Intro: Bigdata/SQL

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

Plan

1. **Intro**
2. From SQL to NoSQL
3. Scalling MySQL : @LesFurets.com
4. TP: PostgreSQL

Background

- XML databases
- software developper/consultant
- data engineer @LF

tp-bigdata@lesfurets.com

Planning

```
16/11 : Intro +  PostgreSQL
22/11 : MongoDB Intro + Data Modelling
23/11 : MongoDB Application
30/11 : Cassandra Intro + Replication

06/12 : Cassandra modeling (timeseries)
07/12 : Apache Spark Intro + Shell
13/12 : Apache Spark 2 - Netbook
14/12 : Apache Spark 3 - MLlib

04/01 : Projet 1
10/01 : Projet 2
11/01 : Projet 3
18/01 : Projet 4

25/01, 01/02 : Soutenances
```

- LesFurets.com:
  - Andrei Arion, Geoffrey Berard , Charles Herriau
- Télécom ParisTech:
  - Ons Jelassi, Siwar Garouachi, Guillaume Pape

How

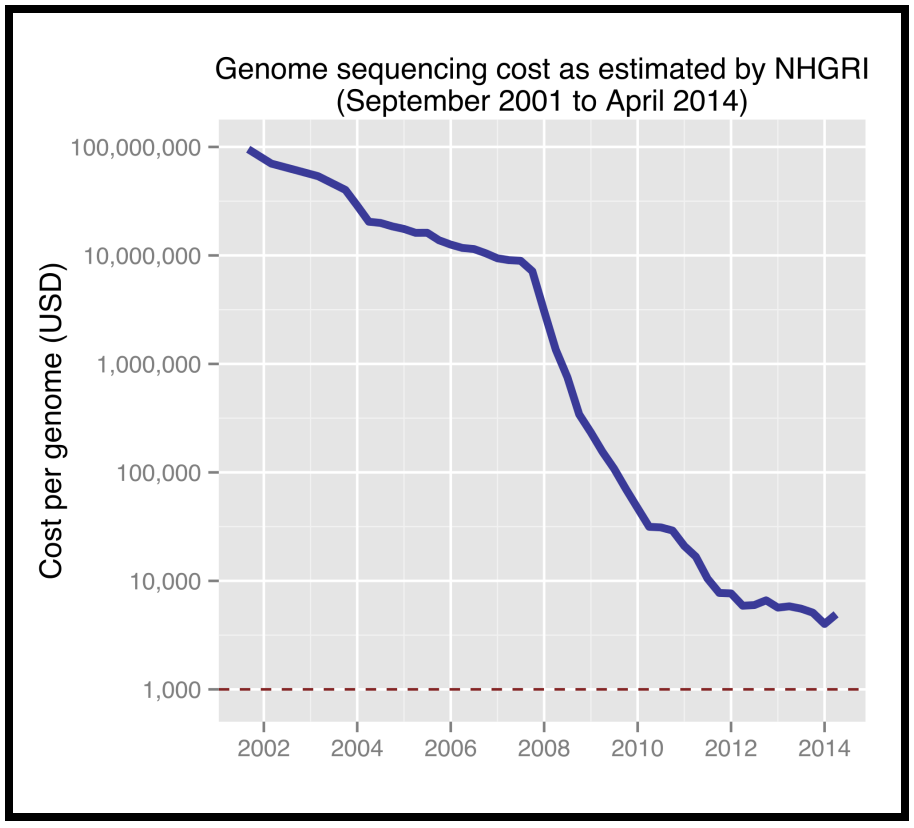
- Intro: 1h-1h30
- TP: 30 min + 2h30 pratique
- Travail en petit group (2p TP, 3p Projet)
- Évaluation en continuu (questions/etapes projet)
- Soutenance 30 minutes + 5 min questions

Plan

1. Intro
2. **From SQL to NoSQL**
    1. Scalling a simple application
    2. From relational to non-relational databases
    3. Scalling MySQL : @LesFurets.com
    4. PostgreSQL + TP

Why !

ADN sequencing



Processing the sequenced data has become the bottleneck!

— Andy Petrella

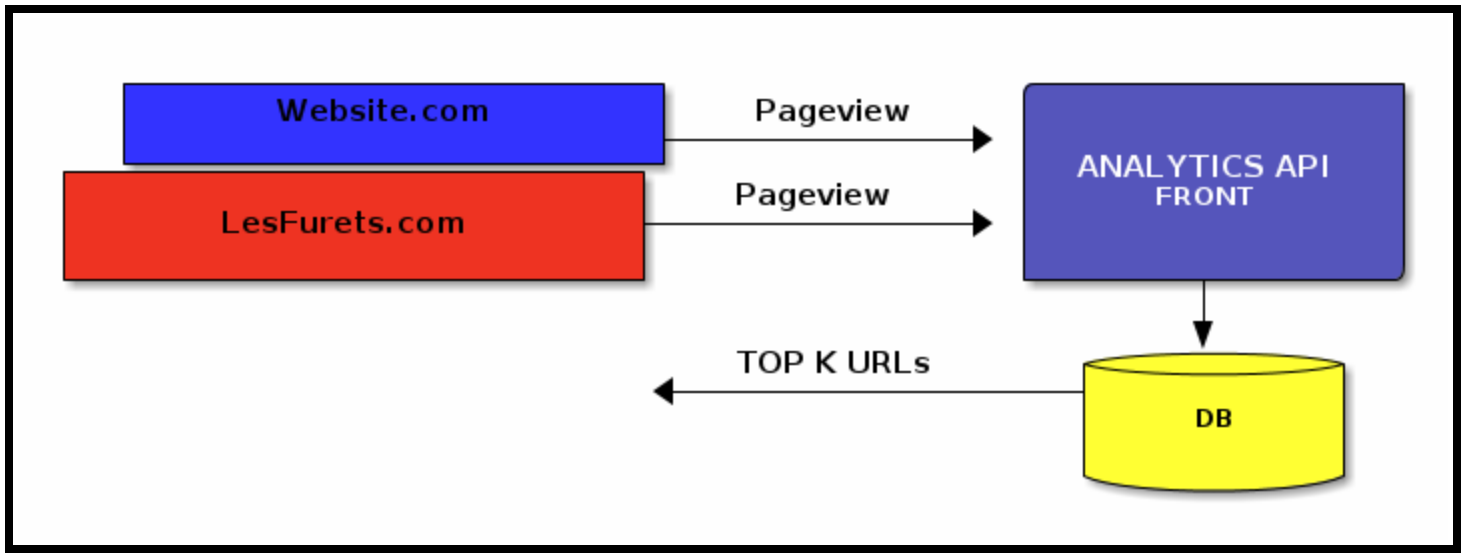
Lightning fast genomics with Spark Adam and Scala

Building a simple web analytics application

Bigdata by Nathan Marz

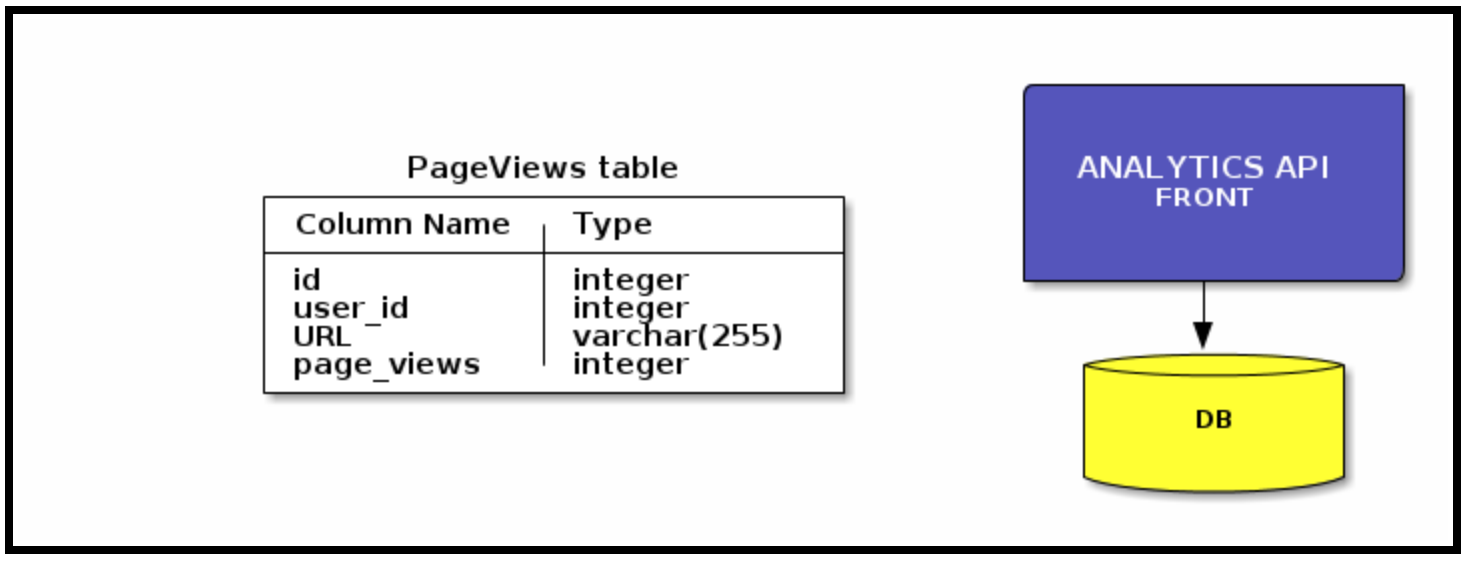
Simple web analytics application

- track the numbre of pageviews for each URL
- what are the top 100 URIs



Simplest architecture

- track the numbre of pageviews for each URL
- what are the top 100 URIs



Queries

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

Insert pageviews

PageViews table

Column Name	Type
id	integer
user_id	integer
URL	varchar(255)
page_views	integer

```
INSERT INTO PageViews VALUES
    (1,1,"http://www.lesfurets.com/index.html",1);
INSERT INTO PageViews VALUES
    (2,2,"http://Website.com/base.html",1);
INSERT INTO PageViews VALUES
    (3,1,"http://www.lesfurets.com/assurance-auto",1);
```

Update pageviews

PageViews table

Column Name	Type
id	integer
user_id	integer
URL	varchar(255)
page_views	integer

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

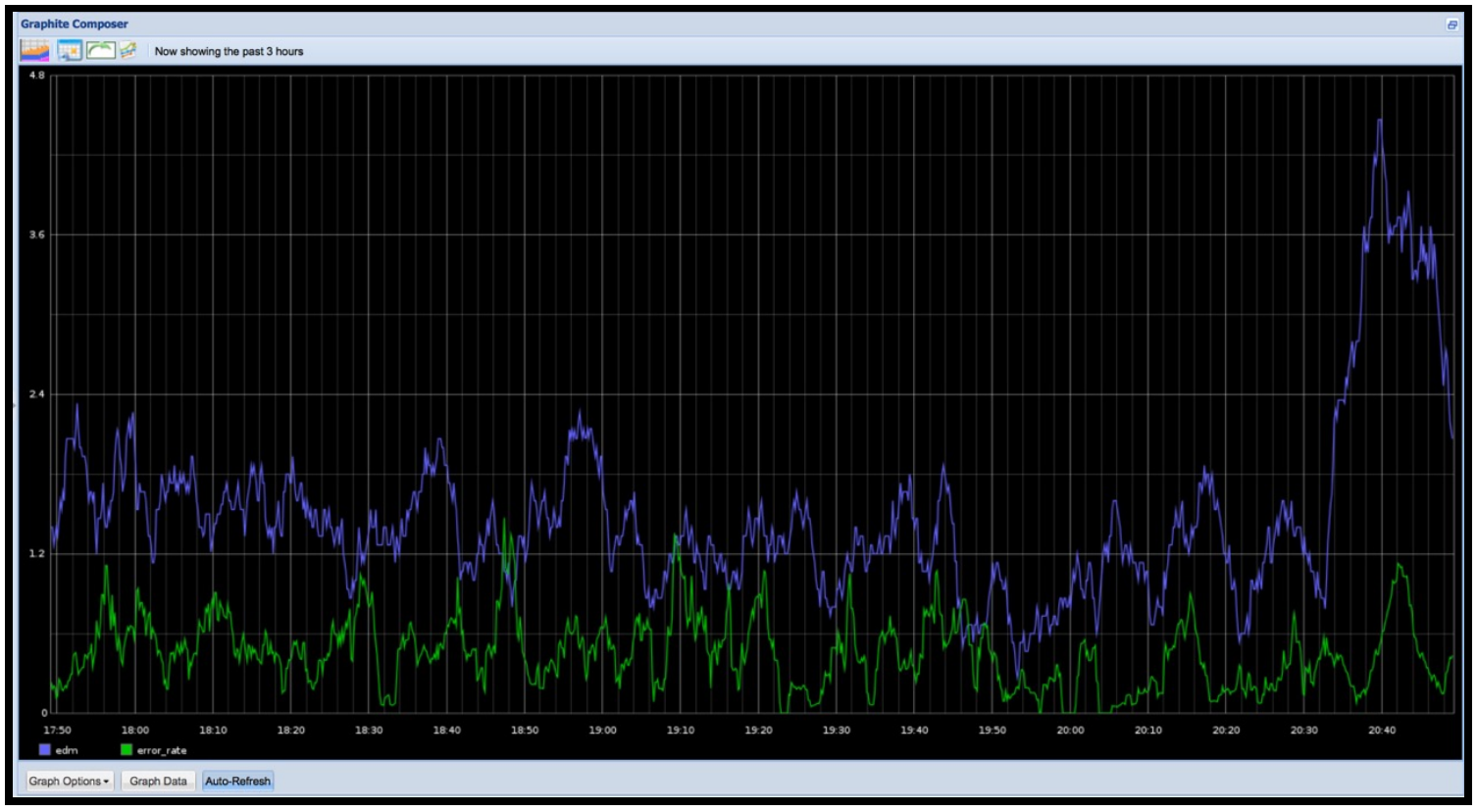
Top 100 URLs for a client

PageViews table

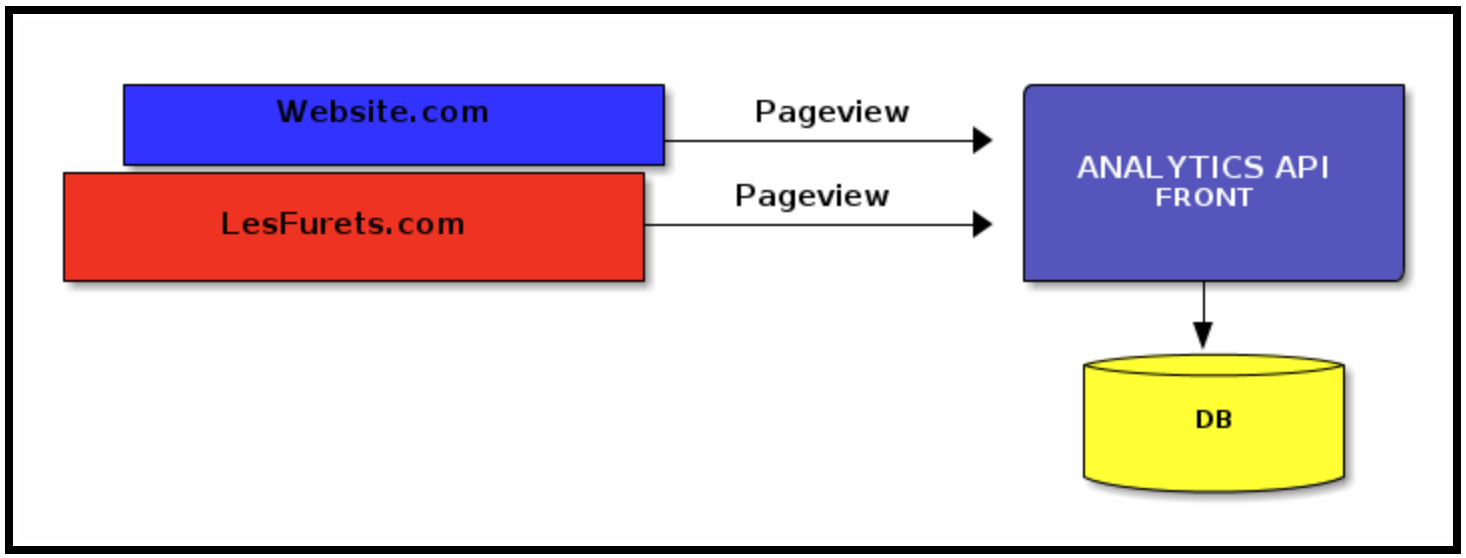
Column Name	Type
id	integer
user_id	integer
URL	varchar(255)
page_views	integer

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

Production load

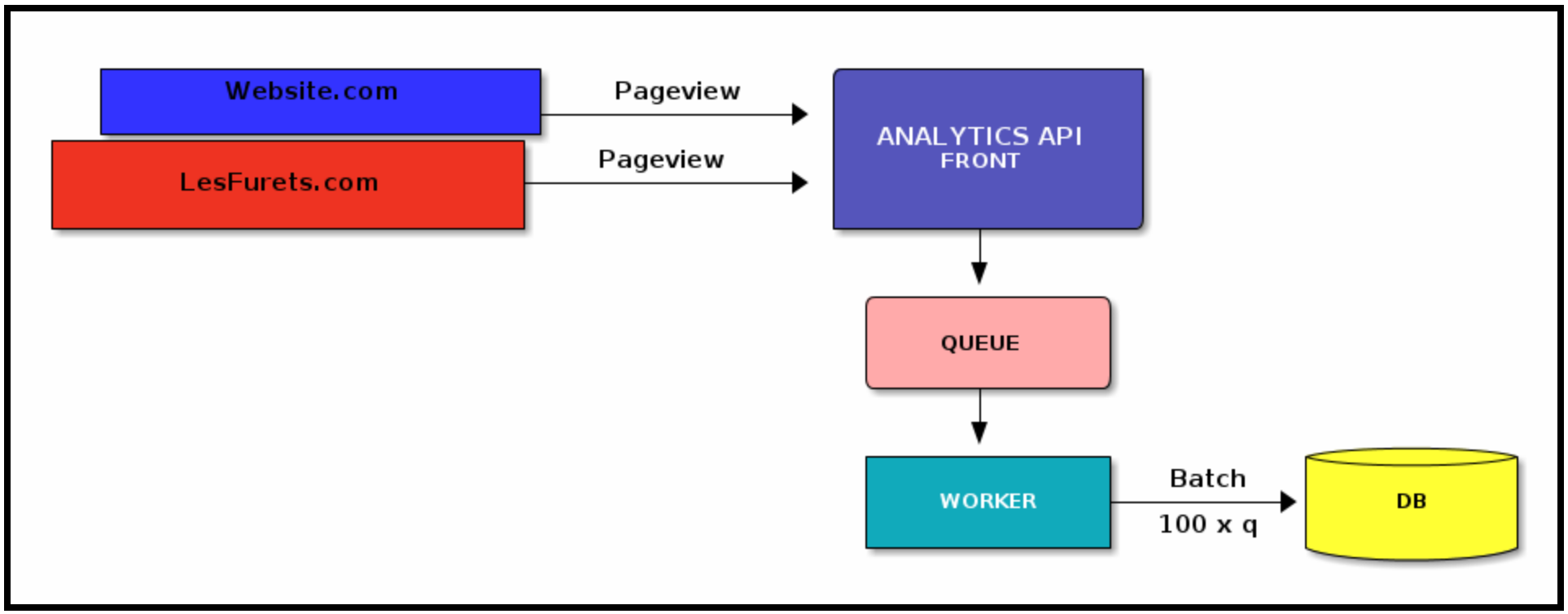


Timeouts



Timeout error on inserting/updating the database

Fix#1 Queuing



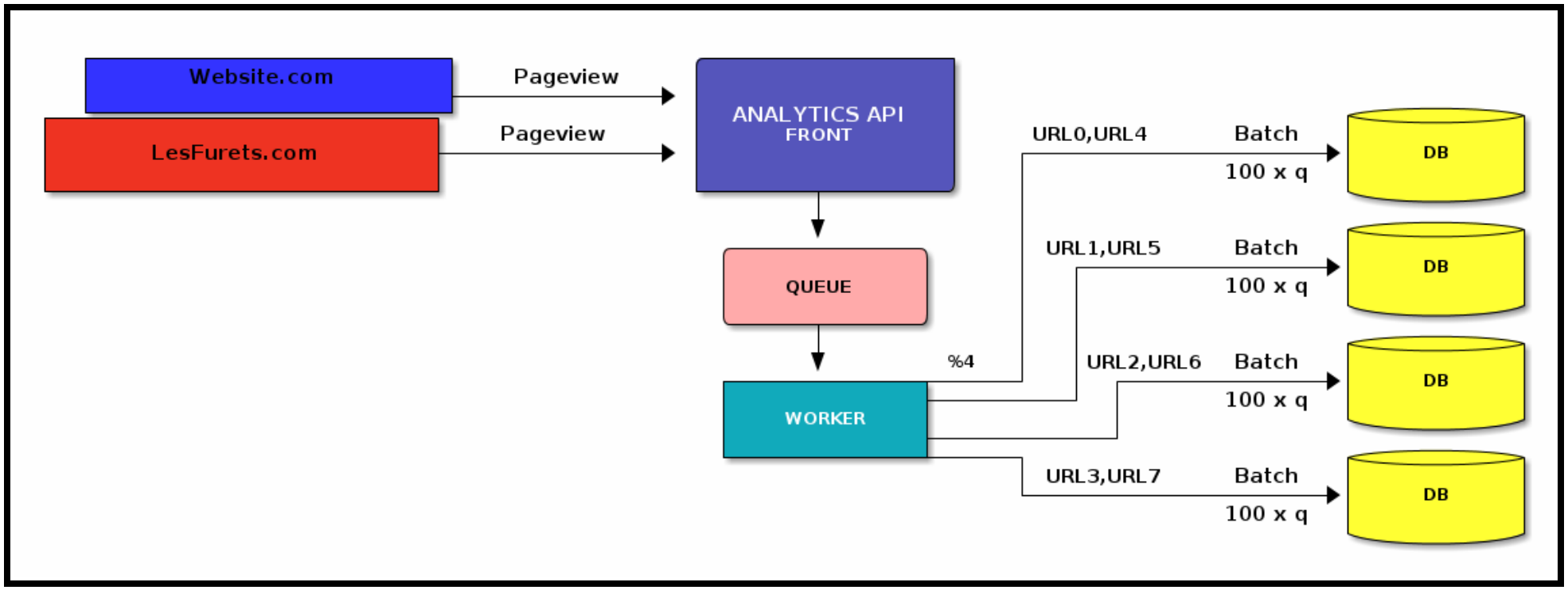
Fix#1 Queuing

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

Fix#2 Sharding / Vertical 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

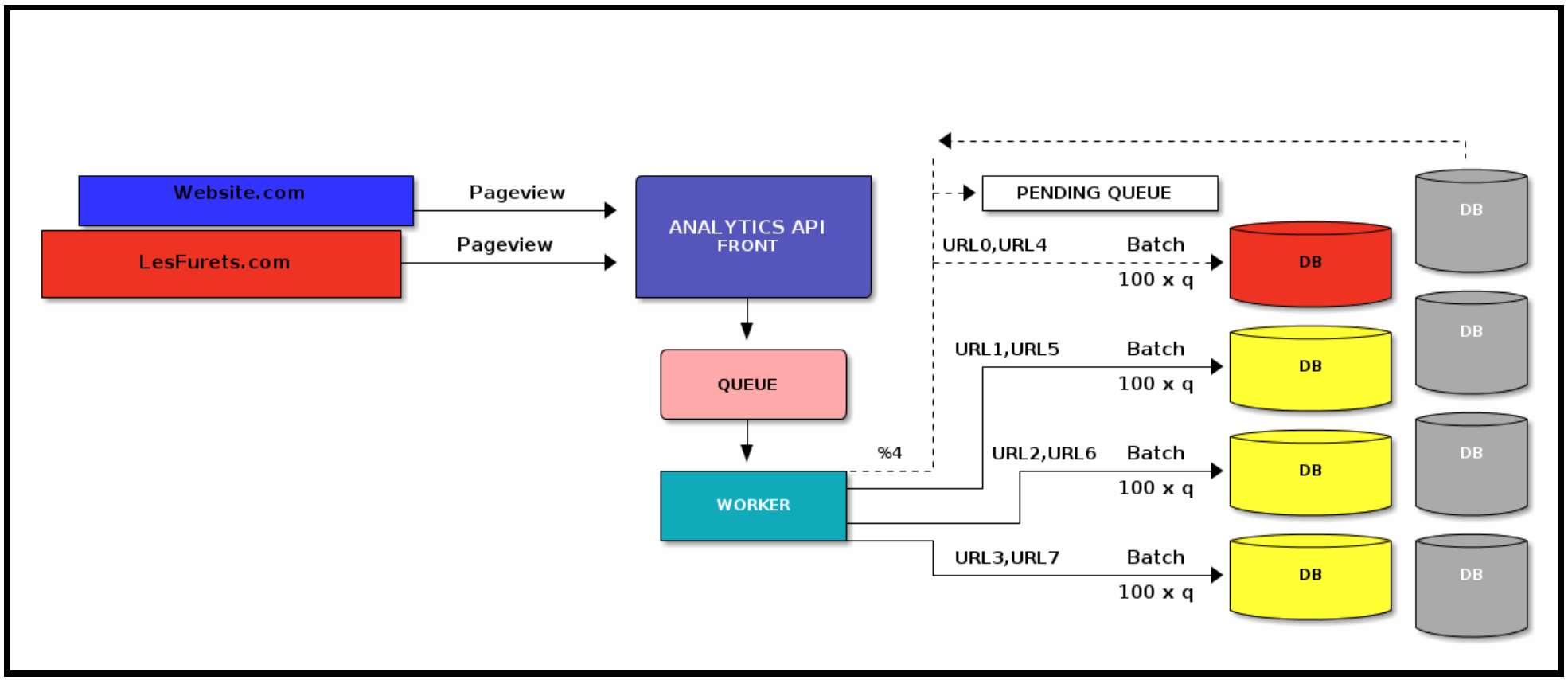
- Impacts
  - distribute the keys to the new servers
  - write to the "right" DB instance
  - agregate data from all the shards !
- **Sharding more and more**
  - new shards to follow the load
  - repeat the last steps

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

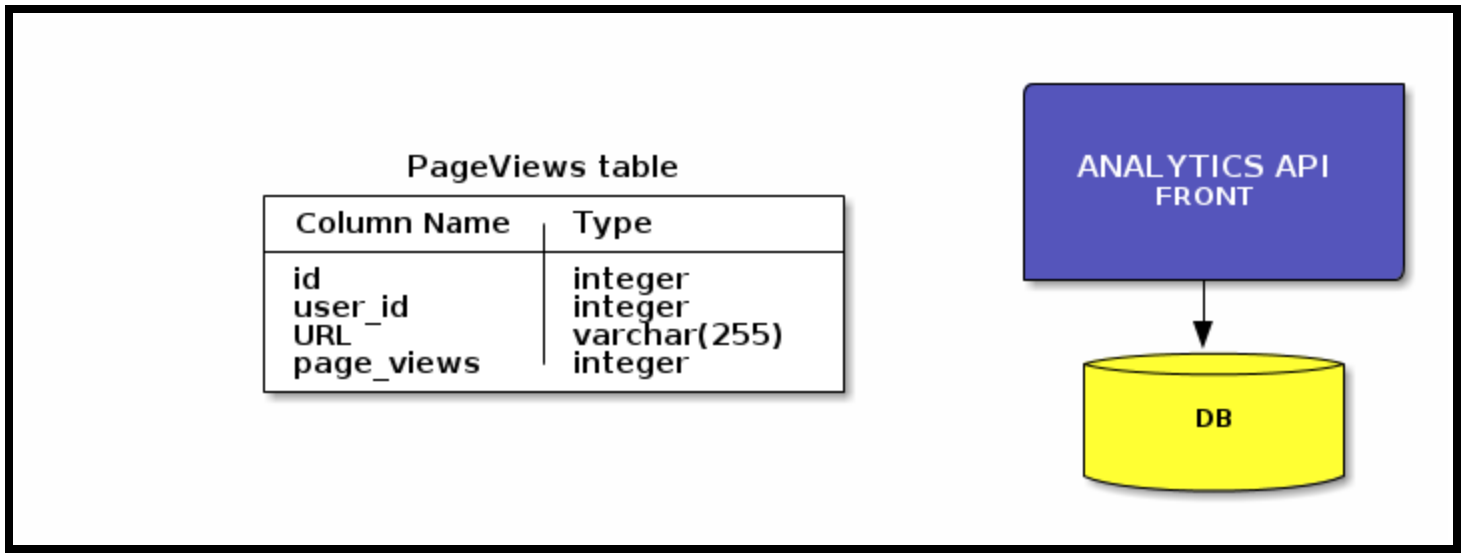
- **Data written to the wrong shards**
  - redistribute data to the right shard
  - while still accepting queries ?!

Human failures

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

Increments the number of pageviews + 2

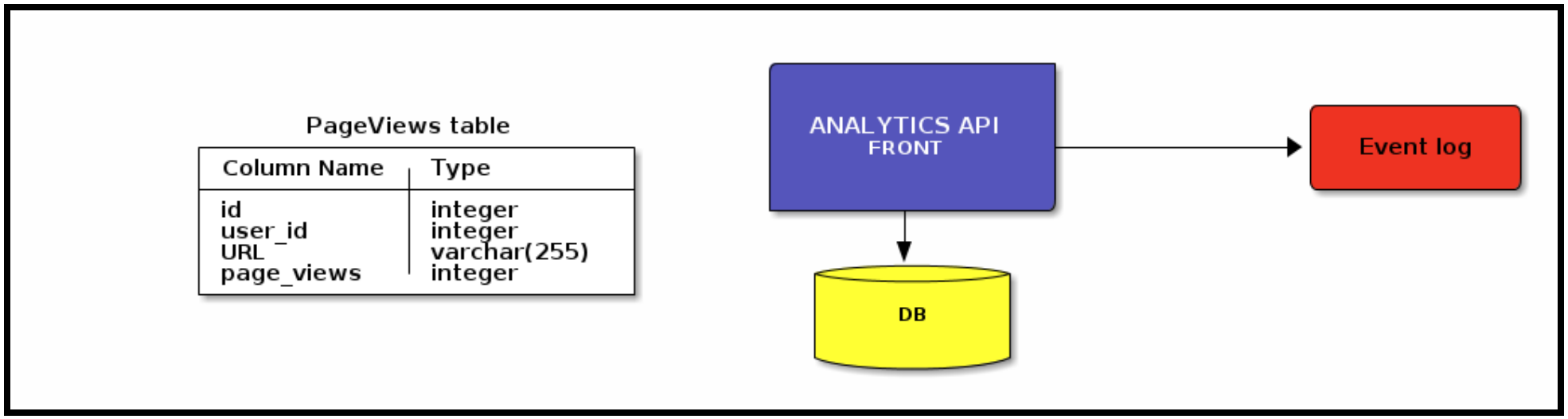
Human failures




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

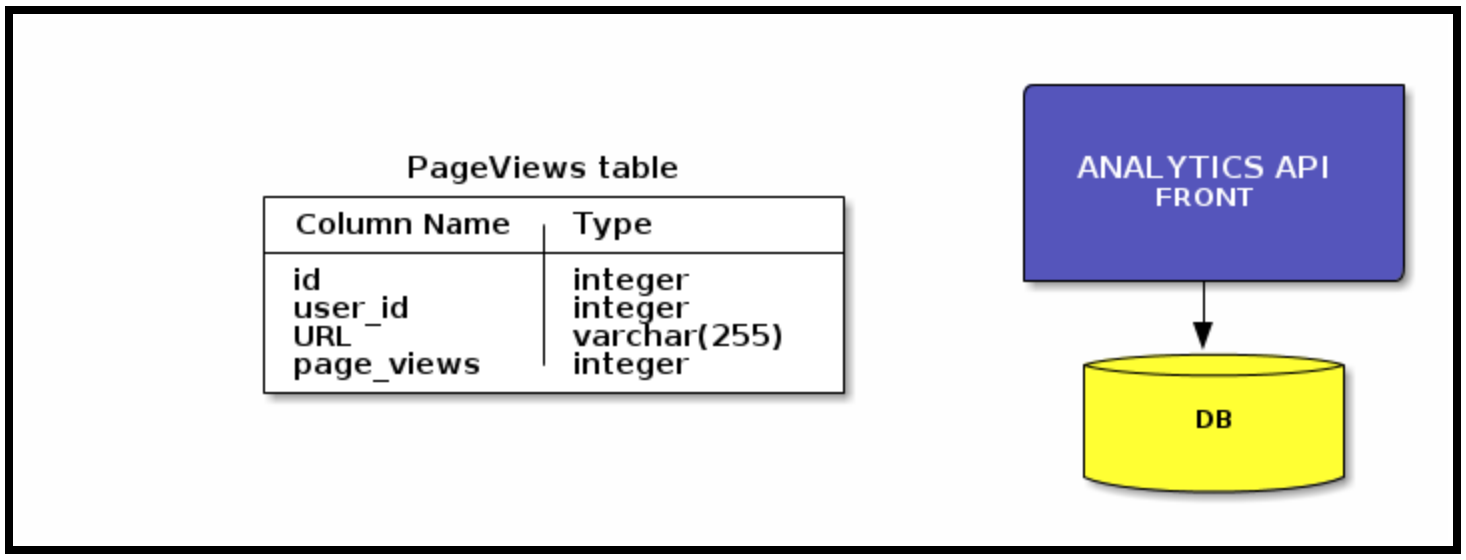
Human failures

- **event logging**



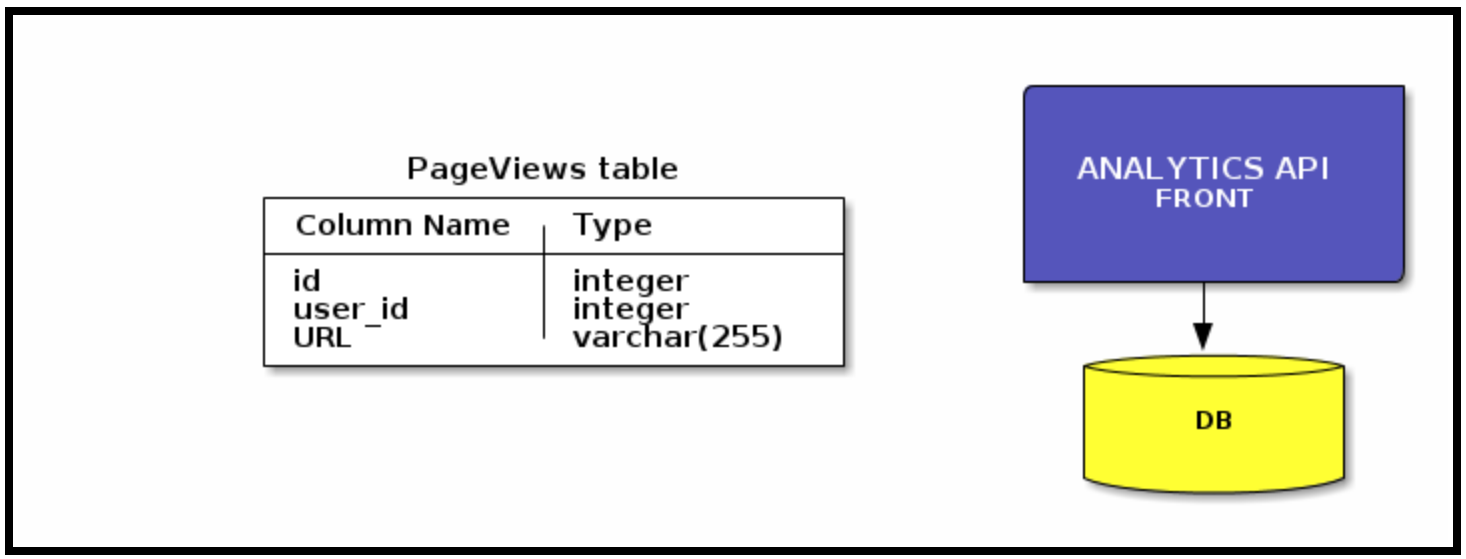
Human failures

- Incremental data model



Human failures

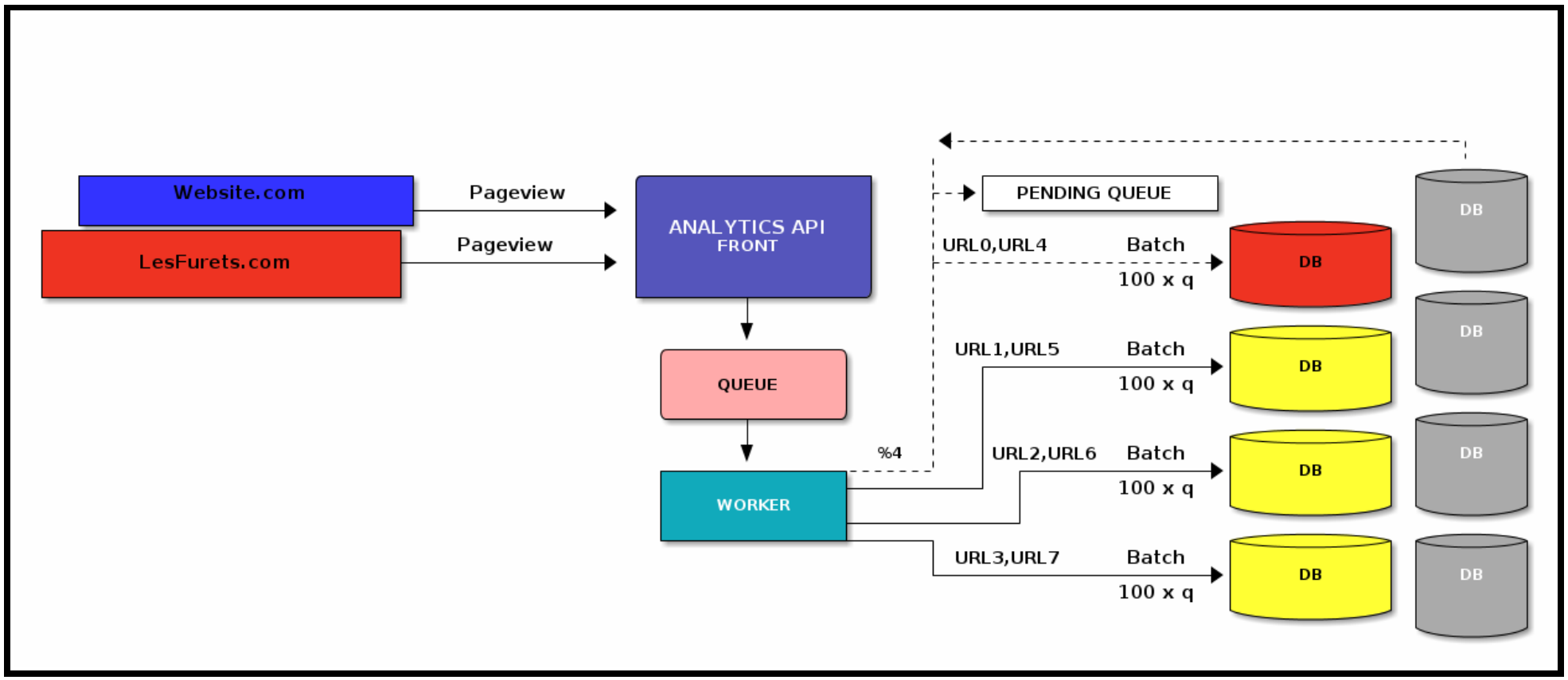
- Incremental data model ⇒ **immutable data model**



Human failures

- incremental data model ⇒ **data corruption**
- hard to correct

What went wrong?



- do I build new features for customers?
- or just dealing with reading/writting 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 (**scalling**)
  - transparent data distribution (**auto-sharding**)
  - handle failures (**auto-replication**)

⇒ **Distributed databases**: *Redis*, ***Cassandra***, ***MongoDB***, …

Wishlist 2 : Queries

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

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

Wishlist 3 : Data model

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

⇒ **Immutability**

Plan

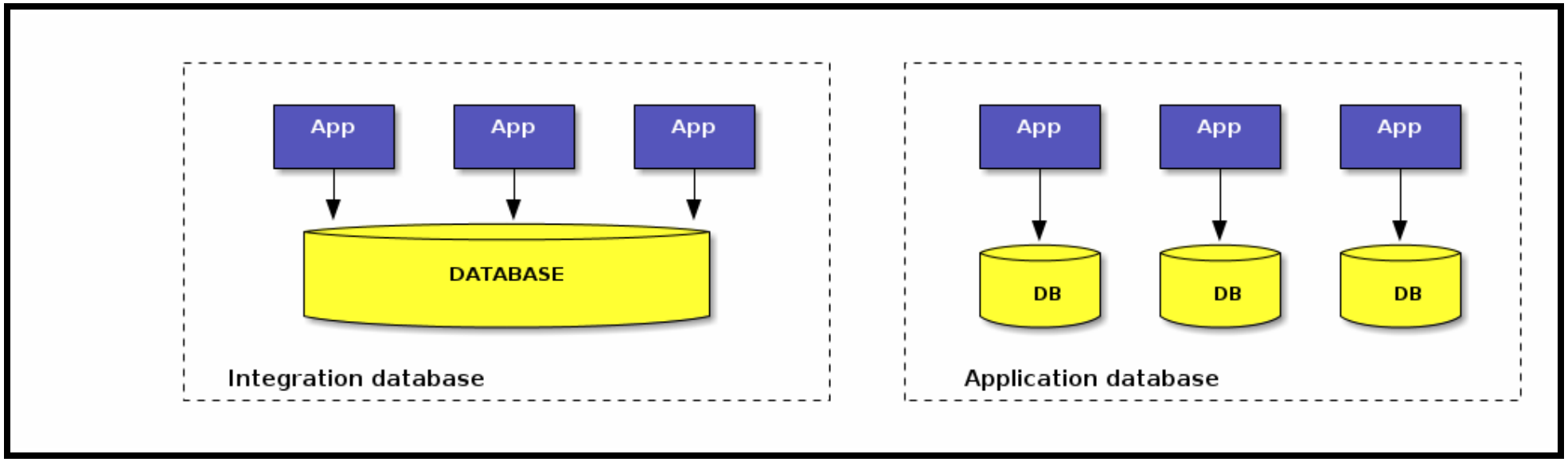
1. Intro
2. From SQL to NoSQL
    1. Scalling a simple application
    2. **From relational to non-relational databases**
3. Scalling MySQL : @LesFurets.com
4. PostgreSQL + TP

From relational to non-relational databases

- RDBMS (Relational database management systems)
- RDBMS/Memory : model missmatch and scalling concerns
- RDBMS : modeling complex data

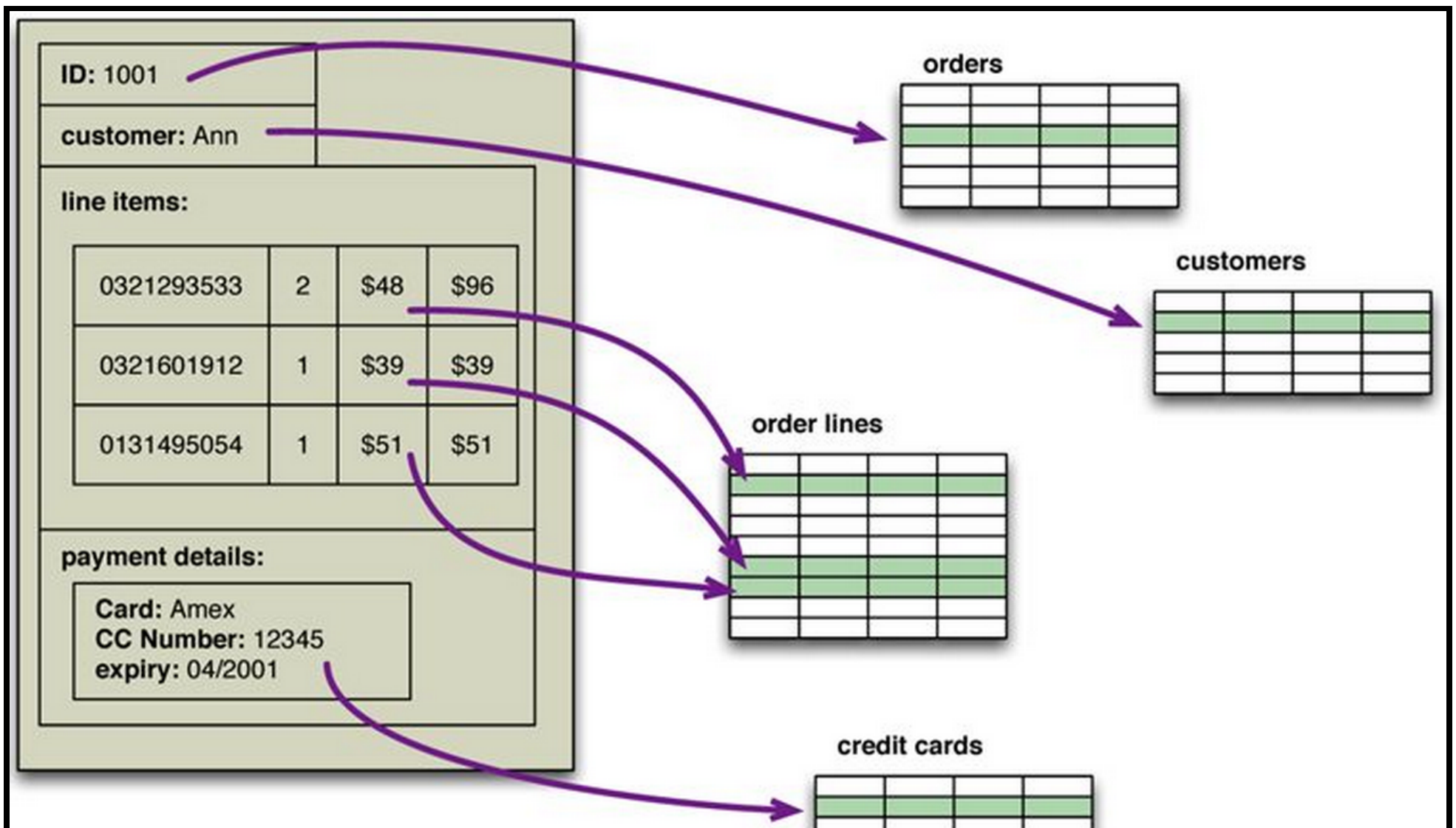
RDBMS: the good parts

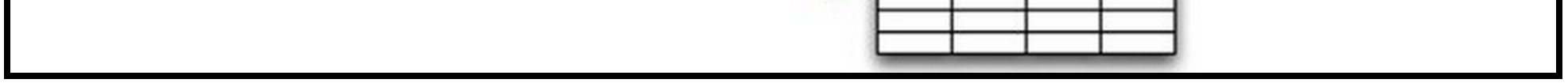
- simple model with sound matemathical properties (ACID)
- 45+ years of research (Codd '70, System R '74, Oracle '79)
- industry patterns : integration database / shared database pattern



Model missmatch (1)

- **Relational model**: relations / tuples + normalisation (avoid duplication)
- **Memory**: rich data structures !

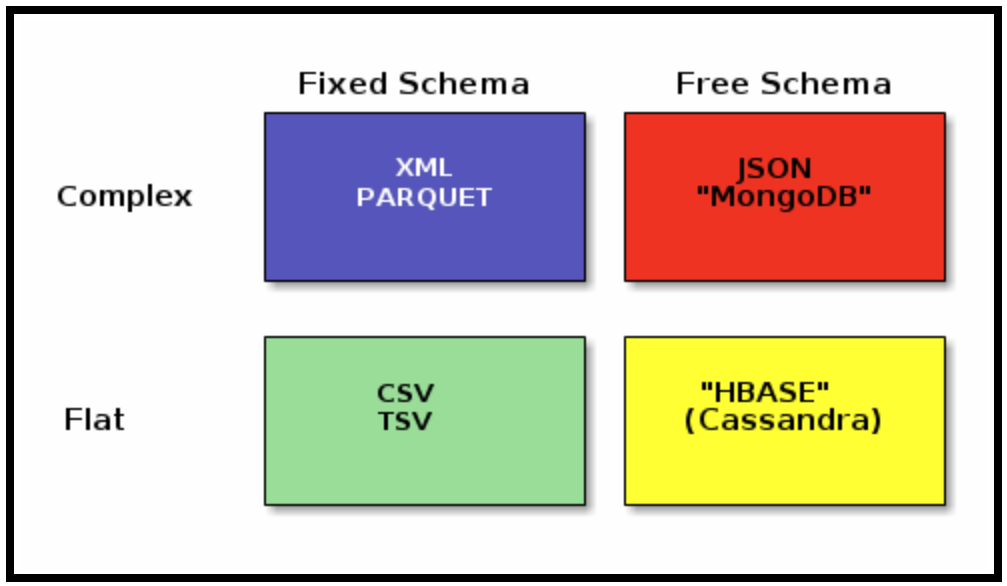




Model missmatch (2)

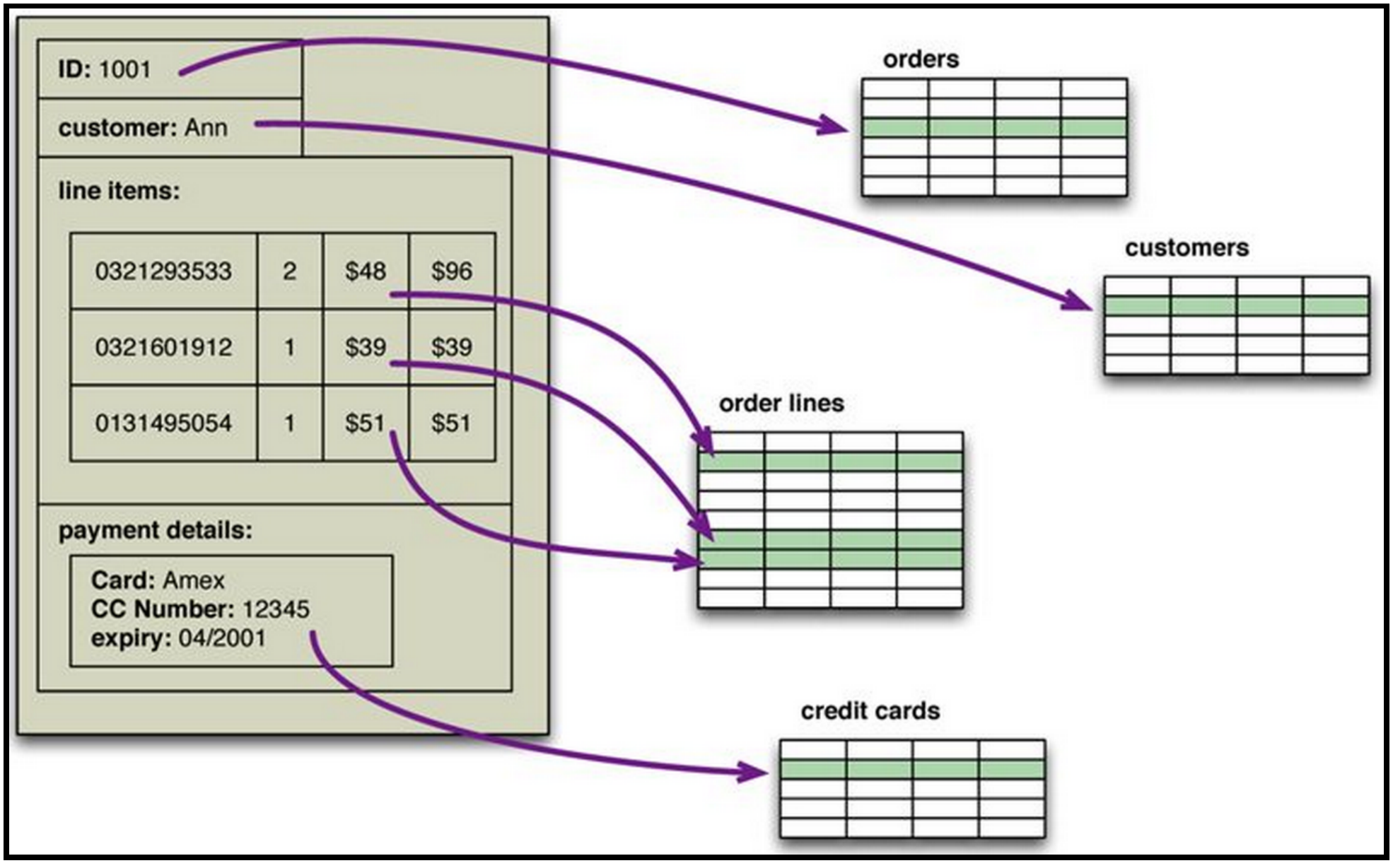
- **Memory** - object graphs
  - complex mapping from object to relations → ORMs
  - leads to performance issues
    - many rows/tables/**JOINS**
- Schema evolution:
  - adding an attribute → adding a whole column
  - expensive locks → application downtime

Modeling complex data (1)



Modeling complex data = model missmatch

- **Relational model**: relations / tuples + normalisation (avoid duplication)
- **Semi-structured** data hard to model (eg. XML, JSON):
  - blob vs name/value


ID: 1001
customer: Ann
line items:
0321293533
2
$48
$96
0321601912
1
$39
$39
0131495054
1
$51
$51
payment details:
Card: Amex
CC Number: 12345
expiry: 04/2001
orders
customers
order lines
credit cards

RDBMs: scalling

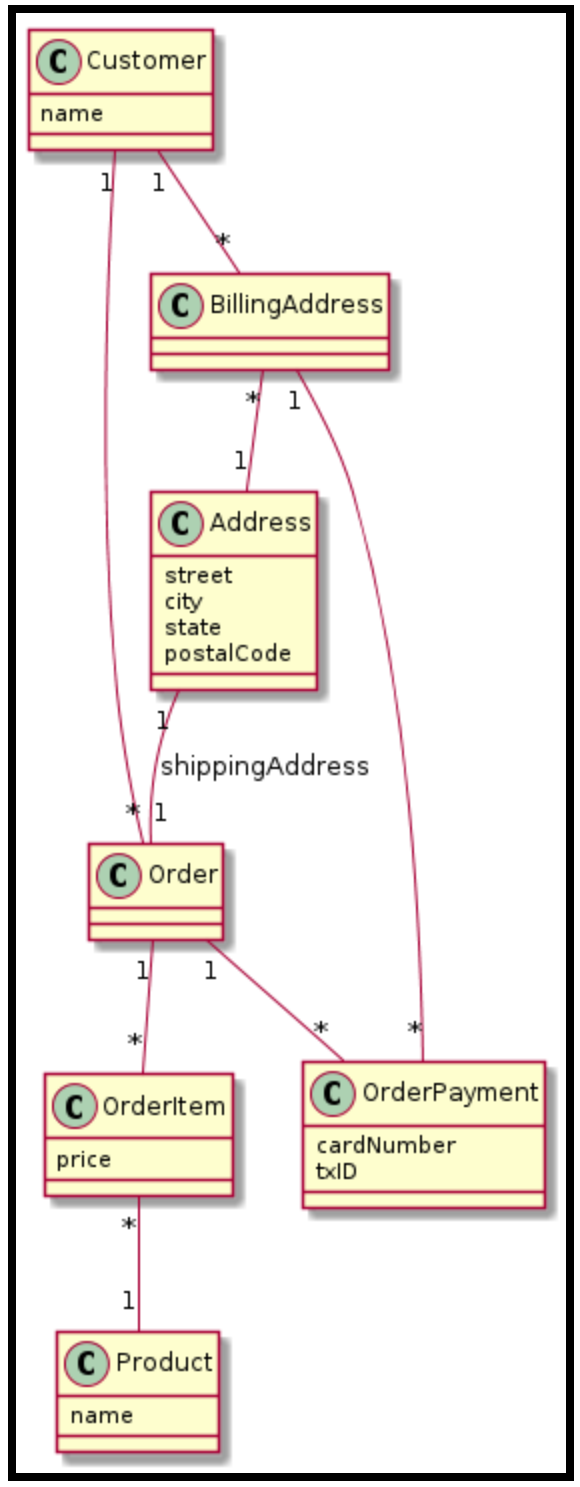
- **Scalling** is expensive:
  - scalling writes → locking + partitioning (sharding)
  - scalling reads → replication (latency, error recovery)

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

Not only SQL

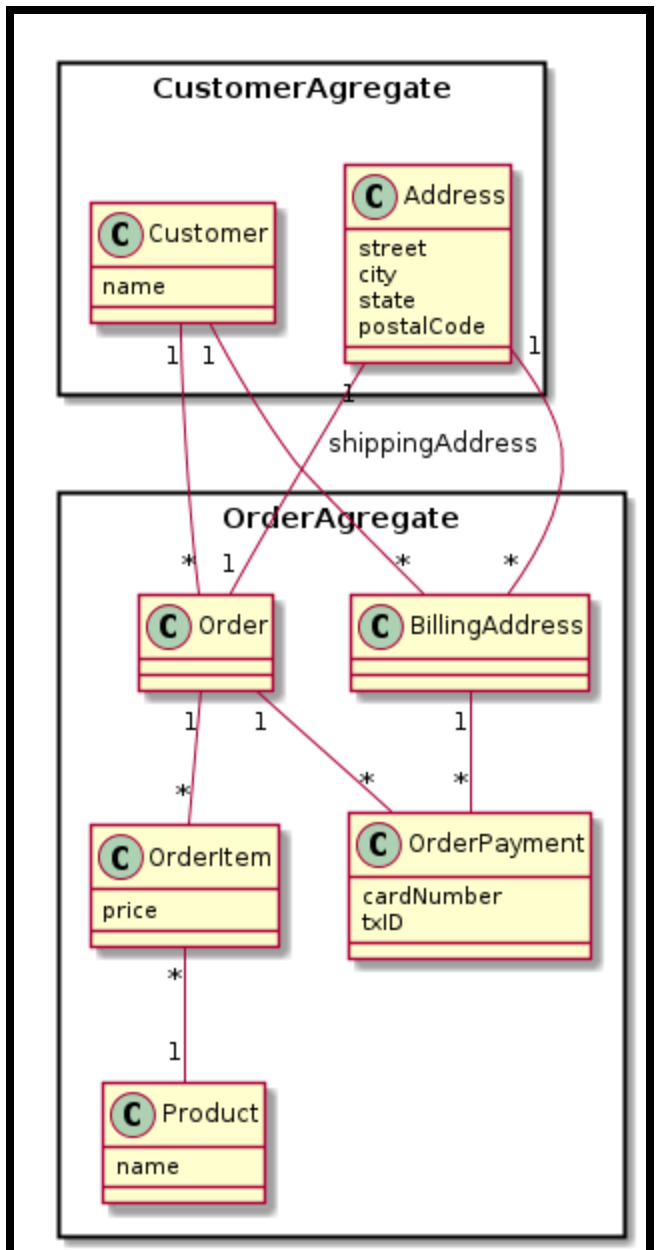
- 2009, Johan Oskarson, #NoSQL meetup for distributed, open-source, non-relational databases
  - not using relational model SQL
  - run on clusters
  - ACID ⇒ tunnable consistency
  - fixed schema ⇒ flexible schema
  - **polyglot persistance**

SQL models tuples and joins



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 to guvern them all
- *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

Model: HashTable (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

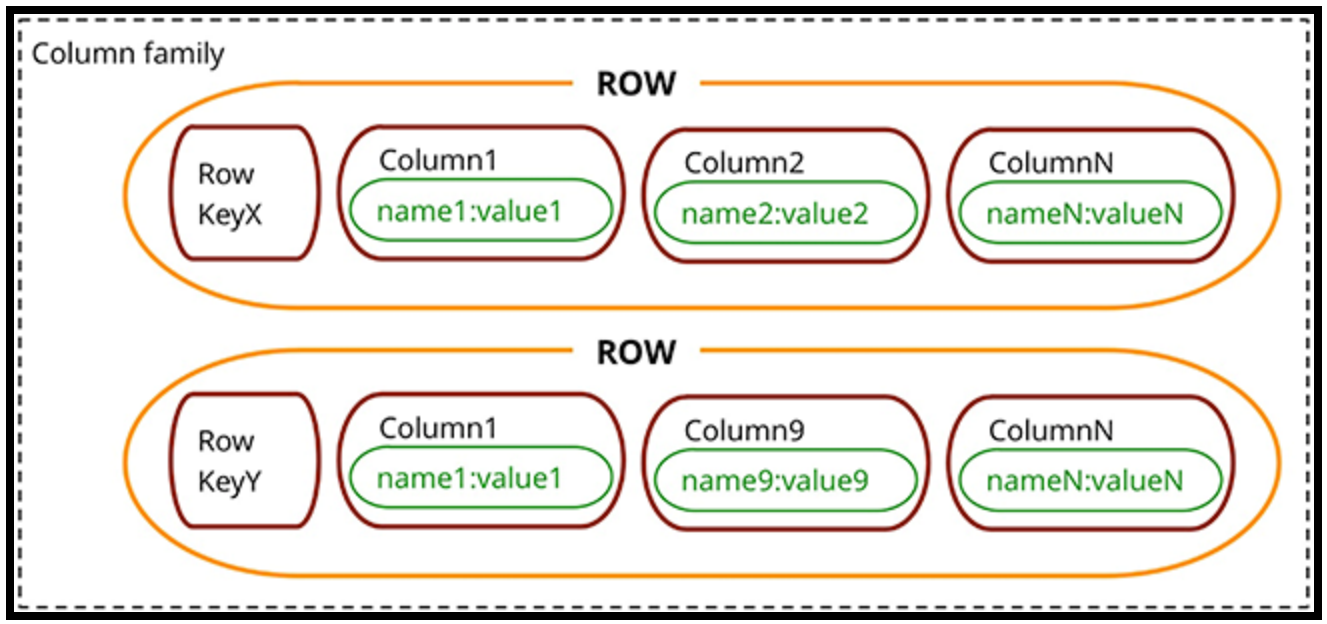
Model: key-value stores where the value is queryable

Use case: CMS, Product catalog, ...

Column oriented databases

- Stores data in **column families** as rows that have many columns associated to a row key
- Free format: two rows can have different columns

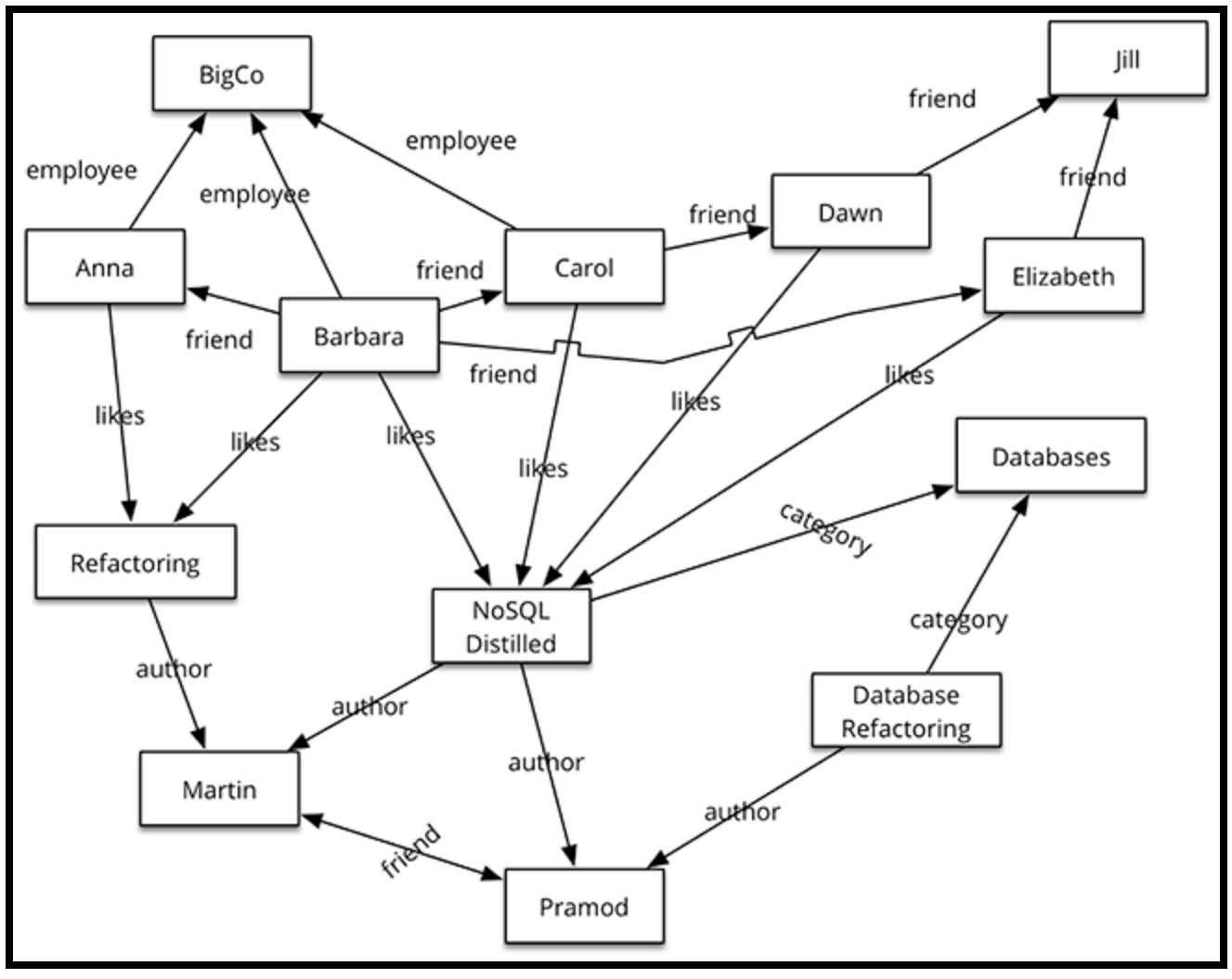
"Map of map model (rowId → (columnName → columnValue))"



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

Graph databases

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



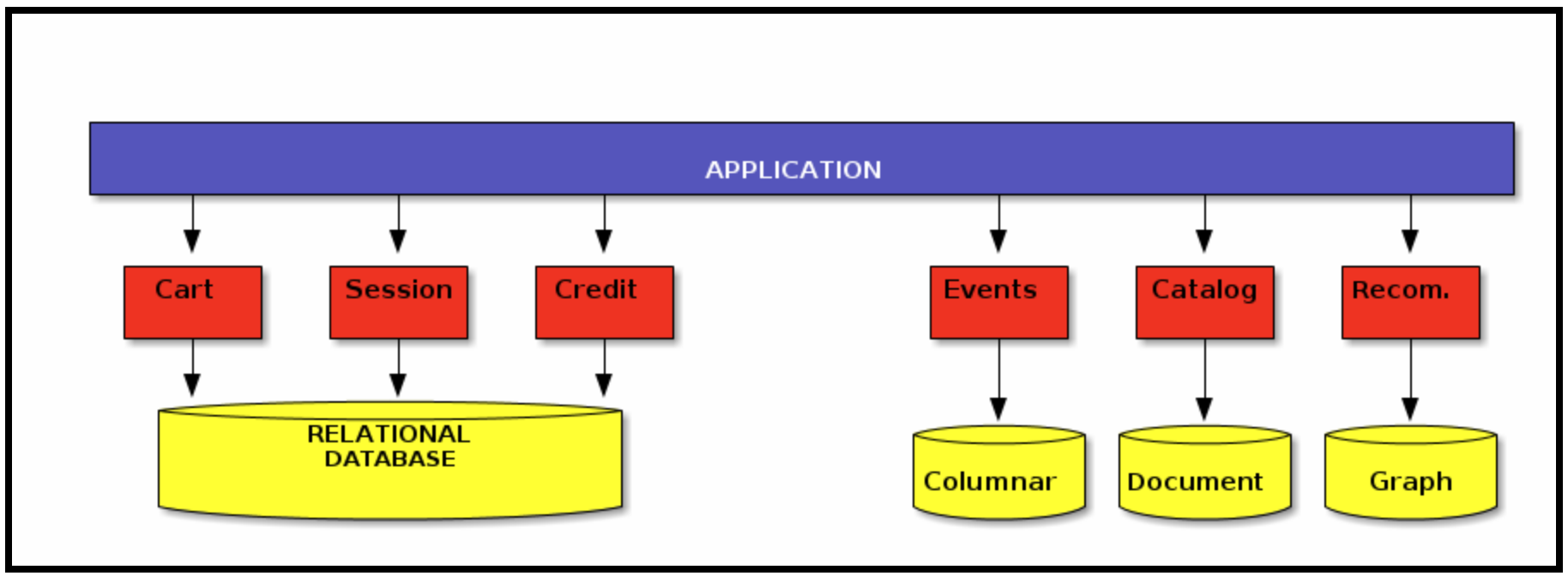
Use cases: Social networks, recommendation engines

NoSQL not a panacea

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

Polyglot persistance

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



Plan

1. Intro
2. From SQL to NoSQL
3. **Scalling MySQL : @LesFurets.com**
4. PostgreSQL + TP

Scalling MySQL

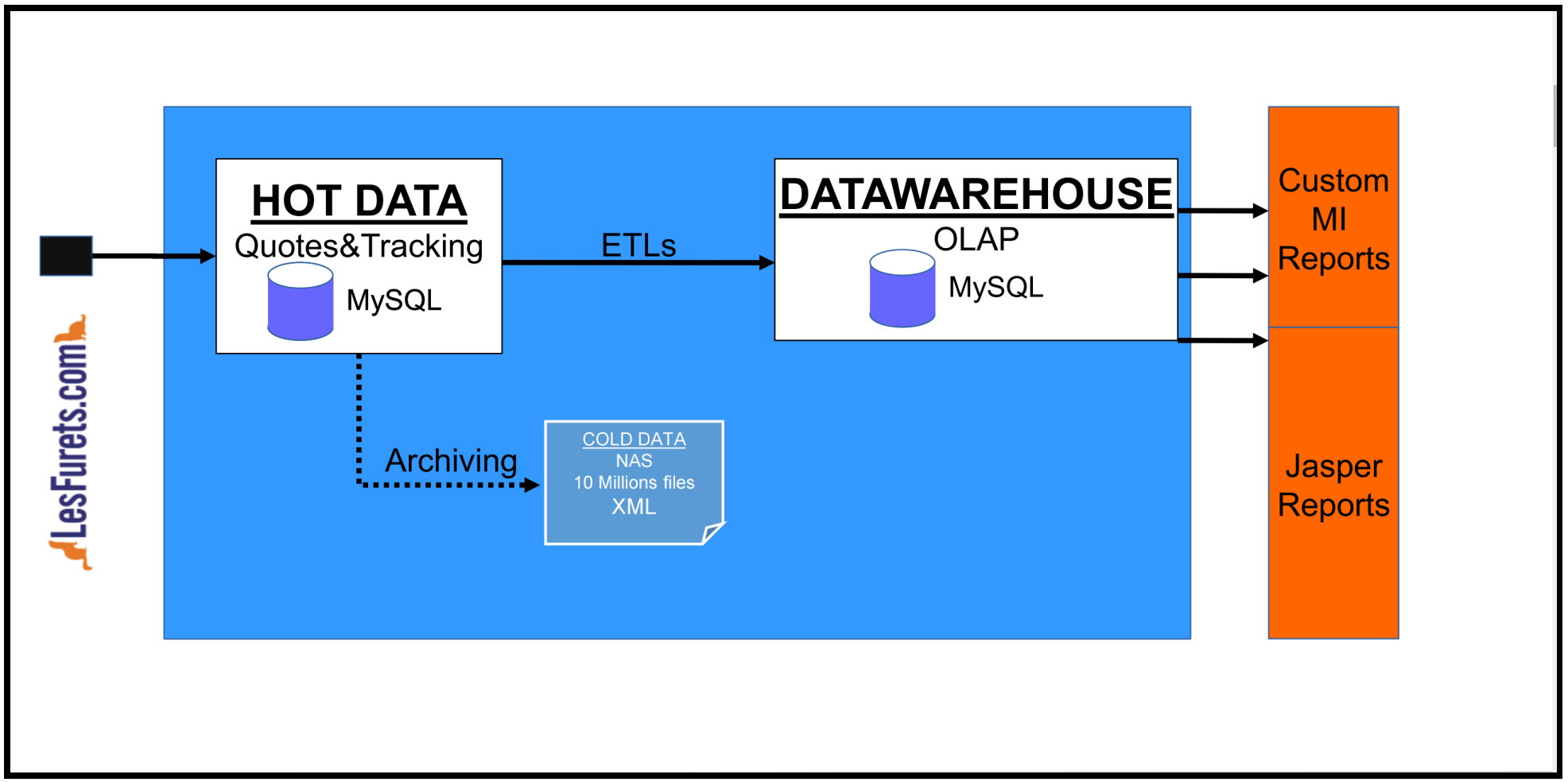
LesFurets.com

- Independent insurance agregator
- Persistance:
  - replicated MySQL db ⇒ DRBD ⇒ Galera Cluster
  - Cassandra / Spark (migration ongoing)
- BI/reporting
  - Olap (JasperReports)
  - QlikView dashboarding
  - Spark/Zeppelin over MySQL/Cassandra: ad-hoc analyses

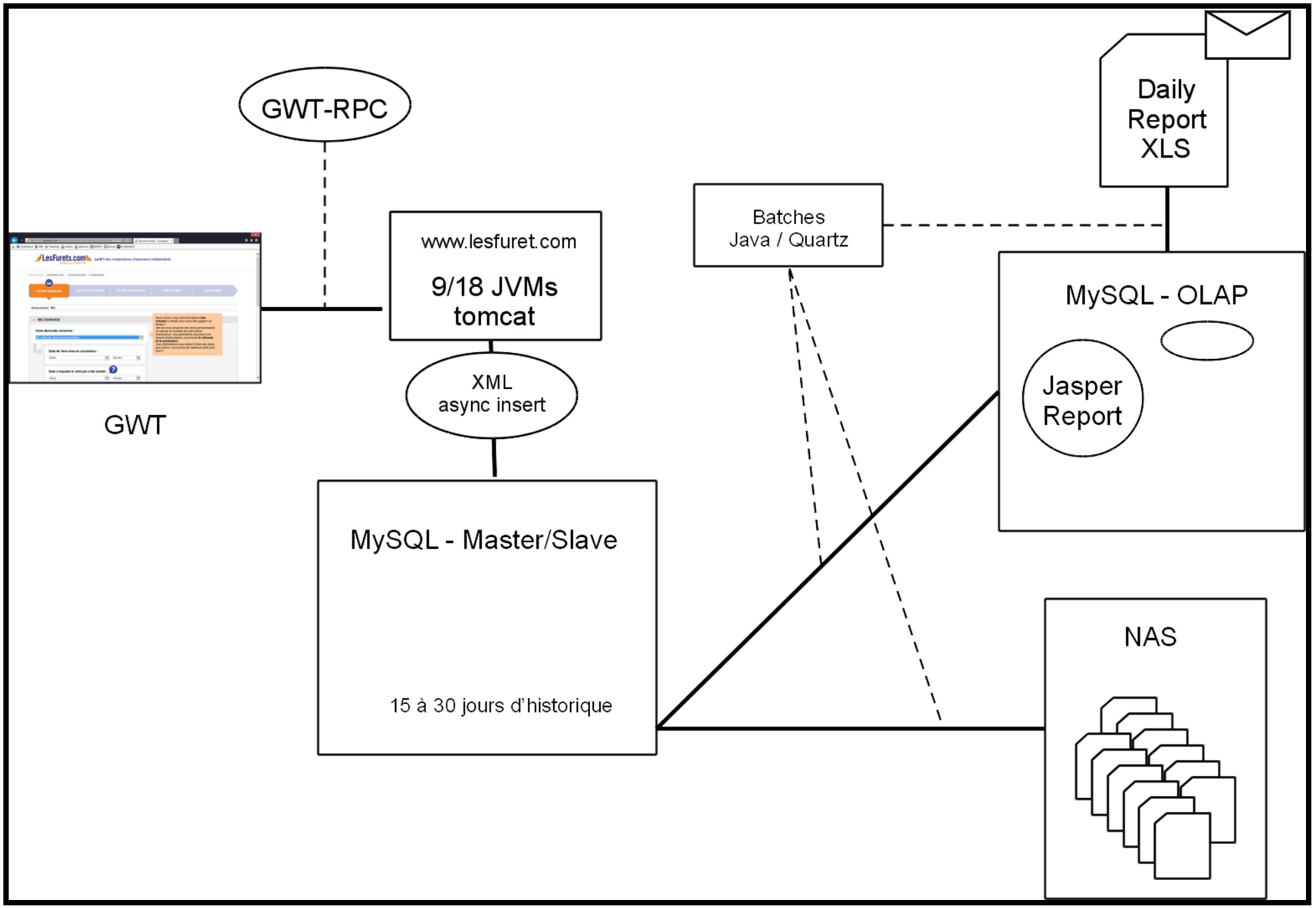
Data caracteristics

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

Data workflow



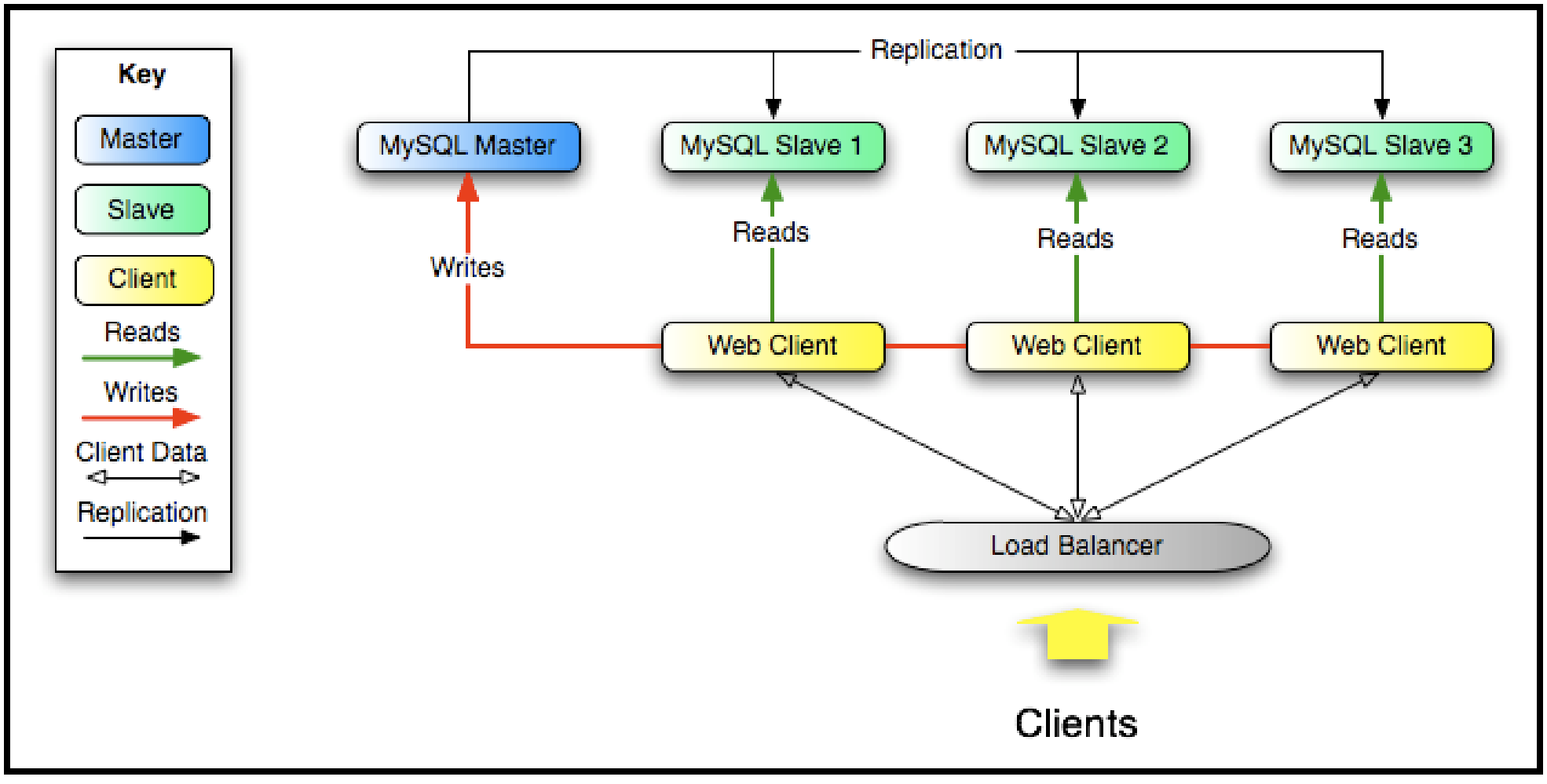
Architecture



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



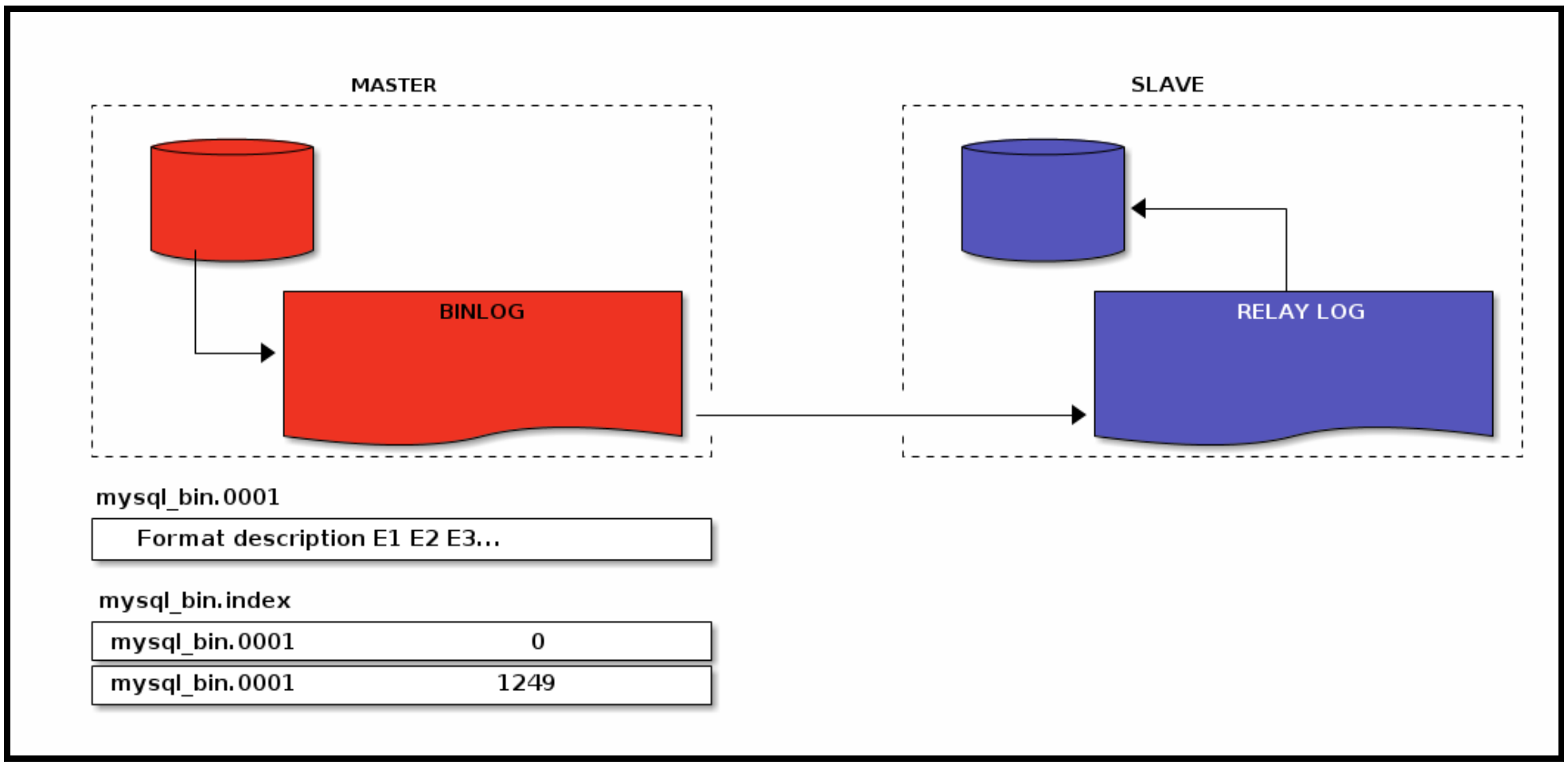
MySQL replication properties

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

MySQL replication

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

MySQL replication: binlog



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

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

RBR

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

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

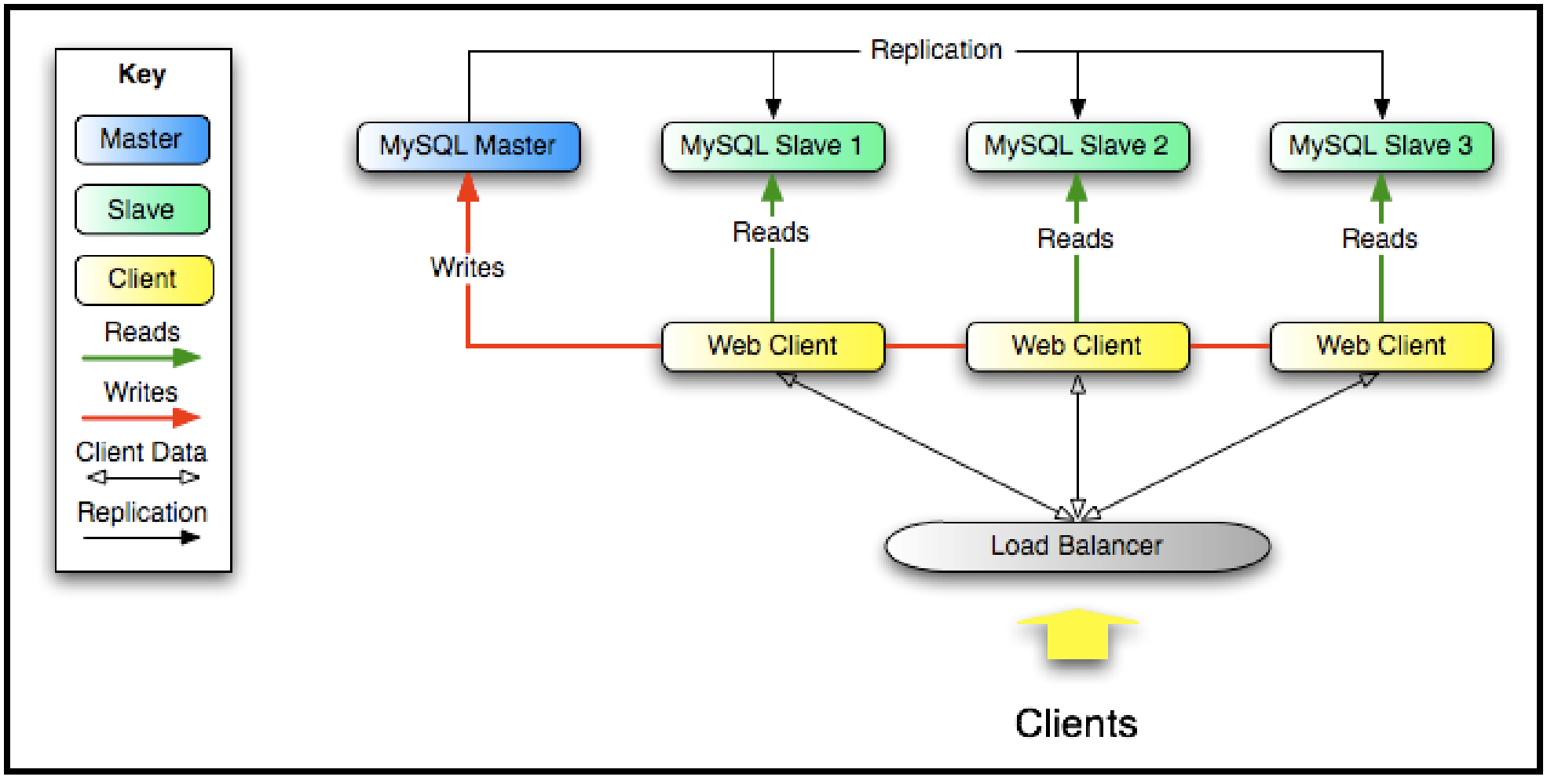
Semi-synchronous replication

- 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

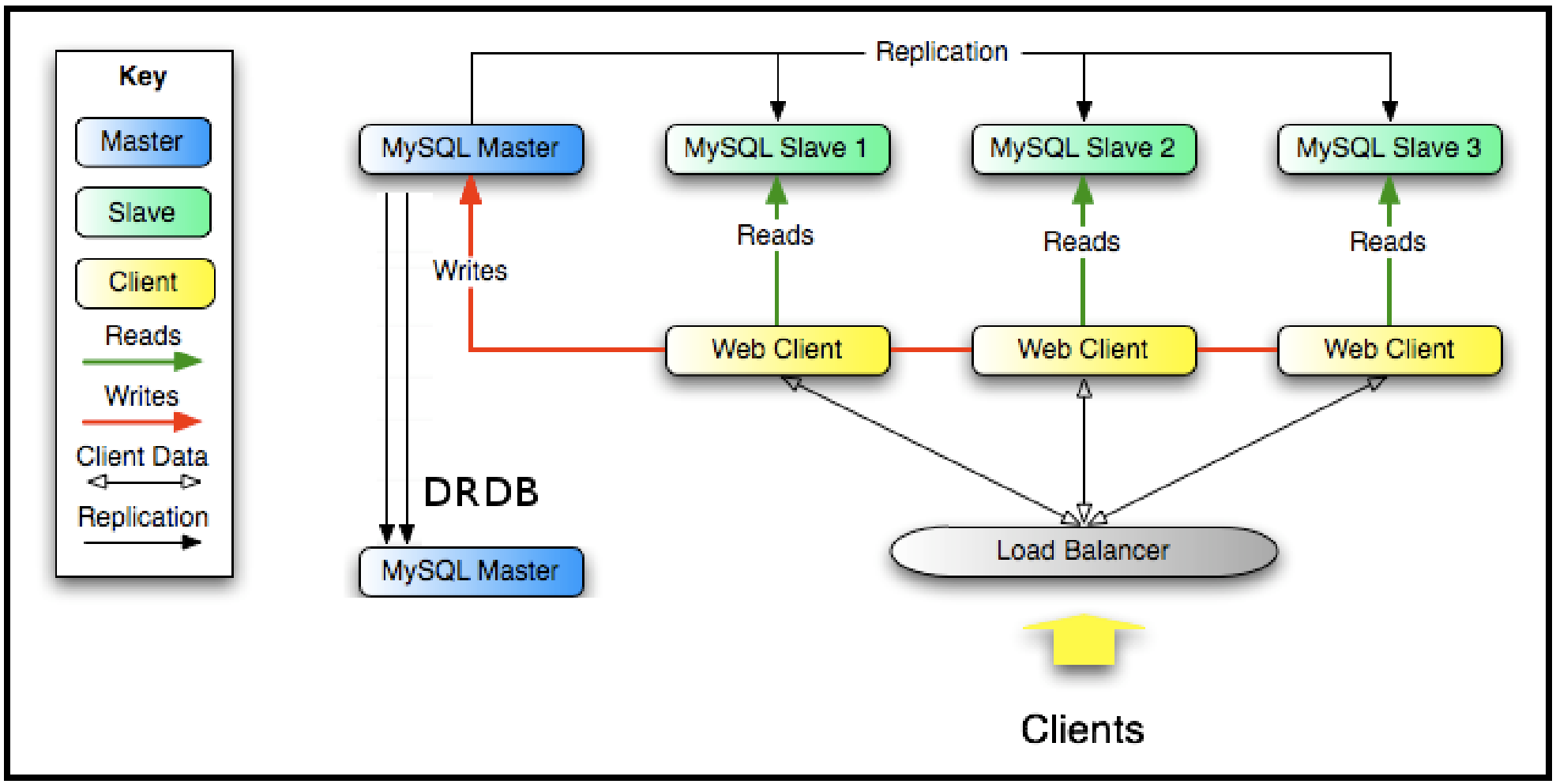
Crash recovery (M)

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

Master/Slave

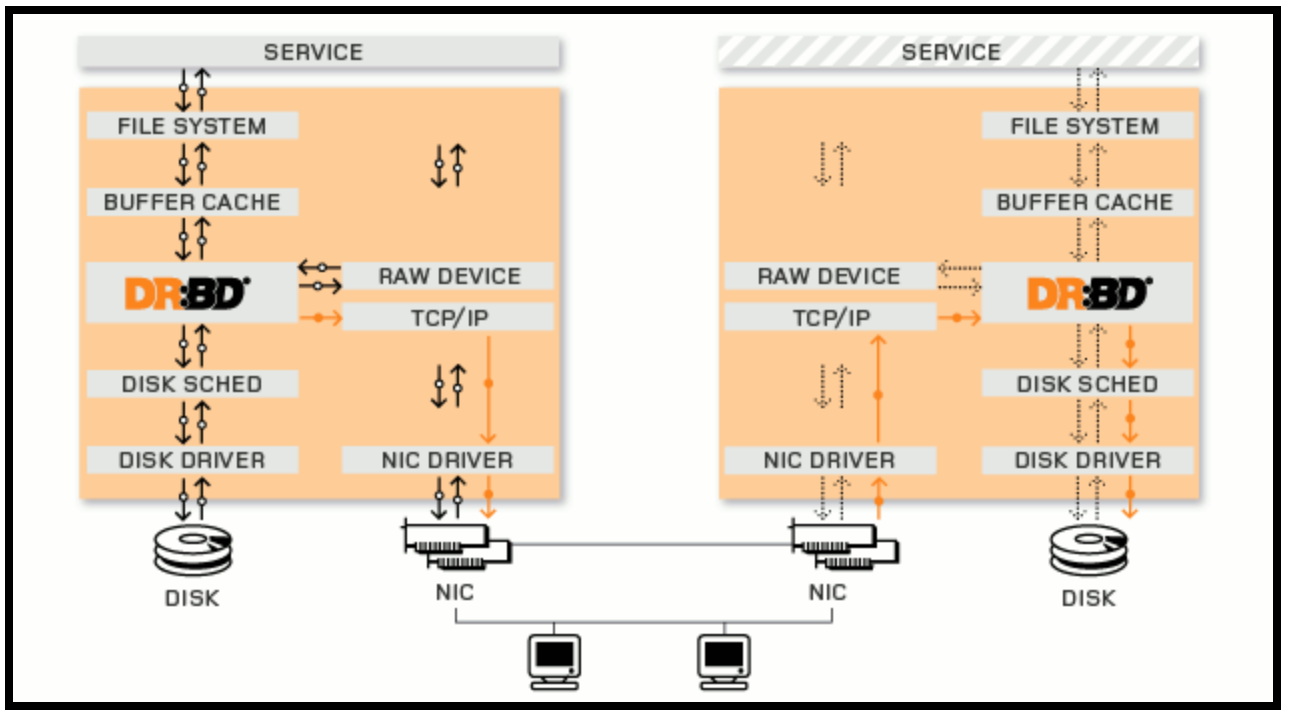


Multi - Master replication with DRBD



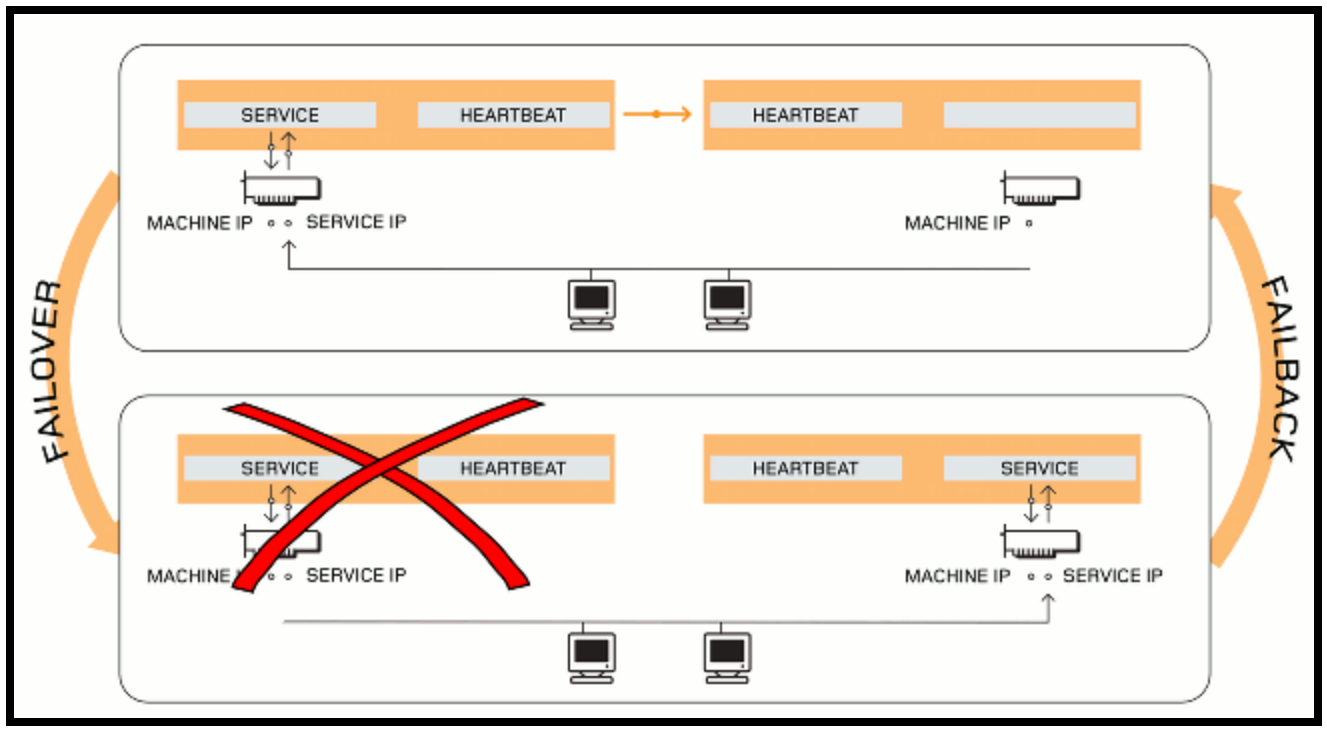
DRBD mirroring

- mirroring a linux partition over IP (sync/async)



DRBD HA

- heartbeat protocol monitors failures
- triggers service switch via IPFOs



more...

DRBD recovery

- node(s) outage
  - background sync (most up-to date node if both were down)
    - replication network outage
      - automatic recovery
    - storage subsystem
      - mostly transparent
- network partition
  - ***split brain!*** both nodes switched to the primary role while disconnected
  - ***Manual intervention needed***

Scalling Writes

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

Galera Cluster

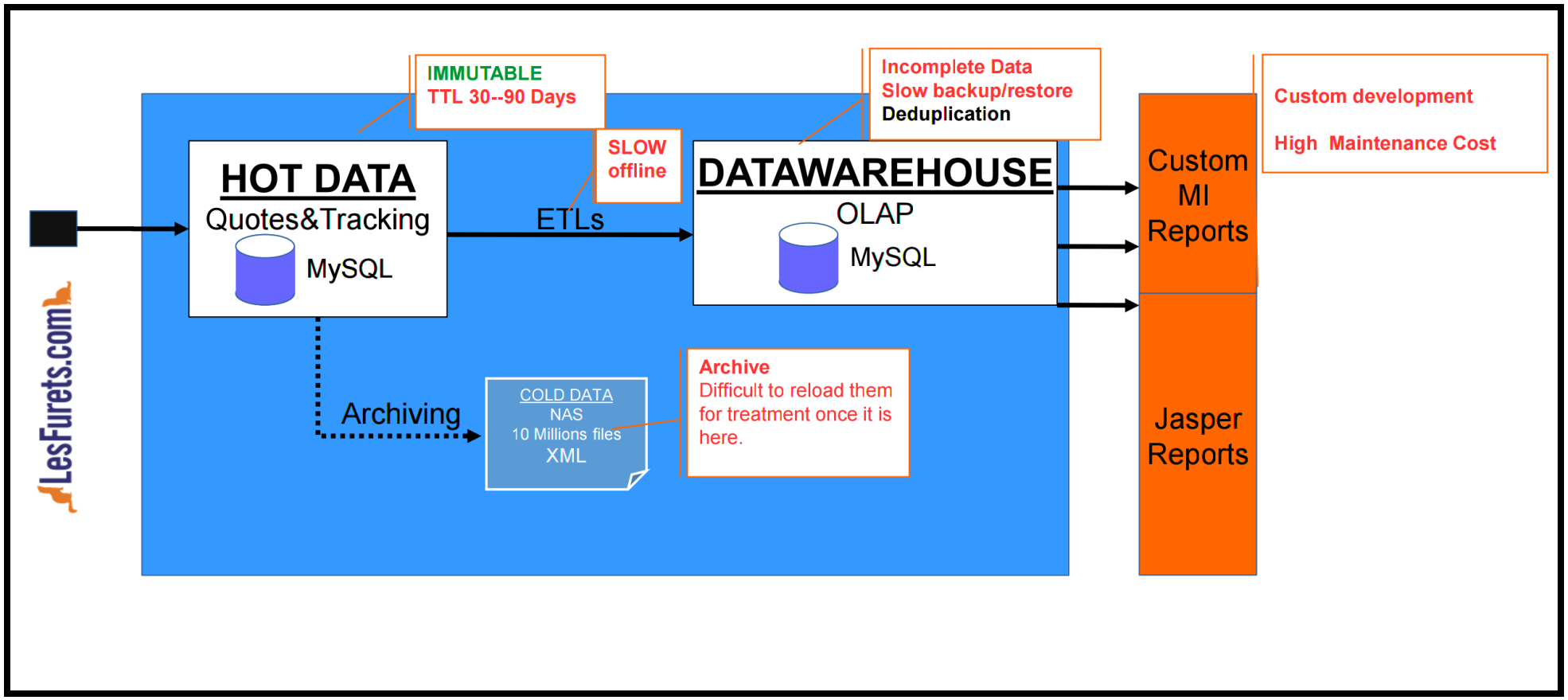
- multi-master: R/W at any node
- synchronous replication (tx commit)
  - traditonally implemented by distributed locking (2PH commit)
    - slow and complex
  - Galera ⇒ Certification Based Replication Method
    - good latency with increased consistancy (*Don't be lazy be consistent paper*)

      - Easy migration from MySQL
        - combine with MySQL binlog replication
        - automatic node provisioning (XtraBackup)
        - transparent to applications*

LF Mariadb Galera Cluster

- 3 nodes cluster (3M)
- relaxed ACID (read after write)
- wsrep_sync_wait: ensures synchronous read view before executing an operation of the type specified by bitmask:
    1. READ (SELECT, SHOW & BEGIN/START TRANSACTION);
    2. UPDATE and DELETE;
    3. INSERT and REPLACE.
- Benefits
    - recover from failures ⇒ < 1s
    - typical replication lag < 10ms

Current architecture concerns



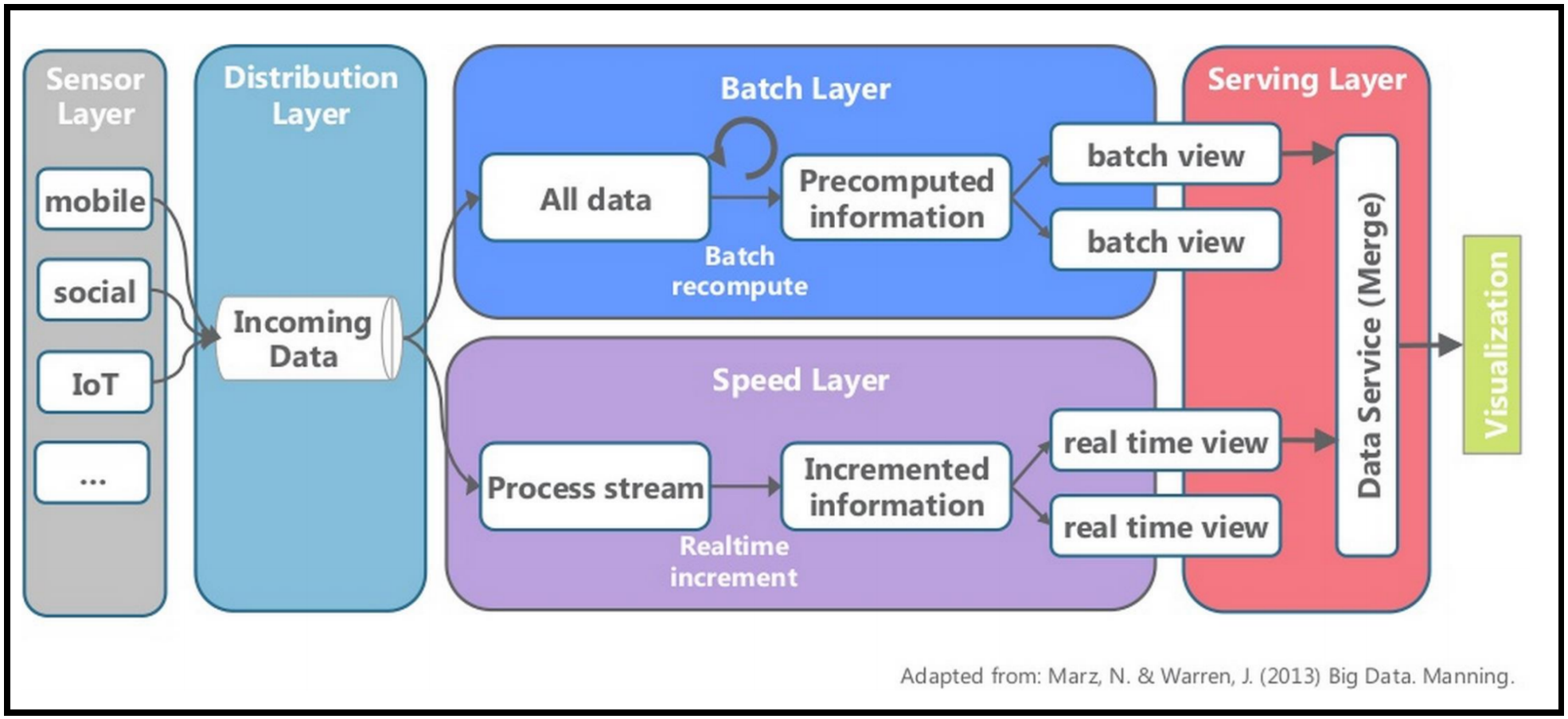
Feedback on scalling MySQL

- Scale UP limits ⇒ Olap size
- Scale OUT ⇒ replication and sharding
- Replication incidents
  - DRDB split brain
  - broken replication: network incidents
- **MySQL replication vs consistency**:
  - slowly diverging Master/Slave
    - not automatic check/resynchronisation → costly automatic
    - problematic when failover switch
- **Galera Cluster**: 1 year of transparent, incident free production

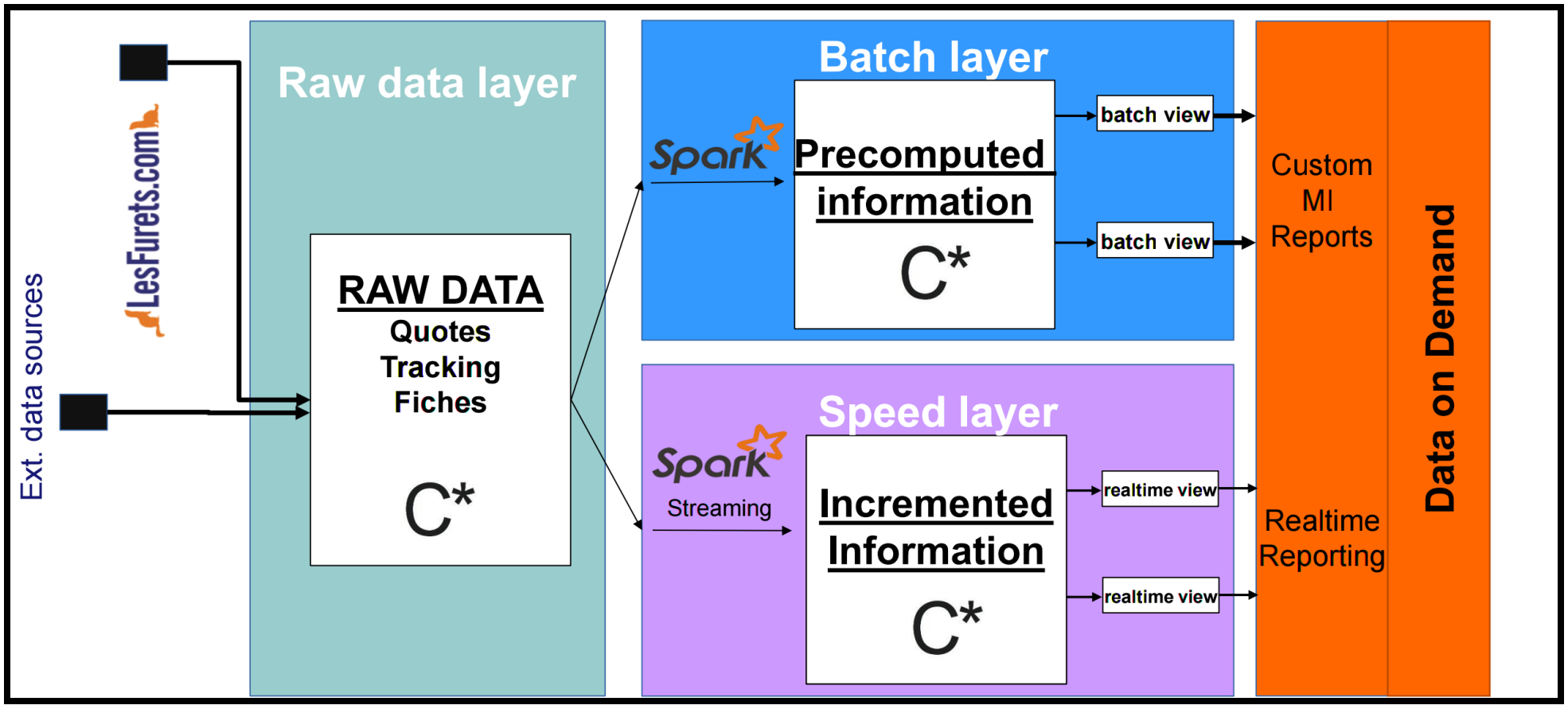
Lambda architecture (1)

- Combine best from No-SQL world to achieve
  - scalable systems
  - for arbitrary data problems
  - with human fault tolerance
  - and minimum complexity

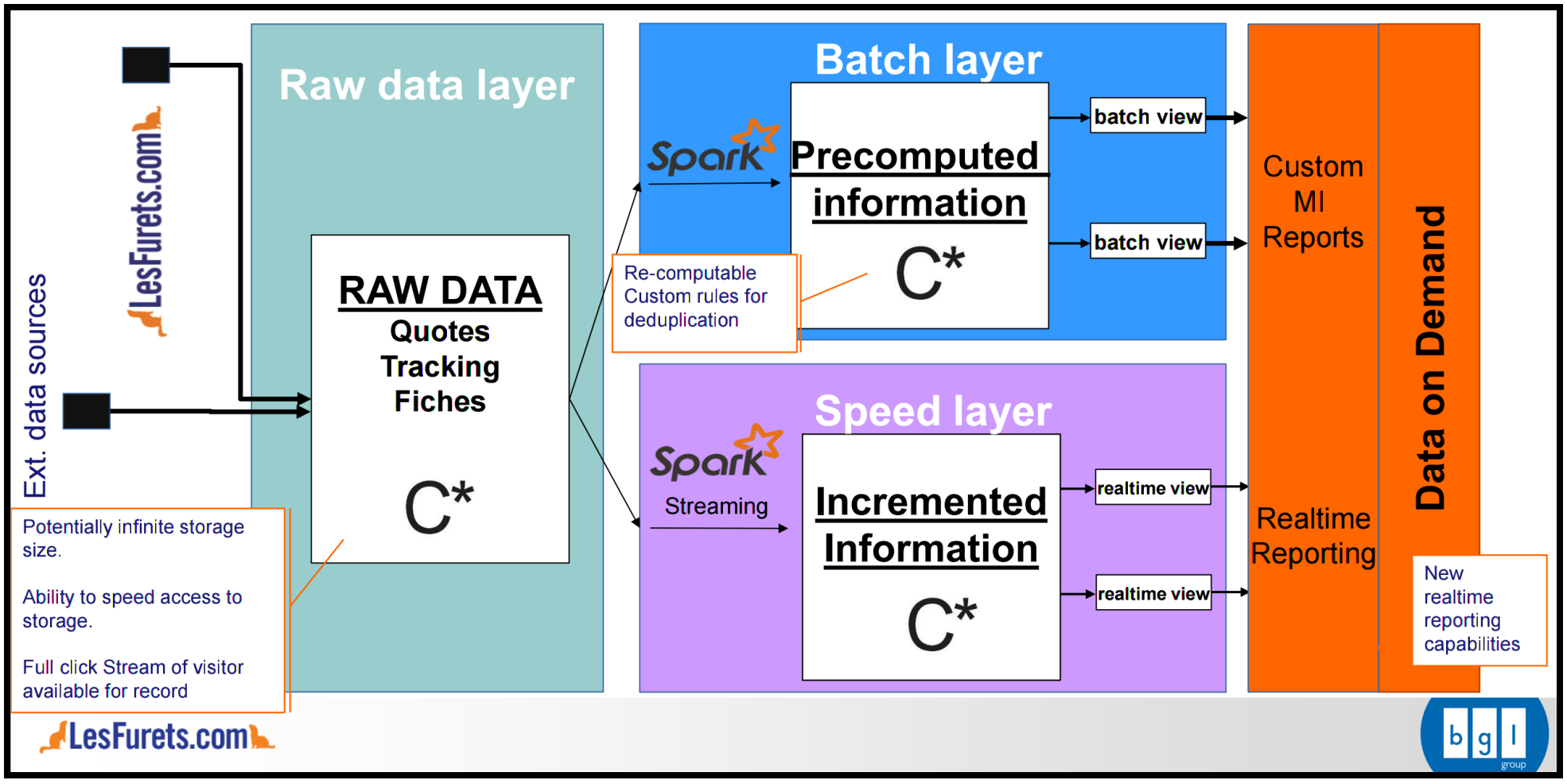
Lambda architecture (2)



On-going migration



On-going migration benefits



Plan

1. Intro
2. From SQL to NoSQL
3. Scalling MySQL : @LesFurets.com
4. **PostgreSQL + TP**

PostgreSQL

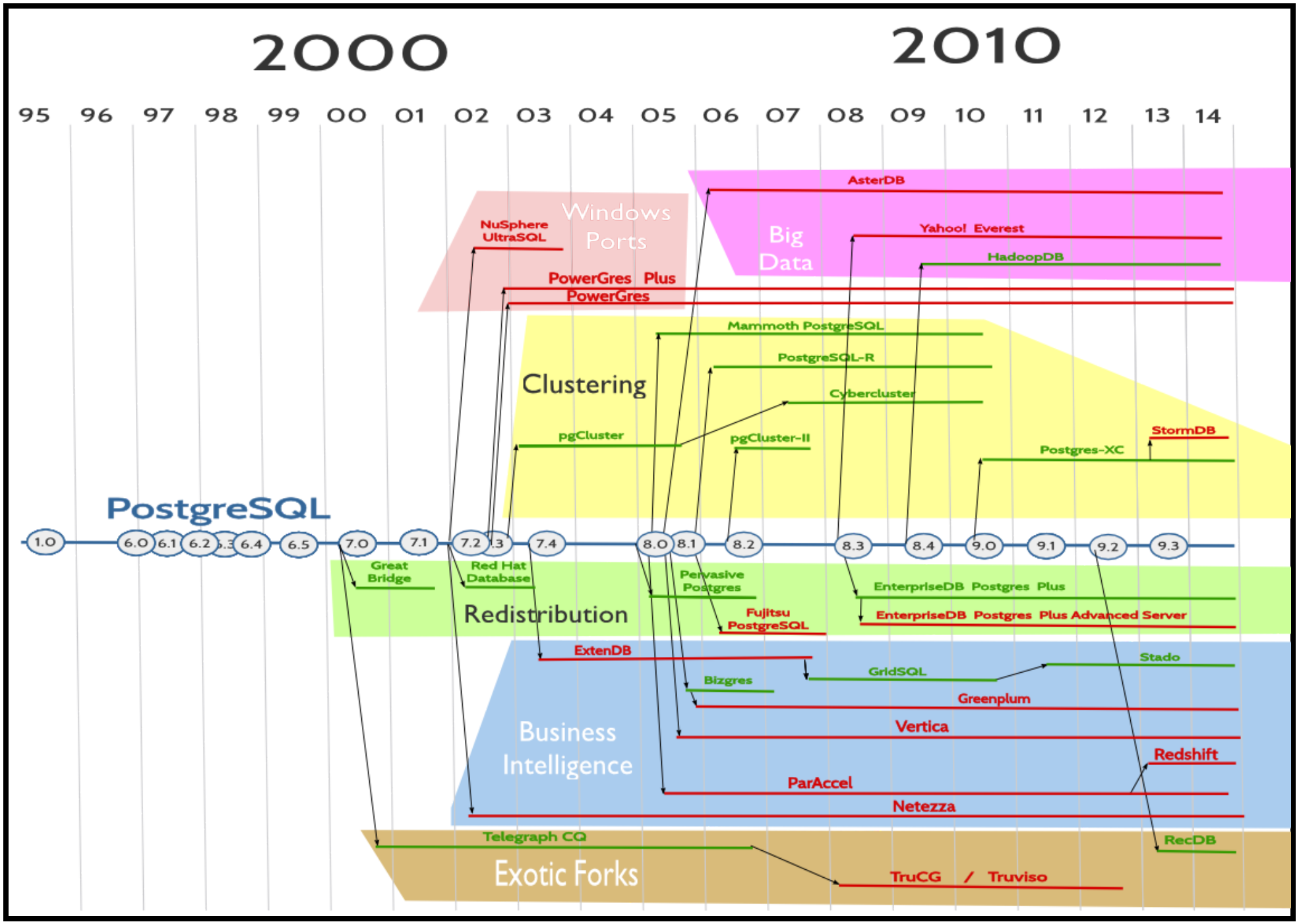
Postgres intro,
http://momjian.us/main/presentations/overview.html

PostgreSQL

- PostgreSQL
  - Berkeley : Ingres 1974
- Extensions
  - custom data types / operators
  - natural-language parsing
  - multidimensional indexing
  - geographic queries
- TP ⇒ PostgreSQL & extensions (*tablefunc*,*dict_xsyn*, *fuzzystrmatch*, *pg_trgm*, *cube*)

more…

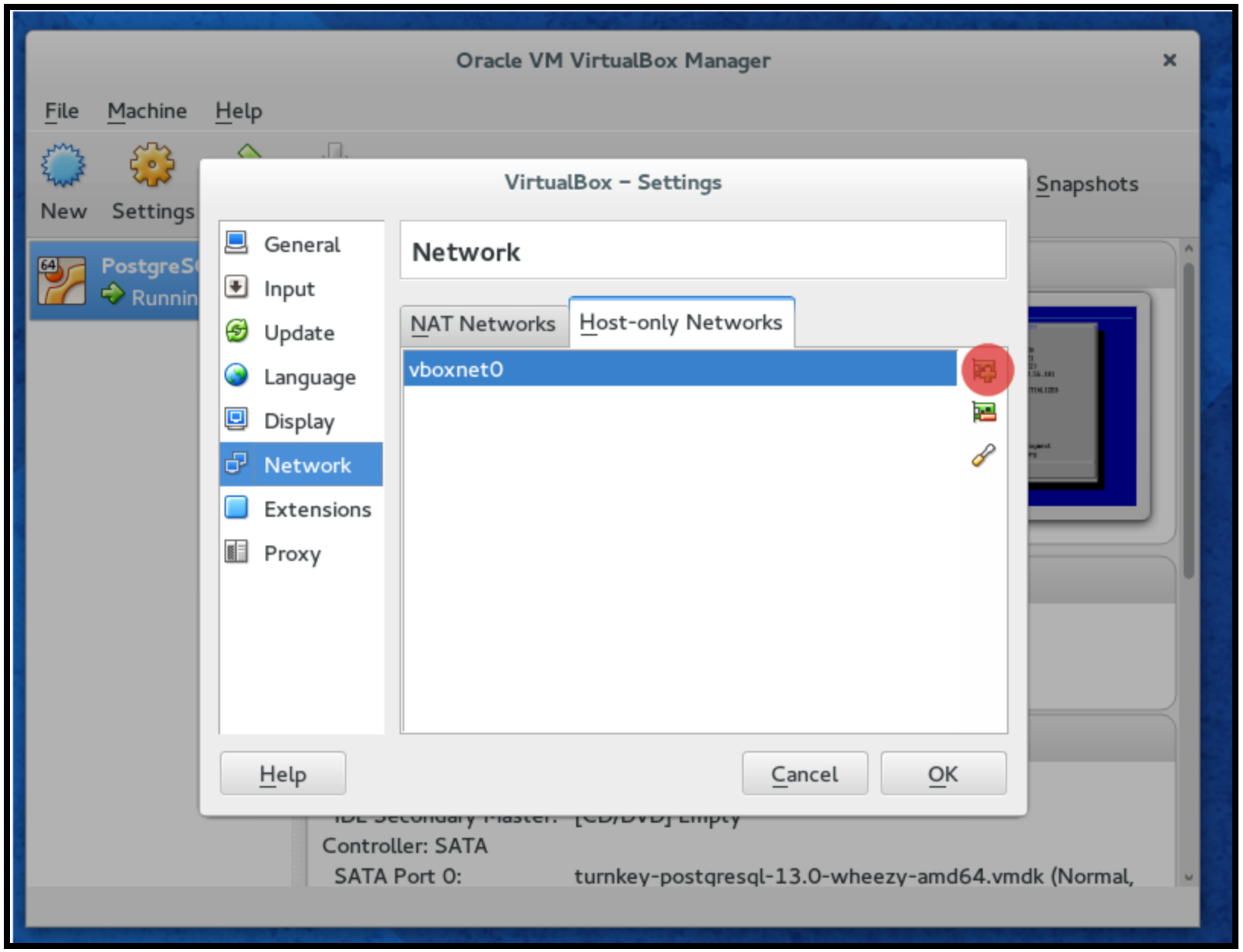
PostgreSQL offsprings



TP1: Installation de l'environnement

1. Démarrez l'application **Virtualbox**
2. Configurez un réseau host-only ***vboxnet0*** (File/Preferences/Network puis voir image) ou via les commandes suivantes:

```
VBoxManage hostonlyif create
VBoxManage hostonlyif ipconfig vboxnet0 --ip 192.168.56.1
VBoxManage dhcpserver add --ifname vboxnet0 --ip 192.168.56.1\
           -netmask 255.255.255.0 --lowerip 192.168.56.100\
           --upperip 192.168.56.200
VBoxManage dhcpserver modify --ifname vboxnet0 --enable
```

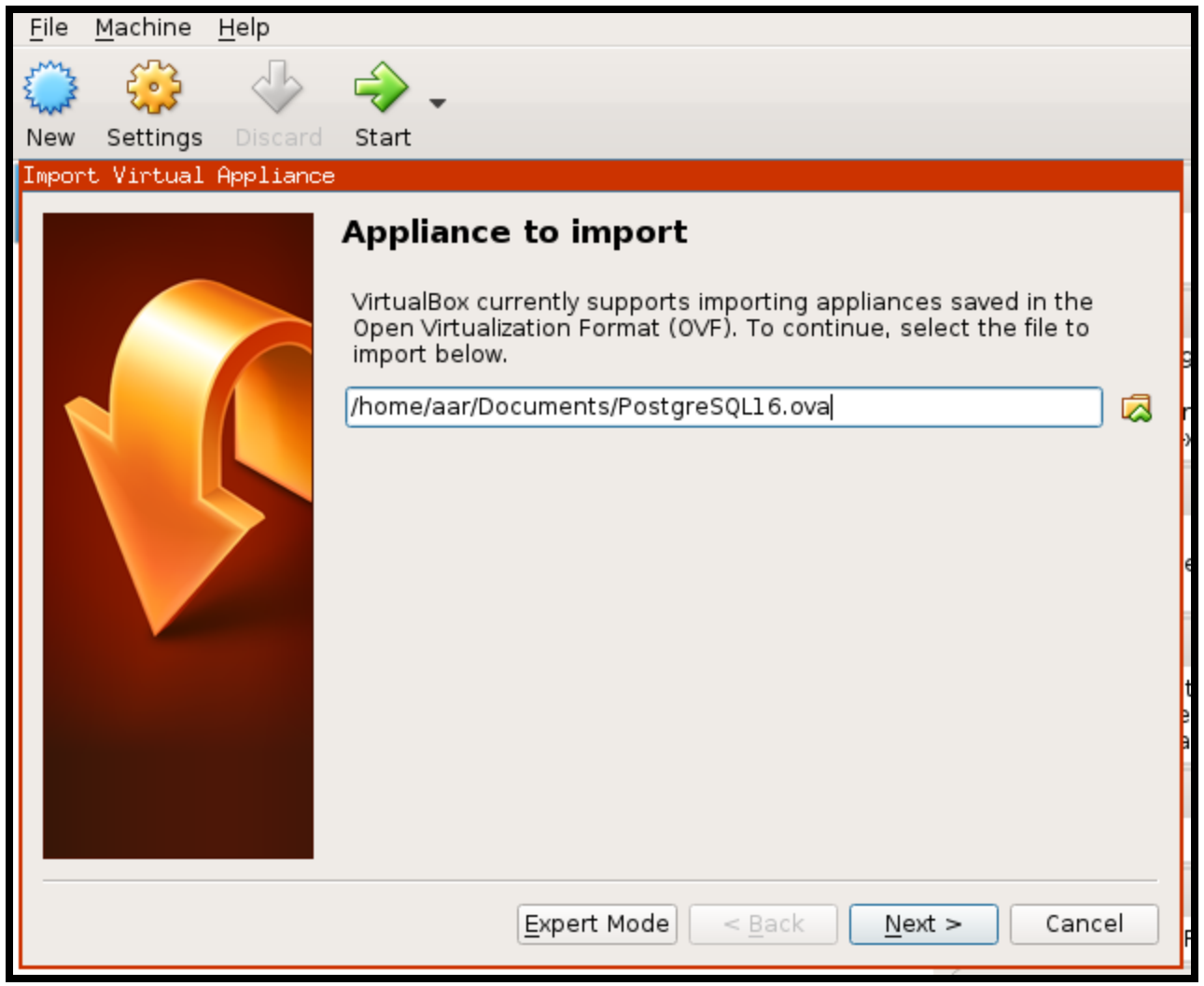

Oracle VM VirtualBox Manager
File
Machine
Help
New
Settings
Snapshots
VirtualBox – Settings
General
Input
Update
Language
Display
Network
Extensions
Proxy
Network
NAT Networks
Host-only Networks
vboxnet0
Help
Cancel
OK
Controller: SATA
SATA Port 0:
turnkey-postgresql-13.0-wheezy-amd64.vmdk (Normal,

TP1: Telecharger la VM pour le tp

1 Telecharger l'image Virtualbox: http://bit.ly/telecom-postgresql

TP1: Importer la VM

2 File/Import appliance …



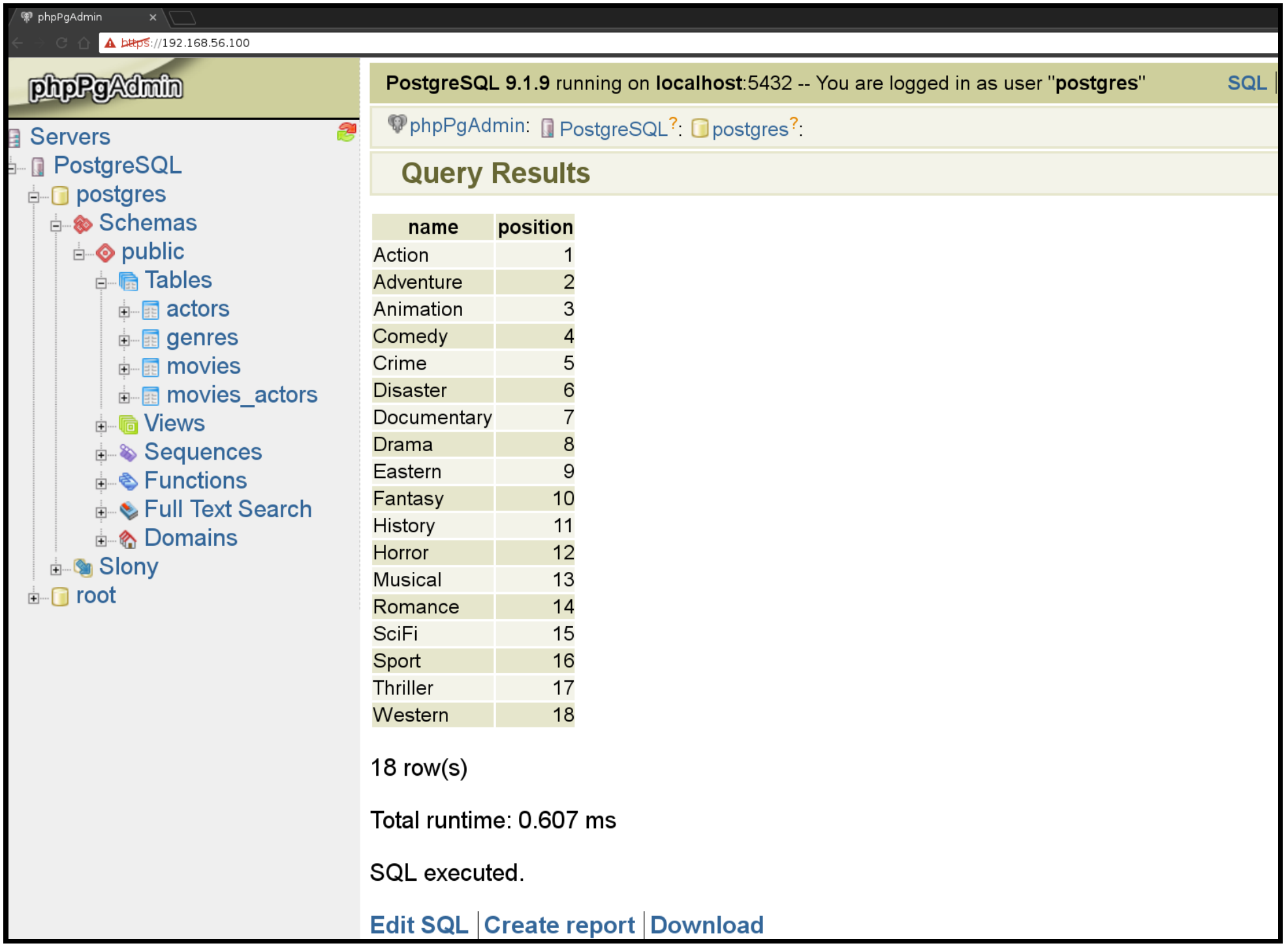
TP1: Start login to the VM

3 Démarrez la VM

4 Accédez a l'interface web https://192.168.X.Y/ (voir la console de la VM)

```
Login
User: postgres
Pass: bigdata14
```

TP1: phpPgAdmin



TP1: Recherche et recommandation

Moteur de recherche et recommandation des filmes:

- recherche: textuelle, approximative, phonétique
- moteur de recommandation très basique

TP1: Schéma (déjà créé)

```
CREATE TABLE genres (
        name text UNIQUE,
        position integer
);
CREATE TABLE movies (
        movie_id SERIAL PRIMARY KEY,
        title text,
        genre cube
);
CREATE TABLE actors (
        actor_id SERIAL PRIMARY KEY,
        name text
);

CREATE TABLE movies_actors (
        movie_id integer REFERENCES movies NOT NULL,
        actor_id integer REFERENCES actors NOT NULL,
        UNIQUE (movie_id, actor_id)
```

```
CREATE INDEX movies_actors_movie_id ON movies_actors (movie_id);
CREATE INDEX movies_actors_actor_id ON movies_actors (actor_id);
CREATE INDEX movies_genres_cube ON movies USING gist (genre);
```

Create schema script Import data script

TP1: Recherche

- ***Recherche exacte / pattern matching***
- ***Distance de Levenshtein*** → typos simples
- ***N-gram/similarité*** → trouver les erreurs modérées
- ***Full text match @@*** → similarité grammaticale
- ***Métaphone*** → similarité phonétique

TP1: Recherche textuelle/patterns

Utilisez les opérateurs LIKE ou RegEX pour les requêtes suivantes:

1. Tous les filmes qui ont le mot ***stardust*** dans leur nom.
2. Compter tous les filmes dont le titre ne commence pas par le mot ***the***
3. Tous les filmes qui ont le mot ***war*** dans le titre mais pas en dernière position

TP1: Distance Levenshtein

```
I N T E * N T I O N
| | | | | | | | | |
* E X E C U T I O N
d s s   i s
```

- Opérations: **S**ubstitute, **I**nsert, **D**elete
- Distance *Levenshtein* : nb minimal d'opérations

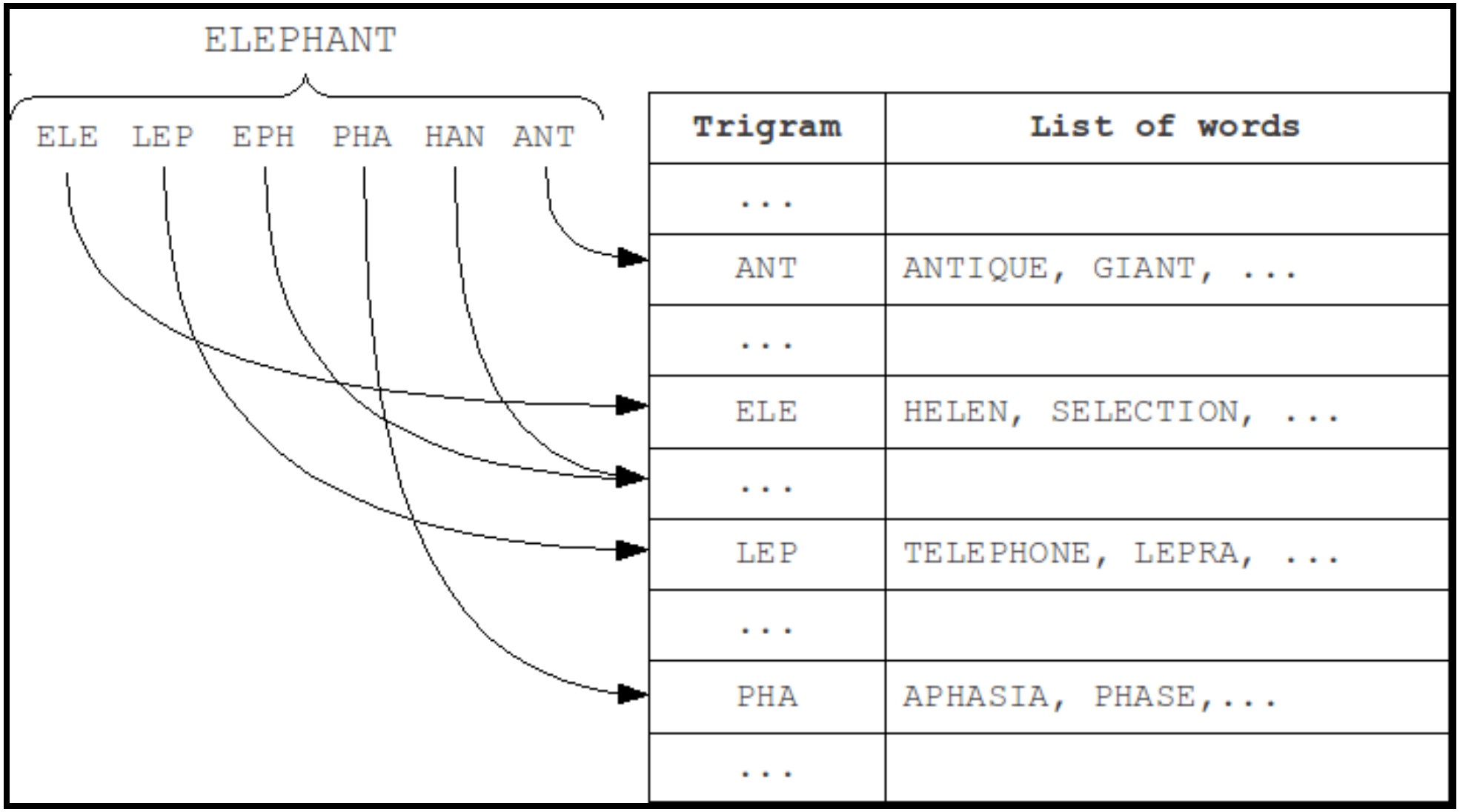
TP1: Distance Levenshtein

Utilisez les fonctions du package fuzzystrgmatch pour trouver :

1. La distance levenshtein entre les mots ***execution*** et ***intention***
2. Tous les films qui sont a une distance *levenshtein* inférieure a 9 de la chaine suivante: ***a hard day nght***

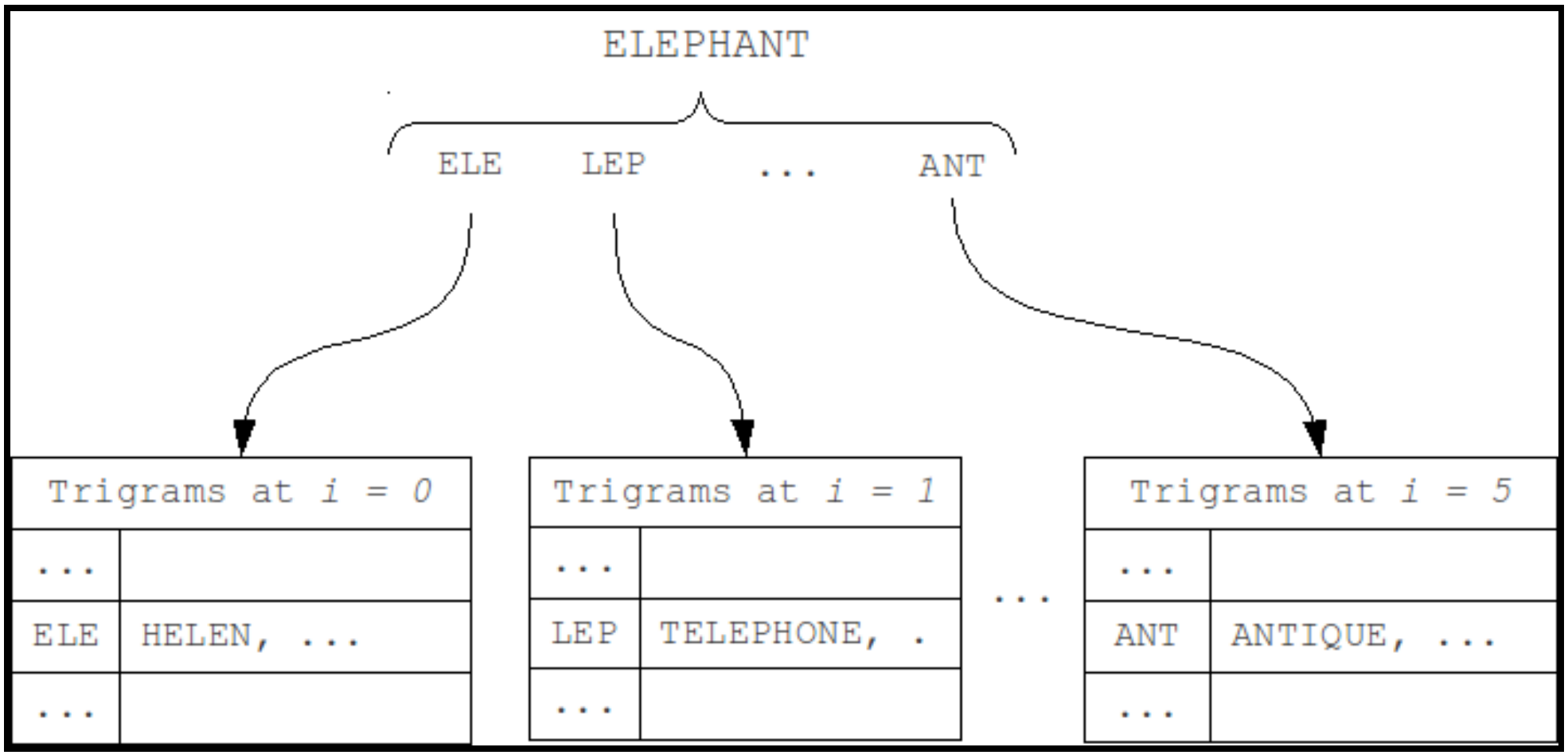
Documentation extension fuzzystrgmatch

TP1: N-gram



Trigram	List of words
...	
ANT	ANTIQUE, GIANT, ...
...	
ELE	HELEN, SELECTION, ...
...	
LEP	TELEPHONE, LEPRA, ...
...	
PHA	APHASIA, PHASE,...
...	

TP1: N-gram, similarity search



TP1: N-gram, similarity search (%)

Écrivez les requêtes pour trouver :

1. Tous les tri-grammes du mot ***Avatar***
2. La similarité entre ***VOTKA*** et ***VODKA***
3. Tous les filmes dont le titre est similaire a plus de 0.1% du titre ***Avatar*** .

Documentation extension trgm

TP1: Full text search

Trouver les filmes qui contiennent les formes grammaticales des mots 'night' et 'day':

(ignorer les mots de liaison/ pluriel/etc..)

Algorithme:

1. ***extraire les racine des mots (lexèmes) → spécifiques au langage !***
2. comparer les vecteurs des lexèmes

TP1: Full text search

```
SELECT to_tsvector('A Hard Day''s Night'),
     to_tsquery('english', 'night & day');

       to_tsvector          |   to_tsquery
----------------------------+-----------------
'day':3 'hard':2 'night':5 | 'night' & 'day'
```

- ***tsvector*** : lexèmes :position
- ***tsquery*** : lexèmes séparées par &
- ***specifique au language !***

Full text search

```sql
SELECT title
FROM movies
WHERE to_tsvector(title) @@ to_tsquery('english', 'night & day');
```

```sql
SELECT title
FROM movies
WHERE title @@ 'night & day';

A Hard Day's Night
Six Days Seven Nights
Long Day's Journey Into Night
```

TP1: Recherche phonétique

- plusieurs fonctions pour la codification phonétique des mots

```
SELECT name, dmetaphone(name), dmetaphone_alt(name),
  metaphone(name, 8), soundex(name)
FROM actors;

     name       | dmetaphone | dmetaphone_alt | metaphone | soundex
----------------+------------+----------------+-----------+--------
50 Cent         | SNT        | SNT            | SNT       | C530
Aaron Eckhart   | ARNK       | ARNK           | ARNKHRT   | A652
Agatha Hurle    | AK0R       | AKTR           | AK0HRL    | A236
```

Documentation fuzzystrmatch…

TP1: Recherche phonétique

1. Trouver les filmes qui ont des acteurs dont les noms se prononcent pareil.
2. Trouver les acteurs avec un nom similaire a ***Robin Wiliams***, triés par similarité (combiner %, metaphone et levenshtein):

```
 actor_id | name
----------+-----------------
     4093 | Robin Williams
     2442 | John Williams
     4479 | Steven Williams
     4090 | Robin Shou
```

TP1: Search

- ***Recherche exacte / pattern matching***
- ***Distance de Levenstein*** → typos simples
- ***N-gram/similarite*** → trouver les erreurs modérées
- ***Full text match @@*** → similarité grammaticale
- ***Métaphone*** → similarité phonétique

TP1: Recherche multi-dimensionnelle

```
CREATE TABLE movies (
        movie_id SERIAL PRIMARY KEY,
        title text,
        genre cube (1)
);

INSERT INTO movies (movie_id,title,genre) VALUES
(1,'Star Wars',
'(0,7,0,0,0,0,0,0,0,7,0,0,0,0,10,0,0,0)') (2)
),
```

- on utilise le type cube <1> pour mapper les notes sur un vecteur n-dimensionnel de valeurs (= score du film <2>)

TP1: Recherche multi-dimensionnelle

- les noms pour les dimensions sont définis dans la table genres

```
CREATE TABLE genres (
        name text UNIQUE,
        position integer
);

INSERT INTO genres (name,position) VALUES
('Action',1),
('Adventure',2),
('Animation',3),
...
('Sport',16),
('Thriller',17),
('Western',18);
```

TP1: Recherche multi-dimensionnelle

Utiliser le module cube pour recommander des filmes ***similaires*** (du même genre)

- Afficher les notes du film ***Star Wars***
- Quelle est la note du film ***Star Wars*** dans la catégorie 'Animation'
- Afficher les films avec les meilleurs notes dans la catégorie SciFi

TP1: Recherche multi-dimensionnelle

- Afficher les filmes similaire (**cube_distance**) a ***Star Wars*** (vecteur = (0, 7, 0, 0, 0, 0, 0, 0, 0, 7, 0, 0, 0, 0, 10, 0, 0, 0)) du plus similaire au moins similaire

```
                  title                   |       dist
------------------------------------------+------------------
Star Wars                                 |                0
Star Wars: Episode V - The Empire Strikes Back |                2
Avatar                                    |                5
Explorers                                 | 5.74456264653803
Krull                                     | 6.48074069840786
E.T. The Extra-Terrestrial                | 7.61577310586391
```

- Écrivez une requête pour trouver les filmes qui sont a moins de 5 points de différence sur chaque dimension (utiliser *cube_enlarge* et @>).

Ressources:

Bigdata - Nathan Marz book

NoSQL Distilled - Martin Fowler

7 databases

BigTable paper

Postgres cheatsheat

MovieLens dataset

TP1: Installation en détail

Fedora Ubuntu

```
yum install postgresql postgresql-server postgresql-contrib (1)
postgresql-setup initdb (2)

systemctl start postgresql.service (3)

yum install pgadmin3   (4)

CREATE EXTENSION tablefunc; (5)
CREATE EXTENSION dict_xsyn;
CREATE EXTENSION fuzzystrmatch;
CREATE EXTENSION pg_trgm;
CREATE EXTENSION cube;
```

1. Instalation du client/serveur/extensions suplementaires
2. Initialisation de la base
3. Demarage du serveur
4. Front-end requetage
5. Installation des extensions Verifier les extensions installees

Create index

```
CREATE INDEX  [ nom ] ON table [ USING method ]
    ( { colonne | ( expression ) } [ classeop ]  ... )
```

- ***method***: btree/hash/gin/gist
- ***classeop*** : operator class that can use the index

Documentation