events

MySQL replication: binlog



MySQL replication: master configuration

```
/etc/mysql/my.cnf
server-id               = 1
log_bin                 = /var/log/mysql/mysql-bin.log
binlog_do_db            = database_name

GRANT REPLICATION SLAVE ON *.* TO 'slave_user'@'IP' IDENTIFIED BY 'password';
FLUSH PRIVILEGES;

-- backup: myslqdump/dumper, innobackupex/xtrabackup
```

MySQL replication: slave configuration

```
-- restore backup

/etc/mysql/my.cnf
server-id               = 2
relay-log               = /var/log/mysql/mysql-relay-bin.log
log_bin                 = /var/log/mysql/mysql-bin.log
binlog_do_db            = database_name
```

MySQL replication: slave configuration

```
CHANGE MASTER TO
MASTER_HOST='mariadb1.vrack.courtanet.net',
MASTER_USER='slave_user',
MASTER_PASSWORD='****',
MASTER_LOG_FILE='mariadb-bin.002716',
MASTER_LOG_POS=972282938;

START SLAVE;
```

MySQL replication: SHOW SLAVE STATUS

```
+---------------------------------+-----------------------------+-------------+-------------
| Slave_IO_State                  | Master_Host                 | Master_User | Master_Port
+---------------------------------+-----------------------------+-------------+-------------
| Waiting for master to send event | mariadb1.vrack.courtanet.net | slave-data2 |        3306
+---------------------------------+-----------------------------+-------------+-------------

+---------------+-------------------+---------------------+------------------+---------------
| Connect_Retry | Master_Log_File   | Read_Master_Log_Pos | Relay_Log_File   | Relay_Log_Pos
+---------------+-------------------+---------------------+------------------+---------------
|            60 | mariadb-bin.002725 |           957457227 | relay-bin.000023 |     264745891
+---------------+-------------------+---------------------+------------------+---------------

+----------------------+------------------+-------------------+------------+------------+--------------
| Relay_Master_Log_File | Slave_IO_Running | Slave_SQL_Running | Last_Errno | Last_Error | Skip_Counter
+----------------------+------------------+-------------------+------------+------------+--------------
| mariadb-bin.002725    | Yes              | Yes               |          0 |            |            0
+----------------------+------------------+-------------------+------------+------------+--------------

+--------------------+-----------------+-----------------------+-------------------------------+---------------+---------------+---------------+----------------
| Exec_Master_Log_Pos | Relay_Log_Space | Seconds_Behind_Master | Master_SSL_Verify_Server_Cert | Last_IO_Errno | Last_IO_Error | Last_SQL_Errno | Last_SQL_Error
+--------------------+-----------------+-----------------------+-------------------------------+---------------+---------------+---------------+----------------
|          928618688 |       957458099 |                   783 | No                            |             0 |               |             0 |
+--------------------+-----------------+-----------------------+-------------------------------+---------------+---------------+---------------+----------------
```

MySQL replication

- one-way M → S replication
  - single threaded/multi-threaded (5.6+ 1 worker thread per database/slave)
  - **asynchronous** : wait until change recorded in local binlog
  - **semi-synchronous** : wait until one Slave ack received and stored event!)

What is replicated?

- SBR (statement based replication)
- RBR (row base replication)
- Mixed (choose by event size for every transaction)

SBR

- send **queries** to the slave
- not all queries are safe for replication (!)
  - **SERVER STATE** : NOW(), AUTOINCREMENT, TRIGGERS, TRANSACTIONS

RBR

- send all **modified rows** to the slave
- safer but costly
- 5.6+ optimizations (ignore blobs, increment, hashing..)

Mixed

- Mixed (choose by event size for every transaction)

Semi-synchronous replication

- since 5.6+
- log SERVER_ID+TxID in a regular table
  - master blocks after the commit and waits until one semisynchronous slave acknowledges that it has received all events for the transaction or timeout
  - slave acknowledges receipt of a transaction's events only after the events have been written to its relay log

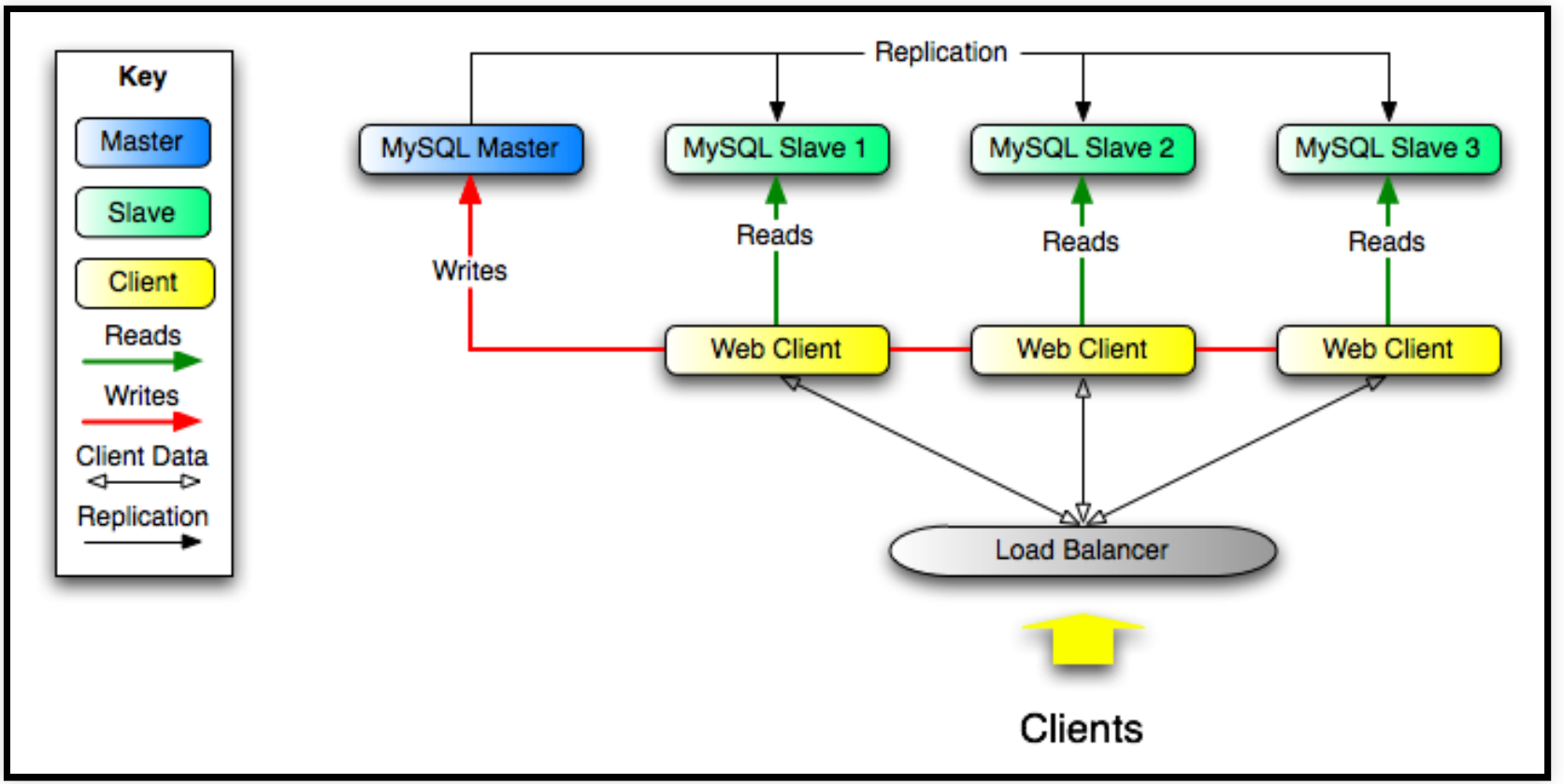
Semi-synchronous replication

- temporary switch to async-replication if no ACK from the slave
- **guarantees consistency between master and slave**
  - as long as all transactions committed on the master have also been applied on a slave

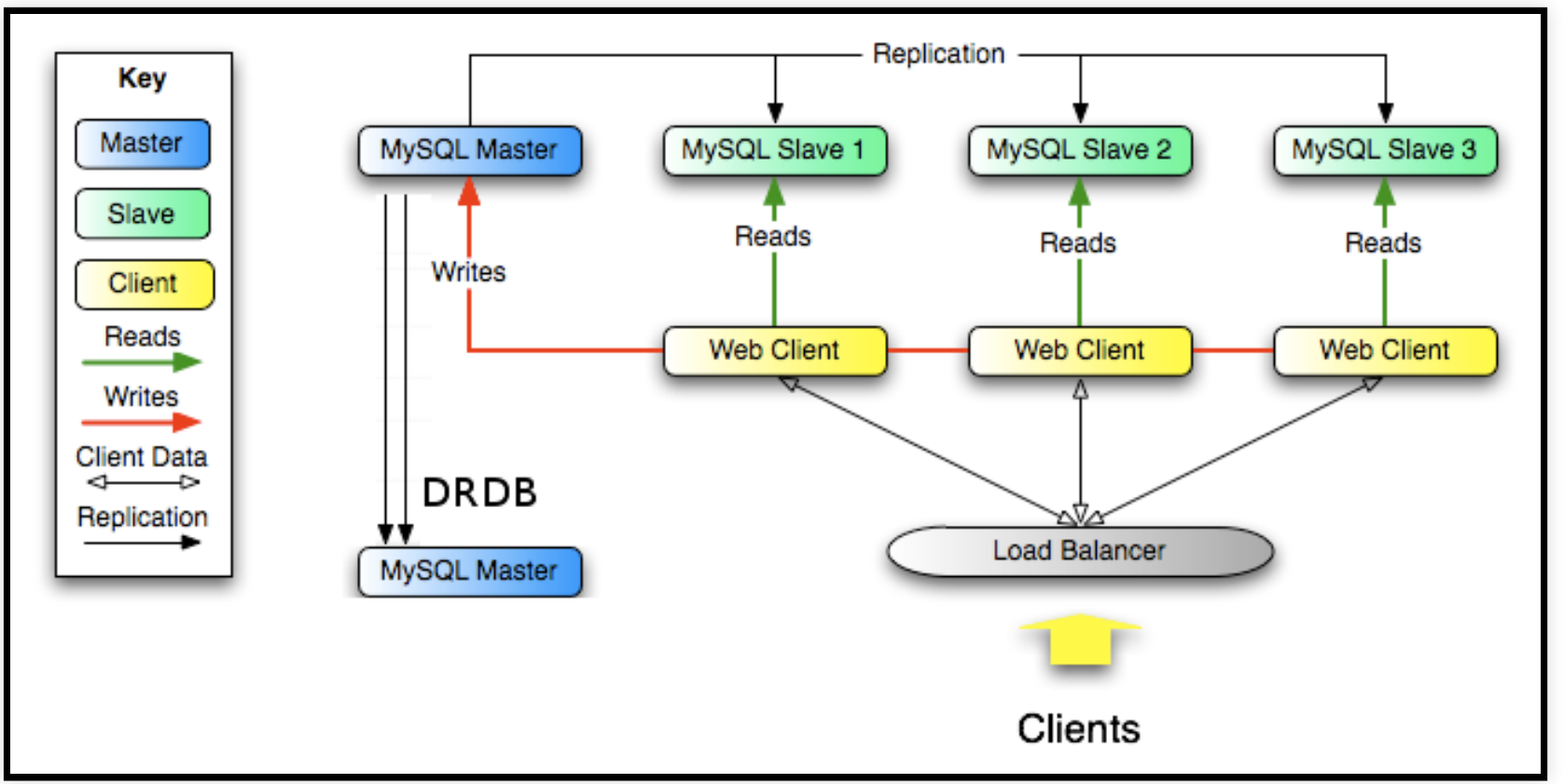
Crash recovery (M)

- manually promote a slave as a new master (<5.6)
- automatic slave promotion via mysqlfailover (5.6+)

Master/Slave



Multi - Master replication with DRBD



Scaling Writes

- Sharding : MySQL Fabric
- Multi-master replication
  - MySQL cluster
  - ***Galera Cluster***

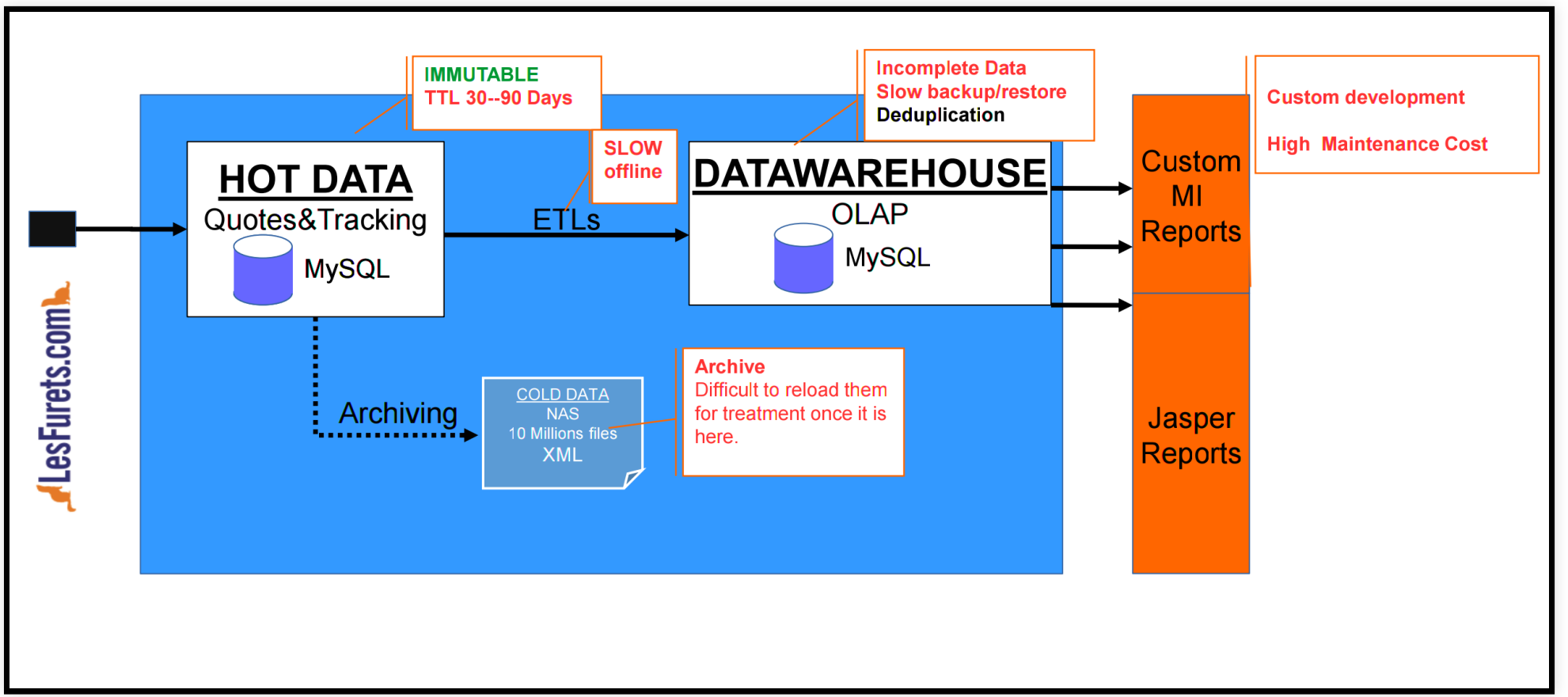
Galera Cluster

- multi-master: R/W at any node
- synchronous* replication by Certification Based Replication Method
  - good latency with increased consistency (*more…*)
    - Easy migration from MySQL
      - combine with MySQL binlog replication
      - automatic node provisioning (XtraBackup)
      - transparent to applications

LF MariaDB Galera Cluster

- relaxed ACID
- *wsrep_sync_wait* - ensures sync before:
  - READ (SELECT/SHOW/BEGIN/START TRANSACTION)
  - UPDATE/DELETE
  - INSERT/REPLACE
- Benefits:
  - configurable consistency
  - recover from failures (typical < 1s)
  - optimize replication lag (typical < 10ms)

Architecture concerns



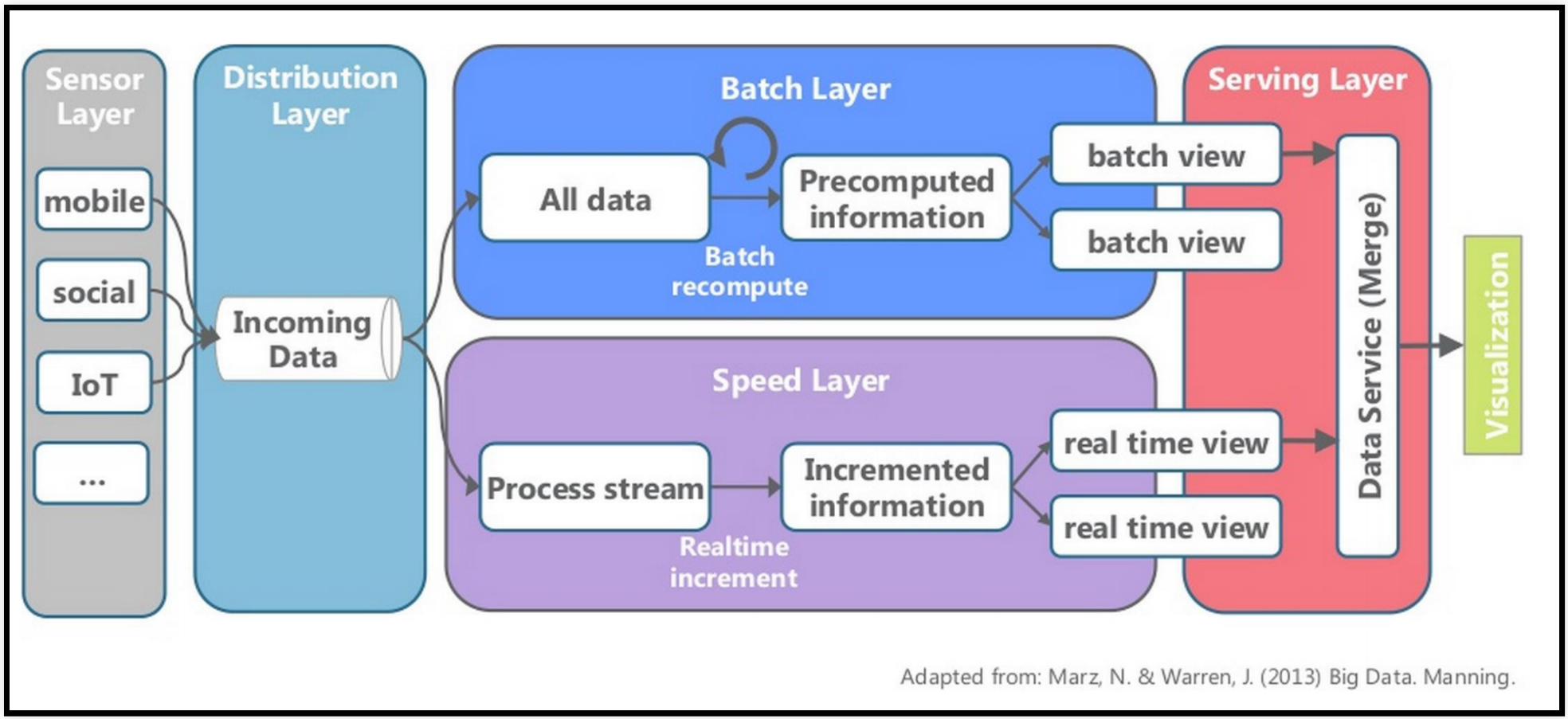
Scaling MySQL @LF

- *Master/Slave* replication:
  - network partitions and **split brain**
  - slowly diverging Master/Slave, not automatic check/resynchronisation
  - problematic when failover switch
- *Multi-master* (**Galera**): 3 years of incident free operation
- **Table size limits** ⇒ "ALTER TABLE" problem

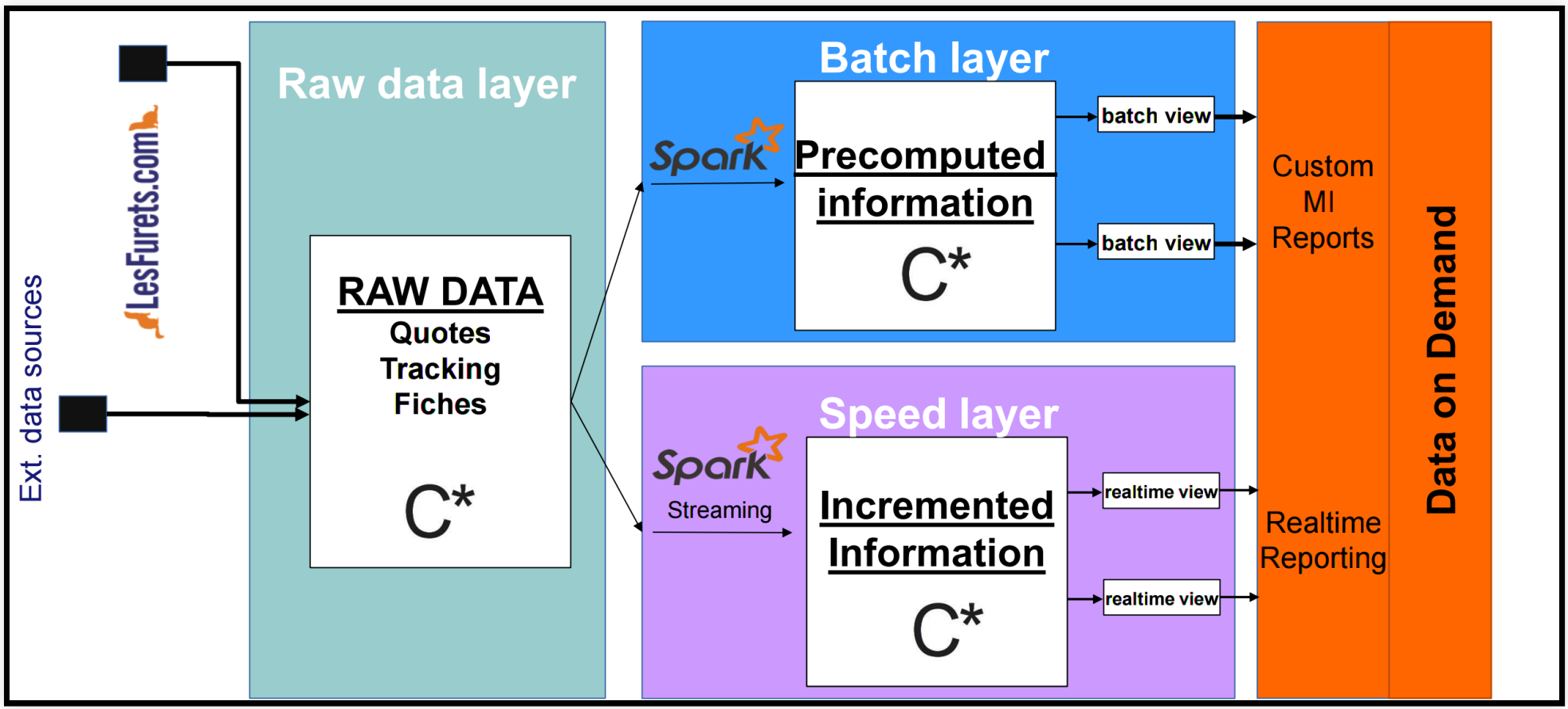
Transition to NoSQL

- **Lambda architecture -** NoSQL pattern to achieve
  - scalable systems
  - for arbitrary data problems
  - with human fault tolerance
  - and minimum complexity

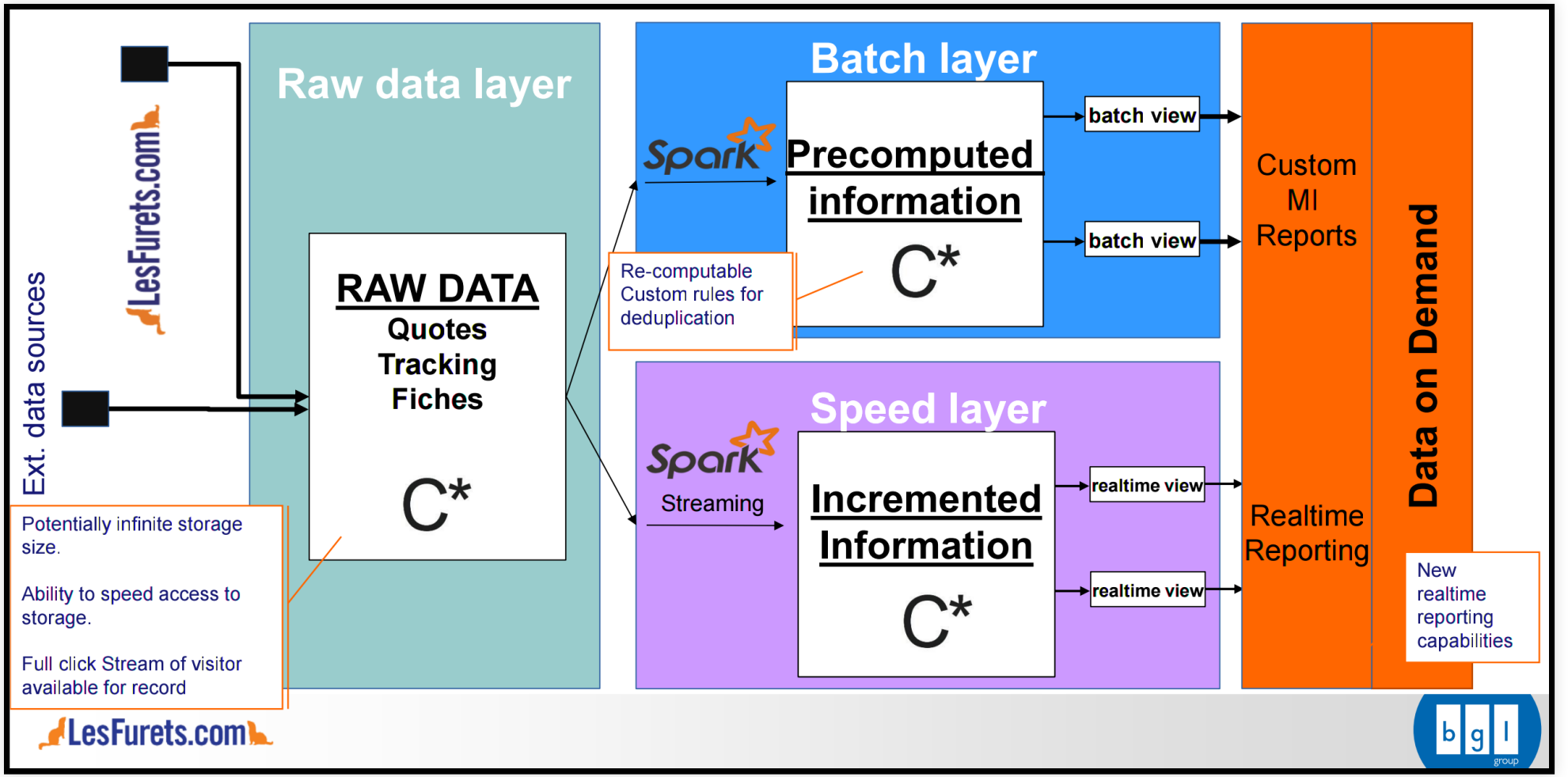
Lambda architecture



Lambda architecture @LesFurets



Lambda architecture @LesFurets



Plan

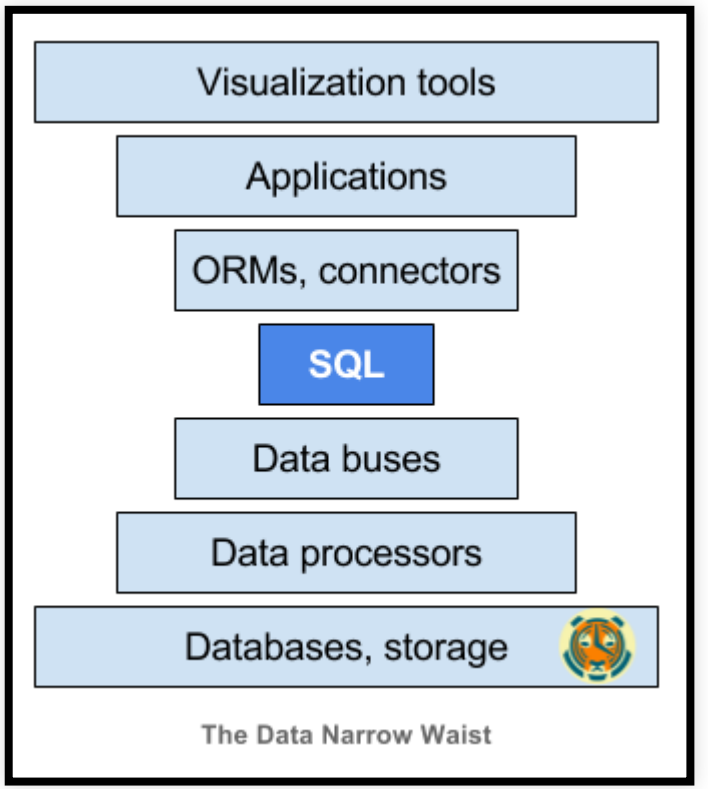
1. Course info
2. From SQL to NoSQL
3. Scaling MySQL @LesFurets.com
4. **TP PostgreSQL**

SQL in 2018 ?

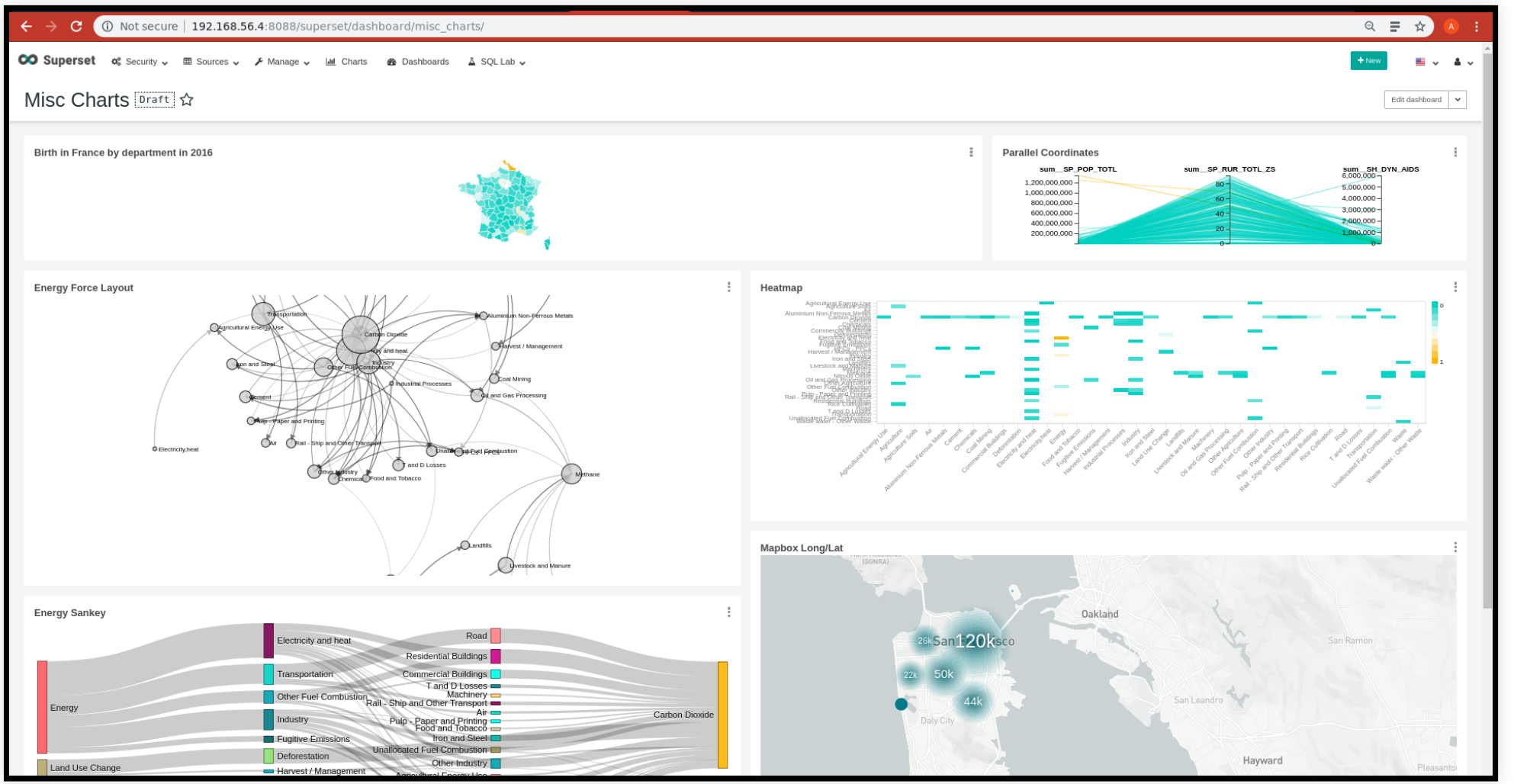
- **MapReduce: A major step backwards?** (*2008*)
  - MapReduce is difficult to integrate with other DBMS tools (*BI*, *Reporting tools*..)
    - Schemas are good
    - Separation of the schema from the application is good
    - High-level access languages are good

SQL in 2018 ?

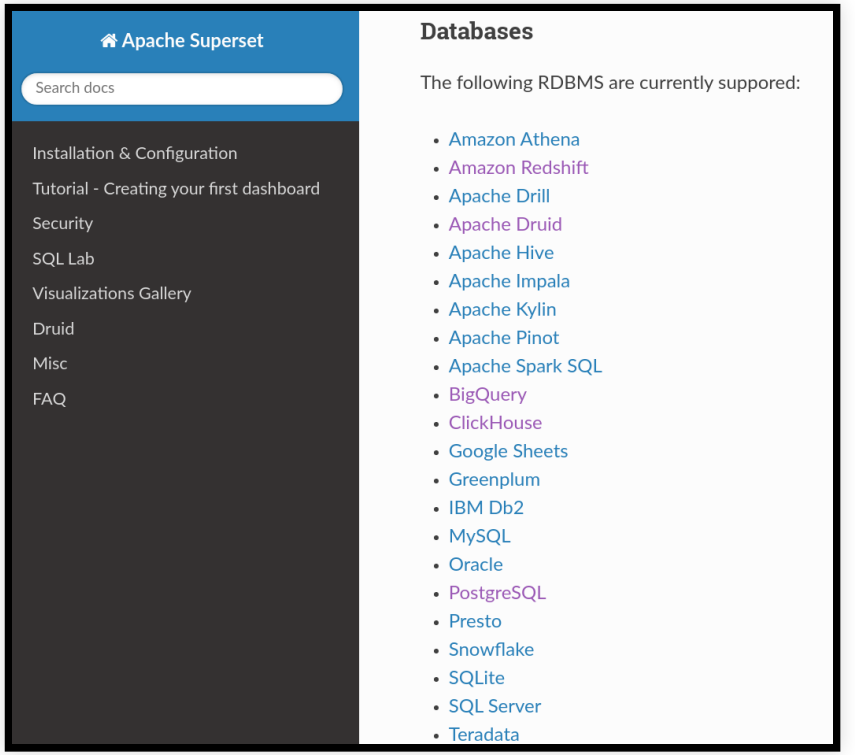
- **Why SQL is beating NoSQL, and what this means for the future of data** (2017)
  - SQL has become the narrow waist for data analysis



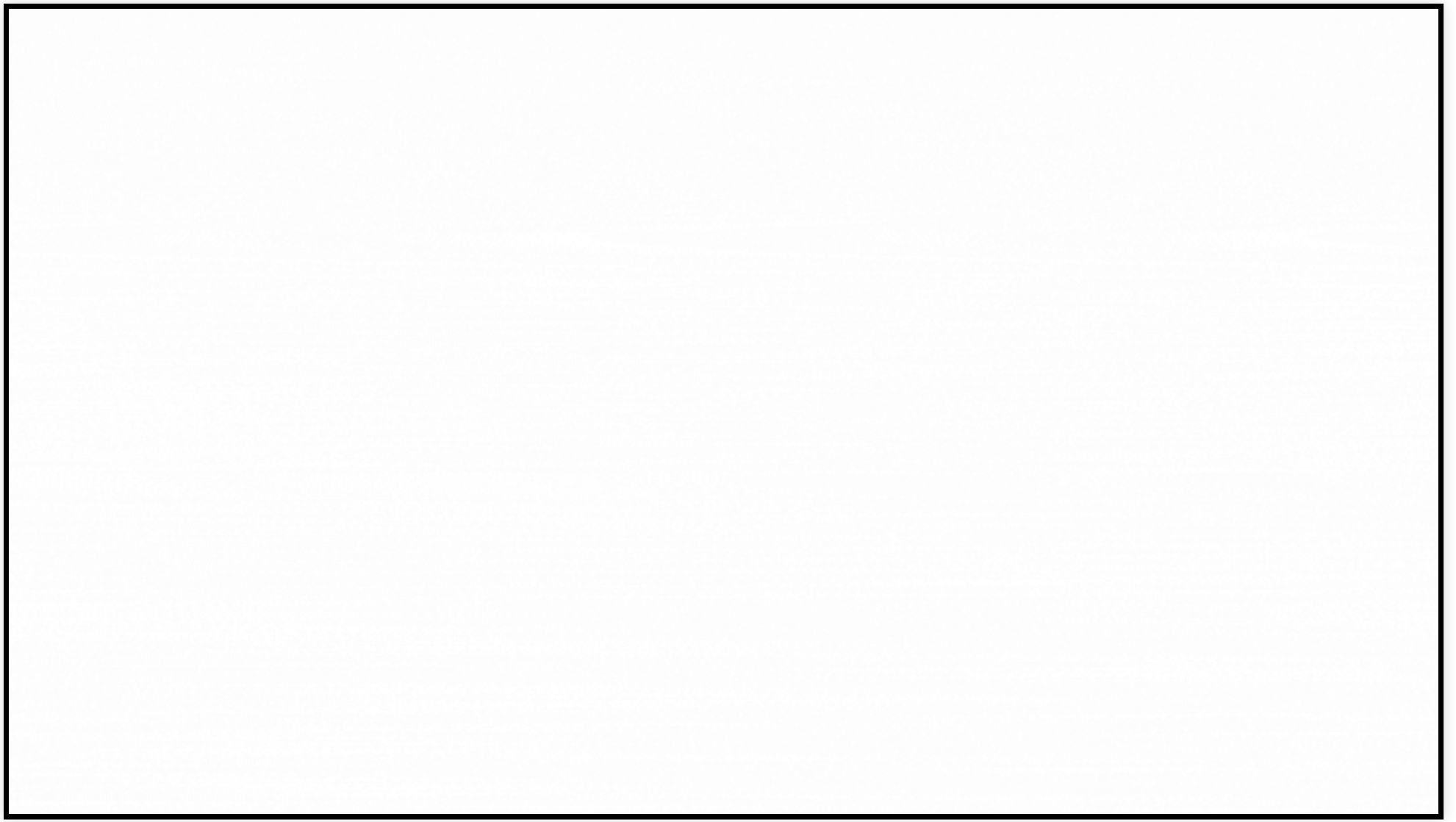
Apache Superset: business intelligence web application



Apache Superset: RDBMS support



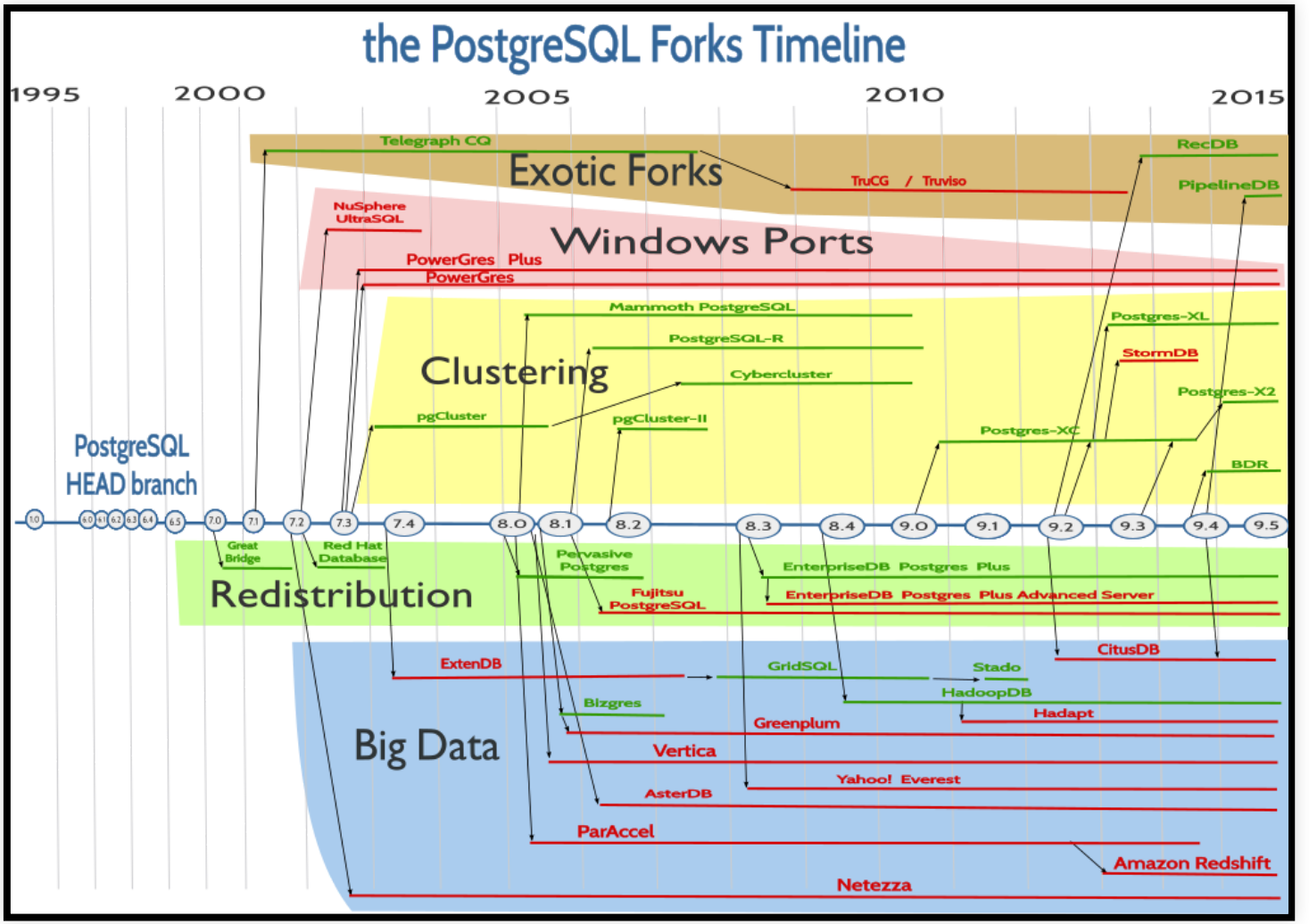
BigQuery ML



Why PostgreSQL?

- PostgreSQL
  - 1974: **INtelligent Graphic RElational System**, *Michael Stonebraker (Berkeley University)*
  - 1985: POSTinGRES then **PostgreSQL** (1995)
- Features
  - easy to extend/customize : **data types** (*JSONB*, *Arrays*, *Cube*), **operators**, **UDF** (*Python*, *Ruby*, *R*, *Javascript*…)
  - **Foreign Data Wrappers**: map an external DBs to tables ⇒ strong ACID properties @scale

PostgreSQL offsprings



PostgreSQL features

- PostGIS - advanced geospacial database
- Rich indexes: Gin/GIST/KNN/Sp-GIST
- natural-language processing
- multidimensional indexing
- TP ⇒ PostgreSQL & extensions (*tablefunc*,*dict_xsyn*, *fuzzystrmatch*, *pg_trgm*, *cube*)

more...