Trino(Presto)DB: Zero Copy Lakehouse

Artem Aliev

Huawei

Artem Aliev

- Huawei Cloud Hybrid Integration Platform
  - Expert and solution architect
- 20+ years in Software Development
  - Big data platforms integrations
  - Apache Hadoop, Spark, Cassandra, TinkerPop
  - Storage optimizations
  - JVM development
- SpbU teacher

artem.aliev@gmail.com

Application scenarios

- Data enrichment and composition services
- Multi-datasource, multi-cloud, micro service environment
- Exploration analytic
  - What else we have for analyses?
- Fraud/Security breach detection and prevention
- ML model inference

Application scenarios

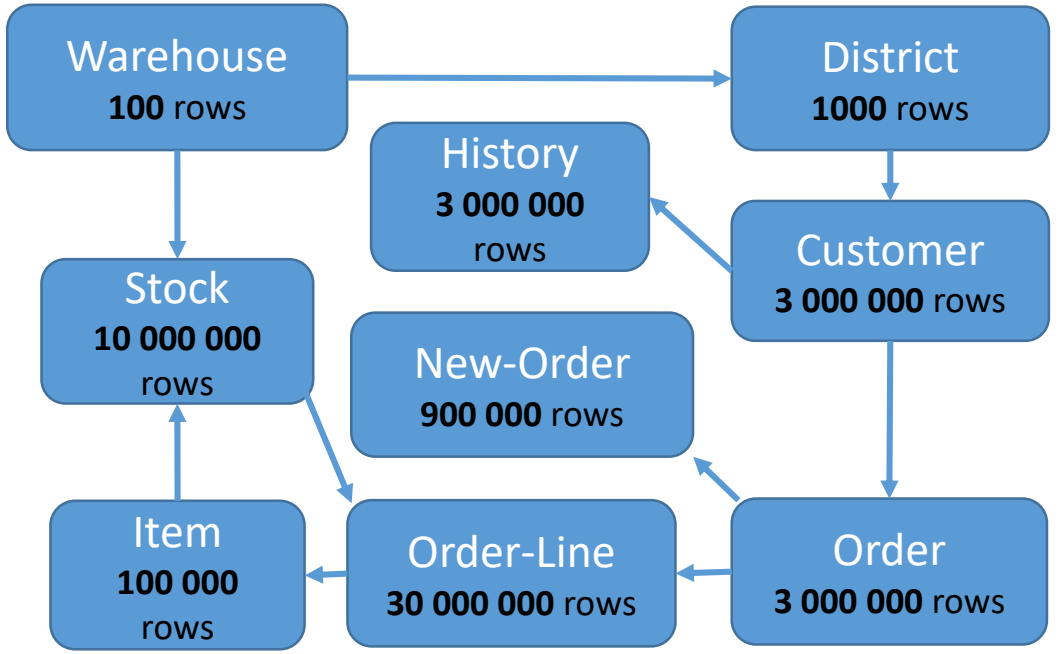
- Data enrichment and composition services
- Multi-datasource, multi-cloud, micro service environment
- Exploration analytic
  - What else we have for analyses?
- Fraud/Security breach detection and prevention
- ML model inference

Requirements

- Interactive queries (join queries)
  - Seconds for analytics
  - Sub-seconds for user services
- Different Data Sources
  - SQL/NoSQL databases
  - S3 files and Hadoop Systems
  - REST Services
- Consistent up-to-date results
- Open Source

Example (tpc-c)

Show user history for the given warehouse.




```
select distinct i_name, i_price
from warehouse
join district on (w_id = d_w_id)
join customer on (d_w_id = c_w_id and d_id = c_d_id)
join orders on (o_w_id = w_id and o_d_id = d_id and o_c_id = c_id)
join order_line on (o_w_id = ol_w_id and o_d_id = ol_d_id and o_id = ol_o_id)
join stock on (ol_supply_w_id = s_w_id and ol_i_id = s_i_id)
join item on (s_i_id = i_id)
where w_id = 50 and c_id = 101;
```

	seconds
MPP DB	20-80
Tuned Trino	4
Postgres	0.7

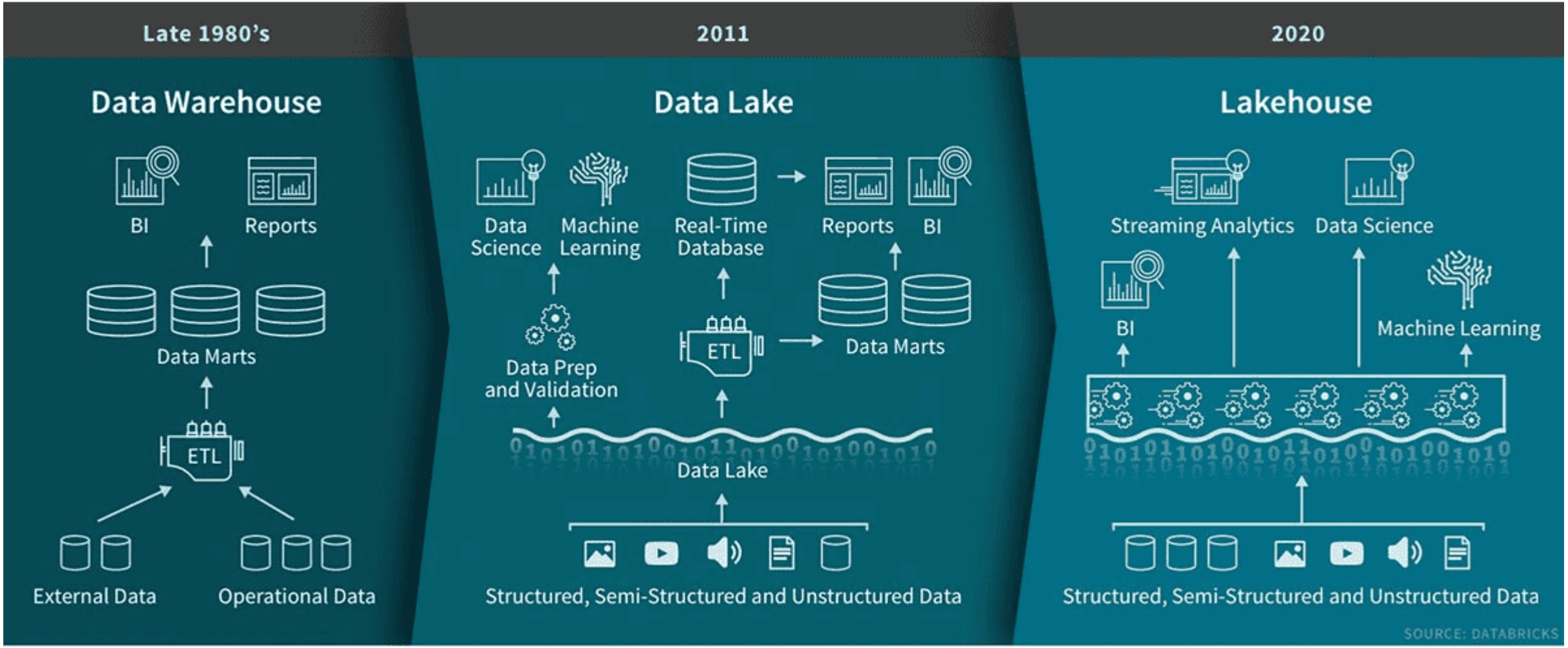
Traditional Stack

- Data Lake
  - Hive, Spark, Impala, Trino, Drill, Dremio*
- Data warehouse
  - ClickHouse, Greenplum, Vertica*
- Data marts
  - Postgres, Mysql, ClickHouse

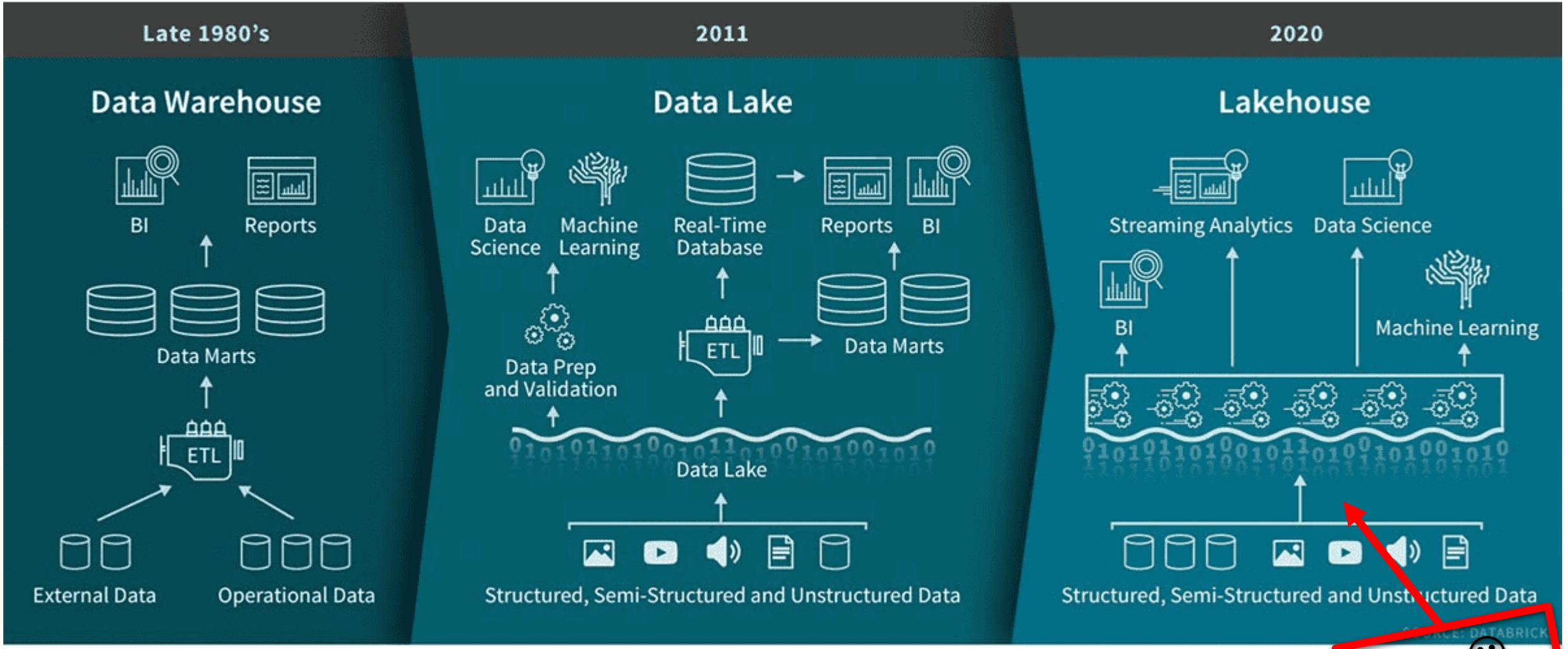
ETL/ELT from sources to data marts

- Nightly by batches
- Streaming
  - Fast
  - Need special database to enrich and join data in the stream
    - Redis, Cassandra, etc..
  - Eager enrichments
- Both fights with:
  - Data source model changes
  - Loading failures
  - Inconsistent loading

Databricks Solution: Lakehouse



Databricks Solution: Lakehouse



NO ETL!

- Big Data as usual DataBase
- Direct request to Data Sources

Micro service architecture support

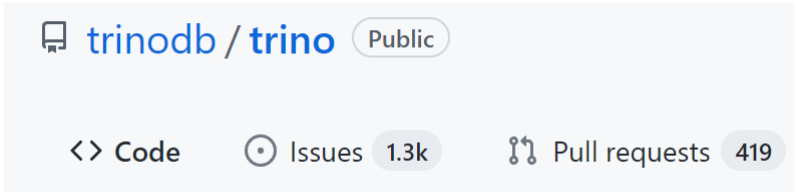
- A lot of small exotic databases
- “Agile” development with a lot of schema changes
- REST API data access only
- Pay per request
  - Google API, etc

Feature requirements summary

- Schema changes tolerance
- Advanced pushdowns to data sources and optimizations
  - Legacy databases are still better in indexing
- No ETL
  - Extreme: No caches, local materialized views, reflections, etc.
- Avoid full scans
- REST endpoint support
- Open Source

Candidate tested

- Postgres with FDW
  - Very old and unsupported plugins
  - Pushdowns works only with other Postgres
- Drill – schema-free for Hadoop
  - Not in active development
  - Optimizer is not good
- TrinoDB
  - Very easy REST connector development
- Dremio -- not really Open Source
- Hive, Spark – files and manual jdbc only

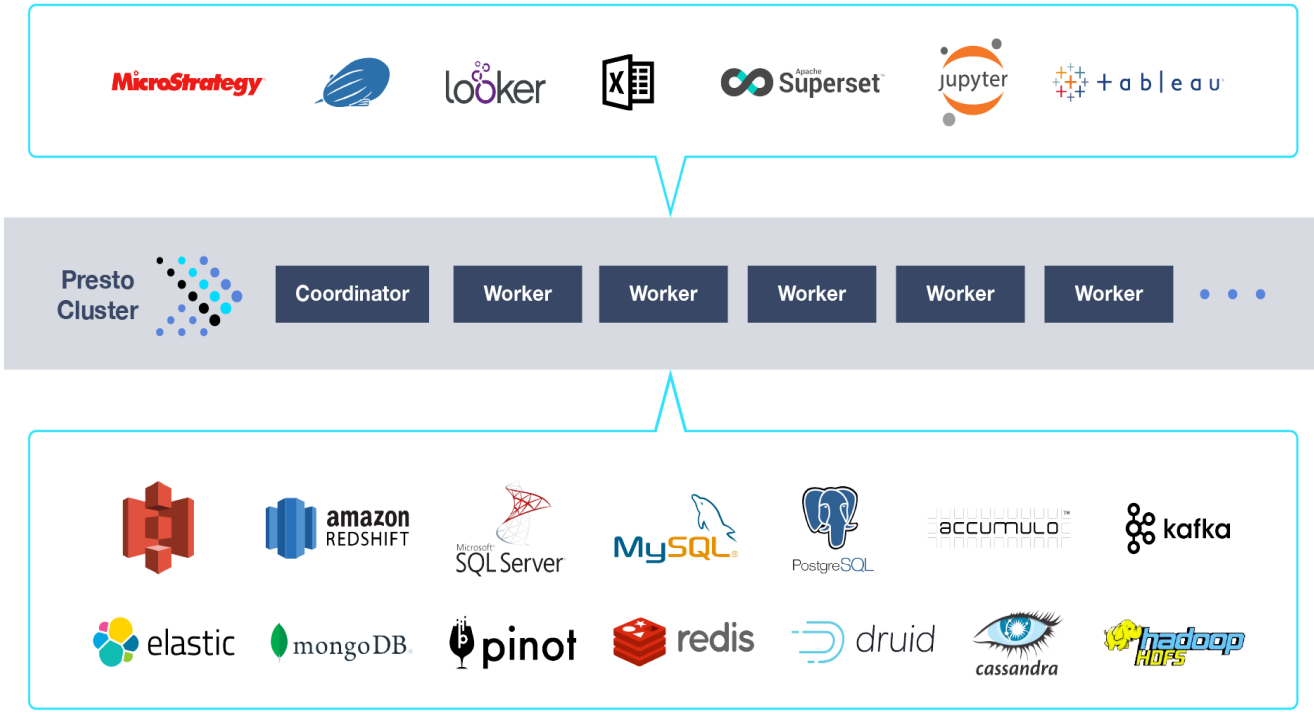


The winner is: Presto

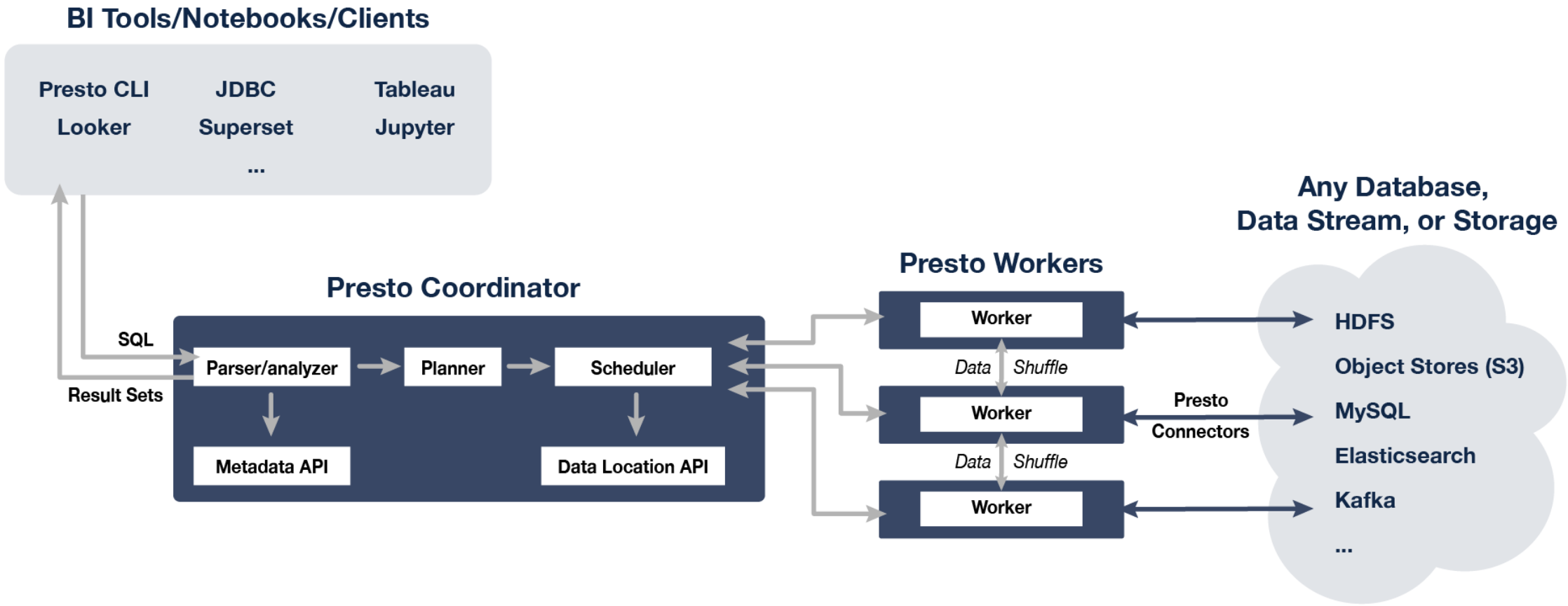
- Facebook develop Presto at 2012 and release to OS at 2013
- 2019
  - PrestoDB supported by Facebook in Linux Foundation
    - https://github.com/prestodb/presto
  - PrestoSQL supported by Starburst
    - 2020 Renamed to TrinoDB
      - https://github.com/trinodb/trino
    - 2020 OpenLooKeng from Huawei
      - https://gitee.com/openlookeng/hetu-core
- Cloud Services

TrinoDB/PrestoDB

- SQL
- 30+ connectors
- Easy to develop new connectors
- Dynamic Catalog
  - Represent data as tables
  - In schema, in catalog
  - Common type system
    - Type conversions for columns
    - Query planner is types aware



Classical Distributed Architecture



Adding Datasouce

- Just drop a property file into etc/catalog directory
- File name is a catalog name
- Schemas and tables will be loaded from the connector

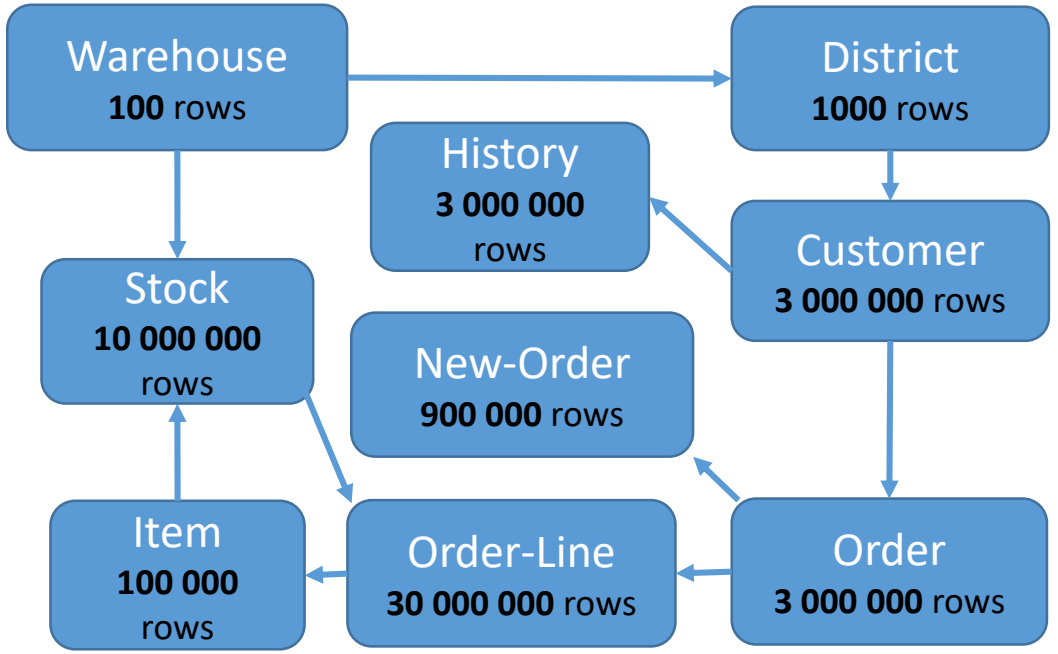
```
connector.name=postgresql
connection-url=jdbc:postgresql://localhost:5432/tpcc
connection-user=postgres
connection-password=password
```

Great Optimization Engine

- Cost based optimizations (CBO)
  - Hive connector only ☺
- Pushdowns
  - Predicate
    - Optimizer propagates constants through joins
    - Dynamic filtering support for joins (base on CBO)
  - Projection
  - Aggregation!
  - **JOIN***
  - TOP-N and LIMITs
    - ORDER BY ... LIMIT N or ORDER BY ... FETCH FIRST N ROWS

Highly-Selective Join

Show user history for given warehouse.




```
select distinct i_name, i_price
from warehouse
join district on (w_id = d_w_id)
join customer on (d_w_id = c_w_id and d_id = c_d_id)
join orders on (o_w_id = w_id and o_d_id = d_id and o_c_id = c_id)
join order_line on (o_w_id = ol_w_id and o_d_id = ol_d_id and o_id = ol_o_id)
join stock on (ol_supply_w_id = s_w_id and ol_i_id = s_i_id)
join item on (s_i_id = i_id)
where w_id = 50 and c_id = 101;
```

	seconds
MPP DB	20-80
Tuned Trino	4
Postgres	0.7

Nested Loop Join

Postgres

```
Unique  (cost=1389.63..1390.36 rows=97 width=24)
  ->  Sort  (cost=1389.63..1389.88 rows=97 width=24)
        Sort Key: item.i_name, item.i_price
        ->  Nested Loop  (cost=2.42..1386.43 rows=97 width=24)
              ->  Nested Loop  (cost=2.13..964.80 rows=97 width=8)
                    ->  Nested Loop  (cost=1.70..424.70 rows=97 width=8)
 Join Filter: (district.d_id = order_line.ol_d_id)
 ->  Nested Loop  (cost=1.14..178.08 rows=10 width=20)
       ->  Nested Loop  (cost=0.71..108.69 rows=9 width=24)
             ->  Nested Loop  (cost=0.71..105.32 rows=9 width=24)
                   ->  Index Only Scan using pk_district on district  (cost=0.28..20.70 rows=10
 Index Cond: (d_w_id = 50)
                   ->  Index Scan using i_orders on orders  (cost=0.43..8.45 rows=1 width=16
 Index Cond: ((o_w_id = 50) AND (o_d_id = district.d_id) AND (o_c_id = 101))
              ->  Materialize  (cost=0.00..3.25 rows=1 width=4)
                    ->  Seq Scan on warehouse  (cost=0.00..3.25 rows=1 width=4)
 Filter: (w_id = 50)
       ->  Index Only Scan using customer_pkey on customer  (cost=0.43..8.45 rows=1 w
             Index Cond: (**(c_w_id = 50)** AND (c_d_id = district.d_id) AND **(c_id = 101)**)
 ->  Index Scan using pk_order_line on order_line  (cost=0.56..24.54 rows=10 width=2
       Index Cond: ((ol_w_id = 50) AND (ol_d_id = orders.o_d_id) AND (ol_o_id = orders
                   ->  **Index Only Scan** using pk_stock on stock  (cost=0.43..5.57 rows=1 wid
 Index Cond: ((s_w_id = order_line.ol_supply_w_id) AND (s_i_id = order_line.ol_i_id))
              ->  **Index Scan** using pk_item on item  (cost=0.29..4.35 rows=1 width=28)
                    Index Cond: (i_id = order_line.ol_i_id)
```

Trino

```
└─ InnerJoin[("d_id" = "o_d_id")][$hashvalue_15, $hashvalue_20]                >
   Layout: [d_id:integer, o_id:integer]                                        >
   Estimates: {rows: ? (?), cpu: ?, memory: ?, network: ?}                     >
   Distribution: PARTITIONED                                                   >
   maySkipOutputDuplicates = true                                              >
   dynamicFilterAssignments = {o_d_id -> #df_1300}                             >
   ├─ InnerJoin[("d_id" = "c_d_id")][$hashvalue_15, $hashvalue_17]             >
   │  Layout: [d_id:integer, $hashvalue_15:bigint]                             >
   │  Estimates: {rows: ? (?), cpu: ?, memory: ?, network: ?}                  >
   │  Distribution: PARTITIONED>
   │  maySkipOutputDuplicates = true                                          >
   │  dynamicFilterAssignments = {c_d_id -> #df_1301}                         >
   │  ├─ RemoteExchange[REPARTITION][$hashvalue_15]                           >
   │  │  Layout: [d_id:integer, $hashvalue_15:bigint]                         >
   │  │  Estimates: {rows: ? (?), cpu: ?, memory: 0B, network: ?}             >
   │  │  └─ Project[]                                                         >
   │  │     Layout: [d_id:integer, $hashvalue_16:bigint]>
   │  │     Estimates: {rows: ? (?), cpu: ?, memory: 0B, network: 0B}         >
   │  │     $hashvalue_16 := combine_hash(bigint '0', COALESCE("$operator$hash_co
   │  │     └─ CrossJoin                                                      >
   │  │        Layout: [d_id:integer]                                         >
   │  │        Estimates: {rows: ? (?), cpu: ?, memory: 0B, network: 0B}      >
   │  │        Distribution: REPLICATED                                       >
   │  │        maySkipOutputDuplicates = true                                 >
   │  │        ├─ ScanFilter[table = postgresql:public.district public.district constraint on [
```

Nested Loop Join

Postgres

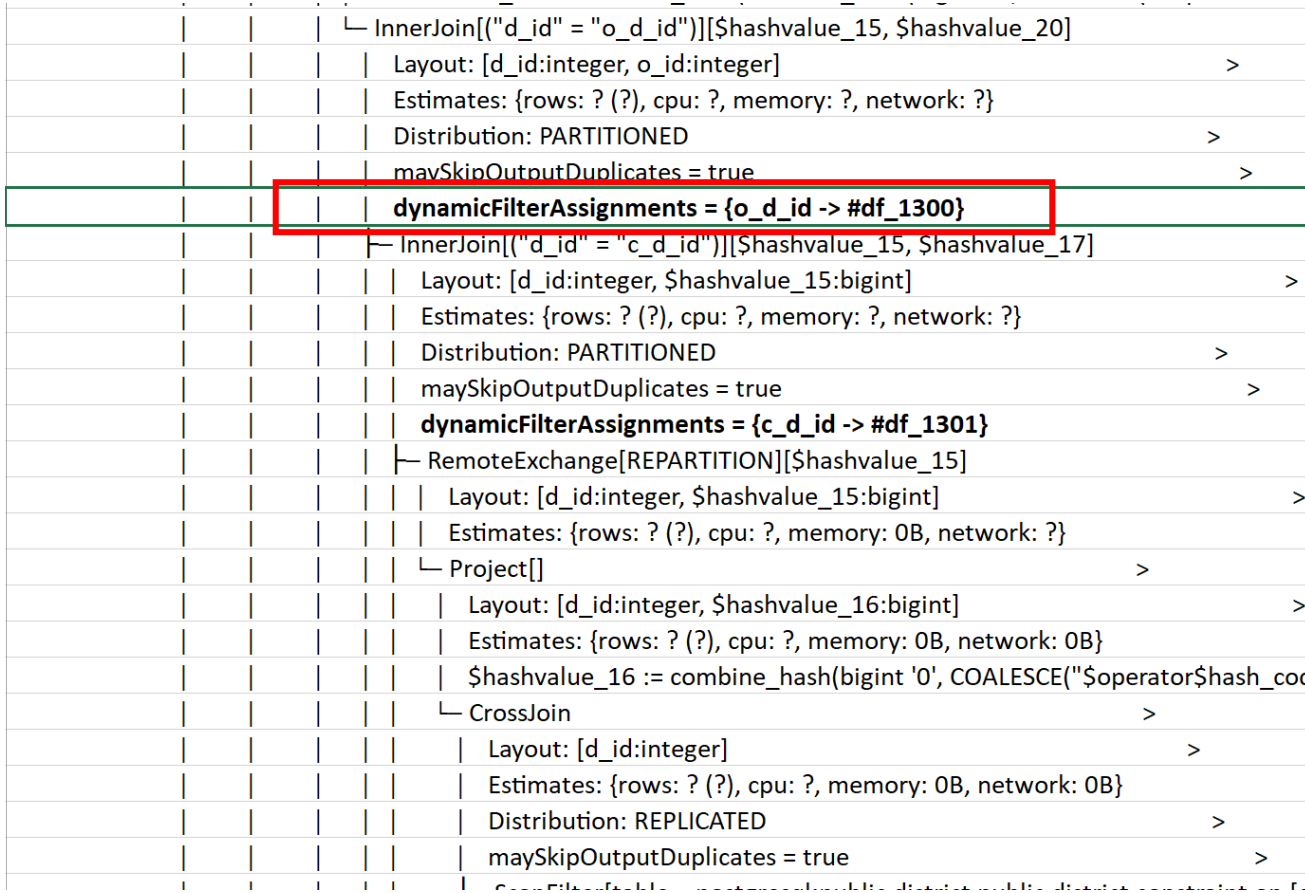
```
Unique  (cost=1389.63..1390.36 rows=97 width=24)
  ->  Sort  (cost=1389.63..1389.88 rows=97 width=24)
        Sort Key: item.i_name, item.i_price
        ->  Nested Loop  (cost=2.42..1386.43 rows=97 width=24)
              ->  Nested Loop  (cost=2.13..964.80 rows=97 width=8)
                    ->  Nested Loop  (cost=1.70..424.70 rows=97 width=8)
 Join Filter: (district.d_id = order_line.ol_d_id)
 ->  Nested Loop  (cost=1.14..178.08 rows=10 width=20)
       ->  Nested Loop  (cost=0.71..108.69 rows=9 width=24)
             ->  Nested Loop  (cost=0.71..105.32 rows=9 width=24)
                   ->  Index Only Scan using pk_district on district  (cost=0.28..20.70 rows=10
 Index Cond: (d_w_id = 50)
                   ->  Index Scan using i_orders on orders  (cost=0.43..8.45 rows=1 width=16
 Index Cond: ((o_w_id = 50) AND (o_d_id = district.d_id) AND (o_c_id = 101))
             ->  Materialize  (cost=0.00..3.25 rows=1 width=4)
                   ->  Seq Scan on warehouse  (cost=0.00..3.25 rows=1 width=4)
 Filter: (w_id = 50)
    ->  Index Only Scan using customer_pkey on customer  (cost=0.43..8.45 rows=1 w
          Index Cond: ((c_w_id = 50) AND (c_d_id = district.d_id) AND (c_id = 101))
 ->  Index Scan using pk_order_line on order_line  (cost=0.56..24.54 rows=10 width=2
       Index Cond: ((ol_w_id = 50) AND (ol_d_id = orders.o_d_id) AND (ol_o_id = orders
                   ->  Index Only Scan using pk_stock on stock  (cost=0.43..5.57 rows=1 wid
 Index Cond: ((s_w_id = order_line.ol_supply_w_id) AND (s_i_id = order_line.ol_i_id))
             ->  Index Scan using pk_item on item  (cost=0.29..4.35 rows=1 width=28)
```

Trino

```
                    └─ InnerJoin[("d_id" = "o_d_id")][$hashvalue_15, $hashvalue_20]                >
                       Layout: [d_id:integer, o_id:integer]                >
                       Estimates: {rows: ? (?), cpu: ?, memory: ?, network: ?}                >
                       Distribution: PARTITIONED  >
                       maySkipOutputDuplicates = true                >
                       dynamicFilterAssignments = {o_d_id -> #df_1300}     >
                    ├─ InnerJoin[("d_id" = "c_d_id")][$hashvalue_15, $hashvalue_17]                >
                    │  Layout: [d_id:integer, $hashvalue_15:bigint]     >
                    │  Estimates: {rows: ? (?), cpu: ?, memory: ?, network: ?}                >
                    │  Distribution: PARTITIONED>
                    │  maySkipOutputDuplicates = true                >
                    │  dynamicFilterAssignments = {c_d_id -> #df_1301}  >
                    ├─ RemoteExchange[REPARTITION][$hashvalue_15]       >
                    │  │  Layout: [d_id:integer, $hashvalue_15:bigint]  >
                    │  │  Estimates: {rows: ? (?), cpu: ?, memory: 0B, network: ?}                >
                    │  └─ Project[]           >
                    │     Layout: [d_id:integer, $hashvalue_16:bigint]>
                    │     Estimates: {rows: ? (?), cpu: ?, memory: 0B, network: 0B}          >
                    │     $hashvalue_16 := combine_hash(bigint '0', COALESCE("$operator$hash
                    │     └─ CrossJoin          >
                    │        Layout: [d_id:integer]                >
                    │        Estimates: {rows: ? (?), cpu: ?, memory: 0B, network: 0B}       >
                    │        Distribution: REPLICATED           >
                    │        maySkipOutputDuplicates = true         >
```

First Attempt: Dynamic Filtering

- Collect ids from the right side
- Push ids to the left side join
- CBO is recommended
- Hive and Memory supported
- JDBC PR #7968



Secret Index Joins for Thrift Connector

- Is used to integrate external storage system without connector.
- Just wrap you service with ThriftServer
- Works for REST API!
- Wrapping JDBC

Is inconvenient

```
/**
 * Trino Thrift service definition.
 * This thrift service needs to be implemented
 */
service TrinoThriftService {
  /**
   * Returns available schema names.
   */
  list<string> trinoListSchemaNames()
```

```
  TrinoThriftSplitBatch trinoGetIndexSplits(
```

Apache Thrift overview

- Thrift is Remote Procedure Call Server development framework
- Development:
  - Describe interface in .thrift file.
  - Generate service interface and client code:

    ```
    thrift --gen java TrinoThriftService.thrift
    ```
  - Implement interfaces for the server
- Trino example ThriftTpchServer

Adding Index to JDBC connector

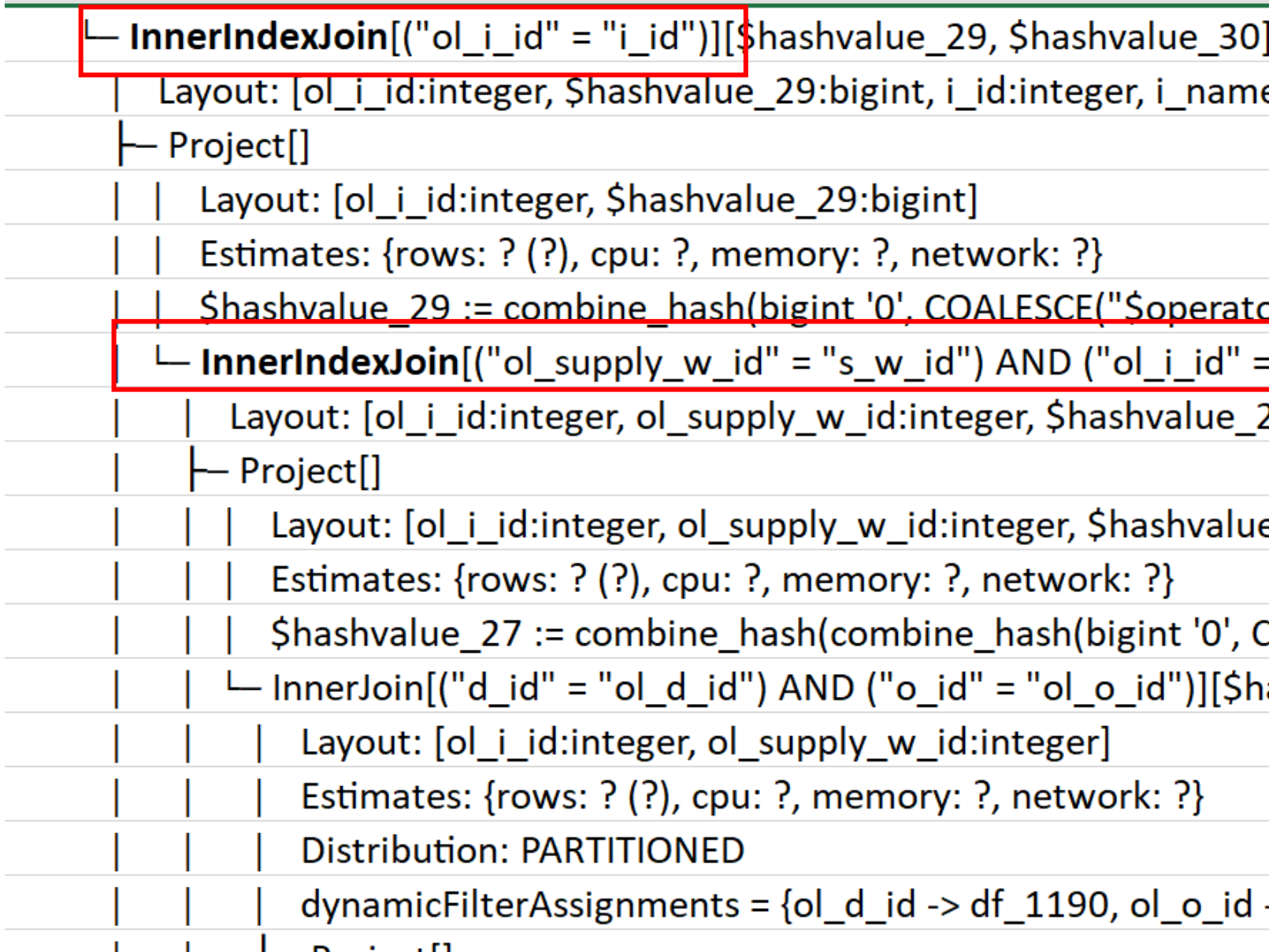
- Just add ;)

```
public interface ConnectorIndexProvider
{
    ConnectorIndex getIndex(ConnectorTransactionHandle transactionHandle,
            ConnectorSession session,
            ConnectorIndexHandle indexHandle,
            List<ColumnHandle> lookupSchema,
            List<ColumnHandle> outputSchema);
}
```

- Not in open source yet

Fixed:

- From 80 sec to 4



REST API and micro services

- Faceboook use(d) ThriftService
  - Create thrift server for your microservices
- trino-example-http connector
  - Modify for your needs
  - Don’t forget about Index Provider
- We developed simple configurable connector for our internal services

Zero Copy Done!

- No need to build huge data lake with a lot of servers a head of time
- Single node TrinoDB could do data exploration

Let see other features:

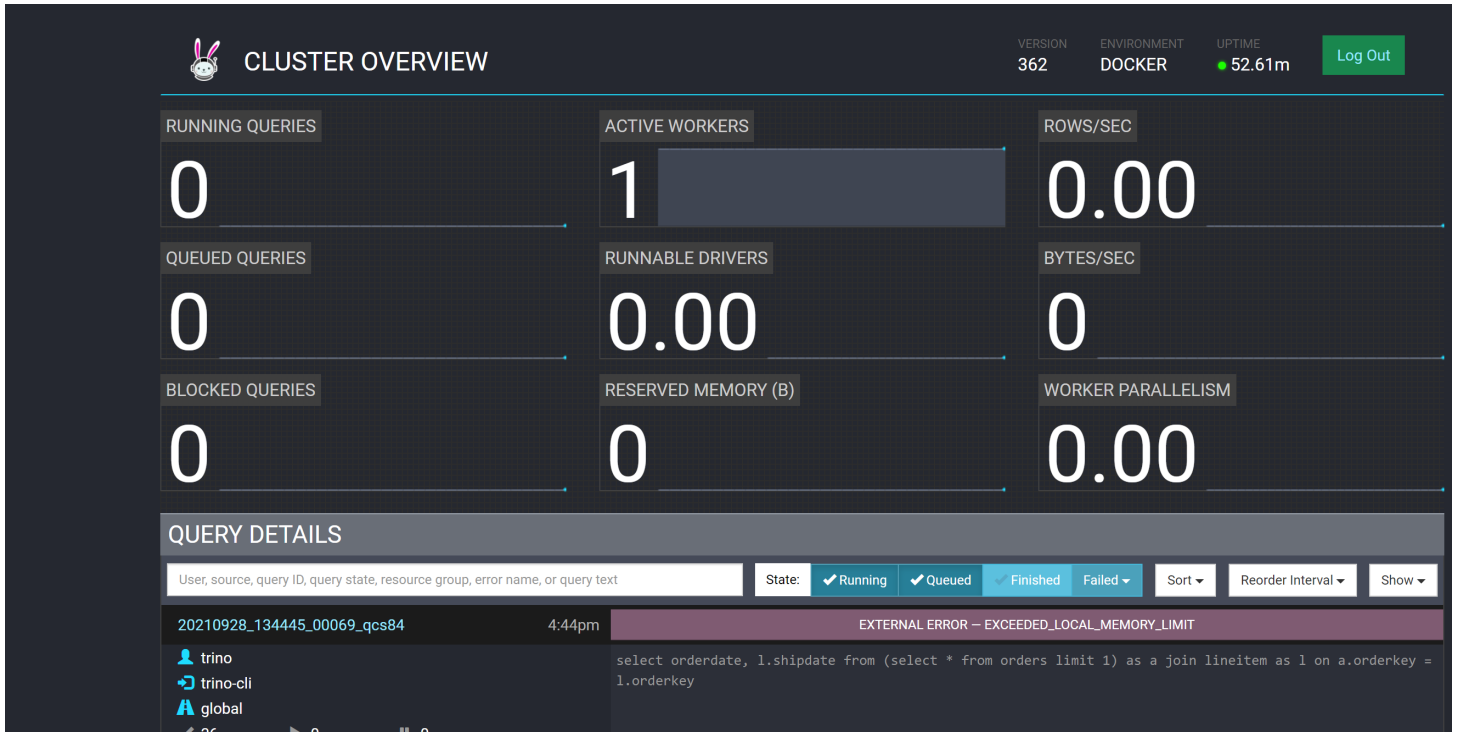
Security

- HTTPS with TLS 1.2, 1.3
- User auth: Password,LDAP,Oauth,Kerberos,JWT,Certificate
- Access Control
  - up to table operations
  - System operations

```
{
  "tables": [
    {
      "role": "admin",
      "privileges": ["SELECT", "INSERT", "DELETE", "OWNERSHIP"]
    },
    {
      "user": "banned_user",
      "privileges": []
    },
    {
      "catalog": "default",
      "schema": "hr",
      "table": "employee",
      "privileges": ["SELECT"],
      "filter": "user = current_user",
      "filter_environment": {
        "user": "system_user"
      }
    },
```

Administration

- Web UI for monitoring
- JMX monitoring
- Resource groups
  - Memory, CPU limits
  - Queues
- Spill to disk support



Dynamic datasource reconfiguration

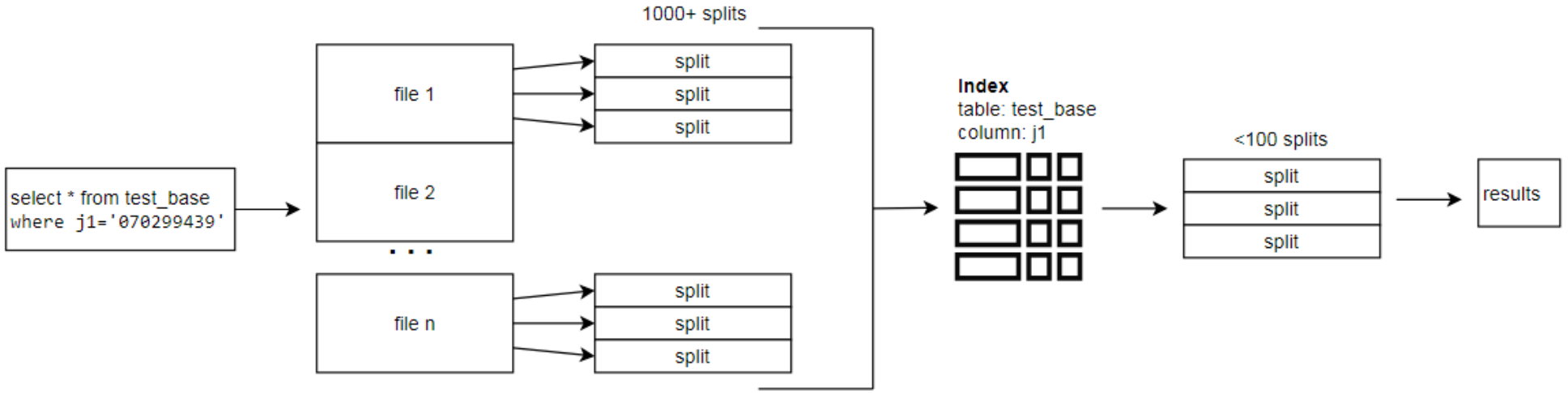
- Static property files by default
- PR: #12605
- OpenLooKeng fork

Caching

- Alluxio FS cache for Hive
- Memory connector

Indexing for Hive

- OpenLooKeng exclusive feature
- Bloom, Btree, MinMax,Bitmap indexes



High Availability

- OpenLooKeng
  - Active-Active base on distributed cache
- Use standard approaches for microservices
  - K8s

Try it: Lakehouse microserivce

```
#> docker run -p 8080:8080 --name trino trinodb/trino
```

Connect cli:

```
#> docker exec -ti trino trino
```

For “production” usage just store catalog in the git and mount it into the docker

```
#> docker run --rm -p 8080:8080 \
        -v /opt/trino_catalog_git:/etc/trino/catalog \
        --name trino trinodb/trino
```

Run some commands

```
trino> show catalogs;
 Catalog
---------
 jmx
 memory
 system
 tpcds
 tpch
(5 rows)

Query 20210928_130006_00000_qcs84, FINISHED, 1 node
Splits: 19 total, 19 done (100.00%)
1.35 [0 rows, 0B] [0 rows/s, 0B/s]
```

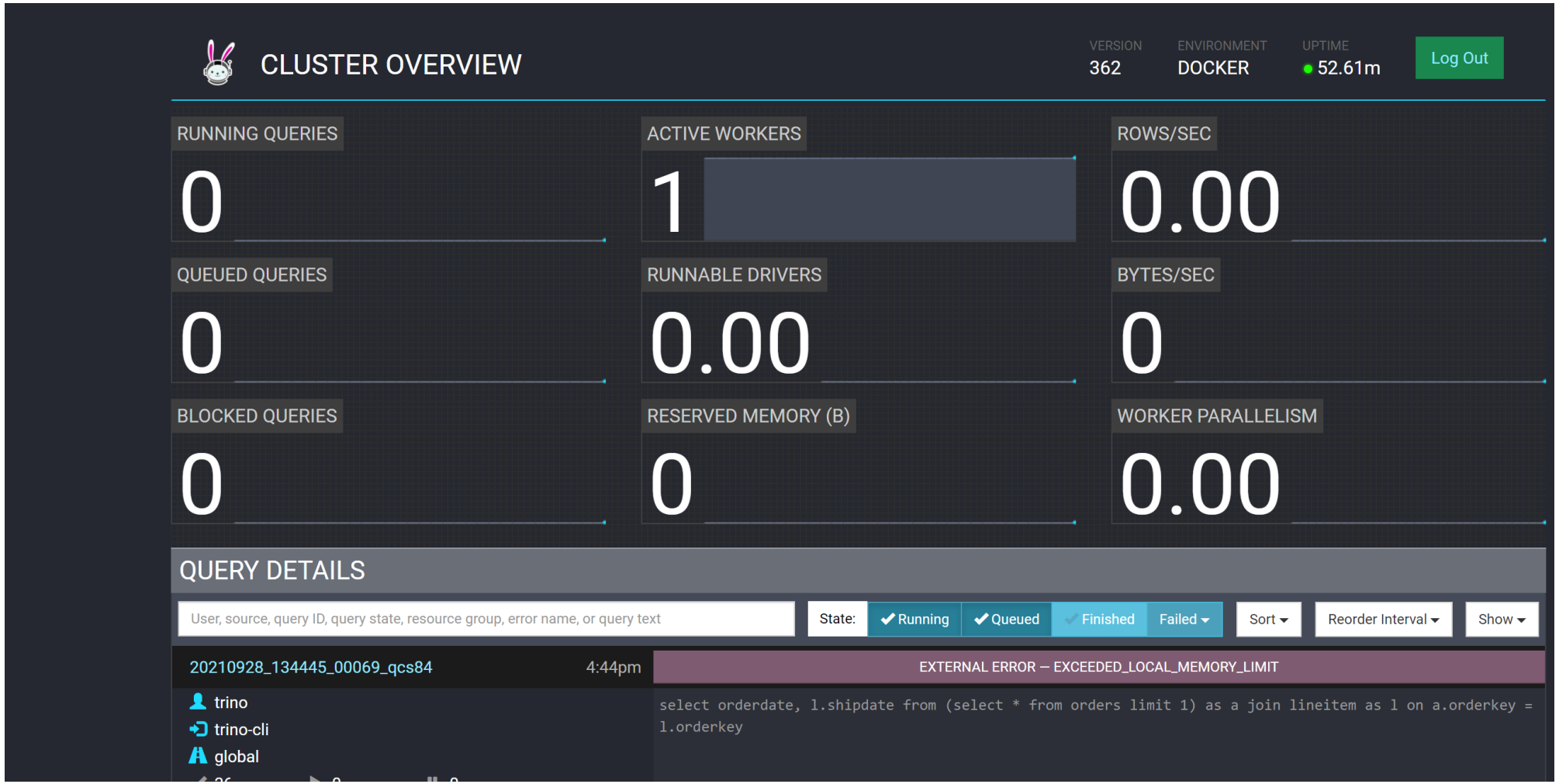
Sample data the right way

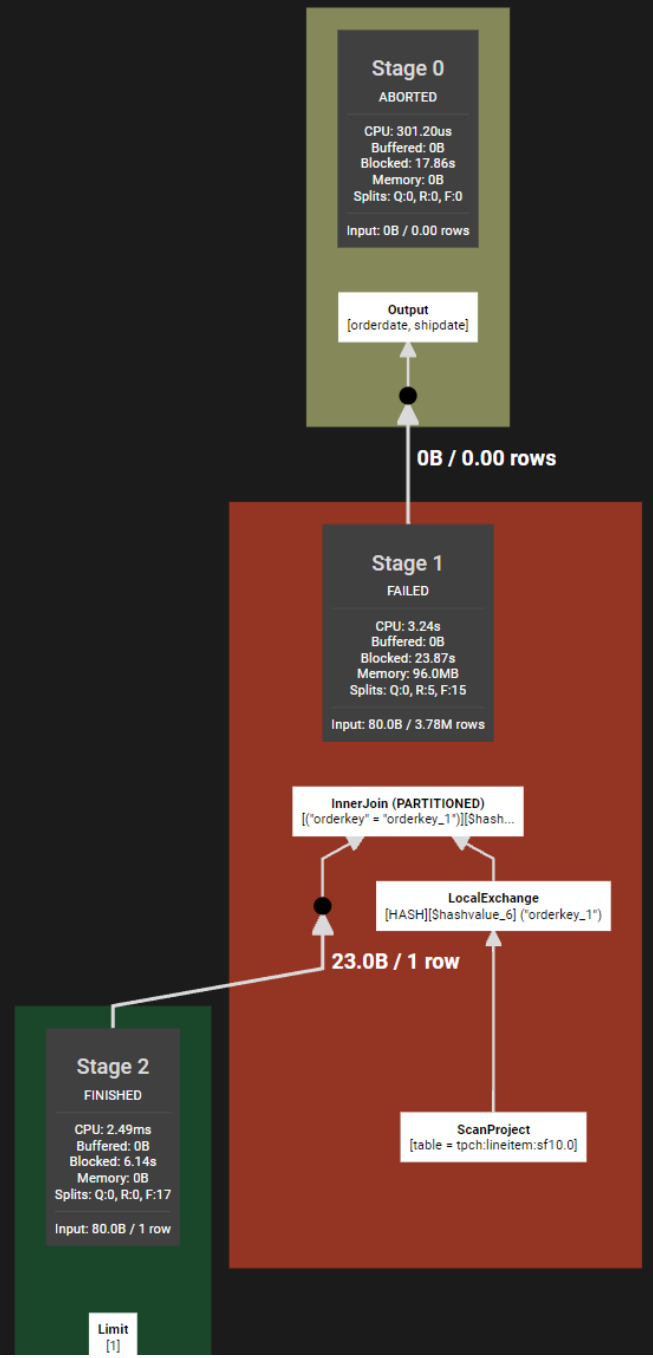
```
trino> use tpch.sf10;
USE
trino:sf10>
```

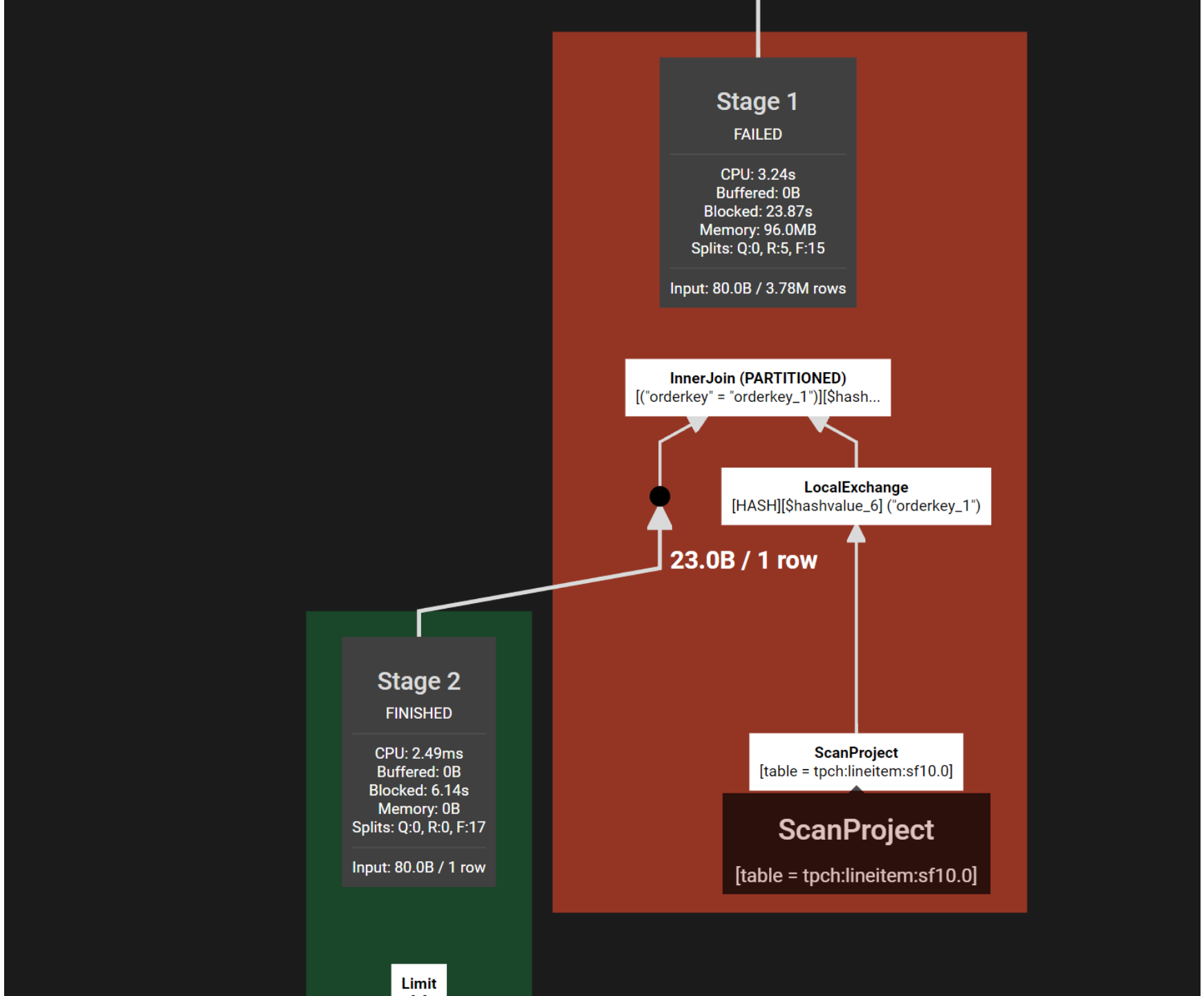
```
trino:sf10> select a.orderkey , orderdate, l.shipdate from orders as a join lineitem
as l on a.orderkey = l.orderkey limit 1;

Query 20210928_134134_00066_qcs84, FAILED, 1 node
Splits: 57 total, 0 done (0.00%)
1.56 [0 rows, 0B] [0 rows/s, 0B/s]

Query 20210928_134134_00066_qcs84 failed: Query exceeded per-node user memory limit o
f 102.40MB [Allocated: 102.39MB, Delta: 28.61kB, Top Consumers: {HashBuilderOperator=
102.39MB, PartitionedOutputOperator=1.50kB, ScanFilterAndProjectOperator=1.50kB}]
```

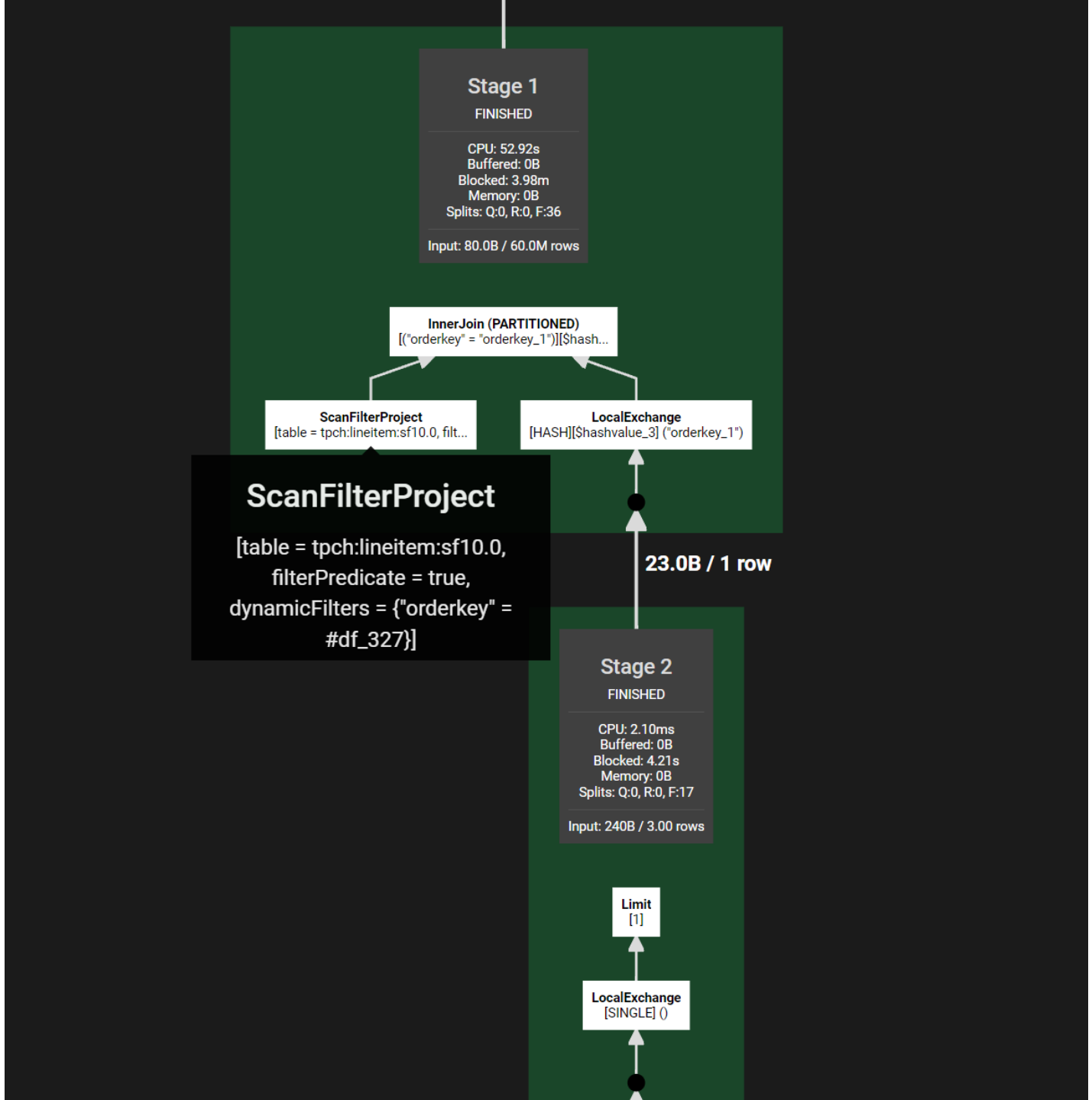
```
trino:sf10> select orderdate, l.shipdate from orders as a join (select * from lineite
m limit 1) as l on a.orderkey = l.orderkey;
 orderdate  |  shipdate
------------+------------
 1996-01-02 | 1996-03-13
(1 row)
```

Web UI


CLUSTER OVERVIEW
VERSION
362
ENVIRONMENT
DOCKER
UPTIME
52.61m
Log Out
RUNNING QUERIES
0
ACTIVE WORKERS
1
ROWS/SEC
0.00
QUEUED QUERIES
0
RUNNABLE DRIVERS
0.00
BYTES/SEC
0
BLOCKED QUERIES
0
RESERVED MEMORY (B)
0
WORKER PARALLELISM
0.00
QUERY DETAILS
User, source, query ID, query state, resource group, error name, or query text
State:
Running
Queued
Finished
Failed
Sort
Reorder Interval
Show
20210928_134445_00069_qcs84
4:44pm
EXTERNAL ERROR — EXCEEDED_LOCAL_MEMORY_LIMIT
trino
trino-cli
global
select orderdate, l.shipdate from (select * from orders limit 1) as a join lineitem as l on a.orderkey = l.orderkey


Stage 0
ABORTED
CPU: 301.20us
Buffered: 0B
Blocked: 17.86s
Memory: 0B
Splits: Q:0, R:0, F:0
Input: 0B / 0.00 rows
Output
[orderdate, shipdate]
0B / 0.00 rows
Stage 1
FAILED
CPU: 3.24s
Buffered: 0B
Blocked: 23.87s
Memory: 96.0MB
Splits: Q:0, R:5, F:15
Input: 80.0B / 3.78M rows
InnerJoin (PARTITIONED)
[("orderkey" = "orderkey_1")][$hash...
LocalExchange
[HASH][$hashvalue_6] ("orderkey_1")
23.0B / 1 row
ScanProject
[table = tpch:lineitem:sf10.0]
Stage 2
FINISHED
CPU: 2.49ms
Buffered: 0B
Blocked: 6.14s
Memory: 0B
Splits: Q:0, R:0, F:17
Input: 80.0B / 1 row
Limit
[1]


Stage 1
FAILED
CPU: 3.24s
Buffered: 0B
Blocked: 23.87s
Memory: 96.0MB
Splits: Q:0, R:5, F:15
Input: 80.0B / 3.78M rows
InnerJoin (PARTITIONED)
[("orderkey" = "orderkey_1")][$hash...
LocalExchange
[HASH][$hashvalue_6] ("orderkey_1")
23.0B / 1 row
Stage 2
FINISHED
CPU: 2.49ms
Buffered: 0B
Blocked: 6.14s
Memory: 0B
Splits: Q:0, R:0, F:17
Input: 80.0B / 1 row
ScanProject
[table = tpch:lineitem:sf10.0]
ScanProject
[table = tpch:lineitem:sf10.0]
Limit


Stage 1
FINISHED
CPU: 52.92s
Buffered: 0B
Blocked: 3.98m
Memory: 0B
Splits: Q:0, R:0, F:36
Input: 80.0B / 60.0M rows
InnerJoin (PARTITIONED)
[("orderkey" = "orderkey_1")][$hash...
ScanFilterProject
[table = tpch:lineitem:sf10.0, filt...
LocalExchange
[HASH][$hashvalue_3] ("orderkey_1")
ScanFilterProject
[table = tpch:lineitem:sf10.0,
filterPredicate = true,
dynamicFilters = {"orderkey" =
#df_327}]
23.0B / 1 row
Stage 2
FINISHED
CPU: 2.10ms
Buffered: 0B
Blocked: 4.21s
Memory: 0B
Splits: Q:0, R:0, F:17
Input: 240B / 3.00 rows
Limit
[1]
LocalExchange
[SINGLE] ()

System catalog

```
trino:runtime> select * from system.runtime.queries limit 1;
          query_id           |  state   | user  |  source   |     query     | resource_group_id | queued_time_ms | analysis_time_ms | planning_time_ms |
-----------------------------+----------+-------+-----------+---------------+-------------------+----------------+------------------+------------------+
 20210928_130006_00000_qcs84 | FINISHED | trino | trino-cli | show catalogs | [global]          |             17 |              203 |              418 |
```

JMX support

- A lot of System Mbeans

```
trino:current> SELECT freebytes, node, object_name
            -> FROM jmx.current."trino.memory:*type=memorypool*";
 freebytes |     node     |                 object_name
-----------+--------------+---------------------------------------
 751619277 | 48943075b7d3 | trino.memory:type=MemoryPool,name=general
(1 row)
```

And so on and so far