



Contributed by

In this blog post, we'll take a look at the inner workings of Apache Drill, learn what services are involved, and find out what happens in Apache Drill when we submit a query.



Carol McDonald  
(/blog/author/carol-mcdonald/)



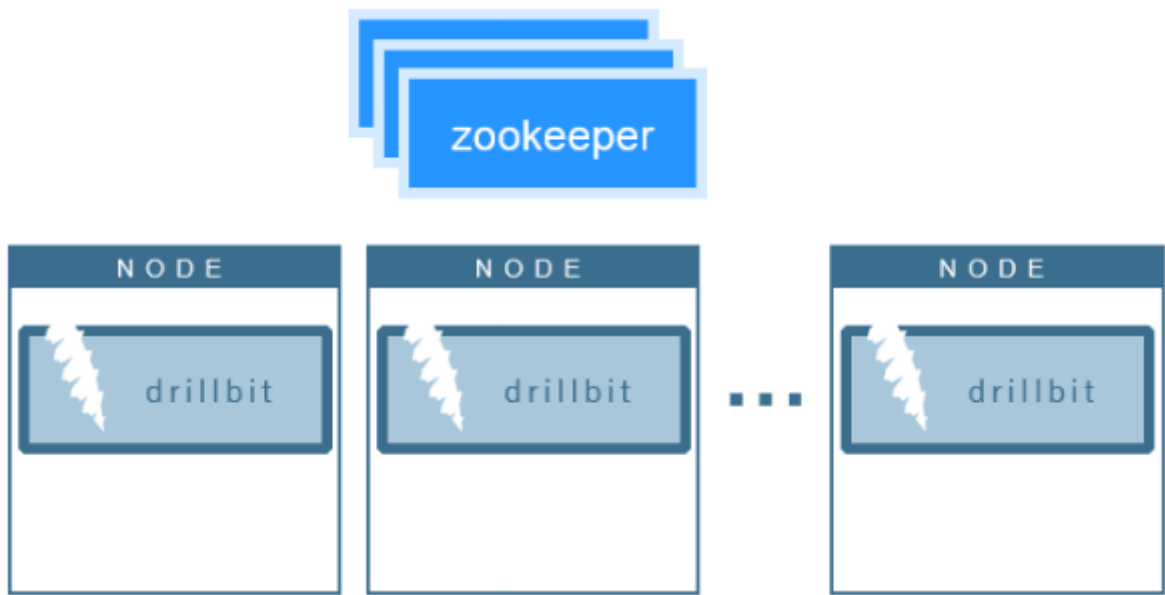
Jonathan Allen  
(/blog/author/jon-allen/)

**Editor's note:** This post is derived from the DA 400 Introduction to SQL Analytics with Apache Drill – to learn more about Apache Drill, be sure to sign-up for our free on-demand training course Apache Drill Essentials (/training/on-demand/da-400-introduction-to-sql-analytics-with-apache-drill/). Course material was developed by Jonathan Allen (/blog/author/jon-allen/).

## Core Modules within a Drillbit

Drill consists of a Daemon service called the Drillbit. The Drillbit service can be installed on an individual server, or on any or all nodes of a Hadoop cluster. When installed on a Hadoop cluster, Drill performs distributed SQL query execution and optimizes for data locality, providing for unparalleled SQL performance and reliability when scaling to an enterprise level.

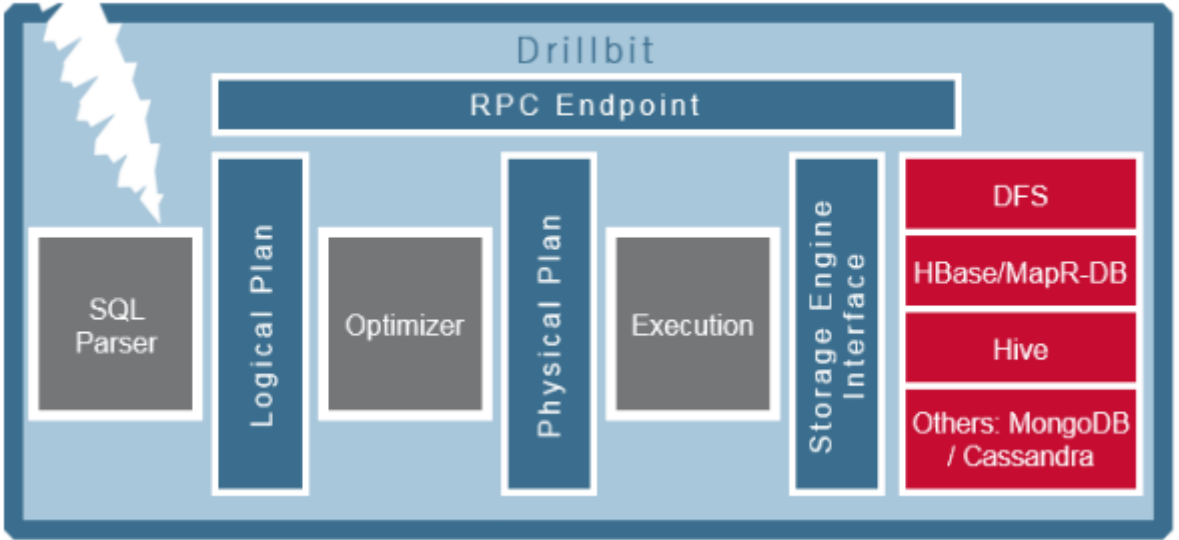
ZooKeeper maintains ephemeral cluster membership information. The Drillbits use ZooKeeper to find other Drillbits in the cluster, and the client uses ZooKeeper to find Drillbits to submit a query. Drill uses its own SQL execution engine, and does not rely on MapReduce, Spark or Tez like other SQL on Hadoop solutions.



### Key Core Modules within a Drillbit

### Each Drillbit consists of the following key components:

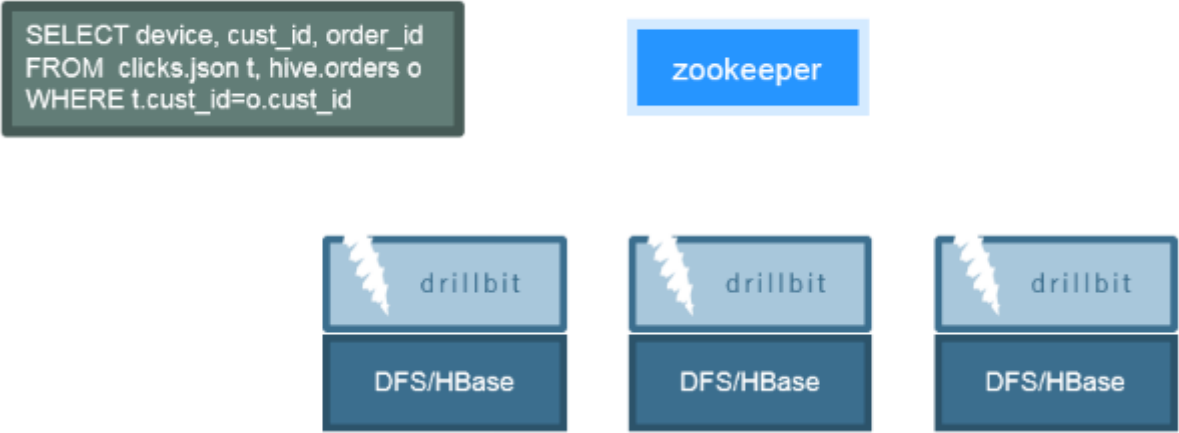
- The **RPC end point** exposes a low overhead protobuf-based RPC protocol to communicate with the clients and receive queries.
- The **SQL parser** parses incoming queries, and is based on the open source framework, Calcite. The output of the parser component is a language agnostic, computer-friendly logical plan that represents the query.
- Drill serves as a query layer on top of several data sources. **Storage plugin interfaces** in Drill represent the abstractions that Drill uses to interact with the data sources. Storage plugins provide Drill with the metadata available in the data source, the interfaces for Drill to read from and write to data sources, and the location of data and a set of optimization rules to help with efficient and faster execution of Drill queries on a specific data source.
- On Hadoop, storage plugin information is saved in the Zookeeper. Drill provides **storage plugins** for files, for HBase and MapR-DB tables, and for Hive. The plugins are extensible, allowing you to write new plugins for any additional data sources you may have. Additional plugins already available include MongoDB and Cassandra.
- Drill integrates with Hive as a storage plugin because Hive provides a metadata abstraction layer on top of files, HBase or MapR-DB, and provides libraries to read data and operate on these sources. When users query files and HBase or MapR-DB tables with Drill, they can do it directly or go through Hive if they have metadata defined there. Drill integration with Hive is only for metadata. Drill does not invoke the Hive execution engine for any requests.
- Certain Drill services are extensible, and can take advantage of third party plugins including: custom client APIs such as Python, a custom DSL layer such as Pig, or you can extend the storage plugin interface to include additional data sources. The SQL layer in Drill is extensible and you can create custom user defined functions and user defined aggregate functions using a high- performance Java API included with Drill.



Query Execution

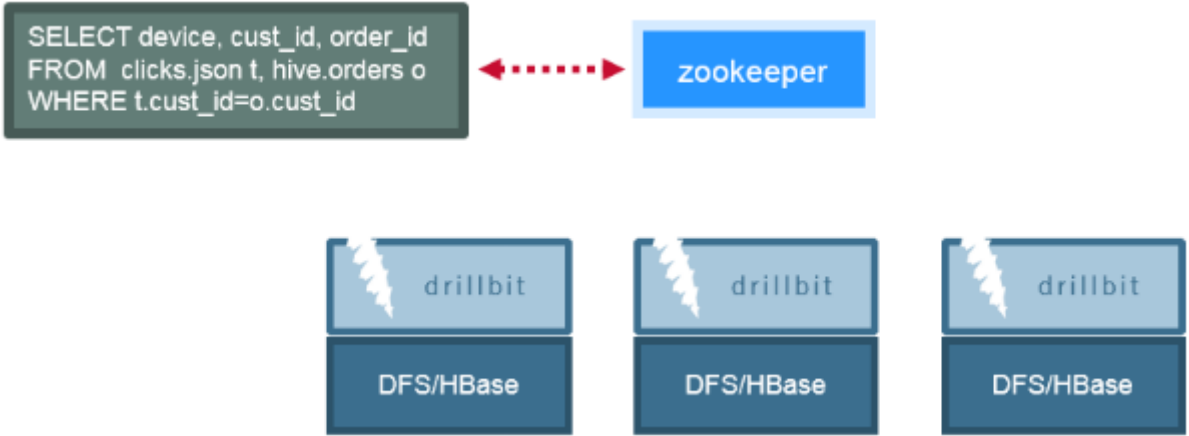
Drill is a flexible, reliable SQL query engine. It can accept query input from a wide variety of data sources and query requests from a JDBC, ODBC, or REST interface, and from a C++ or Java API.

1. Query comes to any Drillbit (JDBC, ODBC, REST, C++, Java)

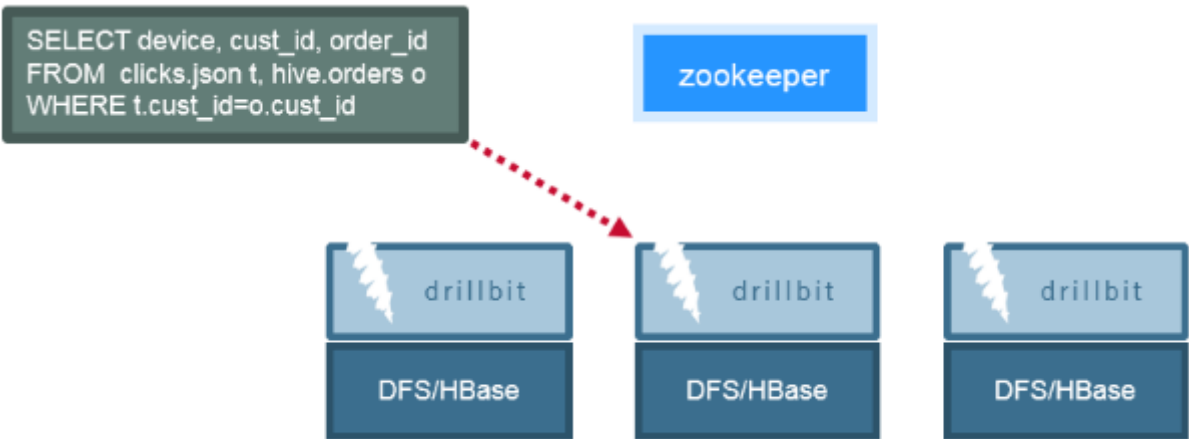


When a client issues a query to Drill, the execution process typically consists of the following steps:

- The client first contacts the zookeeper. Zookeeper returns to the client the available Drillbits in the cluster to which the query can be submitted. Clients can connect directly to a specific Drillbit, though this is generally not recommended due to the hardcoding that can happen in client.



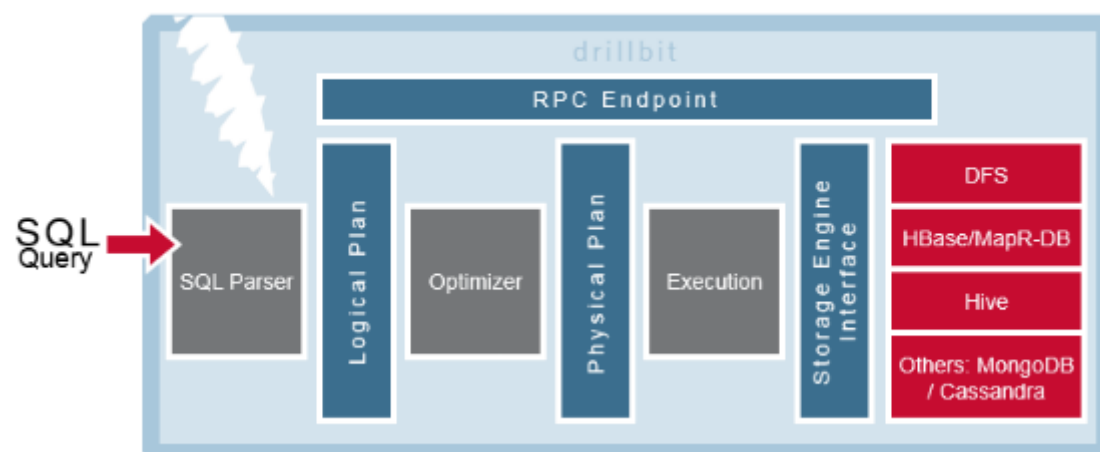
- The Drillbit that accepts the initial query becomes the Foreman for the request.
- Any Drillbit in the cluster can accept queries from clients. There is no master-slave concept. Every Drillbit contains all services and capabilities of Drill, allowing Drill to scale to thousands of users and thousands of nodes.



Query Execution Plan

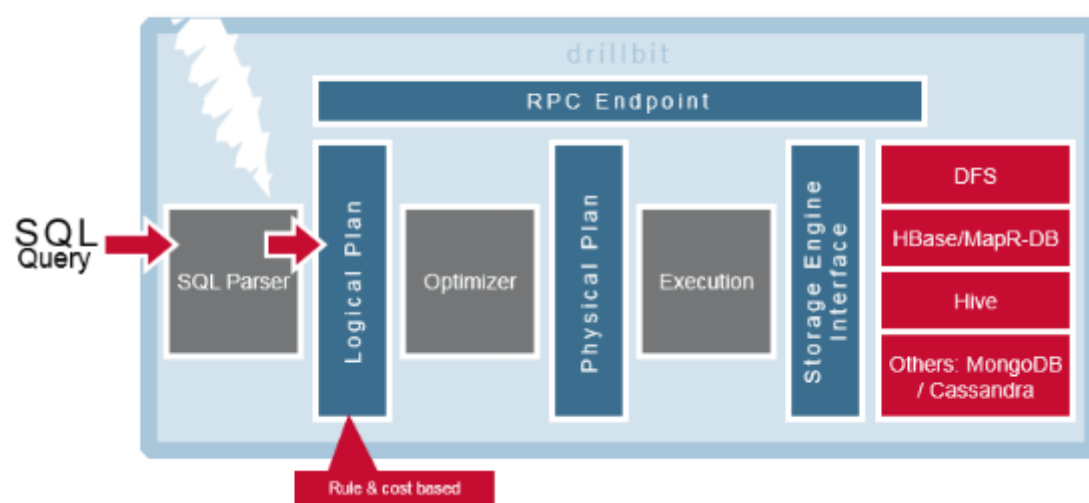
The Foreman parses the query and generates a distributed execution plan based on query optimization and locality. The Drillbit is capable of parsing a provided query into a logical plan.

## 2. Drillbit generates execution plan based on query optimization & locality



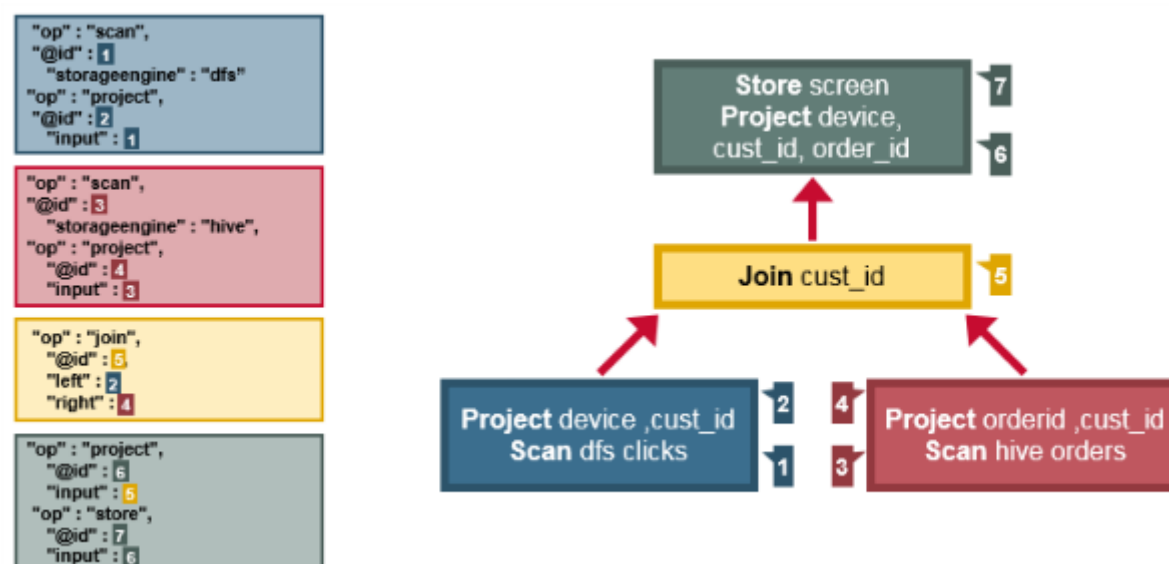
The logical plan describes the abstract dataflow of a query. The logical plan is a computer friendly description of the query that is language agnostic.

## 3. Logical Plan: What we want to do (language agnostic, computer friendly)



## Logical Plan in JSON

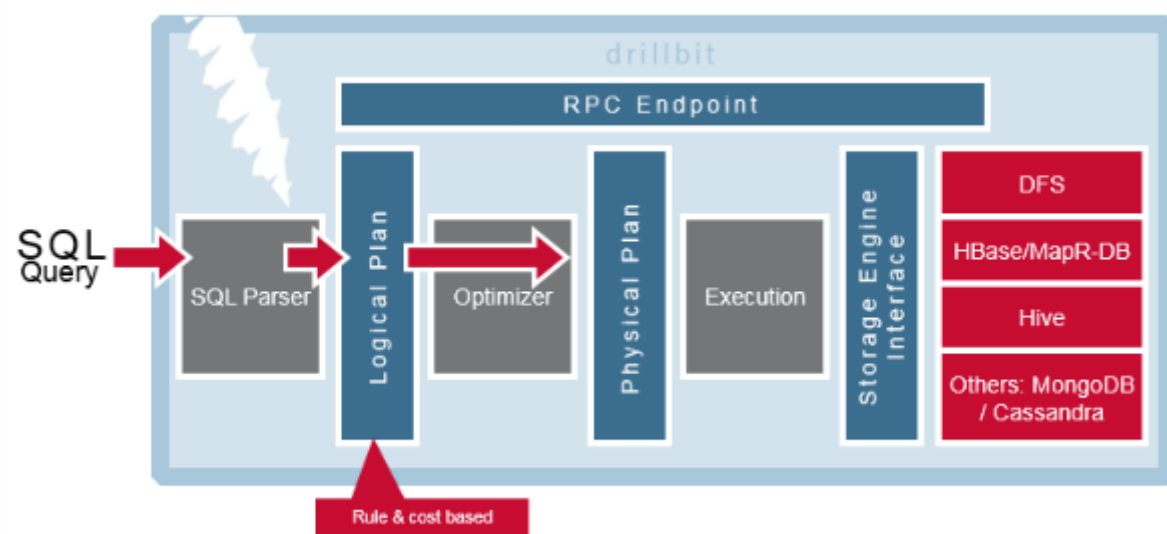
The logical plan is a tree. In the plan shown below, the bottom level consists of two scans and two projections. A project returns a particular field subset of the incoming stream of data corresponding to the expressions provided. The "projections" parameter specifies the values to be retained in the output records. The next layer contains a join of the output from the previous step. In this case, output from the Hive orders table and the dfs clicks file. The top layer has one projection and a screen store for the output.



Once a query is parsed into a logical plan, the Drill optimizer determines the most efficient execution plan using a variety of rule-based and cost-based techniques, translating the logical plan into a physical plan.

The physical plan, also called the execution plan, is a description of the physical operations the execution engine will undertake to get the desired result.

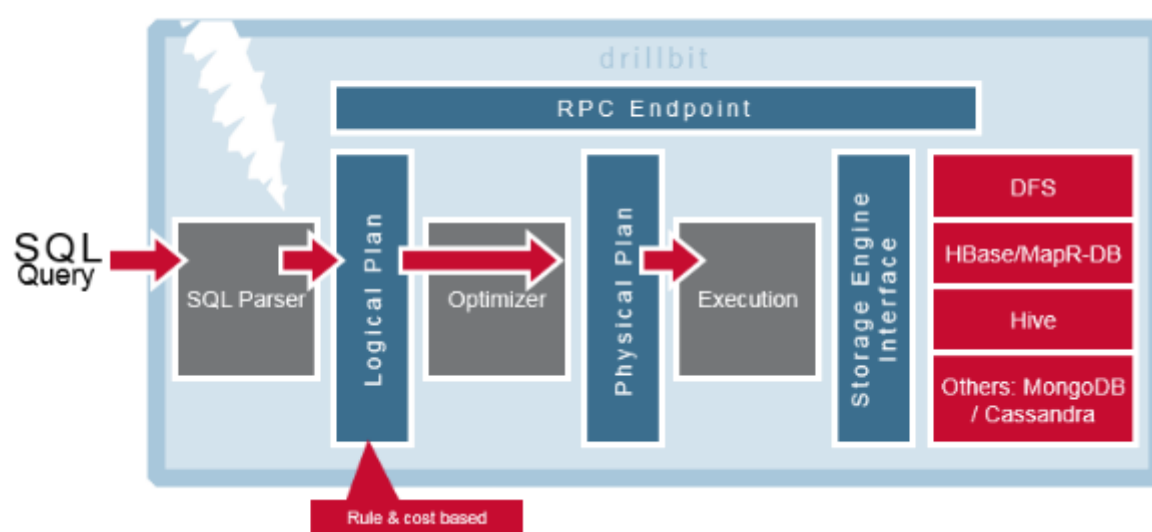
The physical plan is optimized for performance, and Drill considers data locality while determining the best physical plan. To maximize data locality and efficiency of performance, it is important to either run Drill on all nodes in the cluster, or to locate the data used for queries on the nodes that are running Drill.



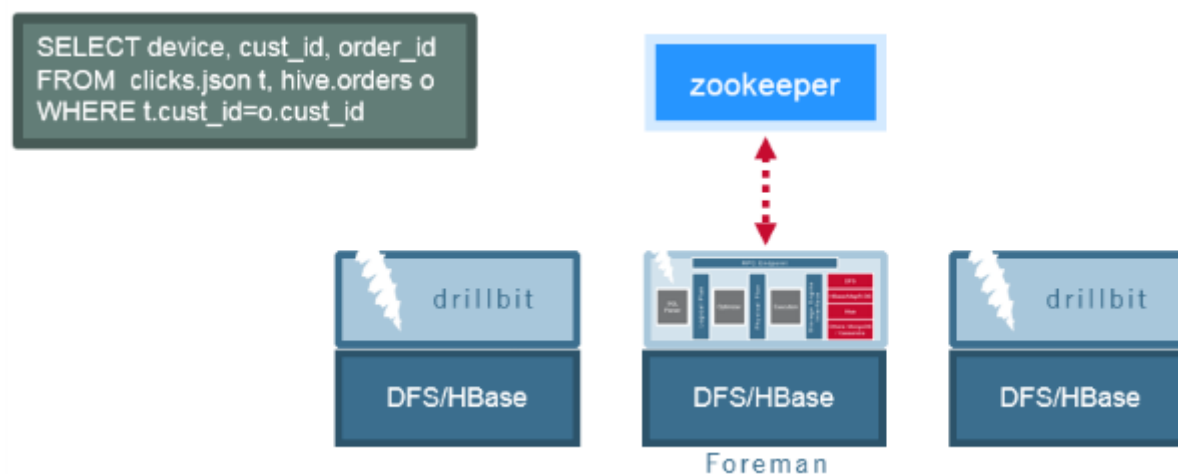
Once a physical plan is generated, it is then rendered into a set of detailed executional plan fragments, or EPFs. Execution Planning defines where the plan will be executed, and which Drillbits will execute which fragments of the plan.

This rendering is based on available resources, cluster load, query priority and detailed information about data distribution. In the case of large clusters, a subset of nodes will be responsible for rendering the plan fragments.

#### 4. Physical Plan: How we want to do it (the best way we can tell)

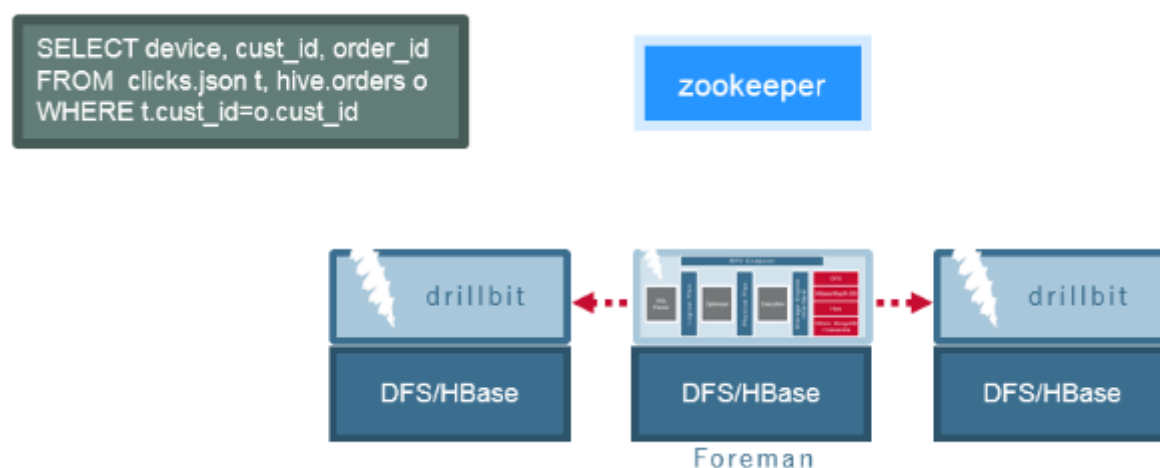


The foreman gets a list of available Drillbit nodes in the cluster from ZooKeeper, and determines the appropriate nodes to execute various query plan fragments to maximize data locality.



The foreman schedules the execution of query fragments on individual nodes according to the execution plan.

#### 5. Execution Plan (fragments): **Where** we want to do it

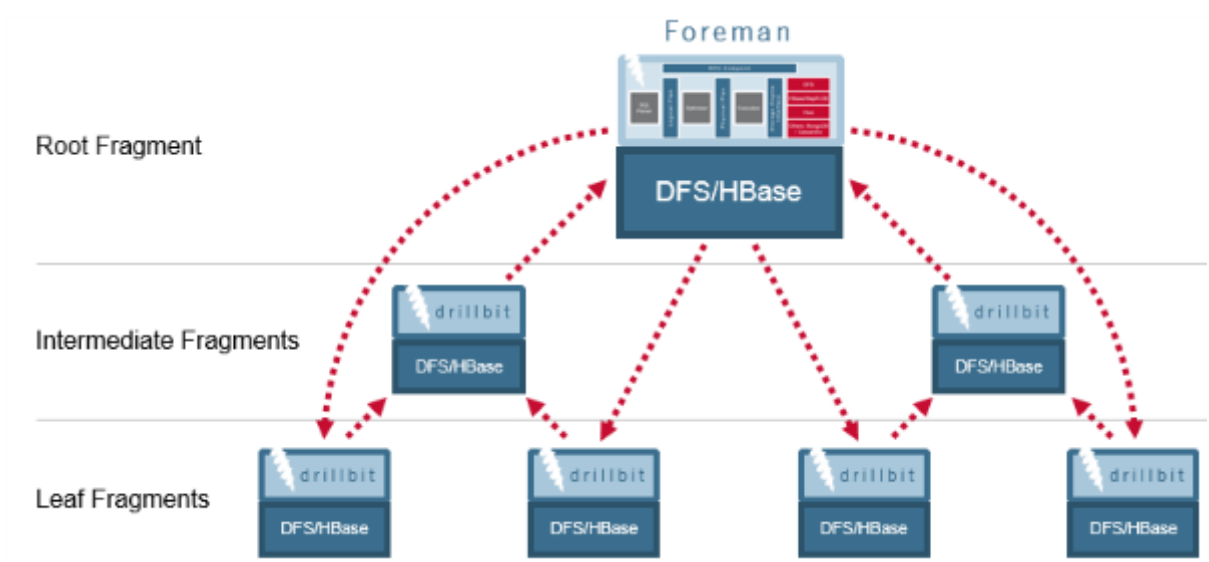


The physical plan is a DAG, or directed acyclic graph, of physical operators, and each child/parent relationship implies how data flows through the graph.

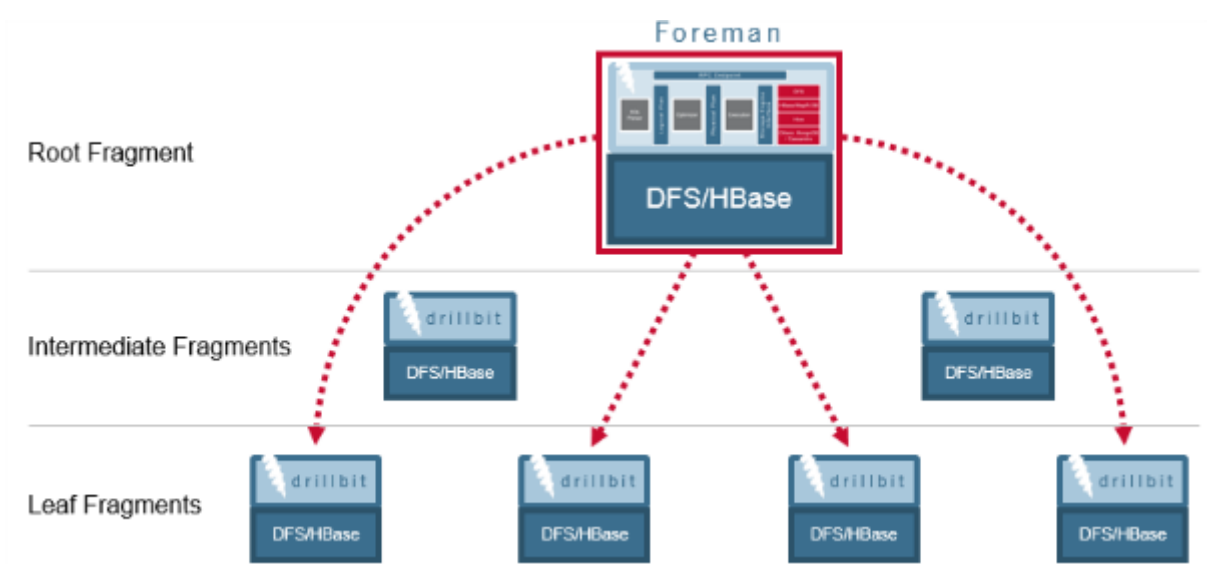
In the multi-level execution tree, each node represents a different Drillbit process and they each depend on each other's results for computation.

With the physical plan, the parallelizer in the Foreman transforms this plan into a number of fragments. Each fragment is simply a self-contained Physical plan that is designed to run on a Drillbit node. In any given Physical plan, there will be only one Root Fragment that runs on the Foreman Drillbit, possibly one or more Leaf fragments and possibly one or more intermediate fragments.

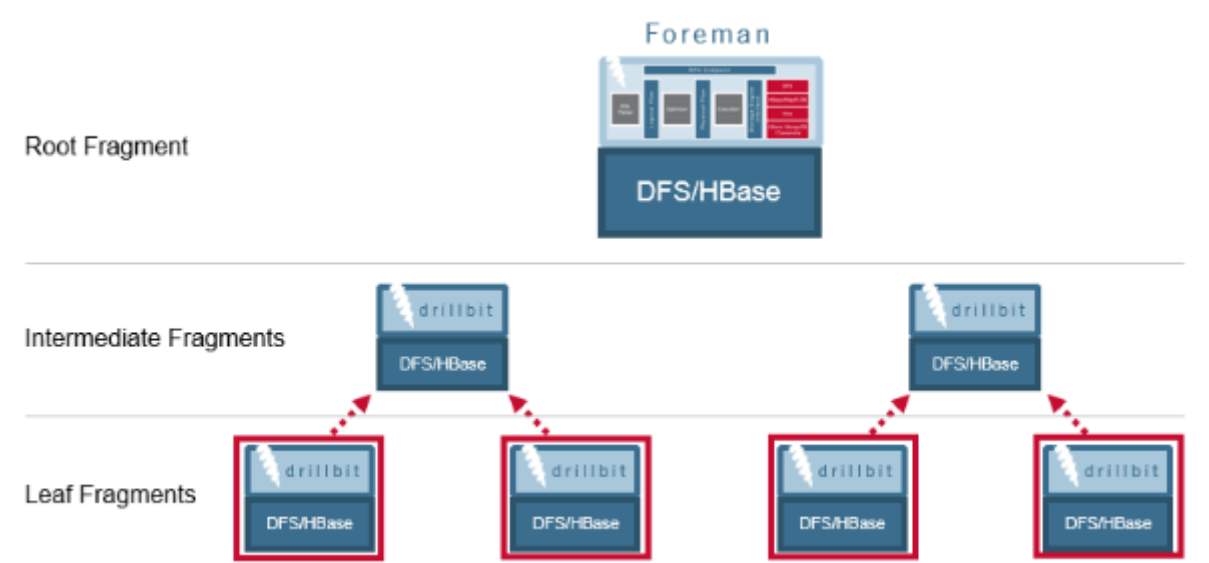
Query execution starts with each Drillbit being provided with one or more execution plan fragments, associated with the plan. A portion of these fragments may be identified as initial **Leaf** EPFs and thus they are executed immediately, while other **Intermediate** EPFs are executed as data flows into them.



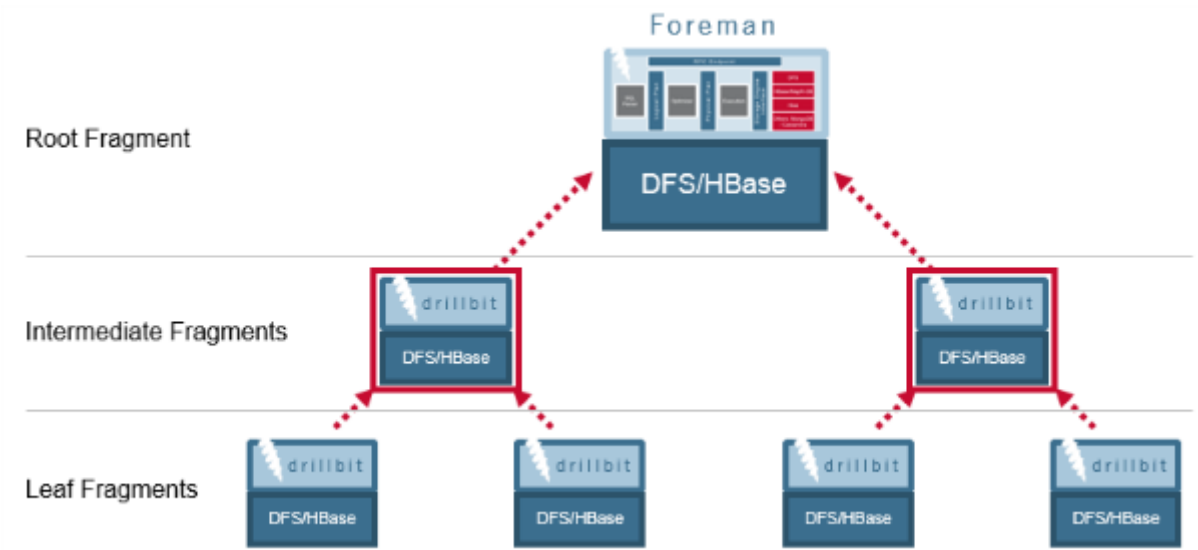
Each execution plan has one root fragment that runs on the Foreman, effectively forming the root node of a multi-level execution tree. The Foreman will direct processing to appropriate additional Drillbits on other nodes, to maximize locality.



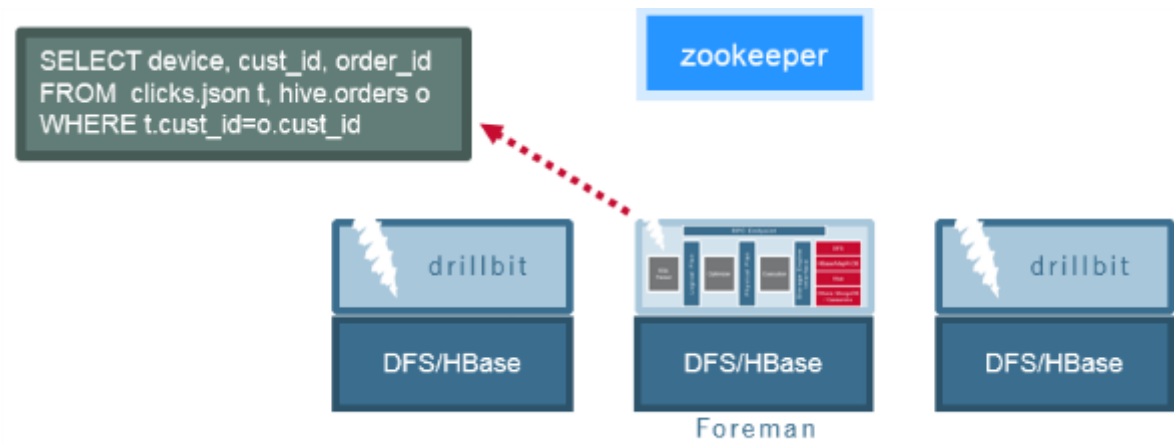
Leaf fragments use their storage engine interface to scan respective data sources. The leaf nodes run query fragments on the data sources and send the results up the tree.



Intermediate nodes aggregate the results from leaf nodes and send the results up the tree. Intermediate fragments perform a parallel aggregation of partial results, and start when they receive data from their children.



The foreman finally returns results back to the client.



## Performance and Flexibility of Apache Drill

Drill is fault-tolerant, and is the only SQL-on-Hadoop engine with no central servers. Drill is built from the ground up for short and low-latency queries on large datasets. Drill doesn't use MapReduce; instead, it comes with a distributed SQL MPP engine to execute queries in parallel on a cluster. The optimizer in Drill is sophisticated and leverages various rule-based and cost-based techniques, as well as the optimization capabilities of the data sources, along with data locality to determine the most efficient query plan and then distributes the execution across multiple nodes in the cluster. Drill also provides a columnar and vectorized execution engine to offer high memory and CPU efficiencies along with rapid performance for a wide variety of analytic queries.

### Columnar Storage

Drill is optimized for columnar storage. Take a typical csv file represented by this table:

Logical table representation

a	b	c
a1	b1	c1
a2	b2	c2
a3	b3	c3
a4	b4	c4
a5	b5	c5

When stored on disk in a row format, the values of each row are stored adjacent. A standard relational database loads an entire row of data before getting to the next data point in a column. In order to find a cell in a particular row, all cells of all prior rows are scanned. This incurs a lot of unnecessary disk IO if you're ignoring most of the column data.

Logical table representation

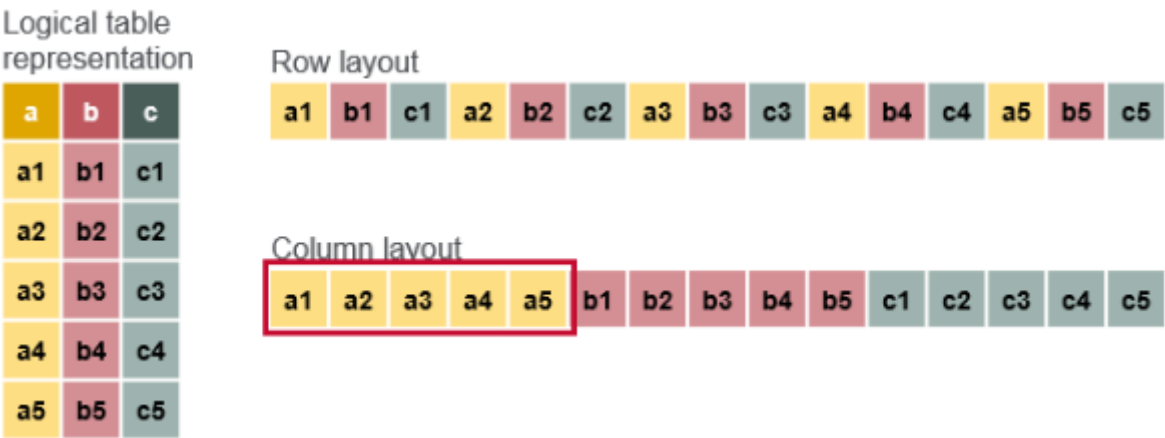
a	b	c
a1	b1	c1
a2	b2	c2
a3	b3	c3
a4	b4	c4
a5	b5	c5

Row layout

a1	b1	c1	a2	b2	c2	a3	b3	c3	a4	b4	c4	a5	b5	c5
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

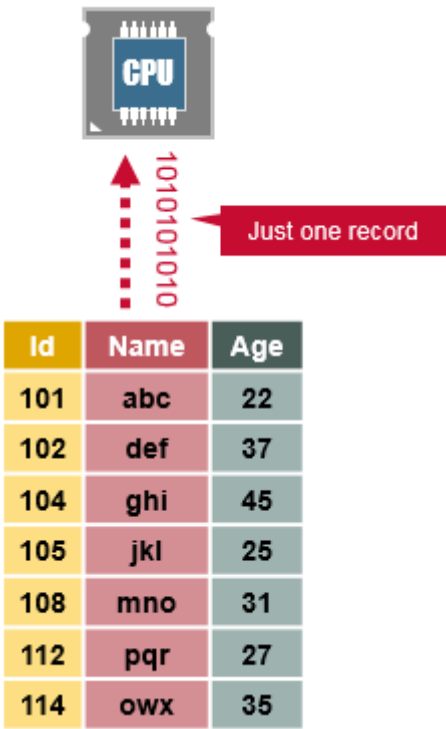


With data saved in columnar format, such as a Parquet file, the values in a column are stored next to one another instead of a row. When working with data stored in columnar formats, Drill avoids disk access for columns that are not involved in an analytic query. Only the single point of each column is needed, and the entire row does not need to be loaded. Since elements tend to repeat, this allows compression of common elements, giving a much better overall compression. For non-columnar data storage, Drill optimizes for both columnar storage and execution by using an in-memory data model that is hierarchical and columnar.

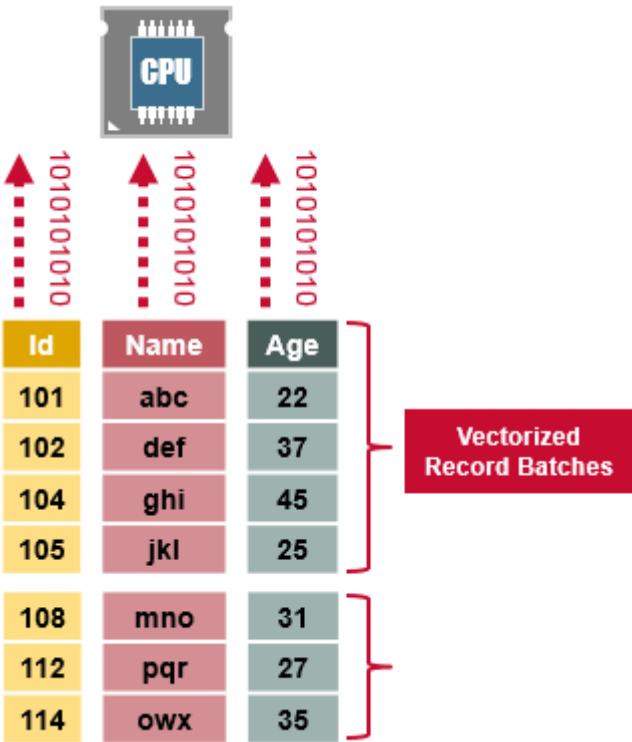


An RDBM processes one record at a time. Keeping all CPU pipelines full to achieve efficiency near peak performance is impossible to achieve in traditional database engines, primarily due to code complexity.

Vectorization

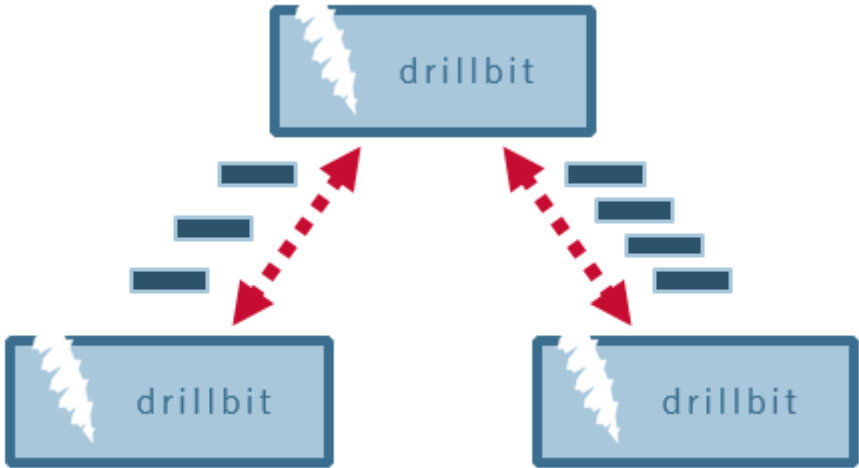


Vectorization in Drill allows the CPU to operate on vectors, or Record Batches, which are arrays of values from many different records. Drill processes sets of columns values from multiple records. Drill is able to take advantage of modern chip technology which has deep-pipelined CPU designs, keeping all pipelines full to achieve efficiency.

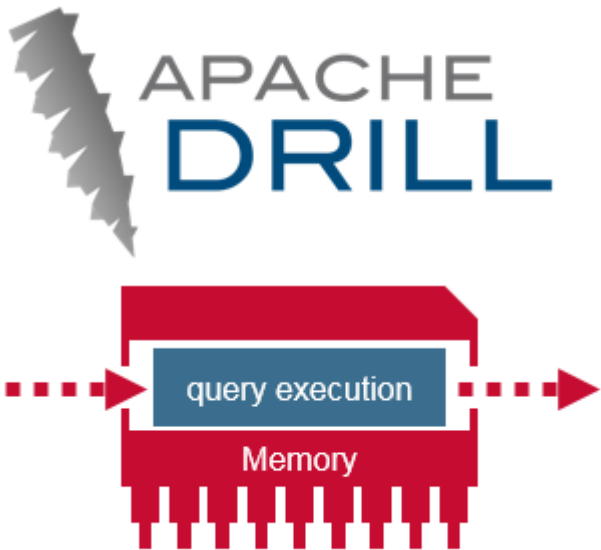


Optimistic/Pipelined Execution

Drill execution uses a pipeline model where all tasks are scheduled at once. The query execution happens in-memory as much as possible to move data through task pipelines, persisting to disk only if there is memory overflow. Record batches are pipelined between nodes, usually at 256KB per batch.

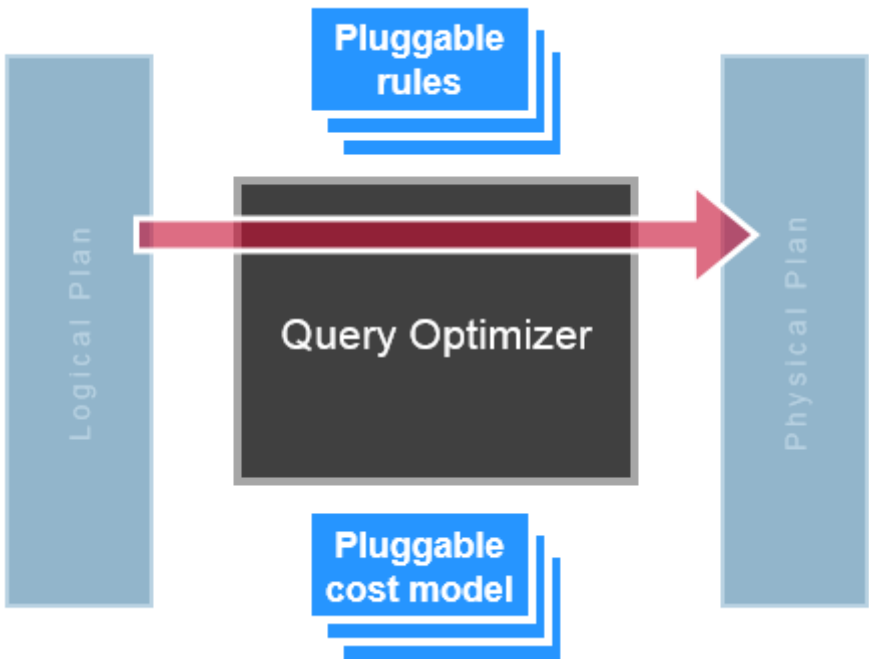


Drill adopts an optimistic execution model to process queries. Drill does not go to disk between phases of the query. All query execution is performed in memory. This is much faster, but loses failover protection. Drill assumes that failures are infrequent within the short span of a query and therefore does not spend time creating boundaries or checkpoints to minimize recovery time. Failures at node level are handled gracefully. In the instance of a single query failure, the query is rerun.



Cost-based Optimization

Drill uses various database optimizations such as rule based/cost based, as well as data locality and other optimization rules exposed by the storage engine to re-write and split the query. The output of the optimizer is a distributed physical query plan that represents the most efficient and fastest way to execute the query across the different nodes in the cluster. Optimization in Drill is pluggable, and you can provide rules for optimization at various parts of the query execution. Storage plugins can expose optimization rules to Drill, leveraging underlying storage engine features to achieve efficient execution.



The schema for data can change over the course of a Drill query, especially when querying live, streaming data. All of the Drill operators are designed to reconfigure themselves when such schema changing events occur.



cust_id	name	state	gender	age	agg_rev	membership
16841	Lisa Wells	tx	FEMALE	26-35	40	basic
19905	Neely Nova	va	MALE	15-20	45	basic
17718	William Bush	il	FEMALE	51-100	29	basic
21975	Gerald Staples	ma	MALE	26-35	28	basic

cust_id	name	state	gender	age	agg_rev	membership	reviews
16841	Lisa Wells	tx	FEMALE	26-35	40	basic	NULL
19905	Neely Nova	va	MALE	15-20	45	basic	NULL
17718	William Bush	il	FEMALE	51-100	29	basic	NULL
21975	Gerald Staples	ma	MALE	26-35	28	basic	TiqA9ybgewk

## Conclusion

In this tutorial, you learned about the components of a Drillbit, the steps involved with executing a query, how Drill executes an SQL query, the stages involved with query planning, and the techniques that Apache Drill leverages to optimize the execution of a query.

If you have any additional questions about the Apache Drill architecture, please ask them in the comments section below.

### Want to learn more?

- [Apache Drill \(https://drill.apache.org/\)](https://drill.apache.org/) – Apache.org
- [Apache Drill \(/products/apache-drill\)](/products/apache-drill) – MapR
- [Delivering Fastest Time-to-Value for SQL-on-Hadoop \(/whitepapers/delivering-fastest-time-value-sql-hadoop/\)](/whitepapers/delivering-fastest-time-value-sql-hadoop/) whitepaper
- [Learn Drill for Free: Apache Drill Essentials \(/services/mapr-academy/apache-drill-training-course-on-demand\)](/services/mapr-academy/apache-drill-training-course-on-demand)
- [Drill Explorer \(/resources/videos/drill-explorer-demo\)](/resources/videos/drill-explorer-demo) video
- [Apache Drill with Tableau \(/resources/videos/apache-drill-tableau-demo\)](/resources/videos/apache-drill-tableau-demo) video

**This blog post was published August 31, 2015.**