Deep Dive Into Catalyst: Apache Spark's Optimizer

Herman van Hövell

Spark Summit Europe, Brussels
October 26th 2016

databricks

Who is Databricks

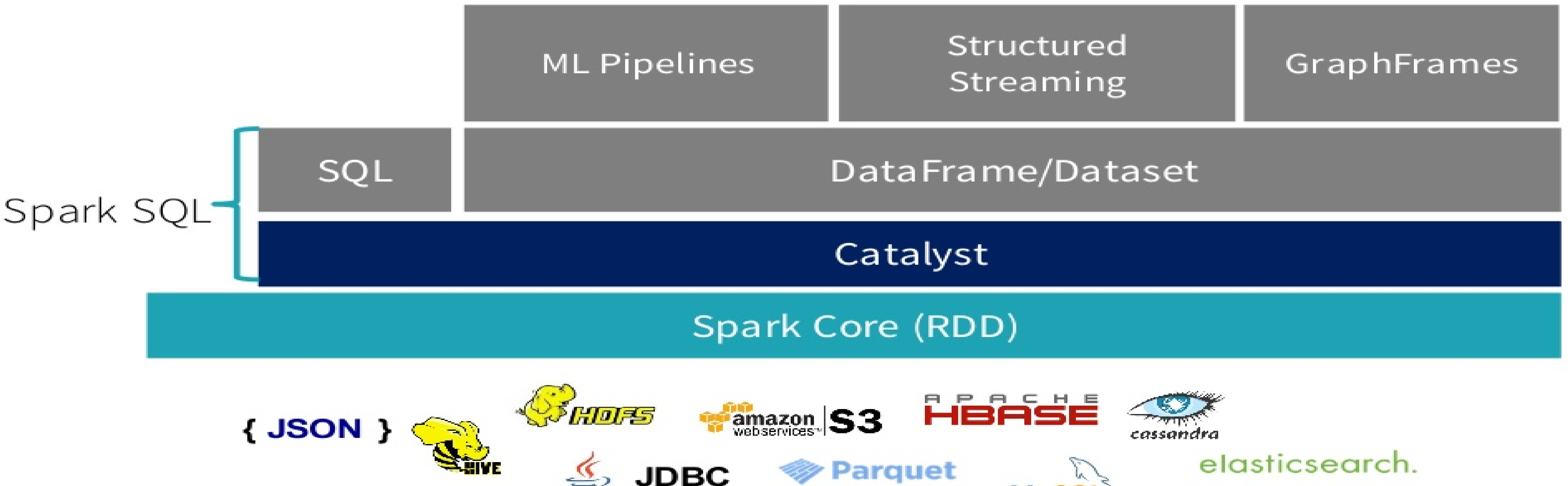
Why Us

- Created Apache Spark to enable big data use cases with a single engine.
- Contributes 75% of Spark's code

Our Product

- Bring Spark to the enterprise: The just-in-time data platform.
- Fully managed platform powered by Apache Spark.
- A unified solution for data science and engineering teams.

Overview



ML Pipelines
Structured Streaming
GraphFrames
SQL
DataFrame/Dataset
Spark SQL
Catalyst
Spark Core (RDD)
{ JSON }
HIVE
HDFS
amazon webservices S3
APACHE HBASE
cassandra
JDBC
Parquet
MySQL
elasticsearch.
and more…

Why structure?

- By definition, structure will *limit* what can be expressed.
- In practice, we can accommodate the vast majority of computations.

Limiting the space of what can be expressed enables optimizations.

Why structure?

RDD

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

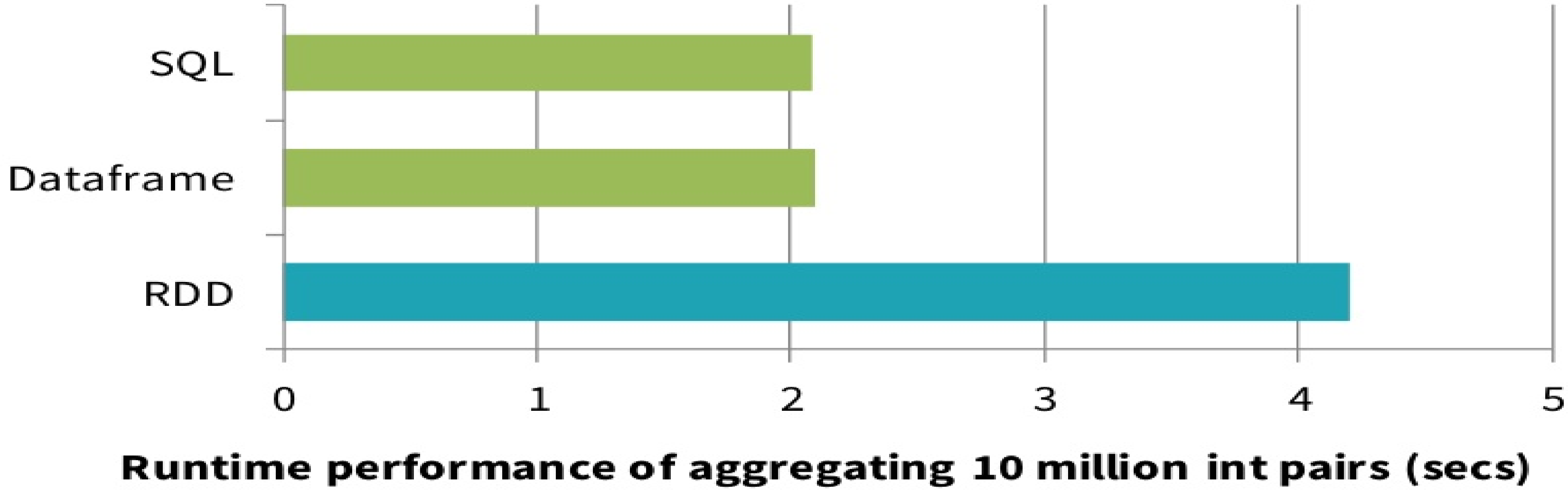
Dataframe

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

SQL

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

Why structure?



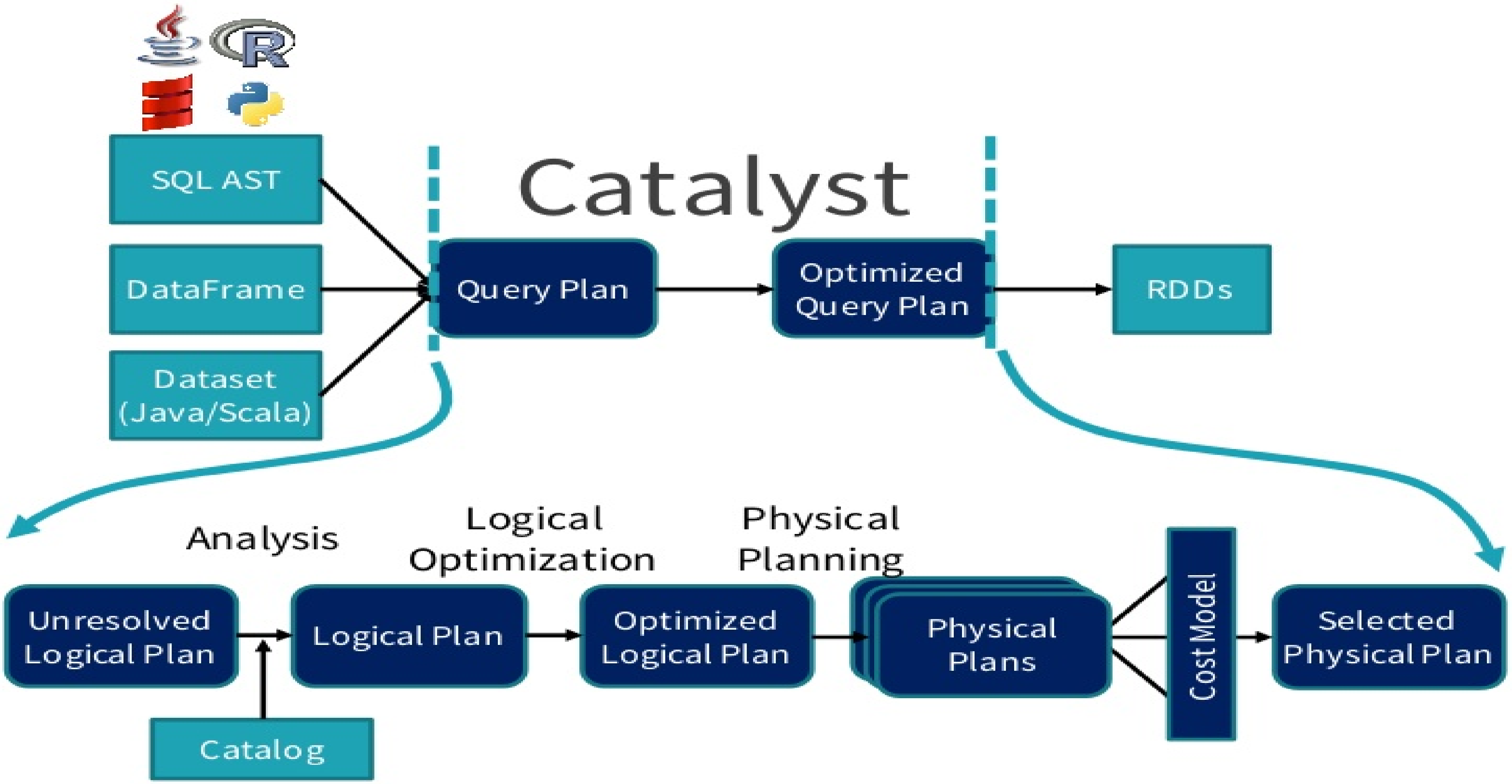
databricks

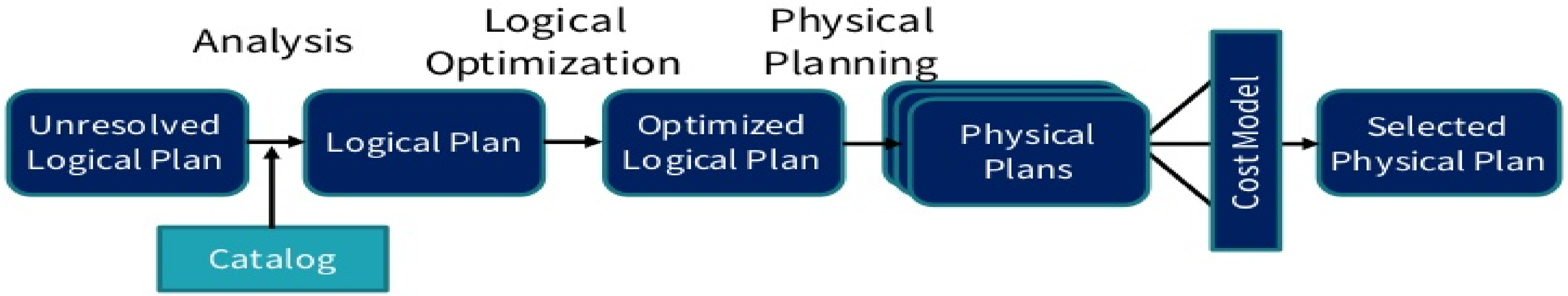
How?

- Write programs using high level programming interfaces
  - Programs are used to describe what data operations are needed without specifying how to execute those operations
  - High level programming interfaces: SQL, DataFrames, and Dataset

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

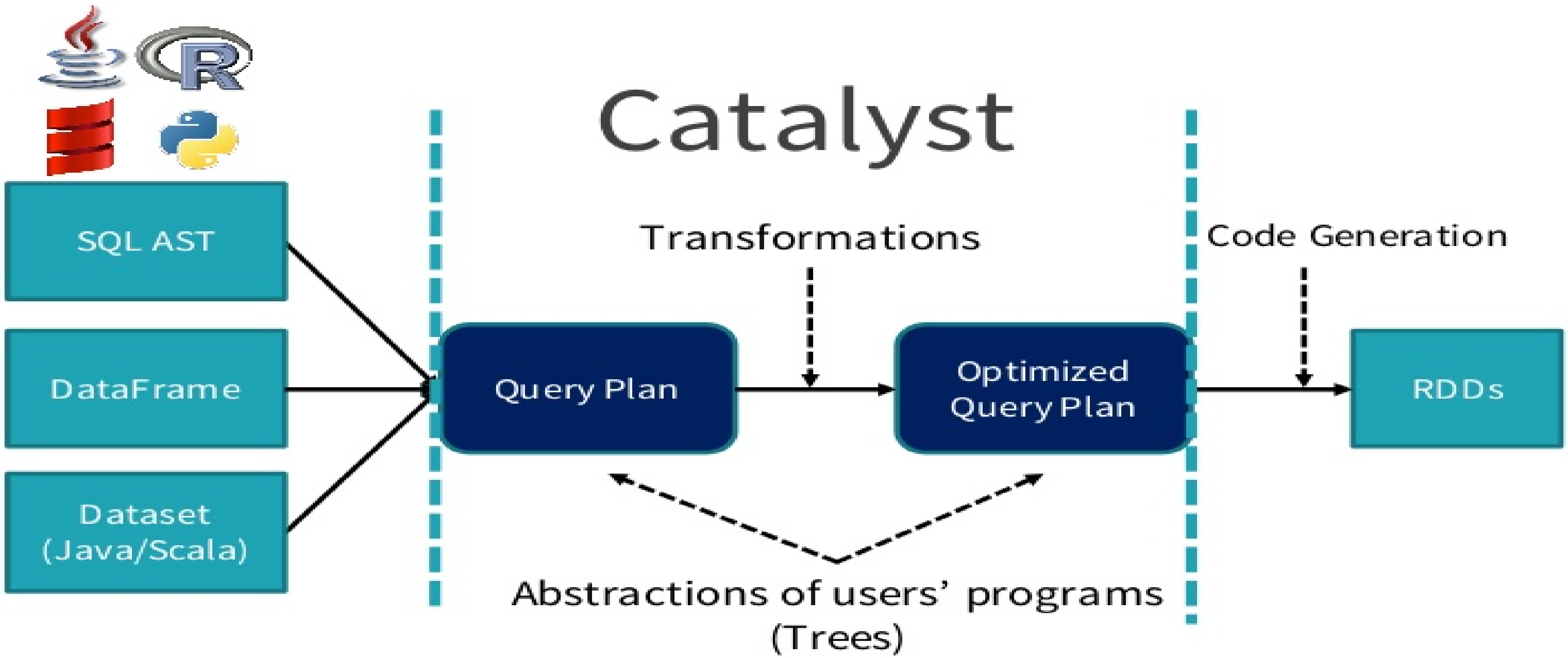
Catalyst:
Apache Spark's Optimizer


Catalyst
SQL AST
DataFrame
Dataset (Java/Scala)
Query Plan
Optimized Query Plan
RDDs
Analysis
Logical Optimization
Physical Planning
Unresolved Logical Plan
Logical Plan
Optimized Logical Plan
Physical Plans
Cost Model
Selected Physical Plan
Catalog

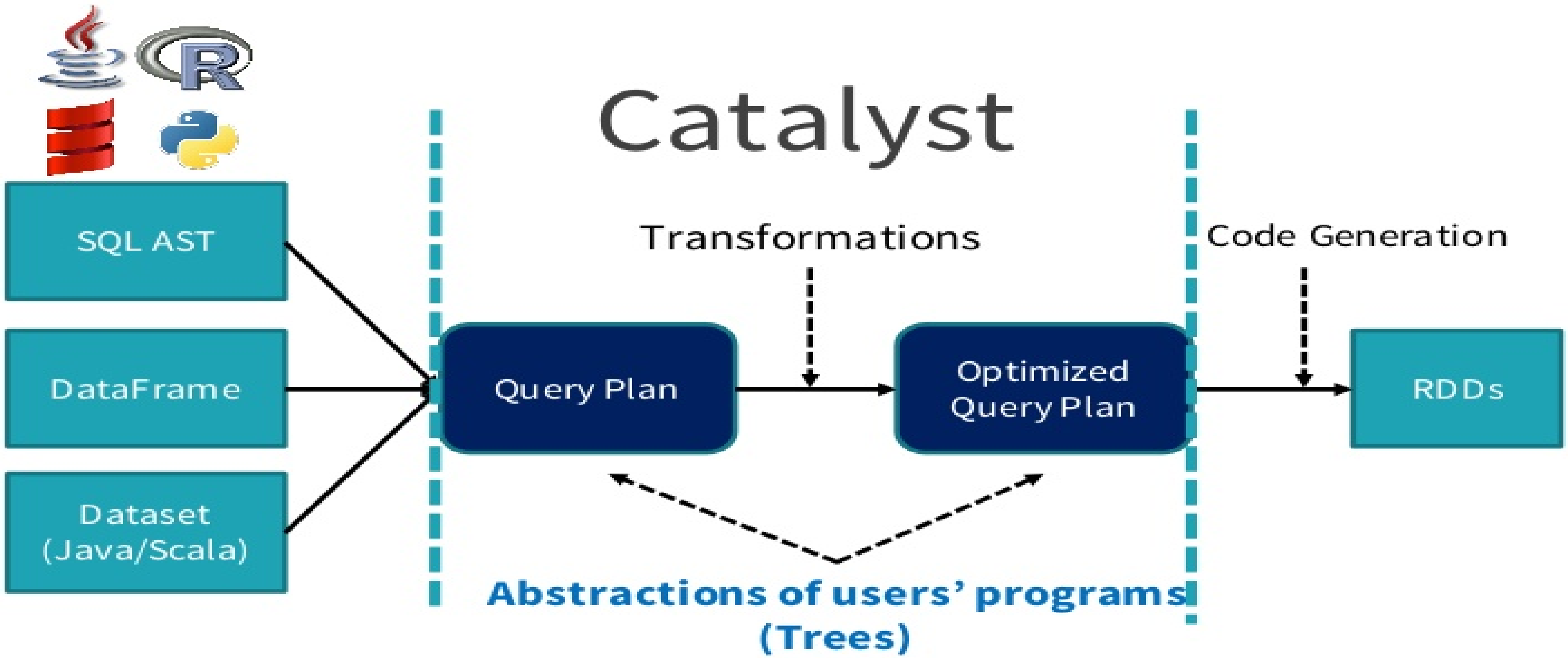


- **Analysis (Rule Executor):** Transforms an Unresolved Logical Plan to a Resolved Logical Plan
  - Unresolved => Resolved: Use Catalog to find where datasets and columns are coming from and types of columns
- **Logical Optimization (Rule Executor)**: Transforms a Resolved Logical Plan to an Optimized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

How Catalyst Works: An Overview



How Catalyst Works: An Overview



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 > 50 * 1000) tmp
```

Trees: Abstractions of Users' Programs

Expression

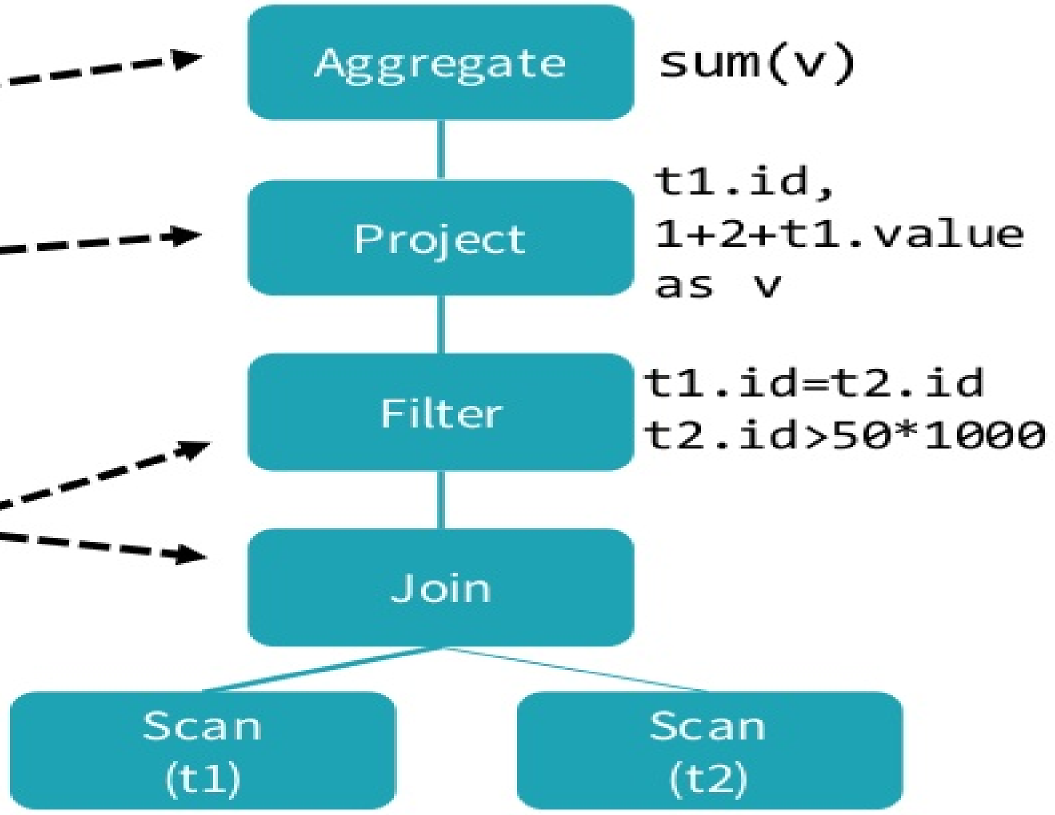
```
SELECT sum(v)
FROM (
  SELECT
    t1.id,
    1 + 2 + t1.value AS v
  FROM t1 JOIN t2
  WHERE
    t1.id = t2.id AND
    t2.id > 50 * 1000) tmp
```

- 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`**)

Trees: Abstractions of Users’ Programs

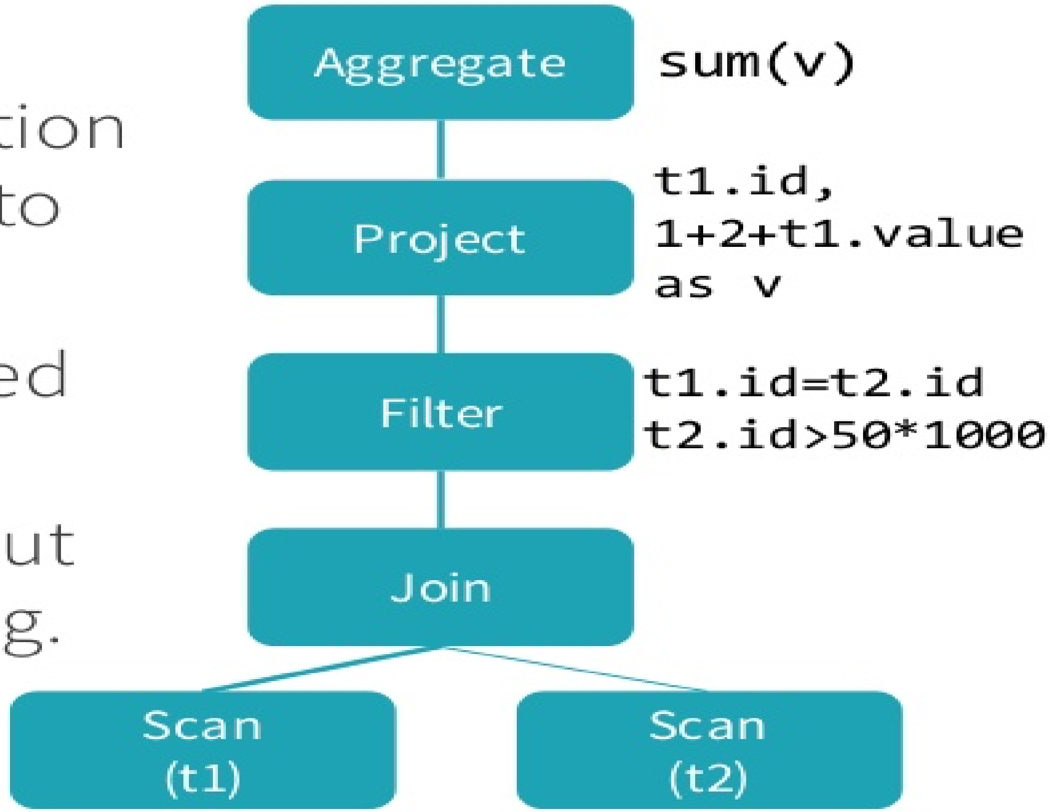
Query Plan

```
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 > 50 * 1000) tmp
```



Logical Plan

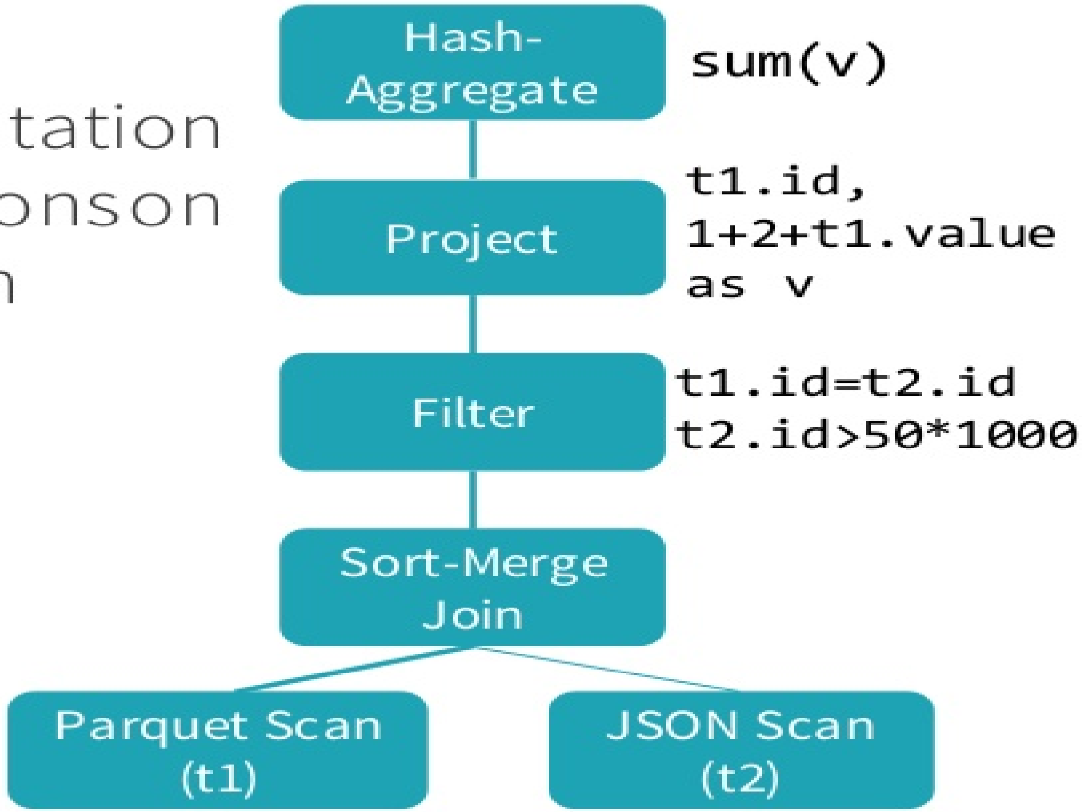
- A Logical Plan describes computation on datasets **without** defining how to conduct the computation
- **output**: a list of attributes generated by this Logical Plan, e.g. [`id, v`]
- **constraints**: a set of invariants about the rows generated by this plan, e.g. `t2.id > 50 * 1000`



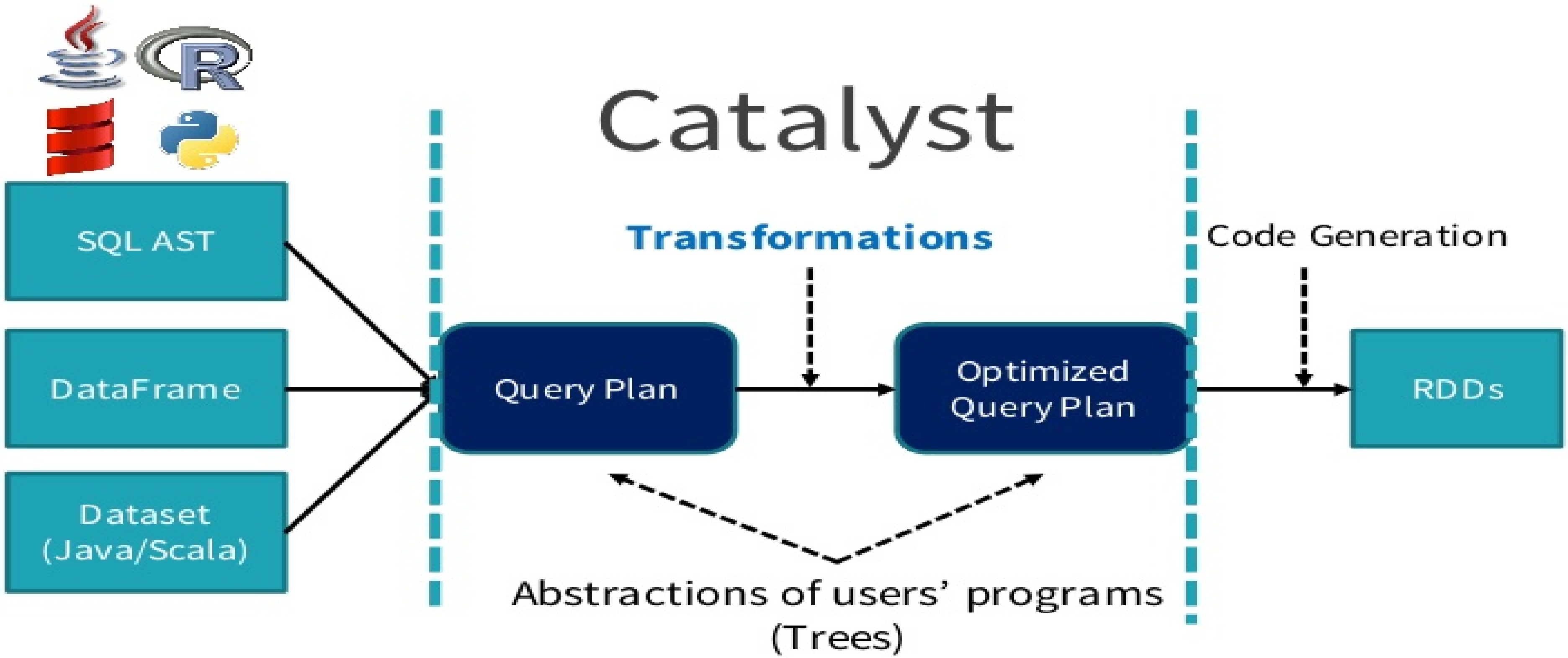
Physical Plan

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

- A Physical Plan is executable



How Catalyst Works: An Overview



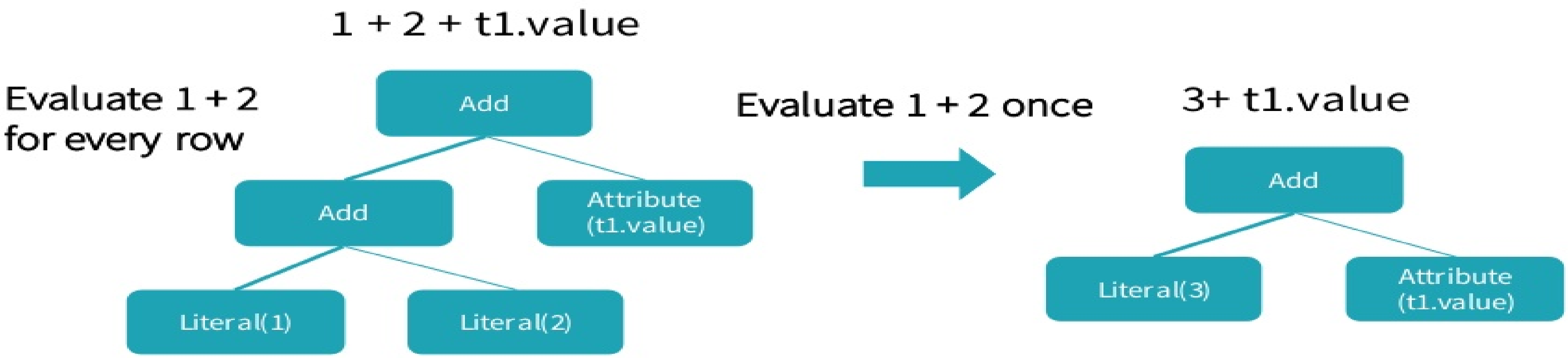
Transformations

- 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

Transform

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



Transform

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

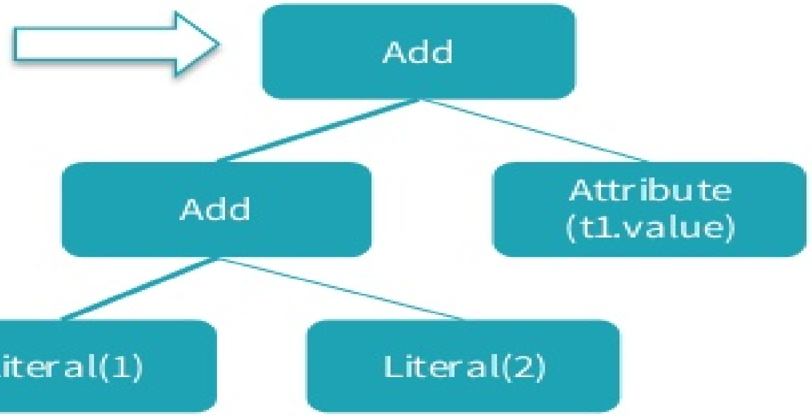
```
val expression: Expression = ...
expression.transform {
  case Add(Literal(x, IntegerType), Literal(y, IntegerType)) =>
    Literal(x + y)
}
```

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

Transform

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

1 + 2 + t1.value

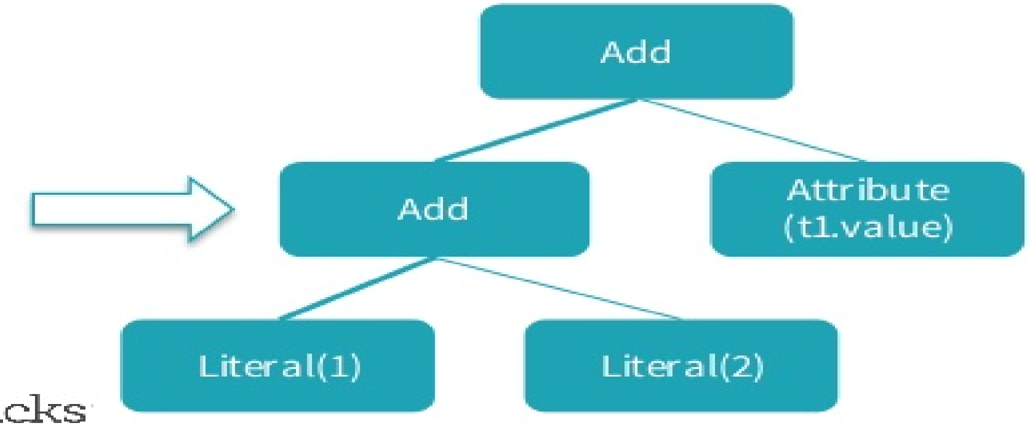


databricks

Transform

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

1 + 2 + t1.value

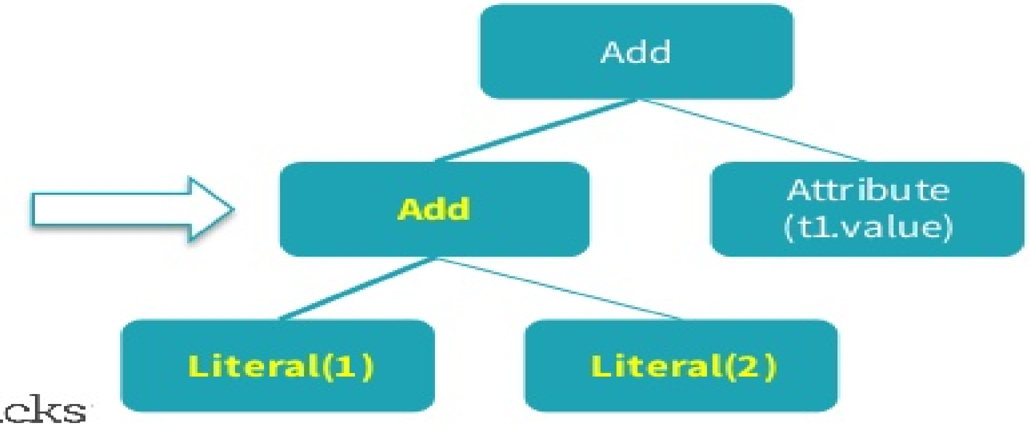


databricks

Transform

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

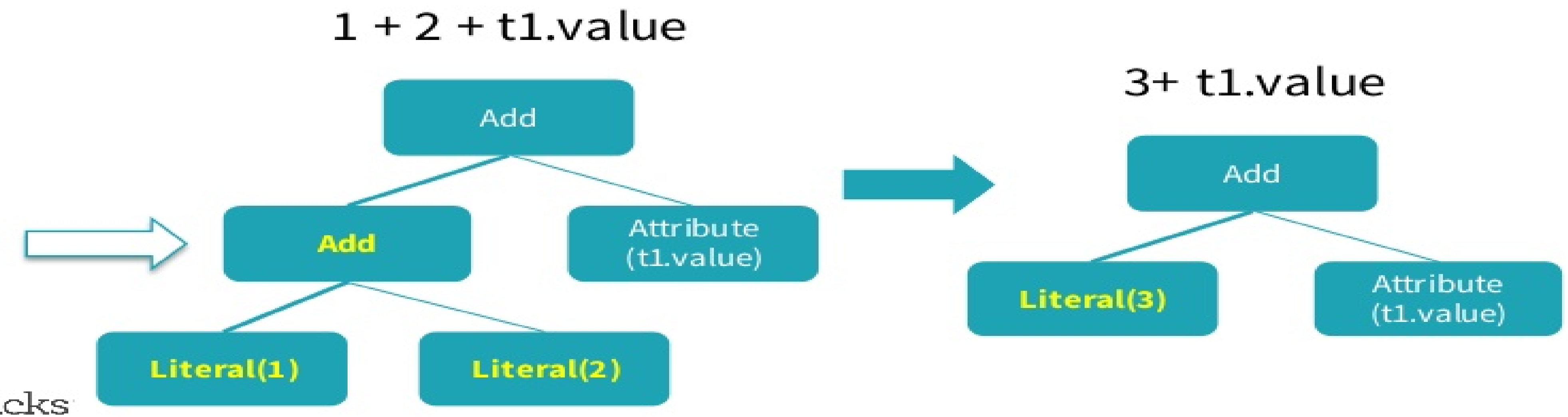
1 + 2 + t1.value



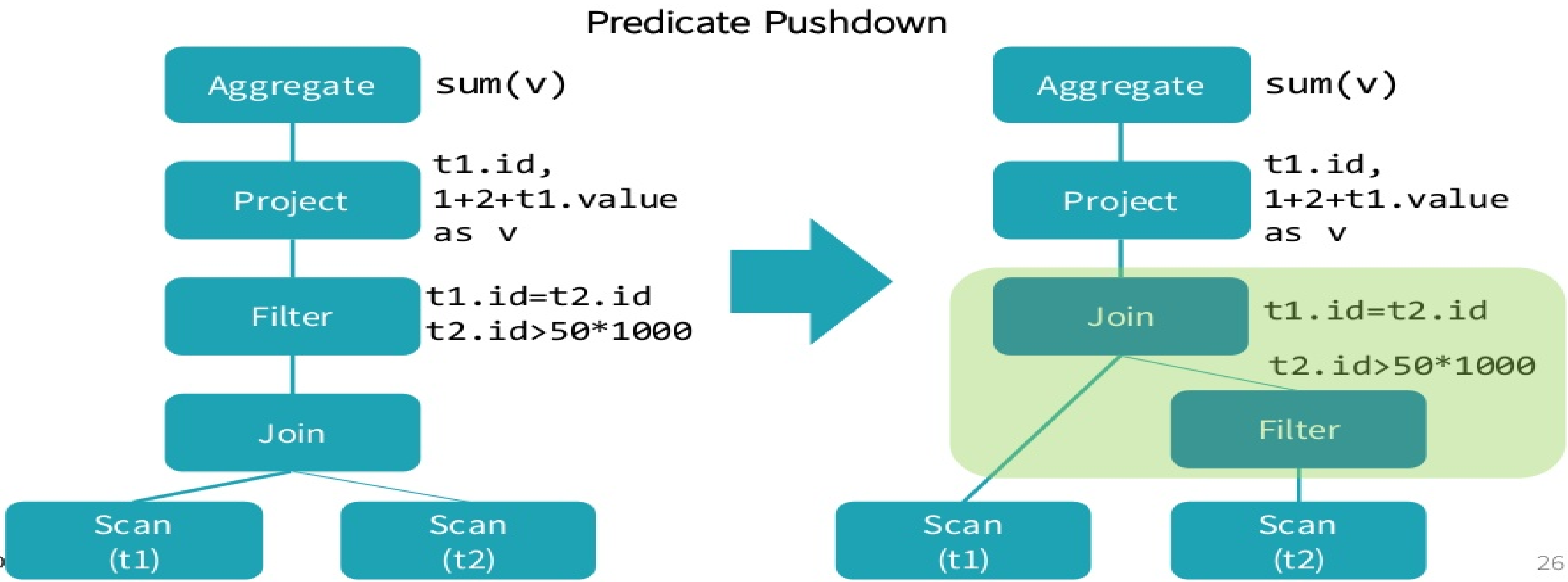
databricks

Transform

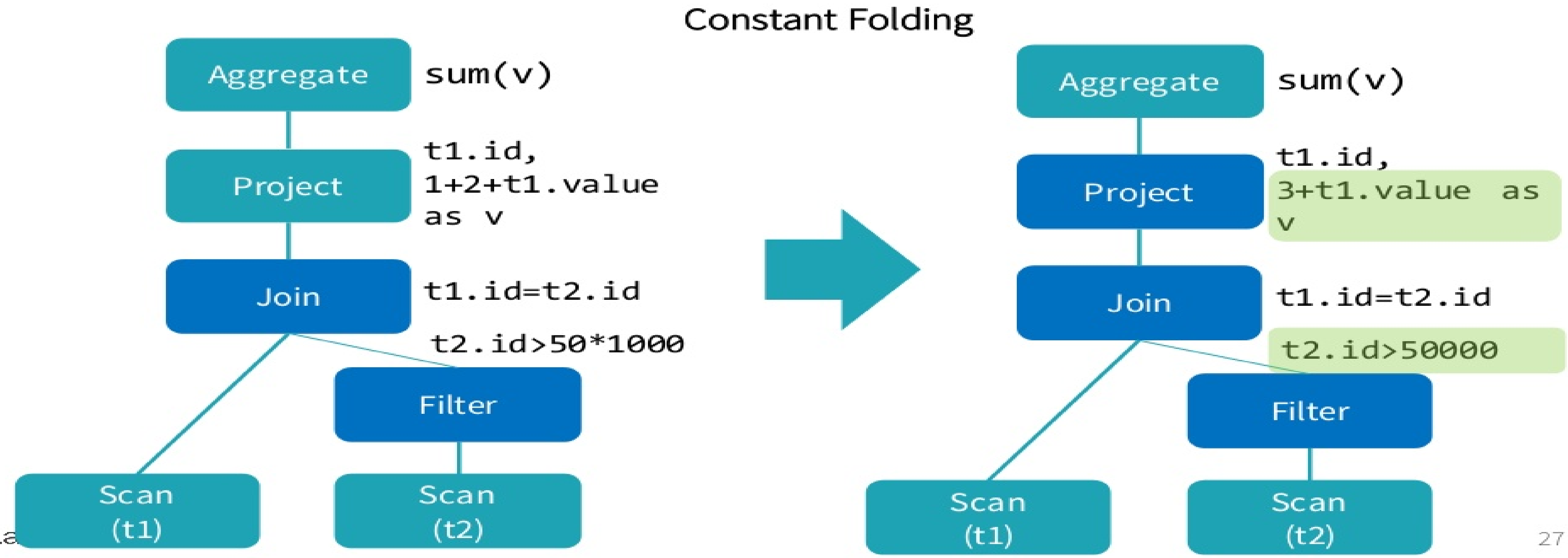
```
val expression: Expression = ...
expression.transform {
  case Add(Literal(x, IntegerType), Literal(y, IntegerType)) =>
    Literal(x + y)
}
```



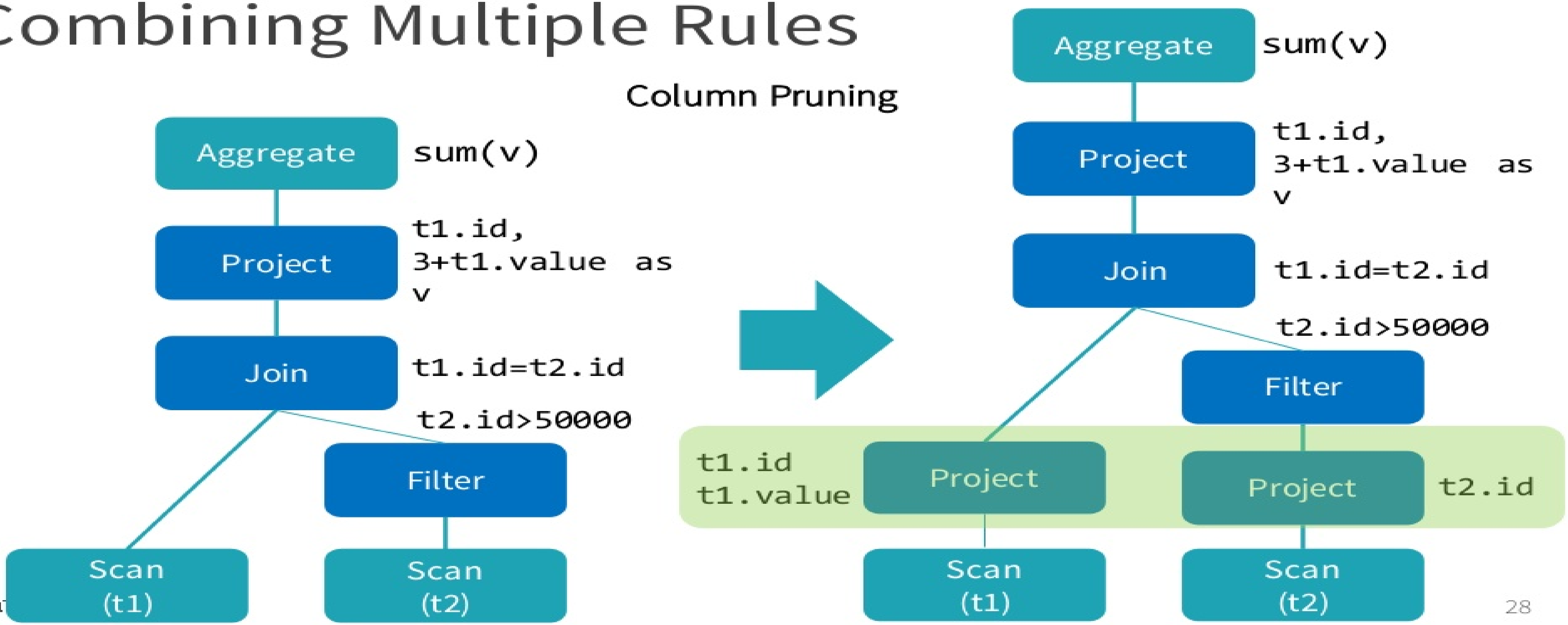
Combining Multiple Rules



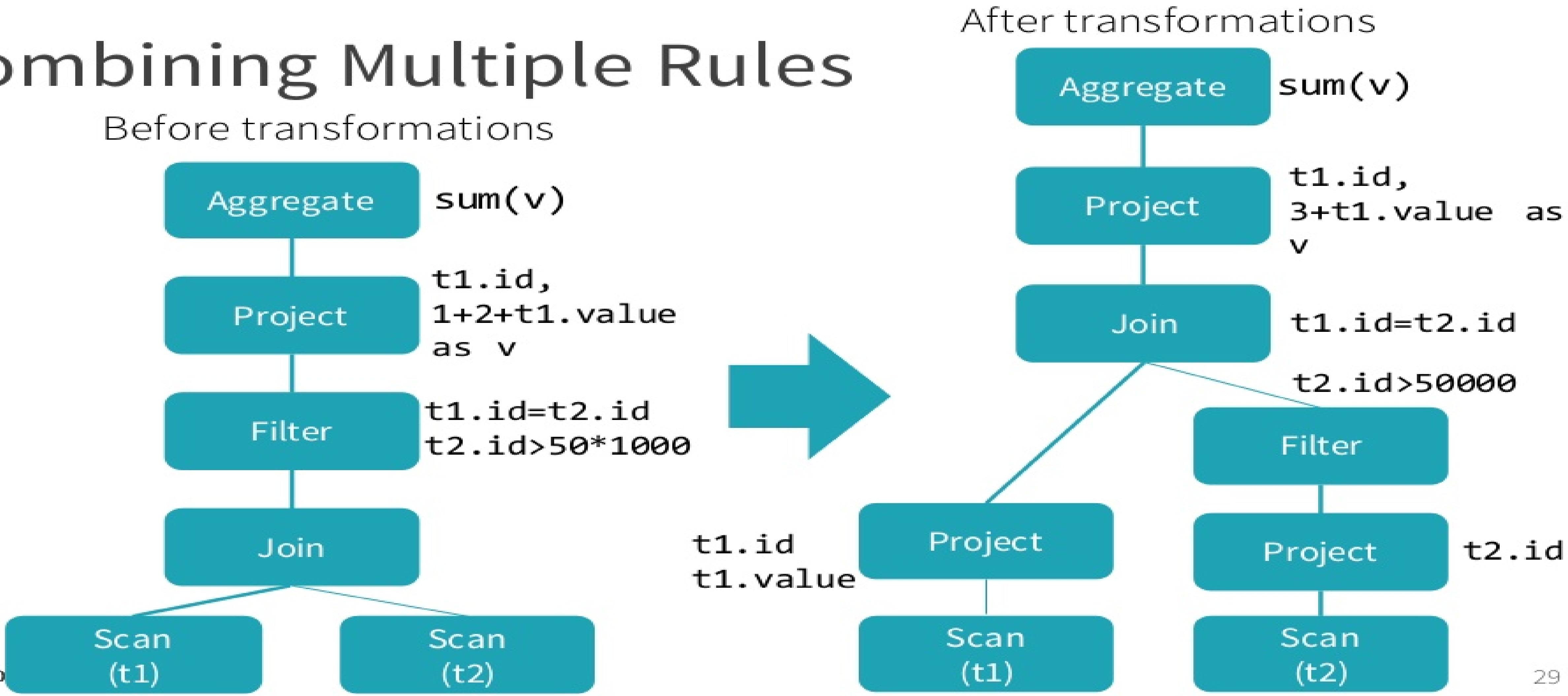
Combining Multiple Rules



Combining Multiple Rules

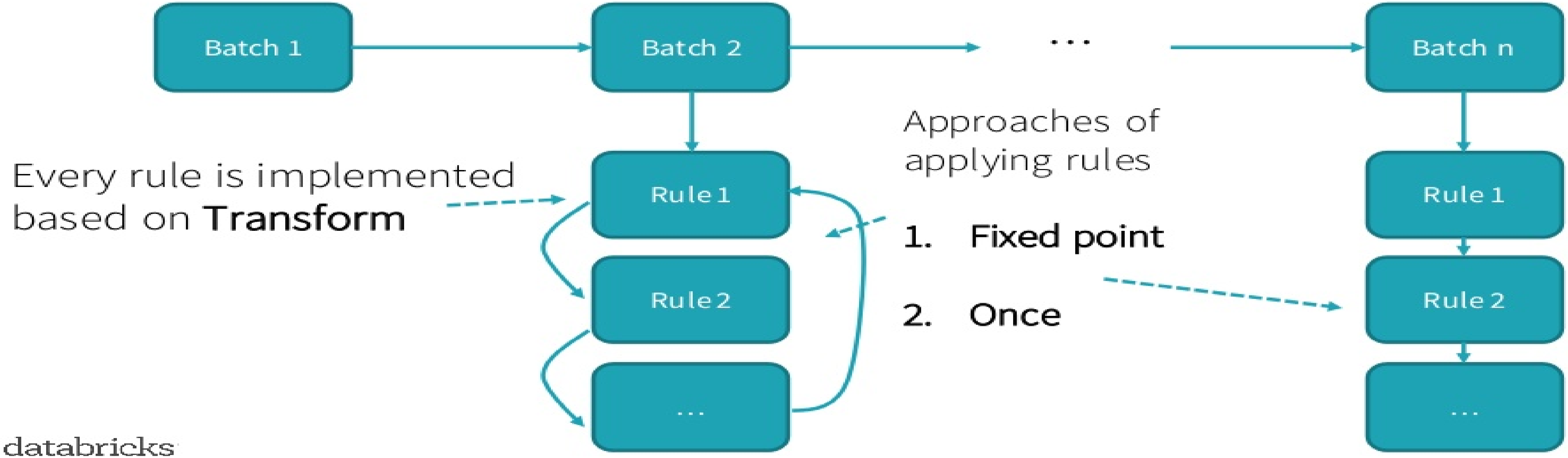


Combining Multiple Rules



Combining Multiple Rules: Rule Executor

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



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

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 Tree to another kind of Tree

```
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
    …
  }
}
```

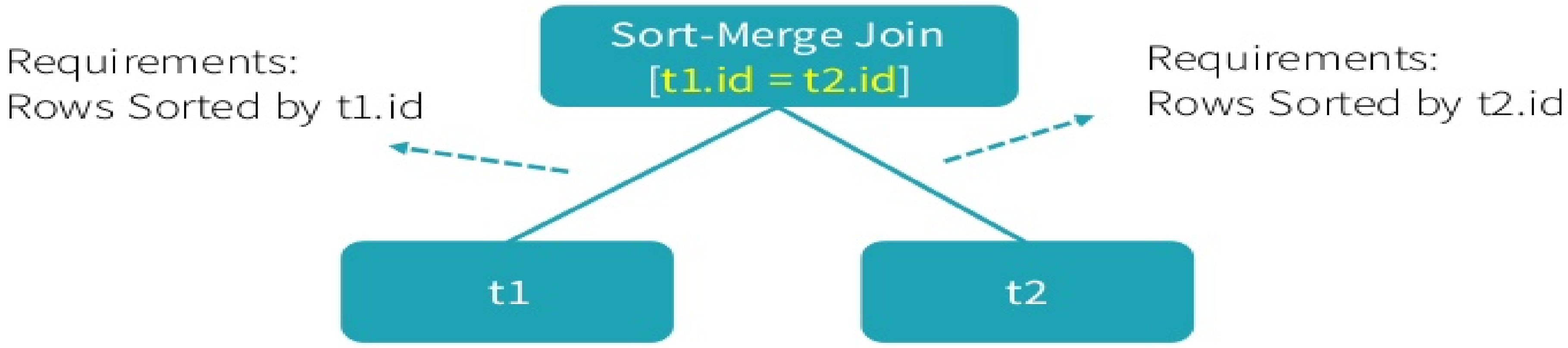
Triggers other Strategies

Spark's Planner

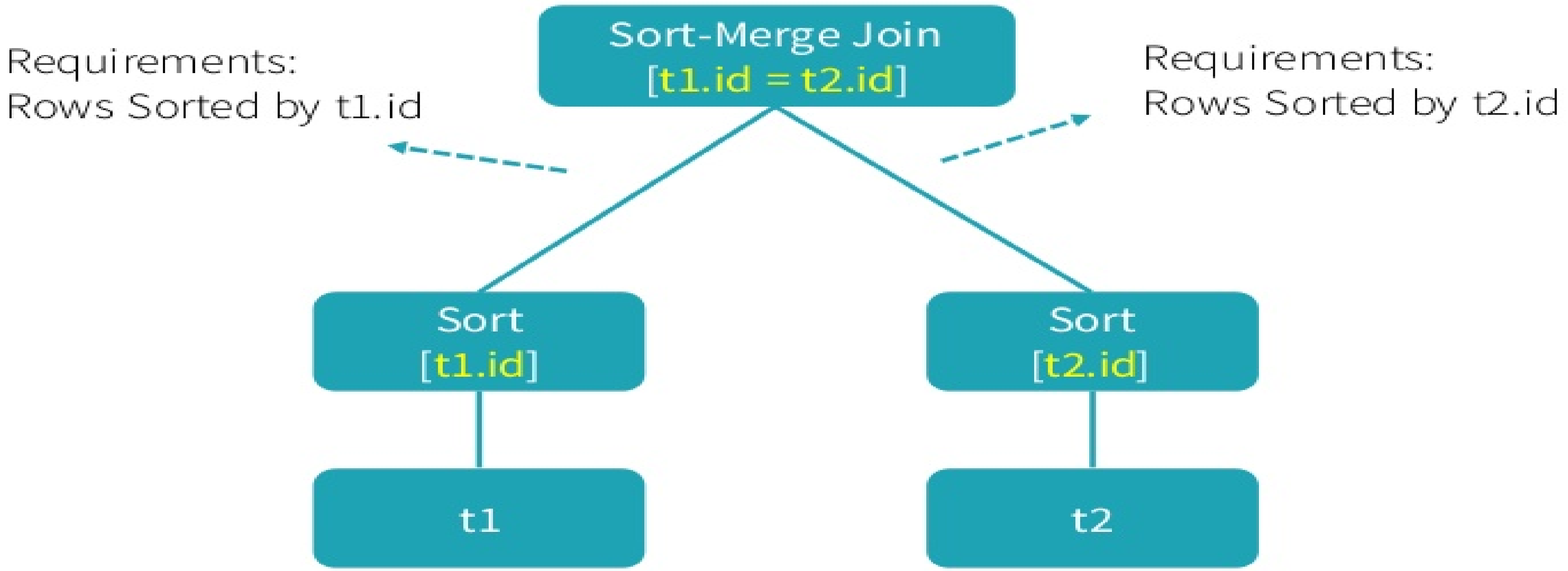
- 1st Phase: Transforms the Logical Plan to the Physical Plan using Strategies

- 2nd Phase: Use a Rule Executor to make the Physical Plan ready for execution
  - Prepare Scalar sub-queries
  - Ensure requirements on input rows
  - Apply physical optimizations

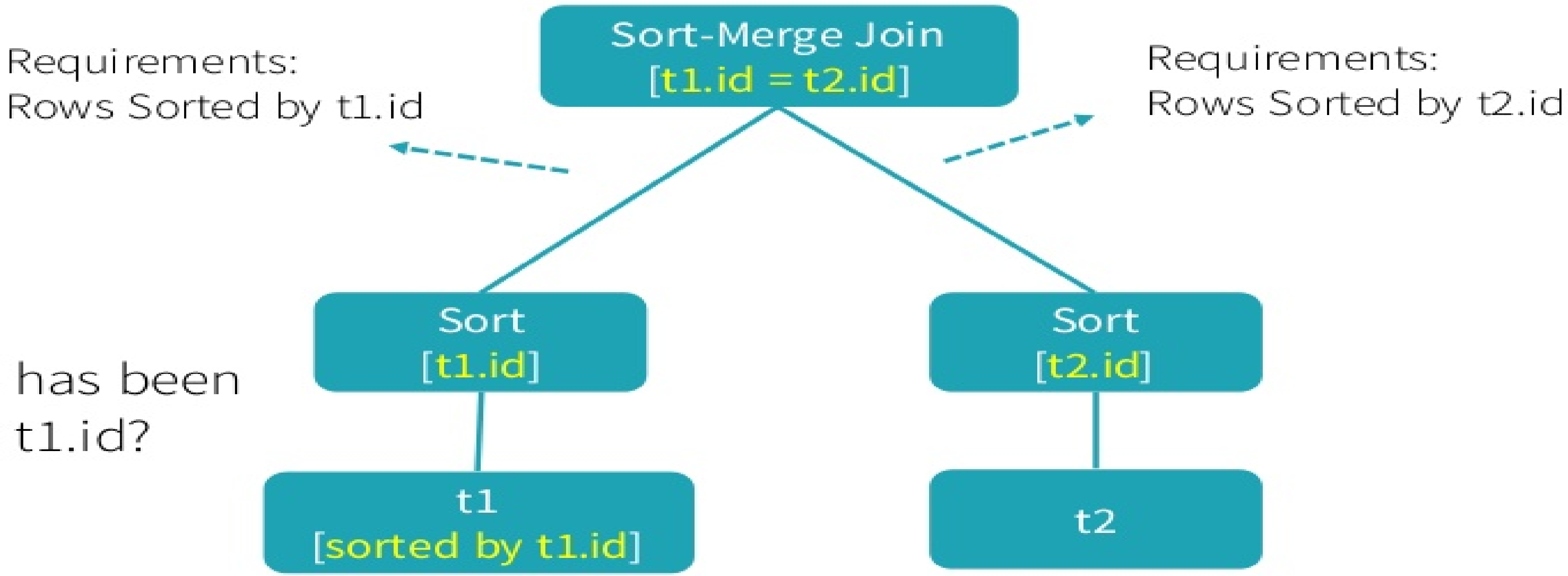
Ensure Requirements on Input Rows



Ensure Requirements on Input Rows

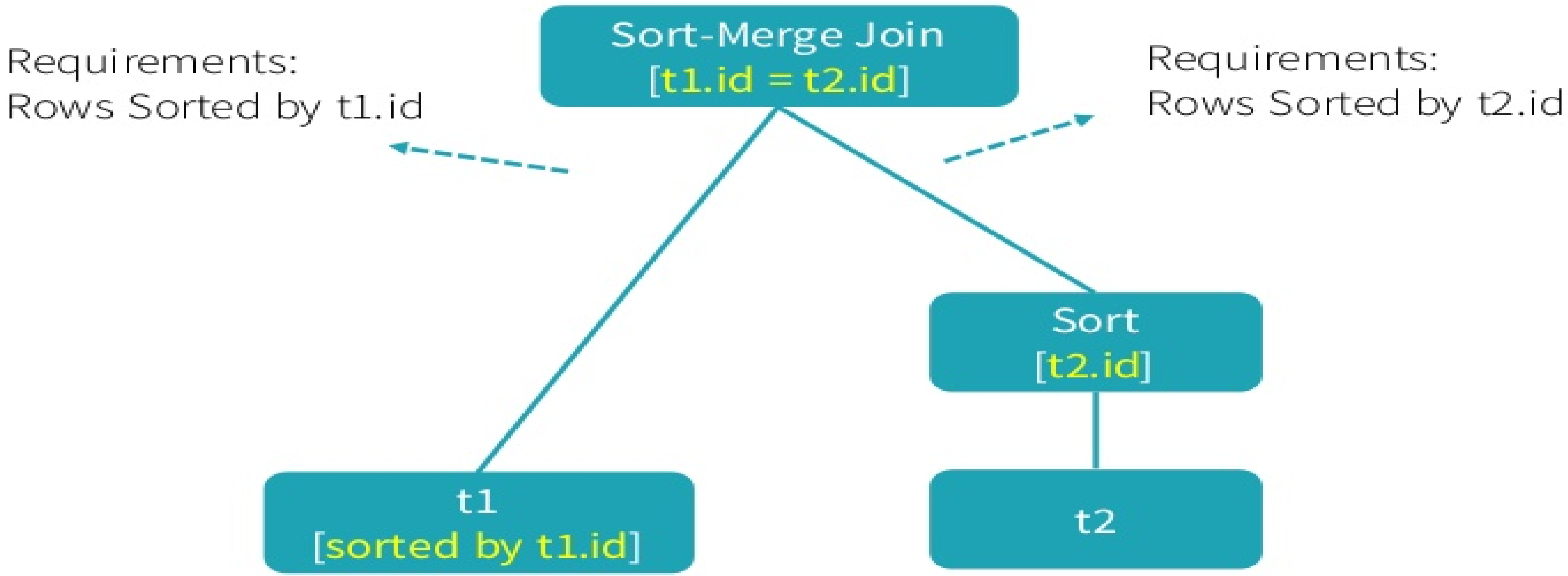


Ensure Requirements on Input Rows



What if t1 has been sorted by t1.id?

Ensure Requirements on Input Rows



Roll your own Planning Rule

Roll your own Planner Rule

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

val tableA = spark.range(100000000).as('a)
val tableB = spark.range(100000000).as('b)

val result = tableA
    .join(tableB, $"a.id" === $"b.id")
    .groupBy()
    .count()
result.count()
```

This takes ~22 Seconds on Databricks Community edition

Roll your own Planner Rule

Can we do better?

Roll your own Planner Rule - Analysis

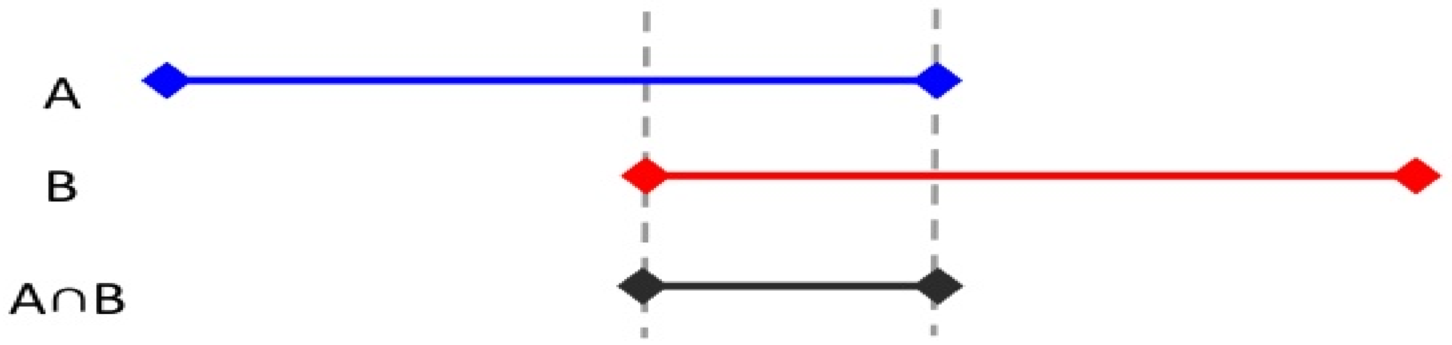
```
== Physical Plan ==
*HashAggregate(keys=[], functions=[count(1)], output=[count#43L])
+- Exchange SinglePartition
   +- *HashAggregate(keys=[], functions=[partial_count(1)], output=[count#48L])
      +- *Project
         +- *SortMergeJoin [id#21L], [id#25L], Inner
            :- *Sort [id#21L ASC], false, 0
            :  +- Exchange hashpartitioning(id#21L, 200)
            :     +- *Range (0, 100000000, step=1, splits=Some(8))
            +- *Sort [id#25L ASC], false, 0
               +- Exchange hashpartitioning(id#25L, 200)
                  +- *Range (0, 100000000, step=1, splits=Some(8))
```

databricks

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



databricks

Roll your own Planner Rule - Matching

```
case object IntervalJoin extends Strategy with Serializable {
  def apply(plan: LogicalPlan): Seq[SparkPlan] = plan match {
    case Join(
           Range(start1, end1, 1, part1, Seq(o1)),
           Range(start2, end2, 1, part2, Seq(o2)),
           Inner,
           Some(EqualTo(e1, e2)))
               if ((o1 semanticEquals e1) && (o2 semanticEquals e2)) ||
                  ((o1 semanticEquals e2) && (o2 semanticEquals e1)) =>
           // Rule...
    case _ => Nil
  }
}
```

Roll your own Planner Rule - Body

```
if astart1 <= end2) && (end1 >= end2)) {
  val start = math.max(start1, start2)
  val end = math.min(end1, end2)
  val part = math.max(part1.getOrElse(200), part2.getOrElse(200))
  val result = RangeExec(Range(start, end, 1, part, o1 :: Nil))
  val twoColumns = ProjectExec(
    Alias(o1, o1.name)(exprId = o1.exprId) :: Nil,
    result)
  twoColumns :: Nil
} else {
  Nil
}
```

databricks

Roll your own Planner Rule

Hook it up with Spark

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

Use it

```
result.count()
```

This now takes 0.46 seconds to complete

Roll your own Planner Rule

```
== Physical Plan ==
*HashAggregate(keys=[], functions=[count(1)],
output=[count#43L])
+- Exchange SinglePartition
   +- *HashAggregate(keys=[], functions=[partial_count(1)],
output=[count#48L])
      +- *Project
         +- *Project [id#21L AS id#21L]
            +- *Range (0, 100000000, step=1, splits=Some(8))
```

Community Contributed Transformations

SPARK-3462 push down filters and projections into Unions #2345

Closed **koeninger** wants to merge 3 commits into `apache:master` from `mediacrossinginc:SPARK-3462`

Conversation 15 Commits 3 Files changed 2 +110 −0

Showing 2 changed files with **110 additions** and **0 deletions**. Unified Split

110 line patch took this user's query from "never finishing" to 200s.

Overall 200+ people have contributed to the analyzer/optimizer/planner in the last 2 years.

Where to Start

- Source Code:
  - Trees: TreeNode, Expression, Logical Plan, and Physical Plan
  - Transformations: Analyzer, Optimizer, and Planner
- Check out previous pull requests
- Start to write code using Catalyst
- Open a pull request!

Try Apache Spark with Databricks

- Try latest version of Apache Spark
  http://databricks.com/try

Questions?

I will be available in the Databricks booth (D1) afterwards

@Westerflyer

databricks