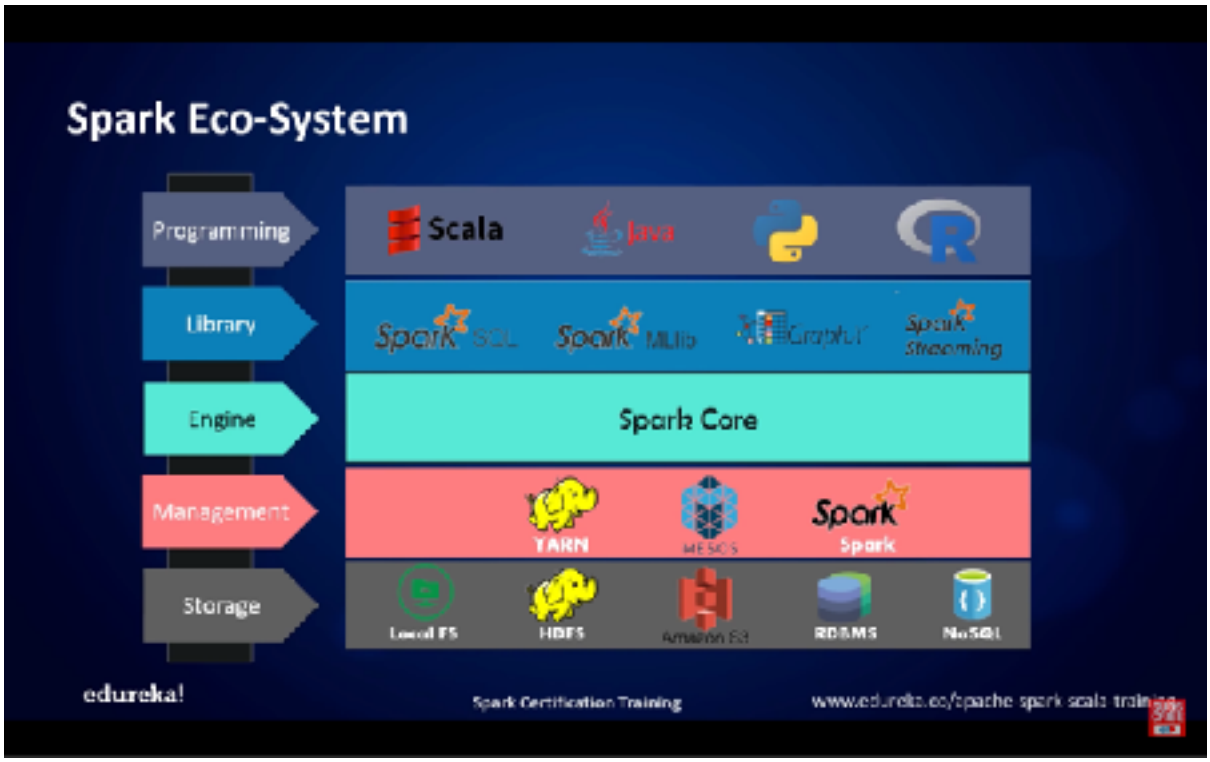
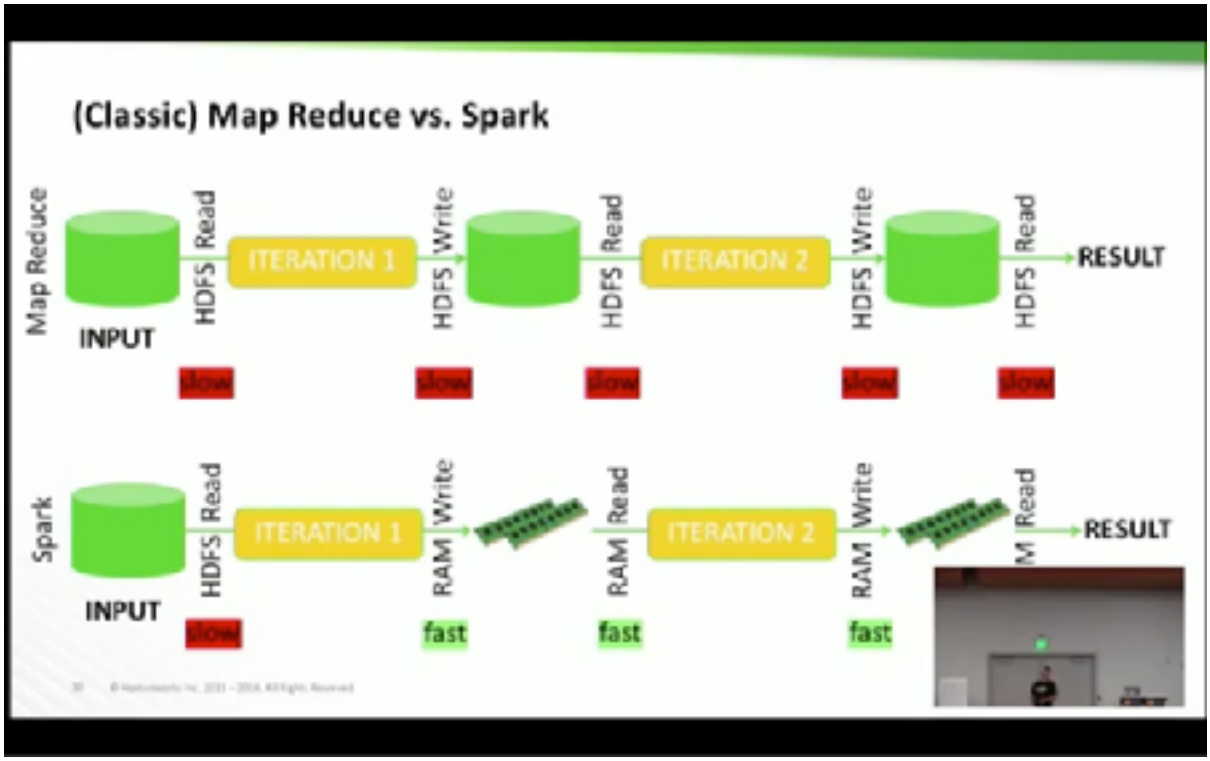
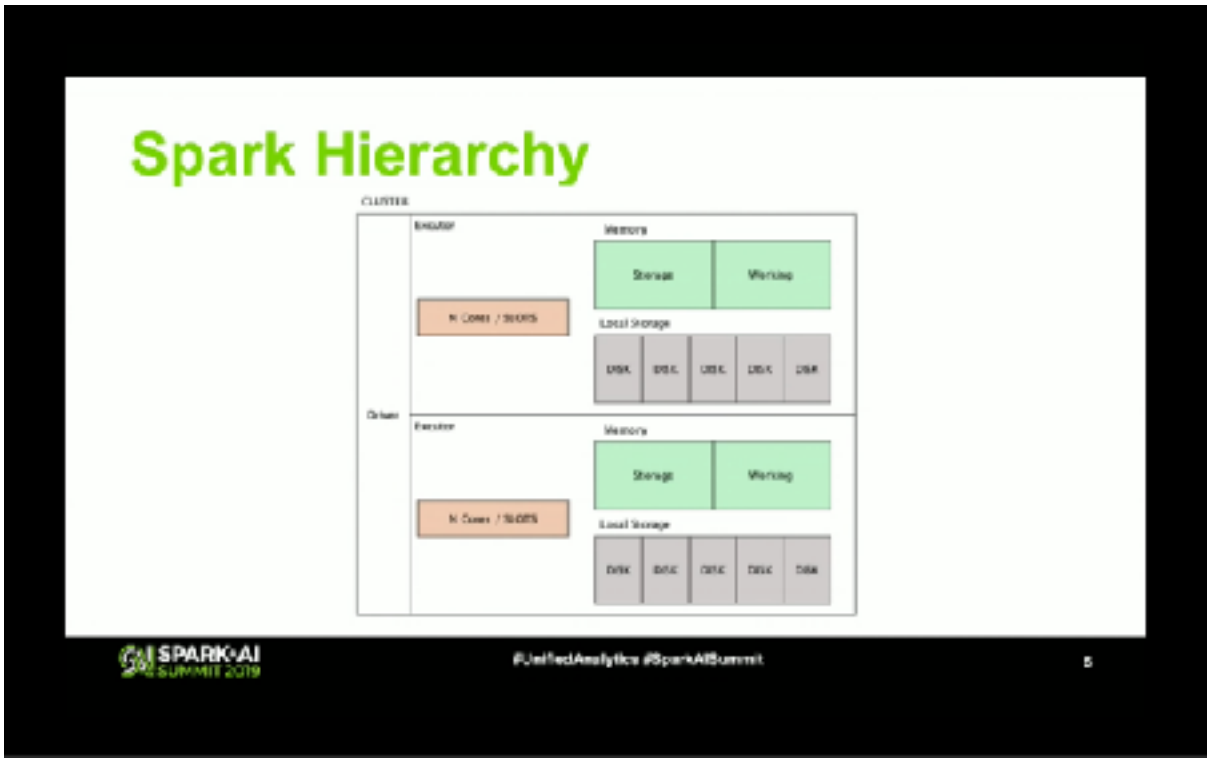
Spark Eco-System

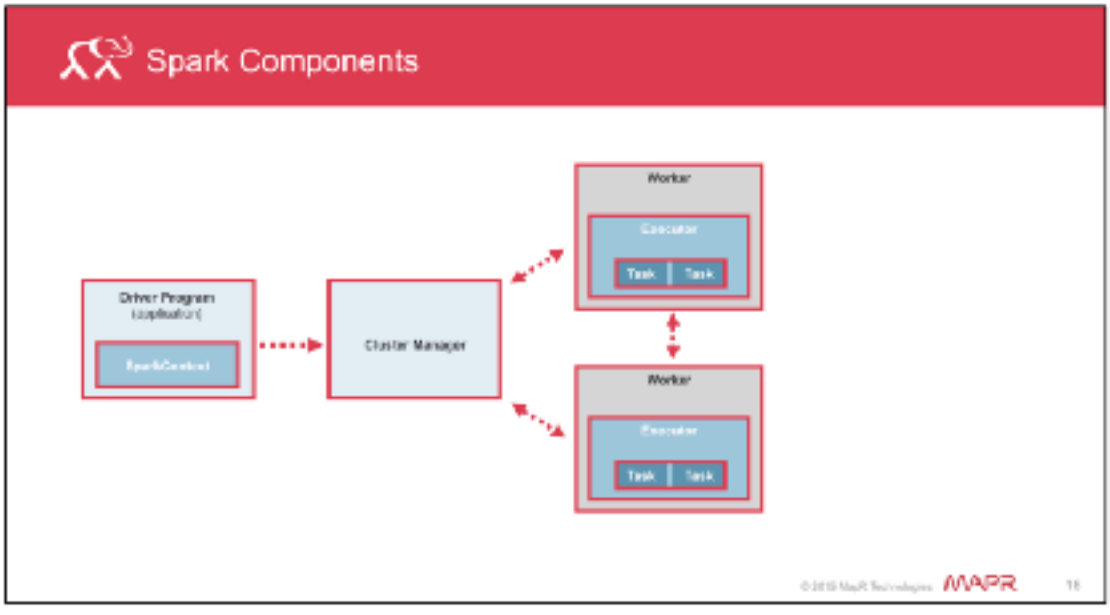


Classic Diff. between MapReduce and Spark(MR vs Spark)



Spark Architecure:





A Spark cluster consists of two processes, a driver program and multiple workers nodes each running an executor process. The driver program runs on the driver machine, the worker programs run on cluster nodes or in local threads.

1. The first thing a program does is to create a SparkContext object. This tells Spark how and where to access a cluster
2. SparkContext connects to cluster manager. Cluster manager allocates resources across applications
3. Once connected, Spark acquires executors in the worker nodes (an executor is a process that runs computations and stores data for your application)
4. Jar or python files passed to the SparkContext are then sent to the executors.
5. SparkContext will then send the tasks for the executor to run.
6. The worker nodes can access data storage sources to ingest and output data as needed.

Spark internals:

Reference:
https://www.youtube.com/watch?v=dmL0N3qfSc8

Why understand internals?

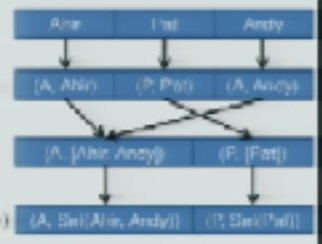
Goal: Find number of distinct names per "first letter"

```
sc.textFile("hdfs://names")
  .map(name => (name.charAt(0), name))
  .groupByKey()
  .mapValues(names => names.toSet.size)
  .collect()
```

Why understand internals?

Goal: Find number of distinct names per "first letter"

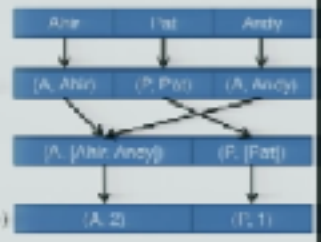
```
sc.textFile("hdfs://names")
  .map(name => (name.charAt(0), name))
  .groupByKey()
  .mapValues(names => names.toSet.size)
  .collect()
```



Why understand internals?

Goal: Find number of distinct names per "first letter"

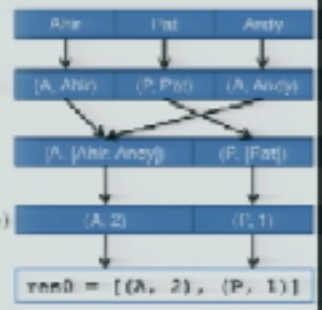
```
sc.textFile("hdfs://names")
  .map(name => (name.charAt(0), name))
  .groupByKey()
  .mapValues(names => names.toSet.size)
  .collect()
```

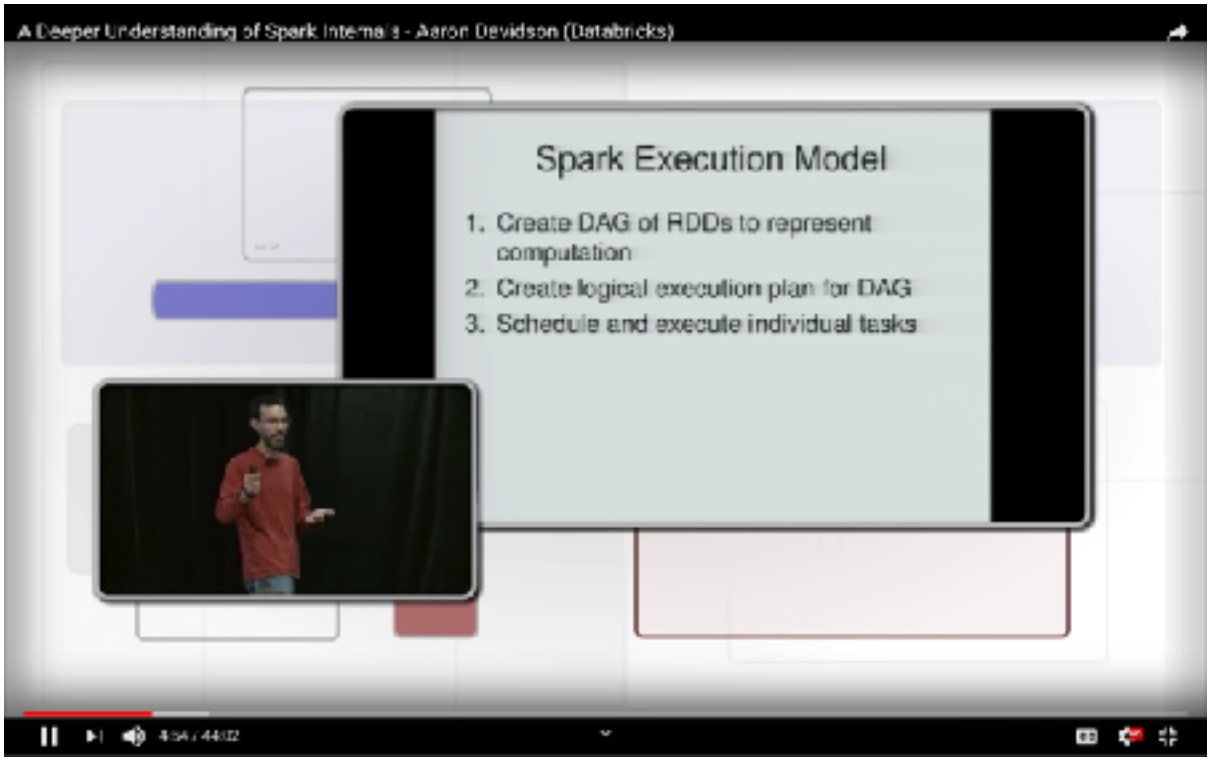


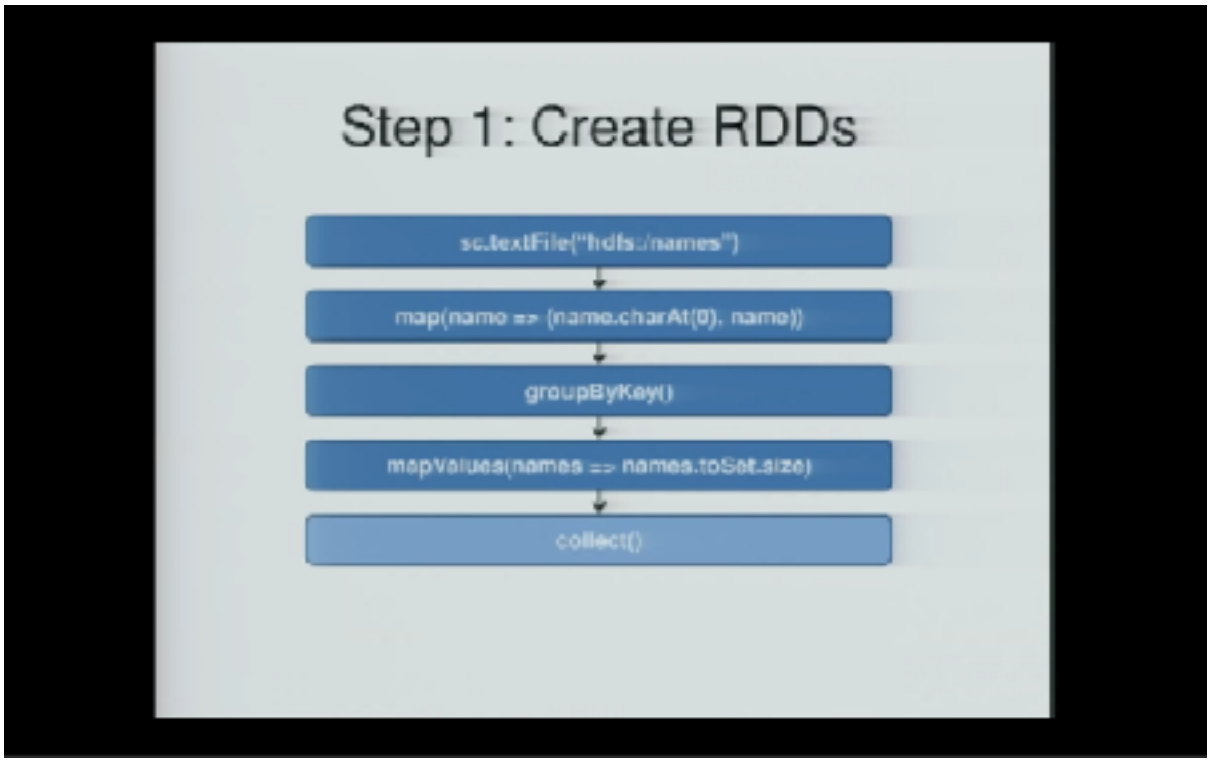
Why understand internals?

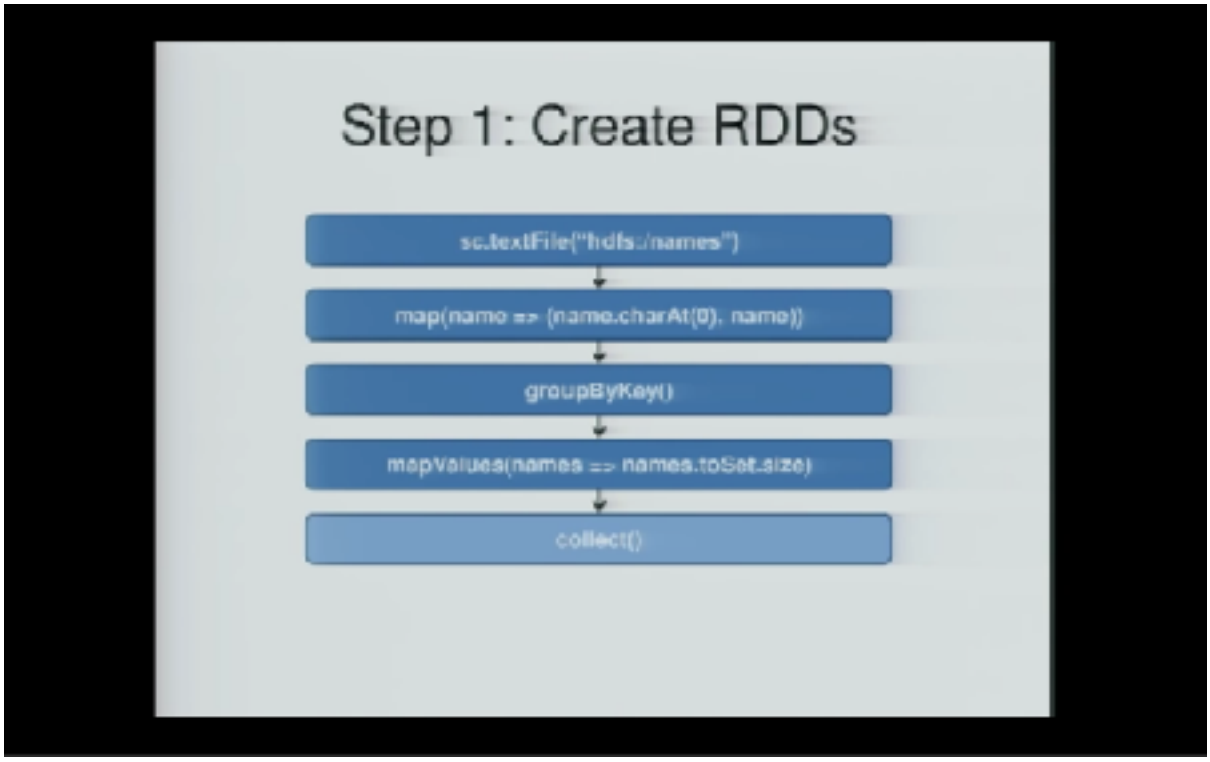
Goal: Find number of distinct names per "first letter"

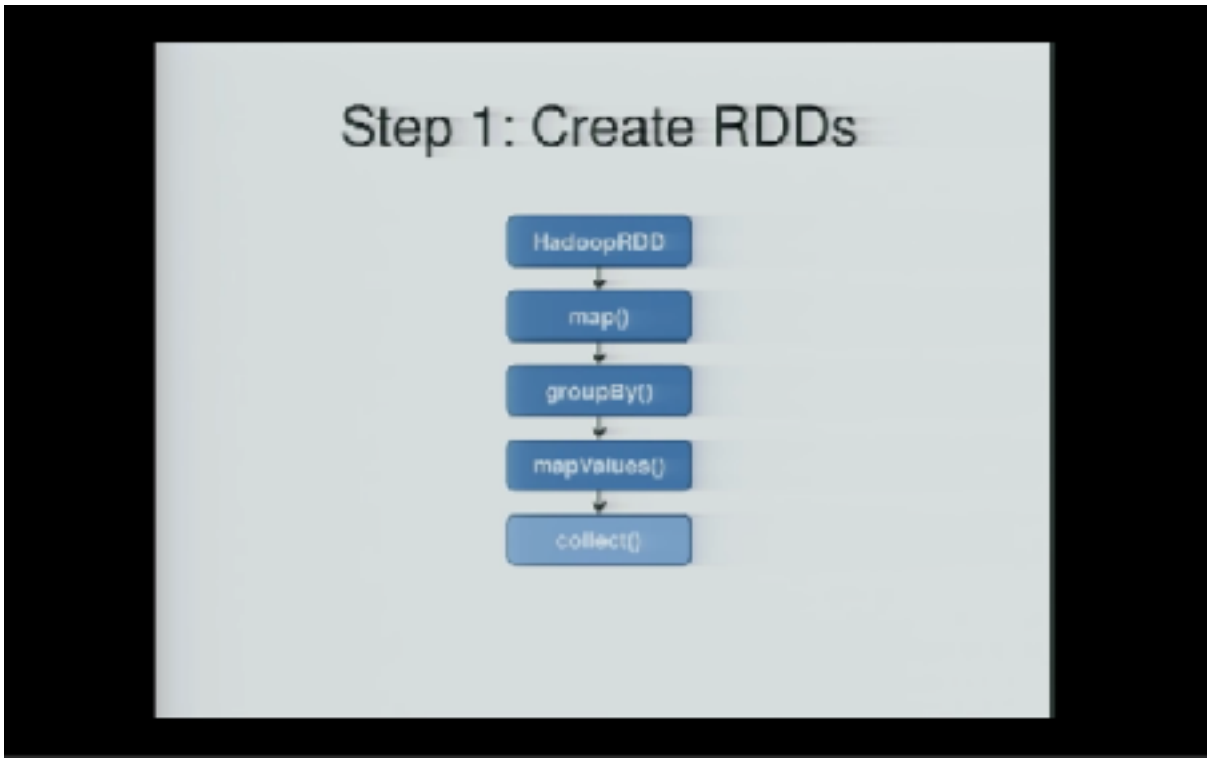
```
sc.textFile("hdfs://names")
  .map(name => (name.charAt(0), name))
  .groupByKey()
  .mapValues(names => names.toSet.size)
  .collect()
```

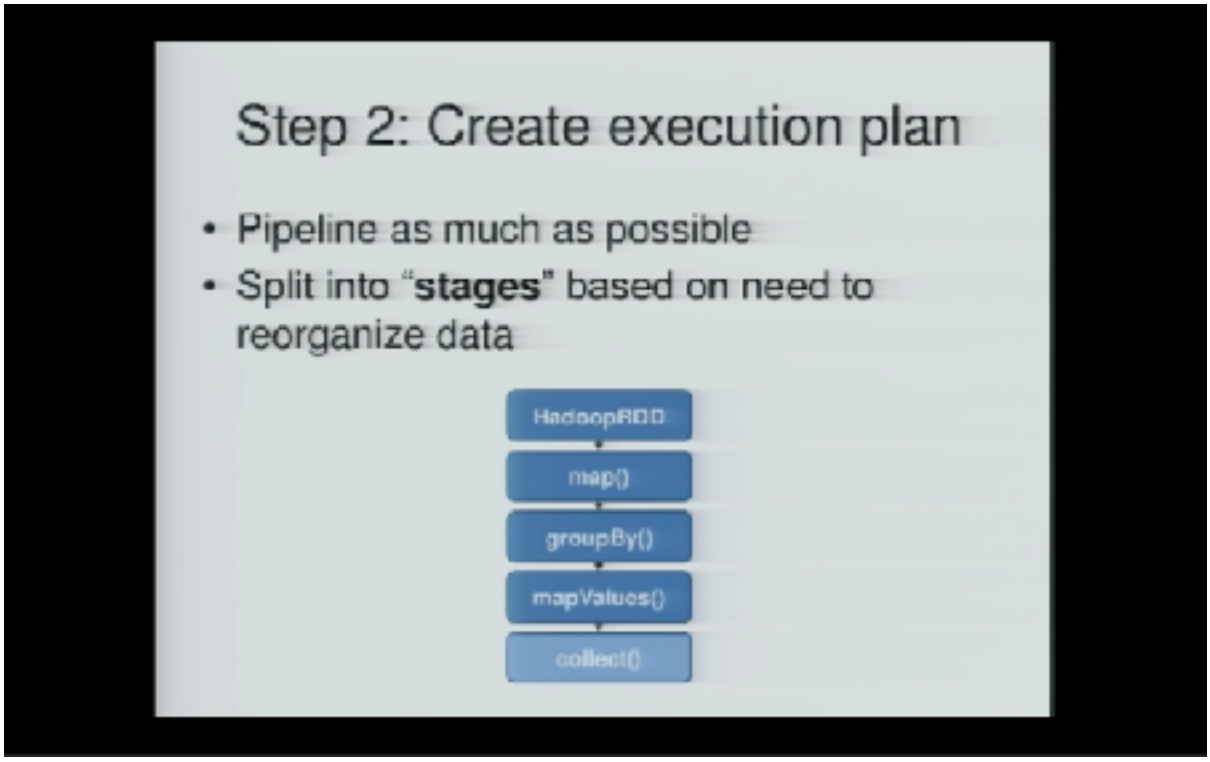


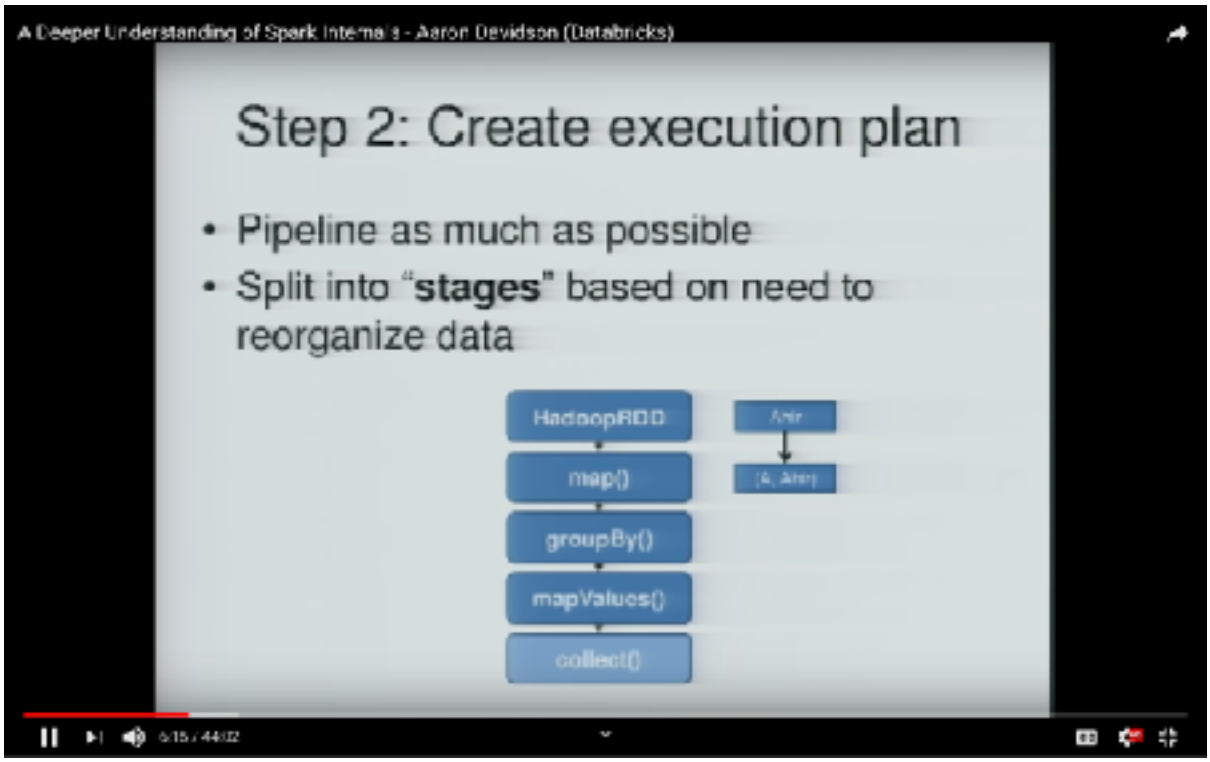

A Deeper Understanding of Spark Internals - Aaron Davidson (Databricks)
Spark Execution Model
1. Create DAG of RDDs to represent computation
2. Create logical execution plan for DAG
3. Schedule and execute individual tasks


Step 1: Create RDDs
sc.textFile("hdfs://names")
map(name => (name.charAt(0), name))
groupByKey()
mapValues(names => names.toSet.size)
collect()


Step 1: Create RDDs
sc.textFile("hdfs:/names")
map(name => (name.charAt(0), name))
groupByKey()
mapValues(names => names.toSet.size)
collect()

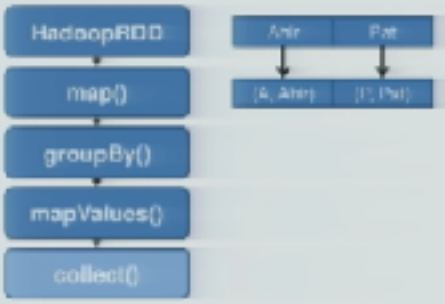

Step 1: Create RDDs
HadoopRDD
map()
groupBy()
mapValues()
collect()


Step 2: Create execution plan
• Pipeline as much as possible
• Split into "stages" based on need to reorganize data
HadoopRDD
map()
groupBy()
mapValues()
collect()


A Deeper Understanding of Spark Internals - Aaron Davidson (Databricks)
Step 2: Create execution plan
• Pipeline as much as possible
• Split into "stages" based on need to reorganize data
HadoopRDD
map()
groupBy()
mapValues()
collect()
Arr
(A, Arr)

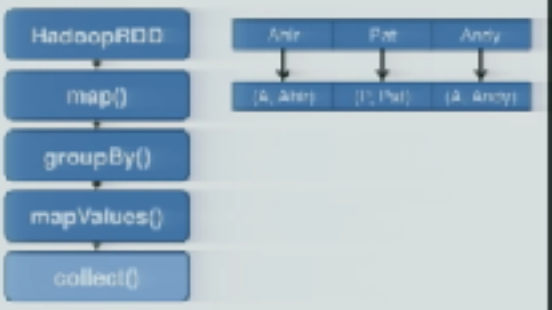
Step 2: Create execution plan

- Pipeline as much as possible
- Split into "**stages**" based on need to reorganize data



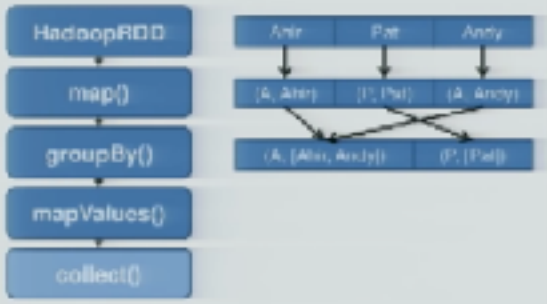
Step 2: Create execution plan

- Pipeline as much as possible
- Split into "**stages**" based on need to reorganize data



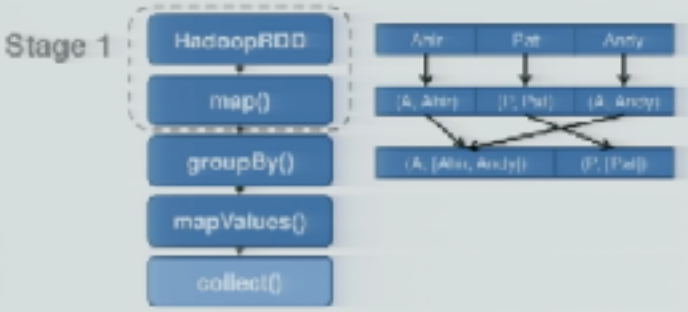
Step 2: Create execution plan

- Pipeline as much as possible
- Split into "**stages**" based on need to reorganize data



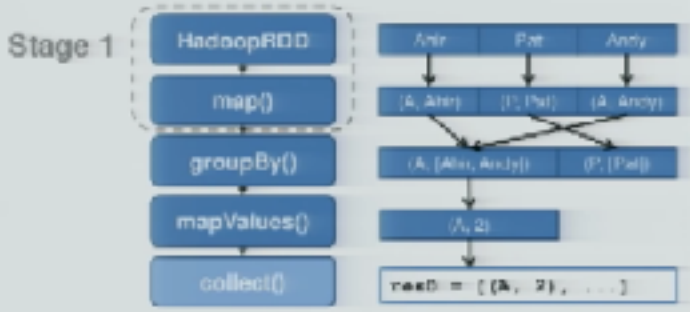
Step 2: Create execution plan

- Pipeline as much as possible
- Split into "**stages**" based on need to reorganize data



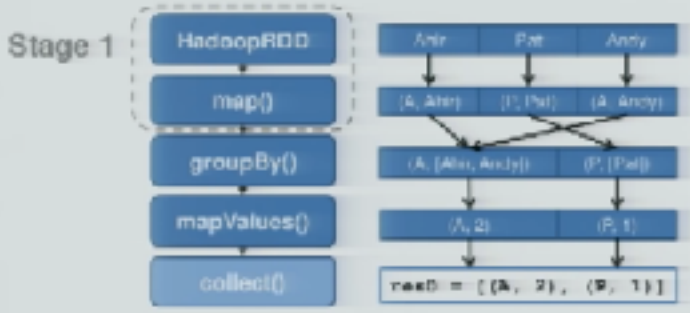
Step 2: Create execution plan

- Pipeline as much as possible
- Split into "**stages**" based on need to reorganize data



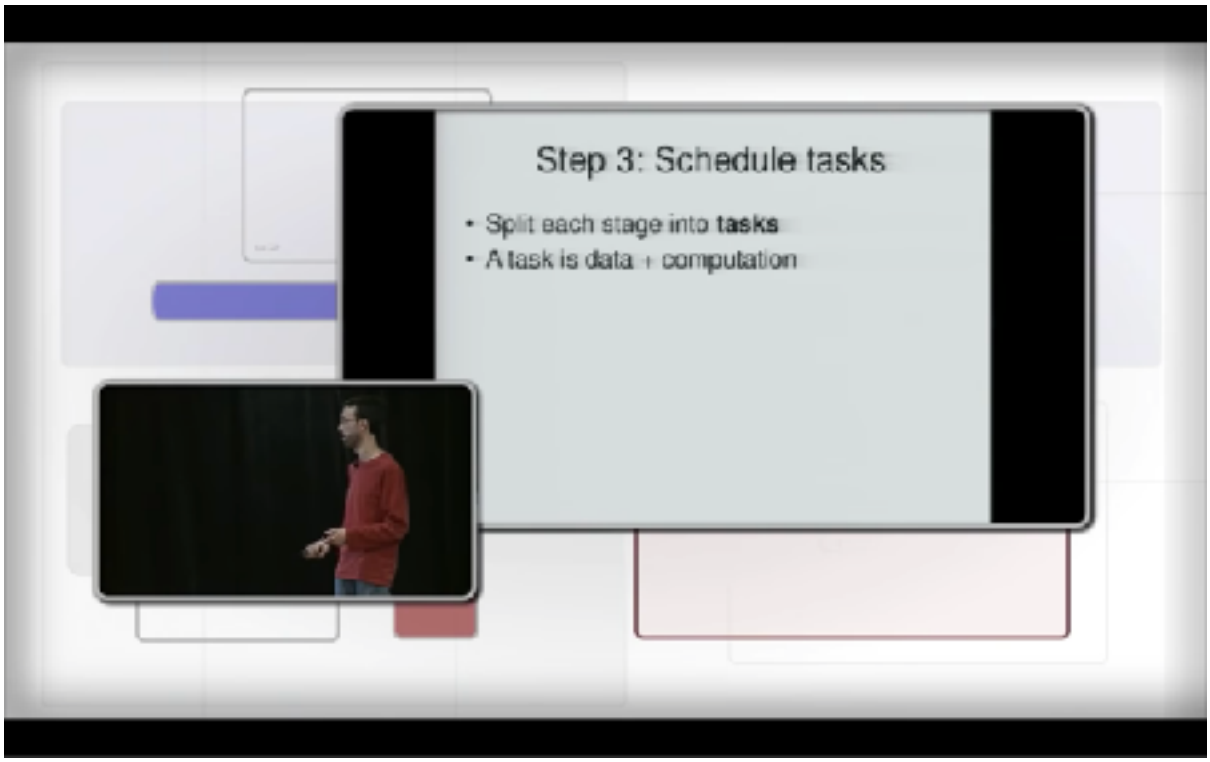
Step 2: Create execution plan

- Pipeline as much as possible
- Split into "**stages**" based on need to reorganize data



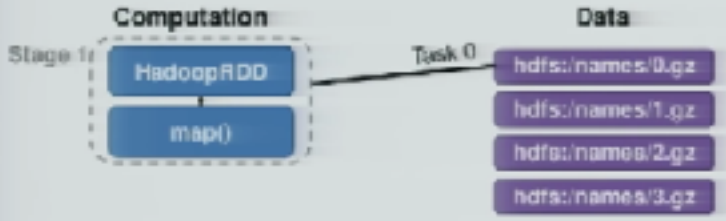
Step 3: Schedule tasks

- Split each stage into **tasks**

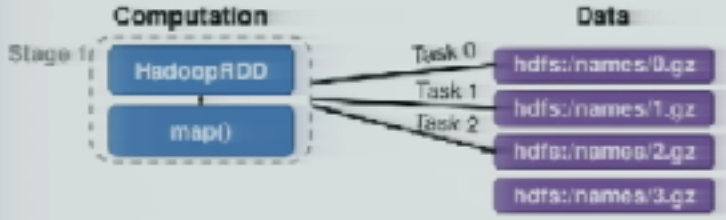


Step 3: Schedule tasks

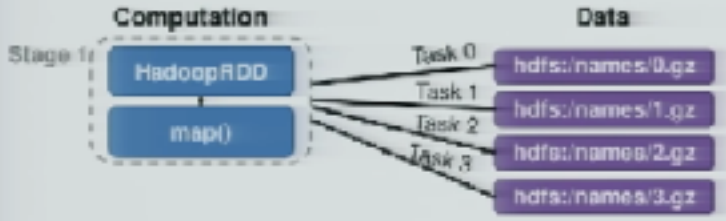
Step 3: Schedule tasks

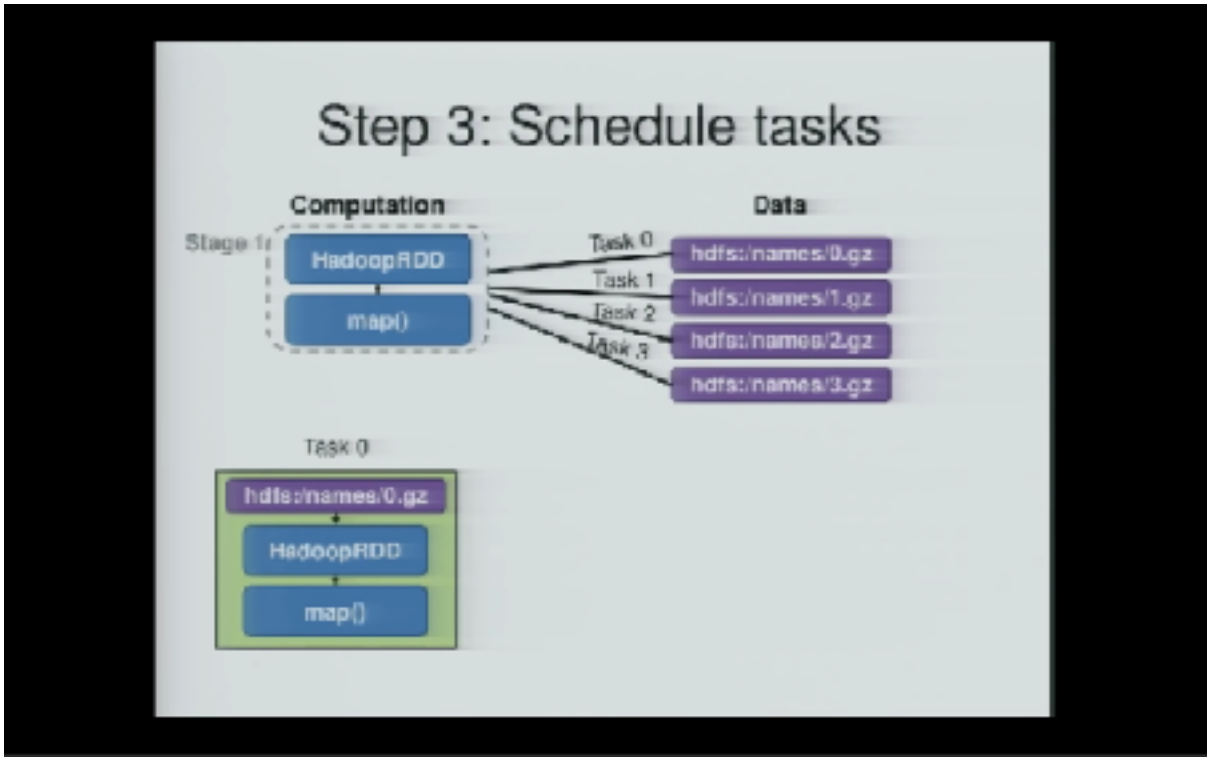


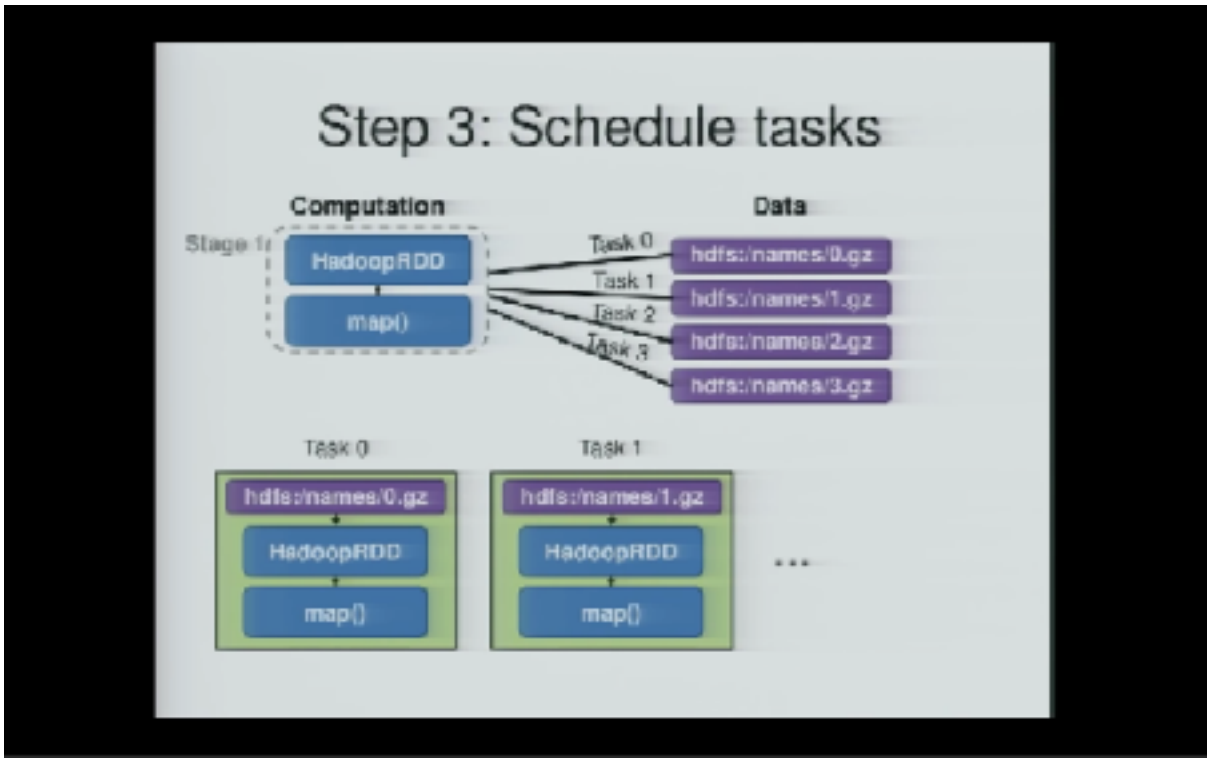
Step 3: Schedule tasks

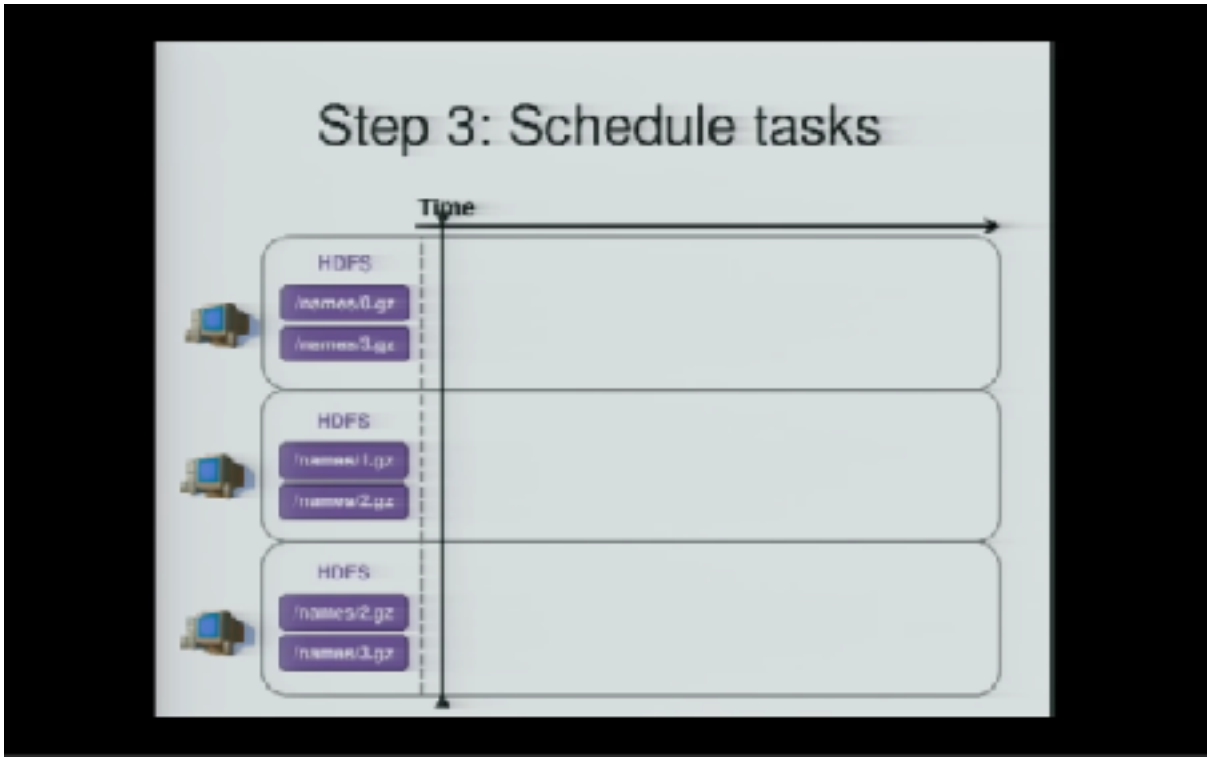


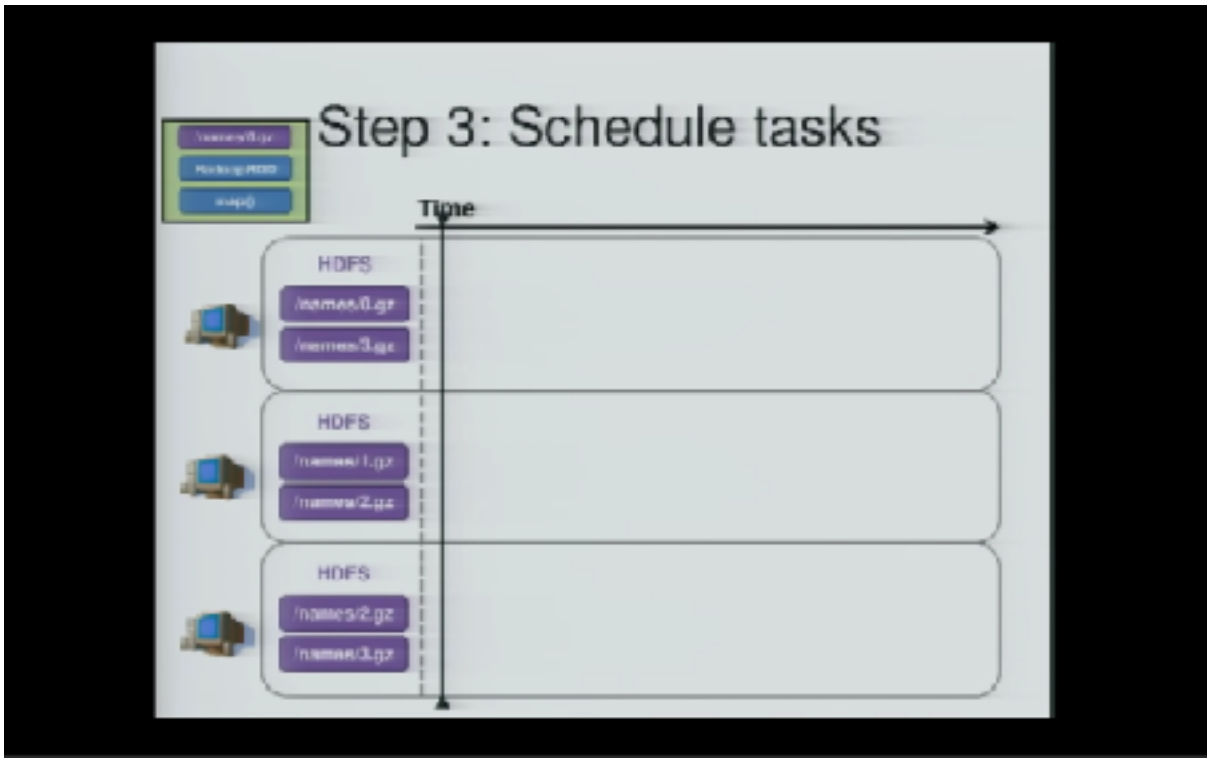
Step 3: Schedule tasks

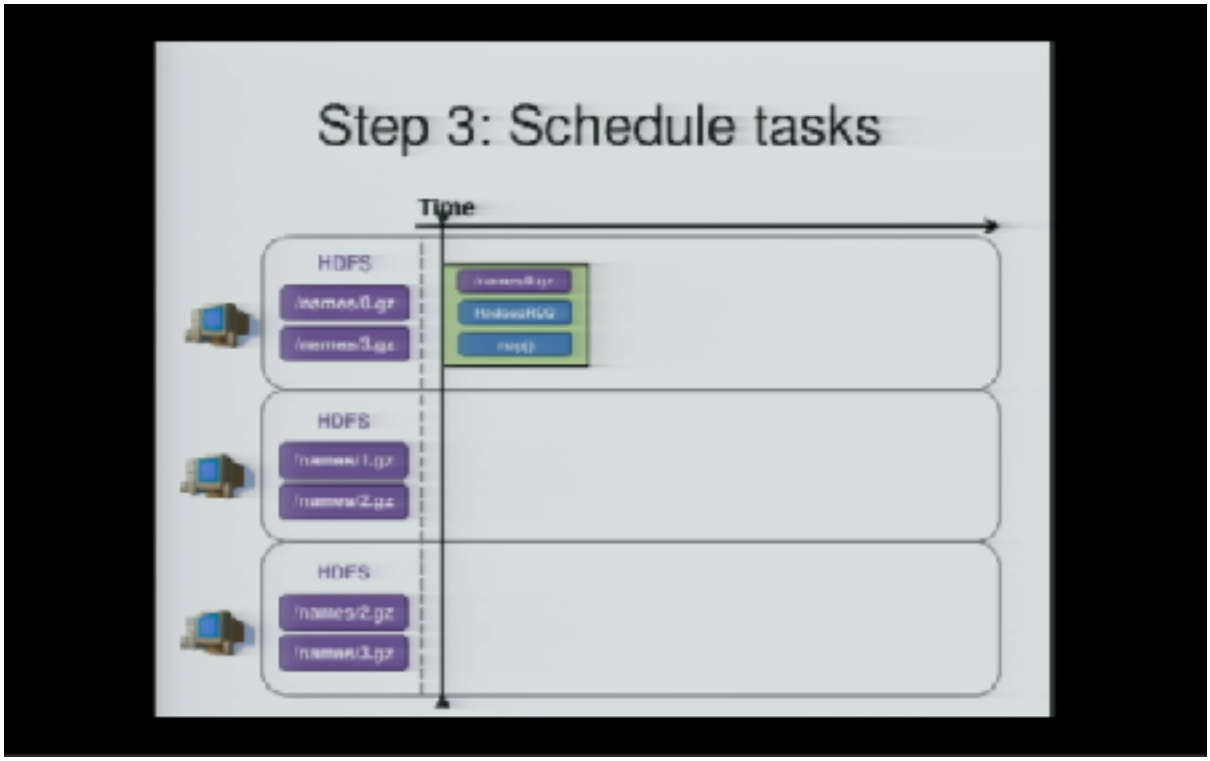


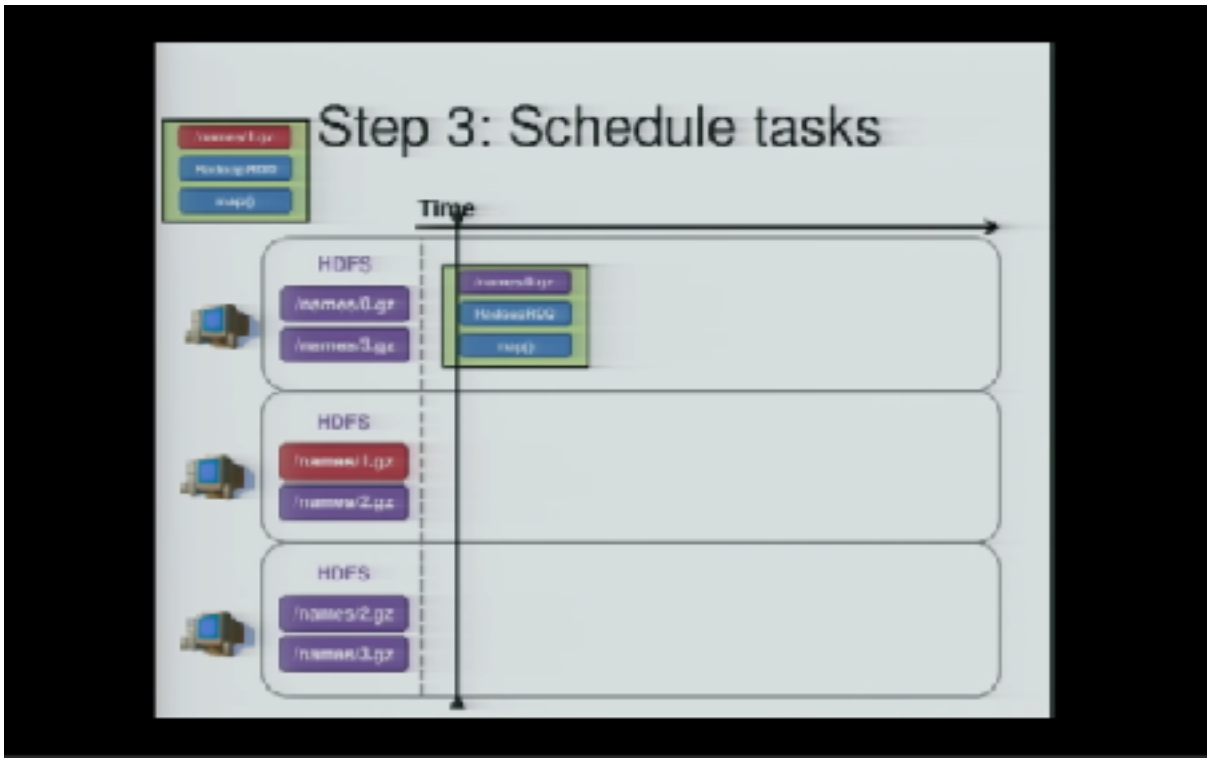

Step 3: Schedule tasks
Computation
Data
Stage 1
HadoopRDD
map()
Task 0
Task 1
Task 2
Task 3
hdfs://names/0.gz
hdfs://names/1.gz
hdfs://names/2.gz
hdfs://names/3.gz
Task 0
hdfs://names/0.gz
HadoopRDD
map()

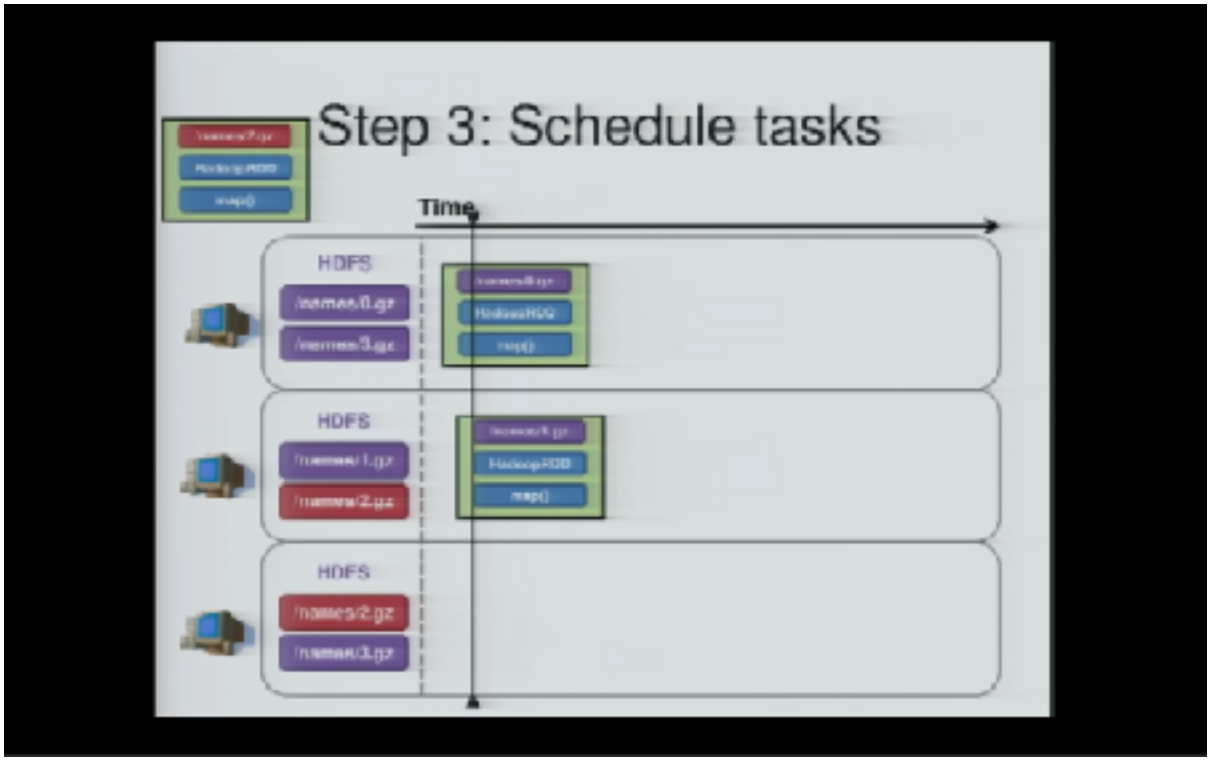

Step 3: Schedule tasks
Computation
Data
Stage 1
HadoopRDD
map()
Task 0
Task 1
Task 2
Task 3
hdfs://names/0.gz
hdfs://names/1.gz
hdfs://names/2.gz
hdfs://names/3.gz
Task 0
hdfs://names/0.gz
HadoopRDD
map()
Task 1
hdfs://names/1.gz
HadoopRDD
map()
...

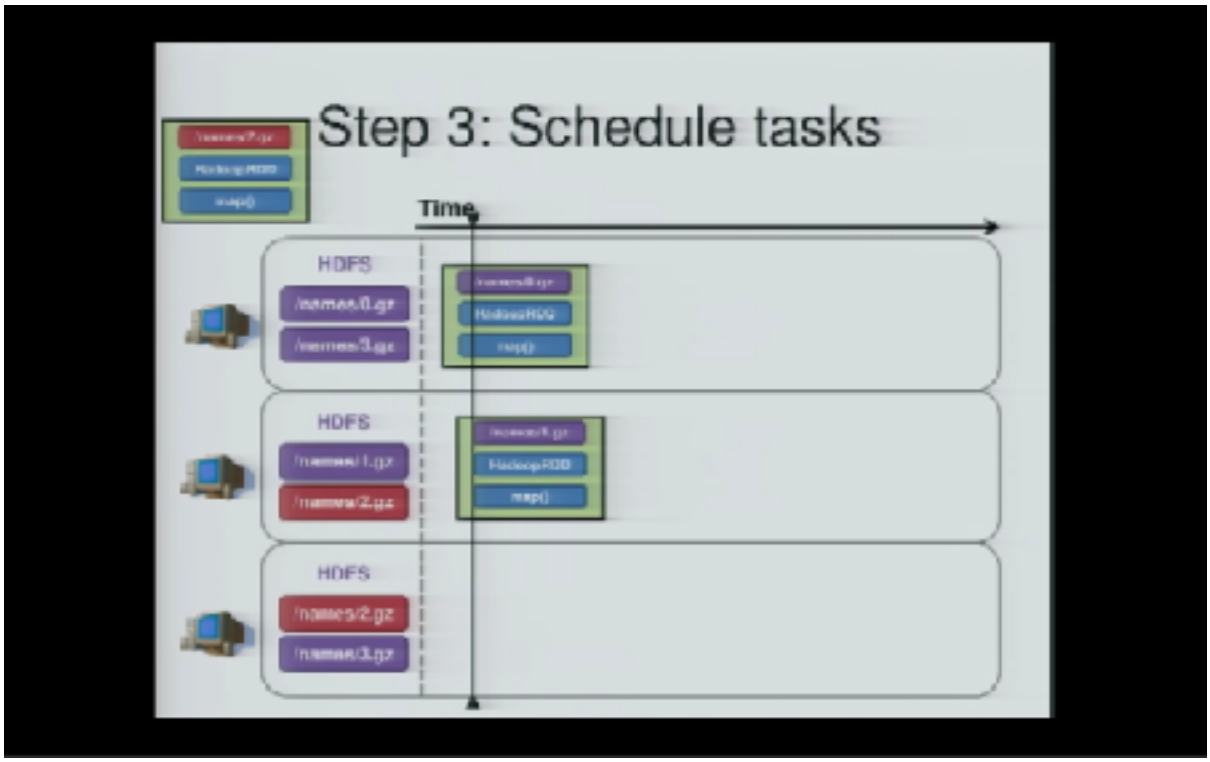

Step 3: Schedule tasks
Time
HDFS
HDFS
HDFS

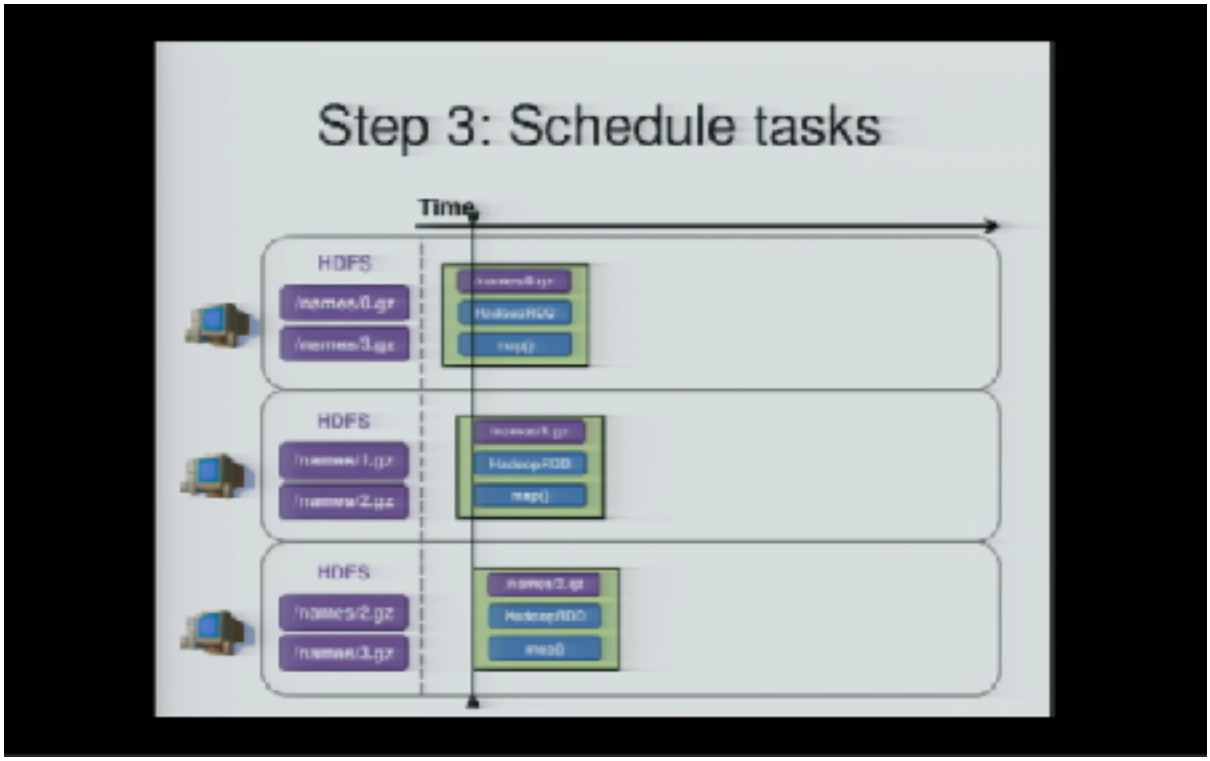

Step 3: Schedule tasks
Time
HDFS
HDFS
HDFS

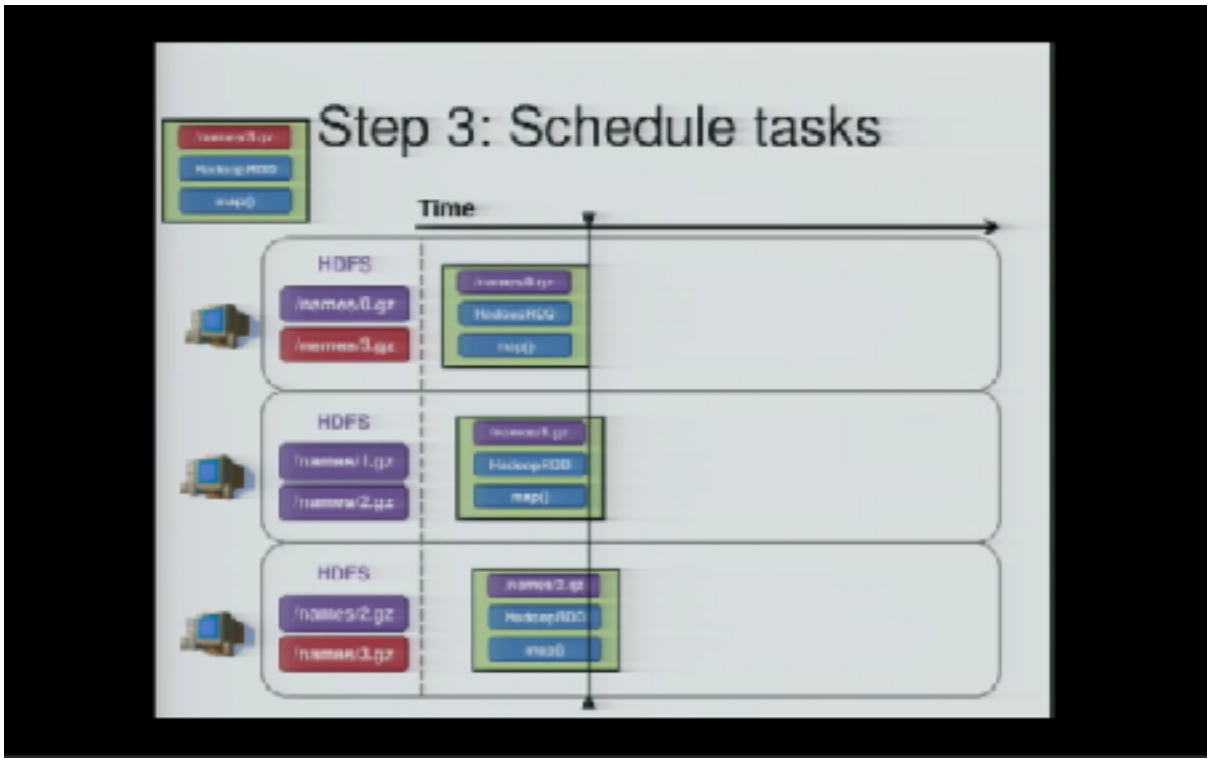

Step 3: Schedule tasks
Time
HDFS
HDFS
HDFS

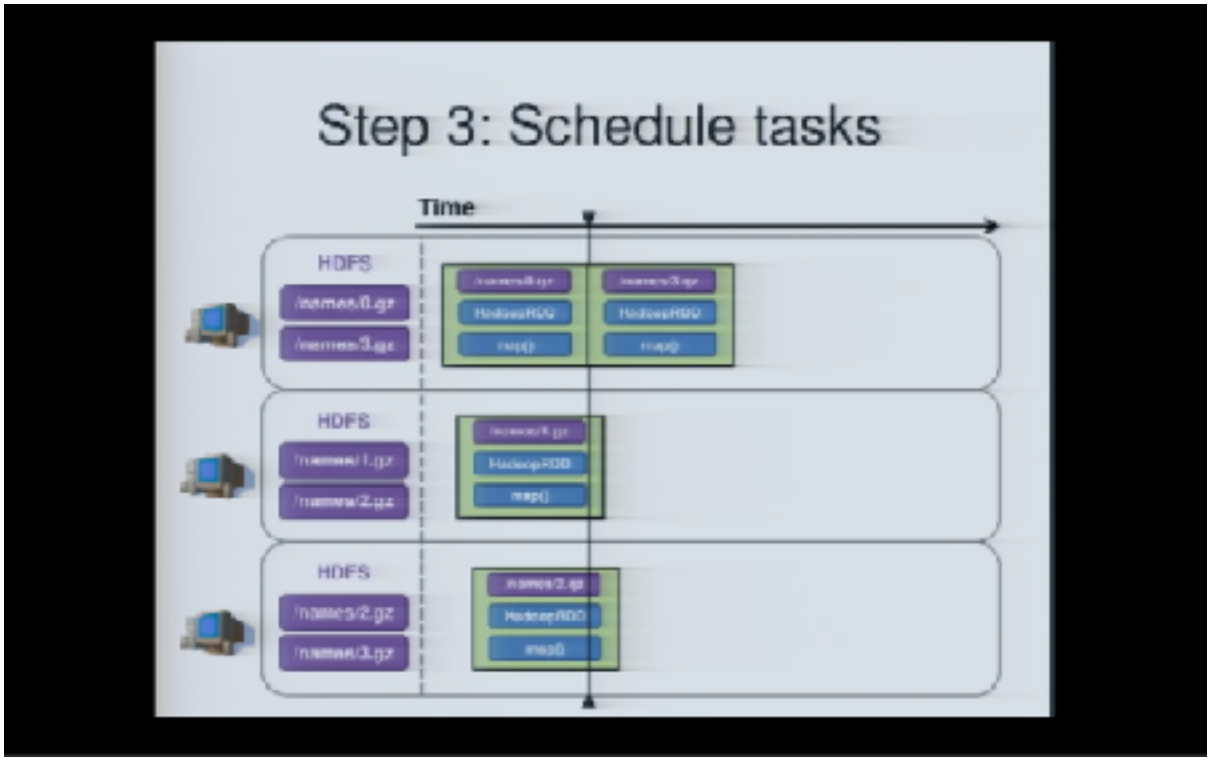

Step 3: Schedule tasks
Time
HDFS
HDFS
HDFS

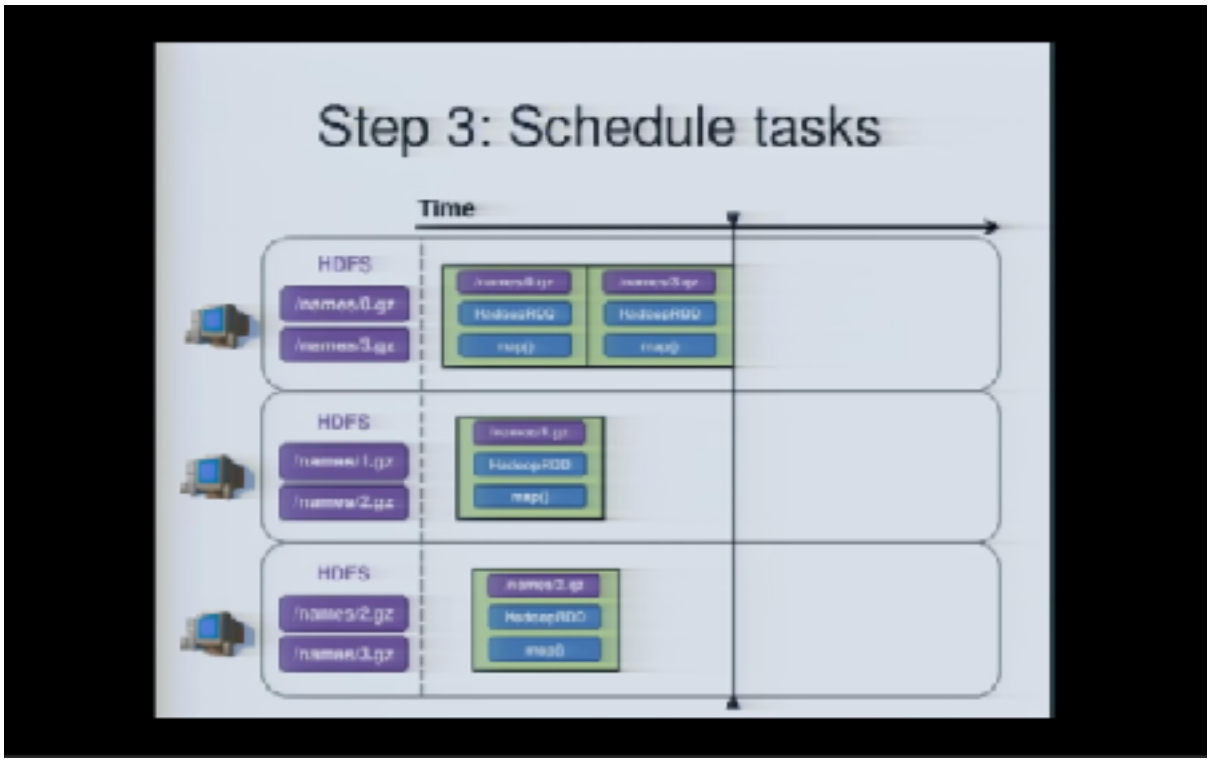

Step 3: Schedule tasks
Time
HDFS
HDFS
HDFS


Step 3: Schedule tasks
Time
HDFS
HDFS
HDFS


Step 3: Schedule tasks
Time
HDFS
HDFS
HDFS

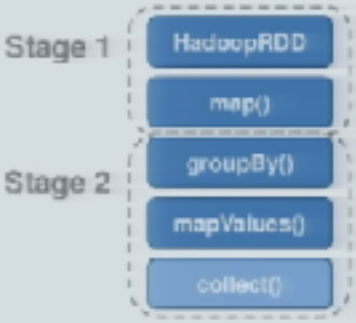

Step 3: Schedule tasks
Time
HDFS
HDFS
HDFS


Step 3: Schedule tasks
Time
HDFS
HDFS
HDFS


Step 3: Schedule tasks
Time
HDFS
HDFS
HDFS

Step 3: Schedule tasks

The Shuffle



The Shuffle

- Redistributes data among partitions

The Shuffle

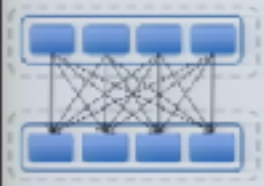
- Redistributes data among partitions
- Hash keys into buckets

The same hashcode contains keys will be moving to same partition/bucket

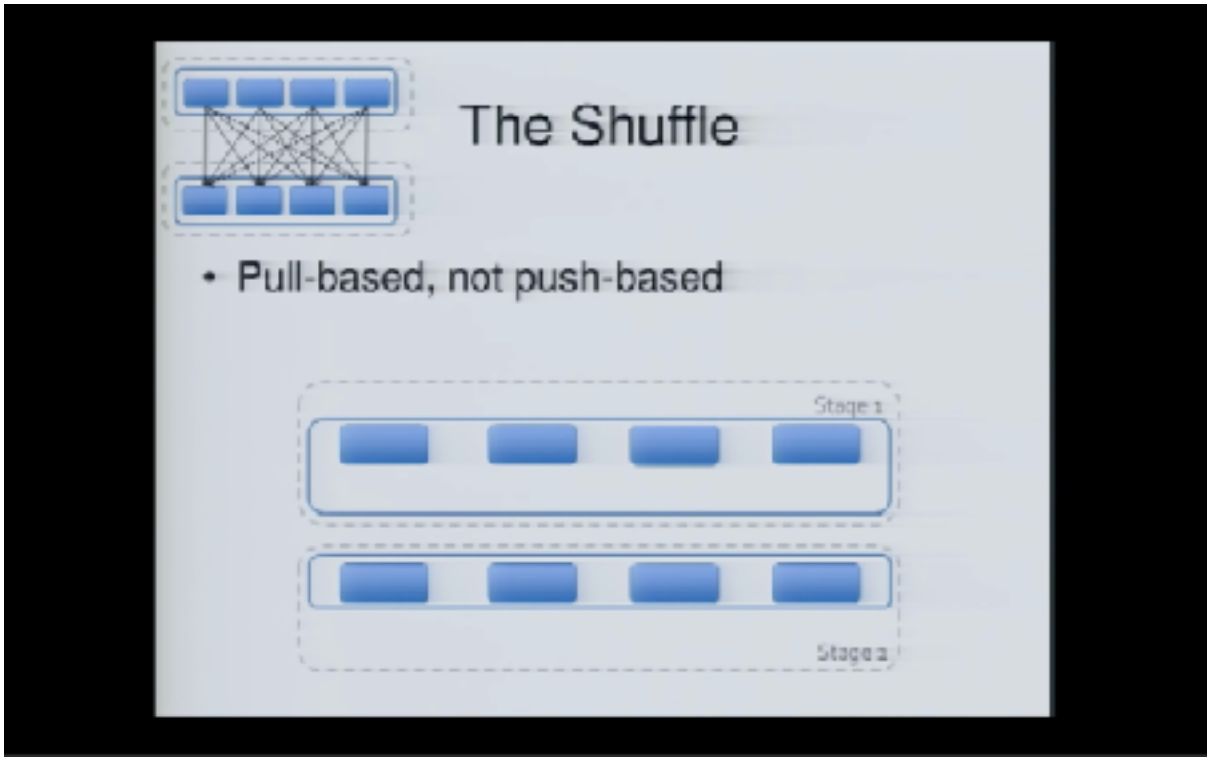
A Deeper Understanding of Spark Internals - Aaron Davidson (Databricks)

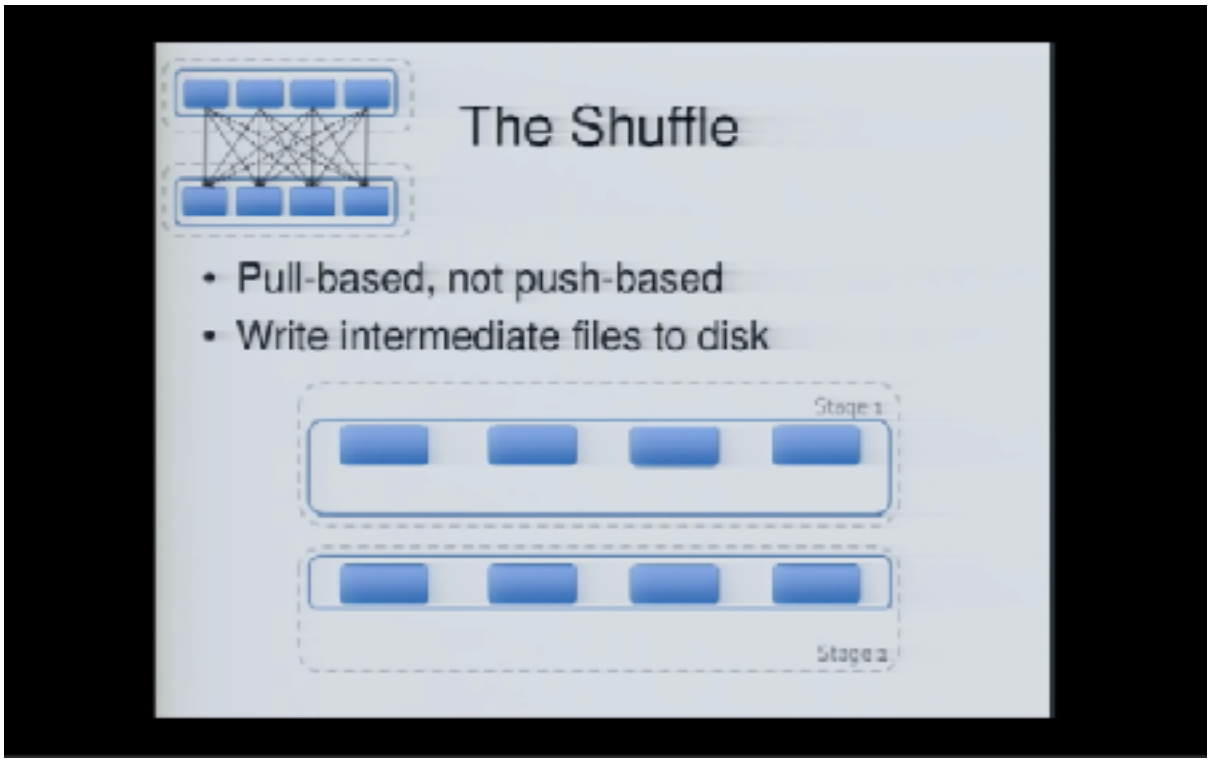
The Shuffle

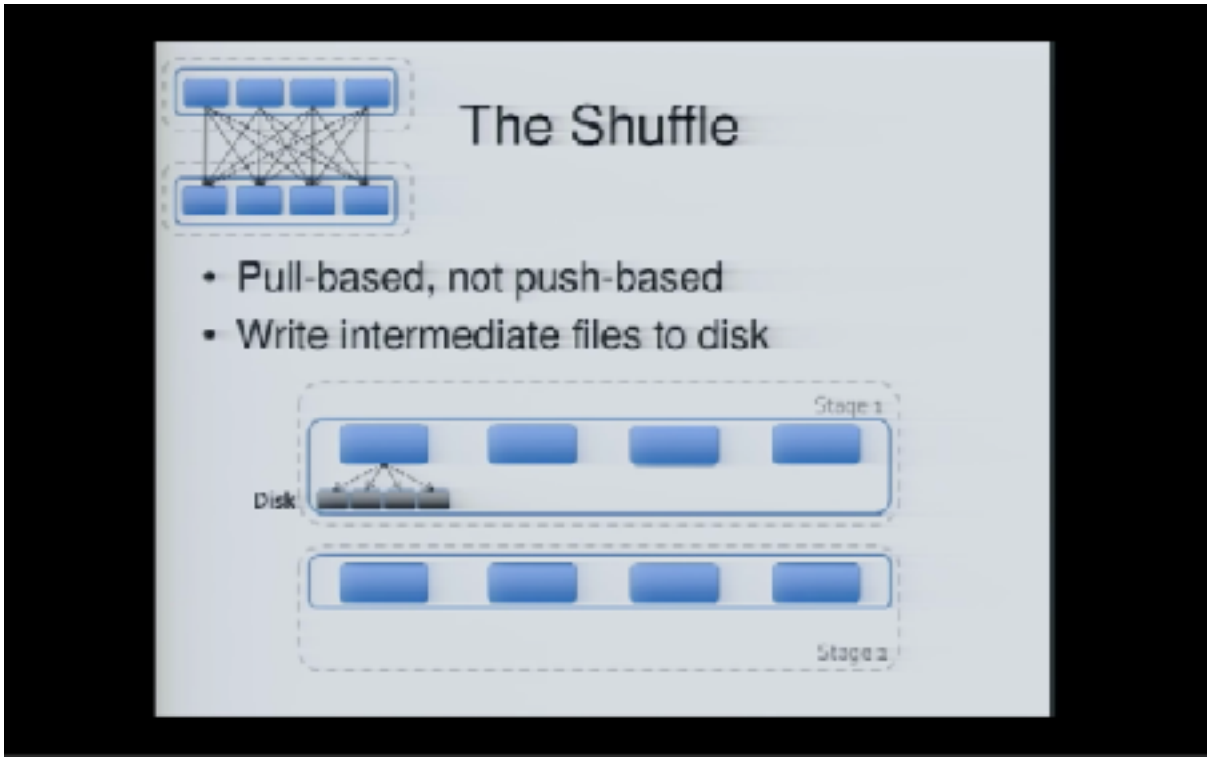
- Redistributes data among partitions
- Hash keys into buckets
- Optimizations:
  - Avoided when possible, if data is already properly partitioned
  - Partial aggregation reduces data movement

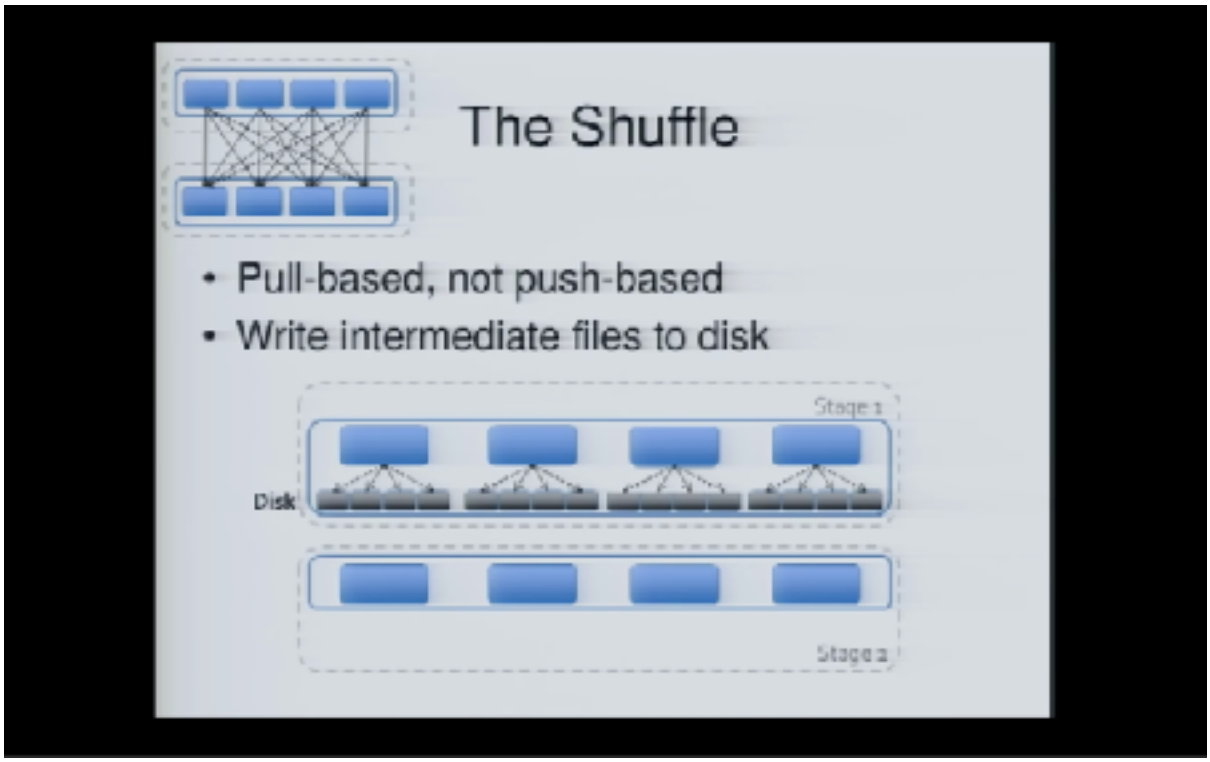


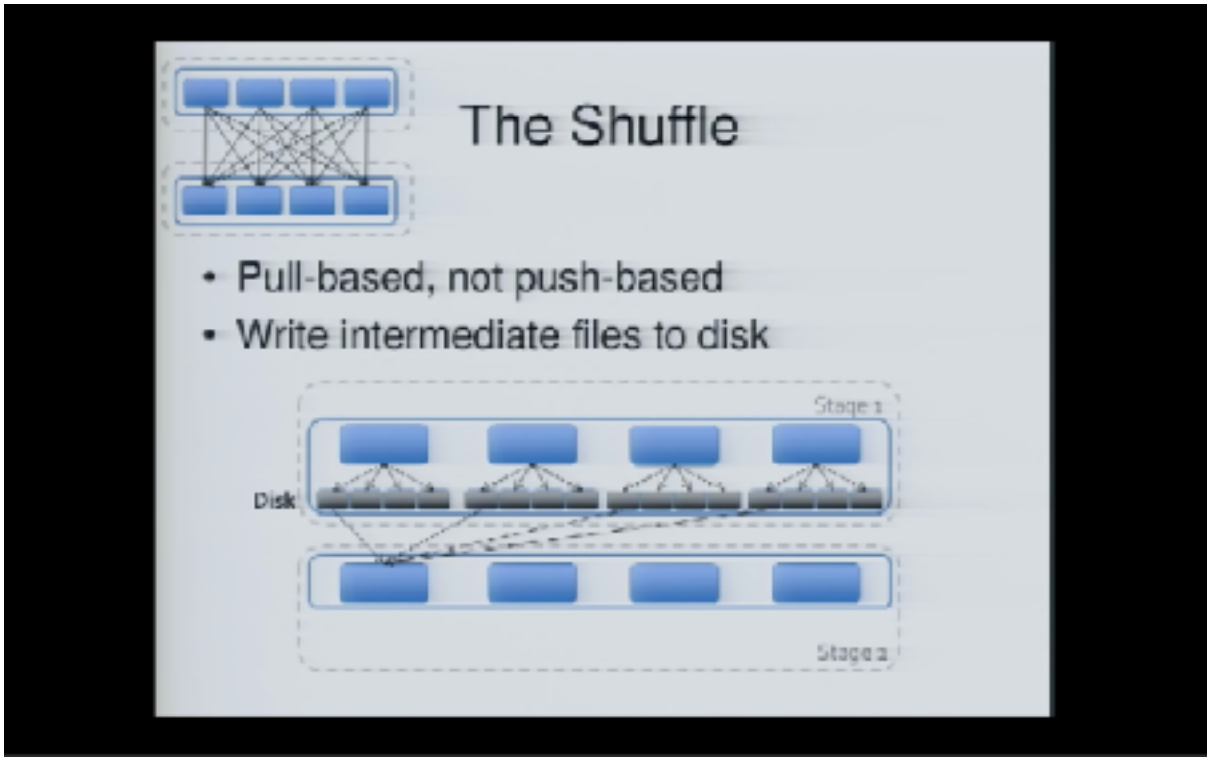
The Shuffle

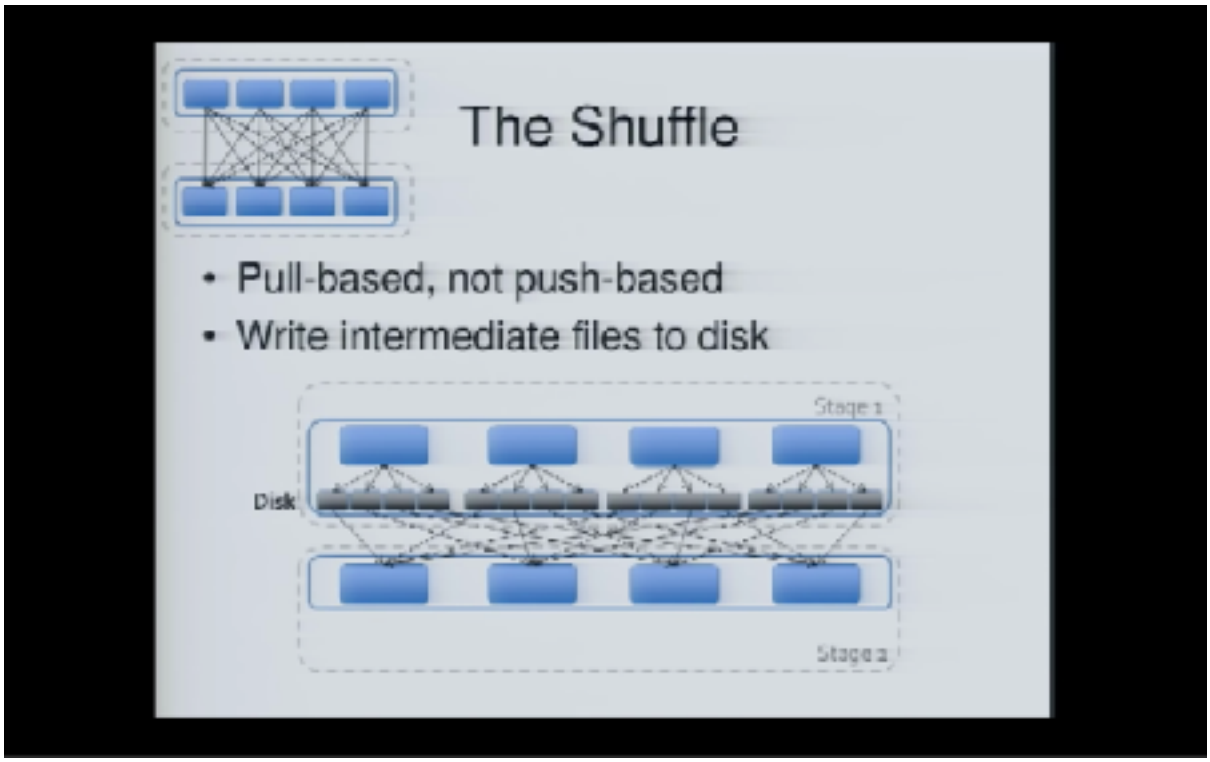

The Shuffle
• Pull-based, not push-based
Stage 1
Stage 2


The Shuffle
• Pull-based, not push-based
• Write intermediate files to disk
Stage 1
Stage 2


The Shuffle
• Pull-based, not push-based
• Write intermediate files to disk
Stage 1
Disk
Stage 2


The Shuffle
• Pull-based, not push-based
• Write intermediate files to disk
Stage 1
Disk
Stage 2

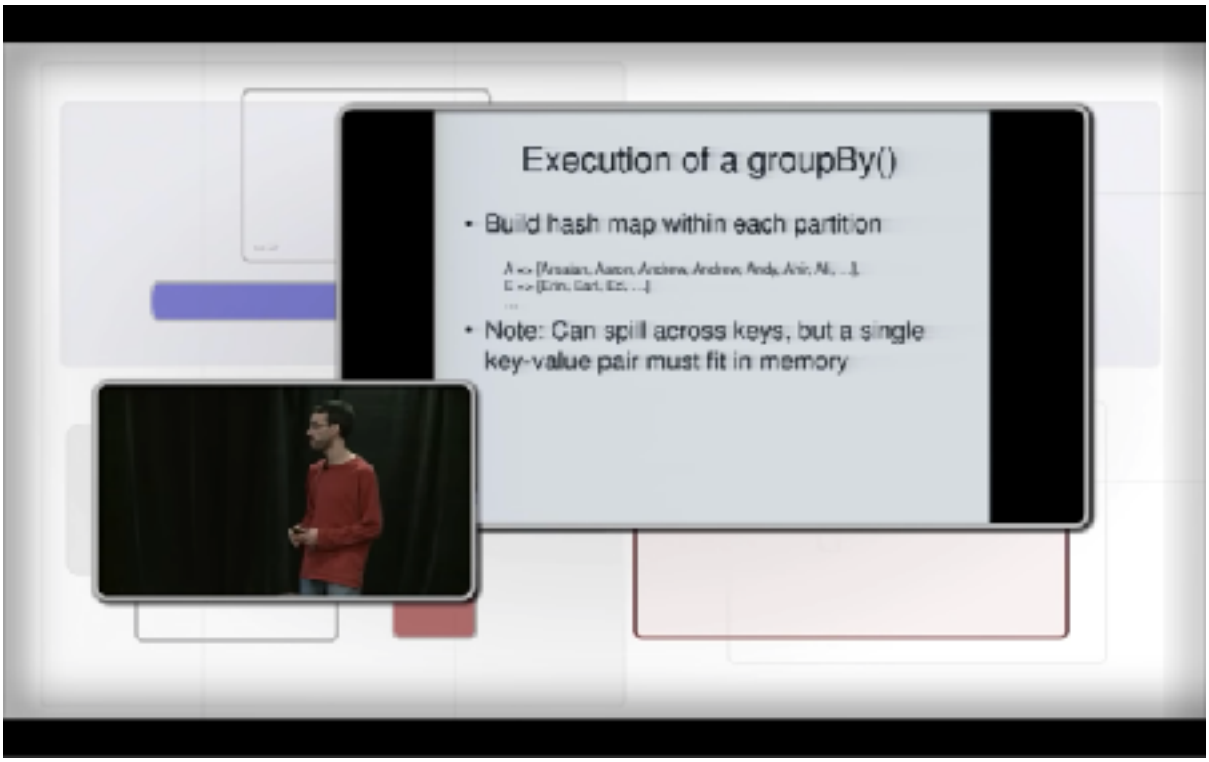

The Shuffle
• Pull-based, not push-based
• Write intermediate files to disk
Stage 1
Disk
Stage 2

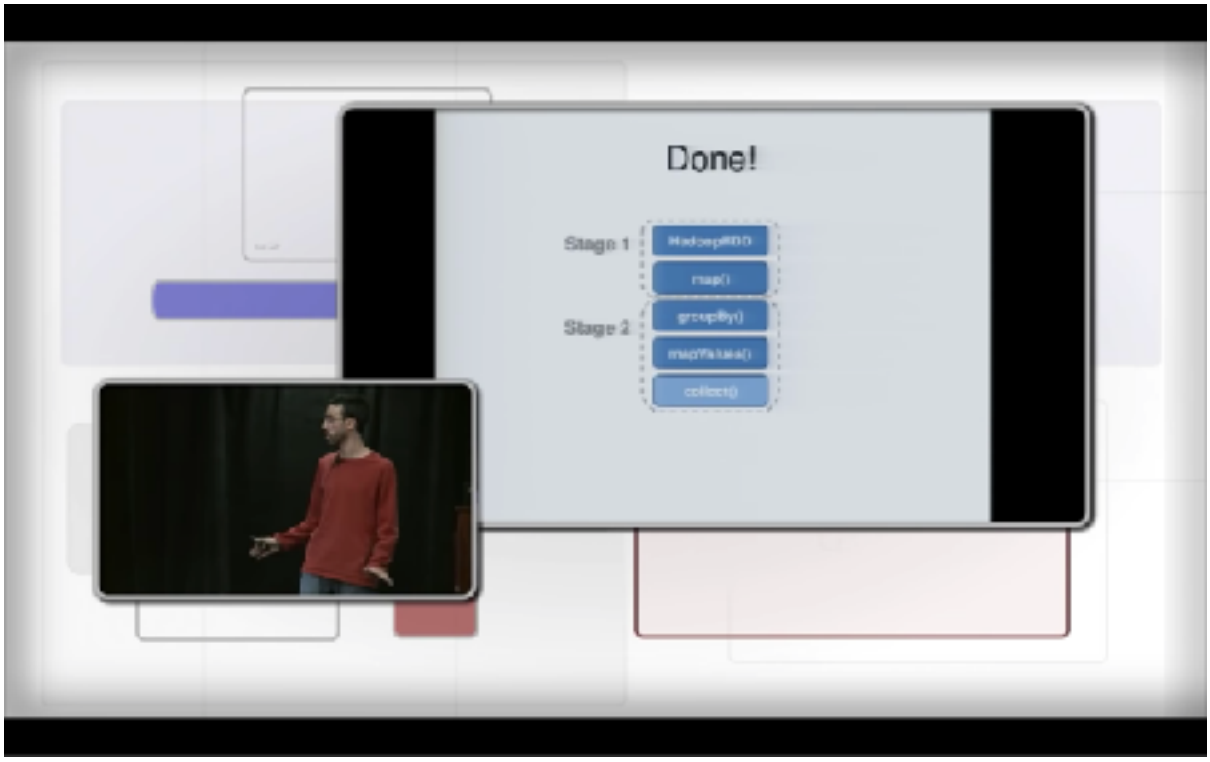

The Shuffle
• Pull-based, not push-based
• Write intermediate files to disk
Stage 1
Disk
Stage 2

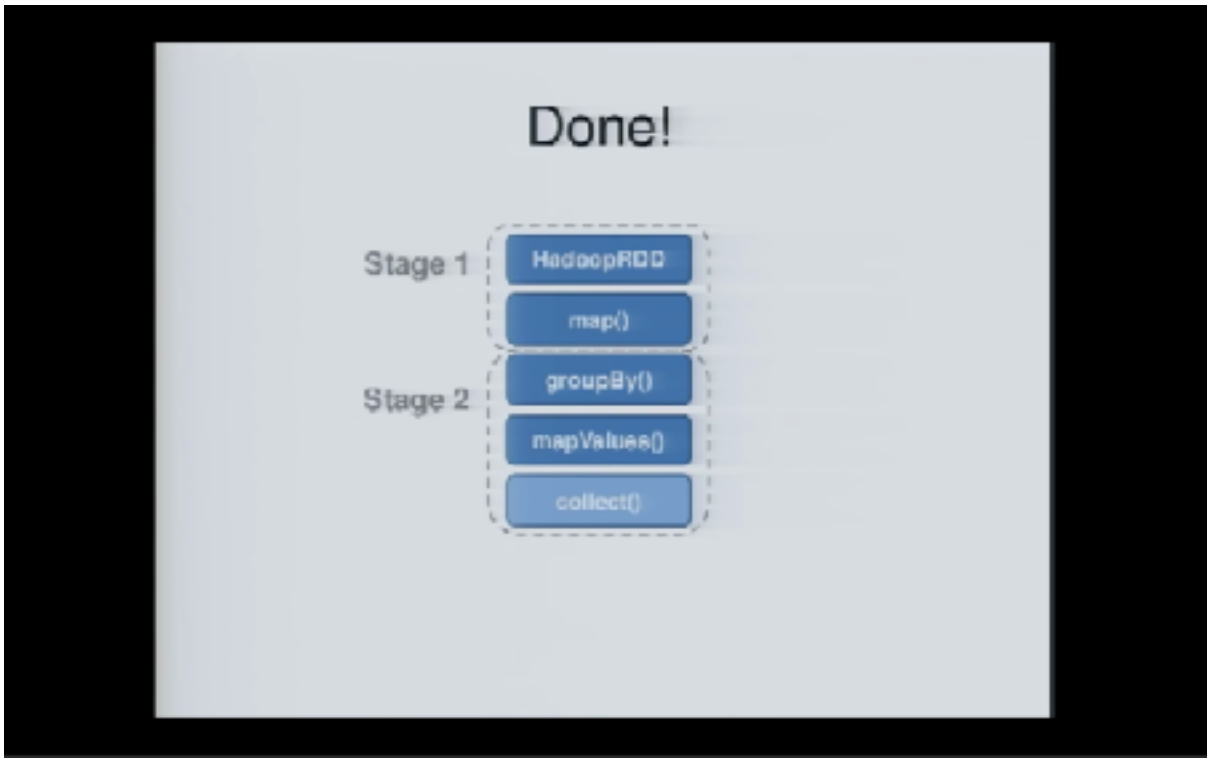
Execution of a groupBy()

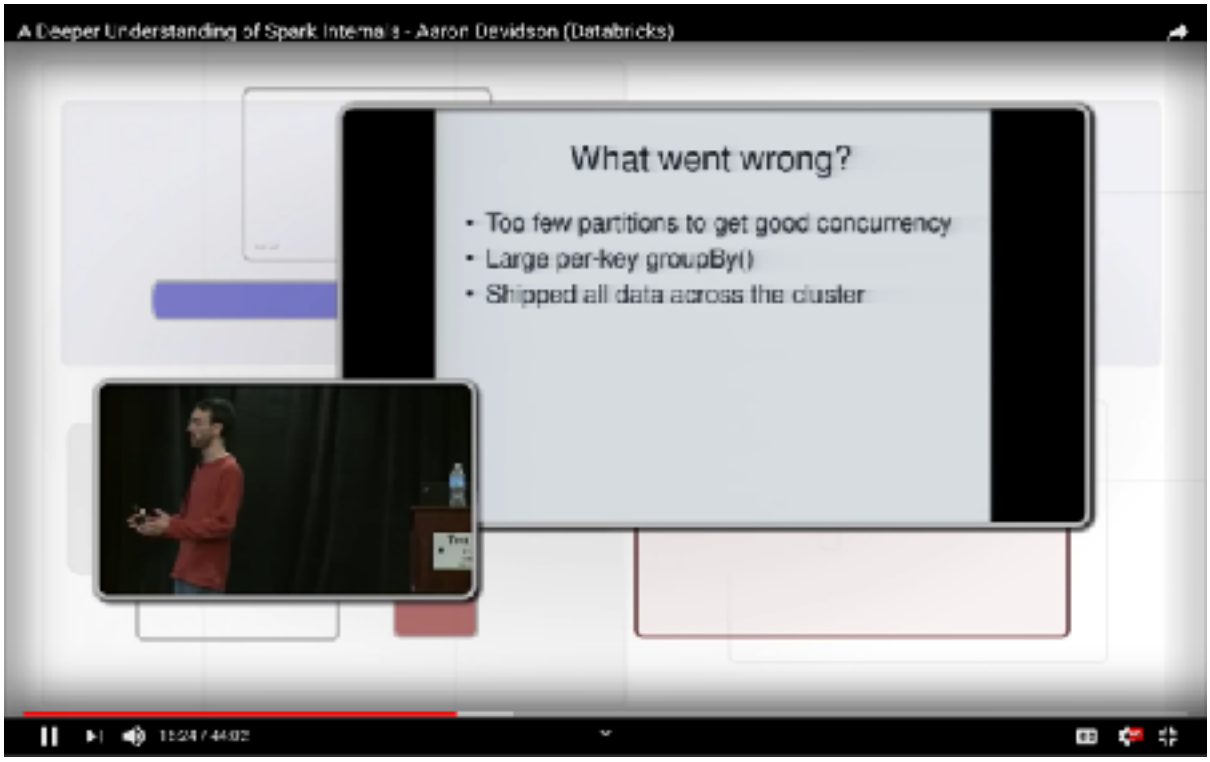
- Build hash map within each partition

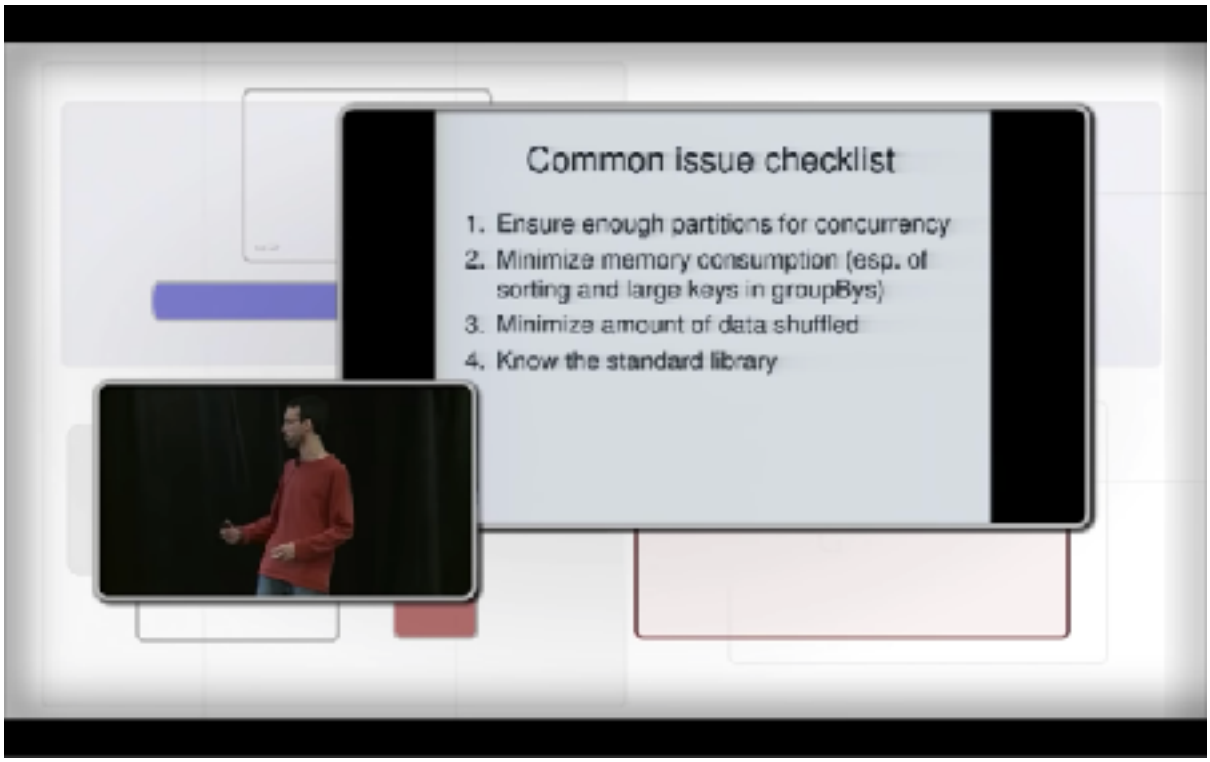
A => [Arsalan, Aaron, Andrew, Andrew, Andy, Ahir, Ali, …],
E => [Erin, Earl, Ed, …]
…




Done!
Stage 1
Stage 2


Done!
Stage 1
HadoopRDD
map()
Stage 2
groupBy()
mapValues()
collect()


A Deeper Understanding of Spark Internals - Aaron Davidson (Databricks)
What went wrong?
• Too few partitions to get good concurrency
• Large per-key groupBy()
• Shipped all data across the cluster


Common issue checklist
1. Ensure enough partitions for concurrency
2. Minimize memory consumption (esp. of sorting and large keys in groupBys)
3. Minimize amount of data shuffled
4. Know the standard library

Common issue checklist

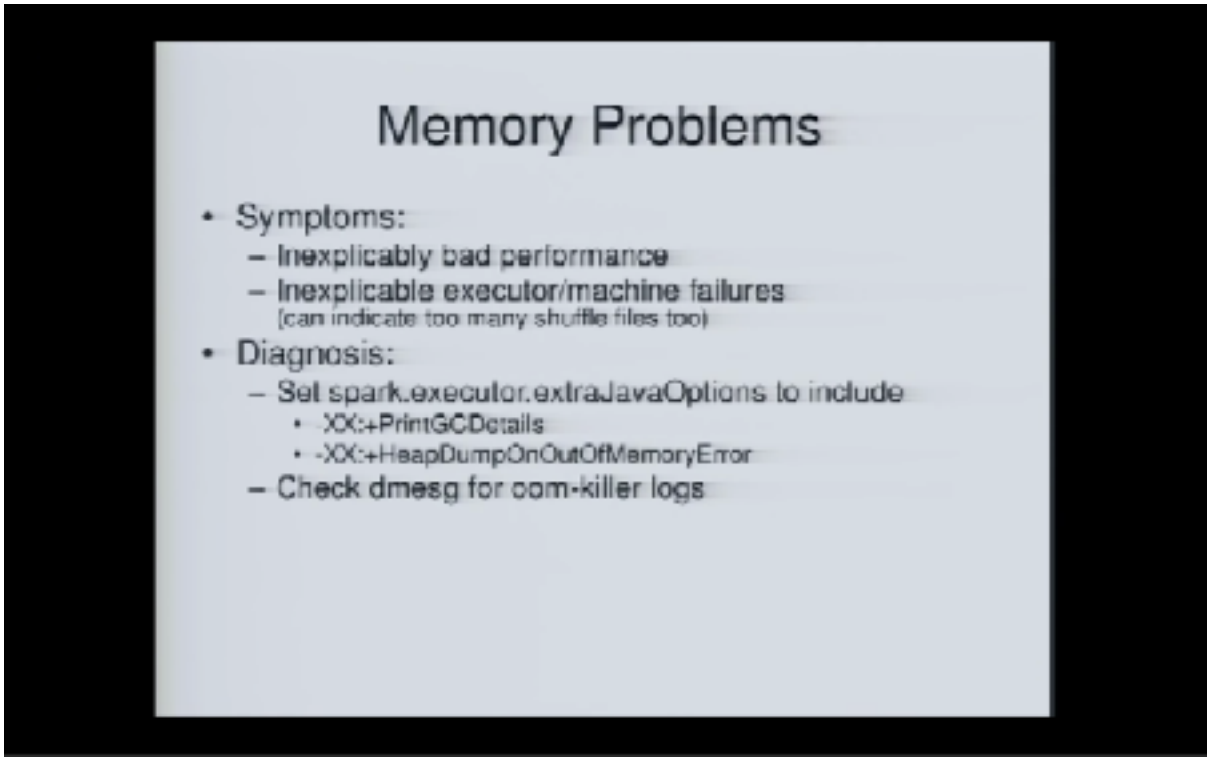
1. Ensure enough partitions for concurrency
2. Minimize memory consumption (esp. of sorting and large keys in groupBys)
3. Minimize amount of data shuffled
4. Know the standard library

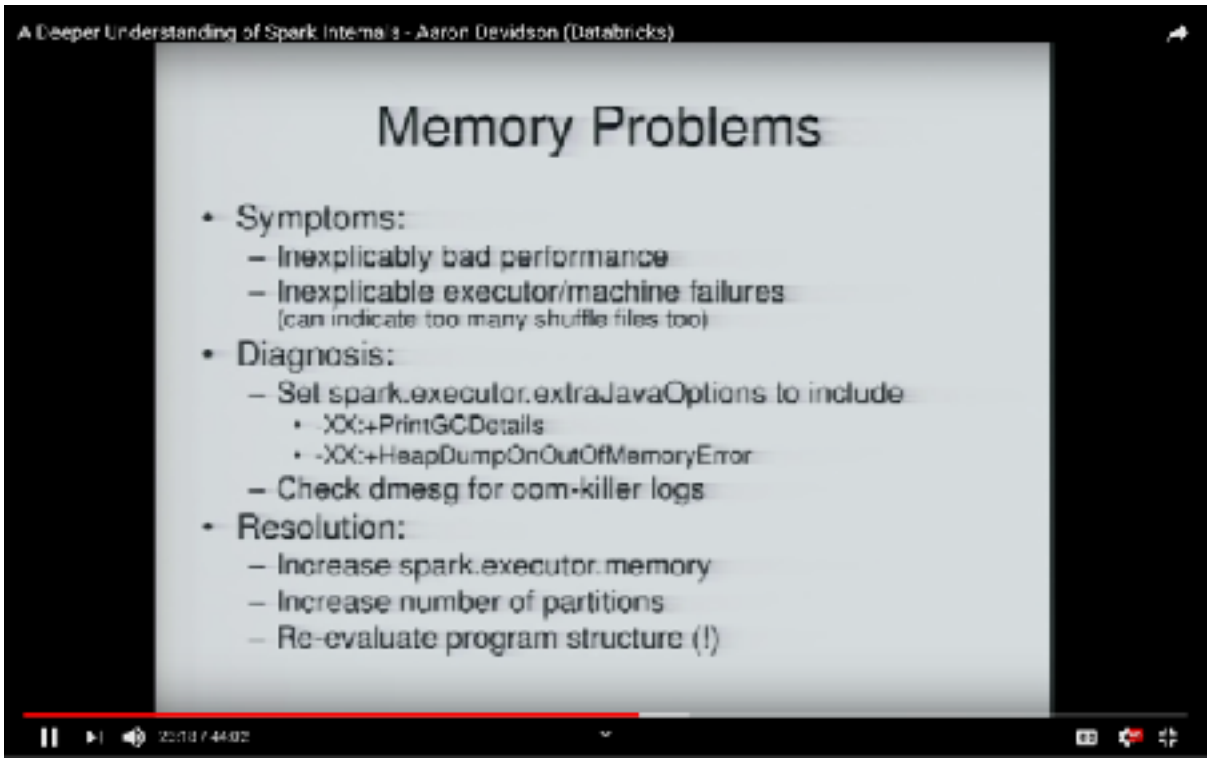
1 & 2 are about tuning number of partitions!

Importance of Partition Tuning

- Main issue: too few partitions
  - Less concurrency
  - More susceptible to data skew
  - Increased memory pressure for groupBy, reduceByKey, sortByKey, etc.
- Secondary issue: too many partitions
- Need "reasonable number" of partitions
  - Commonly between 100 and 10,000 partitions
  - Lower bound: At least ~2x number of cores in cluster
  - Upper bound: Ensure tasks take at least 100ms

Susceptible : capable / admitting of





Inexplicablbe: unable to be explained

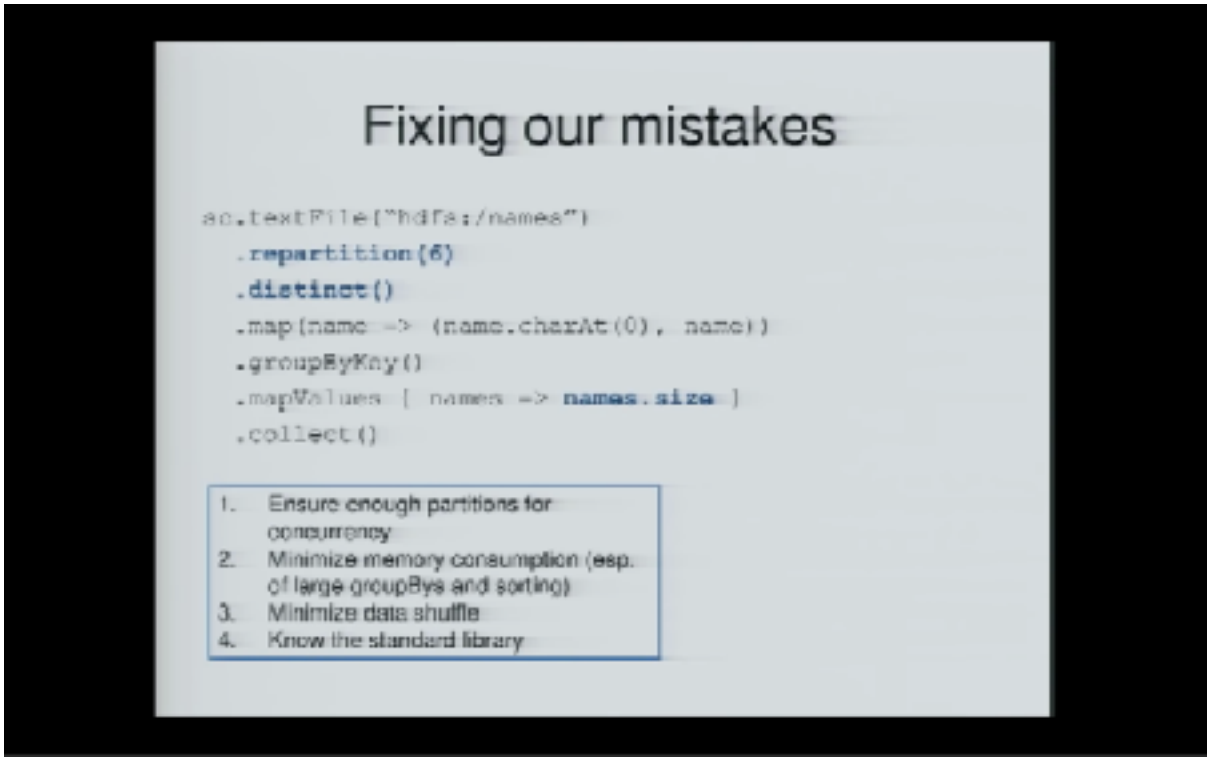
Adding repatition & distict:

Original:

scala> sc.textFile("/tmp/names.txt").map(name =>(name.charAt(0),name)).groupByKey().mapValues(names => names.toSet.size).collect().foreach(println)

Modified:

```
scala> sc.textFile("/tmp/names.txt").repartition(6).distinct().map(name
=>(name.charAt(0),name)).groupByKey().mapValues(names =>
names.toSet.size).collect().foreach(println)
```




```
scala> sc.textFile("/tmp/names.txt").distinct(numPartitions=6).map(name
=>(name.charAt(0),name)).groupByKey().mapValues(names =>
names.size).collect().foreach(println)
```

Fixing our mistakes

```
sc.textFile("hdfs://names")
  .distinct(numPartitions = 6)
  .map(name => (name.charAt(0), name))
  .groupByKey()
  .mapValues { names => names.size }
  .collect()
```

1. Ensure enough partitions for concurrency
2. Minimize memory consumption (esp. of large groupBys and sorting)
3. Minimize data shuffle
4. Know the standard library

Fixing our mistakes

```
sc.textFile("hdfs://names")
  .distinct(numPartitions = 6)
  .map(name => (name.charAt(0), 1))
  .reduceByKey(_ + _)
  .collect()
```

1. Ensure enough partitions for concurrency
2. Minimize memory consumption (esp. of large groupBys and sorting)
3. Minimize data shuffle
4. Know the standard library

Spark Joins:

https://www.youtube.com/watch?v=fp53QhSfQcI&t=1146s

Spark SQL Joins

```
SELECT …
FROM TABLE A
JOIN TABLE B
ON A.KEY1 = B.KEY2
```

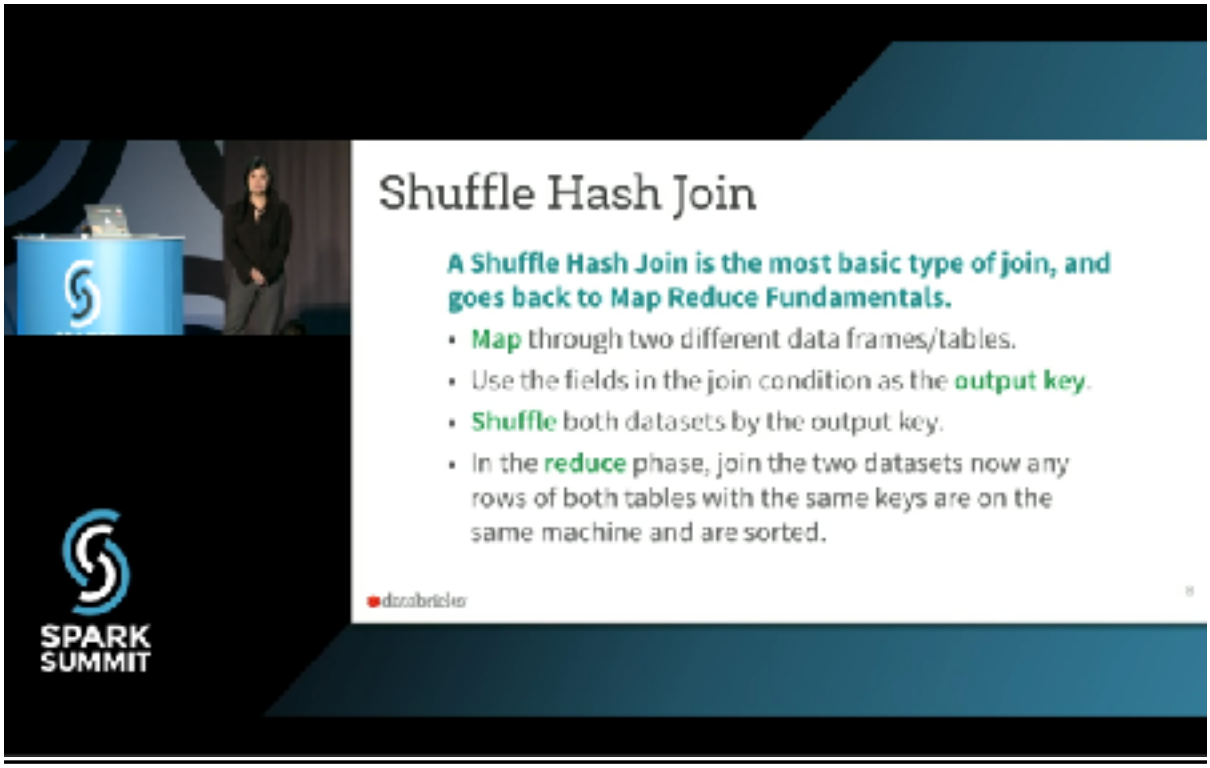
Topics Covered Today

Basic Joins:

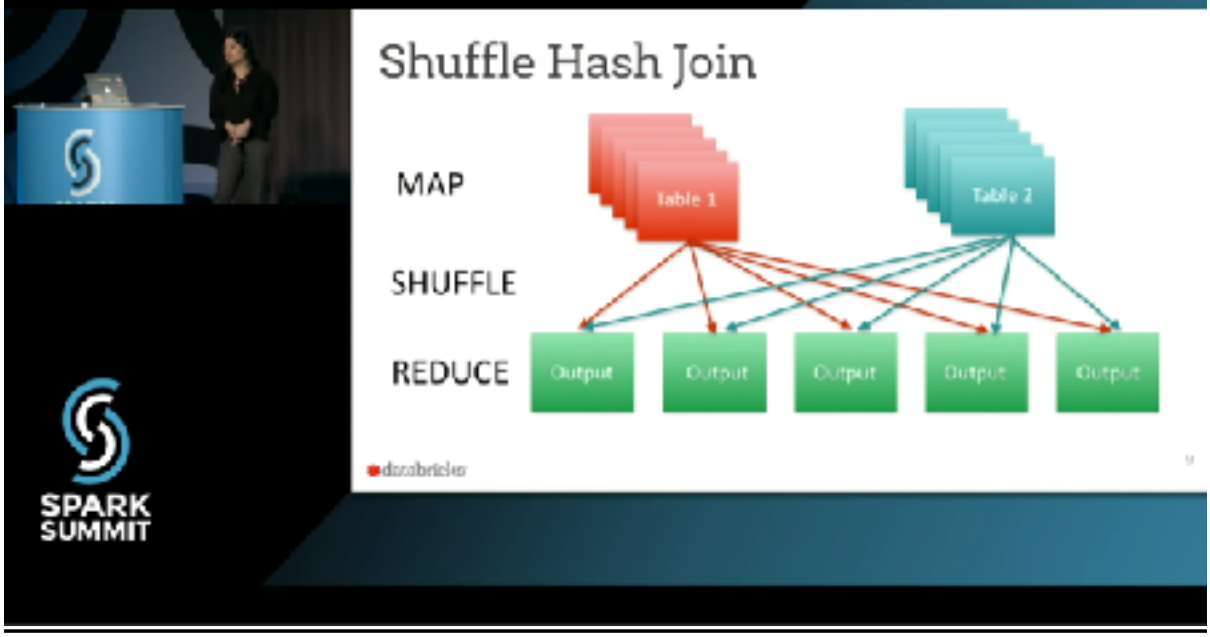
- Shuffle Hash Join
  - Troubleshooting
- Broadcast Hash Join
- Cartesian Join

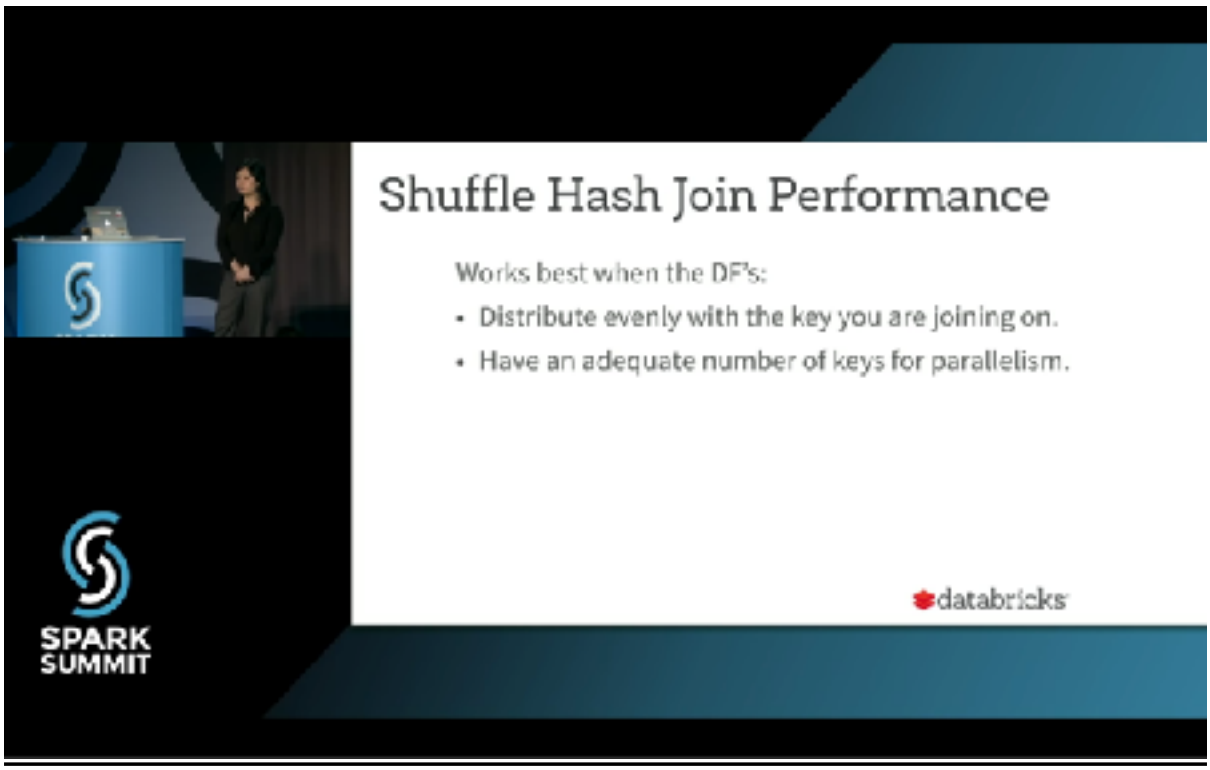
Special Cases:

- Theta Join
- One to Many Join



[Shuffle Hash (SH) Join]


Shuffle Hash Join
MAP
SHUFFLE
REDUCE
Table 1
Table 2
Output
Output
Output
Output
Output
databricks
SPARK
SUMMIT


Shuffle Hash Join Performance
Works best when the DF's:
• Distribute evenly with the key you are joining on.
• Have an adequate number of keys for parallelism.
databricks
SPARK
SUMMIT

SPARK SUMMIT

Shuffle Hash Join Performance

Works best when the DF's:

- Distribute evenly with the key you are joining on.
- Have an adequate number of keys for parallelism.

```
join_rdd = sqlContext.sql("select *
  FROM people_in_the_us
  JOIN states
  ON people_in_the_us.state = states.name")
```

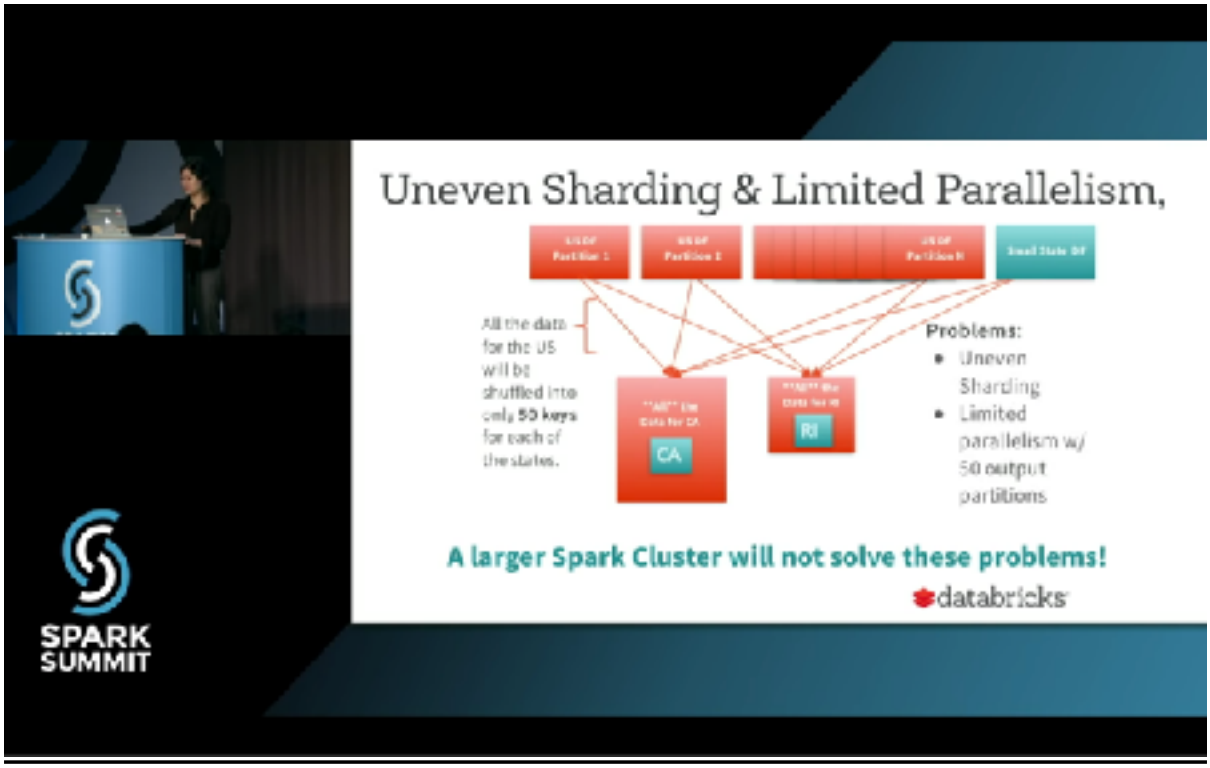
databricks

SPARK SUMMIT

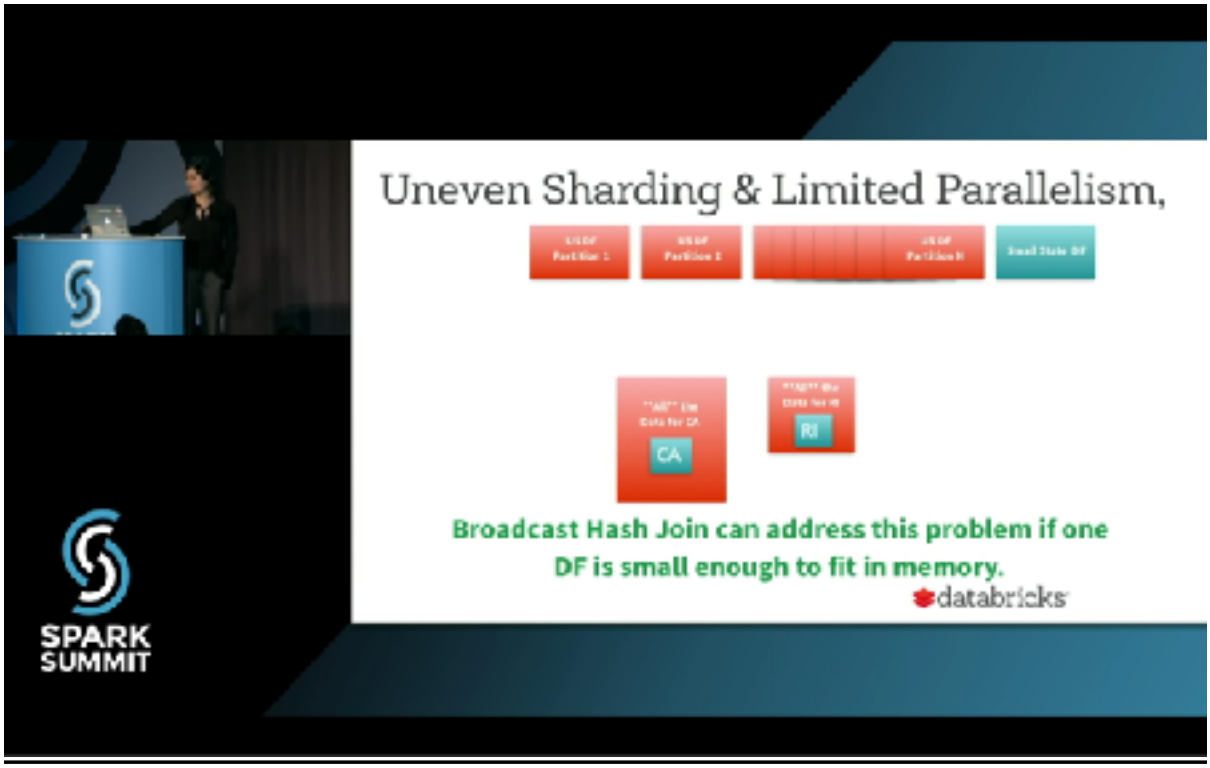
Uneven Sharding & Limited Parallelism,

A larger Spark Cluster will not solve these problems!

databricks



Broadcast Hash (BH) Join



Left Outer Join

All keys will get as output in the final output table

SPARK SUMMIT

More Performance Considerations

```
join_rdd = sqlContext.sql("select *
  FROM people_in_california
  LEFT JOIN all_the_people_in_the_world
  ON people_in_california.id =
     all_the_people_in_the_world.id")
```

Final output keys = # of people in CA, so don't need a huge Spark cluster, right?

databricks

SPARK SUMMIT

Left Join - Shuffle Step

Not a Problem:

- Even Sharding
- Good Parallelism

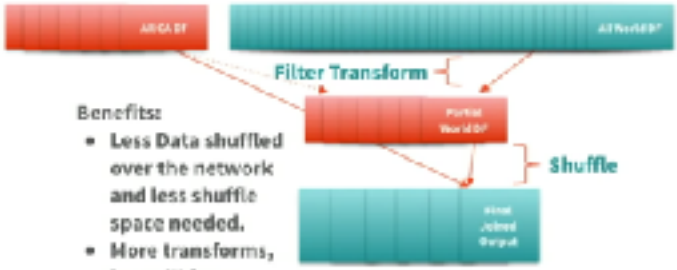
The Size of the Spark Cluster to run this job is limited by the Large table rather than the Medium Sized Table.

databricks

SPARK SUMMIT

A Better Solution

Filter the World DF for only entries that match the CA ID



Benefits:

- Less Data shuffled over the network and less shuffle space needed.
- More transforms, but still faster.

databricks

SPARK SUMMIT

What's the Tipping Point for Huge?

- Can't tell you.

You should understand your data and it's unique properties in order to best optimize your Spark Job.

databricks

What's the Tipping Point for Huge?

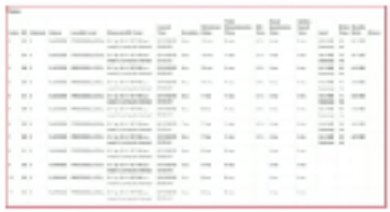
- Can't tell you.
- There aren't always strict rules for optimizing.
- If you were only considering two small columns from the World RDD in Parquet format, the filtering step may not be worth it.

You should understand your data and it's unique properties in order to best optimize your Spark Job.

SPARK SUMMIT

databricks

In Practice: Detecting Shuffle Problems



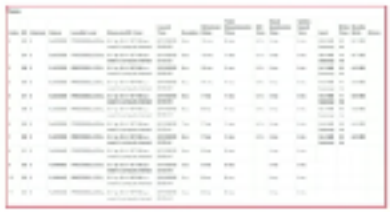
Check the Spark UI pages for task level detail about your Spark job.

SPARK SUMMIT

databricks

SPARK SUMMIT

In Practice: Detecting Shuffle Problems



Check the Spark UI pages for task level detail about your Spark job.

Things to Look for:

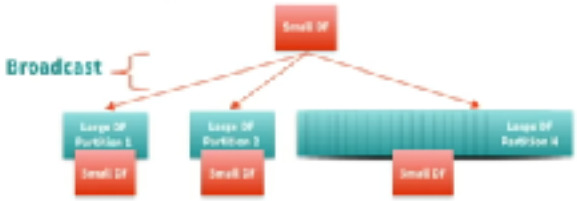
- Tasks that take much longer to run than others.
- Speculative tasks that are launching.
- Shards that have a lot more input or shuffle output.

databricks

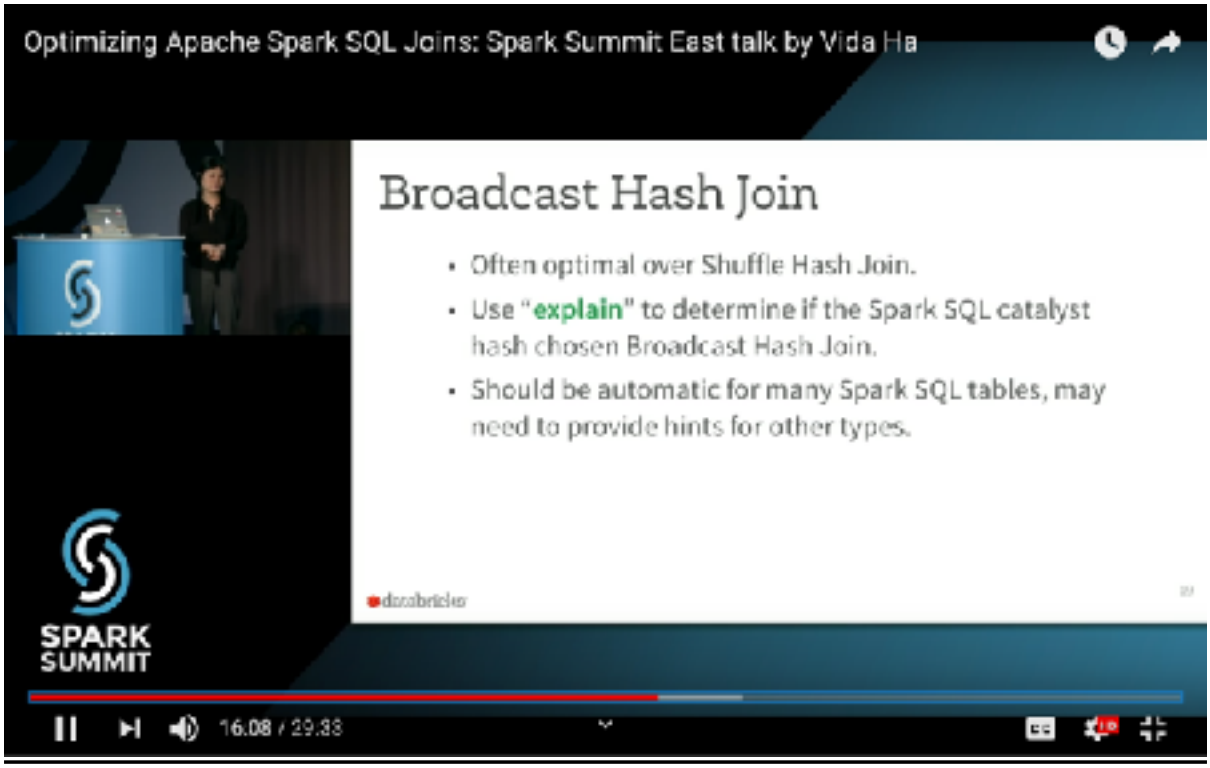
SPARK SUMMIT

Broadcast Hash Join

Optimization: When one of the DF's is small enough to fit in memory on a single machine.



databricks


Optimizing Apache Spark SQL Joins: Spark Summit East talk by Vida Ha
Broadcast Hash Join
• Often optimal over Shuffle Hash Join.
• Use "explain" to determine if the Spark SQL catalyst hash chosen Broadcast Hash Join.
• Should be automatic for many Spark SQL tables, may need to provide hints for other types.
SPARK SUMMIT
16:08 / 29:33


One To Many Join
• A single row on one table can map to many rows on the 2nd table.
• Can explode the number of output rows.
• Not a problem if you use parquet - the size of the output files is not that much since the duplicate data encodes well.
SPARK SUMMIT

[Theta Join]

Theta Join

- Spark SQL consider each keyA against each keyB in the example above and loop to see if the theta condition is met.
- Better Solution - create buckets for keyA and KeyB can be matched on.

SPARK SUMMIT

Theta Join

```
join_rdd = sqlContext.sql("select *
  FROM tableA
  JOIN tableB
  ON (keyA < keyB + 10)")
```

- Spark SQL consider each keyA against each keyB in the example above and loop to see if the theta condition is met.
- Better Solution - create buckets for keyA and KeyB can be matched on.

SPARK SUMMIT

Partitions – Right Sizing – Shuffle – Master Equation

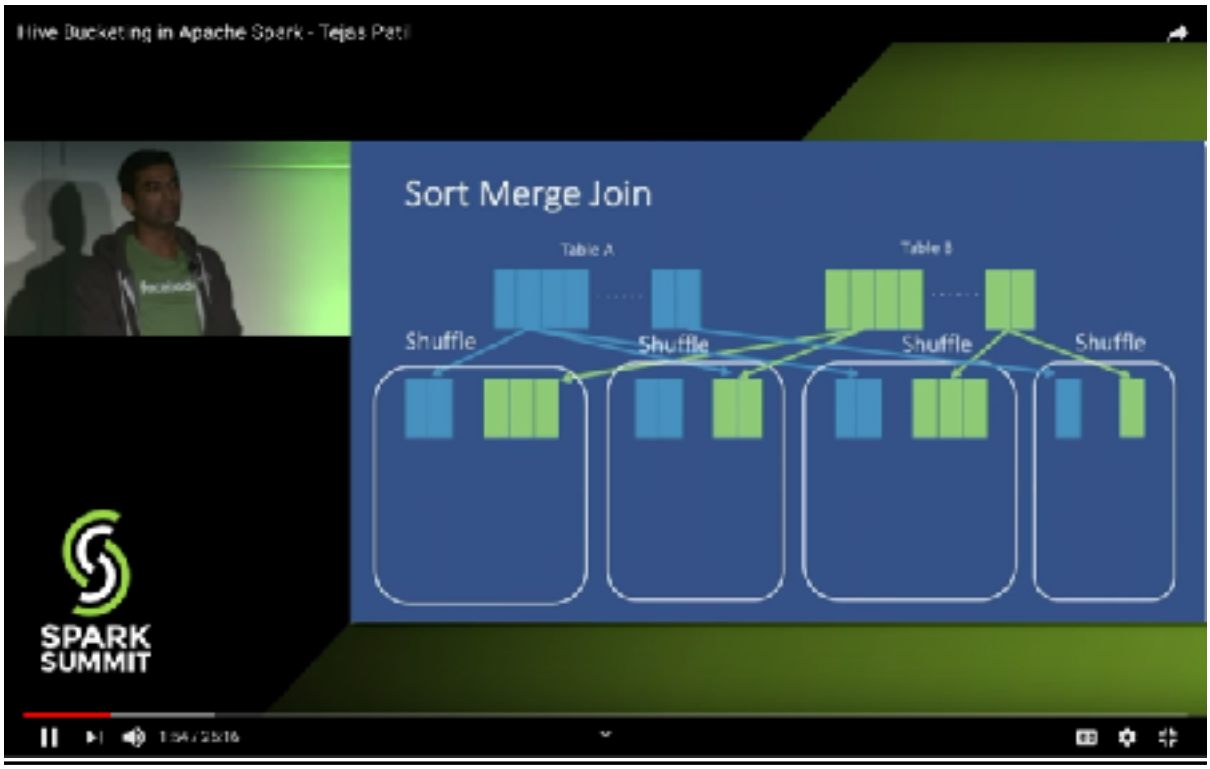
- Largest Shuffle Stage
  - Target Size <= 200 MB/partition
- Partition Count = Stage Input Data / Target Size
  - Solve for Partition Count

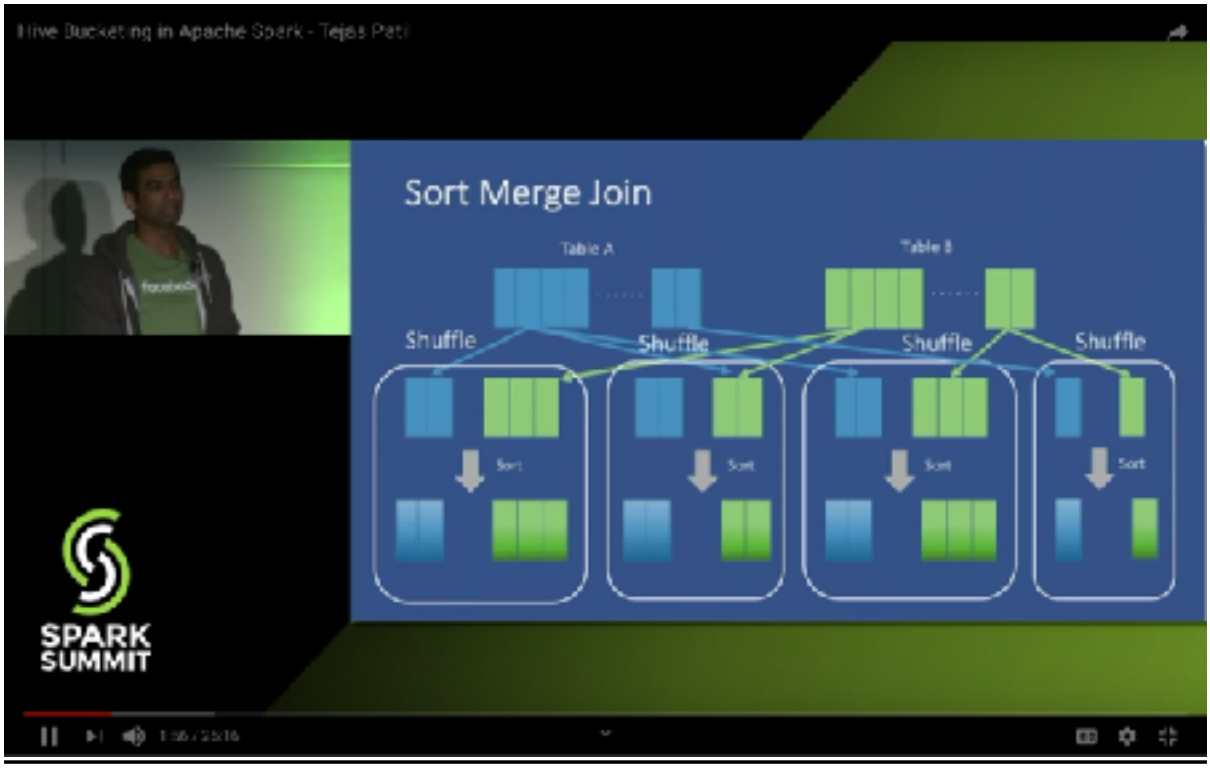
EXAMPLE
Shuffle Stage Input = 210GB
x = 210000MB / 200MB = 1050
spark.conf.set("spark.sql.shuffle.partitions", 1050)
BUT -> If cluster has 2000 cores
spark.conf.set("spark.sql.shuffle.partitions", 2000)

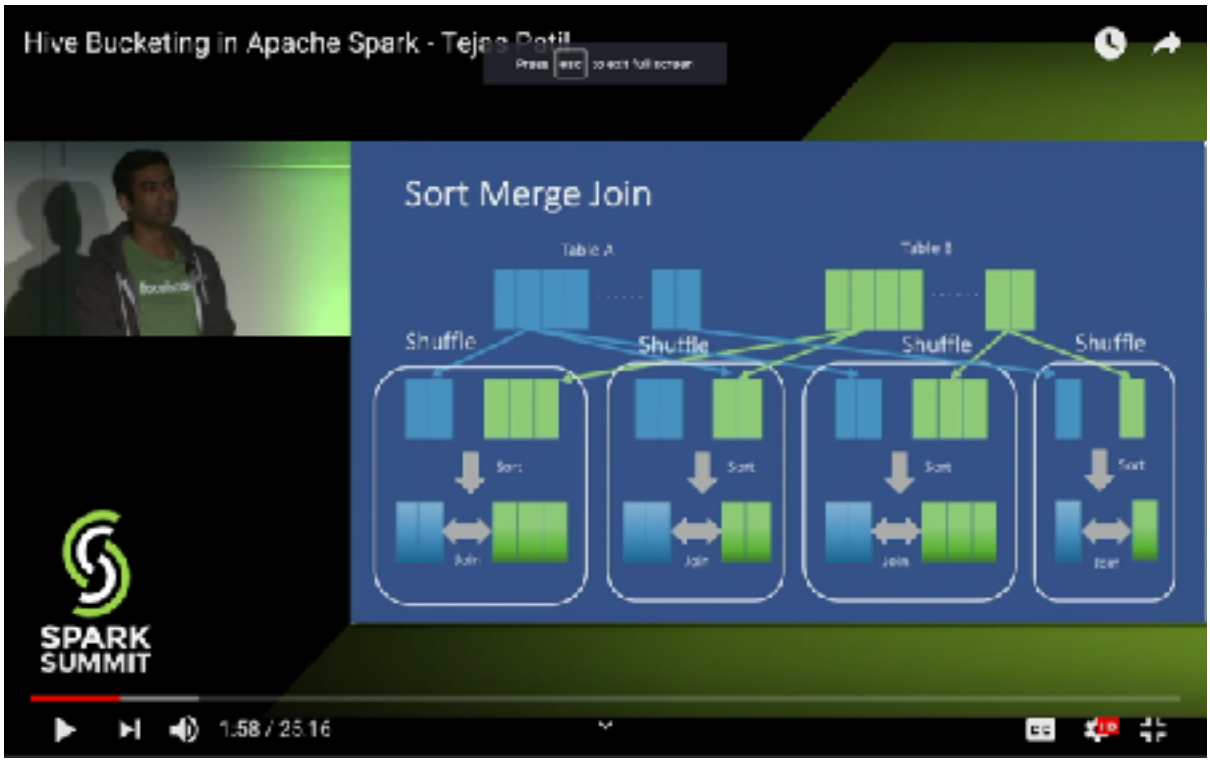
Join Optimization

- SortMergeJoins (Standard)
- Broadcast Joins (Fastest)
- Skew Joins
- Range Joins
- BroadcastedNestedLoop Joins (BNLJ)


Hive Bucketing in Apache Spark - Tejas Patil
Table A
JOIN
Table B
Broadcast hash join
- Ship smaller table to all nodes
- Stream the other table
Shuffle hash join
- Shuffle both tables,
- Hash smaller one, stream the bigger one
Sort merge join
- Shuffle both tables,
- Sort both tables, buffer one, stream the bigger one
SPARK SUMMIT


Hive Bucketing in Apache Spark - Tejas Patil
Sort Merge Join
Table A
Table B
Shuffle
Shuffle
Shuffle
Shuffle
SPARK SUMMIT





Spark Joins Examples:

```
scala>   case class Row(id: Int, value: String)
defined class Row

scala>   val r1 = Seq(Row(1, "A1"), Row(2, "A2"), Row(3, "A3"),
Row(4, "A4")).toDS()
r1: org.apache.spark.sql.Dataset[Row] = [id: int, value: string]

scala>   val r2 = Seq(Row(3, "A3"), Row(4, "A4"), Row(4, "A4_1"),
Row(5, "A5"), Row(6, "A6")).toDS()
r2: org.apache.spark.sql.Dataset[Row] = [id: int, value: string]
```

Inner join

```
scala> val innerJOin = r1.join(r2, usingColumns = Seq("id"),
joinType="inner").show
+---+-----+-----+
| id|value|value|
+---+-----+-----+
|  3|   A3|   A3|
|  4|   A4|   A4|
|  4|   A4| A4_1|
+---+-----+-----+

innerJOin: Unit = ()
```

Outer join/ Full join / Full outer join: (All are same)

```
scala> val outerJOin = r1.join(r2, usingColumns = Seq("id"),
joinType="outer").show
+---+-----+-----+
| id|value|value|
+---+-----+-----+
|  1|   A1| null|
|  6| null|   A6|
|  3|   A3|   A3|
|  5| null|   A5|
|  4|   A4|   A4|
|  4|   A4| A4_1|
|  2|   A2| null|
+---+-----+-----+

outerJOin: Unit = ()

scala> val fullJOin = r1.join(r2, usingColumns = Seq("id"),
joinType="full").show
+---+-----+-----+
| id|value|value|
+---+-----+-----+
|  1|   A1| null|
|  6| null|   A6|
|  3|   A3|   A3|
|  5| null|   A5|
|  4|   A4|   A4|
|  4|   A4| A4_1|
|  2|   A2| null|
+---+-----+-----+

fullJOin: Unit = ()

scala> val fullOuterJOin = r1.join(r2, usingColumns = Seq("id"),
joinType="full_outer").show
+---+-----+-----+
| id|value|value|
+---+-----+-----+
|  1|   A1| null|
|  6| null|   A6|
|  3|   A3|   A3|
|  5| null|   A5|
|  4|   A4|   A4|
|  4|   A4| A4_1|
|  2|   A2| null|
+---+-----+-----+
```

Left join/ Left Outer join-(both same)

```
scala> val leftJOin = r1.join(r2, usingColumns = Seq("id"),
joinType="left").show
+---+-----+-----+
| id|value|value|
+---+-----+-----+
|  1|   A1| null|
|  2|   A2| null|
|  3|   A3|   A3|
|  4|   A4| A4_1|
|  4|   A4|   A4|
+---+-----+-----+

leftJOin: Unit = ()

scala> val leftOuterJOin = r1.join(r2, usingColumns = Seq("id"),
joinType="left_Outer").show
+---+-----+-----+
| id|value|value|
+---+-----+-----+
|  1|   A1| null|
|  2|   A2| null|
|  3|   A3|   A3|
|  4|   A4| A4_1|
|  4|   A4|   A4|
+---+-----+-----+
```

Left Semi join

```
scala> val left_semiJOin = r1.join(r2, usingColumns = Seq("id"),
joinType="left_semi").show
+---+-----+
| id|value|
+---+-----+
|  3|   A3|
|  4|   A4|
+---+-----+
```

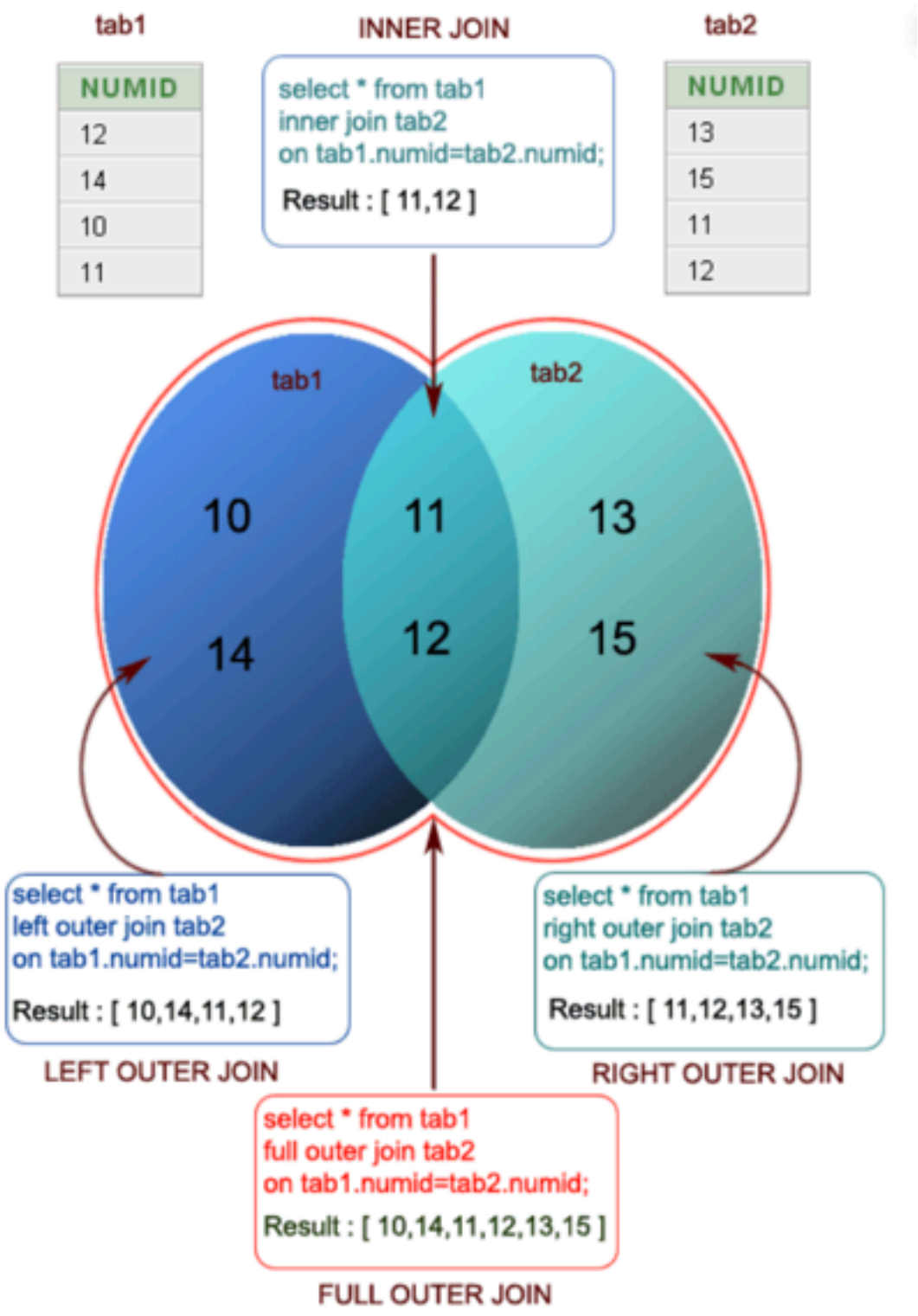
Left Anti join

```
scala> val left_antiJOin = r1.join(r2, usingColumns = Seq("id"),
joinType="left_anti").show
+---+-----+
| id|value|
+---+-----+
|  1|   A1|
|  2|   A2|
+---+-----+

left_antiJOin: Unit = ()
```

Right outer join/ right join : both are same

```
scala> val rightJOin = r1.join(r2, usingColumns = Seq("id"),
joinType="right").show
+---+-----+-----+
| id|value|value|
+---+-----+-----+
|  3|   A3|   A3|
|  4|   A4|   A4|
|  4|   A4| A4_1|
|  5| null|   A5|
|  6| null|   A6|
+---+-----+-----+

rightJOin: Unit = ()

scala> val right_Outer_JOin = r1.join(r2, usingColumns =
Seq("id"), joinType="right_outer").show
+---+-----+-----+
| id|value|value|
+---+-----+-----+
|  3|   A3|   A3|
|  4|   A4|   A4|
|  4|   A4| A4_1|
|  5| null|   A5|
|  6| null|   A6|
+---+-----+-----+
```

Reference:
https://stackoverflow.com/questions/45990633/what-are-the-various-join-types-in-spark



List of spark supported join between two dataframes

1. inner : 'inner'
2. cross: 'cross'
3. outer: 'outer'
4. full: 'full'
5. full outer: 'fullouter'
6. left : 'left'
7. left outer : 'leftouter'
8. right : 'right'
9. right outer : 'rightouter'
10. left semi: 'leftsemi'
11. left anti: 'leftanti'

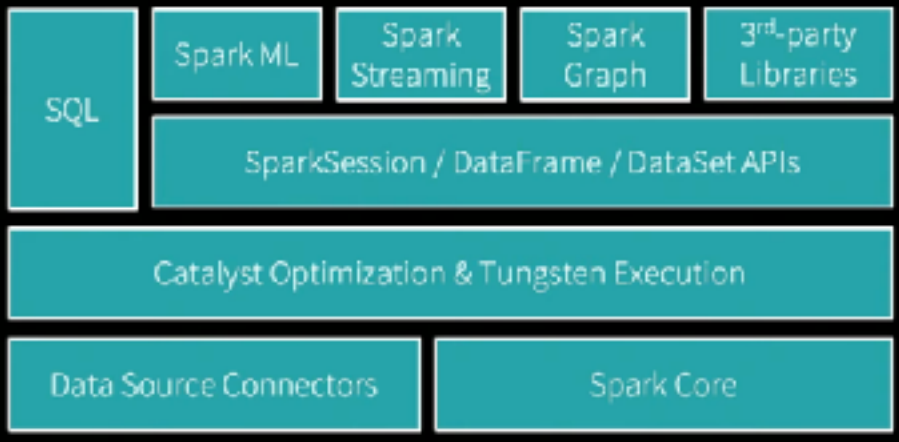
Omit Expensive Ops

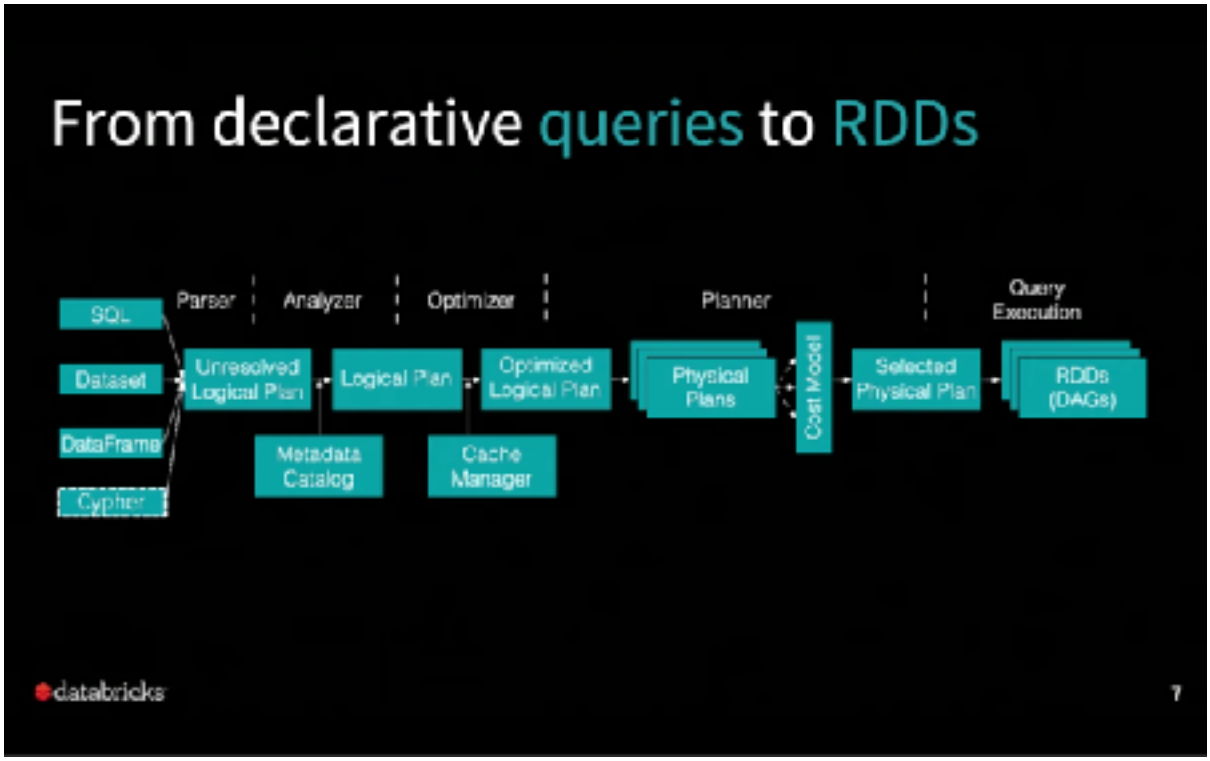
- Repartition
  - Use Coalesce or Shuffle Partition Count
- Count – Do you really need it?
- DistinctCount
  - use approxCountDistinct()
- If distincts are required, put them in the right place
  - Use dropDuplicates
  - dropDuplicates BEFORE the join
  - dropDuplicates BEFORE the groupBy

Summary

- Utilize Lazy Loading (Data Skipping)
- Maximize Your Hardware
- Right Size Spark Partitions
- Balance
- Optimized Joins
- Minimize Data Movement
- Minimize Repetition
- Only Use Vectorized UDFs

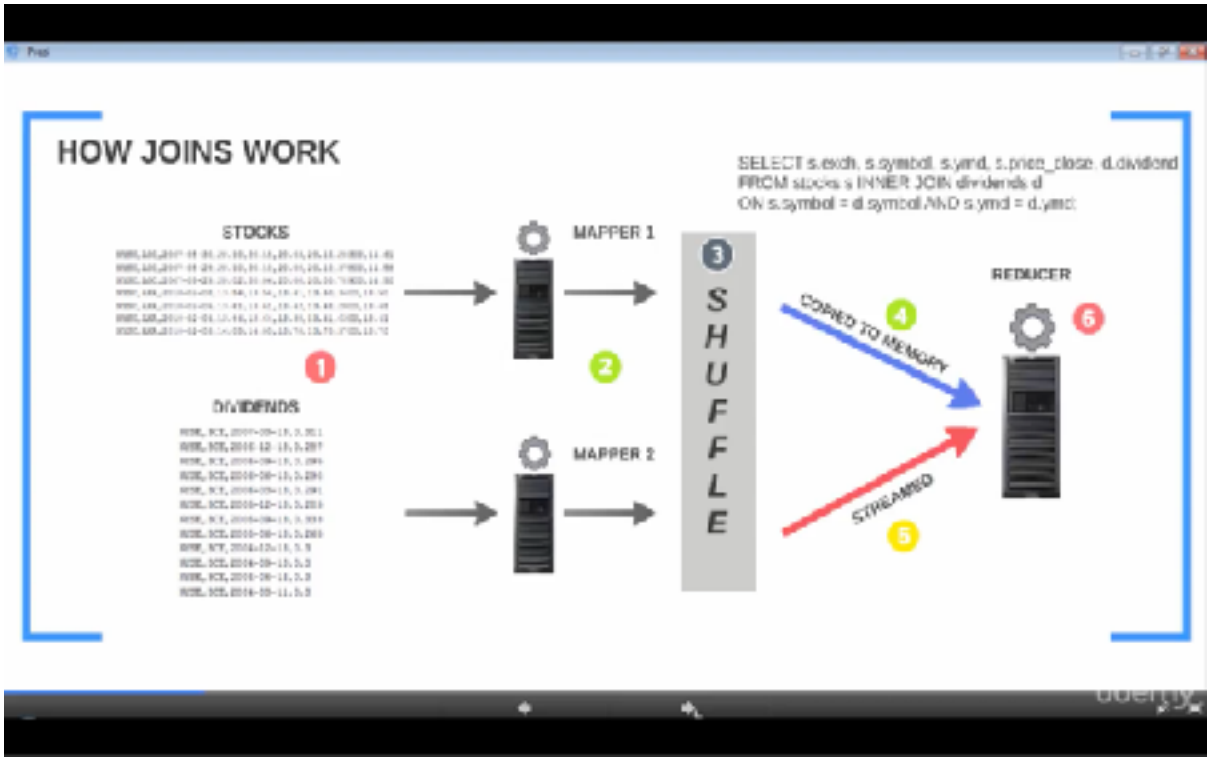
Apache Spark 3.x

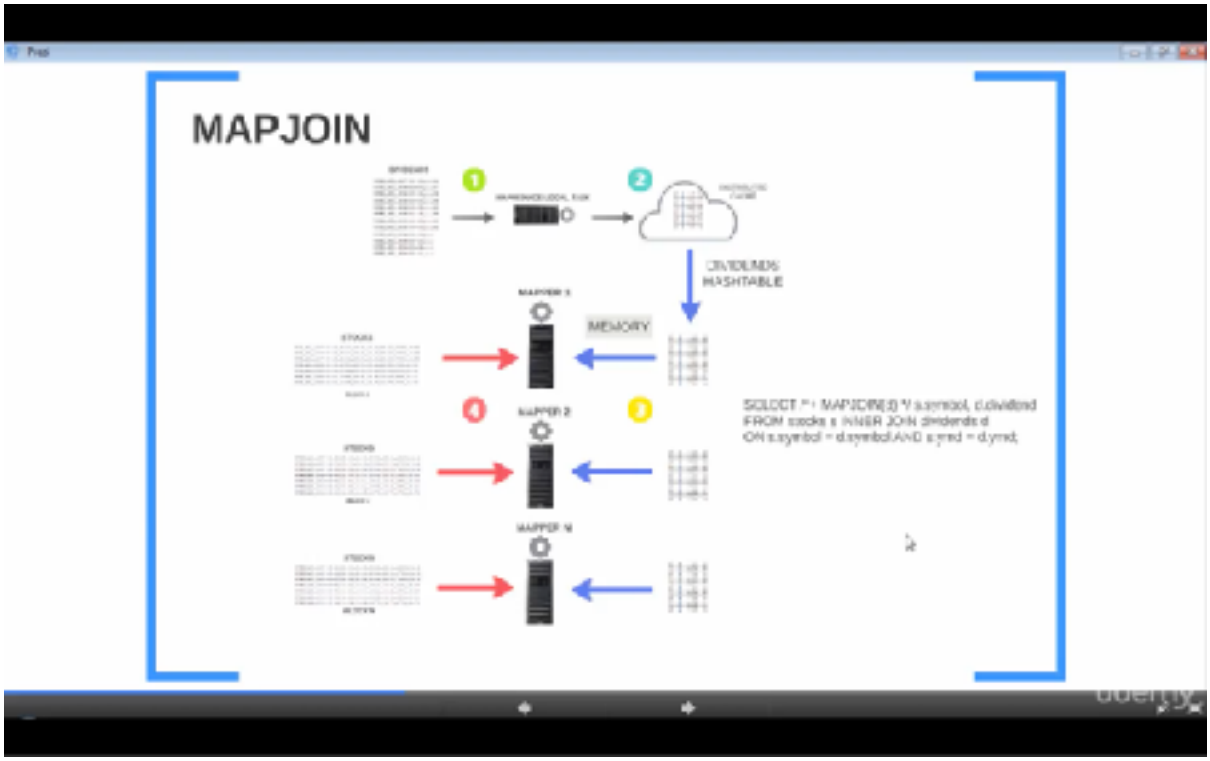


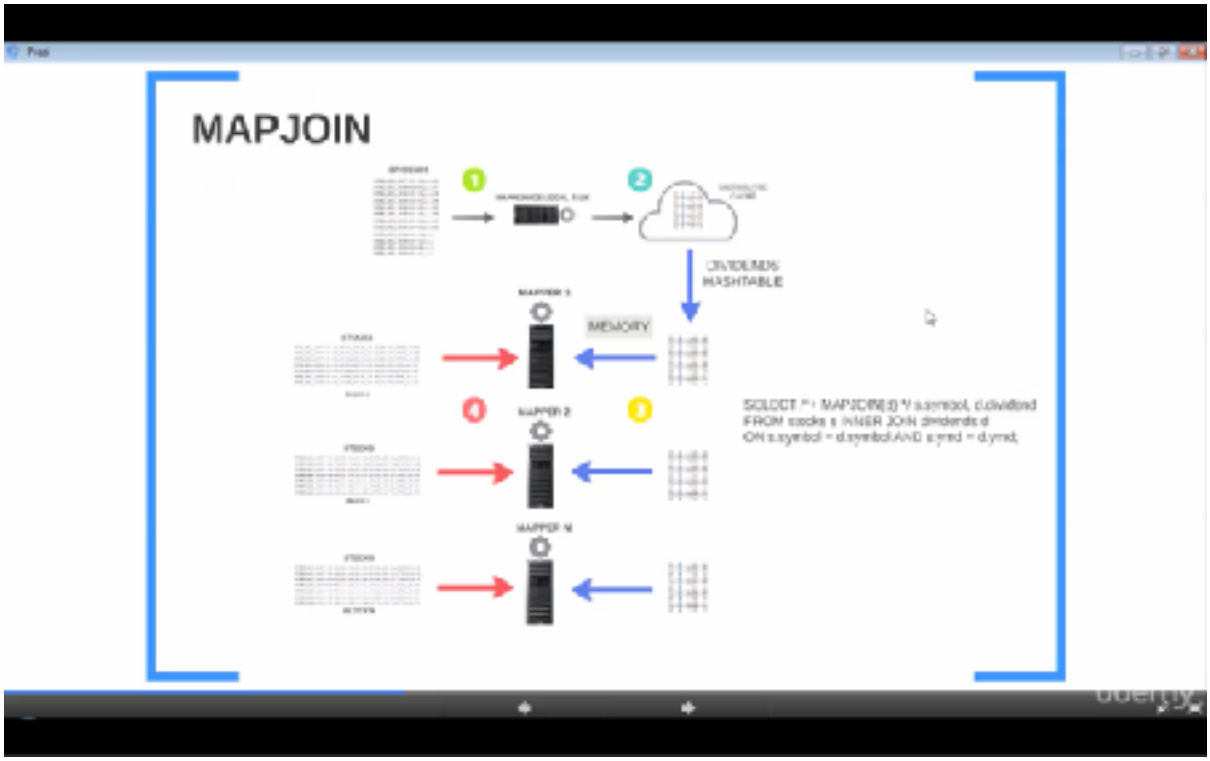


https://www.youtube.com/watch?v=HJvuU0CQS44

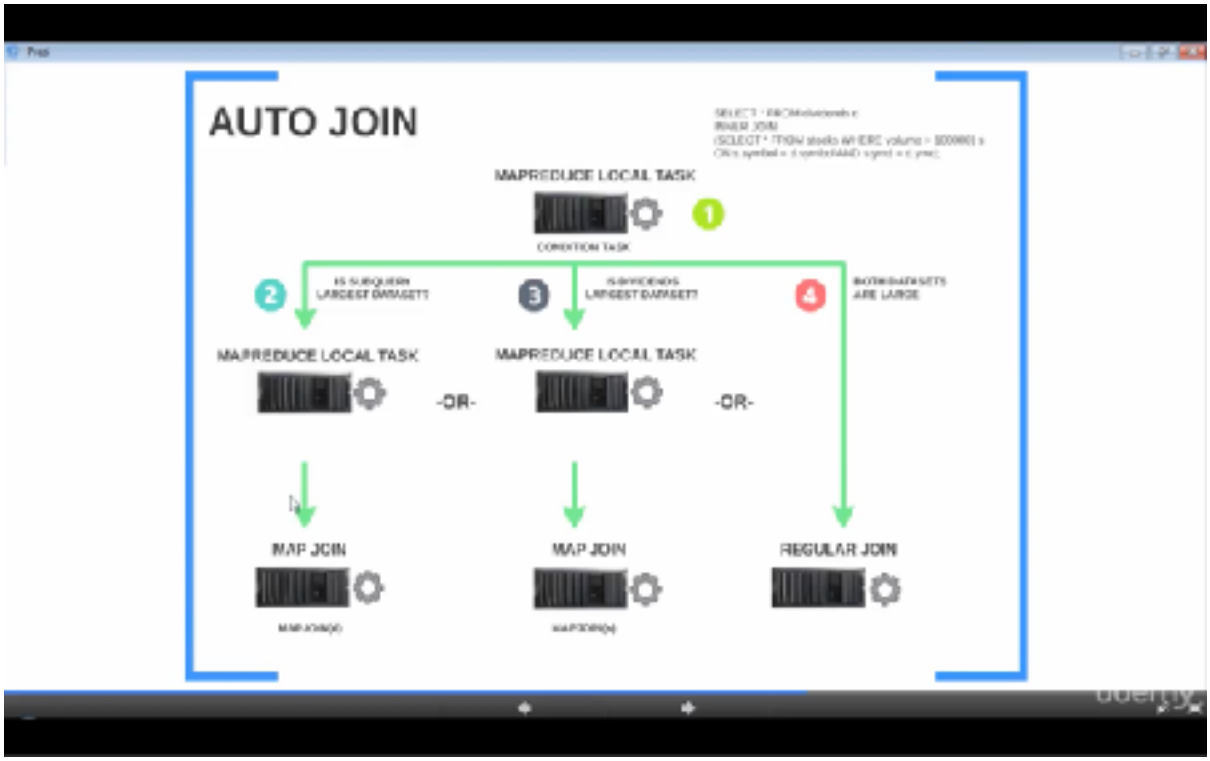
Hive Joins

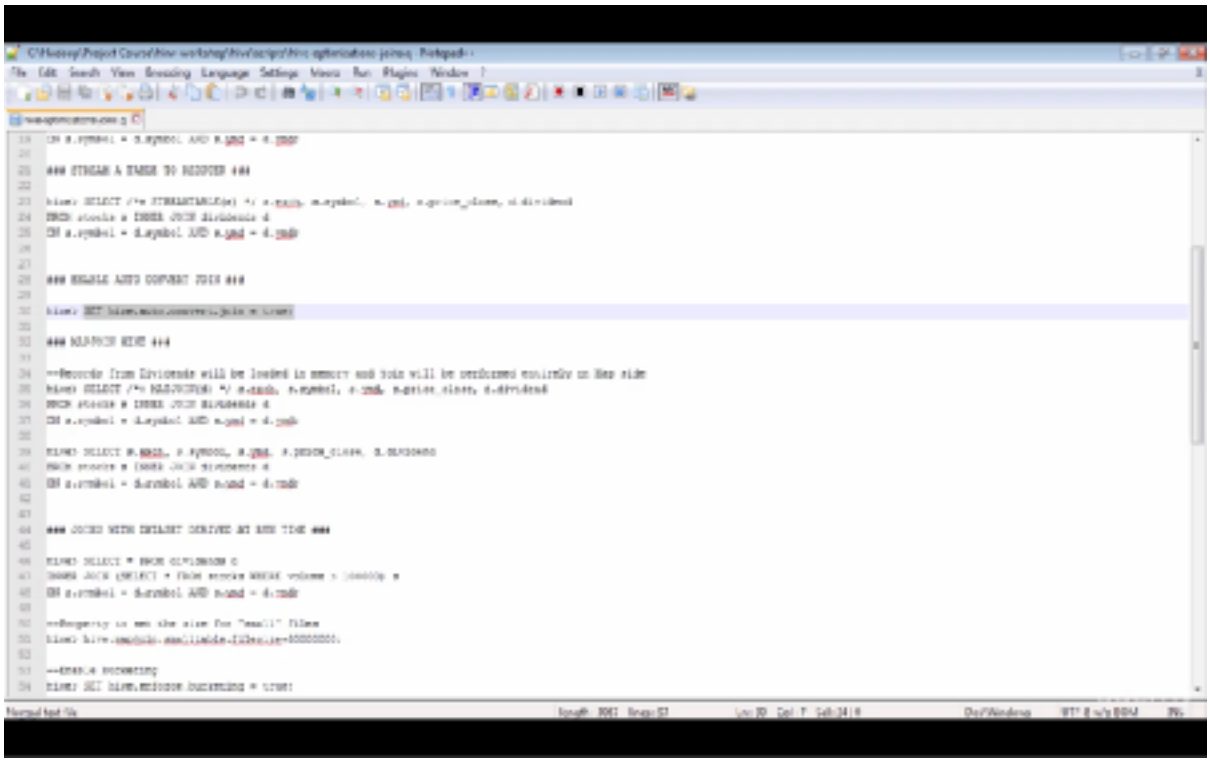

HOW JOINS WORK
SELECT s.exch, s.symbol, s.ymd, s.price_close, d.dividend
FROM stocks s INNER JOIN dividends d
ON s.symbol = d.symbol AND s.ymd = d.ymd;
STOCKS
DIVIDENDS
MAPPER 1
MAPPER 2
SHUFFLE
COPIED TO MEMORY
STREAMED
REDUCER
1
2
3
4
5
6

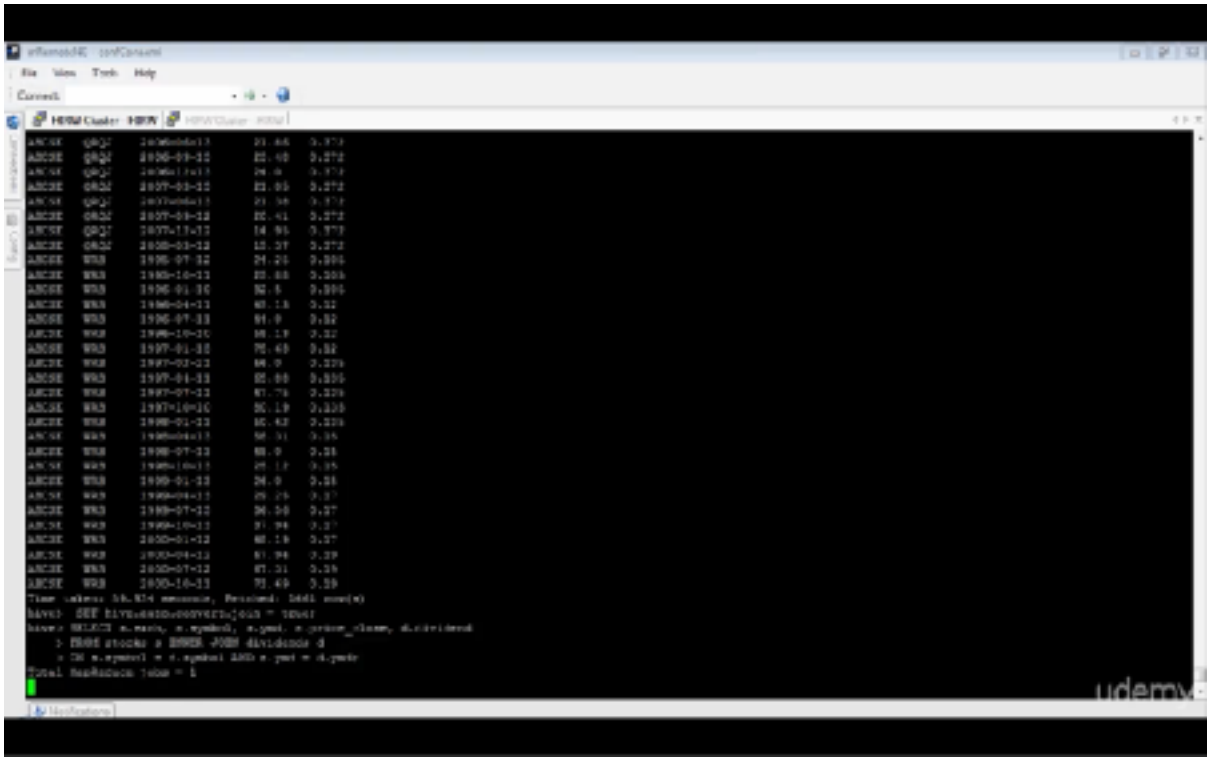

MAPJOIN
DIVIDENDS HASHTABLE
MEMORY
MAPPER 1
MAPPER 2
MAPPER N
SELECT /*+ MAPJOIN(d) */ s.symbol, d.dividend
FROM stocks s INNER JOIN dividends d
ON s.symbol = d.symbol AND s.ymd = d.ymd;

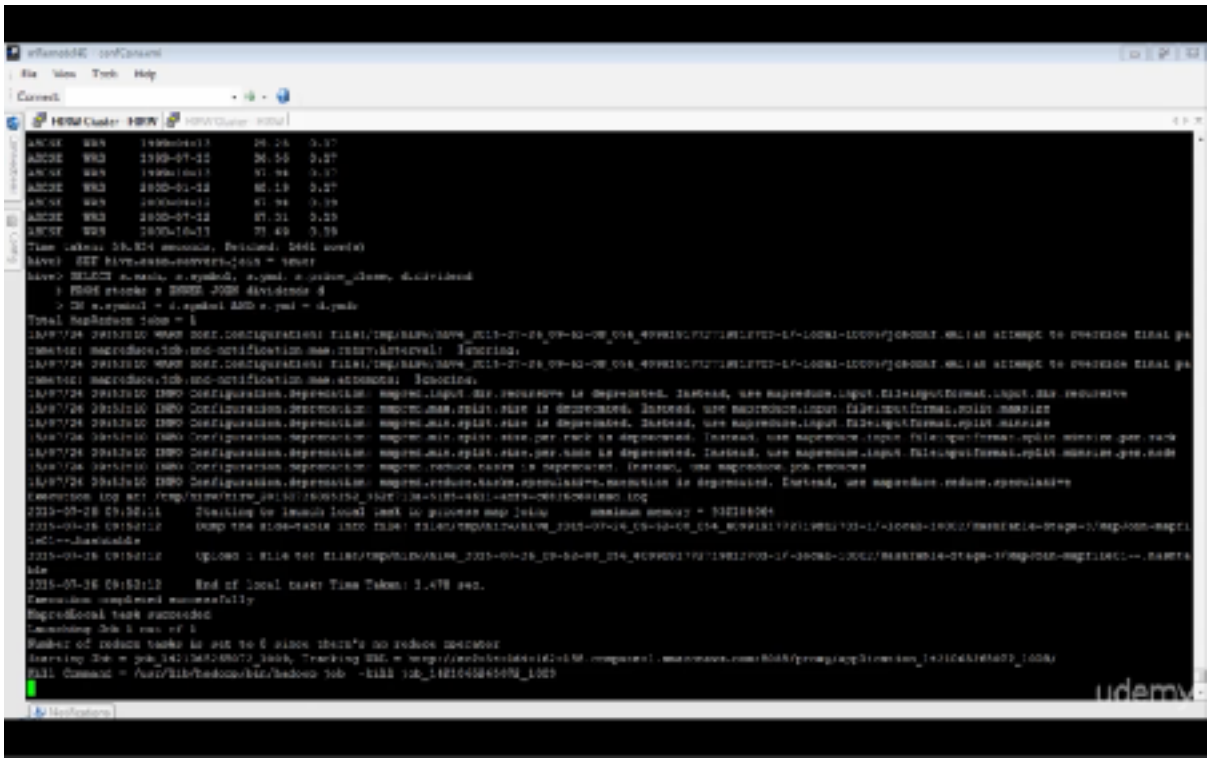

MAPJOIN
1
2
DIVIDENDS HASHTABLE
MEMORY
4
3

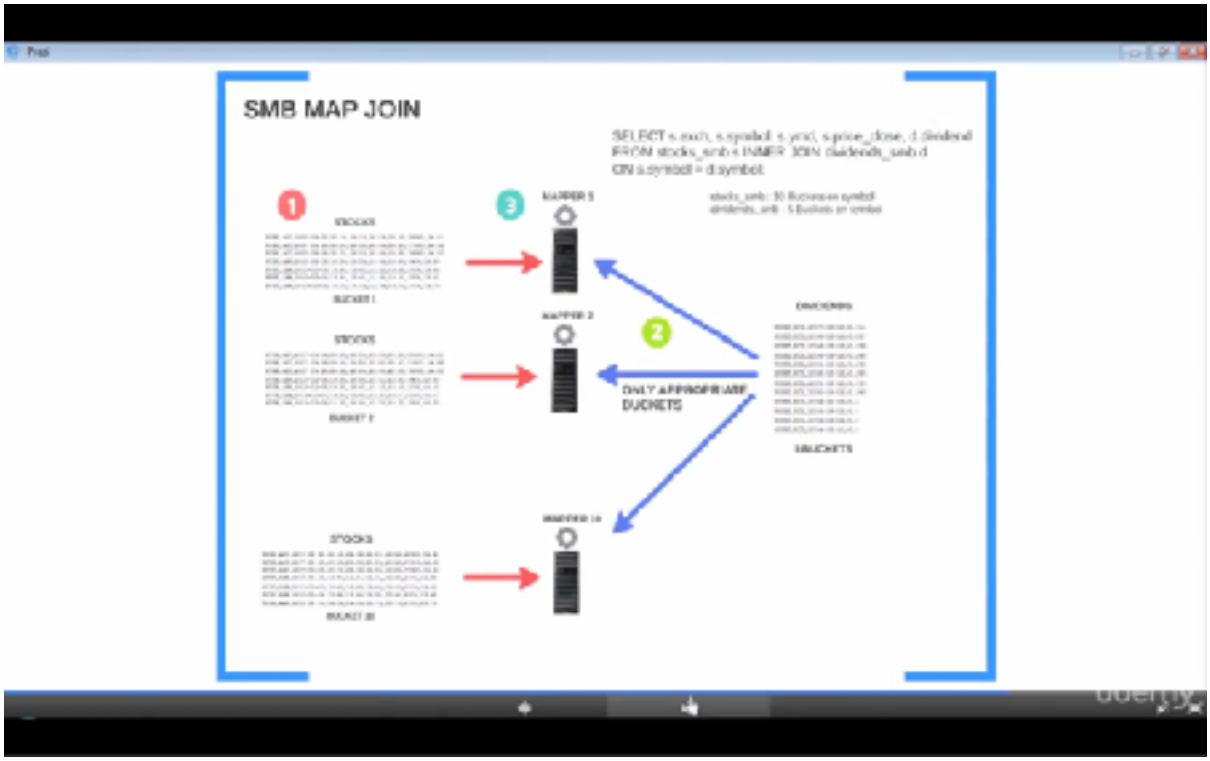

JOIN OPTIMIZATIONS IN HIVE
• How Join Works
• STREAMTABLE
• Map Join
• Auto Map Join
• Sort Merge Bucket (SMB) Join

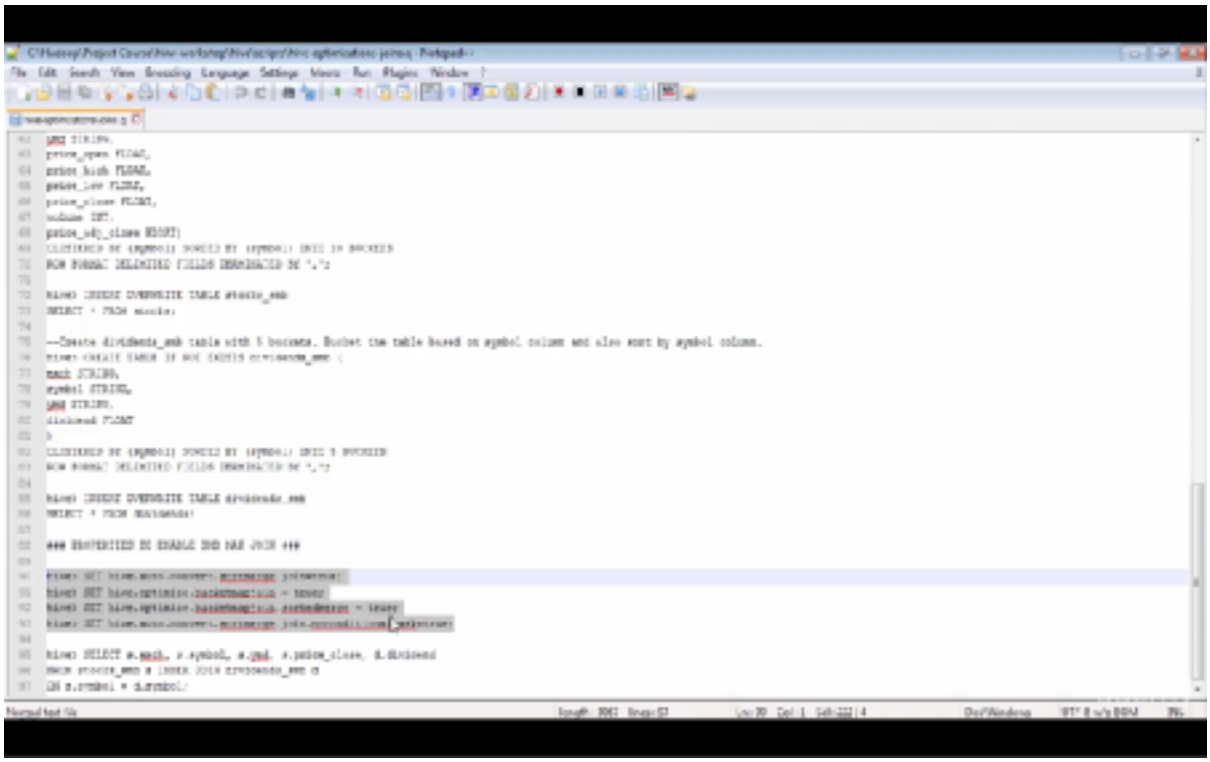

AUTO JOIN
MAPREDUCE LOCAL TASK
CONDITION TASK
IS SUBQUERY LARGEST DATASET?
IS ANYONE ONE LARGEST DATASET?
BOTH DATASETS ARE LARGE
MAPREDUCE LOCAL TASK
-OR-
MAPREDUCE LOCAL TASK
-OR-
MAP JOIN
MAP JOIN
REGULAR JOIN

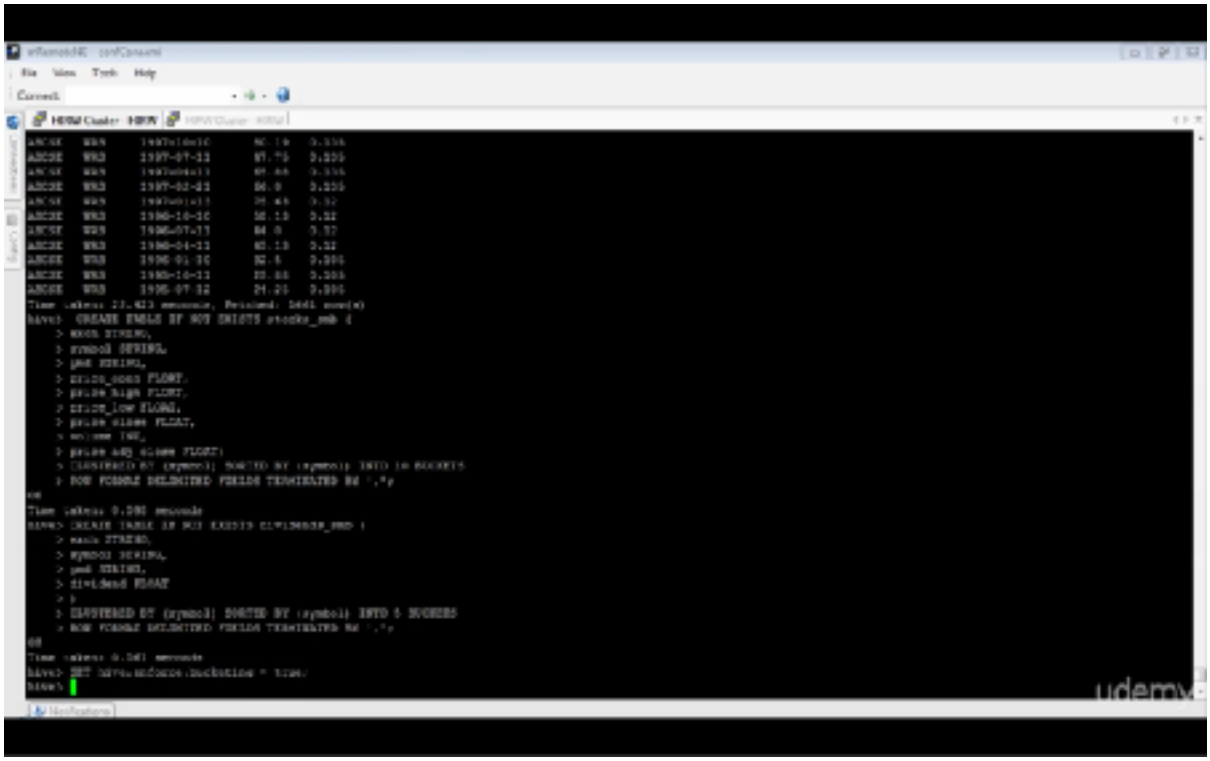


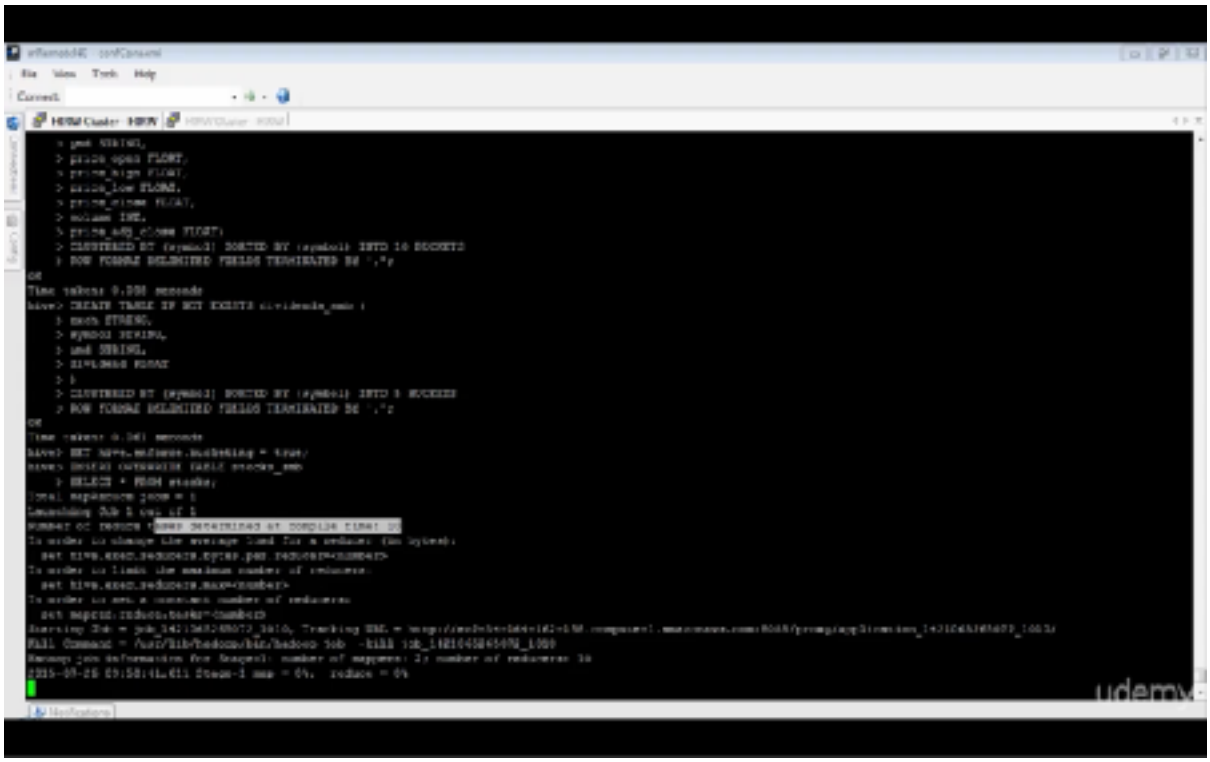


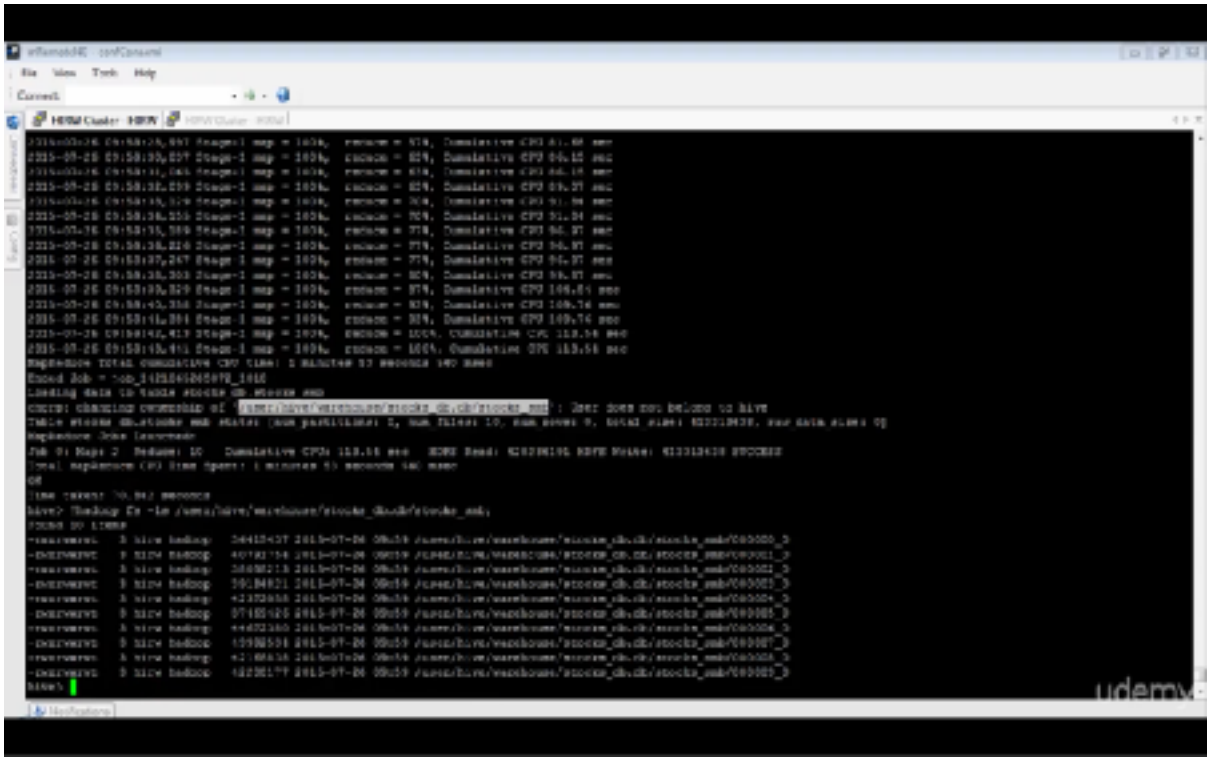


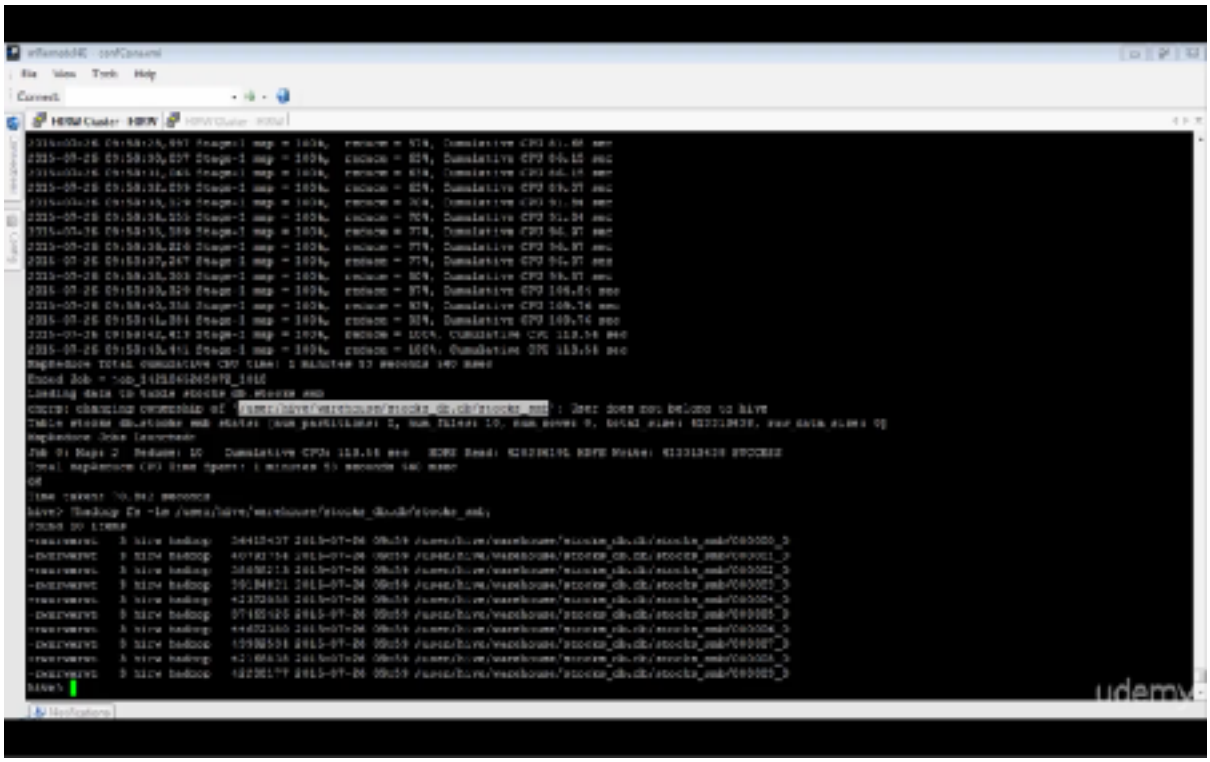

SMB MAP JOIN
SELECT s.ymd, s.symbol, s.year, s.price_close, d.dividend
FROM stocks_smb s INNER JOIN dividends_smb d
ON s.symbol = d.symbol;
1
3
2
DAILY APPROPRIATE BUCKETS
STOCKS
STOCKS
STOCKS

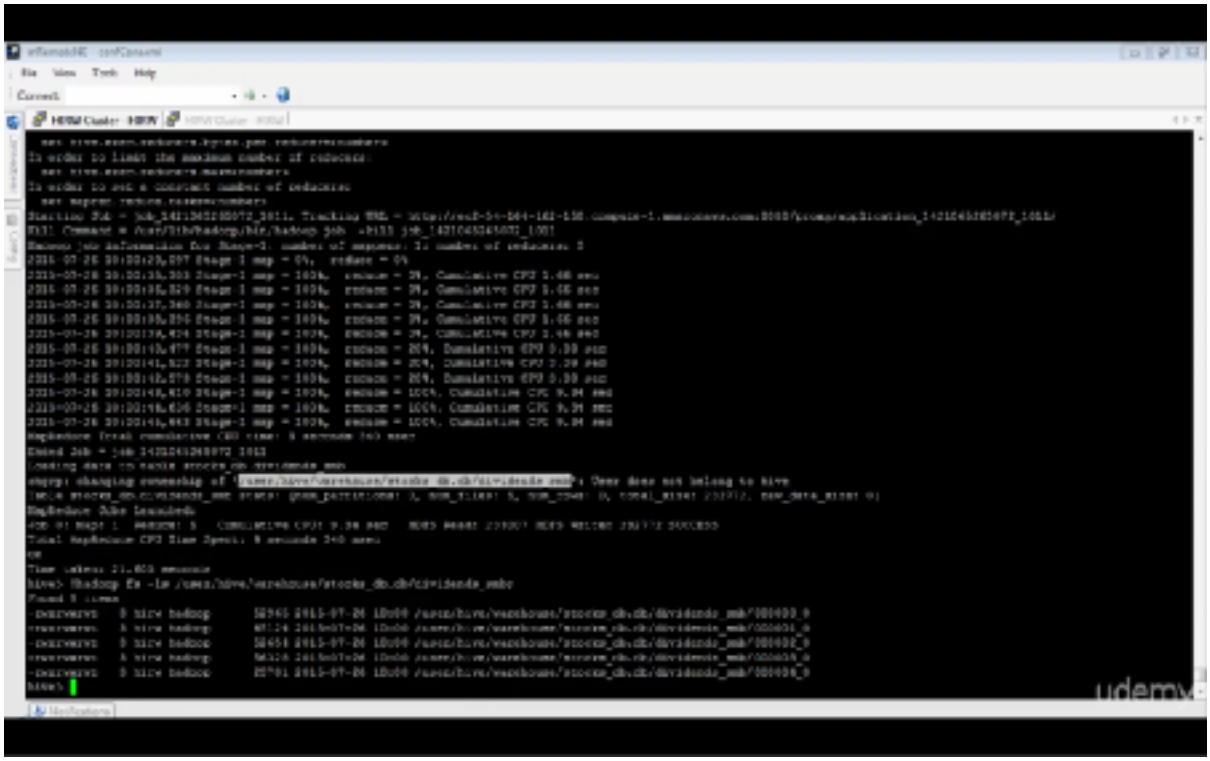

SET hive.auto.convert.sortmerge.join=true;
SET hive.optimize.bucketmapjoin = true;
SET hive.optimize.bucketmapjoin.sortedmerge = true
SET hive.auto.convert.sortmerge.join.noconditional.task=true;
SELECT s.ymd, s.symbol, s.yea, s.price_close, d.dividend
FROM stocks_smb s JOIN dividends_smb d
ON s.symbol = d.symbol;

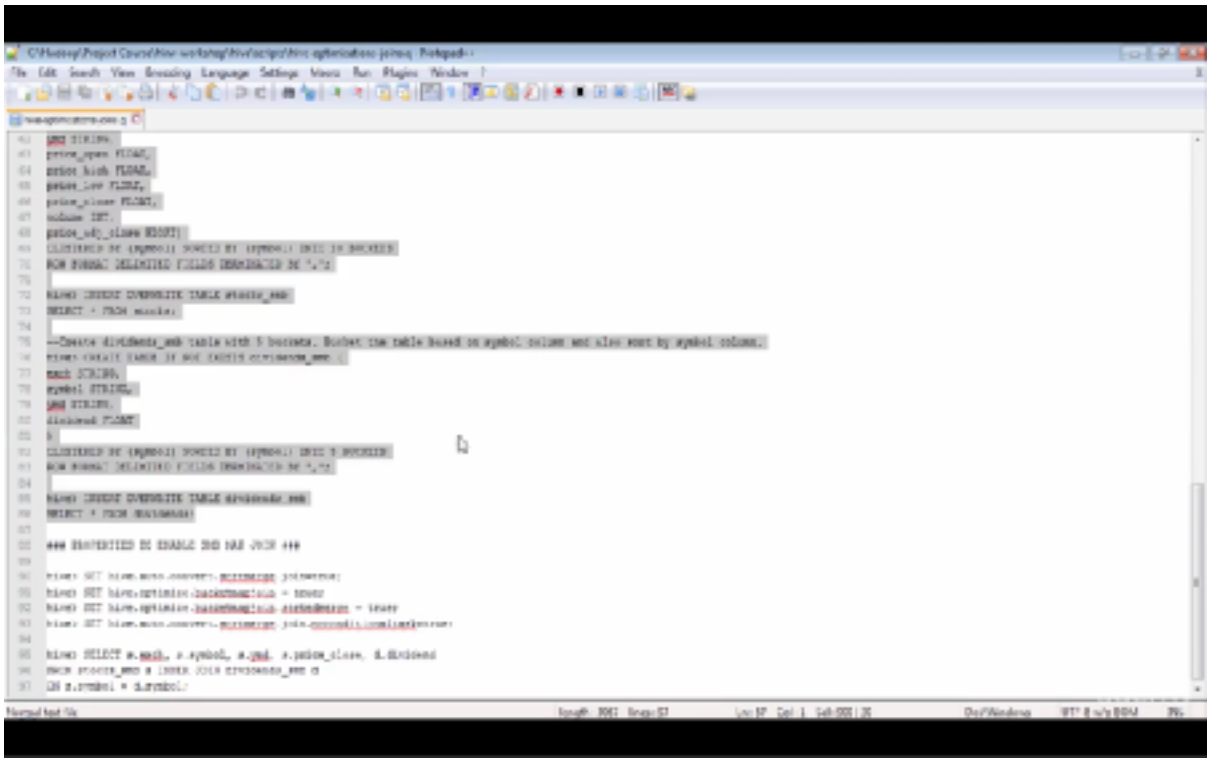


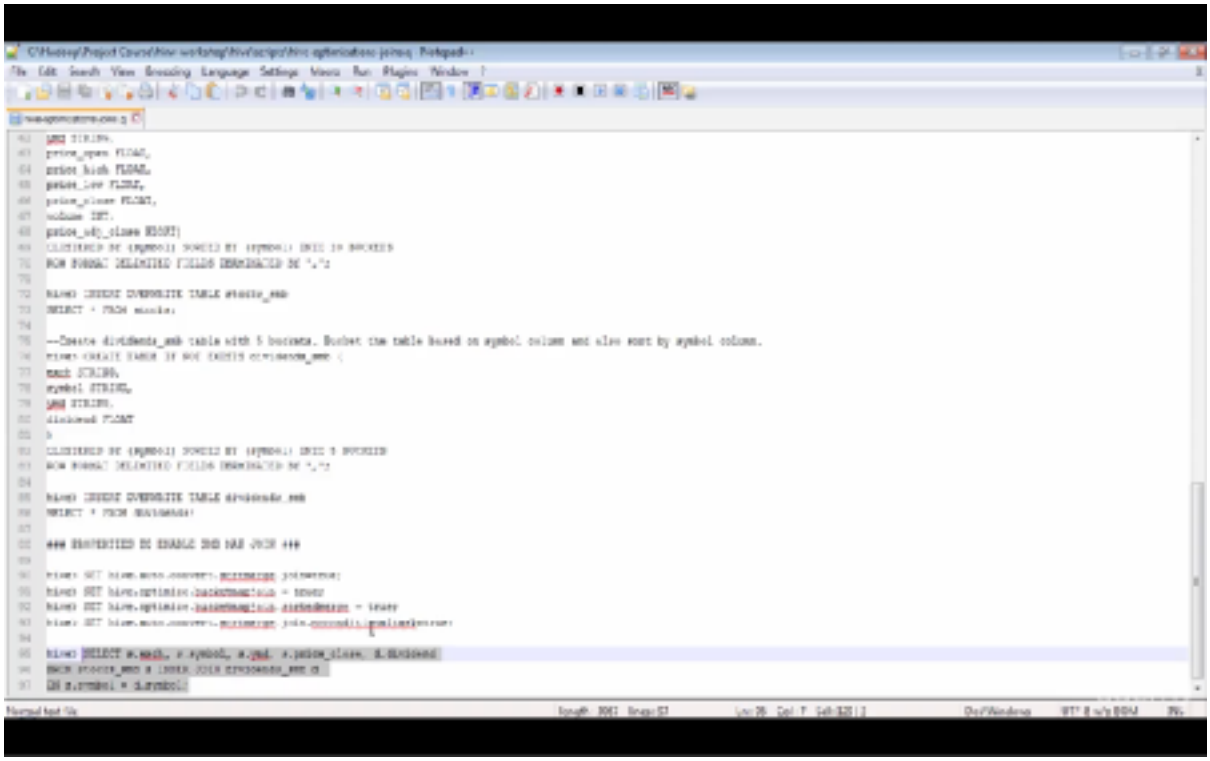


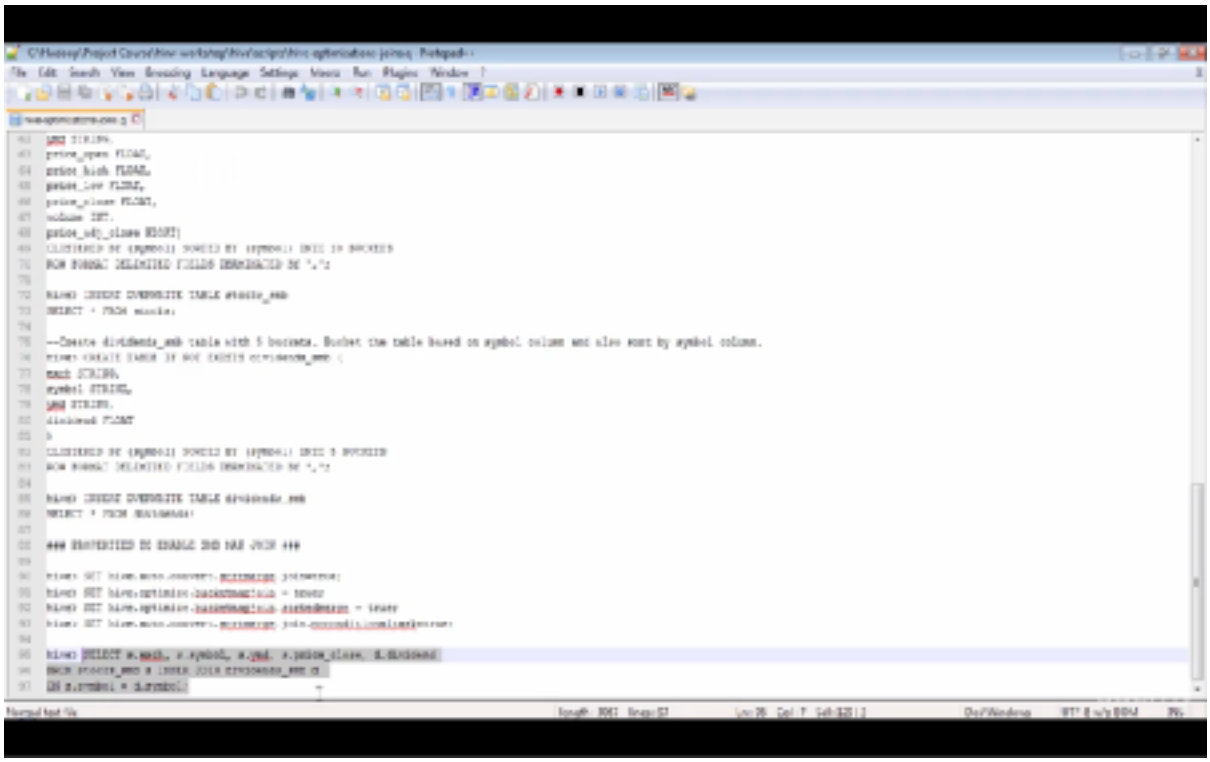


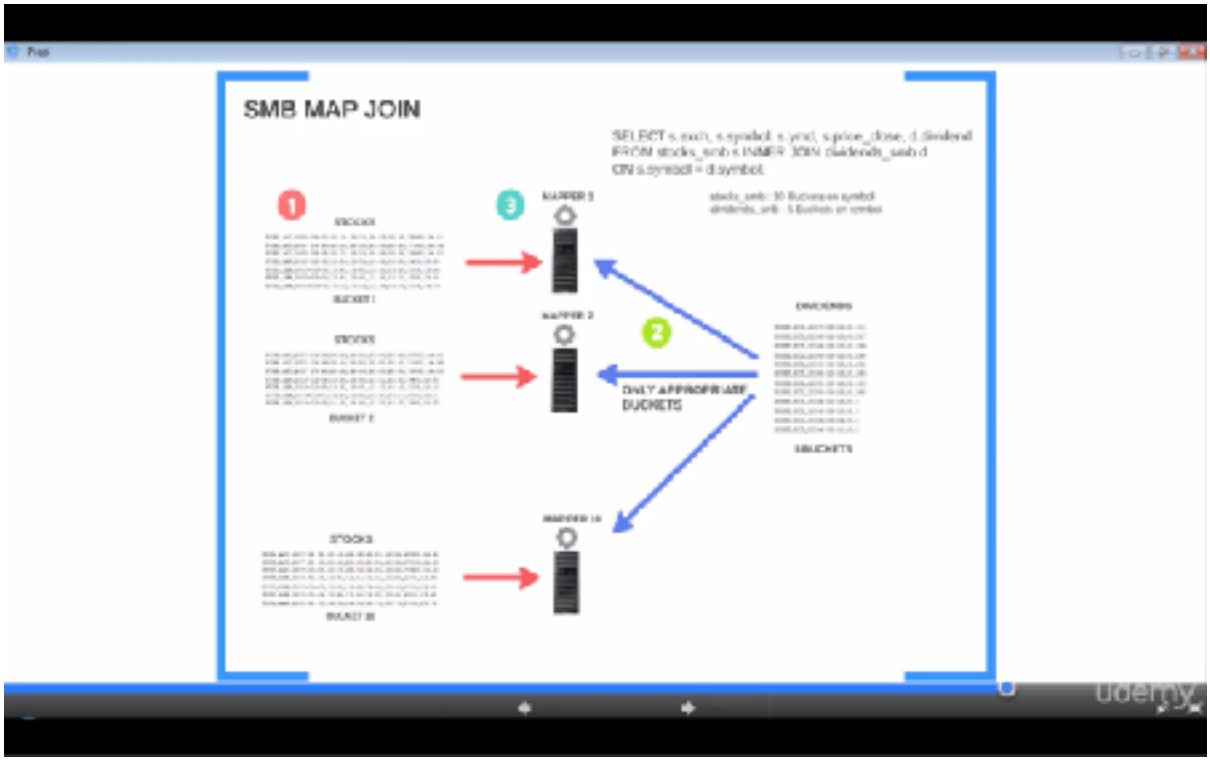


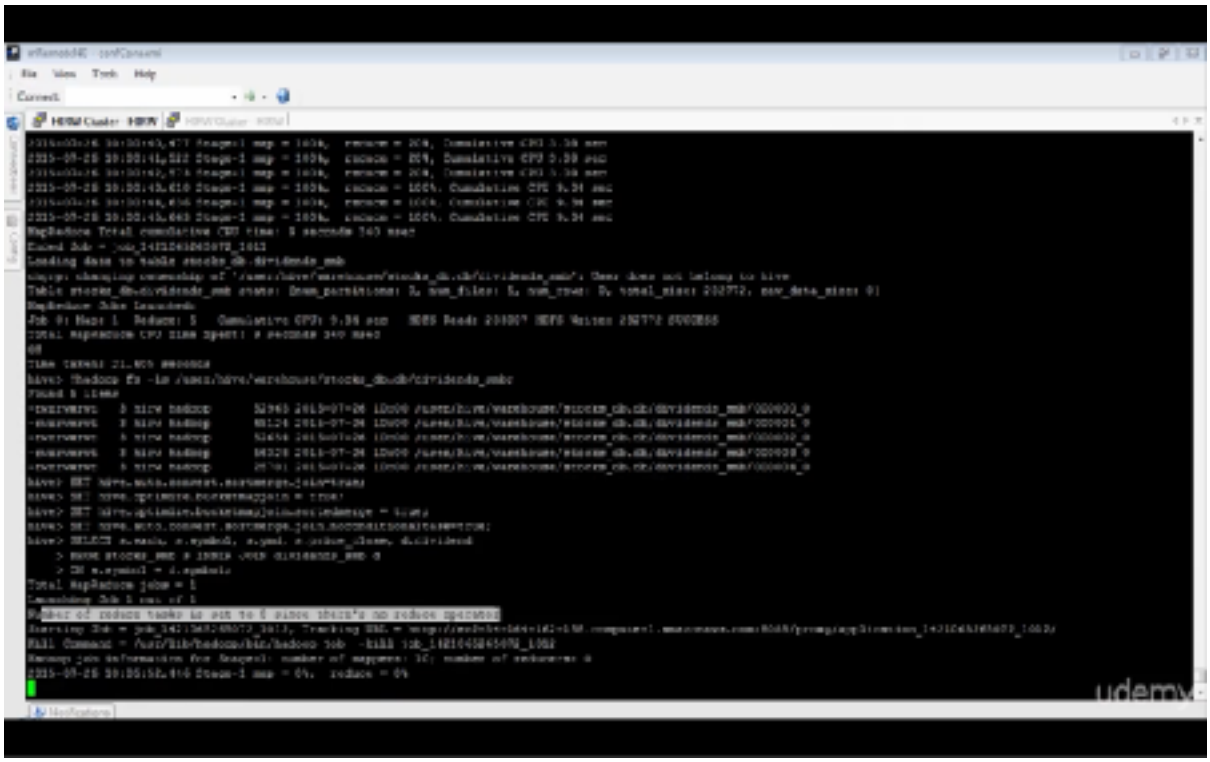









SMB MAP JOIN
SELECT s.ymd, s.symbol, s.price_close, d.dividend
FROM stocks_smb s INNER JOIN dividends_smb d
ON s.symbol = d.symbol;
1
2
3
DAILY APPROPRIATE BUCKETS
udemy


udemy

Spark Best Performances:

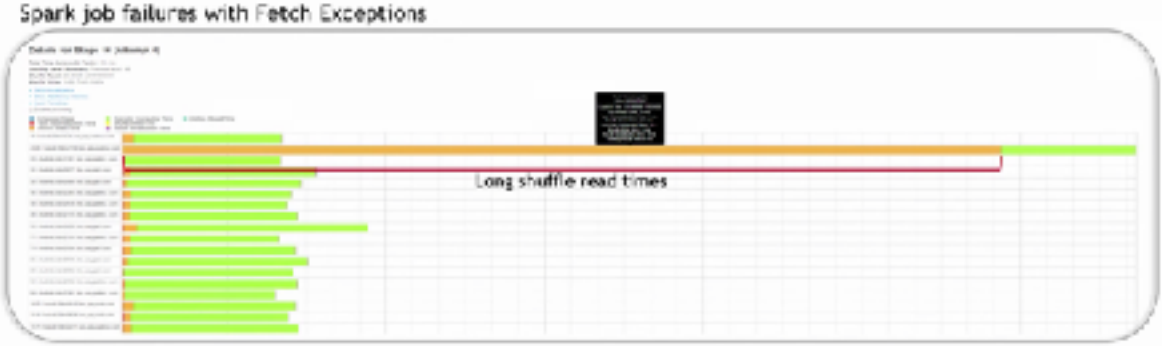
Spark Best Practices Checklist

Data Serialization	✓ Use Kyro Serializer with SparkConf, which is faster and compact ✓ Tune kyroserializer buffer to hold large objects
Garbage Collection	✓ Clean up cached/persisted collections when they are no longer needed ✓ Tuned concurrent mbortable preclean time from 10sec to 30sec to push out stop the world GC
Memory Management	✓ Avoided using executors with too much memory
Parallelism	✓ Optimize number of cores to partitions*
Action-Transformation	✓ Minimize shuffles on join() by broadcasting the smaller collection ✓ Optimize wider transformations as much as possible*
Caching & Persisting	✓ Used MEMORY_AND_DISK storage level for caching large ✓ Repartition data before persisting to HDFS for better performance in downstream jobs

*Specific examples later

Learnings

Spark job failures with Fetch Exceptions



Observations

- Executor spends long time on shuffle reads. Then times out , terminates and results in job failure
- Resource constraints on executor nodes causing delay in executor node

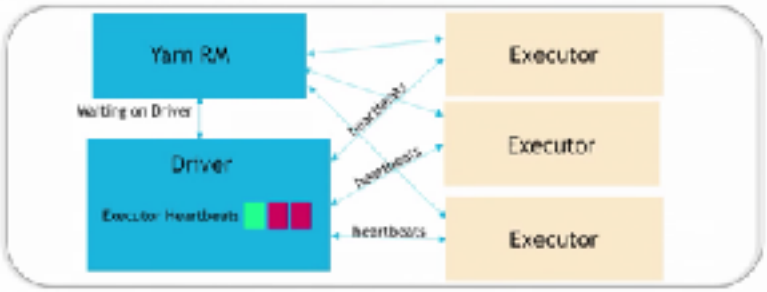
Resolution

To address memory constraints, tuned

1. config from 200 executor * 4 cores to 400 executor * 2 cores
2. executor memory allocation (reduced)

Learnings

Tuning between Spark driver and executors



Observations

- Spark Driver was left with too many heartbeat requests to process even after the job was complete
- Yarn kills the Spark job after waiting on the Driver to complete processing the Heartbeats

Resolution

- The setting "spark.executor.heartbeatInterval" was set too low. Increasing it to 50s fixed the issue
- Allocate more memory to Driver to handle overheads other than typical Driver processes

Learnings

Optimize joins for efficient use of cluster resources (Memory, CPU etc..,)

Learnings

Optimize joins for efficient use of cluster resources (Memory, CPU etc..)

Observation

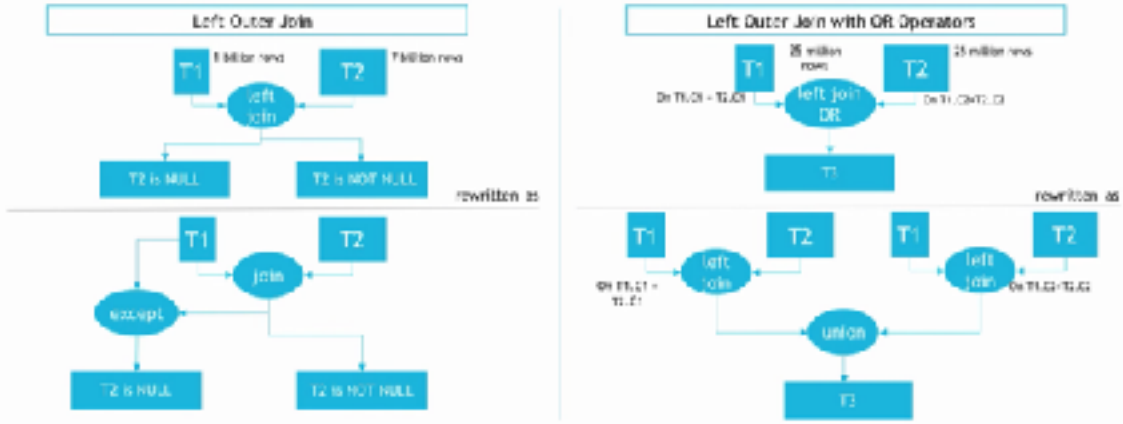
With the default shuffle partitions of 200, the Join Stage was running with too many tasks causing performance overhead

Resolution

Reduce the spark.sql.shuffle.partitions settings to a lower threshold

Learnings

Optimize wide transformations

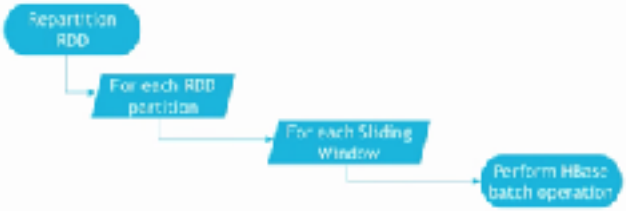


Learnings

Optimize throughput for HBase Spark Connection

Observations

- Batch puts and gets slow due to HBase overloaded connections
- Since our HBase row was wide, HBase operations for partitions containing larger groups were slow

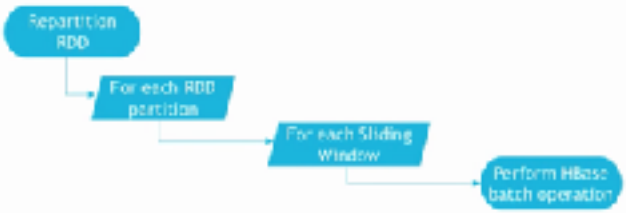


Learnings

Optimize throughput for HBase Spark Connection

Observations

- Batch puts and gets slow due to HBase overloaded connections
- Since our HBase row was wide, HBase operations for partitions containing larger groups were slow



Example

```
ValidPartitionedRDD  RDD[Event] = filteredRDD.repartition(2000)
...
...
groupedEventRDD.mapPartitions( p =>
p.sliding(2000,2000).foreach(
Create HBase Connection
Batch HBase Read or Write
Close HBase Connection
)
)
```

PayPal

Learnings

Optimize throughput for HBase Spark Connection

Observations

- Batch puts and gets slow due to HBase overloaded connections
- Since our HBase row was wide, HBase operations for partitions containing larger groups were slow

Resolution

- Implemented sliding window for HBase Operations to reduce HBase connection overload

Learnings

Optimize throughput for HBase Spark Connection

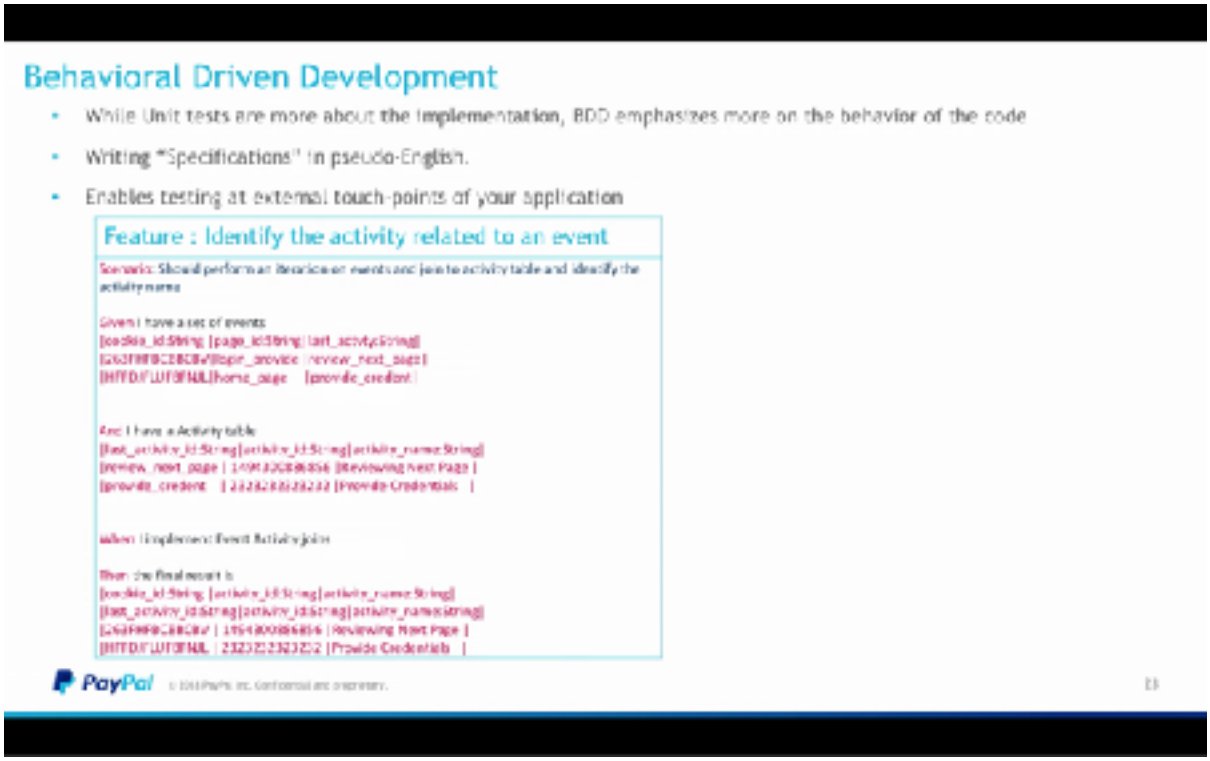
Observations

- Batch puts and gets slow due to HBase overloaded connections
- Since our HBase row was wide, HBase operations for partitions containing larger groups were slow

Repartition RDD
For each RDD partition
For each Sliding Window
Perform HBase batch operation
Example
groupedEventRDD.mapPartitions(p =>
p.sliding(200,200).foreach(
create hbase Connection
Batch hbase Read or Write
Close HBase Connection
)
)

Resolution

- Implemented sliding window for HBase Operations to reduce HBase connection overload



Catalyst Optimizer

At the core of Spark SQL is the Catalyst optimizer, which leverages advanced programming language features (e.g. **Scala's pattern matching and quasi quotes**) in a novel way to build an extensible query optimizer.

Catalyst is based on functional programming constructs in Scala and designed with these key two purposes:

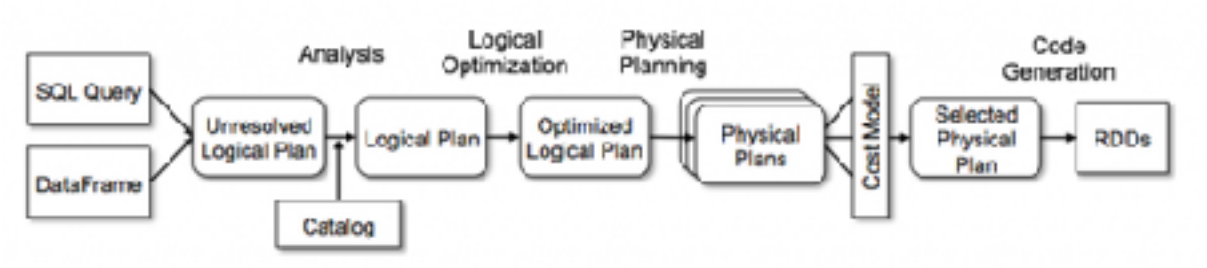
1. **Easily add new optimization techniques and features to Spark SQL**

- Catalyst contains a general library for representing trees and applying rules to manipulate them.
- On top of this framework, it has libraries specific to relational query processing (e.g., **expressions, logical query plans**), and several **sets of rules** that handle different phases of query execution:
  - Analysis,
  - Logical optimization,
  - Physical planning,
  - Code generation to compile parts of queries to Java bytecode

2. **Enable external developers to extend the optimizer**

   (e.g. adding data source specific rules, support for new data types, etc.)

a. it uses another Scala feature, **quasiquotes**,
b. that makes it easy to generate code at runtime from composable expressions.
c. Catalyst also offers several public extension points, including external data sources and user-defined types.
d. As well, Catalyst supports both rule-based and cost-based optimization



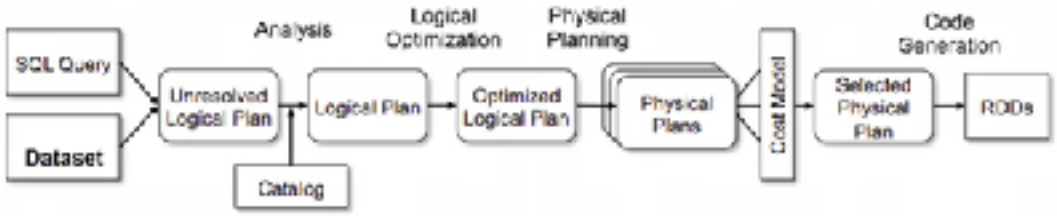
References:
https://databricks.com/glossary/catalyst-optimizer
https://databricks.com/blog/2015/04/13/deep-dive-into-spark-sqls-catalyst-optimizer.html

Example for Catalyst Optimizer:

Spark SQL Architecture

Terminology:

- Logical and Physical plans are trees representing query evaluation
- Logical plan is higher-level and algebraic
- Physical plan is lower-level and operational



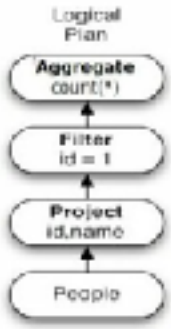
Catalyst Optimizer Phases

- Analysis:
  - resolve unresolved attribute references or relations
- Logical Optimization:
  - standard sql query optimizations like constant folding, predicate pushdown, project pruning are applied.
- Physical Planning:
  - one or more physical plans are formed from the optimized logical plan, using physical operator matching the Spark execution engine
- Code Generation:
  - fuses multiple physical operators together into a single optimized function
  - to avoid large amount of branches and virtual function calls, generates Java code to run on each machine
  - generated code is compiled using Janino compiler

Logical Plan Optimization Example

- An example query

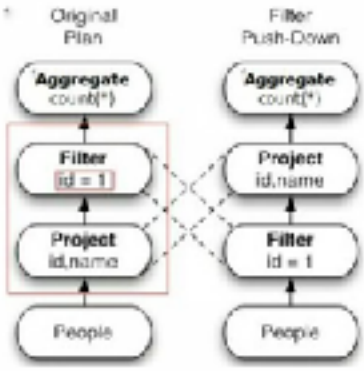
```
SELECT count(*)
FROM (
    SELECT id, name
    FROM People ) p
WHERE p.id = 1
```



Logical Plan Optimization Example (Contd..)

- Optimization Rules example

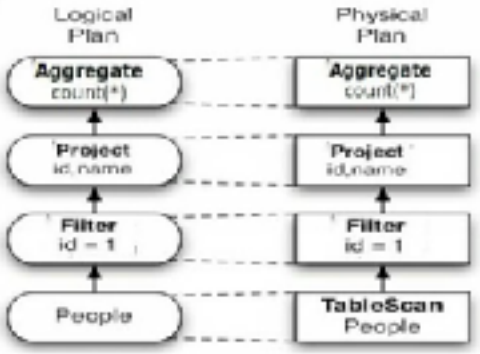
1. Find Filters on top of projections.
2. Check that the filter can be evaluated without the result of the project.
3. If so, switch the operators.



Physical Planning

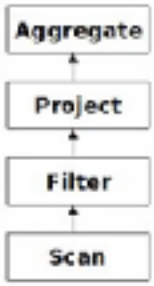
- Native query planning

```
SELECT count(*)
FROM (
    SELECT id, name
    FROM People ) p
WHERE p.id = 1
```



Catalyst Code Generation

- Without Code Generation, each of the operator in the tree has to be interpreted for each row of data by walking down the tree
  - large amounts of branches and virtual function calls
  - unnecessary memory allocations/de-allocations from creating new rows
- With Code Generation, this can be rewritten as one iterator which takes in a row
  - creates one row object per output row, and doesn't use any extraneous method calls
  - allows the compiler/processor optimizations to kick in




```
int count = 0;
for (Row r : rows) {
  if (r.filter) {
    count++;
  }
}
return new Row(count)
```

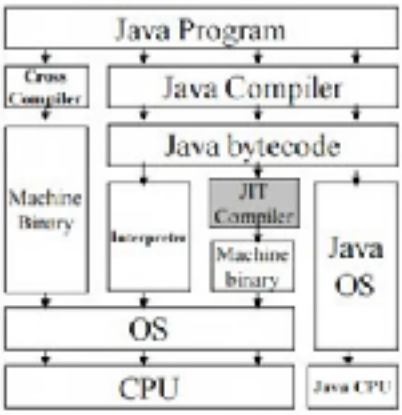
Spark SQL Catelyst Code Optimization using Function Outlining with Madhusudana...

Traditional Java Compilation and Execution

- A Java Compiler compiles high level Java source code to Java bytecode readable by Java Virtual Machine (JVM)
- JVM interprets bytecode to machine instructions at runtime
- Advantages
  - platform independence (JVM present on most machines)
- Drawbacks
  - needs memory
  - not as fast as running pre-compiled machine instructions

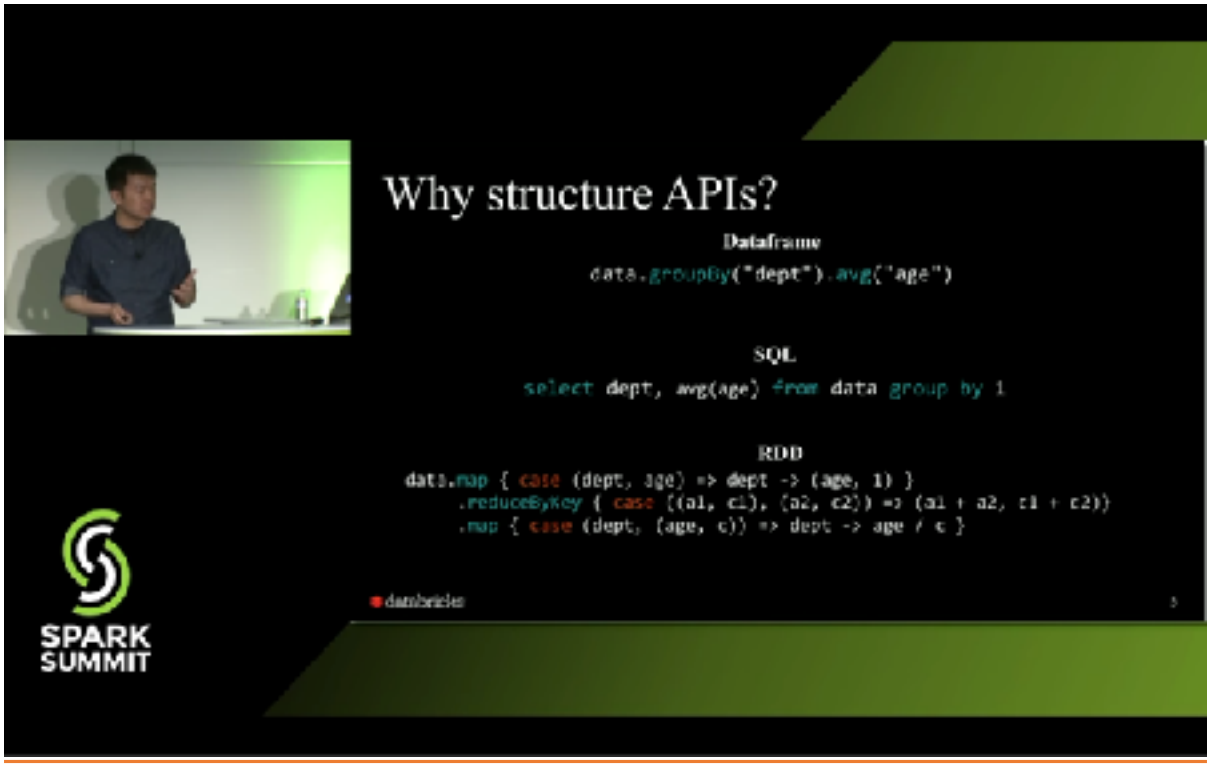
9:04 / 31:18

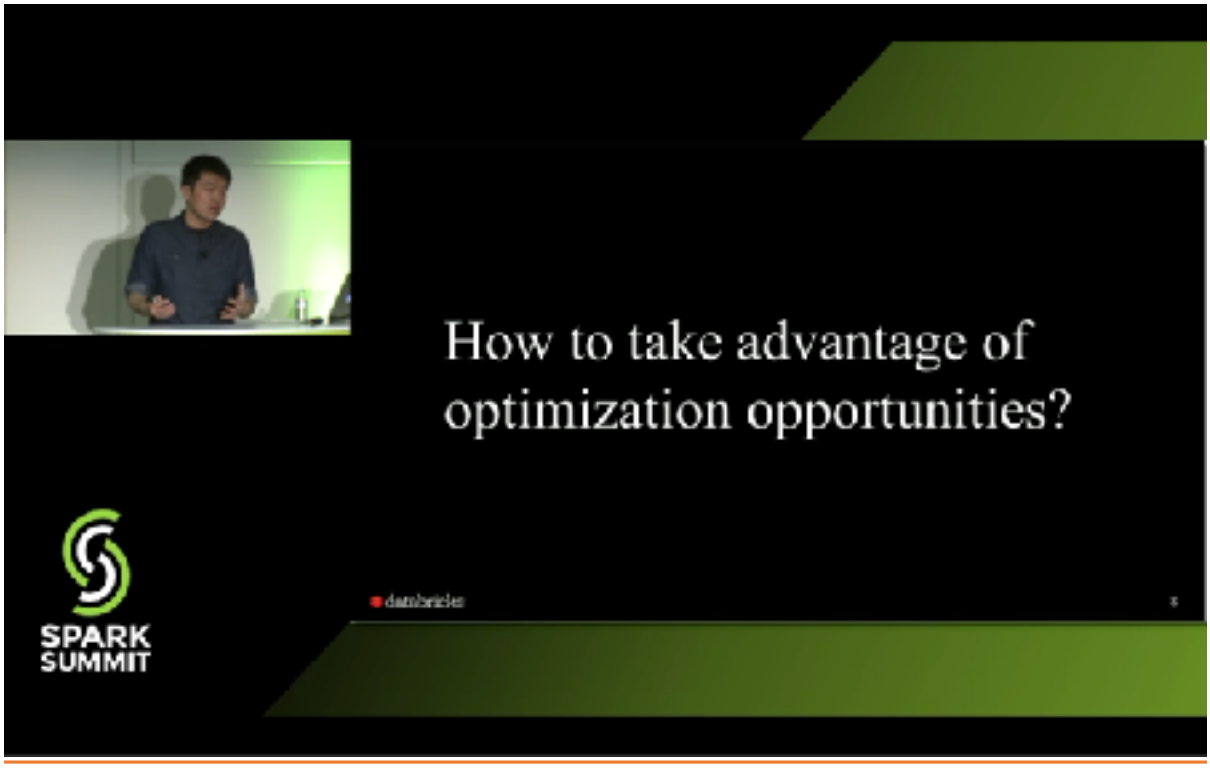
Java JIT Compiler Overview:



- A just-in-time (JIT) compiler is a compiler that compiles code into machine instructions during program execution, rather than ahead of time.
- Combines speed of compiled code w/ flexibility of interpretation
- JVM interprets the application code initially
- Collects execution statistics for each method
- Invokes JIT compiler for identified hot methods
- JIT code generally offers far better performance than interpreted code

Another Example for above **Catalyst: The Apache Spark Optimizer**





Get an optimizer that automatically finds out the most efficient plan to execute data operations specified in the user's program

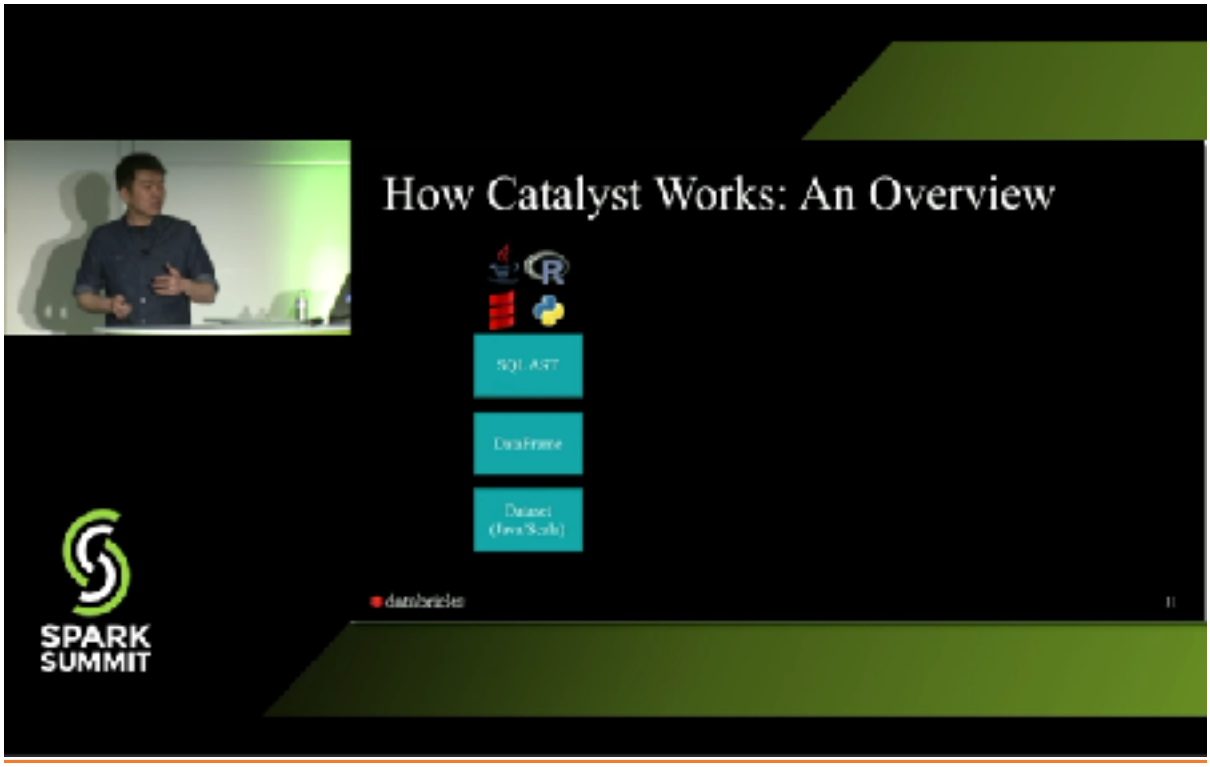
SPARK SUMMIT

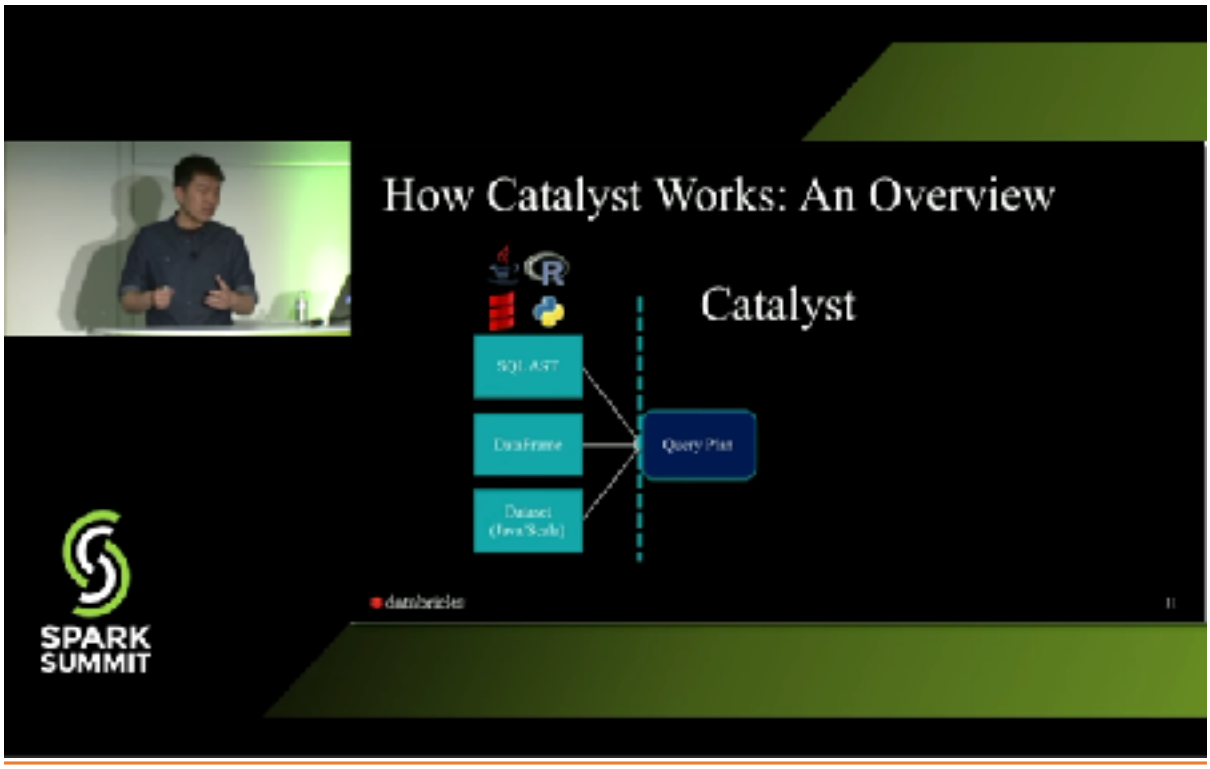
databricks

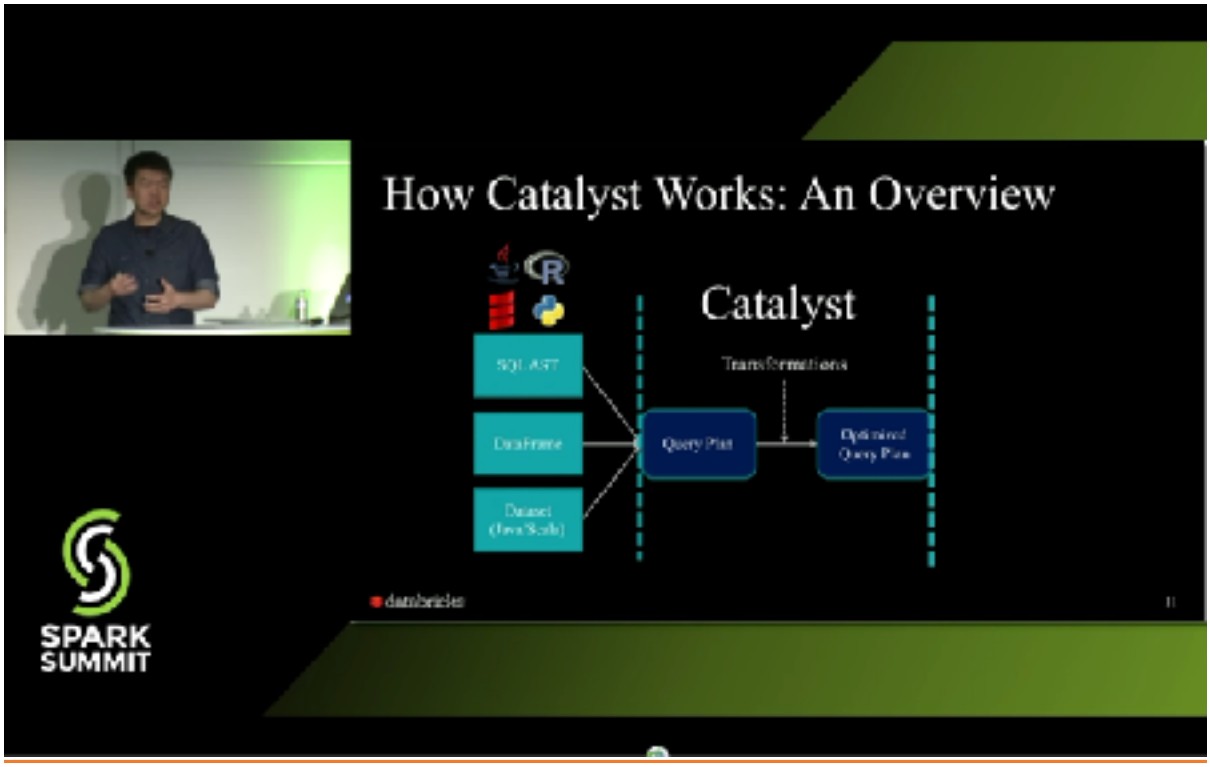
Catalyst:
Apache Spark's Optimizer

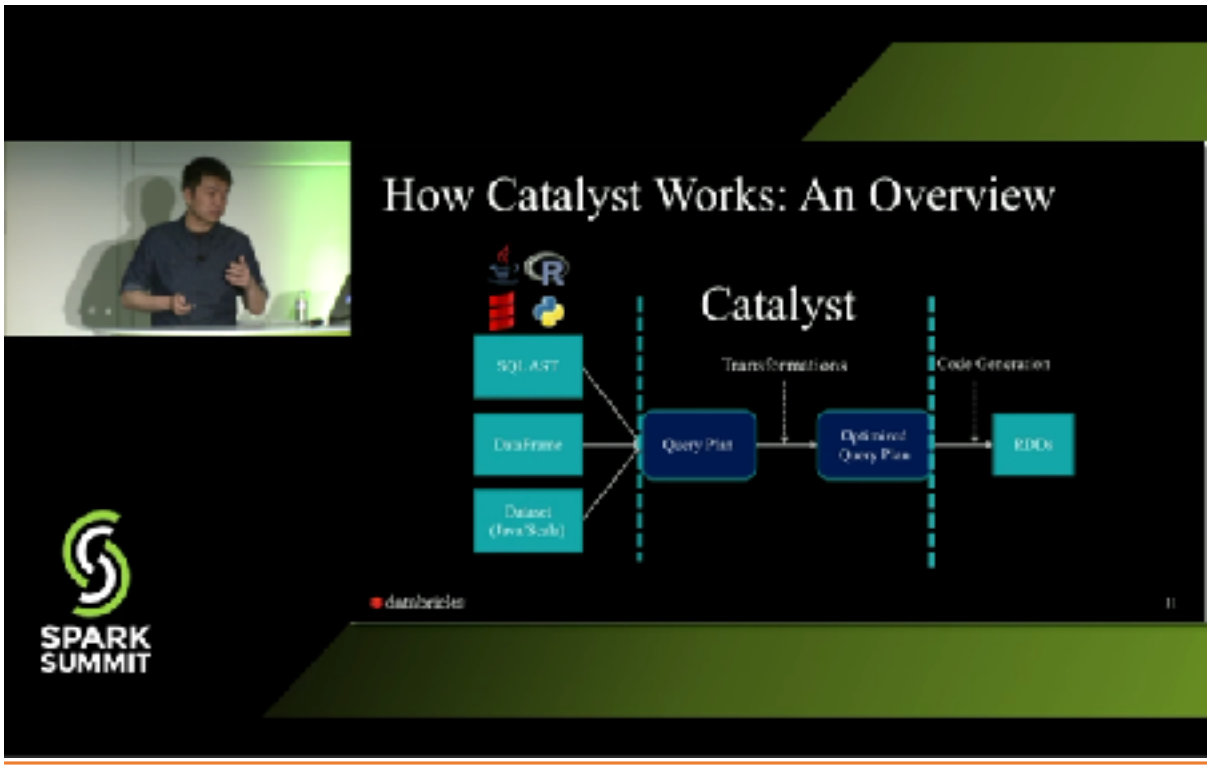
SPARK SUMMIT

databricks

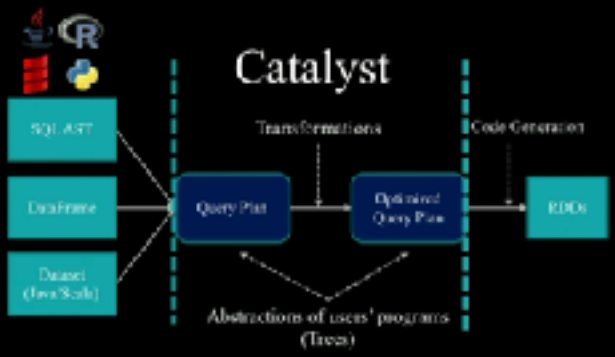

How Catalyst Works: An Overview
SQL AST
DataFrame
Dataset
(Java/Scala)
databricks
SPARK
SUMMIT


How Catalyst Works: An Overview
Catalyst
SQL AST
DataFrame
Dataset
(Java/Scala)
Query Plan
databricks
SPARK
SUMMIT


How Catalyst Works: An Overview
Catalyst
SQL AST
DataFrame
Dataset
(Java/Scala)
Query Plan
Optimized
Query Plan
Transformations
SPARK
SUMMIT
databricks


How Catalyst Works: An Overview
Catalyst
SQL AST
DataFrame
Dataset
(Java/Scala)
Query Plan
Optimized
Query Plan
RDDs
Transformations
Code Generation
SPARK
SUMMIT
databricks

How Catalyst Works: An Overview



SPARK SUMMIT

databricks

Trees: Abstractions of Users' Programs

```
SELECT sum(v)
FROM (
  SELECT
    t1.id,
    1 + 2 + t1.value AS v
  FROM t1 JOIN t2
  WHERE
    t1.id = t2.id AND
    t2.id > 50000) tmp
```

SPARK SUMMIT

databricks

Trees: Abstractions of Users' Programs

Expression

```
SELECT sum(v)
FROM (
  SELECT
    t1.id,
    1 + 2 + t1.value AS v
  FROM t1 JOIN t2
  WHERE
    t1.id = t2.id AND
    t2.id > 50000) tmp
```

SPARK SUMMIT

databricks

Trees: Abstractions of Users' Programs

Expression

```
SELECT sum(v)
FROM (
  SELECT
    t1.id,
    1 + 2 + t1.value AS v
  FROM t1 JOIN t2
  WHERE
    t1.id = t2.id AND
    t2.id > 50000) tmp
```

- An expression represents a new value, computed based on input values
  - e.g. `1 + 2 + t1.value`

SPARK SUMMIT

databricks

Trees: Abstractions of Users' Programs

Expression

```
SELECT sum(v)
FROM (
  SELECT
    t1.id,
    1 + 2 + t1.value AS v
  FROM t1 JOIN t2
  WHERE
    t1.id = t2.id AND
    t2.id > 50000) tmp
```

- An expression represents a new value, computed based on input values
  - e.g. `1 + 2 + t1.value`
- Attribute: A column of a dataset (e.g. `t1.id`) or a column generated by a specific data operation (e.g. `v`)

SPARK SUMMIT

Trees: Abstractions of Users' Programs

Query Plan

```
SELECT sum(v)
FROM (
  SELECT
    t1.id,
    1 + 2 + t1.value AS v
  FROM t1 JOIN t2
  WHERE
    t1.id = t2.id AND
    t2.id > 50000) tmp
```


SPARK SUMMIT

Trees: Abstractions of Users' Programs

Query Plan

```
SELECT sum(v)
FROM (
  SELECT
    t1.id,
    1 + 2 + t1.value AS v
  FROM t1 JOIN t2
  WHERE
    t1.id = t2.id AND
    t2.id > 50000) tmp
```

SPARK SUMMIT

Logical Plan

SPARK SUMMIT

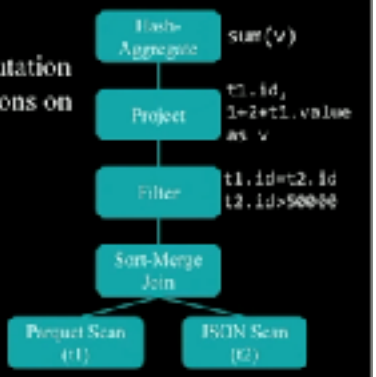
Logical Plan

- A Logical Plan describes computation on datasets **without** defining how to conduct the computation
- **output**: a list of attributes generated by this Logical Plan, e.g. `[id, v]`
- **constraints**: a set of invariants about the rows generated by this plan, e.g. `t2.id > 50000`
- `statistics`: `size of the plan in rows/bytes. Per column stats (min/max/ndv/nulls).`

SPARK SUMMIT

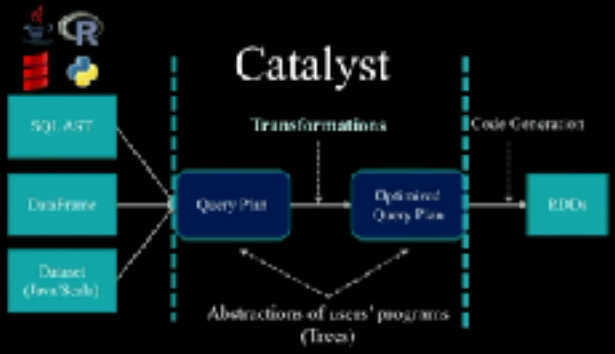
Physical Plan

- A Physical Plan describes computation on datasets with specific definitions on how to conduct the computation
- A Physical Plan is executable



SPARK SUMMIT

How Catalyst Works: An Overview



SPARK SUMMIT

Transformations

SPARK SUMMIT

Transformations

- Transformations without changing the tree type (Transform and Rule Executor)
  - Expression => Expression
  - Logical Plan => Logical Plan
  - Physical Plan => Physical Plan
- Transforming a tree to another kind of tree
  - Logical Plan => Physical Plan

SPARK SUMMIT

Transform

- A function associated with every tree used to implement a single rule

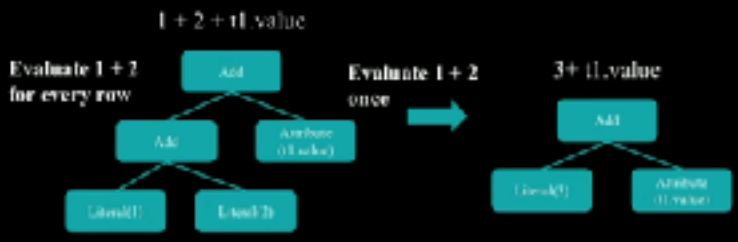
1 + 2 + nl value

SPARK SUMMIT

SPARK SUMMIT

Transform

- A function associated with every tree used to implement a single rule



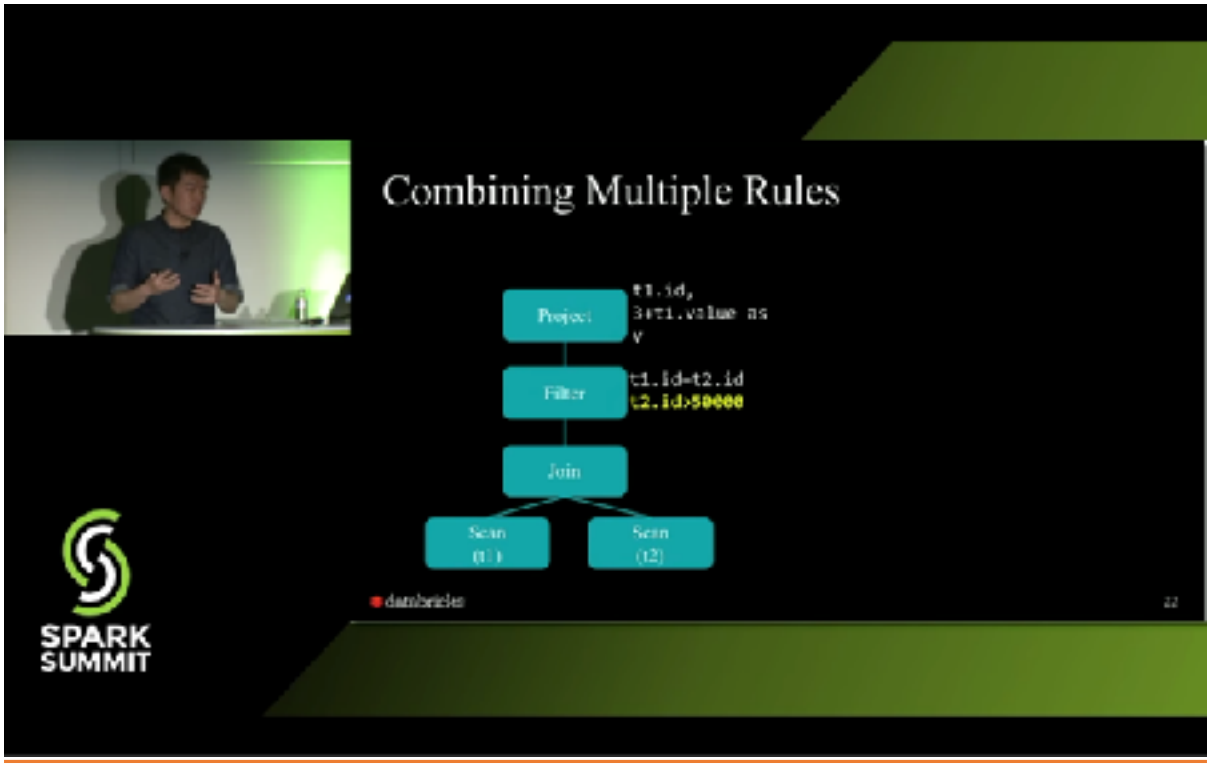
SPARK SUMMIT

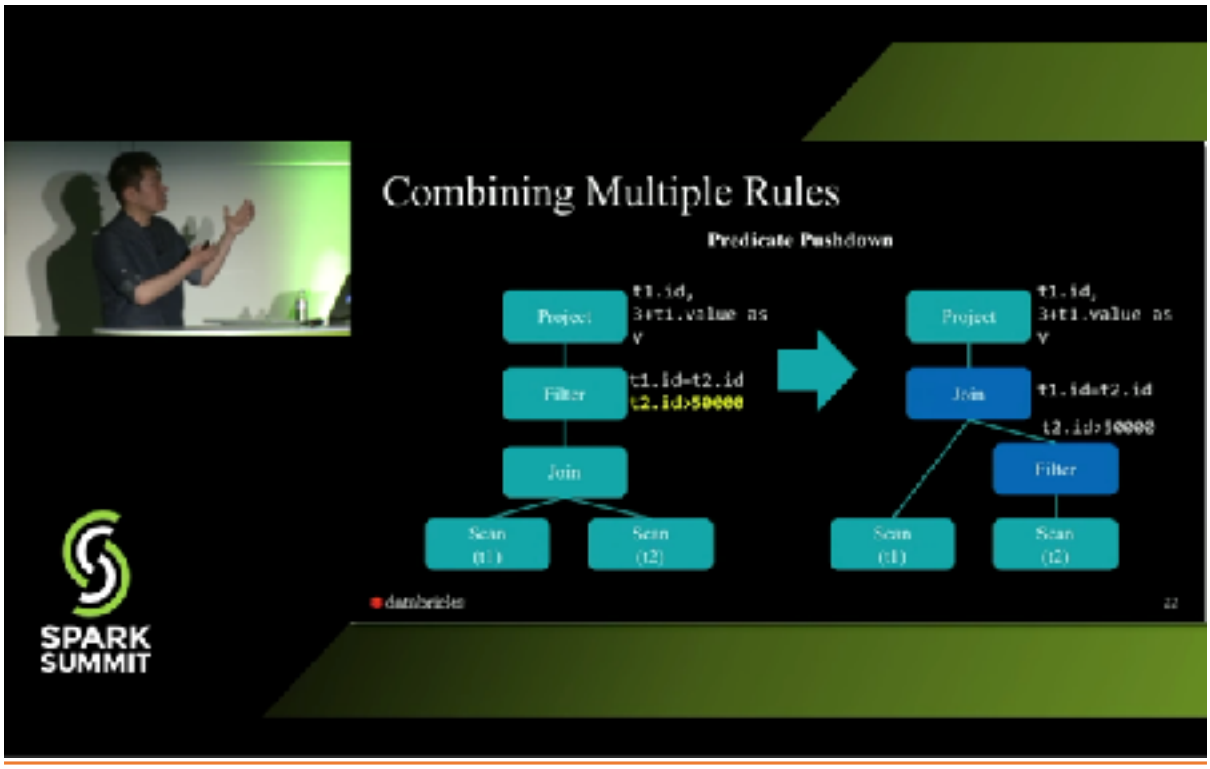
Transform

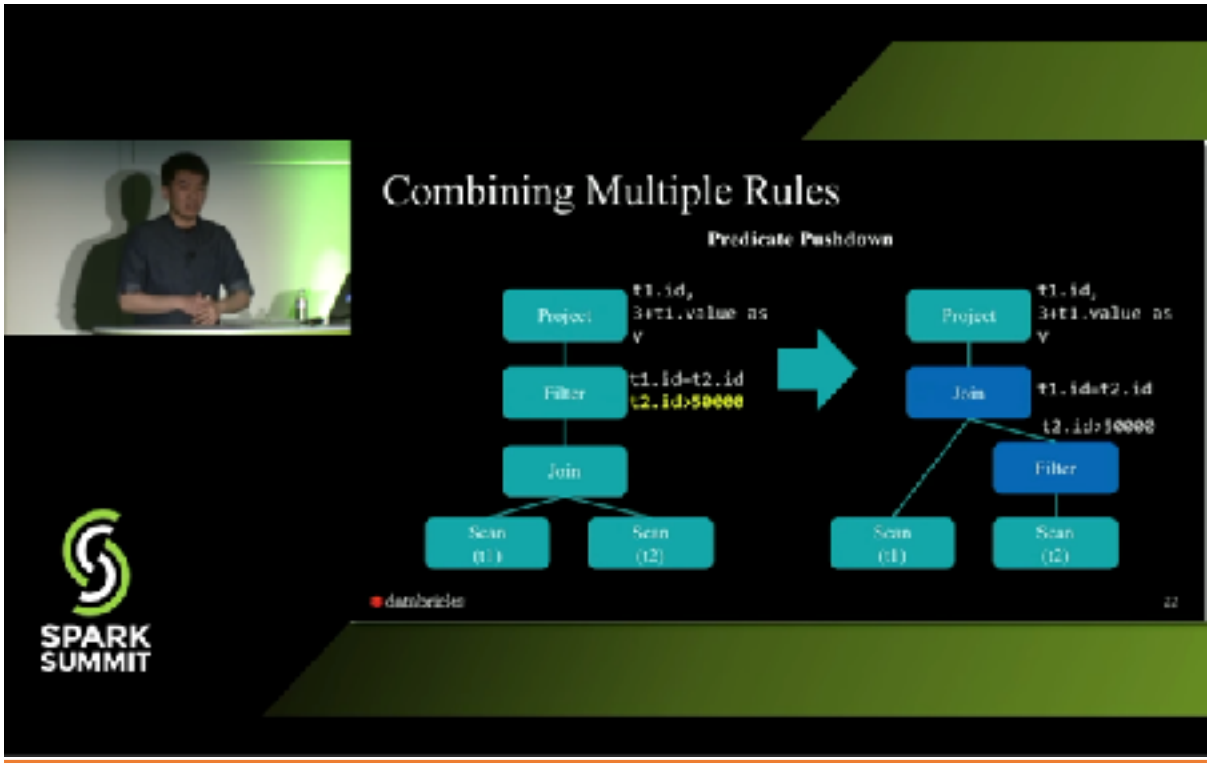
- A transformation is defined as a Partial Function
- Partial Function: A function that is defined for a subset of its possible arguments

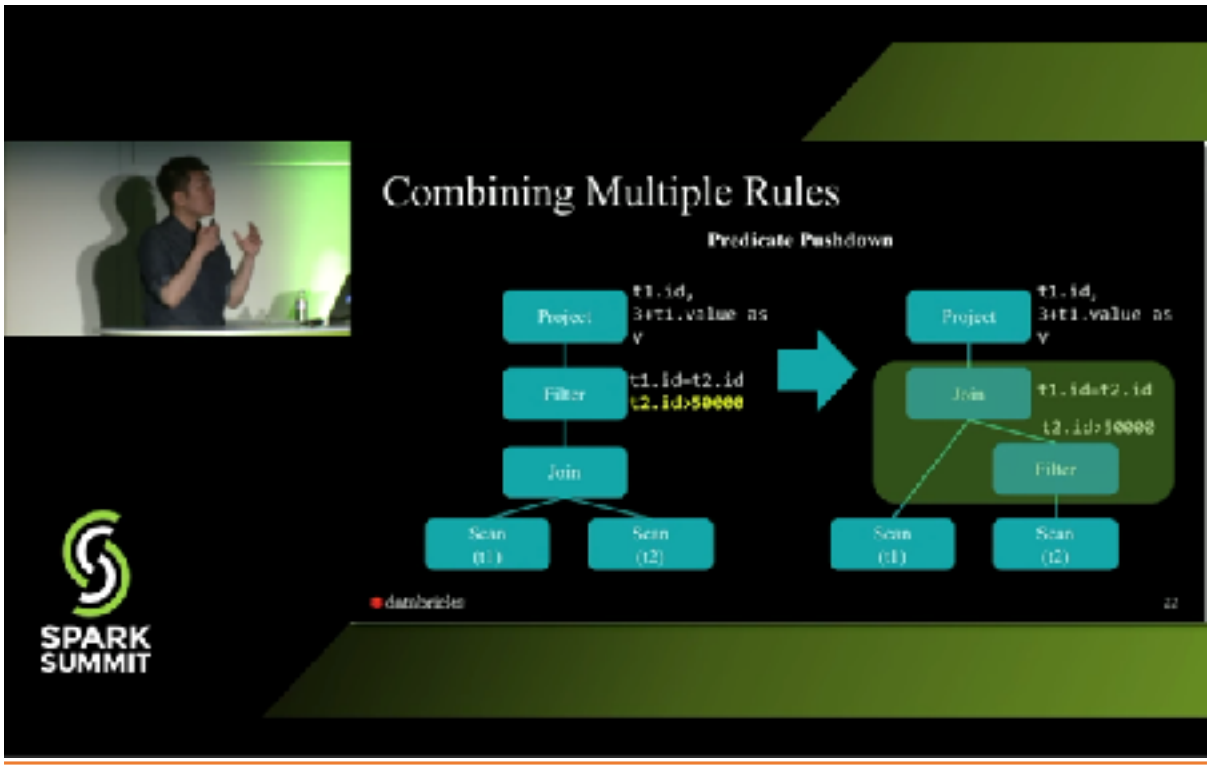
```
val expression: Expression = ...
expression.transform {
  case Add(Literal(x, IntegerType), Literal(y, IntegerType)) =>
    Literal(x + y)
}
```

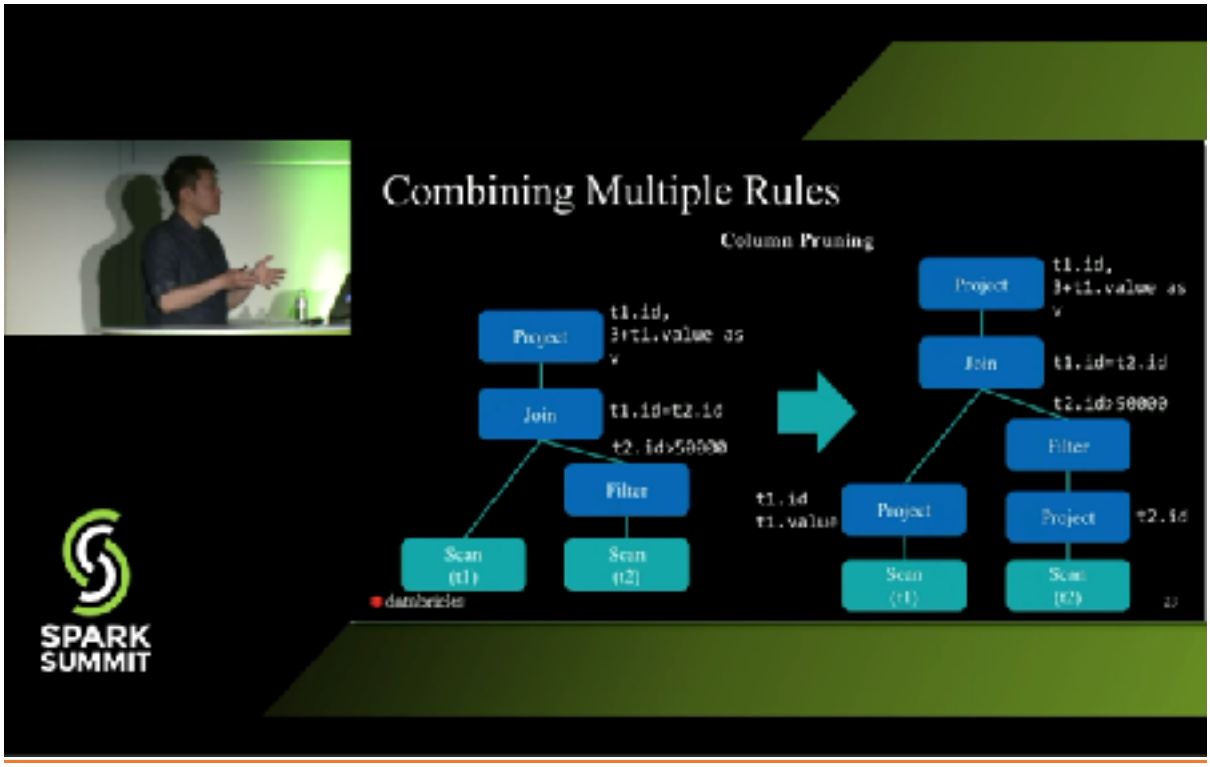
Case statement determines if the partial function is defined for a given input

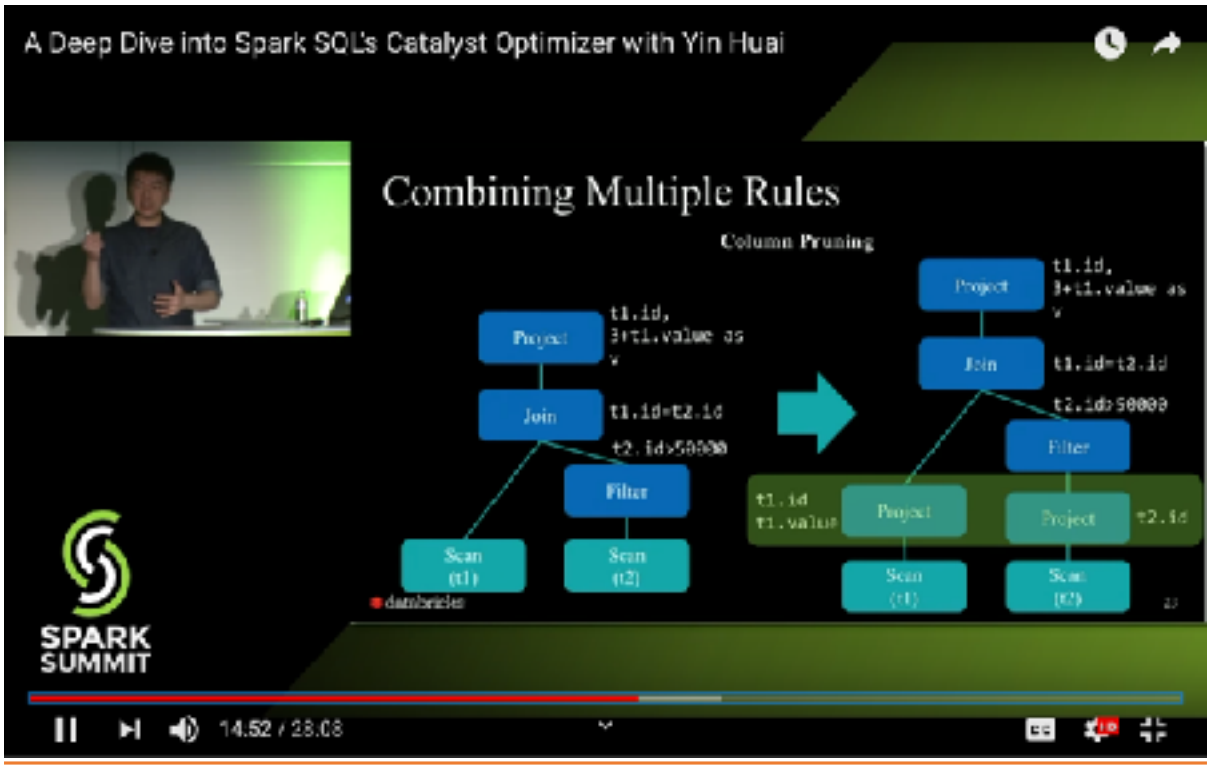

Combining Multiple Rules
Project
t1.id,
3+t1.value as
v
Filter
t1.id=t2.id
t2.id>50000
Join
Scan
(t1)
Scan
(t2)
SPARK
SUMMIT
databricks
22

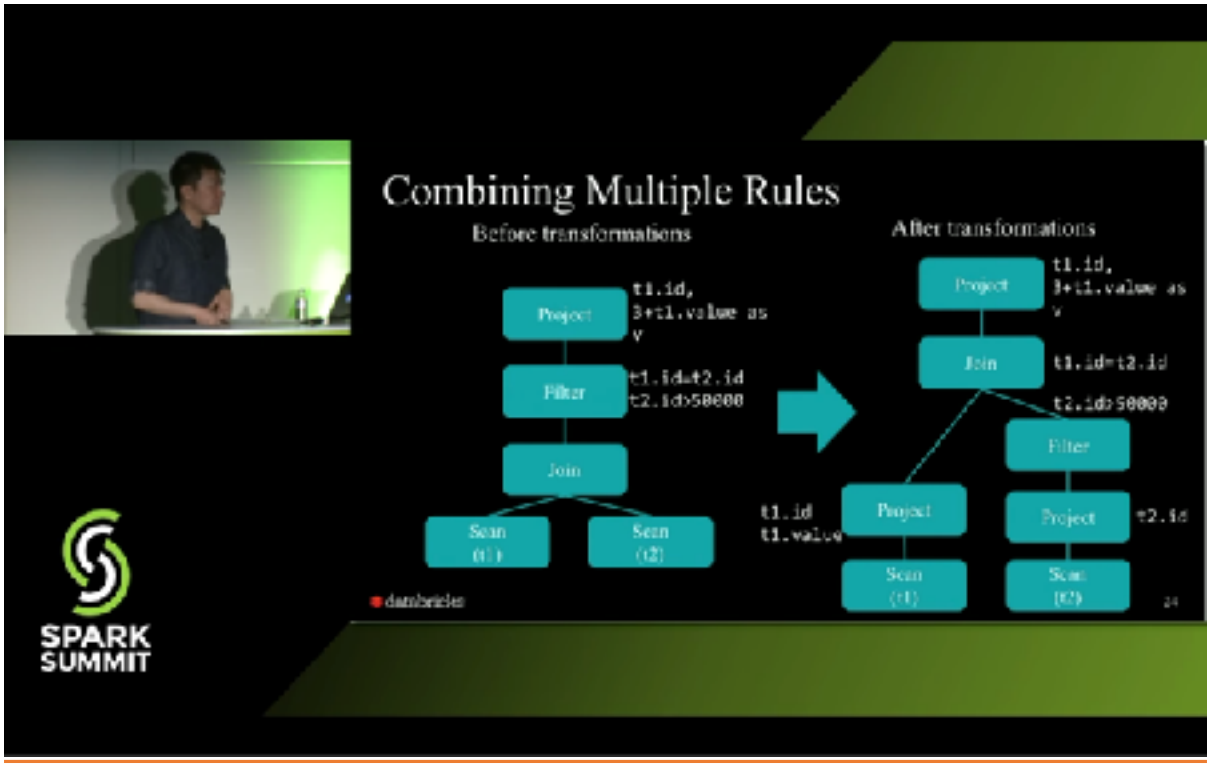

Combining Multiple Rules
Predicate Pushdown
Project
t1.id,
3+t1.value as
v
Filter
t1.id=t2.id
t2.id>50000
Join
Scan
(t1)
Scan
(t2)
Project
t1.id,
3+t1.value as
v
Join
t1.id=t2.id
t2.id>50000
Filter
Scan
(t1)
Scan
(t2)
SPARK
SUMMIT
databricks
22

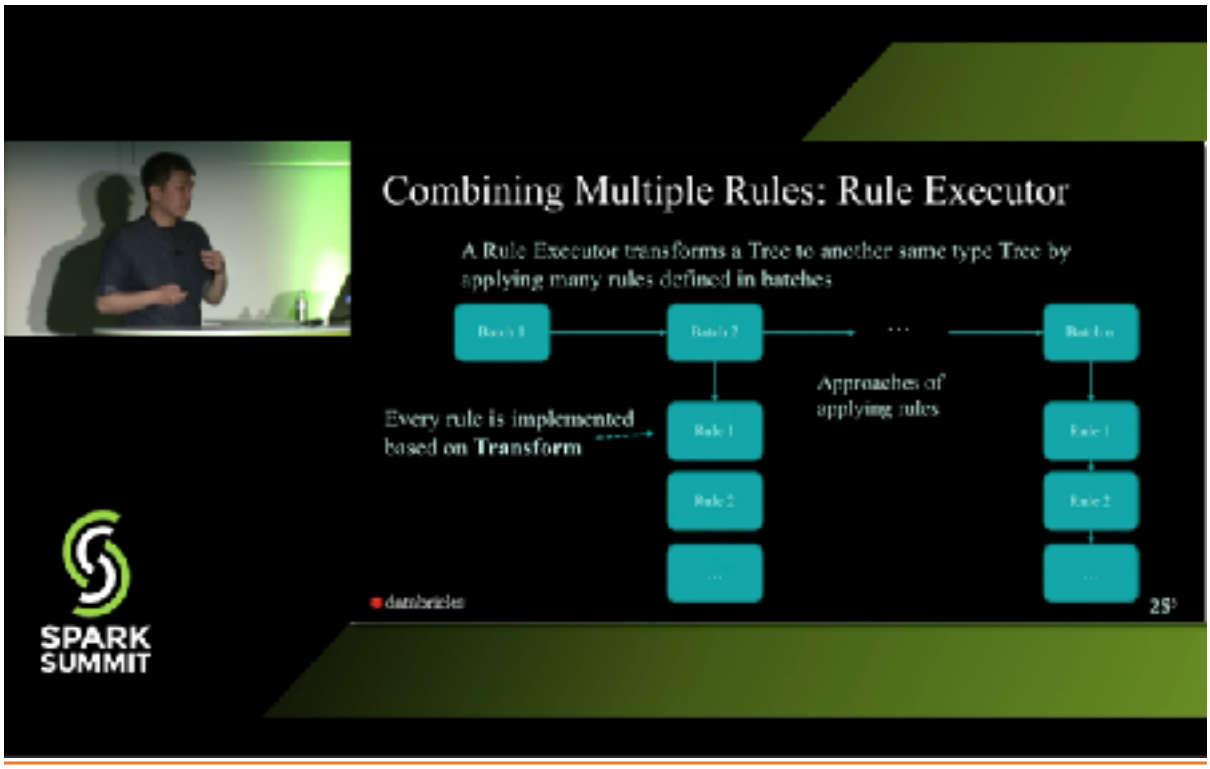

Combining Multiple Rules
Predicate Pushdown
Project
t1.id,
3+t1.value as
v
Filter
t1.id=t2.id
t2.id>50000
Join
Scan
(t1)
Scan
(t2)
Project
t1.id,
3+t1.value as
v
Join
t1.id=t2.id
t2.id>10000
Filter
Scan
(t1)
Scan
(t2)
SPARK
SUMMIT
22


Combining Multiple Rules
Predicate Pushdown
Project
t1.id,
3+t1.value as
v
Filter
t1.id=t2.id
t2.id>50000
Join
Scan
(t1)
Scan
(t2)
Project
t1.id,
3+t1.value as
v
Join
t1.id=t2.id
t2.id>10000
Filter
Scan
(t1)
Scan
(t2)
SPARK
SUMMIT
22


Combining Multiple Rules
Column Pruning
Project
Join
Filter
Scan (t1)
Scan (t2)
t1.id=t2.id
t2.id>50000
t1.id
t1.value
t2.id
SPARK SUMMIT
databricks

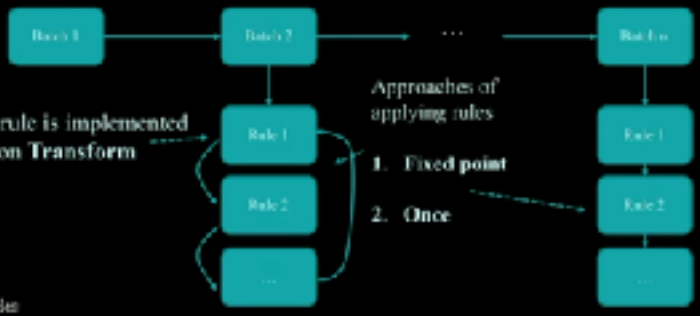

A Deep Dive into Spark SQL's Catalyst Optimizer with Yin Huai
Combining Multiple Rules
Column Pruning
Project
Join
Filter
Scan (t1)
Scan (t2)
t1.id=t2.id
t2.id>50000
t1.id
t1.value
t2.id
SPARK SUMMIT
databricks
14:52 / 28:08


Combining Multiple Rules
Before transformations
After transformations
Project
t1.id,
3+t1.value as
v
Filter
t1.id=t2.id
t2.id>50000
Join
Scan
(t1)
Scan
(t2)
Project
Join
Filter
Project
Scan
(t1)
Scan
(t2)
SPARK
SUMMIT
databricks


Combining Multiple Rules: Rule Executor
A Rule Executor transforms a Tree to another same type Tree by
applying many rules defined in batches
Every rule is implemented
based on Transform
Approaches of
applying rules
Rule 1
Rule 2
SPARK
SUMMIT
databricks

Combining Multiple Rules: Rule Executor

A Rule Executor transforms a Tree to another same type Tree by applying many rules defined in batches



SPARK SUMMIT

Transformations

- Transformations without changing the tree type (Transform and Rule Executor)
  - Expression => Expression
  - Logical Plan => Logical Plan
  - Physical Plan => Physical Plan
- Transforming a tree to another kind of tree
  - Logical Plan => Physical Plan

SPARK SUMMIT

From Logical Plan to Physical Plan

SPARK SUMMIT

From Logical Plan to Physical Plan

- A Logical Plan is transformed to a Physical Plan by applying a set of Strategies
- Every Strategy uses pattern matching to convert a Logical Plan to a Physical Plan

```
object BasicOperators extends Strategy {
  def apply(plan: LogicalPlan): Seq[SparkPlan] = plan match {
    …
    case logical.Project(projectList, child) =>
      execution.ProjectExec(projectList, planLater(child)) :: Nil
    case logical.Filter(condition, child) =>
      execution.FilterExec(condition, planLater(child)) :: Nil
    …
  }
}
```


SPARK SUMMIT

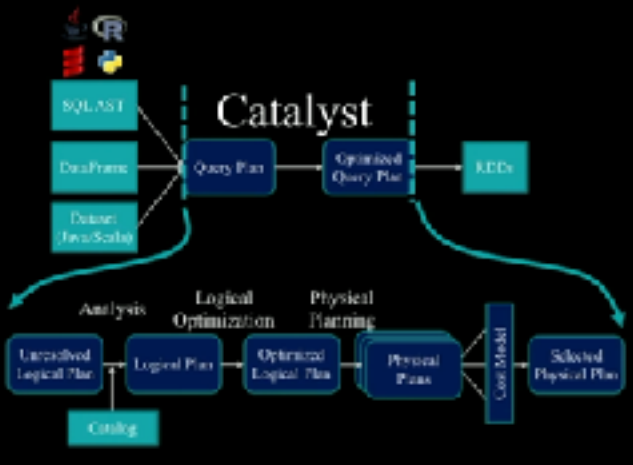
From Logical Plan to Physical Plan

- A Logical Plan is transformed to a Physical Plan by applying a set of Strategies
- Every Strategy uses pattern matching to convert a Logical Plan to a Physical Plan

```
object BasicOperators extends Strategy {
  def apply(plan: LogicalPlan): Seq[SparkPlan] = plan match {
    ...
    case logical.Project(projectList, child) =>
      execution.ProjectExec(projectList, planLater(child)) :: Nil
    case logical.Filter(condition, child) =>
      execution.FilterExec(condition, planLater(child)) :: Nil
    ...
  }
}
```

Triggers other Strategies

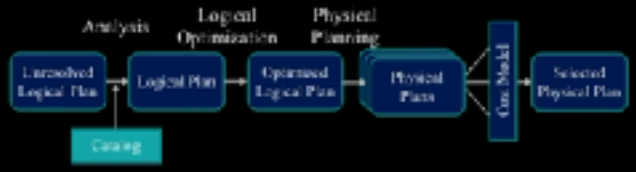
SPARK SUMMIT



SPARK SUMMIT

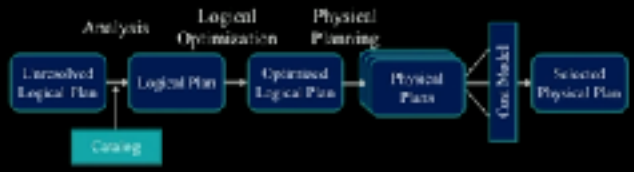
- **Analysis (Rule Executor):** Transforms an Unresolved Logical Plan to a Resolved Logical Plan

SPARK SUMMIT



- **Analysis (Rule Executor):** Transforms an Unresolved Logical Plan to a Resolved Logical Plan
  - Unresolved => Resolved: Use Catalog to find where datasets and columns are coming from and types of columns
- **Logical Optimization (Rule Executor):** Transforms a Resolved Logical Plan to an Optimized Logical Plan

SPARK SUMMIT



- **Analysis (Rule Executor):** Transforms an Unresolved Logical Plan to a Resolved Logical Plan
  - Unresolved => Resolved: Use Catalog to find where datasets and columns are coming from and types of columns
- **Logical Optimization (Rule Executor):** Transforms a Resolved Logical Plan to an Optimized Logical Plan
- **Physical Planning (Strategies + Rule Executor):**
  - **Phase 1:** Transforms an Optimized Logical Plan to a Physical Plan
  - **Phase 2:** Rule executor is used to adjust the physical plan to make it ready for execution

SPARK SUMMIT

Put what we have learned in action

SPARK SUMMIT

Roll your own Planner Rule

```
import org.apache.spark.sql.functions._

 // tableA is a dataset of integers in the range of [0, 19999999]
val tableA = spark.range(20000000).as('a)
// tableB is a dataset of integers in the range of [0, 9999999]
val tableB = spark.range(10000000).as('b)
// result shows the number of records after joining tableA and tableB
val result = tableA
  .join(tableB, $"a.id" === $"b.id")
  .groupBy()
  .count()
result.show()
```


SPARK SUMMIT

Roll your own Planner Rule

```
import org.apache.spark.sql.functions._

 // tableA is a dataset of integers in the range of [0, 19999999]
val tableA = spark.range(20000000).as('a)
// tableB is a dataset of integers in the range of [0, 9999999]
val tableB = spark.range(10000000).as('b)
// result shows the number of records after joining tableA and tableB
val result = tableA
  .join(tableB, $"a.id" === $"b.id")
  .groupBy()
  .count()
result.show()
```

This takes 4-8s on Databricks Community edition

SPARK SUMMIT

Roll your own Planner Rule

```
result.explain()

== Physical Plan ==
*HashAggregate(keys=[], functions=[count(1)])
+- Exchange SinglePartition
   +- *HashAggregate(keys=[], functions=[partial_count(1)])
      +- *Project
         +- *SortMergeJoin [id#642L], [id#646L], Inner
            :- *Sort [id#642L ASC NULLS FIRST], false, 0
            :  +- Exchange hashpartitioning(id#642L, 200)
            :     +- *Range (0, 20000000, step=1, splits=8)
            +- *Sort [id#646L ASC NULLS FIRST], false, 0
               +- Exchange hashpartitioning(id#646L, 200)
                  +- *Range (0, 10000000, step=1, splits=8)
```

SPARK SUMMIT

databricks

Roll your own Planner Rule

```
result.explain()

== Physical Plan ==
*HashAggregate(keys=[], functions=[count(1)])
+- Exchange SinglePartition
   +- *HashAggregate(keys=[], functions=[partial_count(1)])
      +- *Project
         +- *SortMergeJoin [id#642L], [id#646L], Inner
            :- *Sort [id#642L ASC NULLS FIRST], false, 0
            :  +- Exchange hashpartitioning(id#642L, 200)
            :     +- *Range (0, 20000000, step=1, splits=8)
            +- *Sort [id#646L ASC NULLS FIRST], false, 0
               +- Exchange hashpartitioning(id#646L, 200)
                  +- *Range (0, 10000000, step=1, splits=8)
```

SPARK SUMMIT

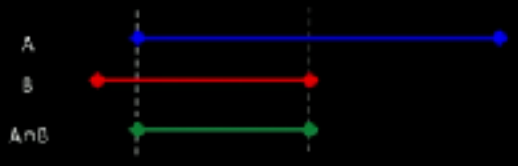
databricks

SPARK SUMMIT

Roll your own Planner Rule

Exploit the structure of the problem

We are joining two intervals; the result will be the intersection of these intervals



SPARK SUMMIT

Roll your own Planner Rule

```
// Import internal APIs of Catalyst
import org.apache.spark.sql.Strategy
import org.apache.spark.sql.catalyst.expressions.{Alias, Literal}
import org.apache.spark.sql.catalyst.plans.logical.{LogicalPlan, Join, Range}
import org.apache.spark.sql.catalyst.plans.Inner
import org.apache.spark.sql.execution.{ProjectExec, RangeExec, SparkPlan}

case object IntervalJoin extends Strategy with Serializable {
  def apply(plan: LogicalPlan): Seq[SparkPlan] = plan match {
    case Join(
      Range(start1, end1, 1, part1, Seq(o1)), // matches tableA
      Range(start2, end2, 1, part2, Seq(o2)), // matches tableB
      Inner, Some(EqualTo(e1, e2)))           // matches the join
        if ((o1 semanticEquals e1) && (o2 semanticEquals e2)) ||
           ((o1 semanticEquals e2) && (o2 semanticEquals e1)) =>
        // See next page for rule body
    case _ => Nil
  }
```

Roll your own Planner Rule

```
// matches cases like:
// table1: start1|-------------------------|end1
// table2: ...|-------------------|end2
if ((end1 >= start1) && (end1 <= end2)) {
  // start of the intersection
  val start = math.max(start1, start2)
  // end of the intersection
  val end = math.min(end1, end2)
  val part = math.max(part1.getOrElse(200), part2.getOrElse(200))
  // Create a new Range to represent the intersection
  val result = RangeExec(Range(start, end, 1, Some(part), o1 :: Nil))
  val twoColumns = ProjectExec(
    Alias(o1, o1.name)(exprId = o1.exprId) :: Nil,
    result)
  twoColumns :: Nil
} else {
  Nil
}
```

SPARK SUMMIT

Roll your own Planner Rule

Hook it up with Spark

```
spark.experimental.extraStrategies = IntervalJoin :: Nil
```

Use it

```
result.count()
```

This now takes ~0.5s to complete

SPARK SUMMIT

Roll your own Planner Rule

```
result.explain()

== Physical Plan ==
*HashAggregate(keys=[], functions=[count(1)])
+- Exchange SinglePartition
   +- *HashAggregate(keys=[], functions=[partial_count(1)])
      +- *Project
         +- *Project [id#642L AS id#642L]
            +- *Range (0, 100000000, step=1, splits=8)
```

SPARK SUMMIT

Roll your own Planner Rule

```
result.explain()

== Physical Plan ==
*HashAggregate(keys=[], functions=[count(1)])
+- Exchange SinglePartition
   +- *HashAggregate(keys=[], functions=[partial_count(1)])
      +- *Project
         +- *Project [id#642L AS id#642L]
            +- *Range (0, 100000000, step=1, splits=8)
```

SPARK SUMMIT

databricks 35

Tungsten :

improving the efficiency of *memory and CPU* for Spark applications,
[TMC- (**T**ungsten – **M**emory -**C**PU)
TMC- M2-BP-CA-CG
M2 = **M**emory **M**anagement
BP- **B**inary **P**rocessing
CA- **C**ache **A**ware
CG- **C**ode **G**enetation
]
Project Tungsten: Bringing Apache Spark efficiency Closer to Bare Metal

Project Tungsten will be the largest change to Spark's execution engine since the project's inception. It focuses on substantially improving the efficiency of *memory and CPU* for Spark applications, to push performance closer to the limits of modern hardware.

This effort includes three initiatives:

1. **Memory Management and Binary Processing:**

   Leveraging application semantics to manage memory explicitly and eliminate the overhead of JVM object model and garbage collection

2. **Cache-aware computation:**

   Algorithms and data structures to exploit/utilize memory hierarchy

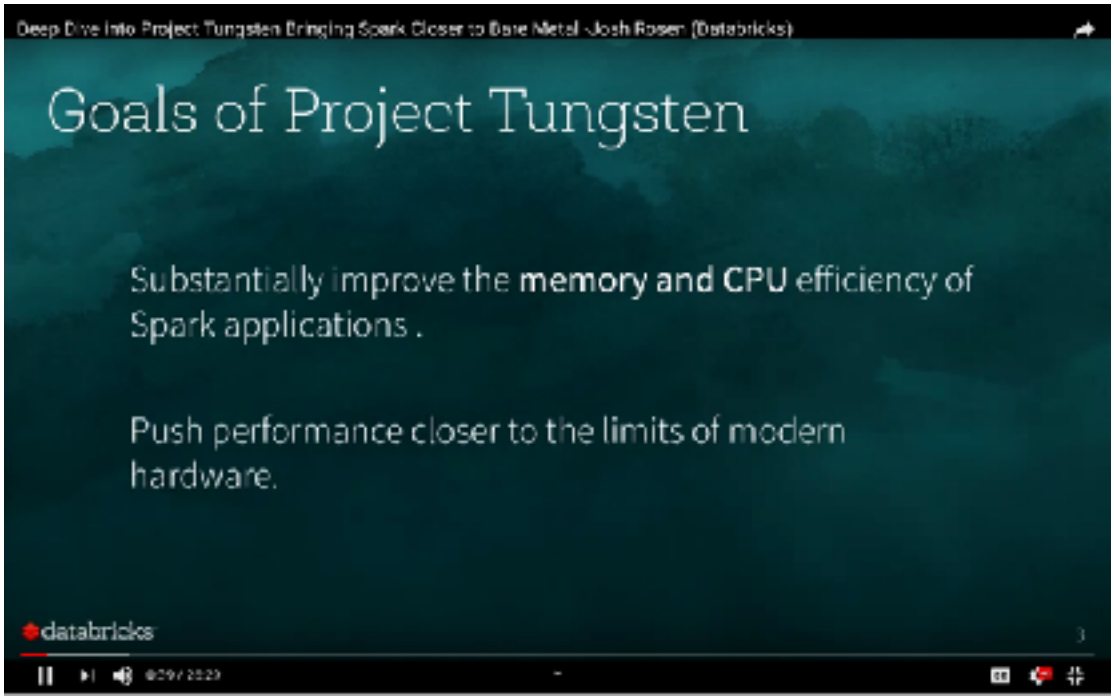
3. **Code generation:**

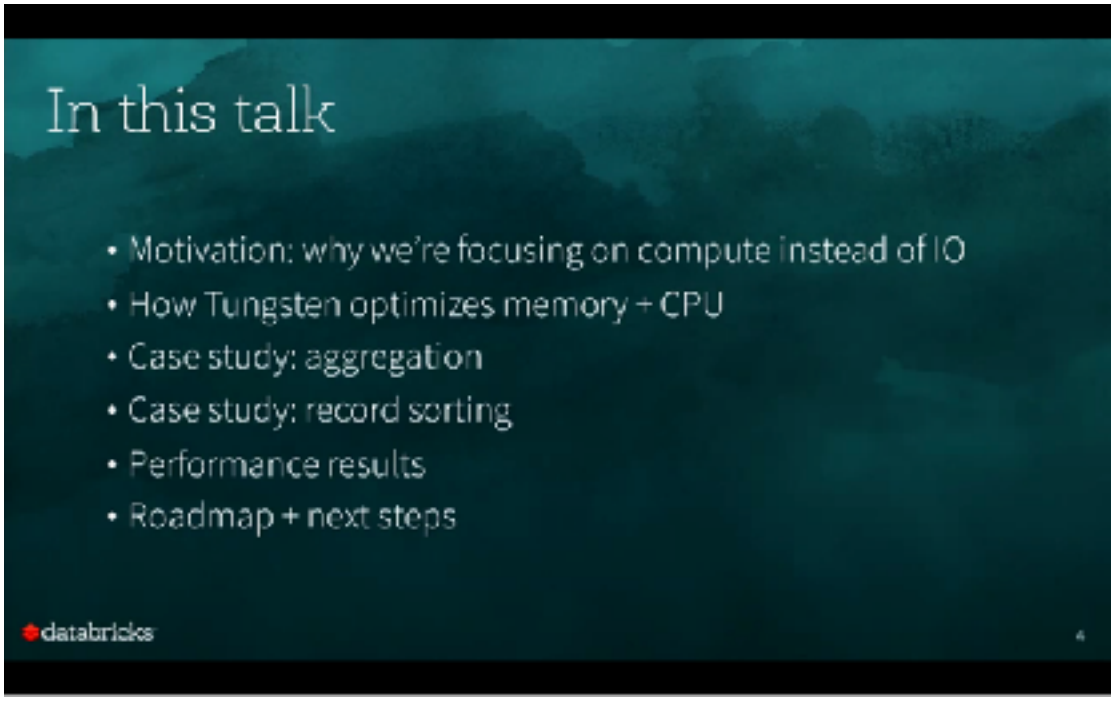
   Using code generation to exploit/utlize modern compilers and CPUs

Project tungsten uses **Janino compiler** to reduce code generate time.

Reference

https://www.youtube.com/watch?v=5ajs8EIPWGI


Deep Dive into Project Tungsten Bringing Spark Closer to Bare Metal -Josh Rosen (Databricks)
Goals of Project Tungsten
Substantially improve the memory and CPU efficiency of Spark applications .
Push performance closer to the limits of modern hardware.
databricks


In this talk
• Motivation: why we're focusing on compute instead of IO
• How Tungsten optimizes memory + CPU
• Case study: aggregation
• Case study: record sorting
• Performance results
• Roadmap + next steps
databricks

Many big data workloads are now compute bound

NSDI'15:

Making Sense of Performance in Data Analytics Frameworks

Kay Ousterhout*, Ryan Rasti*[1], Sylvia Ratnasamy*, Scott Shenker*[1], Byung-Gon Chun†
*UC Berkeley, [1]ICSI, †VMware, †Seoul National University

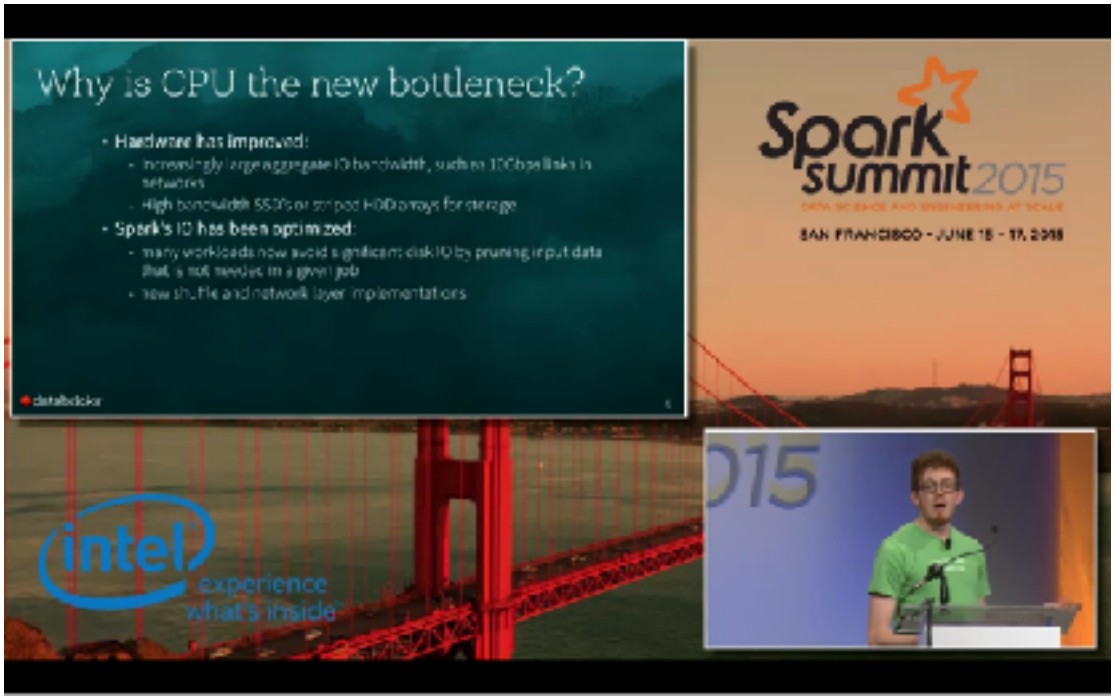
Many big data workloads are now compute bound

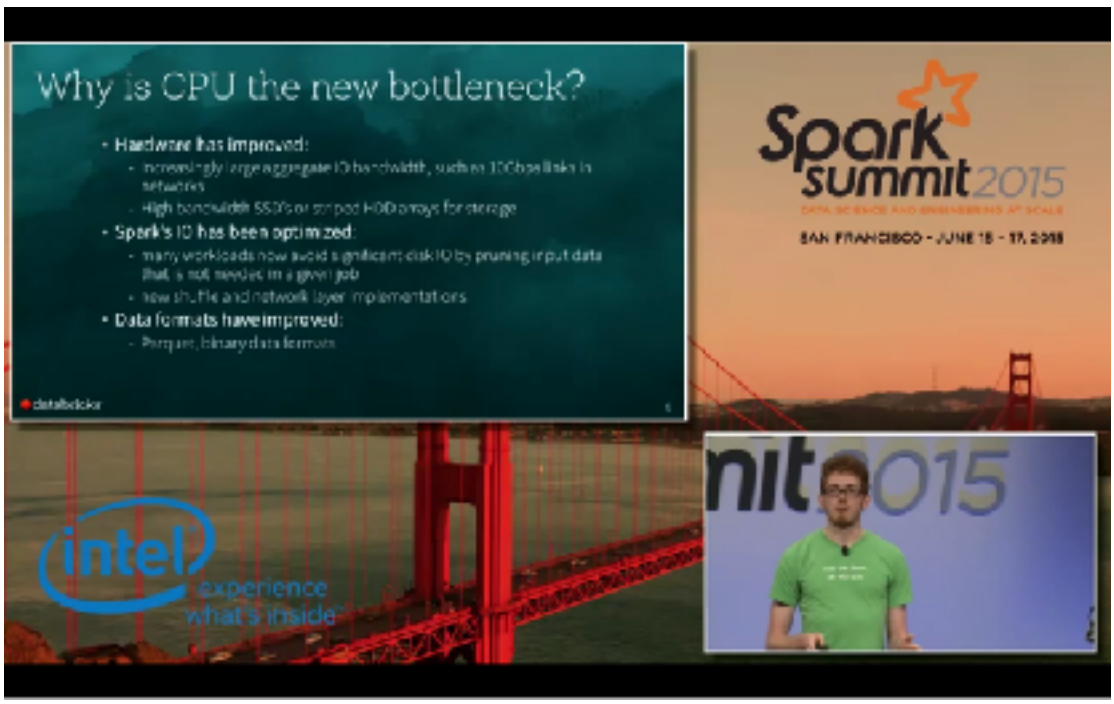
NSDI'15:

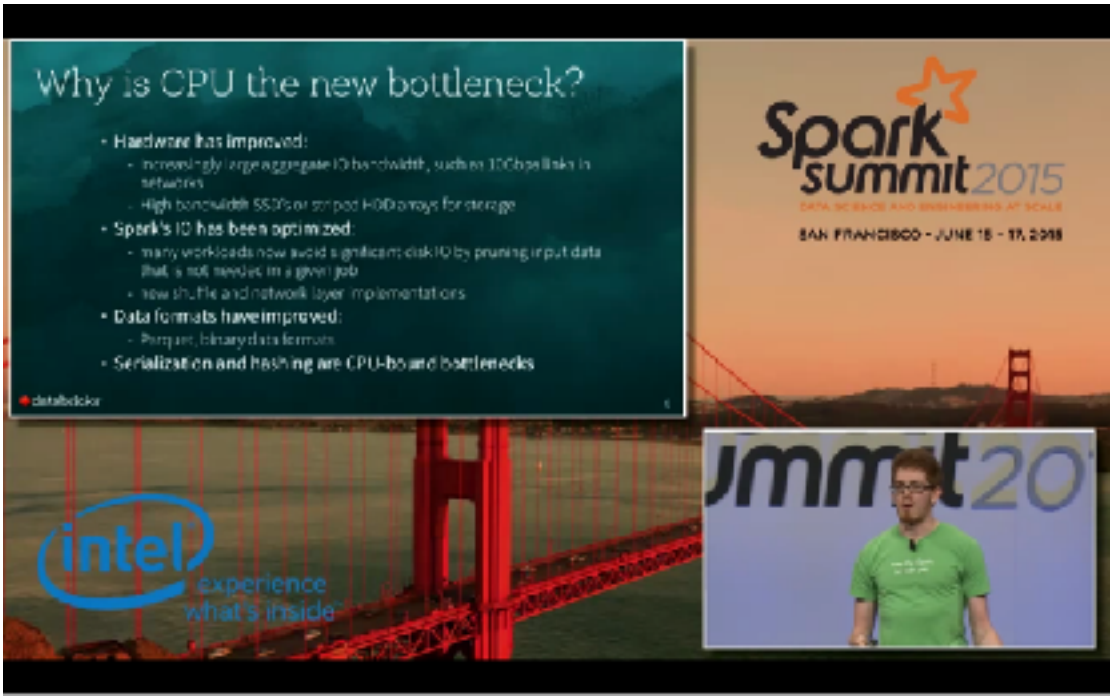
Making Sense of Performance in Data Analytics Frameworks

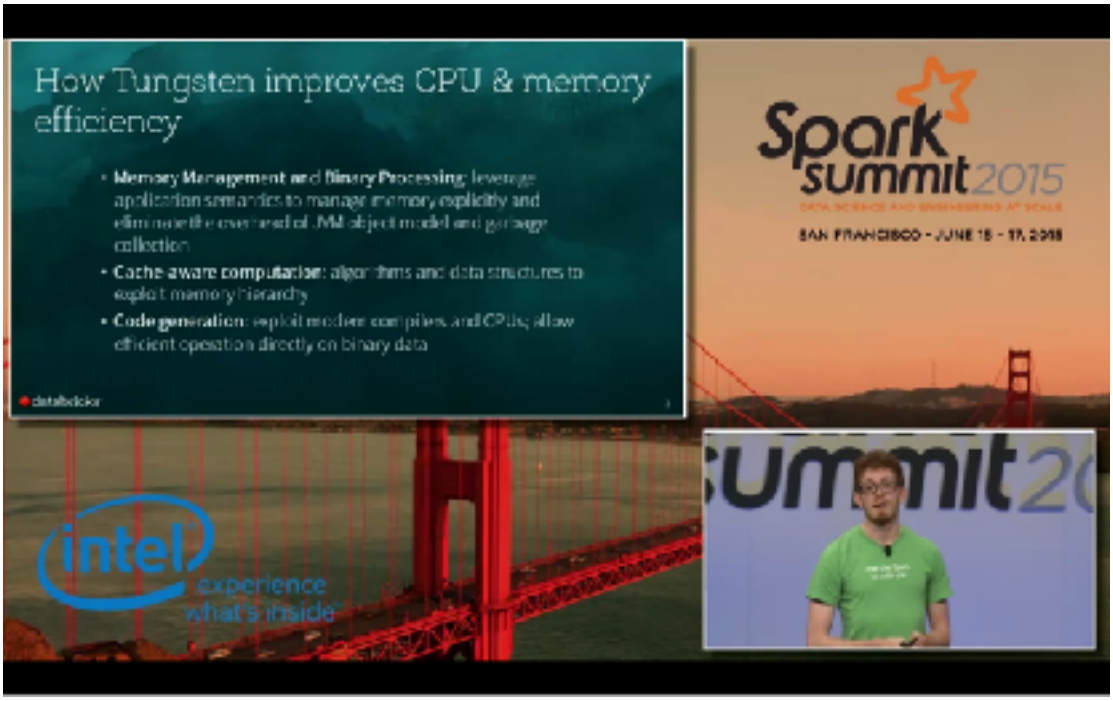
Kay Ousterhout*, Ryan Rasti*[1], Sylvia Ratnasamy*, Scott Shenker*[1], Byung-Gon Chun†
*UC Berkeley, [1]ICSI, †VMware, †Seoul National University

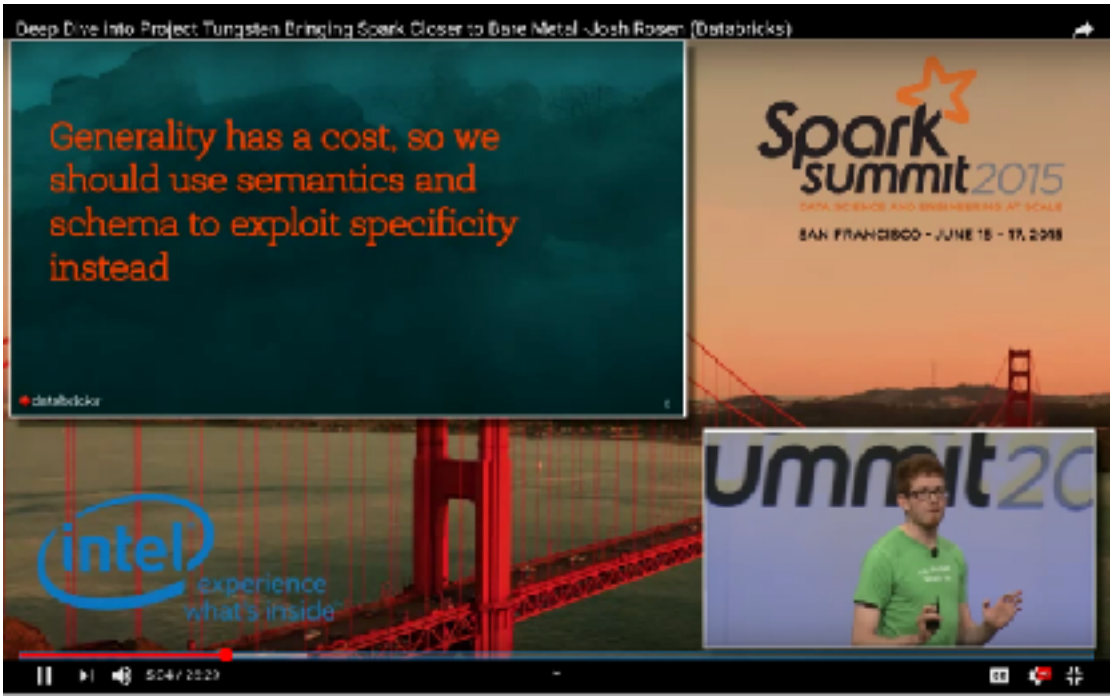
- "Network optimizations can only reduce job completion time by a median of at most 2%."
- "Optimizing or eliminating disk accesses can only reduce job completion time by a median of at most 19%."
- We've observed similar characteristics in many Databricks Cloud customer workloads.

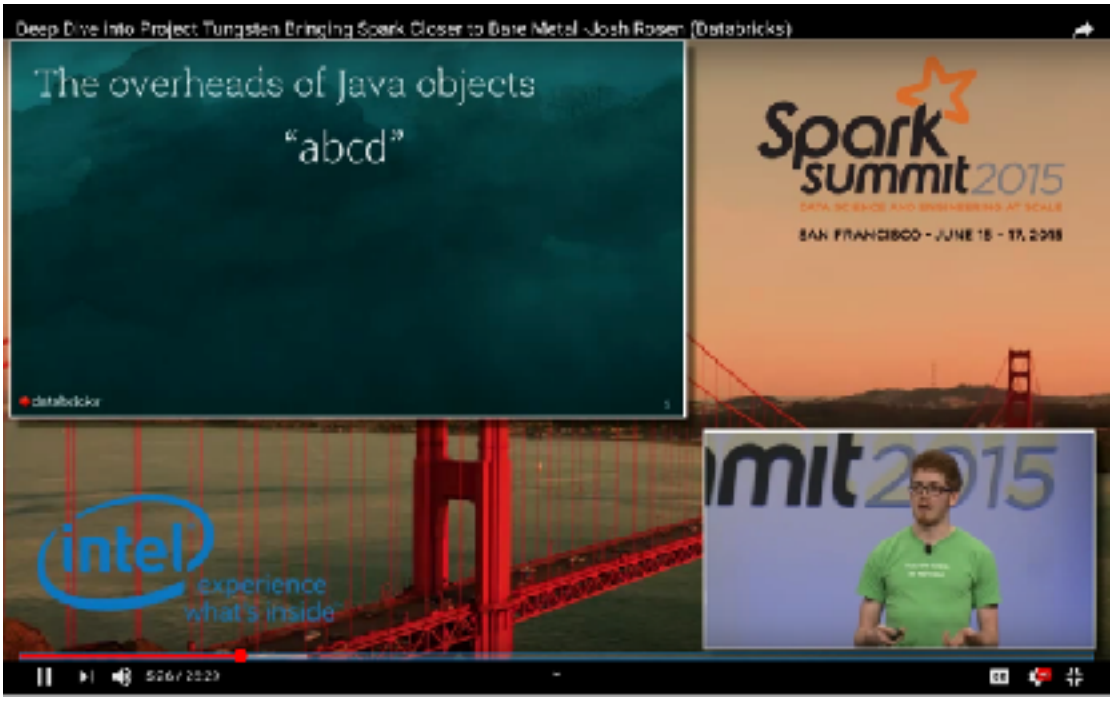

Why is CPU the new bottleneck?
- Hardware has improved:
- Spark's IO has been optimized:
Spark summit 2015
SAN FRANCISCO - JUNE 15 - 17, 2015
intel
experience what's inside

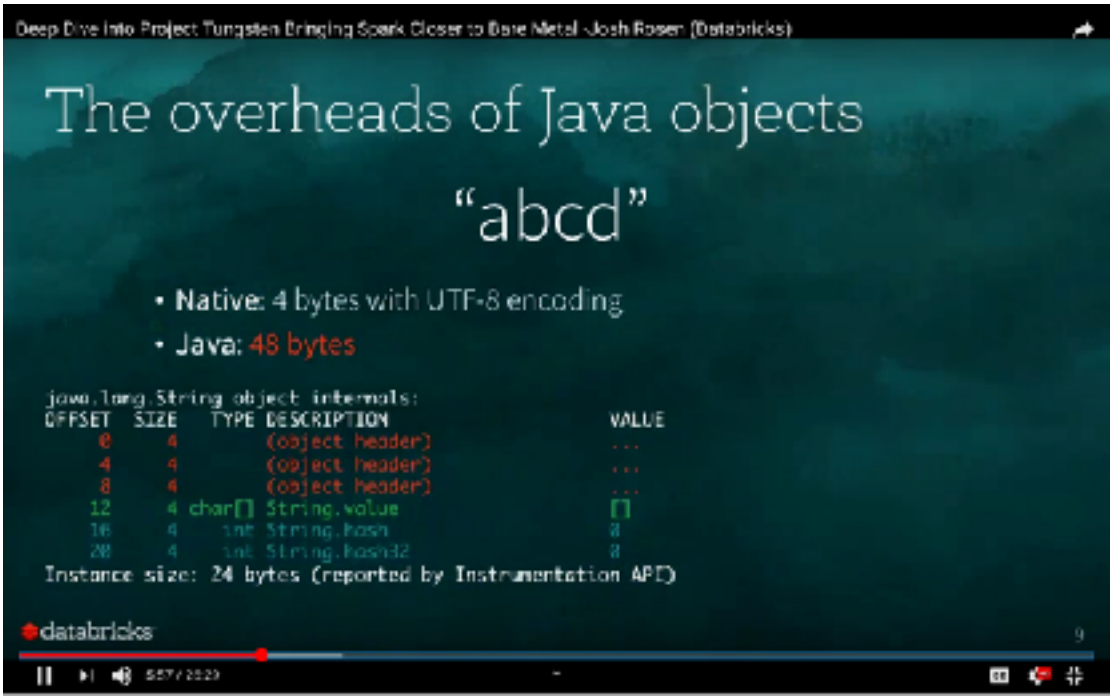

Why is CPU the new bottleneck?
- Hardware has improved:
- Spark's IO has been optimized:
- Data formats have improved:
Spark summit 2015
SAN FRANCISCO - JUNE 15 - 17, 2015
intel
experience what's inside

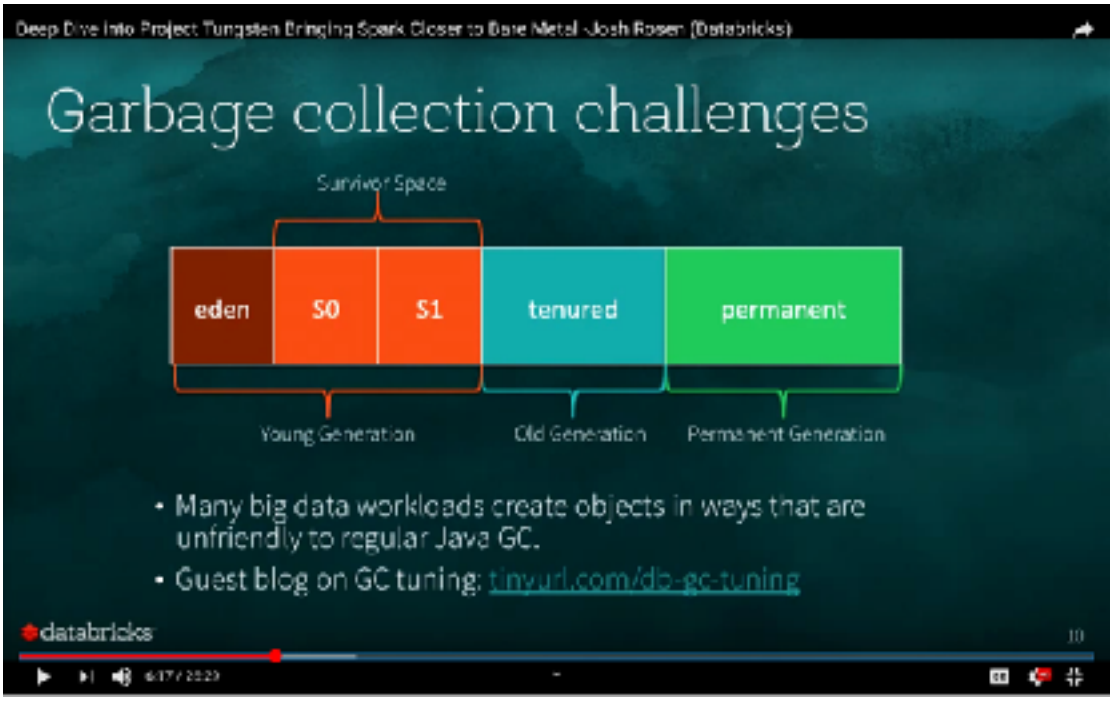

Why is CPU the new bottleneck?
• Hardware has improved:
- Increasingly large aggregate IO bandwidth, such as 10Gb ethernet in networks
- High bandwidth SSDs or striped HDD arrays for storage
• Spark's IO has been optimized:
- many workloads now avoid significant disk IO by pruning input data that is not needed in a given job
- new shuffle and network layer implementations
• Data formats have improved:
- Parquet, binary data formats
• Serialization and hashing are CPU-bound bottlenecks
databricks
Spark summit 2015
SAN FRANCISCO - JUNE 15 - 17, 2015
intel
experience what's inside

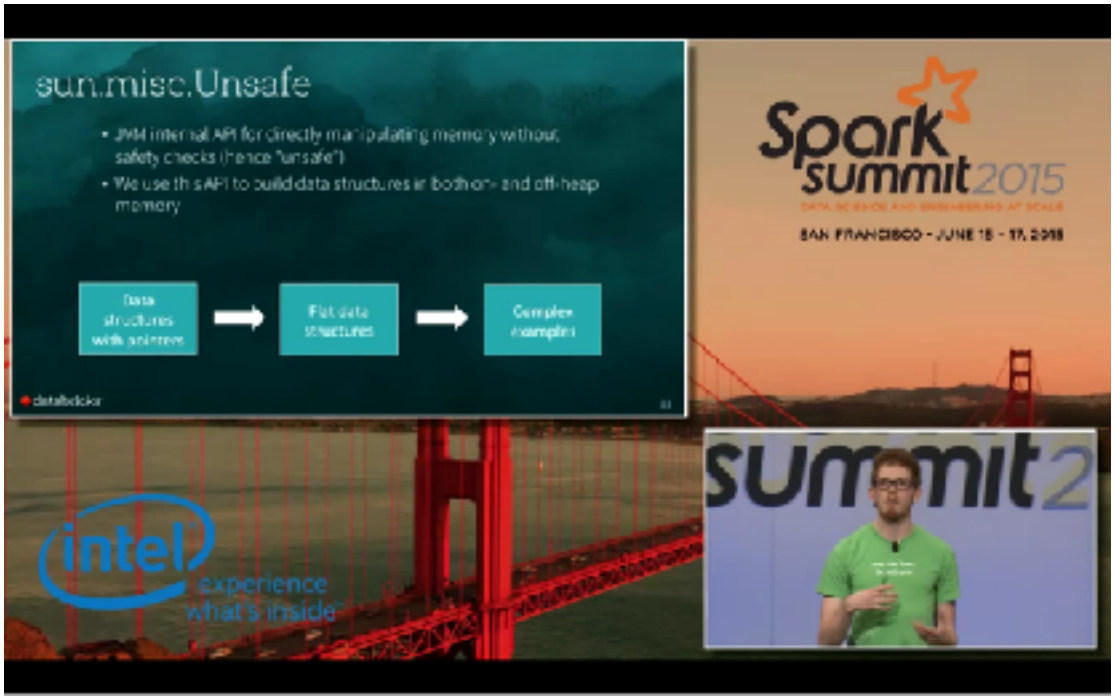

How Tungsten improves CPU & memory efficiency
• Memory Management and Binary Processing: leverage application semantics to manage memory explicitly and eliminate the overhead of JVM object model and garbage collection
• Cache-aware computation: algorithms and data structures to exploit memory hierarchy
• Code generation: exploit modern compilers and CPUs; allow efficient operation directly on binary data
databricks
Spark summit 2015
SAN FRANCISCO - JUNE 15 - 17, 2015
intel
experience what's inside

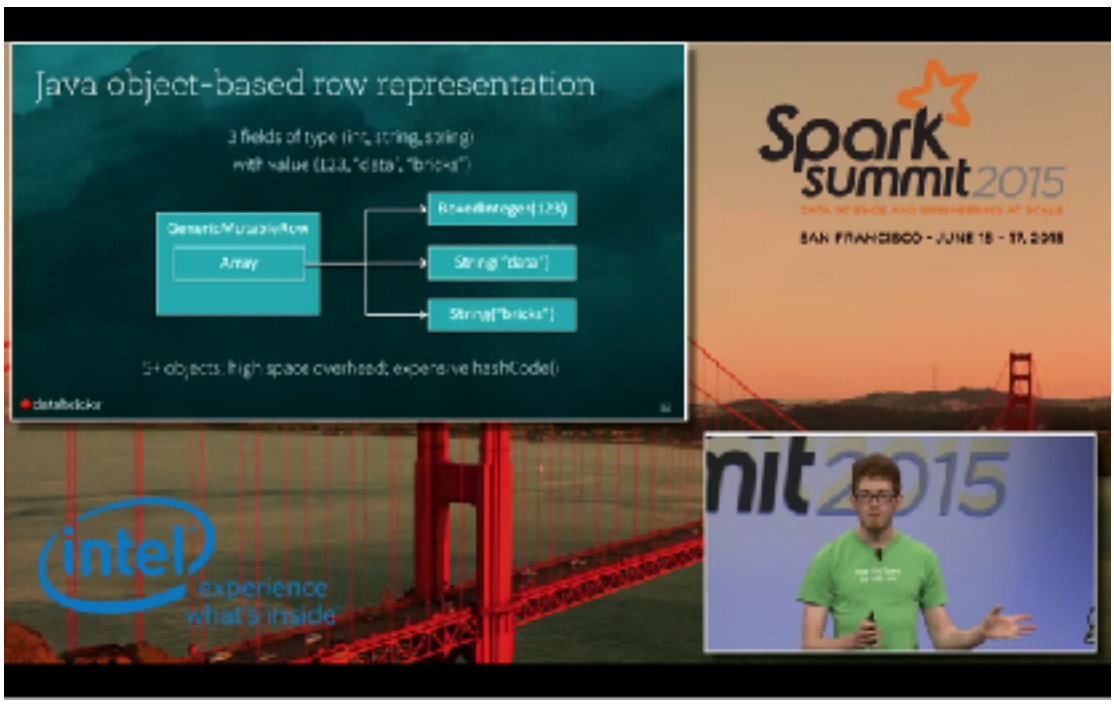

Deep Dive into Project Tungsten Bringing Spark Closer to Bare Metal -Josh Rosen (Databricks)
Generality has a cost, so we should use semantics and schema to exploit specificity instead
Spark summit 2015
intel
experience what's inside

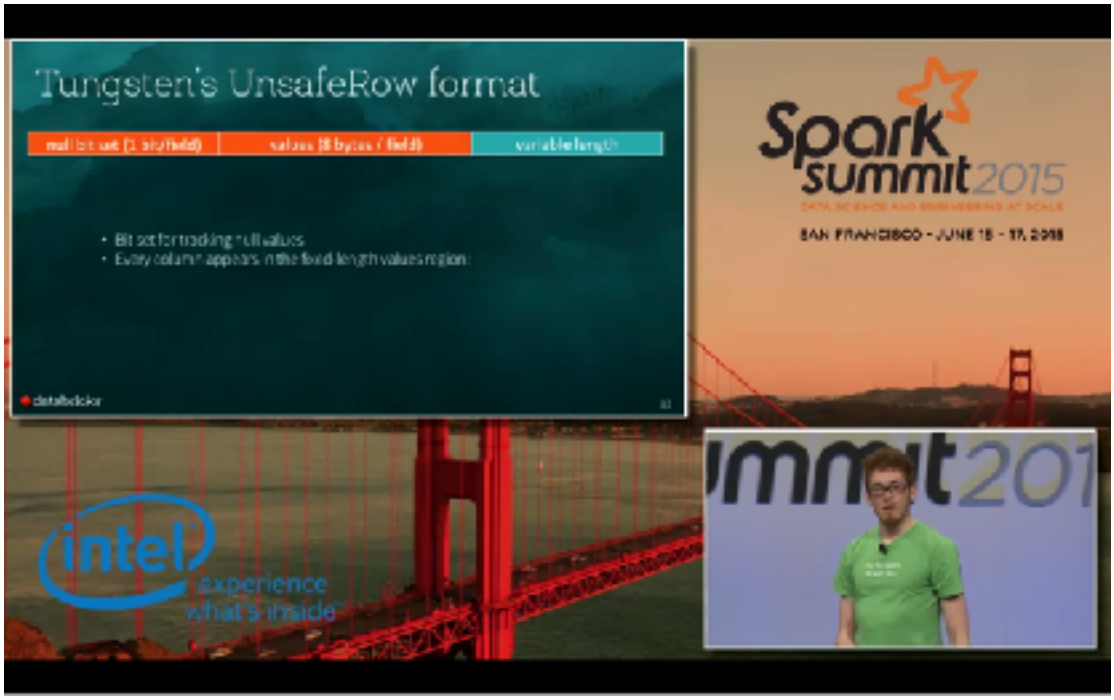

Deep Dive into Project Tungsten Bringing Spark Closer to Bare Metal -Josh Rosen (Databricks)
The overheads of Java objects
"abcd"
Spark summit 2015
intel
experience what's inside

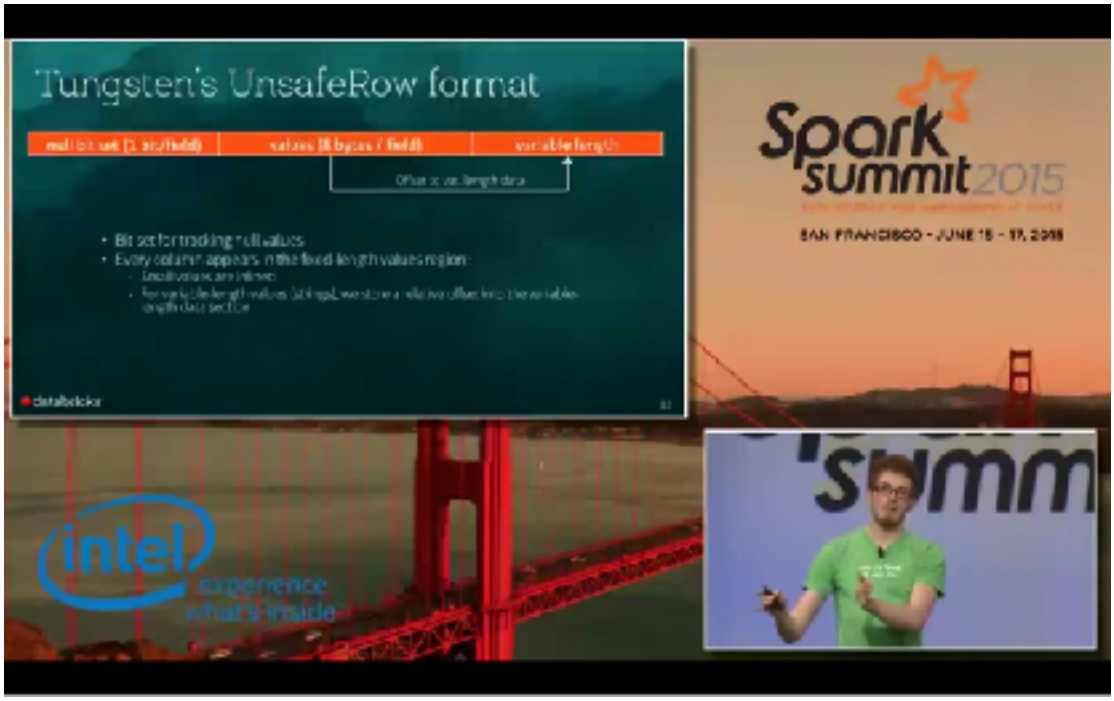

Deep Dive into Project Tungsten Bringing Spark Closer to Bare Metal -Josh Rosen (Databricks)
The overheads of Java objects
"abcd"
• Native: 4 bytes with UTF-8 encoding
• Java: 48 bytes
java.lang.String object internals:
OFFSET SIZE TYPE DESCRIPTION VALUE
0 4 (object header) ...
4 4 (object header) ...
8 4 (object header) ...
12 4 char[] String.value []
16 4 int String.hash 0
20 4 int String.hash32 0
Instance size: 24 bytes (reported by Instrumentation API)
databricks
9

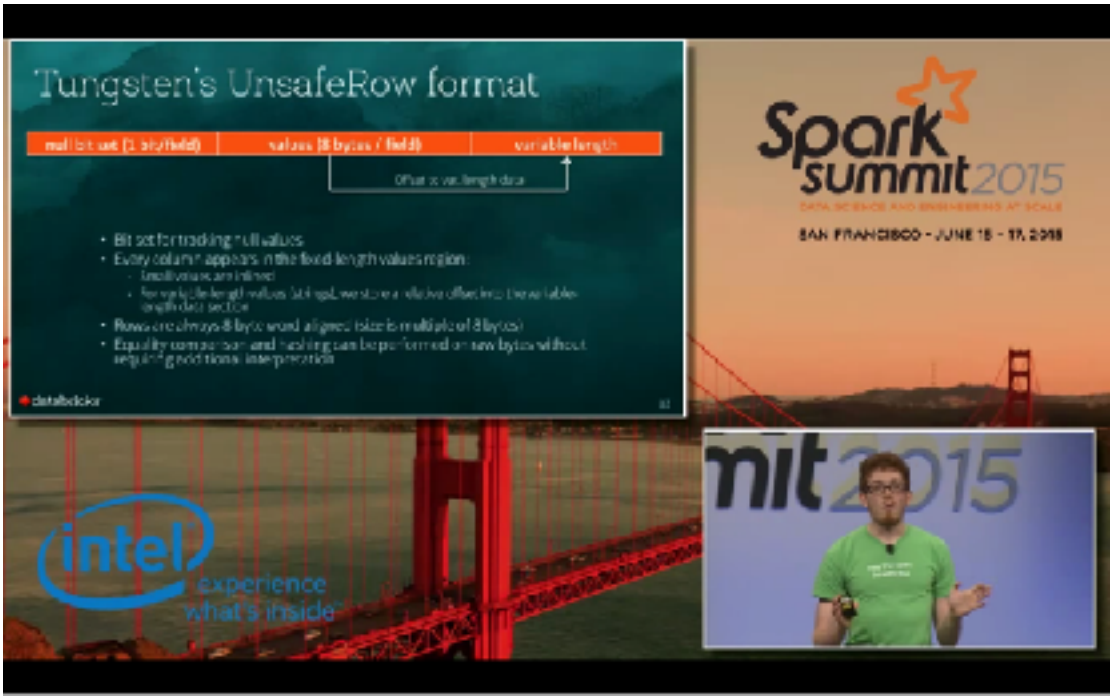

Deep Dive into Project Tungsten Bringing Spark Closer to Bare Metal -Josh Rosen (Databricks)
Garbage collection challenges
Survivor Space
eden
S0
S1
tenured
permanent
Young Generation
Old Generation
Permanent Generation
• Many big data workloads create objects in ways that are unfriendly to regular Java GC.
• Guest blog on GC tuning: tinyurl.com/db-gc-tuning
databricks
10

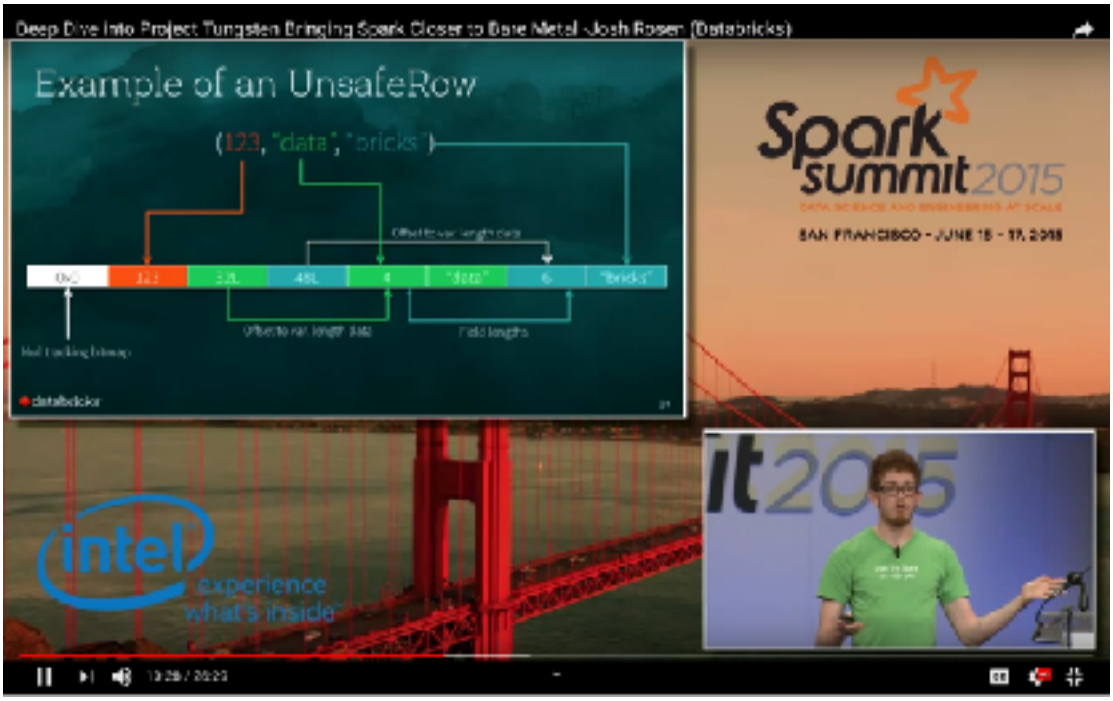

sun.misc.Unsafe
• JVM internal API for directly manipulating memory without safety checks (hence "unsafe")
• We use this API to build data structures in both on- and off-heap memory
Data structures with pointers
Flat data structures
Compiler compiles
Spark summit 2015
DATA SCIENCE AND ENGINEERING AT SCALE
SAN FRANCISCO - JUNE 15 - 17, 2015
intel
experience what's inside

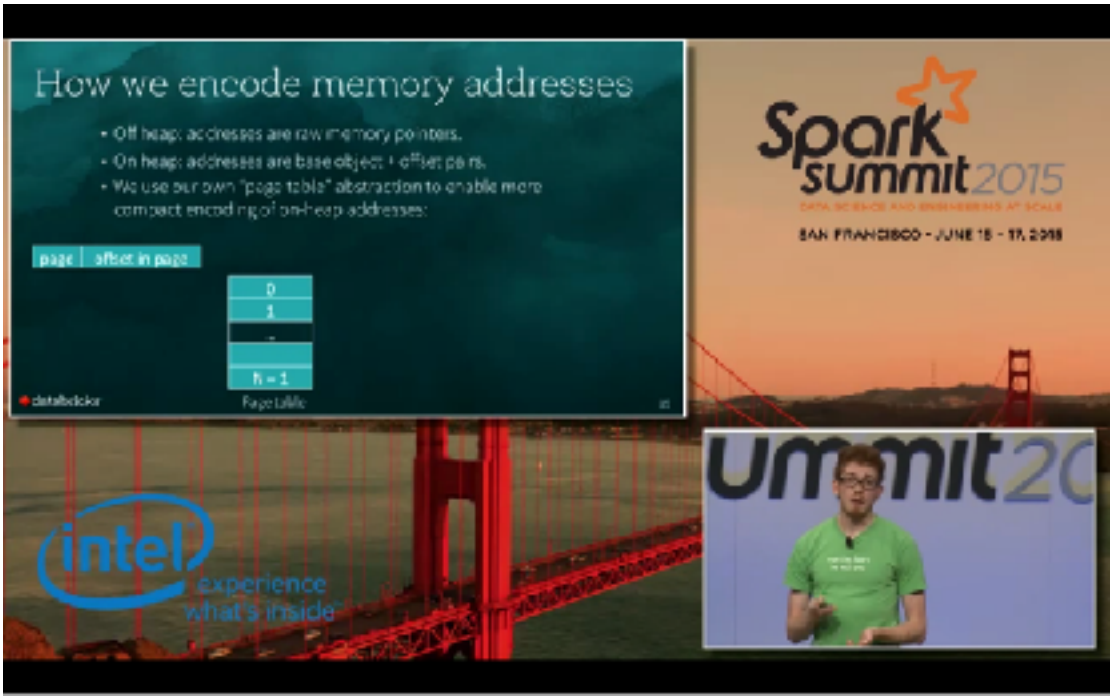

Java object-based row representation
3 fields of type (int, string, string)
with value (123, "data", "bricks")
Array
String["data"]
String["bricks"]
54 objects, high space overhead, expensive hashCode()
Spark summit 2015
DATA SCIENCE AND ENGINEERING AT SCALE
SAN FRANCISCO - JUNE 15 - 17, 2015
intel
experience what's inside

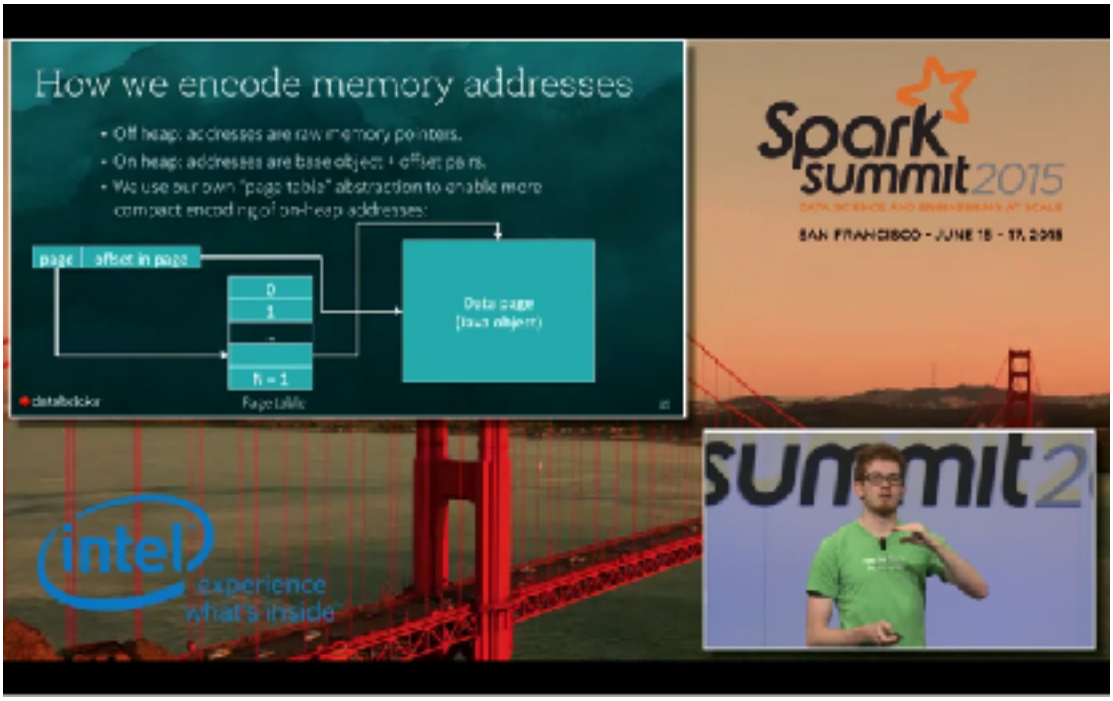

Tungsten's UnsafeRow format
null bit set (1 bit/field)
values (8 bytes / field)
variable length
• Bit set for tracking null values
• Every column appears in the fixed length values region
databricks
Spark summit 2015
SAN FRANCISCO - JUNE 15 - 17, 2015
intel
experience
what's inside

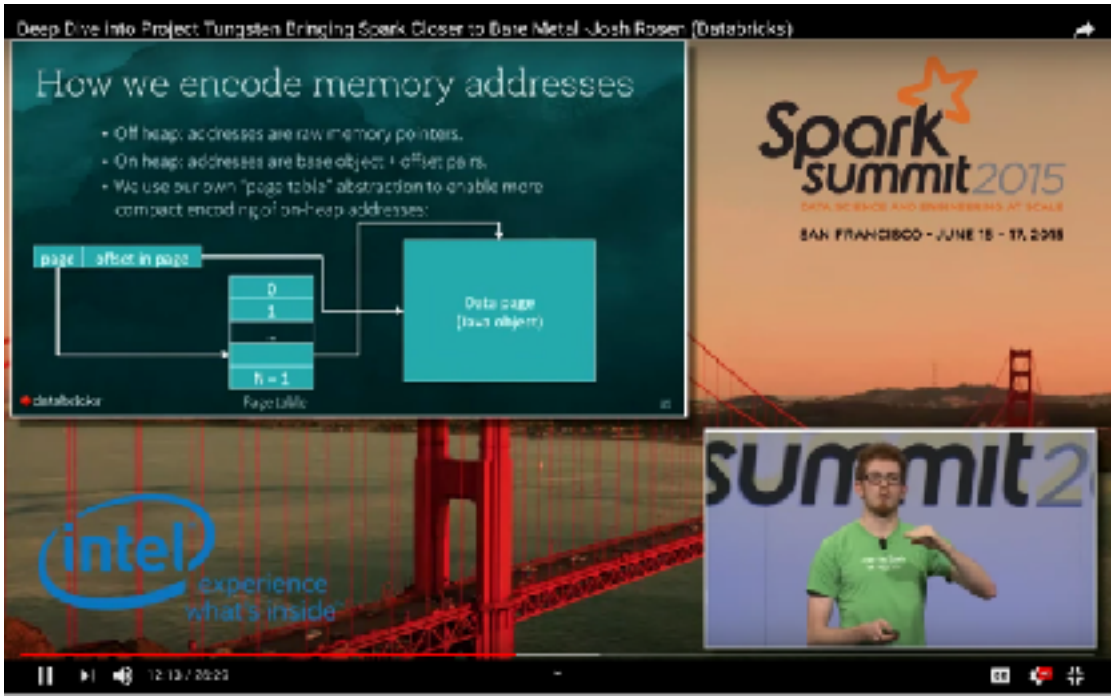

Tungsten's UnsafeRow format
null bit set (1 bit/field)
values (8 bytes / field)
variable length
Offset to variable length data
• Bit set for tracking null values
• Every column appears in the fixed length values region
databricks
Spark summit 2015
SAN FRANCISCO - JUNE 15 - 17, 2015
intel
experience
what's inside

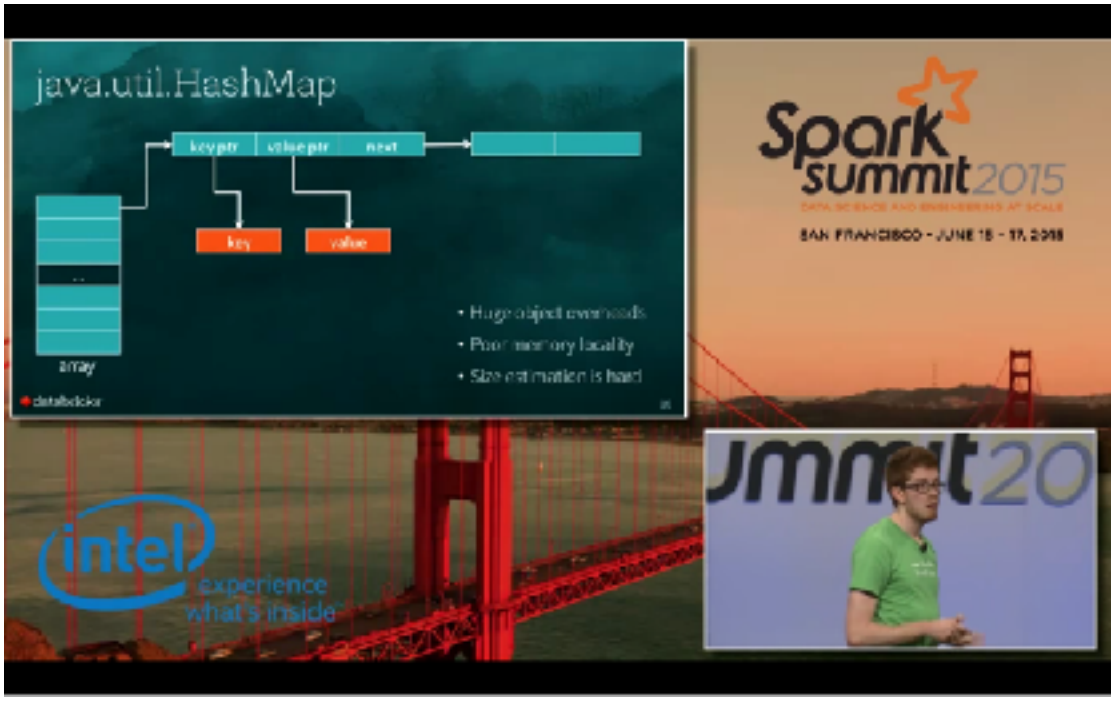

Tungsten's UnsafeRow format
Spark summit 2015
intel
experience what's inside

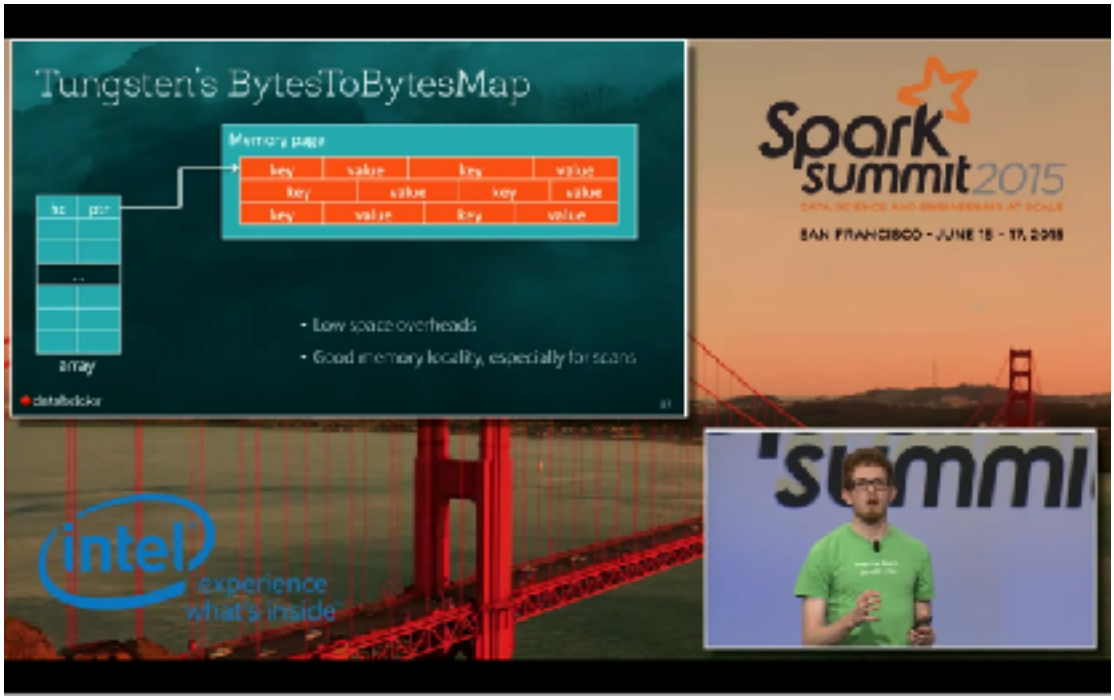

Deep Dive into Project Tungsten Bringing Spark Closer to Bare Metal - Josh Rosen (Databricks)
Example of an UnsafeRow
(123, "data", "bricks")
Spark summit 2015
intel
experience what's inside

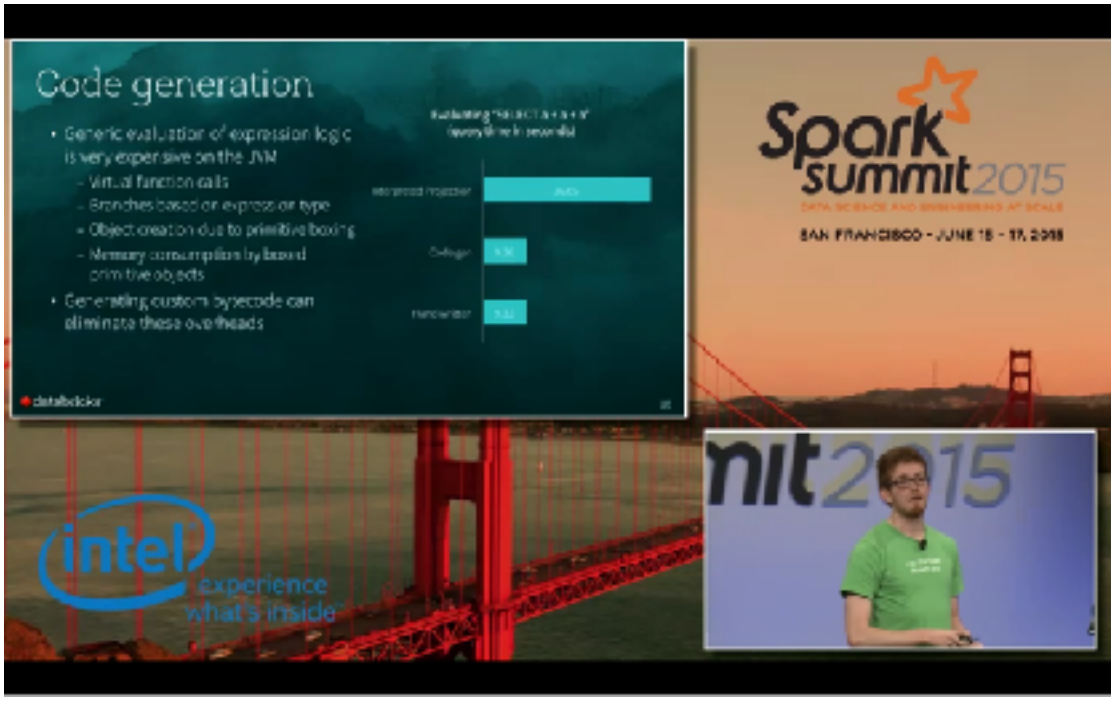

How we encode memory addresses
• Off heap: addresses are raw memory pointers.
• On heap: addresses are base object + offset pairs.
• We use our own "page table" abstraction to enable more compact encoding of on-heap addresses:
page
offset in page
0
1
...
n - 1
Page table
databricks
Spark summit 2015
SAN FRANCISCO - JUNE 15 - 17, 2015
intel
experience what's inside


How we encode memory addresses
• Off heap: addresses are raw memory pointers.
• On heap: addresses are base object + offset pairs.
• We use our own "page table" abstraction to enable more compact encoding of on-heap addresses:
page
offset in page
0
1
...
n - 1
Page table
Data page
databricks
Spark summit 2015
SAN FRANCISCO - JUNE 15 - 17, 2015
intel
experience what's inside


Deep Dive into Project Tungsten Bringing Spark Closer to Bare Metal -Josh Rosen (Databricks)
How we encode memory addresses
• Off heap: addresses are raw memory pointers.
• On heap: addresses are base object + offset pairs.
• We use our own "page table" abstraction to enable more compact encoding of on-heap addresses:
page
offset in page
0
1
n - 1
Page table
Data page
(Java object)
databricks
Spark summit 2015
SAN FRANCISCO - JUNE 15 - 17, 2015
intel
experience what's inside

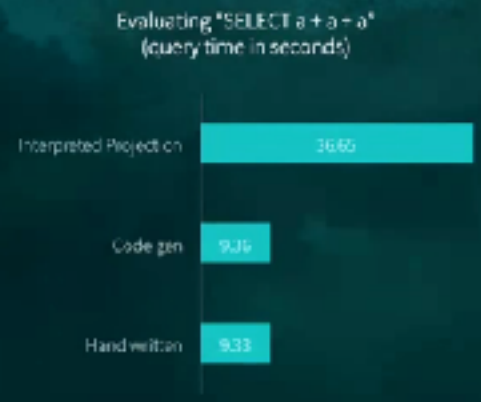

java.util.HashMap
key ptr
value ptr
next
key
value
array
• Huge object overheads
• Poor memory locality
• Size estimation is hard
databricks
Spark summit 2015
SAN FRANCISCO - JUNE 15 - 17, 2015
intel
experience what's inside


Tungsten's BytesToBytesMap
Memory page
key
value
array
• Low space overheads
• Good memory locality, especially for scans
databricks
Spark summit 2015
SAN FRANCISCO - JUNE 15 - 17, 2015
intel
experience what's inside


Code generation
• Generic evaluation of expression logic is very expensive on the JVM
– Virtual function calls
– Branches based on expression type
– Object creation due to primitive boxing
– Memory consumption by boxed primitive objects
• Generating custom bytecode can eliminate these overheads
databricks
Spark summit 2015
SAN FRANCISCO - JUNE 15 - 17, 2015
intel
experience what's inside

Code generation

- Generic evaluation of expression logic is very expensive on the JVM
  - Virtual function calls
  - Branches based on expression type
  - Object creation due to primitive boxing
  - Memory consumption by boxed primitive objects
- Generating custom bytecode can eliminate these overheads

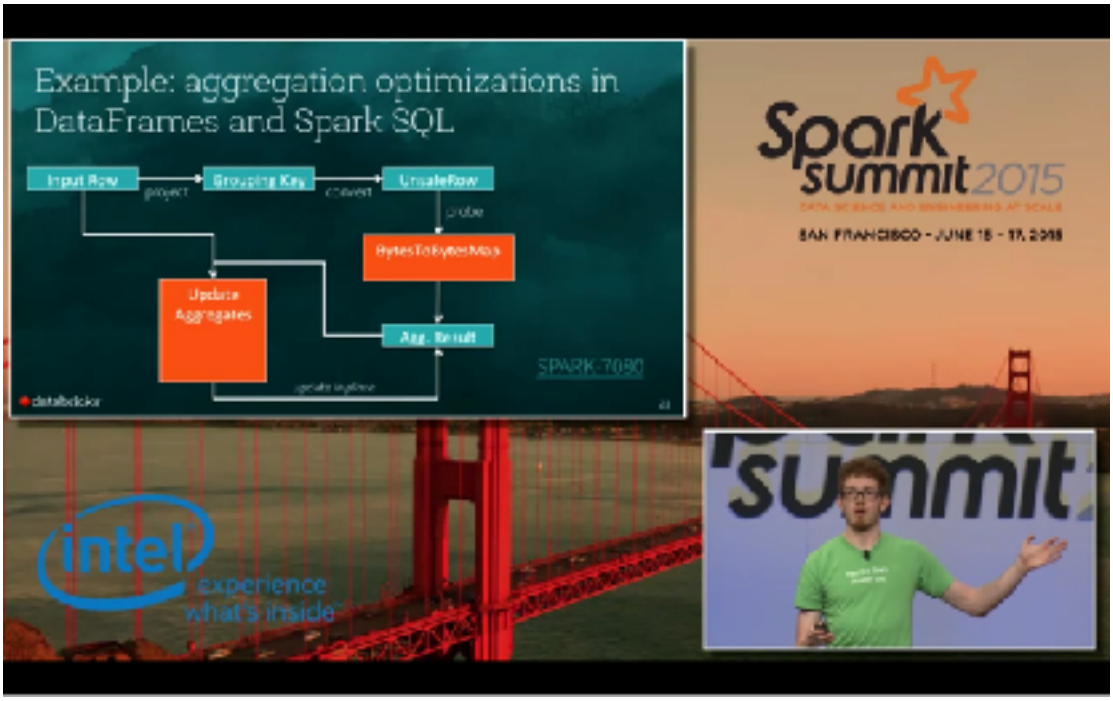


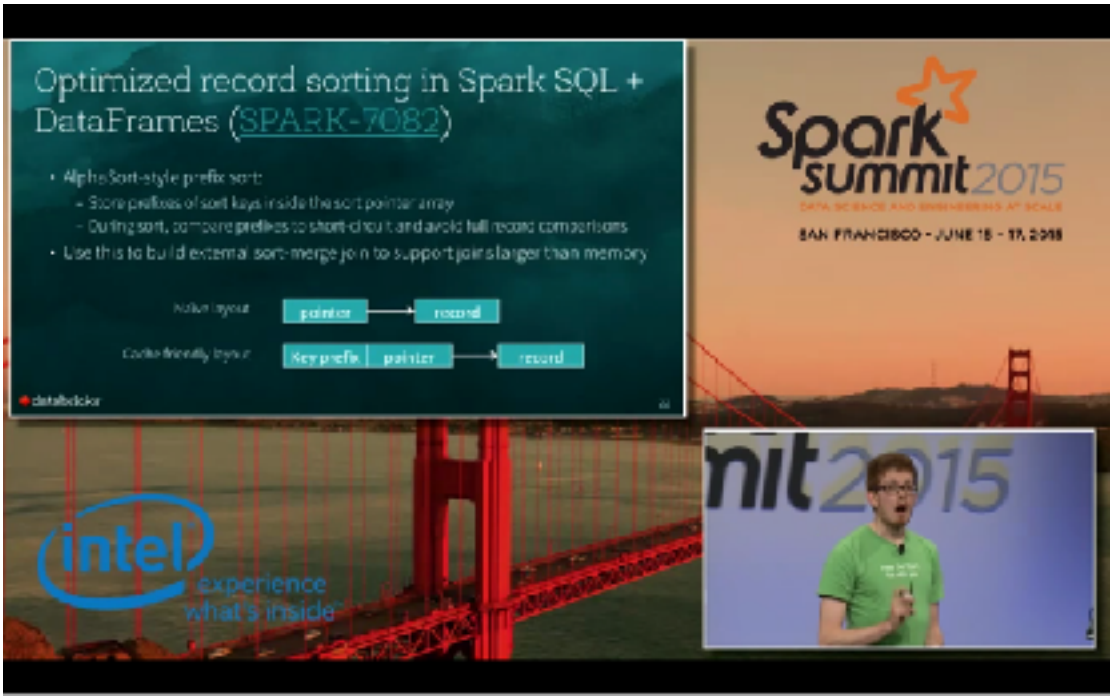
Deep Dive into Project Tungsten Bringing Spark Closer to Bare Metal - Josh Rosen (Databricks)

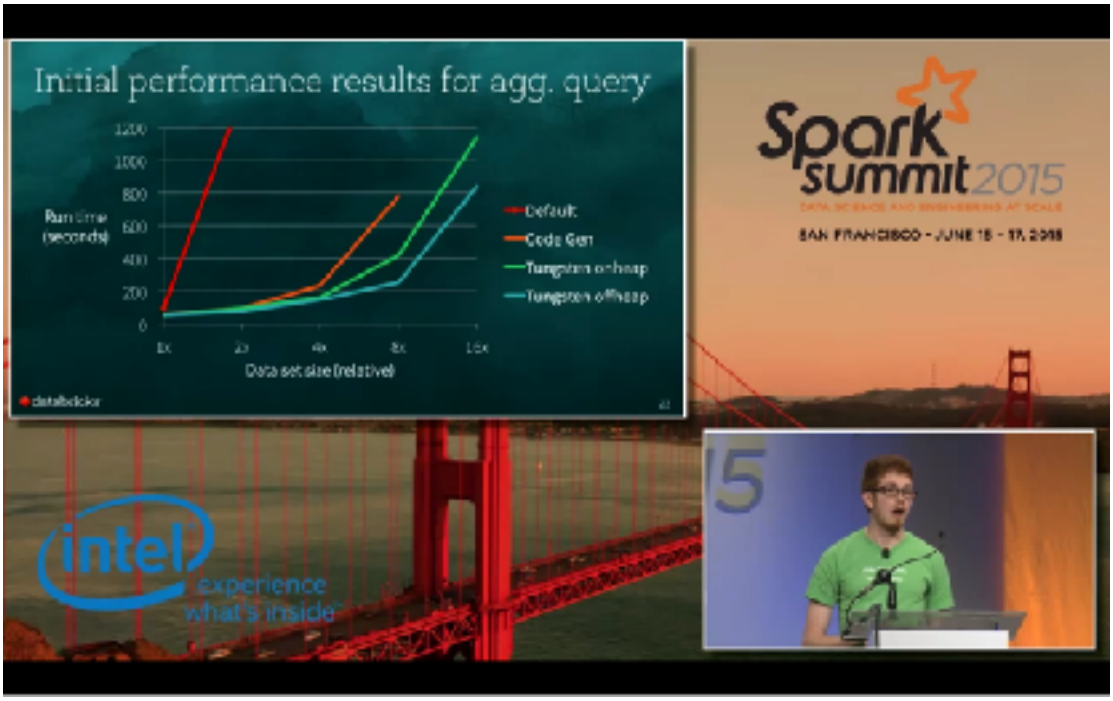
Code generation

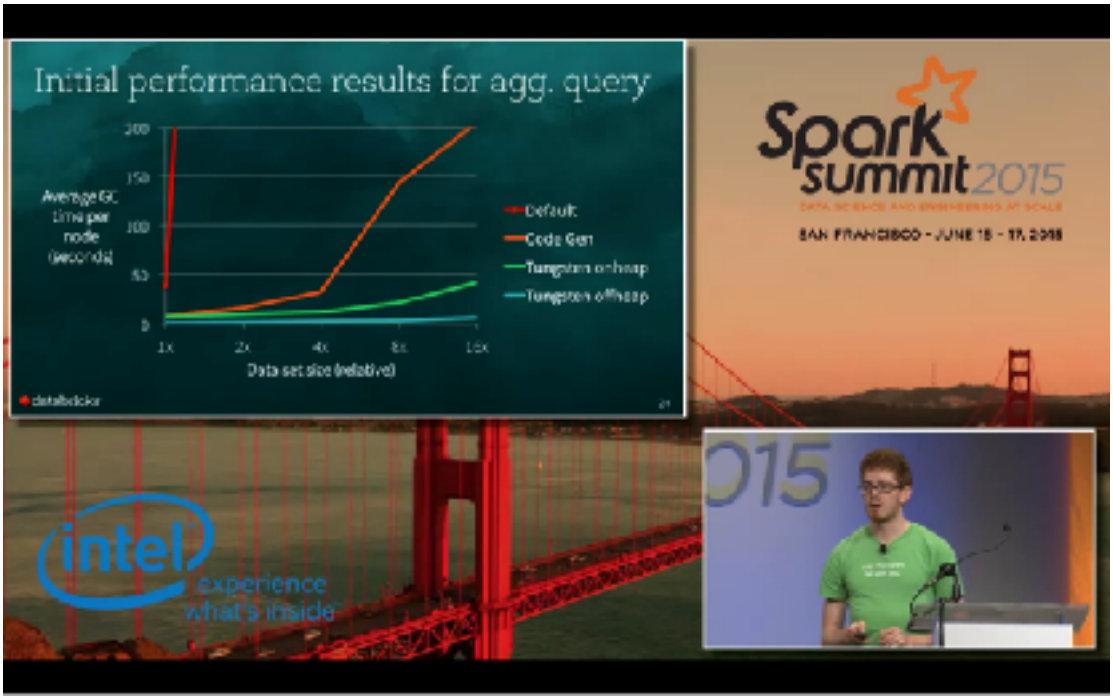
- Project Tungsten uses the Janino compiler to reduce code generation time.
- Spark 1.5 will greatly expand the number of expressions that support code generation:
  - SPARK-8159

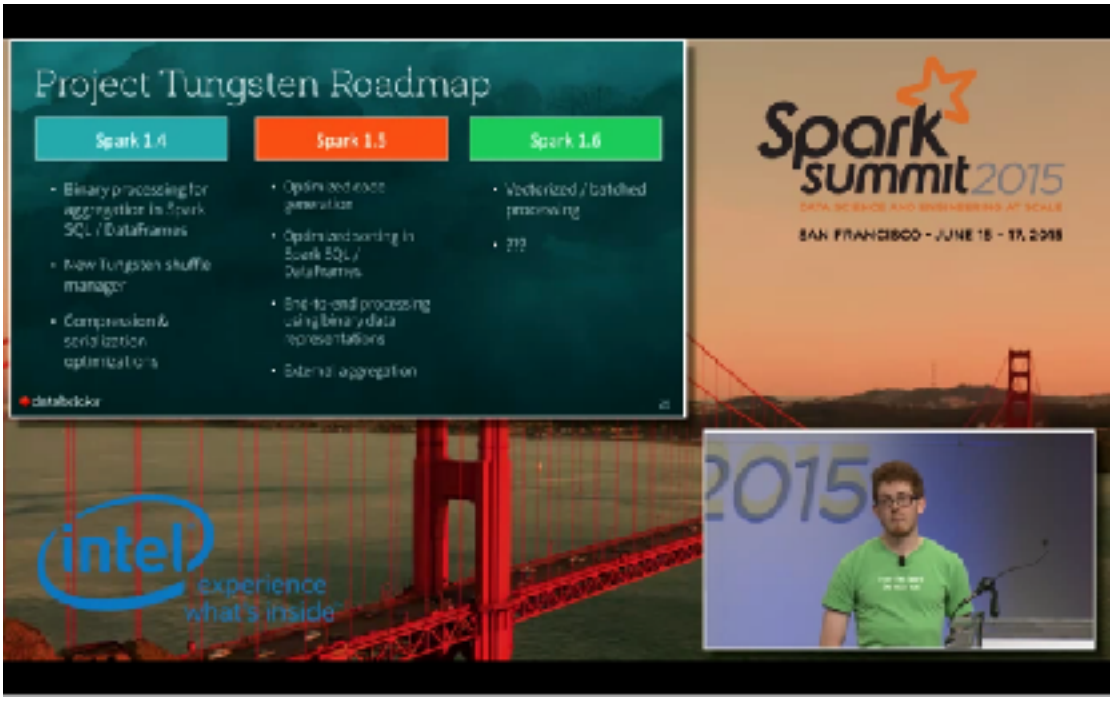

Example: aggregation optimizations in DataFrames and Spark SQL
df.groupBy("department").agg(max("age"), sum("expense"))
databricks
Spark summit 2015
SAN FRANCISCO - JUNE 15 - 17, 2015
intel
experience what's inside

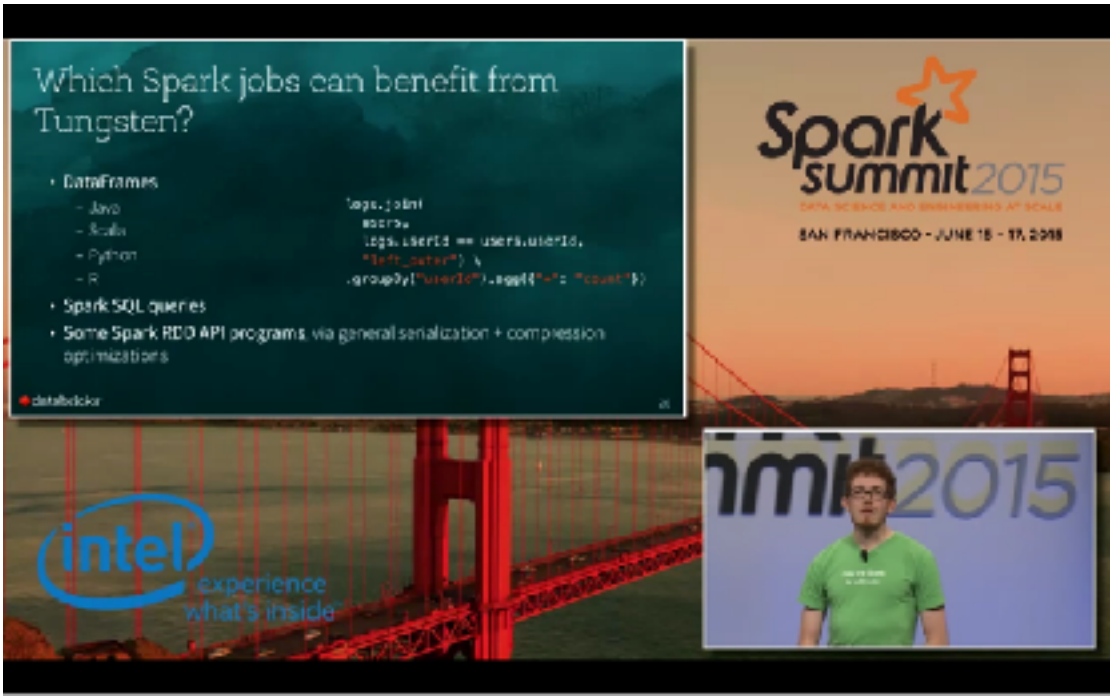

Example: aggregation optimizations in DataFrames and Spark SQL
Input Row
project
Grouping Key
convert
UnsafeRow
probe
Update Aggregates
Agg. Result
SPARK-7080
databricks
Spark summit 2015
SAN FRANCISCO - JUNE 15 - 17, 2015
intel
experience what's inside


Optimized record sorting in Spark SQL + DataFrames (SPARK-7082)
• AlphaSort-style prefix sort:
– Store prefixes of sort keys inside the sort pointer array
– During sort, compare prefixes to short-circuit and avoid full record comparisons
• Use this to build external sort-merge join to support joins larger than memory
pointer
record
Key prefix
pointer
record
databricks
Spark summit 2015
SAN FRANCISCO - JUNE 15 - 17, 2015
intel
experience what's inside


Initial performance results for agg. query
1200
1000
800
600
400
200
0
Run time (seconds)
1x
2x
4x
8x
16x
Data set size (relative)
Default
Code Gen
databricks
Spark summit 2015
SAN FRANCISCO - JUNE 15 - 17, 2015
intel
experience what's inside


Initial performance results for agg. query
Average GC time per node (seconds)
200
150
100
50
0
1x
2x
4x
8x
16x
Data set size (relative)
Default
Code Gen
Tungsten on-heap
Tungsten off-heap
databricks
Spark summit 2015
SAN FRANCISCO - JUNE 15 - 17, 2015
intel
experience what's inside


Project Tungsten Roadmap
Spark 1.4
Spark 1.5
Spark 1.6
Binary processing for aggregation in Spark SQL / DataFrames
New Tungsten shuffle manager
Compression & serialization optimizations
Optimized code generation
Optimized sorting in Spark SQL / DataFrames
End-to-end processing using binary data representations
External aggregation
Vectorized / batched processing
...
databricks
Spark summit 2015
SAN FRANCISCO - JUNE 15 - 17, 2015
intel
experience what's inside



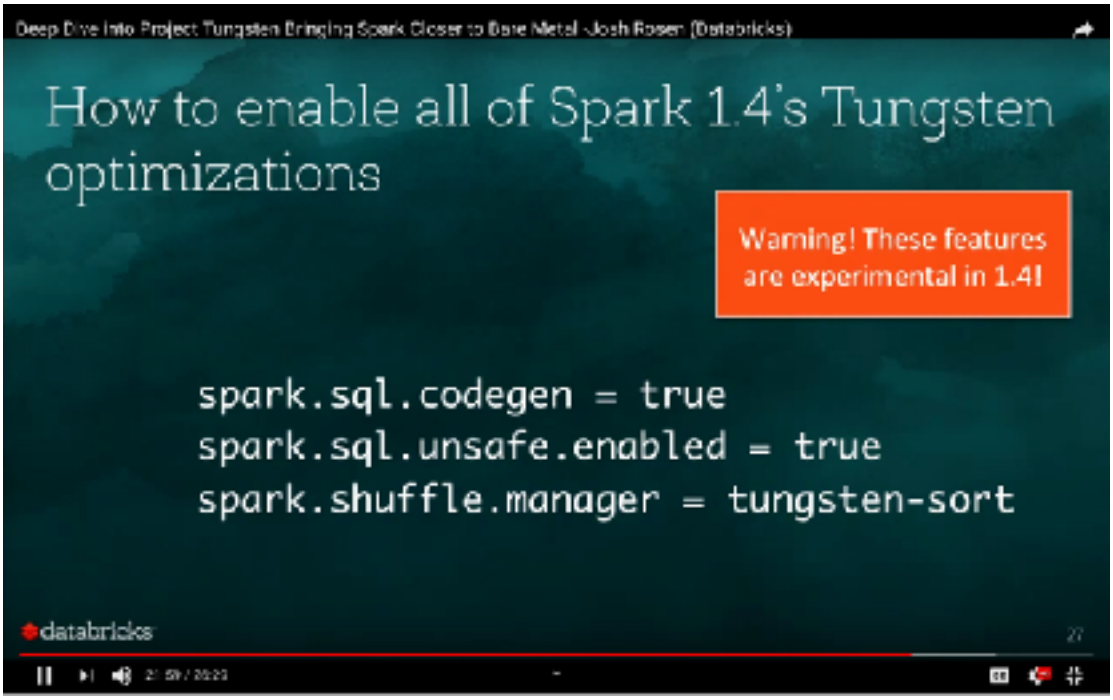
Which Spark jobs can benefit from Tungsten?

- DataFrames
  - Java
  - Scala
  - Python
  - R

```
logs.join(
  users,
  logs.userId == users.userId,
  "left_outer") \
.groupBy("userId").agg({"*": "count"})
```

- Spark SQL queries
- Some Spark RDD API programs, via general serialization + compression optimizations

databricks



https://community.hortonworks.com/articles/72502/what-is-tungsten-for-apache-spark.html

Refernces:
https://databricks.com/blog/2015/04/28/project-tungsten-bringing-spark-closer-to-bare-metal.html

SparkContext(sc)

It resides in Driver program to communicate between the Spark Driver and cluster to get connect with ClusterManager(Mesos, Yarn)

When spark-shell launch, SparkContext is created by default as “SC”

```
$spark-shell --master yarn

Spark context Web UI available at http://play2:4040
Spark context available as 'sc' (master = yarn, app id = application_1

version 2.4.0
```

SC:

- The driver program use the SparkContext to connect and communicate with the cluster and it helps in executing and coordinating the Spark job with the resource managers like YARN or Mesos.
- Using SparkContext you can actually get access to other contexts like SQLContext and HiveContext.

- Using SparkContext we can set configuration parameters to the Spark job.
- If you are in spark-shell, a SparkContext is already available for you and is assigned to the variable sc.
- If you don't have a SparkContext already, you can create one by first creating a SparkConf first.

```
//set up the spark configuration
val sparkConf = new SparkConf().setAppName("hirw").setMaster("yarn")
//get SparkContext using the SparkConf
val sc = new SparkContext(sparkConf)
```

SQL Context (sqlContext)

SQLContext is your gateway to Spark-SQL. Here is how you create a SQL-Context using the SparkContext.

```
// sc is an existing SparkContext.
val sqlContext = new org.apache.spark.sql.SQLContext(sc)
```

Hive Context(hiveContext)

- HiveContext is your gateway to Hive.
- HiveContext has all the functionalities of a SQLContext.
- In fact, if you look at the API documentation you can see that HiveContext extends SQLContext, meaning, it has support the functionalities that SQLContext support plus more (Hive specific functionalities)

```
public class HiveContext extends SQLContext implements Logging

//Way to create HiveContext
// sc is an existing SparkContext
val sqlContext = new org.apache.spark.sql.hive.HiveContext(sc)
```

Spark Session(spark)

SparkSession was introduced in Spark 2.0 to make it easy for the developers so we don't have worry about different contexts and to streamline the access to different contexts. By having access to SparkSession, we automatically have access to the SparkContext
Or

A SparkSession can be created using a builder pattern. The builder will automatically reuse an existing SparkContext if one exists; and create a SparkContext if it does not exist. Configuration options set in the builder are automatically propagated over to Spark and Hadoop during I/O.

```
// A SparkSession can be created using a builder pattern
import org.apache.spark.sql.SparkSession
val sparkSession = SparkSession.builder
```

```
  .master("local")
  .appName("my-spark-app")
  .config("spark.some.config.option", "config-value")
  .getOrCreate()
```

In Glance:

Explanation from spark source code under branch-2.1

SparkContext (SC)

- Main entry point for Spark functionality.
- A SparkContext represents the connection to a Spark cluster, and can be used to create RDDs, accumulators and broadcast variables on that cluster.
- Only one SparkContext may be active per JVM.
- You must stop() the active SparkContext before creating a new one.
- This limitation may eventually be removed; see SPARK-2243 for more details.

JavaSparkContext:

- A Java-friendly version of [[org.apache.spark.SparkContext]] that returns [[org.apache.spark.api.java.JavaRDD]]s and works with Java collections instead of Scala ones.
- Only one SparkContext may be active per JVM.
- You must stop() the active SparkContext before creating a new one.
- This limitation may eventually be removed; see SPARK-2243 for more details.

SQLContext (sqlContext)

- The entry point for working with structured data (rows and columns) in Spark 1.x.
- As of Spark 2.0, this is replaced by [[SparkSession]].
- However, we are keeping the class here for backward compatibility.

SparkSession (spark)

The entry point to programming Spark with the **Dataset** and **DataFrame** API.

NOTE:

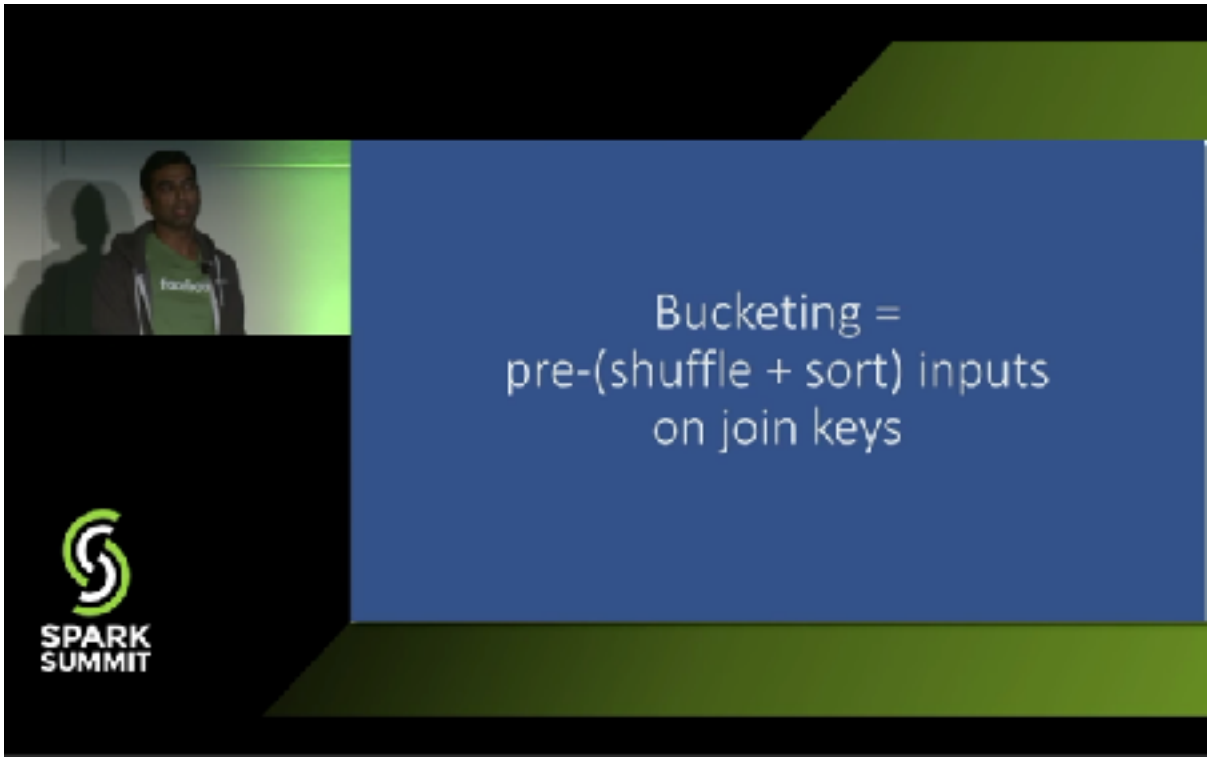
This way we can create SparkSession for Sql operation on Dataframe

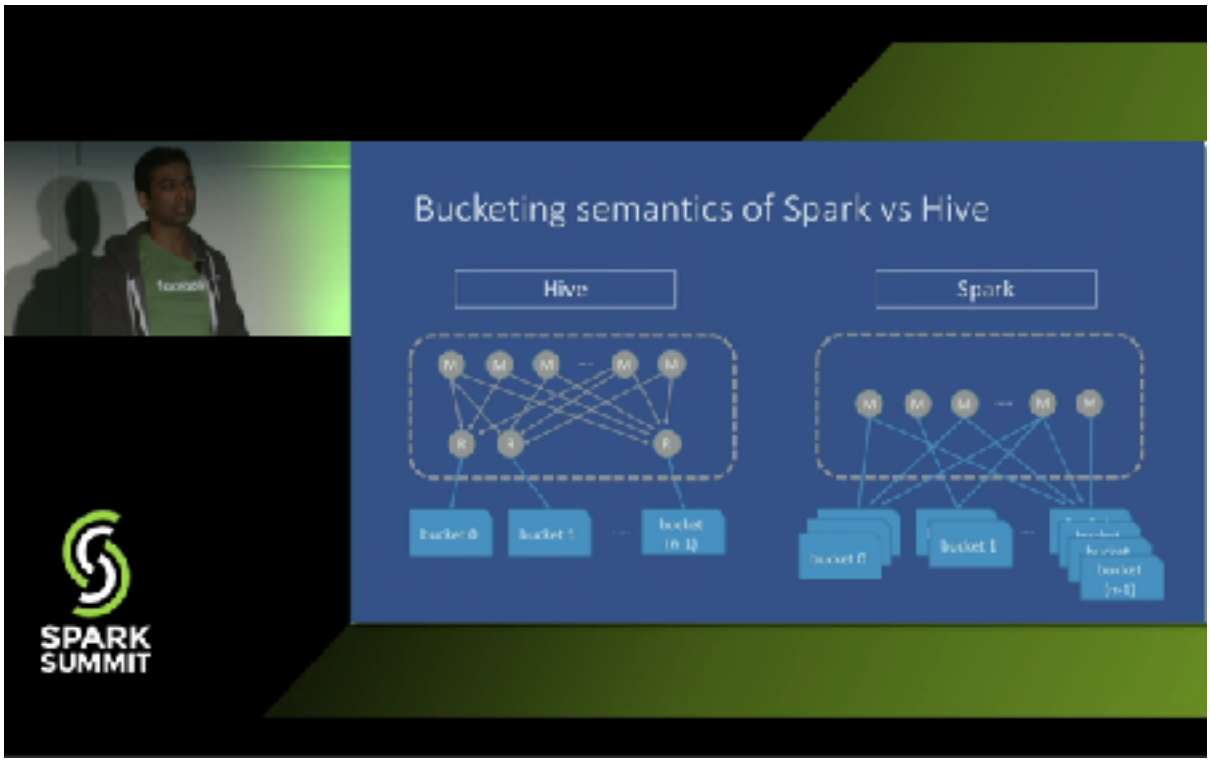
val sparksession=SparkSession.builder().getOrCreate();

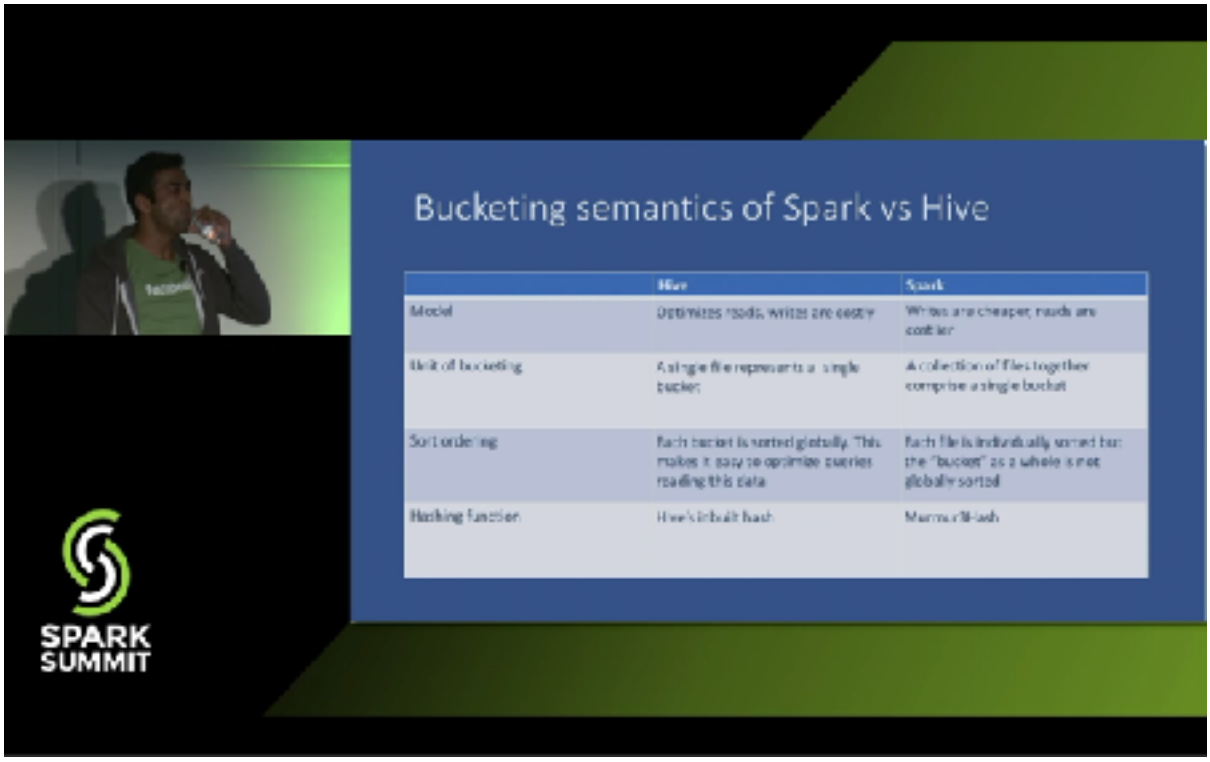
Second way is to create SparkSession for Sql operation on Dataframe as well as Hive Operation.

val sparkSession=SparkSession.builder().enableHiveSupport().getOrCreate()

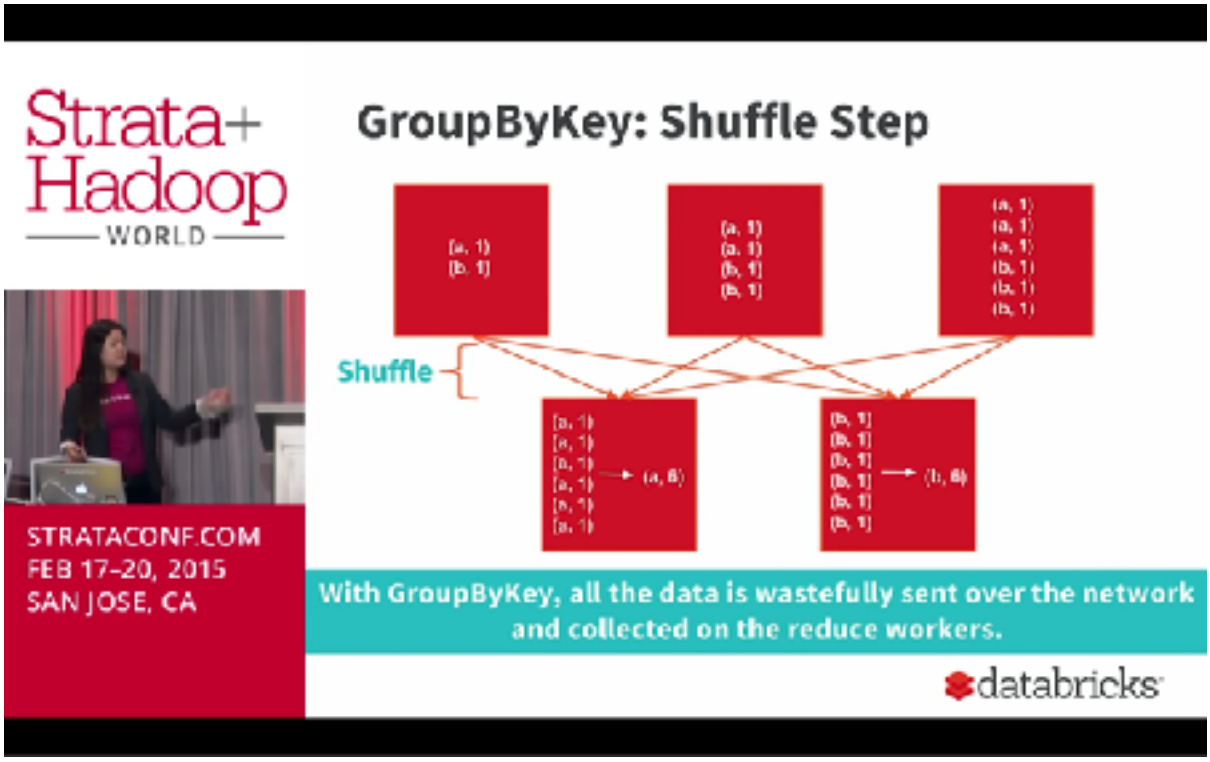
Bucketing Hive vs Spark

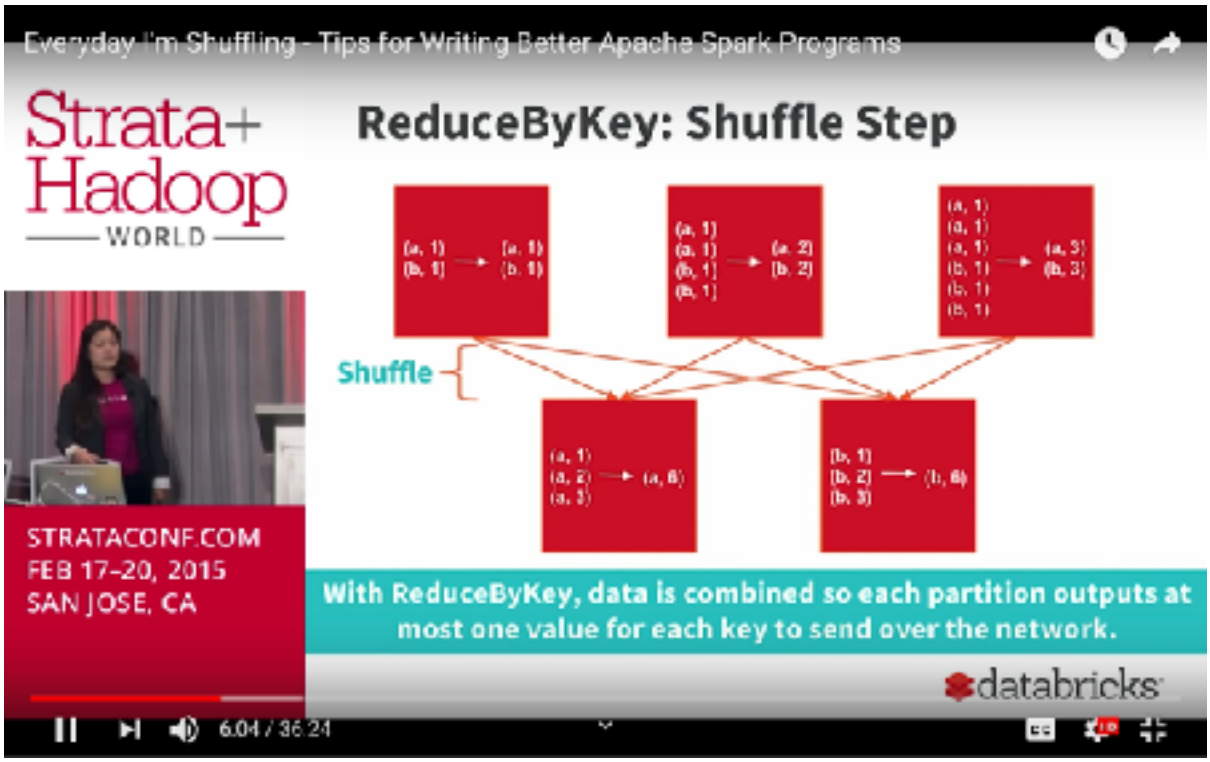

Bucketing =
pre-(shuffle + sort) inputs
on join keys
SPARK
SUMMIT


Bucketing semantics of Spark vs Hive
Hive
Spark
SPARK
SUMMIT



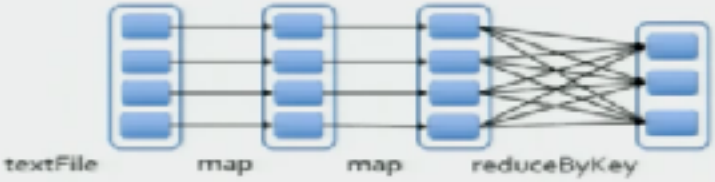
groupByKey() vs ReduceByKey()


Strata+ Hadoop WORLD
STRATACONF.COM
FEB 17-20, 2015
SAN JOSE, CA
GroupByKey: Shuffle Step
Shuffle
With GroupByKey, all the data is wastefully sent over the network and collected on the reduce workers.
databricks


Everyday I'm Shuffling - Tips for Writing Better Apache Spark Programs
Strata+ Hadoop WORLD
STRATACONF.COM
FEB 17-20, 2015
SAN JOSE, CA
ReduceByKey: Shuffle Step
Shuffle
With ReduceByKey, data is combined so each partition outputs at most one value for each key to send over the network.
databricks
6:04 / 36:24

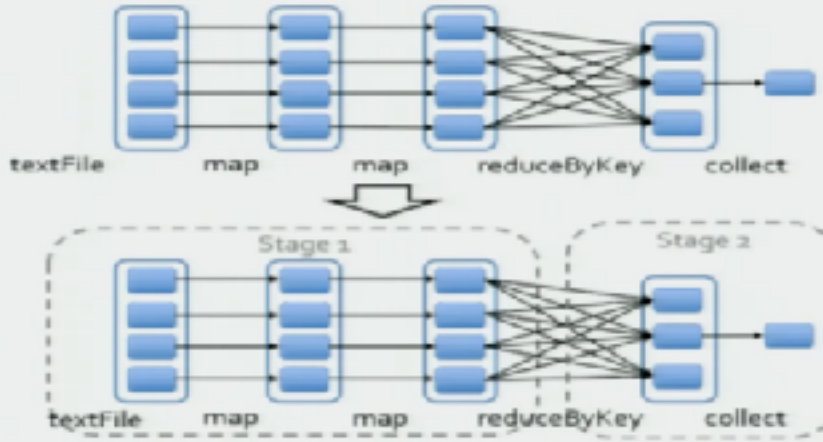
Example

```
sc.textFile("/some-hdfs-data")          RDD[String]
    .map(line => line.split("\t"))      RDD[List[String]]
    .map(parts =>
     (parts[0], int(parts[1])))         RDD[(String, Int)]
    .reduceByKey(_ + _, 3)              RDD[(String, Int)]
```

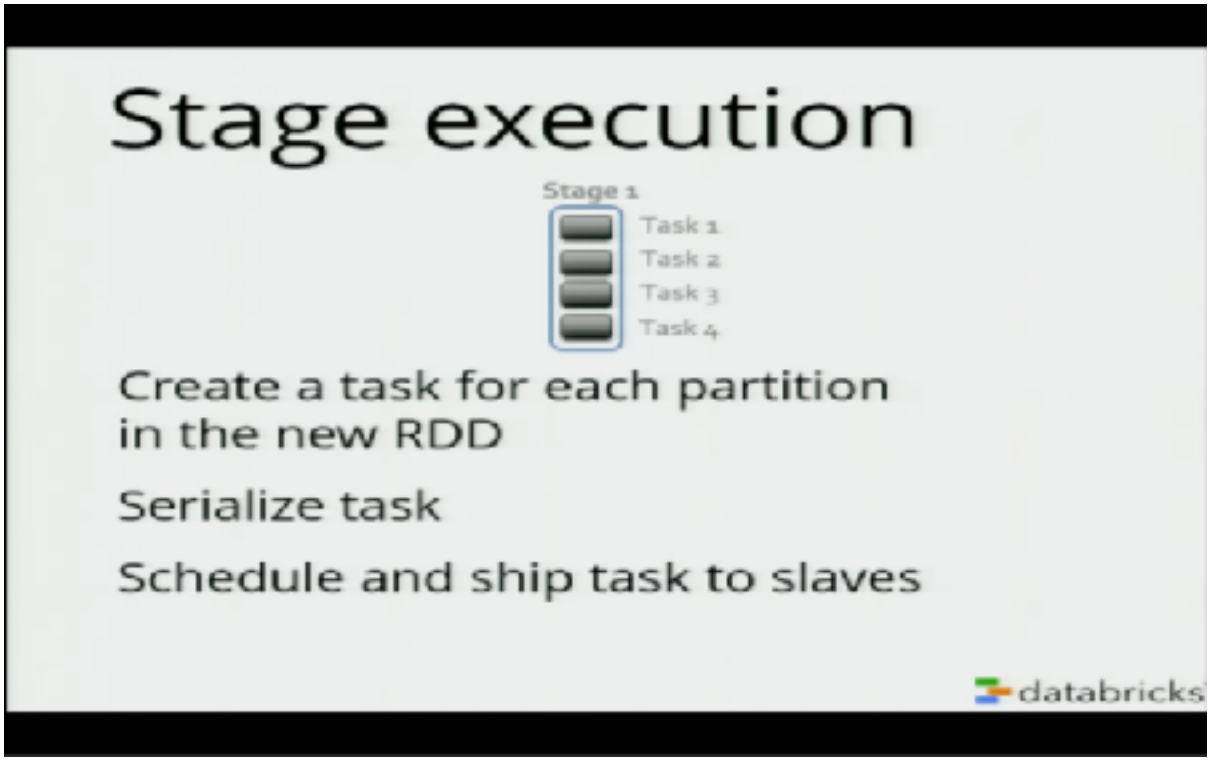


databricks

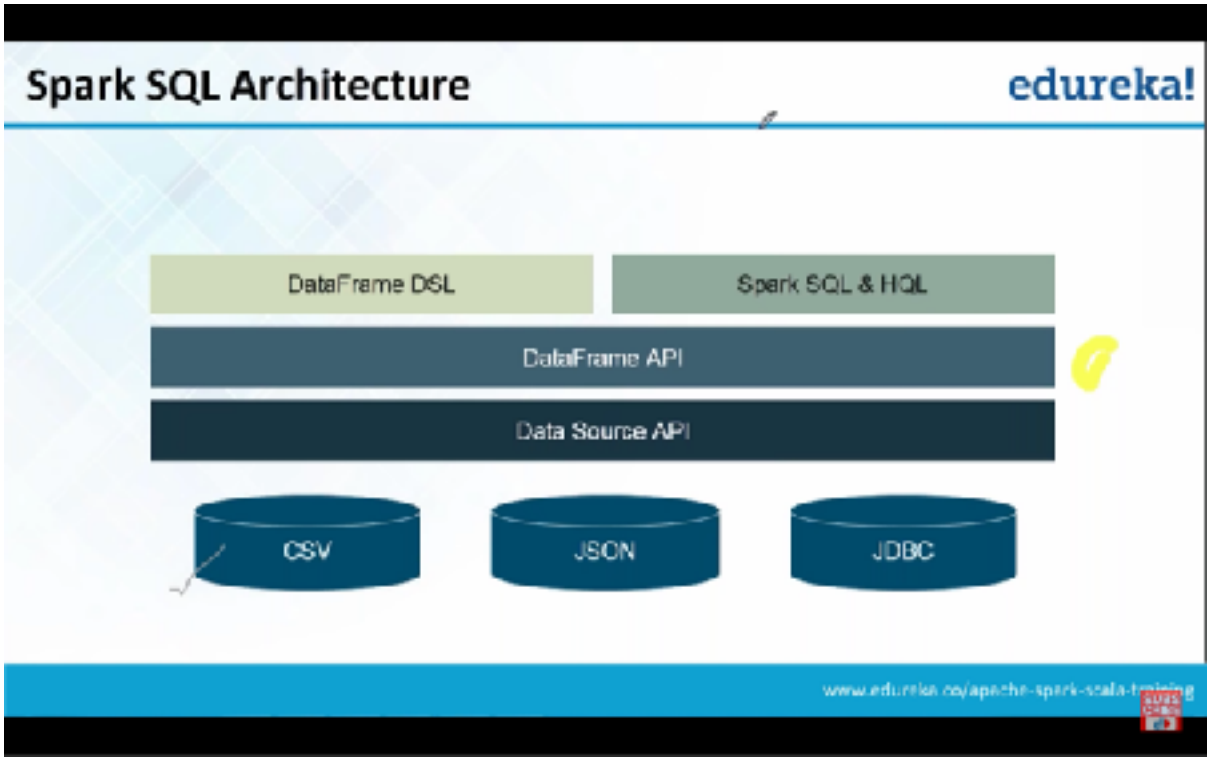
Execution Graph

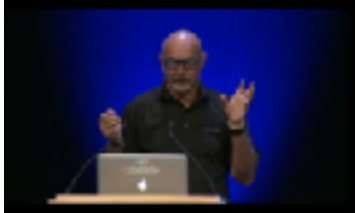


databricks



Datasets & DataFrames





Some code to read Wikipedia

```
val rdd = sc.textFile("/mnt/wikipedi-data/pagecounts.gz")
val parsedRDD = rdd.flatMap {
    line => line.split(""" """) match {
        case Array(project, page, numRequests, _) => Some((project, page, numRequests))
        case _ => None
    }
}
// filter only English pages ; count pages and requests to it.
parsedRDD.filter { case (project, page, numRequests) => project == "en" }.
        map { case (_, page, numRequests) => (page, numRequests) }.
        reduceByKey(_ + _)
        take(100).foreach { case (page, requests) => println(s"$page: $requests") }
```


Structured APIs In Spark

	SQL	DataFrames	Datasets
Syntax Errors	Runtime	Compile Time	Compile Time
Analysis Errors	Runtime	Runtime	Compile Time

Analysis errors are reported before a distributed job starts

DataFrame API code.

```
// convert RDD -> DF with column names
val df = parsedRDD.toDF("project", "page", "numRequests")
//filter, groupBy, sum, and then agg()
df.filter($"project" === "en").
    groupBy($"page").
    agg(sum($"numRequests").as("count")).
    limit(100).
    show(100)
```

project	page	numRequests
en	23	45
en	24	200

SPARK SUMMIT
ORGANIZED BY databricks

databricks

Why structure APIs?

DataFrame

```
data.groupBy("dept").avg("age")
```

SQL

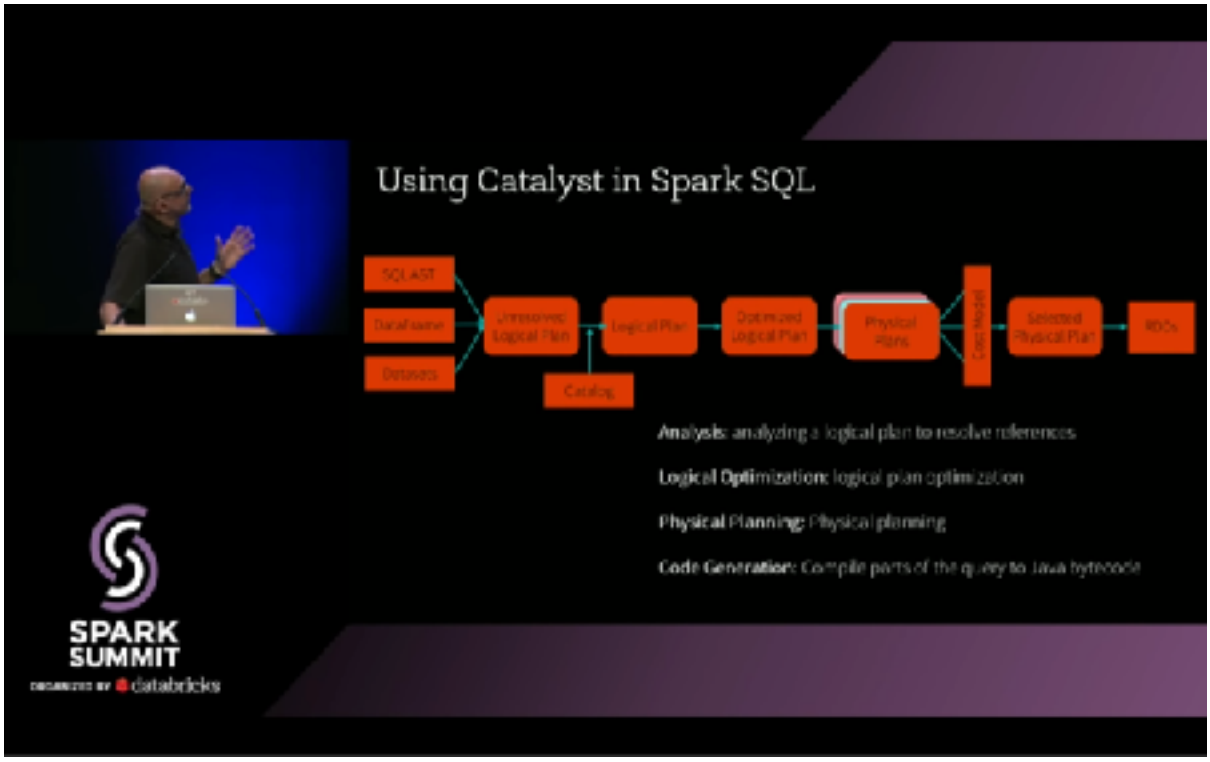
```
select dept, avg(age) from data group by 1
```

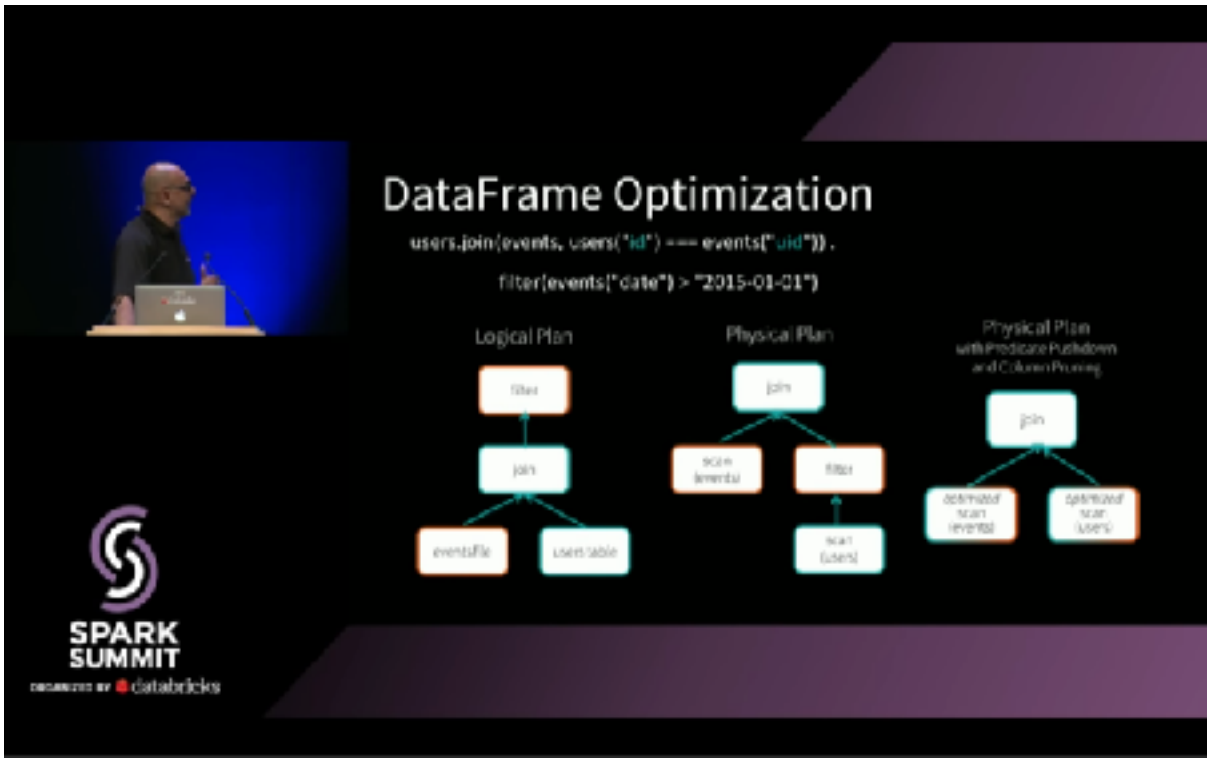
RDD

```
data.map { case (dept, age) => dept -> (age, 1) }
    .reduceByKey { case ((a1, c1), (a2, c2)) => (a1 + a2, c1 + c2)}
    .map { case (dept, (age, c)) => dept -> age / c }
```

SPARK SUMMIT
ORGANIZED BY databricks

databricks


Using Catalyst in Spark SQL
SQL AST
DataFrame
Dataset
Unresolved Logical Plan
Logical Plan
Optimized Logical Plan
Physical Plans
Cost Model
Selected Physical Plan
RDDs
Catalog
Analysis: analyzing a logical plan to resolve references
Logical Optimization: logical plan optimization
Physical Planning: Physical planning
Code Generation: Compile parts of the query to Java bytecode
SPARK SUMMIT
databricks


DataFrame Optimization
users.join(events, users("id") === events("uid")).
filter(events("date") > "2015-01-01")
Logical Plan
Physical Plan
Physical Plan with Predicate Pushdown and Column Pruning
join
filter
events file
users table
join
scan (events)
filter
scan (users)
join
optimized scan (events)
optimized scan (users)
SPARK SUMMIT
databricks

Dataset API in Spark 2.x

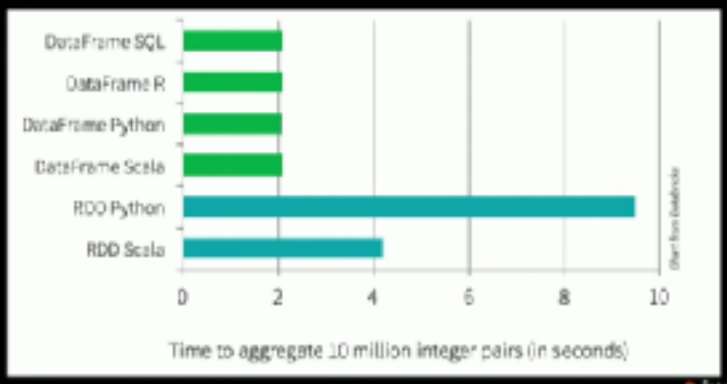
Type-safe: operate on domain objects with compiled lambda functions

```
val df = spark.read.json("people.json")
// Convert data to domain objects.

case class Person(name: String, age: Int)
val ds: Dataset[Person] = df.as[Person]

val filterDS = ds.filter(p=>p.age > 3)
```

SPARK SUMMIT

databricks

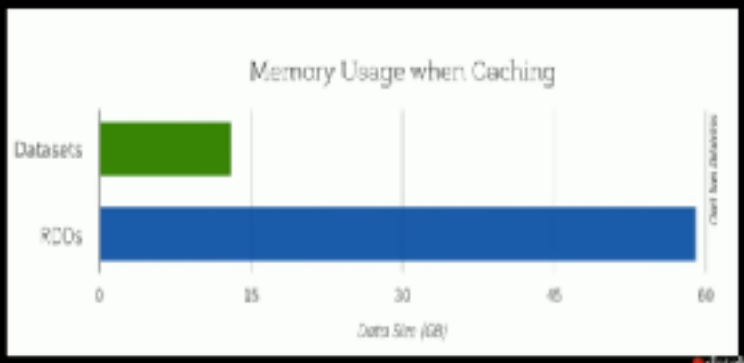
DataFrames are Faster than RDDs



SPARK SUMMIT

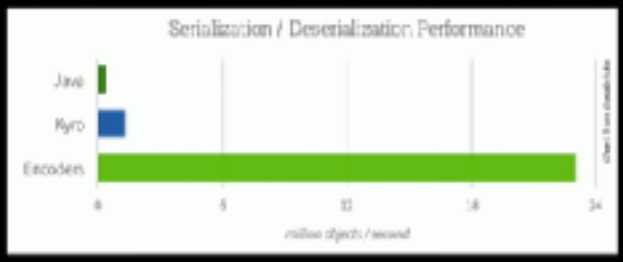
databricks

Datasets < Memory RDDs



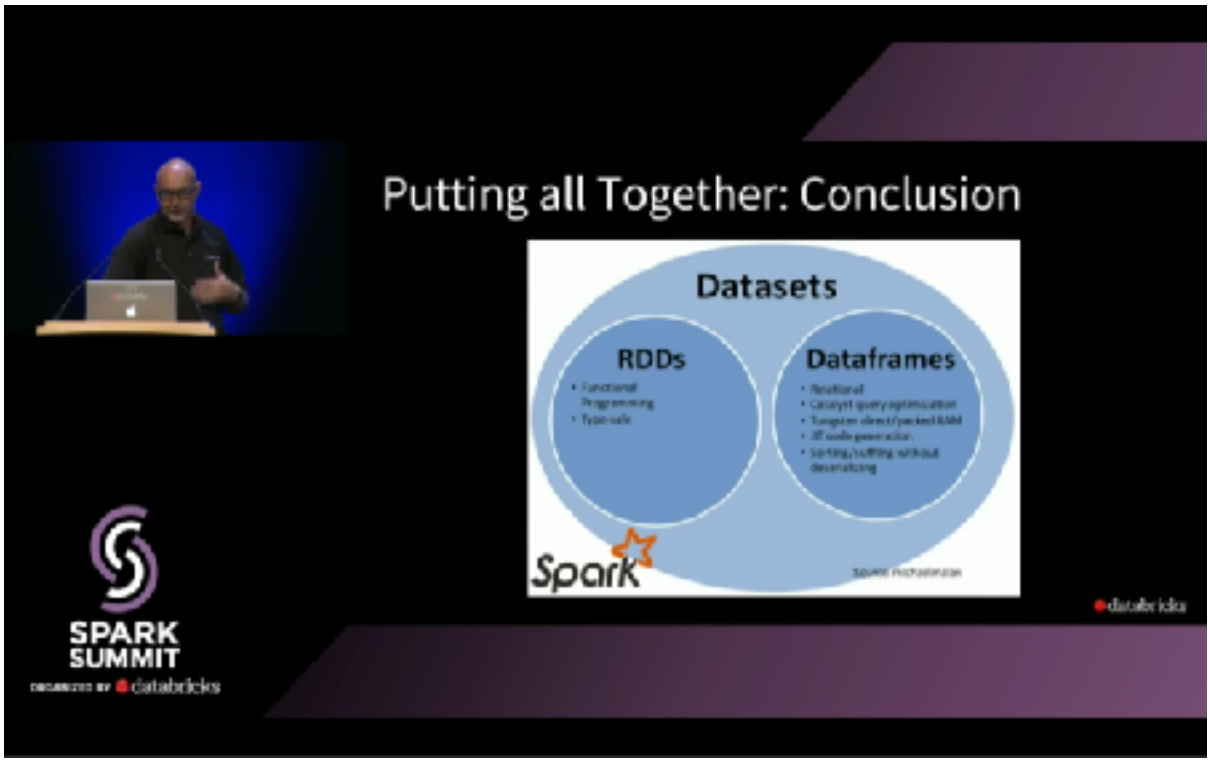
SPARK SUMMIT
databricks

Datasets Faster…



SPARK SUMMIT
databricks


A Tale of Three Apache Spark APIs: RDDs, DataFrames, and Datasets -...
Watch later
Share
DataFrames & Datasets
Why
• High-level APIs and DSL
• Strong Type-safety
• Ease-of-use & Readability
• What-to-do
When
• Structured Data schema
• Code optimization & performance
• Space efficiency with Tungsten
SPARK SUMMIT
databricks
27:59 / 31:18
YouTube


Putting all Together: Conclusion
Datasets
RDDs
Dataframes
Spark
SPARK SUMMIT
databricks

Adaptive Execution Engine/shuffling

Catalyst Optimizer:
Catalyst is an excellent optimizer in SparkSQL, provides open interface for **rule-based optimization in planning stage.** However, the static (rule-based) optimization will not consider any data distribution at runtime.

Adaptive Execution Engine:
A technology called Adaptive Execution has been introduced since Spark 2.0 and aims to cover this part, but still pending in early stage. We enhanced the existing Adaptive Execution feature, **and focus on the execution plan adjustment at runtime** according to different staged intermediate outputs, like

- set partition numbers for joins and aggregations,
- avoid unnecessary data shuffling and disk IO,
- handle data skew cases, and
- even optimize the join order like CBO etc..

In the benchmark comparison experiments, this feature save huge manual efforts in tuning the parameters like

- the shuffled partition number, which is error-prone and misleading.

Challenges with Catalyst optimizer:

- Shuffle Partition
- Number Best Execution Plan
- Data Skew

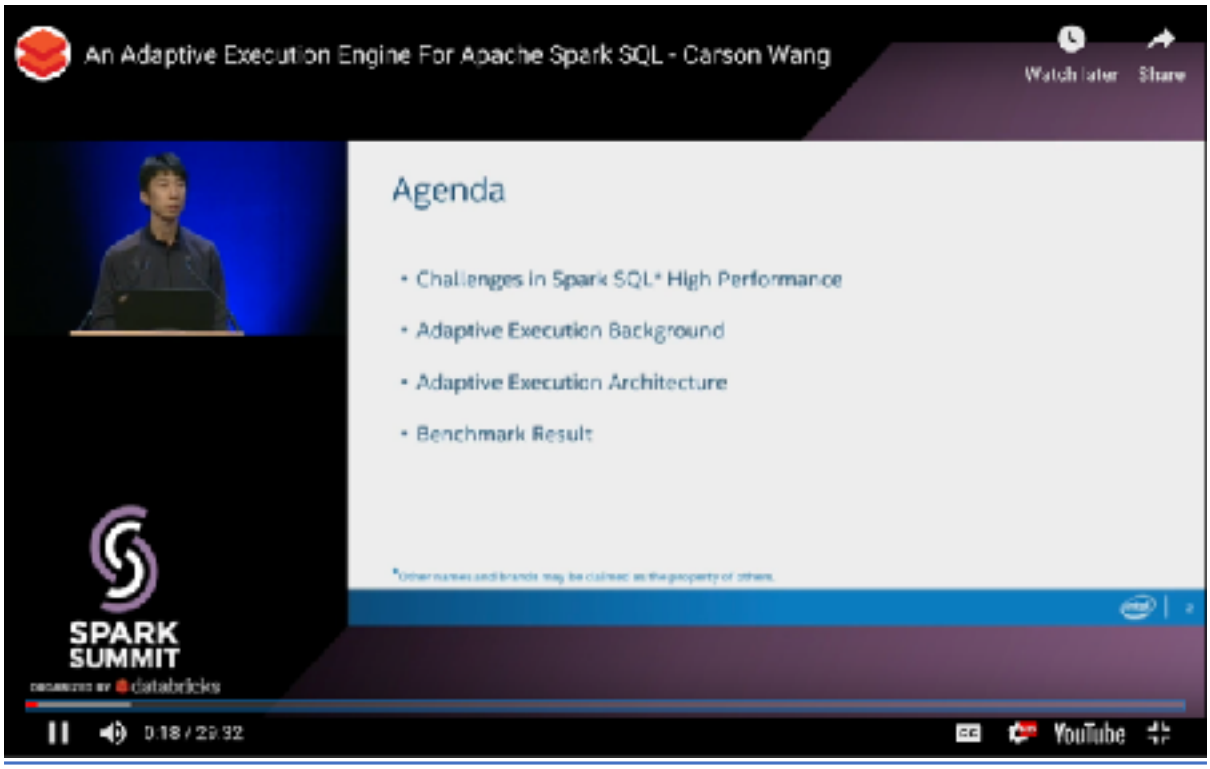
RunTime Decisions:

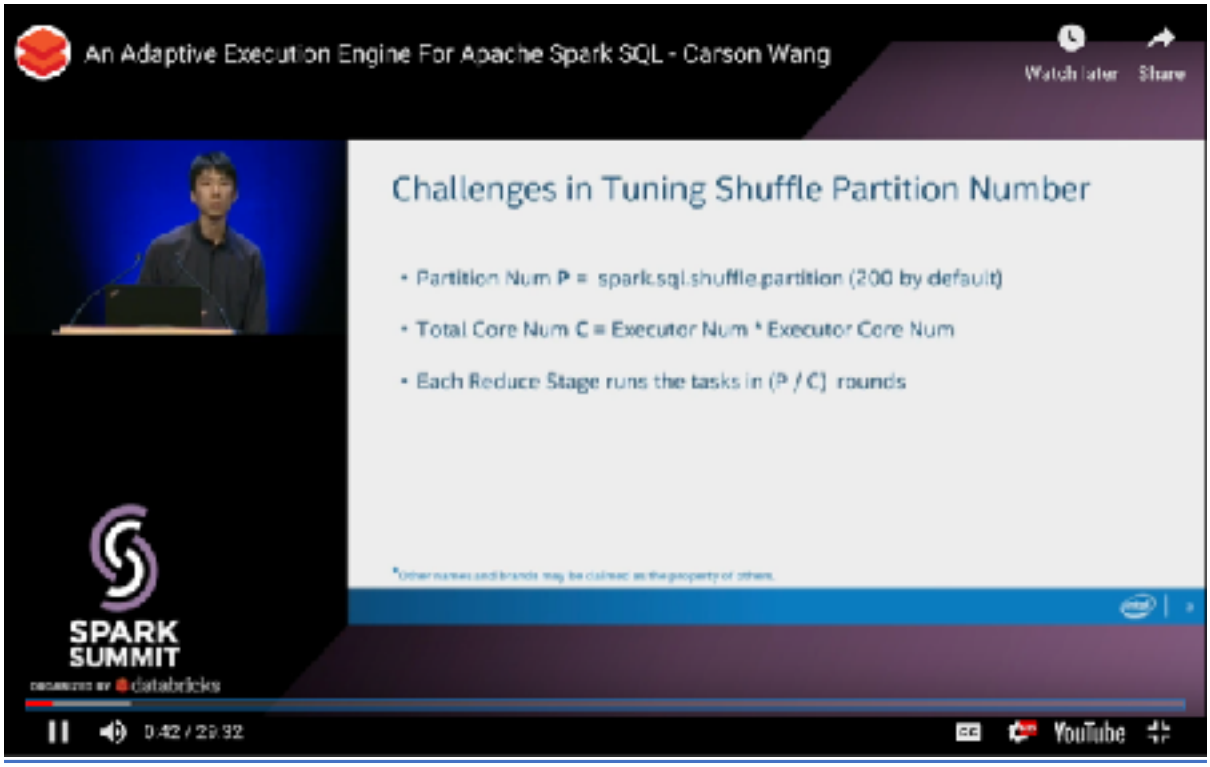
- Auto Setting the Number of Reducers
- Execution Plan Optimization at Runtime
- Handling Skewed Join / HandlingSkewedShuffleJoinInputData
- DeterminingtheJoinStrategy

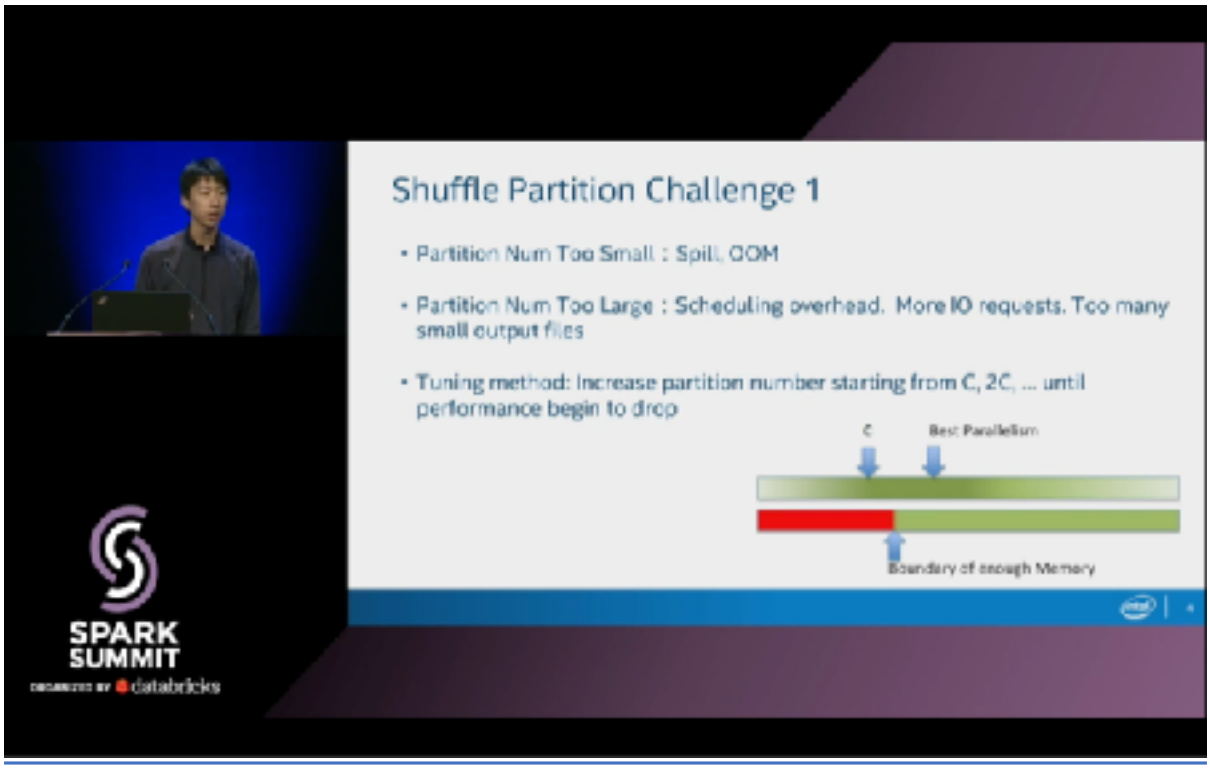
For more info check design doc at:
https://issues.apache.org/jira/browse/SPARK-9850

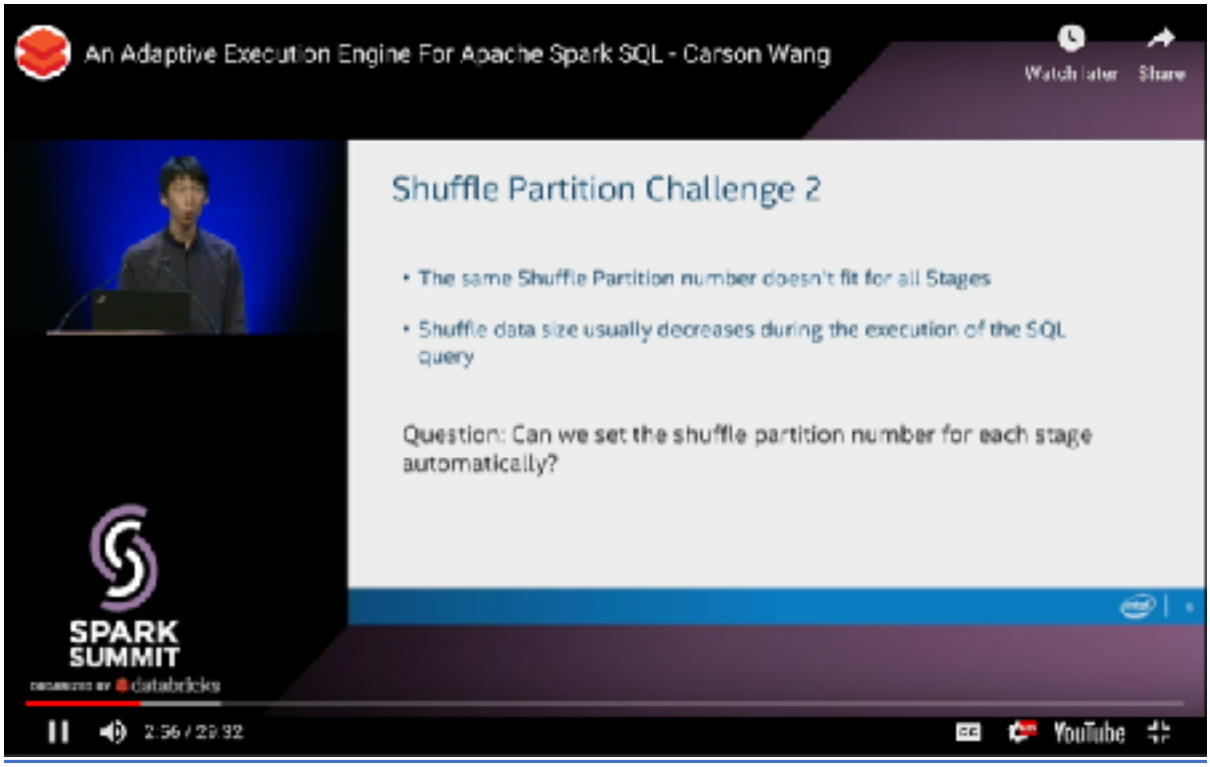
https://software.intel.com/en-us/articles/spark-sql-adaptive-execution-at-100-tb

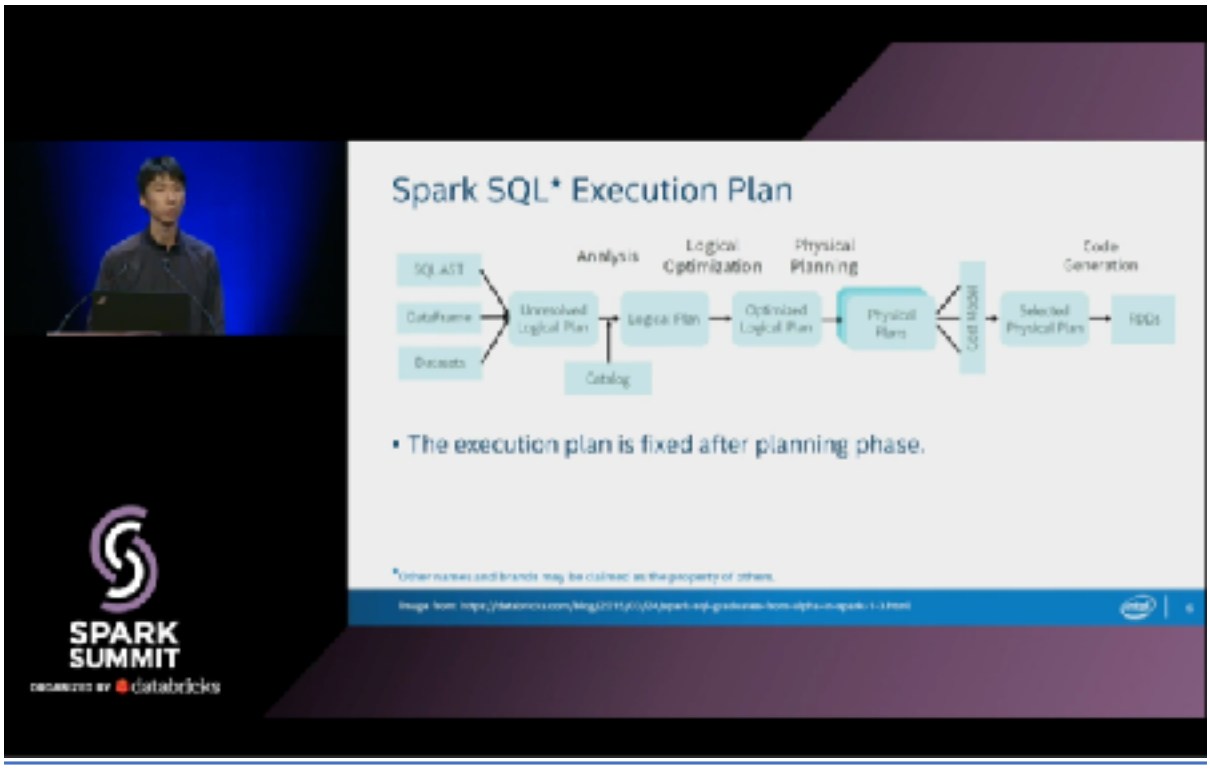

An Adaptive Execution Engine For Apache Spark SQL - Carson Wang
Watch later
Share
intel
AN ADAPTIVE EXECUTION ENGINE FOR APACHE SPARK SQL
Carson Wang (carson.wang@intel.com)
Yucai Yu (yucai.yu@intel.com)
Hao Cheng (hao.cheng@intel.com)
SPARK SUMMIT
0:09 / 29:32
YouTube


An Adaptive Execution Engine For Apache Spark SQL - Carson Wang
Watch later
Share
Agenda
• Challenges in Spark SQL* High Performance
• Adaptive Execution Background
• Adaptive Execution Architecture
• Benchmark Result
SPARK SUMMIT
0:18 / 29:32
YouTube

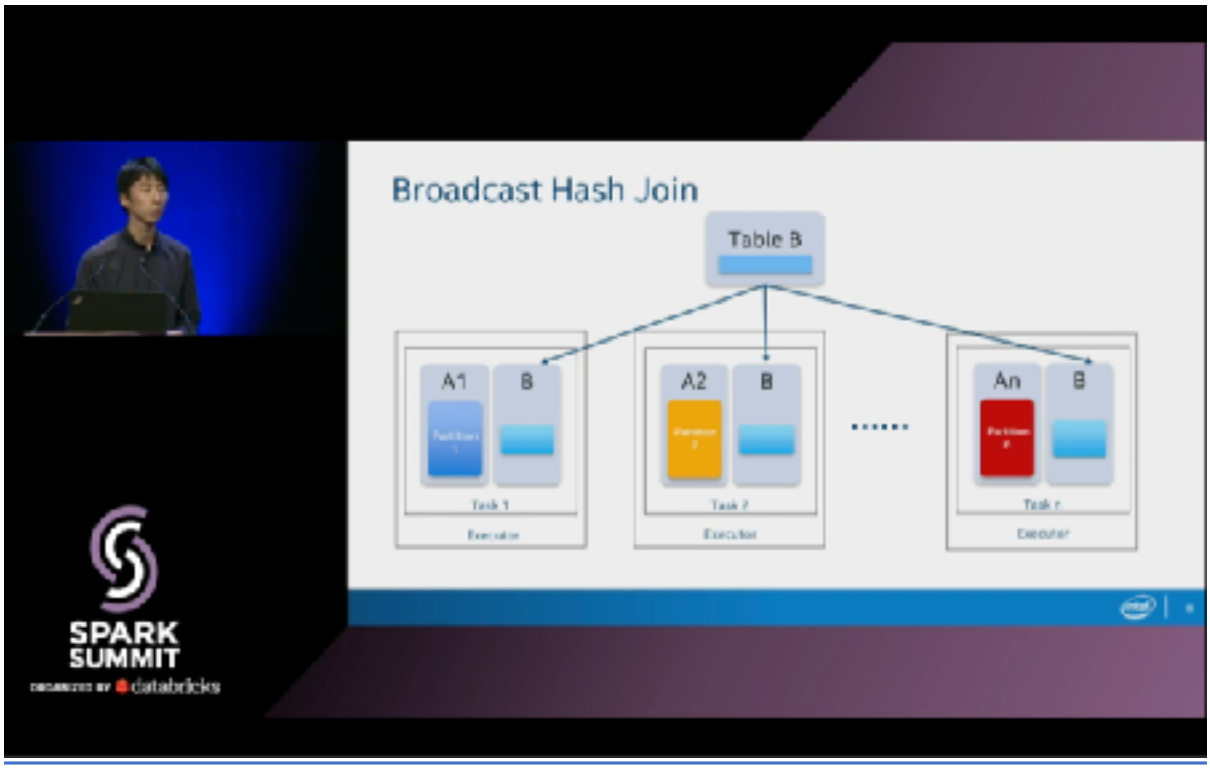

An Adaptive Execution Engine For Apache Spark SQL - Carson Wang
Watch later
Share
Challenges in Tuning Shuffle Partition Number
• Partition Num P = spark.sql.shuffle.partition (200 by default)
• Total Core Num C = Executor Num * Executor Core Num
• Each Reduce Stage runs the tasks in (P / C) rounds
SPARK SUMMIT
0:42 / 29:32
YouTube

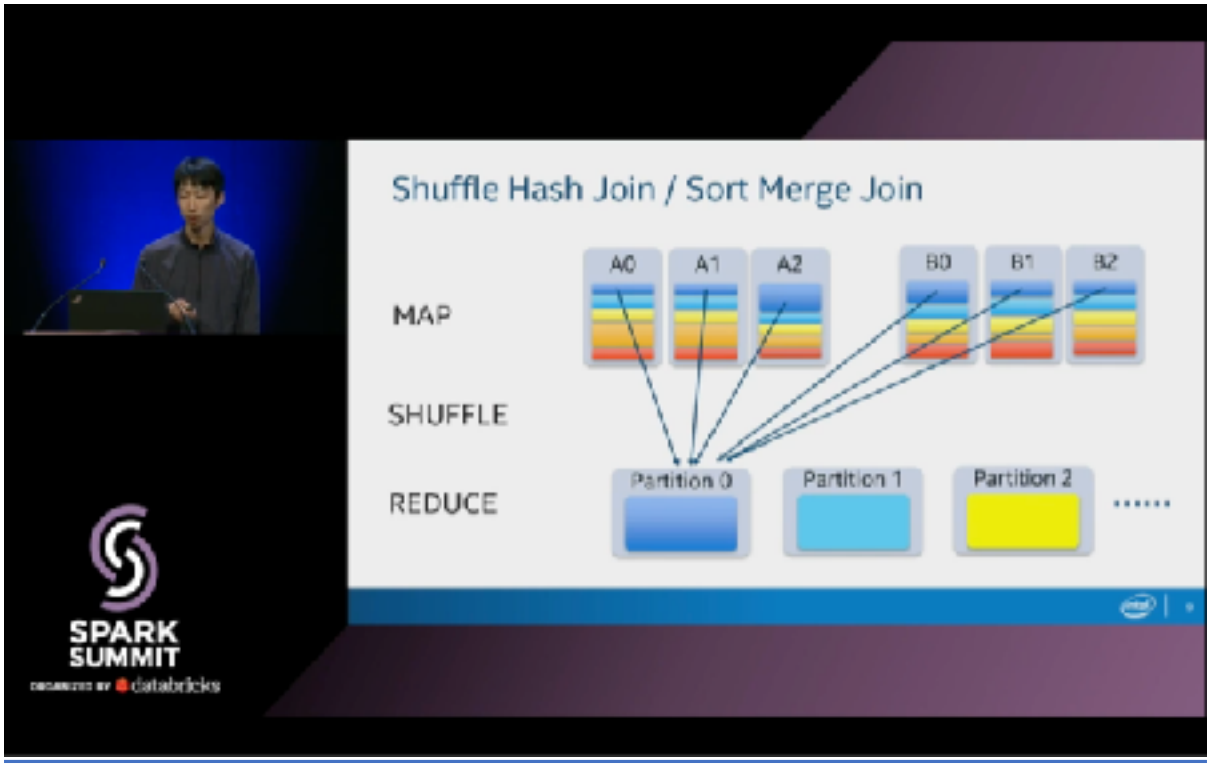

Shuffle Partition Challenge 1
• Partition Num Too Small : Spill, OOM
• Partition Num Too Large : Scheduling overhead. More IO requests. Too many small output files
• Tuning method: Increase partition number starting from C, 2C, ... until performance begin to drop
C
Best Parallelism
Boundary of enough Memory
SPARK SUMMIT


An Adaptive Execution Engine For Apache Spark SQL - Carson Wang
Watch later
Share
Shuffle Partition Challenge 2
• The same Shuffle Partition number doesn't fit for all Stages
• Shuffle data size usually decreases during the execution of the SQL query
Question: Can we set the shuffle partition number for each stage automatically?
SPARK SUMMIT
databricks
2:56 / 29:32
YouTube

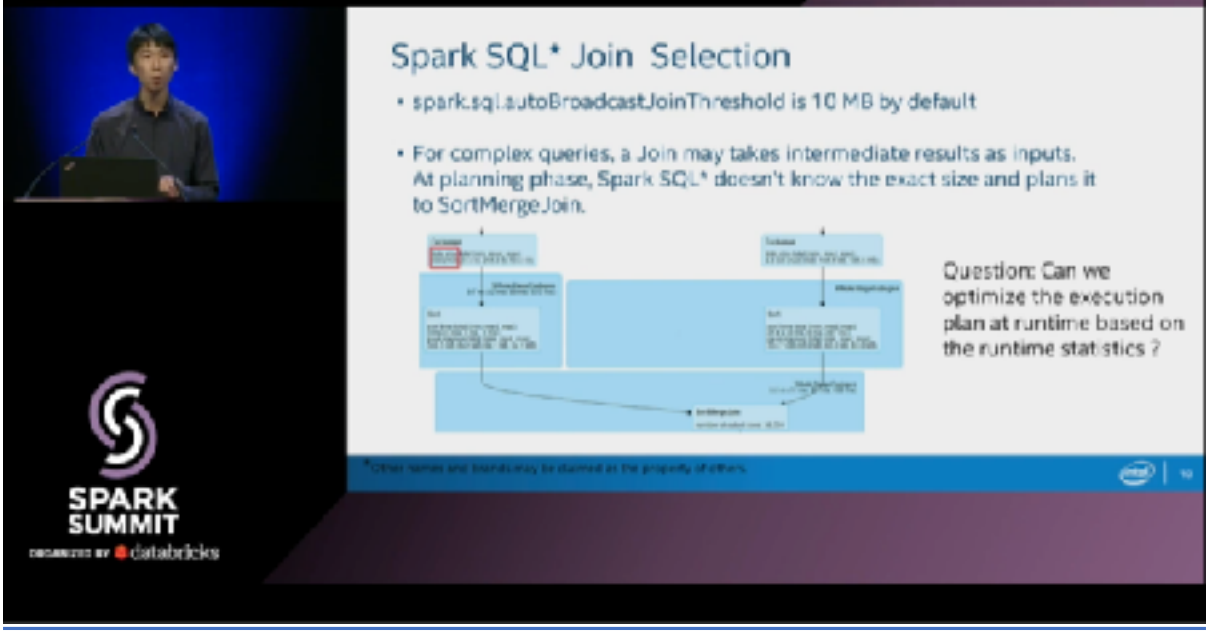

Spark SQL* Execution Plan
Analysis
Logical Optimization
Physical Planning
Code Generation
SQL AST
DataFrame
Dataset
Unresolved Logical Plan
Logical Plan
Optimized Logical Plan
Physical Plans
Cost Model
Selected Physical Plan
RDDs
Catalog
• The execution plan is fixed after planning phase.
SPARK SUMMIT
databricks


Spark SQL* Join Selection
SELECT xxx
FROM A
JOIN B
ON A.Key1 = B.Key2
SPARK SUMMIT


Broadcast Hash Join
Table B
A1
B
A2
B
An
B
SPARK SUMMIT

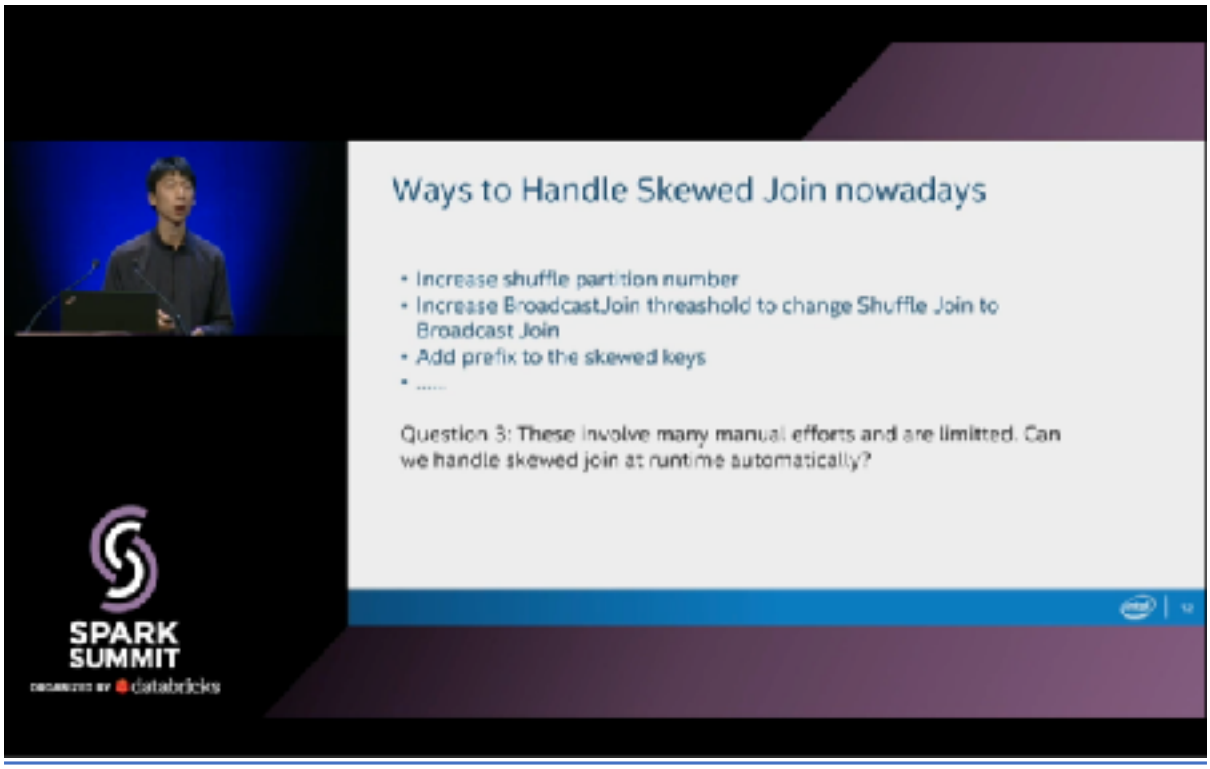

Shuffle Hash Join / Sort Merge Join
MAP
SHUFFLE
REDUCE
A0
A1
A2
B0
B1
B2
Partition 0
Partition 1
Partition 2
SPARK SUMMIT
databricks


Spark SQL* Join Selection
• spark.sql.autoBroadcastJoinThreshold is 10 MB by default
• For complex queries, a Join may takes intermediate results as inputs. At planning phase, Spark SQL* doesn't know the exact size and plans it to SortMergeJoin.
SPARK SUMMIT
databricks


Spark SQL* Join Selection
• spark.sql.autoBroadcastJoinThreshold is 10 MB by default
• For complex queries, a Join may takes intermediate results as inputs. At planning phase, Spark SQL* doesn't know the exact size and plans it to SortMergeJoin.
Question: Can we optimize the execution plan at runtime based on the runtime statistics ?
SPARK SUMMIT
databricks

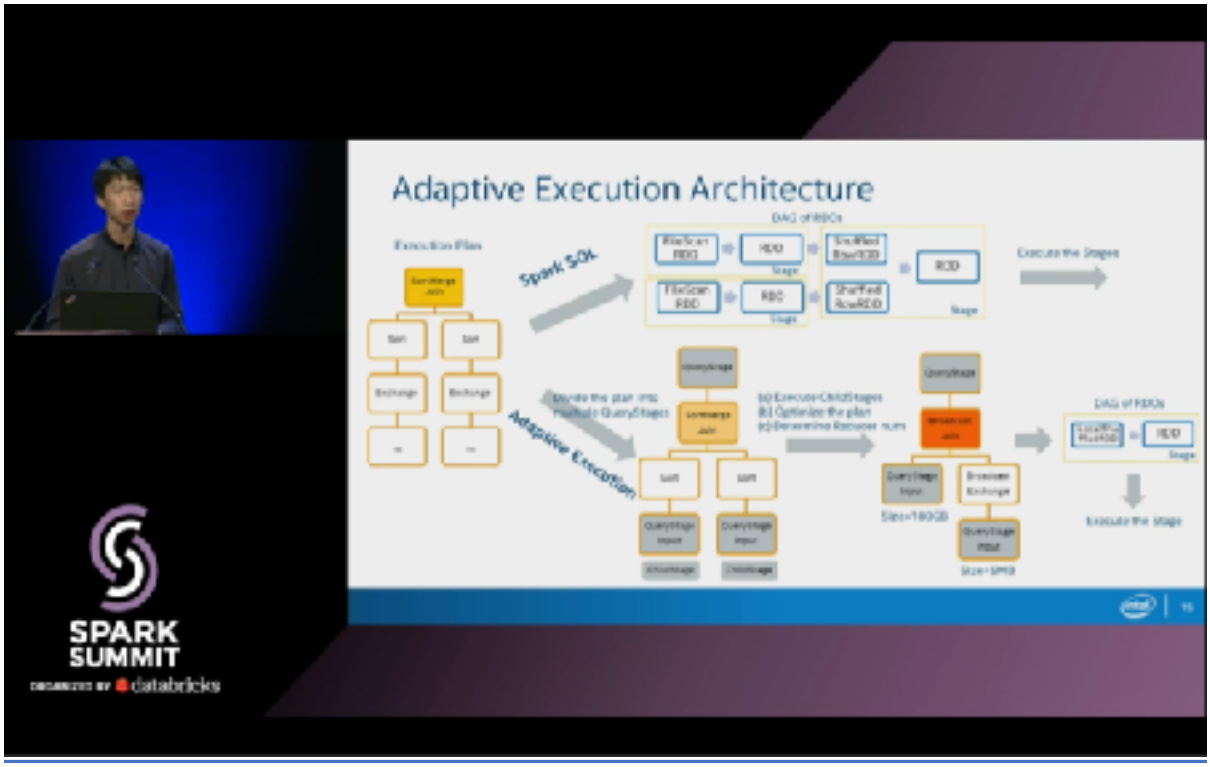

Data Skew in Join
• Data in some partitions are extremely larger than other partitions.
• Data skew is a common source of slowness for Shuffle Joins
SPARK SUMMIT
databricks

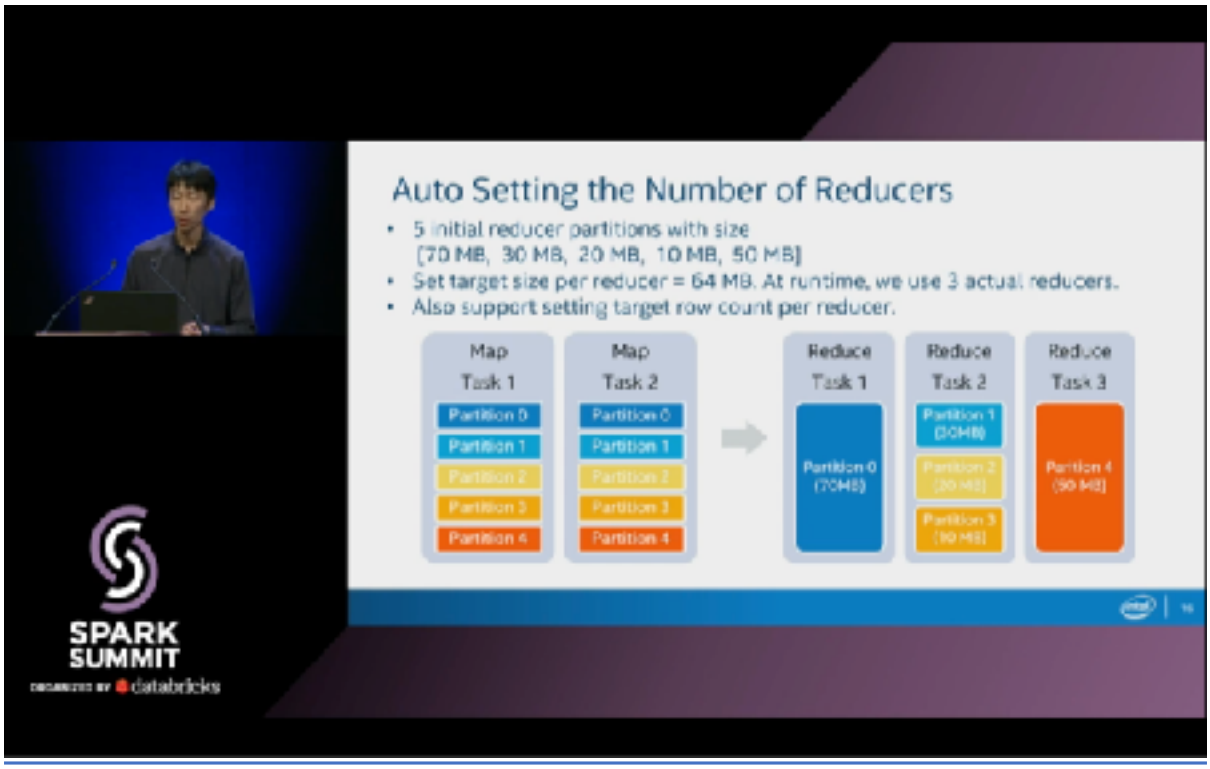

Ways to Handle Skewed Join nowadays
• Increase shuffle partition number
• Increase BroadcastJoin threashold to change Shuffle Join to Broadcast Join
• Add prefix to the skewed keys
• .....
SPARK SUMMIT
databricks

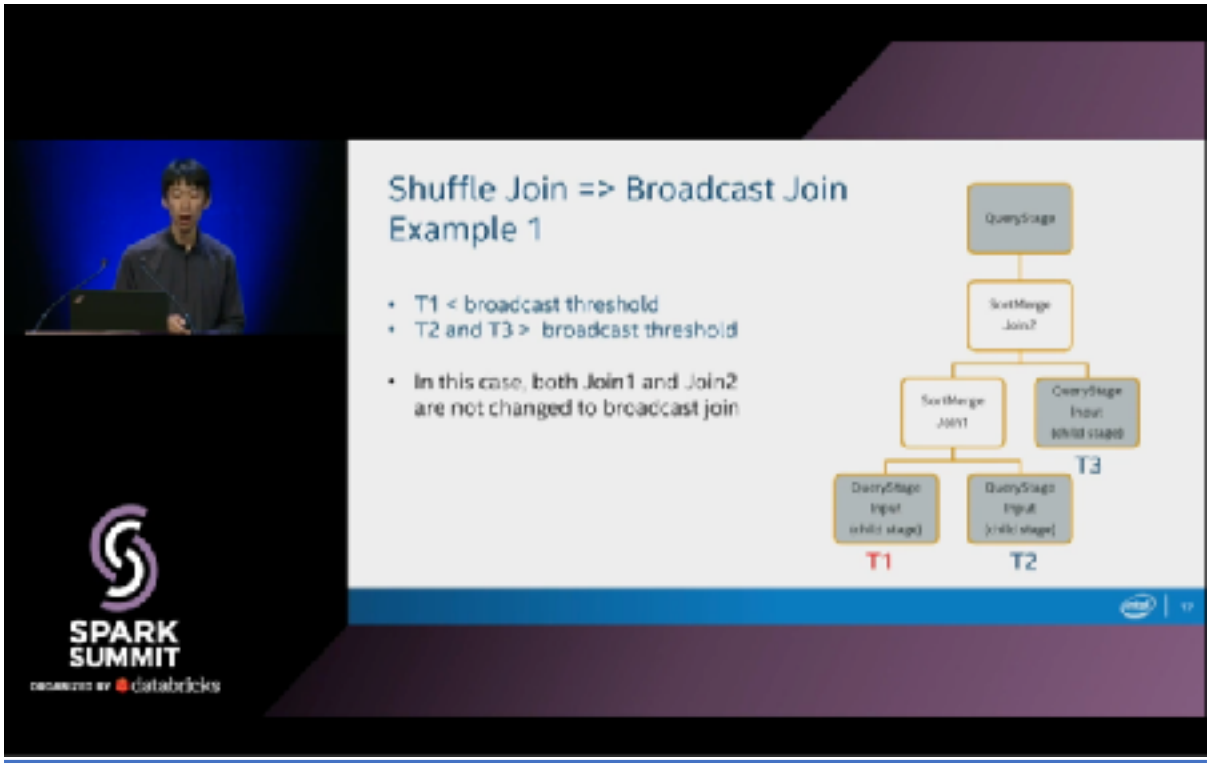

Ways to Handle Skewed Join nowadays
• Increase shuffle partition number
• Increase BroadcastJoin threashold to change Shuffle Join to Broadcast Join
• Add prefix to the skewed keys
• .....
Question 3: These involve many manual efforts and are limited. Can we handle skewed join at runtime automatically?
SPARK SUMMIT
databricks

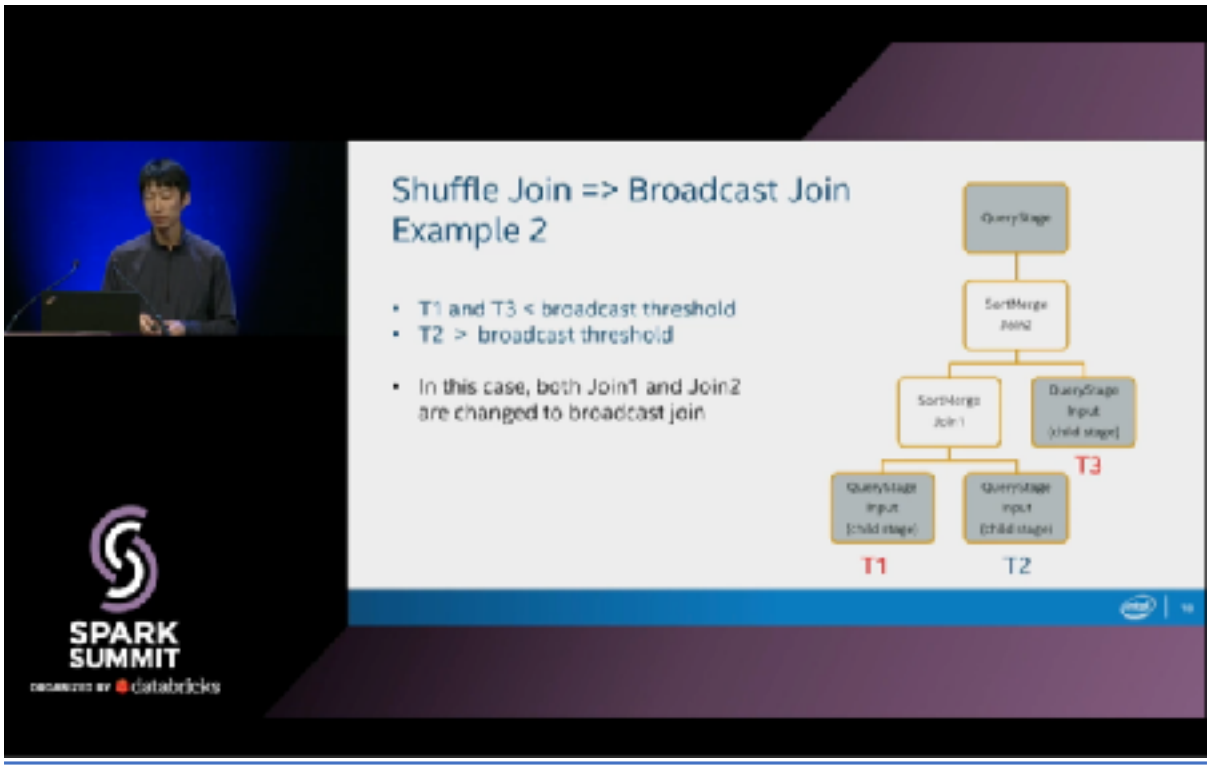

Adaptive Execution Background
• SPARK-9850: Adaptive execution in Spark*
• SPARK-9851: Support submitting map stages individually in DAGScheduler
• SPARK-9858: Introduce an ExchangeCoordinator to estimate the number of post-shuffle partitions.
*Other names and brands may be claimed as the property of others.
SPARK SUMMIT
databricks


A New Adaptive Execution Engine in Spark SQL*
*Other names and brands may be claimed as the property of others.
SPARK SUMMIT
databricks

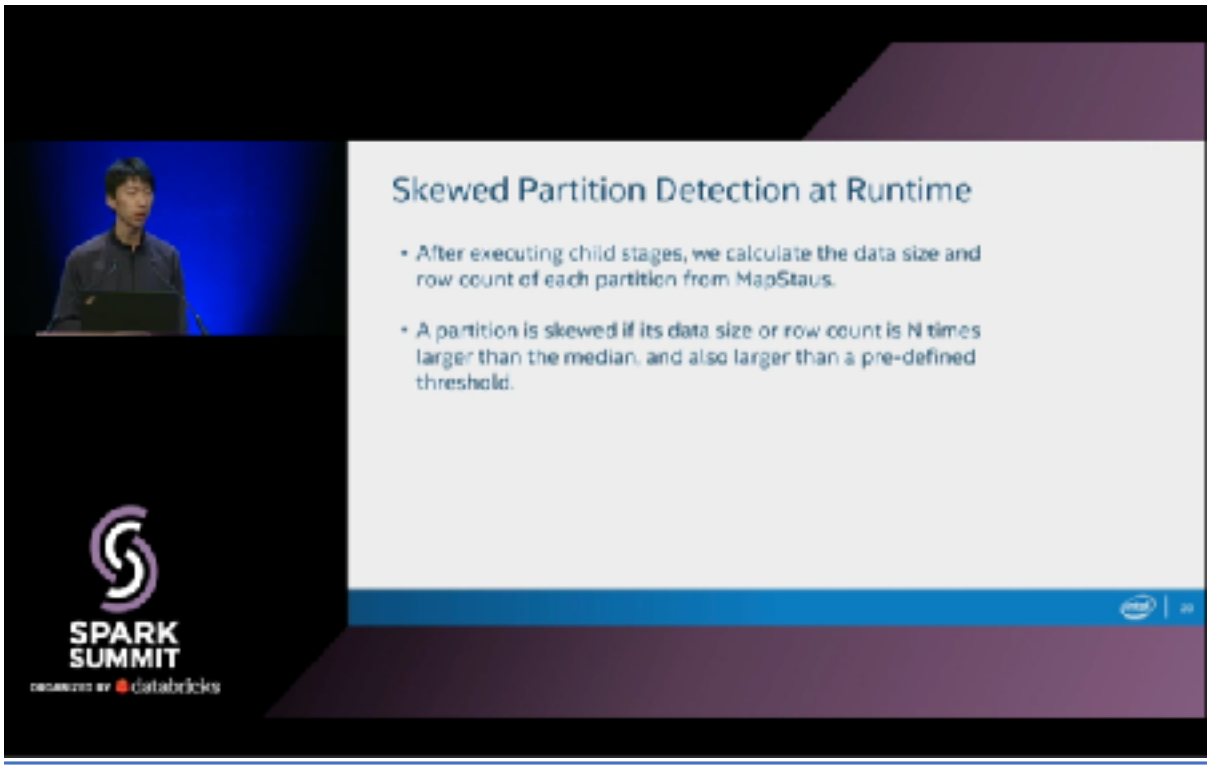

Adaptive Execution Architecture
Spark SQL
Adaptive Execution
SPARK SUMMIT
databricks

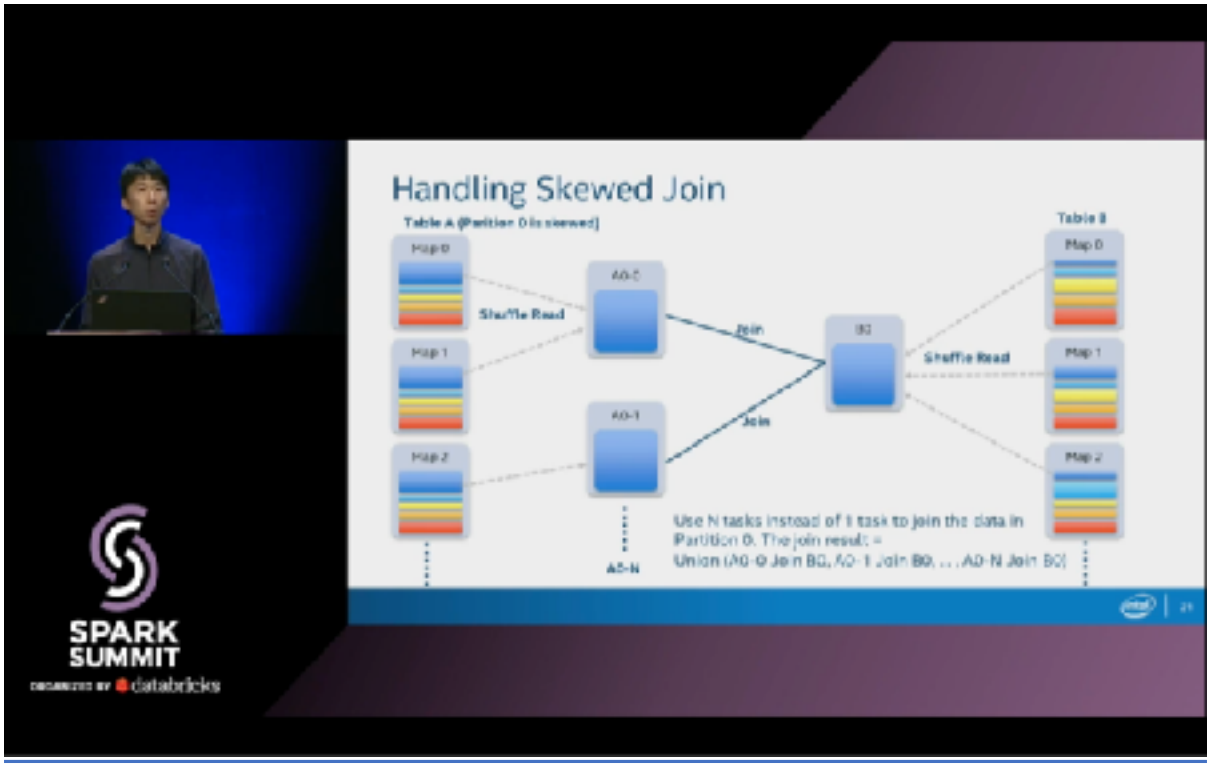

Auto Setting the Number of Reducers
5 initial reducer partitions with size
[70 MB, 30 MB, 20 MB, 10 MB, 50 MB]
Set target size per reducer = 64 MB. At runtime, we use 3 actual reducers.
Also support setting target row count per reducer.
Map Task 1
Map Task 2
Reduce Task 1
Reduce Task 2
Reduce Task 3
Partition 0
Partition 1
Partition 2
Partition 3
Partition 4
Partition 0 (70MB)
Partition 1 (30MB)
Partition 4 (50 MB)
SPARK SUMMIT
databricks

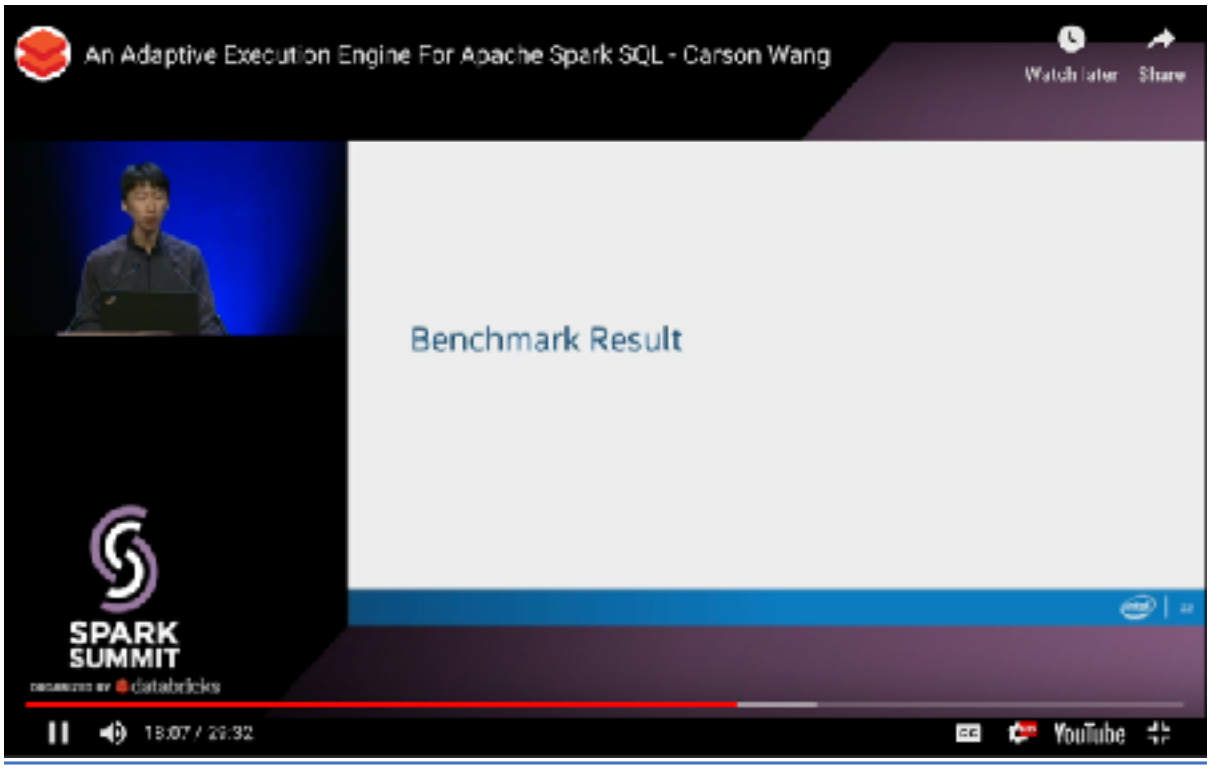

Shuffle Join => Broadcast Join
Example 1
• T1 < broadcast threshold
• T2 and T3 > broadcast threshold
• In this case, both Join1 and Join2 are not changed to broadcast join
T3
T1
T2
SPARK SUMMIT


Shuffle Join => Broadcast Join
Example 2
• T1 and T3 < broadcast threshold
• T2 > broadcast threshold
• In this case, both Join1 and Join2 are changed to broadcast join
T3
T1
T2
SPARK SUMMIT

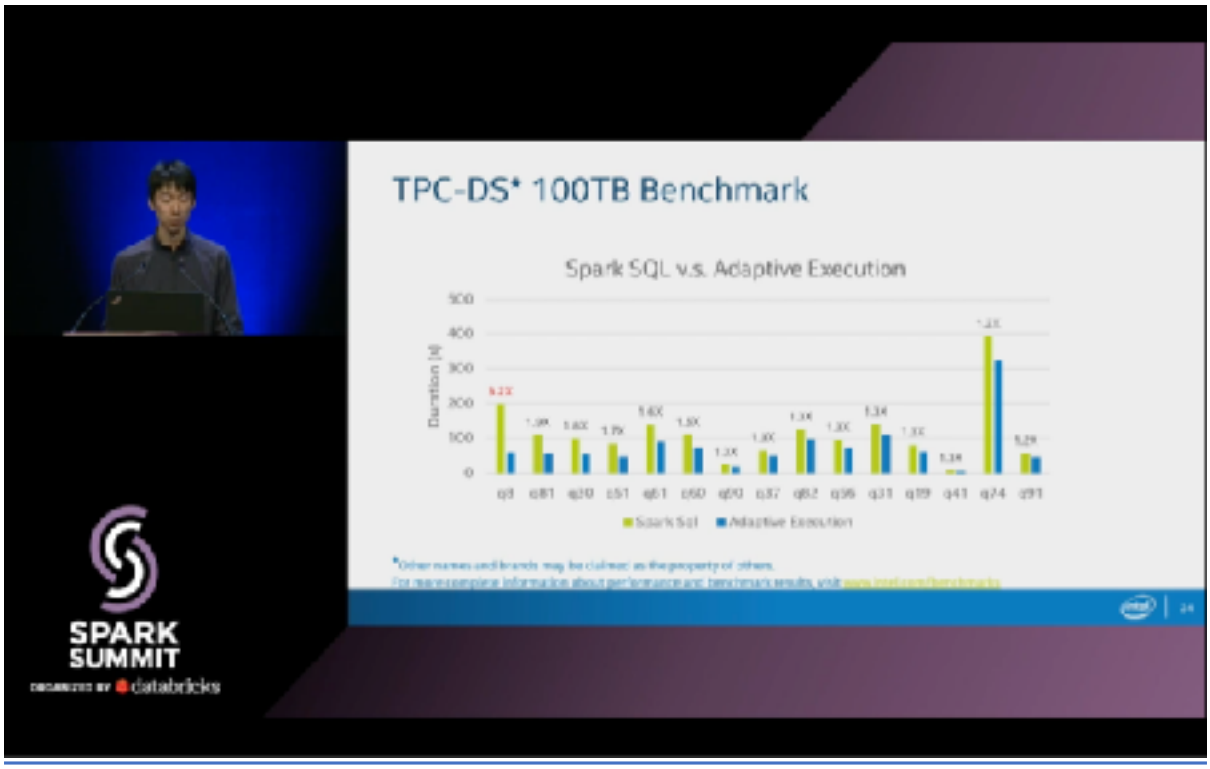

Remote Shuffle Read => Local Shuffle Read
A0
B0
A1
B1
Shuffle Join
Broadcast Join
Remote Shuffle Read
Local Shuffle Read
SPARK SUMMIT
databricks

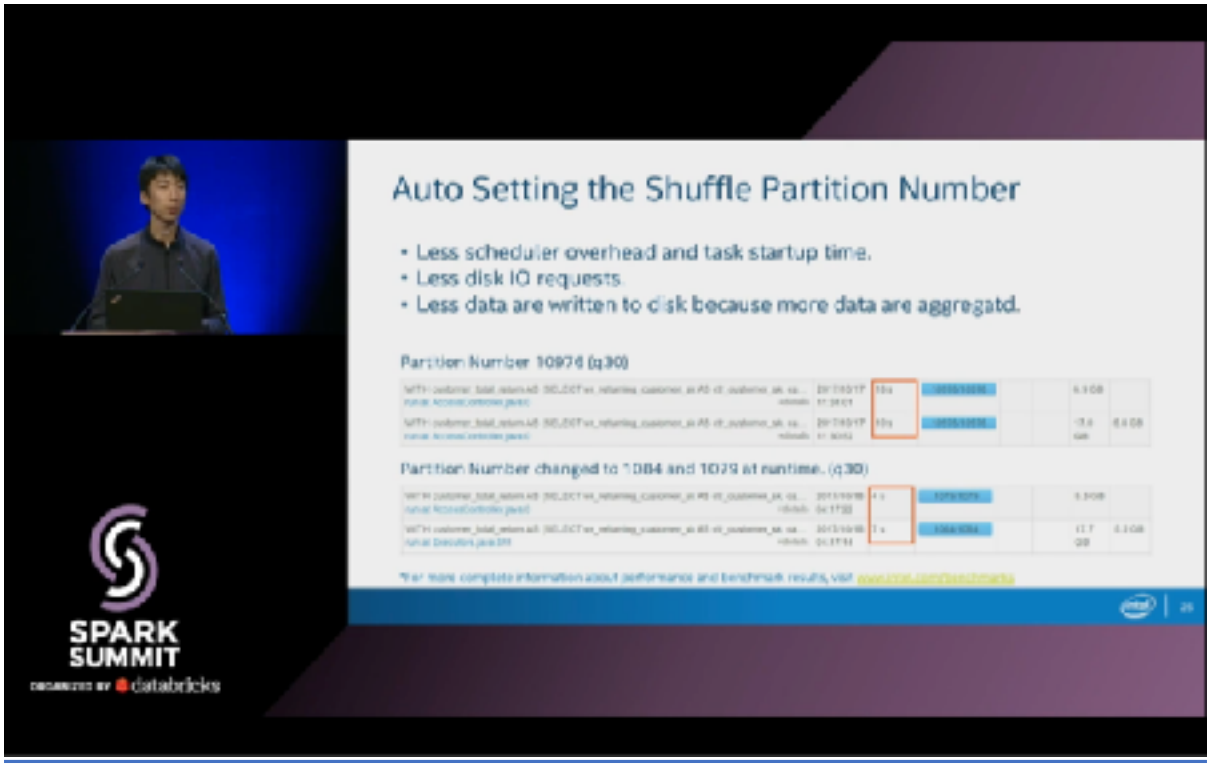

Skewed Partition Detection at Runtime
• After executing child stages, we calculate the data size and row count of each partition from MapStaus.
• A partition is skewed if its data size or row count is N times larger than the median, and also larger than a pre-defined threshold.
SPARK SUMMIT
databricks

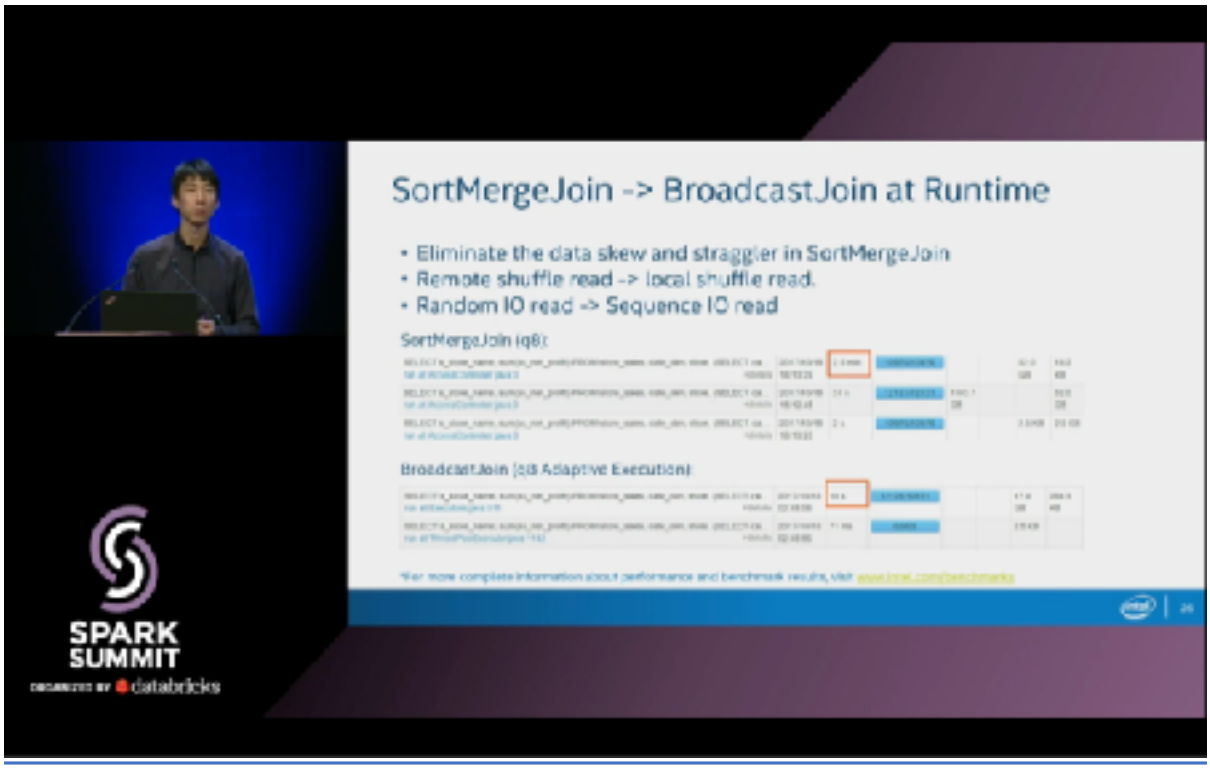

Handling Skewed Join
Table A (Partition 0 is skewed)
Table B
Map 0
Map 1
Map 2
Shuffle Read
A0-0
A0-1
A0-N
Join
B0
Use N tasks instead of 1 task to join the data in Partition 0. The join result: =
SPARK SUMMIT
databricks

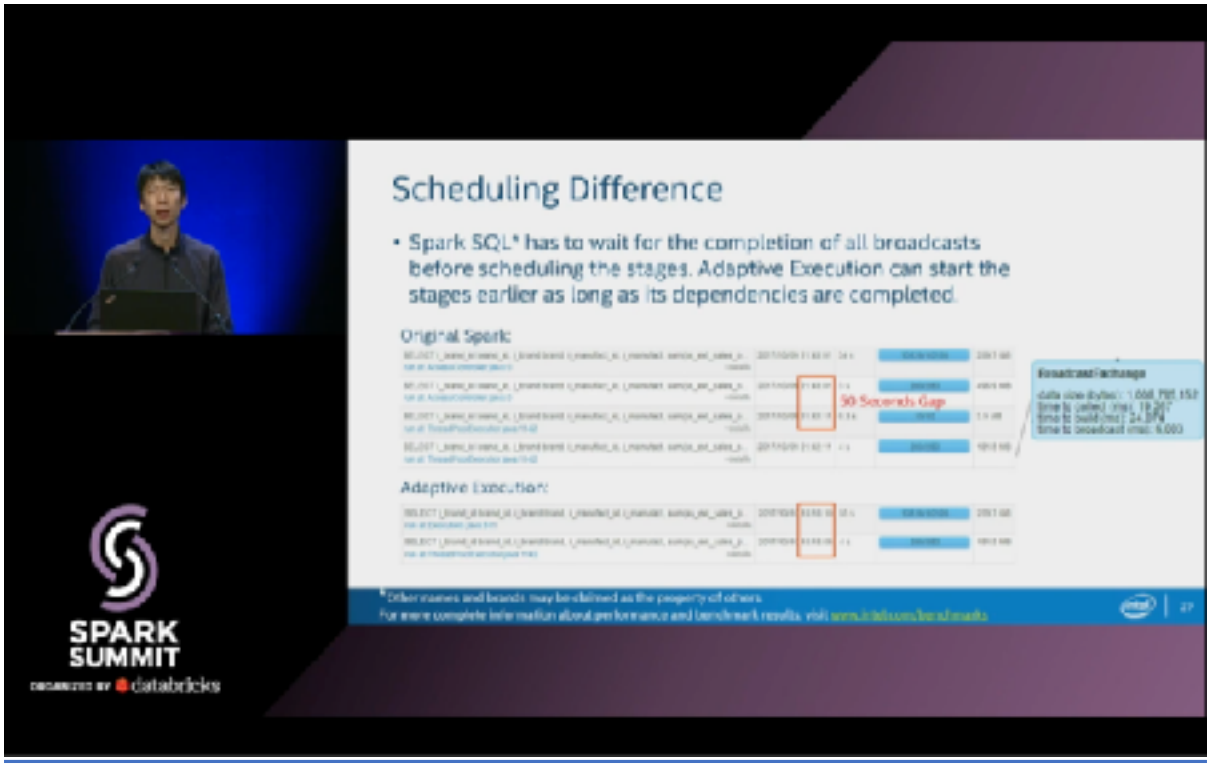

An Adaptive Execution Engine For Apache Spark SQL - Carson Wang
Watch later
Share
Benchmark Result
SPARK SUMMIT
databricks
19:07 / 29:32
YouTube


Cluster Setup
Hardware
Slave
Master
CPU
Memory
Disk
Network
Software
OS
Spark*
Hadoop*/HDFS*
JDK
SPARK SUMMIT
databricks


TPC-DS* 100TB Benchmark
Spark SQL v.s. Adaptive Execution
Duration [s]
Spark SQL
Adaptive Execution
SPARK SUMMIT
databricks


Auto Setting the Shuffle Partition Number
• Less scheduler overhead and task startup time.
• Less disk IO requests.
• Less data are written to disk because more data are aggregatd.
Partition Number 10976 (q30)
Partition Number changed to 1084 and 1079 at runtime. (q30)
SPARK SUMMIT


SortMergeJoin -> BroadcastJoin at Runtime
• Eliminate the data skew and straggler in SortMergeJoin
• Remote shuffle read -> local shuffle read.
• Random IO read -> Sequence IO read
SortMergeJoin (q8):
BroadcastJoin (Still Adaptive Execution):
SPARK SUMMIT



References:
https://databricks.com/session/an-adaptive-execution-engine-for-apache-spark-sql

https://issues.apache.org/jira/browse/SPARK-9850