Apache Drill

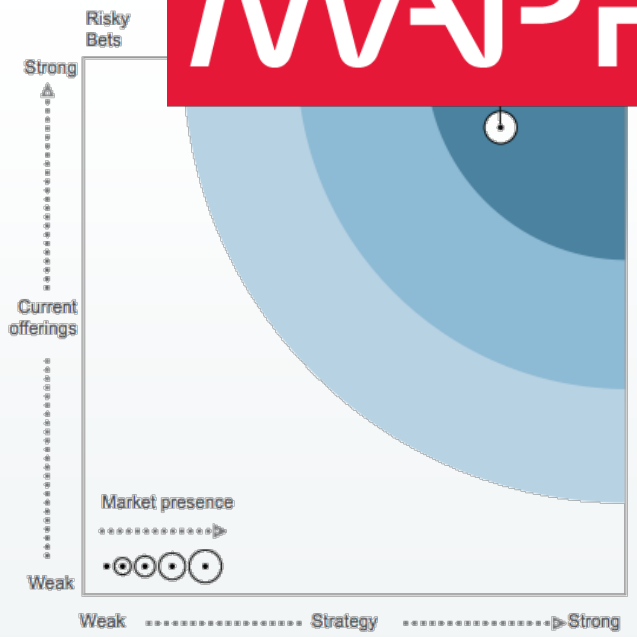
M. C. Srivas, CTO & co-Founder

May 14, 2014

MapR Enterprise Hadoop

Top Ranked

MAPR®



FORRESTER®

Cloud Leaders

amazon web services™

Google

500+ Customers

comScore

hp

URBAN AIRSHIP

LIVE NATION ENTERTAINMENT®

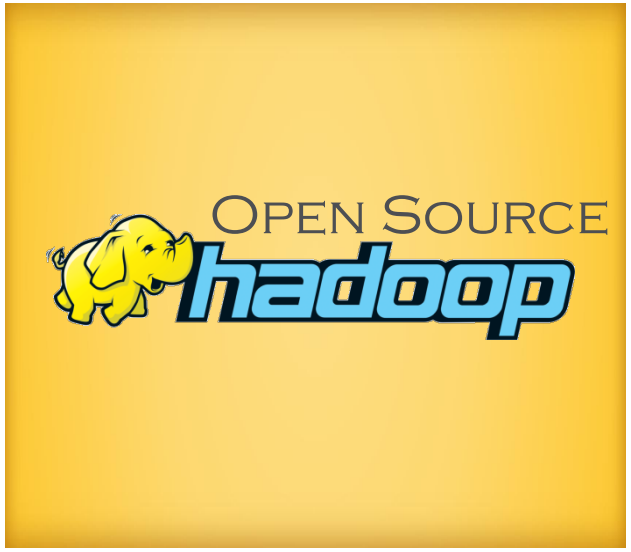
CISCO™

rubicon PROJECT

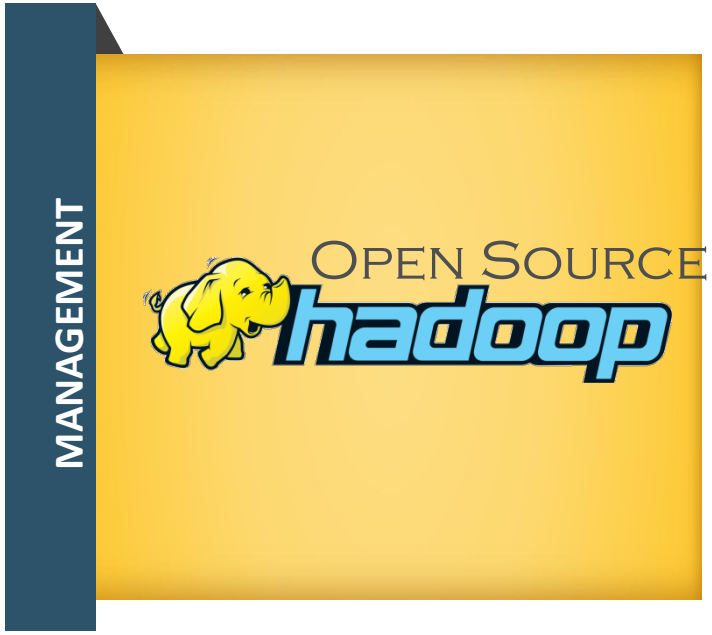
MAPR®

Hadoop Distributions

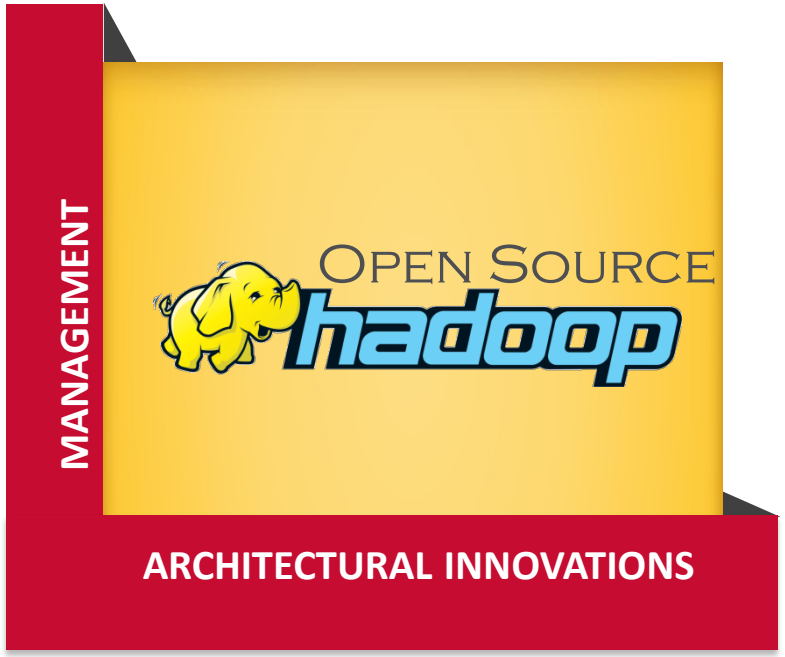
Distribution A



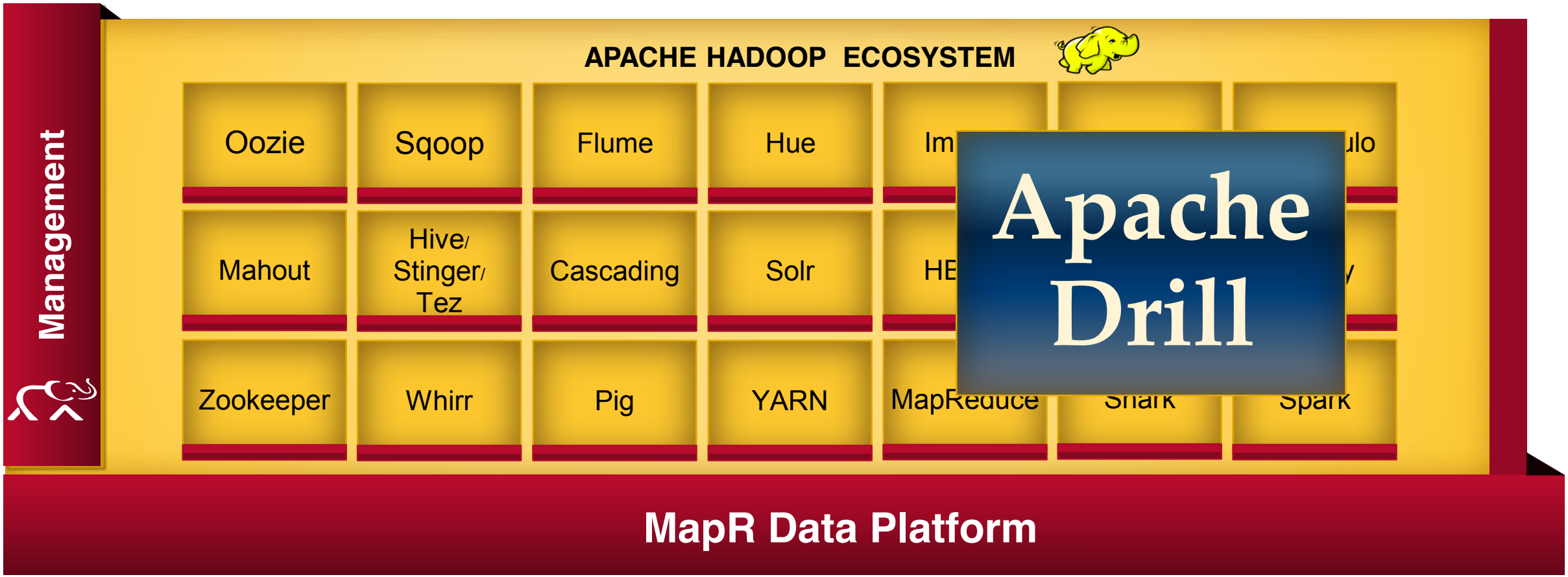
Distribution C



MAPR



Enterprise Hadoop from MapR



MAPR®

Hadoop an augmentation for EDW—Why?

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

CSO/CMO

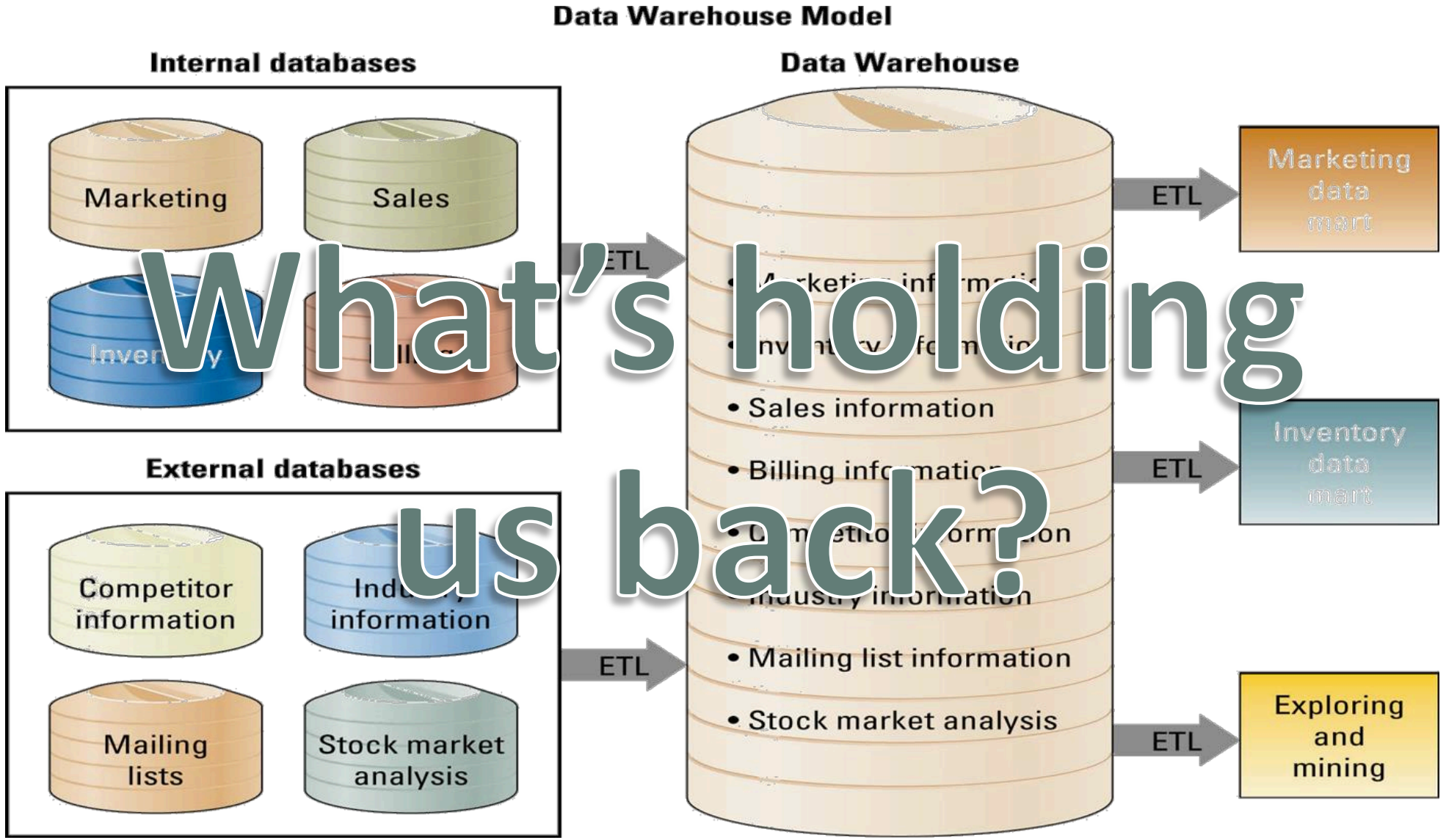
Regulation: "You need to store more stuff, and find it fast"

CRO

Reduce OpEx on IT spend by identifying anomalies faster

CIO

MAPR®



What's holding us back?

Consolidating multiple schemas is very hard.

Why? Since schema-on-write, retrieval is pre-determined.

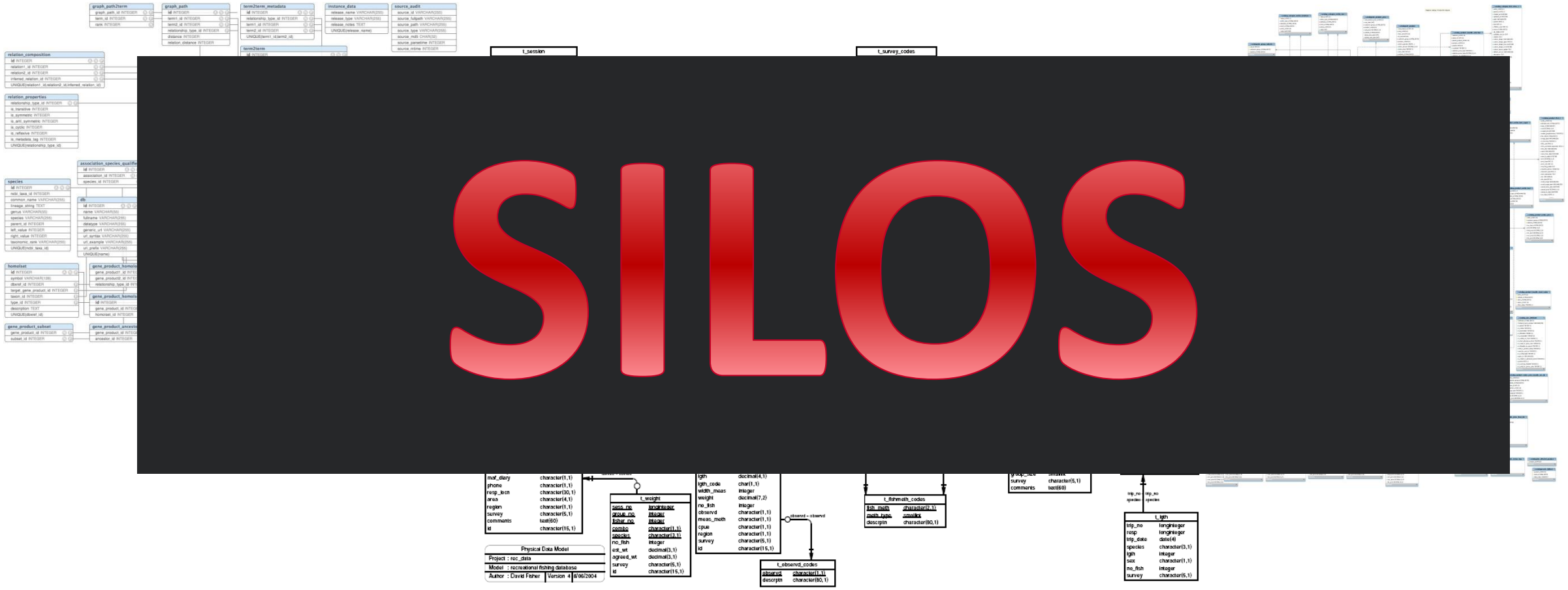
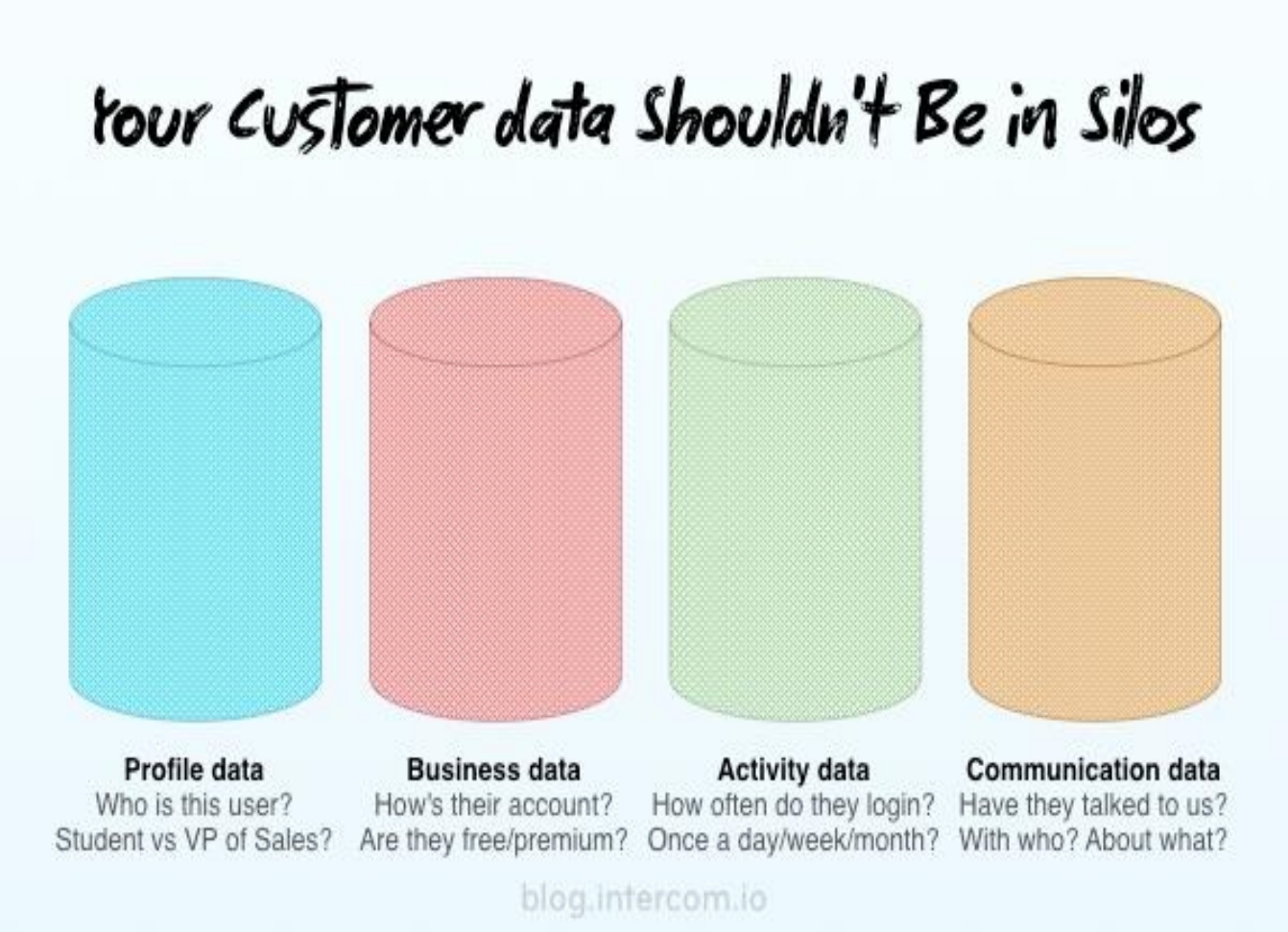


Figure 1: Entity Relationship Diagram (ERD) for the rec_data database.

Silos make analysis very difficult



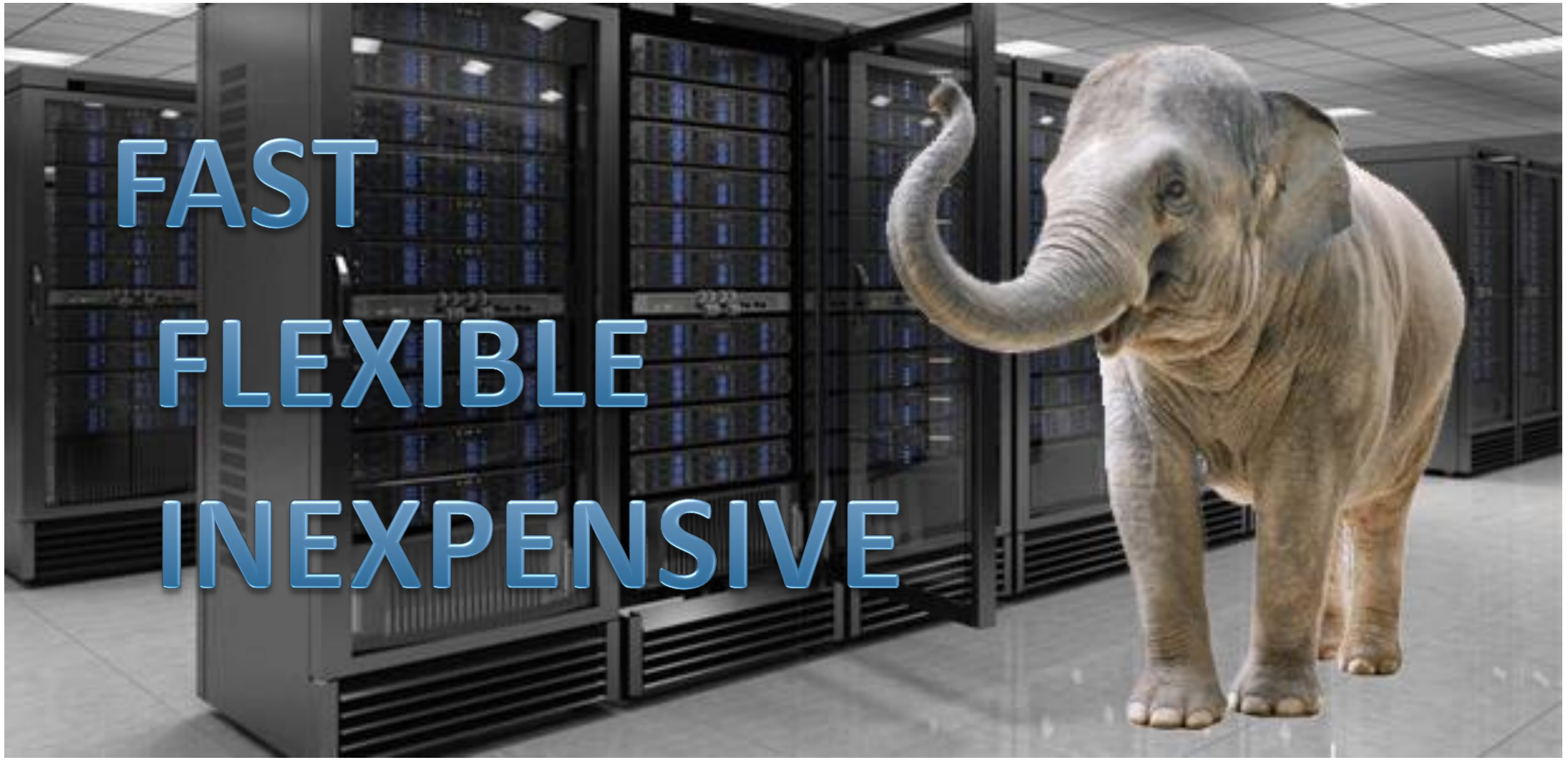
How do I identify a unique {customer, trade} across data sets?

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*

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, …

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

MAPR

SQL is here to stay

Hadoop is here to stay

MAPR®

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

MAPR

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?
- 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

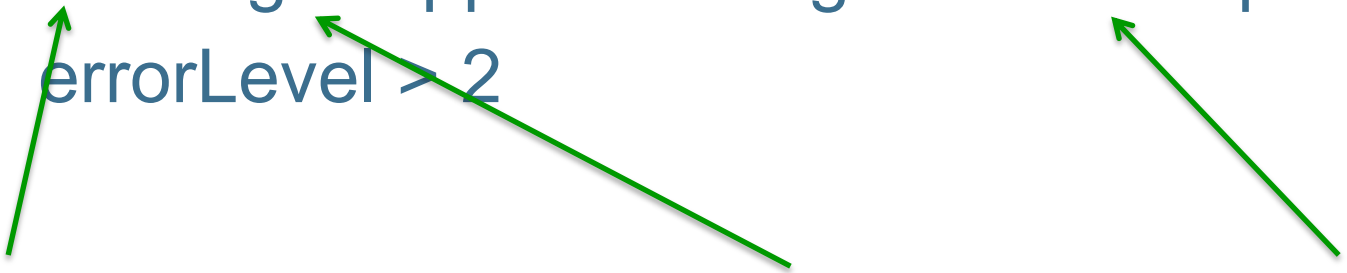
MAPR

A Quick Tour through Apache Drill

MAPR®

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,  PROF.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;
```

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

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

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());
    }
}
```

MAPR

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

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` );
```

MAPR

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;
```

MAPR

- 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

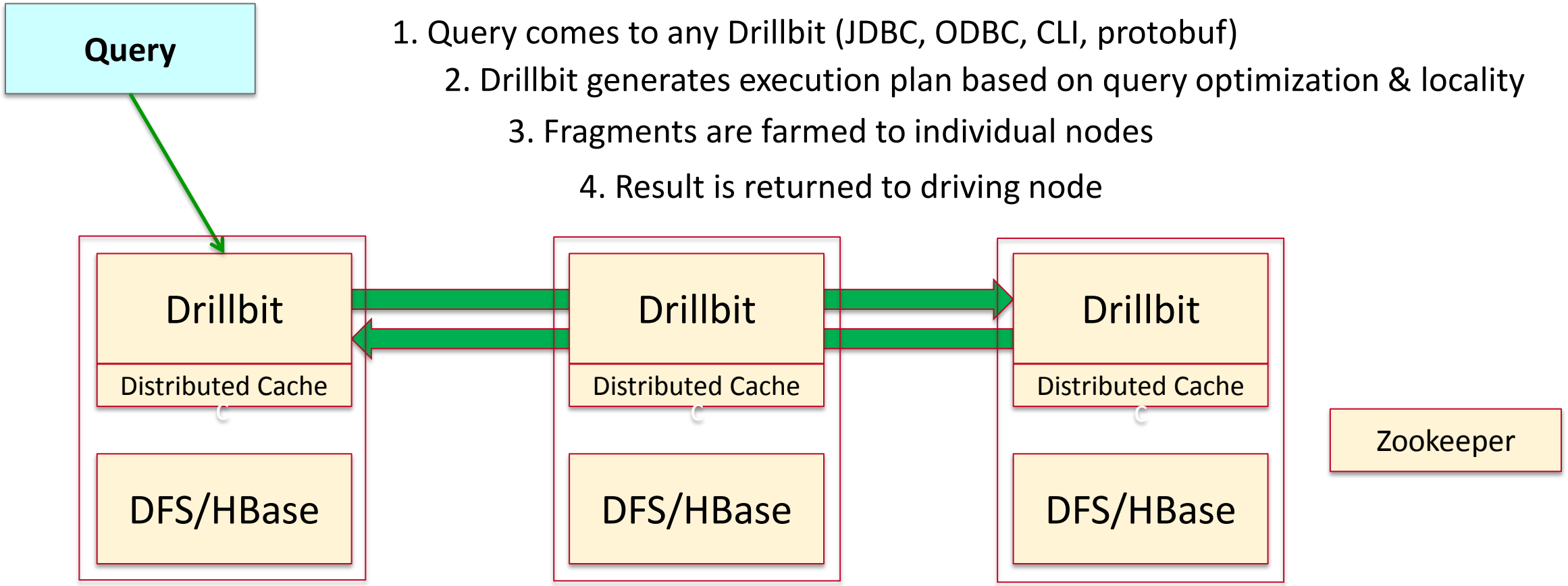
De-centralized metadata

// 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

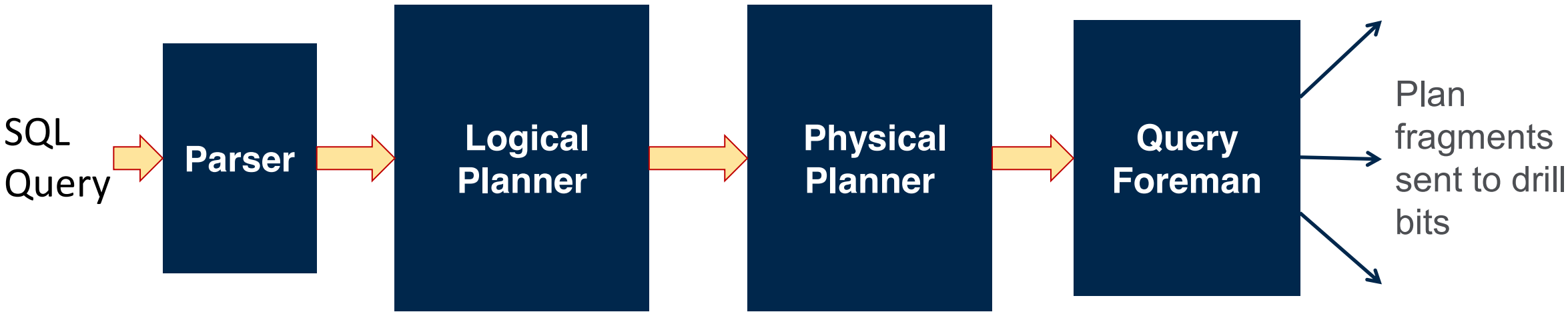
```
select   c.customerName,  hb.tweets.count
from     hive.CustomersDB.`Customers` c
join     hbase.user.`SocialData` hb
on       c.customerId = convert_from( hb.rowkey, UTF-8);
```

Un  vers

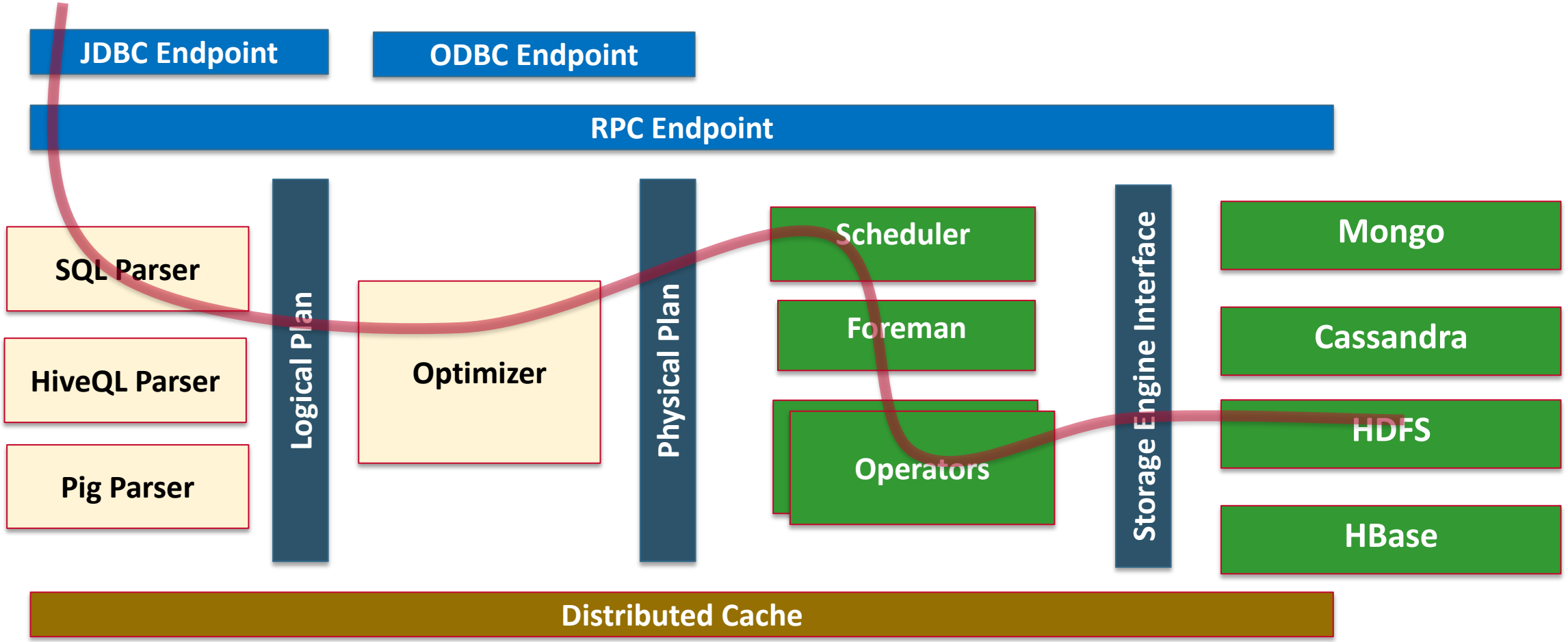
Basic Process



Stages of Query Planning



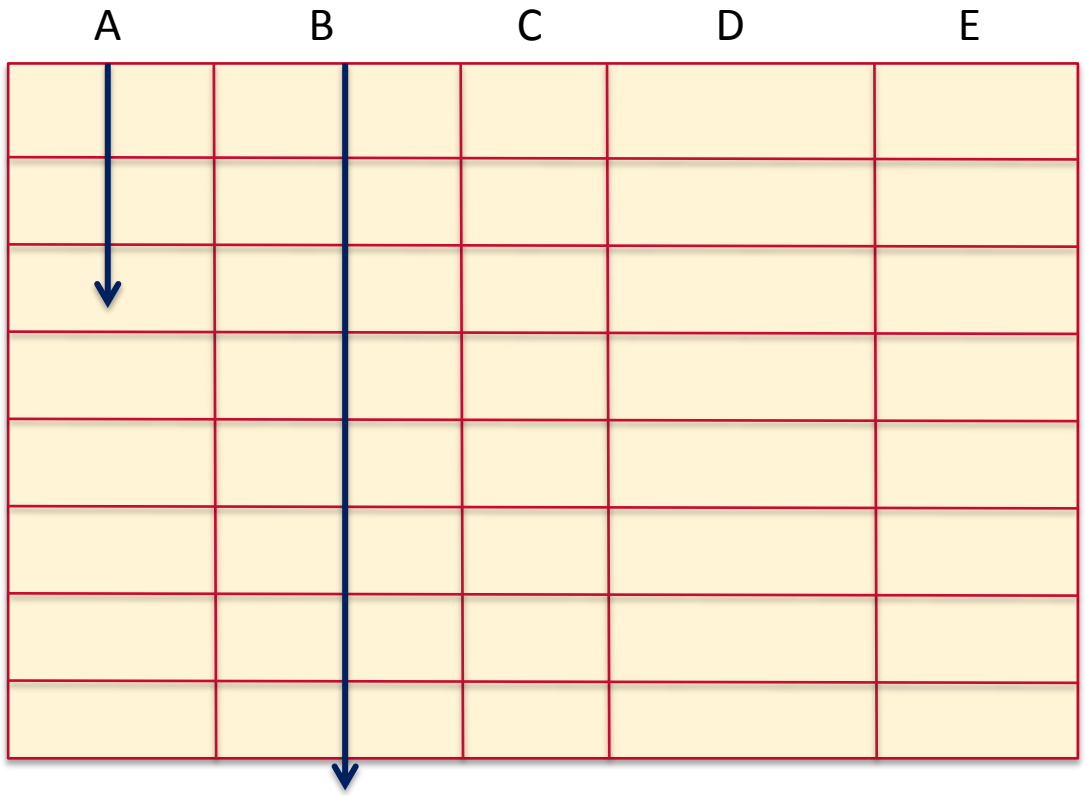
Query Execution

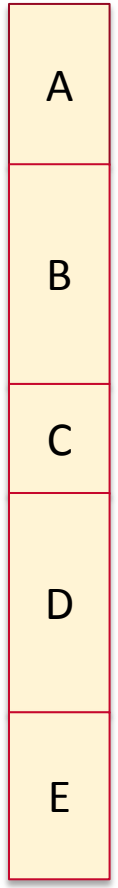


A Query engine that is…

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

Columnar representation

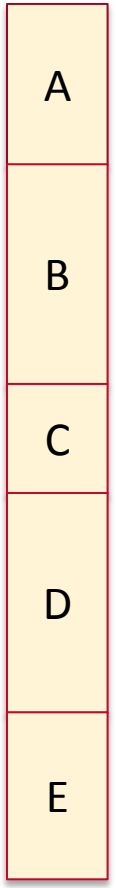




On disk

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



On disk

MAPR

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

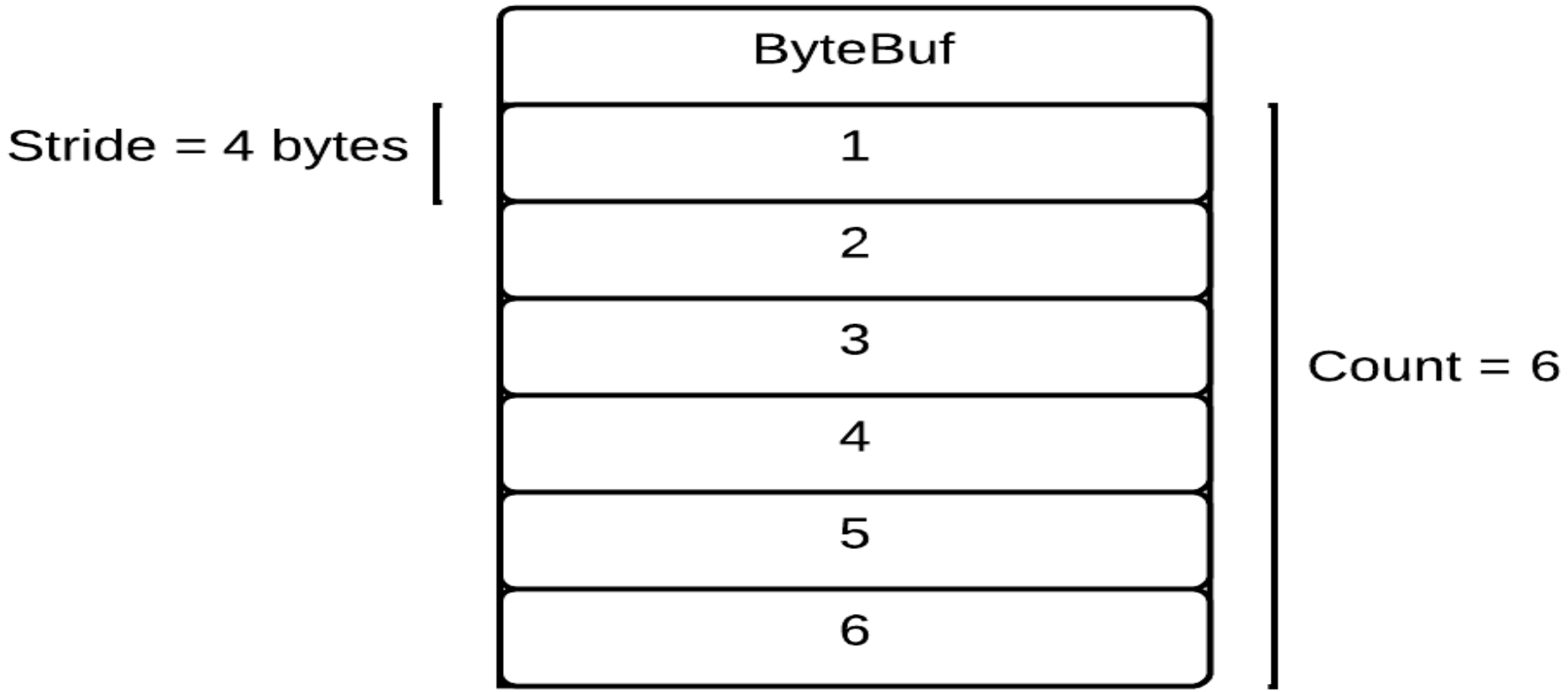
Drill 4-value semantics

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

Batches of Values

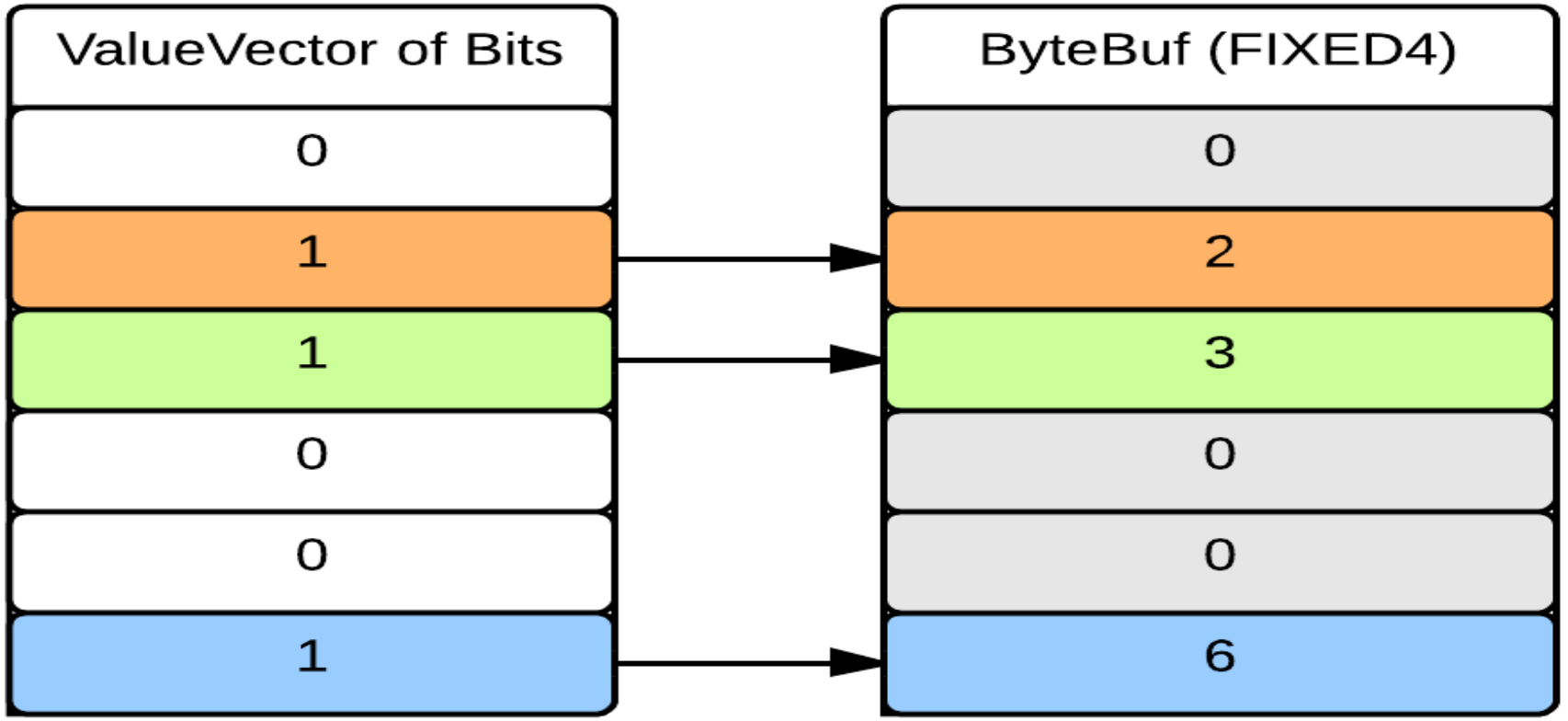
- *Value vectors*
  - List of values, with same schema
  - With the 4-value semantics for each value
- Shipped around in batches
  - max 256k bytes in a batch
  - max 64K rows in a batch
- RPC designed for multiple replies to a request

Fixed Value Vectors

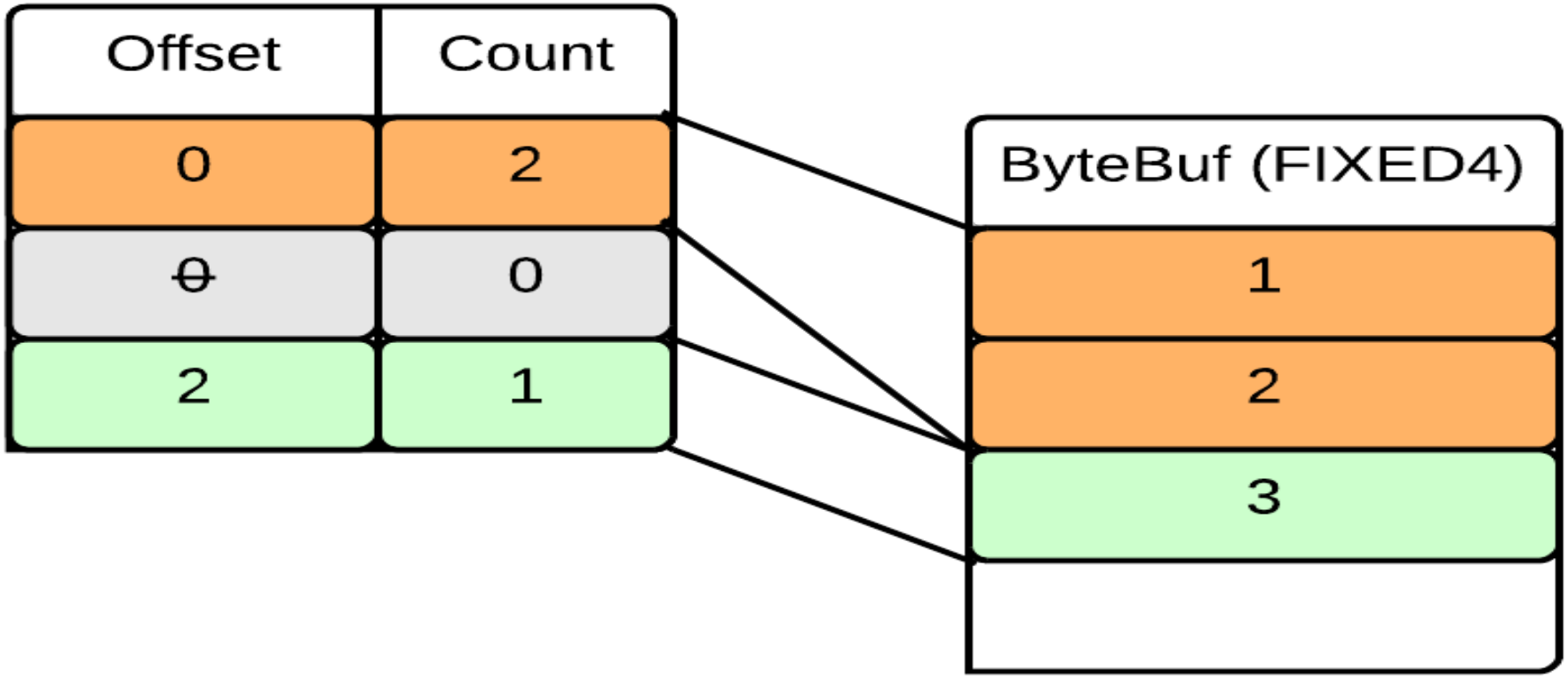


MAPR

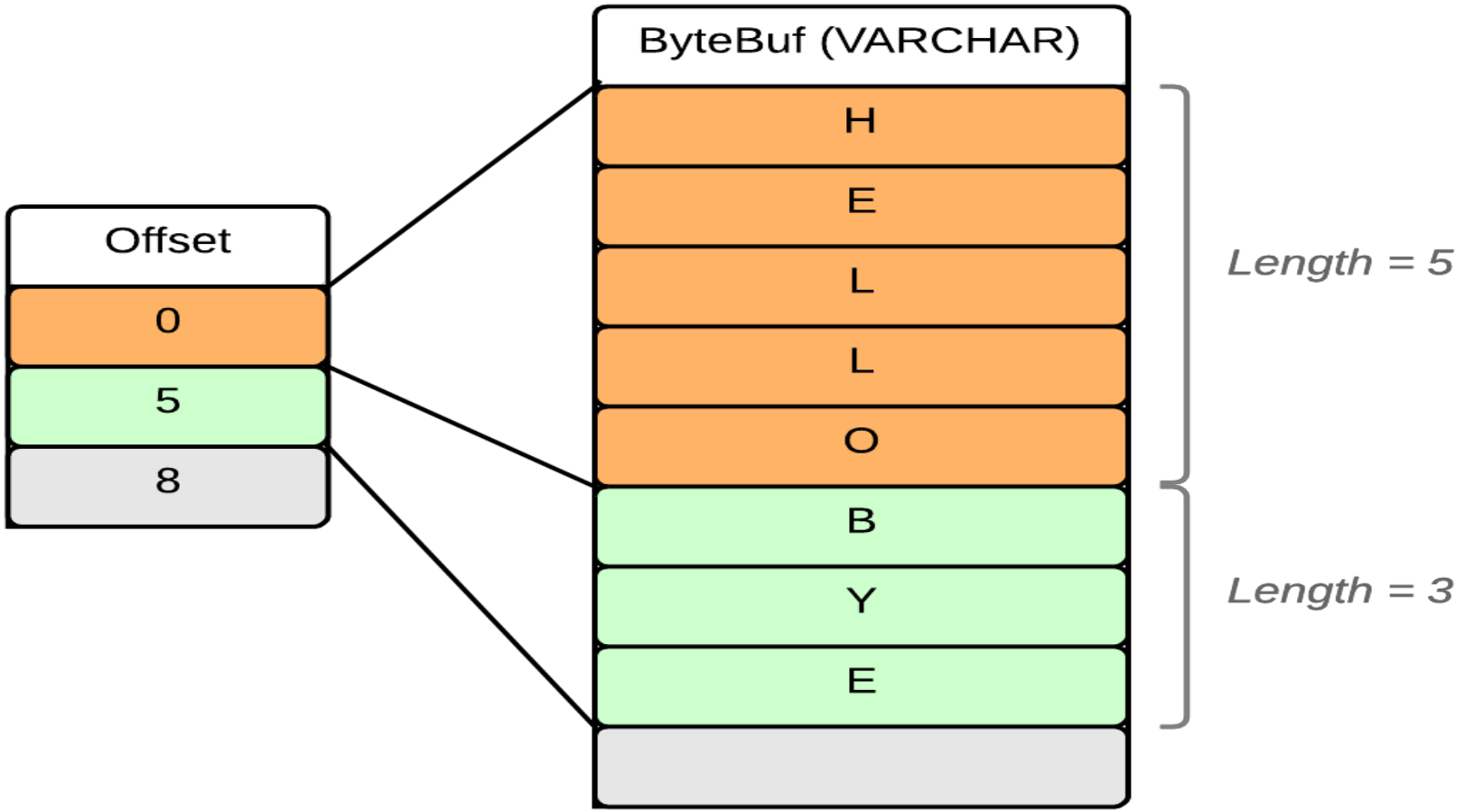
Nullable Values



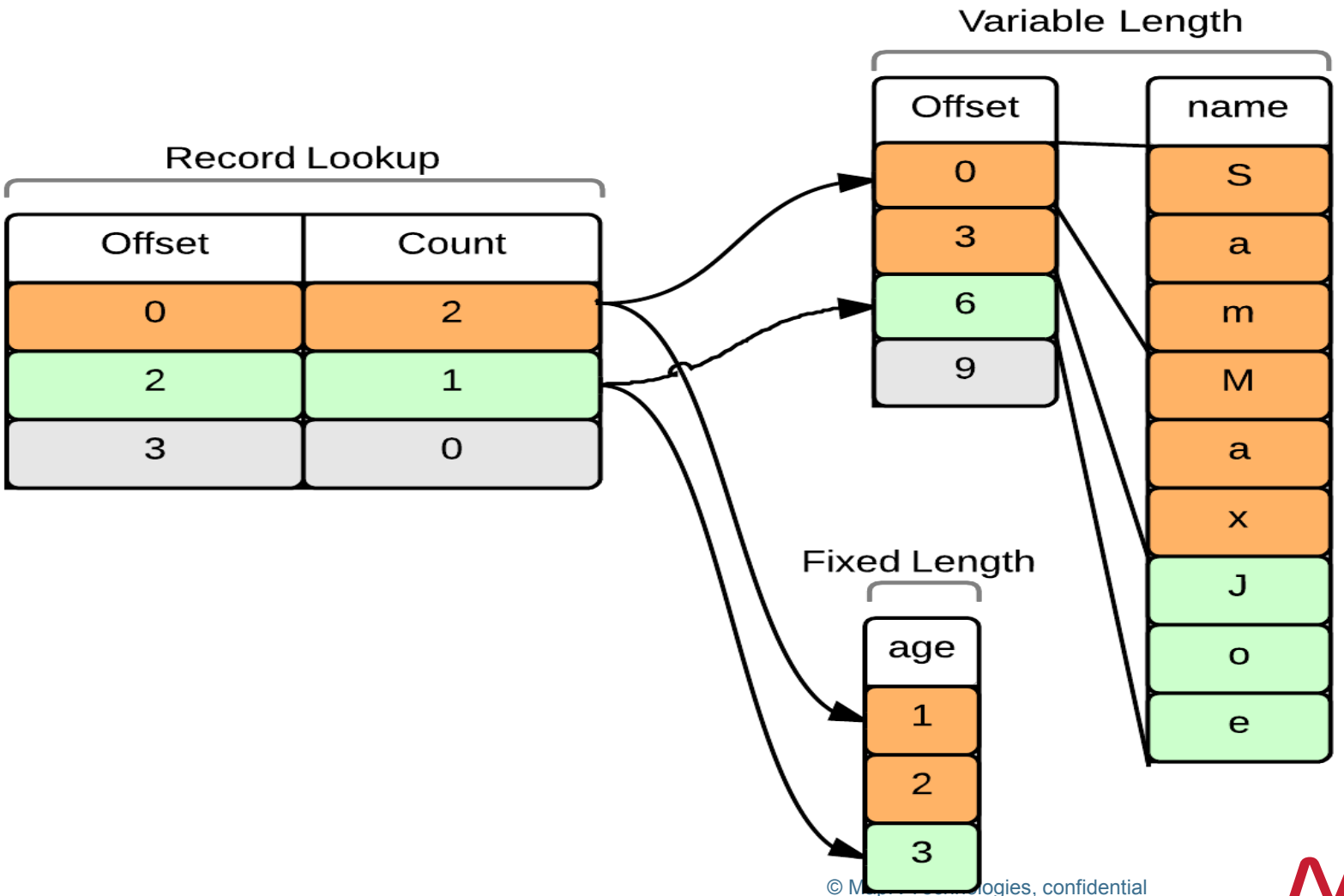
Repeated Values



Variable Width



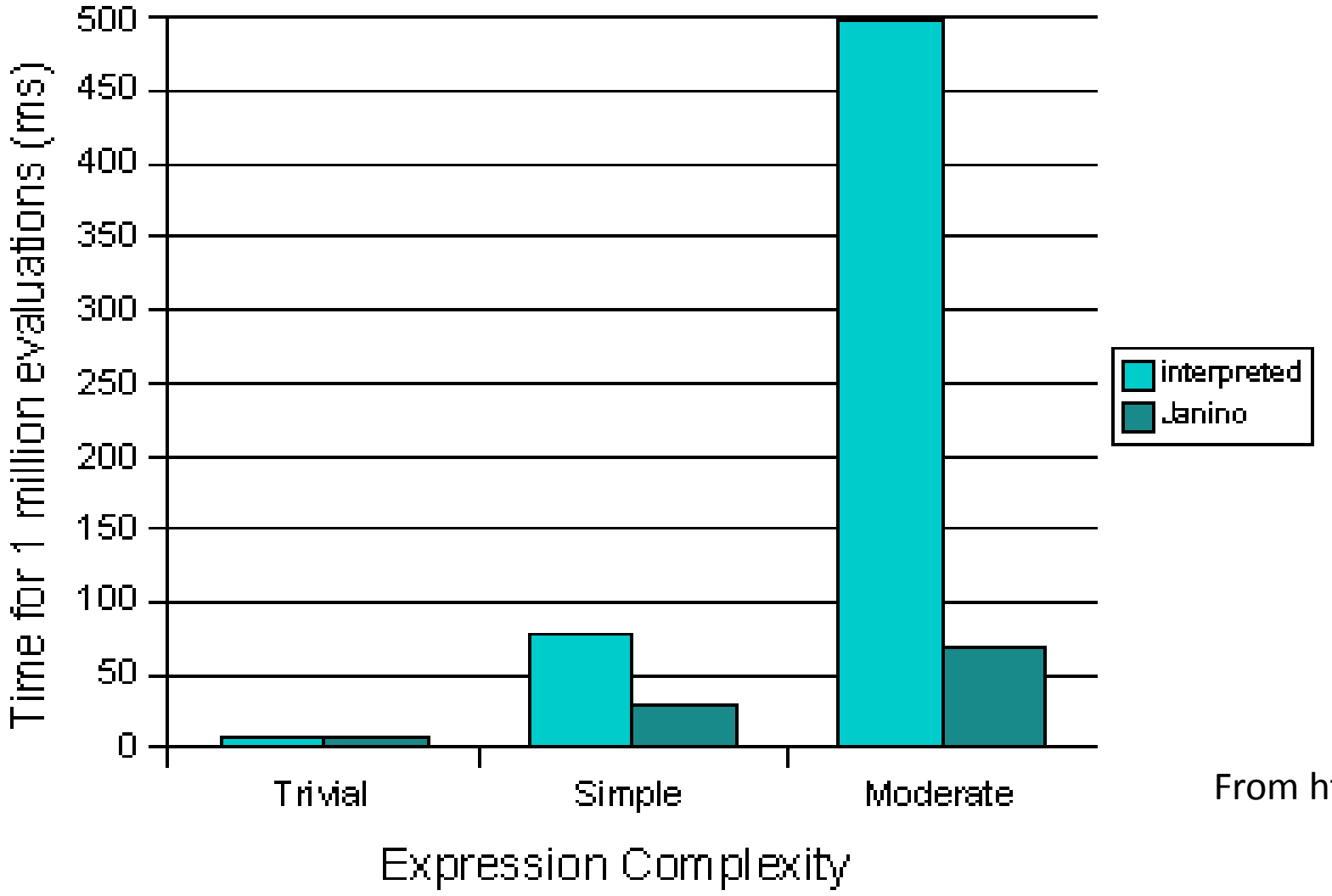
Repeated Map



Vectorization

- Drill operates on more than one record at a time
  - Word-sized manipulations
  - SIMD 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

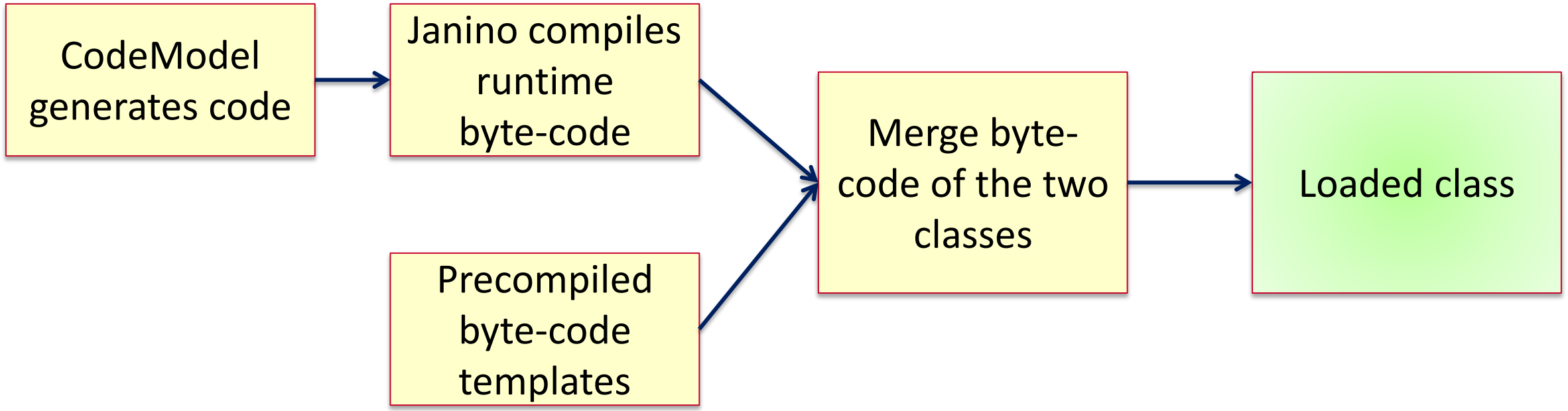


- JIT is smart, but more gains with runtime compilation
- Janino: Java-based Java compiler

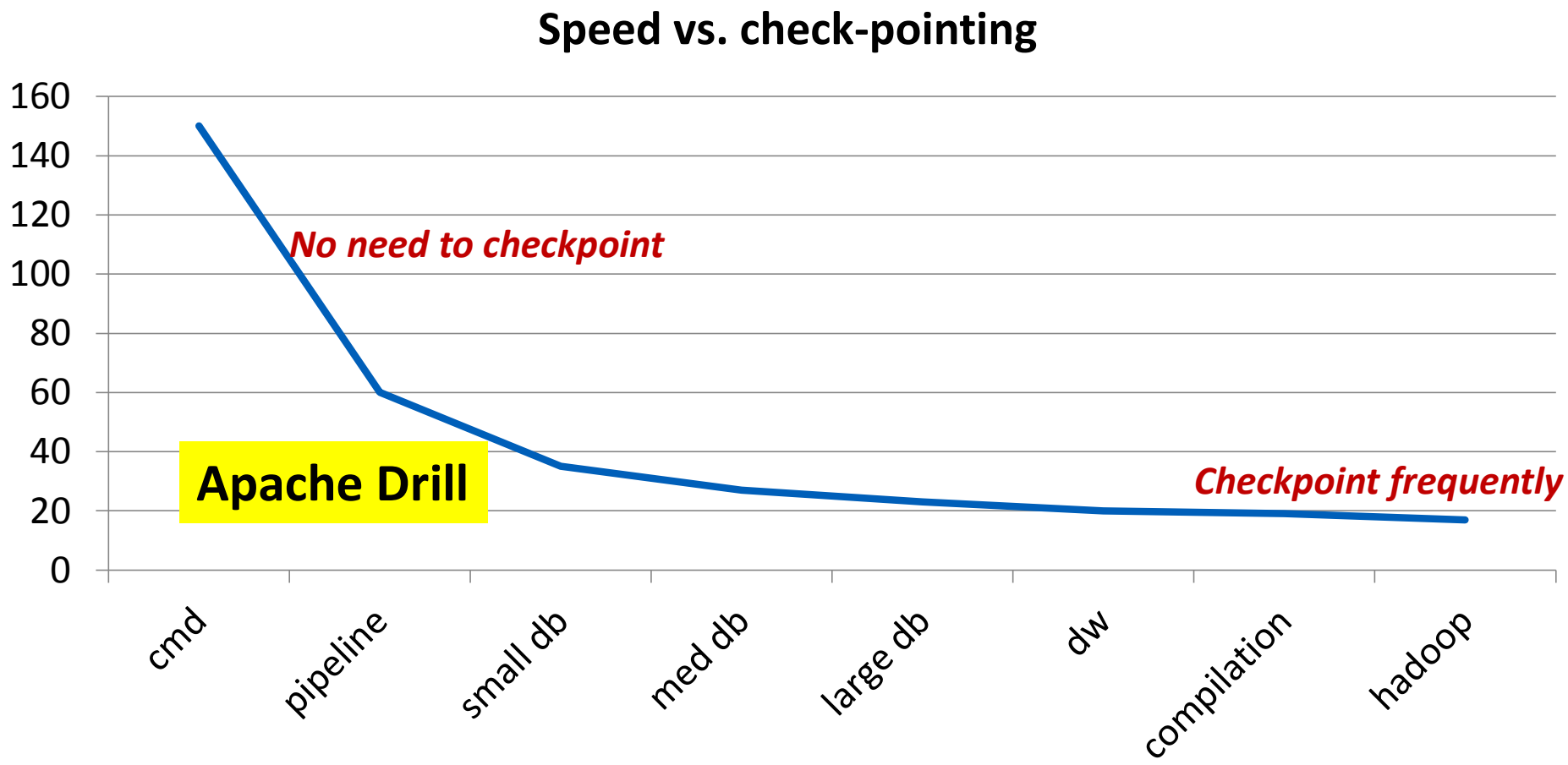
From http://bit.ly/16Xk32x

MAPR

Drill compiler



Optimistic



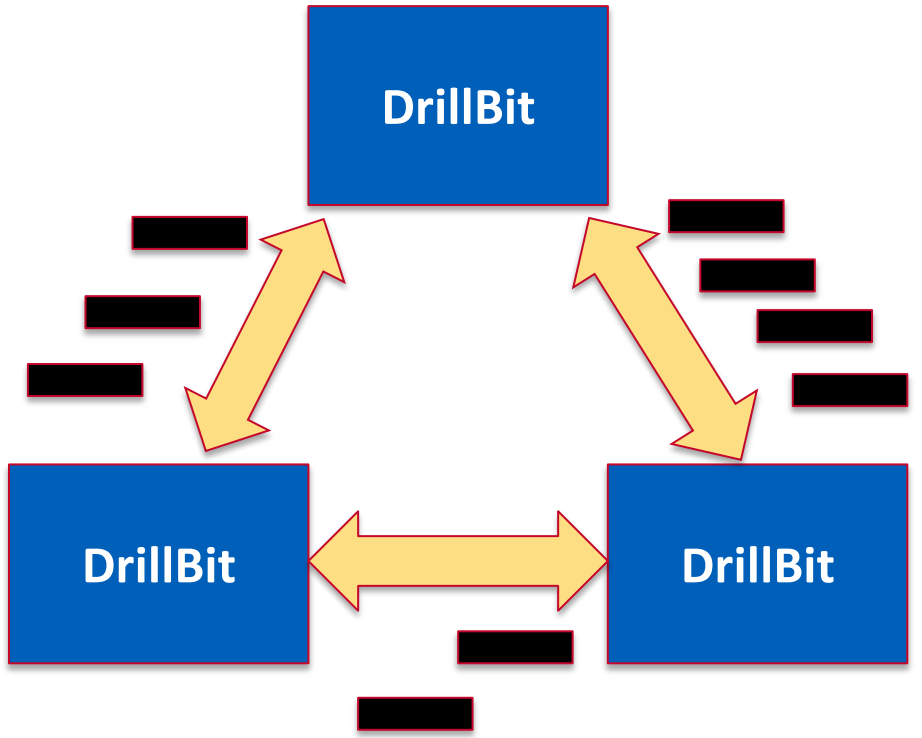
MAPR®

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

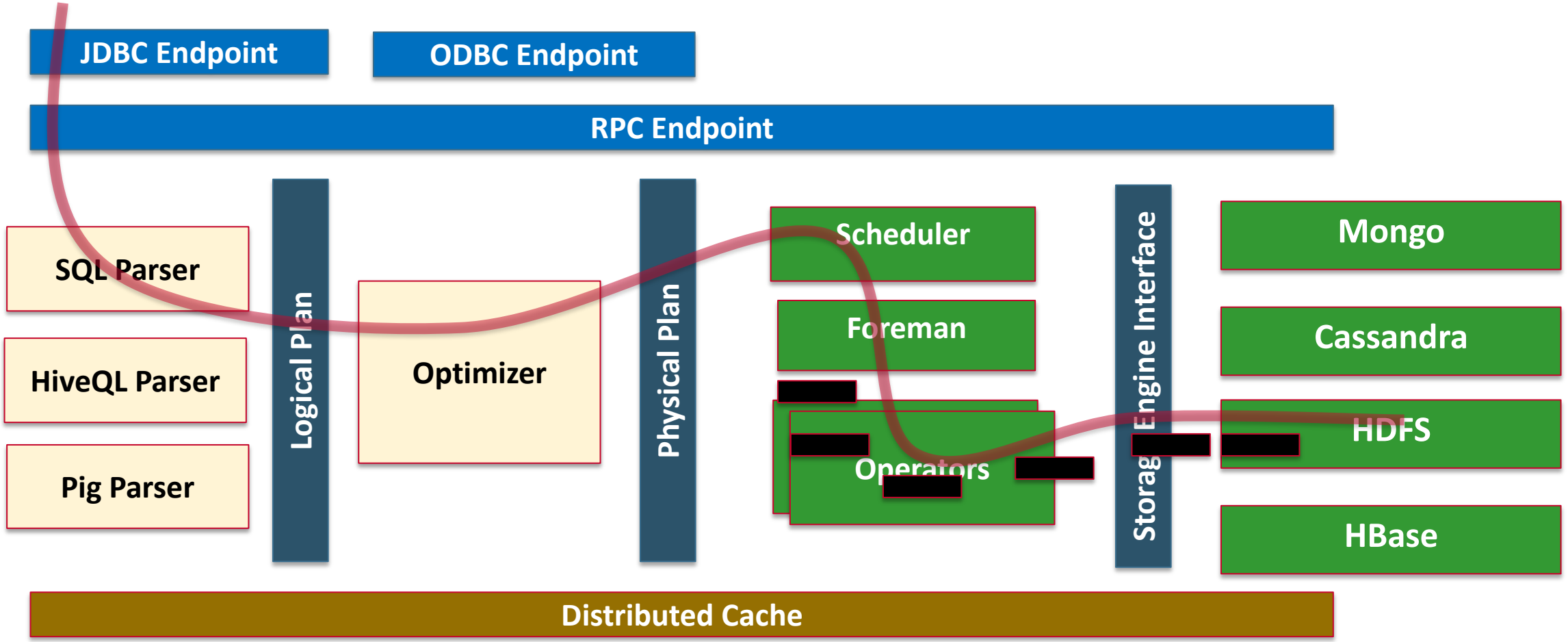
Pipelining

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



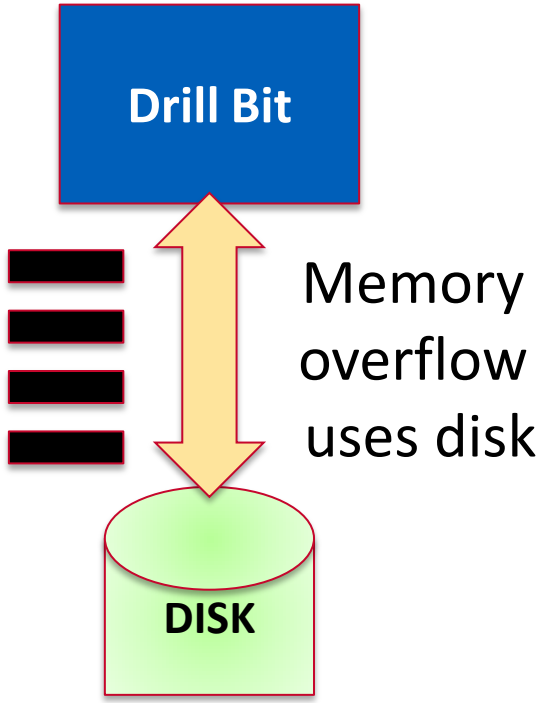
MAPR

Pipelining Record Batches



Pipelining

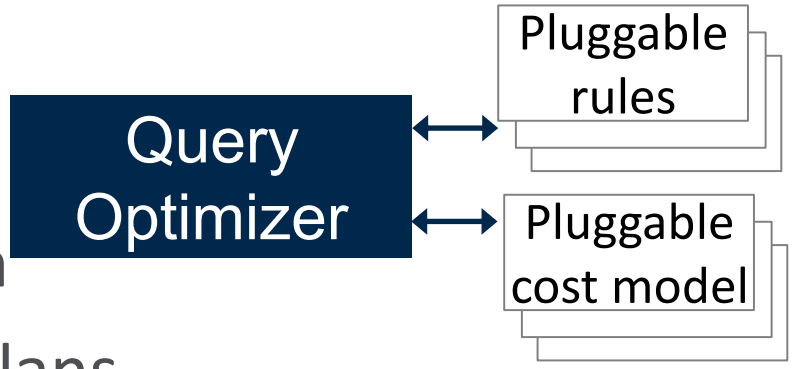
- Random access: sort without copy or restructuring
- Avoids serialization/deserialization
- Off-heap (no GC woes when lots of memory)
- Full specification + off-heap + batch
  - Enables C/C++ operators *(fast!)*
- Read/write to disk
  - when data larger than memory



MAPR

Cost-based Optimization

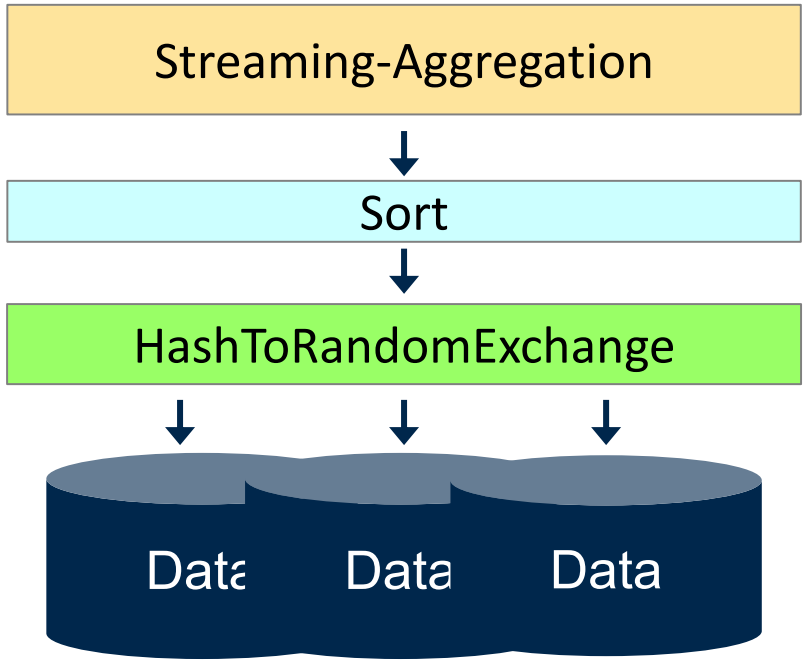
- Using Optiq, an extensible framework
  - Pluggable rules, and cost model
- Rules for distributed plan generation
  - Insert ***Exchange*** operator into physical plan
  - Optiq enhanced to explore parallel query plans
- Pluggable cost model
  - CPU, IO, memory, network cost (**data locality**)
  -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



MAPR

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



FLEXIBLE SCHEMA MANAGEMENT

Analyze data, self-described or central metadata

FRICTIONLESS ANALYTICS ON NESTED DATA

Analyze semi structured & nested data

PLUG AND PLAY WITH EXISTING

Reuse investments in SQL/BI tools and Apache Hive

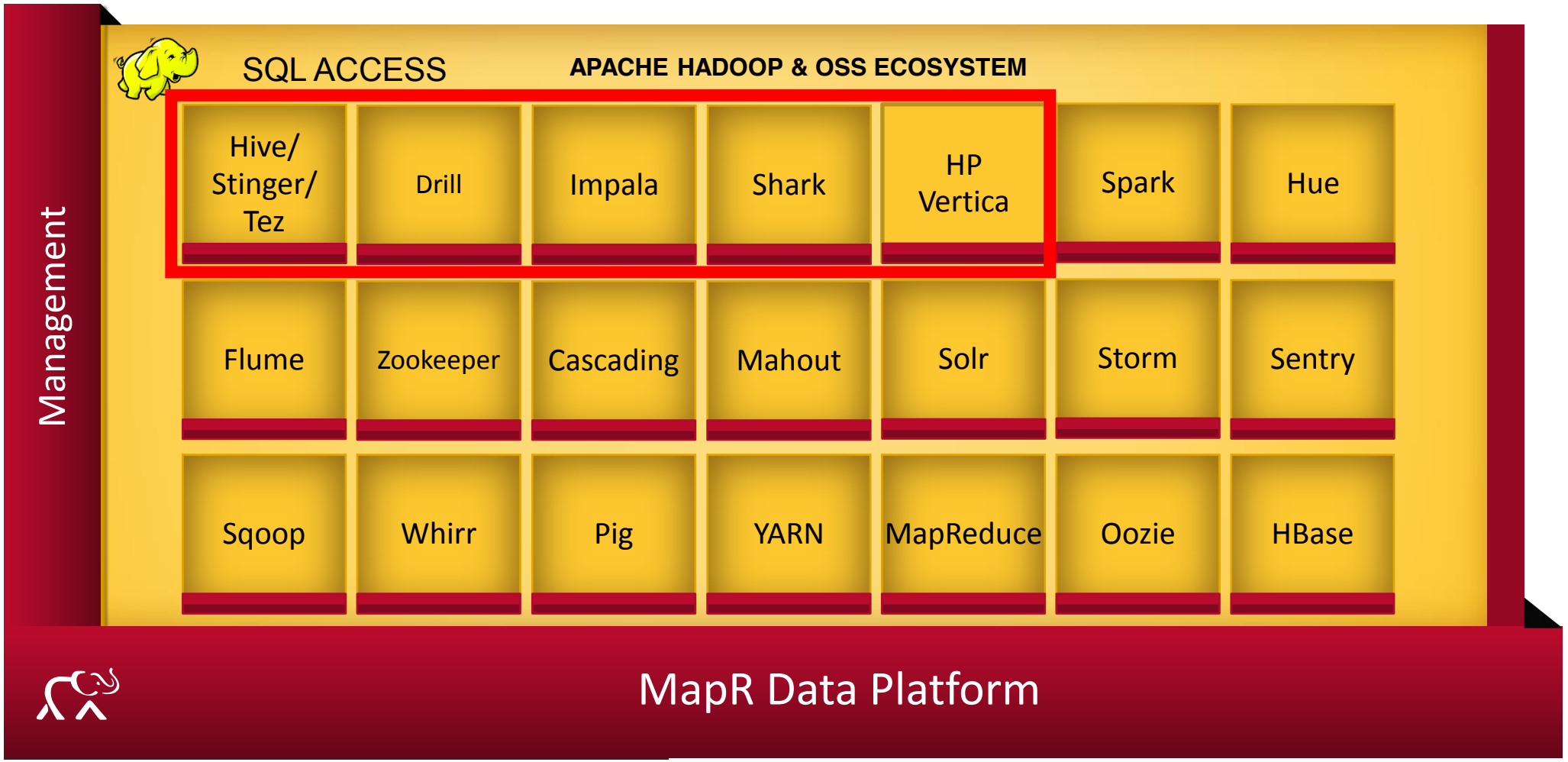
… and with an architecture built ground up for Low Latency queries at Scale

MAPR®

Interactive SQL-on-Hadoop options

	Drill 1.0	Hive 0.13 w/ Tez	Impala 1.x	Shark 0.9
Latency	Low	Medium	Low	Medium
Files	Yes (all Hive file formats, plus JSON, Text, …)	Yes (all Hive file formats)	Yes (Parquet, Sequence, …)	Yes (all Hive file formats)
HBase/M7	Yes	Yes, perf issues	Yes, with issues	Yes, perf issues
Schema	Hive or schema-less	Hive	Hive	Hive
SQL support	ANSI SQL	HiveQL	HiveQL (subset)	HiveQL
Client support	ODBC/JDBC	ODBC/JDBC	ODBC/JDBC	ODBC/JDBC
Hive compat	High	High	Low	High
Large datasets	Yes	Yes	Limited	Limited
Nested data	Yes	Limited	No	Limited
Concurrency	High	Limited	Medium	Limited

SQL-on-Hadoop



Drill at MapR

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

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
  - 2 hackathons, with 40+ participants
- Encourage you to join, learn, contribute and have fun …

Apache Drill Resources

- 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

Thank you!

Did I mention we are hiring…

M. C. Srivas
srivas@mapr.com

MAPR®