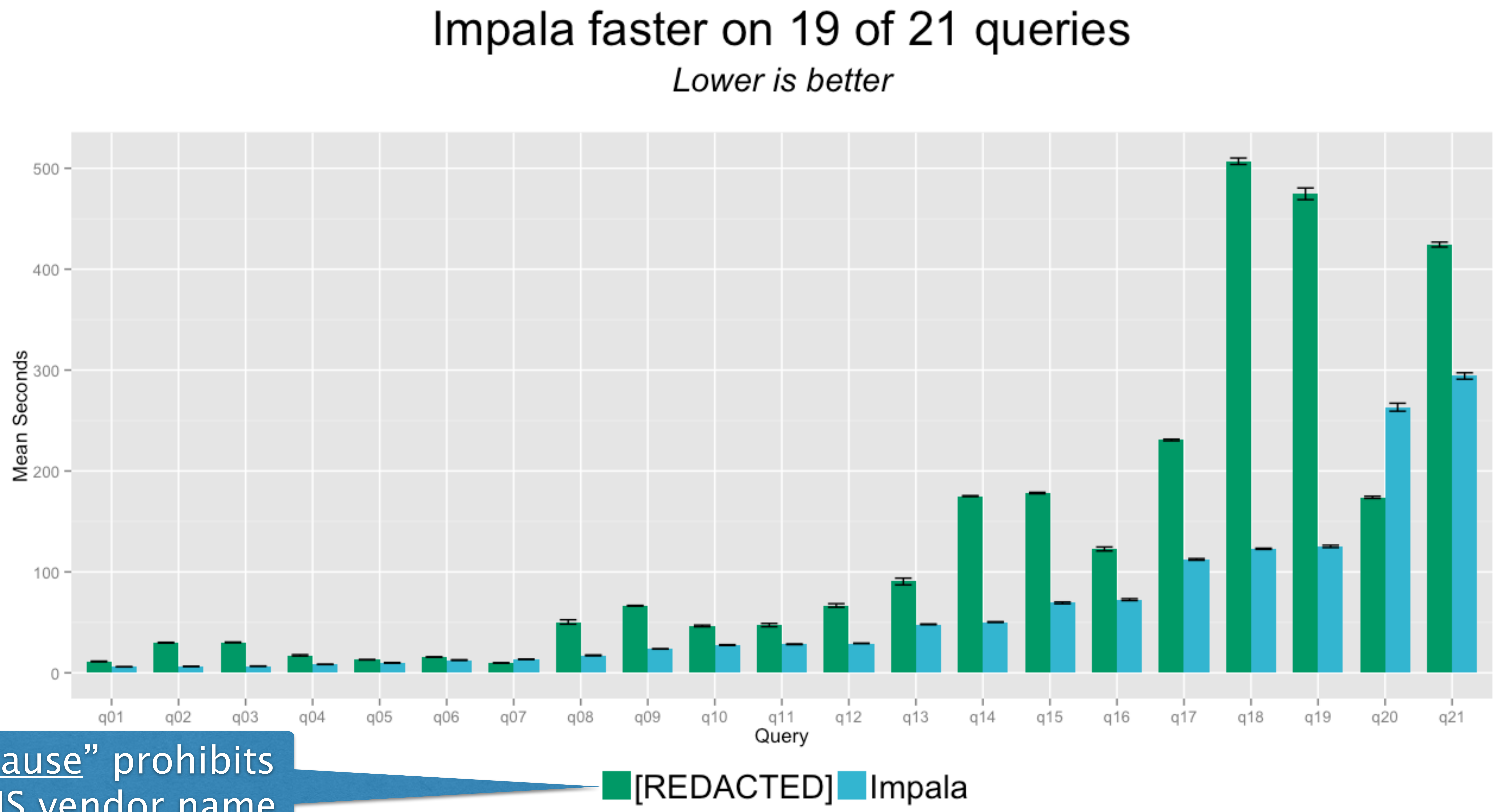
The Next Wave of SQL-on-Hadoop: The Hadoop Data Warehouse

Marcel Kornacker | marcel@cloudera.com

QCon 2014

Analytic Workloads on Hadoop: Where Do We Stand?



"DeWitt Clause" prohibits using DBMS vendor name

Hadoop for Analytic Workloads

- Hadoop has traditional been utilized for offline batch processing: ETL and ELT
- Next step: Hadoop for traditional business intelligence (BI)/data warehouse (EDW) workloads:
  - interactive
  - concurrent users
- Topic of this talk: a Hadoop-based open-source stack for EDW workloads:
  - HDFS: a high-performance storage system
  - Parquet: a state-of-the-art columnar storage format
  - Impala: a modern, open-source SQL engine for Hadoop

Hadoop for Analytic Workloads

- Thesis of this talk:
  - techniques and functionality of established commercial solutions are either already available or are rapidly being implemented in Hadoop stack
  - Hadoop stack is effective solution for certain EDW workloads
  - Hadoop-based EDW solution maintains Hadoop's strengths: flexibility, ease of scaling, cost effectiveness

HDFS: A Storage System for Analytic Workloads

- Available in Hdfs today:
  - high-efficiency data scans at or near hardware speed, both from disk and memory
- On the immediate roadmap:
  - co-partitioned tables for even faster distributed joins
  - temp-FS: write temp table data straight to memory, bypassing disk

HDFS: The Details

- High efficiency data transfers
  - short-circuit reads: bypass DataNode protocol when reading from local disk
    -> read at 100+MB/s per disk
  - HDFS caching: access explicitly cached data w/o copy or checksumming
    -> access memory-resident data at memory bus speed
    -> enable in-memory processing

HDFS: The Details

- Coming attractions:
  - affinity groups: collocate blocks from different files
    -> create co-partitioned tables for improved join performance
  - temp-fs: write temp table data straight to memory, bypassing disk
    -> ideal for iterative interactive data analysis

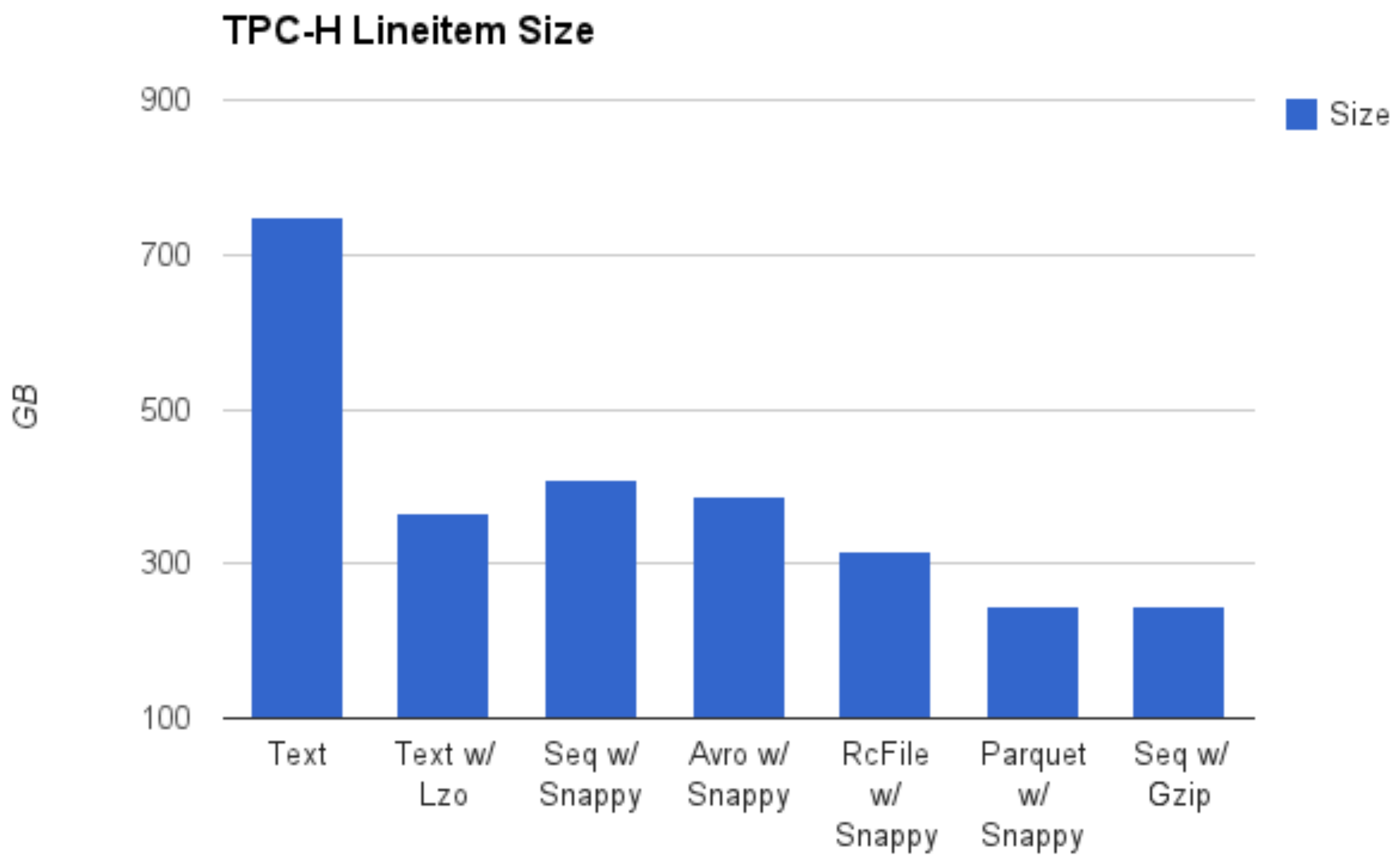
Parquet: Columnar Storage for Hadoop

- What it is:
  - state-of-the-art, open-source columnar file format that's available for (most) Hadoop processing frameworks: Impala, Hive, Pig, MapReduce, Cascading, …
  - offers both high compression and high scan efficiency
  - co-developed by Twitter and Cloudera; hosted on github and soon to be an Apache incubator project
  - with contributors from Criteo, Stripe, Berkeley AMPlab, LinkedIn
  - used in production at Twitter and Criteo

Parquet: The Details

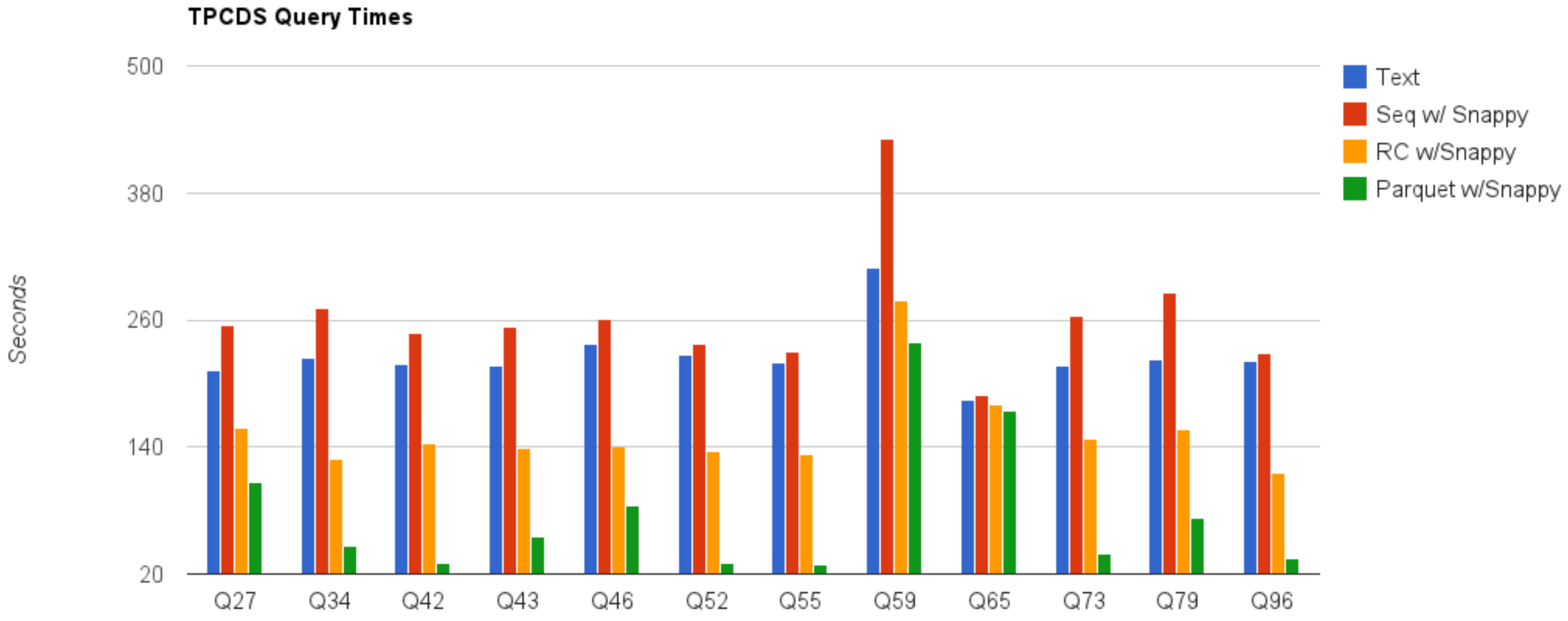
- columnar storage: column-major instead of the traditional row-major layout; used by all high-end analytic DBMSs
- optimized storage of nested data structures: patterned after Dremel's ColumnIO format
- extensible set of column encodings:
  - run-length and dictionary encodings in current version (1.2)
  - delta and optimized string encodings in 2.0
- embedded statistics: version 2.0 stores inlined column statistics for further optimization of scan efficiency

Parquet: Storage Efficiency



cloudera Ask Bigger Questions

Parquet: Scan Efficiency



Impala: A Modern, Open-Source SQL Engine

- implementation of an MPP SQL query engine for the Hadoop environment
- designed for performance: brand-new engine, written in C++
- maintains Hadoop flexibility by utilizing standard Hadoop components (HDFS, Hbase, Metastore, Yarn)
- plays well with traditional BI tools: exposes/interacts with industry-standard interfaces (odbc/ jdbc, Kerberos and LDAP, ANSI SQL)

Impala: A Modern, Open-Source SQL Engine

- history:
  - developed by Cloudera and fully open-source; hosted on github
  - released as beta in 10/2012
  - 1.0 version available in 05/2013
  - current version is 1.3.1, available for CDH4 and CDH5

Impala from The User's Perspective

- create tables as virtual views over data stored in HDFS or Hbase;
  schema metadata is stored in Metastore (shared with Hive, Pig, etc.; basis of HCatalog)
- connect via odbc/jdbc; authenticate via Kerberos or LDAP
- run standard SQL:
  - current version: ANSI SQL-92 (limited to SELECT and bulk insert) minus correlated subqueries, has UDFs and UDAs

New Features in 1.3/1.4

- Admission control: workload management in a distributed environment
  - enforce global limits on # of concurrently executing queries and/or memory consumption
  - admin configures pools with limits and assigns users to pools
  - decentralized: avoids single-node bottlenecks for low-latency, high-throughput scheduling

New Features in 1.3/1.4

- HDFS caching: zero-overhead access to memory-resident data
  - avoids checksumming and data copies
  - Impala contains fast I/O path to take maximum advantage of HDFS caching via new HDFS API (since CDH5.0)
  - new in 1.4: configure cache with Impala DDL:
    - CREATE TABLE T … CACHED IN ‘<pool>’
    - ALTER TABLE T ADD PARTITION(…) CACHED IN ‘<pool>’

New Features in 1.3/1.4

- Support for fixed-point arithmetic: DECIMAL(<precision>, <scale>)
  - up to precision of 38 digits
  - optimized implementation that maps to fixed-size integer arithmetic
  - no performance degradation compared to FLOAT/DOUBLE

New Features in 1.3/1.4

- … and of course performance improvements
  - COMPUTE STATS got ~5x faster
  - Bloom filters for broadcast joins

Roadmap: Impala 2.0

- Analytic [window] functions
  - example: SUM(revenue) OVER (PARTITION BY … ORDER BY …)
- Subqueries:
  - present: inline views with arbitrary levels of nesting
  - in 2.0: correlated and uncorrelated subqueries
- Joins and aggregation can spill to disk:
  - present: joins and aggregation are hash based; hash table needs to fit in memory
  - in 2.0: hash table can spill to disk; join and aggregate tables of arbitrary size

cloudera
Ask Bigger Questions

Roadmap: Impala 2.0

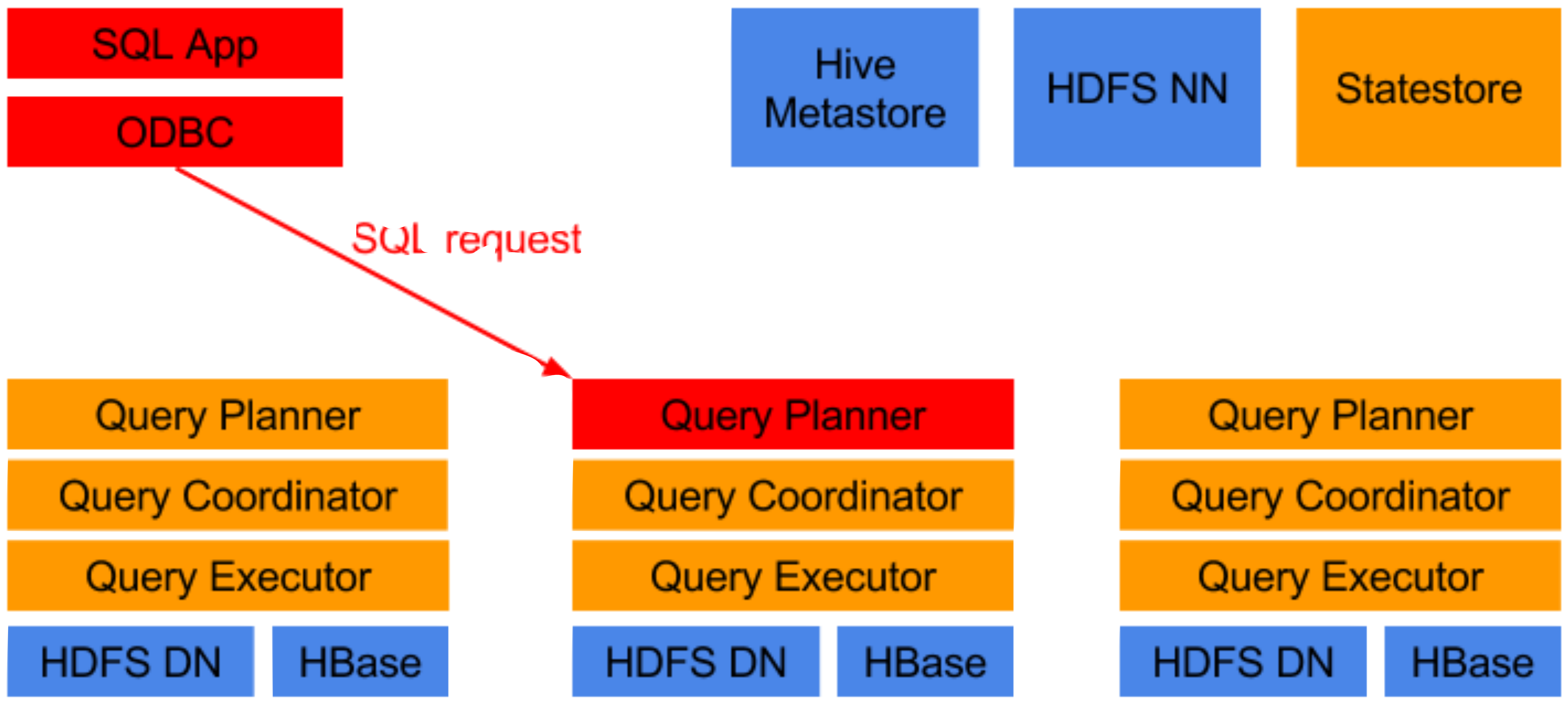
- Nested data structures: structs, arrays, maps in Parquet, Avro, json, …
  - natural extension of SQL: expose nested structures as tables
  - no limitation on nesting levels or number of nested fields in single query

Impala Architecture

- distributed service:
  - daemon process (impalad) runs on every node with data
  - easily deployed with Cloudera Manager
  - each node can handle user requests; load balancer configuration for multi-user environments recommended
- query execution phases:
  - client request arrives via odbc/jdbc
  - planner turns request into collection of plan fragments
  - coordinator initiates execution on remote impala's

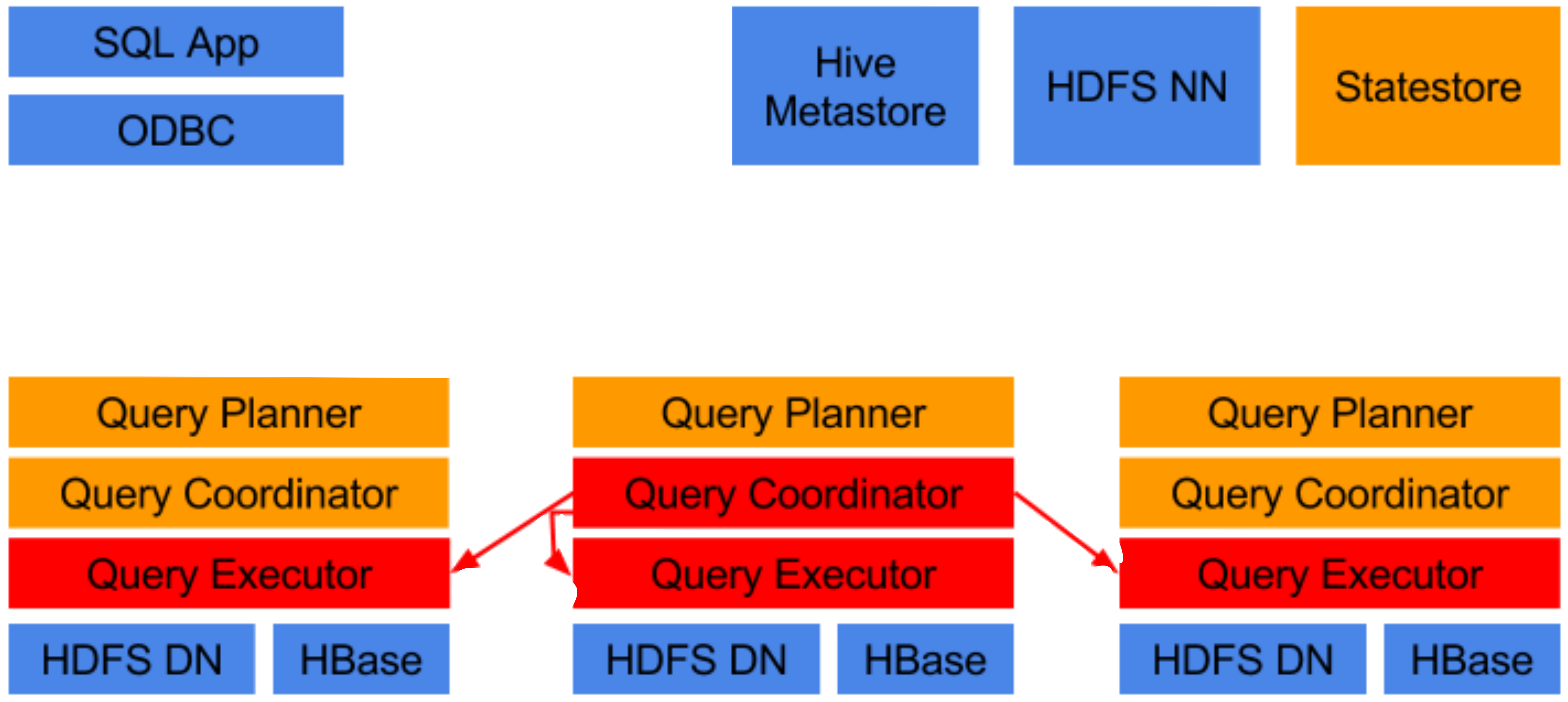
Impala Query Execution

- Request arrives via odbc/jdbc



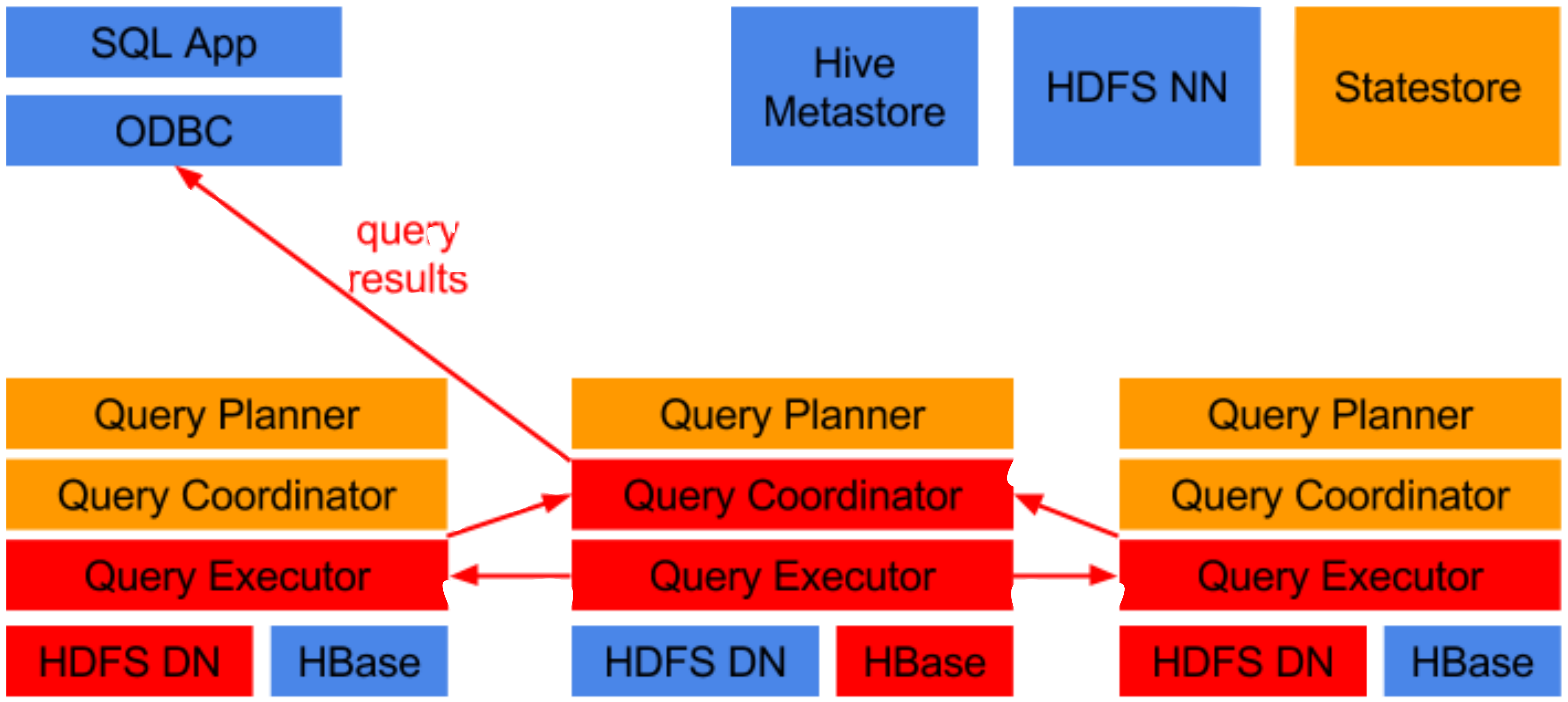
Impala Query Execution

- Planner turns request into collection of plan fragments
- Coordinator initiates execution on remote impalad nodes



Impala Query Execution

- Intermediate results are streamed between impala's
- Query results are streamed back to client



Impala Architecture: Query Planning

- 2-phase process:
  - single-node plan: left-deep tree of query operators
  - partitioning into plan fragments for distributed parallel execution:
    maximize scan locality/minimize data movement, parallelize all query operators
- cost-based join order optimization
- cost-based join distribution optimization

Impala Architecture: Query Execution

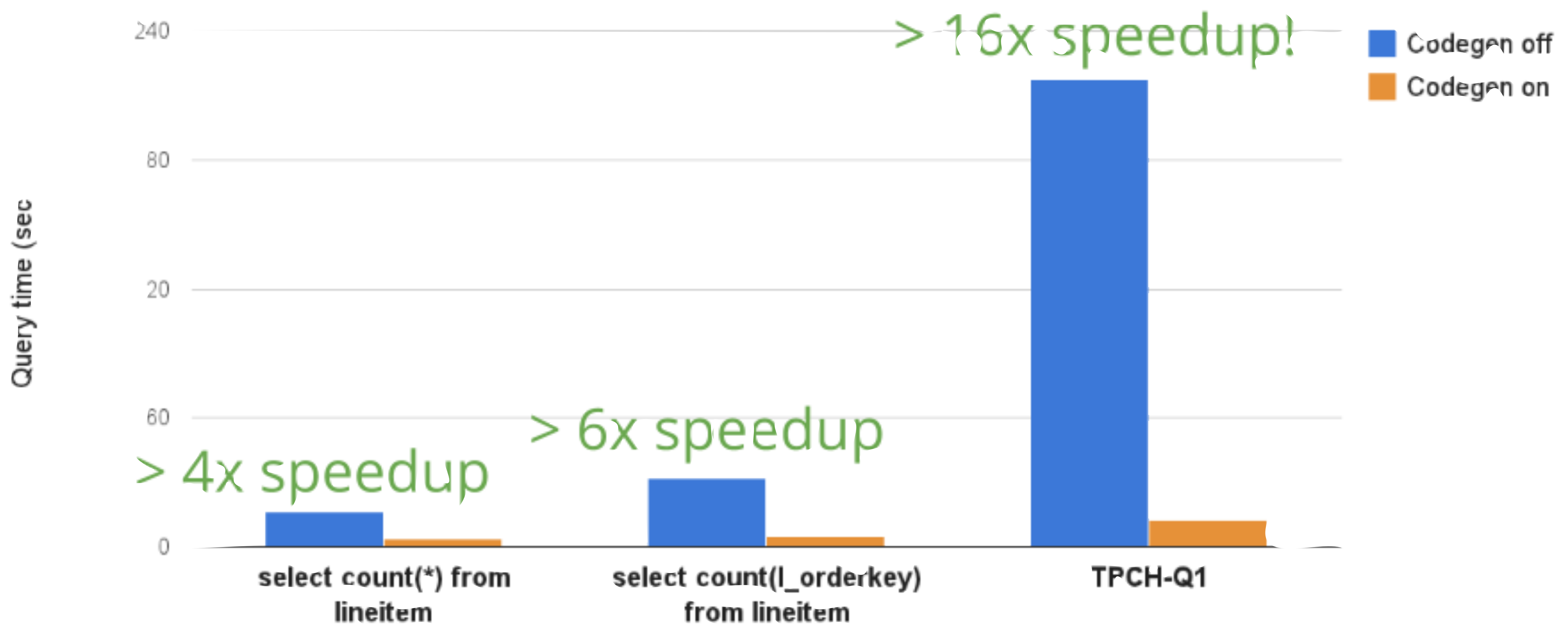
- execution engine designed for efficiency, written from scratch in C++; no reuse of decades-old open-source code
- circumvents MapReduce completely
- in-memory execution:
  - aggregation results and right-hand side inputs of joins are cached in memory
  - example: join with 1TB table, reference 2 of 200 cols, 10% of rows
    -> need to cache 1GB **across all nodes** in cluster
    -> not a limitation for most workloads

Impala Architecture: Query Execution

- runtime code generation:
  - uses llvm to jit-compile the runtime-intensive parts of a query
  - effect the same as custom-coding a query:
    - remove branches
    - propagate constants, offsets, pointers, etc.
    - inline function calls
  - optimized execution for modern CPUs (instruction pipelines)

Impala Architecture: Query Execution

Results

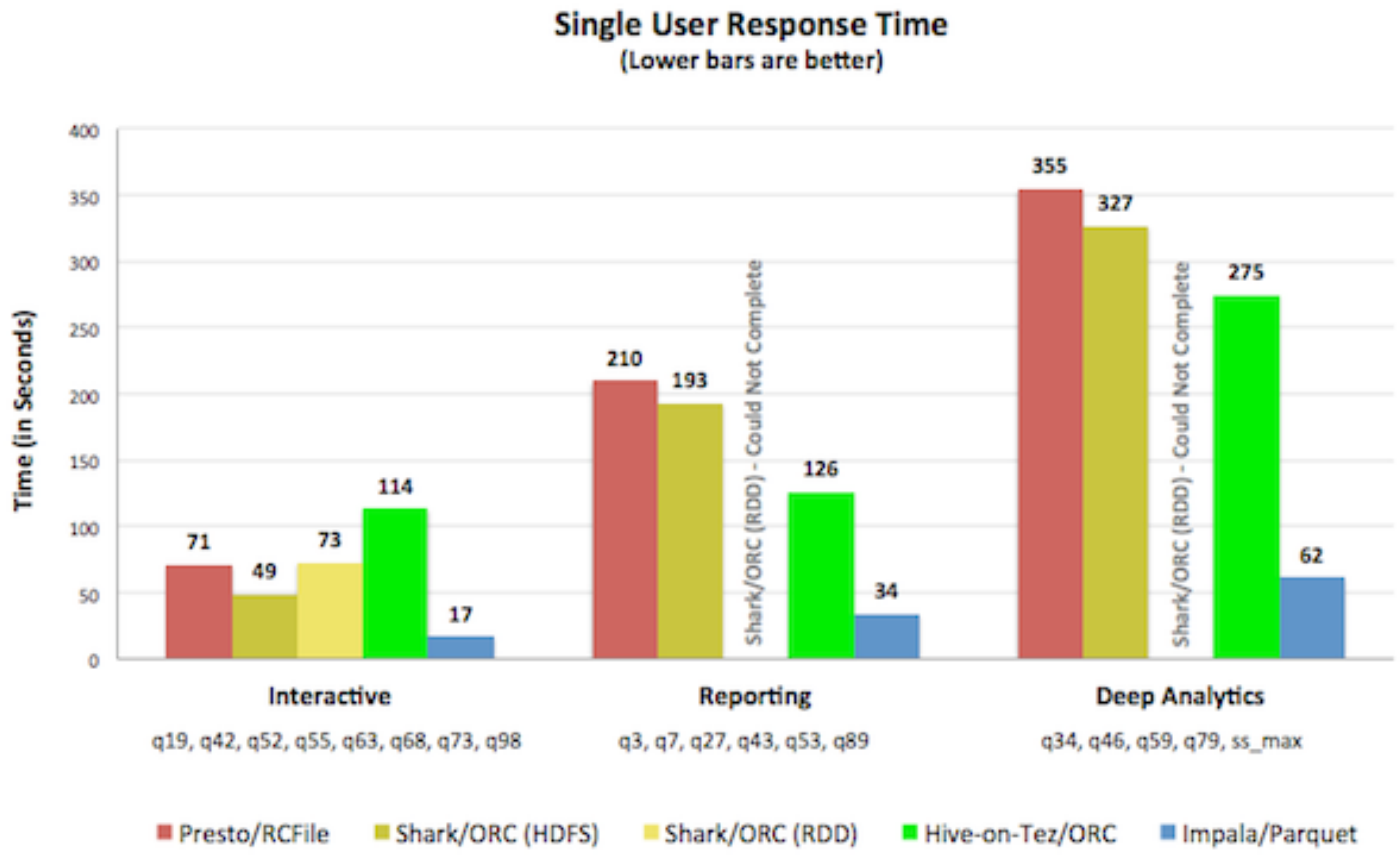


10 node cluster (12 disks / 48GB RAM / 8 cores per node)
40 GB / 60M row Avro dataset

Impala Performance

- benchmark: TPC-DS
  - subset of queries (21 queries)
  - 15TB scale factor data set
  - on 21-node cluster

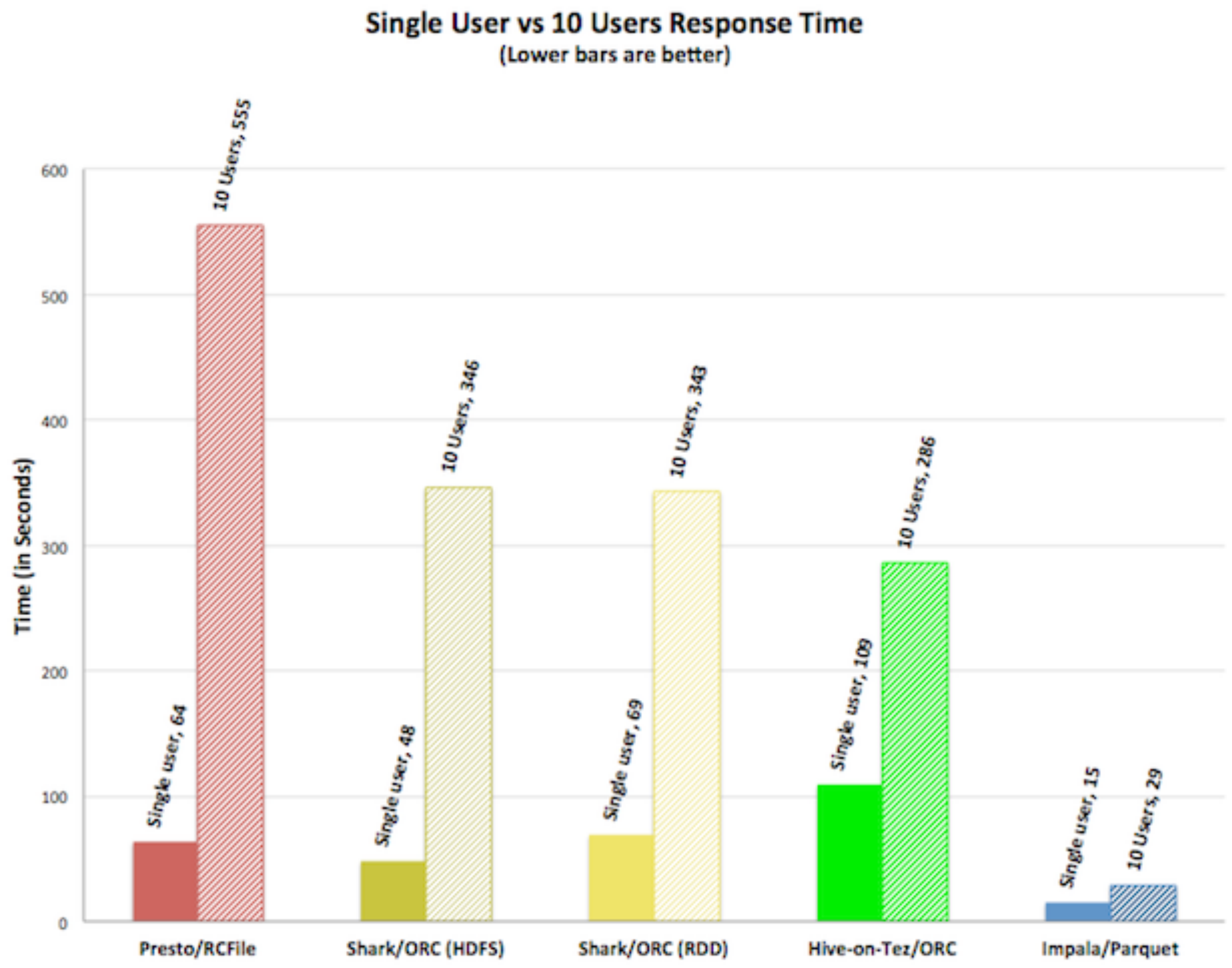
Impala Performance: Single-User



- single-user execution
- group queries by how much data they access:
  - interactive
  - reporting
  - deep analytic

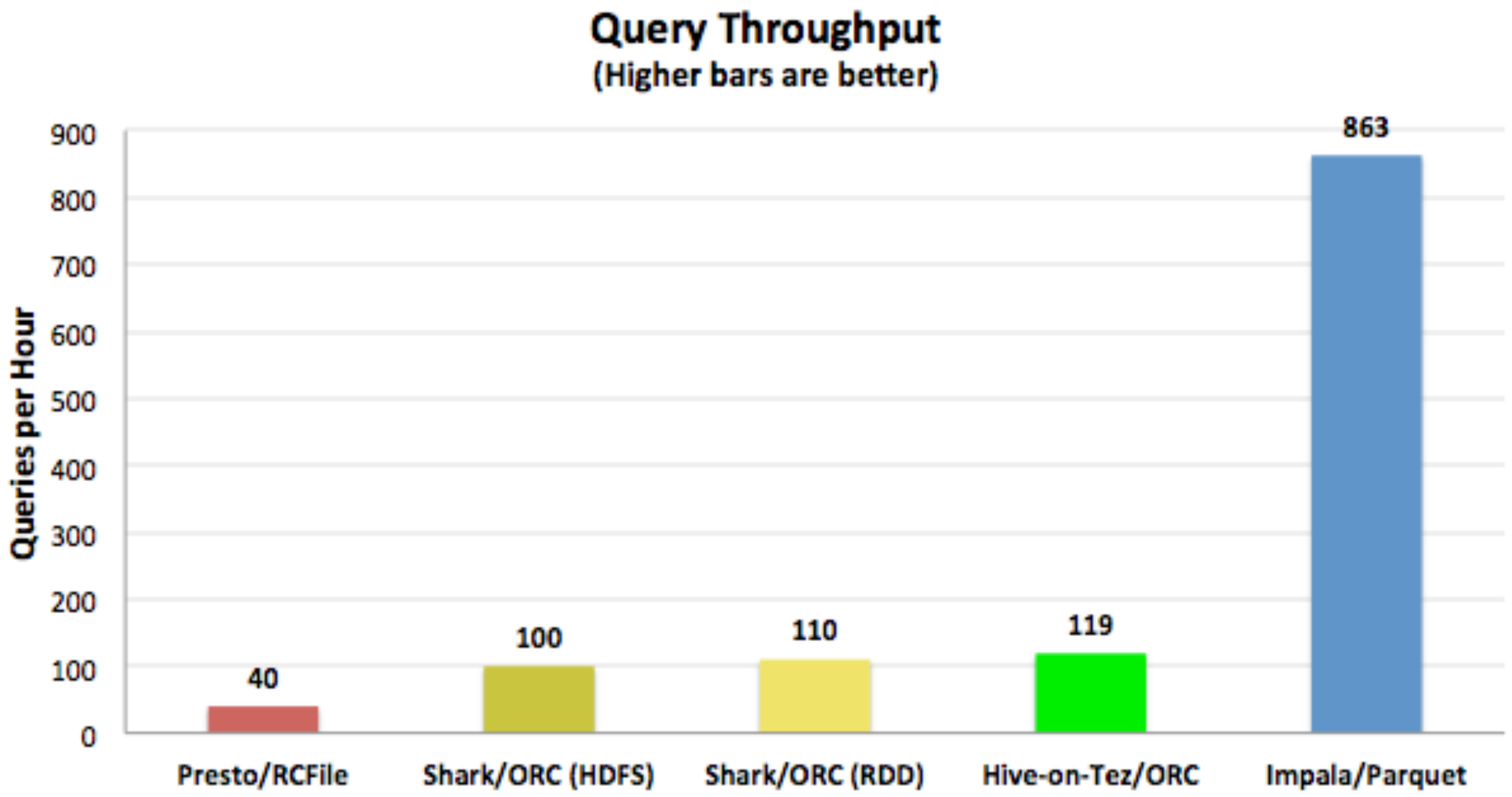
cloudera
Ask Bigger Questions

Impala Performance: Multi-User

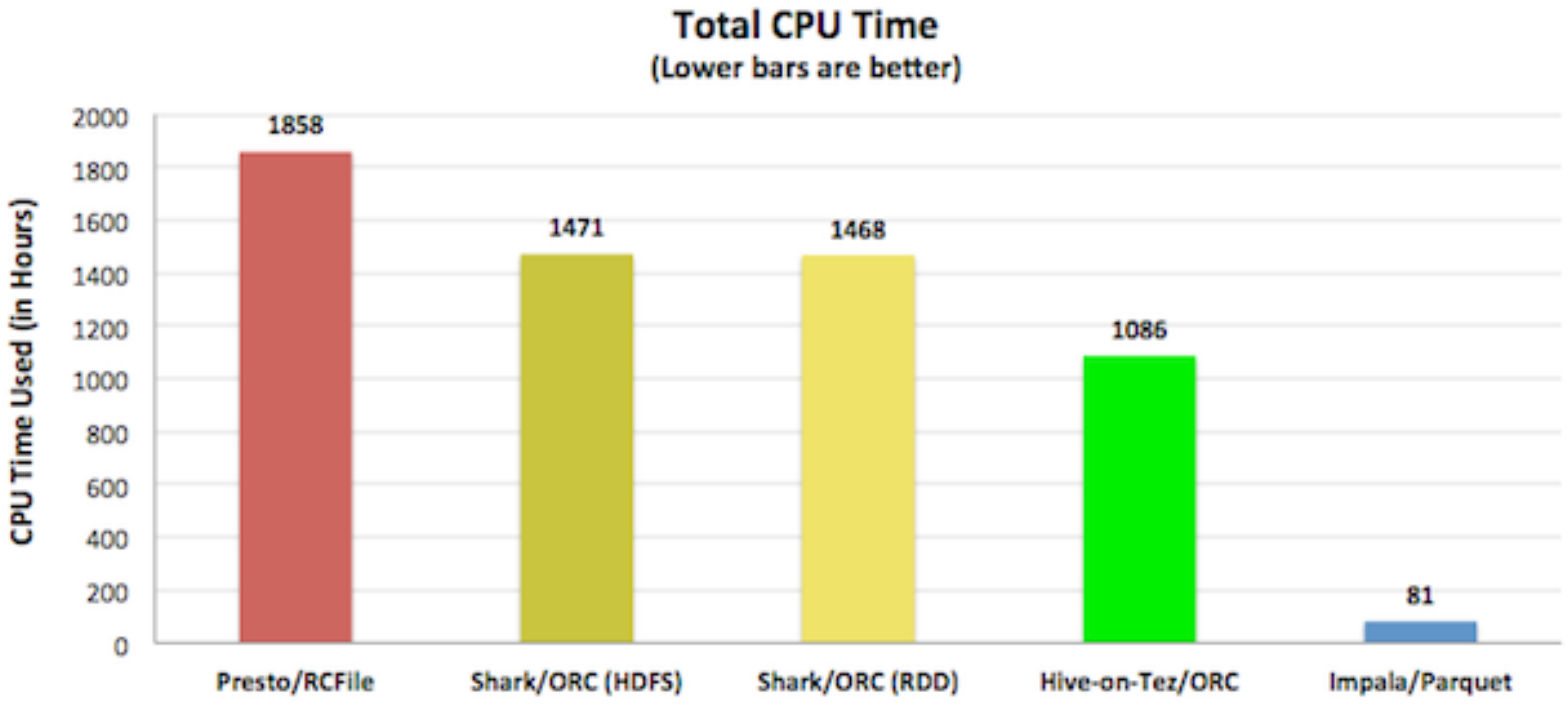


- 10 concurrent queries
- from the interactive bucket

Impala Performance: Multi-User



Impala Performance: Multi-User



Use Case: Global Payments Processor

- Application: fraud detection
- CDH/Impala replaced DB2+SAN
- 52-node cluster, 300TB total
- Daily ingest: 2TB (expanding to 4TB)
- Overall $30M in annual savings
- Storage cost went from $7K/TB to $0.5K/TB

Use Case: Media Math

- Leading Demand-Side Platform, billions of ad impressions per day
- Application: data aggregation in 30-minute intervals for reporting
- CDH/Impala replaced Netezza
- Evaluated Hive/Pig: 6 hour latency
- Impala brought this down to 30 minutes
- See blog post http://developer.mediamath.com/blog/how-hadoop-impala-helped-solve-mediamaths-critical-reporting-problem/

Use Case: Allstate Insurance

- Application: interactive data exploration with Tableau
- Test case: 1.1B rows, 800 columns, 2.3TB as CSV on 35-node cluster
- Select count(distinct …) group by …: 120 seconds
- The same with parquet: <9 seconds
- See blog post http://blog.cloudera.com/blog/2014/05/using-impala-at-scale-at-allstate/

Summary: Hadoop for Analytic Workloads

- Techniques and functionality of established commercial solutions are either already available or are rapidly being implemented in Hadoop stack
- Impala/Parquet/Hdfs is effective solution for certain EDW workloads
- Hadoop-based EDW solution maintains Hadoop's strengths: flexibility, ease of scaling, cost effectiveness

The End