RETHINKING STREAMING ANALYTICS FOR SCALE

Helena Edelson

@helenaedelson

Who Is This Person?

- VP of Product Engineering @Tuplejump
- Big Data, Analytics, Cloud Engineering, Cyber Security
- Committer / Contributor to FiloDB, Spark Cassandra Connector, Akka, Spring Integration

- @helenaedelson
- github.com/helena
- linkedin.com/in/helenaedelson
- slideshare.net/helenaedelson

Unify

Blender empowers line-of-business analysts to connect to disparate data sources and unify the data into a single normalized view ready for consumption.

Enrich

Transform, clean, enrich and curate your data easily using familiar tools and process. Blender will automagically build the storage model and ingestion pipeline.

Analyze

With Blender you can easily uncover hidden patterns, unknown correlations and other useful information to reach business decisions faster than ever.

Learn

Blender's predictive analytics framework and bundled learning algorithms along with an intuitive workflow enable you to harness this power with a few clicks.

tuplejump

Data Blender

www.tuplejump.com
info@tuplejump.com

Tuplejump - Open Source

github.com/tuplejump

- FiloDB - part of this talk
- Calliope - the first Spark-Cassandra integration
- Stargate - an open source Lucene indexer for Cassandra
- SnackFS - open source HDFS for Cassandra

What Will We Talk About

- The Problem Domain
- Example Context
- Rethinking Architecture
  - We don't have to look far to look back
  - Streaming & Data Science
  - Challenging Assumptions
  - Revisiting the goal and the stack
- Integration
- Simplification

Delivering Meaning From A Flood Of Data

THE PROBLEM DOMAIN

@helenaedelson

The Problem Domain

Need to build scalable, fault tolerant, distributed data processing systems that can handle massive amounts of data from disparate sources, with different data structures.

Translation

How to build adaptable, elegant systems
for complex analytics and learning tasks
to run as large-scale clustered dataflows

How Much Data

We all have a lot of data

- Terabytes
- Petabytes...

Yottabyte = *quadrillion gigabytes or septillion bytes*

100 trillion $ in DC fees

http://en.wikipedia.org/wiki/Yottabyte

Delivering Meaning

- Deliver meaning in sec/sub-sec latency
- Disparate data sources & schemas
- Billions of events per second
- High-latency batch processing
- Low-latency stream processing
- Aggregation of historical from the stream

While We Monitor, Predict & Proactively Handle

- Massive event spikes & bursty traffic
- Fast producers / slow consumers
- Network partitioning & out of sync systems
- DC down
- Wait, we've DDOS'd ourselves from fast streams?
- Autoscale issues
  - When we scale down VMs how do we not lose data?

And stay within our AWS / Rackspace budget

Hunting The Hunter

EXAMPLE CONTEXT: CYBER SECURITY

Adversary Profiling & Hunting: Online & Offline

- Track activities of international threat actor groups, nation-state, criminal or hactivist
  - Intrusion attempts
  - Actual breaches
- Profile adversary activity
  - Analysis to understand their motives, anticipate actions and prevent damage

Stream Processing

- Machine events
  - Endpoint intrusion detection
  - Anomalies/indicators of attack or compromise
- Machine learning
  - Training models based on patterns from historical data
  - Predict potential threats
  - profiling for adversary Identification

Data Requirements & Description

- Streaming event data
  - Log messages
  - User activity records
  - System ops & metrics data
- Disparate data sources
- Wildly differing data structures

Massive Amounts Of Data

- One machine can generate 2+ TB per day
- Tracking millions of devices
- 1 million writes per second - bursty
- High % writes, lower % reads

RETHINKING ARCHITECTURE

few years Cloud Engineering team

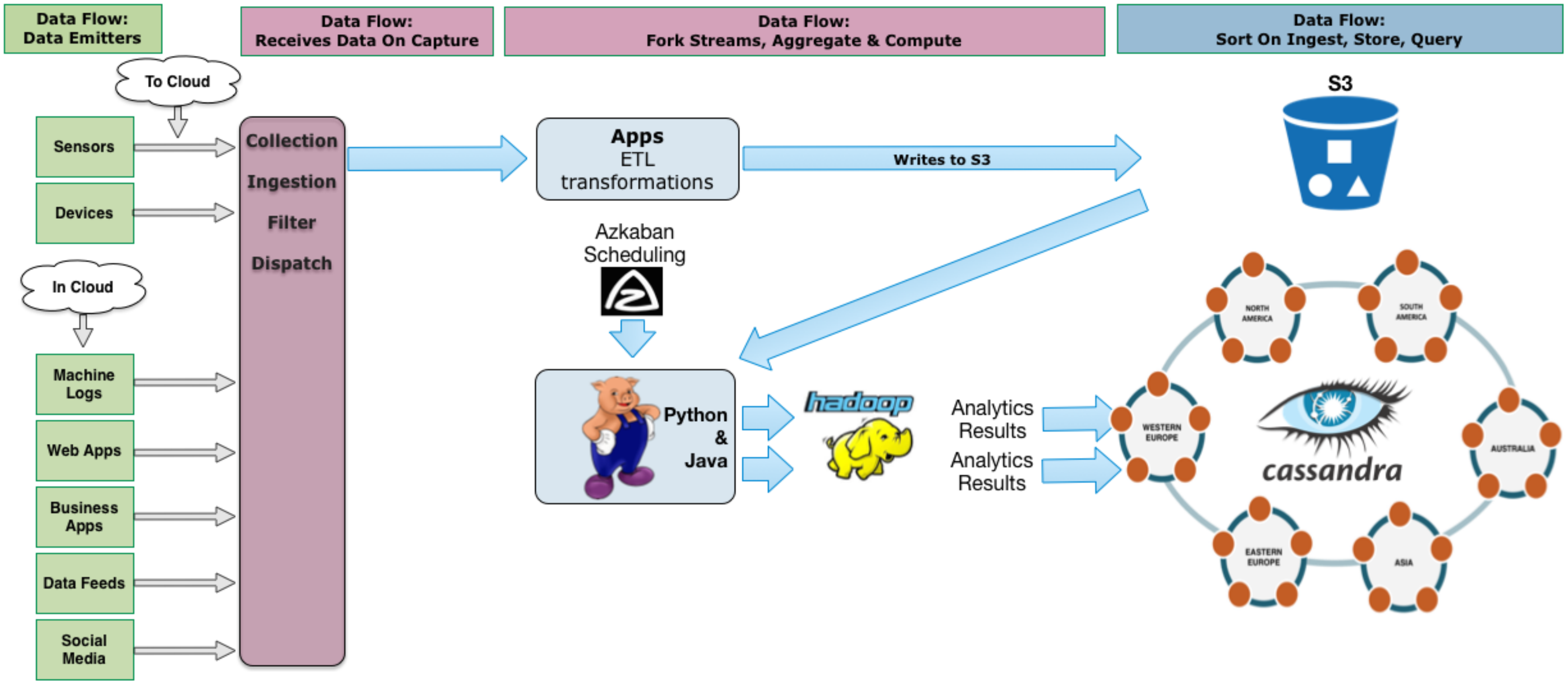
A ~~long time~~ ago in a ~~galaxy far,~~

in Silicon Valley

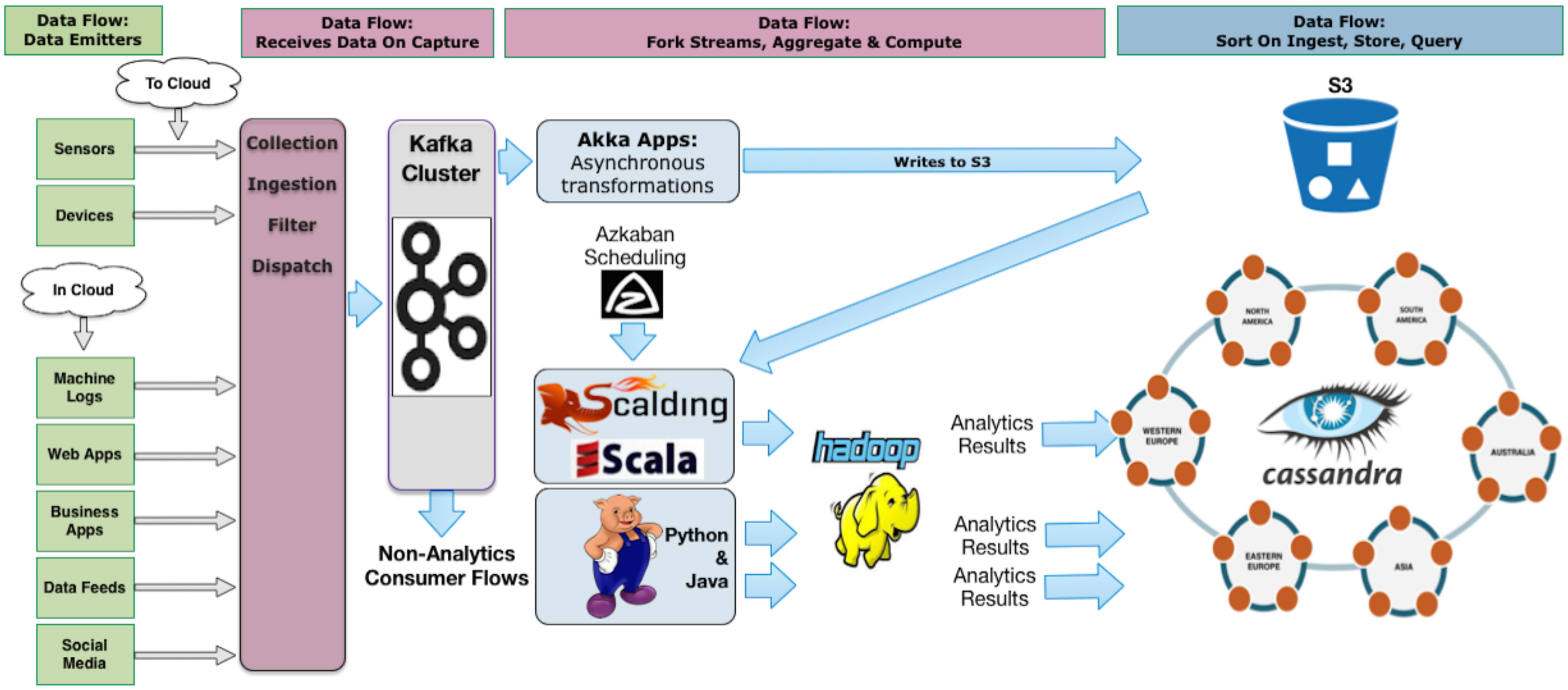
~~far away....~~

@helenaedelson

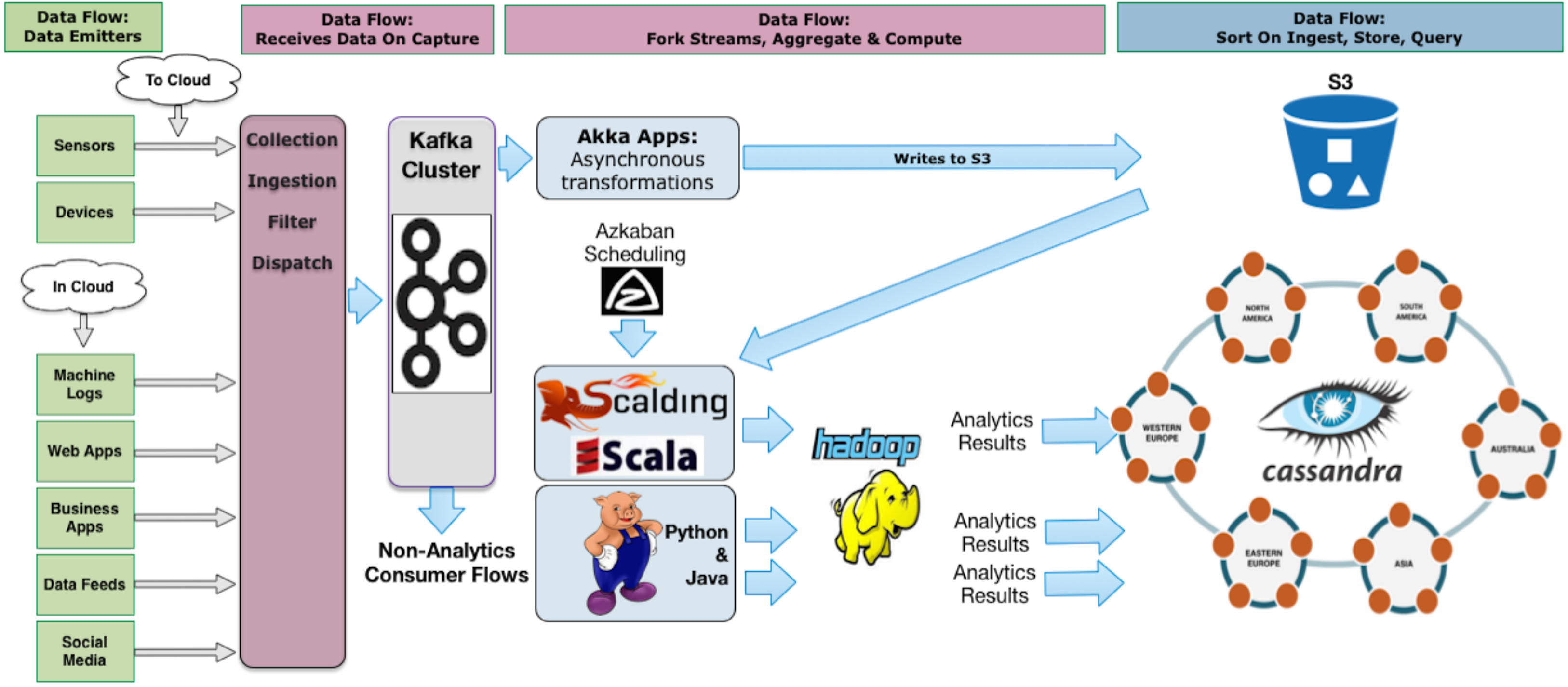
Batch analytics data flow from several years ago looked like...



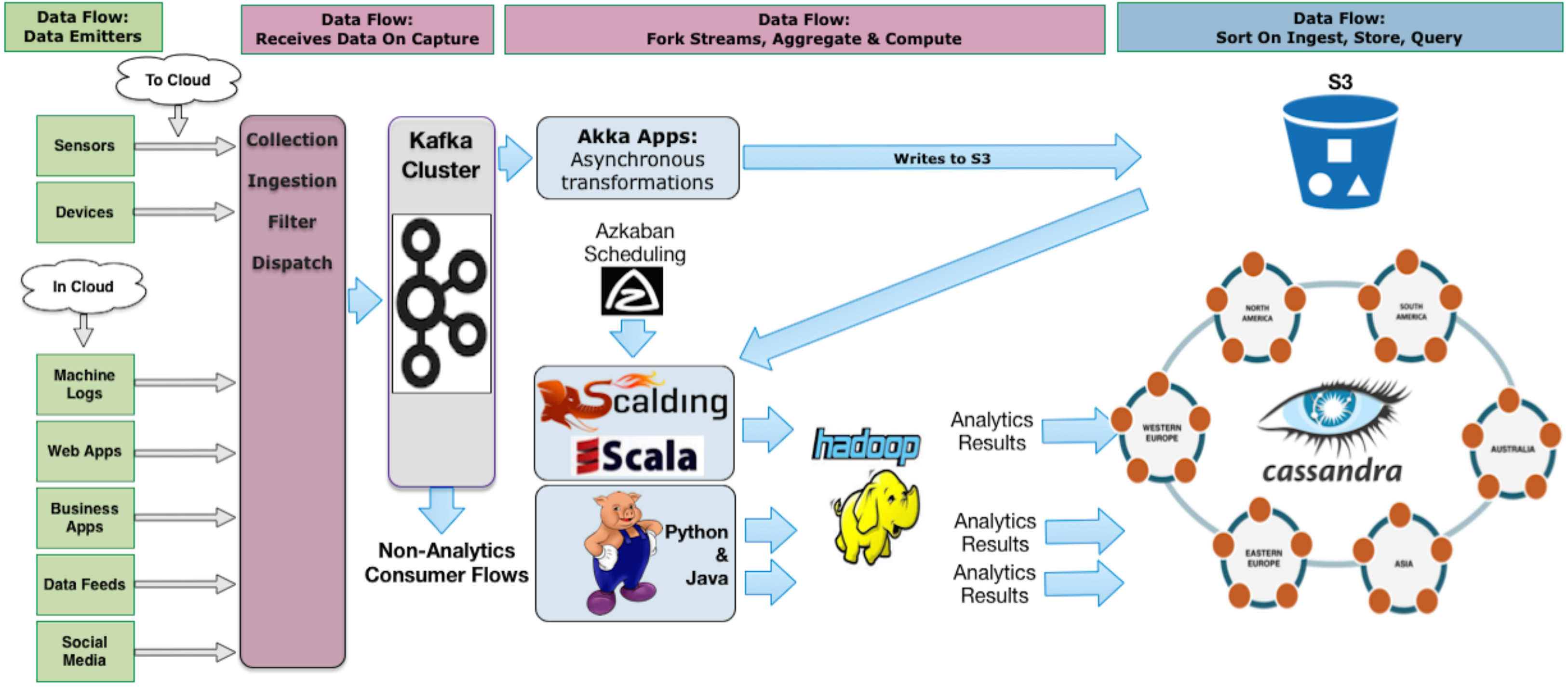
Batch analytics data flow from several years ago looked like...



Transforming data multiple times, multiple ways



Sweet, let's triple the code we have to update and regression test every time our analytics logic changes



@helenaedelson

tuplejump

Enter Streaming for Big Data

STREAMING & DATA SCIENCE

Streaming: Big Data, Fast Data, Fast Timeseries Data

- Reactive processing of data as it comes in to derive instant insights
- Is this enough?
  - Need to combine with existing big data, historical processing, ad hoc queries

New Requirements, Common Use Case

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

It's Not A Stream It's A Flood

- Netflix
  - 50 - 100 billion events per day
  - 1 - 2 million events **per second at peak**
- LinkedIn
  - 500 billion write events per day
  - 2.5 trillion read events per day
  - 4.5 million events **per second at peak *with Kafka***
  - 1 PB of stream data

Which Translates To

- Do it fast
- Do it cheap
- Do it at scale

Oh, and don't loose data

Lambda

AND THEN WE ***GREEKED*** OUT

Lambda Architecture

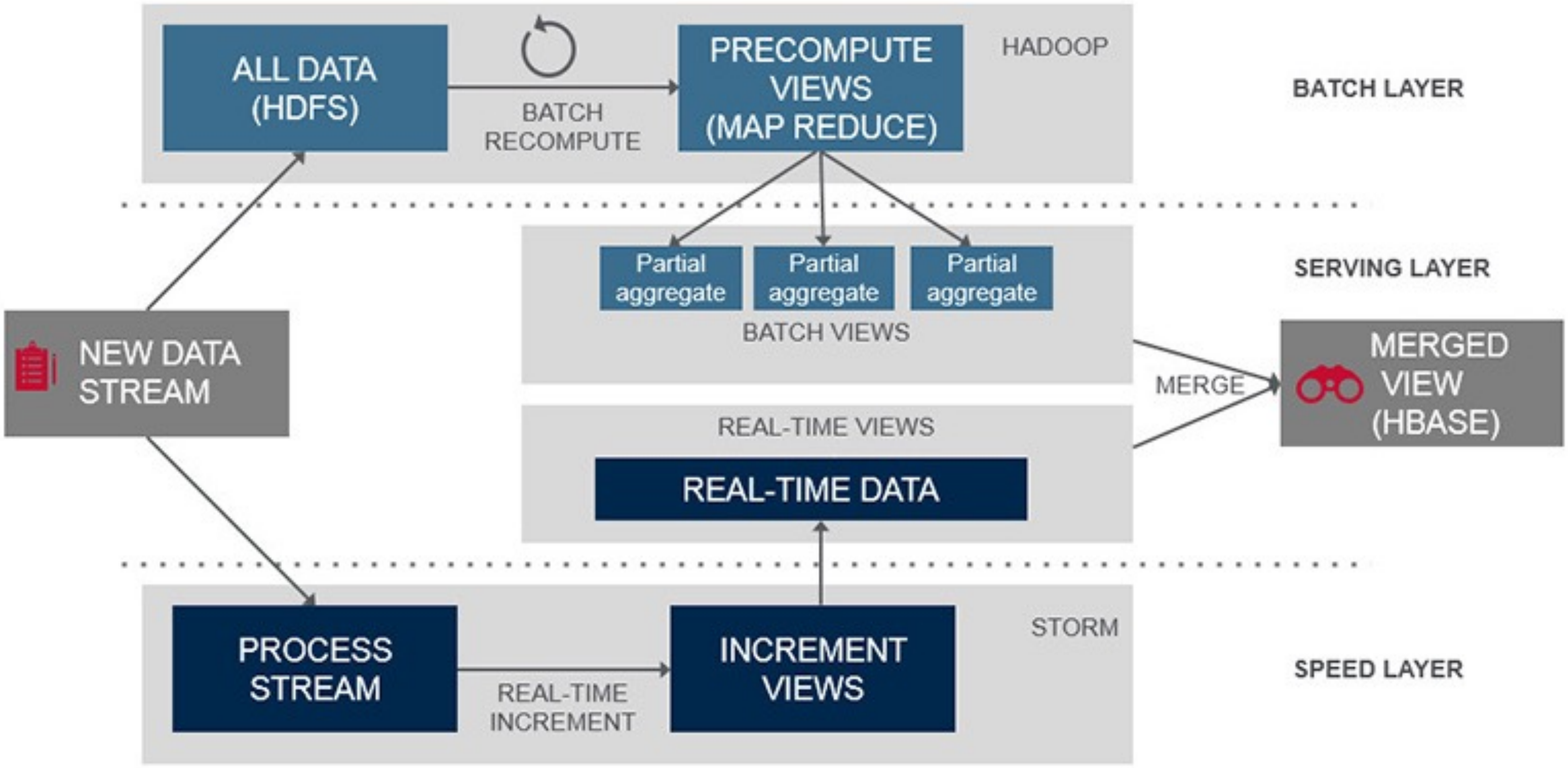
A data-processing architecture designed to handle *massive quantities* of data by taking advantage of both batch and stream processing methods.

- Or, "How to beat the CAP theorum"
- An approach coined by Nathan Mars
- This was a huge stride forward

Applications Using Lambda Architecture

- Doing complex asynchronous transformations
- That need to run with low latency (say, a few seconds to a few hours)
- Examples
  - Weather analytics and prediction system
  - News recommendation system

Lambda Architecture



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

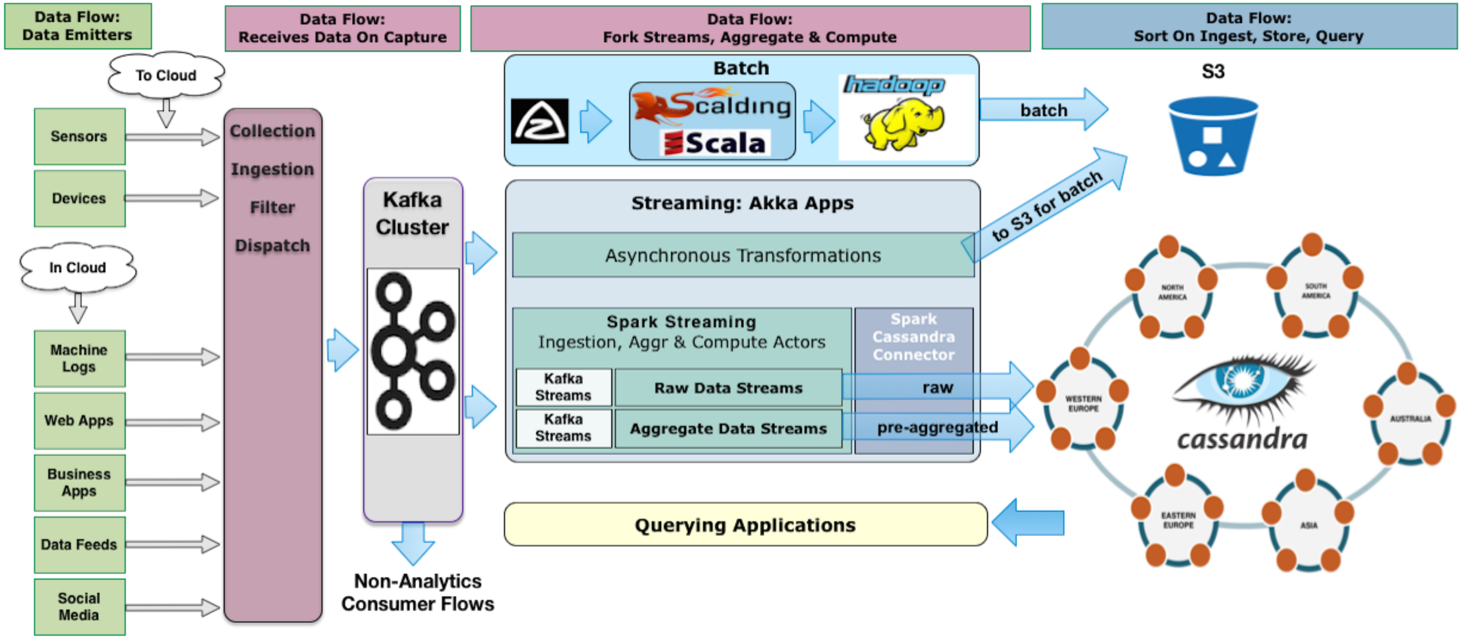
Implementing Is Hard

- Real-time pipeline backed by KV store for updates
- Many moving parts - KV store, real time, batch
- Running similar code in two places
- Still ingesting data to Parquet/HDFS
- Reconcile queries against two different places

Performance Tuning & Monitoring on so many disparate systems

Also Hard

λ: Streaming & Batch Flows



Evolution Or Just Addition?

Or Just Technical Debt?

@helenaedelson

tuplejump

Lambda Architecture

Ingest an immutable sequence of records is captured and fed into

- **a batch system**
- **and a stream processing system**

in parallel

Challenge Assumptions

WAIT, DUAL SYSTEMS?

Which Translates To

- Performing analytical computations & queries in dual systems
- Duplicate Code
- Untyped Code - Strings
- Spaghetti Architecture for Data Flows
- One Busy Network

Why?

- Why support code, machines and running services of two analytics systems?
- Is a separate batch system needed?
- Can we do everything in a streaming system?

YES

- A unified system for streaming and batch
- Real-time processing and reprocessing
  - Code changes
  - Fault tolerance

http://radar.oreilly.com/2014/07/questioning-the-lambda-architecture.html - Jay Kreps

Challenge Assumptions

ANOTHER ASSUMPTION: ETL

Extract, Transform, Load (ETL)

- Extraction of data from one system into another
- Transforming it
- Loading it into another system

ETL

- Each step can introduce errors and risk
- Writing intermediary files
- Parsing and re-parsing plain text
- Tools can cost millions of dollars
- Decreases throughput
- Increased complexity
- Can duplicate data after failover

Extract, Transform, Load (ETL)

"Designing and maintaining the ETL process is often considered one of the most difficult and resource-intensive portions of a data warehouse project."

http://docs.oracle.com/cd/B19306_01/server.102/b14223/ettover.htm

Also unnecessarily redundant and often typeless

And let's duplicate the pattern over all our DataCenters

These are not the solutions you're looking for

REVISITING THE GOAL

Removing The 'E' in ETL

Thanks to technologies like Avro and Protobuf we don't need the "E" in ETL. Instead of text dumps that you need to parse over multiple systems:

E.g Scala and Avro

- A return to strong typing in the big data ecosystem
- Can work with binary data that remains strongly typed

Removing The 'L' in ETL

If data collection is backed by a distributed messaging system (e.g. Kafka) you can do real-time fanout of the ingested data to all consumers. No need to batch "load".

- From there each consumer can do their own transformations

#NoMoreGreekLetterArchitectures

NoETL

Pick Technologies Wisely

Based on your requirements

- Latency
  - Real time / Sub-Second: < 100ms
  - Near real time (low): > 100 ms or a few seconds - a few hours
- Consistency
- Highly Scalable
- Topology-Aware & Multi-Datacenter support
- Partitioning Collaboration - do they play together well

And Remember

- Flows erode
- Entropy happens
- "Everything fails, all the time" - Kyle Kingsbury

REVISITING THE STACK

Stream Processing & Frameworks

+

Spark Streaming

Flink

GearPump

STORM

samza

Strategies

- Partition For Scale & Data Locality
- Replicate For Resiliency
- Share Nothing
- Fault Tolerance
- Asynchrony
- Async Message Passing
- Memory Management
- Data lineage and reprocessing in runtime
- Parallelism
- Elastically Scale
- Isolation
- Location Transparency

Fault Tolerance

- Graceful service degradation
- Data integrity / accuracy under failure
- Resiliency during traffic spikes
- Pipeline congestion / bottlenecks
- Easy to debug and find failure source
- Easy to deploy

My Nerdy Chart

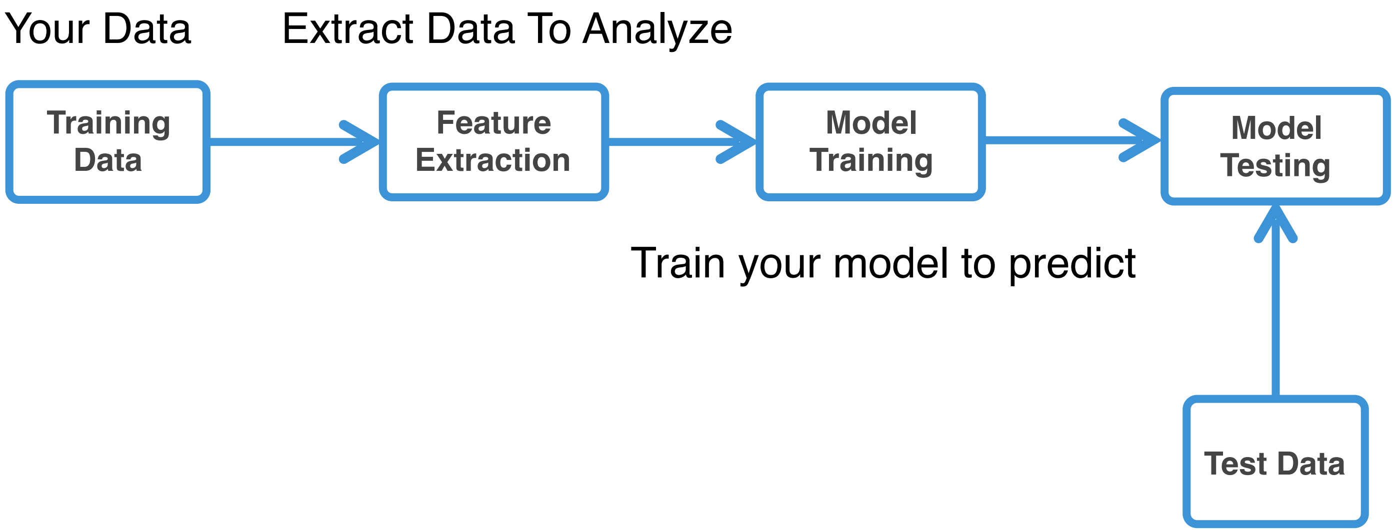
Strategy	Technologies
Scalable Infrastructure / Elastic	Spark, Cassandra, Kafka
Partition For Scale, Network Topology Aware	Cassandra, Spark, Kafka, Akka Cluster
Replicate For Resiliency	Spark,Cassandra, Akka Cluster all hash the node ring
Share Nothing, Masterless	Cassandra, Akka Cluster both Dynamo style
Fault Tolerance / No Single Point of Failure	Spark, Cassandra, Kafka
Replay From Any Point Of Failure	Spark, Cassandra, Kafka, Akka + Akka Persistence
Failure Detection	Cassandra, Spark, Akka, Kafka
Consensus & Gossip	Cassandra & Akka Cluster
Parallelism	Spark, Cassandra, Kafka, Akka
Asynchronous Data Passing	Kafka, Akka, Spark
Fast, Low Latency, Data Locality	Cassandra, Spark, Kafka
Location Transparency	Akka, Spark, Cassandra, Kafka

SMACK

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

Spark Streaming

- One runtime for streaming and batch processing
  - Join streaming and static data sets
- No code duplication
- Easy, flexible data ingestion from disparate sources to disparate sinks
- Easy to reconcile queries against multiple sources
- Easy integration of KV durable storage




```scala
val context = new StreamingContext(conf, Milliseconds(500))
val model = KMeans.train(dataset, ...) // learn offline
val stream = KafkaUtils
  .createStream(ssc, zkQuorum, group,..)
  .map(event => model.predict(event.feature))
```

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

Akka Actors

A distribution and concurrency abstraction

- Compute Isolation
- Behavioral Context Switching
- No Exposed Internal State
- Event-based messaging
- Easy parallelism
- Configurable fault tolerance

High Performance Streaming Built On Akka

- Apache Flink - uses Akka for
  - Actor model and hierarchy, Deathwatch and distributed communication between job and task managers
- GearPump - models the entire streaming system with an actor hierarchy
  - Supervision, Isolation, Concurrency

Apache Cassandra

- Extremely Fast
- Extremely Scalable
- Multi-Region / Multi-Datacenter
- Always On
  - No single point of failure
  - Survive regional outages
- Easy to operate
- Automatic & configurable replication

90% of streaming data at Netflix is stored in Cassandra

STREAM INTEGRATION

KillrWeather

http://github.com/killrweather/killrweather

A reference application showing how to easily integrate streaming and batch data processing with Apache Spark Streaming, Apache Cassandra, Apache Kafka and Akka for fast, streaming computations on time series data in asynchronous event-driven environments.

http://github.com/databricks/reference-apps/tree/master/timeseries/scala/timeseries-weather/src/main/scala/com/databricks/apps/weather

Kafka, Spark Streaming and Cassandra

```
val context = new StreamingContext(conf, Seconds(1))

val stream = KafkaUtils.createDirectStream[Array[Byte],
Array[Byte], DefaultDecoder, DefaultDecoder](
        context, kafkaParams, kafkaTopics)

stream.flatMap(func1).saveToCassandra(ks1,table1)
stream.map(func2).saveToCassandra(ks1,table1)

context.start()
```

Kafka, Spark Streaming, Cassandra & Akka

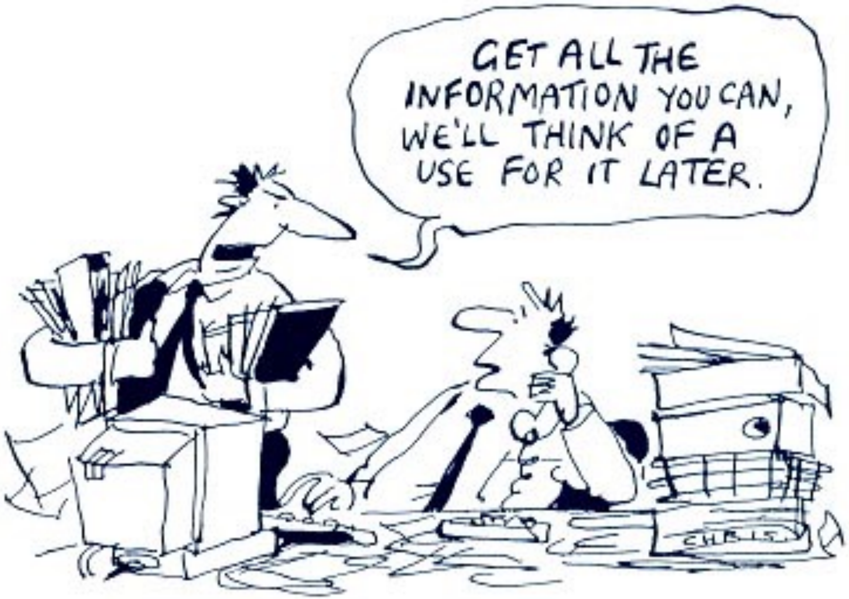
```scala
class KafkaProducerActor[K, V](config: ProducerConfig) extends Actor {

  override val supervisorStrategy =
    OneForOneStrategy(maxNrOfRetries = 10, withinTimeRange = 1.minute) {
      case _: ActorInitializationException   => Stop
      case _: FailedToSendMessageException   => Restart
      case _: ProducerClosedException        => Restart
      case _: NoBrokersForPartitionException => Escalate
      case _: KafkaException                 => Escalate
      case _: Exception                      => Escalate
    }

  private val producer = new KafkaProducer[K, V](producerConfig)

  override def postStop(): Unit = producer.close()

  def receive = {
    case e: KafkaMessageEnvelope[K,V] => producer.send(e)
  }
}
```

Spark Streaming, ML, Kafka & C*

```
val ssc = new StreamingContext(new SparkConf()…, Seconds(5))

val testData = ssc.cassandraTable[String](keyspace,table).map(LabeledPoint.parse)

val trainingStream = KafkaUtils.createStream[K, V, KDecoder, VDecoder](
                       ssc, kafkaParams, topicMap, StorageLevel.MEMORY_ONLY)
                      .map(_._2).map(LabeledPoint.parse)

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

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

  //Making predictions on testData
  model
  .predictOnValues(testData.map(lp => (lp.label, lp.features)))
  .saveToCassandra("ml_keyspace", "predictions")
```

SMACK

STREAM INTEGRATION: DATA LOCALITY & TIMESERIES


GET ALL THE INFORMATION YOU CAN, WE'LL THINK OF A USE FOR IT LATER.

```scala
class KafkaStreamingActor(params: Map[String, String], ssc: StreamingContext)
  extends AggregationActor(settings: Settings) {
  import settings._

  val stream = KafkaUtils.createStream(
    ssc, params, Map(KafkaTopicRaw -> 1), StorageLevel.DISK_ONLY_2)
    .map(_._2.split(","))
    .map(RawWeatherData(_))

  stream.saveToCassandra(CassandraKeyspace, CassandraTableRaw)

  stream
    .map(hour => (hour.wsid, hour.year, hour.month, hour.day, hour.oneHourPrecip))
    .saveToCassandra(CassandraKeyspace, CassandraTableDailyPrecip)

}
```

Kafka, Spark Streaming, Cassandra & Akka

```scala
class KafkaStreamingActor(params: Map[String, String], ssc: StreamingContext)
  extends AggregationActor(settings: Settings) {
  import settings._

  val stream = KafkaUtils.createStream(
    ssc, params, Map(KafkaTopicRaw -> 1), StorageLevel.DISK_ONLY_2)
    .map(_._2.split(","))
    .map(RawWeatherData(_))

  stream.saveToCassandra(CassandraKeyspace, CassandraTableRaw)

  stream
    .map(hour => (hour.wsid, hour.year, hour.month, hour.day, hour.oneHourPrecip))
    .saveToCassandra(CassandraKeyspace, CassandraTableDailyPrecip)

}
```

Now we can replay

- On failure
- Reprocessing on code changes
- Future computation...

```scala
class KafkaStreamingActor(params: Map[String, String], ssc: StreamingContext)
  extends AggregationActor(settings: Settings) {
  import settings._

  val stream = KafkaUtils.createStream(
    ssc, params, Map(KafkaTopicRaw -> 1), StorageLevel.DISK_ONLY_2)
    .map(_._2.split(","))
    .map(RawWeatherData(_))

  stream.saveToCassandra(CassandraKeyspace, CassandraTableRaw)

  stream
    .map(hour => (hour.wsid, hour.year, hour.month, hour.day, hour.oneHourPrecip))
    .saveToCassandra(CassandraKeyspace, CassandraTableDailyPrecip)

}
```

Here we are **pre-aggregating** to a table **for fast querying later** -
in other secondary stream aggregation computations and scheduled computing

Data Model (simplified)

```
CREATE TABLE weather.raw_data (
  wsid text, year int, month int, day int, hour int,
  temperature double, dewpoint double, pressure double,
  wind_direction int, wind_speed double, one_hour_precip
  PRIMARY KEY ((wsid), year, month, day, hour)
) WITH CLUSTERING ORDER BY (year DESC, month DESC, day DESC, hour DESC);


CREATE TABLE daily_aggregate_precip (
  wsid text,
  year int,
  month int,
  day int,
  precipitation counter,
  PRIMARY KEY ((wsid), year, month, day)
) WITH CLUSTERING ORDER BY (year DESC, month DESC, day DESC);
```

```scala
class KafkaStreamingActor(params: Map[String, String], ssc: StreamingContext)
  extends AggregationActor(settings: Settings) {
  import settings._

  val stream = KafkaUtils.createStream(
    ssc, params, Map(KafkaTopicRaw -> 1), StorageLevel.DISK_ONLY_2)
    .map(_._2.split(","))
    .map(RawWeatherData(_))

  stream.saveToCassandra(CassandraKeyspace, CassandraTableRaw)

  stream
    .map(hour => (hour.wsid, hour.year, hour.month, hour.day, hour.oneHourPrecip))
    .saveToCassandra(CassandraKeyspace, CassandraTableDailyPrecip)

}
```

Gets the partition key: Data Locality
Spark C* Connector feeds this to Spark

Cassandra Counter column in our schema, no expensive `reduceByKey` needed. Simply let C* do it: not expensive and fast.

The Thing About S3

"Amazon S3 is a simple key-value store" - docs.aws.amazon.com/AmazonS3/latest/dev/UsingObjects.html

- Keys 2015/05/01 and 2015/05/02 do not live in the "same place"
- You can roll your own with AmazonS3Client and do the heavy lifting yourself and throw that data into Spark

Timeseries Data

```
CREATE TABLE weather.raw_data (
   wsid text, year int, month int, day int, hour int,
   temperature double, dewpoint double, pressure double,
   wind_direction int, wind_speed double, one_hour_precip
   PRIMARY KEY ((wsid), year, month, day, hour)
) WITH CLUSTERING ORDER BY (year DESC, month DESC, day DESC, hour DESC);
```

C* Clustering Columns

Writes by most recent
Reads return most recent first

Cassandra will automatically sort by most recent for both write and read

Record Every Event In The Order In Which It Happened, Per URL

```
CREATE TABLE IF NOT EXISTS requests_ks.timeline (
  timesegment bigint, url text, t_uuid timeuuid, method text, headers map <text, text>, body text,
  PRIMARY KEY ((url, timesegment) , t_uuid)
);
```

timesegment protects from writing unbounded partitions.

timeuuid protects from simultaneous events over-writing one another.

```
val multipleStreams = for (i <- numstreams) {
  streamingContext.receiverStream[HttpRequest](new HttpReceiver(port))
}

streamingContext.union(multipleStreams)
                .map { httpRequest => TimelineRequestEvent(httpRequest)}
                .saveToCassandra("requests_ks", "timeline")
```

Cassandra & Spark Streaming: *Data Locality For Free®*

```
val stream = KafkaUtils.createDirectStream(...)
  .map(_._2.split(","))
  .map(RawWeatherData(_))
```

Replay and Reprocess - Any Time
Data is on the nodes doing the querying
- Spark C* Connector - Partitions

```
stream.saveToCassandra(CassandraKeyspace, CassandraTableRaw)

stream
    .map(hour => (hour.id, hour.year, hour.month, hour.day, hour.oneHourPrecip))
    .saveToCassandra(CassandraKeyspace, CassandraTableDailyPrecip)
```

- Timeseries data with Data Locality
- Co-located Spark + Cassandra nodes
- S3 does not give you

Compute Isolation: Actor

```scala
class PrecipitationActor(ssc: StreamingContext, settings: Settings) extends AggregationActor {
  import akka.pattern.pipe

  def receive : Actor.Receive = {
    case GetTopKPrecipitation(wsid, year, k) => topK(wsid, year, k, sender)
  }

  /** Returns the 10 highest temps for any station in the `year`. */
  def topK(wsid: String, year: Int, k: Int, requester: ActorRef): Unit = {
    val toTopK = (aggregate: Seq[Double]) => TopKPrecipitation(wsid, year,
      ssc.sparkContext.parallelize(aggregate).top(k).toSeq)

    ssc.cassandraTable[Double](keyspace, dailytable)
      .select("precipitation")
      .where("wsid = ? AND year = ?", wsid, year)
      .collectAsync().map(toTopK) pipeTo requester
  }
}
```

Queries pre-aggregated tables from the stream

Efficient Batch Analysis

```
class TemperatureActor(sc: SparkContext, settings: Settings) 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](keyspace, daily_temperature_aggr)
      .where("wsid = ? AND year = ? AND month = ?", e.wsid, e.year, e.month)
      .collectAsync()
      .map(MonthlyTemperature(_, e.wsid, e.year, e.month)) pipeTo requester

}
```

C* data is automatically sorted by most recent - due to our data model.

Additional Spark or collection sort not needed.

Simplification

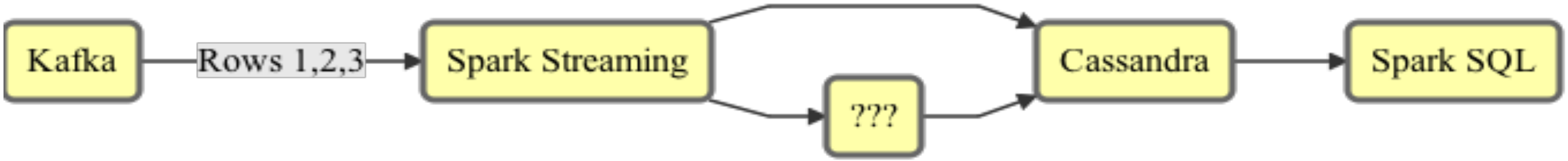
A NEW APPROACH

Everything On The Streaming Platform

Reprocessing

- Start a new stream job to re-process from the beginning
- Save re-processed data as a version table
- Application should then read from new version table
- Stop old version of the job, and delete the old table

One Pipeline For Fast & Big Data



- How do I make the SMACK stack work for ML, Ad-Hoc + Fast Data?
- How do I combine Spark Streaming + Ad Hoc and have good performance?

@helenaedelson

tuplejump

FiloDB

Designed to ingest streaming data, including machine, event, and time-series data, and run very fast analytical queries over them.

- Distributed, versioned, columnar analytics database
- Built for fast **streaming analytics** & OLAP
- Currently based on Apache Cassandra & Spark
- github.com/tuplejump/FiloDB

Breakthrough Performance For Analytical Queries

- Queries run in parallel in Spark for scale-out ad-hoc analysis
- Fast for interactive data science and ad hoc queries
- Up to 200x Faster Queries for Spark on Cassandra 2.x
- Parquet Performance with Cassandra Flexibility
- Increased performance ceiling coming

Versioning & Why It Matters

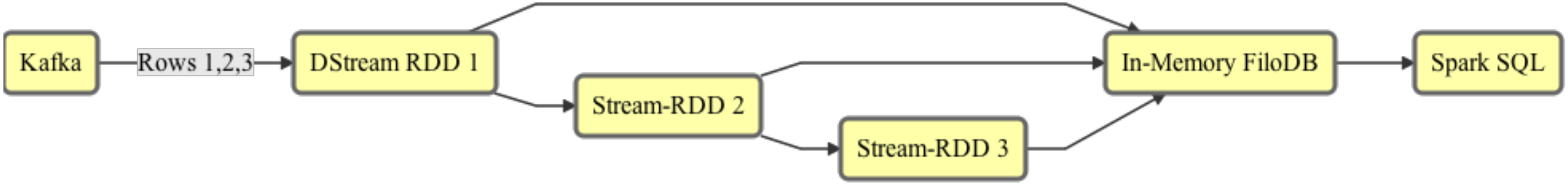
- Immutability
- Databases: let's mutate one giant piece of state in place
  - Basically hasn't changed since 1970's!
- With Big Data and streaming, incremental processing is increasingly important

FiloDB Versioning

FiloDB is built on functional principles and lets you version and layer changes

- Incrementally add a column or a few rows as a new version
- Add changes as new versions, don't mutate!
- Writes are *idempotent* - exactly once ingestion
- Easily control what versions to query
- Roll back changes inexpensively
- Stream out new versions as continuous queries :)

No Cassandra? Keep All In Memory



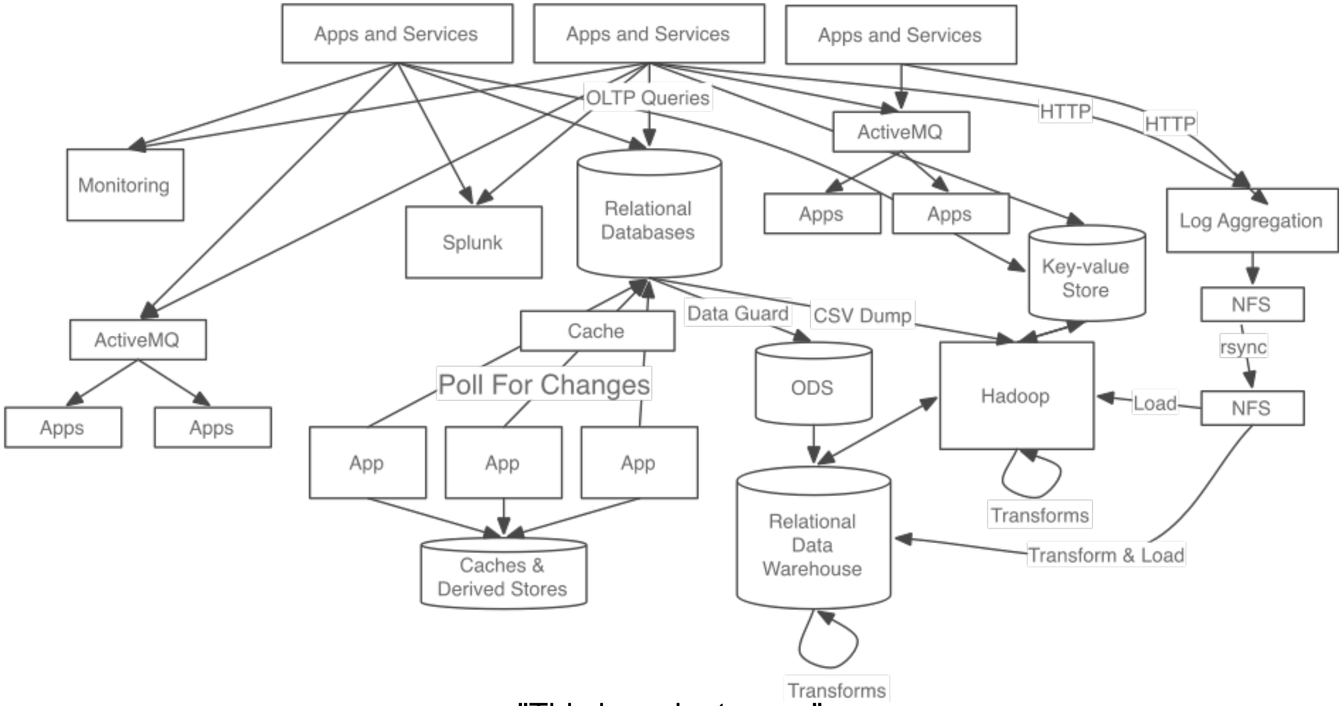
- Unlike RDDs and DataFrames, FiloDB can ingest new data, and still be fast
- Unlike RDDs, FiloDB can filter in multiple ways, no need for entire table scan

Spark Streaming to FiloDB

```
val ratingsStream = KafkaUtils.createDirectStream[String, String, StringDecoder, StringDecoder](
ssc, kafkaParams, topics)

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

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

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

```
.dfWithBatchTime.write.format("org.apache.spark.sql.cassandra")
```

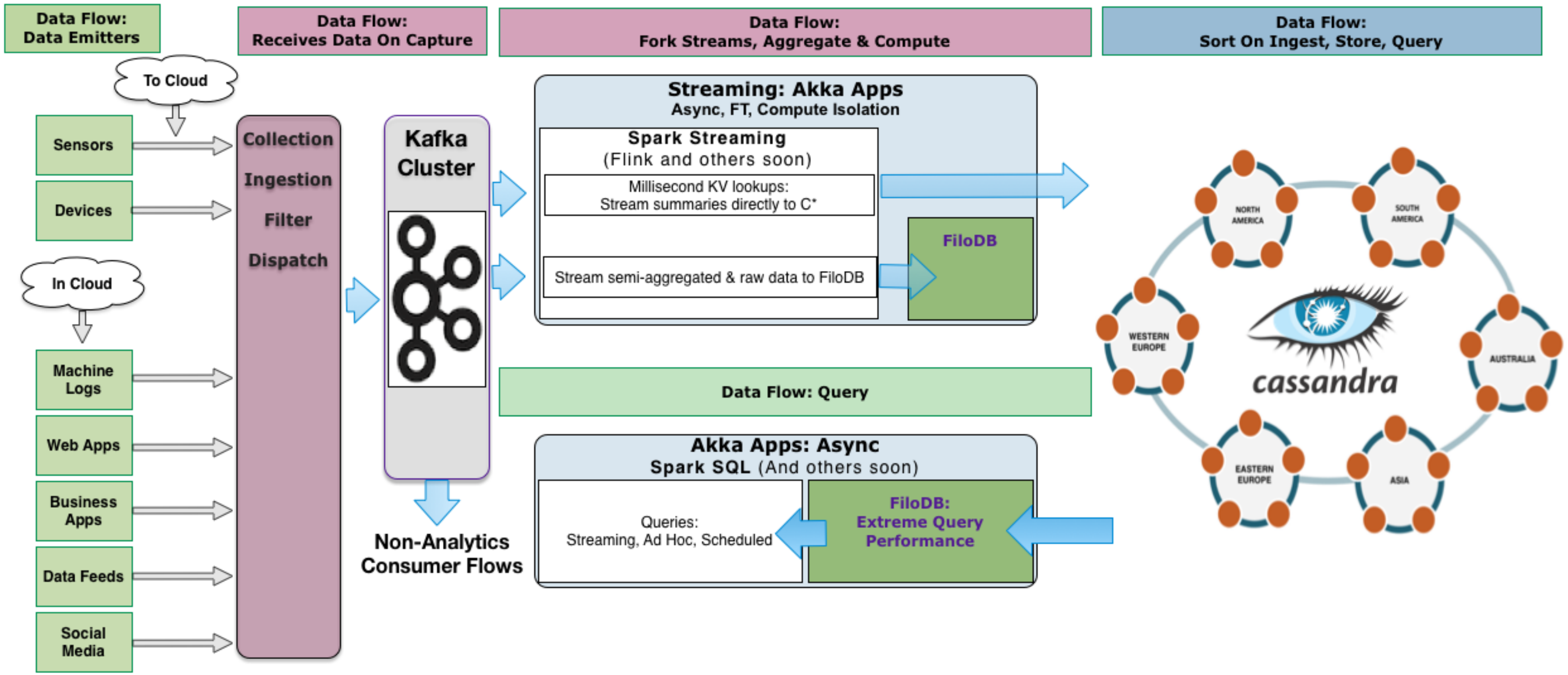
Architectyr?

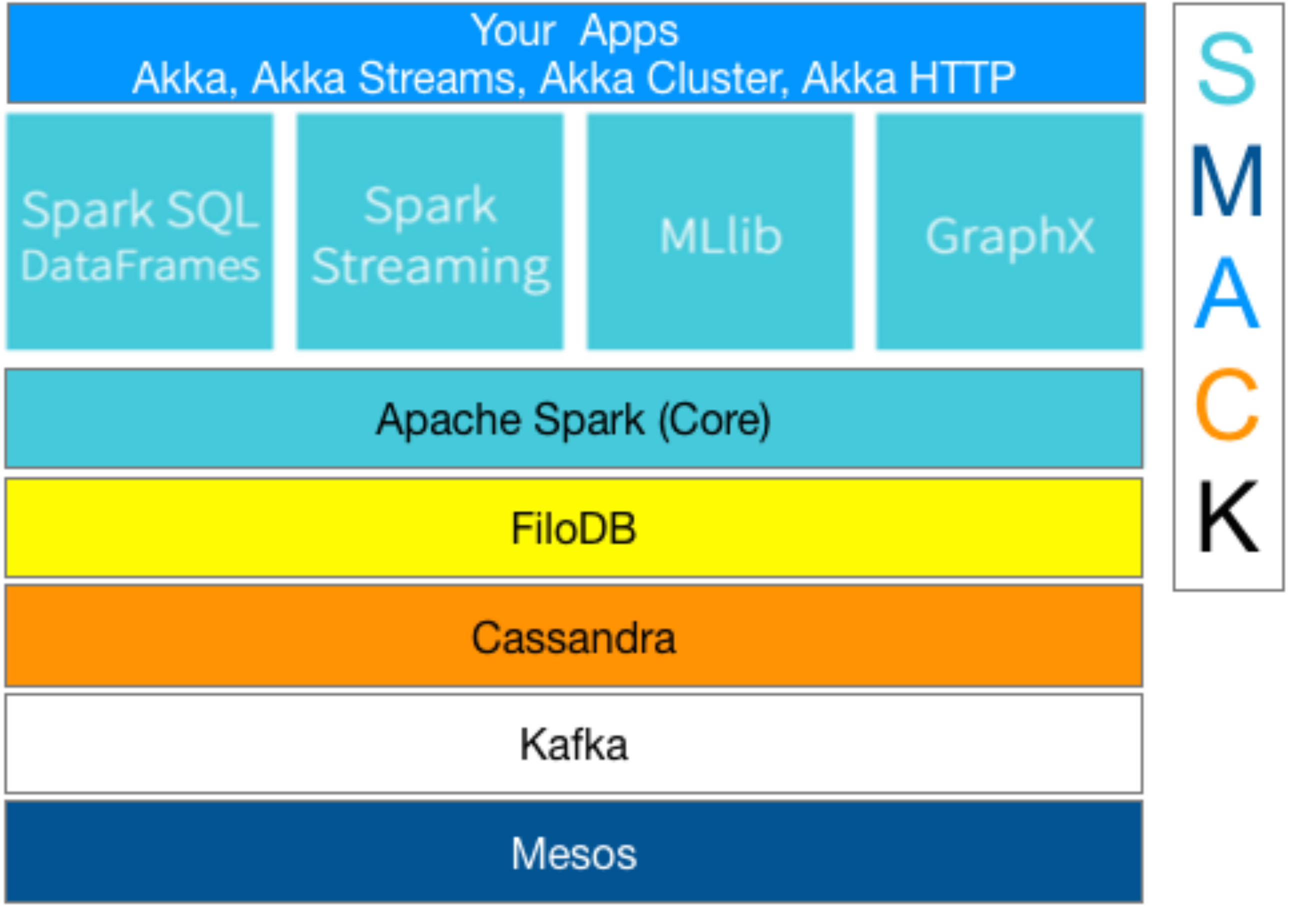


"This is a giant mess"
- Going Real-time - Data Collection and Stream Processing with Apache Kafka, Jay Kreps

@helenaedelson

tuplejump


Data Flow: Data Emitters
Data Flow: Receives Data On Capture
Data Flow: Fork Streams, Aggregate & Compute
Data Flow: Sort On Ingest, Store, Query
To Cloud
Sensors
Devices
In Cloud
Machine Logs
Web Apps
Business Apps
Data Feeds
Social Media
Collection
Ingestion
Filter
Dispatch
Kafka Cluster
Non-Analytics Consumer Flows
Streaming: Akka Apps
Async, FT, Compute Isolation
Spark Streaming
(Flink and others soon)
Millisecond KV lookups:
Stream summaries directly to C*
Stream semi-aggregated & raw data to FiloDB
FiloDB
Data Flow: Query
Akka Apps: Async
Spark SQL (And others soon)
Queries:
Streaming, Ad Hoc, Scheduled
FiloDB:
Extreme Query Performance
NORTH AMERICA
SOUTH AMERICA
WESTERN EUROPE
AUSTRALIA
EASTERN EUROPE
ASIA
cassandra


Your Apps
Akka, Akka Streams, Akka Cluster, Akka HTTP
Spark SQL
DataFrames
Spark
Streaming
MLlib
GraphX
Apache Spark (Core)
FiloDB
Cassandra
Kafka
Mesos
S
M
A
C
K

THANKS!

@helenaedelson

github.com/helena

helena@tuplejump.com

slideshare.net/helenaedelson

Designers
In this role we come in early to augument your product management team. We help you envision and innovate your next-gen data solutions pushing the limits of what's possible.

Builders
Every team can leverage experience. We will work with your team to provide the experience kickstart required. We will mentor them and deliver your data solutions as well as the ability to handle its progress.

Architects
Know what you want, wondering how to get there? We come in as archtiects and work to identify the required technology stack and deliver the architecture to realize your vision.

Firefighters
Sometimes it is too late before you identify the problems. But its never too late for us to step in and clear out the fire. In collaboration with your team we will out the problem and deliver the solution for it too.

tuplejump

Your Software Development Partners

---

www.tuplejump.com
info@tuplejump.com