NoLambda: Combining Streaming, Ad-Hoc, Machine Learning, and Batch Analytics

Evan Chan

April 2016

Evan Chan

- Distinguished Engineer, Tuplejump
- @evanfchan | evan@tuplejump.com
- `http://velvia.github.io`
- User and contributor to Spark since 0.8, Cassandra since 0.6
- Co-creator and maintainer of Spark Job Server
- Speaker: Strata, Spark Summit, Cassandra Summit

Tuplejump

Tuplejump is a big data technology leader providing solutions and development partnership.

Tuplejump Open Source: on GitHub

- FiloDB - Distributed Spark + Cassandra analytics database
- Kafka Connect Cassandra - Kafka-Cassandra Source and Sink
- Calliope - The first Spark Cassandra integration
- Stargate - Lucene indexer for Cassandra
- SnackFS - HDFS for Cassandra

Rescuing struggling big data projects with innovative, effective solutions

- Experience designing, developing, productionizing big data and real time projects
- Intimate source-code knowledge of leading edge big data solutions
- Team leadership, embedding within a team
- Architecture, data modeling
- Deep relationships with open source big data communities

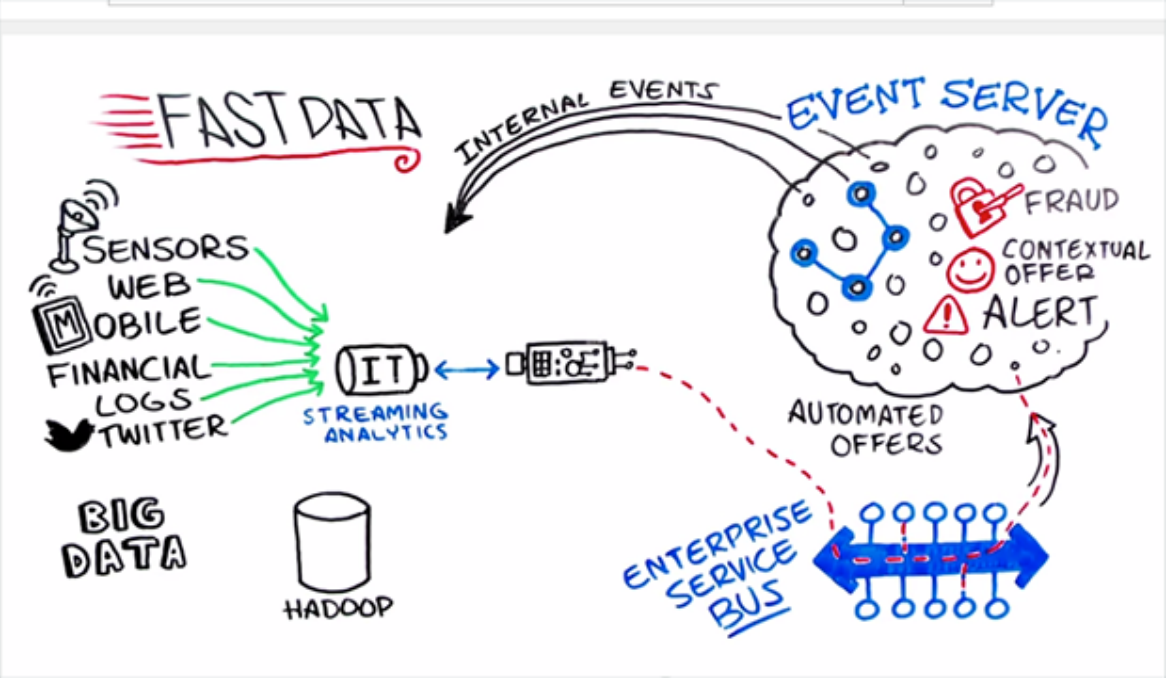
Topics

- Big data + Fast Data: We need to handle the deluge!
- From LAMP to Batch to Lambda Architecture
- Simplifying the stack with the NoLambda / SNACK stack
- Detailed look at stream processing pipelines
- FiloDB and fast analytics with competitive storage cost
- Machine learning with Spark, Cassandra, and FiloDB

Big data is yesterday.

FAST DATA

is now.


FAST DATA
INTERNAL EVENTS
EVENT SERVER
SENSORS
WEB
MOBILE
FINANCIAL
LOGS
TWITTER
IT
STREAMING ANALYTICS
FRAUD
CONTEXTUAL OFFER
ALERT
AUTOMATED OFFERS
BIG DATA
HADOOP
ENTERPRISE SERVICE BUS

Fast Data + Big Data

- **1-30 seconds**: Reactive processing of streaming data as it comes in to derive instant insights.
- **Minutes to Days/Months**: Combine with recent or historical data for deeper insights, trends, ML.
- Not enough just to have stream processing or batch processing.

The Problem Domain

Build scalable, adaptable, self-healing, distributed data processing systems for

- 24 / 7 Uptime
- Auto scale out
- Complex analytics and learning tasks
- Aggregate global data
- Ops for global, multi-dc clustered data flows
- Decoupled services
- Idempotent & Acceptable Consistency
- No data loss

Factors & Constraints in Architecture

- What's the simplest architecture I need for now?
- Scalability for future workloads
- Maintenance and operations - KISS (Keep it Simple)
- Cost
- Interaction with legacy data or legacy architecture
- What SLAs can be met?
- How does this architecture fit into my existing team? Who needs to be hired?

The list goes on...

Need Self-Healing Systems

- Massive event spikes & bursty traffic
- Fast producers / slow consumers
- Network partitioning & out of sync systems
- DC down
- Not DDOS'ing ourselves from fast streams
- No data loss when auto-scaling down

Monitor Everything

Everything fails, all the time

Use Case

I need fast access to historical data on the fly for predictive modeling with real time data from the stream

Only, It's Not A Stream It's A Flood

- Trillions of event writes per day
- Billions of event reads per day
- Massive events per second at peak
- Petabytes of total streaming data

Not All Streams Are Created The Same

Daily, Hourly, Frequency, Event Spikes at Peak, Overall Volume

- Sub-second, low latency stream processing
- Higher latency stream processing
- Scheduled or on request batch processing

Real Time

- Just means **Event Driven** or processing events as they arrive
- Doesn't automatically equal sub-second latency requirements

Event Time

- When an event is created, e.g. on sensor
- Events should be *uniquely* timestamped on ingestion for tracking, metrics and replay

Based on the schema of data in a given stream

Some can aggregate with sliding windows (T1...Tn,Tn+1...) using window length + slide interval:

```
stream.reduceByKeyAndWindow((a:Int,b:Int) => (a + b), Seconds(30), Seconds(10)
```

Some must aggregate by buckets

```
/Event_Type/YYYY/MM/DD/HH/MM/...
```

```
CREATE TABLE timeseries.raw_data_fu (
   sensor_id text, year int, month int, day int, hour int,...
   PRIMARY KEY ((sensor_id), year, month, day, hour)
) WITH CLUSTERING ORDER BY (year DESC, month DESC, day DESC, hour DESC);
```

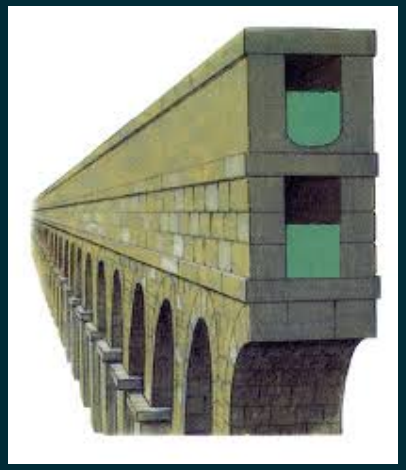
Stream Processing

- Kafka - Foundation to streaming architecture
- Samza - Just streaming
- Gearpump - Real-time big data streaming

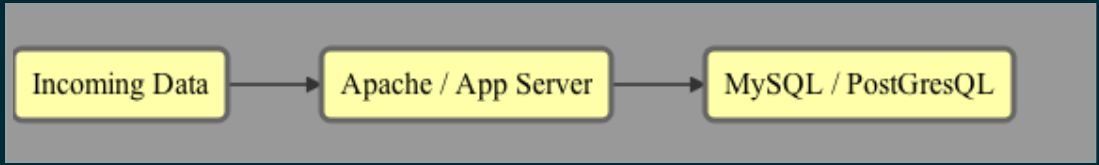
Analytics Stream Processing

- Storm - Real-time Analytics, ML, needs Trident to stream
- Flink - Real-time Analytics, ML, Graph
- Spark Streaming - Micro-batch Analytics, ML, Graph

So, let's talk Architecture!

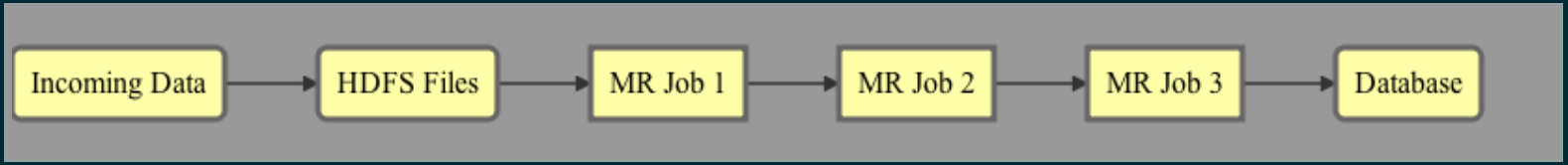


LAMP Architecture



- Could this handle streaming data and analytics?
- Not scalable, but super simple

Hadoop - Big Data Architecture

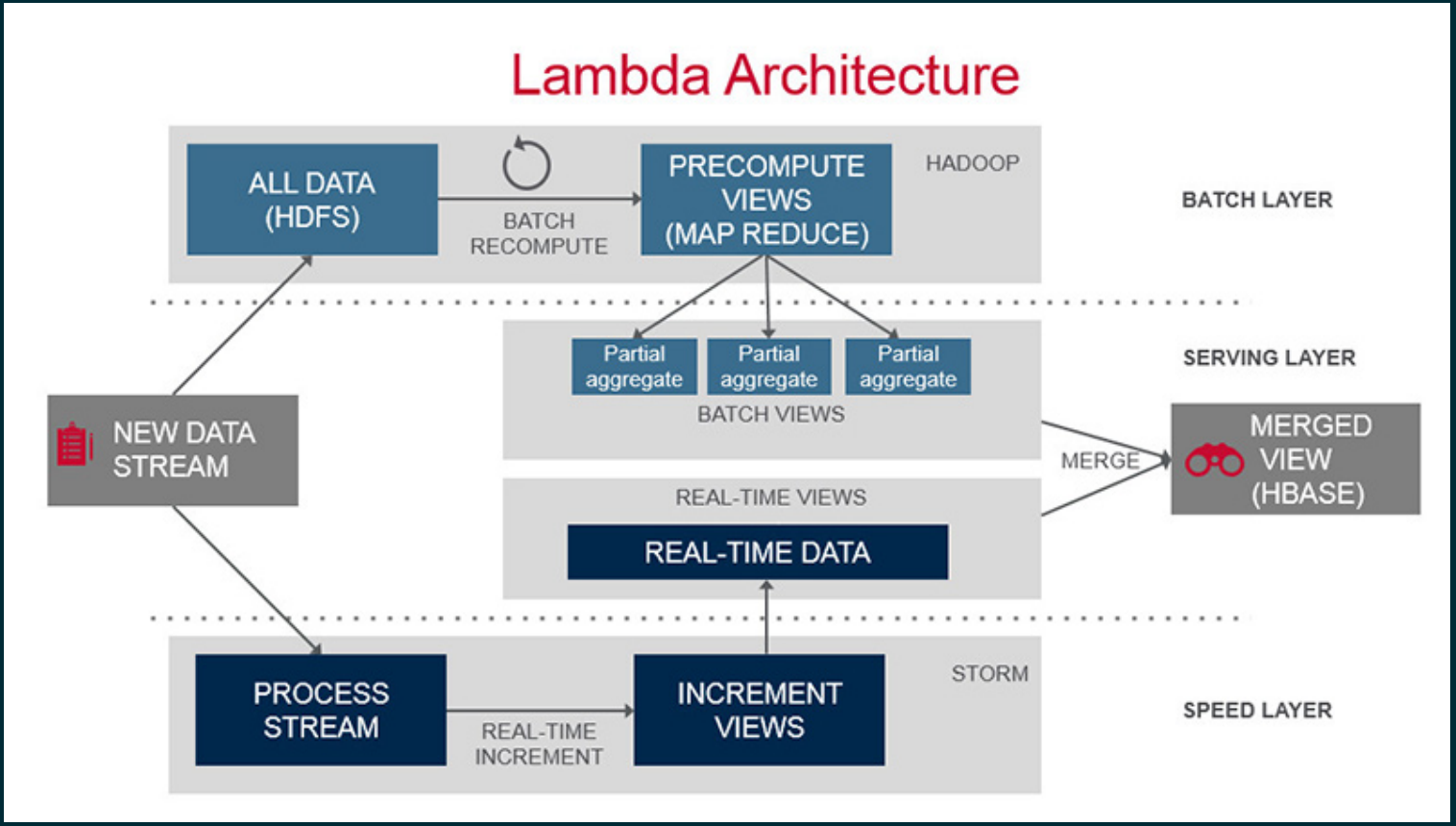


- Everybody jumped into Hadoop clusters to scale up data processing
- Write data into files, process in batches
- Spark: much faster, but same paradigm (batches)
- Pros: easy to understand
- Cons: high latency, not real time, very complex to set up

Lambda Architecture

A data-processing architecture designed to handle massive quantities of data by taking advantage of both batch and stream processing methods.

Lambda Architecture



(https://www.mapr.com/developercentral/lambda-architecture)

λ The Good

- Immutability - retaining master data
  - With timestamped events
  - Appended versus overwritten events
- Attempt to beat CAP
- Pre-computed views for
  - further processing
  - faster ad-hoc querying

λ The Bad

- Two Analytics systems to support
- Operational complexity
- By the time a scheduled job is run 90% of the data is stale
- Many moving parts: KV store, real time platform, batch technologies
- Running similar code and reconciling queries in *dual systems*
- Analytics logic changes on *dual systems*

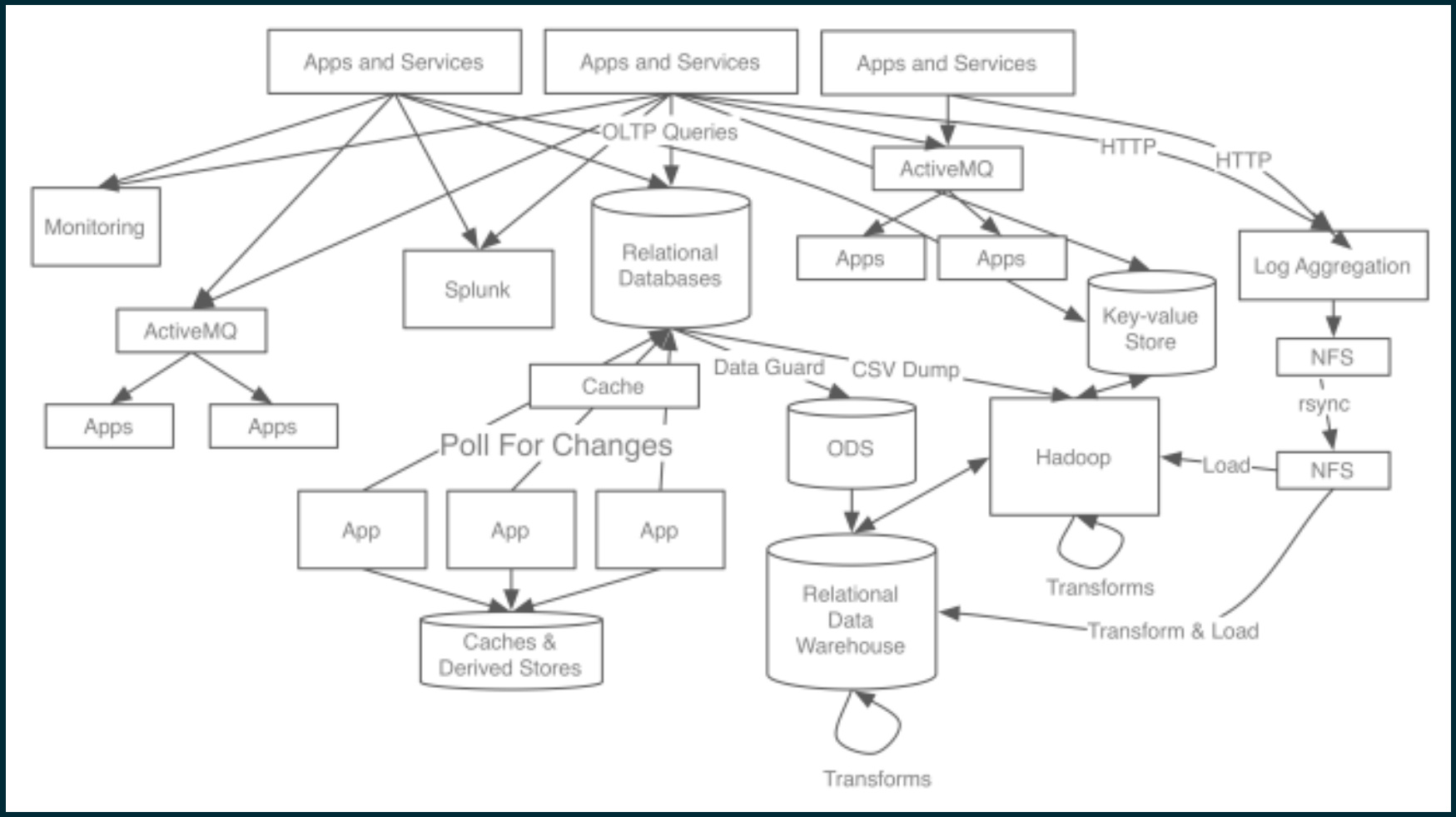
λ The Overly Complicated

Immutable sequence of records is ingested and fed into

- a batch processing system
- and a stream processing system
- in parallel

Ultimately Very High TCO And...

THE WILD WEST



Are Batch and Streaming Systems Fundamentally Different?

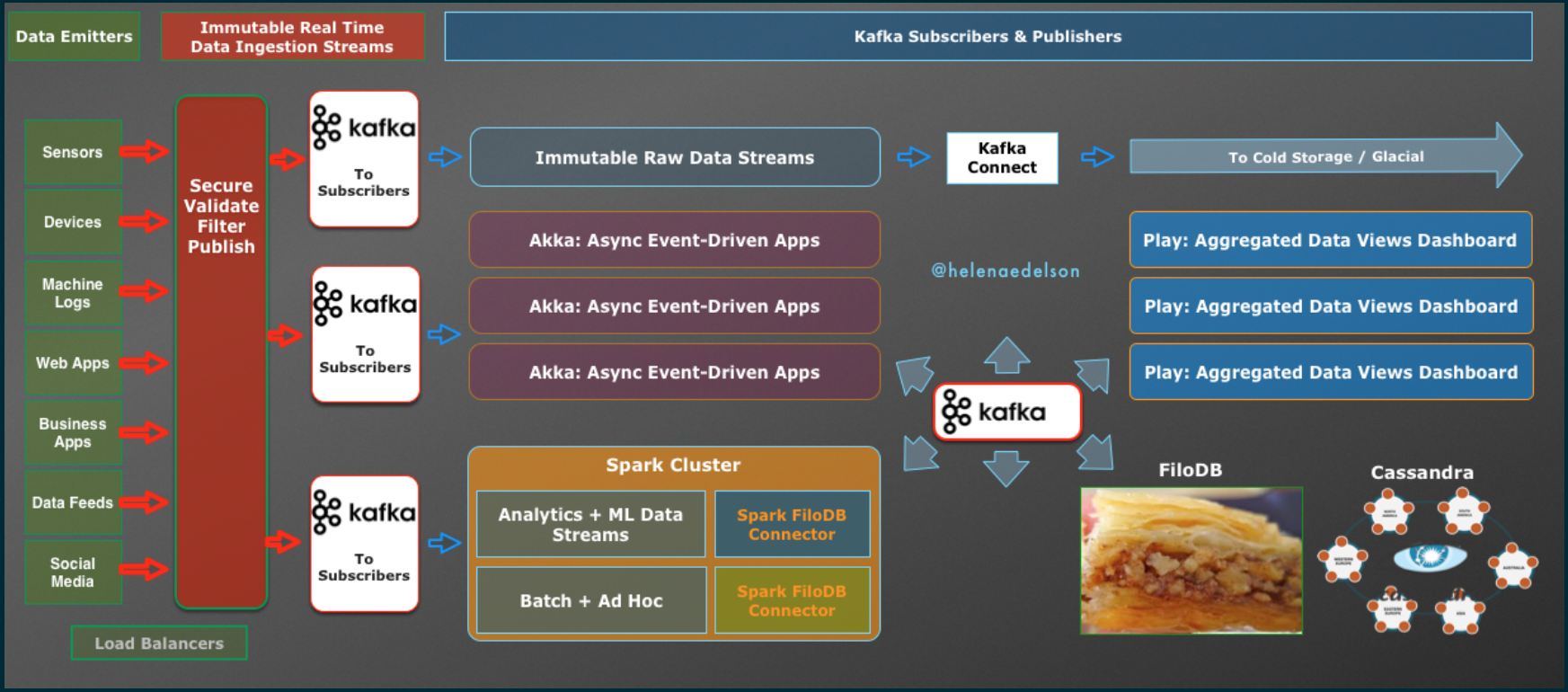
- Why is batch processing not thought of as a stream?
- Are we not accumulating events from *T-1* to *T-n*
- Or bucketed by Year-Month-Day-Hour?
- Can't we process everything as a stream?

A Unified Streaming Architecture

Everything On The Streaming Platform

- Scala / Spark Streaming
- Mesos
- Akka
- Cassandra
- Kafka

SNACK (SMACK) Stack



- High Throughput Distributed Messaging
- High Scalability - billions of events per day
- Durability - no data loss
- Immutability
- Support Massive Number of Consumers
- Very efficient and low latency
- Decouples Data Pipelines
- Automatic recovery from broker failures

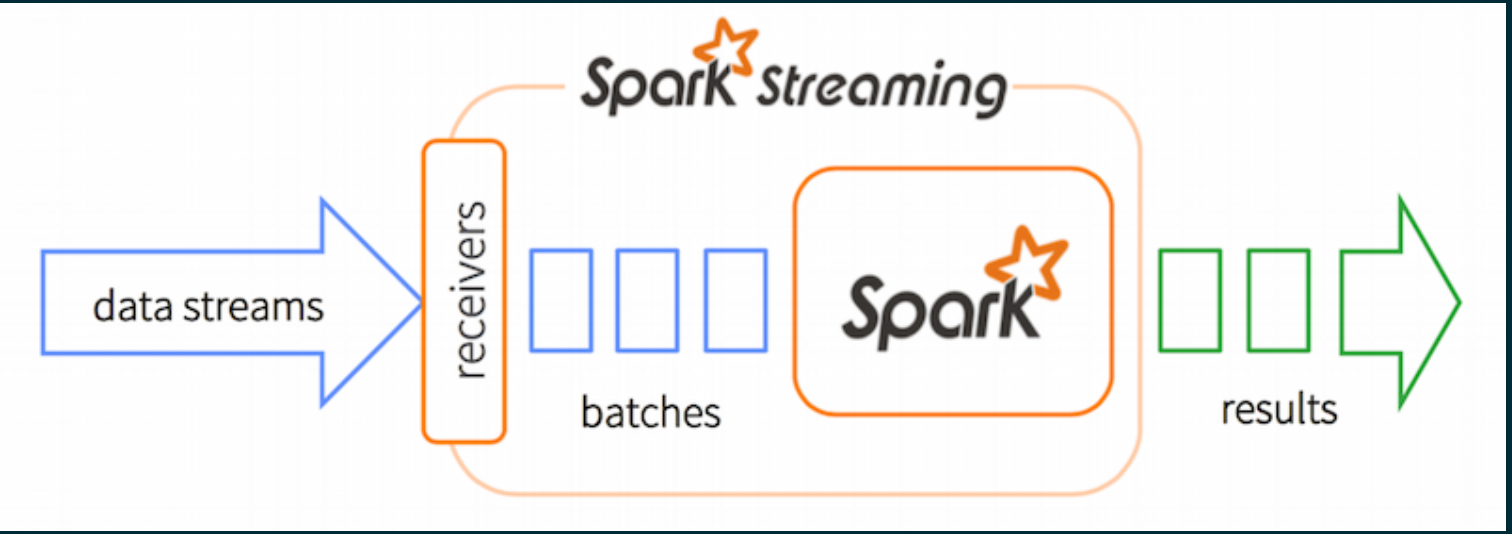
Stream Processing Simplified

Kafka Streams

- In master, coming in v0.10
- Removes the need to run another framework like Storm alongside Kafka
- Removes the need for separate infrastructures
- Common stream operations, e.g. join, filter, map, etc.
- Windowing
- Proper time modeling, e.g. event time vs. processing time
- Local state management with persistence and replication
- Schema and Avro support

Spark Streaming

Iterative ML, Interactive Querying, Graph, DataFrames



- One runtime for streaming and batch processing
- Join streaming and static data sets
- No code duplication
- Easy Kafka stream integration
- Easy to reconcile queries against multiple sources
- Easy integration of KV durable storage

Apache Cassandra

- Horizontally scalable
- Multi-Region / Multi-Datacenter
- Always On - Survive regional outages
- Extremely fast writes: - perfect for ingestion of real time / machine data
- Very flexible data modelling (lists, sets, custom data types)
- Easy to operate
- Best of breed storage technology, huge community
- **BUT: Simple queries only**
- **OLTP-oriented**/center

- High performance concurrency framework for Scala and Java
- Fault Tolerance
- Asynchronous messaging and data processing
- Parallelization
- Location Transparency
- Local / Remote Routing
- Akka: Cluster / Persistence / Streams

Kafka Streams

```
KStreamBuilder builder = new KStreamBuilder();
KStream<K, V> stream = builder.stream(des, des, "raw.data.topic")
  .flatMapValues(value -> Arrays.asList(value.toLowerCase.split(" "))
  .map((k,v) -> new KeyValue(k,v))
  .countByKey(ser, ser, des, des, "kTable")
  .toStream();

stream.to("results.topic", ...);

KafkaStrams streams = new KafkaStreams(builder, props);
streams.start();
```

https://github.com/confluentinc/demos

Spark Streaming Kafka

Immutable Raw Data From Kafka Stream

Replaying data streams: for fault tolerance, logic changes..

```
class KafkaStreamingActor(ssc: StreamingContext) extends MyAggregationActor {

  val stream = KafkaUtils.createDirectStream(...).map(RawWeatherData(_))

  stream
    .foreachRDD(_.toDF.write.format("filodb.spark")
                .option("dataset", "rawdata").save())

  /* Pre-Aggregate data in the stream for fast querying and aggregation later.

   stream.map(hour =>
     (hour.wsid, hour.year, hour.month, hour.day, hour.oneHourPrecip)
  ).saveToCassandra(timeseriesKeyspace, dailyPrecipTable)
}
```

Reading Data From Cassandra On Request, Further Aggregation

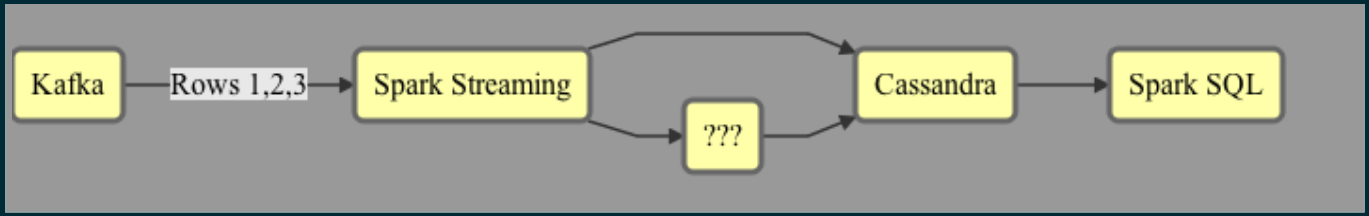
Compute isolation in Akka Actor

```
class TemperatureActor(sc: SparkContext) extends AggregationActor {
  import akka.pattern.pipe

  def receive: Actor.Receive = {
    case e: GetMonthlyHiLowTemperature => highLow(e, sender)
  }

  def highLow(e: GetMonthlyHiLowTemperature, requester: ActorRef): Unit =
    sc.cassandraTable[DailyTemperature](timeseriesKeyspace, dailyTempAggregTab
      .where("wsid = ? AND year = ? AND month = ?", e.wsid, e.year, e.month)
      .collectAsync()
      .map(MonthlyTemperature(_, e.wsid, e.year, e.month)) pipeTo requester
}
```

Spark Streaming, MLLib
Kafka, Cassandra

```
val ssc = new StreamingContext(sparkConf, Seconds(5))
val testData = ssc.cassandraTable[String](keyspace,table)
    .map(LabeledPoint.parse)

val trainingStream = KafkaUtils.createDirectStream[_,_,_,_](..)
    .map(transformFunc)
    .map(LabeledPoint.parse)

trainingStream.saveToCassandra("ml_training_keyspace", "raw_training_data")

val model = new StreamingLinearRegressionWithSGD()
  .setInitialWeights(Vectors.dense(weights))
  .trainOn(trainingStream)

model
  .predictOnValues(testData.map(lp => (lp.label, lp.features)))
  .saveToCassandra("ml_predictions_keyspace", "predictions")
```

What's Missing? One Pipeline For Fast + Big Data



Using Cassandra for Batch Analytics / Event Storage / ML?

- Storage efficiency and scan speeds for reading large volumes of data (for complex analytics, ML) become important concerns
- Regular Cassandra CQL tables are not very good at either storage efficiency or scan speeds
- A different, analytics-optimized solution is needed...

All hard work leads to profit, but mere talk leads to poverty.
- Proverbs 14:23

Introducing FiloDB

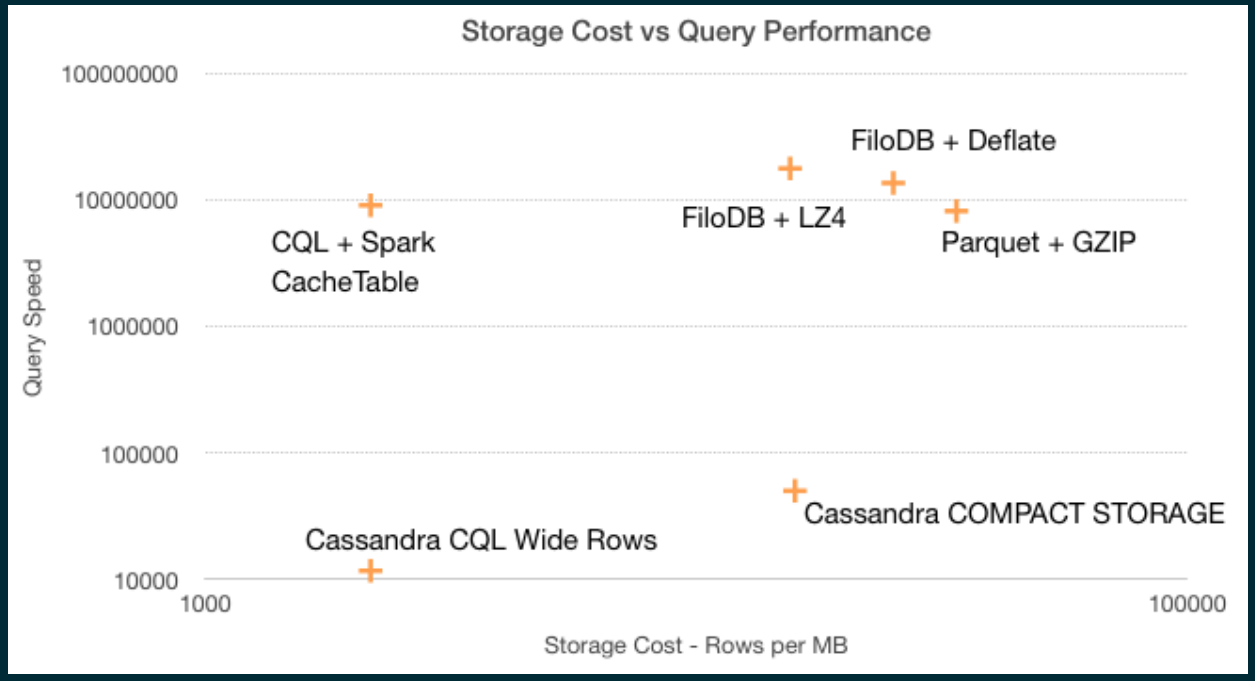
A distributed, versioned, columnar analytics database.
Built for Streaming.

github.com/tuplejump/FiloDB

Fast Analytics Storage

- Scan speeds competitive with Apache Parquet
  - Up to 200x faster scan speeds than with Cassandra 2.x
- Flexible filtering along two dimensions
  - Much more efficient and flexible partition key filtering
- Efficient columnar storage, up to 40x more efficient than Cassandra 2.x

Comparing Storage Costs and Query Speeds



https://www.oreilly.com/ideas/apache-cassandra-for-analytics-a-performance-and-storage-analysis

Robust Distributed Storage

Apache Cassandra as the rock-solid storage engine. Scale out with no SPOF. Cross-datacenter replication. Proven storage and database technology.

Cassandra-Like Data Model

	Column A		Column B	
Partition key 1	Segment 1	Segment 2	Segment 1	Segment 2
Partition key 2	Segment 1	Segment 2	Segment 1	Segment 2

- **partition keys -** distributes data around a cluster, and allows for fine grained and flexible filtering
- **segment keys -** do range scans within a partition, e.g. by time slice
- primary key based ingestion and updates

Designed for Streaming

- New rows appended via Spark Streaming or Kafka
- Writes are *idempotent* - easy **exactly once** ingestion
- Converted to columnar chunks on ingest and stored in C*
- FiloDB keeps your data sorted as it is being ingested

Spark SQL Queries!

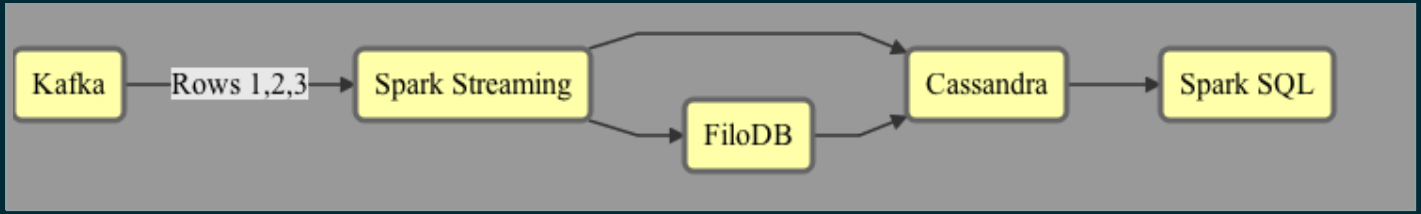
```
CREATE TABLE gdelt USING filodb.spark OPTIONS (dataset "gdelt");

SELECT Actor1Name, Actor2Name, AvgTone FROM gdelt ORDER BY AvgTone DESC LIMIT

INSERT INTO gdelt SELECT * FROM NewMonthData;
```

- Read to and write from Spark Dataframes
- Append/merge to FiloDB table from Spark Streaming
- Use Tableau or any other JDBC tool

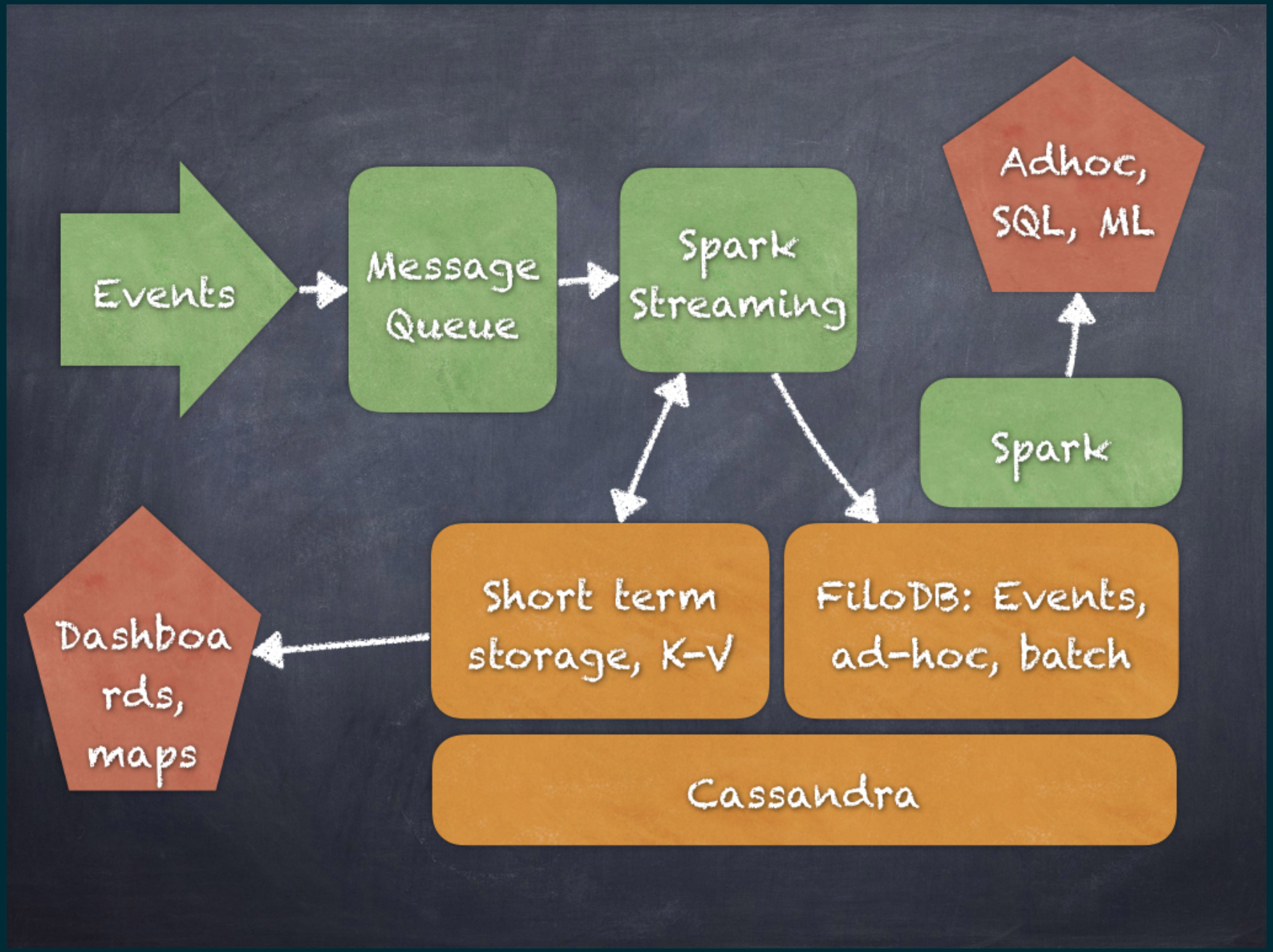
What's in the name?

Rich sweet layers of distributed, versioned database goodness

NoLambda / SNACK stack for Real-Time + AdHoc + Deep Analytics



- Regular Cassandra tables for highly concurrent, aggregate / key-value lookups (dashboards)
- FiloDB + C* + Spark for efficient long term event storage
  - Ad hoc / SQL / BI
  - Data source for MLLib / building models
  - Data storage for classified / predicted / scored data


Events
Message Queue
Spark Streaming
Adhoc, SQL, ML
Spark
Dashboa rds, maps
Short term storage, K-V
FiloDB: Events, ad-hoc, batch
Cassandra

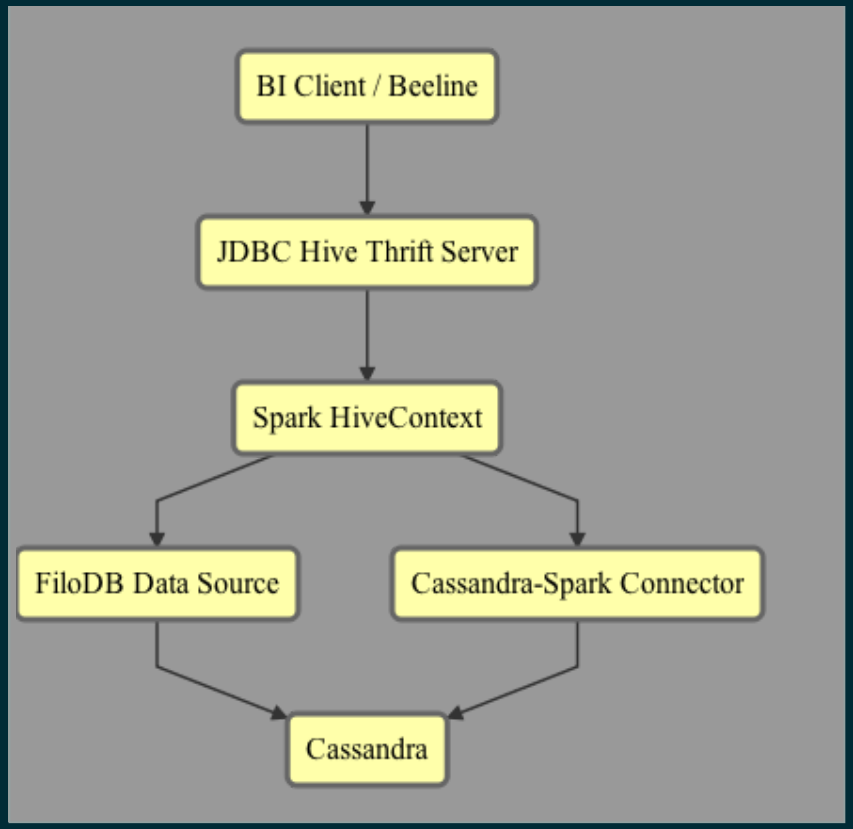
Being Productionized as we speak...

- One enterprise with many TB of financial and reporting data is moving their data warehouse to FiloDB + Cassandra + Spark
- Another startup uses FiloDB as event storage, feeds the events into Spark MLlib, scores incoming data, then stores the results back in FiloDB for low-latency use cases
  - From their CTO: “I see close to MemSQL / Vertica or even better” “More cost effective than Redshift”

FiloDB Use Cases

- Data Warehousing / BI
  - < 10 second SLA, nontrivial reports, some concurrency
  - need to store and query lots of data efficiently
- Time series
  - idempotent write API, simultaneous write and read workloads
- In-memory SQL web server
  - 700 queries per second using in-memory column store

Real-world DW Architecture Stack



Efficient columnar storage + filtering = low latency BI

FiloDB vs HDFS/Parquet

	FiloDB	Parquet
Ingestion	Idempotent primary-key based; appends and replaces; deletes coming	File-based append API only
Filtering	Partition-key and segment-key filtering	Mostly file-based
Scan speeds	Parquet-like	Good for OLAP
Storage cost	Within 35% of Parquet	

FiloDB vs HDFS/Parquet

In practice, with good data modeling, FiloDB is a far better fit for low-latency / concurrent BI / reporting / dashboard applications.

FiloDB vs Druid

Different use cases:

- Druid is optimized mostly for OLAP cube / slice and dice analysis. Append only, keeps only aggregates, not a raw event store.
- FiloDB stores raw data - can be used to build ML models, visualize and analyze raw time series data, do complex event flow analysis - much more flexible
- FiloDB can update/replace data
- FiloDB does not require data denormalization - can handle traditional BI star schemas with slowly changing dimension tables

Modeling example: NYC Taxi Dataset

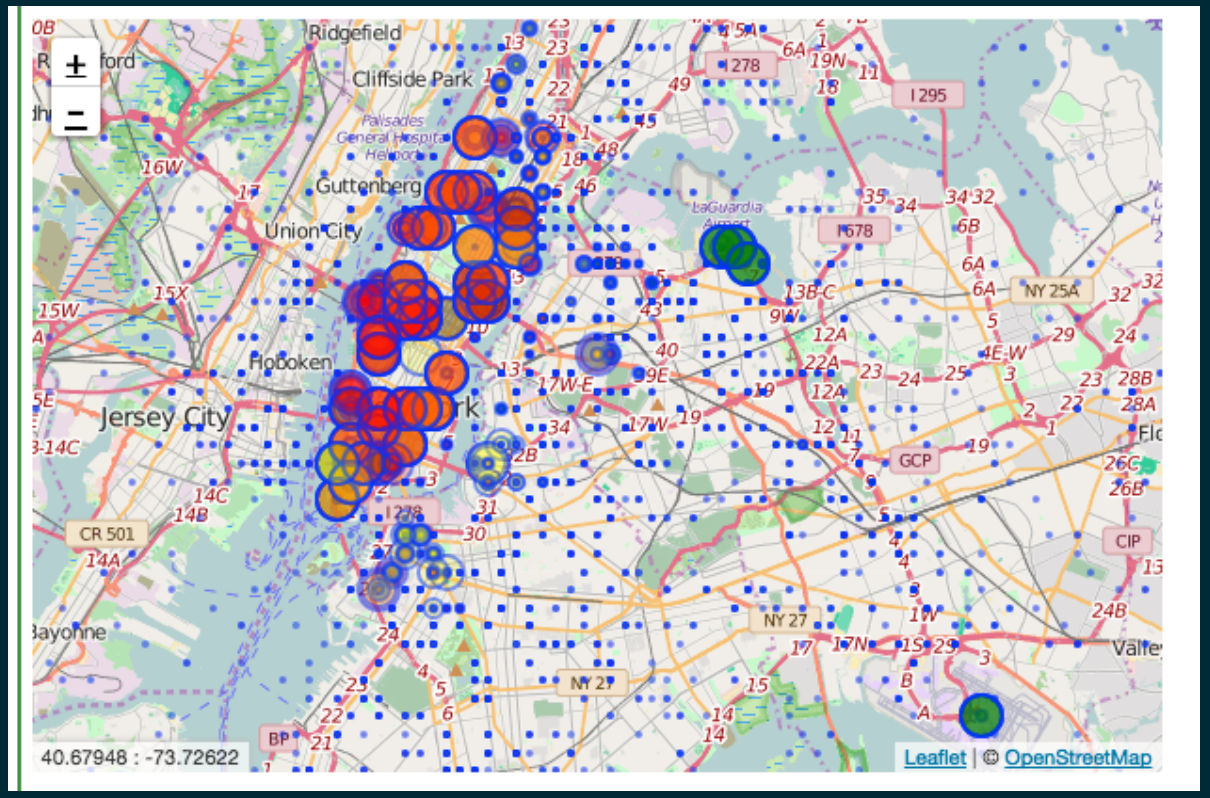
The public NYC Taxi Dataset contains telemetry (pickup, dropoff locations, times) info on millions of taxi rides in NYC.

Medallion prefix	1/1 - 1/6	1/7 - 1/12
AA	records	records
AB	records	records

- Partition key - `:stringPrefix medallion 2` - hash multiple drivers trips into ~300 partitions
- Segment key - `:timeslice pickup_datetime 6d`
- Row key - hack_license, pickup_datetime

Allows for easy filtering by individual drivers, and slicing by time.

New York City Taxi Data Demo (Spark Notebook)



To follow along:

https://github.com/tuplejump/FiloDB/blob/master/doc/FiloDB_Taxi_G

Machine Learning with Spark, Cassandra, and FiloDB

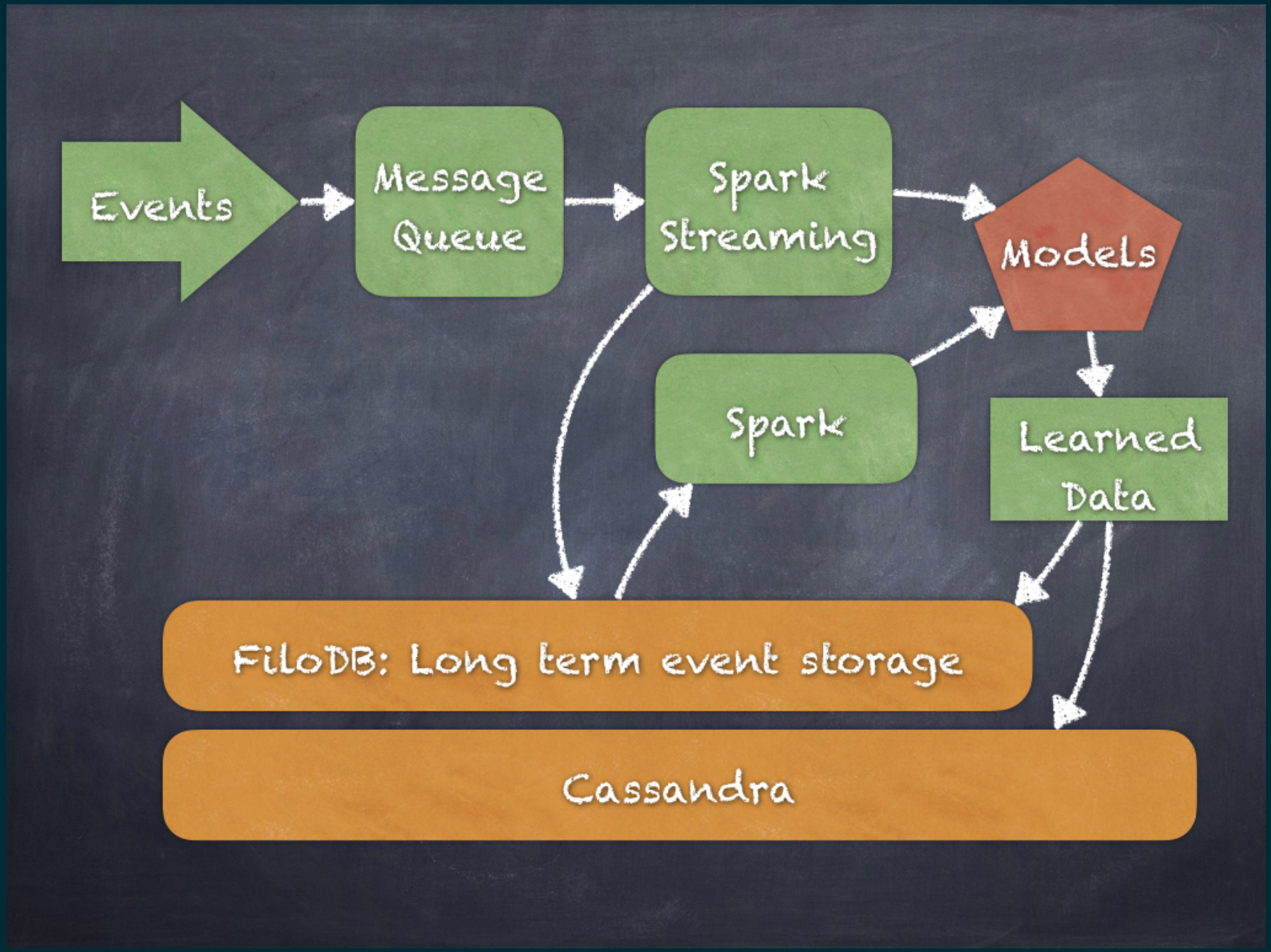
Building a static model of NYC Taxi Trips

- Predict time to get to destination based on pickup point, time of day, other vars
- Need to read all data (full table scan)

Dynamic models are better than static models

- Everything changes!
- Continuously refine model based on recent streaming data + historical data + existing model

```
val ssc = new StreamingContext(sparkConf, Seconds(5))
val dataStream = KafkaUtils.createDirectStream[..](..)
    .map(transformFunc)
    .map(LabeledPoint.parse)

dataStream.foreachRDD(_.toDF.write.format("filodb.spark")
                                  .option("dataset", "training").save())

if (trainNow) {
var model = new StreamingLinearRegressionWithSGD()
    .setInitialWeights(Vectors.dense(weights))
    .trainOn(dataStream.join(historicalEvents))
}

model.predictOnValues(dataStream.map(lp => (lp.label, lp.features)))
     .insertIntoFilo("predictions")
```


Events
Message Queue
Spark Streaming
Models
Spark
Learned Data
FiloDB: Long term event storage
Cassandra

The FiloDB Advantage for ML

- Able to update dynamic models based on massive data flow/updates
  - Integrate historical and recent events to build models
- More data -> better models!
- Can store scored raw data / predictions back in FiloDB
  - for fast user queries

FiloDB - Roadmap

Your input is appreciated!

- Productionization and automated stress testing
- Kafka input API / connector (without needing Spark)
- In-memory caching for significant query speedup
- True columnar querying and execution, using late materialization and vectorization techniques. GPU/SIMD.
- Projections. Often-repeated queries can be sped up significantly with projections.

Thanks For Attending!

- @evanfchan
- evan@tuplejump.com

EXTRA SLIDES

Data Warehousing with FiloDB

Scenarios

- BI Reporting, concurrency + seconds latency
- Ad-hoc queries
- Needing to do JOINs with fact tables + dimension tables
  - Slowly changing dim tables / hard to denormalize
- Need to work with legacy BI tools

Modeling Fact Tables for FiloDB

- Single partition queries are really fast and take up only one thread
  - Given the following two partition key columns: entity_number, year_month
  - `WHERE entity_number = '0453' AND year_month = '2014 December'`
    - Exact match for partition key is pushed down as one partition
- Consider the partition key carefully

Cassandra often requires multiple tables

What about the queries that do not translate to one partition? Cassandra has many restrictions on partition key filtering (as of 2.x).

- Table 1: partition key = `(entity_number, year_month)`
  - Can push down: `WHERE entity_number = NN AND year_month IN ('2014 Jan', '2014 Feb')` as well as equals
- Table 2: partition key = `(year_month, entity_number)`
  - Can push down: `WHERE year_month = YYMM AND entity_number IN (123, 456)` as well as equals

IN clause must be the last column to be pushed down. Two tables are needed just for efficient IN queries on either entity_number or year_month.

Flexible Filtering

Unlike Cassandra, FiloDB offers very flexible and efficient filtering on partition keys. Partial key matches, fast IN queries on any part of the partition key.

No need to write multiple tables to work around answering different queries.

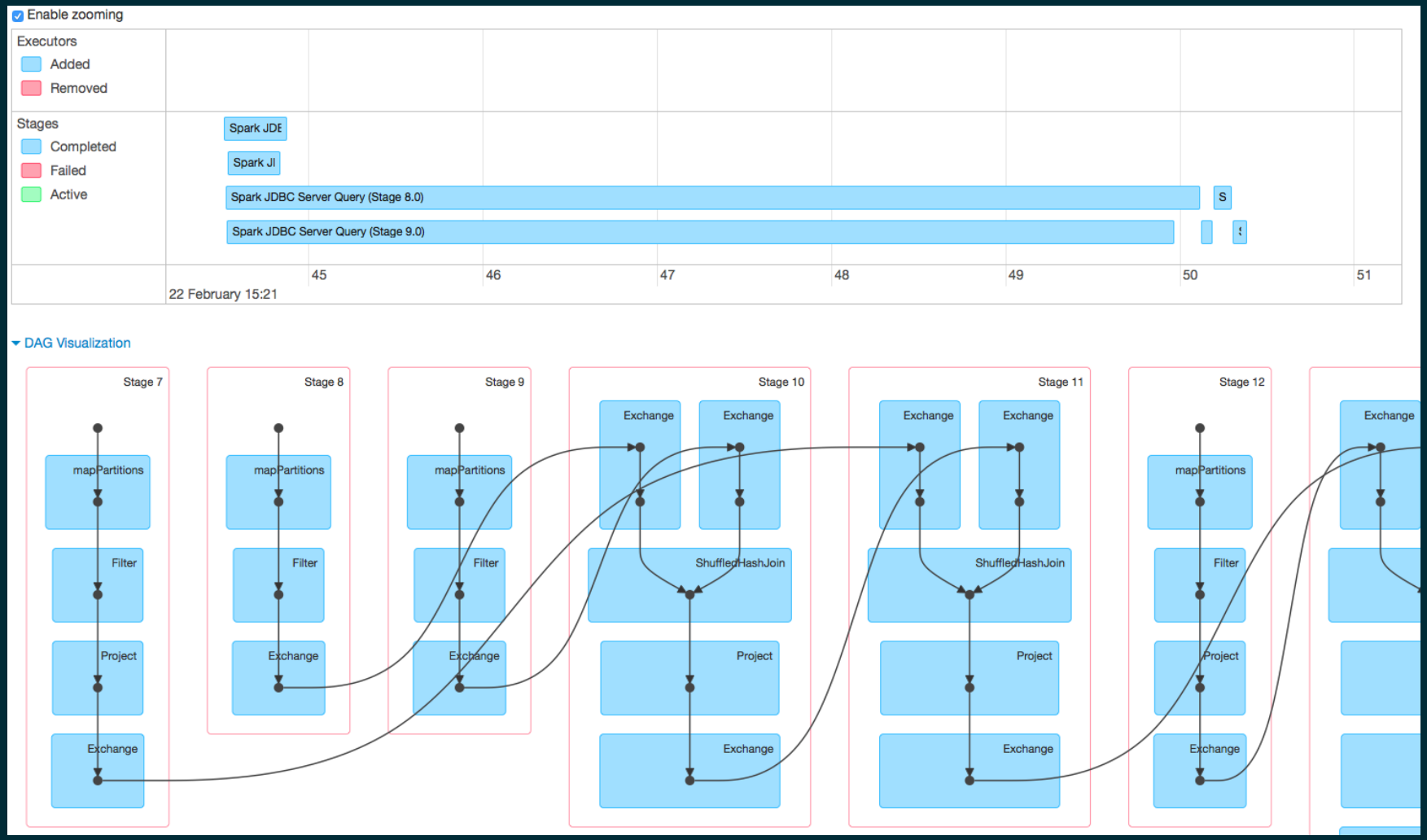
FiloDB Flexible Partition Filters = WIN

With ONE table, FiloDB offers FAST, arbitrary partition key filtering. All of the below are pushed down:

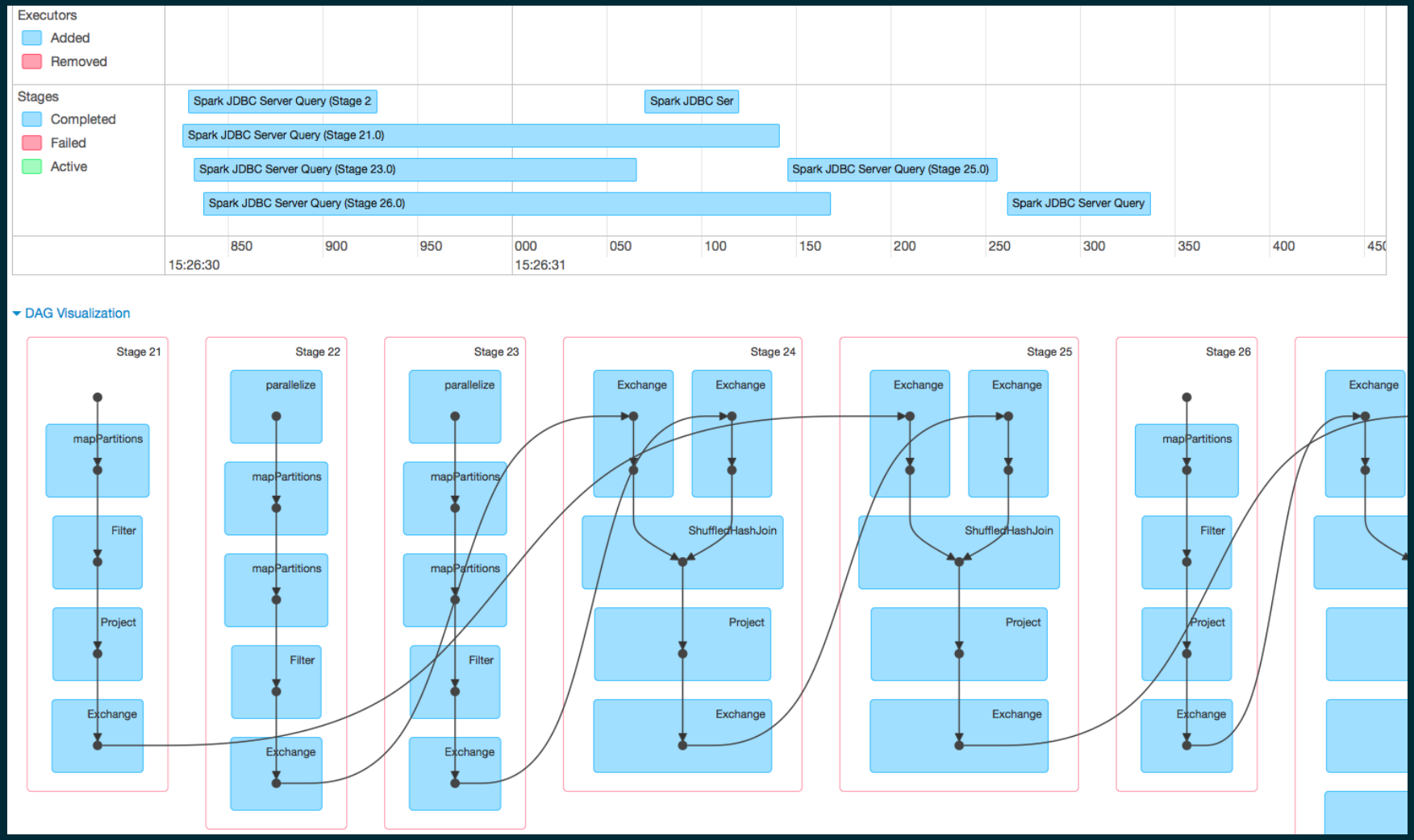
- `WHERE year_month IN ('2014 Jan', '2014 Feb')` (all entities)
- `WHERE entity_number = 146` (all year months)
- Any combo of =, IN

Space savings: 27 * 2 = **54x**

Multi-Table JOINs with just Cassandra



Sub-second Multi-Table JOINs with FiloDB



Sub-second Multi-Table JOINs with FiloDB

- Four tables, all of them single-partition queries
- Two tables were switched from regular Cassandra tables to FiloDB tables. 40-60 columns each, ~60k items in partition.
- Scan times went down from 5-6 seconds to < 250ms

For more details, please see this Planet Cassandra blog post.

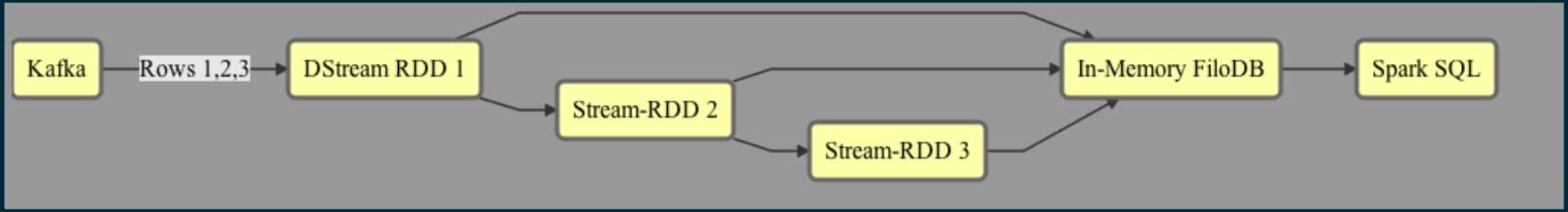
Scalable Time-Series / Event Storage with FiloDB

Spark Streaming -> FiloDB

```
val ratingsStream = KafkaUtils.createDirectStream[String, String, StringDe
ratingsStream.foreachRDD {
  (message: RDD[(String, String)], batchTime: Time) => {
    val df = message.map(_._2.split(",")).map(rating => Rating(rating(0).t
      toDF("fromuserid", "touserid", "rating")

    // add the batch time to the DataFrame
    val dfWithBatchTime = df.withColumn("batch_time", org.apache.spark.sql

    // save the DataFrame to FiloDB
    dfWithBatchTime.write.format("filodb.spark")
      .option("dataset", "ratings")
      .save()
  }
}
```

One-line change to write to FiloDB vs Cassandra

Fast, Updatable In-Memory Columnar Storage



- Unlike RDDs and DataFrames, FiloDB can ingest new data, and still be fast
- Unlike RDDs, FiloDB can filter in multiple ways, no need for entire table scan
- FAIR scheduler + sub-second latencies => web speed queries

700 Queries Per Second in Apache Spark!

- Even for datasets with 15 million rows!
- Using FiloDB's `InMemoryColumnStore`, single host / MBP, 5GB RAM
- SQL to DataFrame caching

For more details, see this blog post.

FiloDB - How?

Multiple ways to Accelerate Queries

- Columnar projection - read fewer columns, saves I/O
- Partition key filtering - read less data
- Sort key / PK filtering - read from subset of keys
  - Possible because FiloDB keeps data sorted
- Versioning - write to multiple versions, read from the one you choose

Cassandra CQL vs Columnar Layout

Cassandra stores CQL tables row-major, each row spans multiple cells:

PartitionKey	01:first	01:last	01:age	02:first	02:last	02:age
Sales	Bob	Jones	34	Susan	O'Connor	40
Engineering	Dilbert	P	?	Dogbert	Dog	1

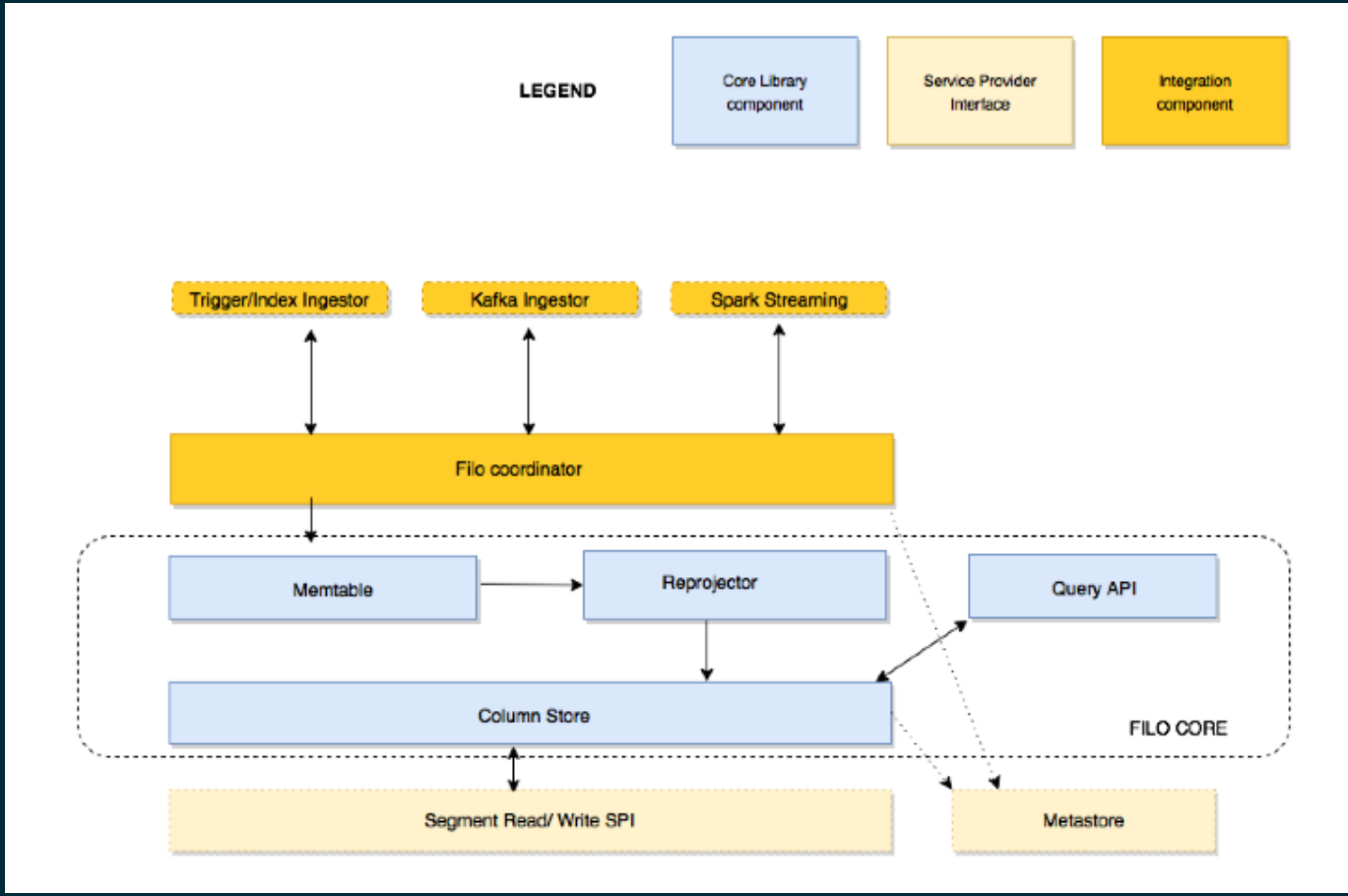
Columnar layouts are column-major:

PartitionKey	first	last	age
Sales	Bob, Susan	Jones, O'Connor	34, 40
Engineering	Dilbert, Dogbert	P, Dog	?, 1

FiloDB Cassandra Schema

```
CREATE TABLE filodb.gdelt_chunks (
    partition text,
    version int,
    columnname text,
    segmentid blob,
    chunkid int,
    data blob,
    PRIMARY KEY ((partition, version), columnname, segmentid, chunkid)
) WITH CLUSTERING ORDER BY (columnname ASC, segmentid ASC, chunkid ASC)
```

FiloDB Architecture



ColumnStore API - currently Cassandra and InMemory, you can implement other backends - ElasticSearch? etc.