Streaming Ecosystem

Reza Farivar

Capital One

Reza.farivar@capitalone.com

Components of a streaming ecosystem

- Gather the data
  - Funnel
- Distributed Queue
- Real-Time Processing
- Semi-Real-Time Processing
- Real-time OLAP

Step 1: Gather the Data

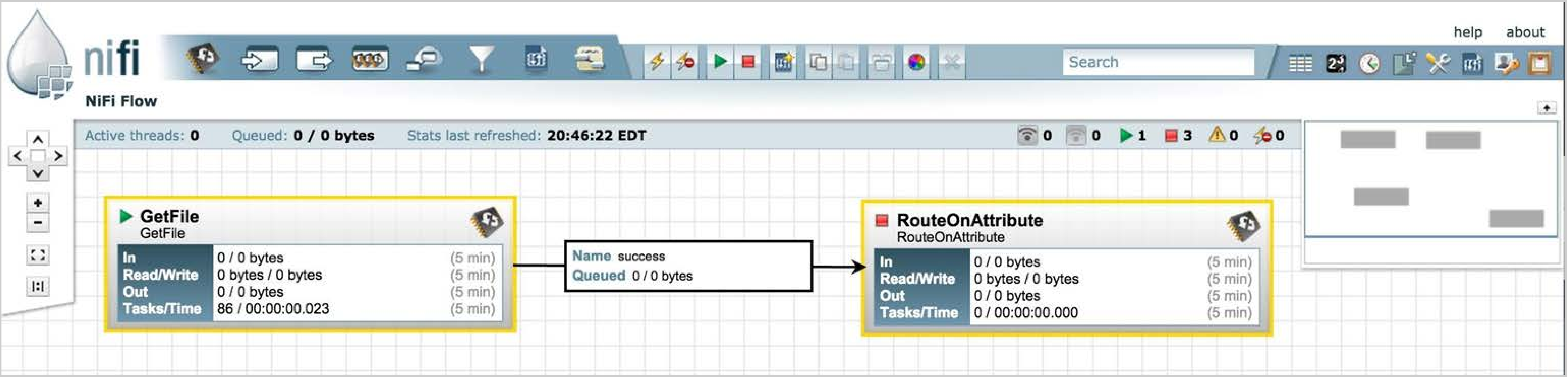
- Apache NiFi is a good distributed funnel
- Was made in NSA
  - Over 8 years of development
- Open sourced in 2014 and picked up by HortonWorks
- Great visual UI to design a data flow
- Has many many processor types in the box
- But not very good for heavy weight distributed processing
  - Same graph is executed on all the nodes

NiFi Components

- FlowFile
  - Unit of data moving through the system
  - Content + Attributes (key/value pairs)
- Processor
  - Performs the work, can access FlowFiles
- Connection
  - Links between processors
  - Queues that can be dynamically prioritized
- Process Group
  - Set of processors and their connections
  - Receive data via input ports, send data via output ports

NiFi GUI

- Drag and drop processors to build a flow
- Start, stop, and configure components in real time
- View errors and corresponding error messages
- View statistics and health of data flow
- Create templates of common processor & connections



Active threads: **0** Queued: **0 / 0 bytes** Stats last refreshed: **20:46:22 EDT** 0 0 ▶ 1 🟥 3 ⚠ 0 0

▶ **GetFile**
GetFile

In	0 / 0 bytes	(5 min)
Read/Write	0 bytes / 0 bytes	(5 min)
Out	0 / 0 bytes	(5 min)
Tasks/Time	86 / 00:00:00.023	(5 min)

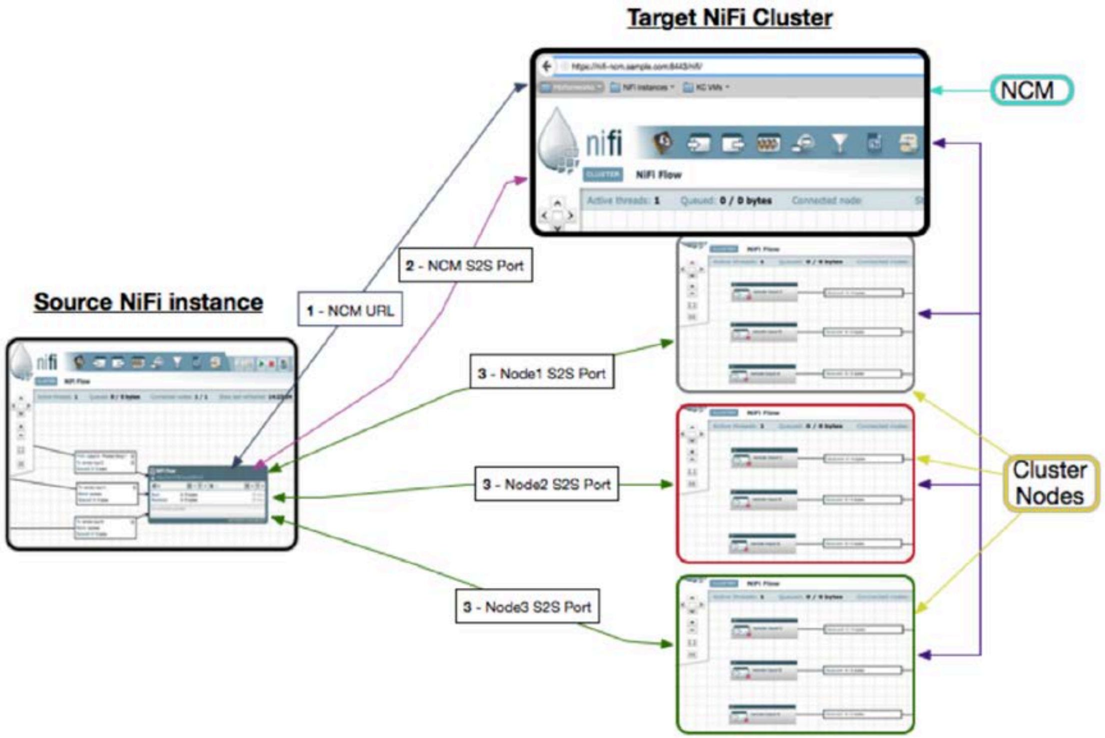
Name success
Queued 0 / 0 bytes

🟥 **RouteOnAttribute**
RouteOnAttribute

In	0 / 0 bytes	(5 min)
Read/Write	0 bytes / 0 bytes	(5 min)
Out	0 / 0 bytes	(5 min)
Tasks/Time	0 / 00:00:00.000	(5 min)

NiFi Site-to-Site

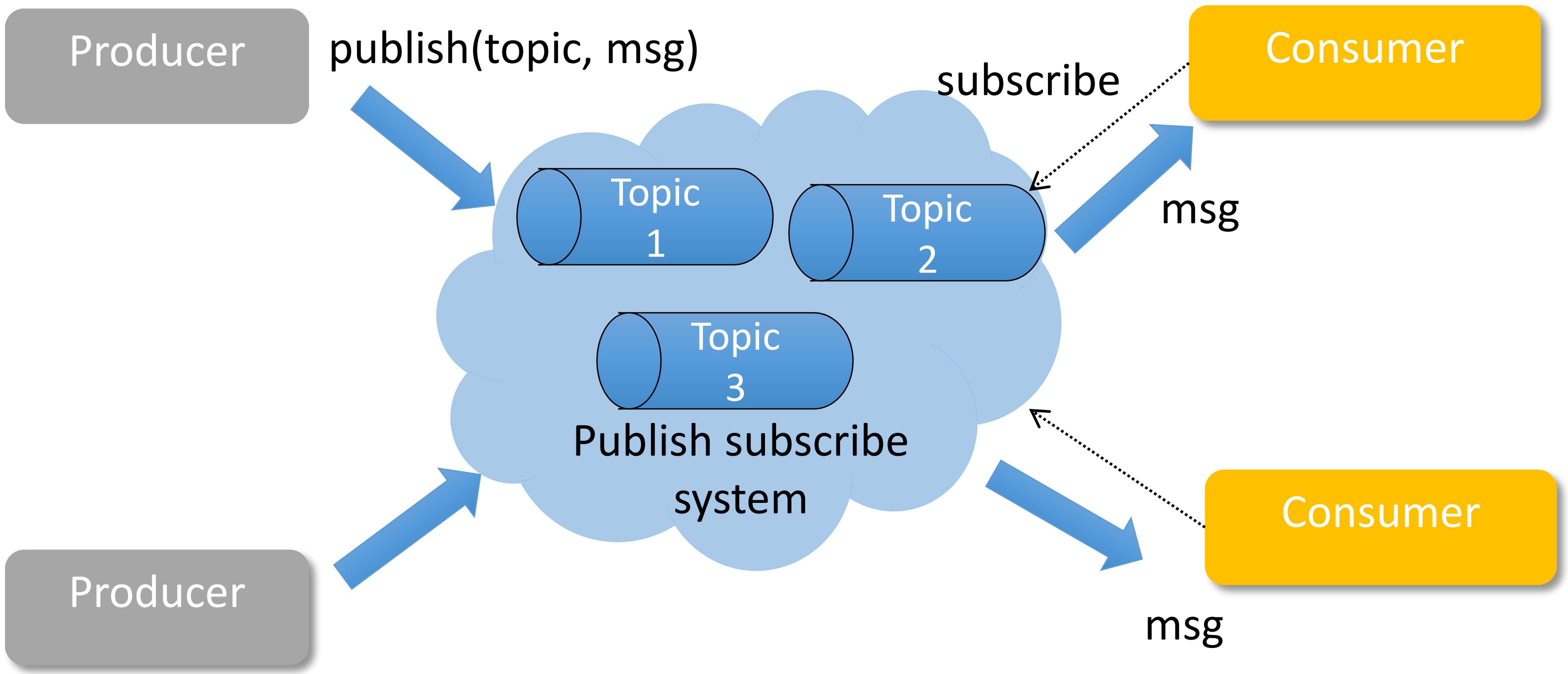
- Site-to-site allows very easy pushing of data from one data center to another
- Makes it a great choice for distributed funnel



Step 2: Distributed Queue

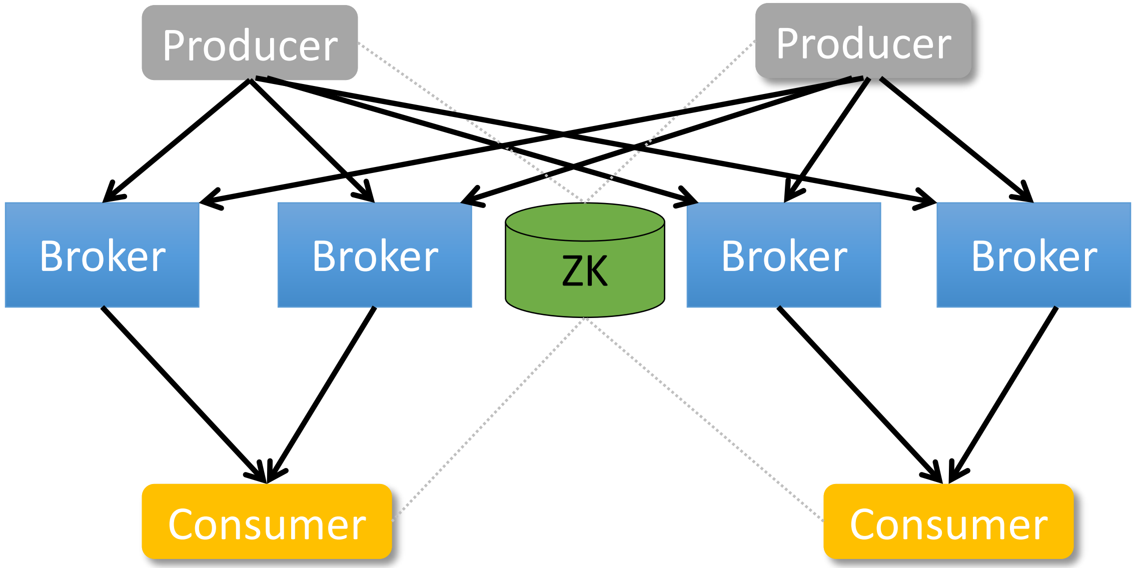
- Pub-sub model

- Kafka a very poular example



Kafka Architecture

- Distributed, high-throughput, pub-sub messaging system
  - Fast, Scalable, Durable
- Main use cases:
  - log aggregation, real-time processing, monitoring, queueing
- Originally developed by LinkedIn
- Implemented in Scala/Java



Kafka Manager

- There are some CLI tools

```
kafka-console-producer
kafka-console-consumer
Kafka-topics
kafka-consumer-offset-checker
```

- Some very new open-source projects for monitoring Kafka
  - Kafka-manager by yahoo
  - https://github.com/yahoo/kafka-manager

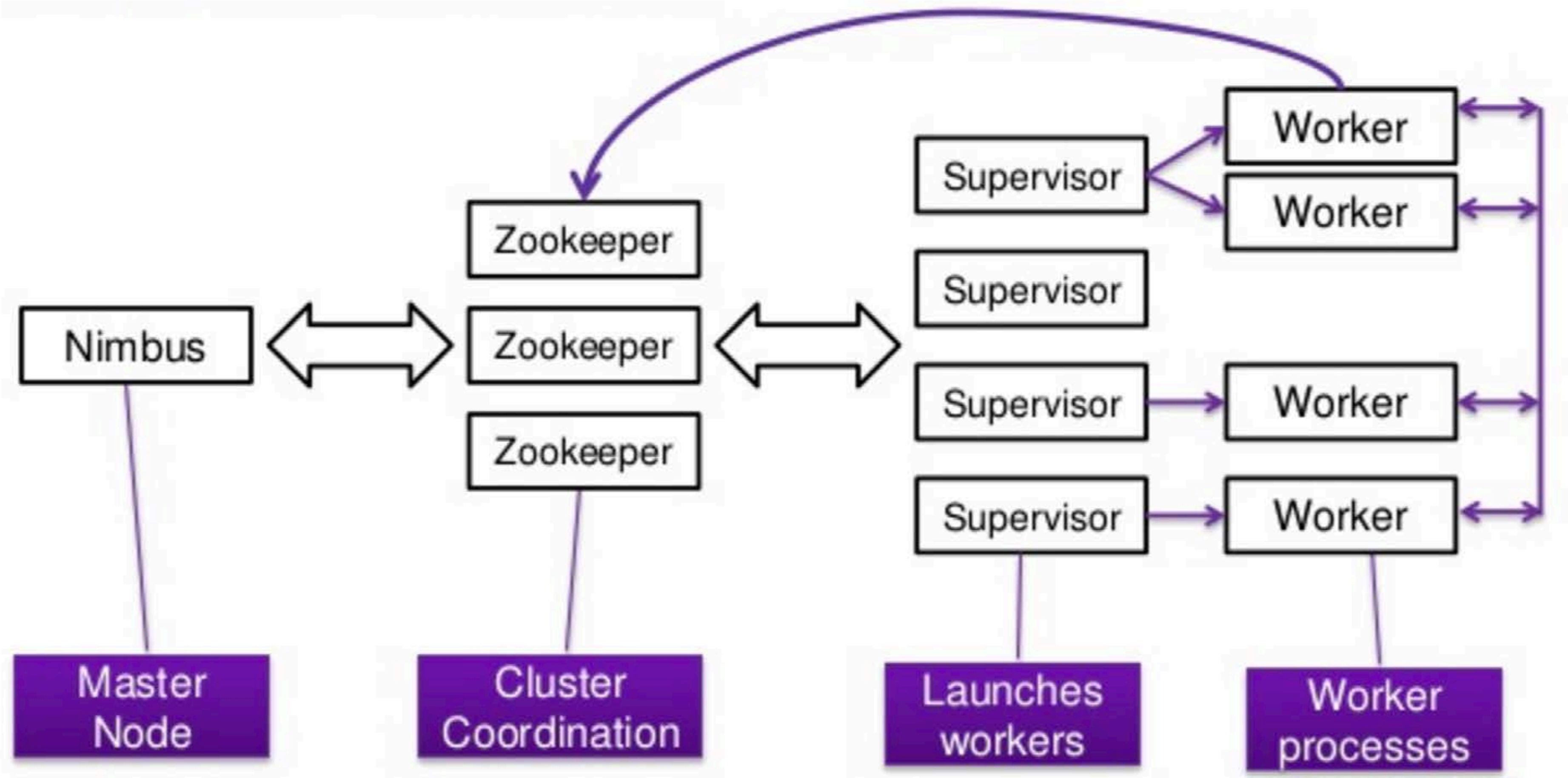
Step 3: Distributed Processing

- Once data is in the Kafka message broker, we need to process it
- Filter
- Join
- Windowing
- Business logic
- Real-time requirements
    - Sub ms to 10 ms

Storm

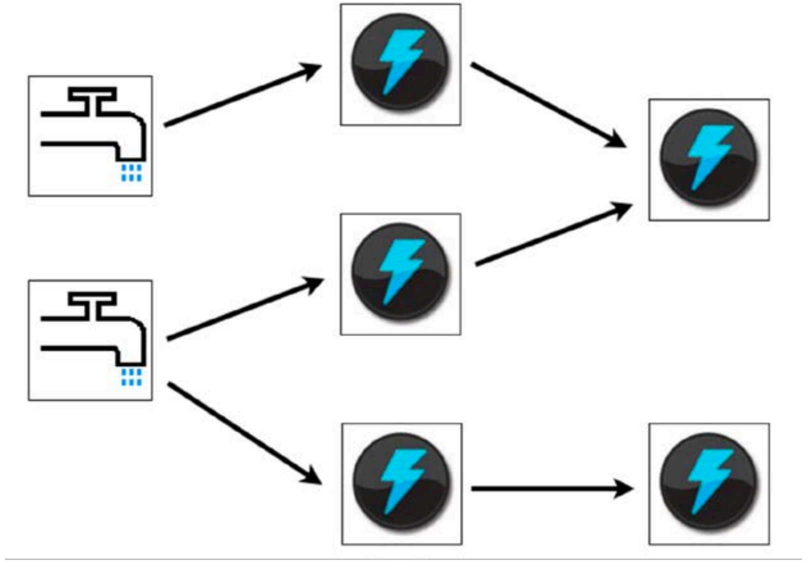
- Apache Storm
- Built in backtype, sold to Twitter
- Written in Clojure

Storm Architecture



Storm programming

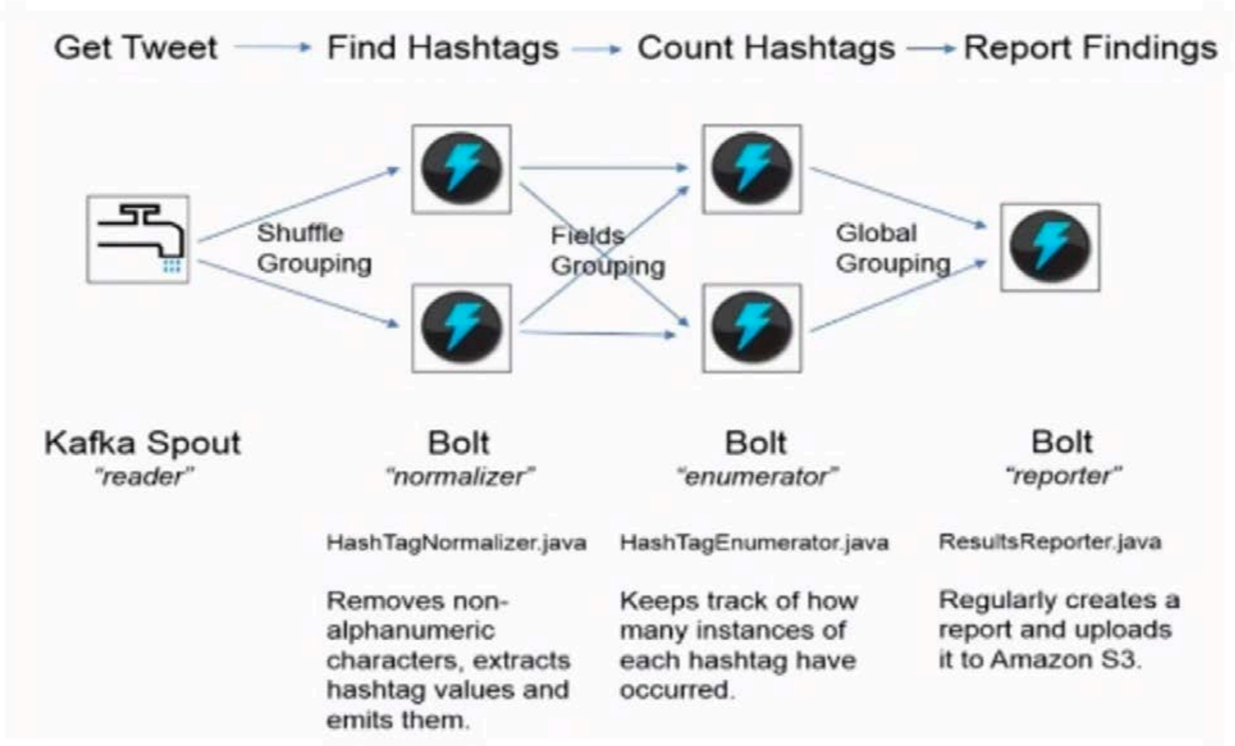
- Topology
  - Spouts
  - Bolts
  - Tuples
  - Streams
- topologyBuilder API



```
TopologyBuilder builder = new TopologyBuilder();
builder.setSpout("words", new TestWordSpout(), 10);
builder.setBolt("exclaim1", ne ExclamationBolt(), 3)
            .shuffleGrouping("words");
builder.setBolt("exclaim2", new ExclamationBolt(), 2)
            .shuffleGrouping("exclaim1");
```

Example topology

- Storm is great for non-trivial large scale processing
- Mature enterprise level features, including multitenancy and security
- Work on resource aware scheduling



Step 5: Micro batch processing / SQL / ML

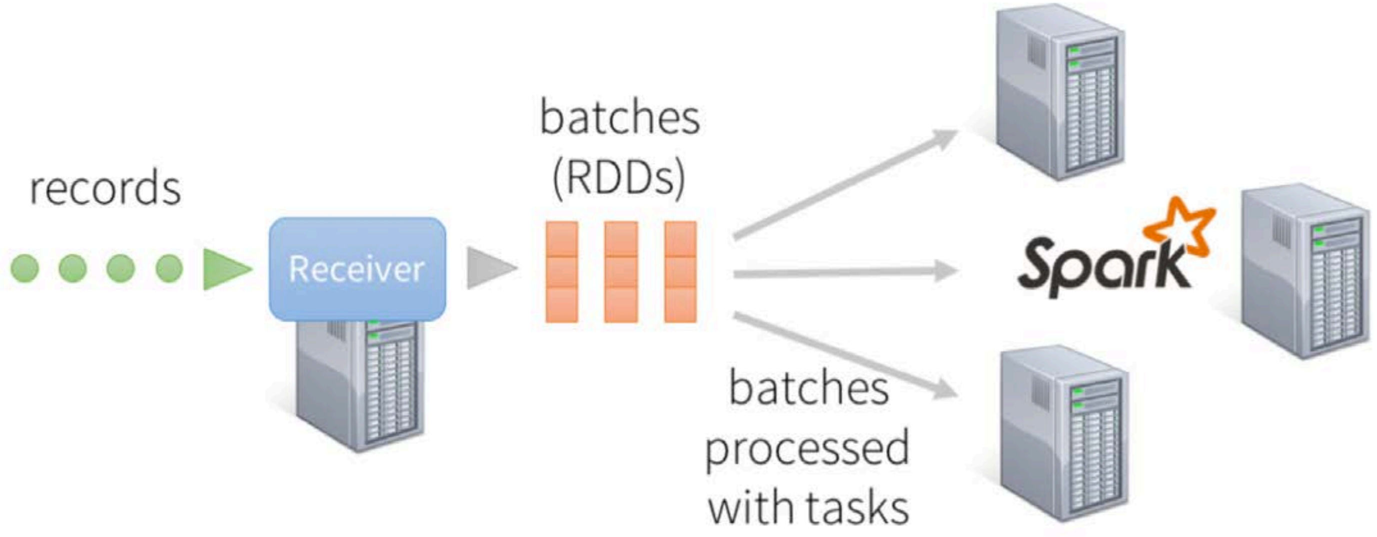
- Instead of real-time event-by event processing, we can do micro batch
- Reduce overheads
- Fault tolerance → Kappa architecture
- High latency

Spark

- Spark was a project out of Berkeley from 2010
- Has become very popular
- Most contributed open source project in big-data domain
- RDD: Resilient Distributed Data Set

Spark Streaming

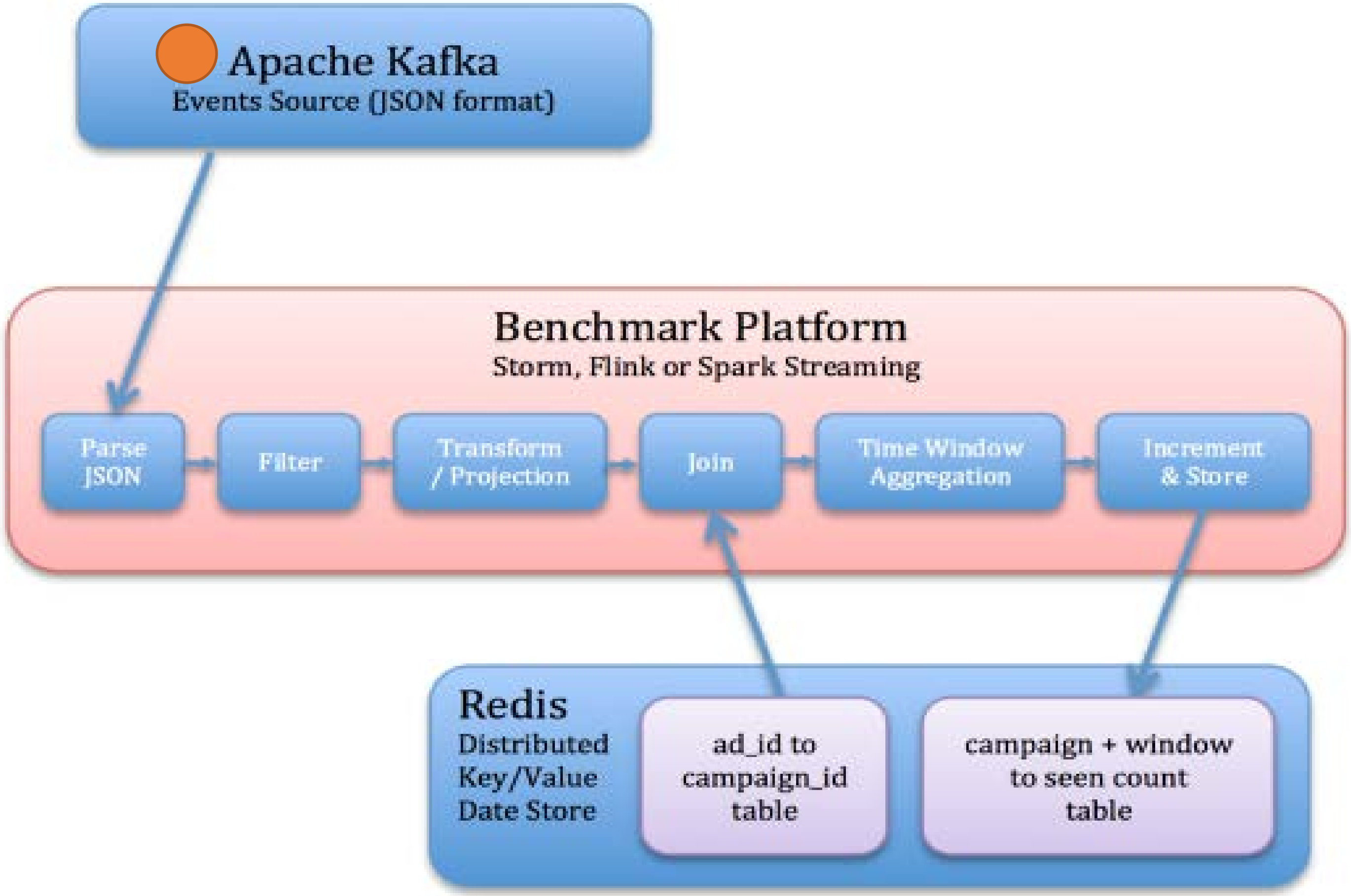
- Window a bit of data
- Run a batch
- Repeat



Spark ML, Graph, etc.

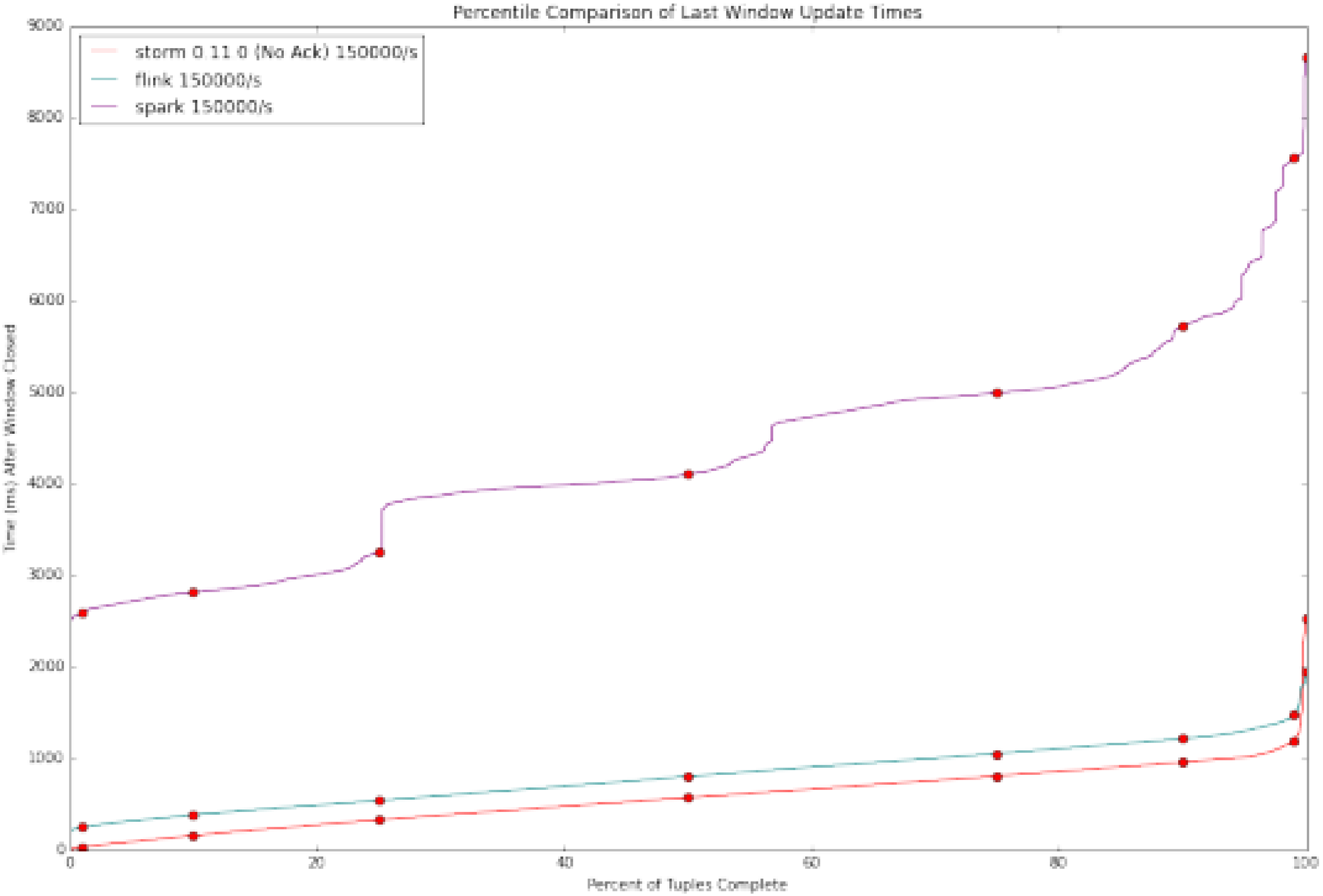
- Advantage of Spark Streaming:
  - Rich ecosystem of big data tools
  - Spark SQL
  - Spark ML
  - Spark GraphX
  - SparkR
- Disadvantage:
  - Not really streaming

Benchmark: ETL pipeline


Apache Kafka
Events Source (JSON format)
Benchmark Platform
Storm, Flink or Spark Streaming
Parse JSON
Filter
Transform / Projection
Join
Time Window Aggregation
Increment & Store
Redis
Distributed Key/Value Date Store
ad_id to campaign_id table
campaign + window to seen count table

Three-way Comparison

- Flink and Storm have similar linear performance profiles
  - These two systems process an incoming event as it becomes available
- Spark Streaming has much higher latency, but is expected to handle higher throughputs
  - System behaves in a stepwise function, a direct result from its micro-batching nature



Side note: in-memory key-value store

- Redis
- Cassandra

Step 6: OLAP (Online Analytical Processing)

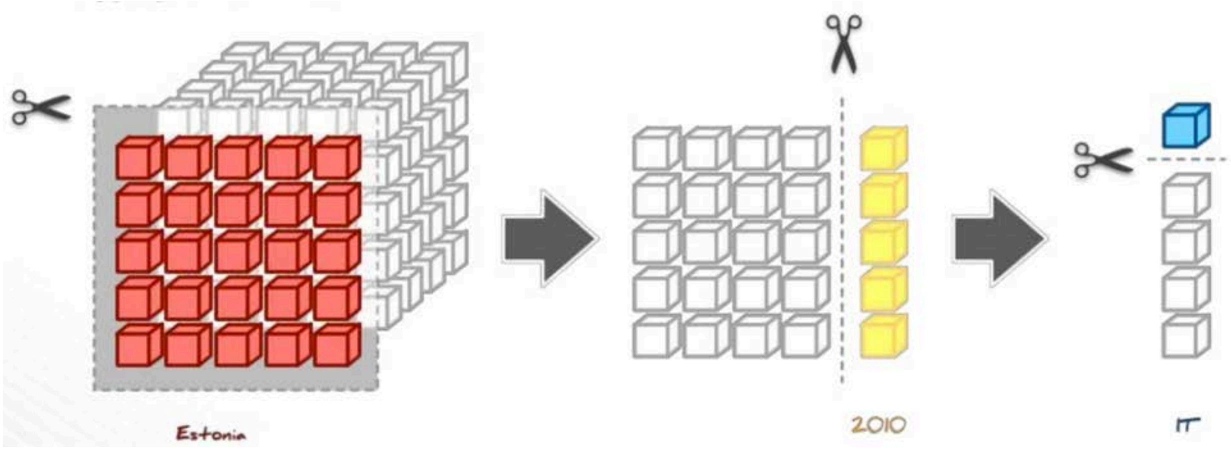
- Business Intelligence
- Multidimensional data analytics
- Analyze multidimensional data interactively
- Basic Operations
  - Consolidation (roll-up, aggregation in dimensions)
  - Drill-down (filter)
  - Slicing and dicing (Look at the data from different viewpoints)

Druid

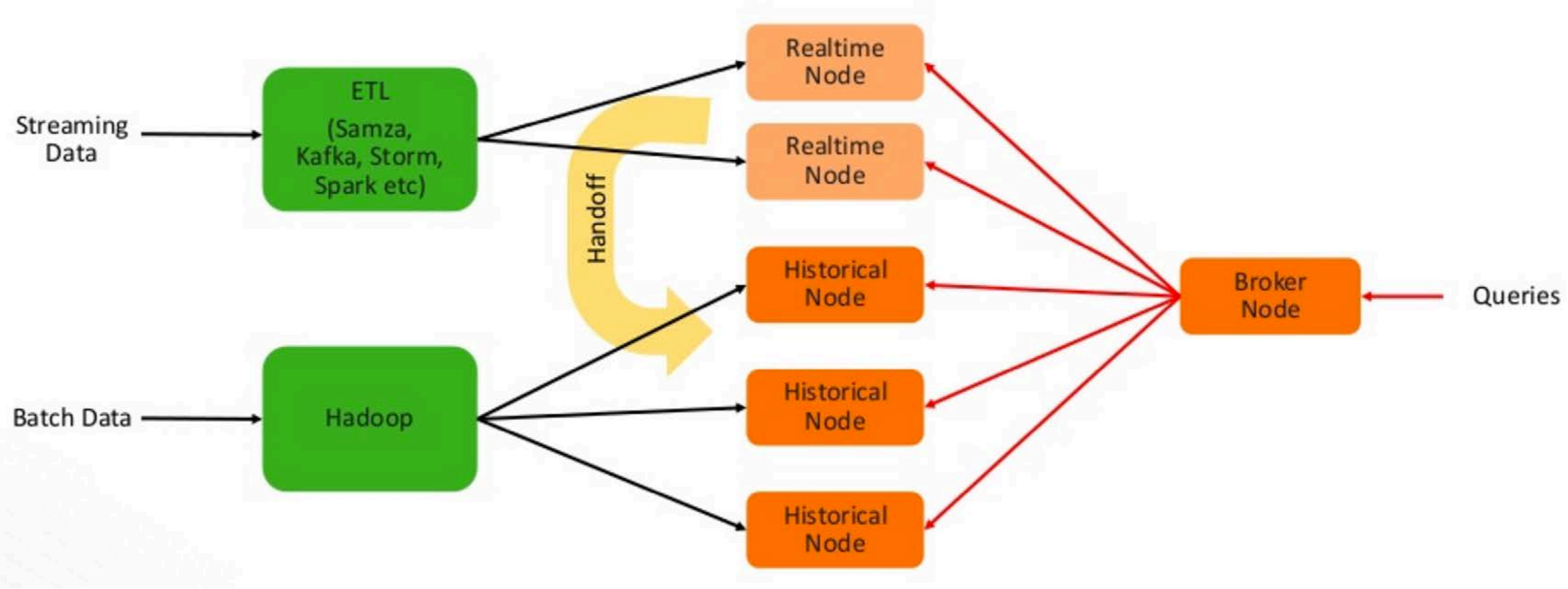
- Developed in Metamarkets in 2011
  - RDBMs: Too slow
  - NoSQL key value store: fast, but exponential memory space, precompute very slow
- Gaining in popularity
- Open Source (Apache license) in late 2012
- OLAP queries
- Column oriented
- Sub second query time (Avg query time 0.5 seconds)
- Real-time streaming ingestion
- Scalable

Druid

- Arbitrary slice and dive of data

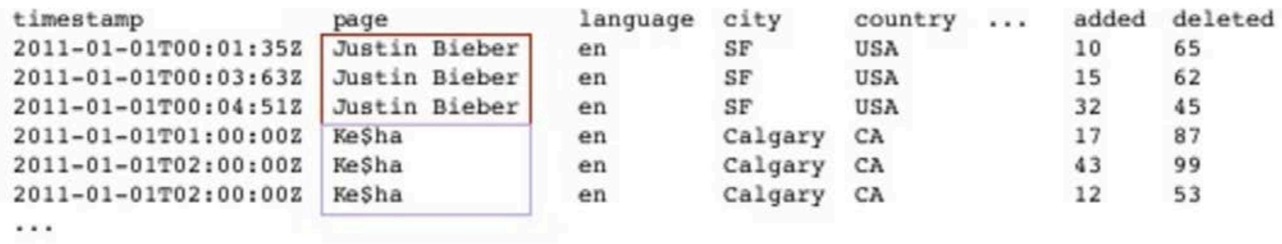


Druid Architecture

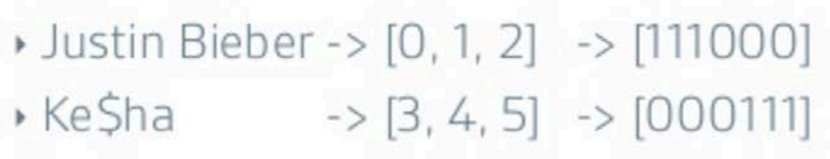


Druid Bitmap Index

- This is one of the reasons Druid is so fast
- Dictionary encoding
- Bitmap Index
- Compression ratio: 1 bit per record
- Logical AND/OR of a few thousand numbers for a query → lightning fast queries



timestamp	page	language	city	country	...	added	deleted
2011-01-01T00:01:35Z	Justin Bieber	en	SF	USA		10	65
2011-01-01T00:03:63Z	Justin Bieber	en	SF	USA		15	62
2011-01-01T00:04:51Z	Justin Bieber	en	SF	USA		32	45
2011-01-01T01:00:00Z	Ke$ha	en	Calgary	CA		17	87
2011-01-01T02:00:00Z	Ke$ha	en	Calgary	CA		43	99
2011-01-01T02:00:00Z	Ke$ha	en	Calgary	CA		12	53
...							

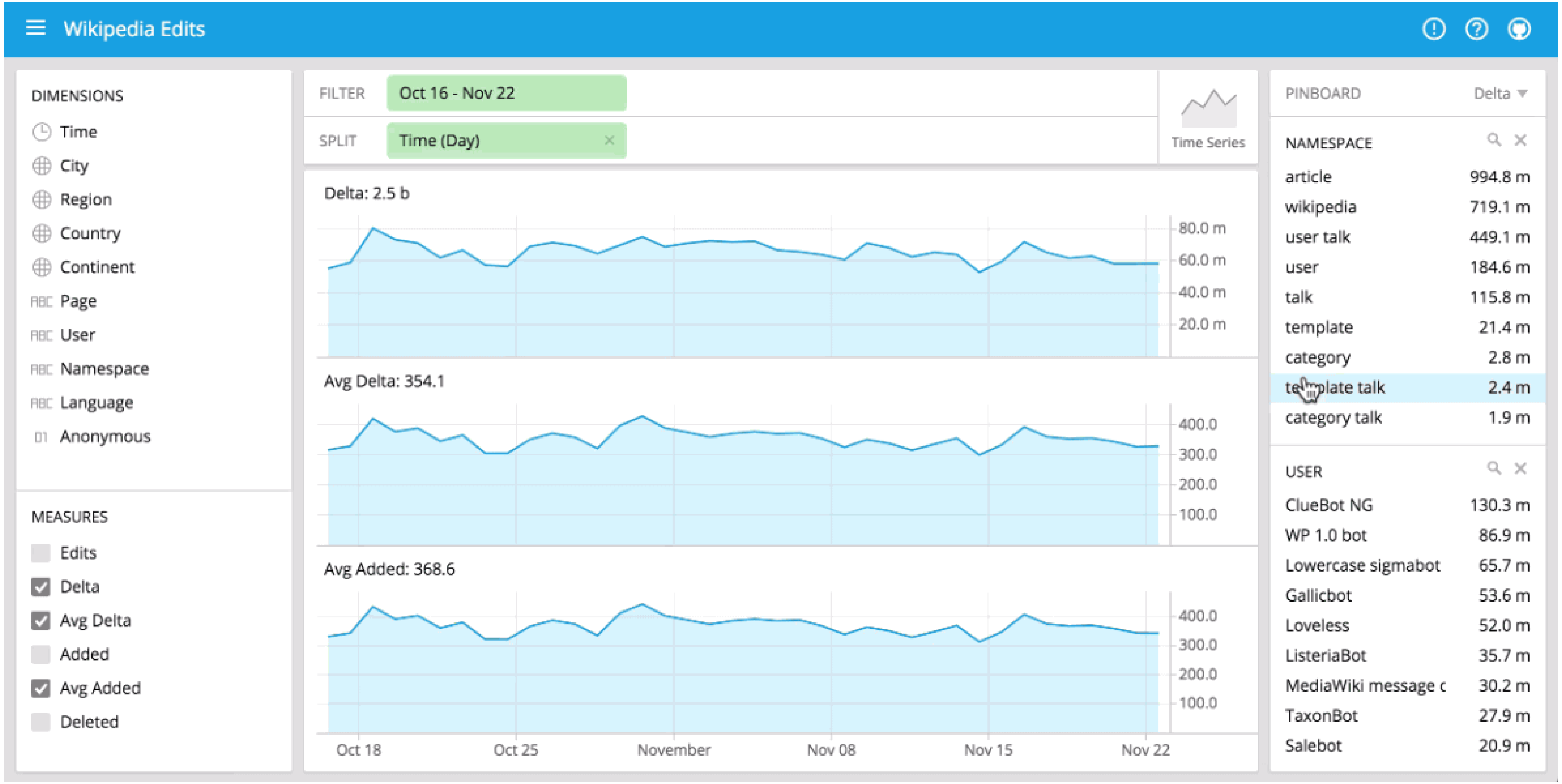


```
‣ Justin Bieber -> [0, 1, 2]  -> [111000]
‣ Ke$ha         -> [3, 4, 5]  -> [000111]
```

Step 7: BI

- Pivot
  - web-based exploratory visualization UI for Druid
  - Easily filter, split, visualize, etc.
- Tableu and SQL not natively supported ☹
  - But wait!

Pivot



FILTER: Oct 16 - Nov 22

SPLIT: Time (Day) ×

MEASURES

- □ Edits
- ☑ Delta
- ☑ Avg Delta
- □ Added
- ☑ Avg Added
- □ Deleted

NAMESPACE	
article	994.8 m
wikipedia	719.1 m
user talk	449.1 m
user	184.6 m
talk	115.8 m
template	21.4 m
category	2.8 m
template talk	2.4 m
category talk	1.9 m

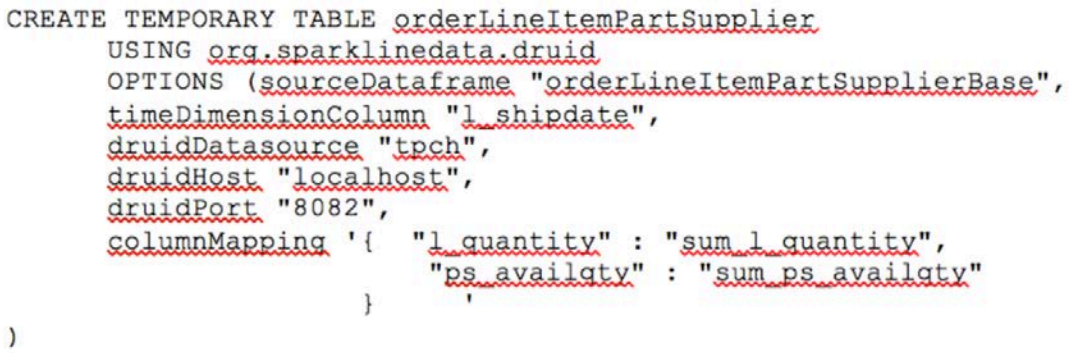
USER	
ClueBot NG	130.3 m
WP 1.0 bot	86.9 m
Lowercase sigmabot	65.7 m
Gallicbot	53.6 m
Loveless	52.0 m
ListeriaBot	35.7 m
MediaWiki message c	30.2 m
TaxonBot	27.9 m
Salebot	20.9 m

Druid and Spark

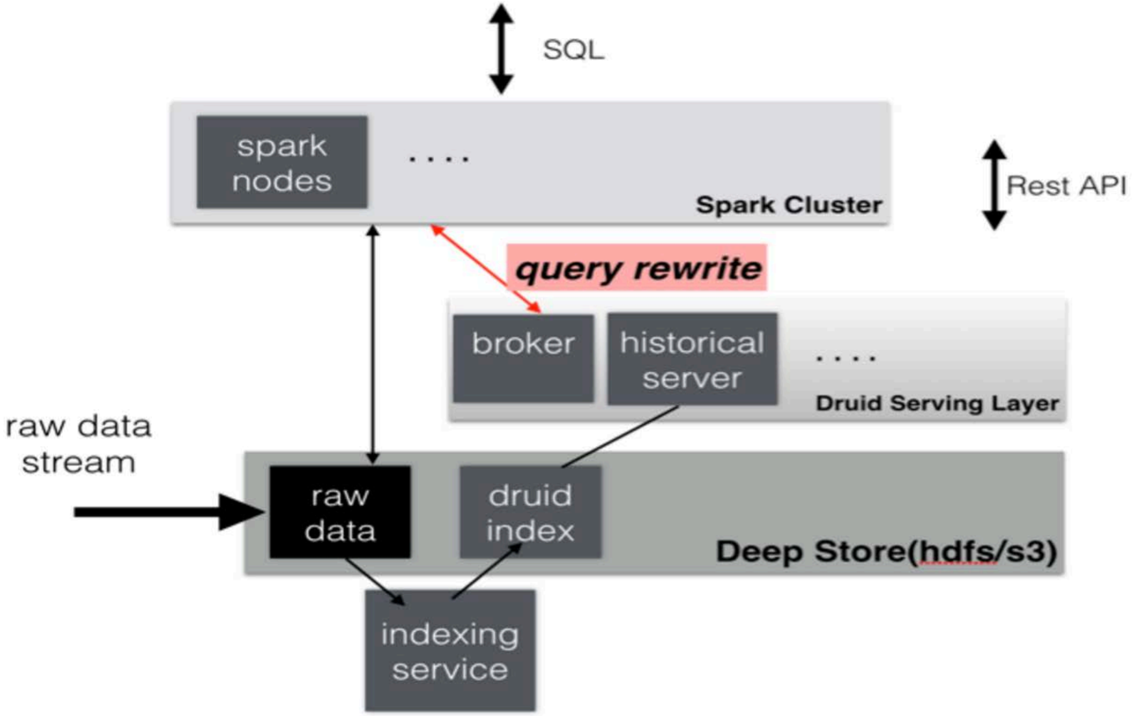
- Druid’s native API is JSON
- No Tableau, SQL support
- But there is hope!

  https://github.com/SparklineData/spark-druid-olap

- Connect Druid to Tableu through Spark

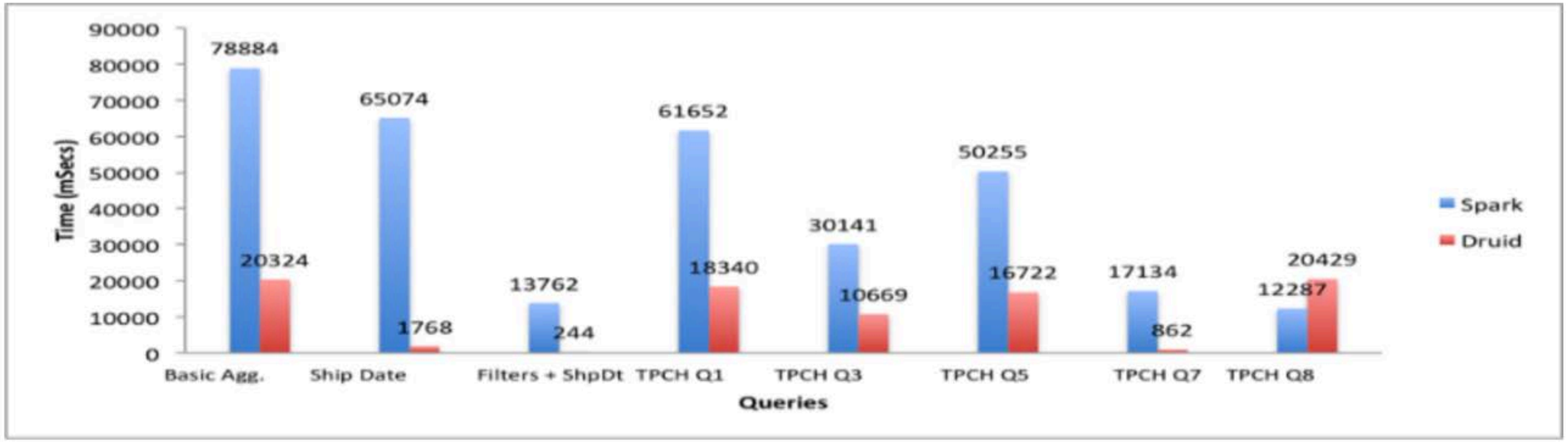


```
CREATE TEMPORARY TABLE orderLineItemPartSupplier
       USING org.sparklinedata.druid
       OPTIONS (sourceDataframe "orderLineItemPartSupplierBase",
       timeDimensionColumn "l_shipdate",
       druidDatasource "tpch",
       druidHost "localhost",
       druidPort "8082",
       columnMapping '{  "l_quantity" : "sum_l_quantity",
                          "ps_availqty" : "sum_ps_availqty"
                     }      '
)
```



Why Druid and Spark together?

- Spark is great as a general engine
- Everything and the kitchen sink
- Queries can take a long time
  - Still much faster than Hive on Yarn
- Druid is optimized for Column based time-series queries



Questions?

Email: reza.farivar@capitalone.com