Facebook Presto

Interactive and Distributed SQL Query Engine for Big Data

liangguorong@baidu.com, 2014. 11.20

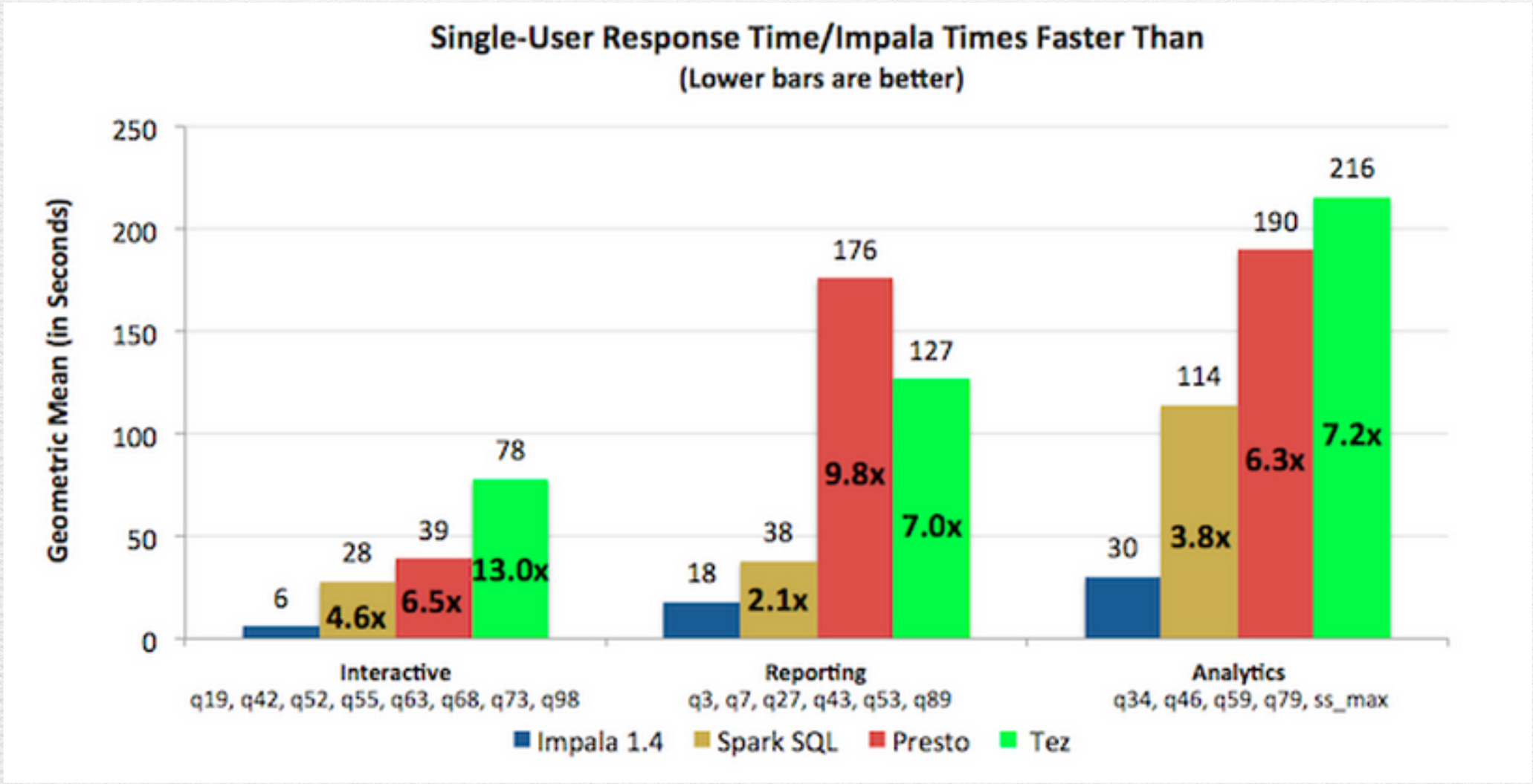
Presto's Brief History

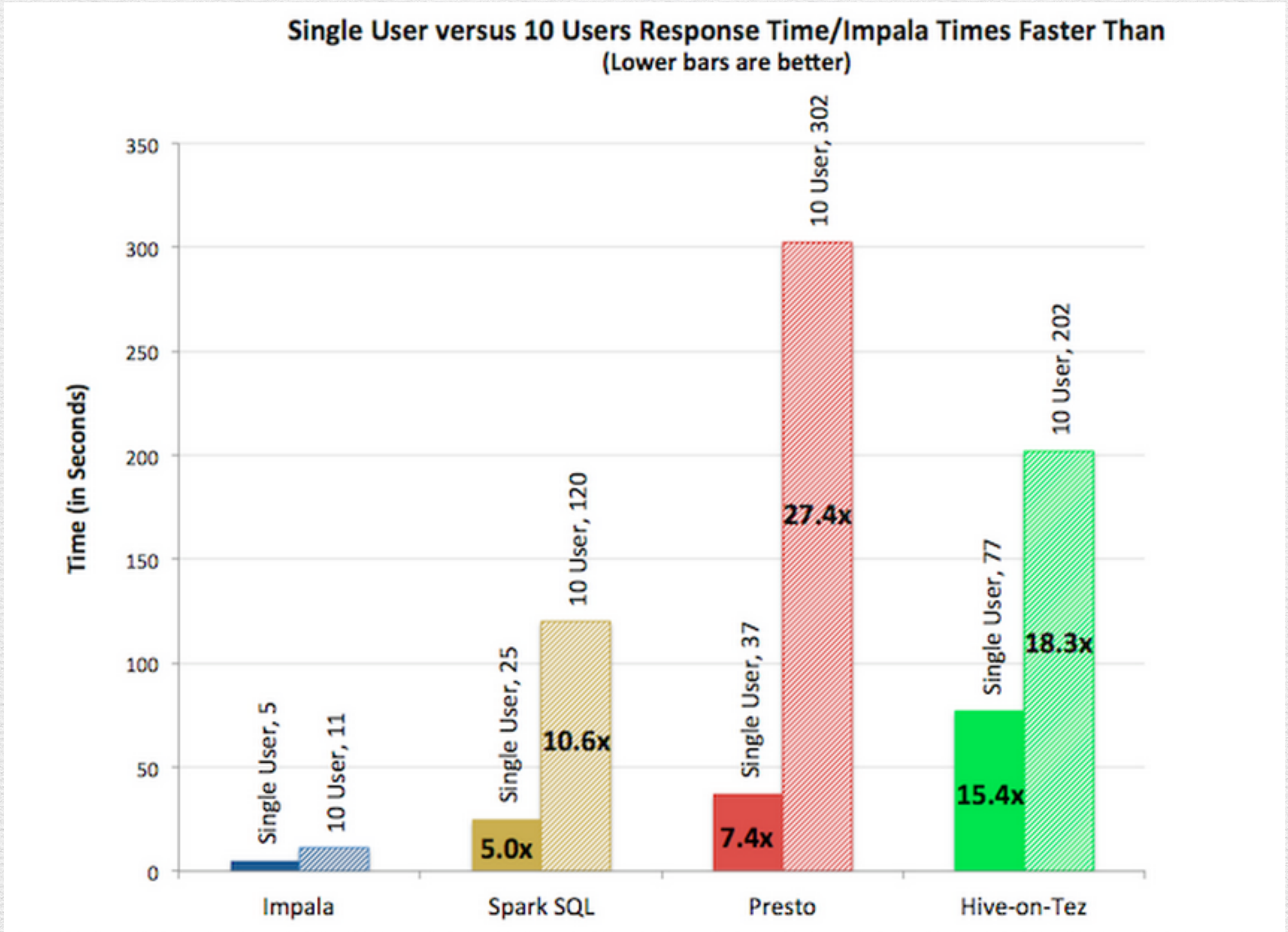
- 2012 fall started at Facebook (6 developers)
  - Designed for interactive SQL query on PB data
  - Hive is for reliable and large scale batch processing
- 2013 spring rolled out to entire company
- 2013 Nov. open sourced (https://github.com/facebook/presto)
- 2014 Nov., 88 releases, 41 contributors, 3943commits
  - current version 0.85 (http://prestodb.io/)
  - java, fast development , java ecosystem, easy integration

Advantages

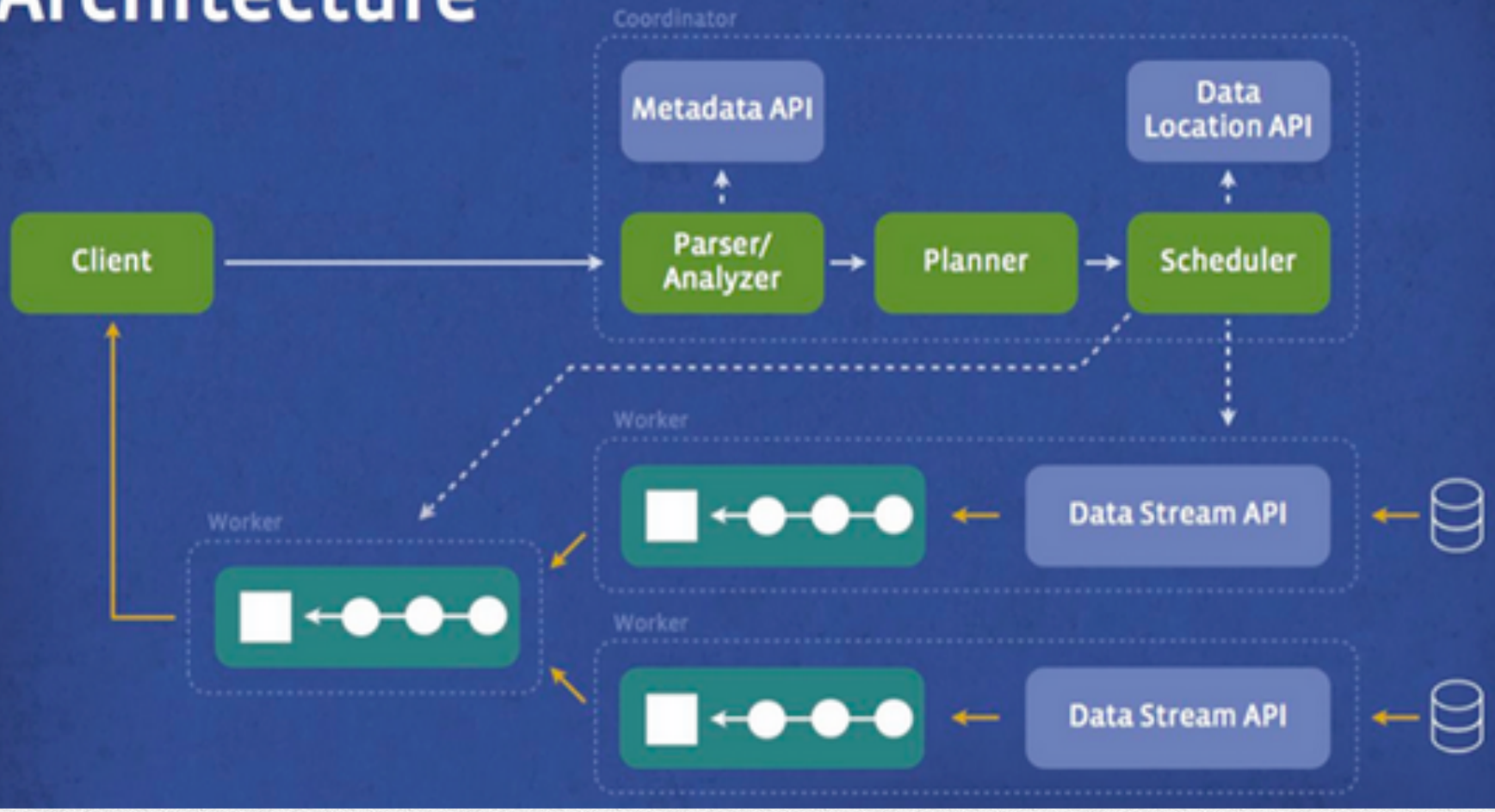
- High Performance: 10x faster than Hive
  - 2013 Nov. Facebook 1000 nodes, 1000 employees run 30,000 queries on 1PB per day
- Extensibility
  - Pluggable backends: Cassandra, Hive, JMX, Kafka, MySQL, PostgreSQL, MySQL, SystemSchema, TPCH
  - JDBC, ODBC(in future) for commercial BI tools or Dashboards, like data visualization
  - Client Protocol: HTTP+JSON, support various languages(Python, Ruby, PHP, Node.js Java(JDBC)…)
- ANSI SQL
  - complex queries, joins, aggregations, various functions(Window functions)

- http://blog.cloudera.com/blog/2014/09/new-benchmarks-for-sql-on-hadoop-impala-1-4-widens-the-performance-gap/

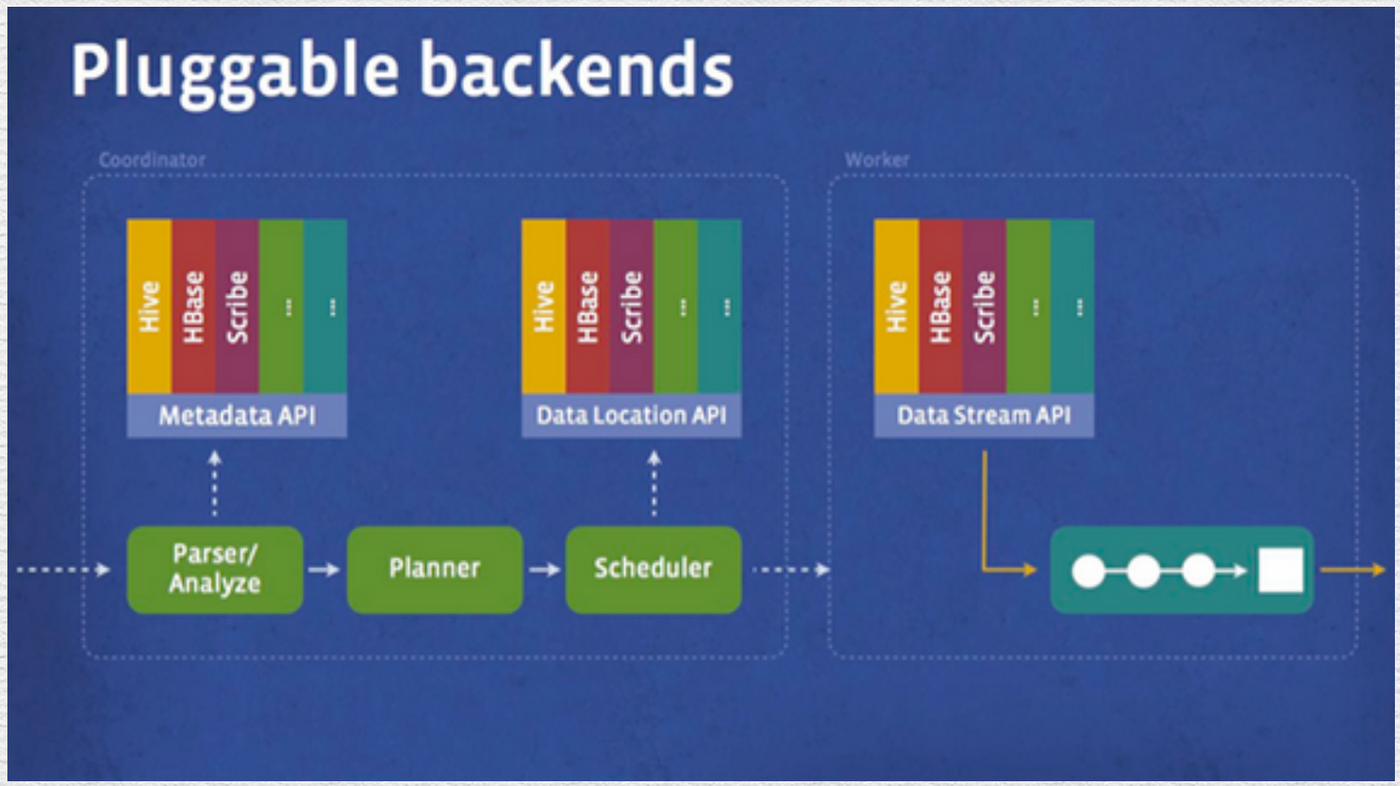



Single User versus 10 Users Response Time/Impala Times Faster Than
(Lower bars are better)
Time (in Seconds)
350
300
250
200
150
100
50
0
Single User, 5
10 User, 11
Single User, 25
10 User, 120
10.6x
5.0x
Single User, 37
10 User, 302
27.4x
7.4x
Single User, 77
10 User, 202
18.3x
15.4x
Impala
Spark SQL
Presto
Hive-on-Tez

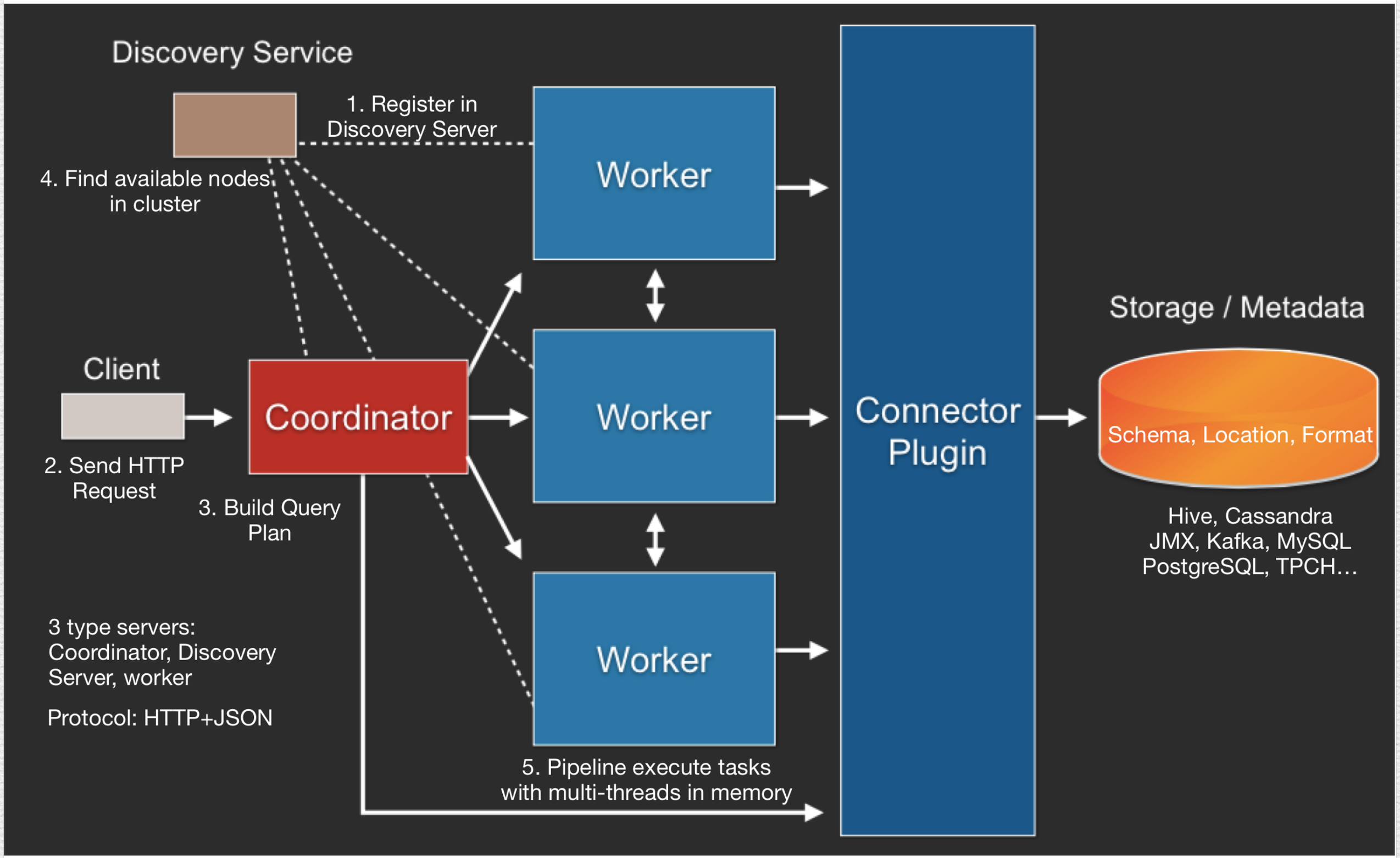
Architecture



Pluggable backends



Architecture

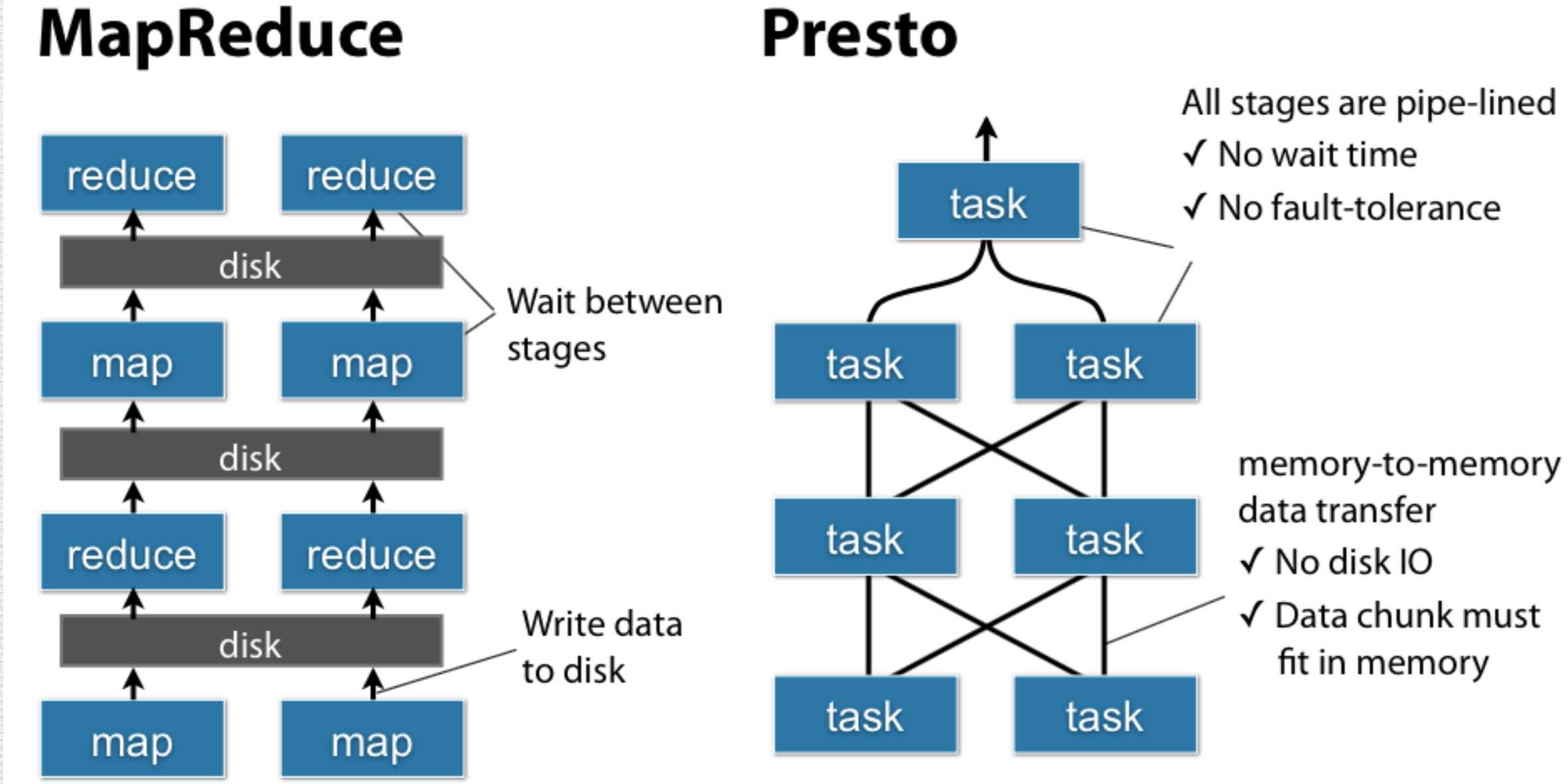


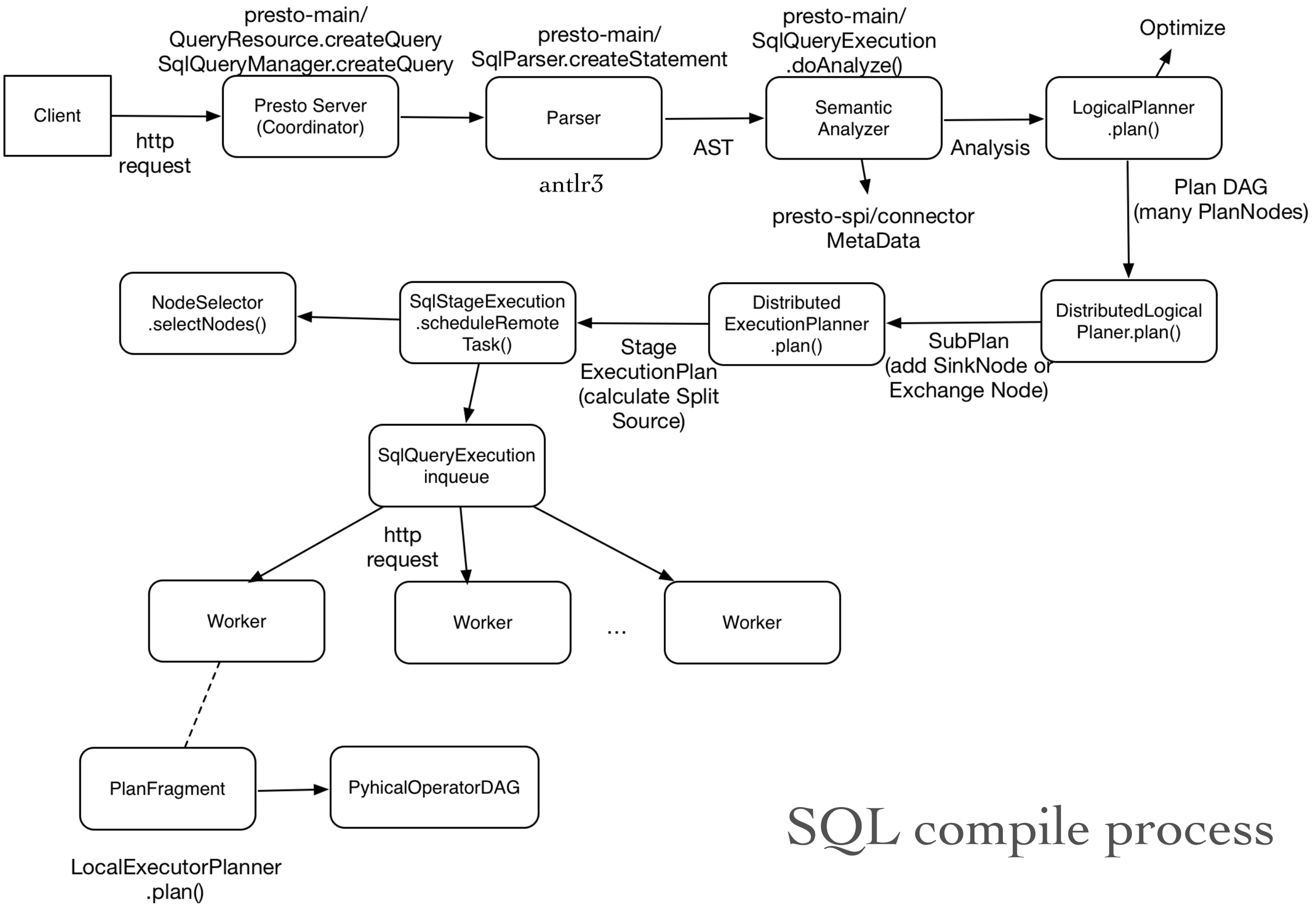
Why Presto Fast?

1. In memory parallel computing
2. Pipeline task execution
3. Data local computation with multi-threads
4. Cache hot queries and data
5. JIT compile operator to byte code
6. SQL optimization
7. Other optimization

1. In memory parallel computing

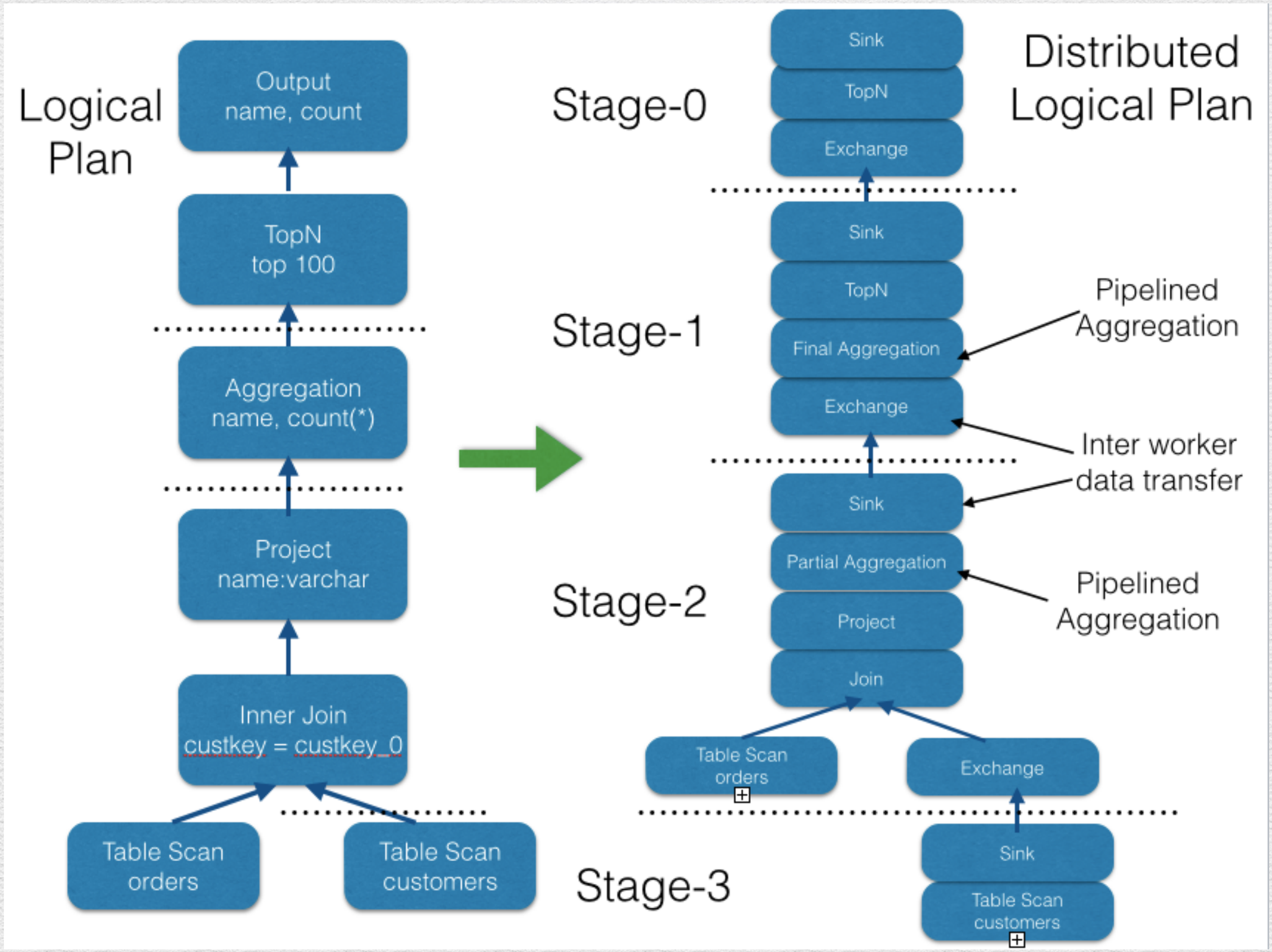
- Custom query engine, not MapReduce





SQL compile process

- select name, count(*) as count from orders as t1 join customer as t2 on t1.custkey = t2.custkey group by name order by count desc limit 100;



Sink
TopN
Exchange
Sink
TopN
Final Aggregation
Exchange
Sink
Partial Aggregation
Project
Join
Table Scan orders
Exchange
Sink
Table Scan customers

2 workers

All tasks in parallel

Worker1

1 thread

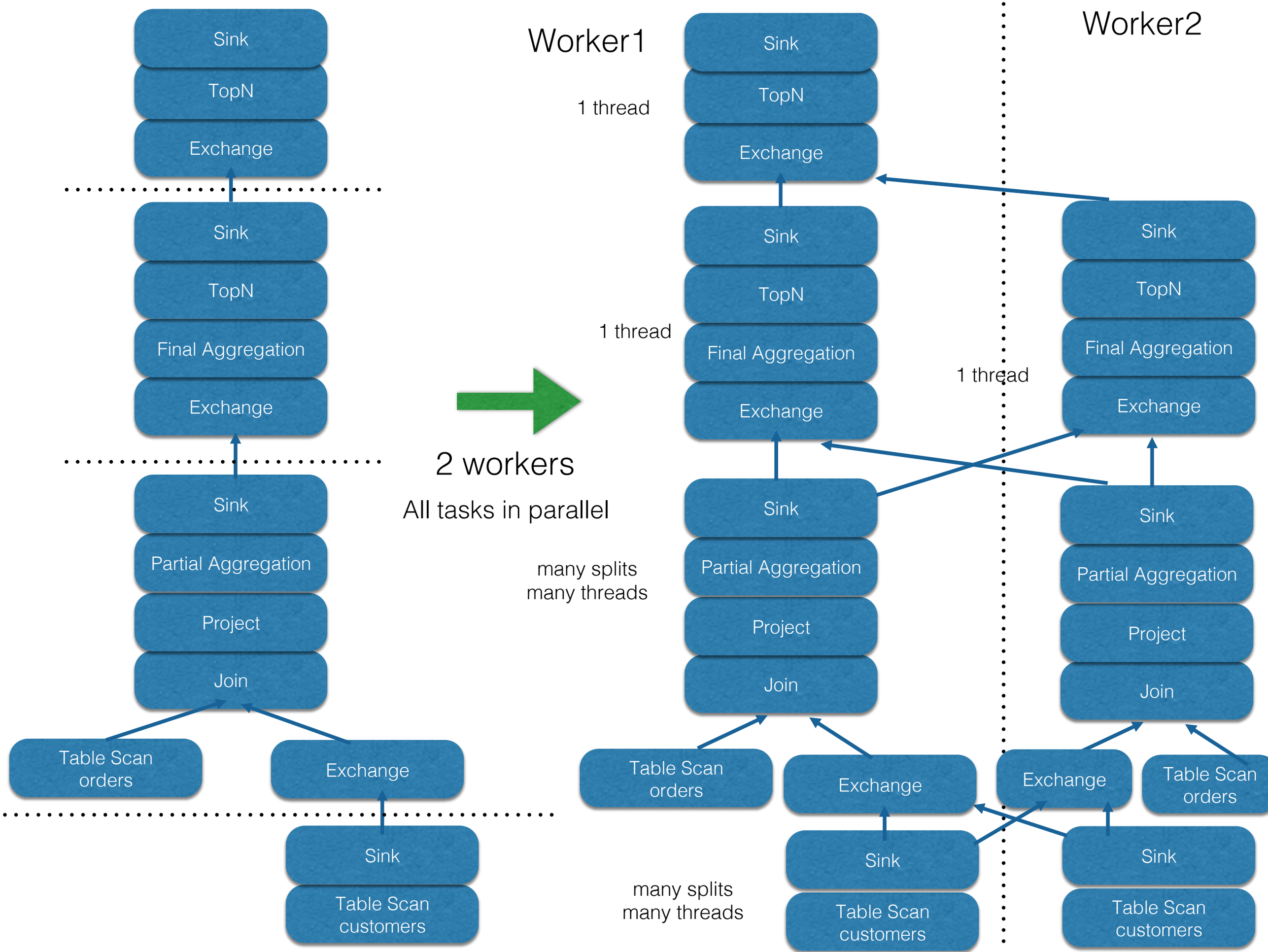
1 thread

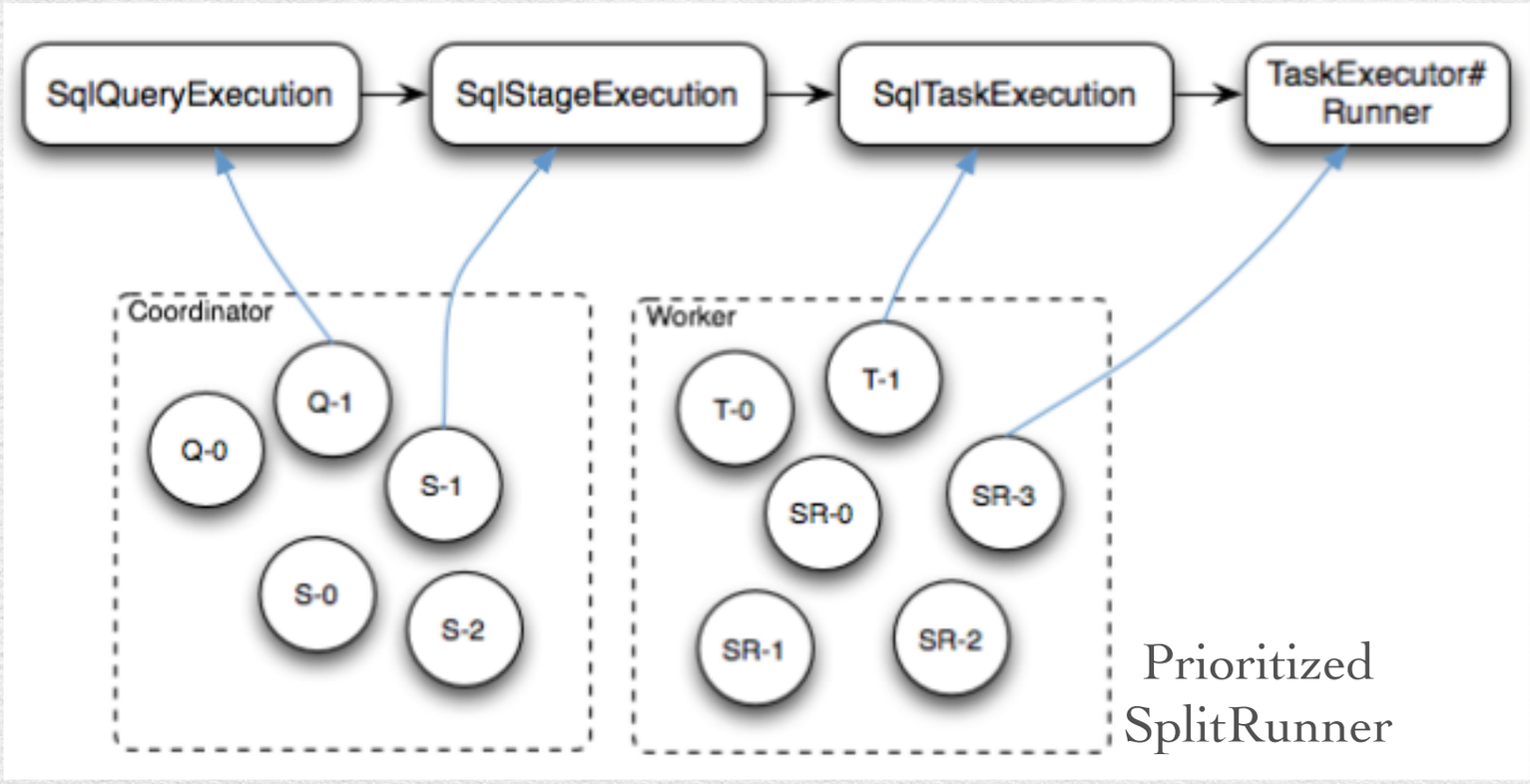
many splits
many threads

many splits
many threads

Worker2

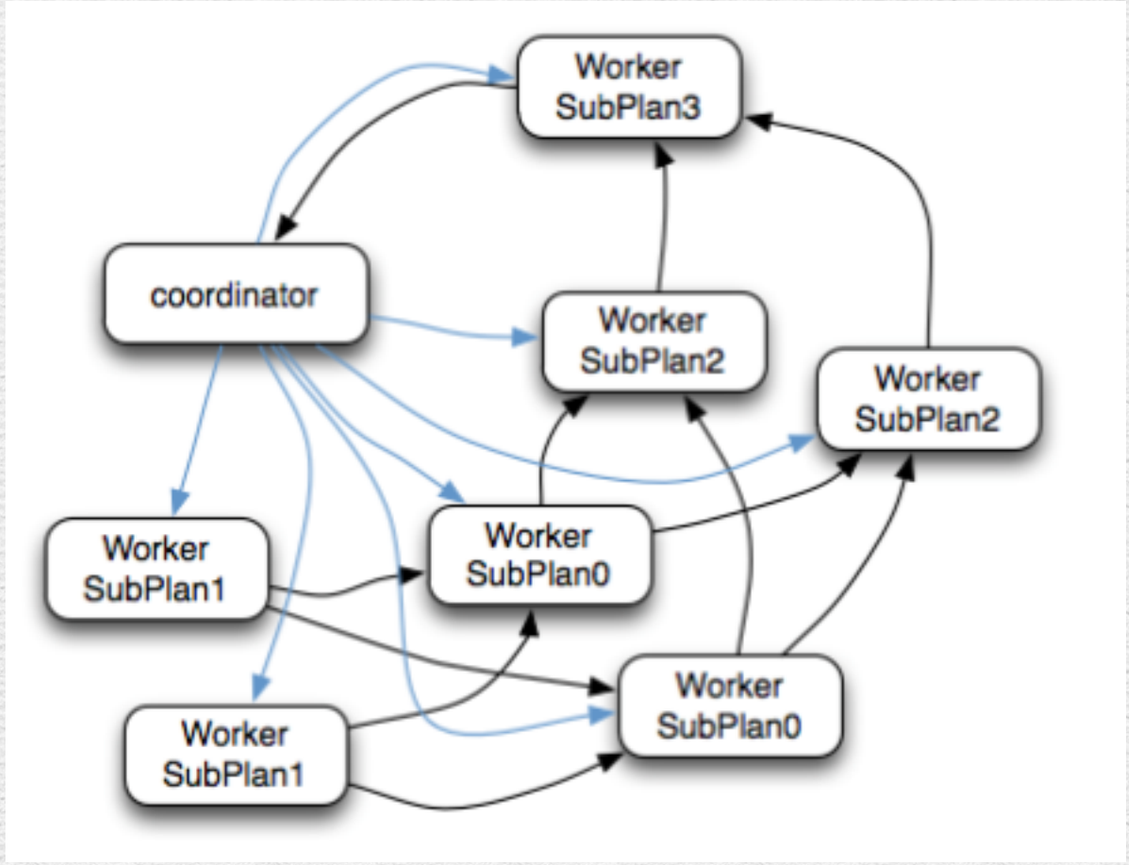
1 thread





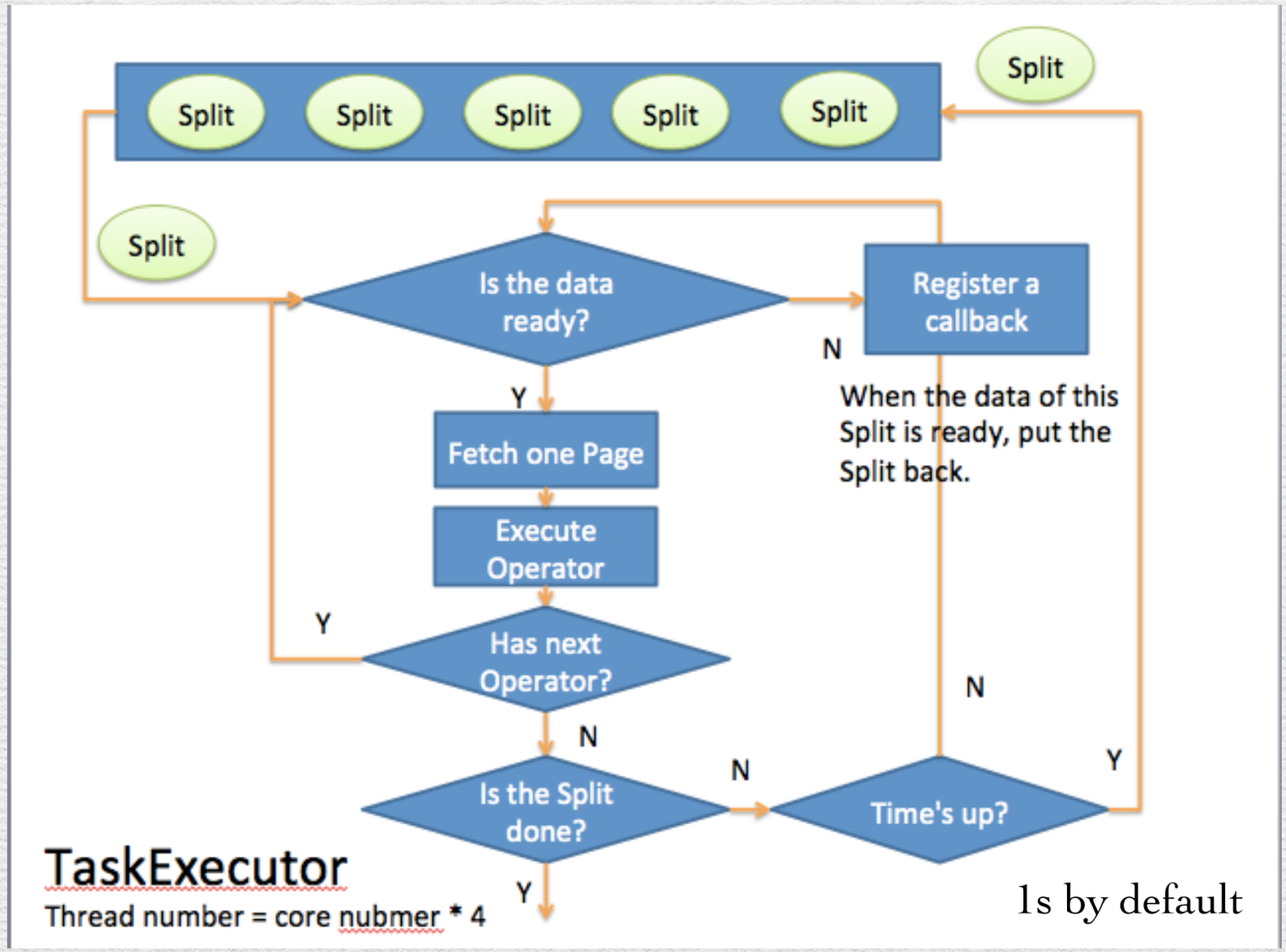
Prioritized SplitRunner

- SQL->Stages, Tasks, Splits
- One task fail, query must rerun
- Aggregation memory limit

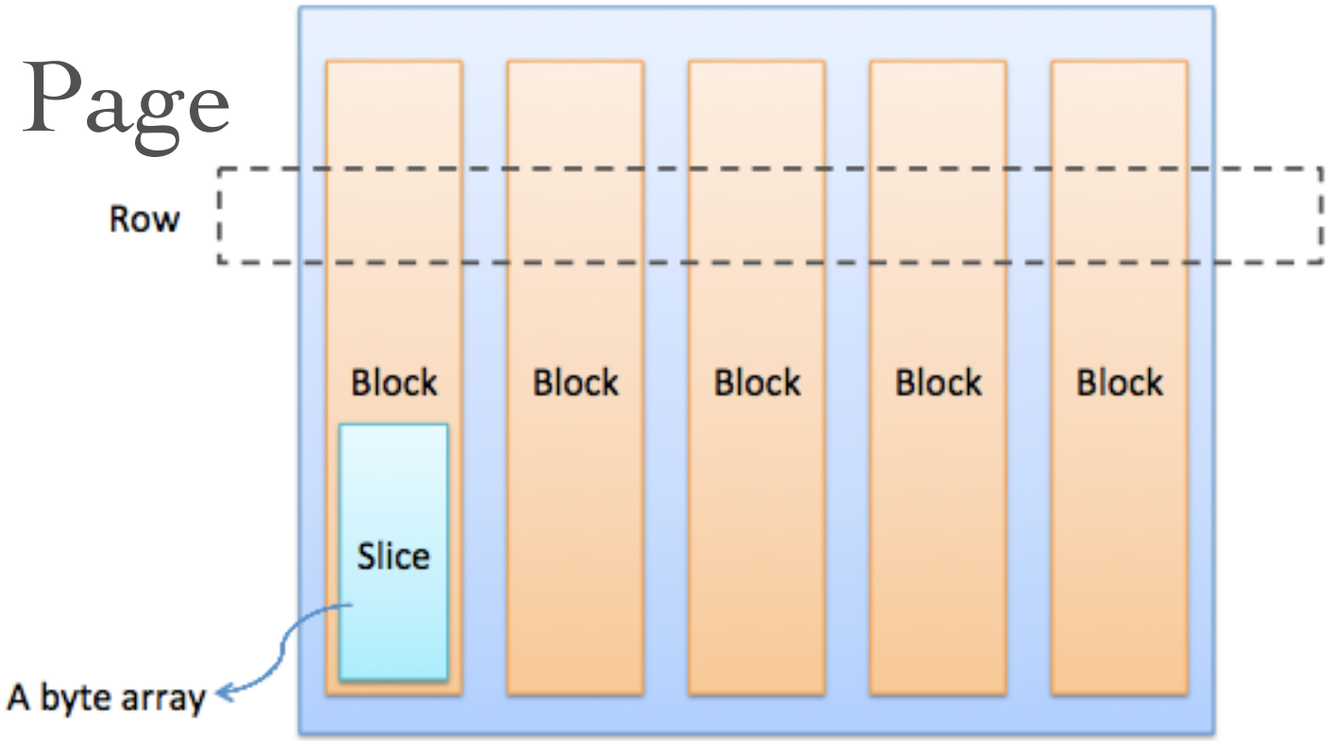


2.Pipeline task execution

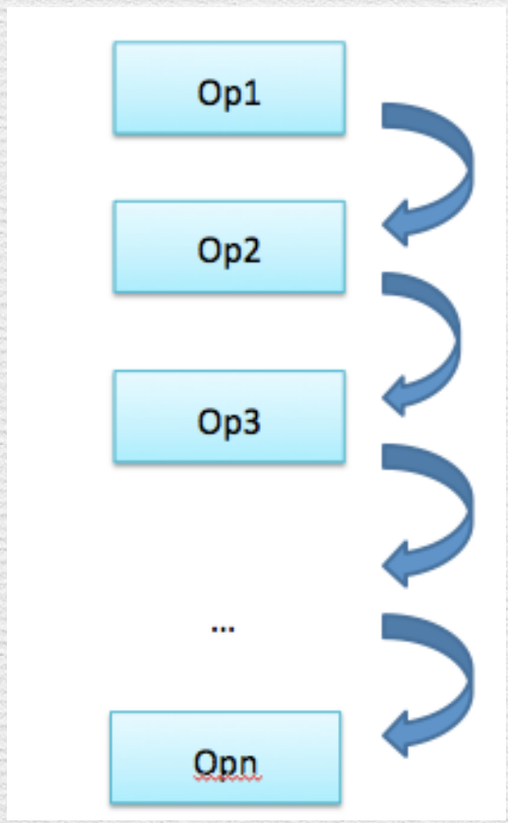
- In worker, TaskExecutor, split pipeline



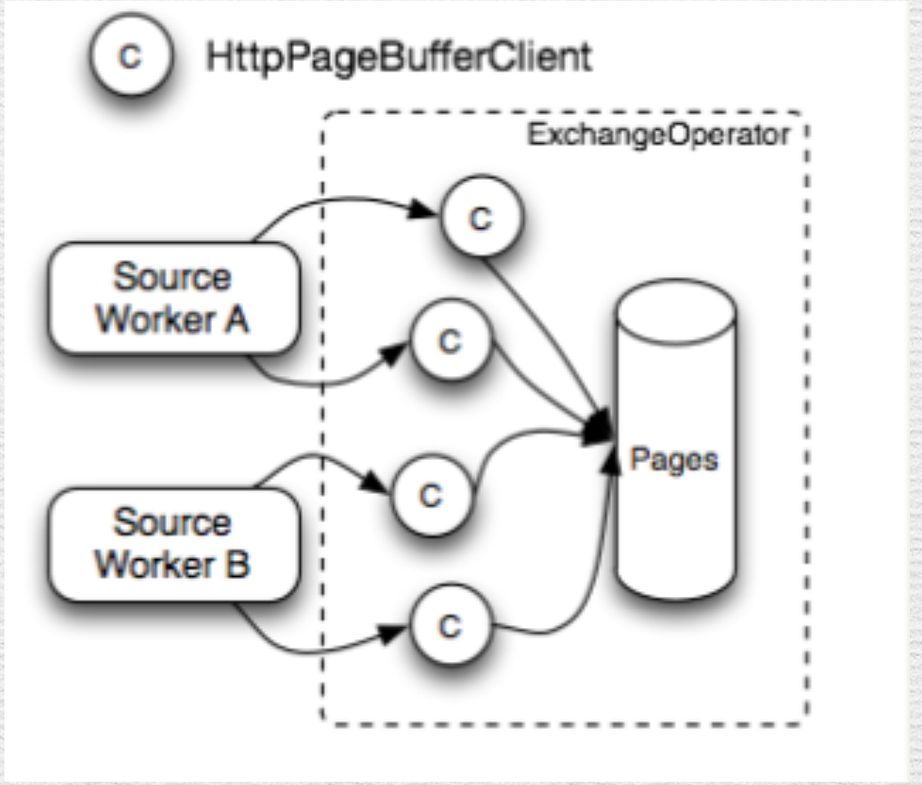
- Operator Pipeline



- Page: smallest data processing unit(like RowBatch)
- max page size 1MB, max rows: 16*1024



Exchange Operator: each client for each split



3. Data local computation with multi-threads

- NodeSelector select available nodes(10 nodes default)
  - Nodes has the same address
  - If not enough, add nodes in the same rack
  - If not enough, randomly select nodes in other racks
- Select the node with the smallest number of assignments (pending tasks)

- 4. Cache hot queries and data
  - Google Guava loading cache byte code
  - Cache Objects: Hive database/table/partition, JIT byte code class, functions
- 5. JIT compile operator to byte code
  - Compile ScanFilterAndProjectOperator , FilterAndProjectOperator

6. SQL Optimization

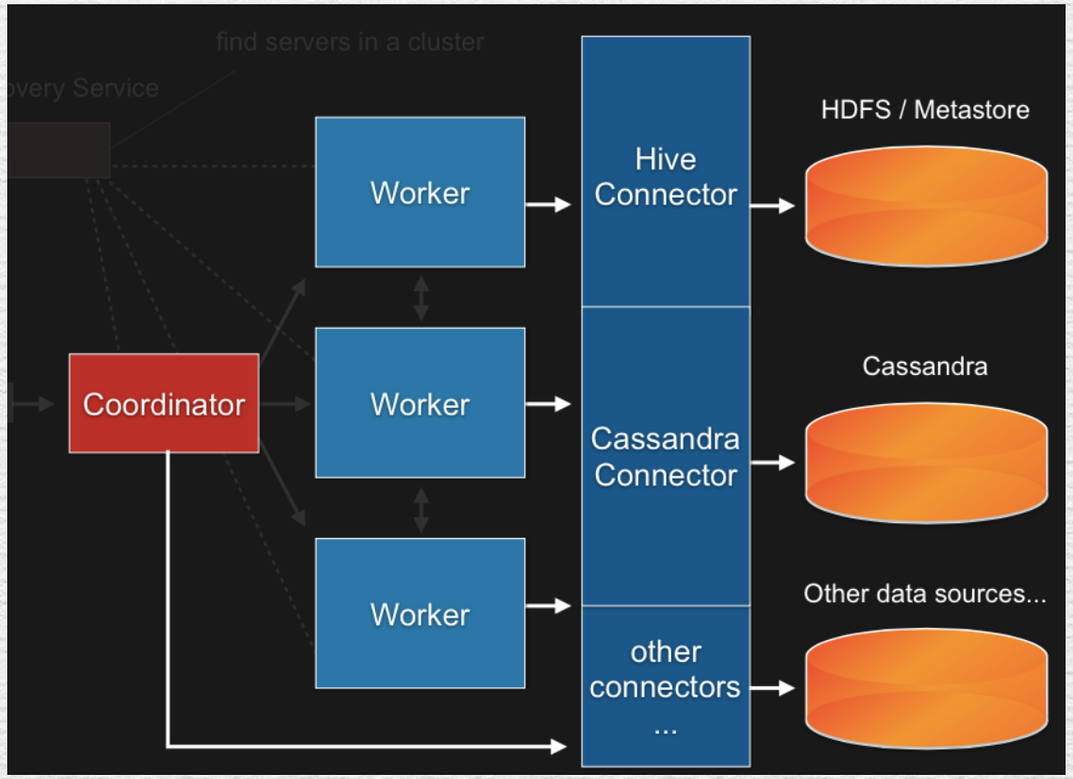
- PredicatePushDown
- PruneRedundantProjections
- PruneUnreferencedOutputs
- MergeProjections
- LimitPushDown
- CanonicalizeExpressions
- CountConstantOptimizer
- ImplementSampleAsFilter
- MetadataQueryOptimizer
- SetFlatteningOptimizer
- SimplifyExpressions
- UnaliasSymbolReferences
- WindowFilterPushDown

7. Other Optimization

- BlinkDB liked approximate queries
- JVM GC Control
  - JDK1.7
  - forcing the code cache evictor make room before the cache fills up
- Careful use mem & data structure
  - Airlift slice for efficient heap and off-heap memory(https://github.com/airlift/slice)
  - Java future async callback

Presto Extensibility

- Connectors(Catalogs): Hive, Cassandra, Hive, JMX, Kafka, MySQL, PostgreSQL, System, TPCH
- Custom connectors (http://prestodb.io/docs/current/spi/overview.html):
  - Service Provider Interface(SPI):
    - ConnectorMetadata
    - ConnectorSplitManager
    - ConnectorRecordSetProvider



Presto's Limitations

- No fault tolerance, Unstable
- Memory Limitations for aggregations, huge joins
- SQL features like:
  - only CTAS
  - no support UDF

Presto's Future

"Presto, Past, Present, and Future" by Dain Sundstrom at Facebook, 2014.May

- Basic Task Recovery
- Huge joins and Group by
- Spill to Disk(Implemented), Insert
- Create View(Implemented), not compatible with hive
- Native Store, Cache Hot data(Implemented)
- Security : Authentication, Authorization, Permissions
- ODBC Driver
- Improve DDL DML

References

- http://prestodb.io/
- https://github.com/facebook/presto
- https://www.facebook.com/notes/facebook-engineering/presto-interacting-with-petabytes-of-data-at-facebook/10151786197628920