Big Data Infrastructure

Session 7: Extending MapReduce

Jimmy Lin
University of Maryland
Monday, March 23, 2015

Today's Agenda

- Making Hadoop more efficient
- Tweaking the MapReduce programming model
- Setup for… What's beyond MapReduce?


Hadoop is slow...
Source: Wikipedia (Tortoise)

A Major Step Backwards?

- MapReduce is a step backward in database access:
  - Schemas are good
  - Separation of the schema from the application is good
  - High-level access languages are good
- MapReduce is poor implementation
  - Brute force and only brute force (no indexes, for example)
- MapReduce is not novel
- MapReduce is missing features
  - Bulk loader, indexing, updates, transactions…
- MapReduce is incompatible with DMBS tools

Source: Blog post by DeWitt and Stonebraker

Hadoop vs. Databases: Grep

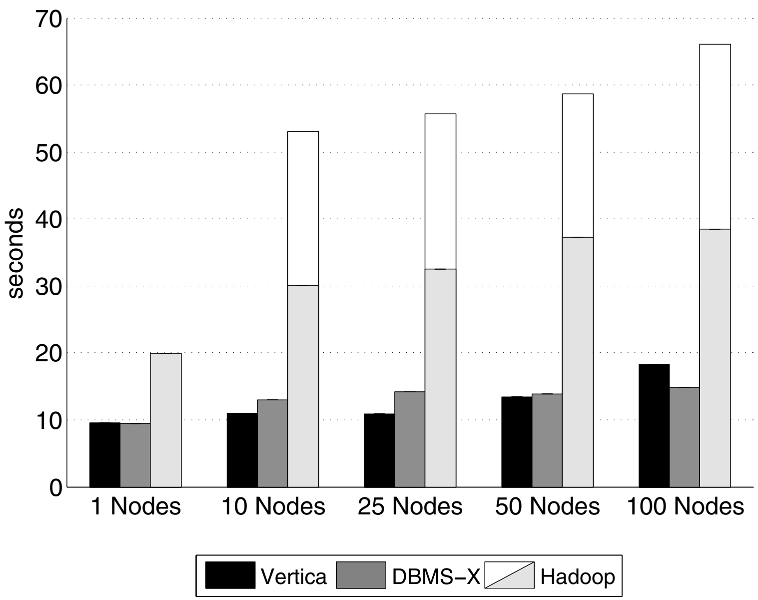


Figure 4: Grep Task Results – 535MB/node Data Set

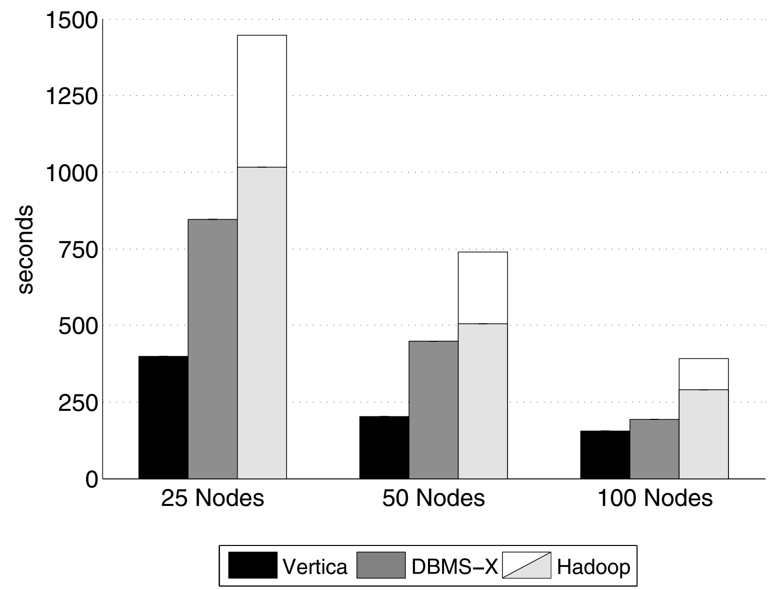


Figure 5: Grep Task Results – 1TB/cluster Data Set

```
SELECT * FROM Data WHERE field LIKE '%XYZ%';
```

Source: Pavlo et al. (2009) A Comparison of Approaches to Large-Scale Data Analysis. SIGMOD.

Hadoop vs. Databases: Select

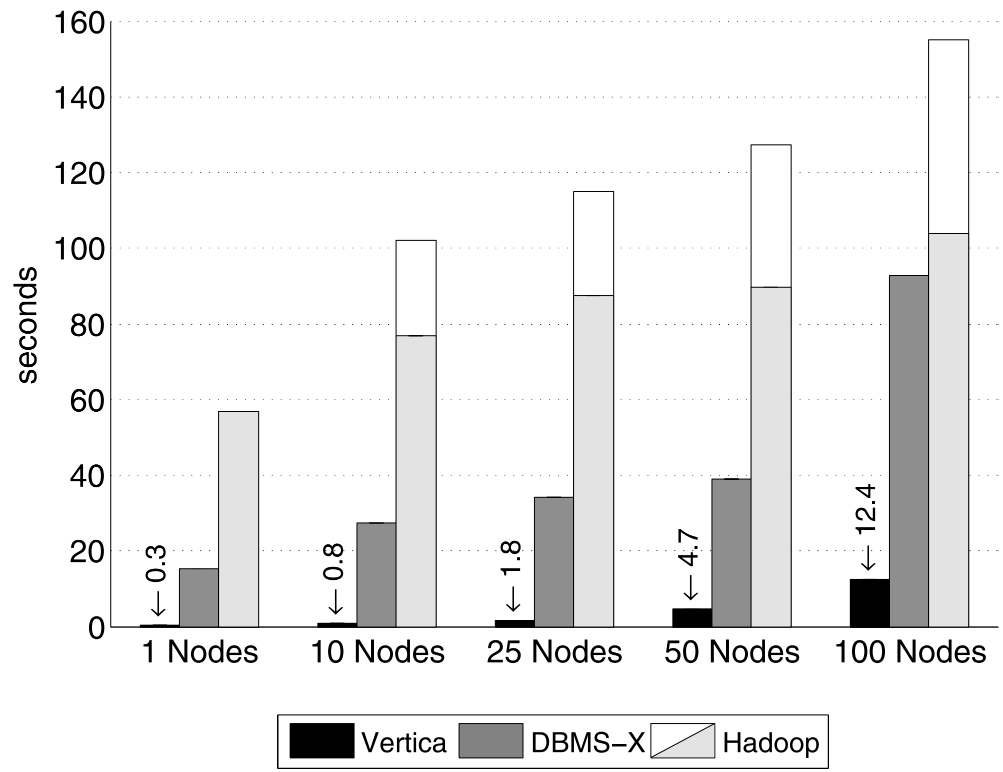


Figure 6: Selection Task Results

```
SELECT pageURL, pageRank
FROM Rankings WHERE pageRank > X;
```

Source: Pavlo et al. (2009) A Comparison of Approaches to Large-Scale Data Analysis. SIGMOD.

Hadoop vs. Databases: Aggregation

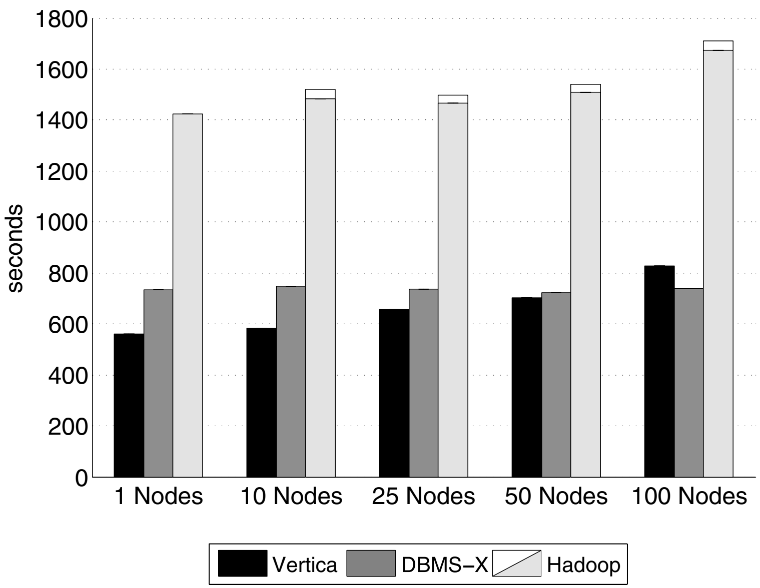


Figure 7: Aggregation Task Results (2.5 million Groups)

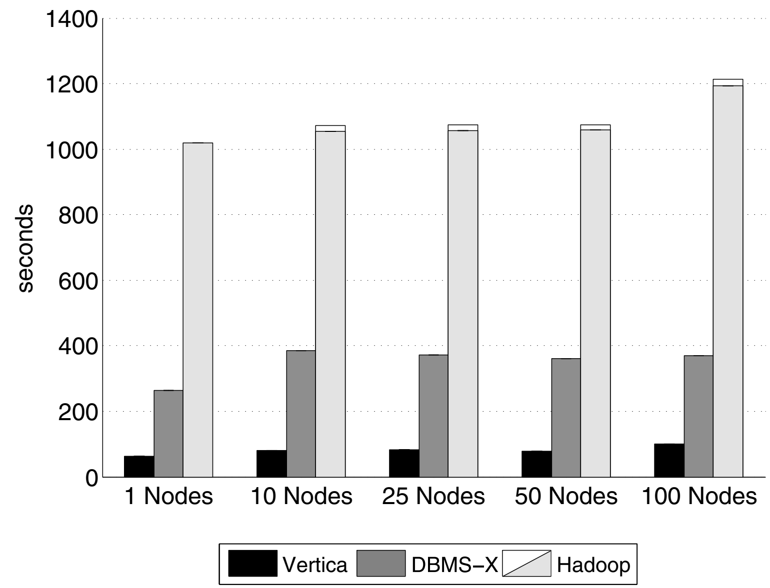


Figure 8: Aggregation Task Results (2,000 Groups)

```
SELECT sourceIP, SUM(adRevenue)
FROM UserVisits GROUP BY sourceIP;
```

Source: Pavlo et al. (2009) A Comparison of Approaches to Large-Scale Data Analysis. SIGMOD.

Hadoop vs. Databases: Join

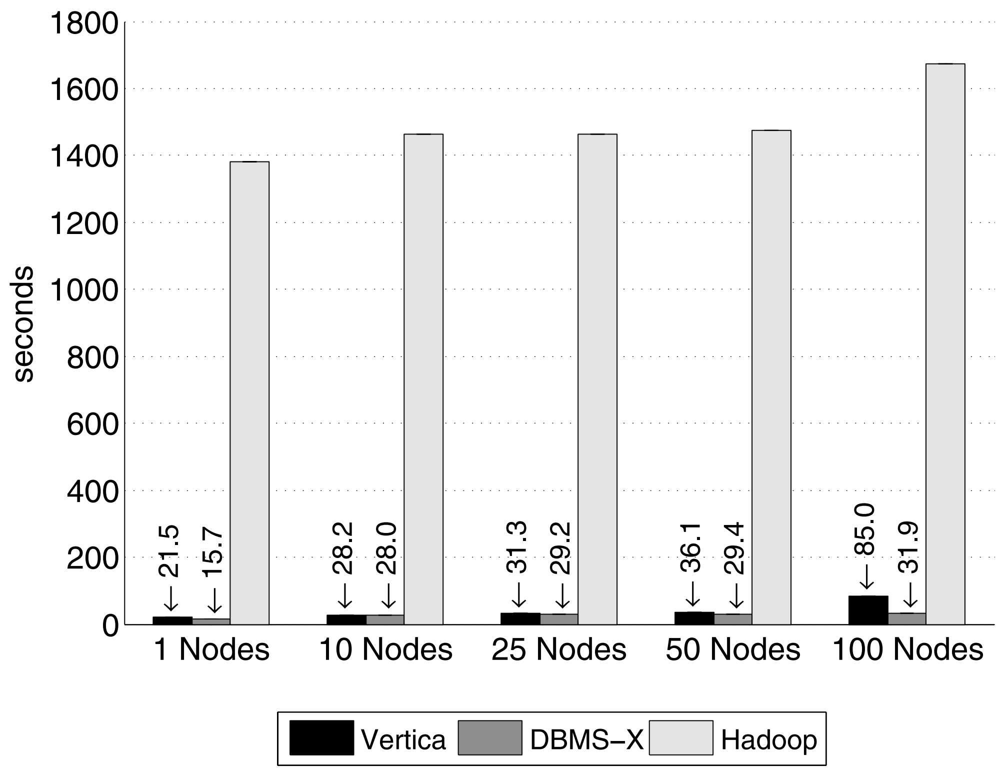


Figure 9: Join Task Results

Source: Pavlo et al. (2009) A Comparison of Approaches to Large-Scale Data Analysis. SIGMOD.

Jeff Hammerbacher, Information Platforms and the Rise of the Data Scientist.
In, *Beautiful Data*, O'Reilly, 2009.

"On the first day of logging the Facebook clickstream, more than 400 gigabytes of data was collected. The load, index, and aggregation processes for this data set really taxed the Oracle data warehouse. Even after significant tuning, we were unable to aggregate a day of clickstream data in less than 24 hours."

Why?

Integer.parseInt
String.substring

Source: Wikipedia (Tortoise)

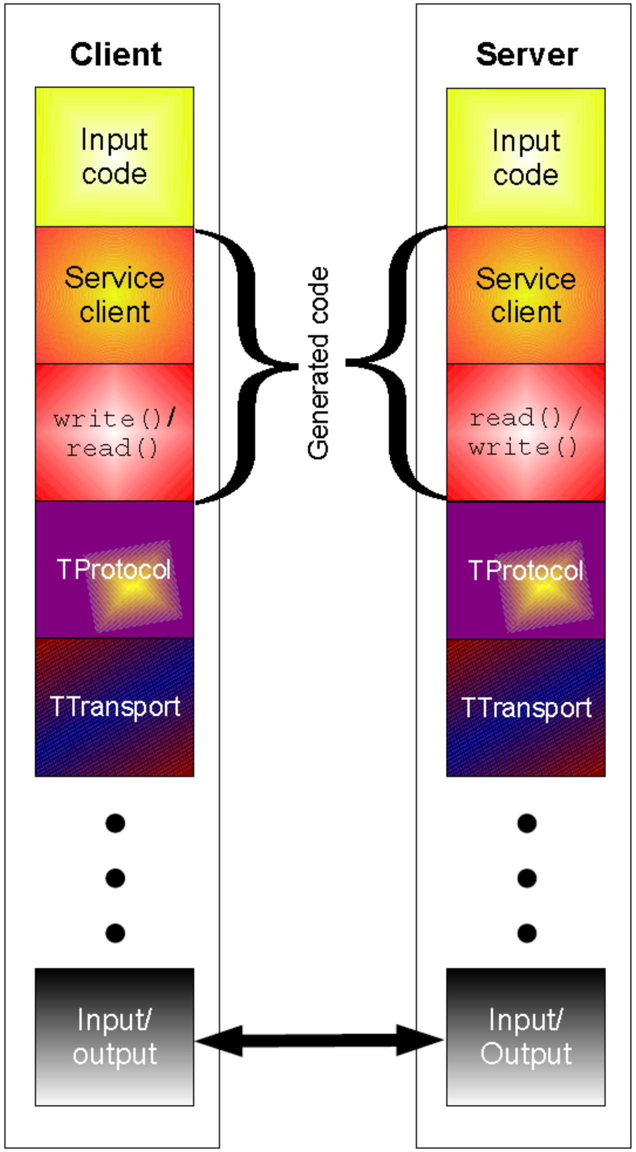
Schemas are a good idea!

- Parsing fields out of flat text files is slow
- Schemas define a contract, decoupling logical from physical

Thrift

- Originally developed by Facebook, now an Apache project
- Provides a DDL with numerous language bindings
  - Compact binary encoding of typed structs
  - Fields can be marked as optional or required
  - Compiler automatically generates code for manipulating messages
- Provides RPC mechanisms for service definitions
- Alternatives include protobufs and Avro

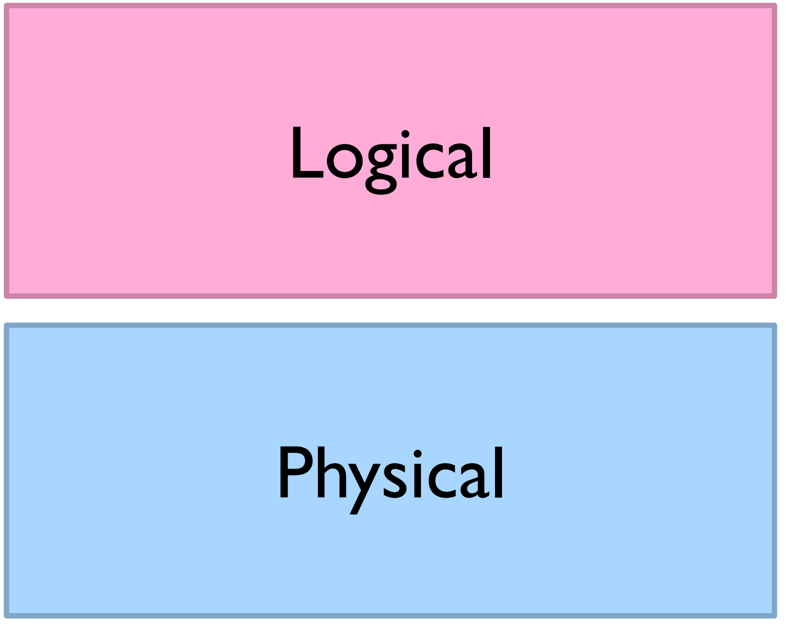
Thrift



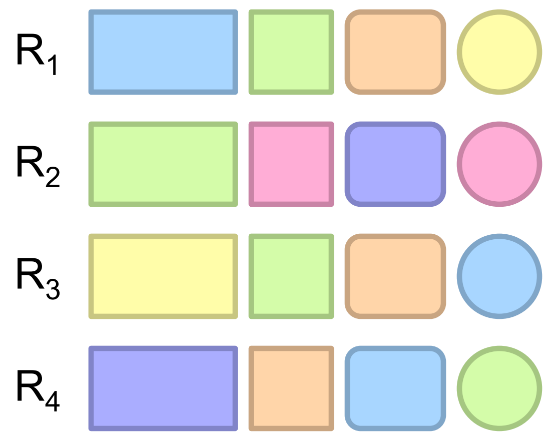

```
struct Tweet {
 1: required i32 userId;
 2: required string userName;
 3: required string text;
 4: optional Location loc;
}

struct Location {
 1: required double latitude;
 2: required double longitude;
}
```

Why not…

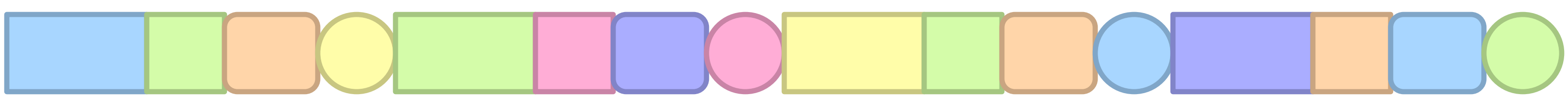
- XML or JSON?
- REST?


Logical
Physical

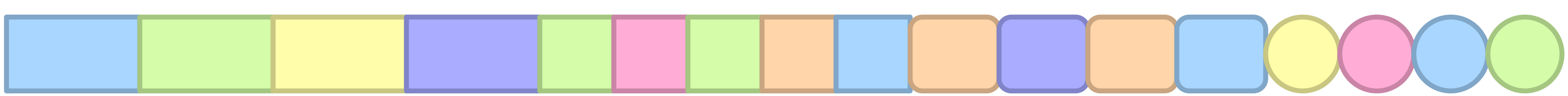
Row vs. Column Stores



Row store



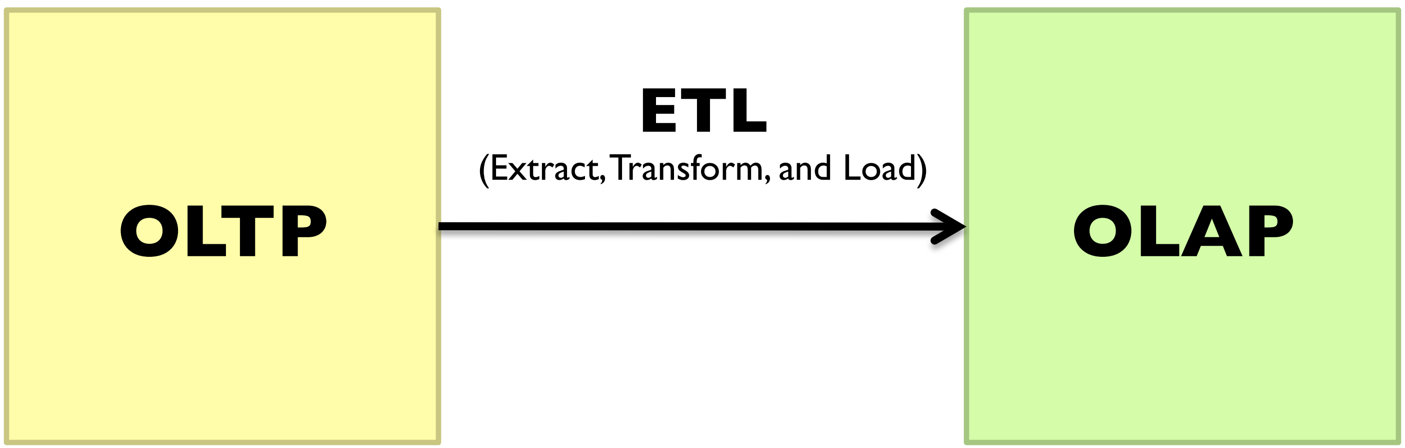
Column store



Row vs. Column Stores

- Row stores
  - Easy to modify a record
  - Might read unnecessary data when processing
- Column stores
  - Only read necessary data when processing
  - Tuple writes require multiple accesses

OLTP/OLAP Architecture

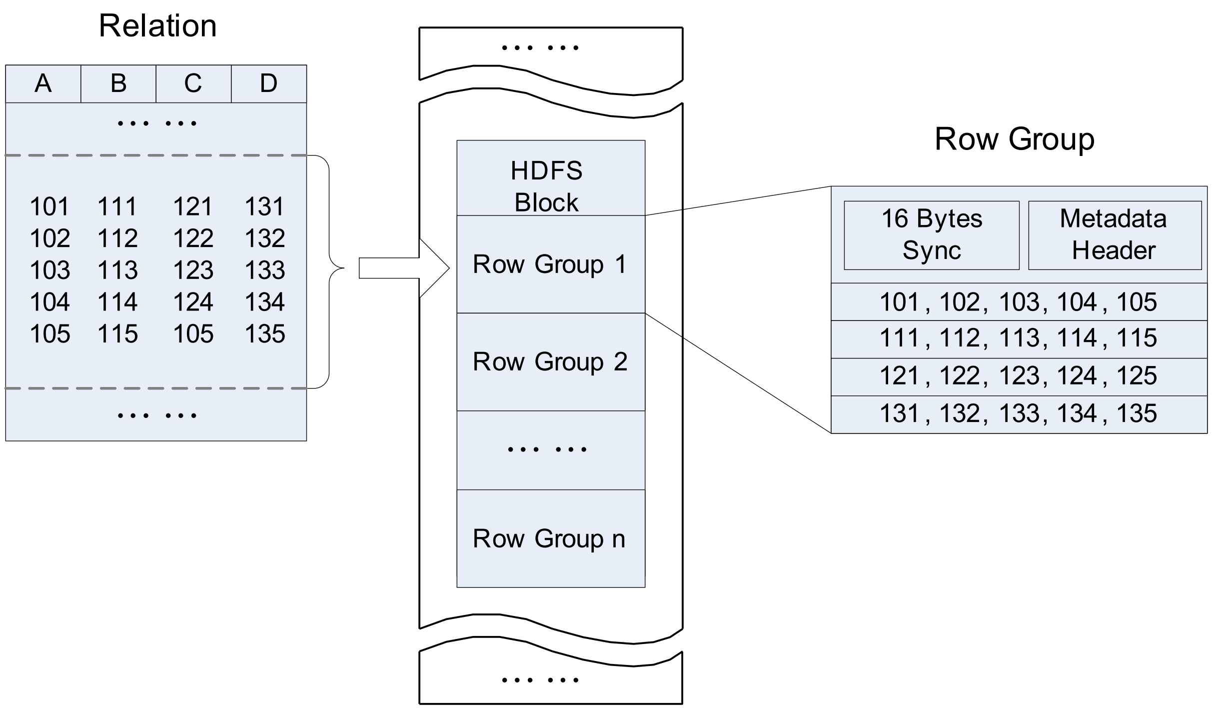


Advantages of Column Stores

- Read efficiency
- Better compression
- Vectorized processing
- Opportunities to operate directly on compressed data

Why not in Hadoop? No reason why not!

RCFile



Source: He et al. (2011) RCFile: A Fast and Space-Efficient Data Placement Structure in MapReduce-based Warehouse Systems. ICDE.

What about semi-structured data?

```
message AddressBook {
  required string owner;
  repeated string ownerPhoneNumbers;
  repeated group contacts {
    required string name;
    optional string phoneNumber;
  }
}
```

Required: exactly one occurrence
Optional: 0 or 1 occurrence
Repeated: 0 or more occurrences

Columnar Decomposition

Column	Type
owner	string
ownerPhoneNumbers	string
contacts.name	string
contacts.phoneNumber	string

What's the issue?

What’s the solution?

- Google’s Dremel storage model
- Open-source implementation in Parquet

Source: https://blog.twitter.com/2013/dremel-made-simple-with-parquet

Optional and Repeated Elements

Schema: List of Strings	**Data:** `[ "a", "b", "c", ...]`

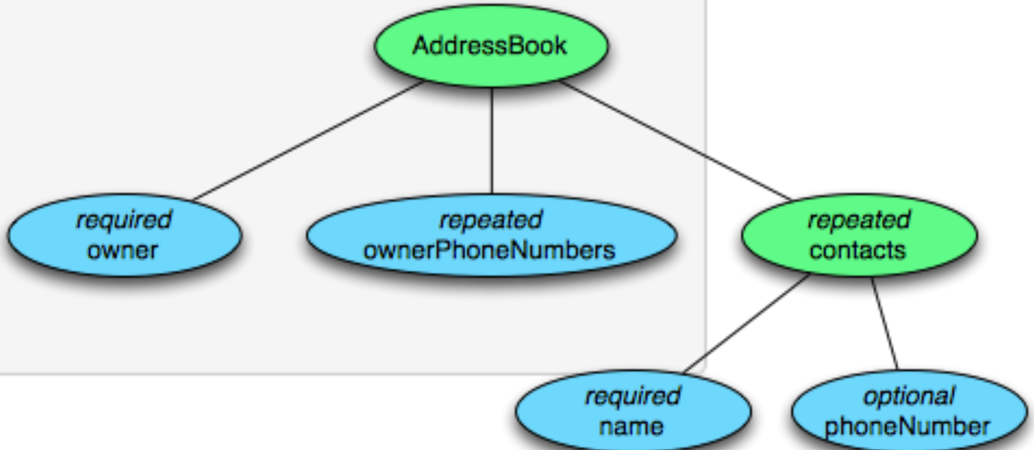
`message ExampleList {`
`    repeated string list;`
`}`
	`{`
	`    list: "a",`
	`    list: "b",`
	`    list: "c",`
	`    …`
	`}`

Schema: Map of strings to strings	**Data:** `{"AL" => "Alabama", ...}`

`message ExampleMap {`
`    repeated group map {`
`      required string key;`
`      optional string value;`
`    }`
`}`
	`{`
	`    map: {`
	`        key: "AL",`
	`        value: "Alabama"`
	`    },`
	`    map: {`
	`        key: "AK",`
	`        value: "Alaska"`
	`    },`
	`    …`
	`}`

Tree Decomposition

```
message AddressBook {
  required string owner;
  repeated string ownerPhoneNumbers;
  repeated group contacts {
    required string name;
    optional string phoneNumber;
  }
}
```



Columnar Decomposition

Column	Type
owner	string
ownerPhoneNumbers	string
contacts.name	string
contacts.phoneNumber	string

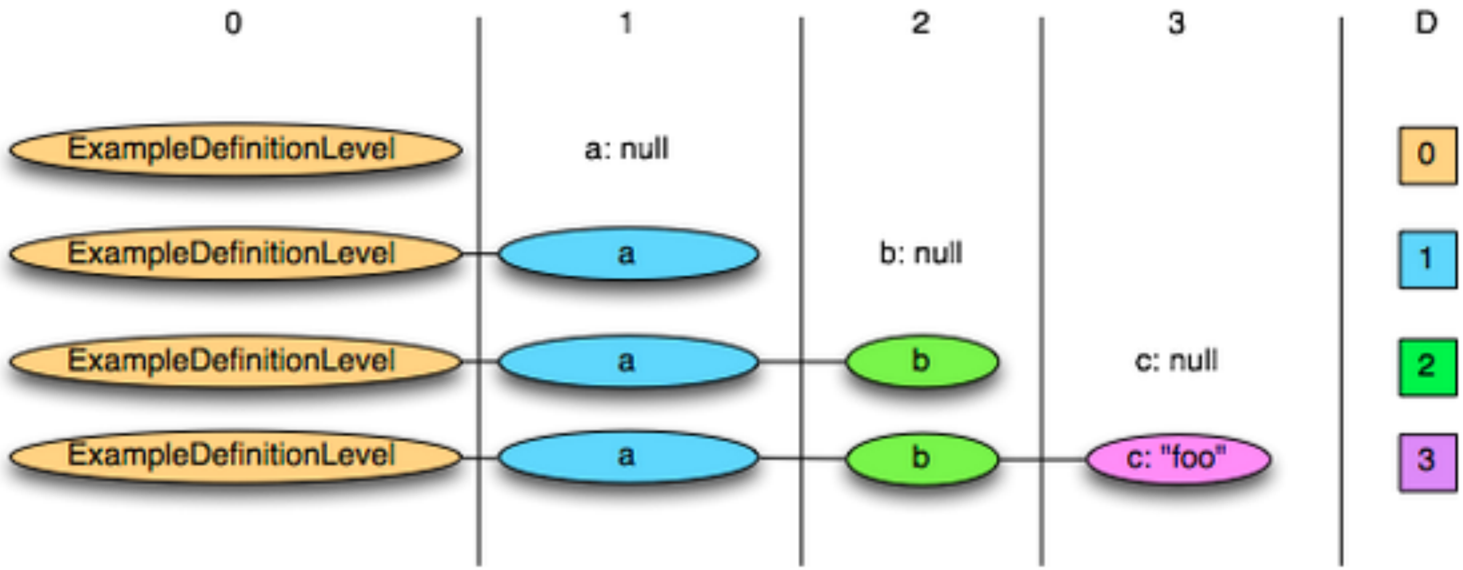
What other information do we need to store?

Definition Level

```
message ExampleDefinitionLevel {
  optional group a {
    optional group b {
      optional string c;
    }
  }
}
```

Value	Definition Level
`a: null`	0
`a: { b: null }`	1
`a: { b: { c: null } }`	2
`a: { b: { c: "foo" } }`	3 (actually defined)

Definition Level: Illustration

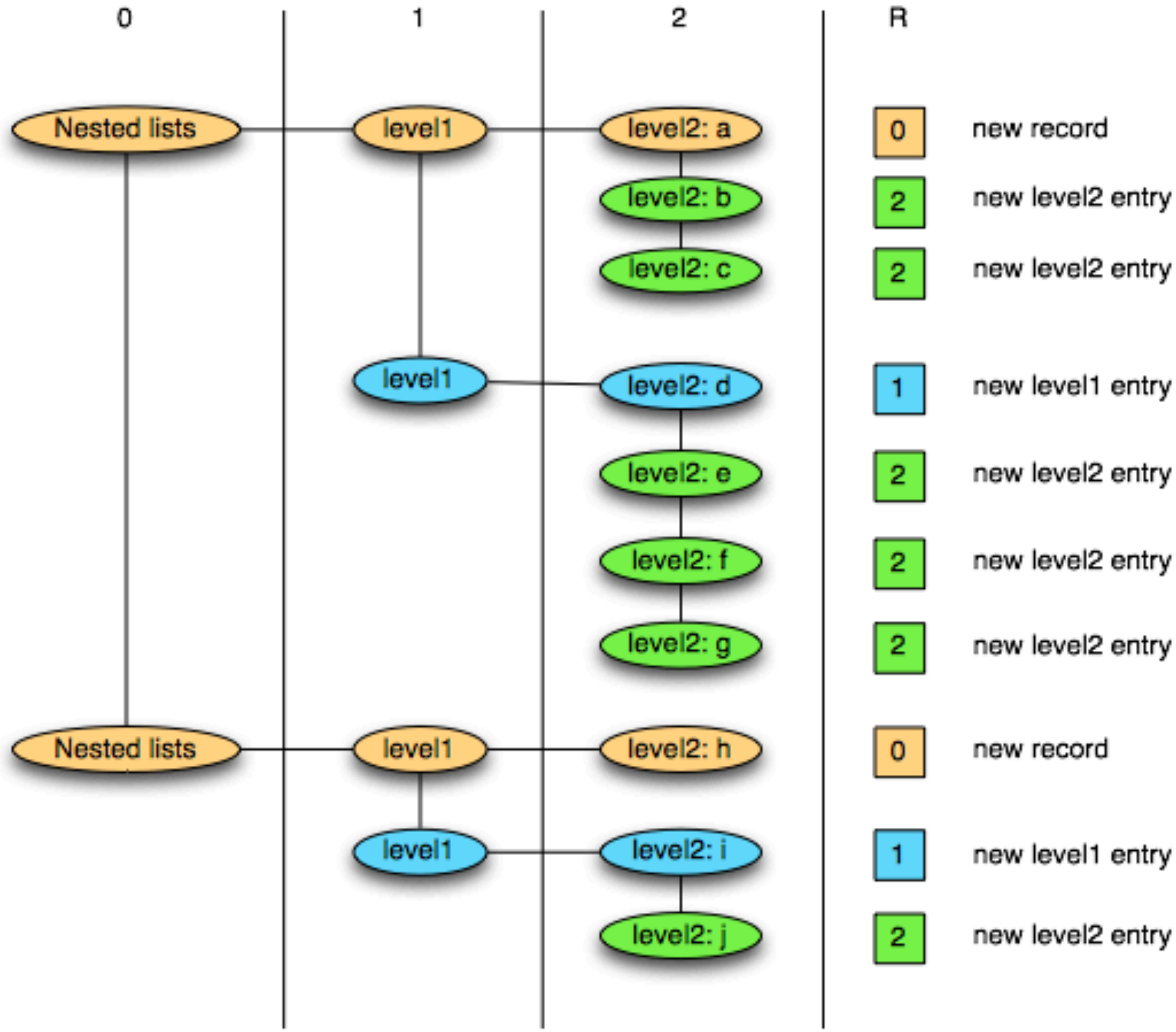


Repetition Level

Schema:	Data: `[[a,b,c],[d,e,f,g]],[[h],[i,j]]`

<pre>message nestedLists {
   repeated group level1 {
      repeated string level2;
   }
}</pre>
	<pre>{
	      level1: {
	            level2: a
	            level2: b
	            level2: c
	      },
	      level1: {
	            level2: d
	            level2: e
	            level2: f
	            level2: g
	      }
	}
	{
	      level1: {
	            level2: h
	      },
	      level1: {
	            level2: i
	            level2: j
	      }
	}</pre>

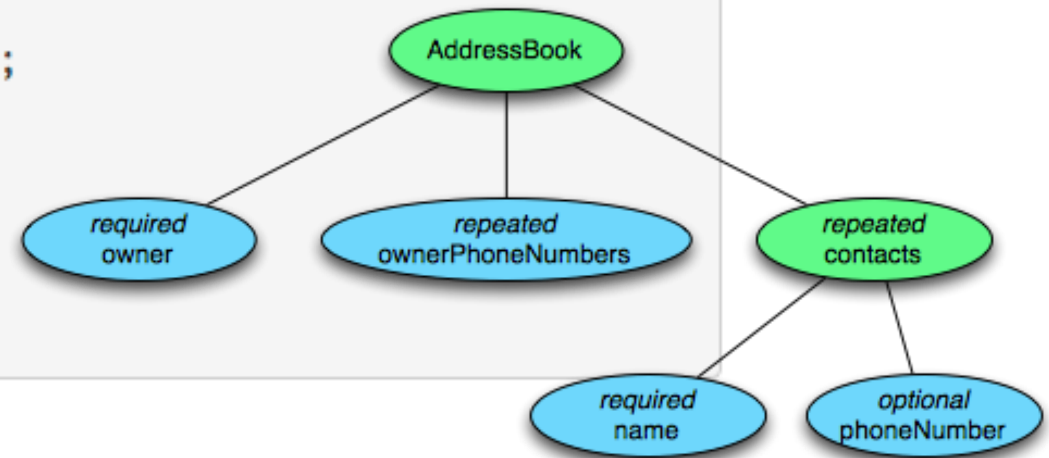
Repetition Level: Illustration



0 marks new record and implies creating a new level1 and level2 list
1 marks new level1 list and implies creating a new level2 list as well.
2 marks every new element in a level2 list.

Putting It Together

```
message AddressBook {
  required string owner;
  repeated string ownerPhoneNumbers;
  repeated group contacts {
    required string name;
    optional string phoneNumber;
  }
}
```



Columnar Decomposition

Column	Max Definition level	Max Repetition level
`owner`	0 (`owner` is *required*)	0 (no repetition)
`ownerPhoneNumbers`	1	1 (*repeated*)
`contacts.name`	1 (`name` is *required*)	1 (`contacts` is *repeated*)
`contacts.phoneNumber`	2 (`phoneNumber` is *optional*)	1 (`contacts` is *repeated*)

Sample Projection

```
AddressBook {
  owner: "Julien Le Dem",
  ownerPhoneNumbers: "555 123 4567",
  ownerPhoneNumbers: "555 666 1337",
  contacts: {
    name: "Dmitriy Ryaboy",
    phoneNumber: "555 987 6543",
  },
  contacts: {
    name: "Chris Aniszczyk"
  }
}
AddressBook {
  owner: "A. Nonymous"
}
```

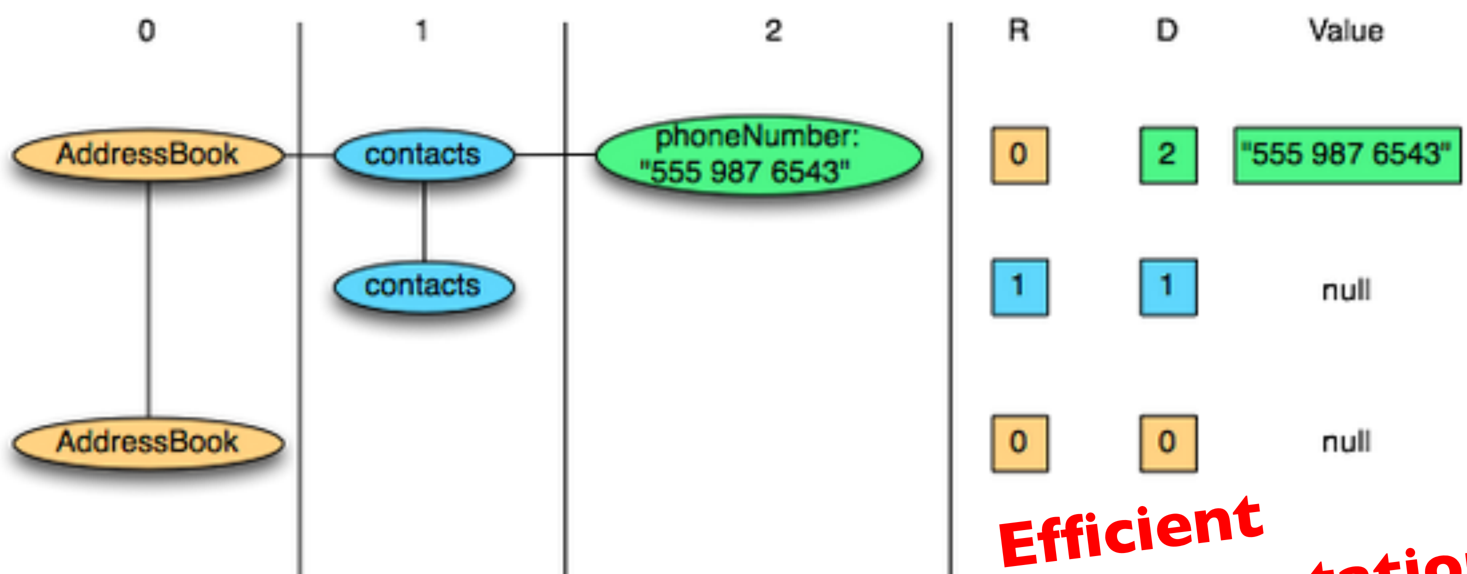
Project onto contacts.phoneNumber

```
AddressBook {
  contacts: {
    phoneNumber: "555 987 6543"
  }
  contacts: {
  }
}
AddressBook {
}
```

Physical Layout

Columnar Decomposition

Column	Type
`owner`	string
`ownerPhoneNumbers`	string
`contacts.name`	string
`contacts.phoneNumber`	string



Efficient Representations?

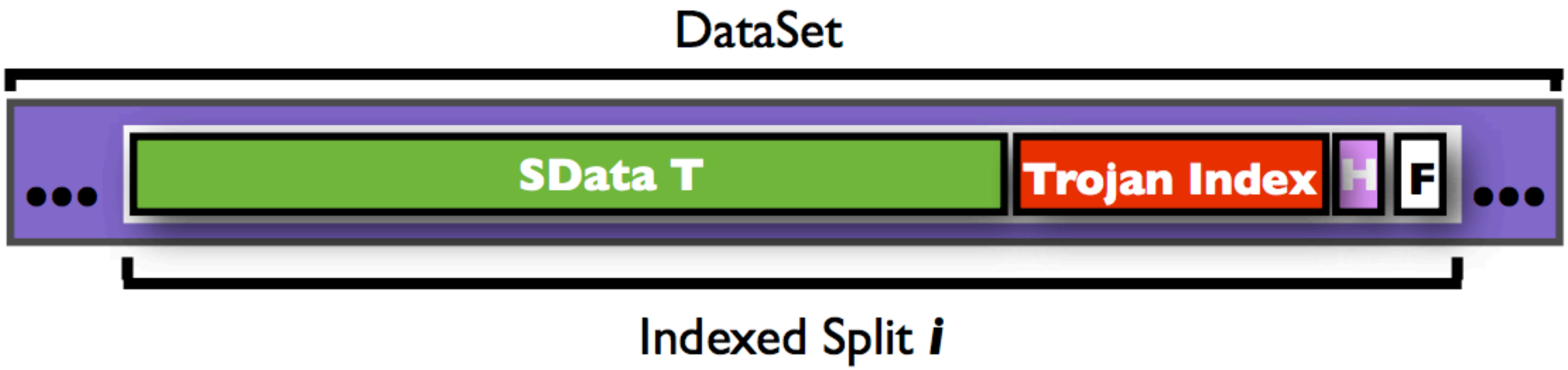
Key Ideas

- Separate logical from physical
- Preserve HDFS block structure
- Hide physical storage layout behind InputFormats

Indexes are a good thing!

Source: Wikipedia (Card Catalog)

Why not in Hadoop? No reason why not!



- Non-invasive: requires no changes to Hadoop infrastructure
- Useful for speeding up selections on joins
- Indexing building itself can be performed using MapReduce

Source: Dittrich et al. (2010) Hadoop++: Making a Yellow Elephant Run Like a Cheetah (Without It Even Noticing). VLDB.

Hadoop + Full-Text Indexes

```
status = load '/tables/statuses/2011/03/01'
    using StatusProtobufPigLoader()
    as (id: long, user_id: long, text: chararray, ...);

filtered = filter status by text matches '.*\\bhadoop\\b.*';
…
```

Pig performs a brute force scan

Then promptly chucks out most of the data Stupid.

Source: Lin et al. (2011) Full-Text Indexing for Optimizing Selection Operations in Large-Scale Data Analytics. MAPREDUCE Workshop.


FARMHAND
“Trying to find a needle in a haystack… with a snowplow”
@squarecog

Hadoop + Full-Text Indexes

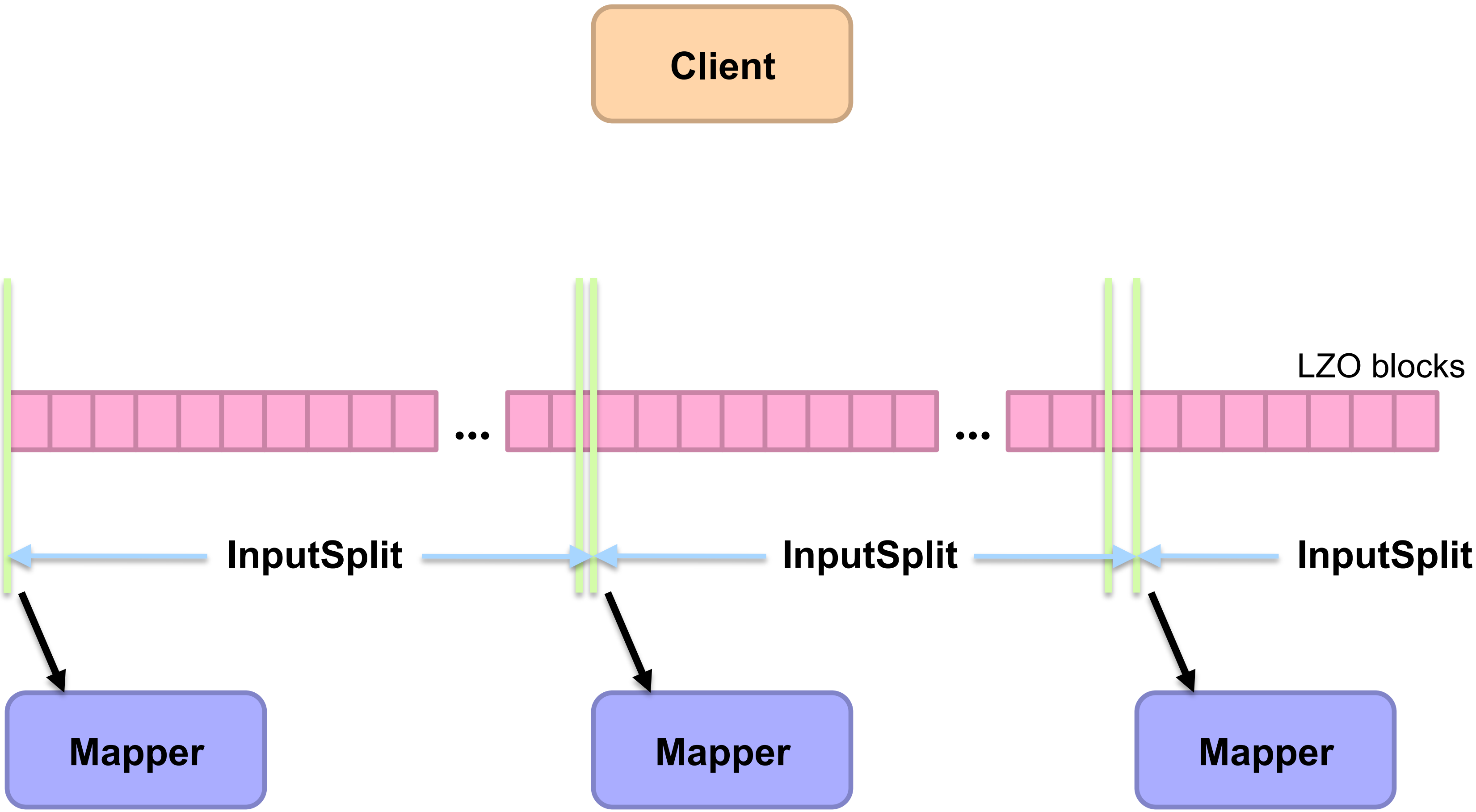
```
status = load '/tables/statuses/2011/03/01'
    using StatusProtobufPigLoader()
    as (id: long, user_id: long, text: chararray, ...);

filtered = filter status by text matches '.*\\bhadoop\\b.*';
…
```

Pig performs a brute force scan

Then promptly chucks out most of the data Stupid.

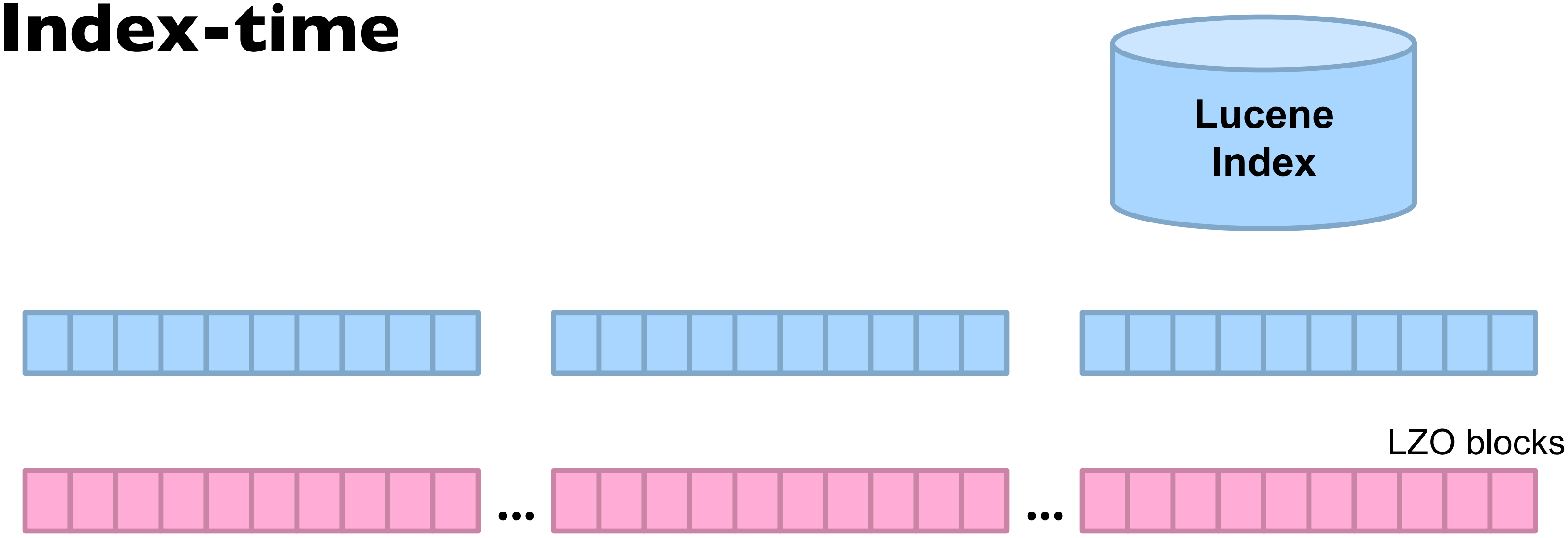
Uhhh… how about an index?

Use Lucene full-text index



Client
LZO blocks
...
...
InputSplit
InputSplit
InputSplit
Mapper
Mapper
Mapper

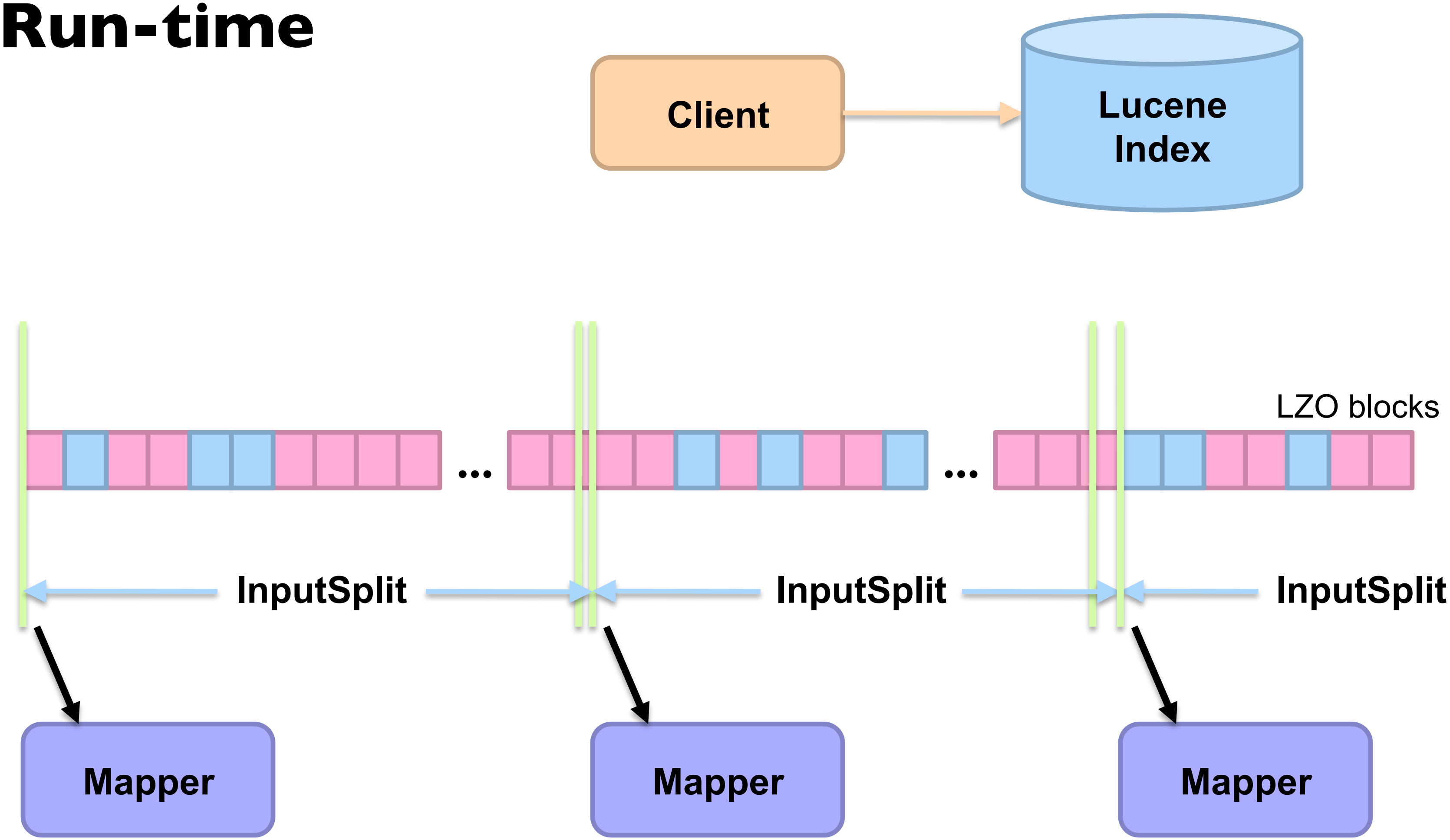
Index-time



Index for selection on tweet content

- Build "pseudo-document" for each Lzo block
- Index pseudo-documents with Lucene

Run-time



Only process blocks known to satisfy selection criteria

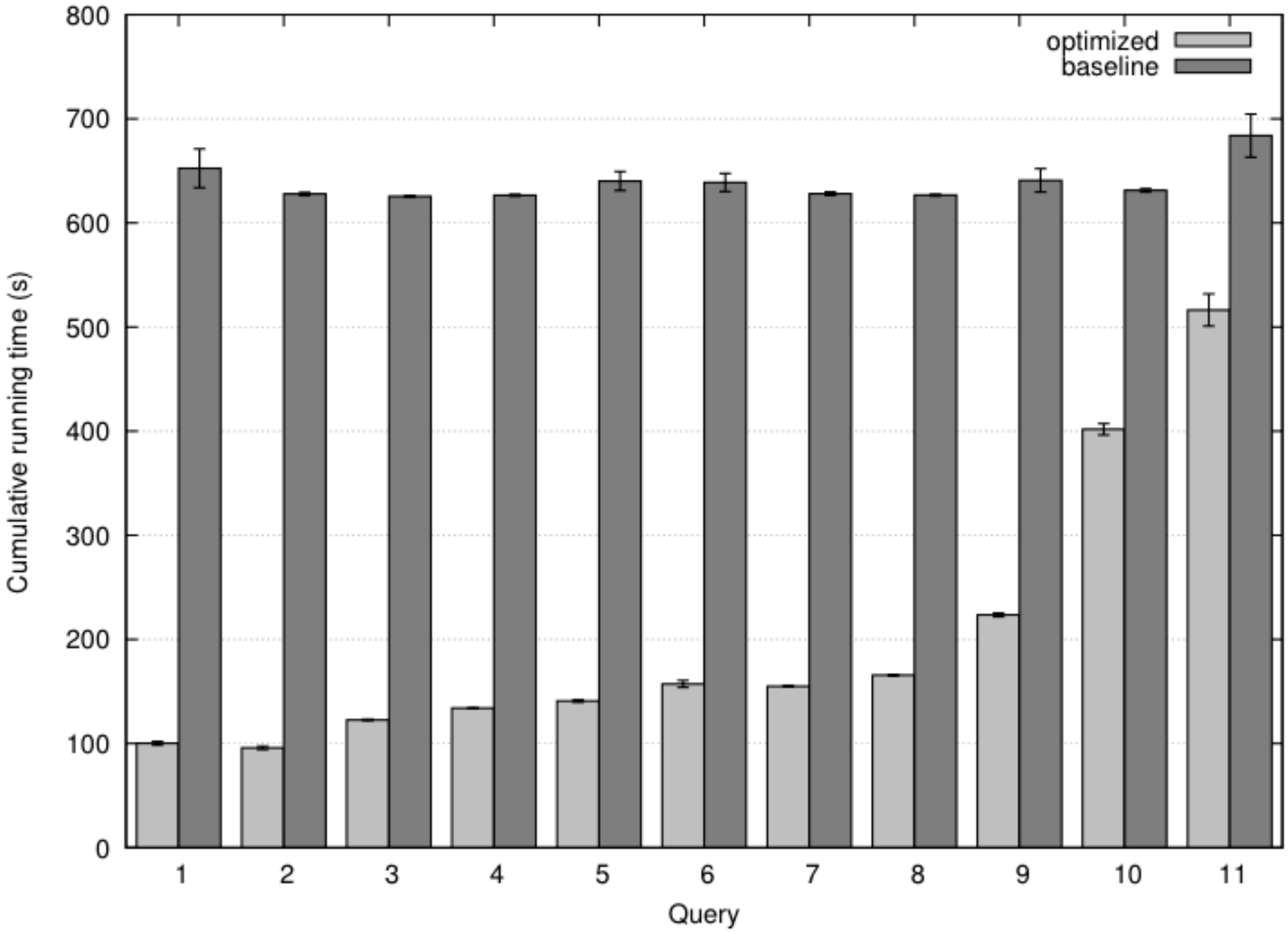
Hadoop Integration

- Everything encapsulated in the InputFormat
- RecordReaders know what blocks to process and skip
- Completely transparent to mappers

Experiments

- Selection on tweet content
- Varied selectivity range
- One day sample data (70m tweets, 8/1/2010)

	Query	Blocks	Records	Selectivity
1	hadoop	97	105	1.517×10^{-6}
2	replication	140	151	2.182×10^{-6}
3	buffer	500	559	8.076×10^{-6}
4	transactions	819	867	1.253×10^{-5}
5	parallel	999	1159	1.674×10^{-5}
6	ibm	1437	1569	2.267×10^{-5}
7	mysql	1511	1664	2.404×10^{-5}
8	oracle	1822	1911	2.761×10^{-5}
9	database	3759	3981	5.752×10^{-5}
10	microsoft	13089	17408	2.515×10^{-4}
11	data	20087	30145	4.355×10^{-4}



800
700
600
500
400
300
200
100
0
Cumulative running time (s)
optimized
baseline
1
2
3
4
5
6
7
8
9
10
11
Query

Analytical model

- Task: prediction LZO blocks scanned by selectivity
- Poisson model: P(observing *k* occurrences in a block)

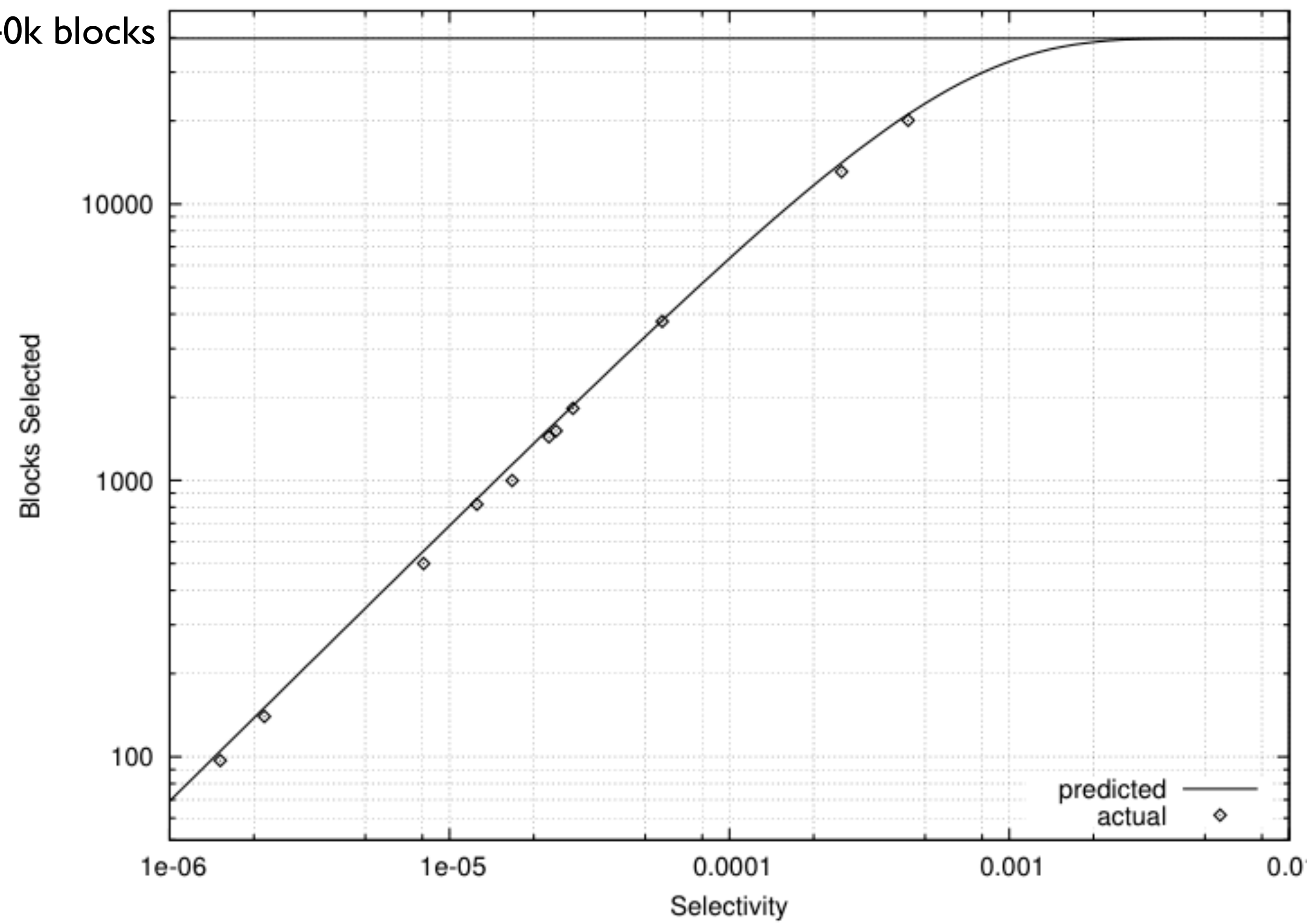
$$f(k;\lambda) = \frac{\lambda^k e^{-\lambda}}{k!} \qquad \lambda : \text{expected number of occurrences within block}$$

- E(fraction of blocks scanned):

$$1 - f(k = 0;\lambda)$$

Selectivity 0.001 → 82% of all blocks
Selectivity 0.002 → 97% of all blocks

Total: ~40k blocks



But: can predict *a priori*!

Key Ideas

- Separate logical from physical
- Preserve HDFS block structure
- Hide physical storage layout behind InputFormats

A Major Step Backwards?

- MapReduce is a step backward in database access:
  - Schemas are good ✔
  - Separation of the schema from the application is good ✔
  - High-level access languages are good ?
- MapReduce is poor implementation
  - Brute force and only brute force (no indexes, for example) ✔
- MapReduce is not novel
- MapReduce is missing features
  - Bulk loader, indexing, updates, transactions…
- MapReduce is incompatible with DMBS tools ?

Source: Blog post by DeWitt and Stonebraker

"there are known knowns; there are things we know we know. We also know there are known unknowns; that is to say we know there are some things we do not know. But there are unknown unknowns – the ones we don't know we don't know…" – Donald Rumsfeld

Source: Wikipedia

Known and Unknown Unknowns

- Databases are great if you know what questions to ask
  - “Known unknowns”
- What if you don’t know what you’re looking for?
  - “Unknown unknowns”

Tweaking Hadoop

Source: Wikipedia (Chisel)

MapReduce Hybrids

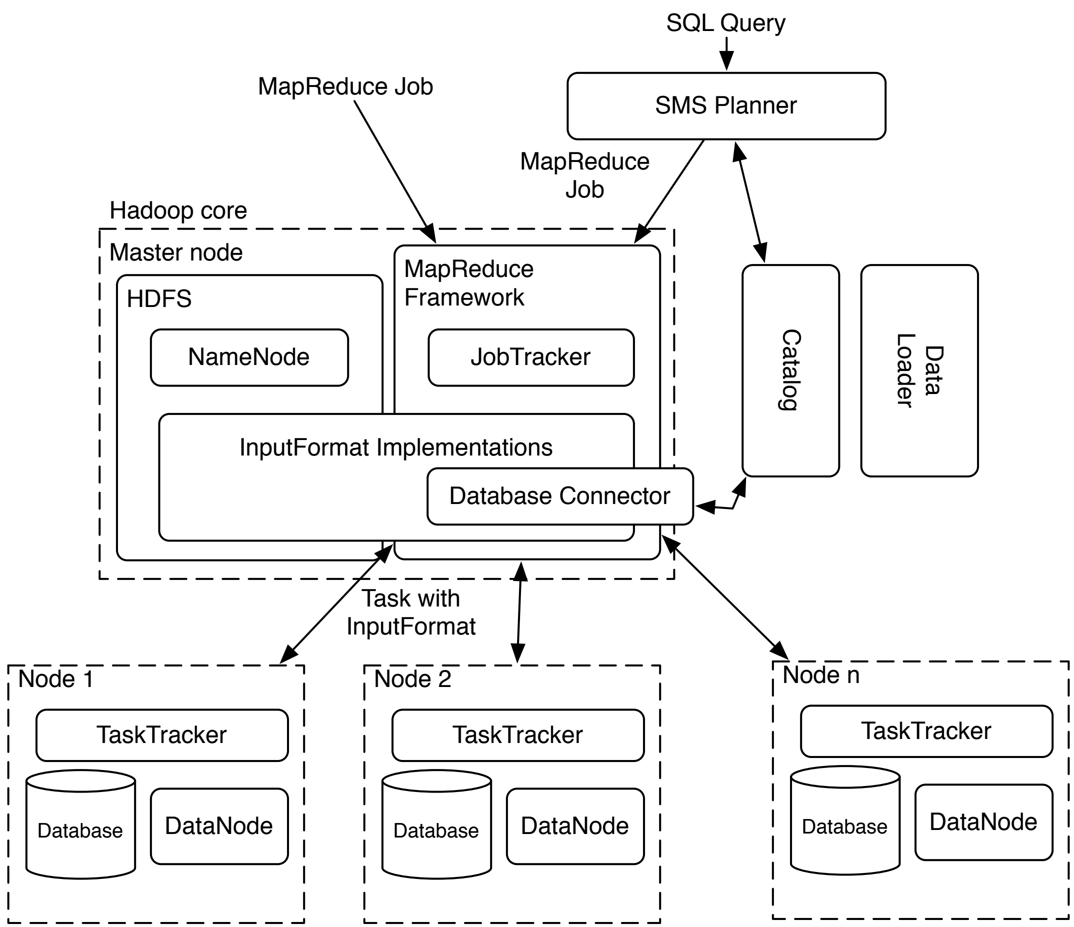
- Proposed fixes to problems with MapReduce
- Mainly presented for historical interest…

Hadoop + DBs = HadoopDB

- Why not have the best of both worlds?
  - Parallel databases focused on performance
  - Hadoop focused on scalability, flexibility, fault tolerance
- Key ideas:
  - Co-locate a RDBMS on every slave node
  - To the extent possible, “push down” operations into the DB

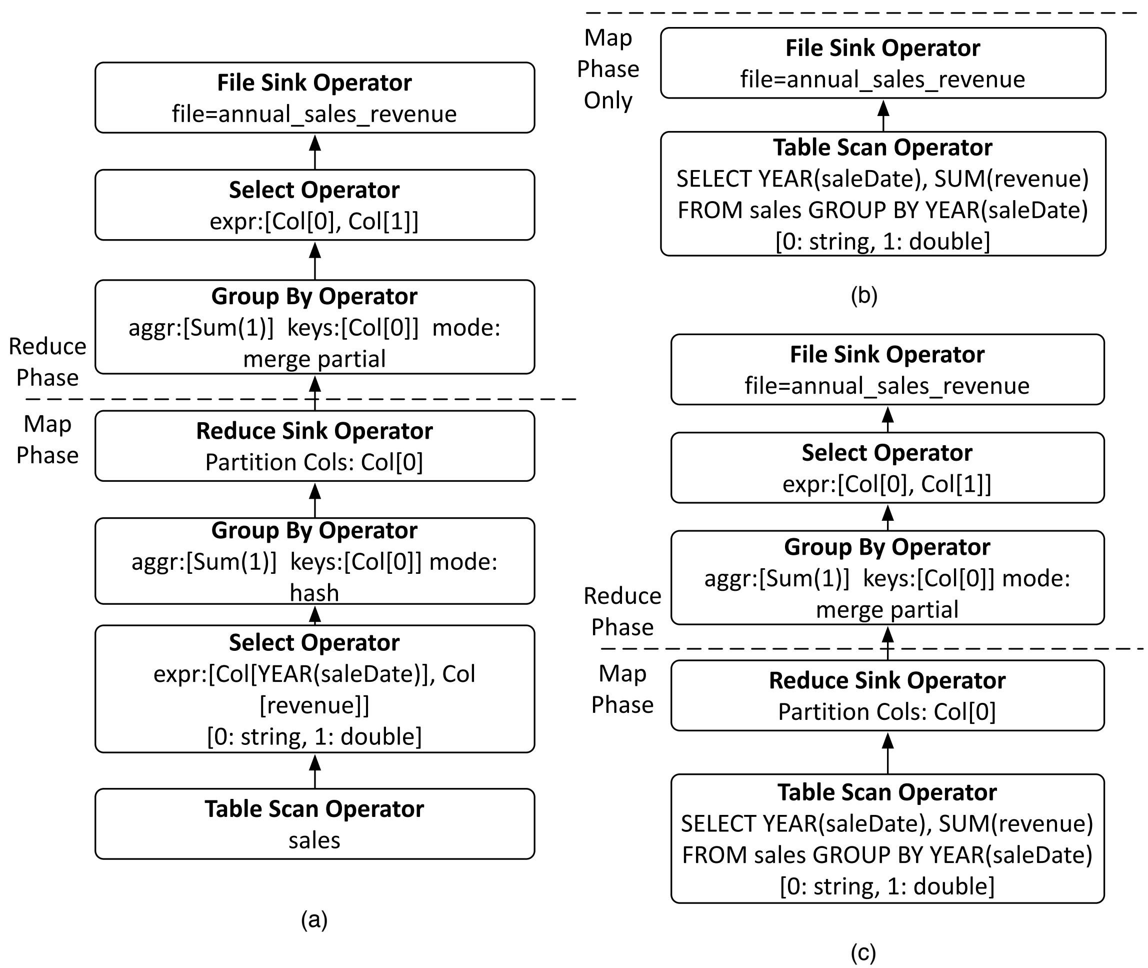
Source: Abouzeid et al. (2009) HadoopDB: An Architectural Hybrid of MapReduce and DBMS Technologies for Analytical Workloads. VLDB.

HadoopDB Architecture



Source: Abouzeid et al. (2009) HadoopDB: An Architectural Hybrid of MapReduce and DBMS Technologies for Analytical Workloads. VLDB.

HadoopDB: Query Plans




```
SELECT YEAR(saleDate), SUM(revenue)
FROM sales GROUP BY YEAR(saleDate);
```

Source: Abouzeid et al. (2009) HadoopDB: An Architectural Hybrid of MapReduce and DBMS Technologies for Analytical Workloads. VLDB.

MapReduce Sucks: Iterative Algorithms

- Java verbosity
- Hadoop task startup time
- Stragglers
- Needless data shuffling
- Checkpointing at each iteration

HaLoop: MapReduce + Iteration

Programming Model

$$R_{i+1} = R_0 \cup (R_i \bowtie L)$$

url	rank
www.a.com	1.0
www.b.com	1.0
www.c.com	1.0
www.d.com	1.0
www.e.com	1.0

(a) Initial Rank Table R_0

url_source	url_dest
www.a.com	www.b.com
www.a.com	www.c.com
www.c.com	www.a.com
www.e.com	www.d.com
www.d.com	www.b.com
www.c.com	www.e.com
www.e.com	www.c.com
www.a.com	www.d.com

(b) Linkage Table L

$$MR_1 \begin{cases} T_1 = R_i \bowtie_{url=url\_source} L \\ T_2 = \gamma_{url, rank, \frac{rank}{\texttt{COUNT}(url\_dest)} \rightarrow new\_rank}(T_1) \\ T_3 = T_2 \bowtie_{url=url\_source} L \end{cases}$$

$$MR_2 \left\{ R_{i+1} = \gamma_{url\_dest \rightarrow url, \texttt{SUM}(new\_rank) \rightarrow rank}(T_3) \right.$$

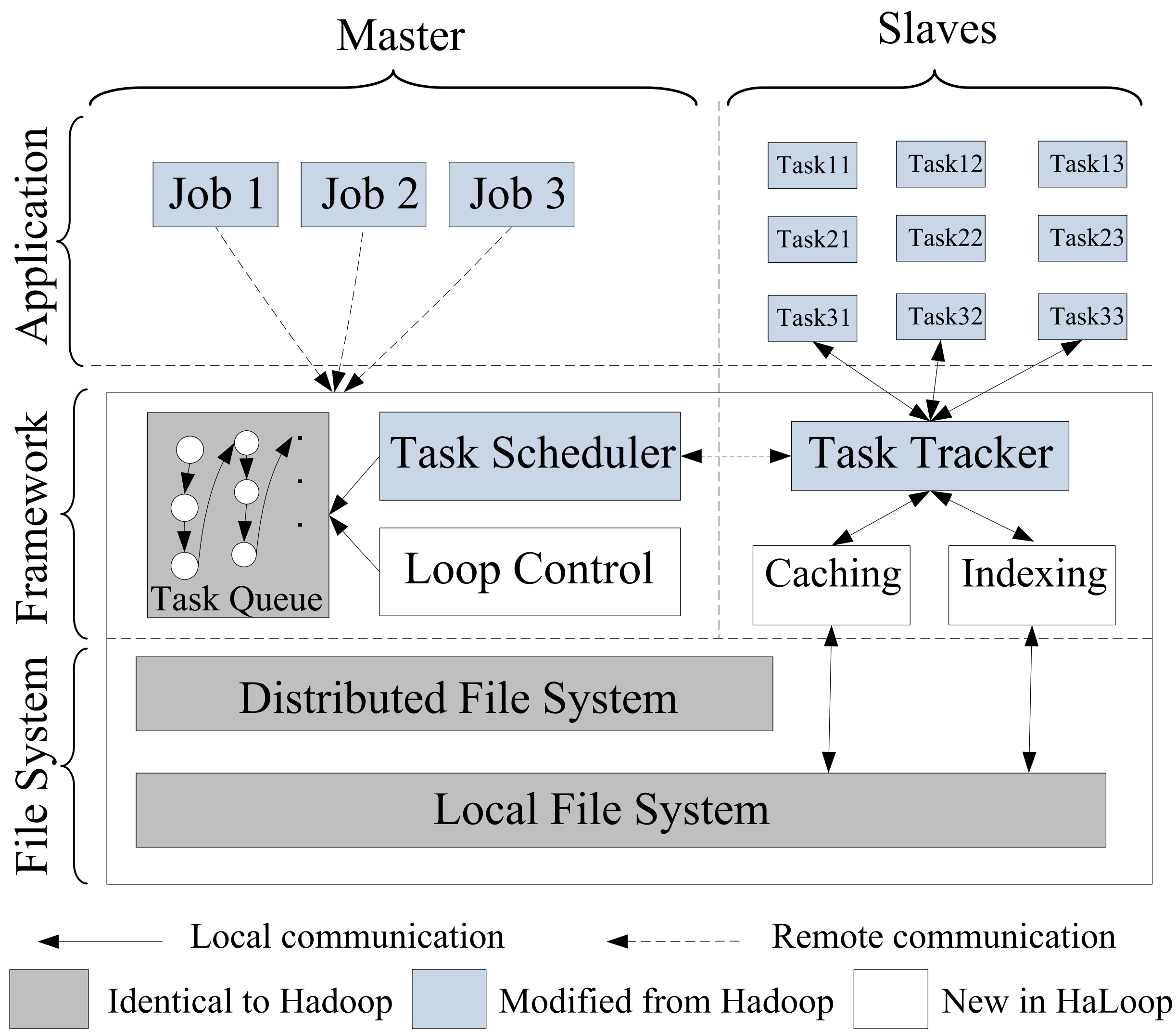
(c) Loop Body

url	rank
www.a.com	2.13
www.b.com	3.89
www.c.com	2.60
www.d.com	2.60
www.e.com	2.13

(d) Rank Table R_3

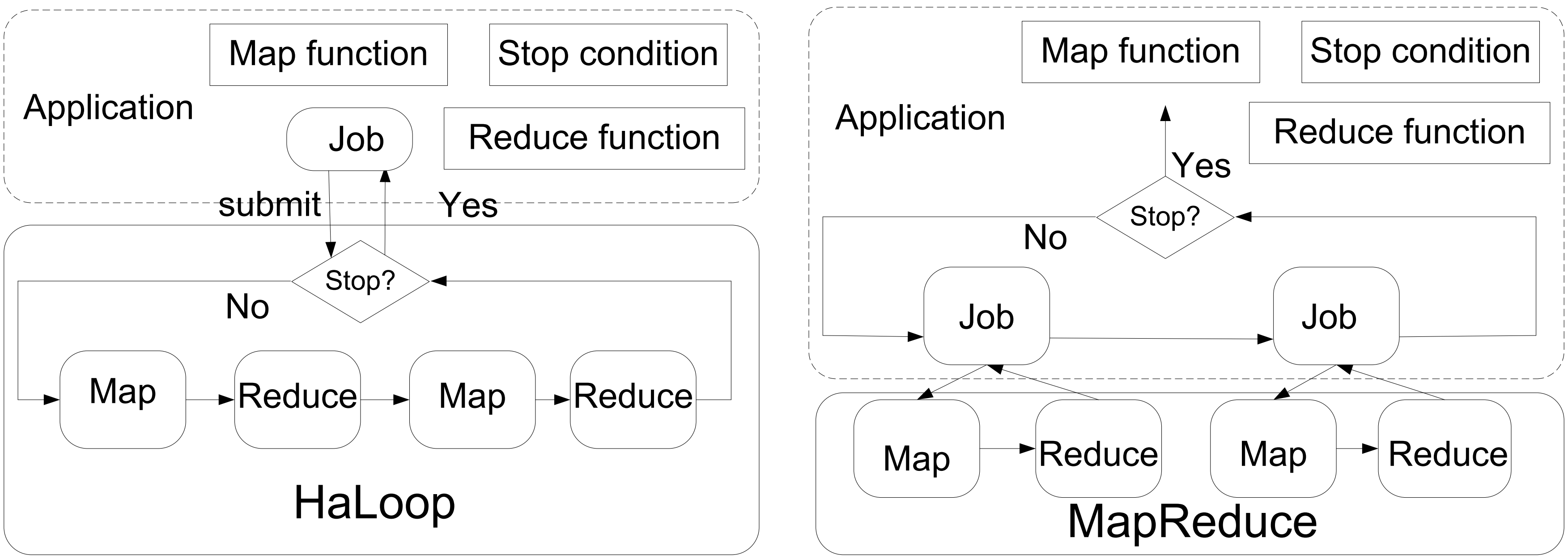
Source: Bu et al. (2010) HaLoop: Efficient Iterative Data Processing on Large Clusters. VLDB.

HaLoop Architecture



Source: Bu et al. (2010) HaLoop: Efficient Iterative Data Processing on Large Clusters. VLDB.

HaLoop: Loop Aware Scheduling



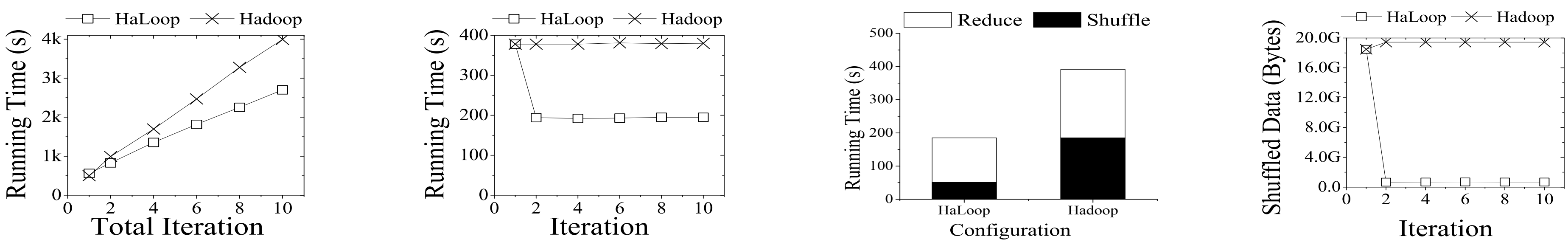
Source: Bu et al. (2010) HaLoop: Efficient Iterative Data Processing on Large Clusters. VLDB.

HaLoop: Optimizations

- Loop-aware scheduling
- Caching
  - Reducer input for invariant data
  - Reducer output speeding up convergence checks

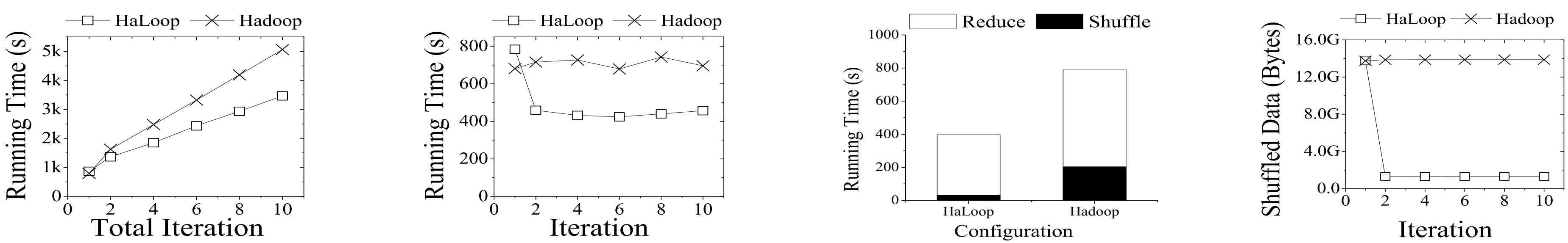
Source: Bu et al. (2010) HaLoop: Efficient Iterative Data Processing on Large Clusters. VLDB.

HaLoop: Performance



(a) Overall Performance (b) Join Step (c) Cost Distribution (d) Shuffled Bytes

Figure 9: PageRank Performance: HaLoop vs. Hadoop (Livejournal Dataset, 50 nodes)



(a) Overall Performance (b) Join Step (c) Cost Distribution (d) Shuffled Bytes

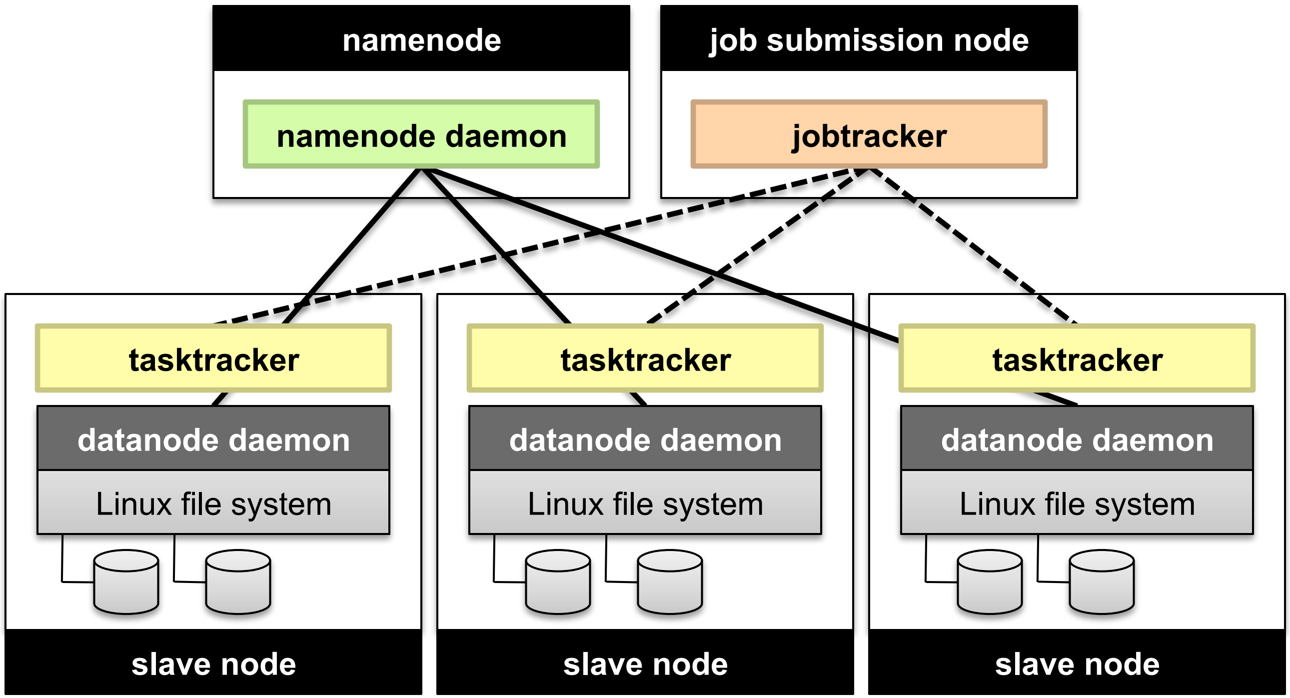
Figure 10: PageRank Performance: HaLoop vs. Hadoop (Freebase Dataset, 90 nodes)

Source: Bu et al. (2010) HaLoop: Efficient Iterative Data Processing on Large Clusters. VLDB.

Beyond MapReduce…

Source: Wikipedia (Demolition)

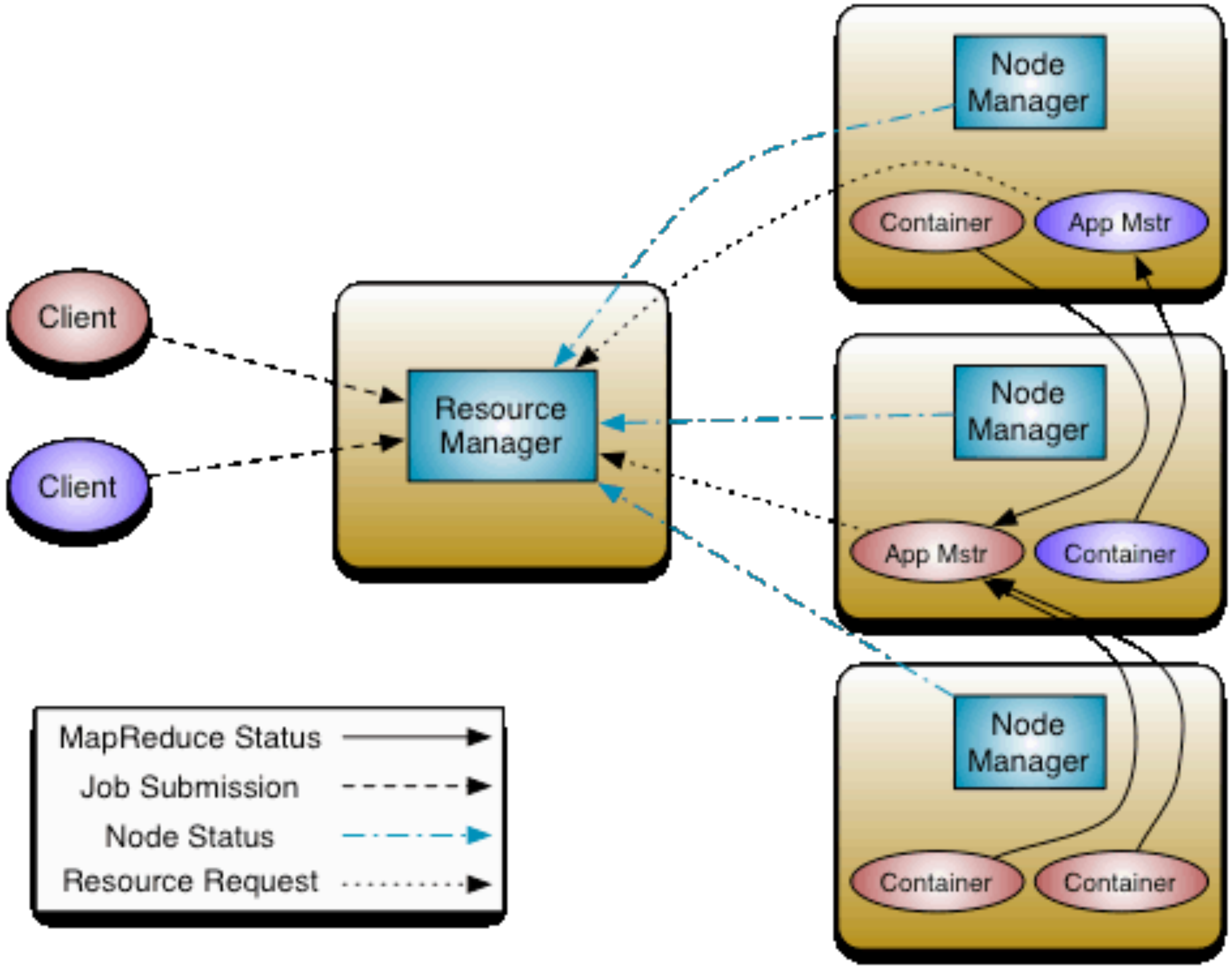
Hadoop Cluster Architecture



YARN

- Hadoop limitations:
  - Can only run MapReduce
  - What if we want to run other distributed frameworks?
- YARN = Yet-Another-Resource-Negotiator
  - Provides API to develop any generic distribution application
  - Handles scheduling and resource request
  - MapReduce (MR2) is one such application in YARN

YARN: Architecture



Today's Agenda

- Making Hadoop more efficient
- Tweaking the MapReduce programming model
- Setup for… What's beyond MapReduce?

Source: Wikipedia (Japanese rock garden)