



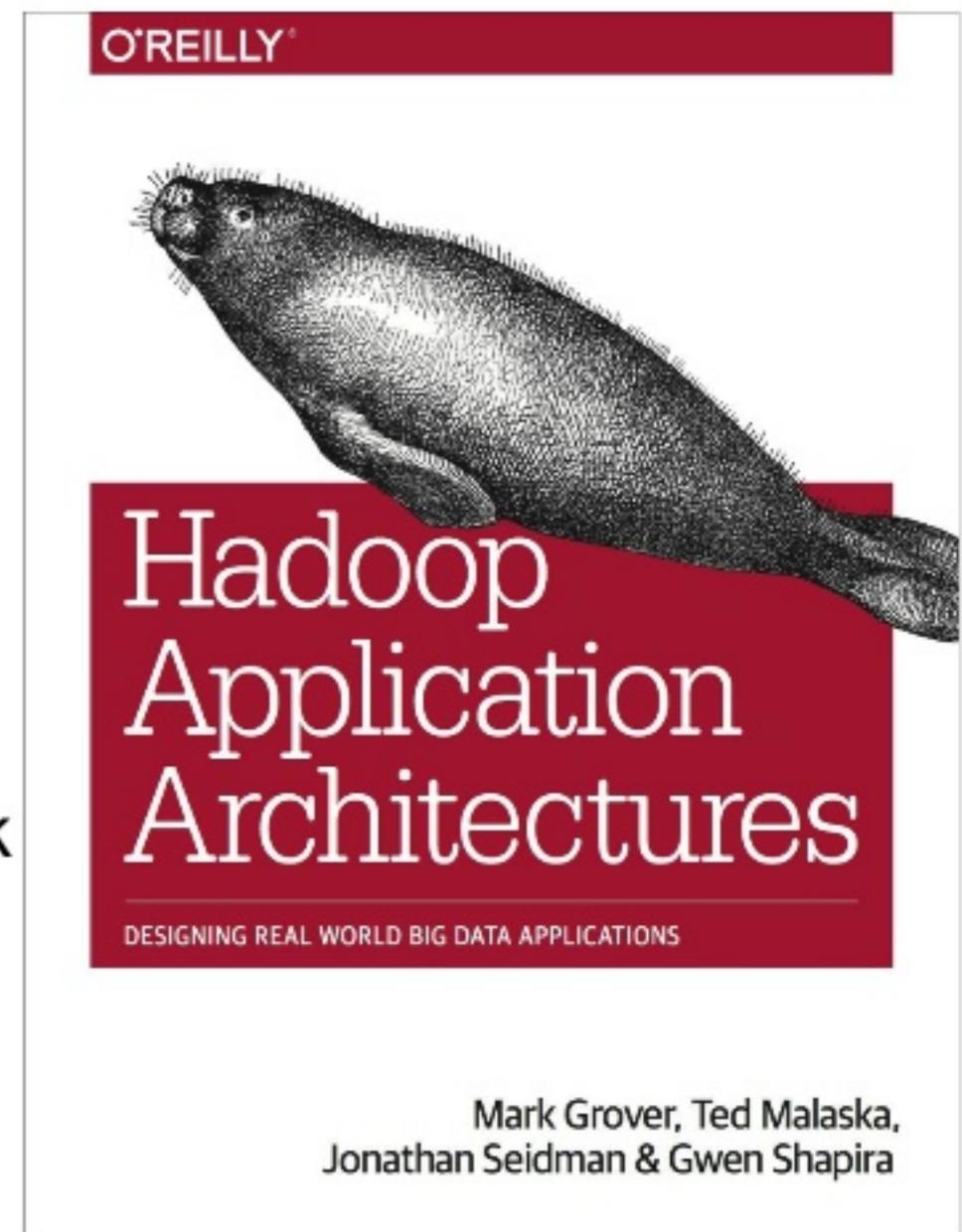
Top 5 mistakes when writing a streaming app

Mark Grover (@mark_grover) - Software Engineer, Cloudera
Ted Malaska (@TedMalaska) - Group Technical Architect, Blizzard

tiny.cloudera.com/streaming-app

Book

- Book signing, at 1pm today
 - Cloudera booth
- hadooparchitecturebook.com
- [@hadooparchbook](https://twitter.com/hadooparchbook)
- github.com/hadooparchitecturebook
- [slideshare.com/hadooparchbook](https://www.slideshare.com/hadooparchbook)



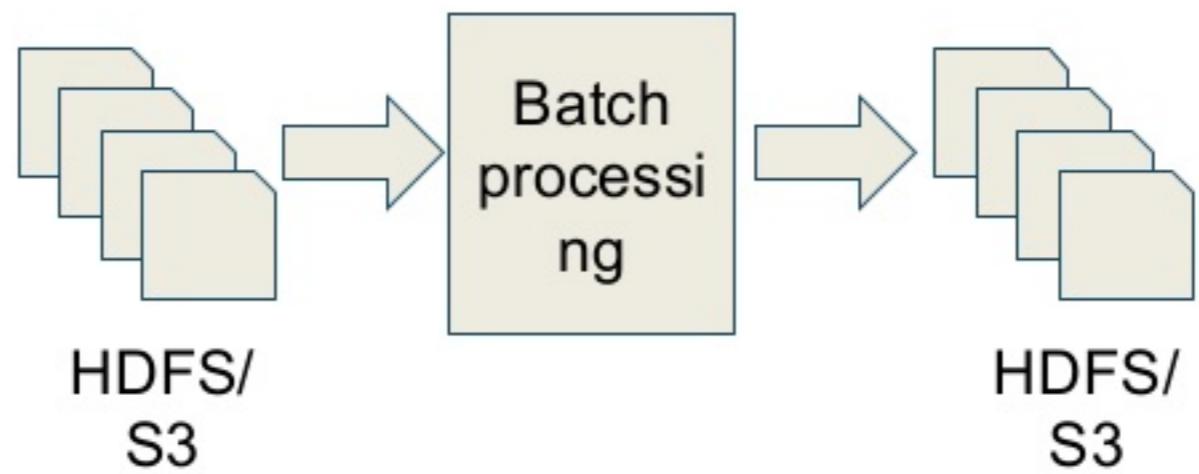
Agenda

- 5 mistakes when writing streaming applications

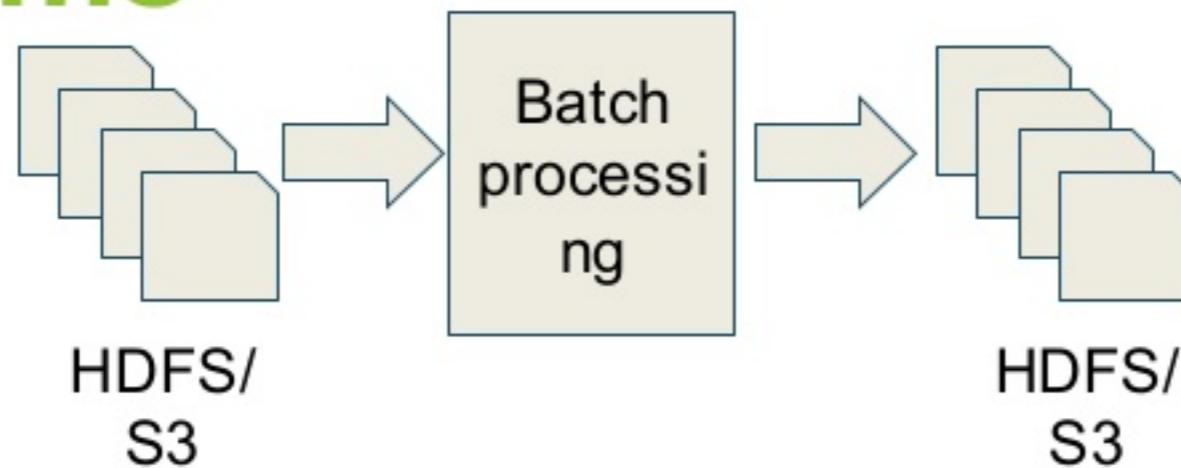


#5 – Not monitoring and managing jobs

Batch systems

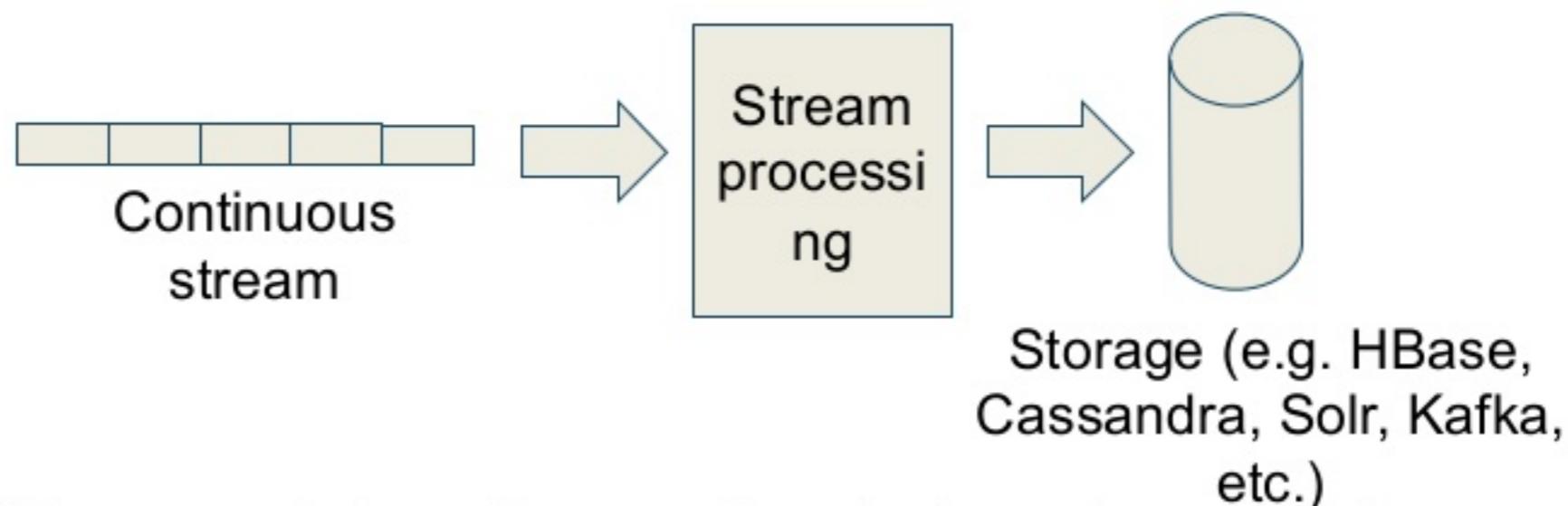


Managing and monitoring batch systems



- Use Cron/Oozie/Azkaban/Luigi for orchestration
- Validation logic in the job (e.g. if input dir is empty)
- Logs aggregated at end of the job
 - Track times against previous runs, etc.

Streaming systems



- “Automatic” orchestration (micro-batching)
- Long running driver process

Not originally built for streaming

- YARN
 - Doesn't aggregate logs until job finishes
- Spark
 - Checkpoints can't survive app or Spark upgrades
 - Need to clear checkpoint directory during upgrade

Big questions remain unanswered

1. Management

- a. Where to run the driver?
- b. How to restart the driver automatically if it fails?
- c. How to pause the job?

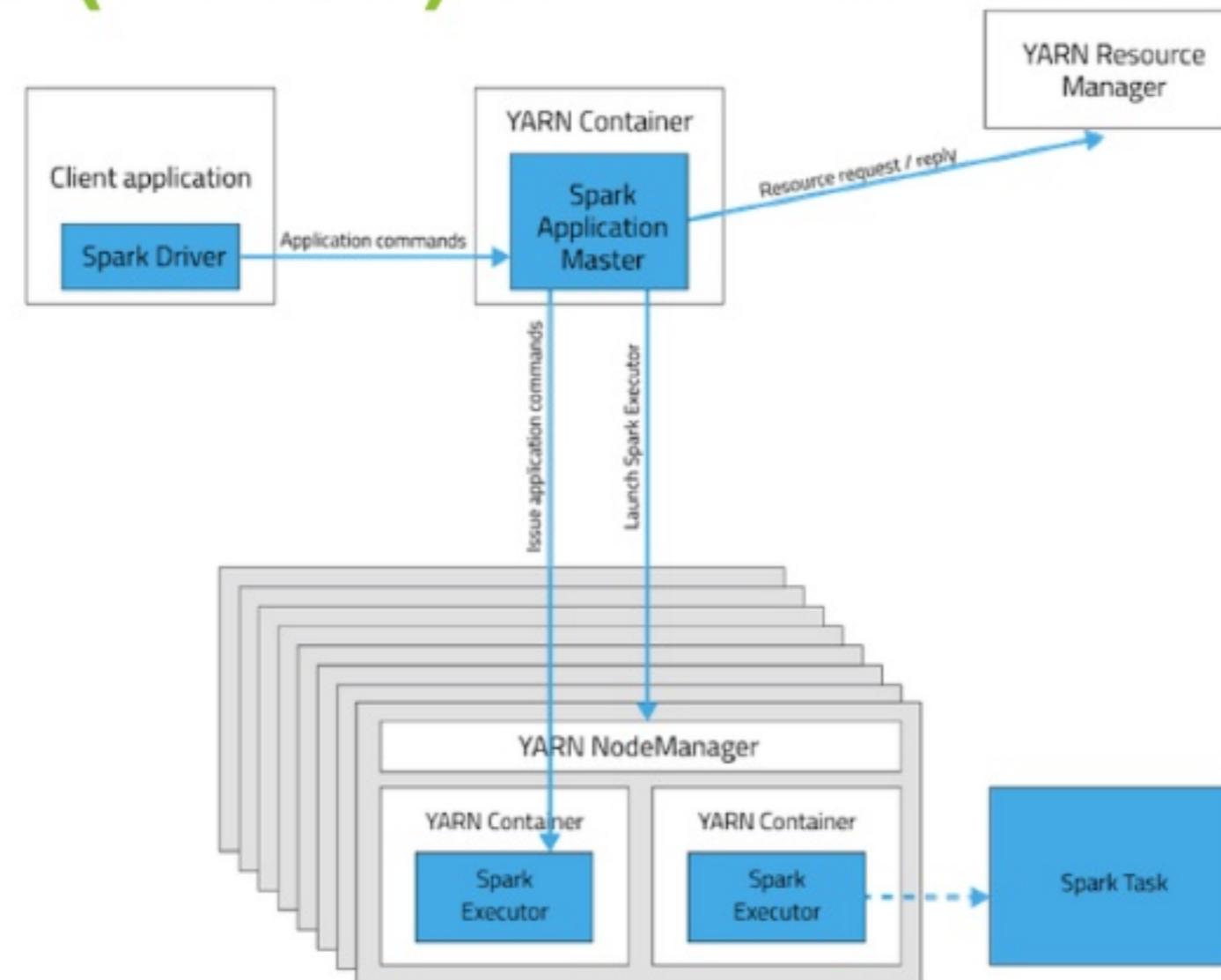
2. Monitoring

- a. How to prevent backlog i.e. make sure processing time < batch interval?
- b. How to keep tabs on health of the long running driver process, etc.?

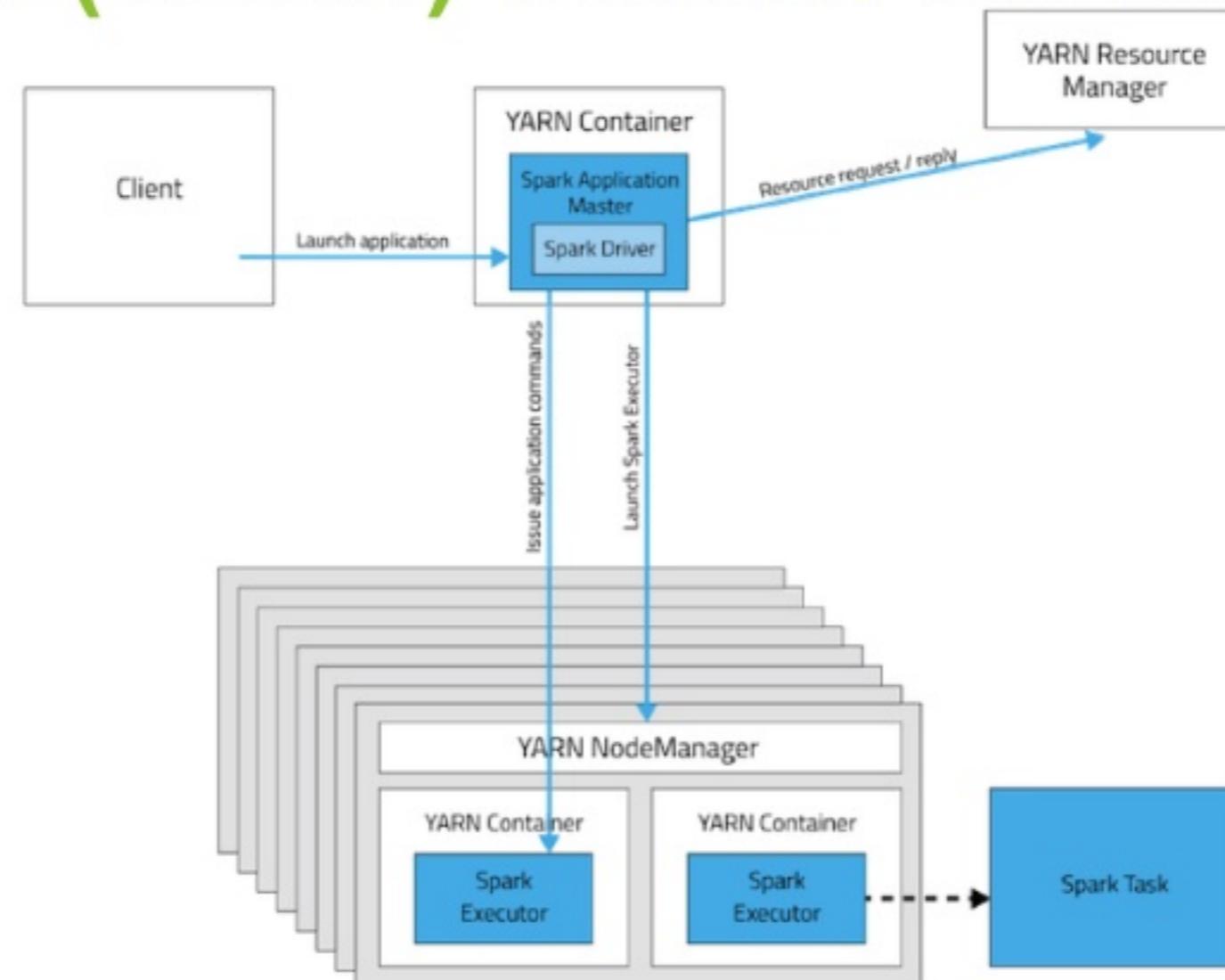
Disclaimer

- Most discussion that follows corresponds to YARN but can be applied to other cluster managers as well.

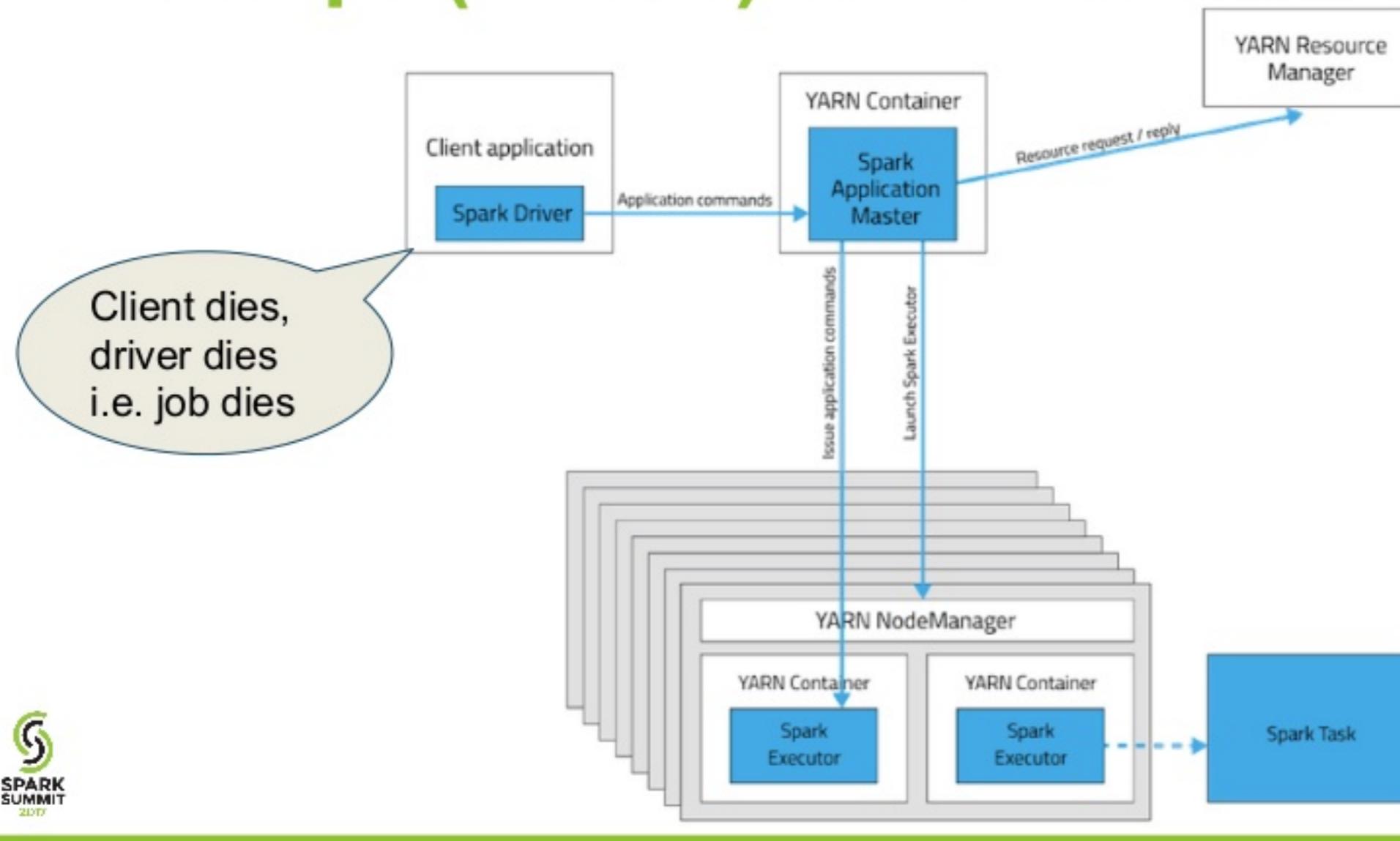
Recap: (YARN) Client mode



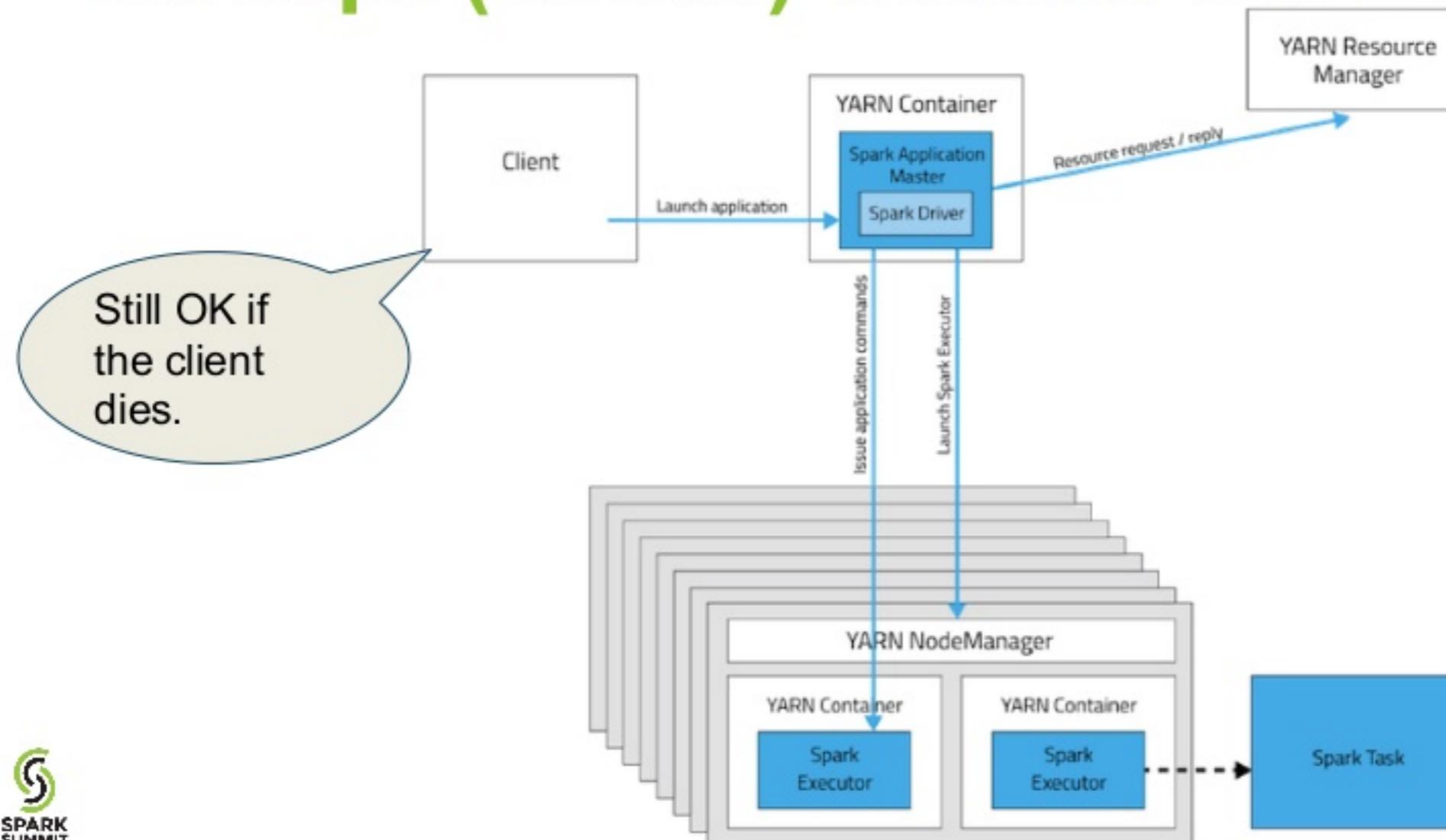
Recap: (YARN) Cluster mode



Recap: (YARN) Client mode



Recap: (YARN) Cluster mode



1a. Where to run the driver?

- Run on YARN Cluster mode
 - Driver will continue running when the client machine goes down

1b. How to restart driver?

- Set up automatic restart

spark.yarn.maxAppAttempts	yarn.resourcemanager.am.max-attempts in YARN	The maximum number of attempts that will be made to submit the application. It should be no larger than the global number of max attempts in the YARN configuration.
spark.yarn.am.attemptFailuresValidityInterval	(none)	Defines the validity interval for AM failure tracking. If the AM has been running for at least the defined interval, the AM failure count will be reset. This feature is not enabled if not configured, and only supported in Hadoop 2.6+.

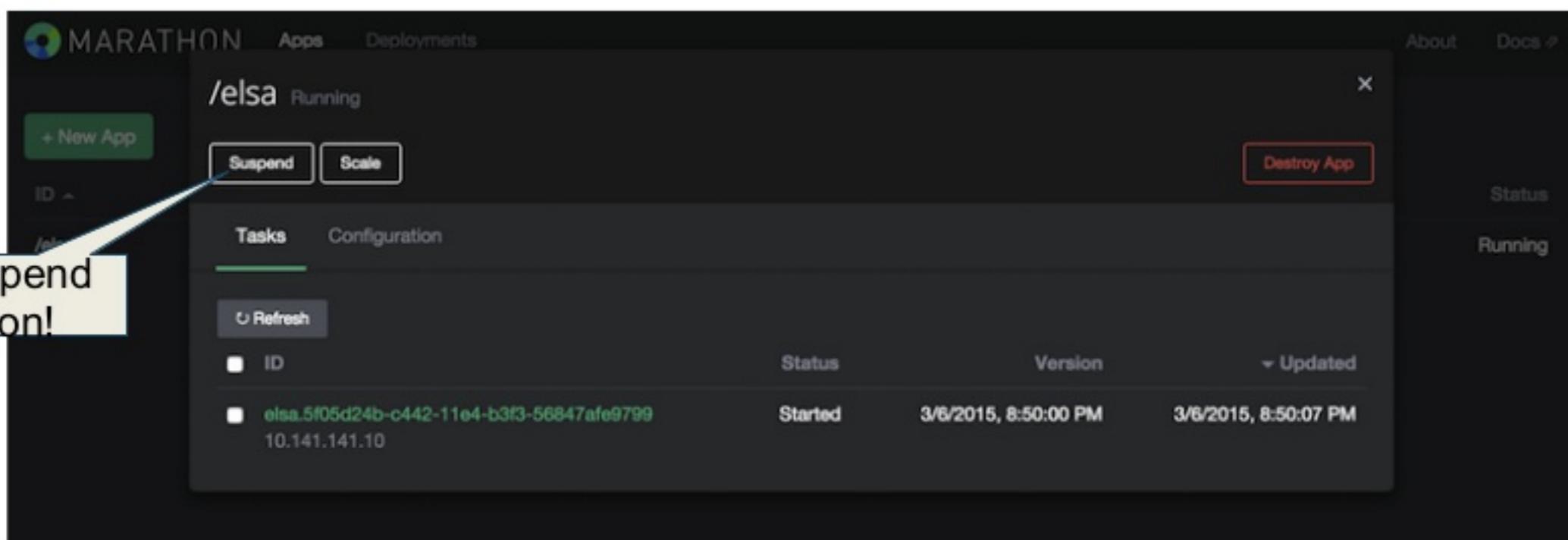
In spark configuration (e.g. `spark-defaults.conf`):

`spark.yarn.maxAppAttempts=2`

`spark.yarn.am.attemptFailuresValidityInterval=1h`

1c. How to pause a job?

- If running on Mesos, use Marathon:
- See “Graceful Shutdown” later for YARN, etc.



The screenshot shows the Marathon web interface. At the top, there are navigation links for 'MARATHON', 'Apps', and 'Deployments'. Below this, a card displays the application '/elsa' in a 'Running' state. On the left of the card, there are buttons for '+ New App', 'ID', and a dropdown menu. In the center, there are 'Suspend' and 'Scale' buttons, with the 'Suspend' button highlighted by a white callout box containing the text 'Suspend button!'. To the right of the card, there is a 'Destroy App' button. Below the card, the 'Tasks' tab is selected, showing a table with one task entry. The table columns are 'ID', 'Status', 'Version', and 'Updated'. The task listed is 'elsa.5f05d24b-c442-11e4-b3f3-56847afe9799' with status 'Started', version '3/6/2015, 8:50:00 PM', and updated at '3/6/2015, 8:50:07 PM'. The IP address '10.141.141.10' is also listed under the ID. On the far right, there is a 'Status' column showing 'Running'. At the bottom left, there is a logo for 'SPARK SUMMIT 2017'.

Suspend button!

ID	Status	Version	Updated
elsa.5f05d24b-c442-11e4-b3f3-56847afe9799 10.141.141.10	Started	3/6/2015, 8:50:00 PM	3/6/2015, 8:50:07 PM

Image source: <https://raw.githubusercontent.com/mhausenblas/elsa/master/doc/elsa-marathon-deploy.png>

2. Monitoring - Spark Streaming UI

The screenshot shows the Spark Streaming UI interface. At the top, there's a navigation bar with tabs: Stages, Storage, Environment, Executors, and **Streaming**. To the right of the bar, it says "test application UI".

Streaming

Started at: Wed Apr 09 22:47:31 PDT 2014
Time since start: 50 seconds 97 ms
Network receivers: 5
Batch Interval: 1 second
Processed batches: 50
Waiting batches: 0

Statistics over last 50 processed batches

Network Input Statistics

Records processed	Min rate (records/sec)	25th percentile	50th percentile (Median)	75th percentile	Max rate (records/sec)
10.7 K	1263.0	9.4 K	10.4 K	11.0 K	12.4 K
10.8 K	1177.0	9.3 K	10.3 K	10.9 K	12.4 K
10.7 K	1256.0	9.3 K	10.4 K	10.8 K	12.5 K
10.8 K	1217.0	9.4 K	10.4 K	10.9 K	12.4 K
10.7 K	1245.0	9.3 K	10.4 K	11.0 K	12.5 K

Batch Processing Statistics

Metric	Last batch	Minimum	25th percentile	Median	75th percentile	Maximum
Processing Time	100 ms	98 ms	113 ms	122 ms	146 ms	534 ms
Scheduling Delay	0 ms	0 ms	0 ms	0 ms	0 ms	5 ms
Total Delay	100 ms	98 ms	113 ms	122 ms	146 ms	539 ms

Processing time,
scheduling delay,
total delay, etc.

Base image from: <http://i.imgur.com/1ooDGhm.png>

2. Monitoring - But I want more!

- Spark has a configurable metrics system
 - Based on Dropwizard Metrics Library
- Use Graphite/Grafana to dashboard metrics like:
 - Number of records processed
 - Memory/GC usage on driver and executors
 - Total delay
 - Utilization of the cluster

Summary - Managing and monitoring

- Afterthought
- But it's possible
- In Structured Streaming, use `StreamingQueryListener` (starting Apache Spark 2.1)

References

- Spark Streaming on YARN by Marcin Kuthan
<http://mkuthan.github.io/blog/2016/09/30/spark-streaming-on-yarn/>
 - Marathon
<https://mesosphere.github.io/marathon/>



#4 – Not considering data loss

Prevent data loss

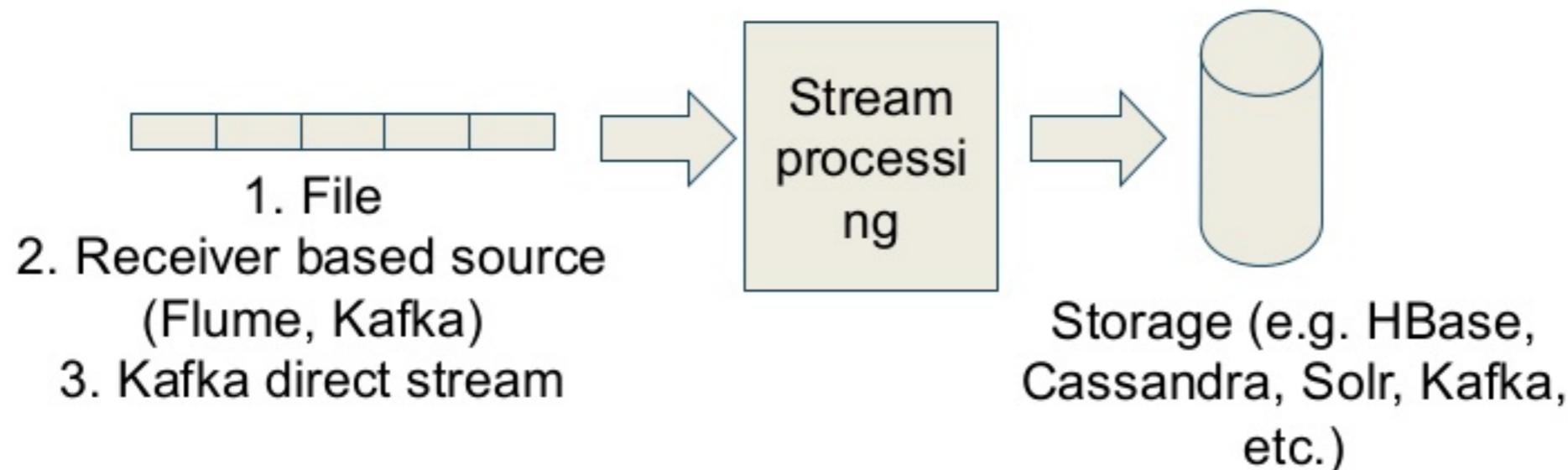
- Ok, my driver is automatically restarting
- Can I lose data in between driver restarts?



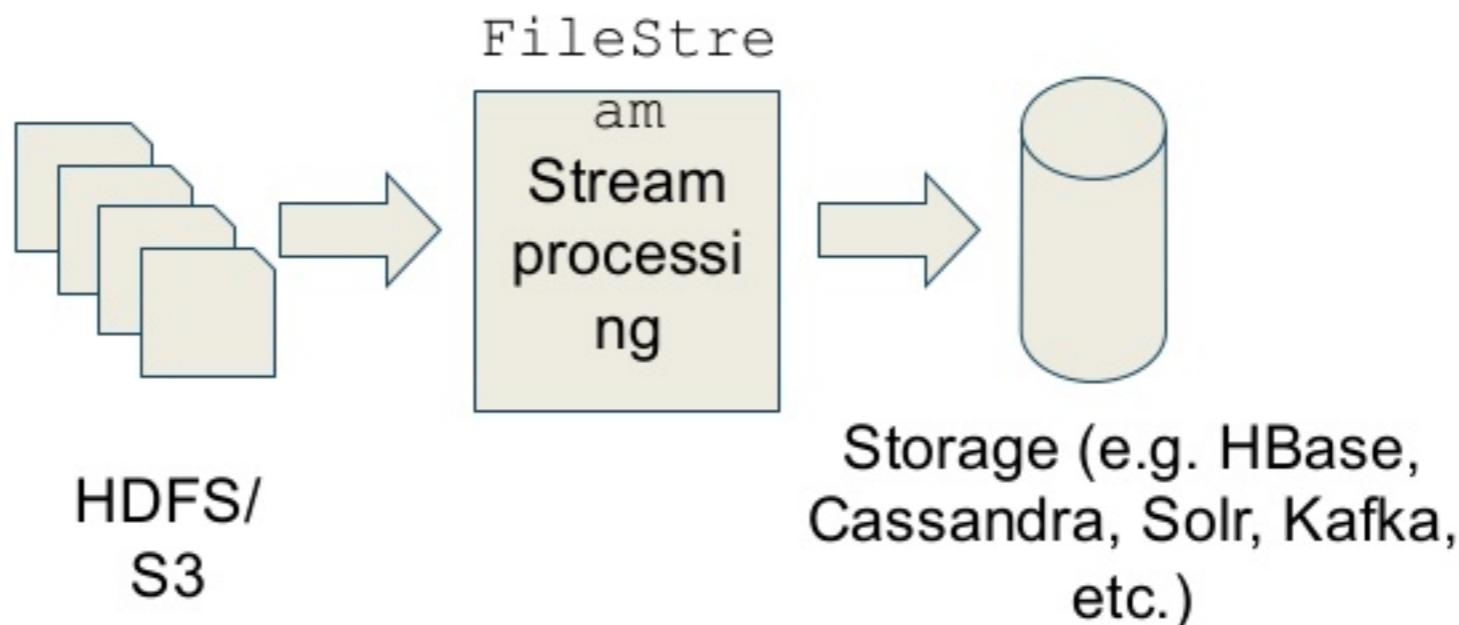
No data loss!

As long as you do things the right way

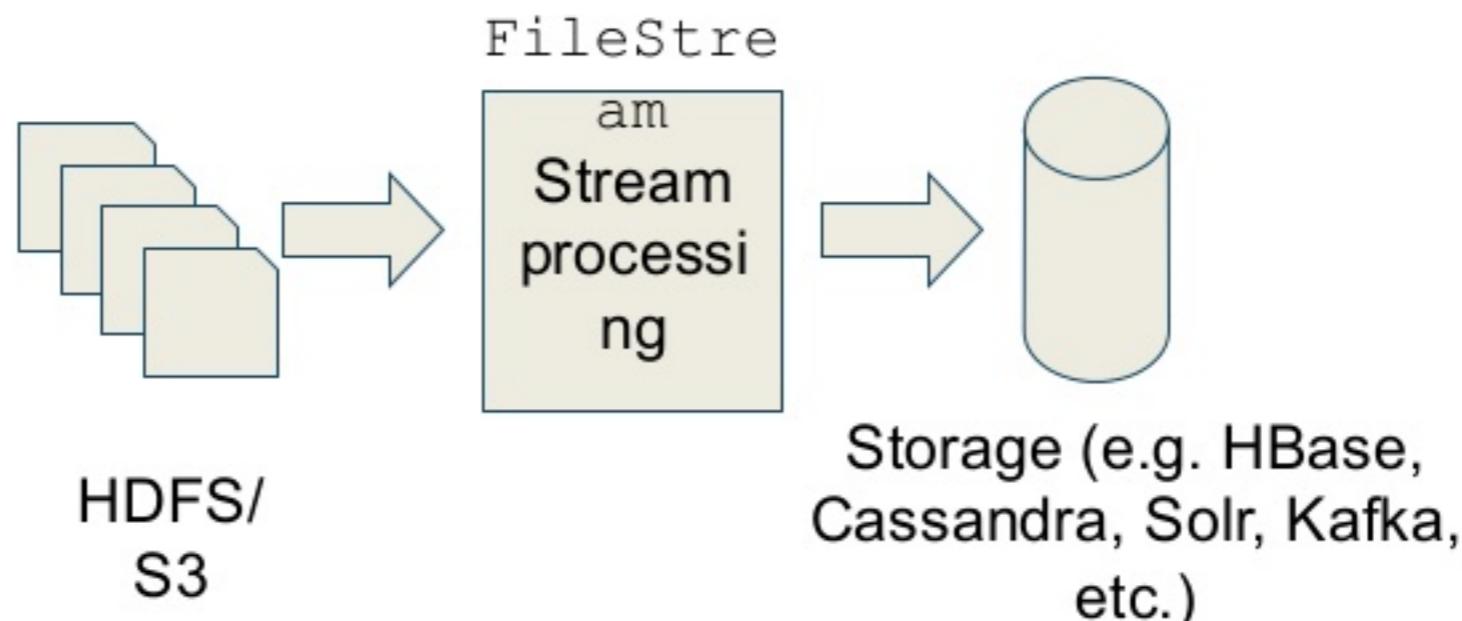
How to do things the right way



1. File sources



1. File sources



- Use checkpointing!

Checkpointing

```
// new context  
val ssc = new StreamingContext(...)  
...  
// set checkpoint directory  
ssc.checkpoint(checkpointDirectory)
```

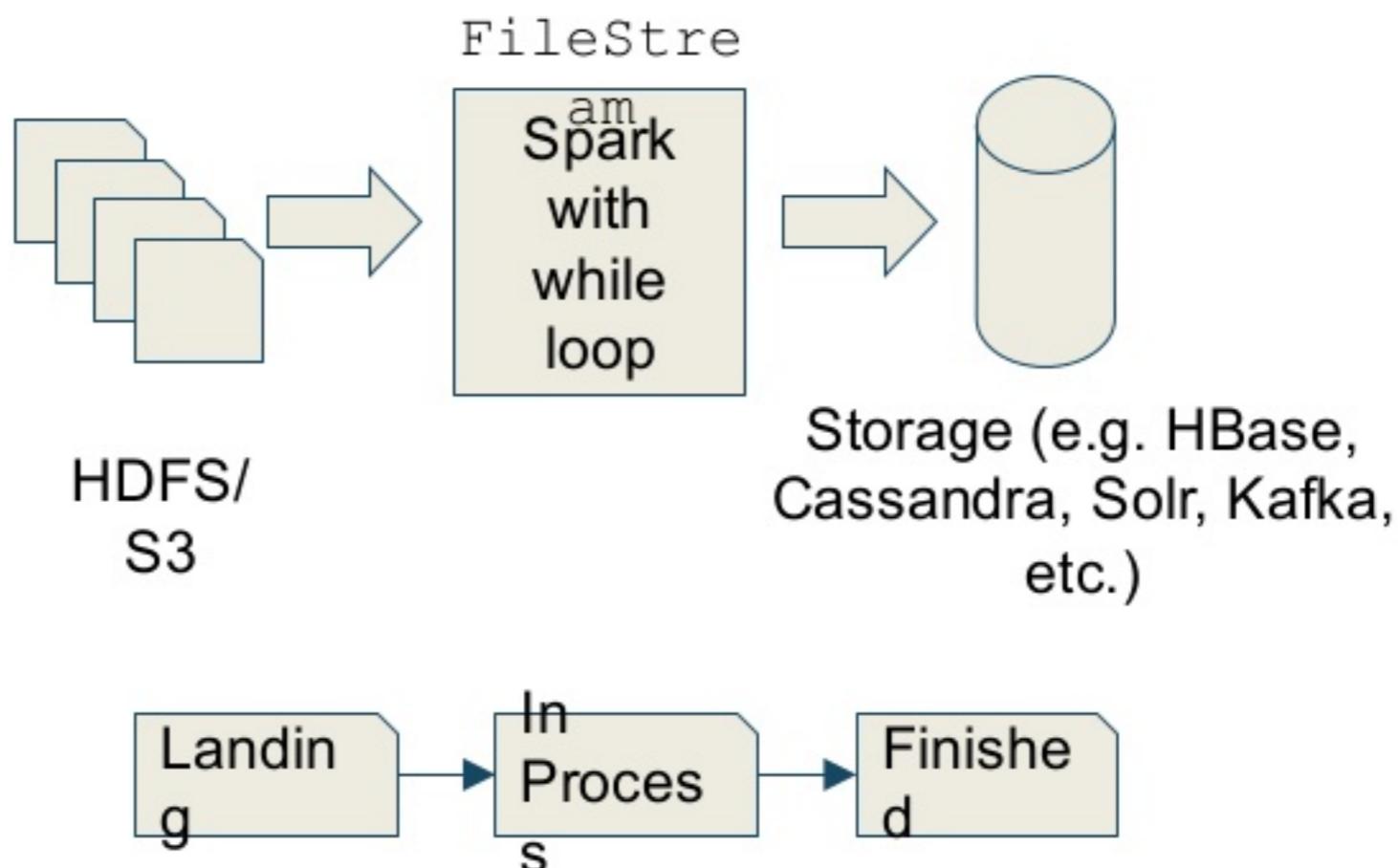
What is checkpointing

- Metadata checkpointing (Configuration, Incomplete batches, etc.)
 - Recover from driver failures
- Data checkpointing
 - Trim lineage when doing stateful transformations (e.g. updateStateByKey)
- Spark streaming checkpoints both data and metadata

Checkpointing gotchas

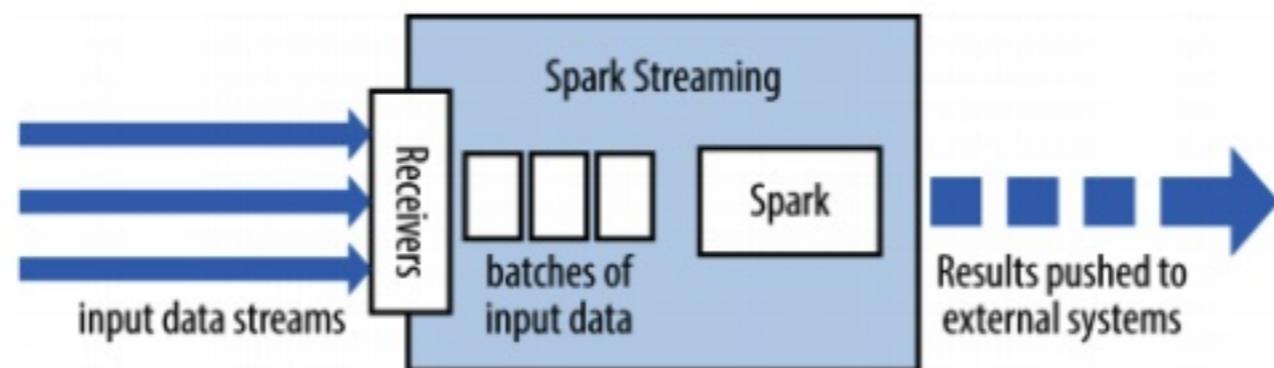
- Checkpoints don't work across app or Spark upgrades
- Clear out (or change) checkpointing directory across upgrades

Ted's Rant



2. Receiver based sources

High-level architecture of Spark Streaming



DStream as a continuous series of RDDs

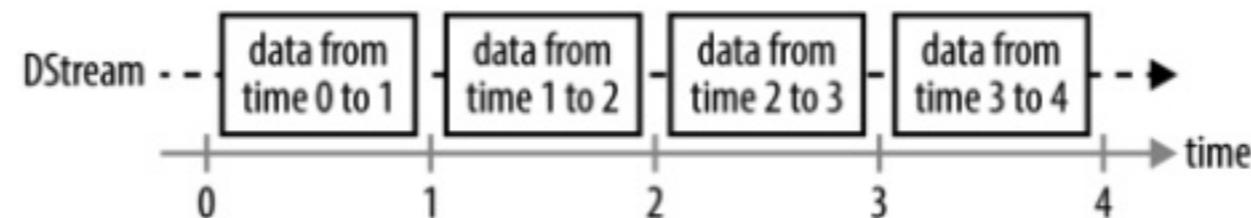


Image source: http://static.oschina.net/uploads/space/2015/0618/110032_9Fvp_1410765.png

Receiver based sources

- Enable checkpointing, AND
- Enable Write Ahead Log (WAL)

– Set the following property to true

```
spark.streaming.receiver.writeAheadLog.enable
```

– Default is false!

Why do we need a WAL?

- Data on Receiver is stored in executor memory
- Without WAL, a failure in the middle of operation can lead to data loss
- With WAL, data is written to durable storage (e.g. HDFS, S3) before being acknowledged back to source.

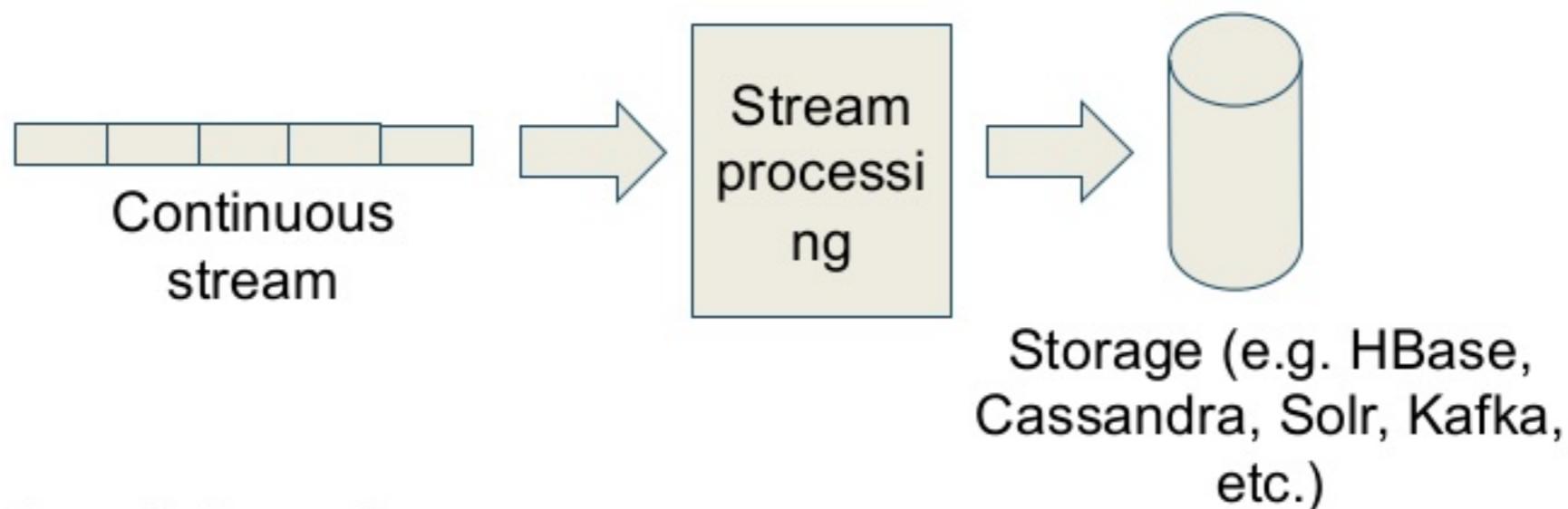
WAL gotchas!

- Makes a copy of all data on disk
- Use `StorageLevel.MEMORY_AND_DISK_SER` storage level for your DStream
 - No need to replicate the data in memory across Spark executors
- For S3
 - `spark.streaming.receiver.writeAheadLog.closeFileAfterWrite` to true (**similar for driver**)

But what about Kafka?

- Kafka already stores a copy of the data
- Why store another copy in WAL?

3. Spark Streaming with Kafka



- Use “direct” connector
- No need for a WAL with the direct connector!

Kafka with WAL

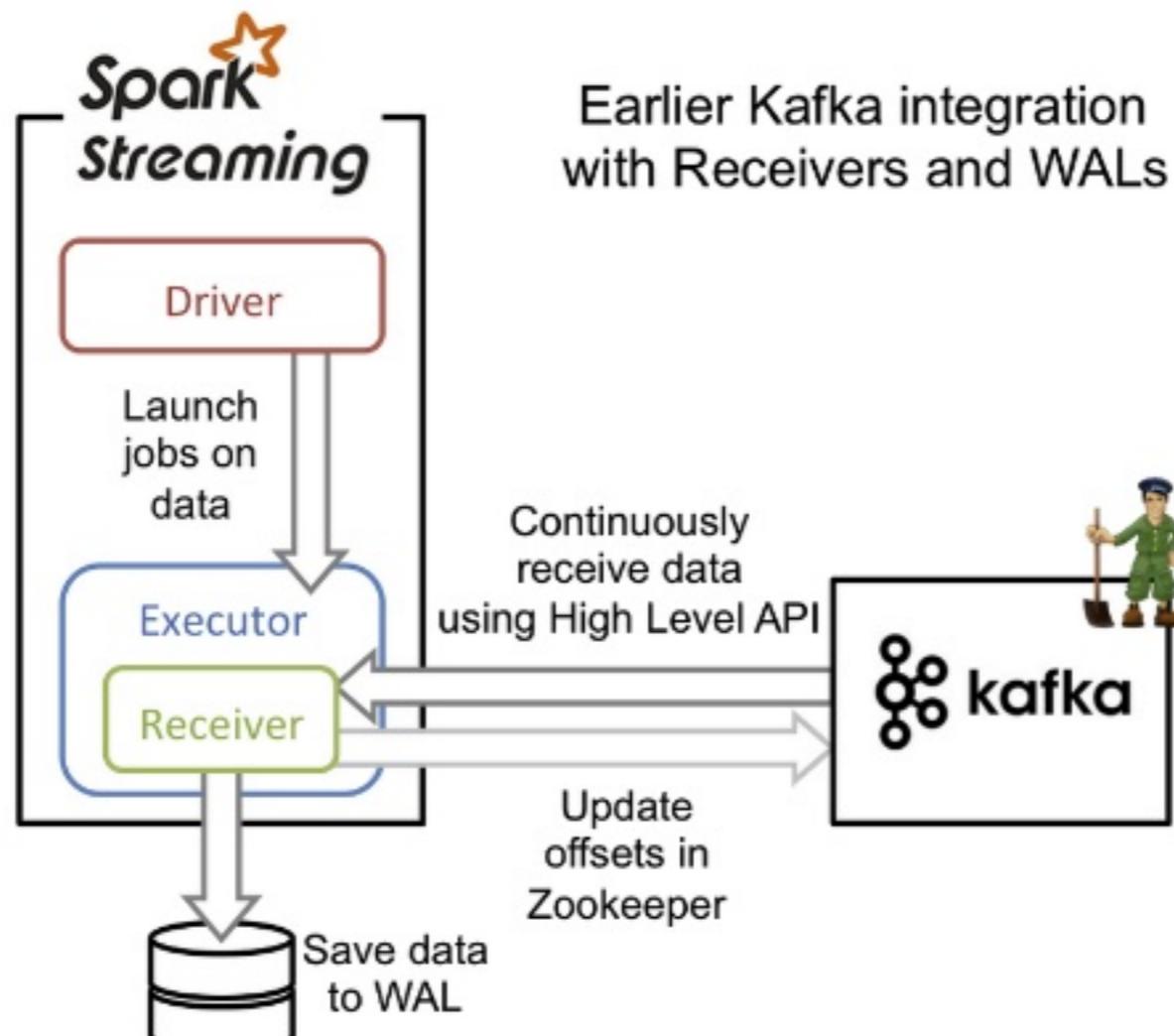


Image source:
<https://databricks.com/wp-content/uploads/2015/03/Screen-Shot-2015-03-29-at-10.11.42-PM.png>

Kafka direct connector (without WAL)

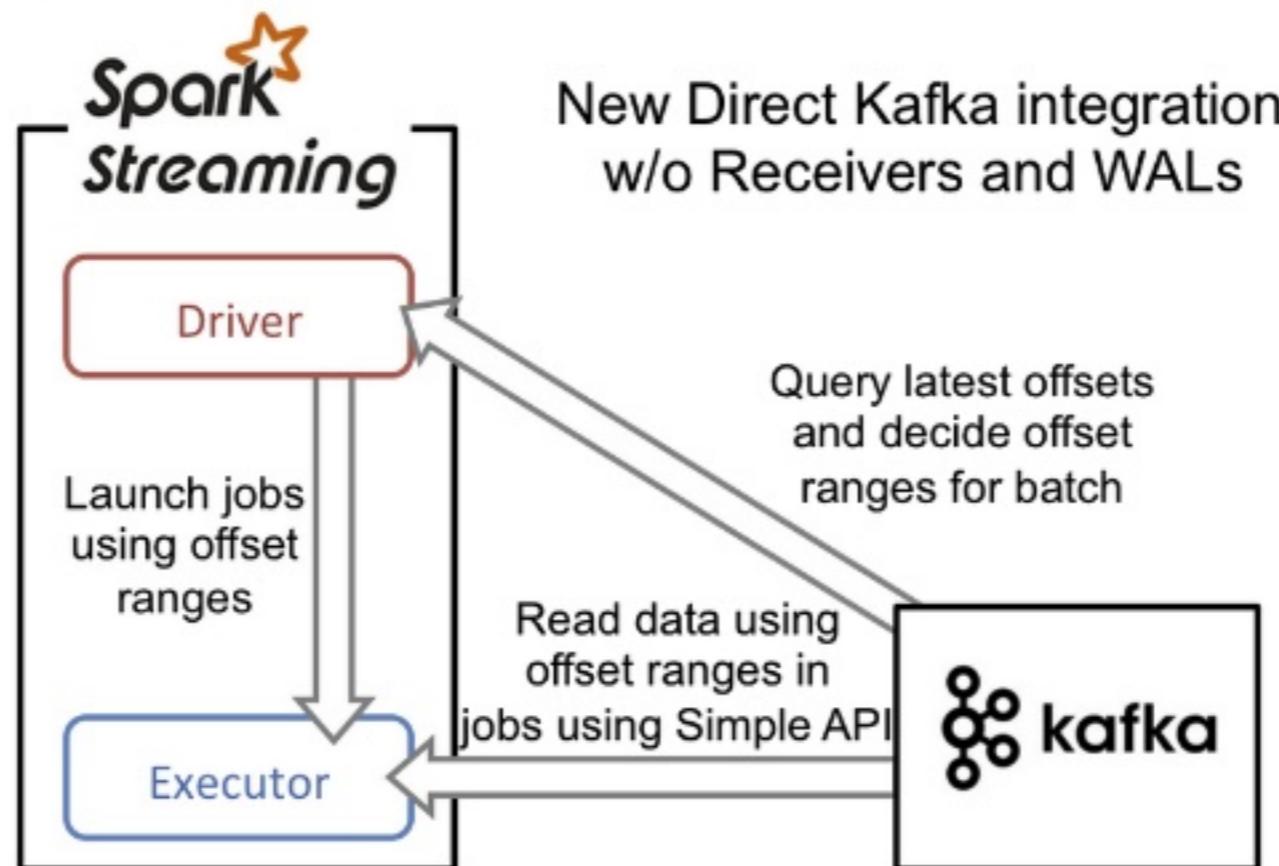


Image source:
<https://databricks.com/wp-content/uploads/2015/03/Screen-Shot-2015-03-29-at-10.14.11-PM.png>

Why no WAL?

- No receiver process creating blocks
- Data is stored in Kafka, can be directly recovered from there

Direct connector gotchas

- Need to track offsets for driver recovery
- Checkpoints?
 - No! Not recoverable across upgrades
- Track them yourself
 - In ZooKeeper, HDFS, or a database
- For accuracy
 - Processing needs to be idempotent, OR
 - Update offsets in the same transaction when updating results

Summary - prevent data loss

- Different for different sources
- Preventable if done right
- In Structured Streaming, state is stored in memory (backed by HDFS/S3 WAL), starting Spark 2.1

References

- Improved fault-tolerance and zero data loss
<https://databricks.com/blog/2015/01/15/improved-driver-fault-tolerance-and-zero-data-loss-in-spark-streaming.html>
 - Tracking offsets when using direct connector
<http://blog.cloudera.com/blog/2015/03/exactly-once-spark-streaming-from-apache-kafka/>



#3 – Use streaming for everything

When do you use streaming?



When do you use streaming?

ALWAYS!

Do I need to use Spark Streaming

- Think about your goals

Types of goals

- Atomic Enrichment
- Notifications
- Joining
- Partitioning
- Ingestion
- Counting
- Incremental machine learning
- Progressive Analysis

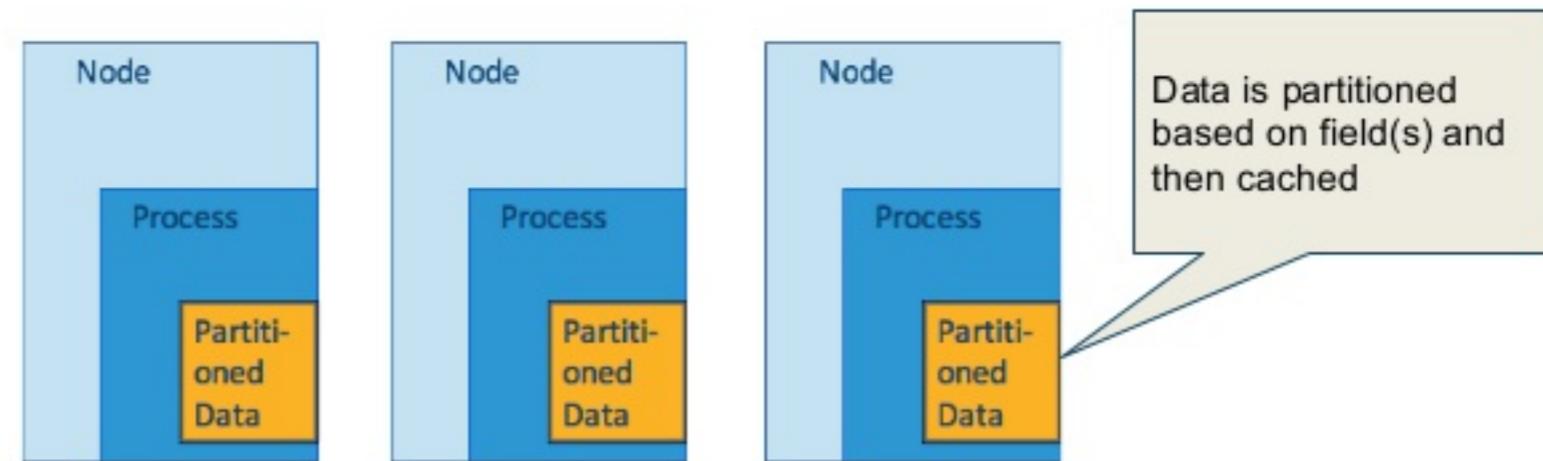
Types of goals

- Atomic enrichment
 - No cache/context needed

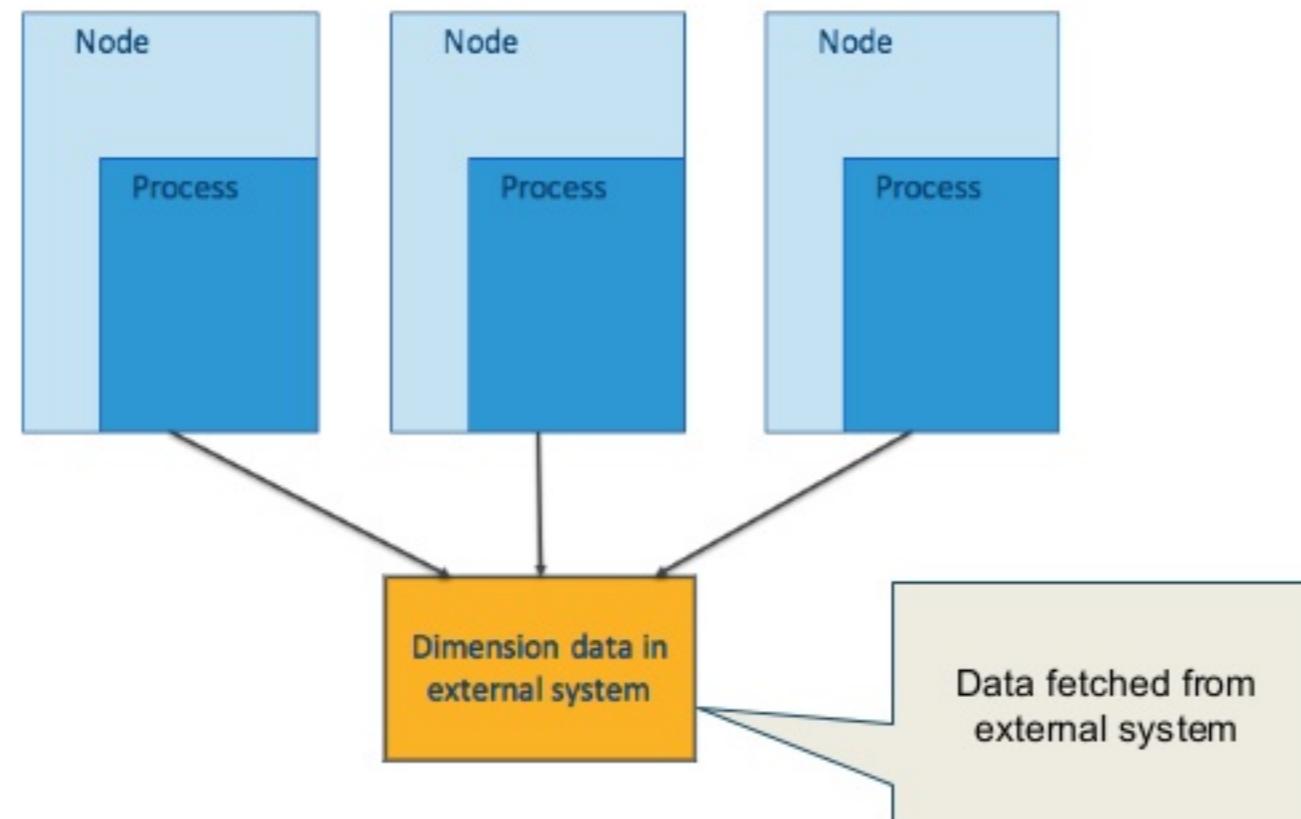
Types of goals

- Notifications
 - NRT
 - Atomic
 - With Context
 - With Global Summary

#2 - Partitioned cache data



#3 - External fetch



Types of goals

- Joining
 - Big Joins
 - Broadcast Joins
 - Streaming Joins

Types of goals

- !!Ted to add a diagram!!
- Partitioning
 - Fan out
 - Fan in
 - Shuffling for key isolation

Types of goals

- Ingestion
 - File Systems or Block Stores
 - No SQLs
 - Lucene
 - Kafka

Types of goals

- Counting
 - Streaming counting
 - Count on query
 - Aggregation on query

Types of goals

- Machine Learning

Types of goals

- Progressive Analysis
 - Reading the stream
 - SQL on the stream

Summary - Do I need to use streaming?

- Spark Streaming is great for
 - Accurate counts
 - Windowing aggregations
 - Progressive Analysis
 - Continuous Machine Learning

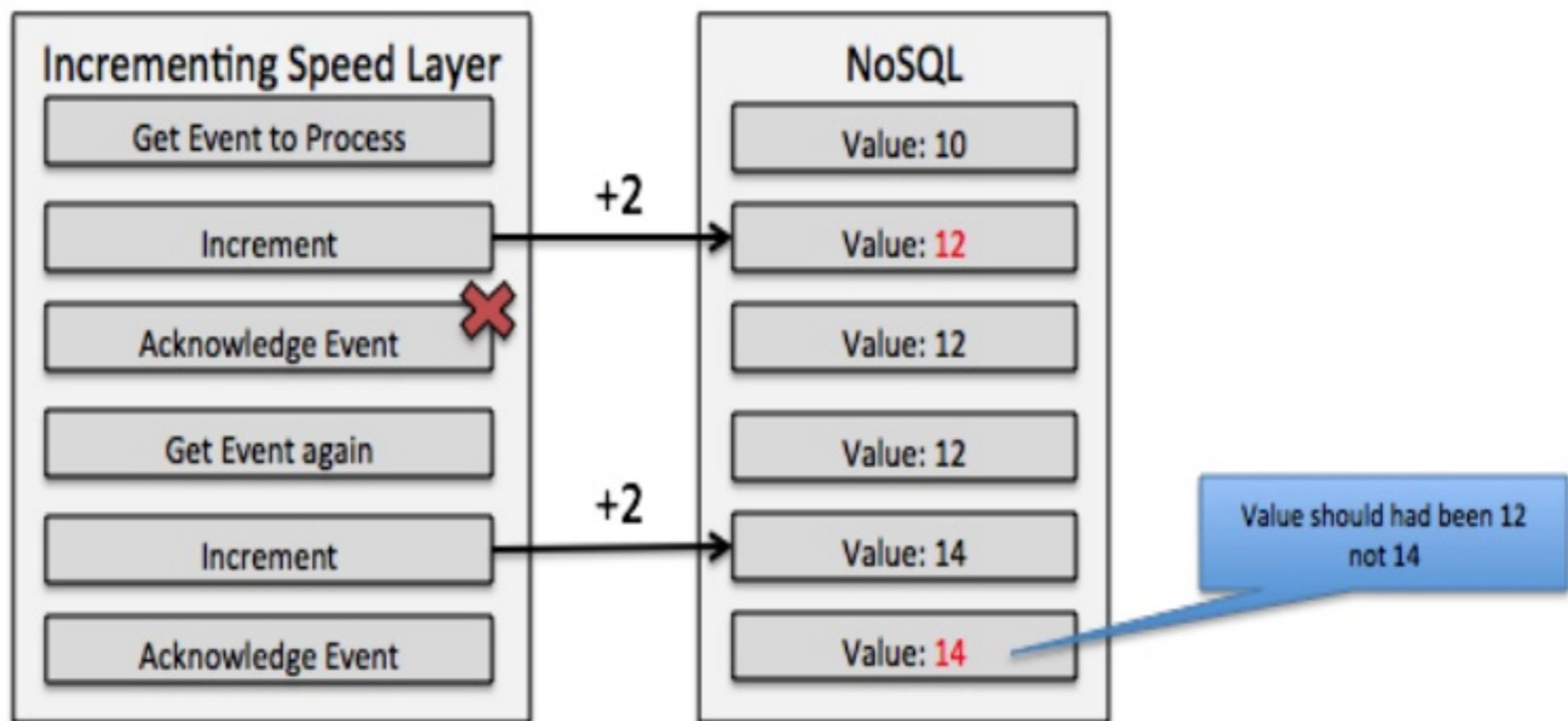


**#2 – Assuming everything
adds up perfectly
(exactly-once)**

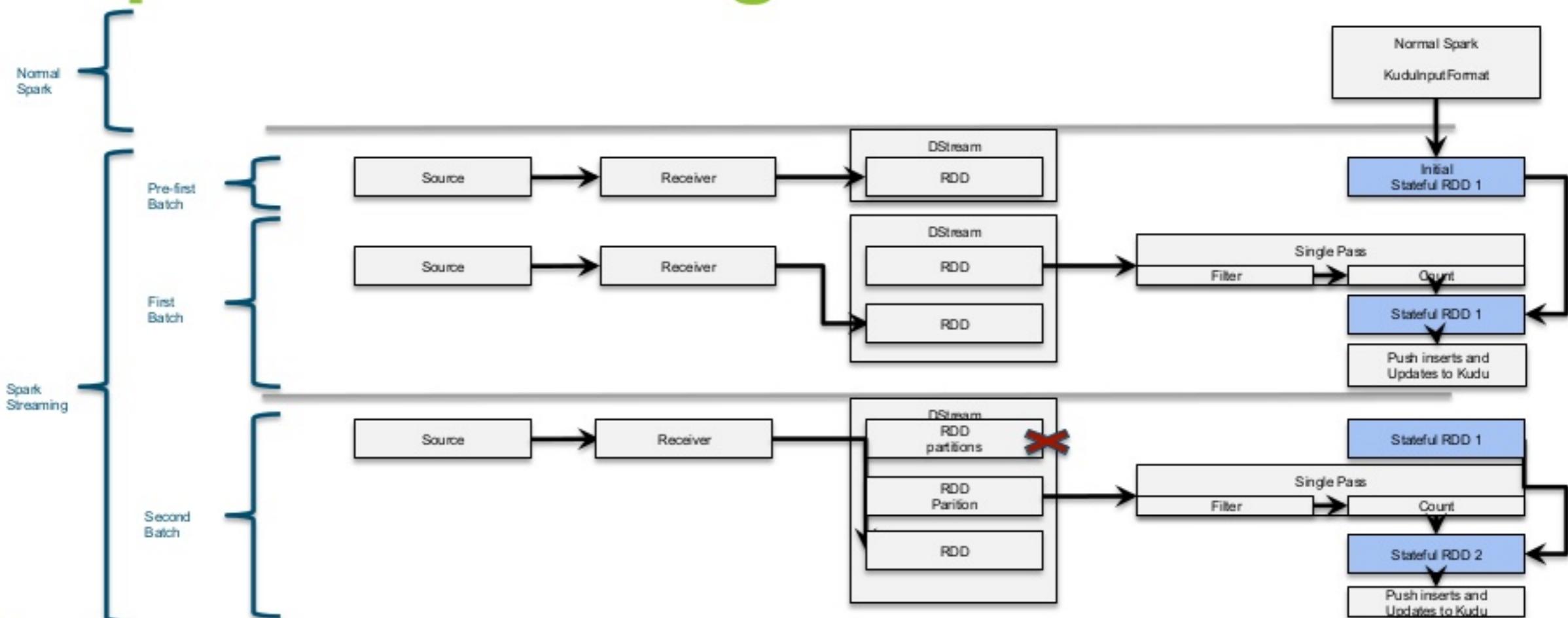
Exactly once semantics

- No duplicate records
- Perfect counting

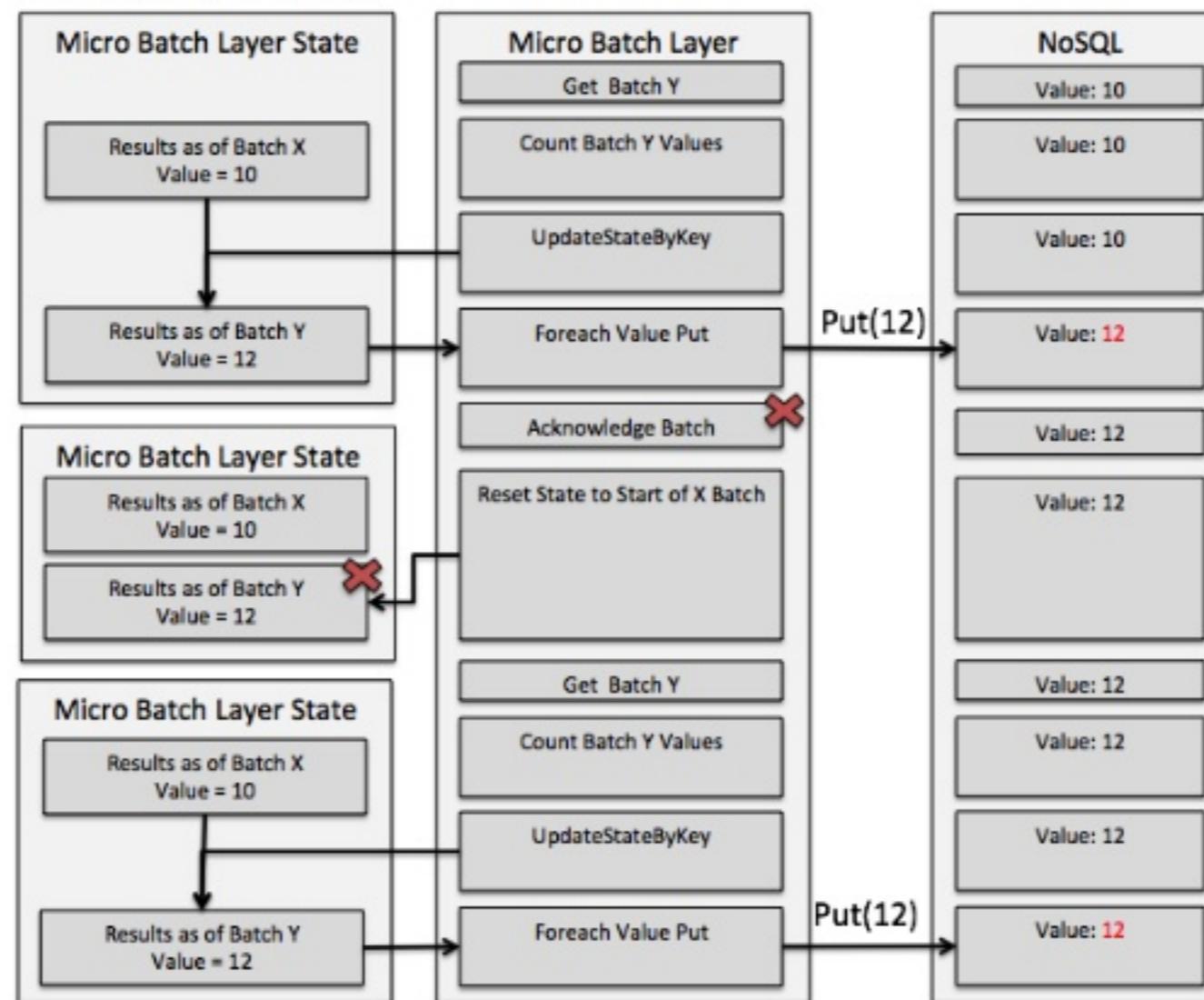
In the days of Lambda



Spark Streaming State



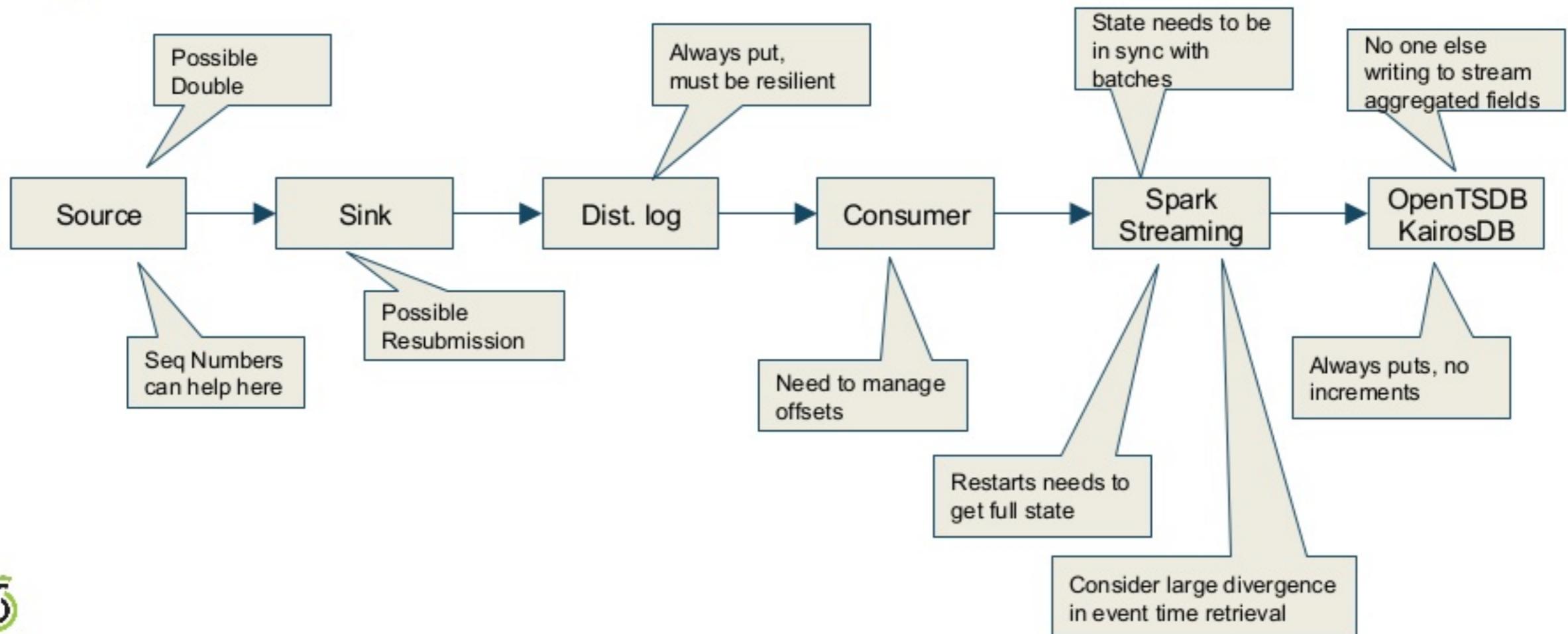
State within



Things can go wrong in many places



Things can go wrong in many places



Summary - Exactly once semantics

- Seq number tied to sources
- Puts for external storage system
- Consider large divergence in event time retrieval
 - Increase divergence window
 - Retrieve state from external storage system *
 - Ignore
 - Process off line
 - Source aggregation (pre-distributed log)



#1 – Not shutting down your streaming app gracefully

Graceful shutdown

- Can we define graceful?
 - Offsets known
 - State stored externally
 - Stopping at the right place (i.e. batch finished)

How to be graceful?

- Thread hooks

- Check for an external flag every N seconds

```
/**  
 * Stop the execution of the streams, with option of ensuring all received data  
 * has been processed.  
 *  
 * @param stopSparkContext if true, stops the associated SparkContext. The underlying SparkContext  
 * will be stopped regardless of whether this StreamingContext has been  
 * started.  
 * @param stopGracefully if true, stops gracefully by waiting for the processing of all  
 * received data to be completed  
 */  
def stop(stopSparkContext: Boolean, stopGracefully: Boolean): Unit = {
```

Under the hood of grace

```
receiverTracker.stop(processAllReceivedData) //default is to wait 10 second, grace waits until done  
jobGenerator.stop(processAllReceivedData) // Will use spark.streaming.gracefulStopTimeout  
jobExecutor.shutdown()  
val terminated = if (processAllReceivedData) {  
    jobExecutor.awaitTermination(1, TimeUnit.HOURS) // just a very large period of time  
} else {  
    jobExecutor.awaitTermination(2, TimeUnit.SECONDS)  
}  
if (!terminated) {  
    jobExecutor.shutdownNow()  
}
```

How to be graceful?

- cmd line

- `$SPARK_HOME_DIR/bin/spark-submit --master $MASTER_REST_URL --kill $DRIVER_ID`
- `spark.streaming.stopGracefullyOnShutdown=true`

```
private def stopOnShutdown(): Unit = {
    val stopGracefully = conf.getBoolean("spark.streaming.stopGracefullyOnShutdown", false)
    logInfo(s"Invoking stop(stopGracefully=$stopGracefully) from shutdown hook")
    // Do not stop SparkContext, let its own shutdown hook stop it
    stop(stopSparkContext = false, stopGracefully = stopGracefully)
}
```

How to be graceful?

- By marker file

- Touch a file when starting the app on HDFS
 - Remove the file when you want to stop
 - Separate thread in Spark app, calls

```
streamingContext.stop(stopSparkContext =  
true, stopGracefully = true)
```

Storing offsets

- Externally in ZK, HDFS, HBase, Database
- Recover on restart

Summary - Graceful shutdown

- Use provided command line, or
- Background thread with a marker file



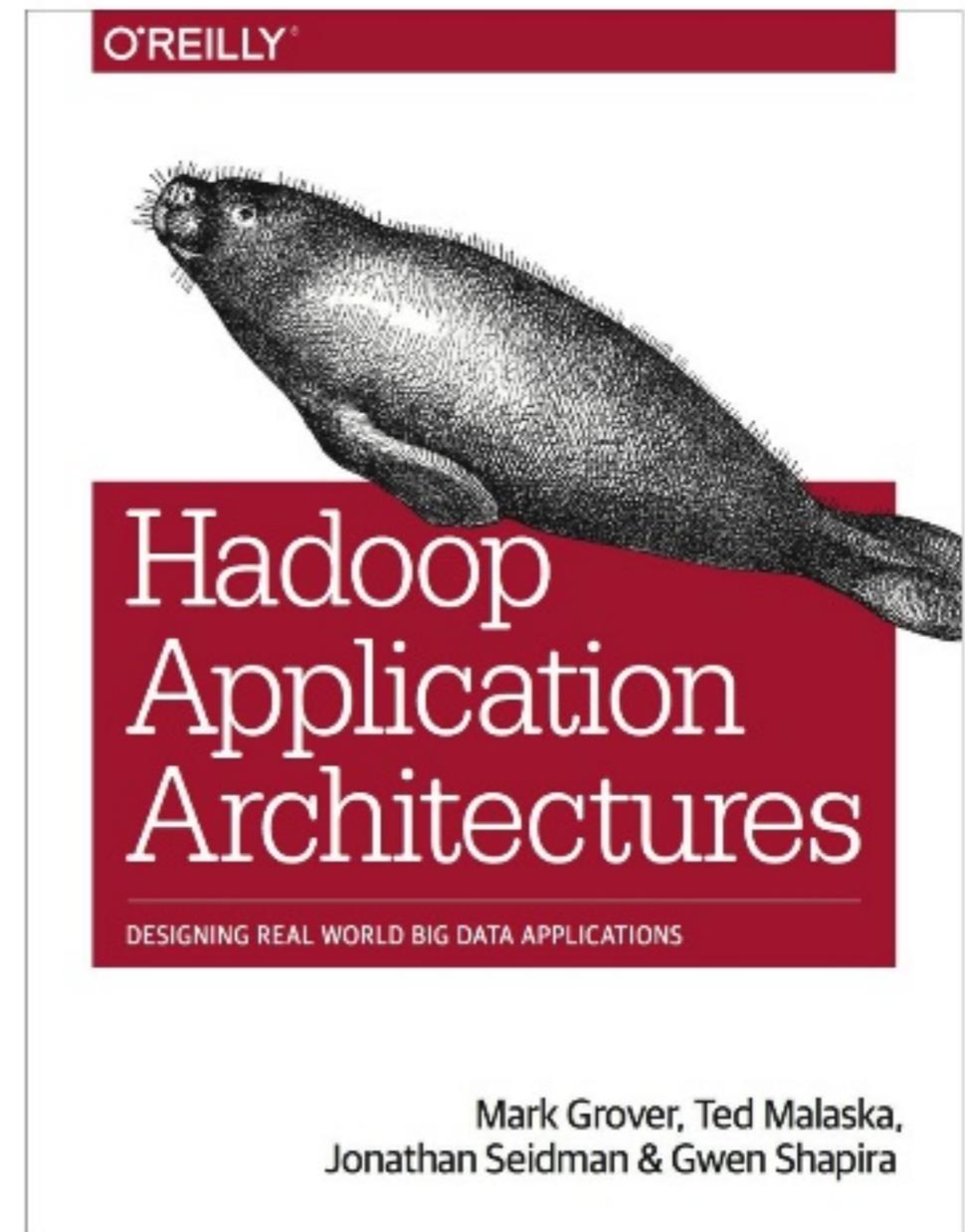
Conclusion

Conclusion

- #5 – How to monitor and manage jobs?
- #4 – How to prevent data loss?
- #3 – Do I need to use Spark Streaming?
- #2 – How to achieve exactly/effectively once semantics?
- #1 – How to gracefully shutdown your app?

Book signing

- Cloudera booth, right after this talk





Thank You.

tiny.cloudera.com/streaming-app