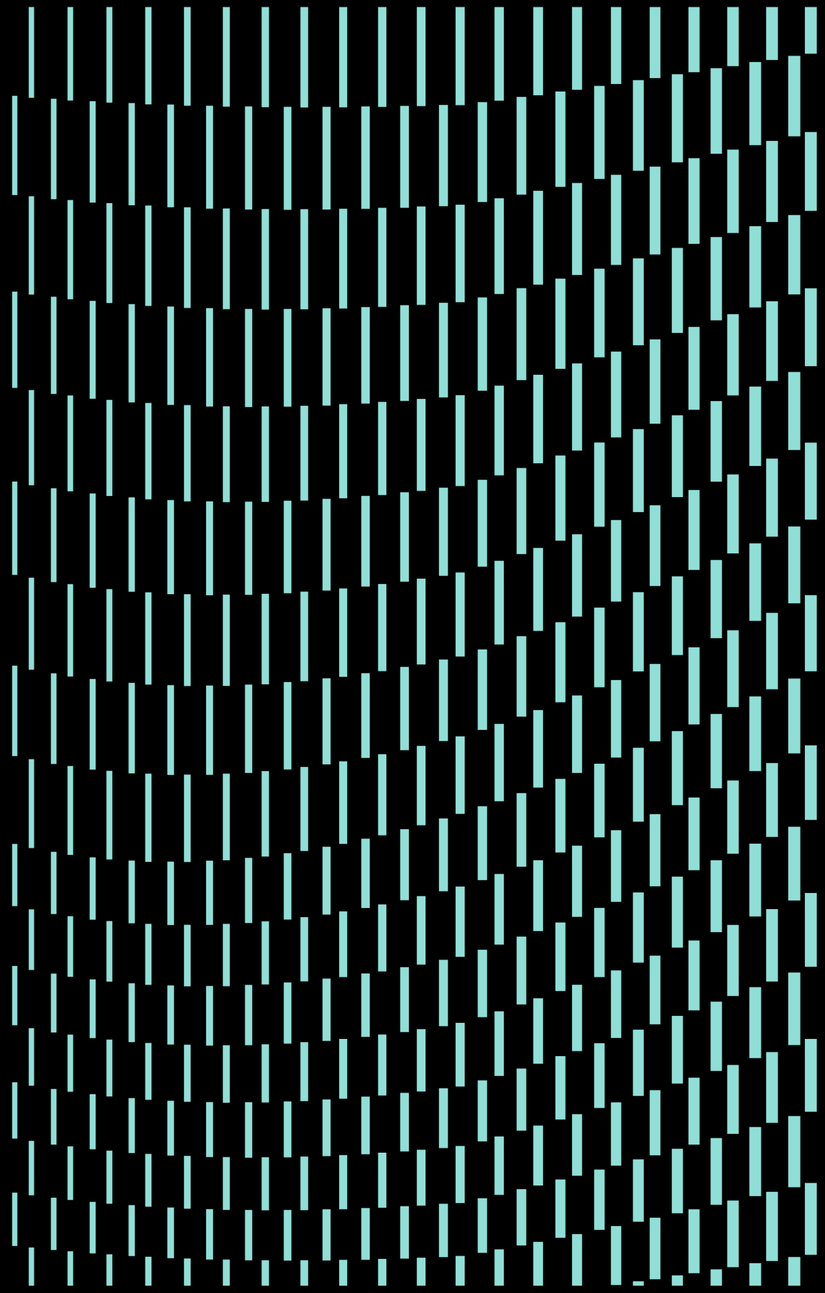
RND DATA
SUMMIT 19

Deep Dive into Spark

Eric Xiao, Data Developer | Storage and Query Technologies



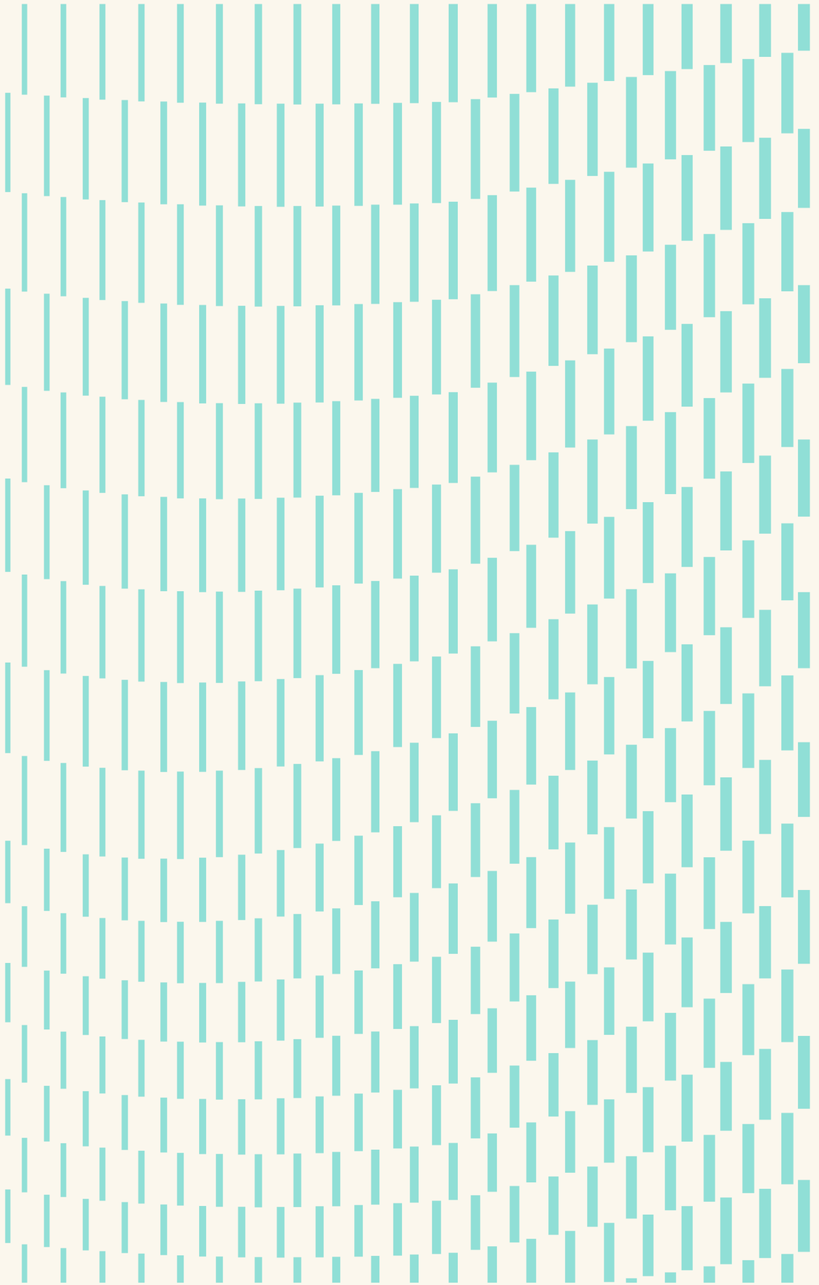
Motivation

We'll be learning how to analyze a spark application through the sparkUI.

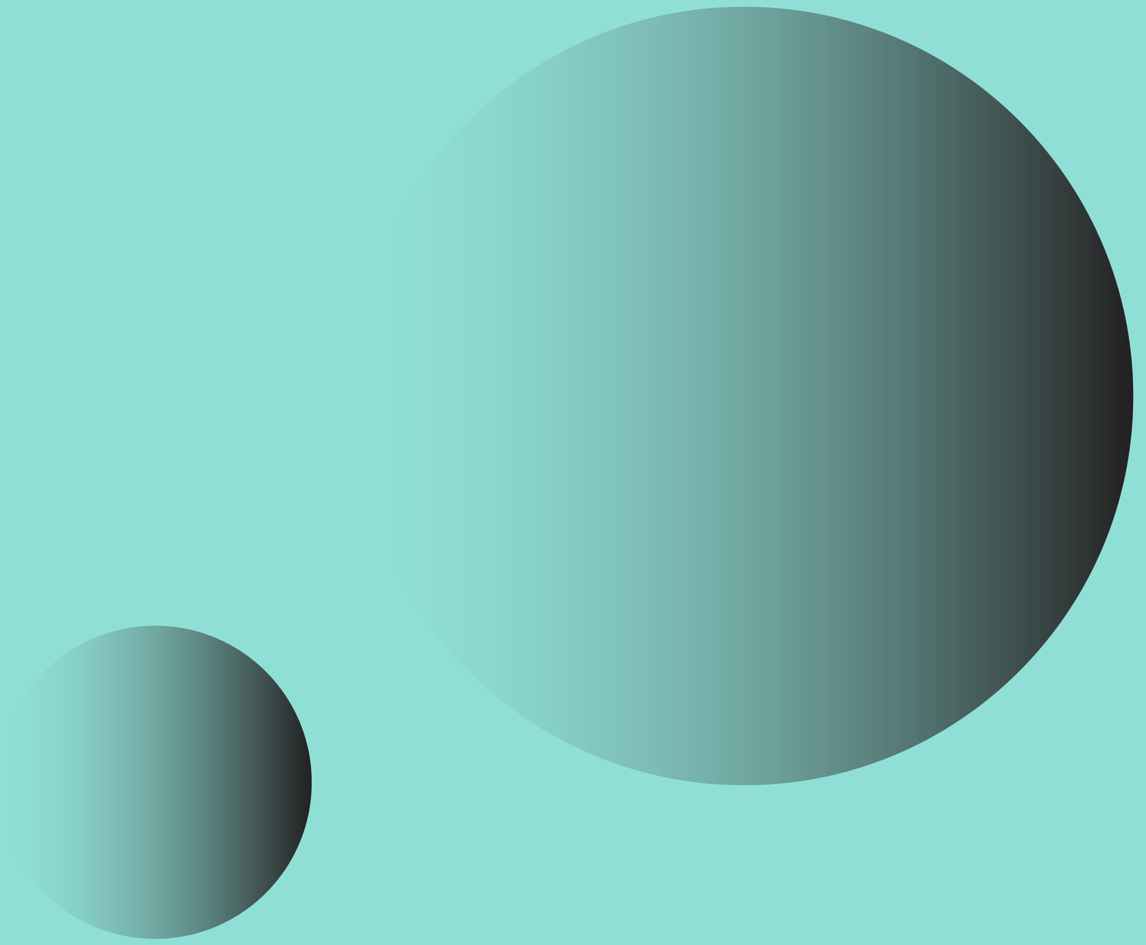
Then implement 2 optimizations that will address 2 main problems that are common in spark applications.

Overview:

1. Concrete Example Query.
2. From Code to Execution.
3. Optimization 1: Improving joins.
4. Optimization 2: Eliminate data spill.
5. Recap.
6. Questions.



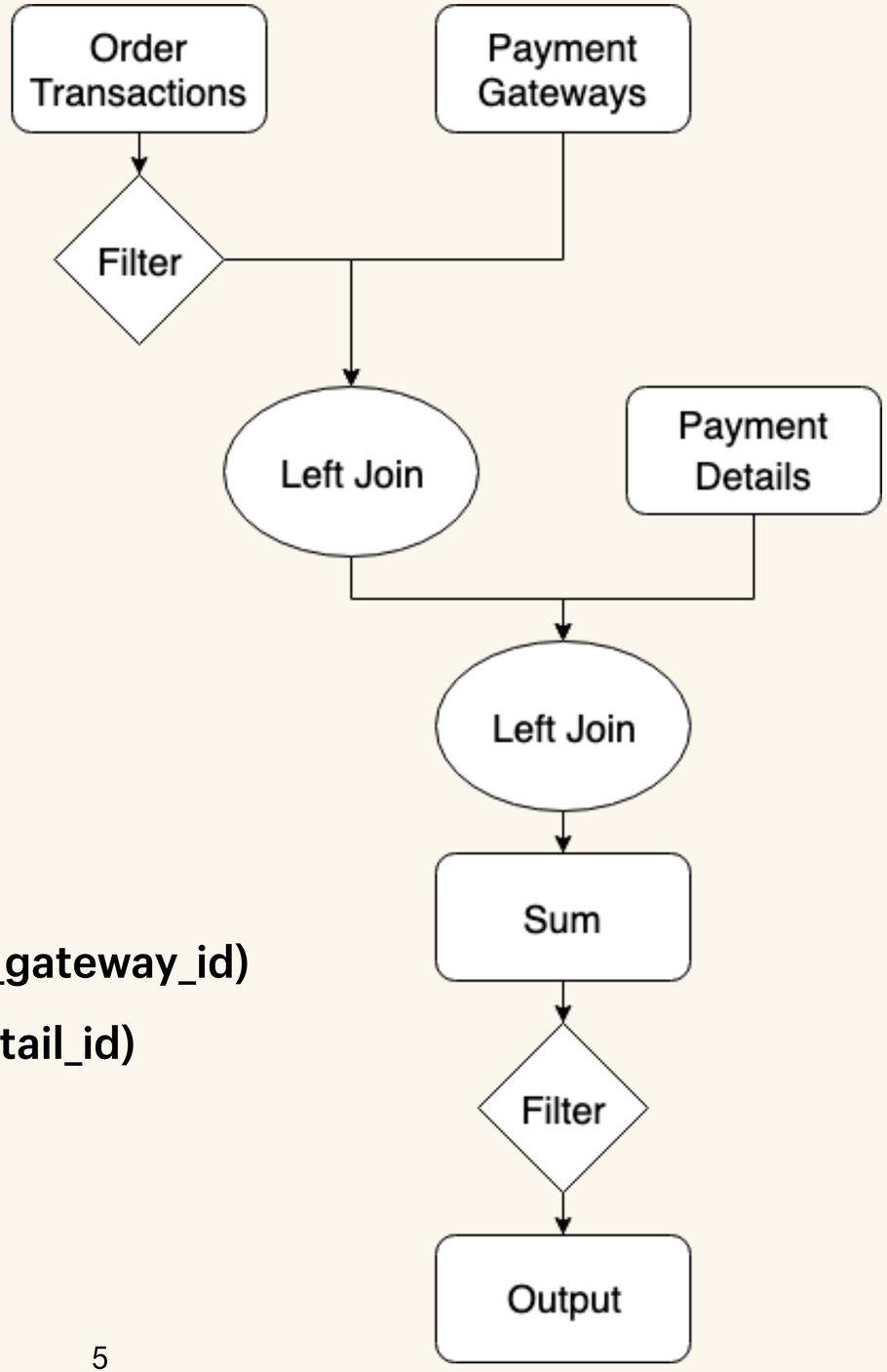
Section 1: Concrete Query Example



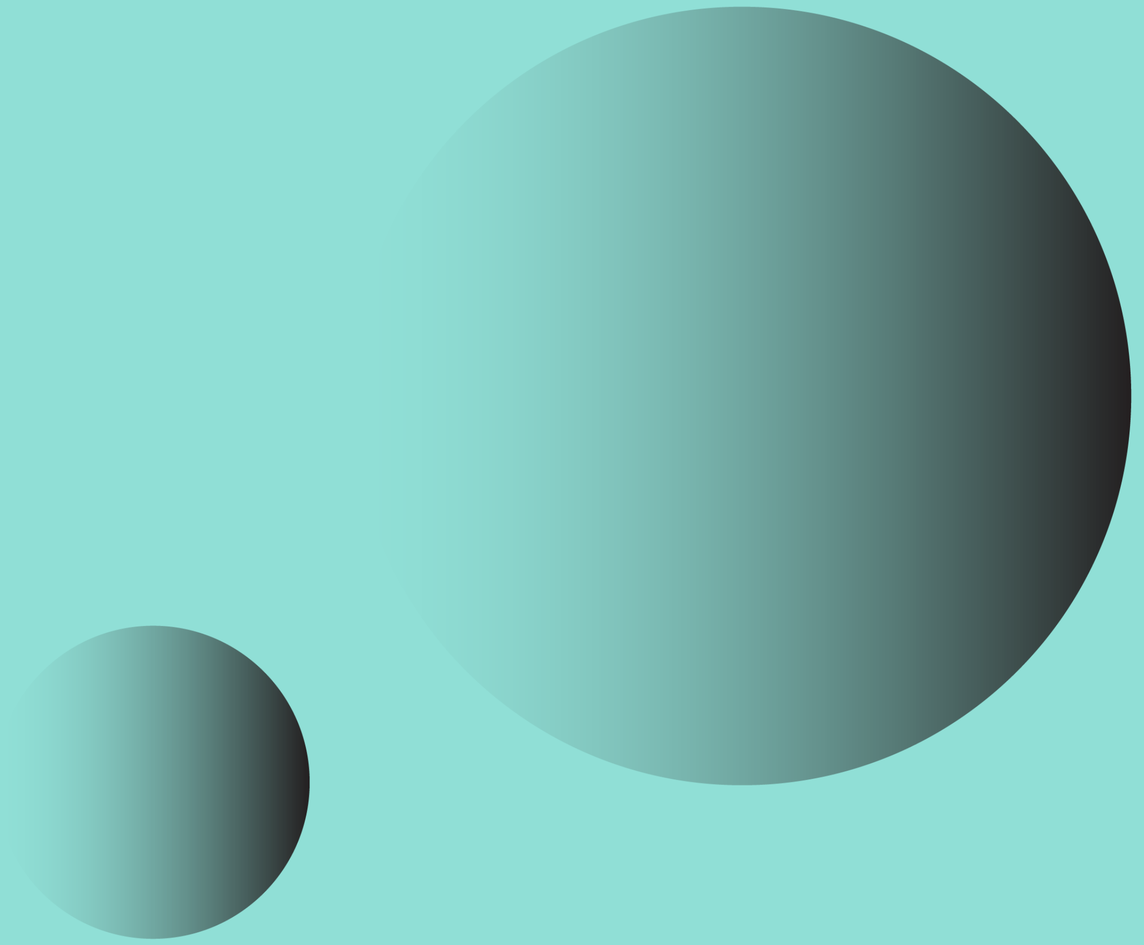
Example Query

I want to know the count of order transactions per credit card types.

```
SELECT
  card_type,
  count(*)
FROM order_transactions
LEFT JOIN payment_gateways USING (payment_gateway_id)
LEFT JOIN payment_details USING (payment_detail_id)
GROUP BY 1
WHERE ....
```



Section 2: From Code to Execution



Lazy Evaluation

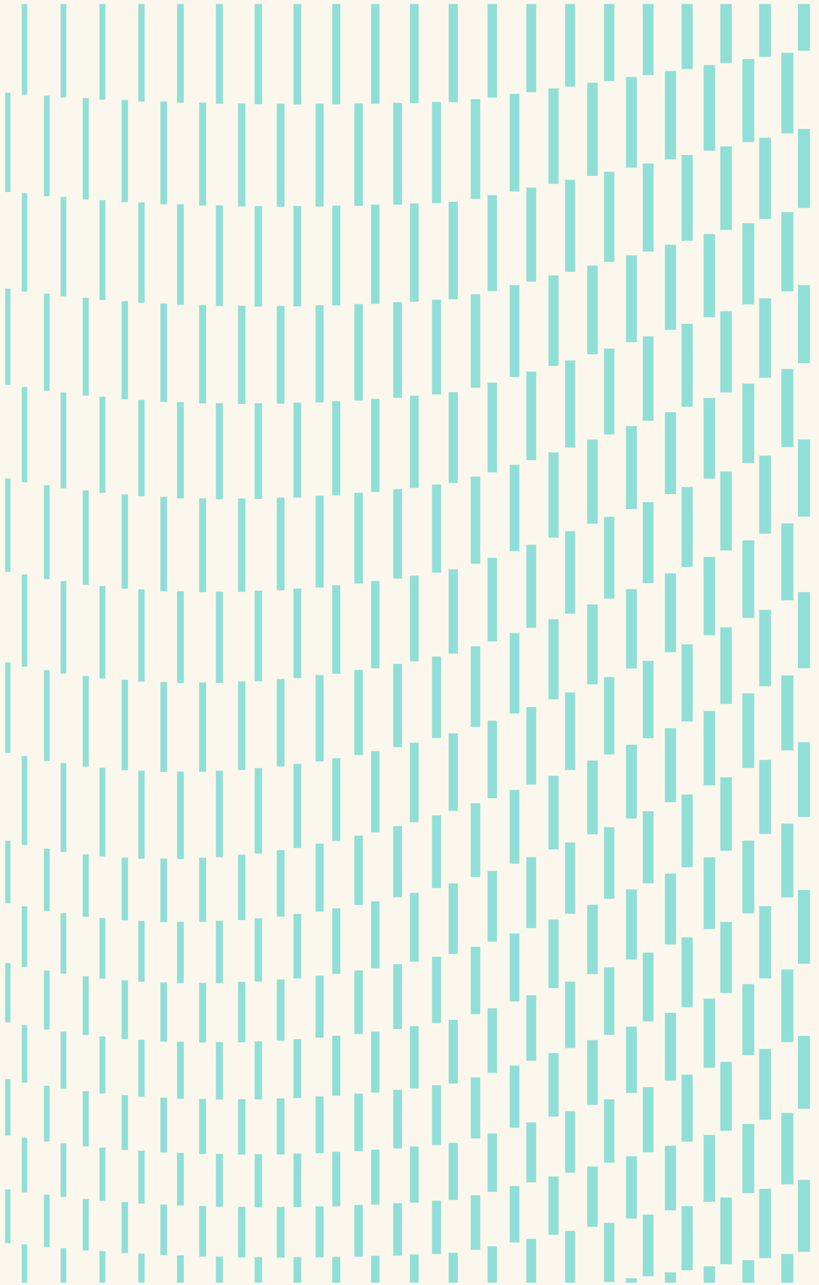
Actions

Spark Plans

Logical Plan

Optimized Logical Plan

Physical Plans



Lazy Evaluation

Spark doesn’t perform any transformations until an “action” is called.

```
1  df
2  > DataFrame[card_type: string, count:
3  bigint]
4
5
6
7
```

Actions

An “action” triggers Spark execution.

```
df.write()

df.head(n)
df.take(n)

df.collect()
df.show()
df.toPandas()
```

Spark Plans

```
df.explain(True)
```

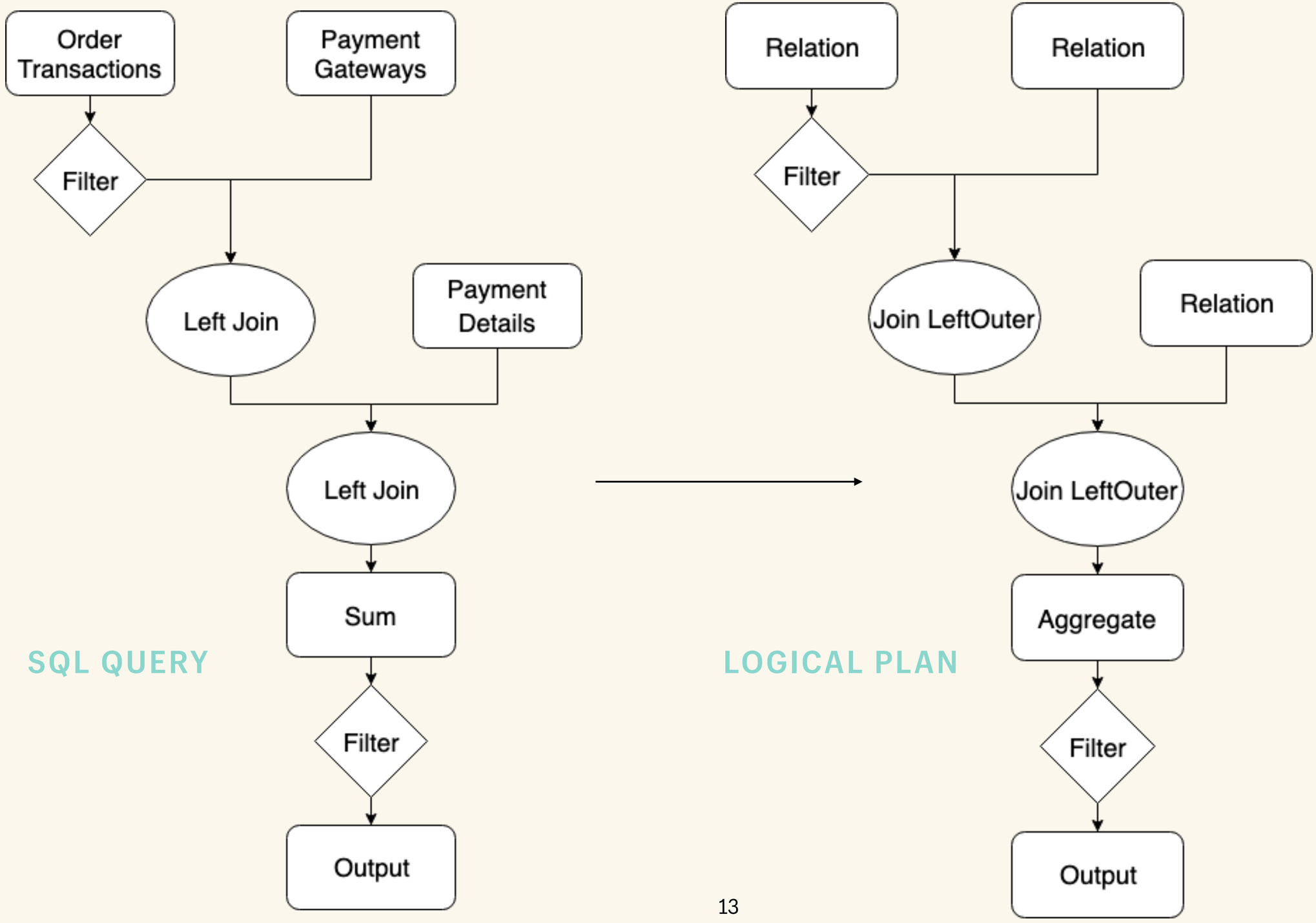
- Spark generates different execution plans from the spark DataFrames/RDD code:
  - Logical Plan
  - Optimized Logical Plan
  - Physical Plan
- Read bottom up, opposite of the plan in SparkUI

Spark Plans

```
== Physical Plan ==
*(10) HashAggregate(keys=[card_type#229], functions=[sum(1)], output=[card_type#229, count#289L])
+- Exchange hashpartitioning(card_type#229, 200)
   +- *(9) HashAggregate(keys=[card_type#229], functions=[partial_sum(1)], output=[card_type#229, sum#296L])
      +- *(9) Project [card_type#229]
         +- *(9) SortMergeJoin [order_transaction_id#202L], [order_transaction_id#160L], Inner
            :- *(6) Sort [order_transaction_id#202L ASC NULLS FIRST], false, 0
...
```

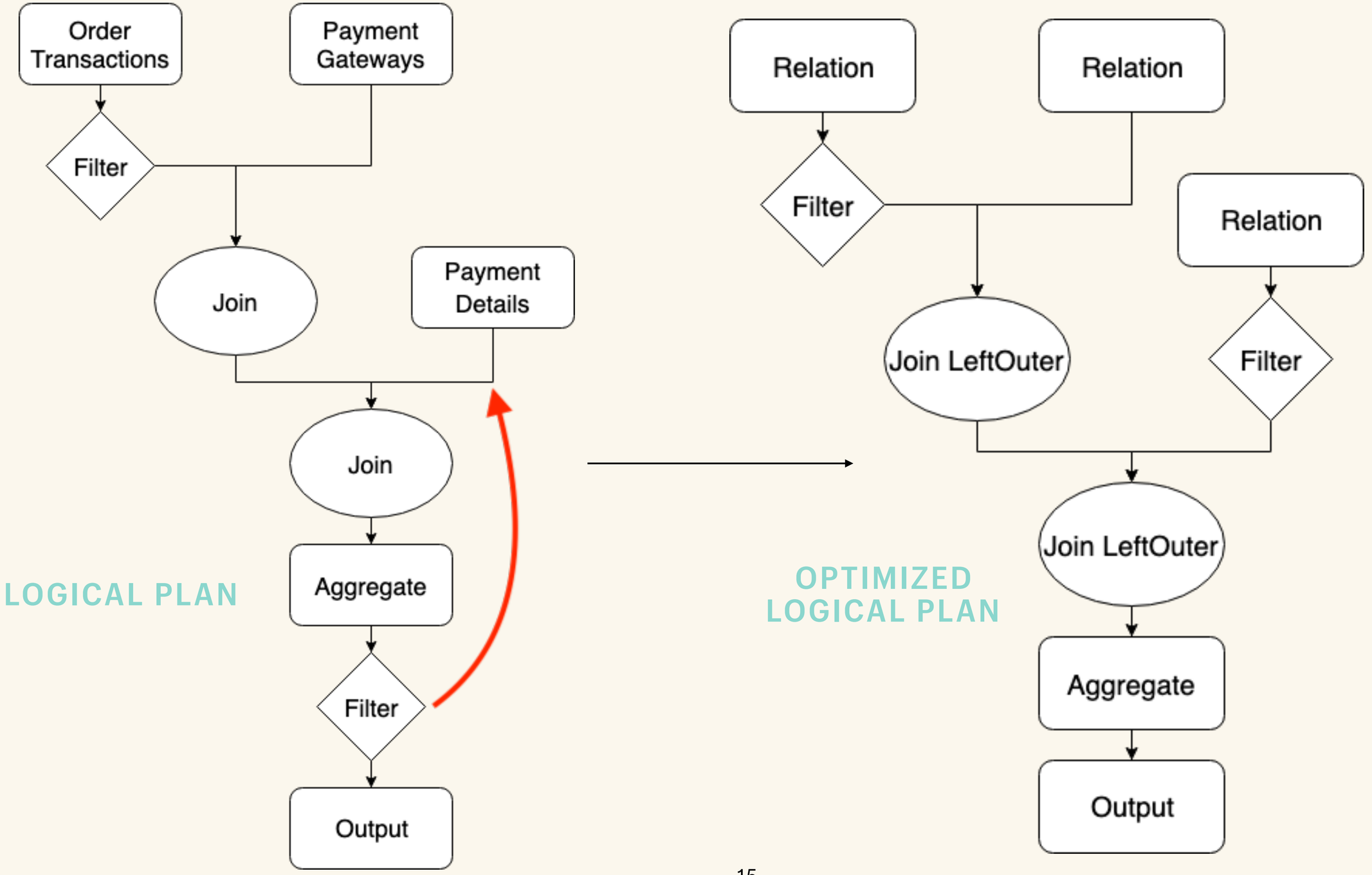
Logical Plan

- Is a set of abstract expressions that represent the spark code.


Order Transactions
Payment Gateways
Filter
Left Join
Payment Details
Left Join
Sum
Filter
Output
SQL QUERY
Relation
Relation
Filter
Join LeftOuter
Relation
Join LeftOuter
Aggregate
Filter
Output
LOGICAL PLAN

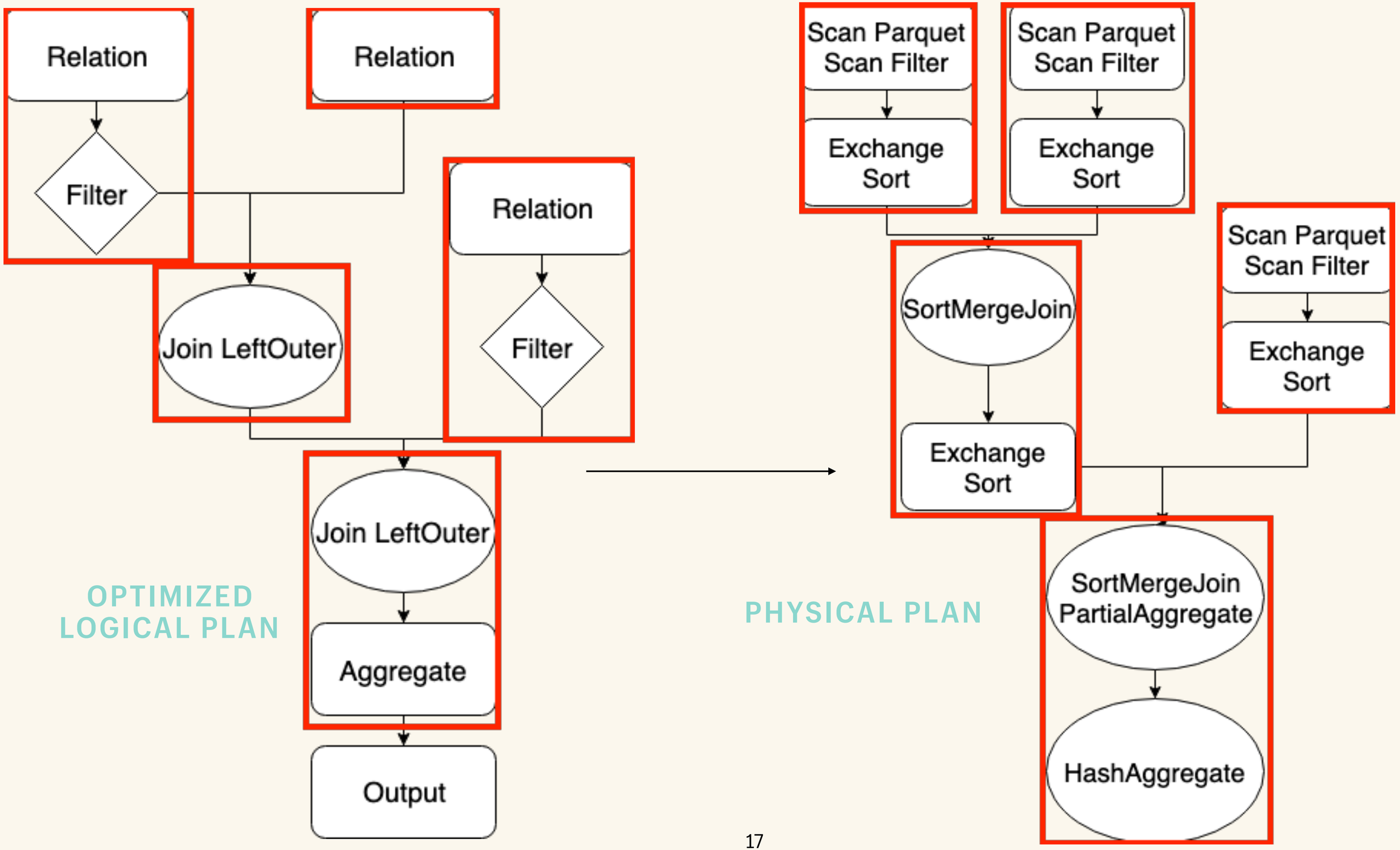
Optimized Logical Plan

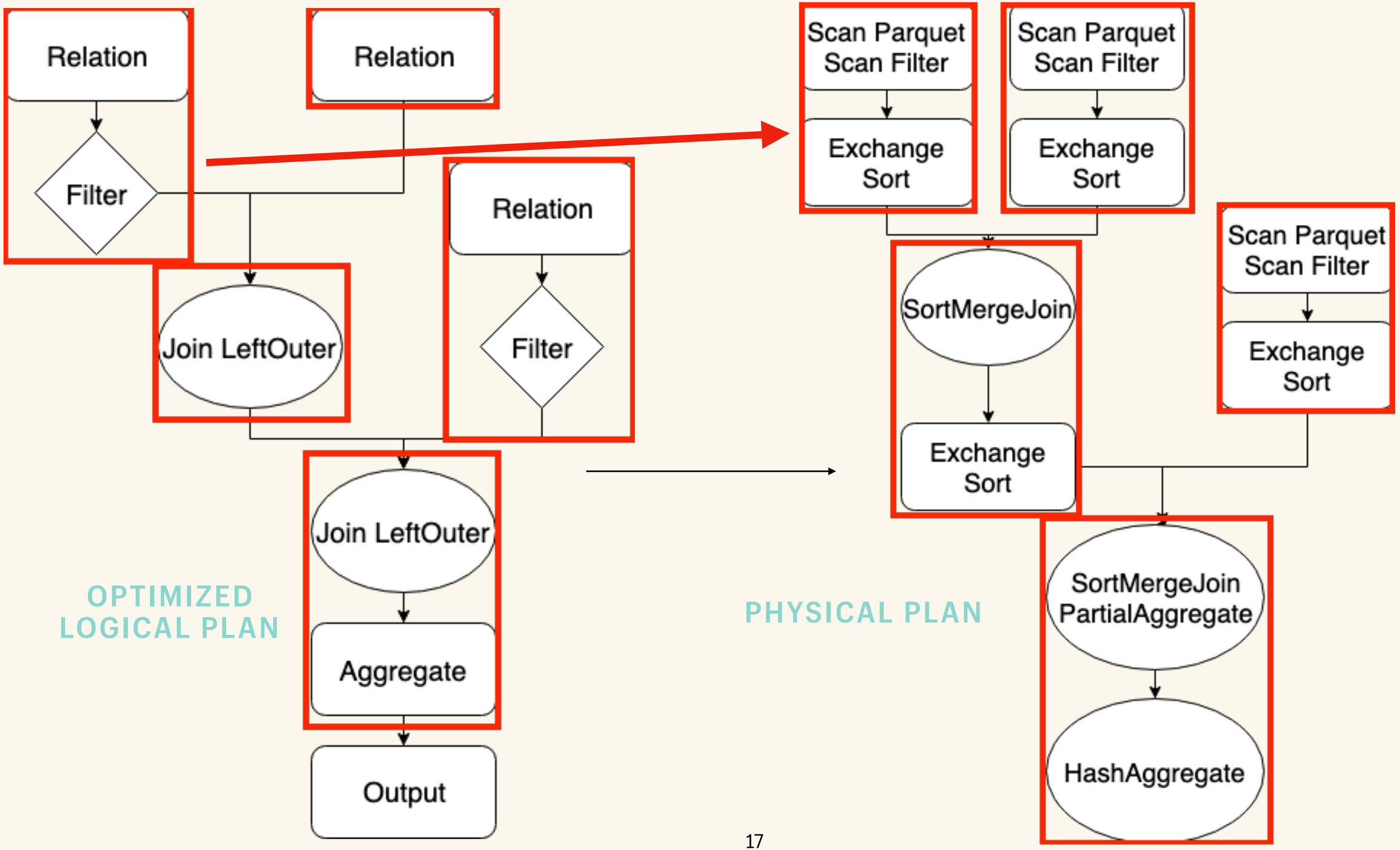
- Checks if set of expressions are valid.
  - ie. tables and columns exists.
- If valid, plan is passed to the Spark Catalyst Optimizer to be optimized.

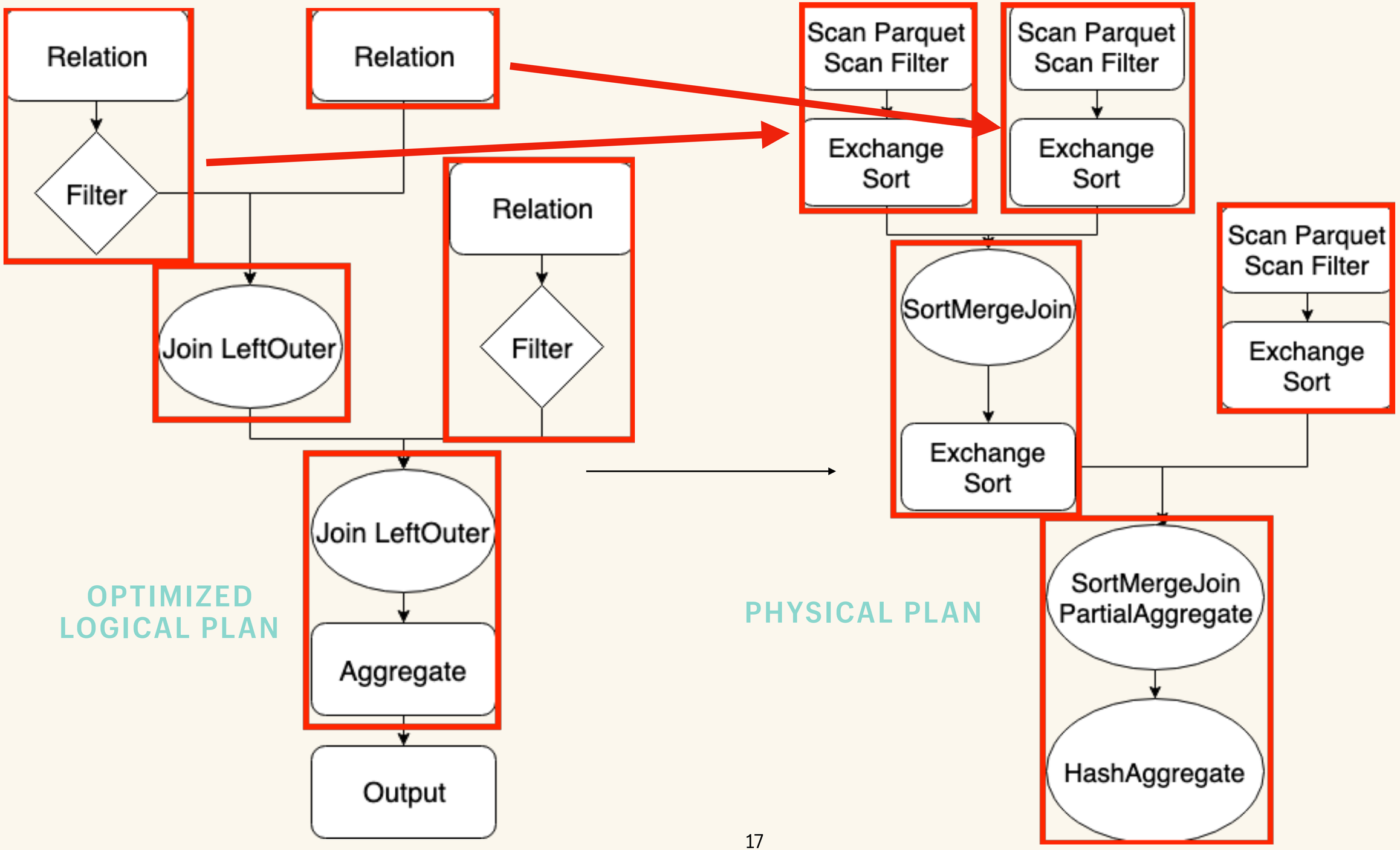

Order Transactions
Payment Gateways
Filter
Join
Payment Details
Join
Aggregate
Filter
Output
LOGICAL PLAN
Relation
Relation
Filter
Relation
Join LeftOuter
Filter
Join LeftOuter
Aggregate
Output
OPTIMIZED LOGICAL PLAN

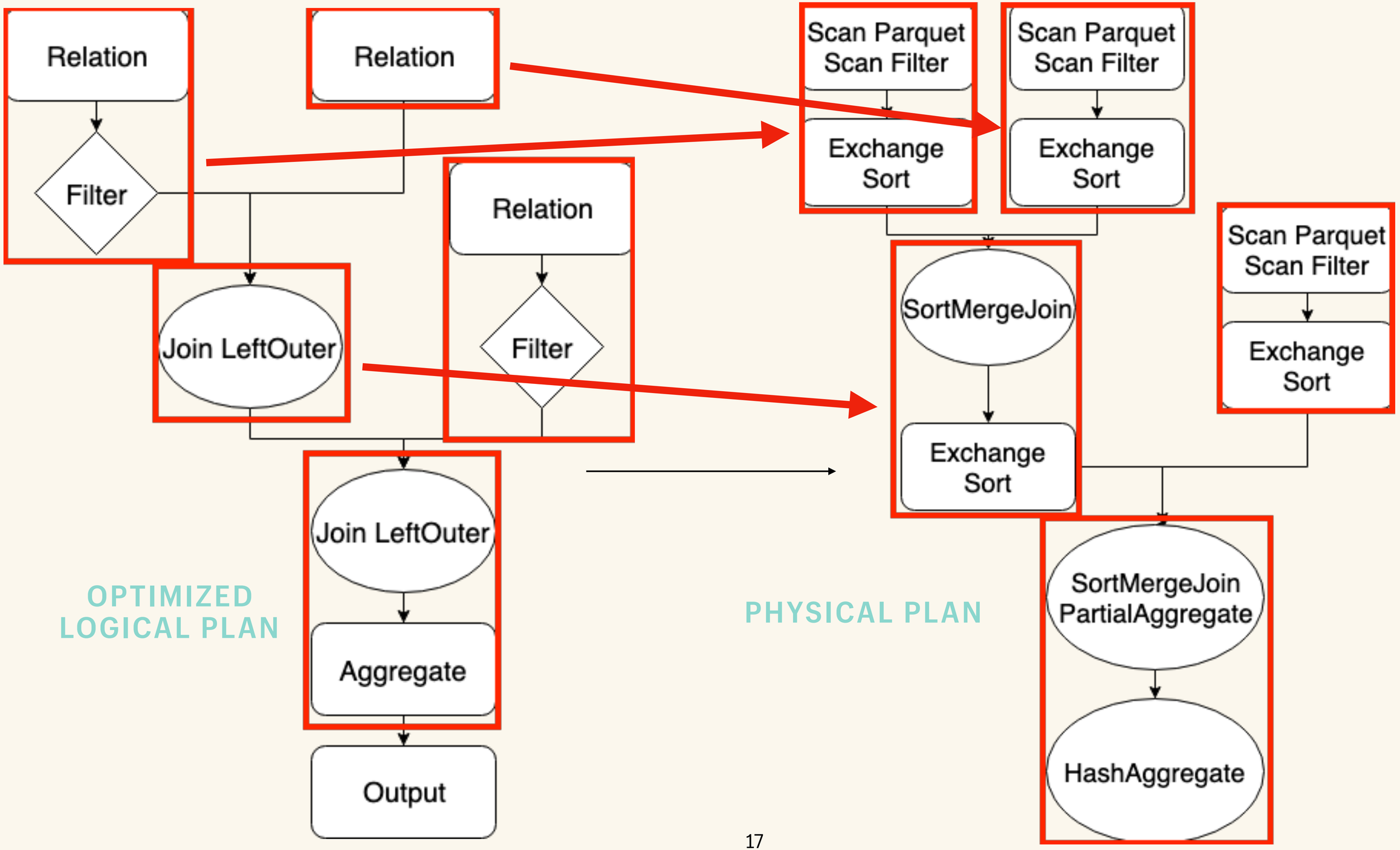
Physical Plan

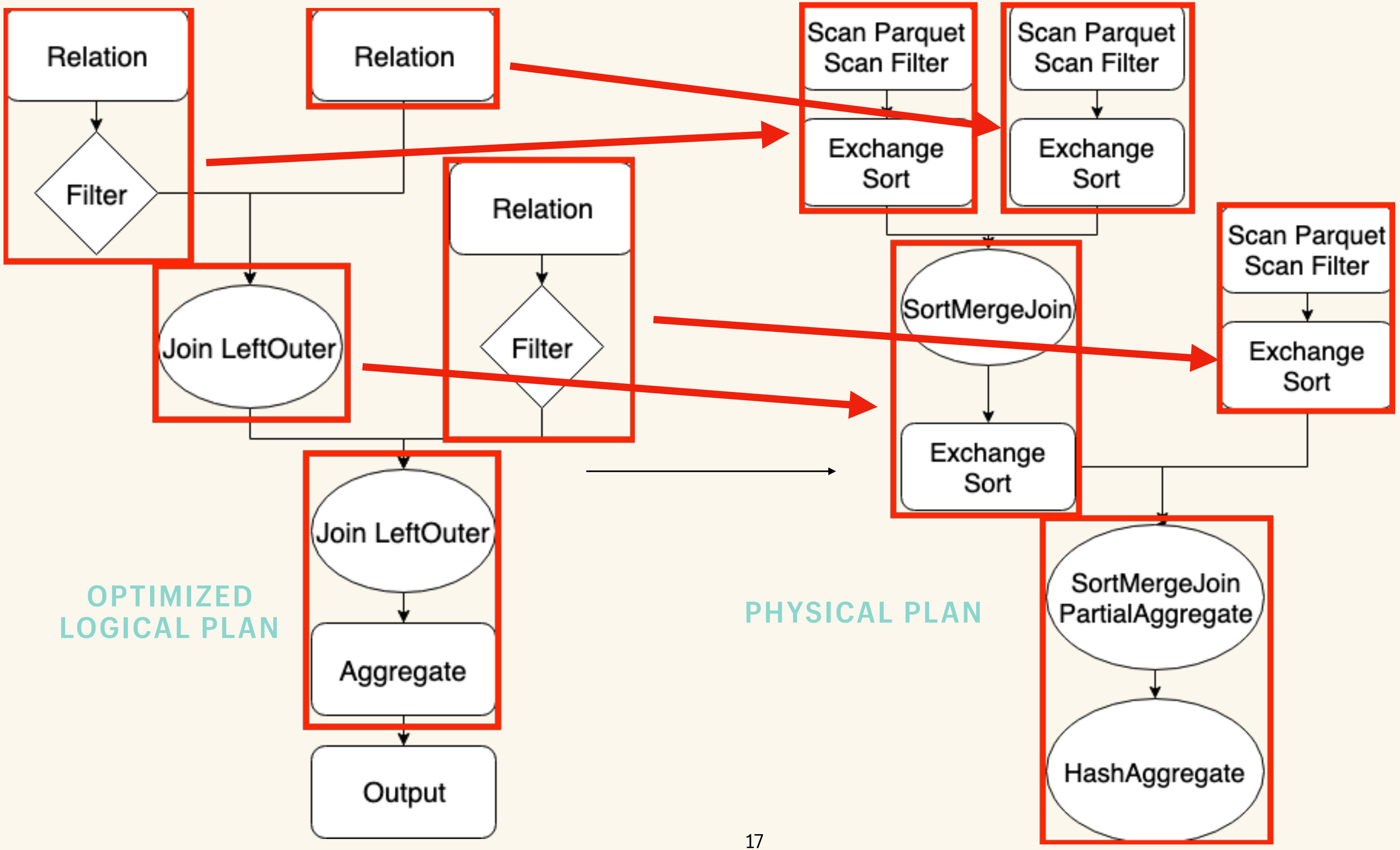
- Specifies exactly how the logical plan will execute.
- Multiple plans are generated and most optimal is selected.
- Based on the physical attributes of the tables.
    - ie. table size, and partition size.

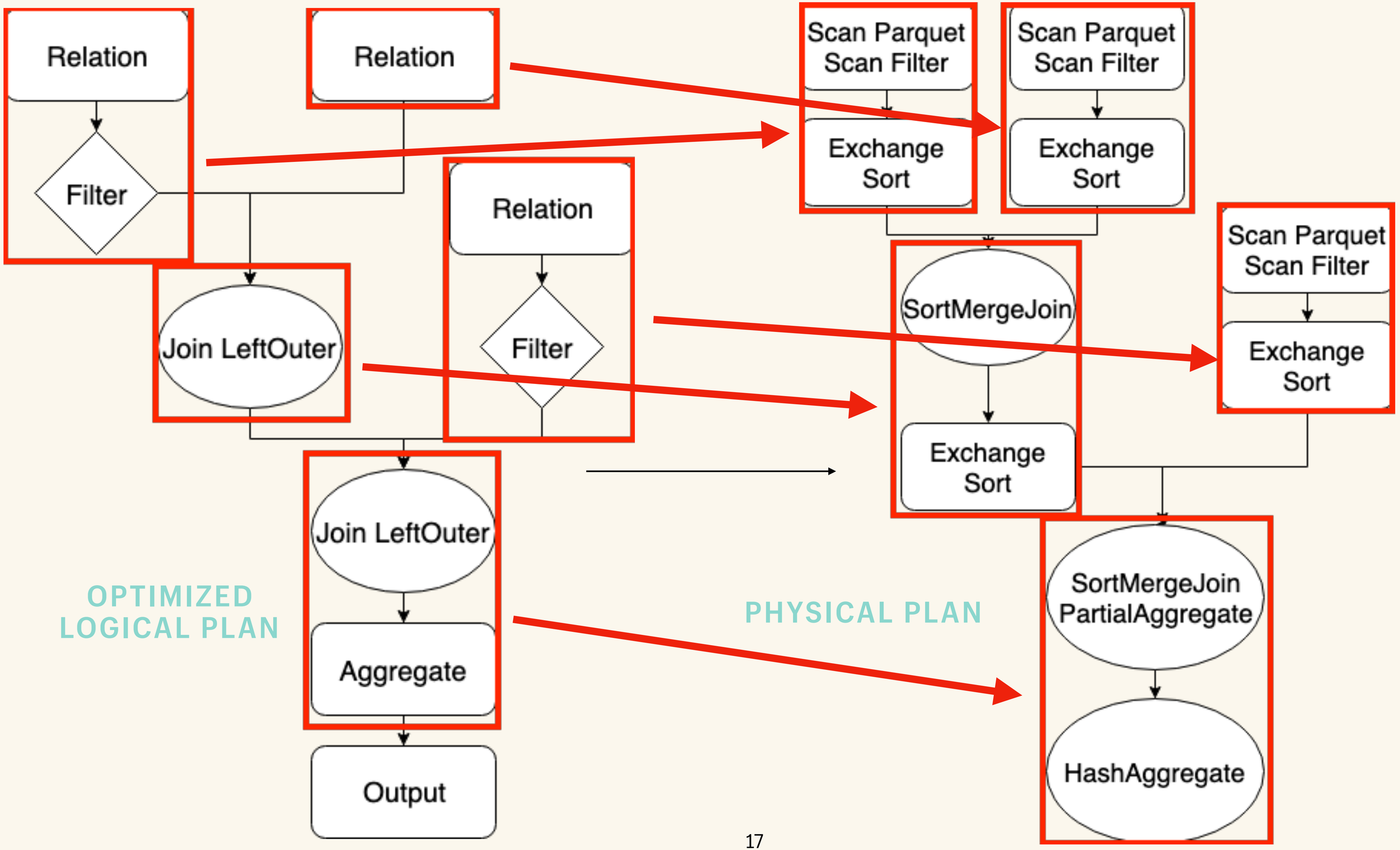

Relation
Filter
Relation
Join LeftOuter
Relation
Filter
Join LeftOuter
Aggregate
Output
OPTIMIZED
LOGICAL PLAN
Scan Parquet
Scan Filter
Exchange
Sort
Scan Parquet
Scan Filter
Exchange
Sort
SortMergeJoin
Exchange
Sort
Scan Parquet
Scan Filter
Exchange
Sort
PHYSICAL PLAN
SortMergeJoin
PartialAggregate
HashAggregate


Relation
Filter
Relation
Join LeftOuter
Relation
Filter
Join LeftOuter
Aggregate
Output
OPTIMIZED LOGICAL PLAN
Scan Parquet Scan Filter
Exchange Sort
Scan Parquet Scan Filter
Exchange Sort
SortMergeJoin
Exchange Sort
Scan Parquet Scan Filter
Exchange Sort
SortMergeJoin PartialAggregate
HashAggregate
PHYSICAL PLAN


Relation
Filter
Relation
Join LeftOuter
Relation
Filter
Join LeftOuter
Aggregate
Output
OPTIMIZED
LOGICAL PLAN
Scan Parquet
Scan Filter
Exchange
Sort
Scan Parquet
Scan Filter
Exchange
Sort
SortMergeJoin
Exchange
Sort
Scan Parquet
Scan Filter
Exchange
Sort
SortMergeJoin
PartialAggregate
HashAggregate
PHYSICAL PLAN


Relation
Filter
Relation
Join LeftOuter
Relation
Filter
Join LeftOuter
Aggregate
Output
OPTIMIZED
LOGICAL PLAN
Scan Parquet
Scan Filter
Exchange
Sort
Scan Parquet
Scan Filter
Exchange
Sort
SortMergeJoin
Exchange
Sort
Scan Parquet
Scan Filter
Exchange
Sort
SortMergeJoin
PartialAggregate
HashAggregate
PHYSICAL PLAN


Relation
Filter
Relation
Join LeftOuter
Relation
Filter
Join LeftOuter
Aggregate
Output
OPTIMIZED
LOGICAL PLAN
Scan Parquet
Scan Filter
Exchange
Sort
Scan Parquet
Scan Filter
Exchange
Sort
SortMergeJoin
Exchange
Sort
Scan Parquet
Scan Filter
Exchange
Sort
SortMergeJoin
PartialAggregate
HashAggregate
PHYSICAL PLAN


Relation
Filter
Relation
Join LeftOuter
Relation
Filter
Join LeftOuter
Aggregate
Output
OPTIMIZED
LOGICAL PLAN
Scan Parquet
Scan Filter
Exchange
Sort
Scan Parquet
Scan Filter
Exchange
Sort
SortMergeJoin
Exchange
Sort
Scan Parquet
Scan Filter
Exchange
Sort
SortMergeJoin
PartialAggregate
HashAggregate
PHYSICAL PLAN

Section 3: Optimization 1 - Improving Joins

SparkUI

Spark DAG

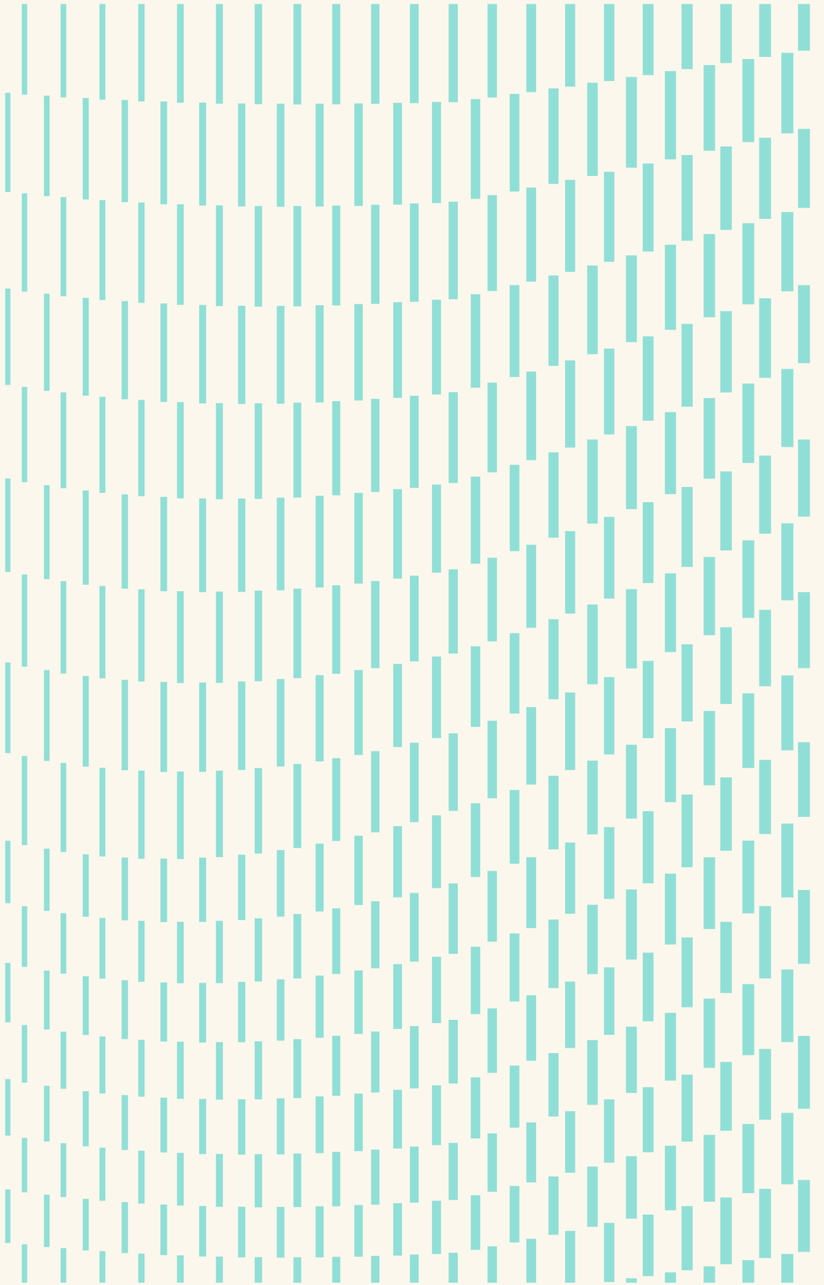
Shuffle

Broadcasting

Spark SQL tab

Join Skew

Join Strategies



Spark Jobs (?)

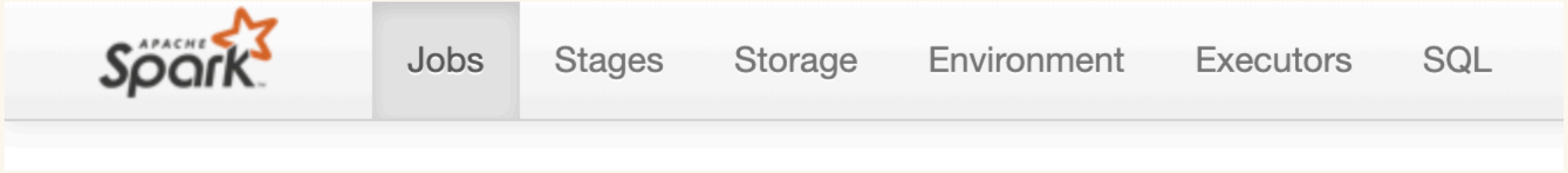
User: ericxiao
Total Uptime: 6.4 h
Scheduling Mode: FIFO
Completed Jobs: 6

▸ Event Timeline

Completed Jobs (6)

Job Id ▾	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
5	showString at NativeMethodAccessorImpl.java:0 showString at NativeMethodAccessorImpl.java:0	2019/07/25 14:44:53	7.9 min	6/6	31795/31795
4	showString at NativeMethodAccessorImpl.java:0 showString at NativeMethodAccessorImpl.java:0	2019/07/25 14:36:31	8.4 min	6/6	31315/31315
3	showString at NativeMethodAccessorImpl.java:0 showString at NativeMethodAccessorImpl.java:0	2019/07/25 14:19:18	17 min	6/6	16686/16686 (1 killed: another attempt succeeded)
2	parquet at NativeMethodAccessorImpl.java:0 parquet at NativeMethodAccessorImpl.java:0	2019/07/25 14:19:15	0.2 s	1/1	1/1
1	parquet at NativeMethodAccessorImpl.java:0 parquet at NativeMethodAccessorImpl.java:0	2019/07/25 14:19:11	0.1 s	1/1	1/1
0	parquet at NativeMethodAccessorImpl.java:0 parquet at NativeMethodAccessorImpl.java:0	2019/07/25 14:18:47	21 s	1/1	1/1

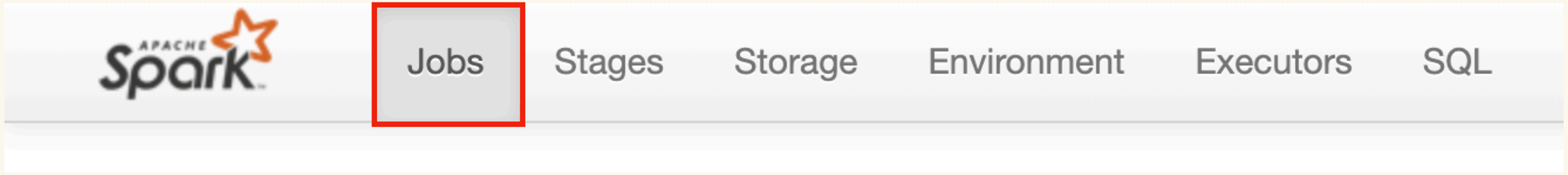
SparkUI Tabs



Main tabs:

- **Jobs**
- **SQL**
- **Storage**

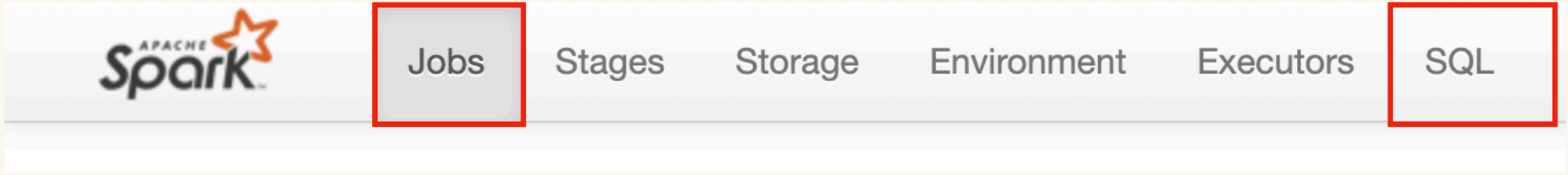
SparkUI Tabs



Main tabs:

- **Jobs**
- **SQL**
- **Storage**

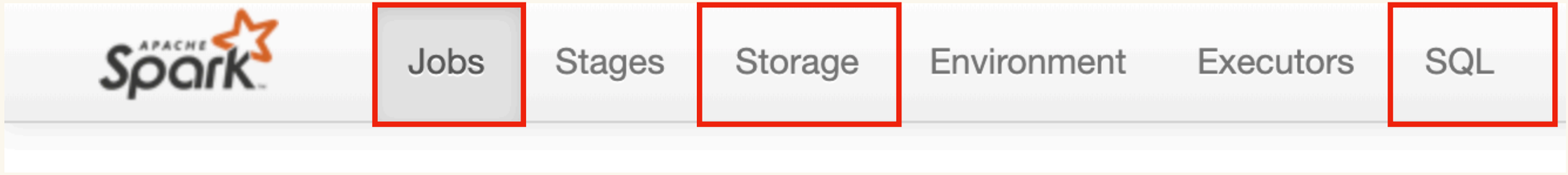
SparkUI Tabs



Main tabs:

- **Jobs**
- **SQL**
- **Storage**

SparkUI Tabs

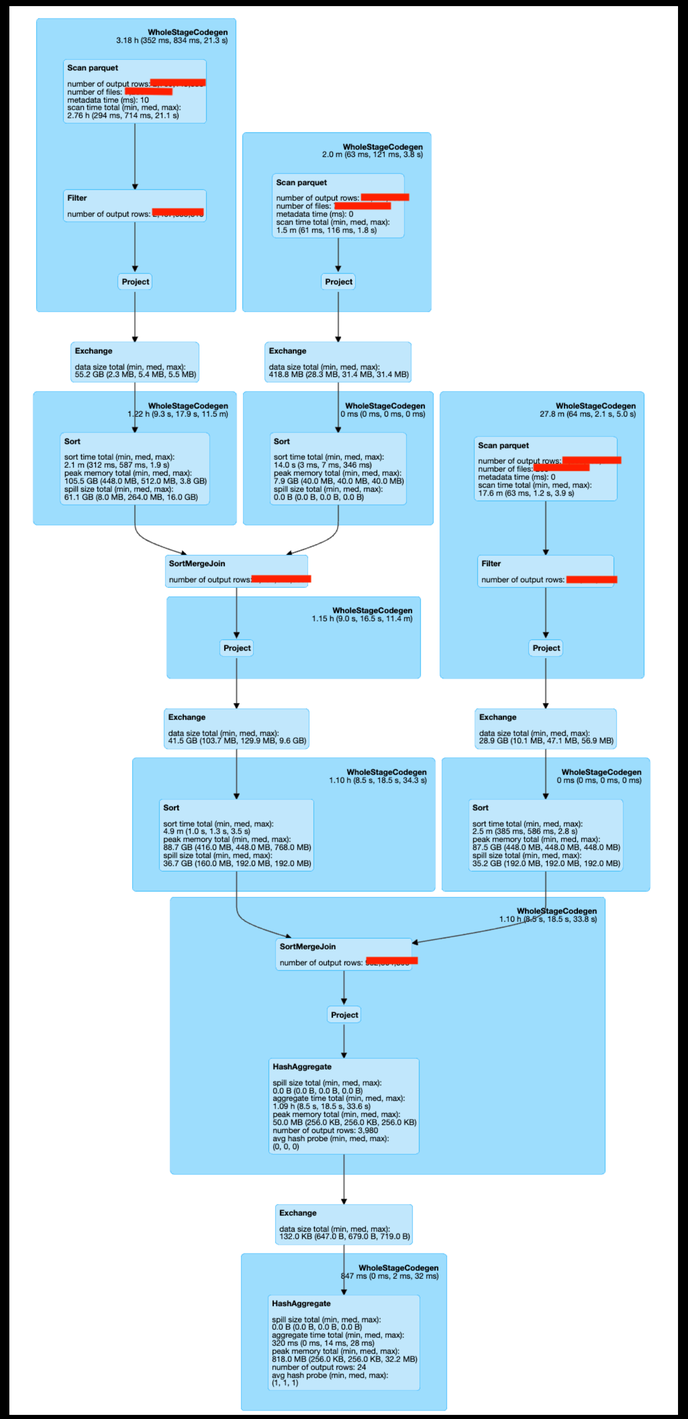


Main tabs:

- **Jobs**
- **SQL**
- **Storage**

Spark DAG

- Directed Acyclic Graph.
  - No cycles.
  - Data flows in one direction.
- A Spark DAG represents a Spark Job.
- A spark job consists of multiple stages.
- These stages *can* run in parallel.



Shuffle

- Certain transformations in Spark trigger an event known as the Shuffle.
- A shuffle re-distributes data so that it's grouped differently across partitions.
- Why? Not all values needed for a transformation exist on the same partition or machine at the time of the transformation, but they must be co-located to perform the transformation.
- This involves copying data across executors and machines.
- Which is why shuffling data is complex and a costly operation.

reference: http://people.apache.org/~pwendell/spark-nightly/spark-master-docs/latest/rdd-programming-guide.html#shuffle-operations

Shuffle

Operations that *might* cause a shuffle:

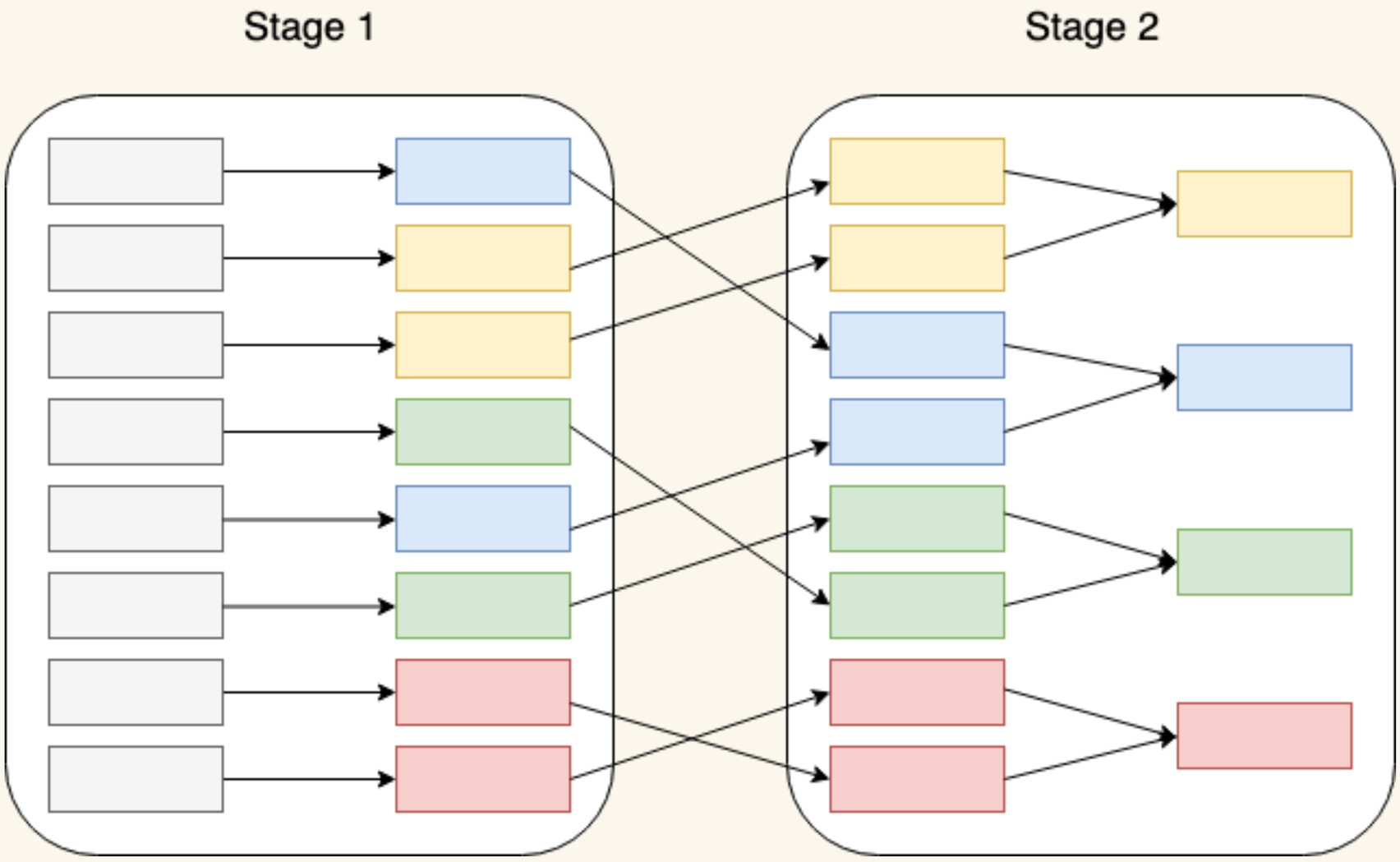
- Joins
- GroupBy
- Distinct
- Repartition
- Coalesce
- Window

Shuffle

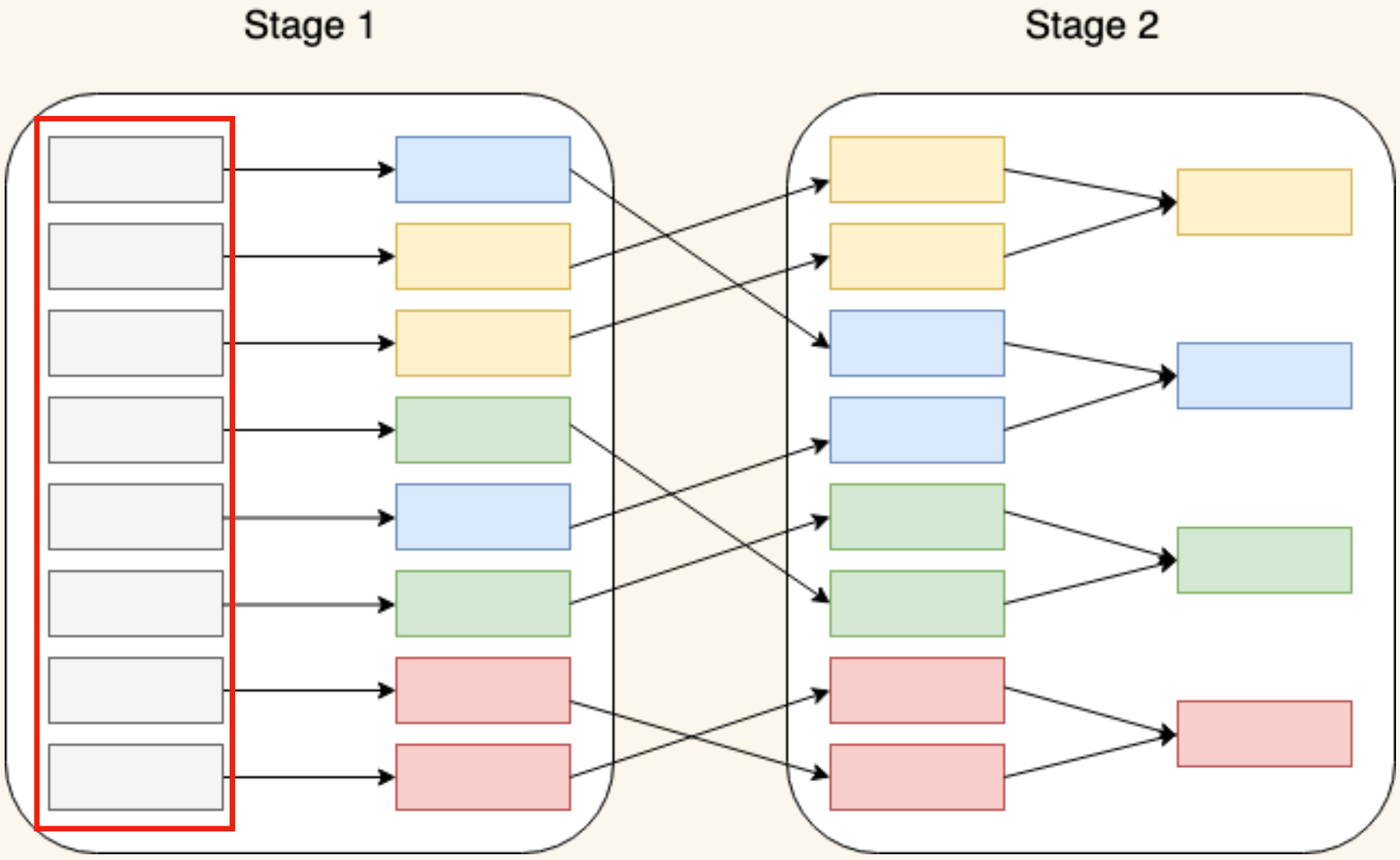
When a shuffle isn’t performed:

- Data is sorted / bucketed by the partition key on disk.
- Data has already been shuffled from a previous operation.
    - ex. When an aggregate happens after a join but both are performed on the same column.

Shuffle



Shuffle

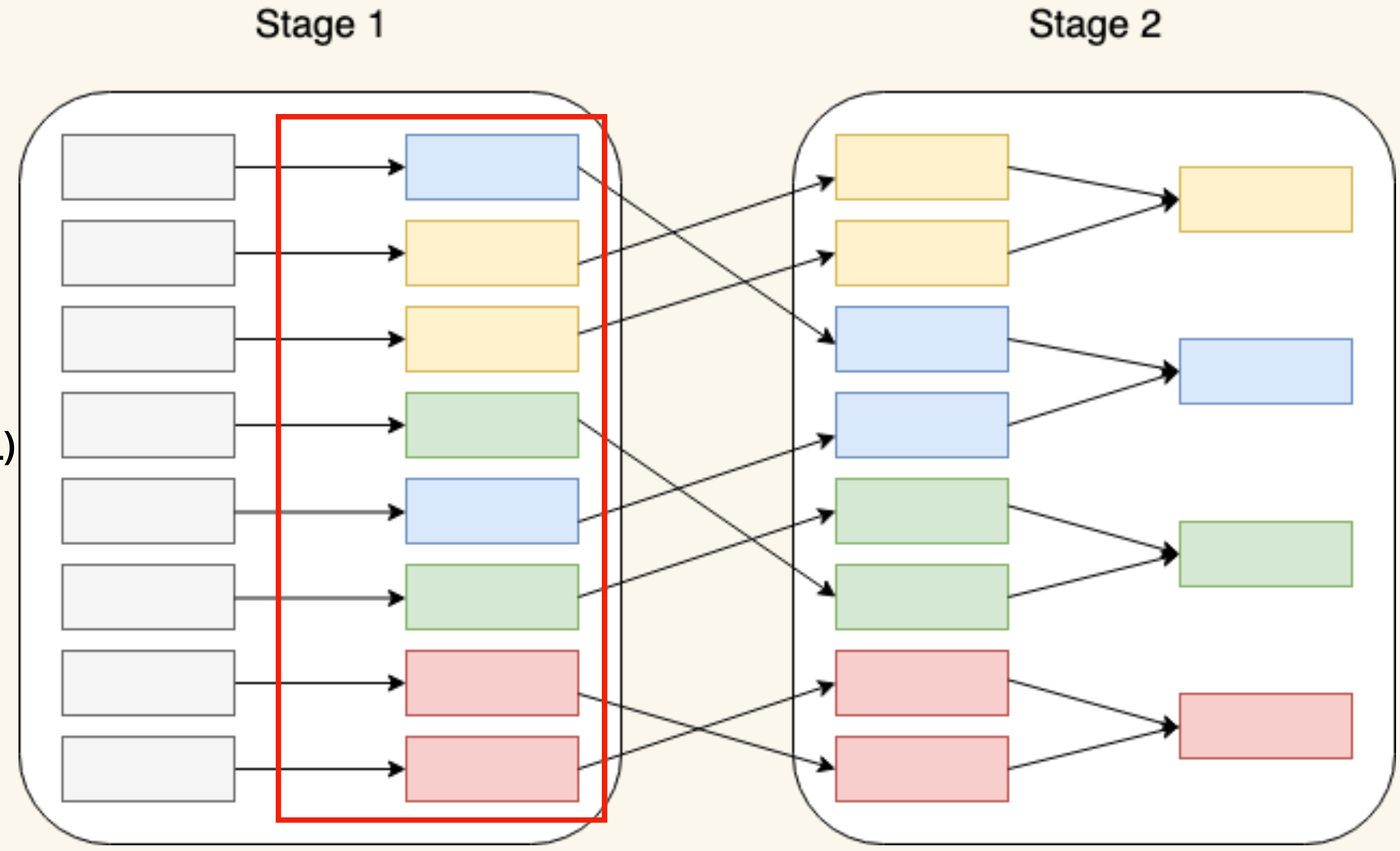


1. Initial partitions read from disk.

Shuffle

ex.

```
(
   df
   .withColumn(x, x + 1)
)
```

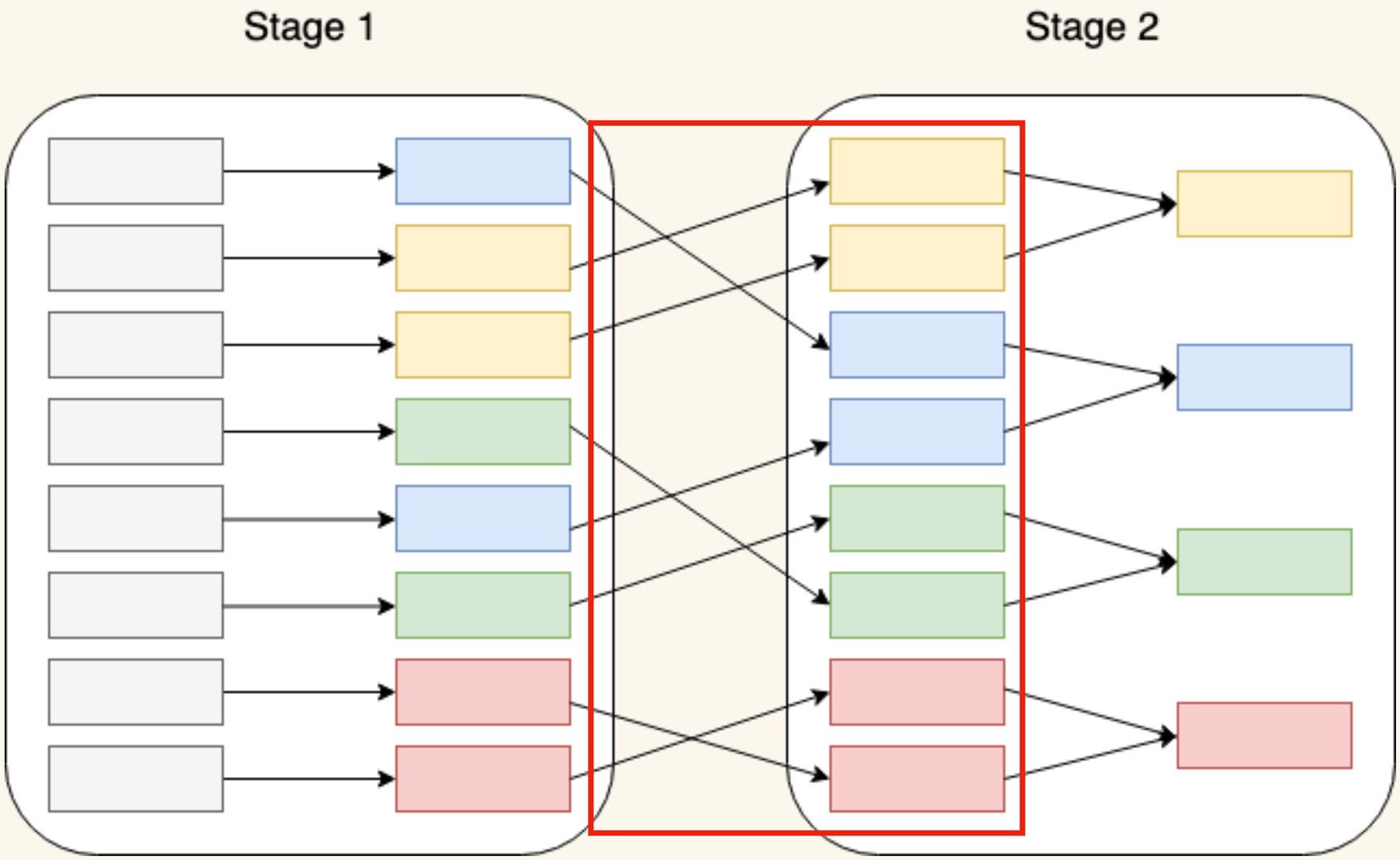


2. apply transformations on each partition.

Shuffle

ex.

```
(
   df
   .groupBy(y)
   .agg(...)
)
```

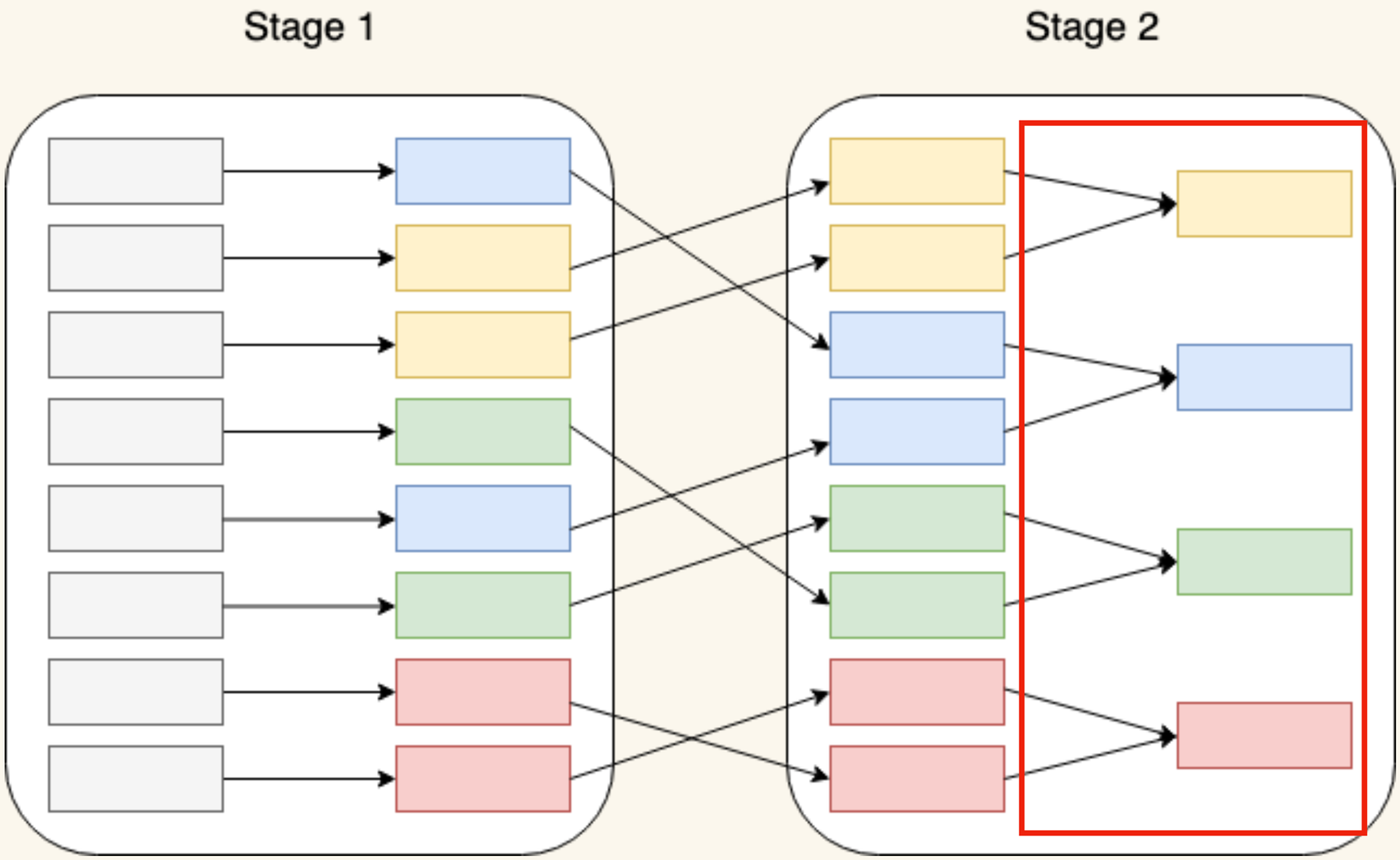


3. shuffle data from stage 1 for stage 2.

Shuffle

ex.

```
(
   df
   .groupBy(y)
   .agg(...)
)
```



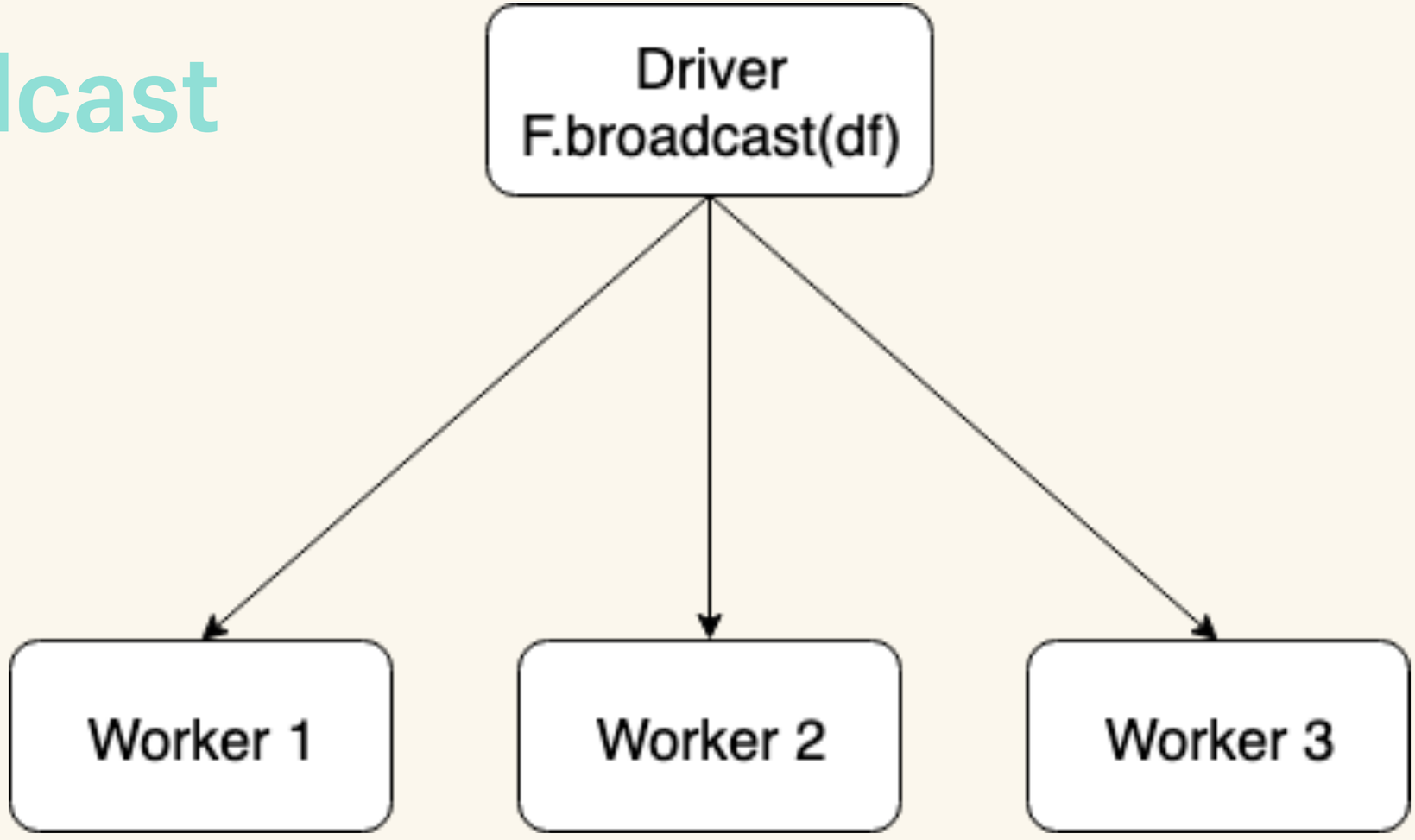
4. perform stage 2 transformations.

Broadcast

- When data is small enough, Spark can broadcast it.
- DataFrame will be executed.
- Instead of shuffling data, Spark (the driver) will “broadcast” the DataFrame to every executor.

```
1  df = ...
2  broadcasted_df = broadcast(df)
3
4  spark.sql.autoBroadcastJoinThreshold
5
6
7
```

Broadcast



Broadcast Gotchas

- As mentioned before the broadcasted dataFrame is executed before being broadcasted.
- If the transformations performed are complex or take long enough, this will cause a timeout.
- This wait time is defined by spark.sql.broadcastTimeout which is 300s (5 minutes by default).

RED FLAGS:

- TimeoutException: Futures timed out after [300 seconds].

Solution:

- Increase the timeout, 10 minutes should be the upper limit.

SQL Tab

- Shows the physical plan of your Spark application.
- Shows an aggregate level of task duration and data volume.
- Dropdown of all the other plans.

1. Overall DAG

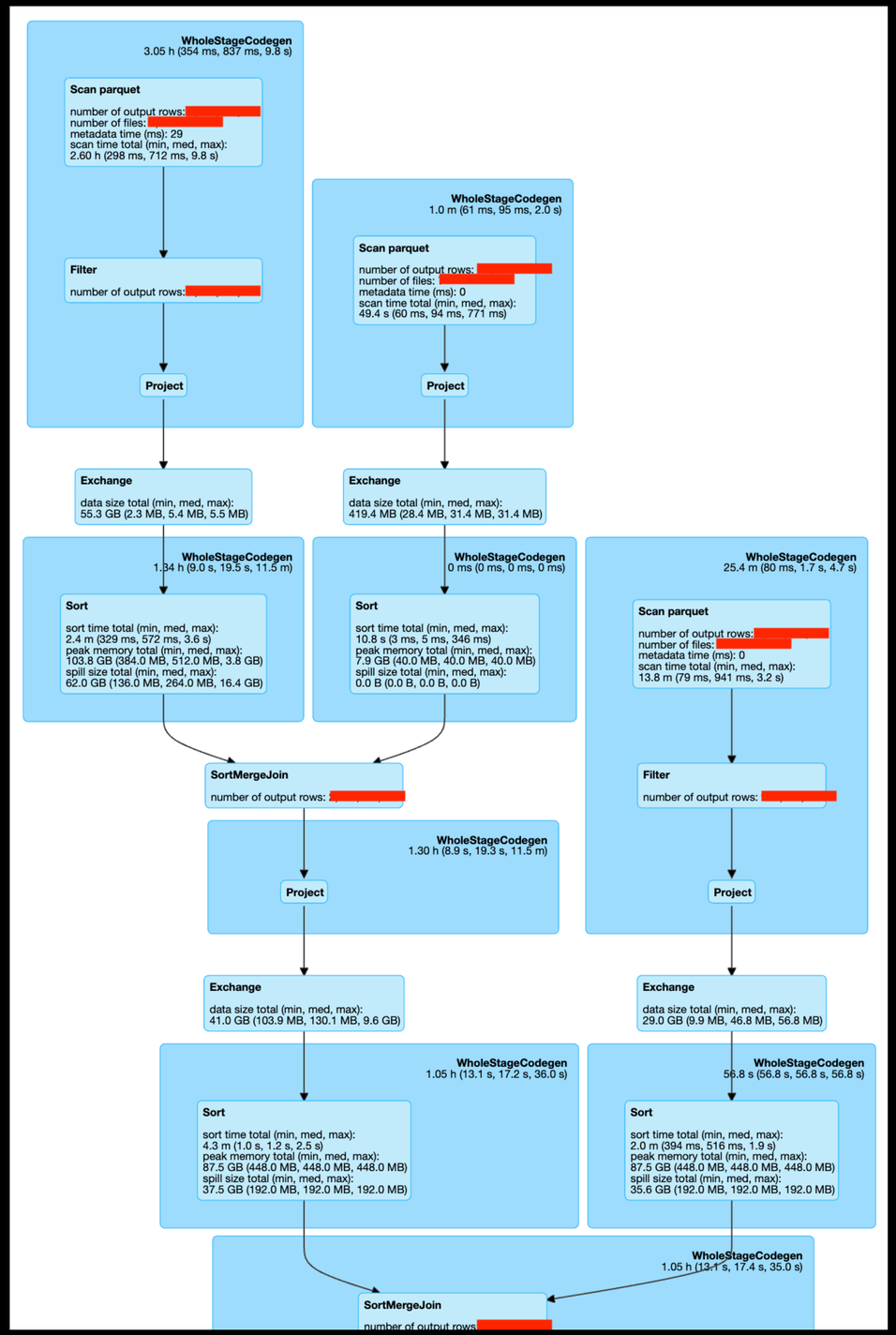
- Visual representation of physical plan.
- It flows from top (reading data) to bottom (writing / outputting data).

RED FLAGS:

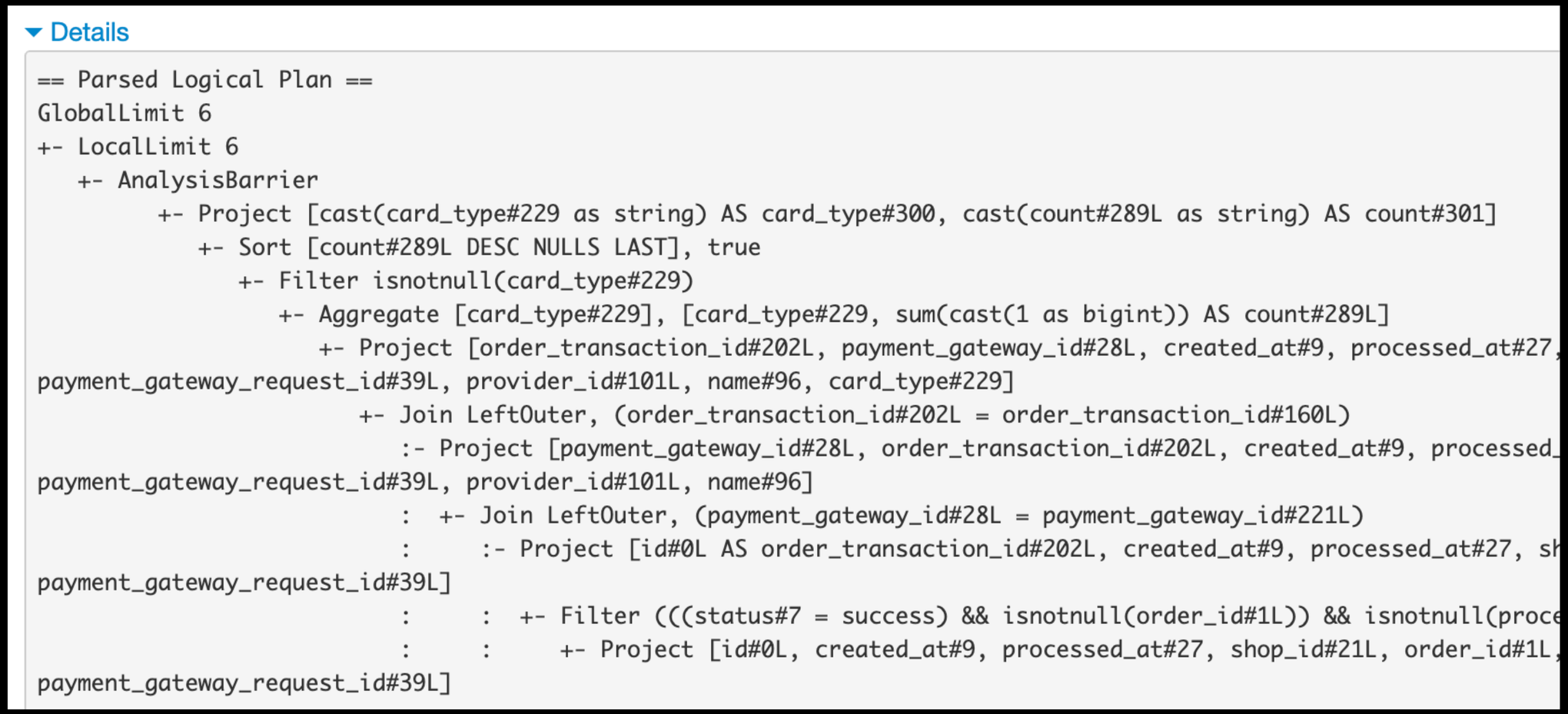
- Too large and wide DAG.

Solution:

- Reduce application complexity.

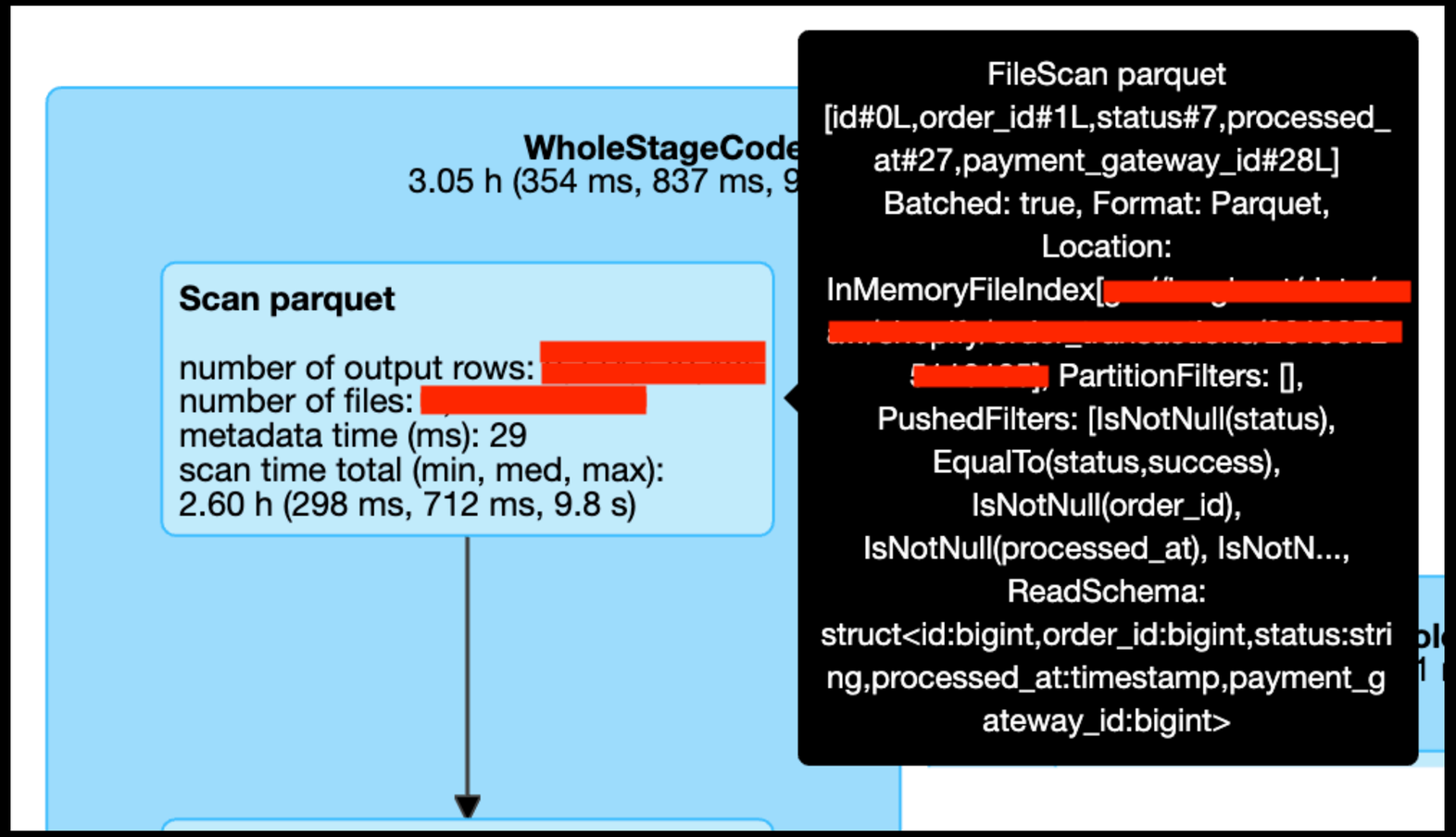


2. Spark Plans



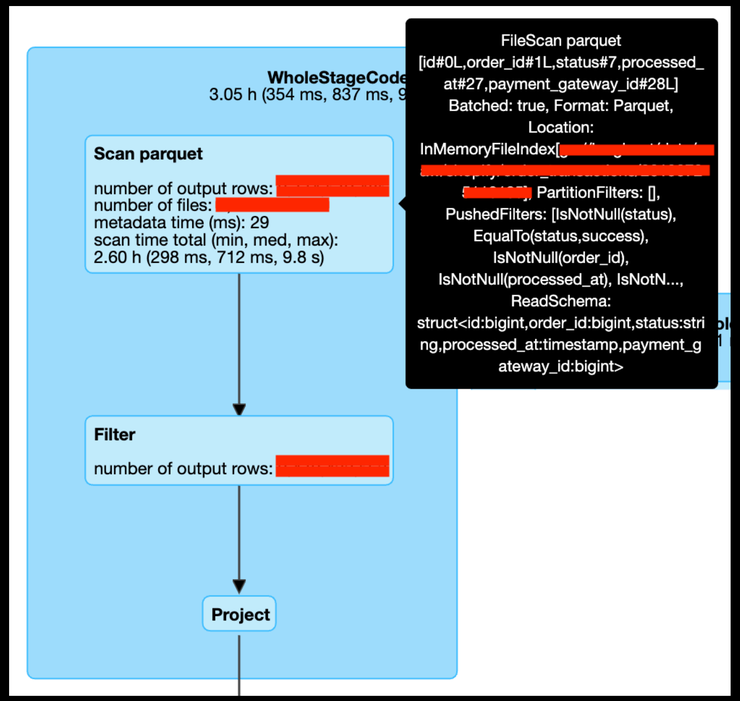
3. Transformation Details

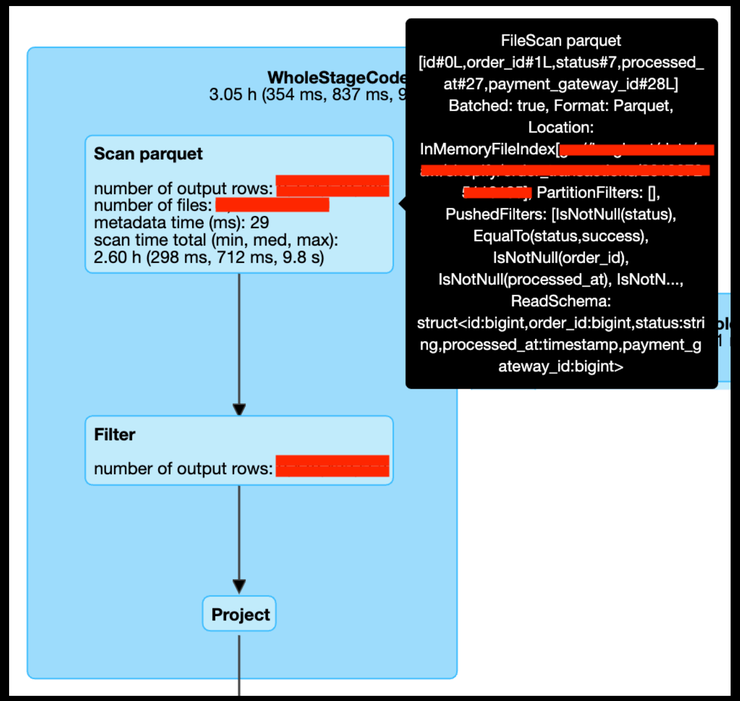
- Hover over light blue blobs to see more code details.
- Gives hints to line of code in Spark application.

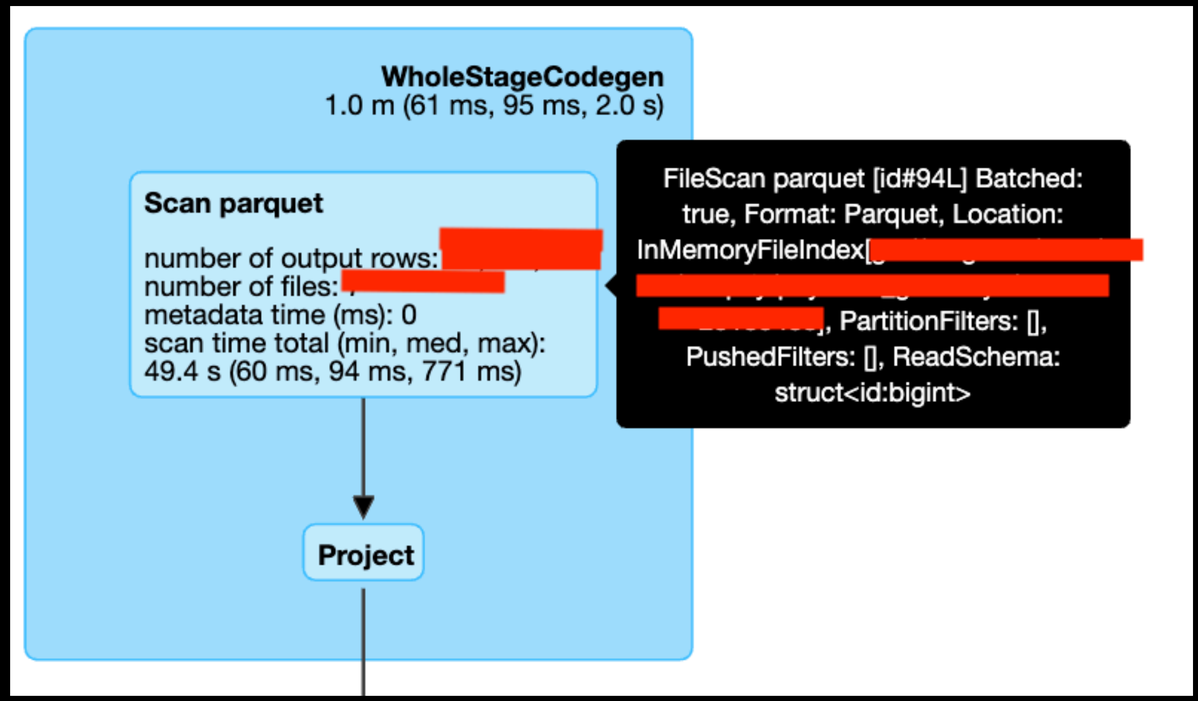


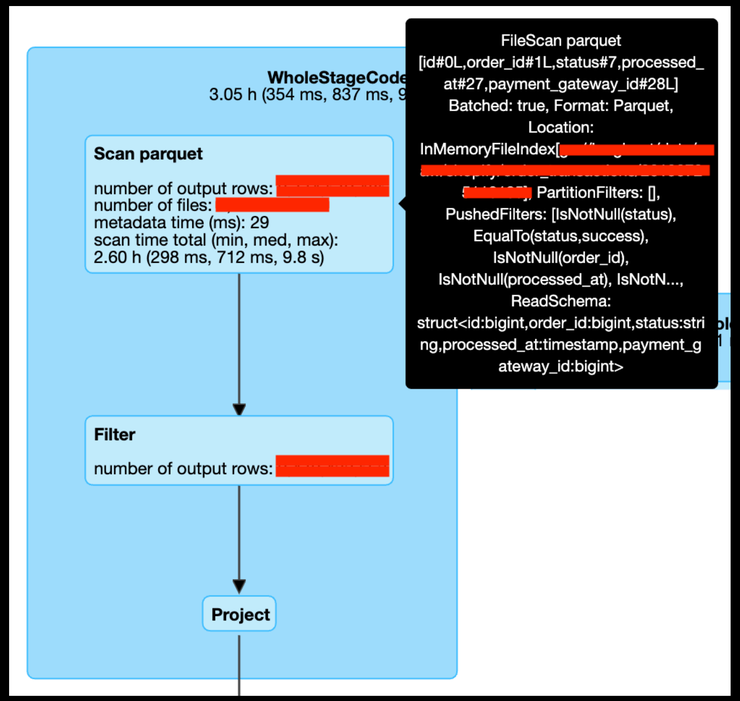
SQL Query (Again)

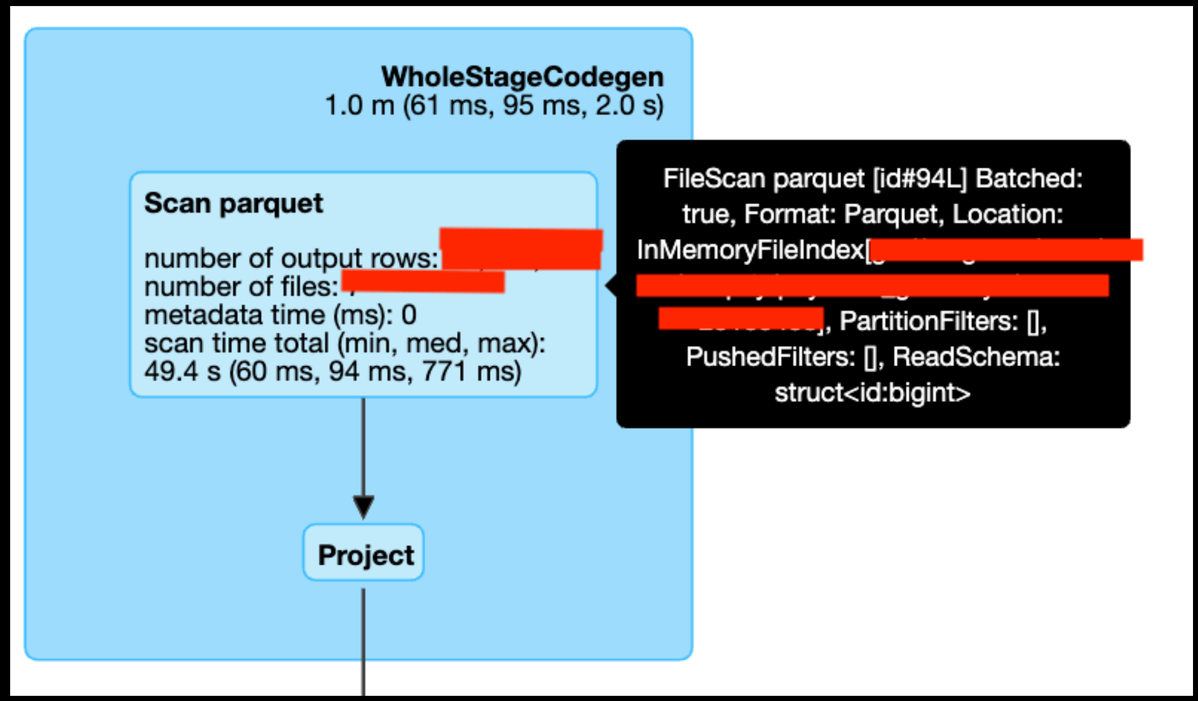
```
SELECT
   card_type,
   count(*)
FROM order_transactions
LEFT JOIN payment_gateways USING (payment_gateway_id)
LEFT JOIN payment_details USING (payment_detail_id)
GROUP BY 1
WHERE ....
```

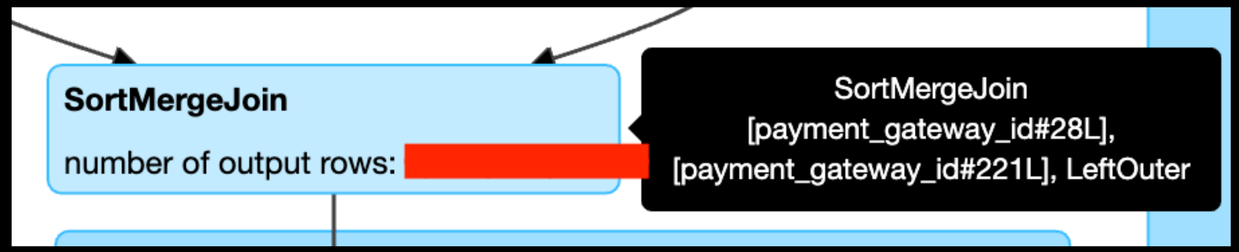

WholeStageCode
3.05 h (354 ms, 837 ms, 9
Scan parquet
number of output rows:
number of files:
metadata time (ms): 29
scan time total (min, med, max):
2.60 h (298 ms, 712 ms, 9.8 s)
Filter
number of output rows:
Project
FileScan parquet
[id#0L,order_id#1L,status#7,processed_
at#27,payment_gateway_id#28L]
Batched: true, Format: Parquet,
Location:
InMemoryFileIndex[
PartitionFilters: [],
PushedFilters: [IsNotNull(status),
EqualTo(status,success),
IsNotNull(order_id),
IsNotNull(processed_at), IsNotN...,
ReadSchema:
struct<id:bigint,order_id:bigint,status:stri
ng,processed_at:timestamp,payment_g
ateway_id:bigint>


WholeStageCode
3.05 h (354 ms, 837 ms, 9
Scan parquet
number of output rows:
number of files:
metadata time (ms): 29
scan time total (min, med, max):
2.60 h (298 ms, 712 ms, 9.8 s)
Filter
number of output rows:
Project
FileScan parquet [id#0L,order_id#1L,status#7,processed_at#27,payment_gateway_id#28L] Batched: true, Format: Parquet, Location: InMemoryFileIndex[
PartitionFilters: [], PushedFilters: [IsNotNull(status), EqualTo(status,success), IsNotNull(order_id), IsNotNull(processed_at), IsNotN..., ReadSchema: struct<id:bigint,order_id:bigint,status:string,processed_at:timestamp,payment_gateway_id:bigint>

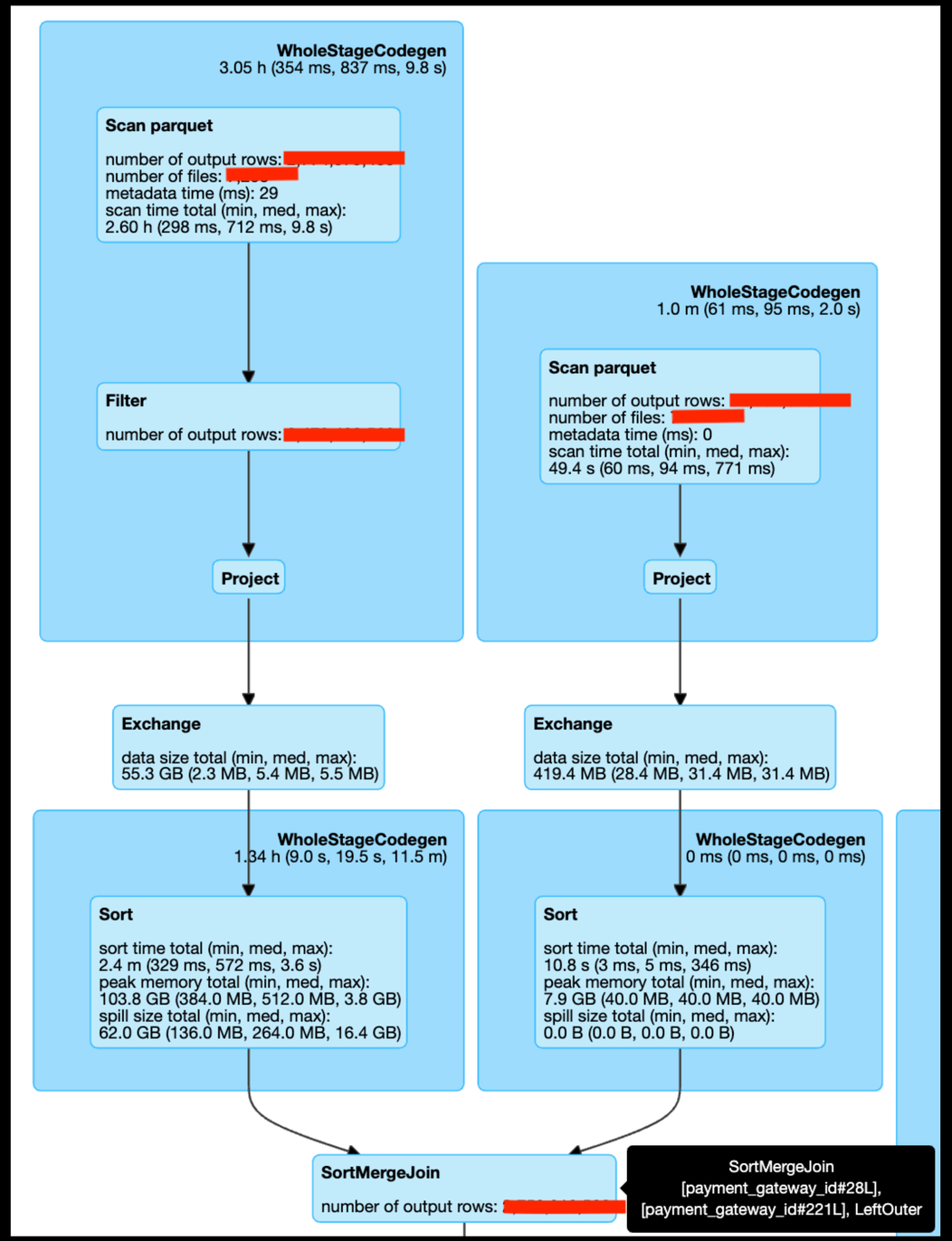

WholeStageCodegen
1.0 m (61 ms, 95 ms, 2.0 s)
Scan parquet
number of output rows:
number of files:
metadata time (ms): 0
scan time total (min, med, max):
49.4 s (60 ms, 94 ms, 771 ms)
Project
FileScan parquet [id#94L] Batched: true, Format: Parquet, Location: InMemoryFileIndex[
, PartitionFilters: [], PushedFilters: [], ReadSchema: struct<id:bigint>


WholeStageCode
3.05 h (354 ms, 837 ms, 9
Scan parquet
number of output rows:
number of files:
metadata time (ms): 29
scan time total (min, med, max):
2.60 h (298 ms, 712 ms, 9.8 s)
Filter
number of output rows:
Project
FileScan parquet [id#0L,order_id#1L,status#7,processed_at#27,payment_gateway_id#28L] Batched: true, Format: Parquet, Location: InMemoryFileIndex[
PartitionFilters: [], PushedFilters: [IsNotNull(status), EqualTo(status,success), IsNotNull(order_id), IsNotNull(processed_at), IsNotN..., ReadSchema: struct<id:bigint,order_id:bigint,status:string,processed_at:timestamp,payment_gateway_id:bigint>


WholeStageCodegen
1.0 m (61 ms, 95 ms, 2.0 s)
Scan parquet
number of output rows:
number of files:
metadata time (ms): 0
scan time total (min, med, max):
49.4 s (60 ms, 94 ms, 771 ms)
Project
FileScan parquet [id#94L] Batched: true, Format: Parquet, Location: InMemoryFileIndex[
, PartitionFilters: [], PushedFilters: [], ReadSchema: struct<id:bigint>


SortMergeJoin
number of output rows:
SortMergeJoin [payment_gateway_id#28L], [payment_gateway_id#221L], LeftOuter

```
SELECT
   card_type,
   count(*)
FROM order_transactions
JOIN payment_gateways ON payment_gateway_id
JOIN payment_details ON payment_detail_id
GROUP BY 1
WHERE ....
```



4. Aggregate Level Stats

- Spark will provide some overall values for the min / median / max values.
- These values are for the task duration and data volume .
- They are provided on a per stage / transformation level.
- Focus on the blocks around a join block.

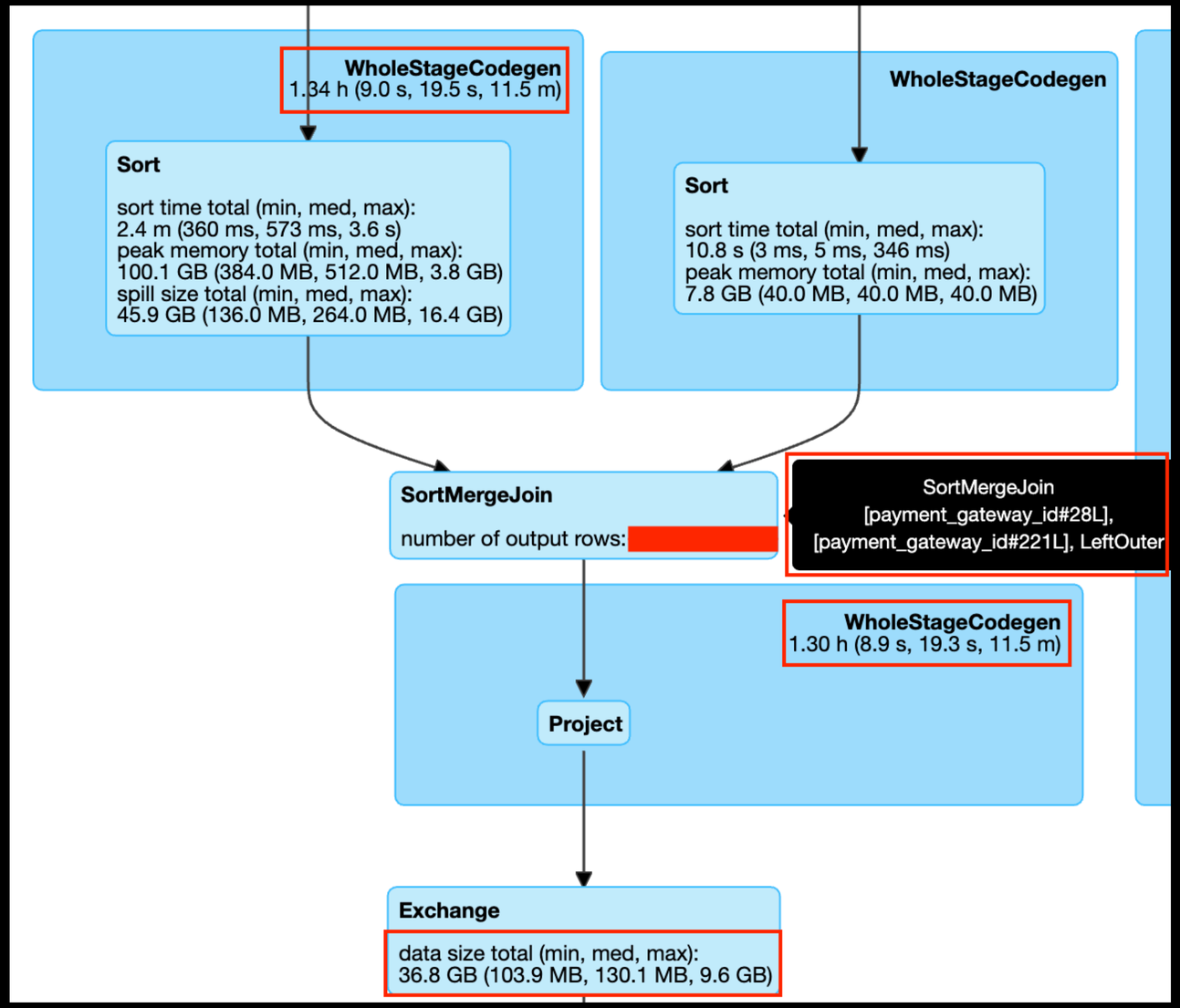
RED FLAGS:

- Highly varying times or data volume.

Solution:

- Skewed join, use a join optimizer.

- 9 seconds vs 11.5m
- Around the join onto payment_gateway_id.



Goal 1

Increase Join Performance. Addressing slow running tasks in a (skewed) join.

Motivation:

When Spark performs a join, spark distributed the data based on the join key to the tasks. If the number of rows per task is not evenly distributed the time it takes per task can vary, thus causing the slower stages to bottleneck the whole application.

Sometimes the skew is so large that the data struggles or can’t fit in memory.

Joins (skew)

2 types of skewed joins:

1. NULL Skewed (only for left joins).
2. Key(s) Skewed (only left and inner joins).

These problems can also appear in our resolvers as well, but they might be implemented differently.

Determining the Skew Type

Steps:

1. Run an analysis (dev skew ...) on the join key.
2. If NULL values make up a majority of the data, data is NULL Skewed.
    1. Replace the join with our NullValueSkewHelper join class.
3. If a single non-null key makes up a large portion of the dataset, Key Skewed.
    - Replace the join with our SkewPartitioner join class.
4. If multiple non-null keys make up a large portion of the dataset, Keys Skewed.
    - Replace the join with our FrequentValuesSkewHelper join class.

Dev Skew

- Calculates the distribution of a provided column in a given dataset.
- Finds the minimal and maximum partition size.

```
dev skew analyze --field 'column' --path
gcs_path
```

*If you have to include legal or source

Dev Skew

- Shows top 5 most frequent values.
- The count and percentage.
- Shows min and max partition data size.

High Frequency Values (Top 5)

Value	Frequency	Percentage
NULL		23.61
4312925		1.18
6993804		0.74
57217927		0.63
94425479		0.58

# Partitions	Minimum Row Count (Percentage, Size)	Maximum Row Count (Percentage, Size)
200	7826449 (0.28, 5.1 GiB)	663687829 (23.91, 433.6 GiB)

NullValueSkewHelper Under the Hood

Motivation:

- Eliminate the shuffle of the null rows.
- Too many rows to fit into memory.

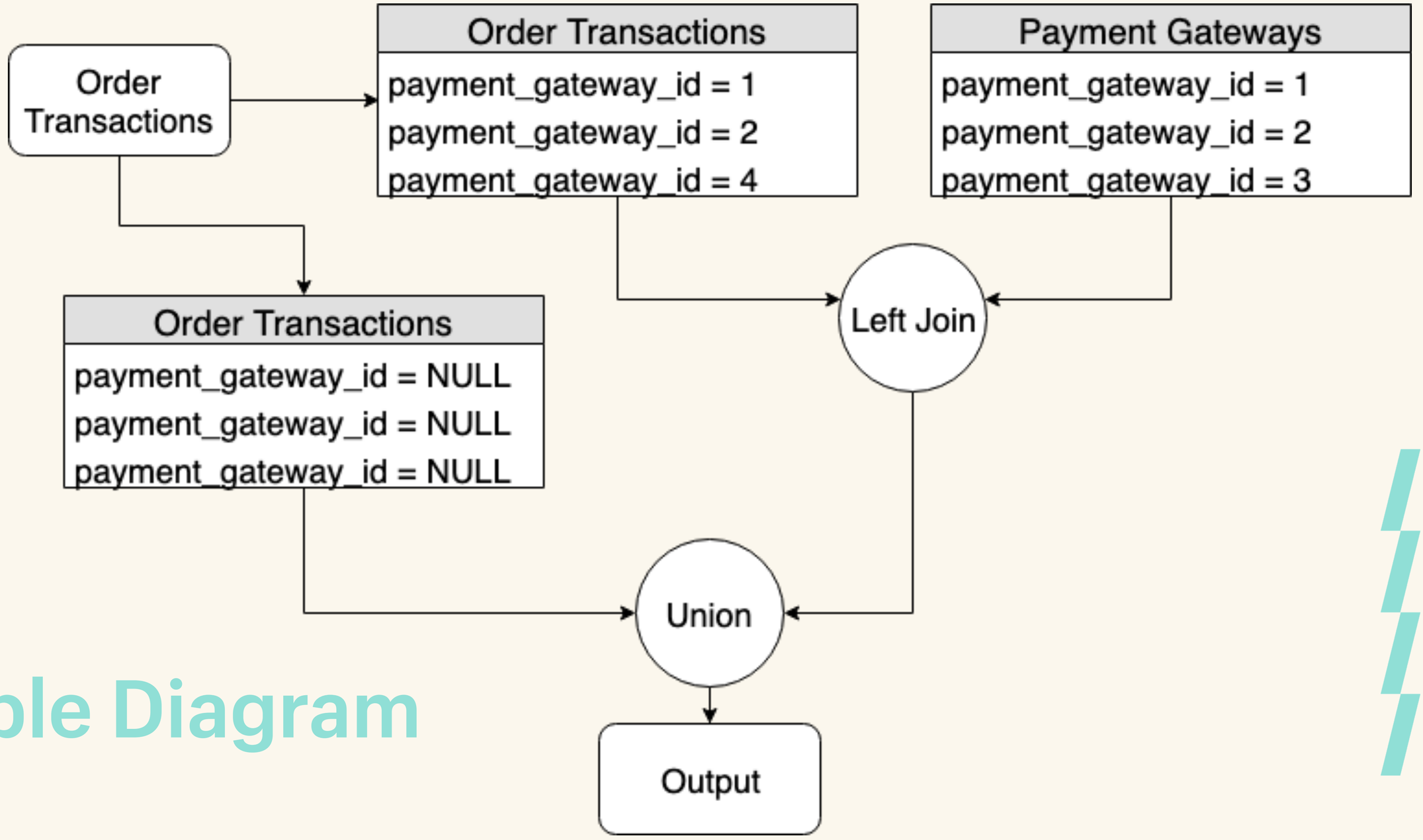
Solution:

- Split the dataset into null and not null.
- Perform the left join on the not null subset.
- Union result with null subset.
- Unions don’t cause a shuffle.

NullValueSkewHelper Under the Hood

- Filter the null skewed dataset to not null values.
- Left joins the to the right table.
- Filter the null skewed dataset to null values.
- Create the right table columns with nulls.
- Unions the joined results with the null values.

Table Diagram



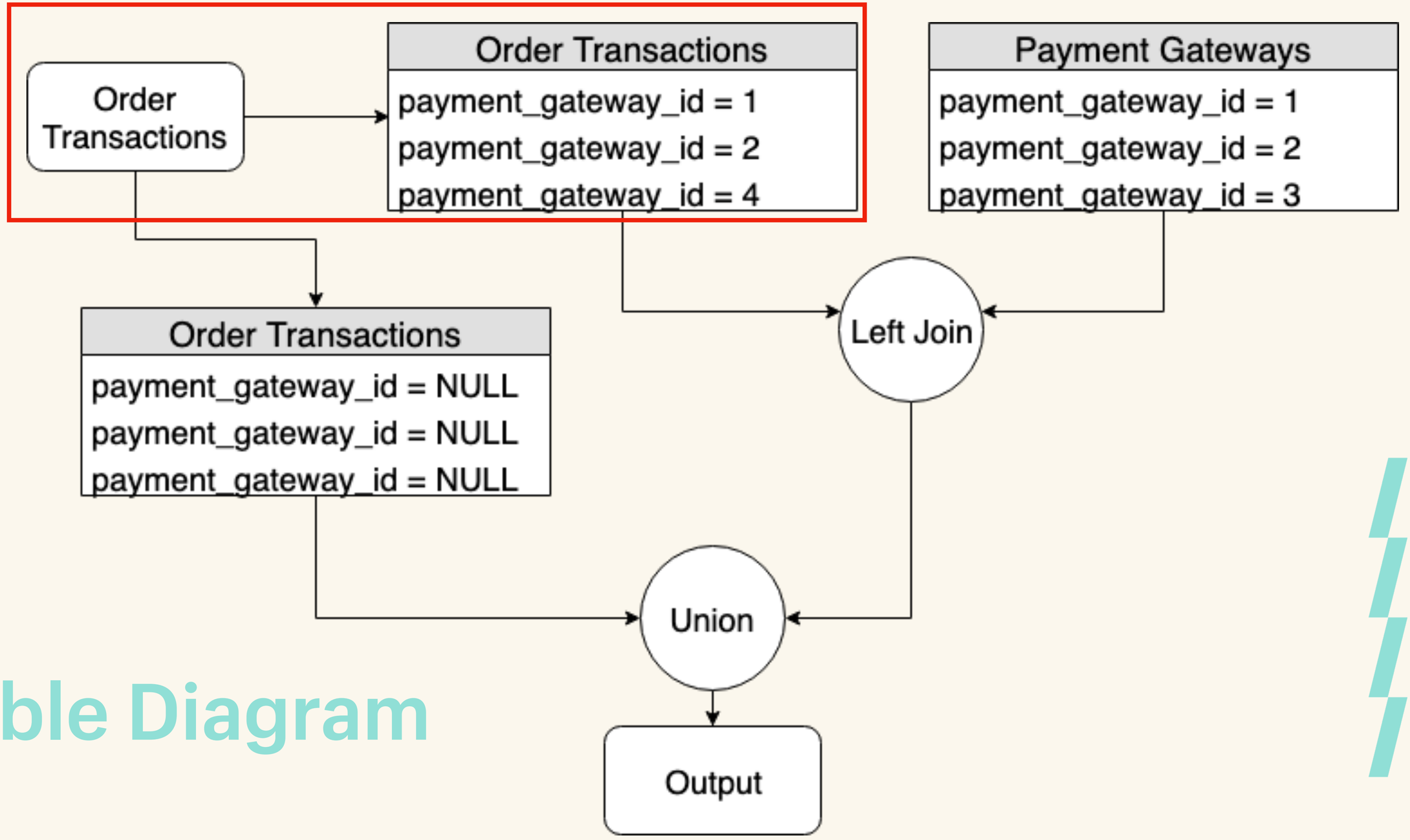


Table Diagram

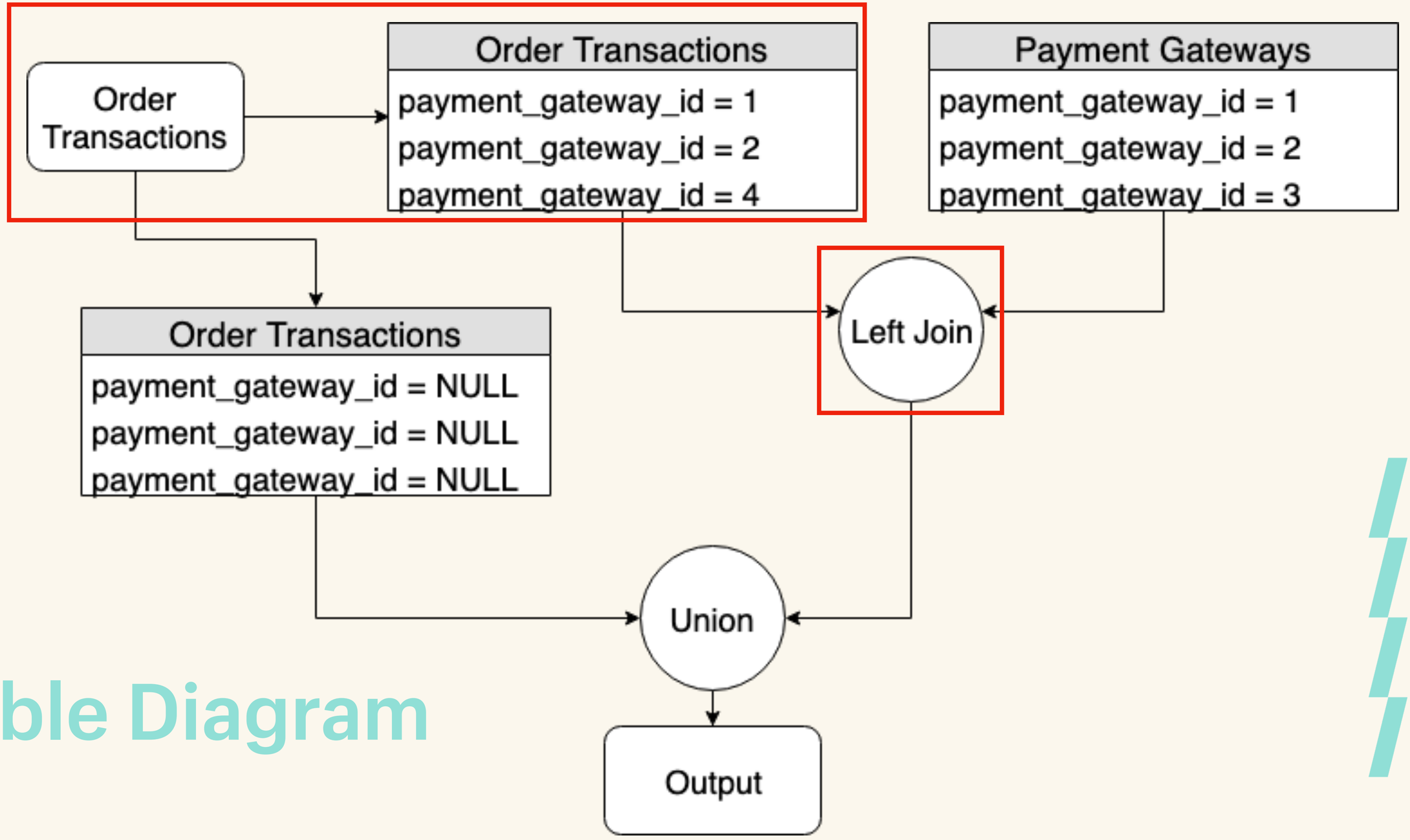


Table Diagram

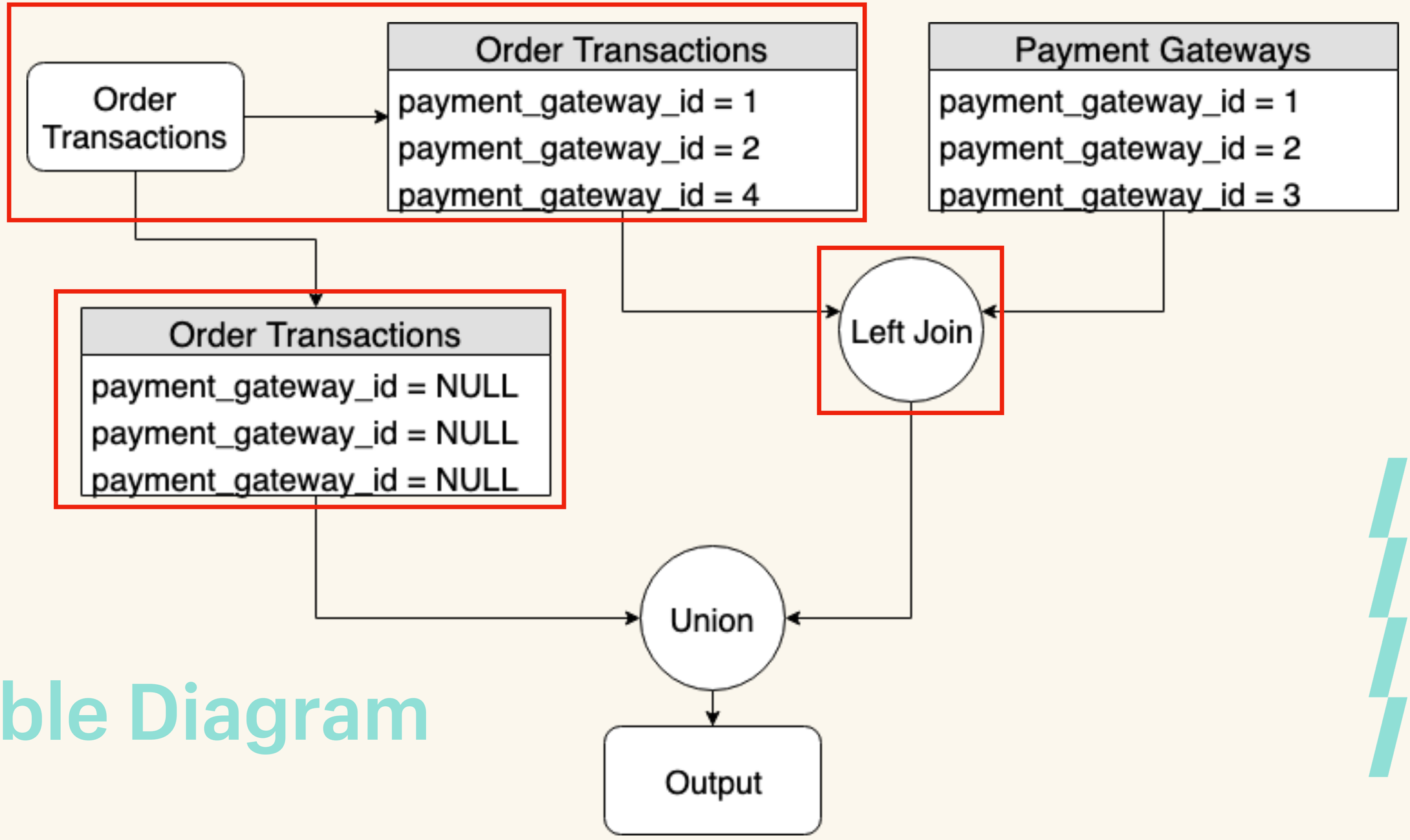


Table Diagram

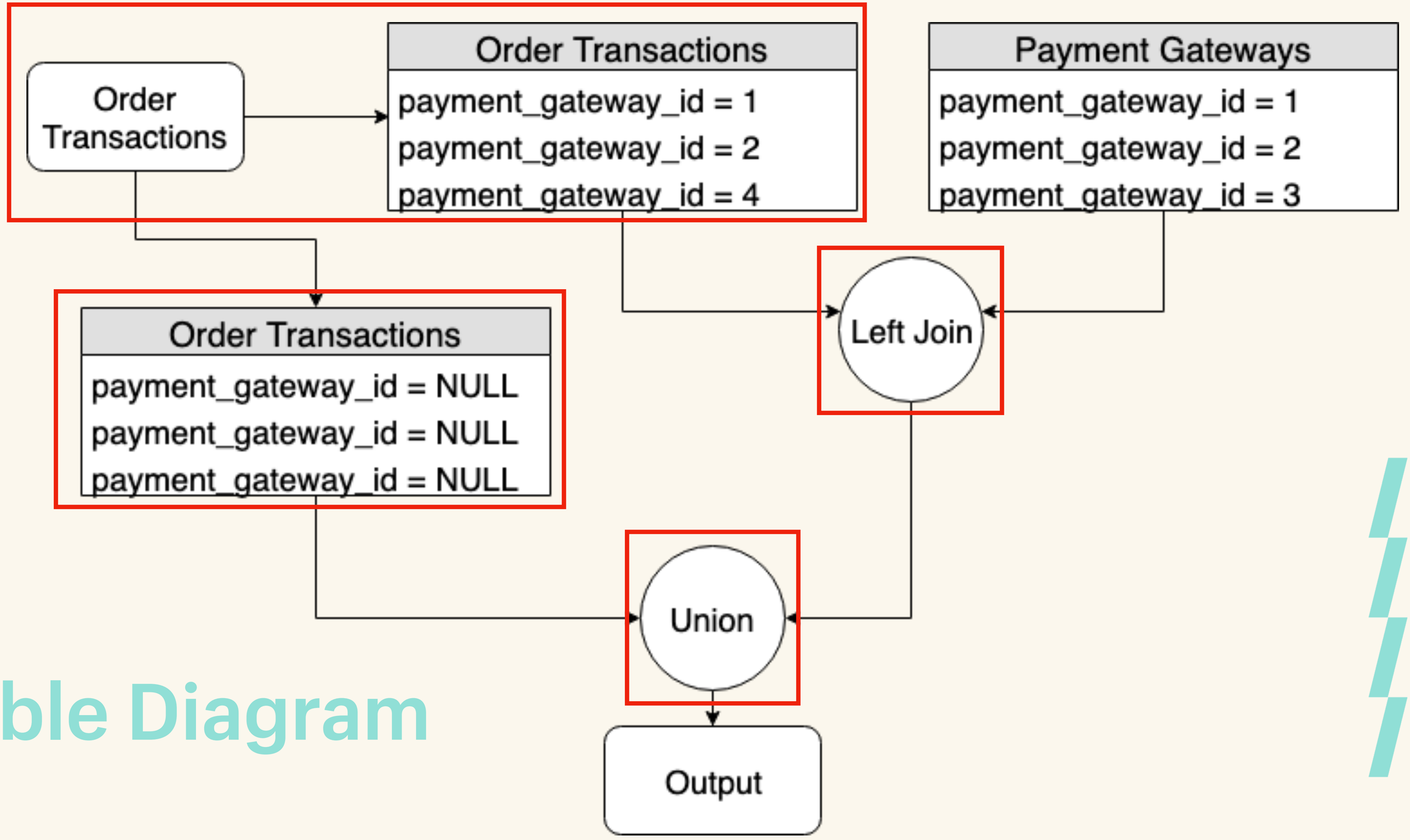
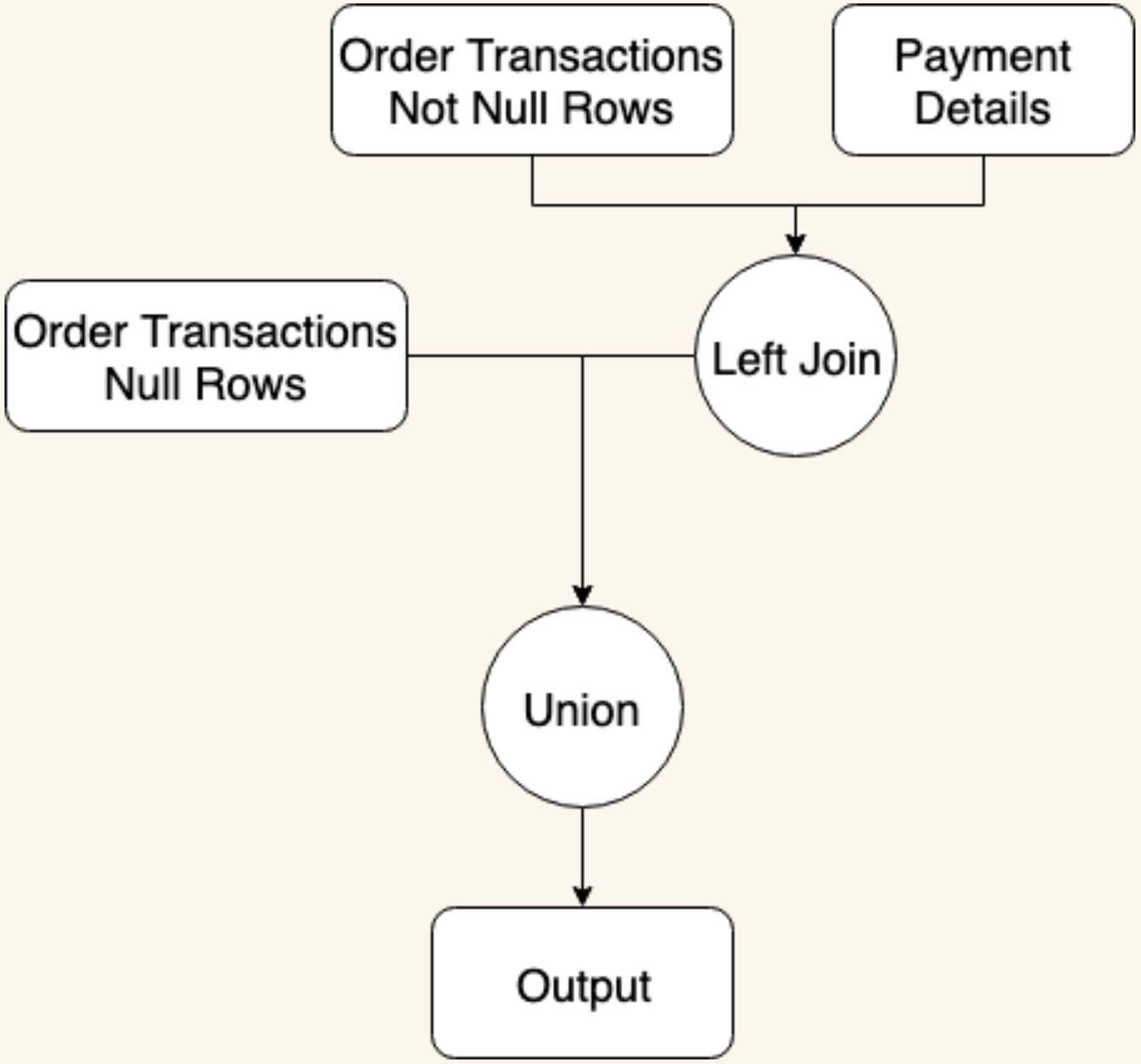
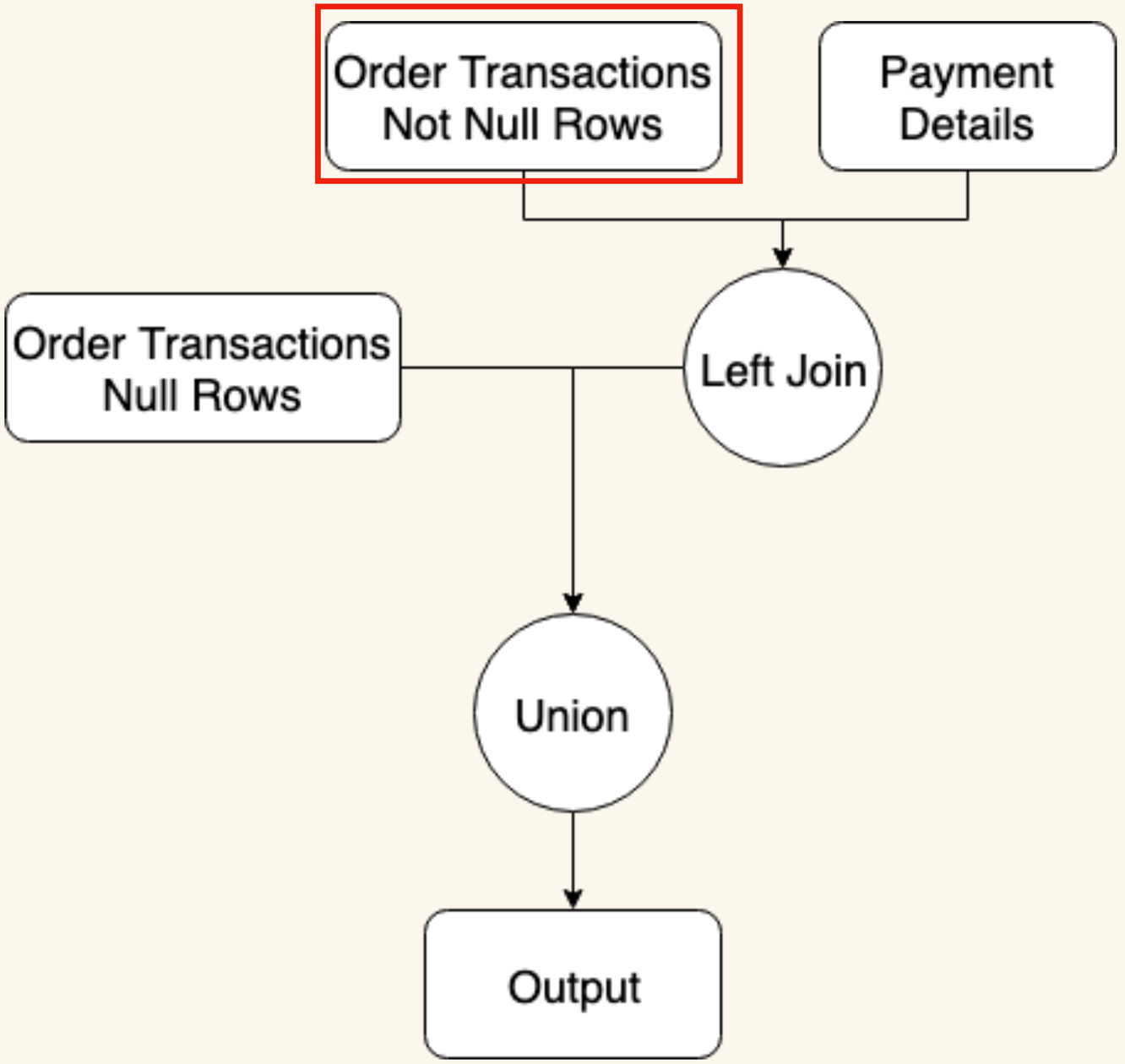


Table Diagram

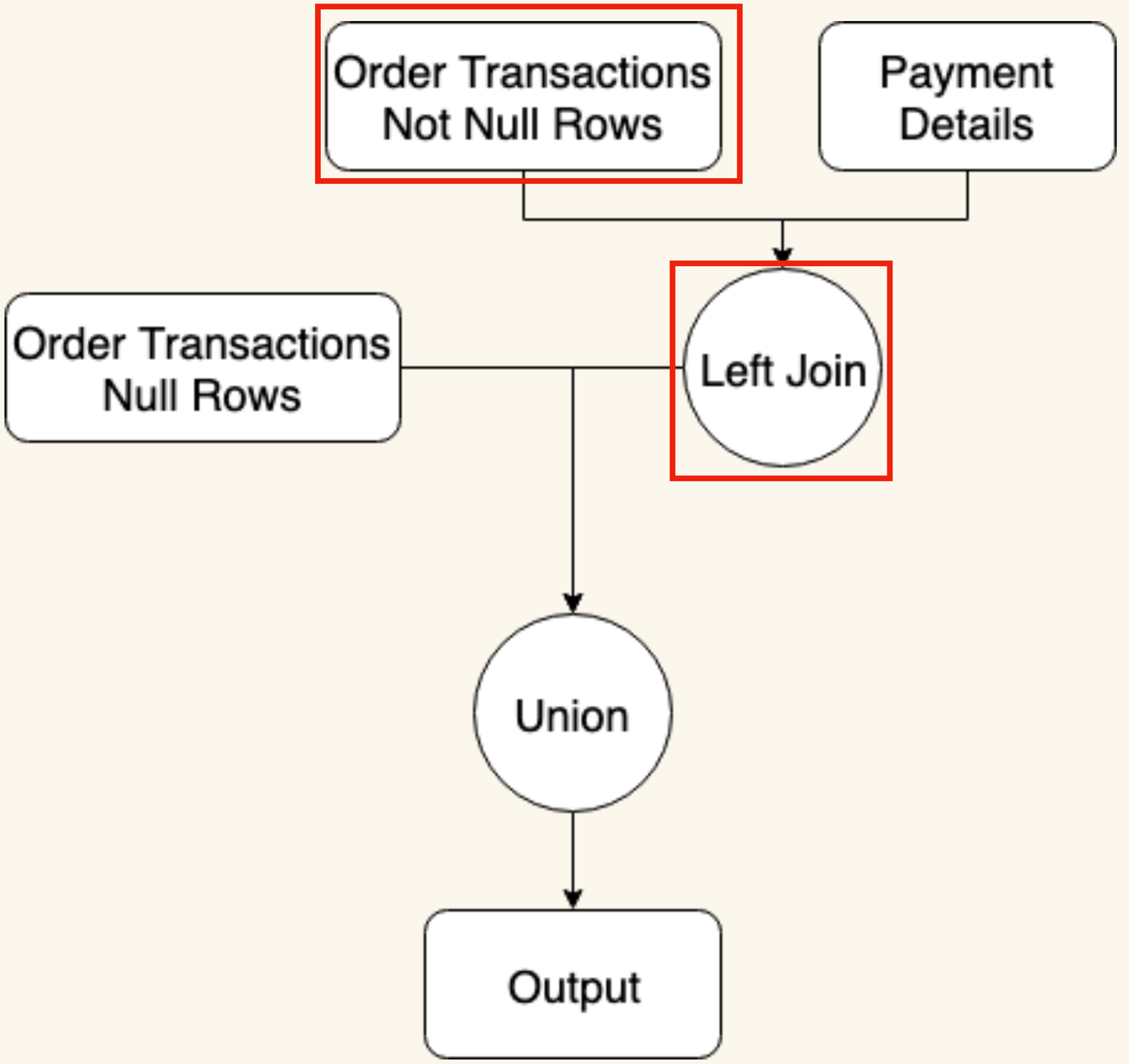
SQL Diagram


Order Transactions
Not Null Rows
Payment
Details
Order Transactions
Null Rows
Left Join
Union
Output

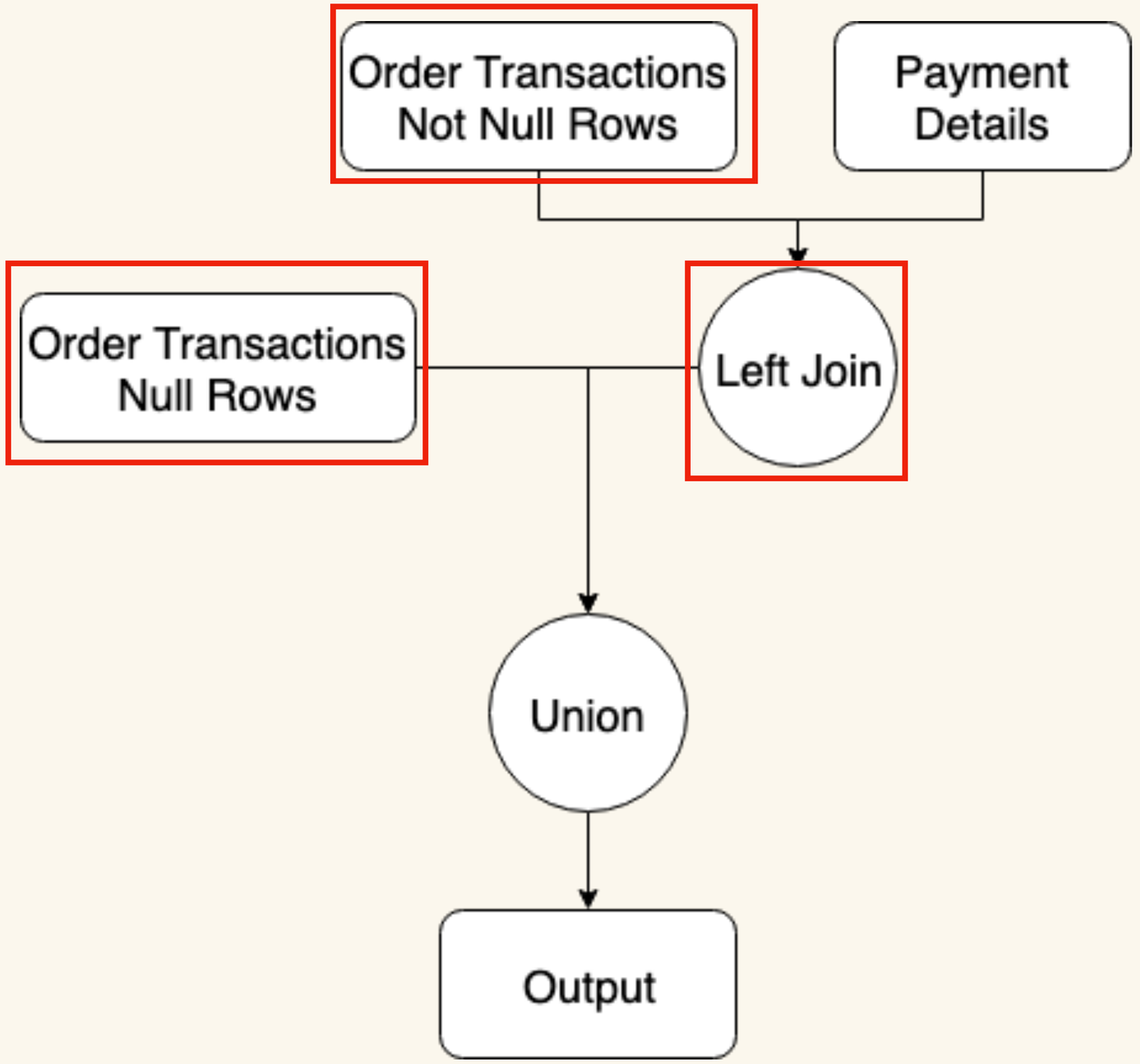
SQL Diagram


Order Transactions
Not Null Rows
Payment
Details
Order Transactions
Null Rows
Left Join
Union
Output

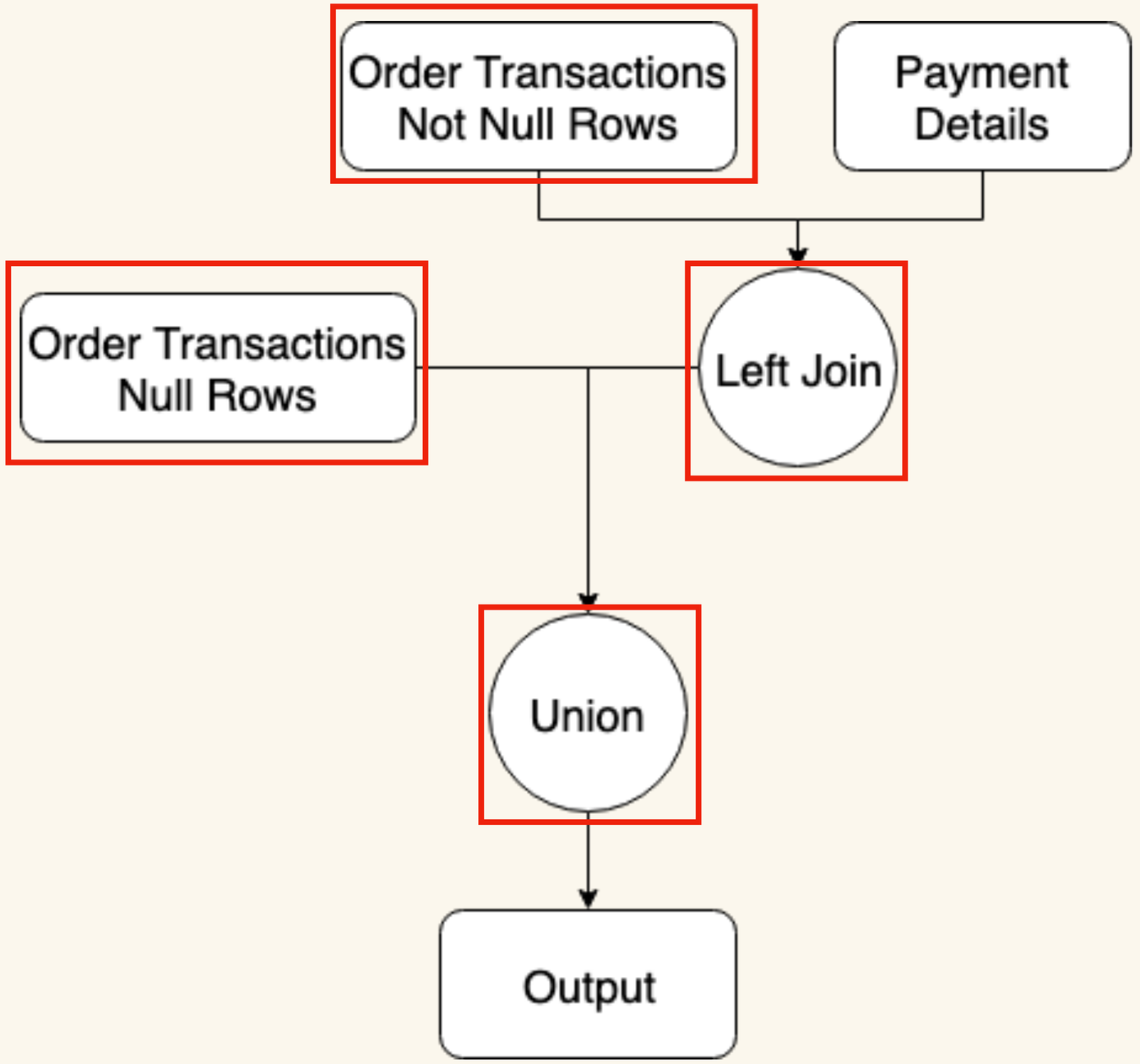
SQL Diagram


Order Transactions
Not Null Rows
Payment
Details
Order Transactions
Null Rows
Left Join
Union
Output

SQL Diagram


Order Transactions
Not Null Rows
Payment
Details
Order Transactions
Null Rows
Left Join
Union
Output

SQL Diagram


Order Transactions
Not Null Rows
Payment
Details
Order Transactions
Null Rows
Left Join
Union
Output

Downsides

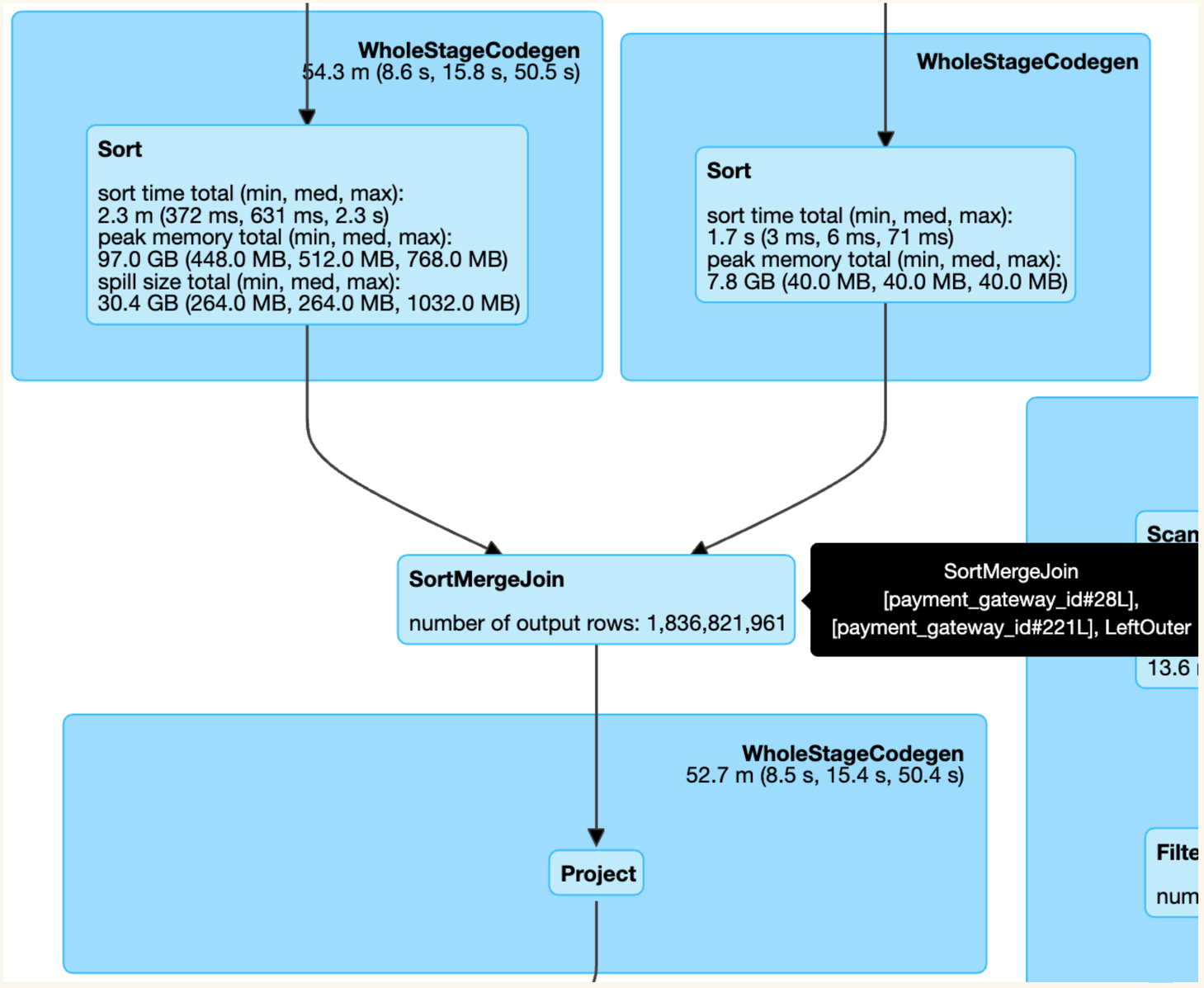
- Spark will recognize that you want to do 2 separate transformations on the skewed dataset:
  - Filter for null values.
  - Filter for not null values.
- Spark will do this read in parallel, so you'll read in twice as much data.
- We recognize this, and "cache"d the skewed dataset, which reduces the read, but now we storing the dataset in memory.
- Thus lowering the amount of total memory for our spark application.
- This can cause memory issues.

Results

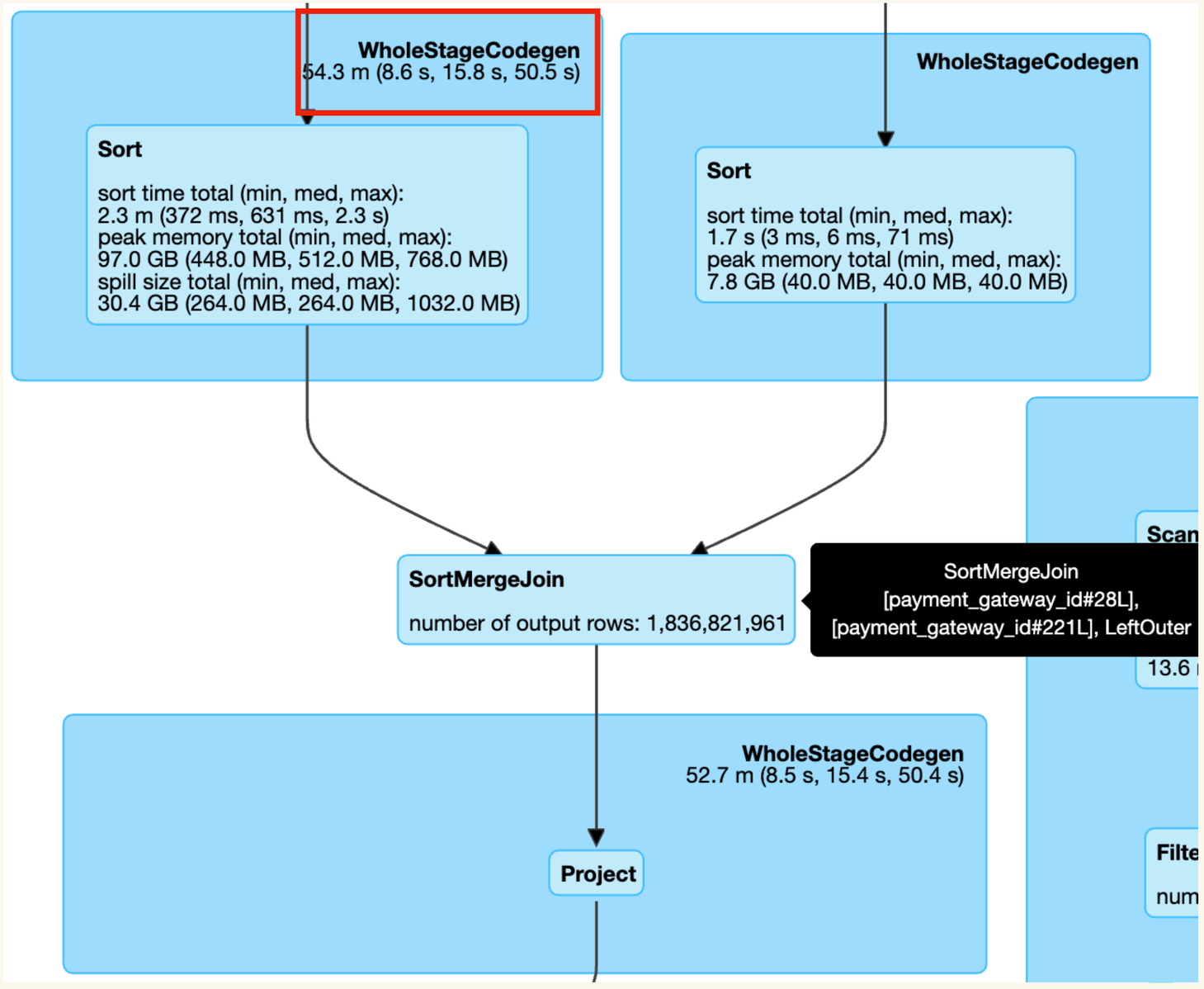
Completed Queries (3)

ID	Description	Submitted ▲	Duration
0	showString at NativeMethodAccessorImpl.java:0	2019/07/24 21:10:15	22 min
1	showString at NativeMethodAccessorImpl.java:0	2019/07/24 21:31:52	10 min

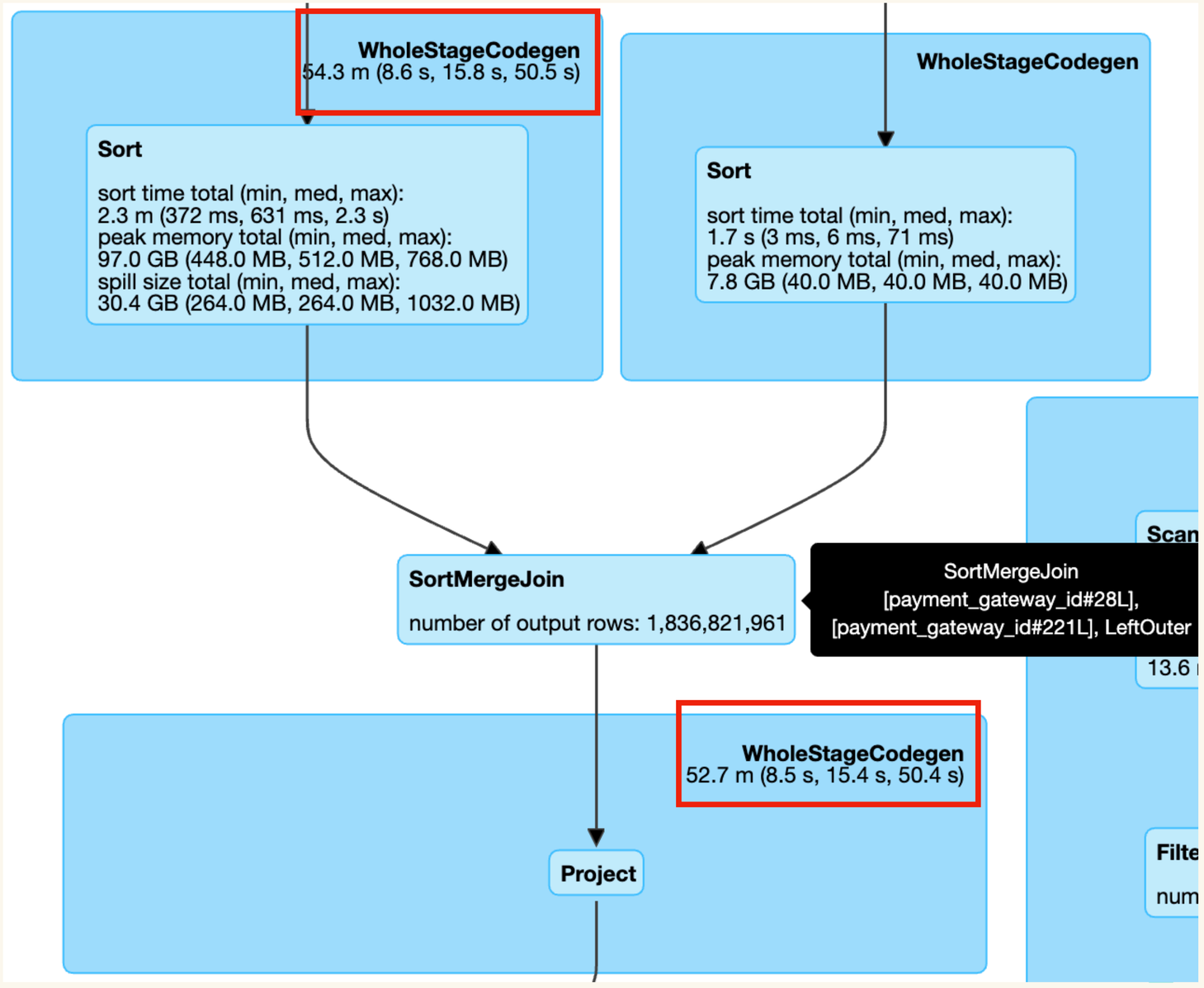
Results


WholeStageCodegen
54.3 m (8.6 s, 15.8 s, 50.5 s)
Sort
sort time total (min, med, max):
2.3 m (372 ms, 631 ms, 2.3 s)
peak memory total (min, med, max):
97.0 GB (448.0 MB, 512.0 MB, 768.0 MB)
spill size total (min, med, max):
30.4 GB (264.0 MB, 264.0 MB, 1032.0 MB)
WholeStageCodegen
Sort
sort time total (min, med, max):
1.7 s (3 ms, 6 ms, 71 ms)
peak memory total (min, med, max):
7.8 GB (40.0 MB, 40.0 MB, 40.0 MB)
SortMergeJoin
number of output rows: 1,836,821,961
SortMergeJoin
[payment_gateway_id#28L],
[payment_gateway_id#221L], LeftOuter
WholeStageCodegen
52.7 m (8.5 s, 15.4 s, 50.4 s)
Project

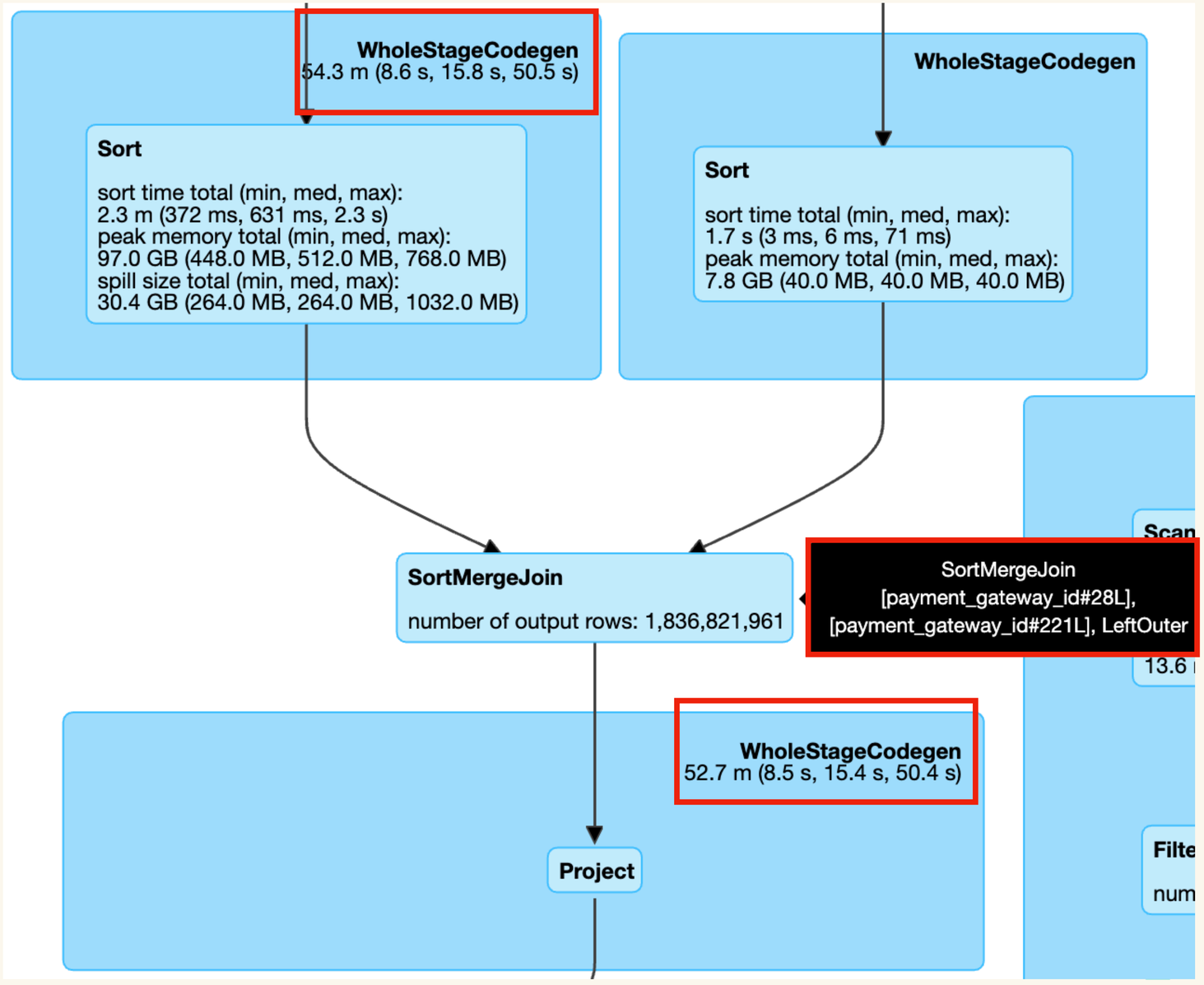
Results


WholeStageCodegen
54.3 m (8.6 s, 15.8 s, 50.5 s)
Sort
sort time total (min, med, max):
2.3 m (372 ms, 631 ms, 2.3 s)
peak memory total (min, med, max):
97.0 GB (448.0 MB, 512.0 MB, 768.0 MB)
spill size total (min, med, max):
30.4 GB (264.0 MB, 264.0 MB, 1032.0 MB)
WholeStageCodegen
Sort
sort time total (min, med, max):
1.7 s (3 ms, 6 ms, 71 ms)
peak memory total (min, med, max):
7.8 GB (40.0 MB, 40.0 MB, 40.0 MB)
SortMergeJoin
number of output rows: 1,836,821,961
SortMergeJoin
[payment_gateway_id#28L],
[payment_gateway_id#221L], LeftOuter
WholeStageCodegen
52.7 m (8.5 s, 15.4 s, 50.4 s)
Project

Results


WholeStageCodegen
54.3 m (8.6 s, 15.8 s, 50.5 s)
Sort
sort time total (min, med, max):
2.3 m (372 ms, 631 ms, 2.3 s)
peak memory total (min, med, max):
97.0 GB (448.0 MB, 512.0 MB, 768.0 MB)
spill size total (min, med, max):
30.4 GB (264.0 MB, 264.0 MB, 1032.0 MB)
WholeStageCodegen
Sort
sort time total (min, med, max):
1.7 s (3 ms, 6 ms, 71 ms)
peak memory total (min, med, max):
7.8 GB (40.0 MB, 40.0 MB, 40.0 MB)
SortMergeJoin
number of output rows: 1,836,821,961
SortMergeJoin
[payment_gateway_id#28L],
[payment_gateway_id#221L], LeftOuter
WholeStageCodegen
52.7 m (8.5 s, 15.4 s, 50.4 s)
Project

Results


WholeStageCodegen
54.3 m (8.6 s, 15.8 s, 50.5 s)
Sort
sort time total (min, med, max):
2.3 m (372 ms, 631 ms, 2.3 s)
peak memory total (min, med, max):
97.0 GB (448.0 MB, 512.0 MB, 768.0 MB)
spill size total (min, med, max):
30.4 GB (264.0 MB, 264.0 MB, 1032.0 MB)
WholeStageCodegen
Sort
sort time total (min, med, max):
1.7 s (3 ms, 6 ms, 71 ms)
peak memory total (min, med, max):
7.8 GB (40.0 MB, 40.0 MB, 40.0 MB)
SortMergeJoin
number of output rows: 1,836,821,961
SortMergeJoin
[payment_gateway_id#28L],
[payment_gateway_id#221L], LeftOuter
WholeStageCodegen
52.7 m (8.5 s, 15.4 s, 50.4 s)
Project

SkewPartitioner

Motivation:

- Reduce the amount of rows that contain the skewed key.

Solution:

- We randomly assign a number from 0 to n - 1, called a “salt”, to each row.
- This splits the number of rows by a factor n.

SkewPartitioner Under the Hood

- Assigns a “salt” number (random % num_partitions) to each row of the left dataFrame.
- Duplicates the right dataFrame num_partitions times.
- Assign a different salt value to the every duplicate.
- Joins the two datasets on the (join key + salt number).

Table Diagram

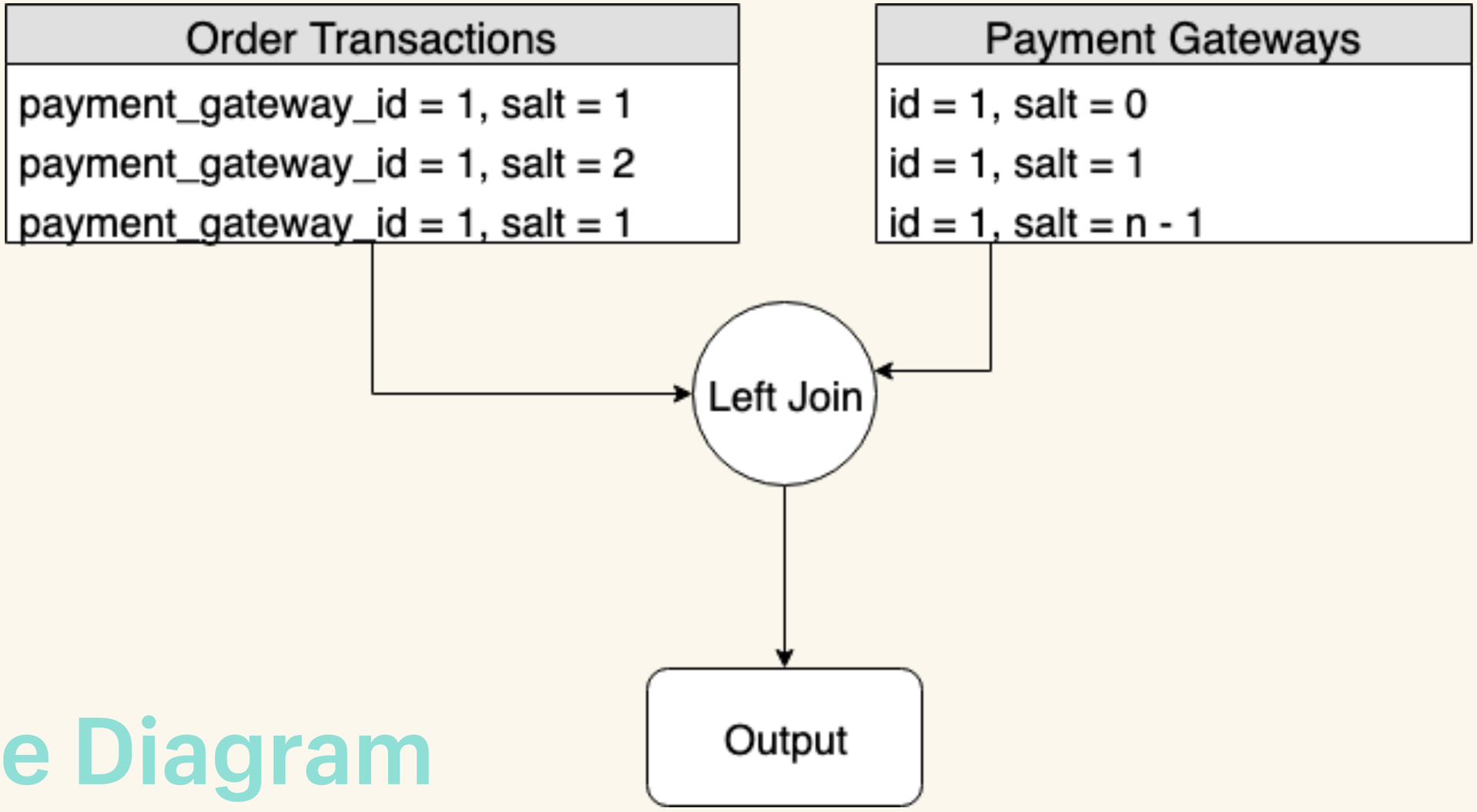
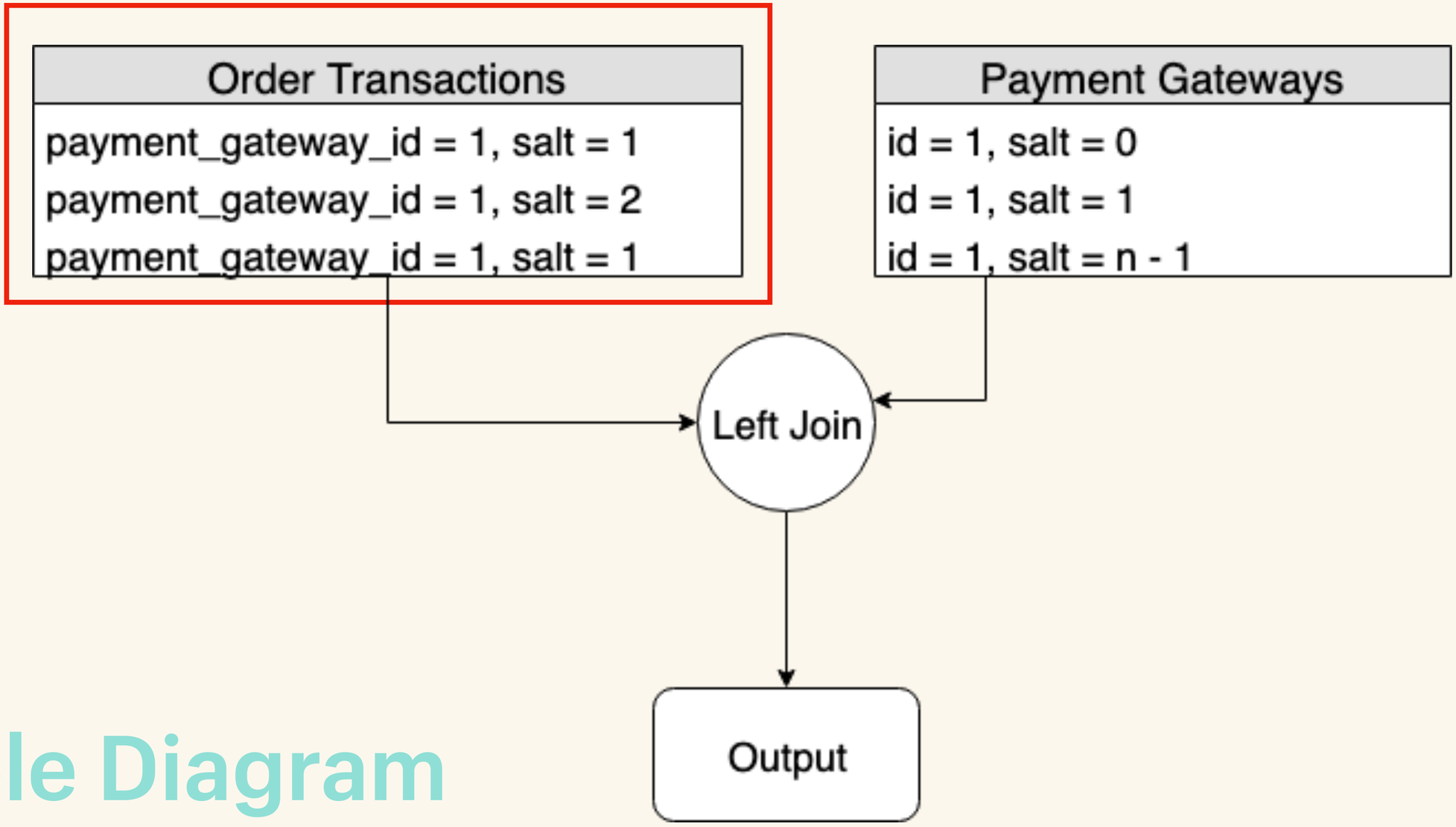


Table Diagram



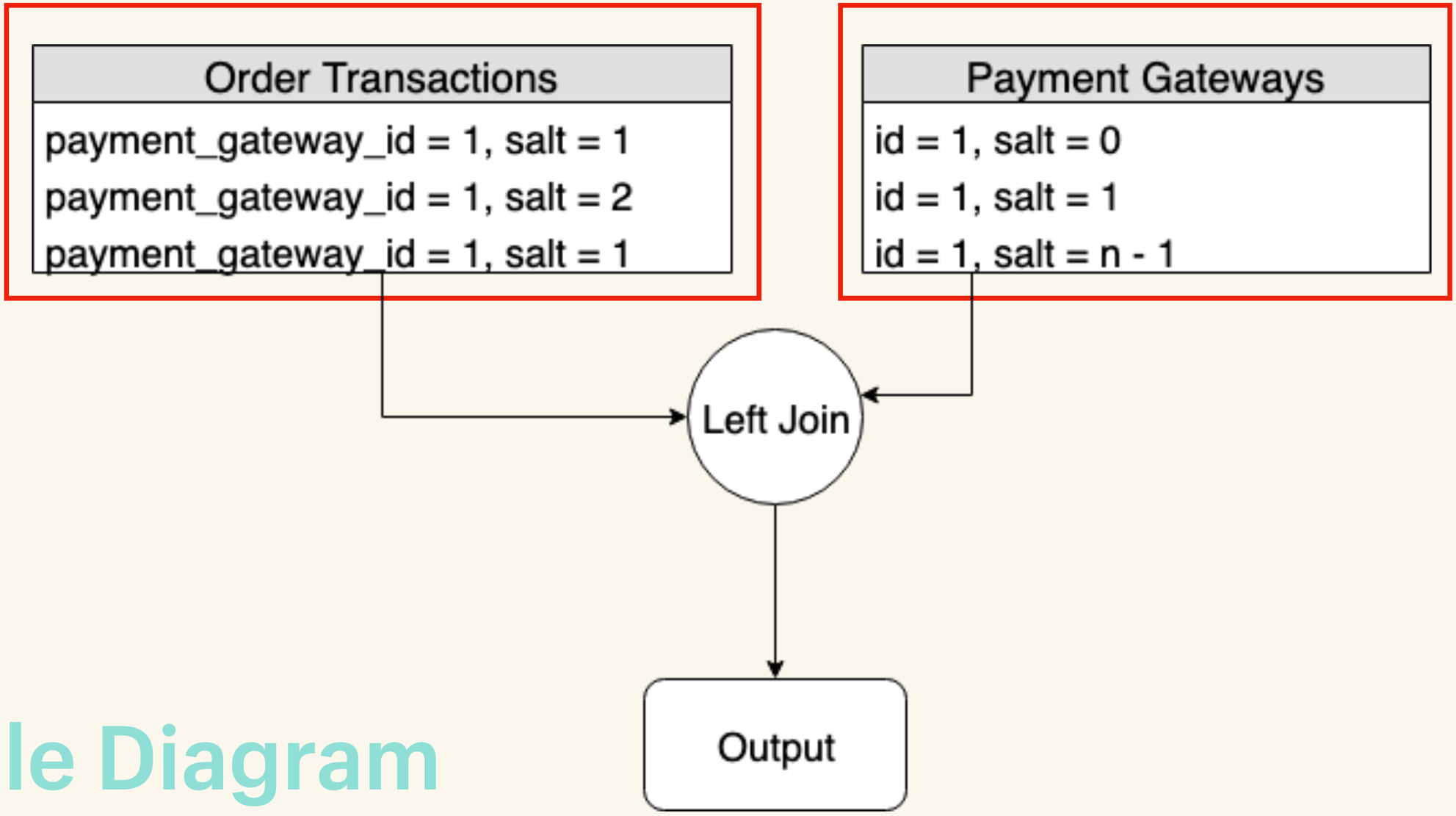
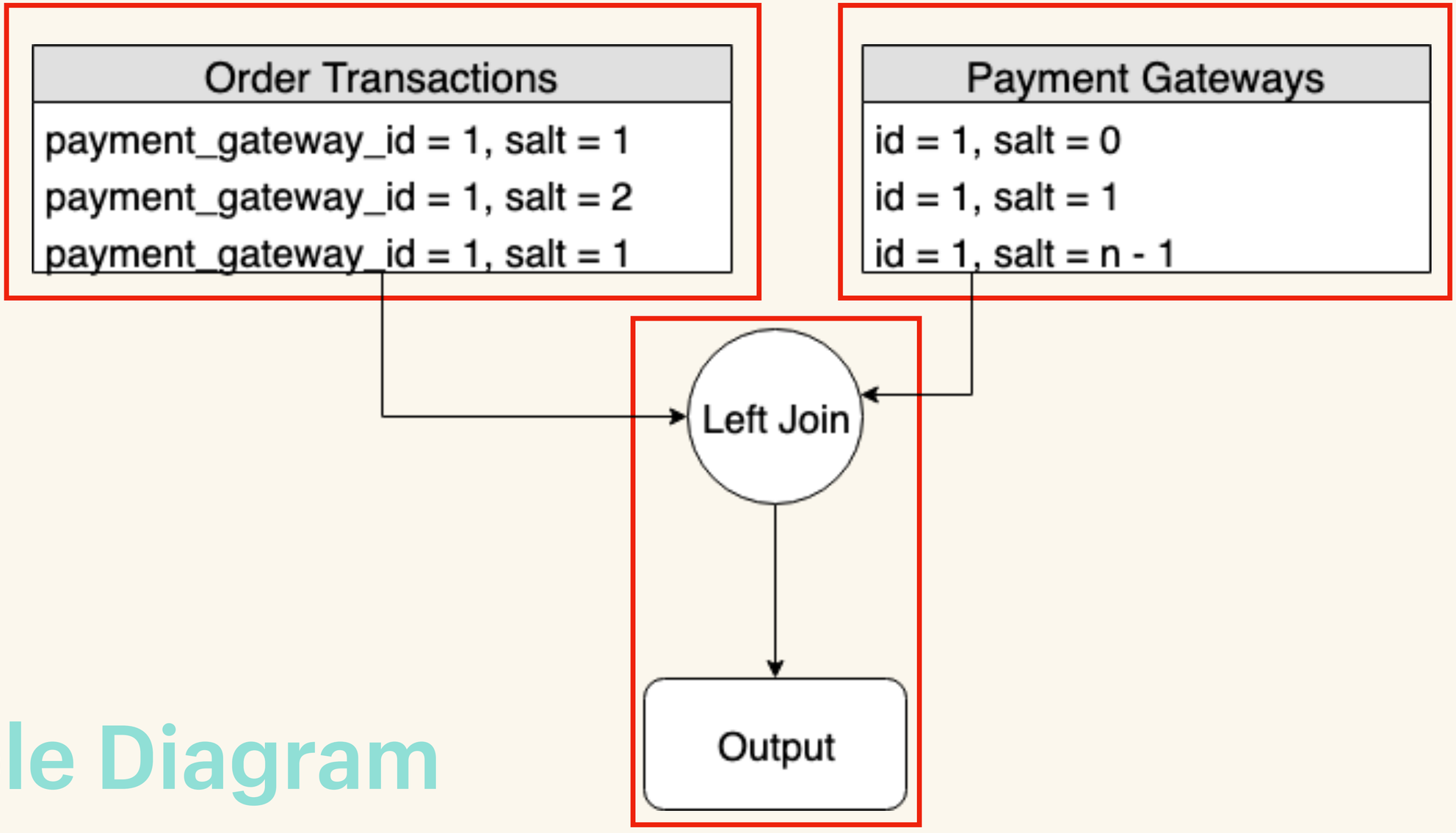
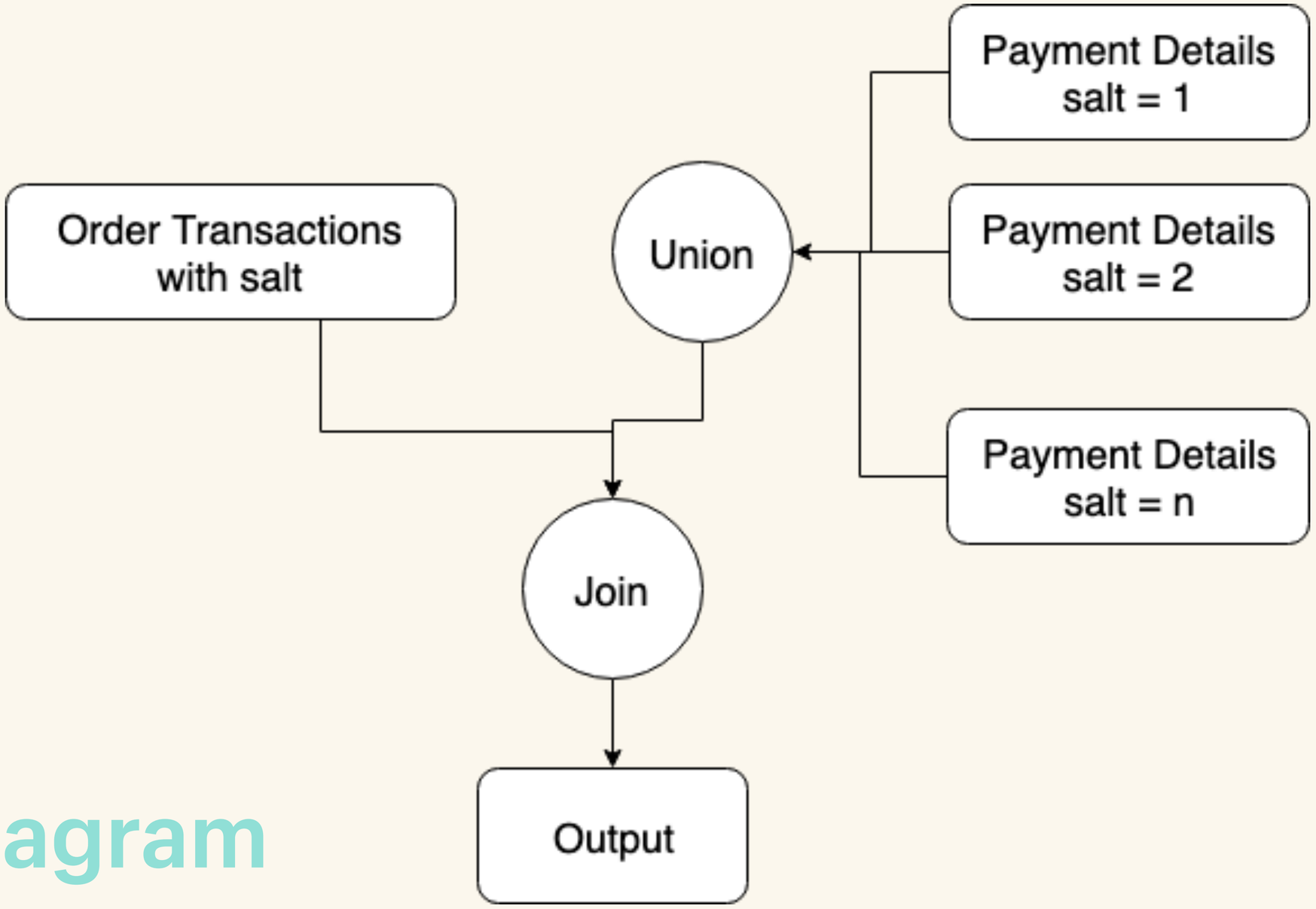


Table Diagram

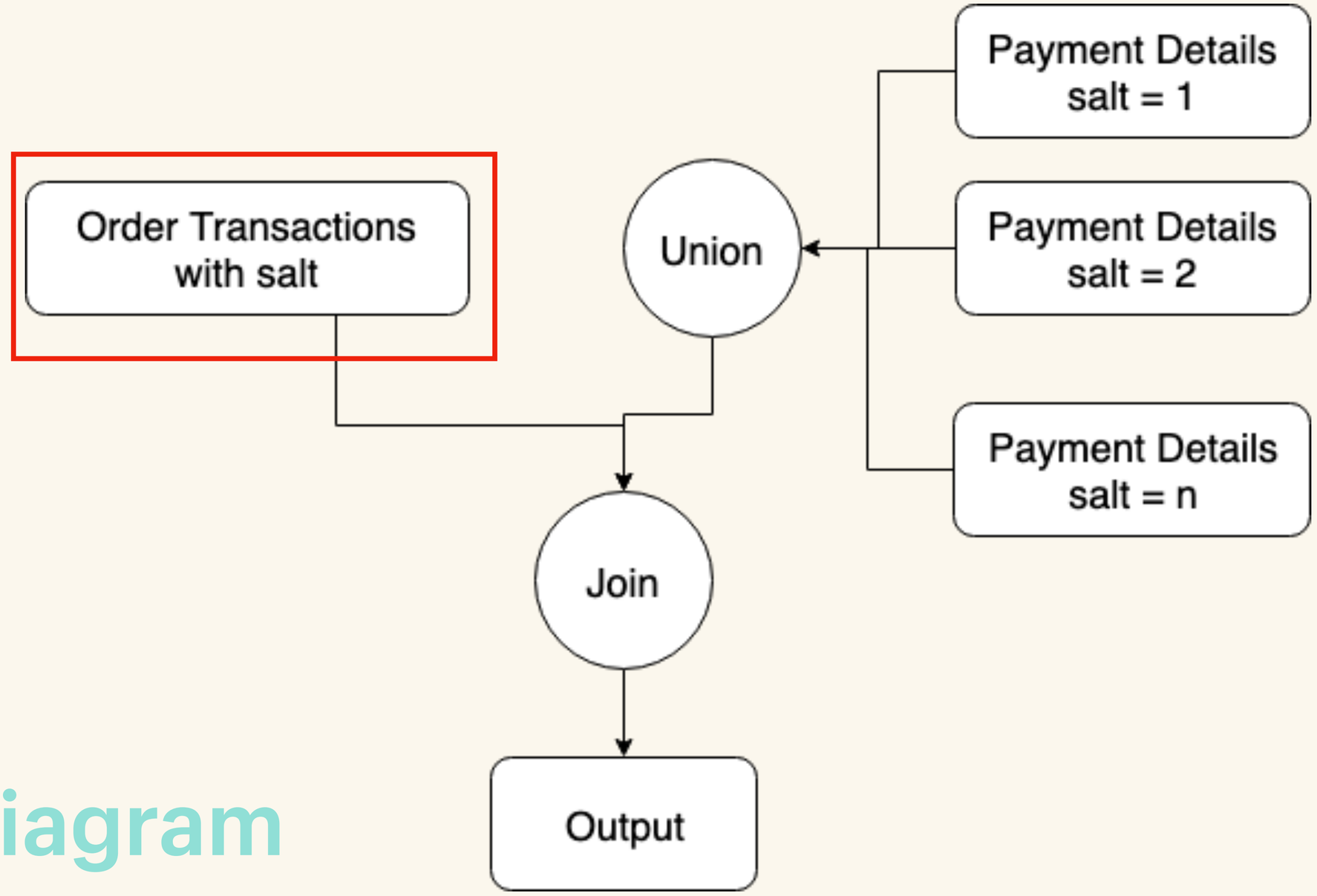
Table Diagram



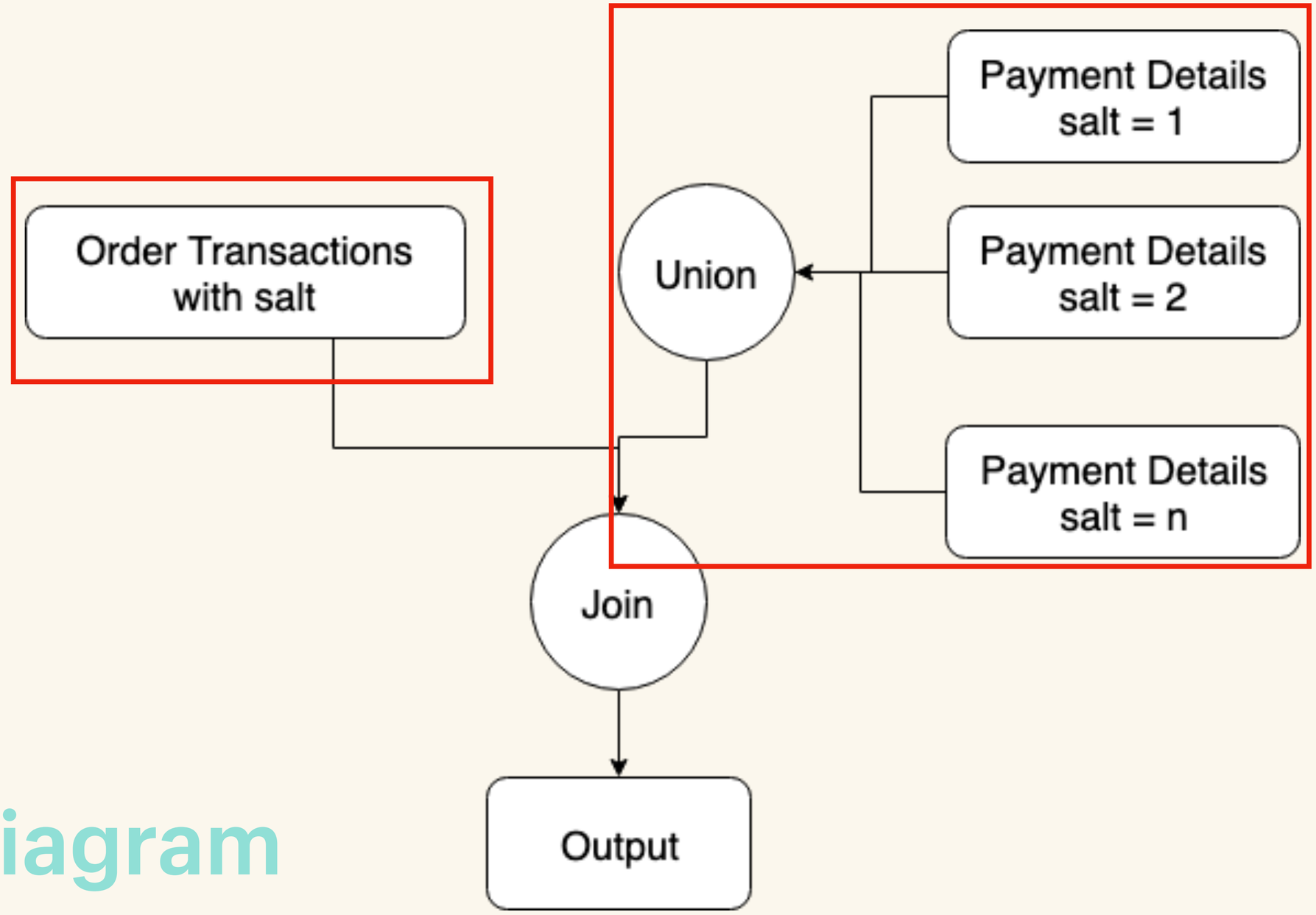
SQL Diagram


Payment Details
salt = 1
Order Transactions
with salt
Union
Payment Details
salt = 2
Payment Details
salt = n
Join
Output

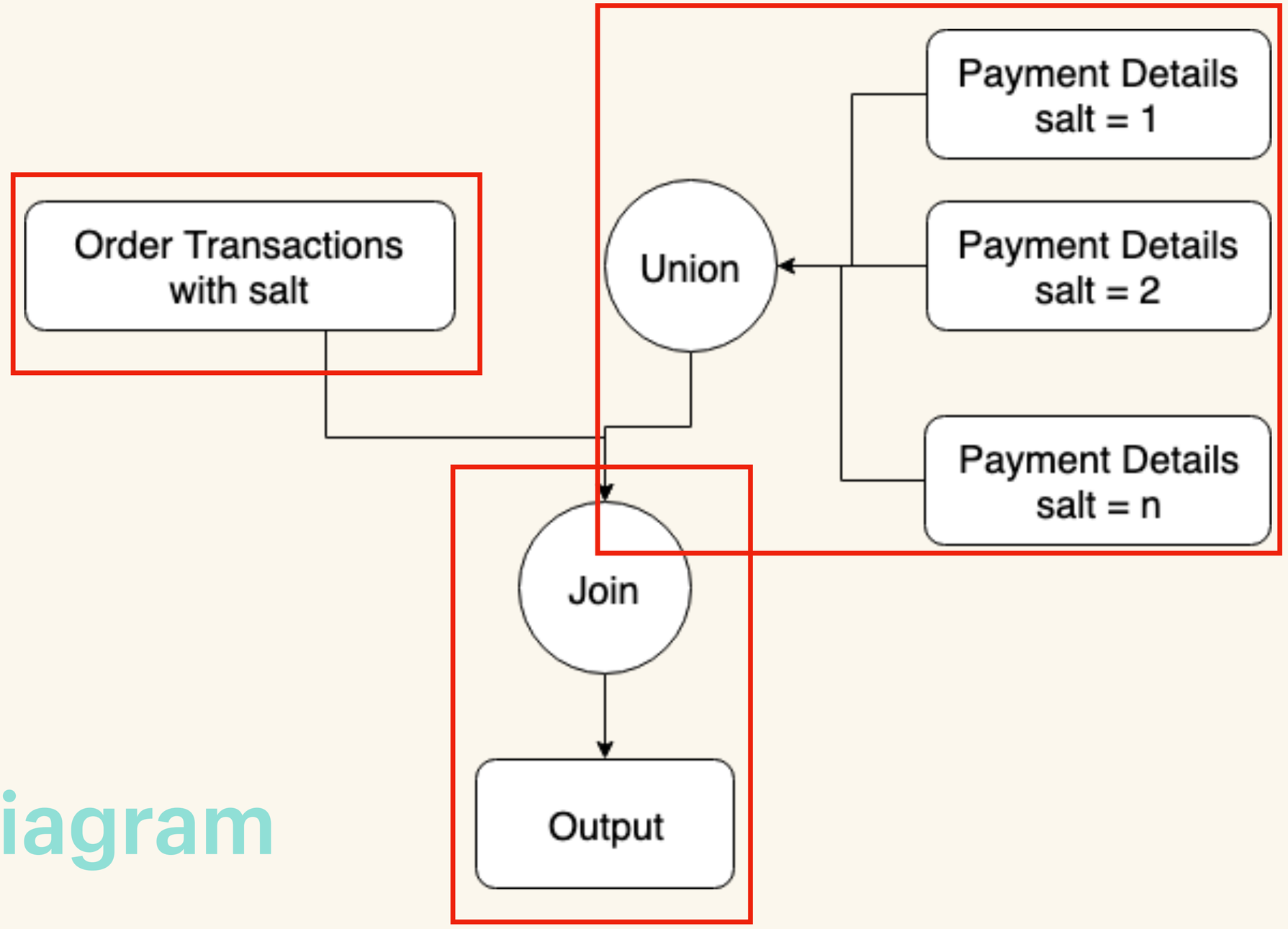
SQL Diagram


Payment Details
salt = 1
Order Transactions
with salt
Union
Payment Details
salt = 2
Payment Details
salt = n
Join
Output

SQL Diagram


Payment Details
salt = 1
Order Transactions
with salt
Union
Payment Details
salt = 2
Payment Details
salt = n
Join
Output

SQL Diagram


Payment Details
salt = 1
Order Transactions
with salt
Union
Payment Details
salt = 2
Payment Details
salt = n
Join
Output

Downsides

- The right side of the join (the dimension) grows linearly with n.
- Depending how large the right side is, this can potentially be bad.

FrequentValuesSkewHelper

Motivation:

- Eliminate the shuffle of rows containing the subset of skewed keys.
- These rows are too big to fit onto tasks.

FrequentValuesSkewHelper

Solution:

- Split the skewed dataset into 2 subsets.
- First subset contains only skewed keys.
- Second contains non skewed keys.
- Do same with right dataset.
- Perform a regular join with non skewed datasets.
- Do a broadcast join on skewed dataset.
- Then union that with the joined result.
- Broadcast join and union do not produce a shuffle. **

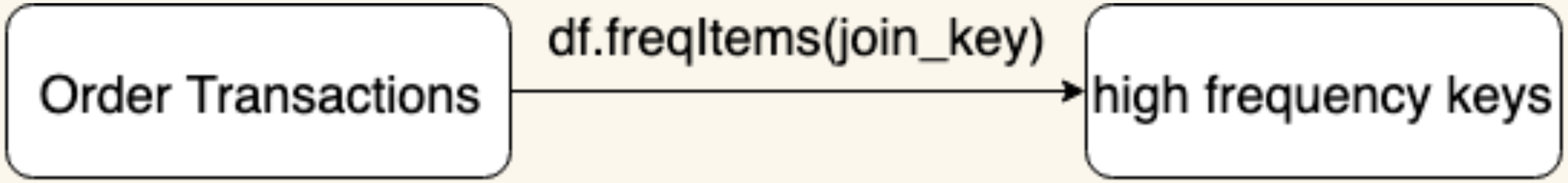
We can broadcast cause only a small subset (under broadcast range) of keys are considered skewed.

FrequentValuesSkewHelper Under the Hood

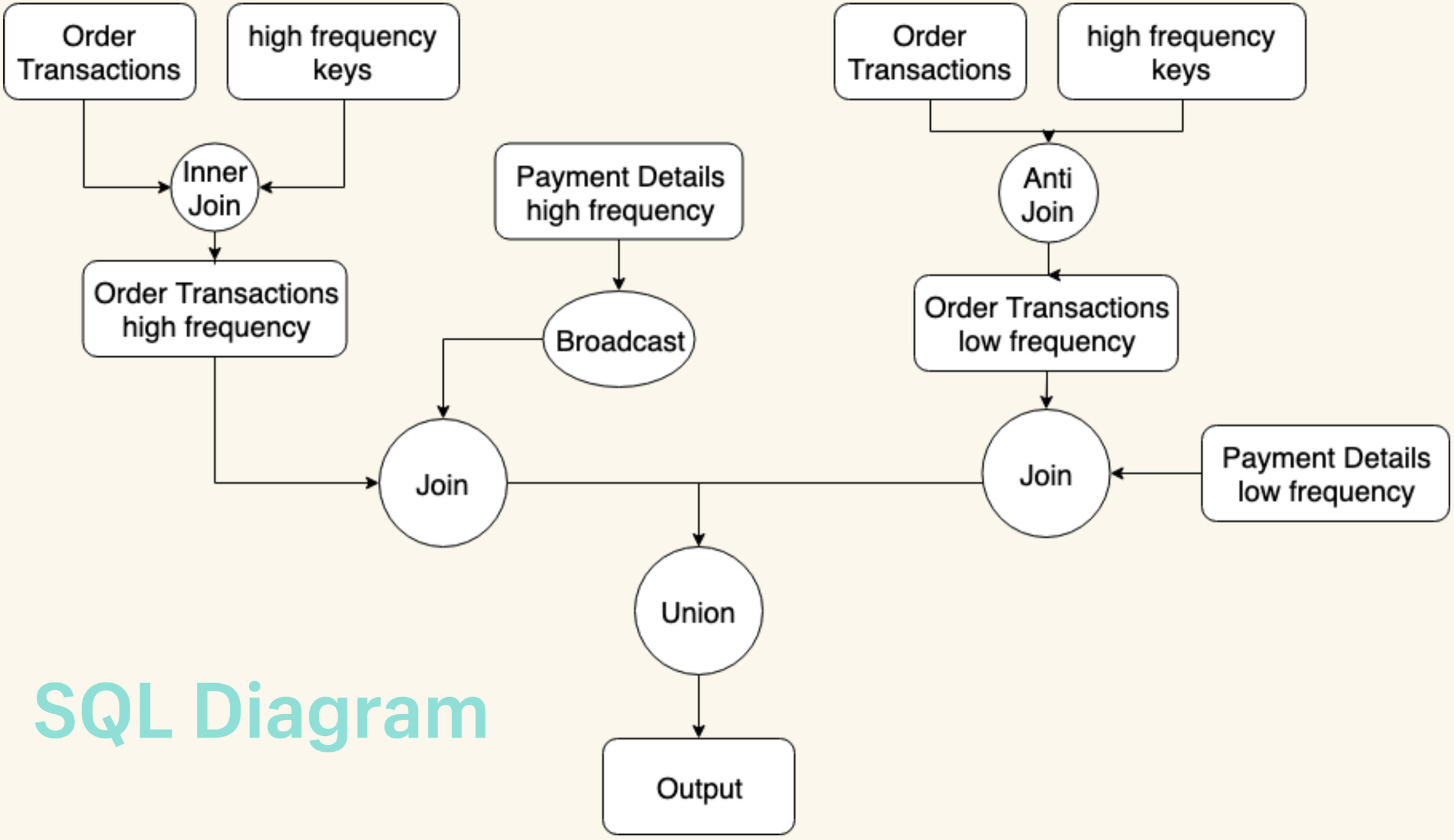
- Calculates the keys that make up a large portion of the left dataset.
- Splits left and right dataset into 2 parts:
    1. All the rows that make up a large portion of the dataset, call this the “high frequency” DataFrame.
    2. Everything else, call this the “low frequency” DataFrame.
- Broadcast the high frequency right side, this eliminates shuffle of left high frequency dataframe.
- Performs the join on both the “high frequency” Dataframes.
- Performs the join on both the “low frequency” Dataframes.
- Unions the two joined datasets together.

Table Diagram

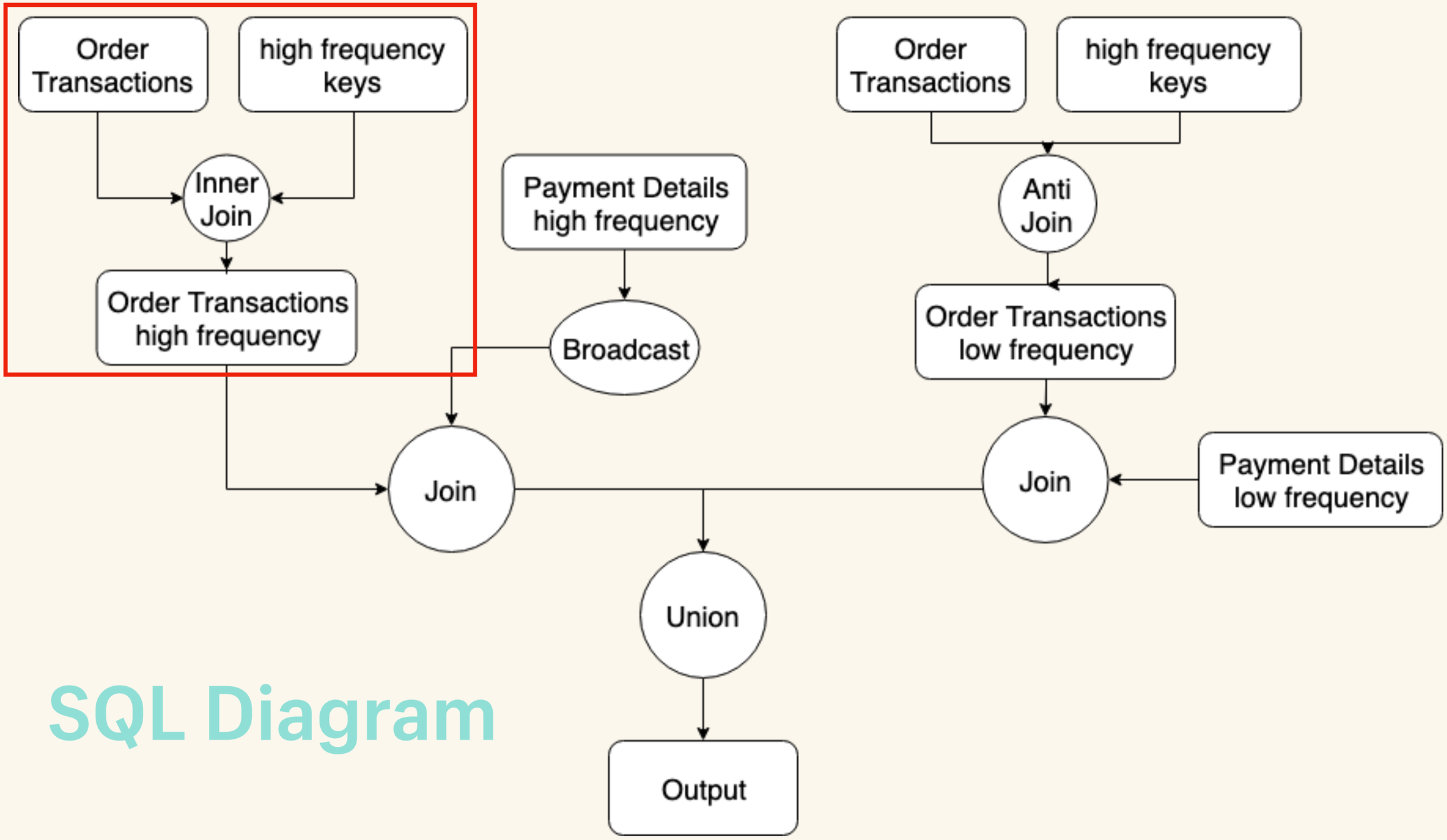
Table Diagram


Order Transactions
df.freqItems(join_key)
high frequency keys

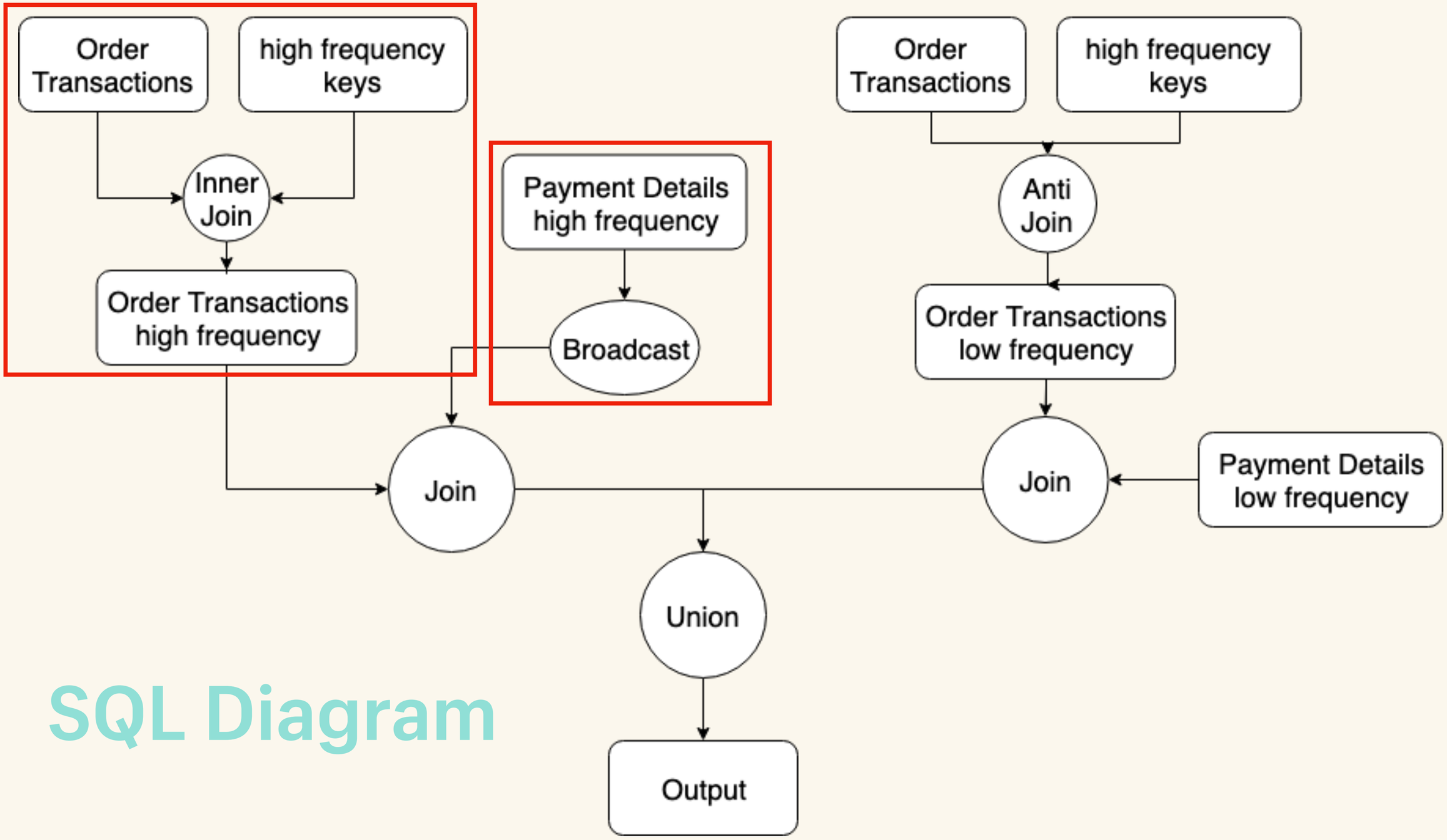
SQL Diagram


Order Transactions
high frequency keys
Inner Join
Order Transactions high frequency
Payment Details high frequency
Broadcast
Order Transactions
high frequency keys
Anti Join
Order Transactions low frequency
Join
Join
Payment Details low frequency
Union
Output

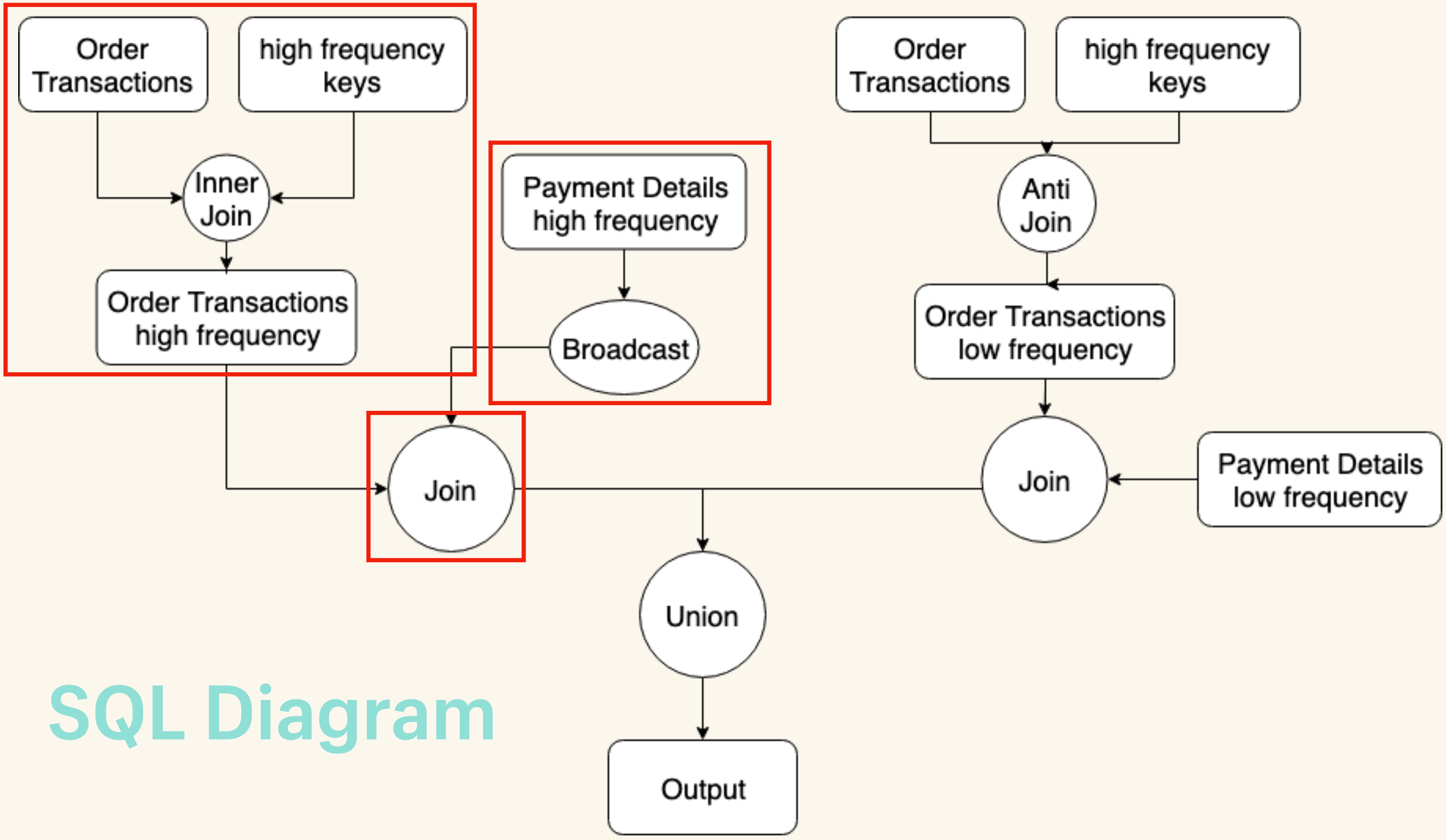
SQL Diagram


Order Transactions
high frequency keys
Inner Join
Order Transactions high frequency
Payment Details high frequency
Broadcast
Order Transactions
high frequency keys
Anti Join
Order Transactions low frequency
Join
Join
Payment Details low frequency
Union
Output

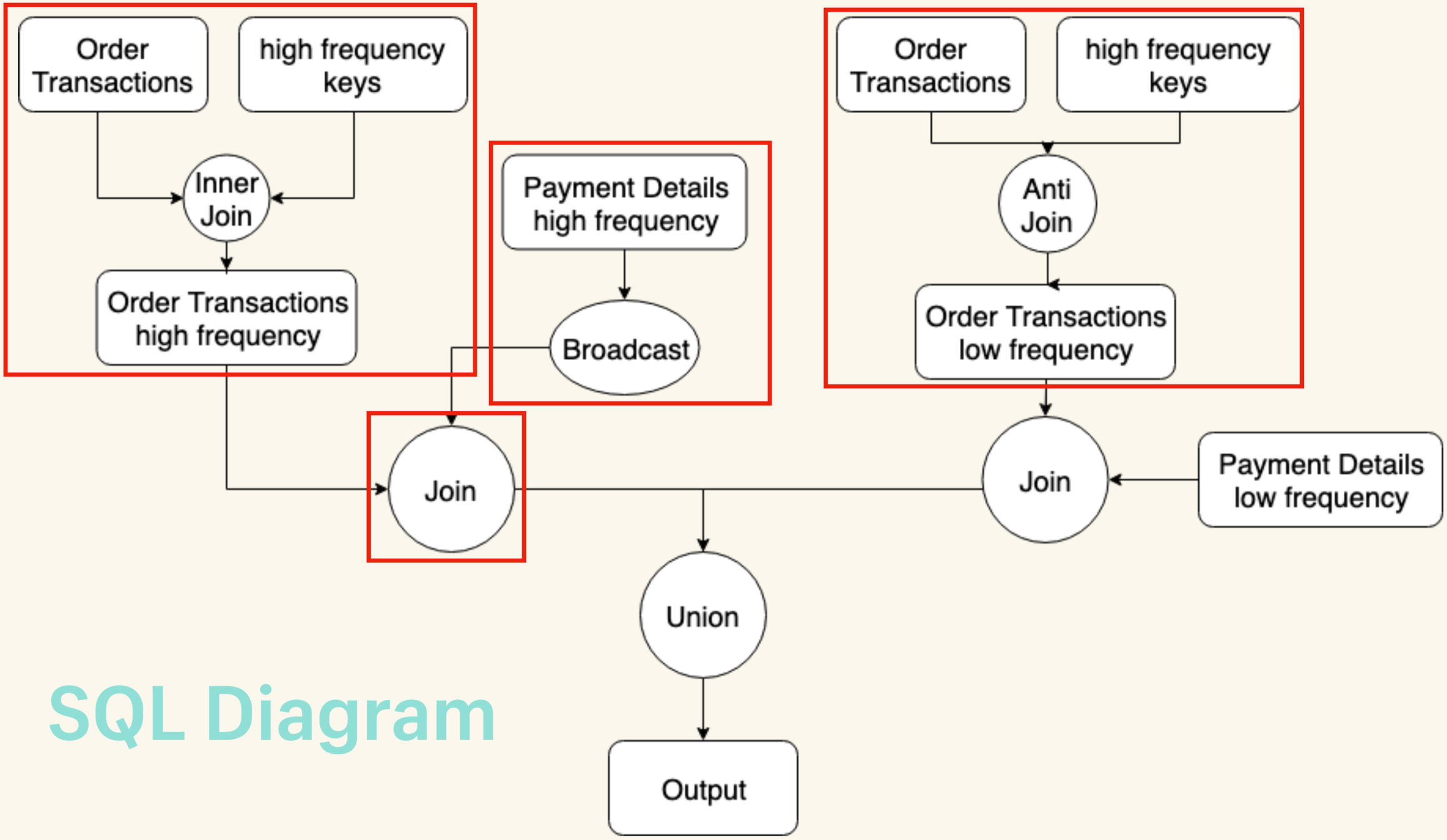
SQL Diagram


Order Transactions
high frequency keys
Inner Join
Order Transactions high frequency
Payment Details high frequency
Broadcast
Order Transactions
high frequency keys
Anti Join
Order Transactions low frequency
Join
Join
Payment Details low frequency
Union
Output

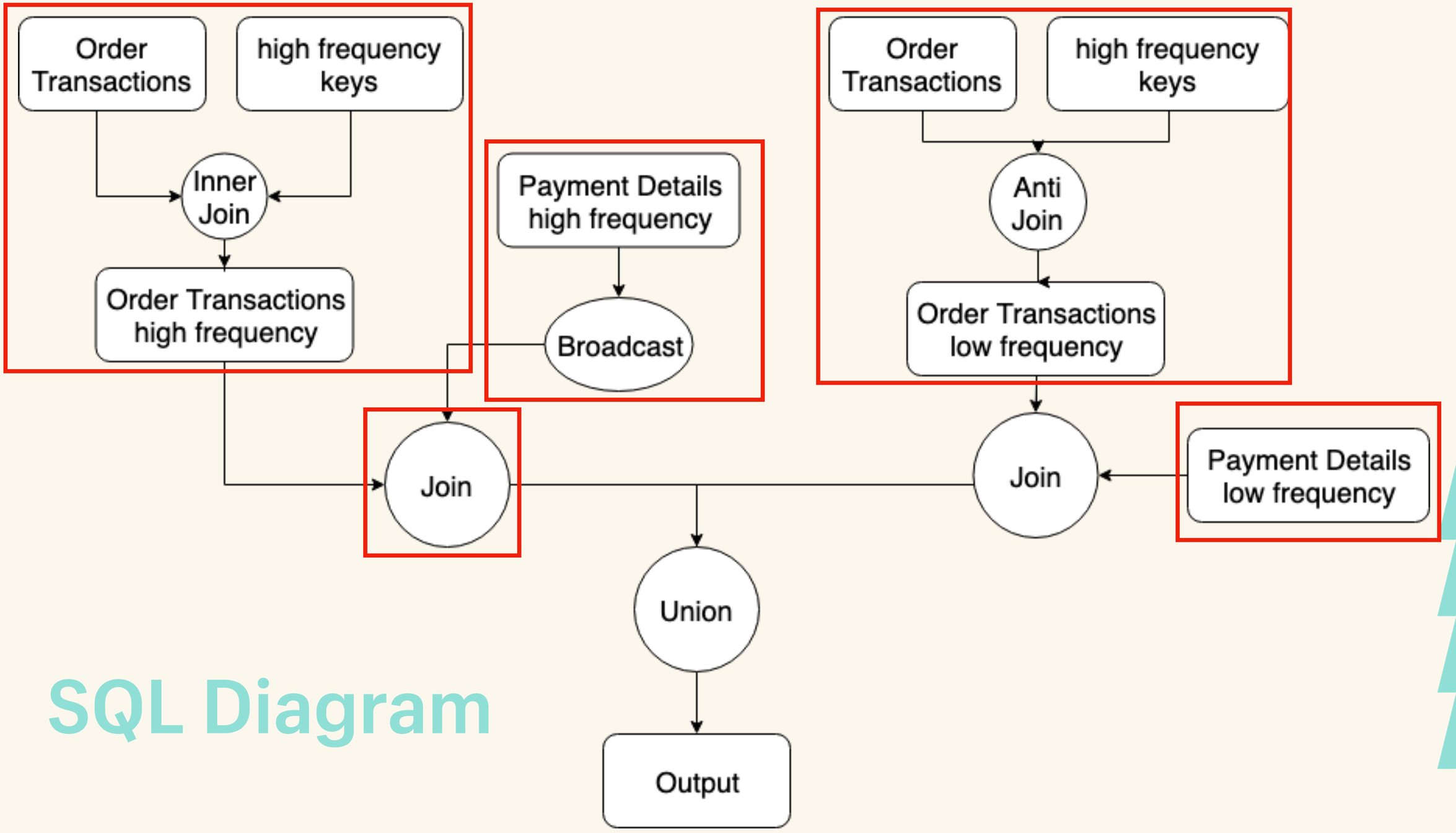
SQL Diagram


Order Transactions
high frequency keys
Inner Join
Order Transactions high frequency
Payment Details high frequency
Broadcast
Order Transactions
high frequency keys
Anti Join
Order Transactions low frequency
Join
Join
Payment Details low frequency
Union
Output

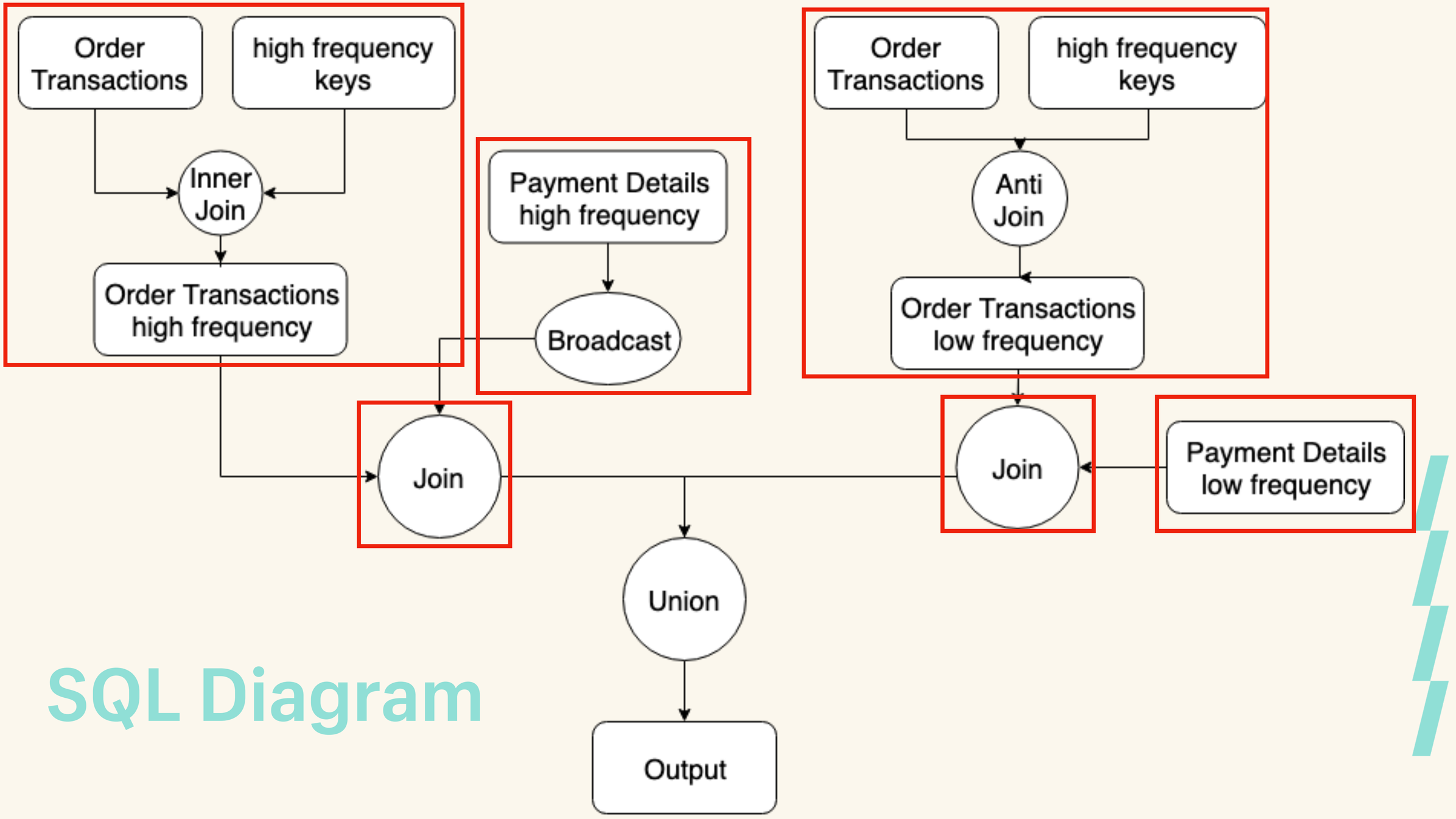
SQL Diagram


Order Transactions
high frequency keys
Inner Join
Order Transactions high frequency
Payment Details high frequency
Broadcast
Order Transactions
high frequency keys
Anti Join
Order Transactions low frequency
Join
Join
Payment Details low frequency
Union
Output

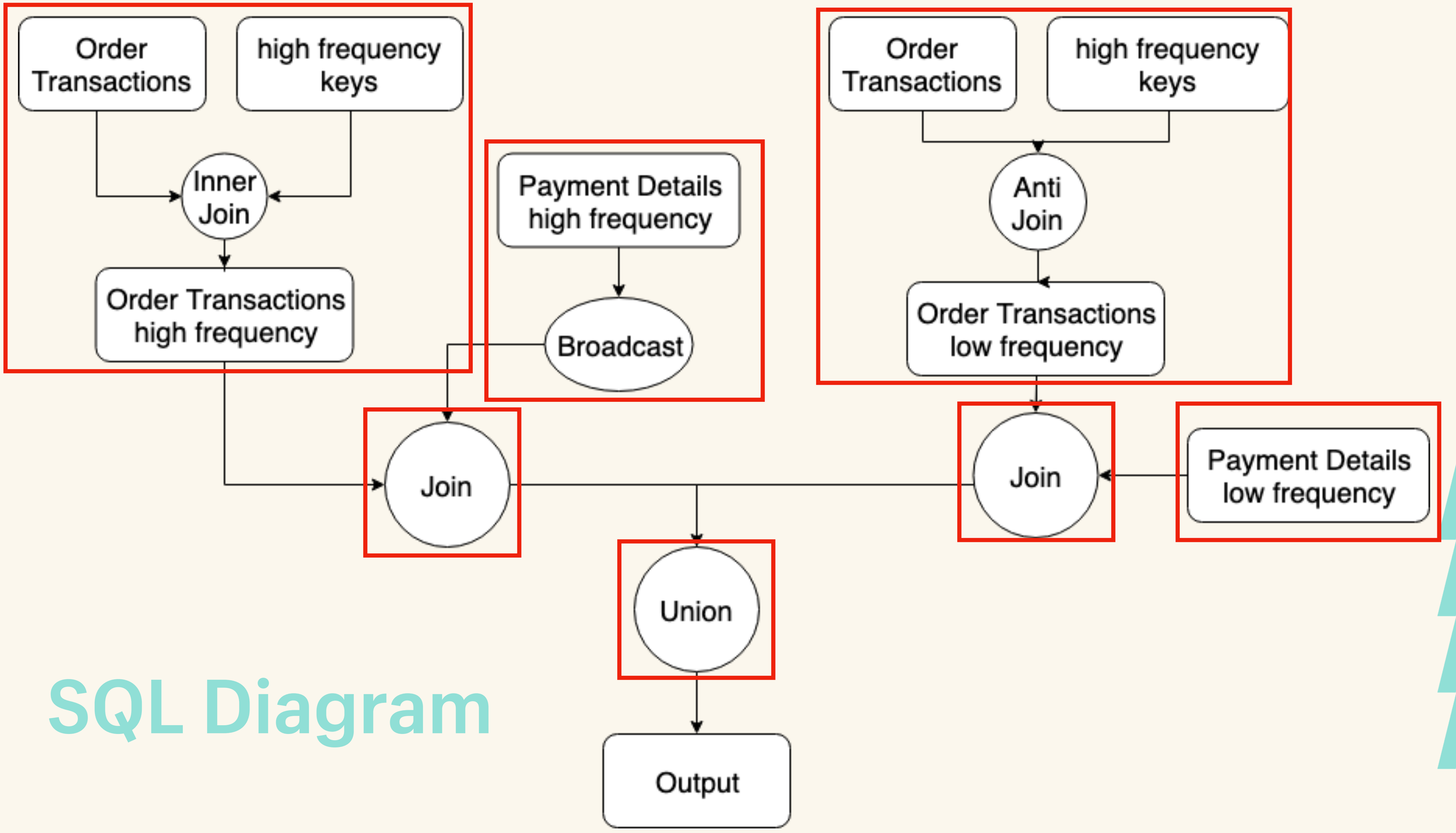
SQL Diagram


Order Transactions
high frequency keys
Inner Join
Order Transactions high frequency
Payment Details high frequency
Broadcast
Order Transactions
high frequency keys
Anti Join
Order Transactions low frequency
Join
Join
Payment Details low frequency
Union
Output

SQL Diagram


Order Transactions
high frequency keys
Inner Join
Order Transactions high frequency
Payment Details high frequency
Broadcast
Order Transactions
high frequency keys
Anti Join
Order Transactions low frequency
Join
Join
Payment Details low frequency
Union
Output

SQL Diagram


Order Transactions
high frequency keys
Inner Join
Order Transactions high frequency
Payment Details high frequency
Broadcast
Order Transactions
high frequency keys
Anti Join
Order Transactions low frequency
Join
Join
Payment Details low frequency
Union
Output

SQL Diagram

Downsides

- The complexity of the job increases as this is a more complex join.
- The broadcasting might timeout?

Key Skew (cont’d)

- The techniques can be used together.
- ie. Largest skew key is null, then a non-null key.
    - Remove the nulls, then salt.

SQL Tab Recap

1. Interpreted the overall spark DAG.
2. Found the spark plans from the details dropdown.
3. Mapped parts of the plan to the exact code line.
4. Analyzed aggregate level stats.

Red Flags:

1. Long and wide DAGs.
2. Big variation in min/med/max times.

** Evaluate plan in the order that they are listed.

Join Optimization Recap

- Spot a join block that have widely varying task times.
- Run dev skew to analyze the distribution of keys.
- Pick the correct join optimization.
- Implement.

Section 4: Optimization 2 - Eliminating Data Spill

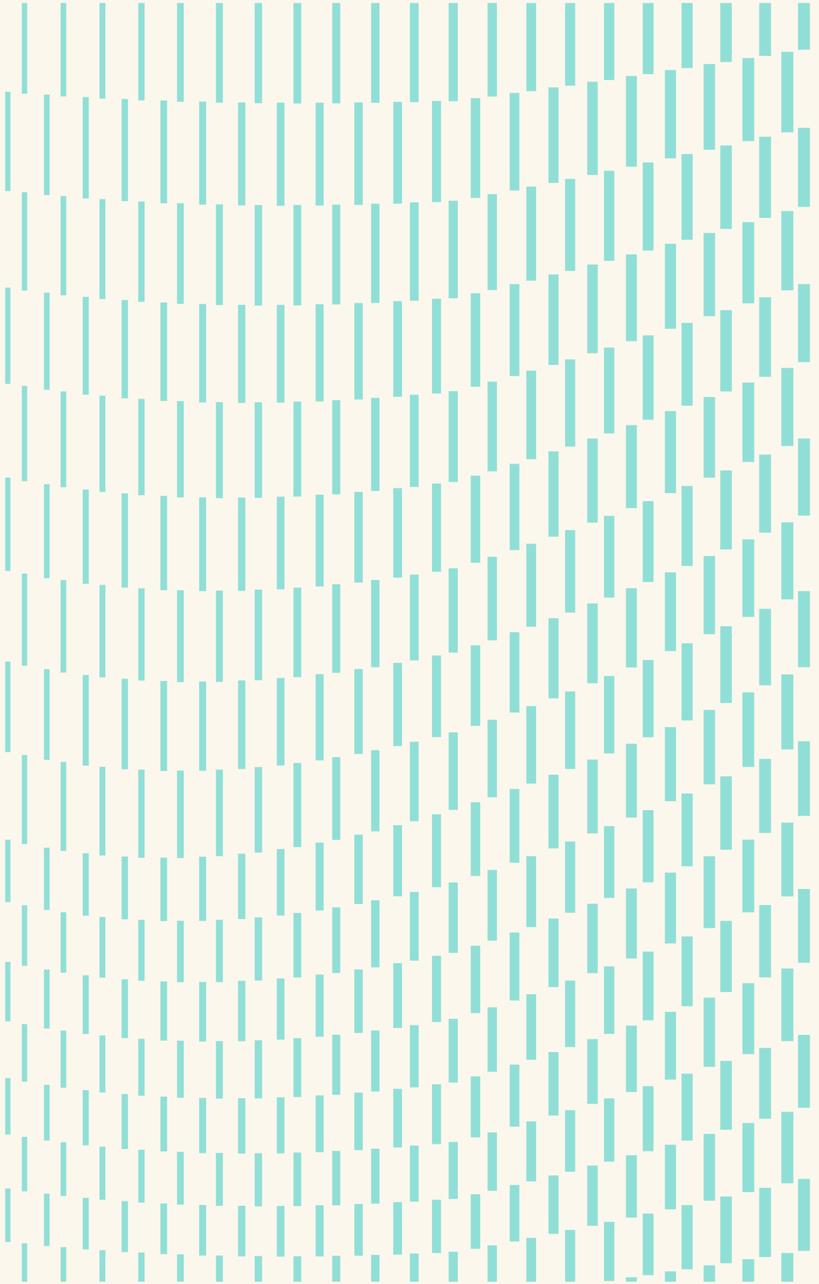
Jobs, Stages, Tasks

Executors

Spill

Partitions / Tasks

Optimal Partition Calculations



Disclaimer

These screenshot are taken of the stages when we had the join skew.

They illustrate more things that you want to be aware of in your stage.

But not needed for the second optimization.

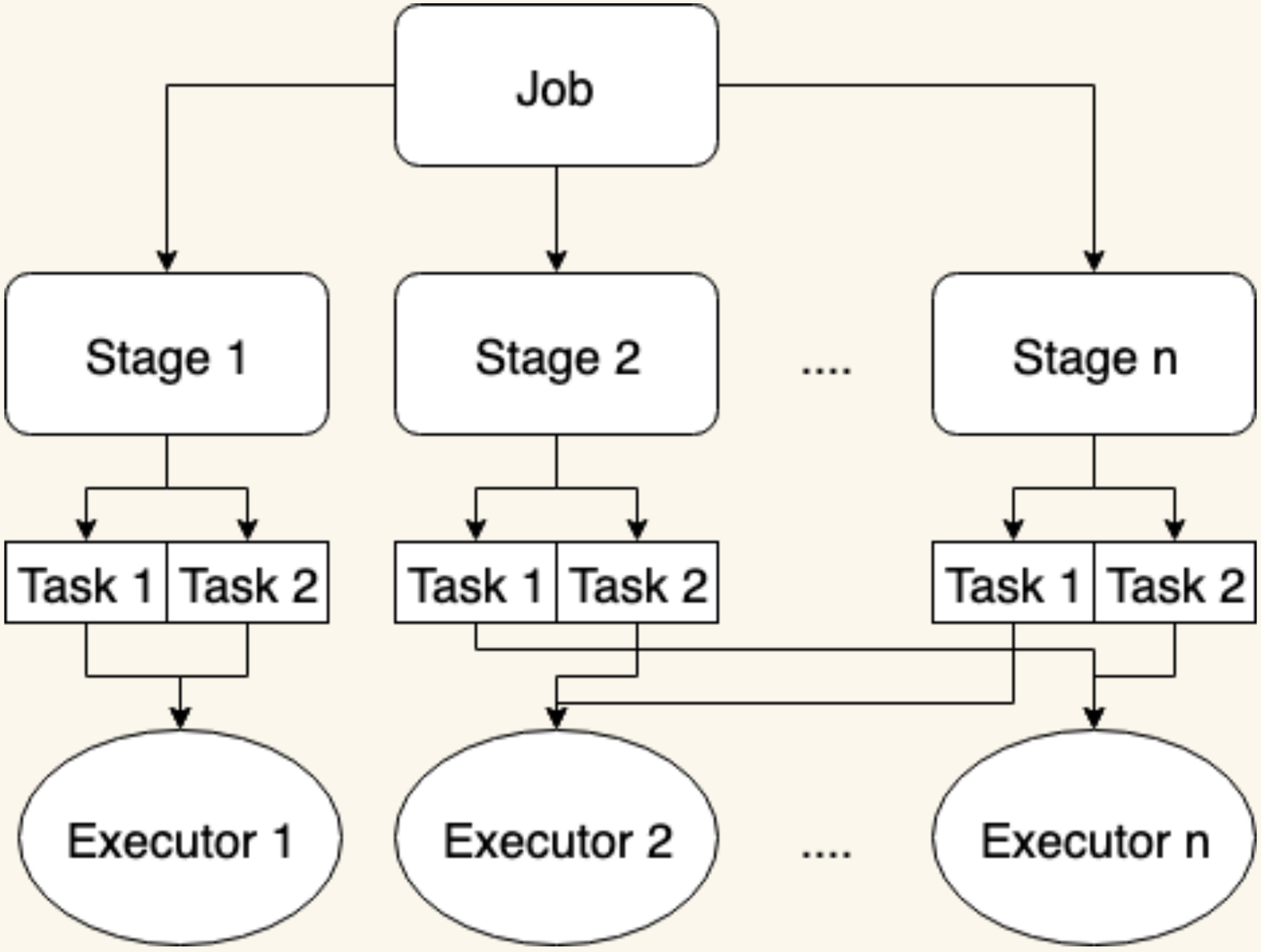
Jobs, Stages, Tasks, Executors

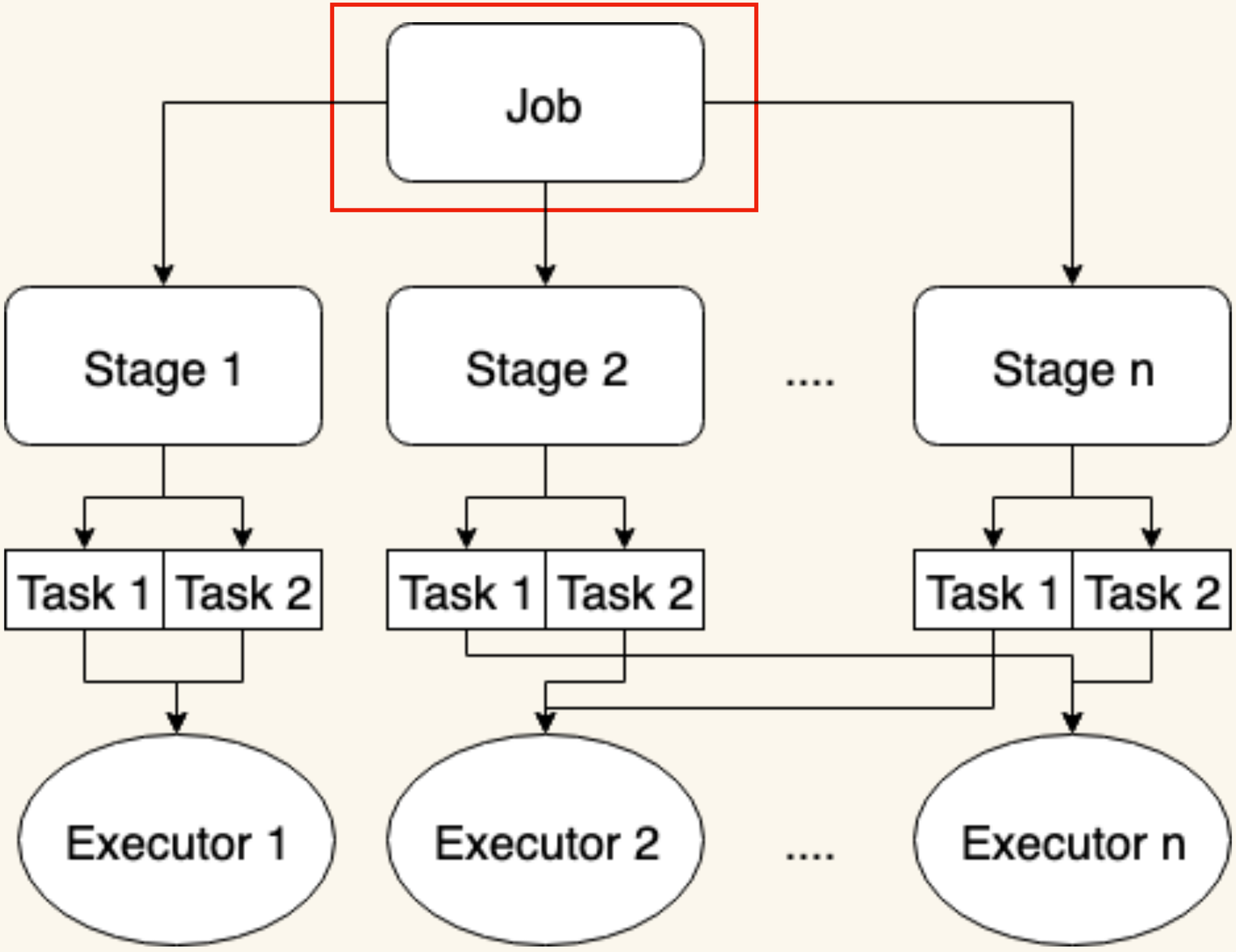
- Every Spark “action” triggers a “job”.
- Every job contains multiple “stages”.
- New stages are created when a shuffle operation is required.
- A stage is a collection of transformations.
- A stage divides the work into a number of “tasks”.
- Tasks run in parallel on “executors”.

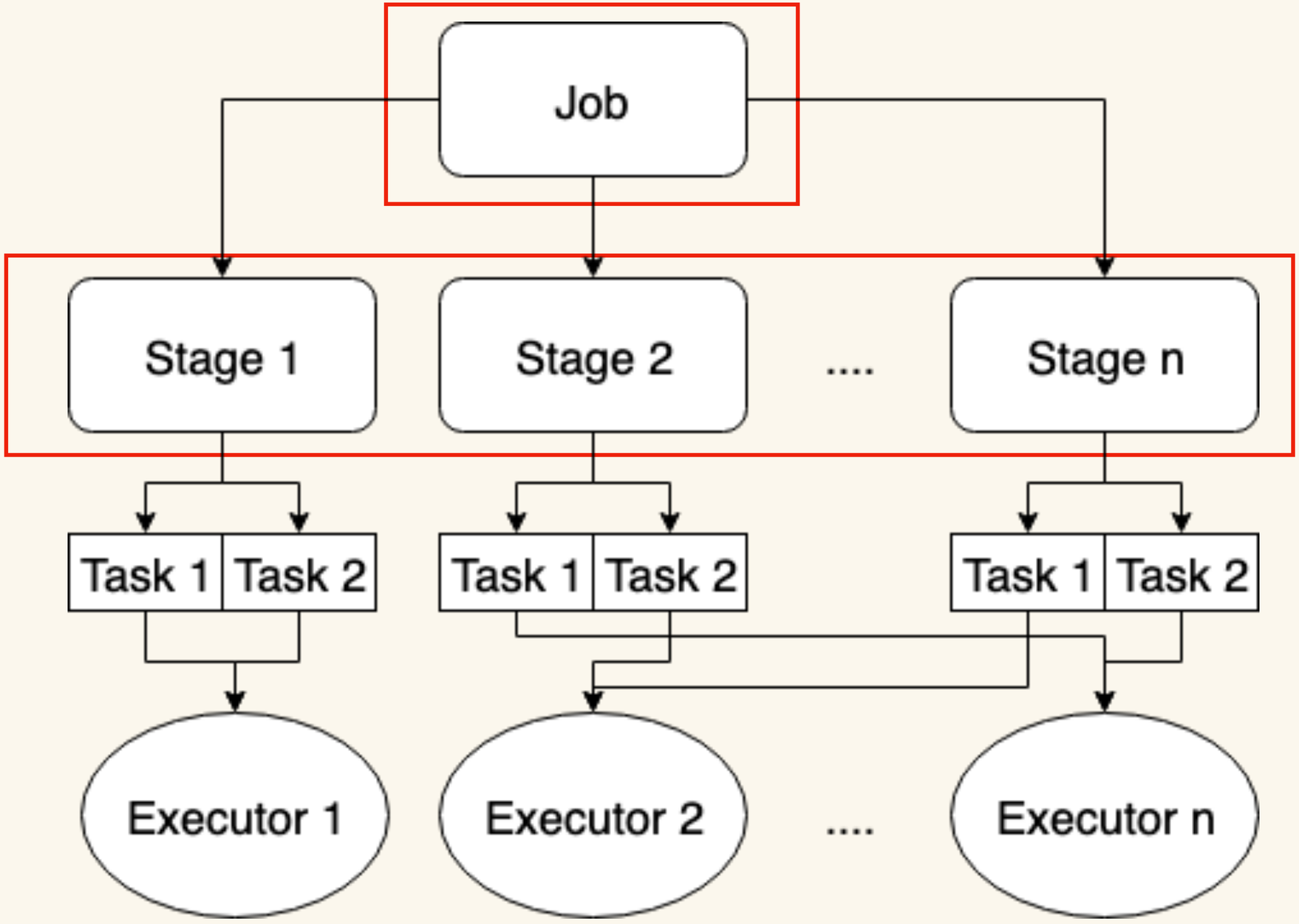
Executors

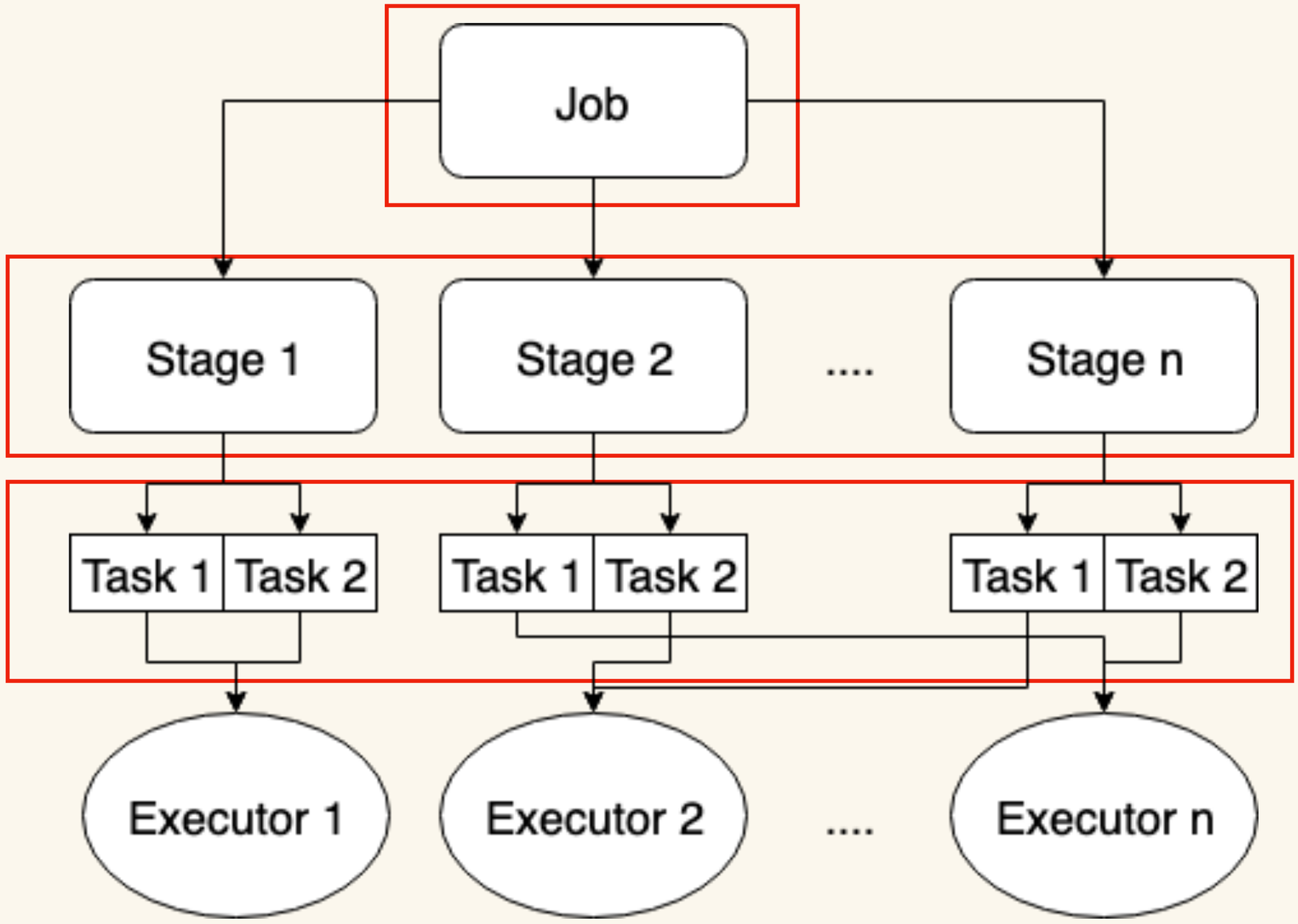
- Read Karl's discourse response to understand the architecture of Spark or go to Michael Style's talk.
- All we need to know is that executors are responsible for running tasks.
- The number of executors and memory per executor is set in the resource classes.

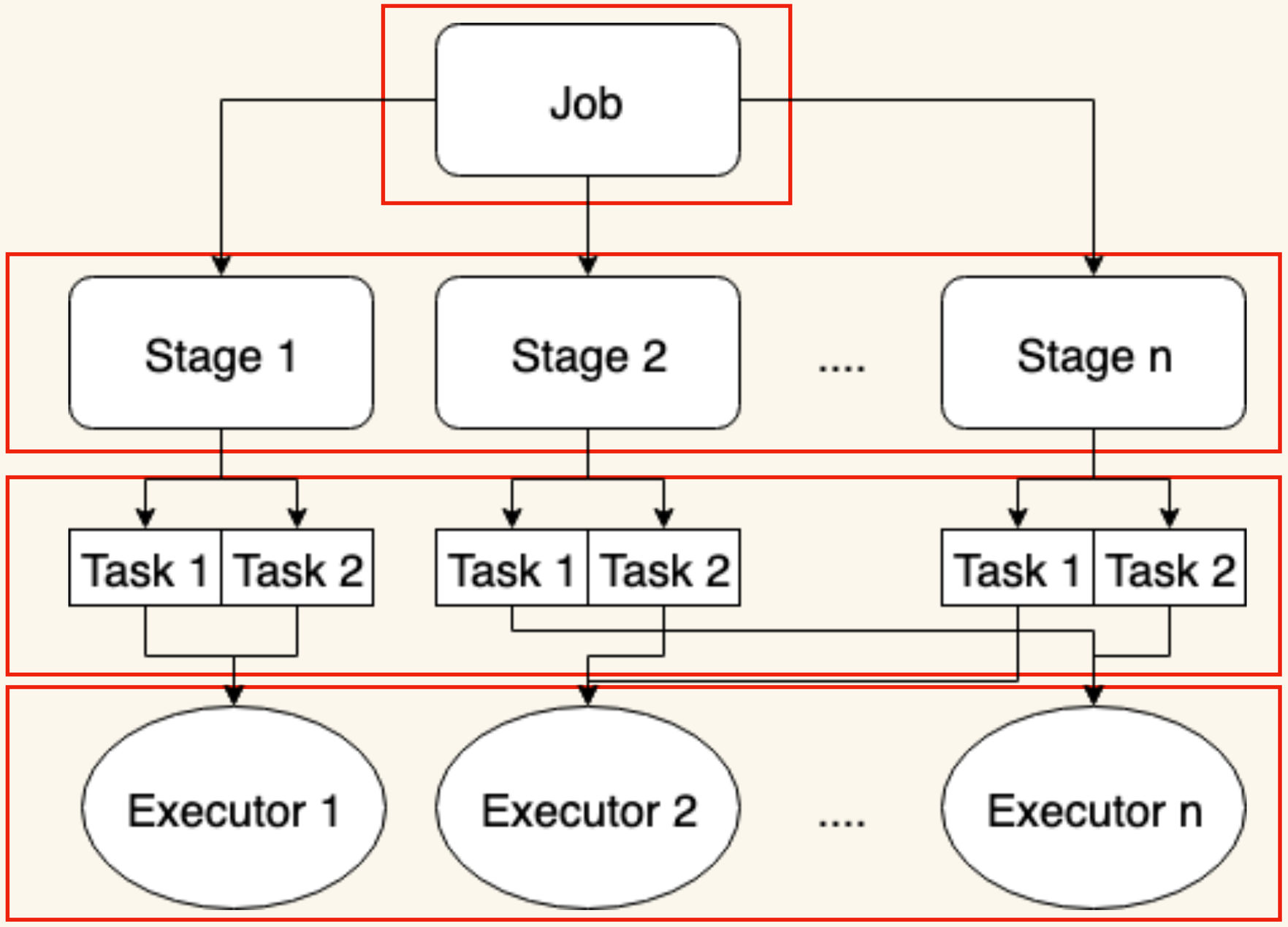
Karl's discourse response: https://discourse.shopify.io/t/what-do-each-of-the-resource-class-arguments-mean/1920/3


Job
Stage 1
Stage 2
....
Stage n
Task 1
Task 2
Task 1
Task 2
Task 1
Task 2
Executor 1
Executor 2
....
Executor n


Job
Stage 1
Stage 2
....
Stage n
Task 1
Task 2
Task 1
Task 2
Task 1
Task 2
Executor 1
Executor 2
....
Executor n


Job
Stage 1
Stage 2
....
Stage n
Task 1
Task 2
Task 1
Task 2
Task 1
Task 2
Executor 1
Executor 2
....
Executor n


Job
Stage 1
Stage 2
....
Stage n
Task 1
Task 2
Task 1
Task 2
Task 1
Task 2
Executor 1
Executor 2
....
Executor n


Job
Stage 1
Stage 2
....
Stage n
Task 1
Task 2
Task 1
Task 2
Task 1
Task 2
Executor 1
Executor 2
....
Executor n

Spill

- An executor has a set amount of memory, set at the start of the spark application.
- An executor is responsible for running tasks.
- A task is a unit of transformation(s) on a partition of data.
- Is the partition of data exceeds the executor memory, it will spill the data from memory to disk.

Spill

- Spilling is very bad, lots of I/O, serialization, etc. costs.
- If we spill too much we can potentially take the executors down.

Jobs

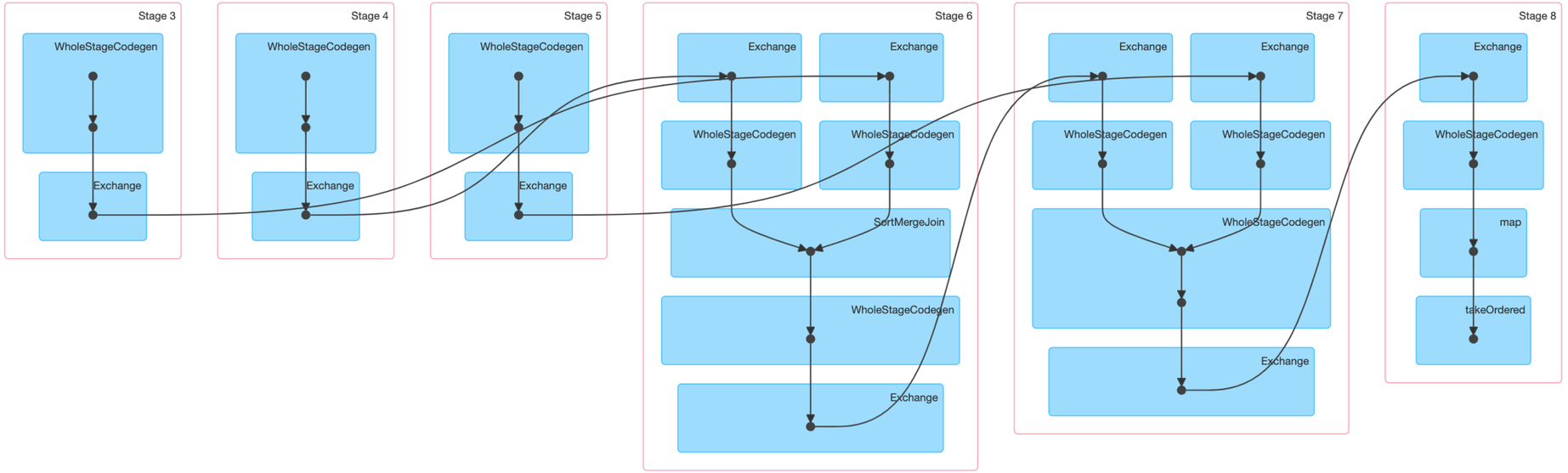
- A job in a Spark application corresponds to a single “action” performed.
- A job consists of many stages.
- A new stage is created for every “shuffle operation”.

```
# Review of actions in Spark:
df.write()
df.head(n)
df.take(n)
df.collect()
df.show()
df.toPandas()
```

Details for Job 3

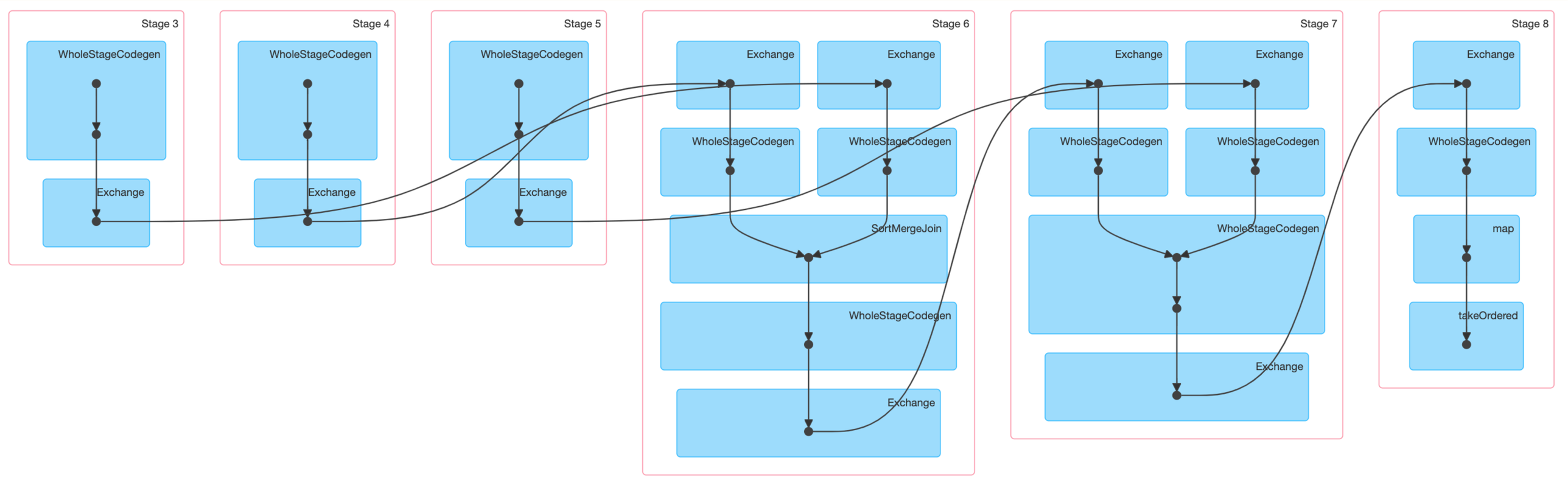
Status: SUCCEEDED
Completed Stages: 6

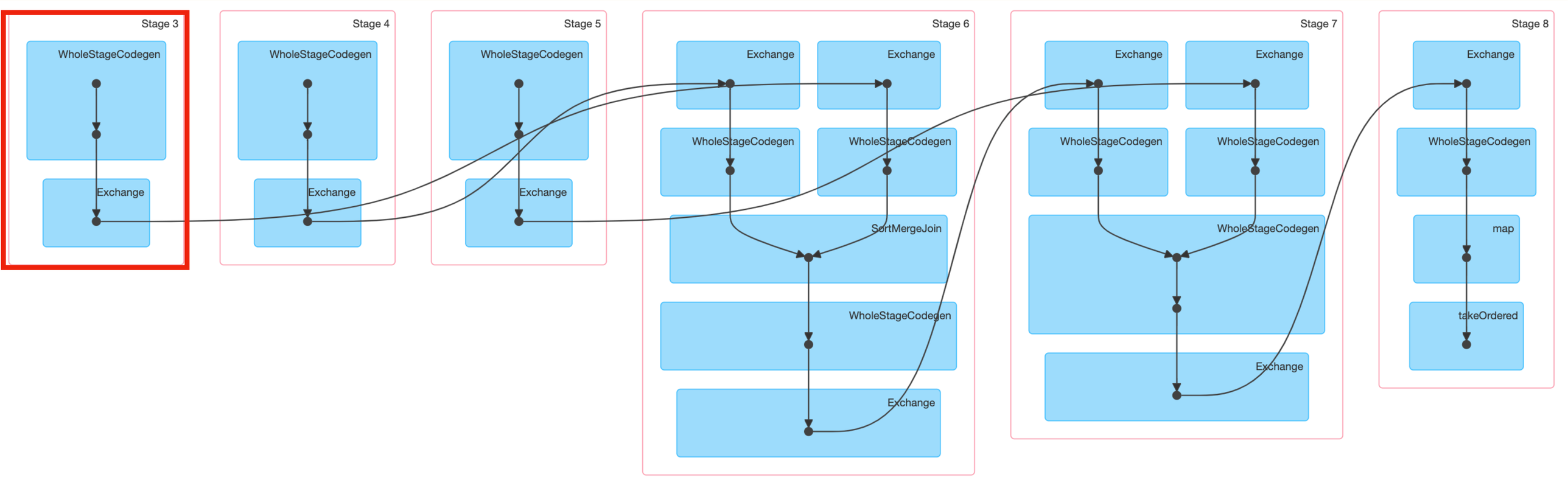
▸ Event Timeline
▾ DAG Visualization

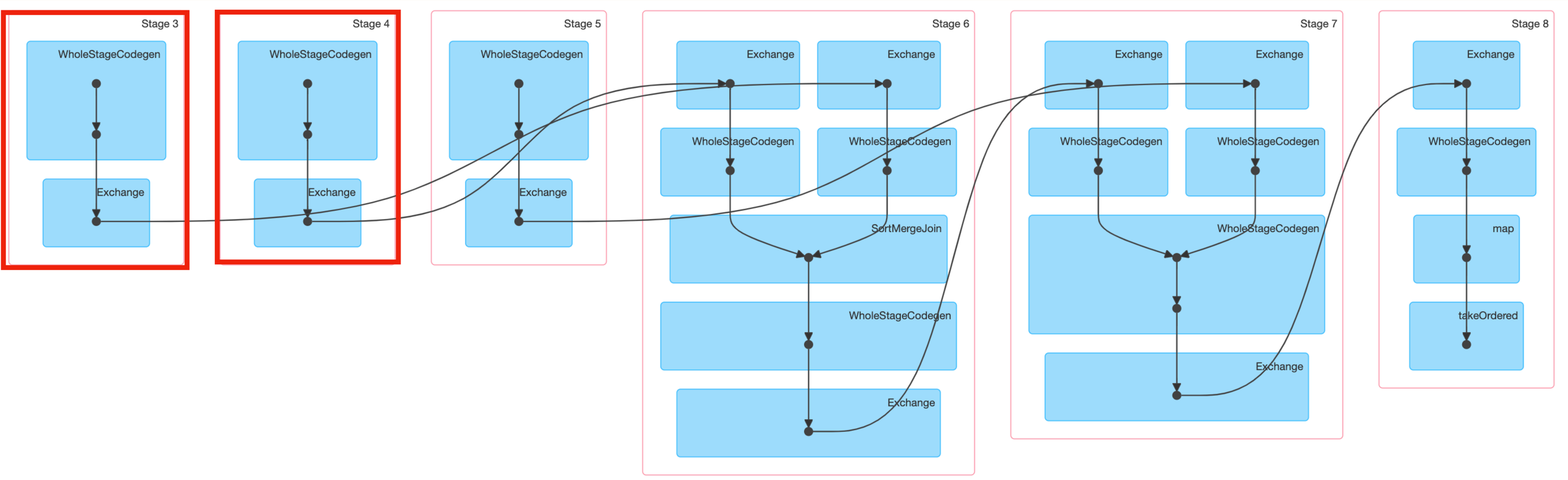


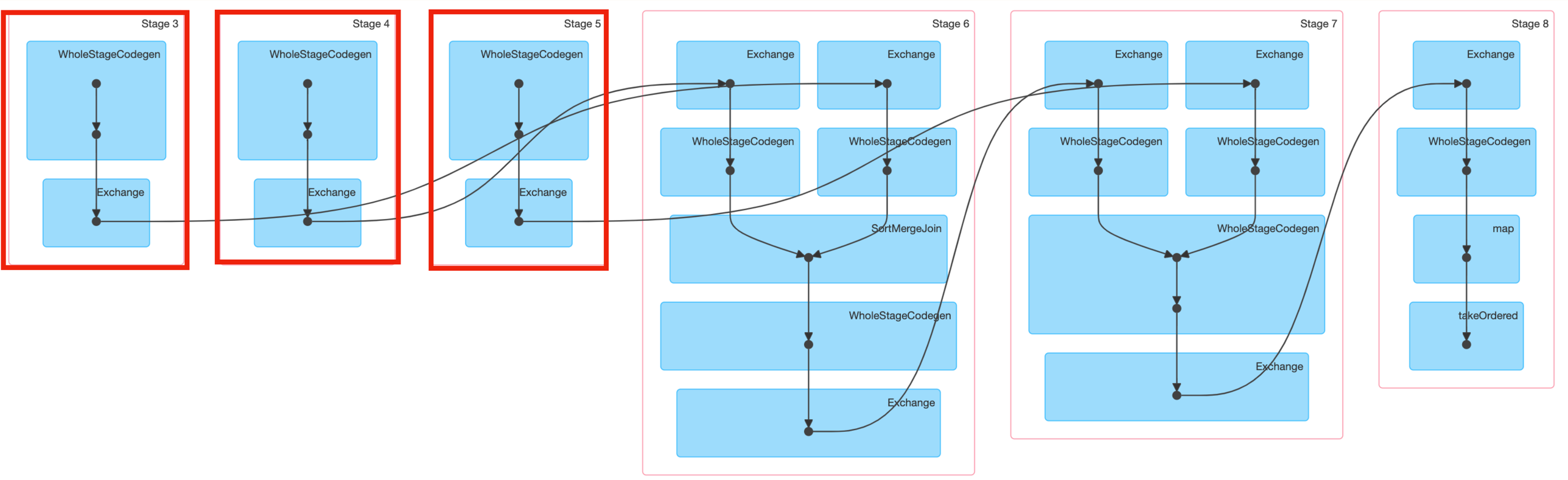
Completed Stages (6)

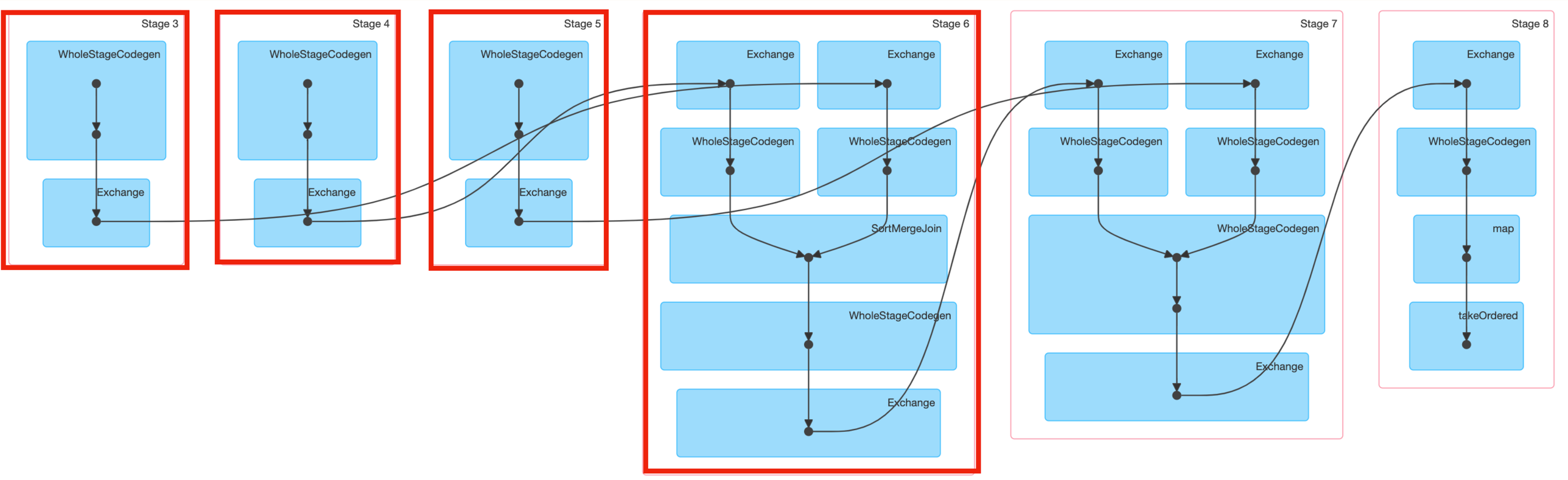
Stage Id ▾	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
8	showString at NativeMethodAccessorImpl.java:0 +details	2019/07/25 14:36:29	0.4 s	200/200			287.4 KB	
7	showString at NativeMethodAccessorImpl.java:0 +details	2019/07/25 14:34:55	1.6 min	200/200			25.1 GB	287.4 KB
6	showString at NativeMethodAccessorImpl.java:0 +details	2019/07/25 14:22:50	12 min	200/200 (1 killed: another attempt succeeded)			29.7 GB	18.6 GB
5	showString at NativeMethodAccessorImpl.java:0 +details	2019/07/25 14:19:18	3.9 min	1044/1044	16.0 GB			8.5 GB
4	showString at NativeMethodAccessorImpl.java:0 +details	2019/07/25 14:19:18	3.3 min	14629/14629	86.2 GB			25.1 GB
3	showString at NativeMethodAccessorImpl.java:0 +details	2019/07/25 14:19:18	12 s	413/413	213.4 MB			151.0 MB

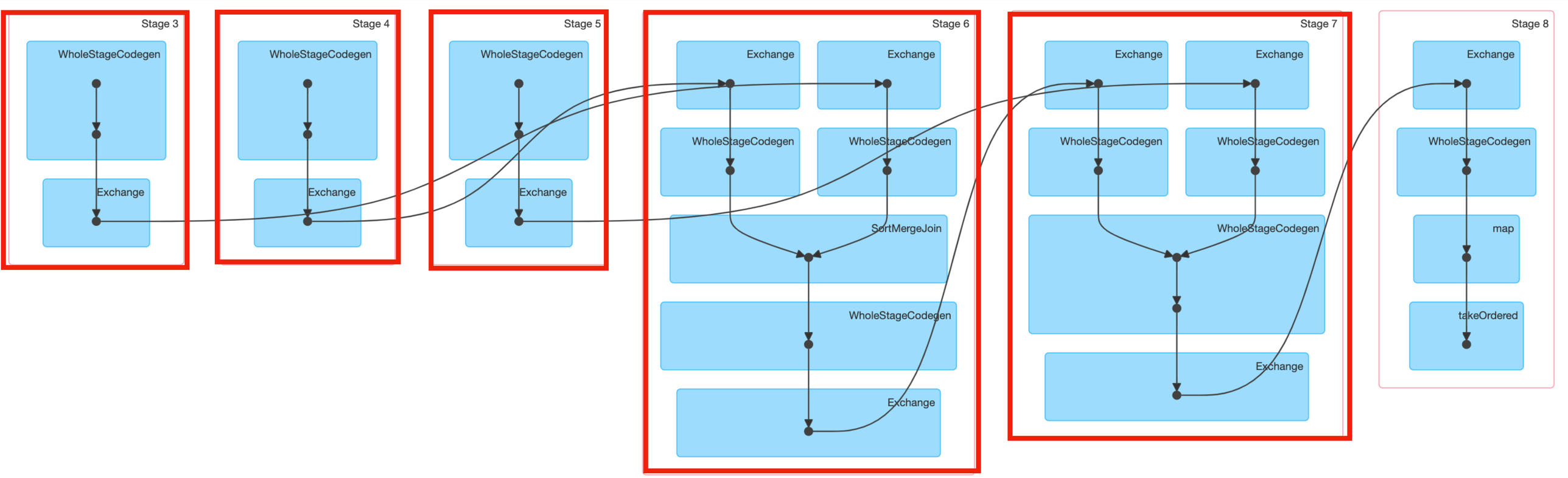

Stage 3
WholeStageCodegen
Exchange
Stage 4
WholeStageCodegen
Exchange
Stage 5
WholeStageCodegen
Exchange
Stage 6
Exchange
Exchange
WholeStageCodegen
WholeStageCodegen
SortMergeJoin
WholeStageCodegen
Exchange
Stage 7
Exchange
Exchange
WholeStageCodegen
WholeStageCodegen
WholeStageCodegen
Exchange
Stage 8
Exchange
WholeStageCodegen
map
takeOrdered

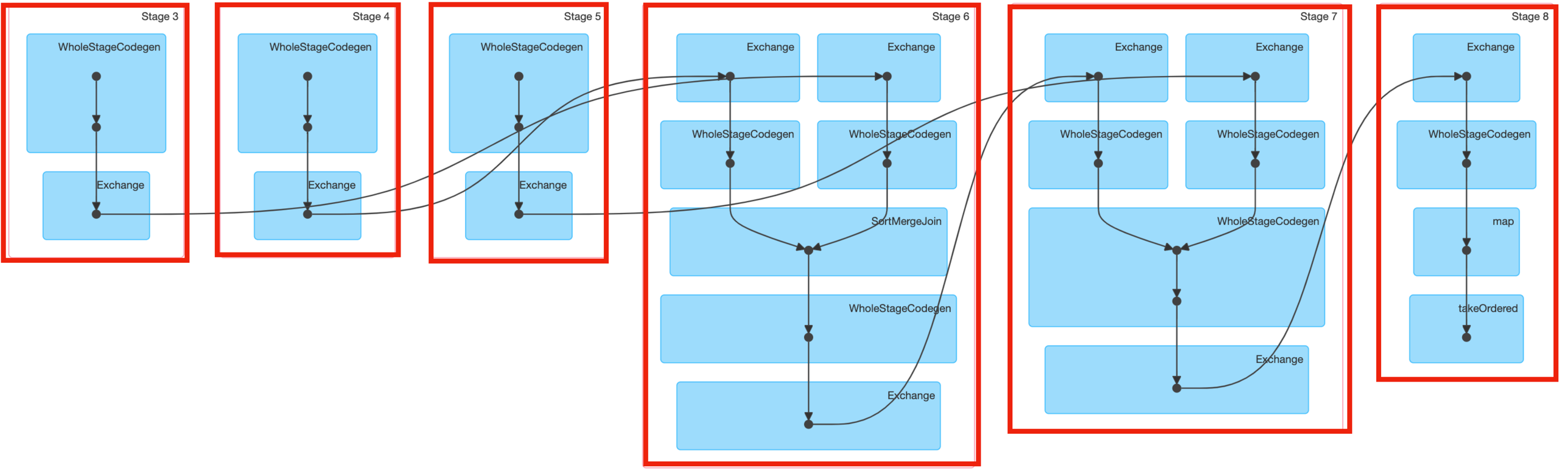

Stage 3
WholeStageCodegen
Exchange
Stage 4
WholeStageCodegen
Exchange
Stage 5
WholeStageCodegen
Exchange
Stage 6
Exchange
Exchange
WholeStageCodegen
WholeStageCodegen
SortMergeJoin
WholeStageCodegen
Exchange
Stage 7
Exchange
Exchange
WholeStageCodegen
WholeStageCodegen
WholeStageCodegen
Exchange
Stage 8
Exchange
WholeStageCodegen
map
takeOrdered

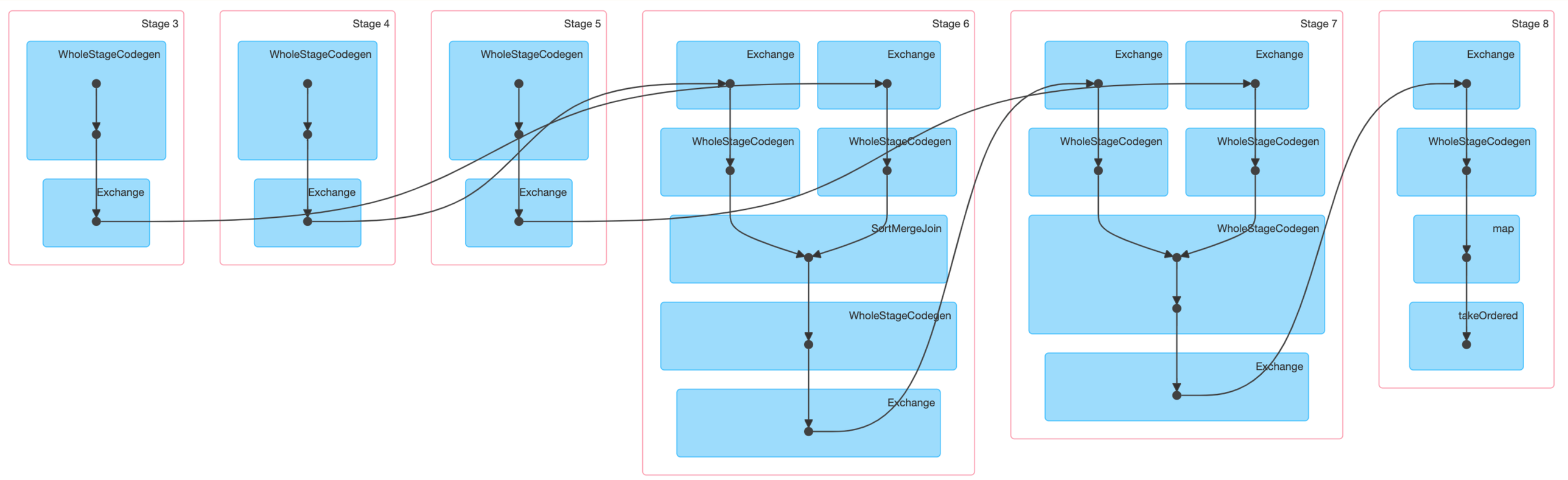

Stage 3
WholeStageCodegen
Exchange
Stage 4
WholeStageCodegen
Exchange
Stage 5
WholeStageCodegen
Exchange
Stage 6
Exchange
Exchange
WholeStageCodegen
WholeStageCodegen
SortMergeJoin
WholeStageCodegen
Exchange
Stage 7
Exchange
Exchange
WholeStageCodegen
WholeStageCodegen
WholeStageCodegen
Exchange
Stage 8
Exchange
WholeStageCodegen
map
takeOrdered

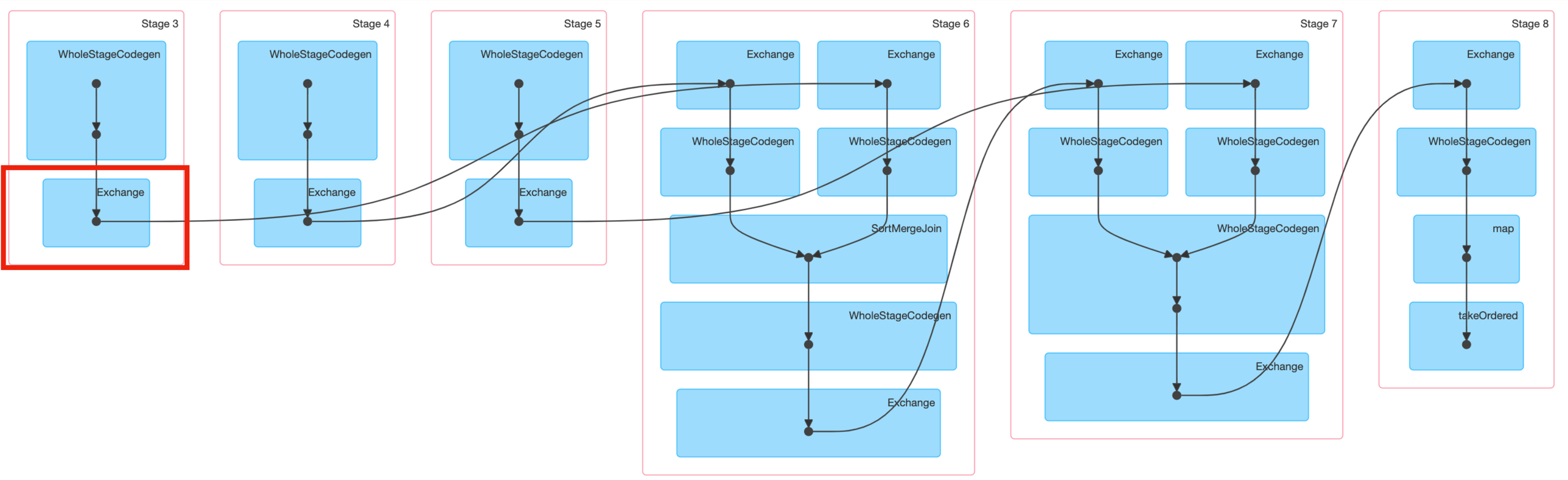

Stage 3
WholeStageCodegen
Exchange
Stage 4
WholeStageCodegen
Exchange
Stage 5
WholeStageCodegen
Exchange
Stage 6
Exchange
Exchange
WholeStageCodegen
WholeStageCodegen
SortMergeJoin
WholeStageCodegen
Exchange
Stage 7
Exchange
Exchange
WholeStageCodegen
WholeStageCodegen
WholeStageCodegen
Exchange
Stage 8
Exchange
WholeStageCodegen
map
takeOrdered

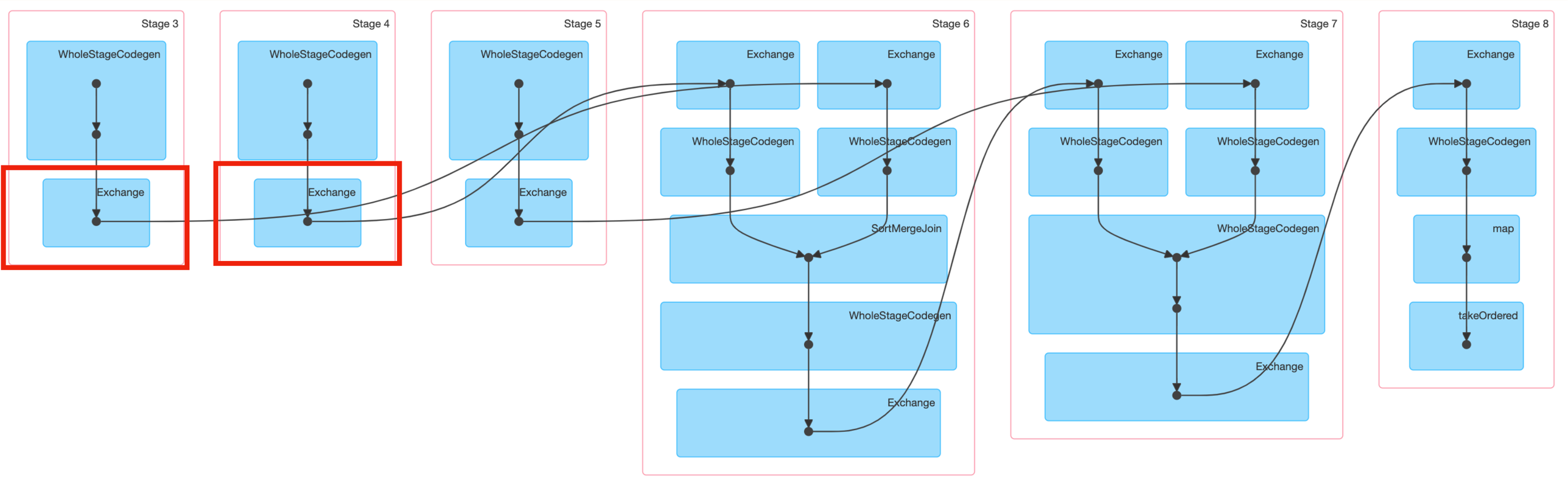

Stage 3
WholeStageCodegen
Exchange
Stage 4
WholeStageCodegen
Exchange
Stage 5
WholeStageCodegen
Exchange
Stage 6
Exchange
Exchange
WholeStageCodegen
WholeStageCodegen
SortMergeJoin
WholeStageCodegen
Exchange
Stage 7
Exchange
Exchange
WholeStageCodegen
WholeStageCodegen
WholeStageCodegen
Exchange
Stage 8
Exchange
WholeStageCodegen
map
takeOrdered

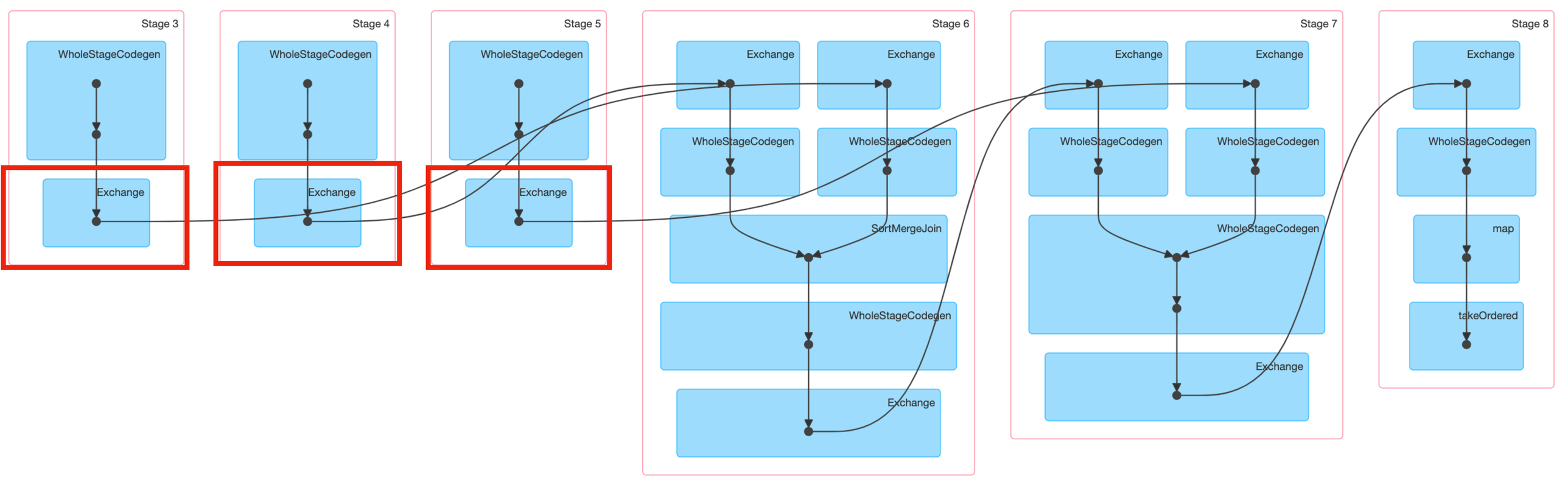

Stage 3
WholeStageCodegen
Exchange
Stage 4
WholeStageCodegen
Exchange
Stage 5
WholeStageCodegen
Exchange
Stage 6
Exchange
Exchange
WholeStageCodegen
WholeStageCodegen
SortMergeJoin
WholeStageCodegen
Exchange
Stage 7
Exchange
Exchange
WholeStageCodegen
WholeStageCodegen
WholeStageCodegen
Exchange
Stage 8
Exchange
WholeStageCodegen
map
takeOrdered

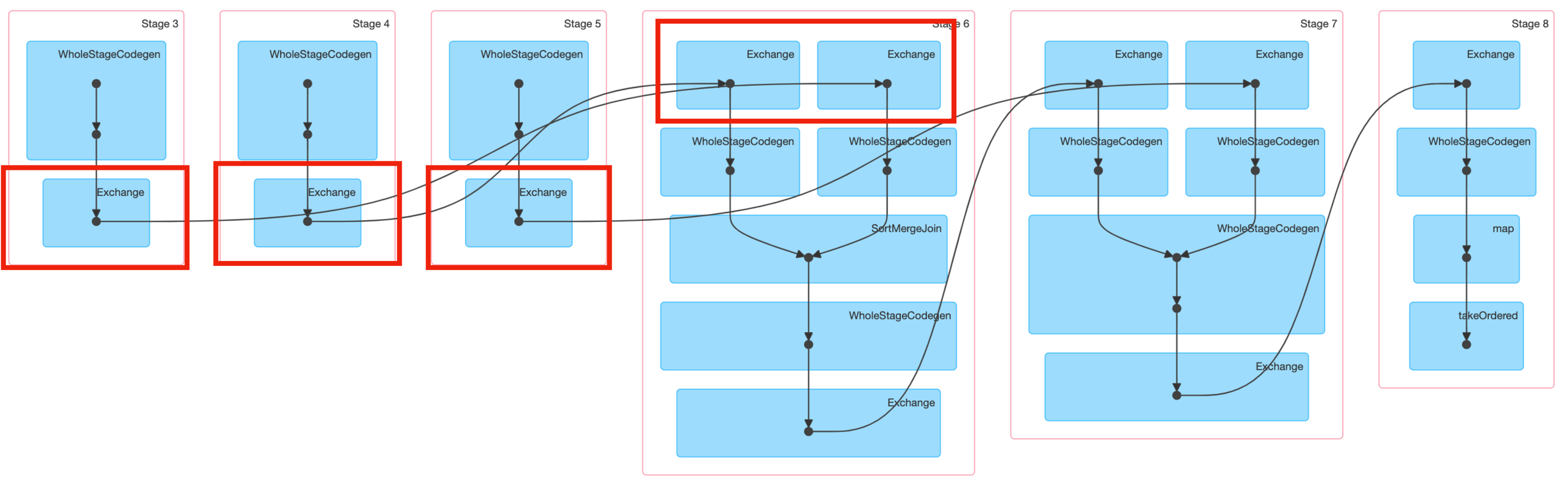

Stage 3
WholeStageCodegen
Exchange
Stage 4
WholeStageCodegen
Exchange
Stage 5
WholeStageCodegen
Exchange
Stage 6
Exchange
Exchange
WholeStageCodegen
WholeStageCodegen
SortMergeJoin
WholeStageCodegen
Exchange
Stage 7
Exchange
Exchange
WholeStageCodegen
WholeStageCodegen
WholeStageCodegen
Exchange
Stage 8
Exchange
WholeStageCodegen
map
takeOrdered

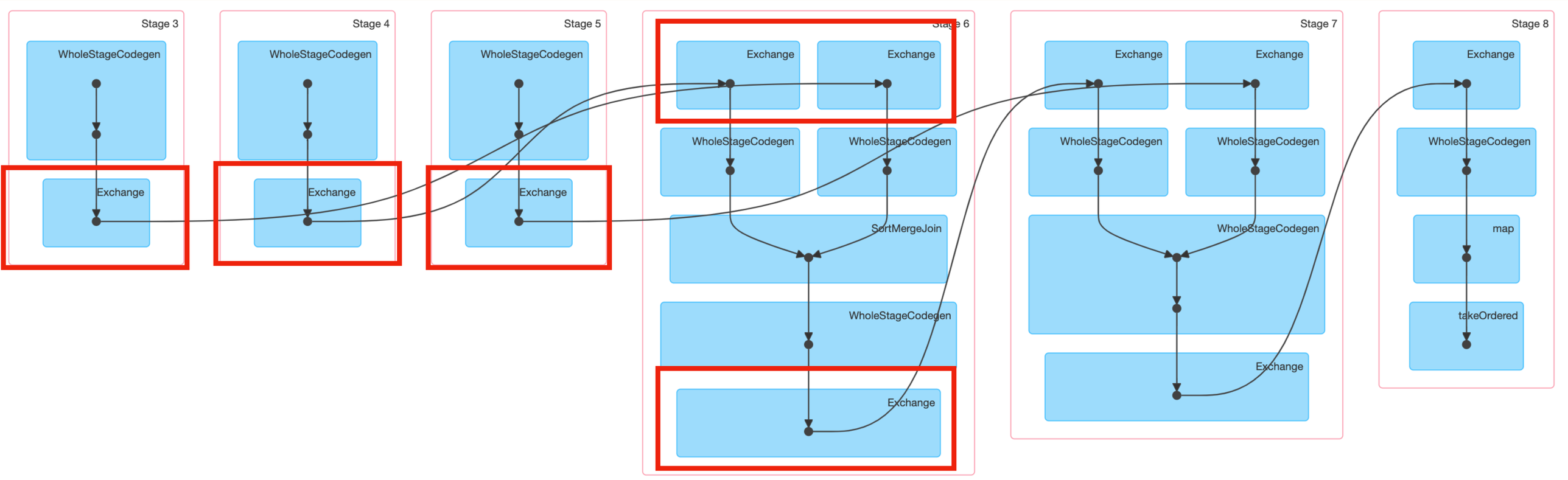

Stage 3
WholeStageCodegen
Exchange
Stage 4
WholeStageCodegen
Exchange
Stage 5
WholeStageCodegen
Exchange
Stage 6
Exchange
Exchange
WholeStageCodegen
WholeStageCodegen
SortMergeJoin
WholeStageCodegen
Exchange
Stage 7
Exchange
Exchange
WholeStageCodegen
WholeStageCodegen
WholeStageCodegen
Exchange
Stage 8
Exchange
WholeStageCodegen
map
takeOrdered

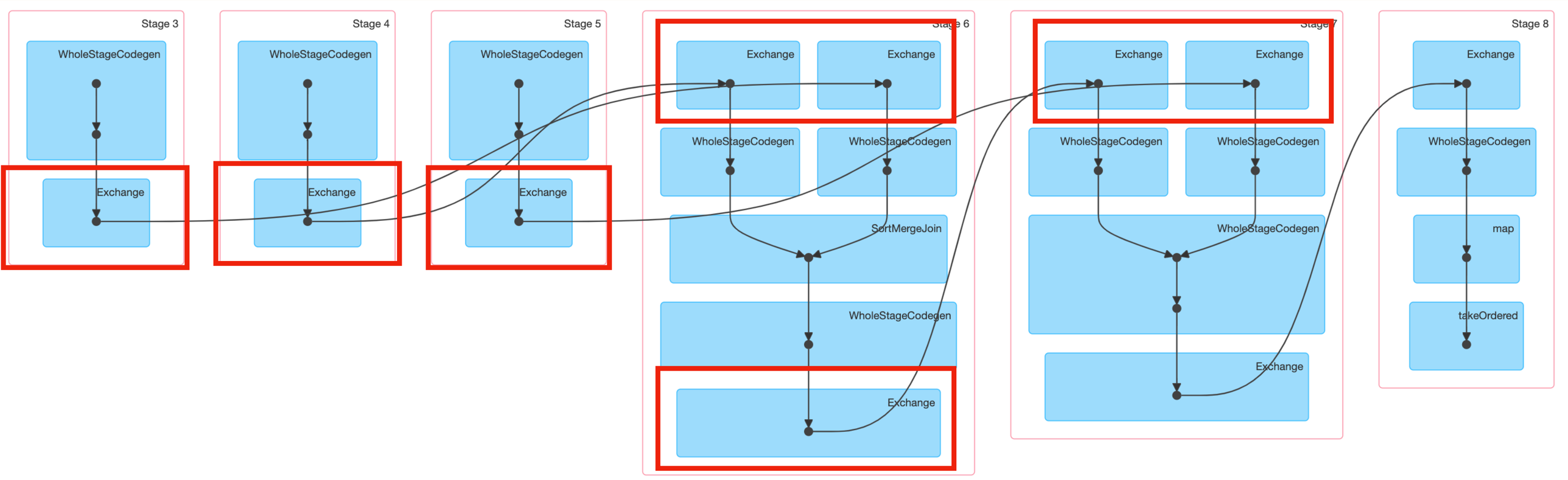

Stage 3
WholeStageCodegen
Exchange
Stage 4
WholeStageCodegen
Exchange
Stage 5
WholeStageCodegen
Exchange
Stage 6
Exchange
Exchange
WholeStageCodegen
WholeStageCodegen
SortMergeJoin
WholeStageCodegen
Exchange
Stage 7
Exchange
Exchange
WholeStageCodegen
WholeStageCodegen
WholeStageCodegen
Exchange
Stage 8
Exchange
WholeStageCodegen
map
takeOrdered

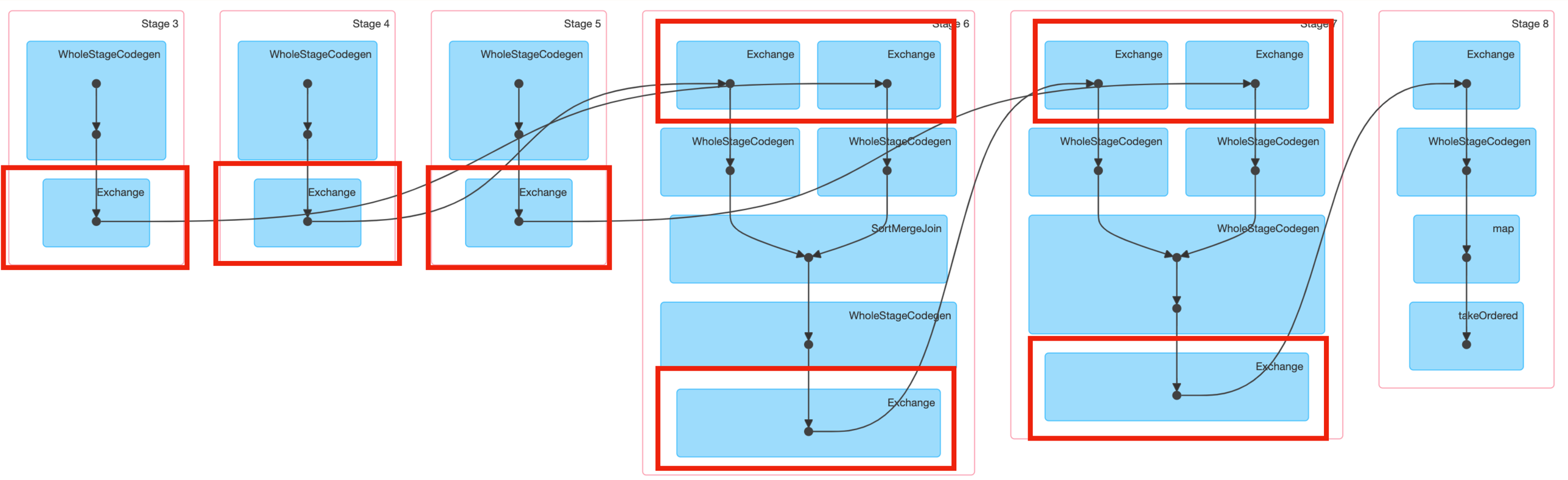

Stage 3
WholeStageCodegen
Exchange
Stage 4
WholeStageCodegen
Exchange
Stage 5
WholeStageCodegen
Exchange
Stage 6
Exchange
Exchange
WholeStageCodegen
WholeStageCodegen
SortMergeJoin
WholeStageCodegen
Exchange
Stage 7
Exchange
Exchange
WholeStageCodegen
WholeStageCodegen
WholeStageCodegen
Exchange
Stage 8
Exchange
WholeStageCodegen
map
takeOrdered

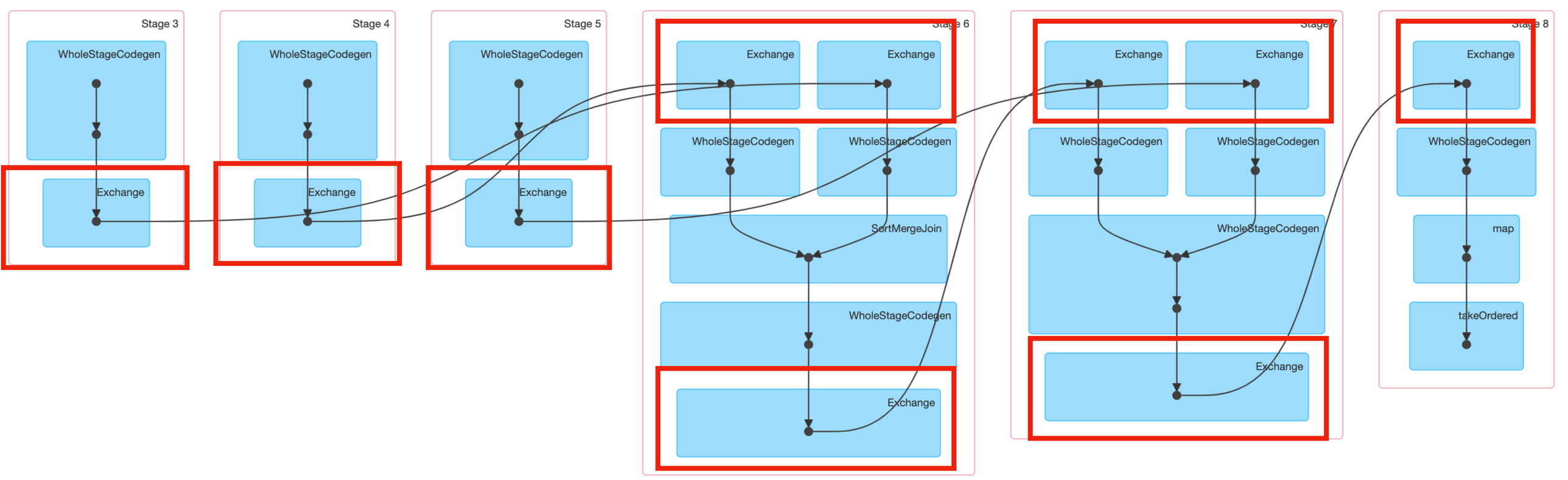

Stage 3
WholeStageCodegen
Exchange
Stage 4
WholeStageCodegen
Exchange
Stage 5
WholeStageCodegen
Exchange
Stage 6
Exchange
Exchange
WholeStageCodegen
WholeStageCodegen
SortMergeJoin
WholeStageCodegen
Exchange
Stage 7
Exchange
Exchange
WholeStageCodegen
WholeStageCodegen
WholeStageCodegen
Exchange
Stage 8
Exchange
WholeStageCodegen
map
takeOrdered


Stage 3
WholeStageCodegen
Exchange
Stage 4
WholeStageCodegen
Exchange
Stage 5
WholeStageCodegen
Exchange
Stage 6
Exchange
Exchange
WholeStageCodegen
WholeStageCodegen
SortMergeJoin
WholeStageCodegen
Exchange
Stage 7
Exchange
Exchange
WholeStageCodegen
WholeStageCodegen
WholeStageCodegen
Exchange
Stage 8
Exchange
WholeStageCodegen
map
takeOrdered


Stage 3
WholeStageCodegen
Exchange
Stage 4
WholeStageCodegen
Exchange
Stage 5
WholeStageCodegen
Exchange
Stage 6
Exchange
Exchange
WholeStageCodegen
WholeStageCodegen
SortMergeJoin
WholeStageCodegen
Exchange
Stage 7
Exchange
Exchange
WholeStageCodegen
WholeStageCodegen
WholeStageCodegen
Exchange
Stage 8
Exchange
WholeStageCodegen
map
takeOrdered


Stage 3
WholeStageCodegen
Exchange
Stage 4
WholeStageCodegen
Exchange
Stage 5
WholeStageCodegen
Exchange
Stage 6
Exchange
Exchange
WholeStageCodegen
WholeStageCodegen
SortMergeJoin
WholeStageCodegen
Exchange
Stage 7
Exchange
Exchange
WholeStageCodegen
WholeStageCodegen
WholeStageCodegen
Exchange
Stage 8
Exchange
WholeStageCodegen
map
takeOrdered


Stage 3
WholeStageCodegen
Exchange
Stage 4
WholeStageCodegen
Exchange
Stage 5
WholeStageCodegen
Exchange
Stage 6
Exchange
Exchange
WholeStageCodegen
WholeStageCodegen
SortMergeJoin
WholeStageCodegen
Exchange
Stage 7
Exchange
Exchange
WholeStageCodegen
WholeStageCodegen
WholeStageCodegen
Exchange
Stage 8
Exchange
WholeStageCodegen
map
takeOrdered

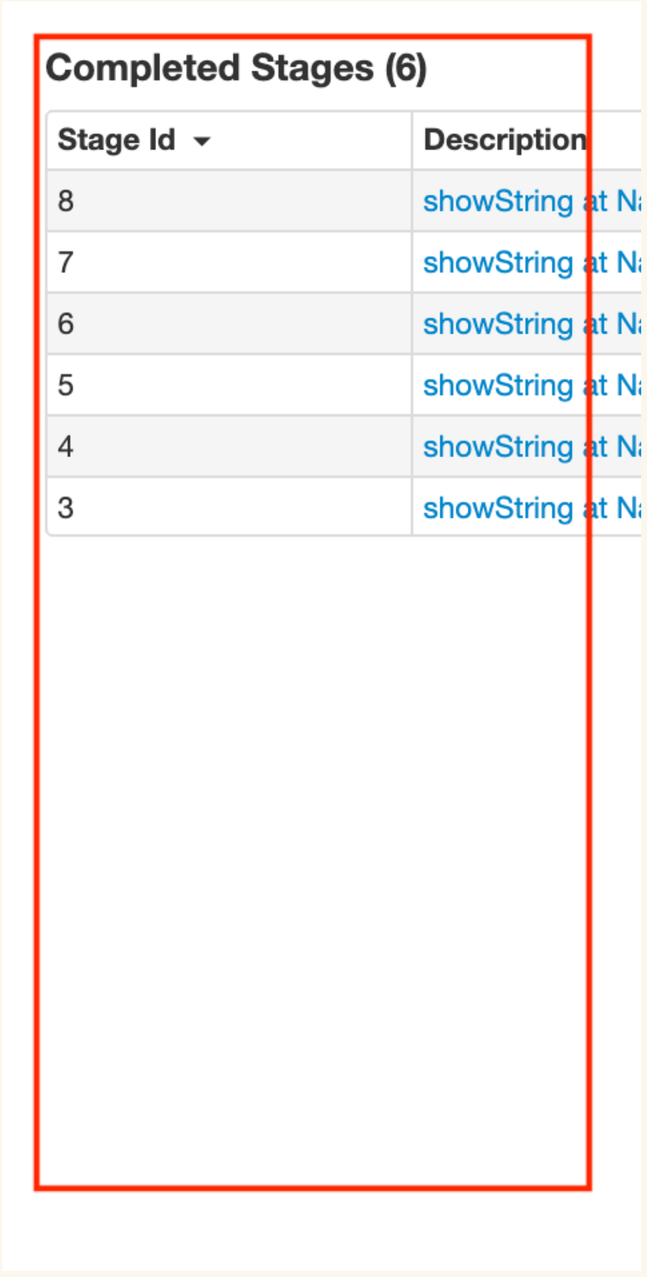

Stage 3
WholeStageCodegen
Exchange
Stage 4
WholeStageCodegen
Exchange
Stage 5
WholeStageCodegen
Exchange
Stage 6
Exchange
Exchange
WholeStageCodegen
WholeStageCodegen
SortMergeJoin
WholeStageCodegen
Exchange
Stage 7
Exchange
Exchange
WholeStageCodegen
WholeStageCodegen
WholeStageCodegen
Exchange
Stage 8
Exchange
WholeStageCodegen
map
takeOrdered

Multiple stages of a stage:

- Pending
- Active
- Completed
- Failed

RED FLAGS:

- Failed stages should be the first thing to check.



	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
	0.4 s	200/200			287.4 KB	
	1.6 min	200/200			25.1 GB	287.4 KB
	12 min	200/200 (1 killed: another attempt succeeded)			29.7 GB	18.6 GB
	3.9 min	1044/1044	16.0 GB			8.5 GB
	3.3 min	14629/14629	86.2 GB			25.1 GB
	12 s	413/413	213.4 MB			151.0 MB

RED FLAGS:

- **Stages with failed tasks.**

tasks.spark.task.maxFailures = 4

Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
0.4 s	200/200			287.4 KB	
1.6 min	200/200			25.1 GB	287.4 KB
12 min	200/200 (1 killed: another attempt succeeded)			29.7 GB	18.6 GB
3.9 min	1044/1044	16.0 GB			8.5 GB
3.3 min	14629/14629	86.2 GB			25.1 GB
12 s	413/413	213.4 MB			151.0 MB

RED FLAGS:

- **Stages with failed failed tasks.**
- **Long running stages.**

Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
0.4 s	200/200			287.4 KB	
1.6 min	200/200			25.1 GB	287.4 KB
12 min	200/200 (1 killed: another attempt succeeded)			29.7 GB	18.6 GB
3.9 min	1044/1044	16.0 GB			8.5 GB
3.3 min	14629/14629	86.2 GB			25.1 GB
12 s	413/413	213.4 MB			151.0 MB

RED FLAGS:

- Stages with failed failed tasks.
- Lots of input data and data being shuffled around.
- Long running stages.

	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
	0.4 s	200/200			287.4 KB	
	1.6 min	200/200			25.1 GB	287.4 KB
	12 min	200/200 (1 killed: another attempt succeeded)			29.7 GB	18.6 GB
	3.9 min	1044/1044	16.0 GB			8.5 GB
	3.3 min	14629/14629	86.2 GB			25.1 GB
	12 s	413/413	213.4 MB			151.0 MB

RED FLAGS:

- Stages with failed failed tasks.
- Long running stages.
- Lots of input data and data being shuffled around.
- Low number of tasks.

Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
0.4 s	200/200			287.4 KB	
1.6 min	200/200			25.1 GB	287.4 KB
12 min	200/200 (1 killed: another attempt succeeded)			29.7 GB	18.6 GB
3.9 min	1044/1044	16.0 GB			8.5 GB
3.3 min	14629/14629	86.2 GB			25.1 GB
12 s	413/413	213.4 MB			151.0 MB

RED FLAGS:

- Stages with failed failed tasks.
- Long running stages.
- Lots of input data and data being shuffled around.
- Low number of tasks.

Stages

- Is a group of transformations.
- Stages are separated by a shuffle of data:
  - join on x
  - group by x
  - window over x
  - etc.
- A stage consists of many tasks.

```
1  # Review of transformations in Spark:
2  df.where(...)
3  df.withColumn(...)
4  df.withRenamedColumn(...)
5
6
7
```

Details for Stage 6 (Attempt 0)

Total Time Across All Tasks: 1.4 h
Locality Level Summary: Node local: 76; Rack local: 125
Shuffle Read: 29.7 GB / 3142989145
Shuffle Write: 18.6 GB / 2750912512
Shuffle Spill (Memory): 62.0 GB
Shuffle Spill (Disk): 15.1 GB

- DAG Visualization
- Show Additional Metrics
- Event Timeline

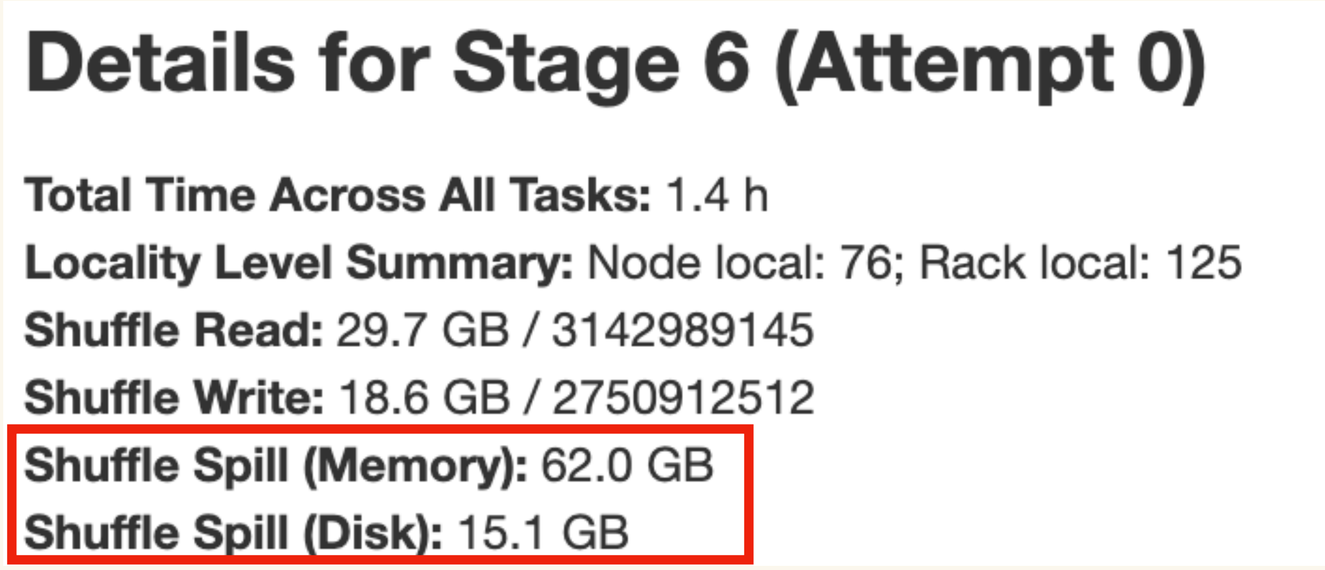
Summary Metrics for 200 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	0 ms	13 s	20 s	29 s	12 min
Scheduler Delay	2 ms	4 ms	5 ms	7 ms	7.5 min
Task Deserialization Time	0 ms	2 ms	3 ms	9 ms	0.2 s
GC Time	91 ms	0.4 s	4 s	8 s	27 s
Result Serialization Time	0 ms	0 ms	0 ms	0 ms	1 ms
Getting Result Time	0 ms	0 ms	0 ms	0 ms	0 ms
Peak Execution Memory	424.0 MB	520.0 MB	552.0 MB	552.0 MB	3.8 GB
Shuffle Read Size / Records	83.9 MB / 6948025	96.0 MB / 8116444	101.6 MB / 8666248	109.8 MB / 9503281	4.5 GB / 643068554
Shuffle Write Size / Records	45.9 MB / 6810020	53.6 MB / 7979098	57.1 MB / 8528805	62.7 MB / 9365457	4.6 GB / 642931319
Shuffle spill (memory)	0.0 B	0.0 B	264.0 MB	264.0 MB	16.4 GB
Shuffle spill (disk)	0.0 B	0.0 B	58.4 MB	59.2 MB	4.3 GB

Aggregated Metrics by Executor

Executor ID ▲	Address	Task Time	Total Tasks	Failed Tasks	Killed Tasks	Succeeded Tasks	Shuffle Read Size / Records	Shuffle Write Size / Records	Shuffle Spill (Memory)	Shuffle Spill (Disk)	Blacklisted
1 stdout stderr	nm-yarn-adhoc-tbm3.yarn-adhoc.starscream-adhoc.yarn.shopifykloud.com:35955	25 min	76	0	1	75	12.5 GB / 1360445397	6.6 GB / 985547990	28.4 GB	6.9 GB	false
2 stdout stderr	nm-yarn-adhoc-5s0m.yarn-adhoc.starscream-adhoc.yarn.shopifykloud.com:37695	8.2 min	23	0	0	23	2.6 GB / 245329024	1618.2 MB / 242167060	4.9 GB	1109.6 MB	false
3 stdout stderr	nm-yarn-adhoc-zq40.yarn-adhoc.starscream-adhoc.yarn.shopifykloud.com:36223	7.1 min	21	0	0	21	2.0 GB / 177559979	1172.5 MB / 174673800	1584.0 MB	354.9 MB	false
4 stdout stderr	nm-yarn-adhoc-32lx.yarn-adhoc.starscream-adhoc.yarn.shopifykloud.com:35857	19 min	16	0	0	16	6.1 GB / 785470154	5.5 GB / 783269709	18.4 GB	4.7 GB	false
5 stdout stderr	nm-yarn-adhoc-32lx.yarn-adhoc.starscream-adhoc.yarn.shopifykloud.com:40177	8.0 min	16	0	0	16	1656.5 MB / 143606734	948.9 MB / 141406594	1848.0 MB	413.0 MB	false
6 stdout stderr	nm-yarn-adhoc-32lx.yarn-adhoc.starscream-adhoc.yarn.shopifykloud.com:35105	7.9 min	16	0	0	16	1623.8 MB / 139477928	918.1 MB / 137279959	2.6 GB	591.2 MB	false
7 stdout stderr	nm-yarn-adhoc-32lx.yarn-adhoc.starscream-adhoc.yarn.shopifykloud.com:36969	7.3 min	17	0	0	17	1730.3 MB / 149373384	986.8 MB / 147038438	1848.0 MB	414.5 MB	false
8 stdout stderr	nm-yarn-adhoc-32lx.yarn-adhoc.starscream-adhoc.yarn.shopifykloud.com:34453	7.9 min	16	0	0	16	1647.4 MB / 141726545	936.3 MB / 139528962	2.6 GB	591.7 MB	false

Tasks (201)



RED FLAGS:

- **Having spill in your stage.**

Stage Level Metrics

Summary Metrics for 200 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	0 ms	13 s	20 s	29 s	12 min
Scheduler Delay	2 ms	4 ms	5 ms	7 ms	7.5 min
Task Deserialization Time	0 ms	2 ms	3 ms	9 ms	0.2 s
GC Time	91 ms	0.4 s	4 s	8 s	27 s
Result Serialization Time	0 ms	0 ms	0 ms	0 ms	1 ms
Getting Result Time	0 ms	0 ms	0 ms	0 ms	0 ms
Peak Execution Memory	424.0 MB	520.0 MB	552.0 MB	552.0 MB	3.8 GB
Shuffle Read Size / Records	83.9 MB / 6948025	96.0 MB / 8116444	101.6 MB / 8666248	109.8 MB / 9503281	4.5 GB / 643068554
Shuffle Write Size / Records	45.9 MB / 6810020	53.6 MB / 7979098	57.1 MB / 8528805	62.7 MB / 9365457	4.6 GB / 642931319
Shuffle spill (memory)	0.0 B	0.0 B	264.0 MB	264.0 MB	16.4 GB
Shuffle spill (disk)	0.0 B	0.0 B	58.4 MB	59.2 MB	4.3 GB

RED FLAGS:

- **Having spill in your stage.**
- **Big difference in the min and max time and memory.**

Stage Level Metrics

Summary Metrics for 200 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	0 ms	13 s	20 s	29 s	12 min
Scheduler Delay	2 ms	4 ms	5 ms	7 ms	7.5 min
Task Deserialization Time	0 ms	2 ms	3 ms	9 ms	0.2 s
GC Time	91 ms	0.4 s	4 s	8 s	27 s
Result Serialization Time	0 ms	0 ms	0 ms	0 ms	1 ms
Getting Result Time	0 ms	0 ms	0 ms	0 ms	0 ms
Peak Execution Memory	424.0 MB	520.0 MB	552.0 MB	552.0 MB	3.8 GB
Shuffle Read Size / Records	83.9 MB / 6948025	96.0 MB / 8116444	101.6 MB / 8666248	109.8 MB / 9503281	4.5 GB / 643068554
Shuffle Write Size / Records	45.9 MB / 6810020	53.6 MB / 7979098	57.1 MB / 8528805	62.7 MB / 9365457	4.6 GB / 642931319
Shuffle spill (memory)	0.0 B	0.0 B	264.0 MB	264.0 MB	16.4 GB
Shuffle spill (disk)	0.0 B	0.0 B	58.4 MB	59.2 MB	4.3 GB

RED FLAGS:

- **Having spill in your stage.**
- **Big difference in the min and max time and memory.**

Stage Level Metrics

Summary Metrics for 200 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	0 ms	13 s	20 s	29 s	12 min
Scheduler Delay	2 ms	4 ms	5 ms	7 ms	7.5 min
Task Deserialization Time	0 ms	2 ms	3 ms	9 ms	0.2 s
GC Time	91 ms	0.4 s	4 s	8 s	27 s
Result Serialization Time	0 ms	0 ms	0 ms	0 ms	1 ms
Getting Result Time	0 ms	0 ms	0 ms	0 ms	0 ms
Peak Execution Memory	424.0 MB	520.0 MB	552.0 MB	552.0 MB	3.8 GB
Shuffle Read Size / Records	83.9 MB / 6948025	96.0 MB / 8116444	101.6 MB / 8666248	109.8 MB / 9503281	4.5 GB / 643068554
Shuffle Write Size / Records	45.9 MB / 6810020	53.6 MB / 7979098	57.1 MB / 8528805	62.7 MB / 9365457	4.6 GB / 642931319
Shuffle spill (memory)	0.0 B	0.0 B	264.0 MB	264.0 MB	16.4 GB
Shuffle spill (disk)	0.0 B	0.0 B	58.4 MB	59.2 MB	4.3 GB

RED FLAGS:

- **Having spill in your stage.**
- **Big difference in the min and max time and memory.**

Executor Level Metrics

Aggregated Metrics by Executor

Executor ID ▲	Address	Task Time	Total Tasks	Failed Tasks	Killed Tasks	Succeeded Tasks
1 stdout stderr	nm-yarn-adhoc-tbm3.yarn-adhoc.starscream-adhoc.yarn.shopifykloud.com:35955	25 min	76	0	1	75
2 stdout stderr	nm-yarn-adhoc-5s0m.yarn-adhoc.starscream-adhoc.yarn.shopifykloud.com:37695	8.2 min	23	0	0	23
3 stdout stderr	nm-yarn-adhoc-zq40.yarn-adhoc.starscream-adhoc.yarn.shopifykloud.com:36223	7.1 min	21	0	0	21
4 stdout stderr	nm-yarn-adhoc-32lx.yarn-adhoc.starscream-adhoc.yarn.shopifykloud.com:35857	19 min	16	0	0	16
5 stdout stderr	nm-yarn-adhoc-32lx.yarn-adhoc.starscream-adhoc.yarn.shopifykloud.com:40177	8.0 min	16	0	0	16
6 stdout stderr	nm-yarn-adhoc-32lx.yarn-adhoc.starscream-adhoc.yarn.shopifykloud.com:35105	7.9 min	16	0	0	16
7 stdout stderr	nm-yarn-adhoc-32lx.yarn-adhoc.starscream-adhoc.yarn.shopifykloud.com:36969	7.3 min	17	0	0	17
8 stdout stderr	nm-yarn-adhoc-32lx.yarn-adhoc.starscream-adhoc.yarn.shopifykloud.com:34453	7.9 min	16	0	0	16

RED FLAGS:

- **Having spill in your stage.**
- **Big difference in the min and max time and memory.**
- **Uneven distribution of work per executor.**

Executor Level Metrics

▾ Aggregated Metrics by Executor

Executor ID ▲	Address	Task Time	Total Tasks	Failed Tasks	Killed Tasks	Succeeded Tasks
1 stdout stderr	nm-yarn-adhoc-tbm3.yarn-adhoc.starscream-adhoc.yarn.shopifykloud.com:35955	25 min	76	0	1	75
2 stdout stderr	nm-yarn-adhoc-5s0m.yarn-adhoc.starscream-adhoc.yarn.shopifykloud.com:37695	8.2 min	23	0	0	23
3 stdout stderr	nm-yarn-adhoc-zq40.yarn-adhoc.starscream-adhoc.yarn.shopifykloud.com:36223	7.1 min	21	0	0	21
4 stdout stderr	nm-yarn-adhoc-32lx.yarn-adhoc.starscream-adhoc.yarn.shopifykloud.com:35857	19 min	16	0	0	16
5 stdout stderr	nm-yarn-adhoc-32lx.yarn-adhoc.starscream-adhoc.yarn.shopifykloud.com:40177	8.0 min	16	0	0	16
6 stdout stderr	nm-yarn-adhoc-32lx.yarn-adhoc.starscream-adhoc.yarn.shopifykloud.com:35105	7.9 min	16	0	0	16
7 stdout stderr	nm-yarn-adhoc-32lx.yarn-adhoc.starscream-adhoc.yarn.shopifykloud.com:36969	7.3 min	17	0	0	17
8 stdout stderr	nm-yarn-adhoc-32lx.yarn-adhoc.starscream-adhoc.yarn.shopifykloud.com:34453	7.9 min	16	0	0	16

RED FLAGS:

- **Having spill in your stage.**
- **Big difference in the min and max time and memory.**
- **Uneven distribution of work per executor.**

Executor Level Metrics

▾ Aggregated Metrics by Executor

Executor ID ▲	Address	Task Time	Total Tasks	Failed Tasks	Killed Tasks	Succeeded Tasks
1 stdout stderr	nm-yarn-adhoc-tbm3.yarn-adhoc.starscream-adhoc.yarn.shopifykloud.com:35955	25 min	76	0	1	75
2 stdout stderr	nm-yarn-adhoc-5s0m.yarn-adhoc.starscream-adhoc.yarn.shopifykloud.com:37695	8.2 min	23	0	0	23
3 stdout stderr	nm-yarn-adhoc-zq40.yarn-adhoc.starscream-adhoc.yarn.shopifykloud.com:36223	7.1 min	21	0	0	21
4 stdout stderr	nm-yarn-adhoc-32lx.yarn-adhoc.starscream-adhoc.yarn.shopifykloud.com:35857	19 min	16	0	0	16
5 stdout stderr	nm-yarn-adhoc-32lx.yarn-adhoc.starscream-adhoc.yarn.shopifykloud.com:40177	8.0 min	16	0	0	16
6 stdout stderr	nm-yarn-adhoc-32lx.yarn-adhoc.starscream-adhoc.yarn.shopifykloud.com:35105	7.9 min	16	0	0	16
7 stdout stderr	nm-yarn-adhoc-32lx.yarn-adhoc.starscream-adhoc.yarn.shopifykloud.com:36969	7.3 min	17	0	0	17
8 stdout stderr	nm-yarn-adhoc-32lx.yarn-adhoc.starscream-adhoc.yarn.shopifykloud.com:34453	7.9 min	16	0	0	16

RED FLAGS:

- **Having spill in your stage.**
- **Big difference in the min and max time and memory.**
- **Uneven distribution of work per executor.**

Executor Level Metrics

▾ Aggregated Metrics by Executor

Executor ID ▲	Address	Task Time	Total Tasks	Failed Tasks	Killed Tasks	Succeeded Tasks
1 stdout stderr	nm-yarn-adhoc-tbm3.yarn-adhoc.starscream-adhoc.yarn.shopifykloud.com:35955	25 min	76	0	1	75
2 stdout stderr	nm-yarn-adhoc-5s0m.yarn-adhoc.starscream-adhoc.yarn.shopifykloud.com:37695	8.2 min	23	0	0	23
3 stdout stderr	nm-yarn-adhoc-zq40.yarn-adhoc.starscream-adhoc.yarn.shopifykloud.com:36223	7.1 min	21	0	0	21
4 stdout stderr	nm-yarn-adhoc-32lx.yarn-adhoc.starscream-adhoc.yarn.shopifykloud.com:35857	19 min	16	0	0	16
5 stdout stderr	nm-yarn-adhoc-32lx.yarn-adhoc.starscream-adhoc.yarn.shopifykloud.com:40177	8.0 min	16	0	0	16
6 stdout stderr	nm-yarn-adhoc-32lx.yarn-adhoc.starscream-adhoc.yarn.shopifykloud.com:35105	7.9 min	16	0	0	16
7 stdout stderr	nm-yarn-adhoc-32lx.yarn-adhoc.starscream-adhoc.yarn.shopifykloud.com:36969	7.3 min	17	0	0	17
8 stdout stderr	nm-yarn-adhoc-32lx.yarn-adhoc.starscream-adhoc.yarn.shopifykloud.com:34453	7.9 min	16	0	0	16

RED FLAGS:

- **Having spill in your stage.**
- **Big difference in the min and max time and memory.**
- **Uneven distribution of work per executor.**

Task Level Metrics

Tasks (201)

Page: 1 2 3 >

3 Pages. Jump to 1 . Show 100 items in a page. Go

Index ▲	ID	Attempt	Status	Locality Level	Executor ID	Host	Launch Time	Duration	Scheduler Delay	Task Deserialization Time	GC Time	Result Serialization Time	Getting Result Time	Peak Execution Memory	Shuffle Read Size / Records	Write Time	Shuffle Write Size / Records	Shuffle Spill (Memory)	Shuffle Spill (Disk)	Errors
0	16089	0	SUCCESS	RACK_LOCAL	4	nm-yarn-adhoc-32lx.yarn-adhoc.starscream-adhoc.yarn.shopifykloud.com stdout stderr	2019/07/25 14:23:12	32 s	6 ms	0.1 s	14 s	0 ms	0 ms	520.0 MB	90.3 MB / 7515616	0.8 s	50.3 MB / 7378423	0.0 B	0.0 B	
1	16090	0	SUCCESS	RACK_LOCAL	7	nm-yarn-adhoc-32lx.yarn-adhoc.starscream-adhoc.yarn.shopifykloud.com stdout stderr	2019/07/25 14:23:12	30 s	2 ms	0.1 s	15 s	0 ms	0 ms	520.0 MB	93.3 MB / 8253612	0.7 s	53.8 MB / 8116264	0.0 B	0.0 B	
2	16091	0	SUCCESS	NODE_LOCAL	1	nm-yarn-adhoc-tbm3.yarn-adhoc.starscream-adhoc.yarn.shopifykloud.com stdout stderr	2019/07/25 14:23:12	26 s	4 ms	0.1 s	5 s	0 ms	0 ms	552.0 MB	122.2 MB / 11025909	0.3 s	73.8 MB / 10888717	264.0 MB	59.4 MB	
3	16092	0	SUCCESS	NODE_LOCAL	1	nm-yarn-adhoc-tbm3.yarn-adhoc.starscream-adhoc.yarn.shopifykloud.com stdout stderr	2019/07/25 14:23:12	23 s	7 ms	95 ms	4 s	0 ms	0 ms	552.0 MB	117.1 MB / 10221419	0.3 s	66.9 MB / 10083829	264.0 MB	59.0 MB	
4	16093	0	SUCCESS	NODE_LOCAL	1	nm-yarn-adhoc-tbm3.yarn-adhoc.starscream-adhoc.yarn.shopifykloud.com stdout stderr	2019/07/25 14:23:12	20 s	3 ms	89 ms	3 s	0 ms	0 ms	552.0 MB	101.3 MB / 8654774	0.8 s	57.1 MB / 8517328	264.0 MB	59.1 MB	
5	16094	0	SUCCESS	NODE_LOCAL	1	nm-yarn-adhoc-tbm3.yarn-adhoc.starscream-adhoc.yarn.shopifykloud.com stdout stderr	2019/07/25 14:23:12	21 s	4 ms	48 ms	3 s	0 ms	0 ms	552.0 MB	122.1 MB / 10710569	0.3 s	71.1 MB / 10573267	264.0 MB	59.4 MB	
6	16095	0	SUCCESS	NODE_LOCAL	1	nm-yarn-adhoc-tbm3.yarn-adhoc.starscream-adhoc.yarn.shopifykloud.com stdout	2019/07/25 14:23:12	17 s	4 ms	2 ms	3 s	0 ms	0 ms	520.0 MB	93.6 MB / 7715981	0.2 s	51.5 MB / 7578841	0.0 B	0.0 B	

Duration	Scheduler Delay	Task Deserialization Time	GC Time	Result Serialization Time	Getting Result Time	Peak Execution Memory	Shuffle Read Size / Records
32 s	6 ms	0.1 s	14 s	0 ms	0 ms	520.0 MB	90.3 MB / 7515616
30 s	2 ms	0.1 s	15 s	0 ms	0 ms	520.0 MB	93.3 MB / 8253612
26 s	4 ms	0.1 s	5 s	0 ms	0 ms	552.0 MB	122.2 MB / 11025909
23 s	7 ms	95 ms	4 s	0 ms	0 ms	552.0 MB	117.1 MB / 10221419
20 s	3 ms	89 ms	3 s	0 ms	0 ms	552.0 MB	101.3 MB / 8654774
21 s	4 ms	48 ms	3 s	0 ms	0 ms	552.0 MB	122.1 MB / 10710569

RED FLAGS:

- **Having spill in your stage.**
- **Big difference in the min and max time and memory.**
- **Uneven distribution of work per executor.**
- **Tasks with more than 128MB of Shuffle Read data.**

Duration	Scheduler Delay	Task Deserialization Time	GC Time	Result Serialization Time	Getting Result Time	Peak Execution Memory	Shuffle Read Size / Records
32 s	6 ms	0.1 s	14 s	0 ms	0 ms	520.0 MB	90.3 MB / 7515616
30 s	2 ms	0.1 s	15 s	0 ms	0 ms	520.0 MB	93.3 MB / 8253612
26 s	4 ms	0.1 s	5 s	0 ms	0 ms	552.0 MB	122.2 MB / 11025909
23 s	7 ms	95 ms	4 s	0 ms	0 ms	552.0 MB	117.1 MB / 10221419
20 s	3 ms	89 ms	3 s	0 ms	0 ms	552.0 MB	101.3 MB / 8654774
21 s	4 ms	48 ms	3 s	0 ms	0 ms	552.0 MB	122.1 MB / 10710569

RED FLAGS:

- **Having spill in your stage.**
- **Big difference in the min and max time and memory.**
- **Uneven distribution of work per executor.**
- **Tasks with more than 128MB of Shuffle Read data.**

Duration	Scheduler Delay	Task Deserialization Time	GC Time	Result Serialization Time	Getting Result Time	Peak Execution Memory	Shuffle Read Size / Records
32 s	6 ms	0.1 s	14 s	0 ms	0 ms	520.0 MB	90.3 MB / 7515616
30 s	2 ms	0.1 s	15 s	0 ms	0 ms	520.0 MB	93.3 MB / 8253612
26 s	4 ms	0.1 s	5 s	0 ms	0 ms	552.0 MB	122.2 MB / 11025909
23 s	7 ms	95 ms	4 s	0 ms	0 ms	552.0 MB	117.1 MB / 10221419
20 s	3 ms	89 ms	3 s	0 ms	0 ms	552.0 MB	101.3 MB / 8654774
21 s	4 ms	48 ms	3 s	0 ms	0 ms	552.0 MB	122.1 MB / 10710569

RED FLAGS:

- **Having spill in your stage.**
- **Big difference in the min and max time and memory.**
- **Uneven distribution of work per executor.**
- **Tasks with more than 128MB of Shuffle Read data.**

After Skew Fix

Summary Metrics for 200 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	14 s	16 s	20 s	25 s	45 s
Scheduler Delay	1 ms	3 ms	4 ms	5 ms	16 ms
Task Deserialization Time	1 ms	2 ms	3 ms	16 ms	35 ms
GC Time	0.1 s	0.8 s	2 s	3 s	14 s
Result Serialization Time	0 ms	0 ms	0 ms	0 ms	1 ms
Getting Result Time	0 ms	0 ms	0 ms	0 ms	0 ms
Peak Execution Memory	0.0 B	0.0 B	0.0 B	0.0 B	0.0 B
Shuffle Read Size / Records	128.6 MB / 17139979	128.7 MB / 17149116	128.7 MB / 17151913	128.7 MB / 17155092	128.8 MB / 17164081
Shuffle Write Size / Records	1403.0 B / 19	1477.0 B / 20	1478.0 B / 20	1479.0 B / 20	1558.0 B / 21
Shuffle spill (memory)	192.0 MB	384.0 MB	384.0 MB	384.0 MB	736.0 MB
Shuffle spill (disk)	66.0 MB	97.9 MB	98.0 MB	98.0 MB	98.0 MB

Varying task times and data volume has been fixed!

After Skew Fix

Summary Metrics for 200 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	14 s	16 s	20 s	25 s	45 s
Scheduler Delay	1 ms	3 ms	4 ms	5 ms	16 ms
Task Deserialization Time	1 ms	2 ms	3 ms	16 ms	35 ms
GC Time	0.1 s	0.8 s	2 s	3 s	14 s
Result Serialization Time	0 ms	0 ms	0 ms	0 ms	1 ms
Getting Result Time	0 ms	0 ms	0 ms	0 ms	0 ms
Peak Execution Memory	0.0 B	0.0 B	0.0 B	0.0 B	0.0 B
Shuffle Read Size / Records	128.6 MB / 17139979	128.7 MB / 17149116	128.7 MB / 17151913	128.7 MB / 17155092	128.8 MB / 17164081
Shuffle Write Size / Records	1403.0 B / 19	1477.0 B / 20	1478.0 B / 20	1479.0 B / 20	1558.0 B / 21
Shuffle spill (memory)	192.0 MB	384.0 MB	384.0 MB	384.0 MB	736.0 MB
Shuffle spill (disk)	66.0 MB	97.9 MB	98.0 MB	98.0 MB	98.0 MB

Varying task times and data volume has been fixed!

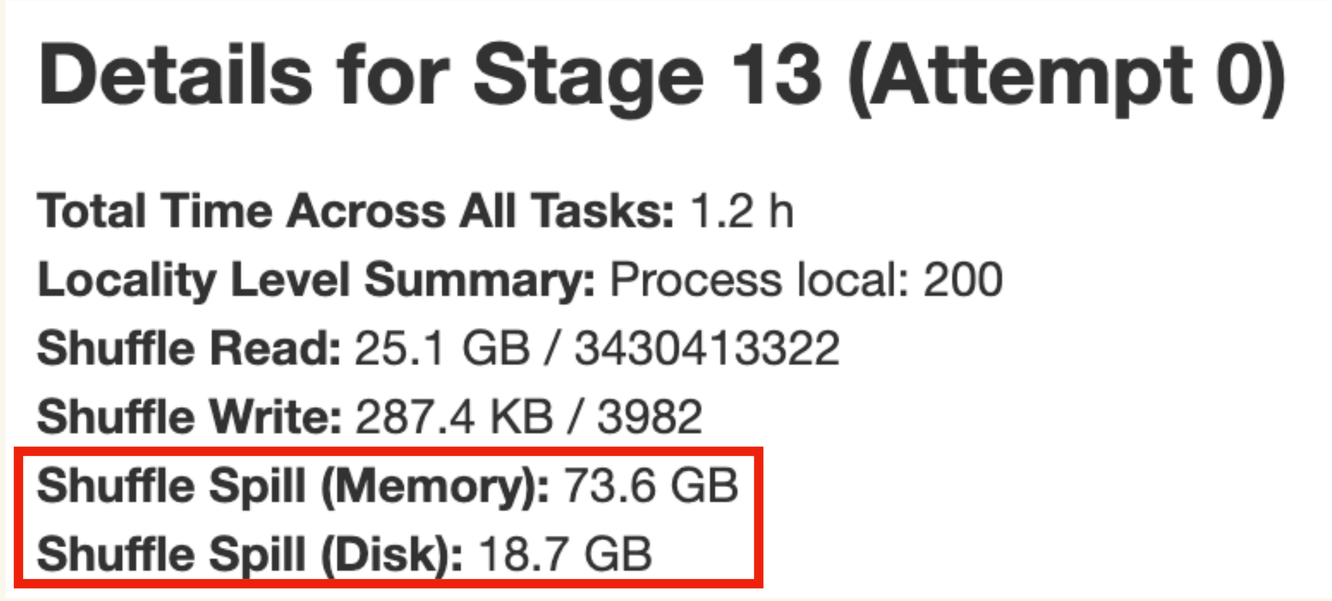
After Skew Fix

Summary Metrics for 200 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	14 s	16 s	20 s	25 s	45 s
Scheduler Delay	1 ms	3 ms	4 ms	5 ms	16 ms
Task Deserialization Time	1 ms	2 ms	3 ms	16 ms	35 ms
GC Time	0.1 s	0.8 s	2 s	3 s	14 s
Result Serialization Time	0 ms	0 ms	0 ms	0 ms	1 ms
Getting Result Time	0 ms	0 ms	0 ms	0 ms	0 ms
Peak Execution Memory	0.0 B	0.0 B	0.0 B	0.0 B	0.0 B
Shuffle Read Size / Records	128.6 MB / 17139979	128.7 MB / 17149116	128.7 MB / 17151913	128.7 MB / 17155092	128.8 MB / 17164081
Shuffle Write Size / Records	1403.0 B / 19	1477.0 B / 20	1478.0 B / 20	1479.0 B / 20	1558.0 B / 21
Shuffle spill (memory)	192.0 MB	384.0 MB	384.0 MB	384.0 MB	736.0 MB
Shuffle spill (disk)	66.0 MB	97.9 MB	98.0 MB	98.0 MB	98.0 MB

Varying task times and data volume has been fixed!

After Skew Fix



Still some spill of data though :(.

Goal 2

Reducing the shuffle spilt to disk.

Motivation:

“When the records destined for these aggregation operations do not easily fit in memory, some mayhem can ensue. First, holding many records in these data structures puts pressure on garbage collection, which can lead to pauses down the line. Second, when the records do not fit in memory, Spark will spill them to disk, which causes disk I/O and sorting. This overhead during large shuffles is probably the number one cause of job stalls I have seen at Cloudera customers.”

Reference: https://blog.cloudera.com/blog/2015/03/how-to-tune-your-apache-spark-jobs-part-2/

Tasks

- The smallest unit of transformation(s) applied on a unit of data.
- Tasks run on executors within a stage.
- Number of tasks per stage is determined by various settings.

Partitions & Tasks

Number of tasks for a given stage can be:

spark.sql.shuffle.partitions = 200

1. If partition number is not tweaked:
    - If stage is reading input data:
        - Number of tasks ~= total number of “blocks” in all the files.
    - Else:
        - Default value of 200 partitions / stage.
2. If partition number is tweaked:
    - Partitions passed to the repartition / coalesce function.
    - Min(cardinality of a dataset, number of partitions argument).

Partitions & Tasks

Adding More Partitions / Tasks:

- Smaller manageable amounts of data to process at a time.
- Data (hopefully) fit in memory on the executor.

DISCLAIMER:

- More tasks also means more load on the driver.

RED FLAGS:

- Lots of shuffle spill.
- Driver OOMs —> too many partitions / tasks.

Review:

Tasks in a stage are processed sequentially by executors.

Choosing an Optimal Number of Tasks

For the most optimal setup, you want to dynamically change the number of partitions on a stage level, as it will differ stage by stage.

But for simplicity sacks we will find an overarching number for the entire job.

Steps:

1. Find the non read input stage with shuffle spill data.
2. Rule of thumb, ideal task data size is 128MB.
3. Add spark.sql.shuffle.partitions: ideal_number to schedule file.

Calculating Ideal Task Number

(1) Memory available for a given task:

$$\frac{\text{Java memory per executor} * \text{margin of error}}{\text{cores per executor}}$$

Margin of error = 0.8 * 0.2 = 0.16

(2) In memory size of shuffle data:

$$\frac{\text{Shuffle spill (memory)} * \text{Shuffle write}}{\text{Shuffle spill (disk)}}$$

reference: https://blog.cloudera.com/blog/2015/03/how-to-tune-your-apache-spark-jobs-part-2/

Calculating Ideal Task Number

Putting the 2 equations together:

$$\frac{\text{Shuffle spill (memory)} * \text{Shuffle write} * \text{cores per executor}}{\text{Shuffle spill (disk)} * \text{java memory per executor} * \text{margin of error}}$$

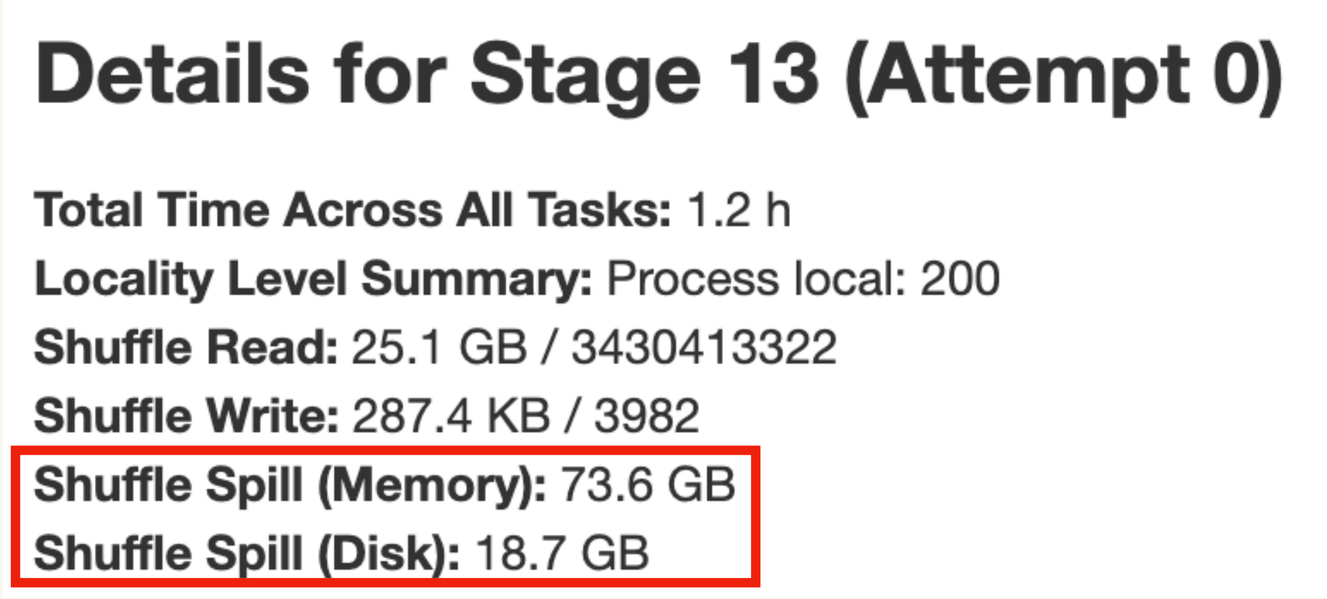
reference: https://blog.cloudera.com/blog/2015/03/how-to-tune-your-apache-spark-jobs-part-2/

All else Fails...

but.... This equation is not bullet proof..

If this equation gives a smaller estimate than the current number of tasks, then we can only guess and check. Increase the number of tasks by 1.5x incrementally until no shuffle spill is observed.

After Skew Fix



Notebooks resource class:

- **8 executors**
- **12GB java memory**

After Skew Fix

$$\frac{\text{Shuffle spill (memory)} * \text{Shuffle write} * \text{cores per executor}}{\text{Shuffle spill (disk)} * \text{java memory per executor} * \text{margin of error}}$$

$$\frac{(73.6\ \text{GB} * 287.4\ \text{KB} * 8)}{(18.7\ \text{GB} * 12\ \text{GB} * 0.8)}$$

0.00094263101 partitions...

So let's try 200 * 1.5 = 300 partitions.

Results

Tasks: Succeeded/Total
300/300
300/300
1044/1044
14934/14934
14634/14634
413/413

Details for Stage 19 (Attempt 0)

Total Time Across All Tasks: 54 min
Locality Level Summary: Process local: 300
Shuffle Read: 25.0 GB / 3435988435
Shuffle Write: 425.5 KB / 5896

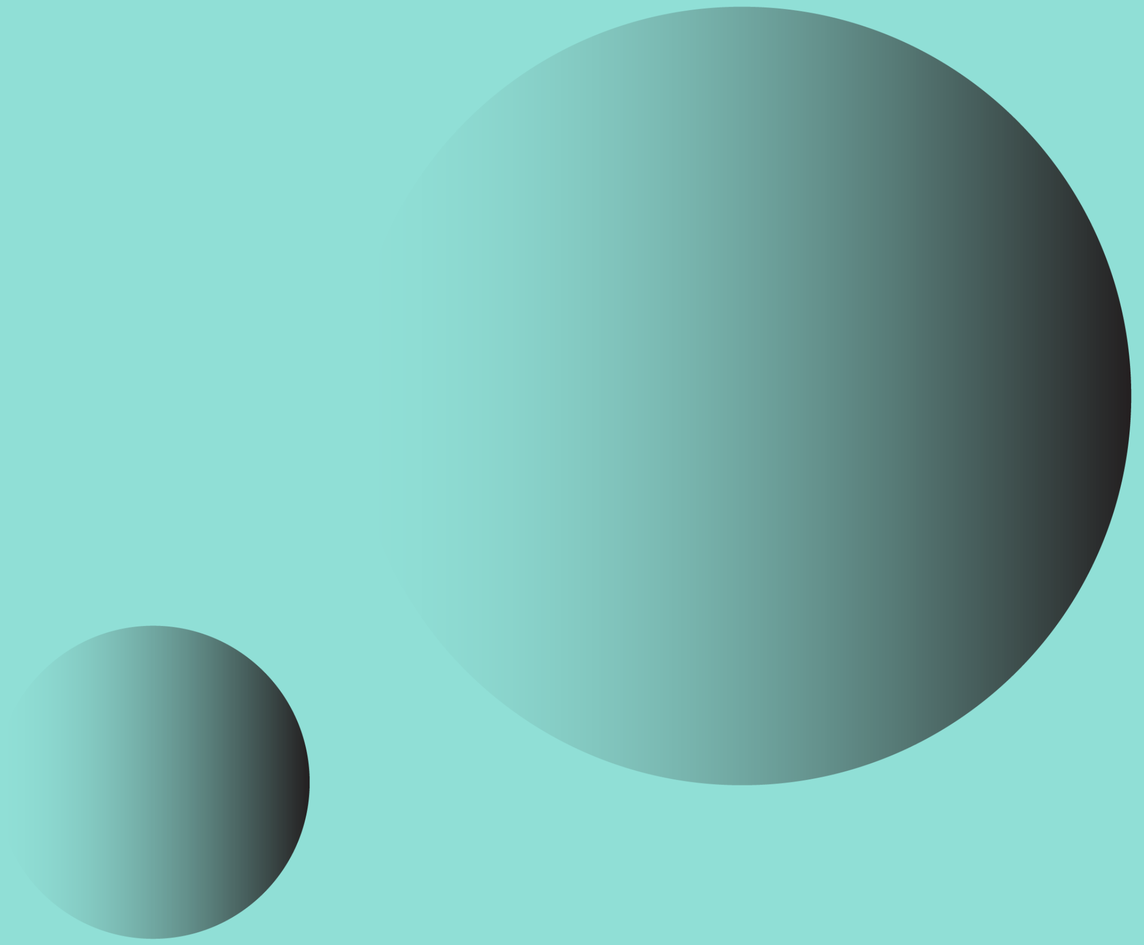
▸ DAG Visualization

Section 5: Recap

Recap

1. Understood how a Spark DataFrames application got executed.
2. Evaluated the Spark application by analyzing the SparkUI.
3. Identified bottlenecks.
4. Implemented optimizations to address the bottlenecks.

Questions?



Thank You