Hortonworks Premier Inside-out Apache Druid

Slim Bouguerra, Staff Software Engineer
Will Xu, Senior Product Manager

May, 2018

Disclaimer

This document may contain product features and technology directions that are under development, may be under development in the future or may ultimately not be developed.

Project capabilities are based on information that is publicly available within the Apache Software Foundation project websites ("Apache"). Progress of the project capabilities can be tracked from inception to release through Apache, however, technical feasibility, market demand, user feedback and the overarching Apache Software Foundation community development process can all affect timing and final delivery.

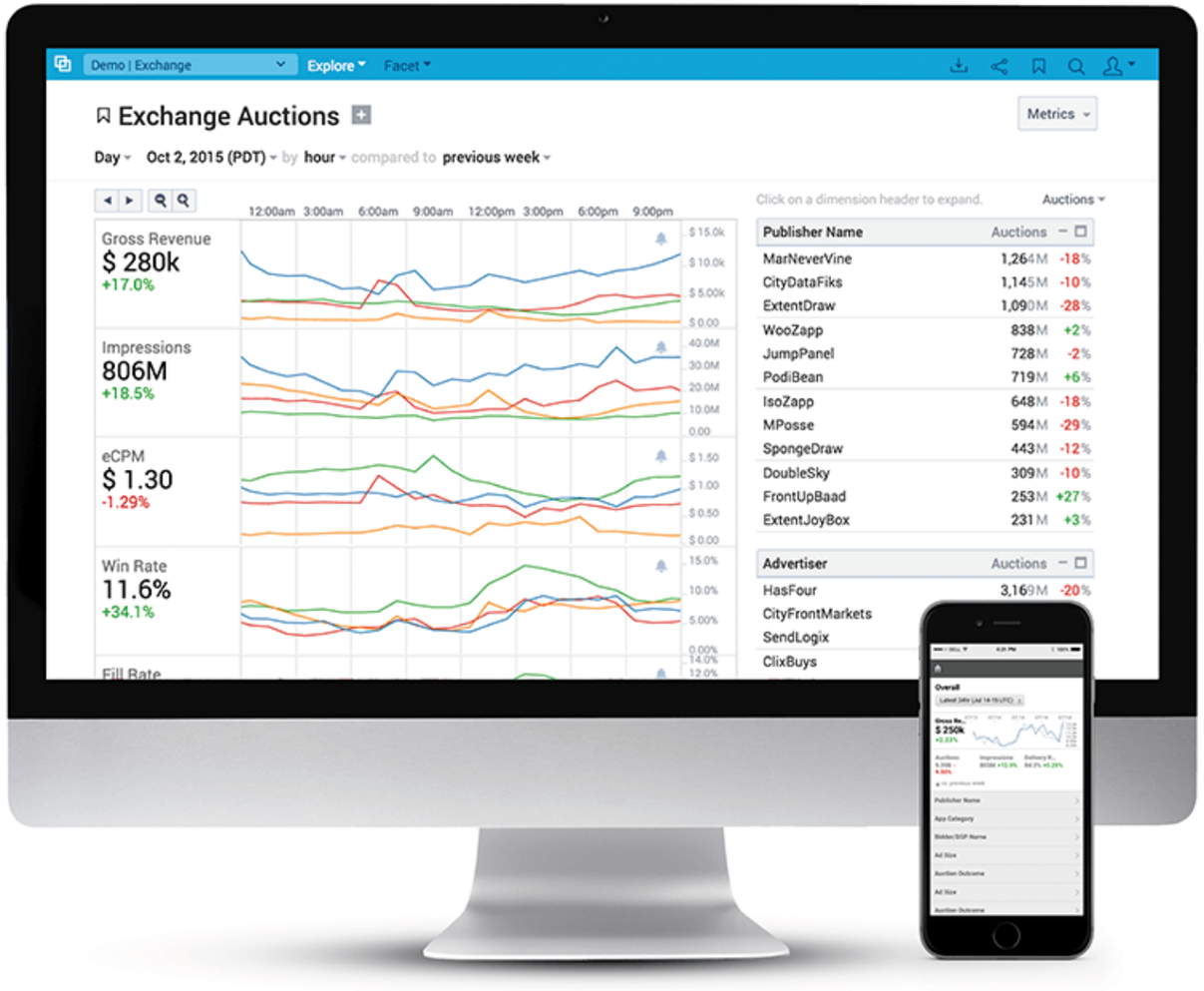
This document's description of these features and technology directions does not represent a contractual commitment, promise or obligation from Hortonworks to deliver these features in any generally available product.

Product features and technology directions are subject to change, and must not be included in contracts, purchase orders, or sales agreements of any kind.

Since this document contains an outline of general product development plans, customers should not rely upon it when making purchasing decisions.

History

- Development started at Metamarkets in 2011
- Initial use case
  - power ad-tech analytics product
- Open sourced in late 2012
  - GPL licensed initially
  - Switched to Apache V2 in early 2015
- 150+ committers today




METAMARKETS
YAHOO!
PayPal
ebay
Alibaba Group
SK telecom
Time Warner Cable
CISCO
NETFLIX
LDmobile
ARCHIVE-IT
WIKIMEDIA FOUNDATION
CONDÉ NAST
ONE APM
hulu
{LIFE}BUZZ
redborder
YOUKU 优酷
Easemob IM&Service Cloud
SKYPORT SYSTEMS
gumgum
airbnb
criteo
DS DripStat
Streamlyzer
JOLATA
Making the network for you
ViralGains
DiDi
More than a journey
ITALIAONLINE
N3TWORK
YeahMobi
vigiglobe
triplelift
Optimizely
AppsFlyer
liquid
videoamp
PubNative
monetate

Why use Druid?

- Sub-second OLAP Queries
- Real-time Streaming Ingestion
- Multi-tenant with support for 1000+ users
- Cost Effective
- Highly Available
- Scalable

How’s Druid used in prod today?

Ad-hoc analytics.
High concurrency user-facing real-time slice-and-dice.
Real-time loads of 10s of billions of events per day.

Powers infrastructure anomaly detection dashboards.
Ingest rates > 2TB per hour.

Exploratory analytics on clickstream sessions.

Real-time user behavior analytics.

Druid is design for Time-Series use cases

- Time series data workload != normal database workload
  - Queries always contain date columns as group by keys
  - Queries Filters by time
  - Queries touch few columns (< 10)
  - Queries have very selective Filters (< thousands of rows out of billions)

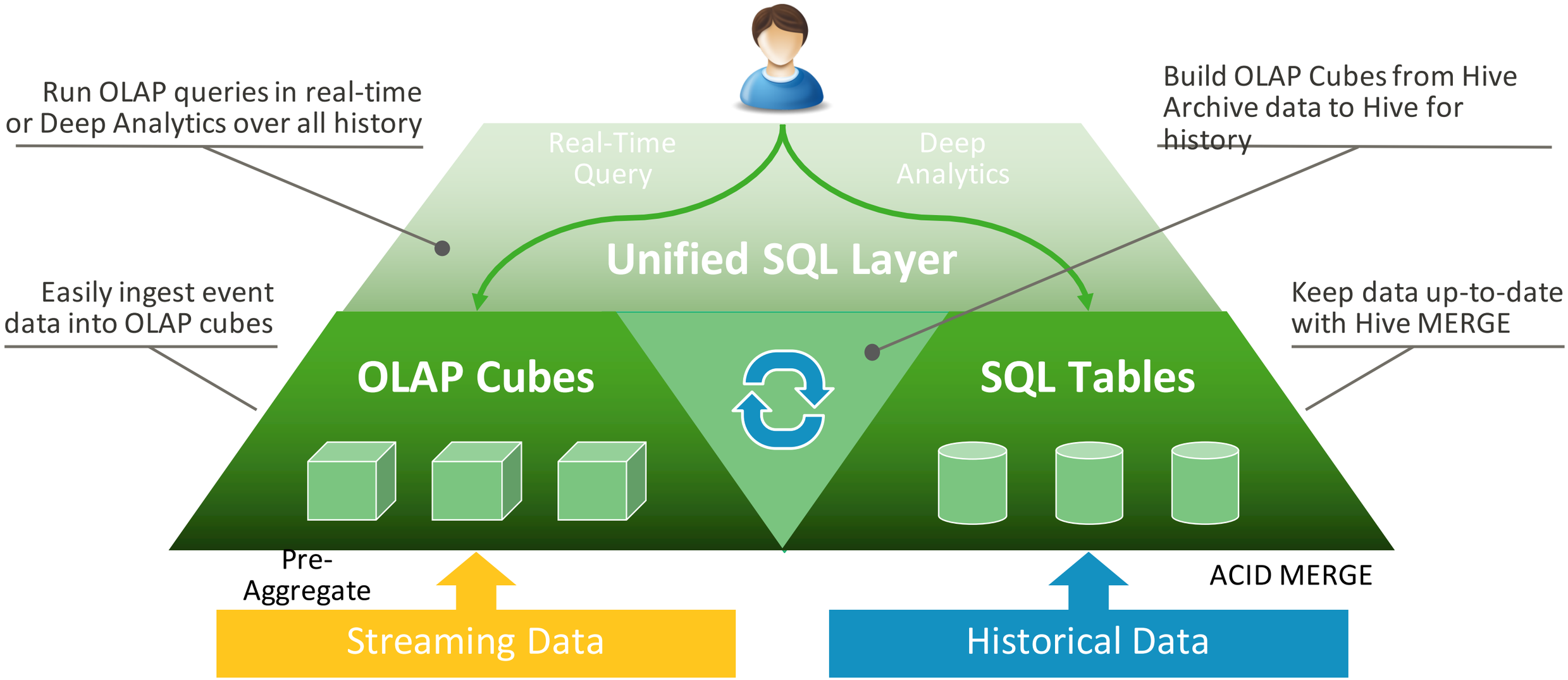
```
SELECT `user`, sum(`c_added`) AS s, EXTRACT(year FROM `__time`)
FROM druid_table
WHERE EXTRACT(year FROM `__time`)
  BETWEEN 2010 AND 2011
GROUP BY `user`, EXTRACT(year FROM `__time`)
ORDER BY s DESC
LIMIT 10;
```

Where do time series data coming from?

- Mostly an insert/append workload, very few updates
  - Event stream data
  - Application/server logs
  - Sensor logs
- Schema less
  - New event types
  - New server/application/OS type or upgrade
  - New sensors

Druid @ HDP

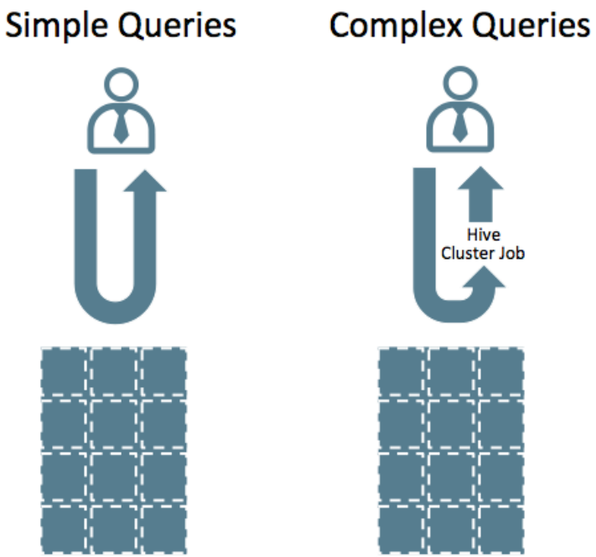
Hive + Druid = Insight When You Need It



Why Use Druid From Hortonworks?

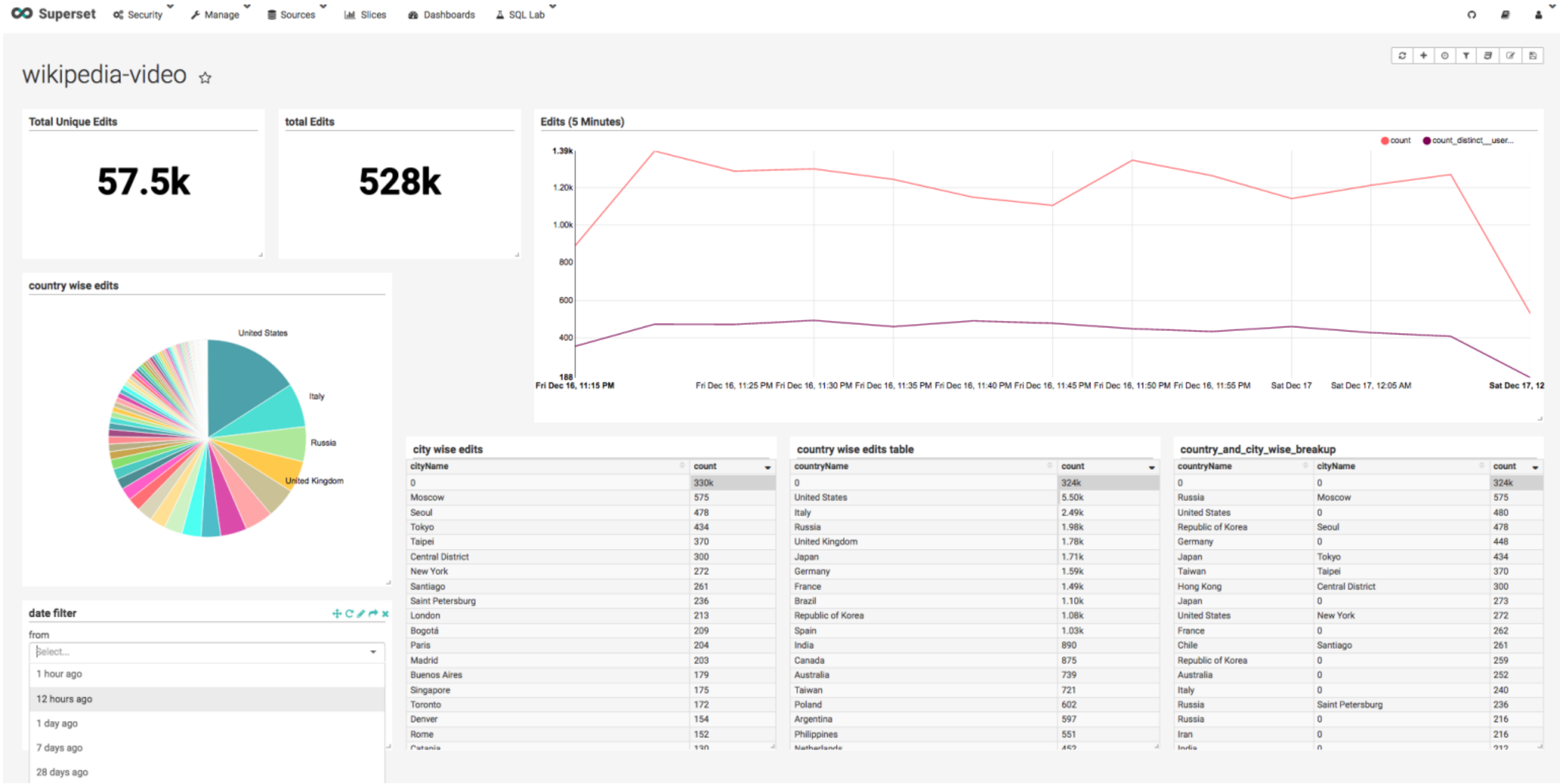
	With HDP	Druid Alone
Interactive Analytics	✓	✓
Analyze Data Streams	✓	✓
Spatial Analytics	✓	✓
Horizontally Scalable	✓	✓
SQL:2011 Interface	✓	✗
Join Historical and Real-time Data	✓	✗
Management and Monitoring with Ambari	✓	✗
Managed Rolling Upgrades	✓	✗
Visualization with Superset	✓	✗
Easy App Development with Hortonworks SAM	✓	✗

Druid-Hive integration makes things easy



Layer	Strongest Access Pattern	Features
Hive Layer	Large scale brute-force analytics	Joins Subqueries Windowing Functions Transformations Complex Aggregations Advanced Sorting UDFs
Druid Layer	Needles-in-a-haystack queries with large numbers of dimensions	Dimensional Aggregates Top N Queries Min / Max Values Timeseries Queries

Superset UI for Fast, Interactive Dashboards and Exploration



Agenda

- Druid overview
- Time series data
- Druid architecture
- Demo
- Questions

Druid Internals: Segments (indexes)

Druid is design for Time-Series use cases

- Time series data workload != normal database workload
- Queries always contain date columns as group by keys.
- Queries Filters by time
- Queries touch few columns (< 10)
- Queries have very selective Filters (< thousands of rows out of billions)

```
SELECT `user`, sum(`c_added`) AS s, EXTRACT(year FROM `__time`)
FROM druid_table
WHERE EXTRACT(year FROM `__time`)
  BETWEEN 2010 AND 2011
GROUP BY `user`, EXTRACT(year FROM `__time`)
ORDER BY s DESC
LIMIT 10;
```

Druid: Segment Data Structures

- Within a Segment:
  - Timestamp Column Group.
  - Dimensions Column Group.
  - Metrics Column Group.
  - Indexes that facilitate fast lookup and aggregation.

Timestamp | Dimensions | Metrics

Timestamp	Page	Username	Gender	City	Characters Added	Characters Removed
2011-01-01T01:00:00Z	Justin Bieber	Boxer	Male	San Francisco	1800	25
2011-01-01T01:00:00Z	Justin Bieber	Reach	Male	Waterloo	2912	42
2011-01-01T02:00:00Z	Ke$ha	Helz	Male	Calgary	1953	17
2011-01-01T02:00:00Z	Ke$ha	Xeno	Male	Taiyuan	3194	170

Data is partitioned by time !

timestamp	publisher	advertiser	gender	country	...	click	price
2011-01-01T00:01:35Z	bieberfever.com	google.com	Male	USA		0	0.65
2011-01-01T00:03:63Z	bieberfever.com	google.com	Male	USA		0	0.62
2011-01-01T00:04:51Z	bieberfever.com	google.com	Male	USA		1	0.45
2011-01-01T01:00:00Z	ultratrimfast.com	google.com	Female	UK		0	0.87
2011-01-01T02:00:00Z	ultratrimfast.com	google.com	Female	UK		0	0.99
2011-01-01T02:00:00Z	ultratrimfast.com	google.com	Female	UK		1	1.53
...							

- Time partitioned immutable partitions.
- Typically 5 Million row per segment.
- Druid segments are stored in a column orientation (only what is needed is actually loaded and scanned).

COLUMN COMPRESSION - DICTIONARIES

timestamp	publisher	advertiser	gender	country	...	click	price
2011-01-01T00:01:35Z	bieberfever.com	google.com	Male	USA		0	0.65
2011-01-01T00:03:63Z	bieberfever.com	google.com	Male	USA		0	0.62
2011-01-01T00:04:51Z	bieberfever.com	google.com	Male	USA		1	0.45
2011-01-01T01:00:00Z	ultratrimfast.com	google.com	Female	UK		0	0.87
2011-01-01T02:00:00Z	ultratrimfast.com	google.com	Female	UK		0	0.99
2011-01-01T02:00:00Z	ultratrimfast.com	google.com	Female	UK		1	1.53
...							

- Create ids
  - bieberfever.com -> 0, ultratrimfast.com -> 1
- Store
  - publisher -> [0, 0, 0, 1, 1, 1]
  - advertiser -> [0, 0, 0, 0, 0, 0]

BITMAP INDEXES

timestamp	publisher	advertiser	gender	country	...	click	price
2011-01-01T00:01:35Z	bieberfever.com	google.com	Male	USA		0	0.65
2011-01-01T00:03:63Z	bieberfever.com	google.com	Male	USA		0	0.62
2011-01-01T00:04:51Z	bieberfever.com	google.com	Male	USA		1	0.45
2011-01-01T01:00:00Z	ultratrimfast.com	google.com	Female	UK		0	0.87
2011-01-01T02:00:00Z	ultratrimfast.com	google.com	Female	UK		0	0.99
2011-01-01T02:00:00Z	ultratrimfast.com	google.com	Female	UK		1	1.53
...							

- bieberfever.com -> [0, 1, 2] -> [111000]
- ultratrimfast.com -> [3, 4, 5] -> [000111]
- Compress using Concice or Roaring (take advantage of dimension sparsity)

FAST AND FLEXIBLE QUERIES

Rows	POETS
0	JUSTIN BIEBER
1	JUSTIN BIEBER
2	KE$HA
3	KE$HA

JUSTIN BIEBER
[1, 1, 0, 0]

KE$HA
[0, 0, 1, 1]

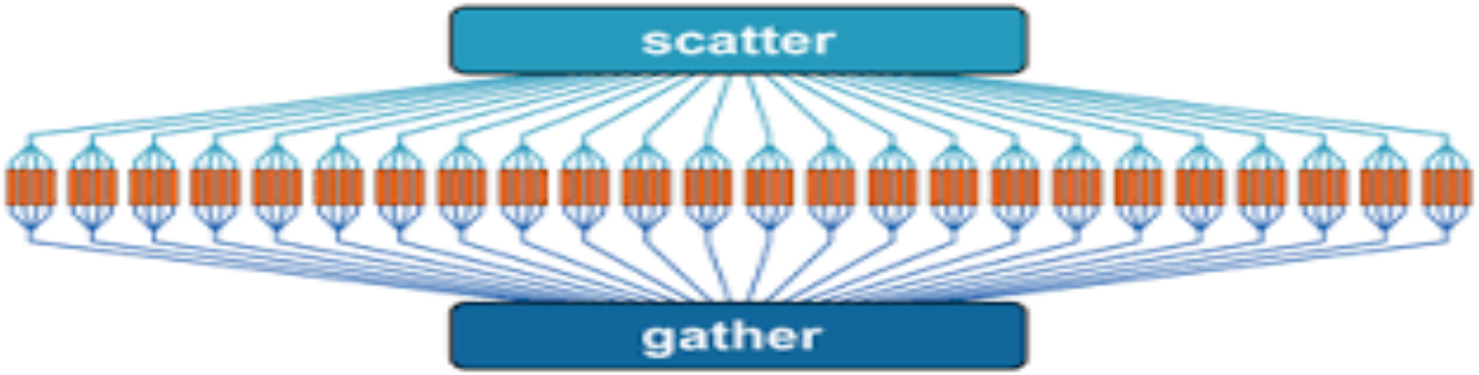
JUSTIN BIEBER
OR
KE$HA
[1, 1, 1, 1]

Queries that solely aggregate metrics based on filters do not need to touch the list of dimension values !

Druid Architecture

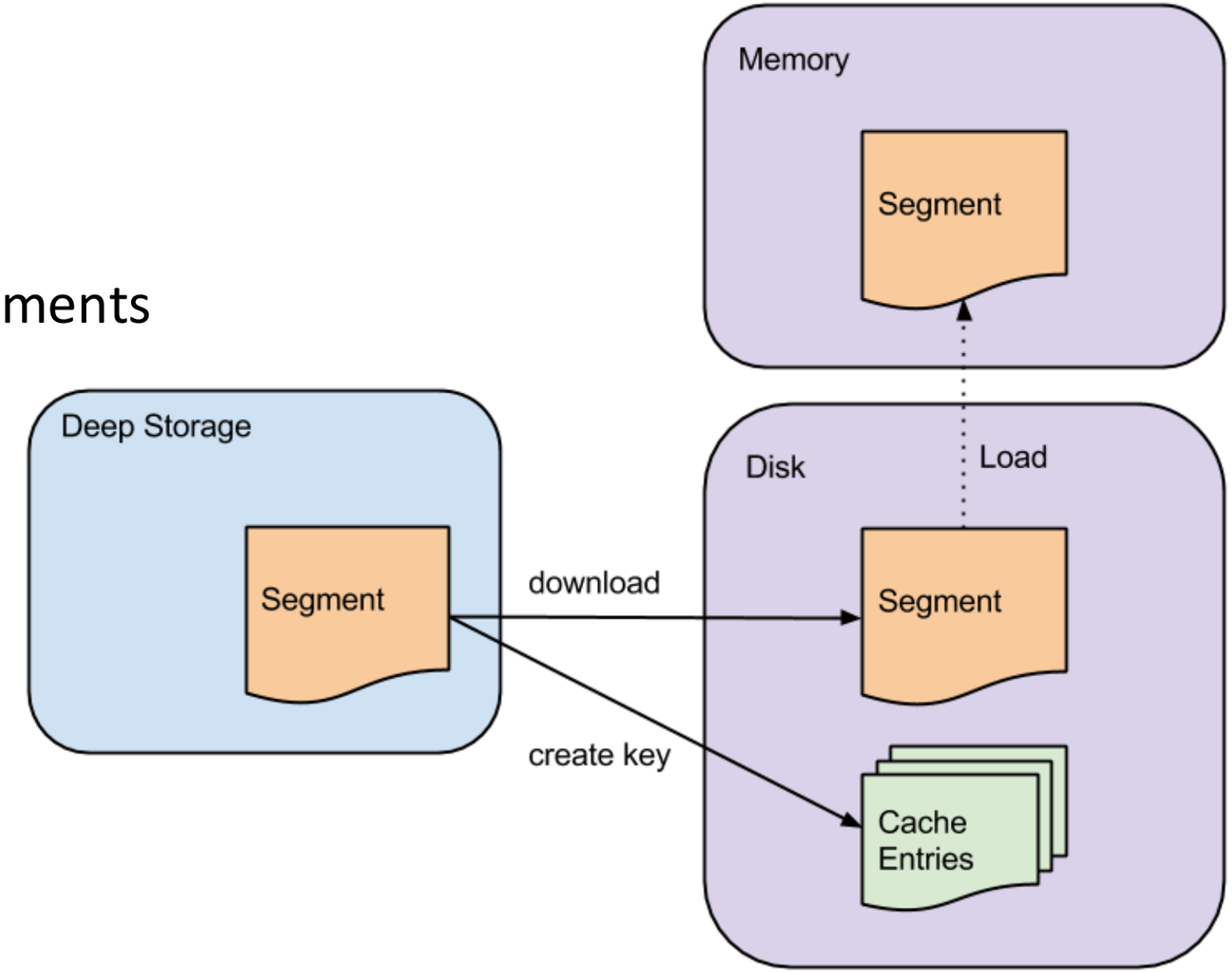
Broker Nodes

- Keeps track of segment announcements in cluster
  - (This information is kept in **Zookeeper**, much like Storm or HBase do.)
- Scatters query across historical and realtime nodes
  - (Clients issue queries to this node, but queries are processed elsewhere.)
- Merge results from different query nodes
- (Distributed) caching layer

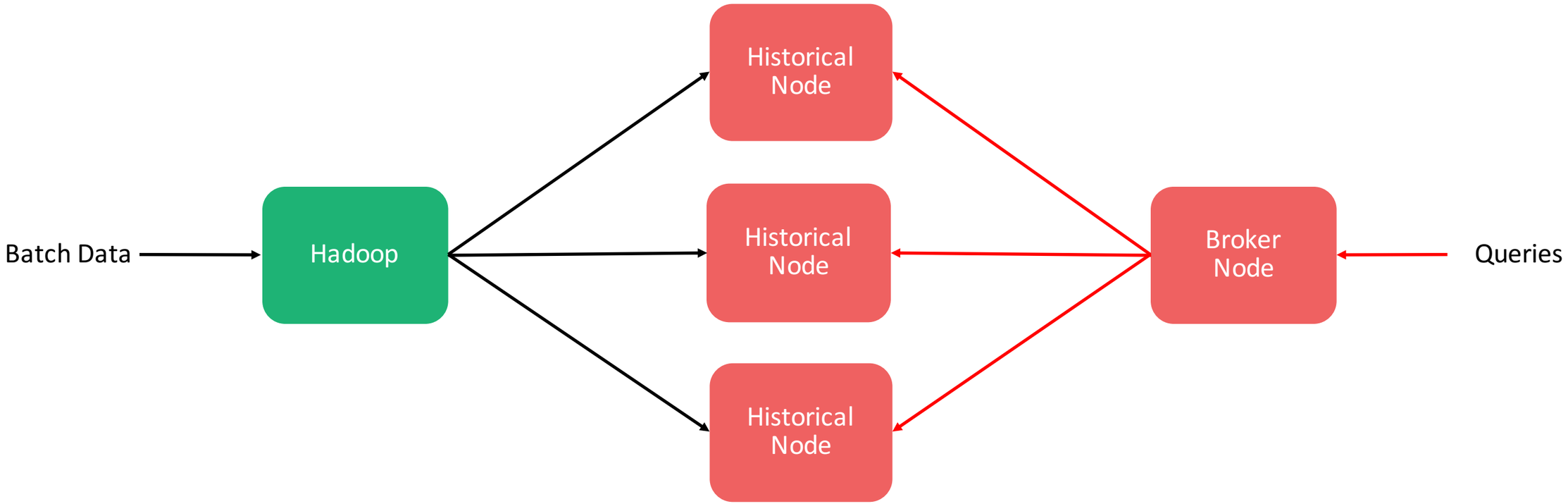


Historical Nodes

- Shared nothing architecture
- Main workhorses of druid cluster
- Load immutable read optimized segments
- Respond to queries
- Use memory mapped files to load segments

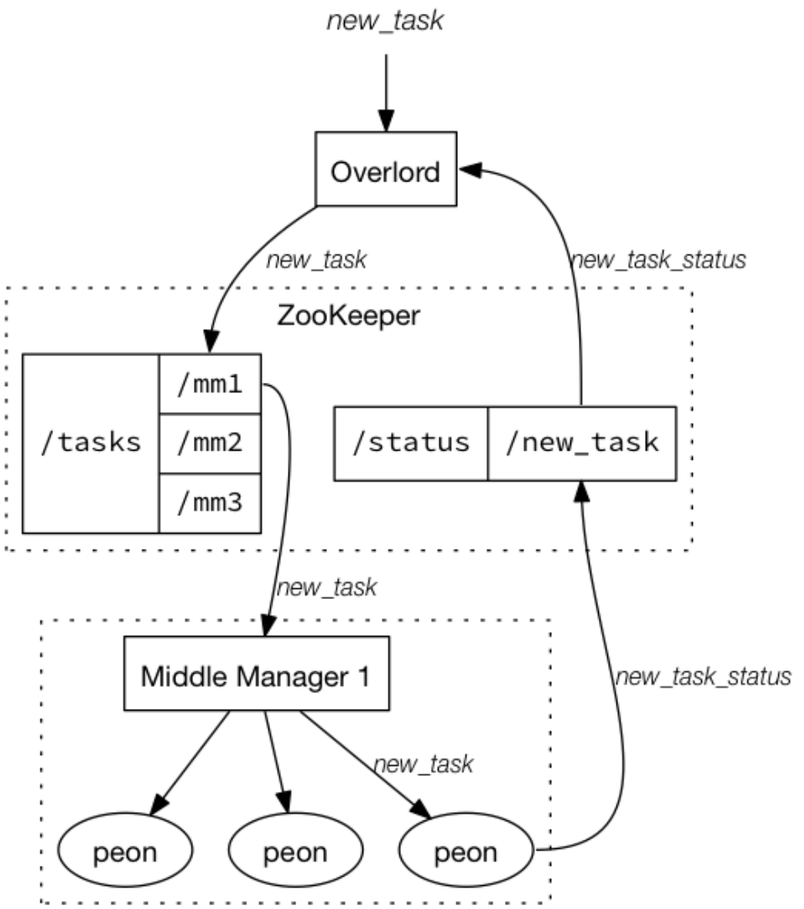


Early Druid Architecture

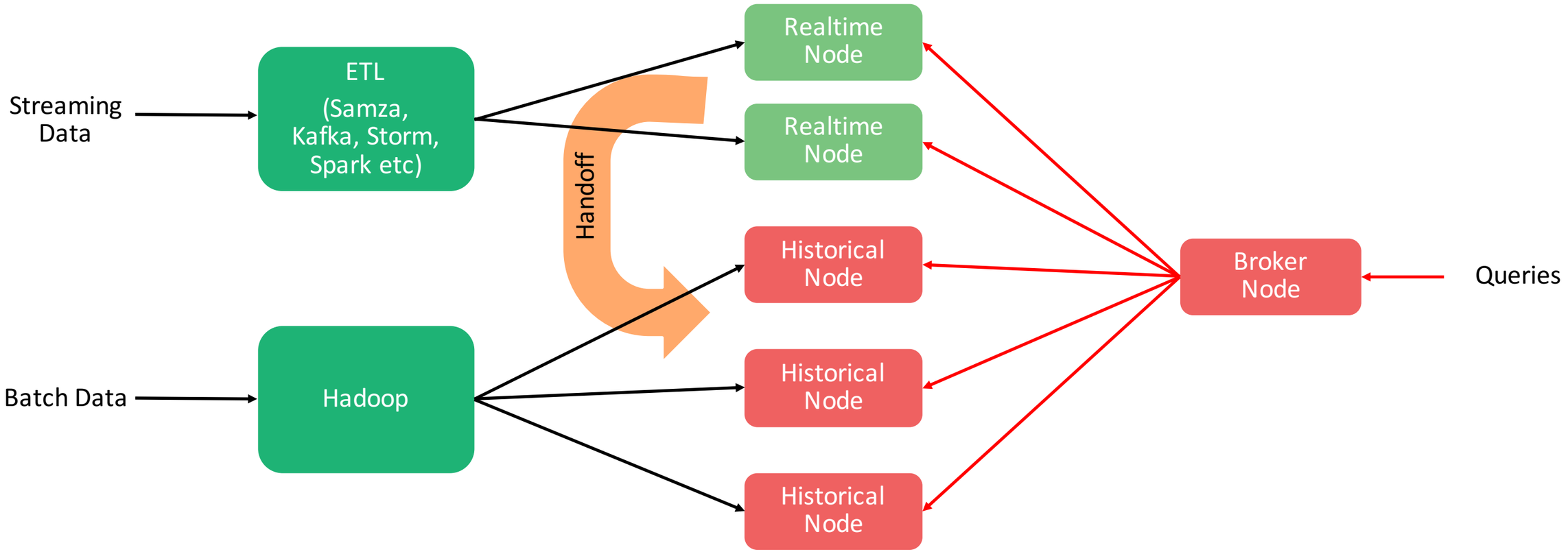


Druid: Batch Indexing

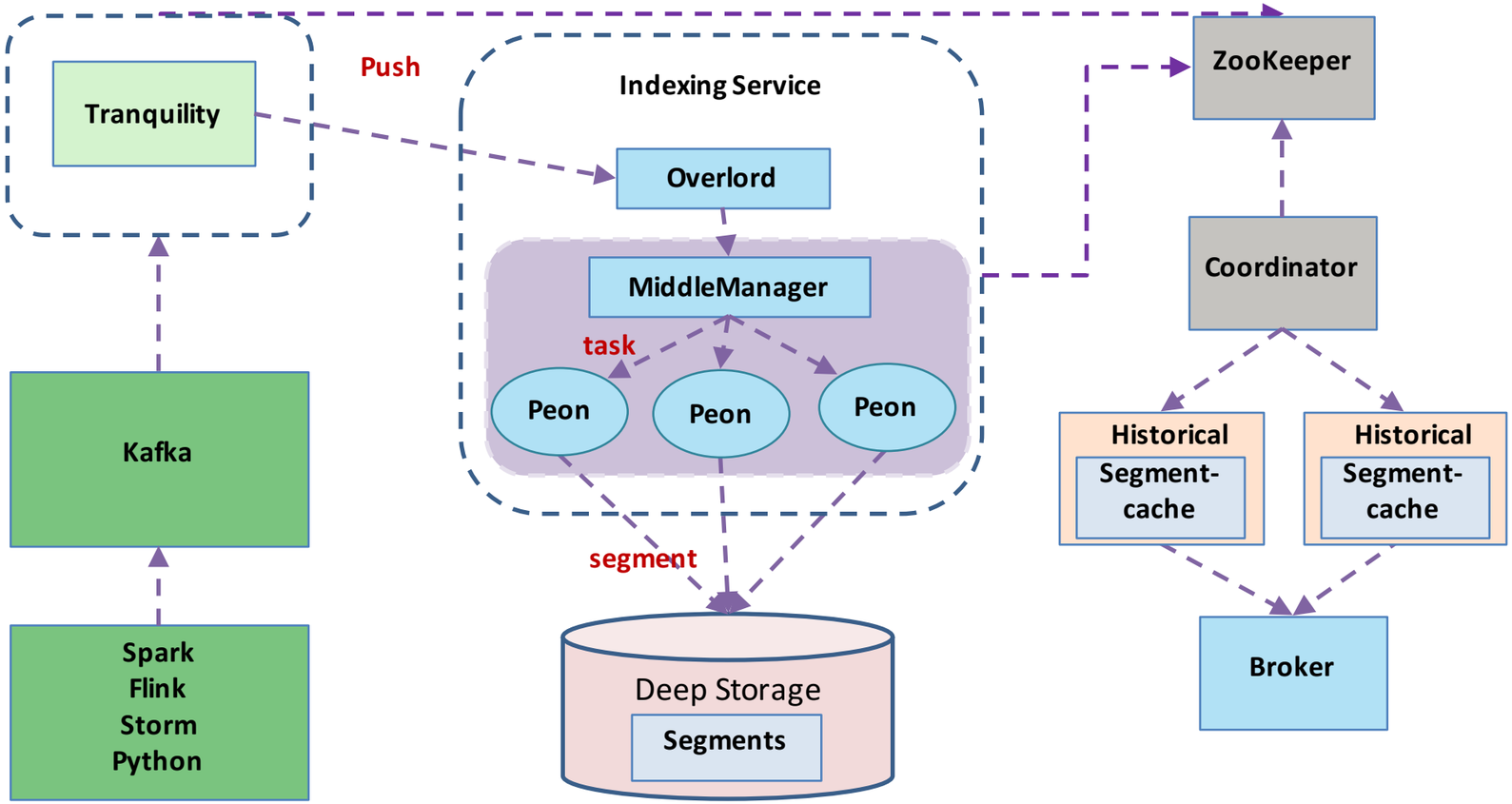
- Indexing is performed by related components:
  - Overlord (N = 1)
  - Middle Managers (N >= 1)
- Indexing is done via MapReduce jobs that build Segment files.
- Batch indexing is done on data that already exists in Deep Storage (e.g. HDFS).
- Index definition is specified via a JSON file and submitted to the Overlord.



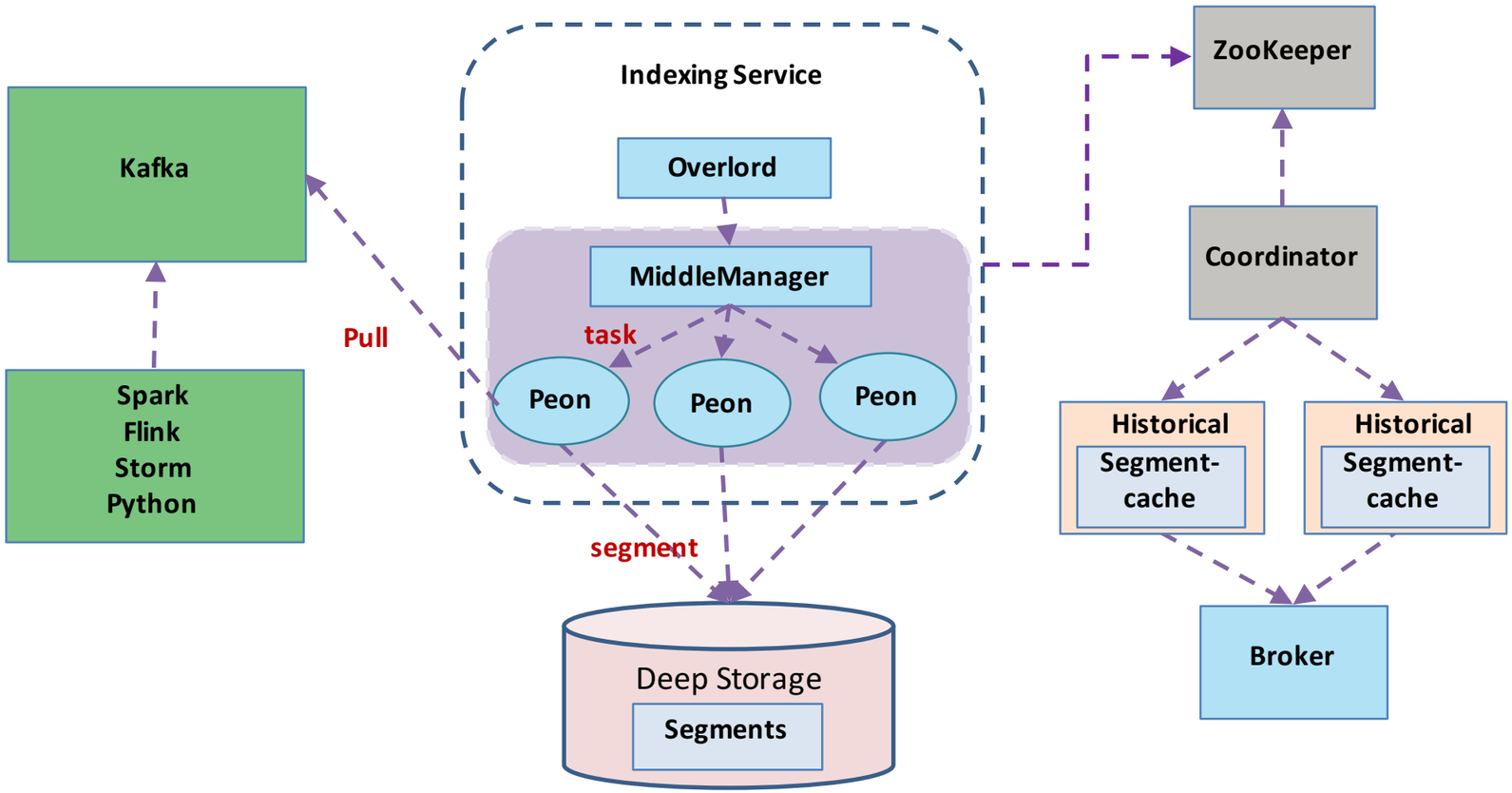
Current Druid Architecture



Druid: Realtime Indexing Push Mode

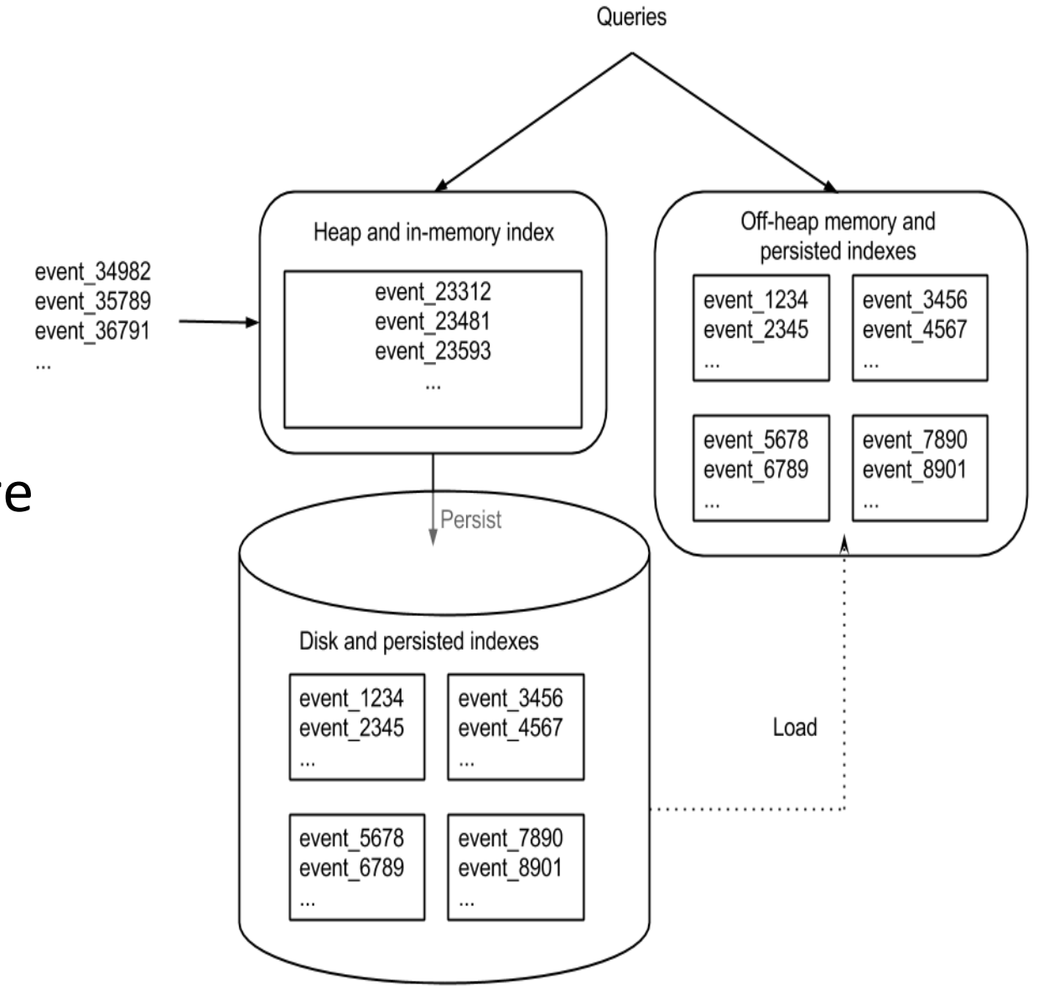


Druid: Realtime Indexing Pull Mode



Realtime Nodes

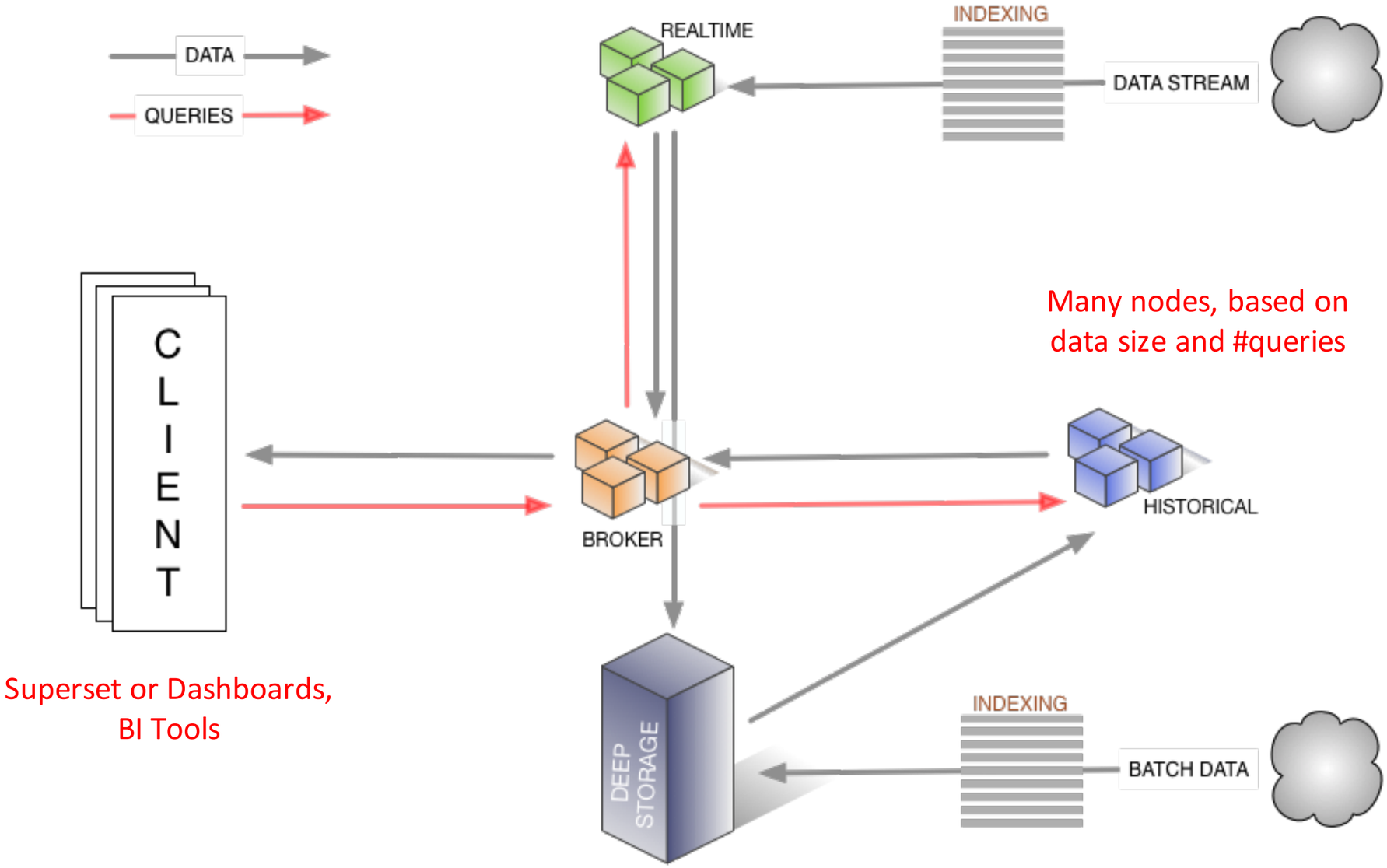
- Ability to ingest streams of data
- Both push and pull based ingestion
- Stores data in write-optimized structure
- Periodically converts write-optimized structure to read-optimized segments
- Event query-able as soon as it is ingested



Coordinator Nodes

- Assigns segments to historical nodes
- Interval based cost function to distribute segments
- Makes sure query load is uniform across historical nodes
- Handles replication of data
- Configurable rules to load/drop data

A Typical Druid Deployment



Registering and creating Druid data sources

Druid data sources in Hive

- User needs to provide Druid data sources information to Hive
- Two different options depending on requirements
  - Register Druid data sources in Hive
    - Data is already stored in Druid
  - Create Druid data sources from Hive
    - Data is stored in Hive
    - User may want to pre-process the data before storing it in Druid

Druid data sources in Hive

Registering Druid data sources

- Simple *CREATE EXTERNAL TABLE* statement

```
CREATE EXTERNAL TABLE druid_table_1
STORED BY 'org.apache.hadoop.hive.druid.DruidStorageHandler'
TBLPROPERTIES ("druid.datasource" = "wikiticker");
```

Hive table name
Hive storage handler classname
Druid data source name

  - Broker node endpoint specified as a Hive configuration parameter
  - Automatic Druid data schema discovery: *segment metadata* query

Druid data sources in Hive

Creating Druid data sources

- Use *Create Table As Select (CTAS)* statement

```
CREATE TABLE druid_table_1
STORED BY 'org.apache.hadoop.hive.druid.DruidStorageHandler'
TBLPROPERTIES ("druid.datasource" = "wikiticker", "druid.segment.granularity" = "DAY")
AS
SELECT __time, page, user, c_added, c_removed
FROM src;
```

Hive table name
Hive storage handler classname
Druid data source name
Druid segment granularity

Querying Druid data sources

Querying Druid data sources

- Automatic rewriting when query is expressed over Druid table
  - Powered by **Apache Calcite**
  - Main challenge: identify patterns in logical plan corresponding to different kinds of Druid queries (*Timeseries*, *TopN*, *GroupBy*, *Select*)
- Translate (sub)plan of operators into valid Druid JSON query
  - Druid query is encapsulated within Hive TableScan operator
- Hive TableScan uses **Druid input format**
  - Submits query to Druid and generates records out of the query results
- It might not be possible to push *all* computation to Druid
  - Our contract is that the query should always be executed

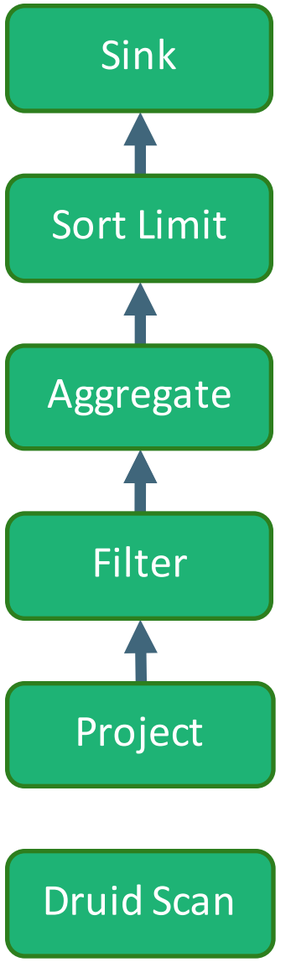
Druid query recognition (powered by Apache Calcite)

Apache Hive - SQL query

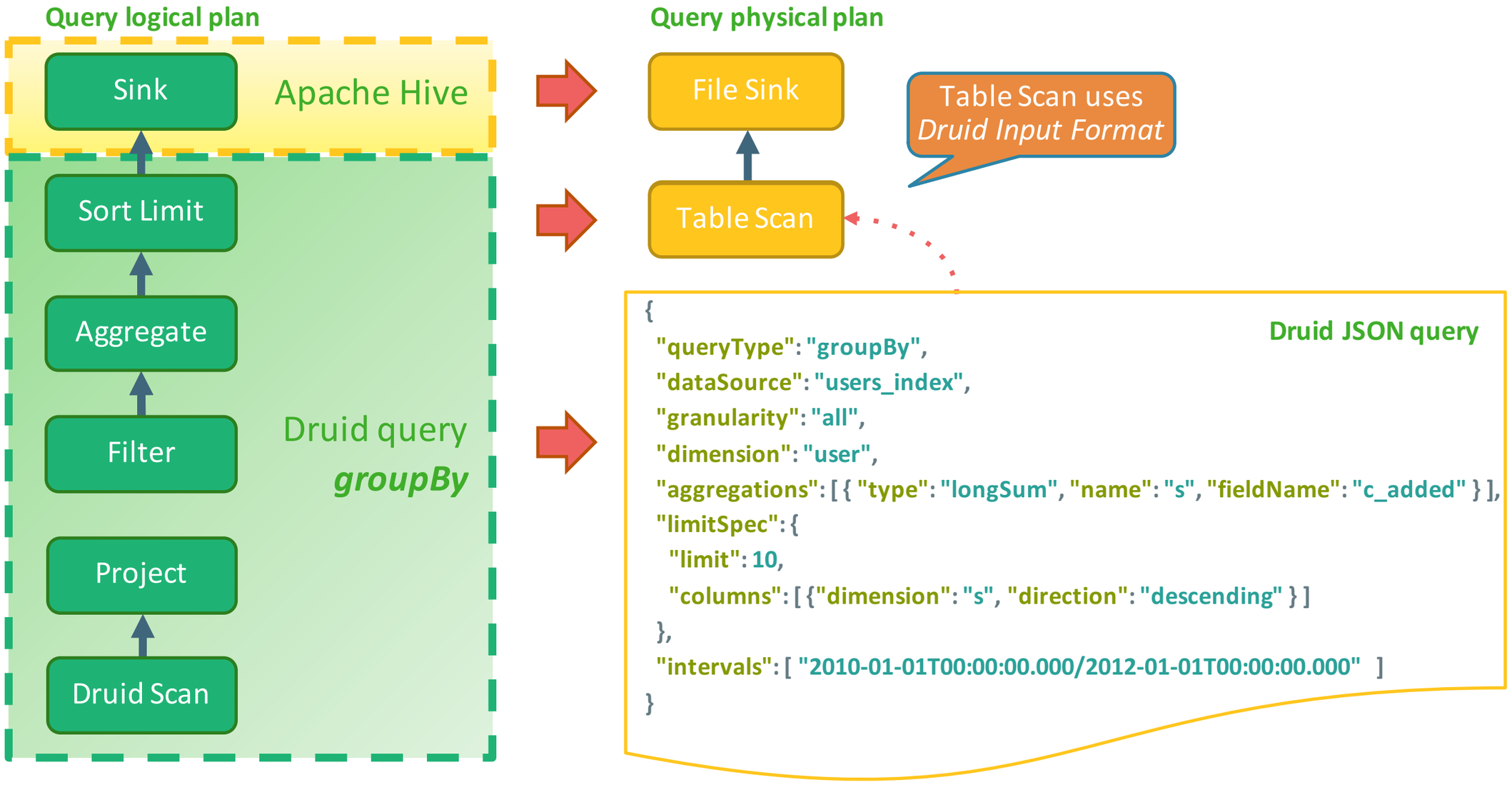
```
SELECT `user`, sum(`c_added`) AS s
FROM druid_table_1
WHERE EXTRACT(year FROM `__time`)
  BETWEEN 2010 AND 2011
GROUP BY `user`
ORDER BY s DESC
LIMIT 10;
```

- *Top 10 users that have added more characters from beginning of 2010 until the end of 2011*

Query logical plan



Physical plan transformation



Road ahead

- Tighten integration between **Druid** and **Apache Hive**/**Apache Calcite**
  - Recognize more functions → Push more computation to Druid
  - Support complex column types
  - Close the gap between semantics of different systems
    - Time zone handling, *null* values

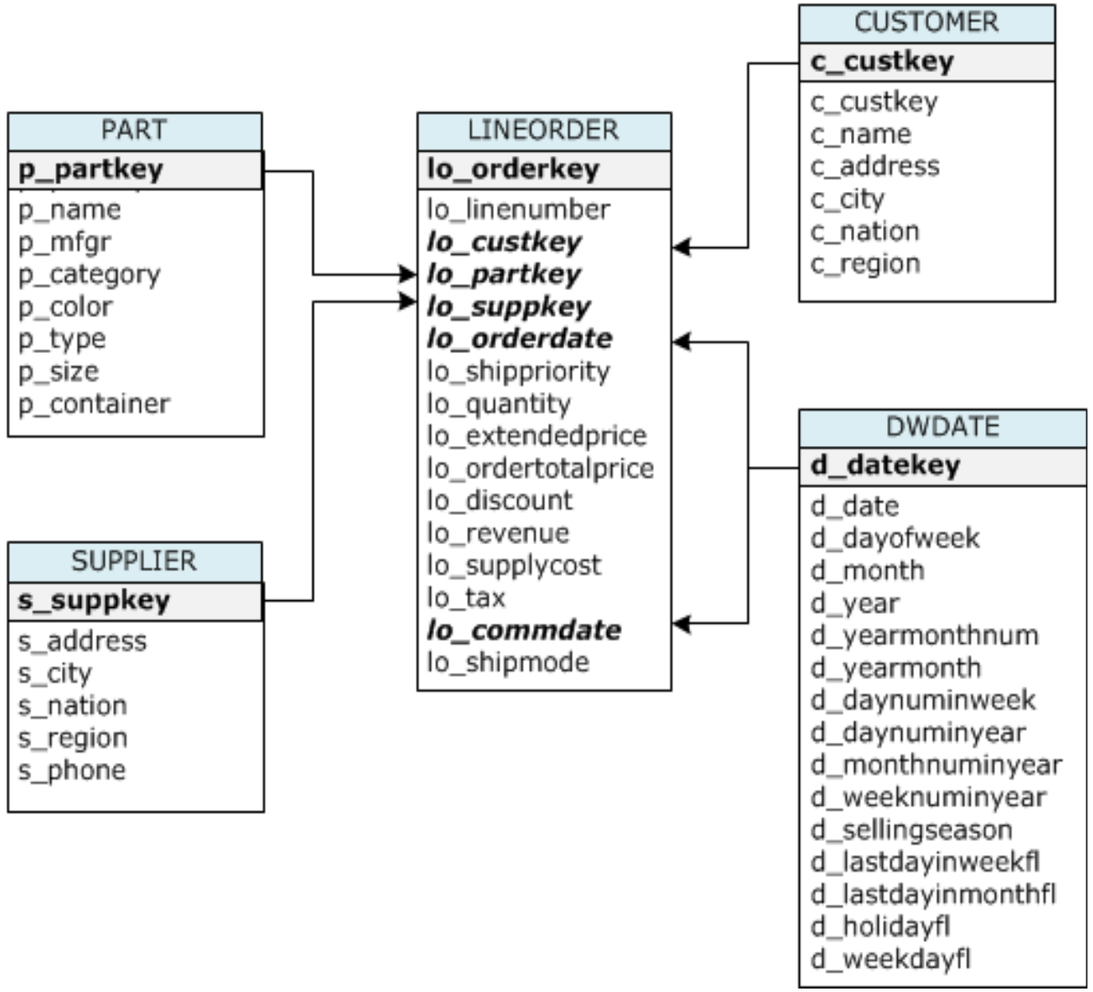
- Broader perspective
  - **Materialized views** support in Apache Hive
    - Data stored in Apache Hive
    - Create materialized view in Druid
      - Denormalized star schema for a certain time period
    - Automatic input query rewriting over the materialized view (Apache Calcite)

SSB Benchmarks

Schema and Data Model: Base Tables

Star schema benchmark

- Line order 6 billion rows
- Customer 30,000,000 rows
- Supplier 2,000,000
- Parts 1,400,000



Picture Credit

http://docs.aws.amazon.com/redshift/latest/dg/tutorial-tuning-tables-create-test-data.html

Benchmark Setup

Platform

- 11 Nodes 2x Intel(R) Xeon(R) CPU E5-2640 v2 @ 2.00GHz with 16 CPU threads each
- 256G of Ram per node
- 6x WDC WD4000FYYZ-0 1K02 4TB SCSI disks per node
- Not 100% dedicated (part of Yarn managed cluster)

Druid

- 10 / 5 /3 Historical nodes (Workhorse)
  - -Xmx12g -Xms12g -XX:NewSize=6g -XX:MaxNewSize=6g -XX:MaxDirectMemorySize=128g
- 1 node Broker
  - -Xmx25g -Xms25g -XX:NewSize=6g -XX:MaxNewSize=6g -XX:MaxDirectMemorySize=64g
- No segments replication
  - Query to server mapping is one to one
- No caching
- No Tuning out of the box configuration

Schema and Data Model: Denormalized Table in Druid

- 6 Billion rows (no Rollups)
- 174 GB over 7 years
- Segmented by month
  - One partition is worth 1/9 month of data
  - 10 Million rows per partition
  - Segment size ~ 262 MB
  - 9 Million rows per partition
  - 707 partitions in total

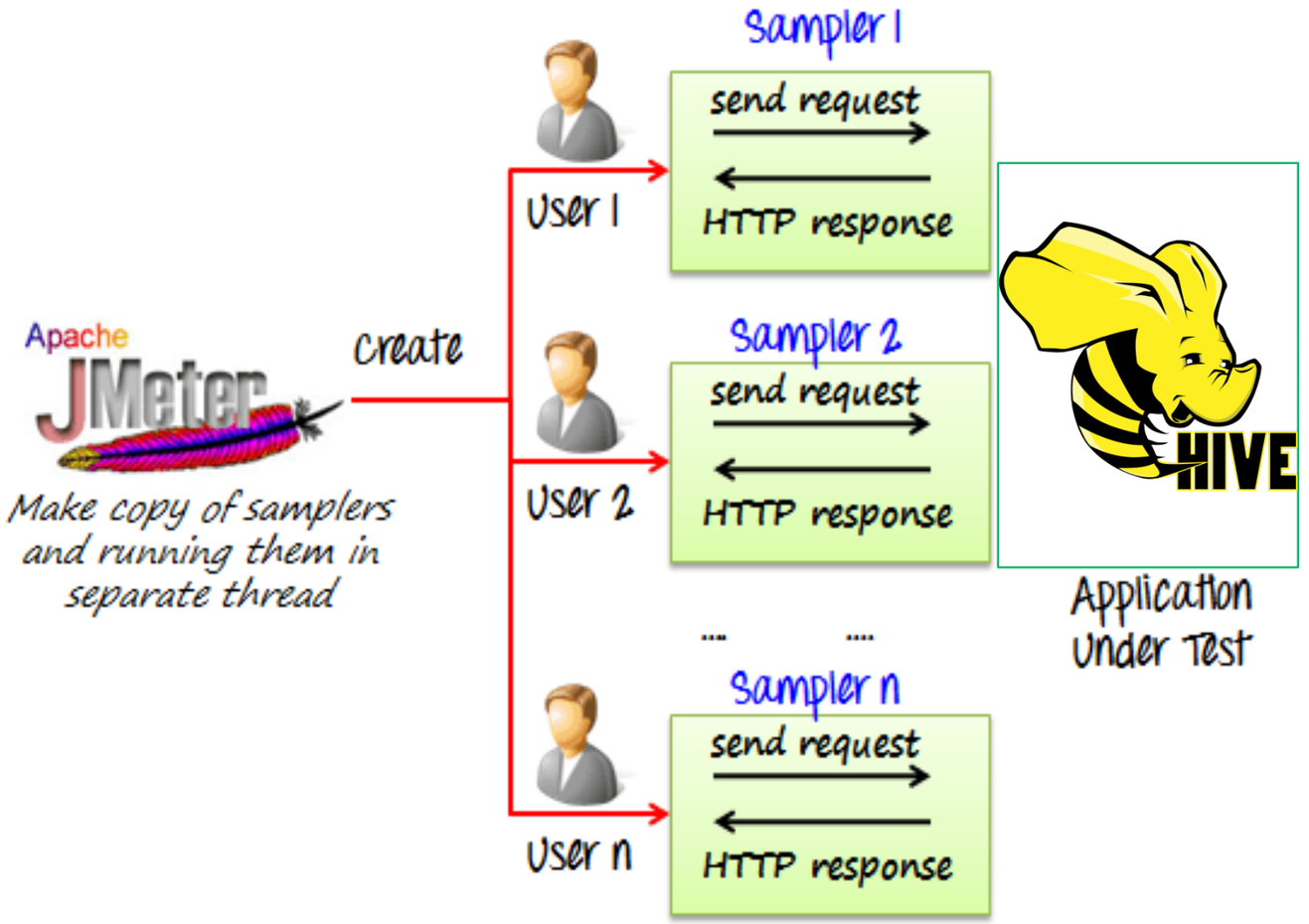
```
CREATE TABLE ssb_druid_month
STORED BY 'org.apache.hadoop.hive.druid.DruidStorageHandl
TBLPROPERTIES (
  "druid.datasource" = "ssb_druid",
  "druid.segment.granularity" = "MONTH",
  "druid.query.granularity" = "DAY")
AS
SELECT
  cast(d_year || '-' ||
       d_monthnuminyear || '-' ||
       d_daynuminmonth as timestamp) as '__time',
  cast(c_city as string) c_city,
  cast(c_nation as string) c_nation,
  cast(c_region as string) c_region,
  cast(d_weeknuminyear as string) d_weeknuminyear,
  cast(d_year as string) d_year,
  cast(d_yearmonth as string) d_yearmonth,
  cast(d_yearmonthnum as string) d_yearmonthnum,
  cast(lo_discount as string) lo_discount,
  cast(lo_quantity as string) lo_quantity,
  cast(p_brand1 as string) p_brand1,
  cast(p_category as string) p_category,
  cast(p_mfgr as string) p_mfgr,
  cast(s_city as string) s_city,
  cast(s_nation as string) s_nation,
  cast(s_region as string) s_region,
  lo_revenue,
  lo_extendedprice * lo_discount discounted_price,
  lo_revenue - lo_supplycost net_revenue
FROM
  ssb_1000_flat_orc.customer, ssb_1000_flat_orc.dates,
  ssb_1000_flat_orc.lineorder,
  ssb_1000_flat_orc.part, ssb_1000_flat_orc.supplier
where
  lo_orderdate = d_datekey and lo_partkey = p_partkey
and lo_suppkey = s_suppkey and lo_custkey = c_custkey;
```


SSB Queries

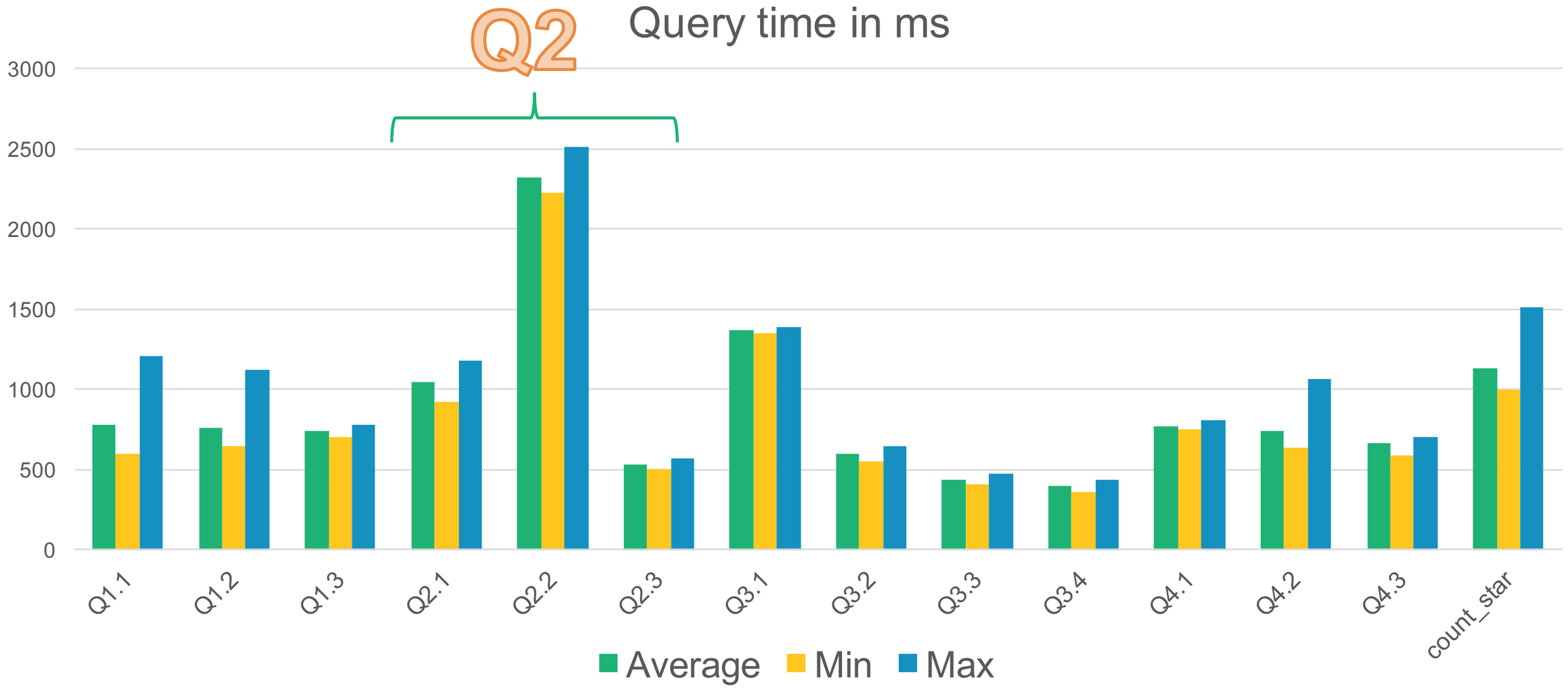
- Q1 (Quick metrics) the amount of revenue increase that would have resulted from eliminating certain company
  - No Group by
  - Years to weeks of data
- Q2 (Product sales insights) compares revenue for some product classes, for suppliers in a certain region, grouped by more restrictive product classes and all years of orders.
  - Couple of Group by keys
  - All data to one month
  - Less selective to very selective filter
- Q3 (Customer insights) revenue volume by customer nation and supplier nation and year within a given region, in a certain time period
  - Greater than 3 Group by keys
  - Years of data
- Q4 "What-If" sequence, of the OLAP type
  - Same as Q3

Benchmark execution setup

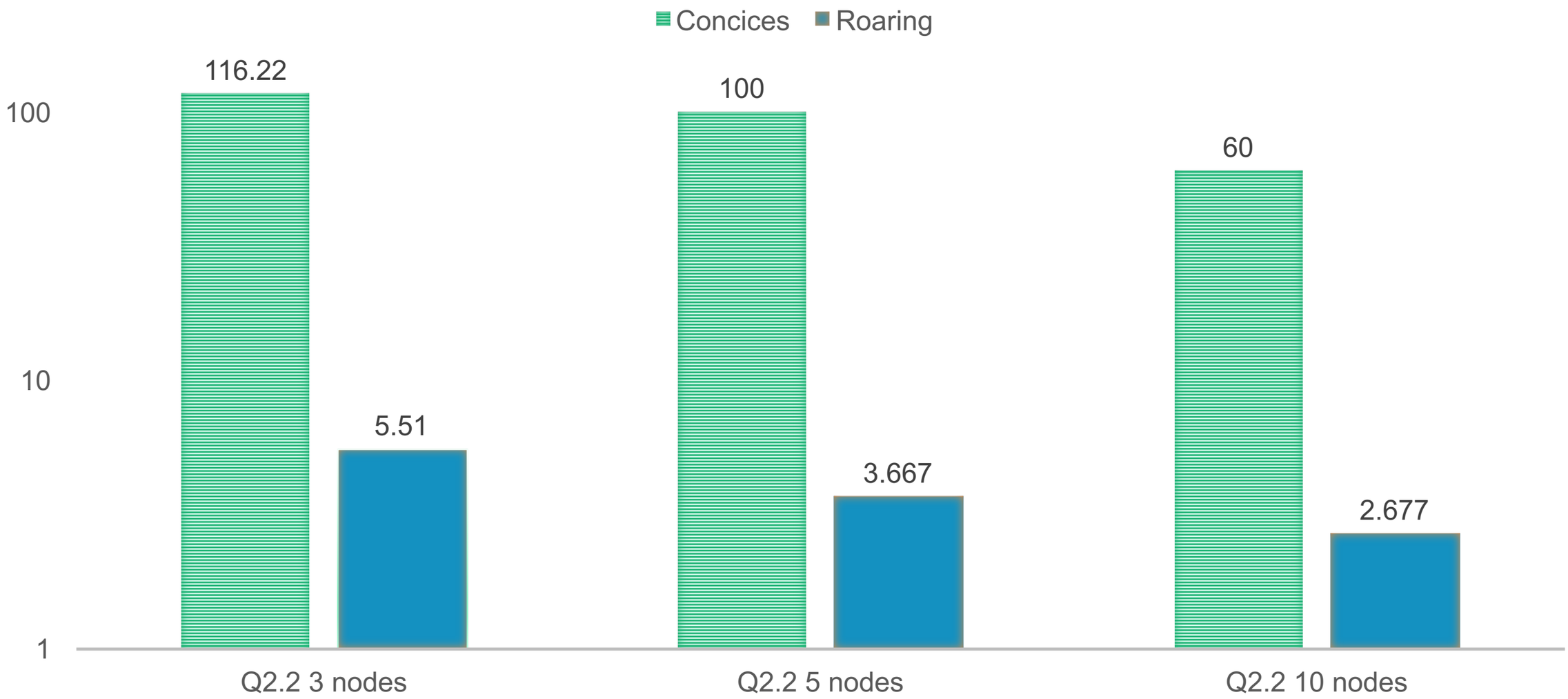
- Run using Apache Jmeter
- Every Query run 15 times capture Min/Max/AVG/STD
- Query time includes JDBC overhead
- Launch a warmup query first to initialize the metadata
- Concurrency mode Launches multiple threads via different JDBC connections
- Druid 0.9.2 (HDP 2.6)
- Hive 2.1 (HDP 2.6)



Q1.1 ({*}, {yearLevel = 1993, 1 < discountLevel < 3, quantityLevel < 25}, {sum_revenue})
Q1.2 ({*}, {yearmonthnumLevel = 199401, 4 < discountLevel < 6, 26 < quantityLevel < 35}, {sum_revenue})
Q1.3 ({weeknuminyearYearLevel}, {5 < discountLevel < 7, 26 < quantityLevel < 35}, {sum_revenue})

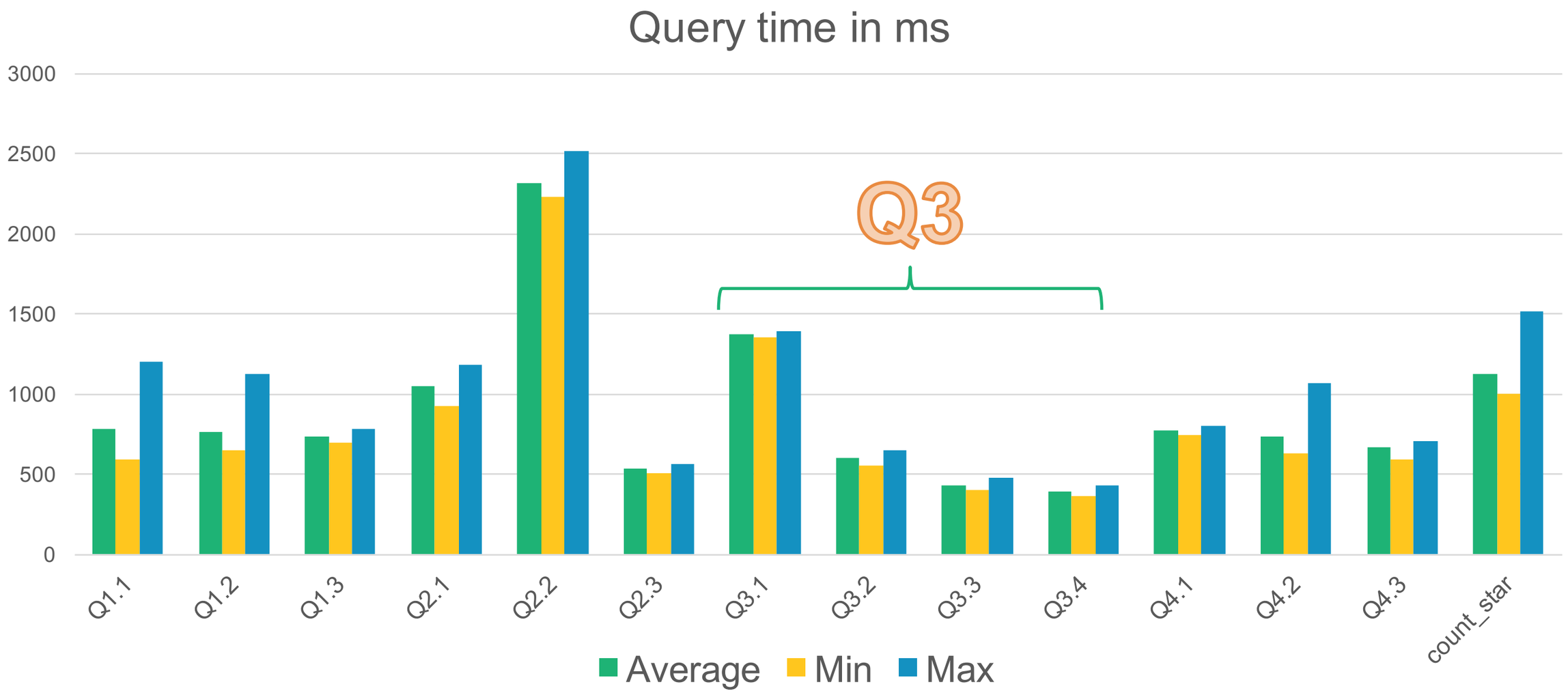


(4) ({yearLevel, brand1Level}, {categoryLevel = MFGR#12, s_regionLevel = AMERICA}, {lo_revenue})
(5)({yearLevel, brand1Level, s_regionLevel}, {brand1Level = MFGR#2221 OR ... OR brand1level = MFGR#2228,
gionLevel = ASIA}, {lo_revenue})
(6)({yearLevel, brand1Level, s_regionLevel}, {brand1Level = MFGR#2239, s_regionLevel = EUROPE}, {lo_revenue})

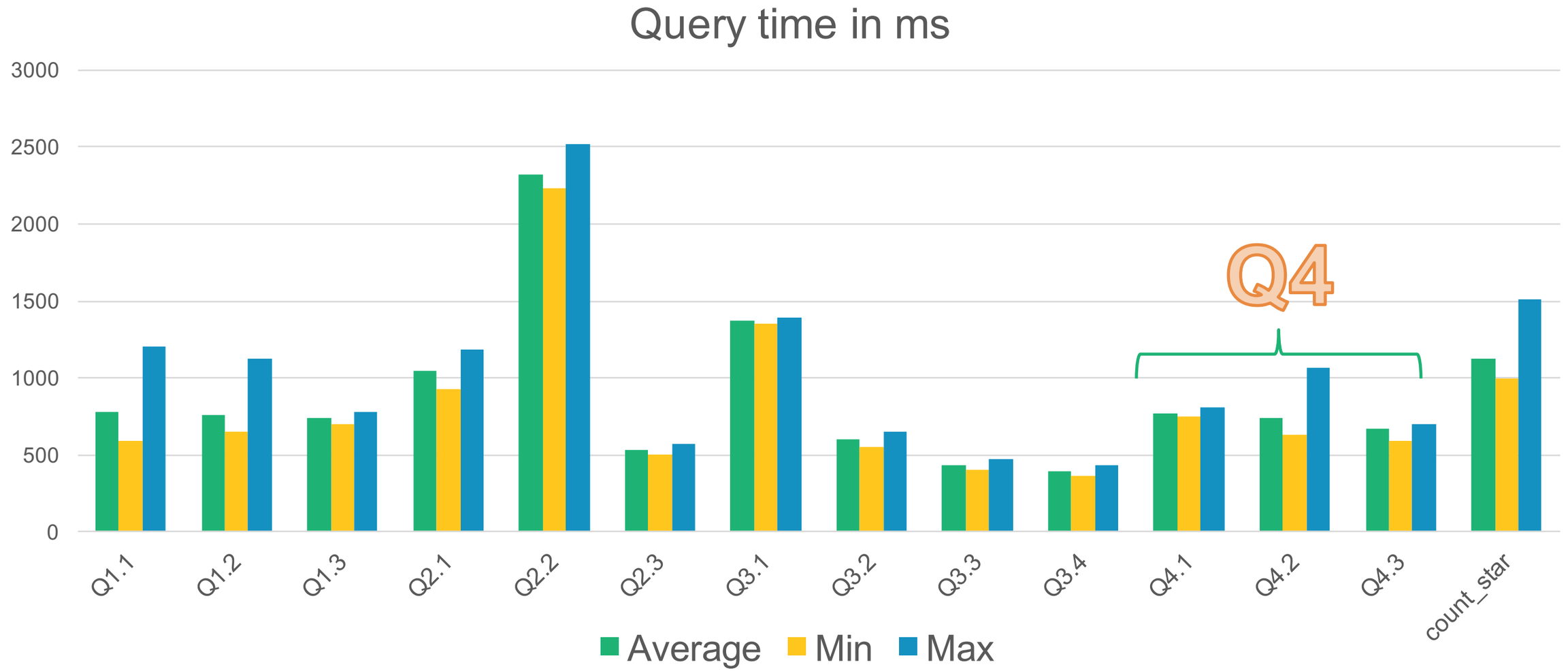


Q2.2 (5)({yearLevel, brand1Level, s_regionLevel}, {brand1Level = MFGR#2221 OR ... OR brand1level = MFGR#2228, s_regionLevel = ASIA}, {lo_revenue})

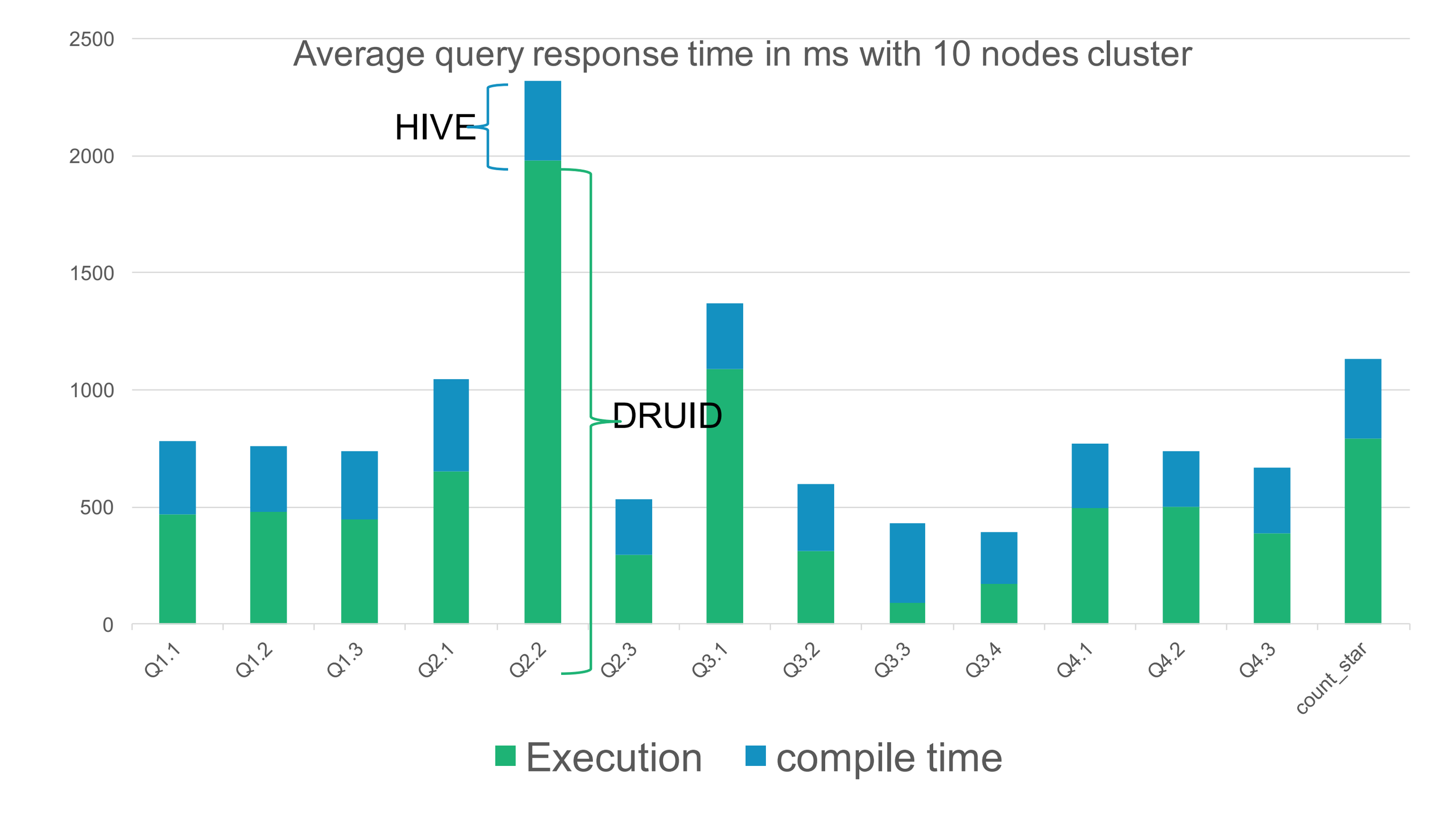
Union of bitmaps



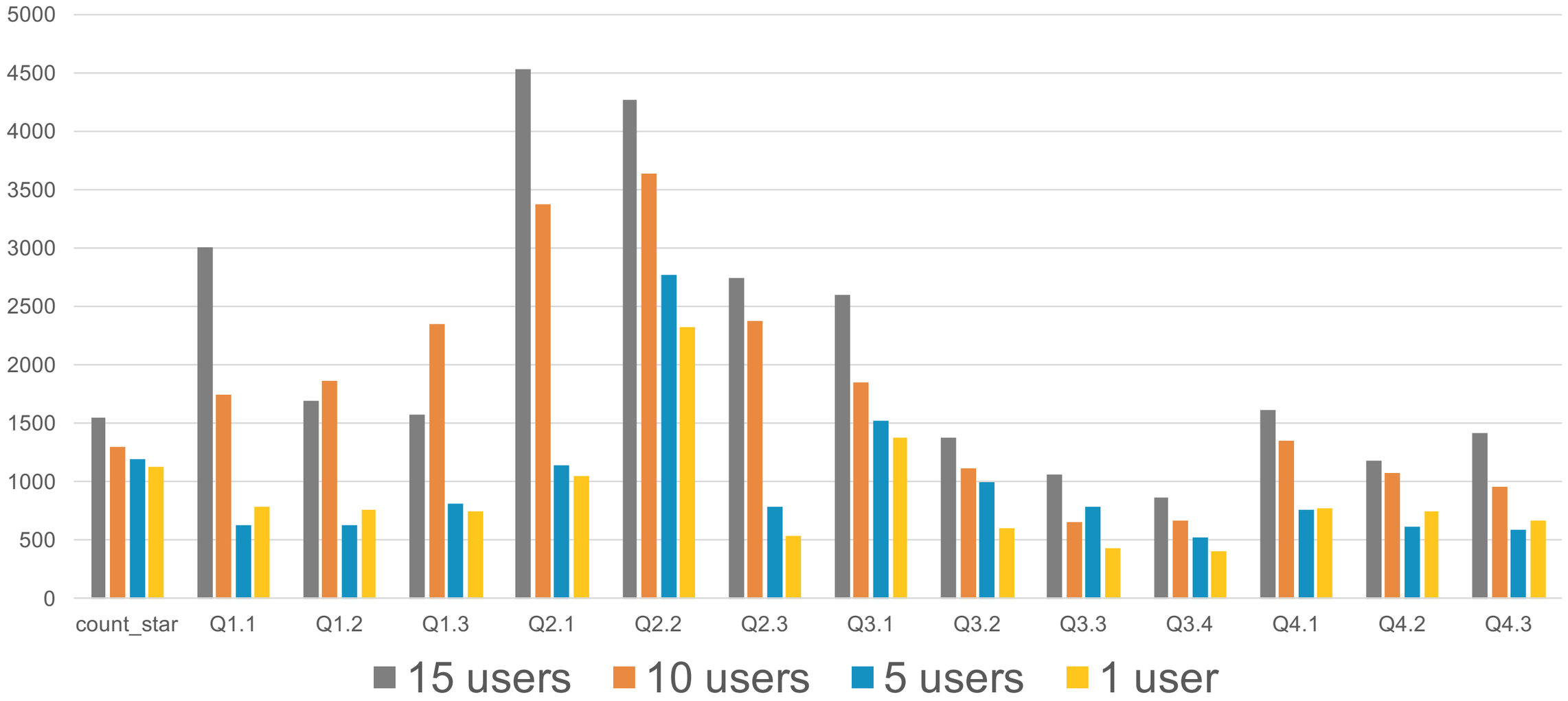
1 (7)({yearLevel, c_nationLevel,s_nationLevel}, {1992 ≤ yearLevel ≤ 1997, c_nationLevel = ASIA, s_nationLevel = ASIA}, {lo_revenue})
2 (8)(yearLevel, c_cityLevel, s_cityLevel}, {5Y, c_nationLevel = UNITED STATES, s_nationLevel = UNITED STATES}, {lo_revenue})
3 (9)(same as 3.2, {5Y, c_cityLevel = UNITED KI1 OR c_cityLevel = UNITED KI5, s_cityLevel = UNITED KI1 OR s_cityLevel = UNITED KI5}, revenue})
4 (10)({yearmonthLevel, c_cityLevel, s_cityLevel}, {yearmonthLevel = Dec1997, c_cityLevel = UNITED KI1 or c_cityLevel = UNITED KI5, tyLevel = UNITED KI1 or s_cityLevel = UNITED KI5}, {lo_revenue})



Q4.1 (11)({yearLevel, c_nationLevel, mfgrLevel, s_regionLevel}, {c_regionLevel = AMERICA, mfgrLevel = MFGR#1 OR mfgrLevel = MFGR#2, s_region = AMERICA}, {sum_profit})
Q4.2 (12)({yearLevel, c_regionLevel, categoryLevel, s_nationLevel}, {yearLevel = 1997 OR yearLevel = 1998, c_regionLevel = AMERICA, mfgrLevel = MFGR#1 OR mfgrLevel = MFGR#2 , s_regionLevel = AMERICA}, {sum_profit})
Q4.3 (13)({yearLevel, regionLevel, categoryLevel, cityLevel}, {yearLevel = 1997 OR yearLevel = 1998, c_regionLevel = AMERICA, categoryLevel = MFGR#14, s_region = UNITED STATES}, {sum_profit})


Average query response time in ms with 10 nodes cluster
HIVE
DRUID
2500
2000
1500
1000
500
0
Q1.1
Q1.2
Q1.3
Q2.1
Q2.2
Q2.3
Q3.1
Q3.2
Q3.3
Q3.4
Q4.1
Q4.2
Q4.3
count_star
Execution
compile time

Concurrency Test: Average execution time in ms
10 node cluster



- Every user is connecting via different JDBC connection.