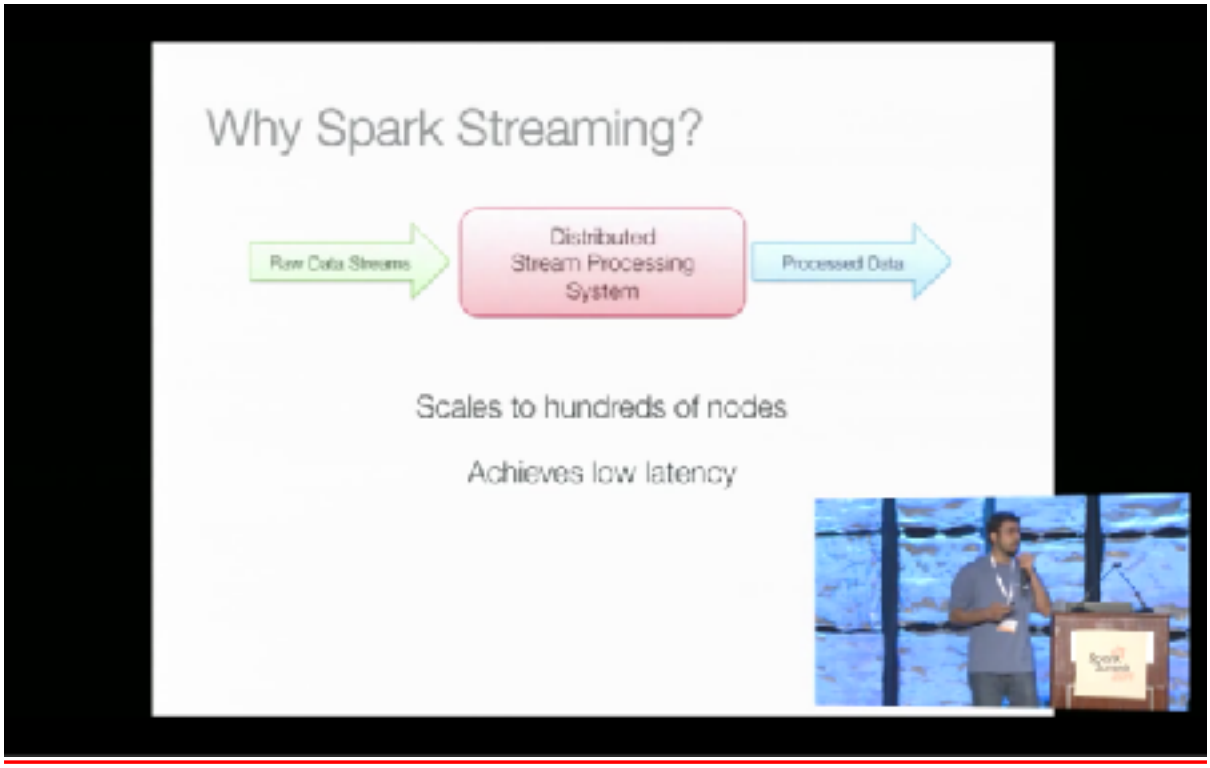
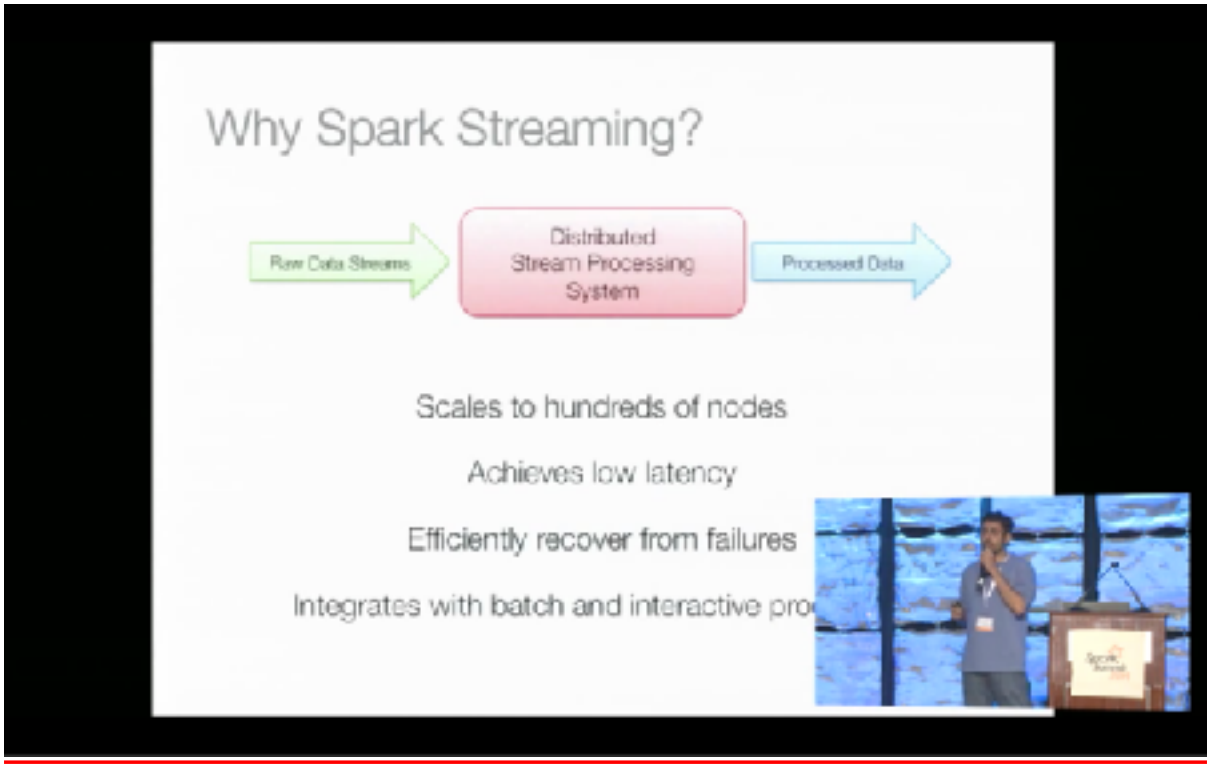
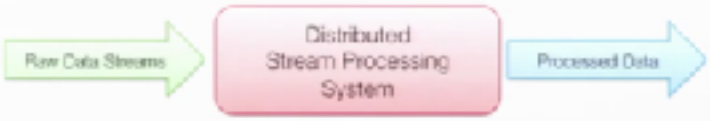
Spark Structed Streaming





Why Spark Streaming?



Scales to hundreds of nodes

Achieves low latency

Efficiently recover from failures

Integrates with batch and interactive processing

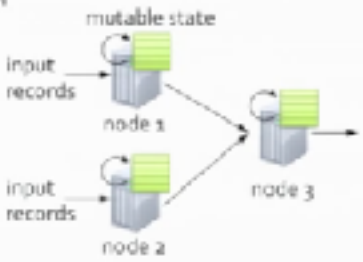
DATABRICKS

Integration with Batch Processing

- Many environments require processing same data in live streaming as well as batch post-processing
- Existing frameworks cannot do both
  - Either, stream processing of 100s of MB/s with low latency
  - Or, batch processing of TBs of data with high latency

Fault-tolerant Stream Processing

- Traditional processing model
  - Pipeline of nodes
  - Each node maintains mutable state
  - Each input record updates the state and new records are sent out



DATABRICKS

Existing Streaming Systems

- Storm
  - Replays record if not processed by a node
  - Processes each record *at least once*
  - May update mutable state twice!
  - Mutable state can be lost due to failure!
- Trident – Use transactions to update state
  - Processes each record *exactly once*
  - Per-state transaction to external database is slow

DATABRICKS

Spark Streaming

DATABRICKS

Spark Streaming

Run a streaming computation as a series of very small, deterministic batch jobs

Spark Streaming

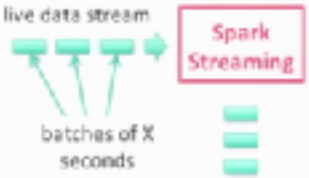
Run a streaming computation as a series of very small, deterministic batch jobs

- Chop up the live stream into batches of X seconds

Spark Streaming

Run a streaming computation as a series of very small, deterministic batch jobs

- Chop up the live stream into batches of X seconds
- Spark treats each batch of data as RDDs and processes them using RDD operations



Spark Streaming

Run a streaming computation as a series of very small, deterministic batch jobs

- Chop up the live stream into batches of X seconds
- Spark treats each batch of data as RDDs and processes them using RDD operations

DATABRICKS

Spark Streaming

Run a streaming computation as a series of very small, deterministic batch jobs

- Chop up the live stream into batches of X seconds
- Spark treats each batch of data as RDDs and processes them using RDD operations

- Finally, the processed results of the RDD operations are returned in batches

DATABRICKS

Spark Streaming

Run a streaming computation as a series of very small, deterministic batch jobs

- Batch sizes as low as ½ sec, latency of about 1 sec
- Potential for combining batch processing and streaming processing in the same system

DATABRICKS

Programming Model - DStream

- Discretized Stream (DStream)
  - Represents a stream of data
  - Implemented as a sequence of RDDs

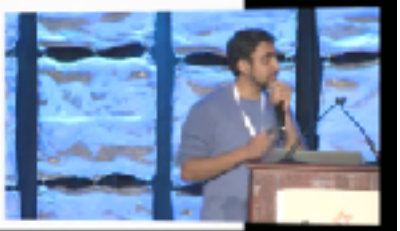
Programming Model - DStream

- Discretized Stream (DStream)
  - Represents a stream of data
  - Implemented as a sequence of RDDs
- DStreams can be either…
  - Created from streaming input sources
  - Created by applying transformations on existing DStreams

DATABRICKS

Example – Get hashtags from Twitter

```
val ssc = new StreamingContext(sparkContext, Seconds(1))
val tweets = TwitterUtils.createStream(ssc, auth)
```



Example – Get hashtags from Twitter

```
val ssc = new StreamingContext(sparkContext, Seconds(1))
val tweets = TwitterUtils.createStream(ssc, auth)
```

Input DStream

Twitter Streaming API | batch @ t | batch @ t+1 | batch @ t+2

Example – Get hashtags from Twitter

```
val ssc = new StreamingContext(sparkContext, Seconds(1))
val tweets = TwitterUtils.createStream(ssc, auth)
```

Input DStream

Twitter Streaming API | batch @ t | batch @ t+1 | batch @ t+2

tweets DStream

Example – Get hashtags from Twitter

```
val ssc = new StreamingContext(sparkContext, Seconds(1))
val tweets = TwitterUtils.createStream(ssc, auth)
```

Input DStream

stored in memory as RDDs

DATABRICKS

Example – Get hashtags from Twitter

```
val tweets = TwitterUtils.createStream(ssc, None)
val hashTags = tweets.flatMap(status => getTags(status))
```

DATABRICKS

Example – Get hashtags from Twitter

```
val tweets = TwitterUtils.createStream(ssc, None)
val hashTags = tweets.flatMap(status => getTags(status))
```

transformation: modify data in one DStream to create another DStream

DATABRICKS

Example – Get hashtags from Twitter

```
val tweets = TwitterUtils.createStream(ssc, None)
val hashTags = tweets.flatMap(status => getTags(status))
```

transformed DStream

transformation: modify data in one DStream to create another DStream

DATABRICKS

Example – Get hashtags from Twitter

```
val tweets = TwitterUtils.createStream(ssc, None)
val hashTags = tweets.flatMap(status => getTags(status))
```

transformed DStream

transformation: modify data in one DStream to create another DStream

DATABRICKS

Example – Get hashtags from Twitter

```
val tweets = TwitterUtils.createStream(ssc, None)
val hashTags = tweets.flatMap(status => getTags(status))
```

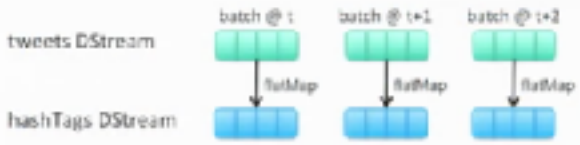
transformed DStream

transformation: modify data in one DStream to create another DStream

DATABRICKS

Example – Get hashtags from Twitter

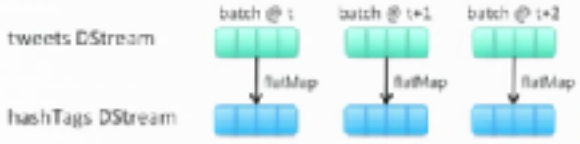
```
val tweets = TwitterUtils.createStream(ssc, None)
val hashTags = tweets.flatMap(status => getTags(status))
hashTags.saveAsHadoopFiles("hdfs://...")
```



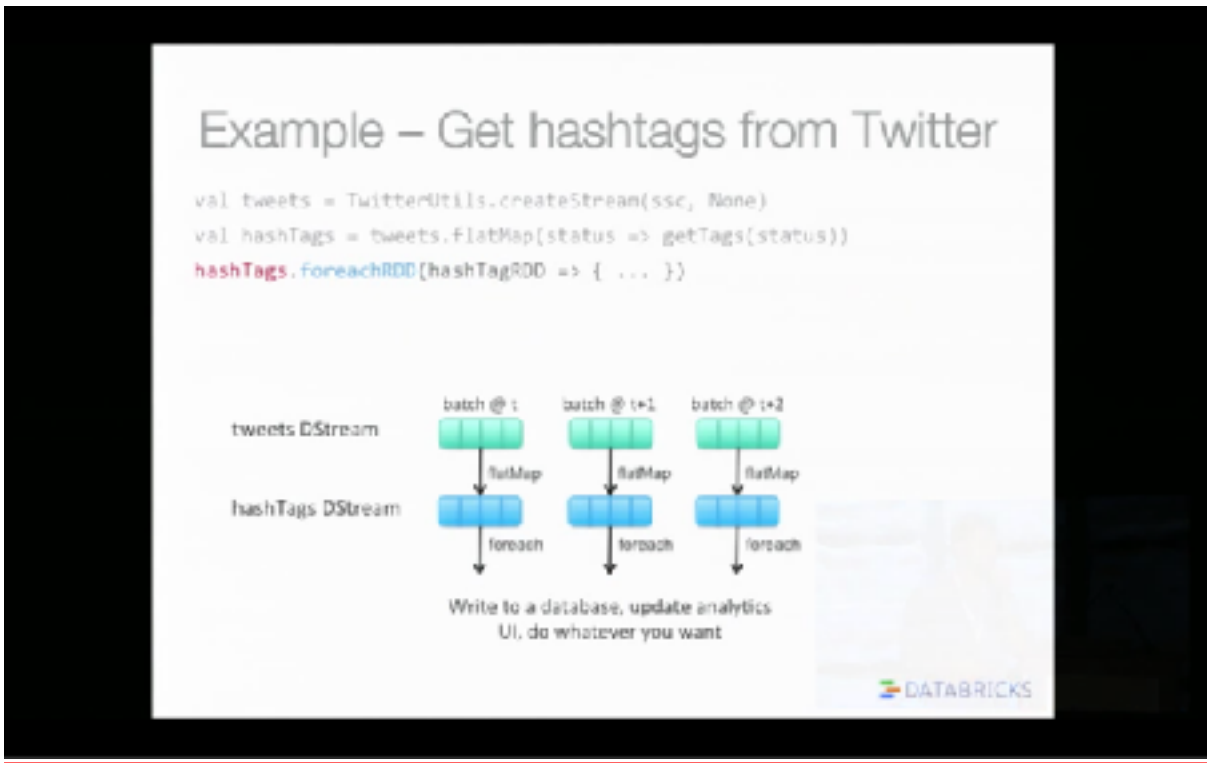
Example – Get hashtags from Twitter

```
val tweets = TwitterUtils.createStream(ssc, None)
val hashTags = tweets.flatMap(status => getTags(status))
hashTags.saveAsHadoopFiles("hdfs://...")
```

output operation: to push data to external storage




Spark Streaming Advanced Training by Lead Developer Tathagata Das (Databri...
Example – Get hashtags from Twitter
val tweets = TwitterUtils.createStream(ssc, None)
val hashTags = tweets.flatMap(status => getTags(status))
hashTags.saveAsHadoopFiles("hdfs://...")
output operation: to push data to external storage
batch @ t
batch @ t+1
batch @ t+2
tweets DStream
hashTags DStream
flatMap
flatMap
flatMap
save
save
save
12:23 / 29:52


Example – Get hashtags from Twitter
val tweets = TwitterUtils.createStream(ssc, None)
val hashTags = tweets.flatMap(status => getTags(status))
hashTags.foreachRDD(hashTagRDD => { ... })
batch @ t
batch @ t+1
batch @ t+2
tweets DStream
hashTags DStream
flatMap
flatMap
flatMap
foreach
foreach
foreach
Write to a database, update analytics
UI, do whatever you want
DATABRICKS

Languages

Scala API

```
val tweets = TwitterUtils.createStream(ssc, auth)
val hashTags = tweets.flatMap(status => getTags(status))
hashTags.saveAsHadoopFiles("hdfs://...")
```

Java API

```
JavaDStream<Status> tweets = ssc.twitterStream()
JavaDStream<String> hashTags = tweets.flatMap(new Function<...> { })
hashTags.saveAsHadoopFiles("hdfs://...")
```

Python API

```
...soon
```

Languages

Scala API

```
val tweets = TwitterUtils.createStream(ssc, auth)
val hashTags = tweets.flatMap(status => getTags(status))
hashTags.saveAsHadoopFiles("hdfs://...")
```

Java API

Function object

```
JavaDStream<Status> tweets = ssc.twitterStream()
JavaDStream<String> hashTags = tweets.flatMap(new Function<...> { })
hashTags.saveAsHadoopFiles("hdfs://...")
```

Python API

```
...soon
```

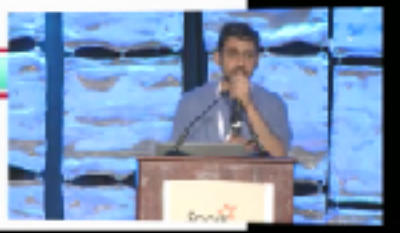
Window-based Transformations

```
val tweets = TwitterUtils.createStream(ssc, auth)
val hashTags = tweets.flatMap(status => getTags(status))
val tagCounts = hashTags.window(Minutes(1), Seconds(5)).countByValue()
```

Window-based Transformations

```
val tweets = TwitterUtils.createStream(ssc, auth)
val hashTags = tweets.flatMap(status => getTags(status))
val tagCounts = hashTags.window(Minutes(1), Seconds(5)).countByValue()
```

sliding window operation

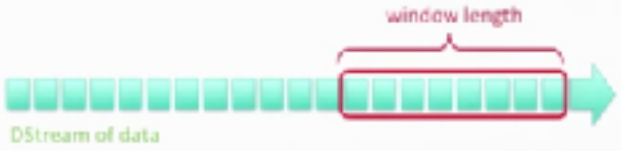


Window-based Transformations

```
val tweets = TwitterUtils.createStream(ssc, auth)
val hashTags = tweets.flatMap(status => getTags(status))
val tagCounts = hashTags.window(Minutes(1), Seconds(5)).countByValue()
```

sliding window operation

window length



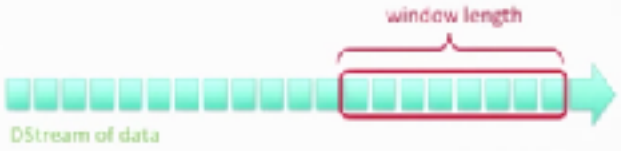
DATABRICKS

Window-based Transformations

```
val tweets = TwitterUtils.createStream(ssc, auth)
val hashTags = tweets.flatMap(status => getTags(status))
val tagCounts = hashTags.window(Minutes(1), Seconds(5)).countByValue()
```

sliding window operation

window length



DATABRICKS

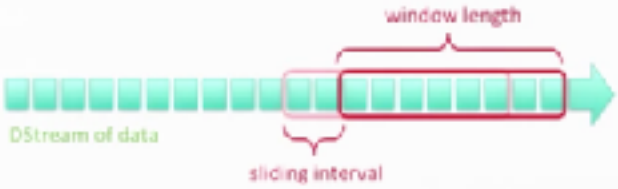
Window-based Transformations

```
val tweets = TwitterUtils.createStream(ssc, auth)
val hashTags = tweets.flatMap(status => getTags(status))
val tagCounts = hashTags.window(Minutes(1), Seconds(5)).countByValue()
```

sliding window operation

window length

sliding interval



DATABRICKS

Arbitrary Stateful Computations

Specify function to generate new state based on previous state and new data

- Example: Maintain per-user mood as state, and update it with their tweets

```
def updateMood(newTweets, lastMood) => newMood

val moods = tweetsByUser.updateStateByKey(updateMood _)
```

DATABRICKS

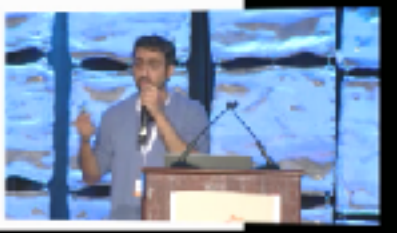
DStreams + RDDs = Power

- Combine live data streams with historical data
  - Generate historical data models with Spark, etc.
  - Use data models to process live data stream

DStreams + RDDs = Power

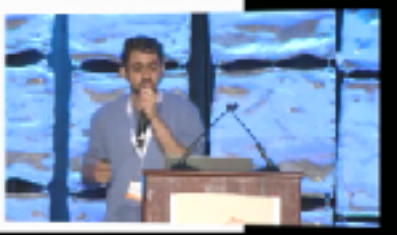
- Combine live data streams with historical data
  - Generate historical data models with Spark, etc.
  - Use data models to process live data stream

- Combine streaming with MLlib, GraphX algos
  - Offline learning, online prediction
  - Online learning and prediction



DStreams + RDDs = Power

- Combine live data streams with historical data
  - Generate historical data models with Spark, etc.
  - Use data models to process live data stream
- Combine streaming with MLlib, GraphX algos
  - Offline learning, online prediction
  - Online learning and prediction
- Query streaming data using SQL
  - select * from table_from_streaming_data



Databricks Keynote Demo

```
val ssc = new StreamingContext(sc, Seconds(5))
val sqlContext = new SQLContext(sc)
val tweets = TwitterUtils.createStream(ssc, auth)
val transformed = tweets.filter(isEnglish).window(Minutes(1))
```

Databricks Keynote Demo

```
val ssc = new StreamingContext(sc, Seconds(5))
val sqlContext = new SQLContext(sc)
val tweets = TwitterUtils.createStream(ssc, auth)
val transformed = tweets.filter(isEnglish).window(Minutes(1))

transformed.foreachRDD { rdd =>
  // Tweet is a case class containing necessary
  rdd.map(Tweet.apply(_)).registerAsTable("tweets")
}
```

```
SELECT text FROM tweets WHERE similarity(tweet) > 0.01
SELECT getClosestCountry(lat, long) FROM tweets
```

DATABRICKS

Advantage of an Unified Stack

- Explore data interactively to identify problems

```
$ ./spark-shell
scala> val file = sc.hadoopFile("smallLogs")
...
scala> val filtered = file.filter(_.contains("ERROR"))
...
scala> val mapped = filtered.map(...)
...
```

Advantage of an Unified Stack

- Explore data interactively to identify problems
- Use same code in Spark for processing large logs

```
$ ./spark-shell
scala> val file = sc.hadoopFile("smallLogs")
...
scala> val filtered = file.filter(_.contains("ERROR"))
...
scala> val mapped = filtered.map(...)
...
```

```
object ProcessProductionData {
  def main(args: Array[String]) {
    val sc = new SparkContext(...)
    val file = sc.hadoopFile("productionLogs")
    val filtered = file.filter(_.contains("ERROR"))
    val mapped = filtered.map(...)
    ...
  }
}
```

Advantage of an Unified Stack

- Explore data interactively to identify problems
- Use same code in Spark for processing large logs

- Use similar code in Spark Streaming for realtime processing


```
$ ./spark-shell
scala> val file = sc.hadoopFile("smallLogs")
...
scala> val filtered = file.filter(_.contains("ERROR"))
...
scala> val mapped = filtered.map(...)
...
```

```
object ProcessProductionData {
  def main(args: Array[String]) {
    val sc = new SparkContext(...)
    val file = sc.hadoopFile("productionLogs")
    val filtered = file.filter(_.contains("ERROR"))
    val mapped = filtered.map(...)
    ...
  }
}
```


```
object ProcessLiveStream {
  def main(args: Array[String]) {
    val sc = new StreamingContext(...)
    val stream = KafkaUtil.createStream(...)
    val filtered = stream.filter(_.contains("ERROR"))
    val mapped = filtered.map(...)
    ...
  }
}
```

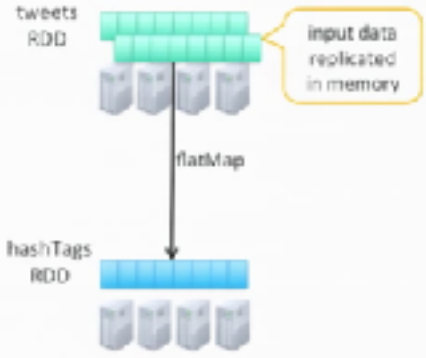
DATABRICKS

Fault-tolerance

DATABRICKS

Fault-tolerance

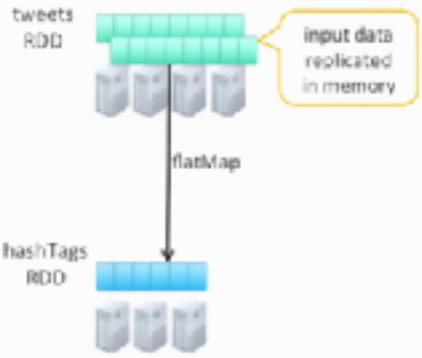
- Batches of input data are replicated in memory for fault-tolerance



DATABRICKS

Fault-tolerance

- Batches of input data are replicated in memory for fault-tolerance
- Data lost due to worker failure, can be recomputed from replicated input data



DATABRICKS

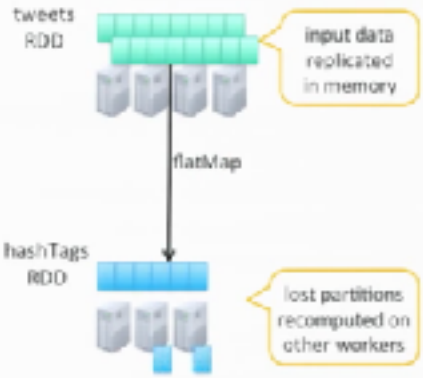
Fault-tolerance

- Batches of input data are replicated in memory for fault-tolerance
- Data lost due to worker failure, can be recomputed from replicated input data

DATABRICKS

Fault-tolerance

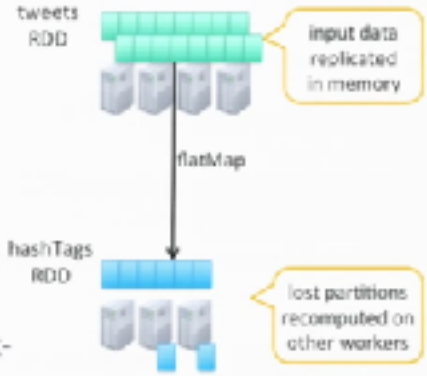
- Batches of input data are replicated in memory for fault-tolerance
- Data lost due to worker failure, can be recomputed from replicated input data



DATABRICKS

Fault-tolerance

- Batches of input data are replicated in memory for fault-tolerance
- Data lost due to worker failure, can be recomputed from replicated input data
- All transformations are fault-tolerant, and *exactly-once* transformations



DATABRICKS

Input Sources

- Out of the box, we provide
  - Kafka, Flume, Akka Actors, Raw TCP sockets, HDFS, etc.
- Very easy to write a custom *receiver*
  - Define what to when receiver is started and stopped
- Also, generate your own sequence of RDDs, etc. and push them in as a "stream"

Future Directions!

- Better integration with SQL, MLlib, GraphX
- Improved integration with different sources
- Improved control over data rates
- Python API

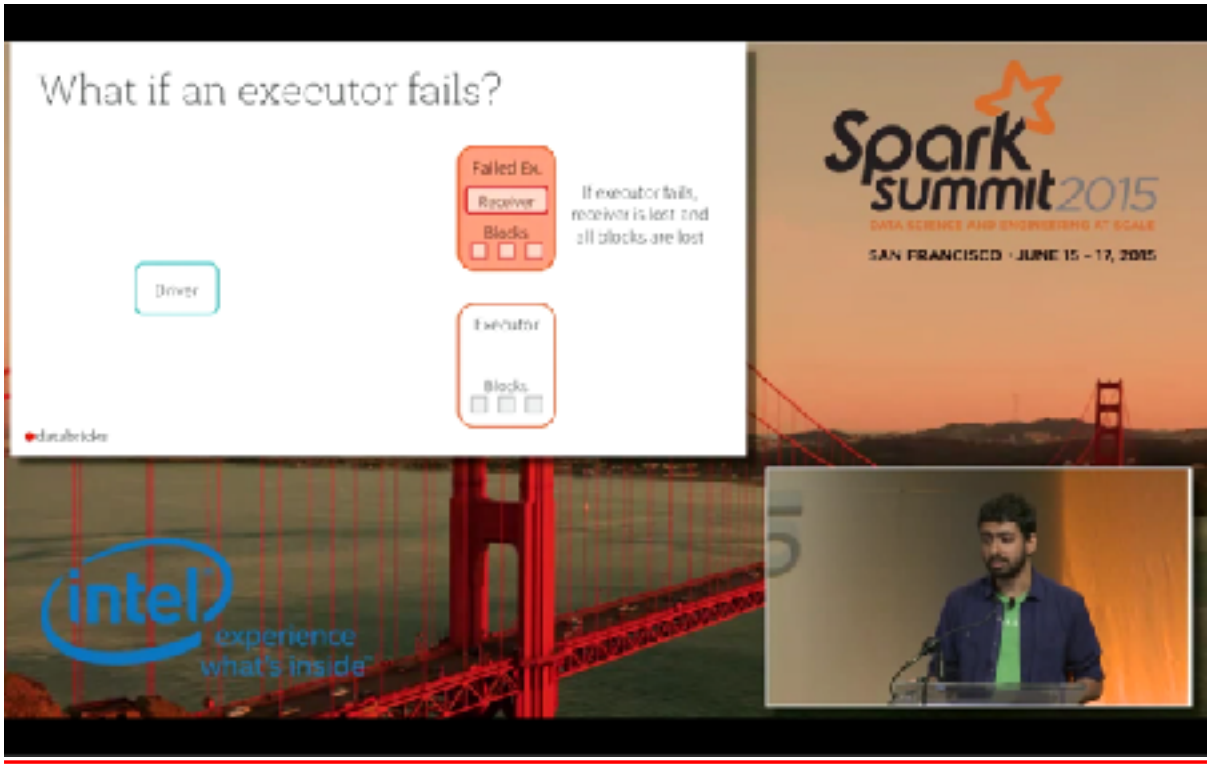
Spark Streaming Failures & Recovery

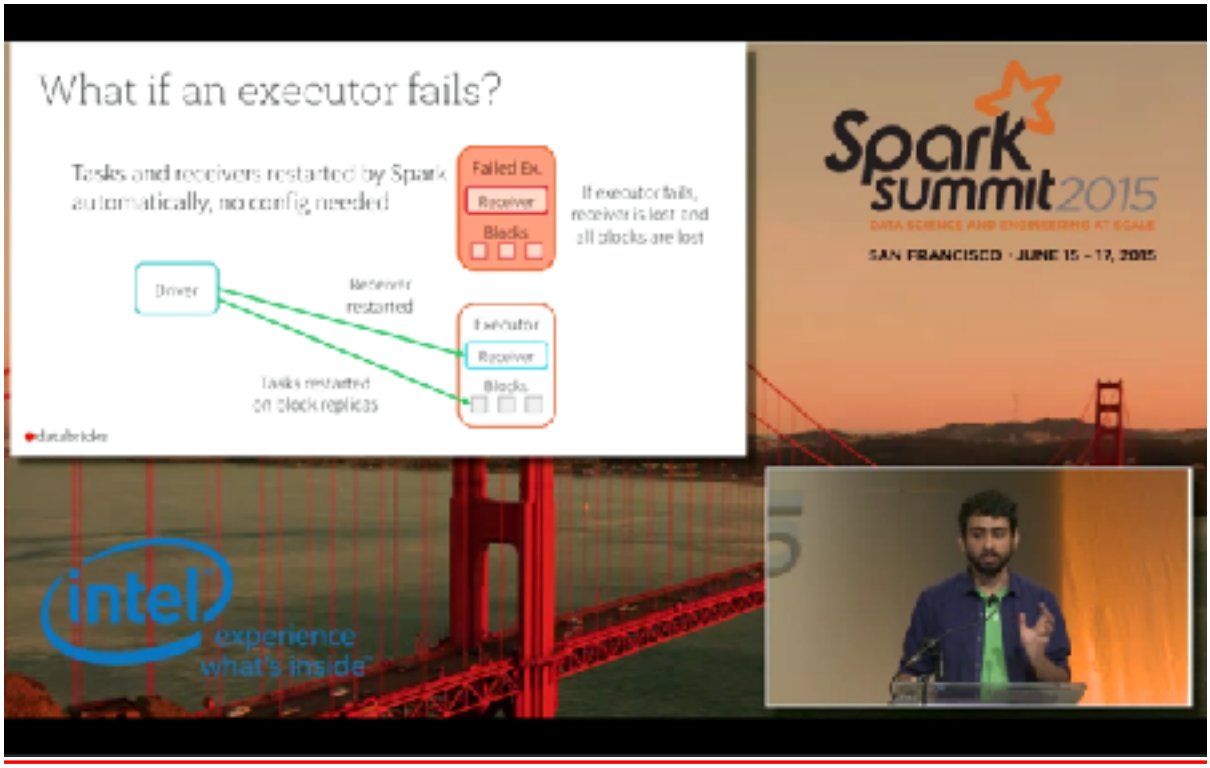
Executor /driver failures

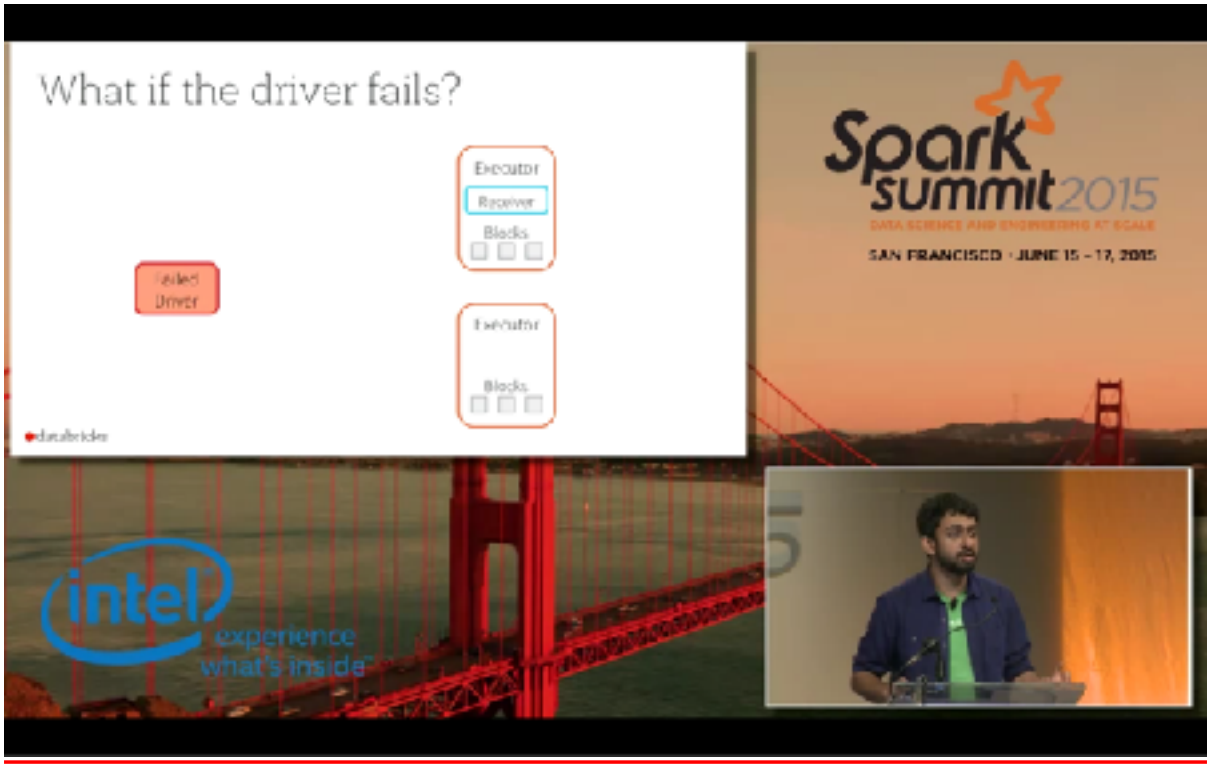
https://www.youtube.com/watch?v=d5UJonrruHk


Failures? Why care?
Many streaming applications need zero data loss guarantees despite any kind of failures in the system
At least once guarantee – every record processed at least once
Exactly once guarantee – every record processed exactly once
Different kinds of failures – executor and driver
Some failures and guarantee requirements need additional configurations and setups
databricks
Spark summit 2015
SAN FRANCISCO · JUNE 15 – 17, 2015
intel
experience what's inside

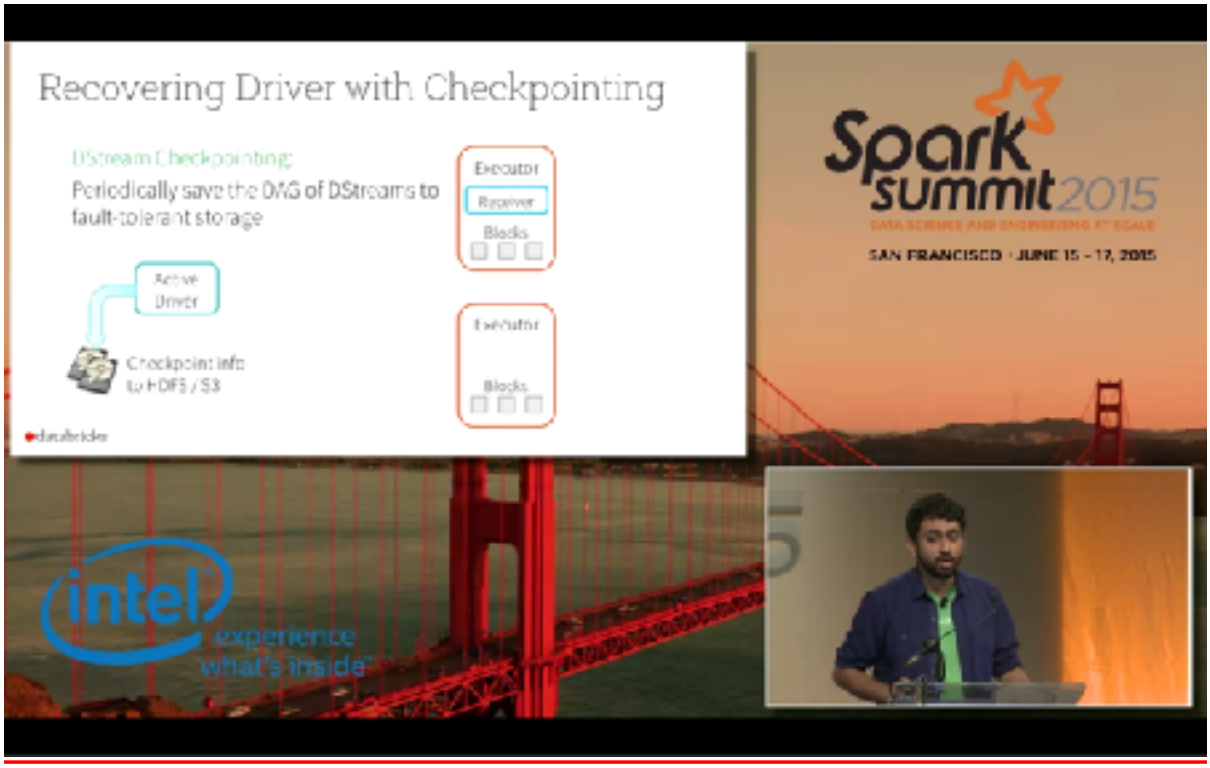

What if an executor fails?
Executor
Receiver
Data Blocks
Driver
Executor
Blocks
databricks
Spark summit 2015
SAN FRANCISCO · JUNE 15 – 17, 2015
intel
experience what's inside


What if an executor fails?
Failed Ex.
Receiver
Blocks
If executor fails, receiver is lost and all blocks are lost
Driver
Executor
Blocks
databricks
Spark summit 2015
SAN FRANCISCO · JUNE 15 – 17, 2015
intel
experience what's inside


What if an executor fails?
Tasks and receivers restarted by Spark automatically, no config needed
Failed Ex.
Receiver
Blocks
If executor fails, receiver is lost and all blocks are lost
Driver
Receiver restarted
Executor
Receiver
Blocks
Tasks restarted on block replicas
databricks
Spark summit 2015
SAN FRANCISCO · JUNE 15 – 17, 2015
intel
experience what's inside

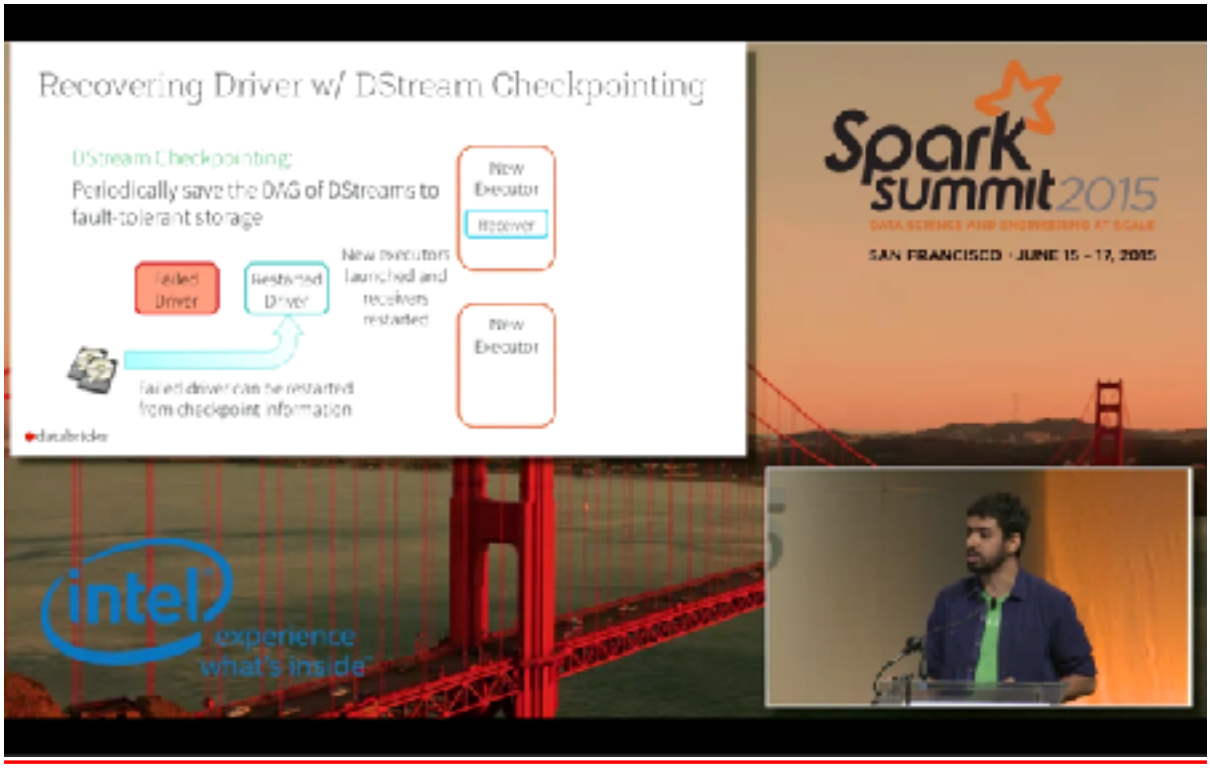

What if the driver fails?
Executor
Receiver
Blocks
Failed Driver
Executor
Blocks
databricks
Spark summit 2015
SAN FRANCISCO · JUNE 15 – 17, 2015
intel
experience what's inside

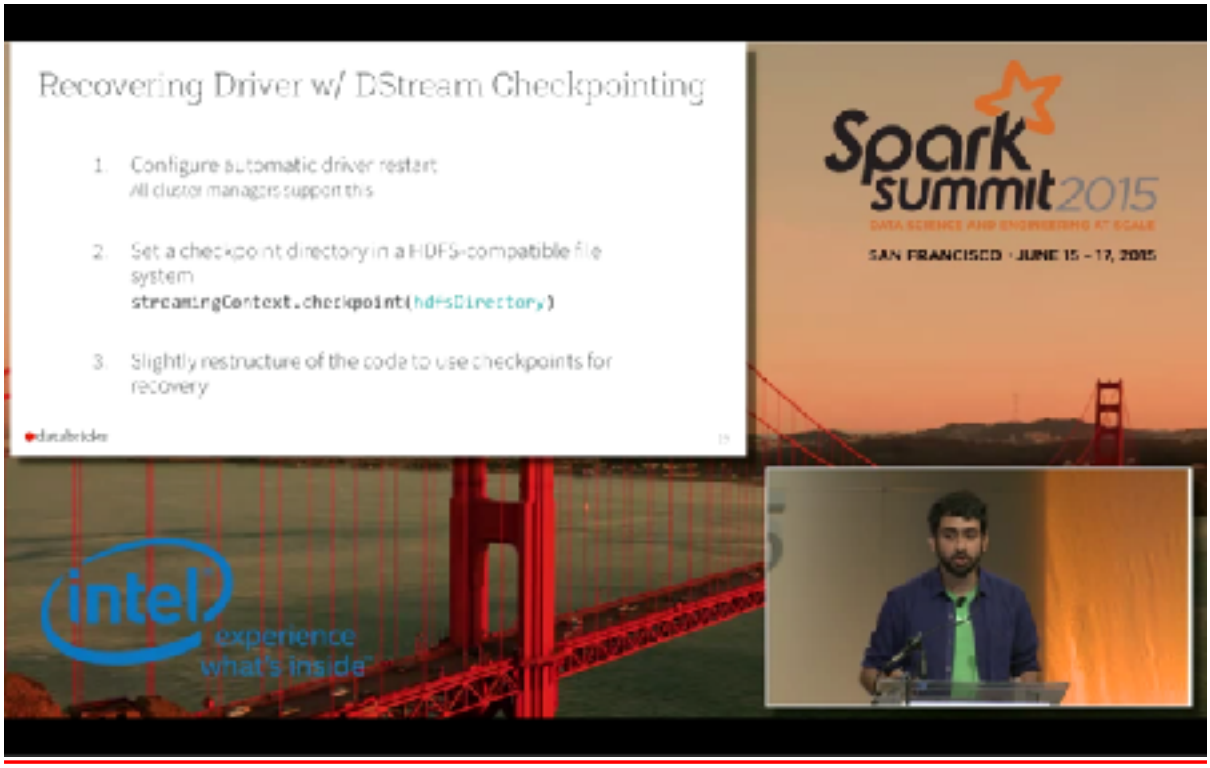

What if the driver fails?
Failed Ex.
Receiver
Blocks
When the driver fails, all the executors fail
Failed Driver
All computation, all received blocks are lost
Failed Executor
Blocks
How do we recover?
databricks
Spark summit 2015
SAN FRANCISCO · JUNE 15 – 17, 2015
intel
experience what's inside

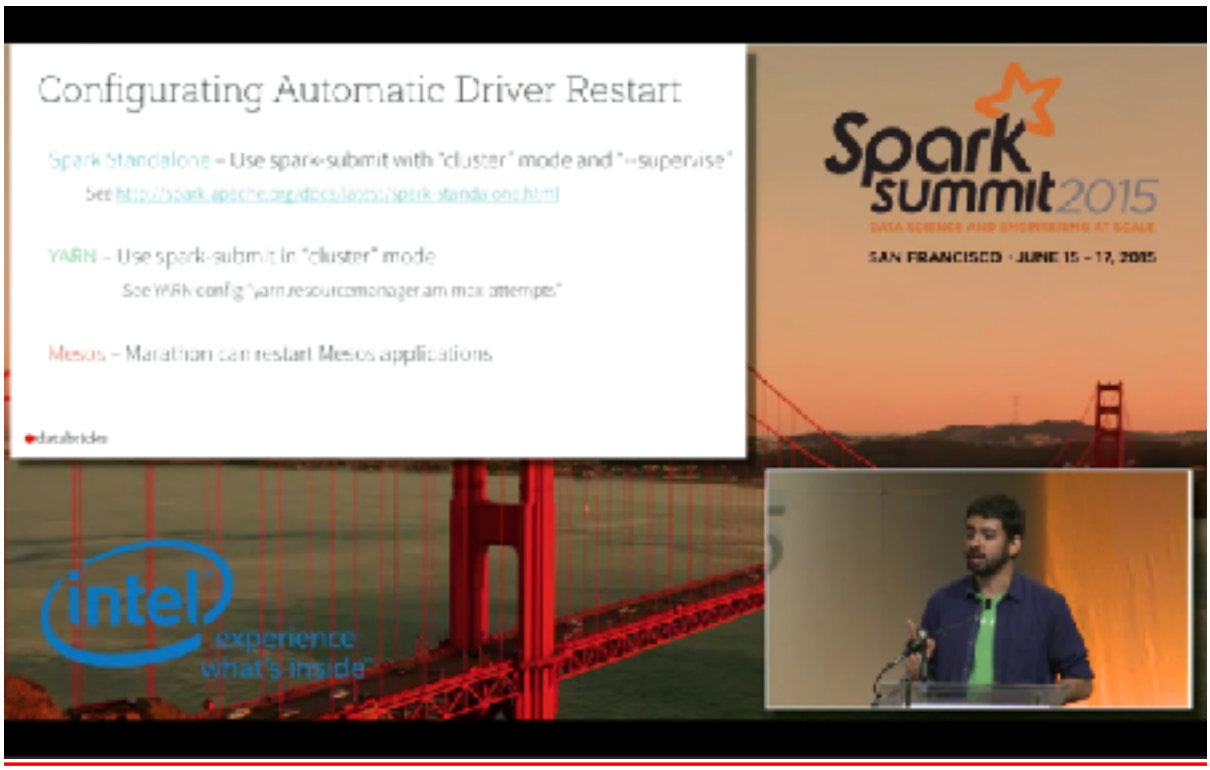

Recovering Driver with Checkpointing
DStream Checkpointing:
Periodically save the DAG of DStreams to fault-tolerant storage
Executor
Receiver
Blocks
Active Driver
Checkpoint info to HDFS / S3
Executor
Blocks
databricks
Spark summit 2015
SAN FRANCISCO · JUNE 15 – 17, 2015
intel
experience what's inside


Recovering Driver w/ DStream Checkpointing
DStream Checkpointing:
Periodically save the DAG of DStreams to fault-tolerant storage
Failed Driver
databricks
Spark summit 2015
SAN FRANCISCO · JUNE 15 – 17, 2015
intel
experience what's inside

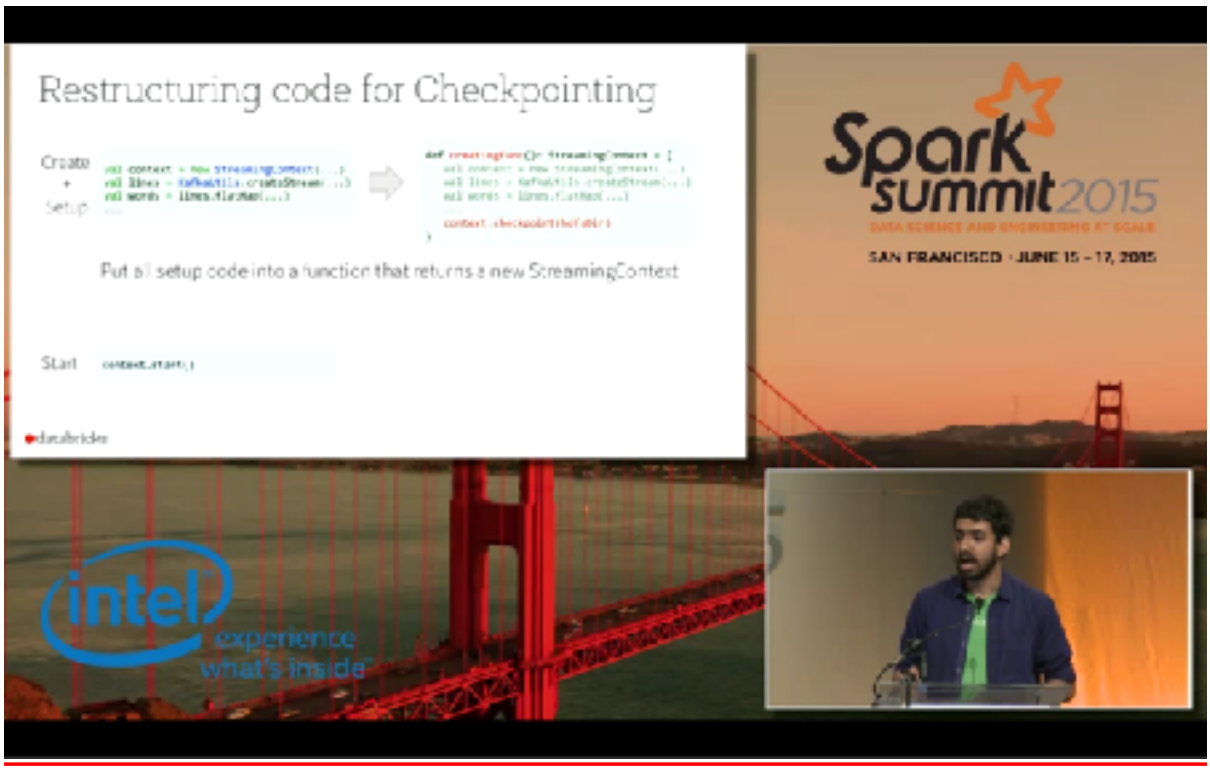

Recovering Driver w/ DStream Checkpointing
DStream Checkpointing:
Periodically save the DAG of DStreams to fault-tolerant storage
Failed Driver
Restarted Driver
Spark summit 2015
SAN FRANCISCO · JUNE 15 – 17, 2015
intel
experience what's inside


Recovering Driver w/ DStream Checkpointing
DStream Checkpointing:
Periodically save the DAG of DStreams to fault-tolerant storage
New Executor
Receiver
New executors launched and receivers restarted
Failed Driver
Restarted Driver
New Executor
Spark summit 2015
SAN FRANCISCO · JUNE 15 – 17, 2015
intel
experience what's inside


Recovering Driver w/ DStream Checkpointing
1. Configure automatic driver restart
All cluster managers support this
2. Set a checkpoint directory in a HDFS-compatible file system
streamingContext.checkpoint(hdfsDirectory)
3. Slightly restructure of the code to use checkpoints for recovery
databricks
Spark summit 2015
SAN FRANCISCO · JUNE 15 – 17, 2015
intel
experience
what's inside


Configurating Automatic Driver Restart
Spark Standalone – Use spark-submit with "cluster" mode and "--supervise"
See http://spark.apache.org/docs/latest/spark-standalone.html
YARN – Use spark-submit in "cluster" mode
See YARN config "yarn.resourcemanager.am.max-attempts"
Mesos – Marathon can restart Mesos applications
databricks
Spark summit 2015
SAN FRANCISCO · JUNE 15 – 17, 2015
intel
experience
what's inside


Restructuring code for Checkpointing
Create
+
Setup
Start
databricks
Spark summit 2015
SAN FRANCISCO · JUNE 15 – 17, 2015
intel
experience
what's inside


Restructuring code for Checkpointing
Create
+
Setup
Put all setup code into a function that returns a new StreamingContext
Start
databricks
Spark summit 2015
SAN FRANCISCO · JUNE 15 – 17, 2015
intel
experience
what's inside

Restructuring code for Checkpointing

Create + Setup

```
val context = new StreamingContext(...)
val lines = KafkaUtils.createStream(...)
val words = lines.flatMap(...)
...
```



```
def creatingFunc(): StreamingContext = {
  val context = new StreamingContext(...)
  val lines = KafkaUtils.createStream(...)
  val words = lines.flatMap(...)
  ...
  context.checkpoint(hdfsDir)
}
```

Put all setup code into a function that returns a new StreamingContext

Start

```
context.start()
```

databricks

Restructuring code for Checkpointing

Create + Setup

```
val context = new StreamingContext(...)
val lines = KafkaUtils.createStream(...)
val words = lines.flatMap(...)
...
```

```
def creatingFunc(): StreamingContext = {
  val context = new StreamingContext(...)
  val lines = KafkaUtils.createStream(...)
  val words = lines.flatMap(...)
  ...
  context.checkpoint(hdfsDir)
}
```

Put all setup code into a function that returns a new StreamingContext

Start

```
context.start()
```



```
val context =
StreamingContext.getOrCreate(
  hdfsDir, creatingFunc)
context.start()
```

Get context setup from HDFS dir OR *create* a new one with the function

databricks

Restructuring code for Checkpointing

StreamingContext.getOrCreate():

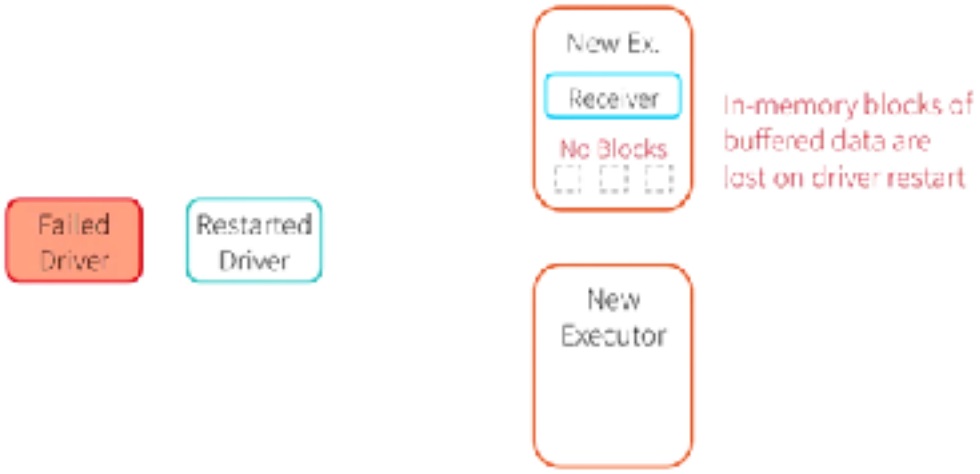
If HDFS directory has checkpoint info
    recover context from info
else
    call `creatingFunc()` to create
    and setup a new context

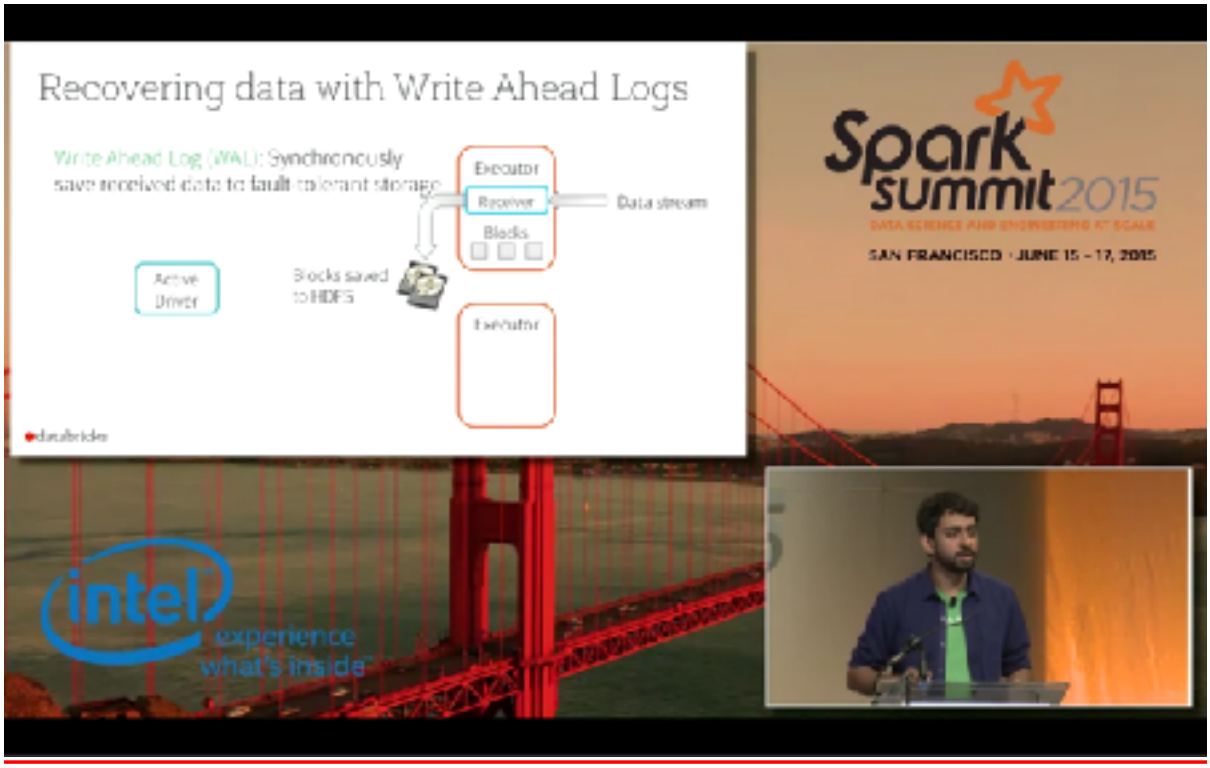
```
def creatingFunc(): StreamingContext = {
   val context = new StreamingContext(...)
   val lines = KafkaUtils.createStream(...)
   val words = lines.flatMap(...)
   ...
   context.checkpoint(hdfsDir)
}
```

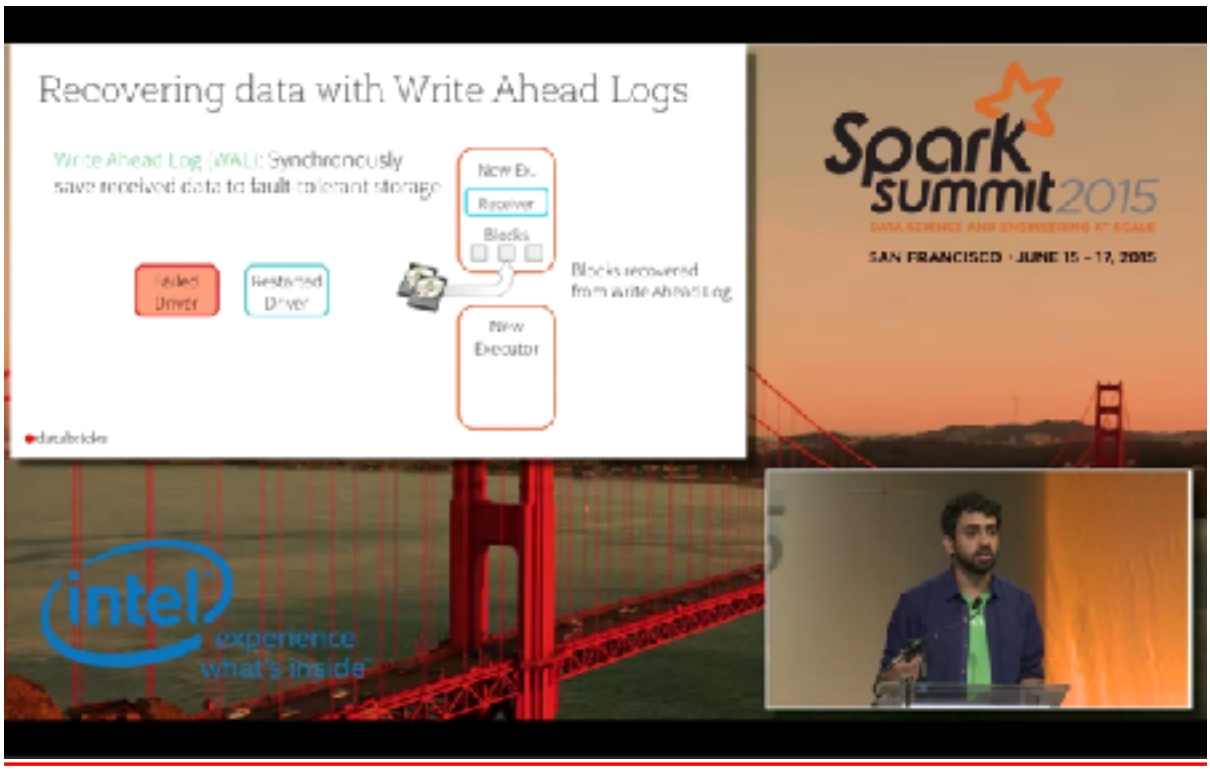
Restarted process can figure out whether to recover using checkpoint info or not

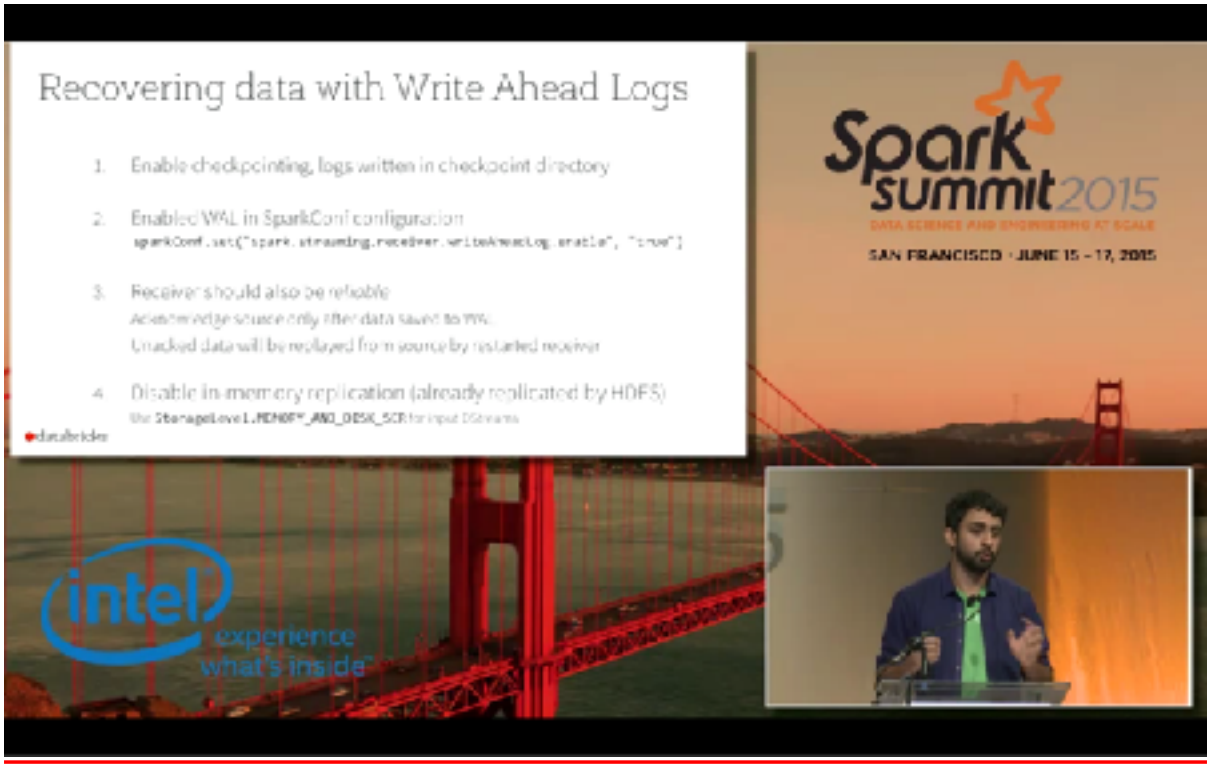
```
val context =
StreamingContext.getOrCreate(
  hdfsDir, creatingFunc)
context.start()
```

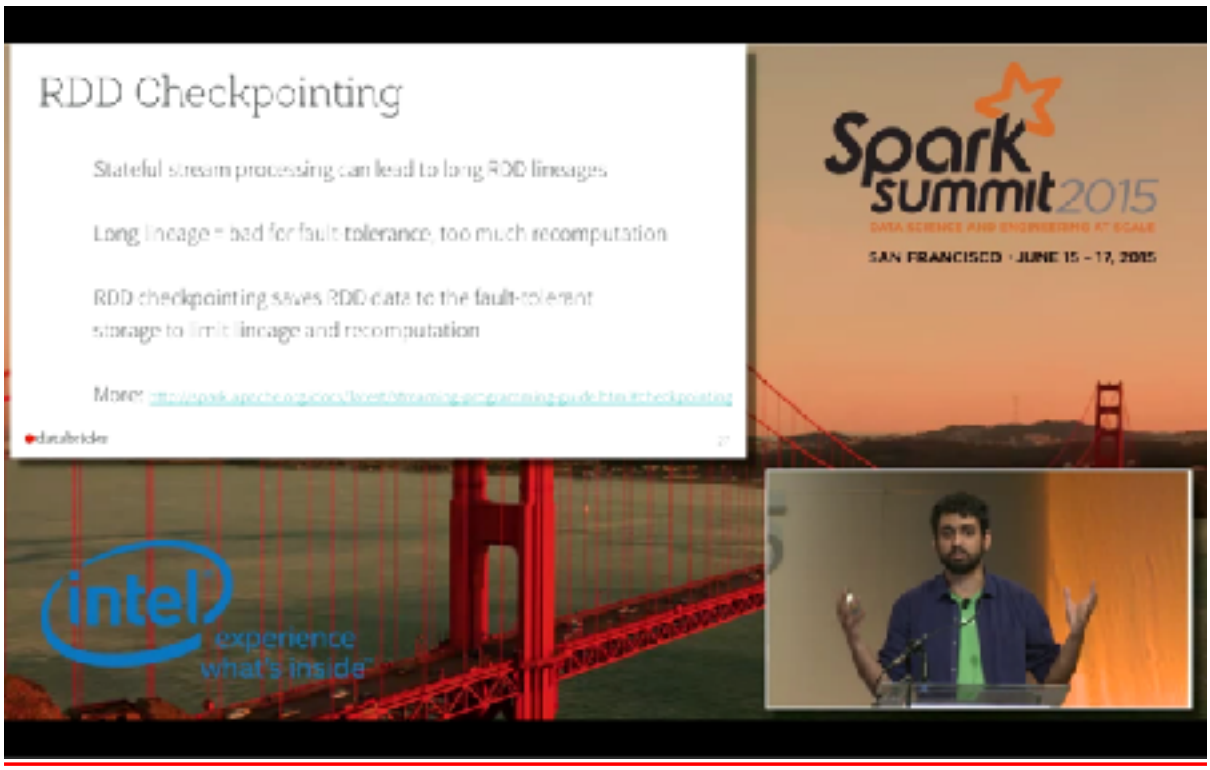

Received blocks lost on Restart!

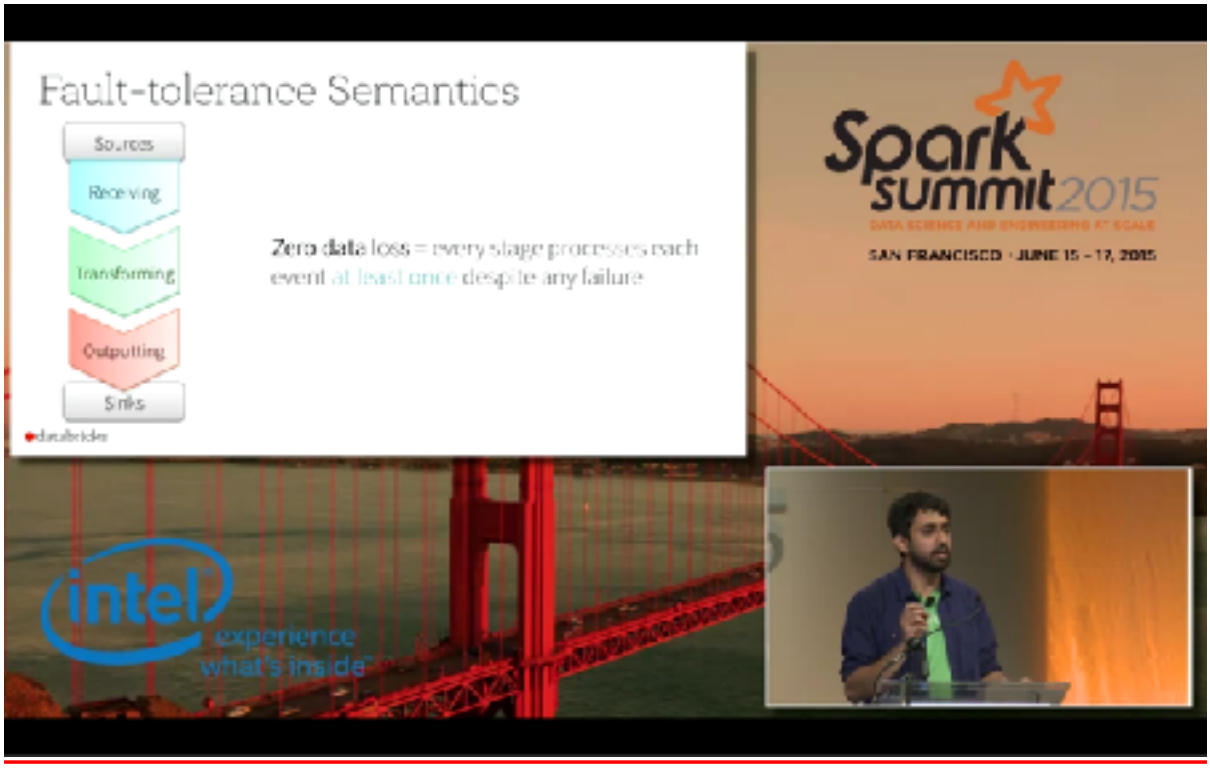


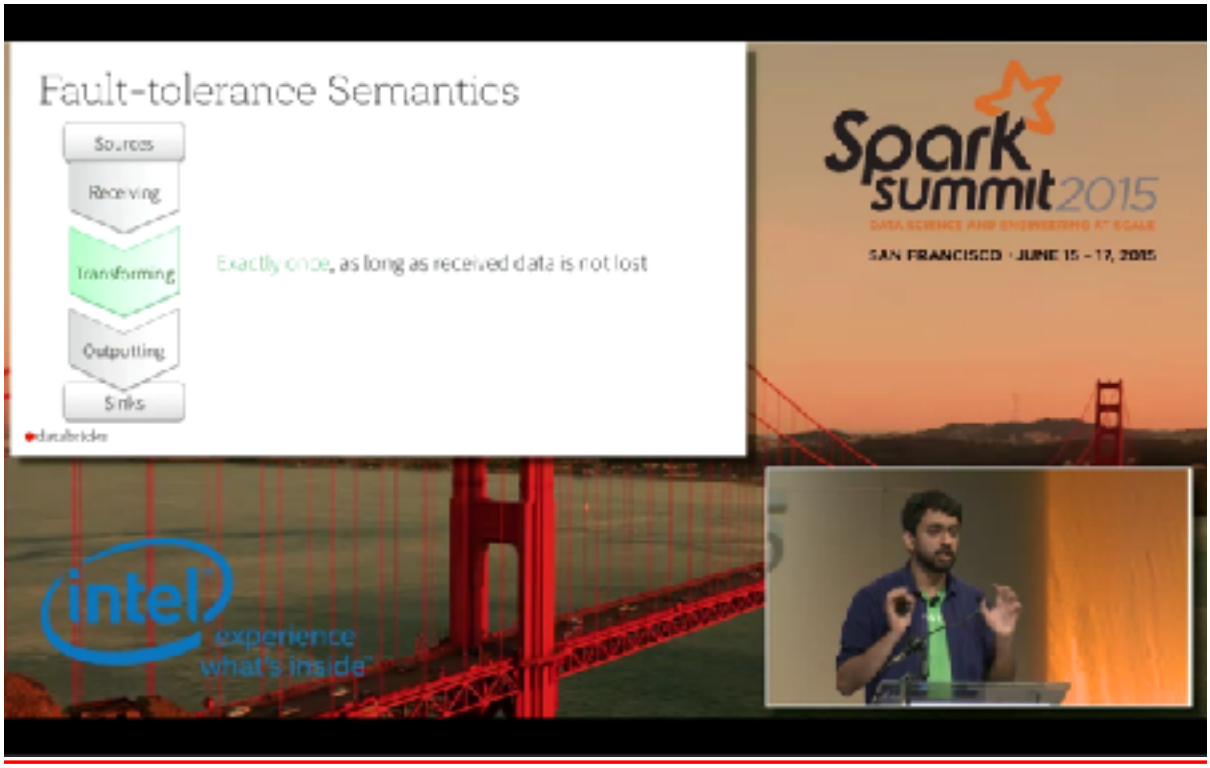

Recovering data with Write Ahead Logs
Write Ahead Log (WAL): Synchronously save received data to fault tolerant storage
Executor
Receiver
Blocks
Data stream
Active Driver
Blocks saved to HDFS
Executor
databricks
Spark summit 2015
SAN FRANCISCO · JUNE 15 – 17, 2015
intel
experience what's inside

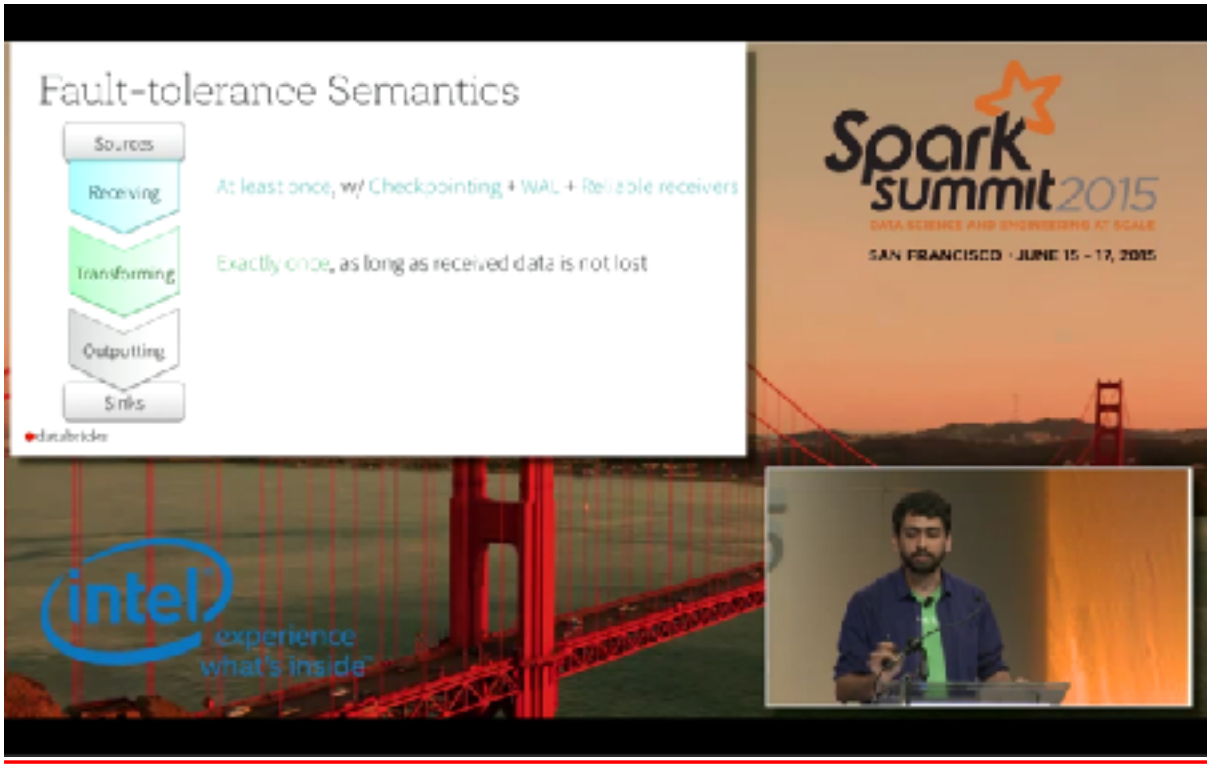

Recovering data with Write Ahead Logs
Write Ahead Log (WAL): Synchronously save received data to fault tolerant storage
New Ex.
Receiver
Blocks
Failed Driver
Restarted Driver
Blocks recovered from write Ahead Log
New Executor
databricks
Spark summit 2015
SAN FRANCISCO · JUNE 15 – 17, 2015
intel
experience what's inside

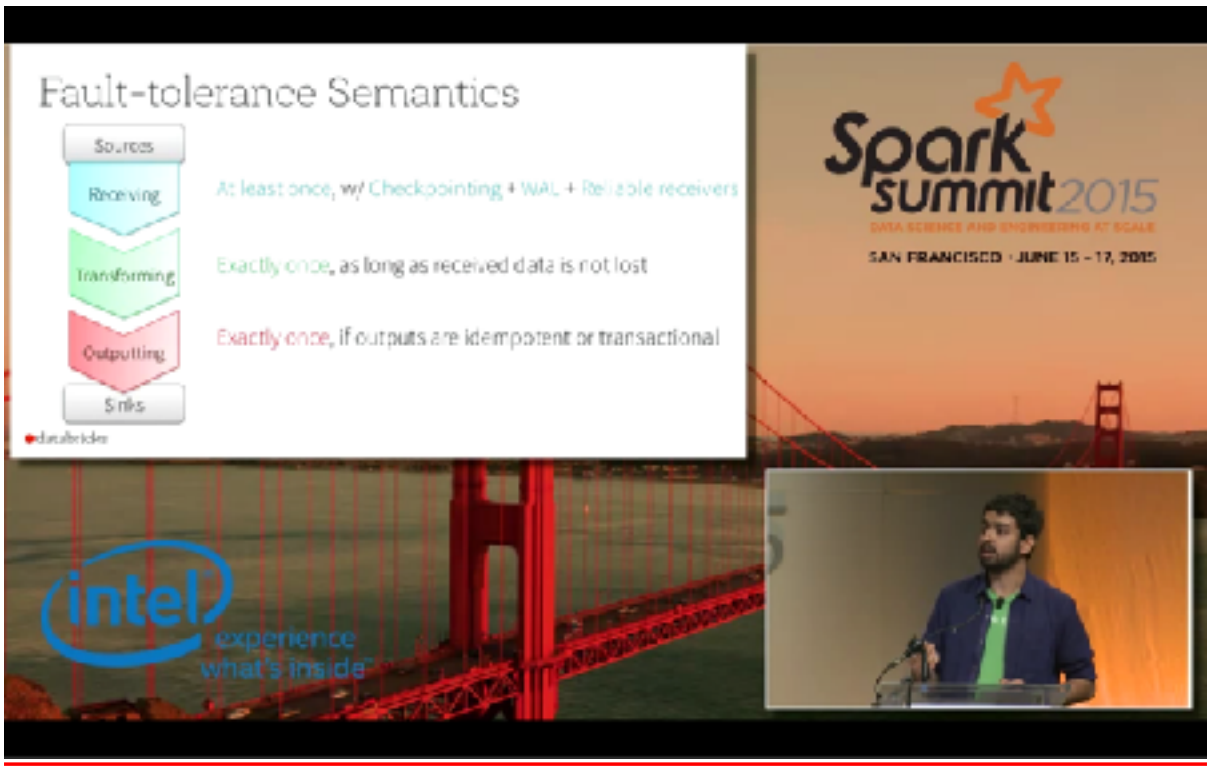

Recovering data with Write Ahead Logs
1. Enable checkpointing, logs written in checkpoint directory
2. Enable WAL in SparkConf configuration
sparkConf.set("spark.streaming.receiver.writeAheadLog.enable", "true")
3. Receiver should also be reliable
Acknowledge source only after data saved to WAL
Unacked data will be replayed from source by restarted receiver
4. Disable in-memory replication (already replicated by HDFS)
Use StorageLevel.MEMORY_AND_DISK_SER for input DStreams
databricks
Spark summit 2015
SAN FRANCISCO · JUNE 15 – 17, 2015
intel
experience
what's inside


RDD Checkpointing
Stateful stream processing can lead to long RDD lineages
Long lineage = bad for fault tolerance, too much recomputation
RDD checkpointing saves RDD data to the fault-tolerant
storage to limit lineage and recomputation
More:
databricks
Spark summit 2015
SAN FRANCISCO · JUNE 15 – 17, 2015
intel
experience
what's inside

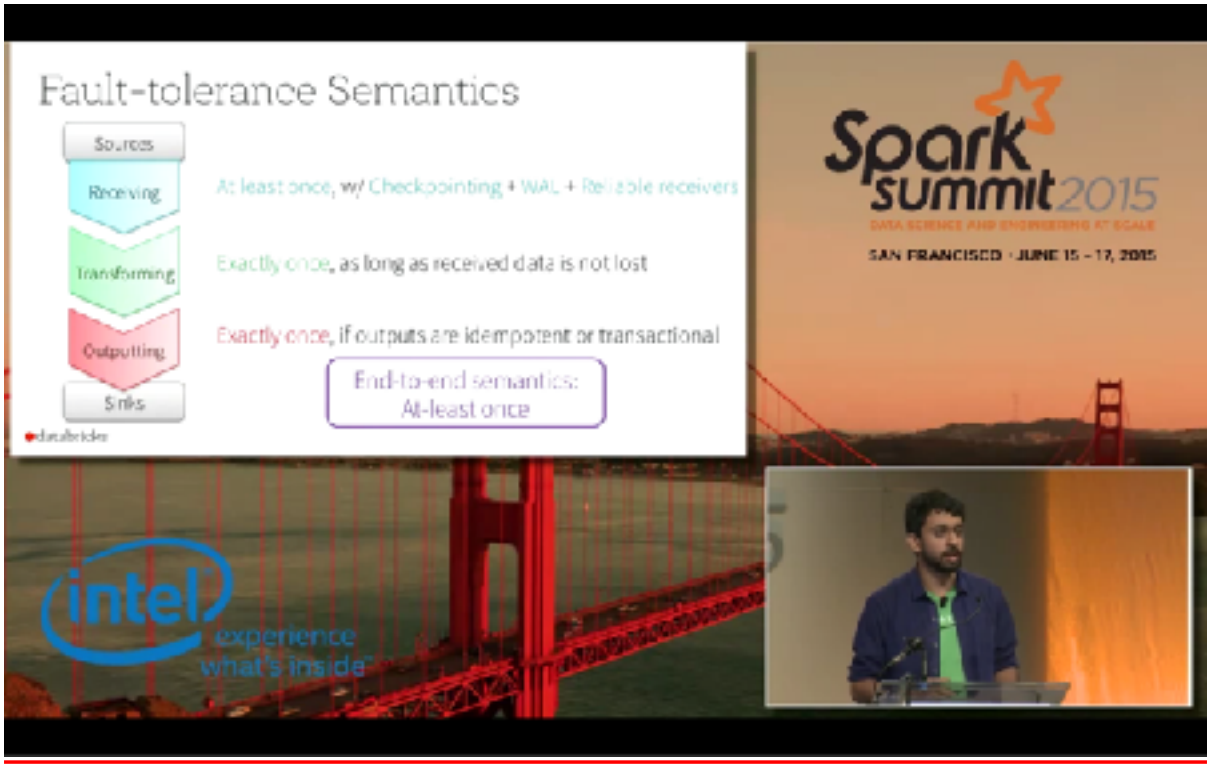

Fault-tolerance Semantics
Sources
Receiving
Transforming
Outputting
Sinks
databricks
Zero data loss = every stage processes each event at least once despite any failure
Spark summit 2015
SAN FRANCISCO · JUNE 15 – 17, 2015
intel
experience what's inside

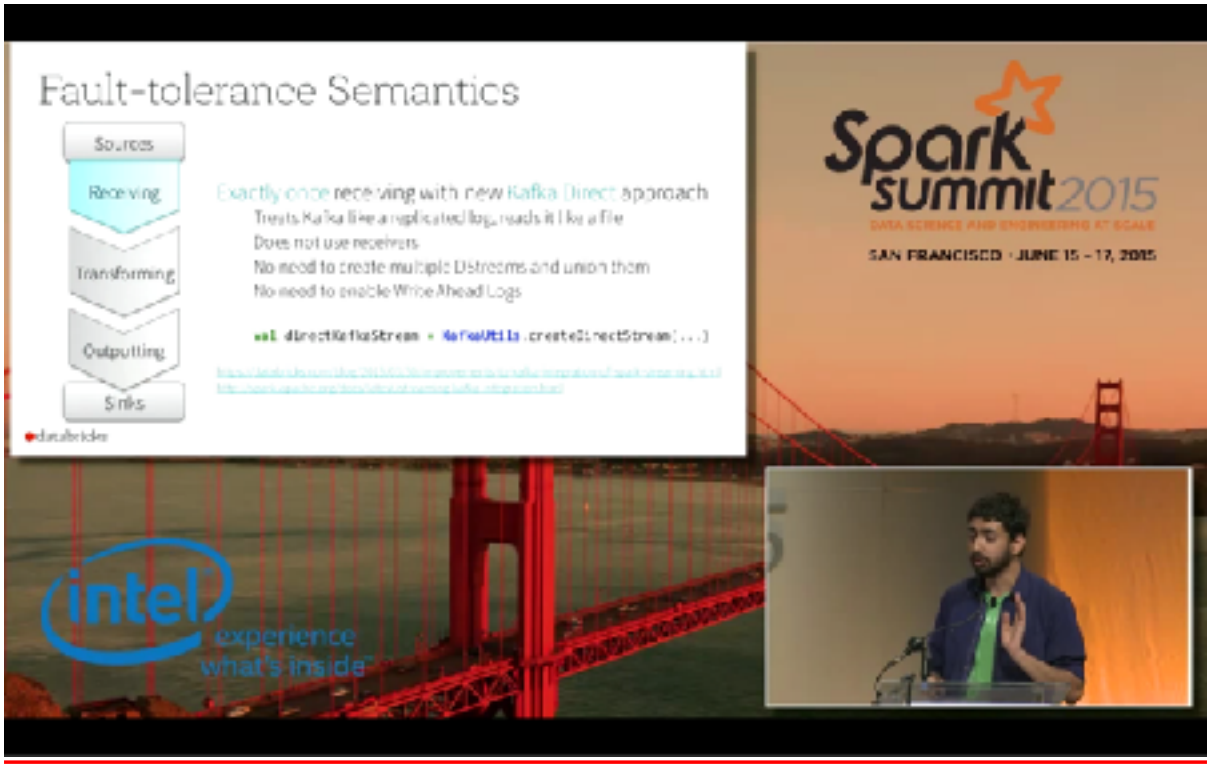

Fault-tolerance Semantics
Sources
Receiving
Transforming
Outputting
Sinks
databricks
Exactly once, as long as received data is not lost
Spark summit 2015
SAN FRANCISCO · JUNE 15 – 17, 2015
intel
experience what's inside

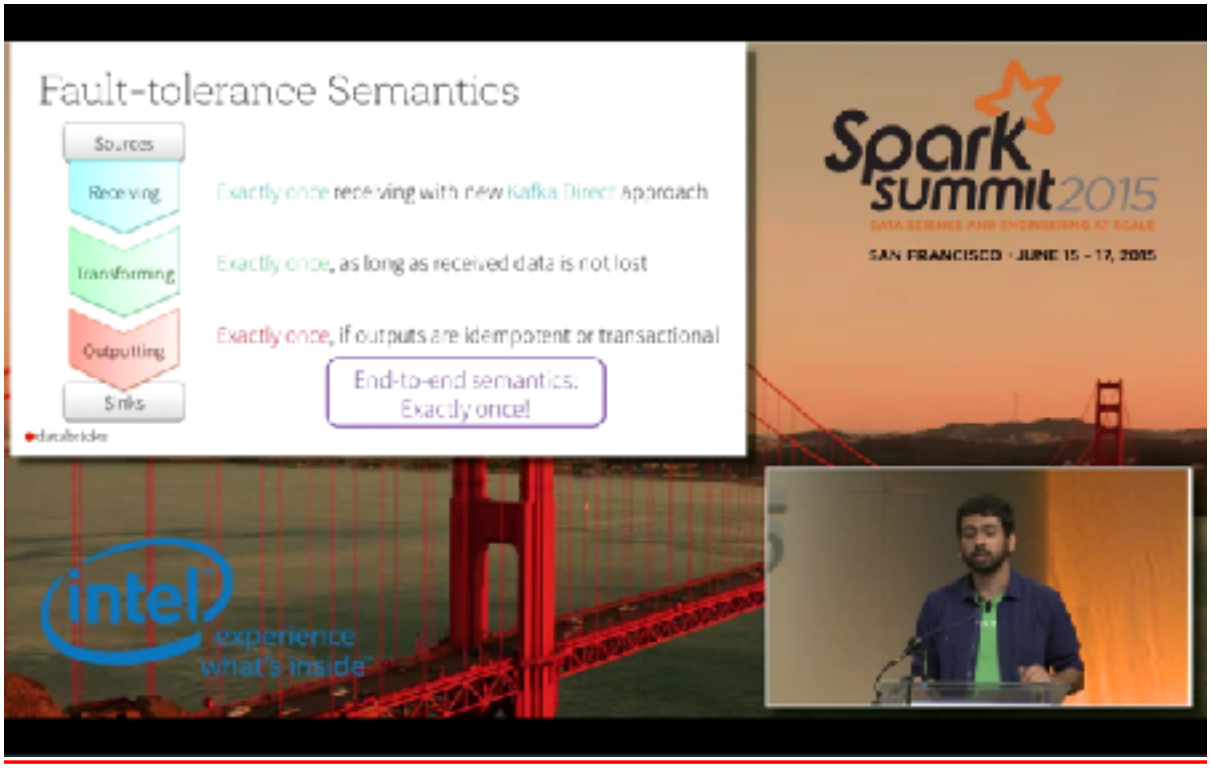


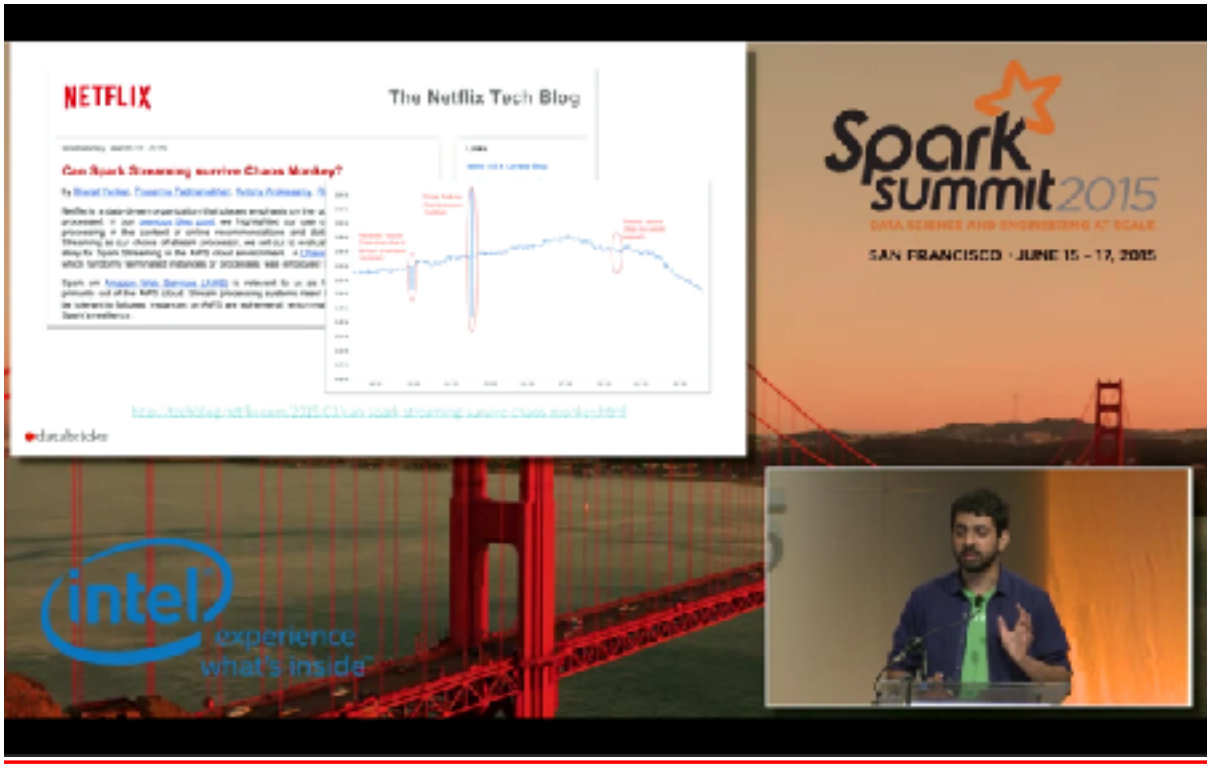


Note:
Idempotent: not changed/ unchanged


Fault-tolerance Semantics
Sources
Receiving
Transforming
Outputting
Sinks
At-least once, w/ Checkpointing + WAL + Reliable receivers
Exactly once, as long as received data is not lost
Exactly once, if outputs are idempotent or transactional
End-to-end semantics:
At-least once
databricks
Spark summit 2015
SAN FRANCISCO · JUNE 15 – 17, 2015
intel
experience
what's inside


Fault-tolerance Semantics
Sources
Receiving
Transforming
Outputting
Sinks
Exactly once receiving with new Kafka Direct approach
Treats Kafka like a replicated log, reads it like a file
Does not use receivers
No need to create multiple DStreams and union them
No need to enable Write Ahead Logs
databricks
Spark summit 2015
SAN FRANCISCO · JUNE 15 – 17, 2015
intel
experience
what's inside


Fault-tolerance Semantics
Sources
Receiving
Transforming
Outputting
Sinks
Exactly once receiving with new Kafka Direct Approach
Exactly once, as long as received data is not lost
Exactly once, if outputs are idempotent or transactional
End-to-end semantics:
Exactly once!
databricks
Spark summit 2015
SAN FRANCISCO · JUNE 15 – 17, 2015
intel
experience
what's inside


NETFLIX
The Netflix Tech Blog
Can Spark Streaming survive Chaos Monkey?
databricks
Spark summit 2015
SAN FRANCISCO · JUNE 15 – 17, 2015
intel
experience
what's inside

Throughput: the amount of data passing through process(data passing/ data travelling)

Scaling the Receivers

Sources must be configured with parallel data streams

#partitions in Kafka topics, #shards in Kinesis streams, ...

databricks

Scaling the Receivers

Sources must be configured with parallel data streams

#partitions in Kafka topics, #shards in Kinesis streams, ...

Streaming app should have multiple receivers that receive the data streams in parallel

Multiple input DStreams, each running a receiver

Can be unioned together to create one DStream

```
val kafkaStream1 = KafkaUtils.createStream(...)
val kafkaStream2 = KafkaUtils.createStream(...)
val unionedStream = kafkaStream1.union(kafkaStream2)
```

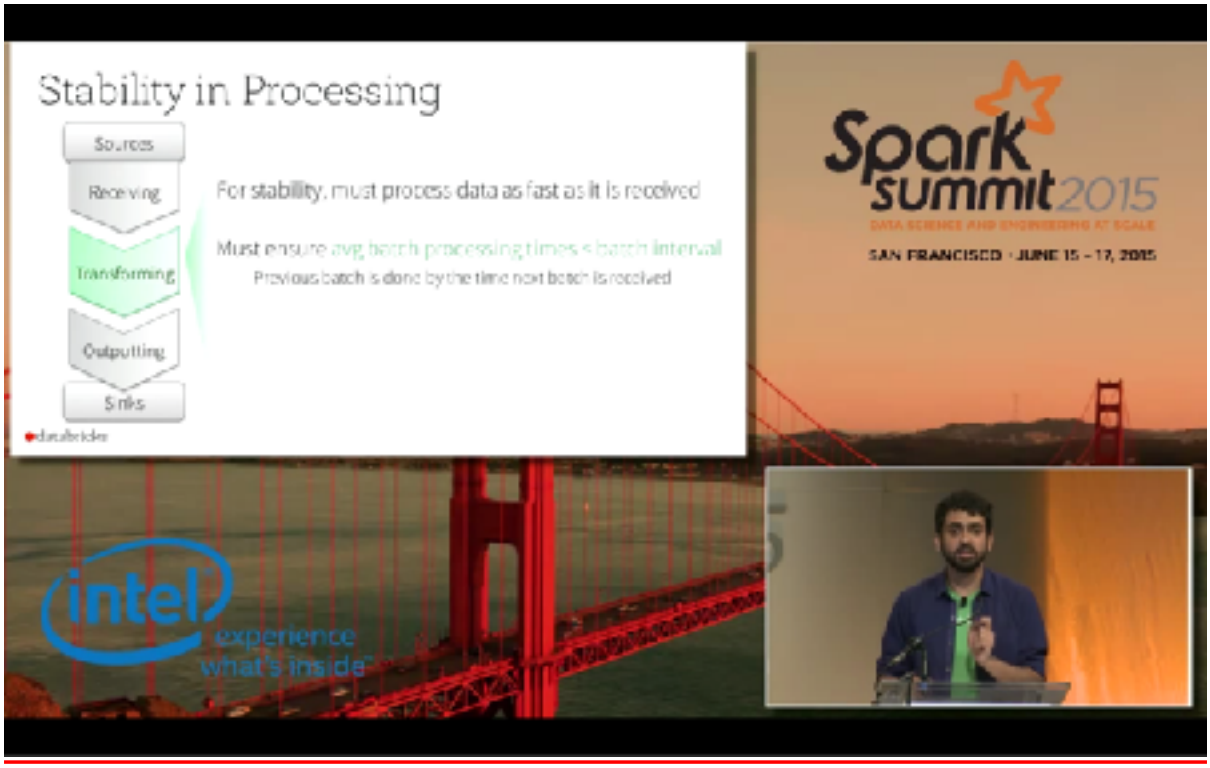
databricks

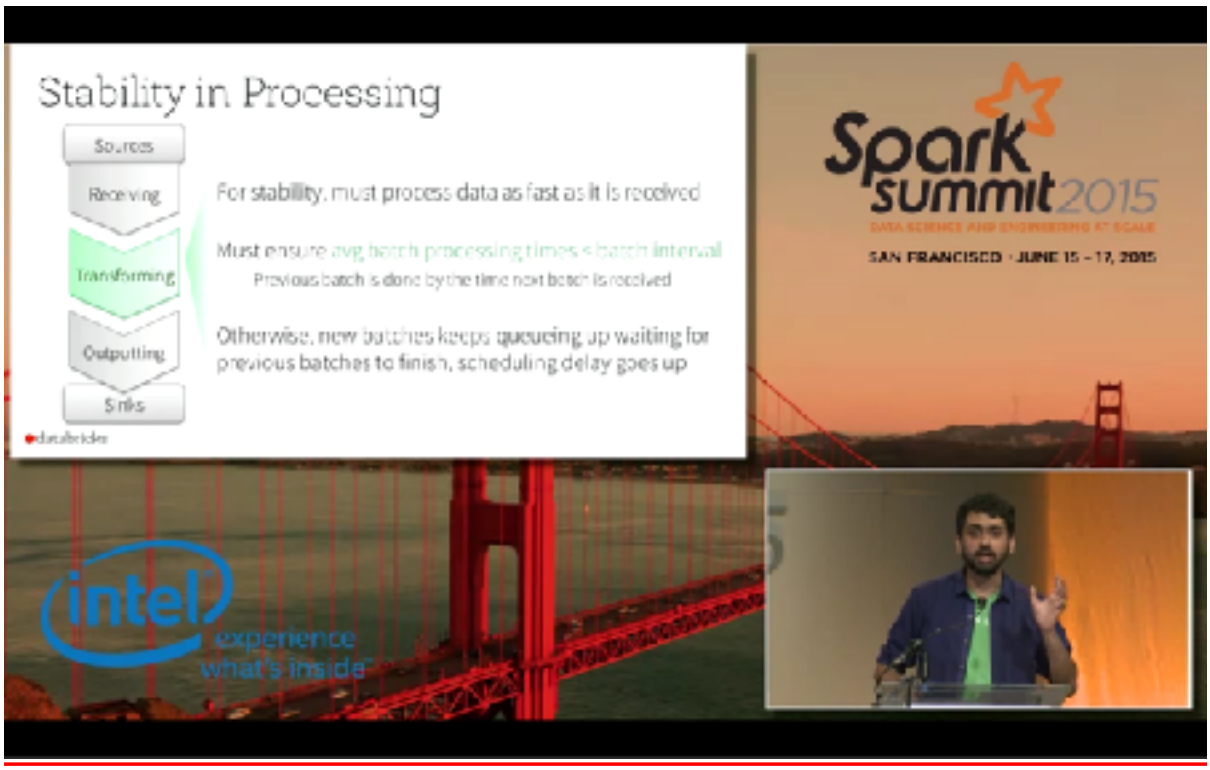
Scaling the Receivers

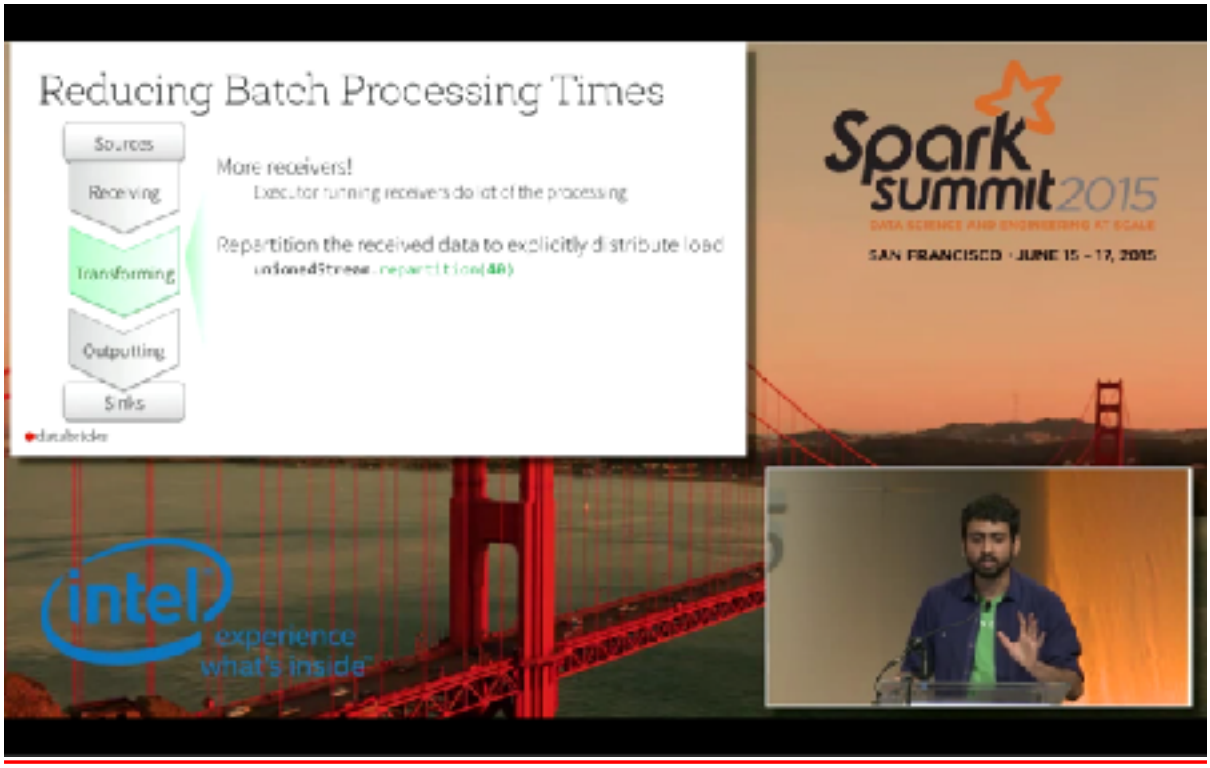
Sufficient number of executors to run all the receivers

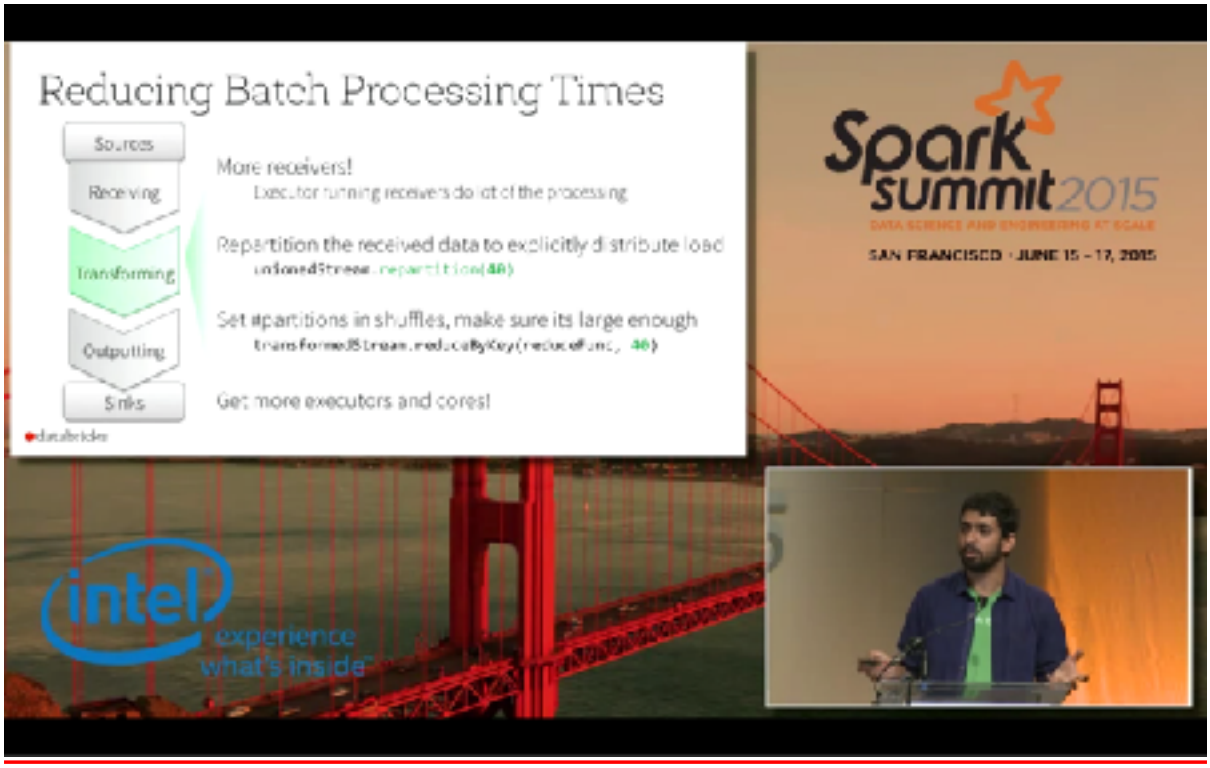
- Absolute necessity: #cores > #receivers
- Good rule of thumb: #executors > #receivers, so that no more than 1 receiver per executor, and network is not shared between receivers

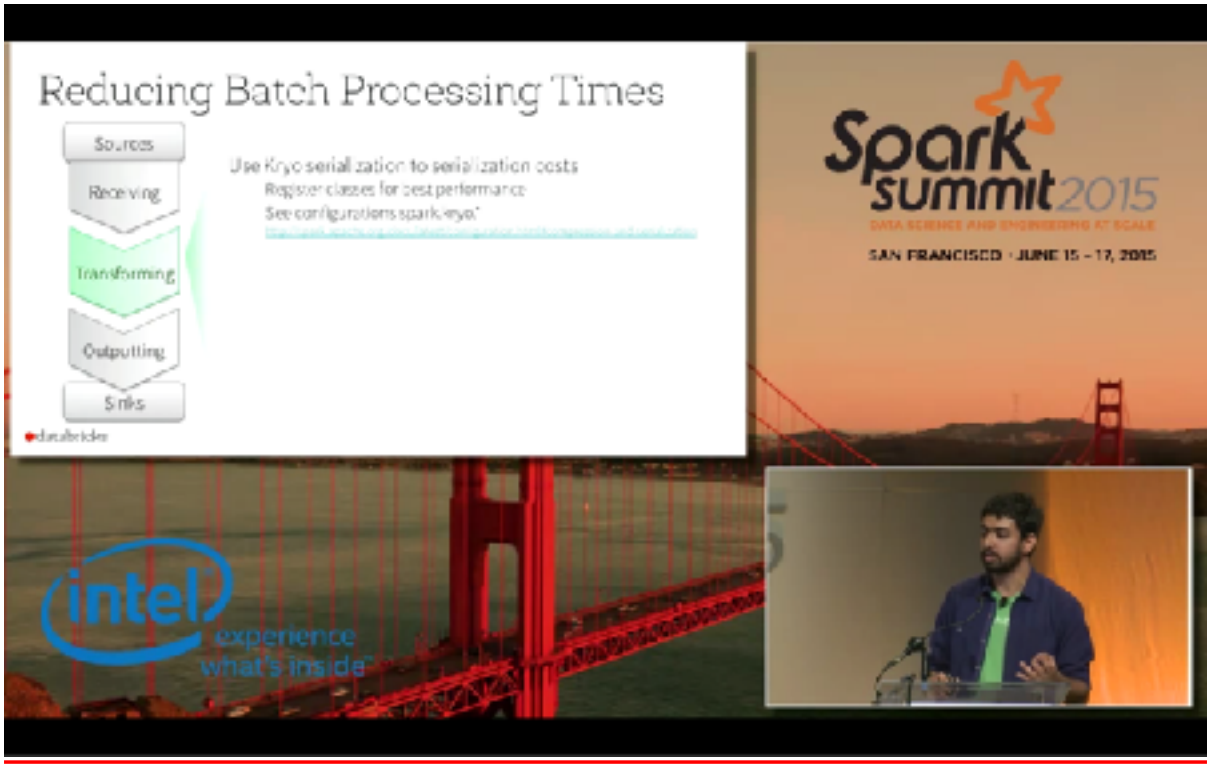
databricks

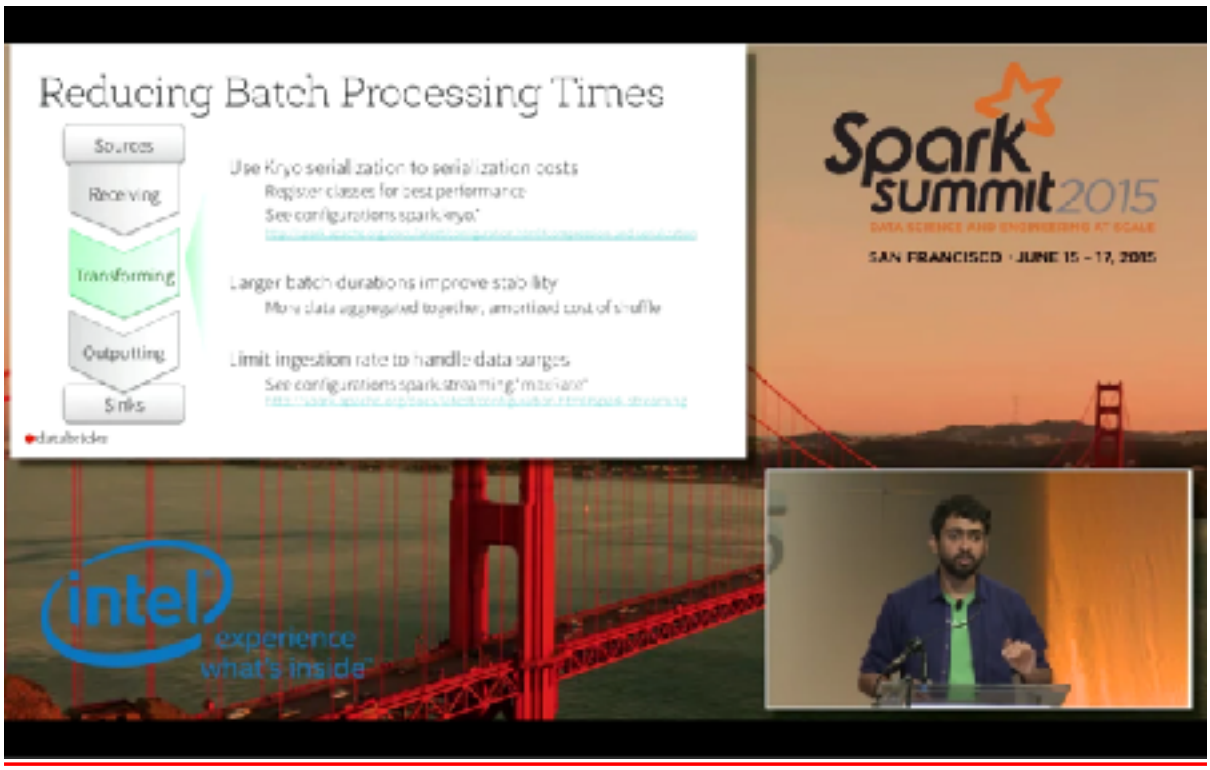

Stability in Processing
Sources
Receiving
Transforming
Outputting
Sinks
For stability, must process data as fast as it is received
Must ensure avg batch processing times < batch interval
Previous batch is done by the time next batch is received
databricks
Spark summit 2015
SAN FRANCISCO · JUNE 15 – 17, 2015
intel
experience
what's inside


Stability in Processing
Sources
Receiving
Transforming
Outputting
Sinks
For stability, must process data as fast as it is received
Must ensure avg batch processing times < batch interval
Previous batch is done by the time next batch is received
Otherwise, new batches keeps queueing up waiting for previous batches to finish, scheduling delay goes up
databricks
Spark summit 2015
SAN FRANCISCO · JUNE 15 – 17, 2015
intel
experience
what's inside


Reducing Batch Processing Times
Sources
Receiving
Transforming
Outputting
Sinks
More receivers!
Executor running receivers do lot of the processing
Repartition the received data to explicitly distribute load
unionedStream.repartition(48)
databricks
Spark summit 2015
SAN FRANCISCO · JUNE 15 – 17, 2015
intel
experience
what's inside

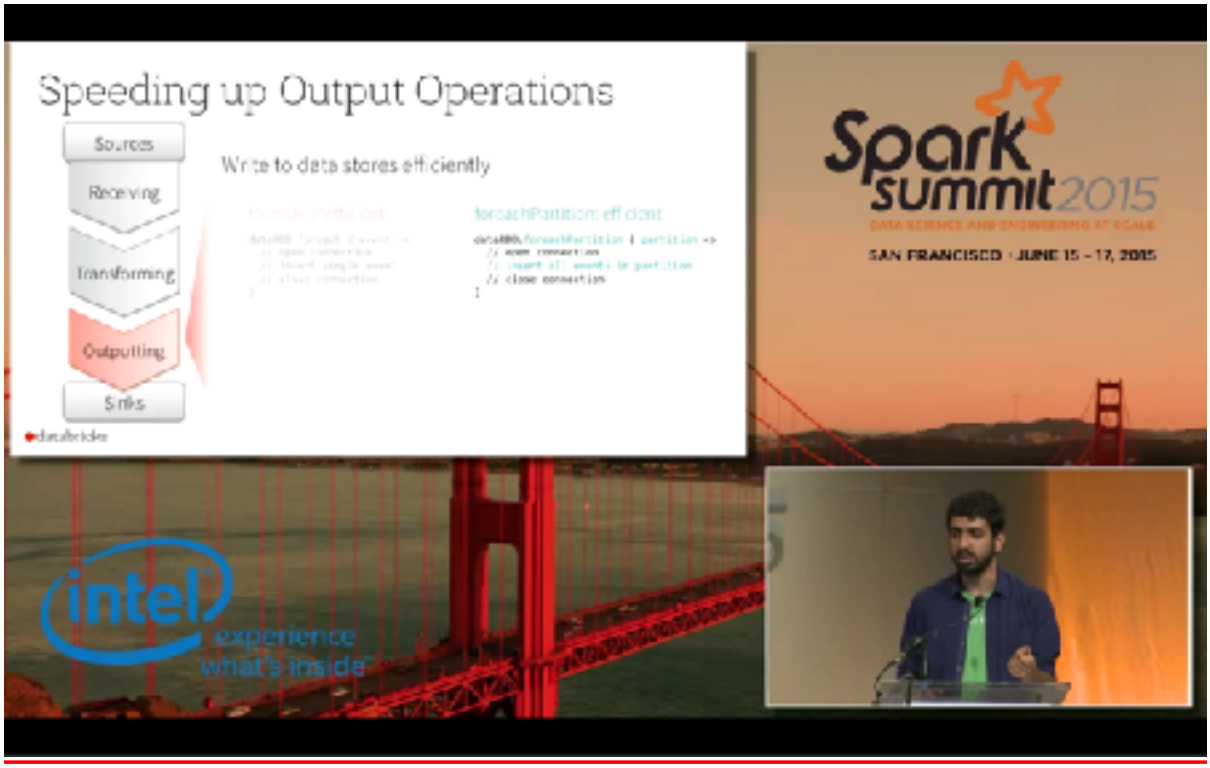

Reducing Batch Processing Times
Sources
Receiving
Transforming
Outputting
Sinks
More receivers!
Executor running receivers do lot of the processing
Repartition the received data to explicitly distribute load
unionedStream.repartition(48)
Set #partitions in shuffles, make sure its large enough
transformedStream.reduceByKey(reduceFunc, 48)
Get more executors and cores!
databricks
Spark summit 2015
SAN FRANCISCO · JUNE 15 – 17, 2015
intel
experience
what's inside

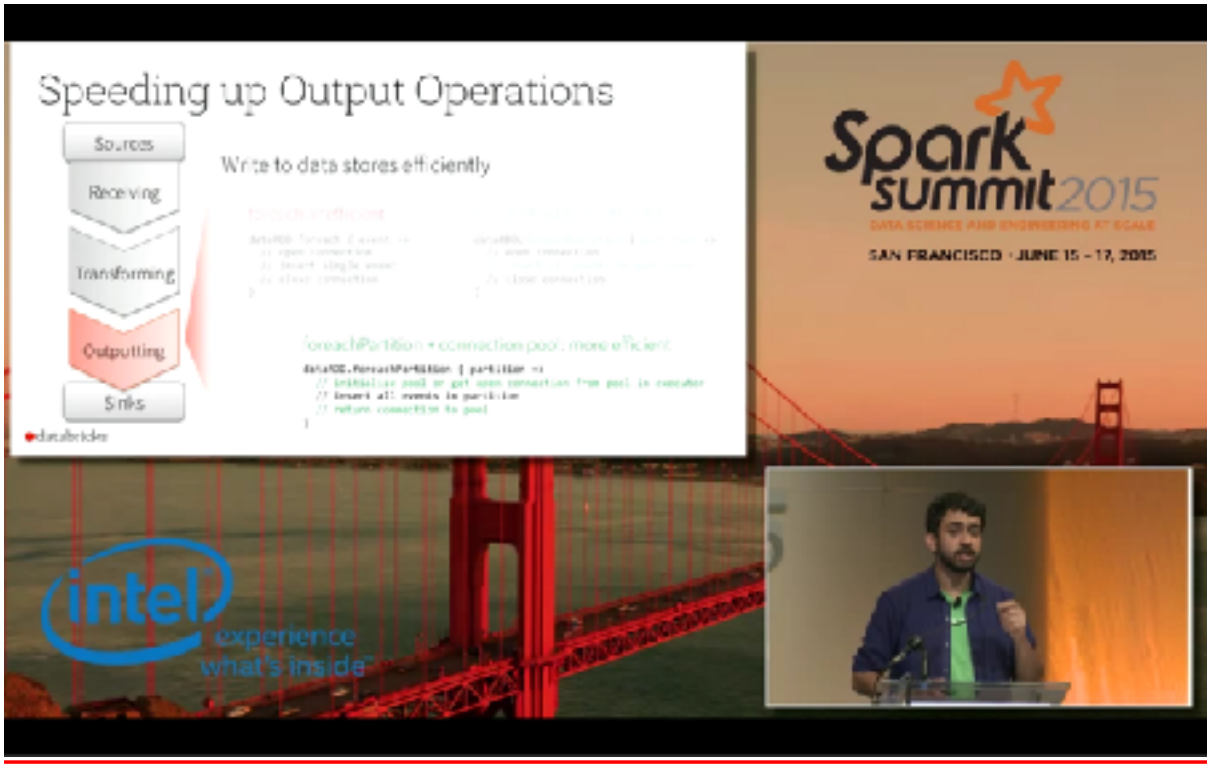


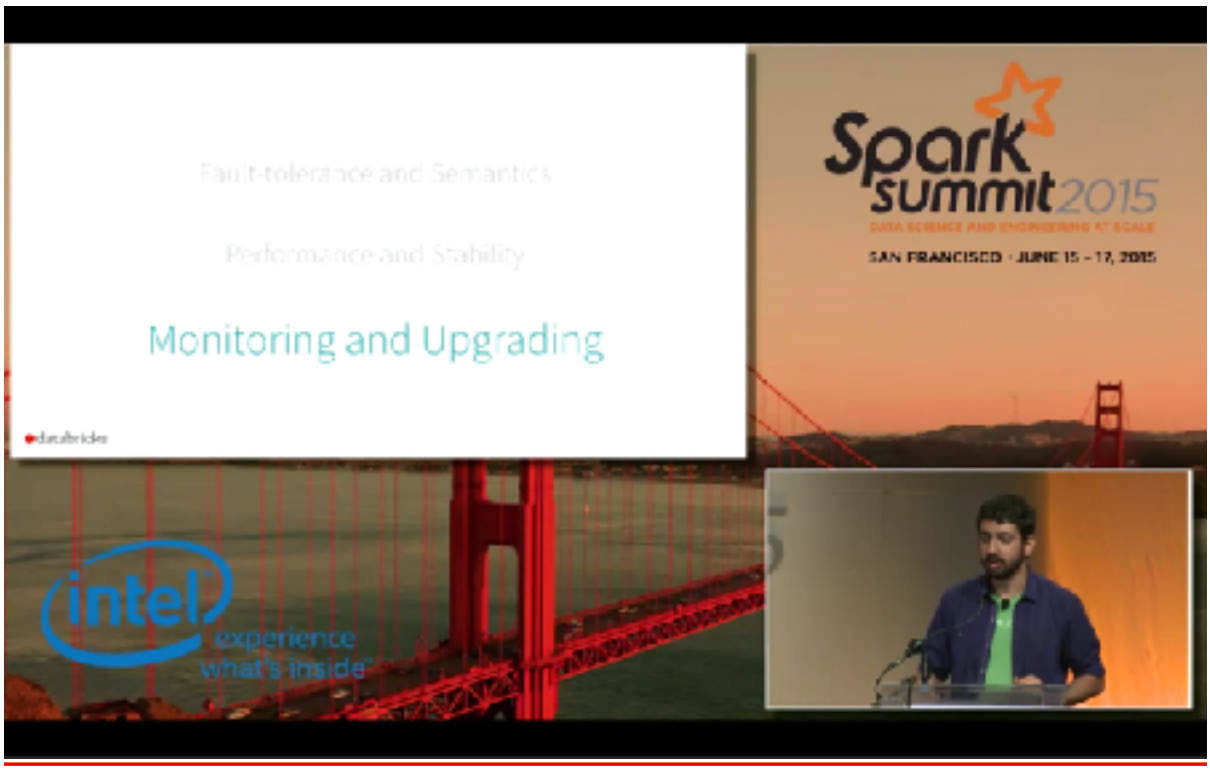


Amortized: reduce the cost (I.e gradually write off the initial cost of (an asset) over a period)
Data surge: data overflow/ a sudden powerful forward or upward movement


Speeding up Output Operations
Sources
Receiving
Transforming
Outputting
Sinks
Write to data stores efficiently
foreach: inefficient
Spark summit 2015
SAN FRANCISCO · JUNE 15 – 17, 2015
intel
experience what's inside


Speeding up Output Operations
Sources
Receiving
Transforming
Outputting
Sinks
Write to data stores efficiently
foreachPartition: efficient
Spark summit 2015
SAN FRANCISCO · JUNE 15 – 17, 2015
intel
experience what's inside


Speeding up Output Operations
Sources
Receiving
Transforming
Outputting
Sinks
Write to data stores efficiently
foreachPartition + connection pool: more efficient
databricks
Spark summit 2015
SAN FRANCISCO · JUNE 15 – 17, 2015
intel
experience what's inside


Fault-tolerance and Semantics
Performance and Stability
Monitoring and Upgrading
databricks
Spark summit 2015
SAN FRANCISCO · JUNE 15 – 17, 2015
intel
experience what's inside

Streaming in Spark Web UI

Stats over last 1000 batches

New in Spark 1.4

For stability

Scheduling delay should be approx 0

Processing Time approx <= batch interval

databricks

Streaming in Spark Web UI

Details of individual batches

Active batches (1)

Batch Time	Input Size	Scheduling Delay	Processing Time	Status
2015/06/08 11:12:45	3 events	0 ms	-	processing

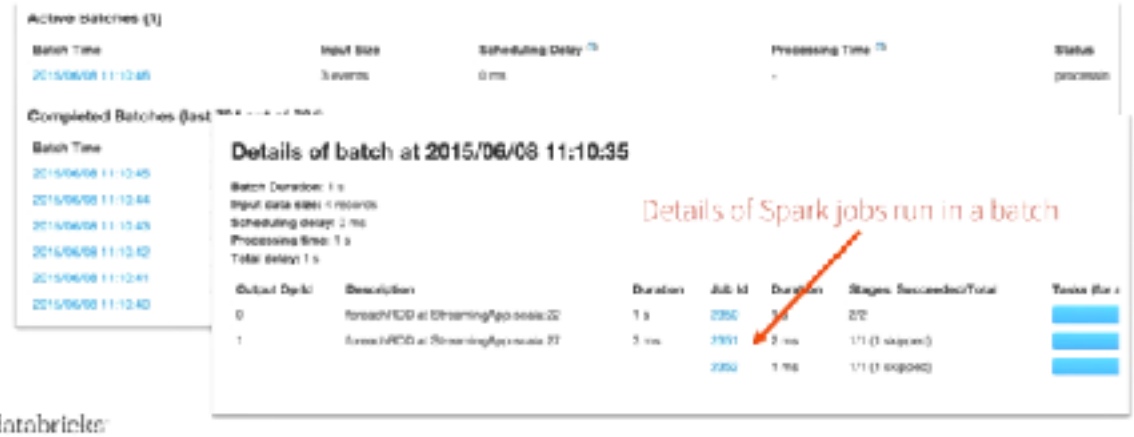
Completed Batches (last 794 out of 794)

Batch Time	Input Size	Scheduling Delay	Processing Time	Total Delay
2015/06/08 11:12:45	2 events	0 ms	0.2 s	0.2 s
2015/06/08 11:12:44	1 events	0 ms	0.4 s	0.4 s
2015/06/08 11:12:43	2 events	0 ms	0.7 s	0.7 s
2015/06/08 11:12:42	2 events	0 ms	0.3 s	0.3 s
2015/06/08 11:12:41	2 events	0 ms	0.9 s	0.9 s
2015/06/08 11:12:40	1 events	0 ms	0.4 s	0.4 s

databricks

Streaming in Spark Web UI

Details of individual batches



databricks

Operational Monitoring

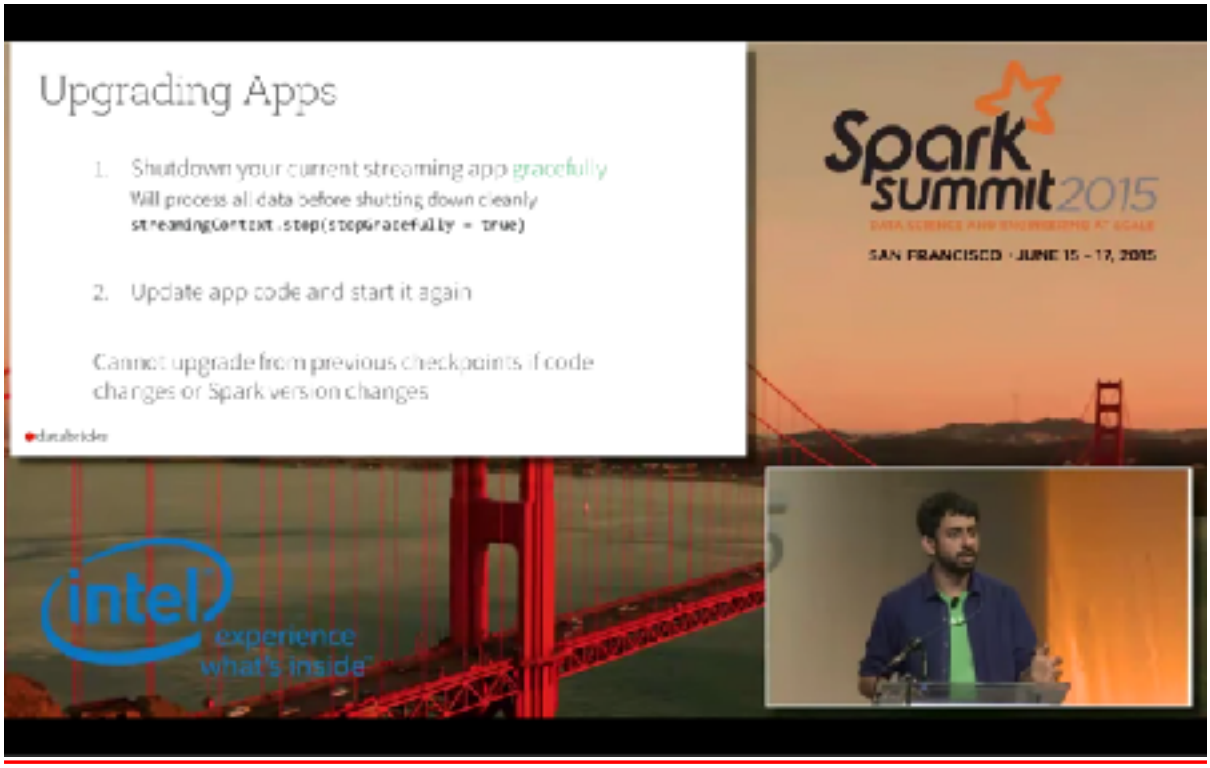
Streaming app stats published through Codahale metrics

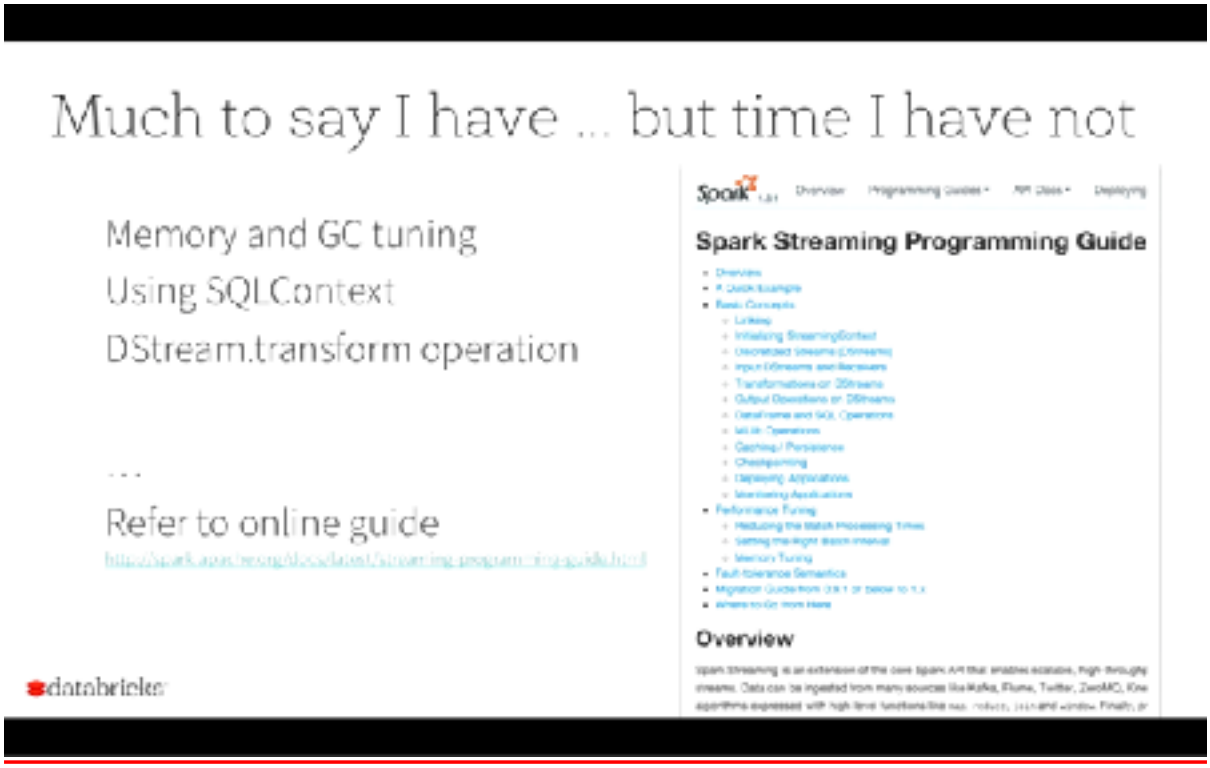
- Ganglia sink, Graphite sink, custom Codahale metrics sinks
- Can see long term trends, across hours and days

Configure the metrics using `$SPARK_HOME/conf/metrics.properties`

- Need to compile Spark with Ganglia LGPL profile for Ganglia support (see http://spark.apache.org/docs/latest/monitoring.html#metrics)

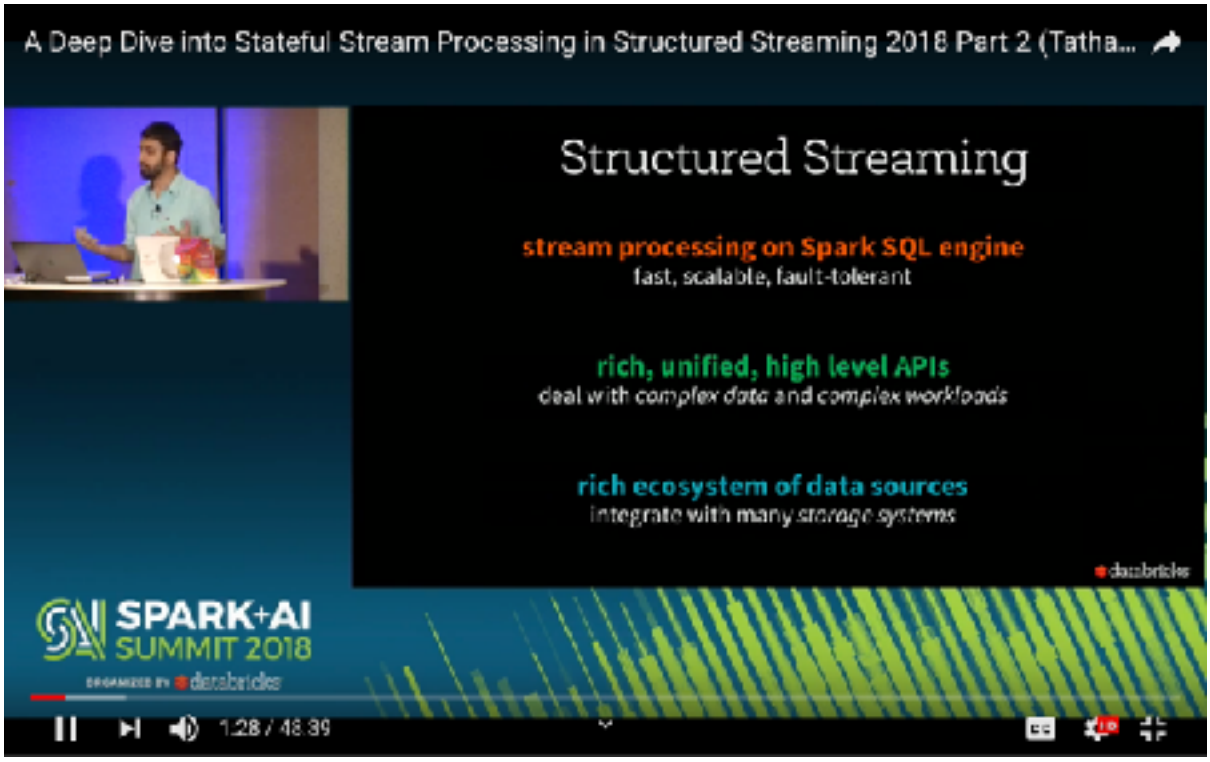

Programmatic Monitoring
StreamingListener – Developer interface to get internal events
onBatchSubmitted, onBatchStarted, onBatchCompleted,
onReceiverStarted, onReceiverStopped, onReceiverError
Take a look at StreamingJobProgressListener (private class) for inspiration
databricks
Spark summit 2015
SAN FRANCISCO · JUNE 15 – 17, 2015
intel
experience
what's inside

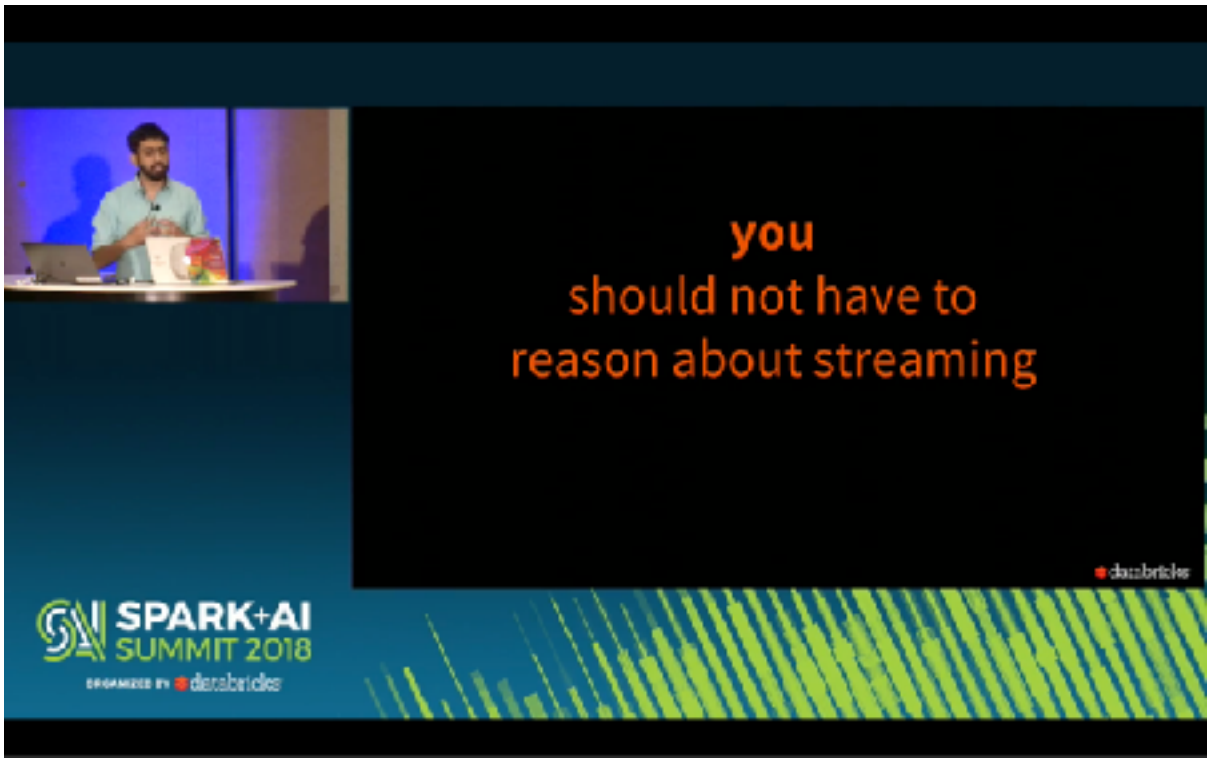

Upgrading Apps
1. Shutdown your current streaming app gracefully
Will process all data before shutting down cleanly
streamingContext.stop(stopGracefully = true)
2. Update app code and start it again
Cannot upgrade from previous checkpoints if code changes or Spark version changes
databricks
Spark summit 2015
SAN FRANCISCO · JUNE 15 – 17, 2015
intel
experience
what's inside

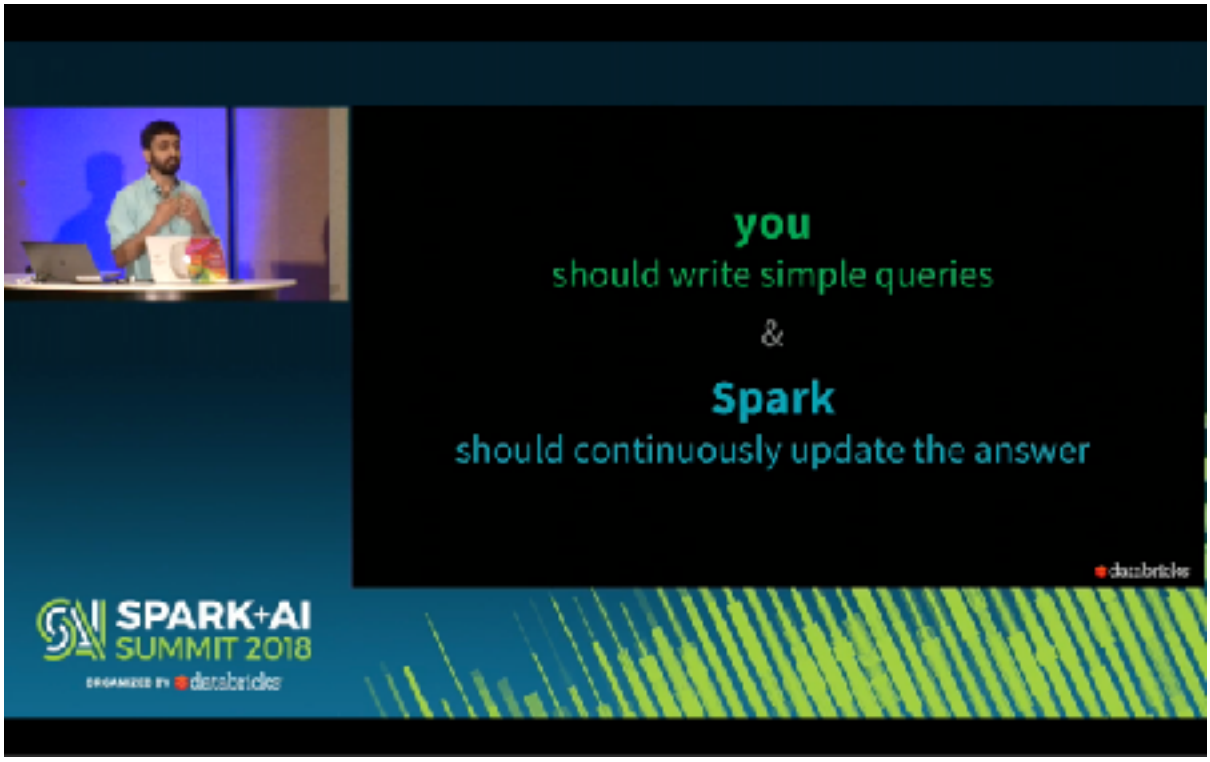


Spark - stateful stream processing on SQL Engine

https://www.youtube.com/watch?v=aV0L7R8NH7Y&t=32s


A Deep Dive into Stateful Stream Processing in Structured Streaming 2018 Part 2 (Tatha...
Structured Streaming
stream processing on Spark SQL engine
fast, scalable, fault-tolerant
rich, unified, high level APIs
deal with complex data and complex workloads
rich ecosystem of data sources
integrate with many storage systems
databricks
SPARK+AI SUMMIT 2018
1:28 / 48:39


you
should not have to
reason about streaming
databricks
SPARK+AI SUMMIT 2018


you
should write simple queries
&
Spark
should continuously update the answer
SPARK+AI SUMMIT 2018
databricks


Treat Streams as Unbounded Tables
data stream
unbounded input table
new data in the data stream
=
new rows appended to a unbounded table
SPARK+AI SUMMIT 2018
databricks

Anatomy of a Streaming Query

Example

Read JSON data from Kafka

Parse nested JSON

Store in structured Parquet table

Get end-to-end failure guarantees

databricks

SPARK+AI SUMMIT 2018

ORGANIZED BY databricks

Anatomy of a Streaming Query

```
spark.readStream.format("kafka")
  .option("kafka.bootstrap.servers",...)
  .option("subscribe", "topic")
  .load()
```

Source

Specify where to read data from

Built-in support for Files / Kafka / Kinesis*

Can include multiple sources of different types using `join()` / `union()`

*Available only on Databricks Runtime

databricks

SPARK+AI SUMMIT 2018

ORGANIZED BY databricks

Anatomy of a Streaming Query

```
spark.readStream.format("kafka")
  .option("kafka.bootstrap.servers",...)
  .option("subscribe", "topic")
  .load()
  .selectExpr("cast (value as string) as json")
  .select(from_json("json", schema).as("data"))
```

Transformations

Cast bytes from Kafka records to a string, parse it as a json, and generate nested columns

100s of built-in, optimized SQL functions like from_json

user-defined functions, lambdas, function literals with map, flatMap...

Anatomy of a Streaming Query

```
spark.readStream.format("kafka")
  .option("kafka.bootstrap.servers",...)
  .option("subscribe", "topic")
  .load()
  .selectExpr("cast (value as string) as json")
  .select(from_json("json", schema).as("data"))
  .writeStream
  .format("parquet")
  .option("path", "/parquetTable/")
```

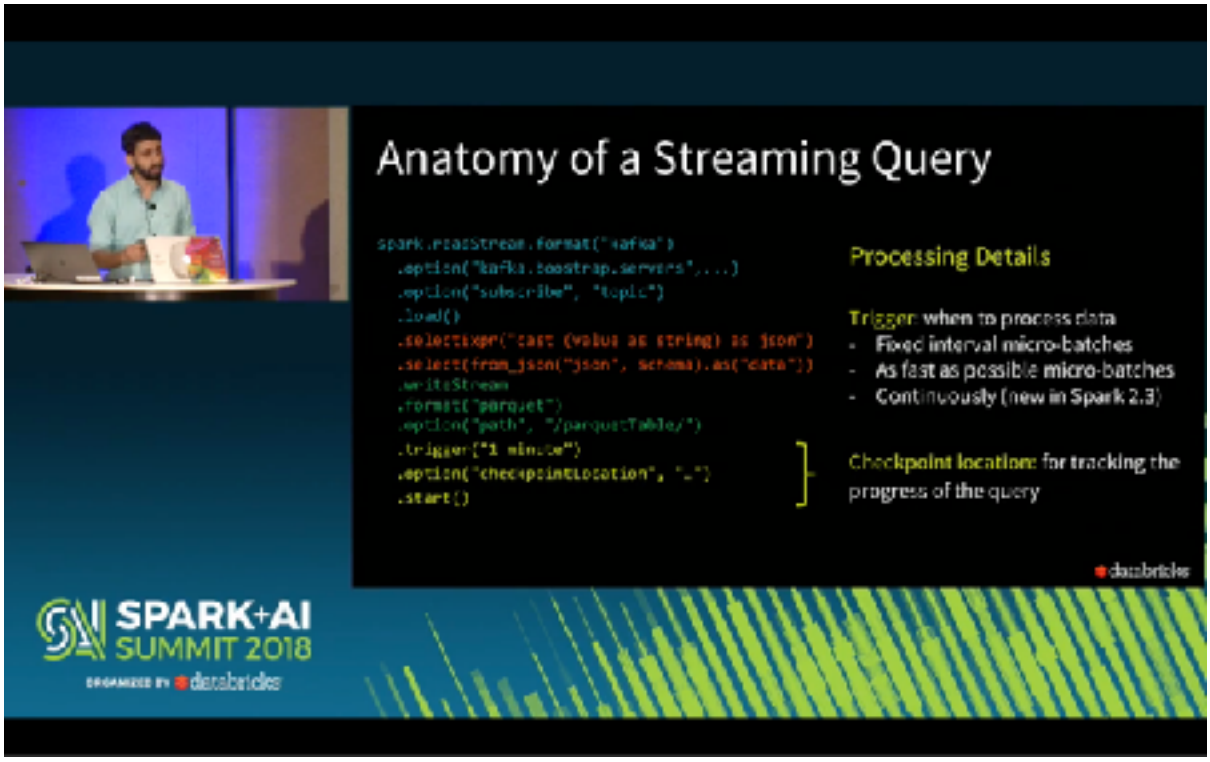
Sink

Write transformed output to external storage systems

Built-in support for Files / Kafka

Use foreach to execute arbitrary code with the output data

Some sinks are transactional and exactly once (e.g. files)



Triggers:

The trigger settings of a streaming query defines the timing of streaming data processing, whether the query is going to executed as micro-batch query with a fixed batch interval or as a continuous processing query.

Trigger Types:

1. **Not Specified- default one:**

   If no trigger setting is explicitly specified, then by default, the query will be executed in micro-batch mode, where micro-batches will be generated as soon as the previous micro-batch has completed processing.

```
import org.apache.spark.sql.streaming.Trigger



// Default trigger (runs micro-batch as soon as it can)

df.writeStream

  .format("console")

  .start()
```

2. **Fixed Interval(micro-baches):**

The query will be executed with micro-batches mode, where micro-batches will be kicked off at the user-specified intervals.

```
// ProcessingTime trigger with two-seconds micro-batch interval

df.writeStream

  .format("console")
```

```
  .trigger(Trigger.ProcessingTime("2 seconds"))
  .start()
```

- If the previous micro-batch completes within the interval, then the engine will wait until the interval is over before kicking off the next micro-batch.
- If the previous micro-batch takes longer than the interval to complete (i.e. if an interval boundary is missed), then the next micro-batch will start as soon as the previous one completes (i.e., it will not wait for the next interval boundary).
- If no new data is available, then no micro-batch will be kicked off.

3. **One-time(micro-bacth)**
   The query will execute *only one* micro-batch to process all the available data and then stop on its own. This is useful in scenarios you want to periodically spin up a cluster, process everything that is available since the last period, and then shutdown the cluster. In some case, this may lead to significant cost savings.

```
// One-time trigger
df.writeStream
  .format("console")
  .trigger(Trigger.Once())
  .start()
```

4. **Continuous with fixed checkpoint interval**
   The query will be executed in the new low-latency, continuous processing mode. Read more about this in the Continuous Processing section below.

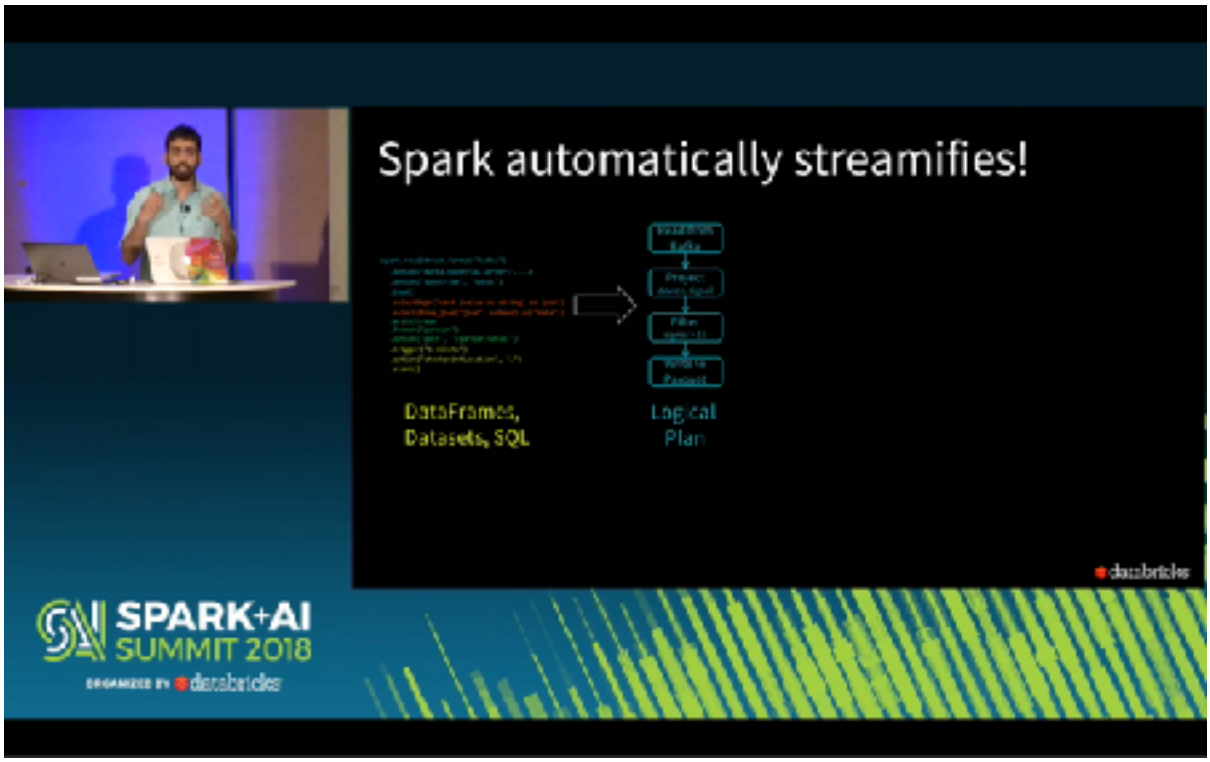
```
// Continuous trigger with one-second checkpointing interval
    df.writeStream
      .format("console")
      .trigger(Trigger.Continuous("1 second"))
      .start()
```

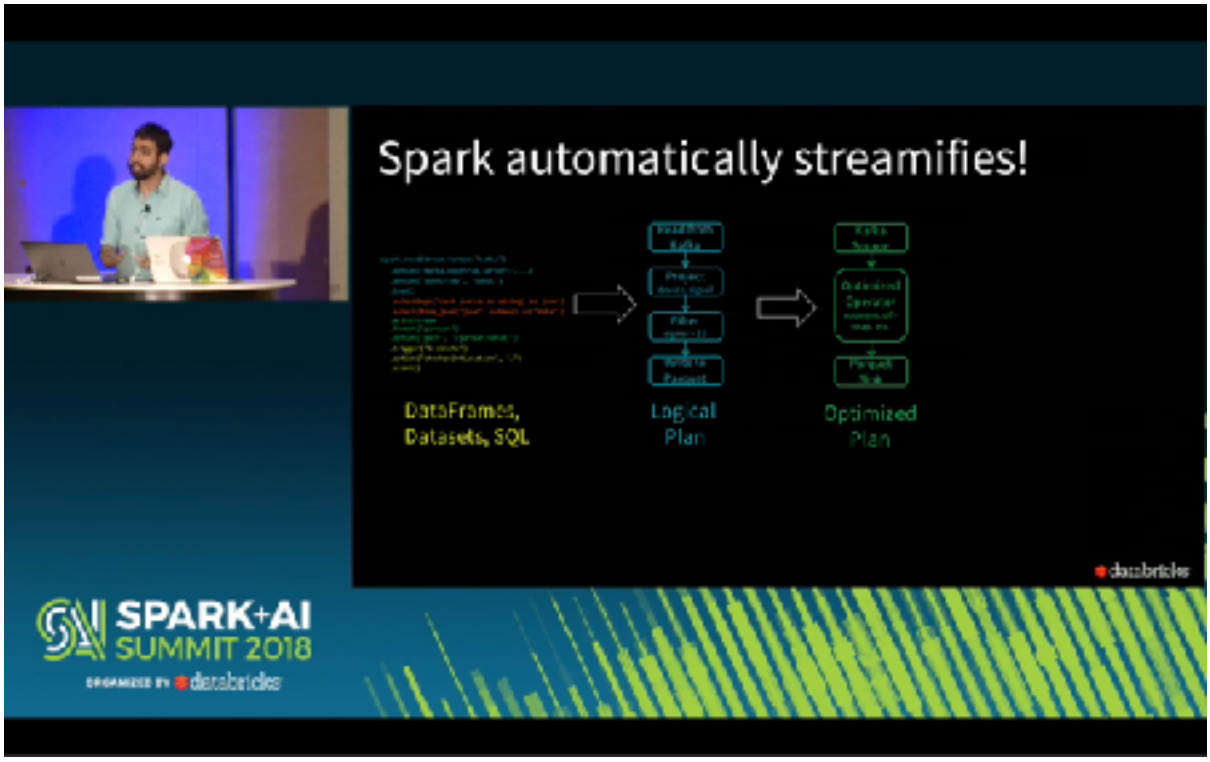
Recovering from Failures with Checkpointing

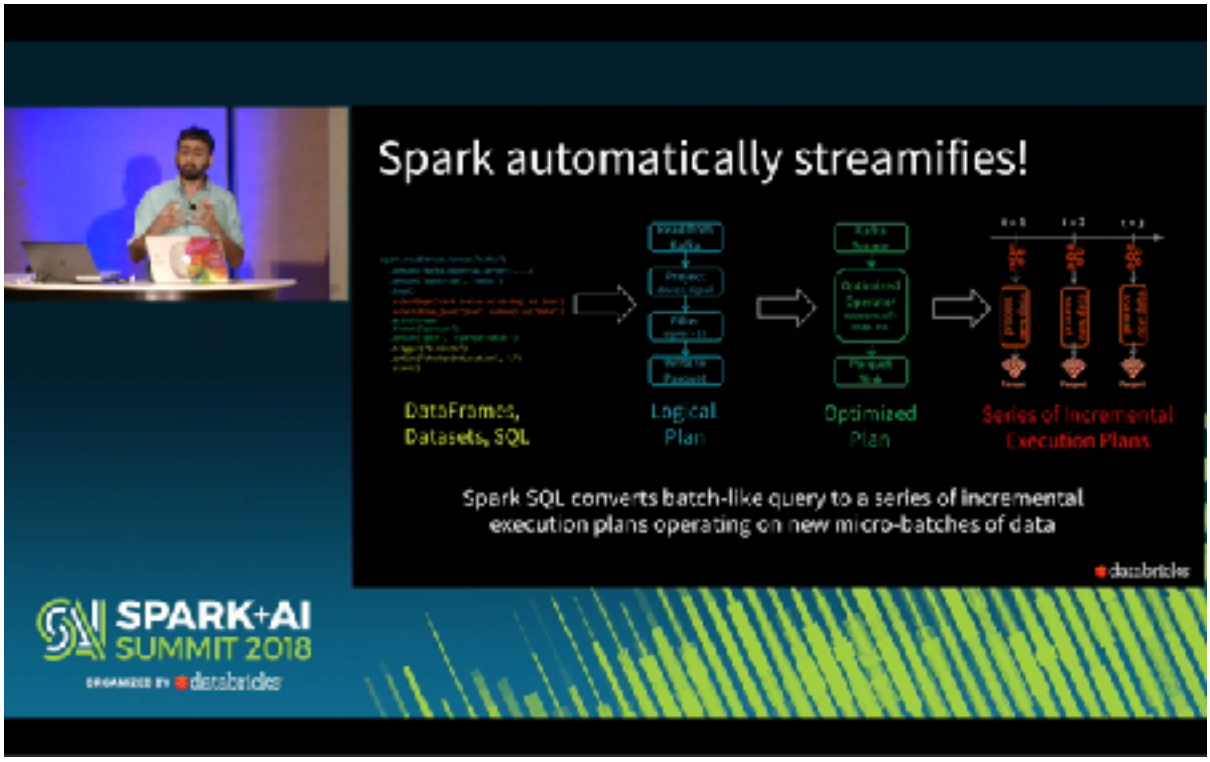
```
  .option("checkpointLocation", "path/to/HDFS/dir")
```

In case of a failure or intentional shutdown, you can recover the previous progress and state of a previous query, and continue where it left off. This is done using checkpointing and write-ahead

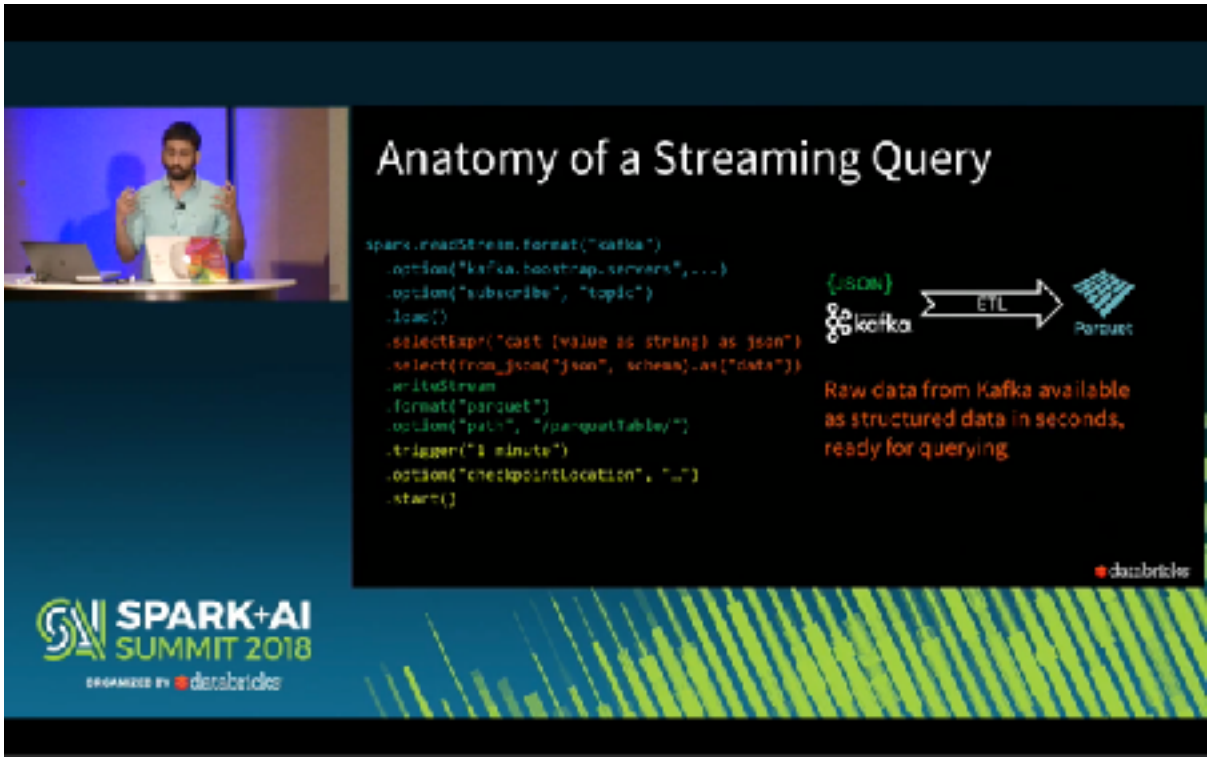
logs. You can configure a query with a checkpoint location, and the query will save all the progress information (i.e. range of offsets processed in each trigger) and the running aggregates (e.g. word counts in the quick example) to the checkpoint location. This checkpoint location has to be a path in an HDFS compatible file system, and can be set as an option in the DataStreamWriter when starting a query.

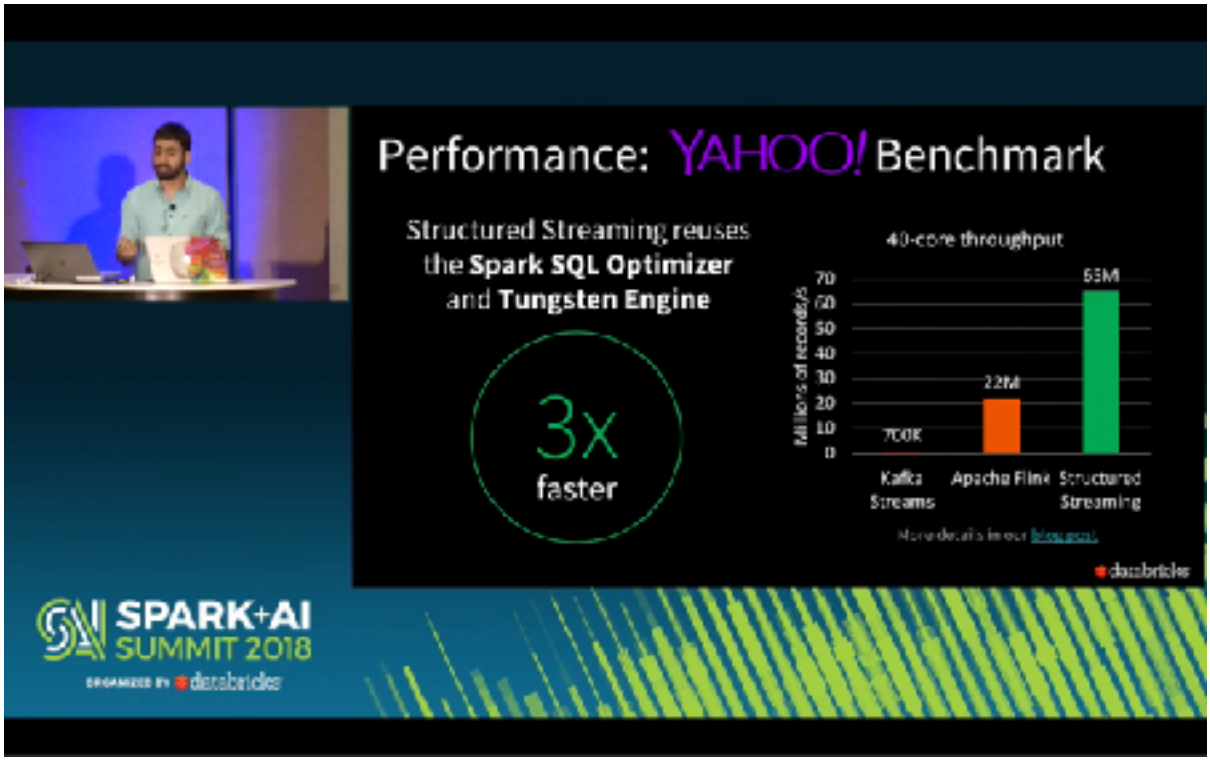


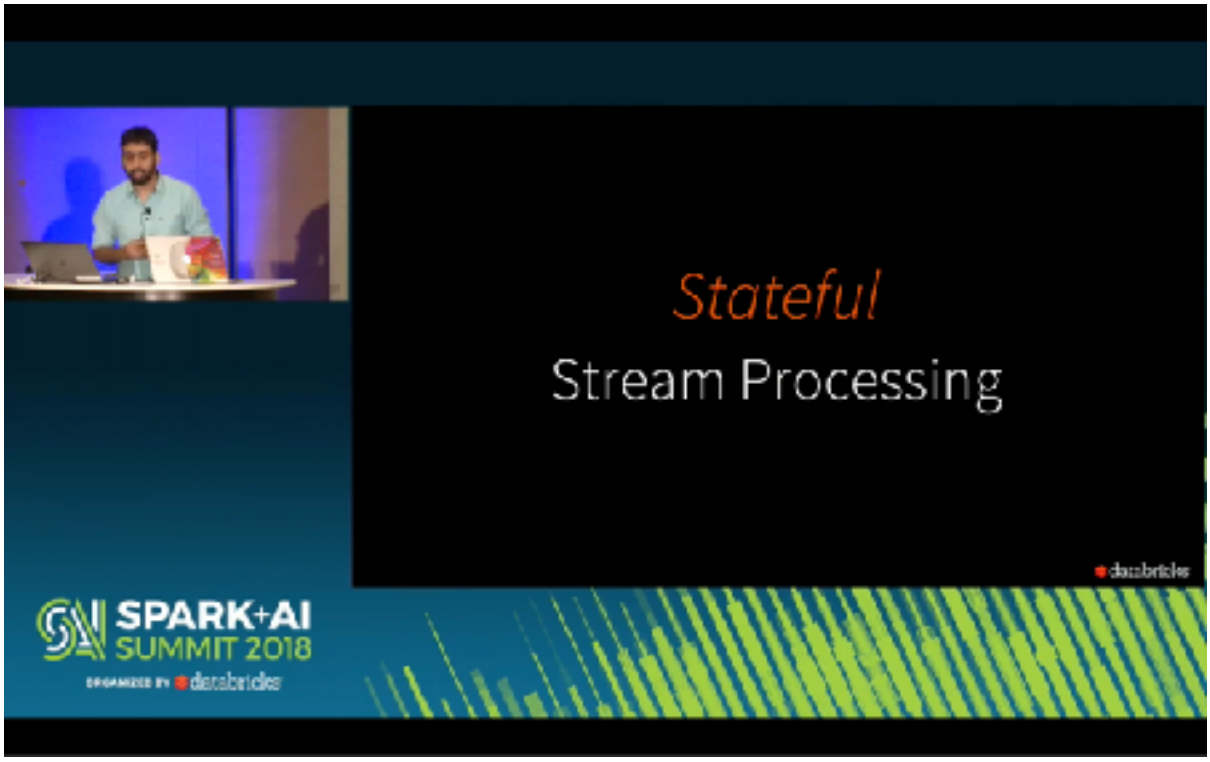


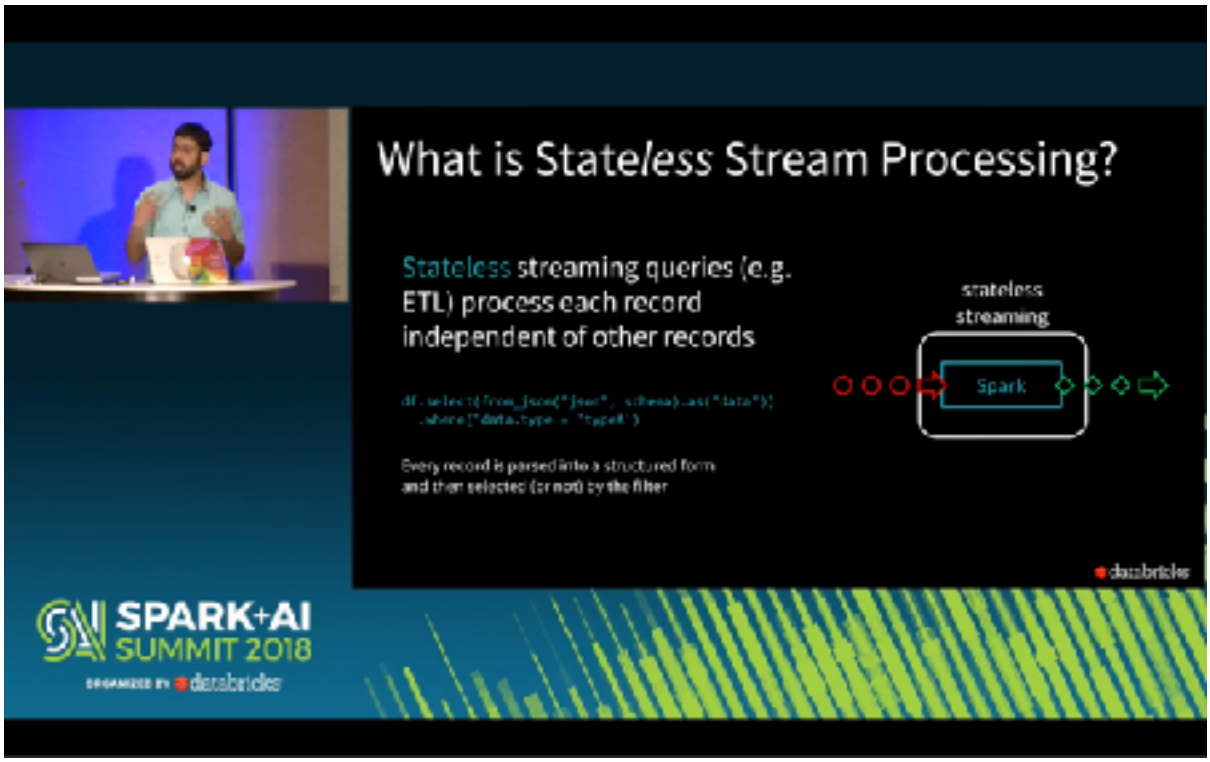

Spark automatically streamifies!
DataFrames, Datasets, SQL
Logical Plan
Optimized Plan
Series of Incremental Execution Plans
Spark SQL converts batch-like query to a series of incremental execution plans operating on new micro-batches of data
databricks
SPARK+AI SUMMIT 2018
ORGANIZED BY databricks


Fault-tolerance with Checkpointing
Checkpointing
Saves processed offset info to stable storage
Saved as JSON for forward-compatibility
Allows recovery from any failure
Can resume after limited changes to your streaming transformations (e.g. adding new filters to drop corrupted data, etc.)
end-to-end exactly-once guarantees
databricks
SPARK+AI SUMMIT 2018
ORGANIZED BY databricks


Anatomy of a Streaming Query
spark.readStream.format("kafka")
.option("kafka.bootstrap.servers",...)
.option("subscribe", "topic")
.load()
.selectExpr("cast (value as string) as json")
.select(from_json("json", schema).as("data"))
.writeStream
.format("parquet")
.option("path", "/parquetTable/")
.trigger("1 minute")
.option("checkpointLocation", "...")
.start()
{JSON}
kafka
ETL
Parquet
Raw data from Kafka available as structured data in seconds, ready for querying
databricks
SPARK+AI SUMMIT 2018
ORGANIZED BY databricks


Performance: YAHOO! Benchmark
Structured Streaming reuses the Spark SQL Optimizer and Tungsten Engine
3x faster
40-core throughput
Millions of records/s
70
60
50
40
30
20
10
0
65M
22M
700K
Kafka Streams
Apache Flink
Structured Streaming
databricks
SPARK+AI SUMMIT 2018
ORGANIZED BY databricks


Stateful
Stream Processing
databricks
SPARK+AI
SUMMIT 2018
ORGANIZED BY databricks


What is State*less* Stream Processing?
Stateless streaming queries (e.g. ETL) process each record independent of other records
stateless streaming
Spark
Every record is parsed into a structured form and then selected (or not) by the filter
databricks
SPARK+AI
SUMMIT 2018
ORGANIZED BY databricks

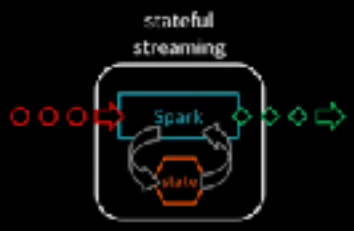
What is State*ful* Stream Processing?

Stateful streaming queries combine information from multiple records together

State is the information that is maintained for future use

```
df.select("from_json(...) as data")
  .where("data.type = 'type0'")
  .count()
```

Count is the streaming state and every selected record increments the count



Stateful Micro-Batch Processing

State is versioned between micro-batches while streaming query is running

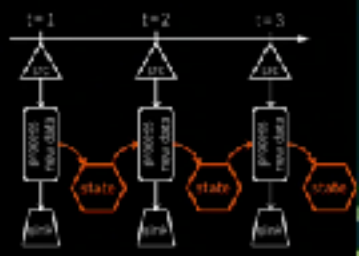
micro-batch incremental execution

Stateful Micro-Batch Processing

State is versioned between micro-batches while streaming query is running

Each micro-batch reads previous version state and updates it to new version

Versions used for fault recovery



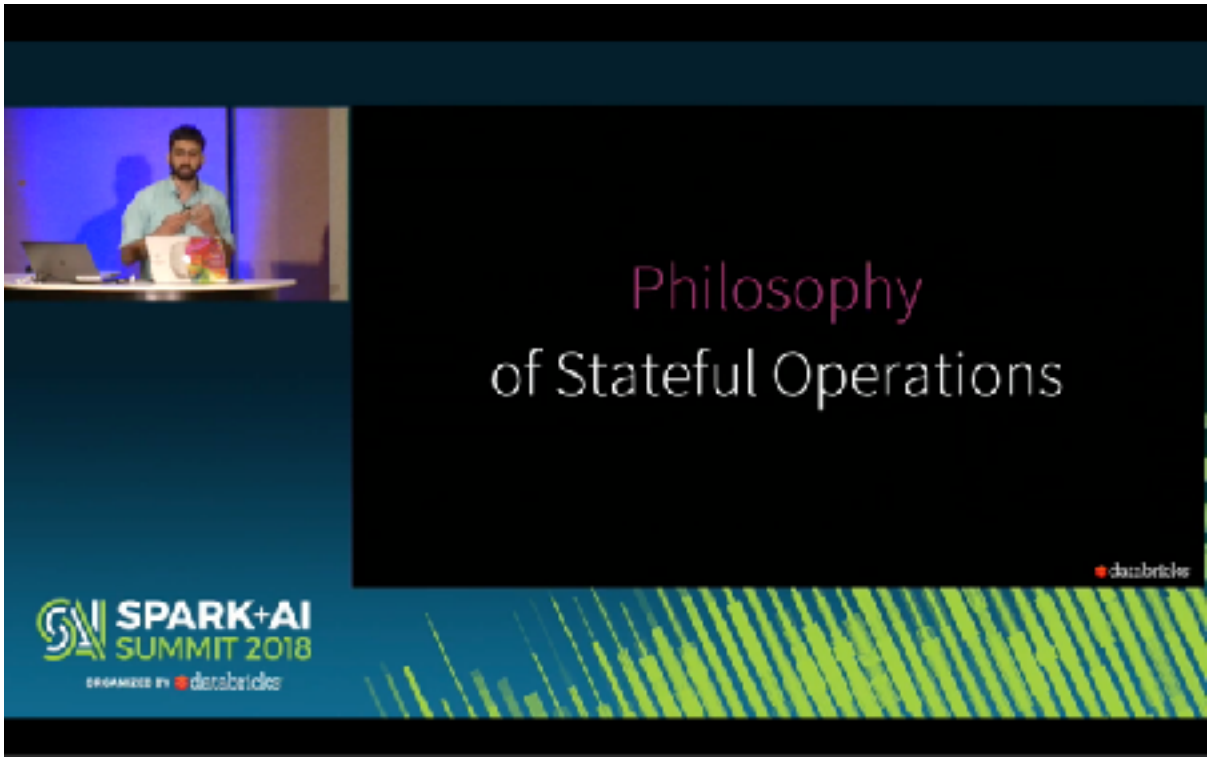
micro-batch incremental execution

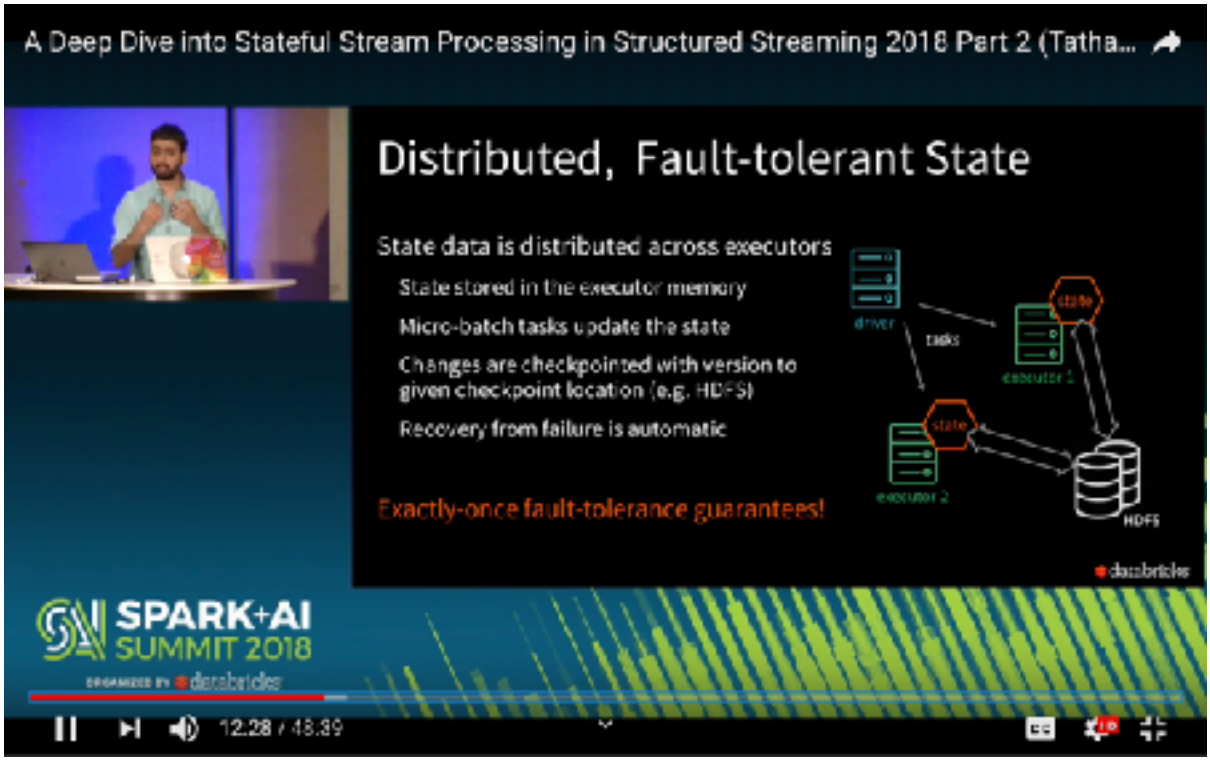
Distributed, Fault-tolerant State

State data is distributed across executors

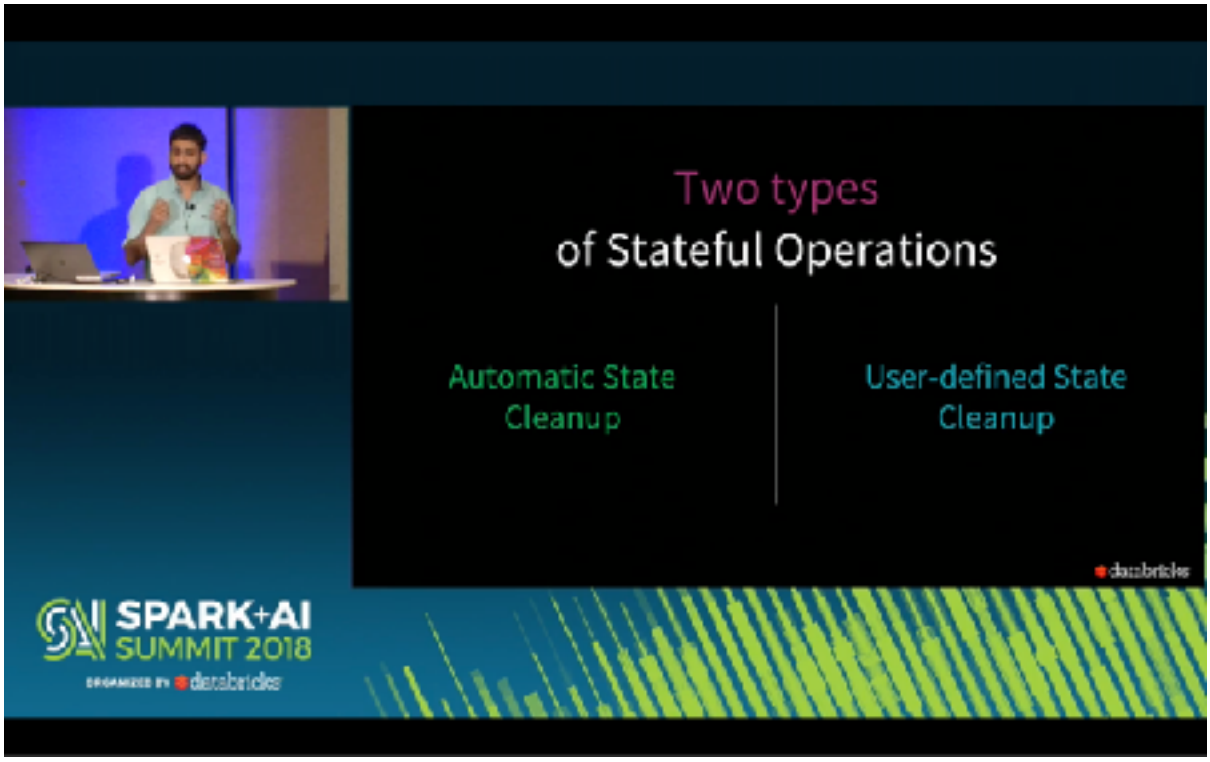
- State stored in the executor memory
- Micro-batch tasks update the state


Distributed, Fault-tolerant State
State data is distributed across executors
State stored in the executor memory
Micro-batch tasks update the state
Changes are checkpointed with version to given checkpoint location (e.g. HDFS)
Recovery from failure is automatic
driver
tasks
state
executor 1
state
executor 2
HDFS
databricks
SPARK+AI SUMMIT 2018
databricks

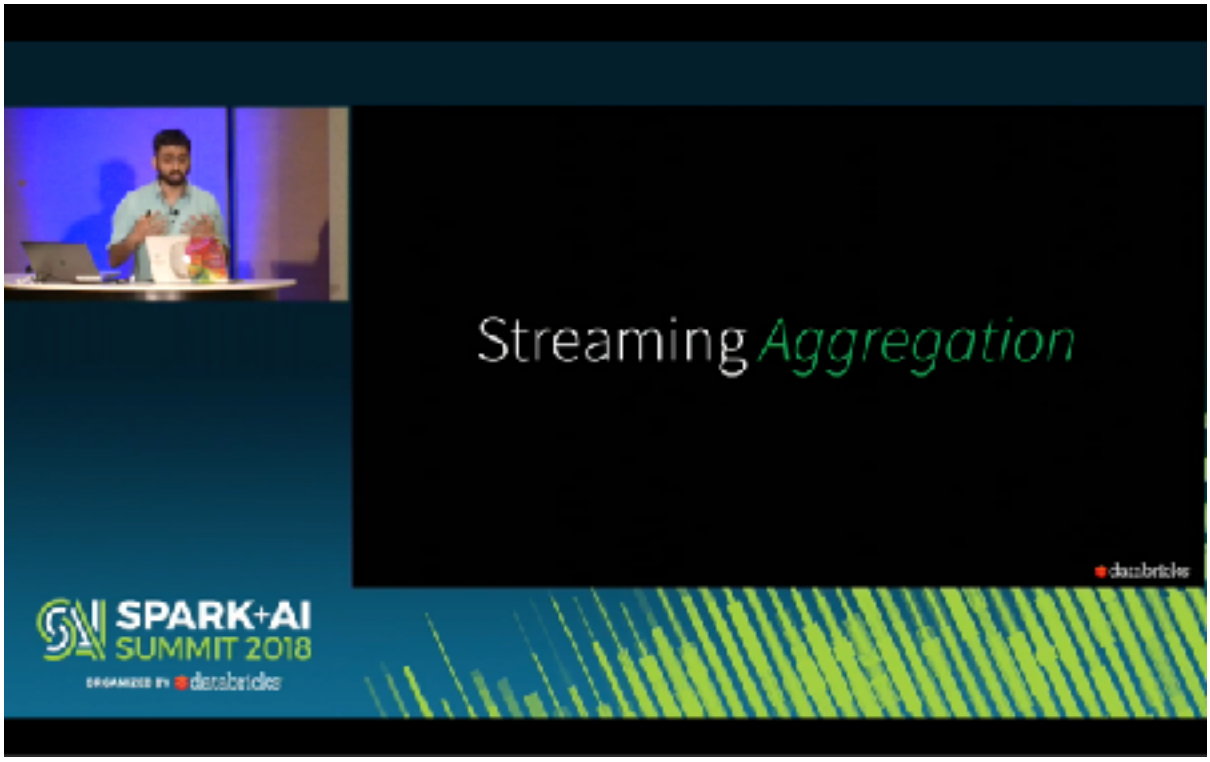

Philosophy
of Stateful Operations
databricks
SPARK+AI SUMMIT 2018
databricks


A Deep Dive into Stateful Stream Processing in Structured Streaming 2018 Part 2 (Tatha...
Distributed, Fault-tolerant State
State data is distributed across executors
State stored in the executor memory
Micro-batch tasks update the state
Changes are checkpointed with version to given checkpoint location (e.g. HDFS)
Recovery from failure is automatic
Exactly-once fault-tolerance guarantees!
driver
tasks
state
executor 1
executor 2
HDFS
databricks
SPARK+AI SUMMIT 2018
12:28 / 48:39


Philosophy
of Stateful Operations
databricks
SPARK+AI SUMMIT 2018


Two types
of Stateful Operations
Automatic State Cleanup
User-defined State Cleanup
SPARK+AI SUMMIT 2018
databricks


Rest of this talk
Explore built-in stateful operations
How to use watermarks to control state size
How to build arbitrary stateful operations
How to monitor and debug stateful queries
SPARK+AI SUMMIT 2018
databricks


Streaming Aggregation
databricks
SPARK+AI
SUMMIT 2018
ORGANIZED BY databricks


Aggregation by key and/or time windows
Aggregation by key only
events
.groupBy("key")
.count()
databricks
SPARK+AI
SUMMIT 2018
ORGANIZED BY databricks

Aggregation by key and/or time windows

Aggregation by key only

```
events
  .groupBy("key")
  .count()
```

Aggregation by event time windows

```
events
  .groupBy(window("timestamp","10 mins"))
  .avg("value")
```


Aggregation by key and/or time windows

Aggregation by key only

```
events
  .groupBy("key")
  .count()
```

Aggregation by event time windows

```
events
  .groupBy(window("timestamp","10 mins"))
  .avg("value")
```

Aggregation by both

```
events
  .groupBy(
    col(key),
    window("timestamp","10 mins"))
  .agg(avg("value"), corr("value"))
```

Supports multiple aggregations, user-defined functions (UDAFs)!

Automatically handles Late Data

Keeping state allows late data to update counts of old windows

But size of the state increases indefinitely if old windows are not dropped

red = state updated with late data

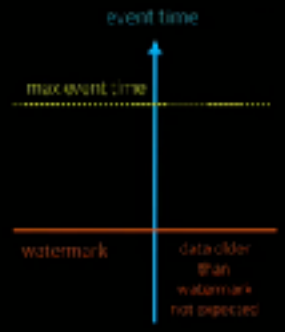
databricks

SPARK+AI SUMMIT 2018

ORGANIZED BY databricks

Watermarking

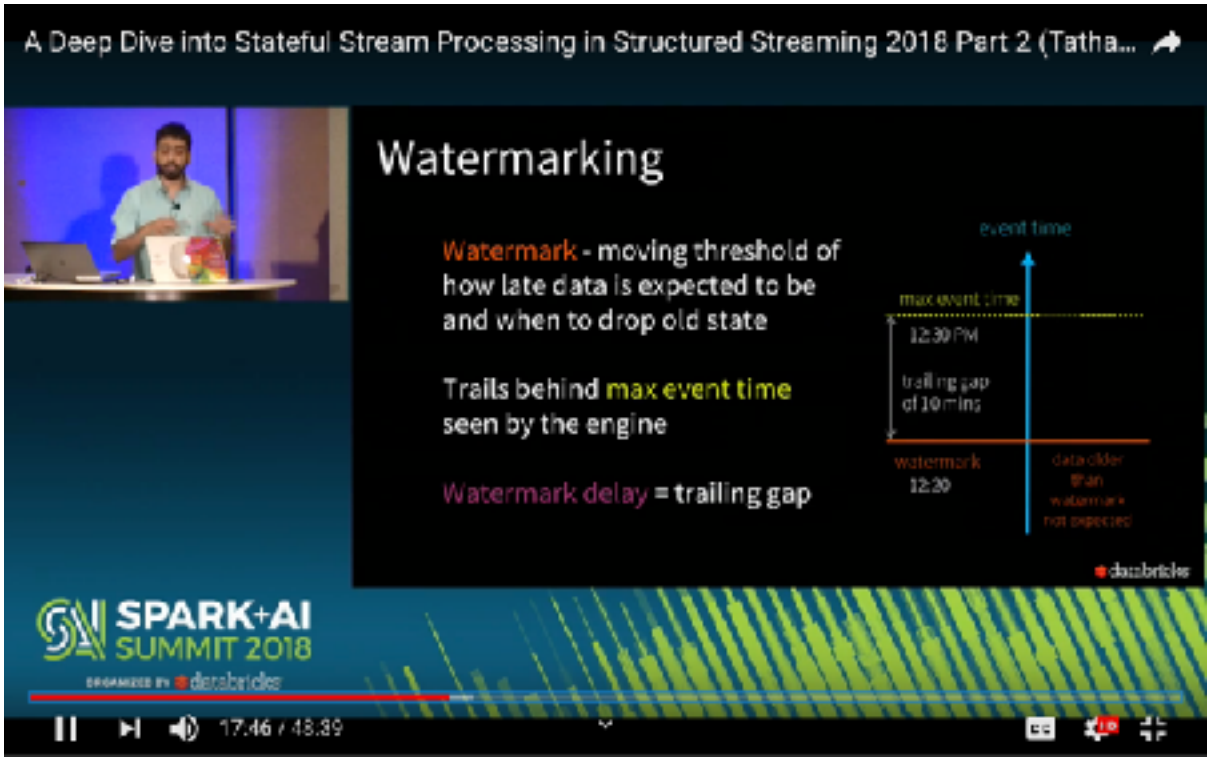
Watermark - moving threshold of how late data is expected to be and when to drop old state

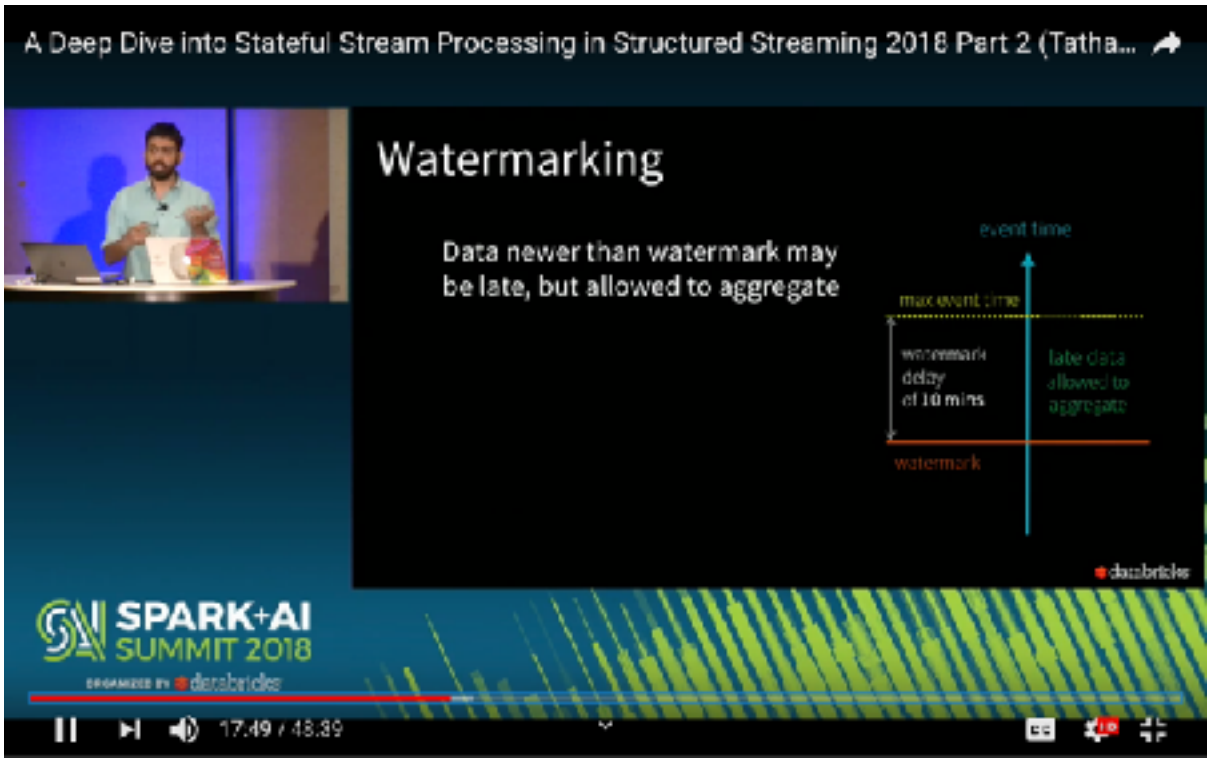


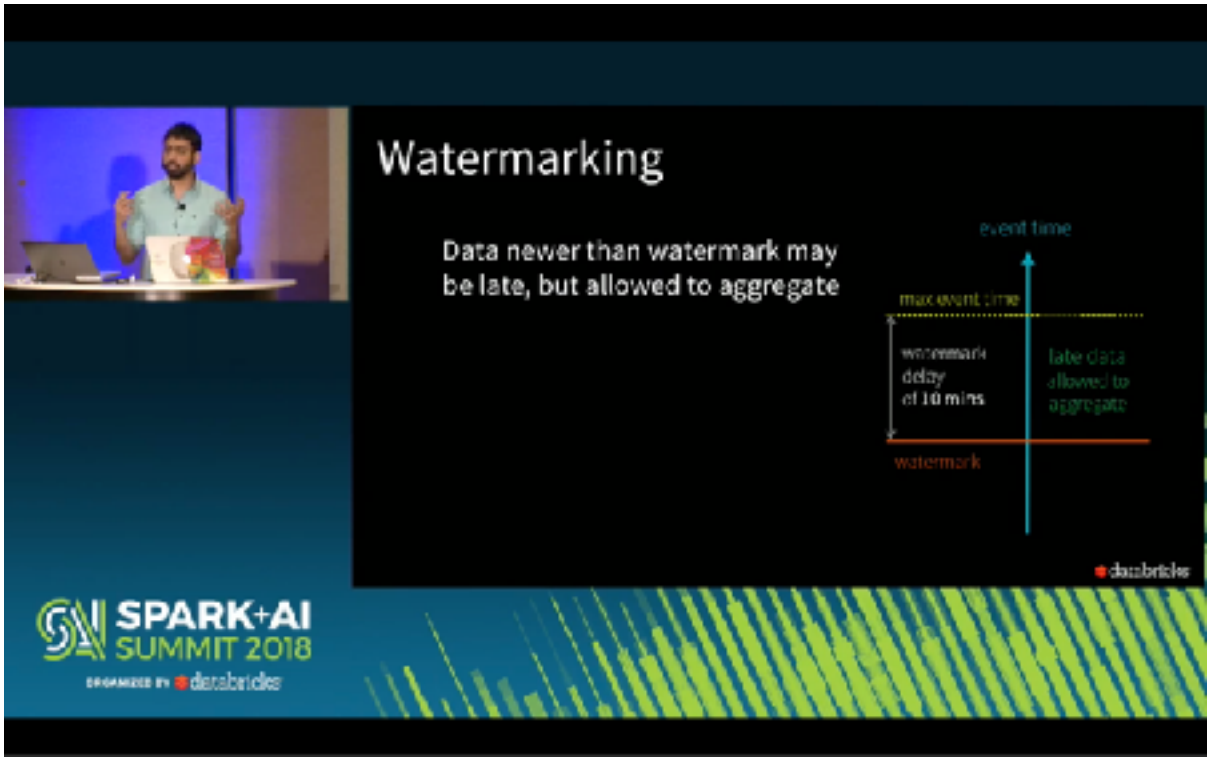
databricks

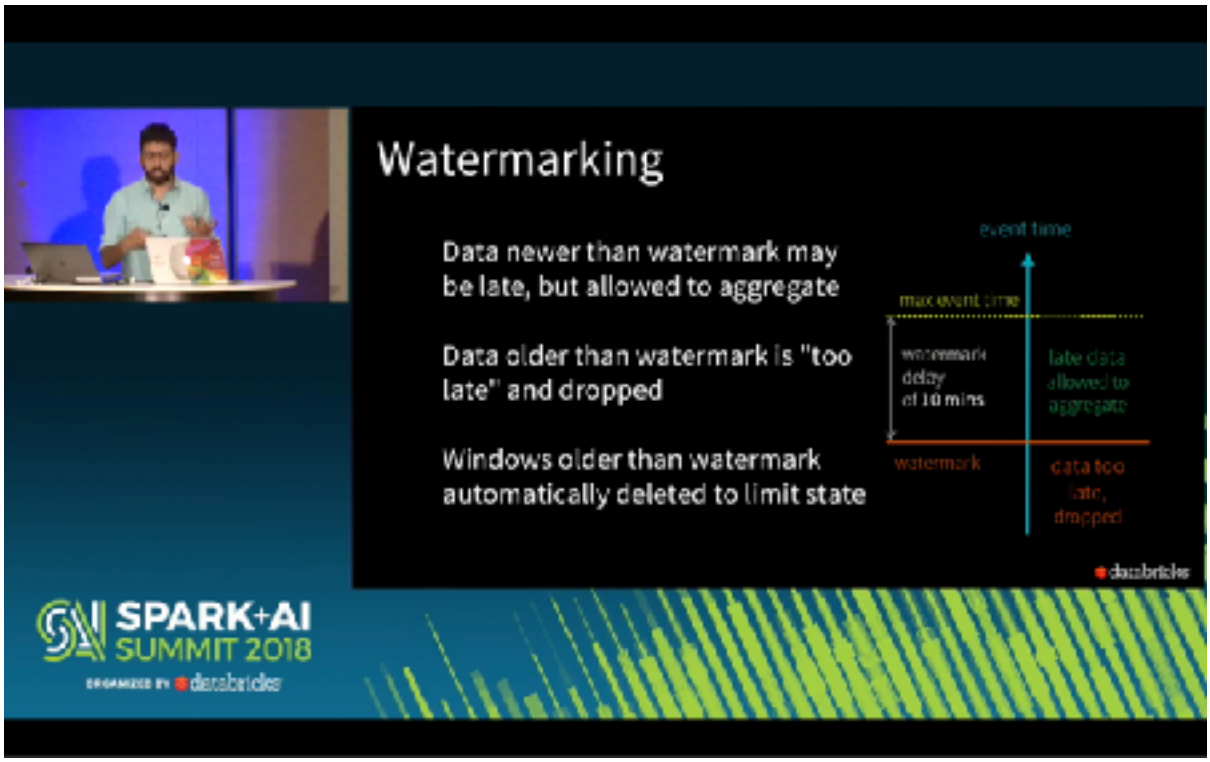
SPARK+AI SUMMIT 2018

ORGANIZED BY databricks

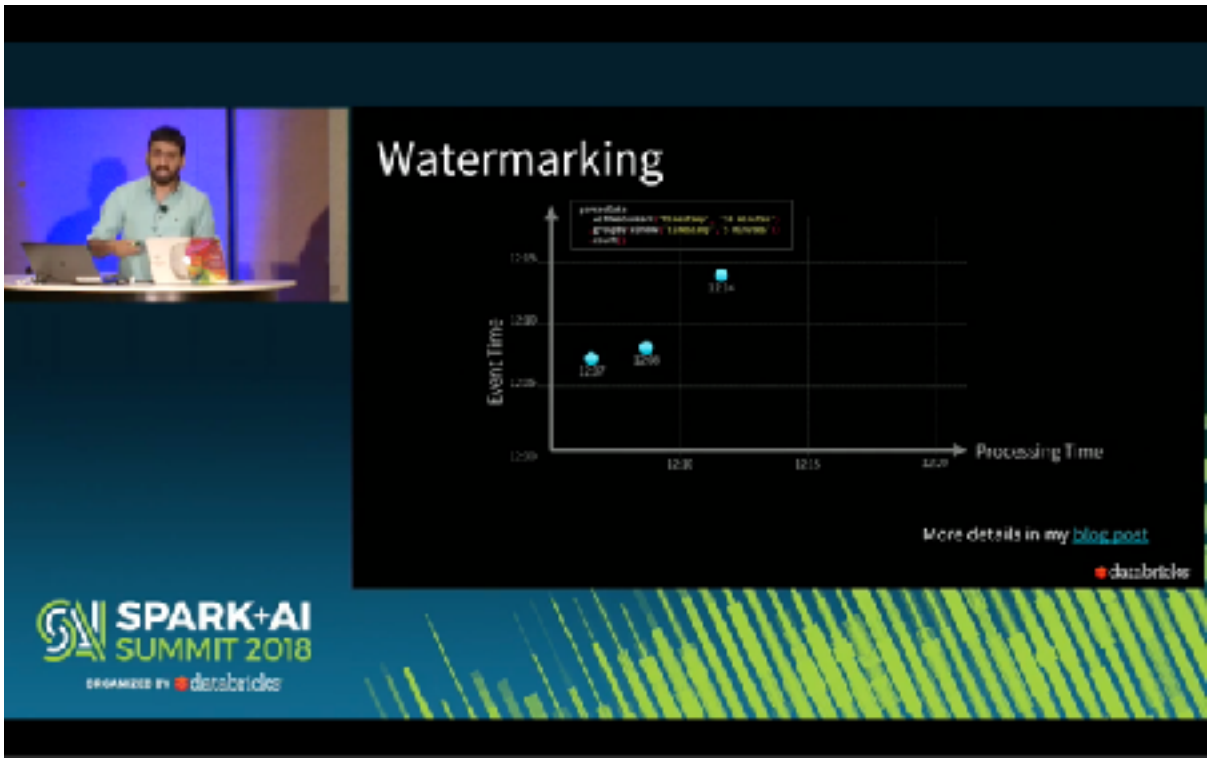

A Deep Dive into Stateful Stream Processing in Structured Streaming 2018 Part 2 (Tatha...
Watermarking
Watermark - moving threshold of how late data is expected to be and when to drop old state
Trails behind max event time seen by the engine
Watermark delay = trailing gap
event time
max event time
12:30 PM
trailing gap of 10 mins
watermark
12:20
data older than watermark not expected
SPARK+AI SUMMIT 2018
17:46 / 48:39

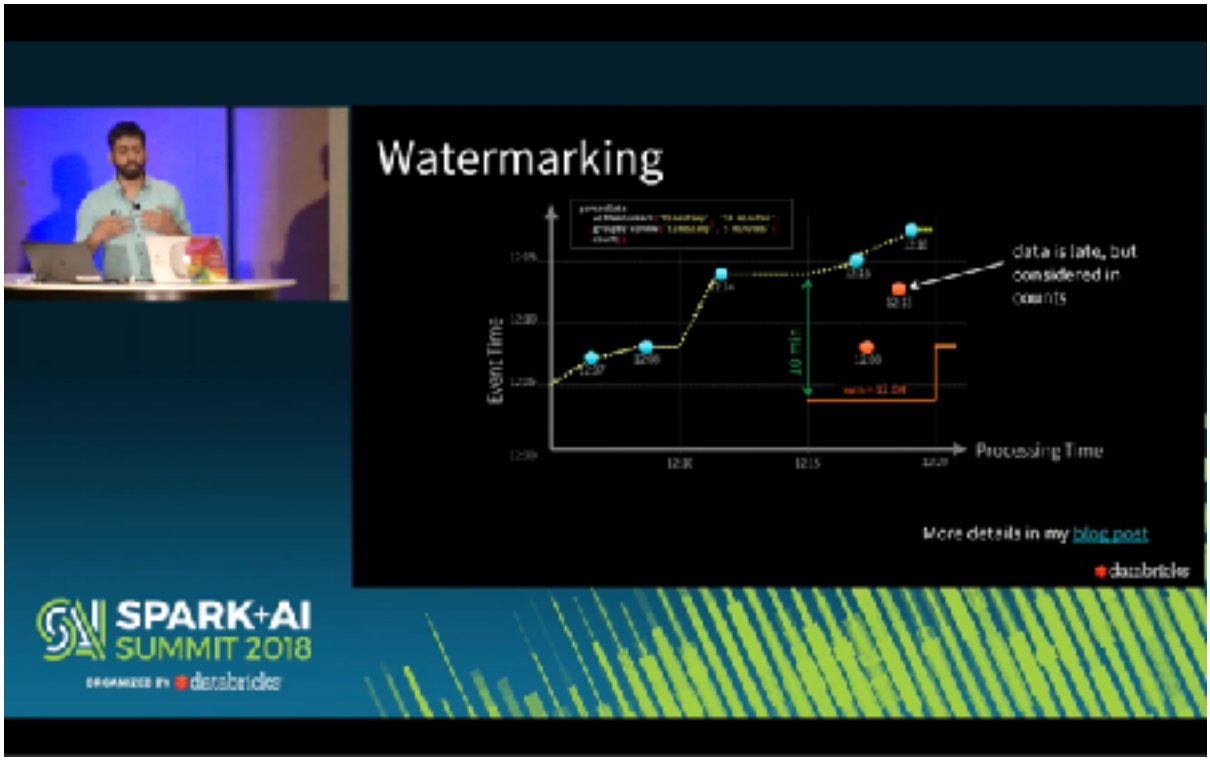

A Deep Dive into Stateful Stream Processing in Structured Streaming 2018 Part 2 (Tatha...
Watermarking
Data newer than watermark may be late, but allowed to aggregate
event time
max event time
watermark delay of 10 mins
late data allowed to aggregate
watermark
SPARK+AI SUMMIT 2018
17:49 / 48:39

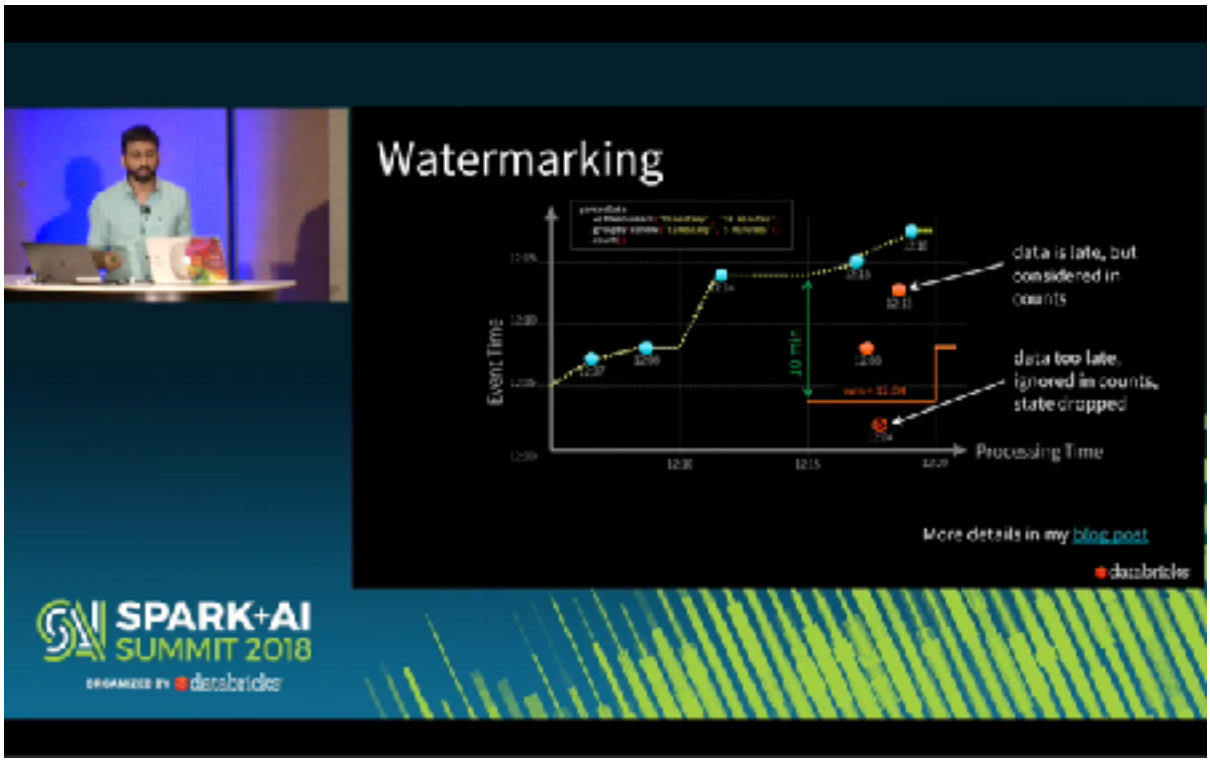

Watermarking
Data newer than watermark may be late, but allowed to aggregate
event time
max event time
late data allowed to aggregate
watermark
SPARK+AI SUMMIT 2018

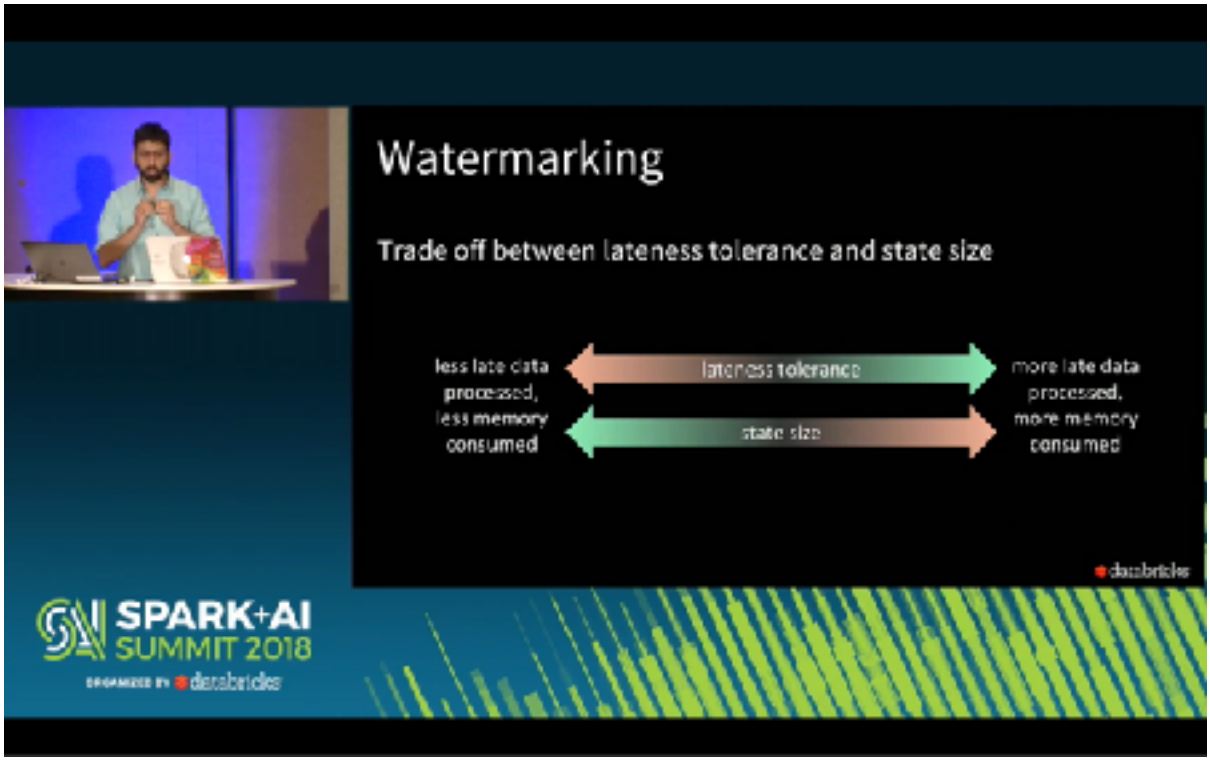

Watermarking
Data newer than watermark may be late, but allowed to aggregate
Data older than watermark is "too late" and dropped
Windows older than watermark automatically deleted to limit state
event time
max event time
late data allowed to aggregate
watermark
SPARK+AI SUMMIT 2018

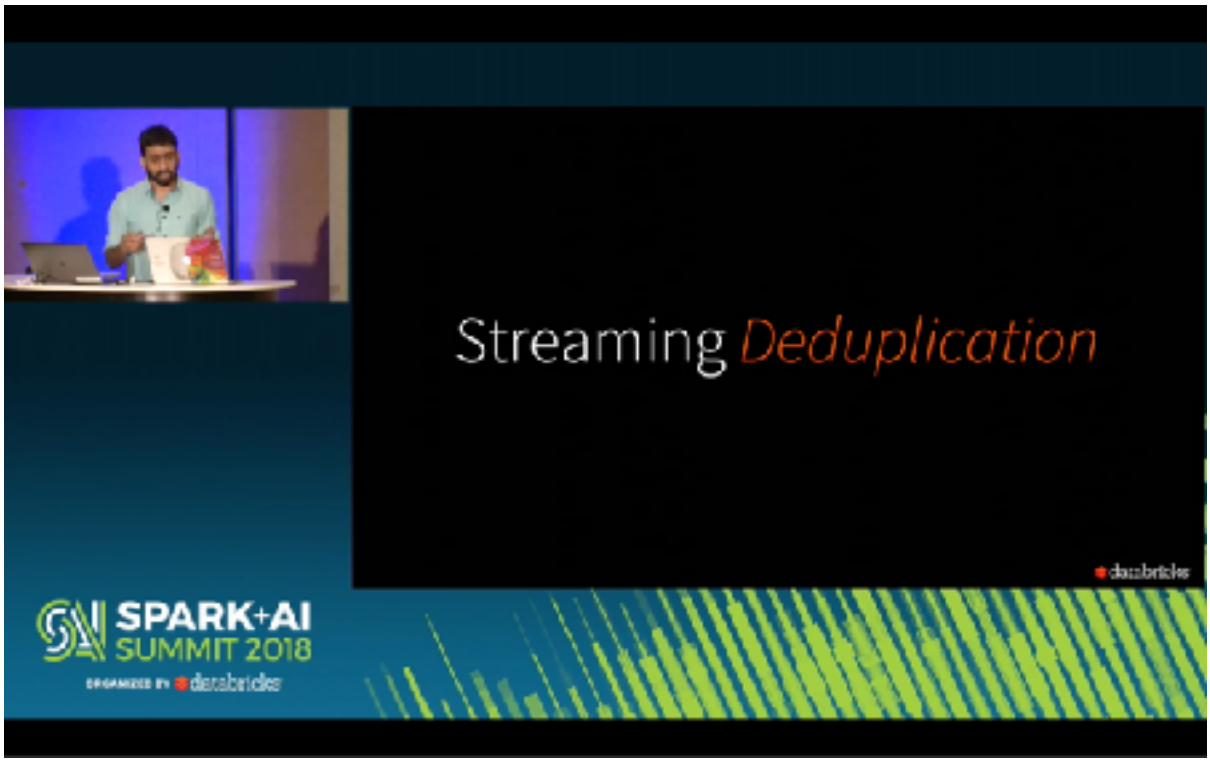

Watermarking
parsedData
.withWatermark("timestamp", "10 minutes")
.groupBy(window("timestamp","5 minutes"))
.count()
Used only in stateful operations
Ignored in non-stateful streaming queries and batch queries
event time
max event time
watermark delay of 10 mins
late data allowed to aggregate
watermark
data too late, dropped
databricks
SPARK+AI SUMMIT 2018
ORGANIZED BY databricks


Watermarking
Event Time
Processing Time
More details in my blog post
databricks
SPARK+AI SUMMIT 2018
ORGANIZED BY databricks


Watermarking
Event Time
10 min
Processing Time
data is late, but considered in counts
More details in my blog post
databricks
SPARK+AI SUMMIT 2018


Watermarking
Event Time
10 min
Processing Time
data is late, but considered in counts
data too late, ignored in counts, state dropped
More details in my blog post
databricks
SPARK+AI SUMMIT 2018


Watermarking
Trade off between lateness tolerance and state size
less late data processed, less memory consumed
lateness tolerance
state size
more late data processed, more memory consumed
databricks
SPARK+AI SUMMIT 2018
databricks


Streaming Deduplication
databricks
SPARK+AI SUMMIT 2018
databricks

Streaming Deduplication

Drop duplicate records in a stream

Specify columns which uniquely identify a record

```
userActions
  .dropDuplicates("uniqueRecordId")
```

Spark SQL will store past unique column values as state and drop any record that matches the state

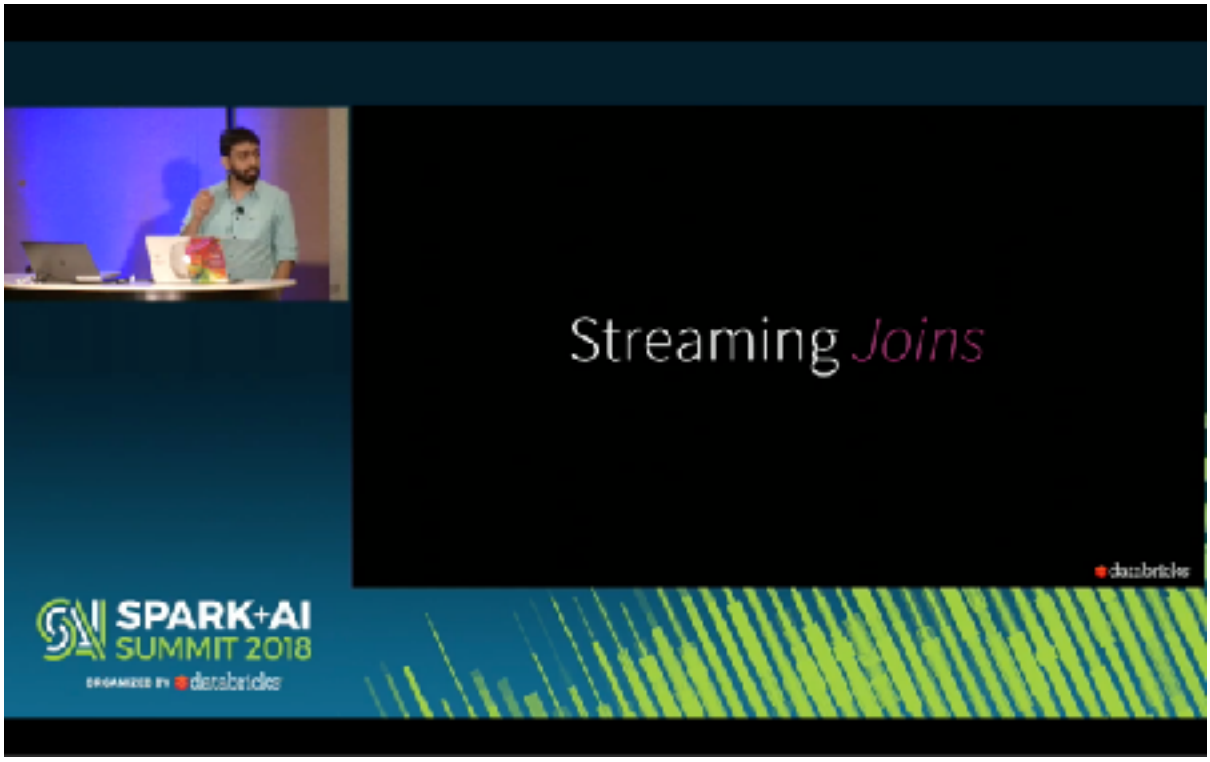
Streaming Deduplication with Watermark

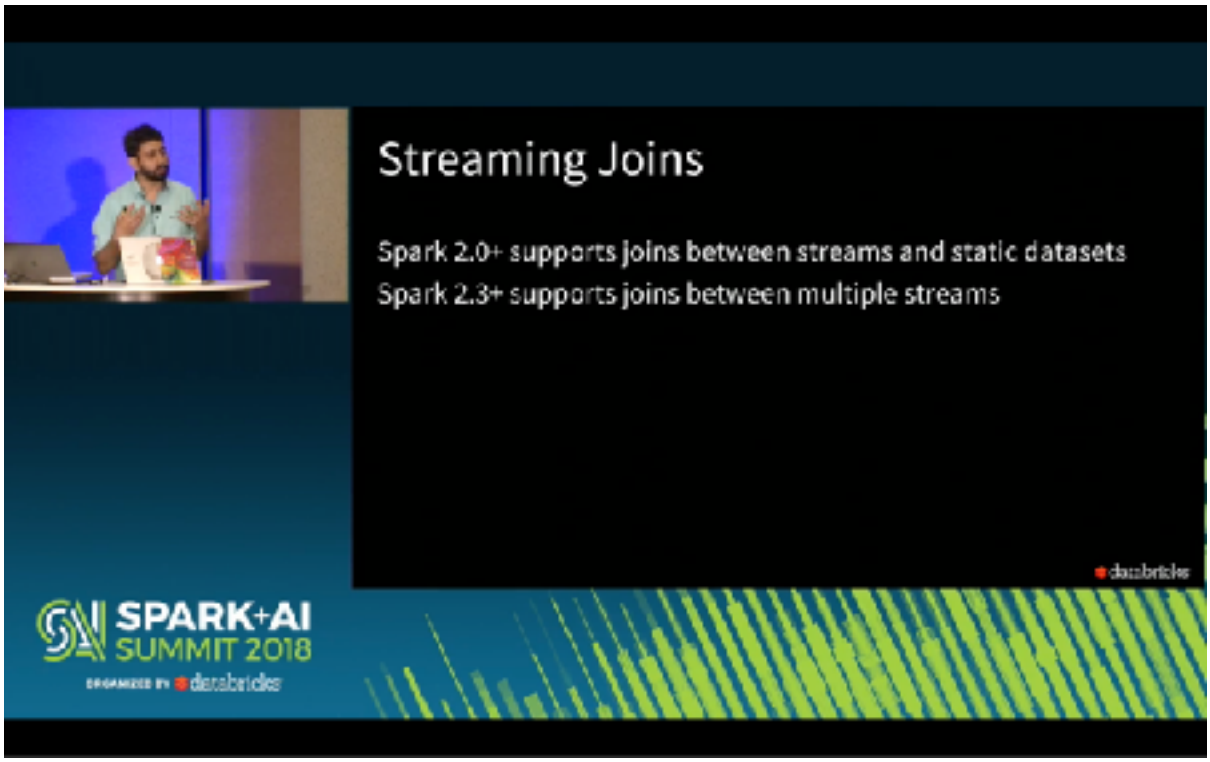
Timestamp as a unique column along with watermark allows old values in state to dropped

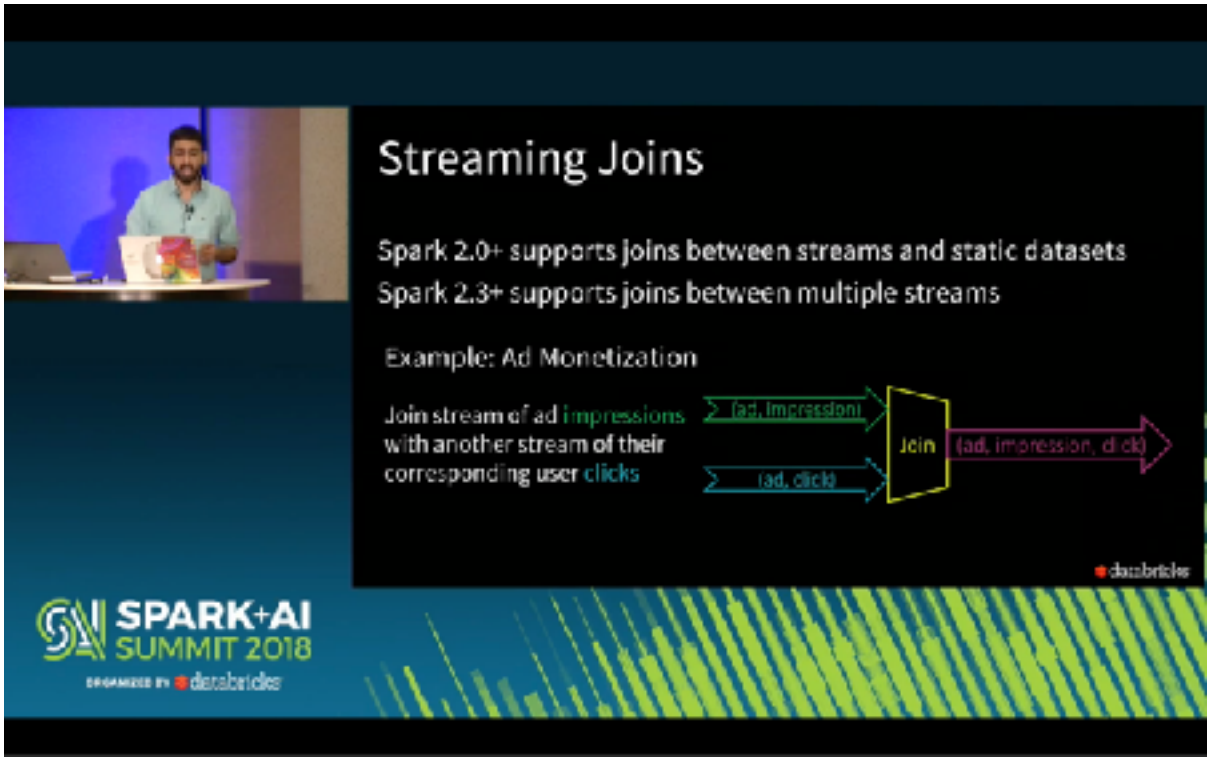
- Records older than watermark delay is not going to get any further duplicates

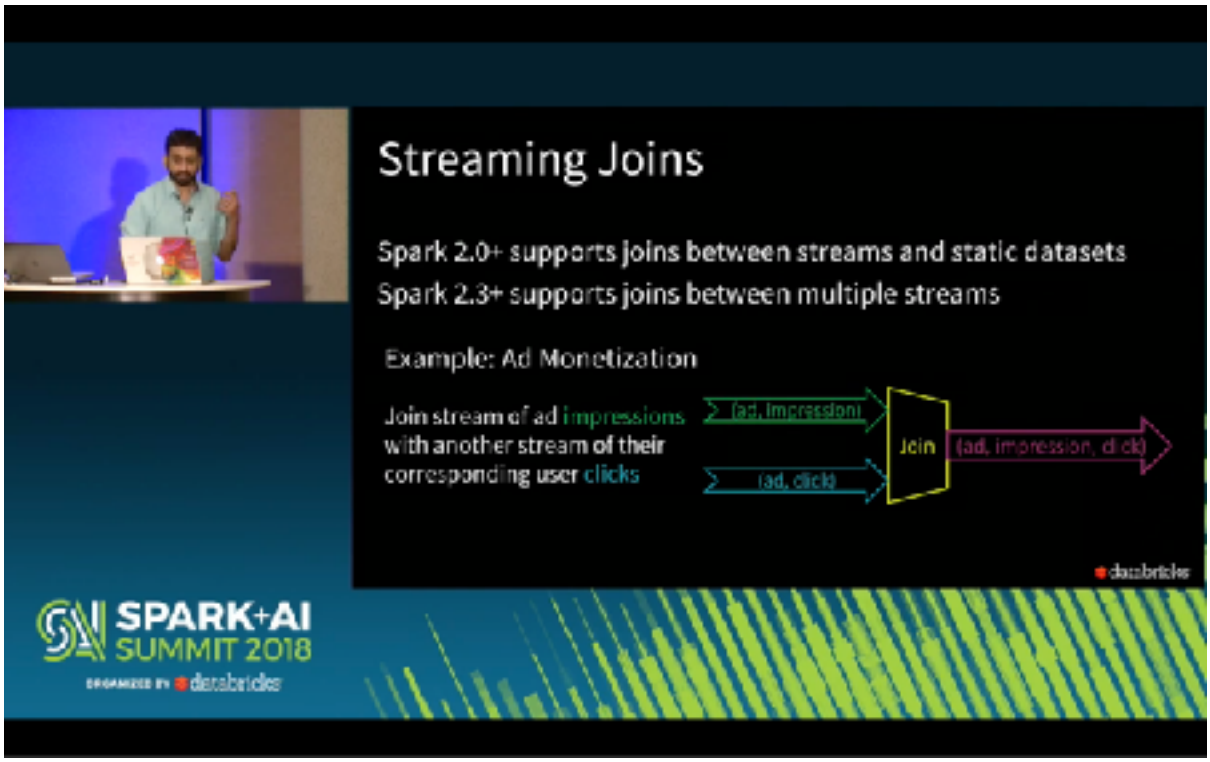
```
userActions
  .withWatermark("timestamp")
  .dropDuplicates(
    "uniqueRecordId",
    "timestamp")
```

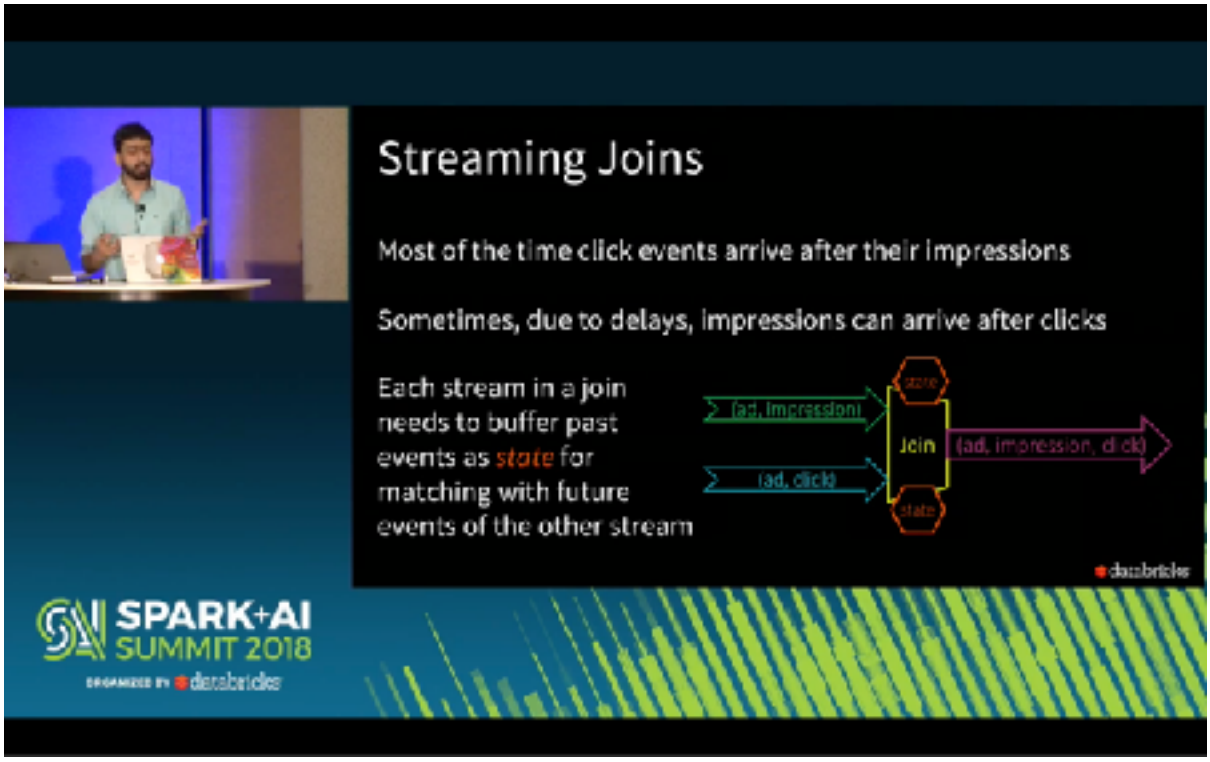
Timestamp must be same for duplicated records

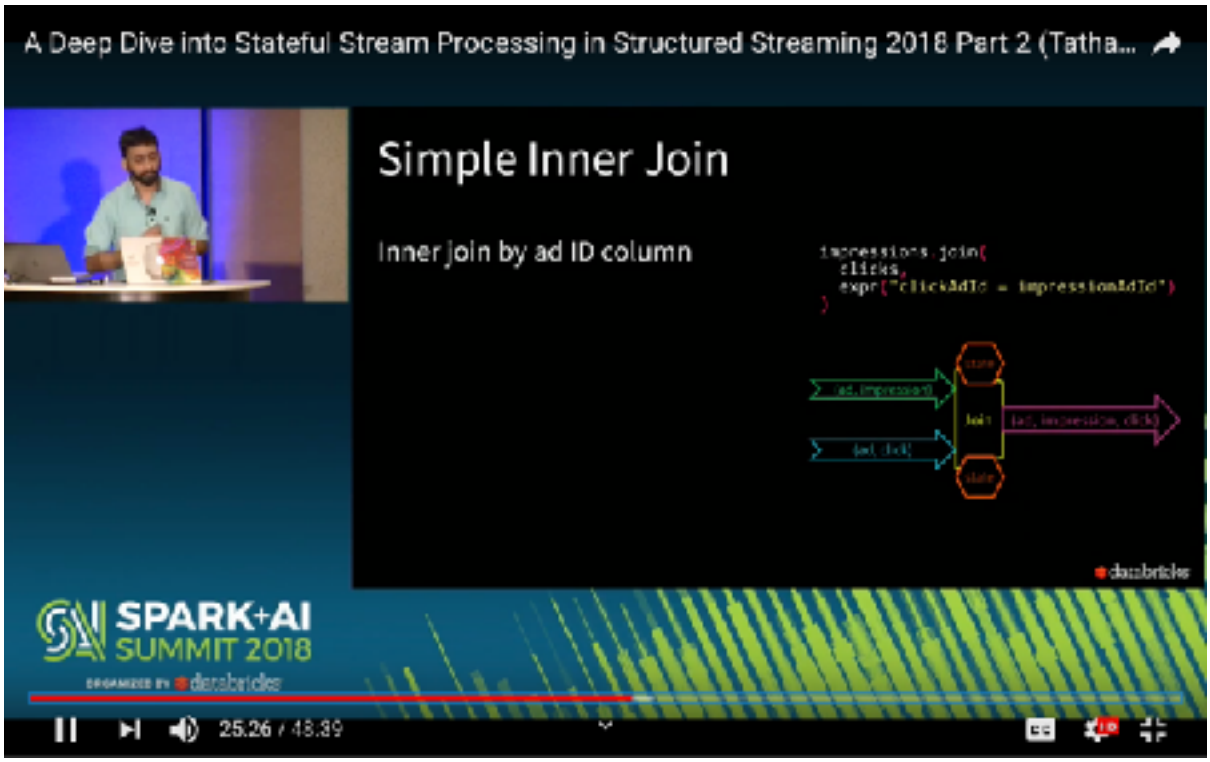

Streaming Joins
databricks
SPARK+AI
SUMMIT 2018
databricks

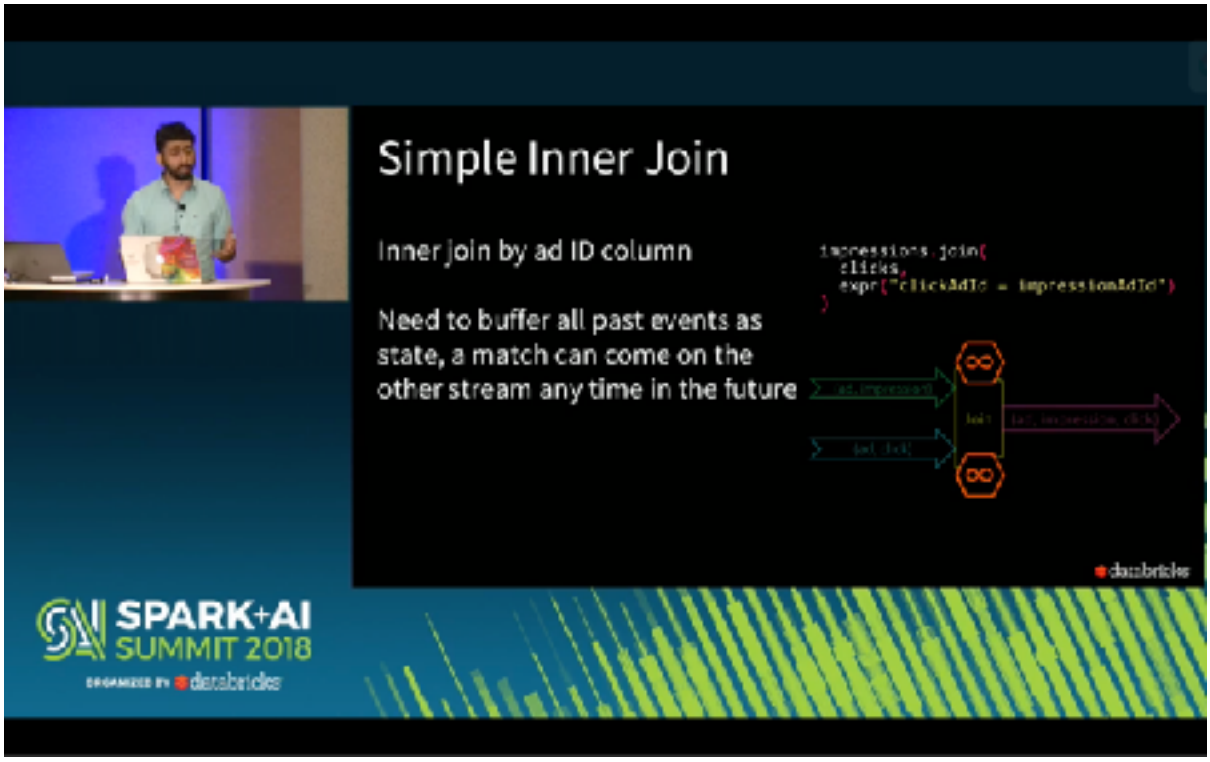

Streaming Joins
Spark 2.0+ supports joins between streams and static datasets
Spark 2.3+ supports joins between multiple streams
databricks
SPARK+AI
SUMMIT 2018
databricks

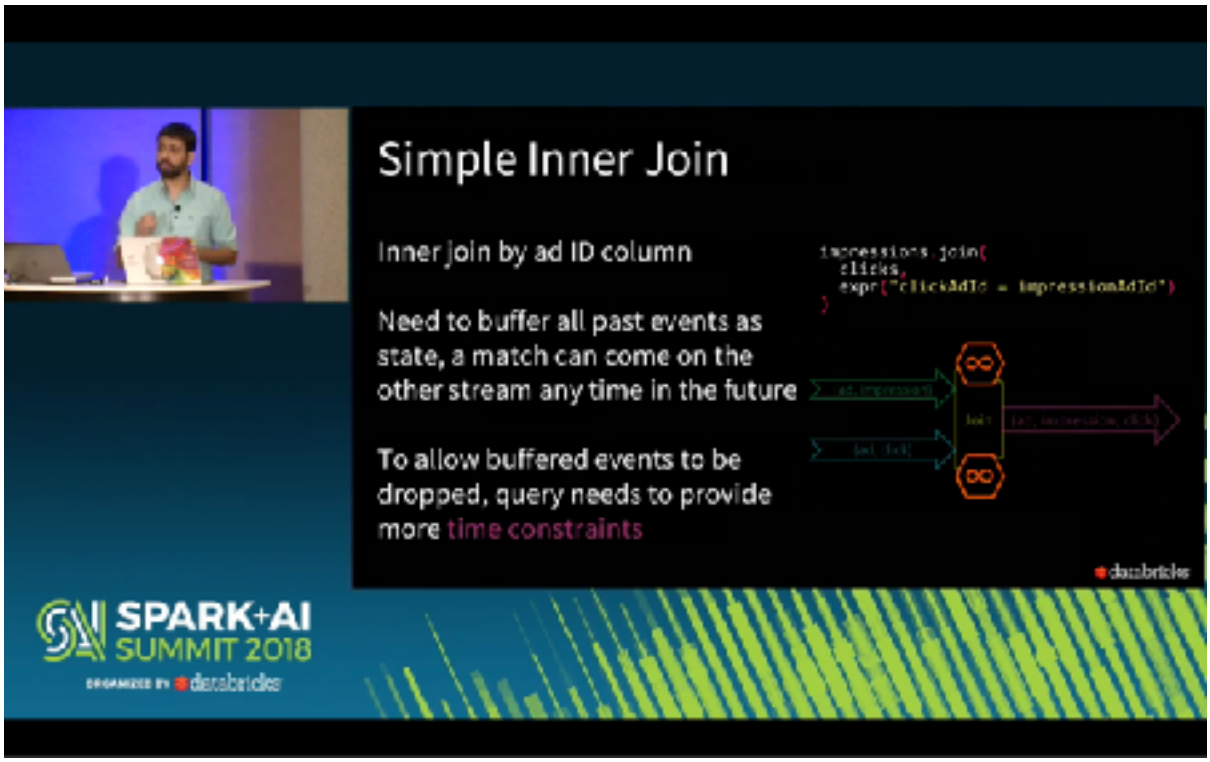

Streaming Joins
Spark 2.0+ supports joins between streams and static datasets
Spark 2.3+ supports joins between multiple streams
Example: Ad Monetization
Join stream of ad impressions with another stream of their corresponding user clicks
(ad, impression)
(ad, click)
Join
(ad, impression, click)
databricks
SPARK+AI SUMMIT 2018
databricks


Streaming Joins
Spark 2.0+ supports joins between streams and static datasets
Spark 2.3+ supports joins between multiple streams
Example: Ad Monetization
Join stream of ad impressions with another stream of their corresponding user clicks
(ad, impression)
(ad, click)
Join
(ad, impression, click)
databricks
SPARK+AI SUMMIT 2018
databricks


Streaming Joins
Most of the time click events arrive after their impressions
Sometimes, due to delays, impressions can arrive after clicks
Each stream in a join needs to buffer past events as state for matching with future events of the other stream
(ad, impression)
(ad, click)
Join
(ad, impression, click)
state
state
databricks
SPARK+AI SUMMIT 2018


A Deep Dive into Stateful Stream Processing in Structured Streaming 2018 Part 2 (Tatha...
Simple Inner Join
Inner join by ad ID column
impressions.join(
clicks,
expr("clickAdId = impressionAdId")
)
state
(ad, impression)
Join
(ad, impression, click)
(ad, click)
state
databricks
SPARK+AI SUMMIT 2018
25:26 / 48:39


Simple Inner Join
Inner join by ad ID column
Need to buffer all past events as state, a match can come on the other stream any time in the future
impressions.join(
clicks,
expr("clickAdId = impressionAdId")
)
SPARK+AI SUMMIT 2018
databricks


Simple Inner Join
Inner join by ad ID column
Need to buffer all past events as state, a match can come on the other stream any time in the future
To allow buffered events to be dropped, query needs to provide more time constraints
impressions.join(
clicks,
expr("clickAdId = impressionAdId")
)
SPARK+AI SUMMIT 2018
databricks


Inner Join + Time constraints + Watermarks
Time constraints
Let's assume
- Impressions can be 2 hours late
- Clicks can be 3 hours late
- A click can occur within 1 hour after the corresponding impression
val impressionsWithWatermark = impressions
.withWatermark("impressionTime", "2 hours")
val clicksWithWatermark = clicks
.withWatermark("clickTime", "3 hours")
impressionsWithWatermark.join(
clicksWithWatermark,
expr("""
clickAdId = impressionAdId AND
clickTime >= impressionTime AND
clickTime <= impressionTime + interval 1 hour
""")
)
Range Join
SPARK+AI SUMMIT 2018
databricks


Arbitrary Stateful
Operations
databricks
SPARK+AI SUMMIT 2018

Arbitrary Stateful Operations

Many use cases require more complicated logic than SQL ops

Example: Tracking user activity on your product

- Input: User actions (login, clicks, logout, ...)
- Output: Latest user status (online, active, inactive, ...)

databricks

SPARK+AI SUMMIT 2018

ORGANIZED BY databricks

Arbitrary Stateful Operations

Many use cases require more complicated logic than SQL ops

Example: Tracking user activity on your product

- Input: User actions (login, clicks, logout, ...)
- Output: Latest user status (online, active, inactive, ...)

Solution: MapGroupsWithState / FlatMapGroupsWithState

- General API for per-key user-defined stateful processing
- Since Spark 2.2, for Scala and Java only

databricks

SPARK+AI SUMMIT 2018

ORGANIZED BY databricks

MapGroupsWithState / FlatMapGroupsWithState

No automatic state clean up and dropping of late data

- Adding watermark does not automatically manage late and state data
- Explicit state clean up by the user

More powerful + efficient than DStream's `mapWithState` and `updateStateByKey`

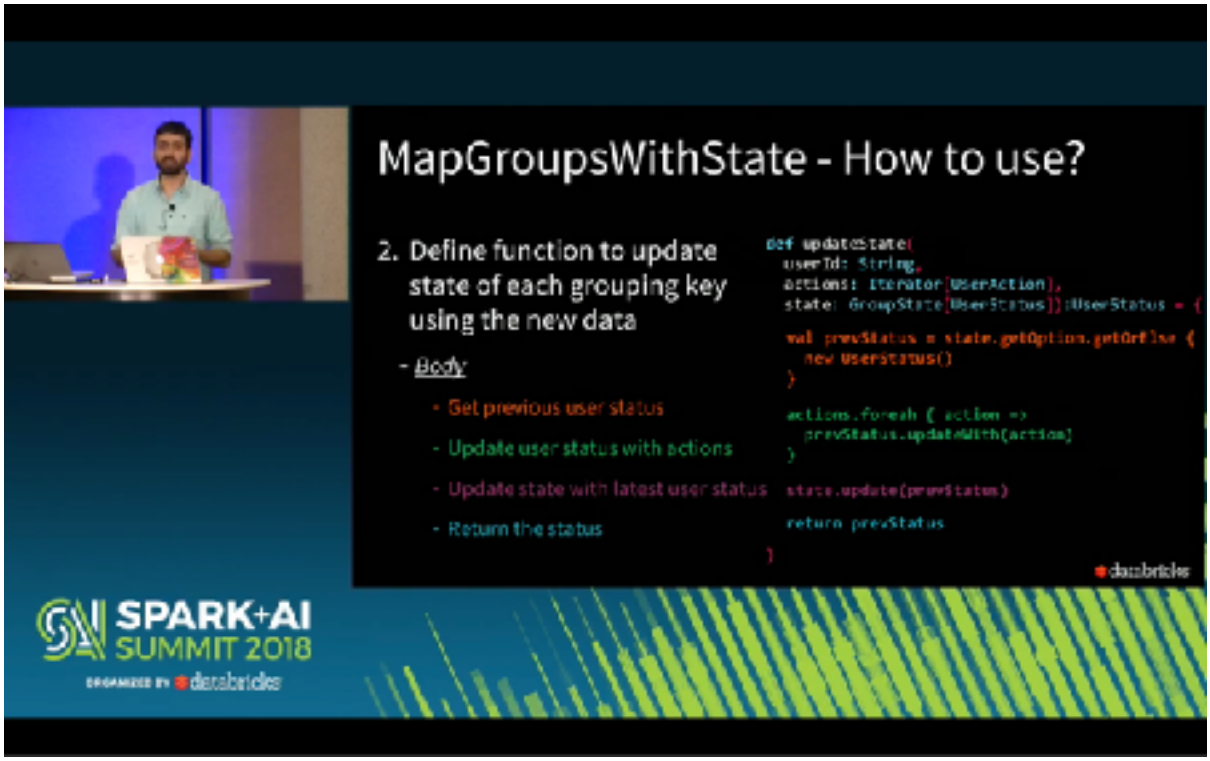
MapGroupsWithState - How to use?

1. Define the data structures
   - *Input event*: UserAction
   - *State data*: UserStatus
   - *Output event*: UserStatus
     (can be different from state)

```
case class UserAction(
    userId: String, action: String)

case class UserStatus(
    userId: String, active: Boolean)
```


MapGroupsWithState - How to use?
2. Define function to update state of each grouping key using the new data
- Input
- Grouping key: userId
- New data: new user actions
- Previous state: previous status of this user
case class UserAction(
userId: String, action: String)
case class UserStatus(
userId: String, active: Boolean)
def updateState(
userId: String,
actions: Iterator[UserAction],
state: GroupState[UserStatus]):UserStatus = {
}
databricks
SPARK+AI SUMMIT 2018
databricks


MapGroupsWithState - How to use?
2. Define function to update state of each grouping key using the new data
- Body
- Get previous user status
- Update user status with actions
- Update state with latest user status
- Return the status
def updateState(
userId: String,
actions: Iterator[UserAction],
state: GroupState[UserStatus]):UserStatus = {
val prevStatus = state.getOption.getOrElse {
new UserStatus()
}
actions.foreach { action =>
prevStatus.updateWith(action)
}
state.update(prevStatus)
return prevStatus
}
databricks
SPARK+AI SUMMIT 2018
databricks

MapGroupsWithState - How to use?

3. Use the user-defined function on a grouped Dataset

```
def updateState(
  userId: String,
  actions: Iterator[UserAction],
  state: GroupState[UserStatus]): UserStatus = {

  // process actions, update and return status

}

userActions
  .groupByKey(_.userId)
  .mapGroupsWithState(updateState)
```


Timeouts

Example: Mark a user as inactive when there is no actions in 1 hour

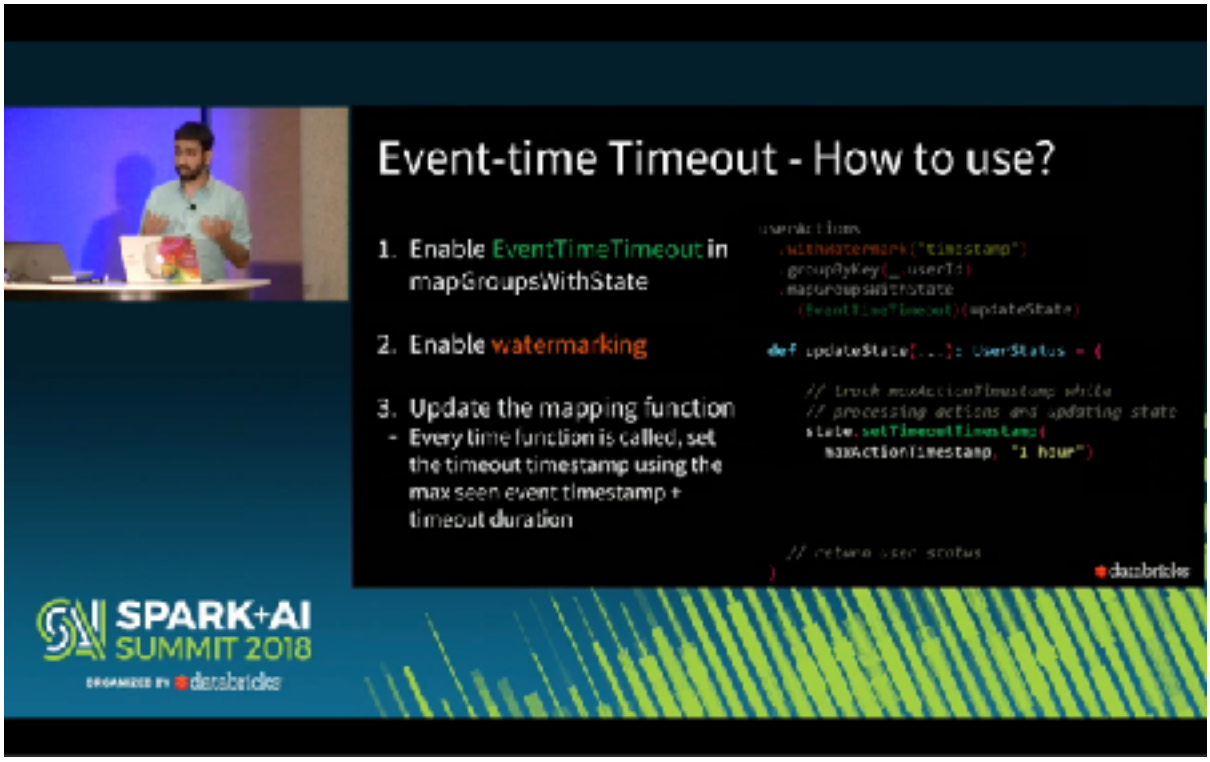
Timeouts: When a group does not get any event for a while, then the function is called for that group with an empty iterator

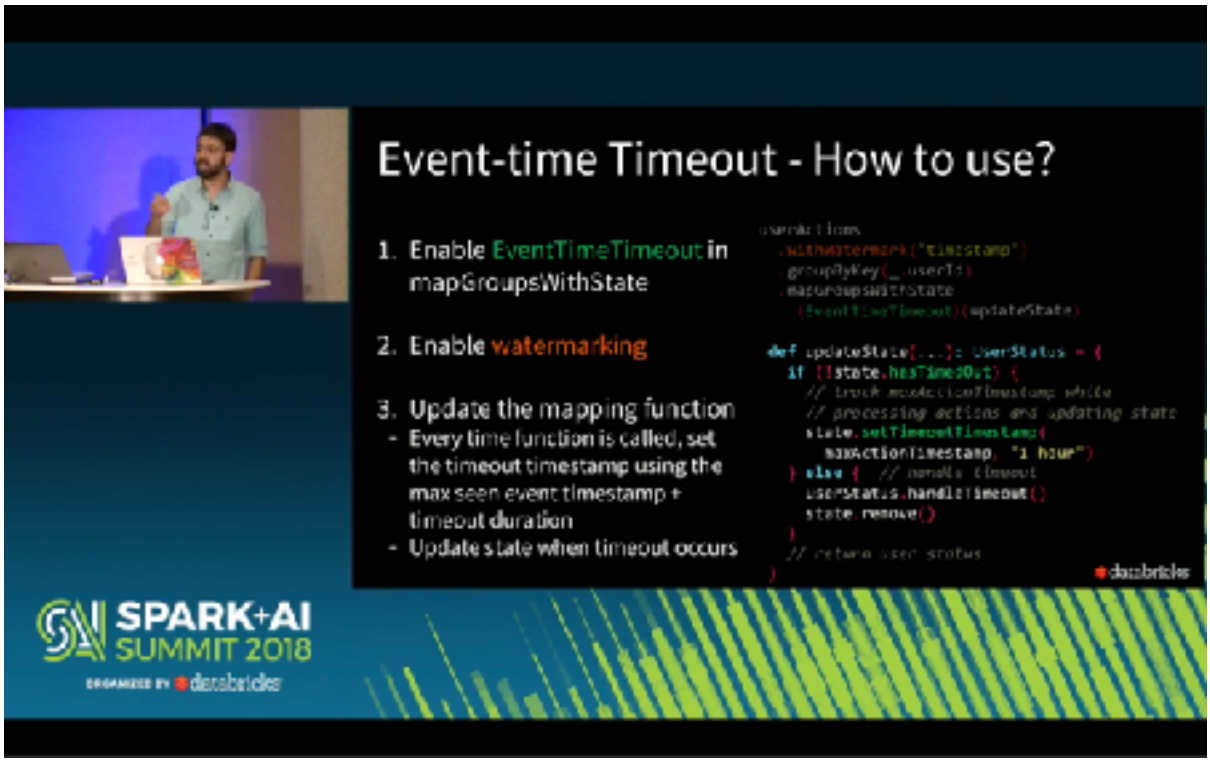
Must specify a global timeout type, and set per-group timeout timestamp/duration

Ignored in a batch queries

```
userActions
  .groupByKey(_.userId)
  .mapGroupsWithState
    (timeoutConf)(updateState)
```

- NoTimeout (default)
- EventTimeTimeout
- ProcessingTimeTimeout


Event-time Timeout - How to use?
1. Enable EventTimeTimeout in mapGroupsWithState
2. Enable watermarking
3. Update the mapping function
- Every time function is called, set the timeout timestamp using the max seen event timestamp + timeout duration
// return user status
SPARK+AI SUMMIT 2018
databricks


Event-time Timeout - How to use?
1. Enable EventTimeTimeout in mapGroupsWithState
2. Enable watermarking
3. Update the mapping function
- Every time function is called, set the timeout timestamp using the max seen event timestamp + timeout duration
- Update state when timeout occurs
// return user status
SPARK+AI SUMMIT 2018
databricks

Event-time Timeout - When?

Watermark is calculated with max event time across all groups

For a specific group, If there is no event till watermark exceeds the timeout timestamp,
Then
Function is called with an empty iterator, and hasTimedOut = true
Else
Function is called with new data, and timeout is disabled
Needs to explicitly set timeout timestamp every time

databricks

SPARK+AI SUMMIT 2018
ORGANIZED BY databricks

Function Output Mode

Function output mode* gives Spark insights into the output from this opaque function

Update Mode - Output events are key-value pairs, each output is updating the value of a key in the result table

Append Mode - Output events are independent rows that being appended to the result table

Allows Spark SQL planner to correctly compose flatMapGroupsWithState with other operations

*Not to be confused with output mode of the query

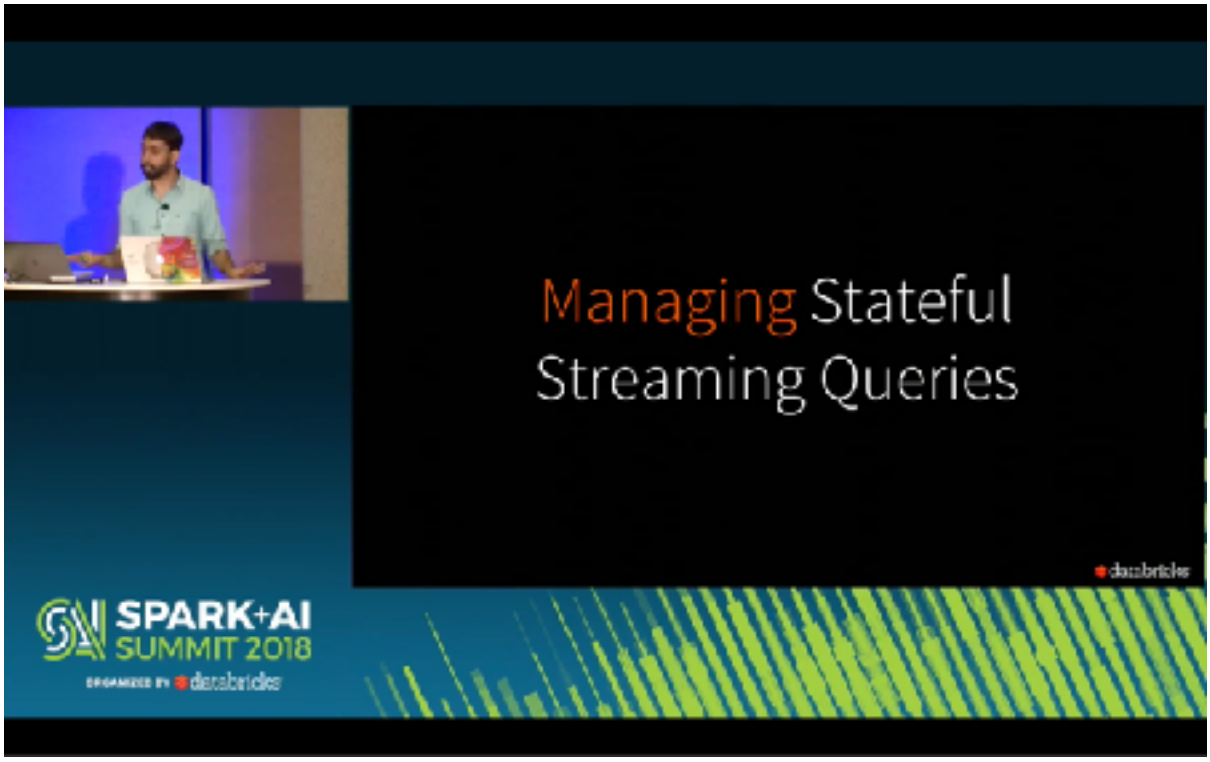
```
userActions
  .groupByKey(_.userId)
  .flatMapGroupsWithState
    (outputMode, timeoutConf)
    (updateState)
```

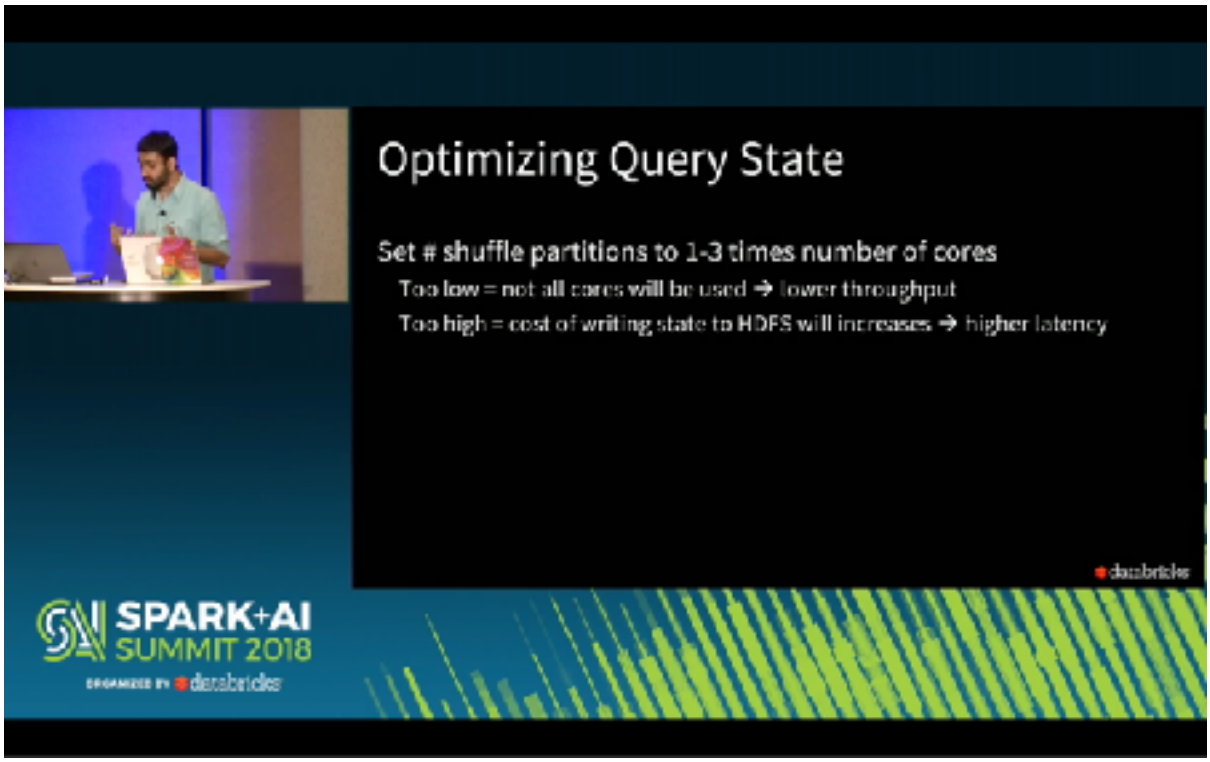
Update Mode

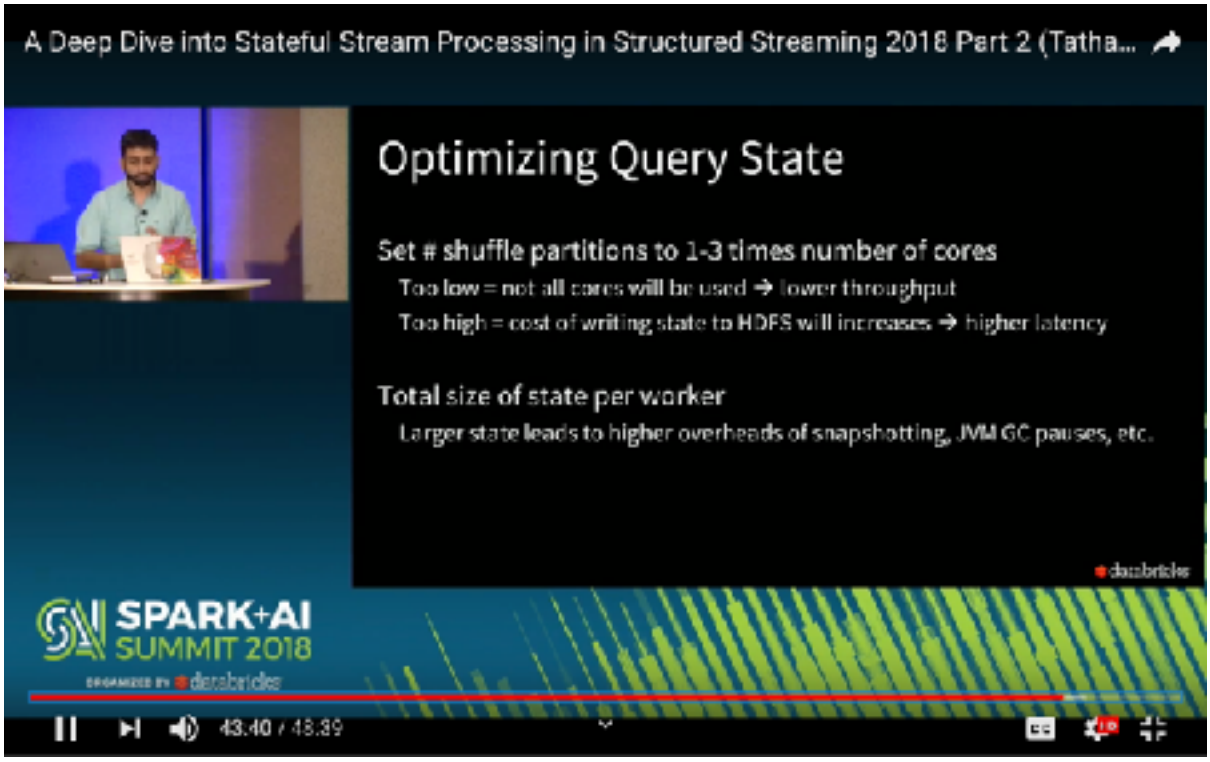
Append Mode

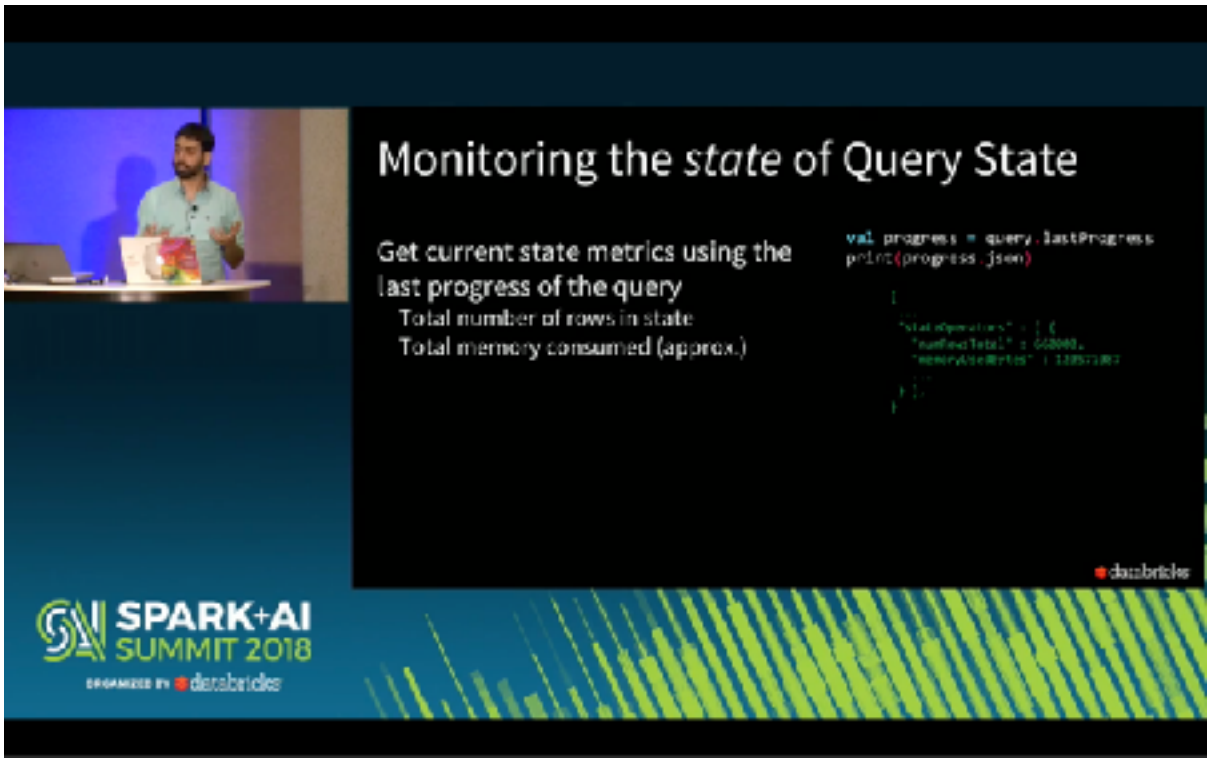
databricks

SPARK+AI SUMMIT 2018
ORGANIZED BY databricks


Managing Stateful
Streaming Queries
SPARK+AI
SUMMIT 2018


Optimizing Query State
Set # shuffle partitions to 1-3 times number of cores
Too low = not all cores will be used → lower throughput
Too high = cost of writing state to HDFS will increases → higher latency
SPARK+AI
SUMMIT 2018


A Deep Dive into Stateful Stream Processing in Structured Streaming 2018 Part 2 (Tatha...
Optimizing Query State
Set # shuffle partitions to 1-3 times number of cores
Too low = not all cores will be used → lower throughput
Too high = cost of writing state to HDFS will increases → higher latency
Total size of state per worker
Larger state leads to higher overheads of snapshotting, JVM GC pauses, etc.
databricks
SPARK+AI SUMMIT 2018
43:40 / 48:39


Monitoring the state of Query State
Get current state metrics using the last progress of the query
Total number of rows in state
Total memory consumed (approx.)
val progress = query.lastProgress
print(progress.json)
databricks
SPARK+AI SUMMIT 2018

Monitoring the *state* of Query State

Get current state metrics using the last progress of the query
- Total number of rows in state
- Total memory consumed (approx.)

```
val progress = query.lastProgress
print(progress.json)
```

```
{
  "stateOperators" : [ {
    "numRowsTotal" : 640000,
    "memoryUsedBytes" : 128571382
  } ],
}
```

Get it asynchronously through StreamingQueryListener API

```
new StreamingQueryListener {
  ...
  def onQueryProgress(
    event: QueryProgressEvent)
}
```


Monitoring the *state* of Query State

Databricks Notebooks integrated with Structured Streaming
- Shows size of state along with other processing metrics

Debugging Query State

SQL metrics in the Spark UI (SQL tab, DAG view) expose more operator-specific stats

Answer questions like

- Is the memory usage skewed?
- Is removing rows slow?
- Is writing checkpoints slow?

databricks

SPARK+AI SUMMIT 2018

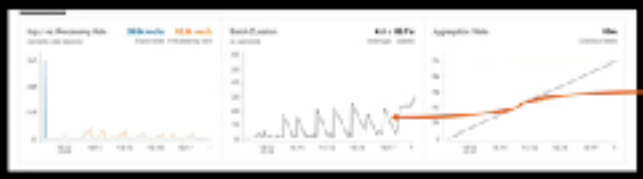
ORGANIZED BY databricks

Managing Very Large State

State data kept on JVM heap

Can have GC issues with *millions* of state keys per worker

Limits depend on the size and complexity of state data structures



databricks

SPARK+AI SUMMIT 2018

ORGANIZED BY databricks

Managing Very Large Statewith RocksDB

In Databricks Runtime, you can store state locally in RocksDB

- Avoids JVM heap, no GC issues with 100 millions state keys per worker
- Local RocksDB snapshot files automatically checkpointed to HDFS
- Same exactly-once fault-tolerant guarantees

Latency capped at 10s

[More info in Databricks Docs]

databricks

SPARK+AI SUMMIT 2018

ORGANIZED BY databricks

Continuous Processing

Millisecond-level latencies with Continuous Processing engine

- Experimental release in Apache Spark 2.3
- Shuffle and stateful operator support coming in future releases

More info:

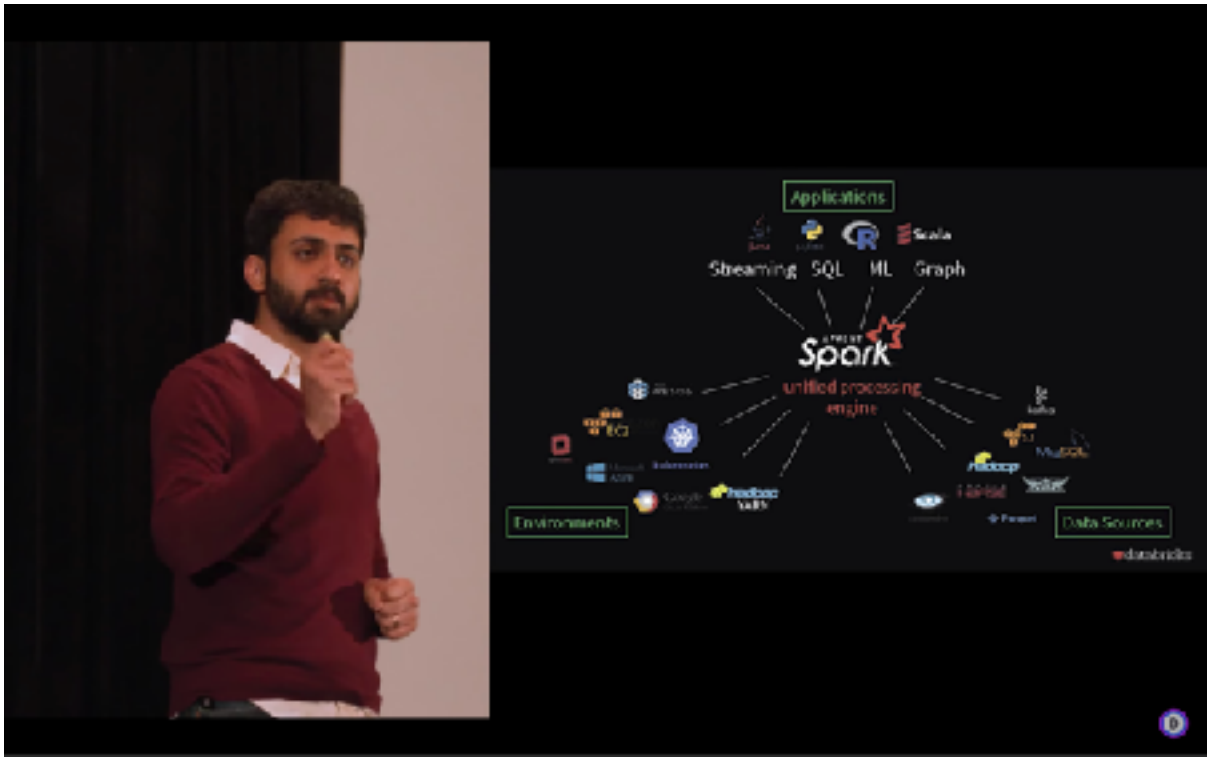
Talk today!

Blog post

databricks

SPARK+AI SUMMIT 2018

ORGANIZED BY databricks

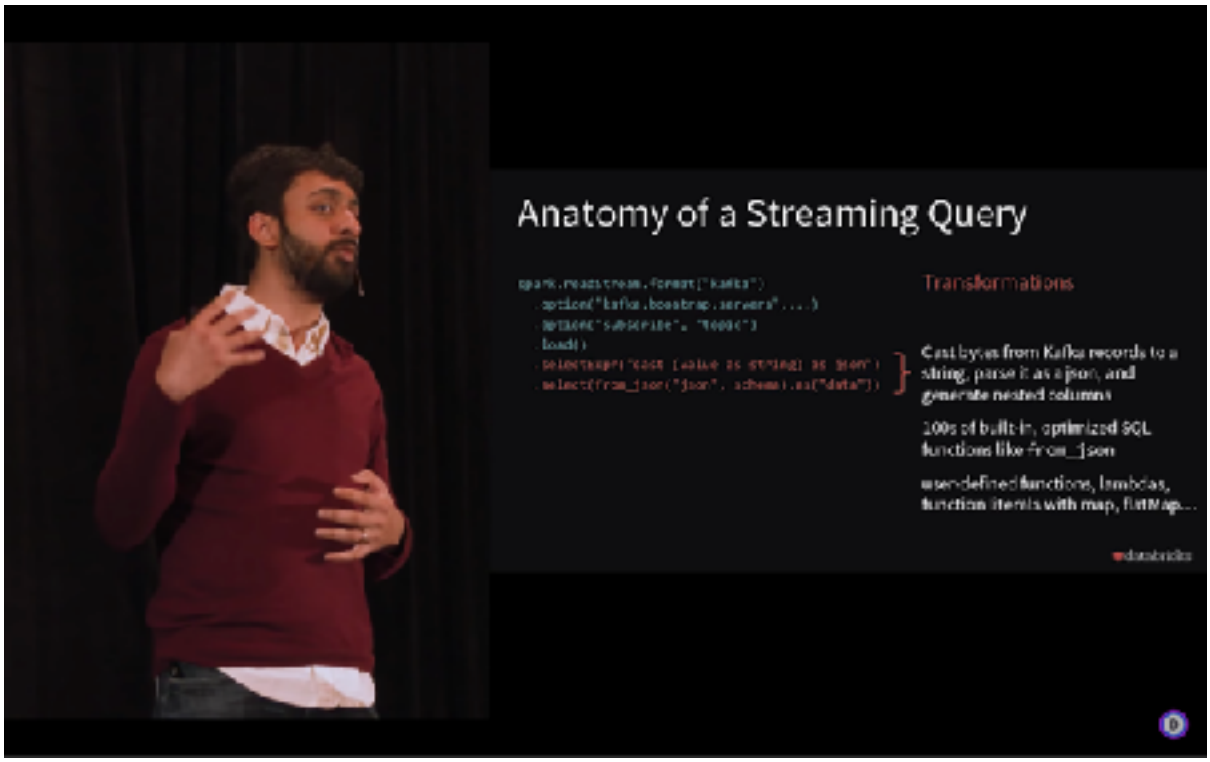

Applications
Streaming
SQL
ML
Graph
Spark
unified processing engine
Environments
Data Sources

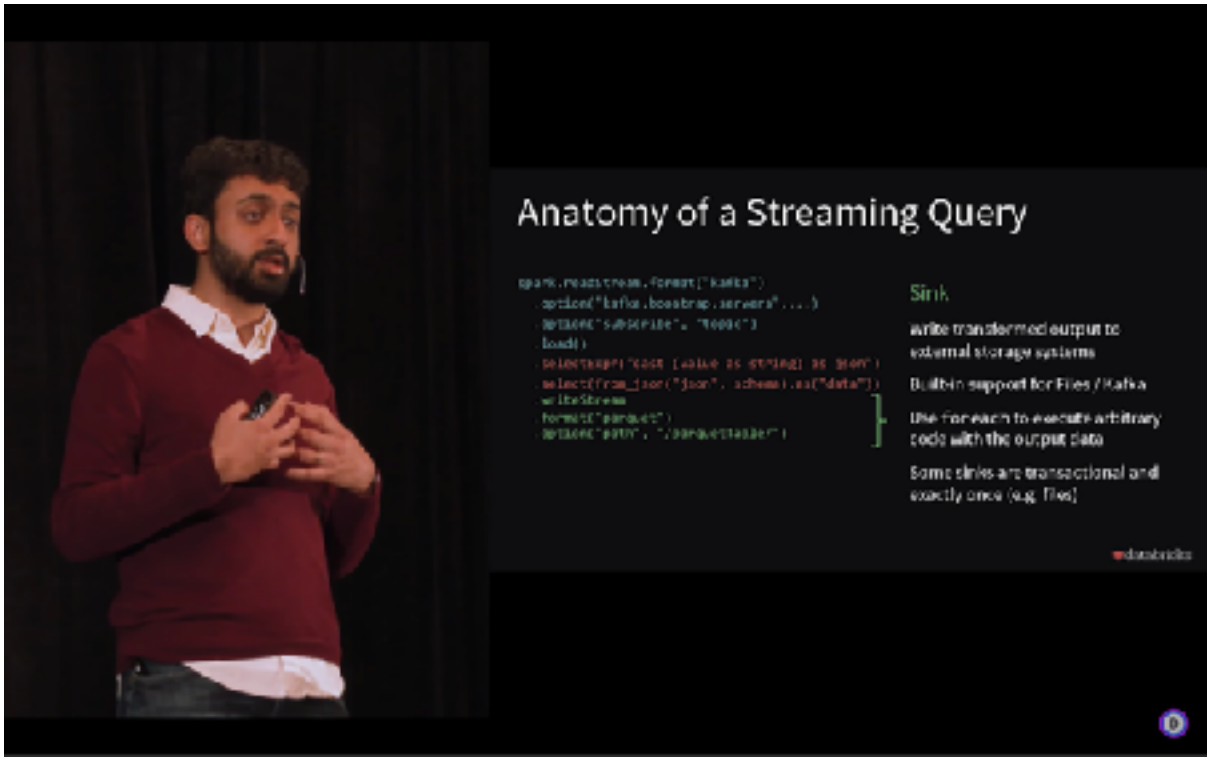

Data Pipelines – 10000ft view
Data Lake
Data Warehouse
unstructured data streams
unstructured data dump
structured data

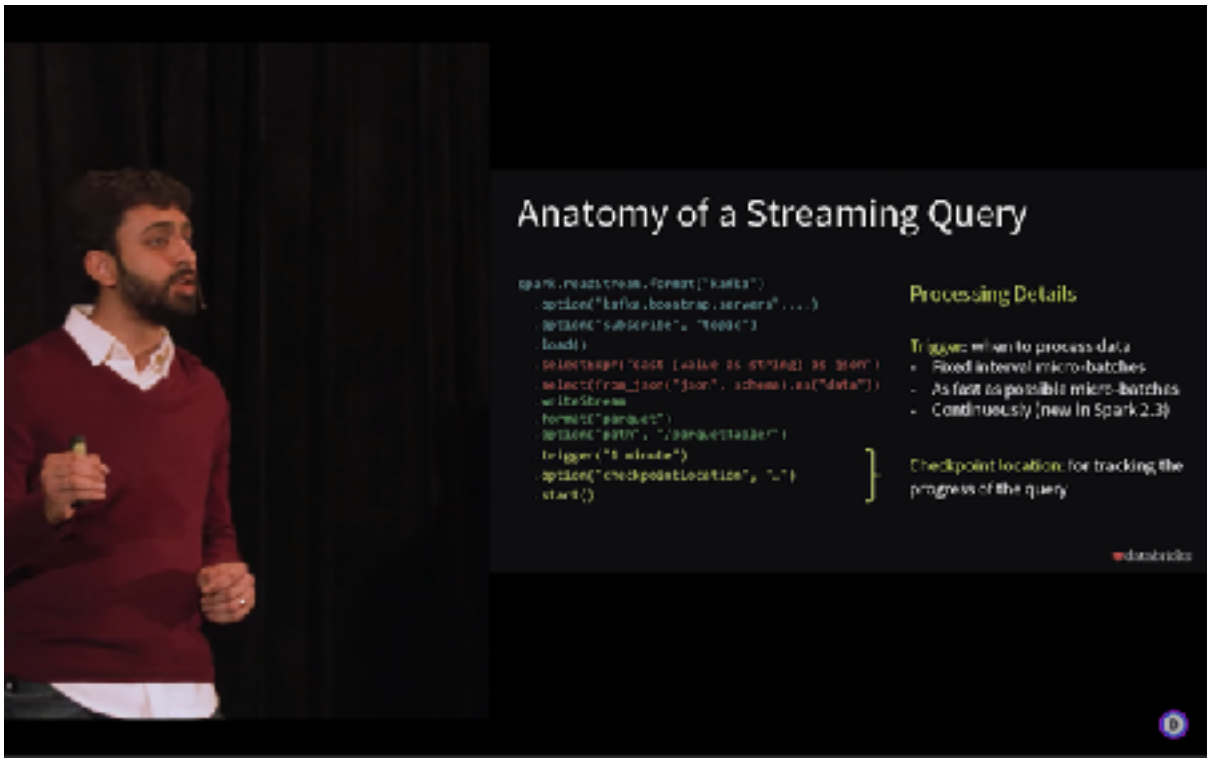

Data Pipeline @ Fortune 100 Company
Trillions of Records
Messy data not ready for analytics
Separate warehouses for each type of analytics
Dump
Complex ETL
DW1
DW2
DW3
Hours of delay in accessing data
Very expensive to scale
Proprietary formats
No advanced analytics (ML)


New Pipeline @ Fortune 100 Company
Spark
STRUCTURED STREAMING
databricks
DELTA
Spark
SQL, ML, STREAMING
Incidence Response
Alerting
Reports
Data usable in minutes/seconds
Easy to scale
Open formats
Enables advanced analytics

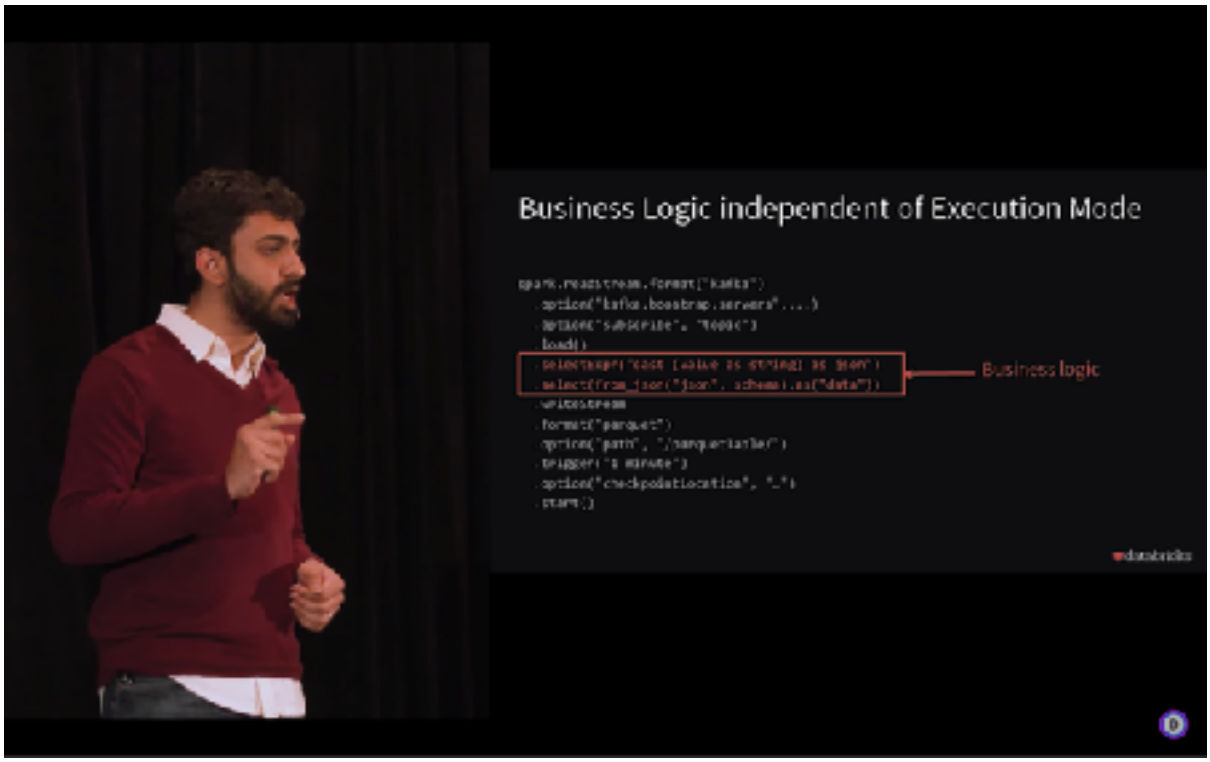

DataFrame/Dataset
SQL
DataFrame
Dataset
Most familiar to BI Analysts
Support SQL-2003, HiveQL
Great for Data Scientists familiar with Pandas, RData.frames
Great for Data Engineers who want compile time type safety
Same semantics, same performance
Choose your hammer for whatever nail you have!
databricks


Anatomy of a Streaming Query
Transformations
Cast bytes from Kafka records to a string, parse it as a json, and generate nested columns
100s of built-in, optimized SQL functions like from_json
user-defined functions, lambdas, function literals with map, flatMap...
databricks


Anatomy of a Streaming Query
spark.readStream.format("kafka")
.option("kafka.bootstrap.servers",...)
.option("subscribe", "topic")
.load()
.selectExpr("cast (value as string) as json")
.select(from_json("json", schema).as("data"))
.writeStream
.format("parquet")
.option("path", "/parquetTable/")
Sink
write transformed output to external storage systems
Built-in support for Files / Kafka
Use foreach to execute arbitrary code with the output data
Some sinks are transactional and exactly once (e.g. files)
databricks


Anatomy of a Streaming Query
spark.readStream.format("kafka")
.option("kafka.bootstrap.servers",...)
.option("subscribe", "topic")
.load()
.selectExpr("cast (value as string) as json")
.select(from_json("json", schema).as("data"))
.writeStream
.format("parquet")
.option("path", "/parquetTable/")
.trigger("1 minute")
.option("checkpointLocation", "...")
.start()
Processing Details
Trigger: when to process data
- Fixed interval micro-batches
- As fast as possible micro-batches
- Continuously (new in Spark 2.3)
Checkpoint location: for tracking the progress of the query
databricks

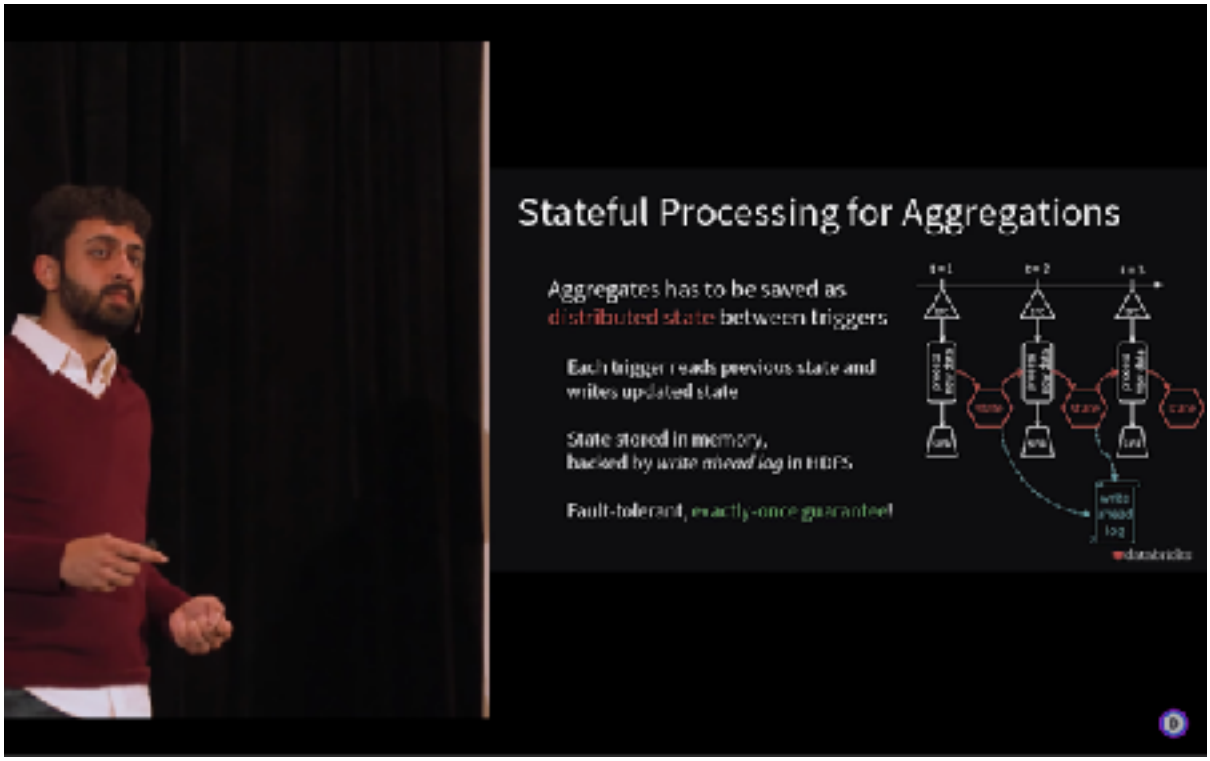

Spark automatically streamifies!
DataFrames, Datasets, SQL
Logical Plan
Optimized Plan
Series of Incremental Execution Plans
Spark SQL converts batch-like query to a series of incremental execution plans operating on new batches of data
databricks


Business Logic independent of Execution Mode
spark.readStream.format("kafka")
.option("kafka.bootstrap.servers",...)
.option("subscribe", "topic")
.load()
.selectExpr('cast (value as string) as json')
.select(from_json("json", schema).as("data"))
Business logic
.writeStream
.format("parquet")
.option("path", "/parquetable")
.trigger("1 minute")
.option("checkpointLocation", "...")
.start()
databricks


Business Logic independent of Execution Mode
Business logic remains unchanged
Peripheral code decides whether it's a batch or a streaming query
databricks


Business Logic independent of Execution Mode
Batch
Micro-batch
Streaming
Continuous**
Streaming
high latency
low latency
(seconds)
ultra-low latency
(milliseconds)
execute on-demand
efficient resource allocation
static resource allocation
high throughput
high throughput
databricks


Event time Aggregations
Windowing is just another type of grouping in Struct. Streaming
number of records every hour
avg signal strength of each device every 10 mins
Support UDAFs!
databricks


Stateful Processing for Aggregations
Aggregates has to be saved as distributed state between triggers
Each trigger reads previous state and writes updated state
State stored in memory, backed by write ahead log in HDFS
Fault-tolerant, exactly-once guaranteed
databricks

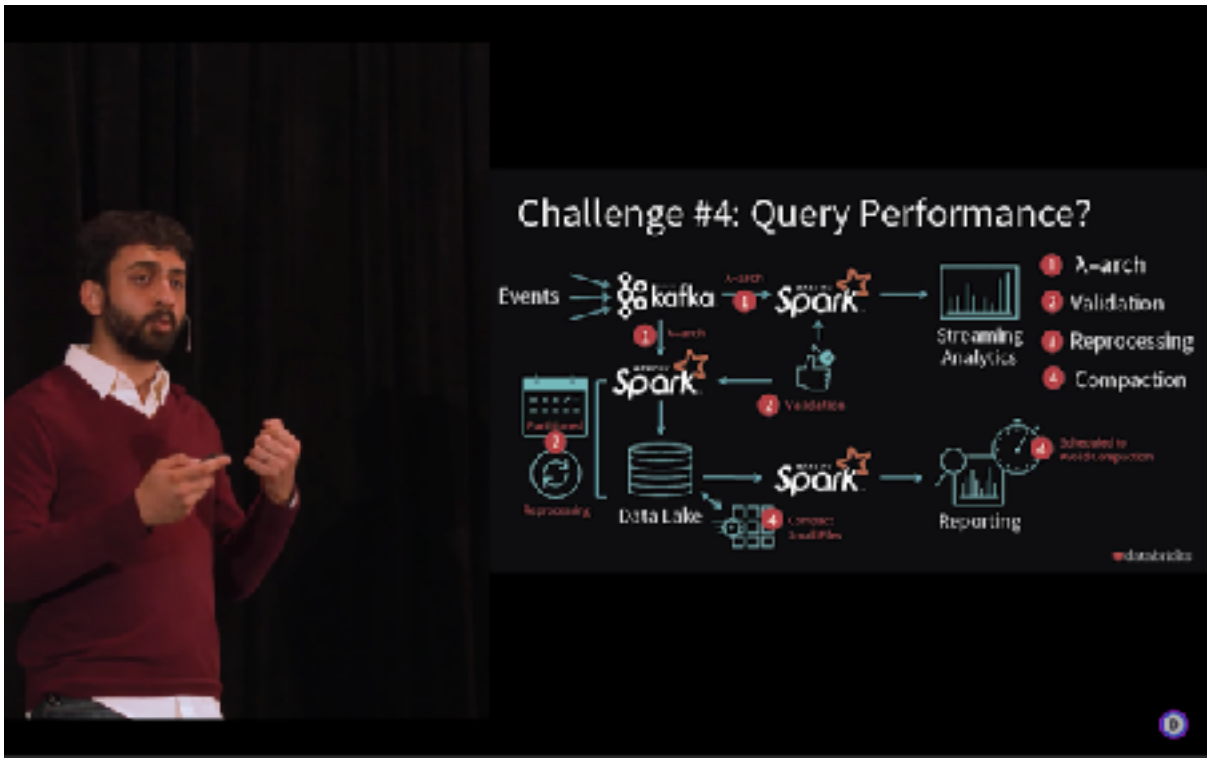

Evolution of a Cutting-Edge Data Pipeline
Events
kafka
?
Streaming Analytics
Data Lake
Reporting


Evolution of a Cutting-Edge Data Pipeline
Events
kafka
Spark
Streaming Analytics
Data Lake
Reporting


Challenge #1: Historical Queries?
Events
kafka
Spark
Streaming Analytics
1 λ-arch
Spark
Delta Lake
Spark
Reporting

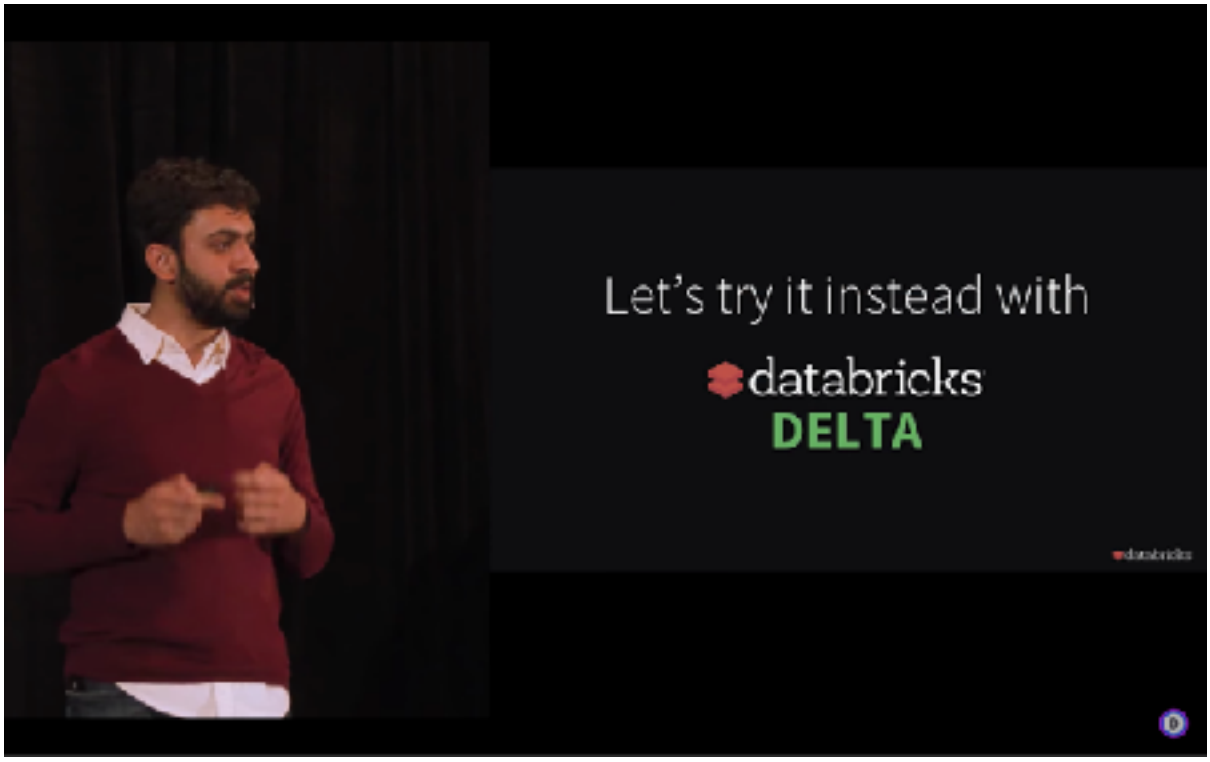

Challenge #2: Messy Data?
Events
kafka
Spark
Streaming Analytics
1 λ-arch
2 Validation
Spark
2 Validation
Delta Lake
Spark
Reporting


Challenge #3: Mistakes and Failures?
Events
kafka
Spark
Streaming Analytics
1 λ-arch
2 Validation
4 Reprocessing
Spark
2 Validation
Delta Lake
Spark
Reporting


Challenge #4: Query Performance?
Events
kafka
Spark
Streaming Analytics
1 λ-arch
2 Validation
4 Reprocessing
4 Compaction
Spark
2 Validation
Delta Lake
Spark
Reporting

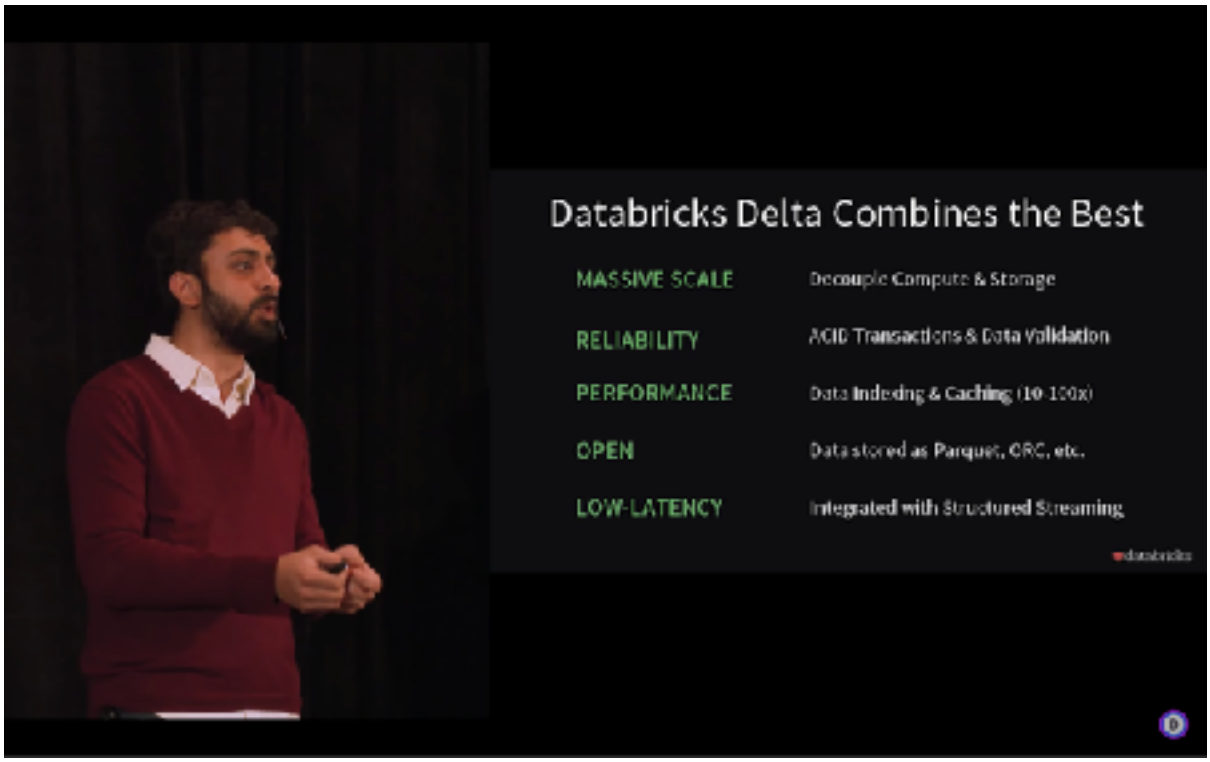

Real-Time Data Pipelines Made Easy with Structured Streaming in Apache Spark | DataEngConf SF '18
Challenge #4: Query Performance?
Events
kafka
Spark
Streaming Analytics
1 λ-arch
2 Validation
3 Reprocessing
4 Compaction
Data Lake
Reporting

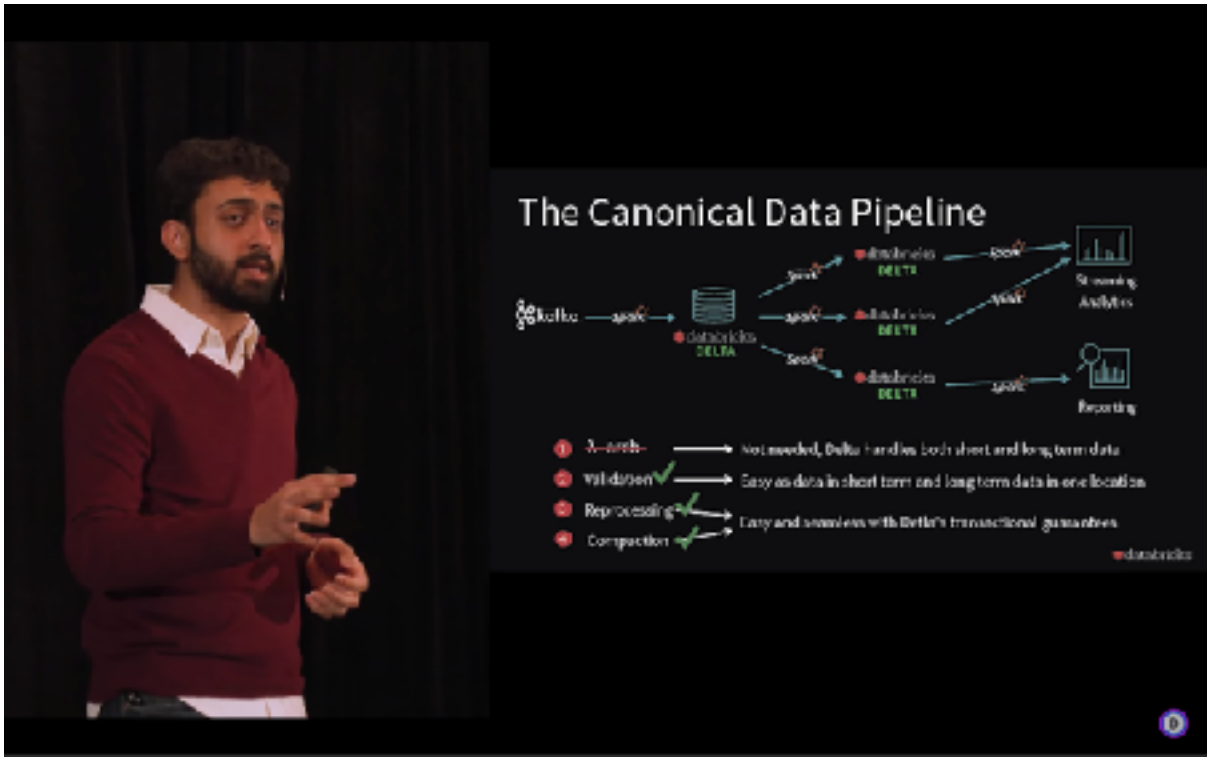

Real-Time Data Pipelines Made Easy with Structured Streaming in Apache Spark | DataEngConf SF '18
Challenge #4: Query Performance?
Events
kafka
Spark
Streaming Analytics
1 λ-arch
2 Validation
3 Reprocessing
4 Compaction
Data Lake
Reporting


Let's try it instead with
databricks
DELTA


Real-Time Data Pipelines Made Easy with Structured Streaming in Apache Spark | DataEngConf SF '18
databricks
DELTA
The SCALE of data lake
The RELIABILITY & PERFORMANCE of data warehouse
The LOW-LATENCY of streaming


THE GOOD
OF DATA WAREHOUSES
• Pristine Data
• Transactional Reliability
• Fast Queries
THE GOOD
OF DATA LAKES
• Massive scale on cloud storage
• Open Formats (Parquet, ORC)
• Predictions (ML) & Streaming
databricks

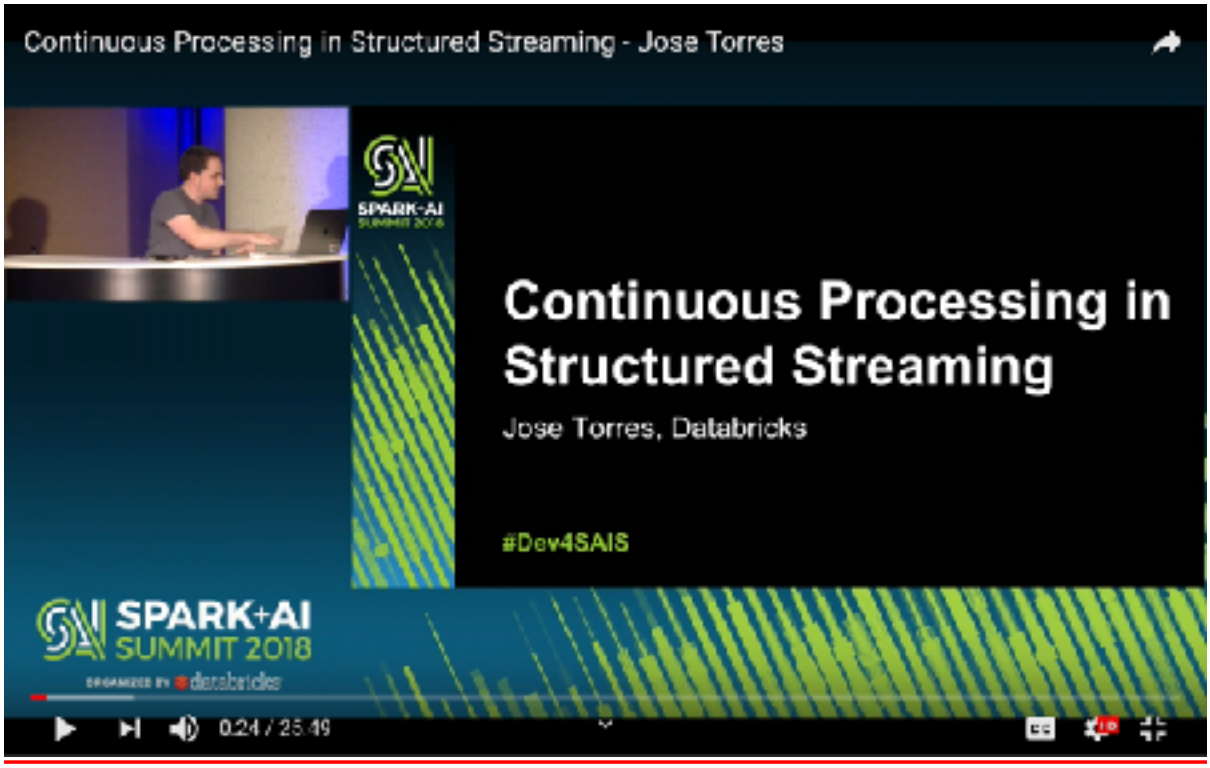

Databricks Delta Combines the Best
MASSIVE SCALE
Decouple Compute & Storage
RELIABILITY
ACID Transactions & Data Validation
PERFORMANCE
Data Indexing & Caching (10-100x)
OPEN
Data stored as Parquet, ORC, etc.
LOW-LATENCY
Integrated with Structured Streaming
databricks

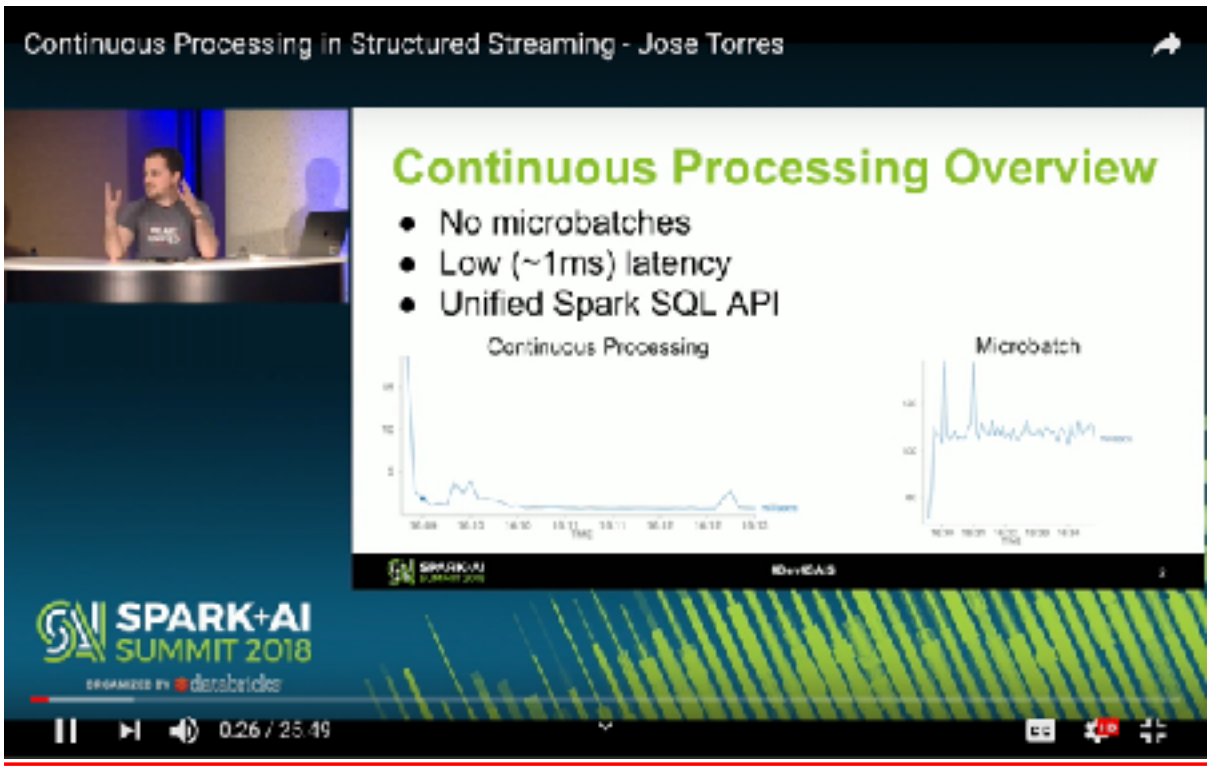

The Canonical Data Pipeline
kafka
databricks DELTA
Streaming Analytics
Reporting
Validation
Reprocessing
Compaction
databricks

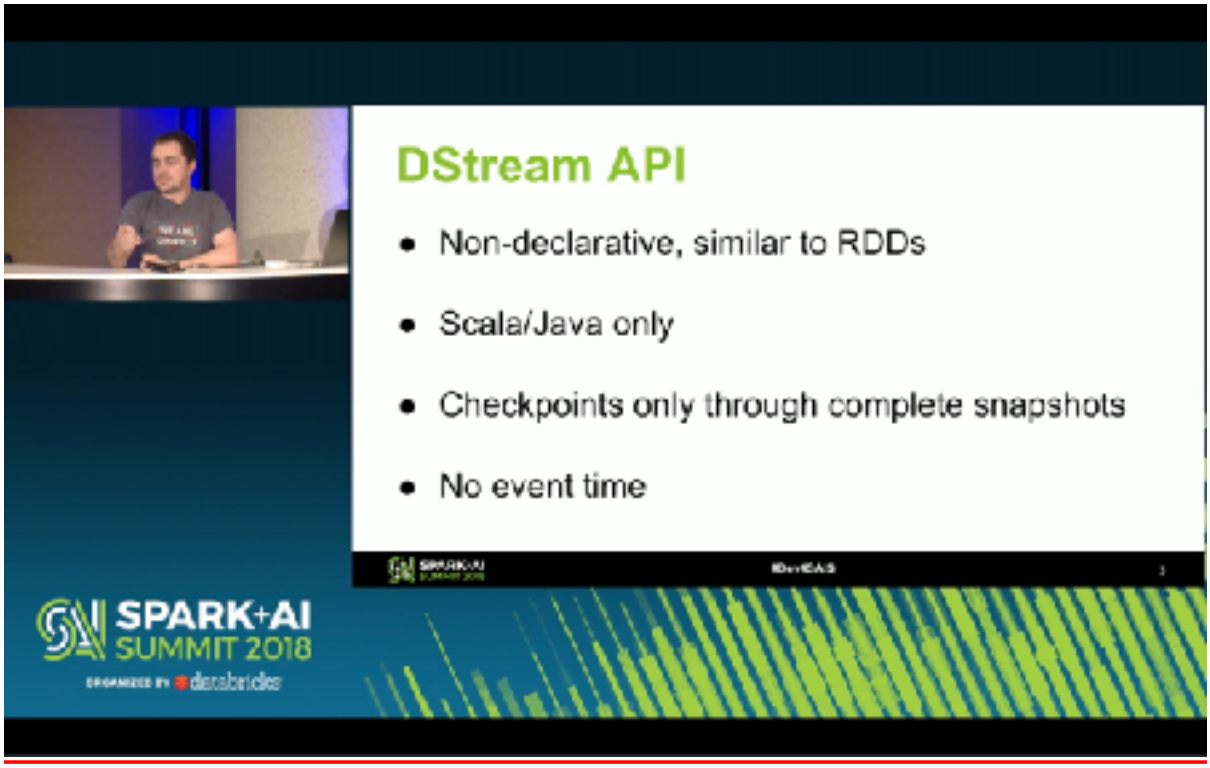

Accelerate Innovation with Databricks
Unified Analytics Platform
DATABRICKS COLLABORATIVE NOTEBOOKS
DATABRICKS RUNTIME
DATABRICKS DELTA
Spark
DATABRICKS MANAGED SERVICE
AWS
AZURE
databricks

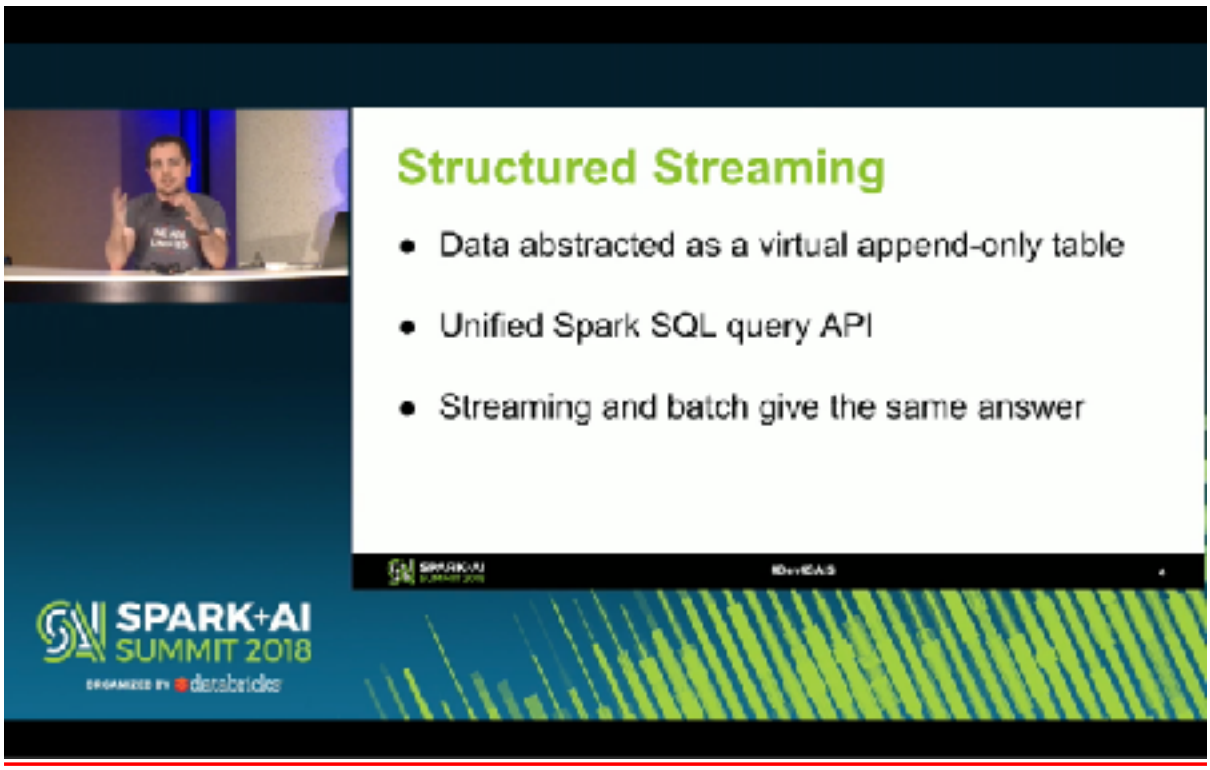
Spark - Continuous Processing Engine in Structured Streaming

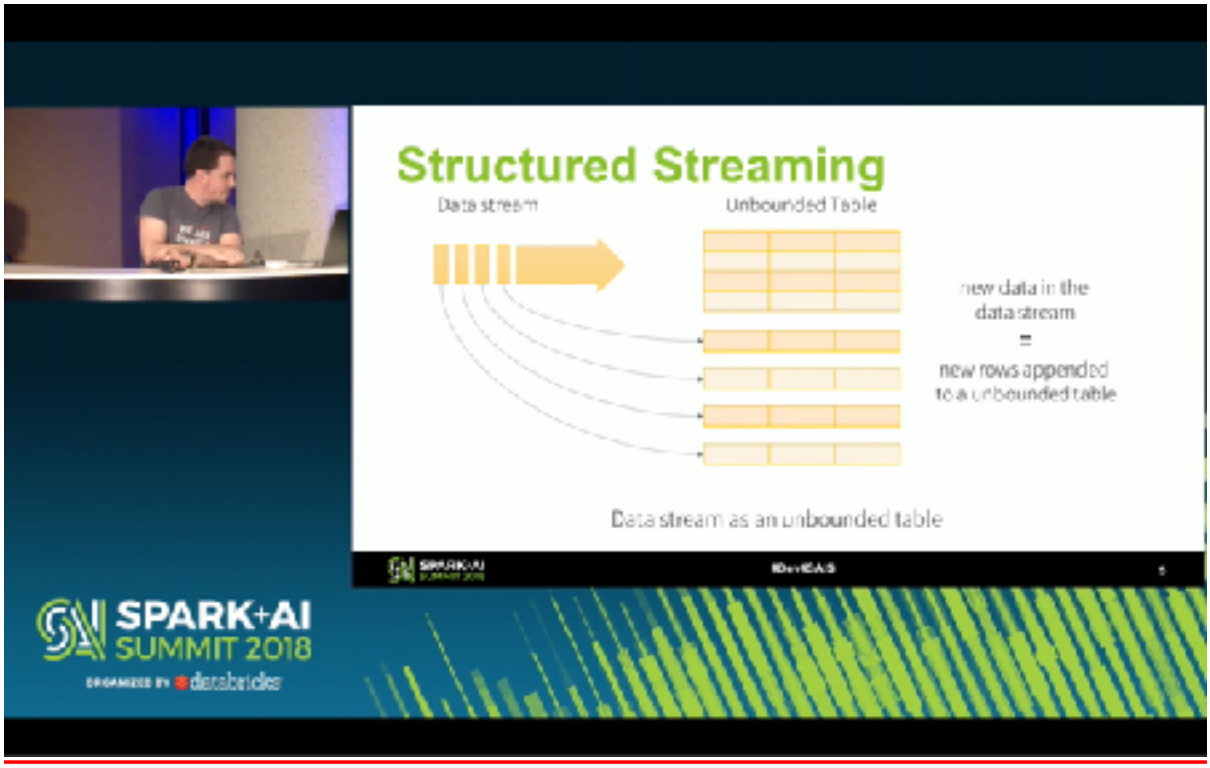
https://www.youtube.com/watch?v=tPaOEZewaek

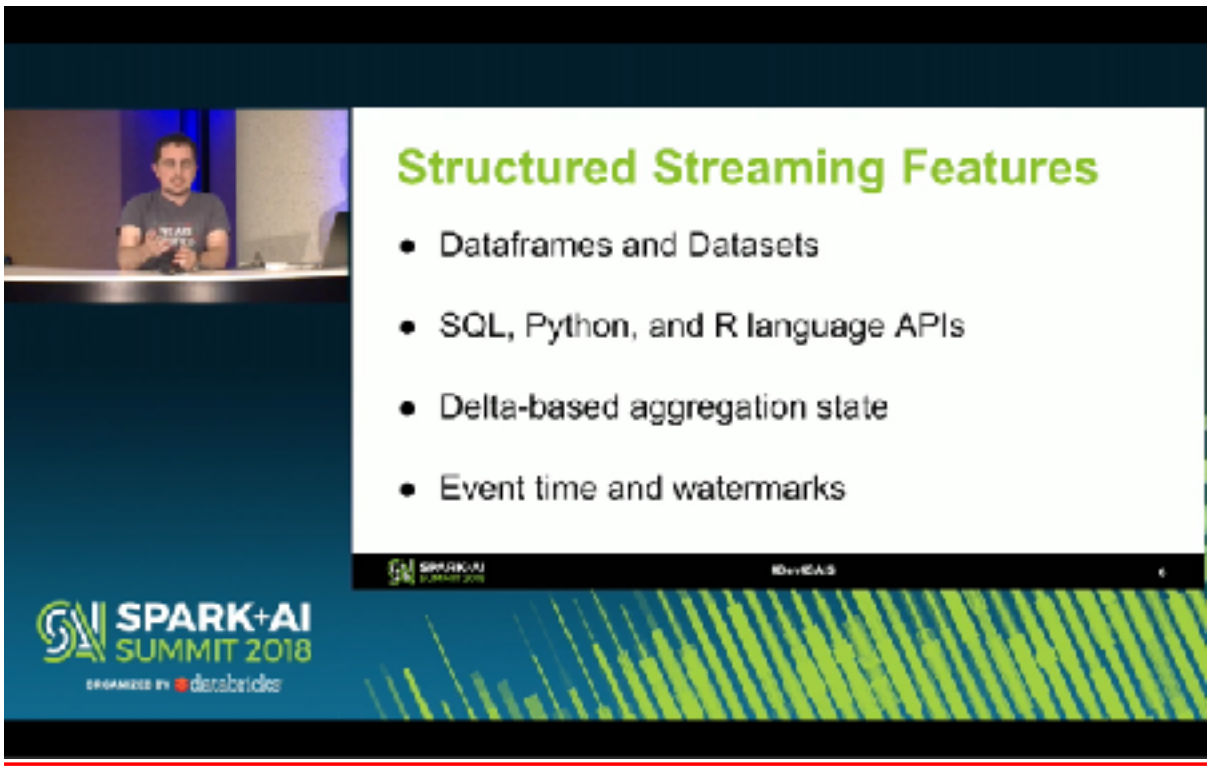

Continuous Processing in Structured Streaming - Jose Torres
SPARK+AI SUMMIT 2018
Continuous Processing in Structured Streaming
Jose Torres, Databricks
#Dev4SAIS
SPARK+AI SUMMIT 2018
0:24 / 25:49

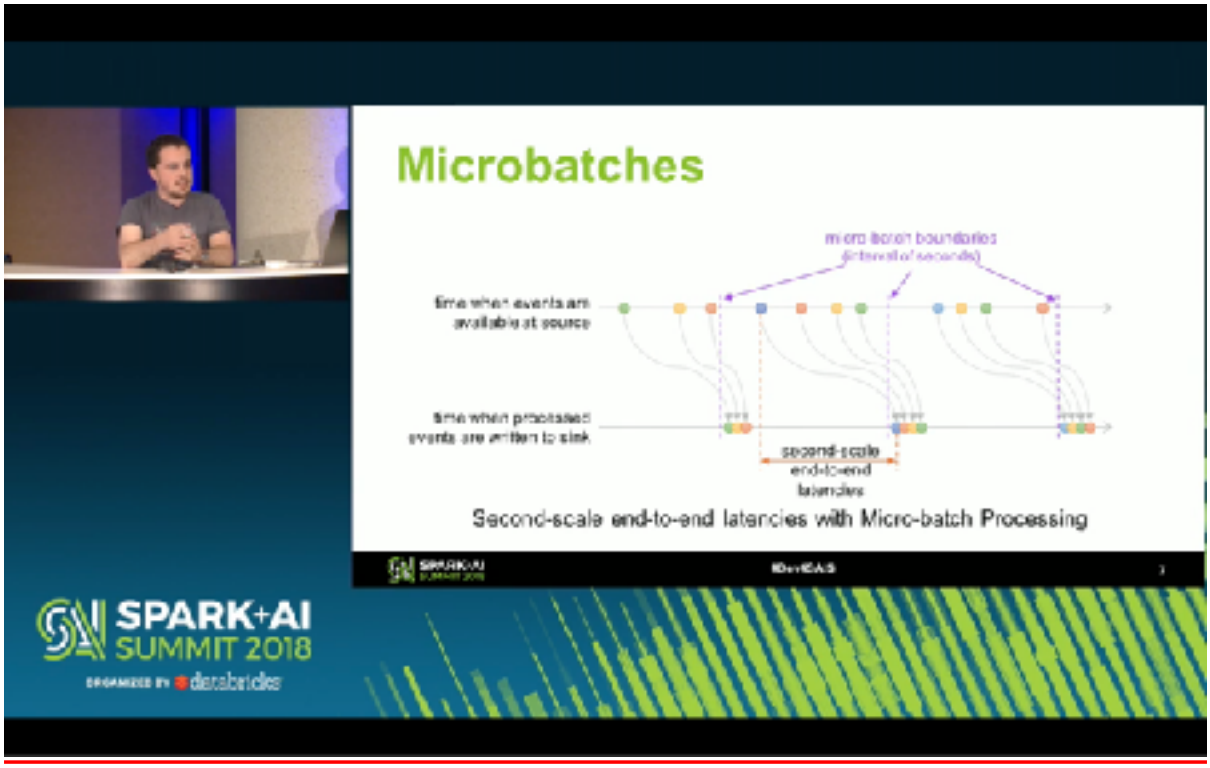

Continuous Processing in Structured Streaming - Jose Torres
Continuous Processing Overview
No microbatches
Low (~1ms) latency
Unified Spark SQL API
Continuous Processing
Microbatch
SPARK+AI SUMMIT 2018
0:26 / 25:49

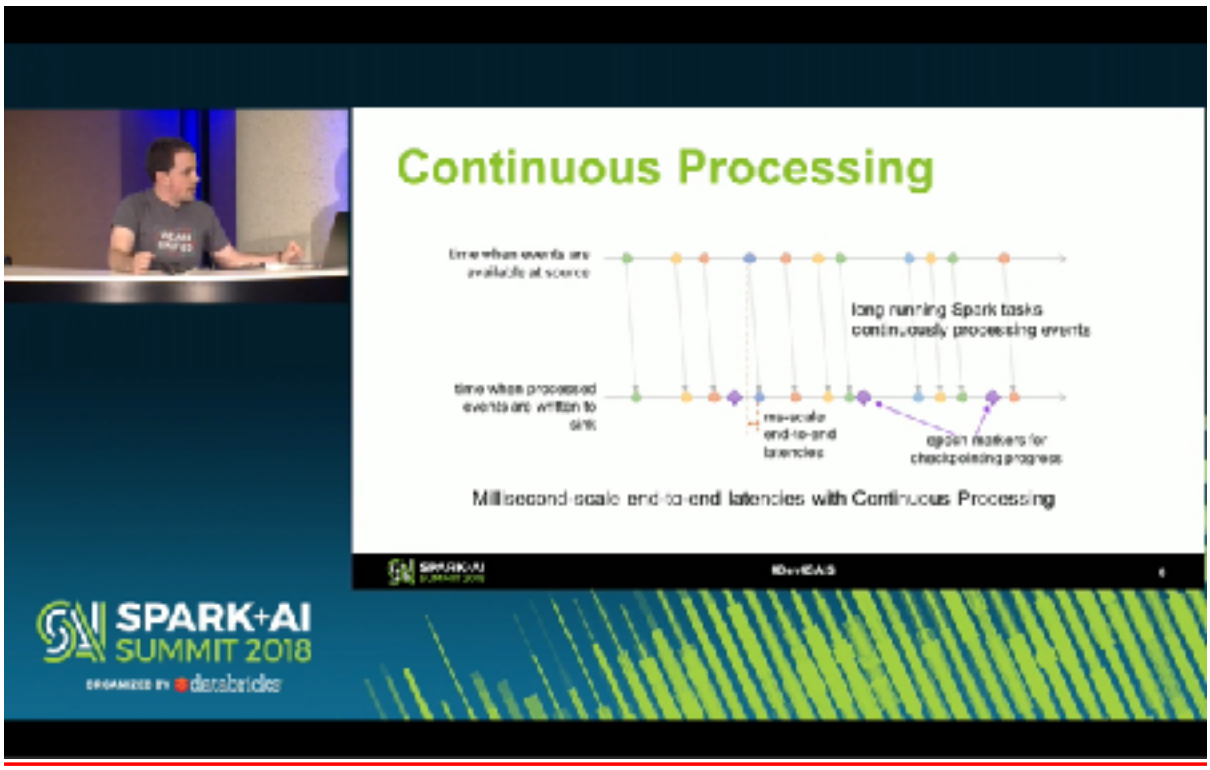

DStream API
• Non-declarative, similar to RDDs
• Scala/Java only
• Checkpoints only through complete snapshots
• No event time
SPARK+AI SUMMIT 2018

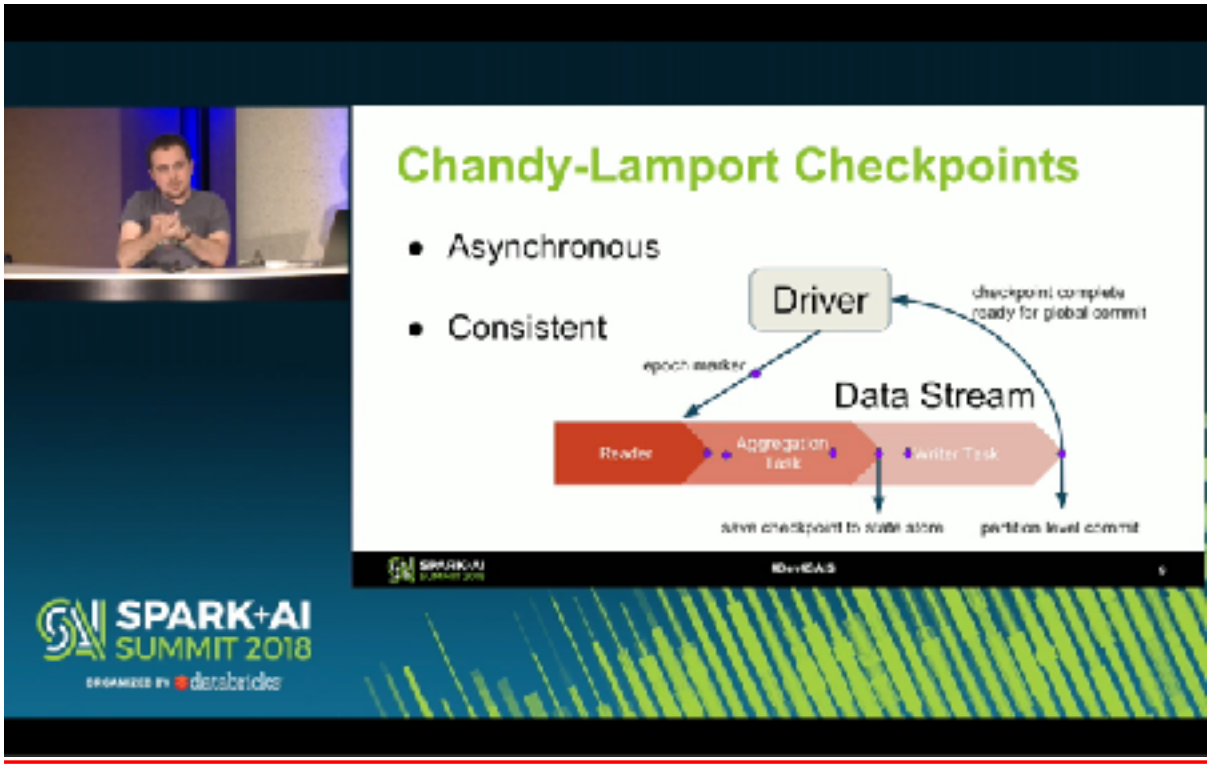

Structured Streaming
• Data abstracted as a virtual append-only table
• Unified Spark SQL query API
• Streaming and batch give the same answer
SPARK+AI SUMMIT 2018

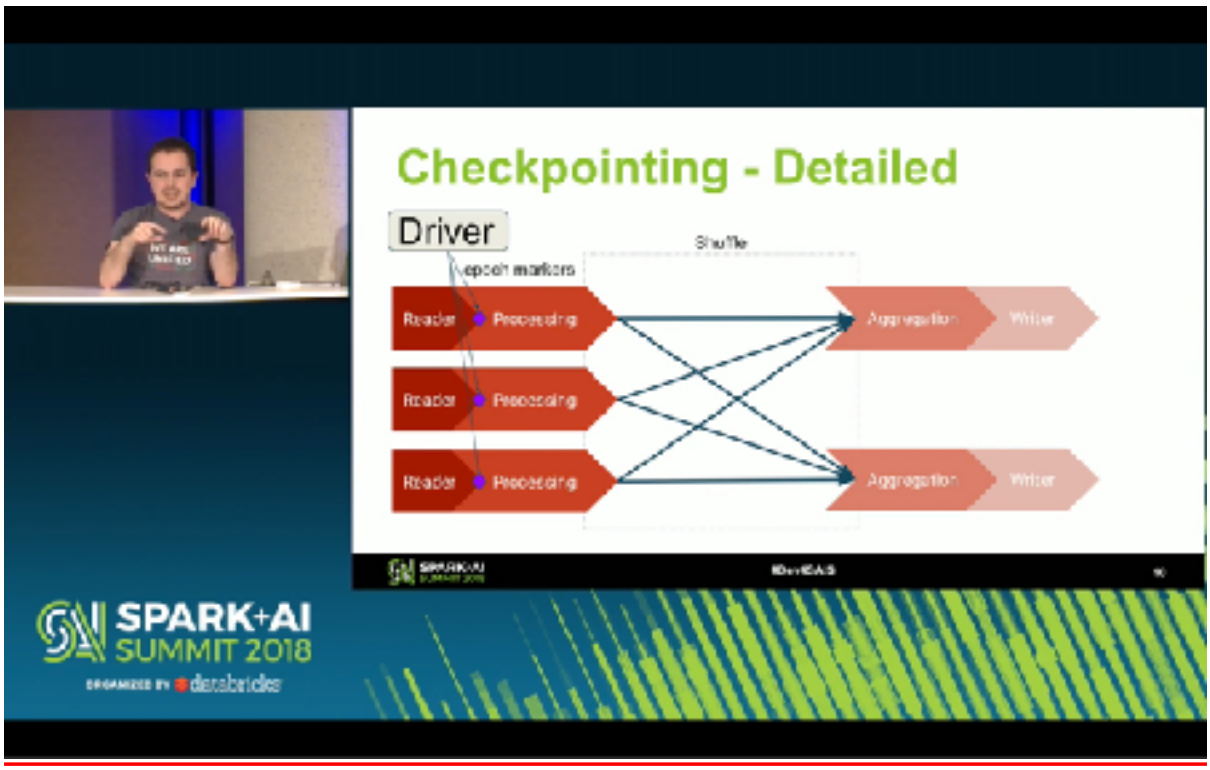

Structured Streaming
Data stream
Unbounded Table
new data in the data stream
=
new rows appended to a unbounded table
Data stream as an unbounded table
SPARK+AI SUMMIT 2018

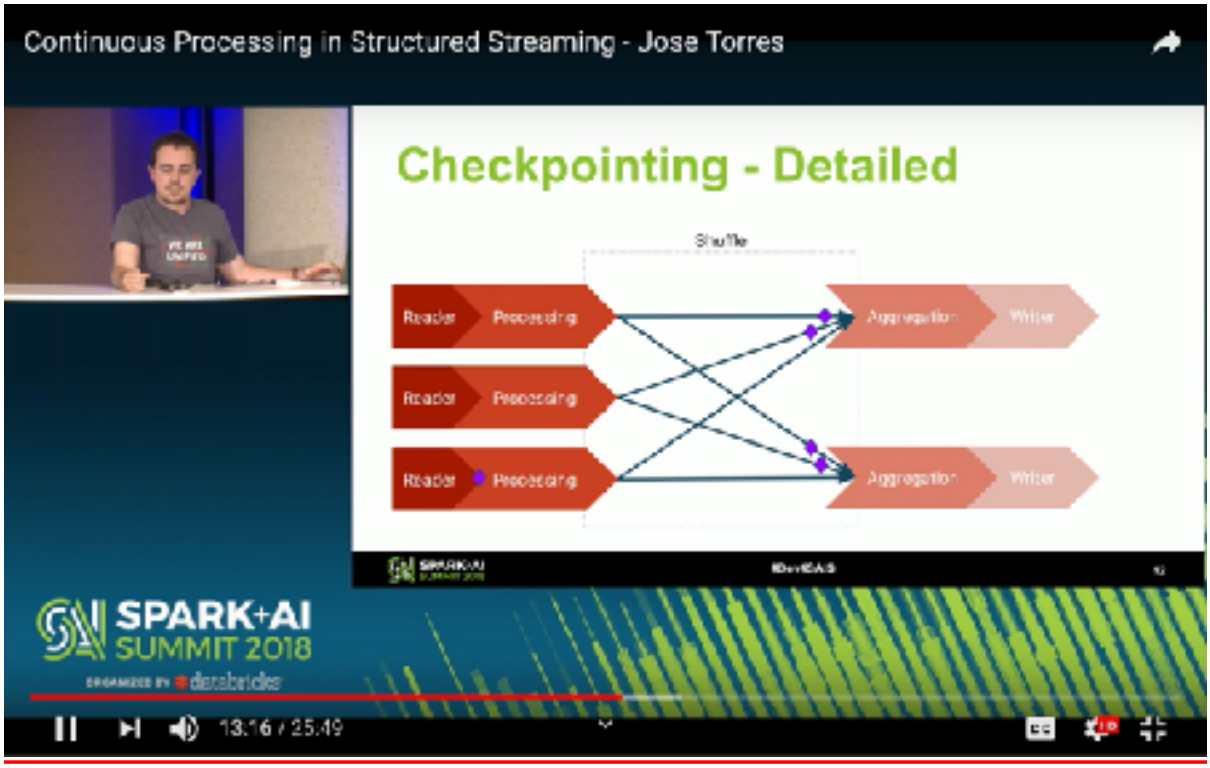

Structured Streaming Features
• Dataframes and Datasets
• SQL, Python, and R language APIs
• Delta-based aggregation state
• Event time and watermarks
SPARK+AI SUMMIT 2018

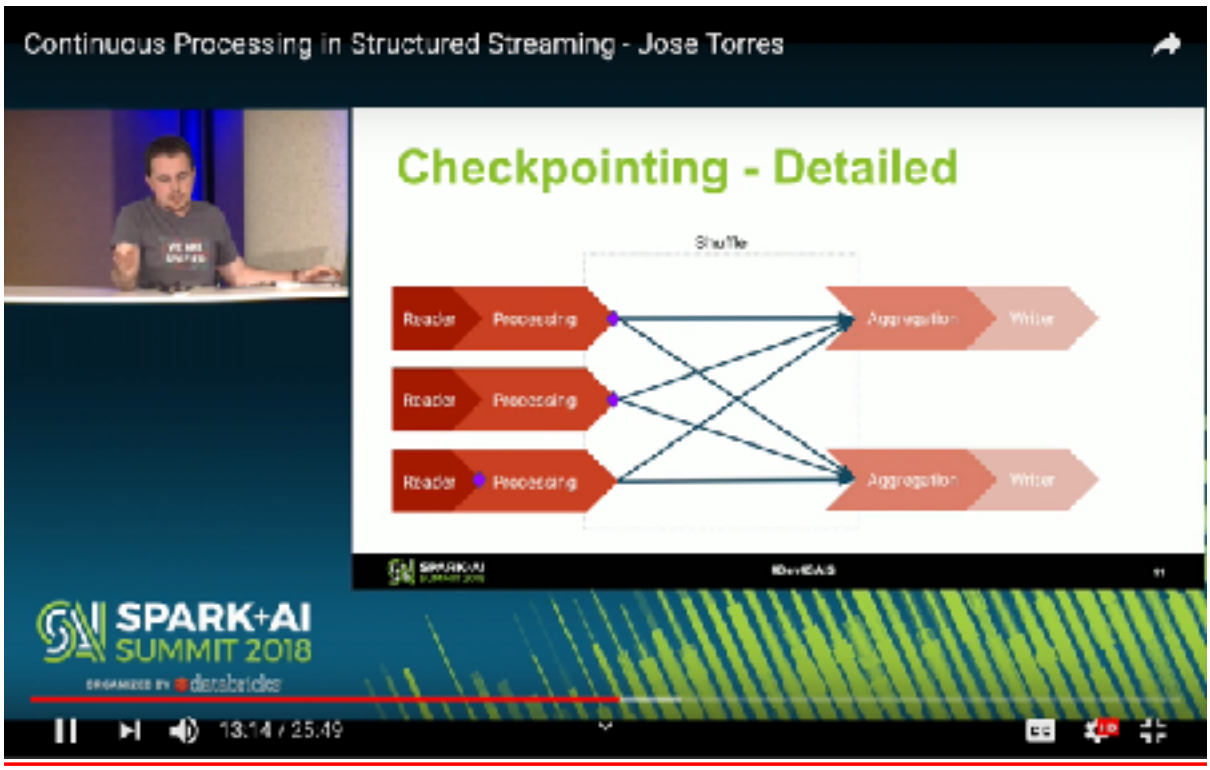

Microbatches
micro-batch boundaries
(interval of seconds)
time when events are available at source
time when processed events are written to sink
second-scale end-to-end latencies
Second-scale end-to-end latencies with Micro-batch Processing
SPARK+AI SUMMIT 2018
ORGANIZED BY databricks

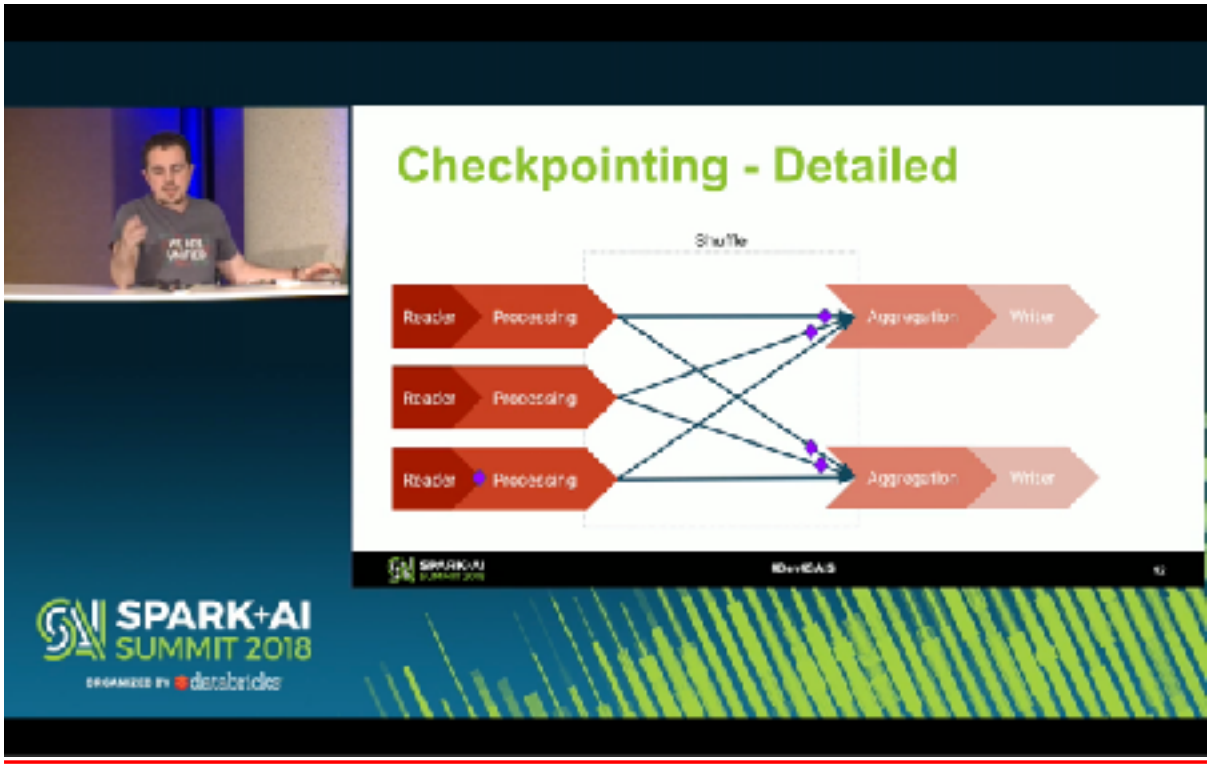

Continuous Processing
time when events are available at source
long running Spark tasks continuously processing events
time when processed events are written to sink
ms-scale end-to-end latencies
epoch markers for checkpointing progress
Millisecond-scale end-to-end latencies with Continuous Processing
SPARK+AI SUMMIT 2018
ORGANIZED BY databricks

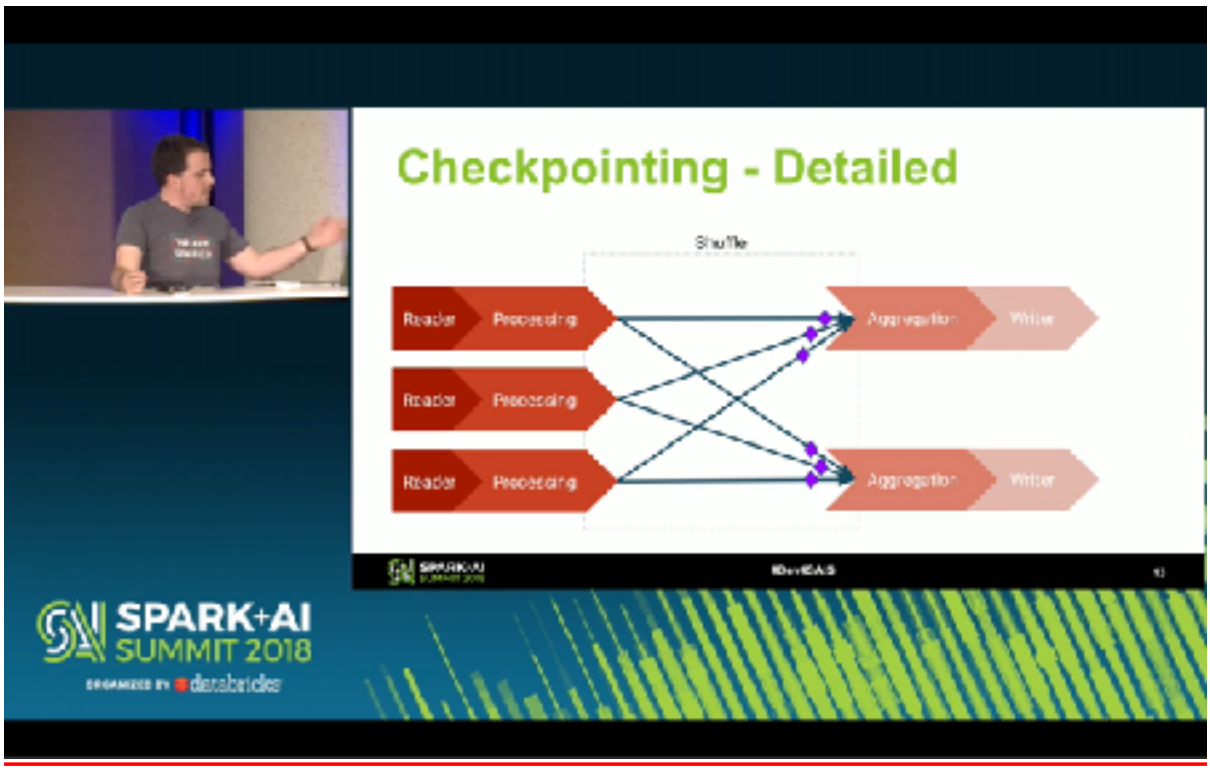

Chandy-Lamport Checkpoints
Asynchronous
Consistent
Driver
checkpoint complete
ready for global commit
epoch marker
Data Stream
Reader
Aggregation Task
Writer Task
save checkpoint to state store
partition level commit
SPARK+AI SUMMIT 2018
databricks

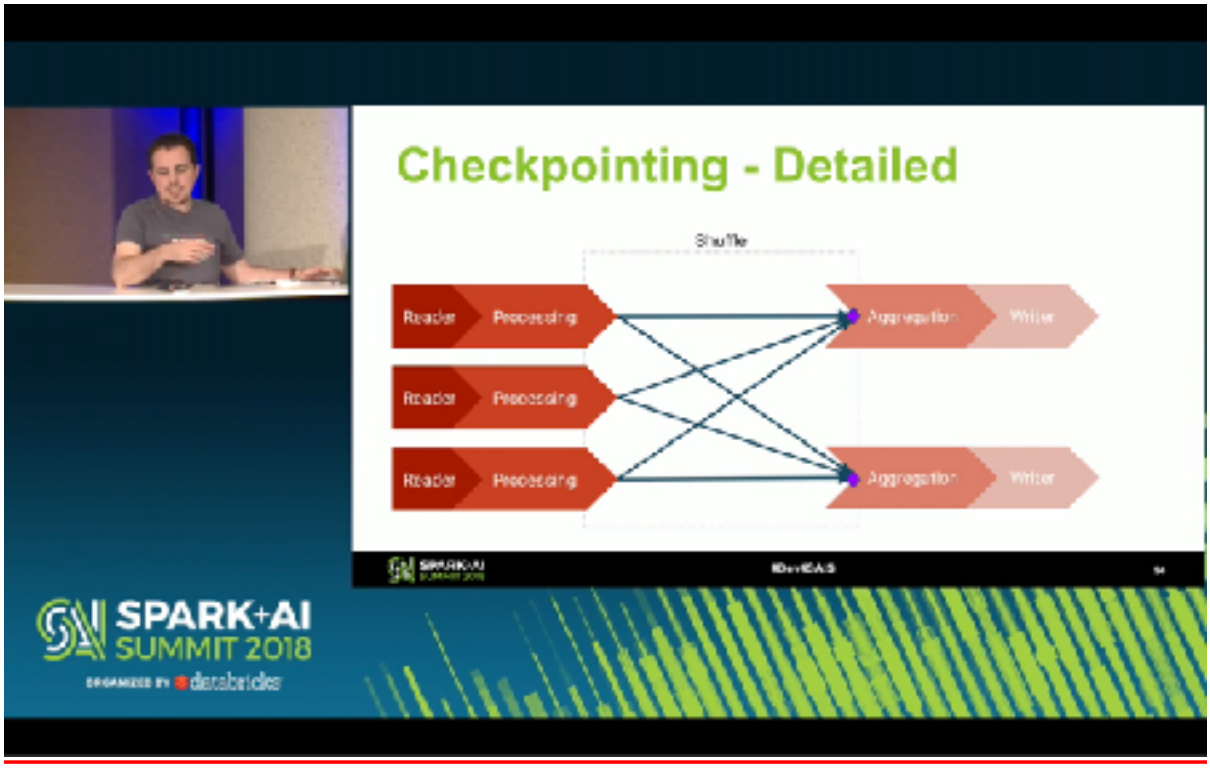

Checkpointing - Detailed
Driver
epoch markers
Shuffle
Reader
Processing
Aggregation
Writer
SPARK+AI SUMMIT 2018
databricks

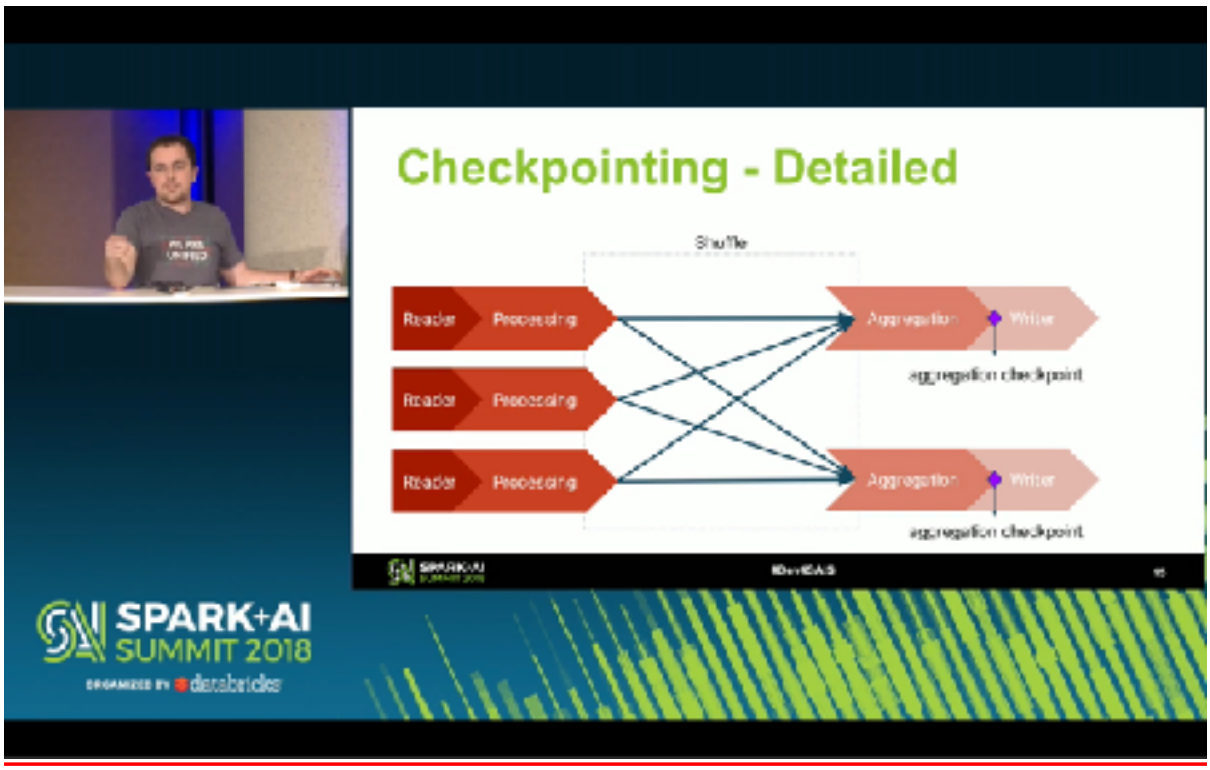

Continuous Processing in Structured Streaming - Jose Torres
Checkpointing - Detailed
Shuffle
Reader
Processing
Reader
Processing
Reader
Processing
Aggregation
Writer
Aggregation
Writer
SPARK+AI SUMMIT 2018
13:16 / 25:49

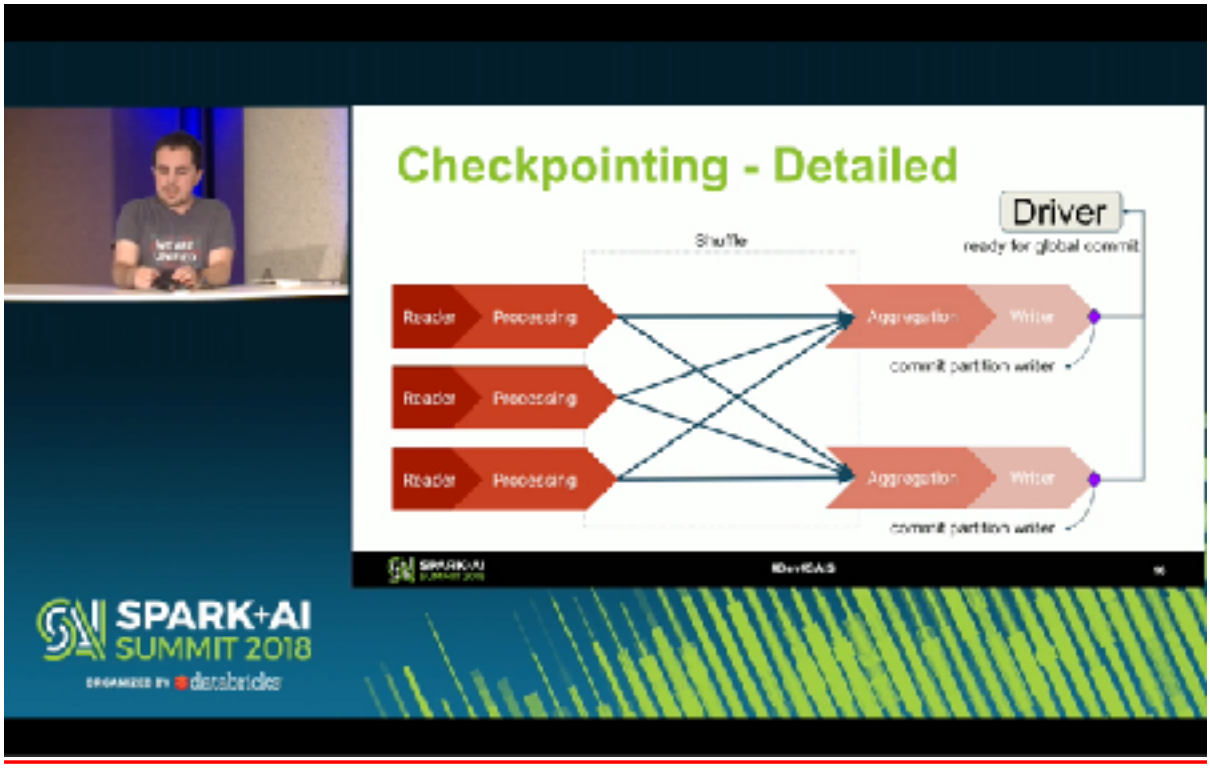

Continuous Processing in Structured Streaming - Jose Torres
Checkpointing - Detailed
Shuffle
Reader
Processing
Reader
Processing
Reader
Processing
Aggregation
Writer
Aggregation
Writer
SPARK+AI SUMMIT 2018
13:14 / 25:49

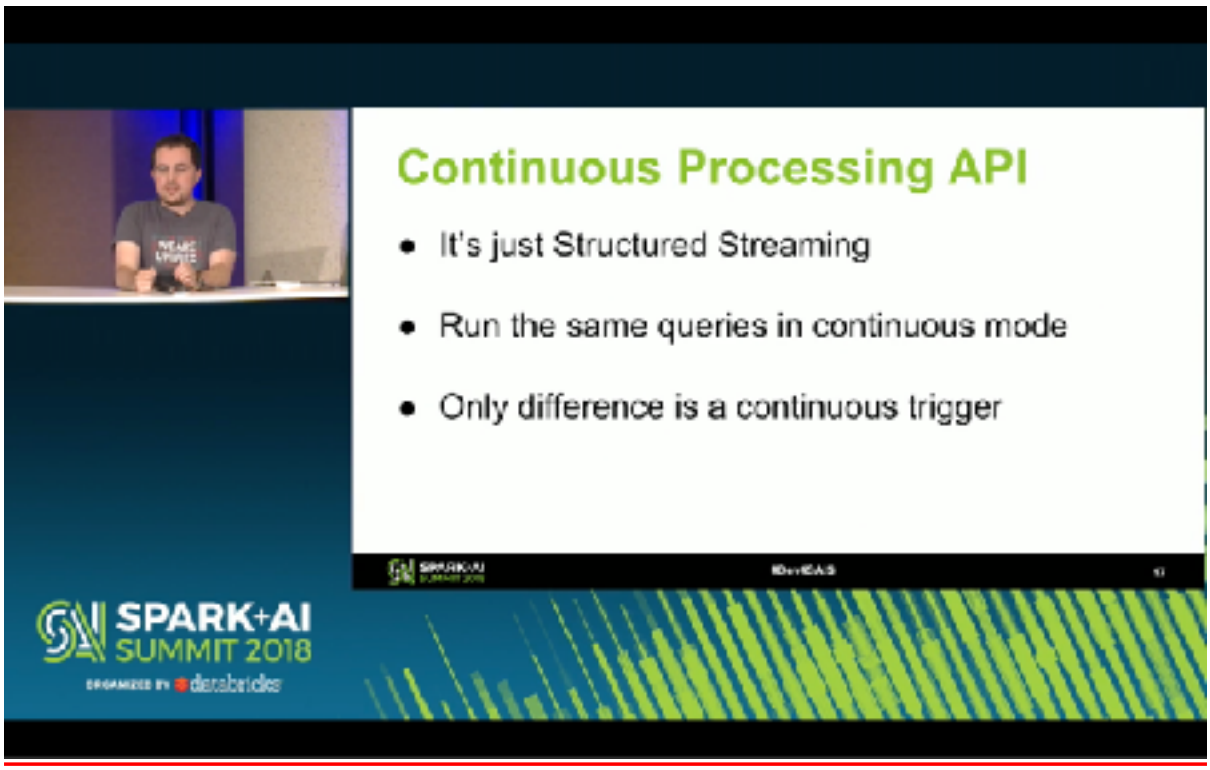

Checkpointing - Detailed
Shuffle
Reader
Processing
Reader
Processing
Reader
Processing
Aggregation
Writer
Aggregation
Writer
SPARK+AI SUMMIT 2018

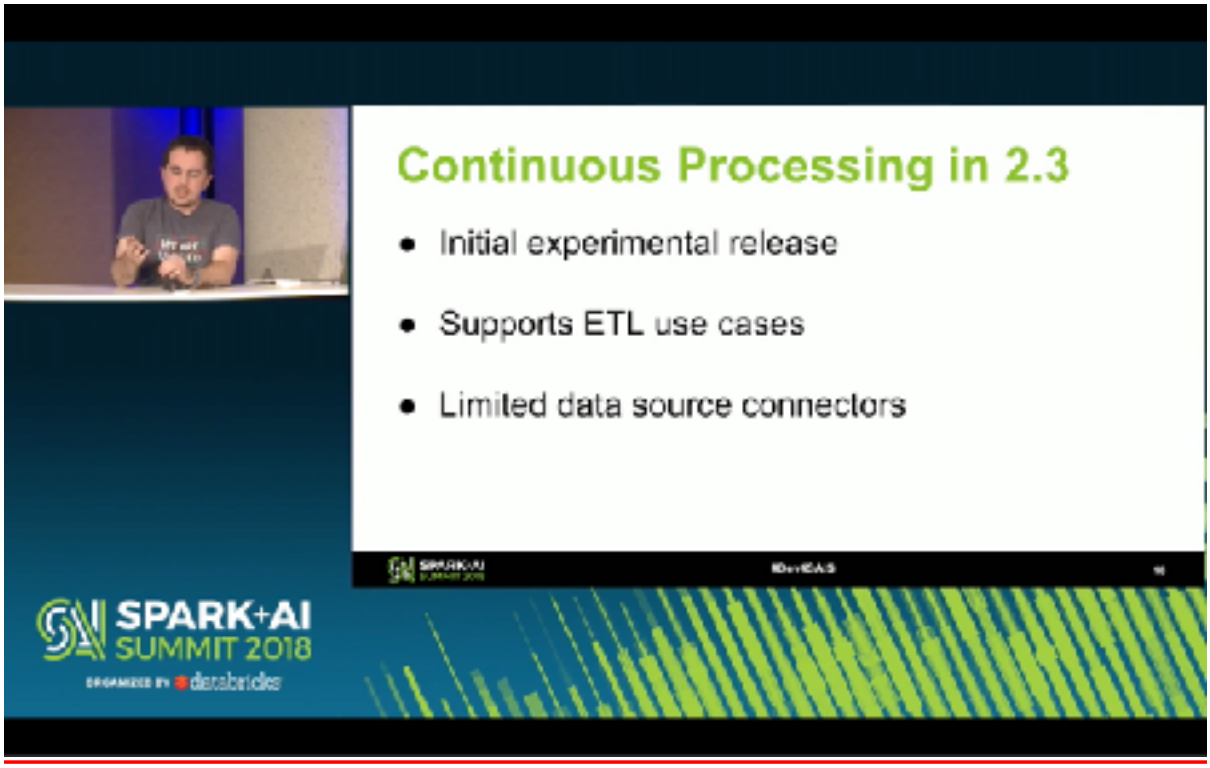

Checkpointing - Detailed
Shuffle
Reader
Processing
Reader
Processing
Reader
Processing
Aggregation
Writer
Aggregation
Writer
SPARK+AI SUMMIT 2018

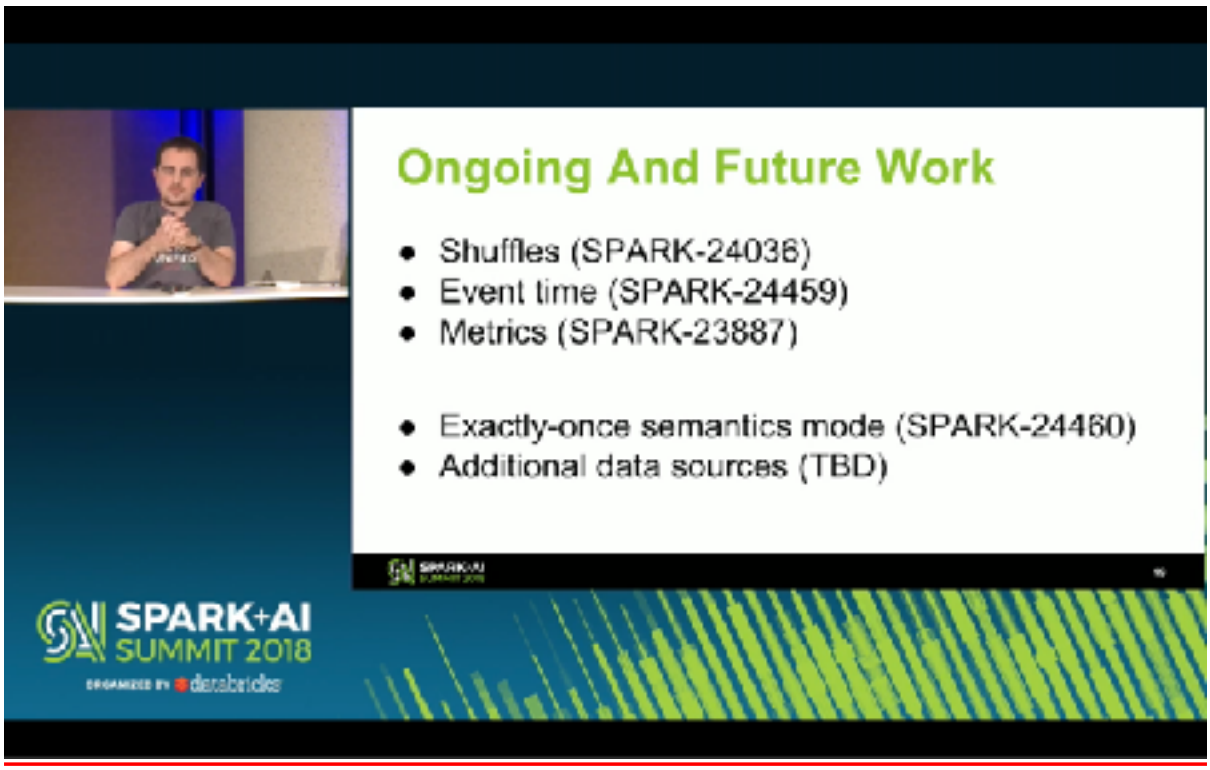

Checkpointing - Detailed
Shuffle
Reader
Processing
Reader
Processing
Reader
Processing
Aggregation
Writer
Aggregation
Writer
SPARK+AI SUMMIT 2018

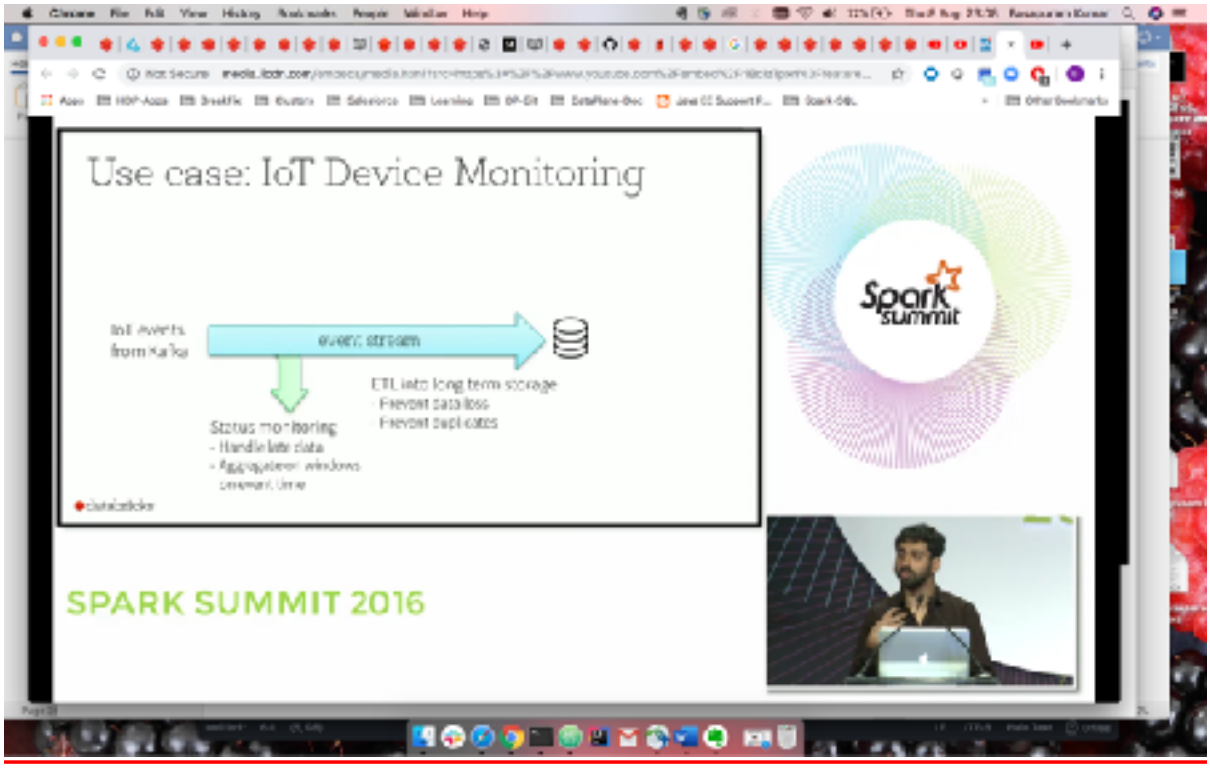

Checkpointing - Detailed
Shuffle
Reader
Processing
Reader
Processing
Reader
Processing
Aggregation
Writer
Aggregation
Writer
SPARK+AI SUMMIT 2018

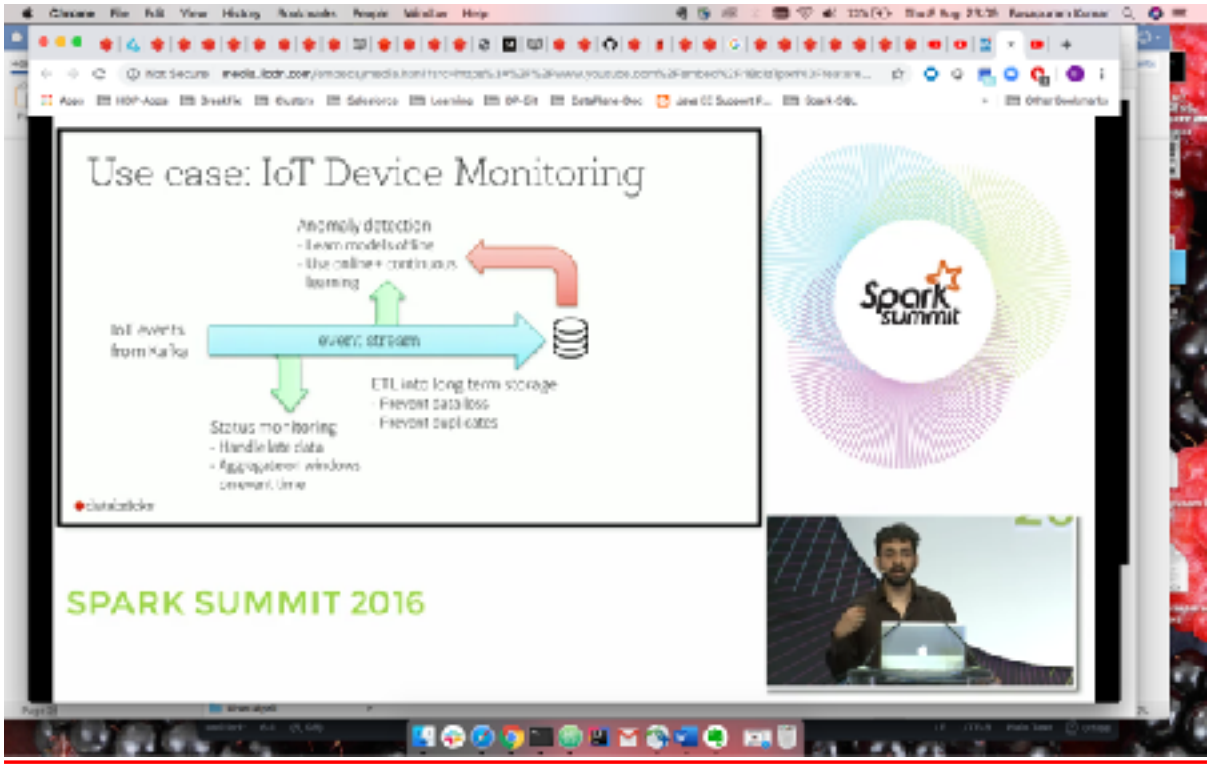

Checkpointing - Detailed
Driver
ready for global commit
Shuffle
Reader
Processing
Reader
Processing
Reader
Processing
Aggregation
Writer
commit partition writer
Aggregation
Writer
commit partition writer
SPARK+AI SUMMIT 2018

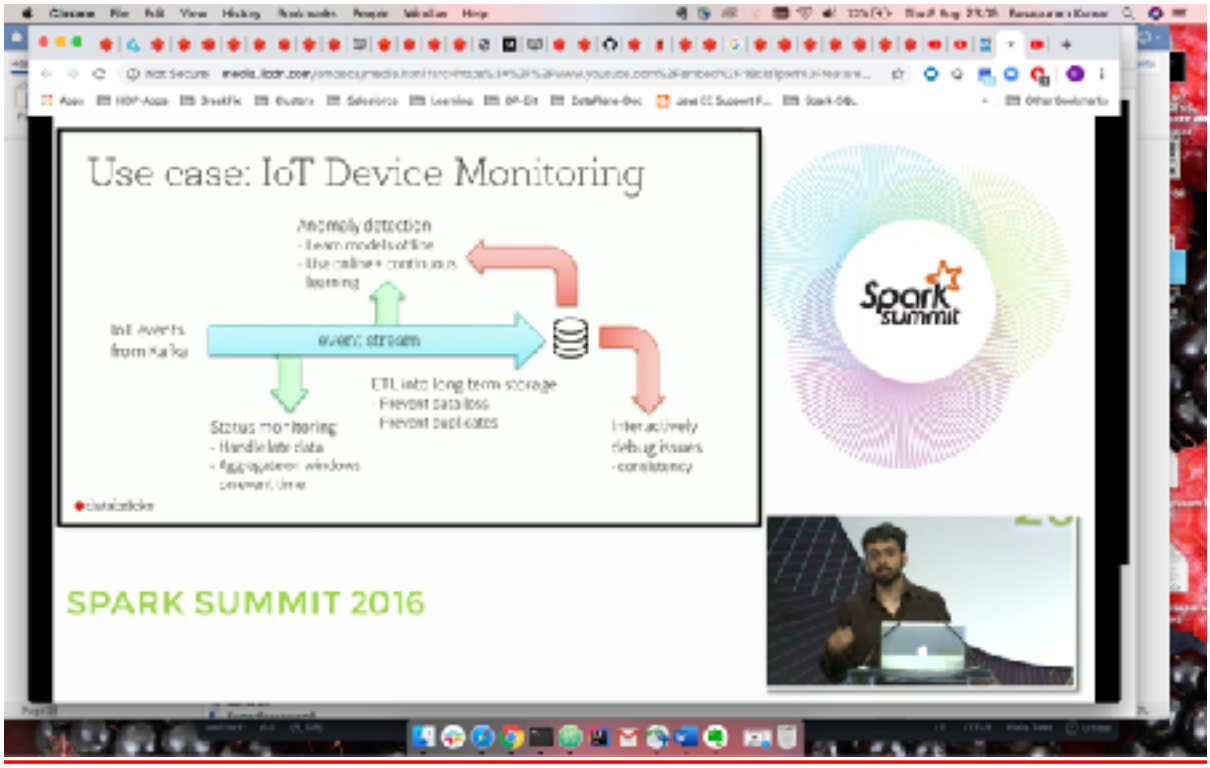

Continuous Processing API
• It's just Structured Streaming
• Run the same queries in continuous mode
• Only difference is a continuous trigger
SPARK+AI SUMMIT 2018

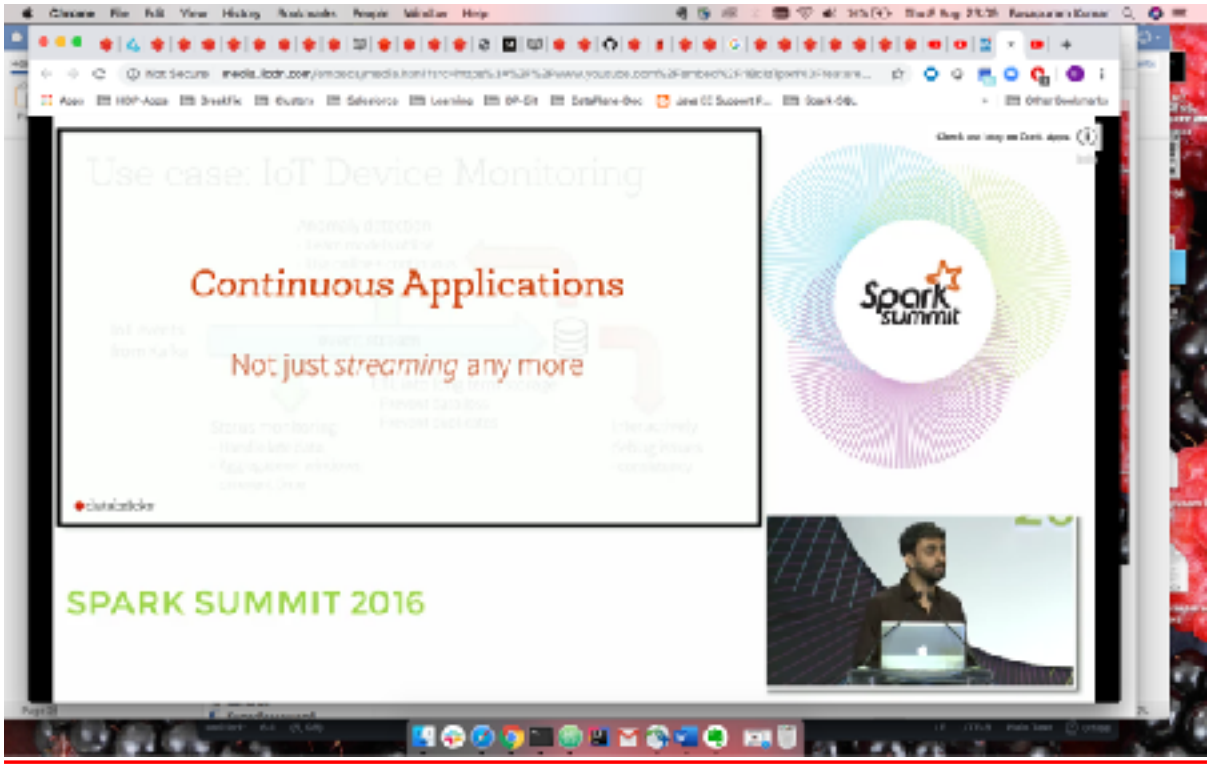

Continuous Processing in 2.3
Initial experimental release
Supports ETL use cases
Limited data source connectors
SPARK+AI SUMMIT 2018

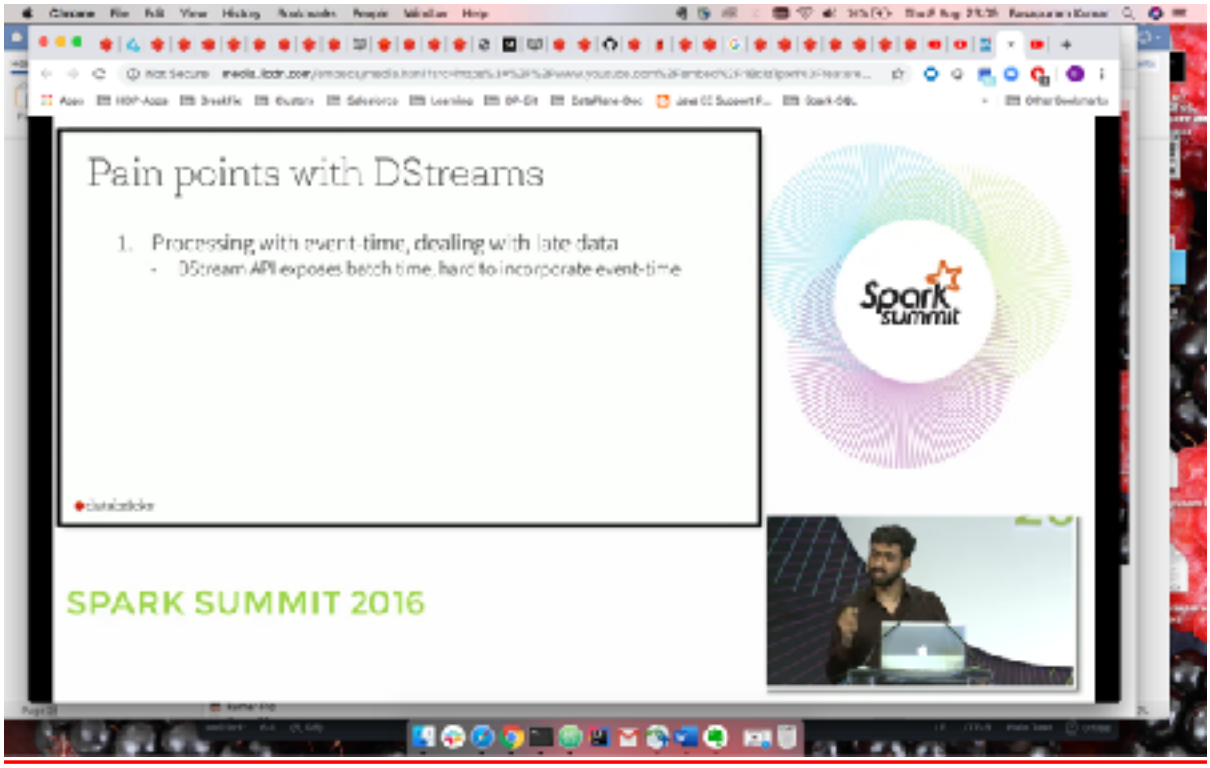

Ongoing And Future Work
Shuffles (SPARK-24036)
Event time (SPARK-24459)
Metrics (SPARK-23887)
Exactly-once semantics mode (SPARK-24460)
Additional data sources (TBD)
SPARK+AI SUMMIT 2018

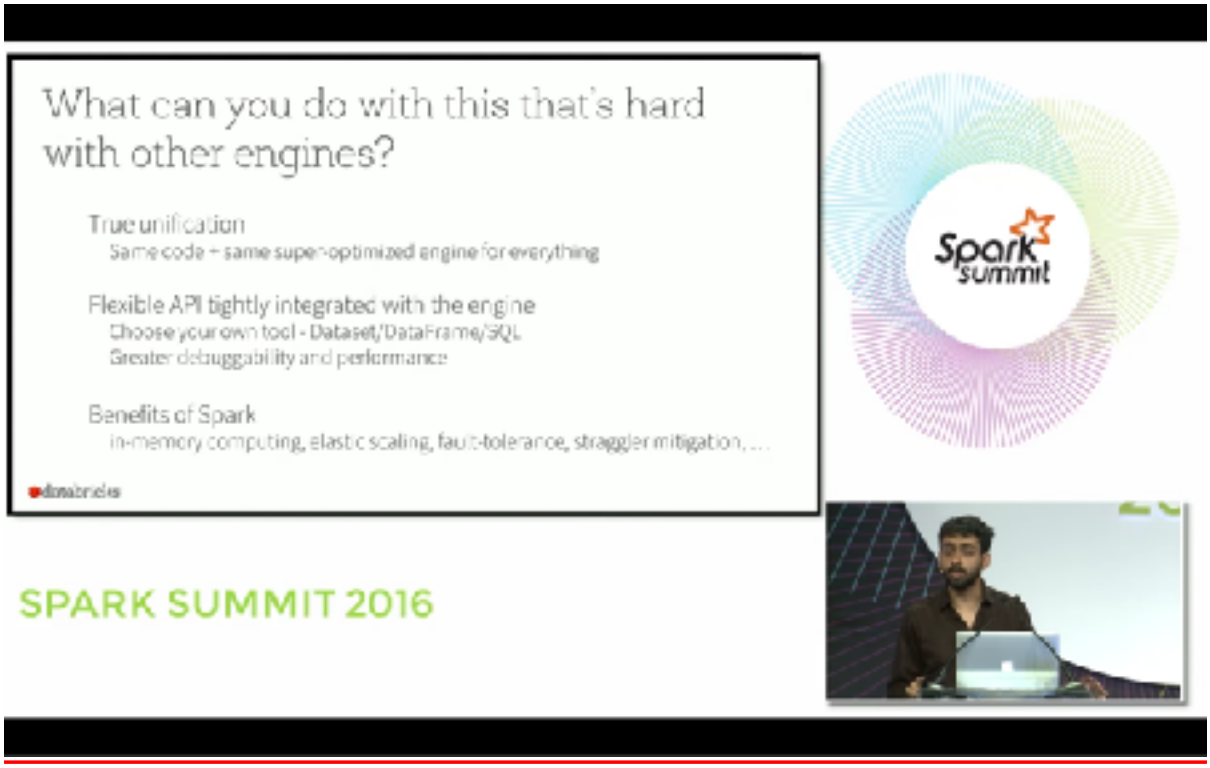

Use case: IoT Device Monitoring
IoT events from Kafka
event stream
ETL into long term storage
- Prevent data loss
- Prevent duplicates
Status monitoring
- Handle late data
- Aggregations: windows
Spark summit
SPARK SUMMIT 2016

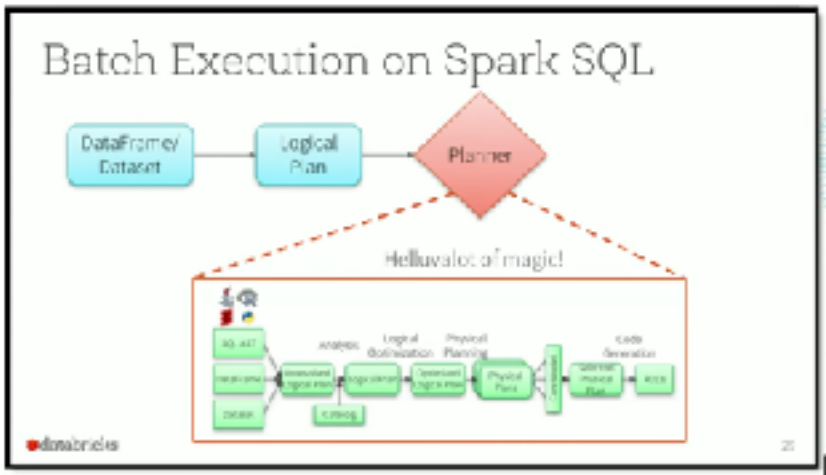

Use case: IoT Device Monitoring
Anomaly detection
- Learn models offline
- Use online + continuous learning
IoT events from Kafka
event stream
ETL into long term storage
- Prevent data loss
- Prevent duplicates
Status monitoring
- Handle late data
- Aggregations: windows
Spark summit
SPARK SUMMIT 2016


Use case: IoT Device Monitoring
Anomaly detection
- Learn models offline
- Use online + continuous learning
IoT events from Kafka
event stream
ETL into long term storage
- Prevent data loss
- Prevent duplicates
Status monitoring
- Handle late data
- Aggregates on windows
Interactively debug issues
- consistency
SPARK SUMMIT 2016

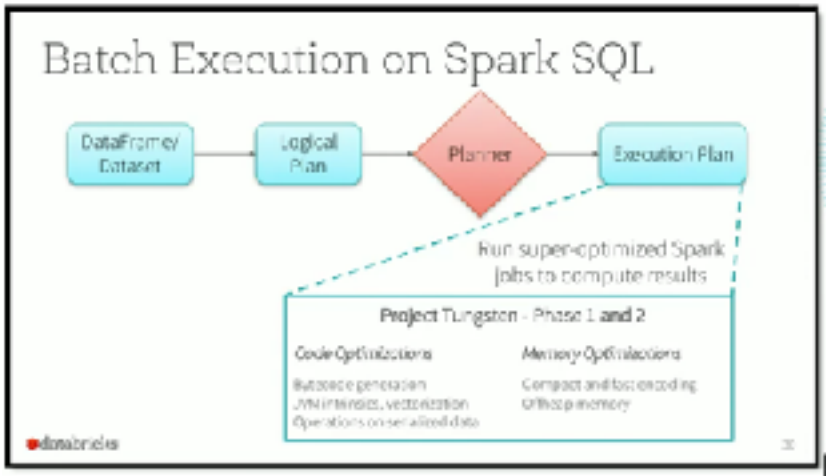

Continuous Applications
Not just streaming any more
SPARK SUMMIT 2016


Pain points with DStreams
1. Processing with event-time, dealing with late data
- DStream API exposes batch time, hard to incorporate event-time
databricks
Spark summit
SPARK SUMMIT 2016

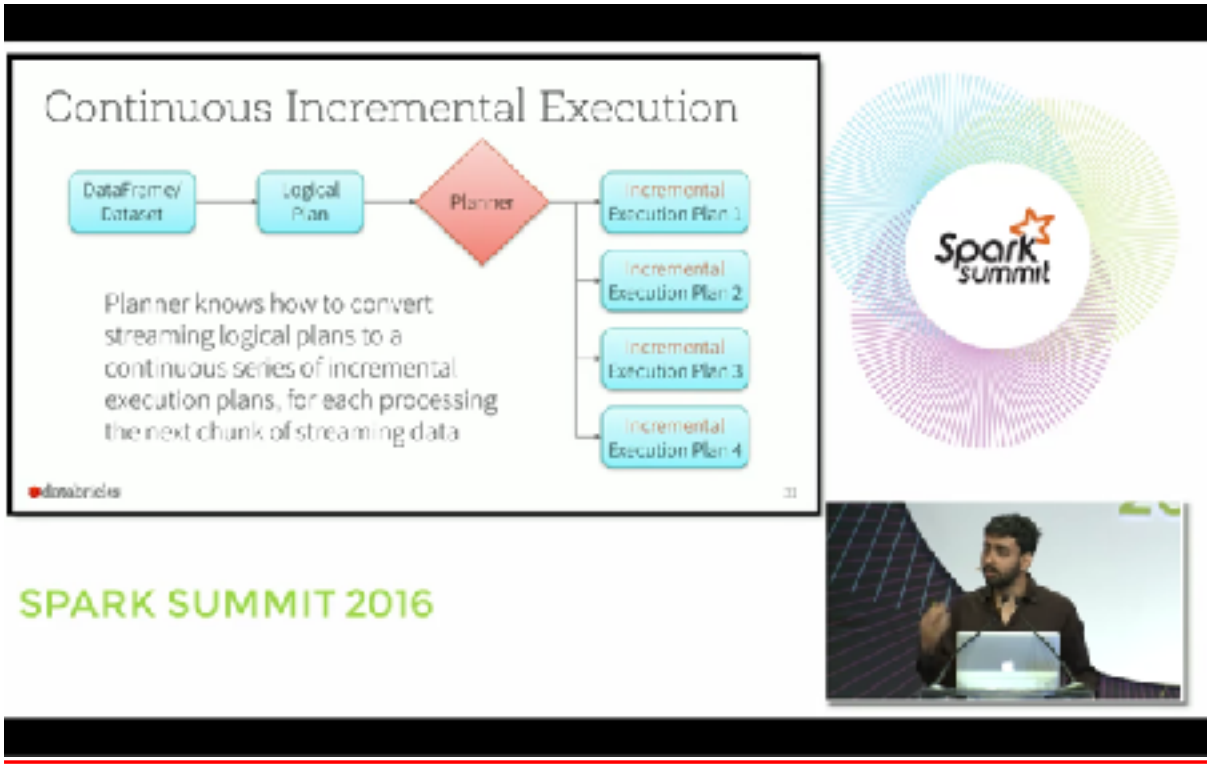

What can you do with this that's hard with other engines?
True unification
Same code = same super-optimized engine for everything
Flexible API tightly integrated with the engine
Choose your own tool - Dataset/DataFrame/SQL
Greater debuggability and performance
Benefits of Spark
in-memory computing, elastic scaling, fault-tolerance, straggler mitigation, ...
databricks
Spark summit
SPARK SUMMIT 2016



SPARK SUMMIT 2016



SPARK SUMMIT 2016



Spark Delta lake

Delta Lake is an open source storage layer that brings reliability to data lakes.

Delta Lake provides

1. ACID transactions,
2. Scalable metadata handling, and
3. Unifies streaming and batch data processing.
4. Delta Lake runs on top of your existing data lake and is fully compatible with Apache Spark APIs.

Challenges faced by current systems:

But the data is not ready for data science & ML

The **majority** of these projects are failing due to **unreliable data!**

Data Lake

Data Science & ML

- **Recommendation Engines**
- **Risk, Fraud Detection**
- **IoT & Predictive Maintenance**
- **Genomics & DNA Sequencing**

databricks

Data reliability challenges with data lakes

Failed production jobs leave data in corrupt state requiring tedious recovery

Lack of schema enforcement creates inconsistent and low quality data

Lack of consistency makes it almost impossible to mix appends ands reads, batch and streaming

databricks

A New Standard for Building Data Lakes

Open Format Based on Parquet

With Transactions

Apache Spark API's

databricks

Delta Lake: makes data ready for Analytics

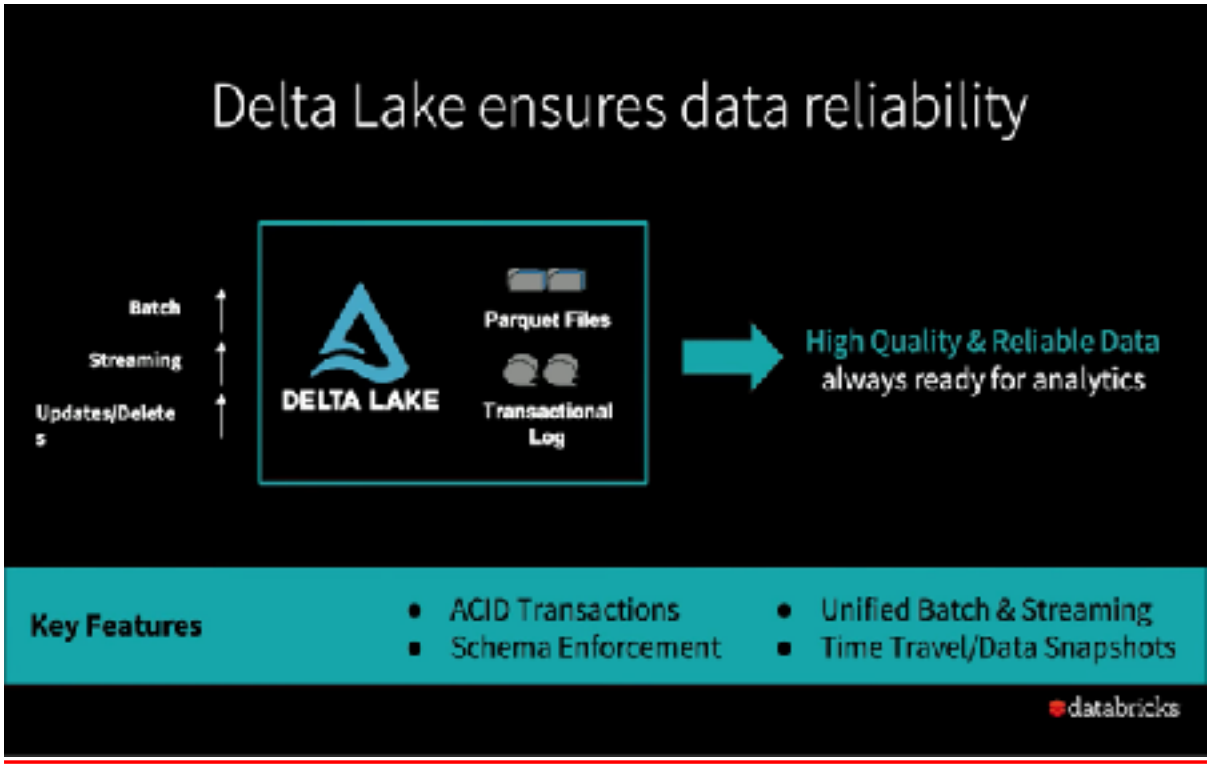
Reliability

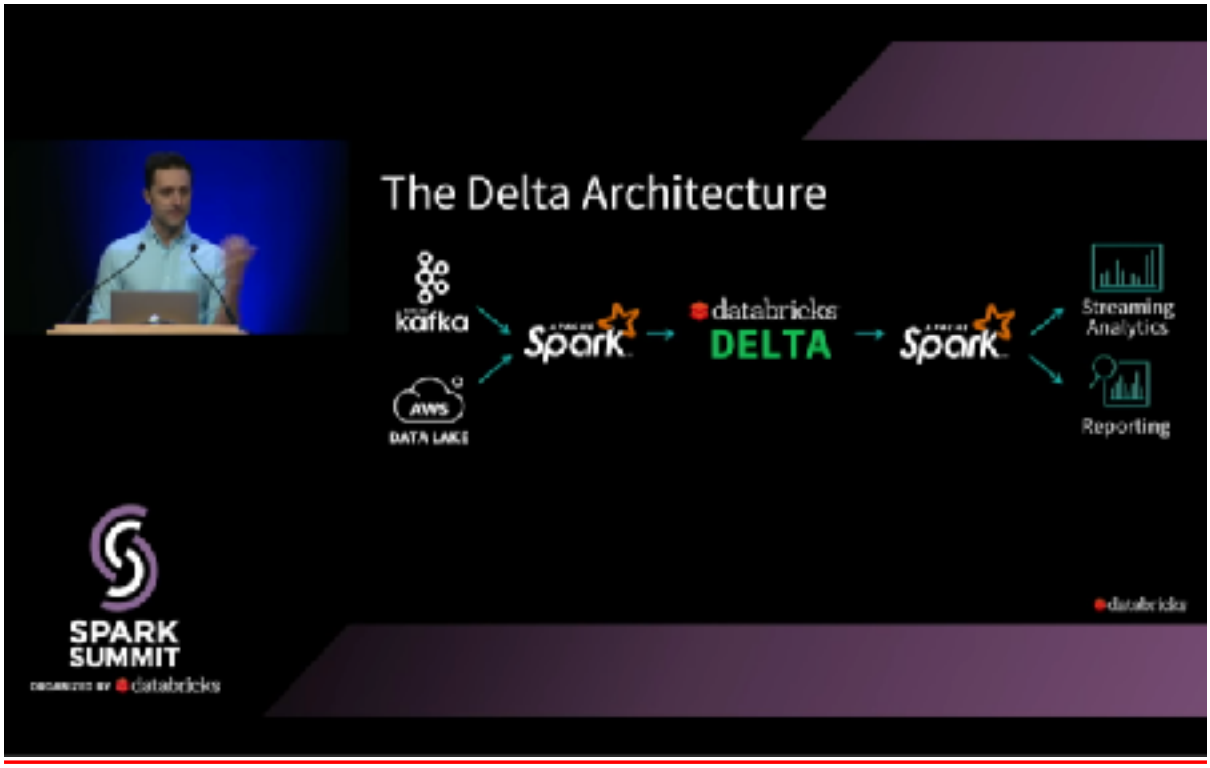
Performance

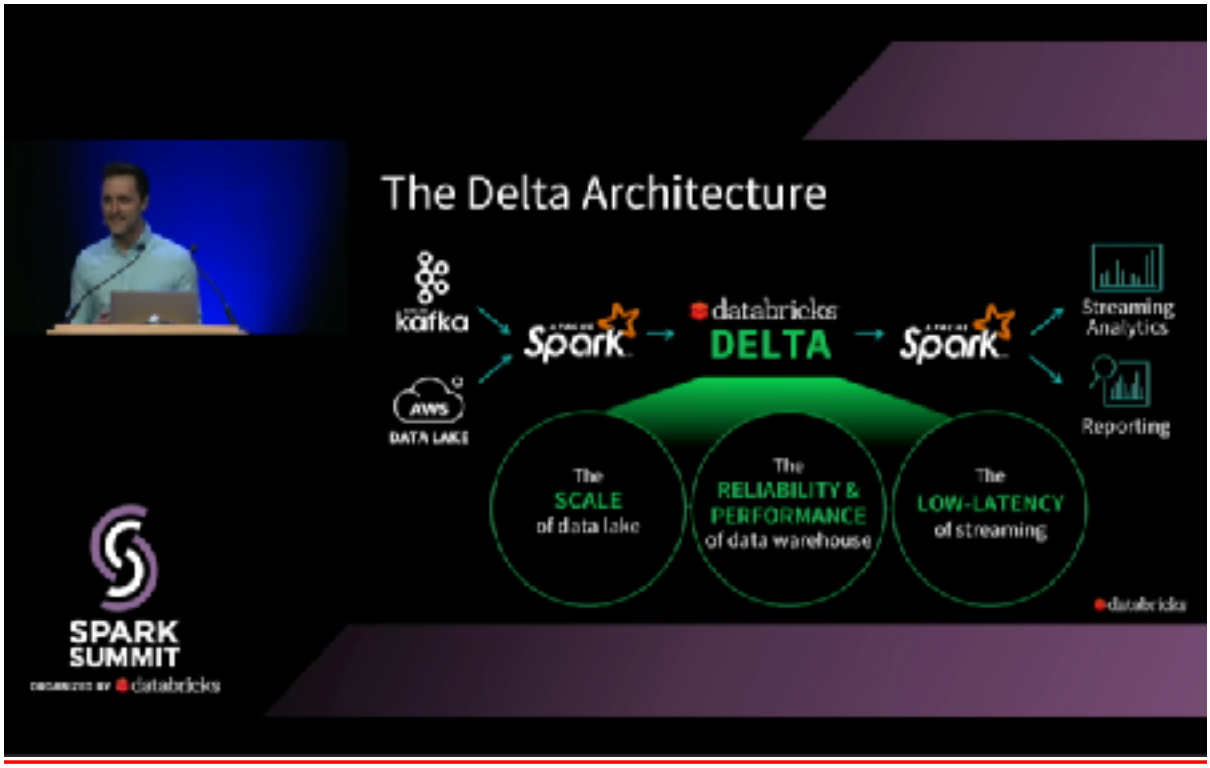
Data Science & ML

- **Recommendation Engines**
- **Risk, Fraud Detection**
- **IoT & Predictive Maintenance**
- **Genomics & DNA Sequencing**

databricks


Delta Lake ensures data reliability
Batch
Streaming
Updates/Deletes
DELTA LAKE
Parquet Files
Transactional Log
High Quality & Reliable Data
always ready for analytics
Key Features
ACID Transactions
Schema Enforcement
Unified Batch & Streaming
Time Travel/Data Snapshots
databricks


The Delta Architecture
kafka
AWS
DATA LAKE
Spark
databricks
DELTA
Spark
Streaming Analytics
Reporting
SPARK SUMMIT
databricks


The Delta Architecture
kafka
AWS
DATA LAKE
Spark
databricks
DELTA
Spark
Streaming Analytics
Reporting
The SCALE of data lake
The RELIABILITY & PERFORMANCE of data warehouse
The LOW-LATENCY of streaming
SPARK SUMMIT
databricks