Inside Hive (for beginners)

Takeshi NAKANO / Recruit Co. Ltd.

Why?

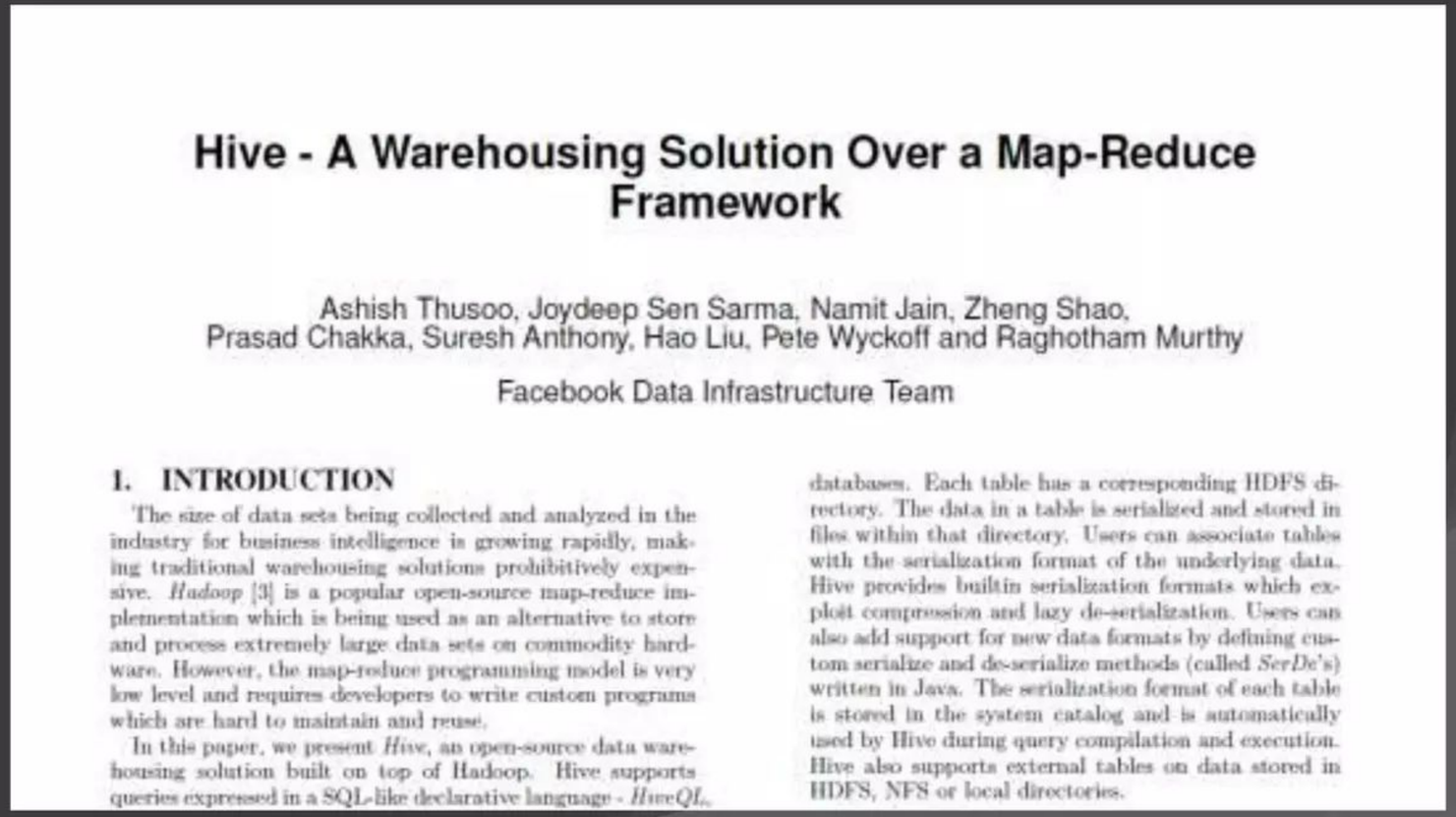
- Hive is good tool for non-specialist!
- The number of M/R controls the Hive processing time.

  ↓
- How can we reduce the number?
- What can we do for this on writing HiveQL?

  ↓
- How does Hive convert HiveQL to M/R jobs?
- On this, what optimizing processes are adopted?

Don’t you have..

- This fb’s paper has a lot of information!
- But this is a little old..



Hive - A Warehousing Solution Over a Map-Reduce Framework

Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao,
Prasad Chakka, Suresh Anthony, Hao Liu, Pete Wyckoff and Raghotham Murthy

Facebook Data Infrastructure Team

1. INTRODUCTION

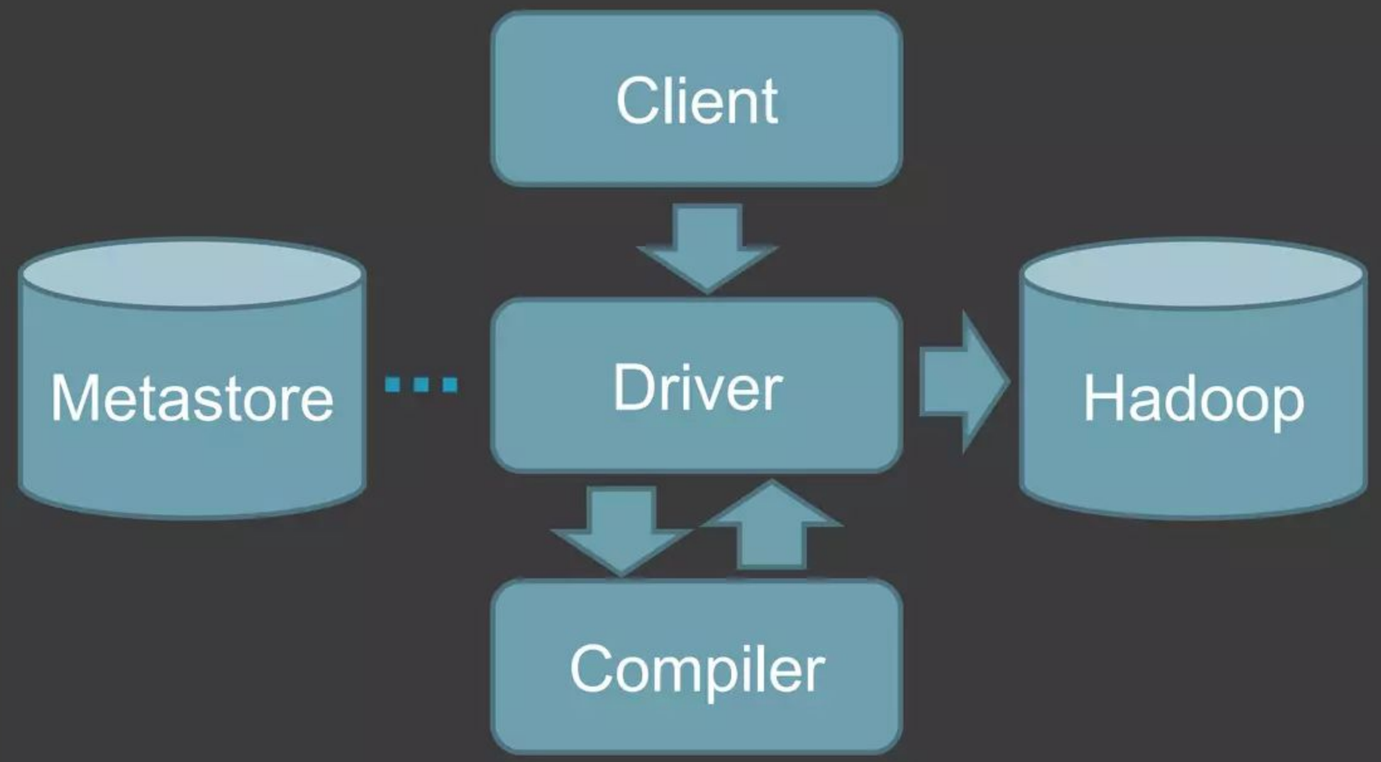
The size of data sets being collected and analyzed in the industry for business intelligence is growing rapidly, making traditional warehousing solutions prohibitively expensive. *Hadoop* [3] is a popular open-source map-reduce implementation which is being used as an alternative to store and process extremely large data sets on commodity hardware. However, the map-reduce programming model is very low level and requires developers to write custom programs which are hard to maintain and reuse.

In this paper, we present *Hive*, an open-source data warehousing solution built on top of Hadoop. Hive supports queries expressed in a SQL-like declarative language - *HiveQL*,

databases. Each table has a corresponding HDFS directory. The data in a table is serialized and stored in files within that directory. Users can associate tables with the serialization format of the underlying data. Hive provides builtin serialization formats which exploit compression and lazy de-serialization. Users can also add support for new data formats by defining custom serialize and de-serialize methods (called *SerDe*'s) written in Java. The serialization format of each table is stored in the system catalog and is automatically used by Hive during query compilation and execution. Hive also supports external tables on data stored in HDFS, NFS or local directories.

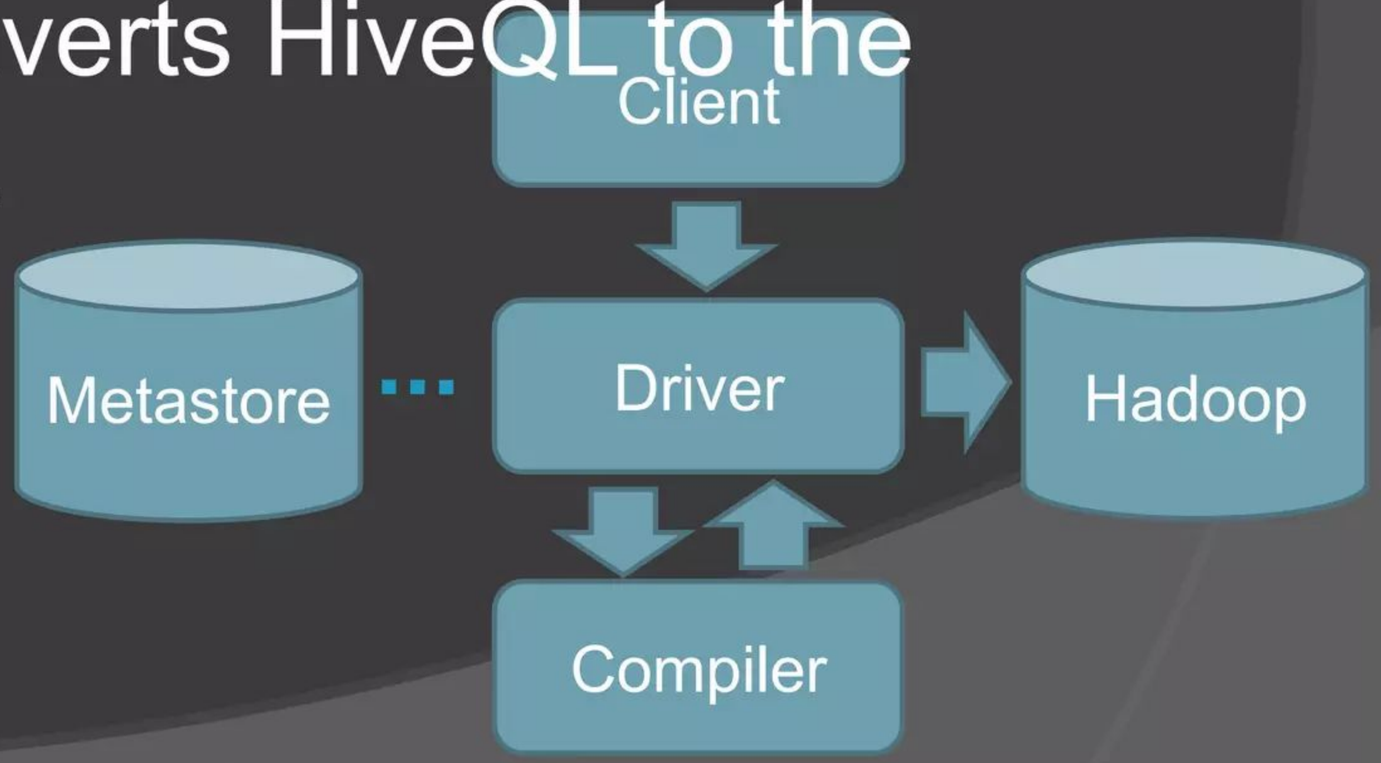
- Component Level Analysis

Hive Architecture / Exec Flow



Hive Workflow

- Hive has the operators which are minimum processing units.
- The process of each operator is done with HDFS operation or M/R jobs.
- The compiler converts HiveQL to the sets of operators.



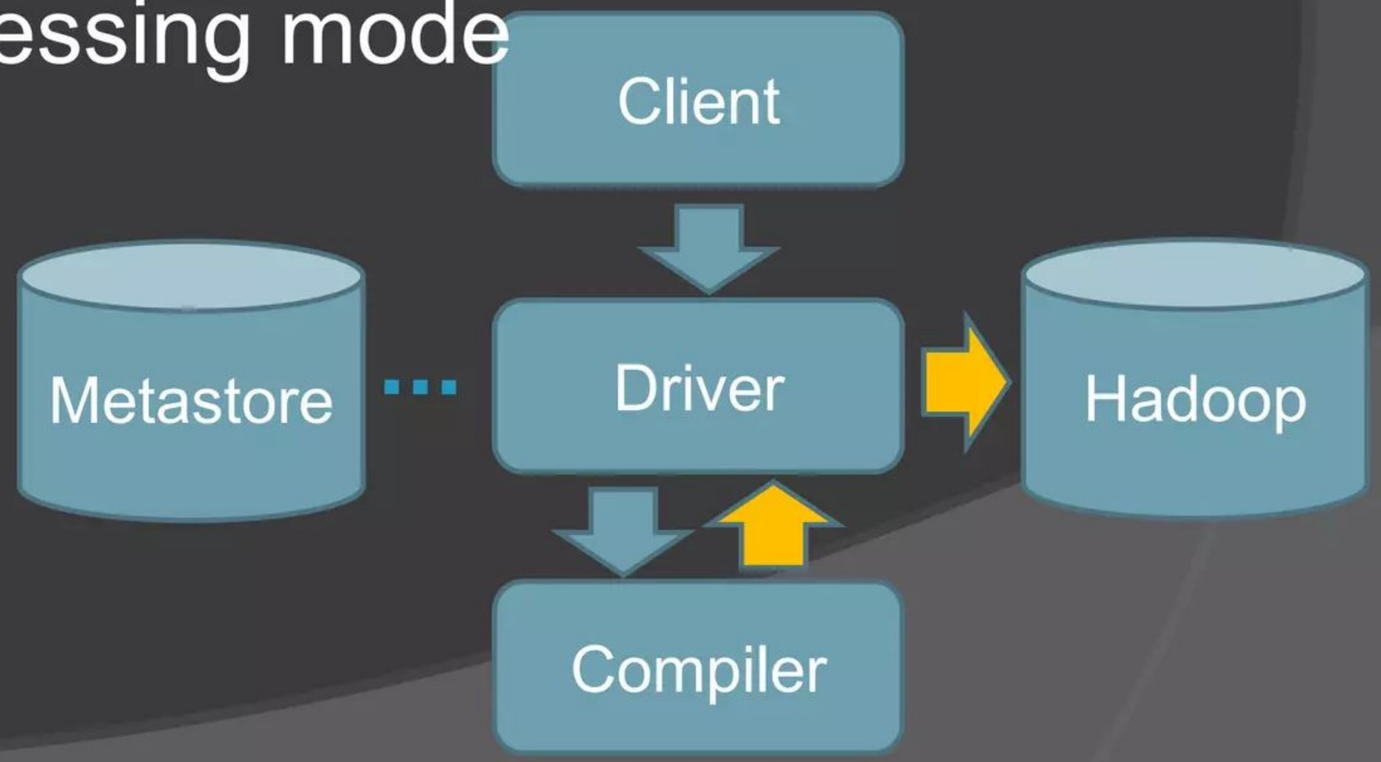
Hive Workflow

- Operators

Operators	Descriptions
TableScanOperator	Read the data from table
ReduceSinkOperator	Build the result data to Reducer
JoinOperator	Join 2 data
SelectOperator	Reduce the output columns.
FileSinkOperator	Build the result data to output (file).
FilterOperator	Filter the input data.
GroupByOperator	. .
MapJoinOperator	. .
LimitOperator	. .
UnionOperator	. .

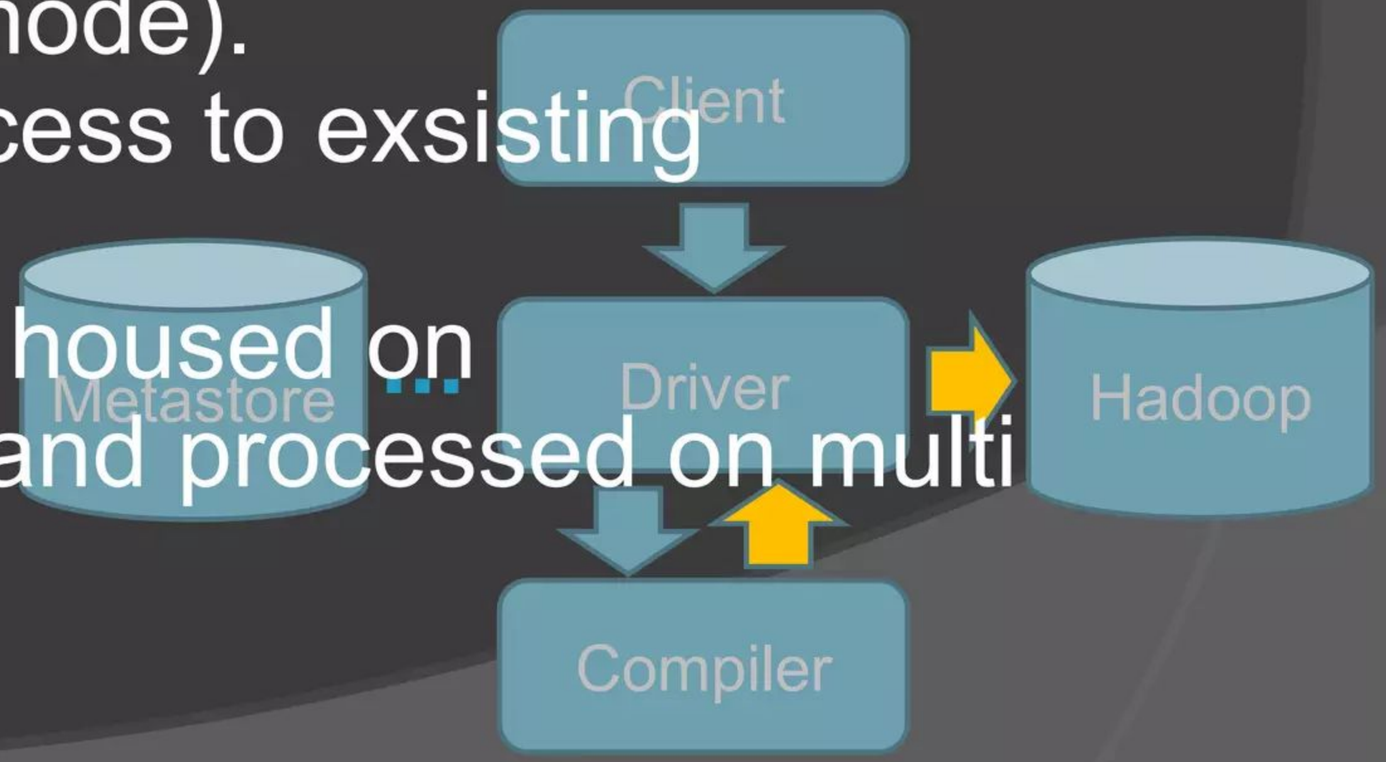
Hive Workflow

- For M/R processing, Hive uses ExecMaper and ExecReducer.
- On processing, we have 2 modes.
  1. Local processing mode
  2. Distributed processing mode

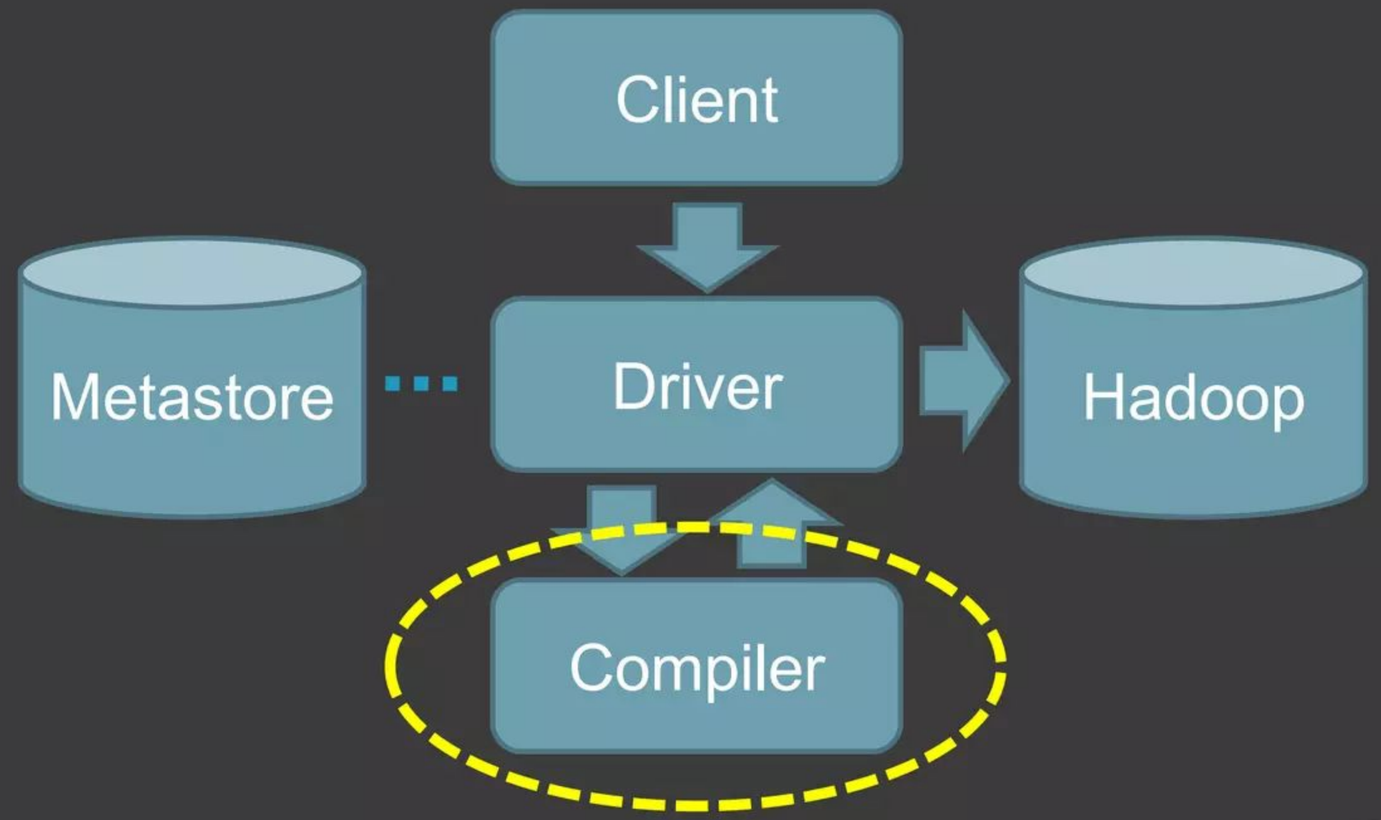


Hive Workflow

- On 1(Local mode)
  Hive fork the process with hadoop command.
  The plan.xml is made just on 1 and the single node processes this.
- On 2(Distributed mode).
  Hive send the process to exsisting JobTracker.
  The information is housed on Metastore ... DistributedCache and processed on multi nodes.



- Compiler : How to Process HiveQL



“Plumbing” of HIVE compiler

Parser

- Convert into Parse Tree Representation

Semantic Analyzer

- Convert into block-base internal query representation

Logical Plan Generator

- Convert into internal query representation

“Plumbing” of HIVE compiler

Logical Optimizer

- Rewrite plans into more optimized plans

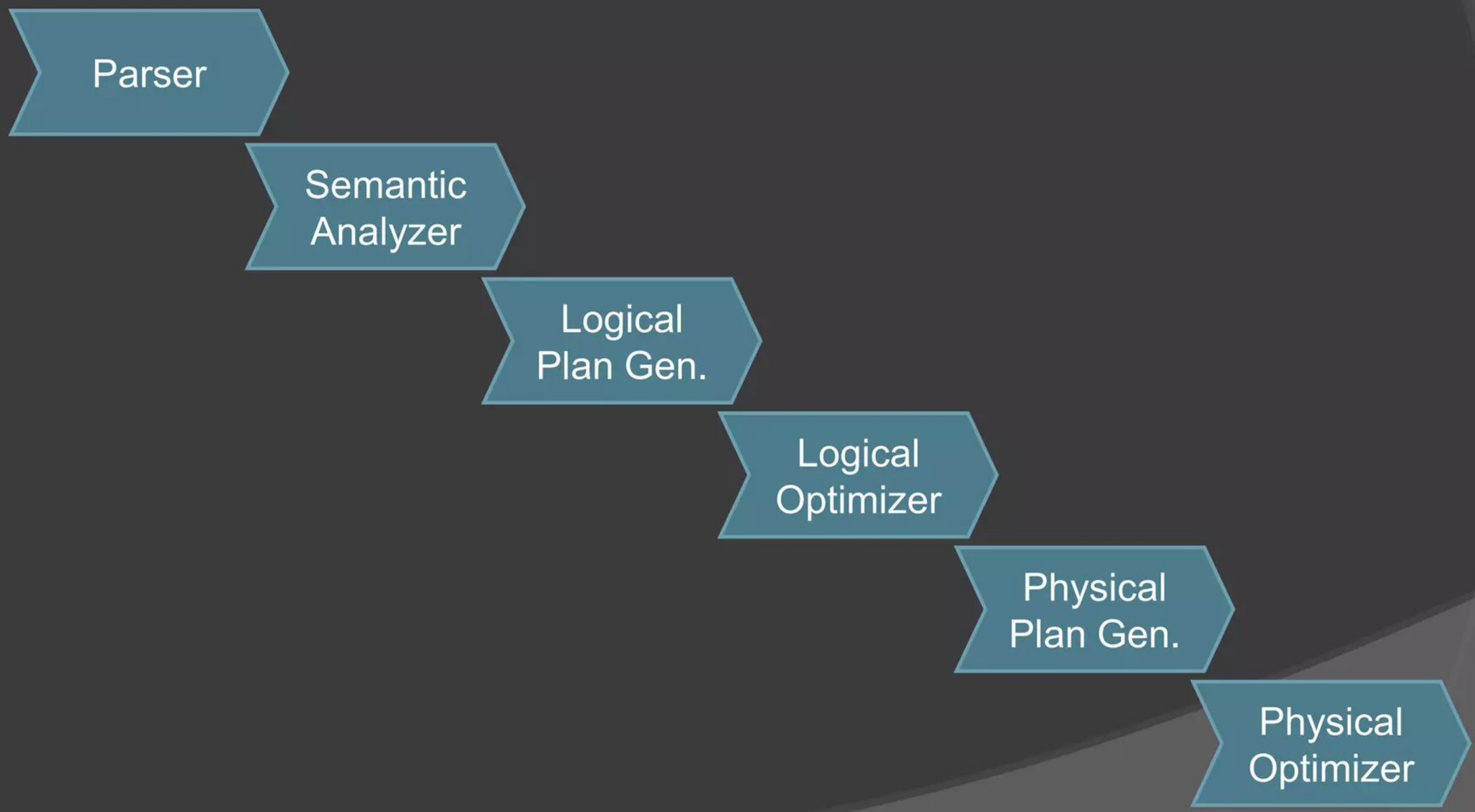
Physical Plan Generator

- Convert into physical plans (M/R jobs)

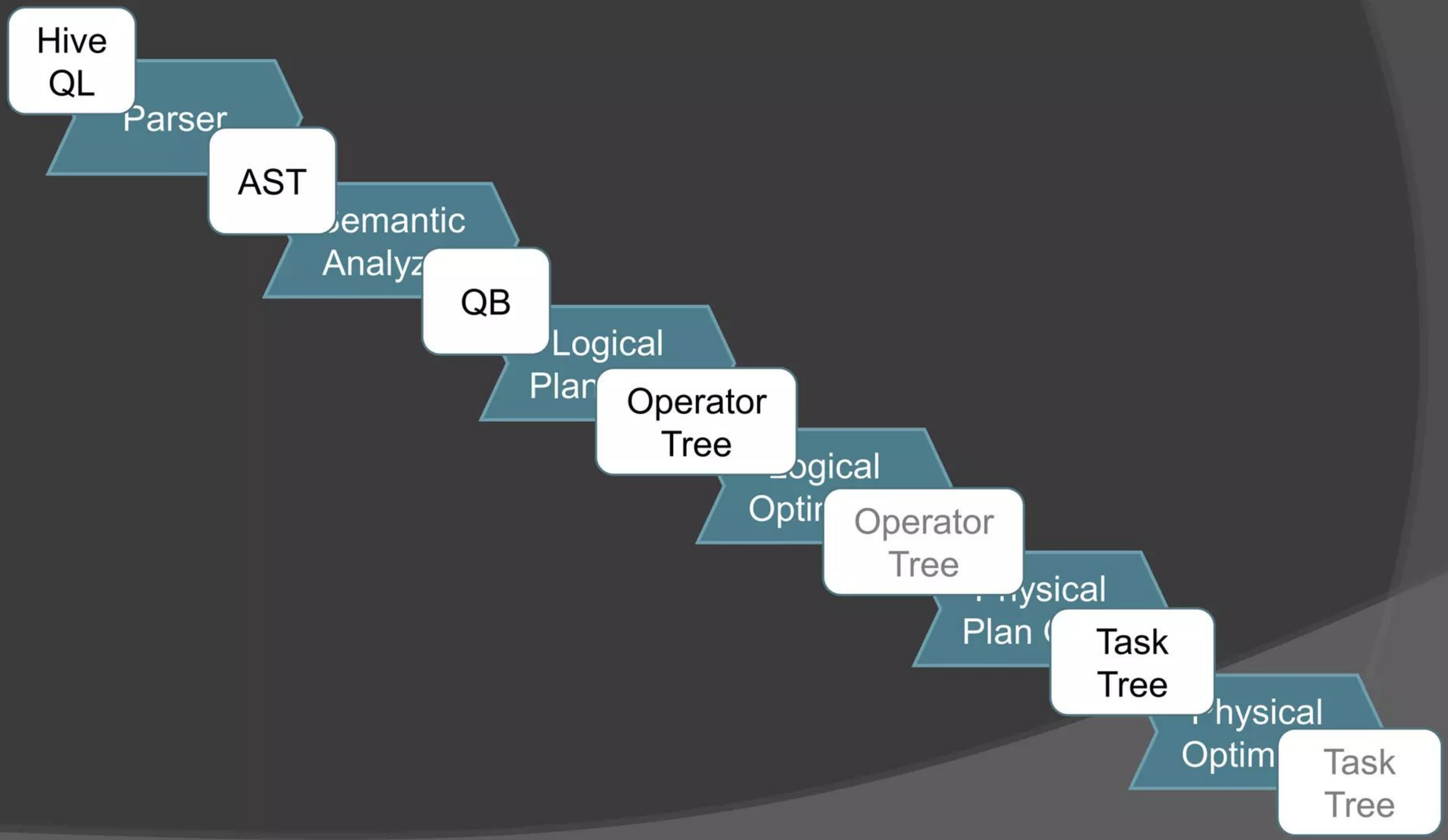
Physical Optimizer

- Adopt join strategy

Compiler Overview



Compiler Overview



Parser

Hive QL → AST

Hive QL

```
INSERT OVERWRITE TABLE access_log_temp2
 SELECT a.user, a.prono, p.maker, p.price
 FROM access_log_hbase a JOIN product_hbase p ON (a.prono = p.prono);
```

AST

```
TOK_QUERY
 + TOK_FROM
    + TOK_JOIN
       + TOK_TABREF
          + TOK_TABNAME
             + "access_log_hbase"
          + a
       + TOK_TABREF
          + TOK_TABNAME
             + "product_hbase"
          + "p"
    + "="
       + "."
          + TOK_TABLE_OR_COL
             + "a"
          + "access_log_hbase"
       + "."
          + TOK_TABLE_OR_COL
             + "p"
          + "prono“
```

```
+ TOK_INSERT
   + TOK_DESTINATION
      + TOK_TAB
         + TOK_TABNAME
            + "access_log_temp2"
   + TOK_SELECT
      + TOK_SELEXPR
         + "."
            + TOK_TABLE_OR_COL
               + "a"
            + "user"
      + TOK_SELEXPR
         + "."
            + TOK_TABLE_OR_COL
               + "a"
            + "prono"
      + TOK_SELEXPR
         + "."
            + TOK_TABLE_OR_COL
               + "p"
            + "maker"
      + TOK_SELEXPR
         + "."
            + TOK_TABLE_OR_COL
               + "p"
            + "price"
```

Parser → Semantic Analyzer → P… → Physical Optimizer

Parser

SQL → AST

SQL

```
INSERT OVERWRITE TABLE access_log_temp2
 SELECT a.user, a.prono, p.maker, p.price
 FROM access_log_hbase a JOIN product_hbase p ON (a.prono = p.prono);
```

AST

```
TOK_QUERY
 + TOK_FROM
    + TOK_JOIN
       + TOK_TABREF
          + TOK_TABNAME
             + "access_log_hbase"
          + a
       + TOK_TABREF
          + TOK_TABNAME
             + "product_hbase"
          + "p"
    + "="
       + "."
          + TOK_TABLE_OR_COL
             + "a"
          + "access_log_hbase"
       + "."
          + TOK_TABLE_OR_COL
             + "p"
          + "prono“
```

1

```
 + TOK_INSERT
    + TOK_DESTINATION
       + TOK_TAB
          + TOK_TABNAME
             + "access_log_temp2"
    + TOK_SELECT
       + TOK_SELEXPR
          + "."
             + TOK_TABLE_OR_COL
                + "a"
             + "user"
       + TOK_SELEXPR
          + "."
             + TOK_TABLE_OR_COL
                + "a"
             + "prono"
       + TOK_SELEXPR
          + "."
             + TOK_TABLE_OR_COL
                + "p"
             + "maker"
       + TOK_SELEXPR
          + "."
             + TOK_TABLE_OR_COL
                + "p"
             + "price"
```

2

3

Parser → Semantic Analyzer → P… → Physical Optimizer

Semantic Analyzer (1/2)

AST → QB

AST

```
+ TOK_FROM
   + TOK_JOIN
      + TOK_TABREF
         + TOK_TABNAME
            + "access_log_hbase"
         + a
      + TOK_TABREF
         + TOK_TABNAME
            + "product_hbase"
         + "p"
   + "="
      + "."
         + TOK_TABLE_OR_COL
            + "a"
         + "access_log_hbase"
      + "."
         + TOK_TABLE_OR_COL
            + "p"
         + "prono“
```

1

QB

MetaData

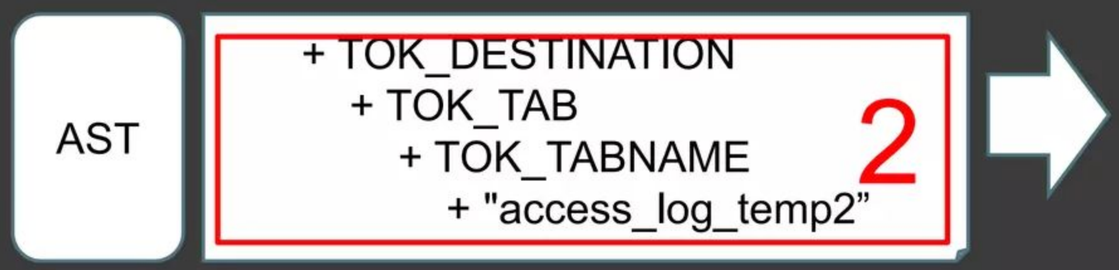
Alias To Table Info
“a”=Table Info(“access_log_hbase”)
“p”=Table Info(“product_hbase”)

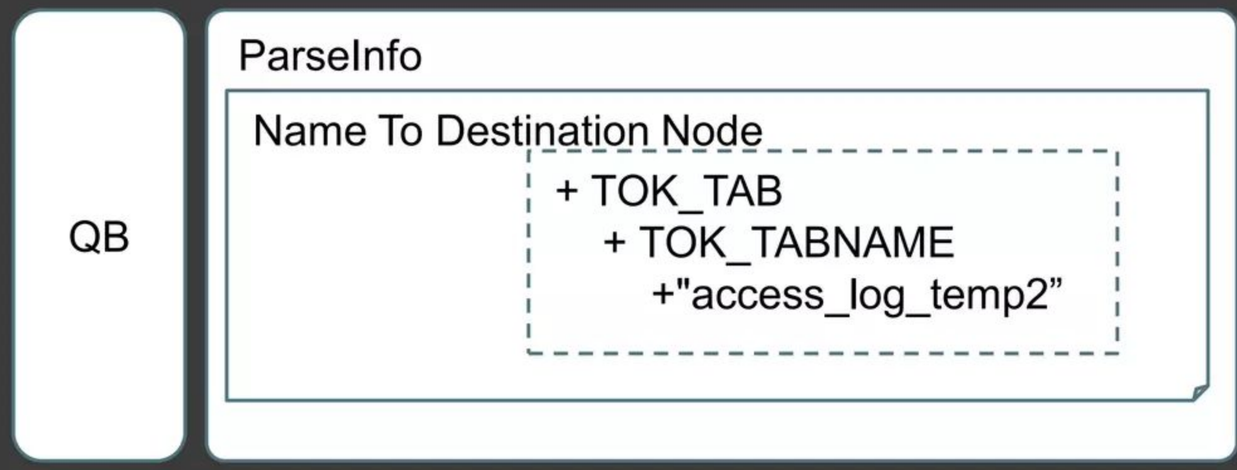
ParseInfo

Join Node
+ TOK_JOIN
  + TOK_TABREF
    …
  + TOK_TABREF
    …
  + “=”
    …

Parser | Semantic Analyzer | Logical Plan Gen. | Logical Optimizer | Physical Plan Gen. | Physical Optimizer

Semantic Analyzer (2/2)





Semantic Analyzer (2/2)

AST

```
+ TOK_SELECT
   + TOK_SELEXPR
      + "."
         + TOK_TABLE_OR_COL
            + "a"
         + "user"
   + TOK_SELEXPR
      + "."
         + TOK_TABLE_OR_COL
            + "a"
         + "prono"
+ TOK_SELEXPR
   + "."
      + TOK_TABLE_OR_COL
         + "p"
      + "maker"
+ TOK_SELEXPR
   + "."
      + TOK_TABLE_OR_COL
         + "p"
      + "price"
```

3

QB

ParseInfo

Name To Select Node

```
+ TOK_SELECT
   + TOK_SELEXPR
      …
   + TOK_SELEXPR
      …
   + TOK_SELEXPR
      …
   + TOK_SELEXPR
      …
```

Parser | Semantic Analyzer | Logical Plan Gen. | Logical Optimizer | Physical Plan Gen. | Physical Optimizer

Logical Plan Generator (1/4)

QB → OP Tree

QB

MetaData

Alias To Table Info
“a”=Table Info(“access_log_hbase”)
“p”=Table Info(“product_hbase”)

OP Tree

TableScanOperator(“access_log_hbase”)
TableScanOperator(“product_hbase”)

Parser | Semantic Analyzer | Logical Plan Gen. | Logical Optimizer | Physical Plan Gen. | Physical Optimizer

Logical Plan Generator (2/4)

QB → OP Tree

QB

ParseInfo

```
+ TOK_JOIN
    + TOK_TABREF
        + TOK_TABNAME
            + "access_log_hbase"
        + a
    + TOK_TABREF
        + TOK_TABNAME
            + "product_hbase"
        + "p"
    + "="
        + "."
            + TOK_TABLE_OR_COL
                + "a"
            + "access_log_hbase"
        + "."
            + TOK_TABLE_OR_COL
                + "p"
            + "prono“
```

OP Tree

ReduceSinkOperator(“access_log_hbase”)
ReduceSinkOperator(“product_hbase”)

JoinOperator

Parser | Semantic Analyzer | Logical Plan Gen. | Logical Optimizer | Physical Plan Gen. | Physical Optimizer

Logical Plan Generator (3/4)

QB → OP Tree

QB

ParseInfo

Name To Select Node

```
+ TOK_SELECT
   + TOK_SELEXPR
      + "."
         + TOK_TABLE_OR_COL
            + "a"
         + "user"
   + TOK_SELEXPR
      + "."
         + TOK_TABLE_OR_COL
            + "a"
         + "prono"
   + TOK_SELEXPR
      + "."
         + TOK_TABLE_OR_COL
            + "p"
         + "maker"
   + TOK_SELEXPR
      + "."
         + TOK_TABLE_OR_COL
            + "p"
         + "price"
```

OP Tree → SelectOperator

Parser | Semantic Analyzer | Logical Plan Gen. | Logical Optimizer | Physical Plan Gen. | Physical Optimizer

Logical Plan Generator (4/4)

QB

MetaData

Name To Destination Table Info
“insclause-0”=
Table Info(“access_log_temp2”)

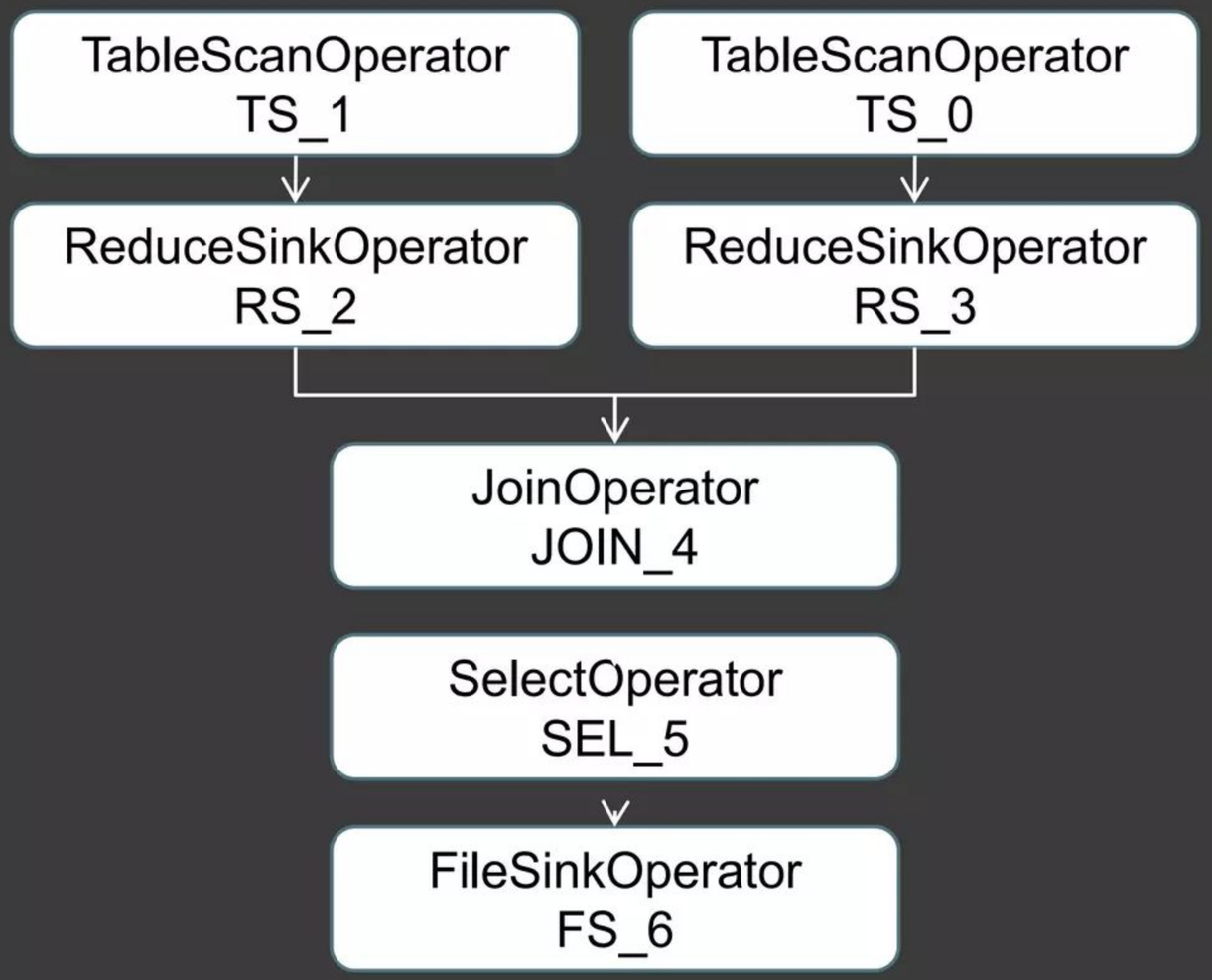
OP Tree

FileSinkOperator

Parser | Semantic Analyzer | Logical Plan Gen. | Logical Optimizer | Physical Plan Gen. | Physical Optimizer

Logical Plan Generator (result)

OP Tree



Parser | Semantic Analyzer | Logical Plan Gen. | Logical Optimizer | Physical Plan Gen. | Physical Optimizer

Logical Optimizer

	概要
LineageGenerator	各Operatorが操作するカラムの情報を設定する。
ColumnPruner	出力に応じて、各Operatorが入出力するカラムを絞り込む。
Predicate PushDown	条件に応じて、Filterの順番を最適化する。
PartitionPruner	条件に応じて、入力対象となるパーティションを絞り込む。
PartitionCondition Remover	PartitionPrunerで対象のパーティションを絞り込む際に、元となったOperationを削除する。

	概要
GroupByOptimizer	Group byでの最適化を行う？
SamplePruner	Samplingでの最適化を行う？
MapJoinProcessor	MAPJOINが指定されていた場合に、JoinOperatorをMapJoinOperatorにコンバートする。
BucketMapJoin Optimizer	
SortedMergeBucket MapJoinOptimizer	
UnionProcessor	Identify if both the subqueries of UNION are map-only.
JoinReader	/*+ STREAMTABLE(A) */
ReduceSink DeDuplication	*If two reducer sink operators share the same partition/sort columns, we should merge them.*

Parser → Semantic Analyzer → Logical Plan Gen. → Logical Optimizer → Physical Plan Gen. → Physical Optimizer

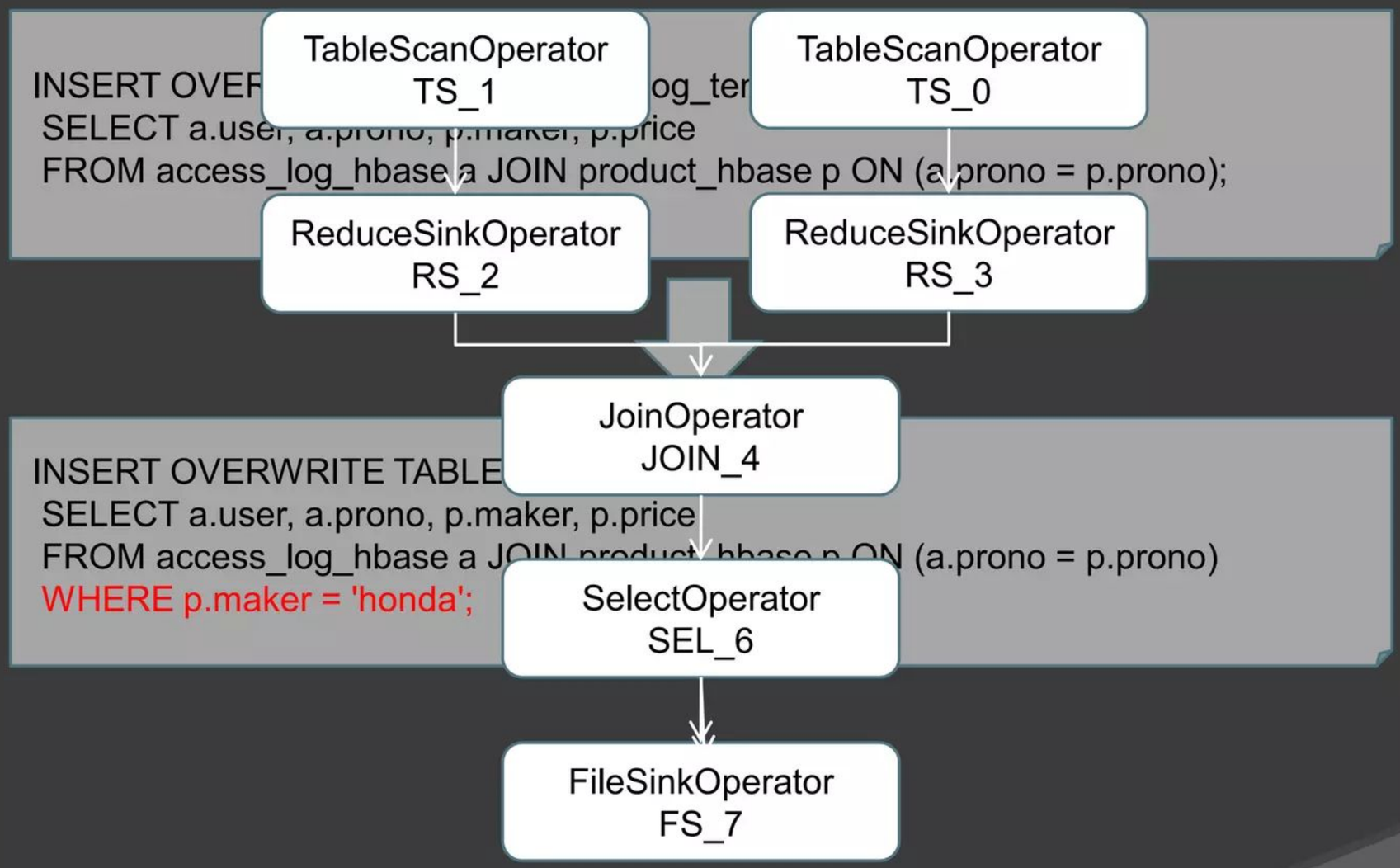
Logical Optimizer (Predicate Push Down)

```
INSERT OVERWRITE TABLE access_log_temp2
 SELECT a.user, a.prono, p.maker, p.price
 FROM access_log_hbase a JOIN product_hbase p ON (a.prono = p.prono);
```

```
INSERT OVERWRITE TABLE access_log_temp2
 SELECT a.user, a.prono, p.maker, p.price
 FROM access_log_hbase a JOIN product_hbase p ON (a.prono = p.prono)
 WHERE p.maker = 'honda';
```

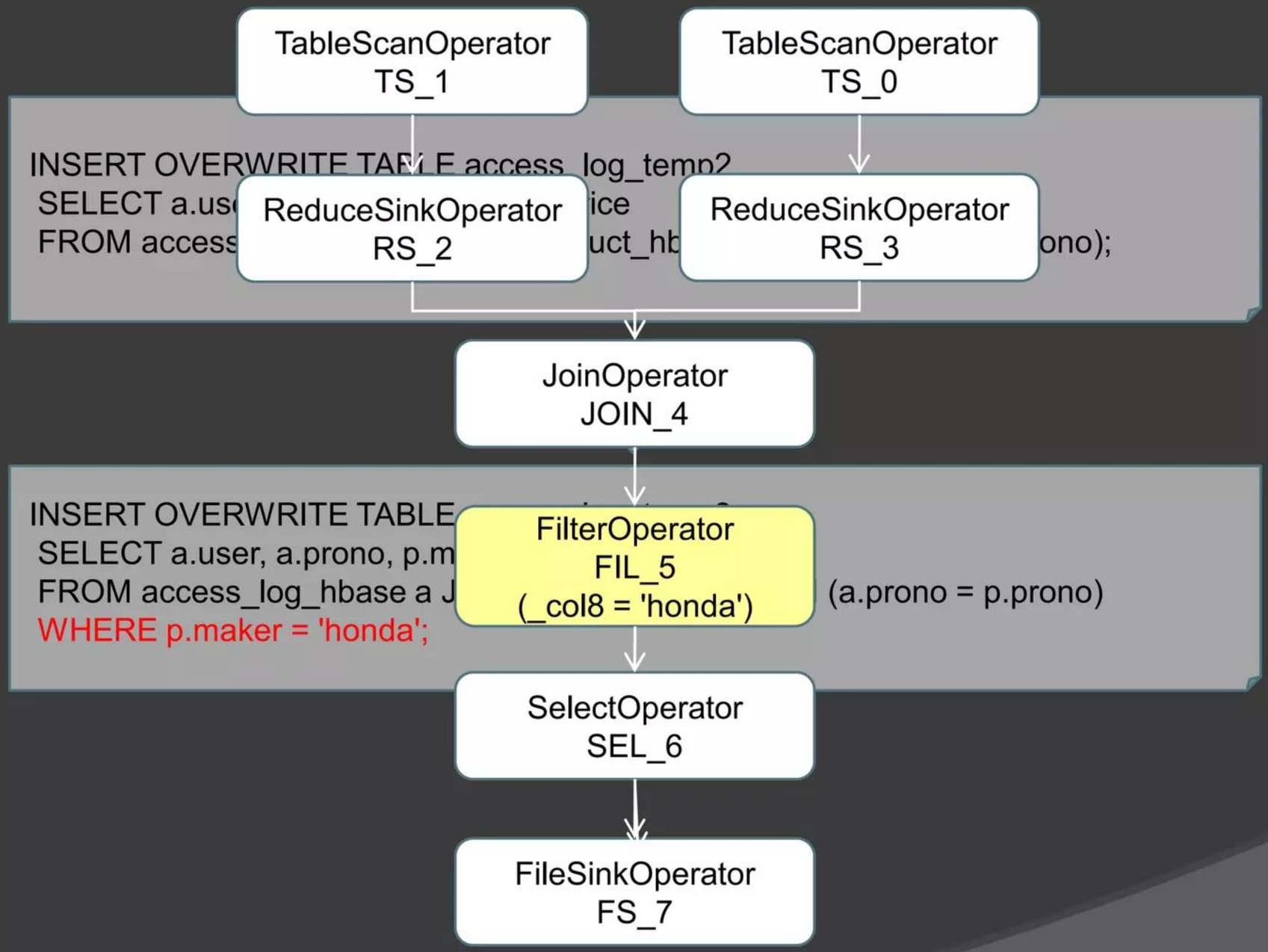
Parser | Semantic Analyzer | Logical Plan Gen. | Logical Optimizer | Physical Plan Gen. | Physical Optimizer

Logical Optimizer (Predicate Push Down)



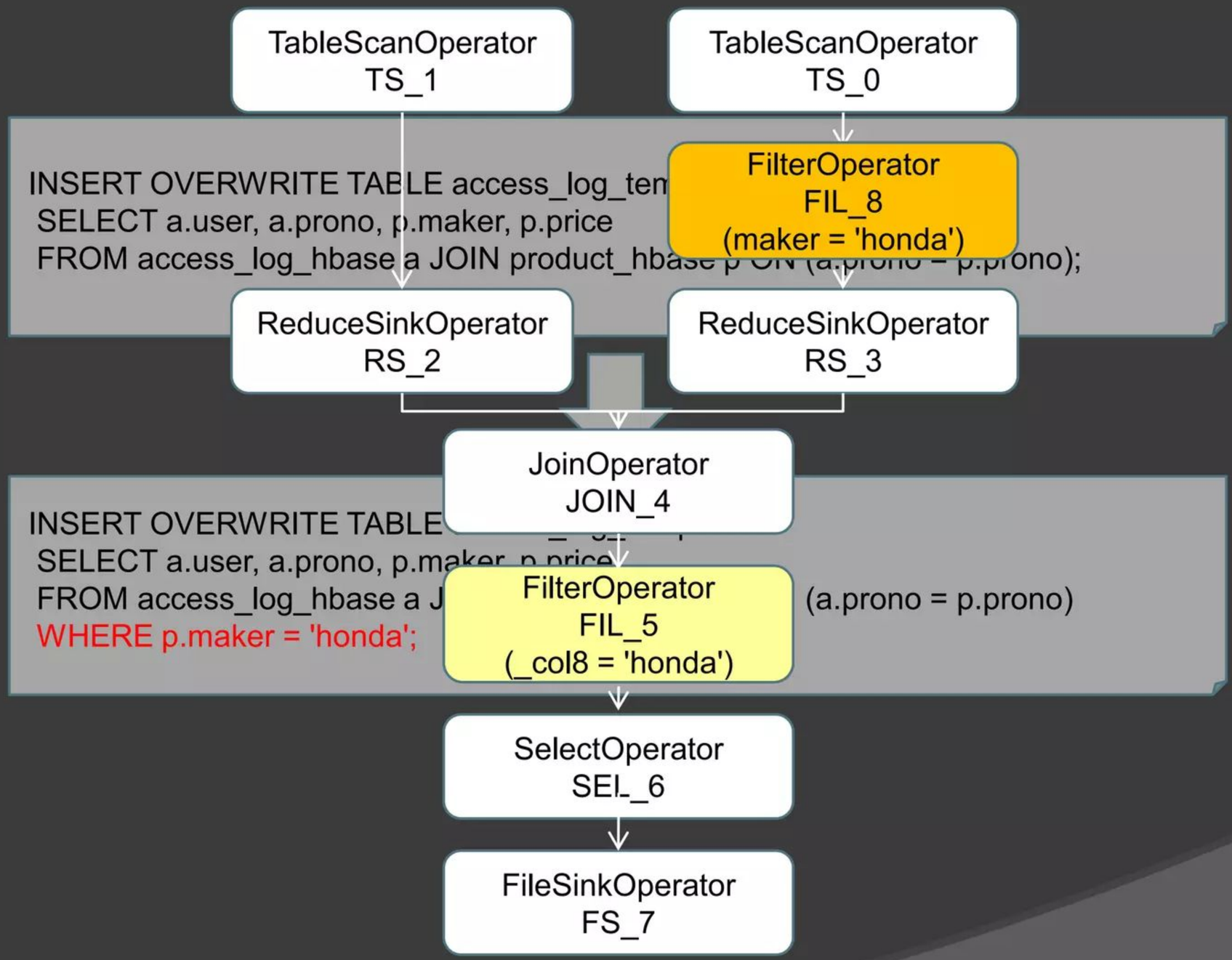
Parser → Semantic Analyzer → Logical Plan Gen. → Logical Optimizer → Physical Plan Gen. → Physical Optimizer

Logical Optimizer (Predicate Push Down)



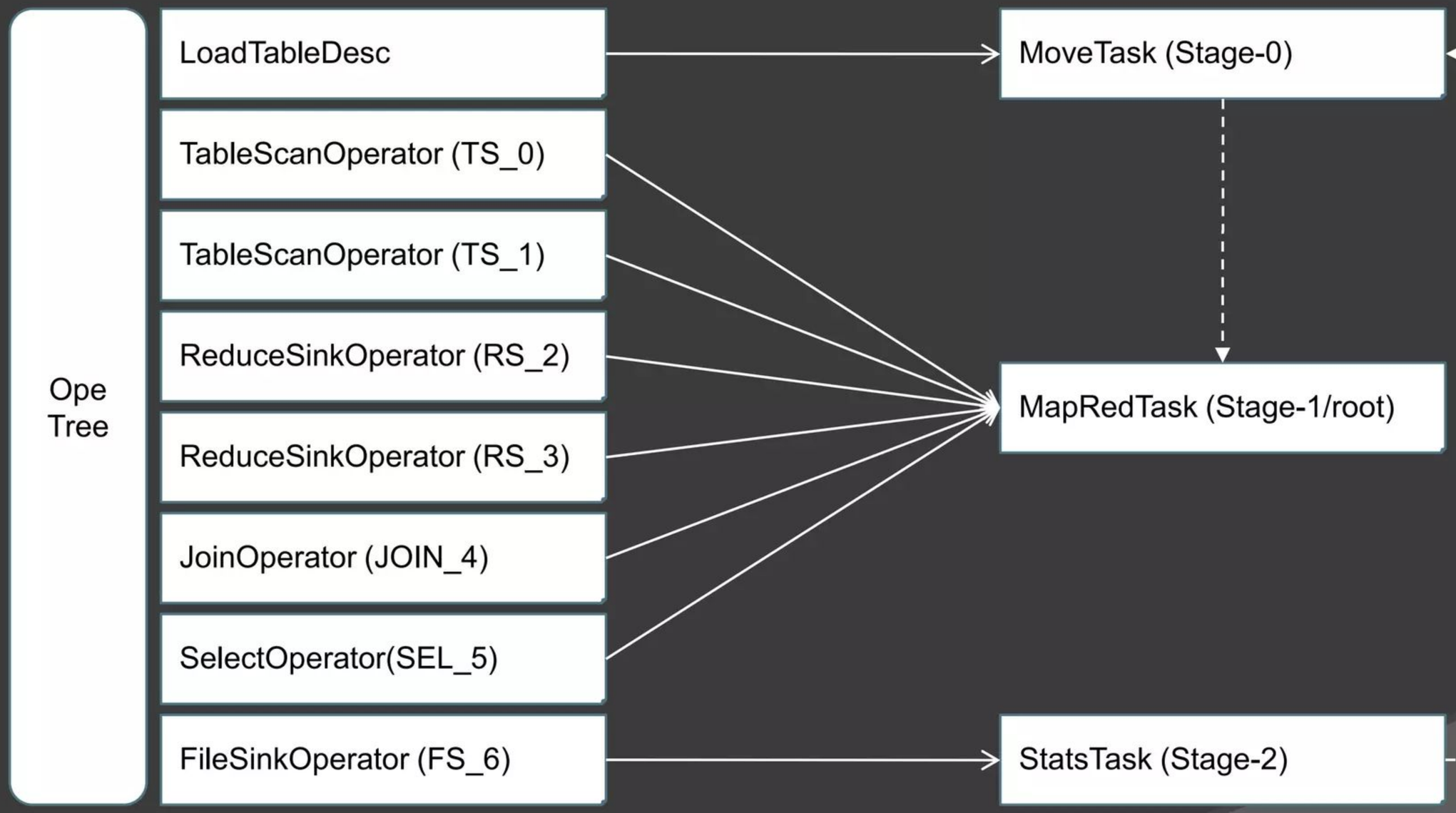
Parser → Semantic Analyzer → Logical Plan Gen. → Logical Optimizer → Physical Plan Gen. → Physical Optimizer

Logical Optimizer (Predicate Push Down)



Parser → Semantic Analyzer → Logical Plan Gen. → Logical Optimizer → Physical Plan Gen. → Physical Optimizer

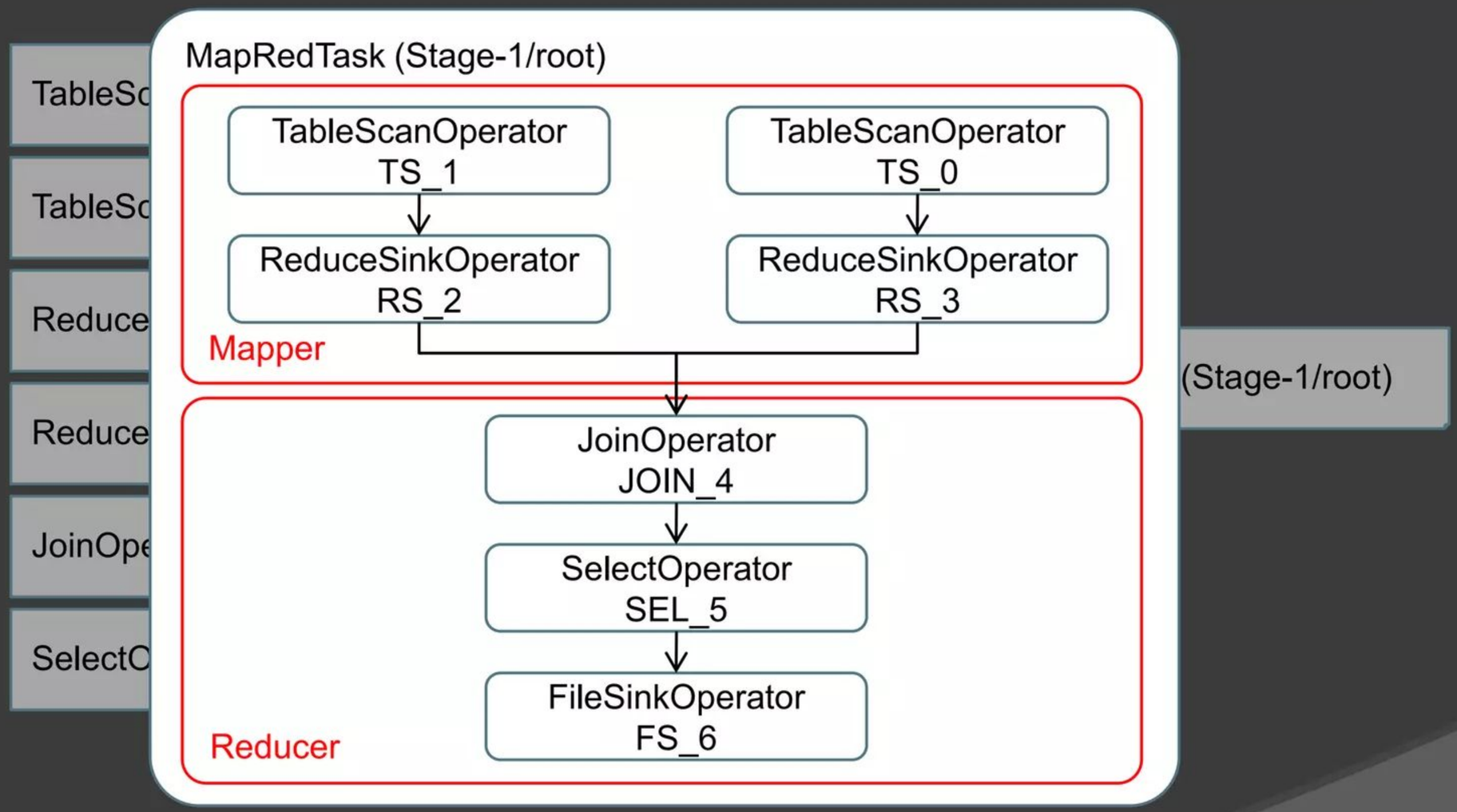
Physical Plan Generator



Parser → Semantic Analyzer → Logical Plan Gen. → Logical Optimizer → Physical Plan Gen. → Physical Optimizer

Physical Plan Generator (result)

OP Tree → Task Tree



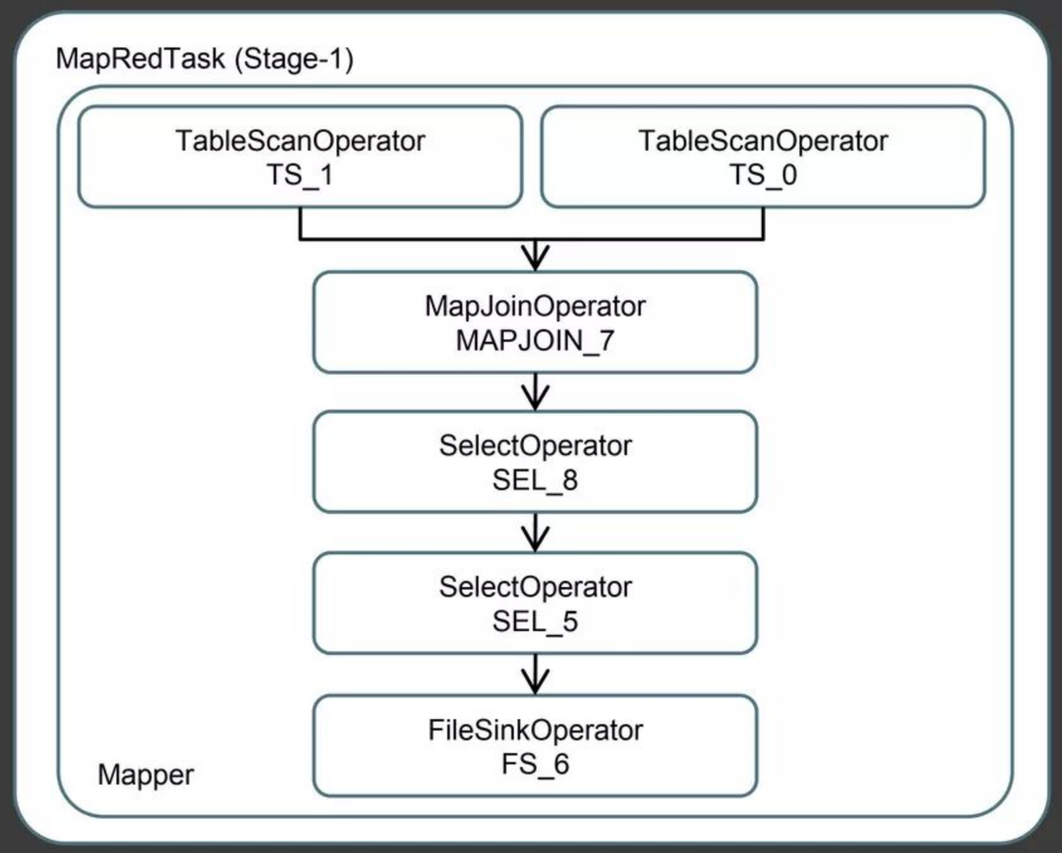
Parser → Semantic Analyzer → Logical Plan Gen. → Logical Optimizer → Physical Plan Gen. → Physical Optimizer

Physical Optimizer

java/org/apache/hadoop/hive/ql/optimizer/physical/以下

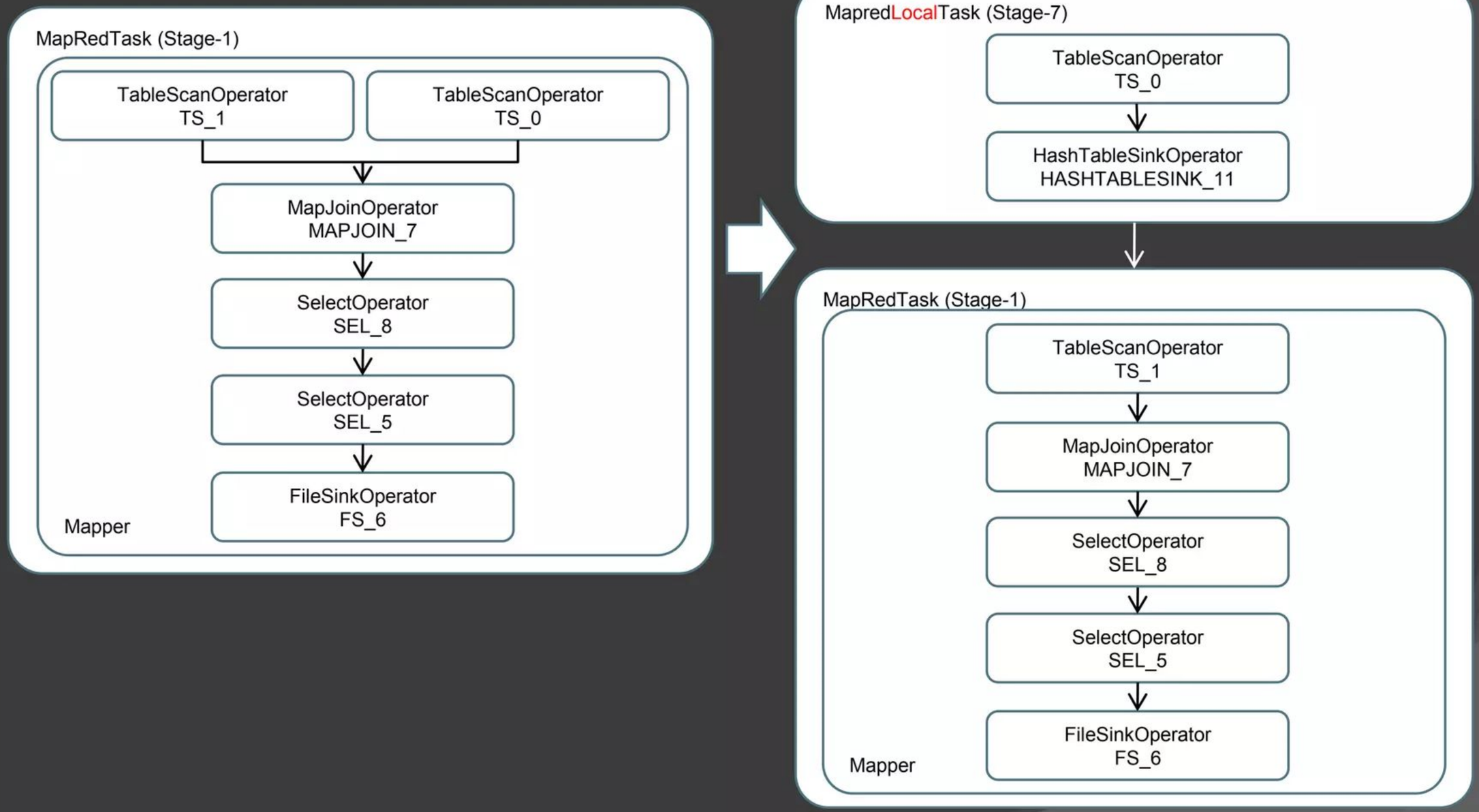
	概要
MapJoinResolver	MapJoinの処理を、ローカルM/Rでのキャッシュ化と実JOINのセット、に変換。
SkewJoinResolver	キーが偏っているテーブルでのJOIN処理を高速化する
CommonJoinResolver	MapJoinへの変換を行う？

Physical Optimizer (MapJoinResolver)



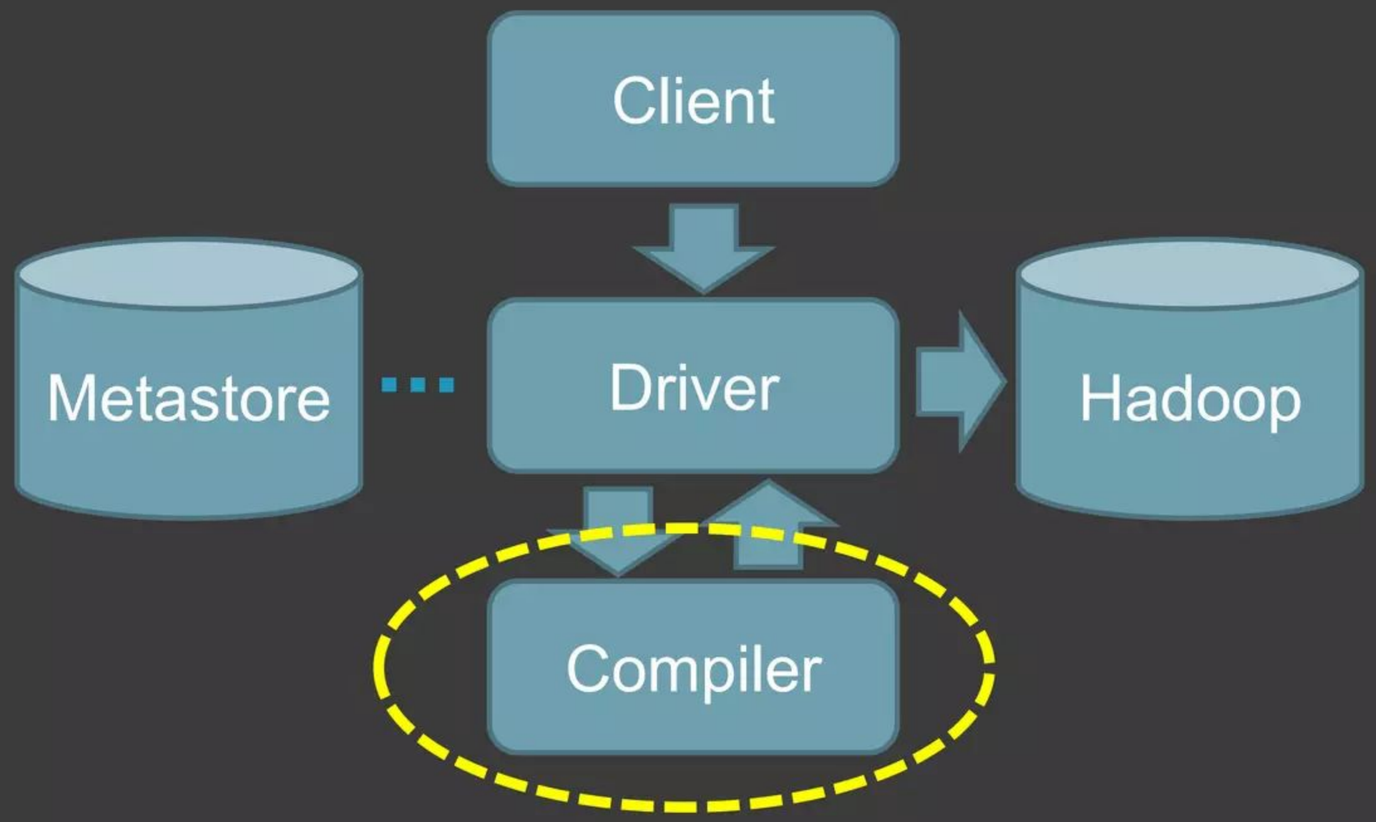
Physical Optimizer (MapJoinResolver)

Task Tree → Task Tree

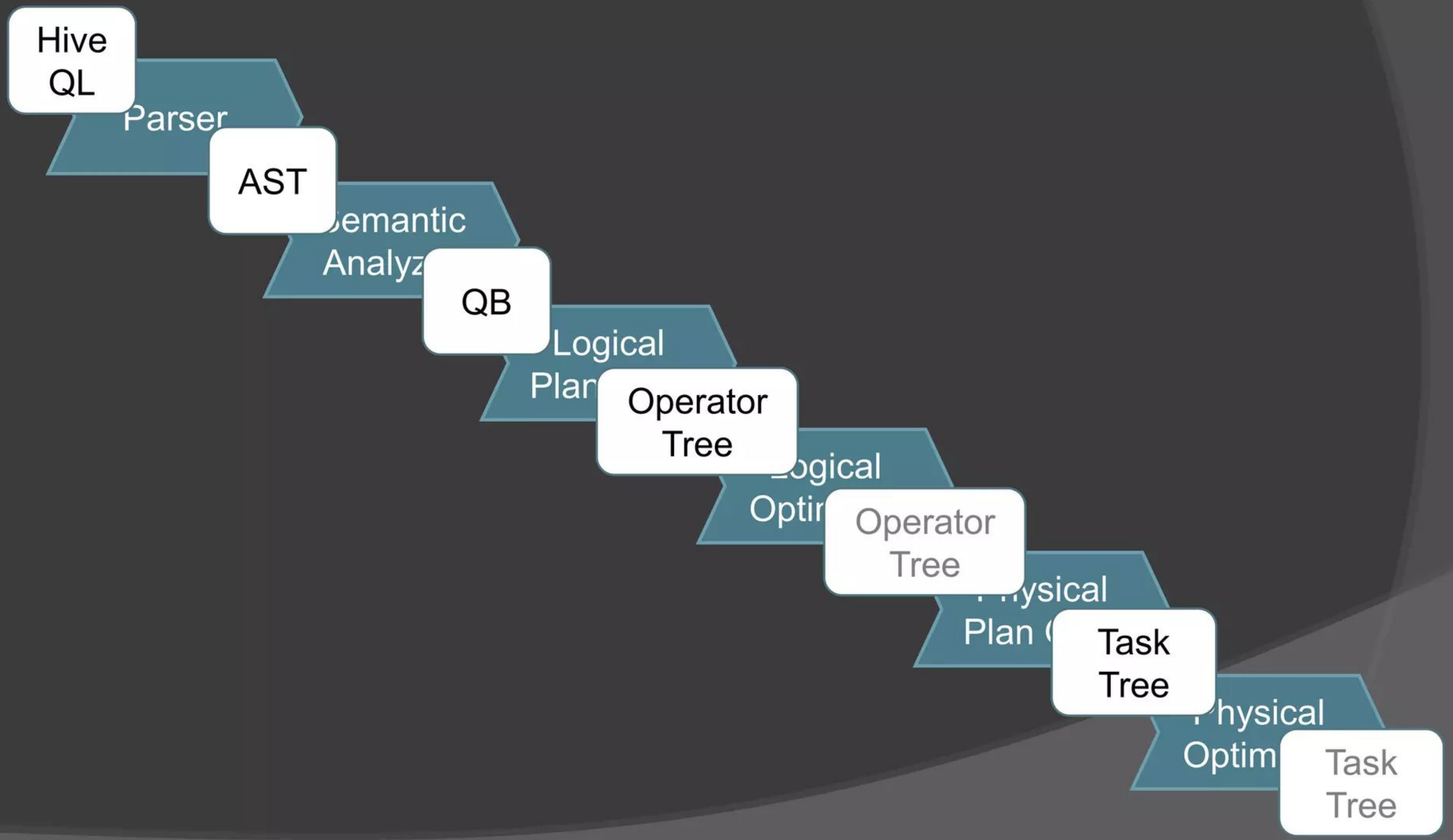


Parser → Semantic Analyzer → Logical Plan Gen. → Logical Optimizer → Physical Plan Gen. → Physical Optimizer

In the end



In the end


Hive QL
Parser
AST
Semantic Analyz
QB
Logical Plan
Operator Tree
Logical Optim
Operator Tree
Physical Plan
Task Tree
Physical Optim
Task Tree

End

- Appendix: What does Explain show?

Appendix: What does Explain show?

```
hive> explain INSERT OVERWRITE TABLE access_log_temp2
    > SELECT a.user, a.prono, p.maker, p.price
    > FROM access_log_hbase a JOIN product_hbase p ON (a.prono = p.prono);
OK
ABSTRACT SYNTAX TREE:
  (TOK_QUERY (TOK_FROM (TOK_JOIN (TOK_TABREF (TOK_TABNAME
access_log_hbase) a) (TOK_TABREF (TOK_TABNAME product_hbase) p) (= (.
(TOK_TABLE_OR_COL a) prono) (. (TOK_TABLE_OR_COL p) prono)))) (TOK_INSERT
(TOK_DESTINATION (TOK_TAB (TOK_TABNAME access_log_temp2))) (TOK_SELECT
(TOK_SELEXPR (. (TOK_TABLE_OR_COL a) user)) (TOK_SELEXPR (.
(TOK_TABLE_OR_COL a) prono)) (TOK_SELEXPR (. (TOK_TABLE_OR_COL p) maker))
(TOK_SELEXPR (. (TOK_TABLE_OR_COL p) price)))))

STAGE DEPENDENCIES:
  Stage-1 is a root stage
  Stage-0 depends on stages: Stage-1
  Stage-2 depends on stages: Stage-0

STAGE PLANS:
  Stage: Stage-1
    Map Reduce
      Alias -> Map Operator Tree:
        a
          TableScan
            alias: a
            Reduce Output Operator
              key expressions:
                    expr: prono
                    type: int
              sort order: +
              Map-reduce partition columns:
                    expr: prono
                    type: int
              tag: 0
              value expressions:
                    expr: user
                    type: string
                    expr: prono
                    type: int
p
          TableScan
            alias: p
            Reduce Output Operator
              key expressions:
                    expr: prono
                    type: int
              sort order: +
              Map-reduce partition columns:
                    expr: prono
                    type: int
              tag: 1
              value expressions:
                    expr: maker
                    type: string
                    expr: price
                    type: int
      Reduce Operator Tree:
            Join Operator
              condition map:
                   Inner Join 0 to 1
              condition expressions:
                0 {VALUE._col0} {VALUE._col2}
                1 {VALUE._col1} {VALUE._col2}
              handleSkewJoin: false
              outputColumnNames: _col0, _col2, _col6, _col7
              Select Operator
                expressions:
                      expr: _col0
                      type: string
                      expr: _col2
                      type: int
                      expr: _col6
                      type: string
                      expr: _col7
                      type: int
                outputColumnNames: _col0, _col1, _col2, _col3
                File Output Operator
                  compressed: false
                  GlobalTableId: 1
                  table:
                      input format: org.apache.hadoop.mapred.TextInputFormat
                      output format: org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat
                      serde: org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
                      name: default.access_log_temp2

  Stage: Stage-0
    Move Operator
      tables:
          replace: true
          table:
              input format: org.apache.hadoop.mapred.TextInputFormat
              output format: org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat
              serde: org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
              name: default.access_log_temp2

  Stage: Stage-2
    Stats-Aggr Operator

Time taken: 0.1 seconds
hive>
```

Appendix: What does Explain show?

```
hive> explain INSERT OVERWRITE TABLE access_log_temp2
    > SELECT a.user, a.prono, p.maker, p.price
    > FROM access_log_hbase a JOIN product_hbase p ON (a.prono = p.prono);
OK
ABSTRACT SYNTAX TREE:
  (TOK_QUERY (TOK_FROM (TOK_JOIN (TOK_TABREF (TOK_TABNAME
access_log_hbase) a) (TOK_TABREF (TOK_TABNAME product_hbase) p) (= (.
(TOK_TABLE_OR_COL a) prono) (. (TOK_TABLE_OR_COL p) prono)))) (TOK_INSERT
(TOK_DESTINATION (TOK_TAB (TOK_TABNAME access_log_temp2))) (TOK_SELECT
(TOK_SELEXPR (. (TOK_TABLE_OR_COL a) user)) (TOK_SELEXPR (.
(TOK_TABLE_OR_COL a) prono)) (TOK_SELEXPR (. (TOK_TABLE_OR_COL p) maker))
(TOK_SELEXPR (. (TOK_TABLE_OR_COL p) price)))))

STAGE DEPENDENCIES:
  Stage-1 is a root stage
  Stage-0 depends on stages: Stage-1
  Stage-2 depends on stages: Stage-0

STAGE PLANS:
  Stage: Stage-1
    Map Reduce
      Alias -> Map Operator Tree:
        a
          TableScan
            alias: a
            Reduce Output Operator
              key expressions:
                    expr: prono
                    type: int
              sort order: +
              Map-reduce partition columns:
                    expr: prono
                    type: int
              tag: 0
              value expressions:
                    expr: user
                    type: string
                    expr: prono
                    type: int
        p
          TableScan
            alias: p
            Reduce Output Operator
              key expressions:
                    expr: prono
                    type: int
              sort order: +
              Map-reduce partition columns:
                    expr: prono
                    type: int
              tag: 1
              value expressions:
                    expr: maker
                    type: string
                    expr: price
                    type: int
      Reduce Operator Tree:
        Join Operator
          condition map:
               Inner Join 0 to 1
          condition expressions:
            0 {VALUE._col0} {VALUE._col2}
            1 {VALUE._col1} {VALUE._col2}
          handleSkewJoin: false
          outputColumnNames: _col0, _col2, _col6, _col7
          Select Operator
            expressions:
                  expr: _col0
                  type: string
                  expr: _col2
                  type: int
                  expr: _col6
                  type: string
                  expr: _col7
                  type: int
            outputColumnNames: _col0, _col1, _col2, _col3
            File Output Operator
              compressed: false
              GlobalTableId: 1
              table:
                  input format: org.apache.hadoop.mapred.TextInputFormat
                  output format: org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat
                  serde: org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
                  name: default.access_log_temp2

  Stage: Stage-0
    Move Operator
      tables:
          replace: true
          table:
              input format: org.apache.hadoop.mapred.TextInputFormat
              output format: org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat
              serde: org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
              name: default.access_log_temp2

  Stage: Stage-2
    Stats-Aggr Operator

Time taken: 0.1 seconds
hive>
```

ABSTRACT SYNTAX TREE:

STAGE DEPENDENCIES:
- Stage-1 is a root stage
- Stage-0 depends on stages: Stage-1
- Stage-2 depends on stages: Stage-0

STAGE PLANS:
- Stage: Stage-1
  - Map Reduce
    - Map Operator Tree:
      - TableScan
        - Reduce Output Operator
      - TableScan
        - Reduce Output Operator
    - Reduce Operator Tree:
      - Join Operator
        - Select Operator
          - File Output Operator
- Stage: Stage-0
  - Move Operator
- Stage: Stage-2
  - Stats-Aggr Operator

Appendix: What does Explain show?

```
ABSTRACT SYNTAX TREE:

STAGE DEPENDENCIES:
  Stage-1 is a root stage
  Stage-0 depends on stages: Stage-1
  Stage-2 depends on stages: Stage-0

STAGE PLANS:
  Stage: Stage-1
    Map Reduce
      Map Operator Tree:
          TableScan
            Reduce Output Operator
          TableScan
            Reduce Output Operator
      Reduce Operator Tree:
        Join Operator
          Select Operator
            File Output Operator

Stage: Stage-0
  Move Operator

Stage: Stage-2
  Stats-Aggr Operator
```

=