Implementing the Lambda Architecture with Spring XD

By Carlos Queiroz

Me

Carlos Queiroz

Advanced Field Engineer

Pivotal

Agenda

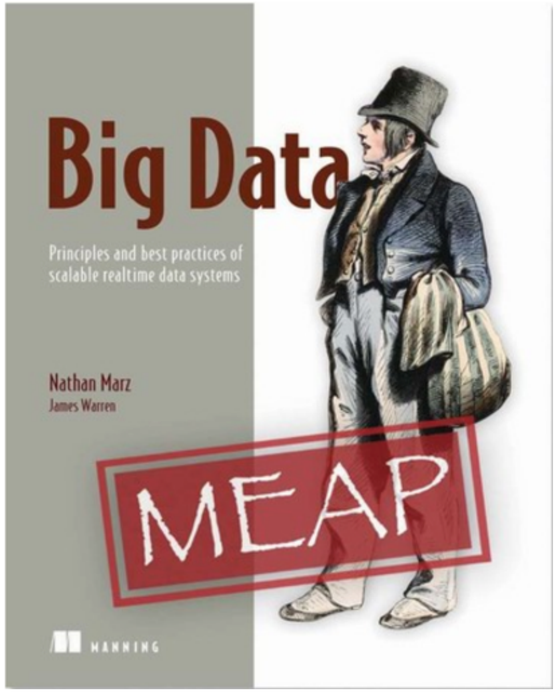
- Introduction to the Lambda Architecture
- Applying Lambda architecture to a business case
- Implementation details

What is Lambda Architecture?

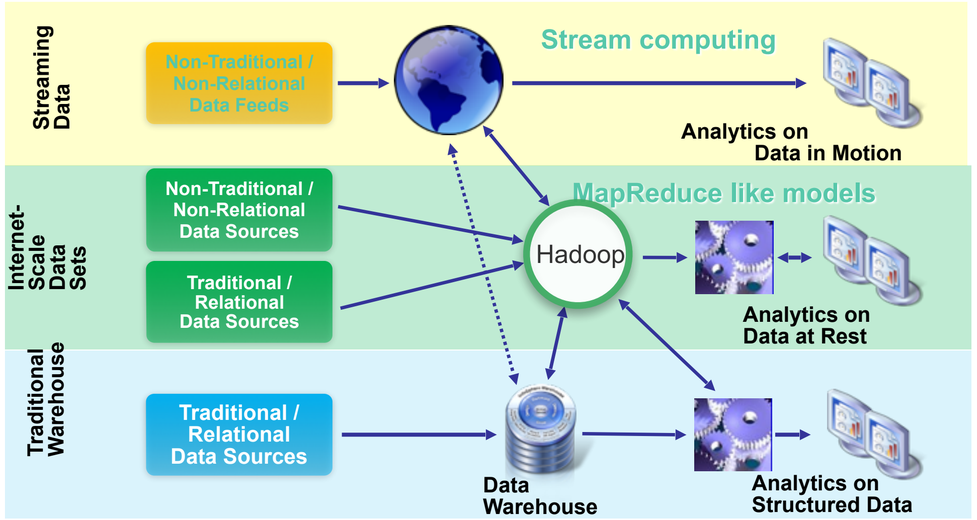
Implementation of a set of **desired properties** on general purpose big data systems.

Generic architecture addressing common requirements in *big data* applications.

Set of **design patterns** for dealing with historical and operational data.



Approach is new, ideas not so…



The Big Data Ecosystem

Why Lambda Architecture?

To build a data system that can answer questions by running functions that take at the **entire dataset** as input. **A general purpose data system**

Desired Properties of such systems

Fault-tolerance

Generic

Scale linearly, horizontally.

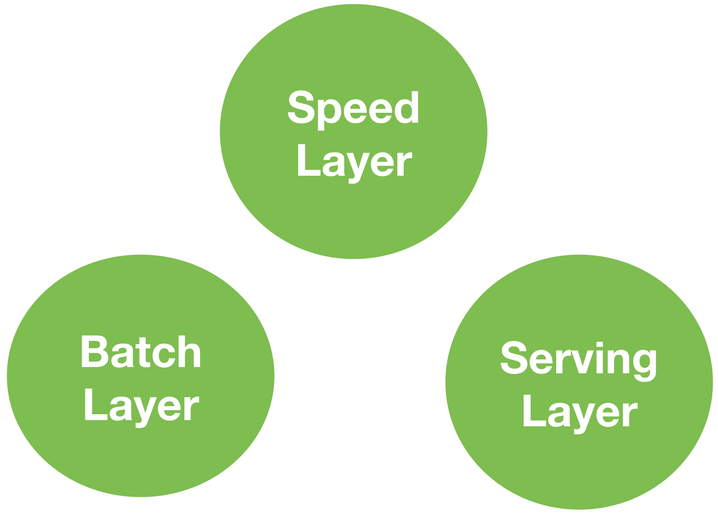
Extensible

Be able to achieve low latency updates when necessary.

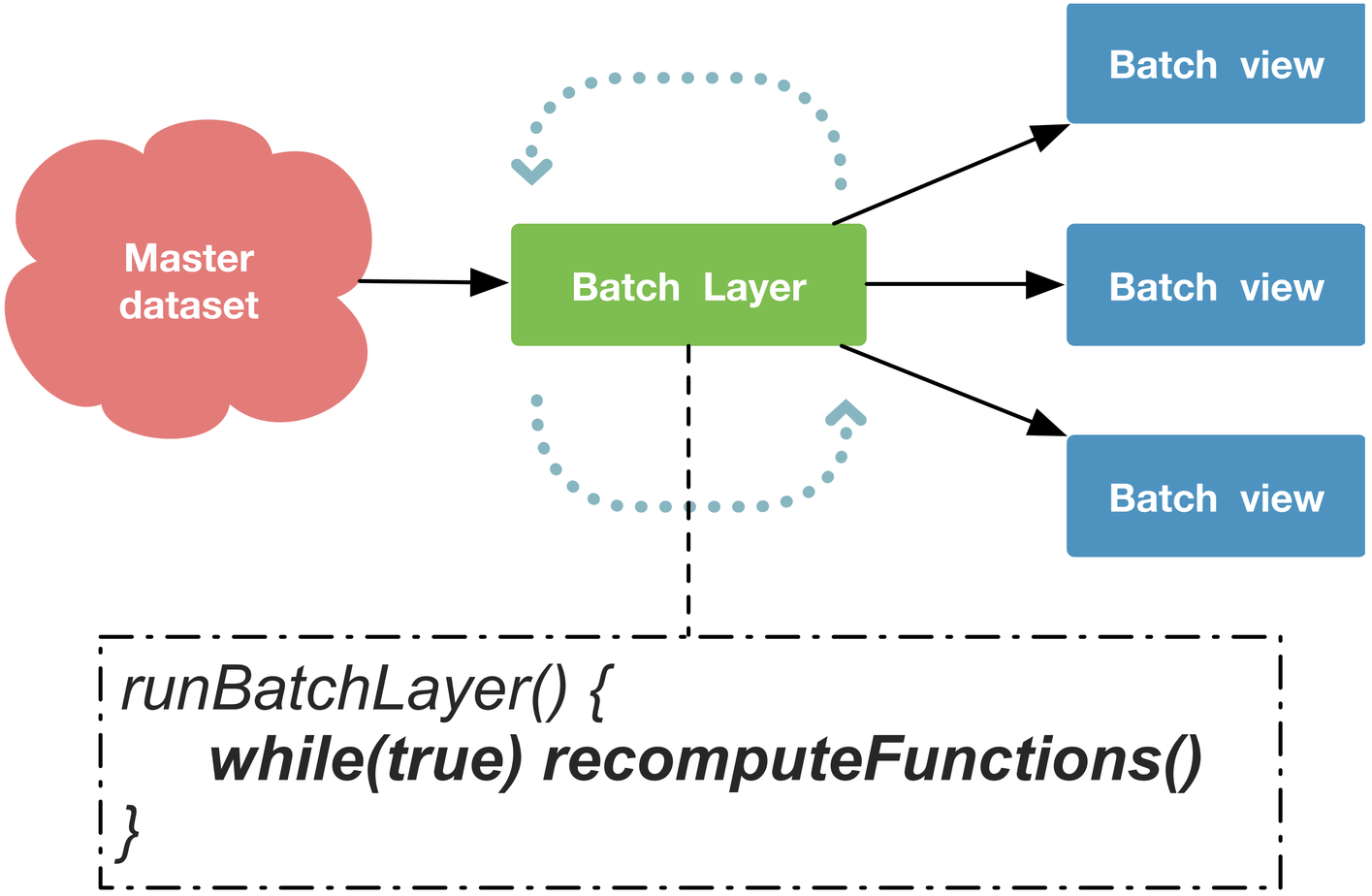
Be able to “ask” arbitrary questions to the system

How to support those properties?

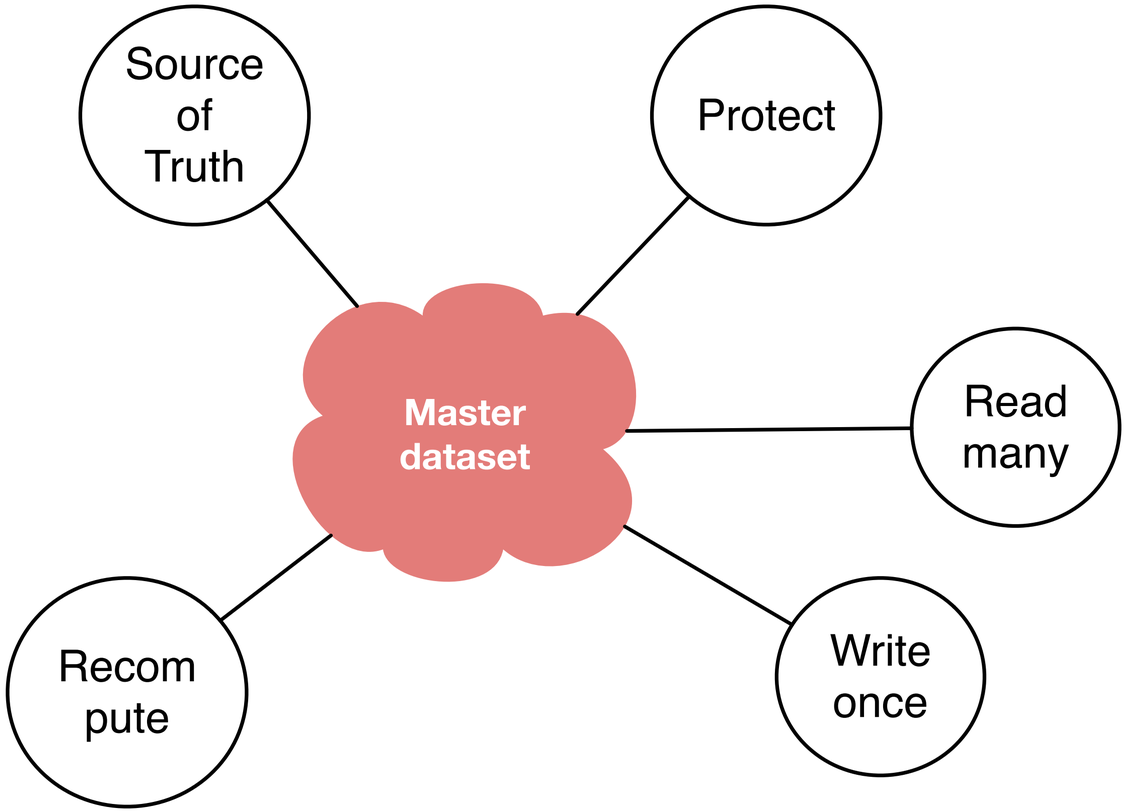
Decompose the problem into pieces



Batch Layer


Master dataset
Batch Layer
Batch view
Batch view
Batch view
runBatchLayer() {
while(true) recomputeFunctions()
}

The Master dataset



Master dataset requirements

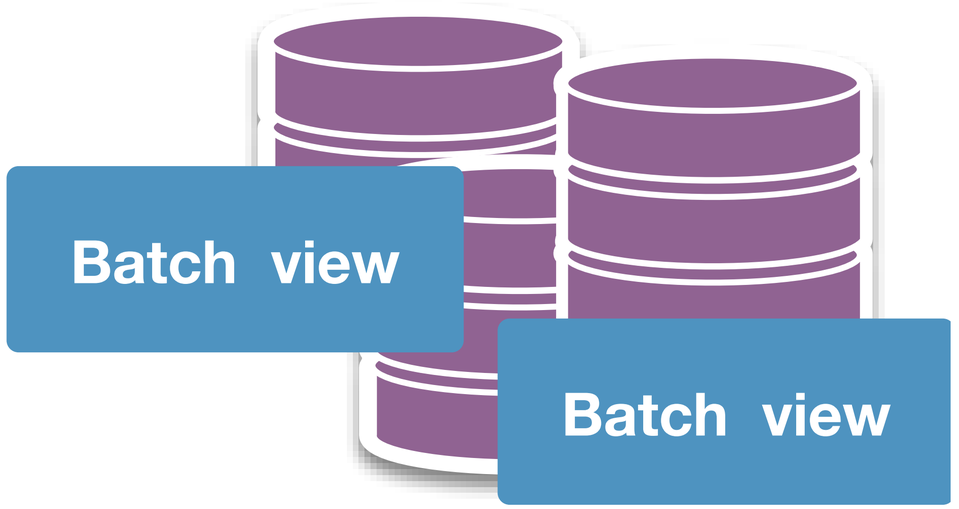
Operation	Requisite	Comments
Writes	***Efficient appends of new data***	***Basically add new pieces of data. Easy to append new data***
Writes	**Scalable storage**	Need to handle possibly, petabytes of data
Reads	***Support for parallel processing***	***Functions usually work on the entire dataset. Need to support handling large amounts of data***
Reads	**Vertically partition data**	Not necessary to look all the data all the time. Some functions may need to look at only relevant data (e.g. 1 week of calls)
Writes/Reads	***Costs for processing. Flexible storage***	***Storage costs money (a lot). Need flexibility on how to store and compress data.***

The Master dataset

HDFS - Hadoop Distributed File System

Serving Layer

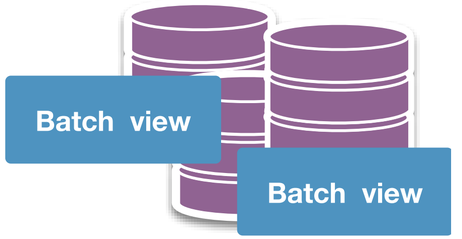
Distributed database

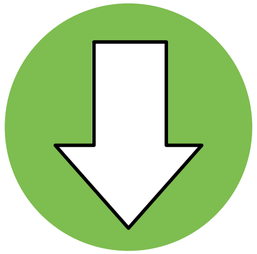


Batch updates and random reads

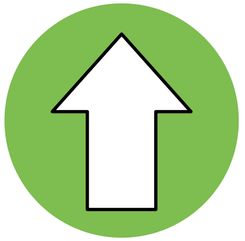
Makes the batch views “*queryable*”

Desired properties on the Serving layer





Low Latency



High Throughput

Serving layer database requirements

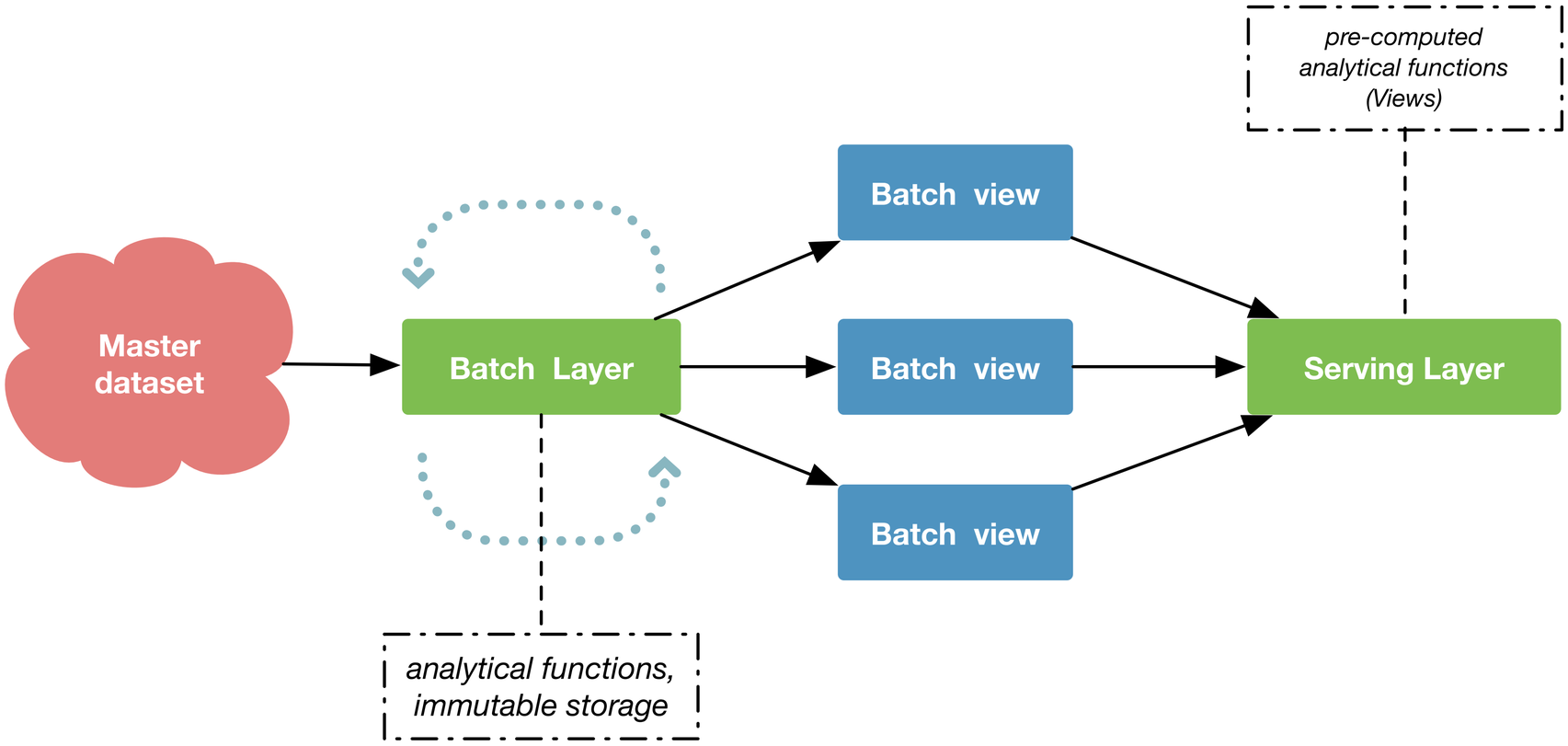
Fault-tolerant

Batch writable

Random reads

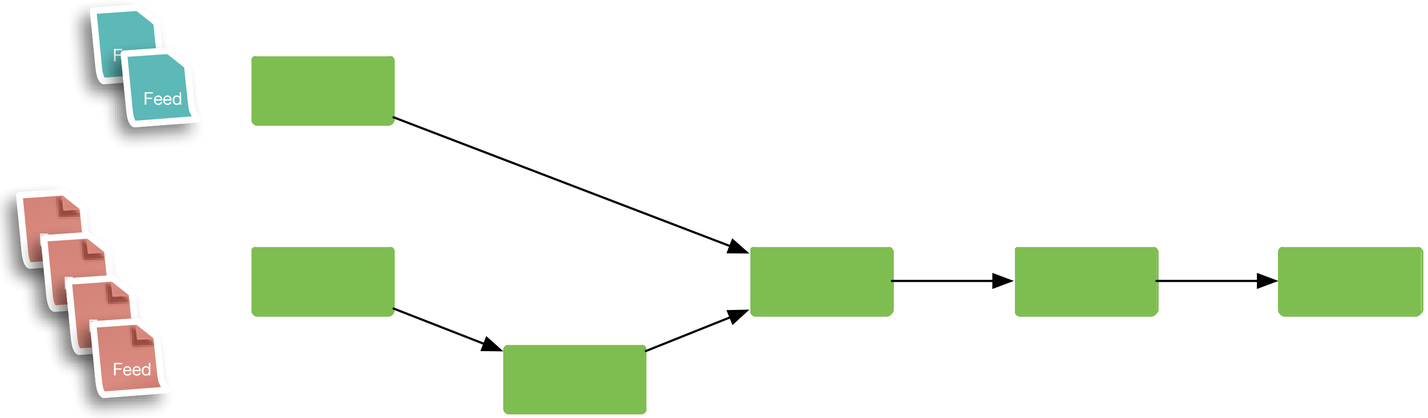
Scalable

Batch and serving layers



Only property missing - **low latency updates**

Speed layer



Allows arbitrary functions computed on arbitrary data on (near) real-time.

Speed layer

Narrow but more up to date view

Depending of functions complexity incremental computation approach is recommended

Incremental computation

realtime view = function(new data, realtime view)

Eventual Accuracy

Some computations are harder to compute

For such cases approximations are used. Results are approximate to the correct answer

Sophisticated queries such as realtime machine learning are usually done with eventual accuracy. Too complex to be computed exactly.

Approximation & Randomisation

Approximation

find an answer correct within some factor

(answer that is within 10% of correct result)

Randomisation

allow a small probability of failure

(1 in 10,000 give the wrong answer)

Synopses structures

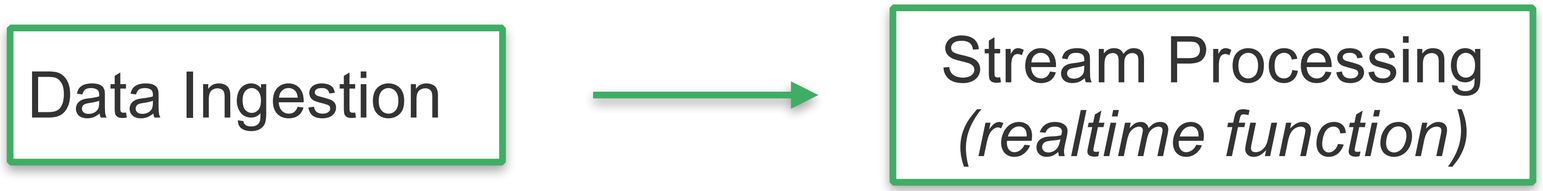
- Sampling
- Sketches
- Histograms
- Wavelets

http://charuaggarwal.net/synopsis.pdf

Library implementations

- algebird (https://github.com/twitter/algebird)
- samoa (https://github.com/yahoo/samoa/wiki)

Stream processing (real-time function)

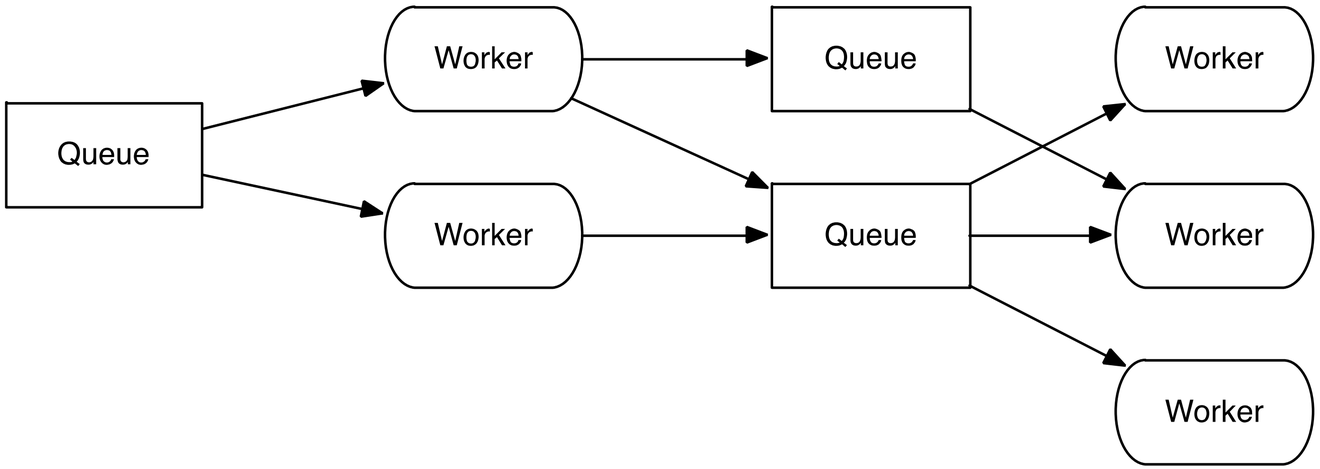


Run the realtime functions to update the realtime views

	one-at-atime	micro-batched
lower latency	✔	✗
High throughput	✗	✔
At-least-once semantics	✔	✔
exactly-once semantics	some cases	✔
Simpler programming model	✔	✗

One-at-a-time

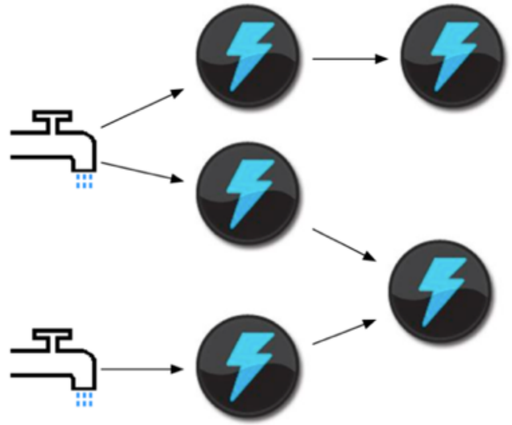
Divide your processing into worker processes, and put queues between the worker processes



Queues and workers paradigm

Generalised one-at-a-time approach

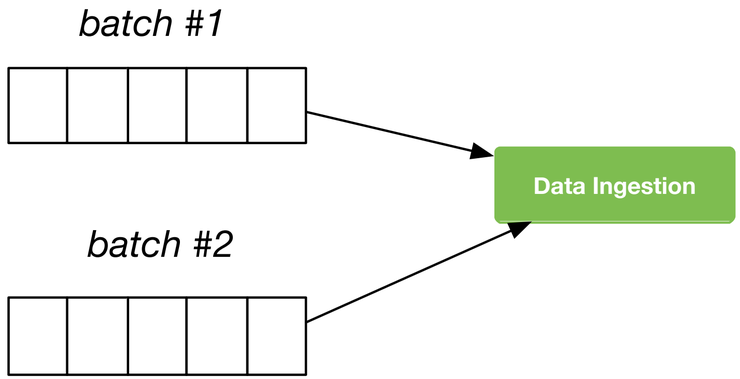
- Works on a higher level
- Stream computation defined as a graph (usually).
  - Storm, InfoSphere Streams models
- Filters and pipes
  - Spring XD
- *At least once* in case of failures.



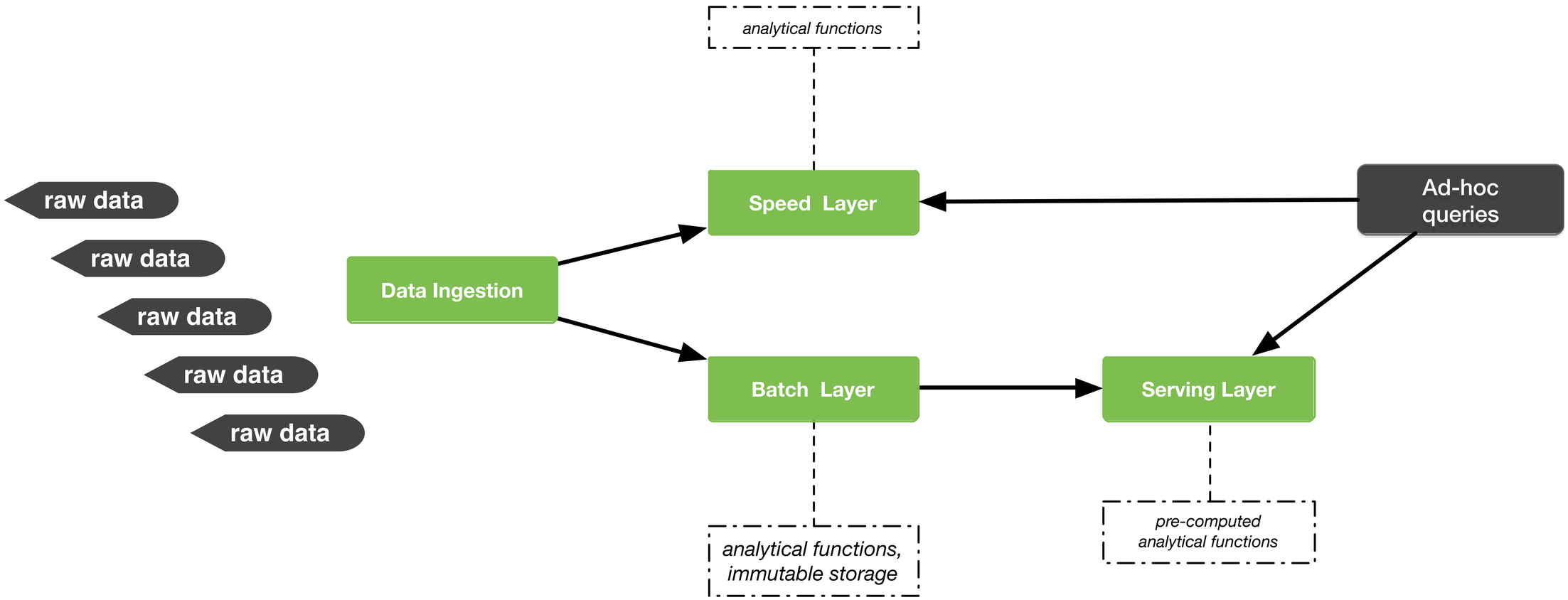
cut -d" " -f1 < access.log | sort | uniq -c | sort -rn | less

Micro-batching

- Small batches of events processed one at a time
- Exactly one processing
- Implementations:
  - Spark Streaming



Lambda Architecture



Principles of Lambda Architecture

Store data in it's rawest form

Immutability and perpetuity

Re-computation

Query = function(all data)

Implementing the Lambda Architecture

The ACM DEBS 2014 Grand Challenge[1]

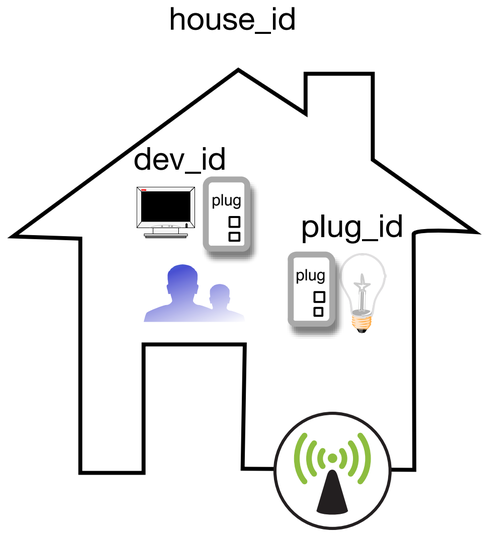
To demonstrate the applicability of event-based systems to provide scalable, real-time analytics over high volume sensor data

[1] http://www.cse.iitb.ac.in/debs2014/

ACM DEBS 2014 Challenge Application

Scenario

Analysis of energy consumption measurement



Short-term load forecasting

makes load forecasts based on current load and what was learned over historical data

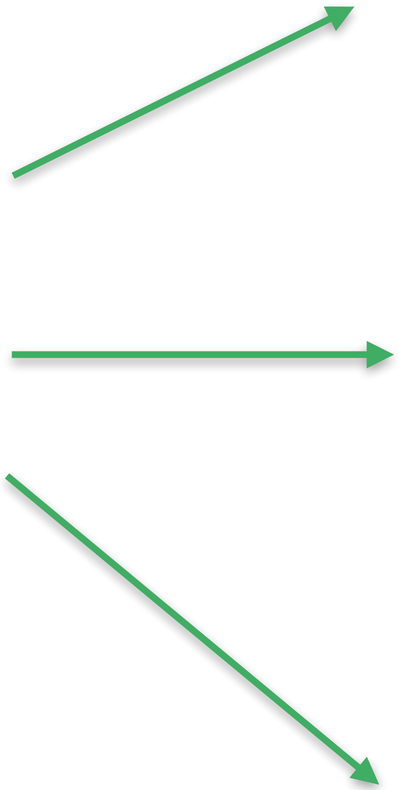
Load statistics for real-time demand management

finds outliers based on the energy consumption

Data model

Data source

Field	Comments
ID	UNIQUE IDENTIFIER
TIMESTAMP	TIMESTAMP OF MEASUREMENT
VALUE	MEASUREMENT
PROPERTY	TYPE: 0 - WORK, 1 - LOAD
PLUG_ID	UNIQUE IDENTIFIER OF A PLUG
HOUSEHOLD_ID	UNIQUE IDENTIFIER OF A HOUSEHOLD
HOUSE_ID	UNIQUE IDENTIFIER OF A HOUSE



PL - House

Field	Comments
TS	TIMESTAMP OF STARTING TIME PREDICTION
HOUSE_ID	HOUSE ID
PREDICTED_LOAD	PREDICTED LOAD

PL - Plug

Field	Comments
TS	TIMESTAMP OF STARTING TIME PREDICTION
HOUSE_ID	HOUSE ID
HOUSEHOLD_ID	
PLUG_ID	
PREDICTED_LOAD	PREDICTED LOAD

Outliers

Field	Comments
TS_START	TIMESTAMP OF START TIME WINDOW
TS_STOP	TIMESTAMP OF STO PTIME WINDOW
HOUSE_ID	HOUSE ID
PERCENTAGE	% PLUGS LOAD HIGHER THAN NORMAL

Analytical models

Load prediction per house, per plug

It is based on the average load of the current
window and the median of the average loads of windows covering the same time of all past days

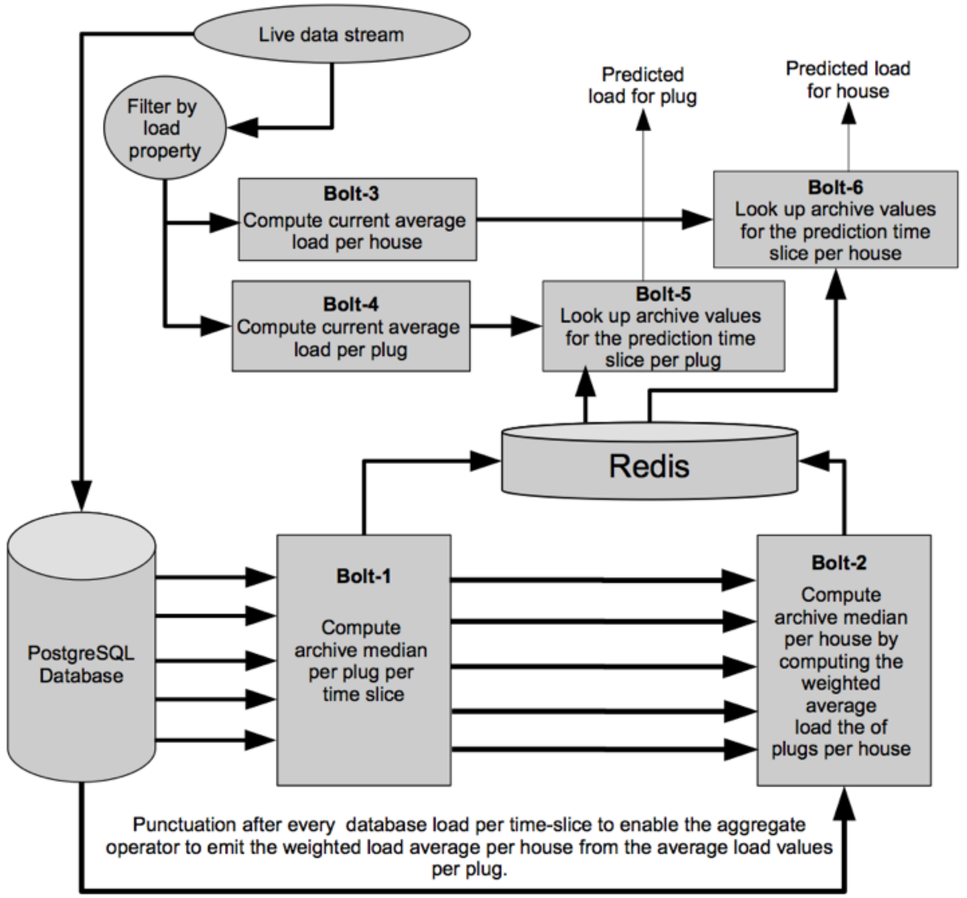
$$L(s_{i+2}) = (avgLoad(s_i) + median(avgLoad(s_j)))/2$$

$$s_j = s_{(i+2-n*k)}$$

Outliers

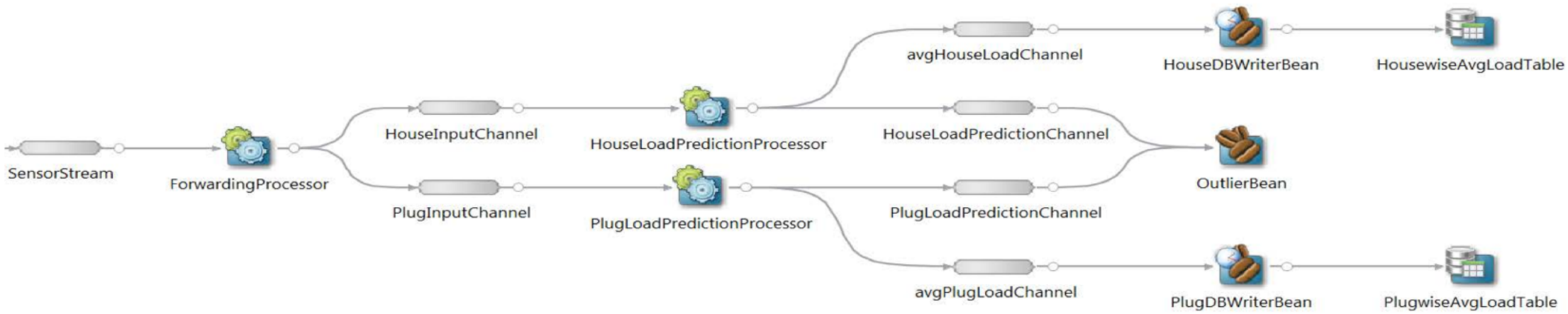
For each house calculate the percentage of plugs which have a median load during the last hour greater than the median load of all plugs (in all households of all houses) during the last hour

How others did



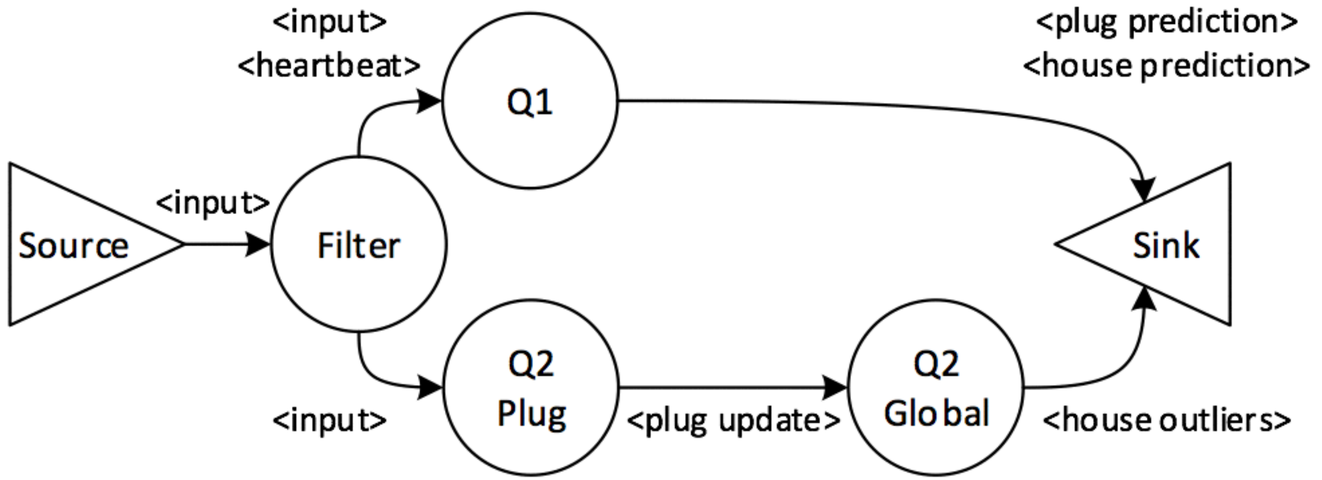
Storm-based implementation

How others did



Oracle DB + Oracle Event Processing (OEP)

How others did

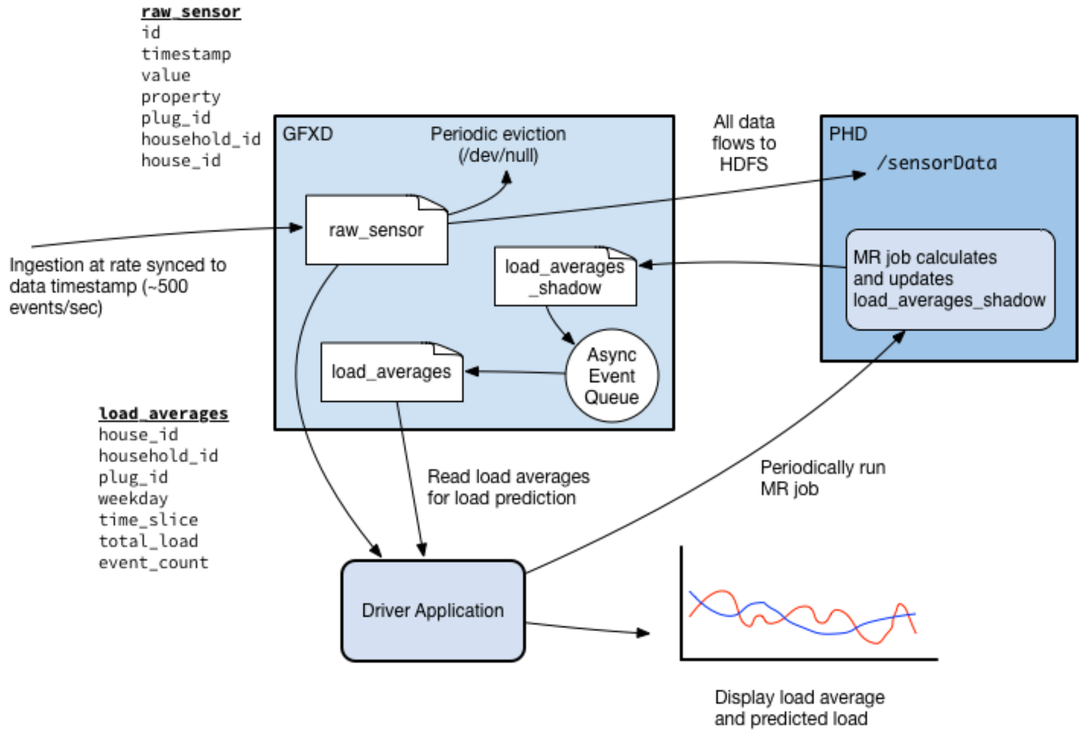


SEEP: Real-Time Stateful Big Data Processing

http://lsds.doc.ic.ac.uk/projects/seep/debs14-gc

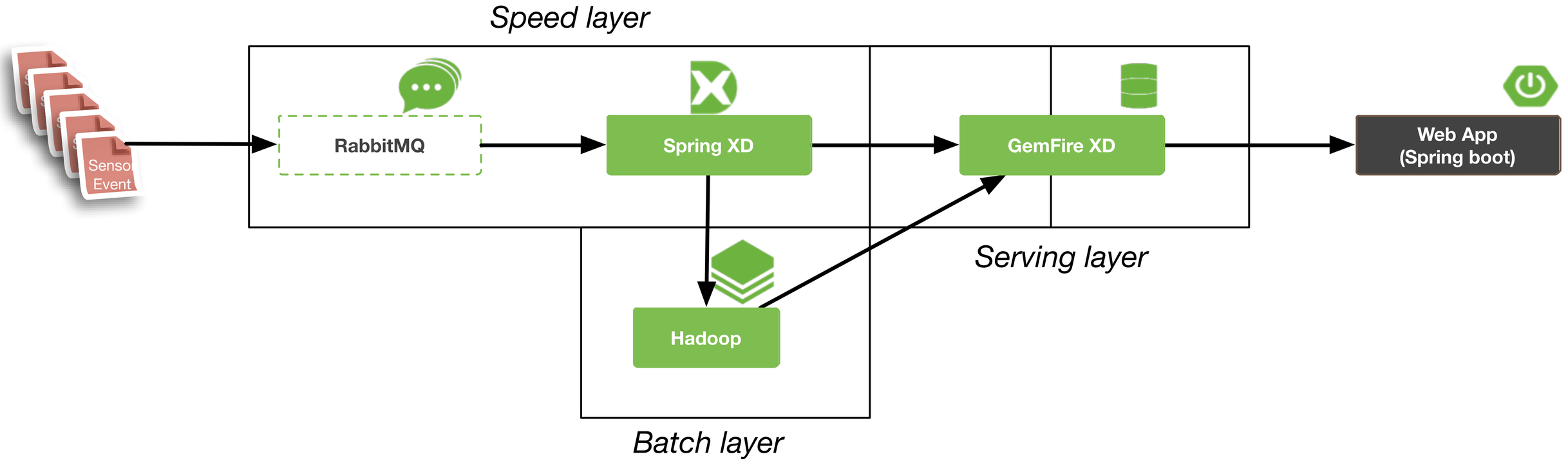
Implementation details

Fork from https://github.com/pivotalsoftware/gfxd-demo

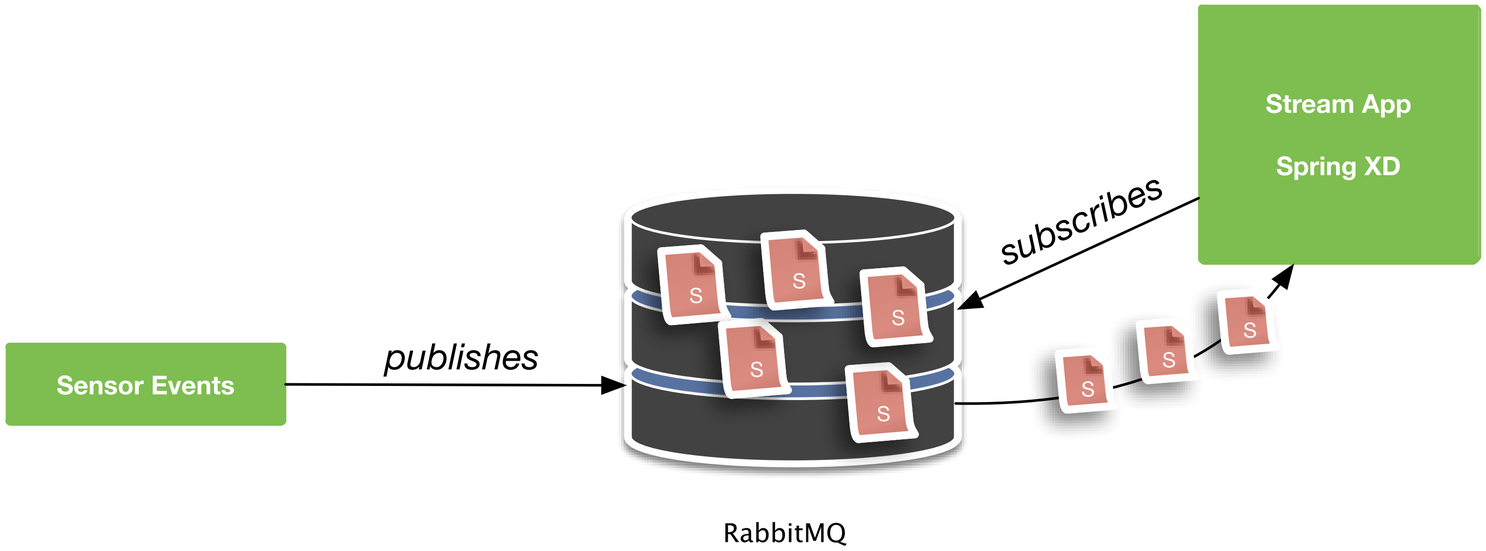


Thanks to Jens Deppe and William Markito

Our approach - ACM DEBS 2014 system architecture

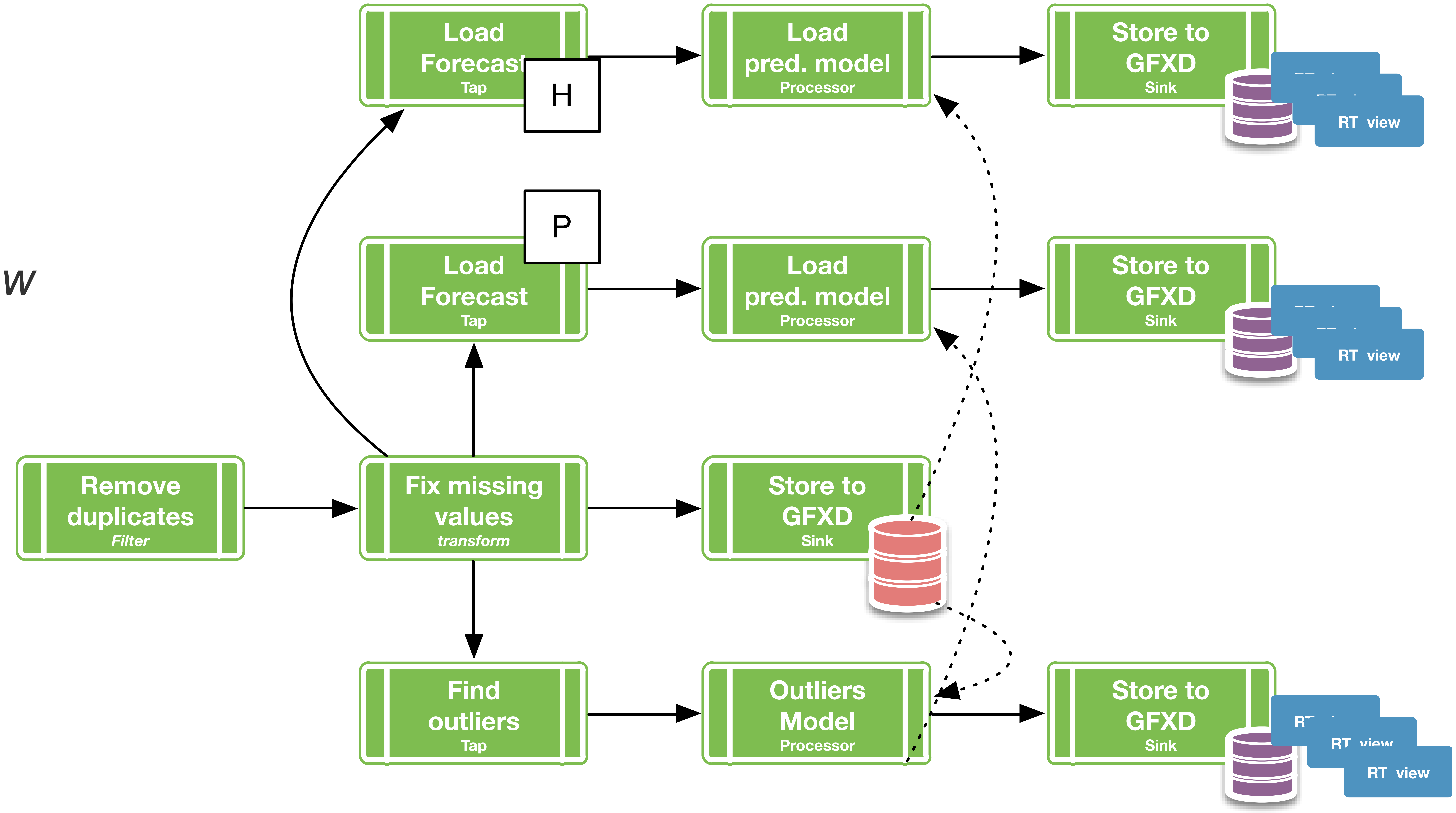


Data ingestion



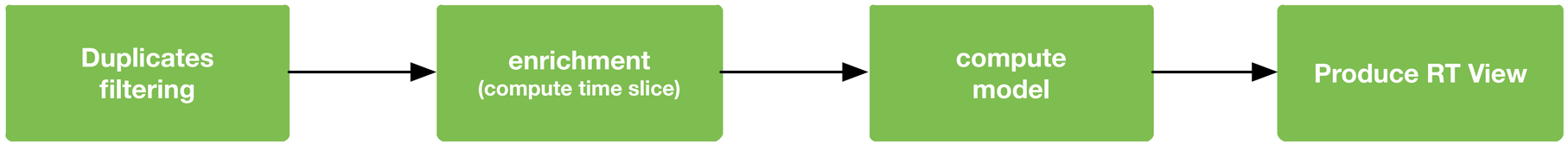
Speed layer

Application flow



not a flow - - - - - - - - - - - - - - - -

Stream actions



Spring XD streams

pumpin - sends all data to a master queue

sensoreventenricher - Consumes the data from the queue, filter and transform the data before store on HDFS

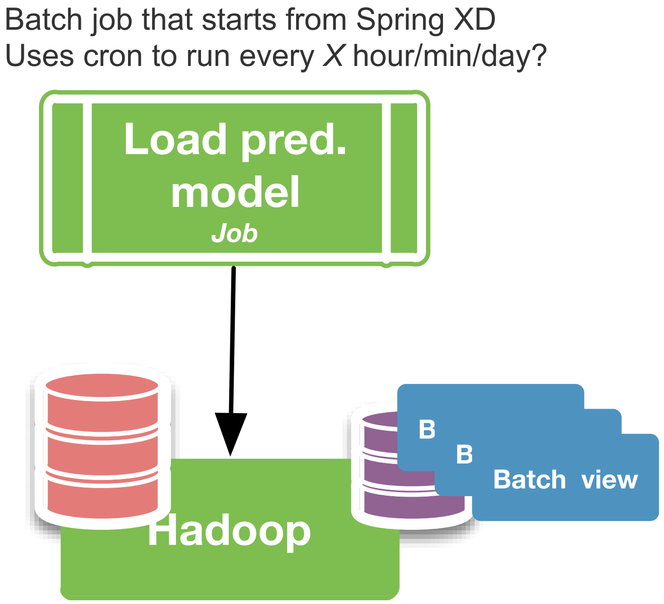
findoutliers - Taps from master_ds stream to compute outlier model

loadpred{h,p} - Taps from master_ds stream to compute load prediction for house and plug (2 streams)

5 streams

Batch Layer

Hadoop based system



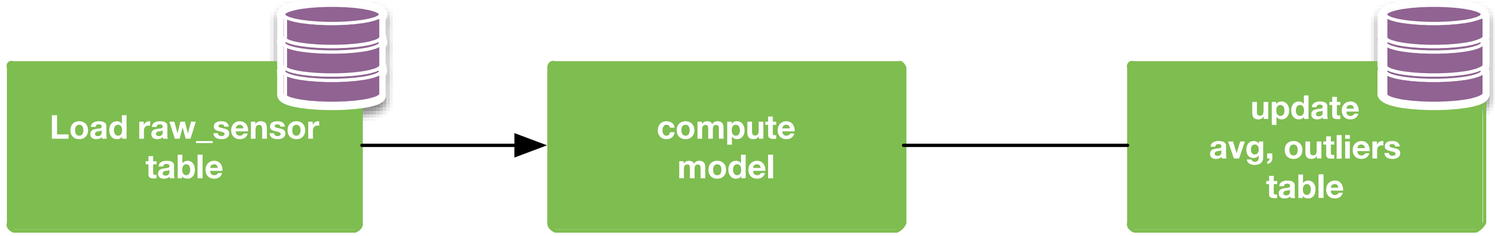
Store an immutable and constantly growing master dataset (of all datasets)

Compute arbitrary functions (models) on the existing datasets.

Essentially, runs the MR models.

Spring XD jobs

- A unique job to run the MR model to compute the historical aspect of the models.
- MR job runs every “day/hour/min”
- Job is completely independent from the other streams



1 job

Serving layer (RT and Batch views)

Gemfire XD

Field	Comments
TS	TIMESTAMP OF STARTING TIME PREDICTION
HOUSE_ID	HOUSE ID
PREDICTED_LOAD	PREDICTED LOAD

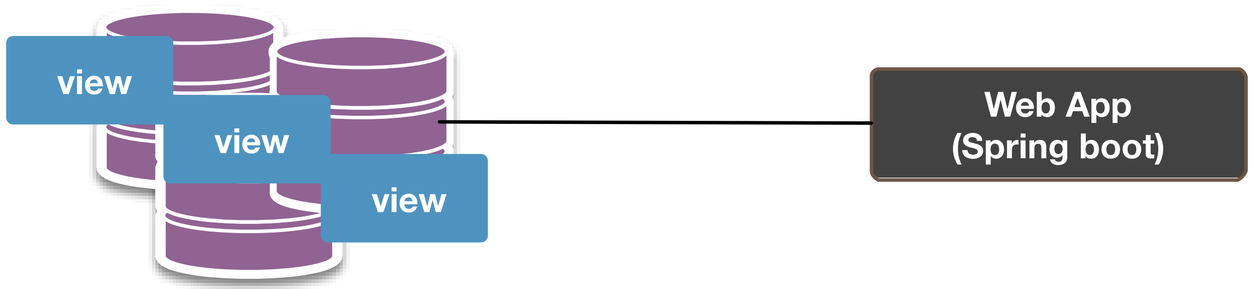
Field	Comments
TS_START	TIMESTAMP OF START TIME WINDOW
TS_STOP	TIMESTAMP OF STO PTIME WINDOW
HOUSE_ID	HOUSE ID
PERCENTAGE	% PLUGS LOAD HIGHER THAN NORMAL

Field	Comments
TS	TIMESTAMP OF STARTING TIME PREDICTION
HOUSE_ID	HOUSE ID
HOUSEHOLD_ID	
PLUG_ID	
PREDICTED_LOAD	PREDICTED LOAD

Get updates from MR and SP models

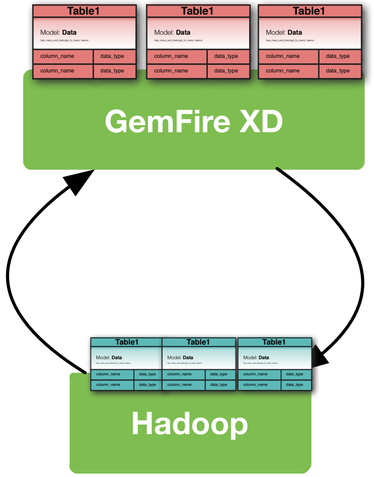
Holds both batch and real-time views

Web UI



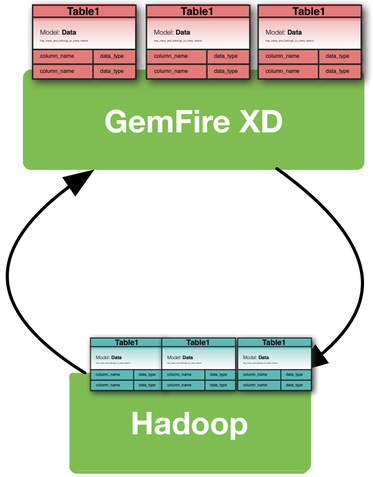
Why *Gemfire XD*?

- Distributed database
- Tightly integrated with Pivotal Hadoop
- SQL support
- Fault-tolerant
- In-memory (fast access)



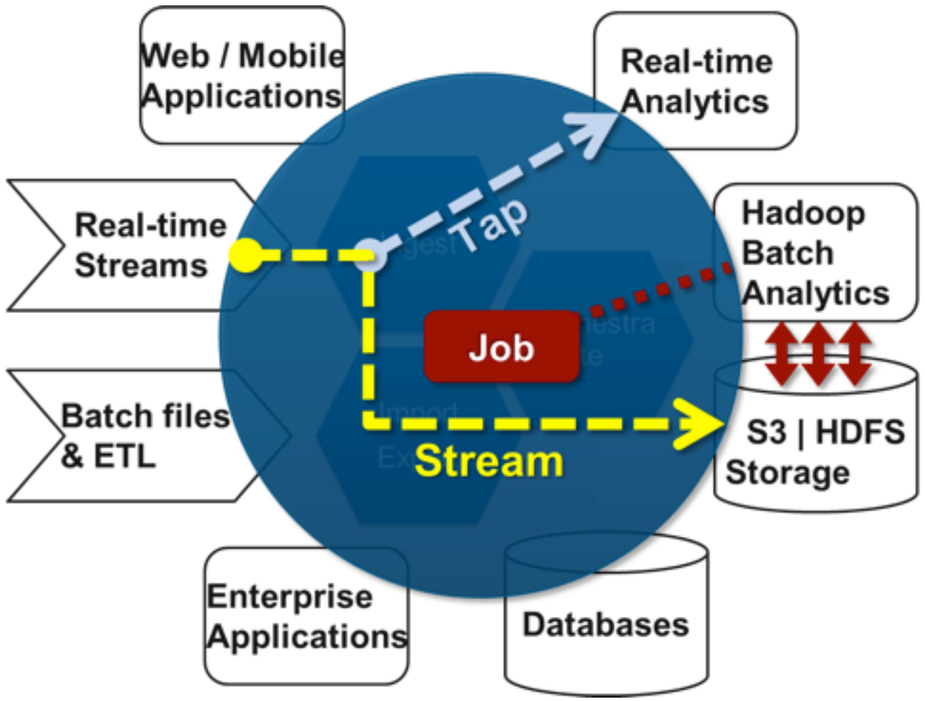
Why Pivotal HD (PHD)

- Hadoop based
- Tightly integrated with *Gemfire XD*



Why Spring XD?

- Data Ingestion
- Real-time Analytics
- Workflow Orchestration
- Integration



Full Open source implementation

HBase??

Hadoop

SpringXD

Is Lambda Architecture perfect?

- Suitable for specific use cases
- Doesn't contemplate reference data access.
- Not always possible to use same model for Real-time and Batch
- Other ideas to improve the lambda architecture
  - Multiple batch layers
  - incremental batch layers

Source code

https://bitbucket.org/caxqueiroz/debs2014-challenge

Thank you!!