Strata+ Hadoop WORLD

PRESENTED BY

O'REILLY cloudera

strataconf.com

#StrataHadoop

Hadoop Application Architectures: Fraud Detection

Strata + Hadoop World, London 2016

tiny.cloudera.com/app-arch-london

tiny.cloudera.com/app-arch-questions

Gwen Shapira | @gwenshap

Jonathan Seidman | @jseidman

Ted Malaska | @ted_malaska

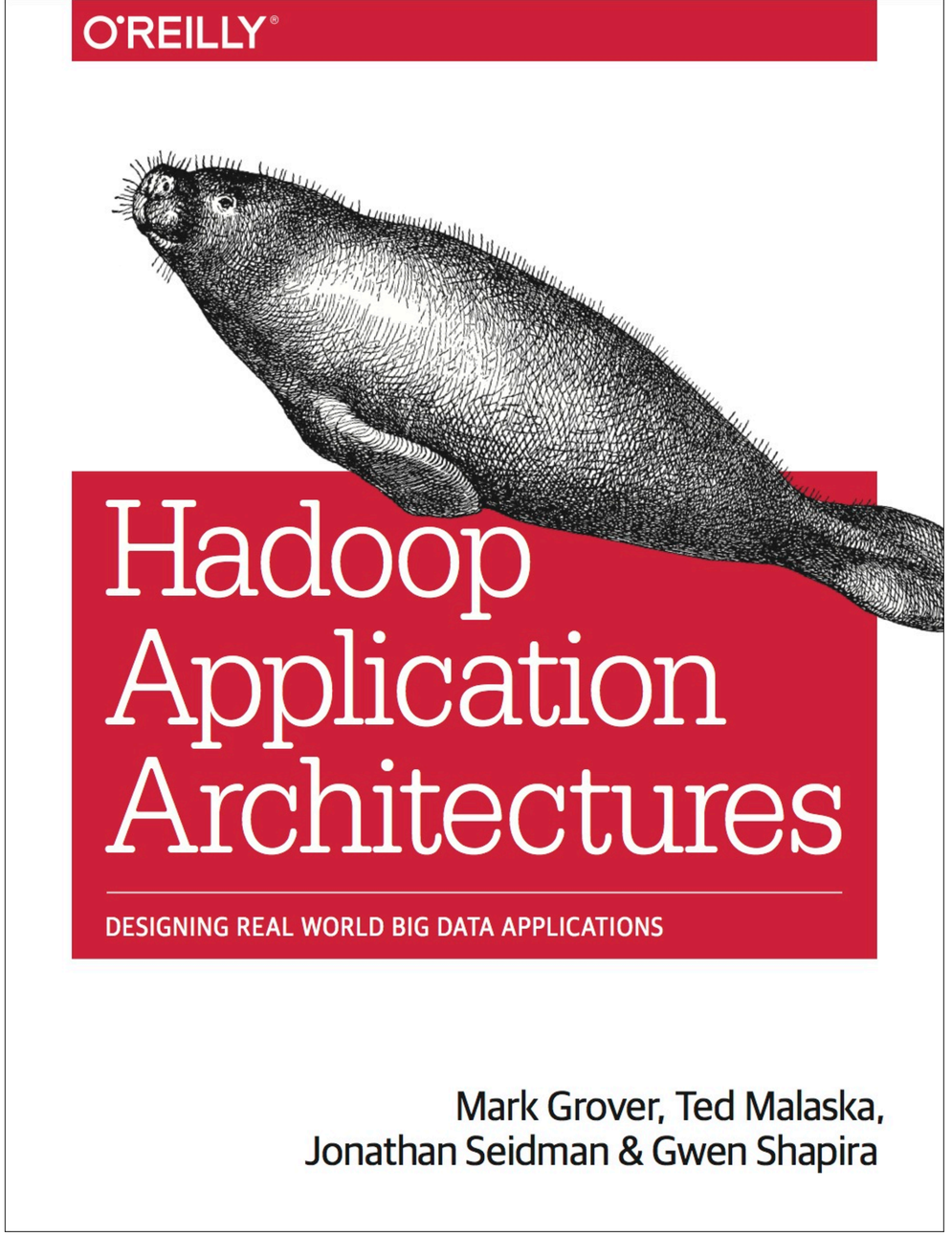
Mark Grover | @mark_grover

Logistics

- Break at 10:30 – 11:00 AM
- Questions at the end of each section
- Slides at tiny.cloudera.com/app-arch-london
- Code at https://github.com/hadooparchitecturebook/fraud-detection-tutorial

About the book

- @hadooparchbook
- hadooparchitecturebook.com
- github.com/hadooparchitecturebook
- slideshare.com/hadooparchbook



About the presenters

Ted Malaska

- Principal Solutions Architect at Cloudera
- Previously, lead architect at FINRA
- Contributor to Apache Hadoop, HBase, Flume, Avro, Pig and Spark

Jonathan Seidman

- Senior Solutions Architect/ Partner Enablement at Cloudera
- Contributor to Apache Sqoop.
- Previously, Technical Lead on the big data team at Orbitz, co-founder of the Chicago Hadoop User Group and Chicago Big Data

About the presenters

Gwen Shapira

- System Architect at Confluent
- Committer on Apache Kafka, Sqoop
- Contributor to Apache Flume

Mark Grover

- Software Engineer on Spark at Cloudera
- Committer on Apache Bigtop, PMC member on Apache Sentry(incubating)
- Contributor to Apache Spark, Hadoop, Hive, Sqoop, Pig, Flume

Strata+ Hadoop WORLD

PRESENTED BY

O'REILLY cloudera

strataconf.com

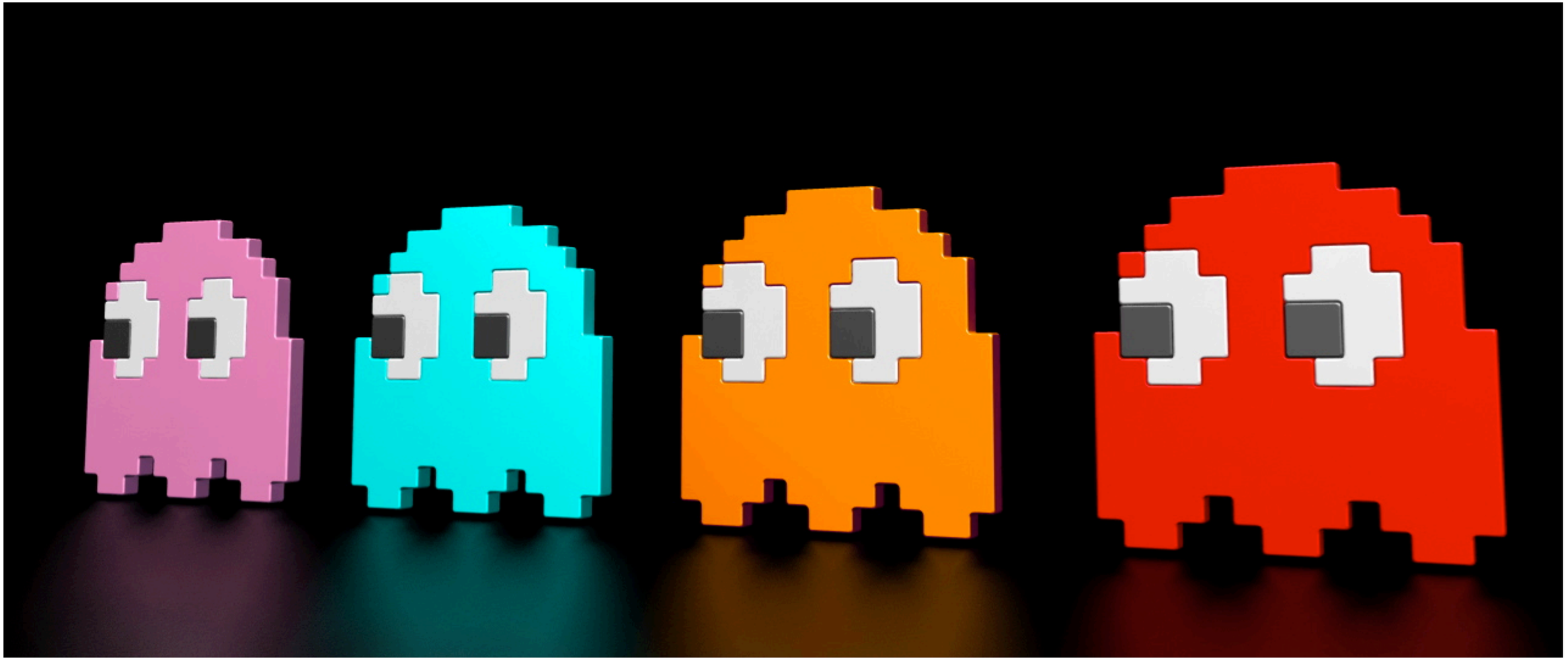
#StrataHadoop

Case Study Overview

Fraud Detection

Credit Card Transaction Fraud

Video Game Strategy



Health Insurance Fraud

Network anomaly detection

Strata+ Hadoop WORLD

PRESENTED BY

O'REILLY cloudera

strataconf.com

#StrataHadoop

Ok, enough with the use-cases

Can we move on?

How Do We React

- Human Brain at Tennis
  - *Muscle Memory*
  - *Fast Thought*
  - *Slow Thought*

Back to network anomaly detection

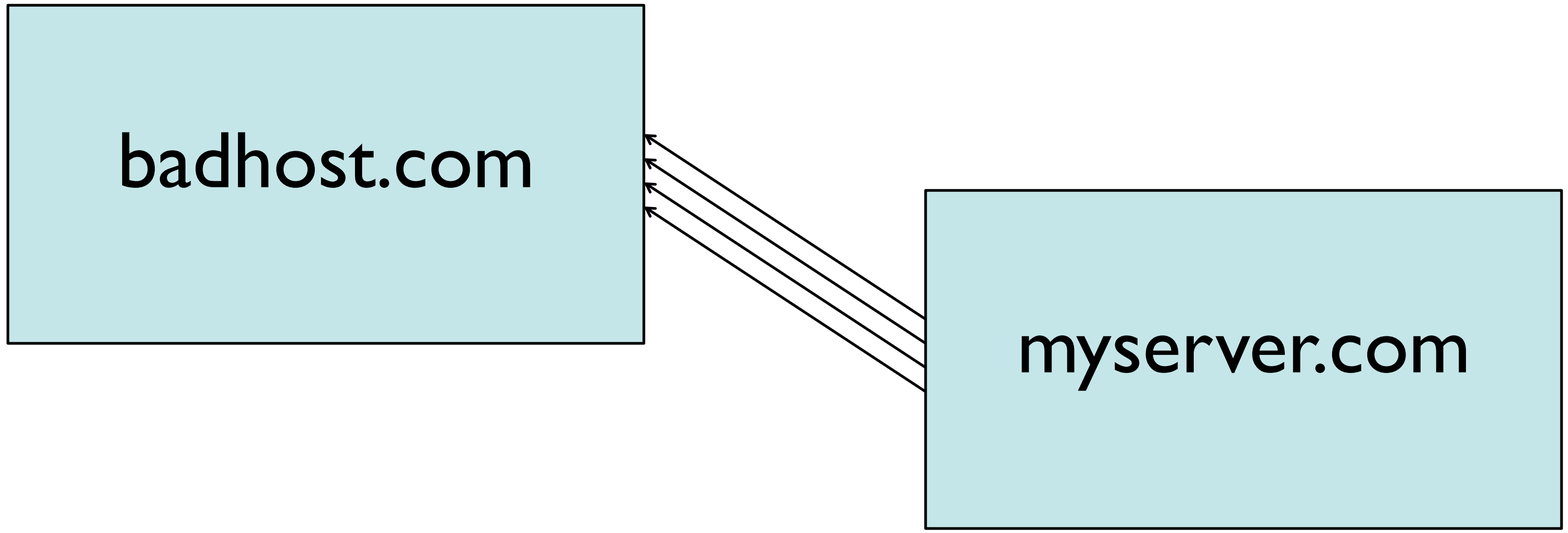
- Muscle memory – for quick action (e.g. reject events)
  - Real time alerts
  - Real time dashboard
- Platform for automated learning and improvement
  - Real time (fast thought)
  - Batch (slow thought)
- Slow thought
  - Audit trail and analytics for human feedback

Use case

Network Anomaly Detection

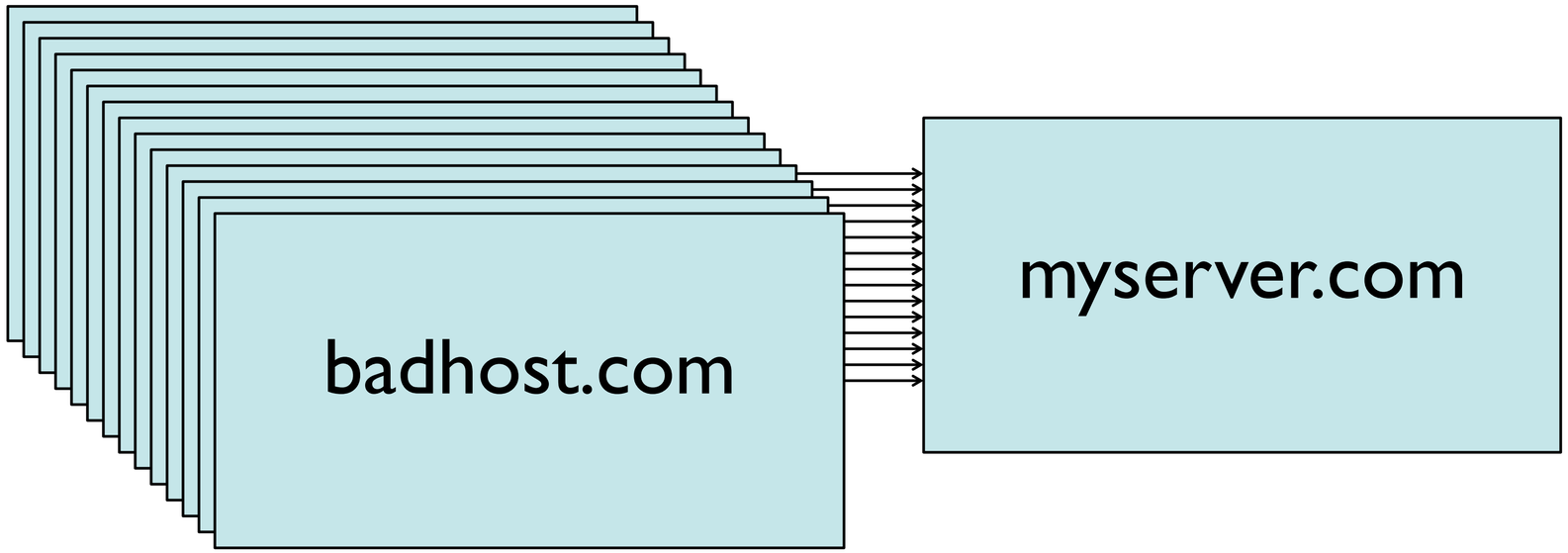
Use Case – Network Anomaly Detection

- “Beaconing” – Malware “phoning home”.



Use Case – Network Anomaly Detection

- Distributed Denial of Service attacks.



Use Case – Network Anomaly Detection

- Network intrusion attempts – attempted or successful break-ins to a network.

Use Case – Network Anomaly Detection

All of these require the ability to detect deviations from known patterns.

Other Use Cases

Could be used for any system requiring detection of anomalous events:

Credit card transactions

Market transactions

Internet of Things

Etc…

Input Data – NetFlow

- Standard protocol defining a format for capturing network traffic flow through routers.
- Captures network data as *flow records*. These flow records provide information on source and destination IP addresses, traffic volumes, ports, etc.
- We can use this data to detect normal and anomalous patterns in network traffic.

Strata+ Hadoop WORLD

PRESENTED BY

O'REILLY cloudera

strataconf.com

#StrataHadoop

Why is Hadoop a great fit?

Hadoop

- In traditional sense
  - HDFS (append-only file system)
  - MapReduce (really slow but robust execution engine)
- Is not a great fit

Why is Hadoop a Great Fit?

- But the Hadoop ecosystem is!
- More than just MapReduce + YARN + HDFS

Volume

- Have to maintain millions of device profiles
- Retain flow history
- Make and keep track of automated rule changes

Velocity

- Events arriving concurrently and at high velocity
- Make decisions in real-time

Variety

- Maintain simple counters in profile (e.g. throughput thresholds, purchase thresholds)
- Iris or finger prints that need to be matched

Challenges of Hadoop Implementation

Challenges of Hadoop Implementation

Challenges - Architectural Considerations

- Storing state (for real-time decisions):
  - Local Caching? Distributed caching (e.g. Memcached)? Distributed storage (e.g. HBase)?
- Profile Storage
  - HDFS? HBase?
- Ingestion frameworks (for events):
  - Flume? Kafka? Sqoop?
- Processing engines (for background processing):
  - Storm? Spark? Trident? Flume interceptors? Kafka Streams?
- Data Modeling
- Orchestration
  - Do we still need it for real-time systems?

Case Study Requirements

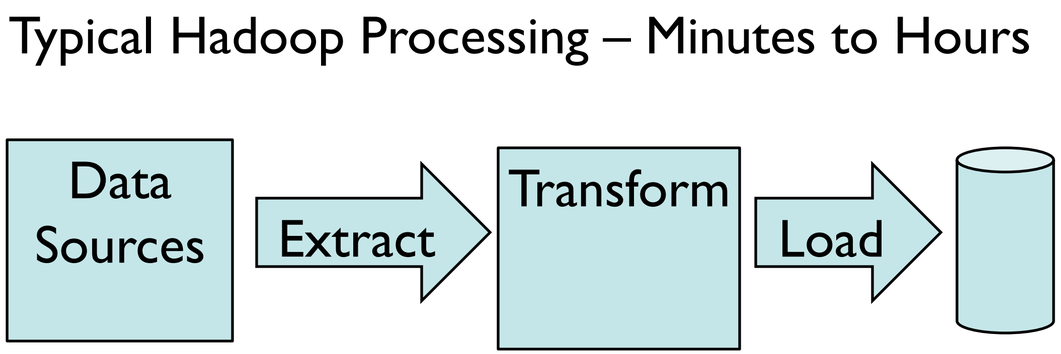
Overview

But First…

Real-Time? Near Real-Time? Stream Processing?

Some Definitions

- Real-time – well, not really. Most of what we're talking about here is…
- Near real-time:
  - Stream processing – continual processing of incoming data.
  - Alerting – immediate (well, as fast as possible) responses to incoming events. Getting closer to real-time.

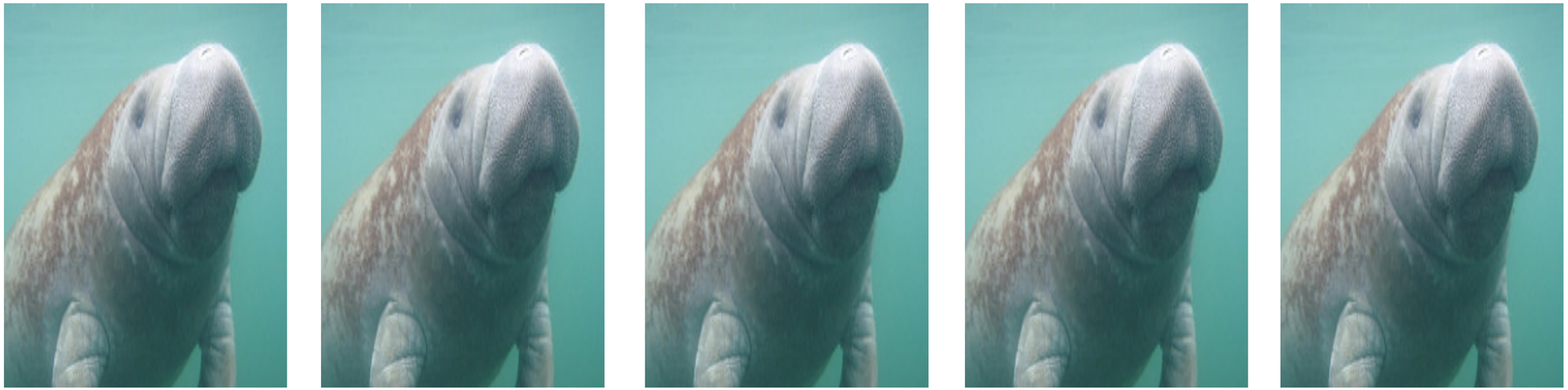


Requirements – Storage/Modeling

- Need long term storage for large volumes of profile, event, transaction, etc. data.

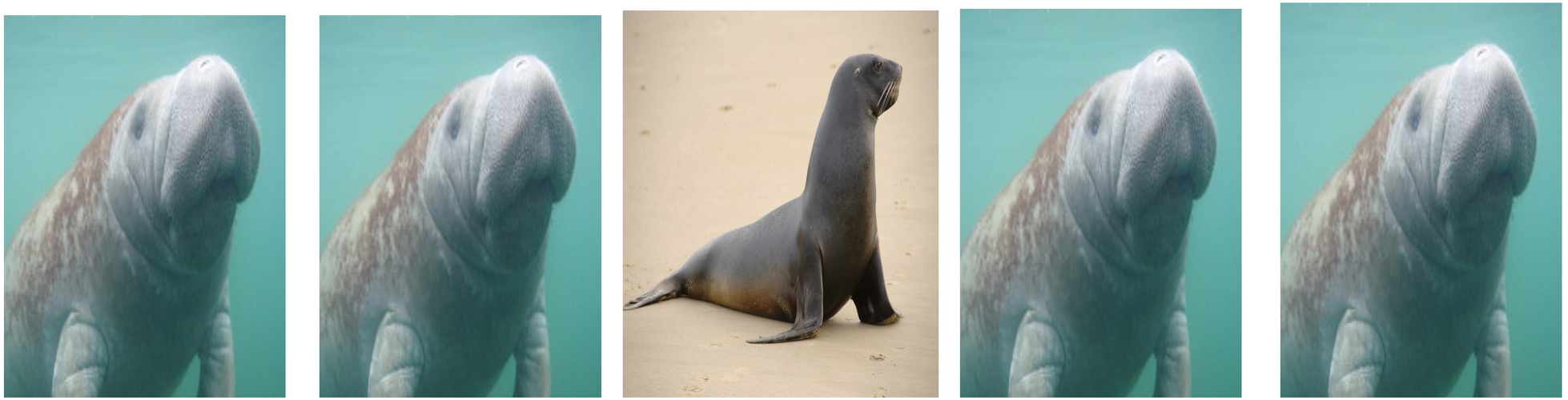
Requirements – Storage/Modeling

- Need to be able to quickly retrieve specific profiles and respond quickly to incoming events.



Requirements – Storage/Modeling

- Need to store sufficient data to analyze whether or not fraud is present.



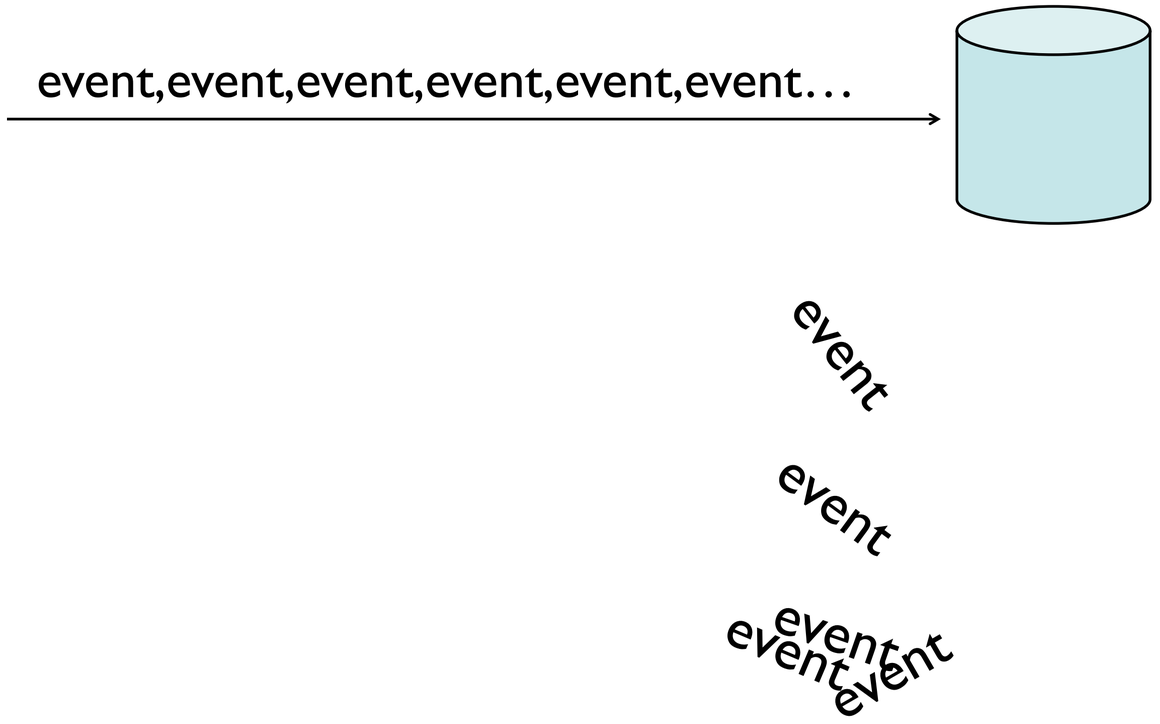
Requirements – Alerts

- Need to be able to respond to incoming events quickly (milliseconds).
- Need high throughput and low latency.
- Processing requirements are minimal.
- Two types of alerts: “atomic”, and “atomic with context”

Requirements – Ingestion

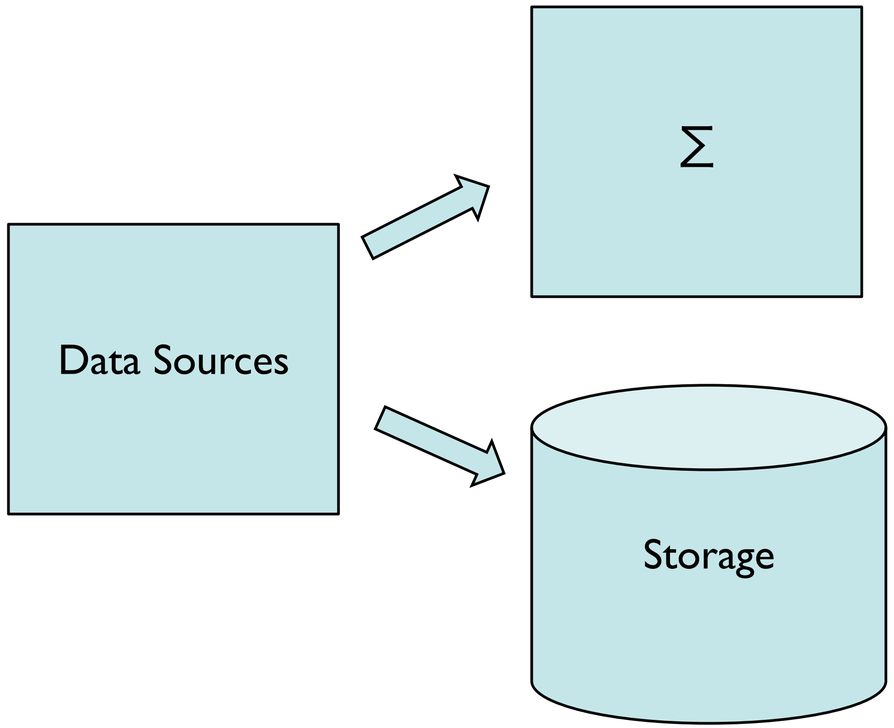
Requirements – Ingestion

- Reliable – we don't want to lose events.



Requirements – Ingestion

- Support for multiple targets.



Requirements – Ingestion

- Needs to support high throughput of large volumes of events.

Requirements – Stream Processing

- A few seconds to minutes response times.
- Need to be able to detect trends, thresholds, etc. across profiles.
- Quick processing more important than 100% accuracy.

Requirements – Batch Processing

- Non real-time, “off-line”, exploratory processing and reporting.
- Data needs to be available for analysts and users to analyze with tools of choice – SQL, MapReduce, etc.
- Results need to be able to feed back into NRT processing to adjust rules, etc.

Strata+ Hadoop WORLD

PRESENTED BY

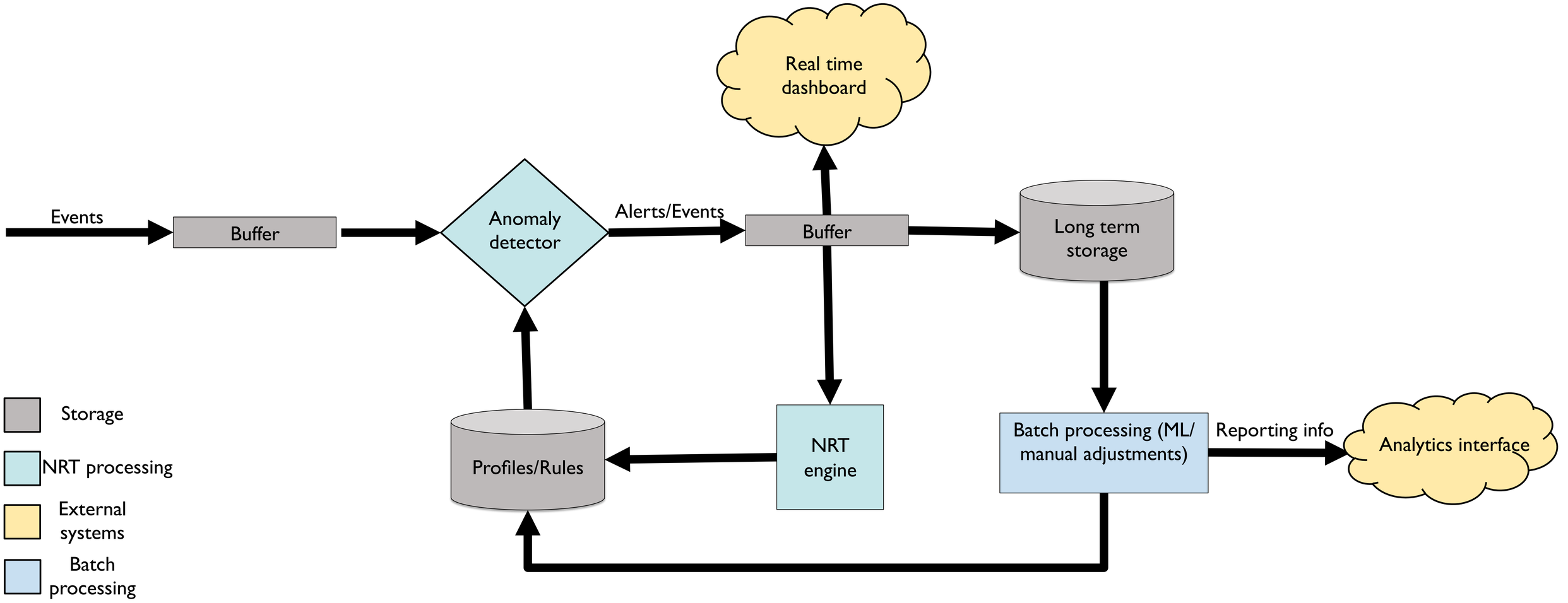
O'REILLY cloudera

strataconf.com

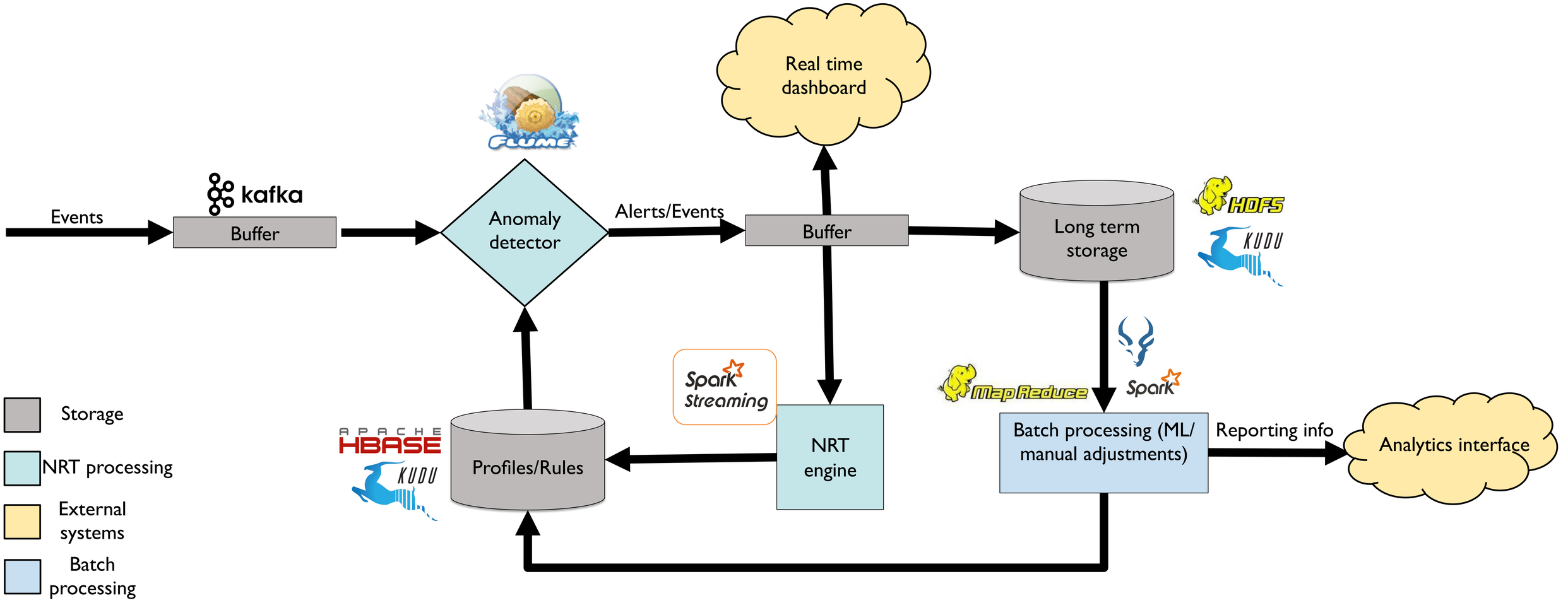
#StrataHadoop

High level architecture

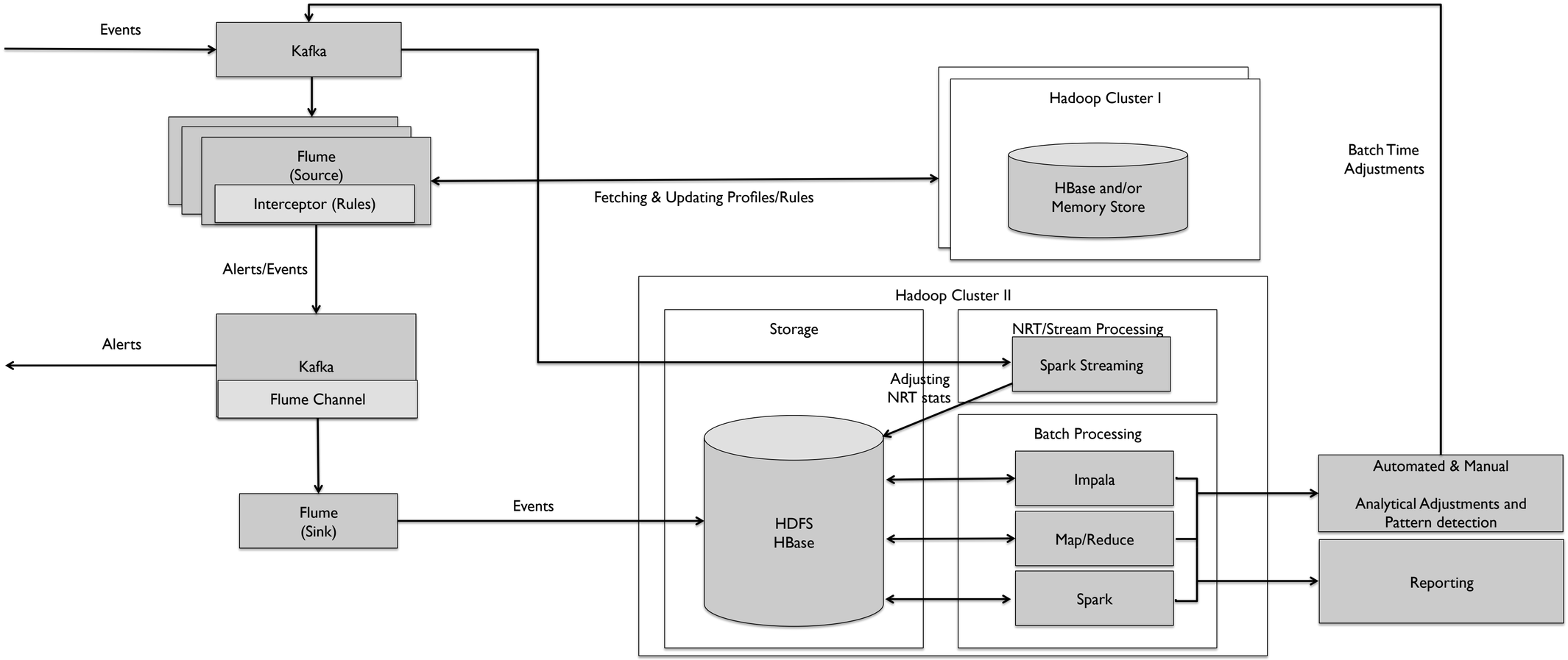
High level architecture



High level architecture



Full Architecture



Storage Layer

Considerations

Storage Layer Considerations

- Two likely choices for long-term storage of data:

Data Storage Layer Choices

- Stores data directly as files
- Fast scans
- Poor random reads/writes

- Stores data as Hfiles on HDFS
- Slow scans
- Fast random reads/writes

Storage Considerations

- Batch processing and reporting requires access to all raw and processed data.
- Stream processing and alerting requires quickly fetching and saving profiles.

Data Storage Layer Choices

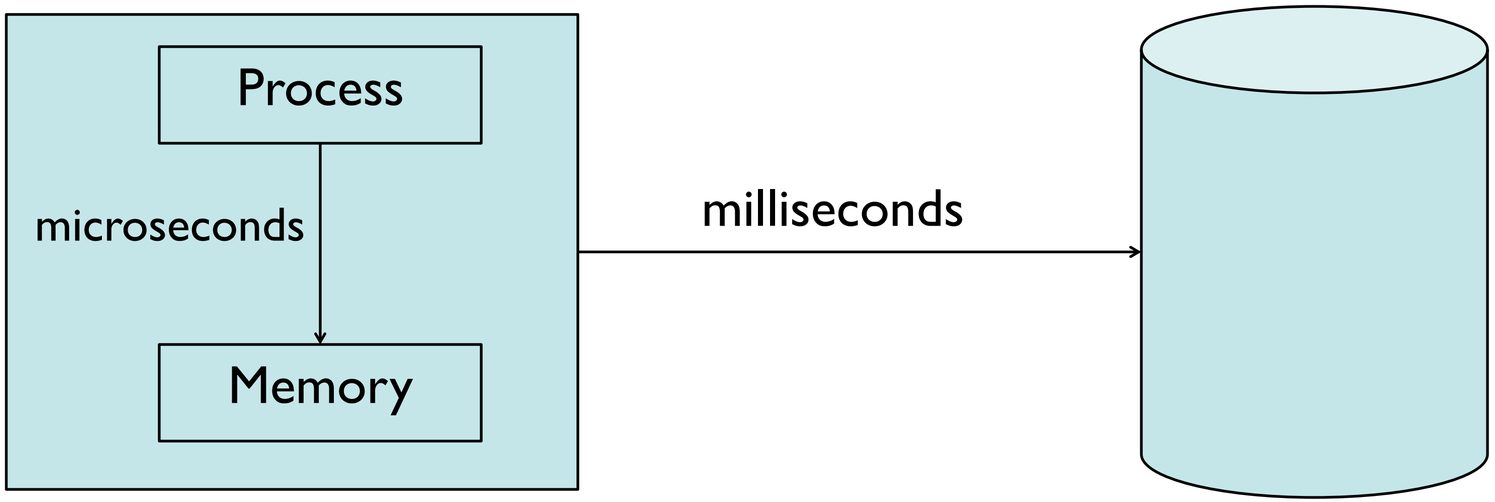
- For ingesting raw data.
- Batch processing, also some stream processing.

- For reading/writing profiles.
- Fast random reads and writes facilitates quick access to large number of profiles.

But…

Is HBase fast enough for fetching profiles?

What About Caching?



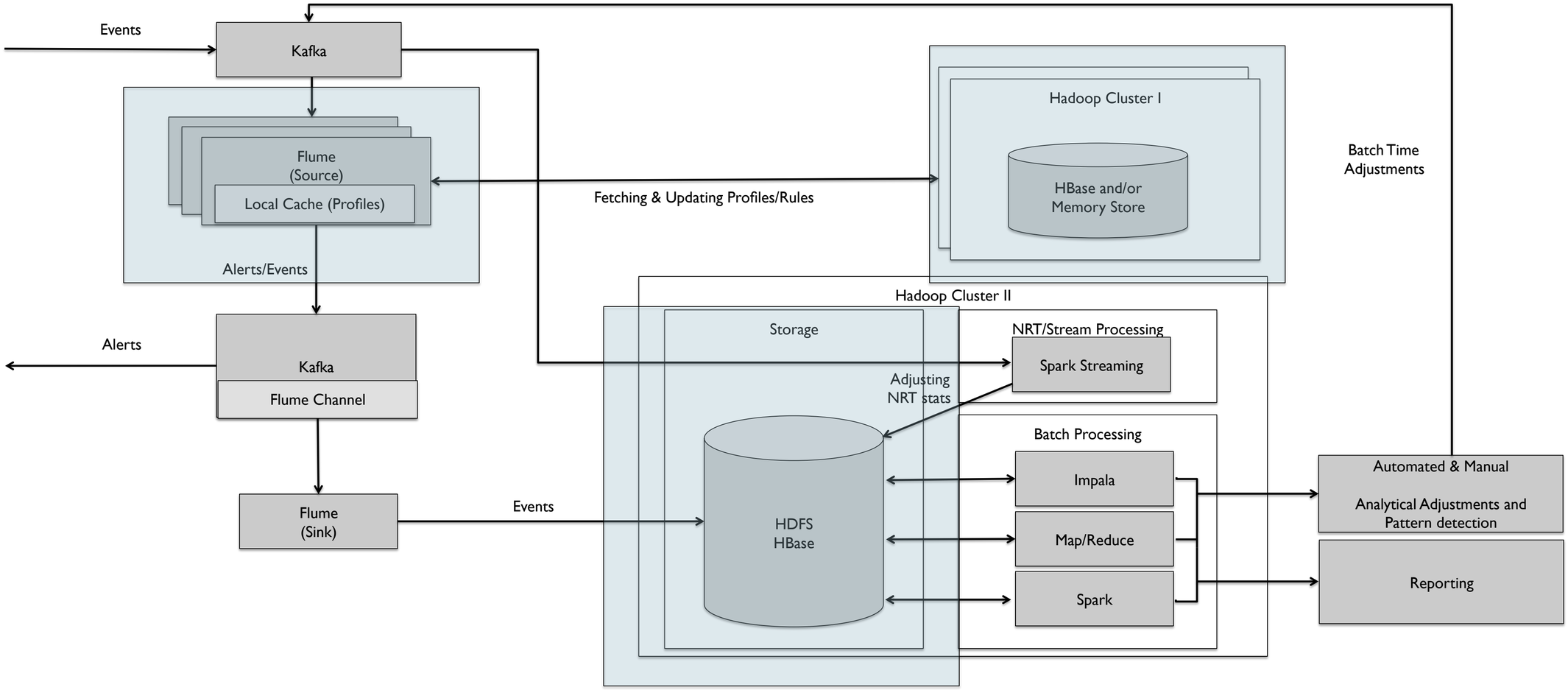
Caching Options

- Local in-memory cache (on-heap, off-heap).
- Remote cache
  - Distributed cache (Memcached, Oracle Coherence, etc.)
  - HBase BlockCache

Caching – HBase BlockCache

- Allows us to keep recently used data blocks in memory.
- Note that this will require recent versions of HBase and specific configurations on our cluster.

Our Storage Choices



Strata+ Hadoop WORLD

PRESENTED BY

O'REILLY cloudera

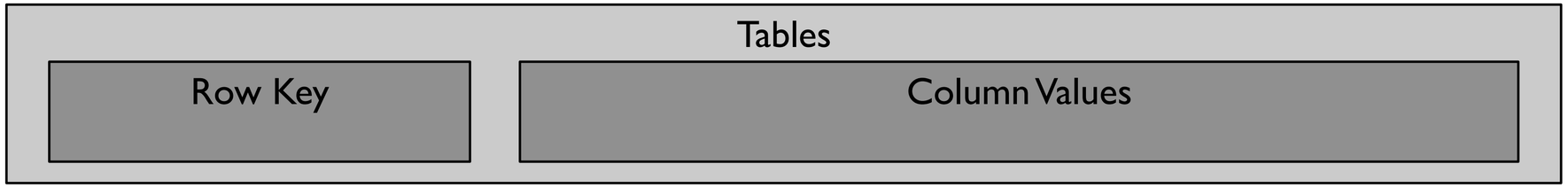
HBase Data Modeling

Considerations

strataconf.com

#StrataHadoop

HBase Data Modeling Considerations



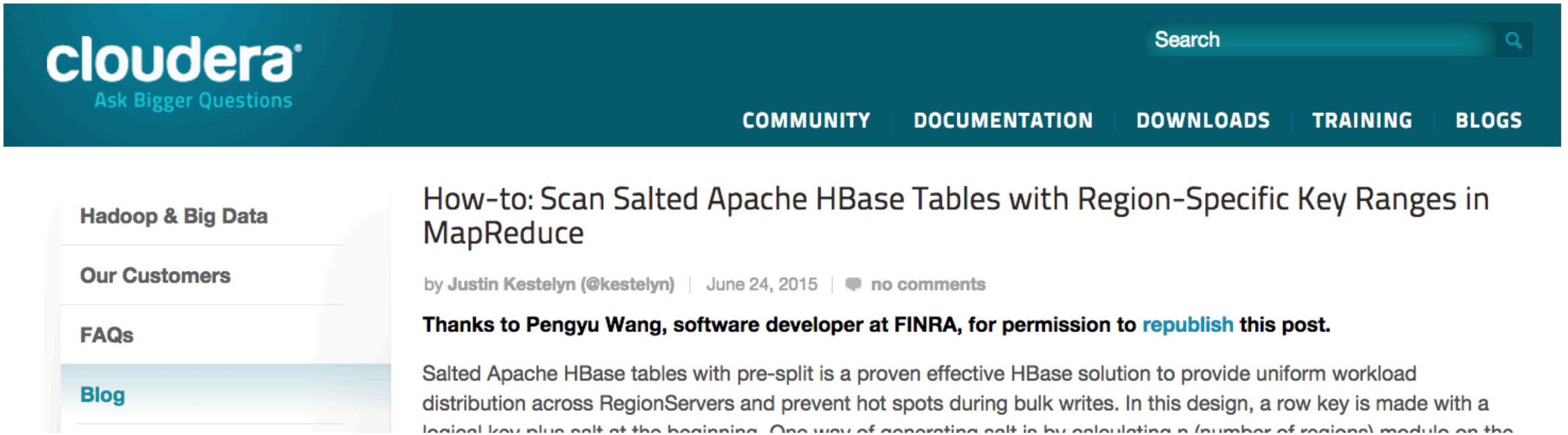
HBase Data Modeling Considerations

- Tables
  - *Minimize # of Tables*
  - *Reduce/Remove Joins*
  - *# Region*
  - *Split policy*

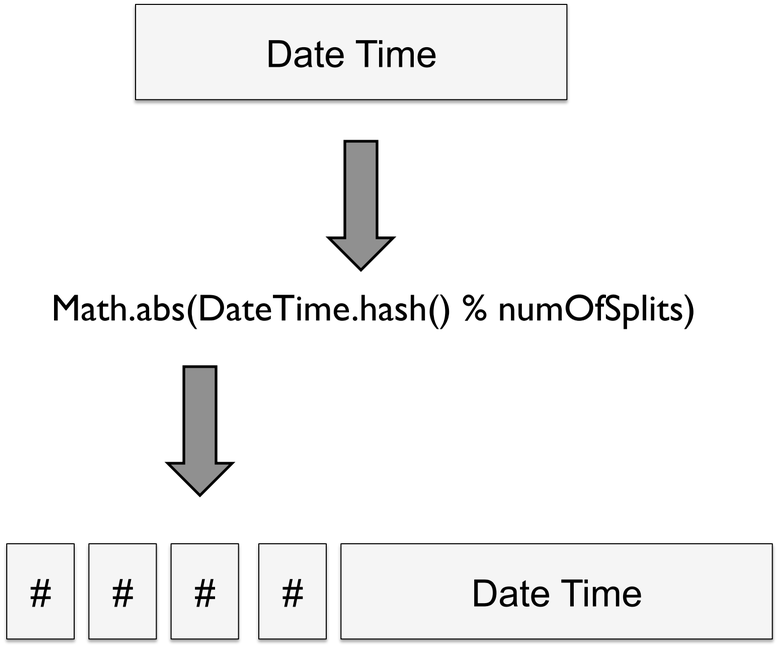
HBase Data Modeling Considerations

- RowKeys
  - *Location Location Location*
  - *Salt is good for you*

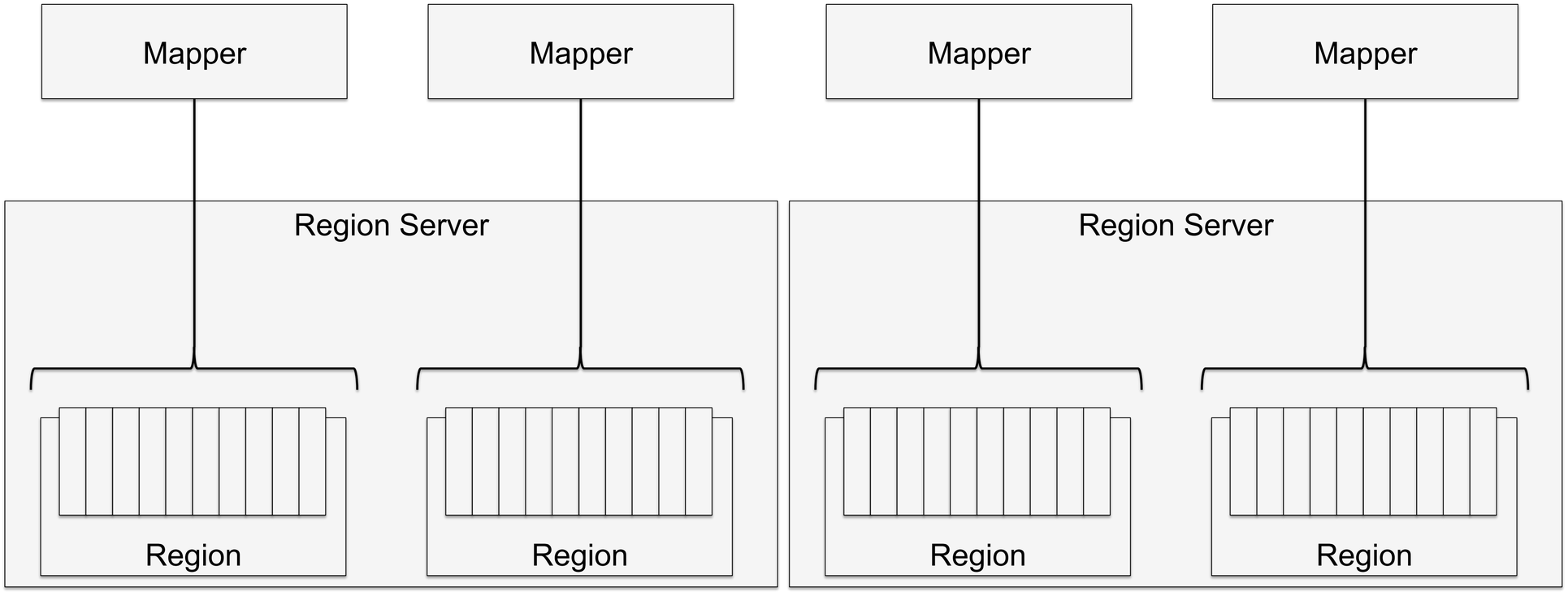
HBase Data Modeling Considerations



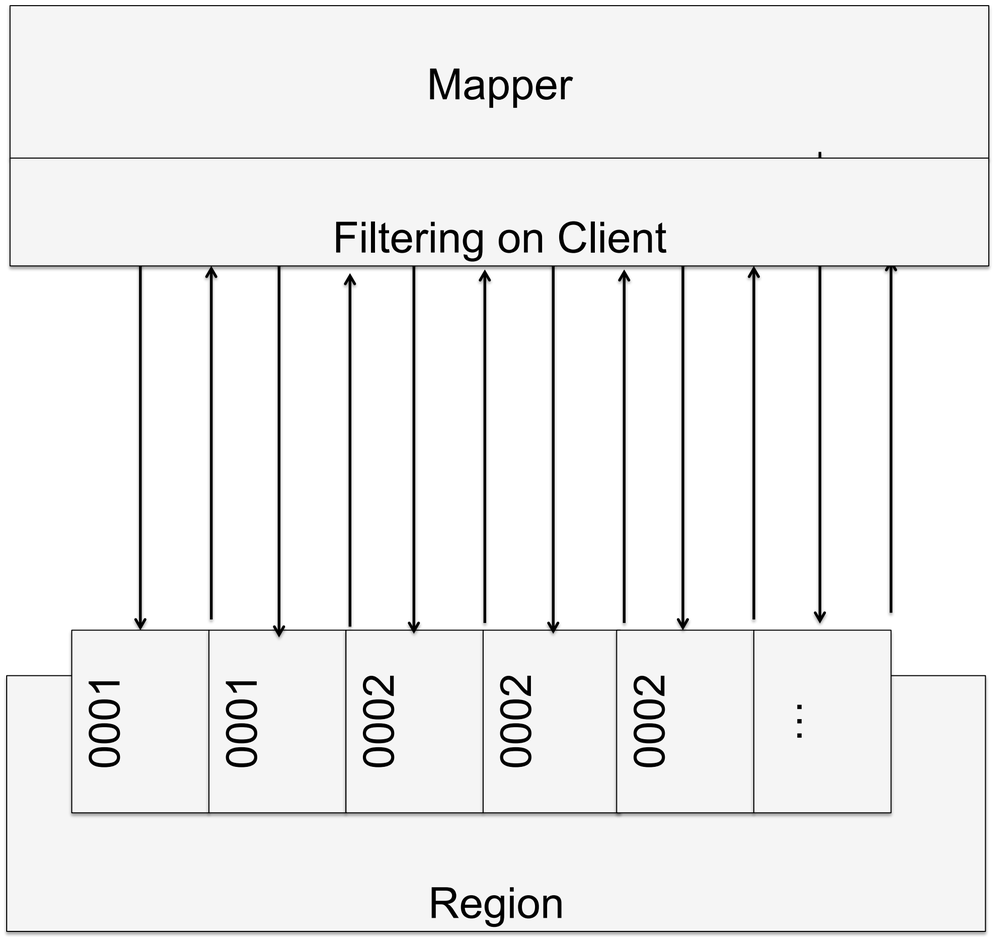
What is a Salt

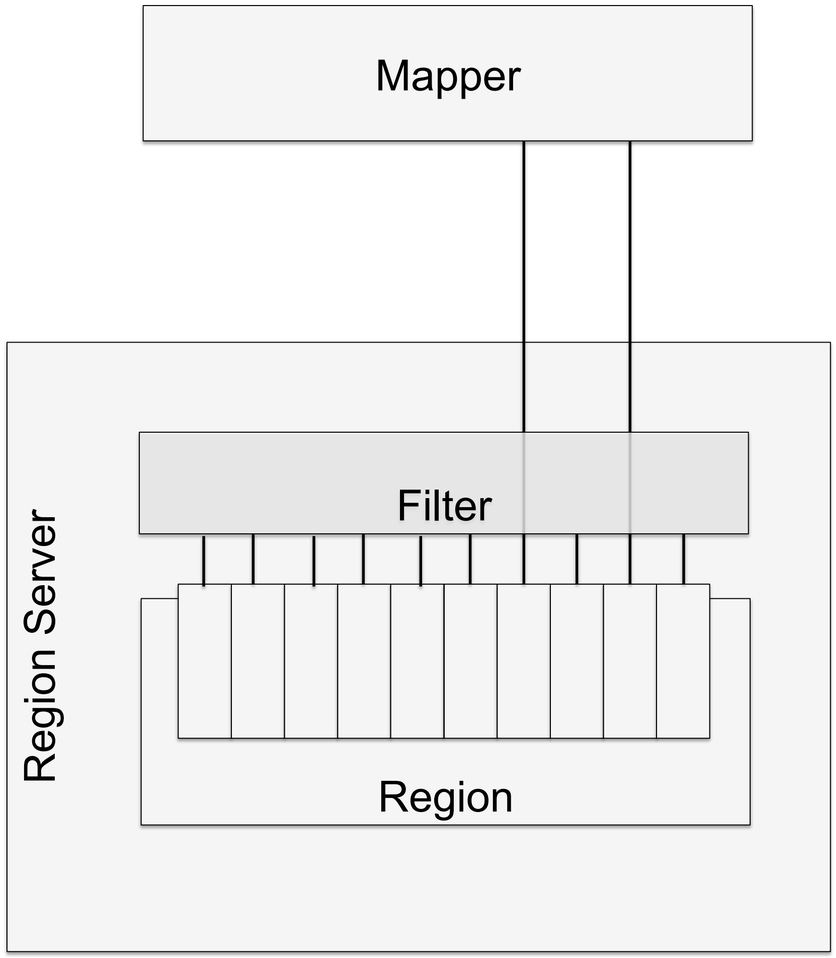


Scan a Salted Table

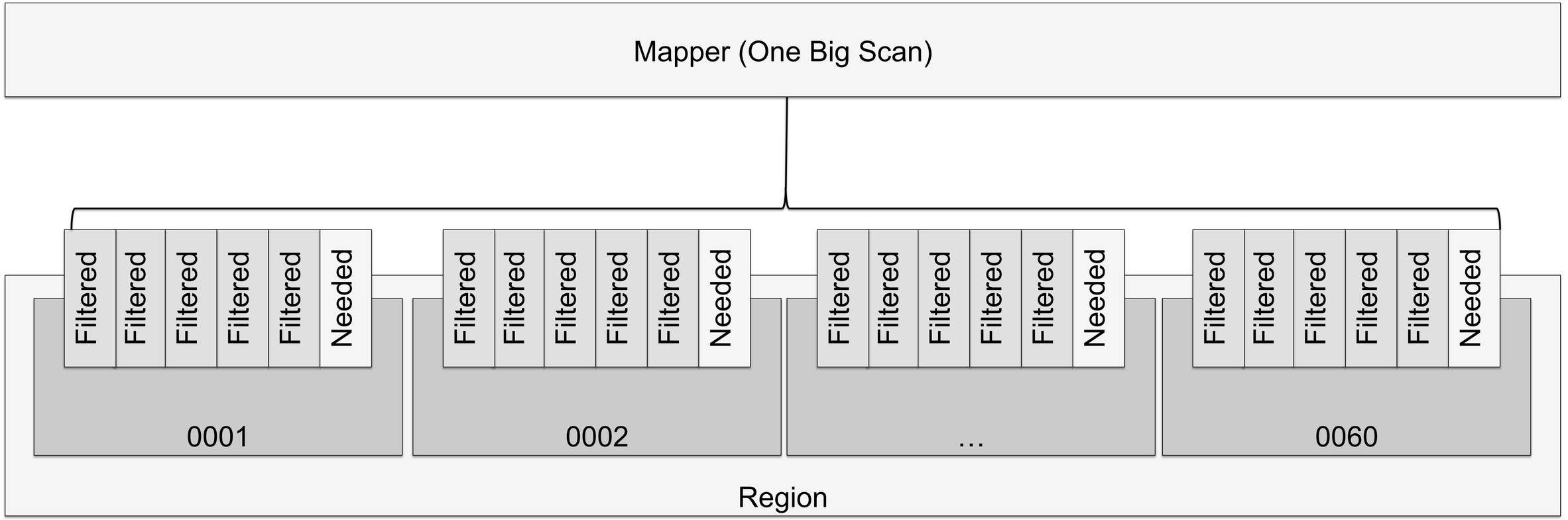


Scan a Salted Table

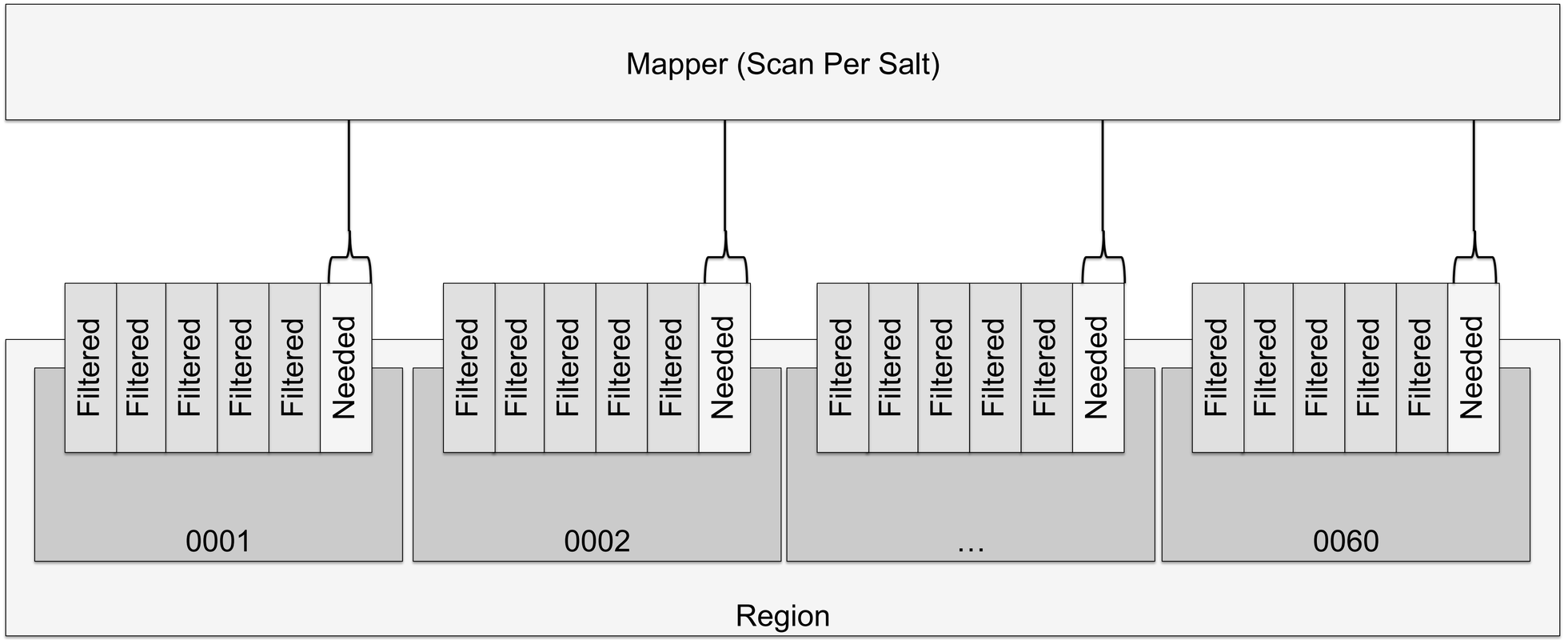




Scan a Salted Table



Scan a Salted Table



HBase Data Modeling Considerations

- Columns
  - *Mega Columns*
  - *Single Value Columns*
  - *Millions of Columns*
  - *Increment Values*

Strata+ Hadoop WORLD

PRESENTED BY

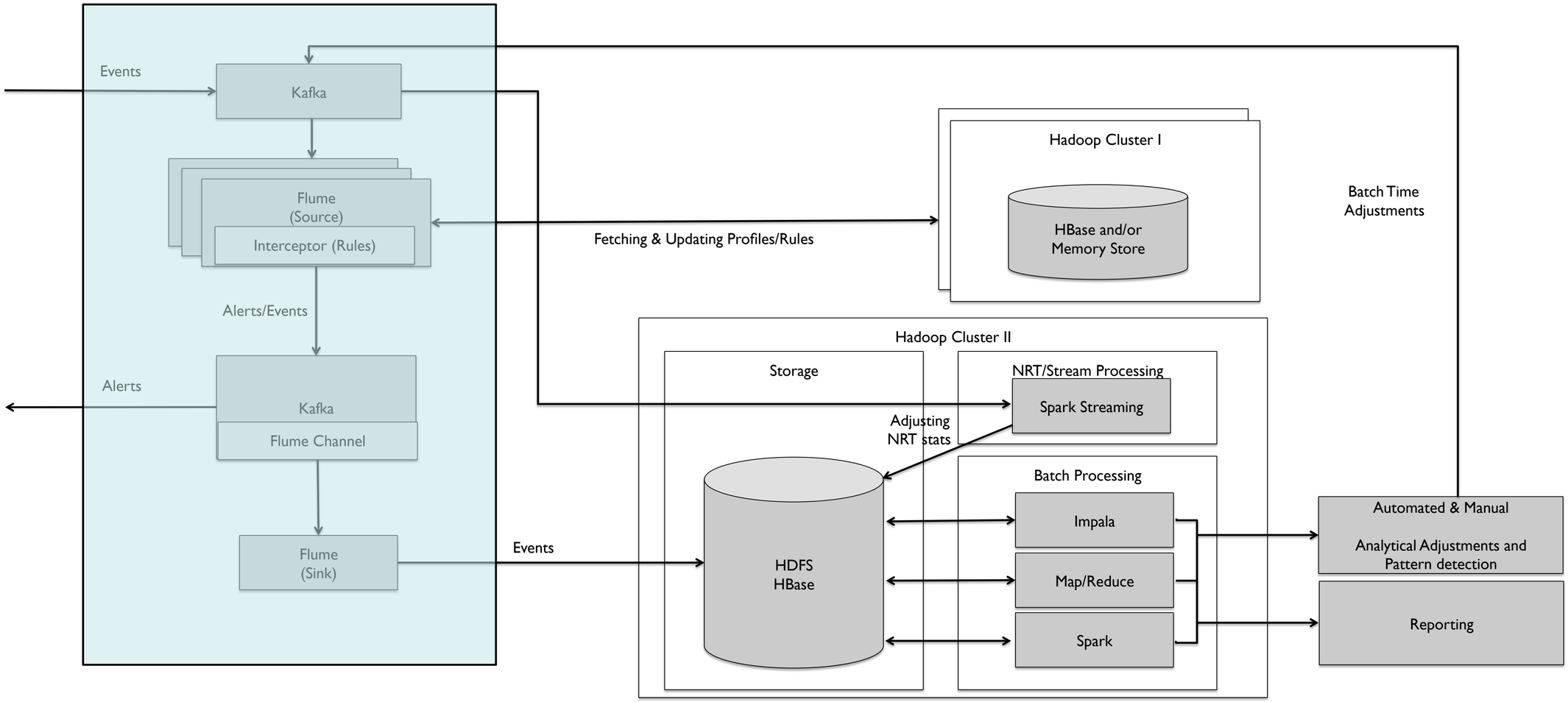
O'REILLY cloudera

strataconf.com

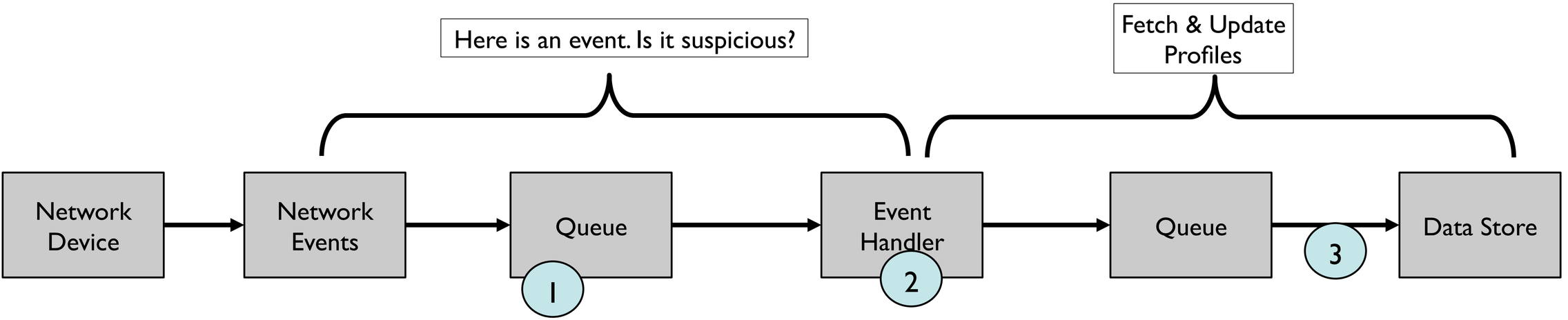
#StrataHadoop

Ingest and Near Real-Time Alerting

Considerations


Events
Kafka
Flume
(Source)
Interceptor (Rules)
Alerts/Events
Alerts
Kafka
Flume Channel
Flume
(Sink)
Fetching & Updating Profiles/Rules
Hadoop Cluster I
HBase and/or
Memory Store
Batch Time
Adjustments
Hadoop Cluster II
Storage
NRT/Stream Processing
Spark Streaming
Adjusting
NRT stats
Batch Processing
Events
HDFS
HBase
Impala
Map/Reduce
Spark
Automated & Manual
Analytical Adjustments and
Pattern detection
Reporting

The basic workflow



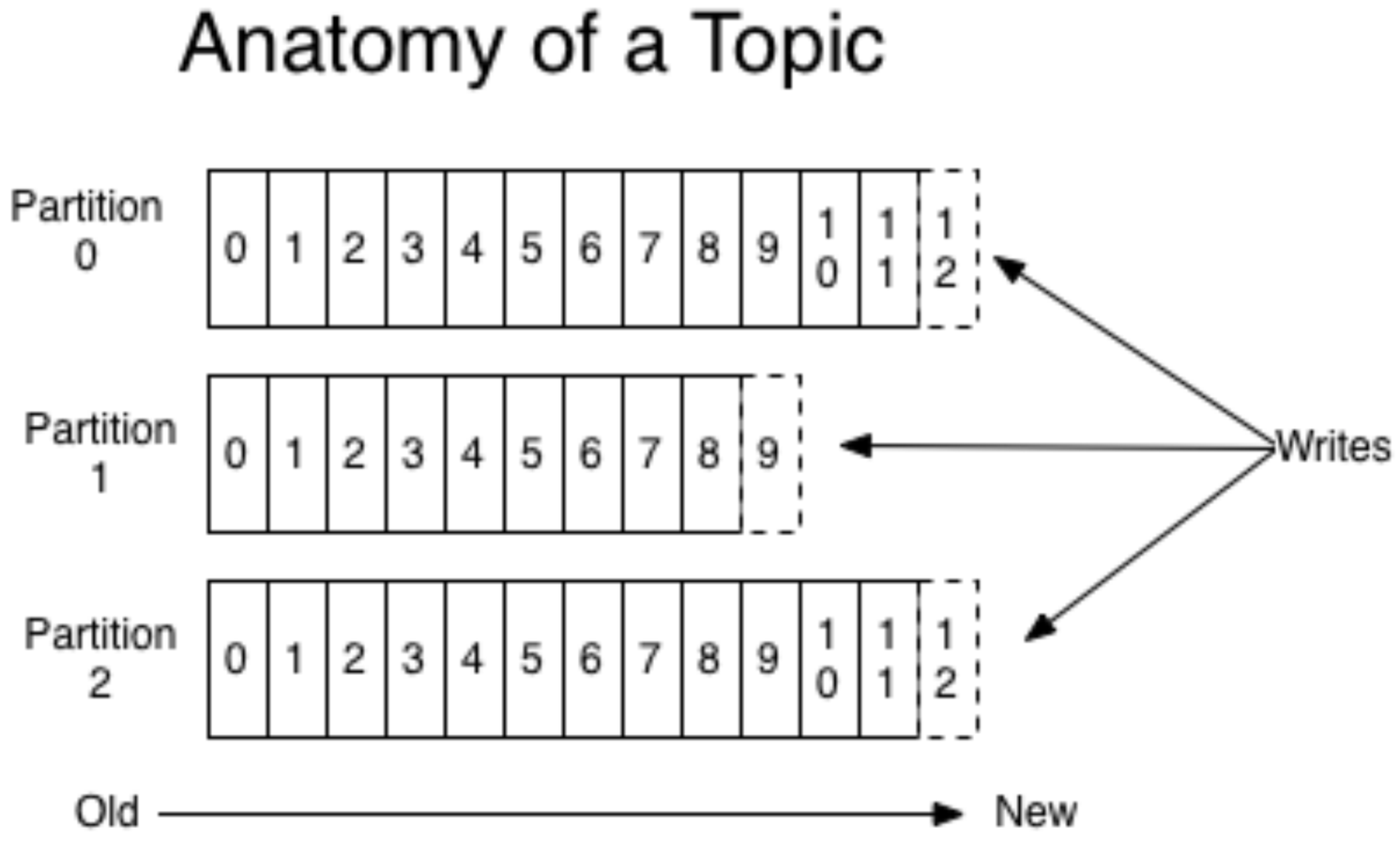
1 The Queue

- What makes Apache Kafka a good choice?
  - Low latency
  - High throughput
  - Partitioned and replicated
  - Easy to plug to applications

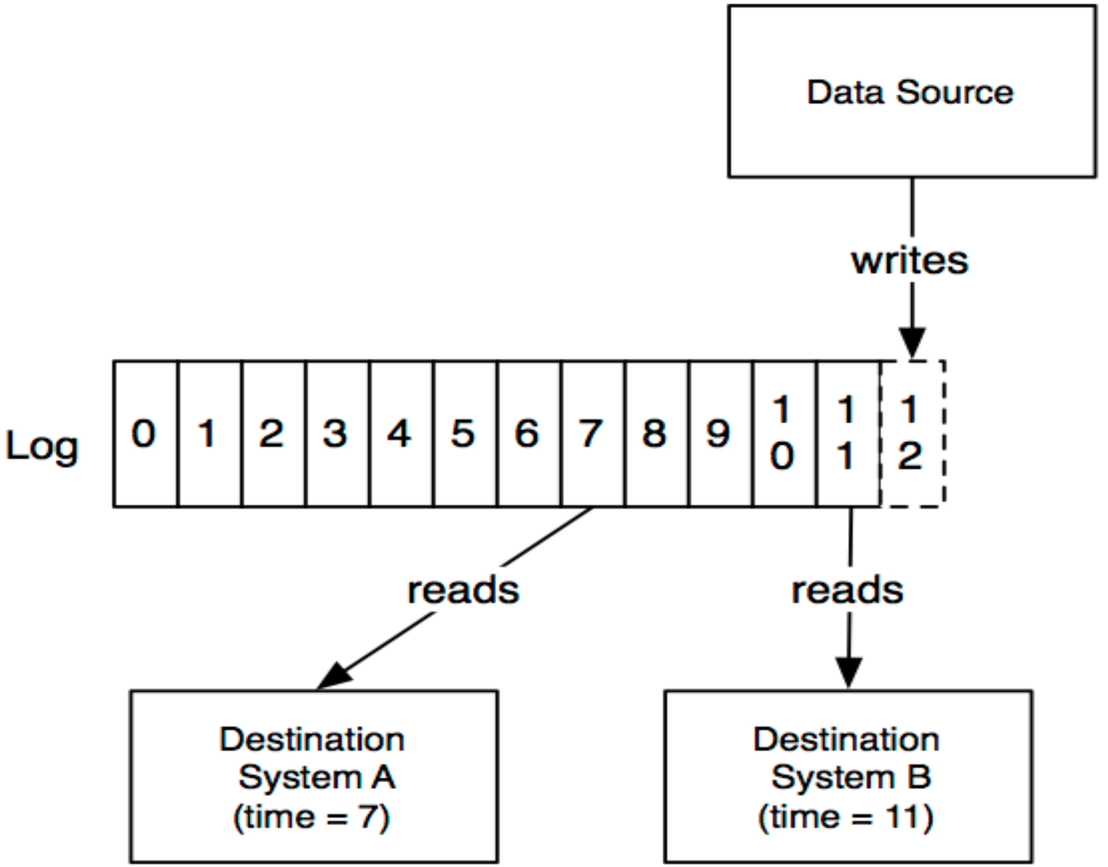
The Basics

- Messages are organized into **topics**
- **Producers** push messages
- **Consumers** pull messages
- Kafka runs in a cluster. Nodes are called **brokers**

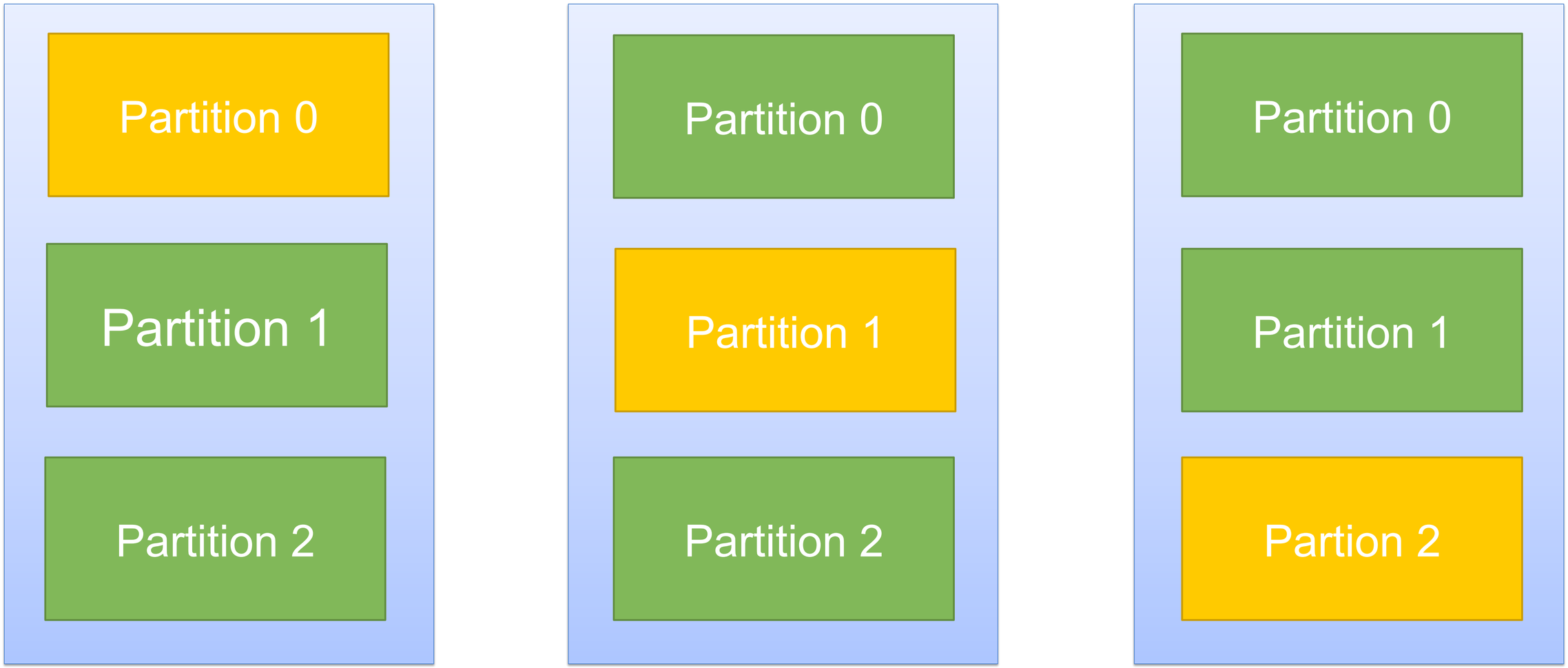
Topics, Partitions and Logs



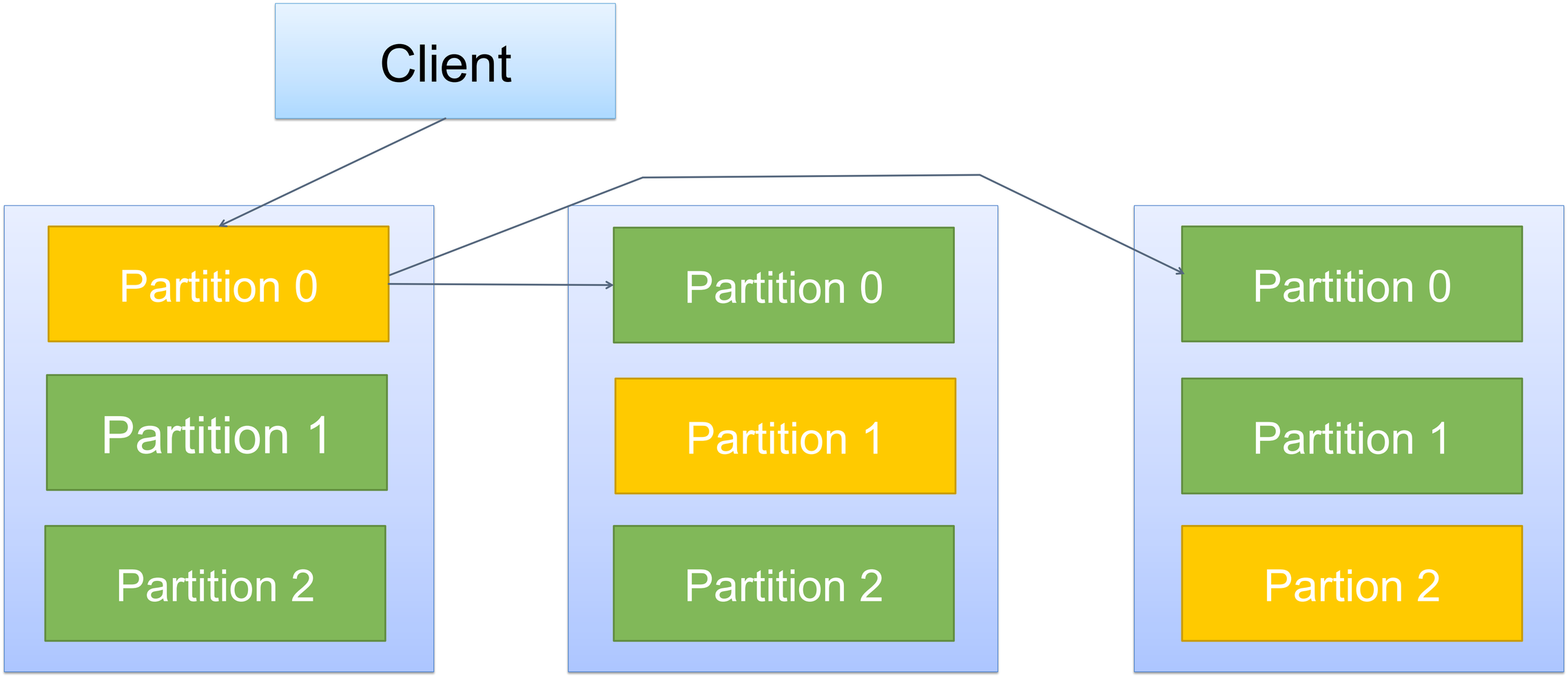
Each partition is a log



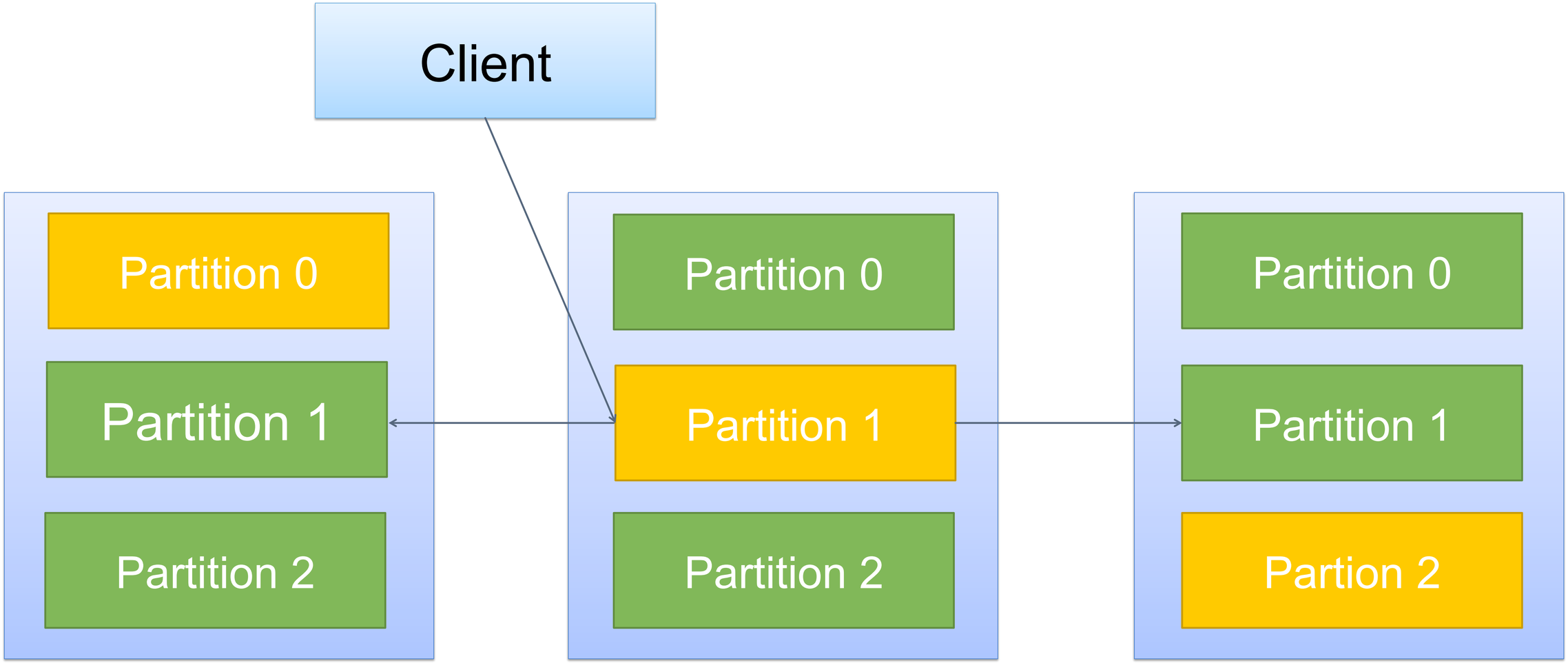
Each Broker has many partitions



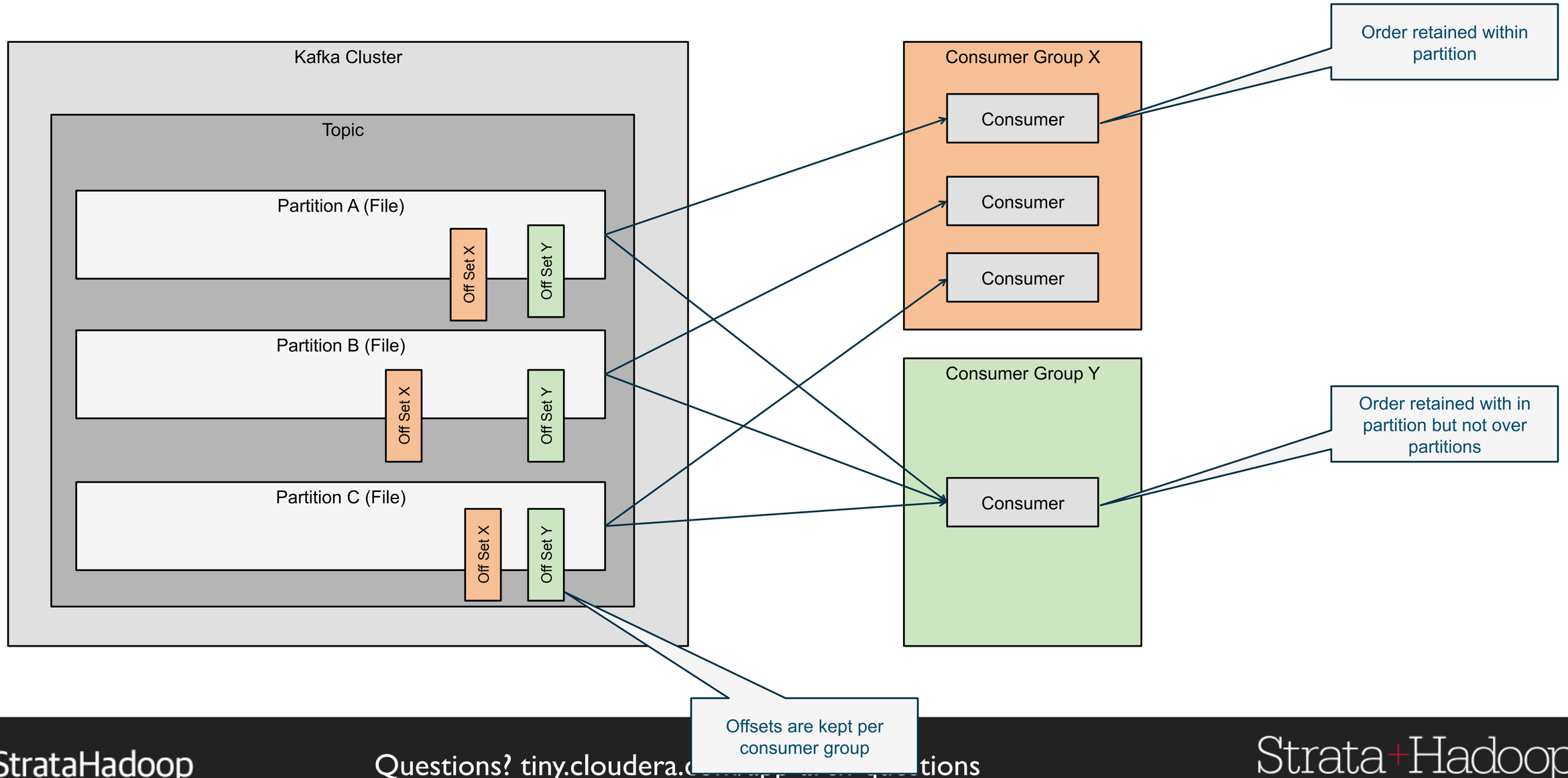
Producers load balance between partitions



Producers load balance between partitions

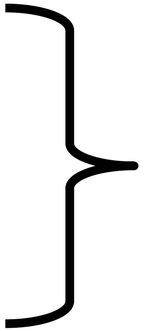


Consumers


Kafka Cluster
Topic
Partition A (File)
Off Set X
Off Set Y
Partition B (File)
Off Set X
Off Set Y
Partition C (File)
Off Set X
Off Set Y
Consumer Group X
Consumer
Consumer
Consumer
Consumer Group Y
Consumer
Order retained within partition
Order retained with in partition but not over partitions
Offsets are kept per consumer group

In our use-case

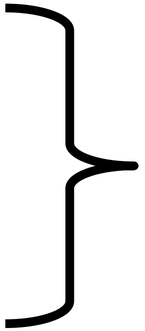
- Topic for profile updates
- Topic for current events
- Topic for alerts
- Topic for rule updates



Partitioned by device ID

2 The event handler

- Very basic pattern:
  - Consume a record
  - “do something to it”
  - Produce zero, one, or more results



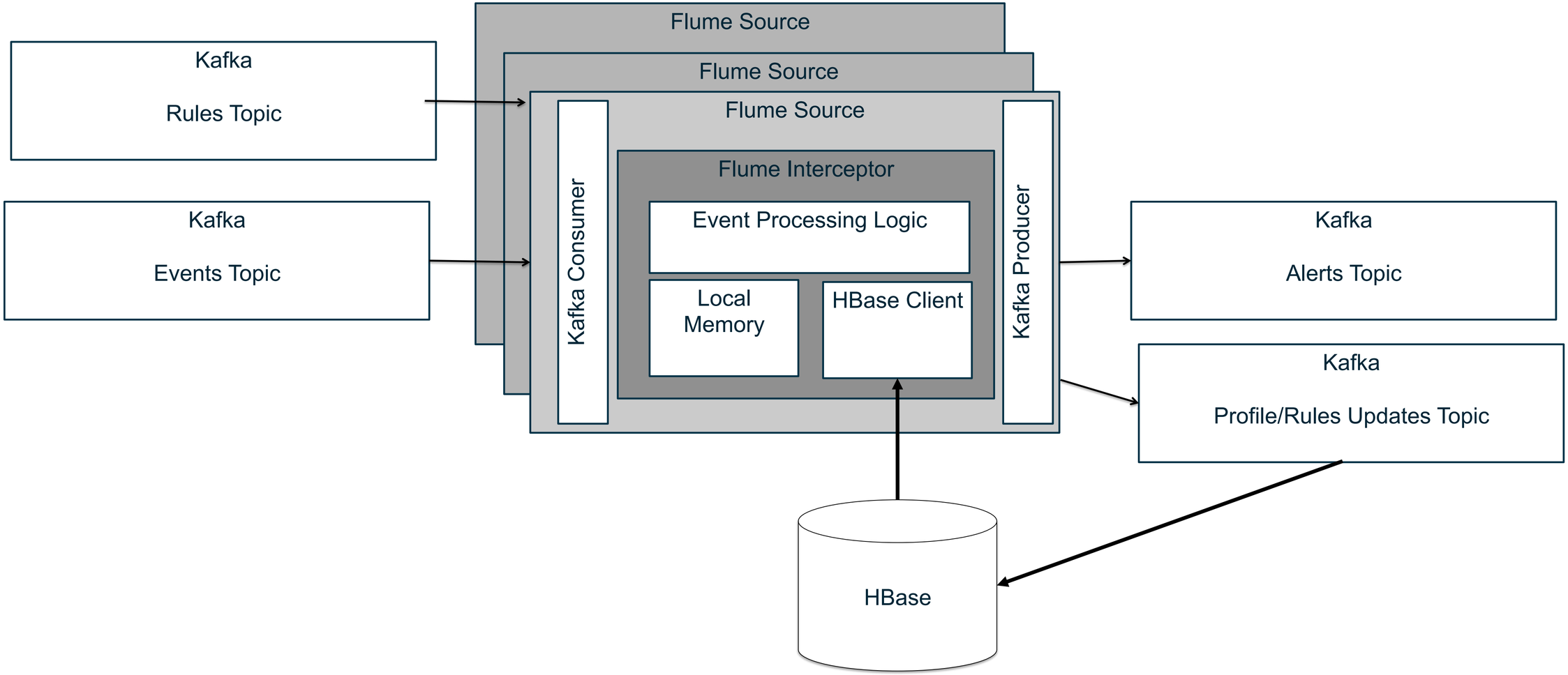
Kafka Processor Pattern

Easier than you think

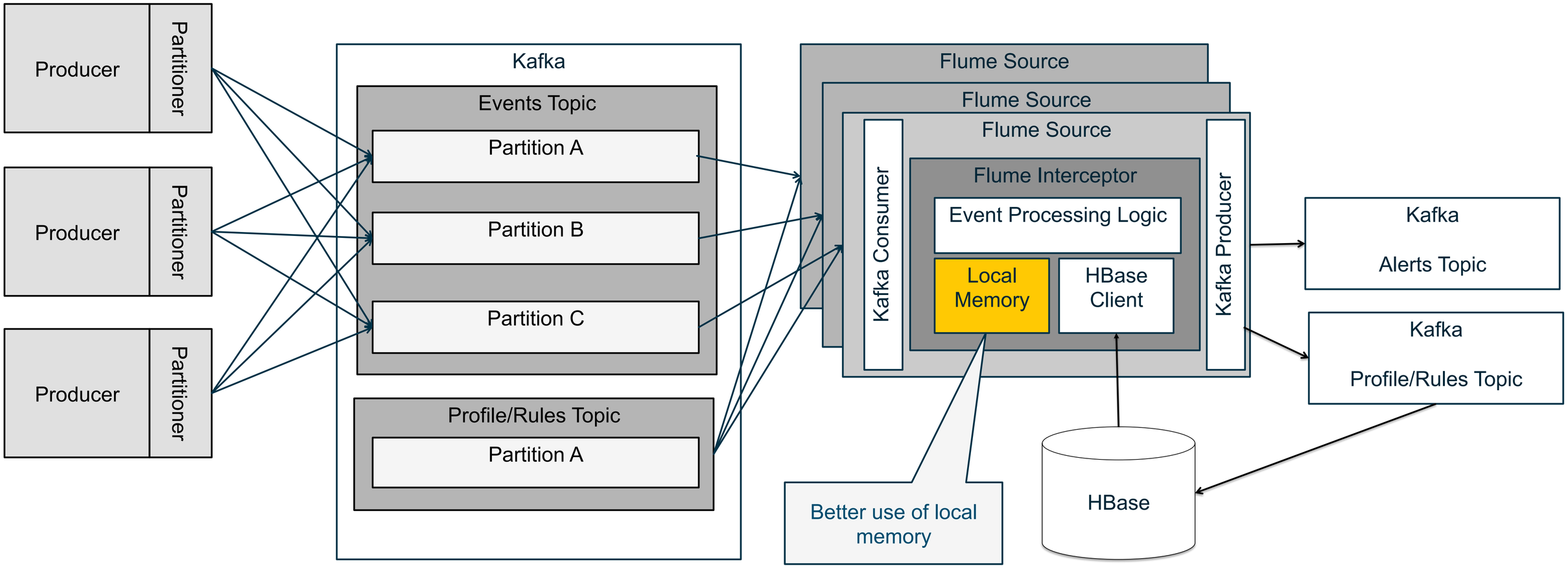
- Kafka provides load balancing and availability
  - Add processes when needed – they will get their share of partitions
  - You control partition assignment strategy
  - If a process crashes, partitions will get automatically reassigned
  - You control when an event is “done”
  - Send events safely and asynchronously
- So you focus on the use-case, not the framework

So easy a PHP developer can do it

Inside The Processor



Scaling the Processor



3 Ingest – Gateway to deep analytics

- Data needs to end up in:
  - SparkStreaming
    - Can read directly from Kafka
  - HBase
  - HDFS

Considerations for Ingest

- **Async** – nothing can slow down the immediate reaction
  - Always ingest the data from a queue in separate thread or process to avoid write-latency
- **Push vs Pull**
- **Useful end-points** – integrates with variety of data sources and sinks
- Supports **standard formats** and possibly some transformations
  - Data format inside the queue can be different than in the data store For example Avro -> Parquet

More considerations - Safety

- **Reliable** – data loss is not acceptable
  - Either “at-least-once” or “exactly-once”
  - When is “exactly-once” possible?
- **Error handling** – reasonable reaction to unreasonable data
- **Security –** Authentication, authorization, audit, lineage

Good Patterns

- Ingest all things. Events, evidence, alerts
  - It has potential value
  - Especially when troubleshooting
  - And for data scientists
- Make sure you take your schema with you
  - Yes, your data probably has a schema
  - Everyone accessing the data (web apps, real time, stream, batch) will need to know about it
  - So make it accessible in a centralized schema registry

Choosing Ingest

Our choice for this example

Flume

- Part of Hadoop
- Key-value based
- Supports HDFS and HBase
- Easy to configure
- Serializers provide conversions
- At-least once delivery guarantee
- Proven in huge production environments

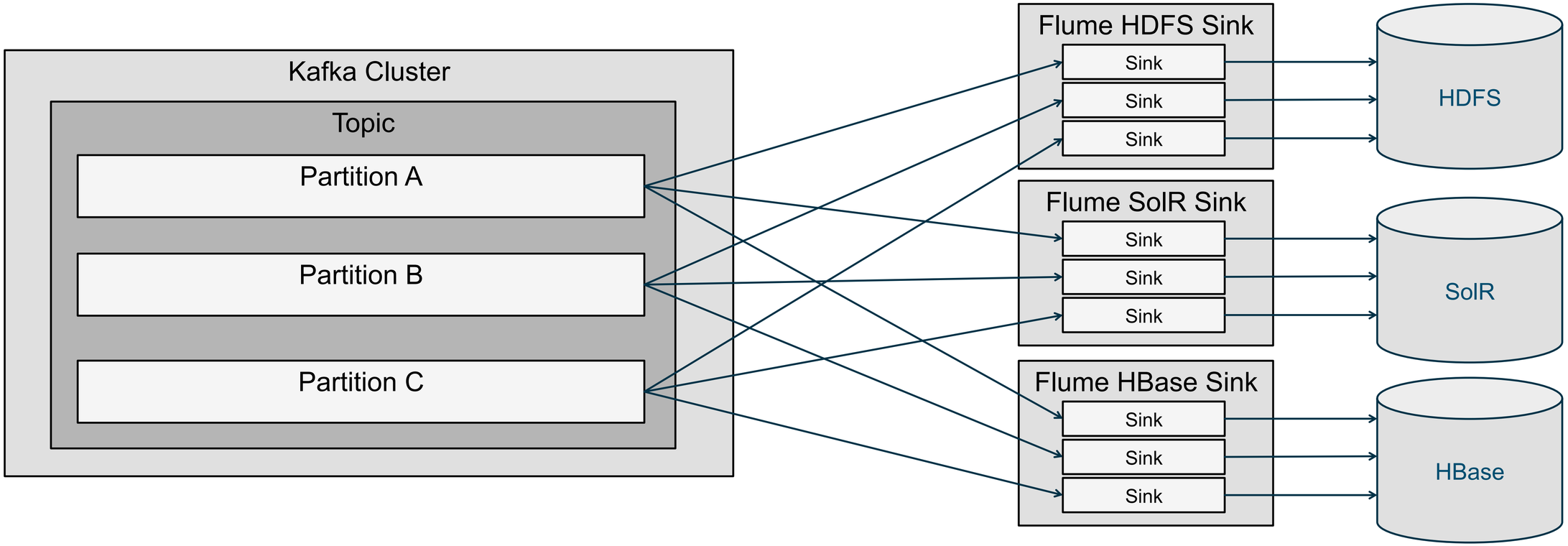
Kafka Connect

- Part of Apache Kafka
- Schema preserving
- Many connectors
- Well integrated support for Parquet, Avro and JSON
- Exactly-once delivery guarantee
- Easy to manage and scale

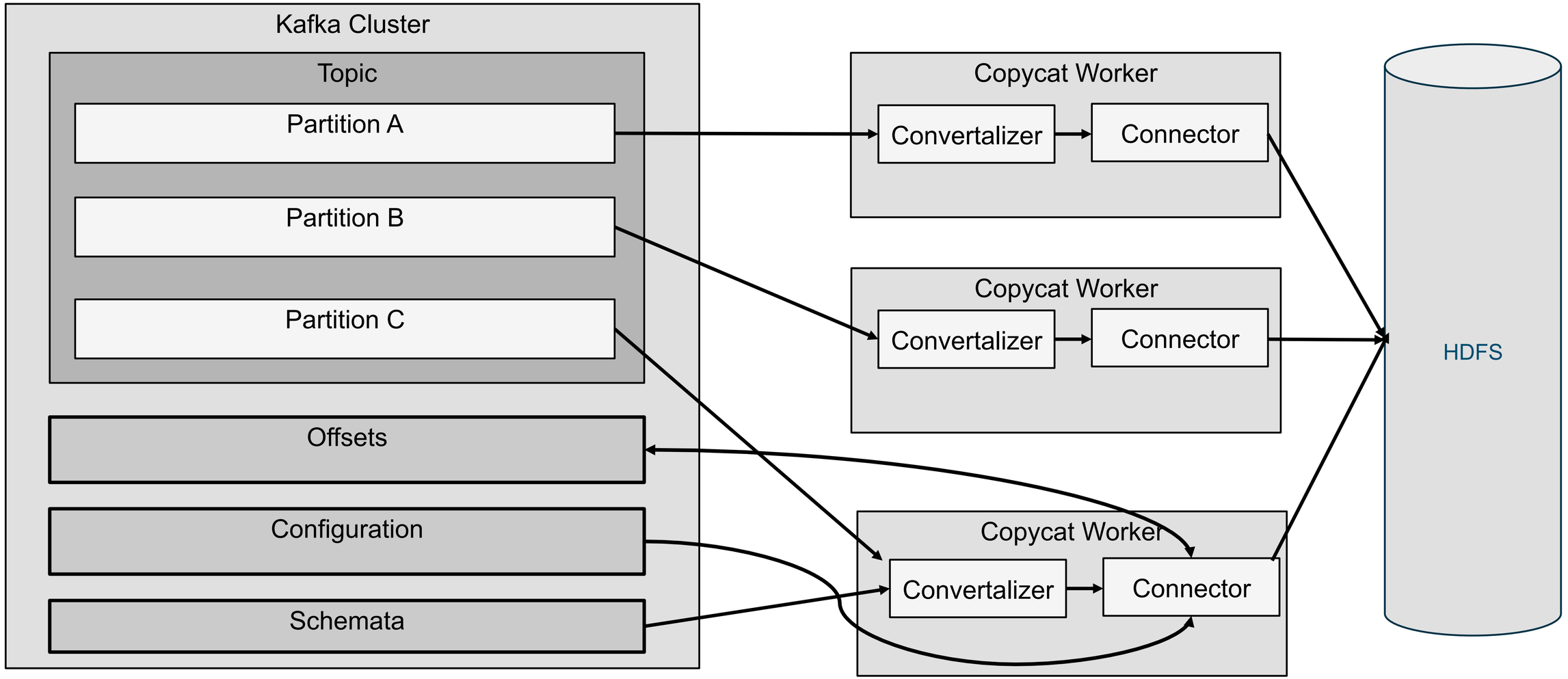
Spark Streaming

- Part of Hadoop
- Very strong transformation story
- Not as simple as Flume or Connect
- Supports HDFS and HBase
- Smart Kafka Integration
- Exactly-once guarantee from Kafka

Ingestion - Flume



Ingestion – Kafka Connect

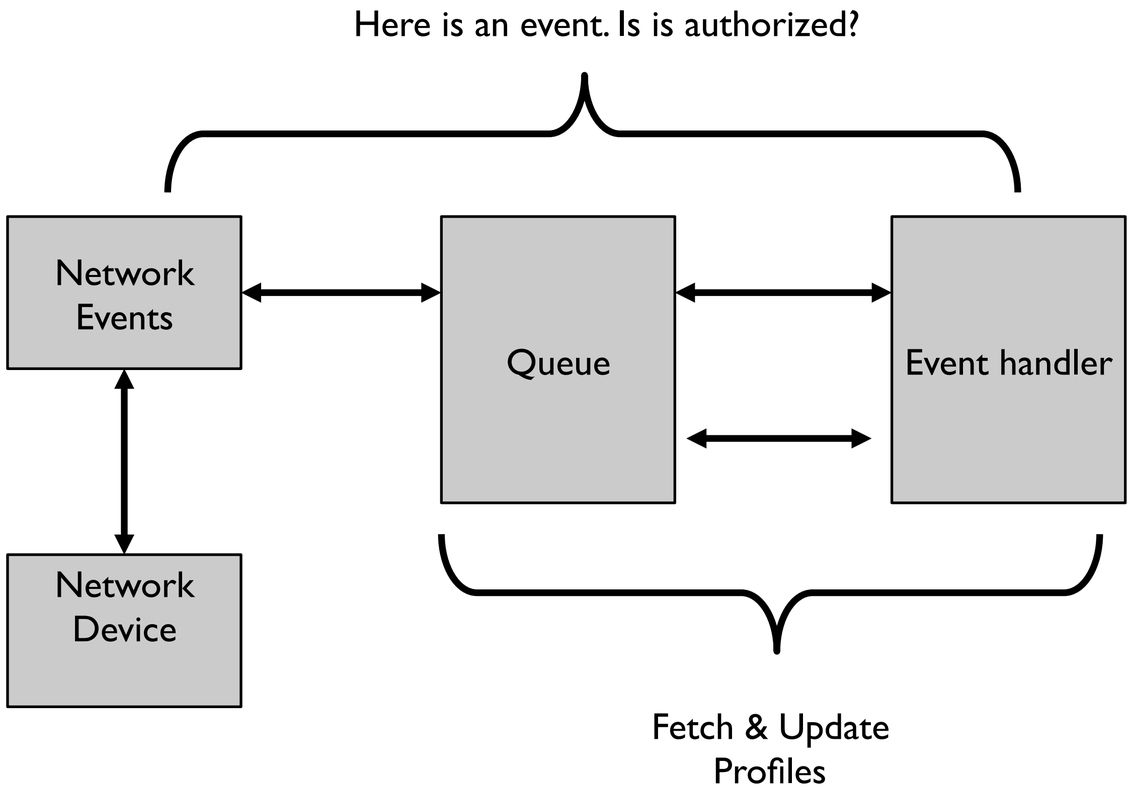


Alternative Architectures

- What if…

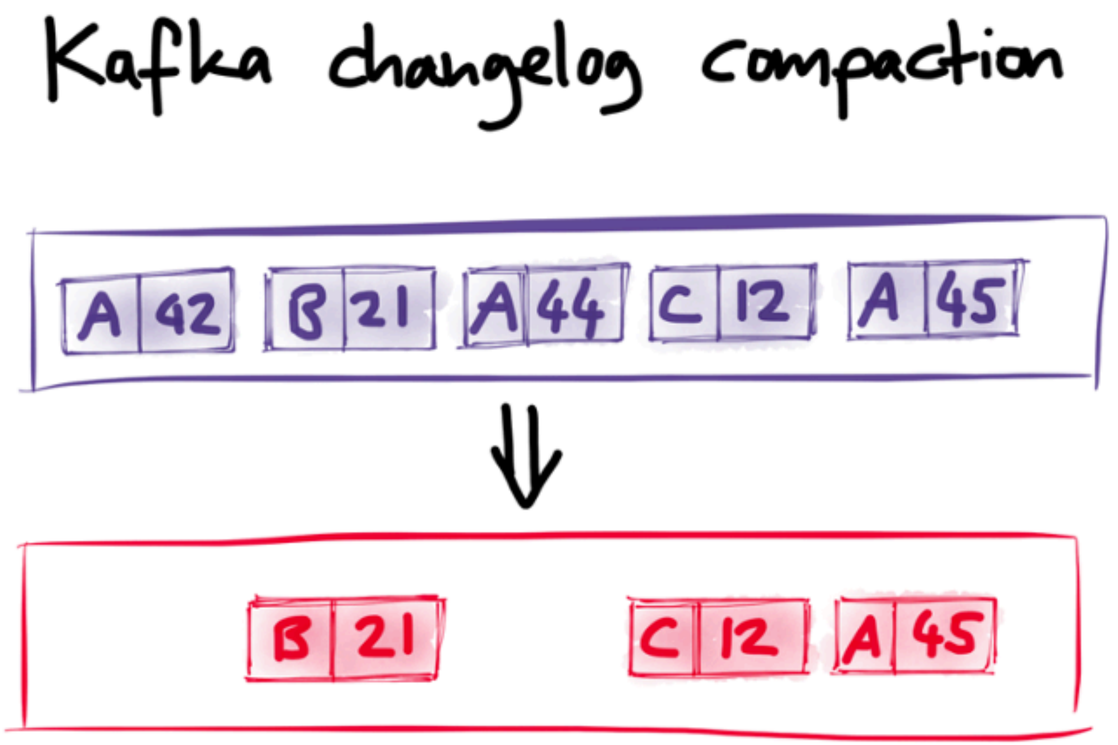
Do we really need HBase?

- What if… Kafka would store all information used for real-time processing?



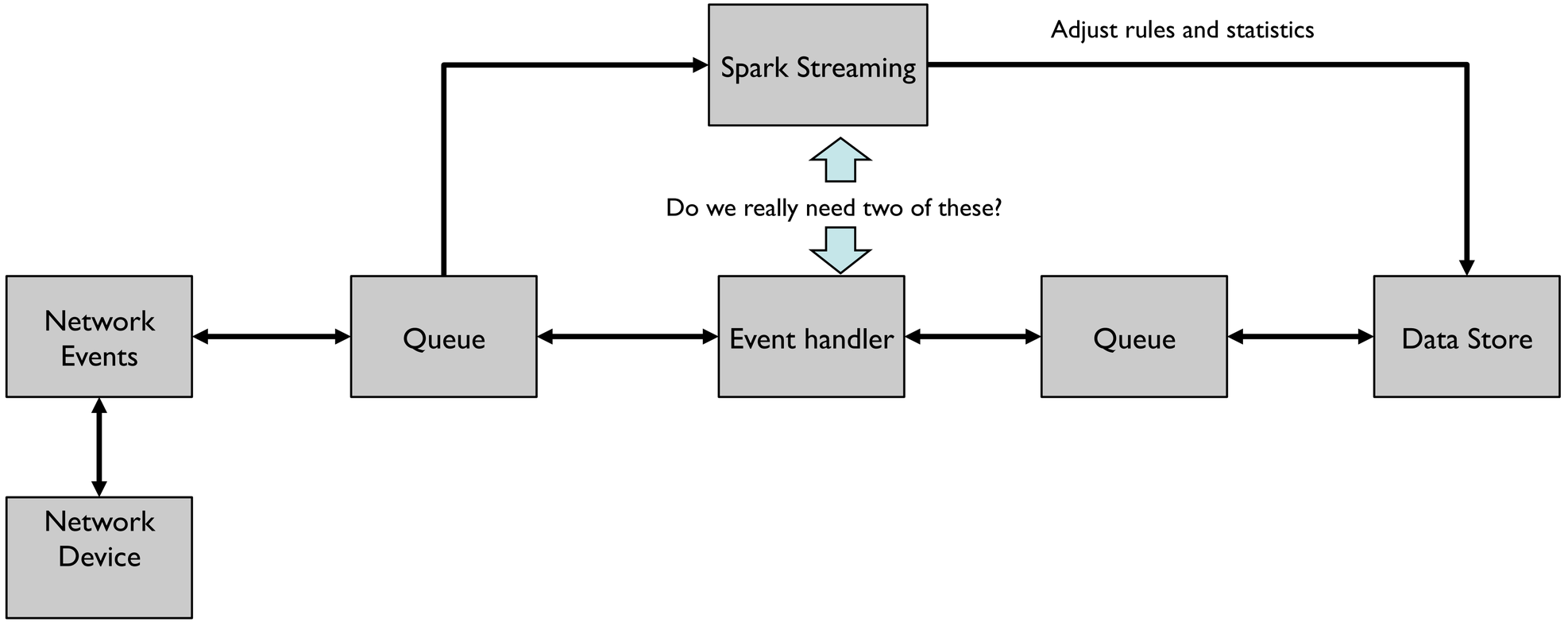
Log Compaction in Kafka

- Kafka stores a “change log” of profiles
- But we want the latest state
- log.cleanup.policy = {delete / compact}



Do we really need Spark Streaming?

- What if… we could do low-latency single event processing and complex stream processing analytics in the same system?



Is complex processing of single events possible?

- The challenge:
  - Low latency vs high throughput
  - No data loss
  - Windows, aggregations, joins
  - Late data

Solutions?

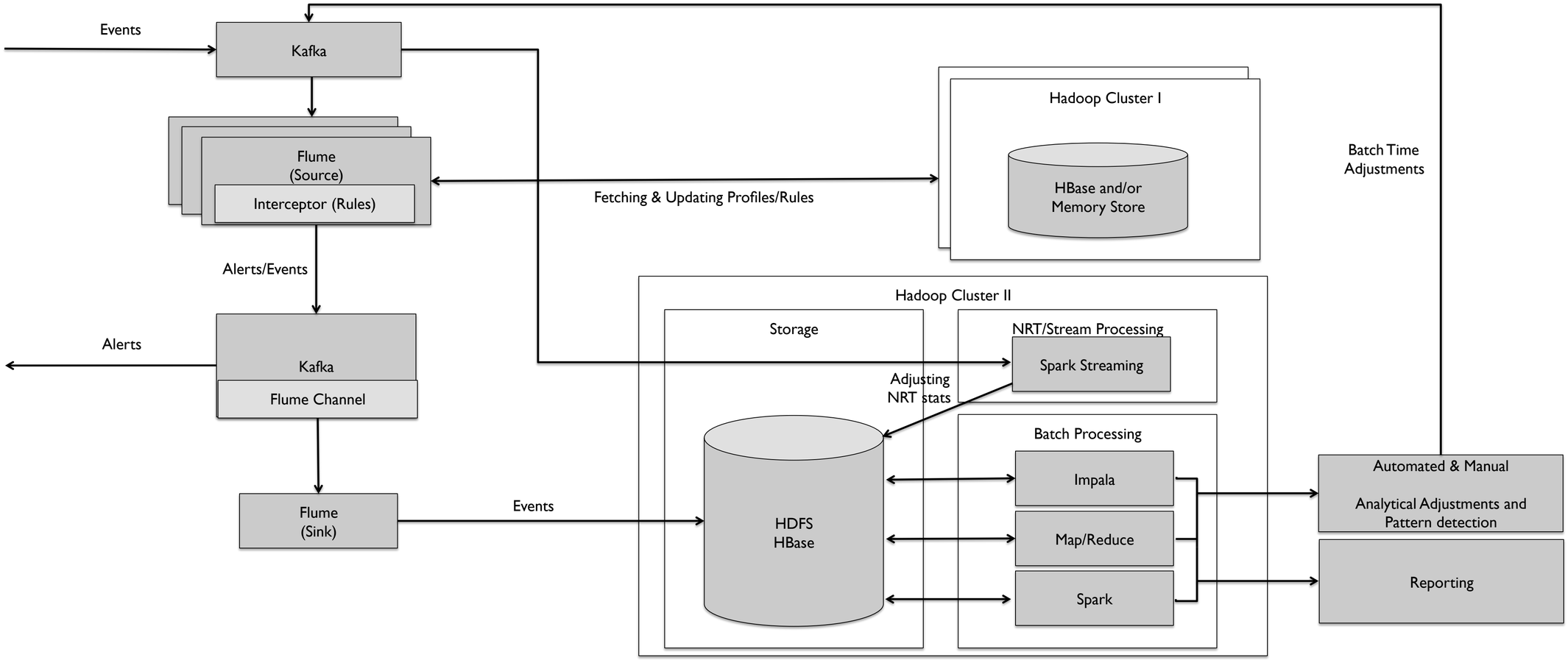
- Apache Flink
  - Process each event, checkpoint in batches
  - Local and distributed state, also checkpointed
- Apache Samza
  - State is persisted to Kafka (Change log pattern)
  - Local state cached in RocksDB
- Kafka Streams
  - Similar to Samza
  - But in simple library that is part of Apache Kafka.
  - Late data as updates to state

You Probably Don't Need Orchestration

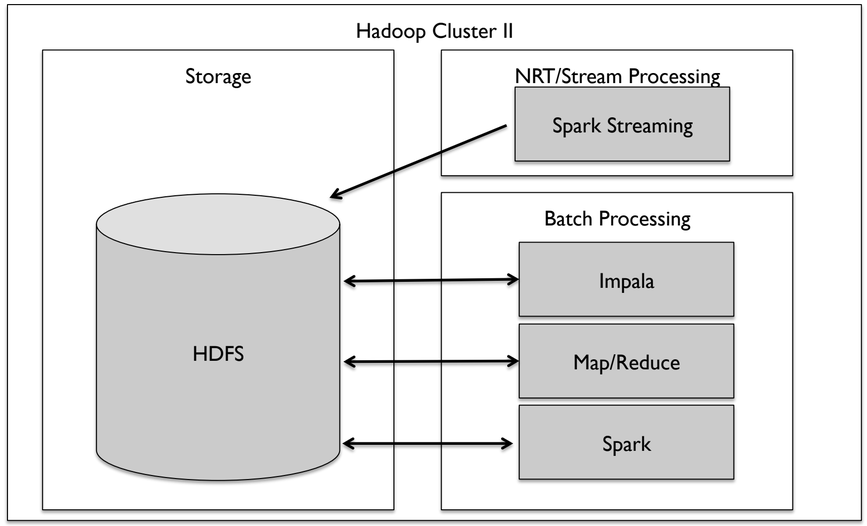
Processing

Considerations

Reminder: Full Architecture

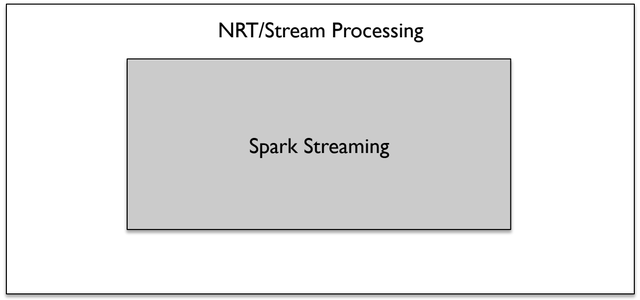


Processing

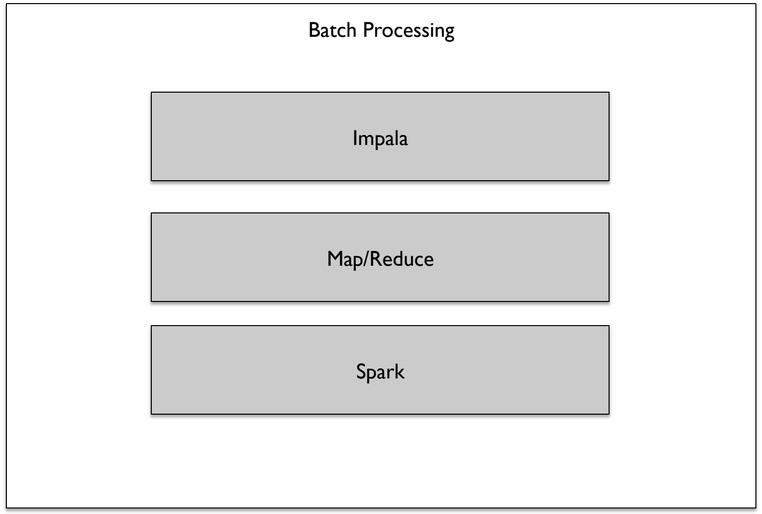

Hadoop Cluster II
Storage
NRT/Stream Processing
Spark Streaming
Batch Processing
HDFS
Impala
Map/Reduce
Spark

Two Kinds of Processing

Streaming



Batch



Strata+ Hadoop WORLD

PRESENTED BY

O'REILLY cloudera

strataconf.com

#StrataHadoop

Stream Processing

Considerations

Stream Processing Options

1. Storm
2. Spark Streaming
3. KafkaStreams
4. Flink

Why Spark Streaming?

- There's one engine to know.
- Micro-batching helps you scale reliably.
- Exactly once counting
- Hadoop ecosystem integration is baked in.
- No risk of data loss.
- It's easy to debug and run.
- State management
- Huge eco-system backing
- You have access to ML libraries.
- You can use SQL where needed.
- Wide-spread use and hardened

Spark Streaming Example

1. val conf = new SparkConf().setMaster("local[2]”)
2. val ssc = new StreamingContext(conf, Seconds(1))
3. val lines = ssc.socketTextStream("localhost", 9999)
4. val words = lines.flatMap(_.split(" "))
5. val pairs = words.map(word => (word, 1))
6. val wordCounts = pairs.reduceByKey(_ + _)
7. wordCounts.print()
8. SSC.start()

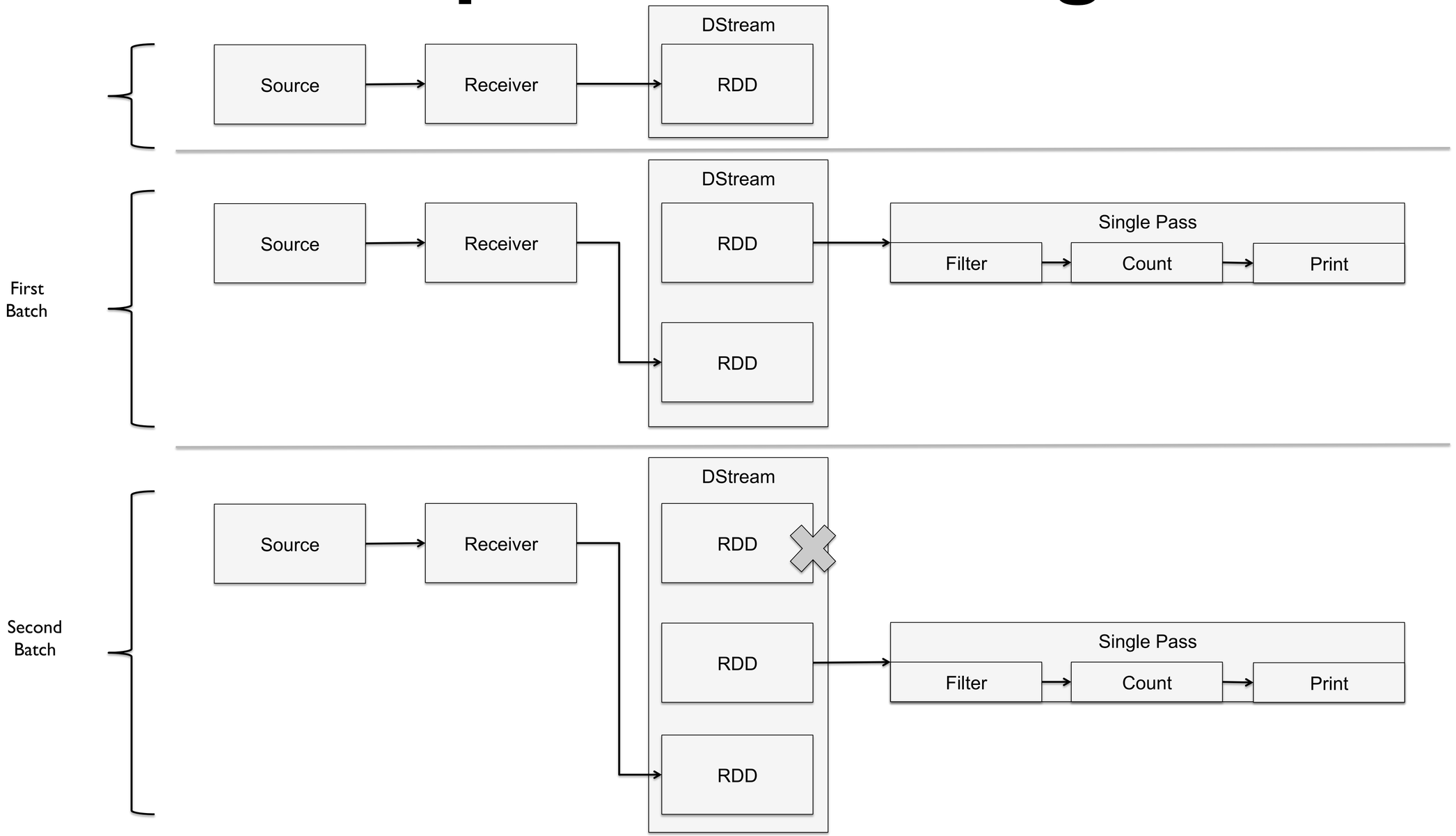
Spark Streaming Example

1. val conf = new SparkConf().setMaster("local[2]”)
2. val sc = new SparkContext(conf)
3. val lines = sc.textFile(path, 2)
4. val words = lines.flatMap(_.split(" "))
5. val pairs = words.map(word => (word, 1))
6. val wordCounts = pairs.reduceByKey(_ + _)
7. wordCounts.print()

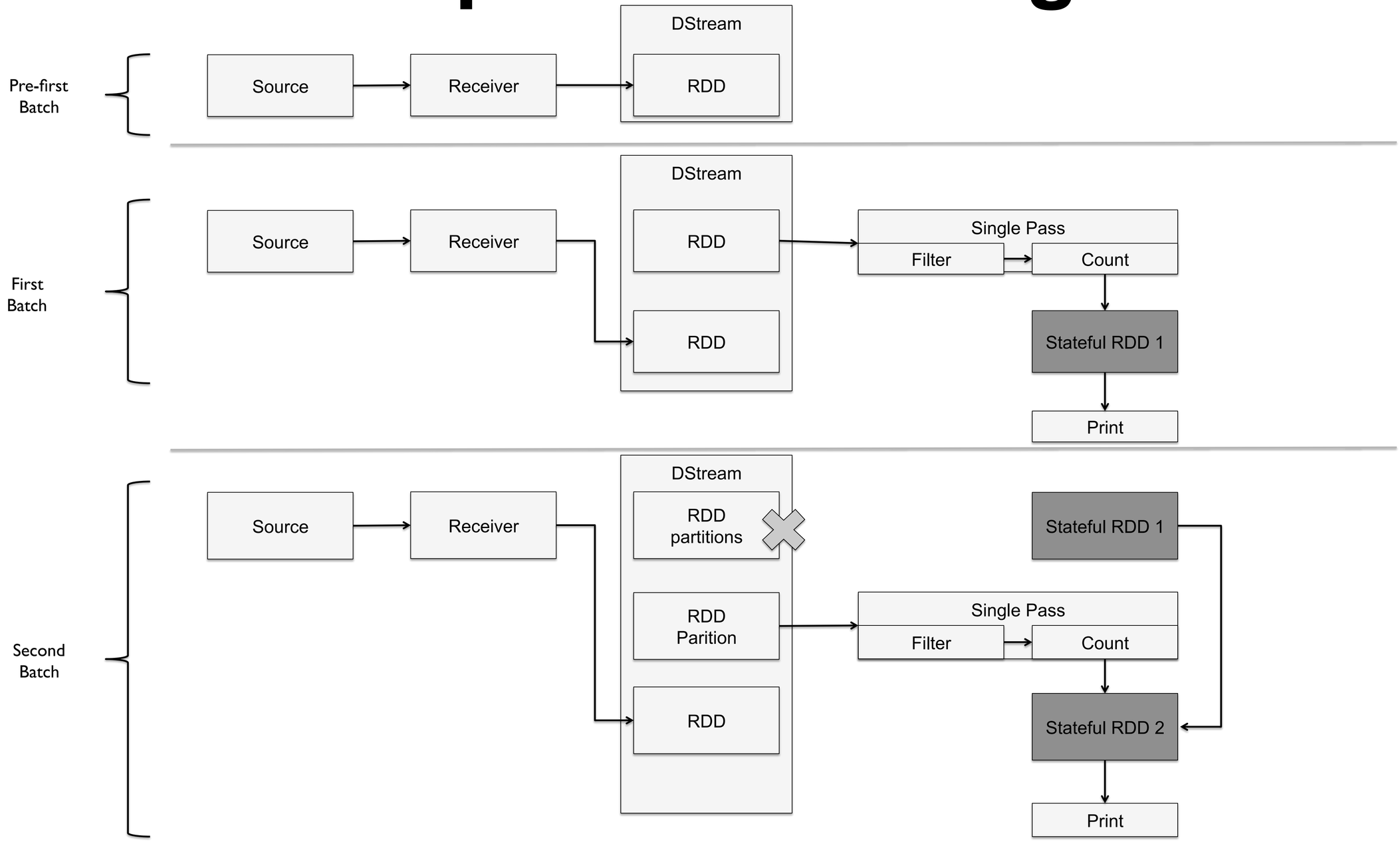
Spark SQL and Spark Streaming

- Catalyst
  - Spark SQL's optimizer
- Bringing Catalyst and Spark SQL integration to streaming

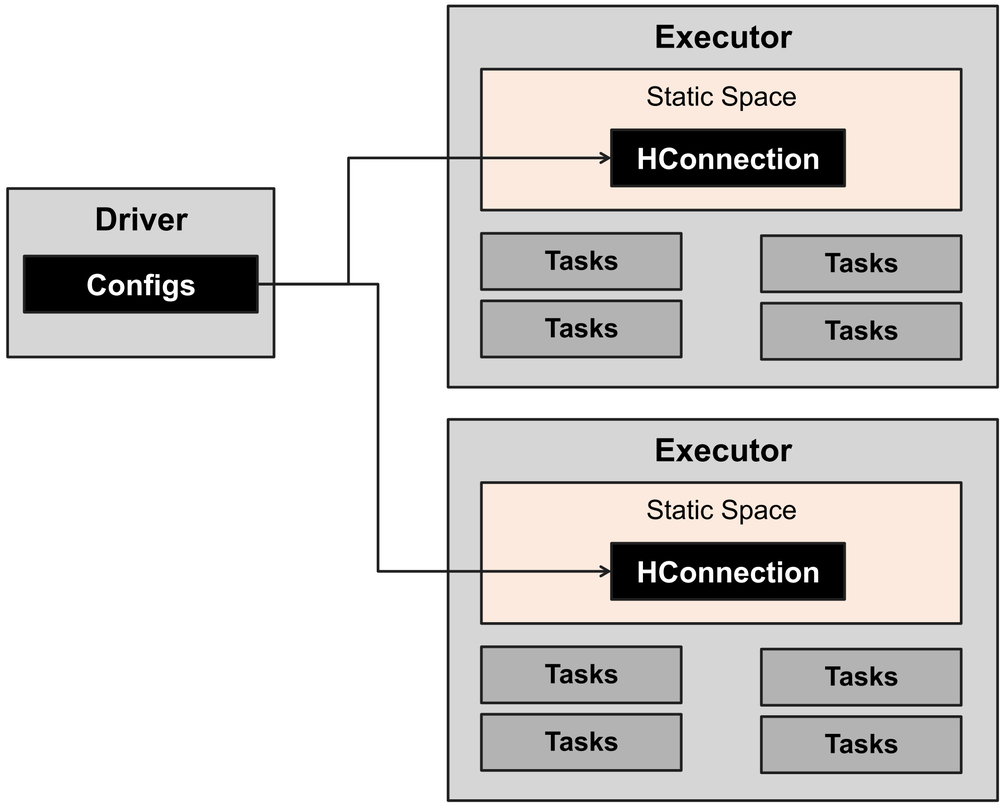
Spark Streaming



Spark Streaming



Spark Streaming and HBase



HBase-Spark Module

- BulkPut
- BulkDelete
- BulkGet
- BulkLoad (Wide and Tail)
- ForeachPartition(It, Conn)
- MapPartition(it, Conn)
- Spark SQL

HBase-Spark Module

```
val hbaseContext = new HBaseContext(sc, config);
hbaseContext.bulkDelete[Array[Byte]](rdd,
                    tableName,
                    putRecord => new Delete(putRecord),
                    4);

val hbaseContext = new HBaseContext(sc, config)
rdd.hbaseBulkDelete(hbaseContext,
                    tableName,
                    putRecord => new Delete(putRecord),
                    4)
```

HBase-Spark Module

```
val hbaseContext = new HBaseContext(sc, config)

rdd.hbaseBulkDelete(tableName)
```

HBase-Spark Module

```
val hbaseContext = new HBaseContext(sc, config)
rdd.hbaseForeachPartition(hbaseContext, (it, conn) => {
      val bufferedMutator = conn.getBufferedMutator(TableName.valueOf("t1"))
      ...
      bufferedMutator.flush()
      bufferedMutator.close()
    })
val getRdd = rdd.hbaseMapPartitions(hbaseContext, (it, conn) => {
        val table = conn.getTable(TableName.valueOf("t1"))
        var res = mutable.MutableList[String]()
        ...
      })
```

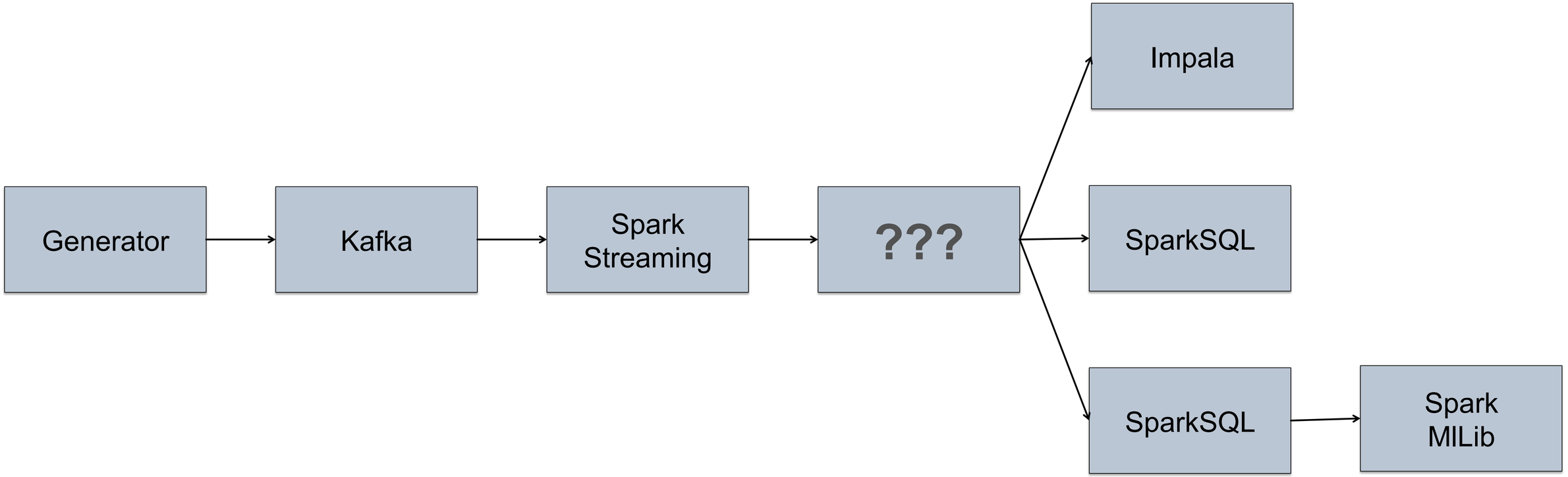
HBase-Spark Module

```
rdd.hbaseBulkLoad (tableName,
                t => {
               Seq((new KeyFamilyQualifier(t.rowKey, t.family,
t.qualifier), t.value)).
               iterator
               },
      stagingFolder)
```

HBase-Spark Module

```
val df = sqlContext.load("org.apache.hadoop.hbase.spark",
      Map("hbase.columns.mapping" -> "KEY_FIELD STRING :key, A_FIELD STRING c:a,
B_FIELD STRING c:b,",
      "hbase.table" -> "t1"))
df.registerTempTable("hbaseTmp")


sqlContext.sql("SELECT KEY_FIELD FROM hbaseTmp " +
      "WHERE " +
      "(KEY_FIELD = 'get1' and B_FIELD < '3') or " +
      "(KEY_FIELD <= 'get3' and B_FIELD = '8')").foreach(r => println(" - " + r))
```

Demo Architecture



KafkaStreams

- *Event-at-a-time processing*
- *State-full processing*
- *Windowing with out-of-order data*
- *Auto scaling / fault tolerant / reprocessing / etc*

KafkaStreams Example

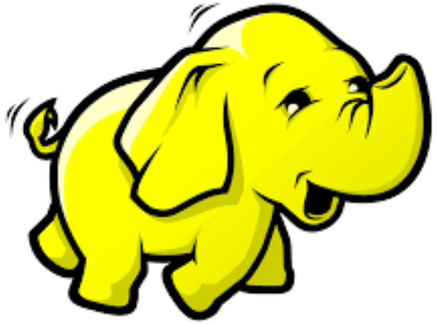
```
public static void main(String[] args) {
    // specify the processing topology by first reading in a stream from a topic
    Kstream<String, String> words = builder.stream("topic1");

    // count the words in this stream as an aggregated table
    Ktable<String, Long> counts = words.countByKey("Counts");

    // write the result table to a new topic
    counts.to("topic2")

    // create stream processing instance and run in
    KafkaStreams streams = new KafkaStreams(builder, config);
    streams.start();
}
```

Key KafkaStreams Ideas

- Operators (join, aggregate, source, sink) are nodes in DAG
- Stateful operators are persisted to Kafka – process states are a STREAM
- Data parallelism via topic partitions
- STREAM can be a changelog of a TABLE
- TABLE is a snapshot of a STREAM
- Flow is based on event time (when the event was created)

New Hadoop Storage Option

Use case break up



Unstructured Data

Deep Storage
Scan Use Cases

Structured Data

SQL + Scan Use Cases

APACHE HBASE

Fixed Columns Schemas

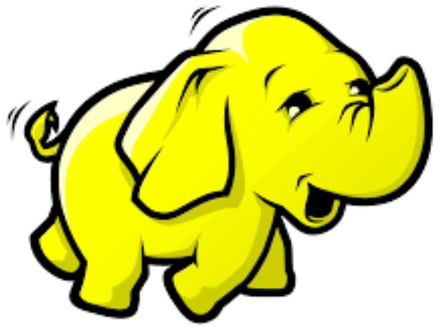
SQL + Scan Use Cases

Any Type of Column Schemas

Gets / Puts / Micro Scans

New Hadoop Storage Option

Use case break up



APACHE HBASE

Unstructured Data

Deep Storage
Scan Use Cases

Structured Data

SQL + Scan Use Cases

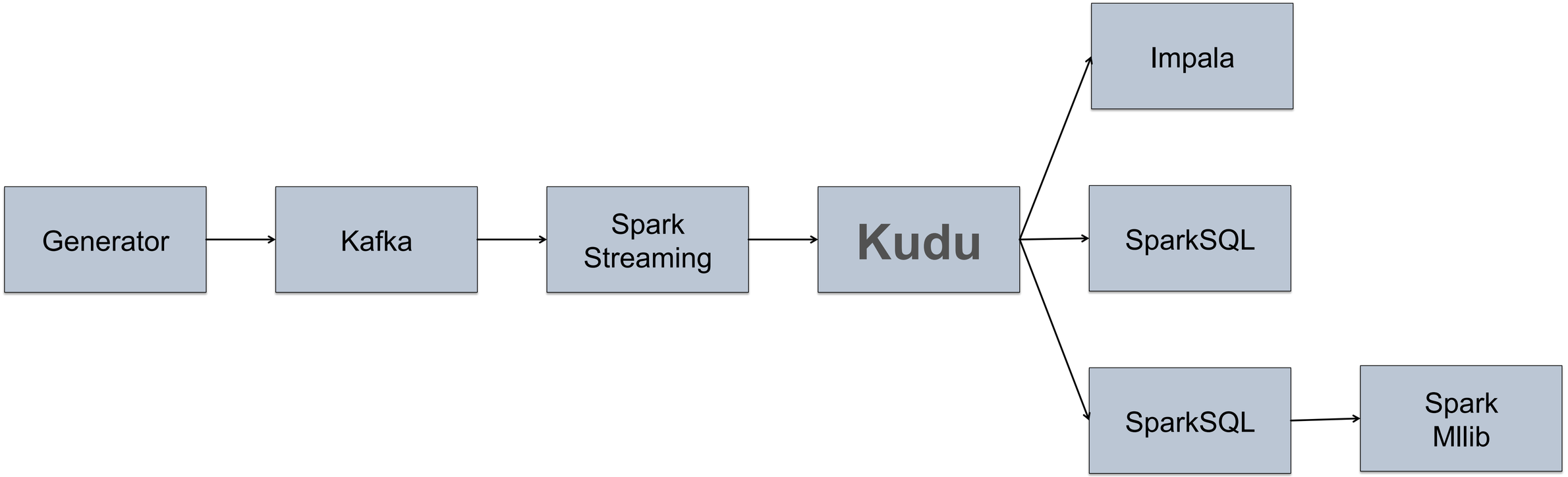
Fixed Columns Schemas

SQL + Scan Use Cases

Any Type of Column Schemas

Gets / Puts / Micro Scans

Real-Time Analytics with Kudu



Kudu and Spark

https://github.com/tmalaska/SparkOnKudu

Same Development From HBase-Spark is now on Kudu and Spark

- Full Spark RDD integration
- Full Spark Streaming integration
- Initial SparkSQL integration

Processing

Batch

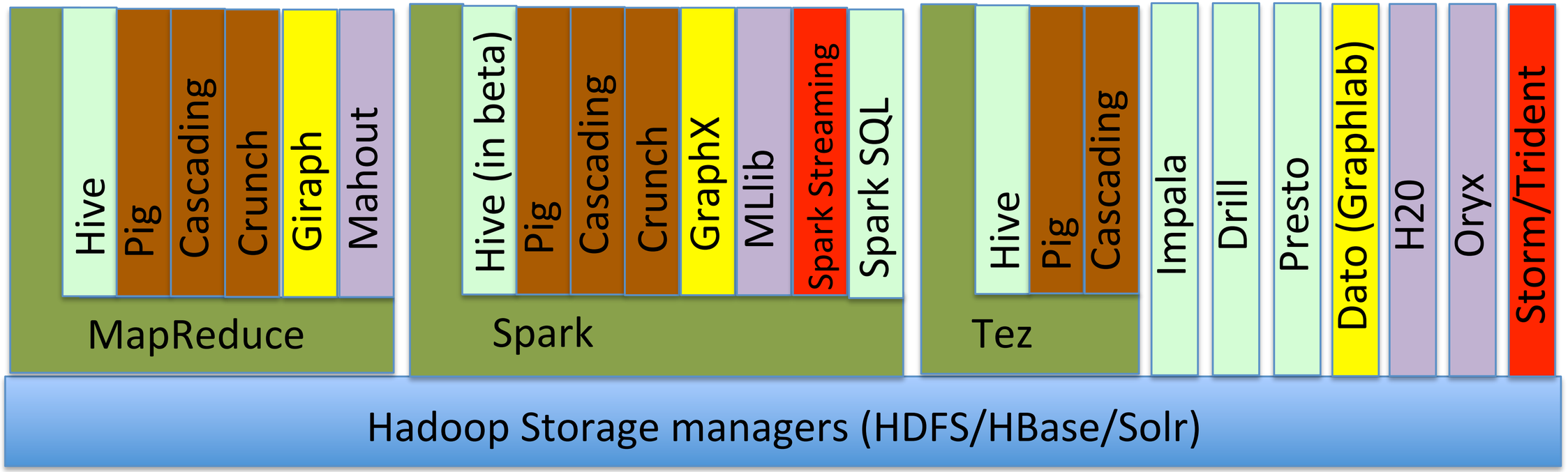
Why batch in a NRT use case?

- For background processing
  - To be later used for quick decision making
- Exploratory analysis
  - Machine Learning
- Automated rule changes
  - Based on new patterns
- Analytics
  - Who's attacking us?

Processing Engines (Past)

- Hadoop = HDFS (distributed FS) + MapReduce (Processing engine)

Processing Engines (present)



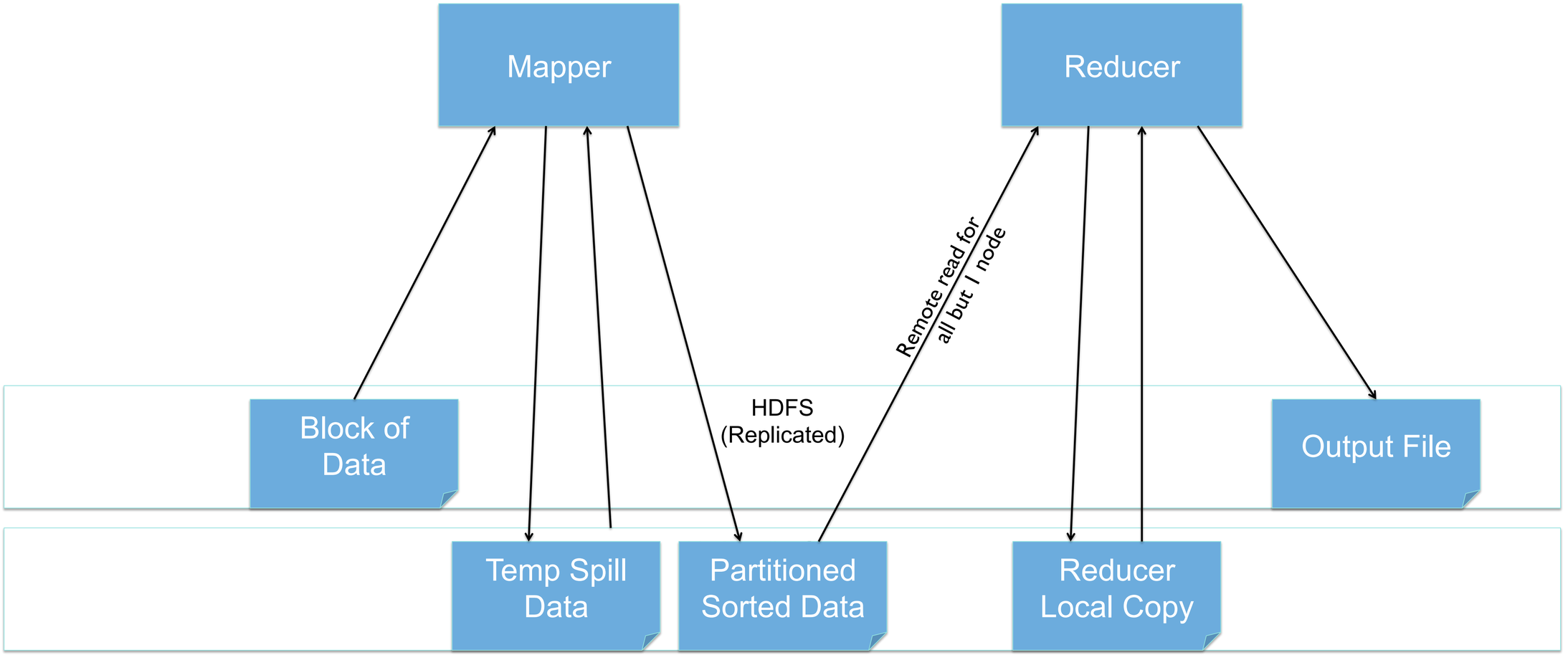
Processing Engines

- MapReduce
- Abstractions
- Spark
- Impala

MapReduce

- Oldie but goody
- Restrictive Framework / Innovative Workarounds
- Extreme Batch

MapReduce Basic High Level



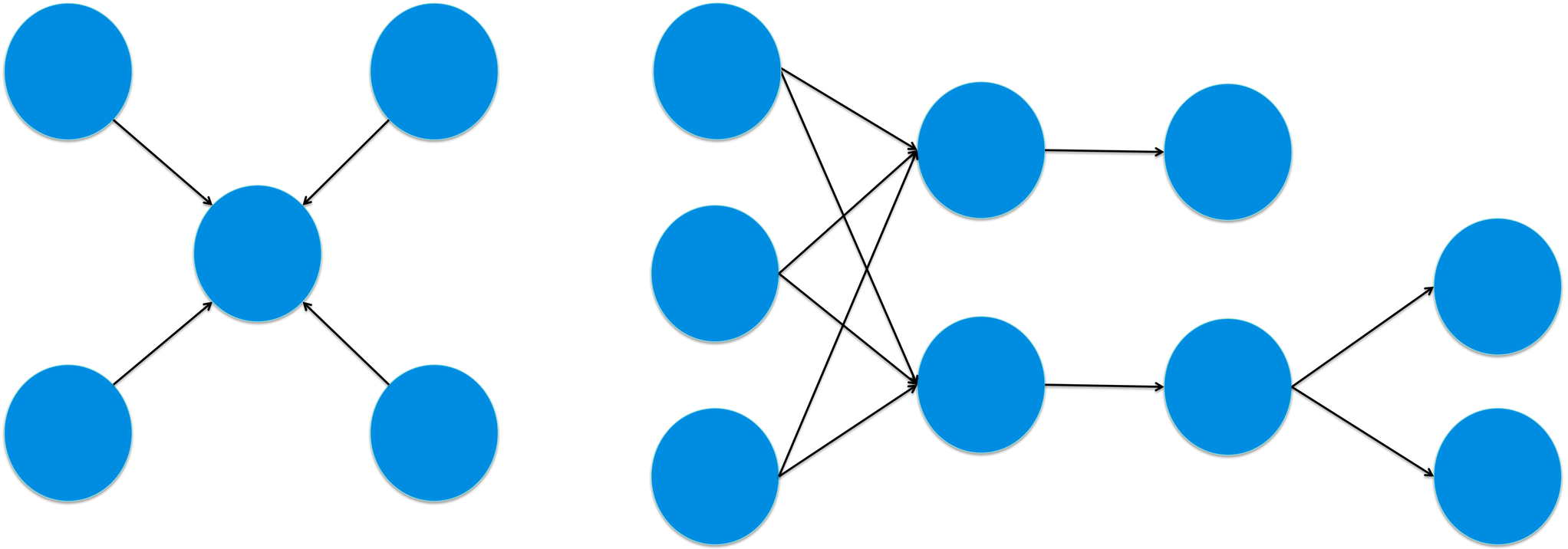
Abstractions

- SQL
  - Hive
- Script/Code
  - Pig: Pig Latin
  - Crunch: Java/Scala
  - Cascading: Java/Scala

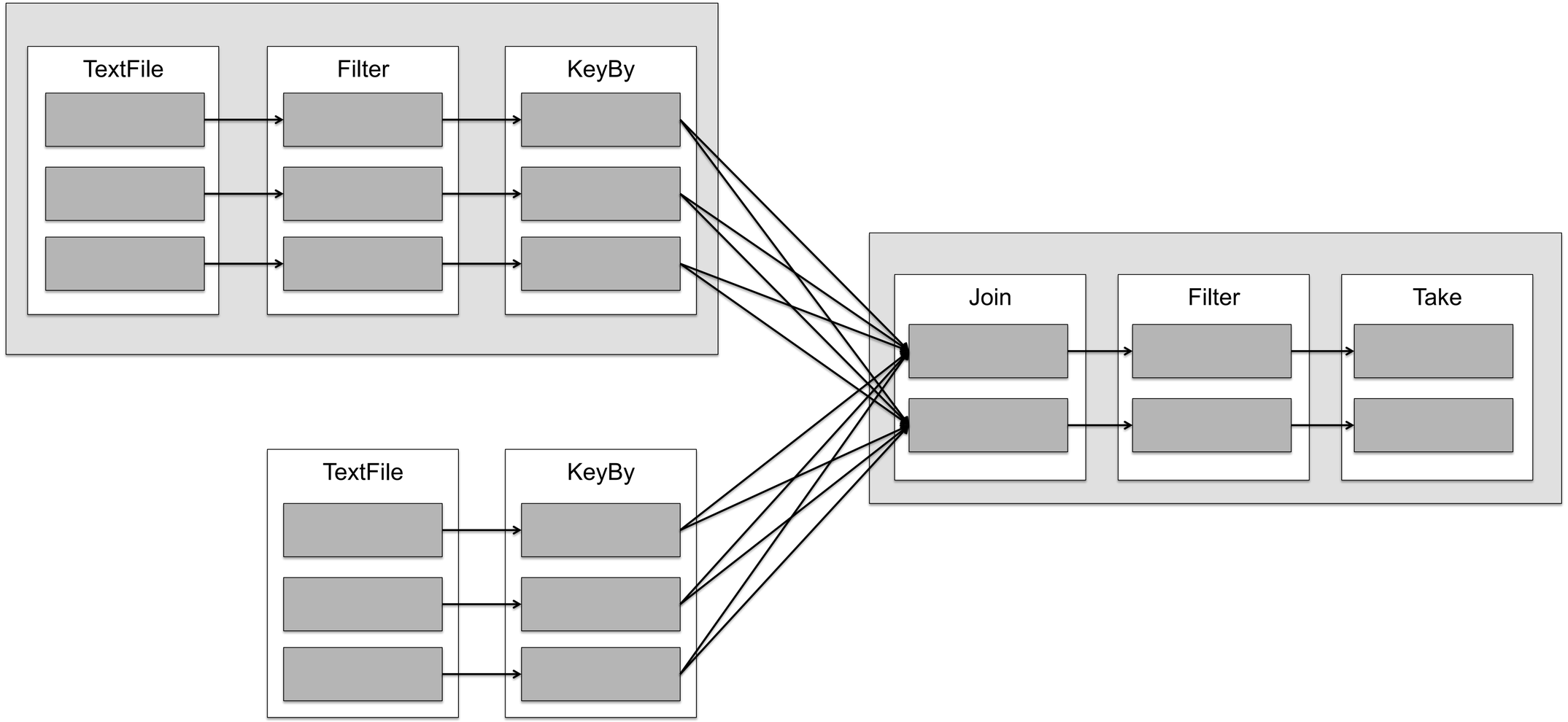
Spark

- The New Kid that isn't that New Anymore
- Easily 10x less code
- Extremely Easy and Powerful API
- Very good for iterative processing (machine learning, graph processing, etc.)
- Scala, Java, and Python support
- RDDs
- Has Graph, ML and SQL engines built on it
- R integration (SparkR)

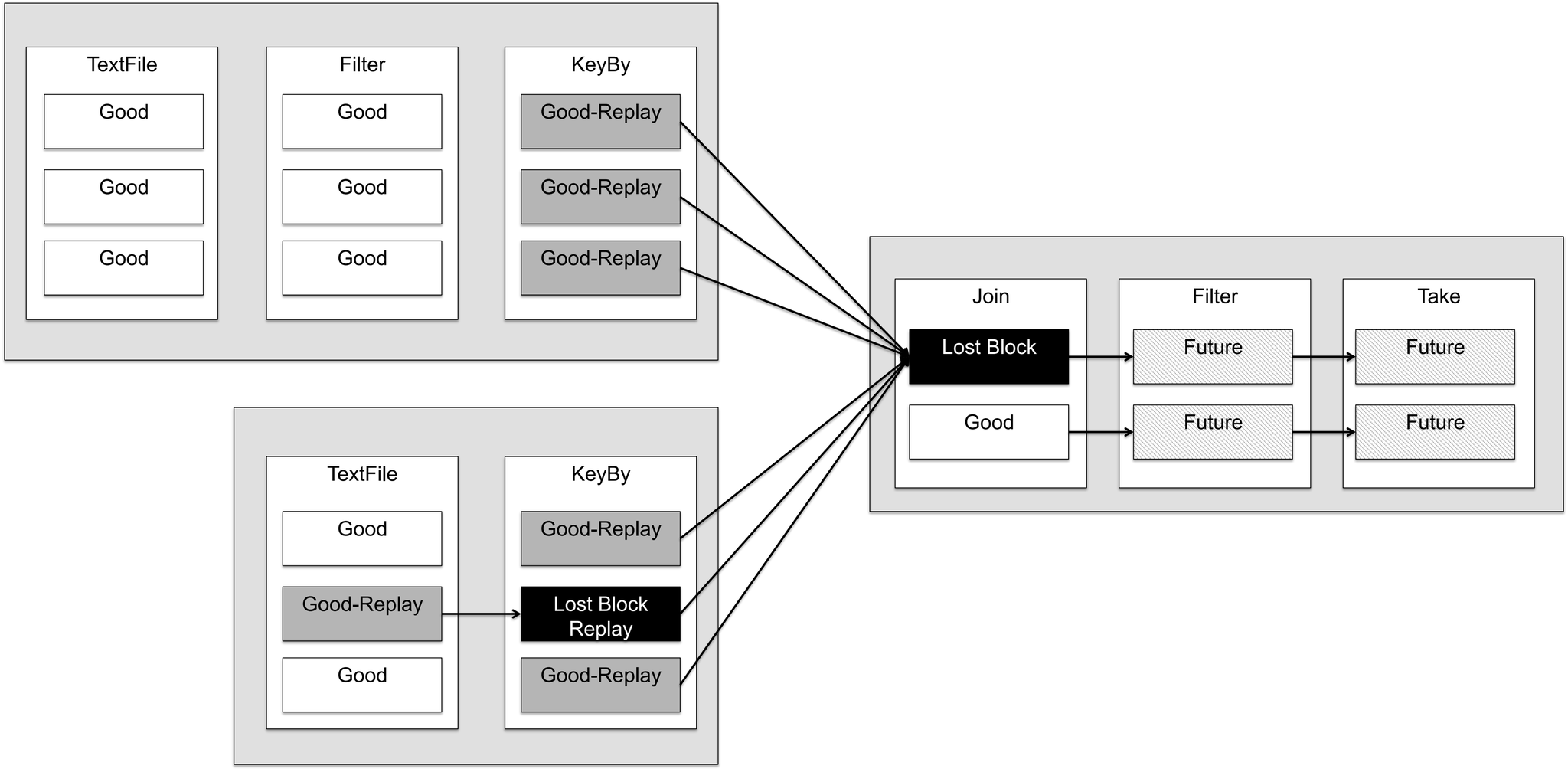
Spark - DAG



Spark - DAG



Spark - DAG


TextFile
Good
Good
Good
Filter
Good
Good
Good
KeyBy
Good-Replay
Good-Replay
Good-Replay
TextFile
Good
Good-Replay
Good
KeyBy
Good-Replay
Lost Block Replay
Good-Replay
Join
Lost Block
Good
Filter
Future
Future
Take
Future
Future

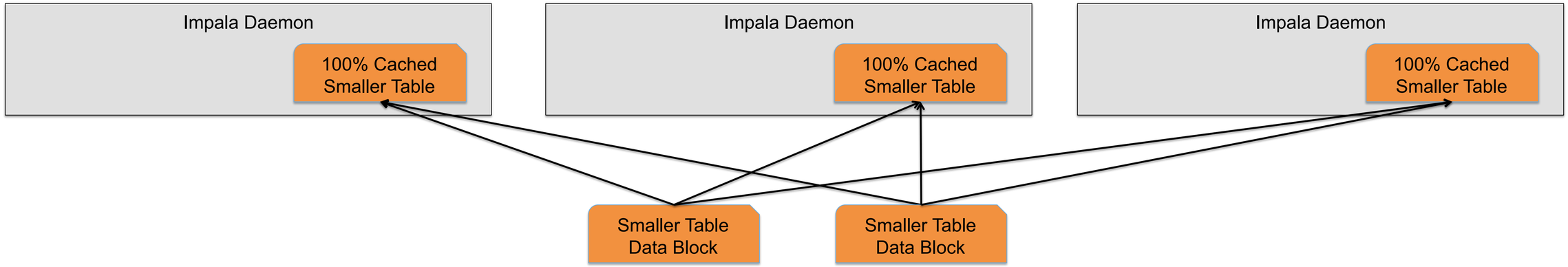
Is Spark replacing Hadoop?

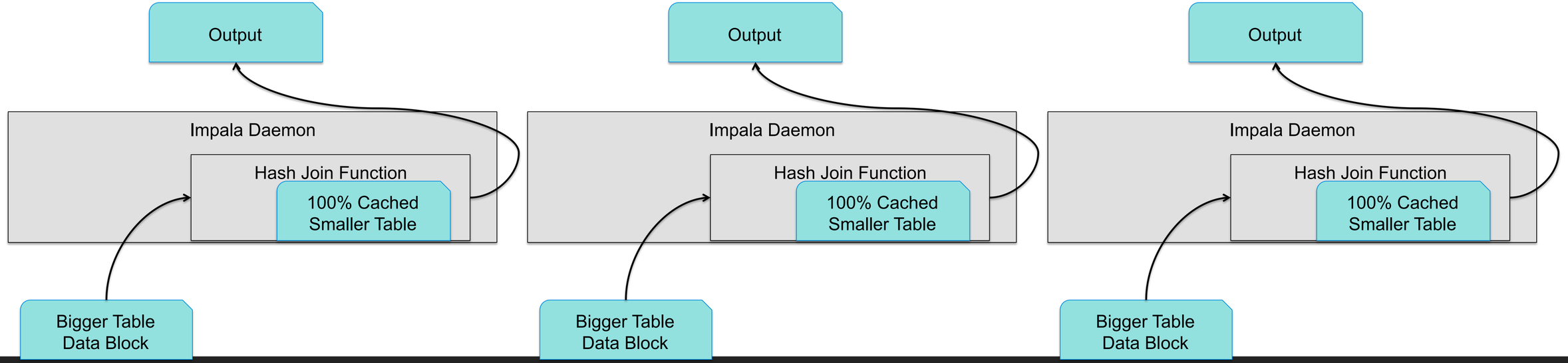
- Spark is an execution engine
  - Much faster than MR
  - Easier to program
- Spark is replacing MR

Impala

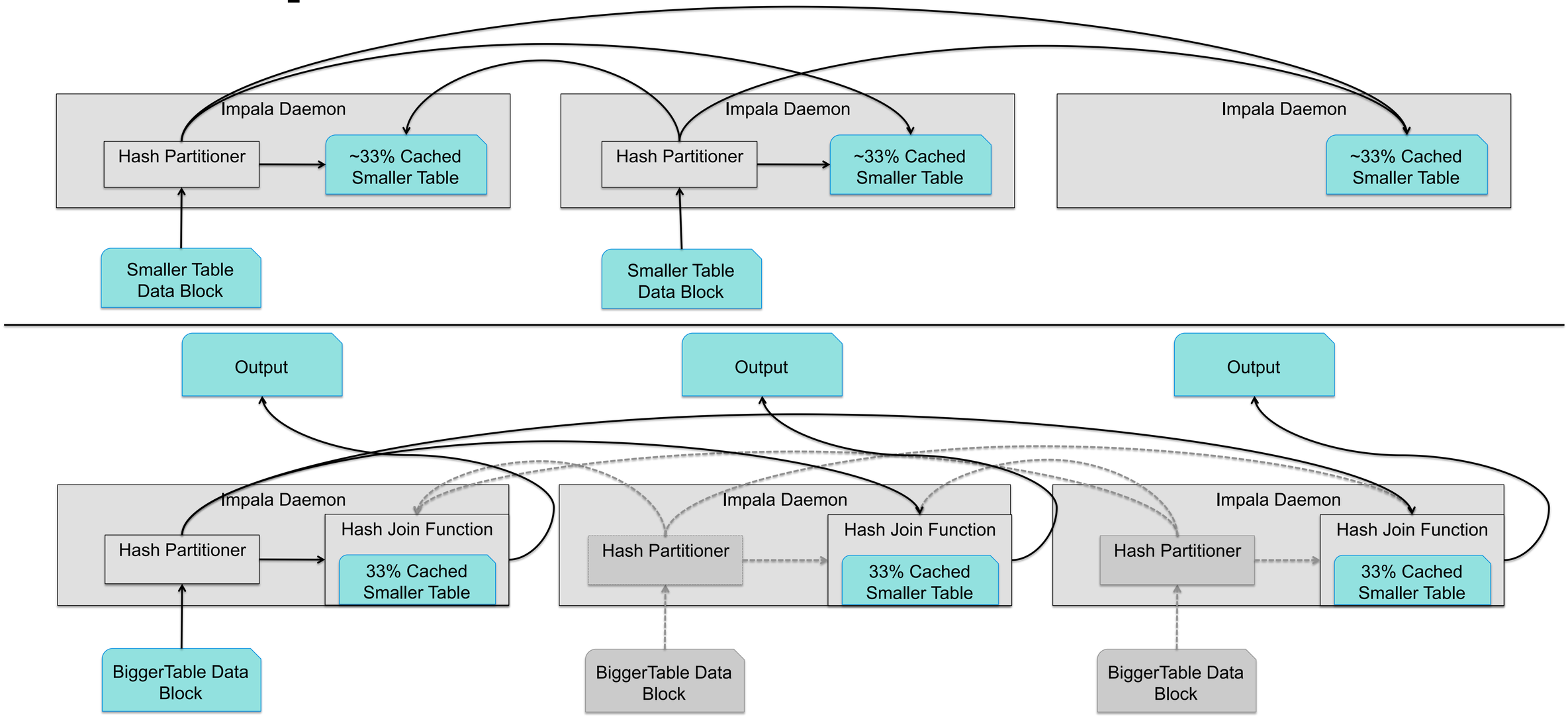
- MPP Style SQL Engine on top of Hadoop
- Very Fast
- High Concurrency
- Analytical windowing functions

Impala – Broadcast Join





Impala – Partitioned Hash Join



Presto

- Facebook's SQL engine replacement for Hive
- Written in Java
- Doesn't build on top of MapReduce
- Very few commercial vendors supporting it

Batch processing in our use-case

Batch processing in fraud-detection

- Reporting
  - How many events come daily?
  - How does it compare to last year?
- Machine Learning
  - Automated rules changes
- Other processing
  - Tracking device history and updating profiles

Reporting

- Need a fast JDBC-compliant SQL framework
- Impala or Hive or Presto?
- We choose Impala
  - Fast
  - Highly concurrent
  - JDBC/ODBC support

Machine Learning

- Options
  - MLLib (with Spark)
  - Oryx
  - H2O
  - Mahout
- We recommend MLLib because of larger use of Spark in our architecture.

Other Processing

- Options:
  - MapReduce
  - Spark
- We choose Spark
  - Much faster than MR
  - Easier to write applications due to higher level API
  - Re-use of code between streaming and batch

Strata+ Hadoop WORLD

PRESENTED BY

O'REILLY cloudera

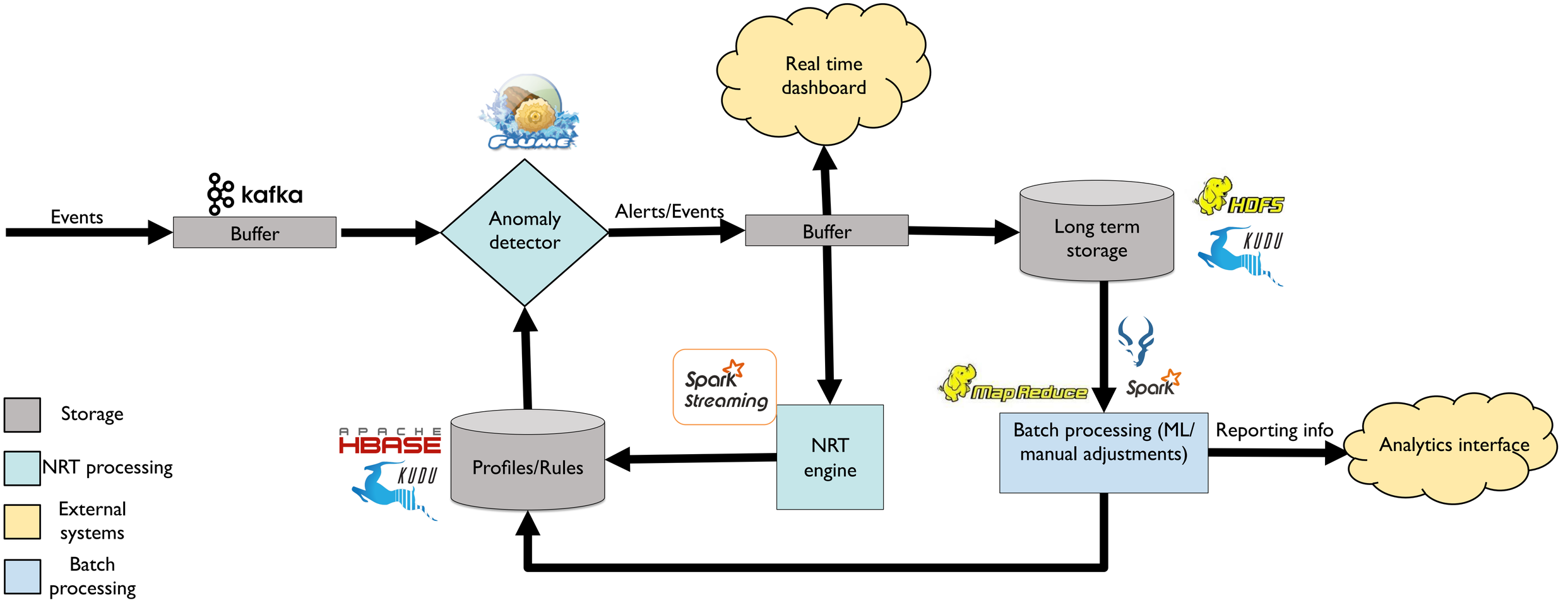
strataconf.com

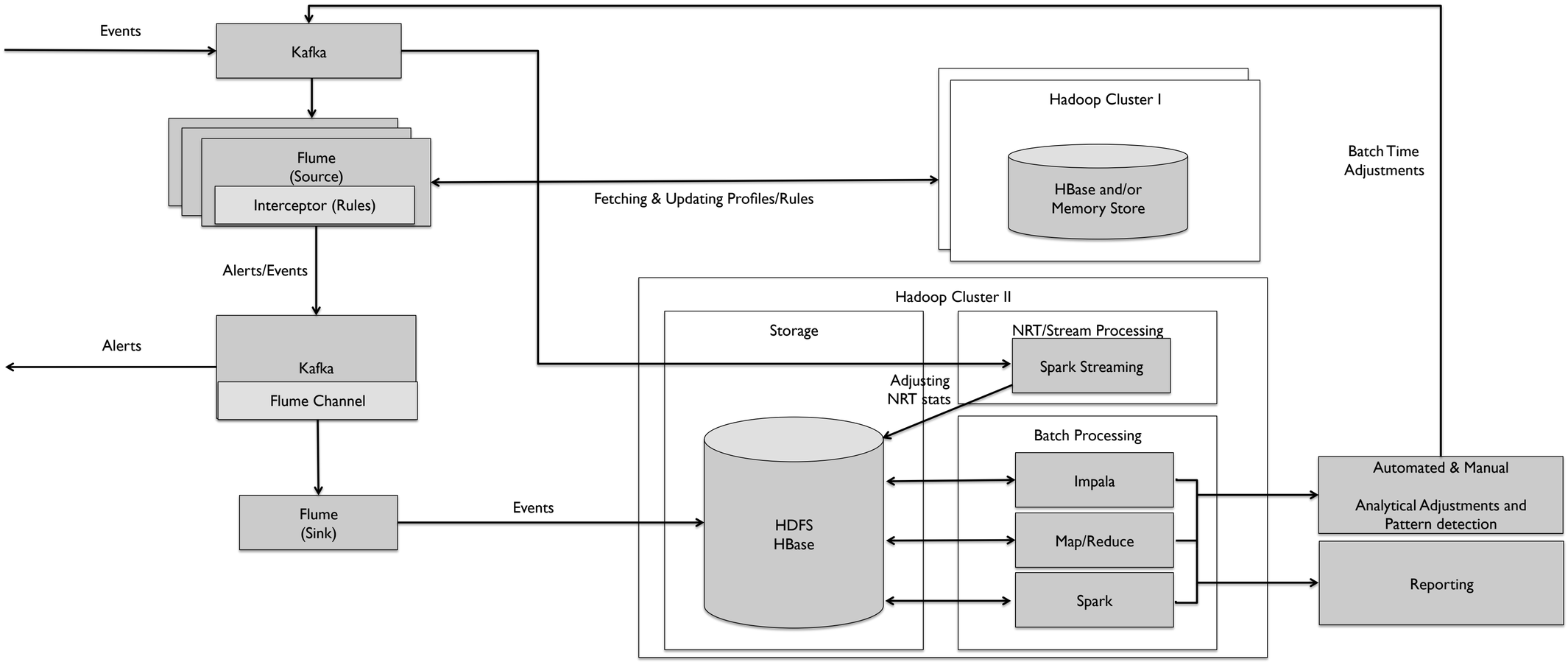
#StrataHadoop

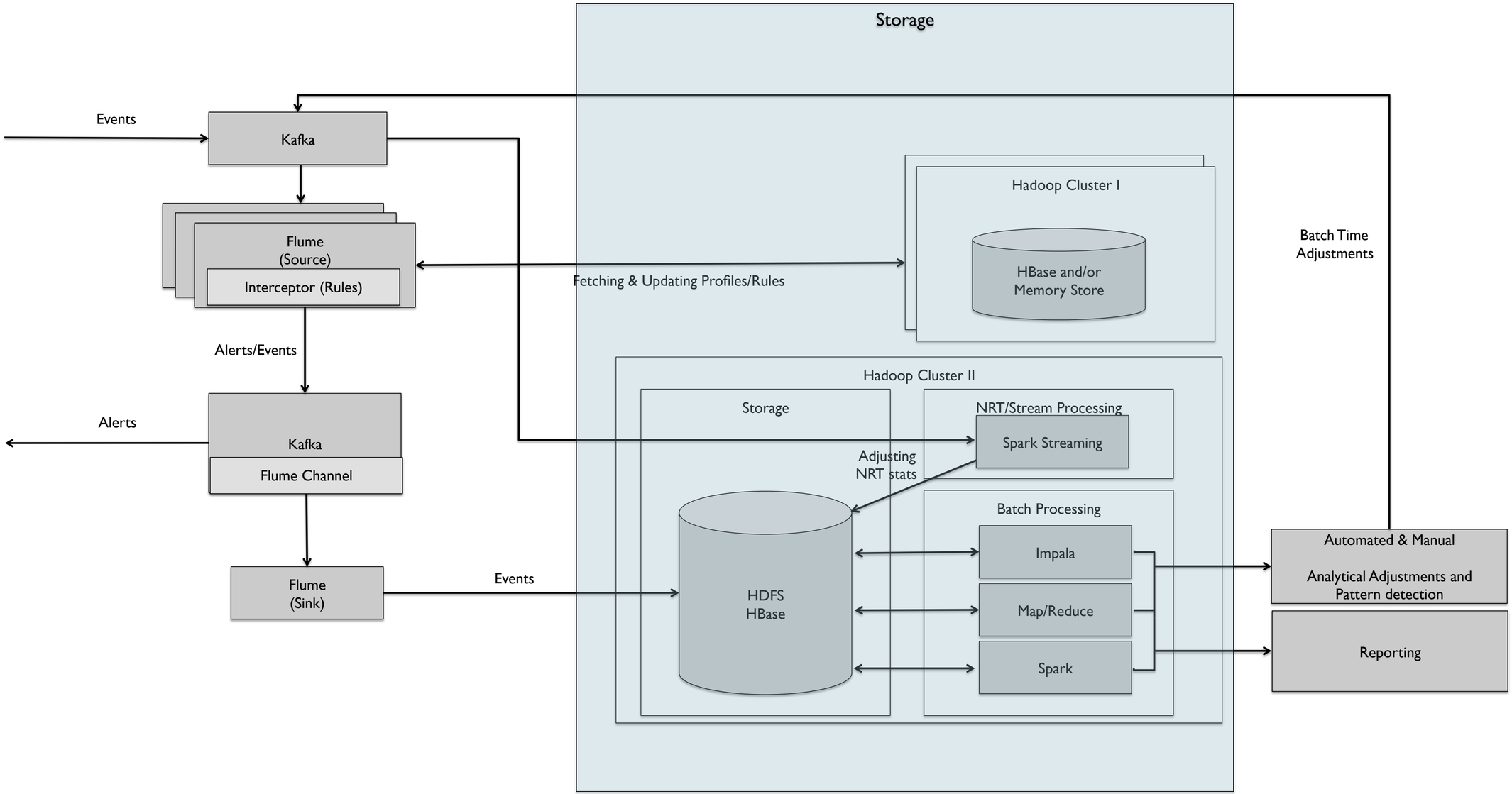
Overall Architecture

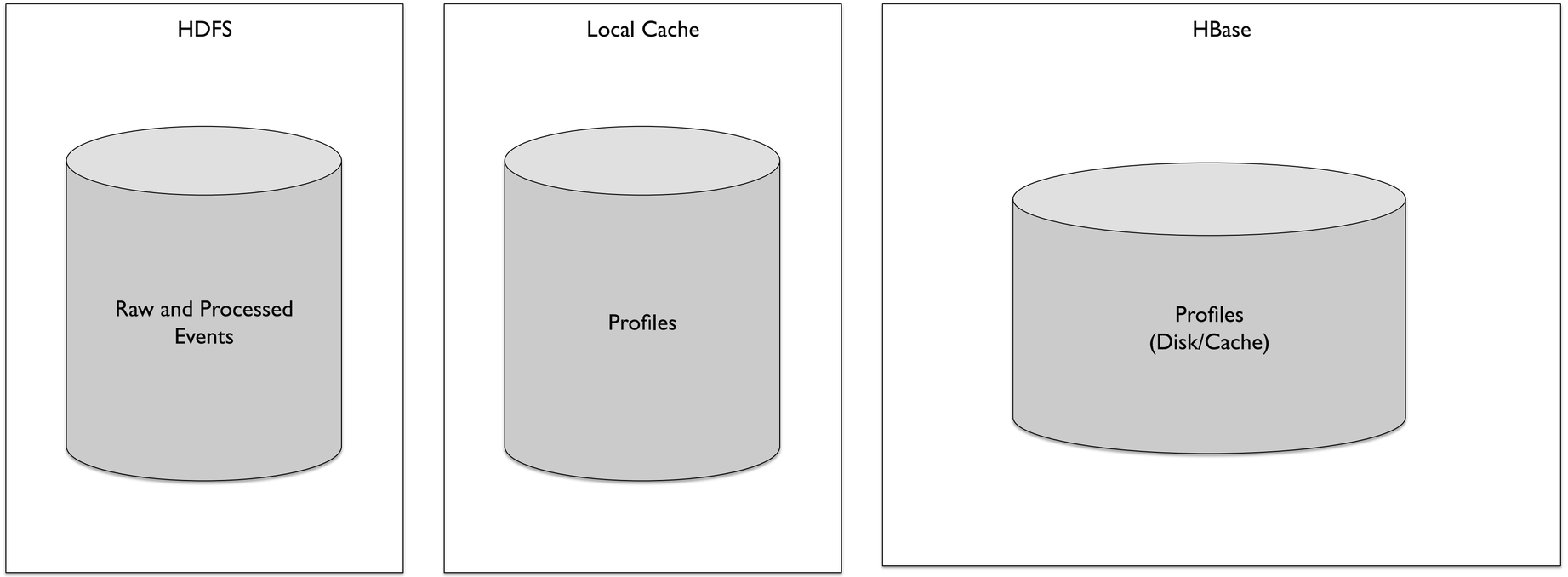
Overview

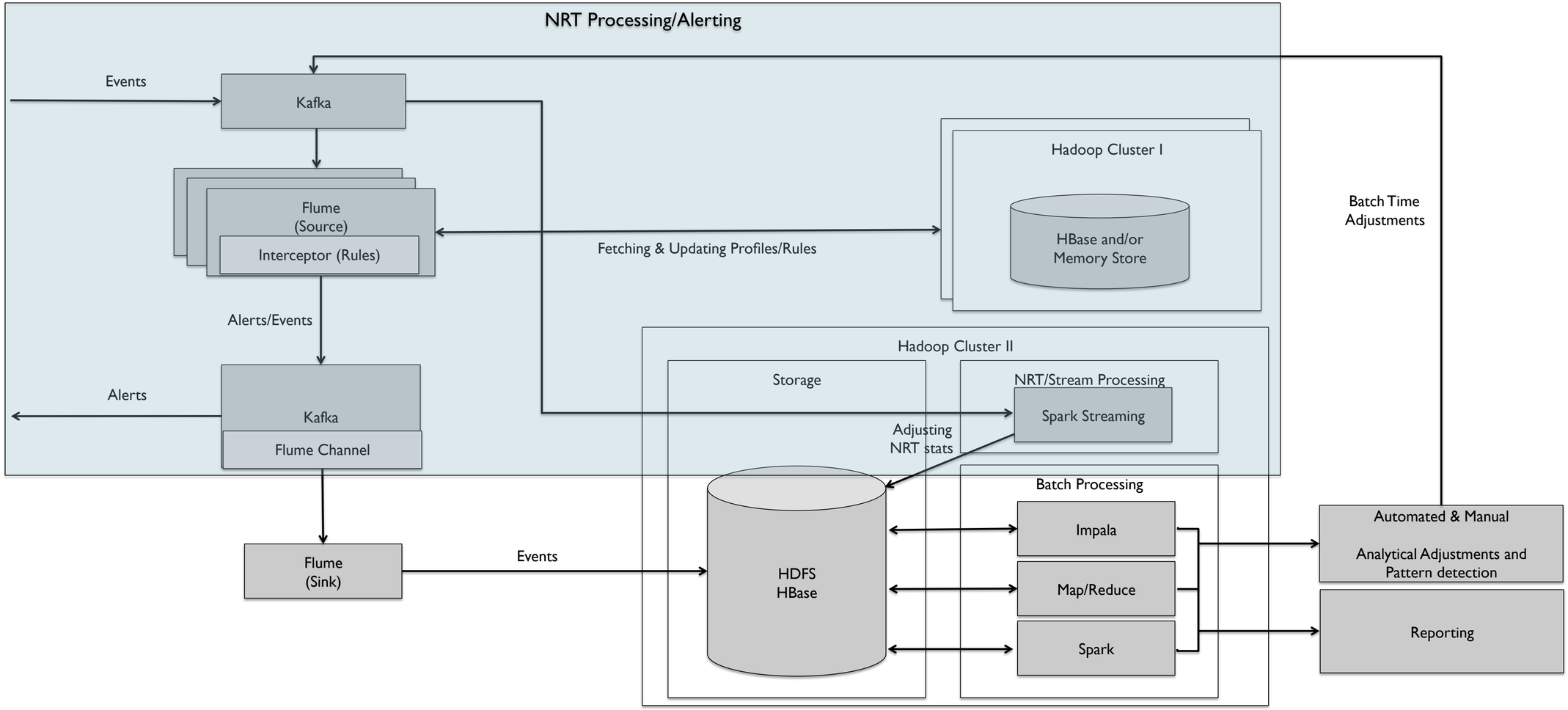
High level architecture

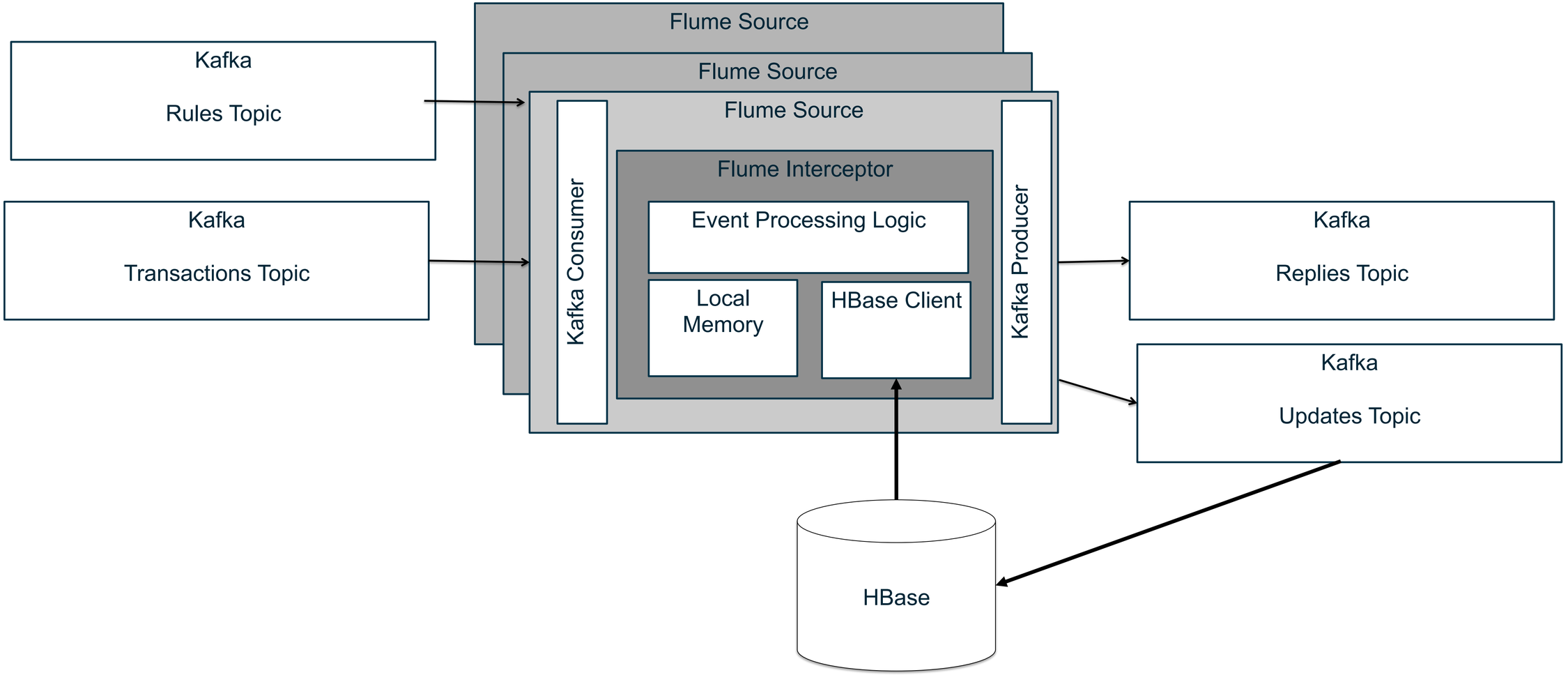


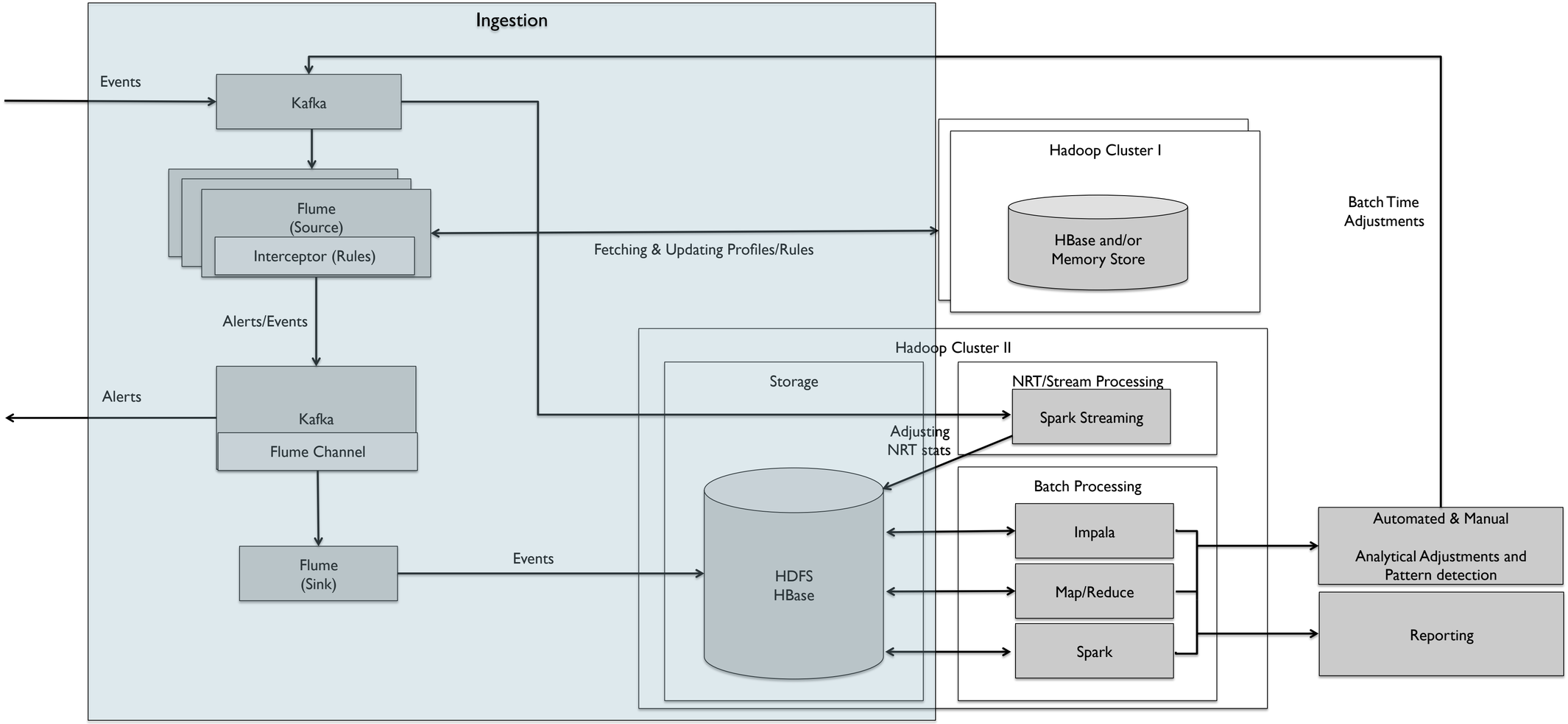

Events
Kafka
Flume
(Source)
Interceptor (Rules)
Fetching & Updating Profiles/Rules
Hadoop Cluster I
HBase and/or
Memory Store
Batch Time
Adjustments
Alerts/Events
Alerts
Kafka
Flume Channel
Hadoop Cluster II
Storage
NRT/Stream Processing
Spark Streaming
Adjusting
NRT stats
Batch Processing
Impala
Map/Reduce
Spark
HDFS
HBase
Flume
(Sink)
Events
Automated & Manual
Analytical Adjustments and
Pattern detection
Reporting

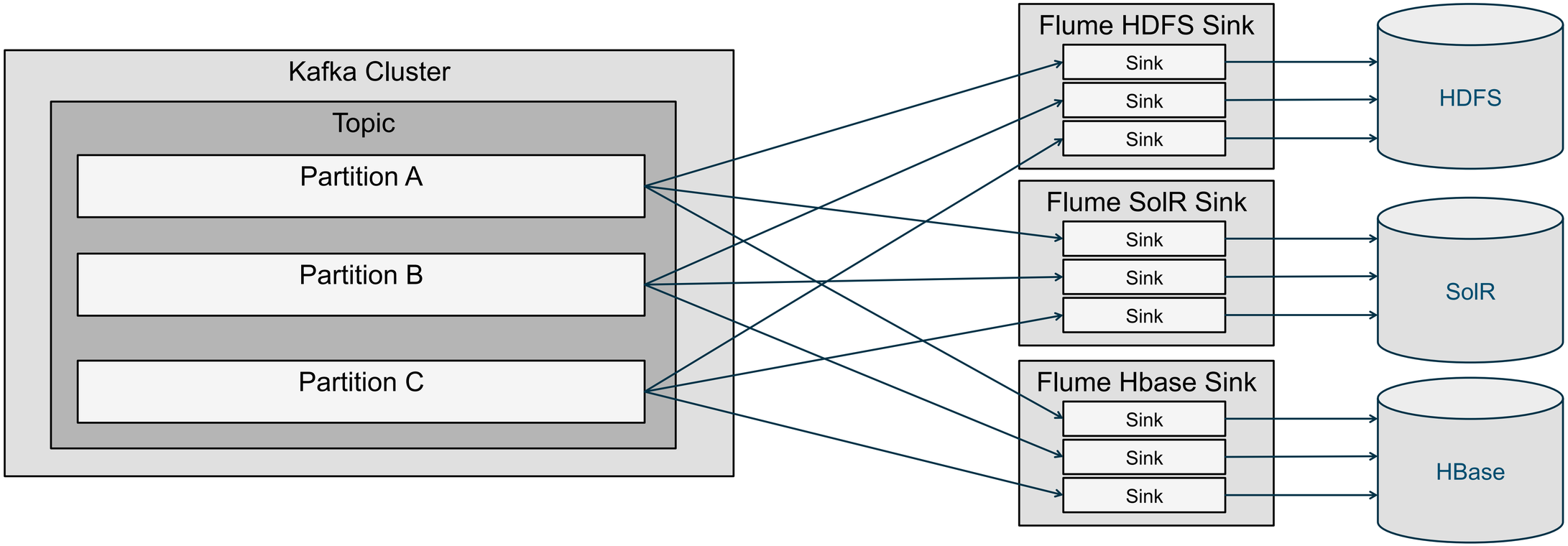

Storage
Events
Kafka
Flume
(Source)
Interceptor (Rules)
Alerts/Events
Alerts
Kafka
Flume Channel
Flume
(Sink)
Events
Fetching & Updating Profiles/Rules
Hadoop Cluster I
HBase and/or
Memory Store
Batch Time
Adjustments
Hadoop Cluster II
Storage
HDFS
HBase
NRT/Stream Processing
Spark Streaming
Adjusting
NRT stats
Batch Processing
Impala
Map/Reduce
Spark
Automated & Manual
Analytical Adjustments and
Pattern detection
Reporting

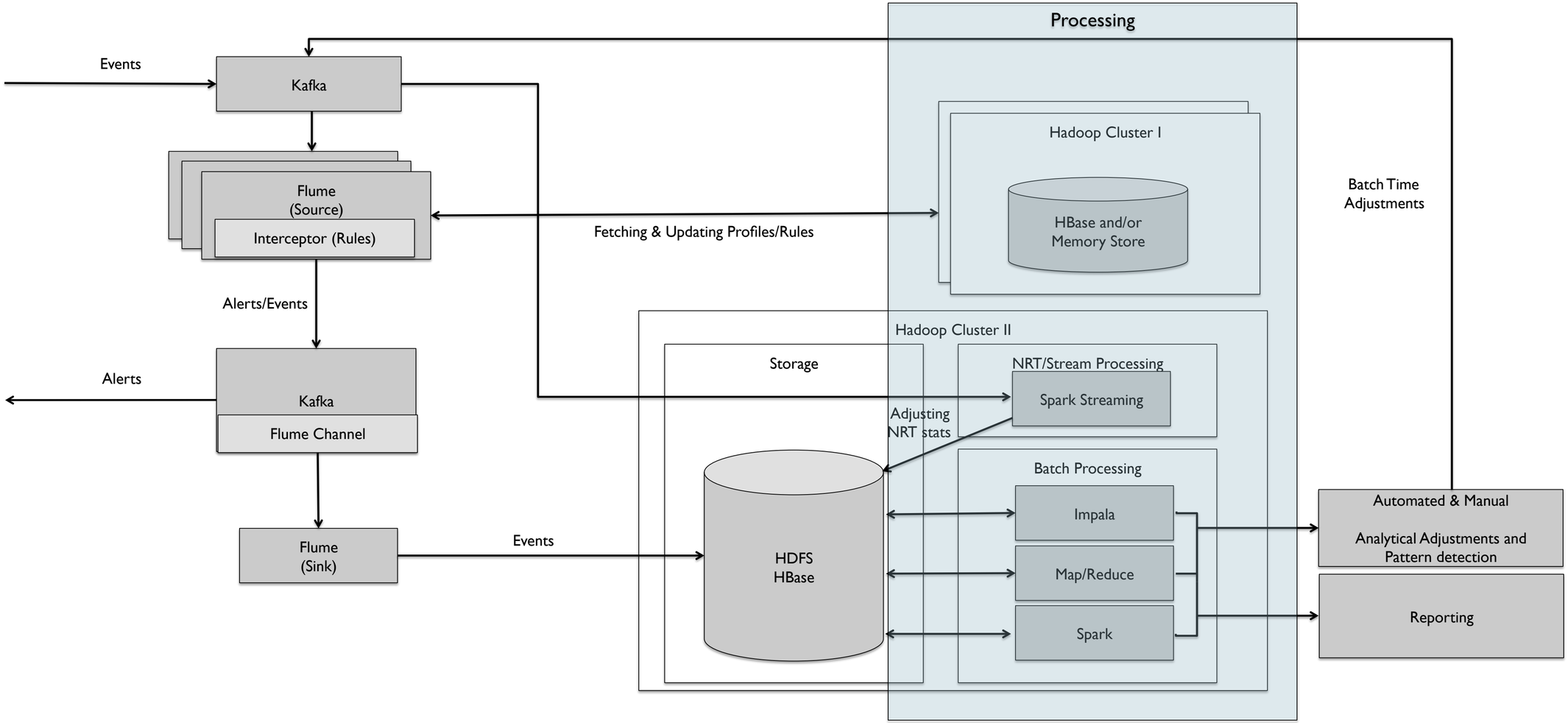

HDFS
Raw and Processed Events
Local Cache
Profiles
HBase
Profiles (Disk/Cache)

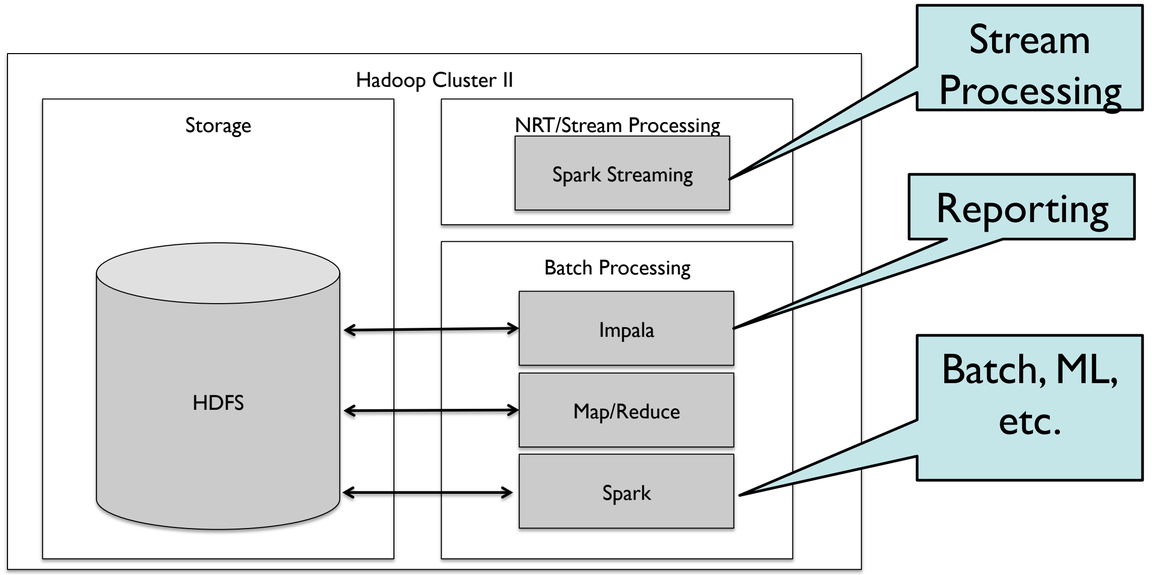

NRT Processing/Alerting
Events
Kafka
Flume
(Source)
Interceptor (Rules)
Fetching & Updating Profiles/Rules
Hadoop Cluster I
HBase and/or
Memory Store
Batch Time
Adjustments
Alerts/Events
Hadoop Cluster II
Storage
NRT/Stream Processing
Spark Streaming
Alerts
Kafka
Flume Channel
Adjusting
NRT stats
Batch Processing
Impala
Map/Reduce
Spark
Automated & Manual
Analytical Adjustments and
Pattern detection
Reporting
Flume
(Sink)
Events
HDFS
HBase


Kafka
Rules Topic
Kafka
Transactions Topic
Flume Source
Flume Source
Flume Source
Kafka Consumer
Flume Interceptor
Event Processing Logic
Local Memory
HBase Client
Kafka Producer
Kafka
Replies Topic
Kafka
Updates Topic
HBase


Ingestion
Events
Kafka
Flume
(Source)
Interceptor (Rules)
Fetching & Updating Profiles/Rules
Hadoop Cluster I
HBase and/or
Memory Store
Batch Time
Adjustments
Alerts/Events
Hadoop Cluster II
Storage
NRT/Stream Processing
Spark Streaming
Alerts
Kafka
Flume Channel
Adjusting
NRT stats
Batch Processing
Impala
Map/Reduce
Spark
Automated & Manual
Analytical Adjustments and
Pattern detection
Reporting
Flume
(Sink)
Events
HDFS
HBase


Kafka Cluster
Topic
Partition A
Partition B
Partition C
Flume HDFS Sink
Sink
Sink
Sink
Flume SolR Sink
Sink
Sink
Sink
Flume Hbase Sink
Sink
Sink
Sink
HDFS
SolR
HBase


Processing
Events
Kafka
Flume
(Source)
Interceptor (Rules)
Fetching & Updating Profiles/Rules
Hadoop Cluster I
HBase and/or
Memory Store
Batch Time
Adjustments
Alerts/Events
Hadoop Cluster II
Storage
NRT/Stream Processing
Spark Streaming
Alerts
Kafka
Flume Channel
Adjusting
NRT stats
Batch Processing
Impala
Automated & Manual
Analytical Adjustments and
Pattern detection
HDFS
HBase
Map/Reduce
Flume
(Sink)
Events
Spark
Reporting


Hadoop Cluster II
Storage
HDFS
NRT/Stream Processing
Spark Streaming
Batch Processing
Impala
Map/Reduce
Spark
Stream Processing
Reporting
Batch, ML, etc.

Strata+ Hadoop WORLD

PRESENTED BY

O'REILLY cloudera

strataconf.com

#StrataHadoop

Demo!

Strata+ Hadoop WORLD

PRESENTED BY

O'REILLY cloudera

strataconf.com

#StrataHadoop

Where else to find us?

Book Signing!

- This evening at 5:35 PM at the Cloudera booth.

Other Sessions

- Ask Us Anything session (all) – Thursday, 2:05 pm
- Reliability guarantees in Kafka (Gwen) – Thursday, 12:05 pm
- Putting Kafka into overdrive (Gwen) – Thursday, 5:25 pm
- Introduction to Apache Spark for Java and Scala developers (Ted) – Friday, 2:05 pm

Strata+ Hadoop WORLD

PRESENTED BY

O'REILLY cloudera

strataconf.com

#StrataHadoop

Thank you!

@hadooparchbook
tiny.cloudera.com/app-arch-london
Gwen Shapira | @gwenshap
Jonathan Seidman | @jseidman
Ted Malaska | @ted_malaska
Mark Grover | @mark_grover