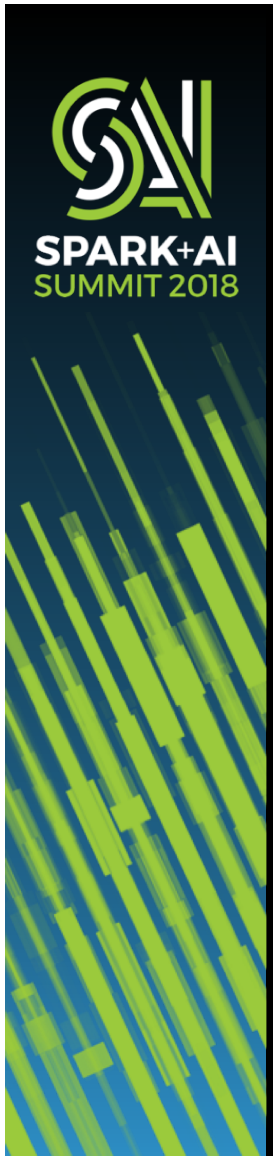


Scale a Near Real-Time AI System by 4X and Beyond with Apache Spark

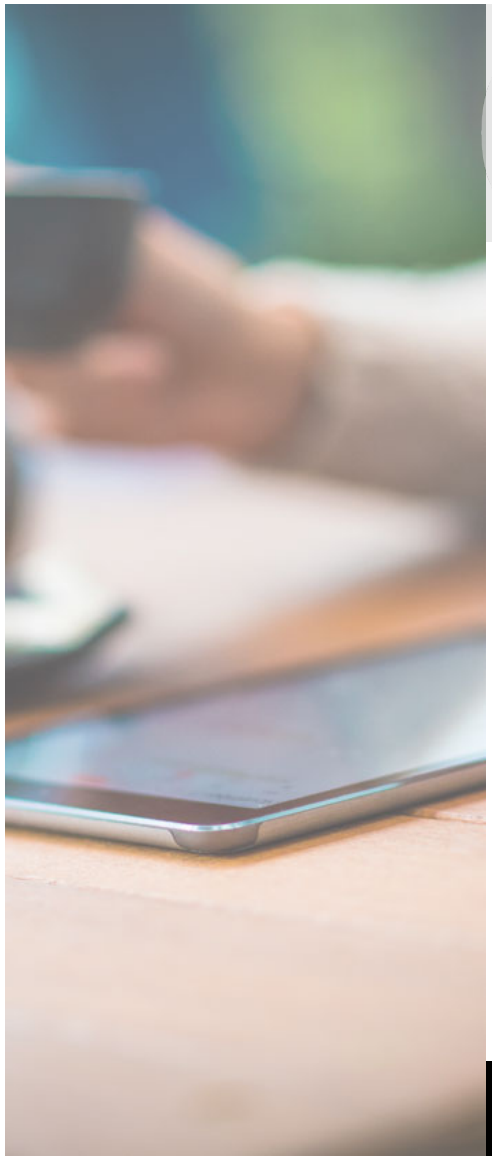
Yan Li, Conviva

Shubo Liu, Conviva

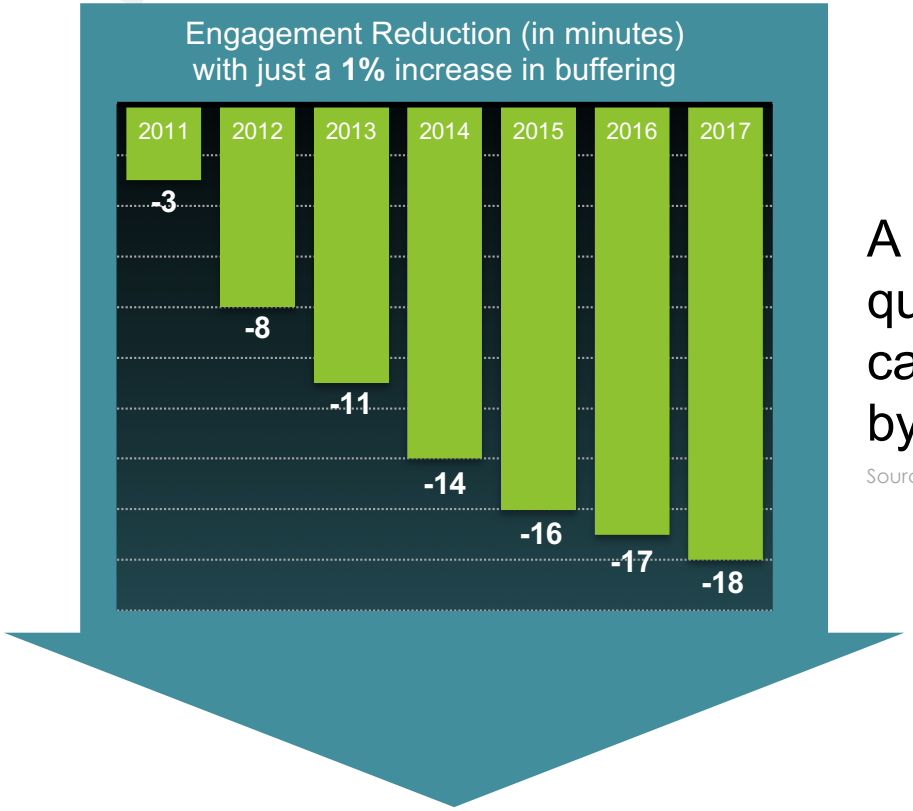
#ExpSAIS14

A service for Internet Video Streaming quality analytics that **in near real-time**

- **Detect anomalies** in quality of viewing experience from massive number of potential issues
- **Diagnose root cause** among all anomalies



Quality Is Critical To Engagement

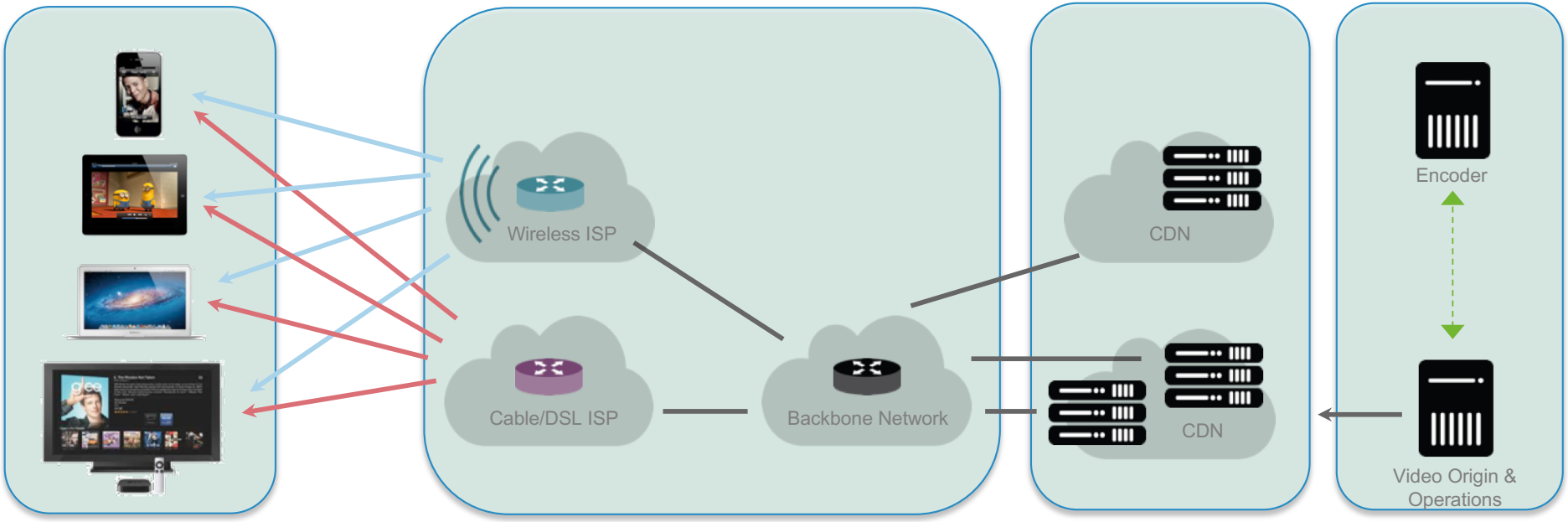


A **1%** change in quality of experience can decrease video views by **18 minutes**

Source: Conviva, "2017 OTT Streaming Market Year in Review"

Internet Video Delivery is Complicated

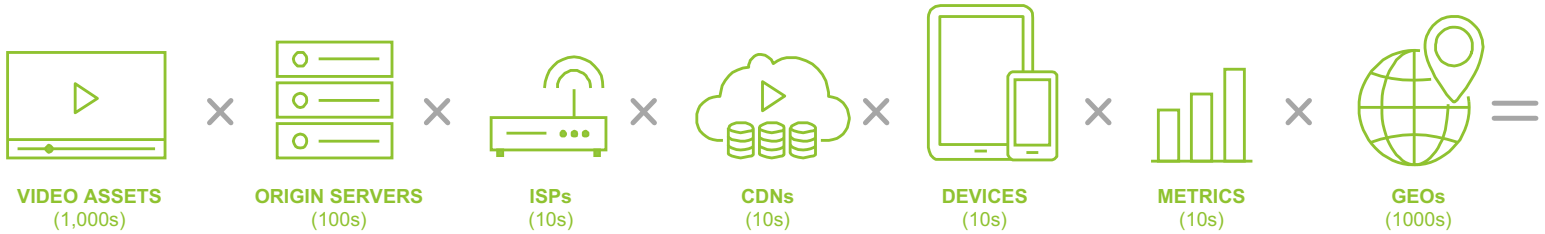
Many parties, many paths but no E2E owner



Any entity can fail any time → degradation of QoE

Internet Video Delivery is Complicated

Many parties, many paths but no E2E owner



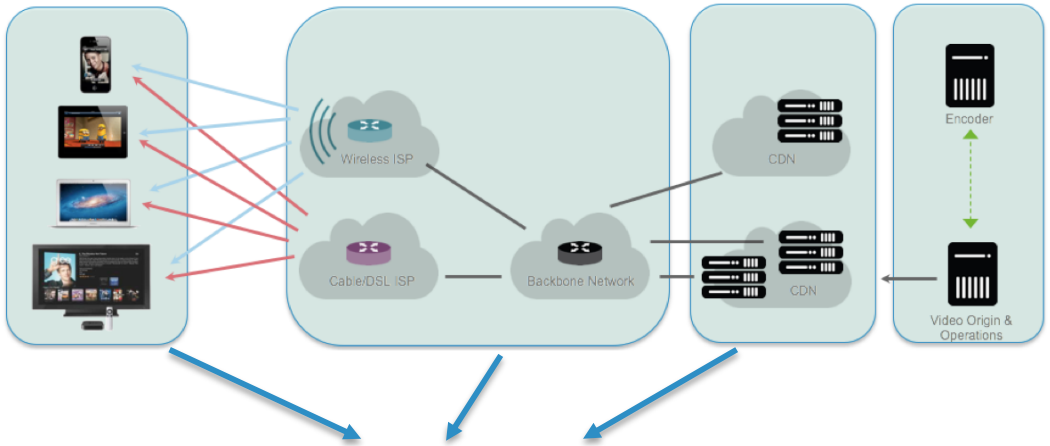
Millions

COMBINATIONS OF POSSIBLE ISSUES

Where are the issues?

What caused the issues?

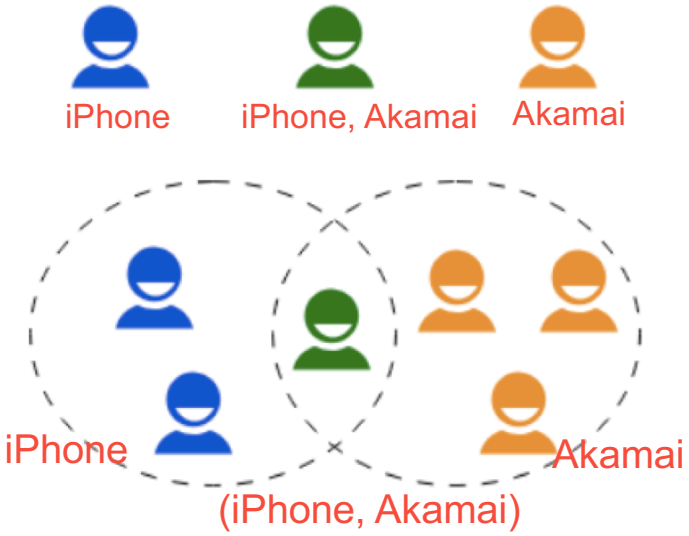
Definitions



Grouping: Device, ISP, CDN, (ISP, CDN),…

Session: data record for video session

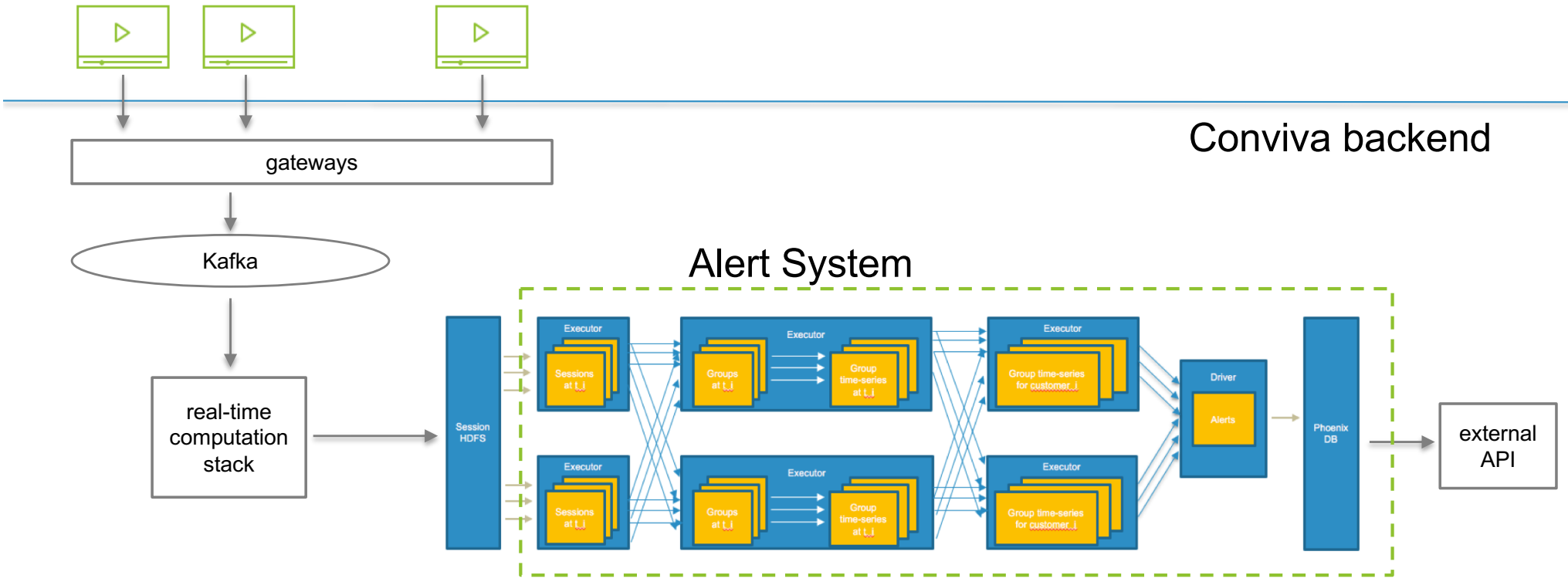
- grouping meta data + QoE measures

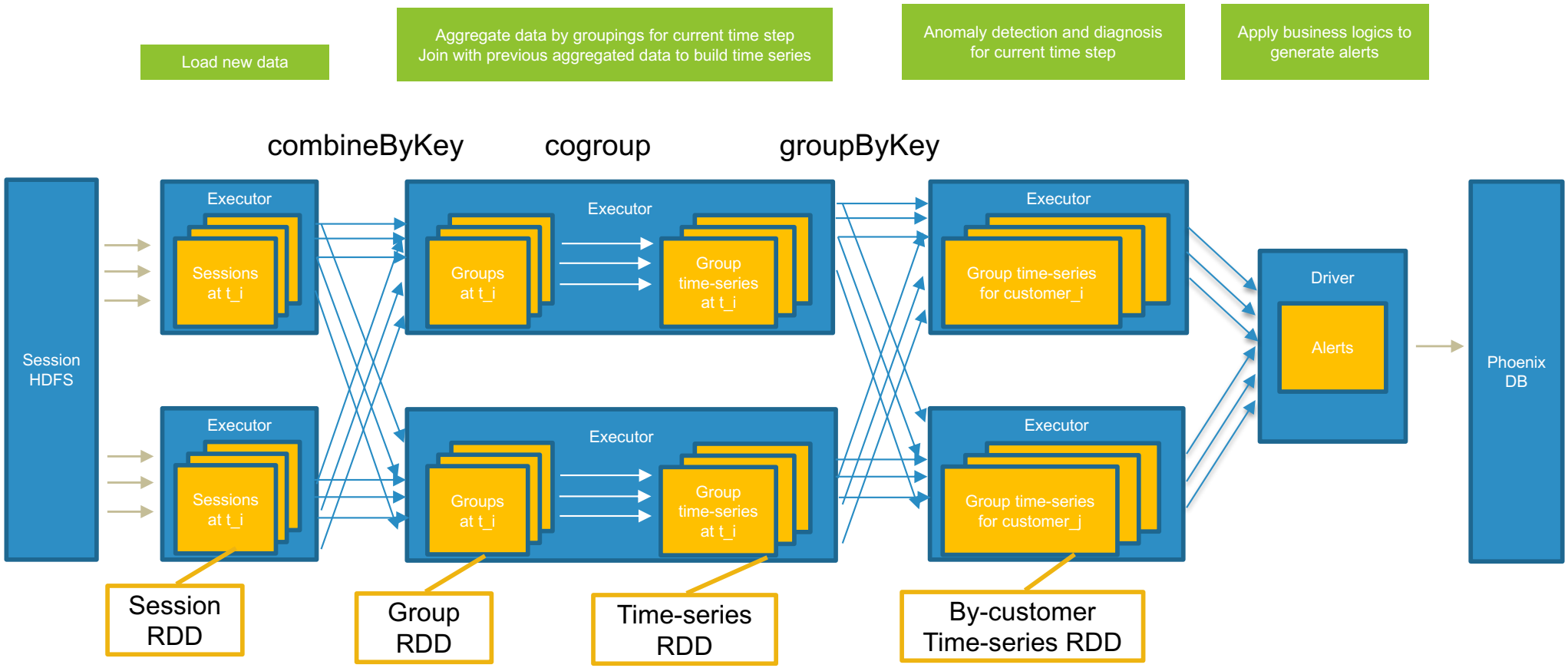


Group: aggregated QoE from sessions

- **inputs to AI Alerts algorithms**

System Architecture




Load new data
Aggregate data by groupings for current time step
Join with previous aggregated data to build time series
Anomaly detection and diagnosis for current time step
Apply business logics to generate alerts
combineByKey
cogroup
groupByKey
Session HDFS
Executor
Sessions at t_i
Executor
Sessions at t_i
Executor
Groups at t_i
Group time-series at t_i
Executor
Groups at t_i
Group time-series at t_i
Executor
Group time-series for customer_i
Executor
Group time-series for customer_j
Driver
Alerts
Phoenix DB
Session RDD
Group RDD
Time-series RDD
By-customer Time-series RDD

Main Challenges (1)

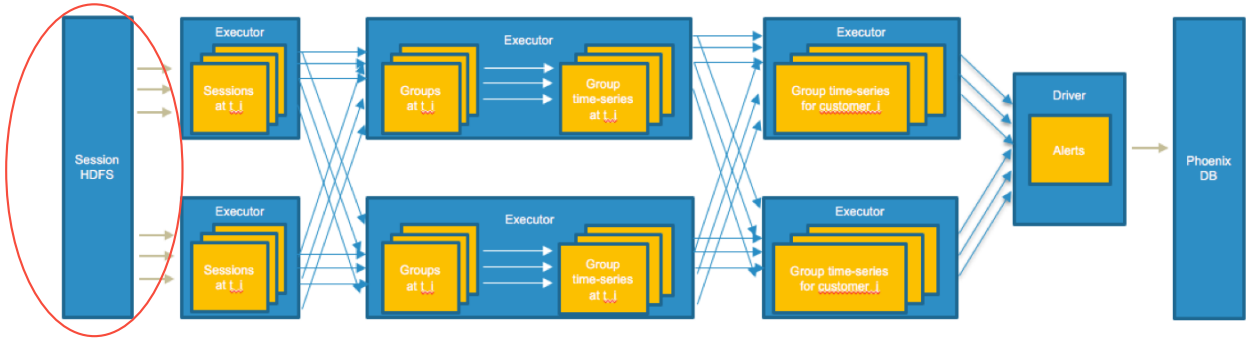
Scale, Latency, Cost

- Long running application with short latency requirement
  - Service keeps running for weeks with minimum performance degradation
  - Minimize latency to report alerts
  - Without pipelining, reading and processing 1-min data needs to take less than 1 min

Main Challenges (2)

Scale, Latency, Cost

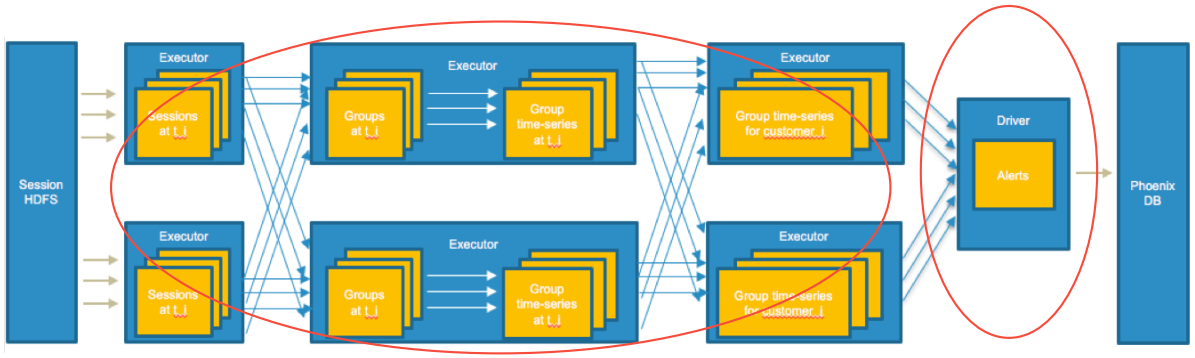
- Upstream components are not always reliable
  - Size of session data can be skewed → skewed partitions of session RDD
  - Number of session files may vary → less utilized CPU
  - HDFS data node failure/degradation → long delay in loading session data



Main Challenges (3)

Scale, Latency, Cost

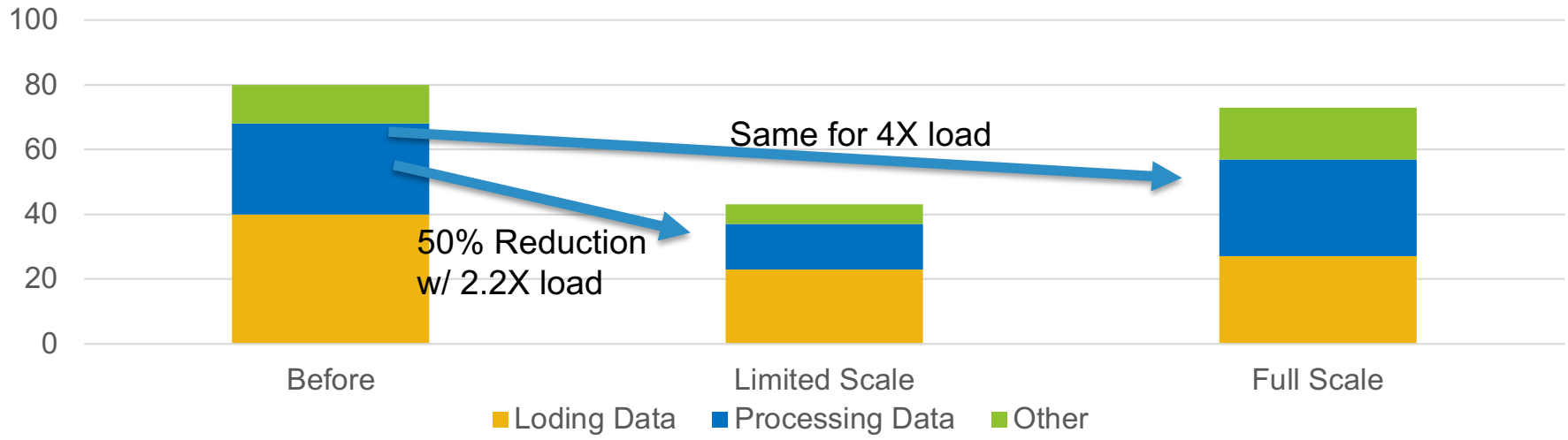
- Cloud service is not always reliable
  - Nodes in cloud can degrade or fail any time



What We Did

- Monitor both spark performance and application health for long-running spark application
- Focus on Scalability, Stability and Resilience (SSR)
  - Fast Recovery on service restart
  - Data Ingestion Optimization
- Software-level Optimization for better scalability
  - Shuffling reduction
  - SerDe Optimization

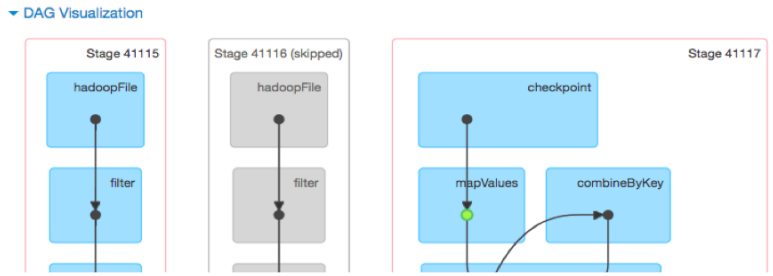
Our Performance Trajectory



All three tests are running on the same setup

Spark Application Monitor

Classic Spark Job M



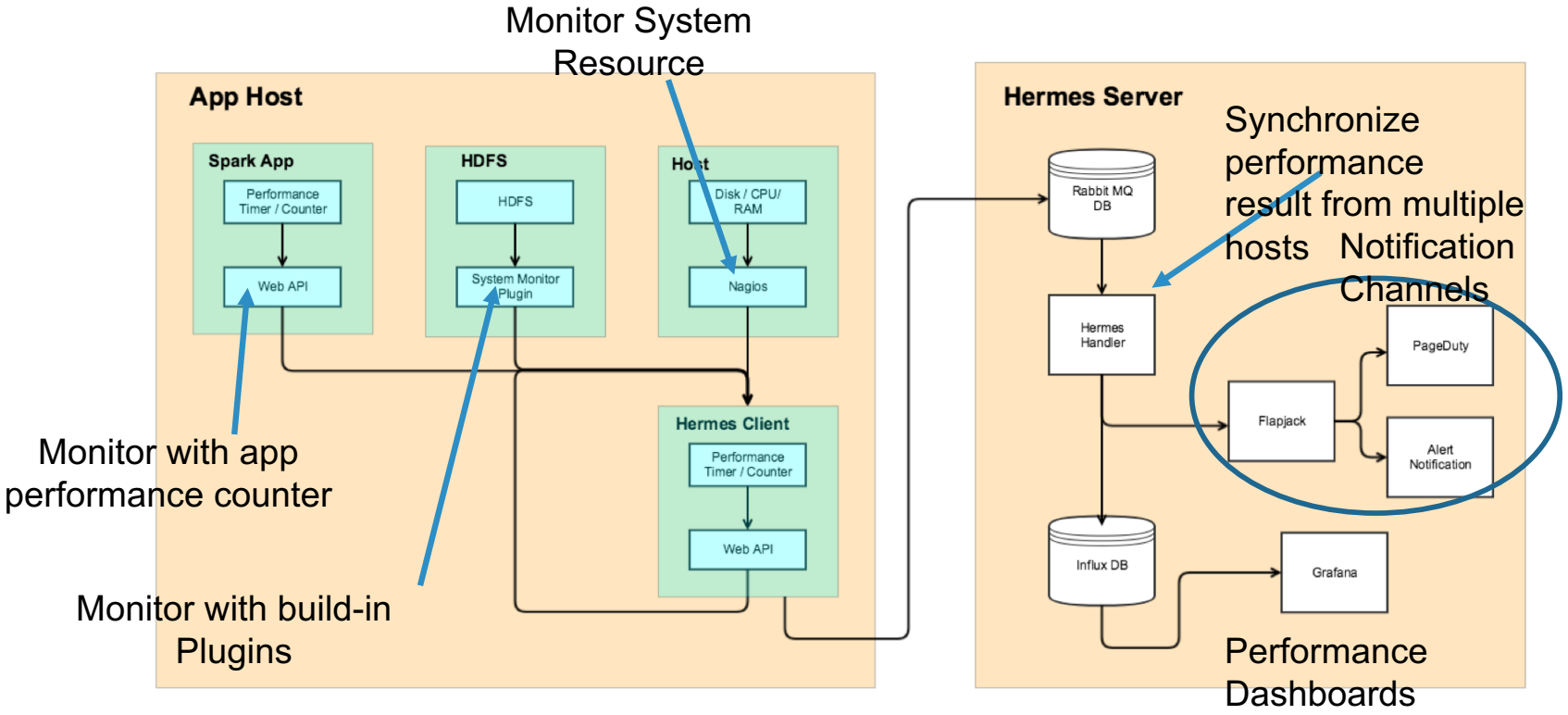
Summary Metrics for 1273 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	46 ms	1 s	1 s	2 s	6 s
Scheduler Delay	0 ms	1 ms	5 ms	0.2 s	0.3 s
Task Deserialization Time	0 ms	1 ms	1 ms	13 ms	59 ms
GC Time	0 ms	0 ms	12 ms	33 ms	0.9 s
Result Serialization Time	0 ms	0 ms	0 ms	0 ms	1 ms
Getting Result Time	0 ms	0 ms	0 ms	0 ms	0 ms
Peak Execution Memory	0.0 B	0.0 B	0.0 B	0.0 B	0.0 B
Input Size / Records	241.0 KB / 125	2.4 MB / 1724	2.5 MB / 1794	2.6 MB / 1864	5.3 MB / 3955

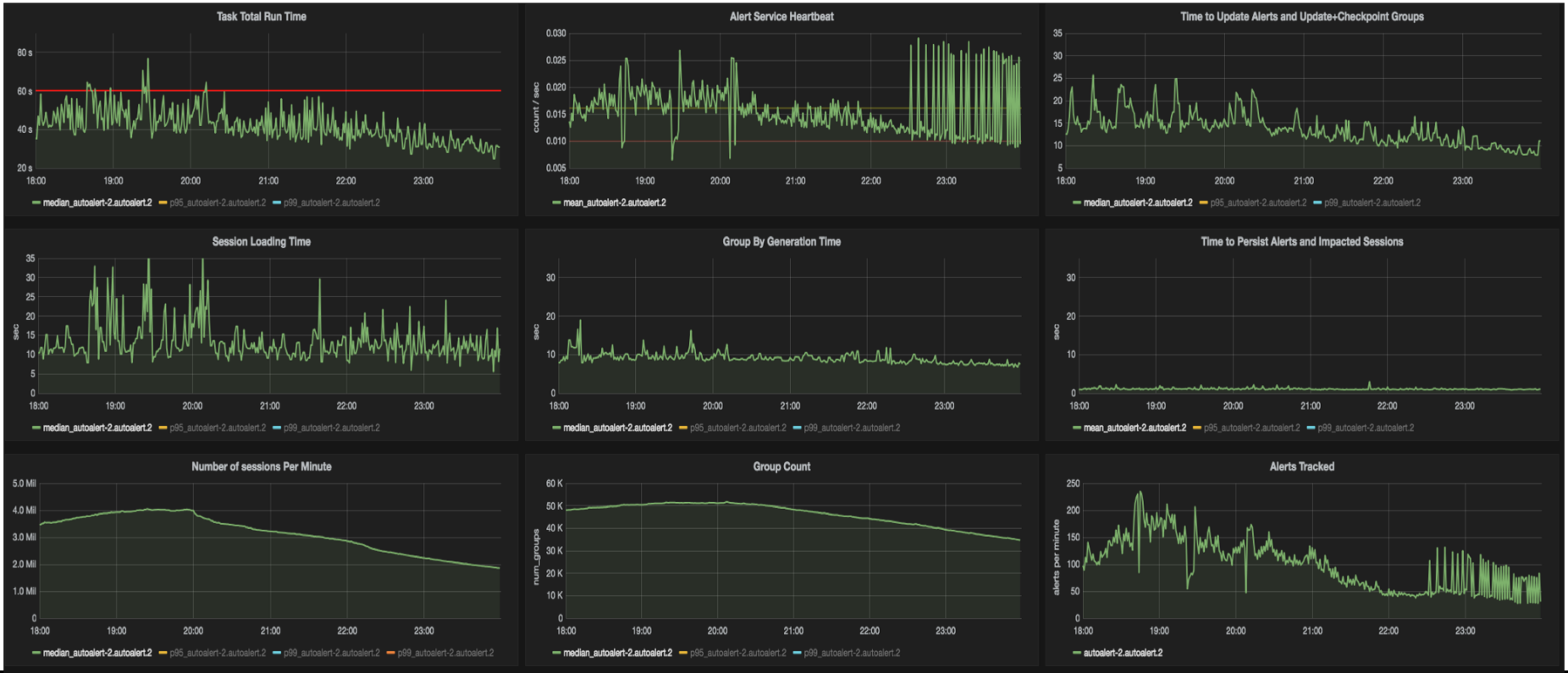
Long Running Spark Application Monitoring

- Why is it important?
  - Spark job performance variance
  - System load bottleneck
- What do we need?
  - Long data retention
  - Correlation with other system KPI, even with business logics
- What are options?
  - Cloudera Manager
  - Application log + Splunk / Elastic Search
  - A light-weighted, independent in-house monitor tool (Hermes)

Monitoring (Conviva Hermes)



Monitor (Conviva Hermes)



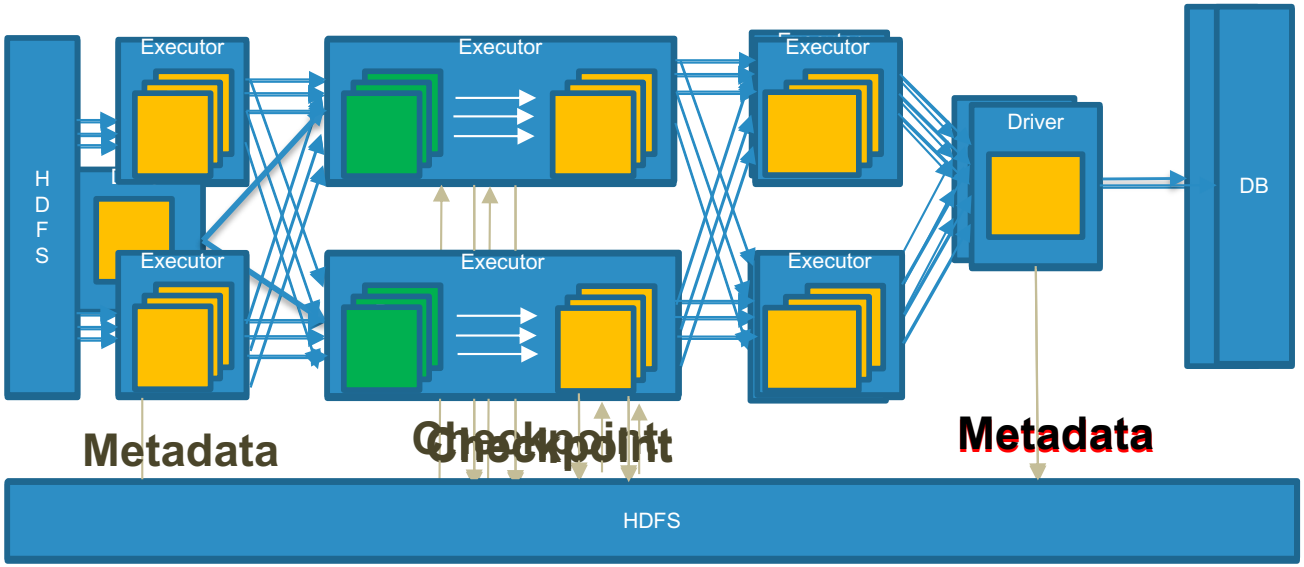
Stability, Scalability, Resilience

Fast Recovery

- Challenges
  - Node Failures: Spark Worker, Master / Driver
  - Warm up after system recovery / restart
  - Scale of our model: 10M parameters
  - Training a anomaly detection model requires a long time window of recent data (~250M raw data points / 2 hours)
- System warm up options
  - Re-compute model using historical raw data
  - Wait till new model trained

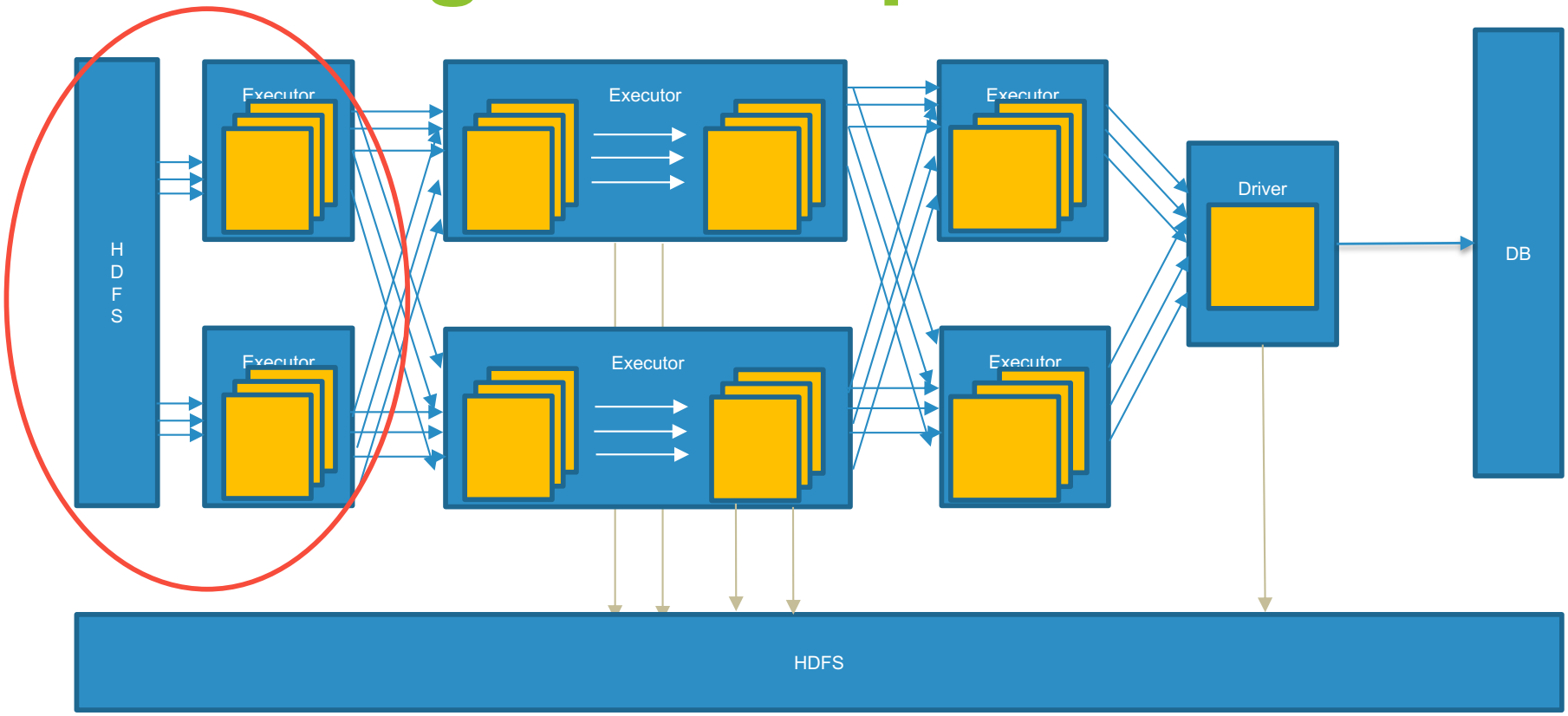
Fast Recovery (Cont.)

- Our solution
  - Recovery through RDD checkpoint



Recovery in < 2 mins

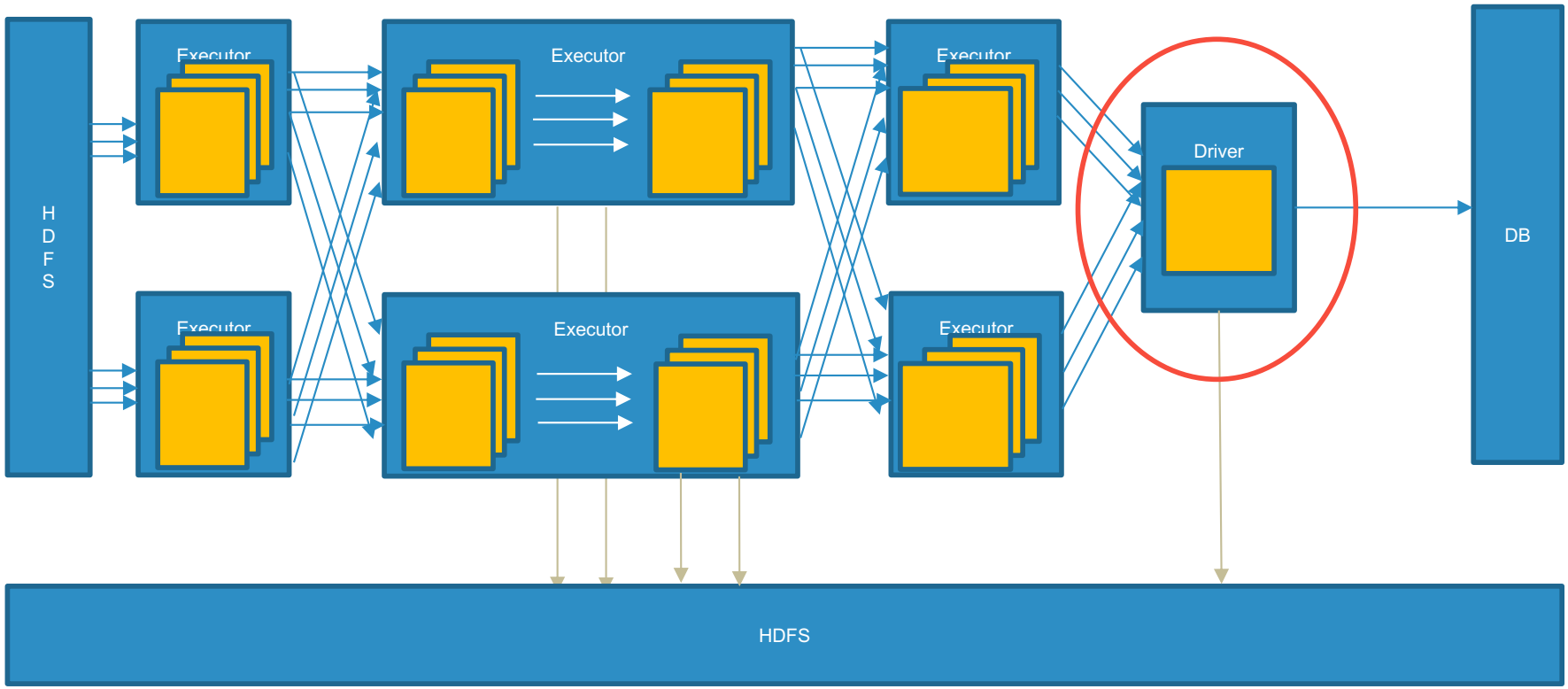
Data Ingestion Optimization



Data Ingestion Optimization

- The upstream data source sometimes serves as a micro-service. As a application, we don't have much control on the data partitions and HDFS performance etc.
  - Optimized partition split
    - Some format of Hadoop files can be split, which is essential to increase parallelism
    - Balanced partition can be achieved by optimized split, which reduces data skewness.
  - Spark Task Speculation
    - The upstream HDFS data node performance varies
    - The resource competition between varies applications

High Availability



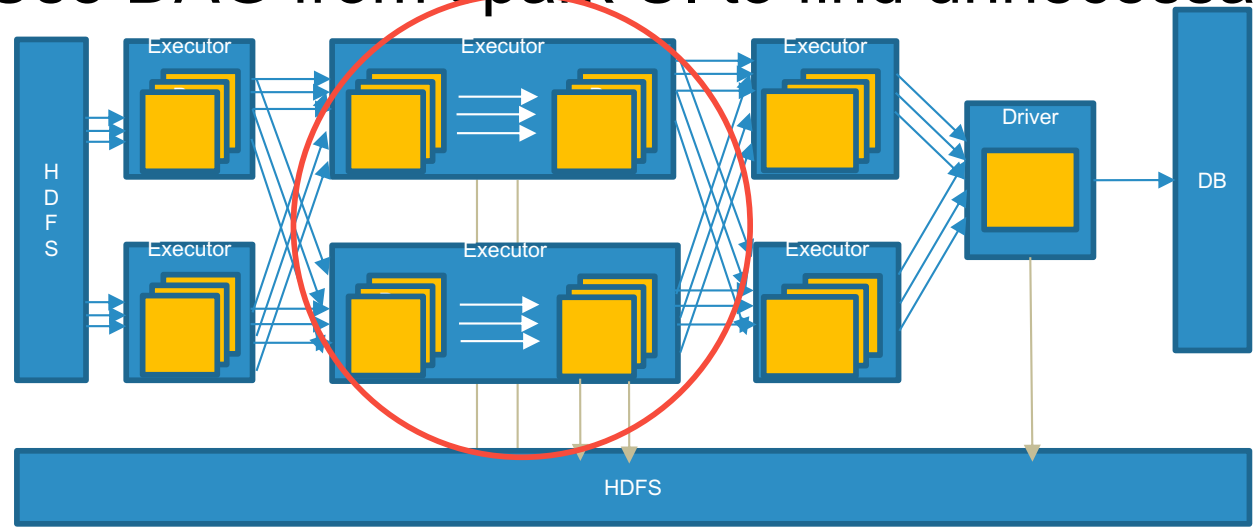
High Availability

- Constraint of YARN
  - Each worker can be selected to be driver
  - Driver preparation may require service not manageable by YARN
- Standalone + ZooKeeper
  - Simple and straightforward resource management
  - Support driver / master failover

Software Level Optimization

Spark Job Optimization

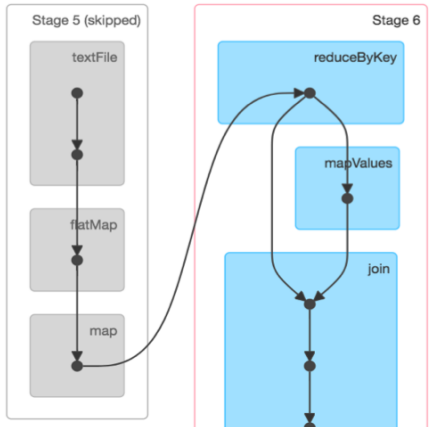
- Reduce intra-job data shuffling
  - Local reduce before shuffling
  - Use DAG from spark UI to find unnecessary shuffling

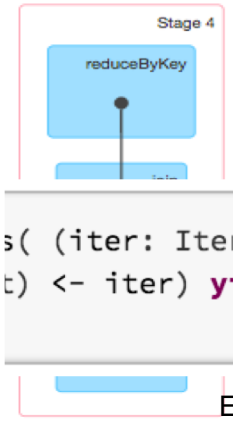


Spark Optimization Example

```
val file = sc.textFile("/mnt/convivaoregon/shubo/Conviva.txt")
val wc = file.flatMap(line => line.toLowerCase.split(" "))
              .map(word => (word, 1))
              .reduceByKey(_ + _)
```

```
val mapRDD = wc.map{case (key, value) => (key, value * 2)}
wc.join(mapRDD).collect
```






```
s( (iter: Iterator[(String, Int)]) =>
t) <- iter) yield (key, value * 2) }, preservesPartitioning = true)
```

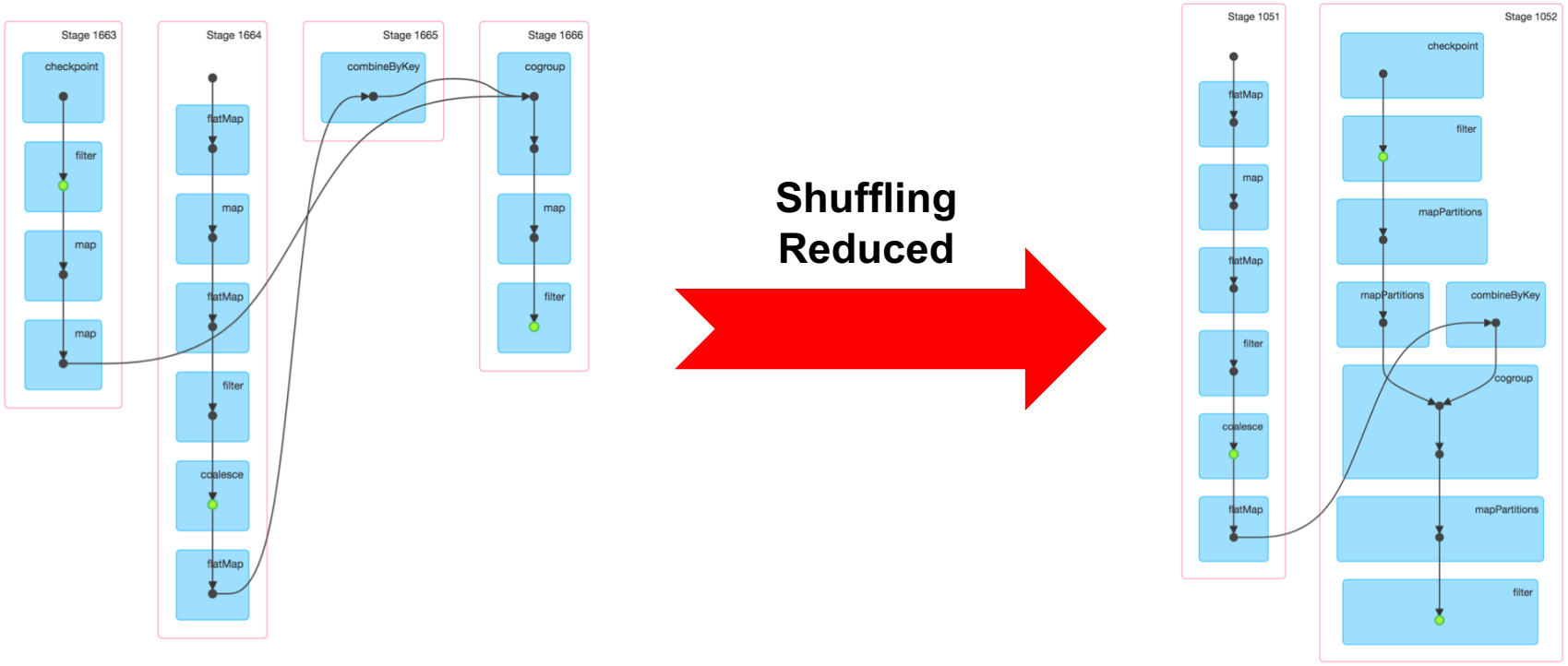
5.1Kb Data Shuffling / 77 ms

2.9Kb Data Shuffling / 33 ms

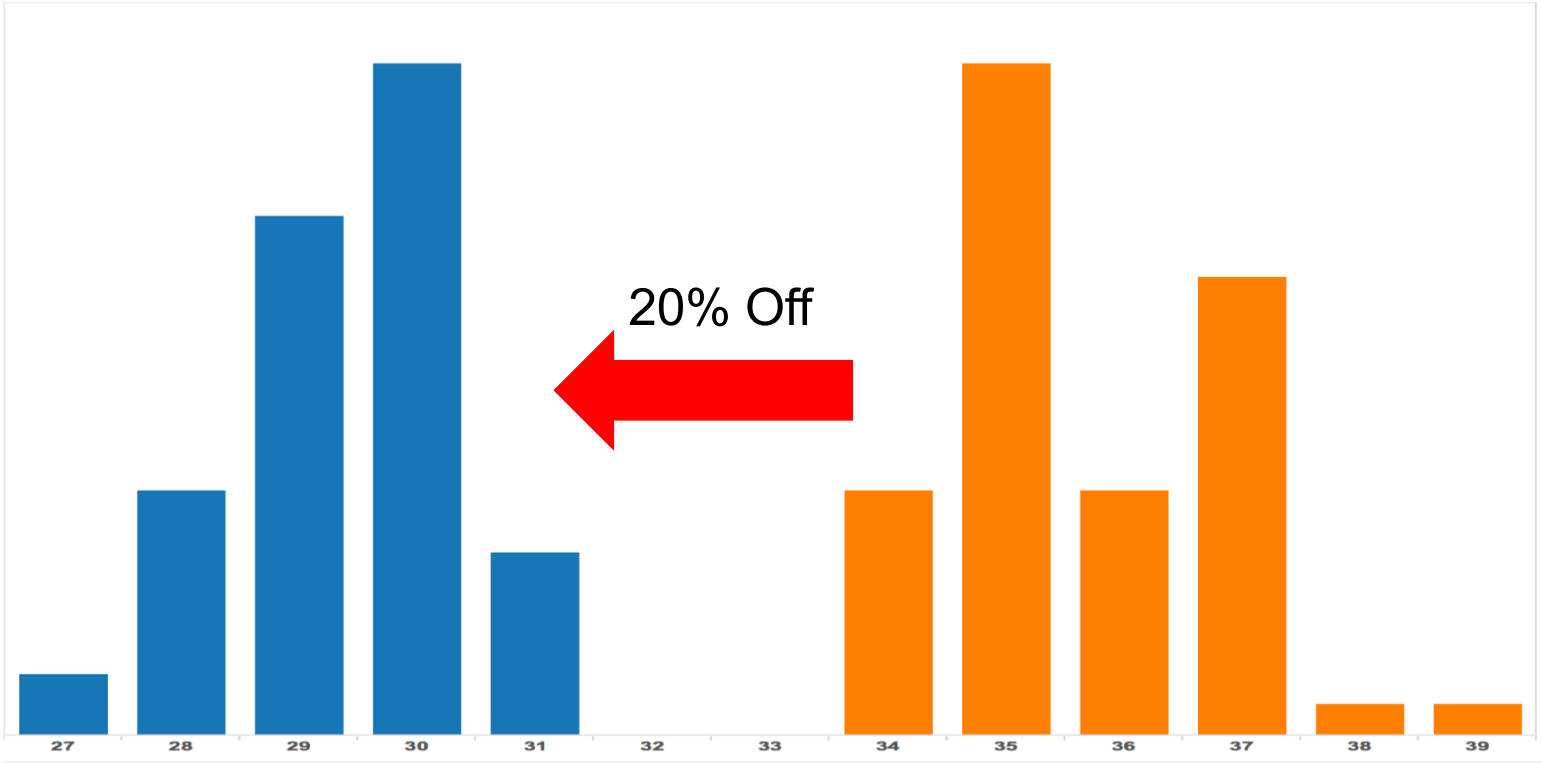
Experiment ran on Spark 2.0 with 1 worker (c4.2x AWS node)

https://conviva.cloud.databricks.com/#notebook/4122547/command/4180381

Optimization Result (DAG)



Optimization Result (Time)



Serialization Optimization

- Optimize Computation or I/O cost?
- Solution
  - Remove derived field and compute it at runtime
  - Customized SerDe
  - Kryo SerDe
- Result
  - Checkpoint file size 2.2Gb → 1.5Gb
  - Data shuffling reduction ~35%

SerDe Optimization Example

```
case class Log1(
  val startDate: Int,
  val endDate: Int,
  val size: Int,
  val inventory: Seq[Int]
)
```

9Mb / 100K Objects

Remove derived field

```
case class Log2(
  startDate: Int,
  endDate: Int,
  inventory: Seq[Int]
) {
  def size = endDate - startDate
}
```

8.54Mb/ 100K Objects

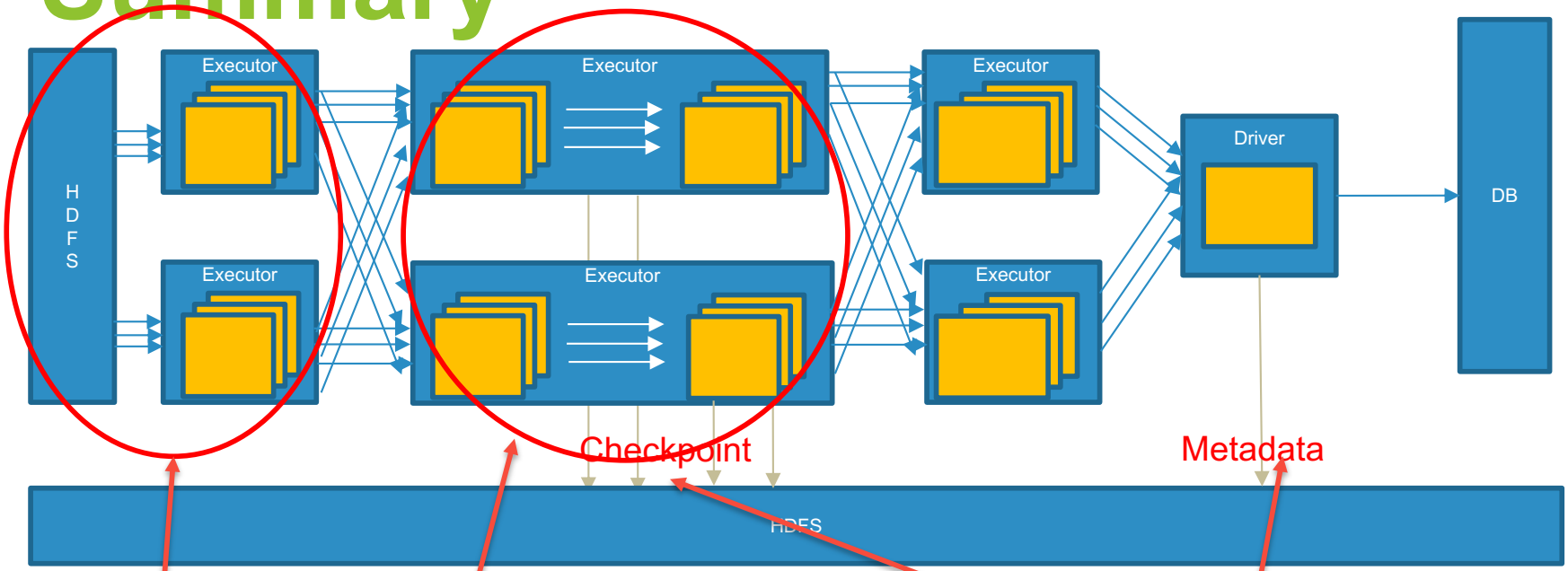
Custom SerDe

```
import java.io.{ObjectInputStream, ObjectOutputStream}
case class Log3(
  var startDate: Int,
  var endDate: Int,
  var inventory: Seq[Int]
) {
  def size = startDate - endDate

  private def writeObject(out: ObjectOutputStream): Unit = {
    out.writeInt(startDate)
    out.writeInt(endDate)
    // Use while loop for better performance
    var i = 0
    while (i < size) { out.writeInt(inventory(i)); i += 1; }
  }

  private def readObject(in: ObjectInputStream): Unit = {
    startDate = in.readInt
    endDate = in.readInt
    inventory = {0 until size}.map(_ => in.readInt)
  }
}
```

1.93Mb/ 100K Objects

https://conviva.cloud.databricks.com/#notebook/4223474/command/4223490

Summary



Data Ingestion Optimization →Higher parallelism, lower variance

Reduce data shuffling →higher performance

Fast Recovery

SerDe → Less I/O

Summary

- Conviva Video AI Alerts is scaled up
  - 7X load
  - 2.5X Computation
  - 99.99% Uptime


conviva
conviva.com/CAREERS
COME WORK WITH US AT CONVIVA!
JOIN US!