SQL-on-Hadoop Tutorial

VLDB 2015

Presenters

Fatma Özcan
IBM Research

IBM Big SQL

Daniel Abadi
Yale University and Teradata

HadoopDB/Hadapt

Ippokratis Pandis
Cloudera

Cloudera Impala

Shivnath Babu
Duke University

Starfish

Why SQL-on-Hadoop?

- People need to process data in parallel
- Hadoop is by far the leading open source parallel data processing platform
- Low costs of HDFS results in heavy usage

➡ **Lots of data in Hadoop with appetite to process it**

MapReduce is **not** the answer

- MapReduce is a powerful primitive to do many kinds of parallel data processing
- BUT
  - Little control of data flow
  - Fault tolerance guarantees not always necessary
  - Simplicity leads to inefficiencies
  - Does not interface with existing analysis software
  - Industry has existing training in SQL

SQL interface for Hadoop critical for mass adoption

The database community knows how to process data

- Decades of research in parallel database systems
  - Efficient data flow
  - Load balancing in the face of skew
  - Query optimization
  - Vectorized processing
  - Dynamic compilation of query operators
  - Co-processing of queries

➡ **Massive talent war between SQL-on-Hadoop companies for members of database community**

SQL-on-Hadoop is not a direct implementation of parallel DBMSs

- Little control of storage
  - Most deployments must be over HDFS
    - Append-only file system
  - Must support many different storage formats
    - Avro, Parquet, RCFiles, ORC, Sequence Files
- Little control of metadata management
  - Optimizer may have limited access to statistics
- Little control of resource management
  - YARN still in its infancy

SQL-on-Hadoop is not a direct implementation of parallel DBMSs

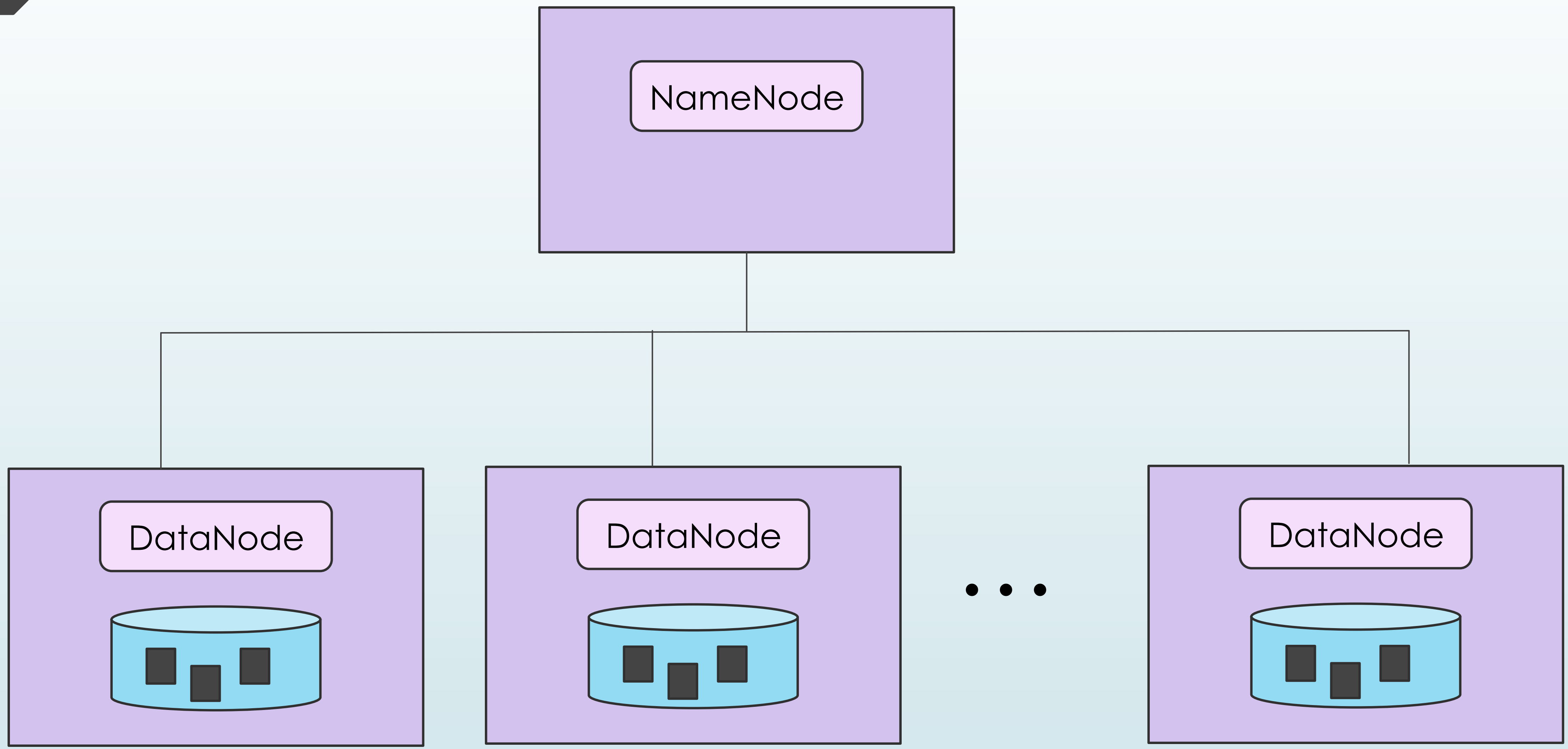
- Hadoop often used a data dump (swamp?)
  - Data often unclean, irregular, and unreliable
- Data not necessarily relational
  - HDFS does not enforce structure in the data
  - Nested data stored as JSON extremely popular
- Scale larger than previous generation parallel database systems
  - Fault tolerance vs. query performance
- Most Hadoop components written in Java
- Want to play nicely with the entire Hadoop ecosystem

Outline of Tutorial

- This session [13:30-15:00]
  - SQL-on-Hadoop Technologies
    - Storage
    - Run-time engine
    - Query optimization
  - Q&A
- Second Session [15:30-17:00]
  - SQL-on-Hadoop examples
    - HadoopDB/Hadapt
    - Presto
    - Impala
    - BigSQL
    - SparkSQL
    - Phoenix/Spice Machine
  - Research directions
  - Q&A

Storage

Quick Look at HDFS



HDFS is

- Good for
  - Storing large files
  - Write once and read many times
  - “Cheap” commodity hardware
- Not good for
  - Low-latency reads
    - Short-circuit reads and HDFS caching help
  - Large amounts of small files
  - Multiple writers

In-situ Data Processing

- HDFS as the data dump
  - Store the data first, figure out what to do later
- Most data arrive in text format
  - Transform, cleanse the data
  - Create data marts in columnar formats
- Lost of nested, JSON data
- Some SQL in data transformations, but mostly other languages, such as Pig, Cascading, etc..
- Columnar formats are good for analytics

SQL-on-Hadoop according to storage formats

- Most SQL-on-Hadoop systems do not control or own the data
  - Hive, Impala, Presto, Big SQL, Spark SQL, Drill
- Other SQL-on-Hadoop systems tolerate HDFS data, but work better with their own proprietary storage
  - HadoopDB/Hadapt
  - HAWQ, Actian Vortex, and HP Vertica

Query Processors with HDFS Native Formats

- Only support native Hadoop formats with open-source reader/writers
- Any Hadoop tool can generate their data
  - Pig, Cascading and other ETL tools
- They are more of a query processor than a database
- Indexing is a challenge !!
- No co-location of multiple tables
  - Due to HDFS

Systems with Proprietary Formats

- Almost all exploit some existing database systems
- They store their own binary format on HDFS
- Hadapt stores the data in a single node database, like postgres
  - Can exploit Postgres indexes
- HAWQ, Actian, HP Vertica, and Hadapt all control how tables are partitioned, and can support co-located joins

HDFS Native Formats

- CSV files are most common for ETL-like workloads
- Lots of nested and complex data
  - Arrays, structs, maps, collections
- Two major columnar formats
  - ORCFile
  - Parquet
- Data serialization
  - JSON and Avro
  - Protocol buffers and Thrift

Parquet

- PAX format, supporting nested data
- Idea came from the Google's Dremel System
- Major contributors: Twitter & Cloudera
- Provides dictionary encoding and several compressions
- Preffered format for Impala, IBM Big SQL, and Drill
- Can use Thrift or Avro to describe the schema

Columnar storage

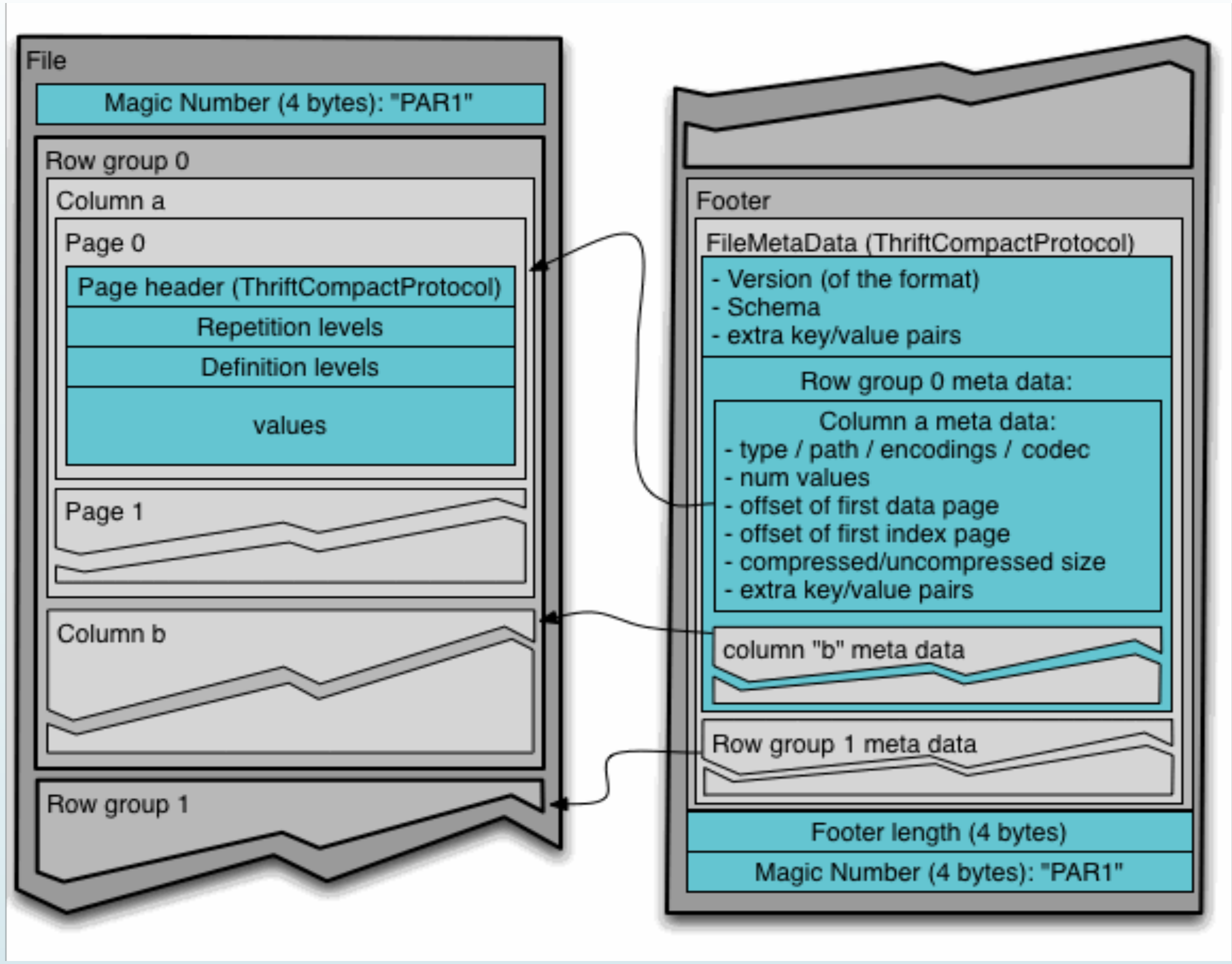
- Fast compression
- Schema projection
- Efficient encoding

Nested data

- A natural schema
- Flexible
- Less duplication applying denormalization

Parquet, cont.

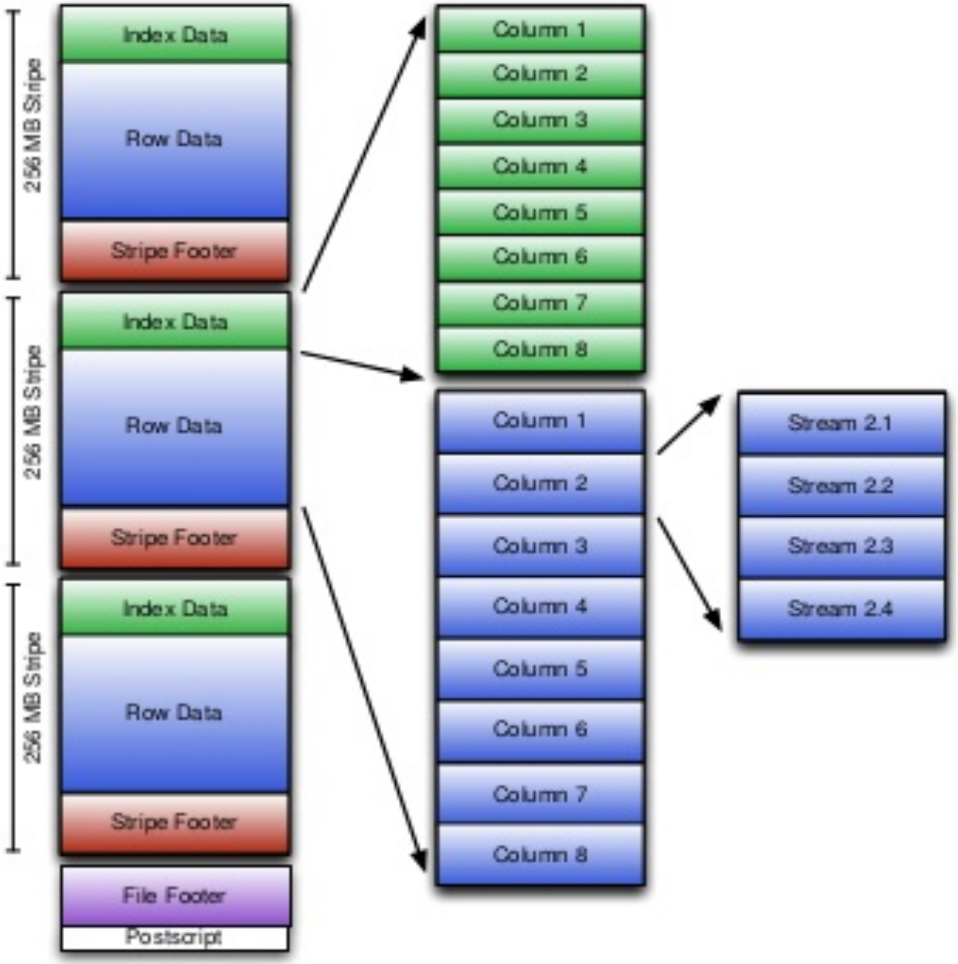
- A table with N columns is split into M row groups.
- The file metadata contains the locations of all the column metadata start locations.
- Metadata is written after the data to allow for single pass writing.
- There are three types of metadata: file metadata, column (chunk) metadata and page header metadata.
- Row group metadata includes
  - Min-max values for skipping



ORCFile

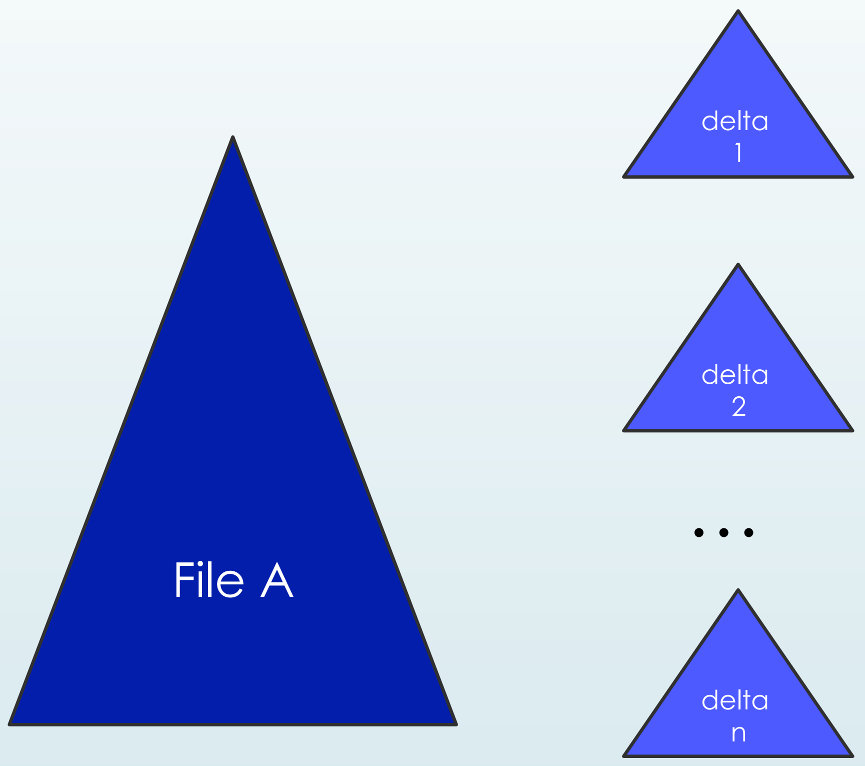
- Second generation, following RC file
- PAX formats with all data in a single file
- Hortonworks is the major contributor, together with Microsoft
- Preferred format for Hive, and Presto
- Supports
  - Dictionary encoding
  - Fast compression
- File, and stripe level metadata
- Stripe indexing for skipping
- Now metadata even includes bloom filters for point query lookups

ORCFile Layout



Handling Updates in HDFS

- No updates in HDFS
- Appends to HDFS files are supported, but not clear how much they are used in production
- Updates are collected in delta files
- At the time of read delta and main files are merged
  - Special inputFormats
- Lazy compaction to merge delta files and main files
  - When delta files reach a certain size
  - Scheduled intervals



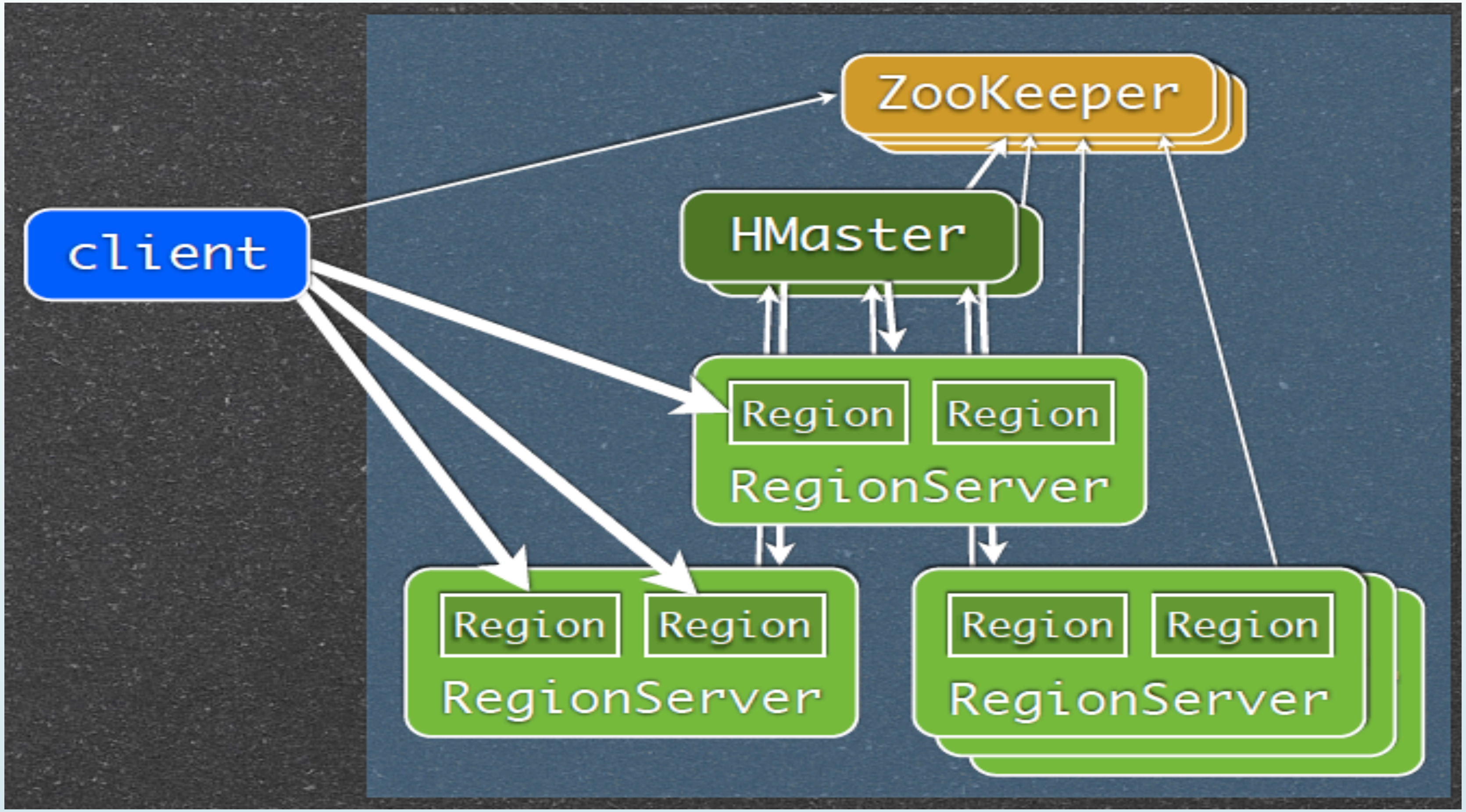
SQL on NoSQL!

- Put a NoSQL solution on top of HDFS
  - For the record, you can avoid HDFS completely
  - But, this is a SQL-on-Hadoop tutorial
- NoSQL solutions can provide CRUD at scale
  - CRUD = Create, Read, Update, Delete
- And, then run SQL on it?
- Sounds crazy? Well, lets see

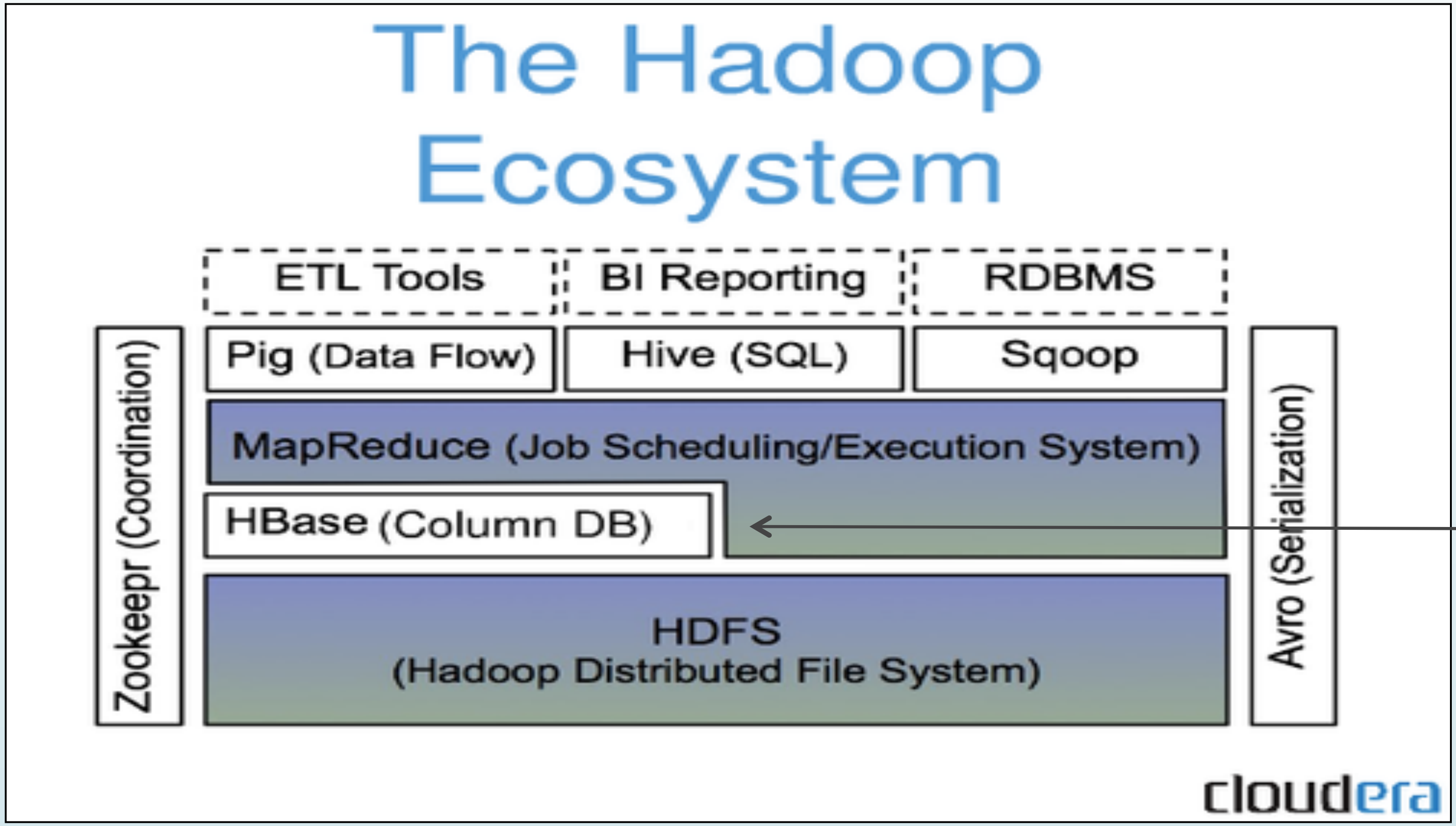
HBase: The Hadoop Database

- Not HadoopDB, which we will see later in the tutorial
- HBase is a data store built on top of HDFS based on Google Bigtable
- Data is logically organized into tables, rows, and columns
  - Although, Key-Value storage principles are used at multiple points in the design
  - Columns are organized into Column Families (CF)
- Supports record-level CRUD, record-level lookup, random updates
- Supports latency-sensitive operations

HBase Architecture



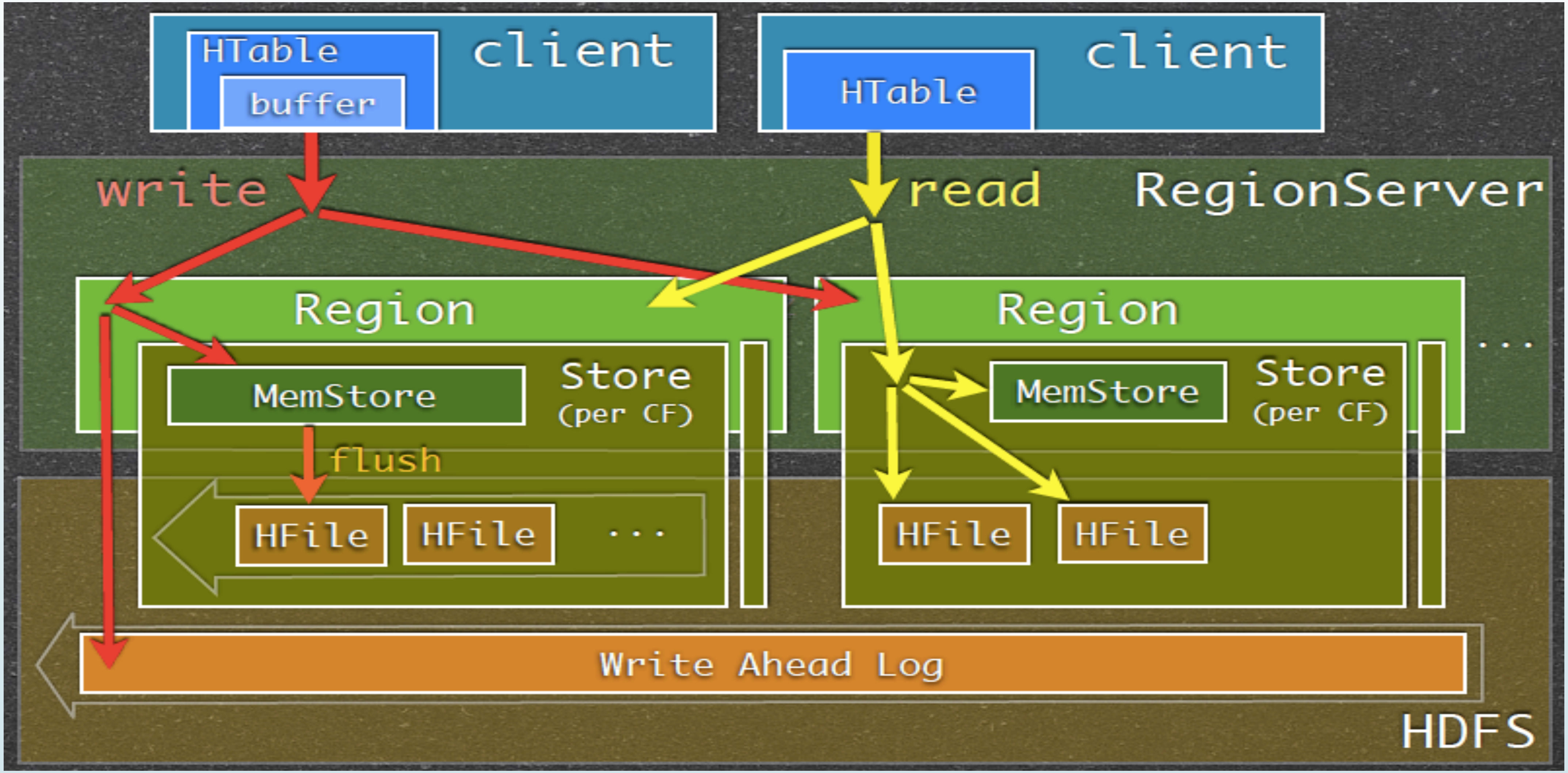
HBase Architecture



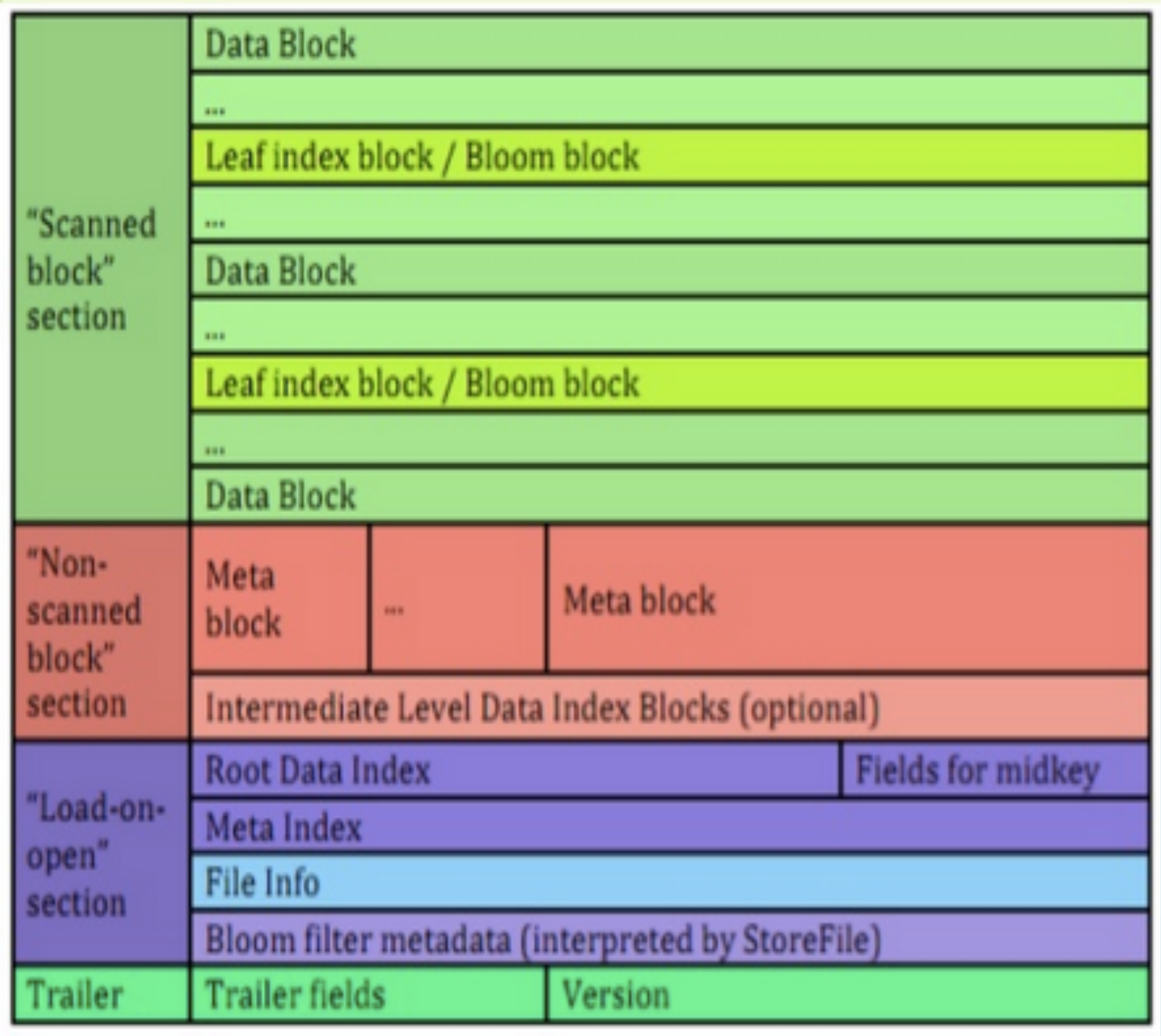
HBase stores three types of files on HDFS:

- **WALs**
- **HFiles**
- **Links**

HBase Read and Write Paths



HFile Format



- **Immutable**
- **Created on flush or compaction**
  - **Sequential writes**
- **Read randomly or sequentially**
- **Data is in blocks**
  - **HFile blocks are not HDFS blocks**
  - **Default data block size == 64K**
  - **Default index block size == 128K**
  - **Default bloom filter block size == 128K**
- **Use smaller block sizes for faster random lookup**
- **Use larger block sizes for faster scans**
- **Compression is recommended**
- **Block encoding is recommended**

Run-time Engine

Design Decisions: Influencers

- Low Latency
- High Throughput
- Degree of tolerance to faults
- Scalability in data size
- Scalability in cluster size
- Resource elasticity
- Multi-tenancy
- Ease of installation in existing environments

Accepted across SQL-on-Hadoop Solutions

- Push computation to data
- Columnar data formats
- Vectorization
- Support for multiple data formats
- Support for UDFs

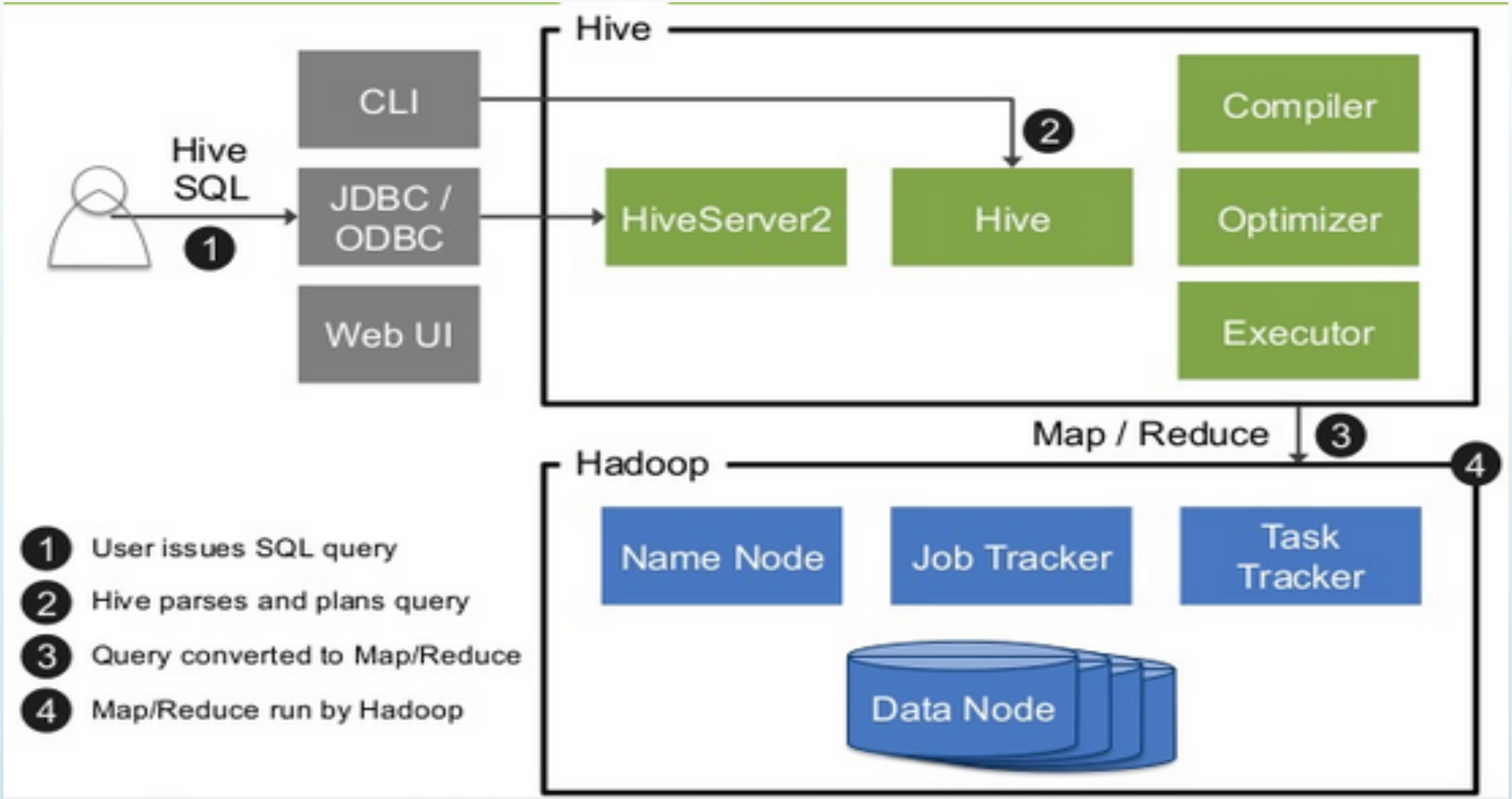
Differences across SQL-on-Hadoop Solutions

- What is the Lowest Common Execution Unit
- Use of Push Vs. Pull
- On the JVM or not
- Fault tolerance: Intra-query or inter-query
- Support for multi-tenancy

SQL on MapReduce

- Hive
- Tenzing

Hive



Hive
CLI
Hive
SQL
JDBC /
ODBC
Web UI
1
2
HiveServer2
Hive
Compiler
Optimizer
Executor
Map / Reduce
3
Hadoop
4
Name Node
Job Tracker
Task
Tracker
Data Node
1 User issues SQL query
2 Hive parses and plans query
3 Query converted to Map/Reduce
4 Map/Reduce run by Hadoop

Example: Joins in MapReduce

customer

first	last	id
Nick	Toner	11911
Jessie	Simonds	11912
Kasi	Lamers	11913
Rodger	Clayton	11914
Verona	Hollen	11915

order

cid	price	quantity
4150	10.50	3
11914	12.25	27
3491	5.99	5
2934	39.99	22
11914	40.50	10

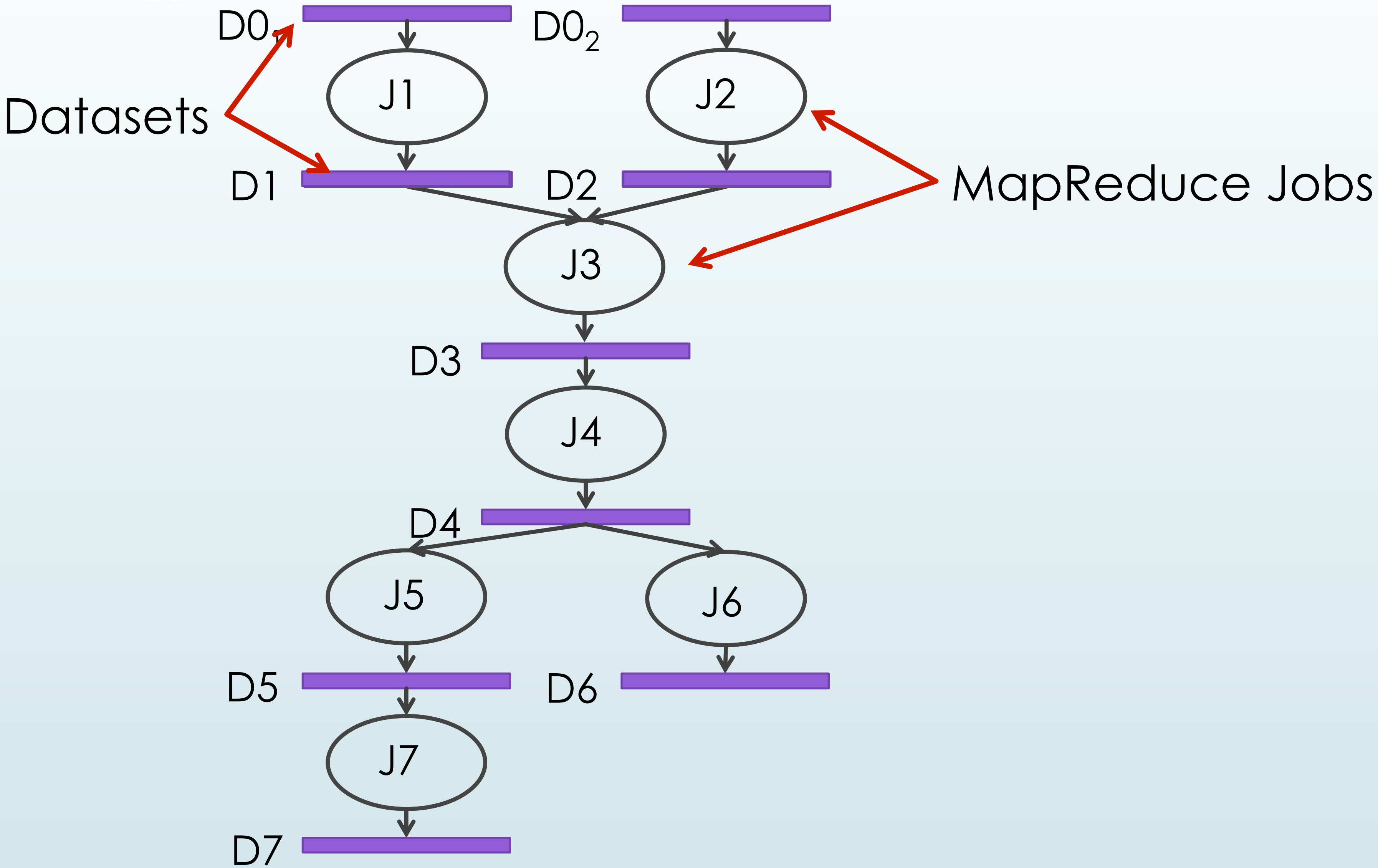
```
SELECT * FROM customer join order ON customer.id = order.cid;
```

Identical keys shuffled to the same reducer. Join done reduce-side.

Limitations

- Having a MapReduce Job as the Lowest Execution Unit quickly becomes restrictive
- Query execution plans become MapReduce workflows

MapReduce Workflows



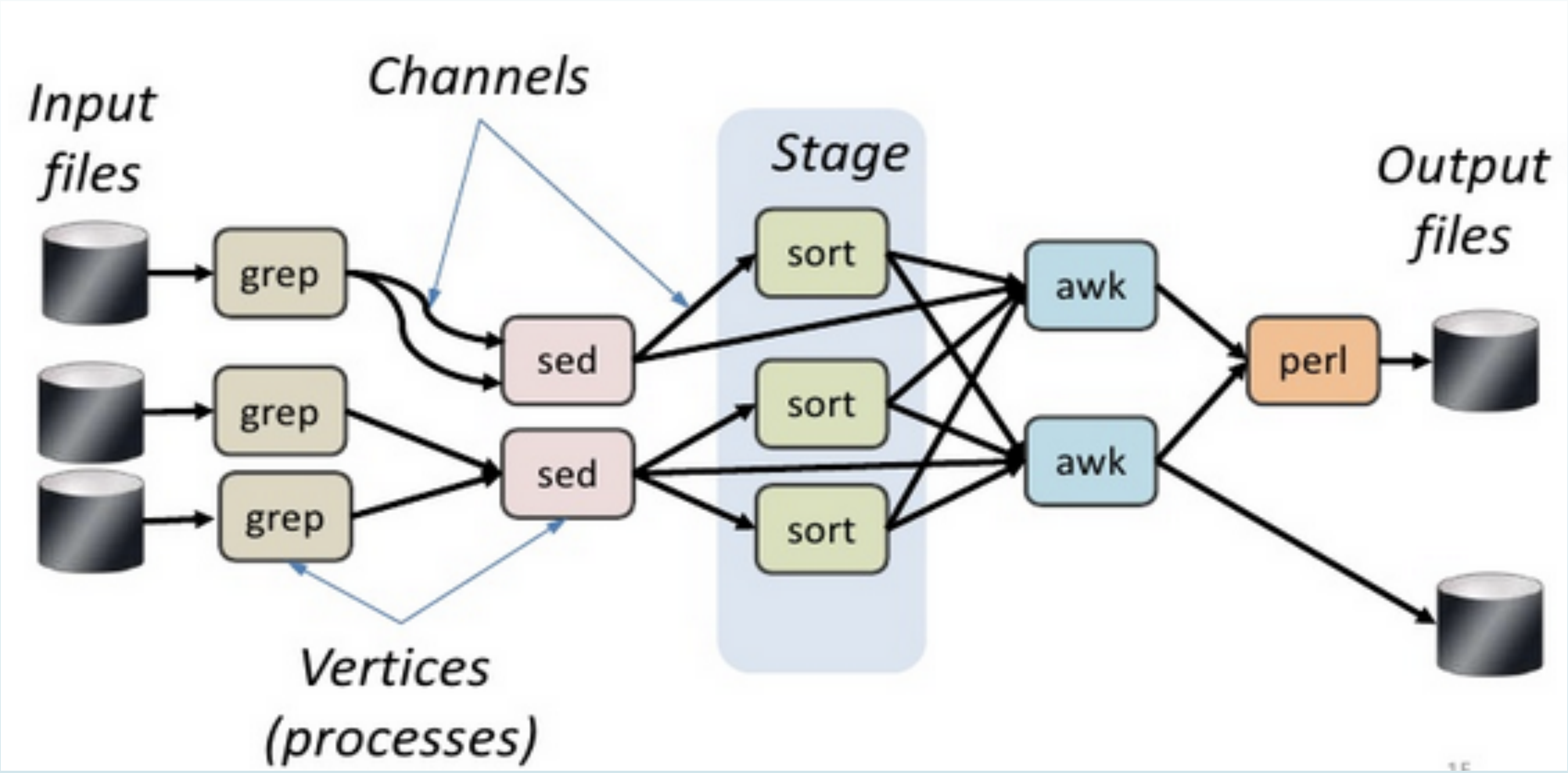
Research Done to Address these Limitations

- On efficient joins in the MapReduce paradigm
- On reducing the number of MapReduce jobs by packing/collapsing the MapReduce workflow
  - Horizontally
    - Shared scans
  - Vertically
    - Making using of static and dynamic partitioning
- On efficient management of intermediate data

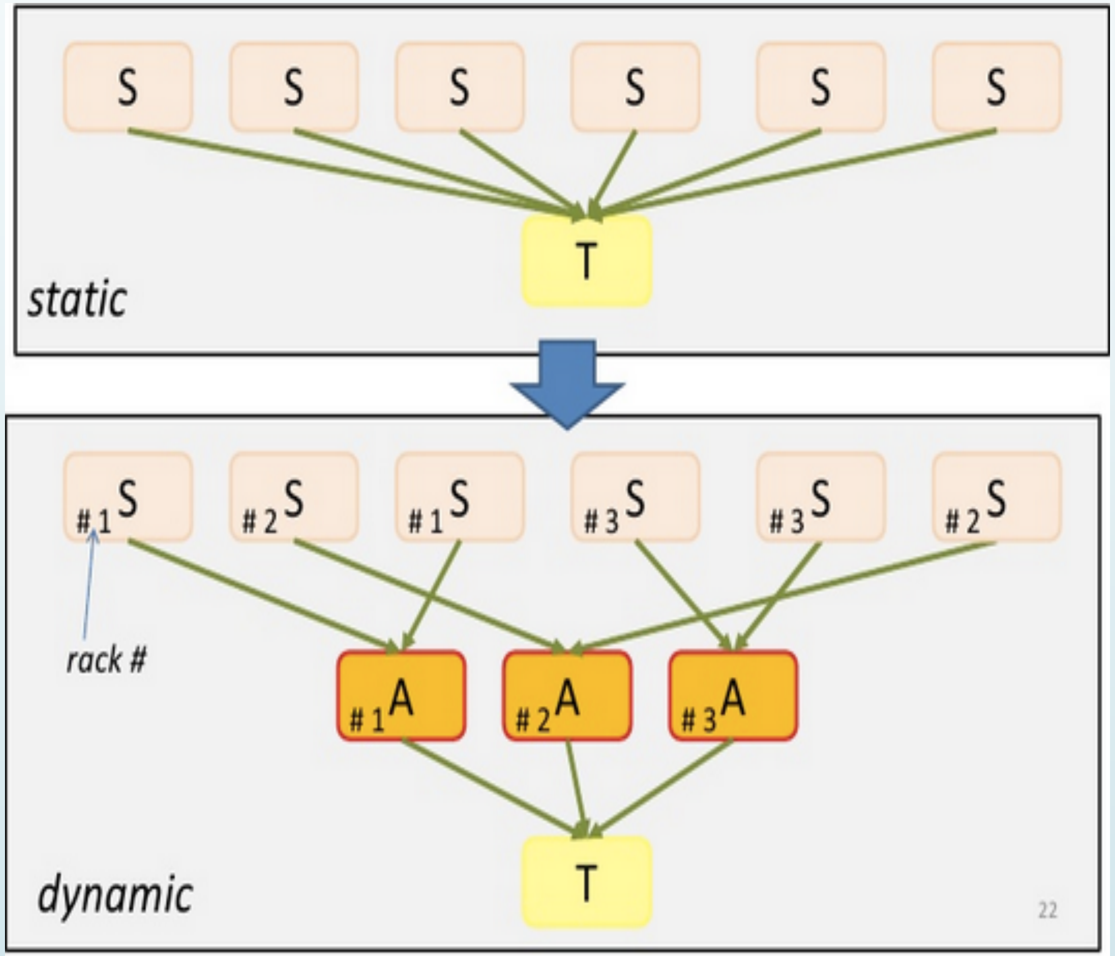
From MapReduce to DAGs

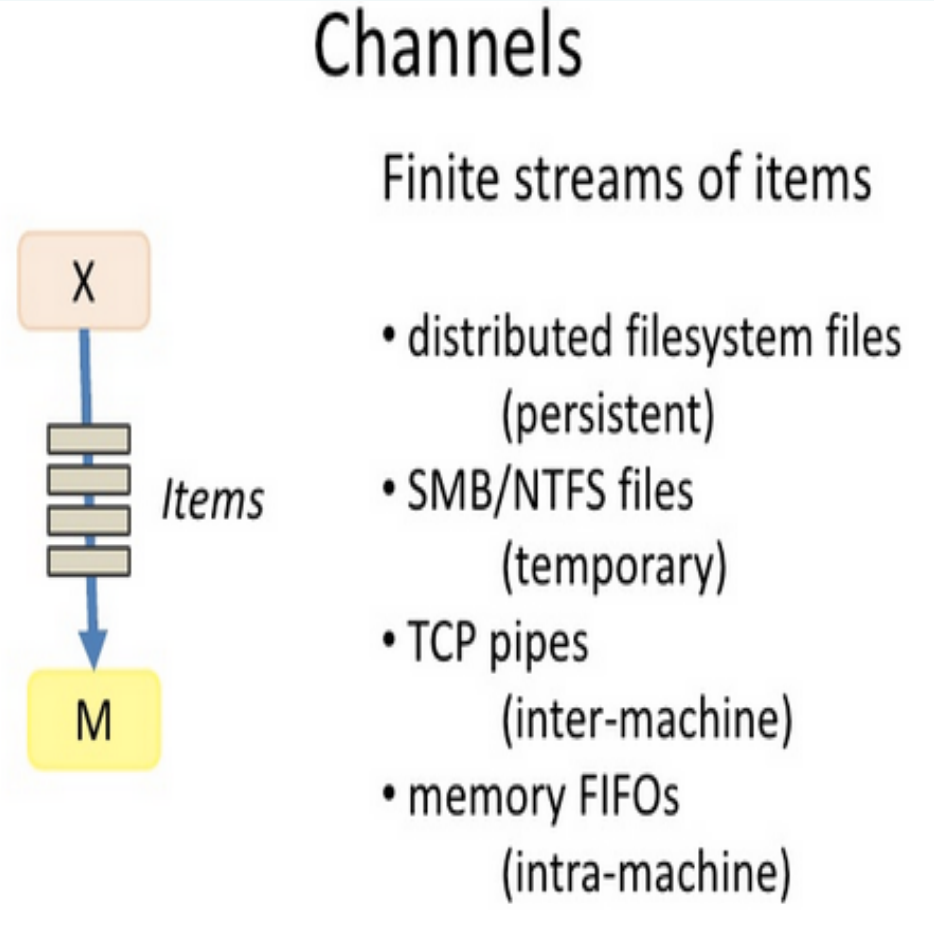
- Dryad
- Tez

Dryad: Dataflows as First-class Citizens



Smart DAG Execution in Dryad





Tez: Inspired by Dryad and Powered by YARN

Hive / HIVE-4660

Let there be Tez

Agile Board

Tez is a new application framework built on Hadoop Yarn that can execute complex directed acyclic graphs of general data processing tasks. Here's the project's page: http://incubator.apache.org/projects/tez.html

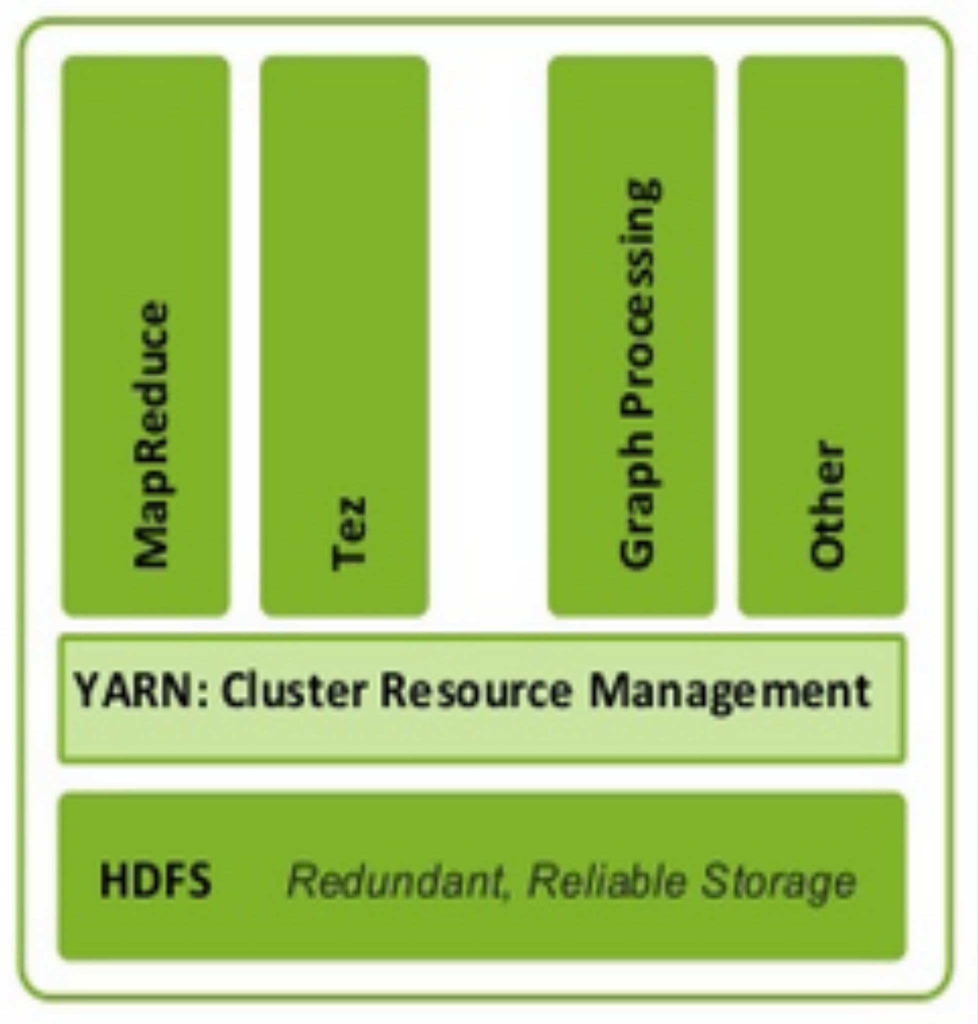
The interesting thing about Tez from Hive's perspective is that it will over time allow us to overcome inefficiencies in query processing due to having to express every algorithm in the map-reduce paradigm.

The barrier to entry is pretty low as well: Tez can actually run unmodified MR jobs; But as a first step we can without much trouble start using more of Tez' features by taking advantage of the MRR pattern.

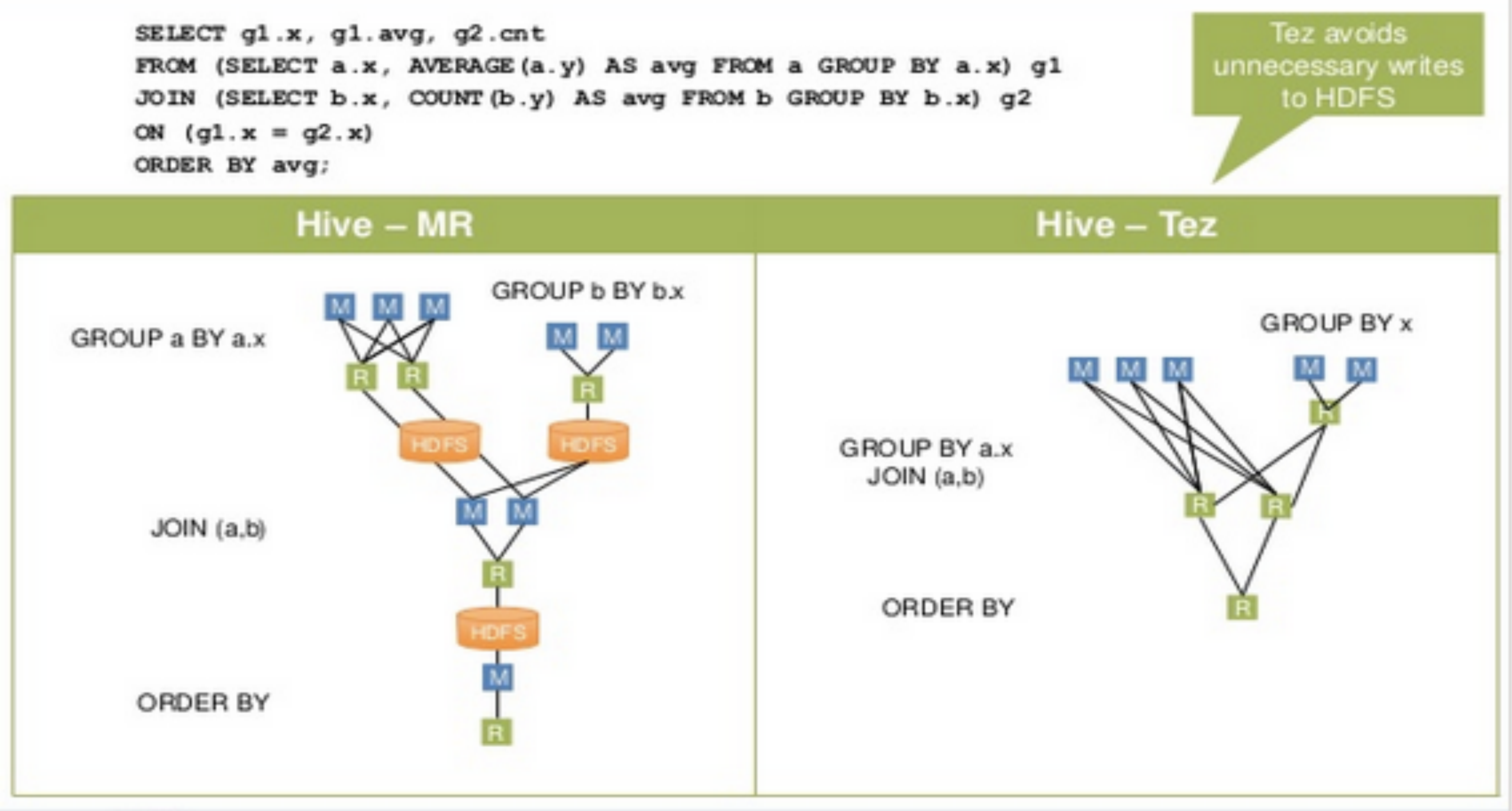
MRR simply means that there can be any number of reduce stages following a single map stage - without having to write intermediate results to HDFS and re-read them in a new job. This is common when queries require multiple shuffles on keys without correlation (e.g.: join - grp by - window function - order by)

Quick Detour on YARN

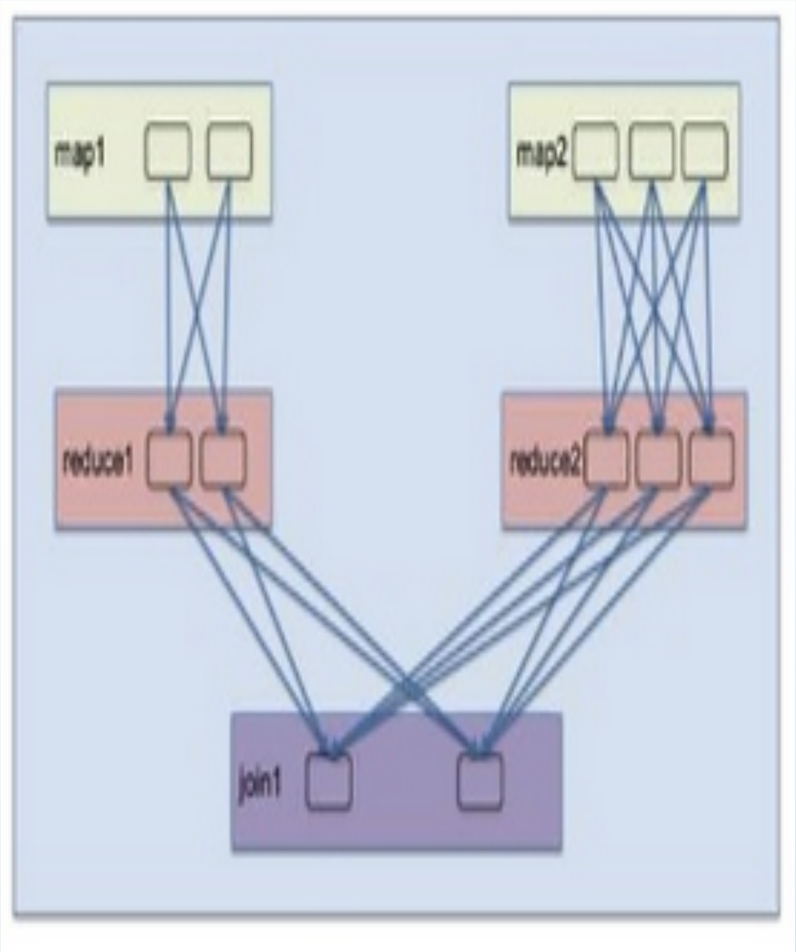
- The Hadoop Community realized that MapReduce cannot be the Lowest Execution Unit for all data apps
- Separated out the resource management aspects from application management
- YARN is best seen as an Operating System for Data Processing Apps
  - Recall the 80s: Databases and Operating Systems: Friends or Foes?

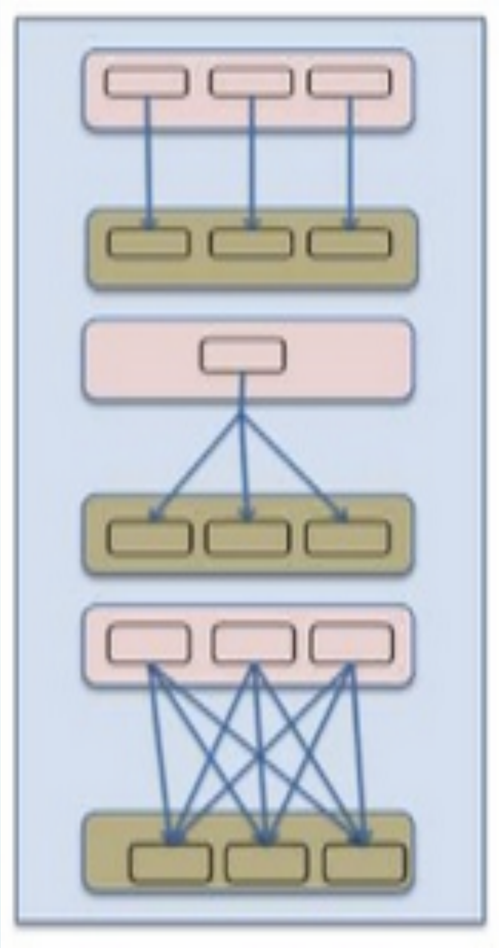


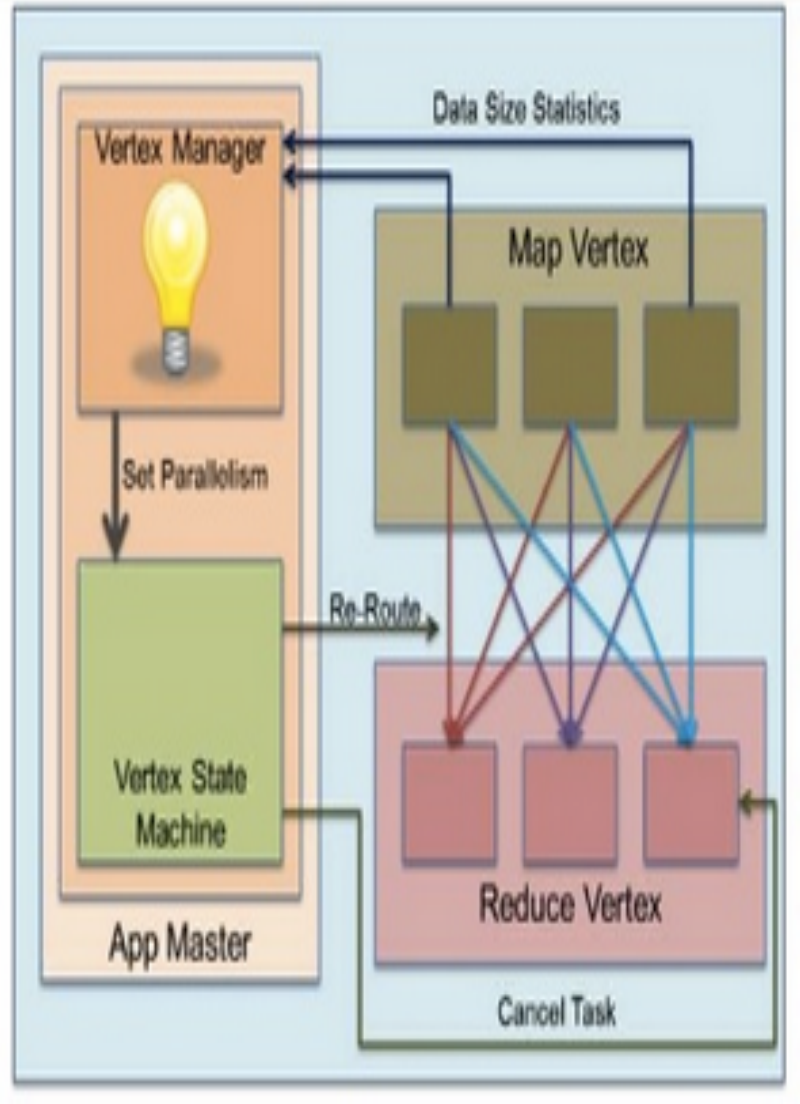
An Example of What Tez Enables



A Tez Slide on Tez

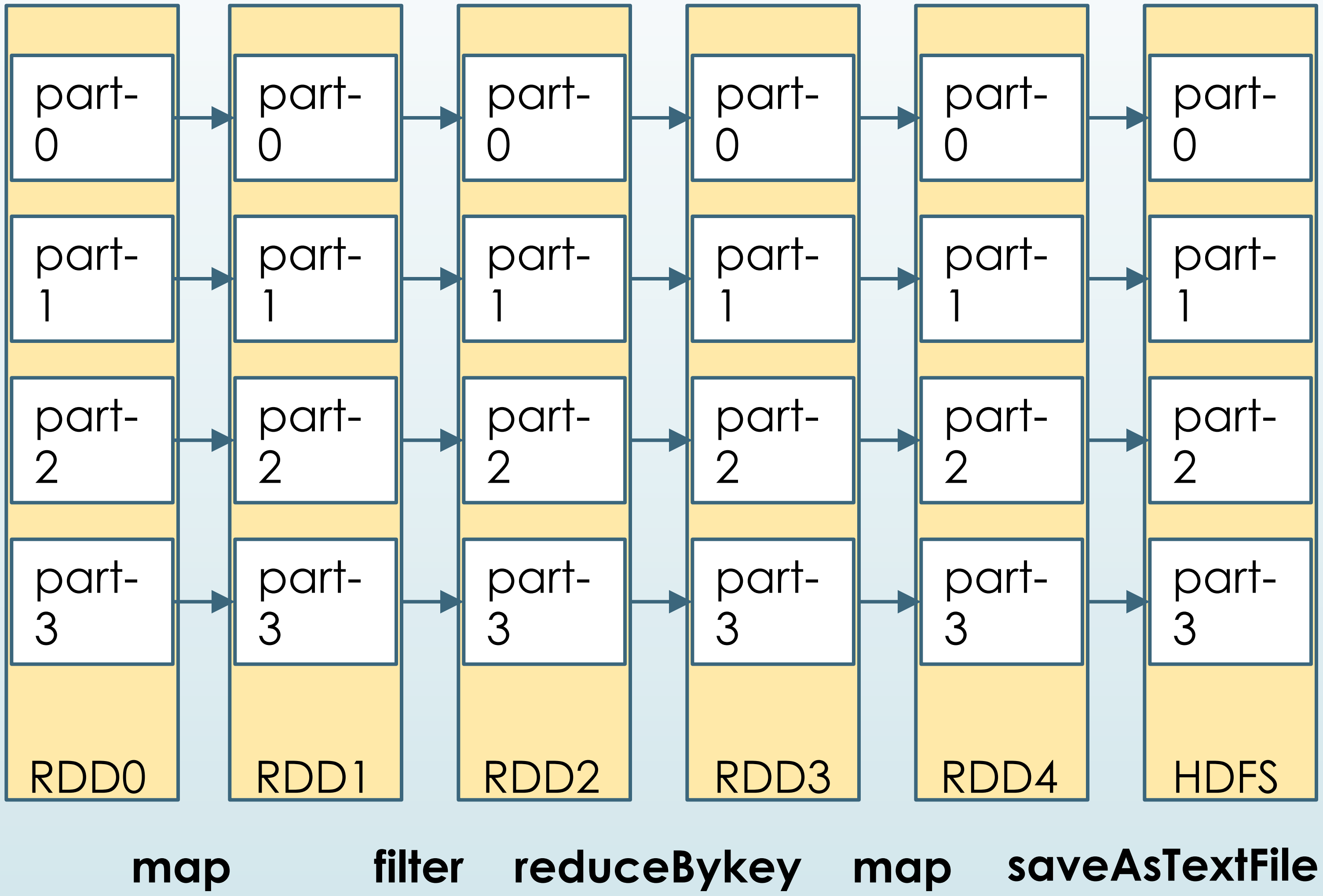






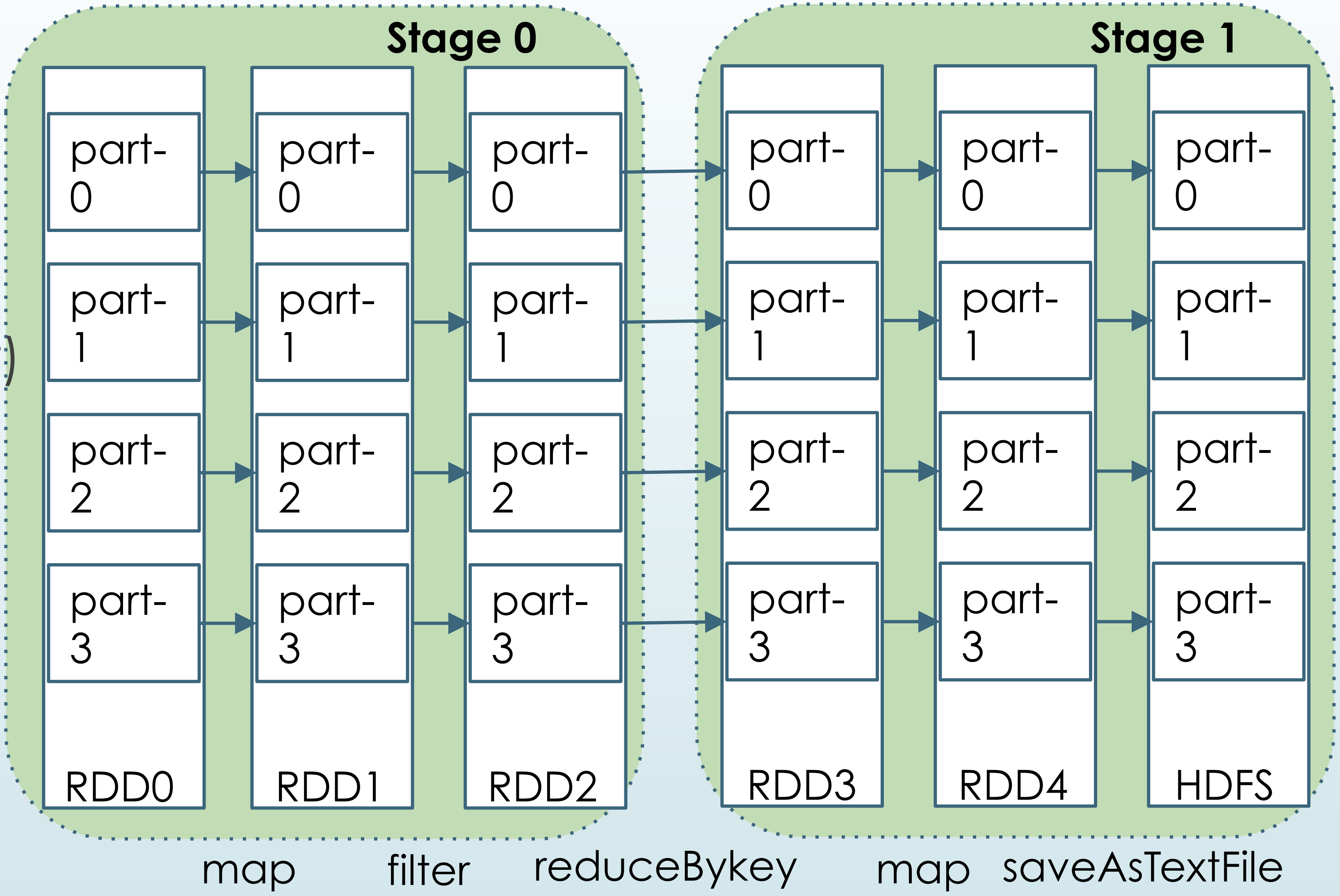
Spark: A Different Way to Look at a Dataflow

```
sc.textFile(hdfsPath)
    .map(parseInput)
    .filter(subThreshold)
    .reduceByKey(tallyCount)
    .map(formatOutput)
    .saveAsTextFile(outPath)
```



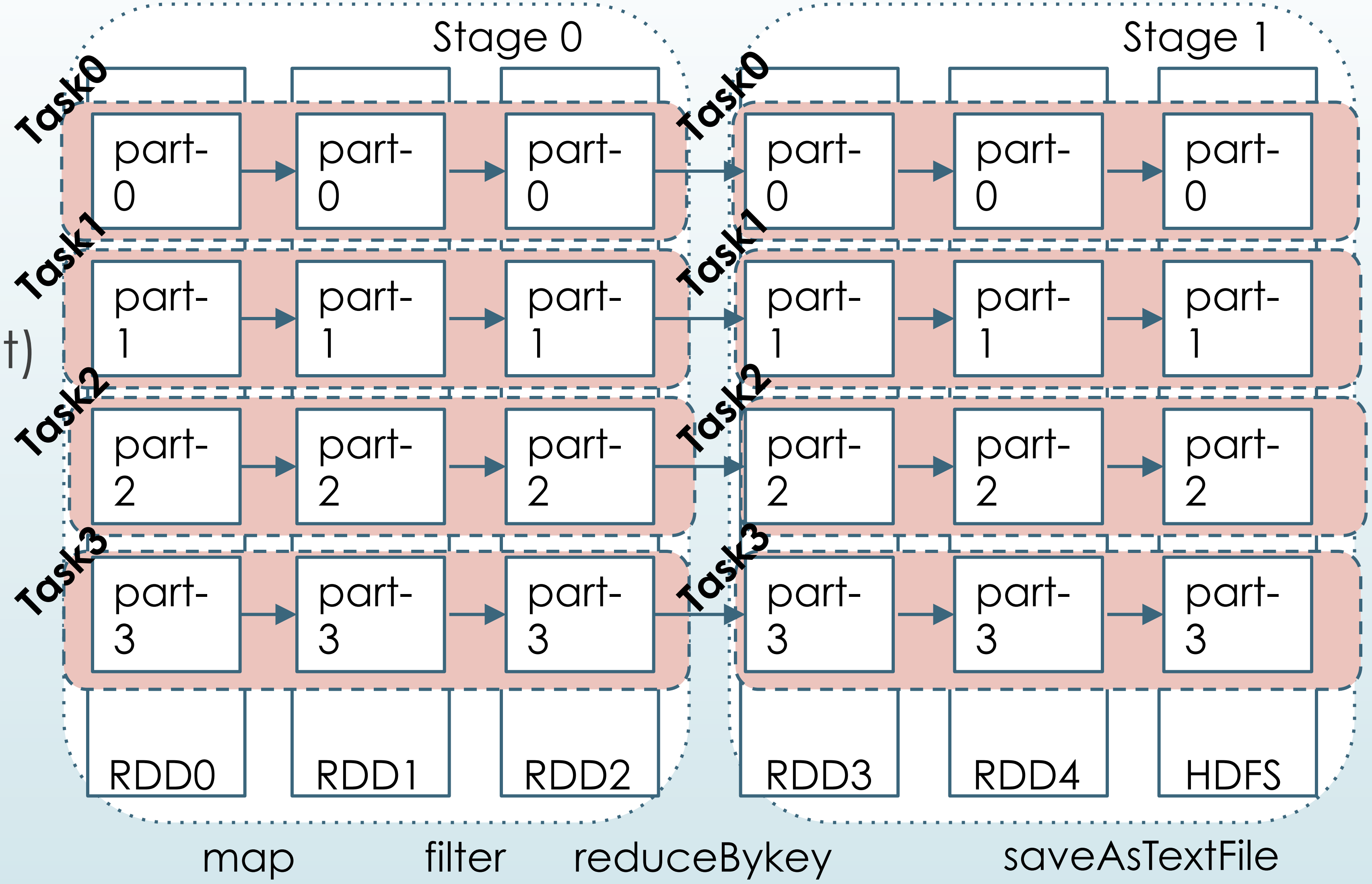
Spark: A Different Way to Look at a Dataflow

```
sc.textFile(hdfsPath)
    .map(parseInput)
    .filter(subThreshold)
    .reduceByKey(tallyCount)
    .map(formatOutput)
    .saveAsTextFile(outPath)
```



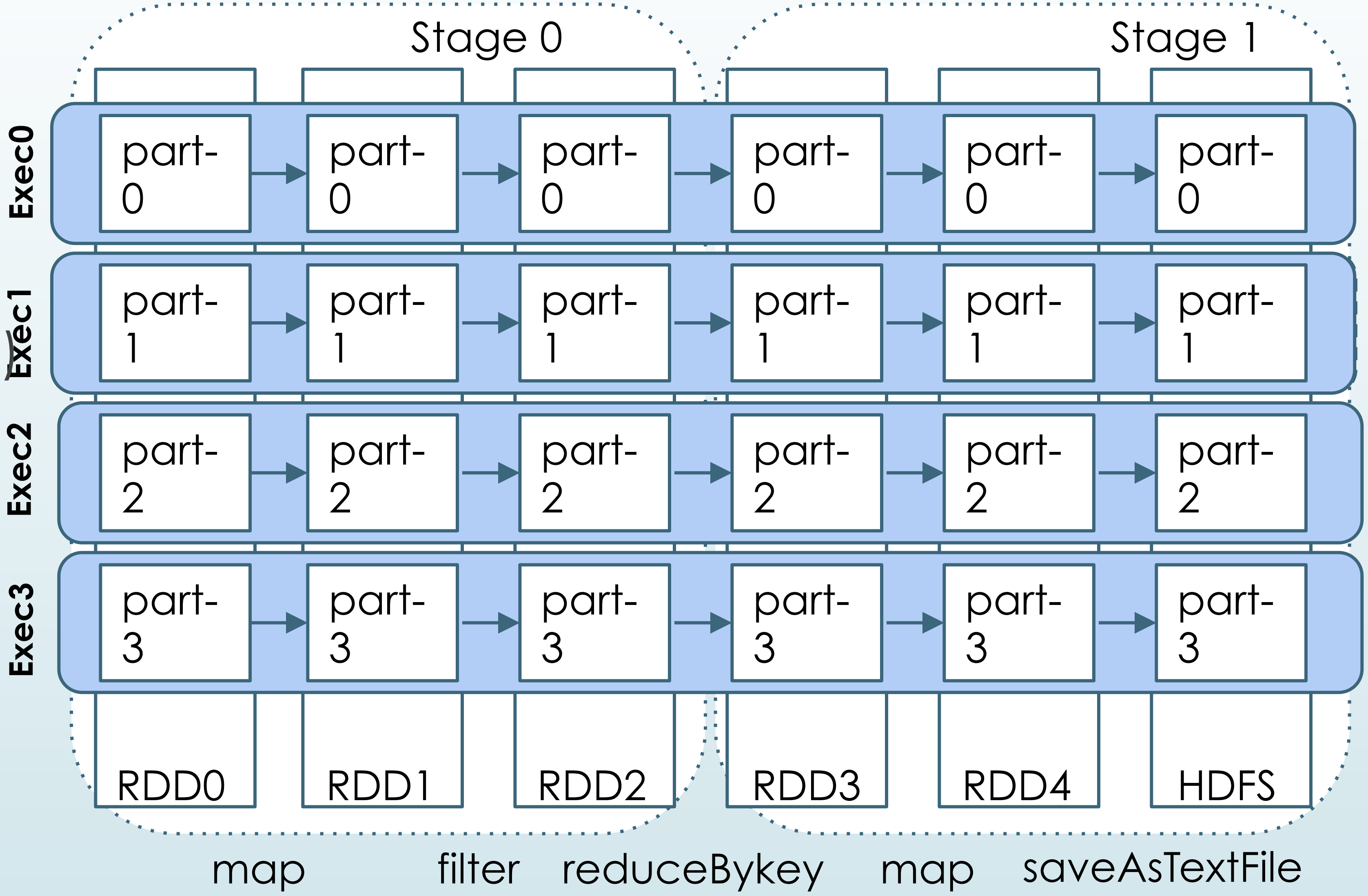
Spark: A Different Way to Look at a Dataflow

```
sc.textFile(hdfsPath)
  .map(parseInput)
  .filter(subThreshold)
  .reduceByKey(tallyCount)
  .map(formatOutput)
  .saveAsTextFile(outPath)
```

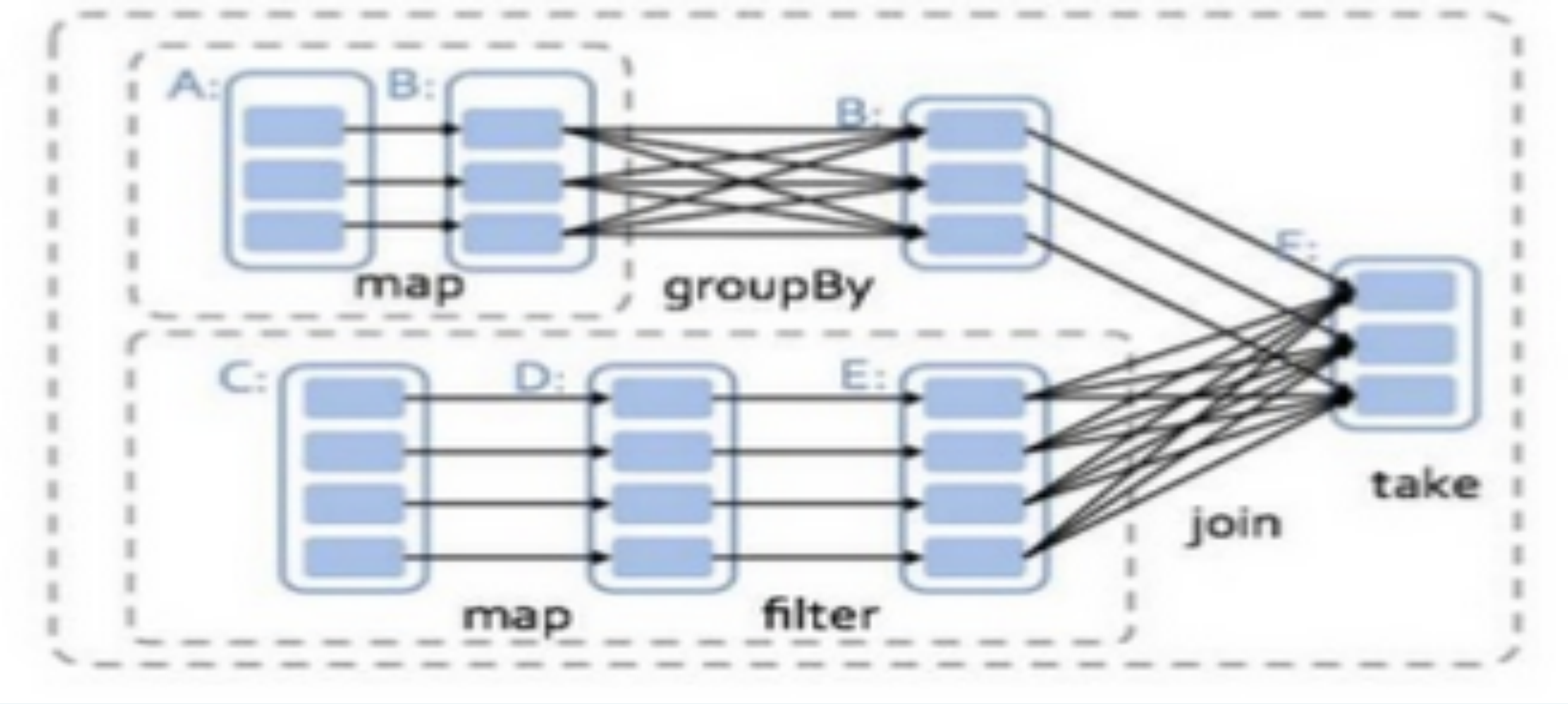


Spark: A Different Way to Look at a Dataflow

```
sc.textFile(hdfsPath)
   .map(parseInput)
   .filter(subThreshold)
   .reduceByKey(tallyCount)
   .map(formatOutput)
   .saveAsTextFile(outPath)
```

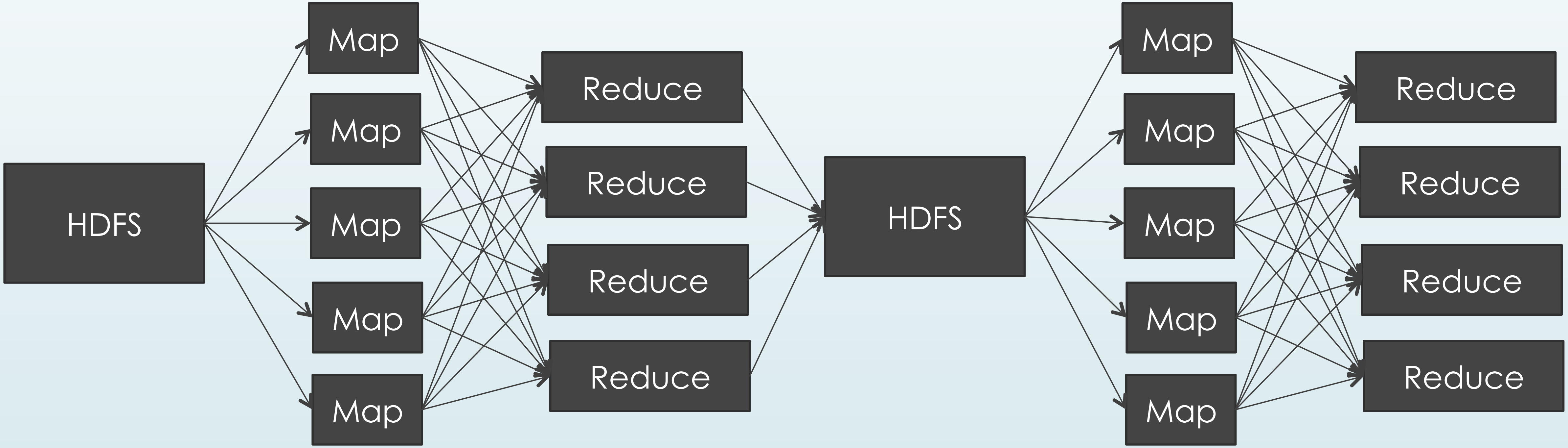


Spark: A Different Way to Look at a Dataflow

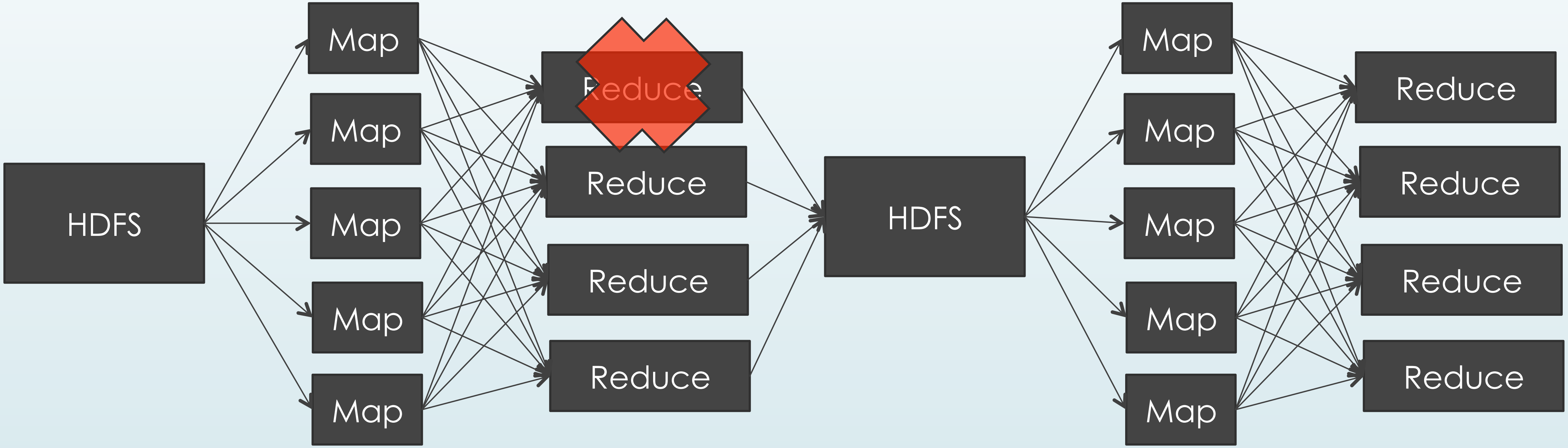


Fault Tolerance

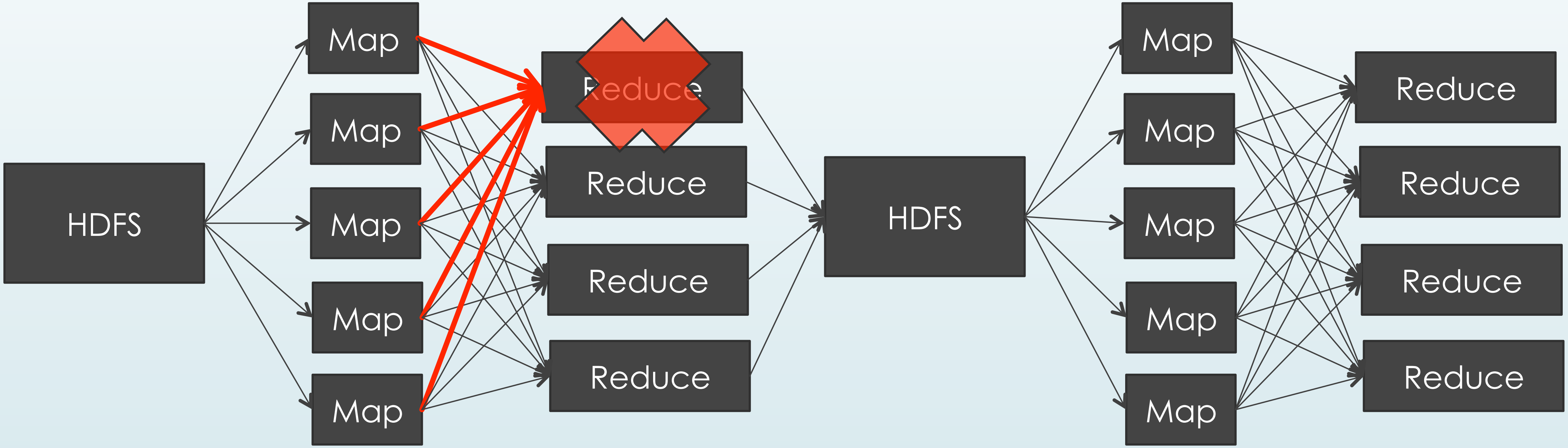
MapReduce Fault Tolerance



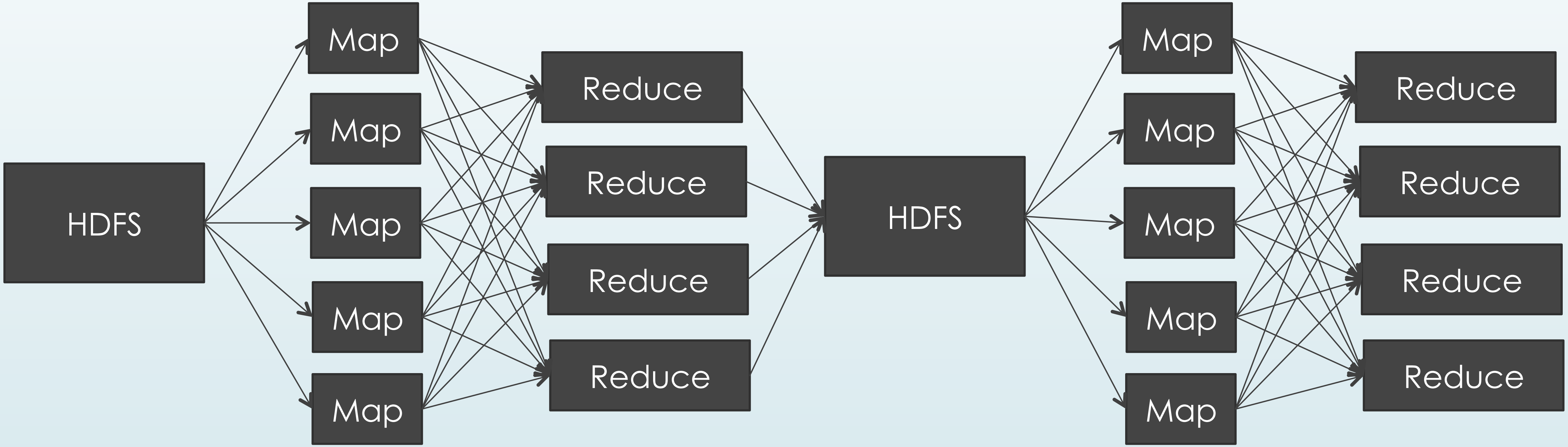
MapReduce Fault Tolerance



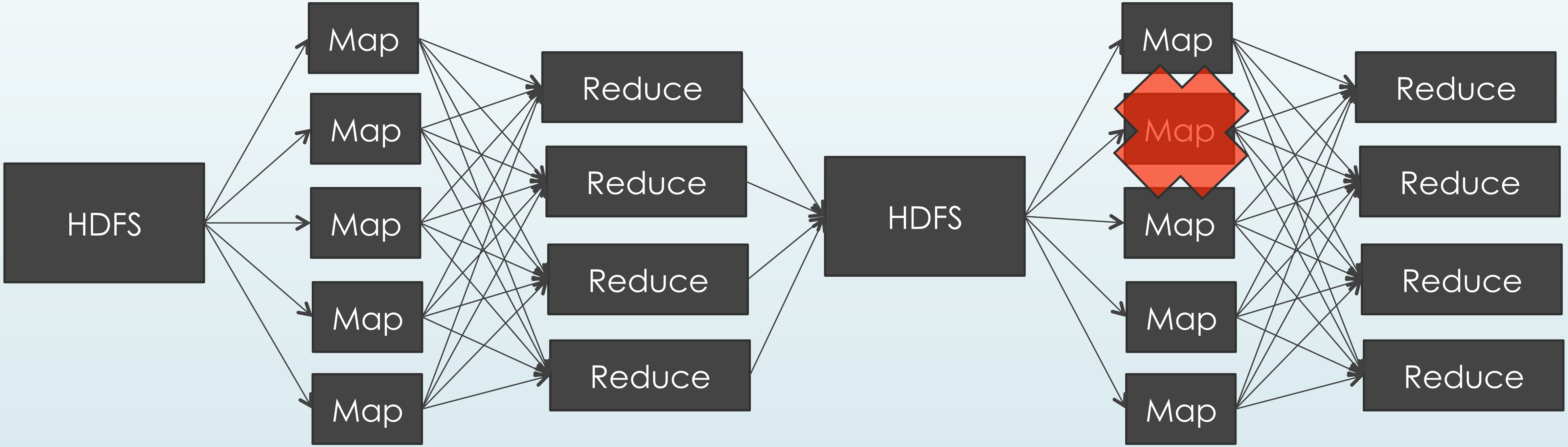
MapReduce Fault Tolerance



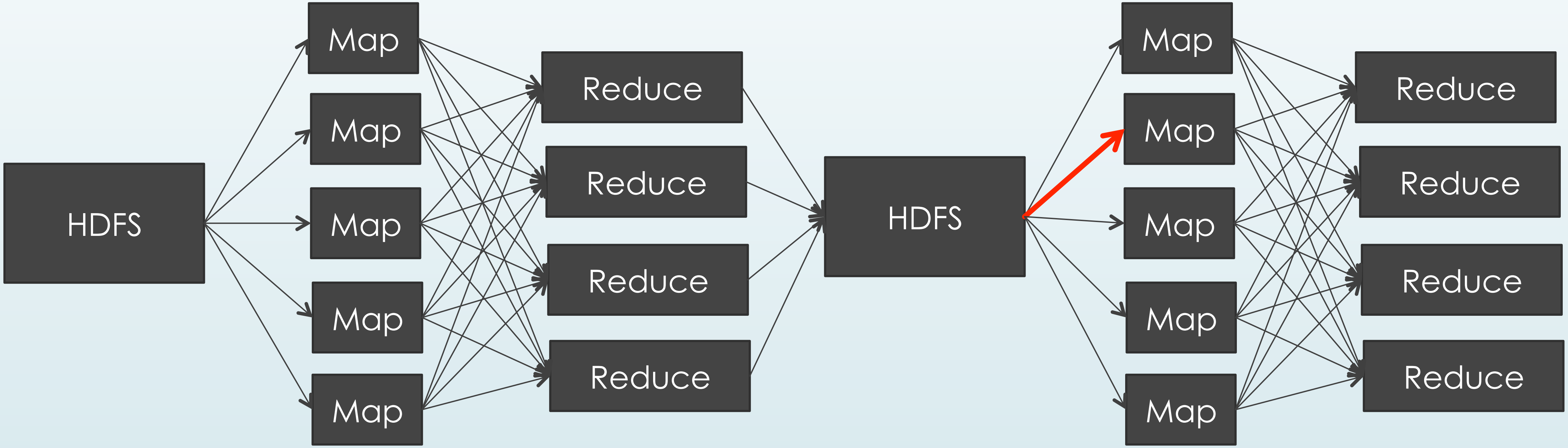
MapReduce Fault Tolerance



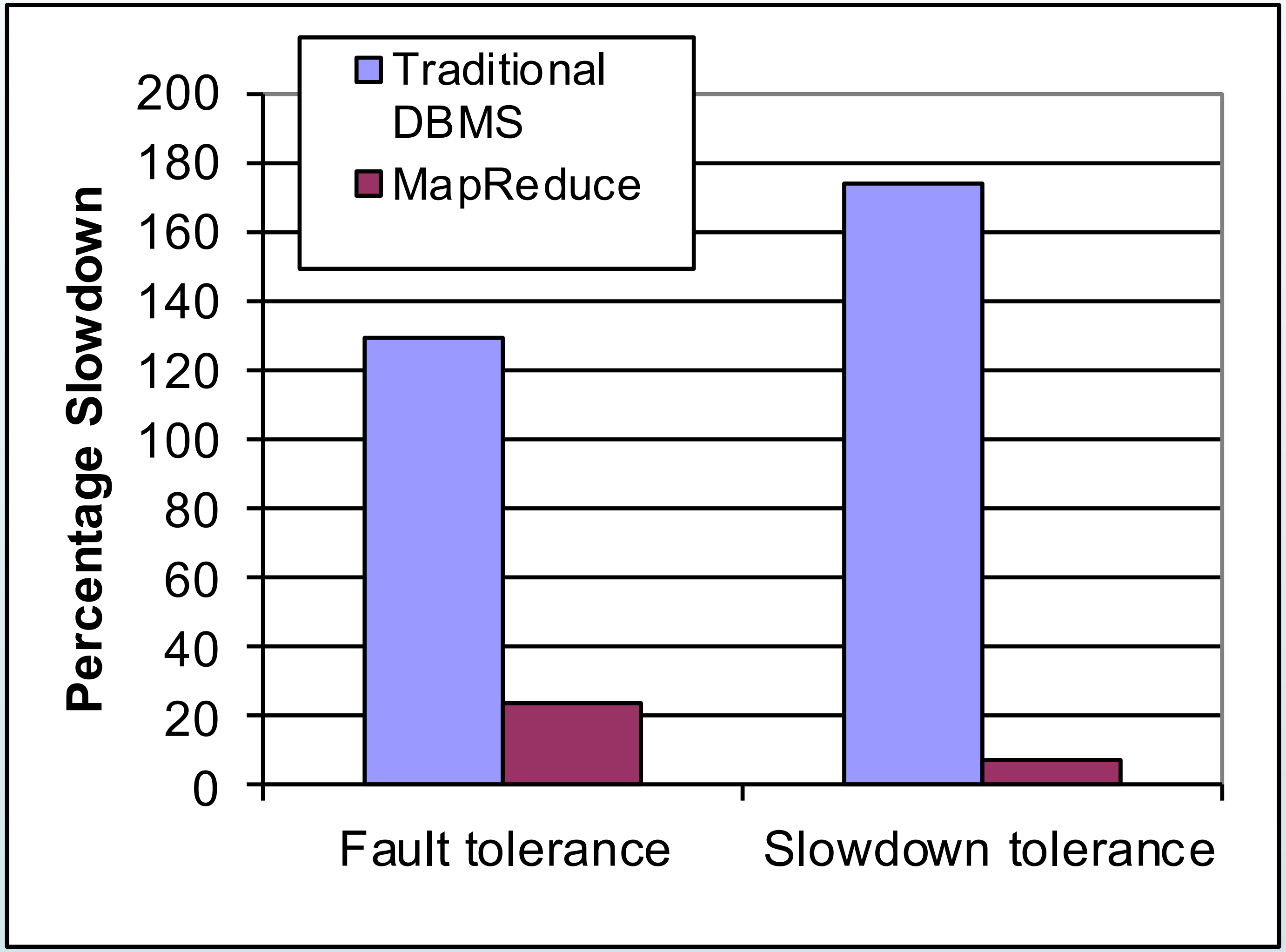
MapReduce Fault Tolerance



MapReduce Fault Tolerance

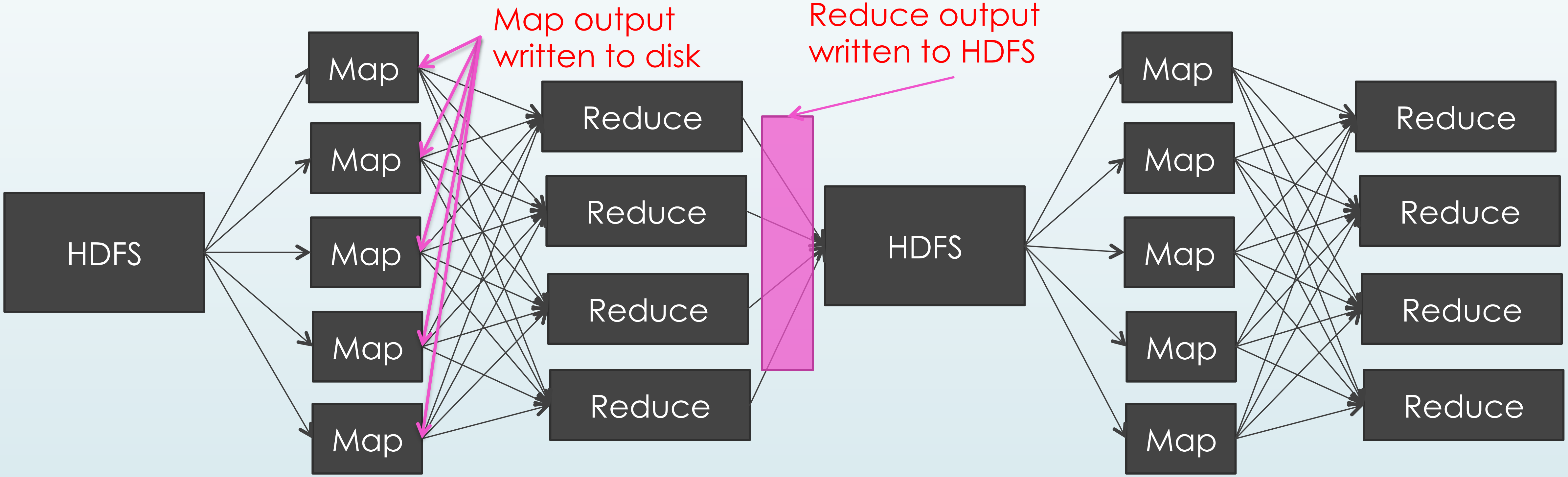


Fault Tolerance



- SELECT sourceIP,
  SUM(adRevenue)
  FROM UserVisits
  GROUP BY sourceIP
- Node fails (or slows down by factor of 2) in the middle of query

Downsides of MapReduce Fault Tolerance



Spark RDDs

- Stores intermediate results in memory rather than disk
  - Advantage: Performance
  - Disadvantage: Memory requirements

Resource Management

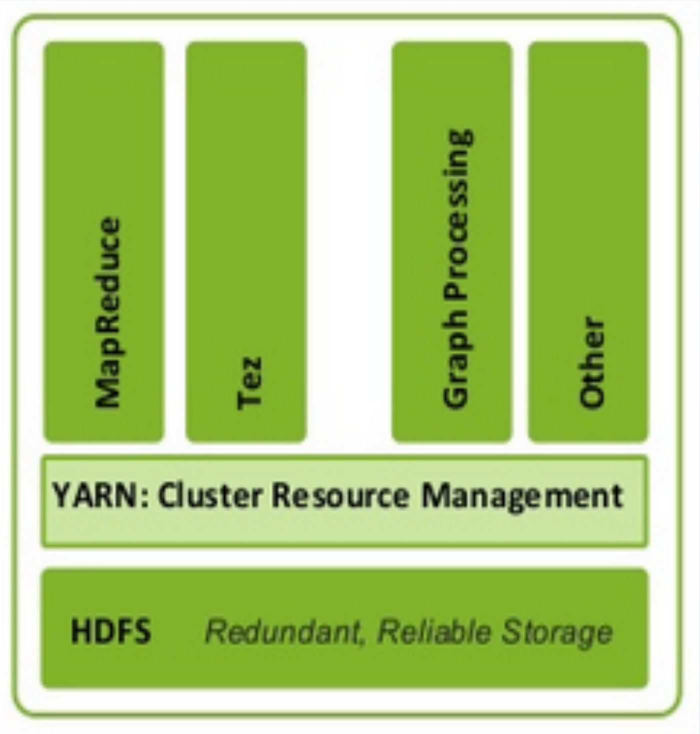
Resource Management

- (At least) Two dimension problem:

1. RM across different frameworks
   - Usually not a dedicated cluster
   - Shared across multiple frameworks
     - ETL (MapReduce, Spark), Hbase
     - SQL-on-Hadoop processing
2. RM across concurrent queries

RM -- Across frameworks

- YARN – Yet Another Resource Negotiator
- Centralized, cluster-wide resource management system
  - Allows frameworks to share resources without partitioning between them
- Designed for batch-mostly processing
  - Not mature
  - Not good for interactive analytics
  - Not meant for long running processes
    - Approaches: Llama and Slider

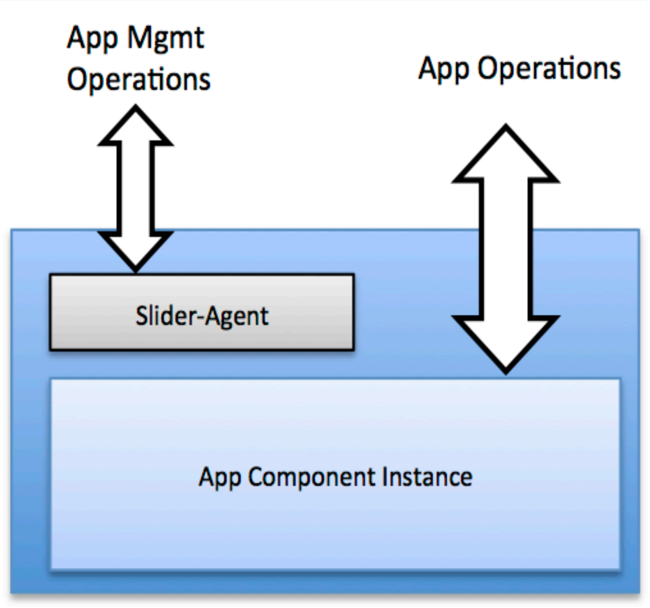


RM -- LLAMA (low-latency application master)

- Introduced by Cloudera
- LLAMA acts as a proxy between Impala and YARN
- Mitigates some of the batch-centric design aspects of YARN:
  - High resource acquisition latency -> solves via resource caching
  - Resource request is immutable -> solves via expansion request
  - Resource allocation is incremental -> solves via gang scheduling

RM -- Apache Slider

- Slider allows running non-YARN enabled applications on YARN
  - Without having to write your own custom Application Master
- Existing applications are **packaged** as Slider applications
  - Encapsulates a set of one or more application components or roles
  - Deployed by Slider, runs in containers across a YARN cluster
- Pre-built packages for HBase, Accumulo, Storm, and jmemcached
  - Packages need to be custom built for other applications
- Some notable Slider features
  - Applications can be stopped and started later → state is persisted
  - Container failures are automatically detected by Slider and restarted



Query Optimization

Some Techniques We Know and Love Are not Directly Applicable

- Indexing
- Zone-maps
- Co-located joins
- Query rewrites
- Cost-based optimization

- Databases own their storage SQL-on-Hadoop systems do not
  - Metadata management is tricky
  - Data inserted/loaded without SQL system knowledge
  - No co-location of related tables
  - HDFS is for most practical purposes, read-only

I/O Elimination for HDFS Data: Partition-level

- Hive Partition tables maintain metadata values as one folder/ directory in HDFS, per distinct value:
  - Example: PARTITIONED BY (country STRING, year INT, month INT, day INT) ;
    - Folder/Directory created for country=US/year=2012/month=12/day=22
  - Partitioning only logical, not physical
- Partition pruning eliminates reading files that are not needed
- Almost all SQL-on-Hadoop offerings support this
  - Hive, Impala, SparkSQL, IBM BigSQL, ...

I/O Elimination for HDFS Data: Rowblock-level

- ORCFile broken into Stripes (250MB default)
  - Index with Min/Max values stored for each Column
  - Data is a "stream" of columns
- Bloom filters for each stripe in ORCFile allow fast lookups
- Parquet also supports min/max values
- Works well when data is sorted, not very effective otherwise

Quick look at query optimizers

- Two types of optimization
  - Logical transformations to transform query into equivalent but simpler form
  - Cost-based enumeration of alternative execution plans
- Most systems support the first one
- Cost-based optimization depends on good statistics and a good model of the execution environment
  - Without controlling data storage, statistics are "gestimates"

Query Rewrite

- Selection/projection pushdown
- Nested SQL queries require more sophisticated rewrites, such as decorrelation
- New systems all have rewrites but lack complex decorrelation and subquery optimization ones
  - Hive, Impala, Presto, Spark SQL
- Systems that leverage mature DB technology offer more sophisticated rewrite engines
  - IBM SQL, Hadapt, HP Vertica

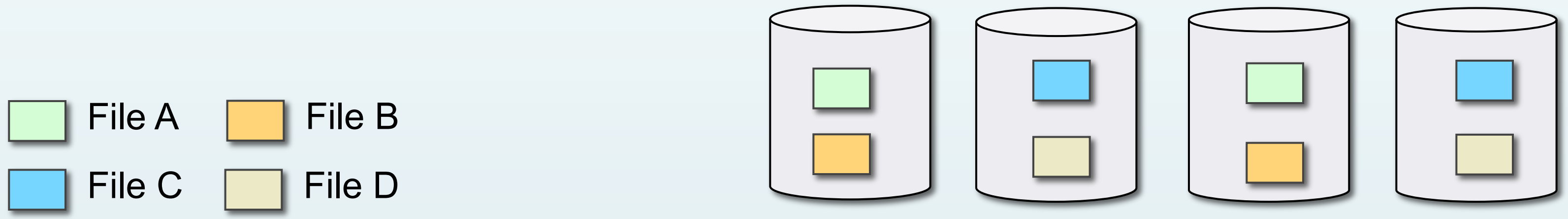
Cost-based Optimization

- Hive analyze table collects basic statistics
  - Column value distributions, min-max, no-of-distinct values
- No control of data → data changes without the systems' knowledge
- Multi-tenant system makes it harder to build a cost model
  - More complex system behavior

More adaptive query processing is needed

Co-located joins

- Co-partitioning two tables on the join key enables local joins



- Files A & B are co-located
- Files C & D are co-located

- HDFS default block placement policy scatters blocks in the cluster
- Actian Vortex changes HDFS default block placement to enforce co-located joins

Outline of Tutorial

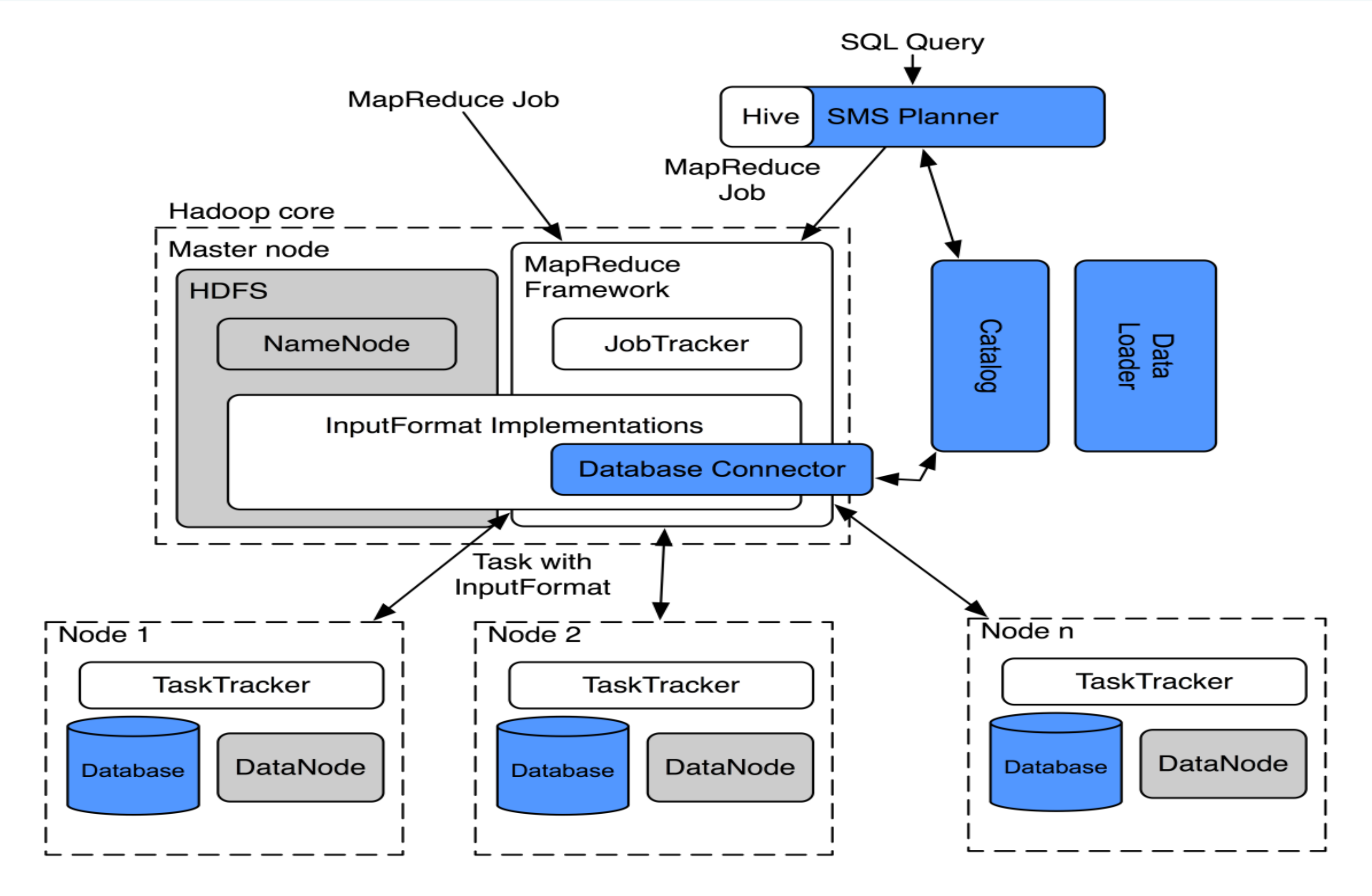
- This session [13:30-15:00]
  - SQL-on-Hadoop Technologies
    - Storage
    - Run-time engine
    - Query optimization
  - Q&A

- Second Session [15:30-17:00]
  - SQL-on-Hadoop examples
    - HadoopDB/Hadapt
    - Presto
    - Impala
    - BigSQL
    - SparkSQL
    - Phoenix/Spice Machine
  - Research directions
  - Q&A

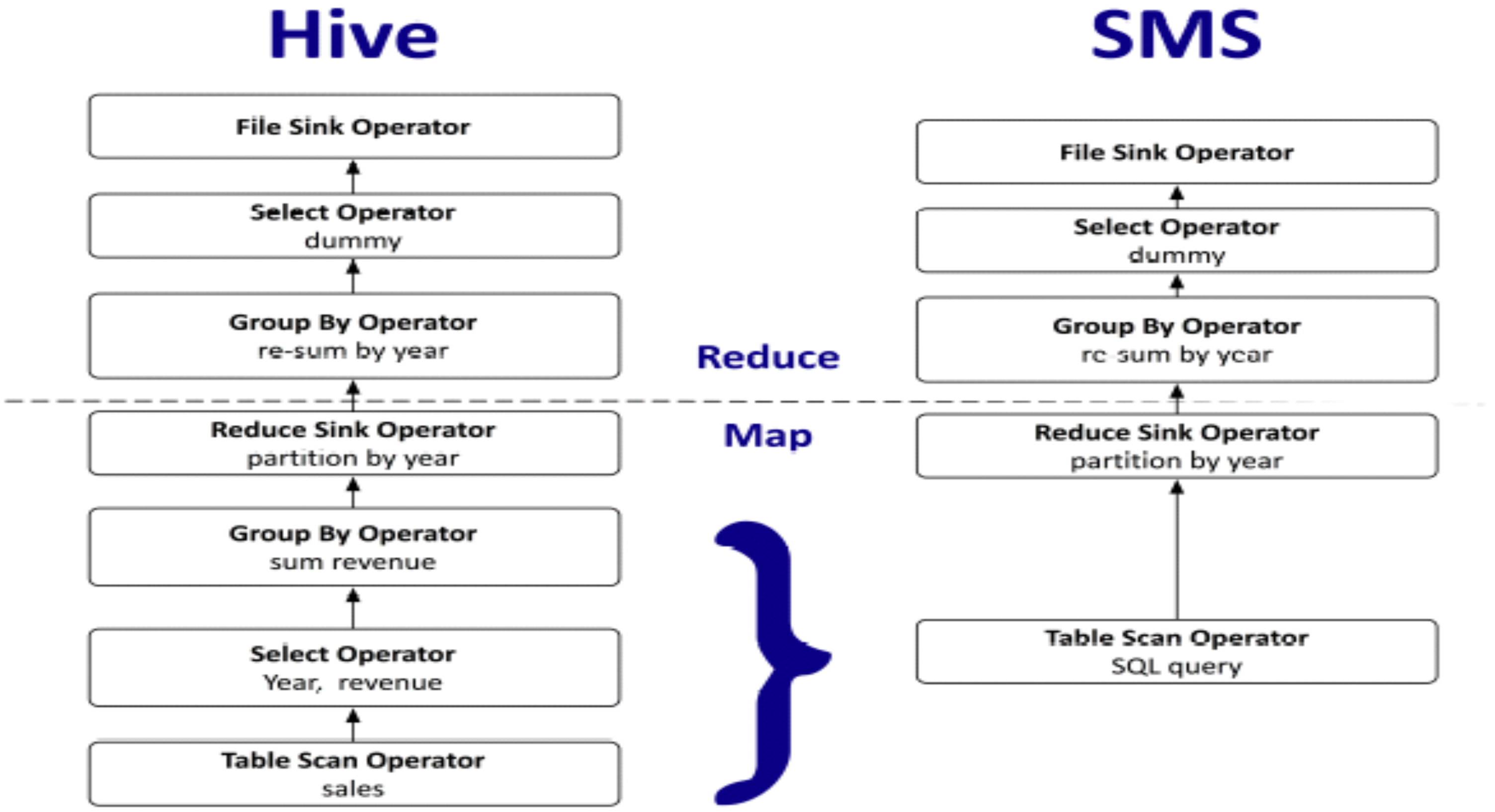
HadoopDB

- First of avalanche of SQL-on-Hadoop solutions to claim 100x faster than Hive (on certain types of queries)
- Used Hadoop MapReduce to coordinate execution of multiple independent (typically single node, open source) database systems
  - Maintained MapReduce's fault tolerance
  - Sped up single-node processing via leveraging database performance optimizations:
    - Compression
    - Vectorization
    - Partitioning
    - Column-orientation
    - Query optimization
    - Broadcast joins
  - Flexible query interface (both SQL and MapReduce)

HadoopDB Architecture



HadoopDB SMS Planner

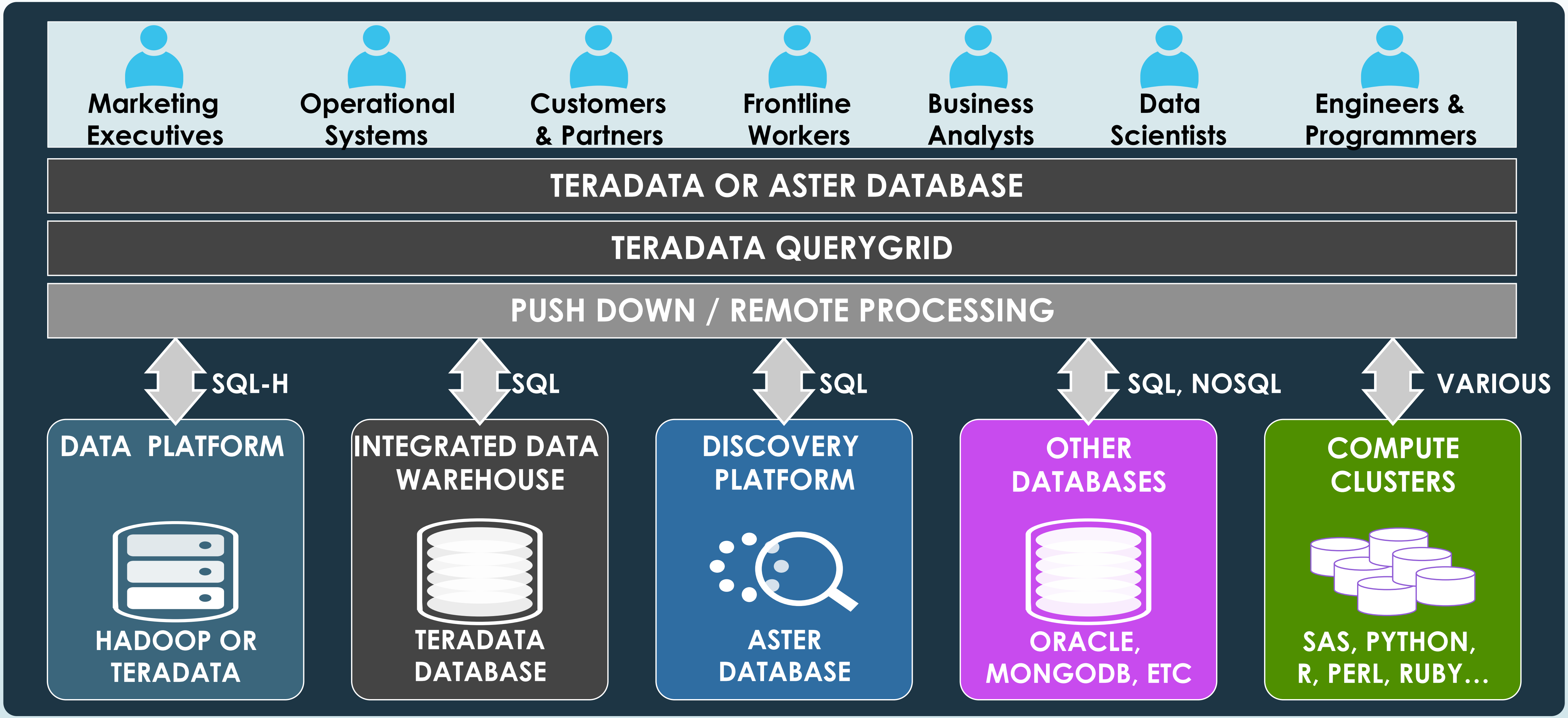


SELECT YEAR(saleDate), SUM(revenue) FROM sales GROUP BY YEAR(saleDate);

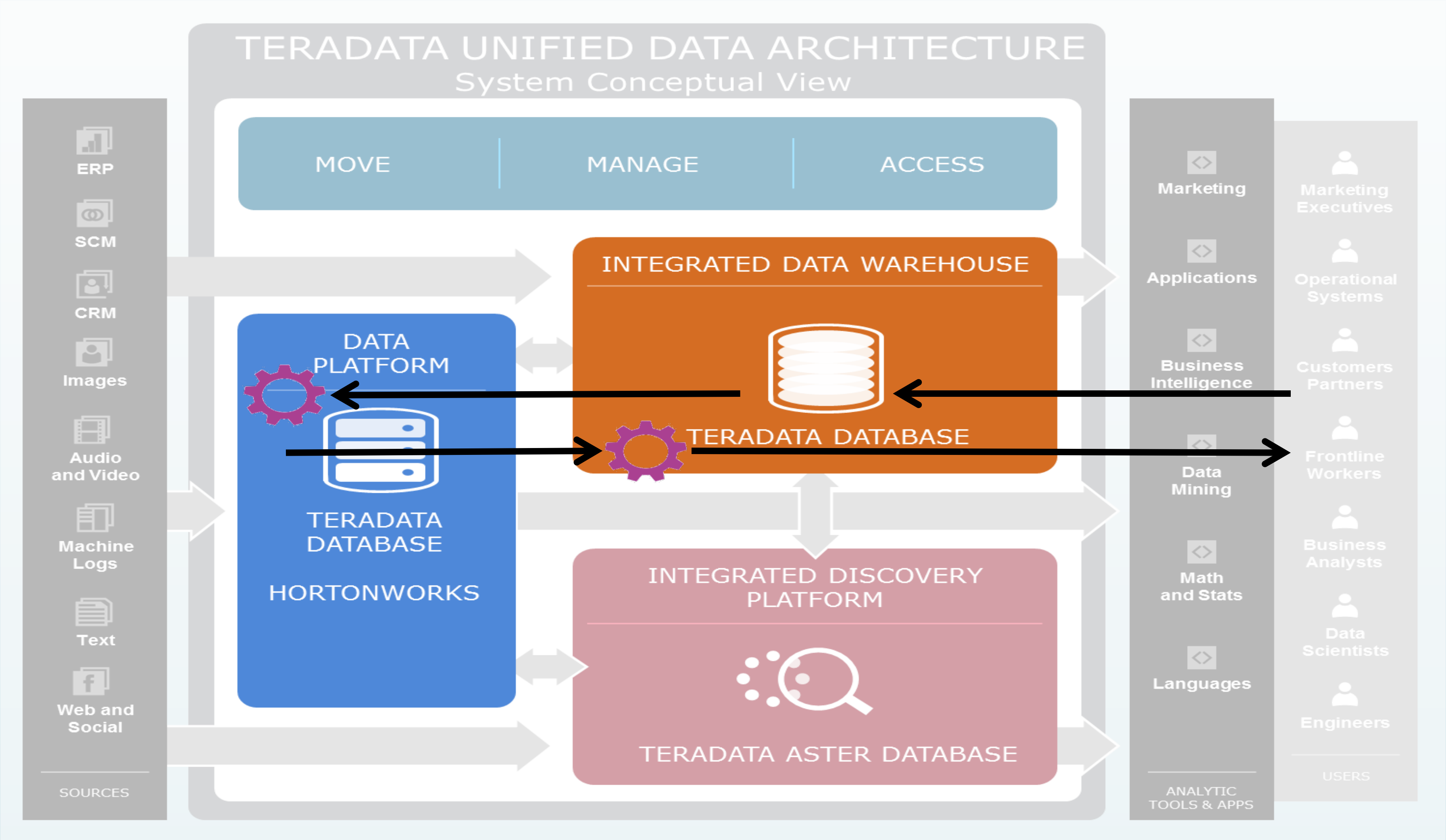
HadoopDB History

- Paper published in 2009
- Company founded in 2010 (Hadapt) to commercialize HadoopDB
- Added support for search in 2011 (for major insurance customer)
- Added JSON support in 2012
- Added interactive query engine in 2013
- Acquired by Teradata in 2014

Teradata Unified Data Architecture: QueryGrid



Remote Processing On Hadoop

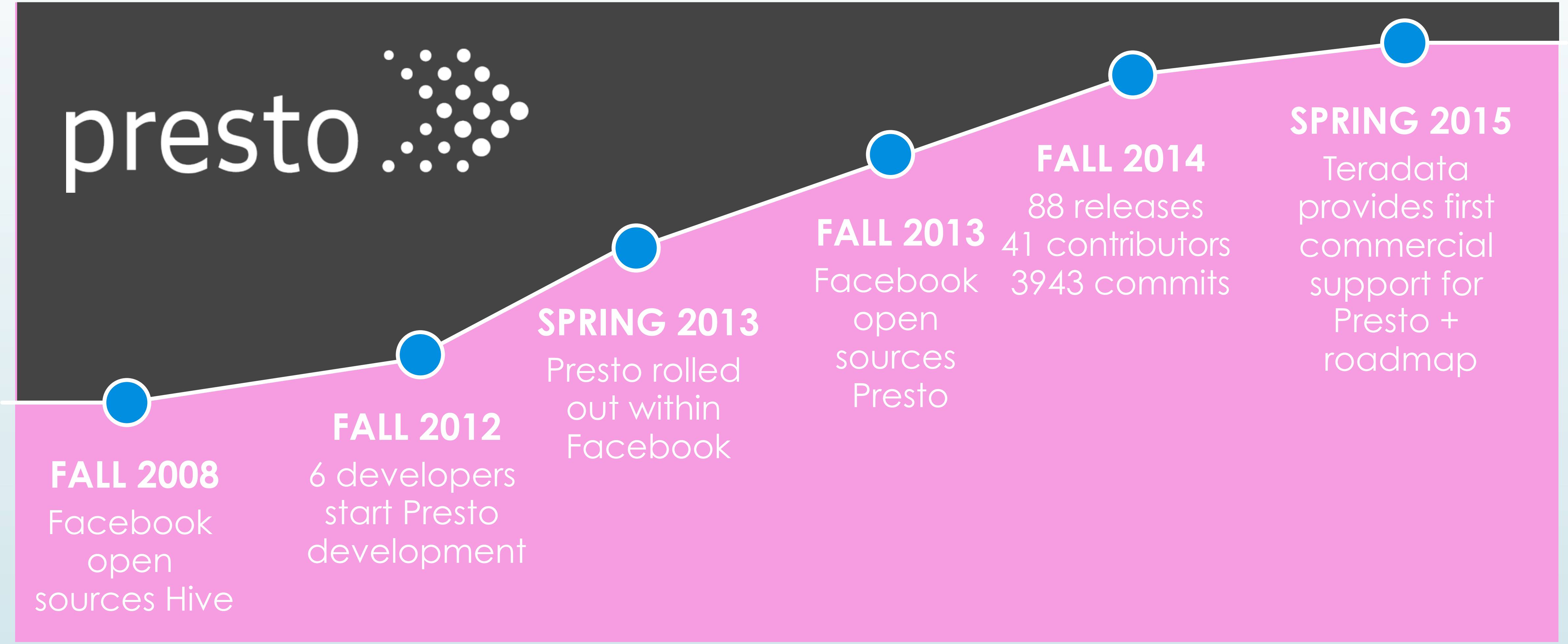


- Query through Teradata
- Leaves of query plan sent to SQL-on-Hadoop engine
- Results returned to Teradata
- Additional query processing done in Teradata
- Final results sent back to application/user
- Teradata 15.0

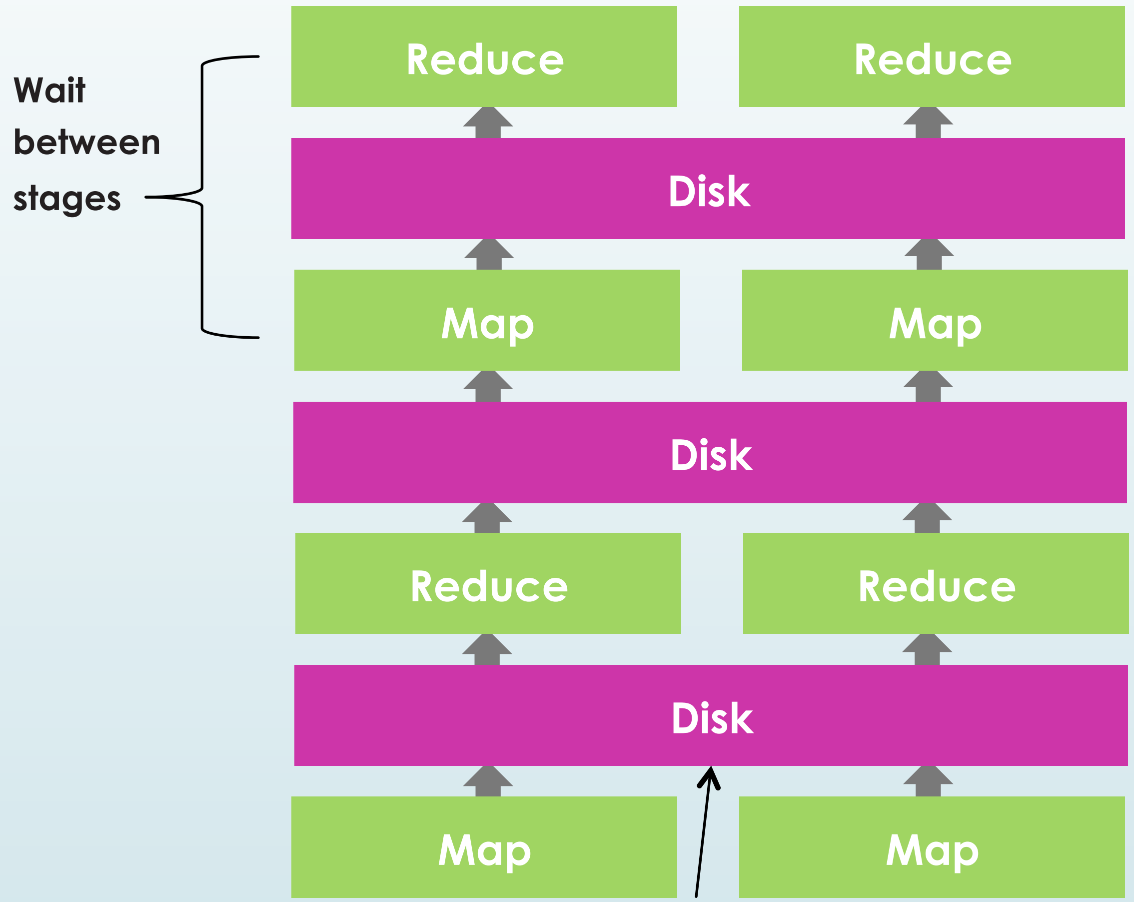
Teradata QueryGrid Teradata-Hadoop

- Bi-directional data movement
  - Read and write data to Hadoop
  - Create new table in Hadoop or insert records
- Query push-down
  - Execute query on Hadoop
  - Qualify rows and columns to reduce data returned
- Easy configuration and simplified queries
  - Create “Hadoop server” definition once
  - Use @foreign_server name to access Hadoop

History of Presto



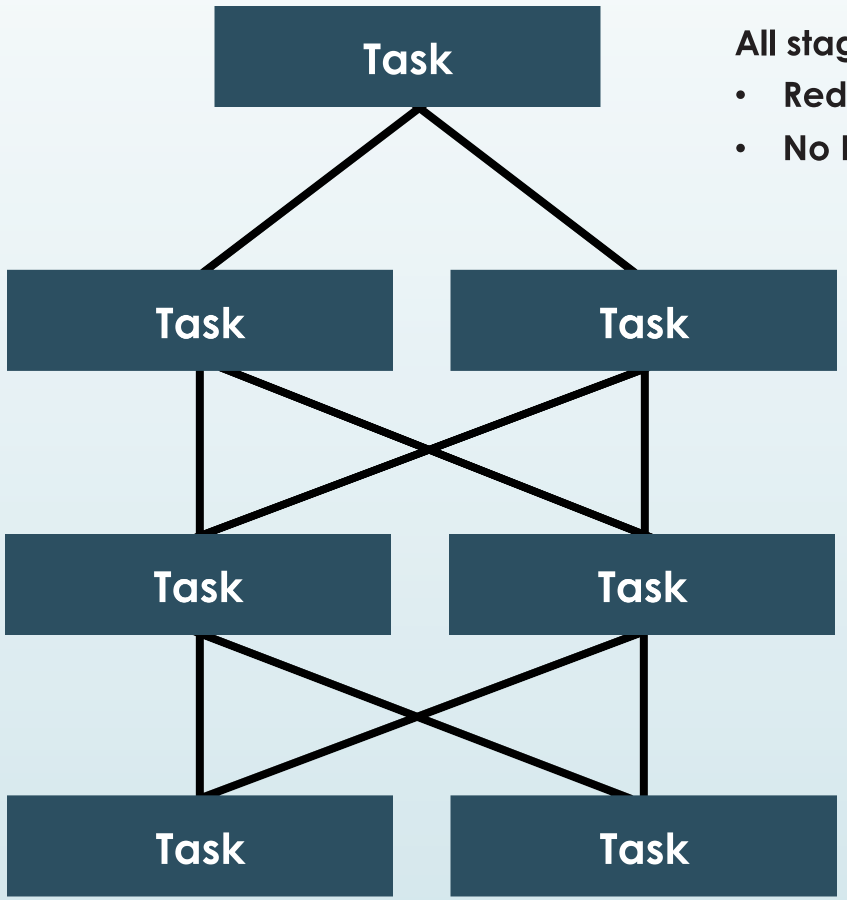
Hive



Write to Disk

- Fault Tolerance
- IO Overhead

Presto



All stages are pipelined

- Reduced wait time
- No Fault Tolerance

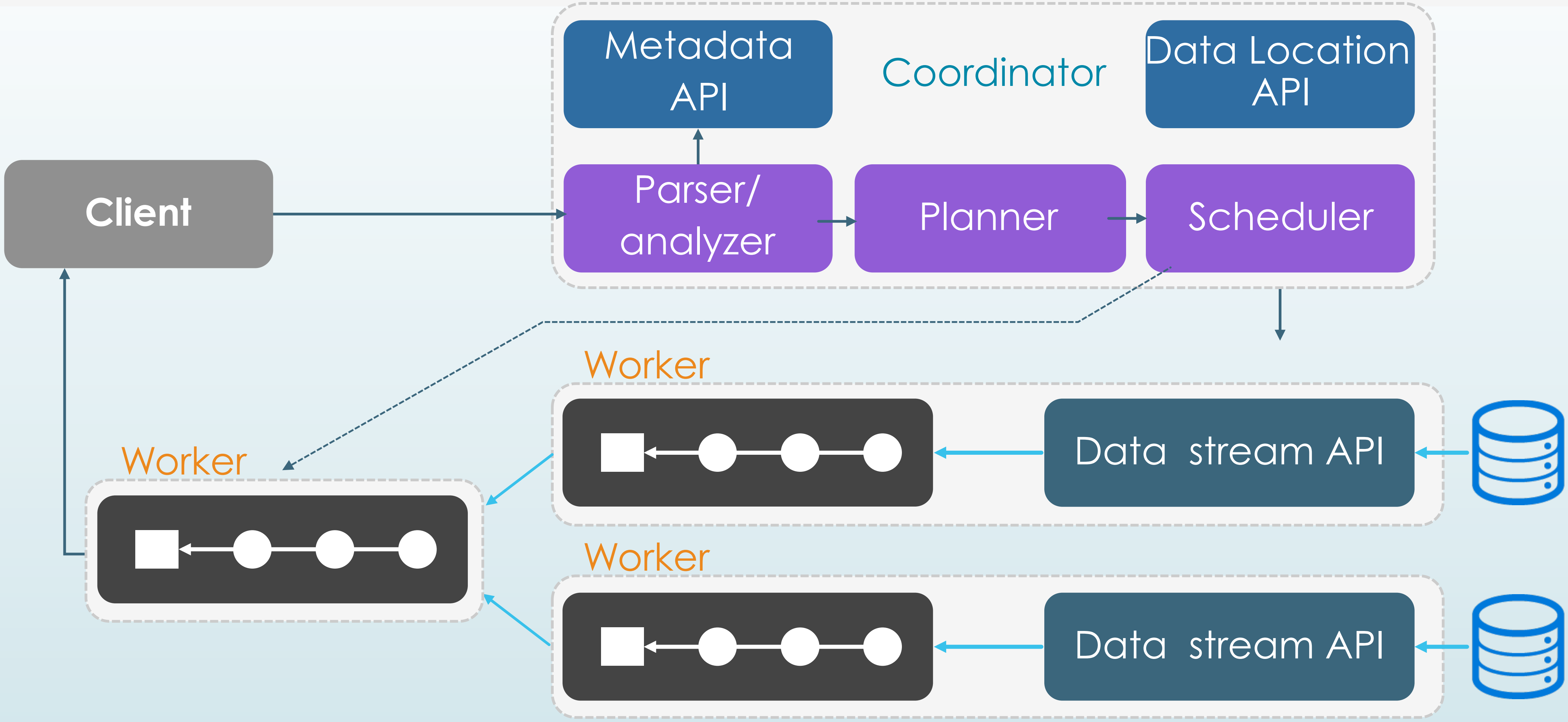
Memory-to-memory
Data transfer

- No disc IO
- Data chunk must fit in memory

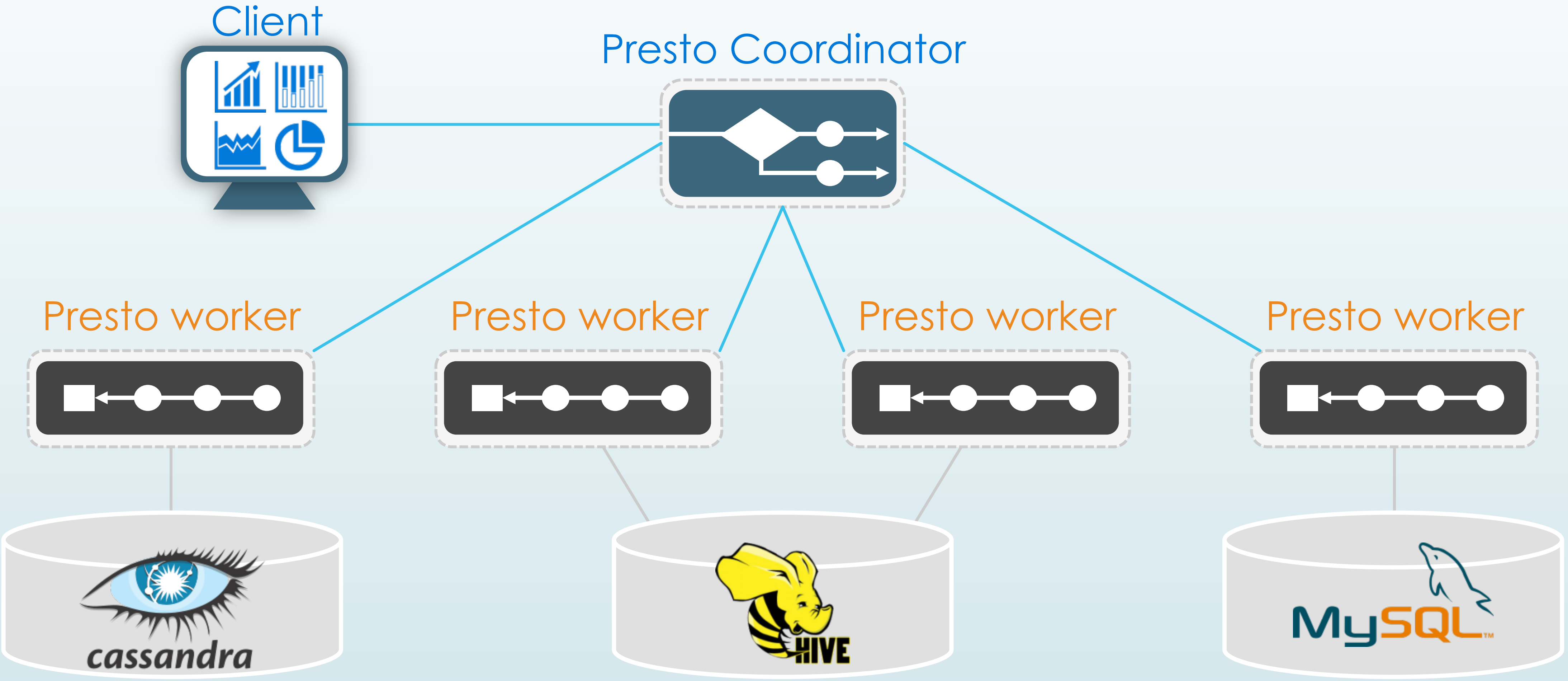
Presto at a Glance

- Written in Java
- 100% ANSI SQL goal
  - Numerous built-in functions
  - Window functions
  - Array/map support
- Plug-in architecture
  - Join across data stores
  - Hive, Cassandra, Kafka, MySQL
  - Amazon S3
- Uses Hive metastore
- Bytecode query compilation
- Approximate queries
  - Return X% sample rows
- Limitations
  - Manual join SQL ordering
  - Non-equi joins not supported
  - Not YARN enabled
  - No Avro support
  - No spill-to-disk

Presto Pipeline Architecture



Presto Connectors

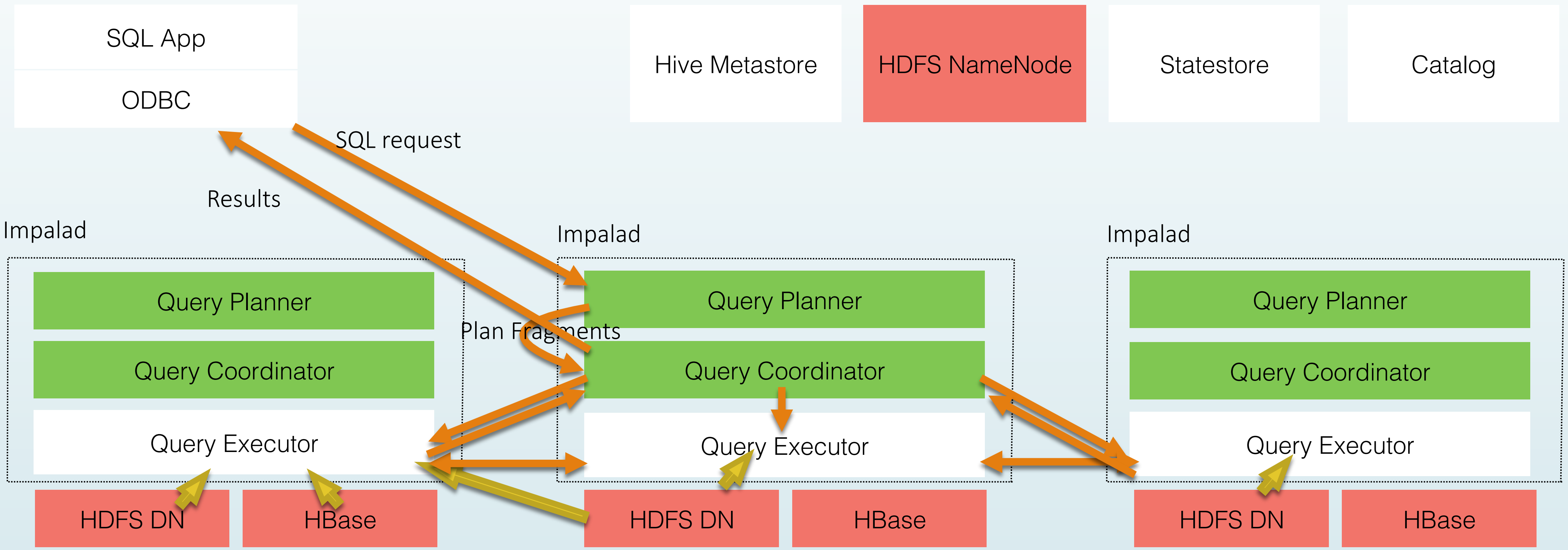


Github: Presto Plug-in Connectors

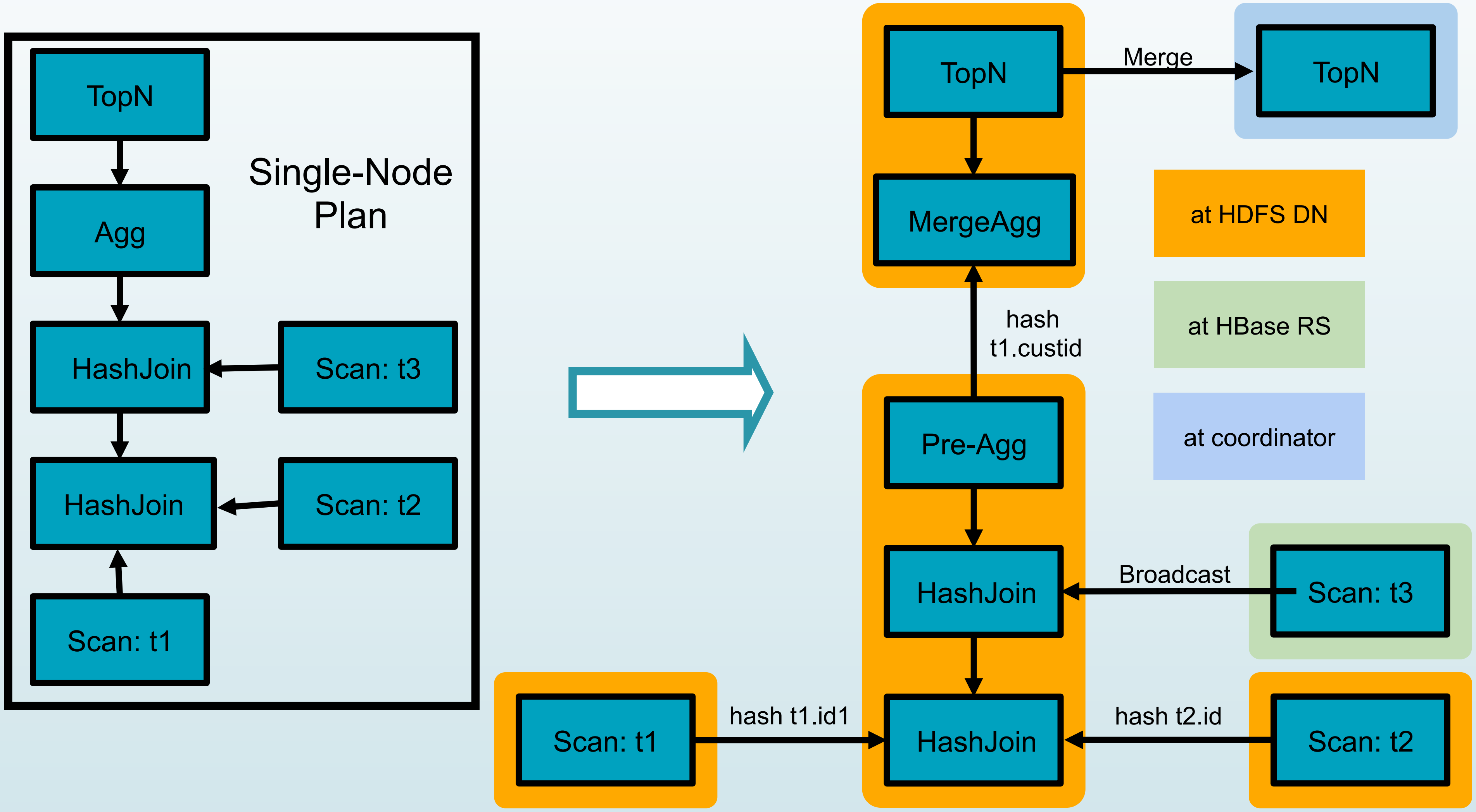
- Hive tables and HCatalog
- Apache Cassandra
- Apache Kafka
  - Kafka topics = Presto tables, messages = rows
- MySQL
  - Single node access only -- no sharding
- Postgres
  - Single node access only
- HBase
  - not released

Cloudera Impala

Query execution at the high level



Query Planning: Distributed Plans



Execution Engine

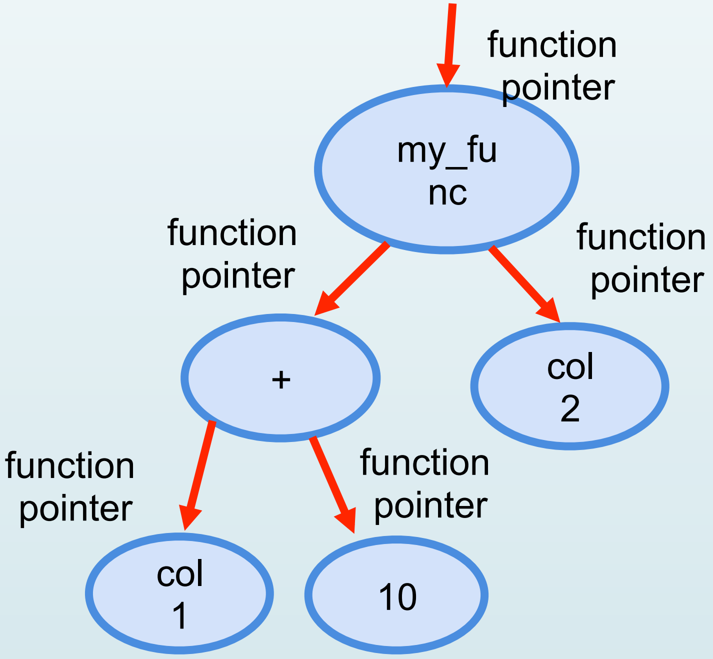
- Written in C++ for minimal cycle and memory overhead
- Leverages decades of parallel DB research
  - Partitioned parallelism
  - Pipelined relational operators
  - Batch-at-a-time runtime
- Focussed on speed and efficiency
  - Intrinsics/machine code for text parsing, hashing, etc.
  - Runtime code generation with LLVM

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, unroll loops
  - Propagate constants, offsets, pointers, etc.
  - Inline function calls
- Optimized execution for modern CPUs (instruction pipelines)

Runtime Code Generation — Example

```
IntVal my_func(const IntVal& v1, const IntVal& v2) {
  return IntVal(v1.val * 7 / v2.val);
}
SELECT my_func(col1 + 10, col2) FROM ...
```



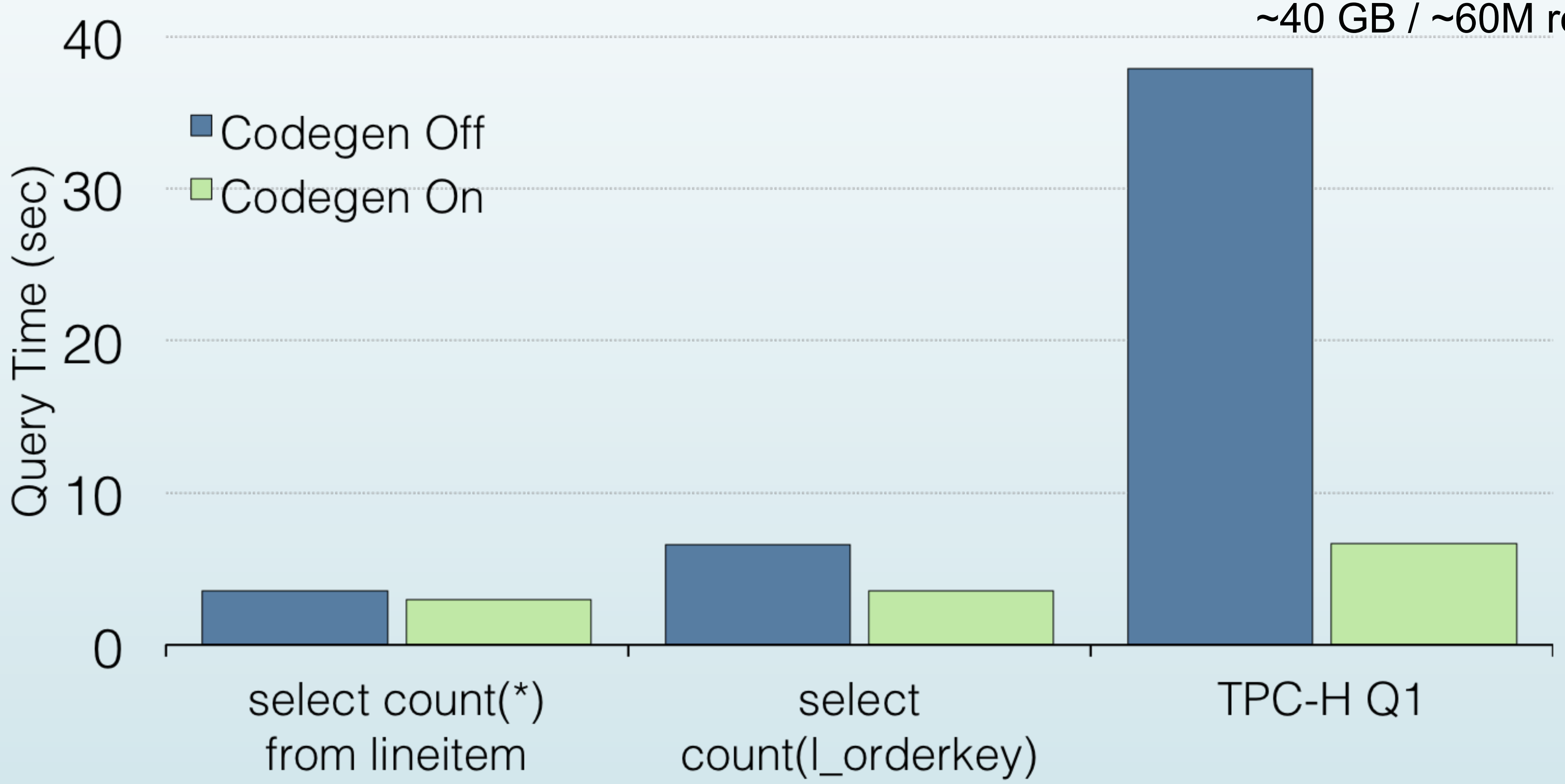
(col1 + 10) * 7 / col2

interpreted

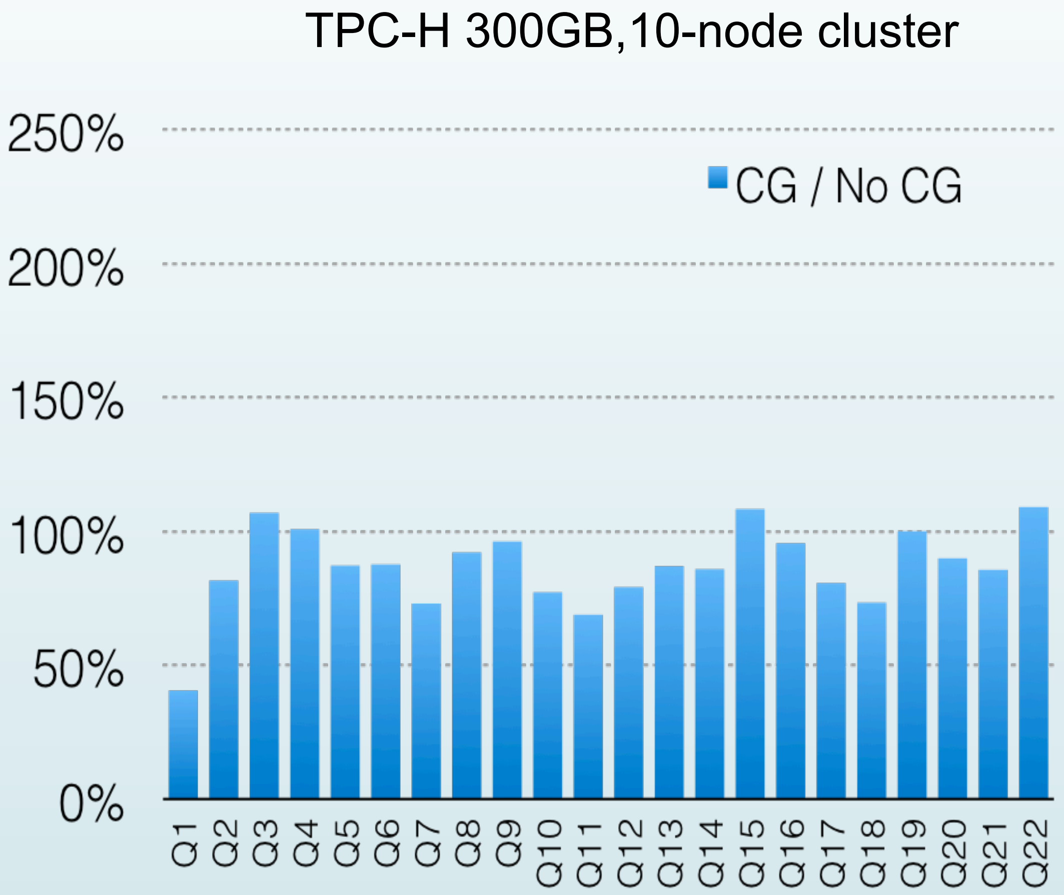
codegen’d

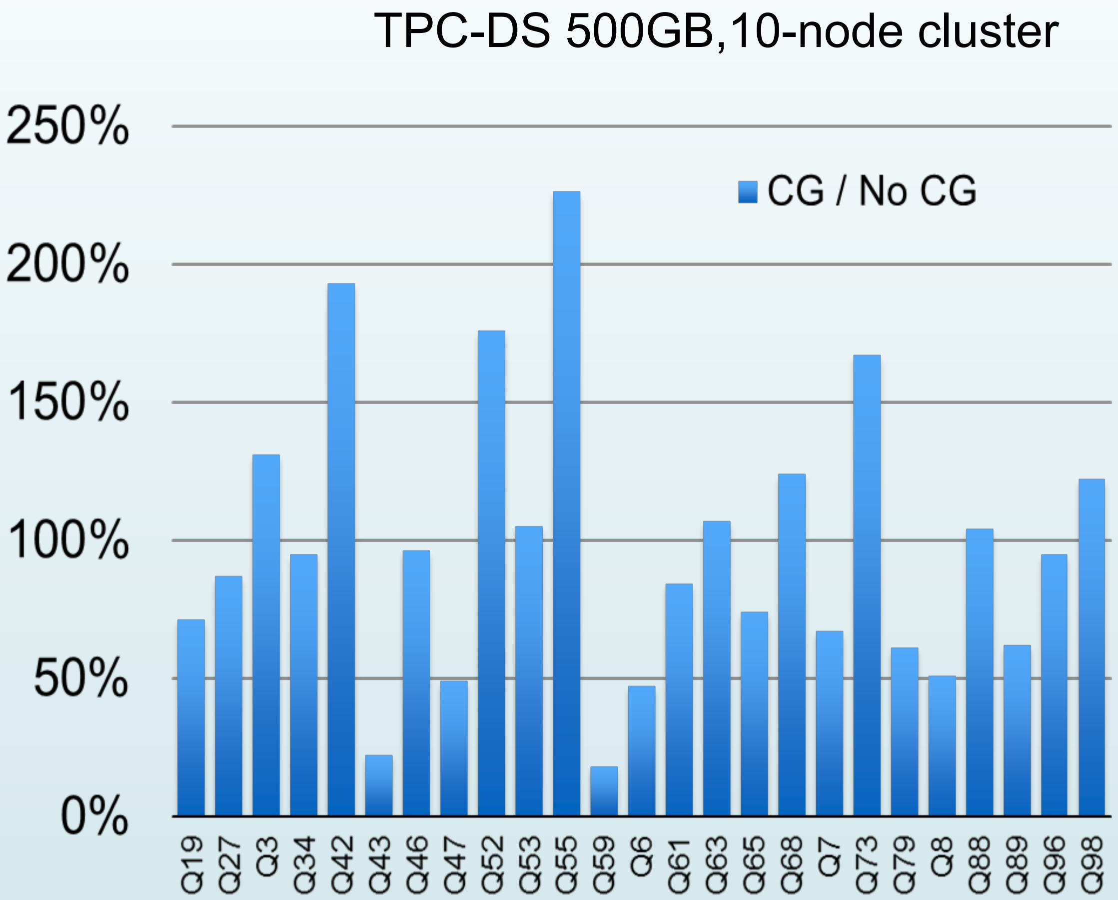
Impala Runtime Code Generation - Performance

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



Codegen is not the panacea!





Resource Management in Impala

- Admission control and Yarn-based RM cater to different workloads
- Use admission control for:
  - Low-latency, high-throughput workloads
  - Mostly running Impala, or resource partitioning is feasible
- Use Llama/Yarn for:
  - Mixed workloads (Impala, MR, Spark, …) and resource partitioning is impractical
  - Latency and throughput SLAs are relatively relaxed

Roadmap: Impala 2.3+

- Nested data: 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
- Multithreaded execution past scan operator
- Resource management and admission control
  - low-latency, high-throughput mixed workloads without resource partitioning
- More SQL: ROLLUP/GROUPING SETS, INTERSECT/MINUS, MERGE
- Improved query planning, using statistics
- Physical tuning

Ibis: Scaling the Python Data Experience

http://www.ibis-project.org/

Target user:

Data scientists and data engineers ("Python data users")

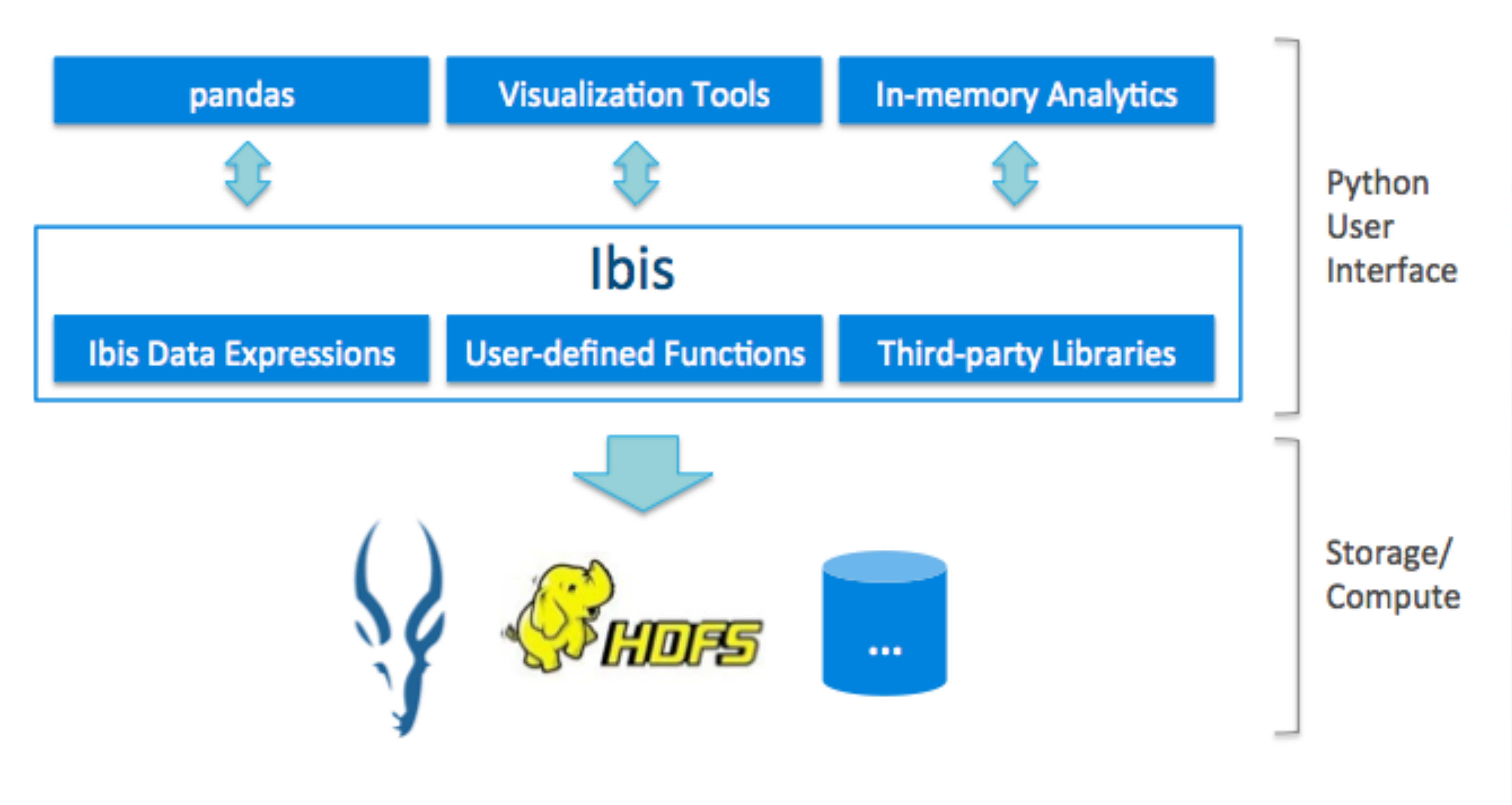
Goals:

Mirror single-node Python experience, maximize productivity

Complete support for SQL engines with Pandas-like API (same designer)

High-performance Python user-defined functions

Integration with Python data ecosystem / libraries


pandas
Visualization Tools
In-memory Analytics
Ibis
Ibis Data Expressions
User-defined Functions
Third-party Libraries
HDFS
...
Python
User
Interface
Storage/
Compute

Ibis/Impala Joint Roadmap

- More natural data modeling
    - Complex types support
- Integration with full Python data ecosystem
    - Advanced analytics + machine learning
    - Enable use of performance computing tools
- User extensibility with native performance
    - In-memory columnar format
    - Python-to-LLVM IR compilation
- Workflow and usability tools

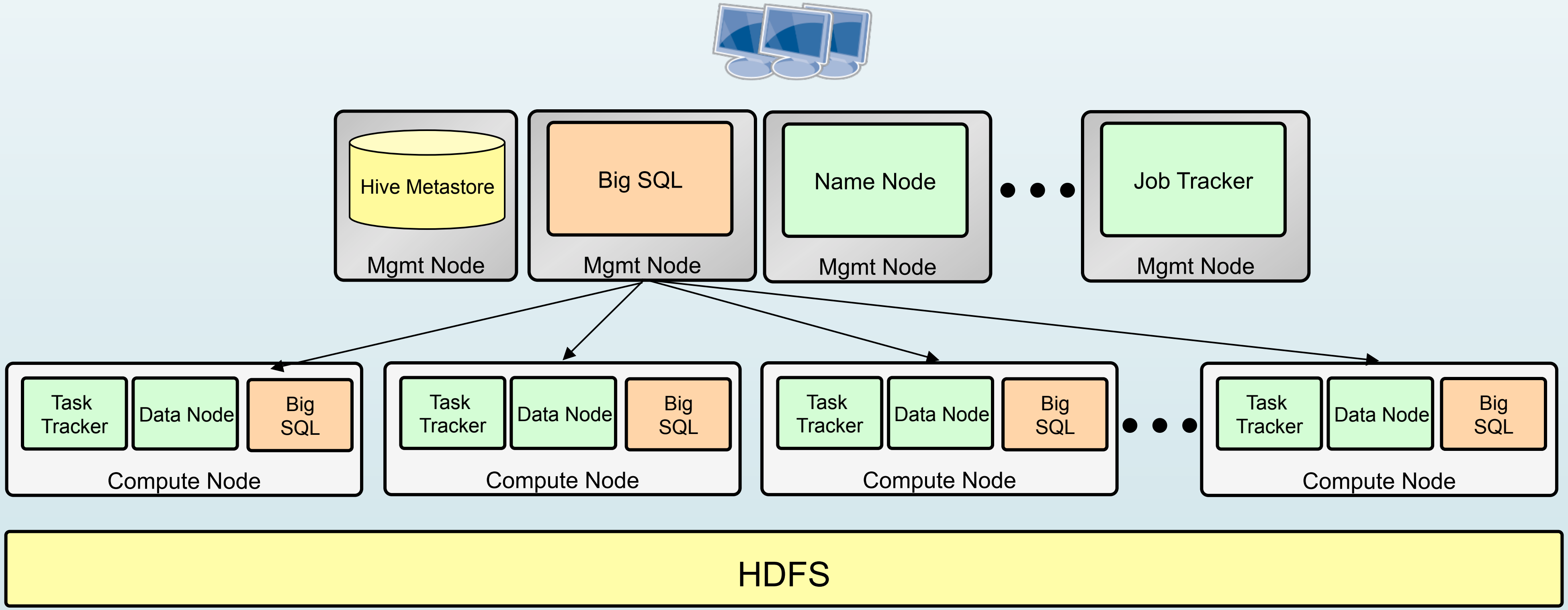
Academic Challenge

- Code at github (https://github.com/cloudera/Impala/)
- Impala Developer Docker Images & Chef scripts
- https://registry.hub.docker.com/u/cloudera/impala-dev/
  - Minimal (7GB) — ready to compile, latest code
  - Default (33GB) — includes test data, e.g. TPC-H
- Shout out to Spyros Blanas (Ohio State)
  - http://web.cse.ohio-state.edu/~sblanas/5242/
- Impala JIRAs, ramp-up tasks

IBM Big SQL

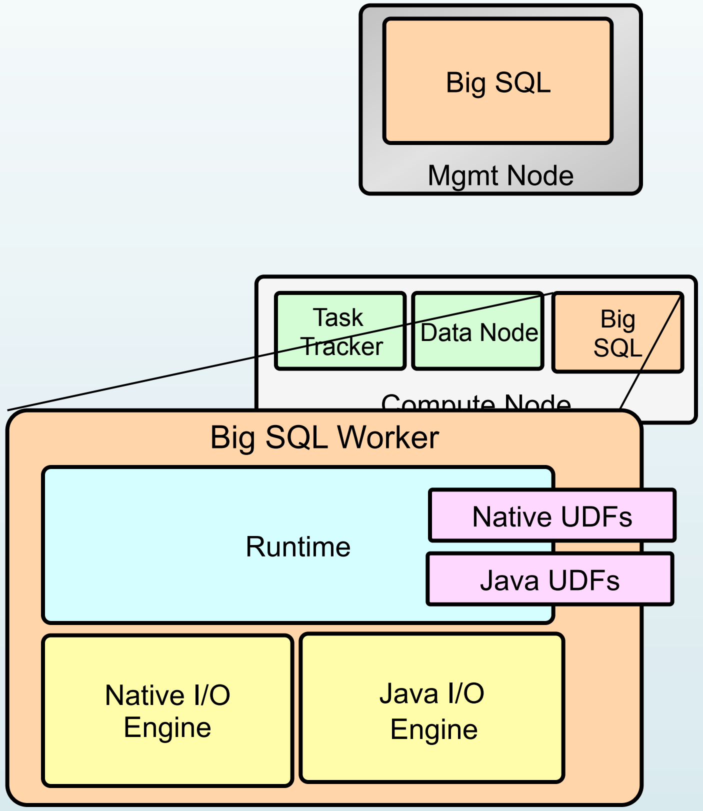
Big SQL – Architecture

- Head (coordinator) node
  - Compiles and optimizes the query
  - Coordinates the execution of the query
- Big SQL worker processes reside on compute nodes (some or all)
- Worker nodes stream data between each other as needed



Big SQL – Architecture (cont.)

- For common table formats a native I/O engine is utilized
  - e.g. delimited, RC, SEQ, Parquet, …
- For all others, a java I/O engine is used
  - Maximizes compatibility with existing tables
  - Allows for custom file formats and SerDe's
- All Big SQL built-in functions are native code
- Customer built UDF's can be developed in C++ or Java

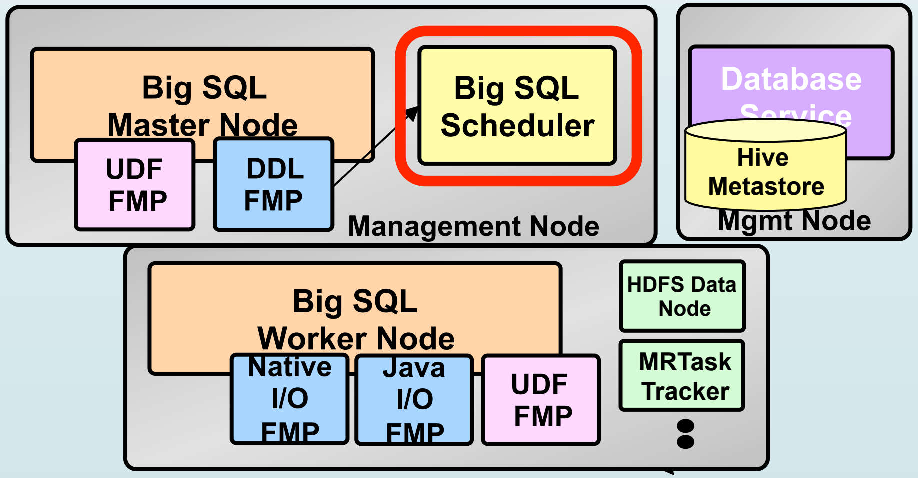


Big SQL works with Hadoop

- All data is Hadoop data
  - In files in HDFS
  - SEQ, ORC, delimited, Parquet …
- Never need to copy data to a proprietary representation
- All data is catalog-ed in the Hive metastore
  - It is the Hadoop catalog
  - It is flexible and extensible

Scheduler Service

- The scheduler is the main RDBMS↔Hadoop service interface
- Interfaces with Hive metastore for table metadata
  - SQL compiler ask it for some "hadoop" metadata, such as partitioning columns
- Acts like the MapReduce job tracker for Big SQL
  - Big SQL provides query predicates for scheduler to perform partition elimination
  - Determines splits for each "table" involved in the query
  - Schedules splits on available Big SQL nodes (with best effort data locality)
  - Decides which I/O library to use and serves work (splits) to them
  - Coordinates "commits" after INSERTs



Query Rewrite

- There are many ways to express the same query
- Query generators often produce suboptimal queries and don't permit "hand optimization"
- Complex queries often result in redundancy, especially with views
- For large data volumes optimal access plans more crucial as penalty for poor planning is greater

```
select sum(l_extendedprice) / 7.0
avg_yearly
from tpcd.lineitem, tp
where p_partkey = l_p
    and p_brand = 'Br
    and p_container =
    and l_quantity < ( select  0.2 *
     avg(l_quantity)  from
tpcd.lineitem
     where l_partkey = p_partkey);
```

- Query correlation eliminated
- Lineitem table accessed only once
- Execution time reduced in half!

```
select sum(l_extendedprice) / 7.0 as avg_yearly
 from temp (l_quantity,  avgquantity,
                      ) as
             antity, avg(l_quantity) over
             by l_partkey)
             antity, l_extenedprice
             neitem, tpcd.part
     where p_partkey = l_partkey
          and p_brand = 'BRAND#23'
          and p_container = 'MED BOX')
 where l_quantity < 0.2 * avgquantity
```

Cost-based Optimization

- Few extensions required to the Cost Model
- Scan operator cost model extended to evaluate cost of reading from Hadoop
  - # of files, size of files, # of partitions, # of nodes
- Data not hash partitioned on a particular columns (aka “Scattered partitioned”)
- New parallel join strategy
  - Every node read data from HDFS, instead of one reading and broadcasting
- Optimizer now knows in which subset of nodes the data resides => better costing!
- Sophisticated statistics for cardinality estimation

```
                    |
               2.66667e-08
                 HSJOIN
                 (   7)
               1.1218e+06
                  8351
          /--------+--------\
    5.30119e+08                3.75e+07
      BTQ                       NLJOIN
      (   8)                    (  11)
      948130                    146345
       7291                      1060
        |                    /----+----\
    5.76923e+08             1          3.75e+07
      LTQ            GRPBY              FILTER
      (   9)         (  12)             (  20)
      855793         114241             126068
       7291           1060               1060
        |               |                 |
    5.76923e+08        13               7.5e+07
      TBSCAN         TBSCAN             BTQ
      (  10)         (  13)             (  21)
      802209         114241             117135
       7291           1060               1060
        |               |                 |
      7.5e+09          13             5.76923e+06
TABLE: TPCH5TB_PARQ  TEMP               LTQ
      ORDERS         (  14)             (  22)
        Q1           114241             108879
                      1060               1060
                        |                 |
                       13             5.76923e+06
                     DTQ                TBSCAN
                     (  15)             (  23)
                     114241             108325
                      1060               1060
                        |                 |
                         1              7.5e+08
                     GRPBY        TABLE: TPCH5TB_PARQ
                     (  16)            CUSTOMER
                     114241               Q5
                      1060
                        |
                         1
                     LTQ
                     (  17)
                     114241
                      1060
                        |
                         1
                     GRPBY
                     (  18)
                     114241
                      1060
                        |
                   5.24479e+06
                     TBSCAN
                     (  19)
                     113931
                      1060
                        |
                     7.5e+08
               TABLE: TPCH5TB_PARQ
                    CUSTOMER
                       Q2
```

Statistics

- Big SQL utilizes Hive statistics collection with some extensions:
  - Additional support for column groups, histograms and frequent values
  - Automatic determination of partitions that require statistics collection vs. explicit
  - Partitioned tables: added table-level versions of NDV, Min, Max, Null count, Average column length
  - Hive catalogs as well as database engine catalogs are also populated
  - We are restructuring the relevant code for submission back to Hive
- Capability for statistic fabrication if no stats available at compile time

Table statistics
- Cardinality (count)
- Number of Files
- Total File Size

Column statistics
- Minimum value (all types)
- Maximum value (all types)
- Cardinality (non-nulls)
- Distribution (Number of Distinct Values NDV)
- Number of null values
- Average Length of the column value (all types)
- Histogram - Number of buckets configurable
- Frequent Values (MFV) – Number configurable

Column group statistics

Big SQL supports HBase tables

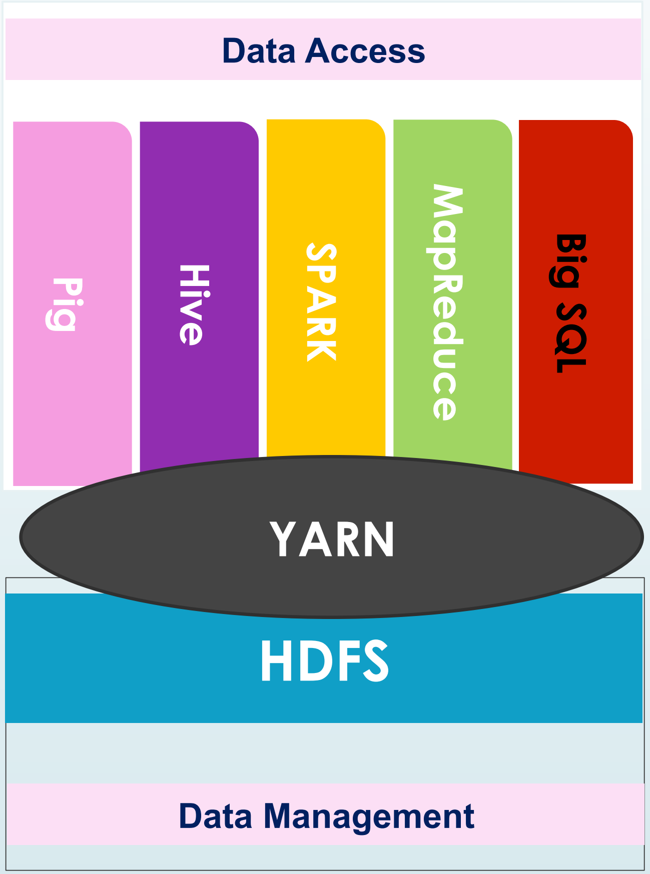
- Big SQL with HBase – basic operations
  - Create tables and views
  - LOAD / INSERT data
  - Query data with full SQL breadth

- HBase-specific design points
  - Column mapping
  - Dense / composite columns
  - FORCE KEY UNIQUE option
  - Secondary indexes
  - ....

APACHE HBASE

Big SQL works under YARN

- Big SQL integrates with YARN via the Slider project
  - YARN chooses suitable hosts for Big SQL worker nodes
  - Big SQL resources are accounted for by YARN
  - Size of the Big SQL cluster may dynamically grow or shrink as needed
  - Configured by user (not by installation default)
  - More Big SQL workers are added when more resources are needed
  - When demand wears off, Big SQL workers are shut down



Summary

- Big SQL provides *rich, robust*, *standards-based* SQL support for data stored in HDFS and HBase
  - Uses IBM common client ODBC/JDBC drivers

- Big SQL *fully integrates* with SQL applications and tools
  - Existing queries run with no or few modifications*
  - Existing JDBC and ODBC compliant tools can be leveraged

- Big SQL provides *faster* and *more reliable performance*
  - Big SQL uses more efficient access paths to the data
  - Big SQL is optimized to more efficiently move data over the network
  - Big SQL is capable of executing *all* 22 TPC-H and *all* 99 TPC-DS queries without modification

- Big SQL provides and *enterprise grade data management*
  - Security, Auditing, workload management …

SparkSQL

What is so great about Spark?

We believe that Spark is the first system that allows a general-purpose programming language to be used at interactive speeds for in-memory data mining on clusters.

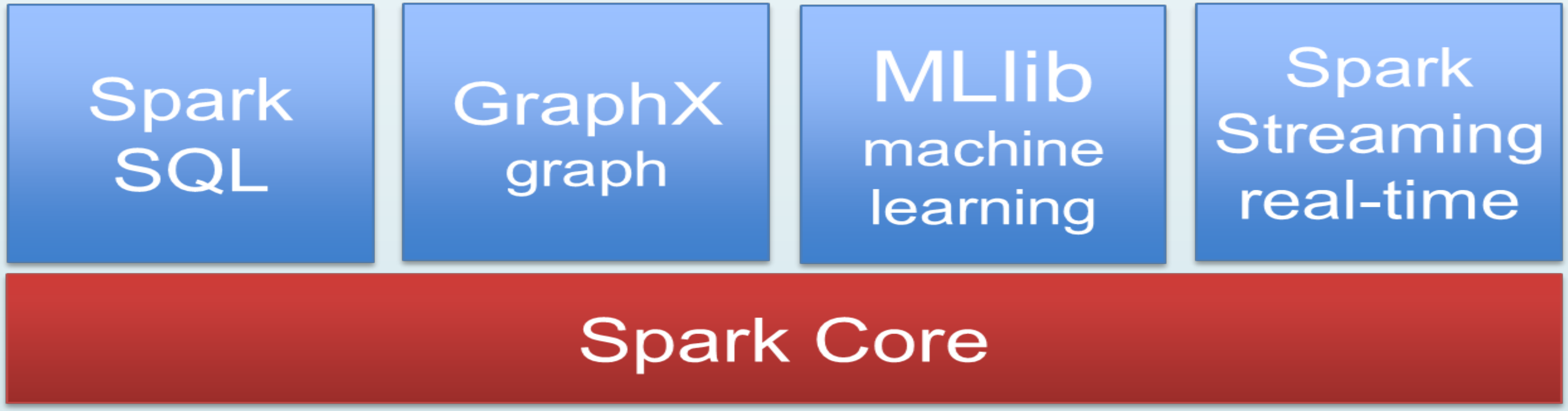
From:

Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing

Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, Ion Stoica
University of California, Berkeley

OK, but what exactly is Spark?

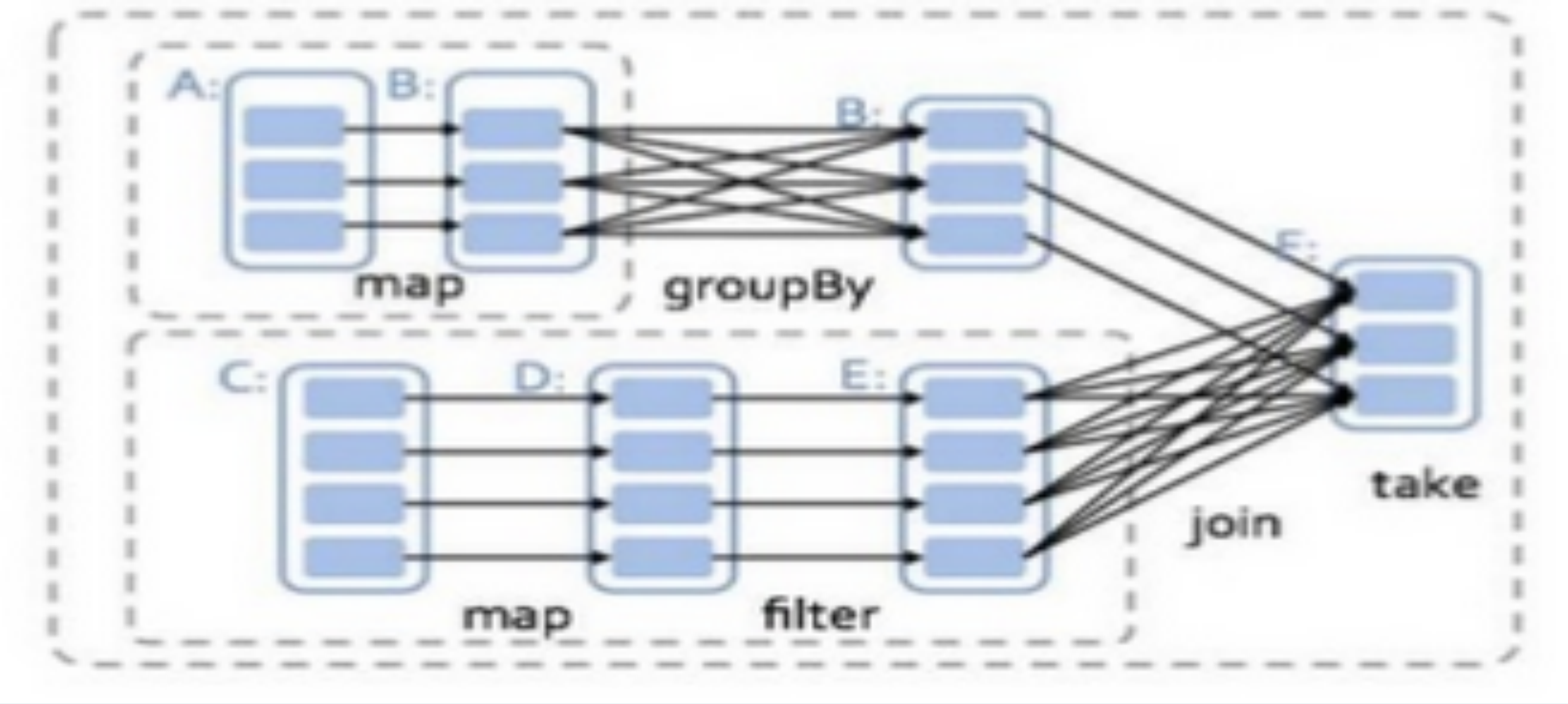
- Distributed data analytics engine, generalizing Map Reduce
- Core engine, with streaming, SQL, machine learning, and graph processing modules



Spark Core: RDDs, Transformations & Actions

- RDDs
  - Distributed collection of objects
  - Can be cached in memory
- Built via parallel transformations (map, filter, …)
  - Automatically rebuilt on failure based on lineage
- DAGs of RDDs and Transformations can be (lazily) executed via actions
  - Examples: Export to HDFS, count number of objects

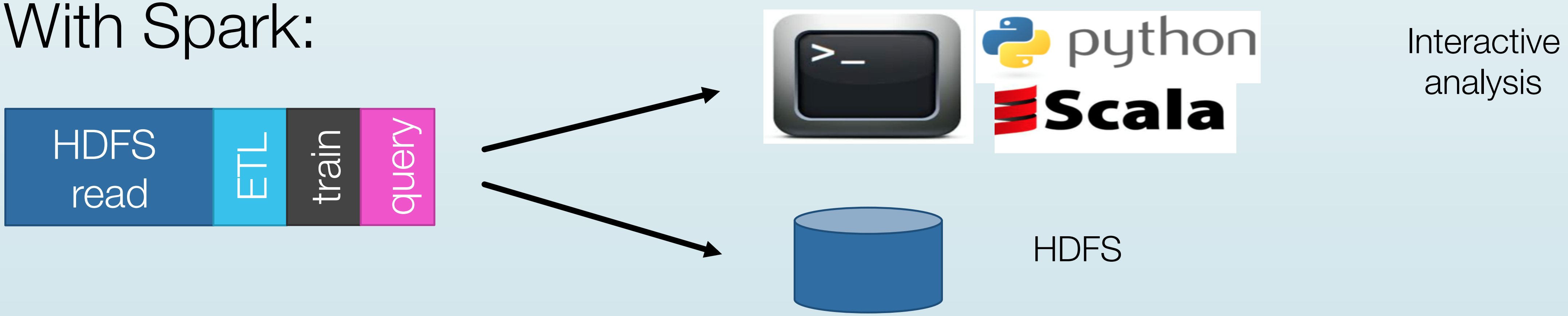
Spark's DAG Execution



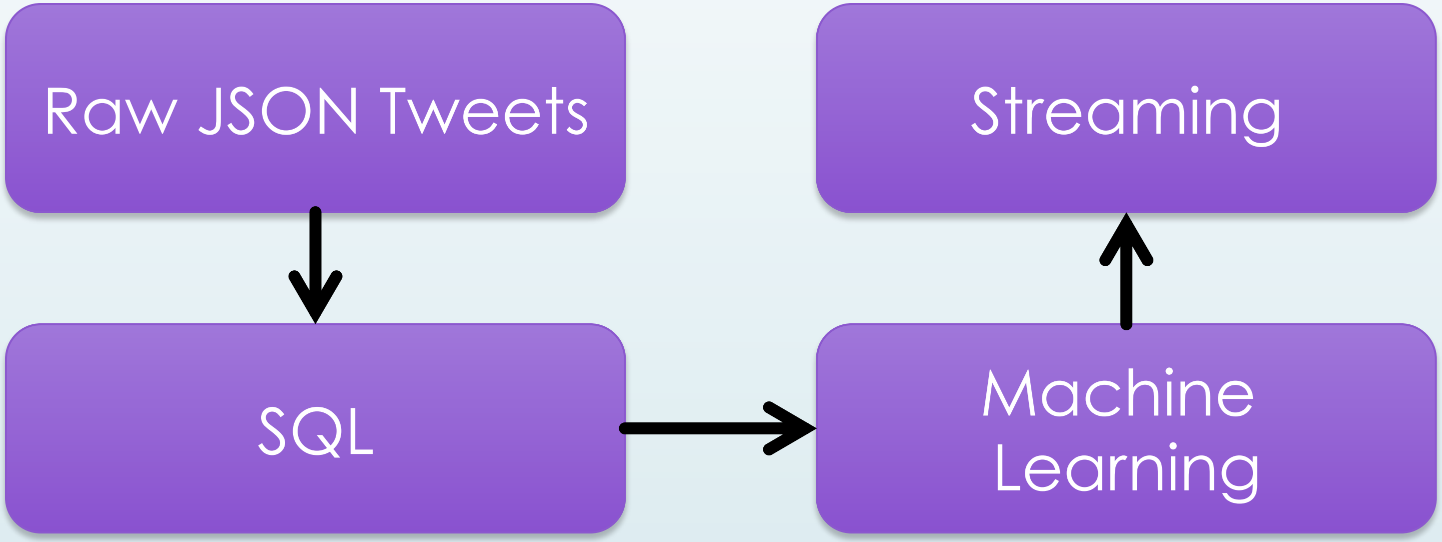
Why Application Developers love Spark

- Building a real-world big data application without and with Spark:

With Spark:



An Example App



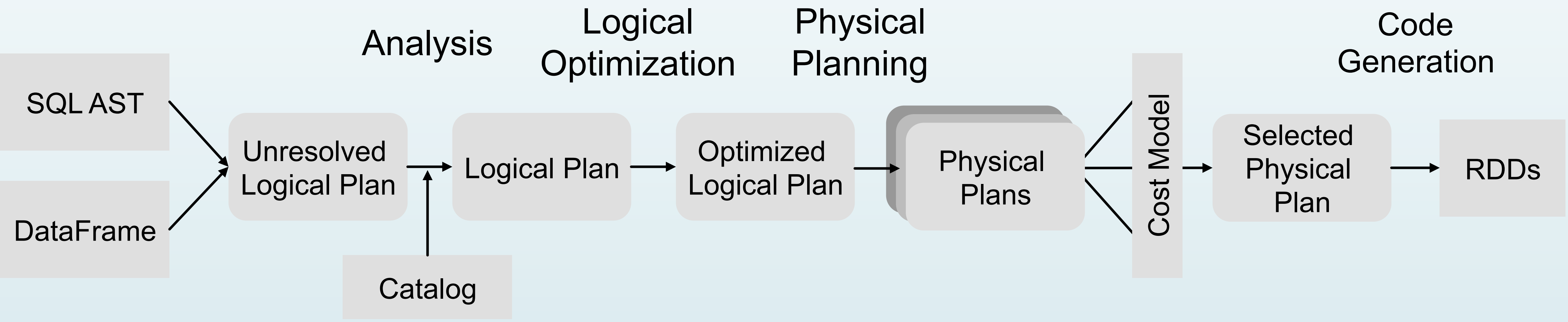
Fits on one slide

```
import org.apache.spark.sql._
val ctx = new org.apache.spark.sql.SQLContext(sc)
val tweets = sc.textFile("hdfs:/twitter")
val tweetTable = JsonTable.fromRDD(sqlContext, tweets, Some(0.1))
tweetTable.registerAsTable("tweetTable")

ctx.sql("SELECT text FROM tweetTable LIMIT 5").collect.foreach(println)
ctx.sql("SELECT lang, COUNT(*) AS cnt FROM tweetTable \
  GROUP BY lang ORDER BY cnt DESC LIMIT 10").collect.foreach(println)
val texts = sql("SELECT text FROM tweetTable").map(_.head.toString)

def featurize(str: String): Vector = { ... }
val vectors = texts.map(featurize).cache()
val model = KMeans.train(vectors, 10, 10)

sc.makeRDD(model.clusterCenters, 10).saveAsObjectFile("hdfs:/model")
val ssc = new StreamingContext(new SparkConf(), Seconds(1))

val model = new KMeansModel(
    ssc.sparkContext.objectFile(modelFile).collect())

// Streaming
val tweets = TwitterUtils.createStream(ssc, /* auth */)
val statuses = tweets.map(_.getText)
val filteredTweets = statuses.filter {
    t => model.predict(featurize(t)) == clusterNumber
}
filteredTweets.print()

ssc.start()
```

Why SparkSQL?

- SQL, SQL, SQL, …
  - Databricks says that 100% of their customers use some SQL
- Schema is very useful
  - Even in complex pipelines that process a lot of un/semi-structured data
- Separation of logical from physical plan is critical for performance and scalability

Plan Optimization & Execution



DataFrames and SQL share the same optimization/execution pipeline

DataFrame

1. A distributed collection of rows organized into named columns
2. An abstraction for selecting, filtering, aggregating and plotting structured data (*cf. R, Pandas, Ibis*)

Catalyst Optimizer: Tree Transformations

- Developers express tree transformations as `PartialFunction[TreeType,TreeType]`

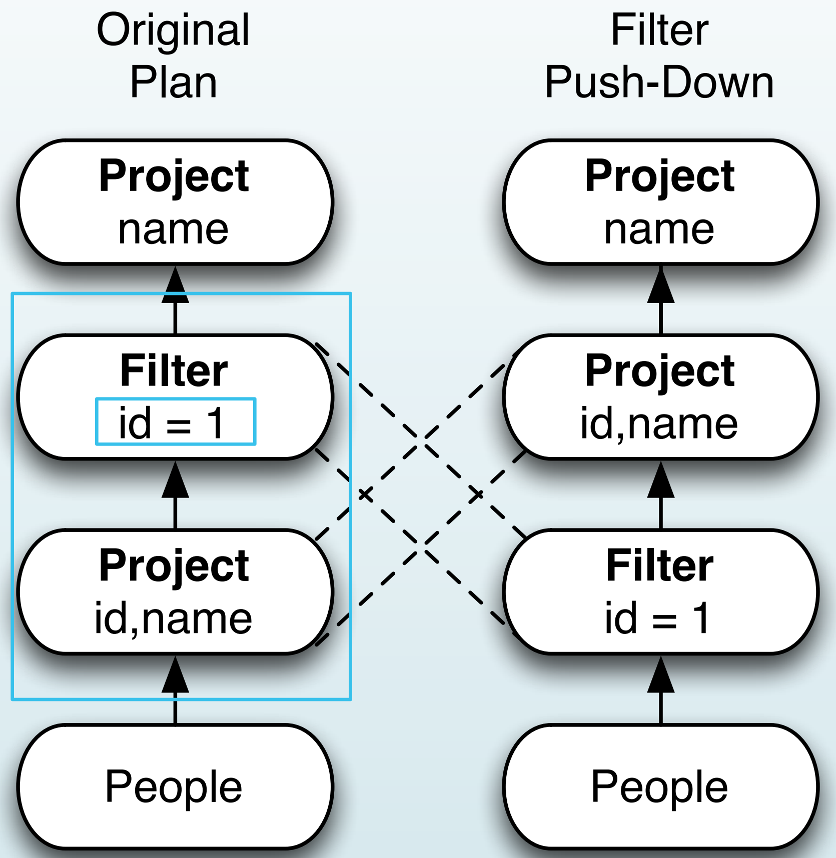
1. If the function **does apply** to an operator, that operator is **replaced** with the result.
2. When the function **does not apply** to an operator, that operator is **left unchanged**.
3. The transformation is **applied recursively to all children**.

Prior Work: Optimizer Generators

- Volcano / Cascades:
- Create a custom language for expressing rules that rewrite trees of relational operators.
- Build a compiler that generates executable code for these rules.

An Example Catalyst Transformation

1. Find filters on top of projections.
2. Check that the filter can be evaluated without the result of the project.
3. If so, switch the operators.



Filter Push Down Transformation

```
val newPlan = queryPlan transform {
  case f @ Filter(_, p @ Project(_, grandChild))
    if(f.references subsetOf grandChild.output) =>
  p.copy(child = f.copy(child = grandChild)
}
```

Community-Contributed Transformations

SPARK-3462 push down filters and projections into Unions #2345

Closed **koeninger** wants to merge 3 commits into `apache:master` from `mediacrossinginc:SPARK-3462`

Conversation 15 | Commits 3 | Files changed 2 | +110 −0

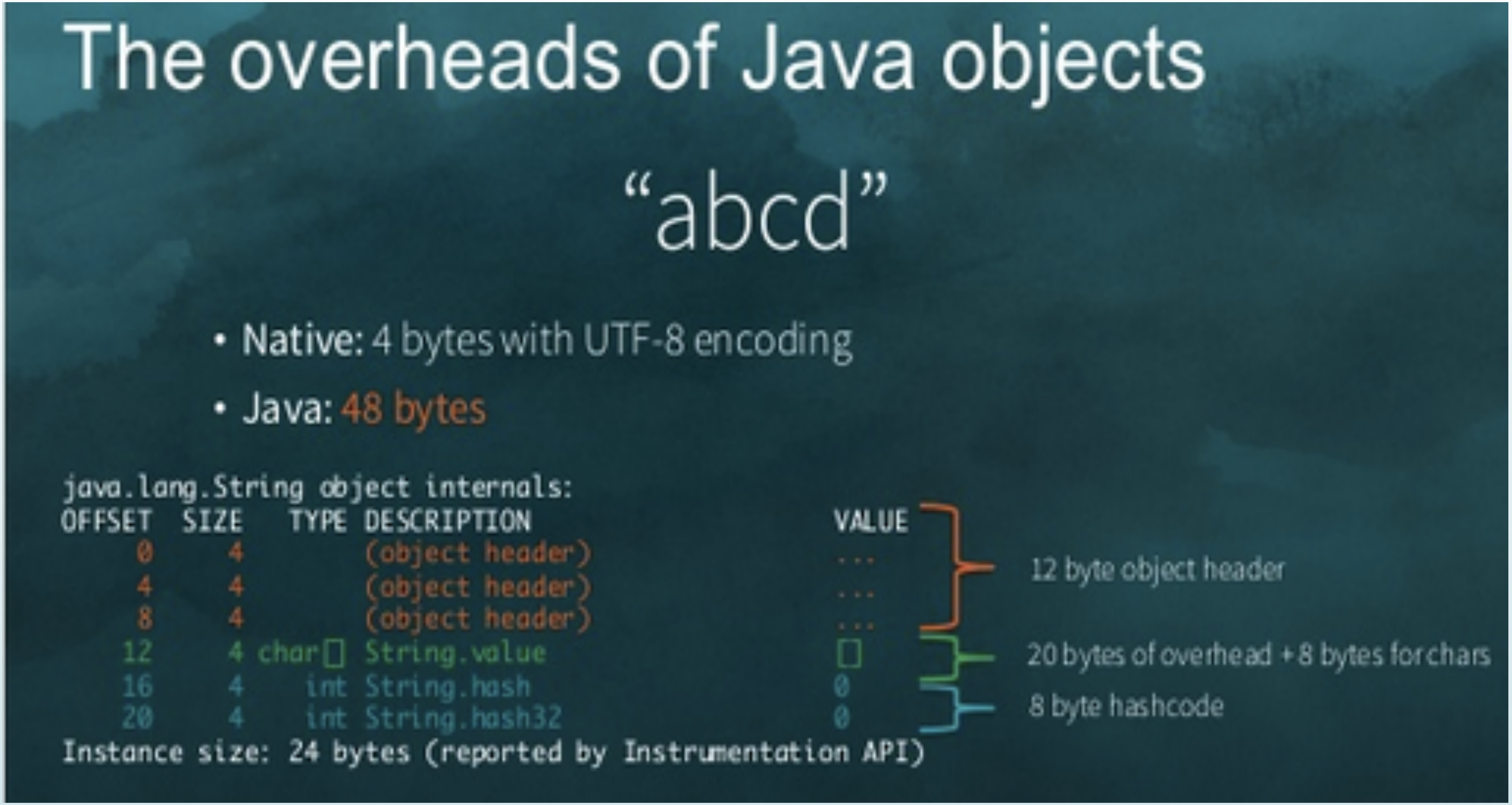
Showing **2 changed files** with **110 additions** and **0 deletions**.

Unified | Split

110 line patch took this user's query from "never finishing" to 200s.

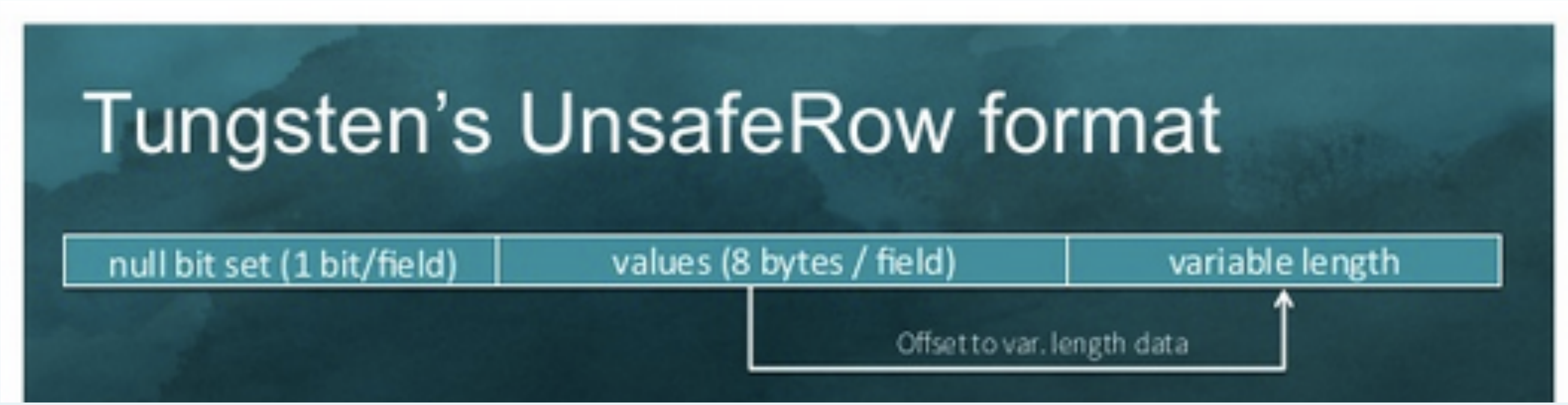
Project Tungsten: Getting Spark to Run Well on the JVM

- Overcoming JVM limitations:
  - **Memory Management and Binary Processing**: leveraging application semantics to manage memory explicitly and eliminate the overhead of JVM object model and garbage collection
  - **Cache-aware computation**: algorithms and data structures to exploit memory hierarchy
  - **Code generation**: using code generation to exploit modern compilers and CPUs


The overheads of Java objects
"abcd"
• Native: 4 bytes with UTF-8 encoding
• Java: 48 bytes
java.lang.String object internals:
OFFSET SIZE TYPE DESCRIPTION VALUE
0 4 (object header) ...
4 4 (object header) ...
8 4 (object header) ...
12 4 char[] String.value []
16 4 int String.hash 0
20 4 int String.hash32 0
Instance size: 24 bytes (reported by Instrumentation API)
12 byte object header
20 bytes of overhead + 8 bytes for chars
8 byte hashcode

Use sun.misc.Unsafe

- JVM internal API
- Can manipulate memory without safety checks



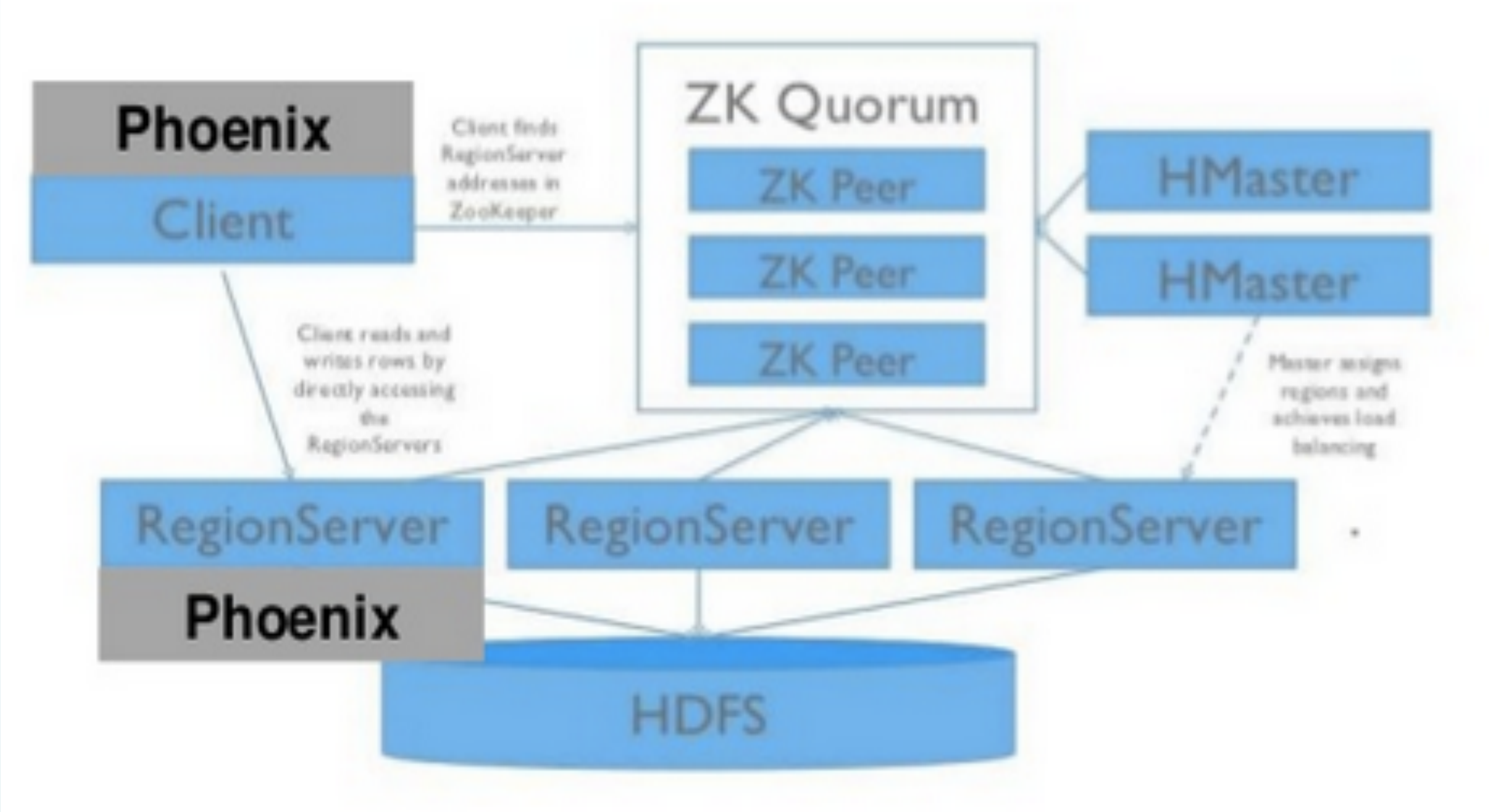
- Null bits
- Inline fixed-length values
- Align on 8-byte word boundaries

Apache Phoenix

The Phoenix Approach

- SQL compiler and execution engine for HBase
  - Query engine transforms SQL into native HBase APIs: put, delete, parallel scans (instead of, say, MapReduce)
- Supports features not provided by HBase: Secondary Indexing, Multi-tenancy, simple Hash Join, etc.

Phoenix Architecture



Open (Research) Challenges

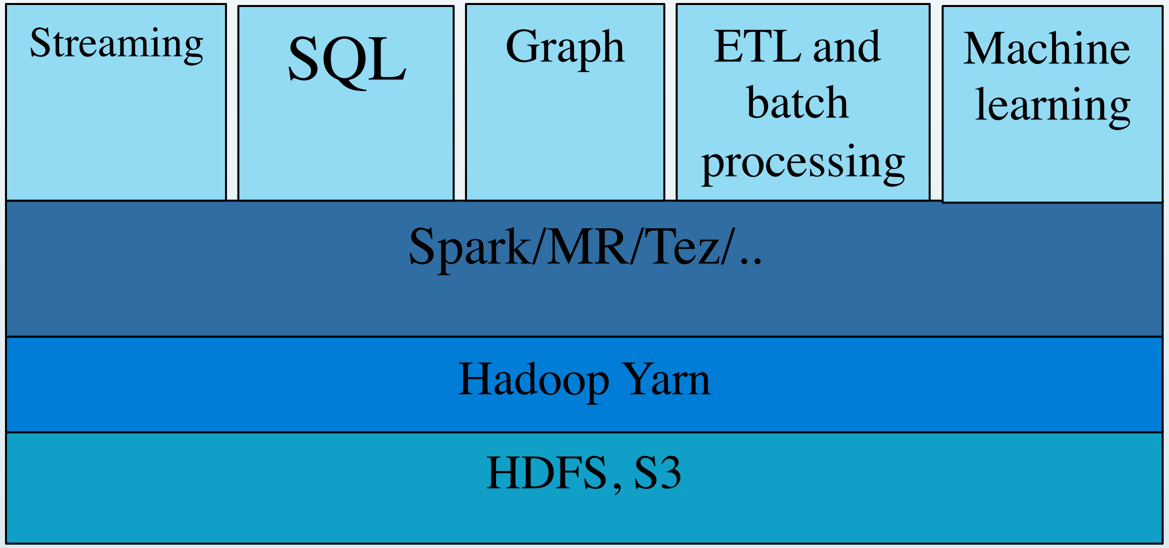
Challenge 1: Query optimization

- Cost-based optimizer relies on
  - Statistics over base relations
  - Formulas for cost estimation
  - Rules for plan enumeration
- Problems:
  - Stats not reliable, do not own the data
  - Prominent use of UDFs
  - Independence assumption between predicates do not hold
  - More nested data, harder to estimate selectivities
  - Bad plans over big data may run "forever"

Defer more cost-based decisions to run-time; robust, adaptive query optimization

Challenge 2: Multi-framework environment

- No single framework owns the data!
- Multiple frameworks, with different resource requirements



- How to share the data?
- How to share resources?
- How to work together seamlessly?

Challenge 3: Transactions and analytics in one system

- HDFS is a problem for transactional workloads
  - Workarounds do not lend itself to high-performance OLAP
  - Object-stores

- Interesting combinations are emerging
  - Hive LLAP + Phoenix, Splice Machine + Spark
- Need more tightly integrated solutions
- Need an updatable, fast, distributed file system

References

- http://www.slideshare.net/enissoz/hbase-and-hdfs-understanding-filesystem-usage
- https://www.mapr.com/blog/in-depth-look-hbase-architecture
- Apache Drill. http://drill.apache.org/.
- Apache Phoenix. http://phoenix.apache.org/.
- Hive on spark. https://cwiki.apache.org/confluence/display/Hive/Hive+on+Spark .
- Splice machine. http://www.splicemachine.com/.
- Teradata query grid. http://www.teradata.com/Teradata- QueryGrid/ #tabbable=0&tab1=0&tab2=0.
- M. Kornacker, A. Behm, V. Bittorf, T. Bobrovytsky, C. Ching, A. Choi, J. Erickson, M. Grund, D. Hecht, M. Jacobs, I. Joshi, L. Kuff, D. Kumar, A. Leblang, N. Li, I. Pandis, H. Robinson, D. Rorke, S. Rus, J. Russell, D. Tsirogiannis, S. Wanderman-Milne, and M. Yoder. "Impala: A Modern, Open-Source SQL Engine for Hadoop." In Proc. CIDR, 2015.
- Y. Huai, A. Chauhan, A. Gates, G. Hagleitner, E. N. Hanson, O. O'Malley, J. Pandey, Y. Yuan, R. Lee, and X. Zhang. "Major technical advancements in apache hive." In Proc. SIGMOD, 2014.

References (cont.)

- A. Ailamaki, D. J. DeWitt, M. D. Hill, and M. Skounakis. "Weaving Relations for Cache Performance." In Proc. of the 27th International Conference on Very Large Data Bases, 2001.
- Y. He, R. Lee, Y. Huai, Z. Shao, N. Jain, X. Zhang, and Z. Xu. "RCFile: A fast and space-efficient data placement structure in MapReduce-based warehouse systems." In Proc. of ICDE, 2011.
- H. Lim, H. Herodotou, and S. Babu. "Stubby: a transformation-based optimizer for MapReduce workflows." PVLDB, 2012.
- T. Neumann. "Efficiently compiling efficient query plans for modern hardware." PVLDB, 2011.
- D. Simmen, E. Shekita, and T. Malkemus. "Fundamental techniques for order optimization." In Proc. of ACM SIGMOD, 1996.
- A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang, S. Anthony, H. Liu, and R. Murthy." Hive - A Petabyte Scale Data Warehouse Using Hadoop." In ICDE, 2010.
- T. Willhalm, N. Popovici, Y. Boshmaf, H. Plattner, A. Zeier, and J. Schaffner. "SIMD-scan: ultra fast in- memory table scan using on-chip vector processing units." PVLDB, 2, 2009.

References (cont.)

- V. Raman, G. Attaluri, R. Barber, N. Chainani, D. Kalmuk, V. KulandaiSamy, J. Leenstra, S. Lightstone, S. Liu, G. M. Lohman, T. Malkemus, R. Mueller, I. Pandis, B. Schiefer, D. Sharpe, R. Sidle, A. Storm, and L. Zhang. "DB2 with BLU Acceleration: So much more than just a column store." PVLDB, 6, 2013.
- A. Abouzeid, K. Bajda-Pawlikowski, D. J. Abadi, A. Rasin, and A. Silberschatz. "HadoopDB: An Architectural Hybrid of MapReduce and DBMS Technologies for Analytical Workloads." PVLDB, 2009.
- A. Abouzied, D. J. Abadi, and A. Silberschatz. "Invisible loading: Access-driven data transfer from raw files into database systems." In EDBT, 2013.
- M. Amburst, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi, and M. Zaharia. "Spark SQL: Relational data processing in Spark." In ACM SIGMOD, 2015.
- V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler. Apache Hadoop YARN: Yet another resource negotiator. In SOCC, 2013.

References (cont.)

- K. Bajda-Pawlikowski, D. J. Abadi, A. Silberschatz, and E. Paulson. "Efficient processing of data warehousing queries in a split execution environment." In ACM SIGMOD, 2011.
- P. Boncz. Vortex: Vectorwise goes Hadoop. http://databasearchitects.blogspot.com/2014/05/vectorwise- goes-hadoop.html.
- L. Chang, Z. Wang, T. Ma, L. Jian, L. Ma, A. Goldshuv, L. Lonergan, J. Cohen, C. Welton, G. Sherry, and M. Bhandarkar. "HAWQ: A massively parallel processing SQL engine in hadoop." In ACM SIGMOD, 2014.
- G. Graefe. "Encapsulation of parallelism in the Volcano query processing system." In ACM SIGMOD, 1990.
- S. Gray, F. Özcan, H. Pereyra, B. van der Linden, and A. Zubiri. "IBM Big SQL 3.0: SQL-on-Hadoop without compromise." http://public.dhe.ibm.com/common/ ssi/ecm/en/sww14019usen/ SWW14019USEN.PDF, 2014.
- F.Özcan, D. Hoa, K. S. Beyer, A. Balmin, C. J. Liu, and Y. Li. "Emerging trends in the enterprise data analytics: Connecting Hadoop and DB2 warehouse." In ACM SIGMOD, 2011.

References (cont.)

- S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, and T. Vassilakis. "Dremel: Interactive analysis of web-scale datasets." PVLDB, 2010.
- S. Padmanabhan, T. Malkemus, R. C. Agarwal, and A. Jhingran. "Block oriented processing of relational database operations in modern computer architectures." In ICDE, 2001.
- B. Saha, H. Shah, S. Seth, G. Vijayaraghavan, A. Murthy, and C. Curino. "Apache Tez: A unifying framework for modeling and building data processing applications." In ACM SIGMOD, 2015.
- P. Seshadri, H. Pirahesh, and T. Y. C. Leung. "Complex query decorrelation." In ICDE, 1996.
- A. Floratou, U. F. Minhas, and F. Özcan. "SQL-on- Hadoop: Full circle back to shared-nothing database architectures." PVLDB 7(12), 2014.
- M. Traverso. Presto: Interacting with petabytes of data at Facebook. https://www.facebook.com/notes/facebook- engineering/presto-interacting-with-petabytes-of-data-at-facebook/10151786197628920.
- S. Wanderman-Milne and N. Li. Runtime code generation in Cloudera Impala. IEEE Data Eng. Bull., 2014.

References (cont.)

- R. S. Xin, J. Rosen, M. Zaharia, M. J. Franklin, S. Shenker, and I. Stoica. "Shark: SQL and rich analytics at scale." In ACM SIGMOD, 2013.
- M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. "Spark: Cluster computing with working sets." In HotCloud, 2010.
- C. Zuzarte, H. Pirahesh, W. Ma, Q. Cheng, L. Liu, and K. Wong. "WinMagic : Subquery elimination using window aggregation." In ACM SIGMOD, 2003.
- F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. Gruber. "Bigtable: A Distributed Storage System for Structured Data.", In OSDI 2006
- B. Chattopadhyay, L. Lin, W. Liu, S. Mittal, P. Aragonda, V. Lychagina, Y. Kwon, and M. Wong. "Tenzing: A SQL Implementation on the MapReduce Framework." In VLDB 2011.
- T. Nykiel, M. Potamias, C. Mishra, G. Kollios, and N. Koudas. "MRShare: Sharing Across Multiple Queries in MapReduce." In VLDB 2010.

References (cont.)

- G. Wang and C.-Y. Chan. "Multi-Query Optimization in MapReduce Framework." In VLDB, 2013.
- F. Afrati and J. Ullman. "Optimizing Multiway Joins in a Map-Reduce Environment." In TKDE 2011.
- Y. Kwon, M. Balazinska, B. Howe, and J. Rolia. "SkewTune: Mitigating Skew in MapReduce Applications." In SIGMOD 2012.
- M. Eltabakh, F. Özcan, Y. Sismanis, P. J. Haas, H. Pirahesh, and J. Vondrak, "Eagle-eyed elephant: split-oriented indexing in Hadoop", in EDBT 2014.
- M. Eltabakh, Y. Tian, F. Özcan, R. Gemulla, A. Krettek, and J. McPherson, "CoHadoop: Flexible Data Placement and Its Exploitation in Hadoop", in PVLDB 4(9), 2011.
- J. Shi, Y. Qiu, U. F. Minhas, L. Jiao, C. Wang, B. Reinwald, and F. Özcan, "Clash of the Titans: MapReduce vs. Spark for Large Scale Data Analytics" in PVLDB 8(13), 2015
- J. Dittrich, J-A. Quiane-Ruiz, A. Jindal, Y. Kargin, V. Setty, and J. Schad, "Hadoop++: Making a Yellos Elephant Run Like a Cheetah (without it even noticing)", in PVLDB 3(1-2), 2010.

References (cont.)

- D. Jiang, B. C. Ooi, L. Shi, and S. Wu, "The Performance of MapReduce: An In-depth Study", in PVLDB 3(1-2), 2010.
- D. J. DeWitt, R. V. Nehme, S. Shankar, J. Aguilar-Saborit, A. Avanes, M. Flasza, and J. Gramling, "Split Query Processing in Polybase", in SIGMOD 2013.
- M. Stonebraker, D. J. Abadi, D. J. DeWitt, S. Madden, E. Paulson, A. Pavlo, and A. Rasin, "MapReduce and parallel DBMSs: Friends or Foes?" CACM, 53(1):64–71, 2010.
- A. Floratou, J. M. Patel, E. J. Shekita, and S. Tata, "Column-oriented Storage Techniques for MapReduce", in PVLDB, 4(7):419–429, 2011
- S. Harris, A. Sundararajan, E. Branish, and K. Chen, "Blistering Fast SQL Access to Hadoop using IBM BigInsights 3.0 with Big SQL 3.0"
- S. Blanas and et al., "A comparison of join algorithms for log processing in mapreduce", in SIGMOD 2010.

References (cont.)

- HDFS caching, http://hadoop.apache.org/docs/current/hadoopproject-dist/hadoop-hdfs/CentralizedCacheManagement.html.
- S. Babu and H. Herodotou, “Massively Parallel Databases and MapReduce Systems”, in Foundations and Trends in Databases 5(1), 2013.
- N. Bruno, Y. Kwon, and M-C Wu, “Advanced Join Strategies for Large-Scale Distributed Computation”, in PVLDB 7(13), 2014
- K. Karanasos, A. Balmin, M. Kutsch, F. Özcan, V. Ercegovac, C. Xia, and J. Jackson, “Dynamically optimizing queries over large scale data platforms”, in SIGMOD 2014
- J. Dean, and S. Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters”, in OSDI 2004.

Thank you!