CLOUDERA

Accelerating distributed joins in **Apache Hive**: Runtime filtering enhancements

ApacheCon 2020

29th September 2020

About us

Panagiotis (Panos) Garefalakis @pgaref

Software Engineer @Cloudera, Hive runtime team

Working on Apache Hive, ORC, Tez

PhD in Distributed Systems, Imperial College London

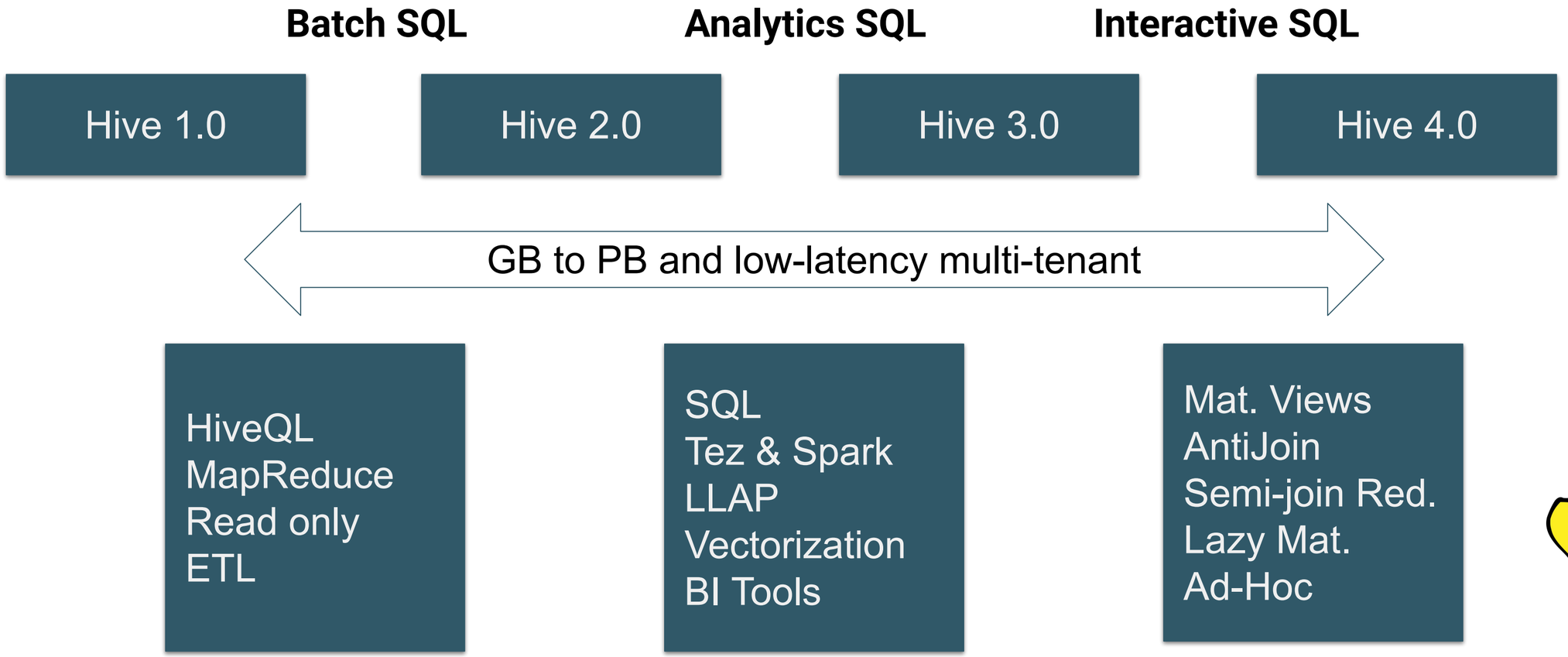
Stamatis Zampetakis

Software Engineer @Cloudera, Hive query optimizer team

Working on Apache Hive, Calcite

PhD in Data Management, INRIA & Paris-Sud University

Apache Hive



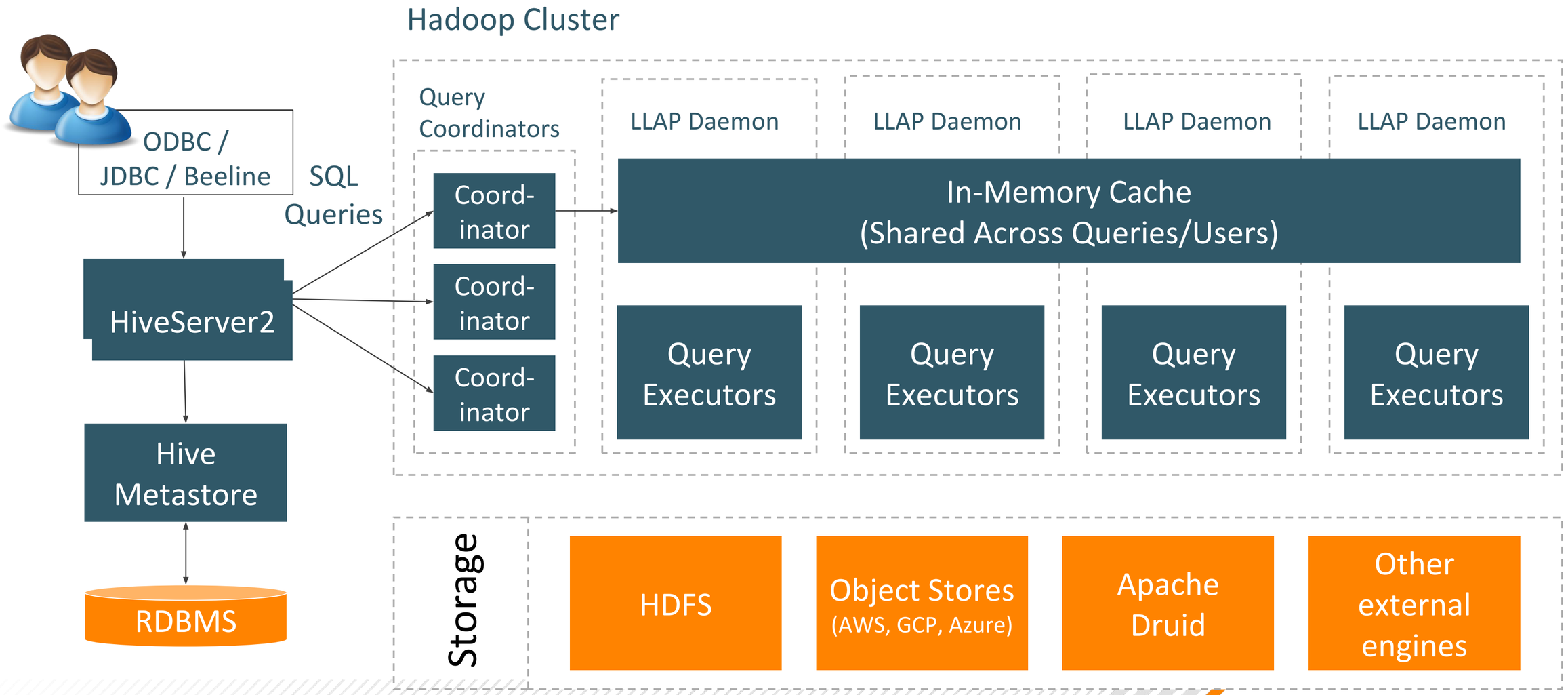
Hive users

7000 analysts, 80ms average latency, 1PB data

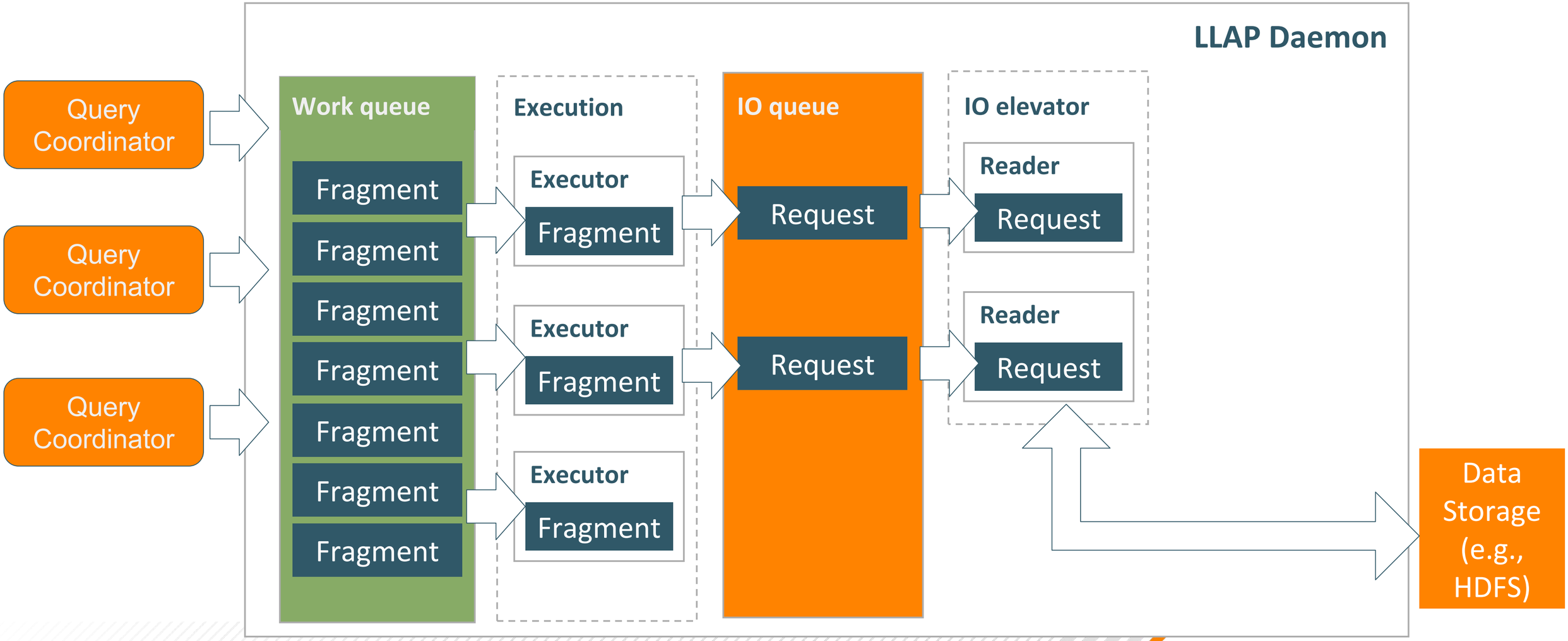
250k BI queries per hour

On demand deep reporting in the cloud over 100Tb in minutes

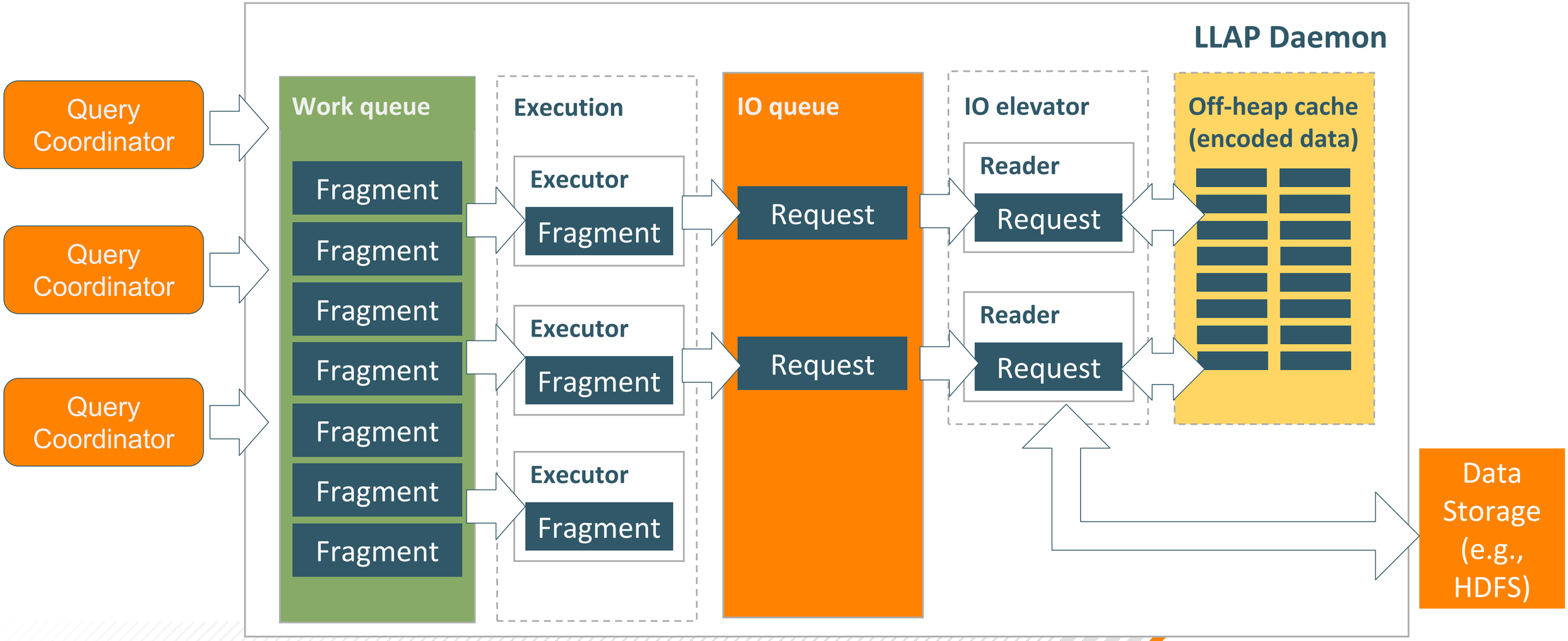
Hive with LLAP



LLAP daemon anatomy



LLAP daemon anatomy



Outline

Data pruning

- Optimizations
- Benefit

Semijoin reduction

- Less data transfer
- Hive semijoin reduction

Lazy materialization

- ORC file format
- Row-level filtering
- Hive Probe-decode

Data pruning

HOW?

Semijoin reduction
Late materialization
... and many more

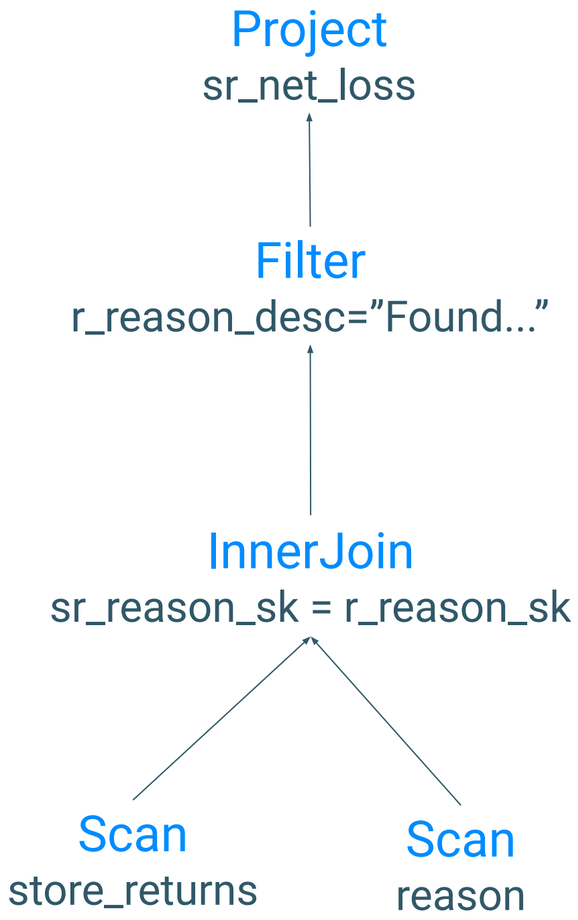
WHY?

Reduce memory footprint
Reduce disk & network I/O
Save CPU cycles

Data pruning

Baseline

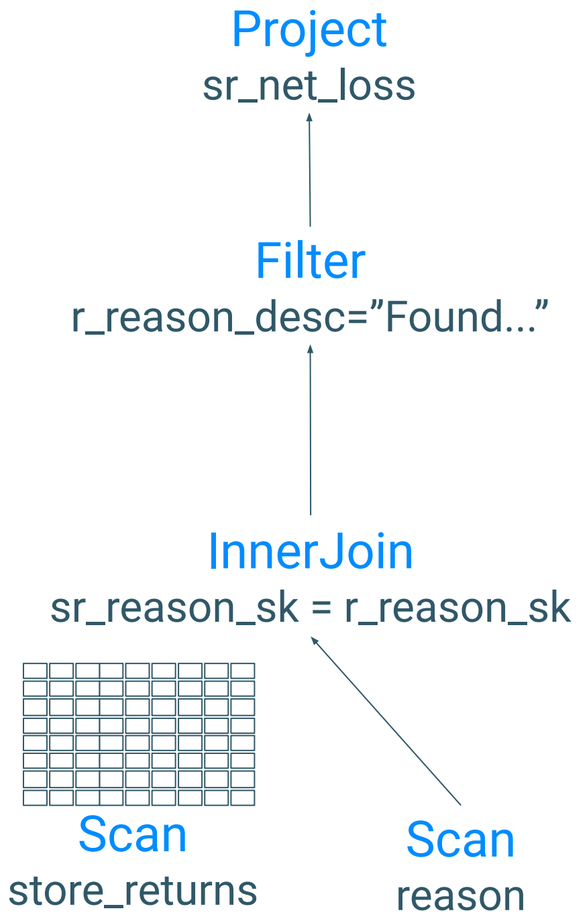
```
SELECT sr_net_loss as net_loss
FROM store_returns
INNER JOIN reason
  ON sr_reason_sk = r_reason_sk
WHERE
 r_reason_desc =
'Found a better price in a store'
```



Data pruning

Baseline

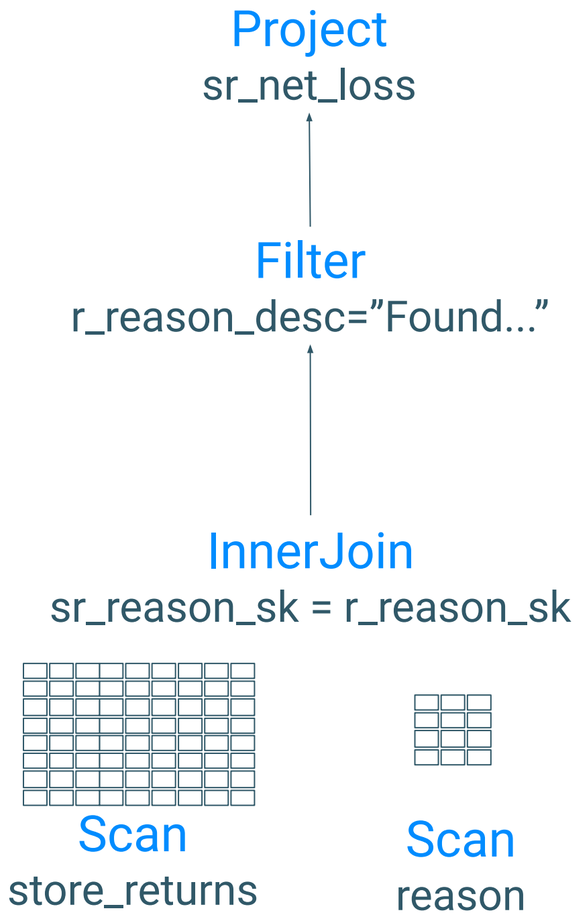
```
SELECT sr_net_loss as net_loss
FROM store_returns
INNER JOIN reason
  ON sr_reason_sk = r_reason_sk
WHERE
 r_reason_desc =
'Found a better price in a store'
```



Data pruning

Baseline

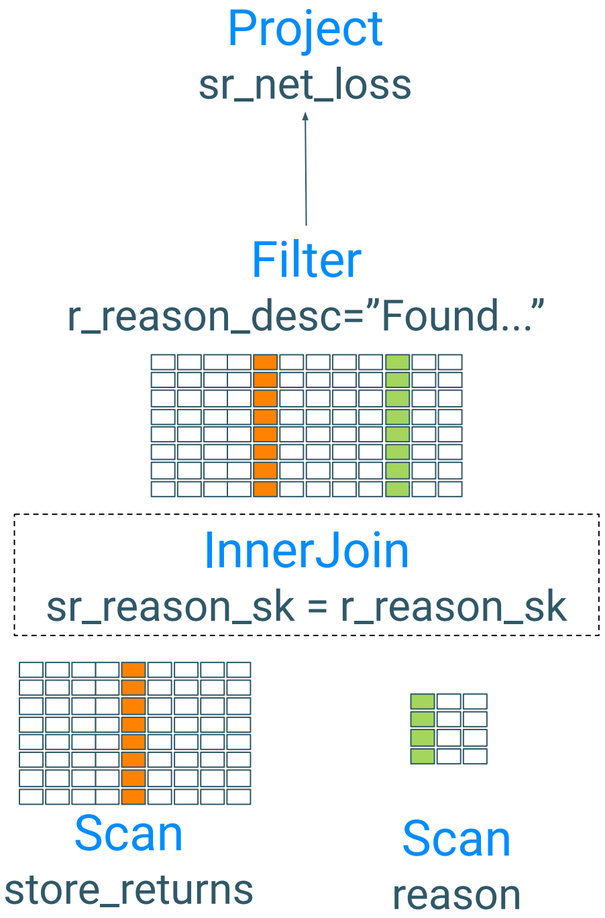
```
SELECT sr_net_loss as net_loss
FROM store_returns
INNER JOIN reason
  ON sr_reason_sk = r_reason_sk
WHERE
 r_reason_desc =
'Found a better price in a store'
```



Data pruning

Baseline

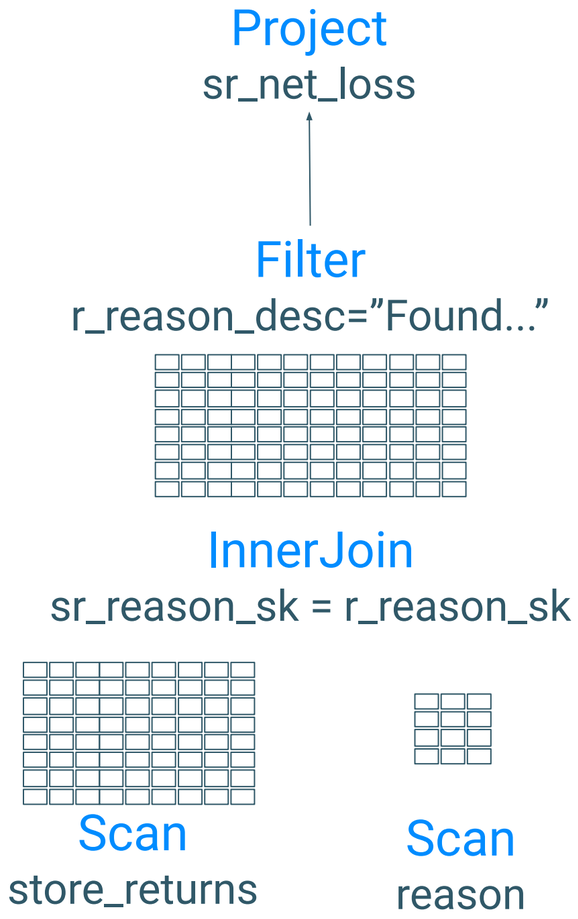
```
SELECT sr_net_loss as net_loss
FROM store_returns
INNER JOIN reason
  ON sr_reason_sk = r_reason_sk
WHERE
 r_reason_desc =
'Found a better price in a store'
```



Data pruning

Baseline

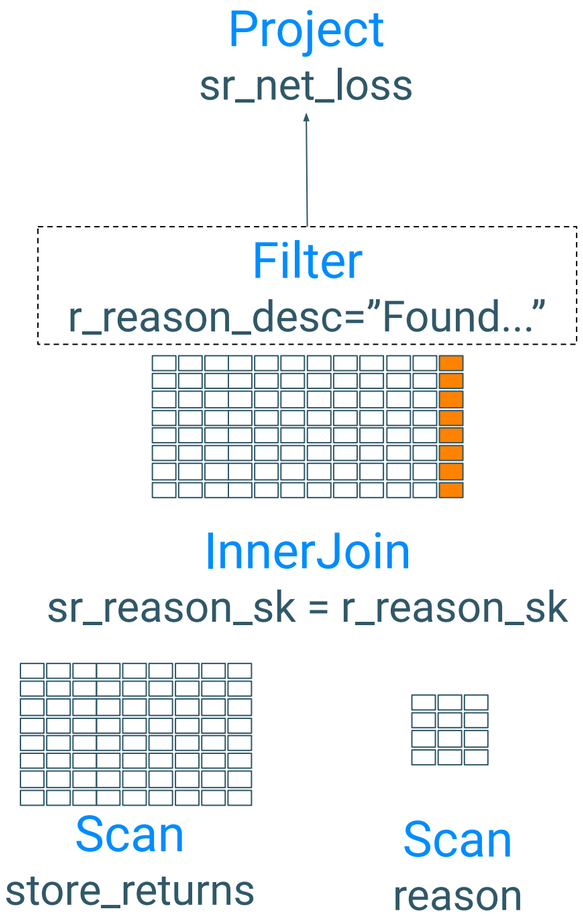
```
SELECT sr_net_loss as net_loss
FROM store_returns
INNER JOIN reason
  ON sr_reason_sk = r_reason_sk
WHERE
 r_reason_desc =
'Found a better price in a store'
```



Data pruning

Baseline

```
SELECT sr_net_loss as net_loss
FROM store_returns
INNER JOIN reason
  ON sr_reason_sk = r_reason_sk
WHERE
 r_reason_desc =
'Found a better price in a store'
```



Data pruning

Baseline

```
SELECT sr_net_loss as net_loss
FROM store_returns
INNER JOIN reason
  ON sr_reason_sk = r_reason_sk
WHERE
 r_reason_desc =
'Found a better price in a store'
```

Data pruning

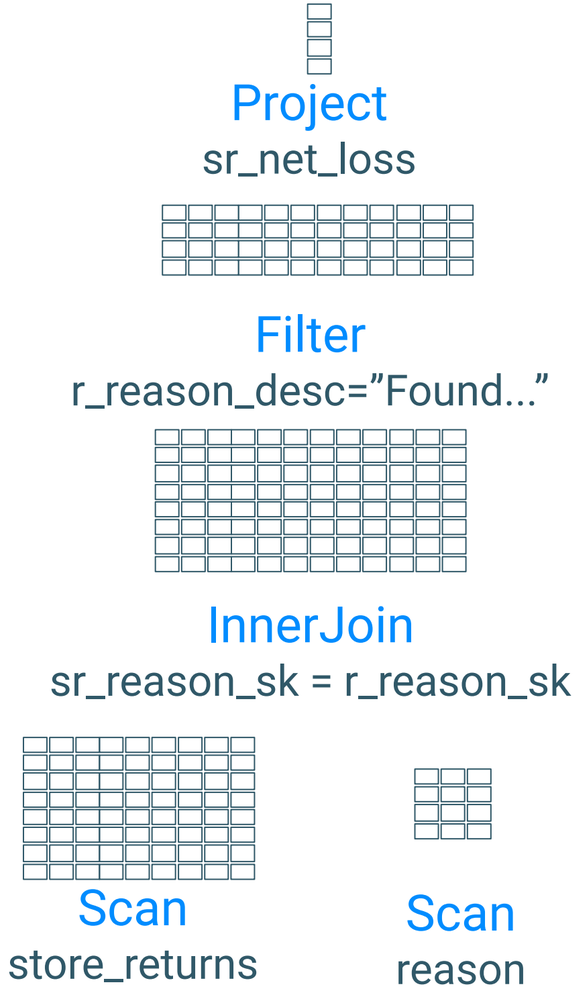
Baseline

```
SELECT sr_net_loss as net_loss
FROM store_returns
INNER JOIN reason
  ON sr_reason_sk = r_reason_sk
WHERE
 r_reason_desc =
'Found a better price in a store'
```

Data pruning

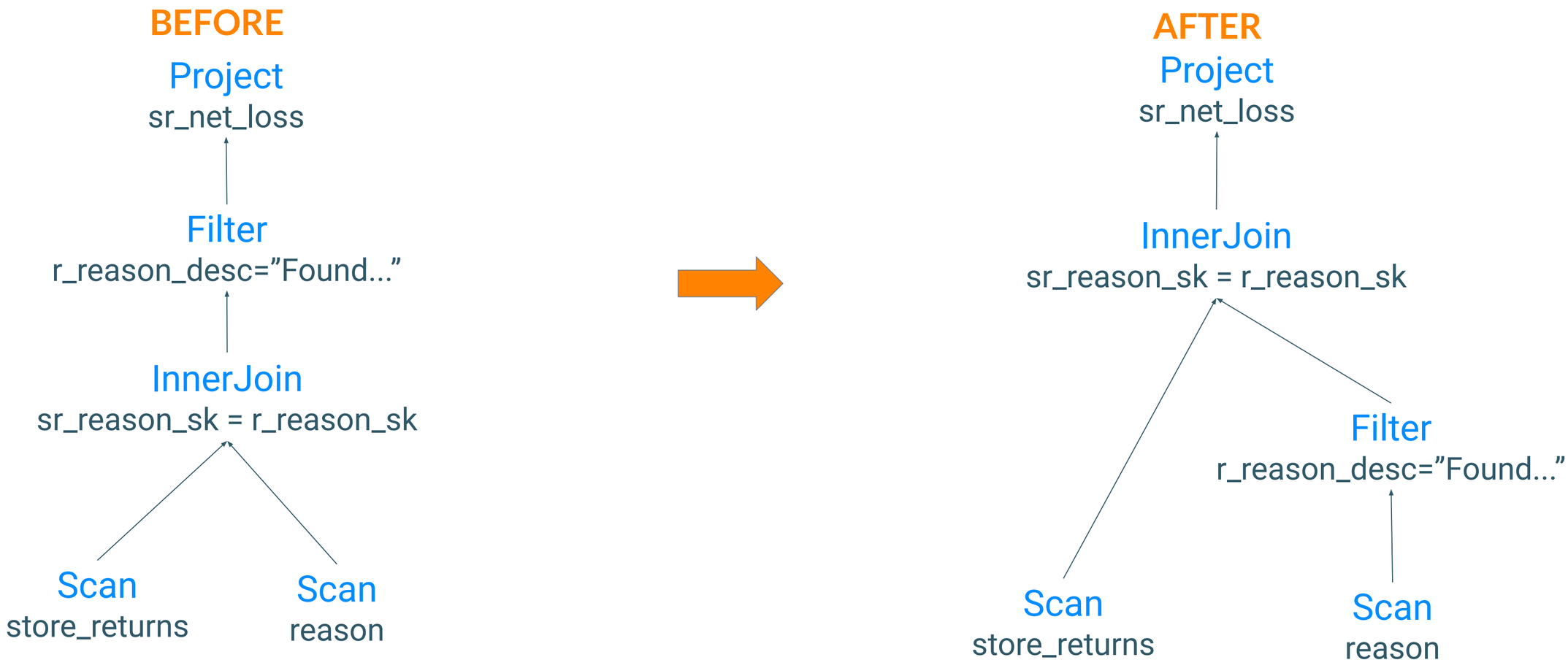
Baseline

```
SELECT sr_net_loss as net_loss
FROM store_returns
INNER JOIN reason
  ON sr_reason_sk = r_reason_sk
WHERE
 r_reason_desc =
'Found a better price in a store'
```



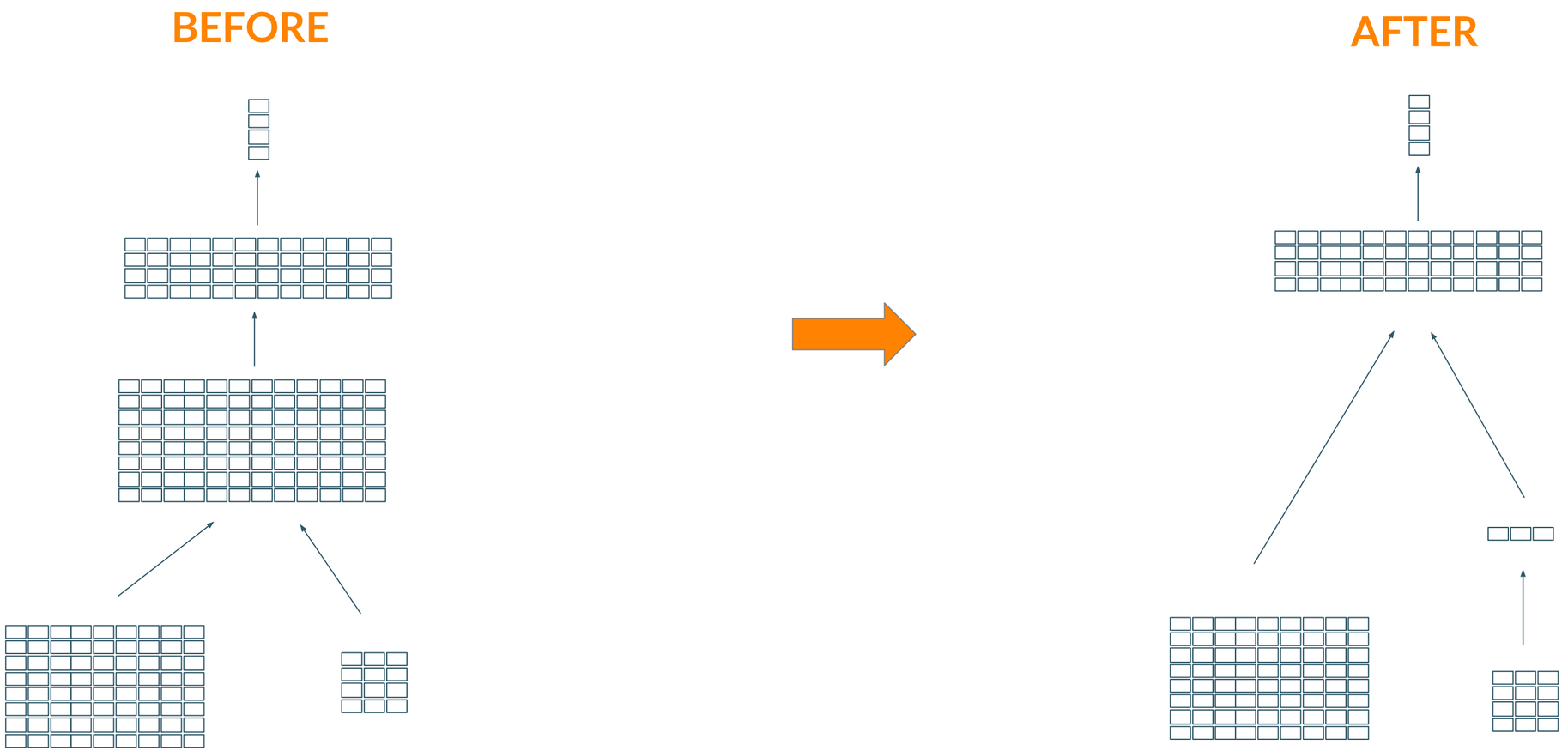
Data pruning

Filter pushdown



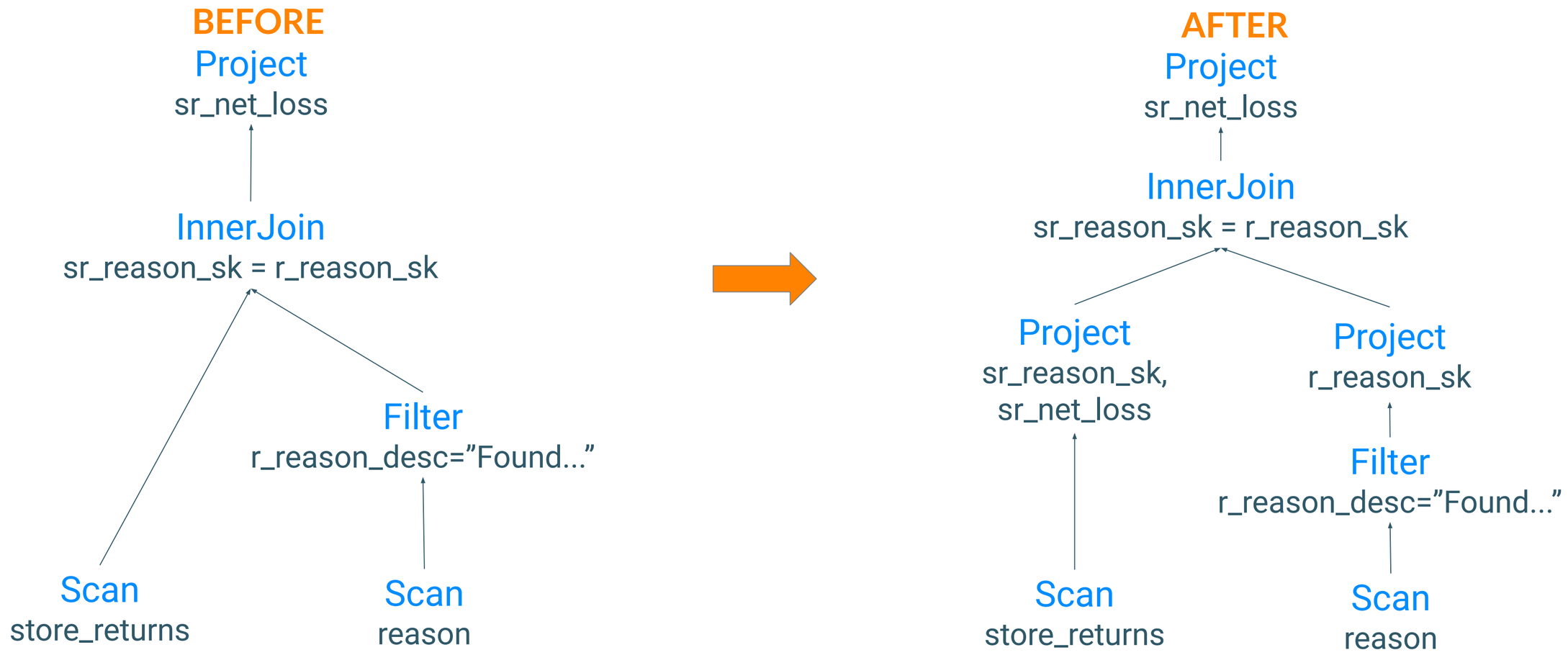
Data pruning

Filter pushdown



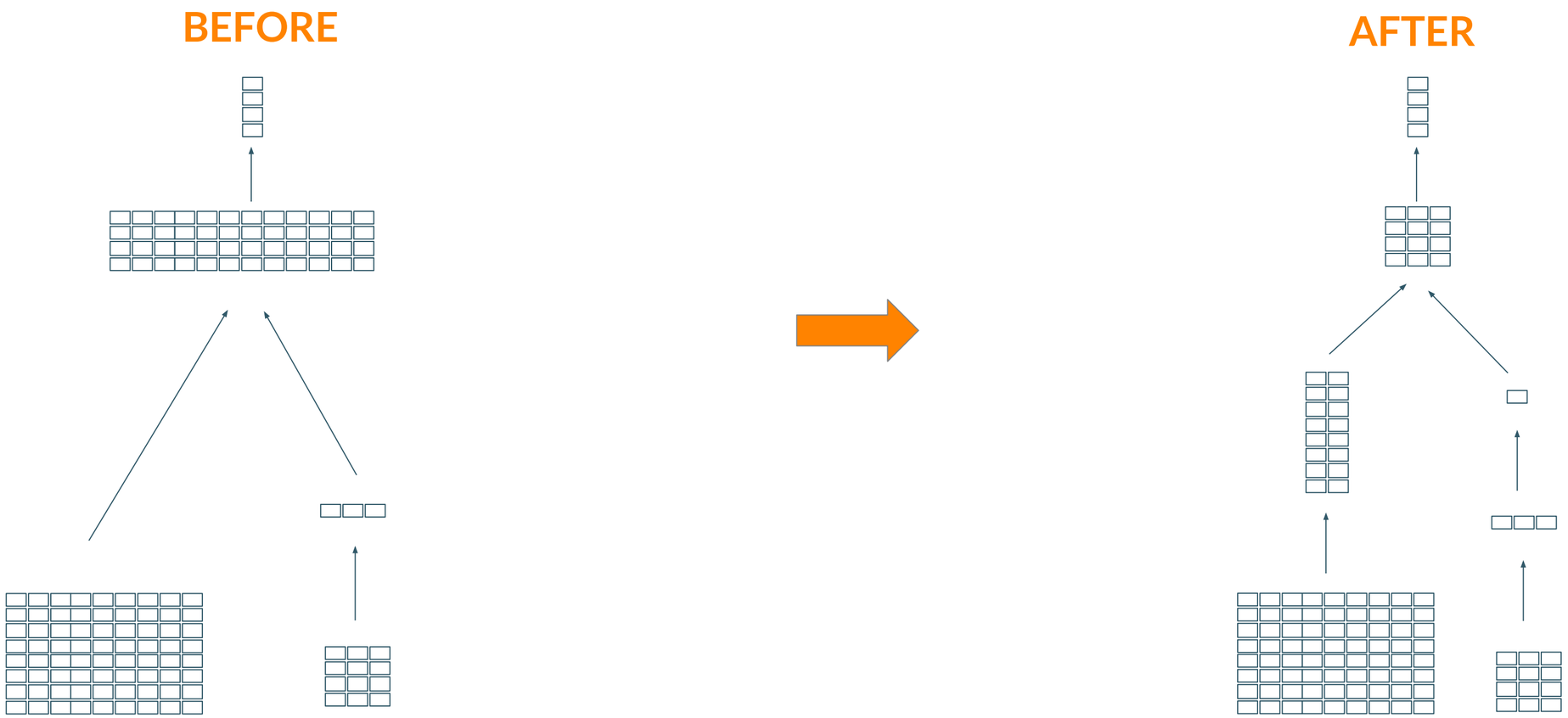
Data pruning

Project pushdown



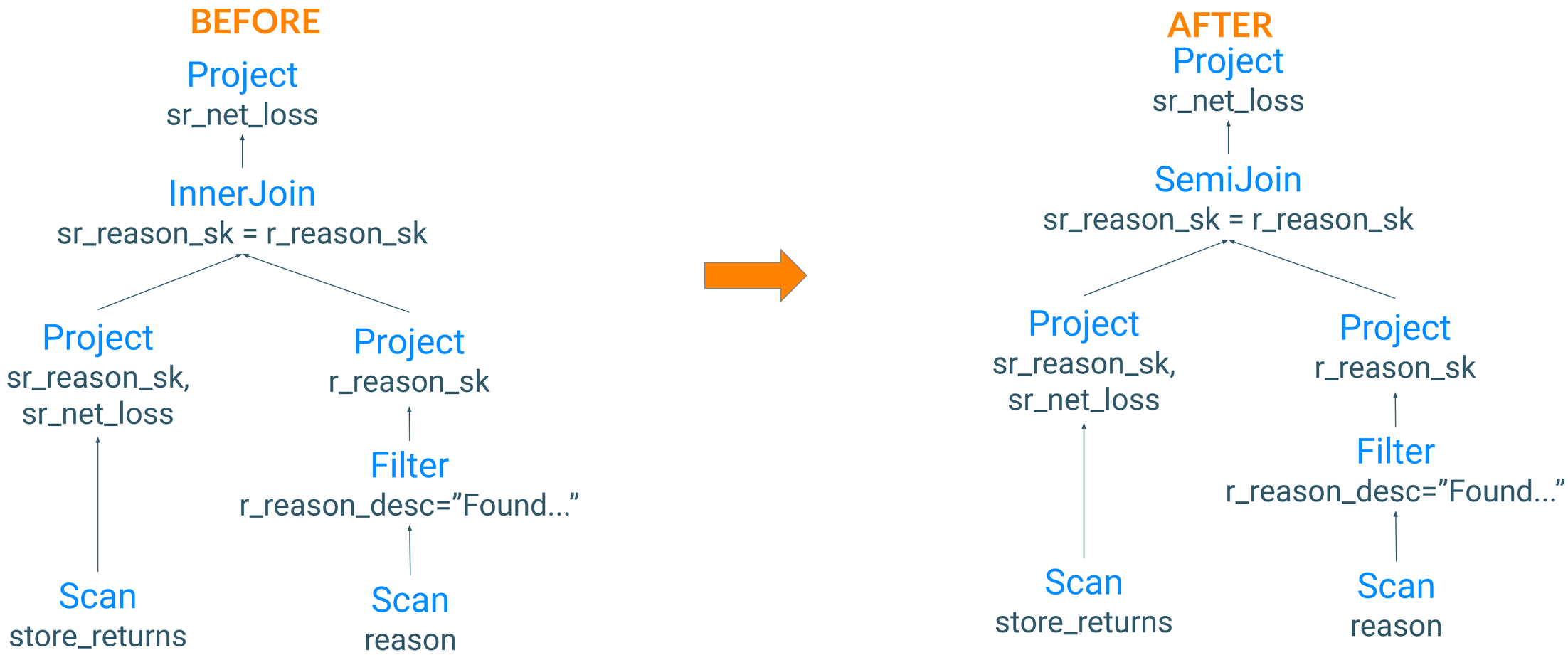
Data pruning

Project pushdown



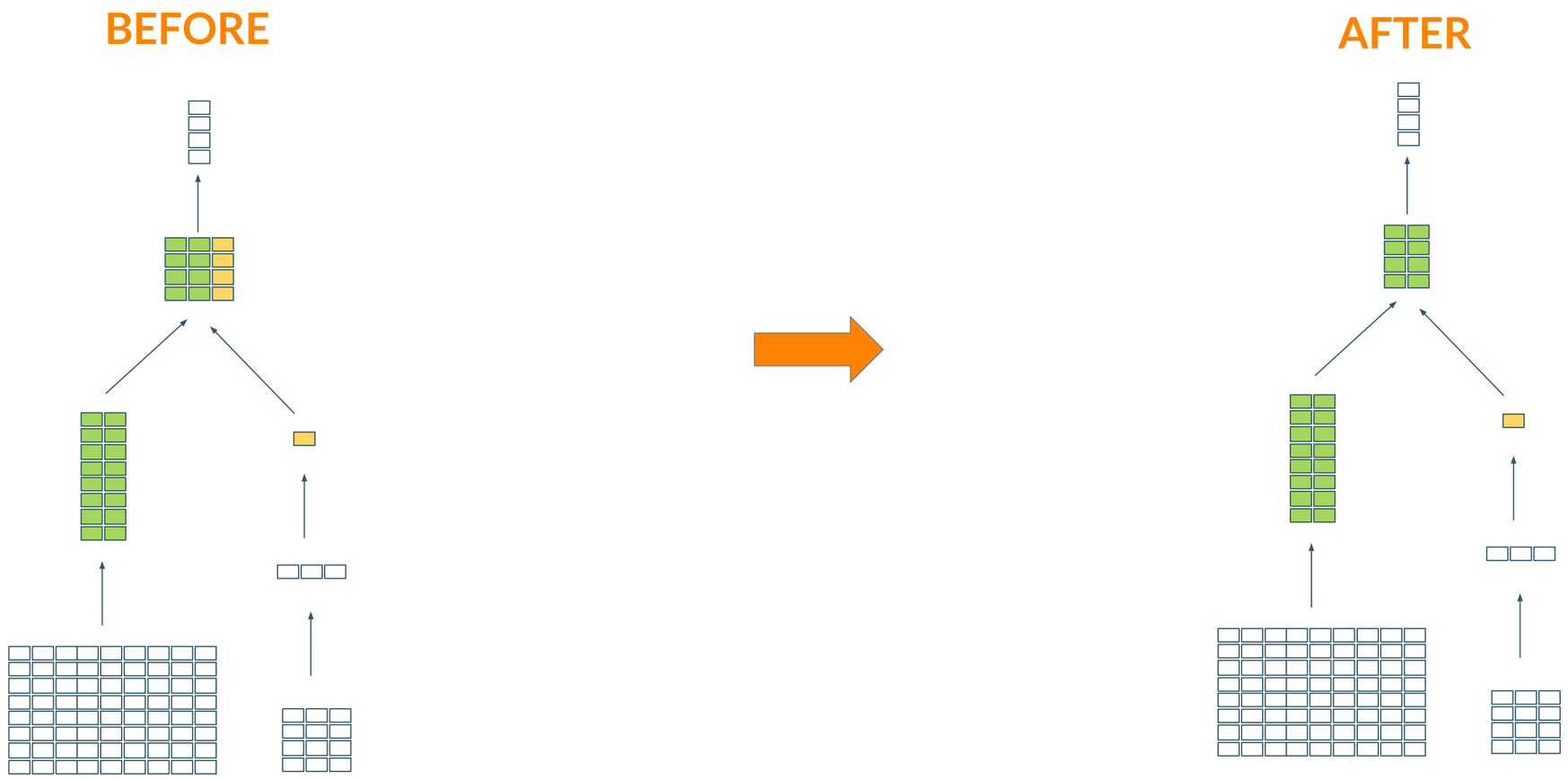
Data pruning

Semijoin reduction ★



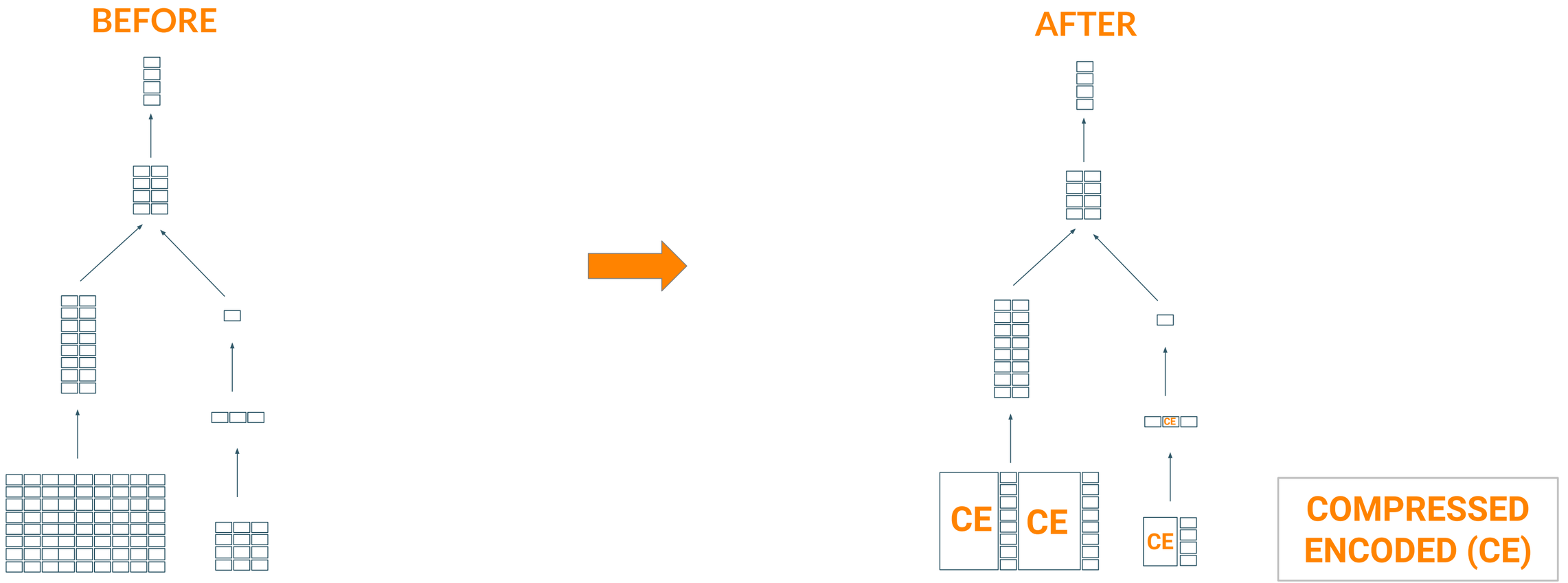
Data pruning

Semijoin reduction ★



Data pruning

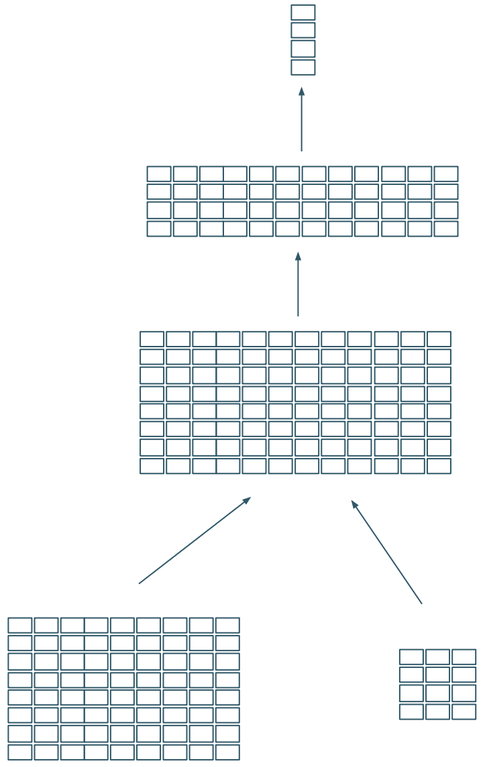
Late materialization ★



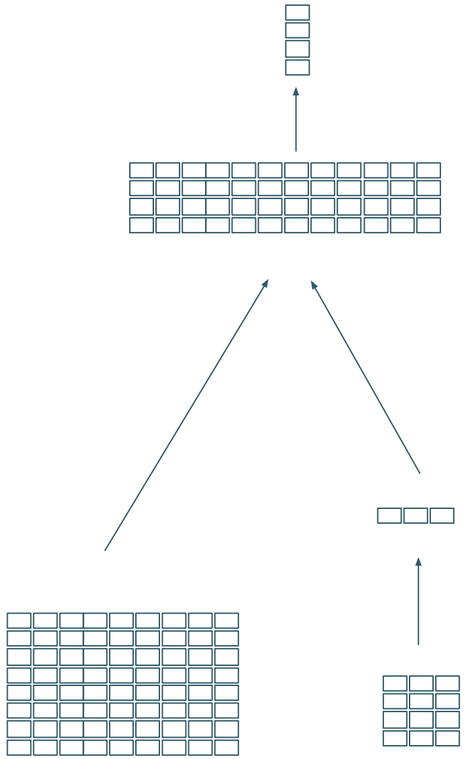
Data pruning

Benefit

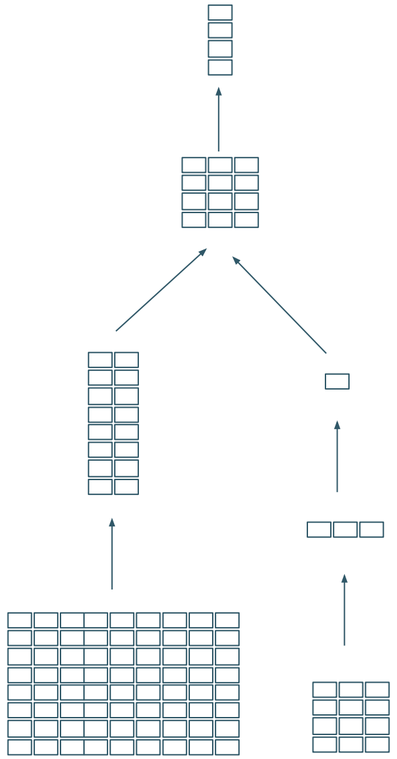
BASELINE



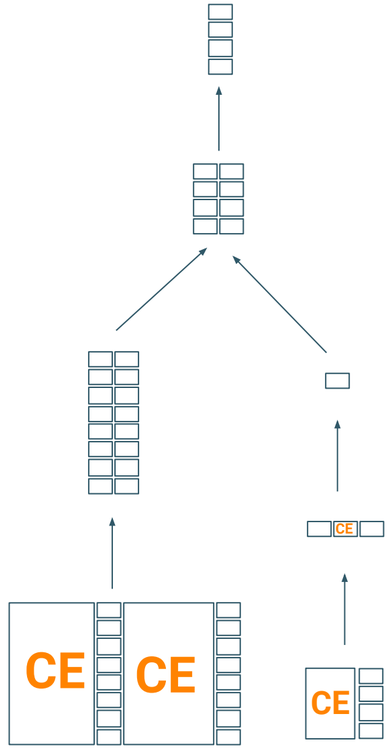
FILTER PUSHDOWN



PROJECT PUSHDOWN



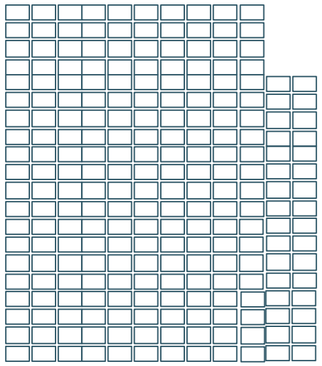
SEMIJOIN REDUCTION & LATE MATERIALIZATION



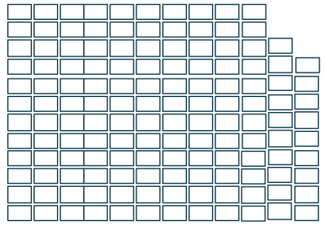
Data pruning

Benefit

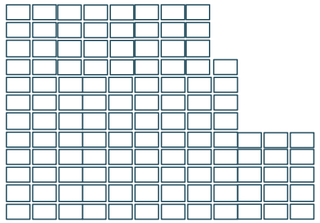
BASELINE



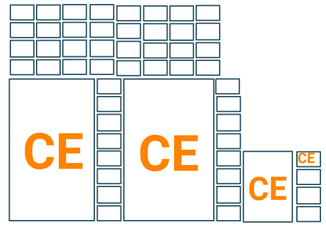
FILTER PUSHDOWN



PROJECT PUSHDOWN



SEMIJOIN REDUCTION & LATE MATERIALIZATION



Semijoin reduction

Common names

Semijoin reduction

Invisible join

Selective join pushdown

Sideways information passing

Benefit

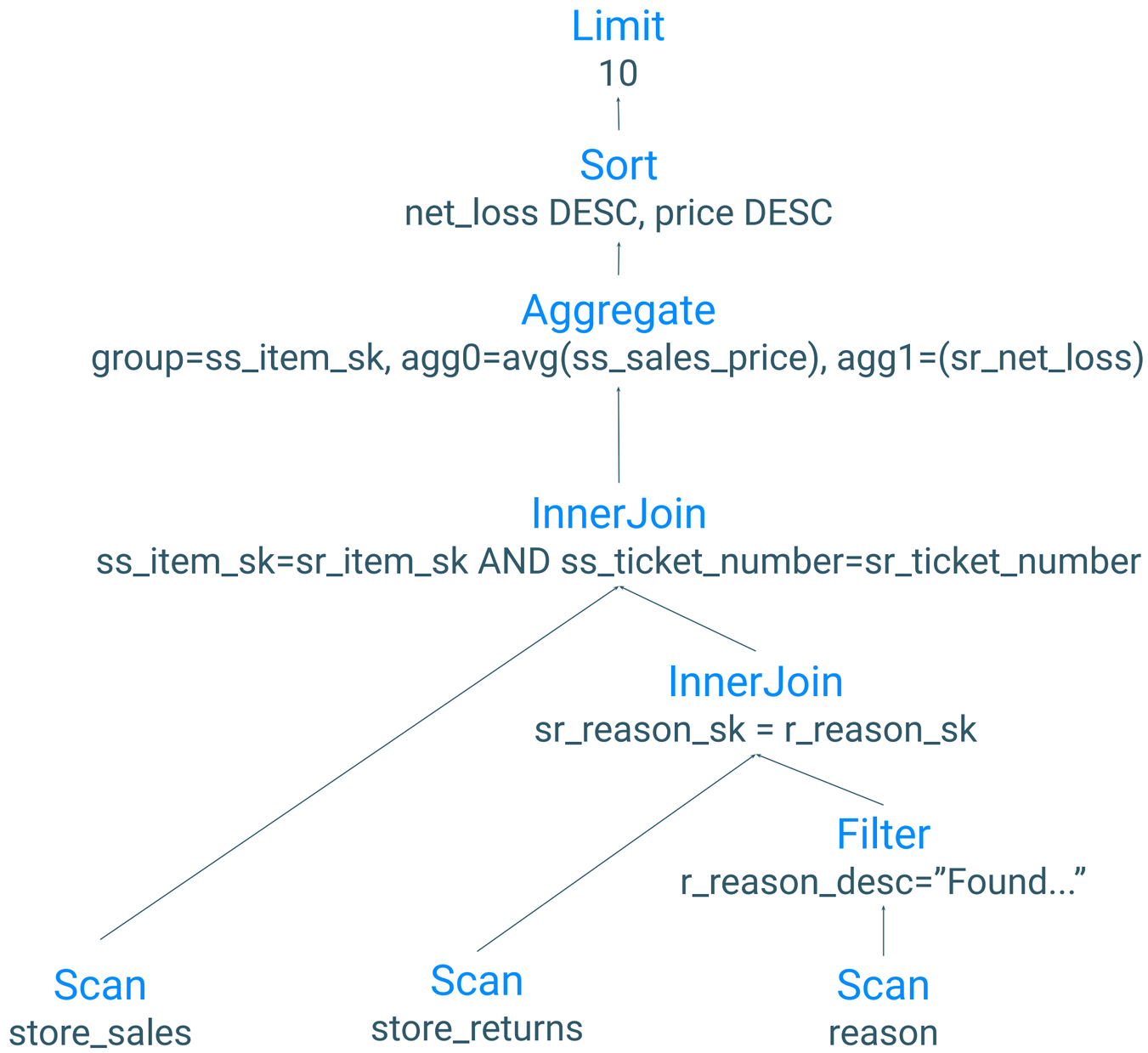
Operators between the semijoin & and the reduced join treat less data:

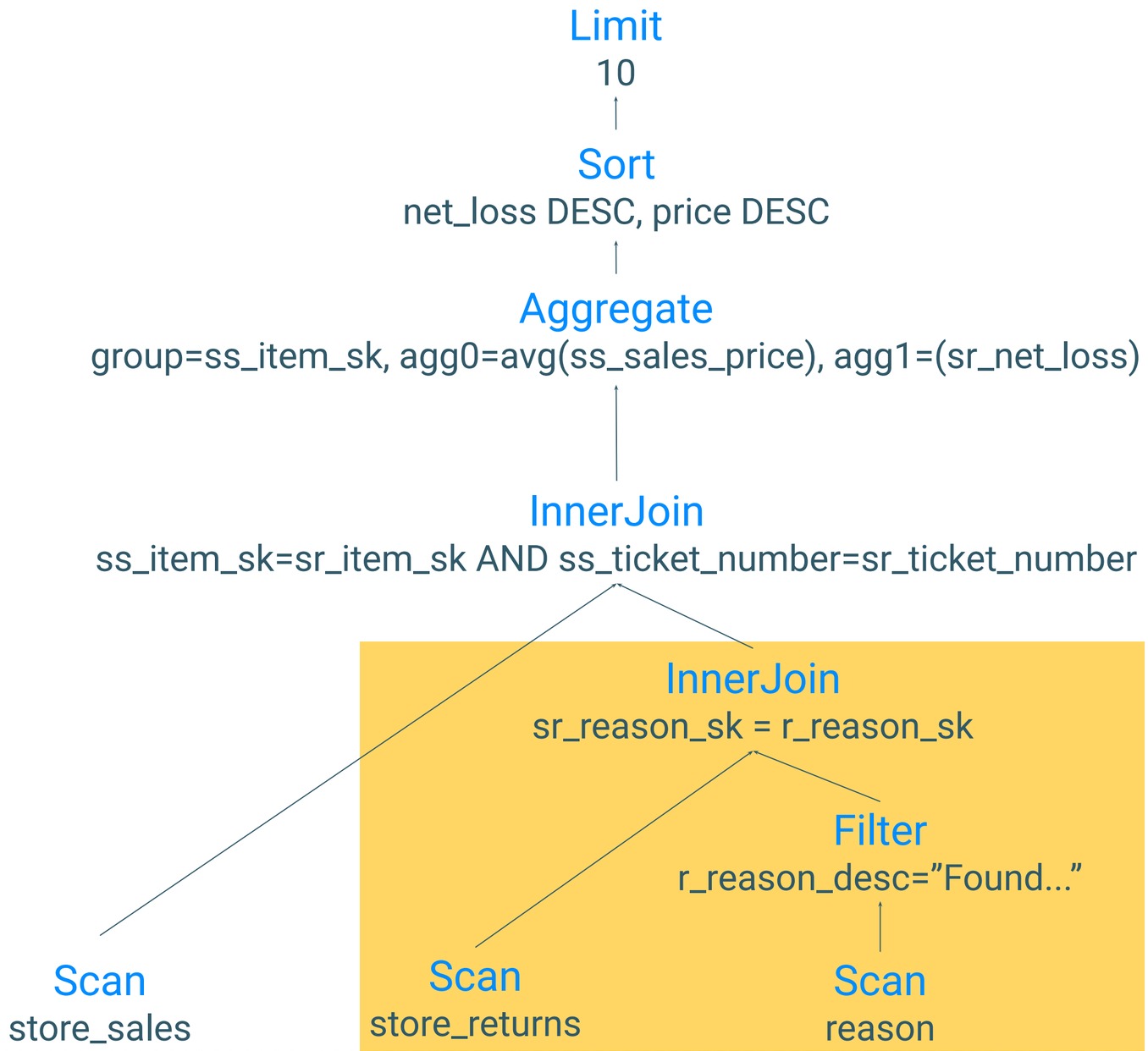
- ➔ Reduce disk & network I/O
- ➔ Save CPU cycles
- ➔ Decrease memory footprint

TPC-DS Query Example

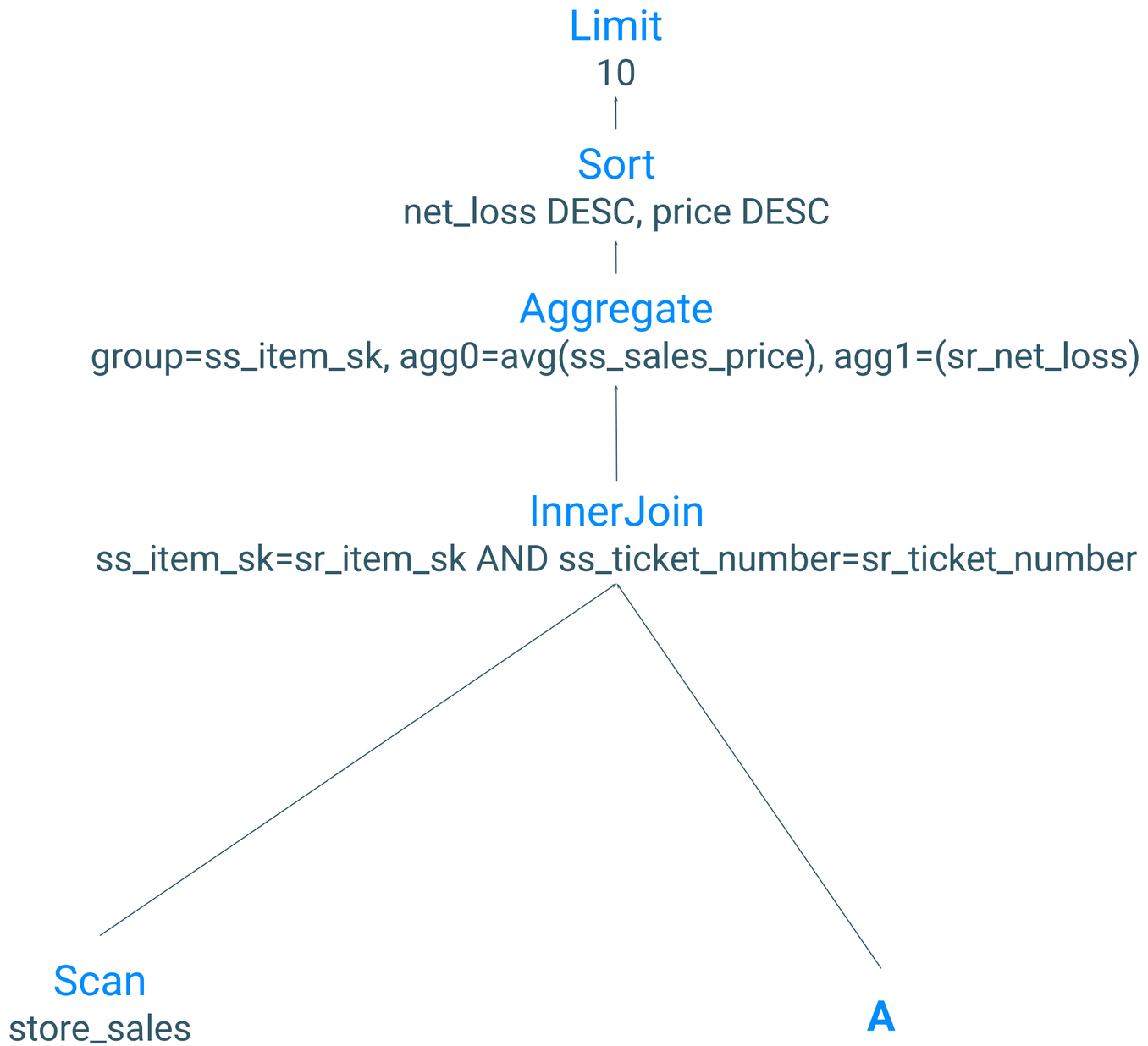
Find the ten items that were returned due to a more competitive price

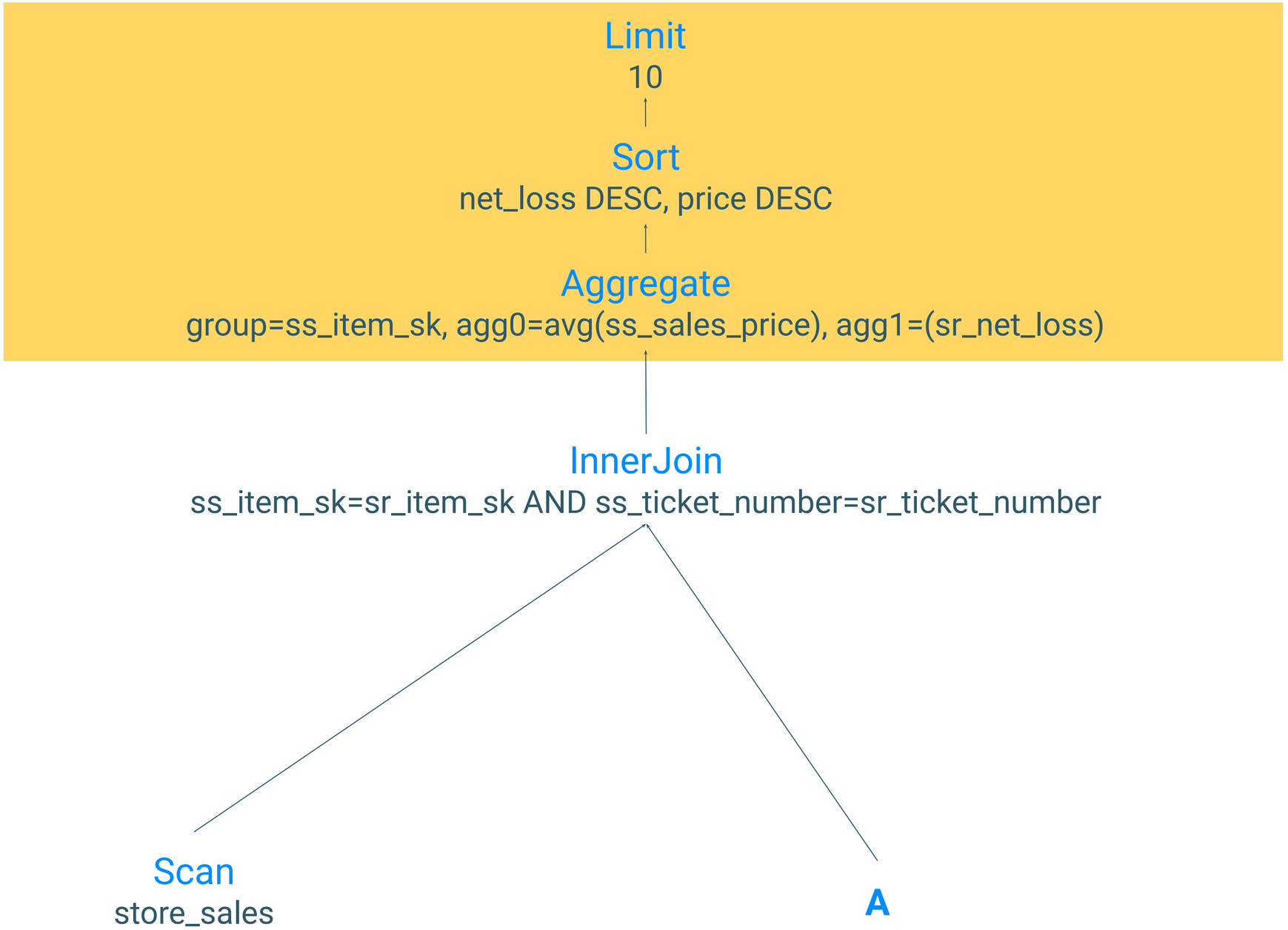
```
SELECT ss_item_sk, sum(sr_net_loss) as net_loss, avg(ss_sales_price) as price
FROM store_sales
INNER JOIN store_returns
  ON ss_item_sk=sr_item_sk AND ss_ticket_number=sr_ticket_number
INNER JOIN reason
  ON sr_reason_sk = r_reason_sk
WHERE r_reason_desc = 'Found a better price in a store'
GROUP BY ss_item_sk
ORDER BY net_loss DESC NULLS LAST, price DESC NULLS LAST
LIMIT 10;
```



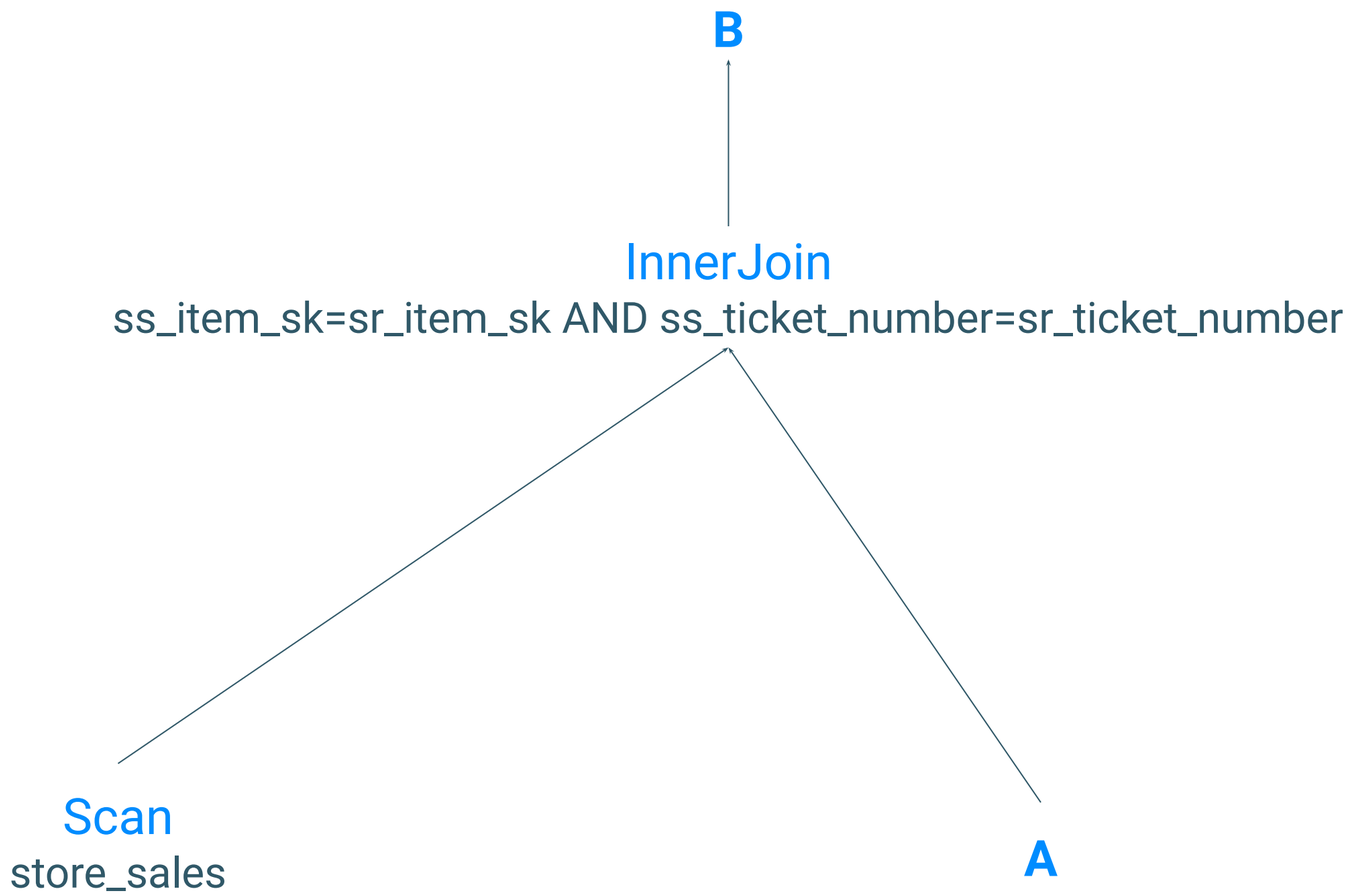


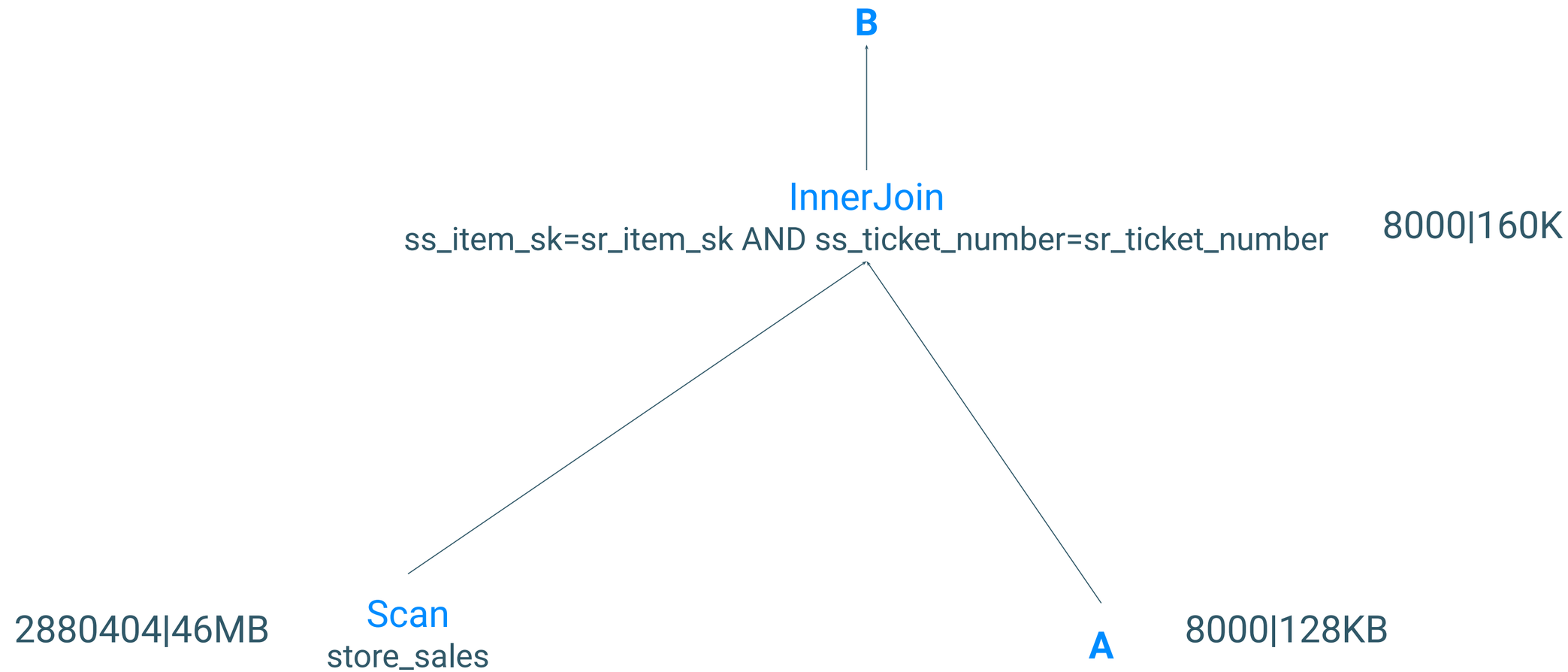
A

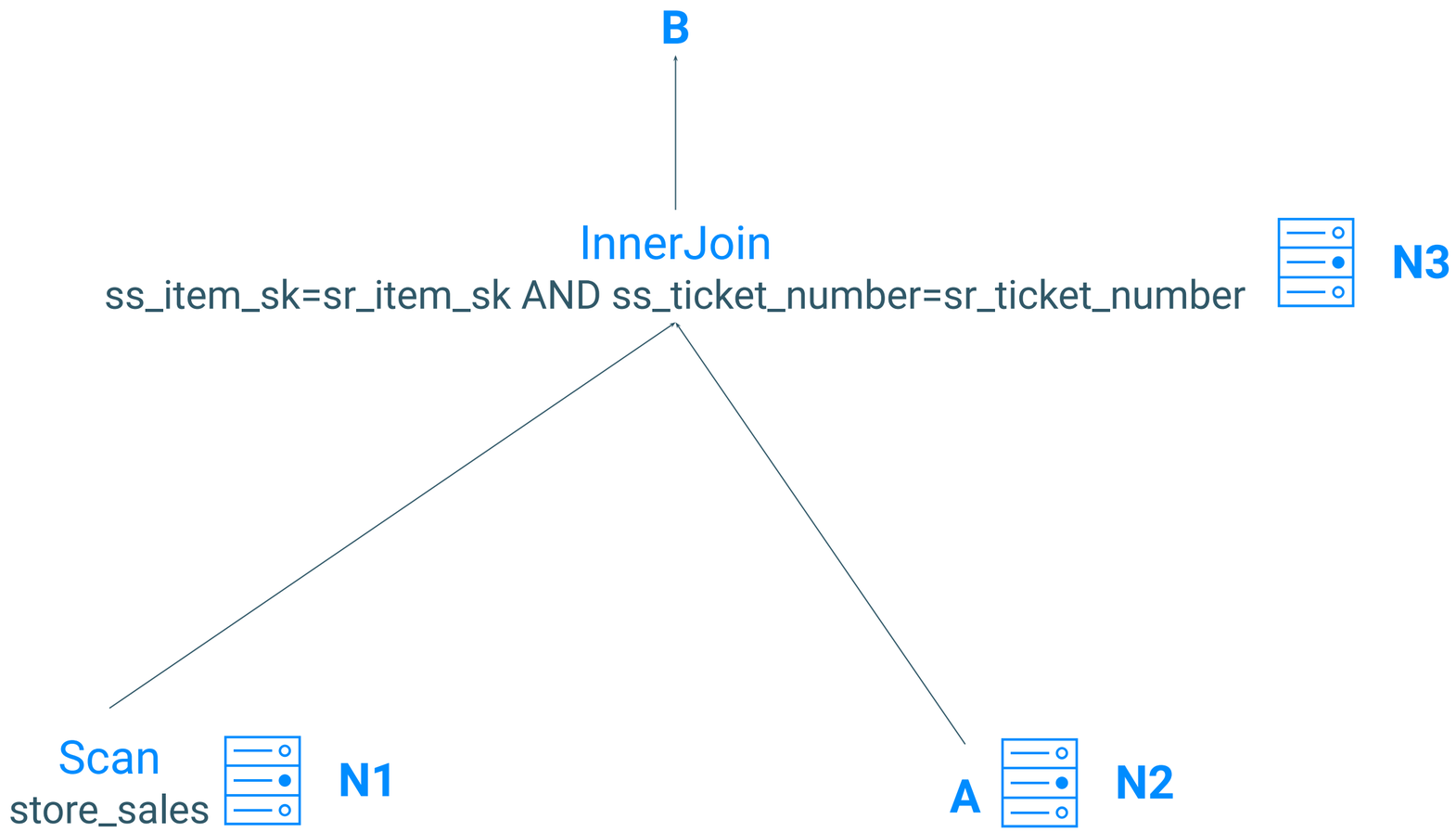


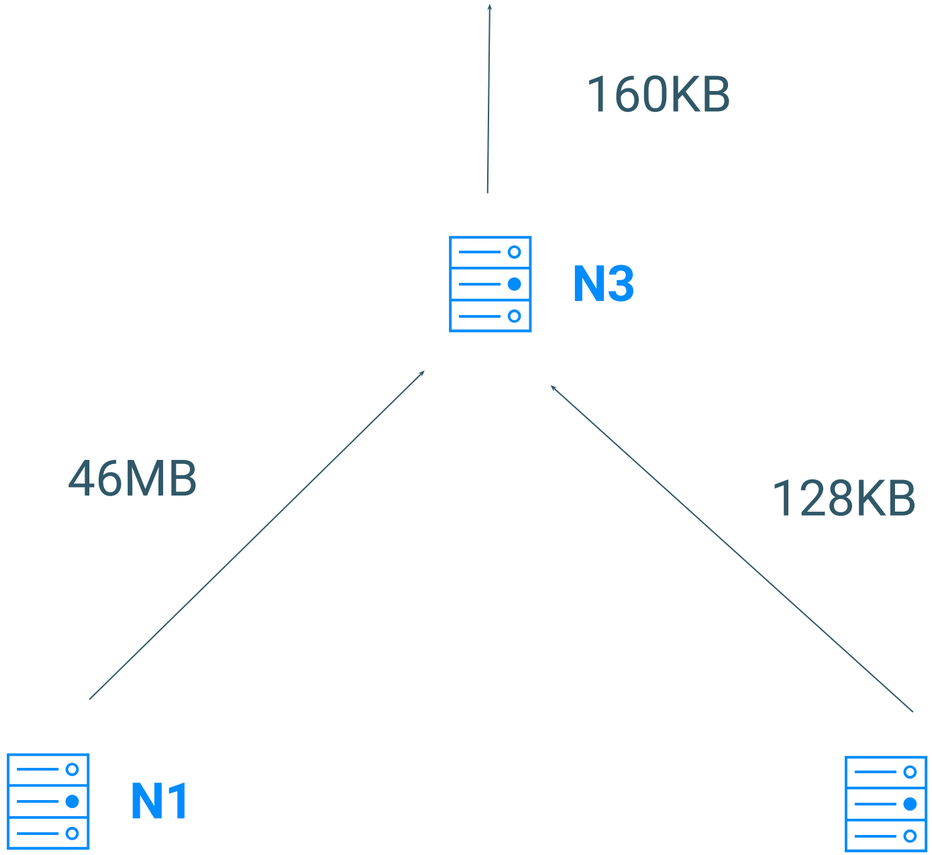


B

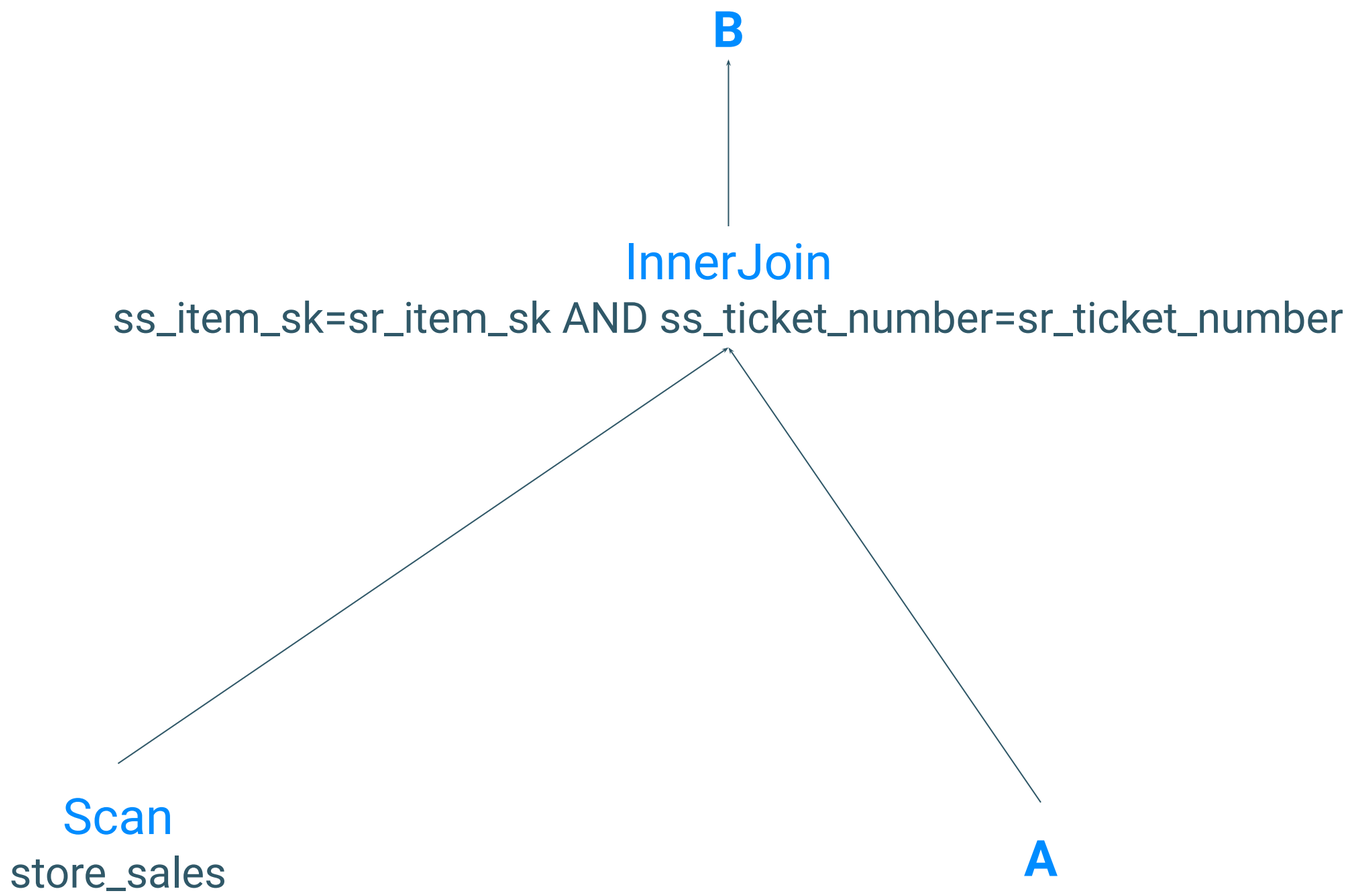


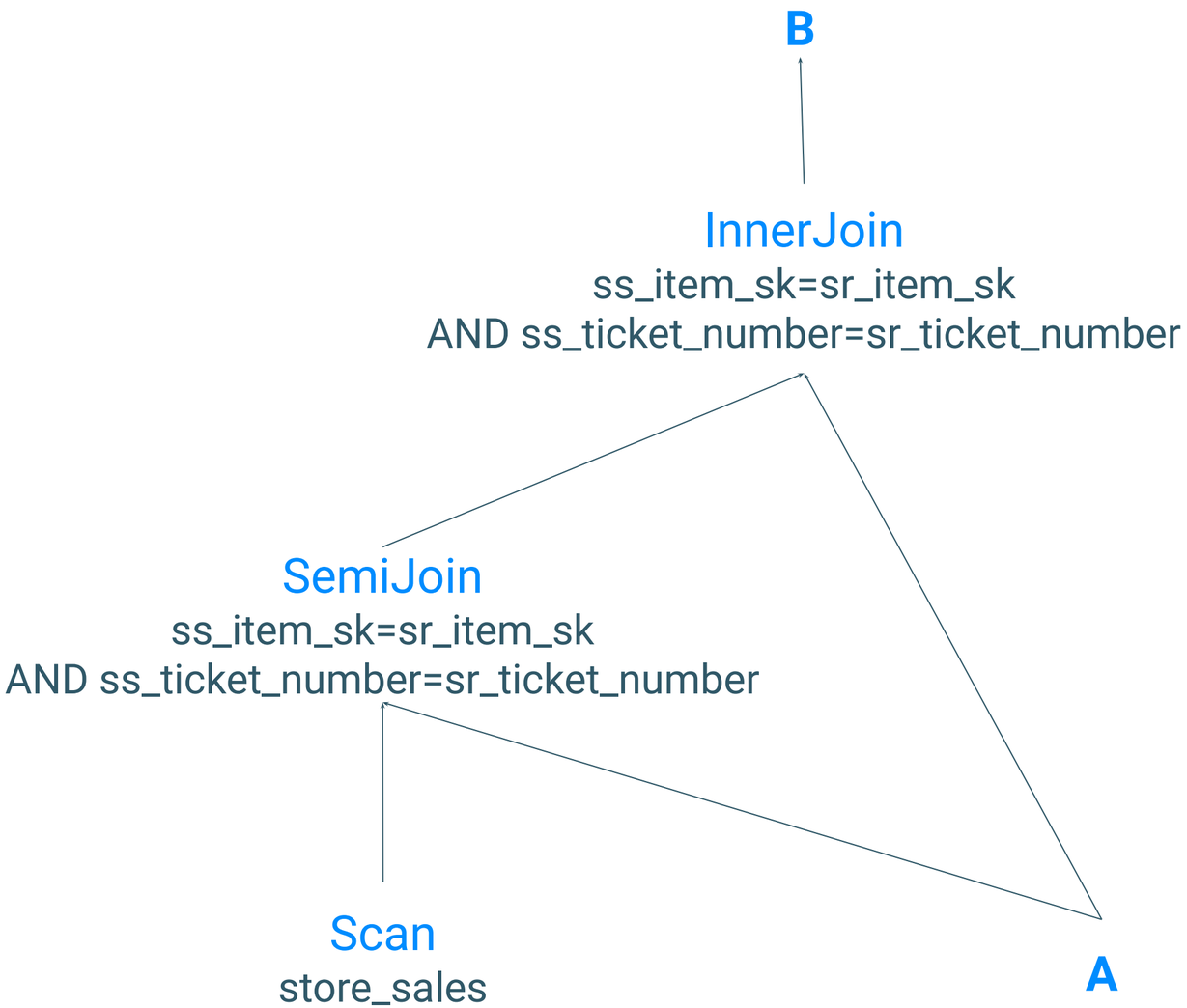


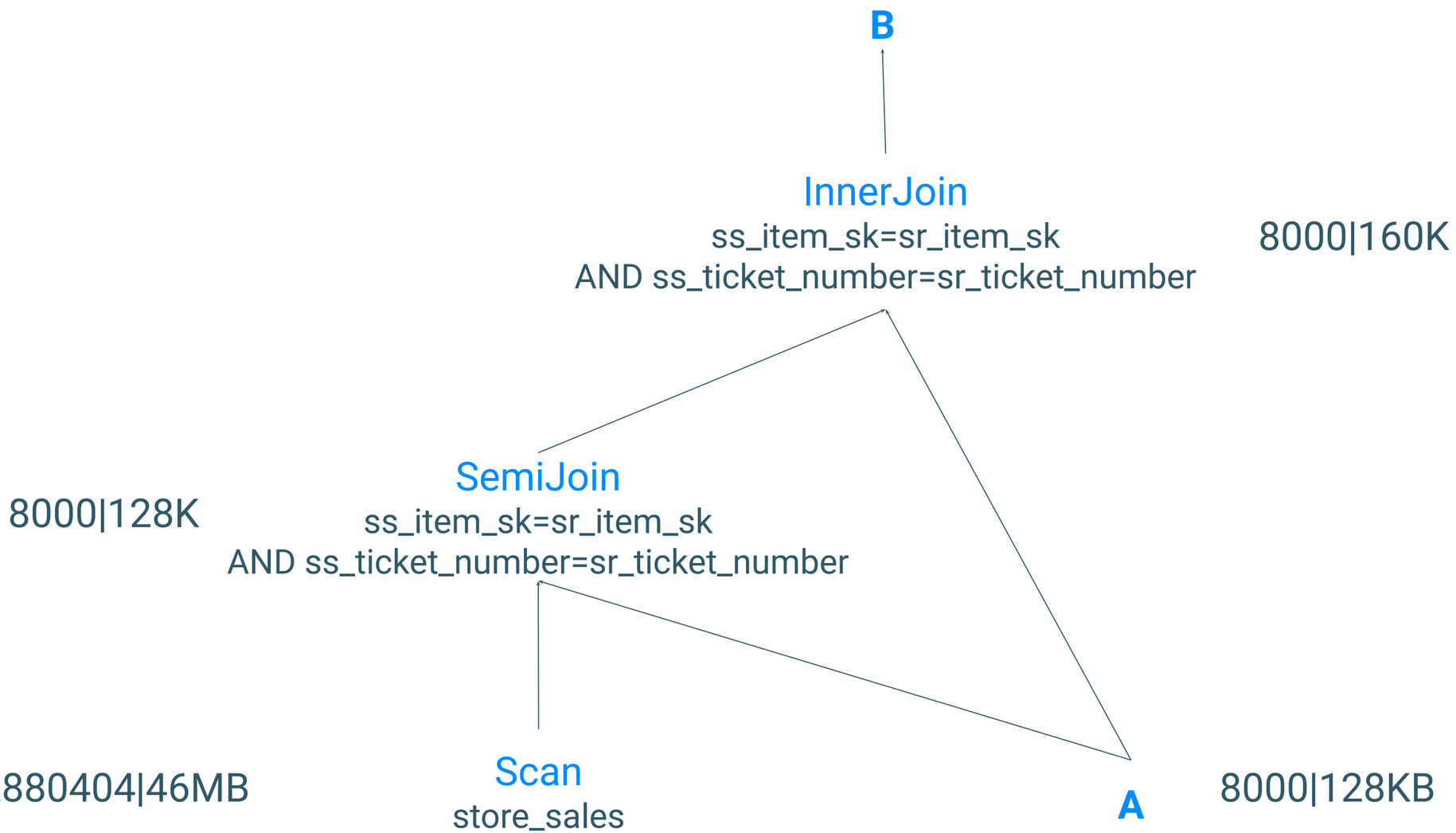


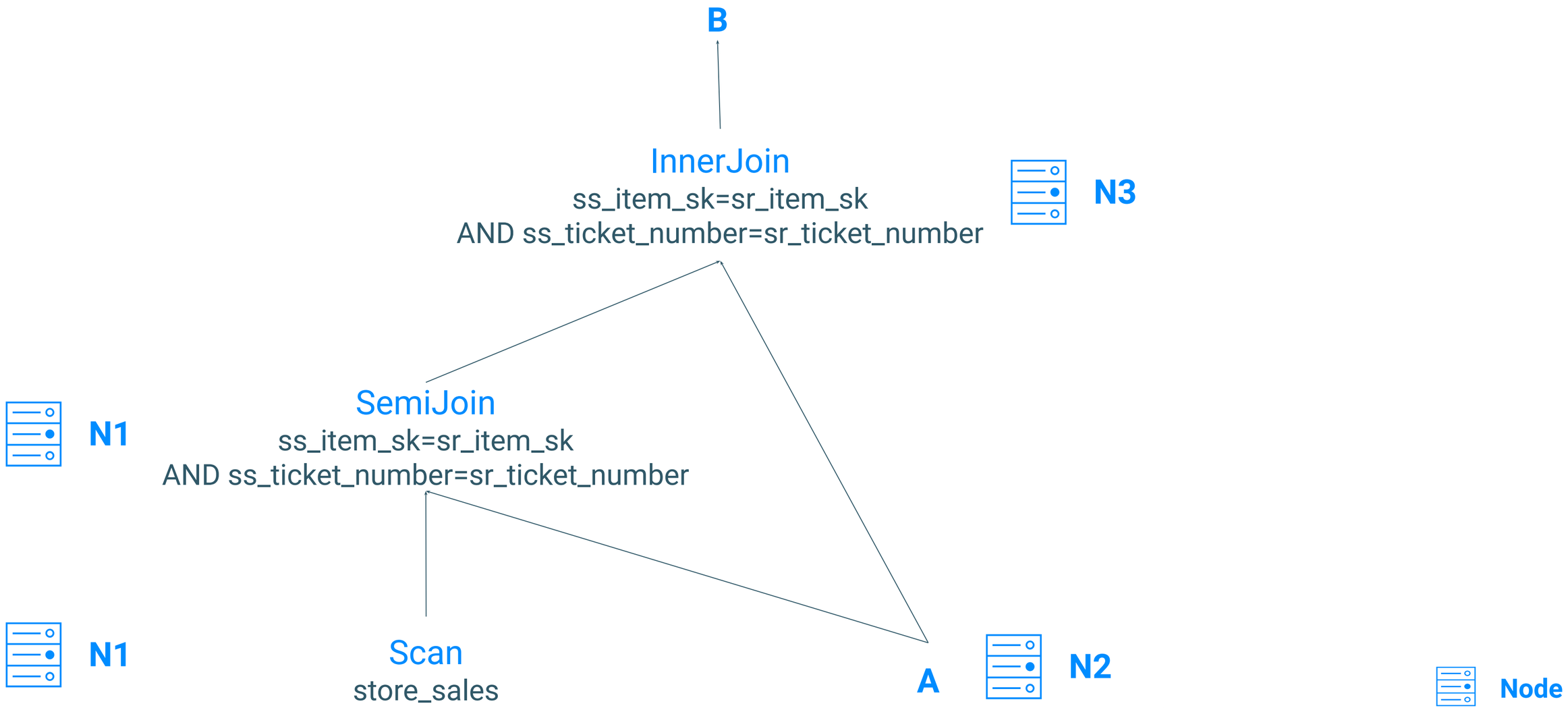


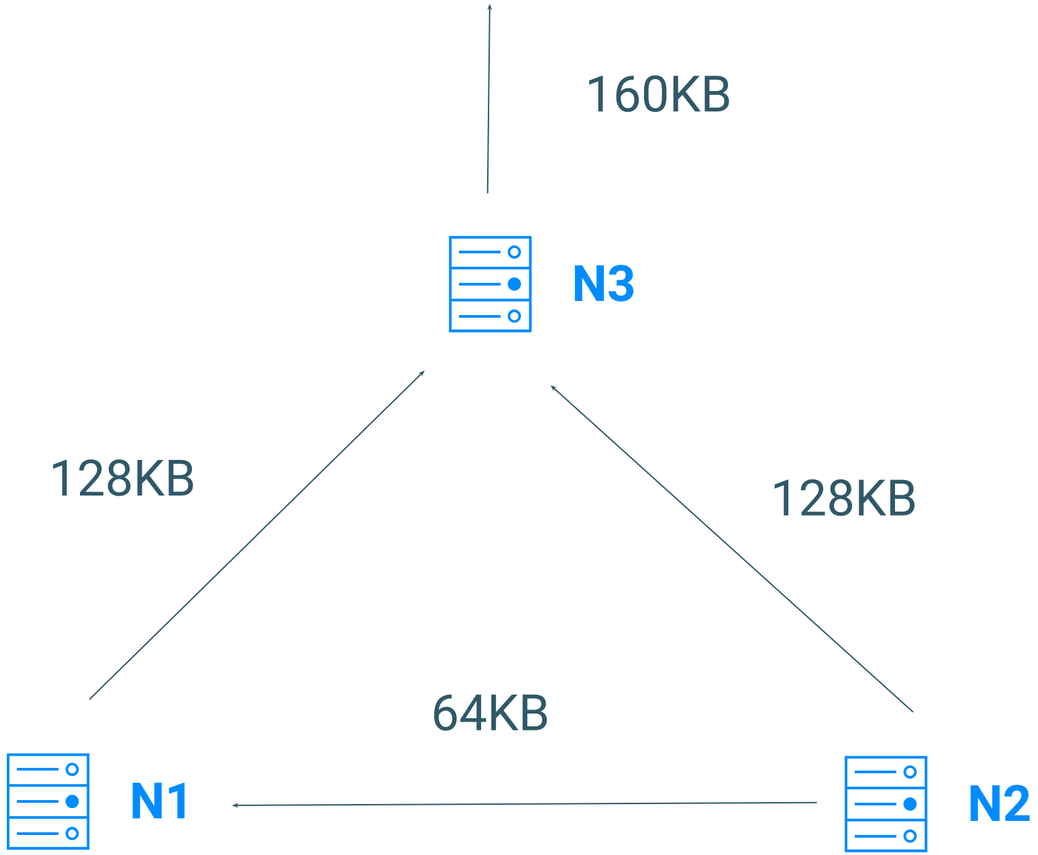
Node





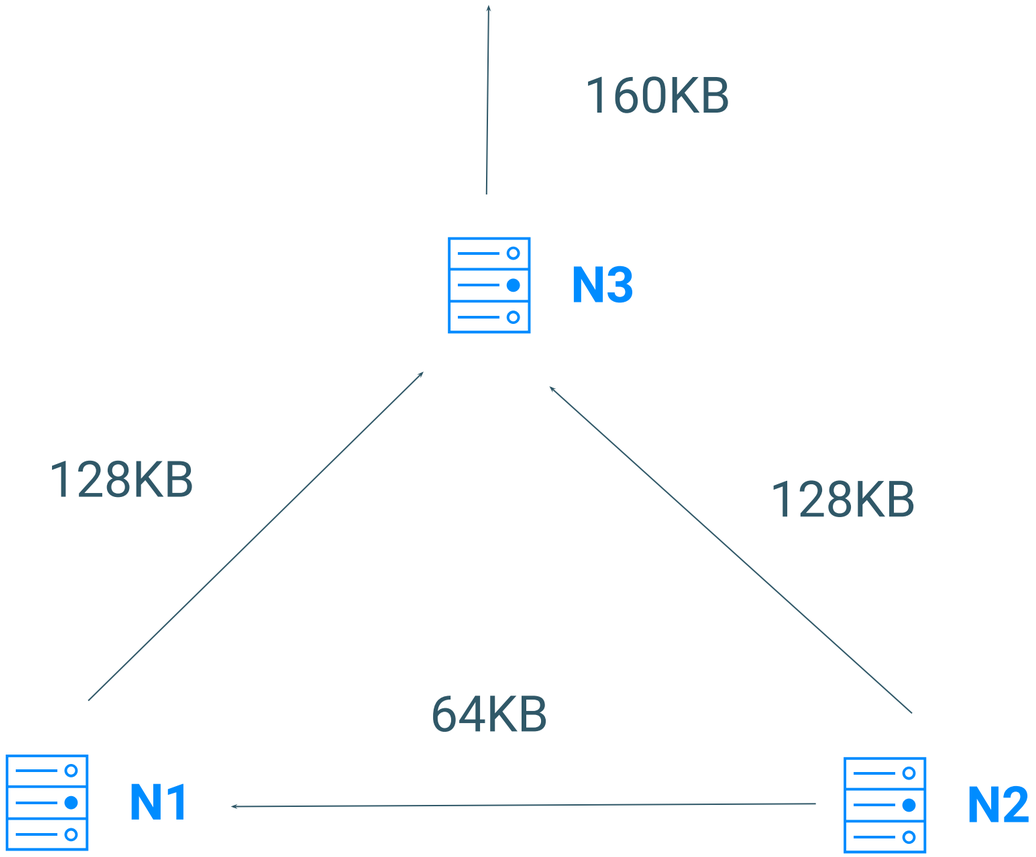




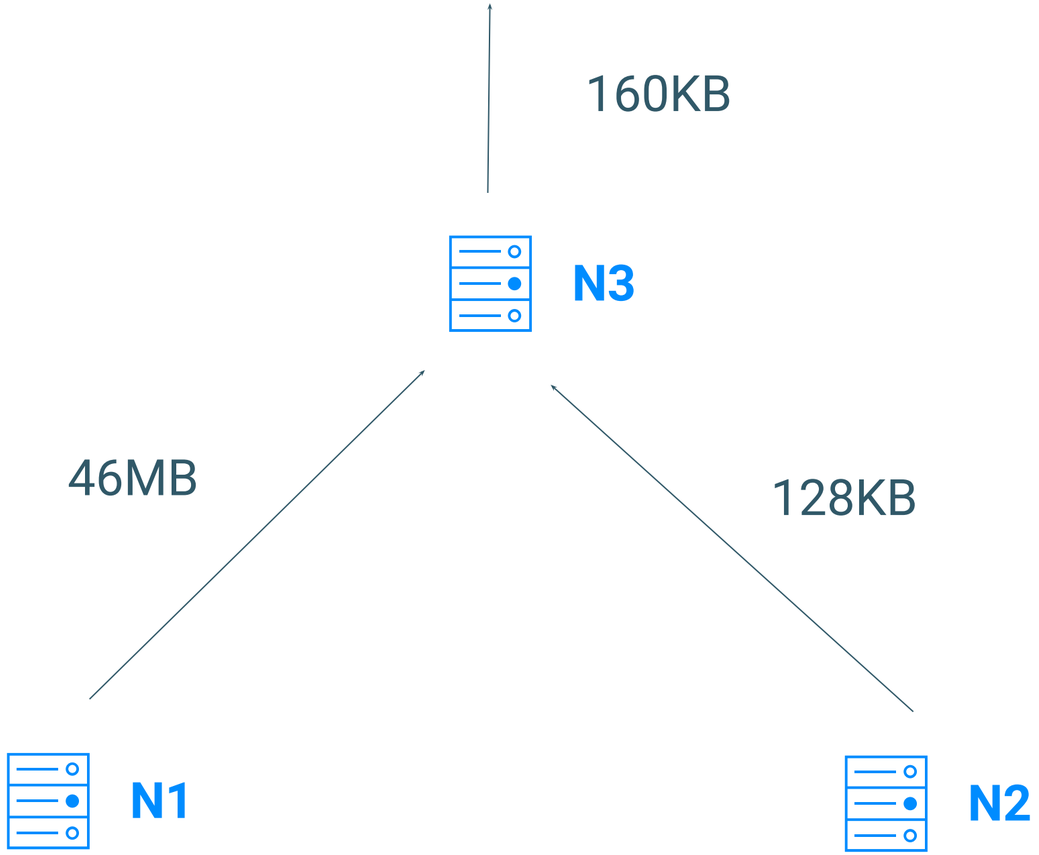


Node

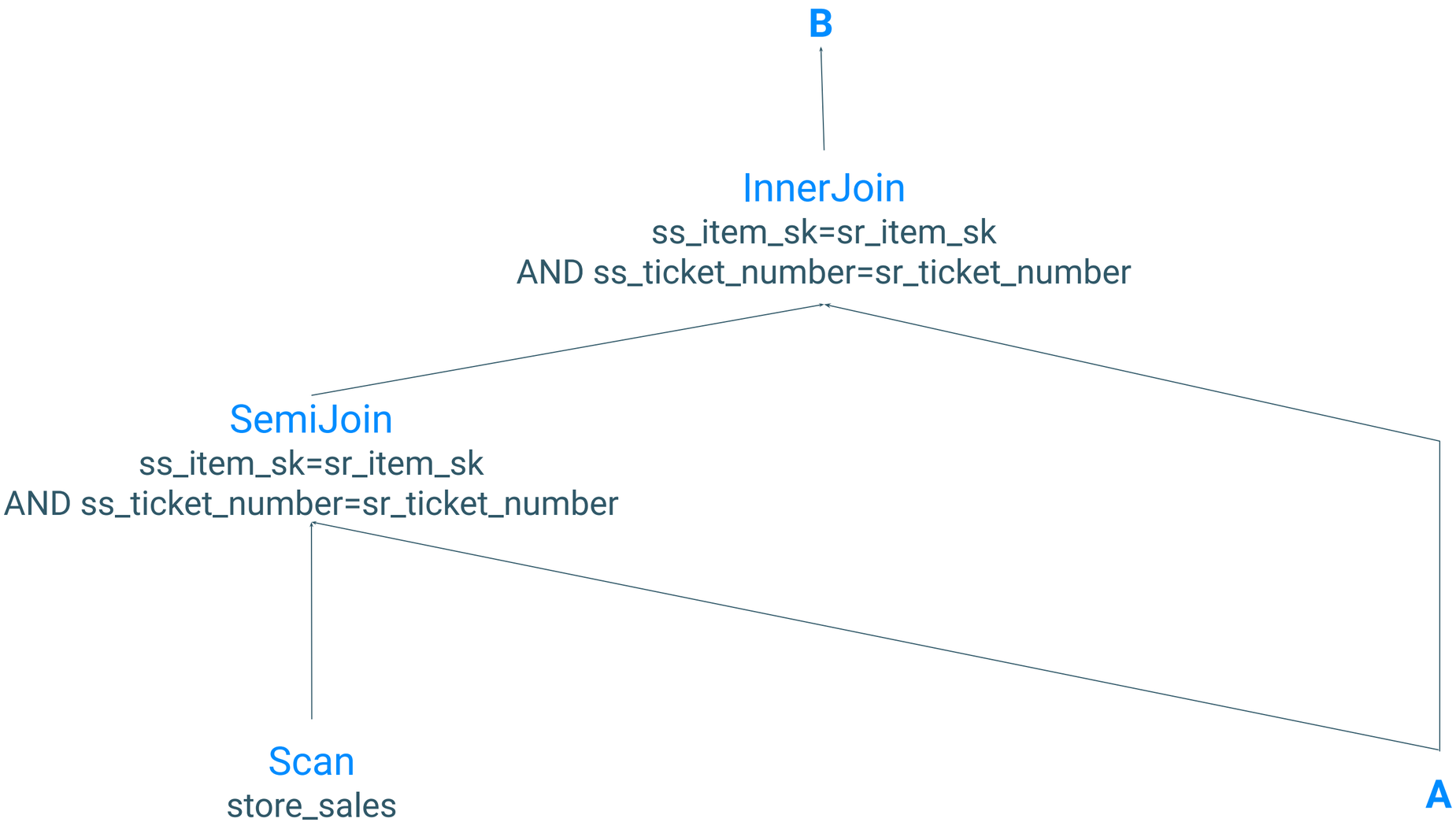
With semijoin
Total I/O=480KB



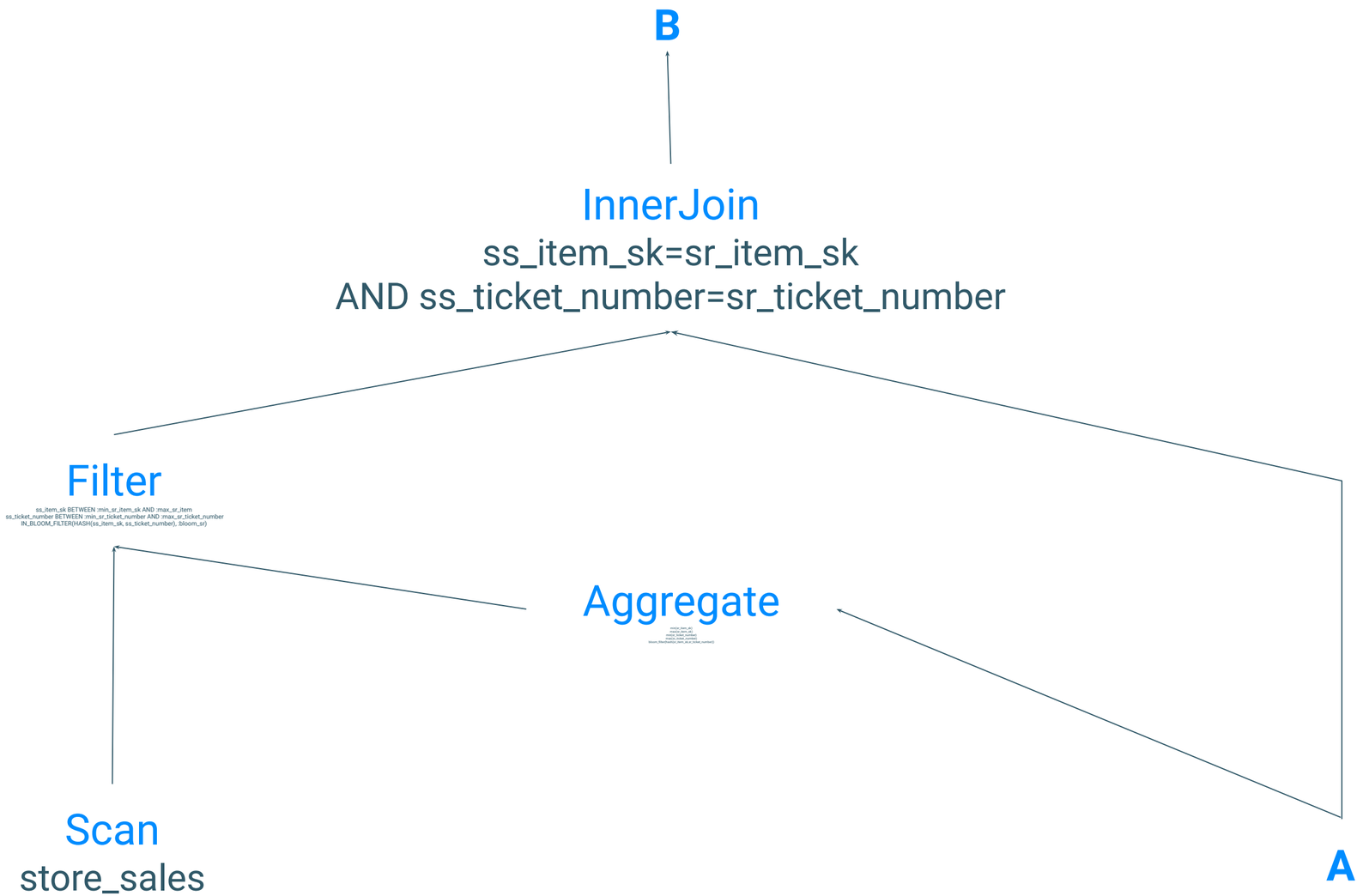
Without semijoin
Total I/O=46MB



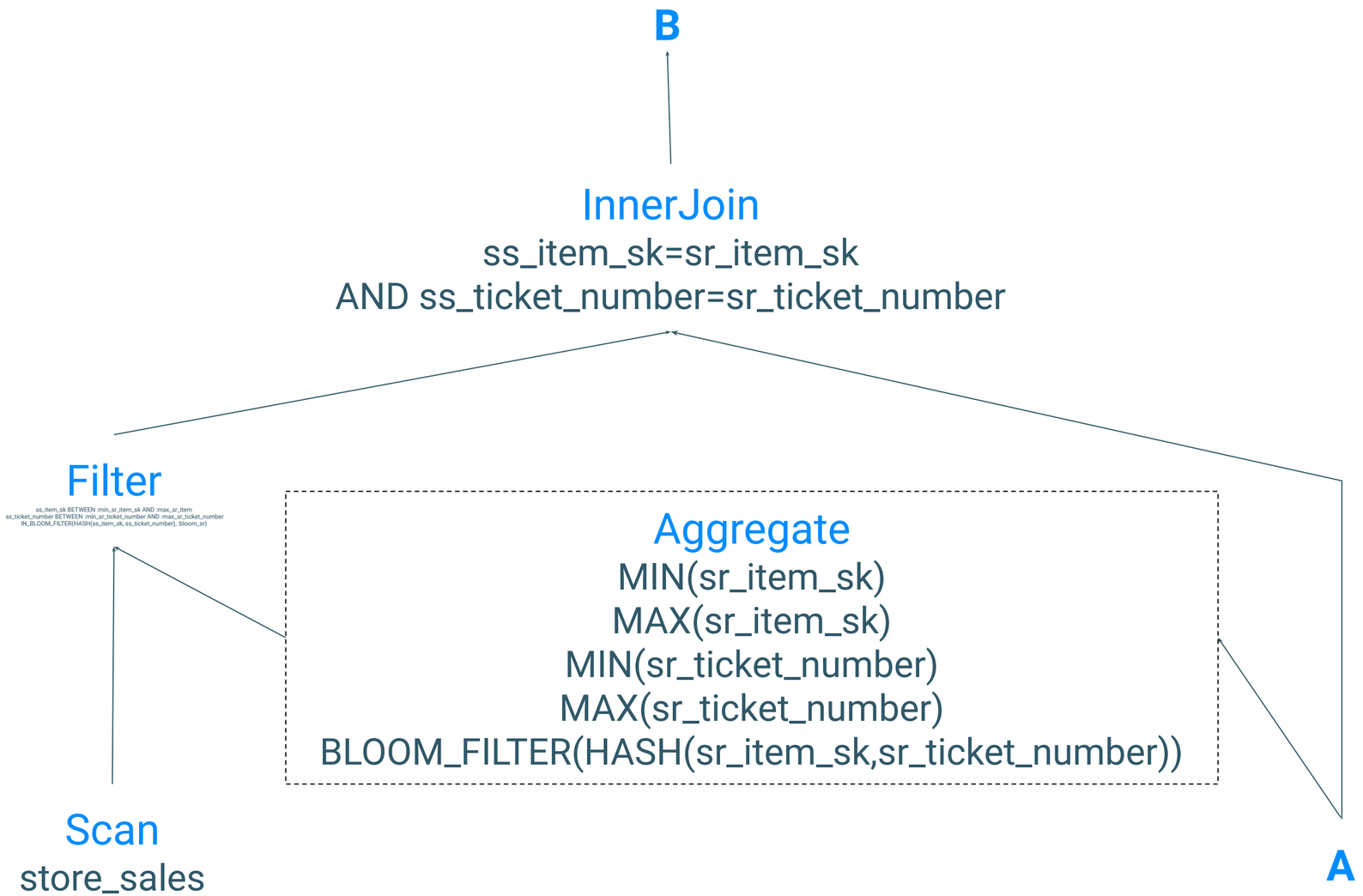
Semijoin reduction in Hive



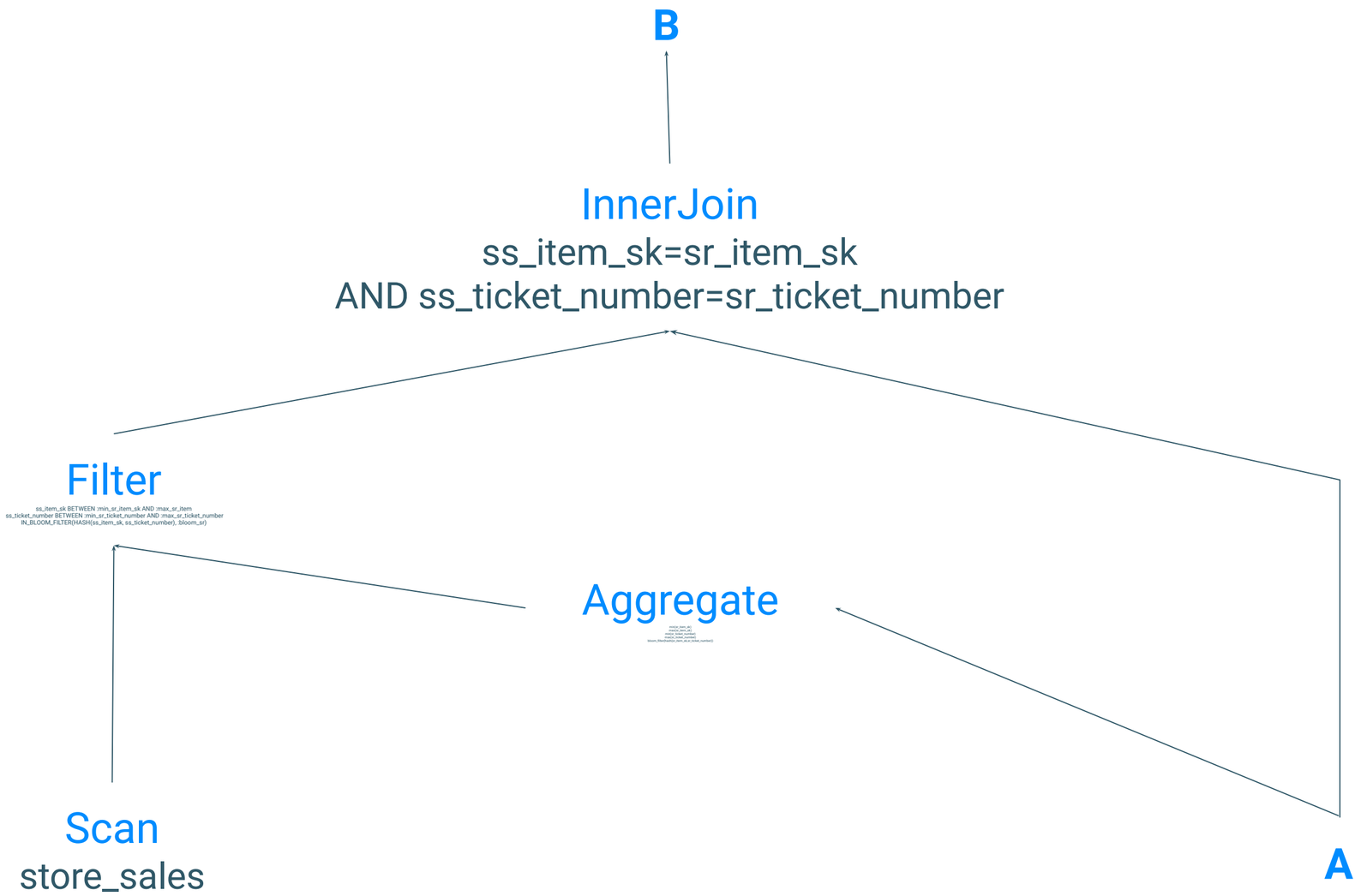
Semijoin reduction in Hive



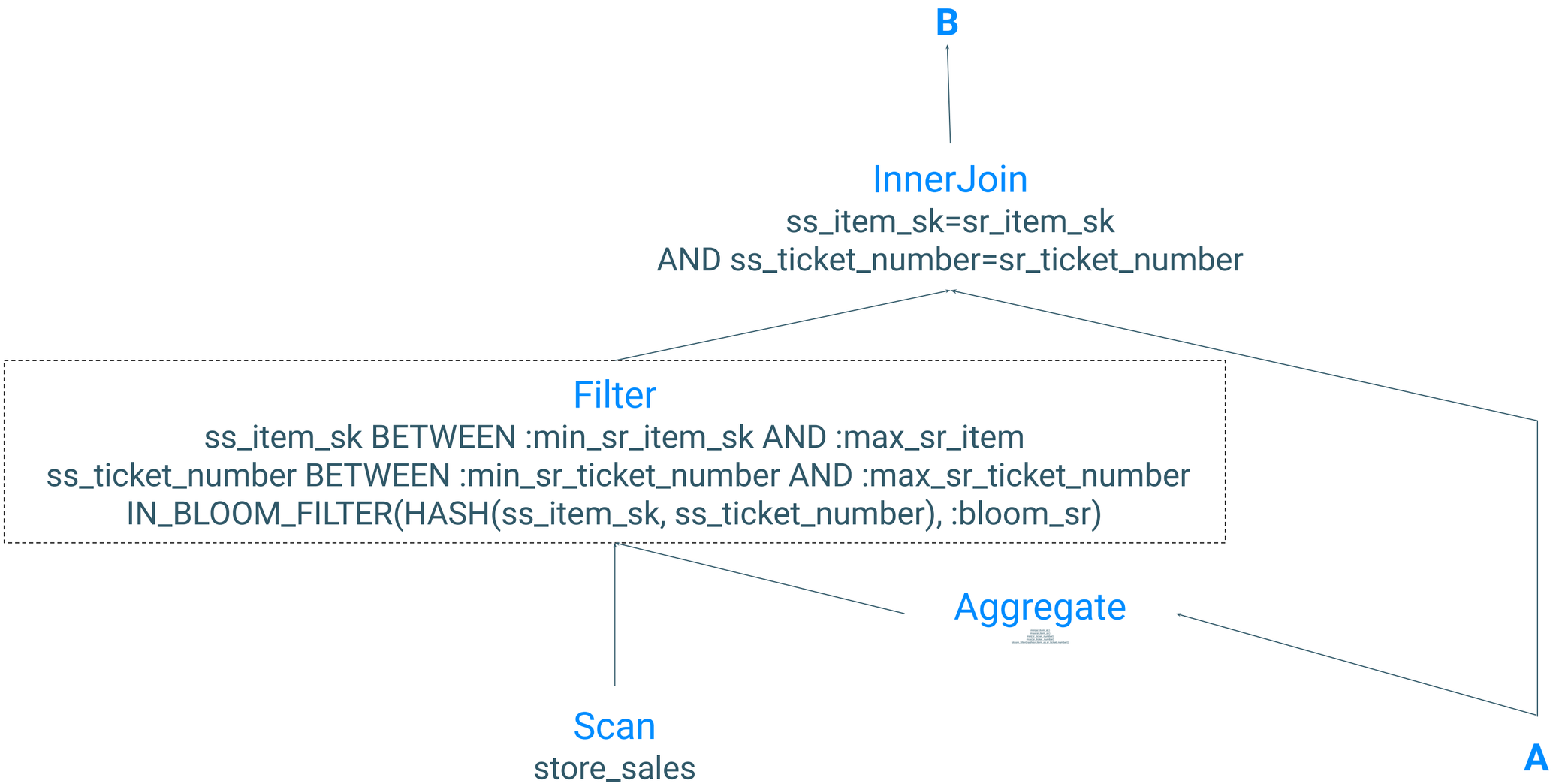
Semijoin reduction in Hive



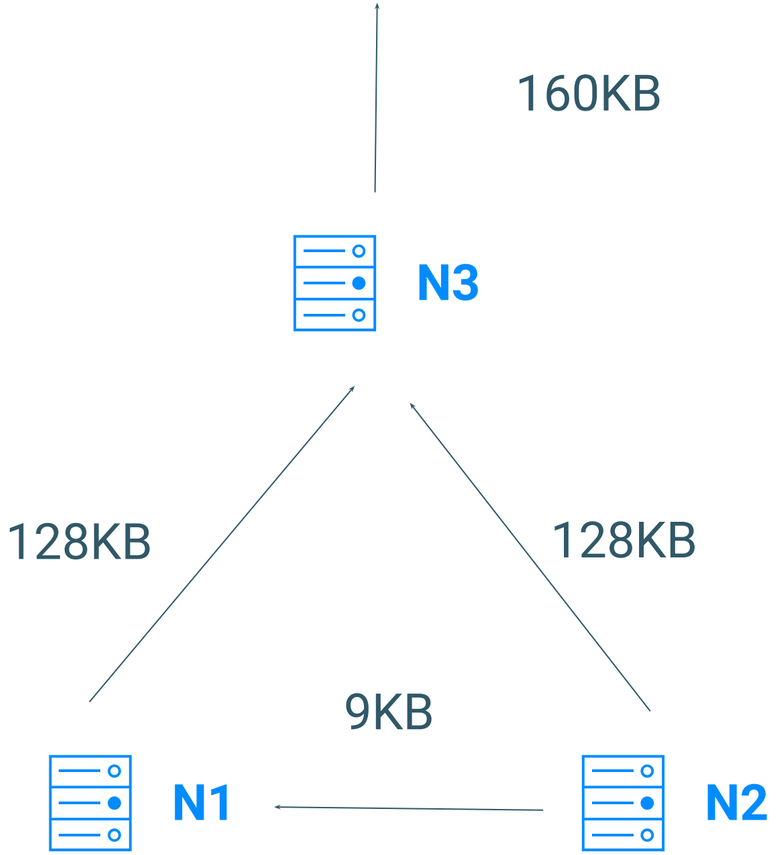
Semijoin reduction in Hive



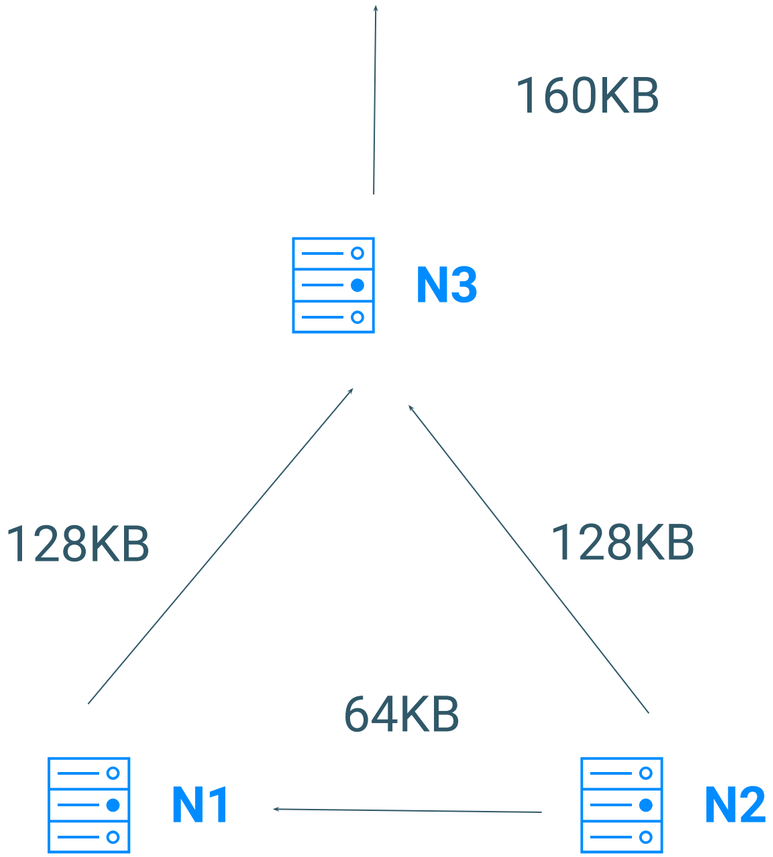
Semijoin reduction in Hive



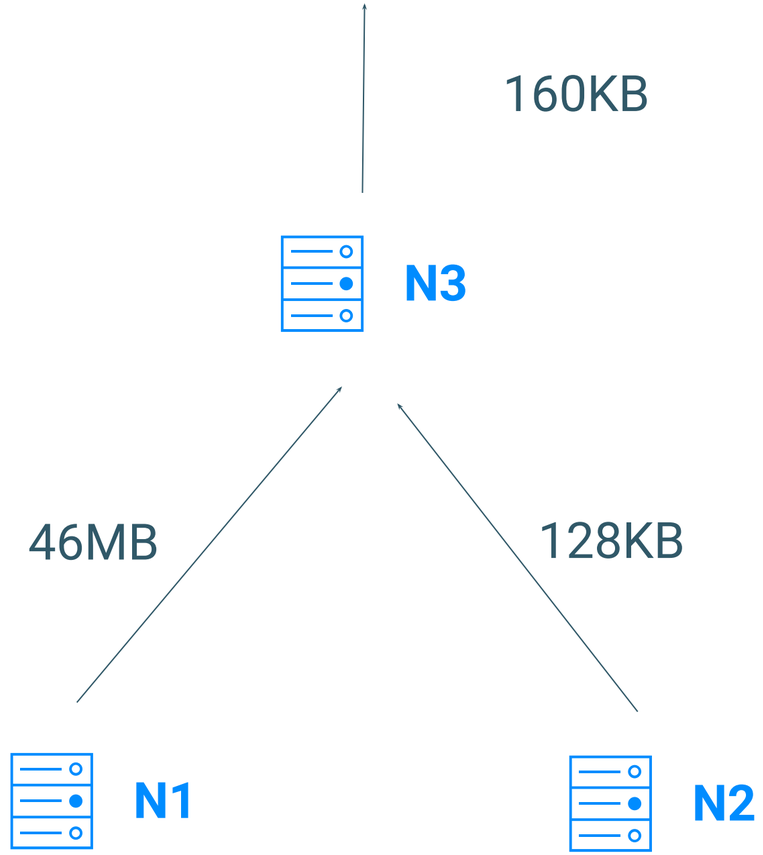
With bloom semijoin
Total I/O=425KB



With semijoin
Total I/O=480KB



Without semijoin
Total I/O=46MB



Semijoin reduction in Hive

Four phase overview

Create synthetic predicate (SyntheticJoinPredicate)

Expand to semijoin reducers (DynamicPartitionPruningOptimization)

Merge to multi column semijoin reducers (SemiJoinReductionMerge)

- HIVE-21196
- HIVE-23976

Remove conditionally semijoin reducers (SemiJoinRemoval & TezCompiler)

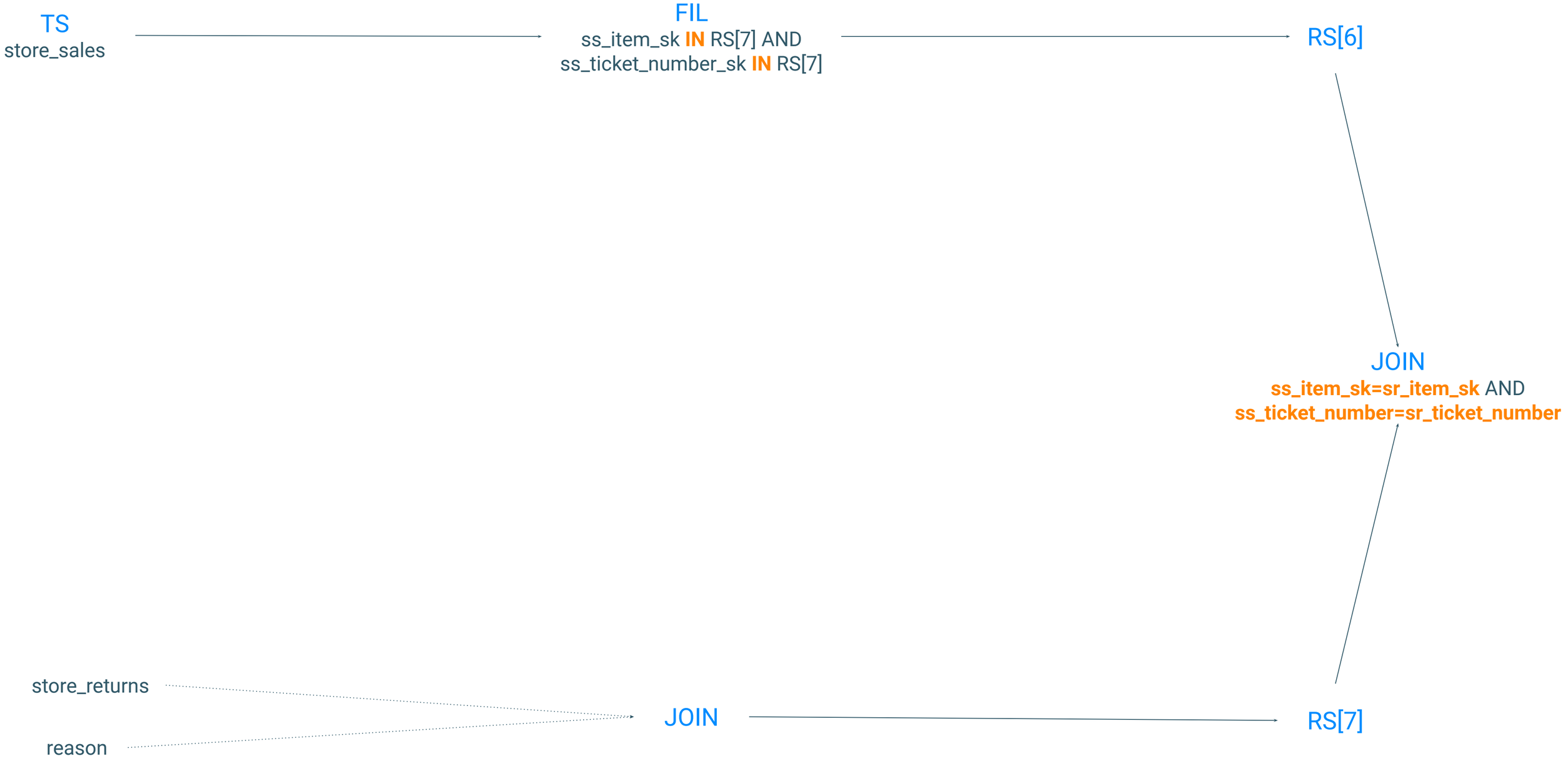
Four phase overview

Create synthetic predicate

Expand to semijoin reducers

Merge to multi column semijoin reducers

Conditionally remove semijoin reducers



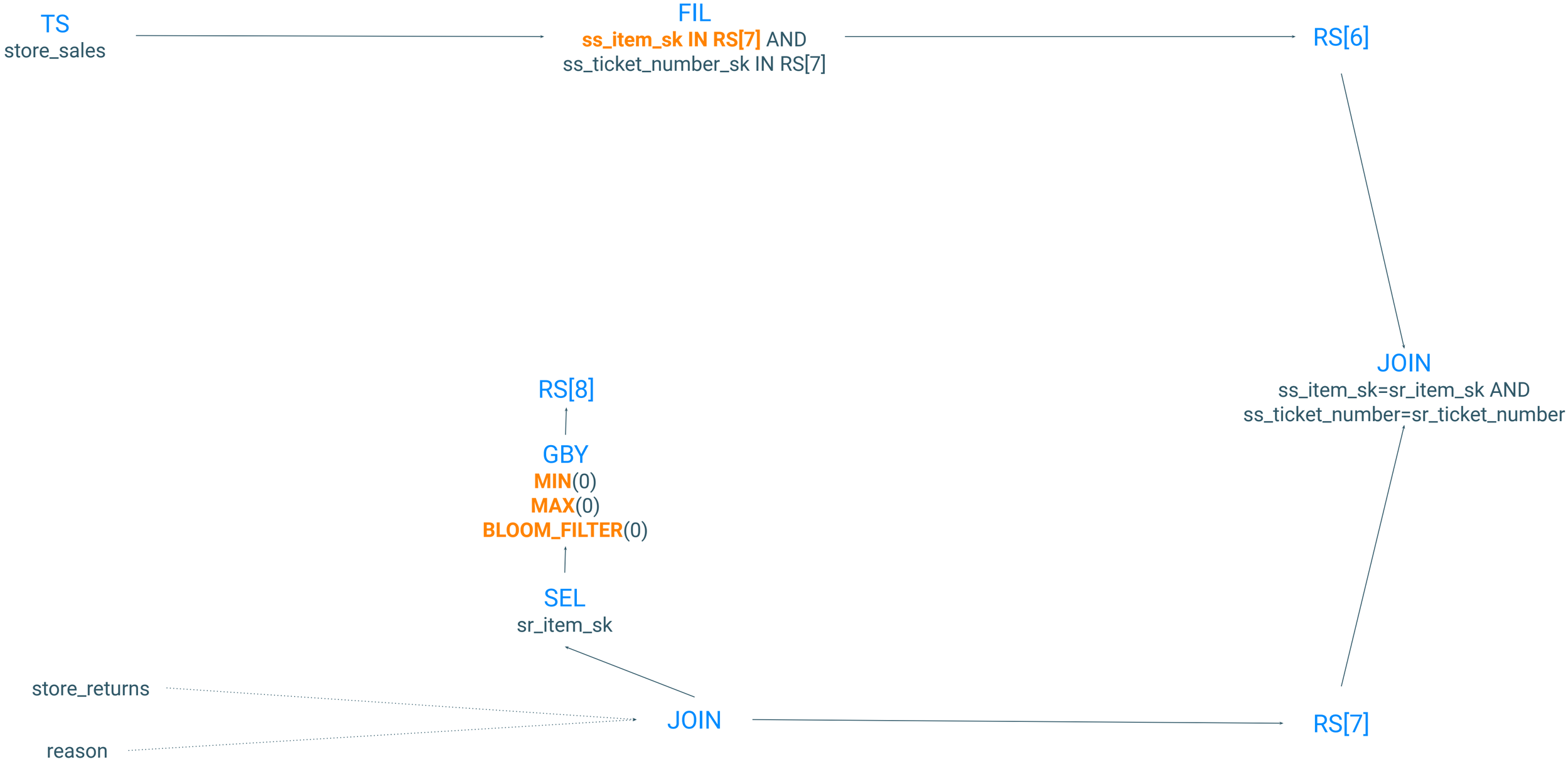
Four phase overview

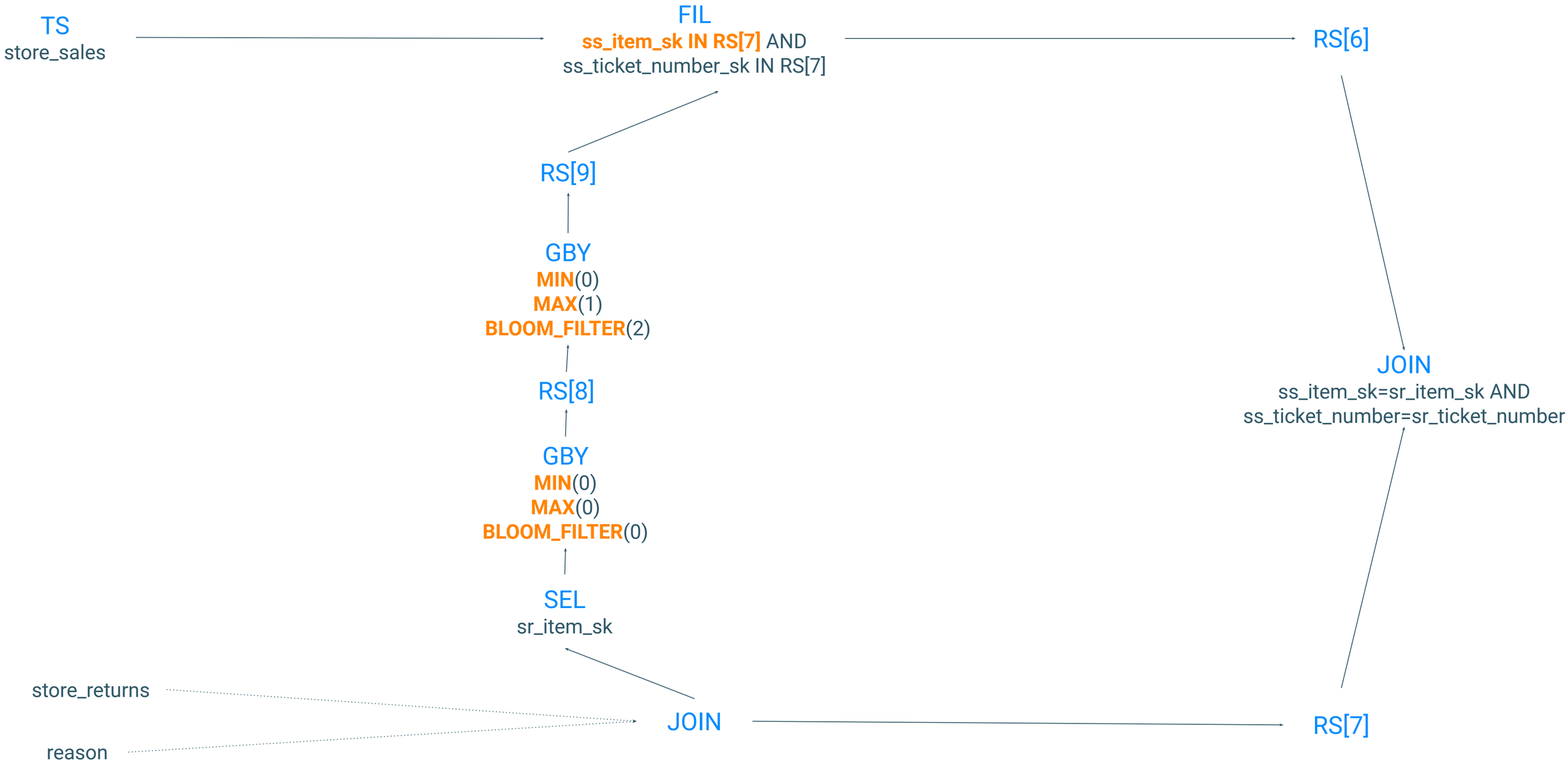
Create synthetic predicate

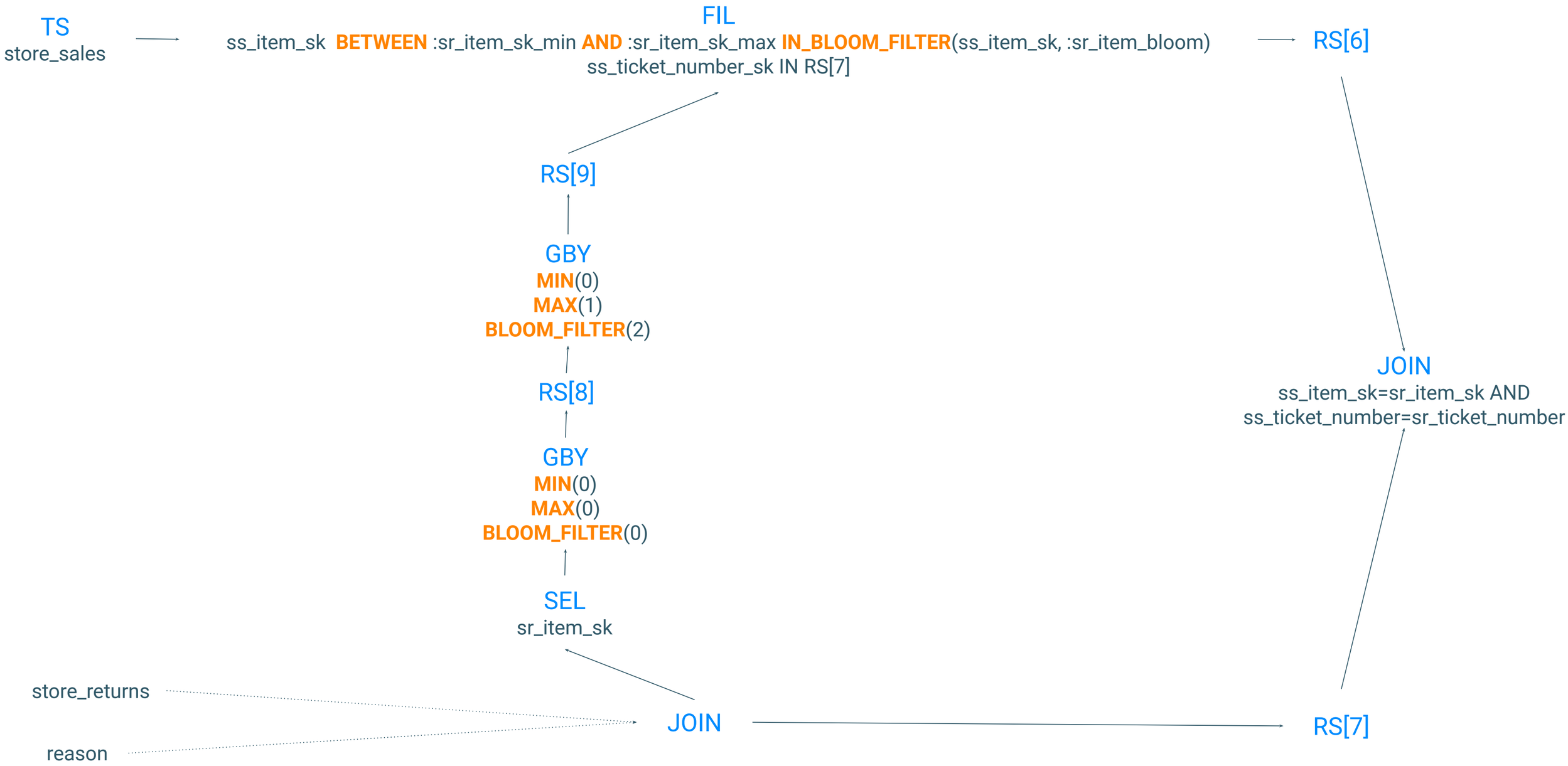
Expand to semijoin reducers

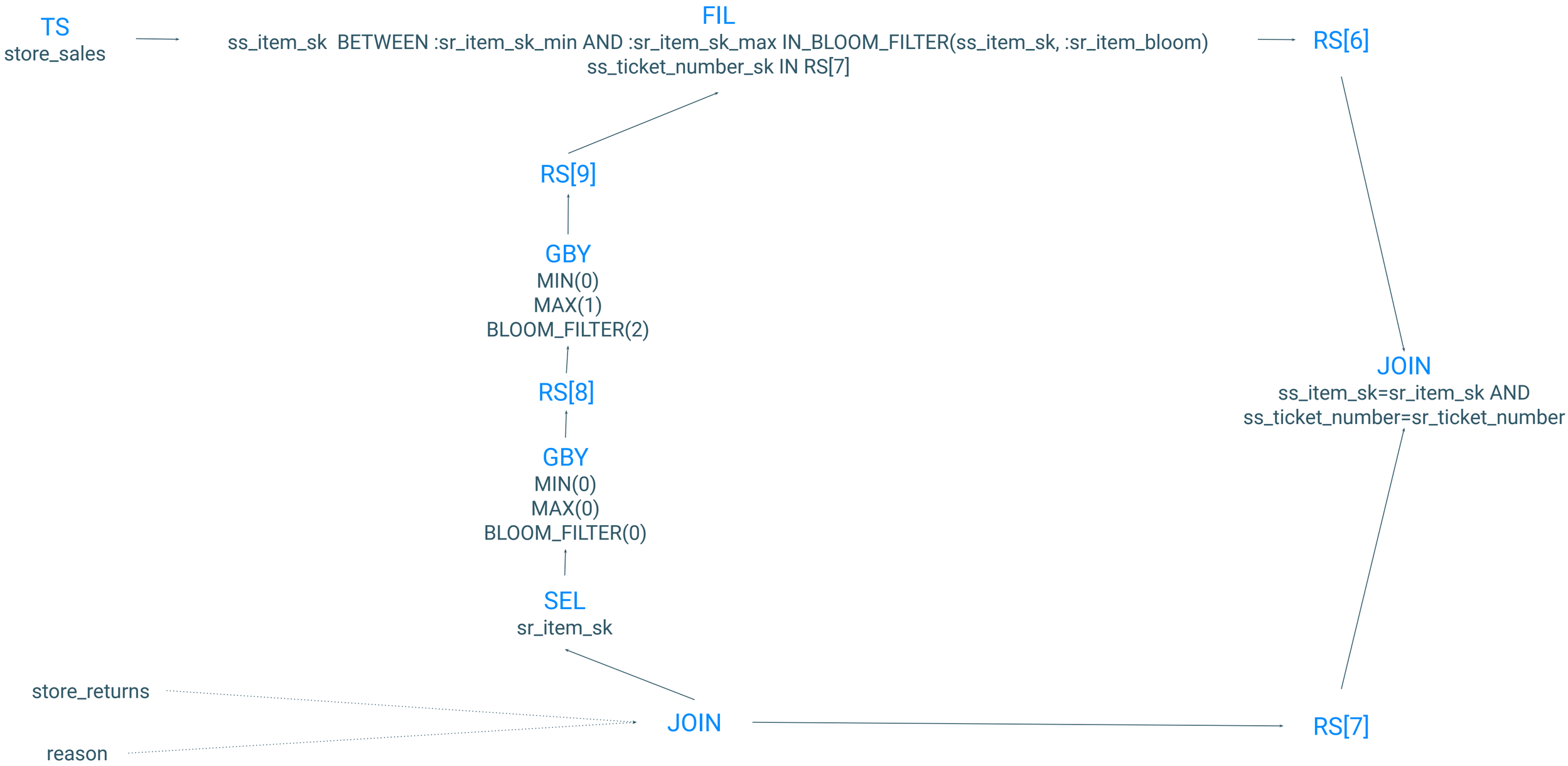
Merge to multi column semijoin reducers

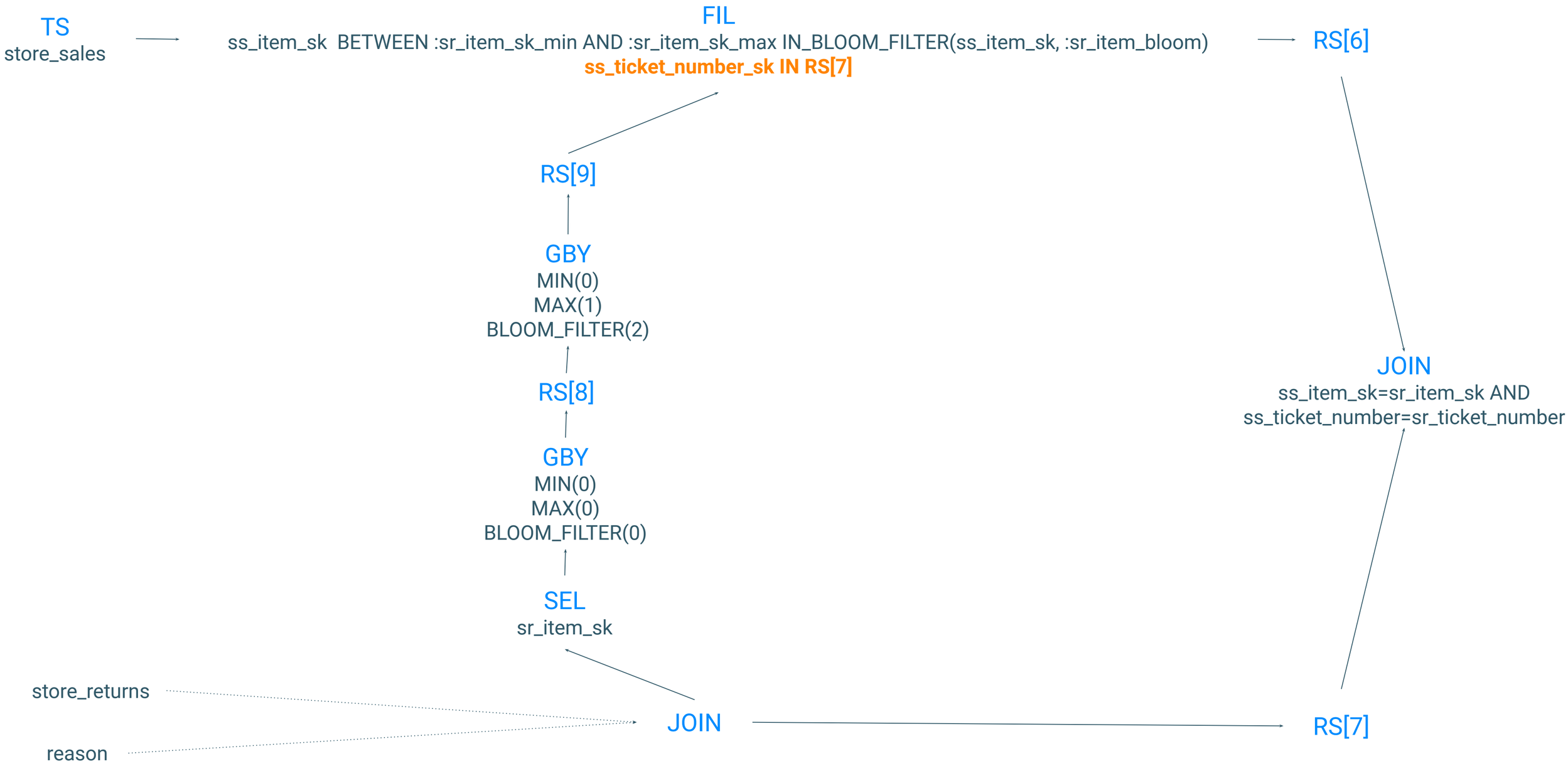
Conditionally remove semijoin reducers

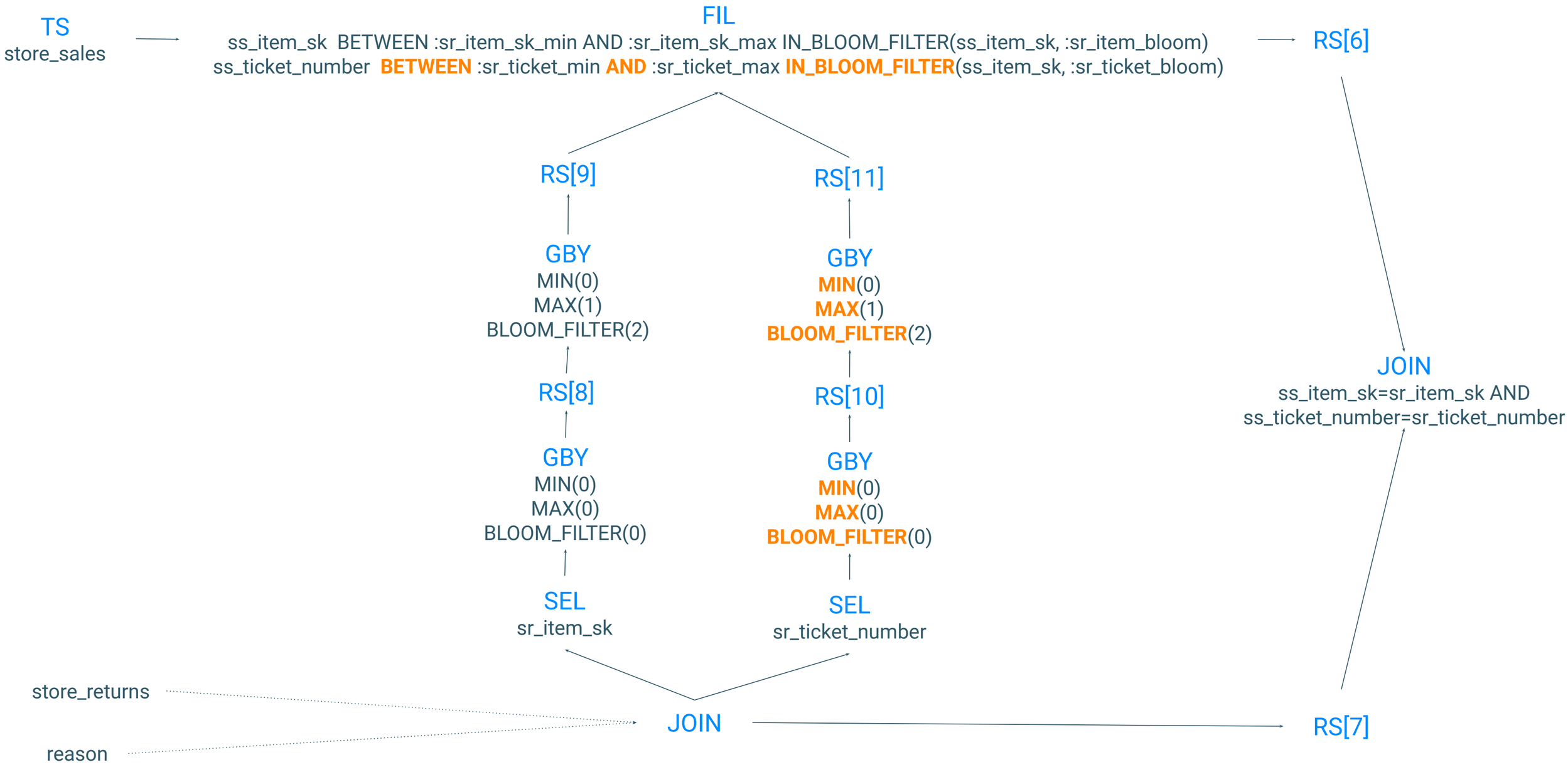


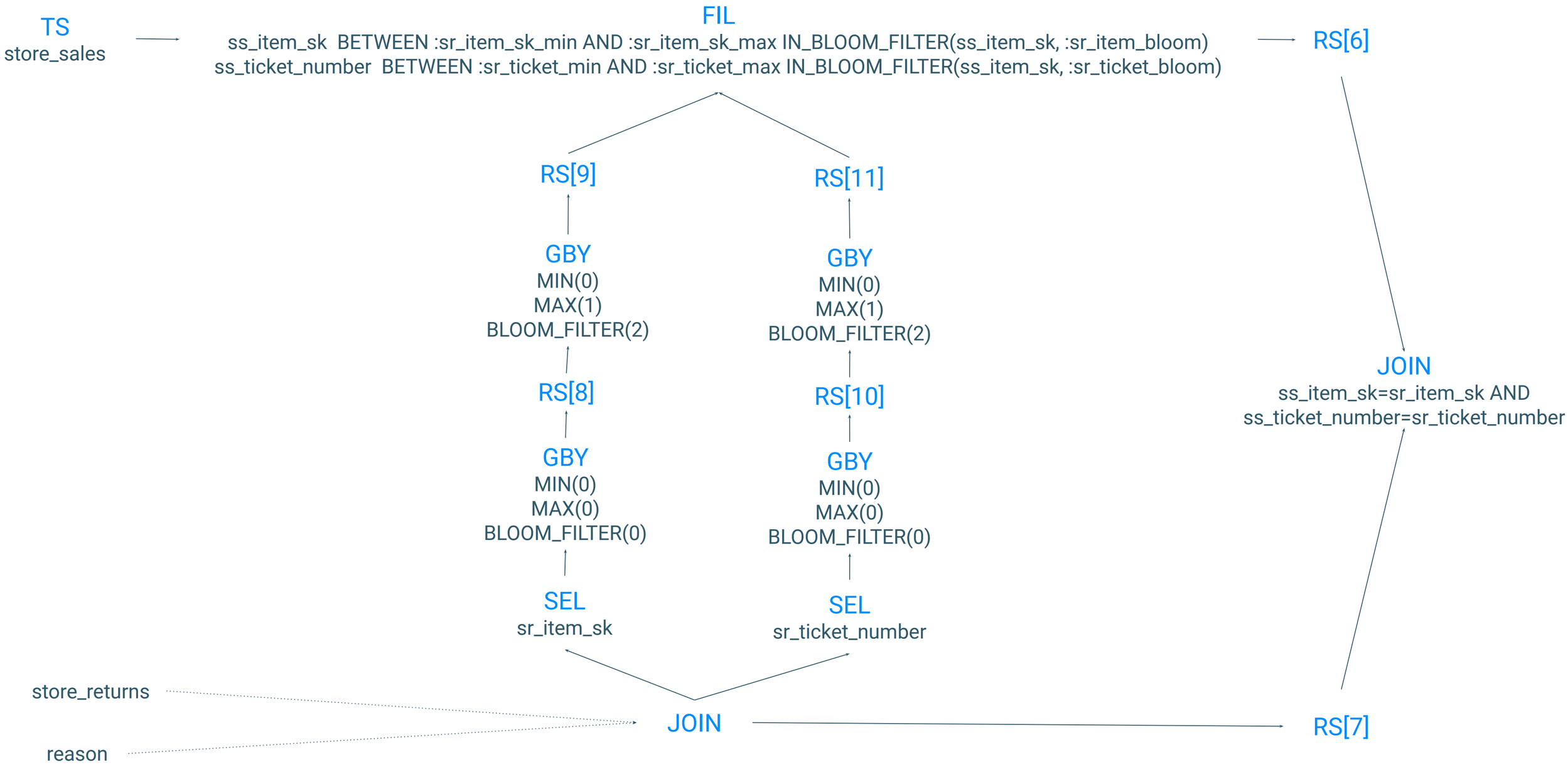












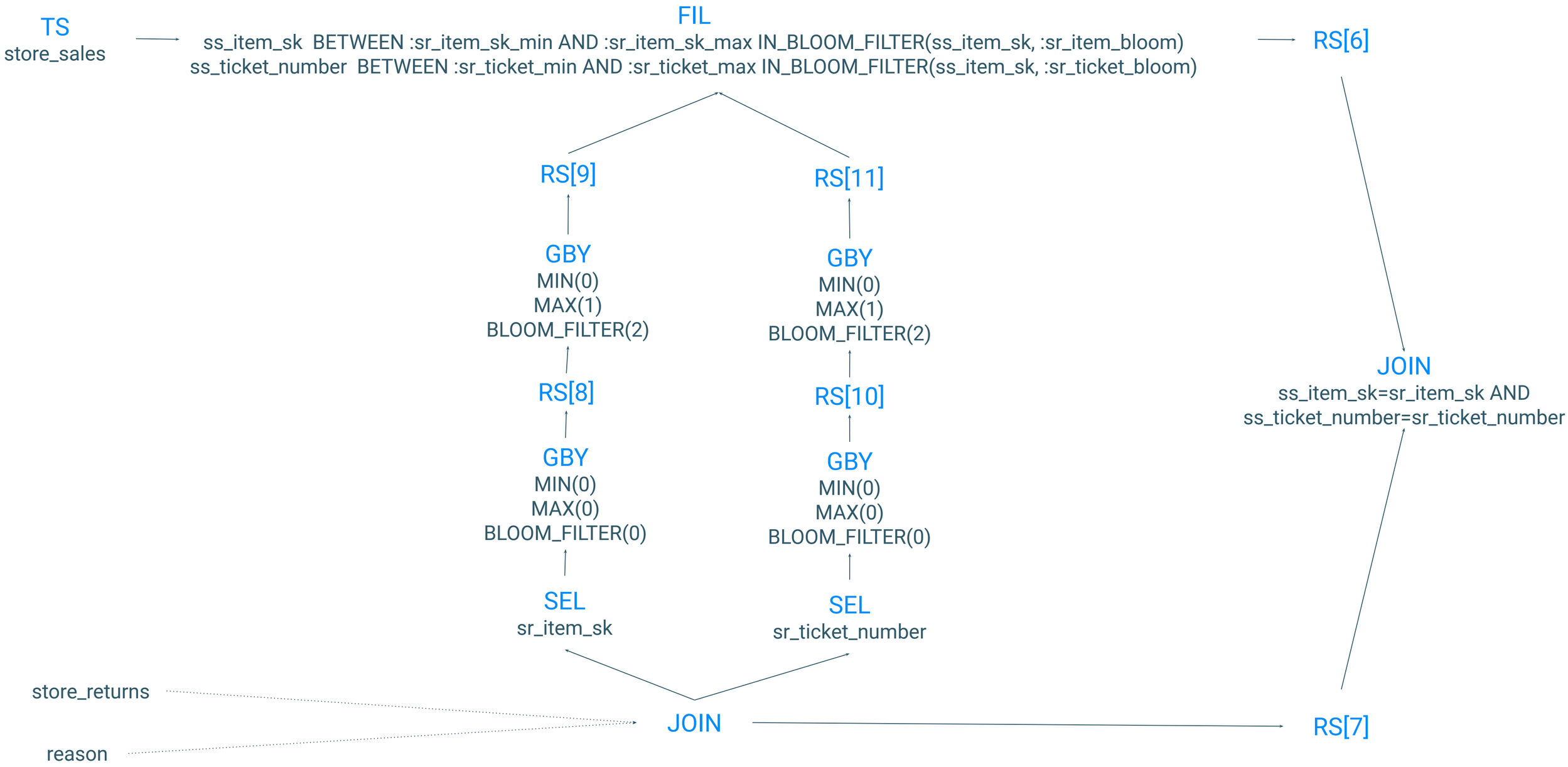
Four phase overview

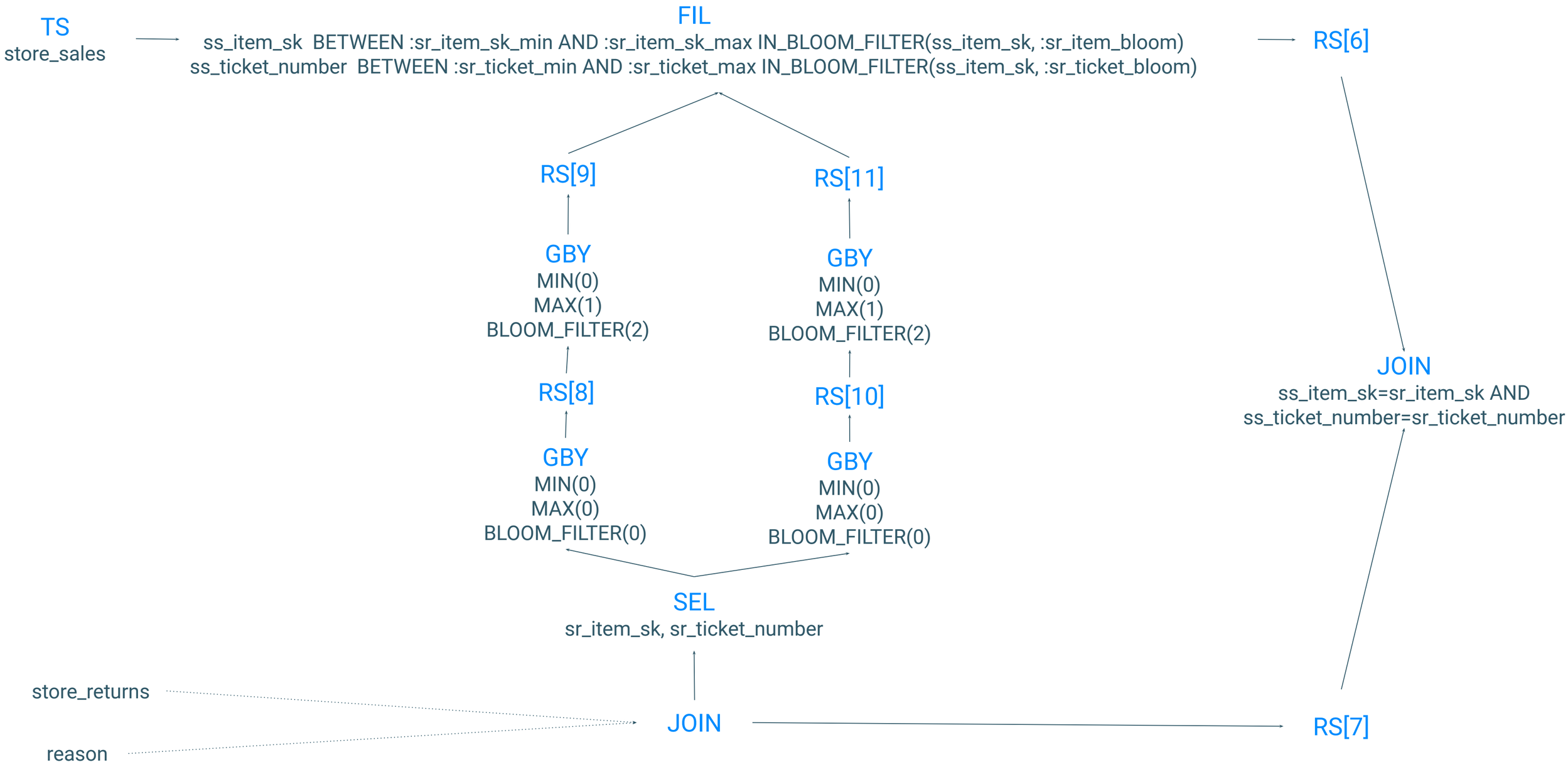
Create synthetic predicate

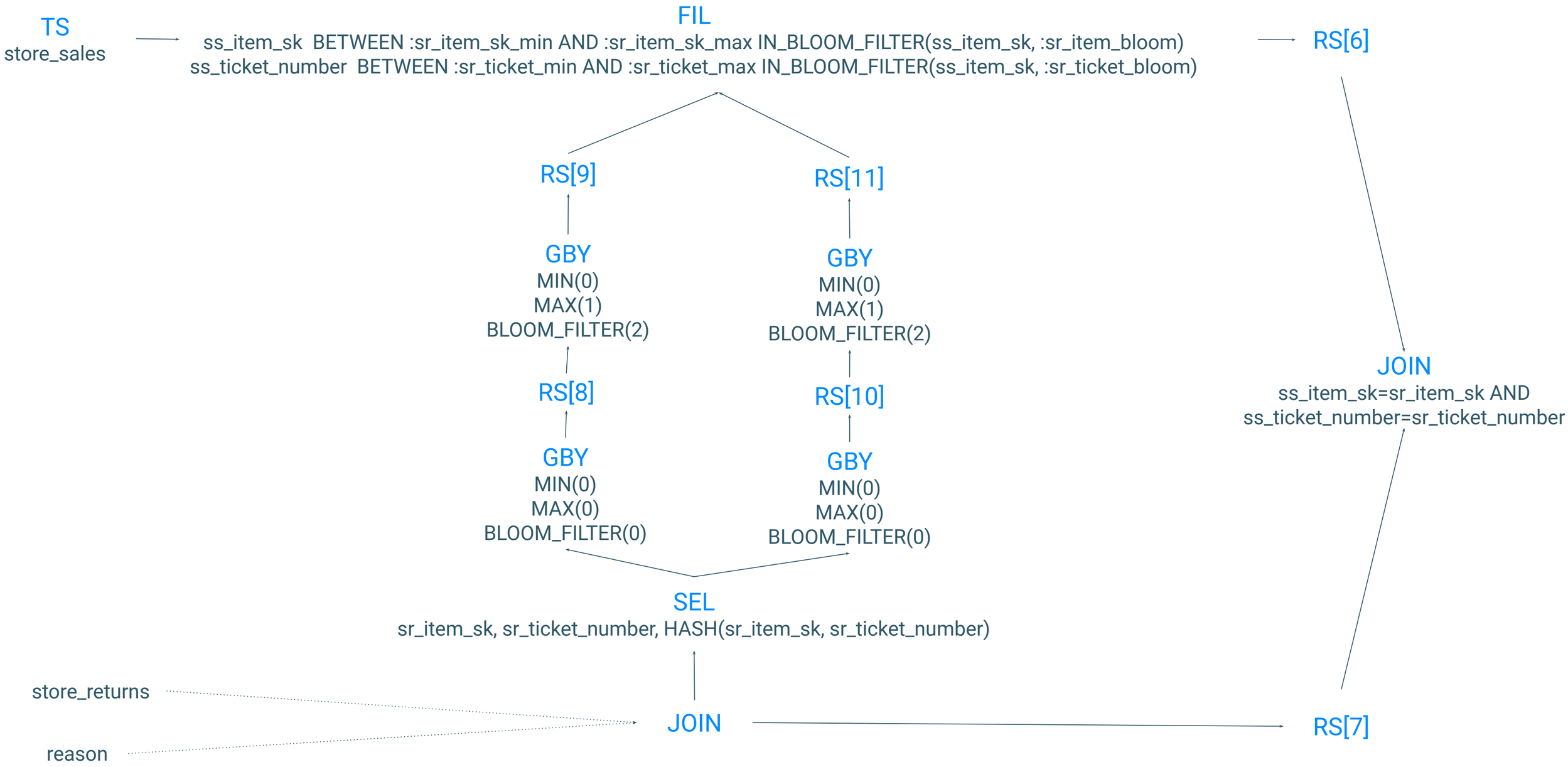
Expand to semijoin reducers

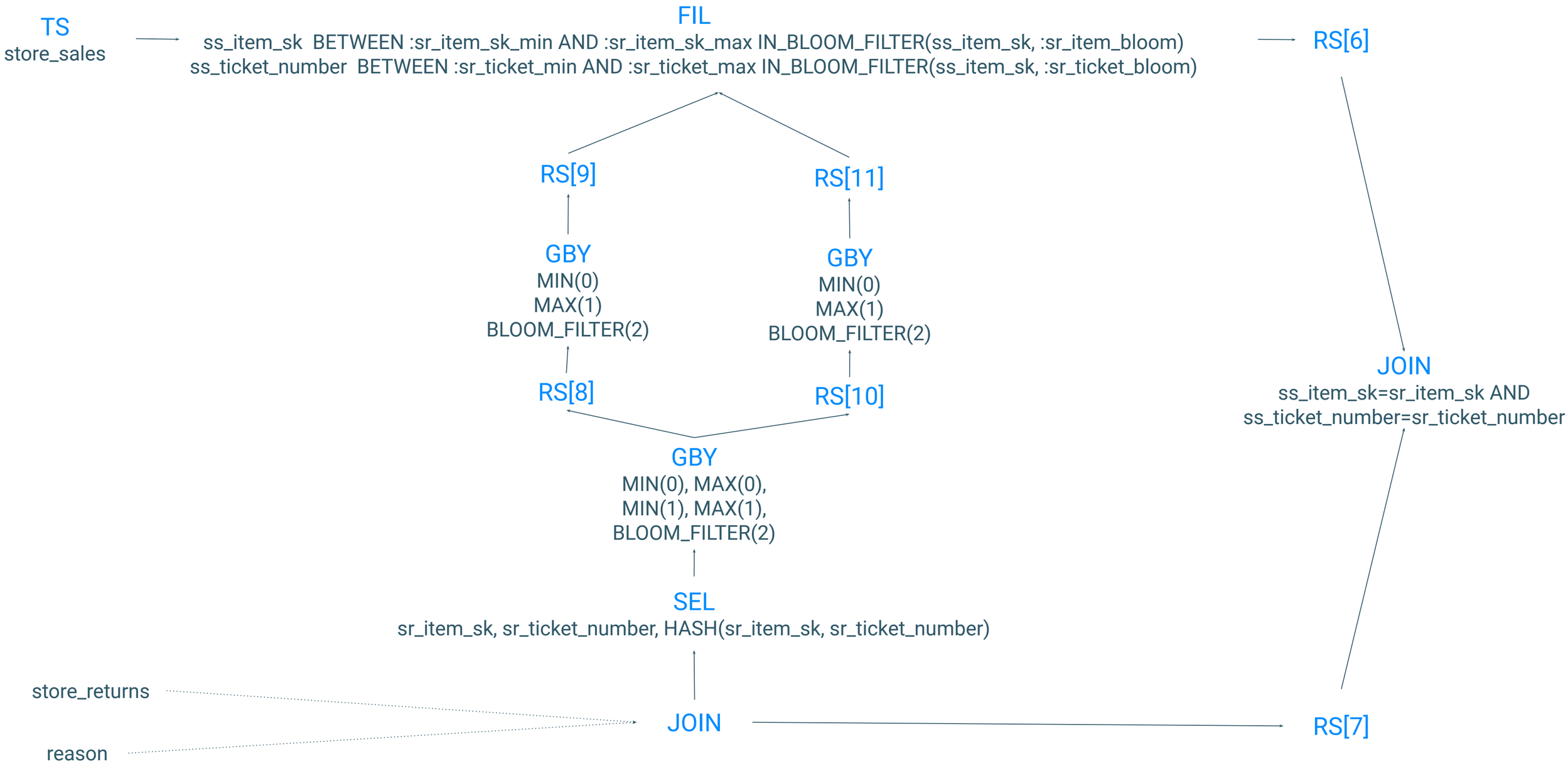
Merge to multi column semijoin reducers

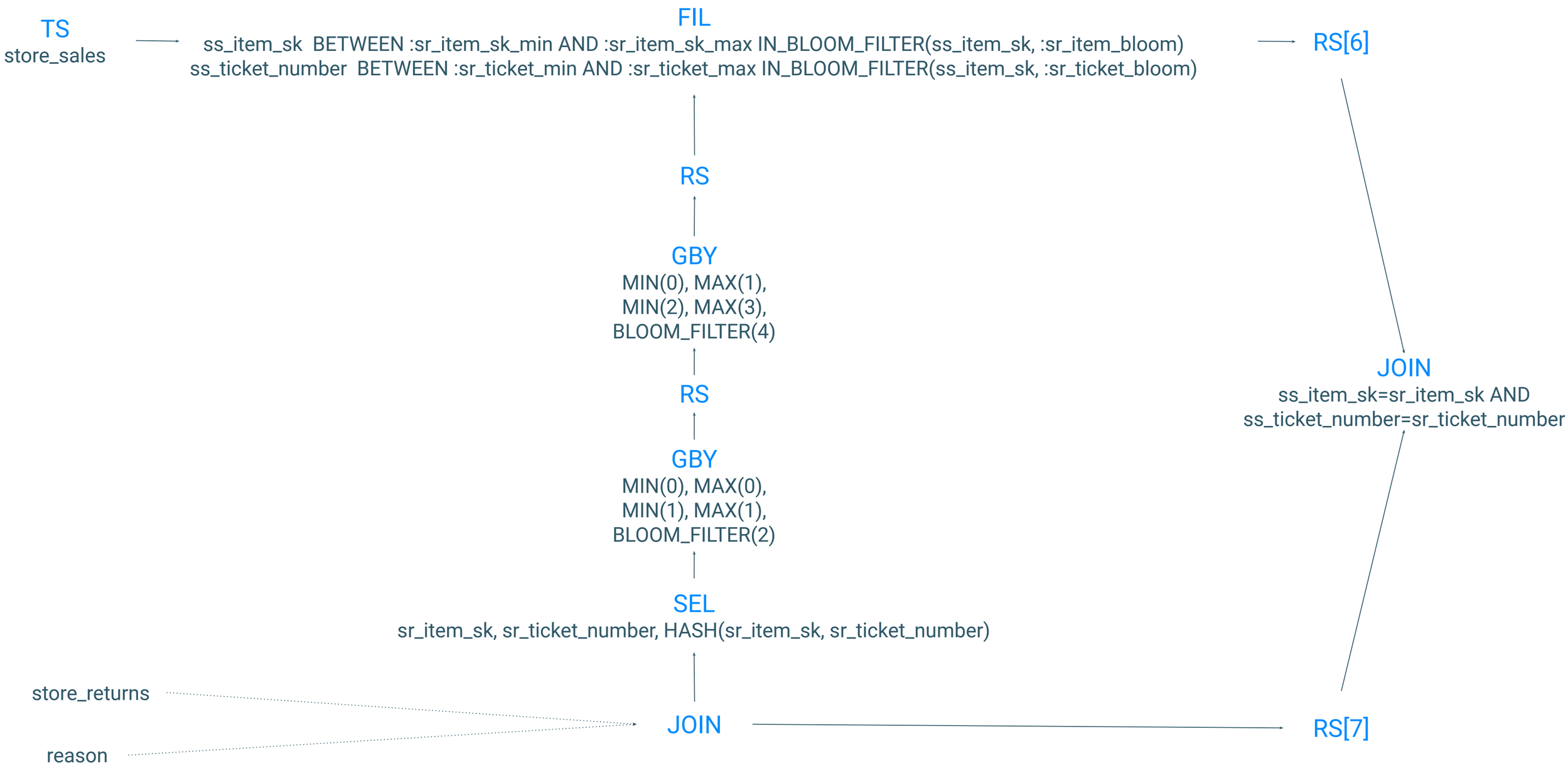
Conditionally remove semijoin reducers

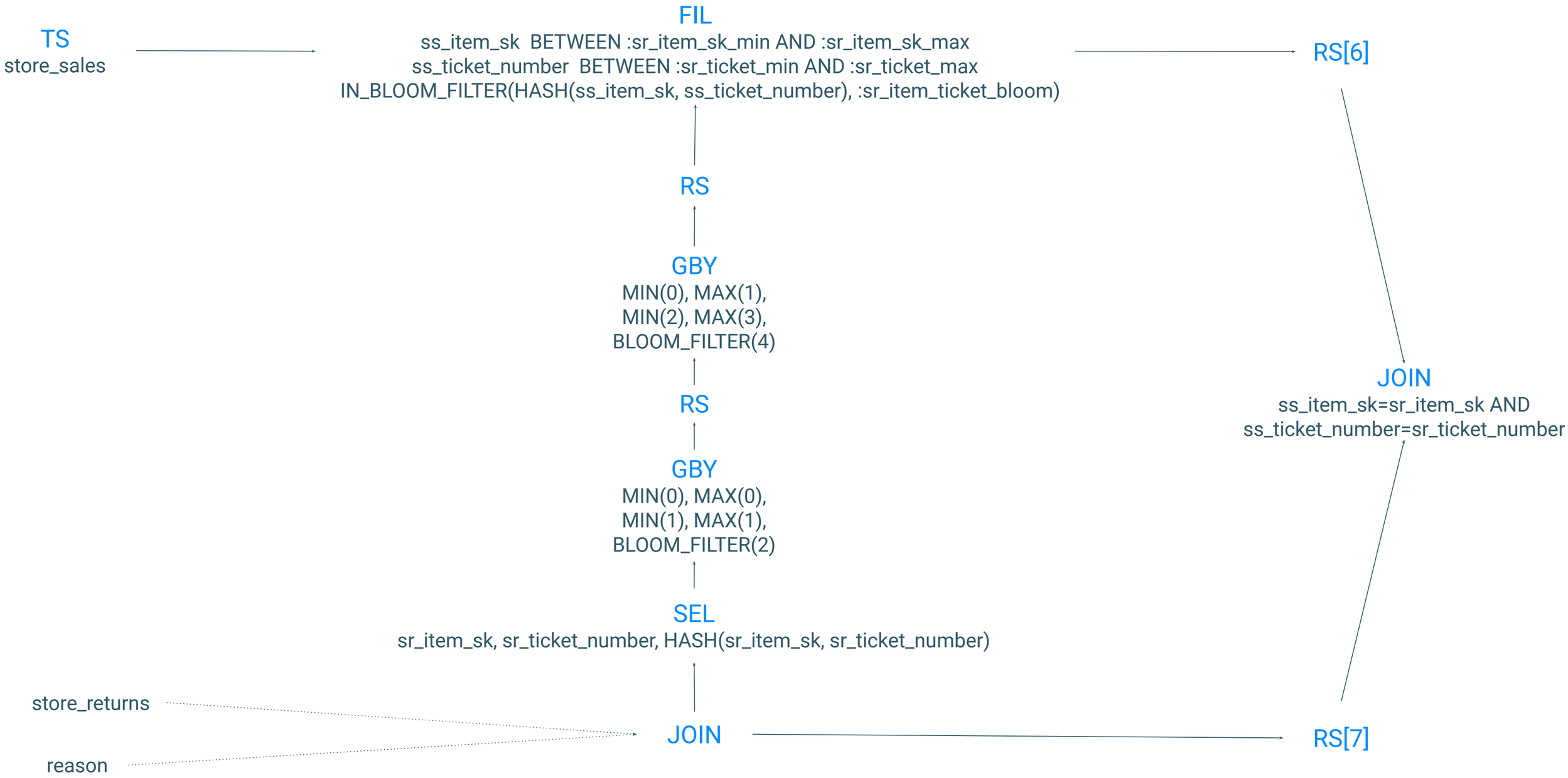












Four phase overview

Create synthetic predicate

Expand to semijoin reducers

Merge to multi column semijoin reducers

Conditionally remove semijoin reducers

Semijoin removal

Remove semijoin reduction:

- after union operator (correctness step)
- with missing statistics
- with parallel map join
- when precedes a sorted merge bucket (SMB) map join
- by benefit estimation

Code lies mostly inside TezCompiler

Lazy materialization

Common names

Row-level filtering

Lazy decoding

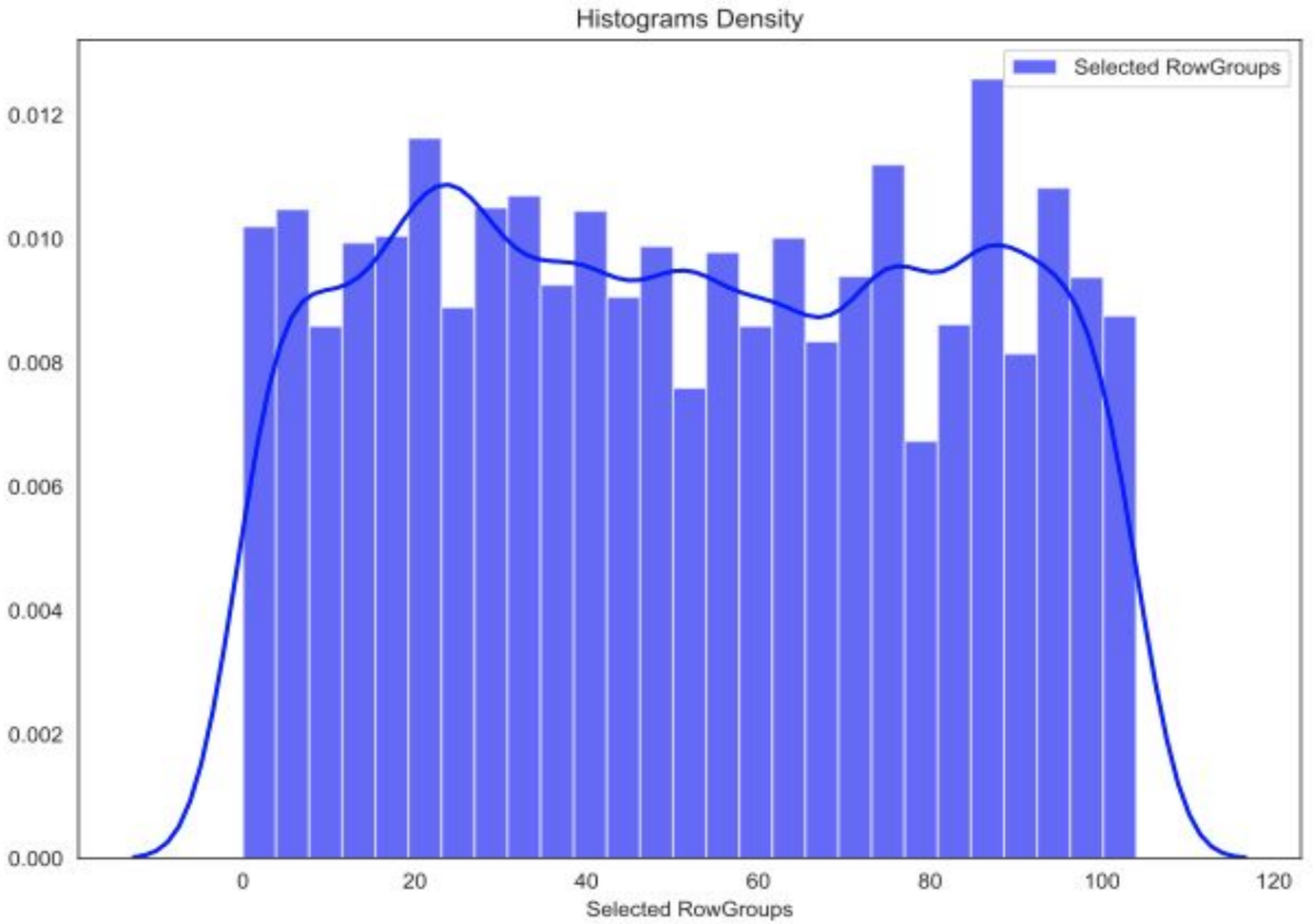
Probe-decode

Benefit

Avoid materializing data, not needed for the evaluation of the remaining of a query

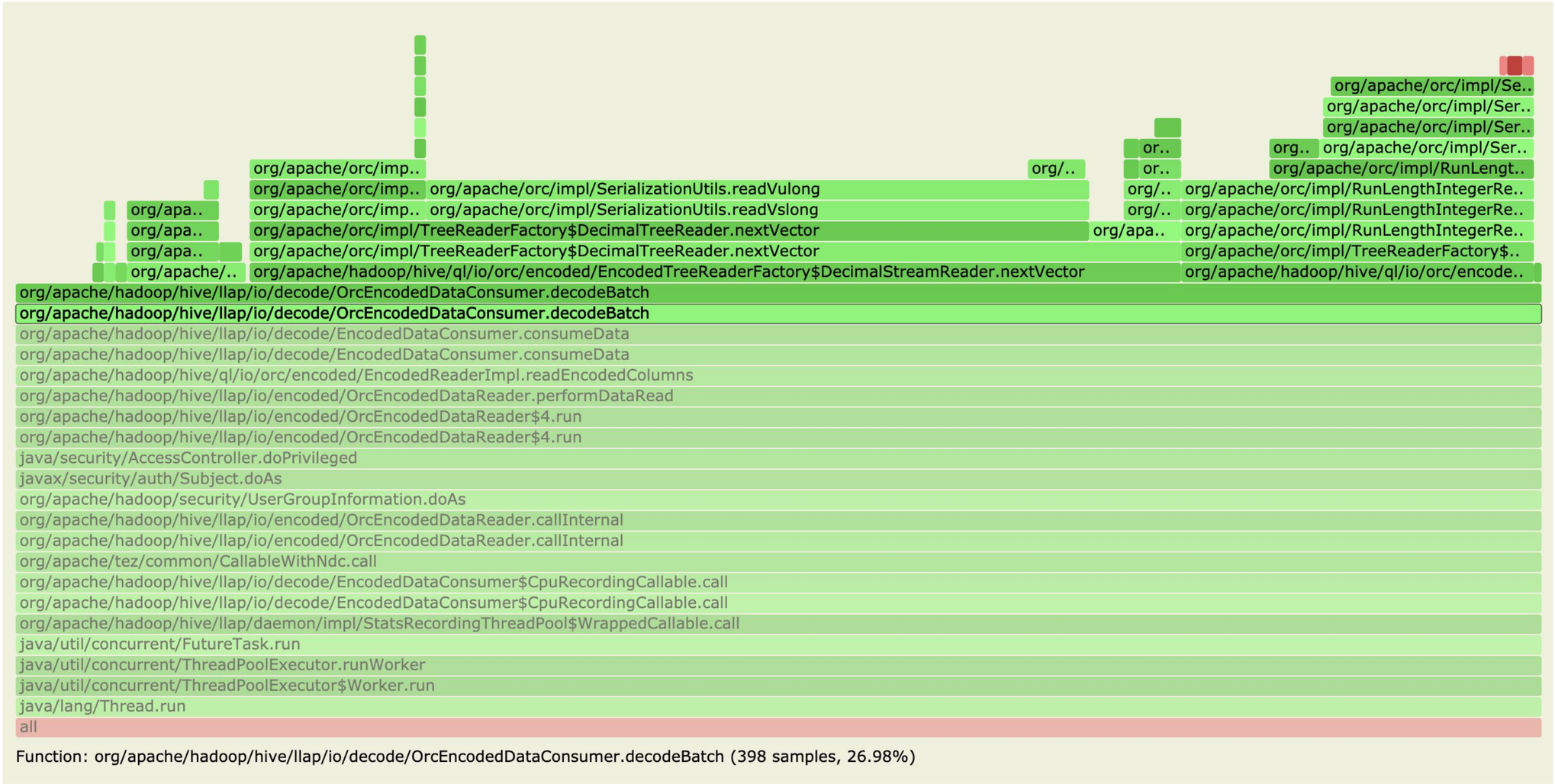
- ➔ Save CPU cycles
- ➔ Reduce memory allocations
- ➔ Eliminate data and filter operators from the pipeline

ORC row group selection



- Data organized in stripes of row groups (row x 10K)
- A single matched key of a row group leads to Ks of decoded rows
- Even for low selectivity queries, most of the row-groups might be read and decoded

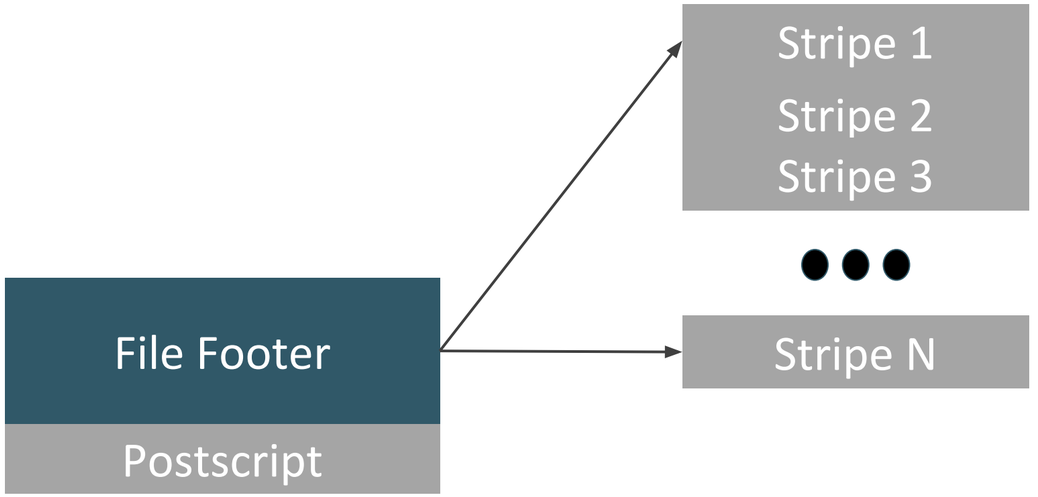
ORC row group decoding



ORC file structure

The **file footer** contains:

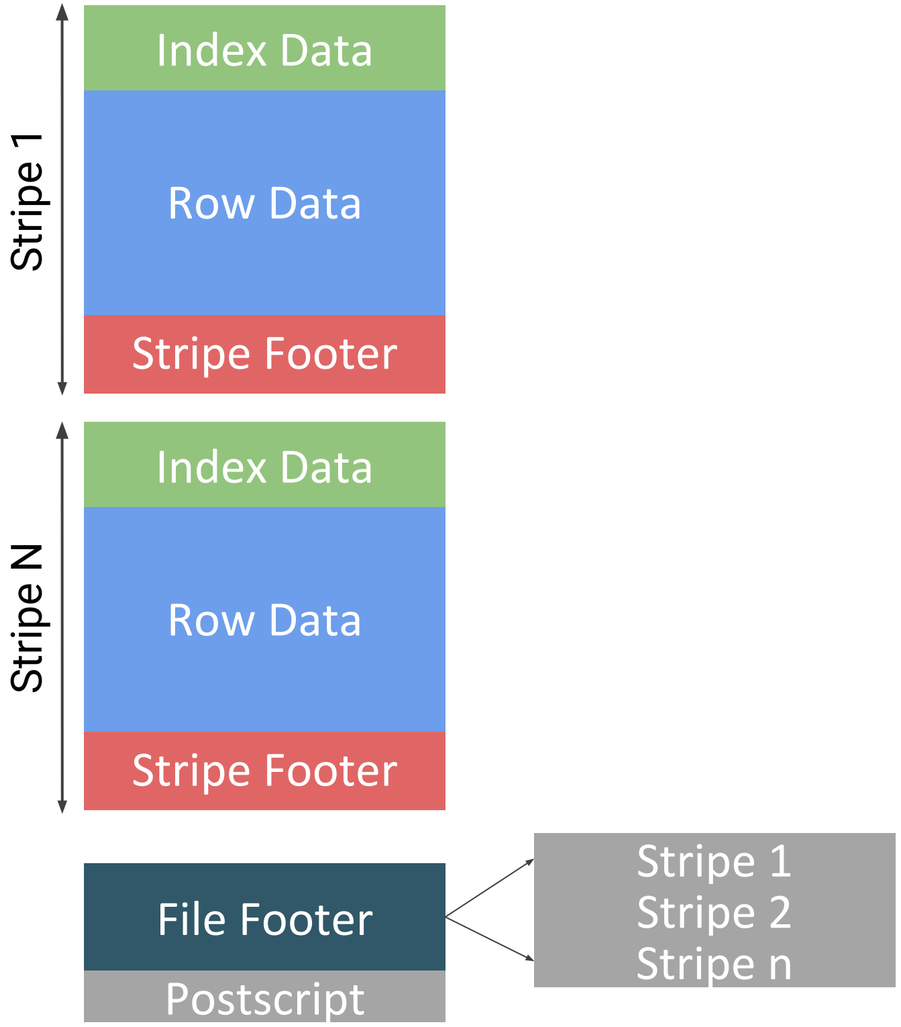
- Metadata
- Stripe information
- Postscript with compression details



ORC file data divided into **stripes**:

- Self contained sets of rows (organized by columns)
- Smallest unit of work for tasks
- ~64MB by default, often larger/smaller (16MB for Hive ACID deltas)

ORC file structure



Stripe footer:

- List of col streams and locations
- Column encoding

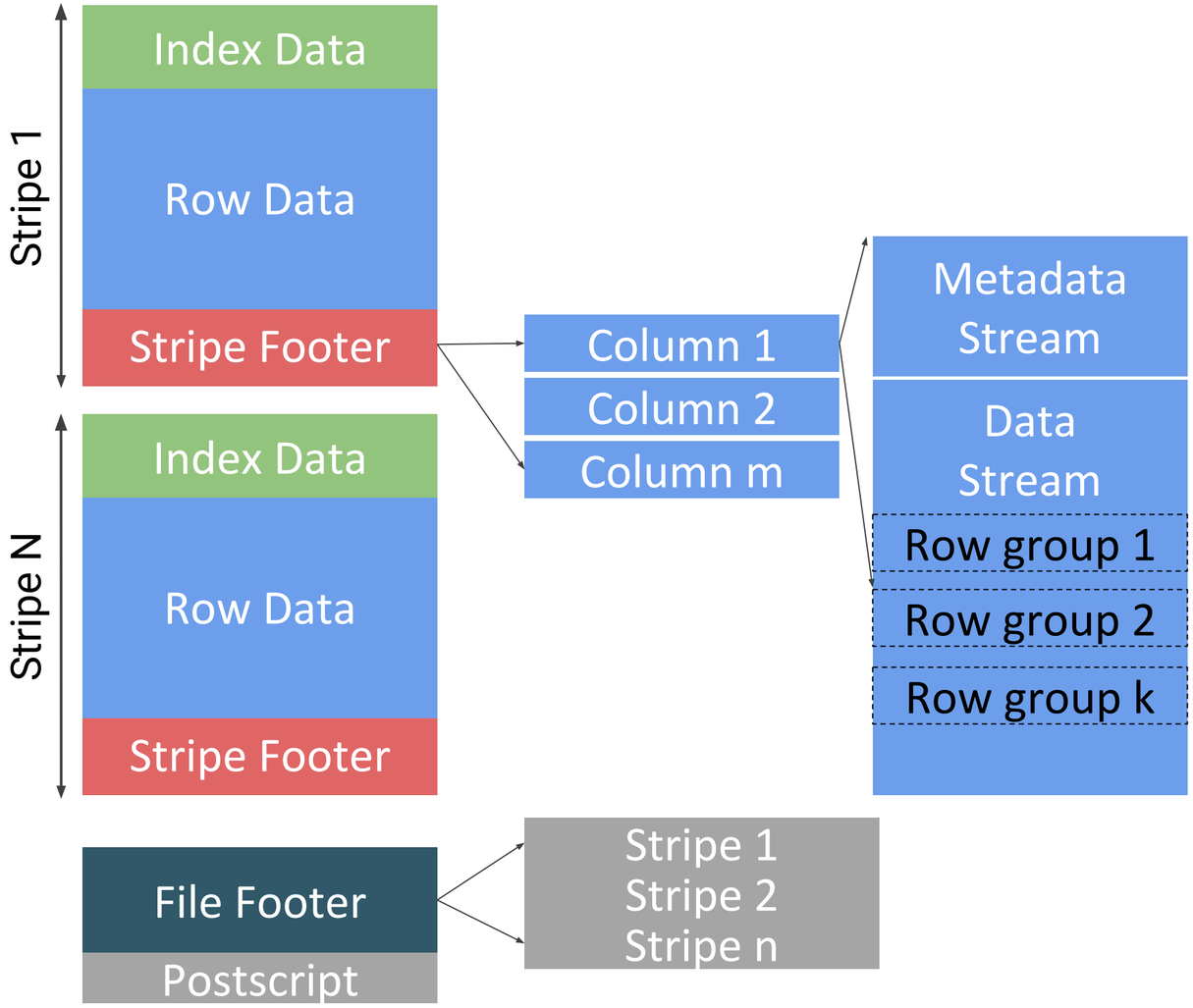
Index data:

- Offset to seek to right row group
- Stats (min, max, sum) per column
- Orc.row.index.stride 10,000

Row data:

- Index stream
- Metadata stream
- Data stream

ORC file structure



Stripe footer:

- List of col streams and locations
- Column encoding

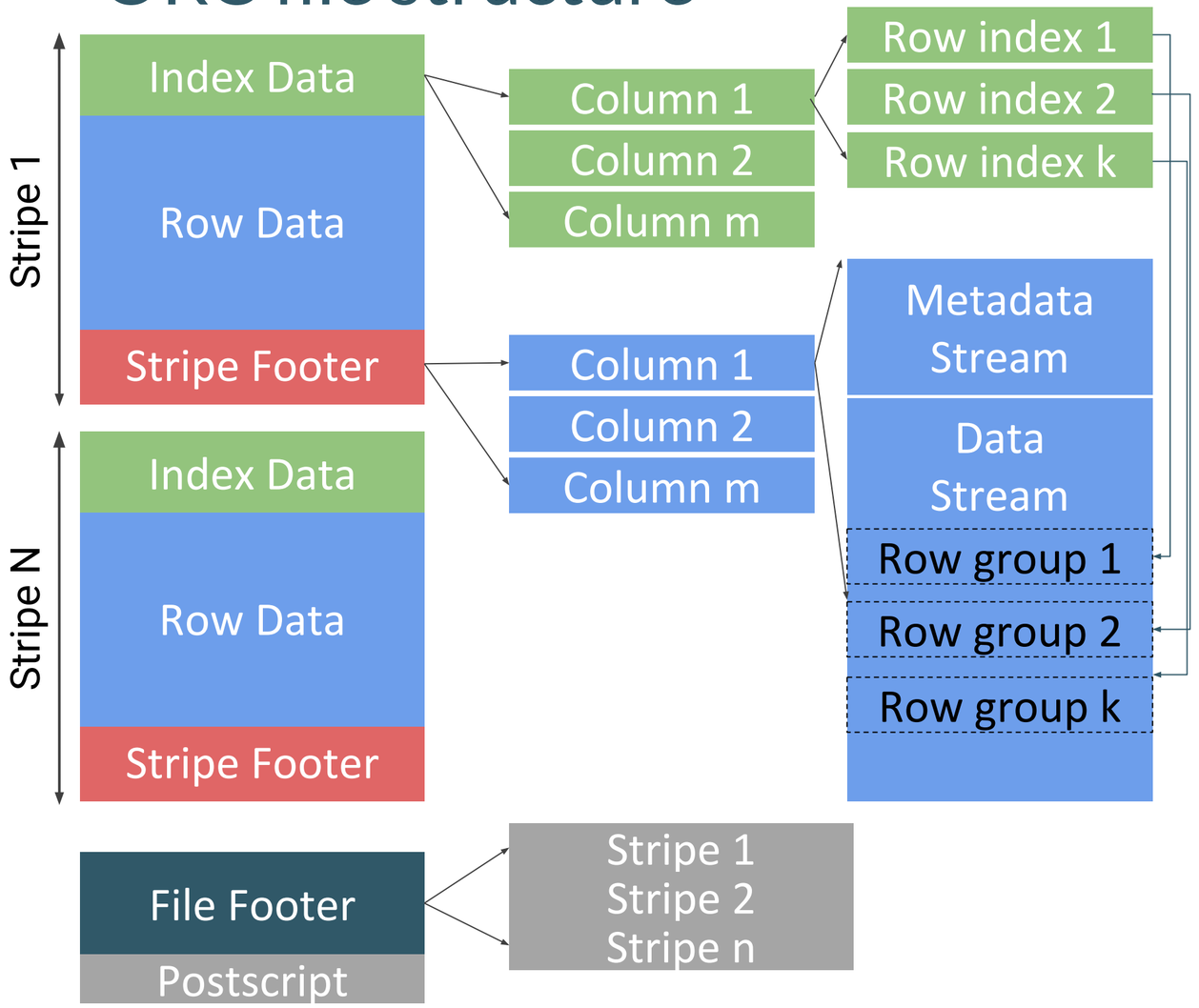
Index data:

- Offset to seek to right row group
- Stats (min, max, sum) per column
- Orc.row.index.stride 10,000

Row data:

- Index stream
- Metadata stream
- Data stream

ORC file structure



Stripe footer:

- List of col streams and locations
- Column encoding

Index data:

- Offset to seek to right row group
- Stats (min, max, sum) per column
- Orc.row.index.stride 10,000

Row data:

- Index stream
- Metadata stream
- Data stream

ORC search arguments

Hive can pass Search Arguments (**SArgs**) filter to the Reader:

- Limited set of operations (=, <=>, <, <=, in, between, is null)
- Compare one column to literal(s) e.g., “age > 100”

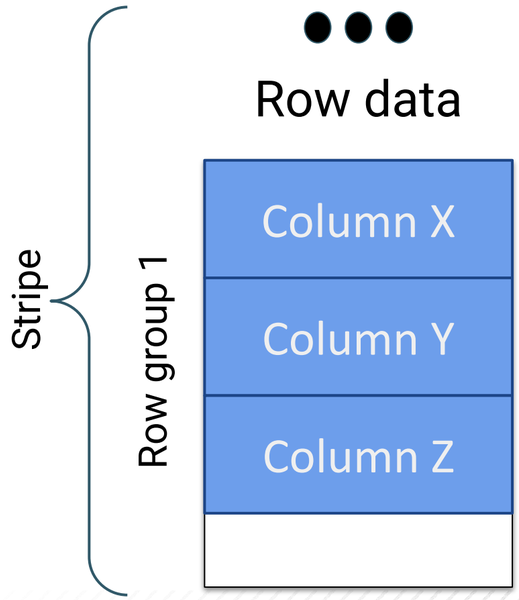
Limitations:

- Can’t support filters with: inequalities, UDFs, or columns with relationships
- Can only eliminate entire row groups, stripes, or files

```
SELECT x,y,z FROM foo WHERE x <> 1;
```

ORC row-level filtering

SArgs filter RGs ONLY if they can guarantee that none of the rows can pass the filter.

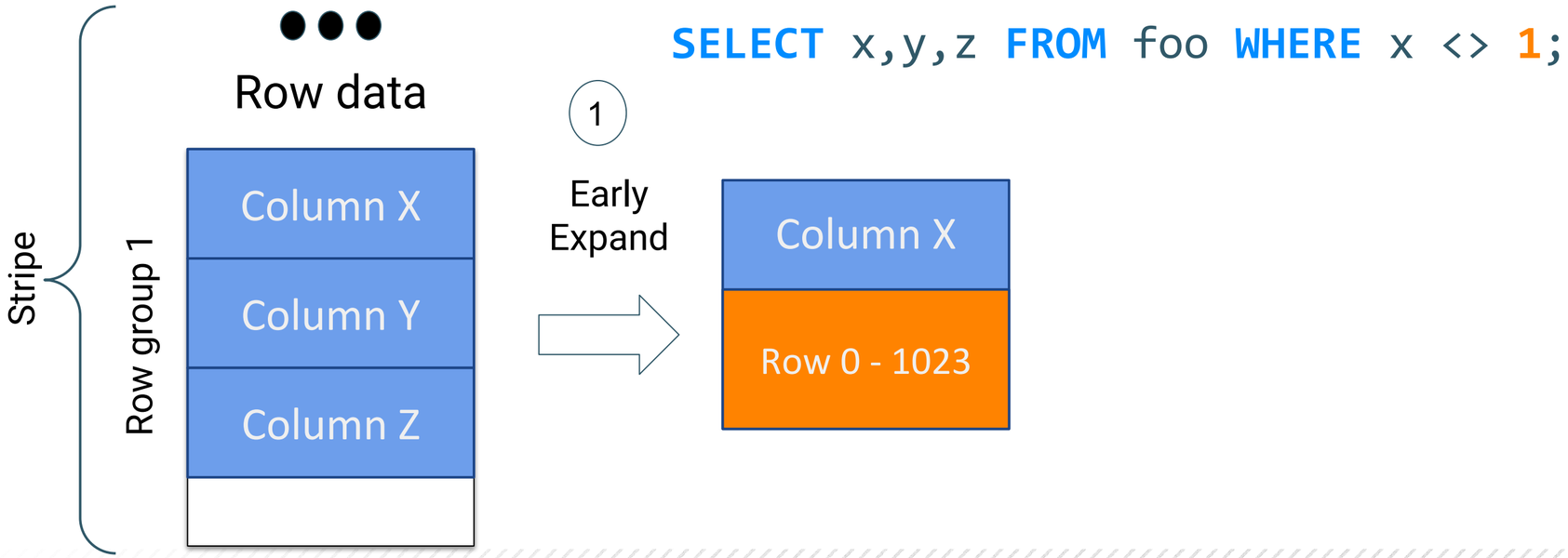



```
SELECT x,y,z FROM foo WHERE x <> 1;
```

ORC row-level filtering

SArgs filter RGs ONLY if they can guarantee that none of the rows can pass the filter. [ORC-577] Finer grained row-level filtering!

Read a subset of the columns and apply more detailed row level filter.

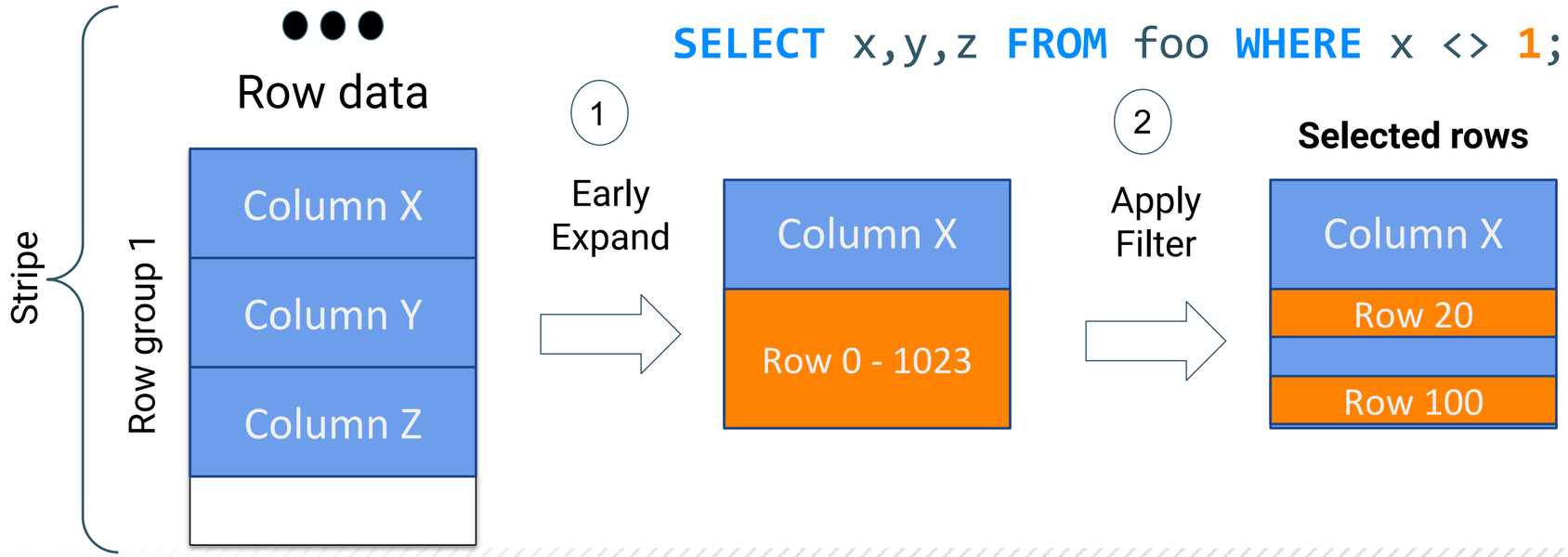



```
SELECT x,y,z FROM foo WHERE x <> 1;
```

ORC row-level filtering

SArgs filter RGs ONLY if they can guarantee that none of the rows can pass the filter. [ORC-577] Finer grained row-level filtering!

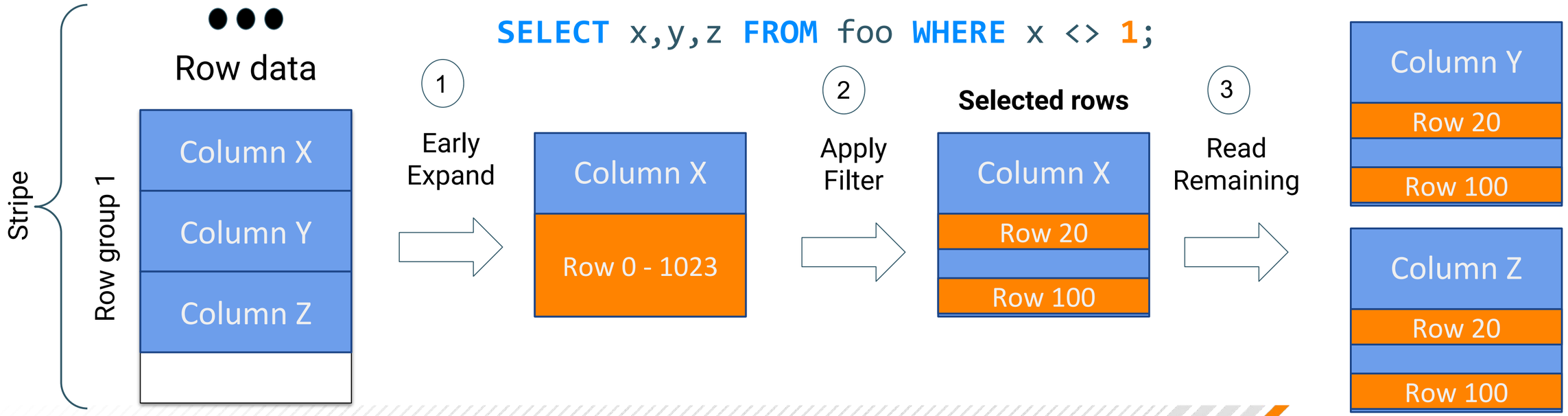
Read a subset of the columns and apply more detailed row level filter.



ORC row-level filtering

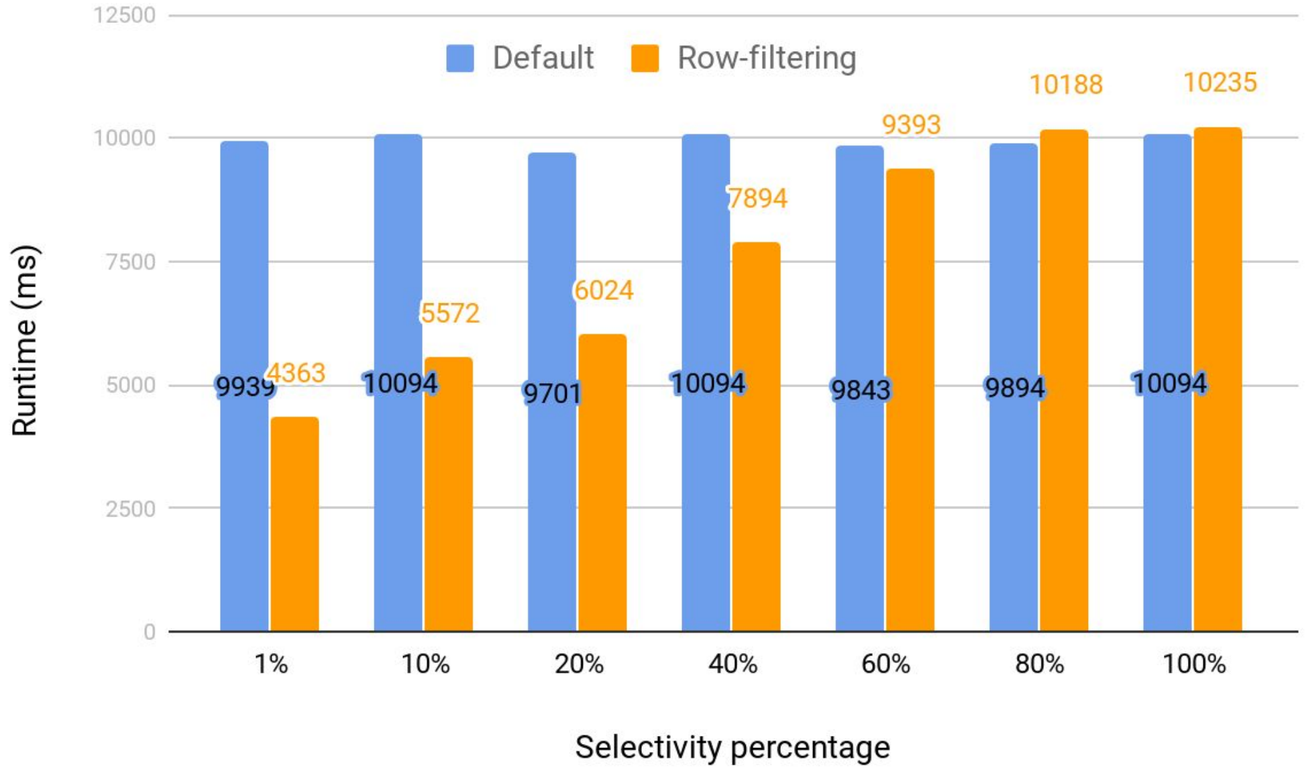
SArgs filter RGs ONLY if they can guarantee that none of the rows can pass the filter. [ORC-577] Finer grained row-level filtering!

Read a subset of the columns and apply more detailed row level filter.



ORC row-level filtering evaluation

[ORC-597] benchmark ORC Reader for complex types such as Decimals, Doubles and real datasets (Taxi, Sales, GitHub)



ORC row-level filtering implementation

[HIVE-22959] introduce FilterContext as part of VectorRowBatch

[ORC-577] extend ORC Reader with a new option:

```
setRowFilter(String[] columnNames,Consumer<VectorizedRowBatch> filter)
```

Consumer callback to implement any kind of filtering logic

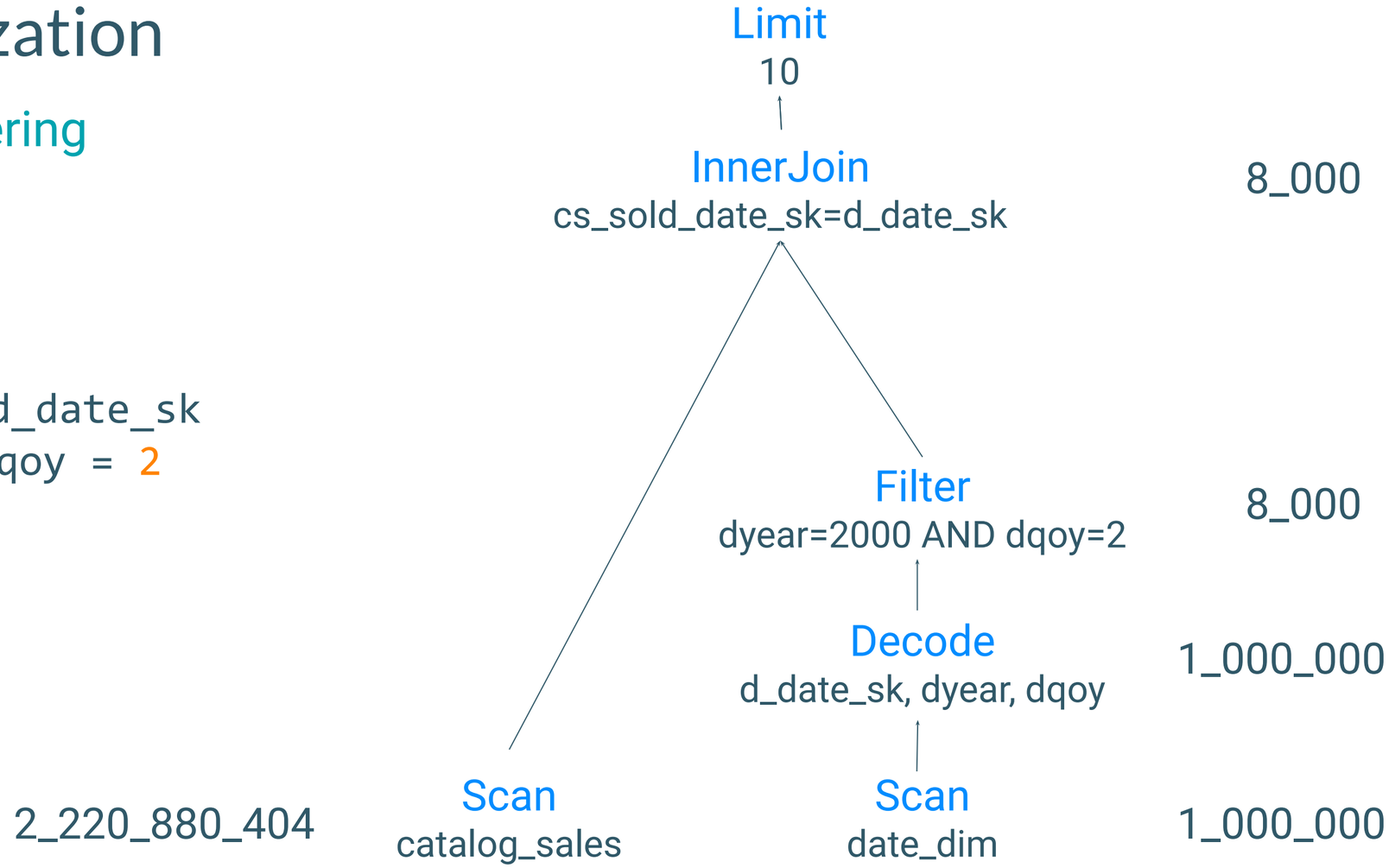
New ORC reader logic:

- Early expand the given column(s)
- Evaluate the filter callback (implemented on the consumer side)
- Decode only the selected rows from the remaining columns

Probe-decode optimization

Leveraging ORC row-level filtering

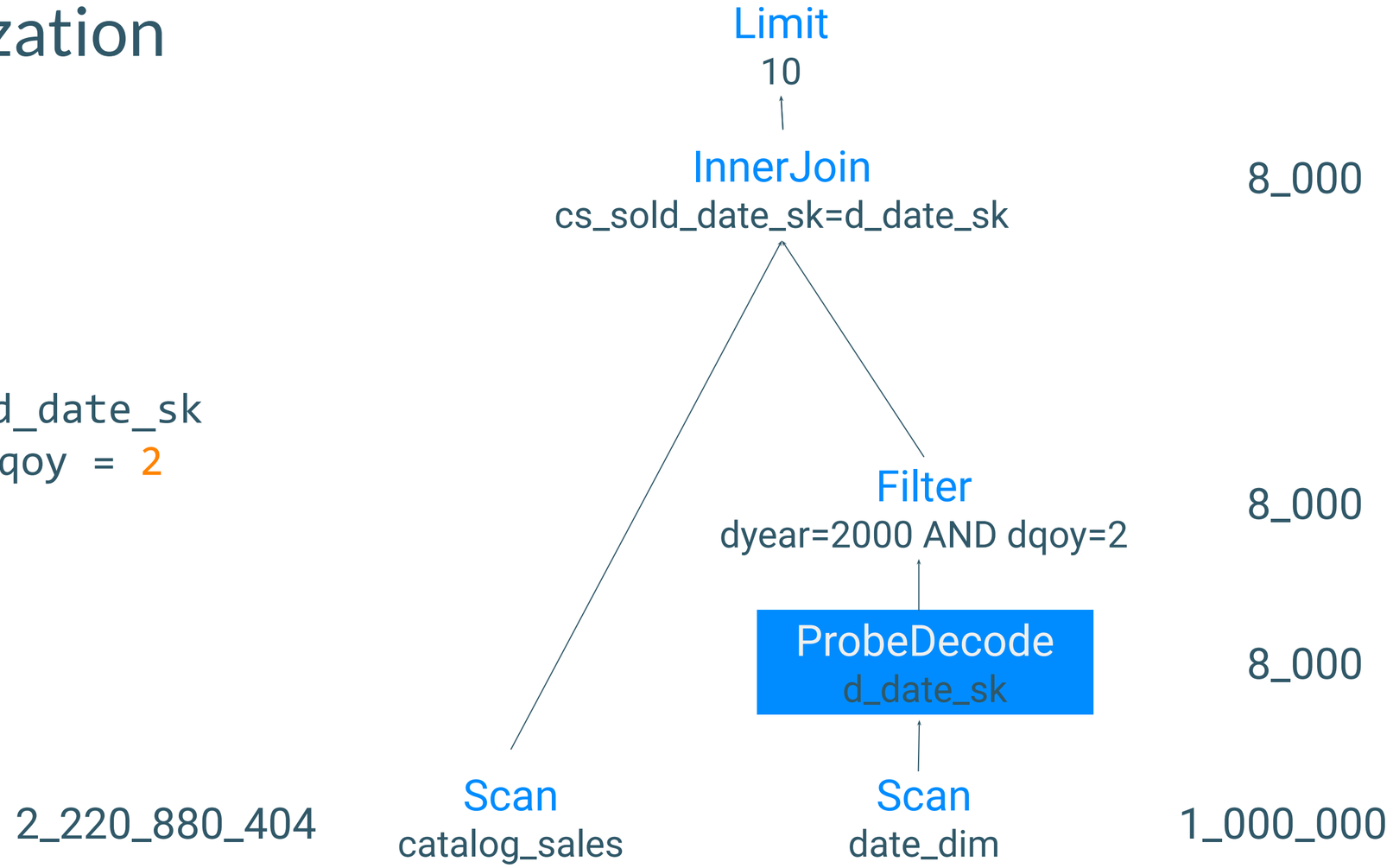
```
SELECT *
FROM catalog_sales
INNER JOIN date_dim
     ON cs_sold_date_sk = d_date_sk
WHERE d_year = 2000 and d_qoy = 2
LIMIT 10
```



Probe-decode optimization

Static row-level filtering

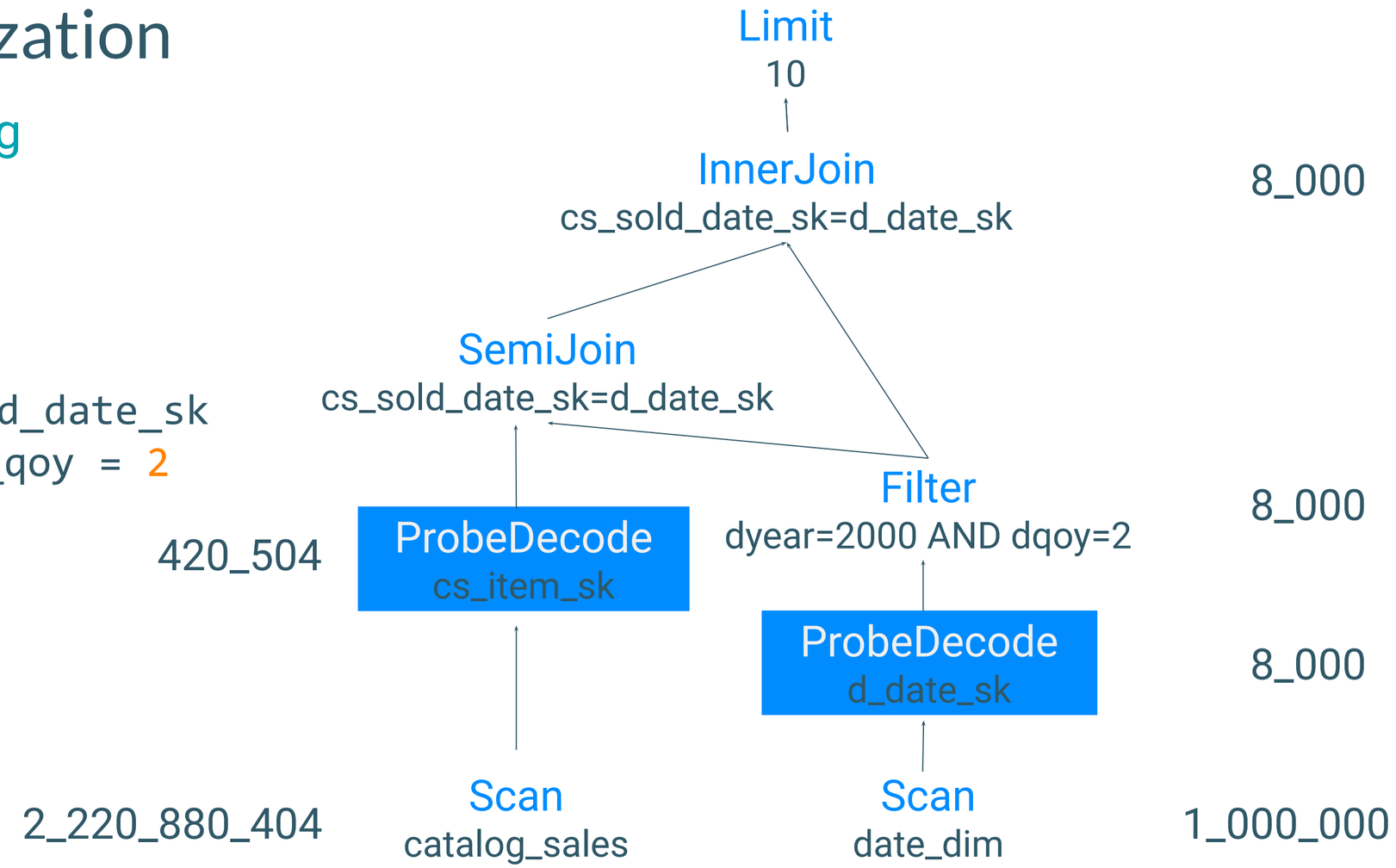
```
SELECT *
FROM catalog_sales
INNER JOIN date_dim
     ON cs_sold_date_sk = d_date_sk
WHERE d_year = 2000 and d_qoy = 2
LIMIT 10
```



Probe-decode optimization

Dynamic MJ row-level filtering

```
SELECT *
FROM catalog_sales
INNER JOIN date_dim
     ON cs_sold_date_sk = d_date_sk
WHERE d_year = 2000 and d_qoy = 2
LIMIT 10
```

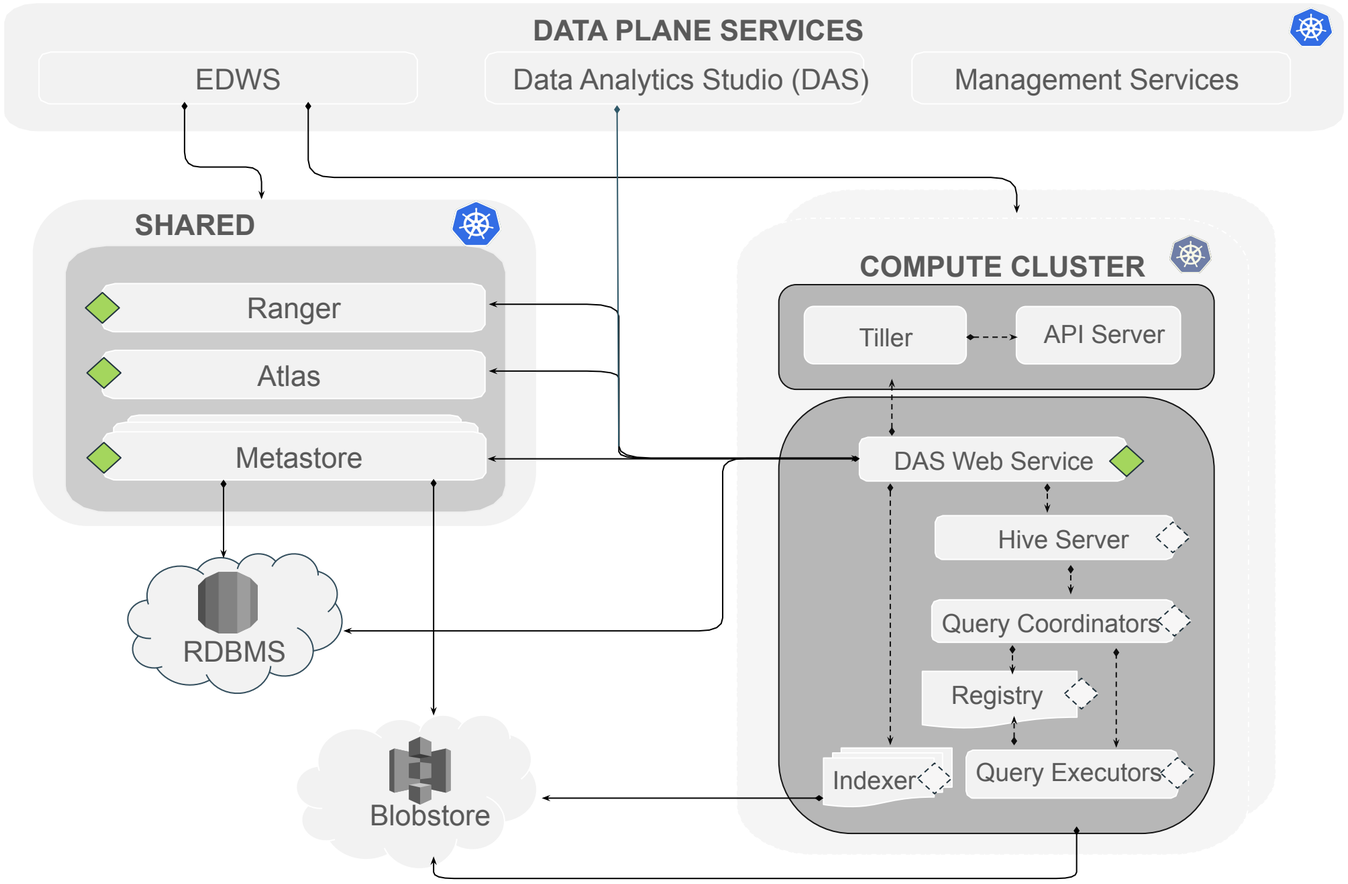


Probe-decode plan

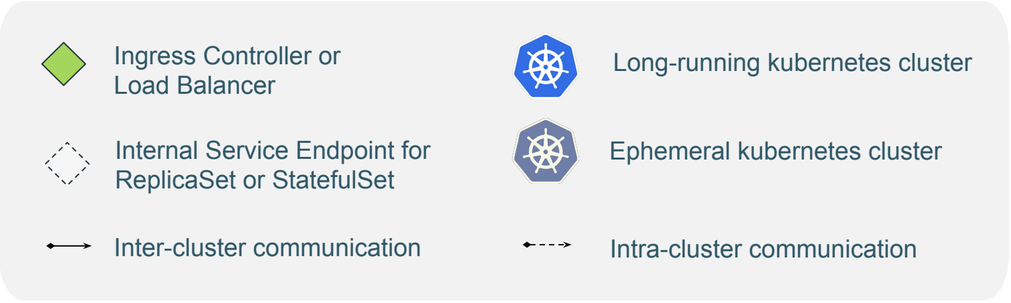
```
Map 1 <- Map 4 (BROADCAST_EDGE), Map 5 (BROADCAST_EDGE)
Reducer 2 <- Map 1 (SIMPLE_EDGE)
Reducer 3 <- Reducer 2 (SIMPLE_EDGE)
```

```
SET hive.optimize.scan.probedecode=true
```

```
        Vertices:
         Map 1
            Map Operator Tree:
                TableScan
                   alias: store_sales
                   probeDecodeDetails: cacheKey:HASH_MAP_MAPJOIN_48_container, bigKeyColName:ss_item_sk, smallTablePos:1, keyRatio:1.6
                   Statistics: Num rows: 82510879939 Data size: 10343396725952 Basic stats: COMPLETE Column stats: COMPLETE
                   TableScan Vectorization:
                       native: true
                    Select Operator
                     expressions: ss_item_sk (type: bigint), ss_ext_sales_price (type: decimal(7,2)), ss_sold_date_sk (type: bigint)
                     outputColumnNames: _col0, _col1, _col2
                     Select Vectorization:
                         className: VectorSelectOperator
                         native: true
                         projectedOutputColumnNums: [1, 14, 22]
                     Statistics: Num rows: 82510879939 Data size: 10343396725952 Basic stats: COMPLETE Column stats: COMPLETE
                     Map Join Operator
                       condition map:
                            Inner Join 0 to 1
                       keys:"
                         0 _col2 (type: bigint)
                         1 _col0 (type: bigint)
                       Map Join Vectorization:
                           bigTableKeyColumns: 22:bigint
                           className: VectorMapJoinInnerBigOnlyLongOperator
                         input vertices:
                                 1 Map 4
```



- CDP Public & Private
  - Hive/LLAP sidecar install
  - Tuned instances for specific hardware
  - Elastic compute resources
  - Multiple versions of Hive
  - Multiple warehouse & compute instances
  - Dynamic configuration and secrets management
  - Stateful and work preserving restarts (cache)
  - Rolling restart for upgrades. Fast rollback to previous good state.



Conclusions

Data pruning

- Prune early → increase query efficiency

Semijoin reduction

- Reduce disk & network I/O

Lazy materialization

- Save CPU cycles and memory allocations

THANK YOU

CLOUDERA

References

1. Stocker, Konrad, et al. "Integrating semi-join-reducers into state-of-the-art query processors." ICDE 2001

2. Abadi, Daniel J., Samuel R. Madden, and Nabil Hachem. "Column-stores vs. row-stores: how different are they really?." SIGMOD 2008

3. Lang, Harald, et al. "Performance-optimal filtering: Bloom overtakes cuckoo at high throughput." VLDB 2019