# How to use Impala query plan and profile to fix performance issues

Juan Yu

Impala Field Engineer, Cloudera

Strata
DATA CONFERENCE

# Profiles?! Explain plans!? Argggghh…

- For the end user, **understanding Impala performance is like…**

  - Lots of commonality between requests, *e.g.*
    - *What's the bottleneck for this query?*
    - *Why this run is fast but that run is slow?*
    - *How can I tune to improve this query's performance.*

# Agenda

- What are query plan and profile
- What kind of issues query plan and profile can help you solve
- Structure of query plan and profile
- Basic troubleshooting
- Advanced query tuning and troubleshooting

# Why did the following queries take different time?

**SELECT AVG(**ss_list_price**) FROM** store_sales;

Fetched 1 row(s) in **3.60s**

**SELECT AVG**(ss_list_price) **FROM** store_sales **WHERE** ss_sold_date_sk **BETWEEN** 2451959 **AND** 2451989;

Fetched 1 row(s) in **0.28s**

| Operator | #Hosts | Avg Time | Max Time | #Rows | Est. |
|---|---|---|---|---|---|
| 03:AGGREGATE | 1 | 870.153us | 870.153us | 1 | |
| 02:EXCHANGE | 1 | 3s245ms | 3s245ms | 8 | |
| 01:AGGREGATE | 8 | 1s340ms | 1s380ms | 8 | |
| 00:SCAN HDFS | 8 | 1s589ms | 1s838ms | 864.00M | 8 |

| Avg Time | Max Time | #Rows | Est. #Rows |
|---|---|---|---|
| 842.478us | 842.478us | 1 | 1 |
| 170.752ms | 170.752ms | 8 | 1 |
| 18.297ms | 20.736ms | 8 | 1 |
| 93.929ms | 122.292ms | 7.84M | 7.84M |

Partition Pruning reduces scan work and intermediate result.

```
00:SCAN HDFS [tpcds_300_decimal_parquet.store_sales
   partitions=1824/1824 files=1829 size=38.81GB
   stats-rows=864001869 extrapolated-rows=disabled
   table stats: rows=864001869 size=38.81GB
   column stats: all
   mem-estimate=32.00MB mem-reservation=0B
   tuple-ids=0 row-size=4B cardinality=864001869
```

```
00:SCAN HDFS [tpcds_300_decimal_parquet.store_sales
   partitions=30/1824 files=30 size=369.51MB
   stats-rows=7844644 extrapolated-rows=disabled
   table stats: rows=864001869 size=38.81GB
   column stats: all
   mem-estimate=24.00MB mem-reservation=0B
   tuple-ids=0 row-size=4B cardinality=7844644
```

# Where to get Query plan and profile

- Cloudera Manager Query history page
- Impala webUI queries page
- Impala-shell

- Profile examples: https://github.com/yjwater/strata-sj

# Query Planning: Goals

- Lower execution cost and get better performance
- Reduce unnecessary work as much as possible by partition pruning, predicate pushdown, runtime filter,etc.
- Maximize scan locality using DN block metadata.
- Minimize data movement (optimize join order, choose better join strategy)
- Parallel tasks and run them on many nodes to shorten execution time.

# Query profile - Execution Results

- Provide metrics and counters to show
  - how execution is ongoing.
  - how much resources are used.
  - how long each piece takes.
  - Is there any bottleneck and/or abnormal behavior.

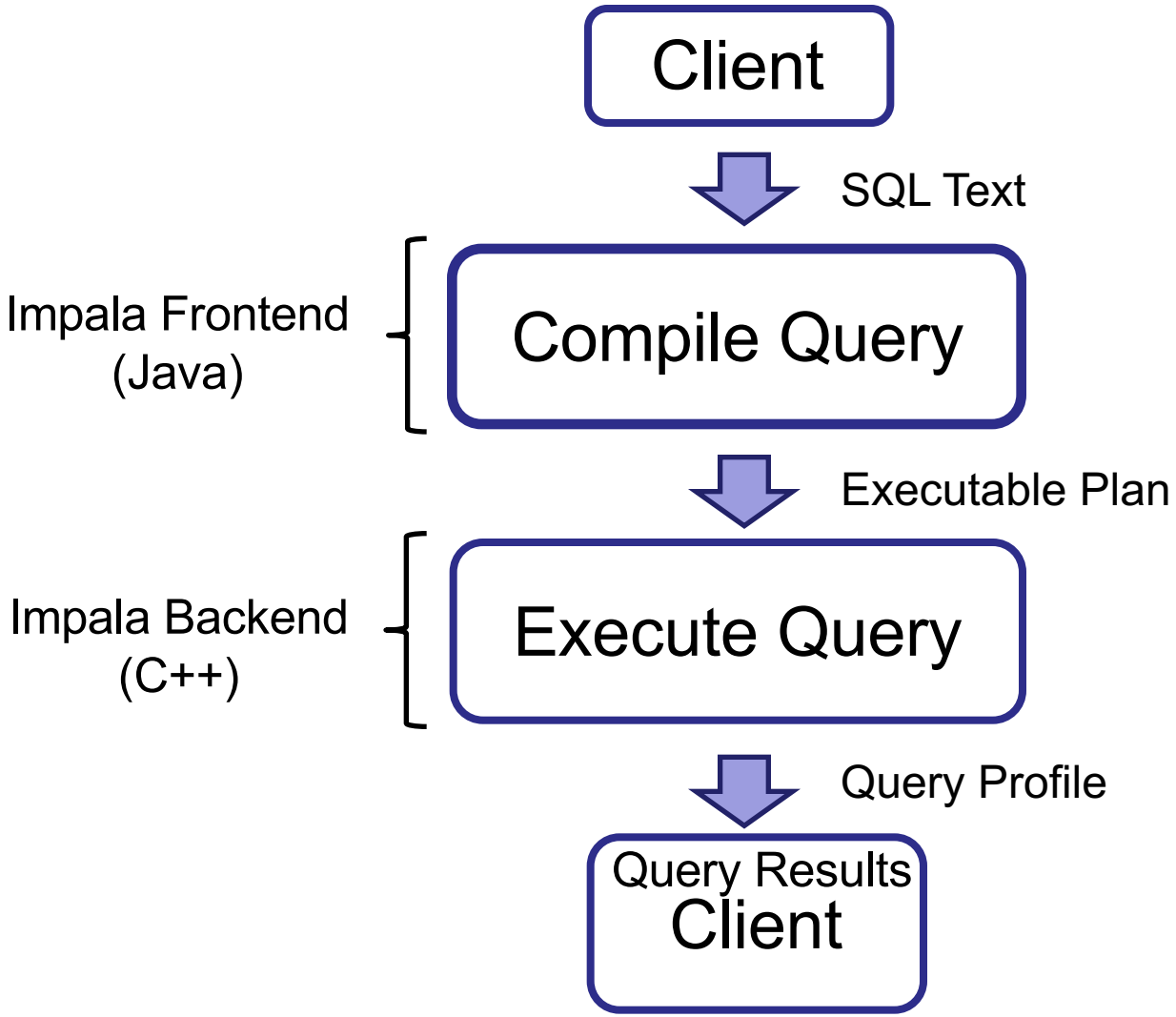
# What issues query plan and profile can help you solve?

- **Plan:**
  - Missing stats
  - Partition pruning
  - Predicate pushdown
  - Join order
  - Join strategy
  - Parallelism

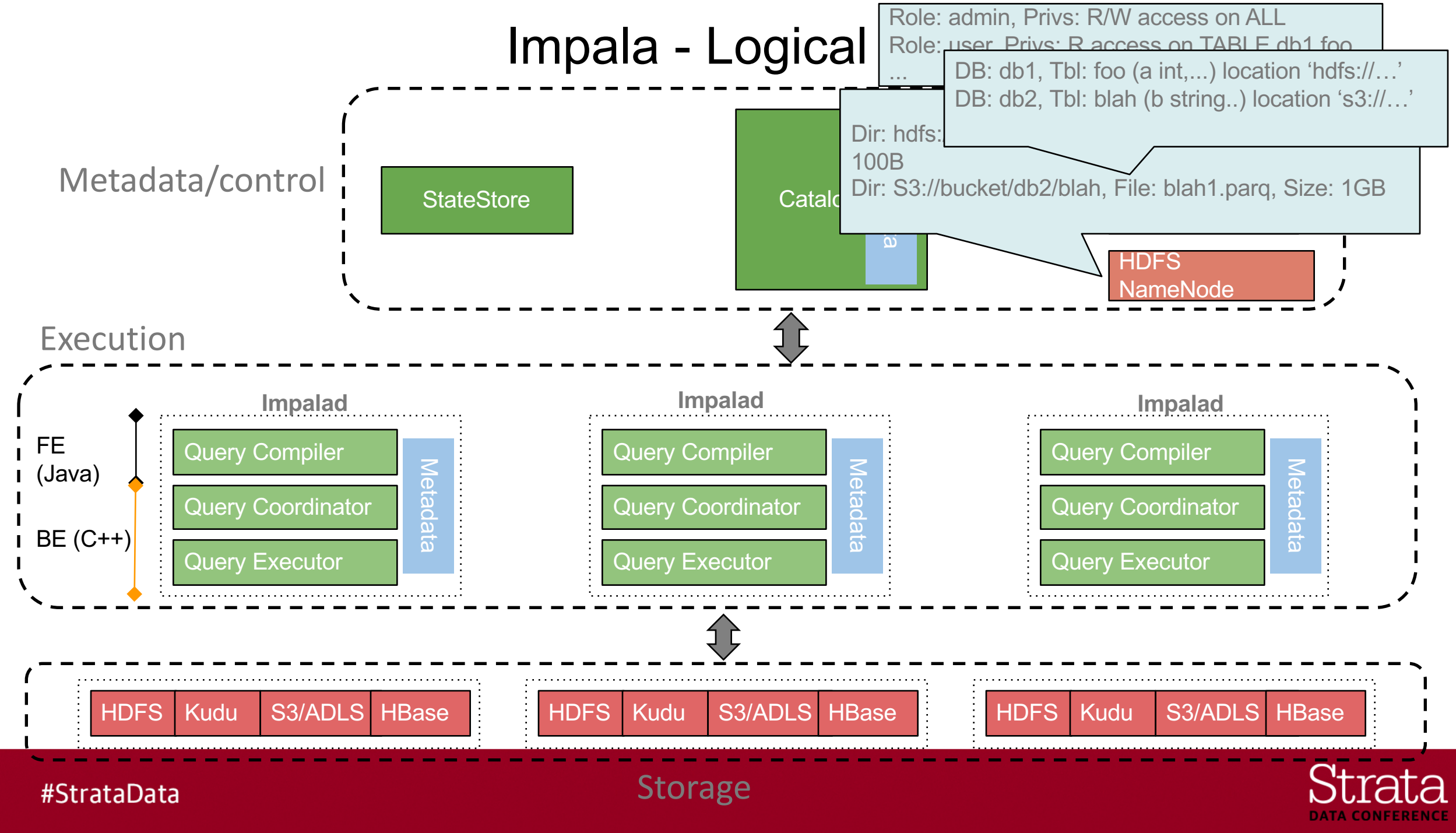
- **Profile:**
  - Identify Bottleneck
  - Runtime filter effectiveness
  - Memory usage/Spill to disk
  - Network slowness
  - Skew

  - Client side issues
  - Metadata loading

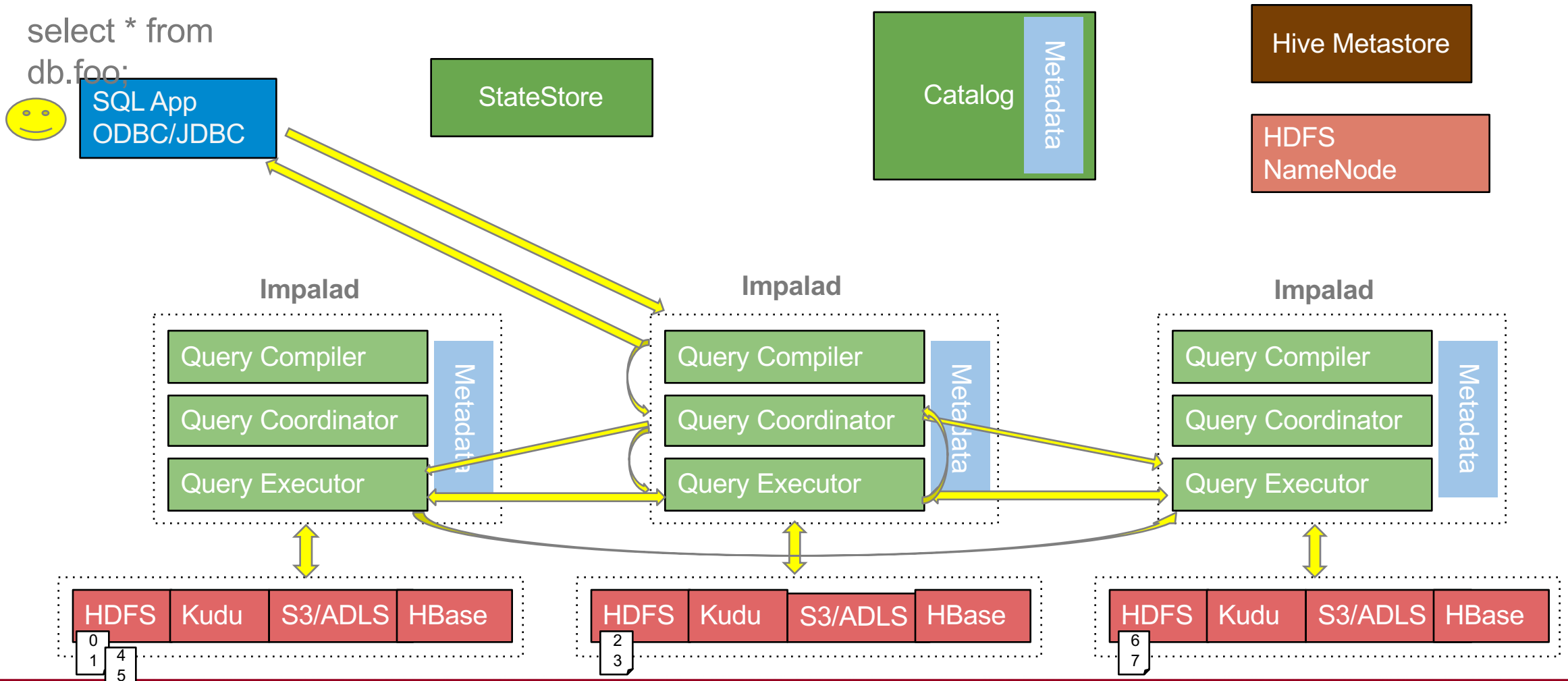
# Flow of a SQL Query



# Impala - Logical



# Impala in action - Select query



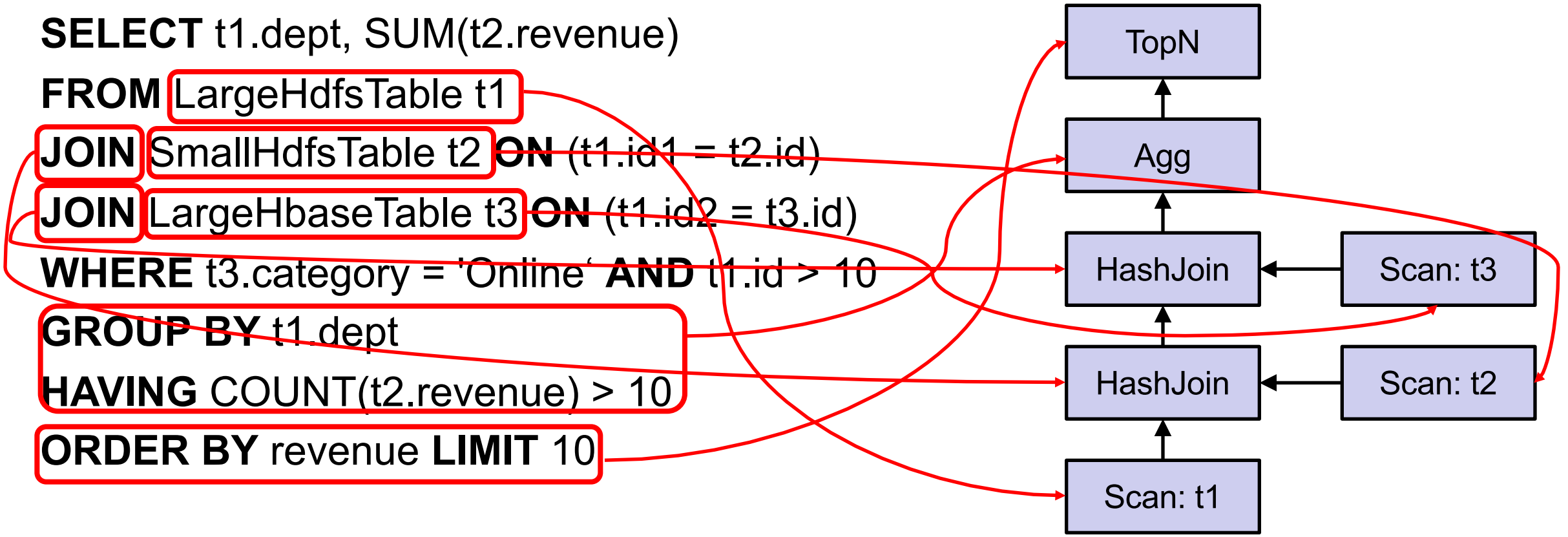
# Query Planning

- Single-Node Plan (SET NUM_NODES=1):
  - Assigns predicates to lowest plan node.
  - Prunes irrelevant columns and partitions (if applicable).
  - Optimizes join order.
  - Determine effective runtime filters

- Distributed Plan:
  - Provides list of best Impalads to do the scan work.
  - Picks an execution strategy for each join (broadcast vs hash-partitioned)
  - Introduces exchange operators and groups nodes in plan fragments (unit of work).

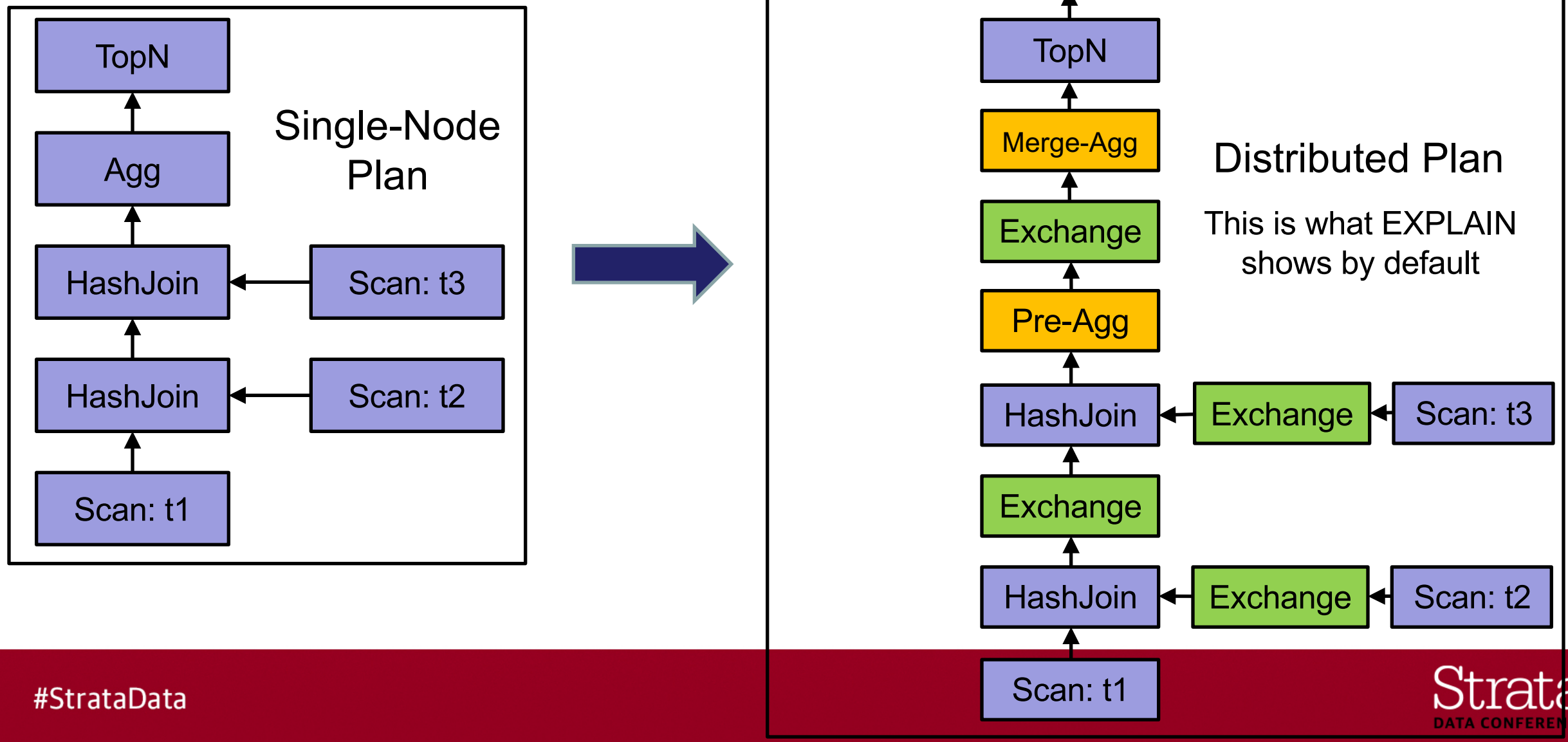
# Compile query

```
SELECT t1.dept, SUM(t2.revenue)
FROM LargeHdfsTable t1
JOIN SmallHdfsTable t2 ON (t1.id1 = t2.id)
JOIN LargeHbaseTable t3 ON (t1.id2 = t3.id)
WHERE t3.category = 'Online' AND t1.id > 10
GROUP BY t1.dept
HAVING COUNT(t2.revenue) > 10
ORDER BY revenue LIMIT 10
```

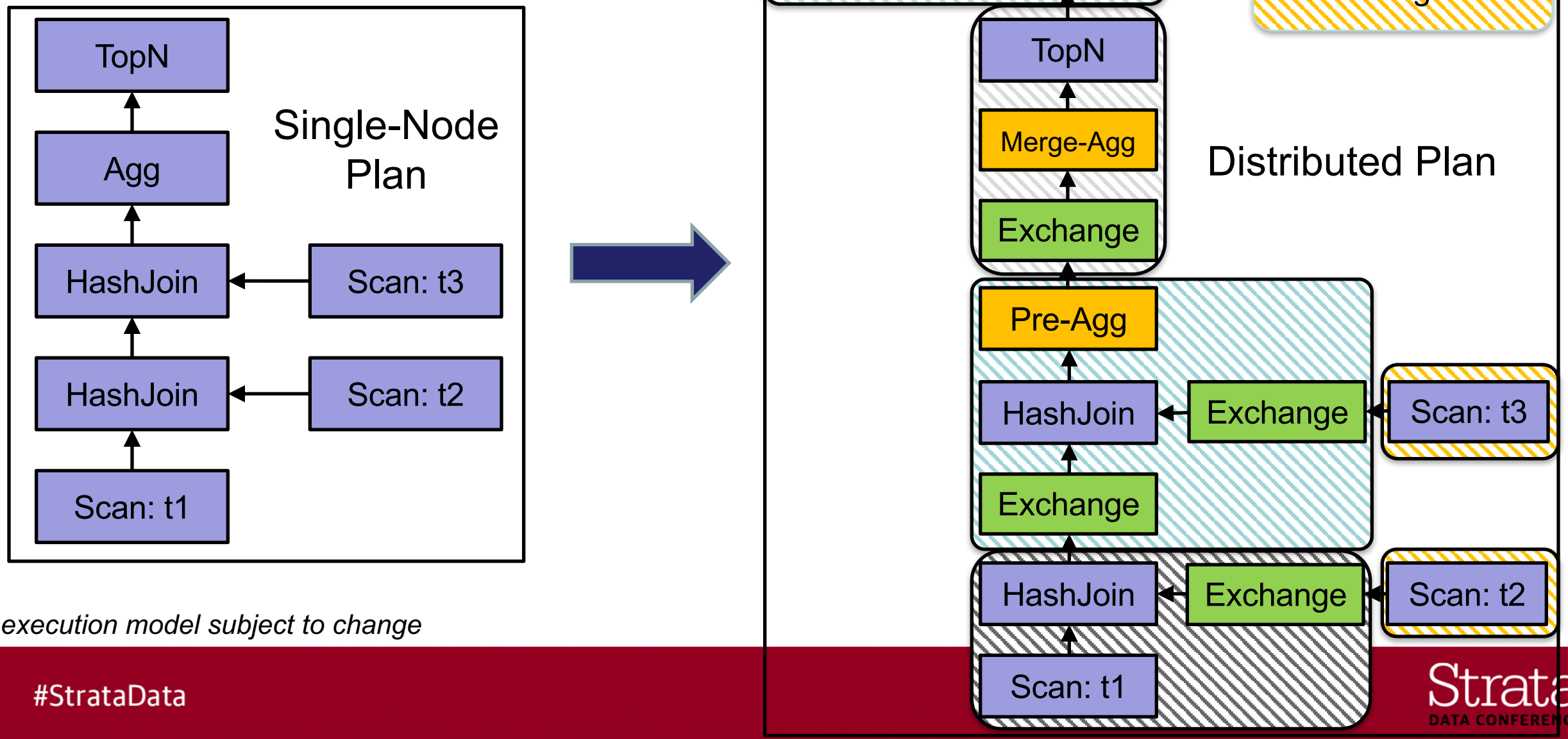



*Hint:* set num_nodes=1; to explain the single node plan

# Single to Distributed Node Plan



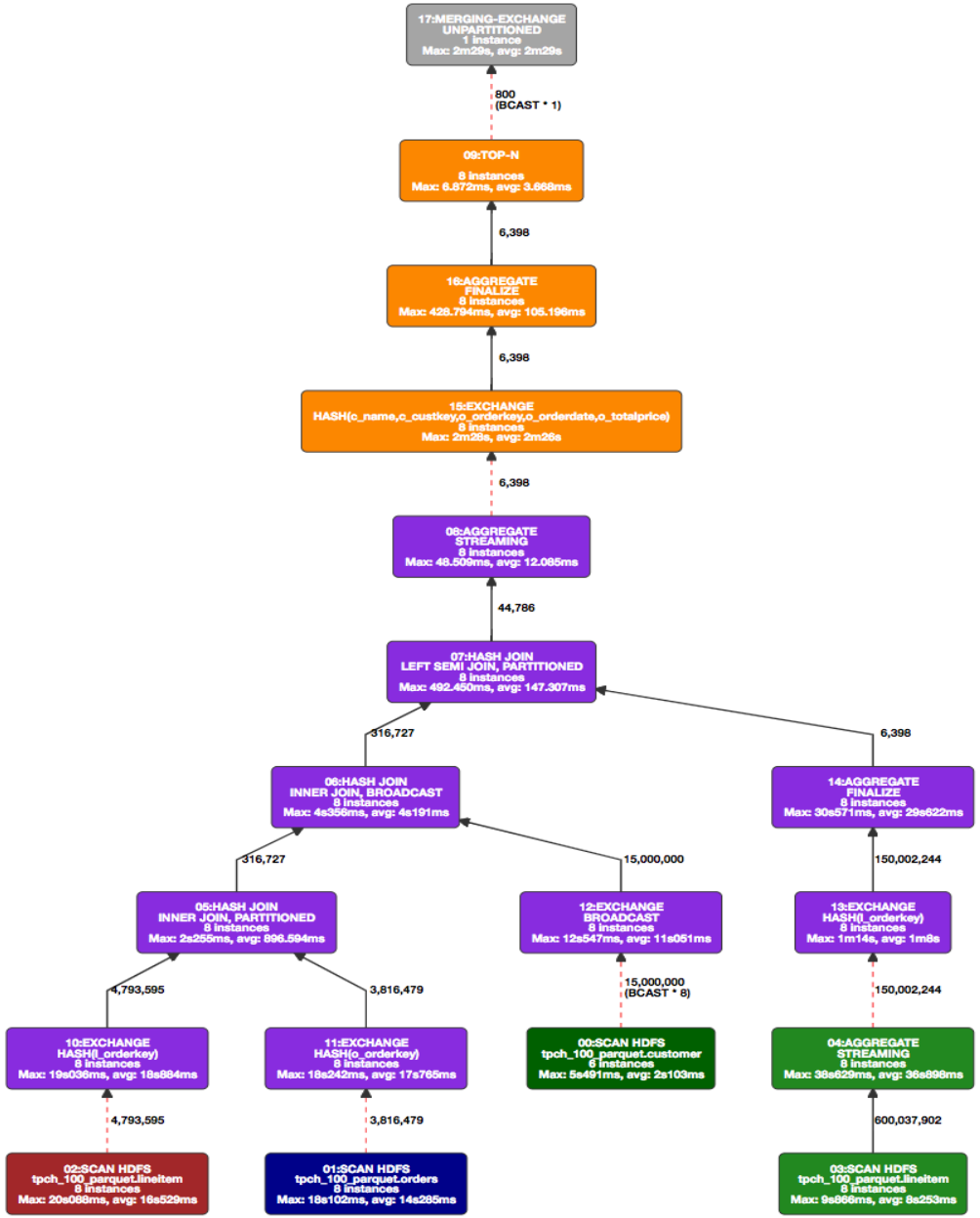
# Plan Fragmentation




*execution model subject to change*

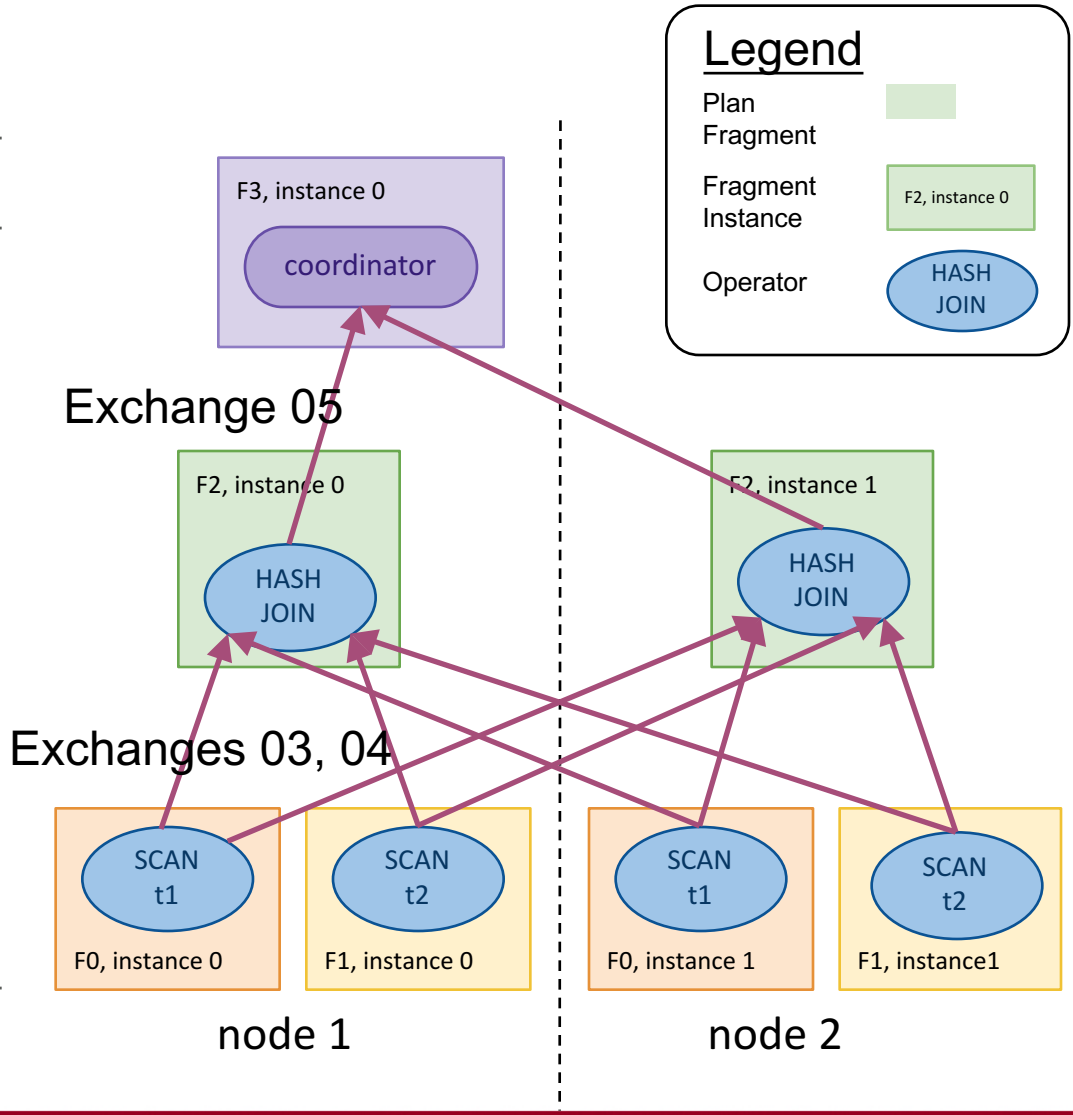
# Query Plan Visualization

```
select c_name, c_custkey, o_orderkey, o_orderdate,
        o_totalprice, sum(l_quantity)
from customer, orders, lineitem
where o_orderkey in (
        select l_orderkey
        from lineitem
        group by l_orderkey
        having sum(l_quantity) > 300 )
and c_custkey = o_custkey
and o_orderkey = l_orderkey
group by c_name, c_custkey, o_orderkey, o_orderdate,
        o_totalprice
order by o_totalprice desc, o_orderdate
limit 100
```



# Query Execution

```
explain SELECT * FROM t1 JOIN [shuffle] t2 ON
t1.id = t2.id;
+-----------------------------------------------------------+
| Explain String                                            |
+-----------------------------------------------------------+
| Estimated Per-Host Requirements: Memory=240.00MB VCores=2 |
|                                                           |
| 05:EXCHANGE [UNPARTITIONED]                               |
| |                                                         |
| 02:HASH JOIN [INNER JOIN, PARTITIONED]                    |
| |  hash predicates: t1.id = t2.id                         |
| |                                                         |
| |--04:EXCHANGE [HASH(t2.id)]                              |
| |  |                                                      |
| |  01:SCAN HDFS [functional.alltypestiny t2]              |
| |     partitions=4/4 files=4 size=460B                    |
| |                                                         |
| 03:EXCHANGE [HASH(t1.id)]                                 |
| |                                                         |
| 00:SCAN HDFS [functional.alltypes t1]                     |
|    partitions=24/24 files=24 size=478.45KB                |
+-----------------------------------------------------------+
```



# Plan and Profile structure

```
+------------------------------------------------------------------------------+
| Explain String                                                               |
+------------------------------------------------------------------------------+
| Max Per-Host Resource Reservation: Memory=0B                                 |
| Per-Host Resource Estimates: Memory=52.00MB                                  |
| Codegen disabled by planner                                                  |
|                                                                              |
| F01:PLAN FRAGMENT [UNPARTITIONED] hosts=1 instances=1                        |
| |  Per-Host Resources: mem-estimate=10.00MB mem-reservation=0B               |
| PLAN-ROOT SINK                                                               |
| |  mem-estimate=0B mem-reservation=0B                                        |
| |                                                                            |
| 03:AGGREGATE [FINALIZE]                                                      |
| |  output: avg:merge(salary)                                                 |
| |  mem-estimate=10.00MB mem-reservation=0B spill-buffer=2.00MB               |
| |  tuple-ids=2 row-size=8B cardinality=1                                     |
| |                                                                            |
| 02:EXCHANGE [UNPARTITIONED]                                                  |
| |  mem-estimate=0B mem-reservation=0B                                        |
| |  tuple-ids=1 row-size=8B cardinality=1                                     |
| |                                                                            |
| F00:PLAN FRAGMENT [RANDOM] hosts=1 instances=1                               |
| Per-Host Resources: mem-estimate=42.00MB mem-reservation=0B                  |
| 01:AGGREGATE                                                                 |
| |  output: avg(salary)                                                       |
| |  mem-estimate=10.00MB mem-reservation=0B spill-buffer=2.00MB               |
| |  tuple-ids=1 row-size=8B cardinality=1                                     |
| |                                                                            |
```

explain_level = 3

**Query Summary**
- |- Basic info: state, type, user, statement, coordinator
- |- Query plan
- |- Execution summary
- |- Timeline

**Client side info**

**Execution details**
- |- Runtime Filter table
- |- Coordinator Fragment
  - |- Instance
  - |- Operator node A
  - …
- |- Average Fragment 3
  - |- Fragment instance 0
    - |- Operator node B
    - ...
  - |- Fragment instance 1
  - ...
- |- Average Fragment 2
- |- Average Fragment 0

# Basic Query information

```
Query (id=5349daec57a4d786:9536a60900000000):
  Summary:
    Session ID: 5245f03b5b3c17ec:1dbd50ff83471f95
    Session Type: HIVESERVER2
    HiveServer2 Protocol Version: V6
    Start Time: 2018-02-09 13:17:31.162274000
    End Time: 2018-02-09 13:20:05.281900000
    Query Type: QUERY
    Query State: FINISHED
    Query Status: OK
    Impala Version: impalad version 2.11.0-cdh5.14.0
    User: REDACTED
    Connected User: REDACTED
    Delegated User:
    Network Address: REDACTED:54129
    Default Db: tpch_100_parquet
    Sql Statement: select * from lineitems limit 100
    Coordinator: REDACTED:22000
    Query Options (set by configuration): ABORT_ON_ERROR=1,MEM_LIMIT=5658116096
    Query Options (set by configuration and planner): ABORT_ON_ERROR=1,MEM_LIMIT=5658116096,MT_DOP=0
```

# Look into the Timeline

Planner Timeline: 218.105ms
- Analysis finished: 159.486ms (159.486ms)
- Value transfer graph computed: 159.522ms (35.958us)
- Single node plan created: 162.606ms (3.083ms)
- Runtime filters computed: 162.869ms (262.984us)
- Distributed plan created: 162.908ms (39.628us)
- Lineage info computed: 163.006ms (97.845us)
- Planning finished: 218.105ms (55.098ms)

Query Timeline: 2m34s
- Query submitted: 12.693ms (12.693ms)
- Planning finished: 350.339ms (337.646ms)
- Submit for admission: 422.437ms (72.097ms)
- Completed admission: 433.900ms (11.462ms)
- Ready to start on 8 backends: 648.182ms (214.282ms)
- All 8 execution backends (47 fragment instances) started: 5s683ms (5s035ms)
- First dynamic filter received: 1m50s (1m44s)
- Rows available: 2m32s (41s835ms)
- First row fetched: 2m32s (447.280ms)
- Unregister query: 2m34s (1s232ms)

# Client side

```
  Query Timeline: 1s414ms
     - Start execution: 49.340us (49.340us)
     - Planning finished: 59.532ms (59.483ms)
     - Rows available: 987.346ms (927.813ms)
     - First row fetched: 1s019ms (32.521ms)
     - Unregister query: 1s412ms (392.159ms)
ImpalaServer:
     - ClientFetchWaitTimer: 415.244ms
     - RowMaterializationTimer: 7.795ms
```

Total query time

Idle time: client isn't fetching

- Avoid large data extract.
  - It's usually not a good idea to dump lots of data out using JDBC/ODBC.
- For Impala-shell, use the –B option to fetch lots of data.

# ExecSummary

ExecSummary:

| Operator | #Hosts | Avg Time | Max Time | #Rows | Est. #Rows | Peak Mem | Est. Peak Mem | Detail |
|---|---|---|---|---|---|---|---|---|
| 30:MERGING-EXCHANGE | 1 | 211.877us | 211.877us | 52 | 5 | 0 | -1.00 B | UNPARTITIONED |
| 16:TOP-N | 15 | 148.816us | 204.292us | 52 | 5 | 12.00 KB | 130.00 B | |
| 29:AGGREGATE | 15 | 2.462ms | 2.795ms | 52 | 5 | 2.34 MB | 10.00 MB | FINALIZE |
| 28:EXCHANGE | 15 | 259.949us | 650.720us | 728 | 51 | 0 | 0 | HASH(a.ca_state) |
| 15:AGGREGATE | 15 | 427.704ms | 575.825ms | 728 | 51 | 17.52 MB | 10.00 MB | STREAMING |
| 14:HASH JOIN | 15 | 352.894ms | 584.100ms | 3.13M | 1.13M | 14.00 MB | 314.00 B | LEFT SEMI JOIN, BROADCAST |
| \|--27:EXCHANGE | 15 | 62.910us | 87.360us | 11 | 11 | 0 | 0 | BROADCAST |
| \| 26:AGGREGATE | 1 | 8.992ms | 8.992ms | 11 | 11 | 10.35 MB | 10.00 MB | FINALIZE |
| \| 25:EXCHANGE | 1 | 64.072us | 64.072us | 11 | 11 | 0 | 0 | HASH(j.i_category) |
| \| 08:AGGREGATE | 1 | 31.863ms | 31.863ms | 11 | 11 | 1.68 MB | 10.00 MB | STREAMING |
| \| 07:SCAN HDFS | 1 | 7.636ms | 7.636ms | 300.00K | 300.00K | 5.75 MB | 80.00 MB | tpcds_1000_parquet.item j |
| 13:HASH JOIN | 15 | 2s311ms | 2s419ms | 31.36M | 1.13M | 8.22 MB | 5.00 B | LEFT SEMI JOIN, BROADCAST |
| \|--24:EXCHANGE | 15 | 13.602us | 17.690us | 1 | 1 | 0 | 0 | BROADCAST |
| \| 23:EXCHANGE | 1 | 8.183us | 8.183us | 1 | 1 | 0 | -1.00 B | UNPARTITIONED |
| \| 22:AGGREGATE | 1 | 1.316ms | 1.316ms | 1 | 1 | 2.27 MB | 10.00 MB | FINALIZE |
| \| 21:EXCHANGE | 1 | 7.680us | 7.680us | 1 | 108 | 0 | 0 | HASH((d_month_seq)) |
| \| 06:AGGREGATE | 1 | 0.000ns | 0.000ns | 1 | 108 | 2.20 MB | 10.00 MB | STREAMING |
| \| 05:SCAN HDFS | 1 | 12.308ms | 12.308ms | 31 | 108 | 1.04 MB | 48.00 MB | tpcds_1000_parquet.date_dim |
| 12:HASH JOIN | 15 | 4s724ms | 4s798ms | 2.69B | 2.73B | 6.41 MB | 627.77 KB | INNER JOIN, BROADCAST |
| \|--20:EXCHANGE | 15 | 2.346ms | 3.762ms | 73.05K | 73.05K | 0 | 0 | BROADCAST |
| \| 03:SCAN HDFS | 1 | 5.785ms | 5.785ms | 73.05K | 73.05K | 2.09 MB | 32.00 MB | tpcds_1000_parquet.date_dim |
| 11:HASH JOIN | 15 | 11s504ms | 12s678ms | 2.75B | 2.73B | 396.38 MB | 138.10 MB | INNER JOIN, BROADCAST |
| \|--19:EXCHANGE | 15 | 445.548ms | 481.789ms | 6.00M | 6.00M | 0 | 0 | BROADCAST |
| \| 00:SCAN HDFS | 13 | 19.199ms | 24.624ms | 6.00M | 6.00M | 10.39 MB | 32.00 MB | tpcds_1000_parquet.customer. |
| 10:HASH JOIN | 15 | 20s971ms | 26s141ms | 2.75B | 2.73B | 148.37 MB | 10.67 MB | INNER JOIN, BROADCAST |
| \|--18:EXCHANGE | 15 | 7.430ms | 8.579ms | 300.00K | 300.00K | 0 | 0 | BROADCAST |
| \| 04:SCAN HDFS | 1 | 14.279ms | 14.279ms | 300.00K | 300.00K | 10.37 MB | 120.00 MB | tpcds_1000_parquet.item i |
| 09:HASH JOIN | 15 | 13s292ms | 14s397ms | 2.75B | 2.77B | 650.67 MB | 100.71 MB | INNER JOIN, BROADCAST |
| \|--17:EXCHANGE | 15 | 650.933ms | 685.678ms | 12.00M | 12.00M | 0 | 0 | BROADCAST |
| \| 01:SCAN HDFS | 13 | 27.058ms | 41.049ms | 12.00M | 12.00M | 20.80 MB | 112.00 MB | tpcds_1000_parquet.customer |
| 02:SCAN HDFS | 15 | 865.306ms | 1s147ms | 2.88B | 2.88B | 528.36 MB | 176.00 MB | tpcds_1000_parquet.store_sa. |

# ExecSummary – Find Bottlenecks

- Use ExecSummary from Query Profile to identify bottlenecks

```
    ExecSummary:
Operator              #Hosts   Avg Time   Max Time  #Rows  Est. #Rows   Peak Mem  Est. Peak Mem  Detail
------------------------------------------------------------------------------------------------------------
09:MERGING-EXCHANGE        1    4.394ms    4.394ms  7.30K       8.16K          0        -1.00 B  UNPARTITIONED
04:SORT                    1   38.492ms   38.492ms  7.30K       8.16K   32.02 MB        8.00 MB
08:AGGREGATE               1    8.397ms    8.397ms  7.30K       8.16K  458.25 KB       10.00 MB  MERGE FINALIZE
07:EXCHANGE                1  779.810us  779.810us  7.30K       8.16K          0              0  HASH(a.id)
03:AGGREGATE               1  161.736ms  161.736ms  7.30K       8.16K  466.25 KB       10.00 MB
02:HASH JOIN               1  289.552ms  289.552ms  5.33M       5.33M  318.25 KB       20.91 KB  INNER JOIN, PARTITIONED
|--06:EXCHANGE             1    1.93ms     1.93ms  7.30K       7.30K          0              0  HASH(b.float_col)
|  01:SCAN HDFS            1  227.978ms  227.978ms  7.30K       7.30K  193.00 KB      160.00 MB  functional.alltypes b
05:EXCHANGE                1  816.252us  816.252us  7.30K       7.30K          0              0  HASH(a.float_col)
00:SCAN HDFS               1  228.362ms  228.362ms  7.30K       7.30K  193.00 KB      160.00 MB  functional.alltypes a
```

# ExecSummary – Find Skew

- Use ExecSummary from Query Profile to identify skew
  - Max Time is significantly more than Avg Time => Skew!

ExecSummary:

| Operator | #Hosts | Avg Time | Max Time | #Rows | Est. #Rows | Peak Mem | Est. Peak Mem |
|---|---|---|---|---|---|---|---|
| 08:EXCHANGE | 1 | 113.775us | 113.775us | 31 | 227 | 0 | -1.00 B |
| 07:AGGREGATE | 81 | 359.985ms | 436.922ms | 31 | 227 | 3.44 MB | 10.00 MB |
| 06:EXCHANGE | 81 | 57.25us | 515.364us | 397 | 227 | 0 | 0 |
| 03:AGGREGATE | 81 | 561.26ms | 1s344ms | 397 | 227 | 3.66 MB | 10.00 MB |
| 02:HASH JOIN | 81 | 3s730ms | 18s695ms | 184.02M | 2.08M | 3.04 MB | 13.64 KB |
| \|--05:EXCHANGE | 81 | 27.471us | 42.597us | 26.74K | 25.40K | 0 | 0 |
| \| 01:SCAN HDFS | 1 | 359.799ms | 359.799ms | 26.74K | 25.40K | 4.47 MB | 16.00 MB |
| 04:EXCHANGE | 81 | 130.853ms | 1s608ms | 184.03M | 2.08M | 0 | 0 |
| 00:SCAN HDFS | 81 | 154.864ms | 553.824ms | 184.03M | 2.08M | 11.46 MB | 88.00 MB |

# Exercises

- Predicate pushdown
- Remote read
- Codegen
- Planning time
- DDLs

# Advanced Query Tuning and Troubleshooting

Strata
DATA CONFERENCE

# Advanced Query Tuning

- Common issues
  - Query succeeds but very slow
  - Query fails with OOM error
-
- How to address them
  - Examine the logic of the query and validate the Explain Plan
  - Use Query Profile to identify bottlenecks.

# Performance analysis – Example 1

| Operator | #Hosts | Avg Time | Max Time | #Rows | Est. #Rows |
|---|---|---|---|---|---|
| 02:AGGREGATE | 1 | 15s129ms | 15s12 | | 0 |
| 01:AGGREGATE | 1 | 46s515ms | 46s515 | | 0 |
| 00:SCAN HDFS | 1 | 2s962ms | 2s962 | | 0 |

Corrupted stats

- **Fix: Compute stats <table>**

| Operator | #Hosts | Avg Time | Max Time | #Rows | Est. #Rows |
|---|---|---|---|---|---|
| 06:AGGREGATE | 1 | 196.914ms | 196.914ms | 1 | 1 |
| 05:EXCHANGE | 1 | 145.541us | 145.541us | 17 | 1 |
| 02:AGGREGATE | 17 | 285.524ms | 308.758ms | 17 | 1 |
| 04:AGGREGATE | 17 | 1s106ms | 1s185ms | 51.63M | 51.63M |
| 03:EXCHANGE | 17 | 79.289ms | 87.953ms | 51.63M | 51.63M |
| 01:AGGREGATE | 17 | 1s487ms | 2s318ms | 51.63M | 51.63M |
| 00:SCAN HDFS | 17 | 190.709ms | 264.350ms | 51.63M | 51.63M |

# Performance analysis – Example 2

```
| Estimated Per-Host Requirements: Memory=84.76GB VCores=3                                  |
| WARNING: The following tables are missing relevant table and/o          tics.             |
| tpcds500gb_parquet.date_dim, tpcds500gb_parquet.store_sale          .item                 |
|                                                                                           |
| 04:HASH JOIN [INNER JOIN, BROADCAST]                                                      |
| |  hash predicates: store_sales.ss_item_sk = item.i_item_sk                               |
| |                                                                                         |
| |--08:EXCHANGE [BROADCAST]                                                                |
| |  |                                                                                      |
| |  02:SCAN HDFS [tpcds500gb_parquet.item]                                                 |
| |     partitions=1/1 size=3.14MB                                                          |
| |     predicates: item.i_manager_id = 1                                                   |
| |                                                                                         |
| 03:HASH JOIN [INNER JOIN, BROADCAST]                                                      |
| |  hash predicates: dt.d_date_sk = store_sales.ss_sold_date_sk                            |
| |                                                                                         |
| |--07:EXCHANGE [BROADCAST]                                                                |
| |  |                                                                                      |
| |  01:SCAN HDFS [tpcds500gb_parquet.store_sales]                                          |
| |     partitions=1823/1823 size=180.21GB                                                  |
| |                                                                                         |
| 00:SCAN HDFS [tpcds500gb_parquet.date_dim dt]                                             |
|    partitions=1/1 size=2.41MB                                                             |
|    predicates: dt.d_moy = 12, dt.d_year = 1998                                            |
+-------------------------------------------------------------------------------------------+
```

**Missing stats**

# Performance analysis – Example 2 (good plan)

```
…
|
|  |--07:EXCHANGE [BROADCAST]
|  |  |  hosts=3 per-host-mem=0B
|  |  |  tuple-ids=0 row-size=16B cardinality=29
|  |  |
|  |  00:SCAN HDFS [tpcds500gb_parquet.date_dim dt, RANDOM]
|  |     partitions=1/1 size=2.41MB
|  |     predicates: dt.d_moy = 12, dt.d_year = 1998
|  |     table stats: 73049 rows total
|  |     column stats: all
|  |     hosts=3 per-host-mem=48.00MB
|  |     tuple-ids=0 row-size=16B cardinality=29
|  |
|  01:SCAN HDFS [tpcds500gb_parquet.store_sales, RANDOM]
|     partitions=1823/1823 size=180.21GB
|     table stats: 4125561494 rows total
|     column stats: all
|     hosts=10 per-host-mem=176.00MB
|     tuple-ids=1 row-size=20B cardinality=4125561494
+-----------------------------------------------------------------------------------------+
```

```
set explain_level=3;
set num_nodes=0;
```

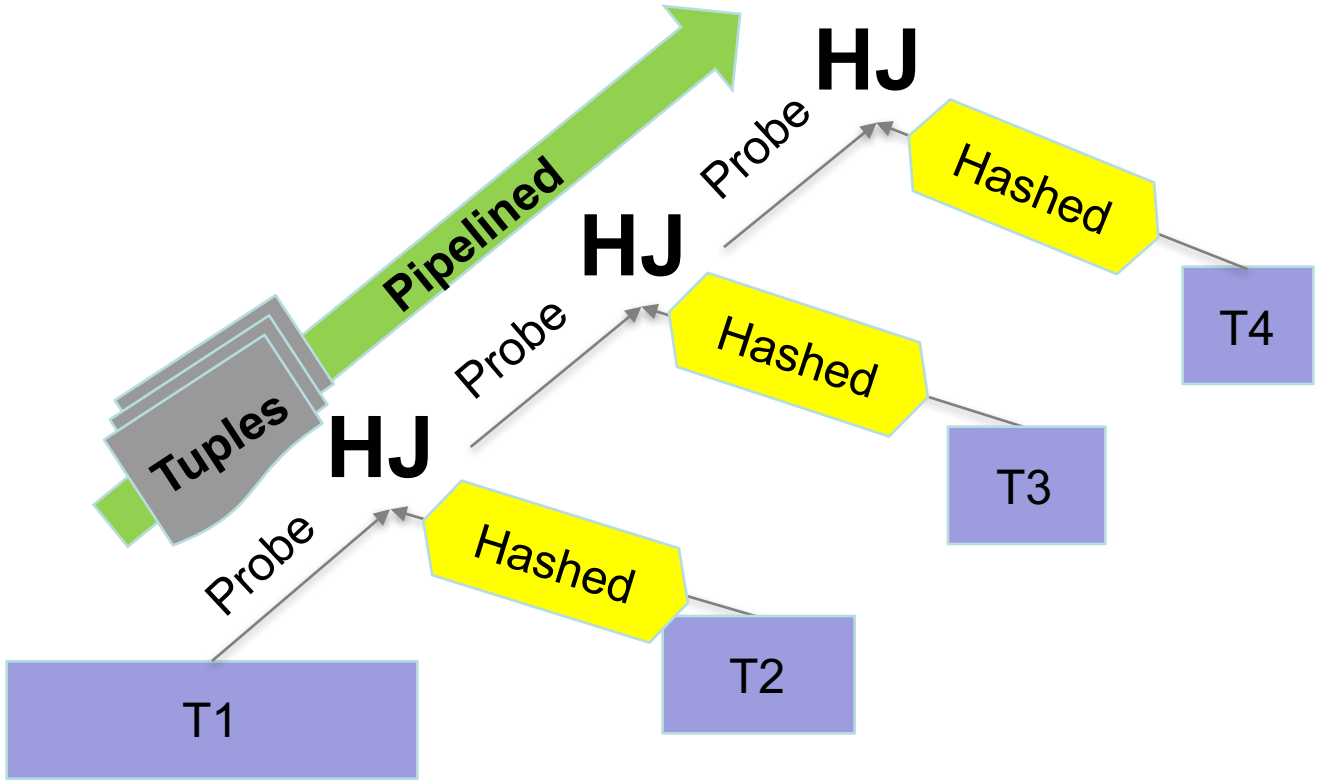
# Query Tuning Basics - Join

- Validate join order and join strategy
  - Optimal Join Order
    - RHS should be smaller than LHS
    - Minimize intermediate results
  - Strategy – Broadcast vs. Partitioned
    - Network costs (partition and send lhs+rhs or broadcast rhs)
    - Memory costs
    - RHS must fit in memory!

# Join-Order Optimization

**SELECT** … **FROM** T1, T2, T3, T4

**WHERE** T1.id = T2.id **AND** T2.id = T3.id **AND** T3.id = T4.id;



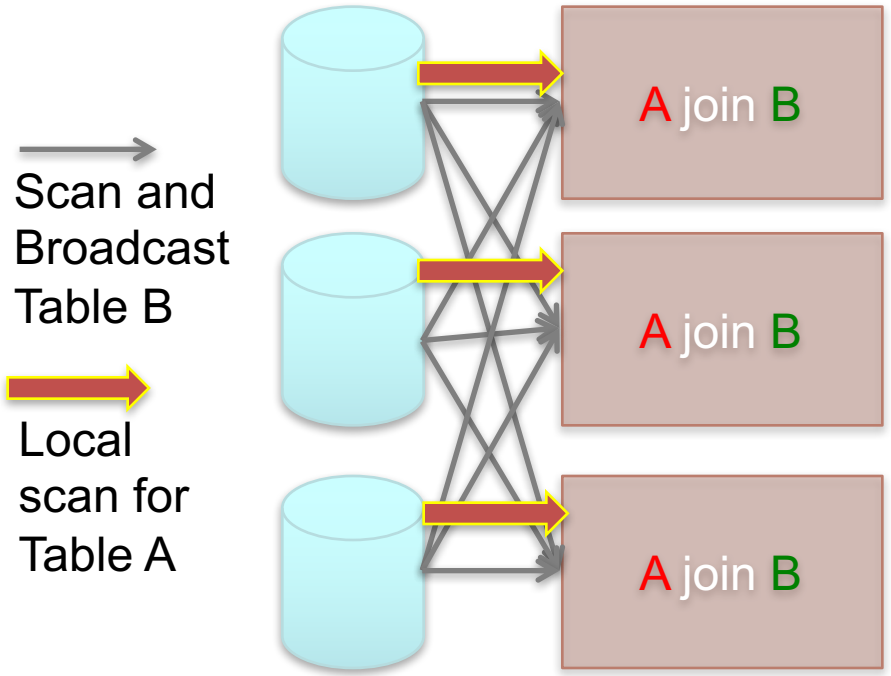
# Use of Statistics during Plan Generation

- **Table statistics:** Number of rows per partition/table.
- **Column statistics:** Number of distinct values per column.

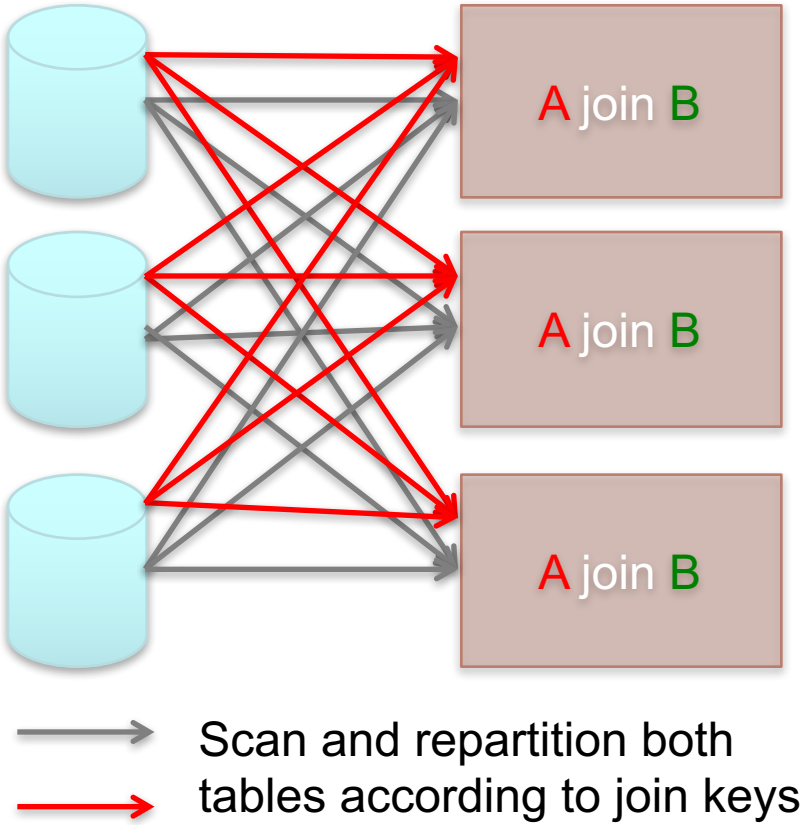
- **Use of table and column statistics**
  - Estimate selectivity of predicates (esp. scan predicates like “month=10”).
  - Estimate selectivity of joins → join cardinality (#rows)
    - Heuristic FK/PK detection.
  - Pick distributed join strategy: broadcast vs. partitioned
    - Join inputs have very different size → broadcast small side.
    - Join inputs have roughly equal size → partitioned.

# Join Execution Strategy

- Broadcast Join




- Repartition Join



# Join Strategy – what's the cost

- Impala choses the right strategy based on stats (**collect stats!**)
- Use the join strategy that minimizes data transfer
- Use explain plan to see the data size
- Join Hint: [shuffle] or [broadcast]

| Join strategy cost | Network Traffic | Memory Usage (HashTable) |
|---|---|---|
| Broadcast Join | RHS table size[1] x number of node | RHS table size[1] x number of node |
| Partitioned Join | LHS + RHS table size[1] | RHS table size[1] |

[1] table size refers to the data flowed from the child node (i.e. only required column data after filtering counts).

# Join - Exercise

- TPCH 100GB data on 8-node cluster
  - Table1: lineitem.  cardinality=600M, row size ~ 260B
  - Table2: orders.  cardinality=150M, row size ~ 200B
  - orderkey is bigint.

- For the following query, what's the optimal join order and join strategy.
  - **SELECT** count(*) **FROM** lineitem **JOIN** orders **ON** l_orderkey = o_orderkey;
  - **SELECT** lineitem.* **FROM** lineitem **JOIN** orders **ON** l_orderkey = o_orderkey;
  - **SELECT** orders.* **FROM** lineitem **JOIN** orders **ON** l_orderkey = o_orderkey;

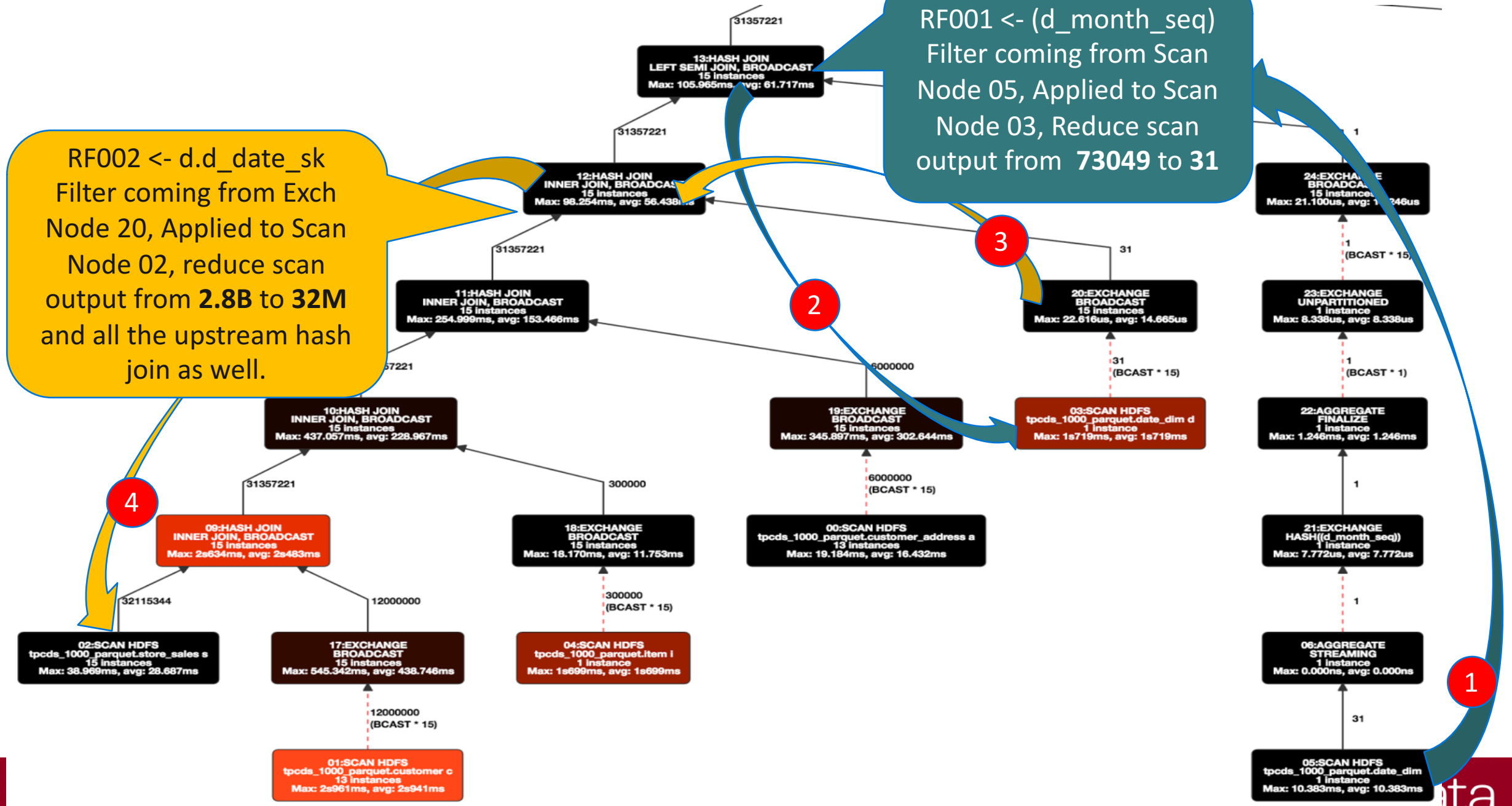
- How about on a 100-node cluster?

# Runtime Filter - Check how selective the join is

ExecSummary:

| Operator | #Hosts | Avg Time | Max Time | #Rows | Est. #Rows | Peak Mem | Est. Peak Mem | Detail |
|---|---|---|---|---|---|---|---|---|
| 30:MERGING-EXCHANGE | 1 | 211.877us | 211.877us | 52 | 5 | | | |
| 16:TOP-N | 15 | 148.816us | 204.292us | 52 | 5 | 12 | | |
| 29:AGGREGATE | 15 | 2.462ms | 2.795ms | 52 | 5 | 2 | | |
| 28:EXCHANGE | 15 | 259.949us | 650.720us | 728 | 51 | | | |
| 15:AGGREGATE | 15 | 427.704ms | 575.825ms | 728 | 51 | 17 | | |
| 14:HASH JOIN | 15 | 352.894ms | 584.100ms | 3.13M | 1.13M | 14 | | ST |
| \|--27:EXCHANGE | 15 | 62.910us | 87.360us | 11 | 11 | | | |
| \| 26:AGGREGATE | 1 | 8.992ms | 8.992ms | 11 | 11 | | | |
| \| 25:EXCHANGE | 1 | 64.072us | 64.072us | 11 | | | | |
| \| 08:AGGREGATE | 1 | 31.863ms | 31.863ms | 11 | 11 | 1 | | |
| \| 07:SCAN HDFS | 1 | 7.636ms | 7.636ms | 300.00K | 300.00K | 5.7 | | em j |
| 13:HASH JOIN | 15 | 2s311ms | 2s419ms | 31.36M | 1.13M | 8.22 MB | 5.00 B | LEFT SEMI JOIN, BROADCAST |
| \|--24:EXCHANGE | 15 | 13.602us | 17.690us | 1 | 1 | 0 | 0 | BROADCAST |
| \| 23:EXCHANGE | 1 | 8.183us | 8.183us | 1 | 1 | 0 | -1.00 B | UNPARTITIONED |
| \| 22:AGGREGATE | 1 | 1.316ms | 1.316ms | 1 | 1 | 2.27 MB | 10.00 MB | FINALIZE |
| \| 21:EXCHANGE | 1 | 7.680us | 7.680us | 1 | 108 | 0 | 0 | HASH((d_month_seq)) |
| \| 06:AGGREGATE | 1 | 0.000ns | 0.000ns | 1 | 108 | 2.20 MB | 10.00 MB | STREAMING |
| \| 05:SCAN HDFS | 1 | 12.308ms | 12.308ms | 31 | 108 | 1.04 MB | 48.00 MB | tpcds_1000_parquet.date_dim |
| 12:HASH JOIN | 15 | 4s724ms | 4s798ms | 2.69B | 2.73B | 6.41 MB | 627.77 KB | INNER JOIN, BROADCAST |
| \|--20:EXCHANGE | 15 | 2.346ms | 3.762ms | 73.05K | 73.05K | 0 | 0 | BROADCAST |
| \| 03:SCAN HDFS | 1 | 5.785ms | 5.785ms | 73.05K | 73.05K | 2.09 MB | 32.00 MB | tpcds_1000_parquet.date_dim |
| 11:HASH JOIN | 15 | 11s504ms | 12s678ms | 2.75B | 2.73B | 396.38 MB | 138.10 MB | INNER JOIN, BROADCAST |
| \|--19:EXCHANGE | 15 | 445.548ms | 481.789ms | 6.00M | 6.00M | 0 | 0 | BROADCAST |
| \| 00:SCAN HDFS | 13 | 19.199ms | 24.624ms | 6.00M | 6.00M | 10.39 MB | 32.00 MB | tpcds_1000_parquet.customer. |
| 10:HASH JOIN | 15 | 20s971ms | 26s141ms | 2.75B | 2.73B | 148.37 MB | 10.67 MB | INNER JOIN, BROADCAST |
| \|--18:EXCHANGE | 15 | 7.430ms | 8.579ms | 300.00K | 300.00K | 0 | 0 | BROADCAST |
| \| 04:SCAN HDFS | 1 | 14.279ms | 14.279ms | 300.00K | 300.00K | 10.37 MB | 120.00 MB | tpcds_1000_parquet.item i |
| 09:HASH JOIN | 15 | 13s292ms | 14s397ms | 2.75B | 2.77B | 650.67 MB | 100.71 MB | INNER JOIN, BROADCAST |
| \|--17:EXCHANGE | 15 | 650.933ms | 685.678ms | 12.00M | 12.00M | 0 | 0 | BROADCAST |
| \| 01:SCAN HDFS | 13 | 27.058ms | 41.049ms | 12.00M | 12.00M | 20.80 MB | 112.00 MB | tpcds_1000_parquet.customer |
| 02:SCAN HDFS | 15 | 865.306ms | 1s147ms | 2.88B | 2.88B | 528.36 MB | 176.00 MB | tpcds_1000_parquet.store_sa. |

Large hash join output (~2.8B), the output of upstream one is much smaller (32M, reduced to only ~1%). This indicates runtime filter can be helpful.


RF001 <- (d_month_seq)
Filter coming from Scan Node 05, Applied to Scan Node 03, Reduce scan output from 73049 to 31
RF002 <- d.d_date_sk
Filter coming from Exch Node 20, Applied to Scan Node 02, reduce scan output from 2.8B to 32M and all the upstream hash join as well.
1
2
3
4
13:HASH JOIN
LEFT SEMI JOIN, BROADCAST
15 instances
Max: 105.965ms, avg: 61.717ms
12:HASH JOIN
INNER JOIN, BROADCAST
15 instances
11:HASH JOIN
INNER JOIN, BROADCAST
15 instances
Max: 254.999ms, avg: 153.466ms
10:HASH JOIN
INNER JOIN, BROADCAST
15 instances
Max: 437.057ms, avg: 228.967ms
09:HASH JOIN
INNER JOIN, BROADCAST
15 instances
Max: 2s634ms, avg: 2s483ms
02:SCAN HDFS
tpcds_1000_parquet.store_sales s
15 instances
Max: 38.969ms, avg: 28.687ms
17:EXCHANGE
BROADCAST
15 instances
Max: 545.342ms, avg: 438.746ms
01:SCAN HDFS
tpcds_1000_parquet.customer c
13 instances
Max: 2s961ms, avg: 2s941ms
18:EXCHANGE
BROADCAST
15 instances
Max: 18.170ms, avg: 11.753ms
04:SCAN HDFS
tpcds_1000_parquet.item i
1 instance
Max: 1s699ms, avg: 1s699ms
19:EXCHANGE
BROADCAST
15 instances
Max: 345.897ms, avg: 302.644ms
00:SCAN HDFS
tpcds_1000_parquet.customer_address a
13 instances
Max: 19.184ms, avg: 16.432ms
20:EXCHANGE
BROADCAST
15 instances
Max: 22.616us, avg: 14.665us
03:SCAN HDFS
tpcds_1000_parquet.date_dim d
1 instance
Max: 1s719ms, avg: 1s719ms
24:EXCHANGE
BROADCAST
15 instances
Max: 21.100us, avg: 1
23:EXCHANGE
UNPARTITIONED
1 instance
Max: 8.338us, avg: 8.338us
22:AGGREGATE
FINALIZE
1 instance
Max: 1.246ms, avg: 1.246ms
21:EXCHANGE
HASH((d_month_seq))
1 instance
Max: 7.772us, avg: 7.772us
06:AGGREGATE
STREAMING
1 instance
Max: 0.000ns, avg: 0.000ns
05:SCAN HDFS
tpcds_1000_parquet.date_dim
1 instance
Max: 10.383ms, avg: 10.383ms
31357221
31357221
31357221
31
6000000
31
(BCAST * 15)
1
(BCAST * 15)
1
(BCAST * 1)
1
1
1
31
31357221
300000
6000000
(BCAST * 15)
32115344
12000000
300000
(BCAST * 15)
12000000
(BCAST * 15)

# Why Runtime Filter doesn't work sometimes?

ExecSummary:

| Operator | #Hosts | Avg Time | Max Time | #Rows | Est. #Rows | Peak Mem | Est. Peak Mem | Detail |
|---|---|---|---|---|---|---|---|---|
| Final filter table: | | | | | | | | |

| ID | Src. Node | Tgt. Node(s) | Targets | Target type | Partition filter | Pending (Expected) | First arrived | Completed | Enabled |
|---|---|---|---|---|---|---|---|---|---|
| 5 | 9 | 2 | 15 | LOCAL | false | 0 (15) | N/A | N/A | true |
| 4 | 10 | 2 | 15 | LOCAL | false | 0 (15) | N/A | N/A | true |
| 3 | 11 | 1 | 13 | REMOTE | false | 0 (3) | 3s276ms | 3s276ms | false |
| 2 | 12 | 2 | 15 | LOCAL | true | 0 (15) | N/A | N/A | true |
| 1 | 13 | 3 | 1 | REMOTE | false | 0 (3) | 1s935ms | 1s935ms | false |
| 0 | 14 | 4 | 1 | REMOTE | false | 0 (3) | 1s943ms | 1s943ms | false |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| \|  07:SCAN HDFS | 1 | 6.700ms | 6.700ms | 300.00K | 300.00K | 3.70 MB | 80.00 MB | tpcds_1000_parquet.item j |
| 13:HASH JOIN | 15 | 2s327ms | 2s439ms | 31.36M | 1.13M | 3.40 MB | 5.00 B | LEFT SEMI JOIN, BROADCAST |
| \|--24:EXCHANGE | 15 | 13.272us | 16.888us | 1 | 1 | 0 | 0 | BROADCAST |
| \|  23:EXCHANGE | 1 | 8.397us | 8.397us | 1 | 1 | 0 | -1.00 B | UNPARTITIONED |
| \|  22:AGGREGATE | 1 | 8.342ms | 8.342ms | 1 | 1 | 2.27 MB | 10.00 MB | FINALIZE |
| \|  21:EXCHANGE | 1 | 10.064us | 10.064us | 1 | 108 | 0 | 0 | HASH((d_month_seq)) |
| \|  06:AGGREGATE | 1 | 0.000ns | 0.000ns | 1 | 108 | 2.20 MB | 10.00 MB | STREAMING |
| \|  05:SCAN HDFS | 1 | 22.849ms | 22.849ms | 31 | | | | tpcds_1000_parquet.date_dim |
| 12:HASH JOIN | 15 | 4s763ms | 5s414ms | 2.69B | | | | , BROADCAST |
| \|--20:EXCHANGE | 15 | 2.481ms | 3.947ms | 73.05K | | | | |
| \|  03:SCAN HDFS | 1 | 972.009ms | 972.009ms | | | | | parquet.date_dim d |
| 11:HASH JOIN | 15 | 10s593ms | 11s725ms | 2.69 | | | | , BROADCAST |
| \|--19:EXCHANGE | 15 | 277.472ms | 314.535ms | 6.00M | | | | |
| \|  00:SCAN HDFS | 13 | 16.104ms | 20.280ms | 6.00M | | | | parquet.customer... |
| 10:HASH JOIN | 15 | 22s428ms | 25s443ms | 2.69B | | | | , BROADCAST |
| \|--18:EXCHANGE | 15 | 6.842ms | 7.392ms | 300.00K | | | | |
| \|  04:SCAN HDFS | 1 | 978.822ms | 978.822ms | 300.00K | | | | parquet.item i |
| 09:HASH JOIN | 15 | 12s937ms | 13s782ms | 2.69B | | | | , BROADCAST |
| \|--17:EXCHANGE | 15 | 416.050ms | 481.783ms | 12.00M | | | | |
| \|  01:SCAN HDFS | 13 | 986.075ms | 993.425ms | 12.00M | 12.00M | 20.80 MB | 112.00 MB | tpcds_1000_parquet.customer c |
| 02:SCAN HDFS | 15 | 857.578ms | 950.789ms | 2.75B | 2.88B | 510.79 MB | 176.00 MB | tpcds_1000_parquet.store_sa... |

Filter doesn't take effect because scan was short and finished before filter arrived.

HDFS_SCAN_NODE (id=3):

Filter 1 (1.00 MB):

- **Rows processed: 16.38K (16383)**
- **Rows rejected: 0 (0)**
- Rows total: 16.38K (16384)

# How to tune it?

- Increase RUNTIME_FILTER_WAIT_TIME_MS to 5000ms to let Scan Node 03 wait longer time for the filter.

HDFS_SCAN_NODE (id=3)

Filter 1 (1.00 MB)

- InactiveTotalTime: 0
- Rows processed: 73049
- **Rows rejected: 73018**
- Rows total: 73049
- TotalTime: 0

- If the cluster is relatively busy, consider increasing the wait time too so that complicated queries do not miss opportunities for optimization.

# Runtime filter profile examples

- Profile walkthrough
  - Effective vs Non-Effective
  - Local vs Global
  - Filter Wait Time
  - Filter Memory Usage

# Memory

- Planner estimation

```
Estimated Per-Host Requirements: Memory=68.01MB VCores=2

06:SORT
|  order by: i_class ASC NULLS FIRST
|  hosts=1 per-host-mem=16.00MB
|  tuple-ids=7 row-size=214B cardinality=48
```

- Actual usage from profile

```
Execution Profile b8414c34981f3ec9:a52048d52a00fb1:(Total: 9s754ms, non-child: 0ns, % non-child: 0.00%)
  Per Node Peak Memory Usage: alan-OptiPlex-790:22000(11.77 MB)
    - FinalizationTimer: 0ns
```

- Metrics per node

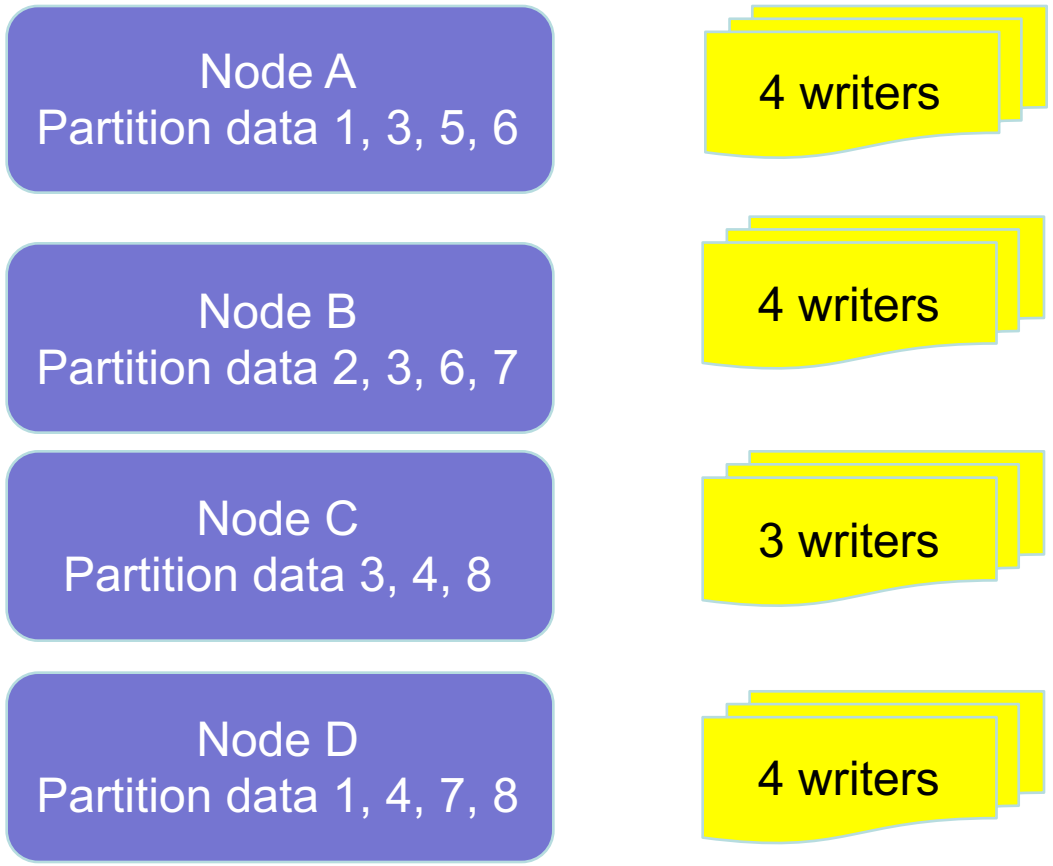
```
- PeakMemoryUsage: 70048                    - MemoryLimit: 85899345920
- PerHostPeakMemUsage: 1779374788           - PeakMemoryUsage: 1219248384
```

# Memory - OOM

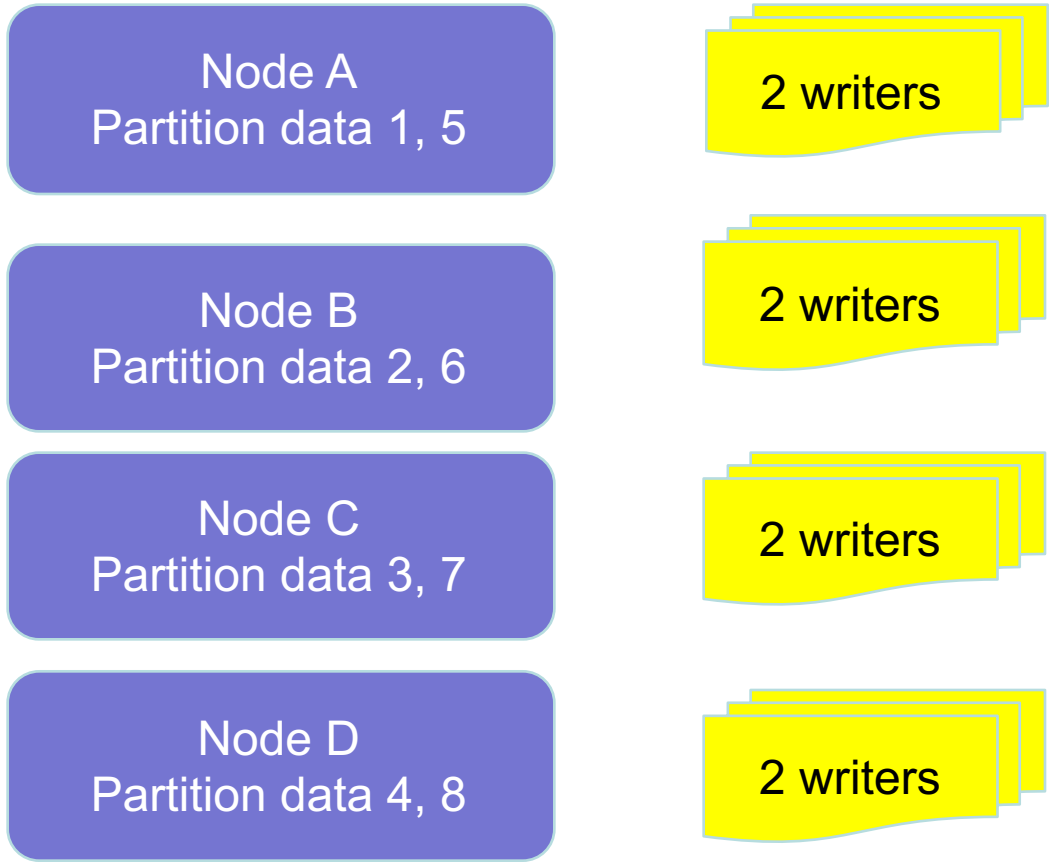
```
Query Status: Memory limit exceeded: FunctionContext::Allocate's allocations exceeded memory limits.
Exprs could not allocate 384.00 B without exceeding limit.
Error occurred on backend vd1337.halxg.cloudera.com:22000 by fragment c74ce10ea42773c1:8f12f6e900000080
Memory left in process limit: -853607.00 B
Process: memory limit exceeded. Limit=201.73 GB Total=201.73 GB Peak=201.73 GB
  RequestPool=root.default: Total=189.29 GB Peak=189.29 GB
    Query(c74ce10ea42773c1:8f12f6e900000000): Total=189.22 GB Peak=189.22 GB
      Fragment c74ce10ea42773c1:8f12f6e900000101: Total=10.30 MB Peak=11.02 MB
        AGGREGATION_NODE (id=2): Total=8.00 KB Peak=8.00 KB
          Exprs: Total=4.00 KB Peak=4.00 KB
        AGGREGATION_NODE (id=4): Total=10.27 MB Peak=10.27 MB
          Exprs: Total=4.00 KB Peak=4.00 KB
        EXCHANGE_NODE (id=3): Total=0 Peak=0
        DataStreamRecvr: Total=0 Peak=0
        DataStreamSender (dst_id=5): Total=7.52 KB Peak=7.52 KB
        CodeGen: Total=6.79 KB Peak=750.50 KB
      Block Manager: Limit=161.39 GB Total=13.63 GB Peak=13.63 GB
      Fragment c74ce10ea42773c1:8f12f6e900000080: Total=189.21 GB Peak=189.21 GB
        AGGREGATION_NODE (id=1): Total=188.83 GB Peak=188.83 GB
          Exprs: Total=175.20 GB Peak=175.20 GB
        HDFS_SCAN_NODE (id=0): Total=385.53 MB Peak=601.23 MB
        DataStreamSender (dst_id=3): Total=660.12 KB Peak=660.12 KB
        CodeGen: Total=4.48 KB Peak=610.00 KB
    Query(8743be49f34a3cb9:8bb6ace100000000): Total=34.30 MB Peak=49.49 MB
      Fragment 8743be49f34a3cb9:8bb6ace1000000f7: Total=34.30 MB Peak=35.22 MB
```

# Memory – Insert strategy

- Non-shuffle




- Shuffle



# Memory – Group By

- **SELECT** product, count(1), sold_date **FROM** sales_history **GROUP BY** product, sold_date;
- We have one million different products, table contains data in the past 5 years.

**Total groups = 1M * 5 * 365 = ~1.8B**

```
Query Status: Memory limit exceeded

Operator       #Hosts   #Rows  Est. #Rows   Peak Mem  Est. Peak Mem
------------------------------------------------------------------
04:EXCHANGE         1       0          10          0       -1.00 B
03:AGGREGATE       68       0       3.24B  260.27 MB      84.34 MB
02:EXCHANGE        68   6.87M       18.50B          0            0
01:AGGREGATE       68   6.38M       18.50B   20.89 GB    984.90 GB
00:SCAN HDFS       68  12.95B       18.50B  414.27 MB      48.00 MB
```

```
AGGREGATION_NODE (id=1)
  ExecOption: Codegen Enabled, Spilled
  - AsyncTotalTime: 0
  - BuildTime: 165828722183
  - GetNewBlockTime: 191148346
  - GetResultsTime: 1035063643
  - HashBuckets: 16777216
  - InactiveTotalTime: 0
  - LargestPartitionPercent: 6
  - MaxPartitionLevel: 0
  - NumRepartitions: 0
  - PartitionsCreated: 16
  - PeakMemoryUsage: 22434481544
  - PinTime: 0
  - RowsRepartitioned: 0
  - RowsReturned: 6375424
  - RowsReturnedRate: 37756
  - SpilledPartitions: 1
  - TotalTime: 168854653348
```

# Memory Usage – Estimation

- EXPLAIN's memory estimation issues
  - Can be way off – much higher or much lower.
  - Group-by/distinct estimate can be particularly off – when there's a large number of group by columns (independence assumption)
    - Memory estimate = NDV of group by column 1 * NDV of group by column 2 * … NDV of group by column n
  - Ignore EXPLAIN's estimation if it's too high!
- Do your own estimate for group by
  - **GROUP BY memory usage = (total number of groups * size of each row) + (total number of groups * size of each row) / number node**

# Memory Usage – Hitting Mem-limit

- Gigantic group by
  - The total number of distinct groups is huge, such as group by userid, phone number.
  - For a simple query, you can try this advanced workaround – per-partition agg
    - Requires the partition key to be part of the group by

```
SELECT part_key, col1, col2, …agg(..) FROM tbl WHERE part_key in (1,2,3)
UNION ALL
SELECT part_key, col1, col2, …agg(..) FROM tbl WHERE part_key in (4,5,6)
```

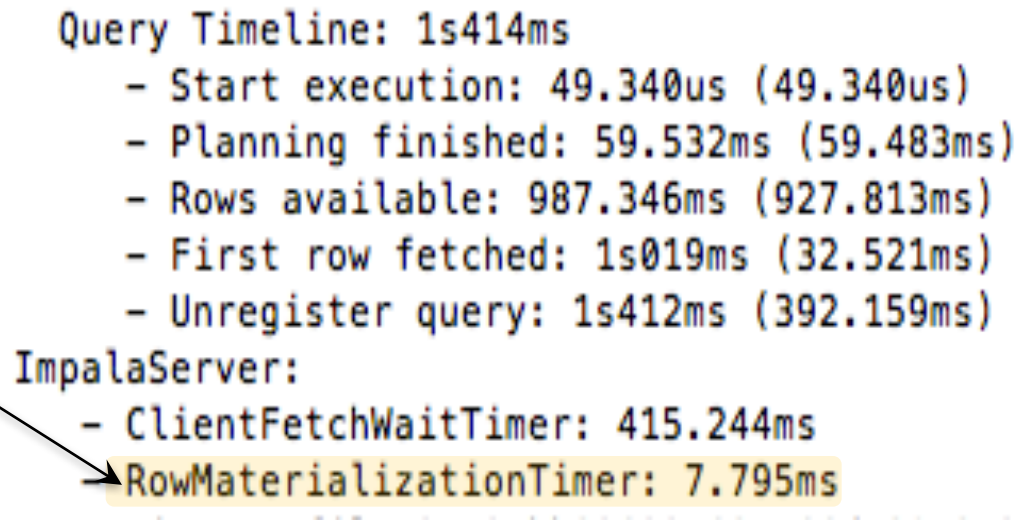
# Memory Usage – Hitting Mem-limit

- Big-table joining big-table
  - Big-table (after decompression, filtering, and projection) is a table that is bigger than total cluster memory size.
  - For a simple query, you can try this advanced workaround – per-partition join
    - Requires the partition key to be part of the join key

```
SELECT … FROM BigTbl_A a JOIN BigTbl_B b WHERE a.part_key =
b.part_key AND a.part_key IN (1,2,3)
UNION ALL
SELECT … FROM BigTbl_A a JOIN BigTbl_B b WHERE a.part_key =
b.part_key AND a.part_key IN (4,5,6)
```

# Query Execution – Typical Speed

- In a typical query, we observed following processing rate:
  - Scan node 8~10M rows per core
  - Join node ~10M rows per sec per core
  - Agg node ~5M rows per sec per core
  - Sort node ~17MB per sec per core
  - Row materialization in coordinator should be tiny
  - Parquet writer 1~5MB per sec per core




- If your processing rate is much lower than that, it's worth a deeper look

# Performance analysis – Example 3

ExecSummary:

| Operator | #Hosts | Avg Time | Max Time | #Rows | Est. #Rows | Peak Mem | Est. Peak Mem |
|---|---|---|---|---|---|---|---|
| 04:EXCHANGE | 1 | 64.320us | 64.320us | 18 | 18 | 0 | -1.00 B |
| 03:AGGREGATE | 3 | 222.379ms | 248.857ms | 18 | 18 | 6.27 MB | 10.00 MB |
| 02:EXCHANGE | 3 | 59.650us | 60.607us | 54 | 18 | 0 | 0 |
| 01:AGGREGATE | 3 | 6s453ms | 6s815ms | 54 | 18 | 6.67 MB | 10.00 MB |
| 00:SCAN HDFS | 3 | 203.150ms | 216.808ms | 144.62M | 144.62M | 105.24 MB | 176.00 MB |

~8M rows per second per core, this is normal aggregation speed.

# Performance analysis – Example 4

```
Operator            #Hosts   Avg Time    Max Time
-----------------------------------------------
04:EXCHANGE              1  999.998us  999.998us
02:HASH JOIN             7    2s636ms    4s879ms
|--03:EXCHANGE           7  142.860us    1.000ms
|  01:SCAN HDFS          7  461.717ms  486.001ms
00:SCAN HDFS             7    2s862ms    3s515ms

Averaged Fragment F00
  split sizes:  min: 982.25 MB, max: 2
  completion times: min:4s474ms  ma
  execution rates: min:219.50 MB/s
  num instances: 7

HDFS_SCAN_NODE (id=0)
  Hdfs split stats (<volume id>:<# splits>/<split lengths>): 0:15/2.96 GB
  - BytesRead: 1710183933
HDFS_SCAN_NODE (id=0)
  Hdfs split stats (<volume id>:<# splits>/<split lengths>): 0:5/982.25 MB
  - BytesRead: 556141382
```

Data Distribution Skew causes execution skew

# Performance analysis – Example 5

```
Operator          #Hosts   Avg Time   Max Time    #Rows   Est
-------------------------------------------------------------
00:SCAN HDFS           7    6s577ms   44s614ms

Averaged Fragment F00:
  split sizes:  min: 1.62 GB, max: 2.06 GB, av
  completion times: min:9s835ms  max:47s249ms
  execution rates: min:35.02 MB/sec  max:190.
  num instances: 7

      - TotalStorageWaitTime: 5m19s
        - BytesRead: 120.19 MB (126032
        - RowBatchQueueGetWaitTime: 44s585
      - TotalStorageWaitTime: 594.017ms
        - BytesRead: 152.74 MB (160156619)
        - RowBatchQueueGetWaitTime: 223.006ms
      - TotalStorageWaitTime: 678.025ms
        - BytesRead: 137.05 MB (143707548)
        - RowBatchQueueGetWaitTime: 151.005ms
```

Possible root cause:

1. The host has a slow/bad disk
2. The host has trouble to communicate to NN
3. Hotspotting, the host does more IO (not just this single query, but overall) than others
4. …

# Performance analysis – Example 6

```
Operator          #Hosts   Avg Time    Max Time  #Rows    Est. #Rows
-------------------------------------------------------------------
05:AGGREGATE           2    10s520s    10s534s    3.14M         2.95M
...
|  03:SCAN HDFS        22   608.374ms  610.741ms      10            10
00:SCAN HDFS           2    3.181s     3.649s     100.95M       100.95M

00:SCAN HDFS [store_sales, RANDOM]
   partitions=2/138 files=2 size=229.00MB
   table stats: 12705876000 rows total
```

Only two parquet files to scan, not enough parallelism. Try using smaller parquet block size for this table to increase parallelism.

# Performance analysis – Example 7

**SELECT** colE, colU, colR, ... **FROM** temp **WHERE** year = 2017 **AND** month = 03
**GROUP BY** colE, colU, colR;

```
03:AGGREGATE [FINALIZE]
|  output:
|  group by: colE, colU, colR
|  hosts=100 per-host-mem=549.20GB
|  tuple-ids=1 row-size=244B cardinality=2197124638
|
02:EXCHANGE [HASH(colE, colU, colR)]
|  hosts=100 per-host-mem=0B
|  tuple-ids=1 row-size=244B cardinality=2197124638
|
01:AGGREGATE [STREAMING]
|  output:
|  group by: colE, colU, colR
|  hosts=100 per-host-mem=549.20GB
|  tuple-ids=1 row-size=244B cardinali
|
00:SCAN HDFS [temp, RANDOM]
   partitions=1/4 files=1040 size=247.54GB
   table stats: 8569254547 rows total
   column stats: all
   hosts=100 per-host-mem=616.00MB
   tuple-ids=0 row-size=166B cardinality=2197124638
```

Slow network

```
Averaged Fragment F01
  - TotalNetworkReceiveTime: 1655023089318
  - TotalNetworkSendTime: 0

    EXCHANGE_NODE (id=2)
      - BytesReceived: 2086387486
      - FirstBatchArrivalWaitTime: 15261428195
      - InactiveTotalTime: 1660605931017
      - SendersBlockedTimer: 0
      - SendersBlockedTotalTimer(*): 0
      - TotalTime: 1662118169847

Averaged Fragment F00
  - TotalNetworkReceiveTime: 0
  - TotalNetworkSendTime: 1295116305796

  DataStreamSender (dst_id=2)
    - BytesSent: 2088864189
```

# Exchange and DataStreamSender

- Impala uses Thrift to transmit compressed data between plan fragments
- Exchange operator is the receiver of the data
- DataStreamSender transmits the output rows of a plan fragment

EXCHANGE_NODE (id=3)
- ConvertRowBatchTime: 0ns (0)
- InactiveTotalTime: 0ns (0)
- RowsReturned: 24 (24)
- RowsReturnedRate: 7 per second (7)
- TotalTime: 3.02s (3016602849)

DataStreamReceiver
- BytesReceived: 402 B (402)
- DeserializeRowBatchTimer: 0ns (0)
- FirstBatchArrivalWaitTime: 1.83s (1830275553)
- InactiveTotalTime: 0ns (0)
- SendersBlockedTimer: 0ns (0)
- SendersBlockedTotalTimer(*): 0ns (0)

DataStreamSender (dst_id=3)
- BytesSent: 402 B (402)
- InactiveTotalTime: 0ns (0)
- NetworkThroughput(*): 87.3 KiB/s (89436)
- OverallThroughput: 283.6 KiB/s (290417)
- PeakMemoryUsage: 120.6 KiB (123488)
- RowsReturned: 24 (24)
- SerializeBatchTime: 0ns (0)
- TotalTime: 888.97us (888969)
- TransmitDataRPCTime: 222.24us (222241)
- UncompressedRowBatchSize: 504 B (504)

# Network slowness

- Exchange performance issues
  - Too much data across network:
    - Check the query on data size reduction.
    - Check join order and join strategy; Wrong join order/strategy can have a serious effect on network!
    - For agg, check the number of groups – affect memory too!
    - Remove unused columns.
  - Keep in mind that network is typically at most 10Gbit
- Cross-rack network slowness

# More examples/Exercises

- Parquet min/max filter, dictionary filter
- Spill to disk
- CPU saturated
- Scanner thread
- Detect small files
- Sink  (DML queries)

# Recap: Query Tuning

- Performance: Examine the logic of the query and validate the Explain Plan
  - Validate join order and join strategy.
  - Validate partition pruning and/or predicate pushdown.
  - Validate plan parallelism.
- Memory usage: Top causes (in order) of hitting mem-limit:
  - Lack of statistics
  - Lots of joins within a single query
  - Big-table joining big-table
  - Gigantic group by (e.g., distinct)
  - Parquet Writer

# More resources

- Impala cookbook https://blog.cloudera.com/blog/2017/02/latest-impala-cookbook/
- Impala perf guideline https://www.cloudera.com/documentation/enterprise/latest/topics/impala_perf_cookbook.html
- Perf optimization https://conferences.oreilly.com/strata/strata-ca-2017/public/schedule/detail/55783

# Thank you

Q & A

cloudera