Apache Drill – Why, What and How

M. C. Srivas, CTO and Founder

Sept 10, 2014

MapR is Unbiased Open Source

Linux Is Unbiased

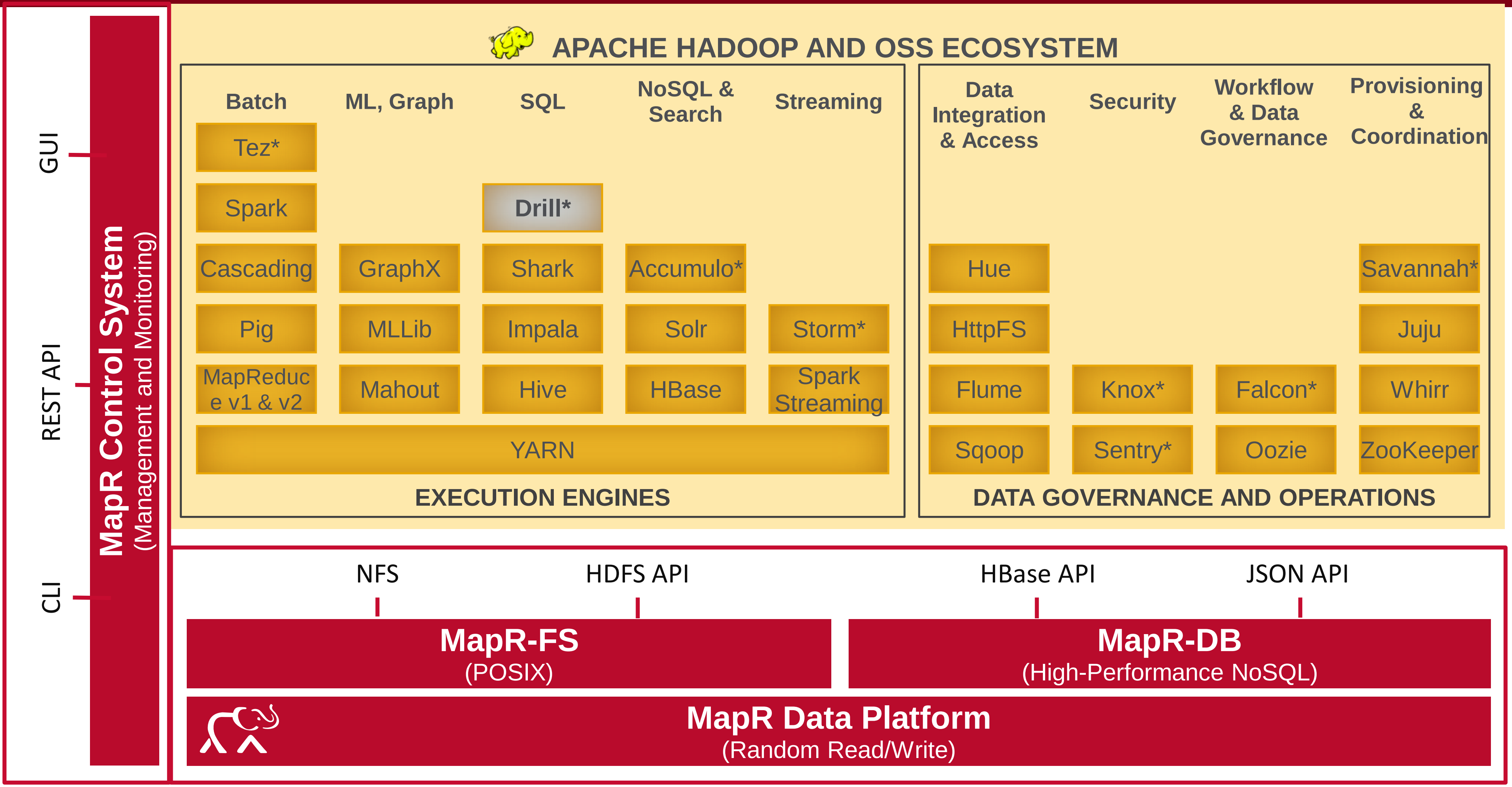
- Linux provides choice
  - MySQL
  - PostgreSQL
  - SQLite
- Linux provides choice
  - Apache httpd
  - Nginx
  - Lighttpd

MapR Is Unbiased

- MapR provides choice

	MapR Distribution for Hadoop	Distribution C	Distribution H
Spark	Spark (*all* of it) and SparkSQL	Spark only	No
Interactive SQL	Impala, Drill, Hive/Tez, SparkSQL	One option (Impala)	One option (Hive/Tez)
Scheduler	YARN, Mesos	One option (YARN)	One option (YARN)
Versions	Hive 0.10, 0.11, 0.12, 0.13 Pig 0.11, 012 HBase 0.94, 0.98	One version	One version

MapR Distribution for Apache Hadoop



Enterprise Grade

Data Hub

Operational

* In Roadmap for inclusion/certification

APACHE DRILL

Hadoop an augmentation for EDW—Why?

Make money from cross-selling: "Customer 360°"

CSO/CMO

Make money from cross-selling: "Customer 360°"

CSO/CMO

Regulation: "You need to store more stuff, and find it fast"

CRO

Make money
from cross-selling:
“Customer 360° ”

CSO/CMO

Regulation:
“You need to
store more stuff,
and find it fast”

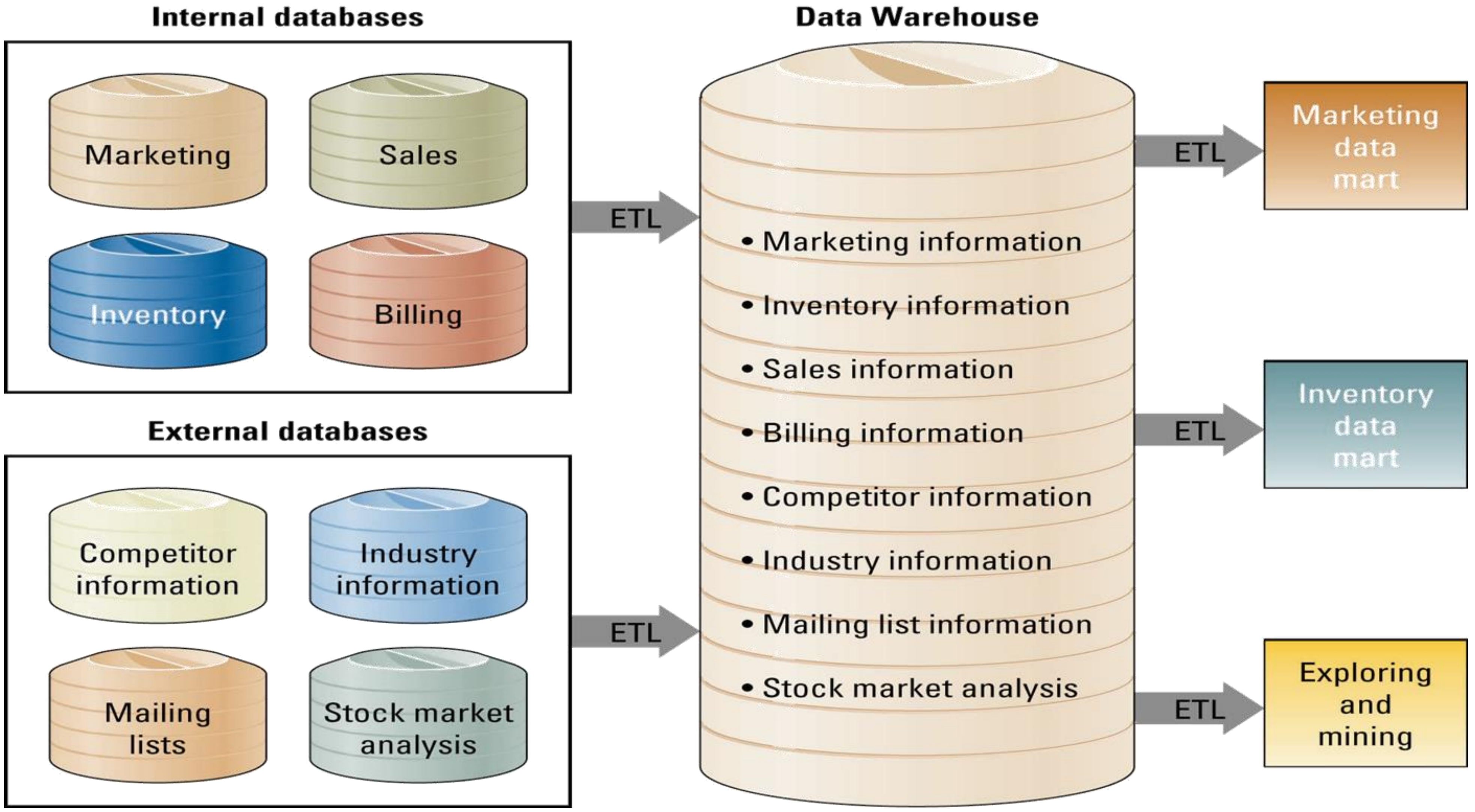
CRO

Reduce OpEx
on IT spend by identifying
anomalies faster

CIO

What’s holding us back?

Data Warehouse Model



But inside, it looks like this …

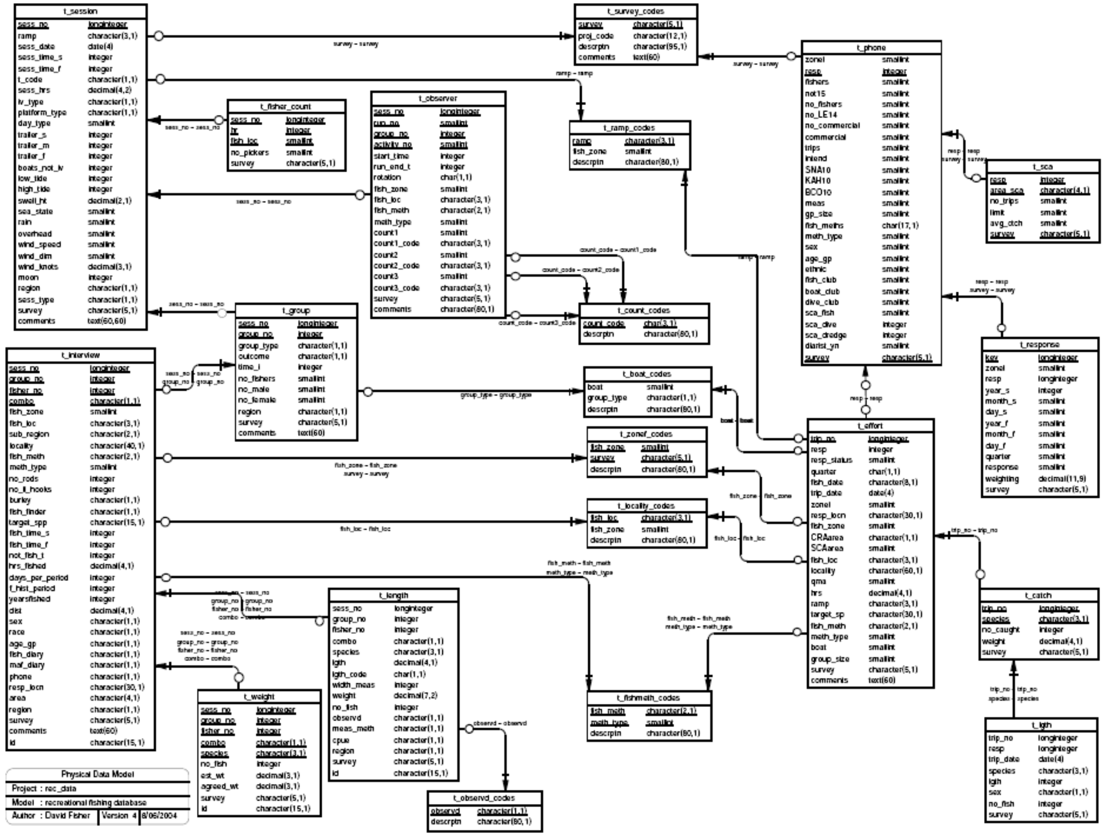
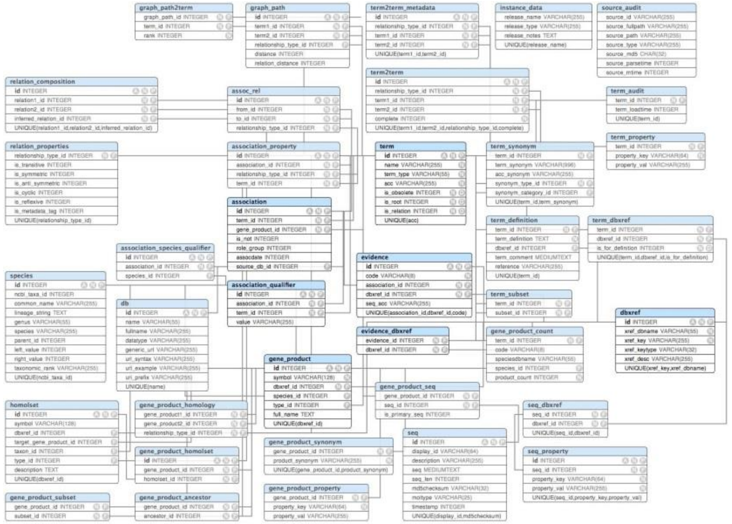


Figure 1: Entity Relationship Diagram (ERD) for the rec_data database.

And this …



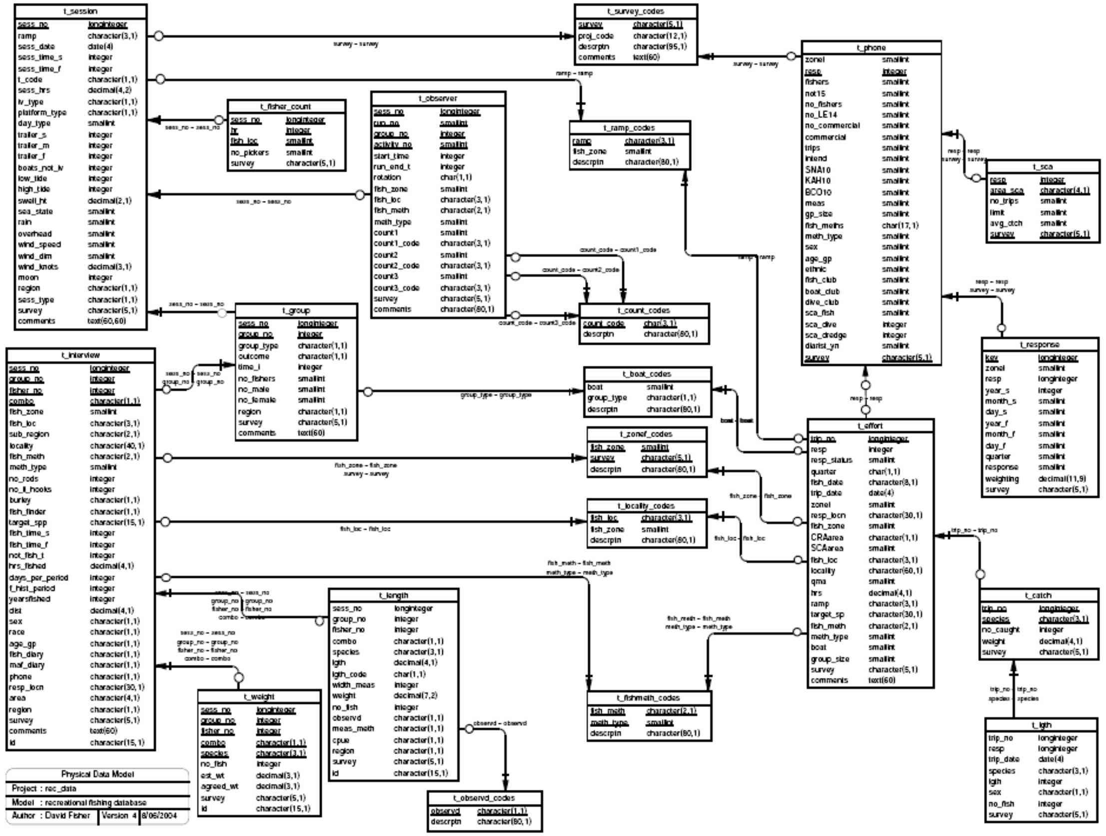
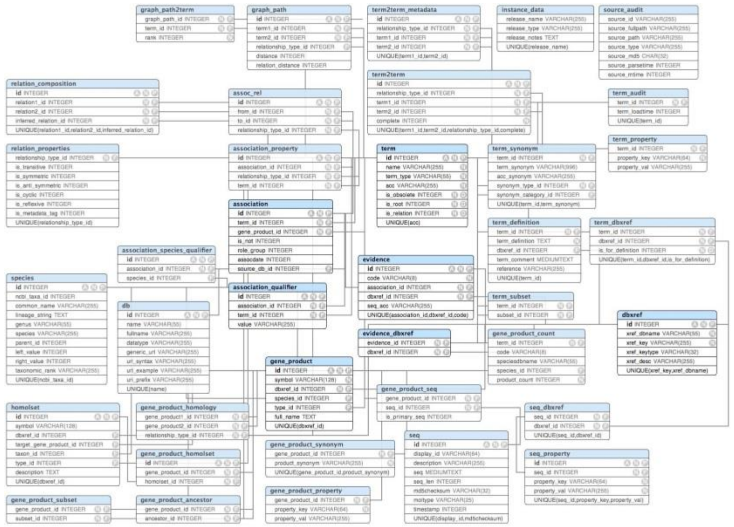


Figure 1: Entity Relationship Diagram (ERD) for the rec_data database.

And this …



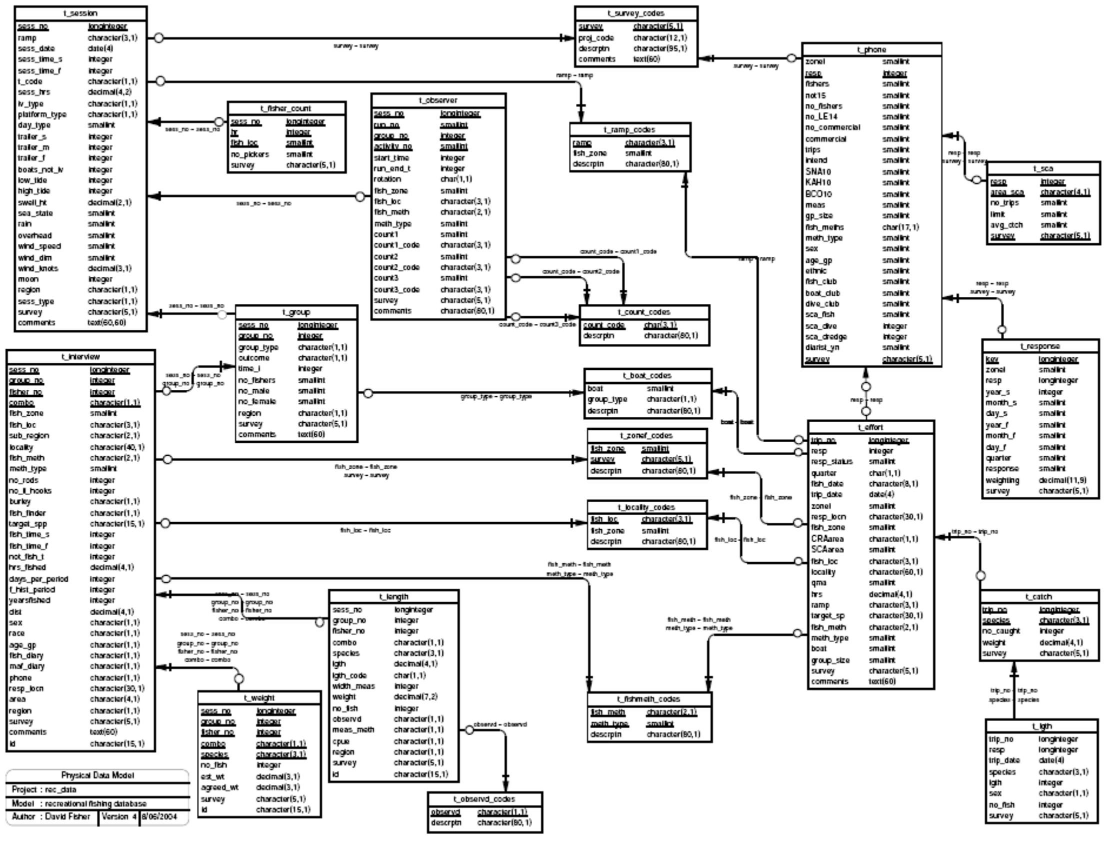
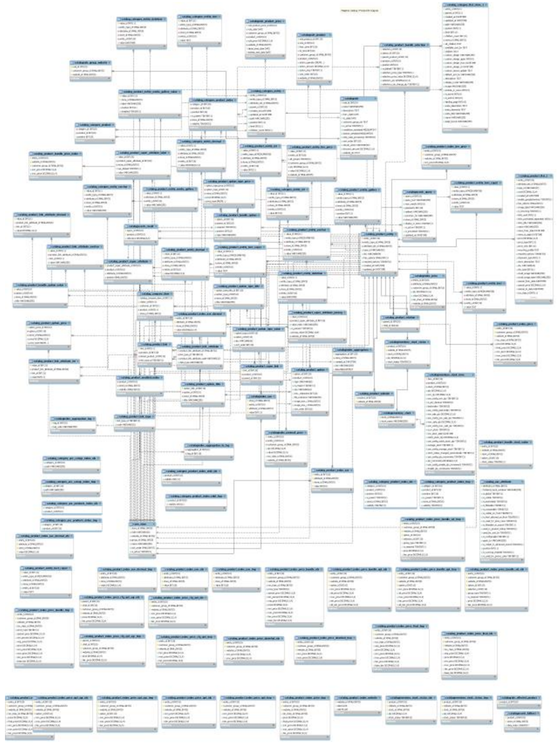
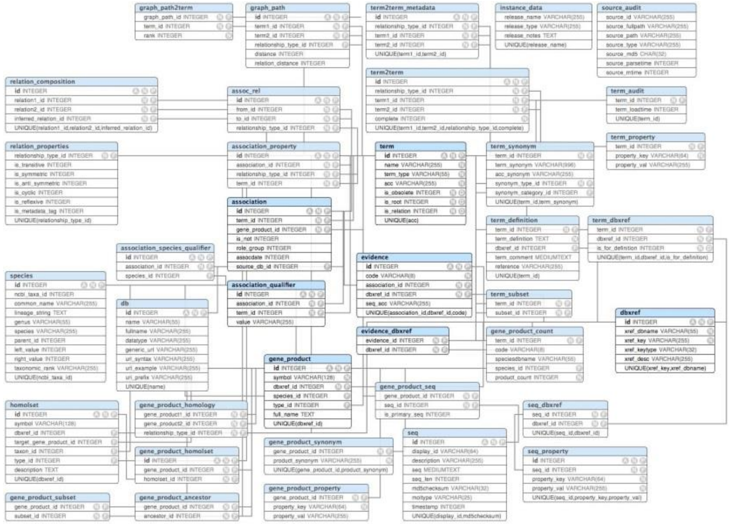


Figure 1: Entity Relationship Diagram (ERD) for the rec_data database.



Consolidating schemas is very hard.



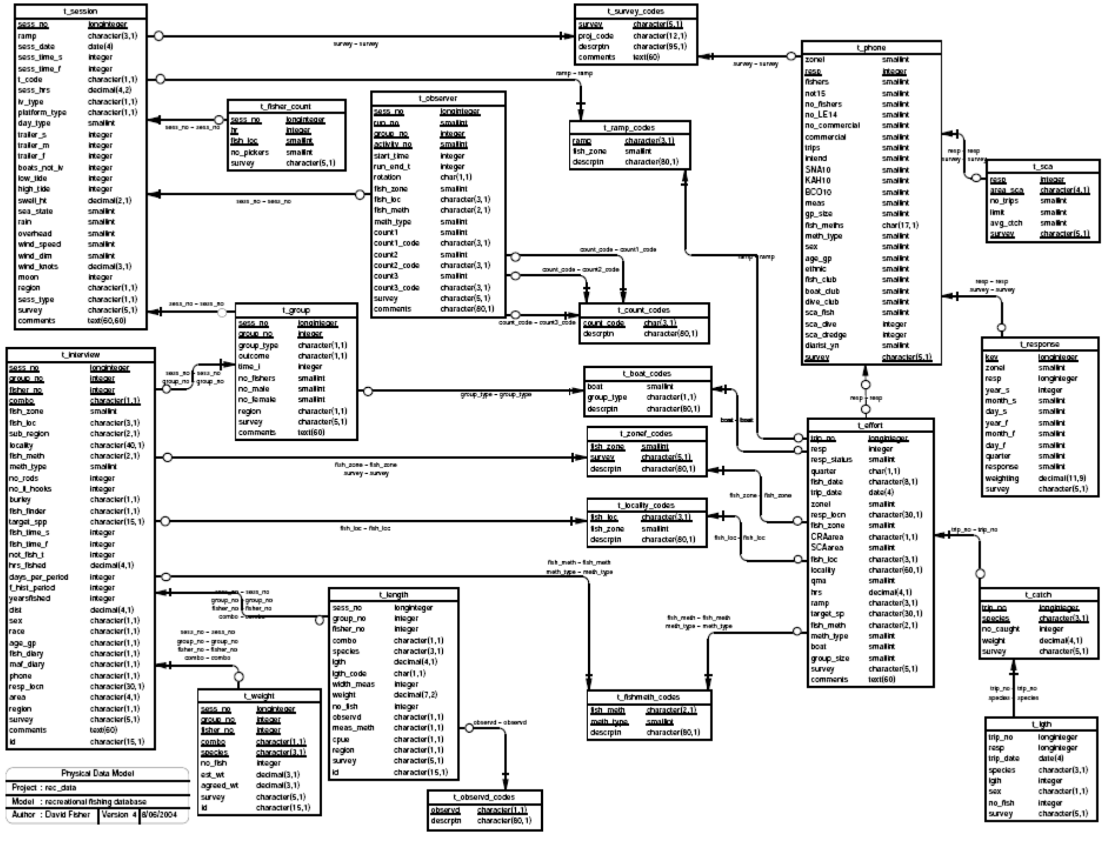
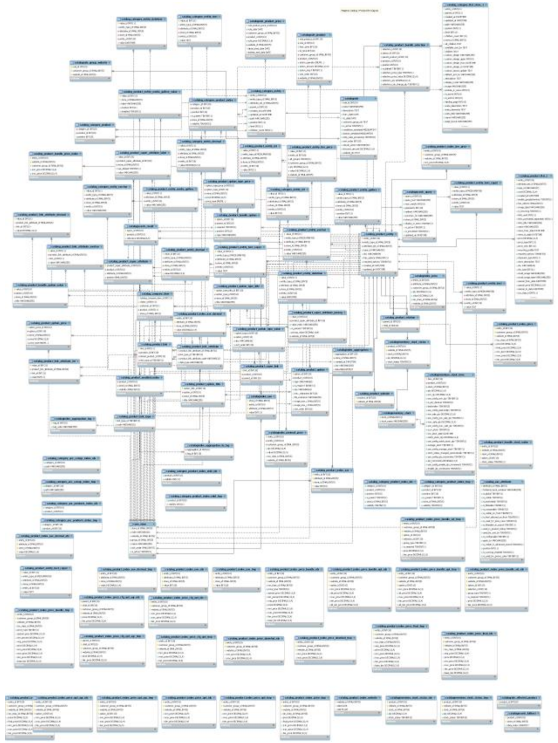
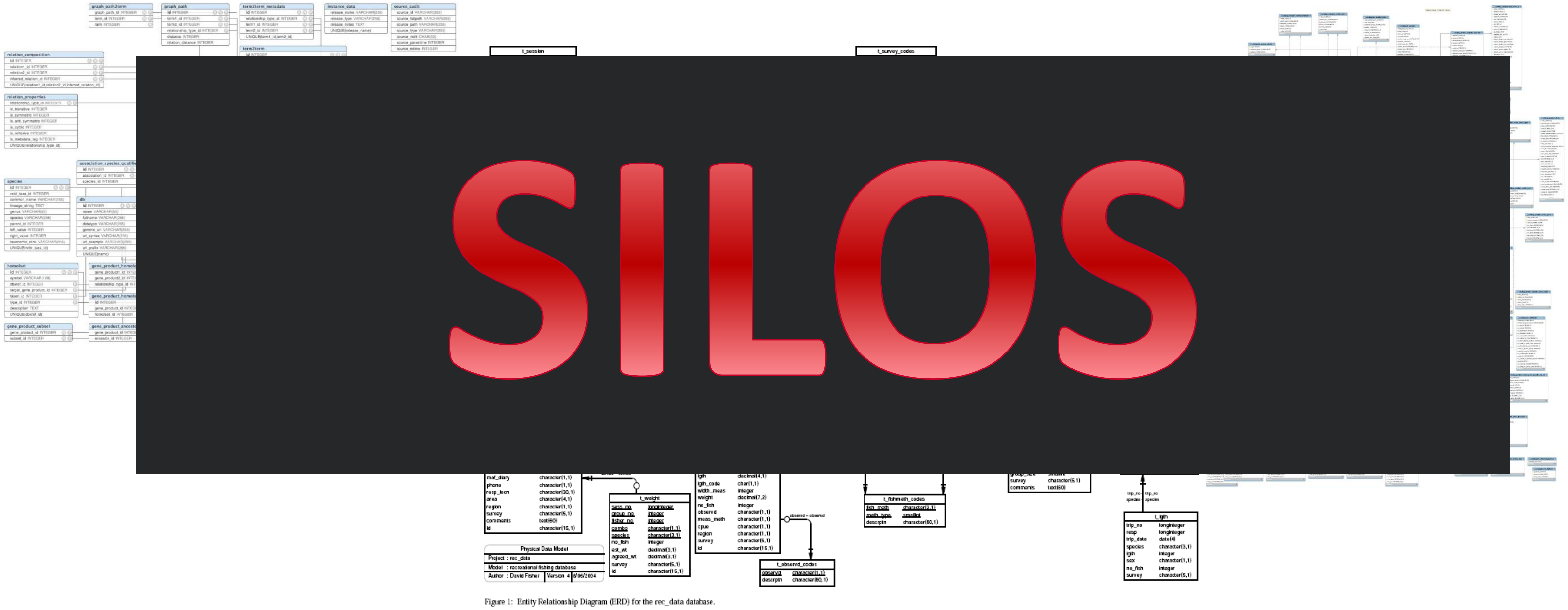


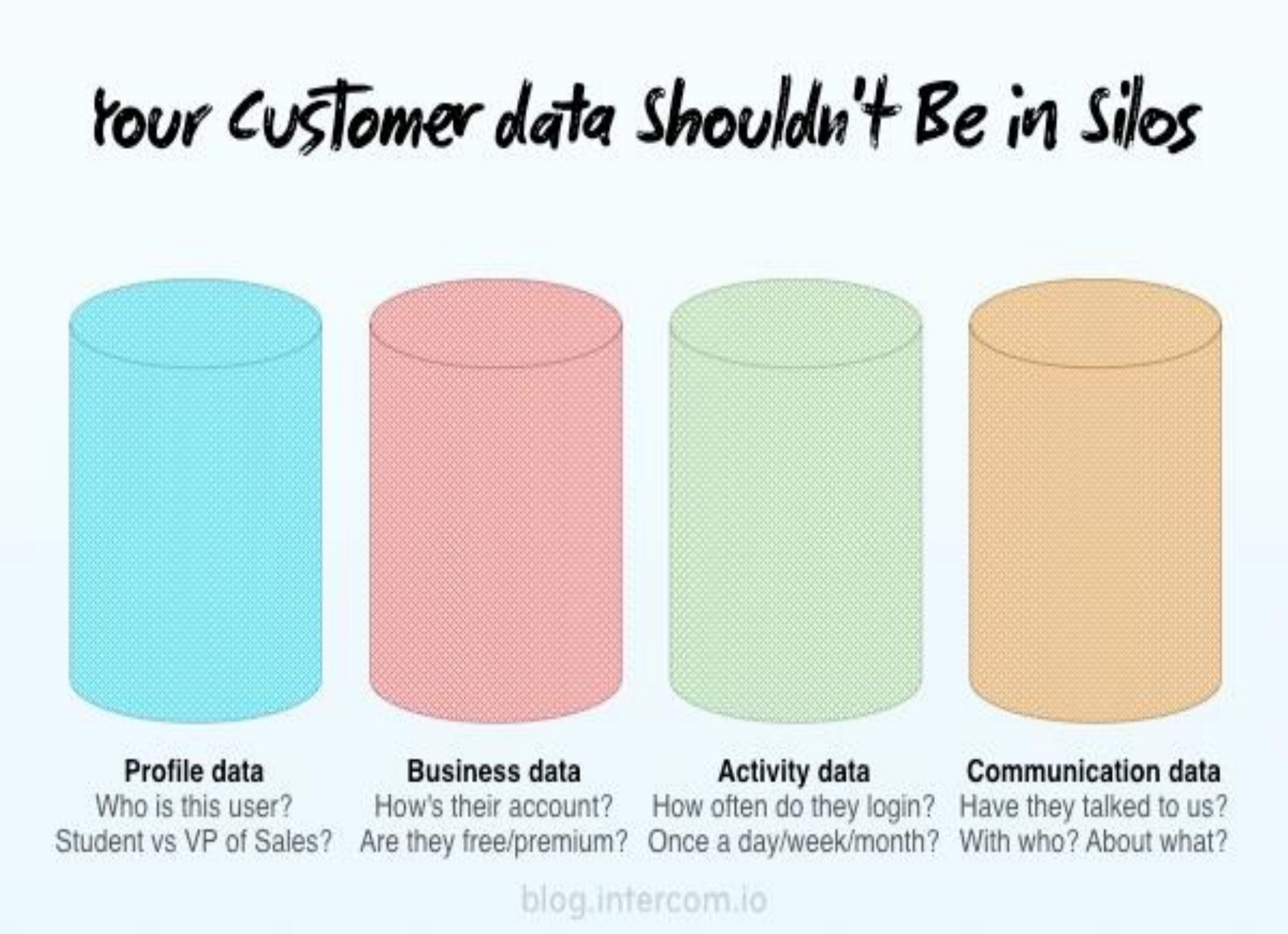
Figure 1: Entity Relationship Diagram (ERD) for the rec_data database.



Consolidating schemas is very hard, causes SILOs



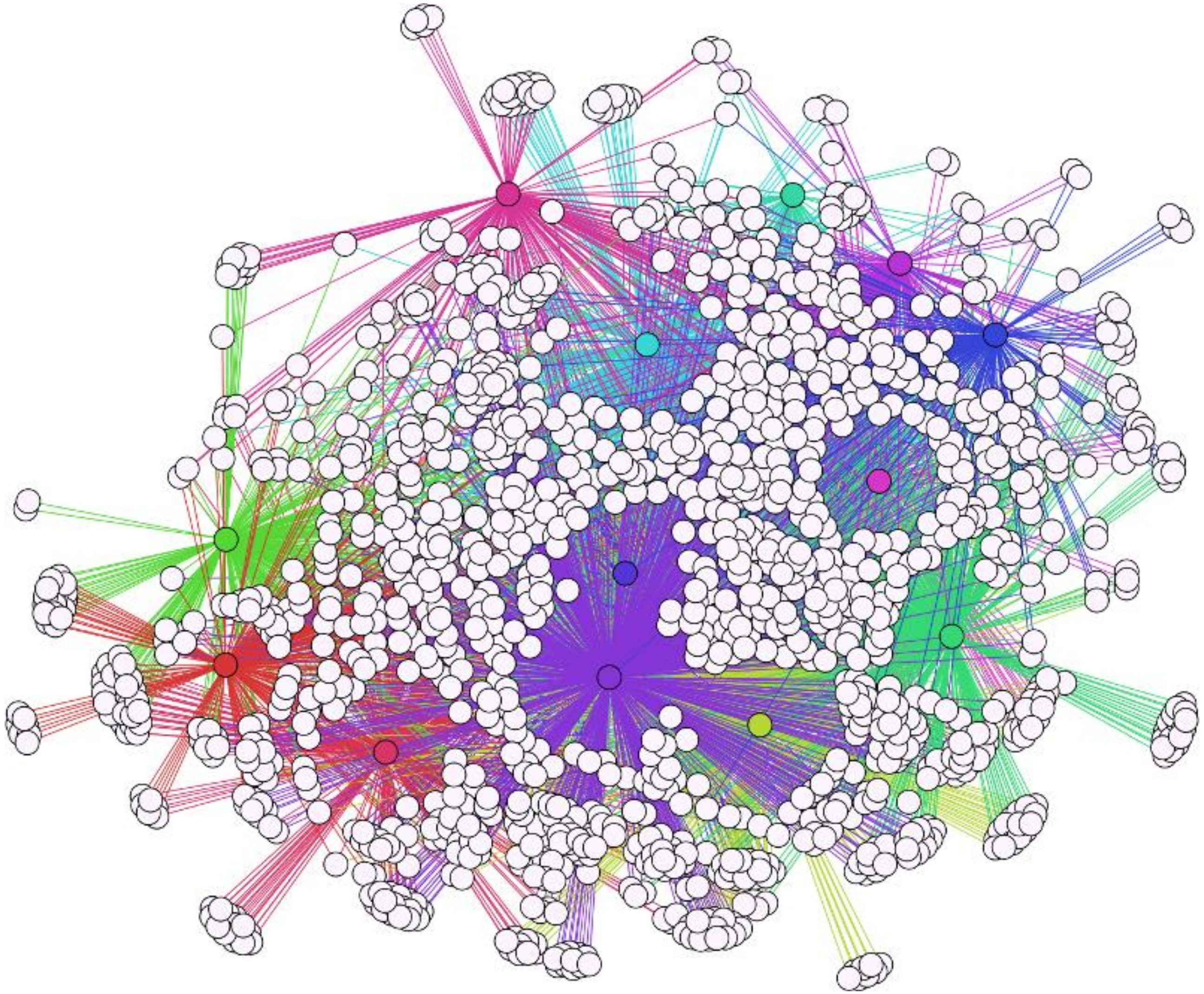
Silos make analysis very difficult



How do I identify a unique {customer, trade} across data sets?

How can I guarantee the lack of anomalous behavior if I can't see all data?

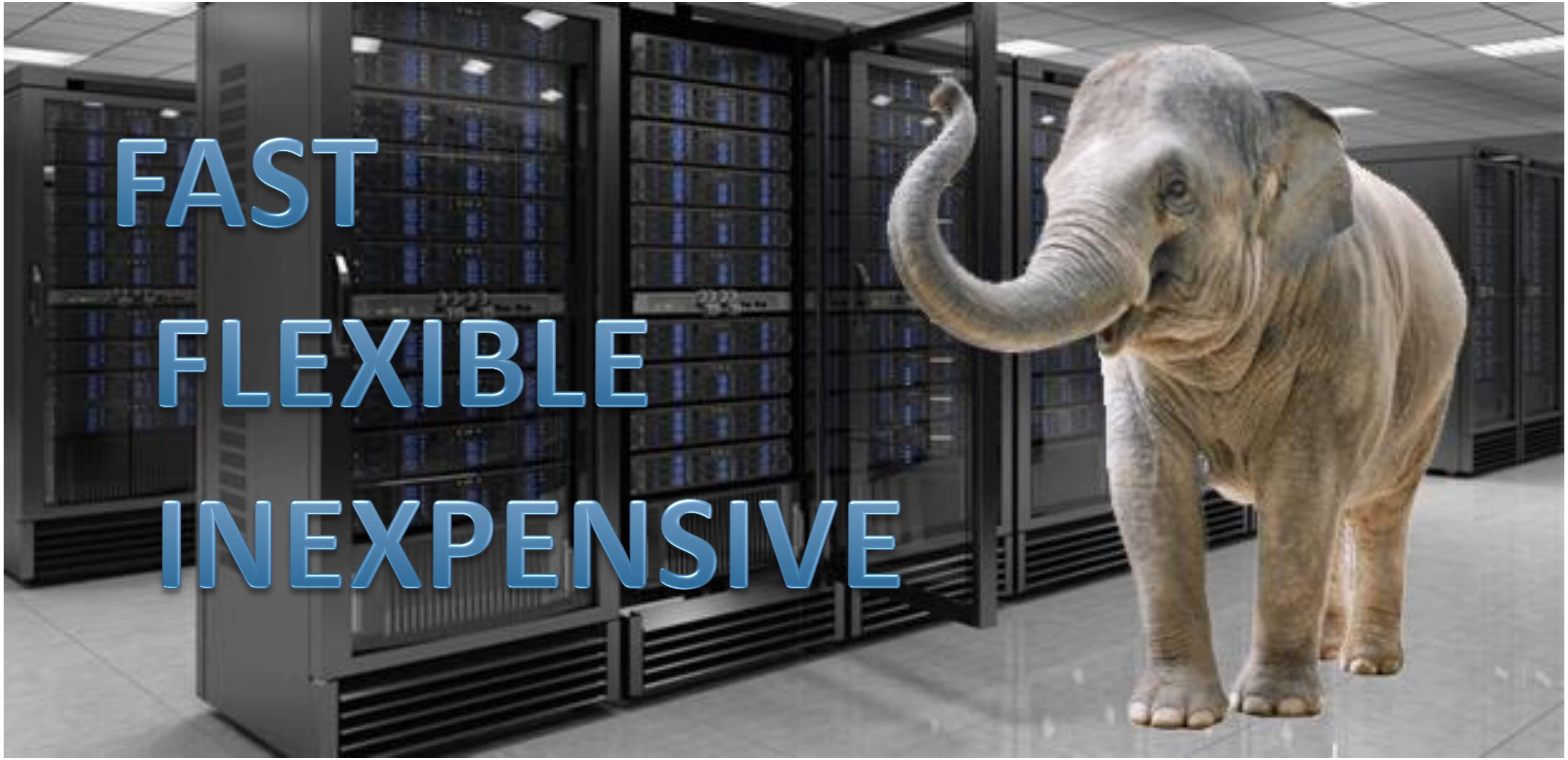
Hard to know what's of value *a priori*



Hard to know what's of value *a priori*

You want to keep all the data but it has to be economical

Why Hadoop



Rethink SQL for Big Data

Preserve

- **ANSI SQL**
  - Familiar and ubiquitous
- **Performance**
  - Interactive nature crucial for BI/Analytics
- **One technology**
  - Painful to manage different technologies
- **Enterprise ready**
  - System-of-record, HA, DR, Security, Multi-tenancy, …

Rethink SQL for Big Data

Preserve

- **ANSI SQL**
  - Familiar and ubiquitous
- **Performance**
  - Interactive nature crucial for BI/Analytics
- **One technology**
  - Painful to manage different technologies
- **Enterprise ready**
  - System-of-record, HA, DR, Security, Multi-tenancy, …

Invent

- **Flexible data-model**
  - Allow schemas to evolve rapidly
  - Support semi-structured data types
- **Agility**
  - Self-service possible when developer and DBA is same
- **Scalability**
  - In all dimensions: data, speed, schemas, processes, management

SQL is here to stay

Hadoop is here to stay


YOU CAN'T HANDLE REAL SQL

SQL

```
select  *  from  A
where  exists (
    select 1 from B where B.b < 100 );
```

- Did you know Apache HIVE cannot compute it?
  - eg, Hive, Impala, Spark/Shark

Self-described Data

```
select  cf.month,  cf.year
 from   hbase.table1;
```

- Did you know normal SQL cannot handle the above?
- Nor can HIVE and its variants like Impala, Shark?

Self-described Data

```
select  cf.month,  cf.year
  from   hbase.table1;
```

- Why?
- Because there's no meta-store definition available

Self-Describing Data Ubiquitous

Apache Drill

Centralized schema

- Static
- Managed by the DBAs
- In a centralized repository

Long, meticulous data preparation process (ETL, create/alter schema, etc.)
– can take 6-18 months

Self-describing, or schema-less, data

- Dynamic/evolving
- Managed by the applications
- Embedded in the data

Less schema, more suitable for data that has higher volume, variety and velocity

A Quick Tour through Apache Drill

Data Source is in the Query

```
select  timestamp, message
from    dfs1.logs.`AppServerLogs/2014/Jan/p001.parquet`
where   errorLevel > 2
```

Data Source is in the Query

```
select    timestamp, message
from      dfs1.logs.`AppServerLogs/2014/Jan/p001.parquet`
where     errorLevel > 2
```

This is a *cluster* in Apache Drill

- DFS
- HBase
- Hive meta-store

Data Source is in the Query

```
select    timestamp, message
from      dfs1.logs.`AppServerLogs/2014/Jan/p001.parquet`
where     errorLevel > 2
```

This is a *cluster* in Apache Drill
- DFS
- HBase
- Hive meta-store

A *work-space*
- Typically a sub-directory
- HIVE database

Data Source is in the Query

```
select    timestamp, message
from      dfs1.logs.`AppServerLogs/2014/Jan/p001.parquet`
where     errorLevel > 2
```

This is a *cluster* in Apache Drill
- DFS
- HBase
- Hive meta-store

A *work-space*
- Typically a sub-directory
- HIVE database

A *table*
- pathnames
- Hbase table
- Hive table

Combine data sources on the fly

- JSON
- CSV
- ORC (ie, all Hive types)
- Parquet
- HBase tables
- … can combine them

```
Select  USERS.name,  USERS.emails.work
from
  dfs.logs.`/data/logs`  LOGS,
  dfs.users.`/profiles.json`  USERS,
where
   LOGS.uid = USERS.uid   and
   errorLevel > 5
order by  count(*);
```

Can be an entire directory tree

// On a file

```
select      errorLevel, count(*)
from        dfs.logs.`/AppServerLogs/2014/Jan/part0001.parquet`
group by    errorLevel;
```

Can be an entire directory tree

// On a file

```
select      errorLevel, count(*)
from        dfs.logs.`/AppServerLogs/2014/Jan/part0001.parquet`
group by    errorLevel;
```

// On the entire data collection: all years, all months

```
select      errorLevel, count(*)
from        dfs.logs.`/AppServerLogs`
group by    errorLevel
```

Can be an entire directory tree

// On a file

dirs[1] dirs[2]

```
select     errorLevel, count(*)
from       dfs.logs.`/AppServerLogs/2014/Jan/part0001.parquet`
group by   errorLevel;
```

// On the entire data collection: all years, all months

```
select     errorLevel, count(*)
from       dfs.logs.`/AppServerLogs`
group by   errorLevel
```

Can be an entire directory tree

// On a file

dirs[1] dirs[2]

```
select     errorLevel, count(*)
from       dfs.logs.`/AppServerLogs/2014/Jan/part0001.parquet`
group by   errorLevel;
```

// On the entire data collection: all years, all months

```
select     errorLevel, count(*)
from       dfs.logs.`/AppServerLogs`  where dirs[1] > 2012
group by   errorLevel  , dirs[2]
```

Querying JSON

donuts.json

```
{  name: classic
    fillings: [
      { name: sugar cal:  400 }]}

{ name: choco
   fillings: [
      { name: sugar  cal: 400 }
      { name: chocolate cal: 300 }]}

{  name: bostoncreme
    fillings:  [
      { name:  sugar  cal: 400 }
      { name:  cream  cal: 1000 }
      { name:  jelly  cal: 600 }]}
```

Cursors inside Drill

```
{  name: classic
   fillings: [
     { name:  sugar cal:  400 }]}

{ name: choco
  fillings: [
     { name: sugar  cal: 400 }
     { name: chocolate cal: 300 }]}

{  name: bostoncreme
   fillings:  [
     { name:  sugar  cal: 400 }
     { name:  cream  cal: 1000 }
     { name:  jelly  cal: 600 }]}
```

```
DrillClient drill = new DrillClient().connect( …);
ResultReader  r =  drill.runSqlQuery( "select * from `donuts.json`");
while( r.next()) {
    String donutName = r.reader( “name").readString();
    ListReader fillings = r.reader( "fillings");
    while( fillings.next()) {
      int calories = fillings.reader( "cal").readInteger();
      if (calories > 400)
        print( donutName,  calories, fillings.reader( "name").readString());
    }
}
```

Direct queries on nested data

```
{  name: classic
   fillings: [
     { name:  sugar cal:  400 }]}

{ name: choco
  fillings: [
     { name: sugar  cal: 400 }
     { name: chocolate cal: 300 }]}

{  name: bostoncreme
   fillings:  [
     { name:  sugar  cal: 400 }
     { name:  cream  cal: 1000 }
     { name:  jelly  cal: 600 }]}
```

// Flattening maps in JSON, parquet and other nested records

```
select name, flatten(fillings) as f
  from   dfs.users.`/donuts.json`
  where  f.cal < 300;
```

// lists the fillings < 300 calories

Complex Data Using SQL or Fluent API

```
{  name: classic
   fillings: [
     { name:  sugar cal:  400 }]}

{ name: choco
  fillings: [
     { name: sugar  cal: 400 }
     { name: plain: 280 }]}

{  name: bostoncreme
   fillings:  [
     { name:  sugar  cal: 400 }
     { name:  cream  cal: 1000 }
     { name:  jelly  cal: 600 }]}
```

```
// SQL
Result r = drill.sql( "select name, flatten(fillings) from
  `donuts.json` where fillings.cal < 300`);

// or Fluent API
Result r = drill.table(“donuts.json”)
  .lt(“fillings.cal”, 300).all();

while( r.next()) {
  String name = r.get( “name").string();
  List fillings = r.get( “fillings”).list();
  while(fillings.next()) {
    print(name, calories, fillings.get(“name”).string());
  }
}
```


MAPR

Queries on embedded data

// embedded JSON value inside column *donut-json* inside column-family *cf1* of an hbase table *donuts*

```
select     d.name, count( d.fillings),
 from (
       select   convert_from( cf1.donut-json, json)   as  d
       from     hbase.user.`donuts` );
```

Queries inside JSON records

// Each JSON record itself can be a whole database

// example: get all donuts with at least 1 filling with > 300 calories

```
select    d.name, count( d.fillings),
          max(d.fillings.cal) within record as mincal
from (  select   convert_from( cf1.donut-json, json)   as  d
        from     hbase.user.`donuts` )
where   mincal > 300;
```

- Schema can change over course of query
- Operators are able to reconfigure themselves on schema change events
  - Minimize flexibility overhead
  - Support more advanced execution optimization based on actual data characteristics

De-centralized metadata

// count the number of tweets per customer, where the customers are in Hive, and their tweets are in HBase. Note that the hbase data has no meta-data information

```
select  c.customerName,  hb.tweets.count
from    hive.CustomersDB.`Customers` c
join    hbase.user.`SocialData` hb
on      c.customerId = convert_from( hb.rowkey, UTF-8);
```

So what does this all mean?

A Drill Database

- What is a database with Drill/MapR?

A Drill Database

- What is a database with Drill/MapR?
- Just a directory, with a bunch of related files

A Drill Database

- What is a database with Drill/MapR?
- Just a directory, with a bunch of related files
- There's no need for artificial boundaries
  - No need to bunch a set of tables together to call it a "database"

A Drill Database

/user/srivas/work/bugs

BugList

symptom	version	date	bugid	dump-name
impala crash	3.1.1	14/7/14	12345	cust1.tgz
cldb slow	3.1.0	12/7/14	45678	cust2.tgz

Customers

name	rep	se	dump-name
xxxx	dkim	junhyuk	cust1.tgz
yyyy	yoshi	aki	cust2.tgz

Queries are simple

```
select   b.bugid, b.symptom, b.date
from     dfs.bugs.'/Customers' c, dfs.bugs.'/BugList' b
where    c.dump-name = b.dump-name
```

Queries are simple

```
select   b.bugid, b.symptom, b.date
from     dfs.bugs.'/Customers' c, dfs.bugs.'/BugList' b
where    c.dump-name = b.dump-name
```

Let's say I want to cross-reference against your list:

```
select bugid, symptom
from dfs.bugs.'/Buglist' b,  dfs.yourbugs.'/YourBugFile' b2
where  b.bugid = b2.xxx
```

What does it mean?

What does it mean?

- No ETL
- Reach out directly to the particular table/file
- As long as the permissions are fine, you can do it
- No need to have the meta-data
  - None needed

Another example

```
select     d.name, count( d.fillings),
from (  select    convert_from( cf1.donut-json, json)   as  d
           from      hbase.user.`donuts` );
```

- convert_from(xx, json) invokes the json parser inside Drill

Another example

```
select     d.name, count( d.fillings),
from (  select    convert_from( cf1.donut-json, json)   as  d
           from      hbase.user.`donuts` );
```

- convert_from(xx, json) invokes the json parser inside Drill
- What if you could plug in any parser?

Another example

```
select     d.name, count( d.fillings),
from (  select    convert_from( cf1.donut-json, json)   as  d
             from      hbase.user.`donuts` );
```

- convert_from(xx, json) invokes the json parser inside Drill
- What if you could plug in any parser
  - XML?
  - Semi-conductor yield-analysis files?   Oil-exploration readings?
  - Telescope readings of stars?
  - RFIDs of various things?

No ETL

- Basically, Drill is querying the raw data directly
- Joining with processed data
- NO ETL
- Folks, this is very, very powerful
- NO ETL

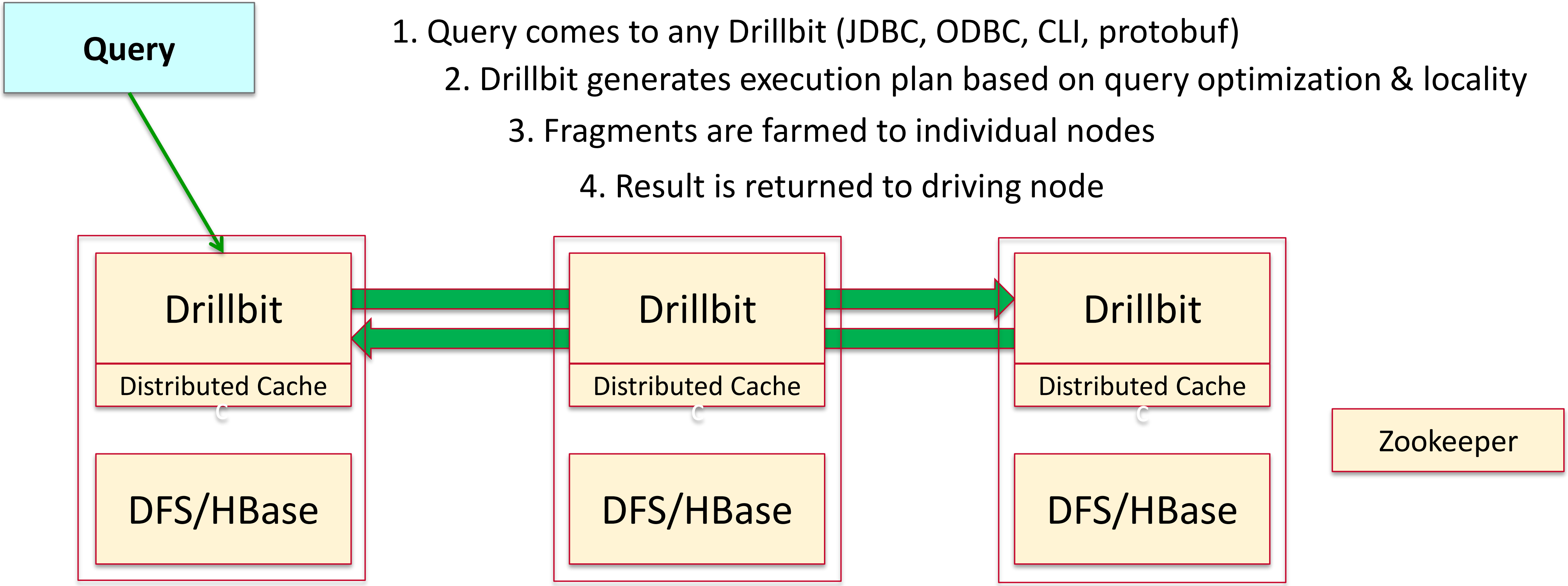
Seamless integration with Apache Hive

- Low latency queries on Hive tables
- Support for 100s of Hive file formats
- Ability to reuse Hive UDFs
- Support for multiple Hive metastores in a single query

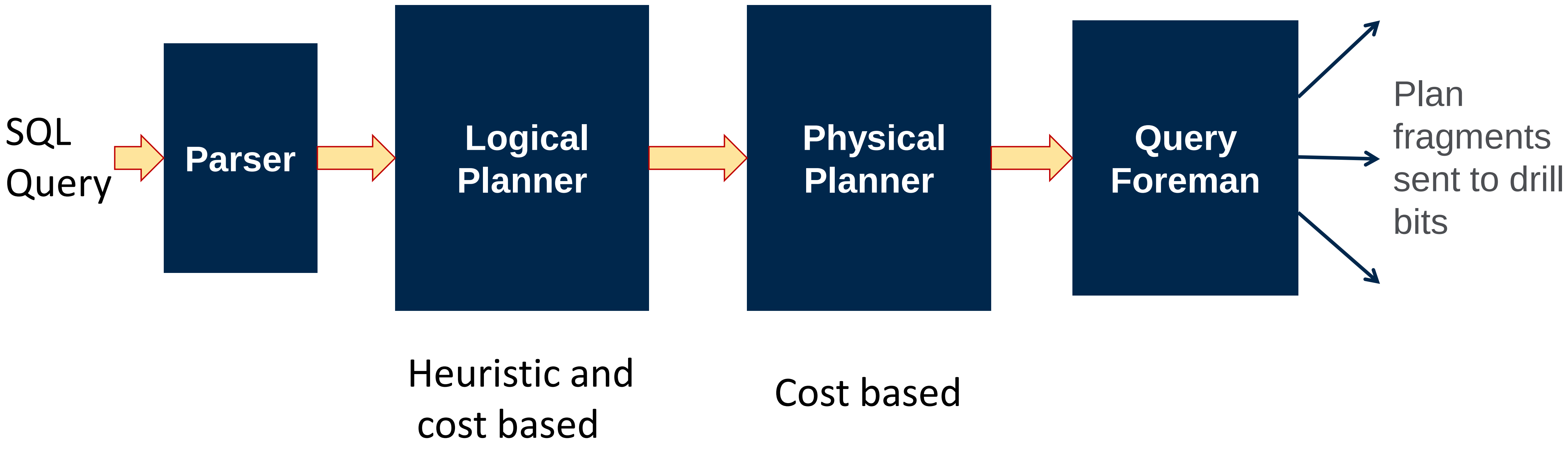
Un APACHE DRILL ers

Basic Process

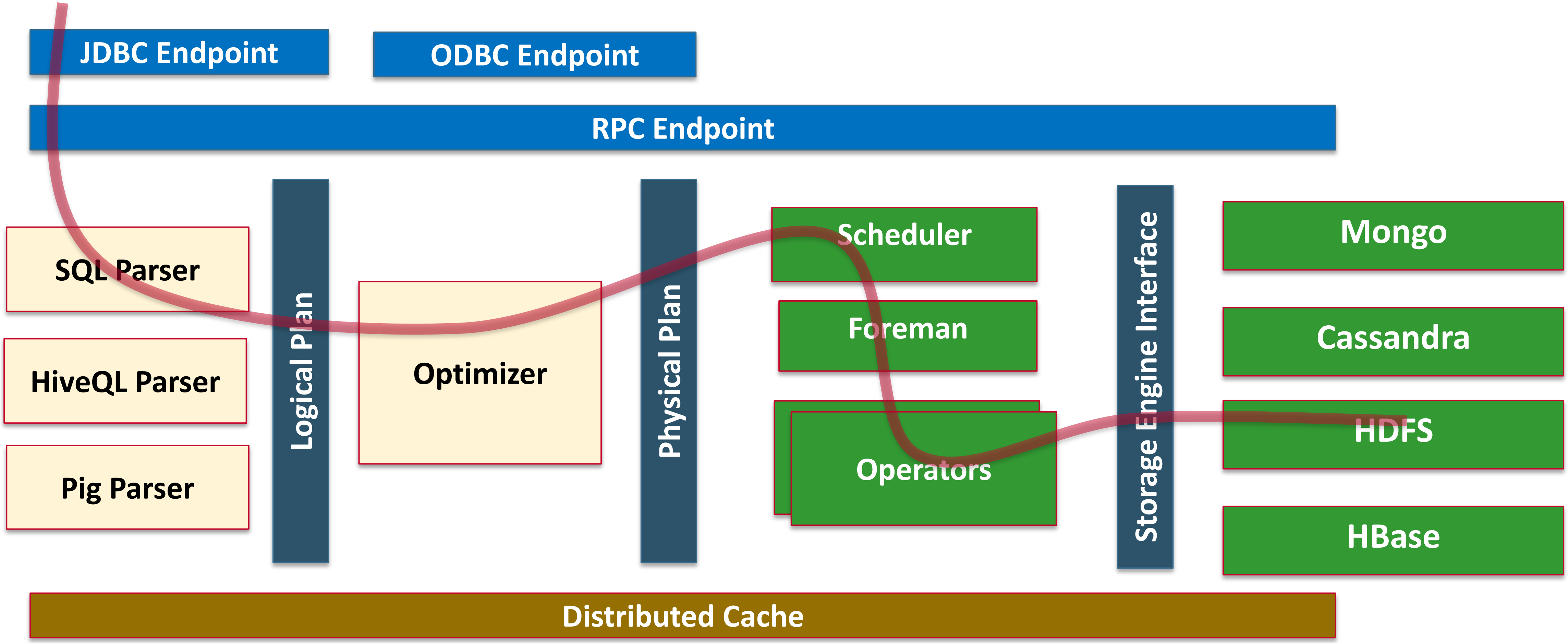
1. Query comes to any Drillbit (JDBC, ODBC, CLI, protobuf)
2. Drillbit generates execution plan based on query optimization & locality
3. Fragments are farmed to individual nodes
4. Result is returned to driving node



Stages of Query Planning



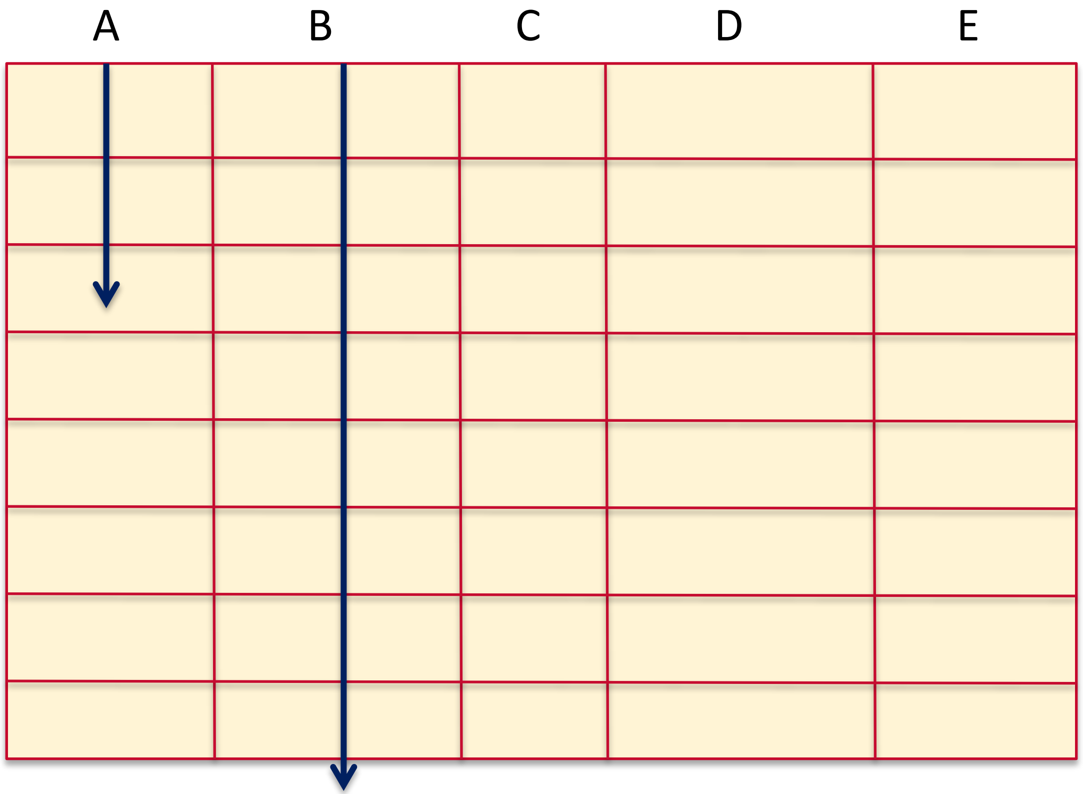
Query Execution

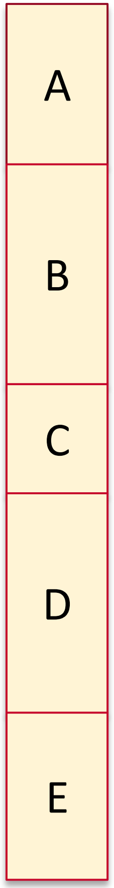


A Query engine that is…

- Columnar/Vectorized
- Optimistic/pipelined
- Runtime compilation
- Late binding
- Extensible

Columnar representation

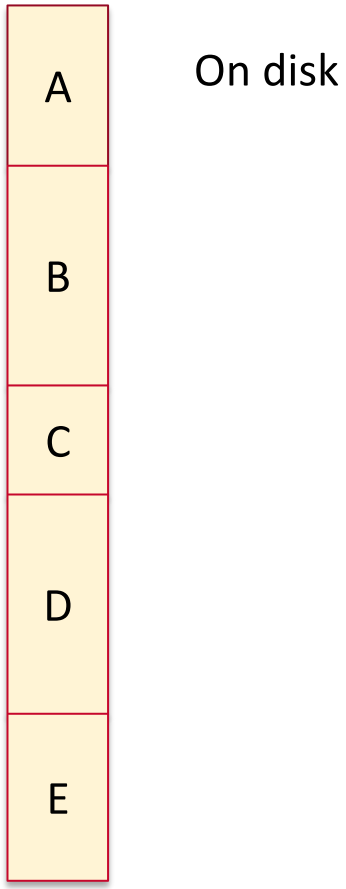




On disk

Columnar Encoding

- Values in a col. stored next to one-another
  – Better compression
  – Range-map: save min-max, can skip if not present
- Only retrieve columns participating in query
- Aggregations can be performed without decoding



Run-length-encoding & Sum

- Dataset encoded as <val> <run-length>:
  – 2, 4 (4 2's)
  – 8, 10 (10 8's)
- Goal: sum all the records
- Normally:
  – Decompress: 2, 2, 2, 2, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8
  – Add: 2 + 2 + 2 + 2 + 8 + 8 + 8 + 8 + 8 + 8 + 8 + 8 + 8 + 8
- Optimized work: 2 * 4 + 8 * 10
  – Less memory, less operations

Bit-packed Dictionary Sort

- Dataset encoded with a dictionary and bit-positions:
  - Dictionary: [Rupert, Bill, Larry] {0, 1, 2}
  - Values: [1,0,1,2,1,2,1,0]
- Normal work
  - Decompress & store: Bill, Rupert, Bill, Larry, Bill, Larry, Bill, Rupert
  - Sort: ~24 comparisons of variable width strings
- Optimized work
  - Sort dictionary: {Bill: 1, Larry: 2, Rupert: 0}
  - Sort bit-packed values
  - Work: max 3 string comparisons, ~24 comparisons of fixed-width dictionary bits

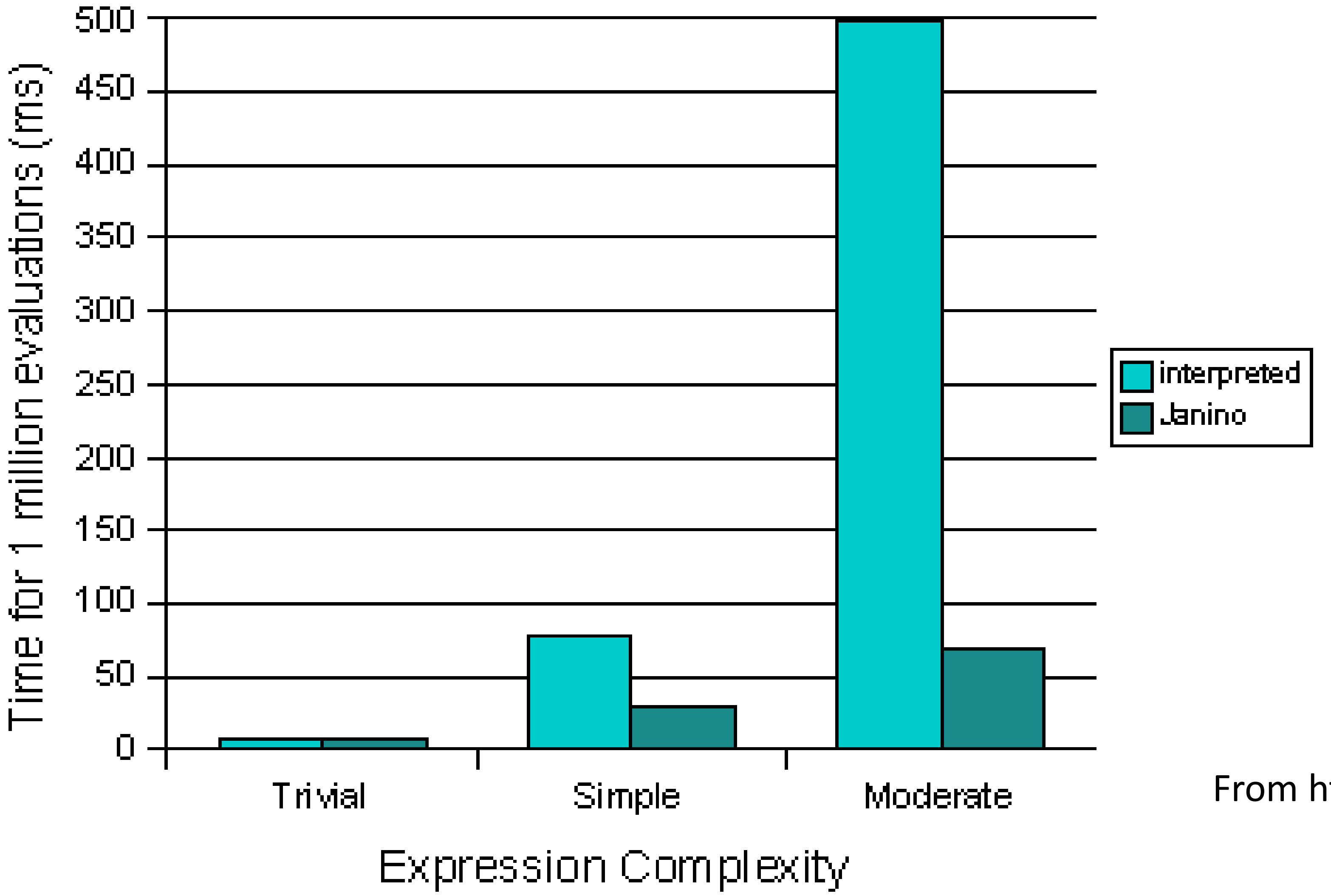
Drill 4-value semantics

- SQL's 3-valued semantics
  - True
  - False
  - Unknown
- Drill adds fourth
  - Repeated

Vectorization

- Drill operates on more than one record at a time
  - Word-sized manipulations
  - SIMD-like instructions
    - GCC, LLVM and JVM all do various optimizations automatically
  - Manually code algorithms
- Logical Vectorization
  - Bitmaps allow lightning fast null-checks
  - Avoid branching to speed CPU pipeline

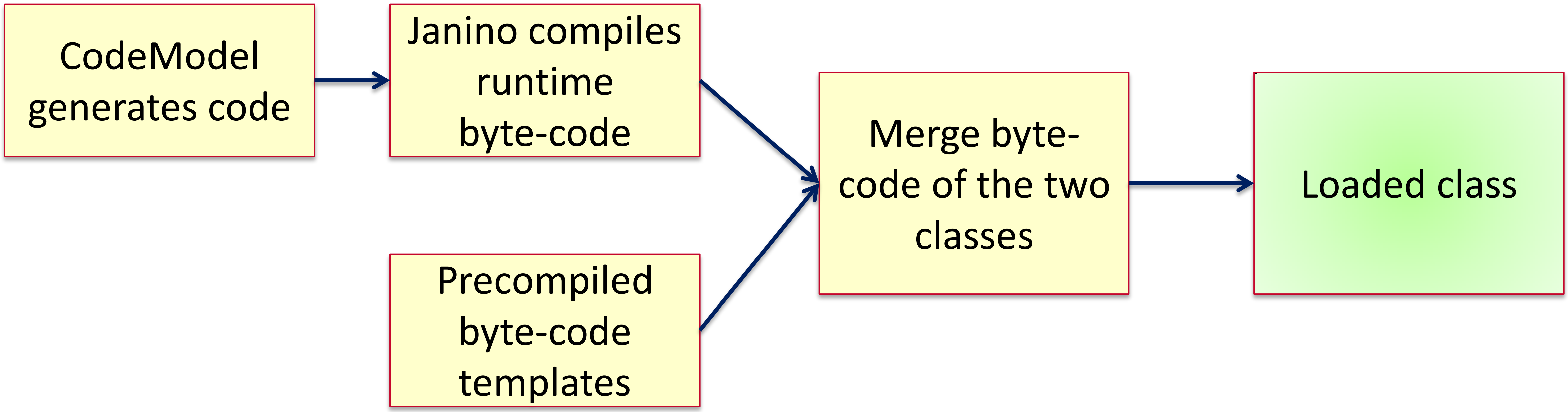
Runtime Compilation is Faster



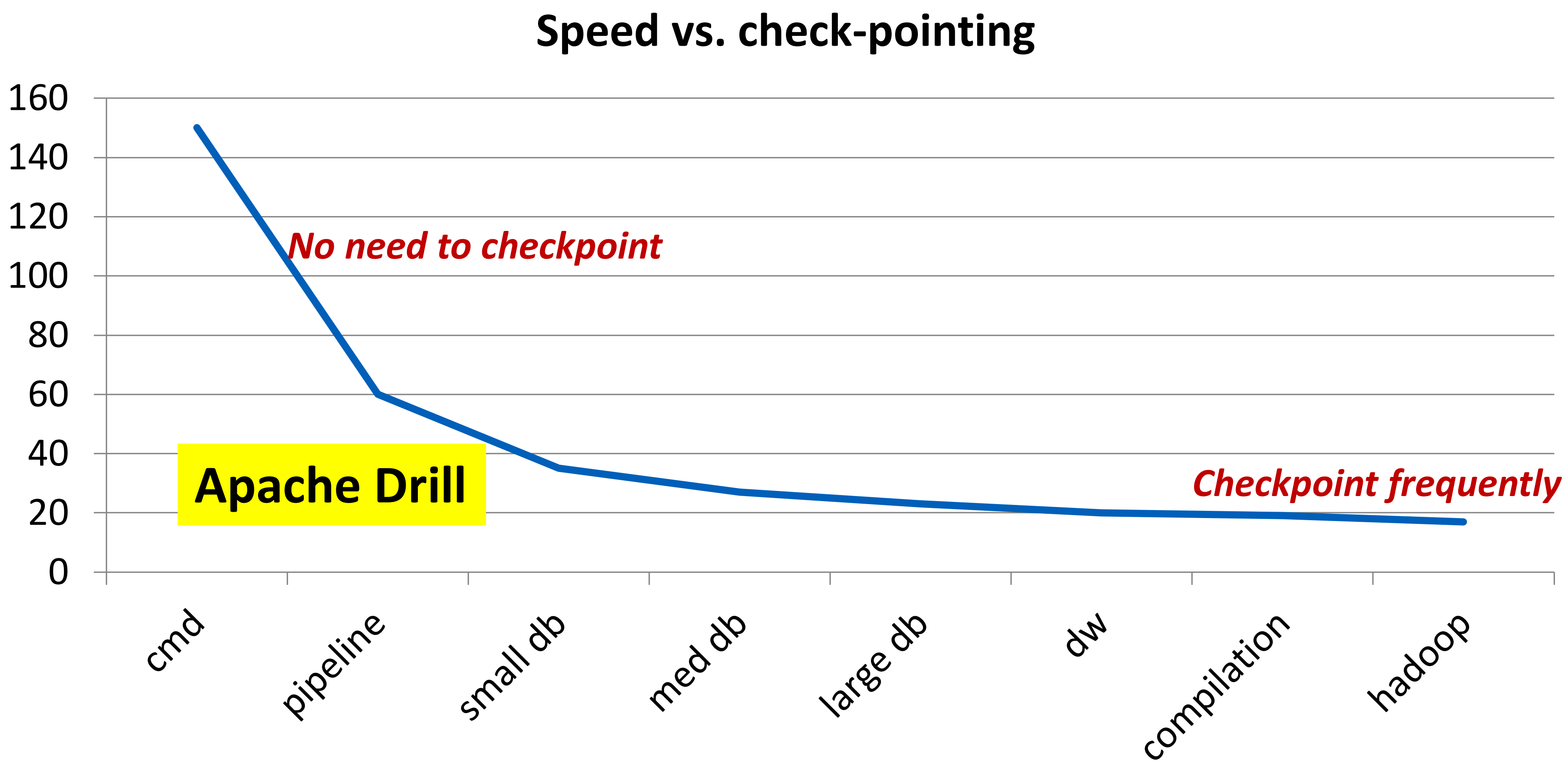
- JIT is smart, but more gains with runtime compilation
- Janino: Java-based Java compiler

From http://bit.ly/16Xk32x

Drill compiler



Optimistic


Speed vs. check-pointing
160
140
120
100
80
60
40
20
0
No need to checkpoint
Apache Drill
Checkpoint frequently
cmd
pipeline
small db
med db
large db
dw
compilation
hadoop

Optimistic Execution

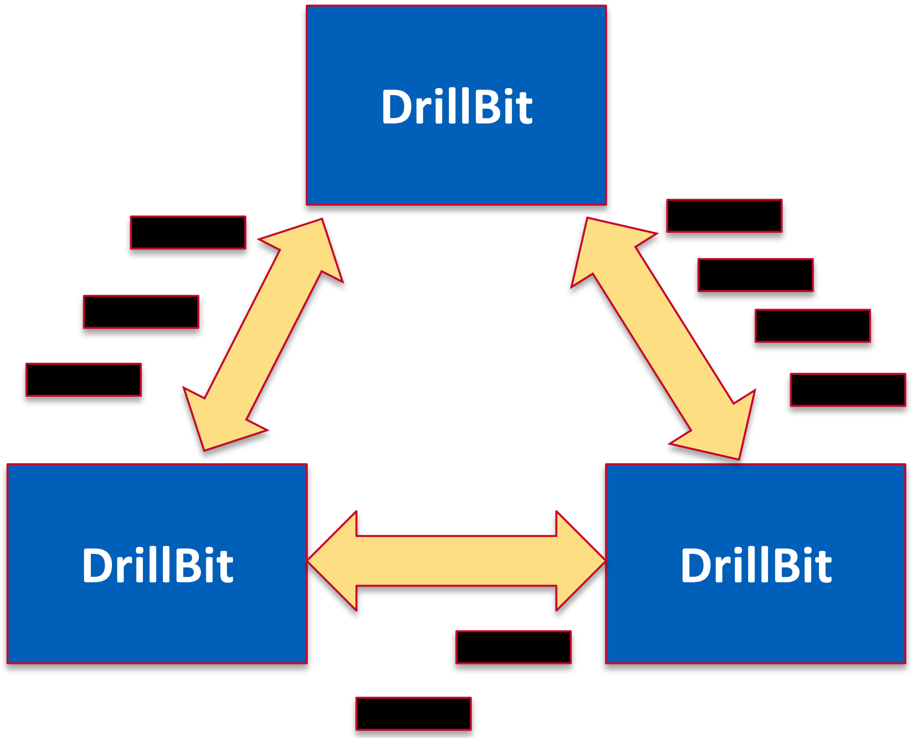
- Recovery code trivial
  – Running instances discard the failed query's intermediate state
- Pipelining possible
  – Send results as soon as batch is large enough
  – Requires barrier-less decomposition of query

Batches of Values

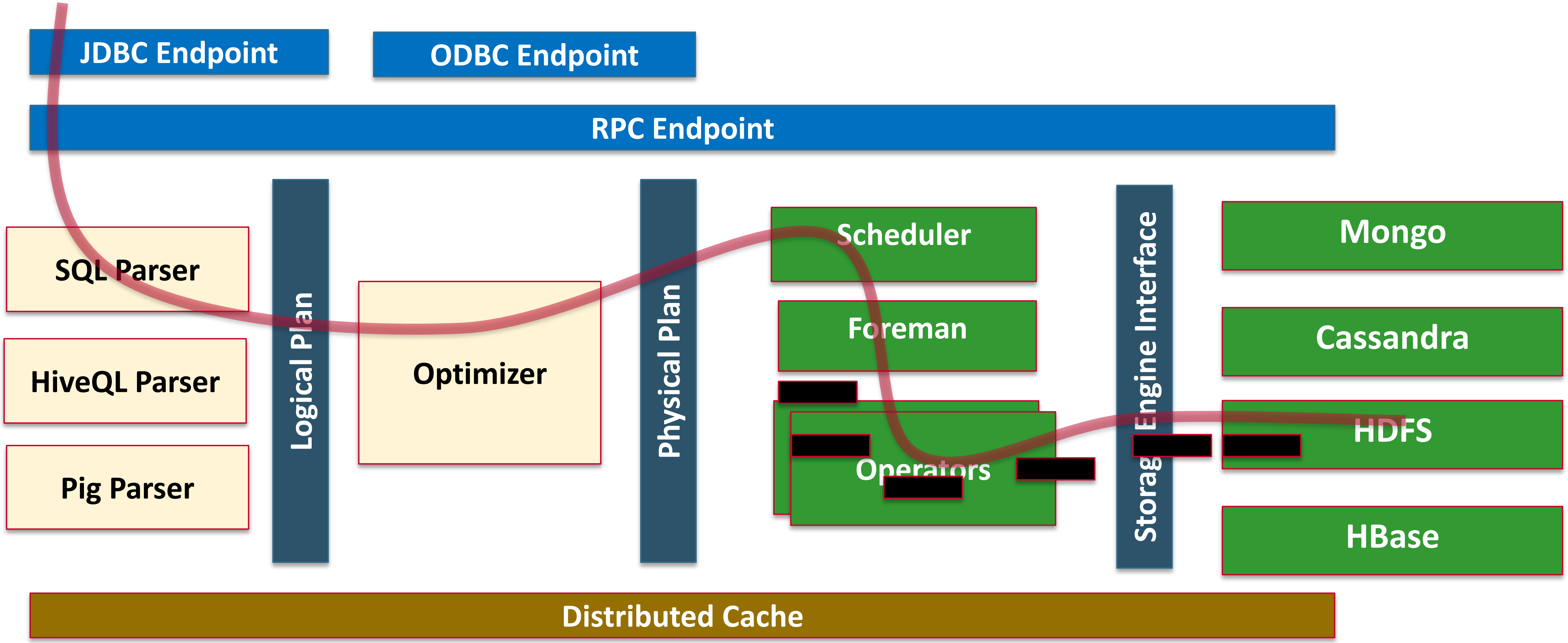
- *Value vectors*
  - List of values, with same schema
  - With the 4-value semantics for each value
- Shipped around in batches
  - max 256k bytes in a batch
  - max 64K rows in a batch
- RPC designed for multiple replies to a request

Pipelining

- Record batches are pipelined between nodes
  – ~256kB usually
- Unit of work for Drill
  – Operators works on a batch
- Operator reconfiguration happens at batch boundaries

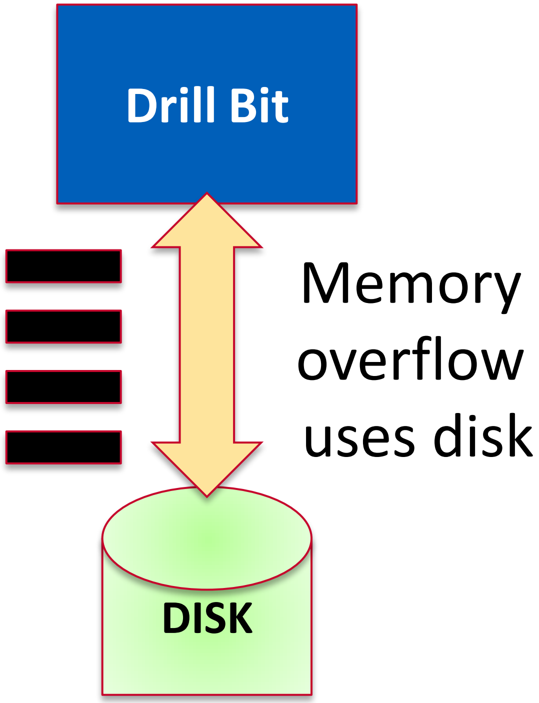


Pipelining Record Batches



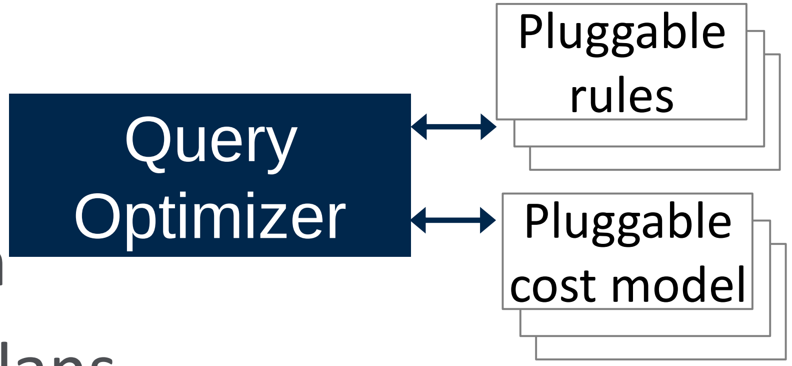
Pipelining

- Random access: sort without copy or restructuring
- Avoids serialization/deserialization
- Off-heap (no GC woes when lots of memory)
- Full specification + off-heap + batch
  - Enables C/C++ operators *(fast!)*
- Read/write to disk
  - when data larger than memory



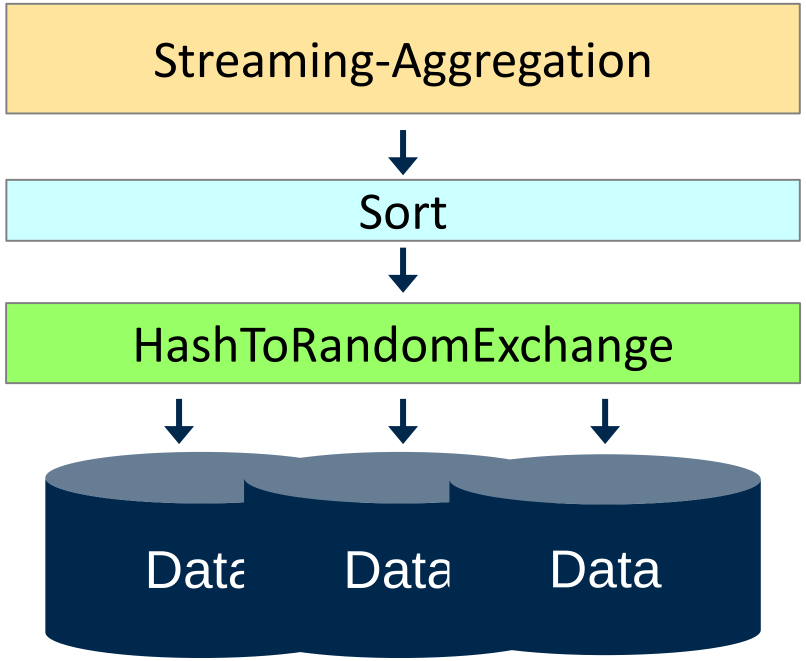
Cost-based Optimization

- Using Optiq, an extensible framework
  - Pluggable rules, and cost model
- Rules for distributed plan generation
  - Insert ***Exchange*** operator into physical plan
  - Optiq enhanced to explore parallel query plans
- Pluggable cost model
  - CPU, IO, memory, network cost (**data locality**)
  - Storage engine features (**HDFS** vs **HIVE** vs **HBase**)



Distributed Plan Cost

- Operators have *distribution* property
  - Hash, Broadcast, Singleton, …
- *Exchange* operator to enforce distributions
  - Hash: HashToRandomExchange
  - Broadcast: BroadcastExchange
  - Singleton: UnionExchange, SingleMergeExchange
- **Enumerate all, use cost to pick best**
  - Merge Join vs Hash Join
  - Partition-based join vs Broadcast-based join
  - Streaming Aggregation vs Hash Aggregation
- Aggregation in one phase or two phases
  - partial local aggregation followed by final aggregation



MAPR

Interactive SQL-on-Hadoop options

	Drill 1.0	Hive 0.13 w/ Tez	Impala 1.x
Latency	Low	Medium	Low
Files	Yes (all Hive file formats, plus JSON, Text, …)	Yes (all Hive file formats)	Yes (Parquet, Sequence, …)
HBase/MapR-DB	Yes	Yes, perf issues	Yes, with issues
Schema	Hive or schema-less	Hive	Hive
SQL support	ANSI SQL	HiveQL	HiveQL (subset)
Client support	ODBC/JDBC	ODBC/JDBC	ODBC/JDBC
Hive compat	High	High	Low
Large datasets	Yes	Yes	Limited
Nested data	Yes	Limited	No
Concurrency	High	Limited	Medium

Apache Drill Roadmap

1.0

Data exploration/ad-hoc queries

- Low-latency SQL
- Schema-less execution
- Files & HBase/M7 support
- Hive integration
- BI and SQL tool support via ODBC/JDBC

1.1

Advanced analytics and operational data

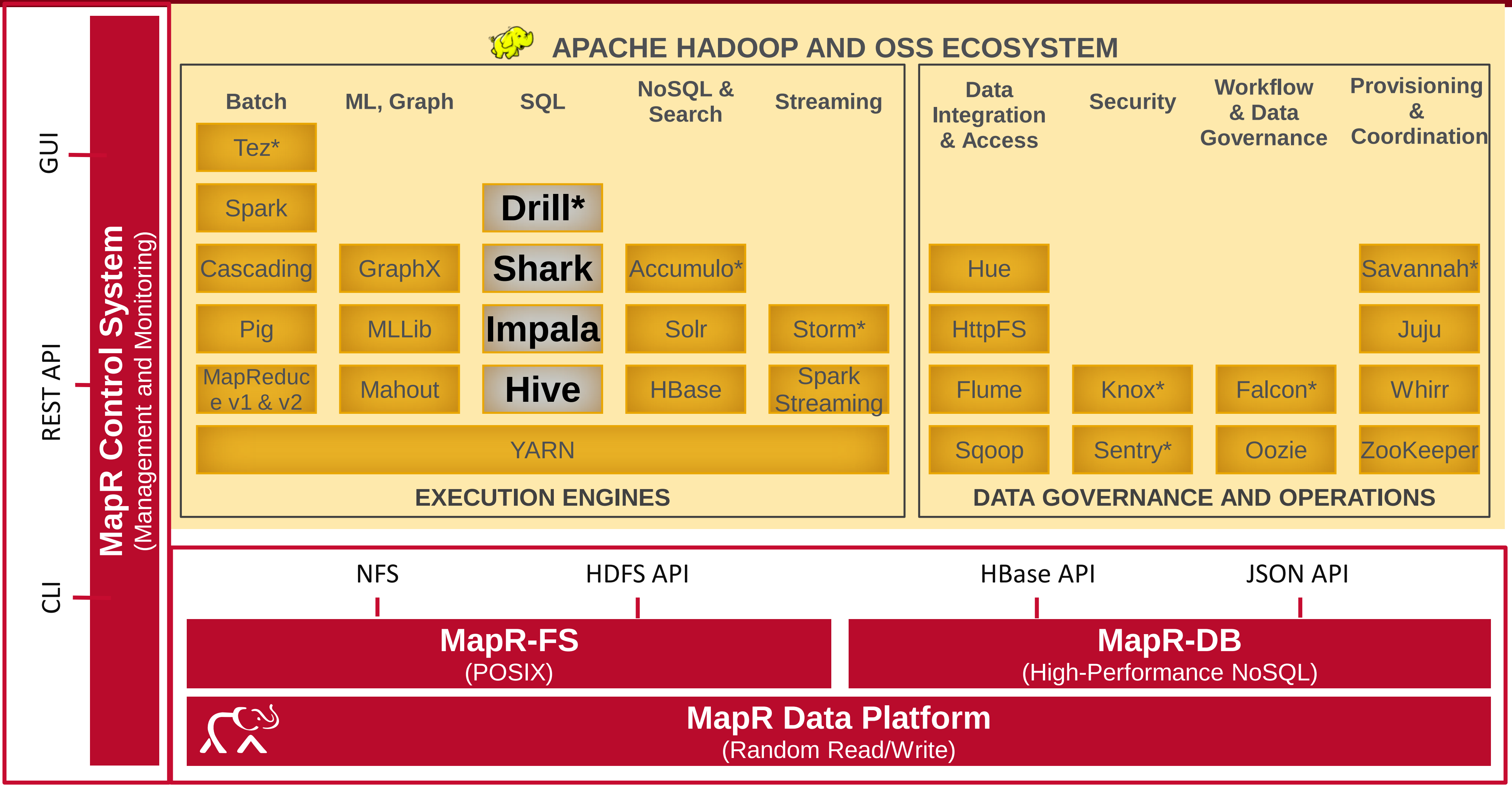
- HBase query speedup
- Nested data functions
- Advanced SQL functionality

2.0

Operational SQL

- Ultra low latency queries
- Single row insert/update/delete
- Workload management

MapR Distribution for Apache Hadoop



Enterprise Grade **Data Hub** **Operational**

* In Roadmap for inclusion/certification

Apache Drill Resources

- Drill 0.5 released last week
- Getting started with Drill is easy
  - just download tarball and start running SQL queries on local files

- Mailing lists
  - drill-user@incubator.apache.org
  - drill-dev@incubator.apache.org
- Docs: https://cwiki.apache.org/confluence/display/DRILL/Apache+Drill+Wiki
- Fork us on GitHub: http://github.com/apache/incubator-drill/
- Create a JIRA: https://issues.apache.org/jira/browse/DRILL

Active Drill Community

- Large community, growing rapidly
  – 35-40 contributors, 16 committers
  – Microsoft, Linked-in, Oracle, Facebook, Visa, Lucidworks, Concurrent, many universities
- In 2014
  – over 20 meet-ups, many more coming soon
  – 3 hackathons, with 40+ participants
- Encourage you to join, learn, contribute and have fun …

Drill at MapR

- World-class SQL team, ~20 people
- 150+ years combined experience building commercial databases
  - Oracle, DB2, ParAccel, Teradata, SQLServer, Vertica
- Team works on Drill, Hive, Impala
- Fixed some of the toughest problems in Apache Hive

Thank you!

Did I mention we are hiring…

M. C. Srivas
srivas@mapr.com