Spark SQL: Past, Present and Future

Wenchen Fan

2017-9-2

databricks

About Me

- Software Engineer @databricks
- Apache Spark Committer and PMC Member
- One of the most active Spark contributors

About Databricks

TEAM

Started Spark project (now Apache Spark) at UC Berkeley in 2009

MISSON

Make Big Data Simple

PRODUCT

Unified Analytics Platform

A long time ago in a galaxy far far away…

Birth of Spark

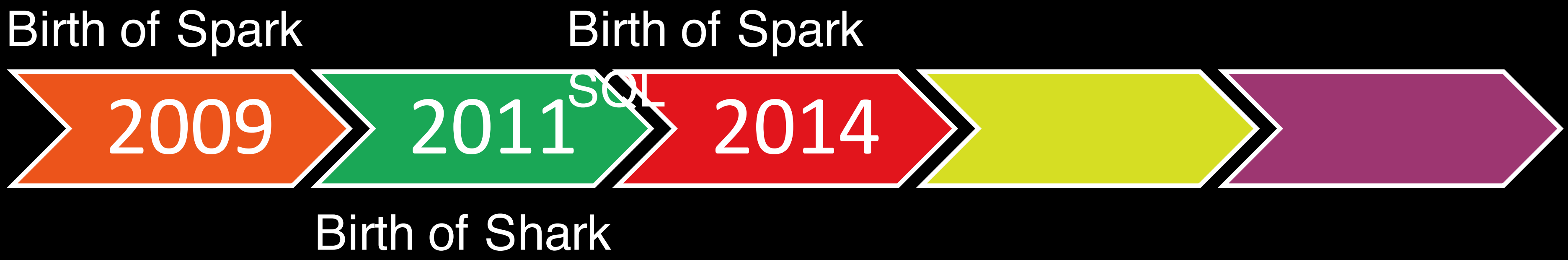
2009

Birth of Spark

2009

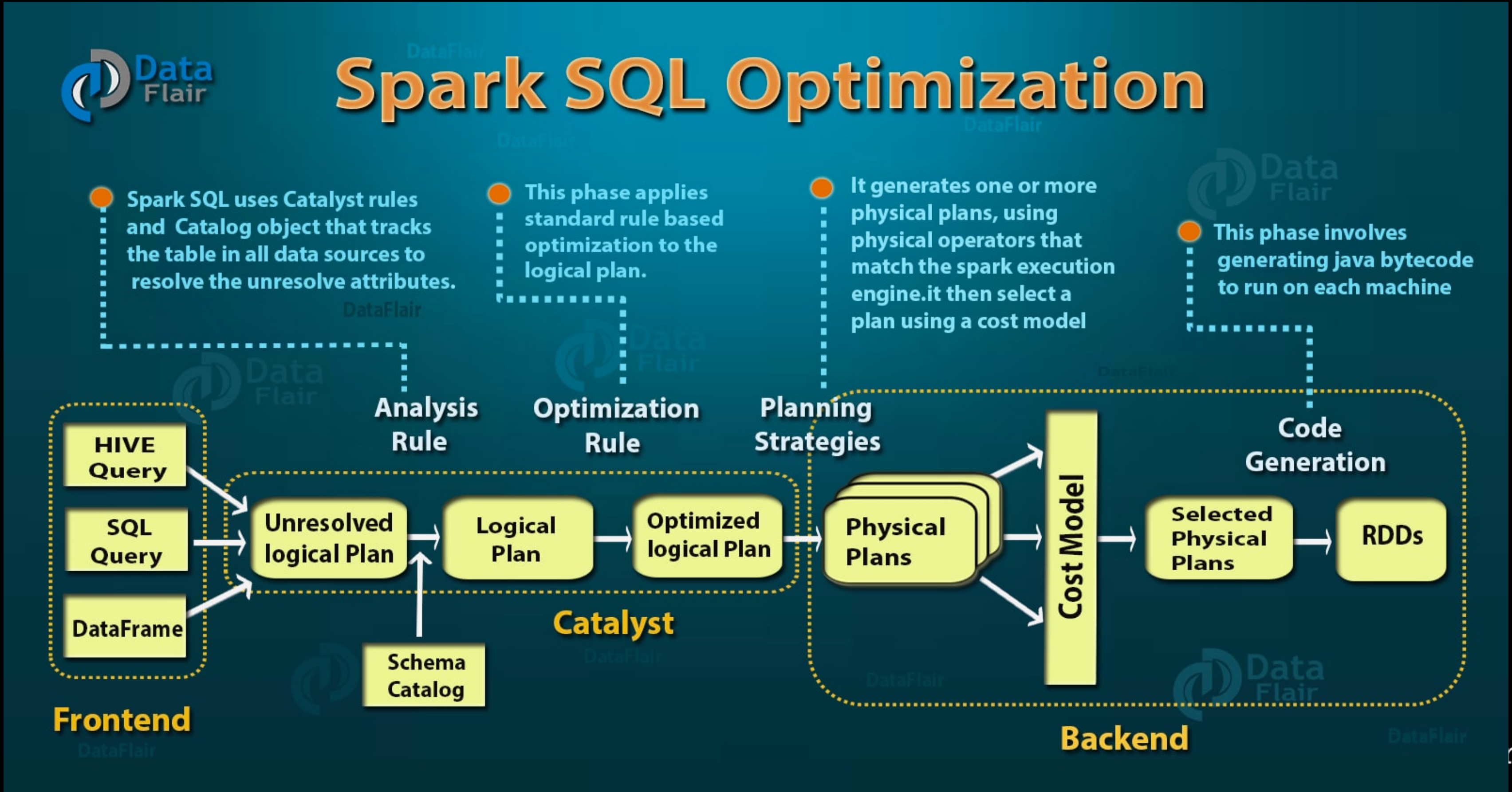
2011

Birth of Shark


Birth of Spark
2009
2011
Birth of Shark
Birth of Spark SQL
2014

databricks

Catalyst: an extensible optimizer



ricks

[SPARK-12032] [SQL] Re-order inner joins to do join with conditions f...

Browse files

…irst

Currently, the order of joins is exactly the same as SQL query, some conditions may not pushed down to the correct join, then those join will become cross product and is extremely slow.

This patch try to re-order the inner joins (which are common in SQL query

delay those that does not have conditions.

After this patch, the TPCDS query Q64/65 can run hundreds times faster.

cc marmbrus nongli

Author: Davies Liu <davies@databricks.com>

Closes #10073 from davies/reorder_joins.

master (#1) 2.0.0-preview

davies committed with davies on Dec 7, 2015

9c1212920a1d9000539b

Showing 3 changed files with 185 additions and 6 deletions.

Unified Split

Certain workloads can run hundreds times faster

~ 200 lines of changes

[SPARK-8992][SQL] Add pivot to dataframe api

Browse files

This adds a pivot method to the dataframe api.

Following the lead of cube and rollup this adds a Pivot operator that is translated into an Aggregate by the analyzer.

Currently the syntax is like:
~~courseSales.pivot(Seq($"year"), $"course", Seq("dotNET", "Java"), sum($"earnings"))~~

~~Would we be interested in the following syntax also/alternatively? and~~

```
    courseSales.groupBy($"year").pivot($"course", "dotNET", "Java").agg(sum($"earnings"))
    //or
    courseSales.groupBy($"year").pivot($"course").agg(sum($"earnings"))
```

Later we can add it to `SQLParser`, but as Hive doesn't support it we cant add it there, right?

~~Also what would be the suggested Java friendly method signature for this?~~

Author: Andrew Ray <ray.andrew@gmail.com>

Closes #7841 from aray/sql-pivot.

master (#3) 2.0.0-preview

aray committed with yhuai on Nov 11, 2015 f2d4b9c759710a195

Showing 6 changed files with 255 additions and 10 deletions.

Unified Split

~ 250 lines of changes

More Details about Catalyst

https://spark-summit.org/2017/events/a-deep-dive-into-spark-sqls-catalyst-optimizer/

Backend Execution Engine:

Compile query plan to RDD

Volcano—An Extensible and Parallel Query Evaluation System

Goetz Graefe

Abstract—To investigate the interactions of extensibility and parallelism in database query processing, we have developed a new dataflow query execution system called Volcano. The Volcano effort provides a rich environment for research and education in database systems design, heuristics for query optimization, parallel query execution, and resource allocation.

Volcano uses a standard interface between algebra operators, allowing easy addition of new operators and operator implementations. Operations on individual items, e.g., predicates, are imported into the query processing operators using *support functions*. The semantics of support functions is not prescribed; any data type including complex objects and any operation can be realized. Thus, Volcano is *extensible* with new operators, algorithms, data types, and type-specific methods.

Volcano includes two novel *meta-operators*. The *choose-plan*

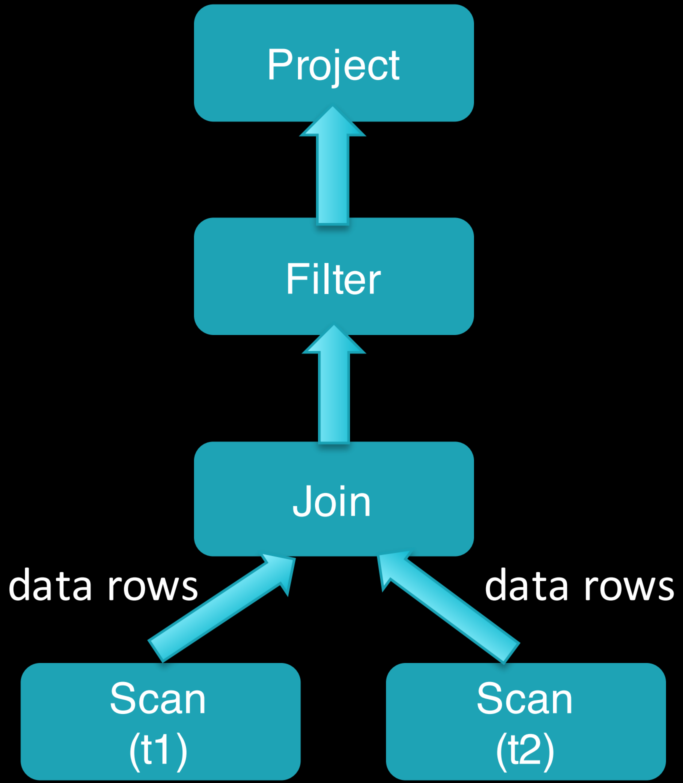
tem as it lacks features such as a user-friendly query language, a type system for instances (record definitions), a query optimizer, and catalogs. Because of this focus, Volcano is able to serve as an experimental vehicle for a multitude of purposes, all of them open-ended, which results in a combination of requirements that have not been integrated in a single system before. First, it is modular and extensible to enable future research, e.g., on algorithms, data models, resource allocation, parallel execution, load balancing, and query optimization heuristics. Thus, Volcano provides an infrastructure for experimental research rather than a final research prototype in itself. Second, it

G. Graefe, **Volcano— An Extensible and Parallel Query Evaluation System**, *In IEEE Transactions on Knowledge and Data Engineering 1994*

databricks

Volcano Iterator Model

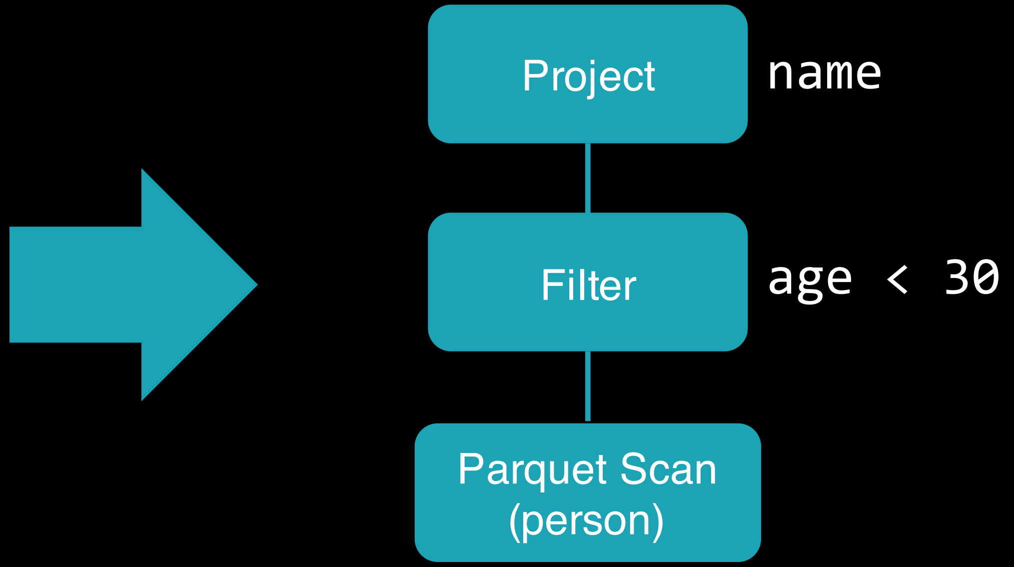
- Standard for 30 years: almost all databases do it
- Each operator is an "iterator" that consumes records from its input operator



databricks

How Spark SQL Run Queries

```
SELECT name
FROM person
WHERE age < 30
```



How Spark SQL Run Queries

```
class ParquetScan {
  def execute(): RDD[Row] = {
     ...
  }
}
```

How Spark SQL Run Queries

```scala
class FilterExec(condition: Expression) {
  def execute(): RDD[Row] = {
    child.execute().mapPartitions { input =>
      val predicate: Row => Boolean = row => {
        condition.eval(row)
      }
      input.filter(predicate)
    }
  }
}
```

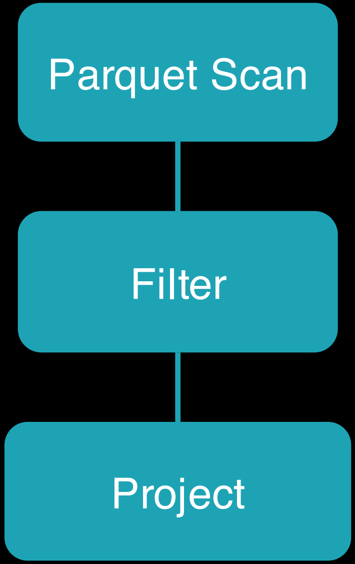
databricks

How Spark SQL Run Queries

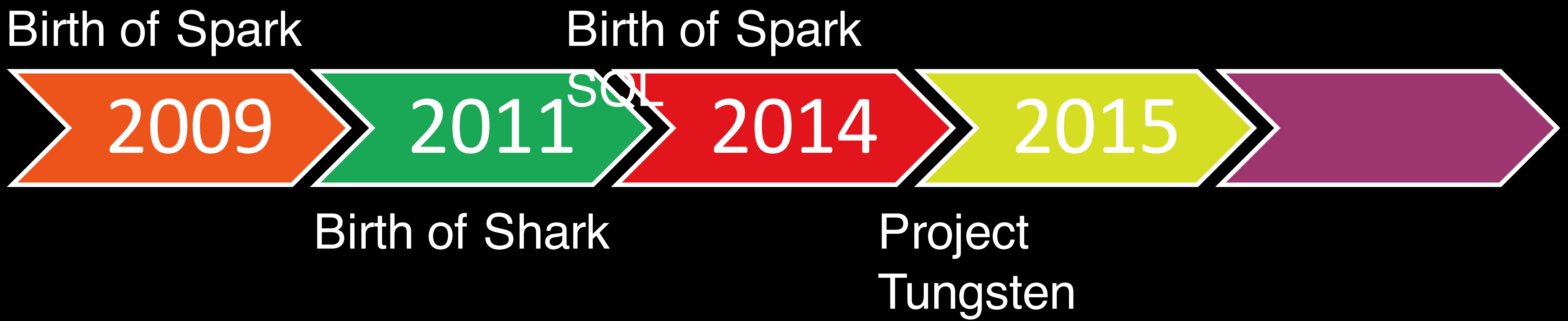
```
class ProjectExec(projectList: Seq[Expression]) {
  def execute(): RDD[Row] = {
    child.execute().mapPartitions { input =>
      val project: Row => Row = ...
      input.map(project)
    }
  }
}
```

databricks

How Spark SQL Run Queries




```scala
val tableScan: RDD[Row] = ...
tableScan.mapPartitions { input =>
  val predicate: Row => Boolean = ...
  input.filter(predicate)
}.mapPartitions { input =>
  val project: Row => Row = ...
  input.map(project)
}
```


Birth of Spark
2009
Birth of Spark SQL
2011
Birth of Shark
2014
2015
Project Tungsten

databricks

Making Sense of Performance in Data Analytics Frameworks

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

Abstract

There has been much research devoted to improving the performance of data analytics frameworks, but comparatively little effort has been spent systematically identifying the performance bottlenecks of these systems. In this paper, we develop blocked time analysis, a methodology for quantifying performance bottlenecks in distributed computation frameworks, and use it to analyze the Spark framework's performance on two SQL benchmarks and a production workload. Contrary to our expectations, we find that (i) CPU (and not I/O) is often the bottleneck, (ii) improving network performance can improve job completion time by a median of at most 2%, and (iii) the causes of most stragglers can be identified.

This paper makes two contributions towards a more comprehensive understanding of performance. First, we develop a methodology for analyzing end-to-end performance of data analytics frameworks; and second, we use our methodology to study performance of two SQL benchmarks and one production workload. Our results run counter to all three of the aforementioned mantras.

The first contribution of this paper is *blocked time analysis*, a methodology for quantifying performance bottlenecks. Identifying bottlenecks is challenging for data analytics frameworks because of pervasive parallelism: jobs are composed of many parallel tasks, and each task uses pipelining to parallelize the use of network, disk, and CPU. One task may be bottlenecked on different

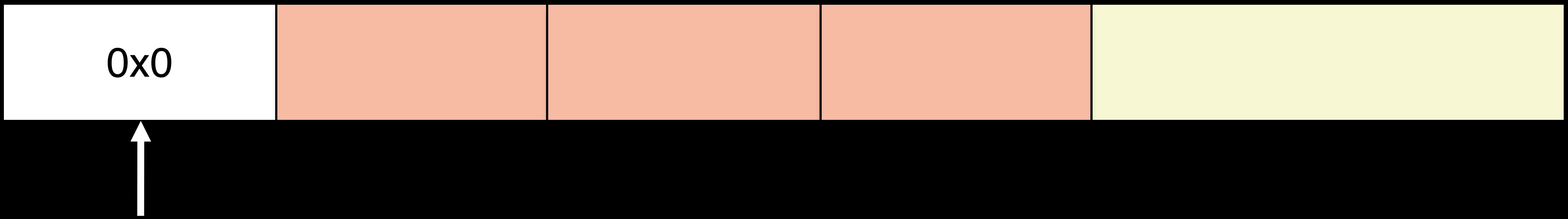
Kay Ousterhout, Making Sense of Performance in Data Analytics Frameworks,
In NSDI on Networked Systems Design 2015

databricks

Tungsten Format:
efficient binary format for Row

Efficient Binary Format

(123, “data”, “bricks”)

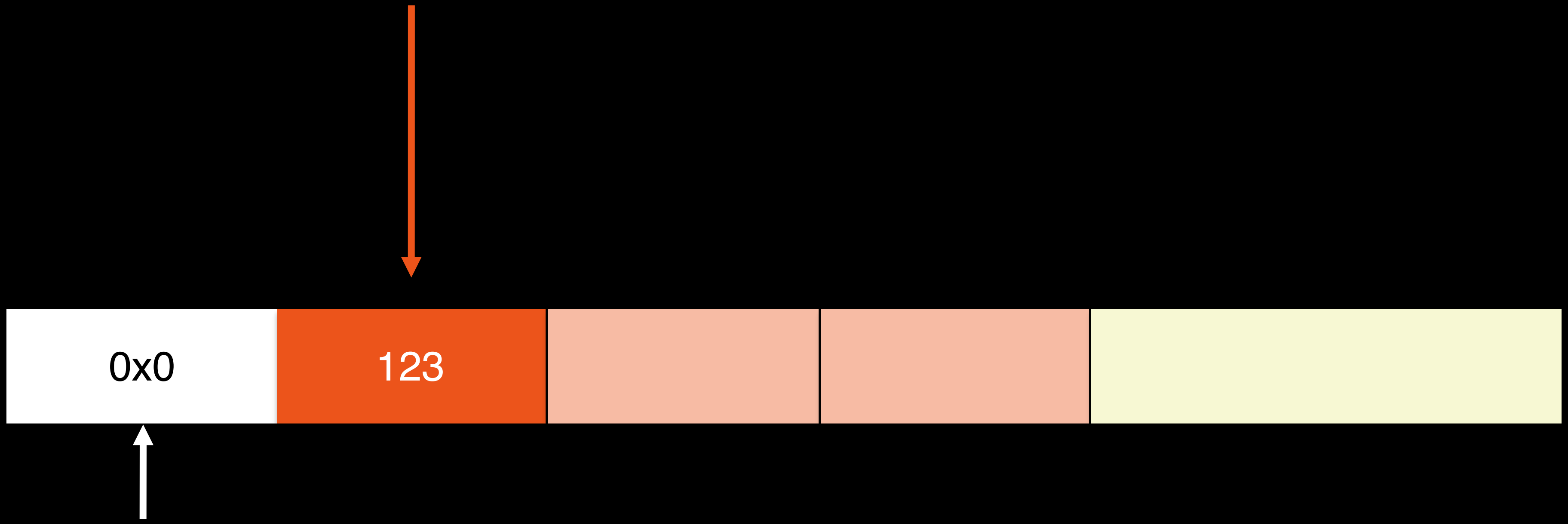


null tracking

databricks

Efficient Binary Format

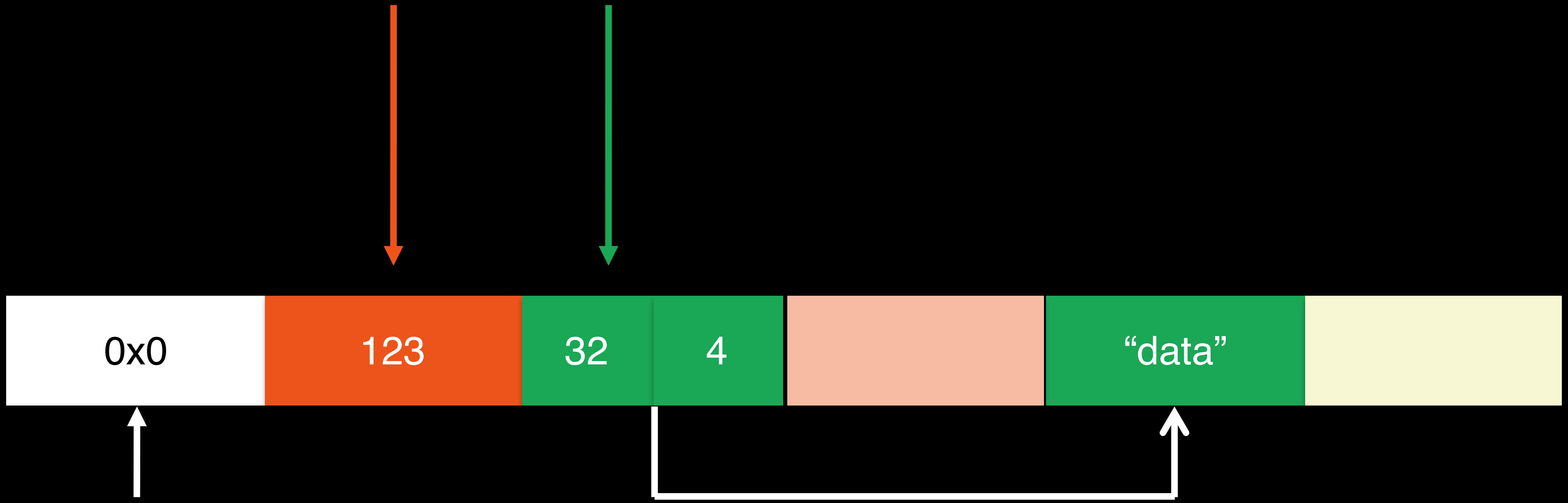
(123, “data”, “bricks”)



null tracking

Efficient Binary Format

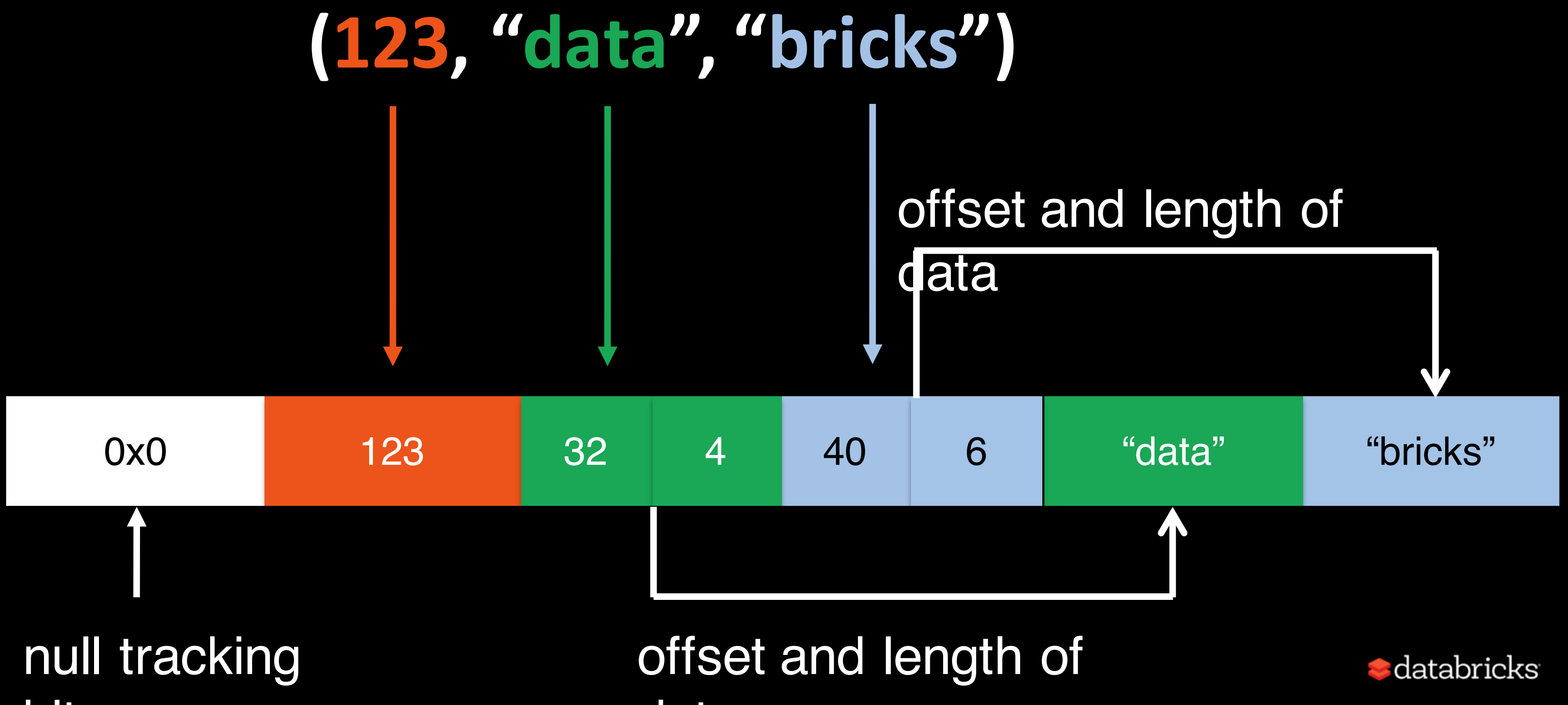
(123, “data”, “bricks”)



null tracking

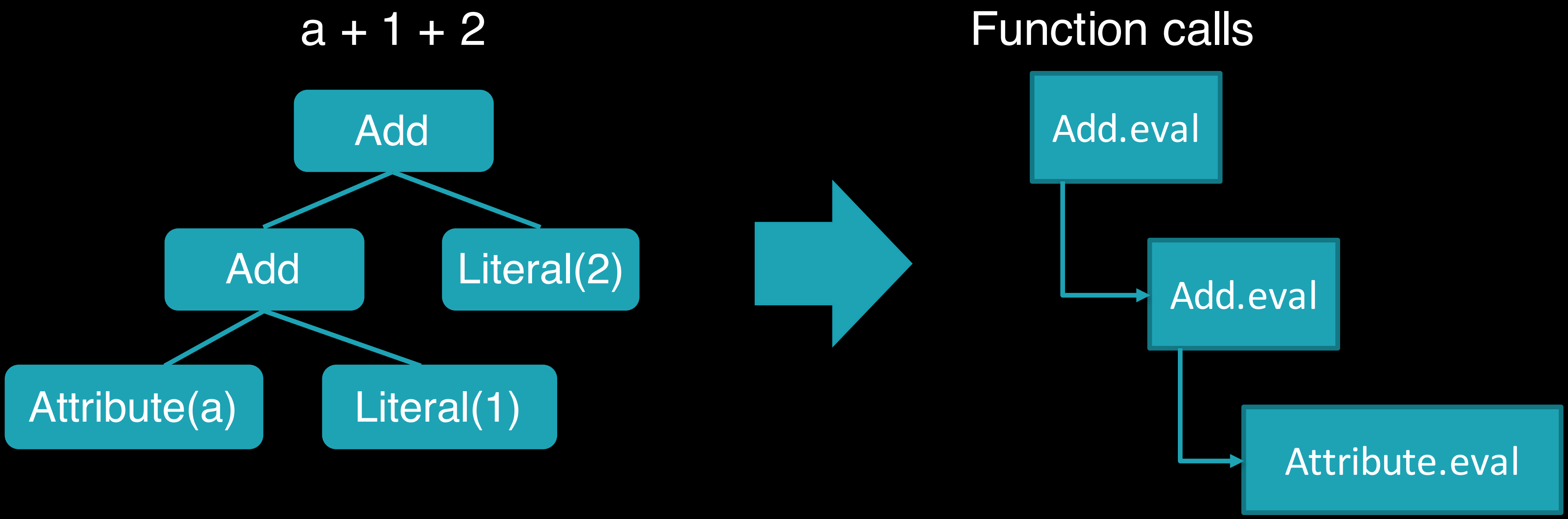
offset and length of

Efficient Binary Format



Expression Code Generation: evaluate expressions faster

How to Evaluate Expression



databricks

Expression Code Generation

DataFrame Code / SQL

```
df.where(df("year") > 2015)
```

Catalyst Expressions

```
GreaterThan(year#234, Literal(2015))
```

Low-level Java code

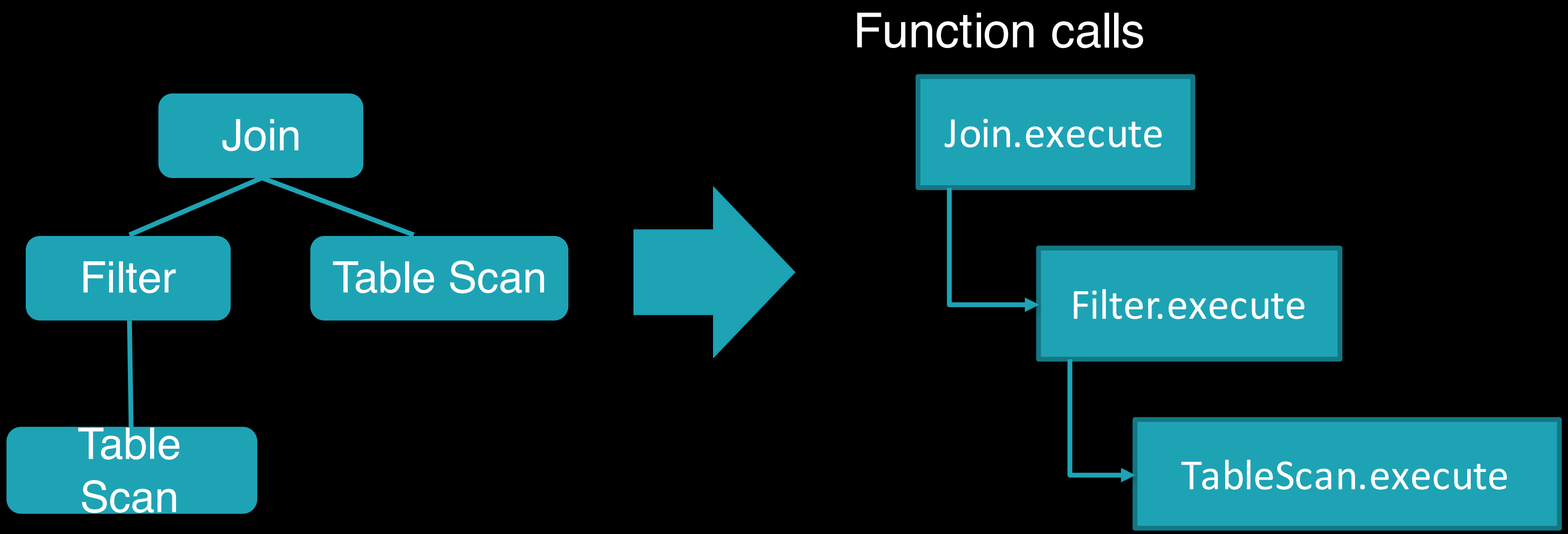
```
boolean filter(Object baseObject) {
    int offset = baseOffset + bitSetWidthInBytes + 3*8L;
    int value = Platform.getInt(baseObject, offset);
    return value34 > 2015;
}
```

JVM ***intrinsic*** JIT-ed to pointer arithmetic

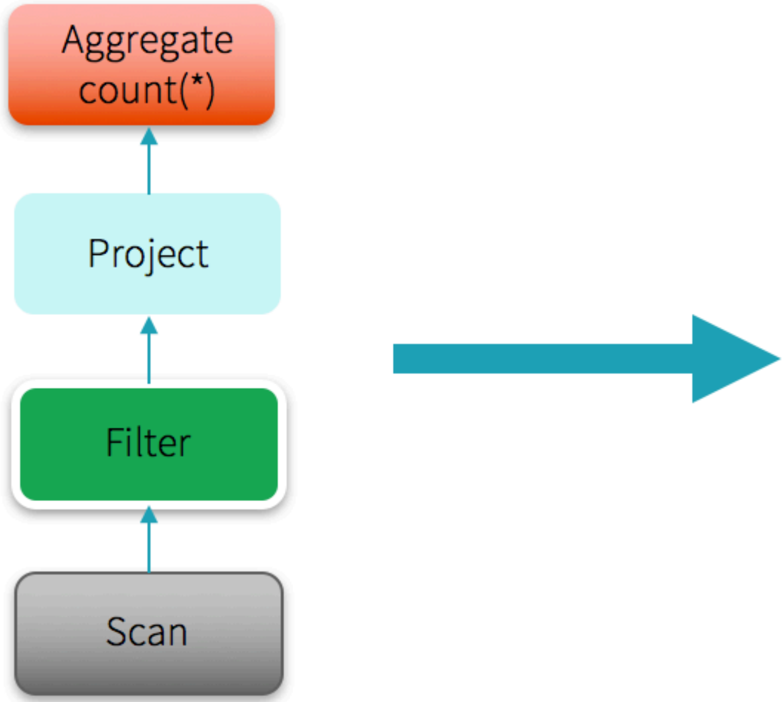
Whole Stage CodeGen: plan-level code generation

databricks

How to Evaluate Query Plan



Generate code like handwritten

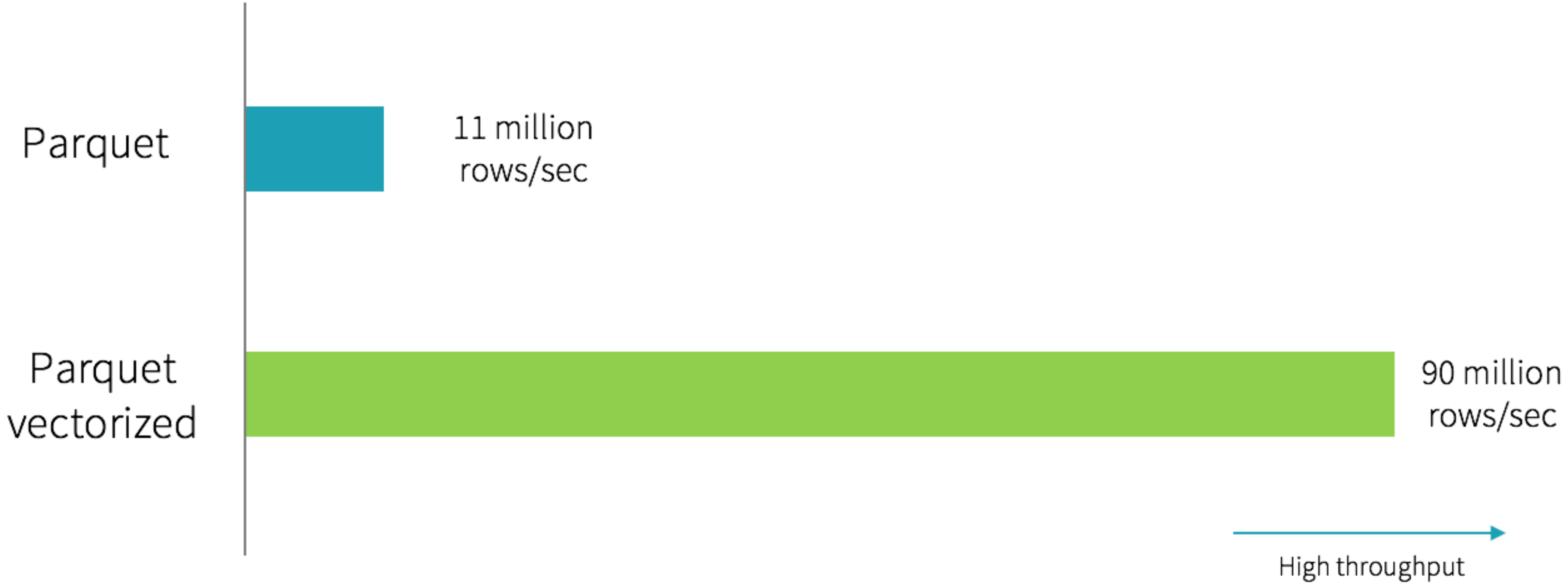



```
long count = 0;
for (ss_item_sk in store_sales) {
  if (ss_item_sk == 1000) {
    count += 1;
  }
}
```

Scan Vectorization: load data faster

databricks

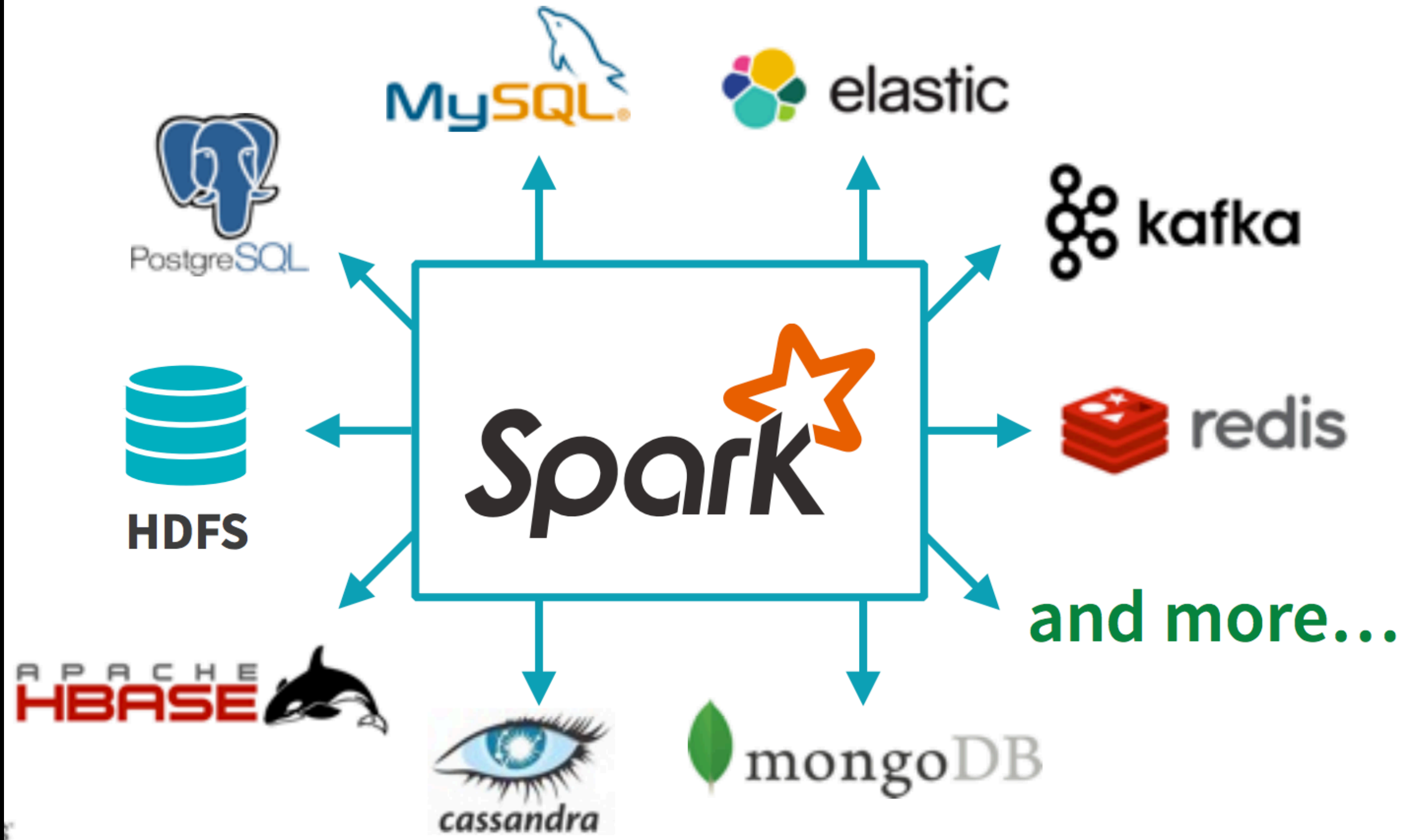
Vectorized Parquet Reader



Scan Vectorization

- more efficient to read columnar data with vectorization.
- more likely for JVM to generate SIMD instructions.
- lazy decompression.
- ......

Current Spark: Not a Database



databricks

2009 — Birth of Spark

2011 — Birth of Shark

2014 — Birth of Spark SQL

2015 — Project Tungsten

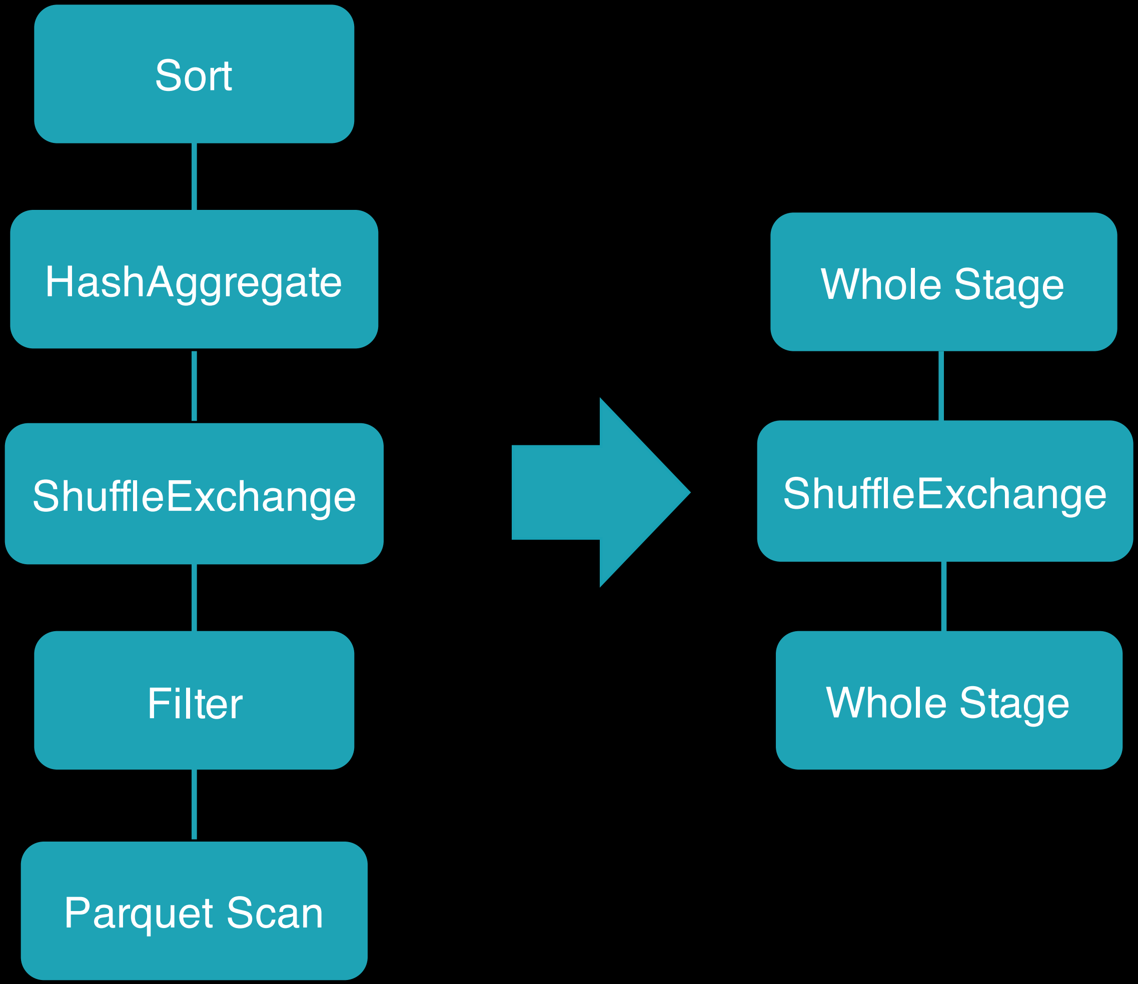
2017~ — What's next?

databricks

SPARK-15689: Data Source V2

more powerful data source

Complete Vectorization: for sink and shuffle


Sort
HashAggregate
ShuffleExchange
Filter
Parquet Scan
Whole Stage
ShuffleExchange
Whole Stage

Native Code Generation

Why Java?

- Because Spark is running on JVM ☺
- Run generated code directly
- Easy to share data

Why not Java?

- not good for vectorization (no explicit SIMD support)
- performance depends on JIT a lot
- Generated code is hard to read (too verbose)
- Alternatives: LLVM, Weld

Thank You