MAPR TECHNOLOGIES

Apache Drill Implementation Deep Dive

Ted Dunning & Michael Hausenblas

Berlin Buzzwords 2013-06-03

Which workloads do **you** encounter in **your** environment?

APACHE DRILL

Batch processing

… for recurring tasks such as large-scale data mining, ETL offloading/data-warehousing ␣ *for the batch layer in Lambda architecture*

APACHE DRILL

MAPR TECHNOLOGIES

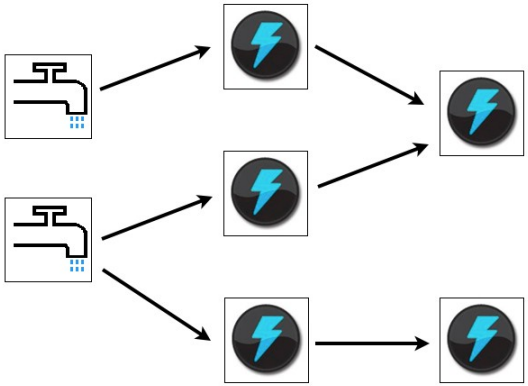
OLTP

… user-facing eCommerce transactions, real-time messaging at scale (FB), time-series processing, etc. ␣ *for the serving layer in Lambda architecture*

Stream processing

Storm

Distributed and fault-tolerant realtime computation



S4 *distributed stream computing platform*

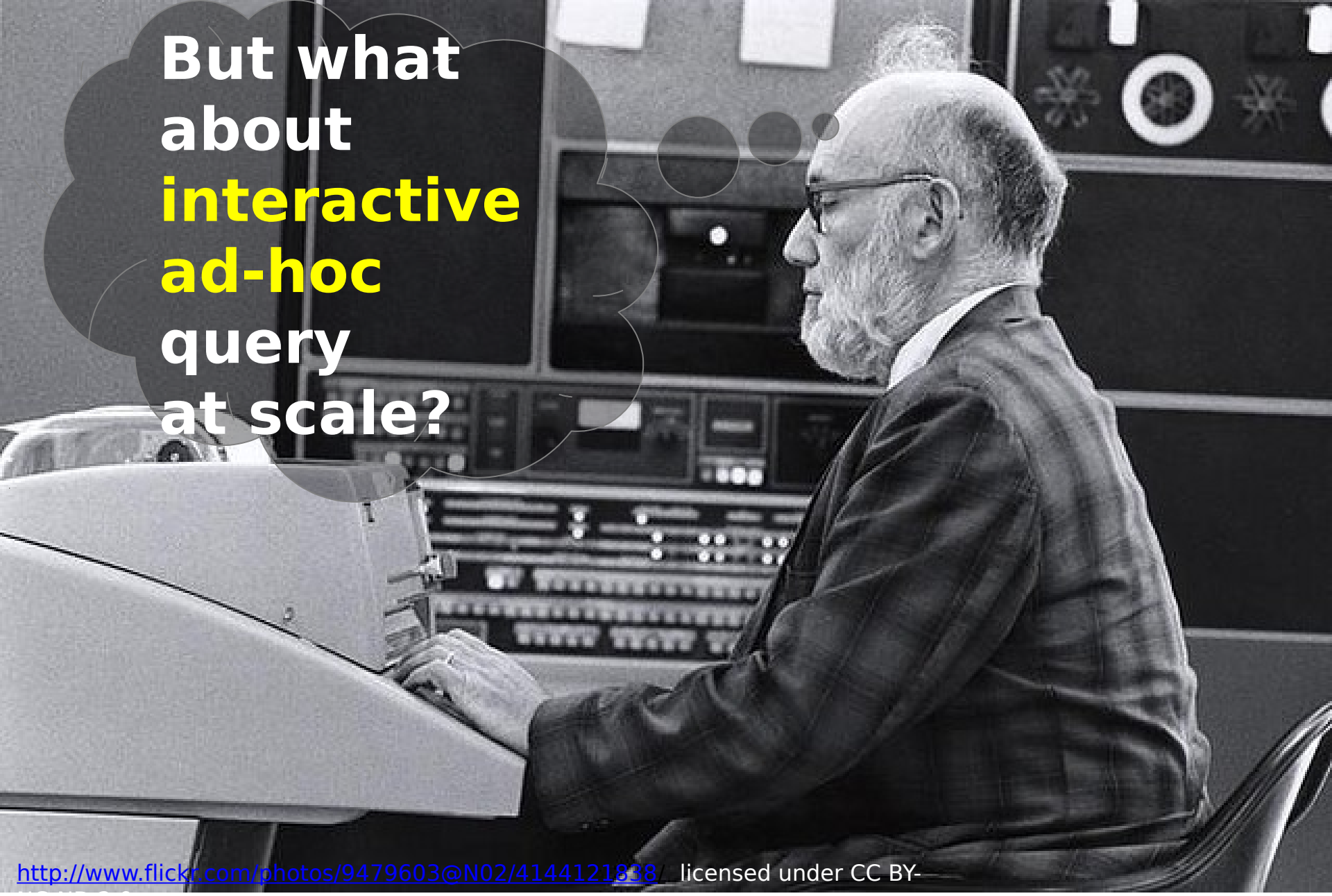
… in order to handle stream sources such as social media feeds or sensor data (mobile phones, RFID, weather stations, etc.) ⇨ *for the speed layer in Lambda architecture*

APACHE DRILL

MAPR TECHNOLOGIES

Search/Information Retrieval

… retrieval of items from unstructured documents (plain text, etc.), semi-structured data formats (JSON, etc.), as well as data stores (MongoDB, CouchDB, etc.)



Interactive Query (?)

Use Case: Logistics

- Supplier tracking and performance
- Queries
  - Shipments from supplier ‘ACM’ in last 24h
  - Shipments in region

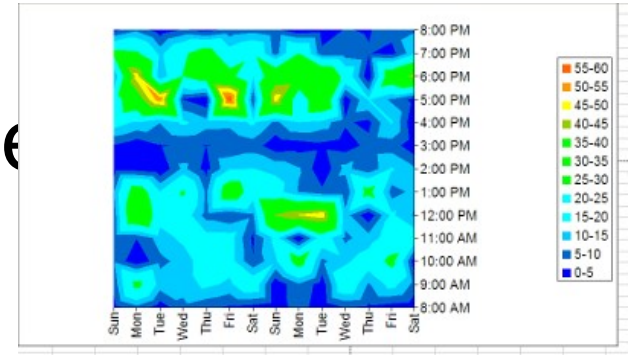
SUPPLIER_ID	NAME	REGION
ACM	ACME Corp	US
GAL	GotALot Inc	US
BAP	Bits and Pieces Ltd	Europe
ZUP	Zu Pli	Asia

```
{
 "shipment": 100123,
 "supplier": "ACM",
 “timestamp": "2013-02-01",
 "description": ”first delivery today”
},
{
 "shipment": 100124,
 "supplier": "BAP",
 "timestamp": "2013-02-02",
 "description": "hope you enjoy it”
}
…
```

APACHE DRILL

Use Case: Crime Detection

- Online purchases
- Fraud, bilking, etc.
- Batch-generated overvie
- Modes
  - Explorative
  - Alerts



APACHE DRILL

MAPR TECHNOLOGIES

Requirements

- Support for different data sources
- Support for different query interfaces
- Low-latency/real-time
- Ad-hoc queries
- Scalable, reliable

d now for something completely different ...

DRILL

MAPR TECHNOLOGIES

Google’s Dremel

> “Dremel is a scalable, interactive ad-hoc query system for analysis of read-only nested data. By combining multi-level execution trees and columnar data layout, it is capable of running aggregation queries over trillion-row tables in seconds. The system scales to thousands of CPUs and petabytes of data, and has thousands of users at Google.
> …”

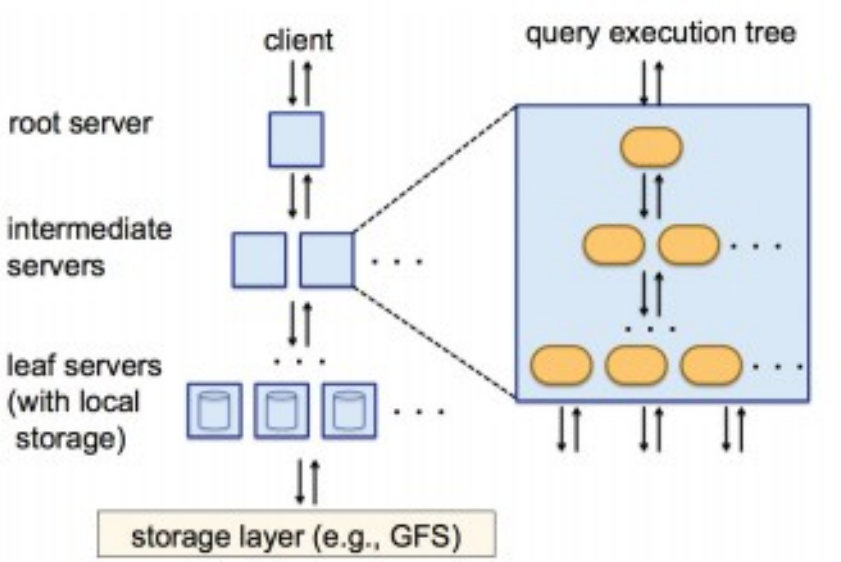
http://research.google.com/pubs/pub36632.html

Sergey Melnik, Andrey Gubarev, Jing Jing Long, Geoffrey Romer, Shiva Shivakumar, Matt Tolton, Theo Vassilakis, Proc. of the 36th Int'l Conf on Very Large Data Bases (2010), pp. 330-339

APACHE DRILL

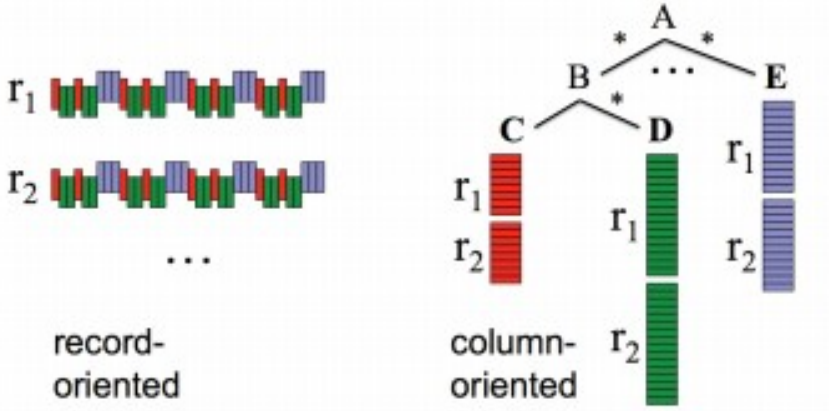
MAPR TECHNOLOGIES

Google's Dremel



multi-level execution trees

columnar data layout



APACHE DRILL

MAPR TECHNOLOGIES

Google's Dremel

```
DocId: 10                    r1
Links
  Forward: 20
  Forward: 40
  Forward: 60
Name
  Language
    Code: 'en-us'
    Country: 'us'
  Language
    Code: 'en'
  Url: 'http://A'
Name
  Url: 'http://B'
Name
  Language
    Code: 'en-gb'
    Country: 'gb'
```

```
message Document {
  required int64 DocId;
  optional group Links {
    repeated int64 Backward;
    repeated int64 Forward; }
  repeated group Name {
    repeated group Language {
      required string Code;
      optional string Country; }
    optional string Url; }}
```

```
DocId: 20                    r2
Links
  Backward: 10
  Backward: 30
  Forward:  80
Name
  Url: 'http://C'
```

DocId

value	r	d
10	0	0
20	0	0

Name.Url

value	r	d
http://A	0	2
http://B	1	2
NULL	1	1
http://C	0	2

Links.Forward

value	r	d
20	0	2
40	1	2
60	1	2
80	0	2

Links.Backward

value	r	d
NULL	0	1
10	0	2
30	1	2

Name.Language.Code

value	r	d
en-us	0	2
en	2	2
NULL	1	1
en-gb	1	2
NULL	0	1

Name.Language.Country

value	r	d
us	0	3
NULL	2	2
NULL	1	1
gb	1	3
NULL	0	1

nested data + schema

column-striped representation

map nested data to tables

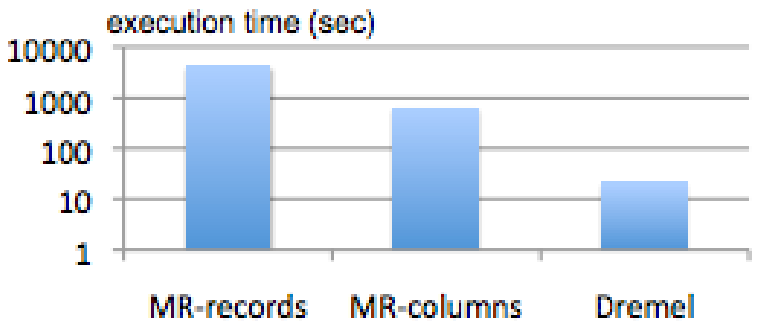
APACHE DRILL

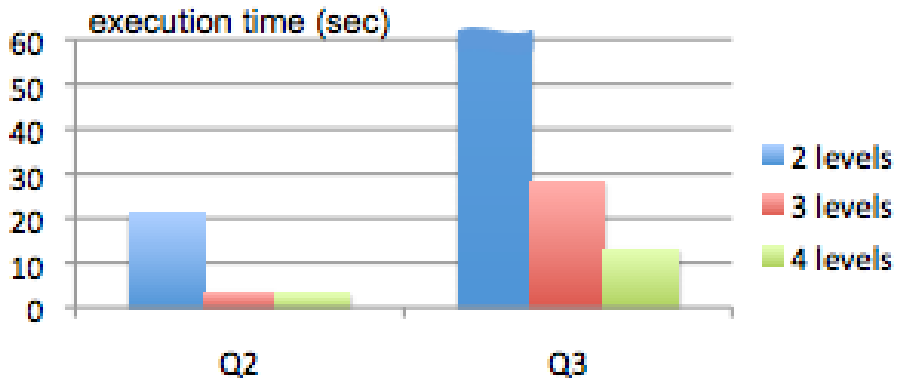
MAPR TECHNOLOGIES

Google's Dremel

Table name	Number of records	Size (unrepl., compressed)	Number of fields	Data center	Repl. factor
T1	85 billion	87 TB	270	A	3×
T2	24 billion	13 TB	530	A	3×
T3	4 billion	70 TB	1200	A	3×
T4	1+ trillion	105 TB	50	B	3×
T5	1+ trillion	20 TB	30	B	2×

experiments:
datasets & query performance





APACHE DRILL

MAPR TECHNOLOGIES

Back to Apache Drill ...

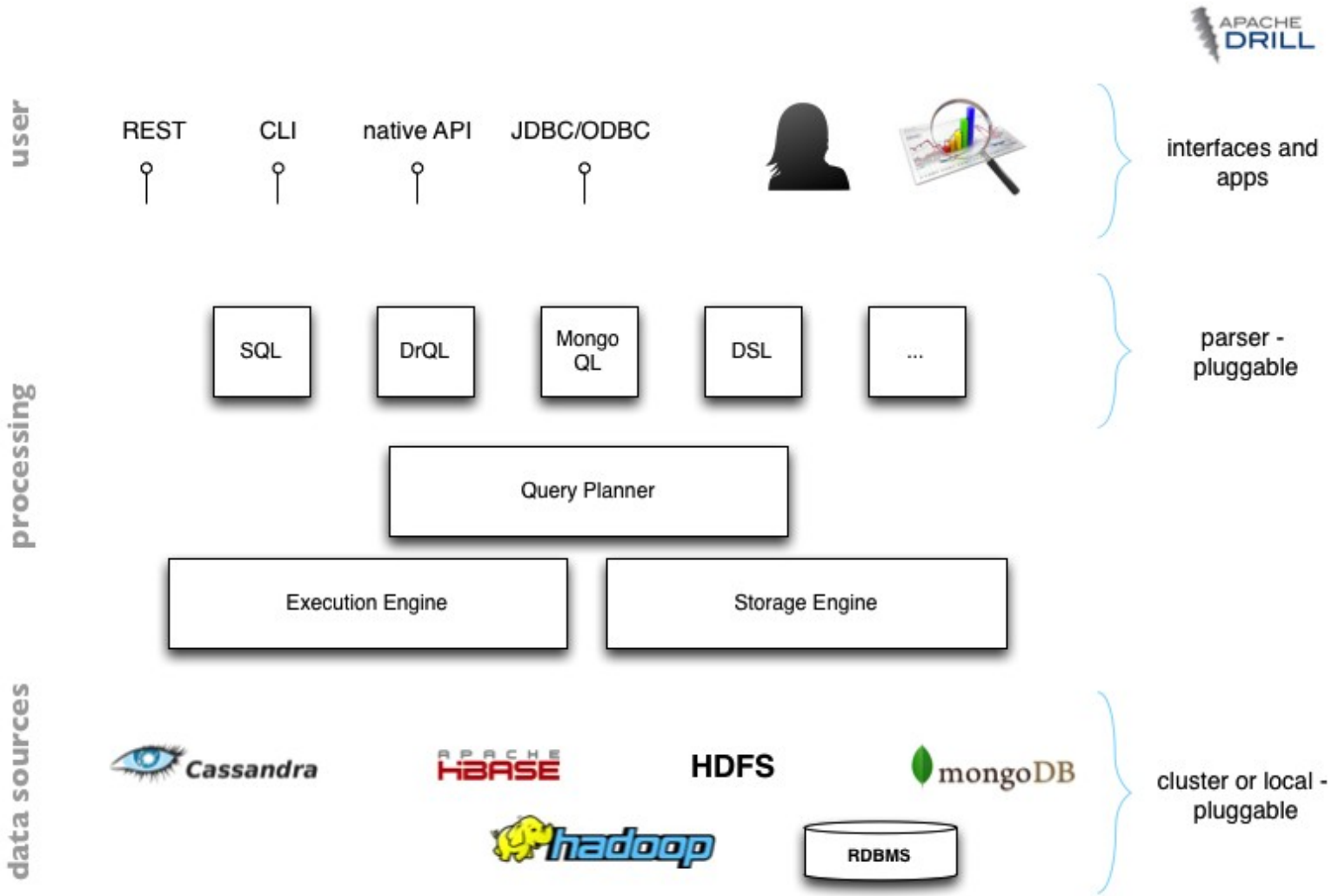
Apache Drill–key facts

- Inspired by Google's **Dremel**
- Standard **SQL 2003** support
- Plug-able **data sources**
- **Nested data** is a first-class citizen
- **Schema** is **optional**
- **Community** driven, **open**, 100's involved

APACHE DRILL

MAPR TECHNOLOGIES

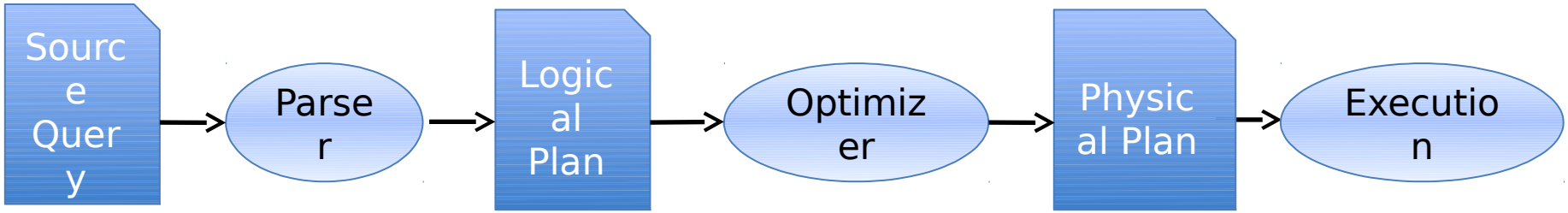
High-level Architecture



Principled Query Execution

- **Source query**—what we want to do (analyst friendly)
- **Logical Plan**— what we want to do (language agnostic, computer friendly)
- **Physical Plan**—how we want to do it (the best way we can tell)
- **Execution Plan**—where we want to do it

Principled Query Execution



SQL 2003
DrQL
MongoQL
DSL

parser API

```
query: [
 {
  @id: "log",
  op: "sequence",
  do: [
   {
    op: "scan",
    source: “logs”
   },
   {
    op: "filter",
    condition:
      "x > 3”
   },
```

Topology
CF
etc.

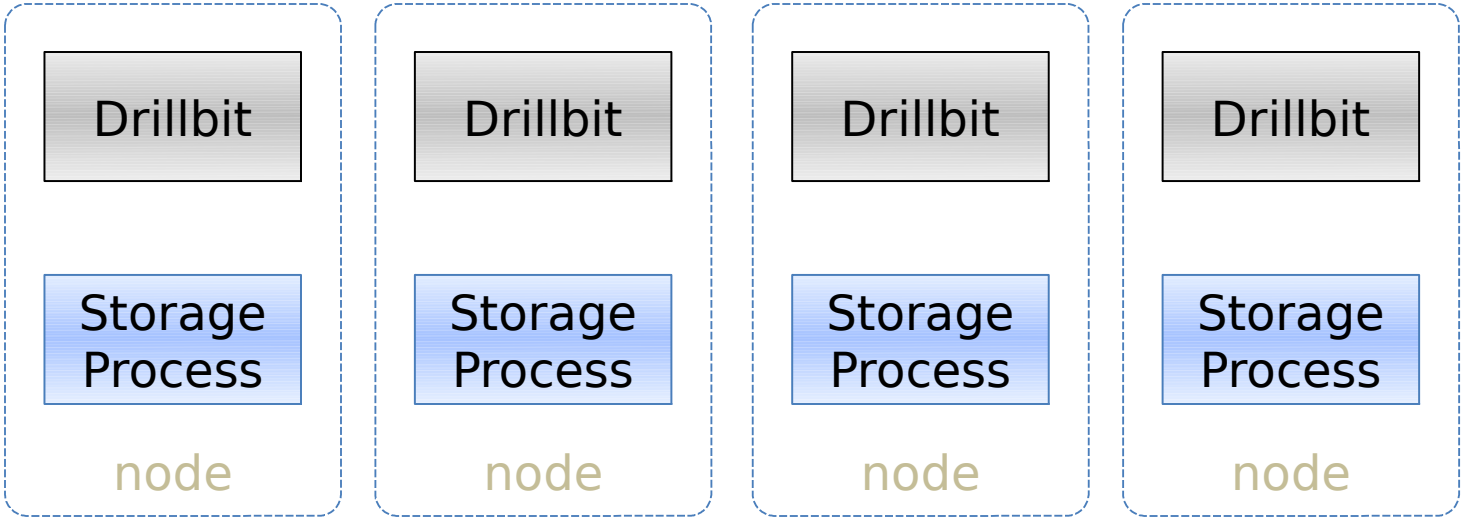
scanner API

APACHE DRILL

MAPR TECHNOLOGIES

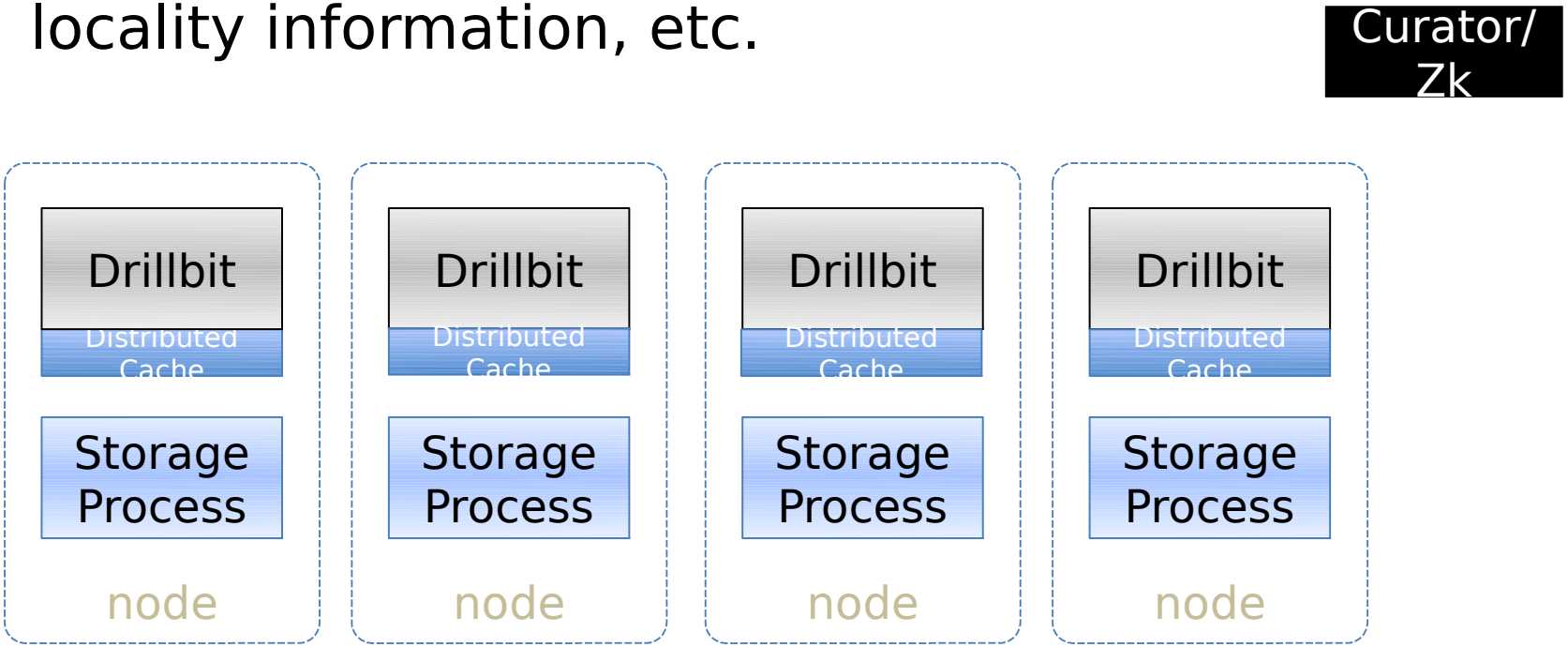
Wire-level Architecture

- Each node: **Drillbit** - maximize data locality
- Co-ordination, query planning, execution, etc, are **distributed**
- Any node can act as endpoint for a query—**foreman**



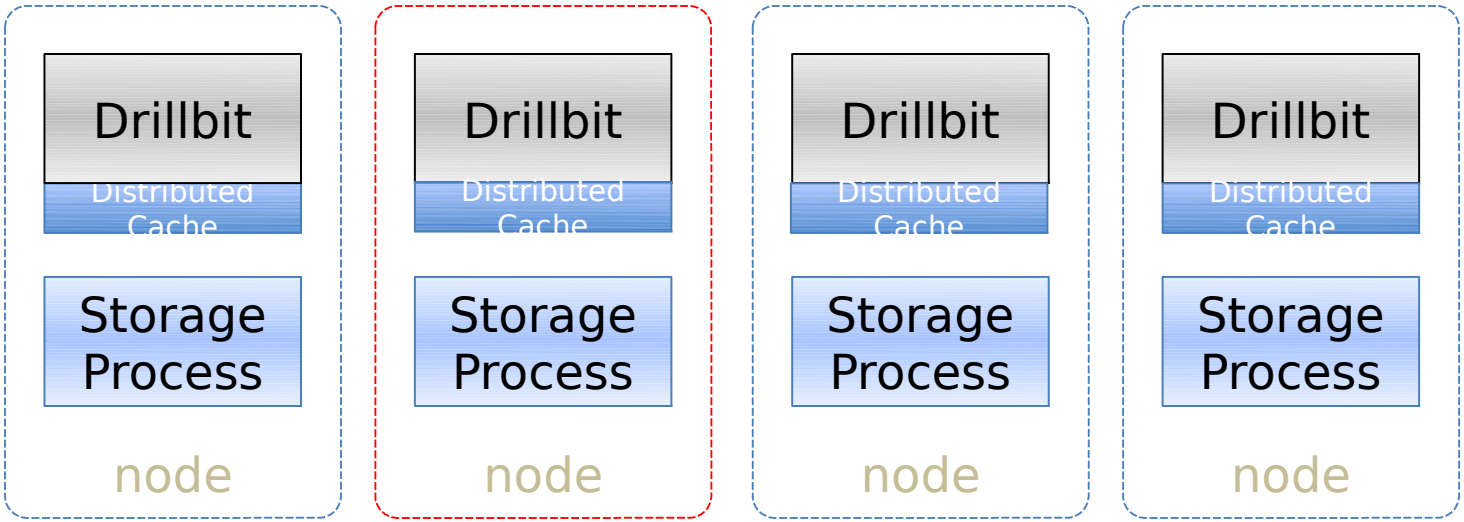
Wire-level Architecture

- **Curator/Zookeeper** for ephemeral cluster membership info
- **Distributed cache** (Hazelcast) for metadata, locality information, etc.



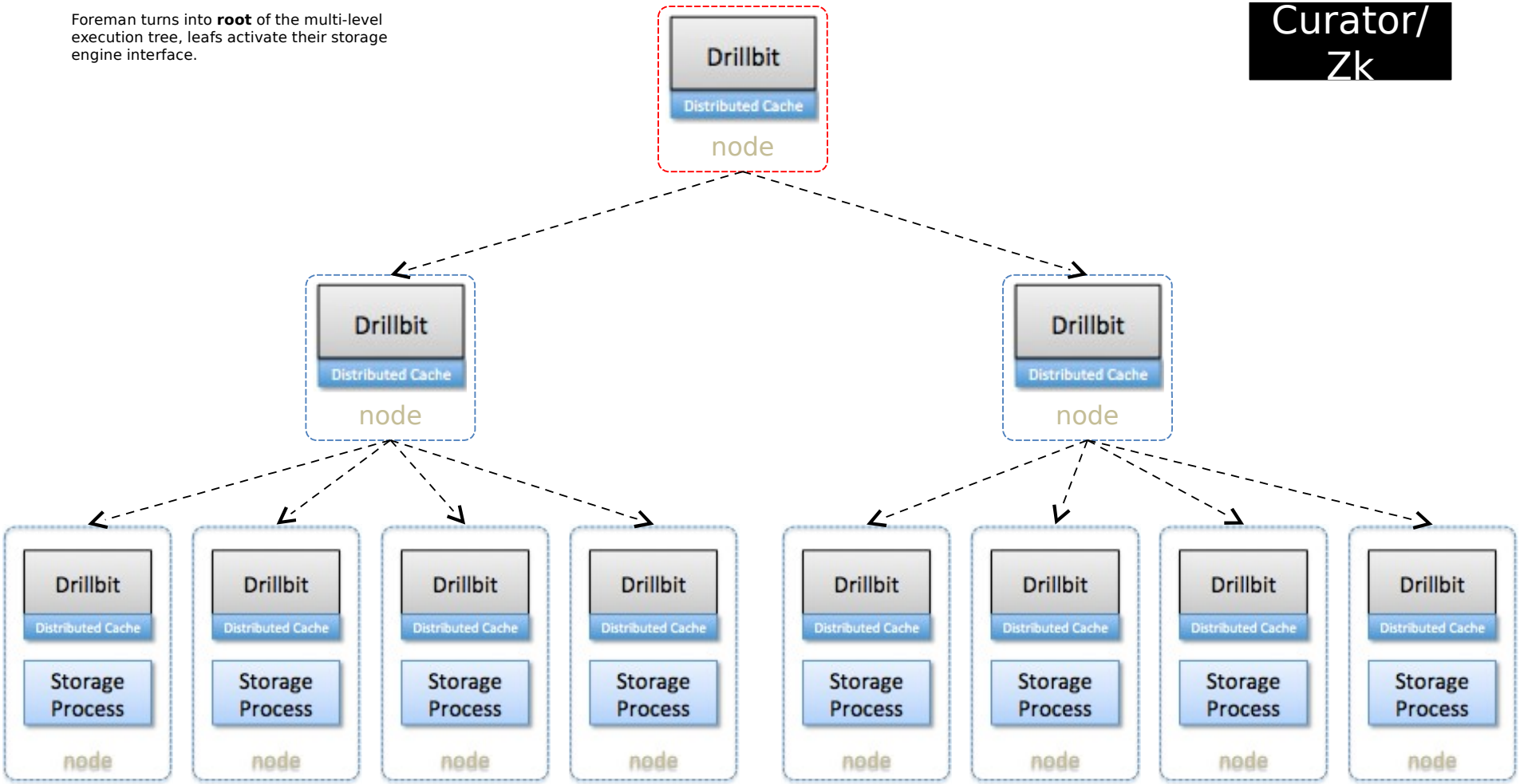
Wire-level Architecture

- Originating Drillbit acts as **foreman**: manages query execution, scheduling, locality information, etc.
- Streaming data **communication** avoiding Curator/ Zk



Wire-level Architecture

Foreman turns into **root** of the multi-level execution tree, leafs activate their storage engine interface.



APACHE DRILL

MAPR TECHNOLOGIES

On the shoulders of giants ...

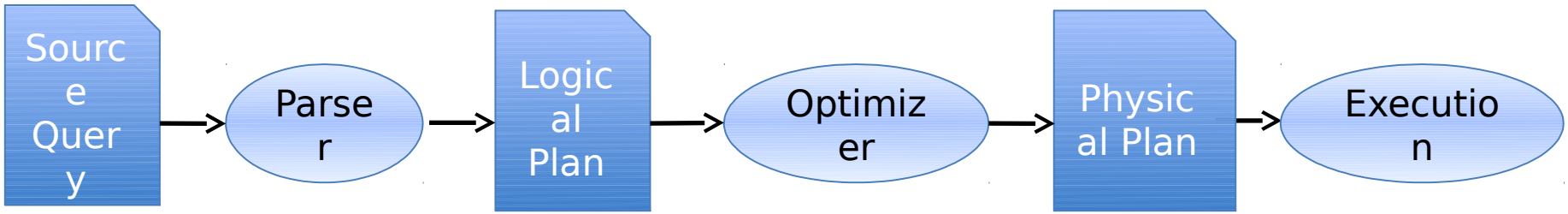
- **Jackson** for JSON SerDe for metadata
- **Typesafe HOCON** for configuration and module management
- **Netty4** as core RPC engine, protobuf for communication
- Vanilla Java, **Larray** and **Netty ByteBuf** for off-heap large data structures
- **Hazelcast** for distributed cache
- Netflix **Curator** on top of Zookeeper for service registry
- **Optiq** for SQL parsing and cost optimization
- **Parquet** (http://parquet.io) as native columnar format
- **Janino** for expression compilation
- **ASM** for ByteCode manipulation
- **Yammer Metrics** for metrics
- **Guava** extensively
- **Carrot HPC** for primitive collections

Key features

- Full SQL – ANSI SQL 2003
- Nested Data as first class citizen
- Optional Schema
- Extensibility Points ...

Extensibility Points

- Source query ▭ parser API
- Custom operators, UDF ▭ logical plan
- Serving tree, CF, topology ▭ physical plan/optimizer
- Data sources &formats ▭ scanner API



User Interfaces

- **API**—DrillClient
  - Encapsulates endpoint discovery
  - Supports logical and physical plan submission, query cancellation, query status
  - Supports streaming return results
- **JDBC** driver, converting JDBC into DrillClient communication.
- **REST** proxy for DrillClient

… and Hadoop?

- How is it different to Hive, Cascading, etc.?
- Complementary use cases*
- … use Apache Drill
  - Find record with specified condition
  - Aggregation under dynamic conditions
- … use MapReduce
  - Data mining with multiple iterations
  - ETL


An Inside Look at Google BigQuery

Table of Contents

Google

*) https://cloud.google.com/files/BigQueryTechnicalWP.pdf

APACHE DRILL

MAPR TECHNOLOGIES

Let’s get our hands dirty…

Basic Demo

```
{
 "id": "0001",
 "type": "donut",
 ”ppu": 0.55,
 "batters":
 {
  "batter”:
  [
      { "id": "1001", "type": "Regular" },
      { "id": "1002", "type": "Chocolate" },
…
```

data source: **donuts.json**

```
 query:[ {
      op:"sequence",
      do:[
          {
            op: "scan",
            ref: "donuts",
            source: "local-logs",
            selection: {data: "activity"}
          },
          {
            op: "filter",
            expr: "donuts.ppu < 2.00"
          },
…
```

ogical plan: **simple_plan.json**


```
{
  "sales" : 700.0,
  "typeCount" : 1,
  "quantity" : 700,
  "ppu" : 1.0
}
 {
  "sales" : 109.71,
  "typeCount" : 2,
  "quantity" : 159,
  "ppu" : 0.69
}
 {
  "sales" : 184.25,
  "typeCount" : 2,
  "quantity" : 335,
  "ppu" : 0.55
}
```

result: **out.json**

https://cwiki.apache.org/confluence/display/DRILL/Demo+HowTo

APACHE DRILL

MAPR TECHNOLOGIES


```
SELECT
   t.cf1.name as name,
   SUM(t.cf1.sales) as total_sales
FROM m7://cluster1/sales t
GROUP BY name
ORDER BY by total_sales desc
LIMIT 10;
```


```
sequence: [
  { op: scan, storageengine: m7,
    selection: {table: sales}}
  { op: project, projections: [
   {ref: name, expr: cf1.name},
   {ref: sales, expr: cf1.sales}]}
  { op: segment, ref: by_name, exprs: [name]}
  { op: collapsingaggregate, target: by_name,
    carryovers: [name],
    aggregations: [{ref: total_sales, expr:
    sum(name)}]}
  { op: order, ordering: [{order: desc, expr:
    total_sales}]}
  { op: store, storageengine: screen}
]
```

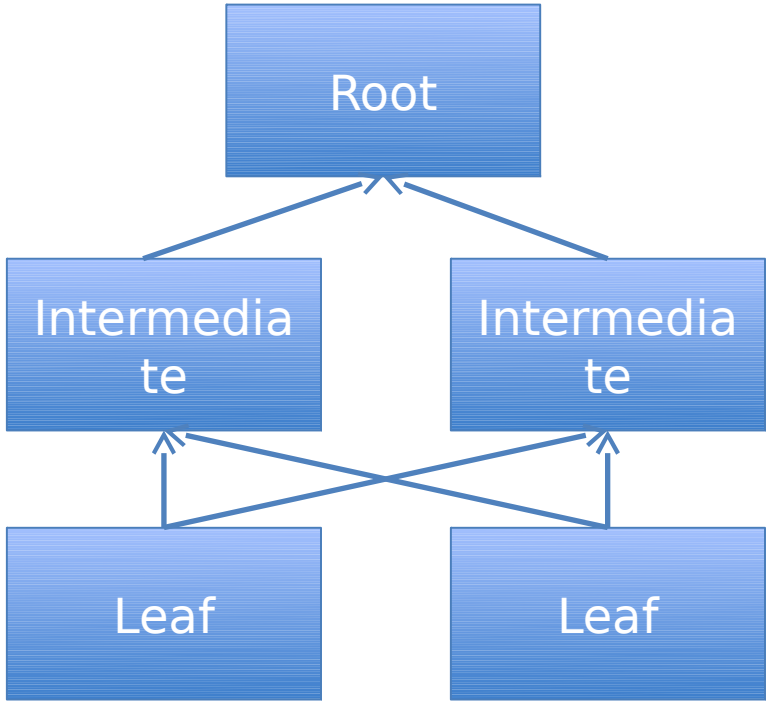

```
{
@id: 1, pop: m7scan, cluster: def,
table: sales, cols: [cf1.name, cf2.name]
}
{
@id: 2, op: hash-random-exchange,
input: 1, expr: 1
}
{
@id: 3, op: sorting-hash-aggregate, input: 2,
grouping: 1, aggr:[sum(2)], carry: [1], sort:
~agrr[0]
}
{
@id: 4, op: screen, input: 4
}
```

Execution Plan

- Break physical plan into fragments
- Determine quantity of parallelization for each task based on estimated costs
- Assign particular nodes based on affinity, load and topology

Execution Plan

- One root fragment (runs on driving node)
- Leaf fragments (first tasks to run)
- Intermediate fragments (won't start until they receive data from their children)
- In the case where the query output is routed to storage, the root operator will often receive metadata to present rather than data



Example Fragments

Root Fragment

```
{
  pop : "screen",
  @id : 1,
  child : {
    pop : "random-receiver",
    @id : 2,
    providingEndpoints :
[ "Cglsb2NhbGhvc3QY0gk=" ]
  }
}
```

Intermediate Fragment

```
{
  pop : "single-sender",
  @id : 1,
  child : {
    pop : "mock-store",
    @id : 2,
    child : {
      pop : "filter",
      @id : 3,
      child : {
        pop : "random-receiver",
        @id : 4,
        providingEndpoints :
[ "Cglsb2NhbGhvc3QYqRI=",
"Cglsb2NhbGhvc3QY0gk=" ]
      },
      expr : " ('b')  > (5) "
    }
  },
  destinations : [ "Cglsb2NhbGhvc3QYqRI=" ]
}
```

Leaf Fragment 1

```
{
  pop : "hash-partition-sender",
  @id : 1,
  child : {
    pop : "mock-scan",
    @id : 2,
    url : "http://apache.org",
    entries : [ {
      id : 1,
      records : 4000}]
  },
  destinations : [ "Cglsb2NhbGhvc3QY0gk=" ]
```

Leaf Fragment 2

```
{
  pop : "hash-partition-sender",
  @id : 1,
  child : {
    pop : "mock-scan",
    @id : 2,
    url : "http://apache.org",
    entries : [ {
      id : 1,
      records : 4000
    }, {
      id : 2,
      records : 4000
    } ]
  },
  destinations : [ "Cglsb2NhbGhvc3QY0gk=" ]
}
```

Optimizer

- Convert Logical to Physical
- Very much TBD
- Likely leverage Optiq
- Hardest problem in system, especially given lack of statistics
- Probably not parallel

Execution Engine

- Single JVM per Drillbit
- Small heap space for object management
- Small set of network event threads to manage socket operations
- Callbacks for each message sent
- Messages contain header and collection of native byte buffers
- Designed to minimize copies and ser/de costs
- Query setup and fragment runners managed via processing queues & thread pools

Data

- Records are broken into batches
- Batches contain a schema and a collection of fields
- Each field has a particular type (e.g. smallint)
- Fields (a.k.a. columns) are stored in ValueVectors
- ValueVectors are façades to byte buffers.
- The in-memory structure of each ValueVector is well defined and language agnostic
- ValueVectors defined based on the width and nature of the underlying data
- There are three sub value vector types

Execution Paradigm

- We will have a large amount of operators
- Each operator works on a batch of records at a time
- A loose goal is batches are roughly a single core's L2 cache in size
- Each batch of records carries a schema
- An operator is responsible for reconfiguring itself if a new schema arrives (or rejecting the record batch if the schema is disallowed)
- Most operators are the combination of a set of static operations along with the evaluation of query specific expressions
- Runtime compiled operators are the combination of a pre-compiled template and a runtime compiled set of expressions
- Exchange operators are converted into Senders and Receiver when execution plan is materialized
- Each operator must support consumption of a SelectionVector, a partial materialization of a filter

Storage Engine

- Input and output is done through storage engines
- Responsible for providing metadata & statistics about the data
- Exposes a set of optimizer (plan rewrite) rules to support things such as predicate pushdown
- Provides one or more storage engine specific scan operators that can support affinity exposure and task splitting
- Primary interfaces are **RecordReader** and **RecordWriter**
- RecordReaders are responsible for
  - Converting stored data into canonical ValueVector format
  - Providing schema for each record batch
- Our initial storage engines will be for DFS and HBase

Be a part of it!

Status

- Heavy development by multiple organizations

- Available
  - Logical plan (ADSP)
  - Reference interpreter
  - Basic SQL parser
  - Basic demo

Status

May 2013

- Full SQL support (+JDBC)
- Physical plan
- In-memory compressed data interfaces
- Distributed execution

Status

May 2013

- HBase and MySQL storage engine
- WebUI client

Contributing

Contributions appreciated (not only code drops) …

- Test data & test queries
- Use case scenarios (textual/SQL queries)
- Documentation
- Further schedule
  - Alpha Q2
  - Beta Q3

Kudos to ...

- Julian Hyde, Pentaho
- Lisen Mu, XingCloud
- Tim Chen, Microsoft
- Chris Merrick, RJMetrics
- David Alves, UT Austin
- Sree Vaadi, SSS
- Jacques Nadeau, MapR

Engage!

- Follow @ApacheDrill on Twitter

- Sign up at mailing lists (user | dev)
  http://incubator.apache.org/drill/mailing-lists.html

- Standing G+ hangouts every Tuesday at 5pm IT
  http://j.mp/apache-drill-hangouts

- Keep an eye on http://drill-user.org/