How we test correctness and performance

Bogdan Ghit and Nicolas Poggi

Vrije Universiteit

December 12, 2018

databricks

About me

BOGDAN GHIT

Databricks, Software Engineer

- Query 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

About Databricks Amsterdam

We are hiring!

https://databricks.com/company/careers
bogdan.ghit@databricks.com

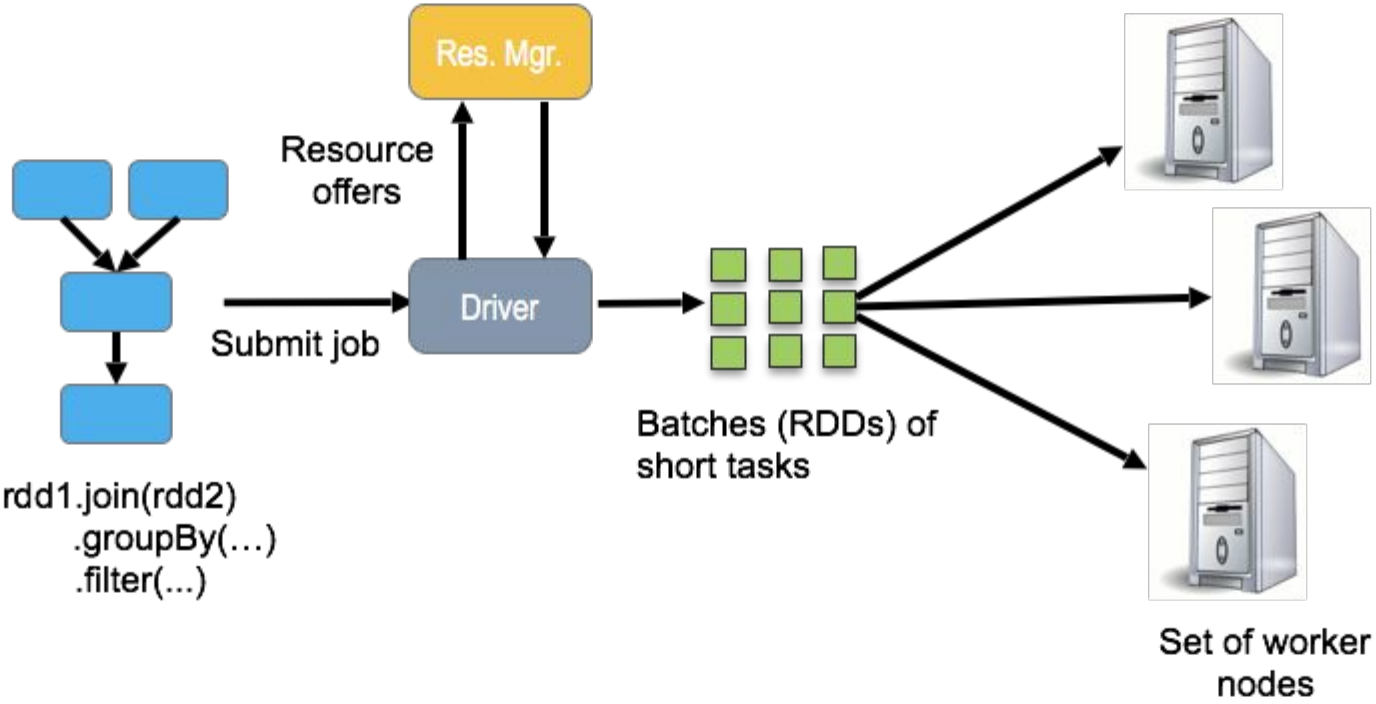
Company mission: *make Databricks Runtime a unified analytics platform to accelerate innovation*

Team mission: *make the Databricks Runtime the fastest execution engine for Apache Spark*

Teams in Amsterdam:

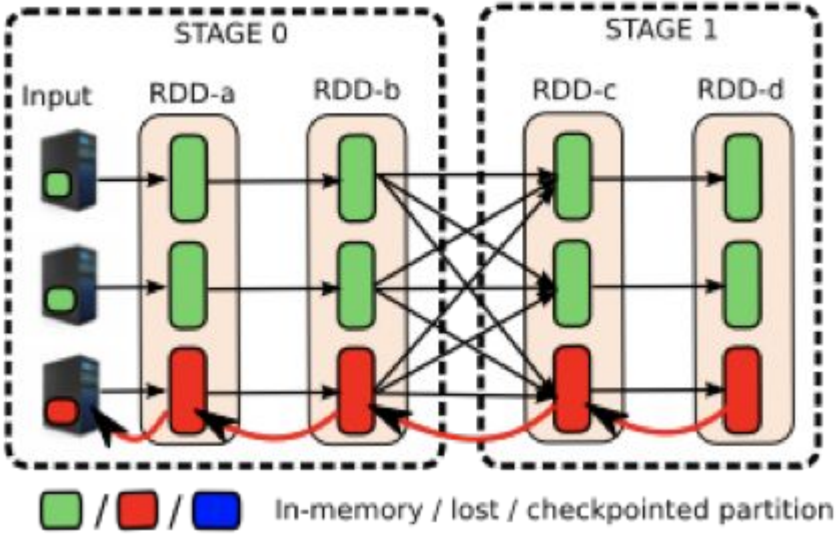
1. Query Performance
2. Storage and IO
3. Spark Benchmarking
4. Databricks UIs
5. Billing Infrastructure (new)

How Spark works

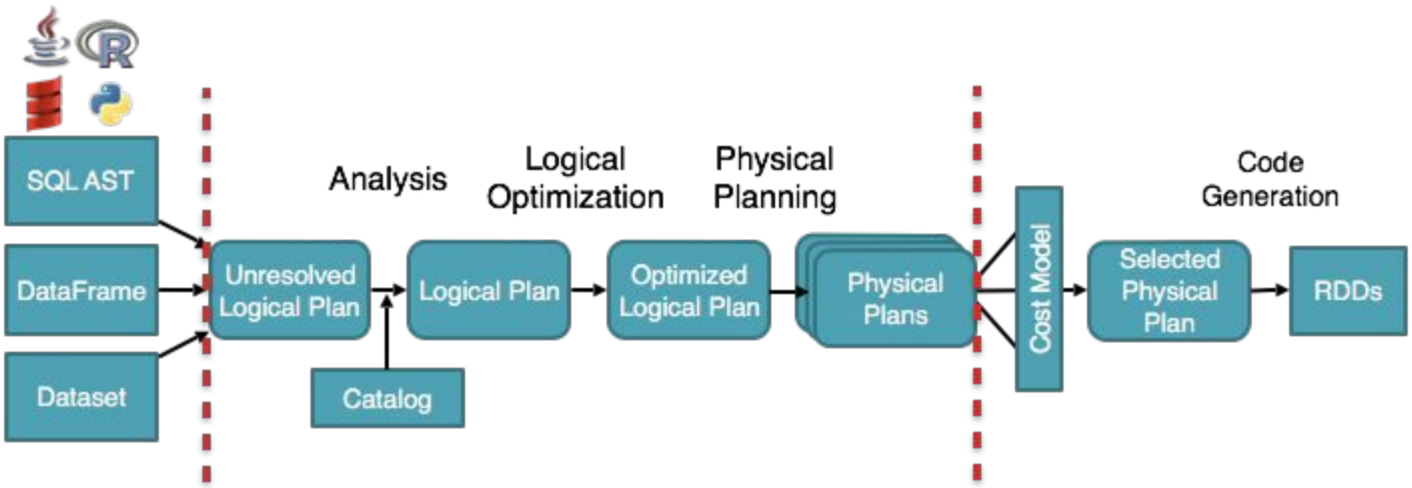


RDD abstraction

Main abstraction: resilient distributed datasets (RDDs)
Collection of data partitions distributed across multiple machines.
Rich set of coarse-grained transformations.

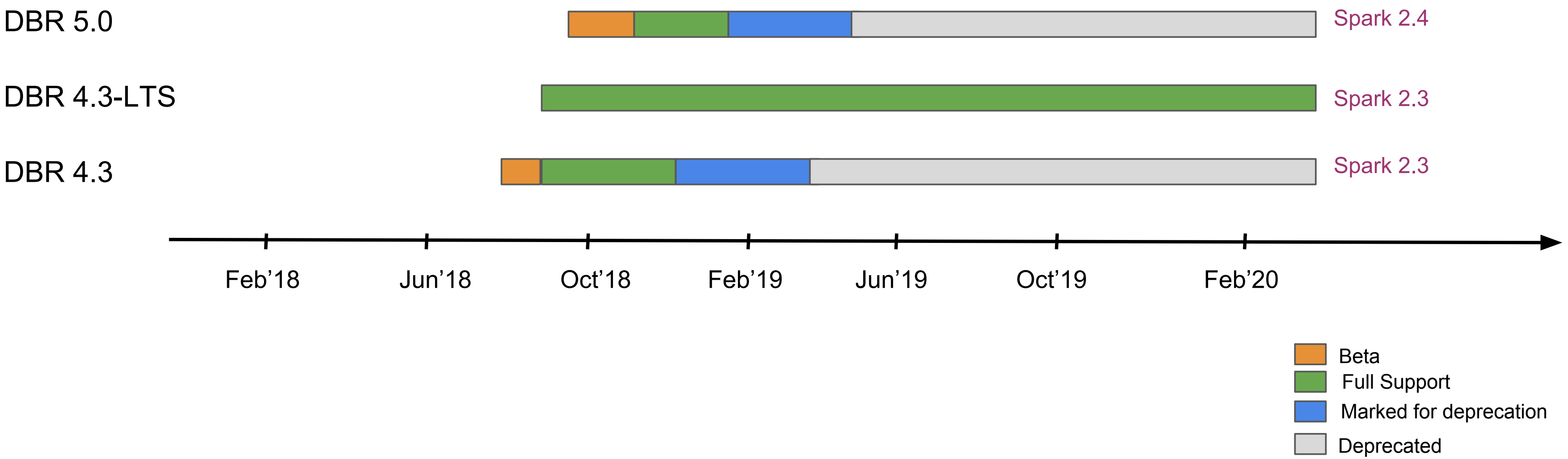


Spark SQL behind the scenes



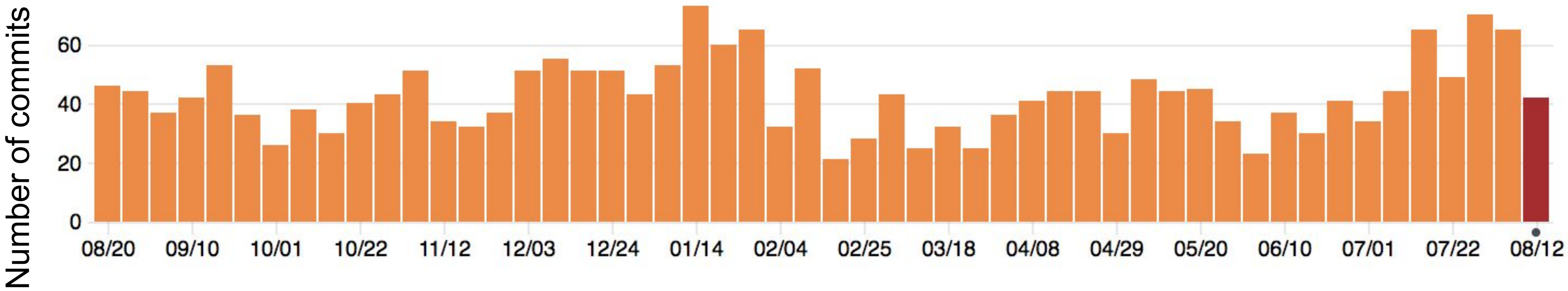
Databricks ecosystem

Databricks runtime (DBR) releases



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

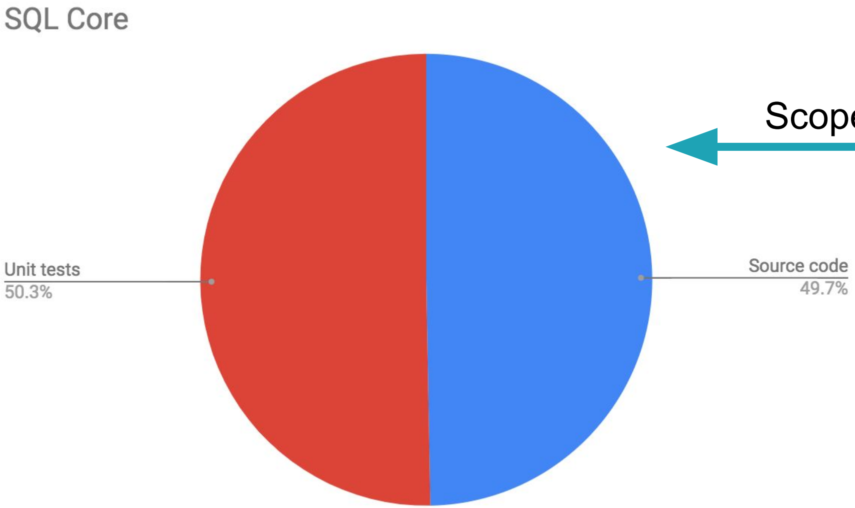
Apache Spark contributions



Hundreds of commits monthly to the Apache Spark project

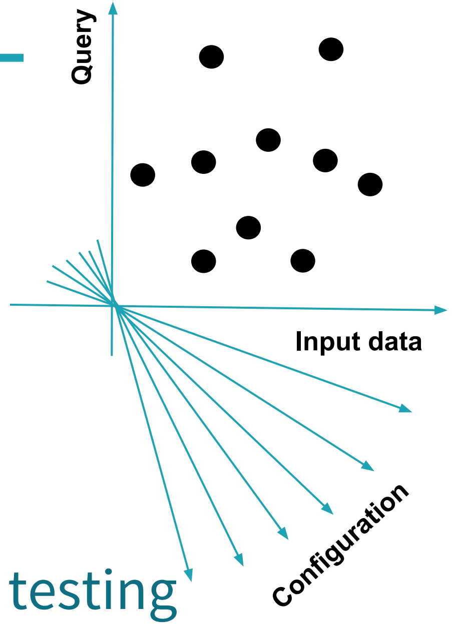
At this pace of development, **mistakes** are bound to happen

Where do these contributions go?



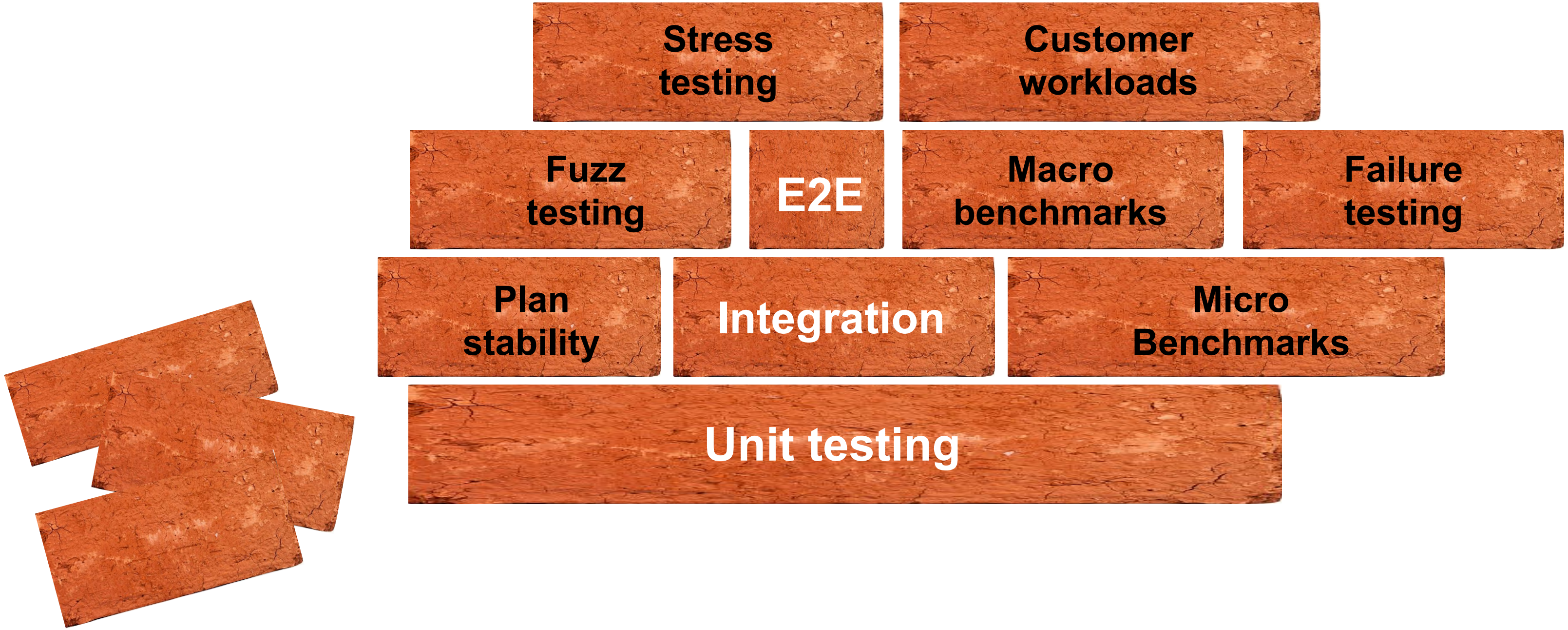
Scope of the testing

Over 200 built-in functions



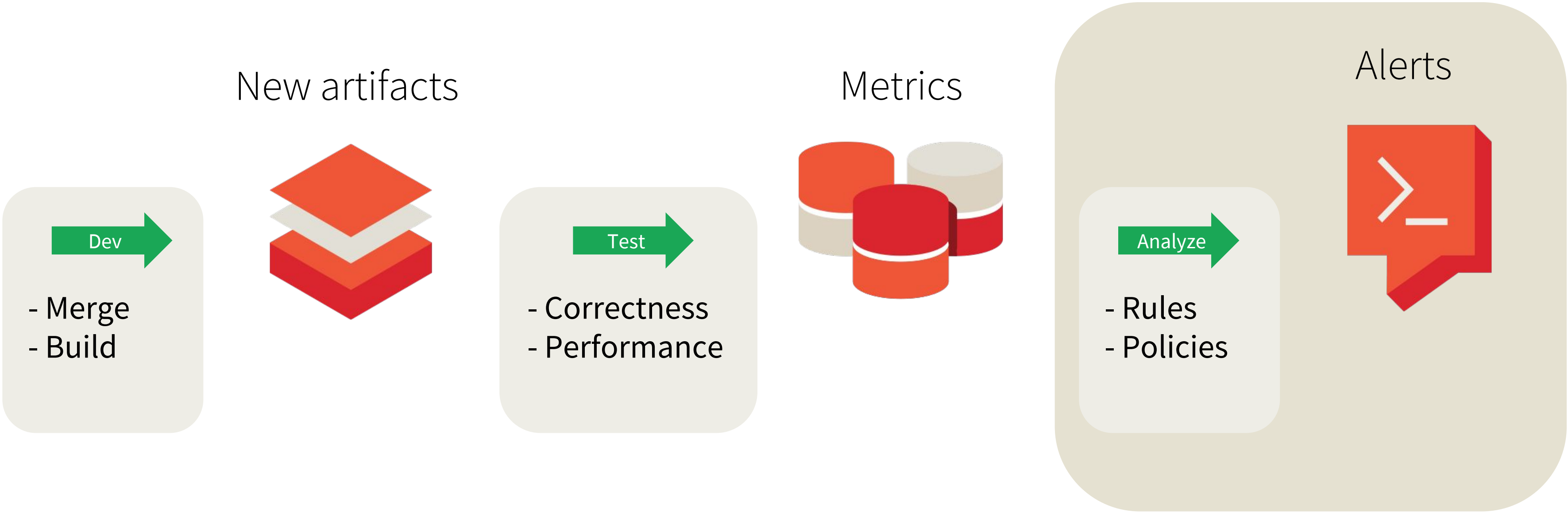
Developers put a significant **engineering effort** in testing

Yet another brick in the wall

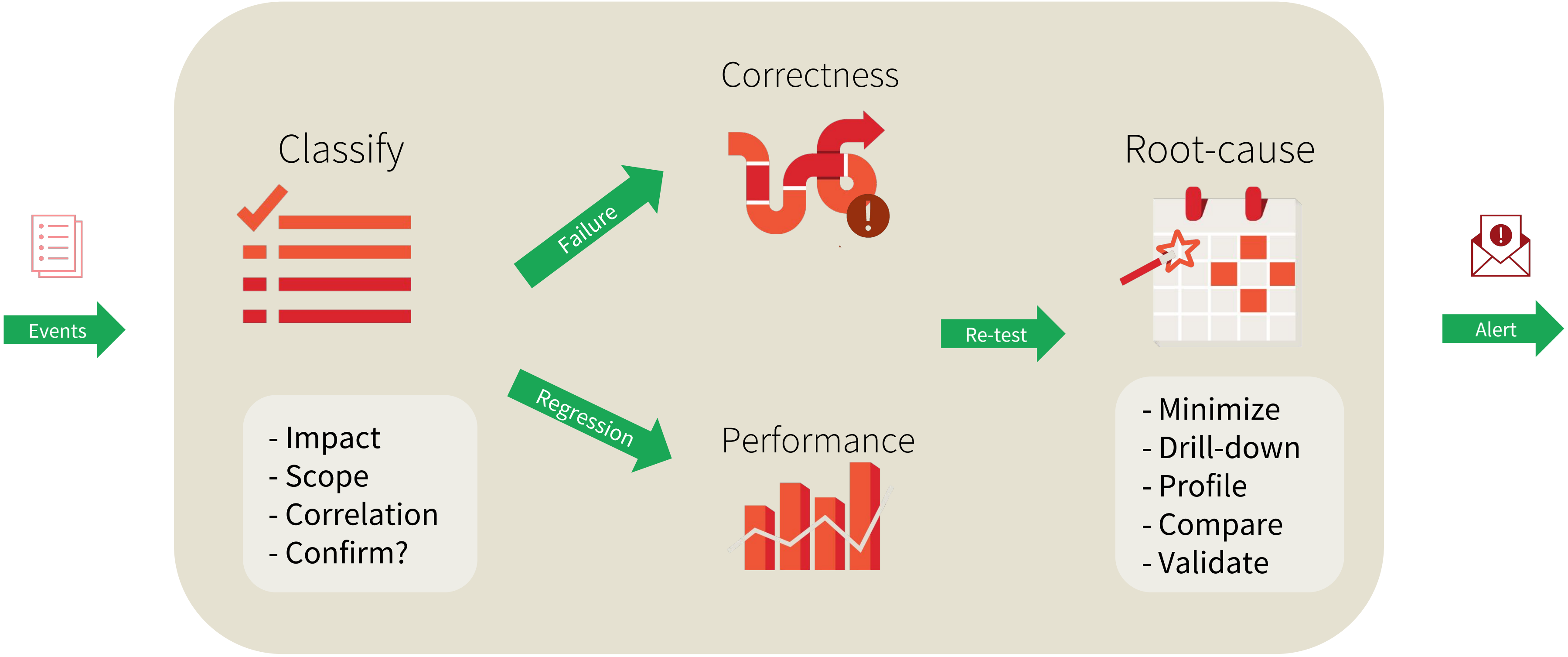


Unit testing ***is not enough*** to guarantee correctness and performance

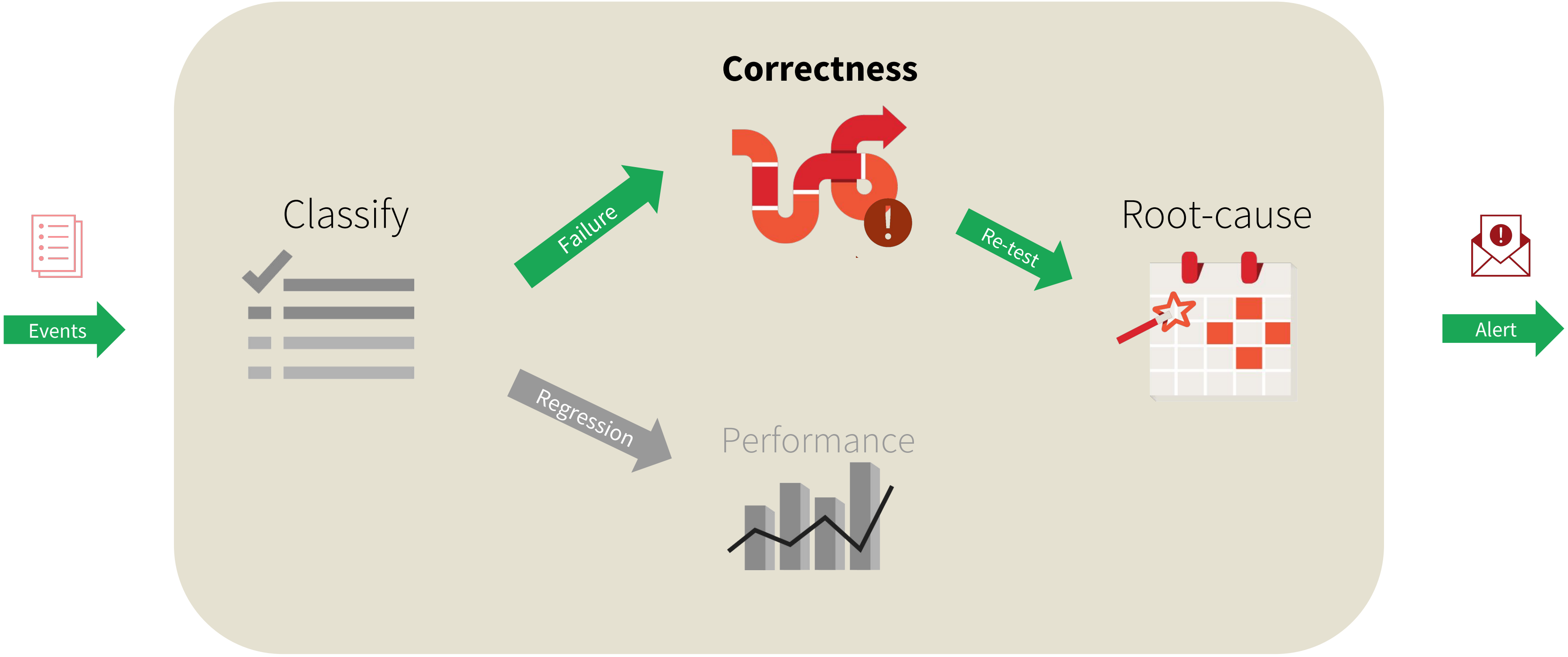
Continuous Integration pipeline



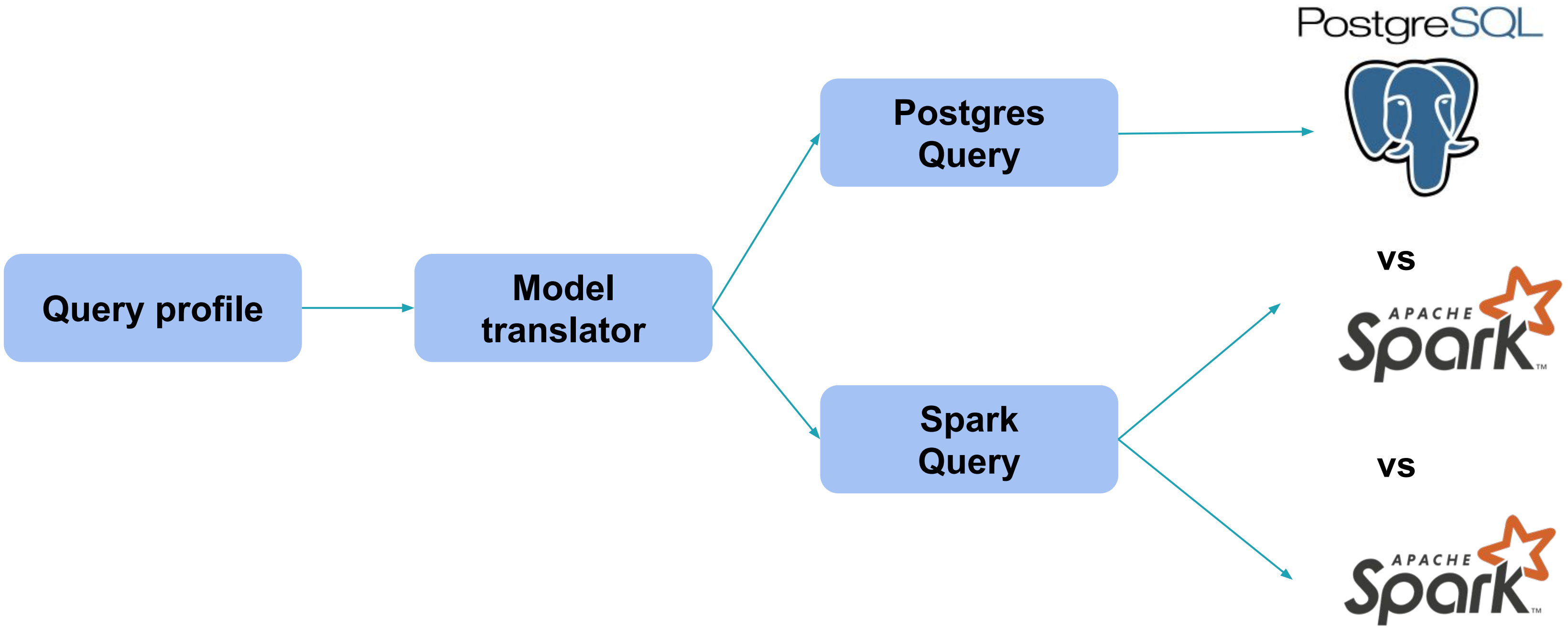
Classification and alerting



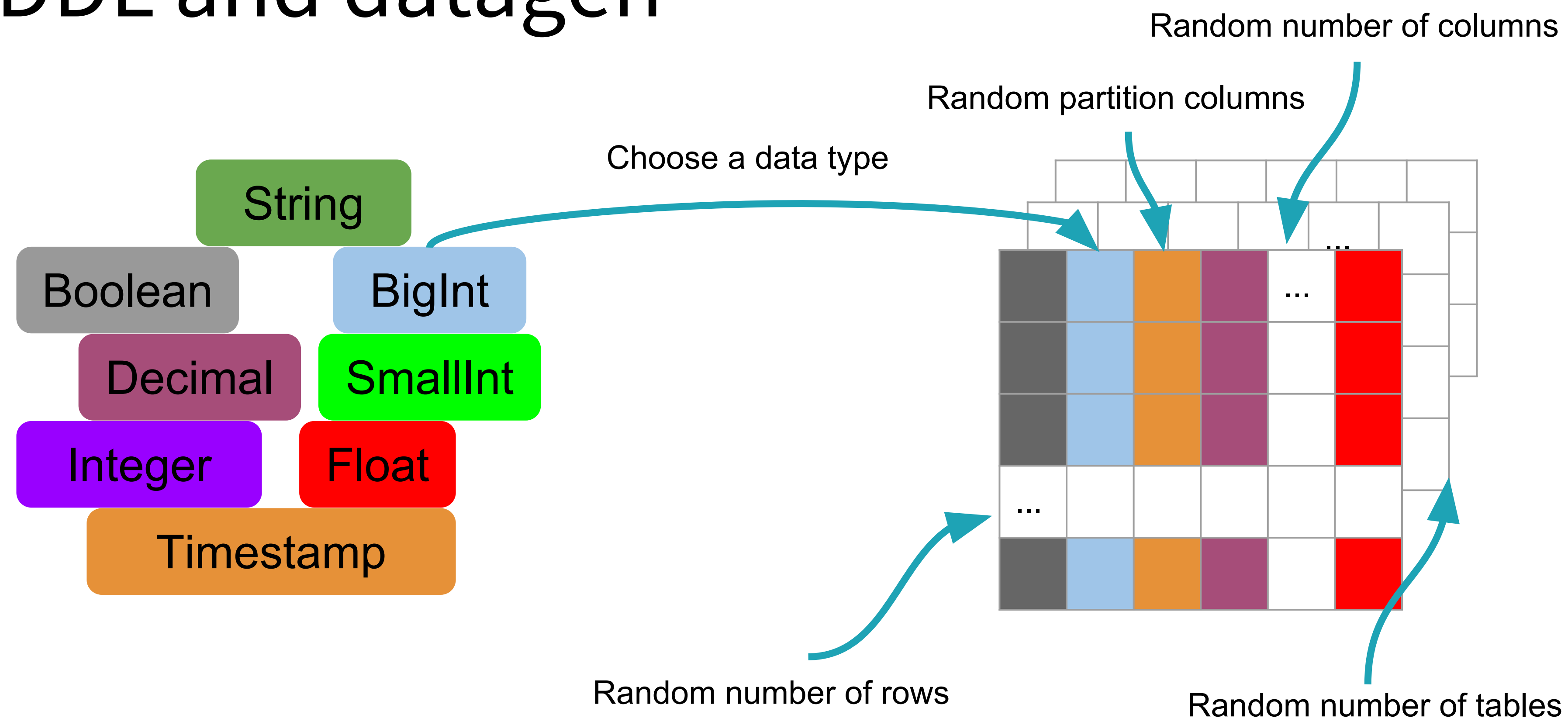
Correctness



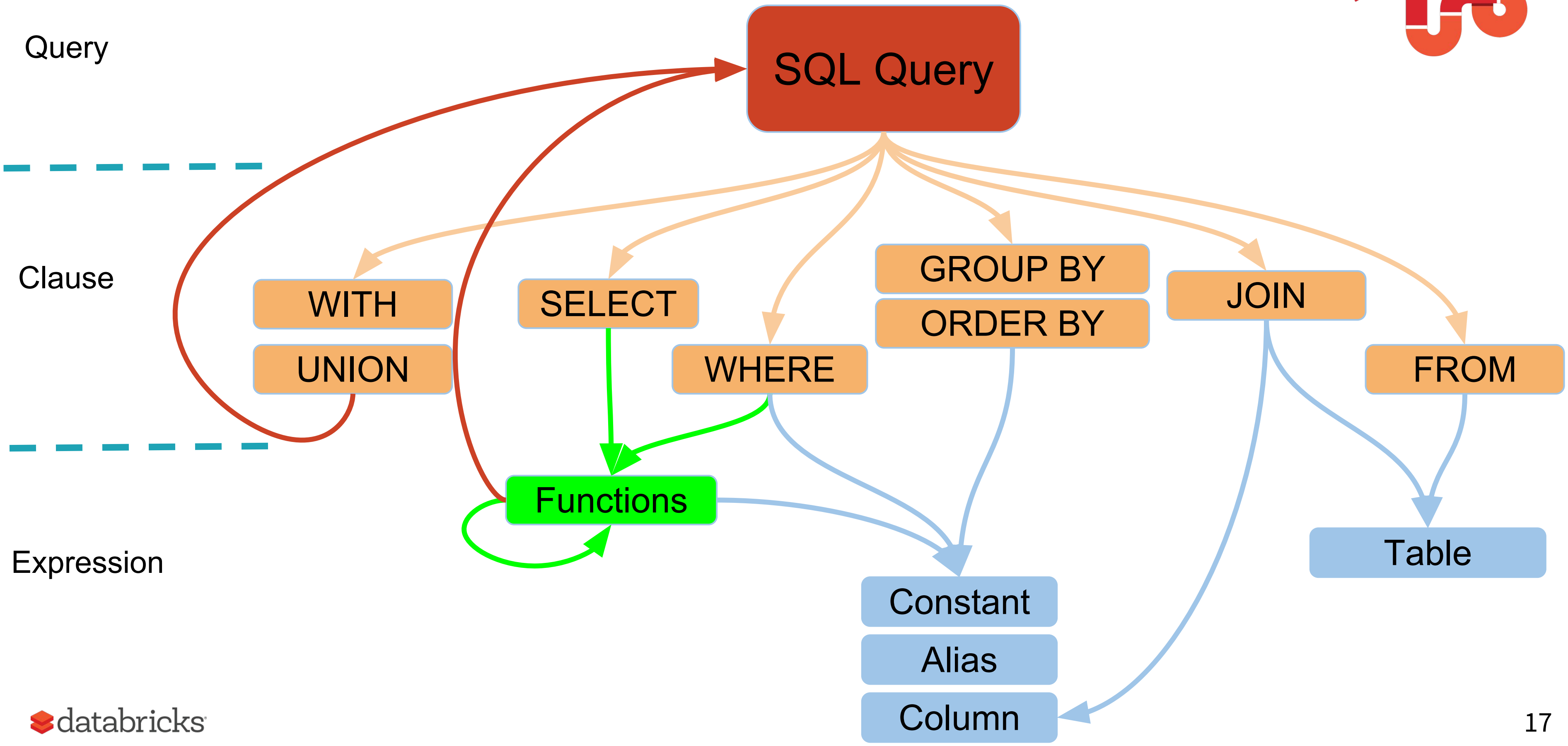
Random query generation



DDL and datagen



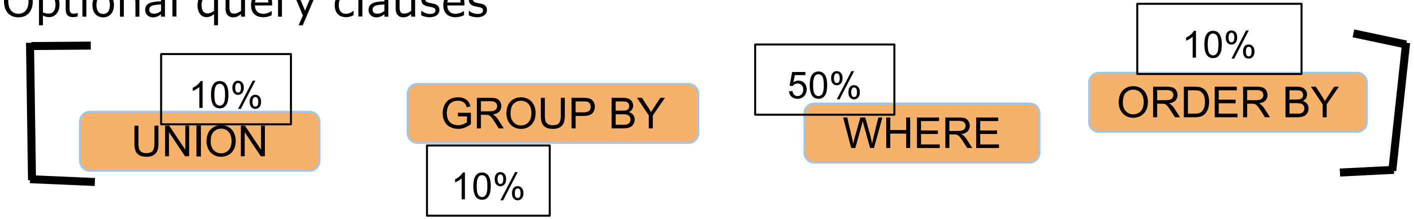
Recursive query model



Probabilistic query profile

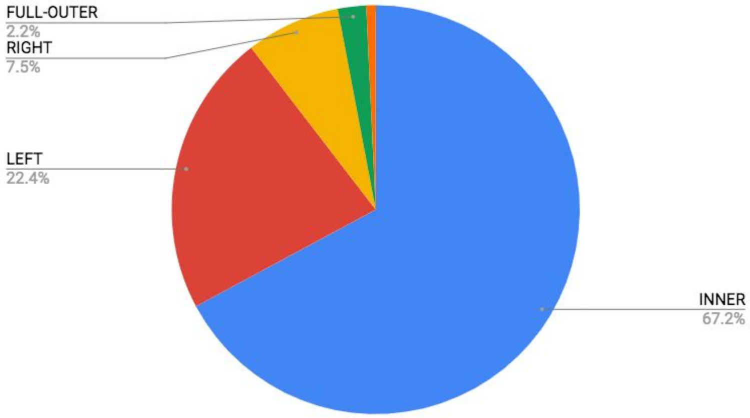
Independent weights

- Optional query clauses



Inter-dependent weights

- Join types
- Select functions



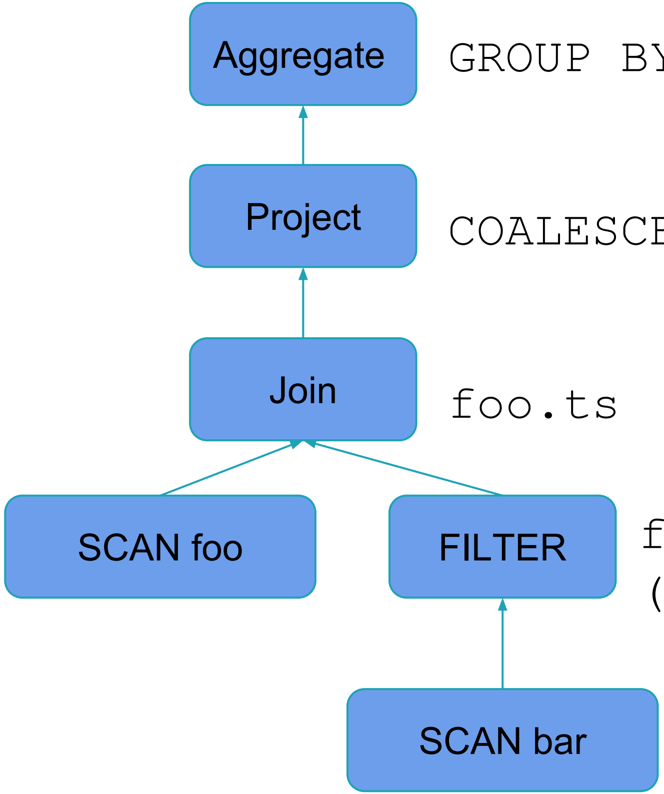
Coalesce flattening (1/4)

```sql
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

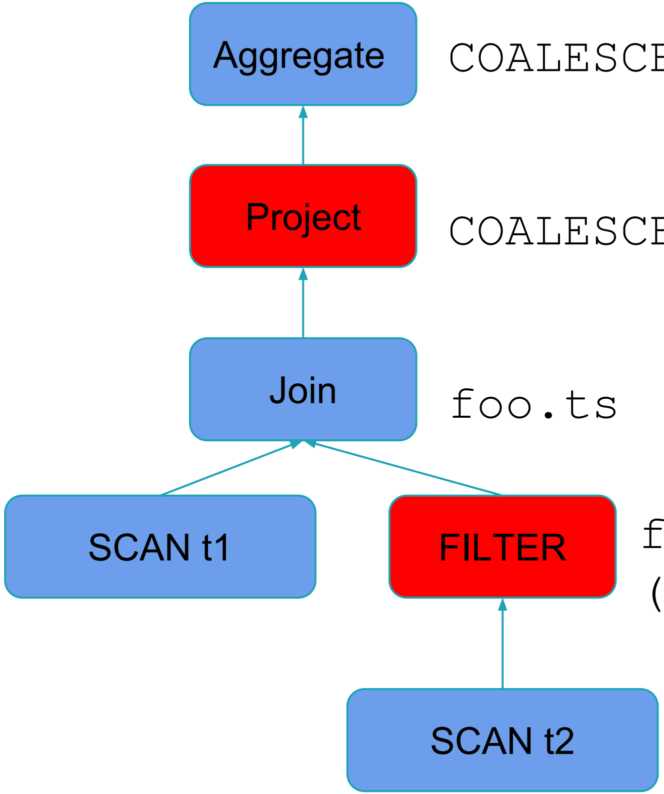
Error: Operation is in ERROR_STATE

Coalesce flattening (2/3)



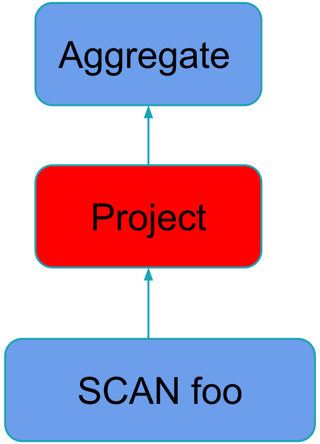

```
GROUP BY COALESCE(foo.id, foo.val)

COALESCE(COALESCE(foo.id, foo.val), 88)

foo.ts  = bar.ts

foo.id IN
(CAST('0.04' AS DECIMAL(10, 10)), foo.id)
```

Coalesce flattening (3/4)




```
COALESCE(foo.id, foo.val)

COALESCE(COALESCE(foo.id, foo.val), 88)

foo.ts  = bar.ts

foo.id IN
(CAST('0.04' AS DECIMAL(10, 10)), foo.id)
```

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)

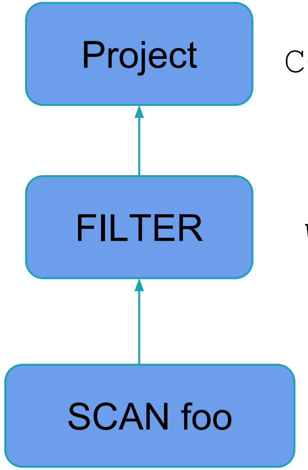
```
SELECT (t1.decimal0803_col_3) / (t1.decimal0803_col_3) AS decimal_col,
        CAST(696 AS STRING) AS char_col, t1.decimal0803_col_3,
       (COALESCE(CAST('0.02' AS DECIMAL(10,10)),
                 CAST('0.47' AS DECIMAL(10,10)),
                 CAST('-0.53' AS DECIMAL(10,10)))) +
       (LEAD(-65, 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] POSTGRESQL row

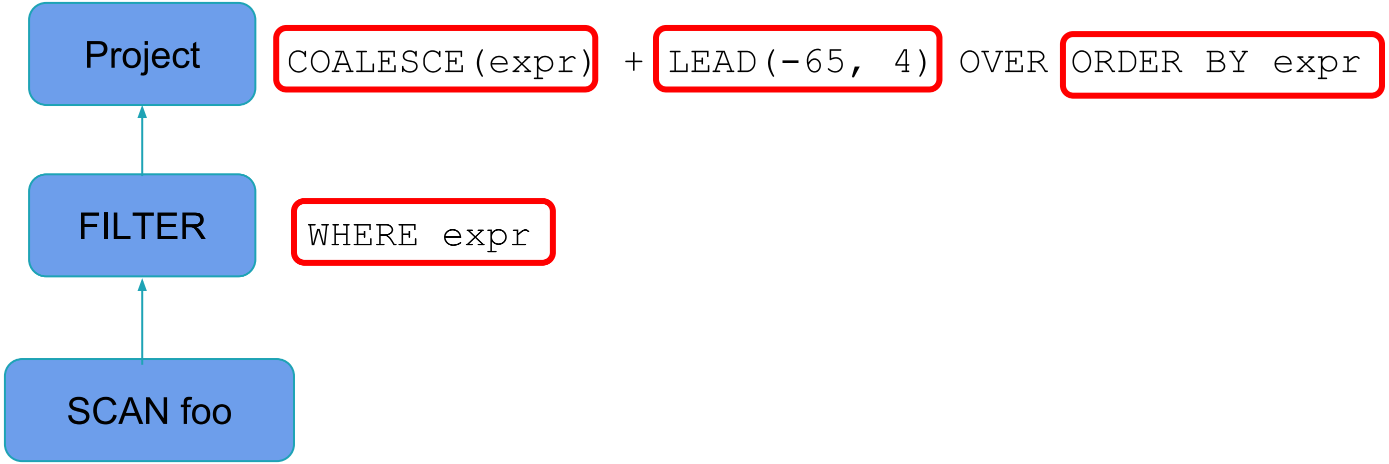
Lead function (2/3)




```
COALESCE(expr) + LEAD(-65, 4) OVER ORDER BY expr
```

```
WHERE expr
```

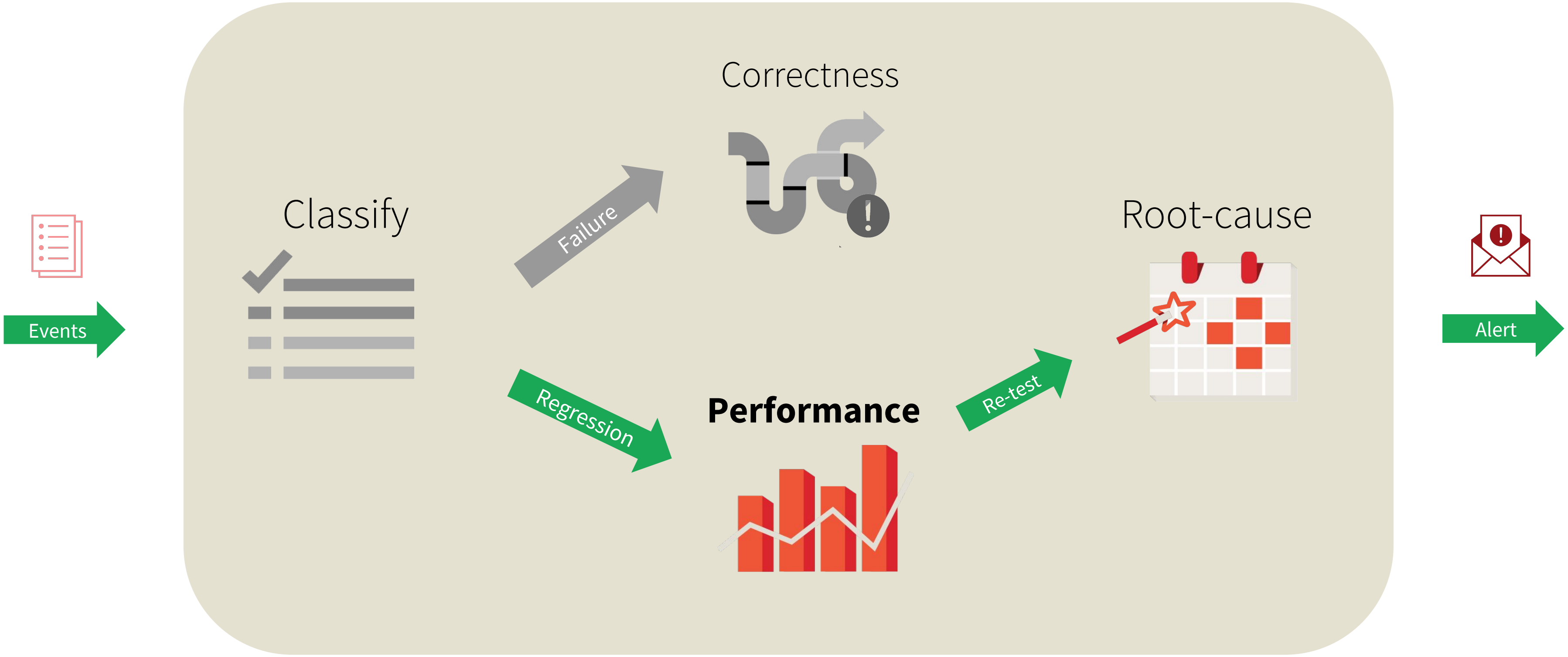
Lead function (3/3)



Analyzing the error

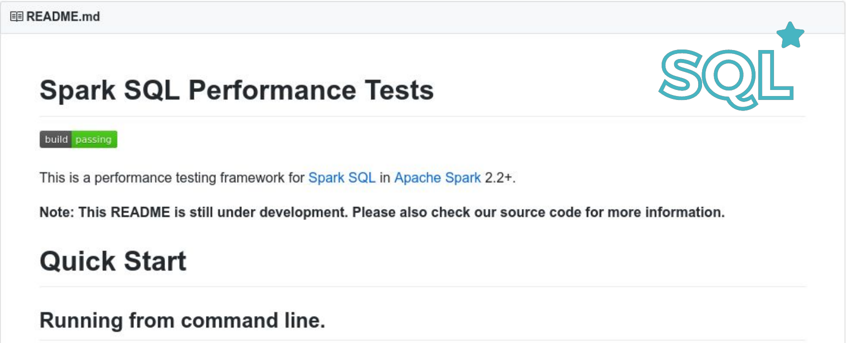
- Using constant input values breaks the behaviour of the LEAD function
- SC-16633: https://github.com/apache/spark/pull/14284

Performance



Benchmarking tools

- spark-sql-perf public library for TPC workloads
- datagen and import scripts local, cluster, S3
- dashboards for analyzing results


README.md

Spark SQL Performance Tests

build passing

This is a performance testing framework for Spark SQL in Apache Spark 2.2+.

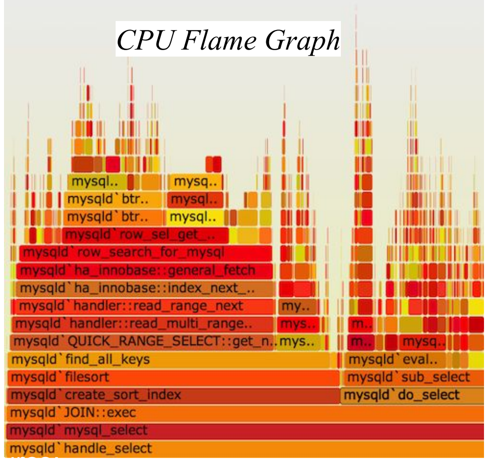
Note: This README is still under development. Please also check our source code for more information.

Quick Start

Running from command line.

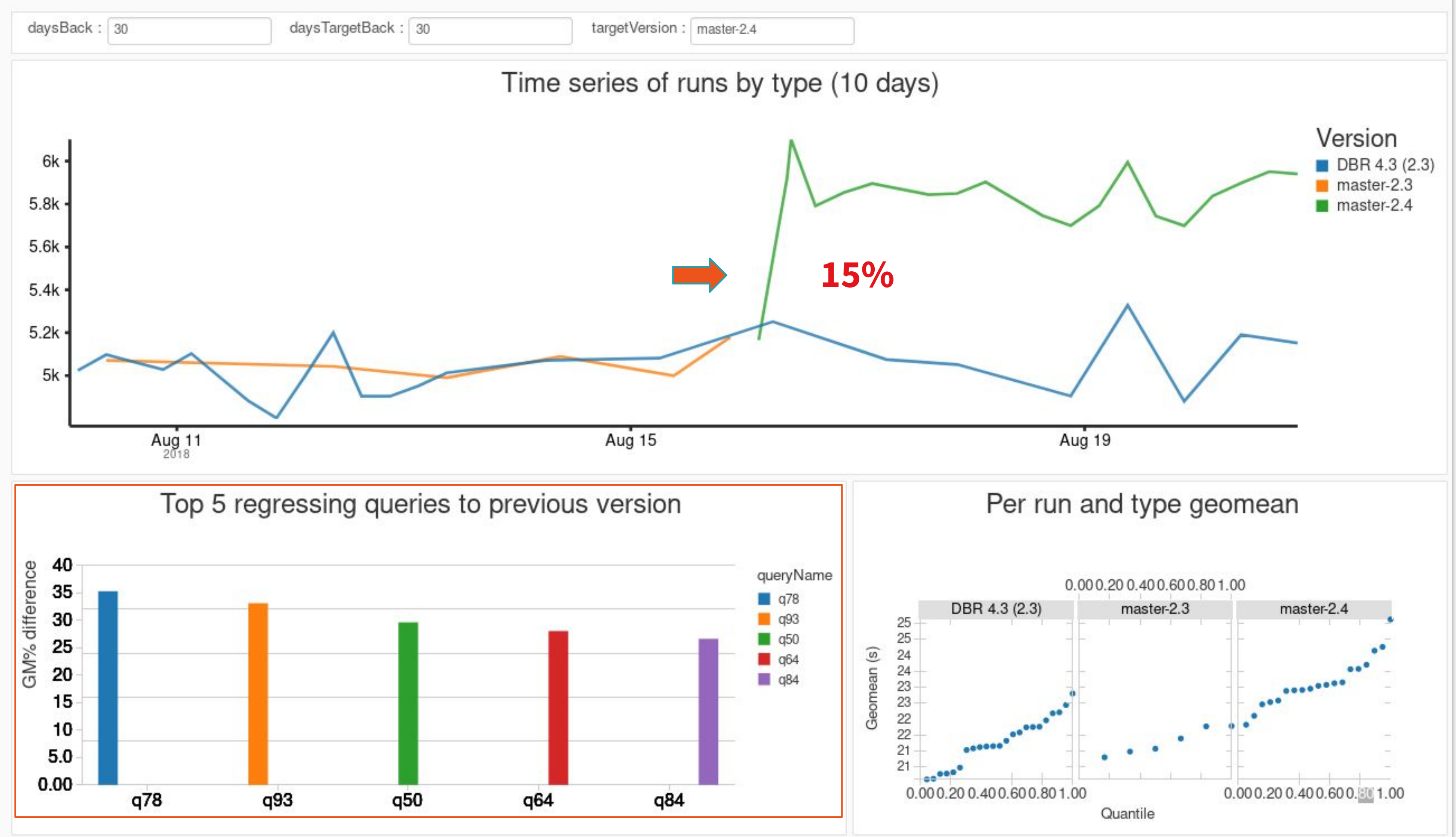
https://github.com/databricks/spark-sql-perf

- Spark micro benchmarks
- Async-profiler to produce flamegraphs



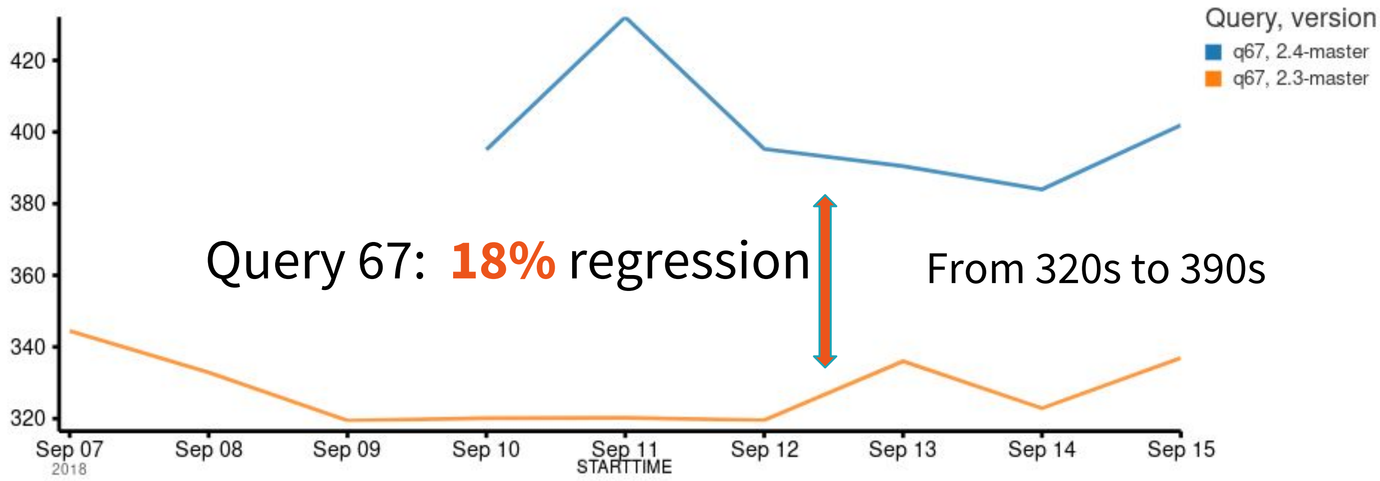
http://www.brendangregg.com/flamegraphs.html

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



Per query drill-down: 67

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

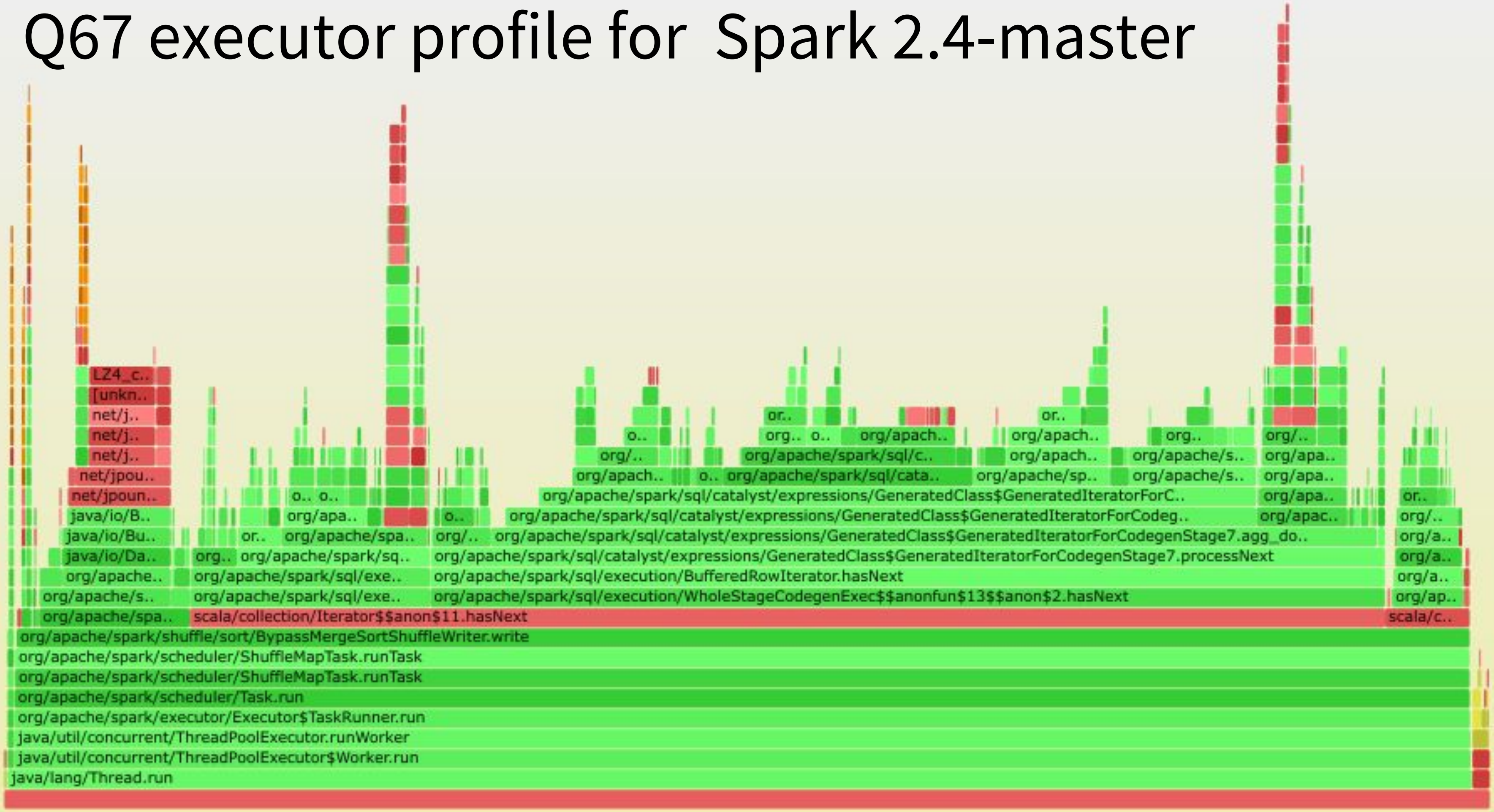


Query 67: **18%** regression

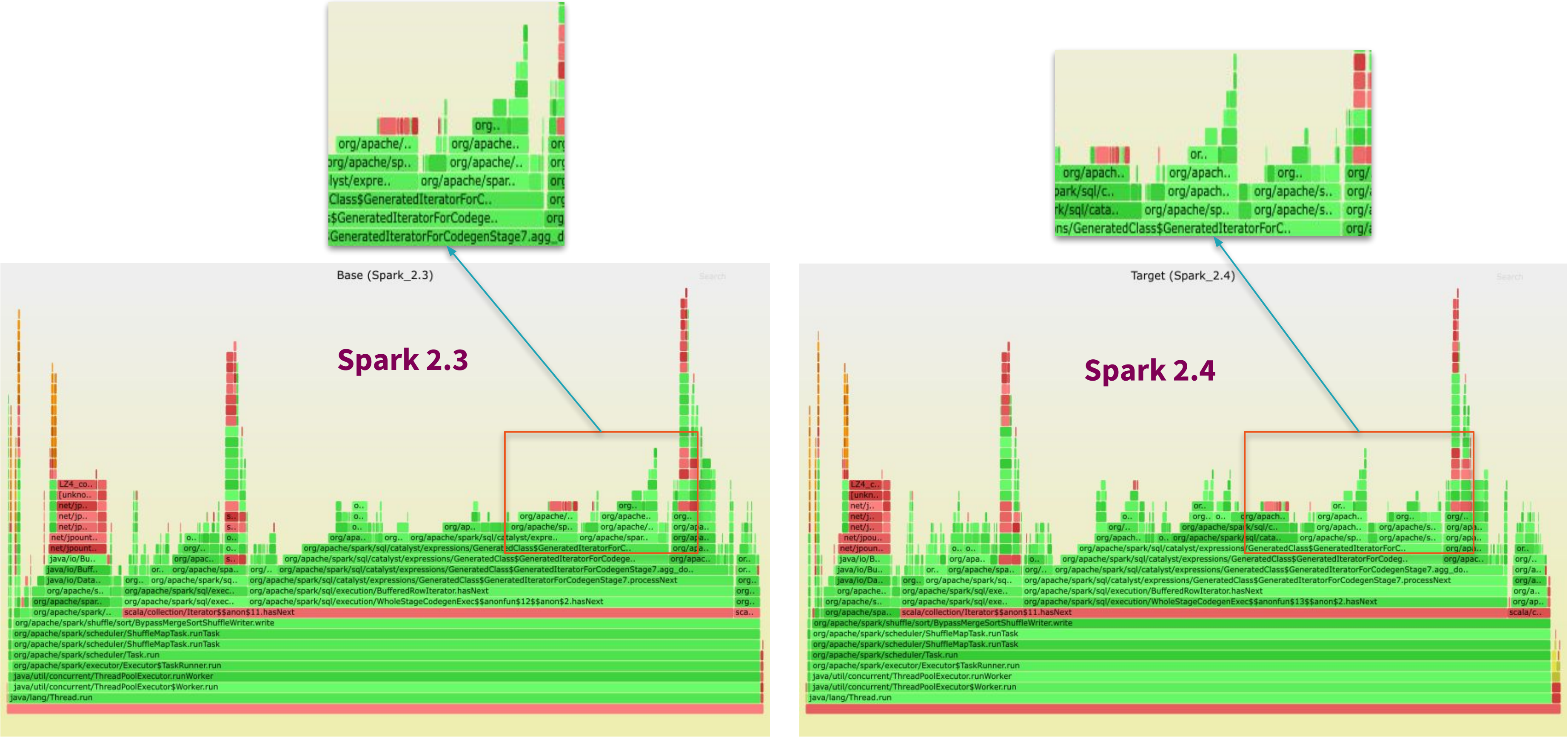
From 320s to 390s

- 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

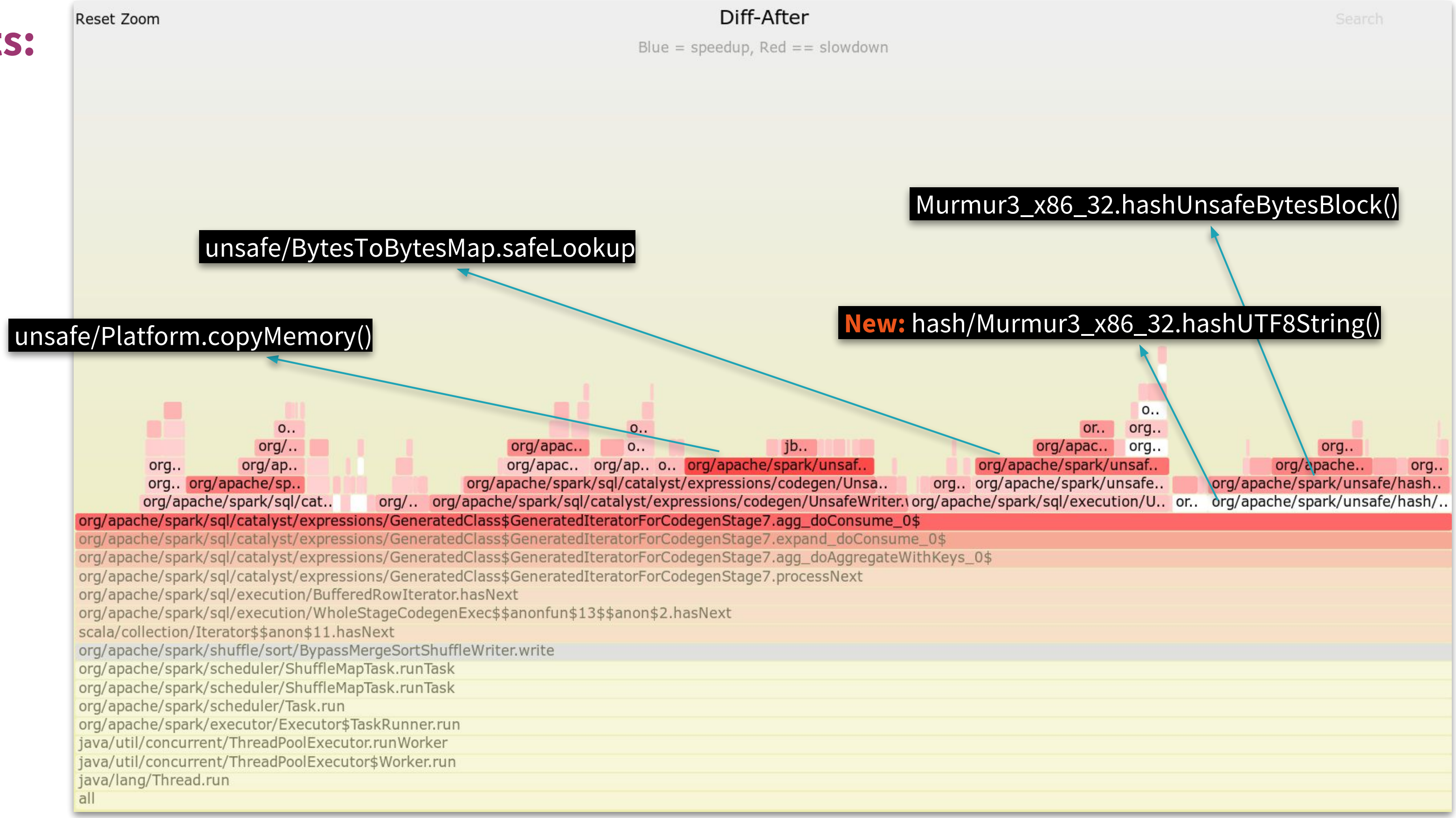


Framegraph diff zoom

Red slower **White** new

Look for hints:

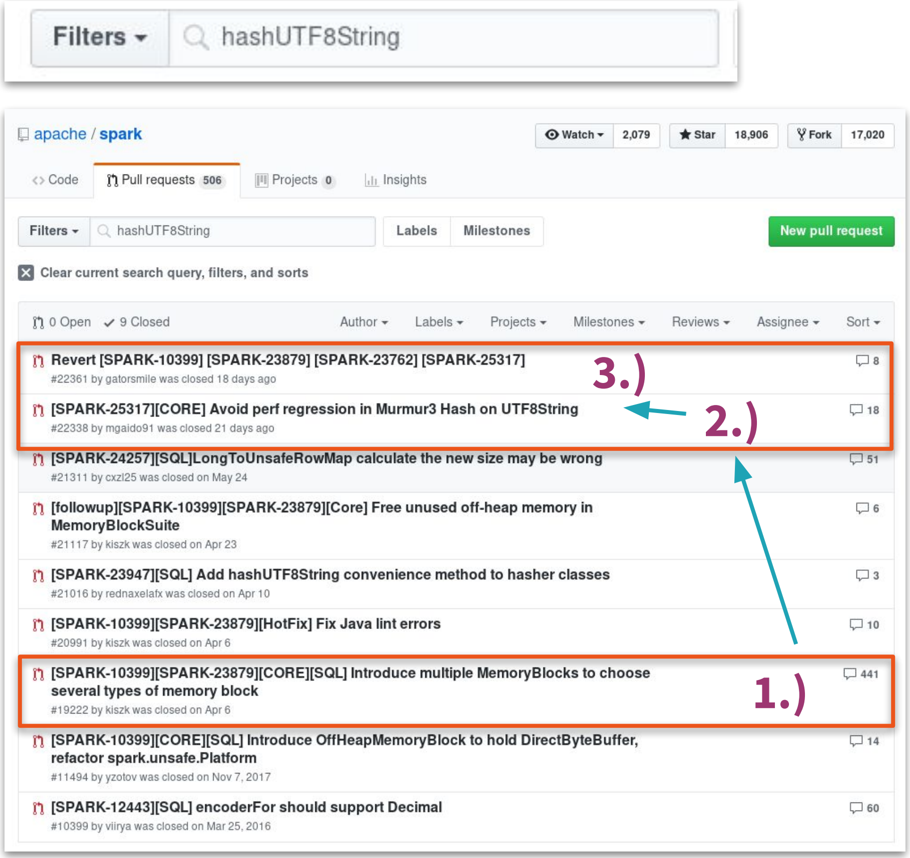
- Mem mgmt
- Hashing
- unsafe



databricks

Root-causing

GIT BISECT



Microbenchmark for UTF8String

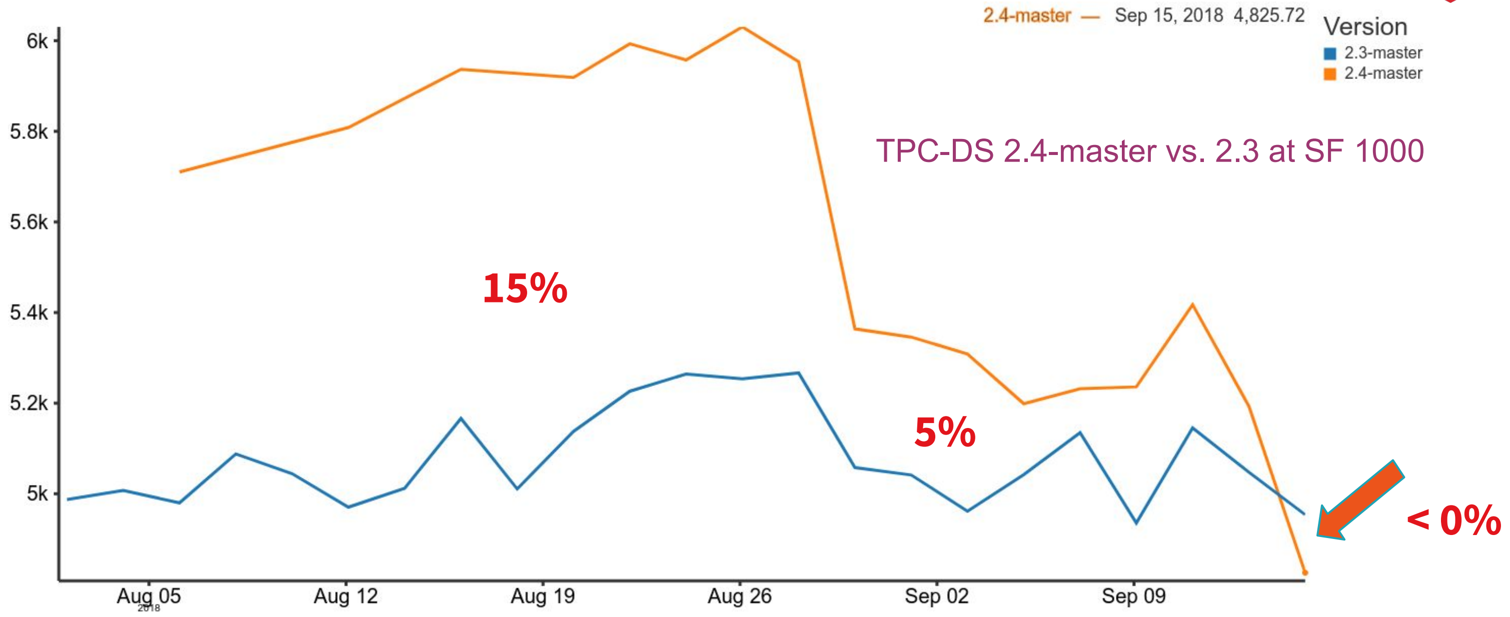
```scala
test("hashing") {
    import org.apache.spark.unsafe.hash.Murmur3_x86_32
    import org.apache.spark.unsafe.types.UTF8String
    val hasher = new Murmur3_x86_32(0)
    val str = UTF8String.fromString("b" * 10001)
    val numIter = 100000
    val start = System.nanoTime
    for(i <- 0 until numIter)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

Results:

- Spark 2.3: hashUnsafeBytes() -> **40µs**
- Spark 2.4 *hashUnsafeBytesBlock()* -> **140µs**

- also slower *UTF8String.getBytes()*

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

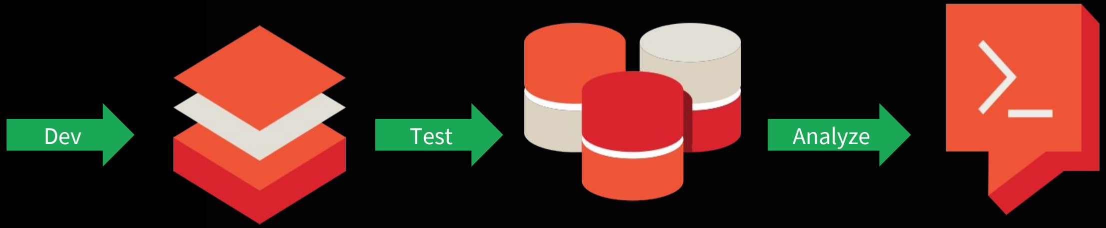
Spark in production is *not just the framework*

Unit and integration testing are not enough

We need Spark specific tools to automate the process

to ensure both correctness and performance

Thanks!



databricks