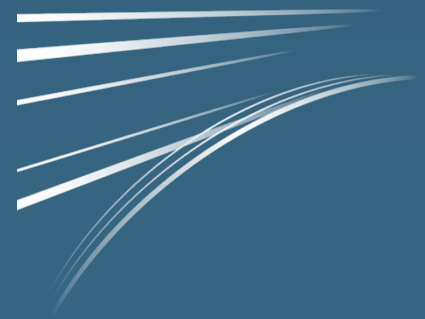
Drilling Into Drill: Flexiperf

Jacques Nadeau, Architect and VP Apache Drill

February 19, 2015



"Drill isn't just about SQL-on-Hadoop. It's about SQL-on-pretty-much-anything, immediately, and without formality."

-Andrew Brust, GigaOM Research, Dec 2014

Agenda

- What?
  - SQL like mom made
  - Punk SQL
- How?
  - Flexibility
  - Performance
- Who & When

SQL Like Mom Made

SoH Table Stakes: Warehousing and Business Intelligence

ANSI Syntax

- SELECT, FROM, WHERE, JOIN, HAVING, ORDER BY, WITH, CTAS, OVER*, ROLLUP*, CUBE*, ALL, EXISTS, ANY, IN, SOME
- VarChar, Int, BigInt, Decimal, VarBinary, Timestamp, Float, Double, etc.
- Subqueries, scalar subqueries*, partition pruning, CTE

Interactive SQL Workloads

- Data warehouse offload
- Tableau, ODBC, JDBC
- TPC-H & TPC-DS-like workloads

Standard Hadoop Tools

- Supports Hive SerDes
- Supports Hive UDFs
- Supports Hive Metastore

*alpha or imminent

Punk SQL

Punk SQL: SQL for a Hadoop World

Modern Syntax

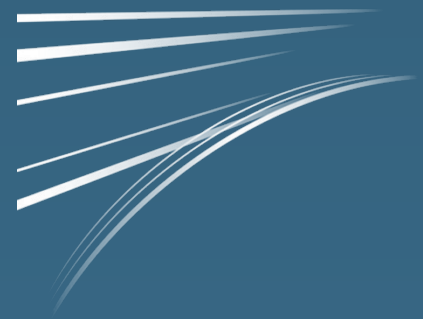
- Path based queries and wildcards
  - select * from /my/logs/
  - select * from /revenue/*/q2
- Modern data types
  - Any, Map, Array (JSON)
- Complex Functions and Relational Operators
  - FLATTEN, kvgen, convert_from, convert_to, repeated_count, etc

New Workloads

- JSON Sensor analytics
- Complex data analysis
- Alternative DSLs

New Ways to Work

- Query without prep
- Workspaces without admin intervention
- Expose query as MapReduce
- Expose query as Spark RDD



How?


Flexibility
your tool should
be flexible…
so you don’t have to be
London 2012
adidas

Flexibility is a Vision, Usability, a Religion

- Deployment
- Data Model
- Schema
- Security
- Access Methods

Supporting the Changing Roles of Big Data

Data Dev Circa 2000

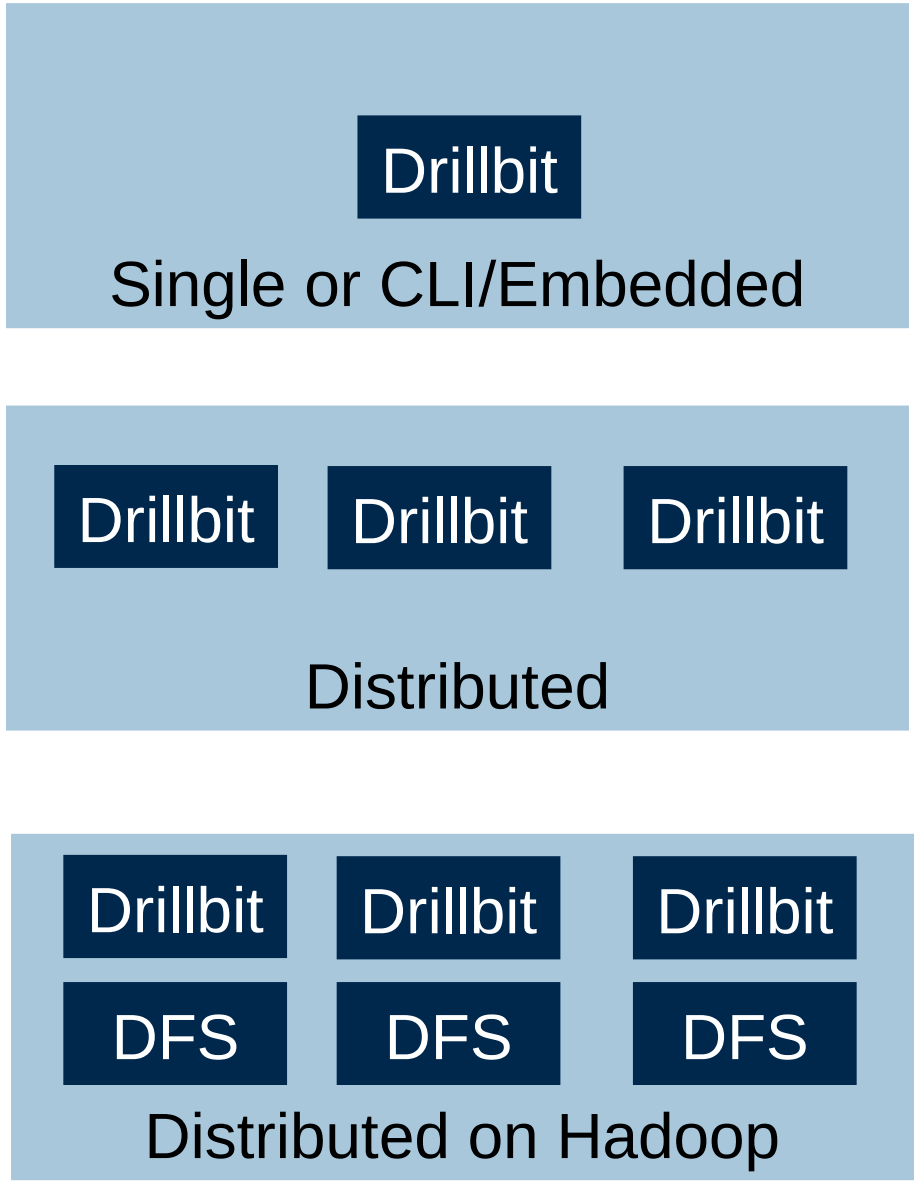
1. Developer comes up with requirements
2. DBA defines tables
3. DBA defines indices
4. DBA defines FK relationships
5. Developer stores data
6. BI builds reports
7. Analyst views reports
8. DBA adds materialized views

Data Today

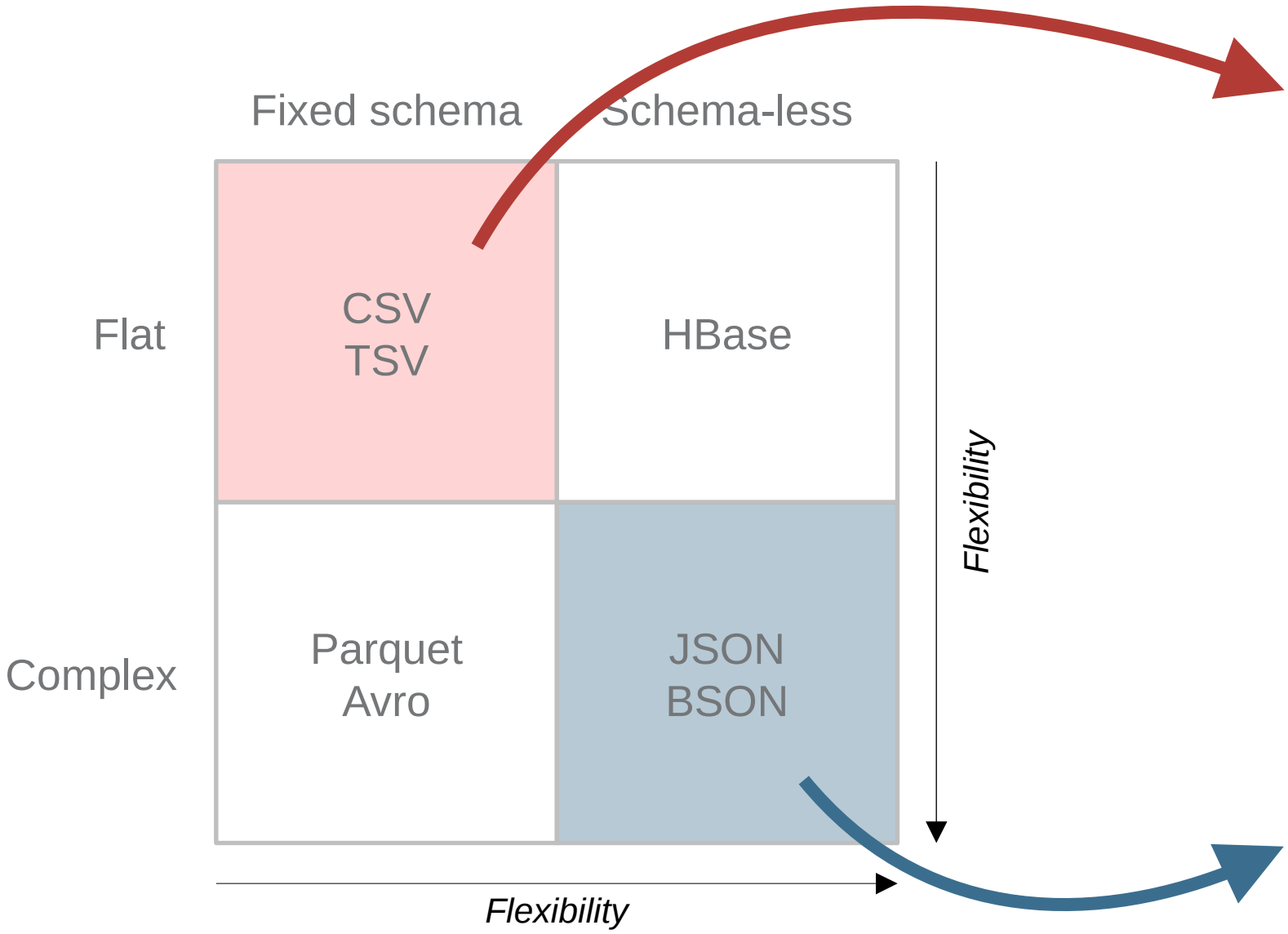
1. Developer builds app, defines schema, stores data
2. Analyst queries data
3. Data engineer fixes performance problems or fills functionality gaps

Drill Deployment is Easy

- Single Daemon for all purposes
- No special considerations for scaling or availability
- With or without DFS
- Works with other data systems (Mongo, Cassanrda & JDBC coming soon)
- Runs on Linux, Mac or Windows
- No separate database
- JSON everything
- Access via HTTP, Java, C, JDBC, ODBC, CLI



Drill Provides A Flexible Data Model



RDBMS/SQL-on-Hadoop table

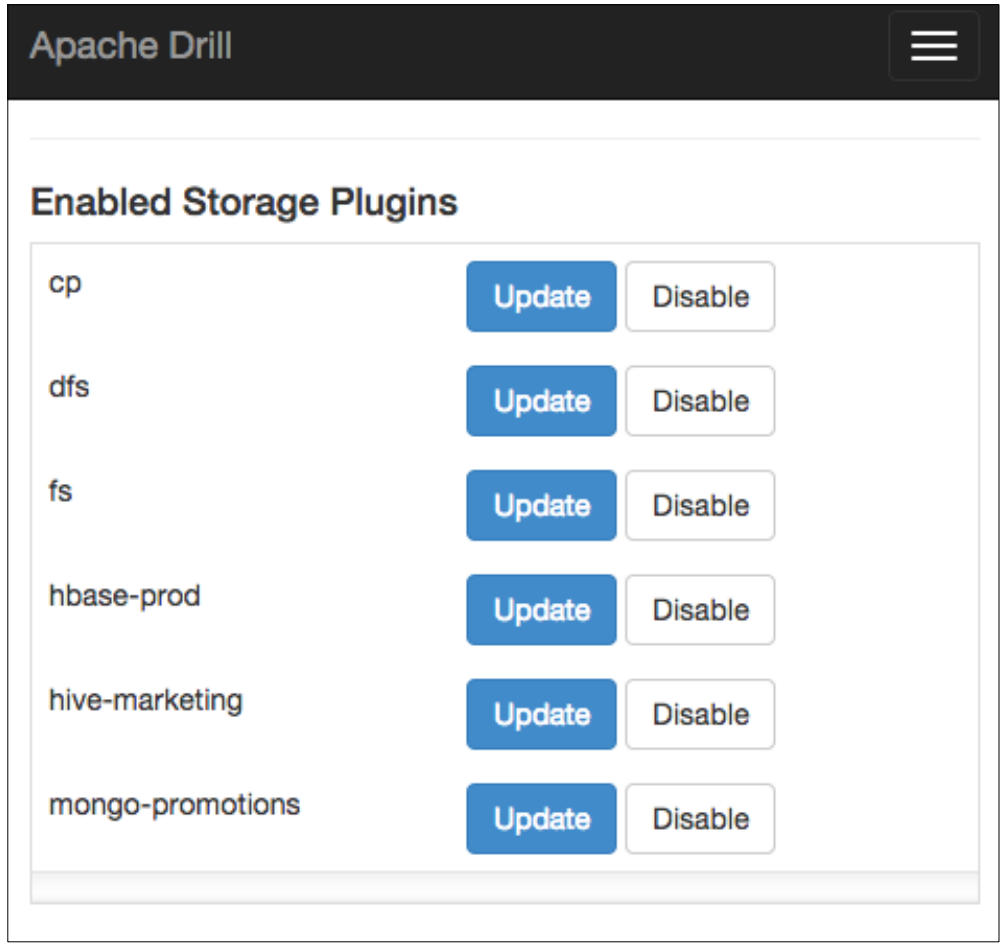
Name	Gender	Age
Michael	M	6
Jennifer	F	3

Apache Drill table

```
{
  name: {
    first: Michael,
    last: Smith
  },
  hobbies: [ski, soccer],
  district: Los Altos
}
{
  name: {
    first: Jennifer,
    last: Gates
  },
  hobbies: [sing],
  preschool: CCLC
}
```

Leave Your Data Where it is. Access it Centrally & Uniformly.

- Drill is storage agnostic
- Interacts to storage through plugins
- Storage plugins expose optimizer rules
  - Optimizer rules work directly on logical operation to expose maximum capabilities
- Reference multiple Hive, HBase, MongoDB, DFS, etc systems



Leverage that Massive Scalable Redundant Infrastructure

Single Store for Data and Metadata

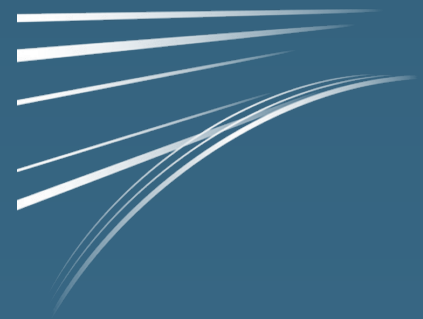
- HDFS is already your single canonical store
- Don’t create a secondary metadata store

Avoid Metadata Management and synchronization

- Store metadata inline
  - If you can’t, store it next to files
- Move directories around at will
- Delete things at will

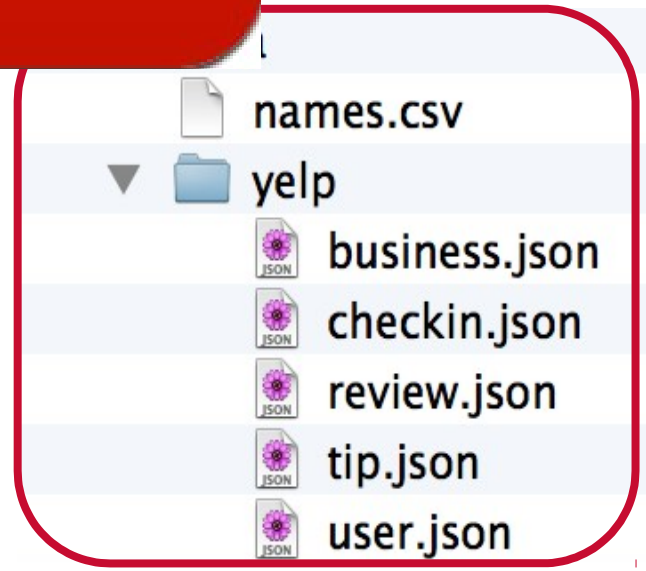
Flexibility in how you describe your data

- Drill doesn’t require schema, detects file types based on
  - extensions
  - magic bytes (e.g. PAR1)
  - systems settings
- Query can be planned on any file, anywhere
- Data types are determined as data arrives
- Some formats have known schema
  - If they don’t, you can expose them as such through views
  - Views are simply JSON files that define view SQL



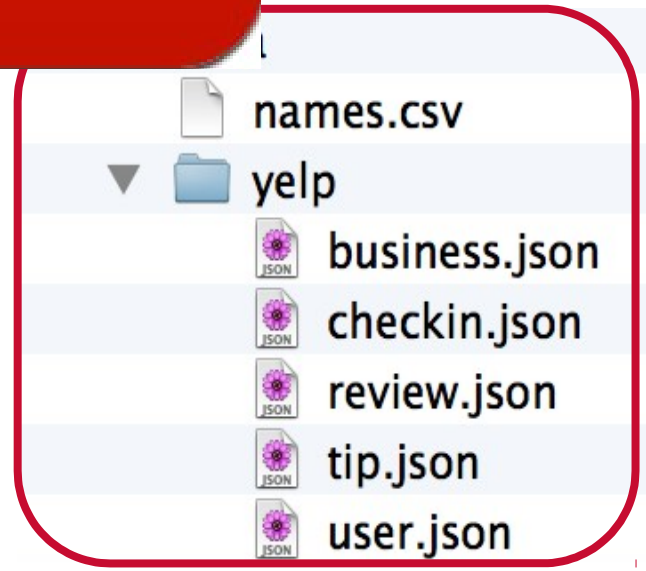
Product Walkthrough

Business dataset




```
{
	"business_id":"4bEjOyTaDG24SY5TxsaUNQ",
	"full_address":"3655 Las Vegas Blvd S\nThe Strip\nLas Vegas,NV 89109",
	"hours":{
		"Monday":{"close":"23:00","open":"07:00"},
		"Tuesday":{"close":"23:00","open":"07:00"},
		"Friday":{"close":"00:00","open":"07:00"},
		"Wednesday":{"close":"23:00","open":"07:00"},
		"Thursday":{"close":"23:00","open":"07:00"},
		"Sunday":{"close":"23:00","open":"07:00"},
		"Saturday":{"close":"00:00","open":"07:00"}
	},
	"open":true,
	"categories":["Breakfast & Brunch","Steakhouses","French","Restaurants"],
	"city":"Las Vegas",
	"review_count":4084,
	"name":"Mon Ami Gabi",
	"neighborhoods":["The Strip"],
	"longitude":-115.172588519464,
	"state":"NV",
	"stars":4.0,
	"attributes":{
		"Alcohol":"full_bar",
		"Noise Level":"average",
		"Has TV":false,
		"Attire":"casual",
		"Ambience":{
			"romantic":true,
			"intimate":false,
			"touristy":false,
			"hipster":false,
			"classy":true,
			"trendy":false,
			"casual":false
		},
		"Good For":{"dessert":false,"latenight":false,"lunch":false,
		  "dinner":true,"breakfast":false,"brunch":false},
	}
}
```

Reviews dataset




```
{
 "votes":{"funny":0,"useful":2,"cool":1},
 "user_id":"Xqd0DzHaiyRqVH3WRG7hzg",
 "review_id":"15SdjuK7DmYqUAj6rjGowg",
 "stars":5,
 "date":"2007-05-17",
 "text":"dr.goldberg offers everything ...",
 "type":"review",
 "business_id":"vcNAWiLM4dR7D2nwwJ7nCA"
}
```

Zero to Results in 2 minutes

```
$ tar -xvzf apache-drill-0.7.0.tar.gz

$ bin/sqlline -u jdbc:drill:zk=local

> SELECT state, city, count(*) AS businesses
  FROM dfs.yelp.`business.json`
  GROUP BY state, city
  ORDER BY businesses DESC LIMIT 10;
```

```
+------------+------------+------------+
|   state    |    city    | businesses |
+------------+------------+------------+
| NV         | Las Vegas  | 12021      |
| AZ         | Phoenix    | 7499       |
| AZ         | Scottsdale | 3605       |
| EDH        | Edinburgh  | 2804       |
| AZ         | Mesa       | 2041       |
| AZ         | Tempe      | 2025       |
| NV         | Henderson  | 1914       |
| AZ         | Chandler   | 1637       |
| WI         | Madison    | 1630       |
| AZ         | Glendale   | 1196       |
+------------+------------+------------+
```

Install

Launch shell (embedded mode)

Query files and directories

Results

Intuitive SQL access to complex data

```
// It's Friday 10pm in Vegas and looking for Hummus

> SELECT name, stars, b.hours.Friday friday, categories
  FROM dfs.yelp.`business.json` b
  WHERE b.hours.Friday.`open` < '22:00' AND
        b.hours.Friday.`close` > '22:00' AND
        REPEATED_CONTAINS(categories, 'Mediterranean') AND
        city = 'Las Vegas'
  ORDER BY stars DESC
  LIMIT 2;

+------------+------------+------------+------------+
|    name    |   stars    |   friday   | categories |
+------------+------------+------------+------------+
| Olives     | 4.0        | {"close":"22:30","open":"11:00"} | ["Mediterranean","Restaurants"] |
| Marrakech Moroccan Restaurant | 4.0 | {"close":"23:00","open":"17:30"} |
["Mediterranean","Middle Eastern","Moroccan","Restaurants"] |
+------------+------------+------------+------------+
```

Query data with any levels of nesting

ANSI SQL compatibility

```
//Get top cool rated businesses

SELECT b.name from dfs.yelp.`business.json` b
WHERE b.business_id IN
(SELECT r.business_id FROM dfs.yelp.`review.json` r
GROUP BY r.business_id HAVING SUM(r.votes.cool) > 2000 ORDER BY
SUM(r.votes.cool) DESC);
```

```
+-----------+
|  name     |
+-----------+
|EarlofSandwich|
|XS Nightclub|
|The Cosmopolitan of Las Vegas|
|Wicked Spoon|
+-----------+
```

Use familiar SQL functionality (Joins, Aggregations, Sorting, Sub-queries, SQL data types)

Logical views

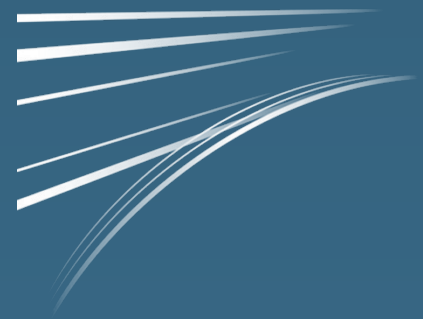
```
//Create a view combining business and reviews datasets

> CREATE OR REPLACE VIEW dfs.tmp.BusinessReviews AS
   SELECT b.name, b.stars, r.votes.funny,
       r.votes.useful, r.votes.cool, r.`date`
    FROM dfs.yelp.`business.json` b, dfs.yelp.`review.json` r
    WHERE r.business_id = b.business_id;

+-----------+-----------+
|   ok      |  summary  |
+-----------+-----------+
|true       |View 'BusinessReviews' created successfully in 'dfs.tmp' schema |
+-----------+-----------+

> SELECT COUNT(*) AS Total FROM dfs.tmp.BusinessReviews;

+------------+
|    Total   |
+------------+
|  1125458   |
+------------+
```

Lightweight file system based views for granular and de-centralized data management

Materialized Views AKA Tables

```
> ALTER SESSION SET `store.format` = 'parquet';

> CREATE TABLE dfs.yelp.BusinessReviewsTbl AS
   SELECT b.name, b.stars, r.votes.funny funny,
       r.votes.useful useful, r.votes.cool cool, r.`date`
    FROM dfs.yelp.`business.json` b, dfs.yelp.`review.json` r
    WHERE r.business_id = b.business_id;

+-----------+-------------------------+
| Fragment  | Number of records written |
+-----------+-------------------------+
| 1_0       | 176448                  |
| 1_1       | 192439                  |
| 1_2       | 198625                  |
| 1_3       | 200863                  |
| 1_4       | 181420                  |
| 1_5       | 175663                  |
+-----------+-------------------------+
```

Save analysis results as tables using familiar CTAS syntax



Working with repeated values

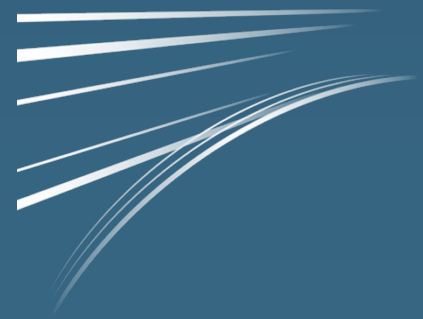
Extensions to ANSI SQL to work with repeated values

```
//Flatten repeated categories

> SELECT name,categories
  FROM dfs.yelp.`business.json` LIMIT 3;

+------------+------------+
|    name    | categories |
+------------+------------+
| Eric Goldberg, MD | ["Doctors","Health & Medical"] |
| Pine Cone Restaurant | ["Restaurants"] |
| Deforest Family Restaurant | ["American (Traditional)","Restaurants"] |
+------------+------------+

> SELECT name,FLATTEN(categories) AS categories
  FROM dfs.yelp.`business.json` LIMIT 5;
+------------+------------+
|    name    | categories |
+------------+------------+
| Eric Goldberg, MD | Doctors |
| Eric Goldberg, MD | Health & Medical |
| Pine Cone Restaurant | Restaurants |
| Deforest Family Restaurant | American (Traditional) |
| Deforest Family Restaurant | Restaurants |
+------------+------------+
```

Dynamically flatten repeated and nested data elements as part of SQL queries. No ETL necessary

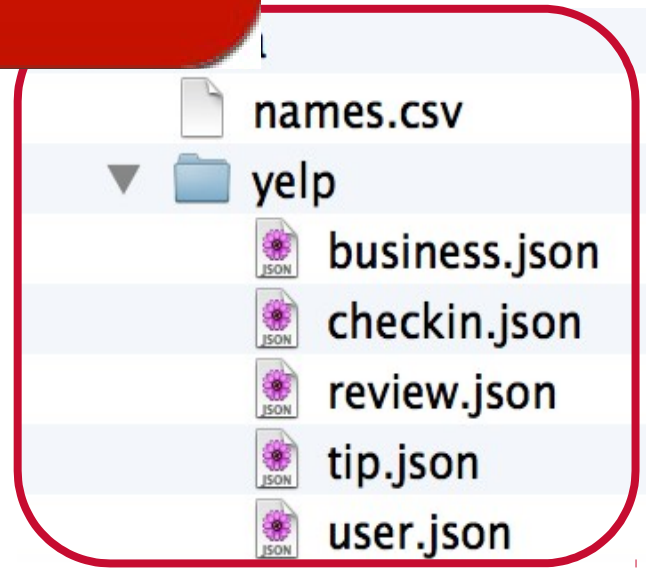
Extensions to ANSI SQL to work with repeated values

```
//Get most common business categories

> SELECT category, count(*) AS categorycount
  FROM (SELECT name, FLATTEN(categories) AS category
        FROM dfs.yelp.`business.json`) c
  GROUP BY category ORDER BY categorycount DESC;

+-----------+-----------+
| category  |categorycount|
+-----------+-----------+
|Restaurants|14303      |
…
|Australian |1          |
|BoatDealers|1          |
|Firewood   |1          |
+-----------+-----------+
```



Working with Dynamic Columns

Check ins dataset




```
{
  "checkin_info":{
    "3-4":1,
    "13-5":1,
    "6-6":1,
    "14-5":1,
    "14-6":1,
    "14-2":1,
    "14-3":1,
    "19-0":1,
    "11-5":1,
    "13-2":1,
    "11-6":2,
    "11-3":1,
    "12-6":1,
    "6-5":1,
    "5-5":1,
    "9-2":1,
    "9-5":1,
    "9-6":1,
    "5-2":1,
    "7-6":1,
    "7-5":1,
    "7-4":1,
    "17-5":1,
    "8-5":1,
    "10-2":1,
    "10-5":1,
    "10-6":1
  },
  "type":"checkin",
  "business_id":"JwUE5GmEO-sH1FuwJgKBlQ"
}
```

Makes it easy to work with dynamic/unknown columns

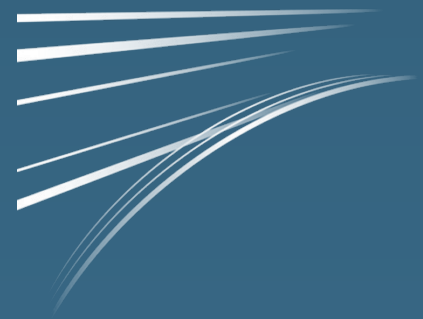
```
> jdbc:drill:zk=local> SELECT KVGEN(checkin_info) checkins
  FROM dfs.yelp.`checkin.json` LIMIT 1;
+------------+
| checkins   |
+------------+
|[{"key":"3-4","value":1},{"key":"13-5","value":1},{"key":"6-6","value":1},{"key":"14-5","value":1},{"key":"14-
6","value":1},{"key":"14-2","value":1},{"key":"14-3","value":1},{"key":"19-0","value":1},{"key":"11-5","value":1},
{"key":"13-2","value":1},{"key":"11-6","value":2},{"key":"11-3","value":1},{"key":"12-6","value":1},{"key":"6-
5","value":1},{"key":"5-5","value":1},{"key":"9-2","value":1},{"key":"9-5","value":1},{"key":"9-6","value":1},
{"key":"5-2","value":1},{"key":"7-6","value":1},{"key":"7-5","value":1},{"key":"7-4","value":1},{"key":"17-
5","value":1},{"key":"8-5","value":1},{"key":"10-2","value":1},{"key":"10-5","value":1},{"key":"10-6","value":1}]|
+------------+
```

ConvertMap with a wide set of dynamic columns into an array of key-value pairs

```
> jdbc:drill:zk=local> SELECT FLATTEN(KVGEN(checkin_info)) checkins
FROM dfs.yelp.`checkin.json` limit 6;

+------------+
| checkins   |
+------------+
|{"key":"3-4","value":1} |
|{"key":"13-5","value":1} |
|{"key":"6-6","value":1} |
|{"key":"14-5","value":1} |
|{"key":"14-6","value":1} |
|{"key":"14-2","value":1} |
+------------+
```

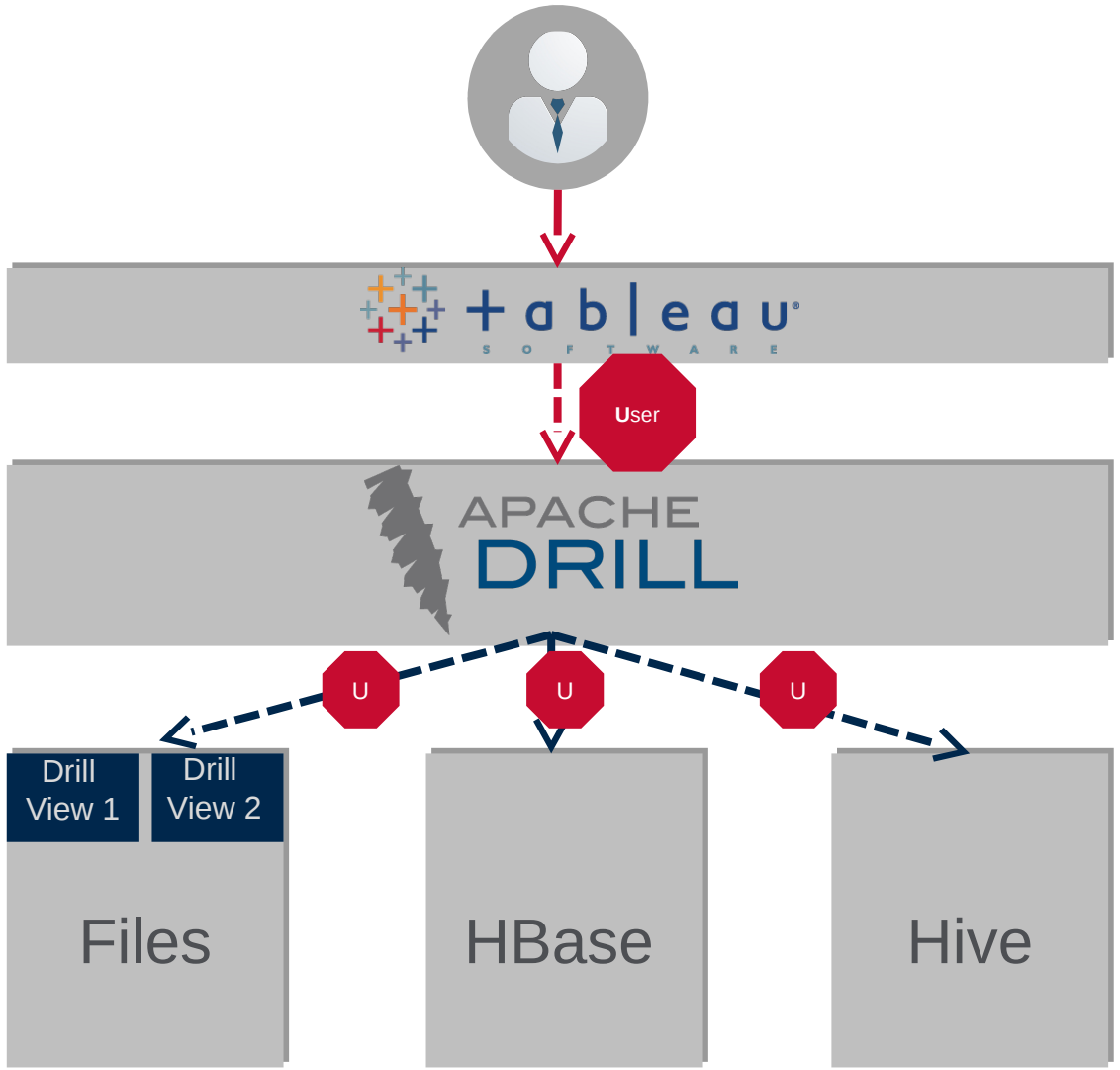
Makes it easy to work with dynamic/unknown columns

```
// Count total number of checkins on Sunday midnight

➢  jdbc:drill:zk=local> SELECT SUM(checkintbl.checkins.`value`) as
   SundayMidnightCheckins FROM
  (SELECT FLATTEN(KVGEN(checkin_info)) checkins
   FROM dfs.yelp.`checkin.json`) checkintbl
   WHERE checkintbl.checkins.key='23-0';

+----------------------+
|SundayMidnightCheckins|
+----------------------+
|8575                  |
+----------------------+
```



Secure Access

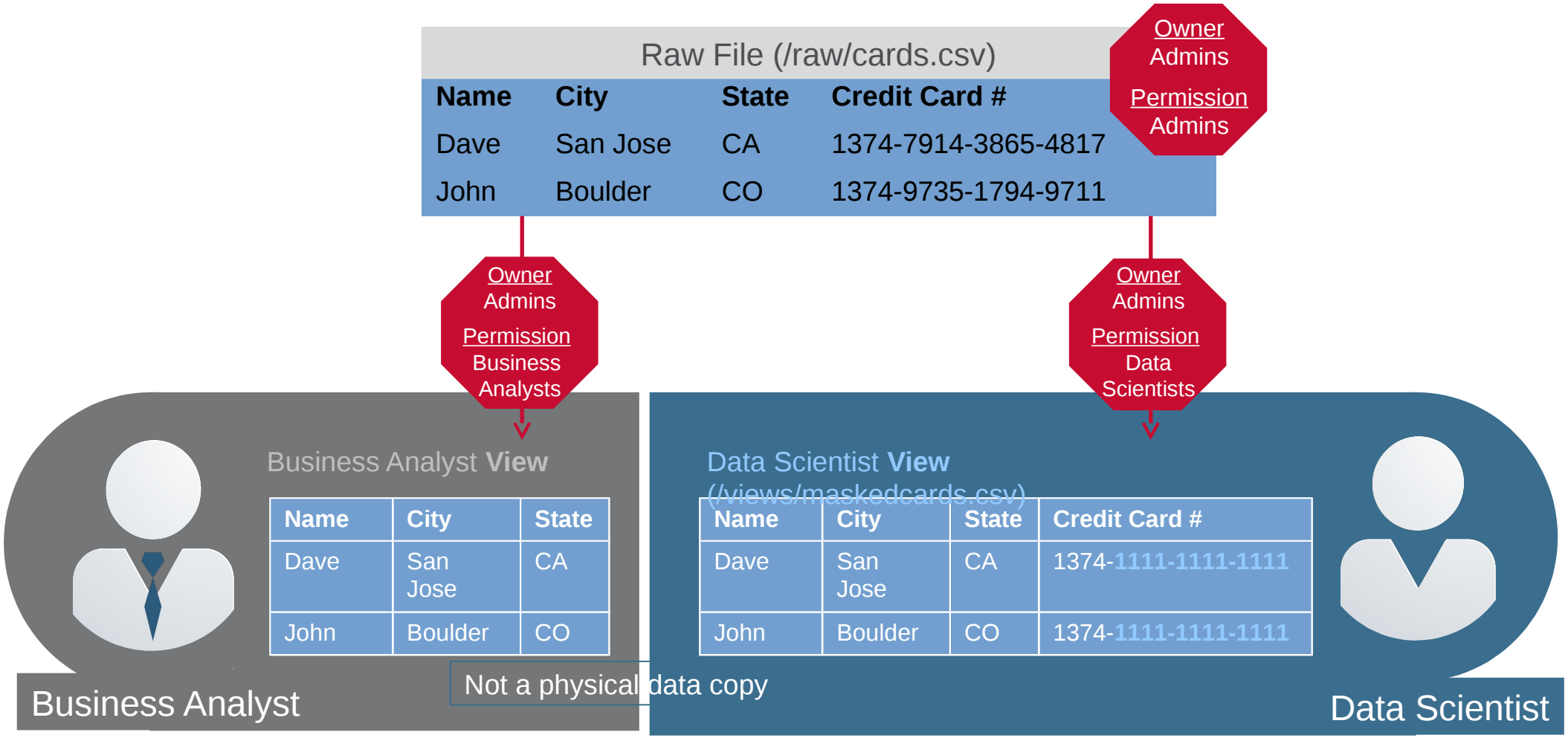
Access control that scales



User PAM Authentication + User Impersonation

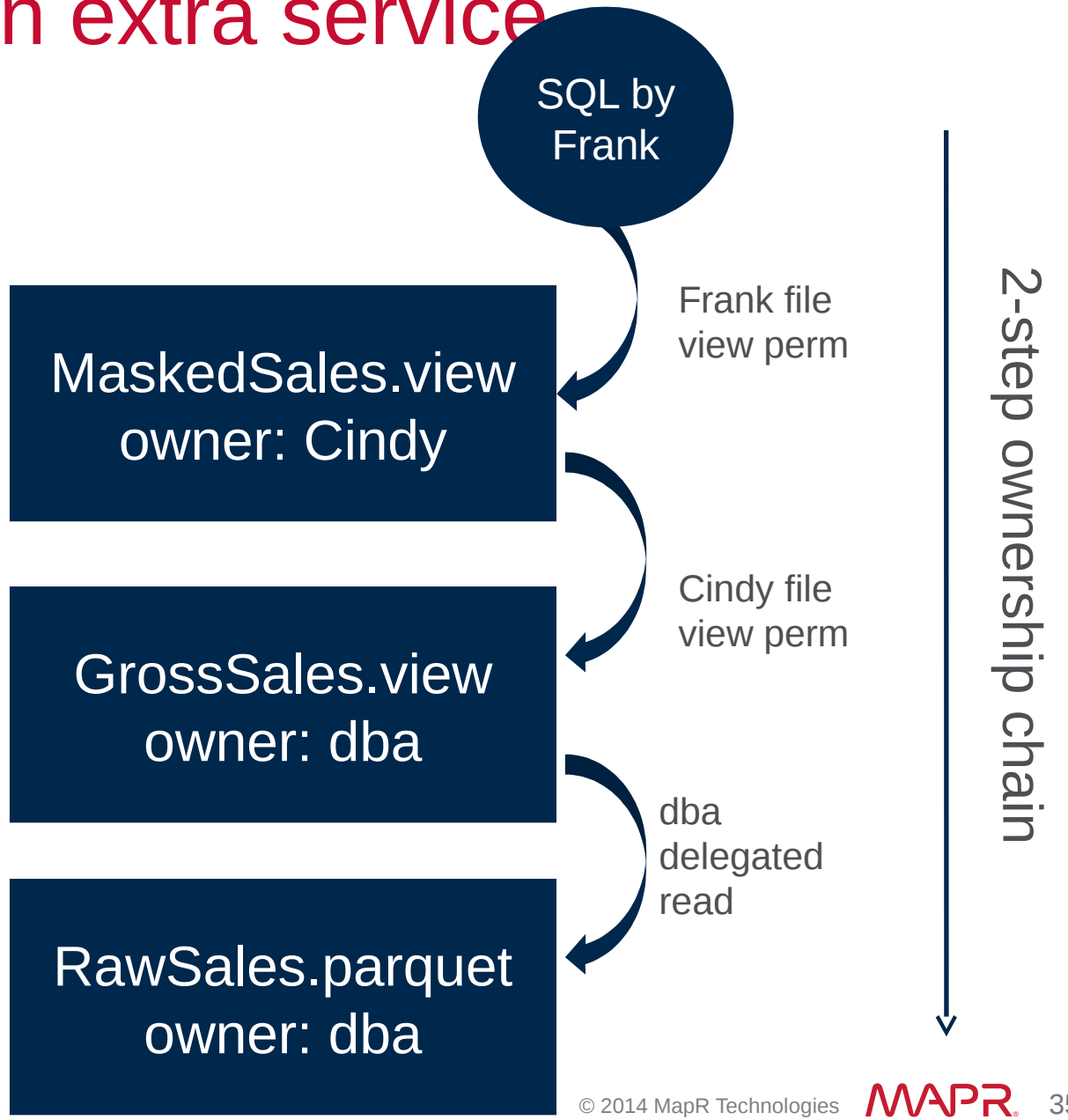
Fine-grained row and column level access control with Drill Views – no centralized security repository required

Granular security permissions through Drill views



Secure your data without an extra service

- Drill Views
- Ownership chaining with configurable delegation TTL
- Leverages existing HDFS ACLs
- Complete security solution without additional services or software



Summary

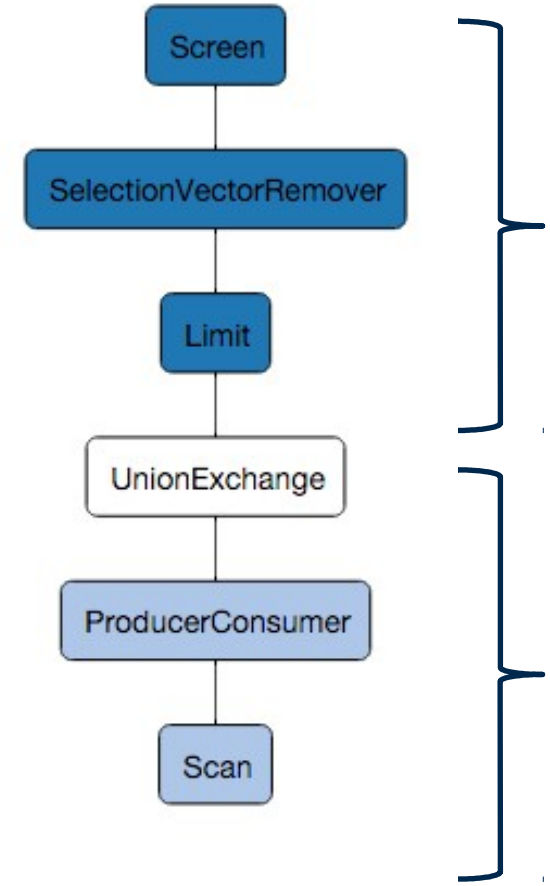
- Logical
  - No physical data copies/silos
- Granular
  - Row level and column level security controls
- De-centralized
  - User impersonation respecting storage system permissions
  - No separate permission repository for granular controls
  - Integrated with Hadoop File System permissions and LDAP
- Self-service w/ governance
  - If you have access to data, you control who and how widely can access it
  - Audits

Core Drill Architectural Goals

- Go fast when you don’t know anything
  - And do “the right thing”
- Go faster when you do know things

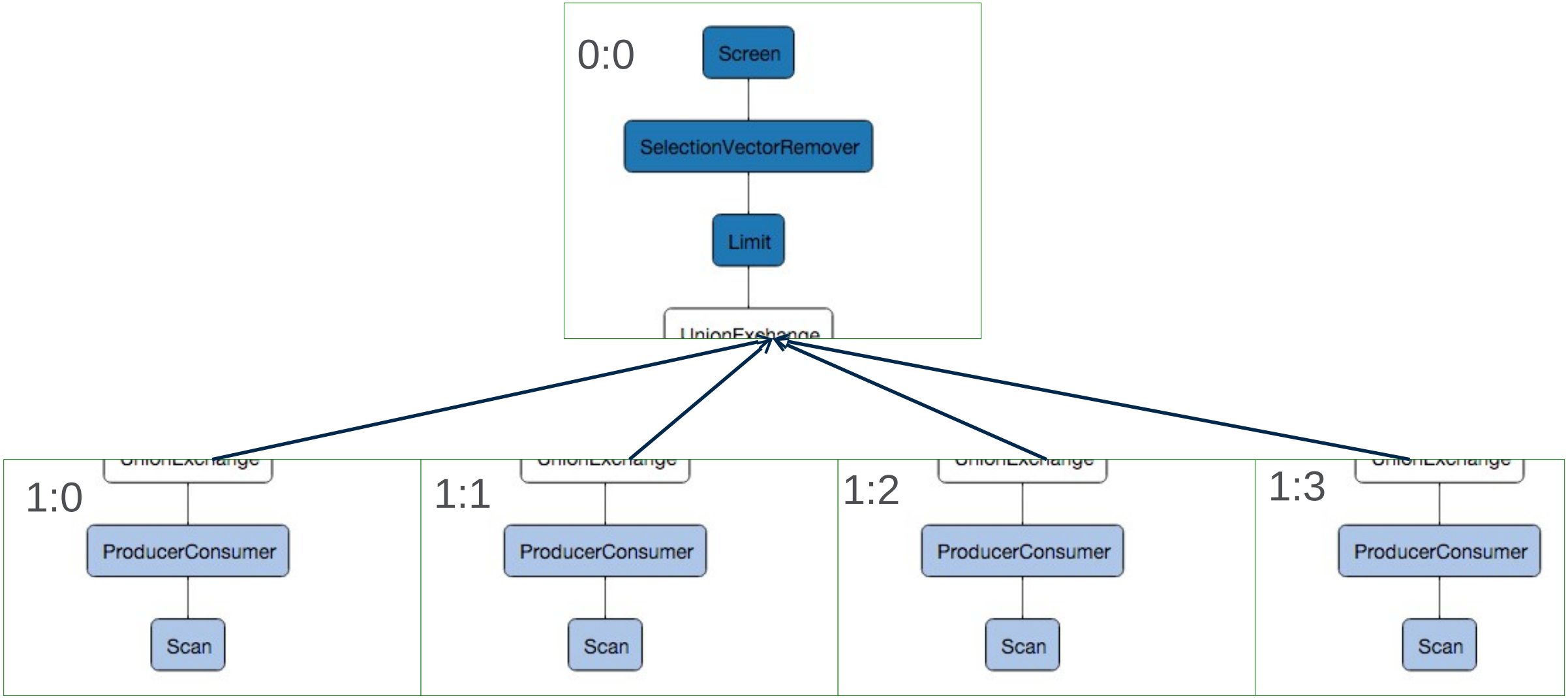
An Optimistic, Pipelined Purpose-Built DAG Engine

- Three-Level DAG
- Major Fragments (phases)
- Minor Fragments (threads)
- Operators (in-thread operations)




```
> explain plan for select * from customer limit 5;
…
00-00    Screen
00-01      SelectionVectorRemover
00-02        Limit(fetch=[5])
00-03          UnionExchange
01-01            ProducerConsumer
01-02              Scan(groupscan=[ParquetGroupScan […
```

Each phase (MajorFragment) gets Parallelized (MinorFragment)



Reading Data Quickly, Moving Data Quickly

- Highly Optimized Native Drill Readers:
  - Vectorized Parquet, Text/CSV, JSON
  - Also works with all Hive supported formats
- Drill supports partition pruning
  - Adding direct physical property exposure soon for highly optimized cases
- Drill parallelizes to maximum level format allows
  - Also balances data locality and maximum parallelization
- Bespoke Asynchronous Zero-Copy RPC Layer
  - Built specifically for Drill's internal data format

Parquet: The Format for the next decade

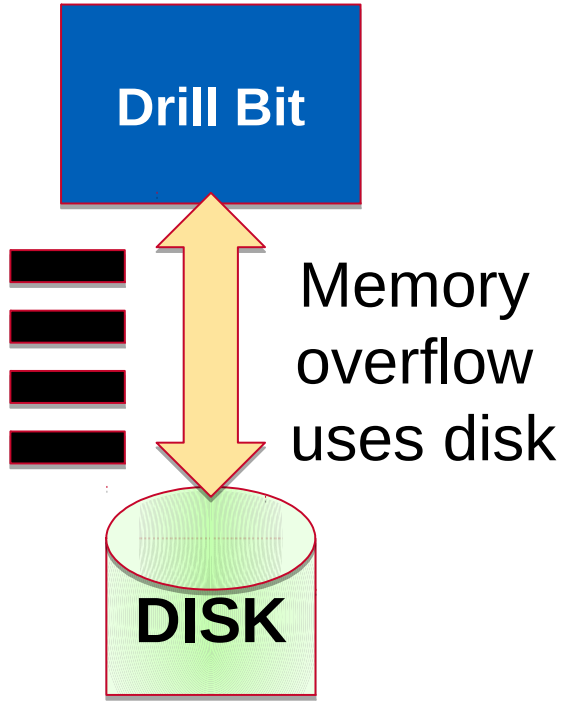
Apache Drill & Apache Parquet communities working together

- Better Ecosystem integration and decentralized metadata
  - Self Description capabilities
  - Ecosystem support for logical data types
- Enhanced Performance
  - New vectorized reading
  - Enhanced memory pooling and management
  - Indexing*
  - Better metadata positioning (for improved page pruning)*
  - Enhanced vectorization and late materialization reading*

*In progress

Value Vectors & Record Batches: Drill's In-memory Columnar Work Units

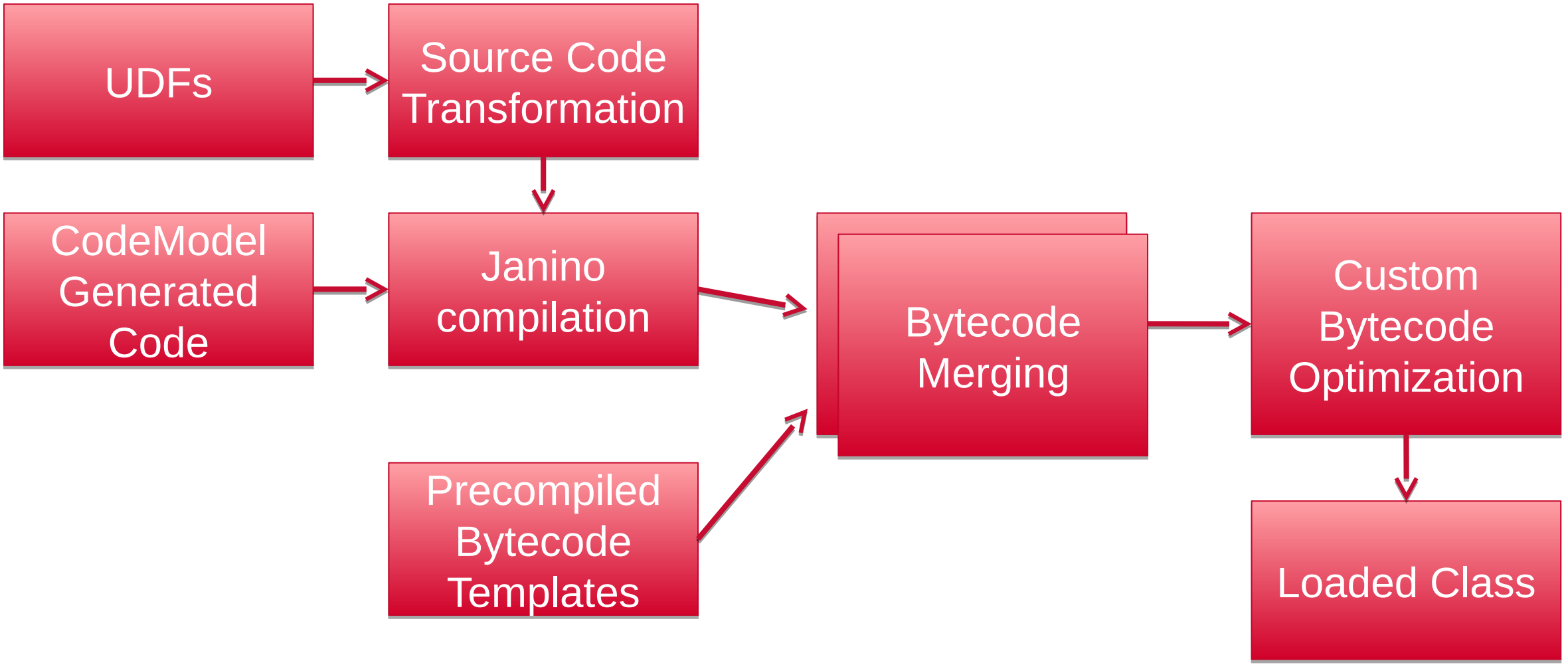
- Random access: sort without copy or restructuring
- Fully specified in memory shredded complex data structures
- Remove serialization or copy overhead at node boundaries
- Spool to disk as necessary
- Interact with data in Java or C without copy or overhead



Runtime Compilation and Multi-phased planning

- Drill does best effort initial query parsing, planning and validation
  - Where Drill doesn't understand data, it provides support for ANY type, allowing late type binding.
- At execution time, individual nodes do secondary pass
  - Schema <--> Query parsing and validation
  - Type casting, coercion and promotion
  - Compilation based on schema requirements
- As schema changes, Drill supports recompilation of each operator as necessary

Runtime Compilation Pattern



Advanced Compilation Techniques

- Optimization based on observation and assembly
- Drill does a number of pre-machine-code-compilation optimizations to ensure efficient execution
- Some examples:
  - Removal of type and bounds checking
  - Direct micro pointers for in-record-batch references
  - Little endian data formats
  - Bytecode-level scalar replacement

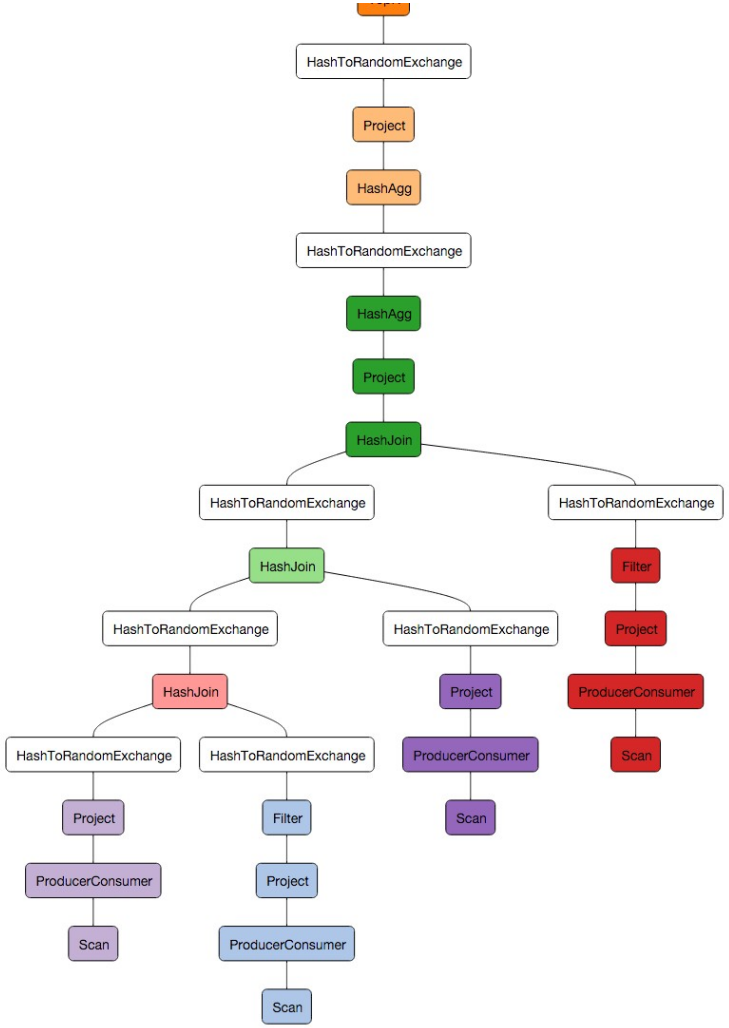
Drill Does Vectorization & Supports Columnar Functions

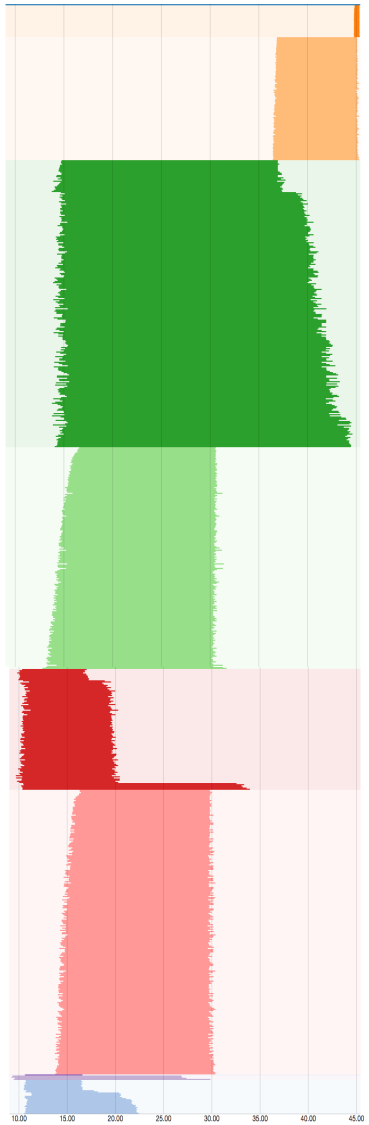
- Drill often operates on more than one record at a time
  - Word-sized manipulations
  - SIMD instructions
  - Manually coded algorithms
- Columnar Functions Improve Many Operations
  - Bitmaps allow lightning fast null-checks and reduction in branching
  - Type demotion to reduce memory and CPU computation overhead
  - Direct Conversions where possible

Drill provides advanced query telemetry

- What happened during query for all three levels of DAG execution
- Each profile is stored as JSON file for easy review, sharing and backup (at end of query execution)
- Profiles can be analyzed using Drill, allows:
  - easy longitudinal analysis of workload
  - multi-tenancy performance analysis
  - impact of configuration changes to benchmark workloads

A Color-coded visual layout and Gant timing chart is provided





Memory Efficiency

- Drill's in memory representation is designed to minimize memory overhead
- Custom implementation of columnar-aware data structures including hash tables, sort operations, etc.
  - For example, entry overhead for hash table is 8 bytes per value
  - Sort pointer overhead for sort is 2 to 4 bytes per entry
- Adding support for compressed columnar representation further to improve compactness

Scale and Concurrency

- Drill's execution model leverages both local and remote exchanges for changes in parallelization
  - Muxing Local, Demuxing Local, Broadcast, Hash to Merge, Hash to Random, Ordered Partition, Single Merge, Union
- All operations can be parallelized at the thread and node level
- Thread count and parallelization are influenced by data size, query phasing and system load
  - Administrators have basic queuing control to manage workload
- All threads are independent and pipelined, all run in a single process per node
- Node <> node communication is multiplexed, push-based with sub-socket back-pressure support
- Testing has proceeded up to 150 nodes. Target is 1000 nodes by GA.
  - Drill adapts data transfer size, buffers, muxing and other operations based on query scale to minimize n^2 multiplier effects