CLOUD COMPUTING APPLICATIONS

Streaming Introduction

Roy Campbell & Reza Farivar

Why Real-Time Stream Processing?

- Real-time data processing at massive scale is becoming a requirement for businesses
  - Real-time search, high frequency trading, social networks
  - Have a stream of events that flow into the system at a given data rate
- The processing system must keep up with the event rate or degrade gracefully by eliminating events. This is typically called load shedding

Why Real-Time Stream Processing?

- MapReduce, Hadoop, etc., store and process data at scale, but not for real-time systems
- There's no hack that will turn Hadoop into a real-time streaming system
  - Fundamentally different set of requirements than batch processing
- Lack of a "Hadoop of real-time" has become the biggest hole in the data processing ecosystem

Cloud Streaming Engines

- Apache Storm
- Twitter Heron
- Apache Flink

- Older non-cloud systems
  - IBM System S
  - Borealis
    - Descendent of Aurora from Brown University. Not active anymore

CLOUD COMPUTING APPLICATIONS

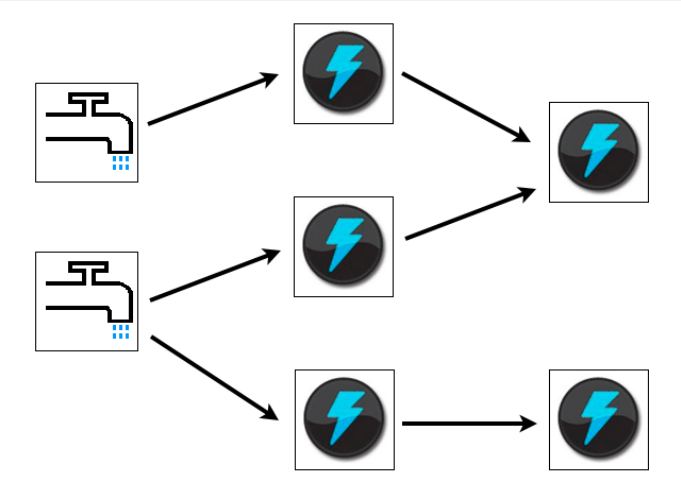
BIG DATA PIPELINES: THE RISE OF REAL-TIME

Matt Ahrens – Yahoo

The Rise of Real-Time

- As Hadoop ramped up to offer batch data availability, a growing need arose to provide data in real-time for analytic and instant feedback use cases

- Storm became stable for production scale in 2012



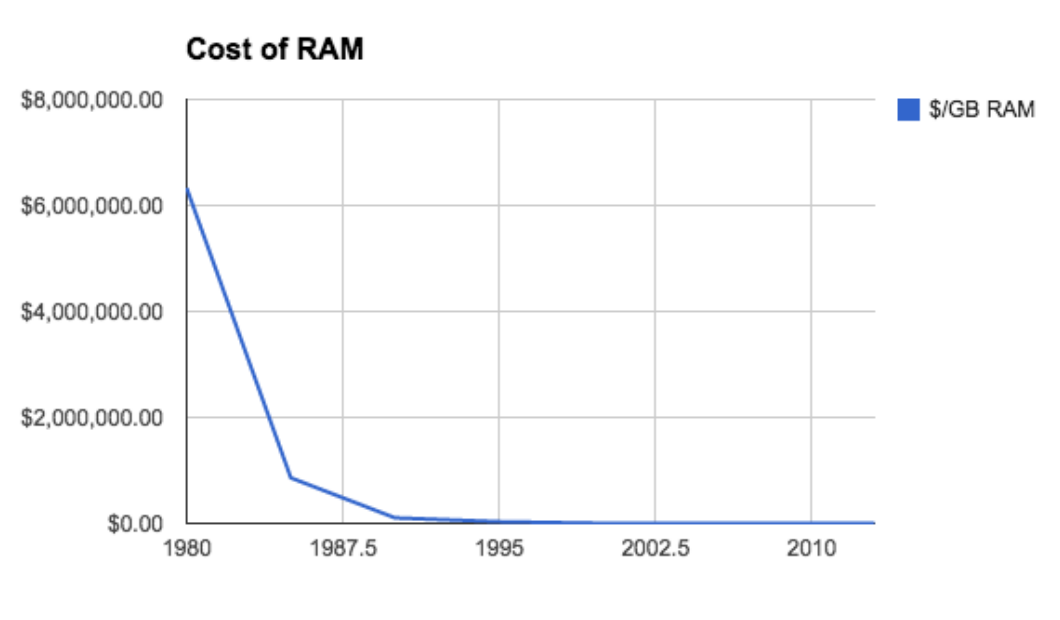
The Storm Fire Hose

- Topologies
  - graph of spouts and bolts that are connected with stream groupings
  - runs indefinitely (no time/batch boundaries)
- Streams
  - unbounded sequence of tuples that is processed and created in parallel in a distributed fashion
- Spouts
  - input source of streams in topology
- Bolts
  - processing container, which can perform transformation, filter, aggregation, join, etc.
  - sinks: special type of bolts that have an output interface

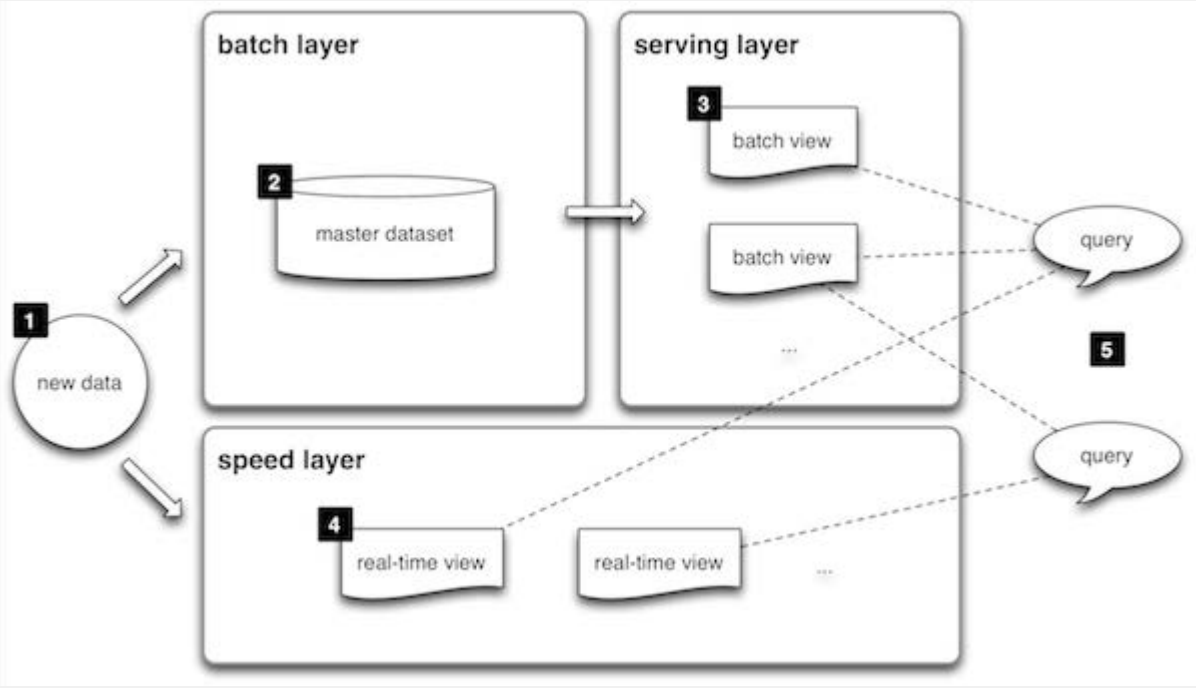
How Did We Get Here?

- People always have wanted data faster
- Finally we had hardware costs that were in line with doing in-memory streaming for billions of events/day

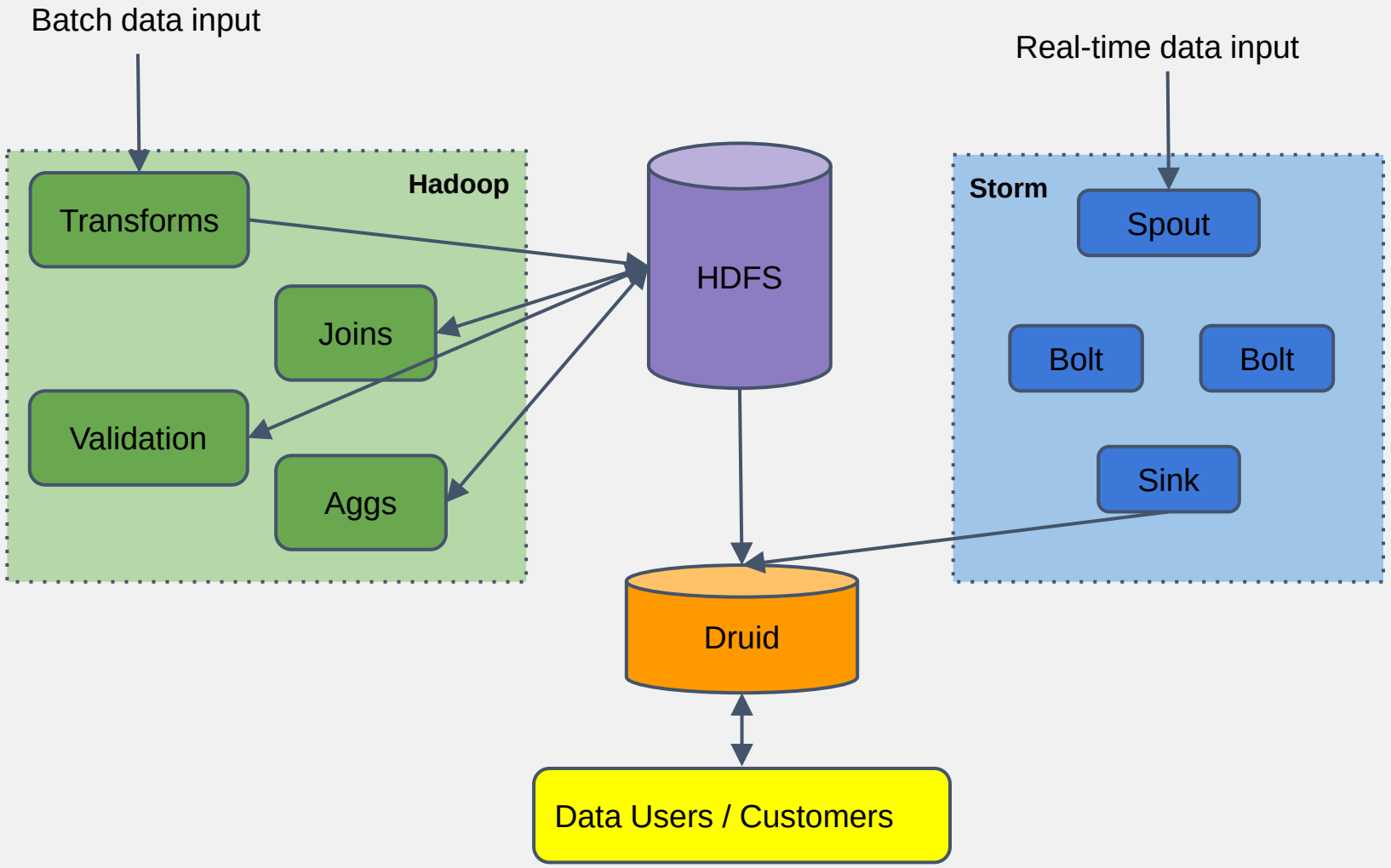
Year	$/GB RAM
1980	$6,328,125.00
1985	$859,375.00
1990	$103,880.00
1995	$30,875.00
2000	$1,107.00
2005	$189.00
2010	$12.37
2013	$5.50



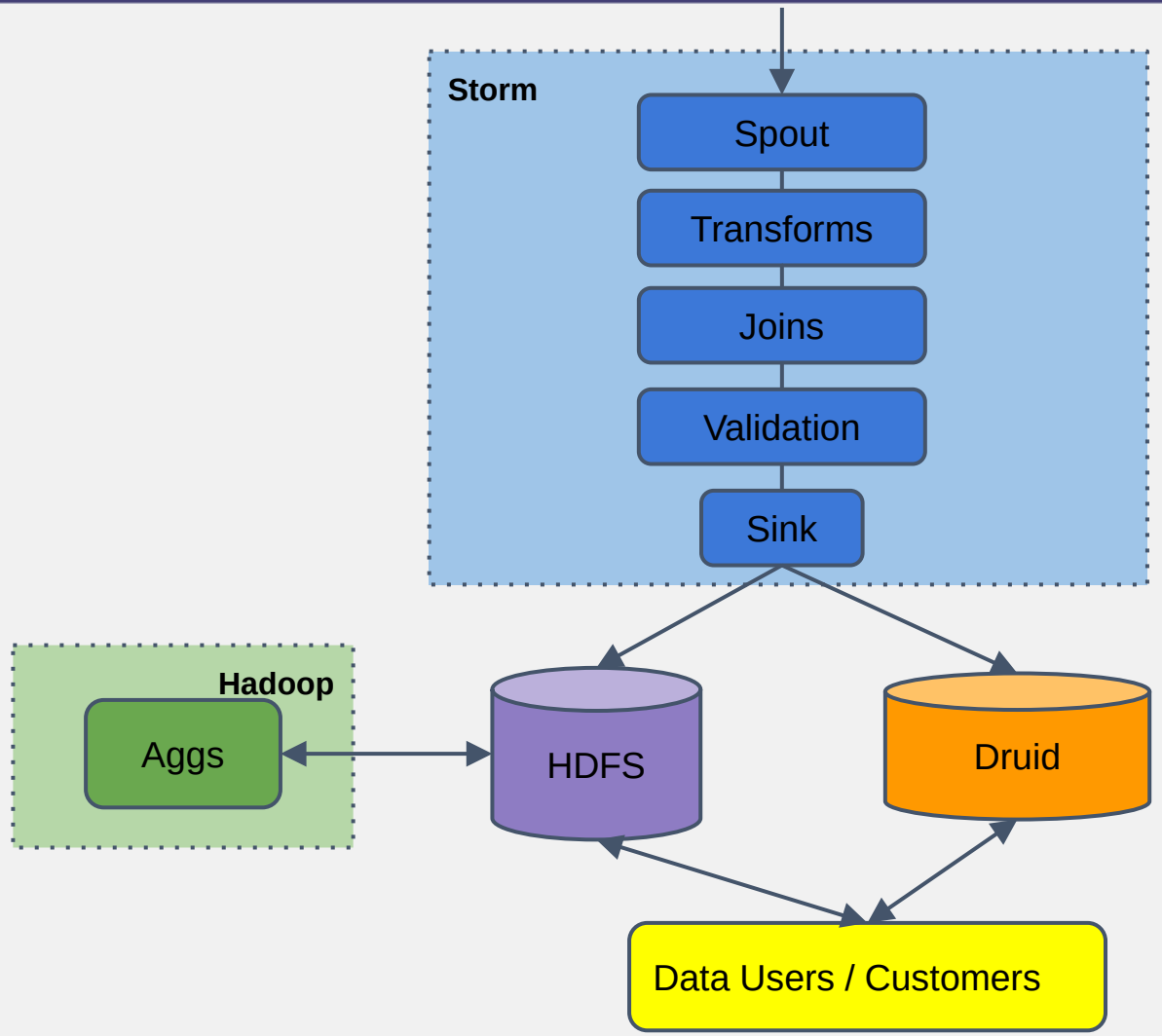
The Lambda Architecture: Real-Time + Batch



The Present Architecture



The Next Frontier:
Real-Time as Source of Truth



CLOUD COMPUTING APPLICATIONS

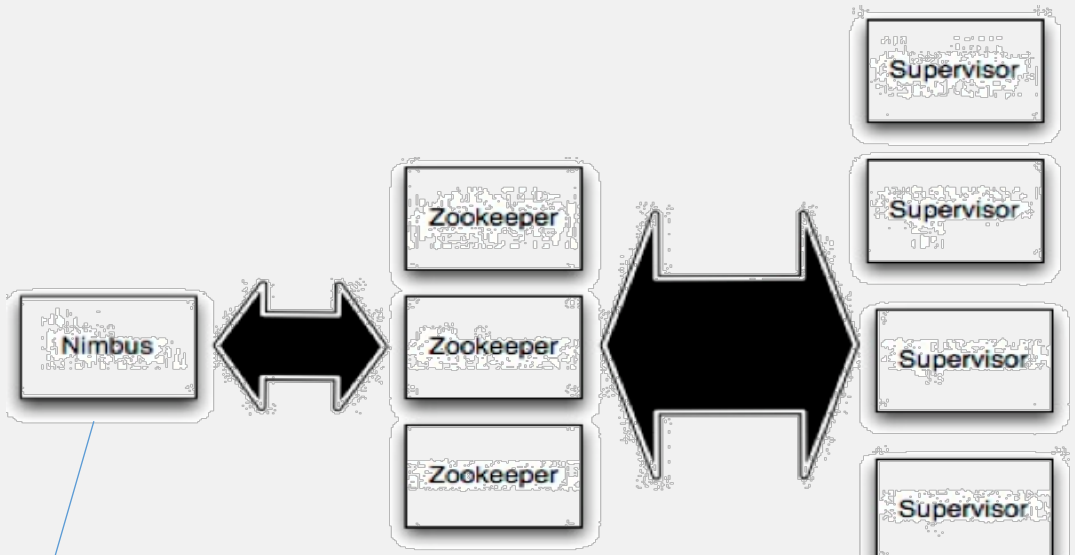
Storm Introduction: Bolts & Spouts

Roy Campbell & Reza Farivar

Apache Storm

- Guaranteed data processing
- Horizontal scalability
- Fault tolerance
- No intermediate message brokers
- Higher-level abstraction than message passing
- “Just works”
  - Hadoop of real-time streaming jobs
- Built by Backtype, then by Twitter, and eventually Apache open source

Storm



Master Node

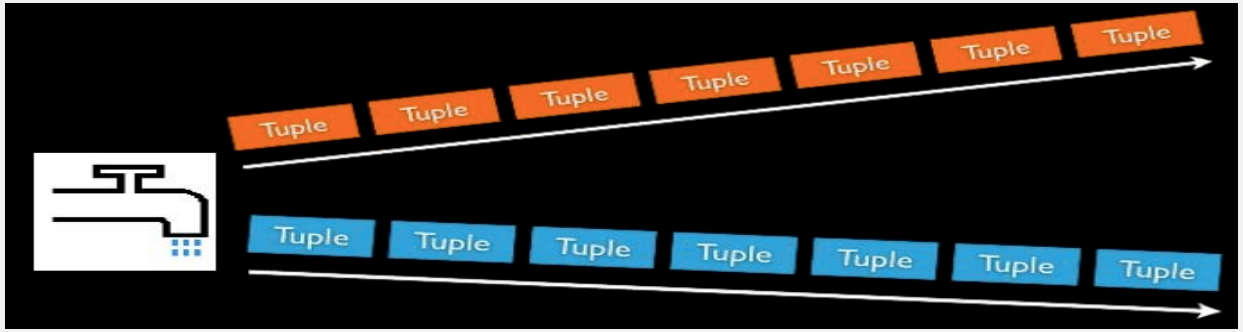
Cluster Coordination

Worker Processes

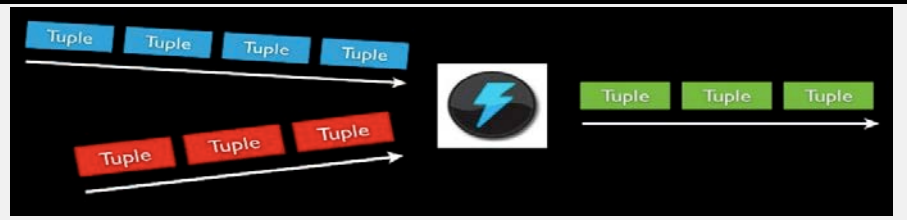
Storm Concepts

- Streams
  - Unbounded sequences of tuples

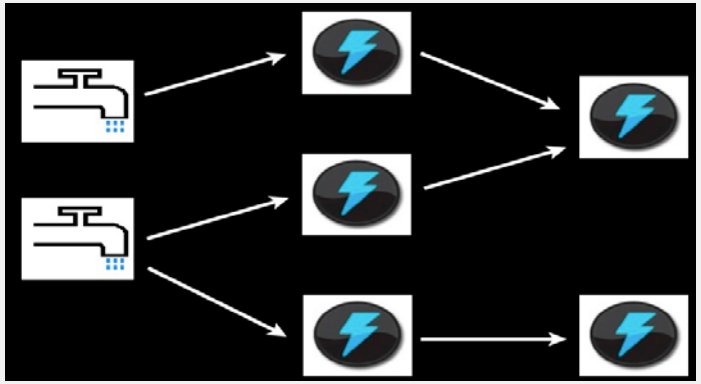
- Spout
  - Source of Streams
  - E.g., Read from Twitter streaming API



- Bolts
  - Processes input streams and produces new streams
  - E.g., Functions, Filters, Aggregation, Joins

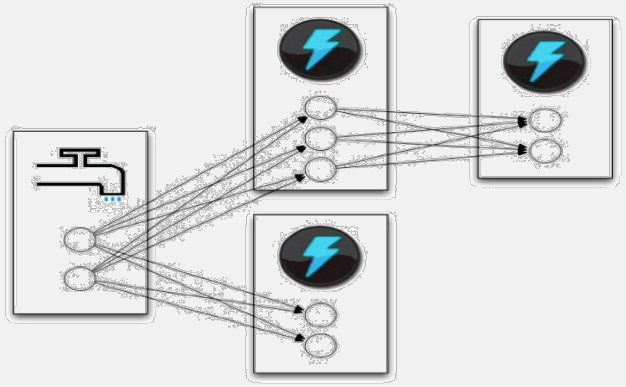


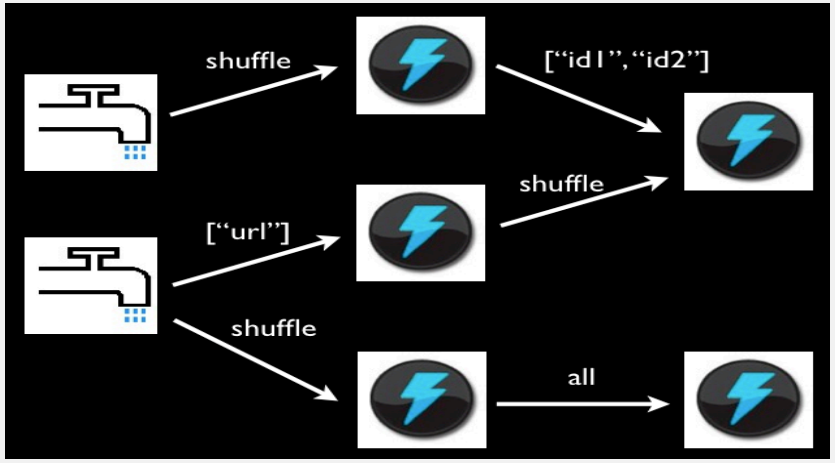
- Topologies
  - Network of spouts and bolts



Storm Tasks

- Spouts and bolts execute as many tasks across the cluster
- When a tuple is emitted, which task does it go to? → User programmable:
  - Shuffle grouping: pick a random task
  - Fields grouping: consistent hashing on a subset of tuple fields
  - All grouping: send to all tasks
  - Global grouping: pick task with lowest id



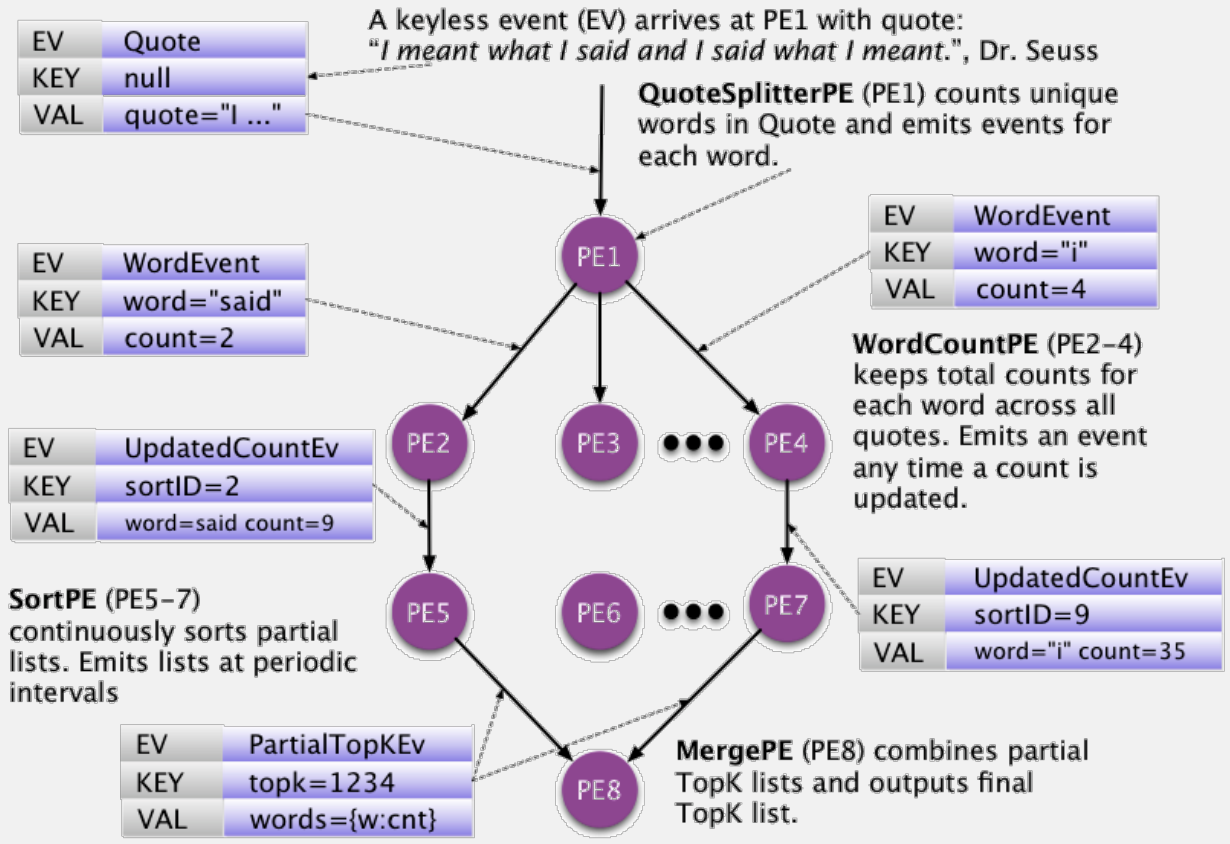


CLOUD COMPUTING APPLICATIONS

Storm Word Count Example

Roy Campbell & Reza Farivar

Streaming Word Count Example



CLOUD COMPUTING APPLICATIONS

Writing the Storm Word Count Example

Roy Campbell & Reza Farivar

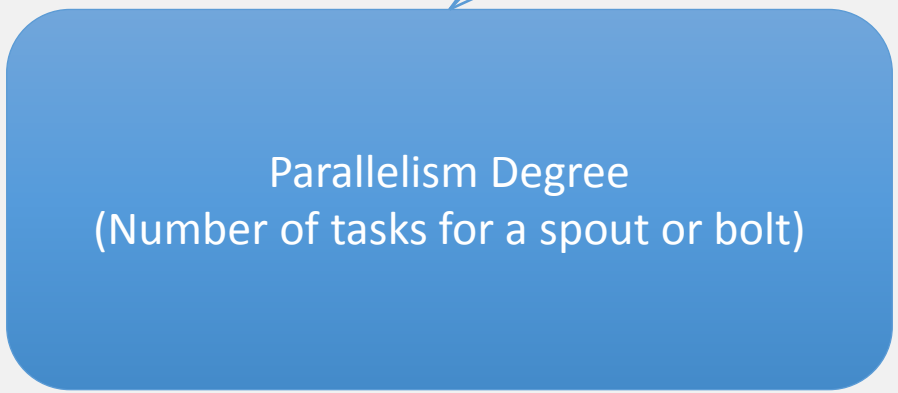
Example: Word Count in Storm

```
Public static class SplitSentence extends ShellBolt implements IRichBolt {
//Code to split a sentence
}

Public static class Word Count implements IBasicBolt{
//Code to count words; have to override the execute function
Public void execute(Tuple tuple, BasicOutputCollector collector){
    //…
    }
}
```

Example: Word Count in Storm

```
TopologyBuilder builder = new TopologyBuilder();
builder.setSpout(1, new KernelSpout("kestrel.twitter.com",
                        22133, "sentence_queue", new StringScheme()), 5);
builder.setBolt(2, new SplitSentence(), 8).shuffleGrouping(1);
builder.setBolt(3, new Word Count(), 12)
                                .fieldGrouping(2, new Fields("word"));
```



Example: Word Count in Storm

```
StormSubmitter.submitTopology(“word count”, builder.createTopology);
```

CLOUD COMPUTING APPLICATIONS

Scaling Storm to 4000 nodes with Bobby Evans, Yahoo

Roy Campbell & Reza Farivar

Open Source Big Data @Yahoo

Bobby (Robert) Evans
bobby@apache.org
@bobbydata
Architect @ Yahoo

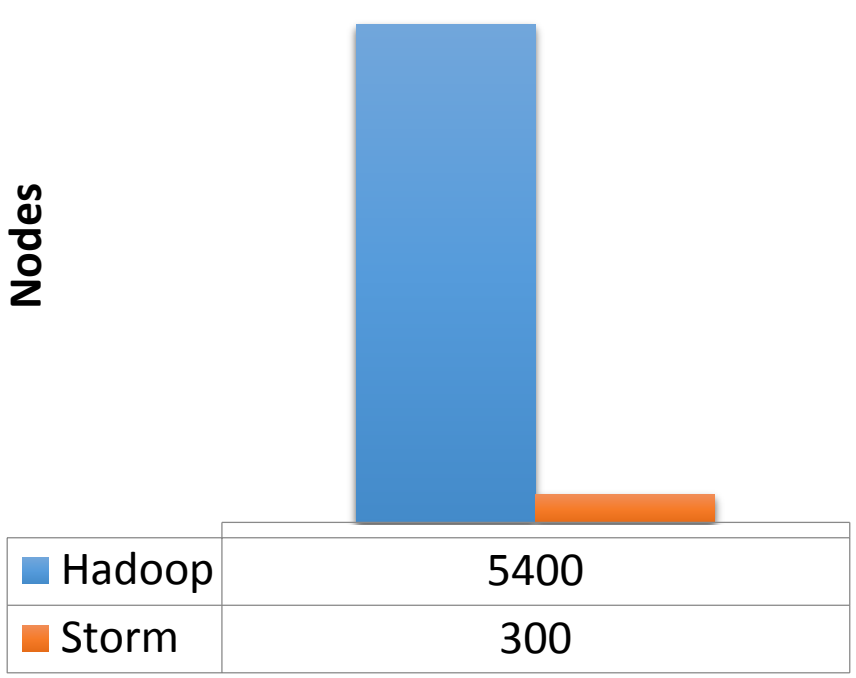
Provide a Hosted Platform for Yahoo

What We Do

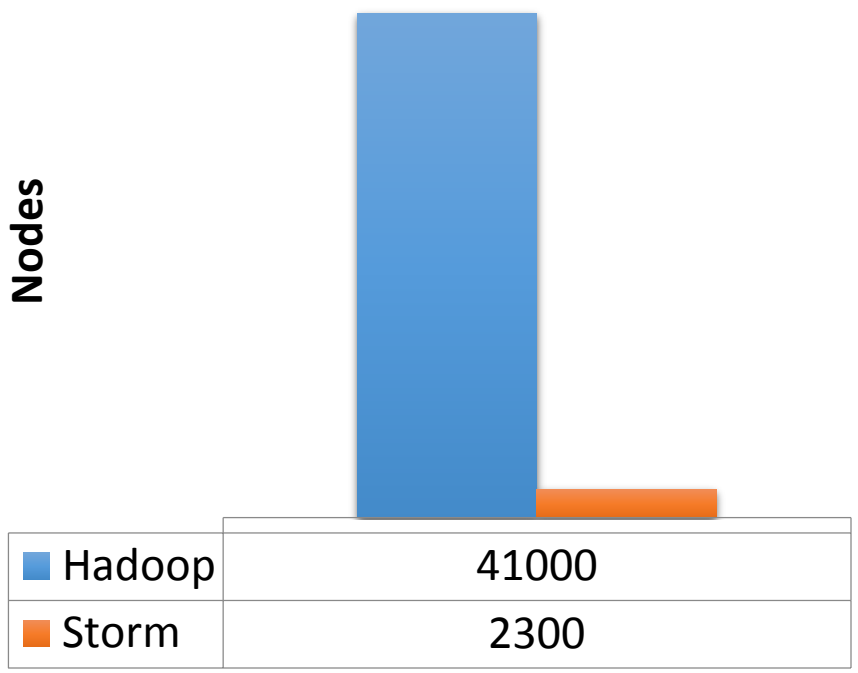
- Yahoo Scale
- Make it Secure
- Make it Easy

Yahoo Scale

Largest Cluster Size

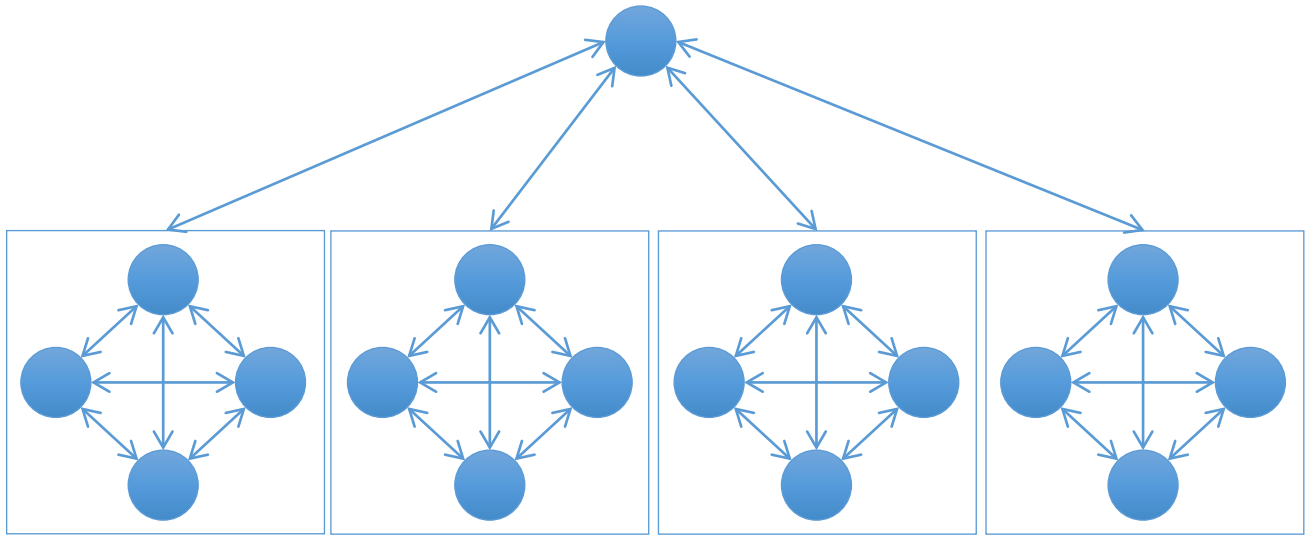


Total Nodes



Yahoo Scale (Solving Hard Problems)

Network Topology Aware Scheduling



https://en.wikipedia.org/wiki/Network_topology
https://en.wikipedia.org/wiki/Knapsack_problem

Understanding Software and Hardware

State Storage (ZooKeeper):

- Limited to disk write speed (80MB/sec typically)
- Scheduling
  - O(num_execs * resched_rate)
- Supervisor
  - O(num_supervisors * hb_rate)
- Topology Metrics (worst case)
  - O(num_execs * num_comps * num_streams * hb_rate)

On one 240-node Yahoo Storm cluster, ZK writes 16 MB/sec, about 99.2% of that is worker heartbeats

Theoretical Limit:
80 MB/sec / 16 MB/sec * 240 nodes = 1,200 nodes

Apply it to Work Around Bottlenecks

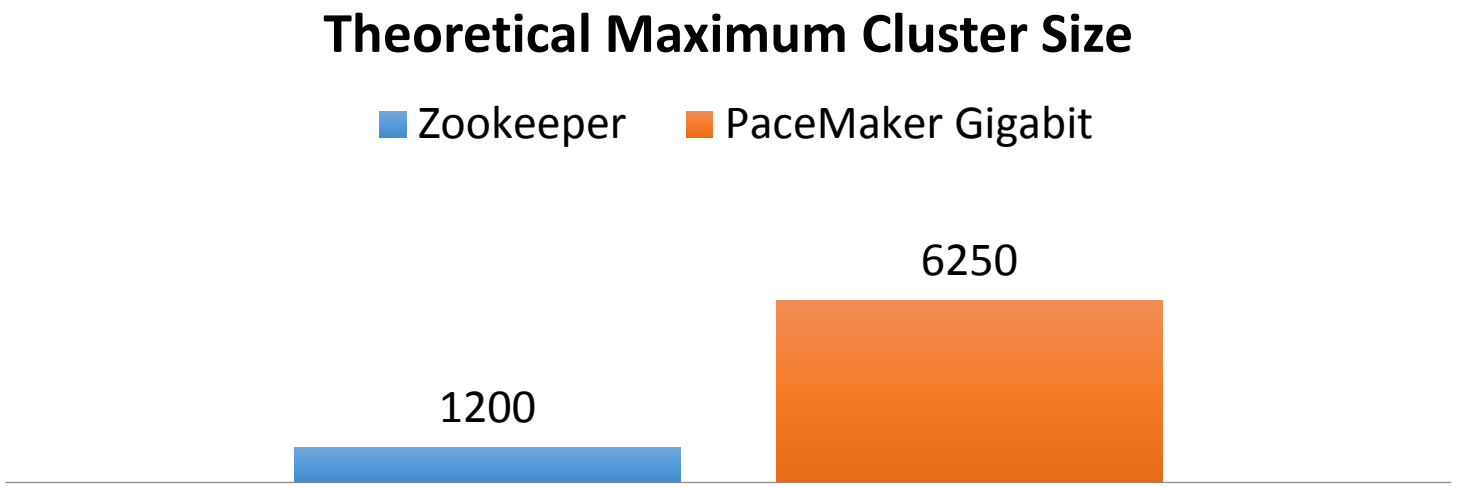
Fix: Secure In-Memory Store for Worker Heartbeats (PaceMaker)

- Removes Disk Limitation
- Writes Scale Linearly

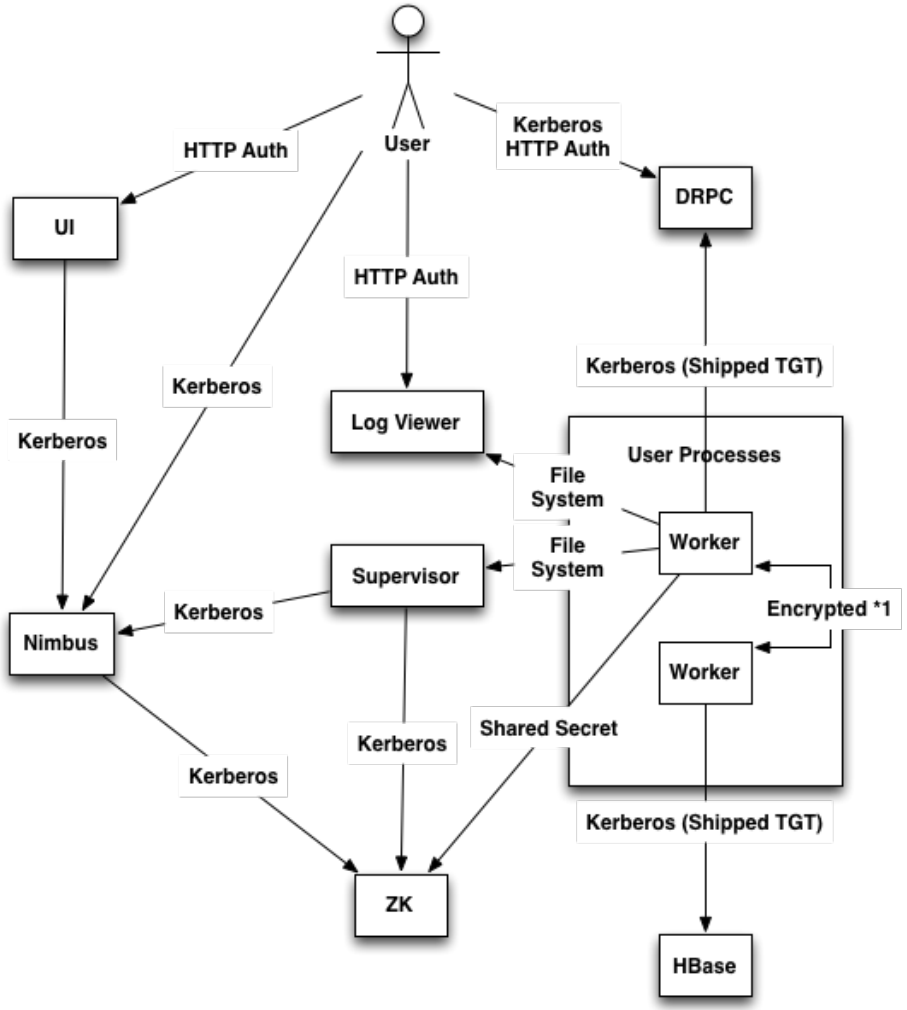
(but nimbus still needs to read it all, ideally in 10 sec or less)

240 node cluster's complete HB state is 48MB, Gigabit is about 125 MB/s

10 s / (48 MB / 125 MB/s) * 240 nodes = 6,250 nodes



Make it Secure



*1 Encrypted is not ideal, we still need to add SASL with a shared secret to netty transport

Make it Easy

- Simple API
- Easy to Debug
- Easy to Setup
- Easy to Upgrade (no downtime ideally)

Heavy lifting done by the platform

CLOUD COMPUTING APPLICATIONS

Anchoring and Spout Replay

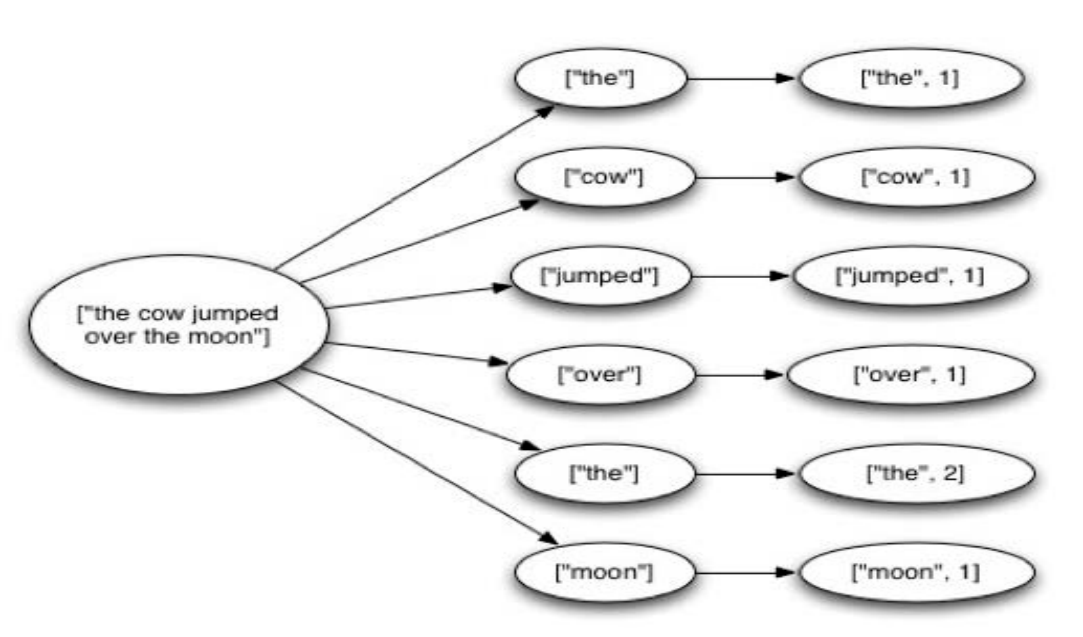
Roy Campbell & Reza Farivar

Guaranteeing Message Processing in Three Tasty Flavors

1. None (like the old S4)
2. *At Least Once: tuple trees, anchoring, and spout replay*
3. Exactly Once (like Hadoop or Puma)

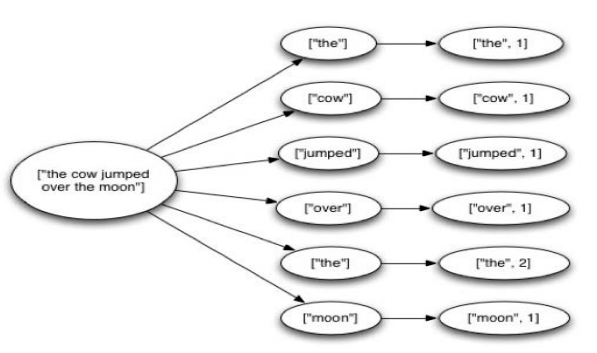
Tuple Tree

Tuple Tree



Tuple Tree

- A spout tuple is not fully processed until all tuples in the tree have been completed
- If the tuple tree is not completed within a specified timeout, the spout tuple is replayed
- Uses acker tasks to keep track of tuple progress



Anchoring

Reliability API for the user:

```
public void execute(Tuple tuple) {
    String sentence = tuple.getString(0);
    for(String word: sentence.split(" ")) {
        _collector.emit(tuple, new Values(word));
    }
    _collector.ack(tuple);
}
```

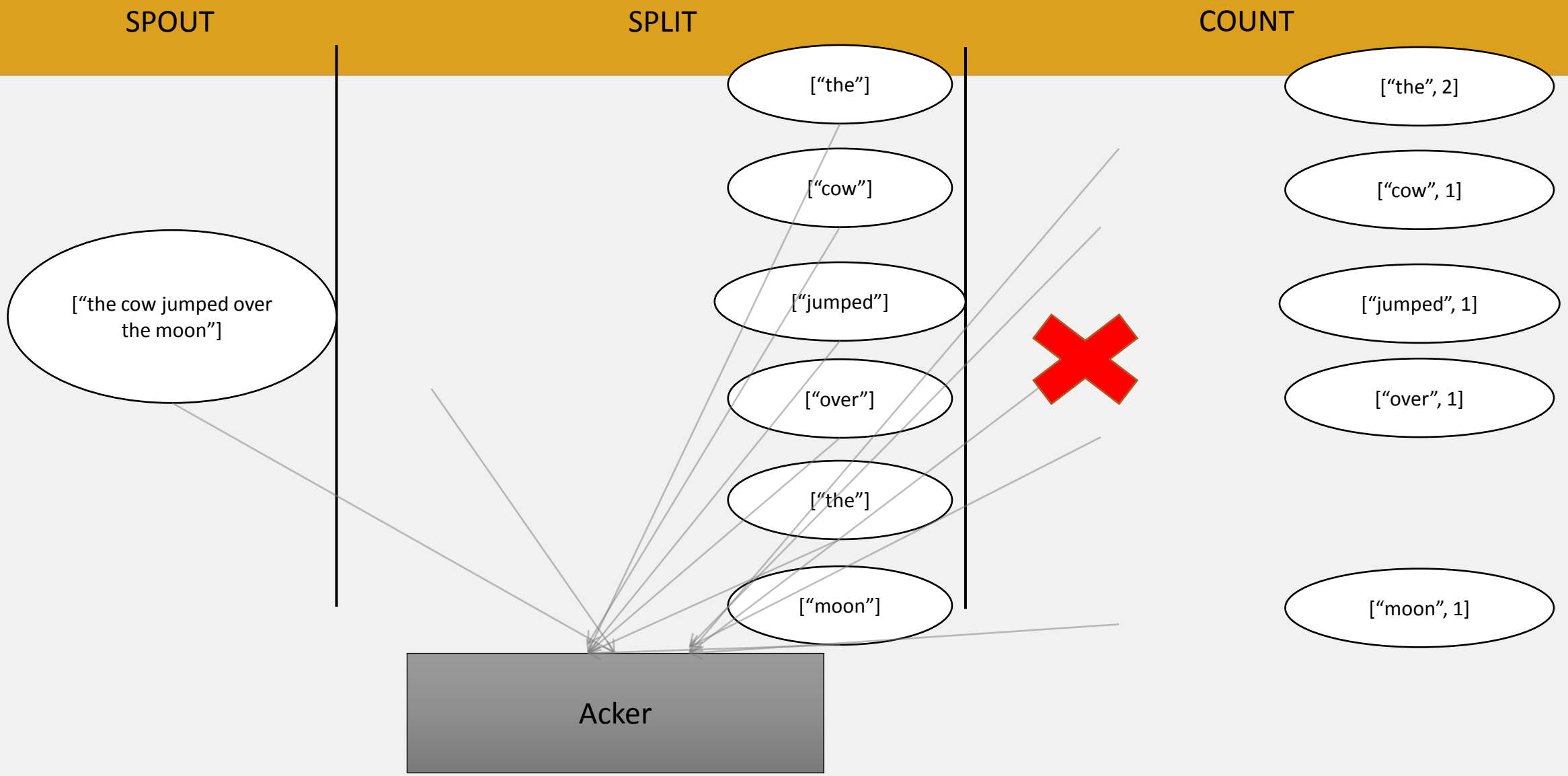
"Anchoring" creates a new edge in the tuple tree

Marks a single node in the tree as complete

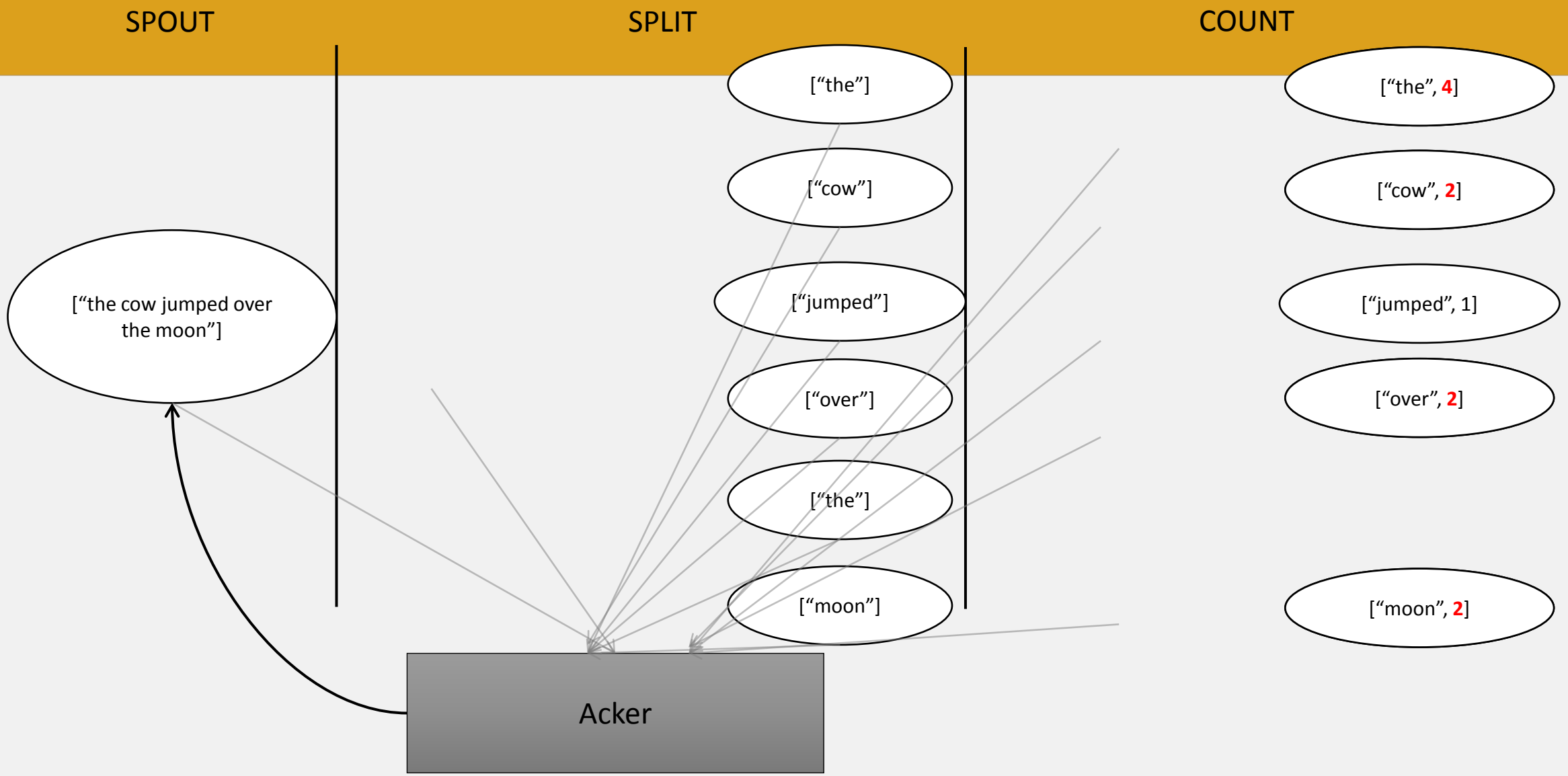
At Least Once

- What happens if there is a failure?
  - You can double process events
  - This is not so critical if you have something like Hadoop to back you up and correct the issue later
  - Or if you are looking at statistical trends and replay does not happen that often
- This requires you to have a spout that supports replay. Not all messaging infrastructure does

Example


SPOUT
SPLIT
COUNT
["the cow jumped over the moon"]
["the"]
["cow"]
["jumped"]
["over"]
["the"]
["moon"]
["the", 2]
["cow", 1]
["jumped", 1]
["over", 1]
["moon", 1]
Acker

Example


SPOUT
SPLIT
COUNT
["the cow jumped over the moon"]
["the"]
["cow"]
["jumped"]
["over"]
["the"]
["moon"]
["the", 4]
["cow", 2]
["jumped", 1]
["over", 2]
["moon", 2]
Acker

CLOUD COMPUTING APPLICATIONS

Trident: Exactly Once Processing

Roy Campbell & Reza Farivar

But What About State

- For most of storm, state storage is left up to you
- If your bolt goes down with 3 weeks of aggregated data that you have not stored anywhere, well too bad!

Enter Trident

- Provides exactly once semantics
- In trident, state is a first-class citizen, but the exact implementation of state is up to you
  - There are many prebuilt connectors to various NoSQL stores like HBase
- Provides a high level API (similar to cascading for Hadoop)

Trident Example

```
public class Split extends BaseFunction {

   public void execute(TridentTuple tuple, TridentCollector collector) {
      String sentence = tuple.getString(0);
      for(String word: sentence.split(" ")) {
         collector.emit(new Values(word));
      }
   }

}
```

No Acking Required

Trident Example

```
TridentTopology topology =
      new TridentTopology();
TridentState wordCounts =
    topology.newStream("spout1", spout)
      .each(new Fields("sentence"), new Split(),
new Fields("word"))
      .groupBy(new Fields("word"))
      .persistentAggregate(new
MemoryMapState.Factory(), new Count(), new
Fields("count"))
      .parallelismHint(6);
```

Aggregates values and stores them.

CLOUD COMPUTING APPLICATIONS

Inside Apache Storm

Roy Campbell & Reza Farivar

Inside Apache Storm

- Open Source, Git
- IntelliJ

DEMO

CLOUD COMPUTING APPLICATIONS

The Structure of a Storm Cluster

Roy Campbell & Reza Farivar

Structure

- Clojure
  - Clojure: functional programming language
  - Dialect of LISP
  - Runs on JVM
    - Complete Java interop
  - Fast!
- Java
  - Lots of code in Java, including the scheduler

CLOUD COMPUTING APPLICATIONS

Using Thrift in Storm

Roy Campbell & Reza Farivar

Thrift

- Storm.thrift
- Thrift compiler

CLOUD COMPUTING APPLICATIONS

How Storm Schedulers work

Roy Campbell & Reza Farivar

Scheduler

- IScheduler
- Multi-tenant scheduler

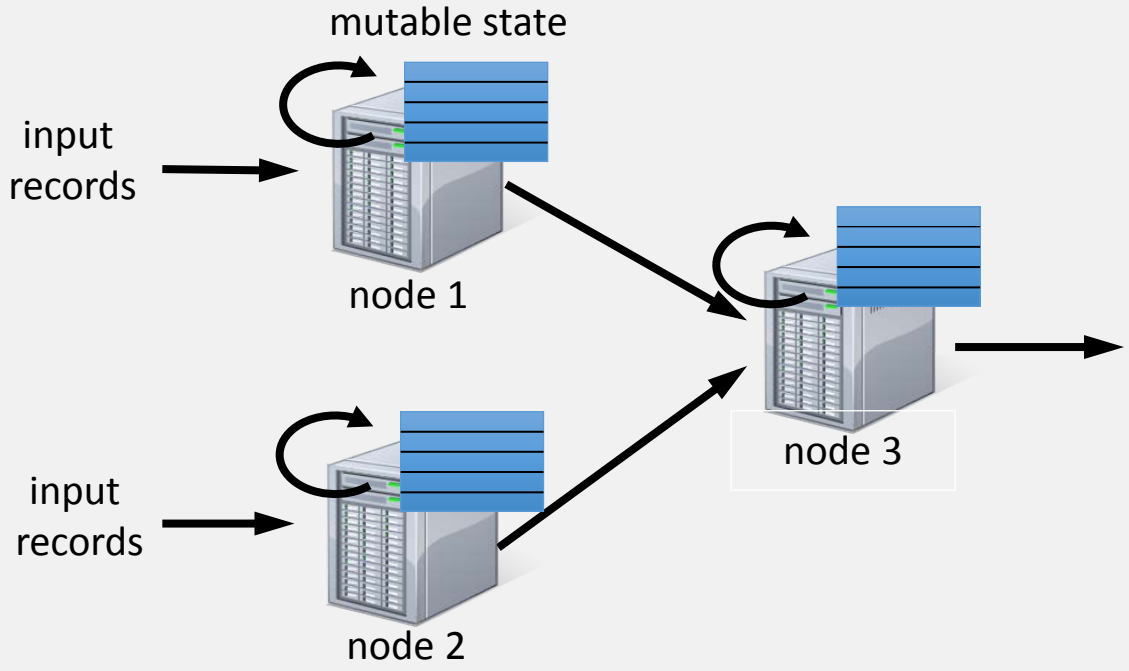
CLOUD COMPUTING APPLICATIONS

Spark Streaming

Roy Campbell & Reza Farivar

Stateful Stream Processing

- Traditional streaming systems have a record-at-a-time processing model
- Each node has mutable state
- For each record, update state and send new records
- State is lost if node dies!
  - Lambda Architecture
- Making stateful stream processing be fault-tolerant is challenging



Existing Streaming Systems

- Storm
  - Replays record if not processed by a node
  - Processes each record at least once
  - May update mutable state twice!
  - Mutable state can be lost due to failure!

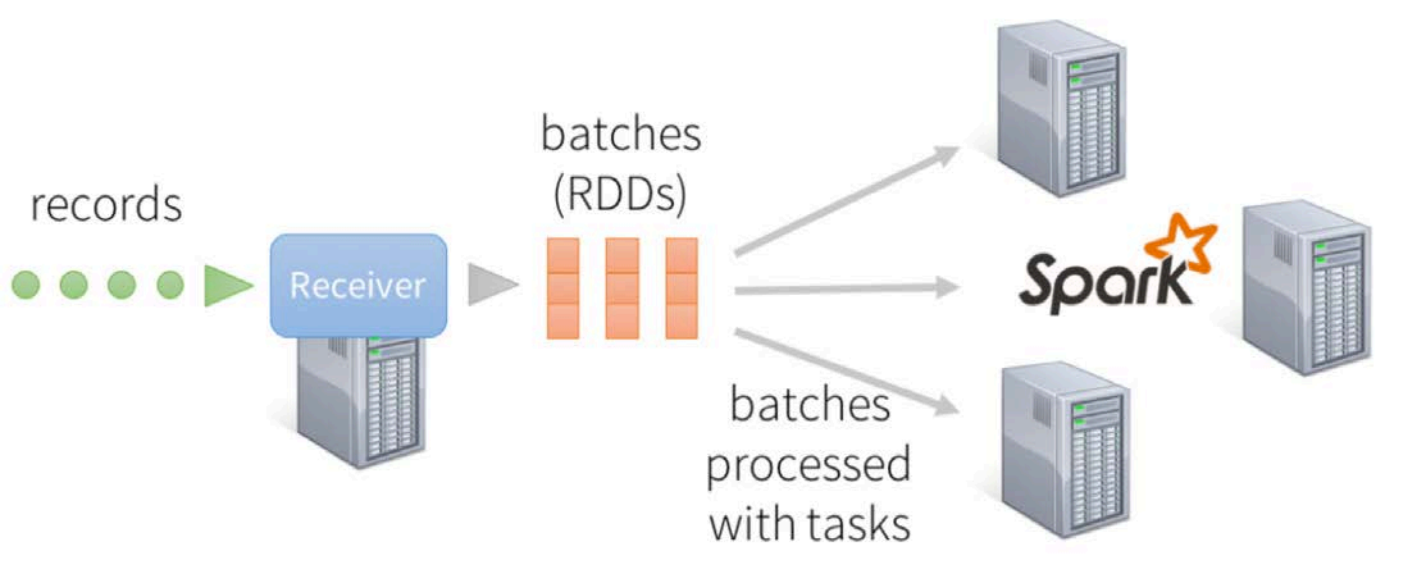
- Trident – Use transactions to update state
  - Processes each record exactly once
  - Per state transaction to external database is slow

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

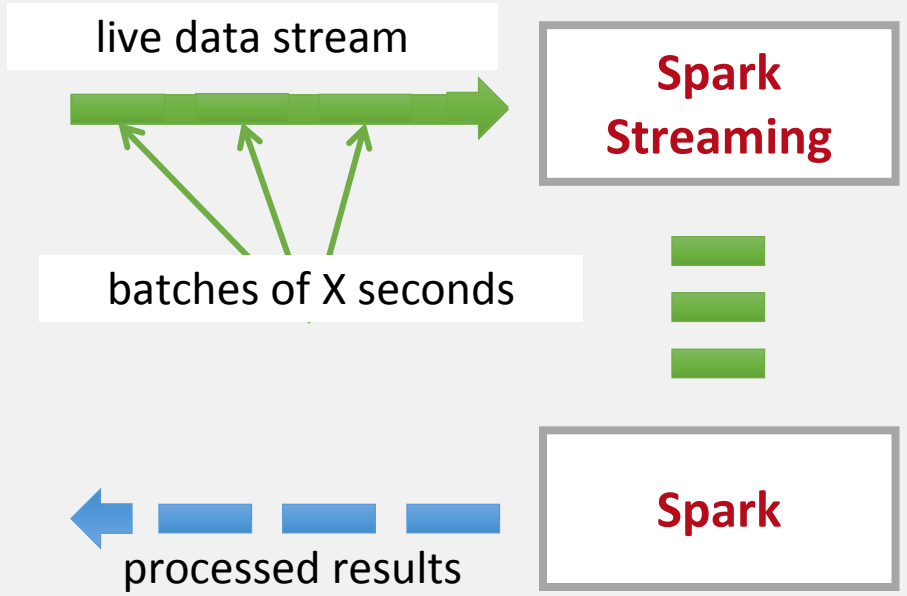
Spark Streaming

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



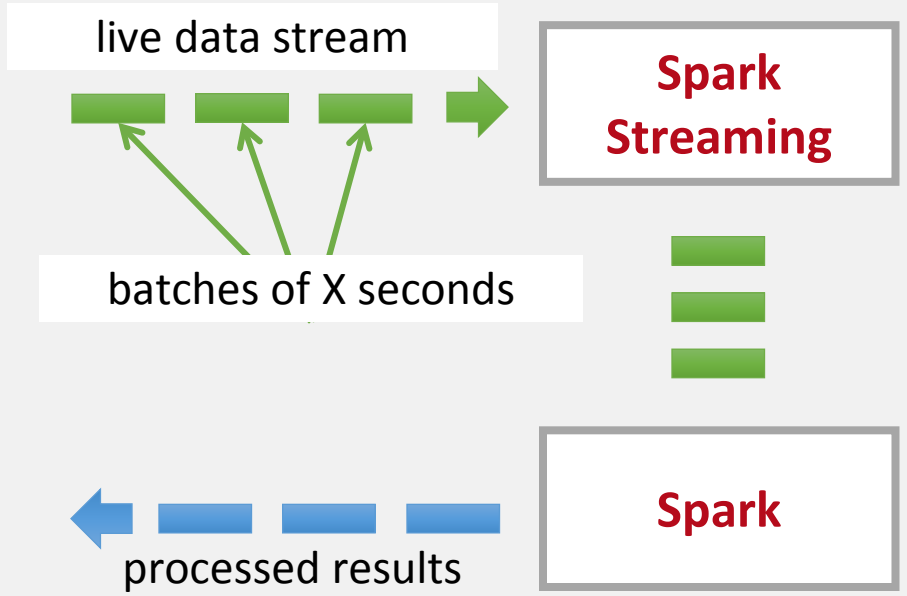
Discretized Stream Processing

- Chop up the live stream into batches of X seconds
- Spark treats each batch of data as RDDs and processes them using RDD operations
- Finally, the processed results of the RDD operations are returned in batches



Discretized Stream Processing

- Batch sizes as low as ½ second, latency of about 1 second
- Potential for combining batch processing and streaming processing in the same system



Spark Streaming example

```
sc = SparkContext("local[2]", "NetworkWordCount")
ssc = StreamingContext(sc, 1)
lines = ssc.socketTextStream("localhost", 9999)
# Split each line into words
words = l
# Count each word in each batch
pairs = words.map(lambda word: (word, 1))
wordCounts = pairs.reduceByKey(lambda x, y: x + y)
# Print the first ten elements of each RDD generated in this DStream to the console
wordCounts.pprint()ines.flatMap(lambda line: line.split(" "))
```

DStream Input Sources

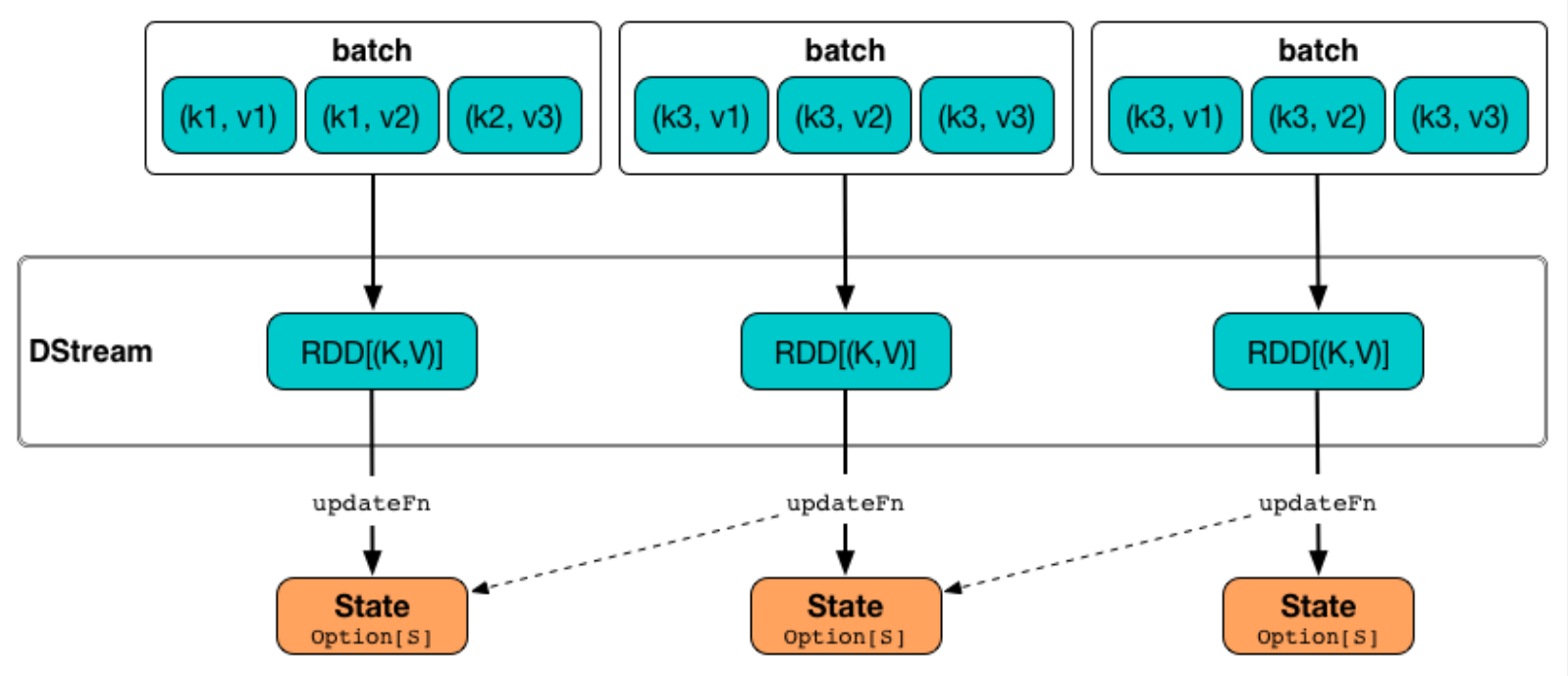
- Out of the box
  - Kafka
  - HDFS
  - Flume
  - Akka Actors
  - Raw TCP sockets

- Very easy to write a *receiver* for your own data source

Arbitrary Stateful Computations

- updateStateByKey
  - maintain arbitrary state while continuously updating it with new information
- How to use
  - Define the state - The state can be an arbitrary data type
  - Define the state update function - Specify with a function how to update the state using the previous state and the new values from an input stream
- state update function applied in every batch for all existing keys

Arbitrary Stateful Computations



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

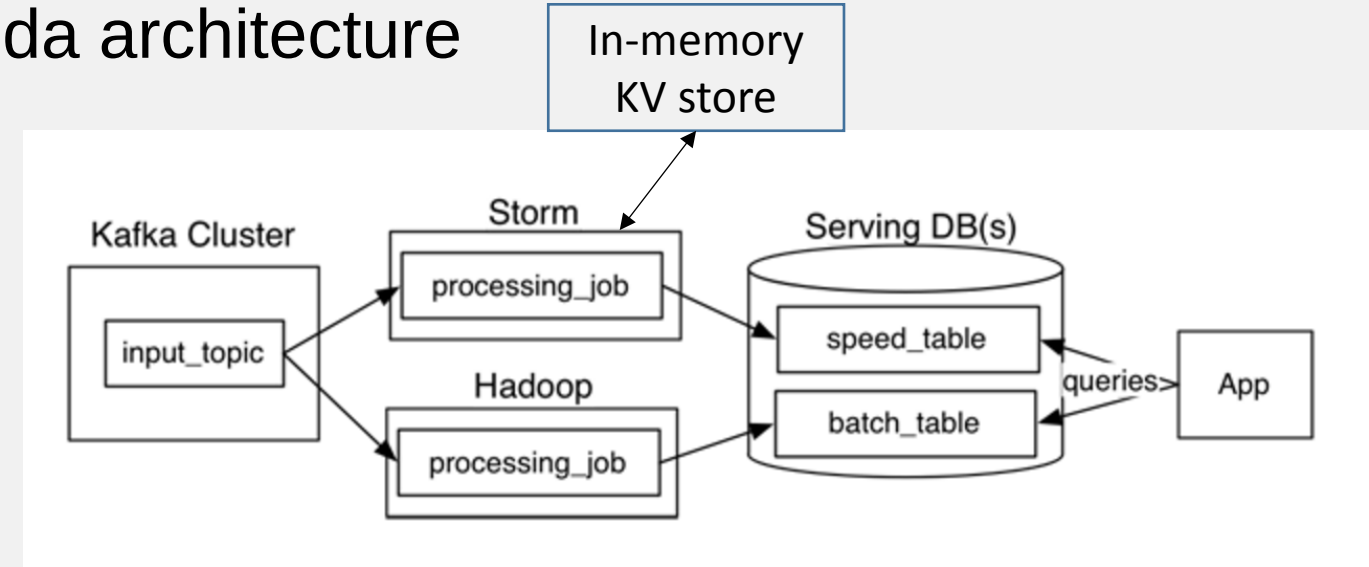
CLOUD COMPUTING APPLICATIONS

Lambda and Kappa Architecture

Roy Campbell & Reza Farivar

Lambda Architecture

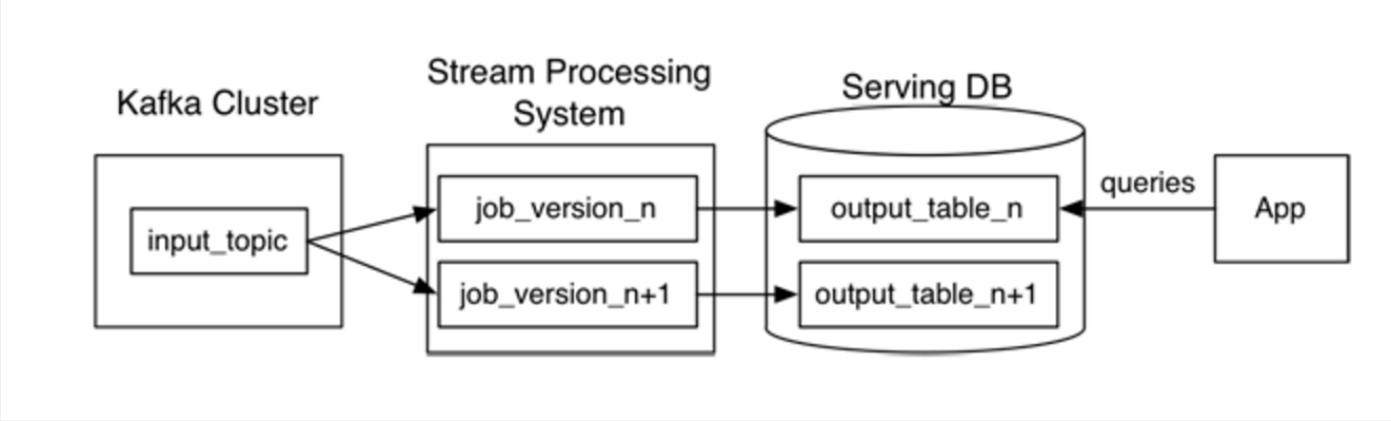
- Lambda architecture



- Why Lambda? Because things fail
  - Batch handles failures just fine
- A true streaming system has to guarantee idempotency

Kappa Architecture

- Kappa Architecture



- Only the streaming path
- But what about the state?
  - Perhaps Microbatch can help

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

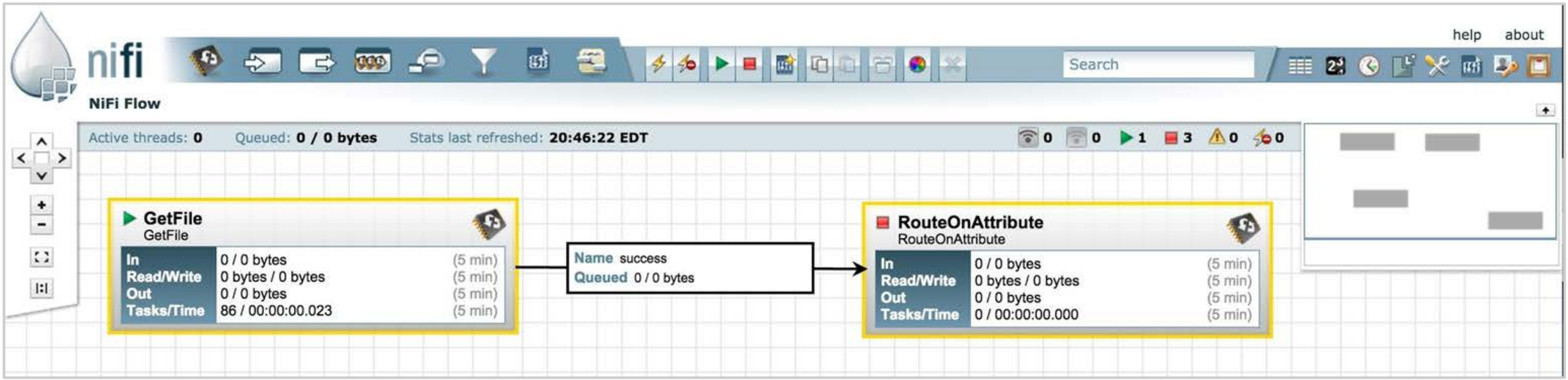
- Apache NiFi is a good distributed funnel
- Was made in NSA
  - Over 8 years of development
- Open sourced in 2014 and picked up by HortonWorks
- Great visual UI to design a data flow
- Has many many processor types in the box
- But not very good for heavy weight distributed processing
  - Same graph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

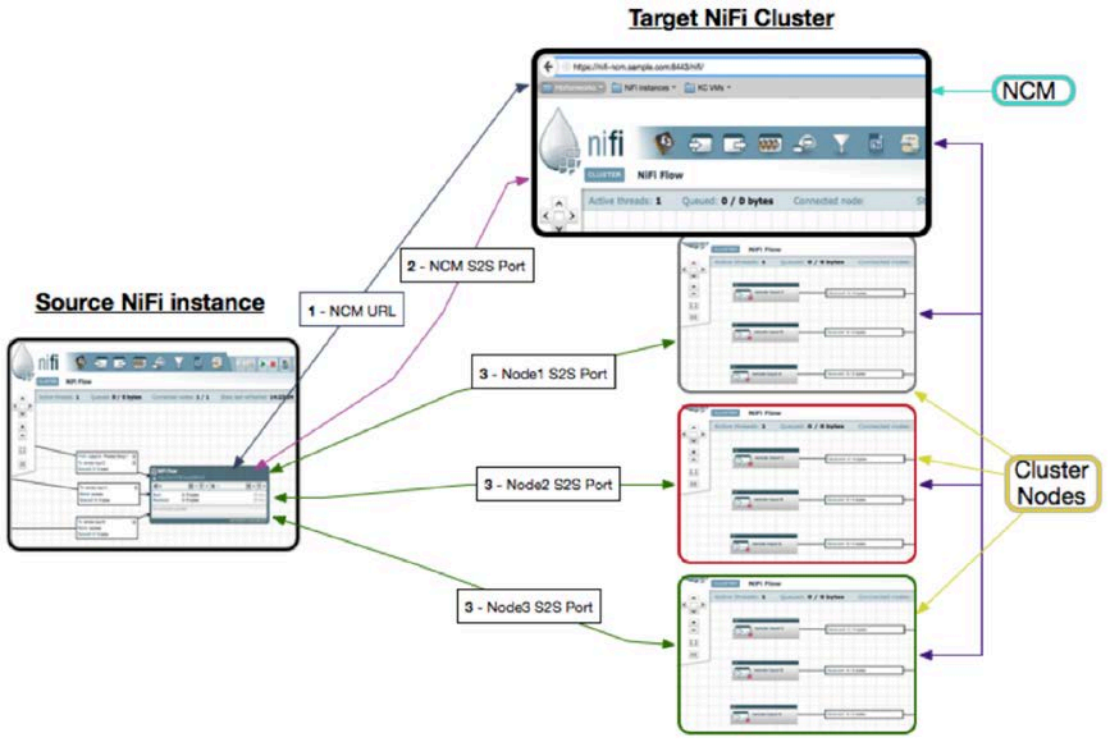
NiFi GUI

- 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



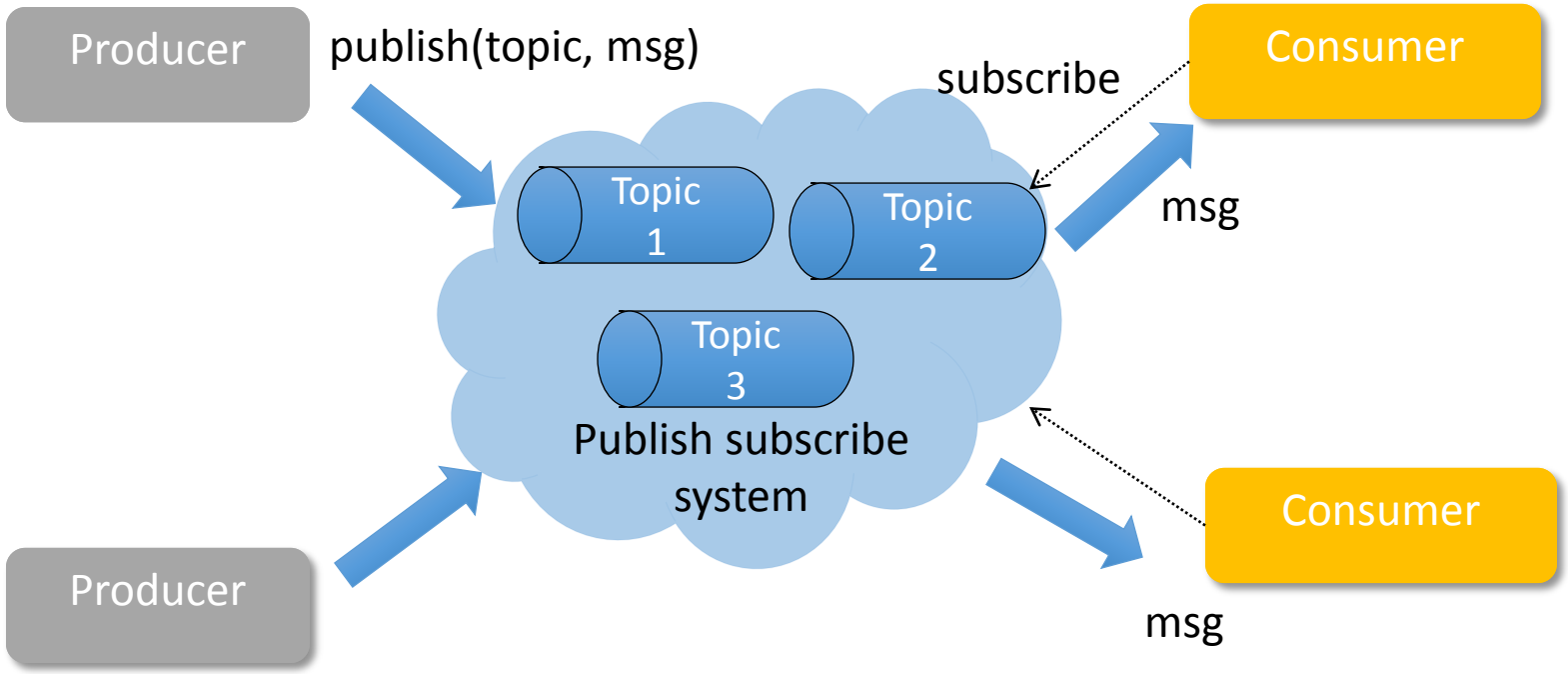
NiFi Site-to-Site

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



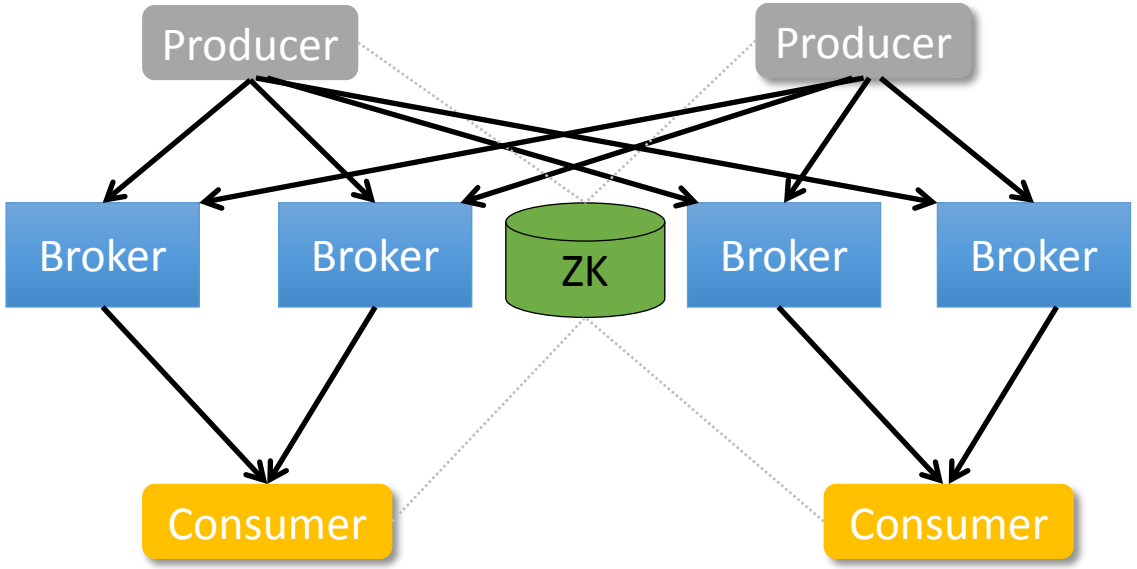
Step 2: Distributed Queue

- 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

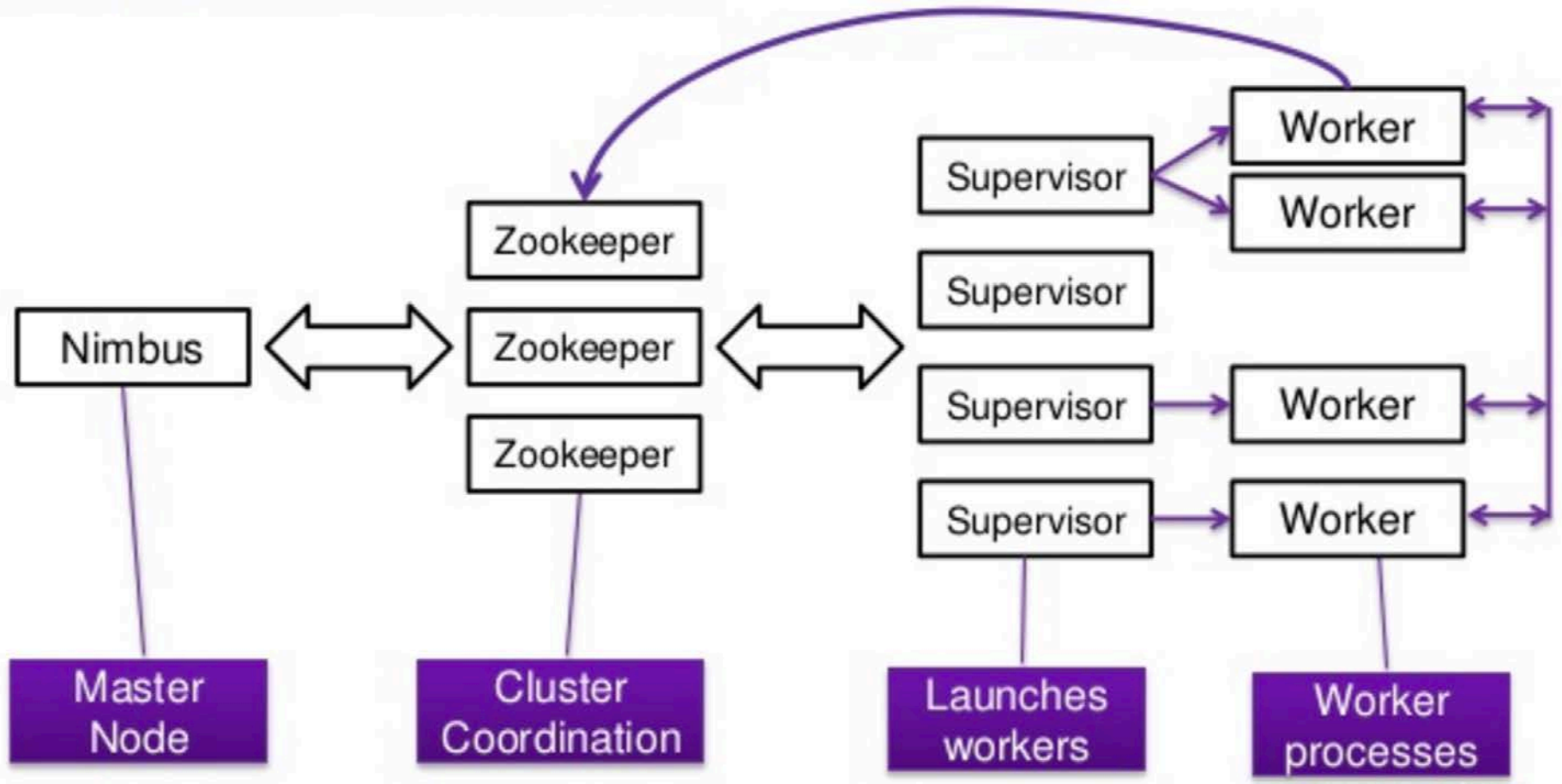
Step 3: Distributed Processing

- 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

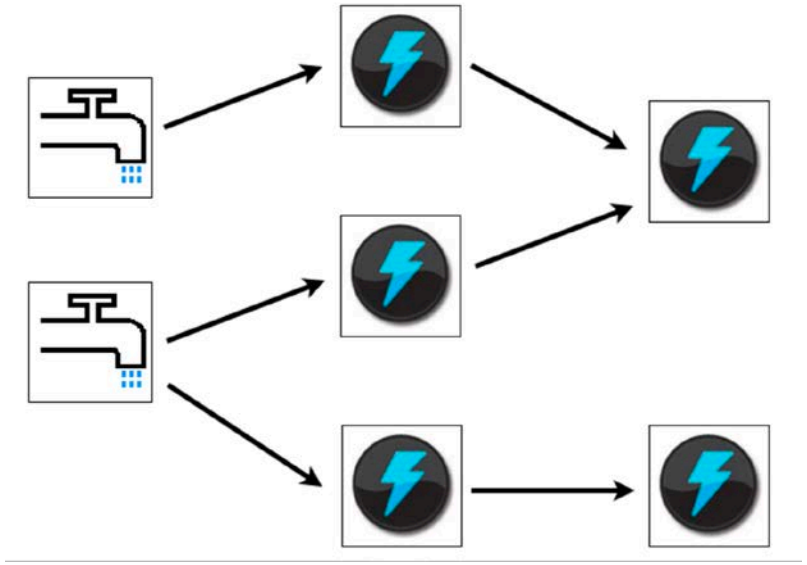
- Apache Storm
- Built in backtype, sold to Twitter
- Written in Clojure

Storm Architecture



Storm programming

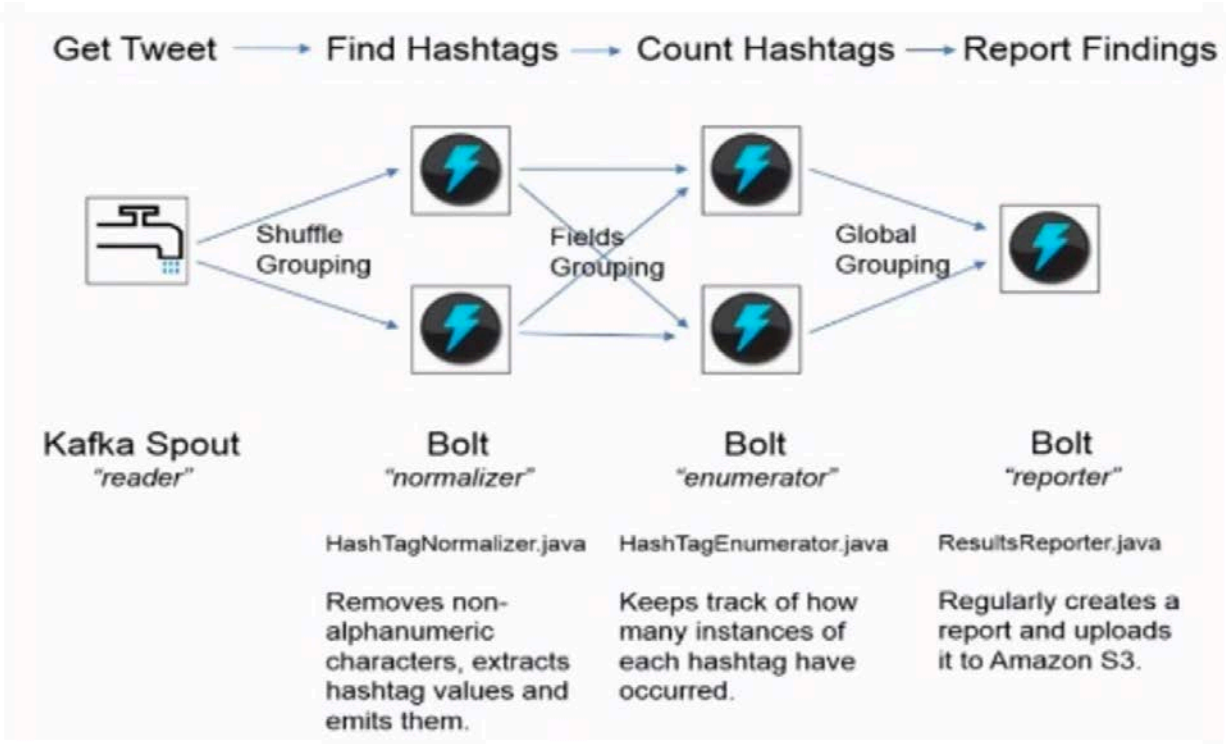
- Topology
  - Spouts
  - Bolts
  - Tuples
  - Streams
- topologyBuilder API



```
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

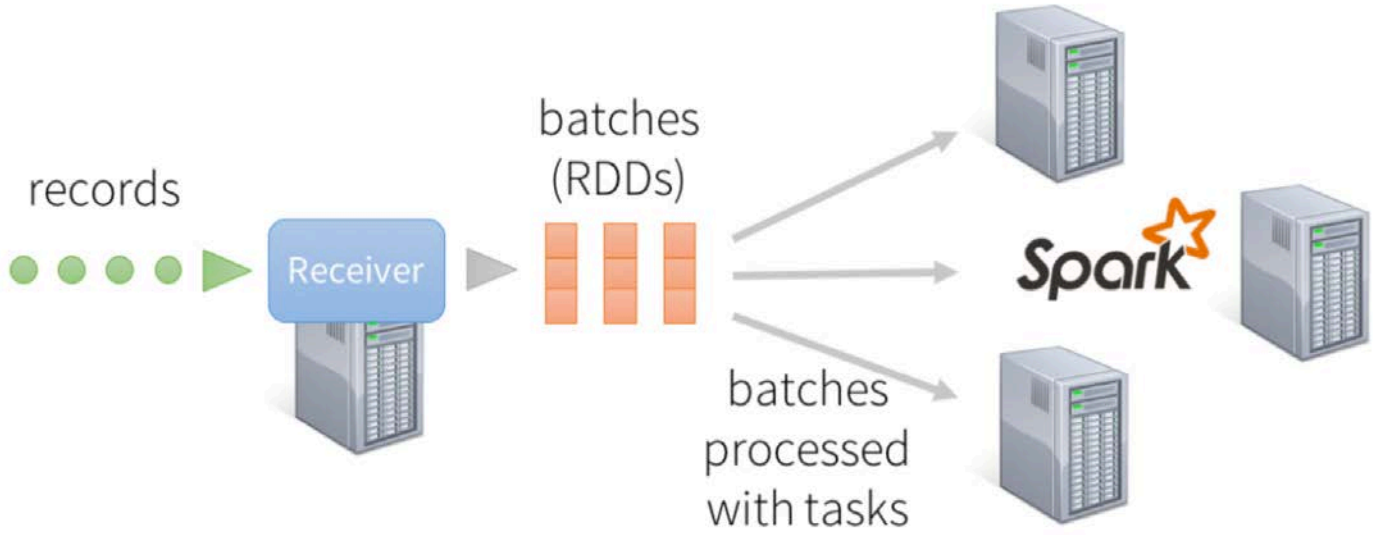
- Instead of real-time event-by event processing, we can do micro batch
- Reduce overheads
- Fault tolerance → Kappa architecture
- High latency

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

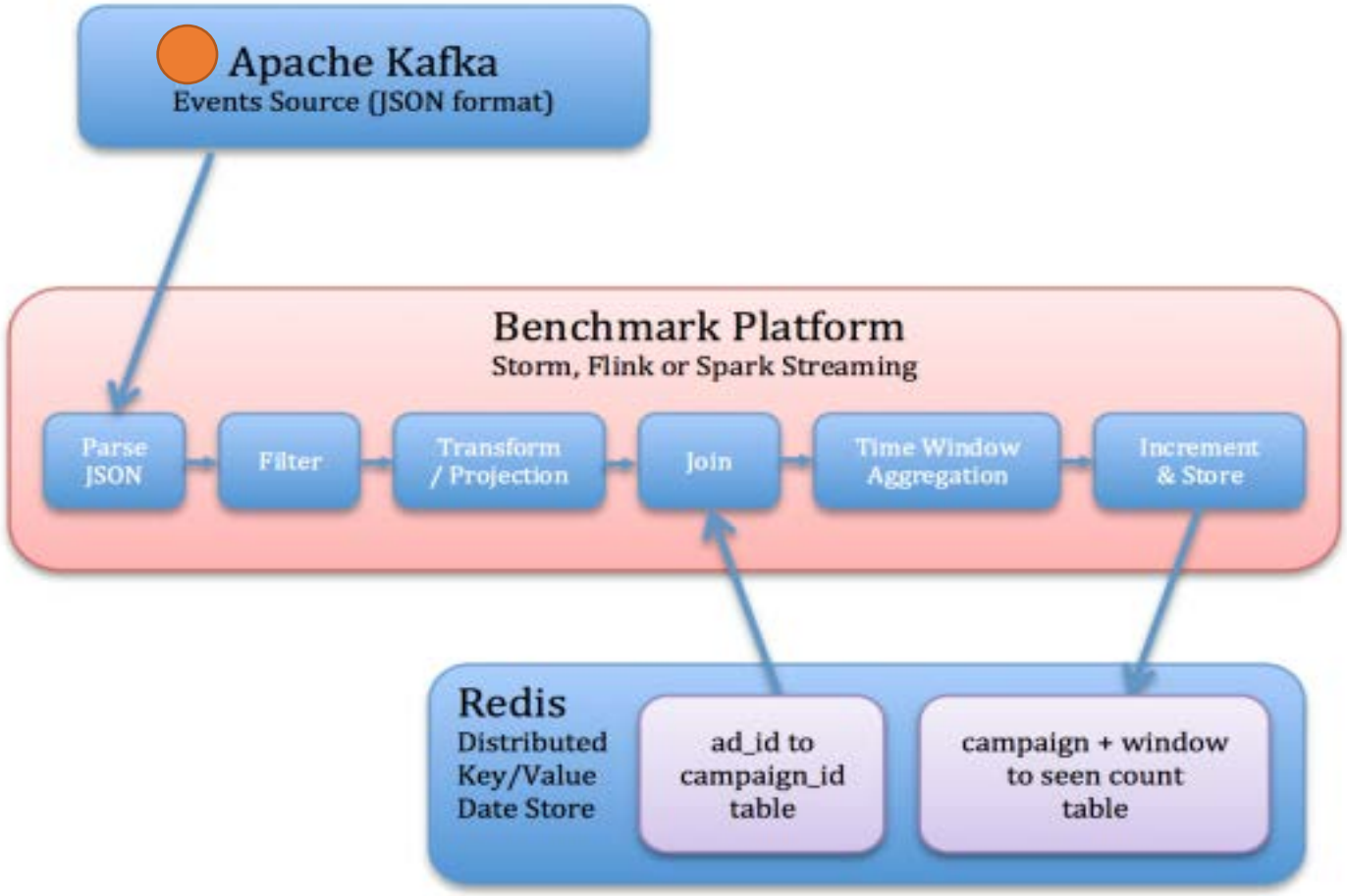
- Window a bit of data
- Run a batch
- Repeat



Spark ML, Graph, etc.

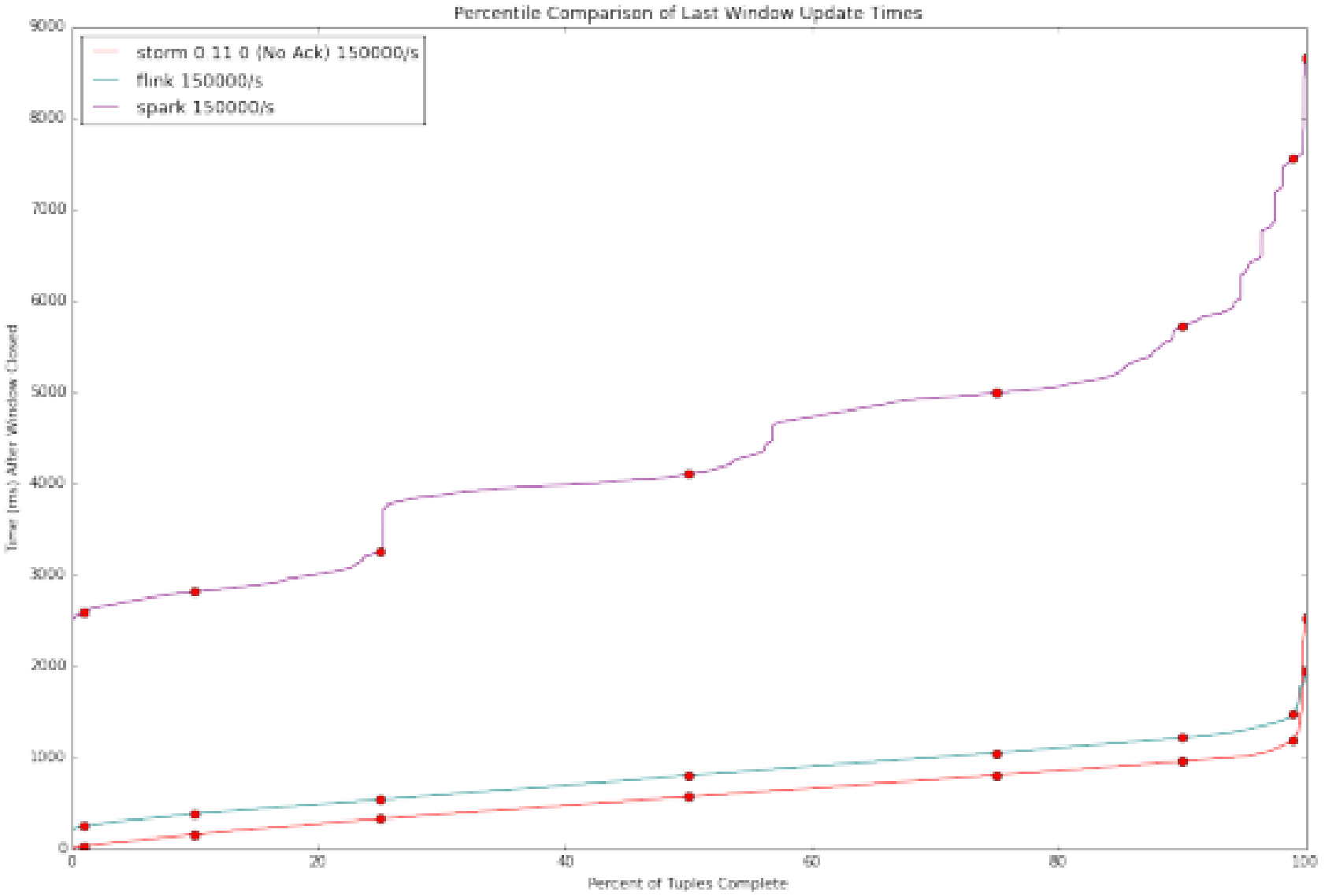
- Advantage of Spark Streaming:
  - Rich ecosystem of big data tools
  - Spark SQL
  - Spark ML
  - Spark GraphX
  - SparkR
- Disadvantage:
  - Not really streaming

Benchmark: ETL pipeline



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)

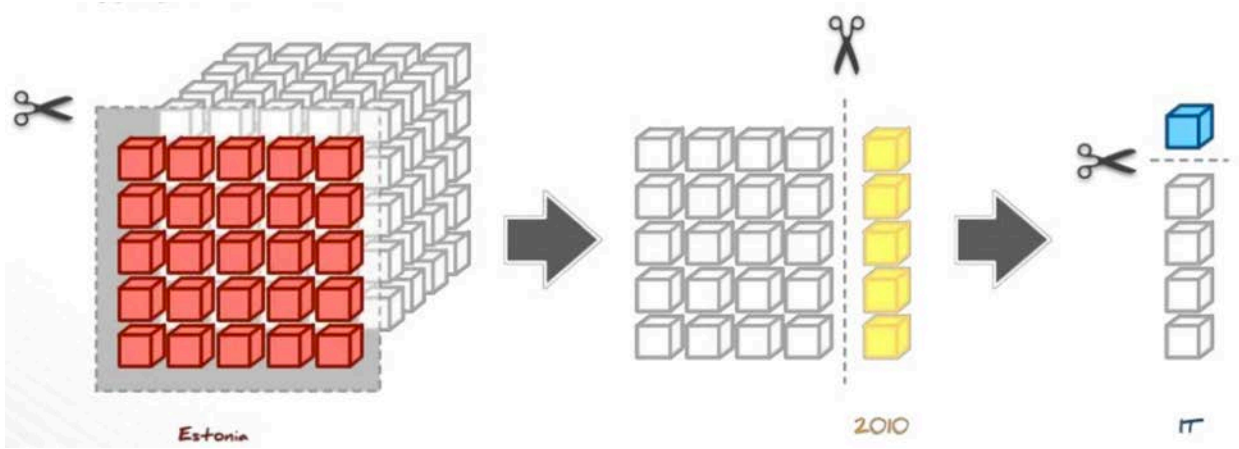
- Business Intelligence
- Multidimensional data analytics
- Analyze multidimensional data interactively
- Basic Operations
  - Consolidation (roll-up, aggregation in dimensions)
  - Drill-down (filter)
  - Slicing and dicing (Look at the data from different viewpoints)

Druid

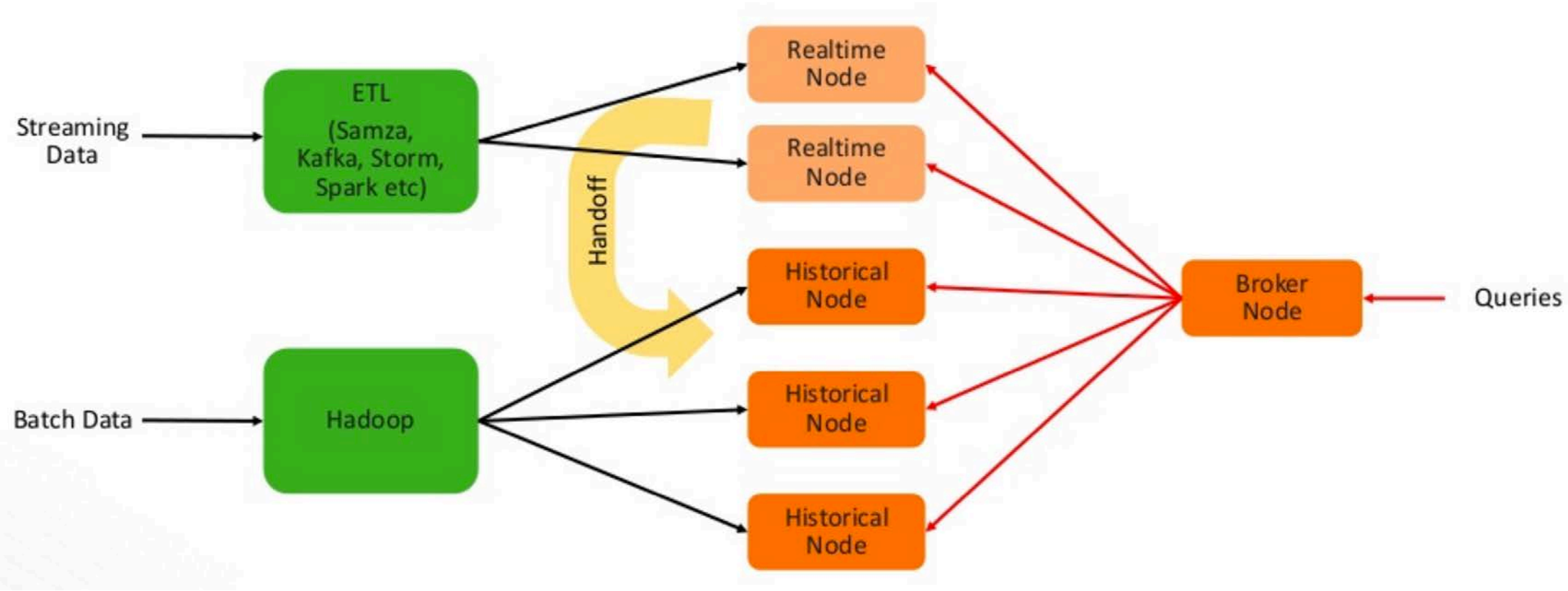
- Developed in Metamarkets in 2011
  - RDBMs: Too slow
  - NoSQL key value store: fast, but exponential memory space, precompute very slow
- Gaining in popularity
- Open Source (Apache license) in late 2012
- OLAP queries
- Column oriented
- Sub second query time (Avg query time 0.5 seconds)
- Real-time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

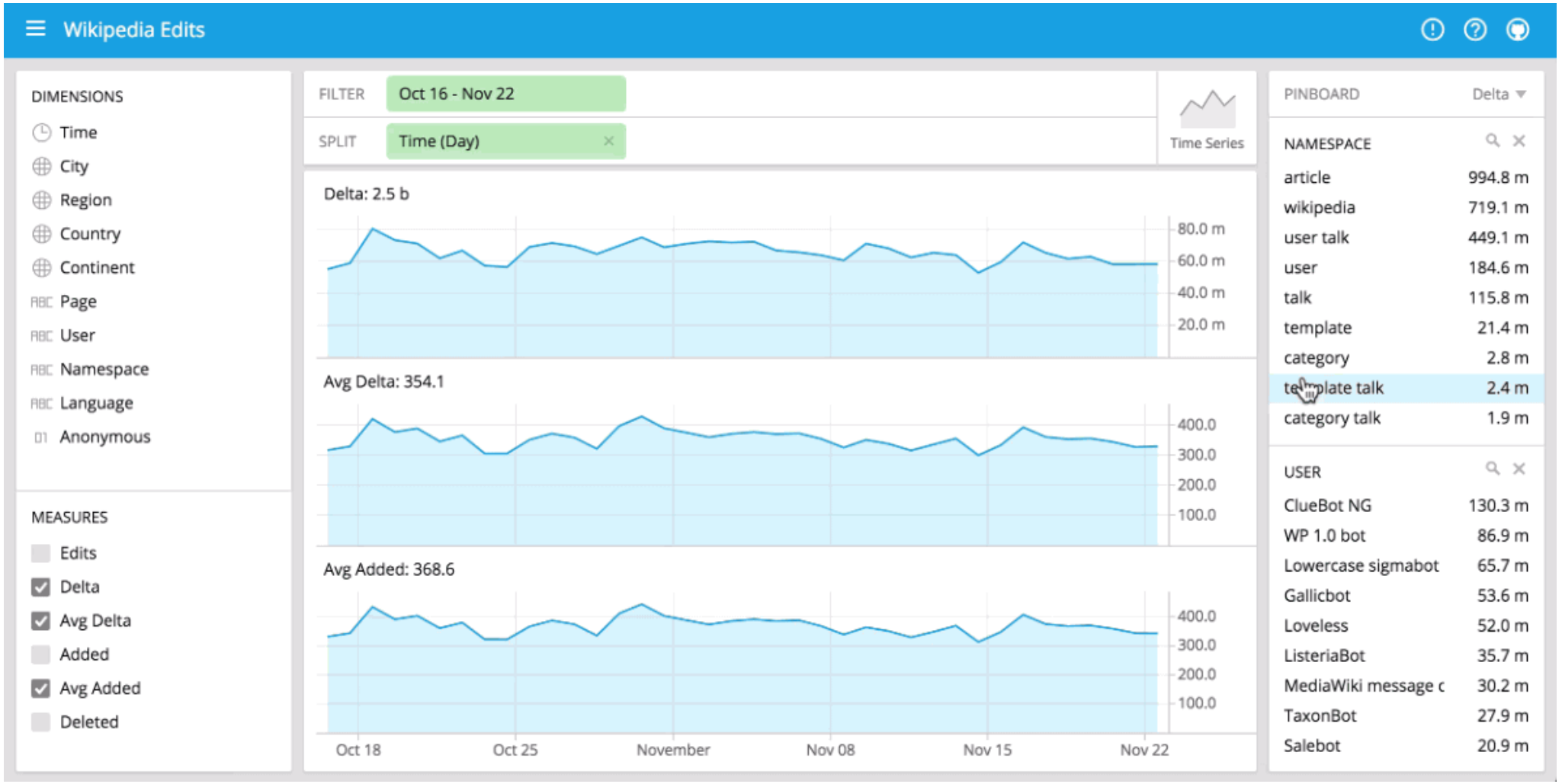
timestamp	page	language	city	country	...	added	deleted
2011-01-01T00:01:35Z	Justin Bieber	en	SF	USA		10	65
2011-01-01T00:03:63Z	Justin Bieber	en	SF	USA		15	62
2011-01-01T00:04:51Z	Justin Bieber	en	SF	USA		32	45
2011-01-01T01:00:00Z	Ke$ha	en	Calgary	CA		17	87
2011-01-01T02:00:00Z	Ke$ha	en	Calgary	CA	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


Wikipedia Edits
DIMENSIONS
Time
City
Region
Country
Continent
Page
User
Namespace
Language
Anonymous
MEASURES
Edits
Delta
Avg Delta
Added
Avg Added
Deleted
FILTER
Oct 16 - Nov 22
SPLIT
Time (Day)
Time Series
Delta: 2.5 b
80.0 m
60.0 m
40.0 m
20.0 m
Avg Delta: 354.1
400.0
300.0
200.0
100.0
Avg Added: 368.6
400.0
300.0
200.0
100.0
Oct 18
Oct 25
November
Nov 08
Nov 15
Nov 22
PINBOARD
Delta
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

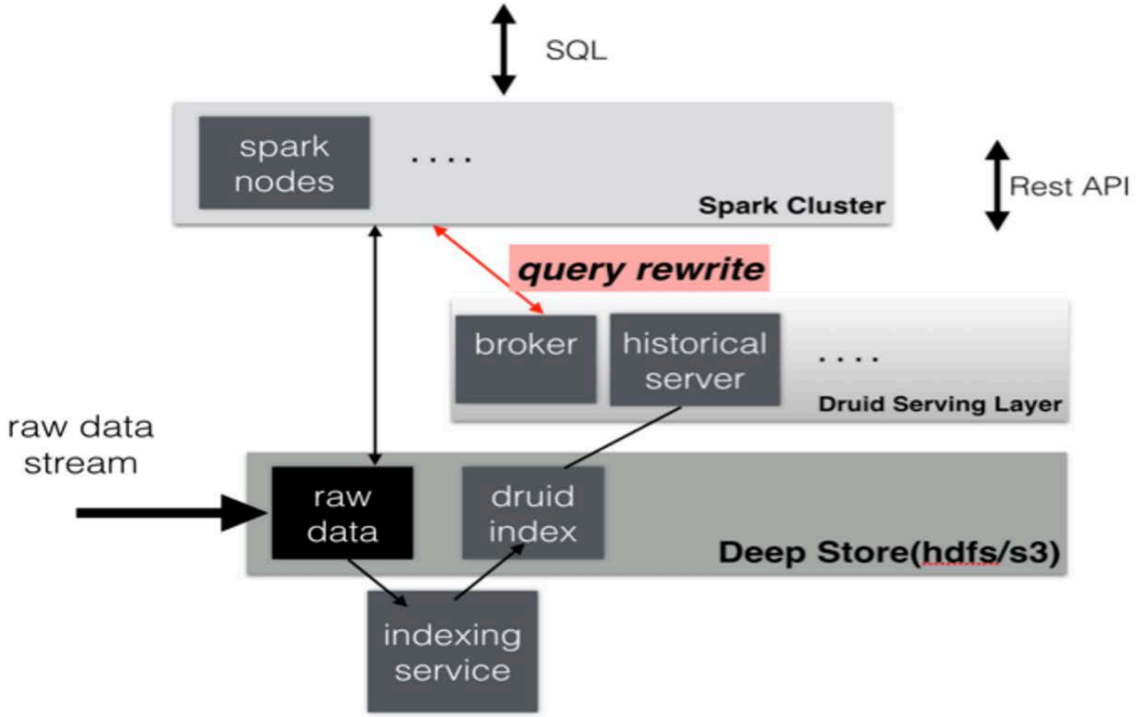
Druid and Spark

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

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

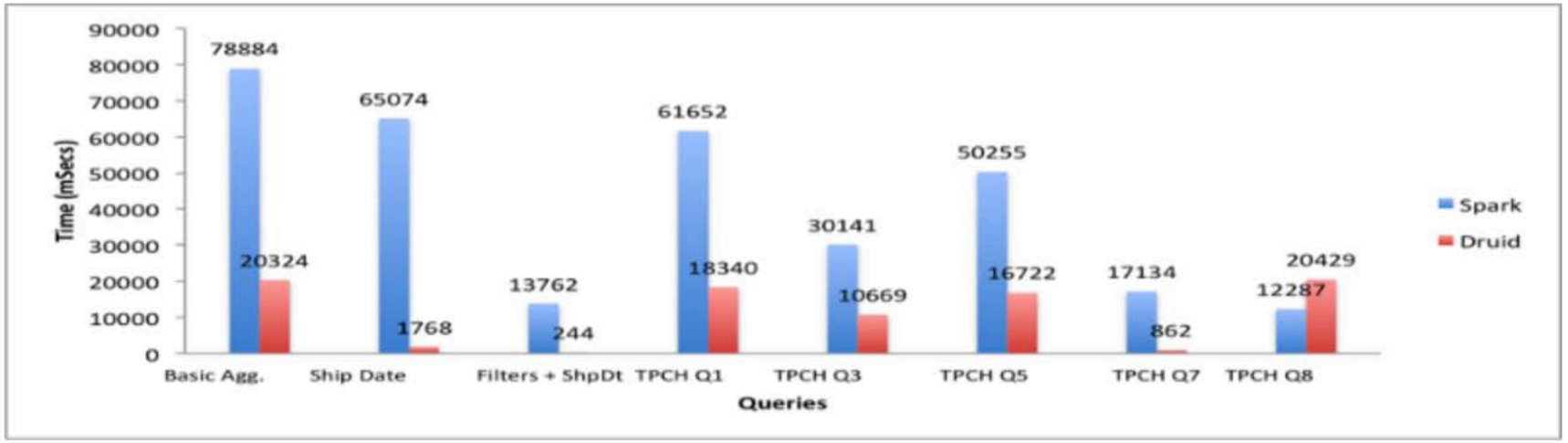
- Connect Druid to Tableu through Spark

```
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