

Correctness and Performance of Apache Spark SQL

Spark + AI Summit, London

October 4, 2018



About us



BOGDAN GHIT

Databricks, Software Engineer

- SQL performance optimizations

IBM T.J. Watson, Research Intern

- Bid advisor for cloud spot markets

Delft University of Technology, PhD in Computer Science

- Resource management in datacenters
- Performance of Spark, Hadoop



NICOLAS POGGI

Databricks, Performance Engineer

- Spark benchmarking

Barcelona Supercomputing - Microsoft Research Centre

- Lead researcher ALOJA project
- New architectures for Big Data

BarcelonaTech (UPC), PhD in Computer Architecture

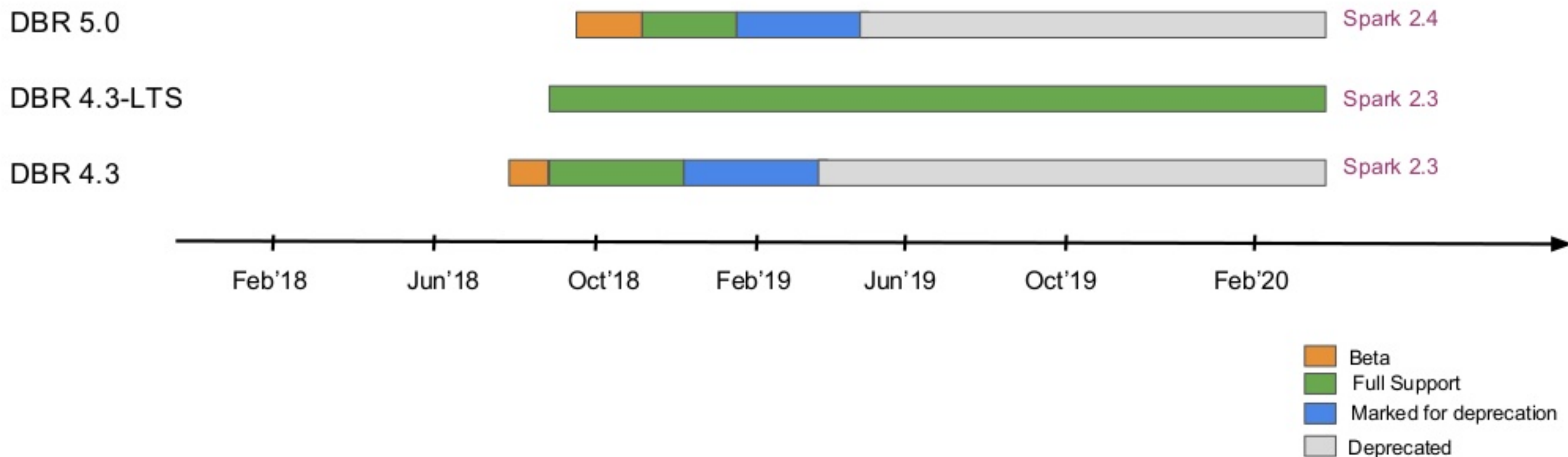
- Autonomic resource manager for the cloud
- Web customer modeling

Databricks ecosystem



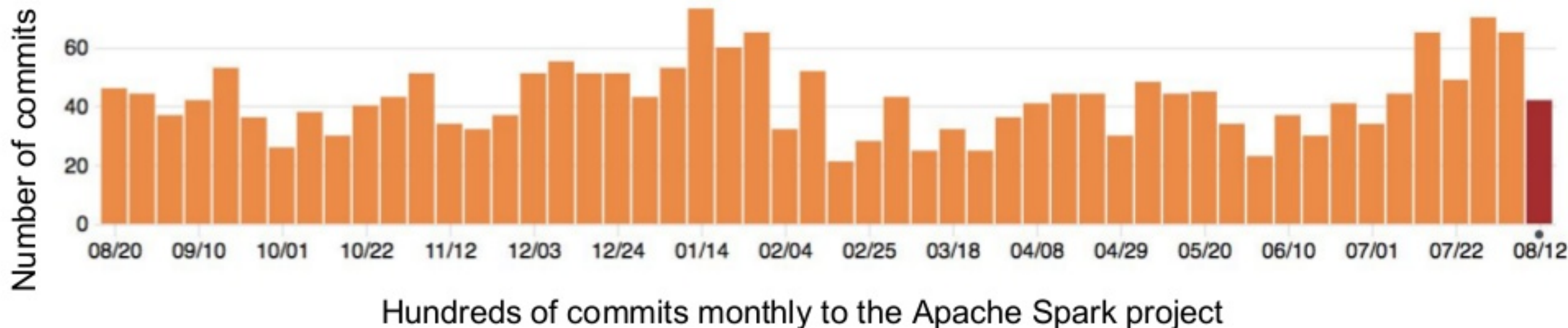
 databricks®

Databricks runtime (DBR) releases



Our goal is to make releases **automatic** and **frequent**

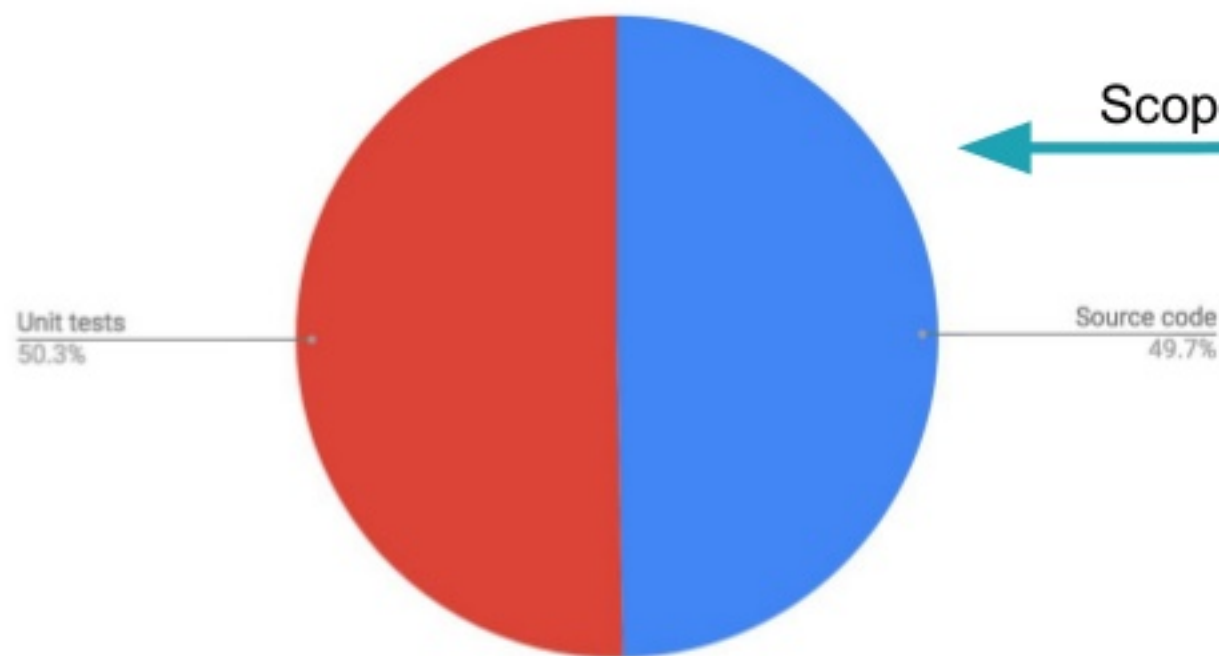
Apache Spark contributions



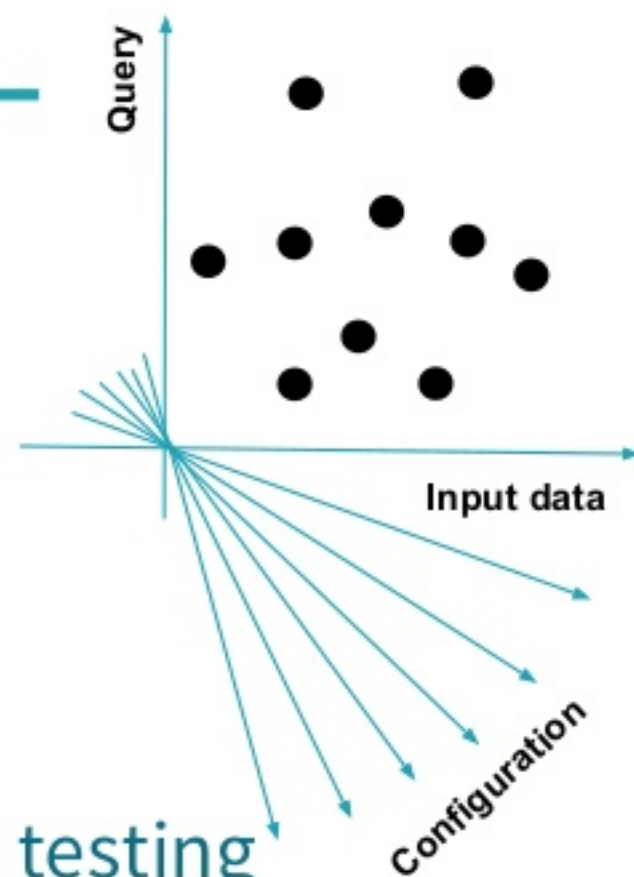
At this pace of development, **mistakes** are bound to happen

Where do these contributions go?

SQL Core

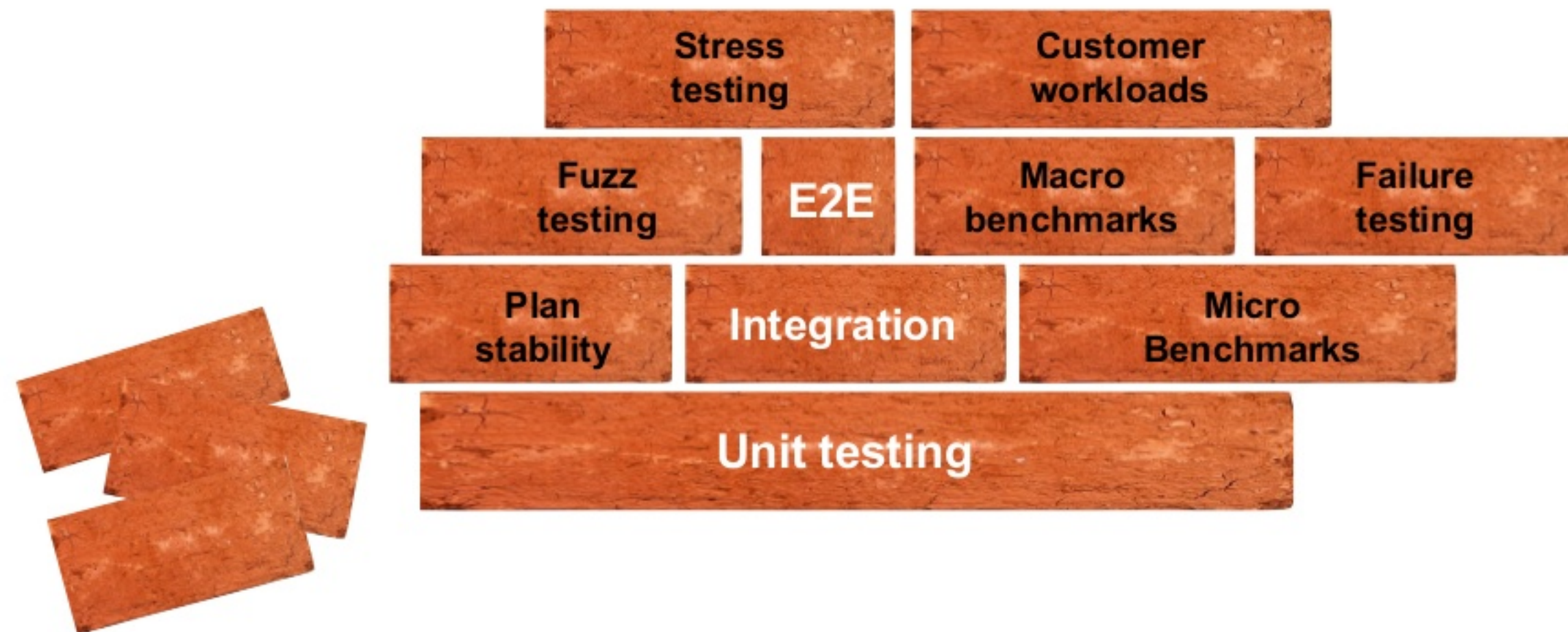


Over 200 built-in functions



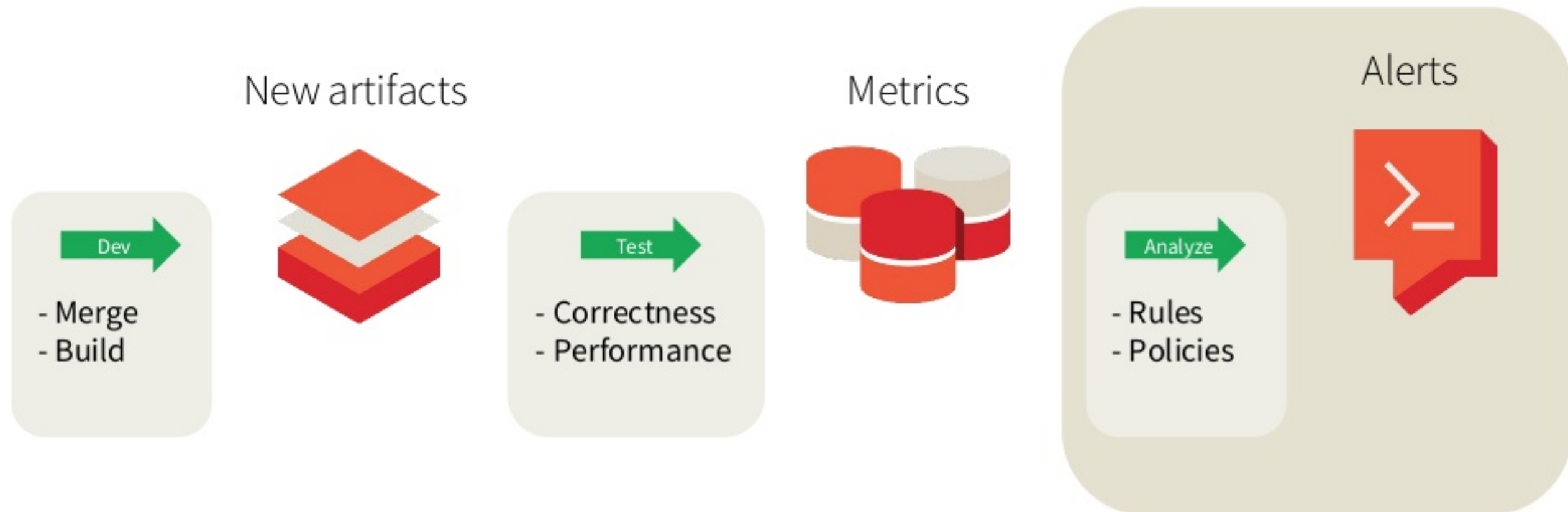
Developers put a significant **engineering effort** in testing

Yet another brick in the wall

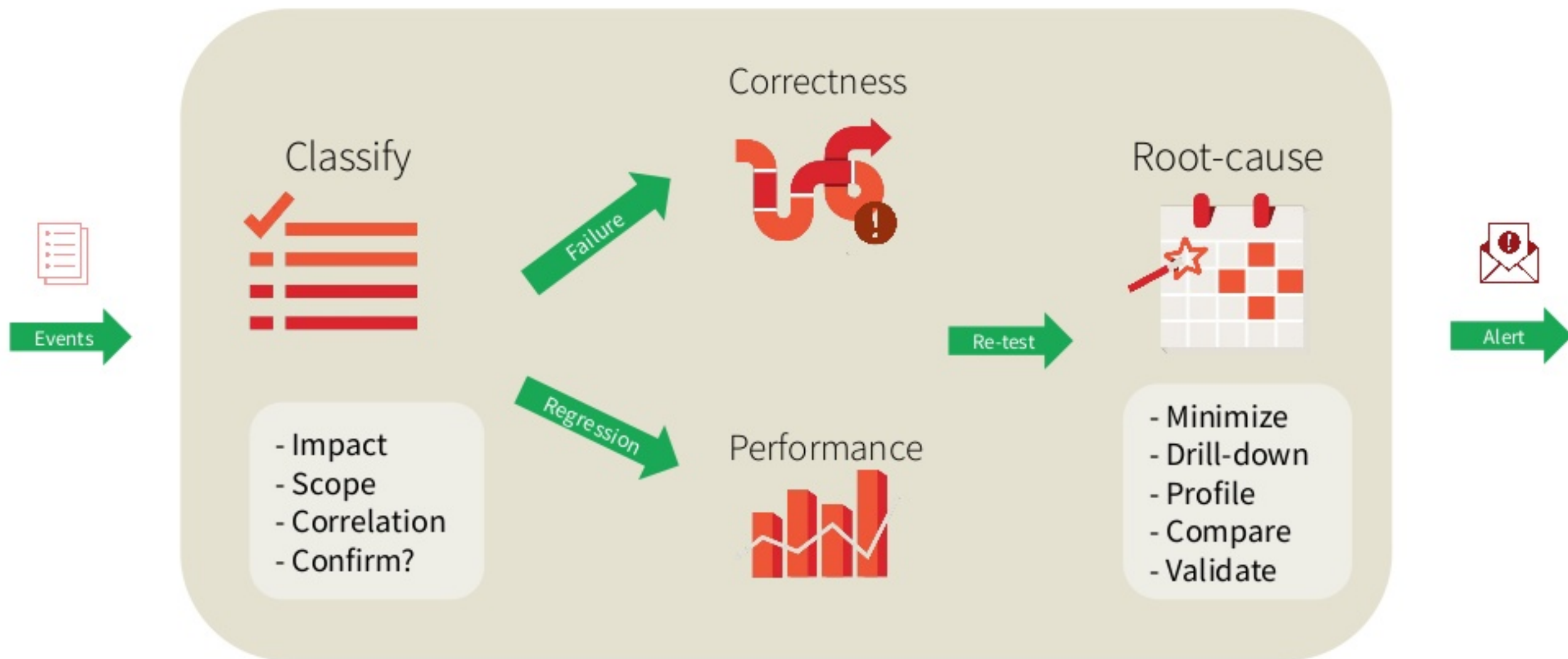


Unit testing *is not enough* to guarantee correctness and performance

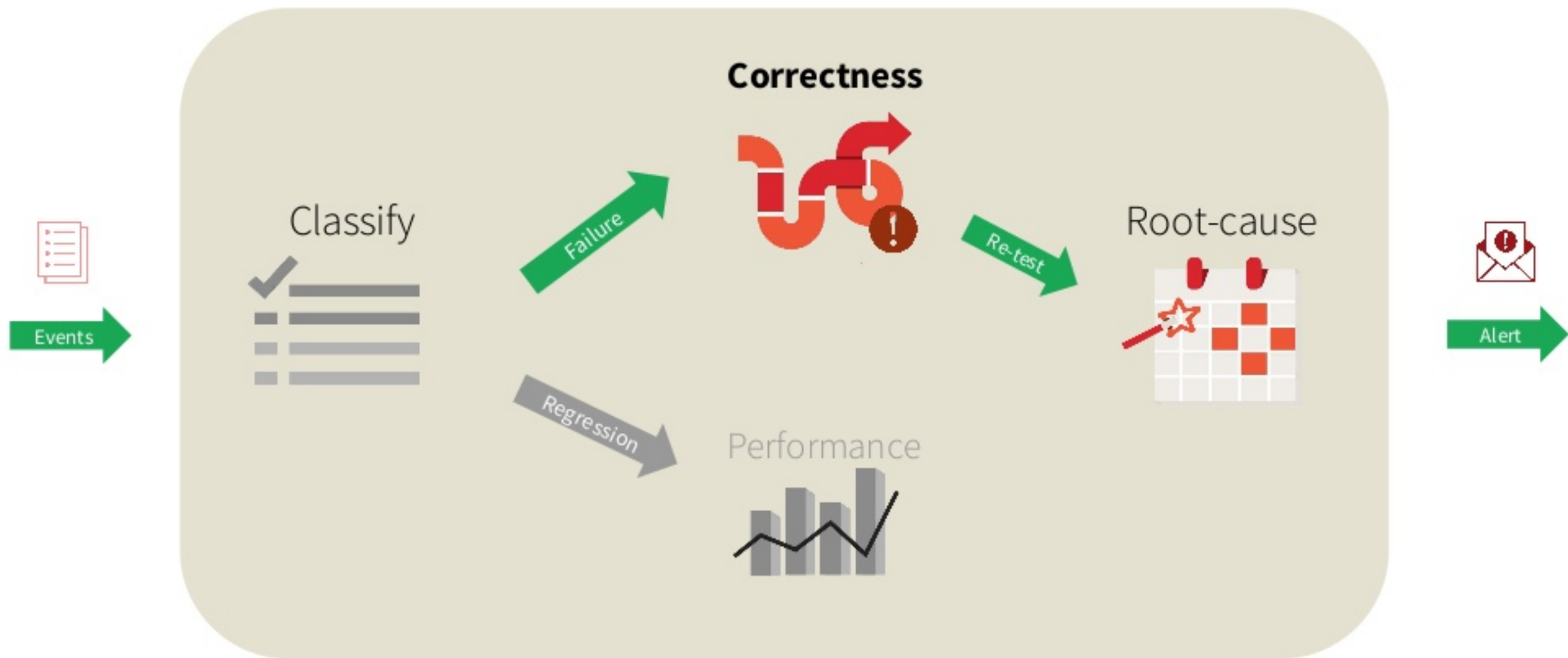
Continuous Integration pipeline



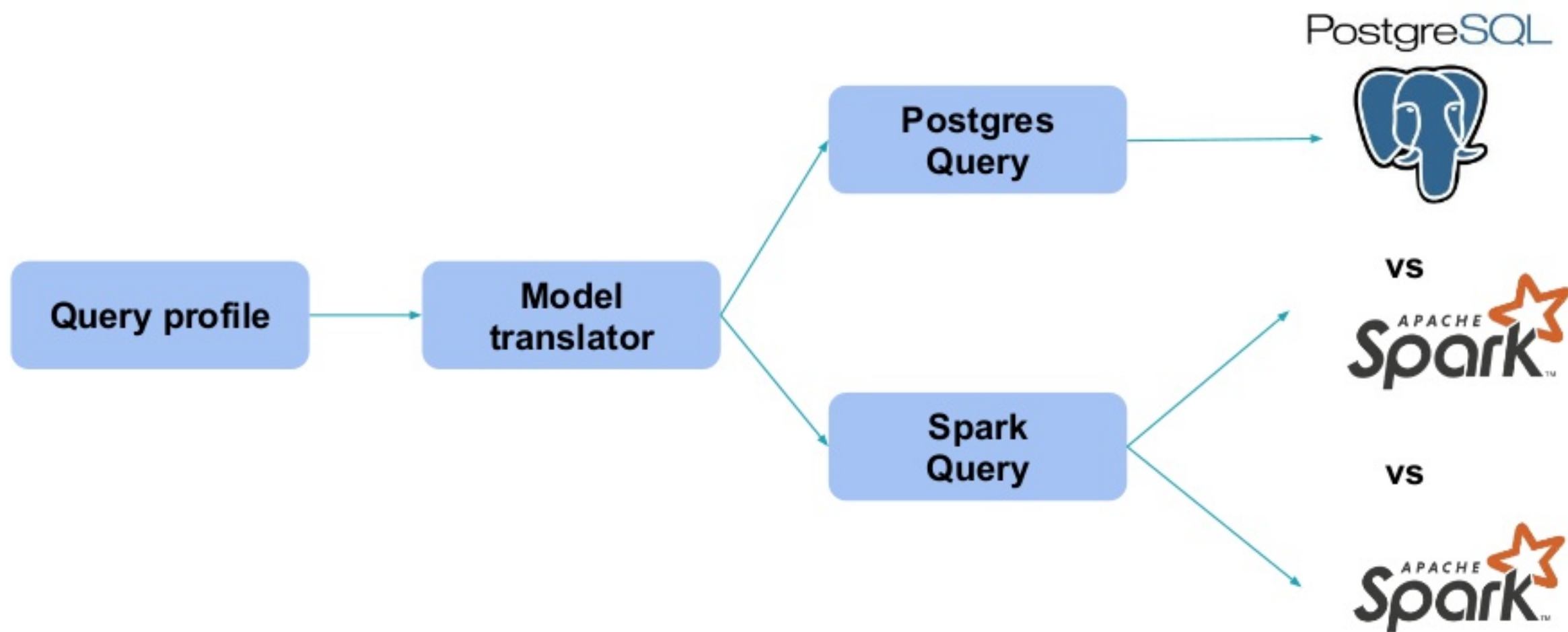
Classification and alerting



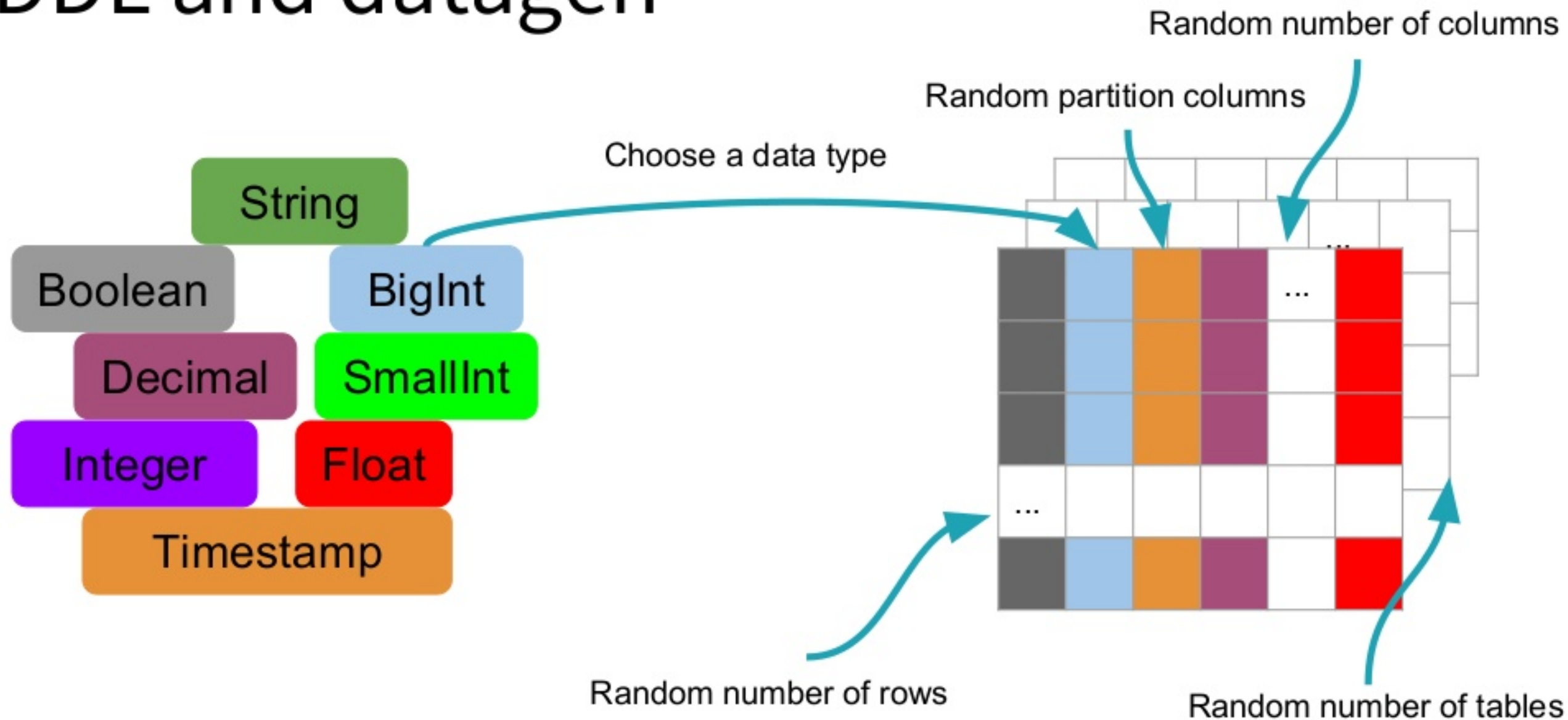
Correctness



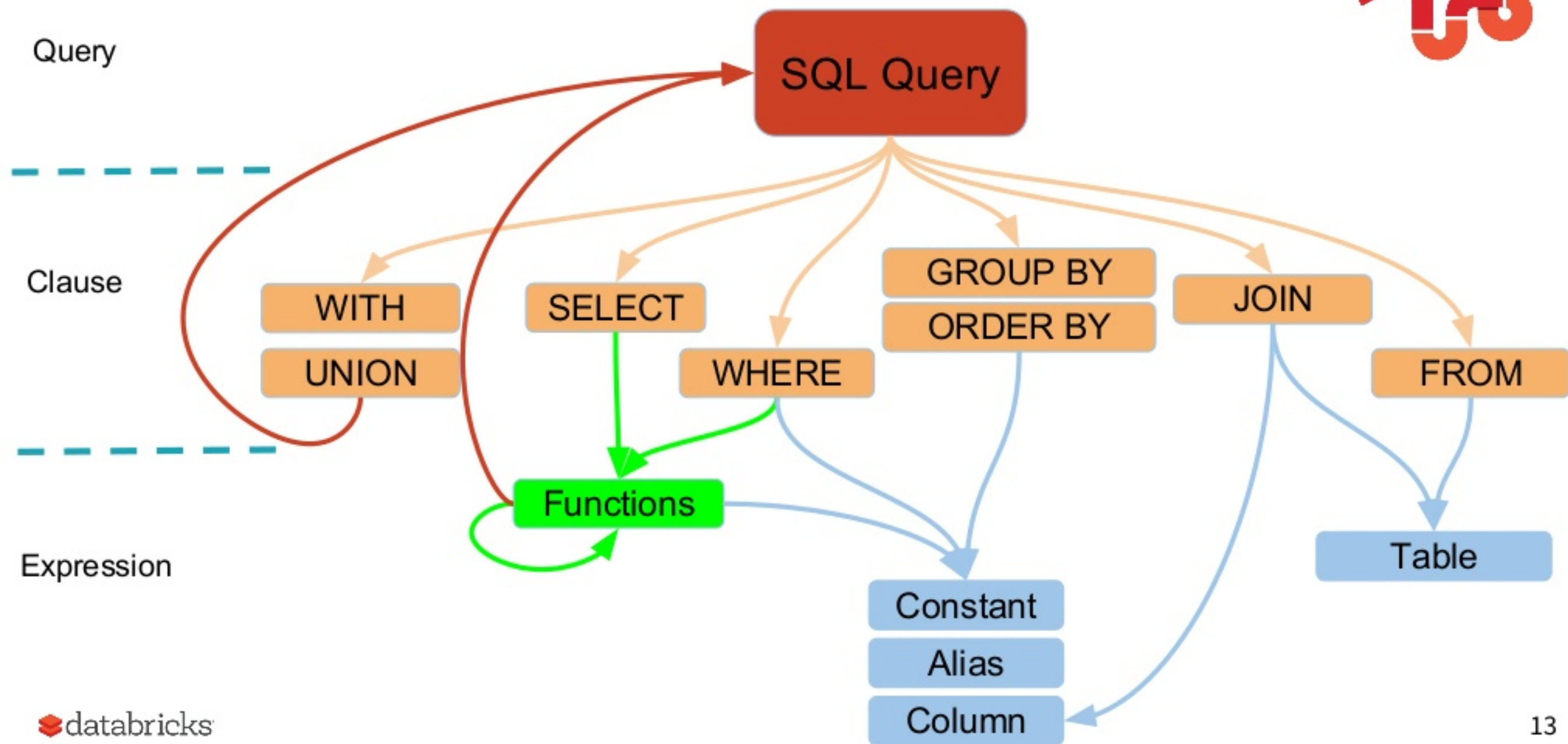
Random query generation



DDL and datagen



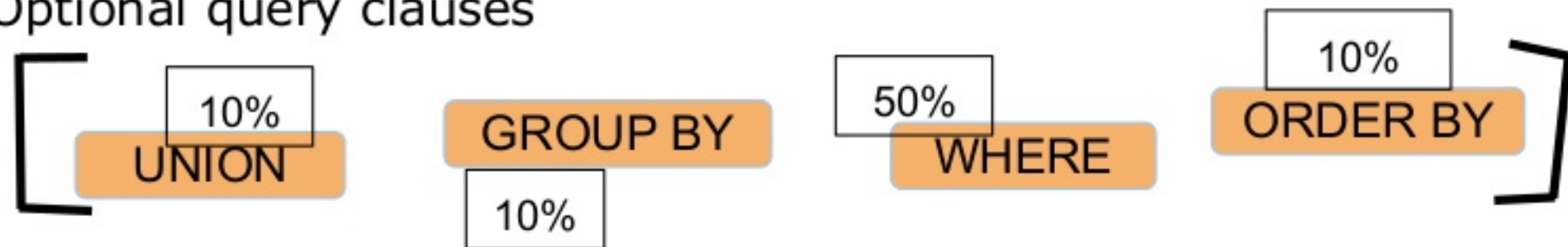
Recursive query model



Probabilistic query profile

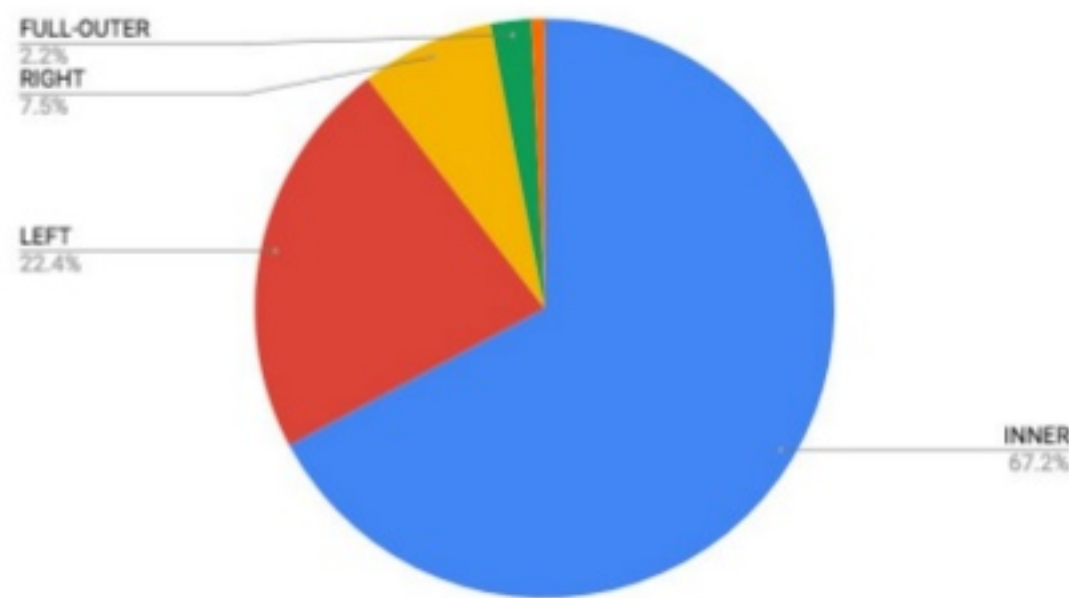
Independent weights

- Optional query clauses



Inter-dependent weights

- Join types
- Select functions



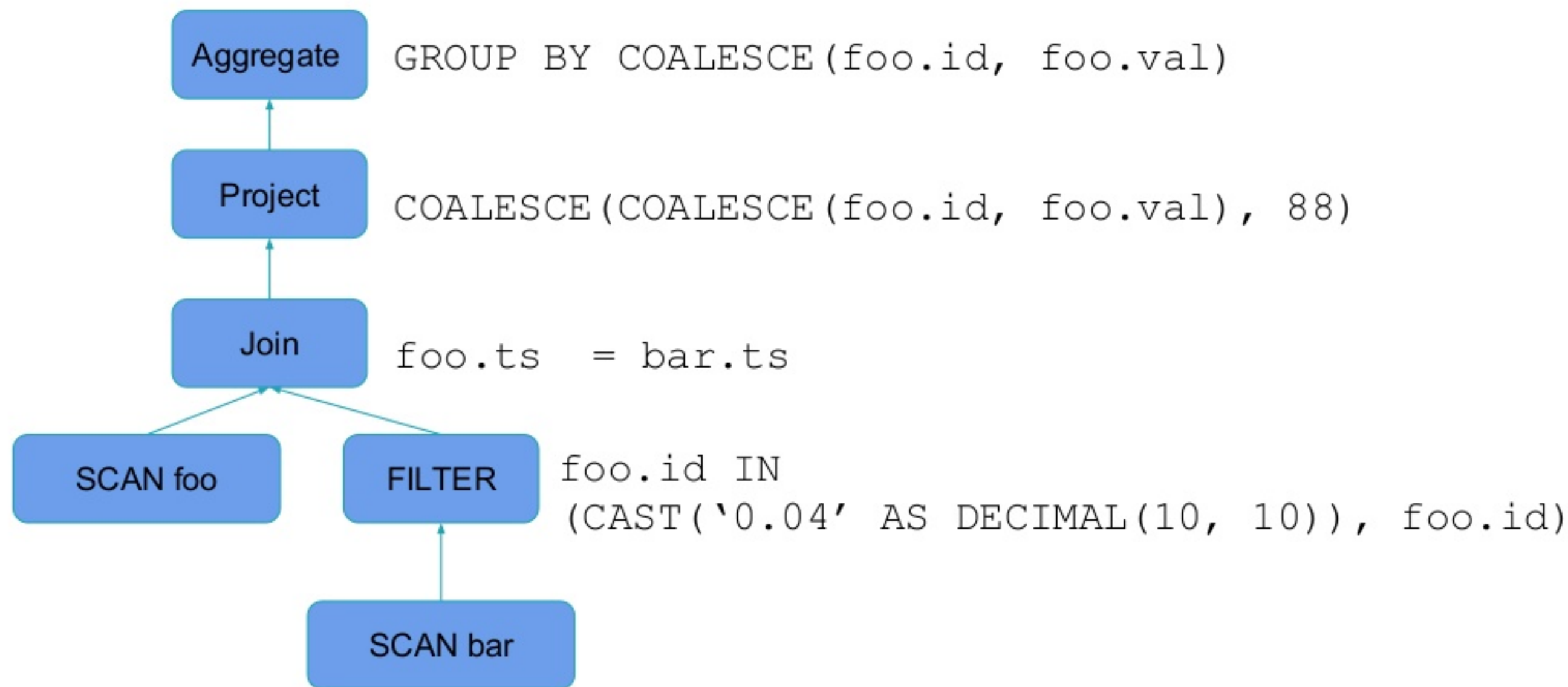
Coalesce flattening (1/4)

```
SELECT COALESCE(t2.smallint_col_3, t1.smallint_col_3, t2.smallint_col_3) AS int_col,  
       IF(NULL, VARIANCE(COALESCE(t2.smallint_col_3, t1.smallint_col_3, t2.smallint_col_3)),  
          COALESCE(t2.smallint_col_3, t1.smallint_col_3, t2.smallint_col_3)) AS int_col_1,  
       STDDEV(t2.double_col_2) AS float_col,  
       COALESCE(MIN((t1.smallint_col_3) - (COALESCE(t2.smallint_col_3, t1.smallint_col_3,  
          t2.smallint_col_3))), COALESCE(t2.smallint_col_3, t1.smallint_col_3, t2.smallint_col_3),  
          COALESCE(t2.smallint_col_3, t1.smallint_col_3, t2.smallint_col_3)) AS int_col_2  
FROM table_4 t1  
INNER JOIN table_4 t2 ON (t2.timestamp_col_7) = (t1.timestamp_col_7)  
WHERE (t1.smallint_col_3) IN (CAST('0.04' AS DECIMAL(10,10)), t1.smallint_col_3)  
GROUP BY COALESCE(t2.smallint_col_3, t1.smallint_col_3, t2.smallint_col_3)
```

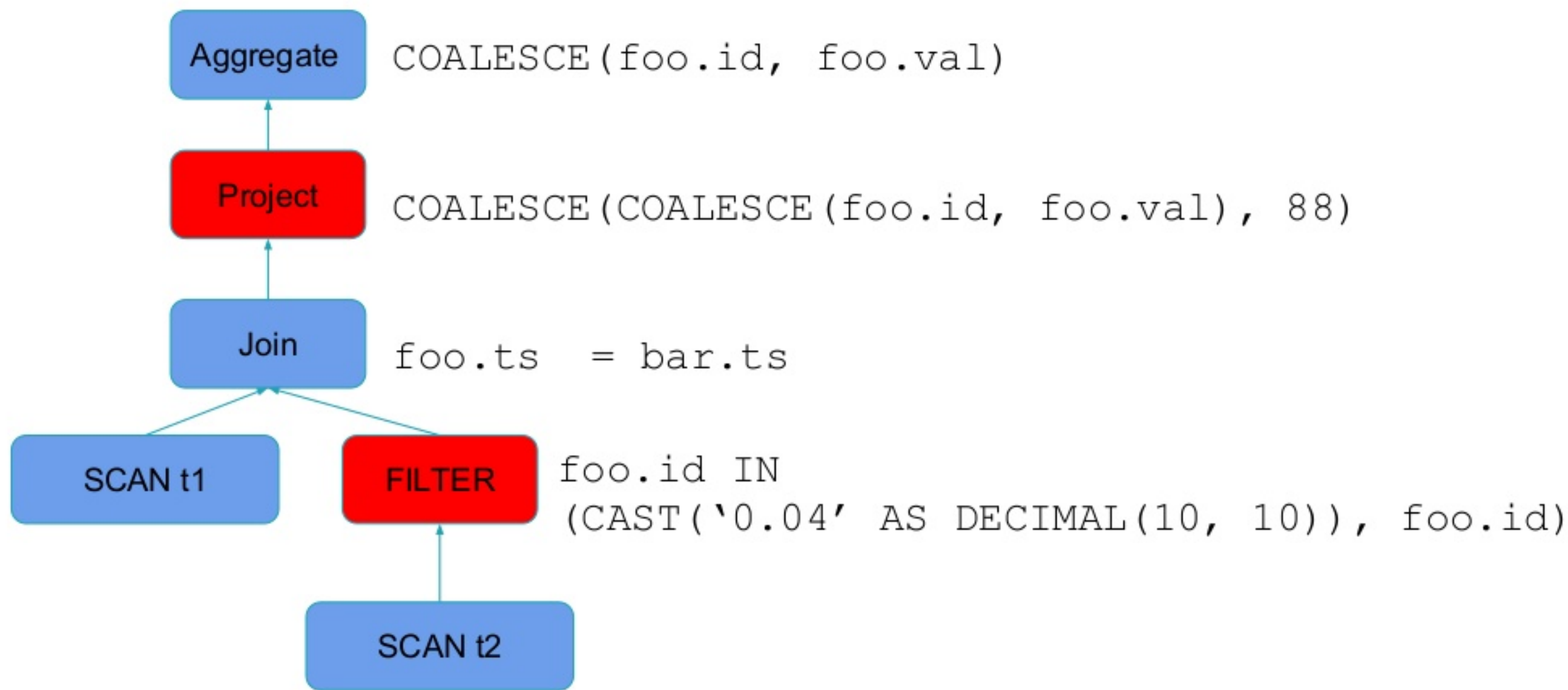
Small dataset with 2 tables of 5x5 size
Within 10 randomly generated queries

Error: Operation is in ERROR_STATE

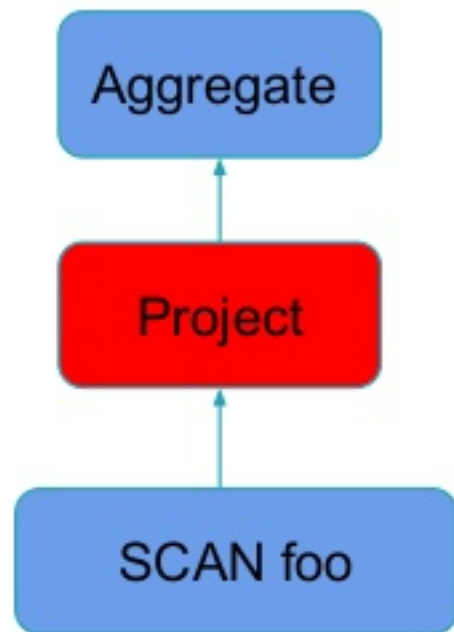
Coalesce flattening (2/3)



Coalesce flattening (3/4)



Coalesce flattening (4/4)



Minimized query:

```
SELECT
    COALESCE (COALESCE (foo.id, foo.val), 88)
FROM foo
GROUP BY
    COALESCE (foo.id, foo.val)
```

Analyzing the error

- The optimizer flattens the nested coalesce calls
- The SELECT clause doesn't contain the GROUP BY expression
- Possibly a problem with any GROUP BY expression that can be optimized

Lead function (1/3)

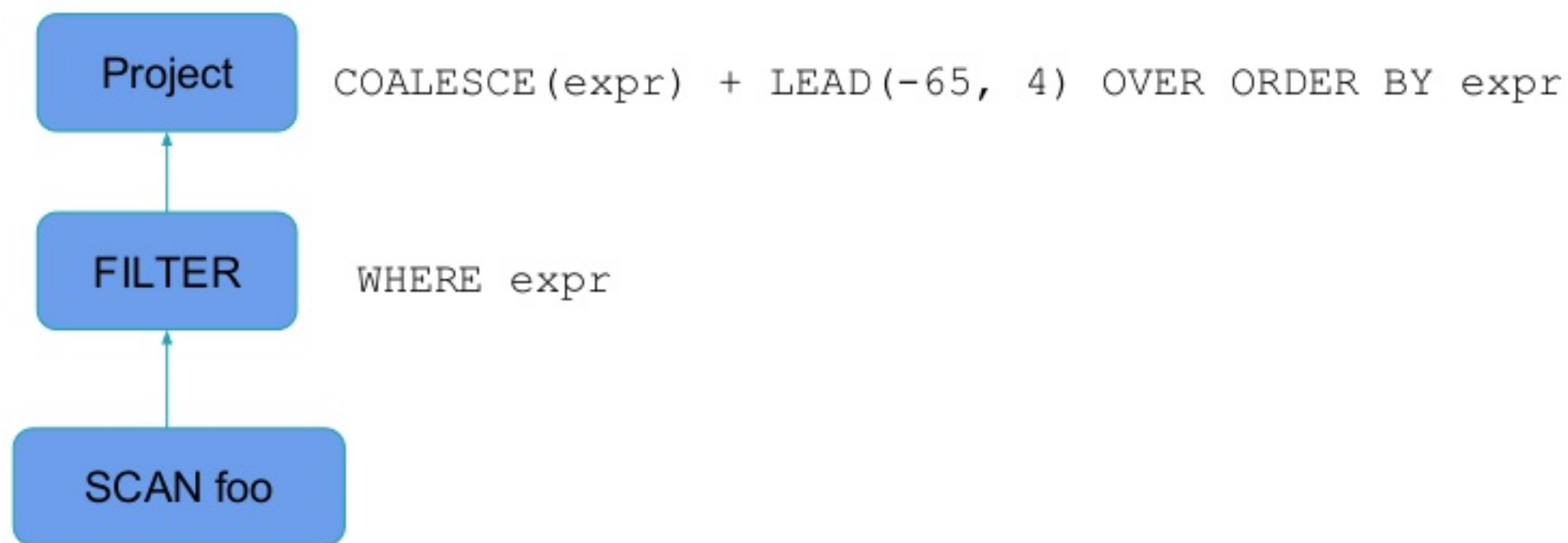
```
SELECT (t1.decimal0803_col_3) / (t1.decimal0803_col_3) AS decimal_col,  
       CAST(696 AS STRING) AS char_col, t1.decimal0803_col_3,  
       (COALESCE(CAST('0.02' AS DECIMAL(10,10)),  
                 CAST('0.47' AS DECIMAL(10,10)),  
                 CAST('-0.53' AS DECIMAL(10,10)))) +  
       (LEAD(-65, 4) OVER (ORDER BY (t1.decimal0803_col_3) / (t1.decimal0803_col_3),  
                          CAST(696 AS STRING))) AS decimal_col_1,  
       CAST(-349 AS STRING) AS char_col_1  
  
FROM table_16 t1  
WHERE (943) > (889)
```

Error: Column 4 in row 10 does not match:

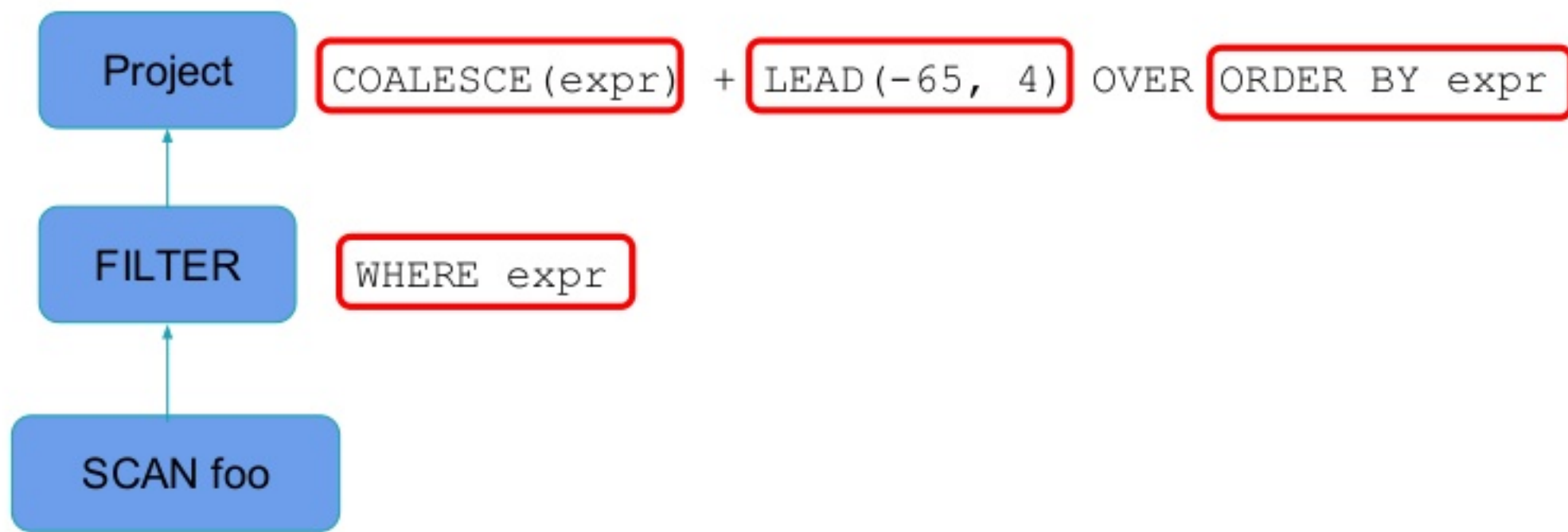
[1.0, 696, -871.81, <<-64.98>>, -349] SPARK row

[1.0, 696, -871.81, <<None>>, -349] POSTGRES SQL row

Lead function (2/3)



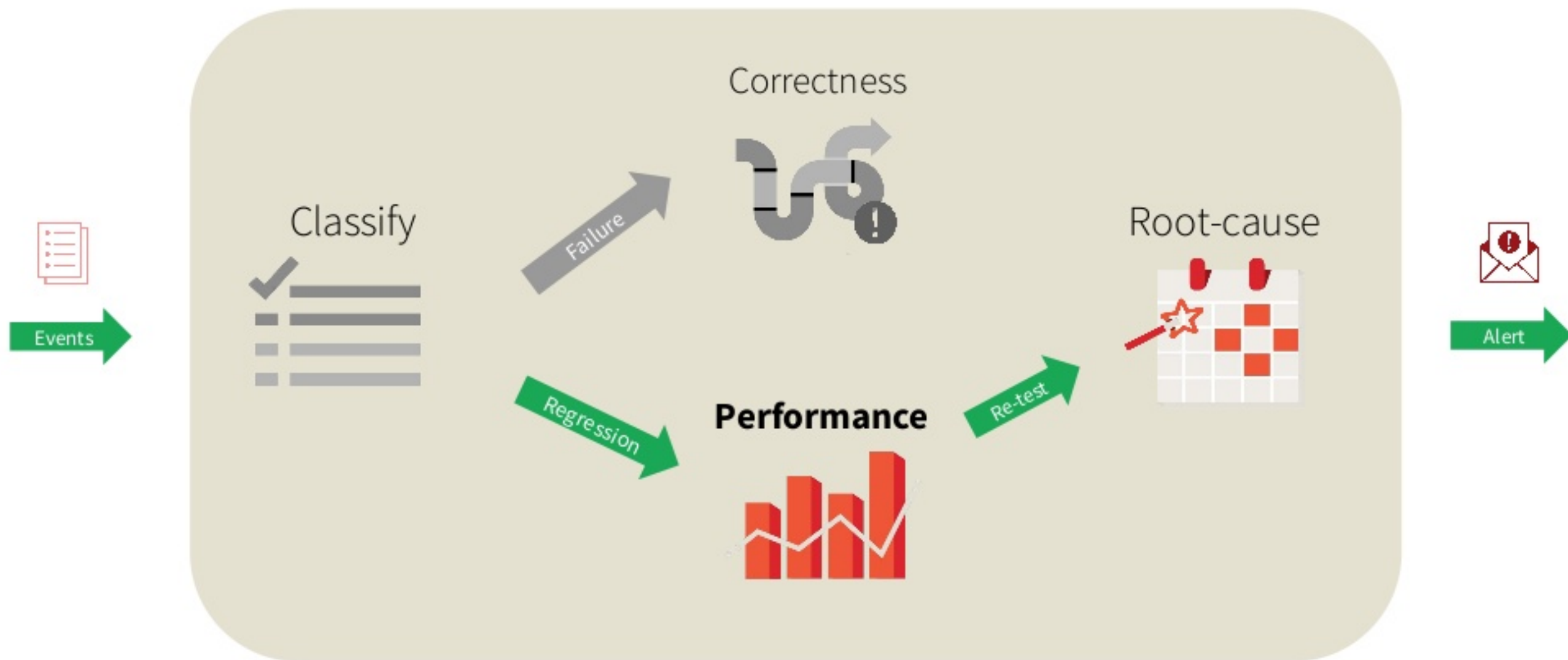
Lead function (3/3)



Analyzing the error

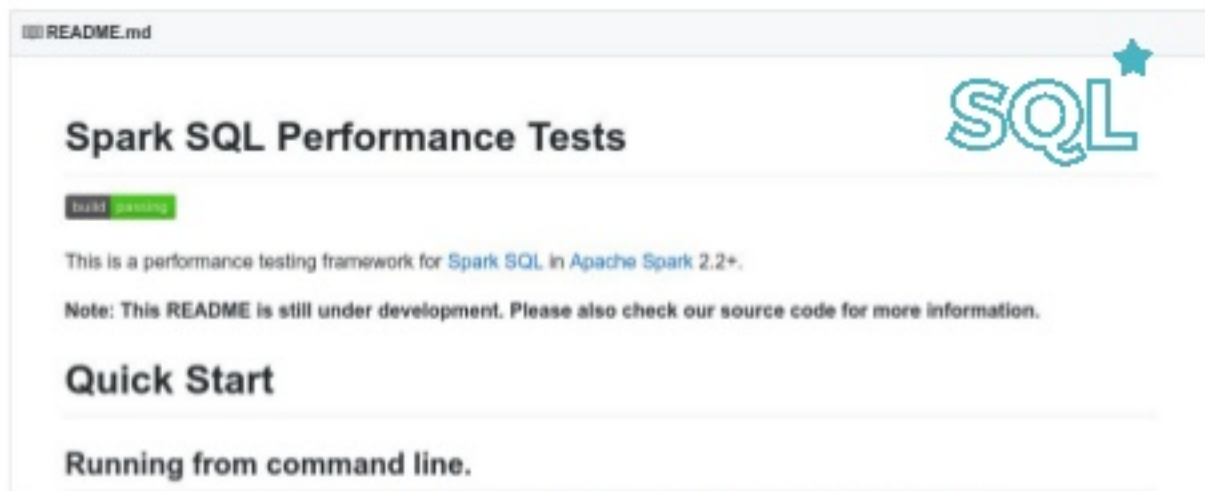
- Using constant input values breaks the behaviour of the LEAD function
- SC-16633: <https://github.com/apache/spark/pull/14284>

Performance

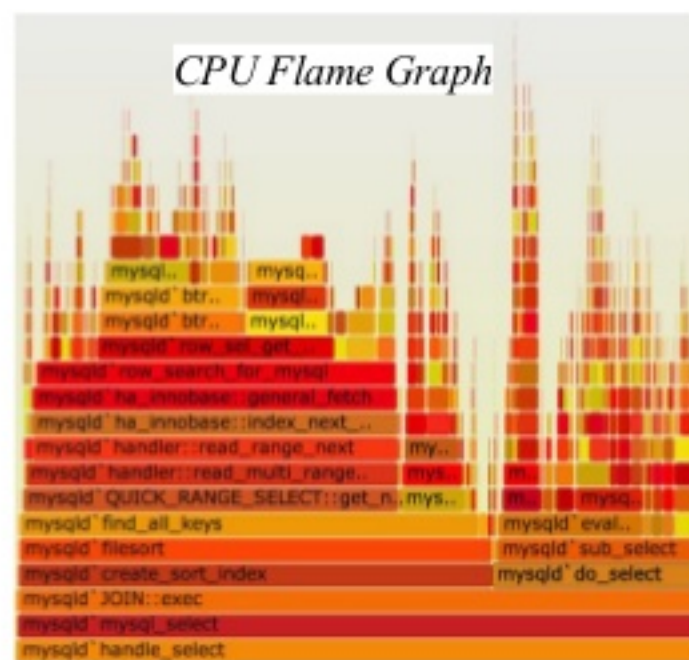


Benchmarking tools

- We use [spark-sql-perf](#) public library for TPC workloads
 - Provides datagen and import scripts
 - local, cluster, S3
 - Dashboards for **analyzing results**
- The Spark micro benchmarks
- And the [async-profiler](#)
 - to produce **flamegraphs**



<https://github.com/databricks/spark-sql-perf>



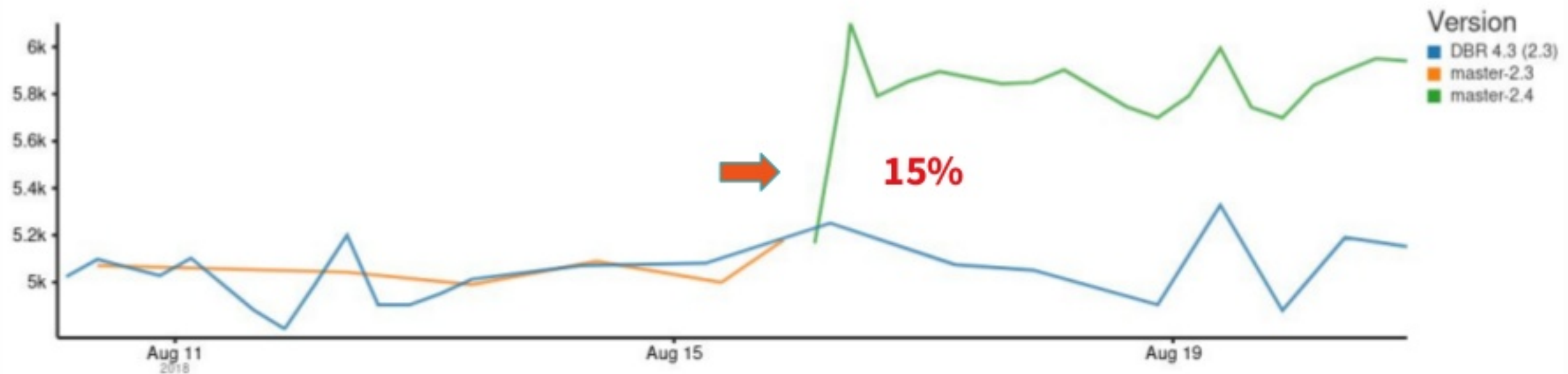
DBR 5.0-beta (v2.4) performance tracking ~~tracking~~ journey

daysBack : 30

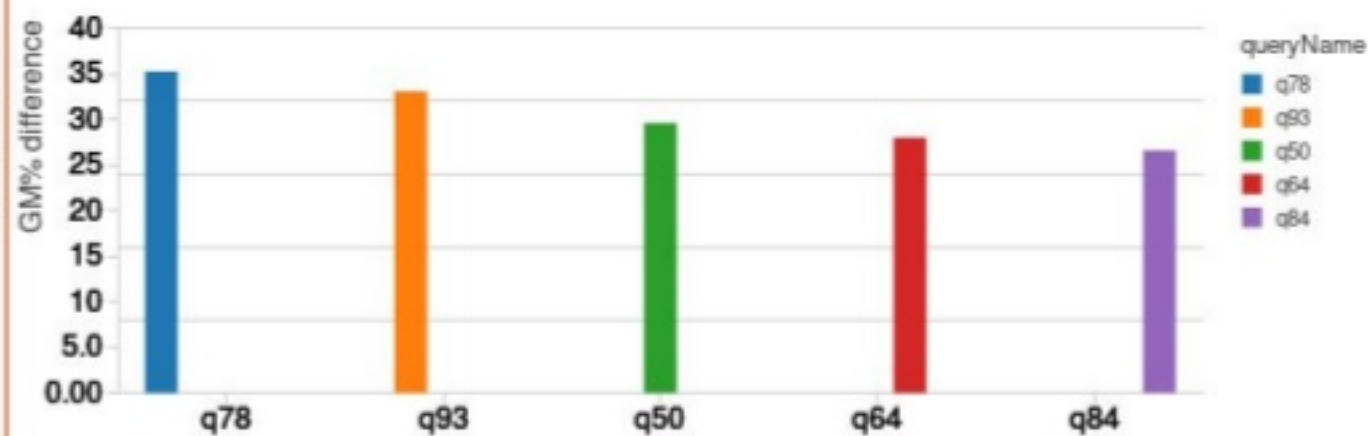
daysTargetBack : 30

targetVersion : master-2.4

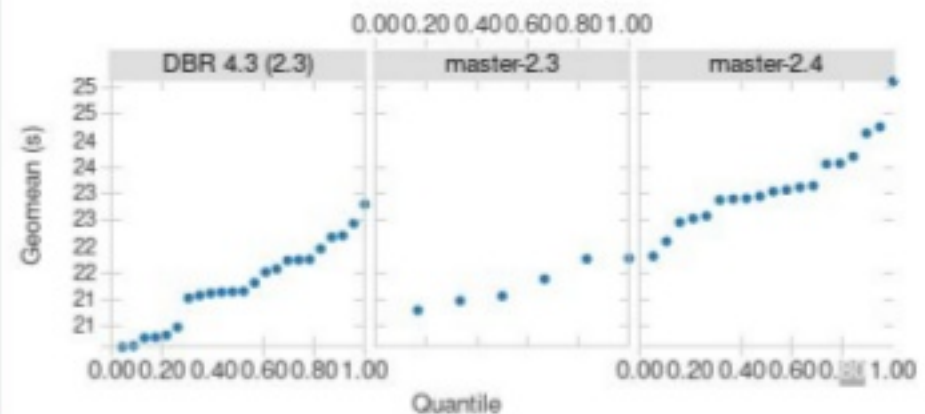
Time series of runs by type (10 days)



Top 5 regressing queries to previous version



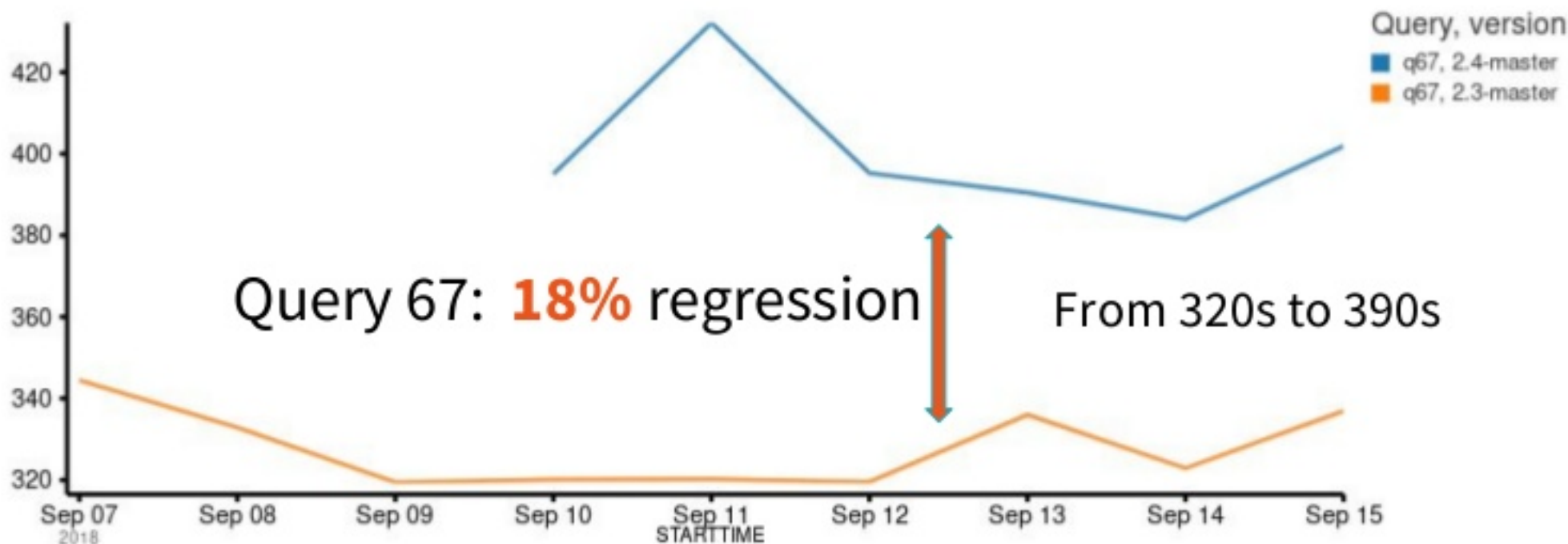
Per run and type geomean



Per query drill-down: 67

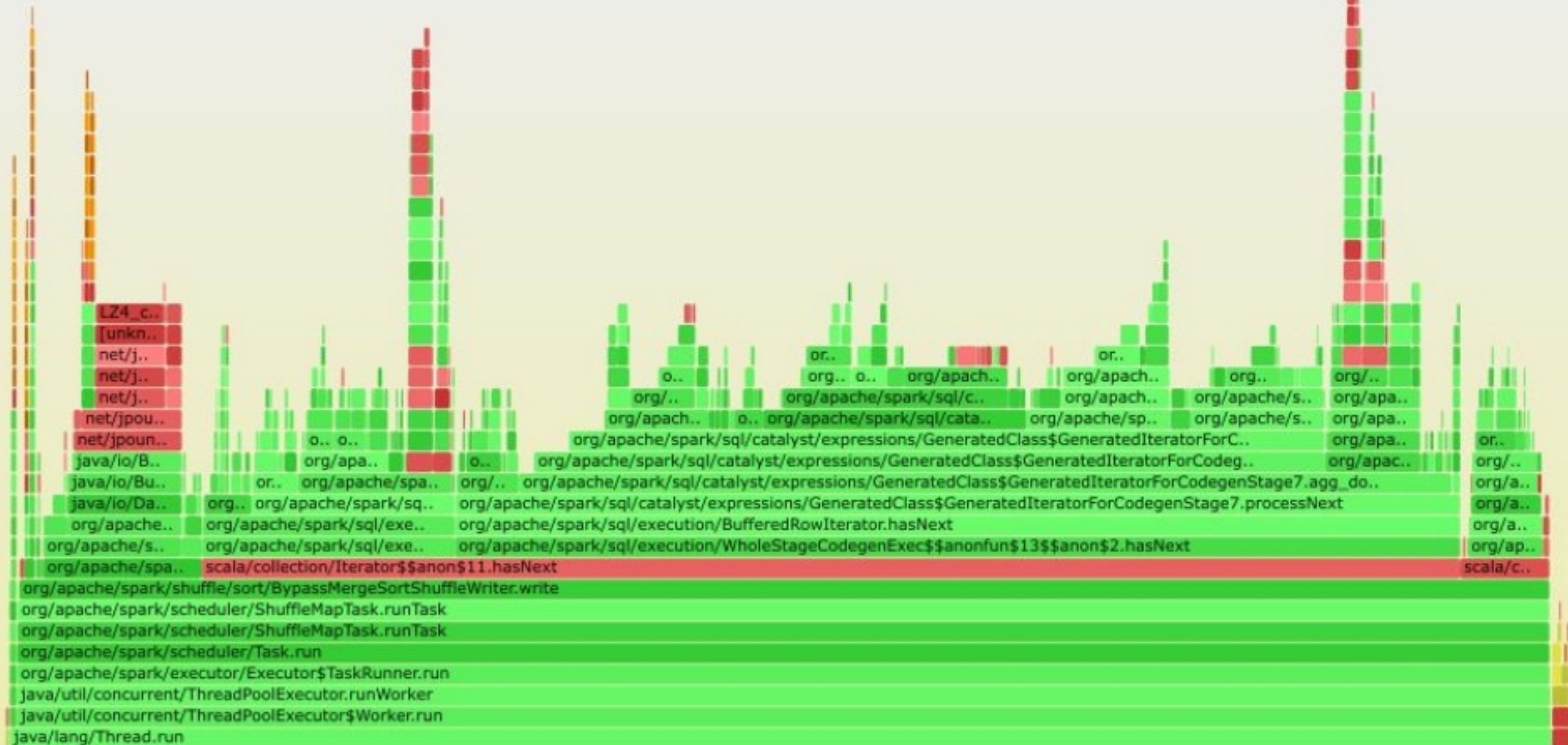


First, **scope** and **validate**



- in 2.4-master (dev) compared
- to 2.3 in DBR 4.3 (prod)

Q67 executor profile for Spark 2.4-master



Side-by-side 2.3 vs 2.4: find the differences



org/apache/...
org/apache/sp...
hyst/expre...
Class\$GeneratedIteratorForC...
\$GeneratedIteratorForCodege...
GeneratedIteratorForCodegenStage7.agg_d

org/apach...
park/sql/c...
k/sql/cata...
ns/GeneratedClass\$GeneratedIteratorForC...

Base (Spark_2.3)

Spark 2.3

Target (Spark_2.4)

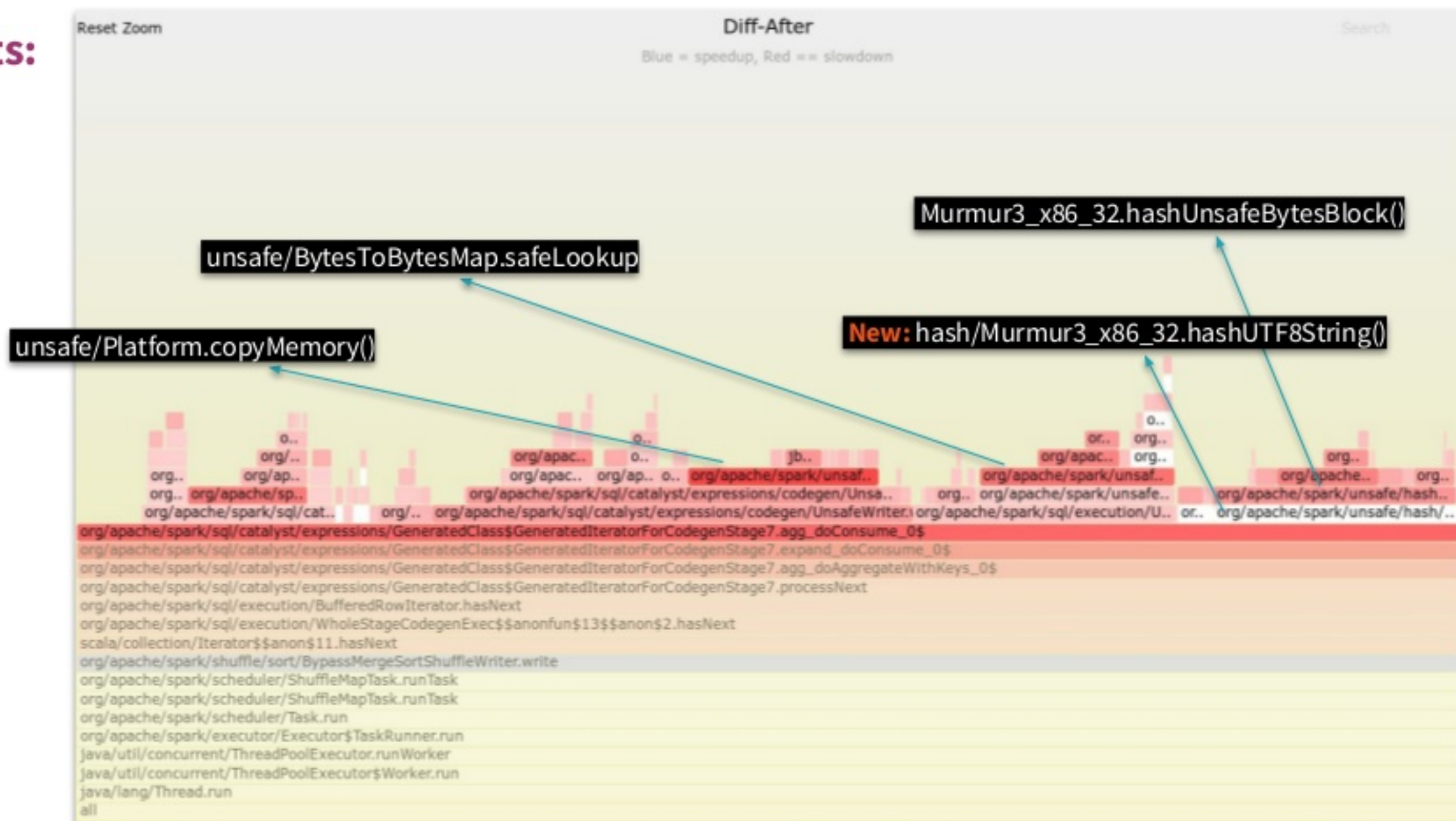
Spark 2.4

Framegraph diff zoom

Red slower **White** new

Look for hints:

- Mem mgmt
- Hashing
- unsafe



Root-causing

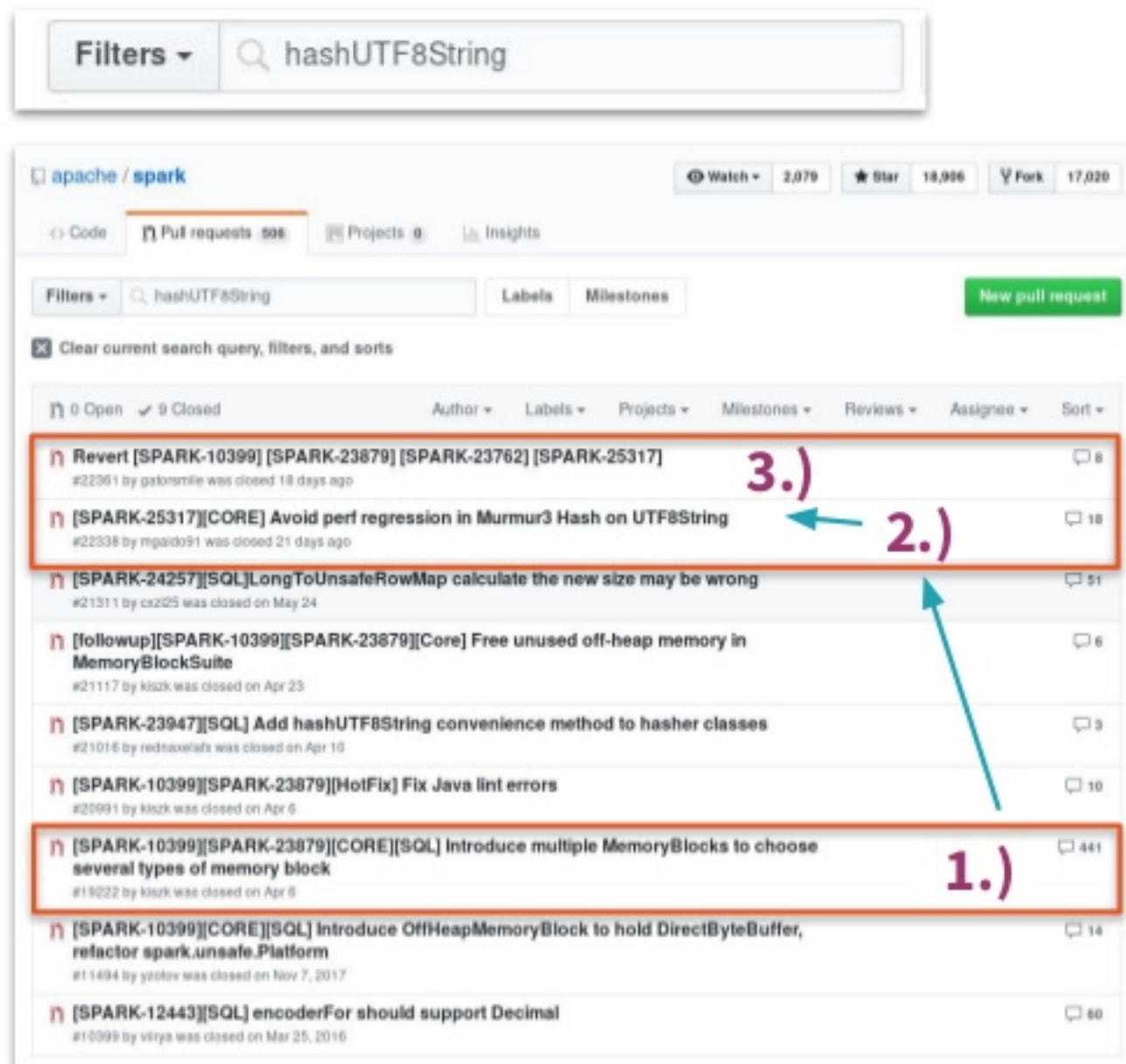
Microbenchmark for UTF8String

```
test("hashing") {  
  import org.apache.spark.unsafe.hash.Murmur3_x86_32  
  import org.apache.spark.unsafe.types.UTF8String  
  val hasher = new Murmur3_x86_32(0)  
  val str = UTF8String.fromString("b" * 10001)  
  val numIter = 100000  
  val start = System.nanoTime  
  for(i <- 0 until numIter) {  
    Murmur3_x86_32.hashUTF8String(str, 0)  
  }  
}
```

Results:

- Spark 2.3: `hashUnsafeBytes()` -> **40µs**
- Spark 2.4 `hashUnsafeBytesBlock()` -> **140µs**
- also slower `UTF8String.getBytes()`

GIT BISECT



Filters ▾ hashUTF8String

apache / spark Watch 2,079 Star 18,986 Fork 17,020

Code Pull requests 508 Projects 0 Insights

Filters ▾ hashUTF8String Labels Milestones New pull request

Clear current search query, filters, and sorts

0 Open 9 Closed Author ▾ Labels ▾ Projects ▾ Milestones ▾ Reviews ▾ Assignee ▾ Sort ▾

- 3.) Revert [SPARK-10399] [SPARK-23879] [SPARK-23762] [SPARK-25317] #22361 by gatorsmile was closed 18 days ago 8
- 2.) [SPARK-25317][CORE] Avoid perf regression in Murmur3 Hash on UTF8String #22338 by mguido91 was closed 21 days ago 18
- [SPARK-24257][SQL] LongToUnsafeRowMap calculate the new size may be wrong #21511 by cz205 was closed on May 24 51
- [followup][SPARK-10399][SPARK-23879][Core] Free unused off-heap memory in MemoryBlockSuite #21117 by klszk was closed on Apr 23 6
- [SPARK-23947][SQL] Add hashUTF8String convenience method to hasher classes #21016 by rednaxelafx was closed on Apr 10 3
- [SPARK-10399][SPARK-23879][HotFix] Fix Java lint errors #20991 by klszk was closed on Apr 6 10
- 1.) [SPARK-10399][SPARK-23879][CORE][SQL] Introduce multiple MemoryBlocks to choose several types of memory block #19222 by klszk was closed on Apr 6 441
- [SPARK-10399][CORE][SQL] Introduce OffHeapMemoryBlock to hold DirectByteBuffer, refactor spark.unsafe.Platform #11494 by yzotov was closed on Nov 7, 2017 14
- [SPARK-12443][SQL] encoderFor should support Decimal #10099 by vliya was closed on Mar 25, 2016 60

It is a journey to get a release out



DBR and Spark testing and performance are a **continuous effort**

- **Over a month** effort to bring performance to improving

Conclusion

Spark in production is *not just the framework*
Unit and integration testing are **not enough**

We need **Spark specific tools** to automate the process
to ensure both **correctness and performance**

Thanks!



Correctness and Performance of Apache Spark SQL

October 2018

 databricks®