Search Engine Architecture

11. Big Data Processing Part Two

Noted slides adapted from Lin et al.'s Big Data Infrastructure, UMD Spring 2015 with cosmetic changes.

Today's Agenda

- Making Hadoop more efficient
- Dataflow languages
- What's next?


Hadoop is slow...
Source: Wikipedia (Tortoise) via Lin et al. Big Data Infrastructure, UMD Spring 2015.

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…

Source: Blog post by DeWitt and Stonebraker via Lin et al. Big Data Infrastructure, UMD Spring 2015.

Hadoop vs. Databases: Grep

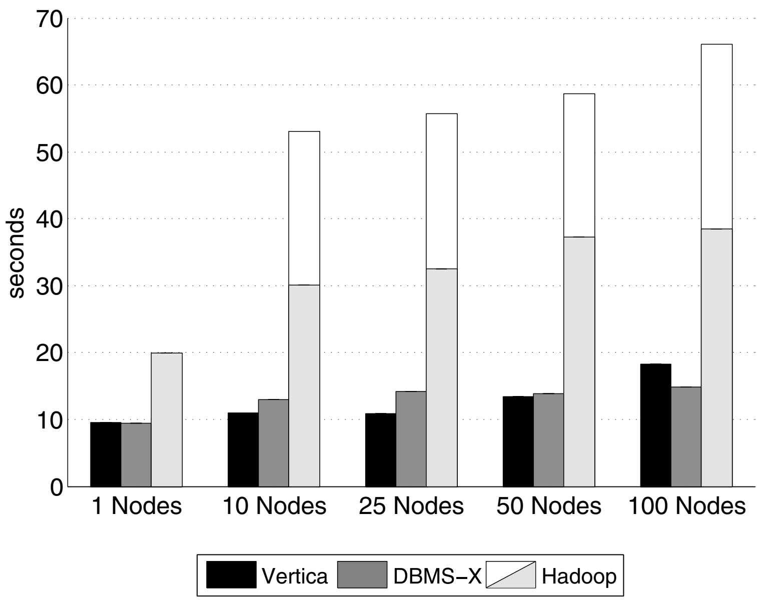


Figure 4: Grep Task Results – 535MB/node Data Set

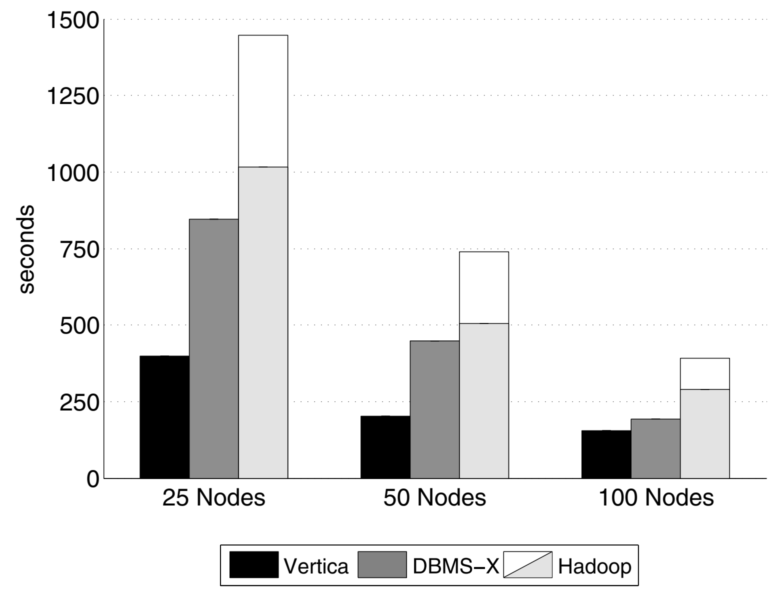


Figure 5: Grep Task Results – 1TB/cluster Data Set

```
SELECT * FROM Data WHERE field LIKE ‘%XYZ%’;
```

Source: Pavlo et al. (2009) A Comparison of Approaches to Large-Scale Data Analysis. SIGMOD via Lin et al. Big Data Infrastructure, UMD Spring 2015.

Hadoop vs. Databases: Select

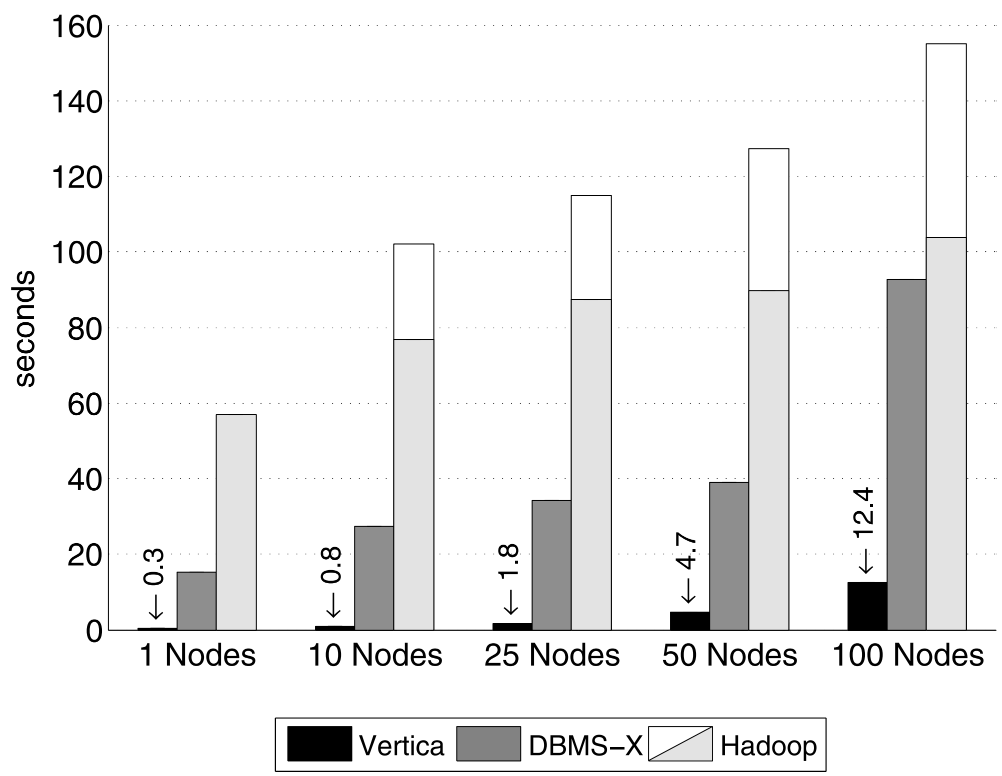


Figure 6: Selection Task Results

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

Source: Pavlo et al. (2009) A Comparison of Approaches to Large-Scale Data Analysis. SIGMOD via Lin et al. Big Data Infrastructure, UMD Spring 2015.

Hadoop vs. Databases: Aggregation

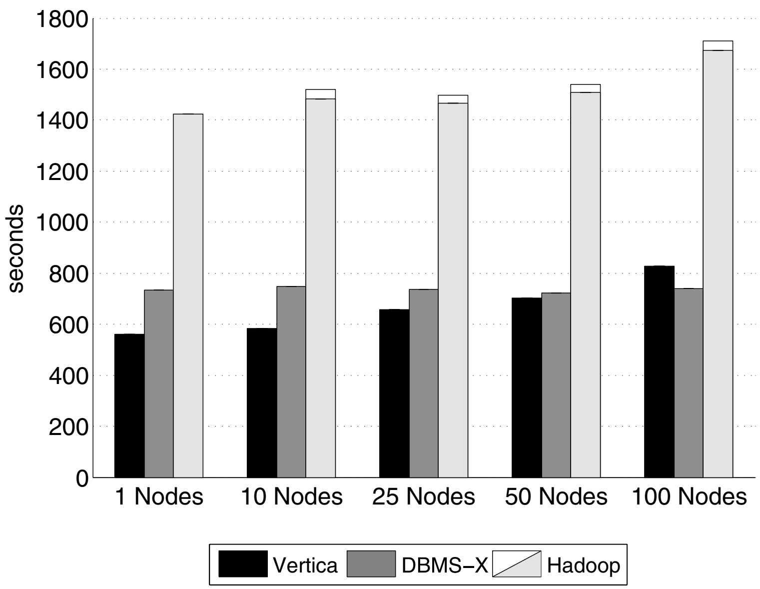


Figure 7: Aggregation Task Results (2.5 million Groups)

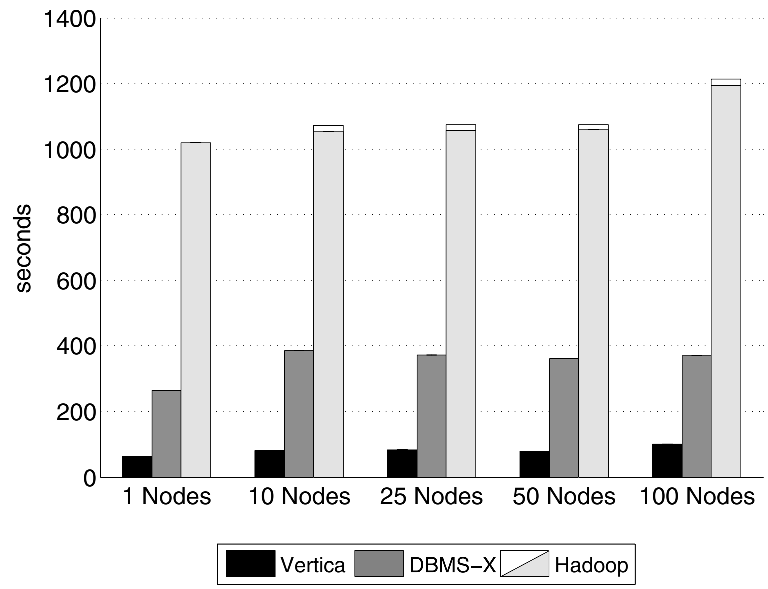


Figure 8: Aggregation Task Results (2,000 Groups)

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

Source: Pavlo et al. (2009) A Comparison of Approaches to Large-Scale Data Analysis. SIGMOD via Lin et al. Big Data Infrastructure, UMD Spring 2015.

Hadoop vs. Databases: Join

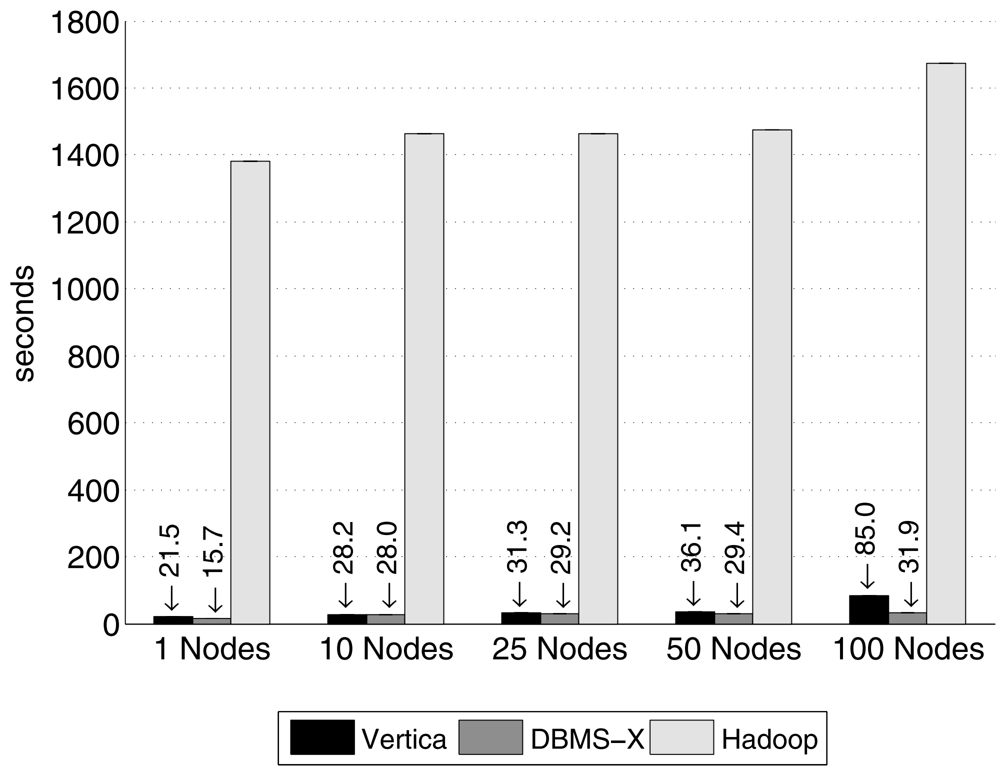


Figure 9: Join Task Results

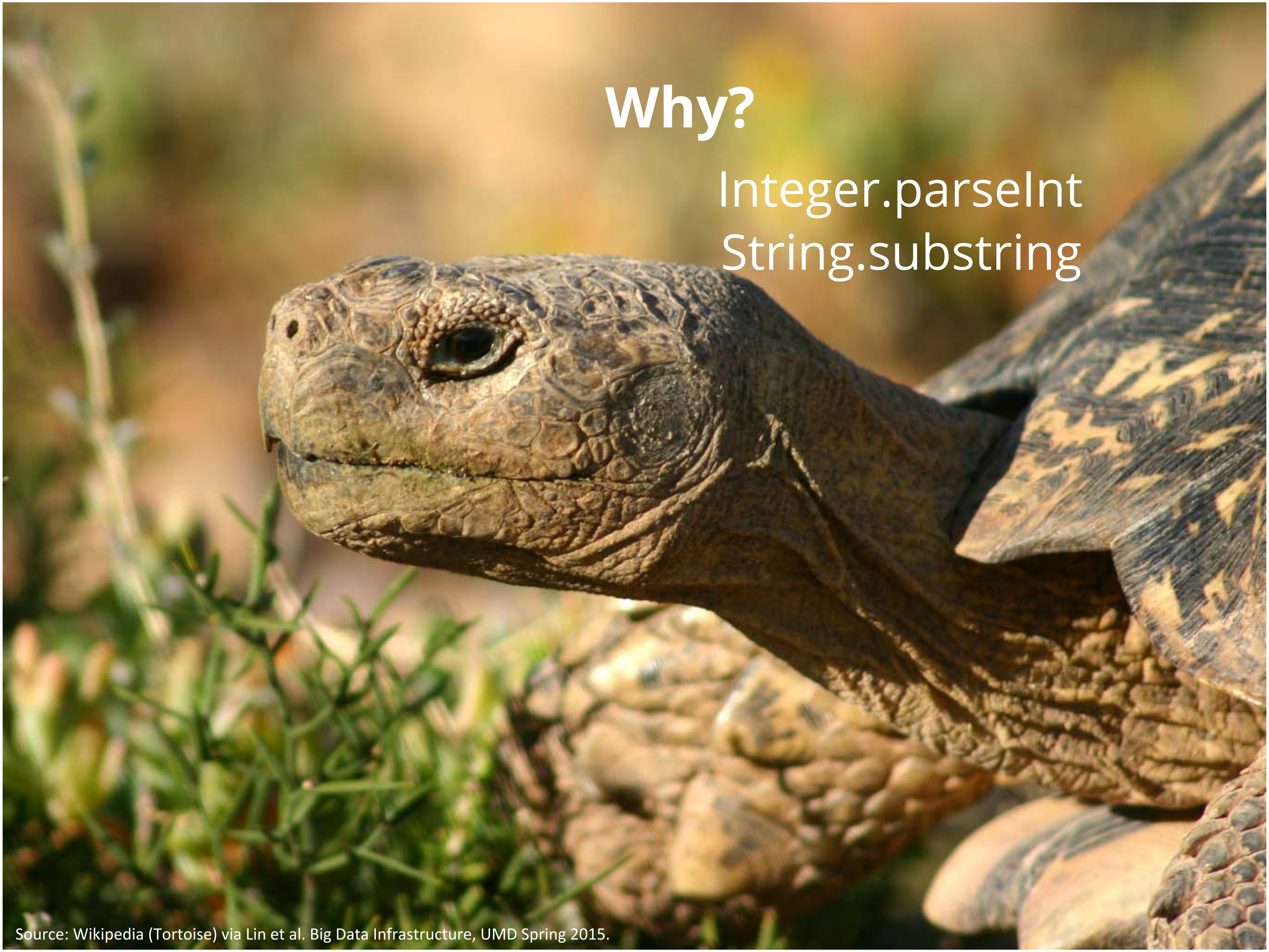
Source: Pavlo et al. (2009) A Comparison of Approaches to Large-Scale Data Analysis. SIGMOD via Lin et al. Big Data Infrastructure, UMD Spring 2015.

facebook®

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."

Source: Lin et al. Big Data Infrastructure, UMD Spring 2015.



Source: Wikipedia (Tortoise) via Lin et al. Big Data Infrastructure, UMD Spring 2015.

Schemas are a good idea!

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

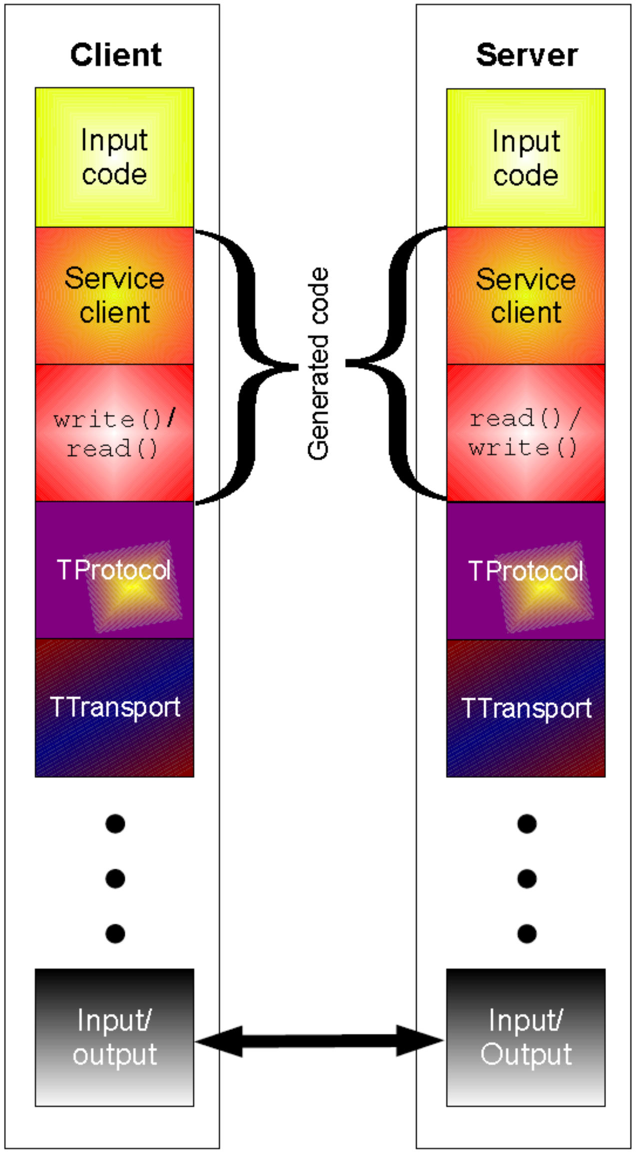
Source: Lin et al. Big Data Infrastructure, UMD Spring 2015 via Lin et al. Big Data Infrastructure, UMD Spring 2015.

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, Avro, Parquet

Source: Lin et al. Big Data Infrastructure, UMD Spring 2015 via Lin et al. Big Data Infrastructure, UMD Spring 2015.

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;
}
```

Source: Lin et al. Big Data Infrastructure, UMD Spring 2015 via Lin et al. Big Data Infrastructure, UMD Spring 2015.

Dataflow Languages

Need for High-Level Languages

- Hadoop is great for large-data processing!
  - But writing Java programs for everything is verbose and slow
  - Data scientists don’t want to write Java
- Solution: develop higher-level data processing languages
  - Hive: HQL is like SQL
  - Pig: Pig Latin is a bit like Perl

Source: Lin et al. Big Data Infrastructure, UMD Spring 2015.

Hive and Pig

- Hive: data warehousing application in Hadoop
  - Query language is HQL, variant of SQL
  - Tables stored on HDFS with different encodings
  - Developed by Facebook, now open source
- Pig: large-scale data processing system
  - Scripts are written in Pig Latin, a dataflow language
  - Programmer focuses on data transformations
  - Developed by Yahoo!, now open source
- Common idea:
  - Provide higher-level language to facilitate large-data processing
  - Higher-level language "compiles down" to Hadoop jobs

Source: Lin et al. Big Data Infrastructure, UMD Spring 2015.

Hive: Example

- Hive looks similar to an SQL database
- Relational join on two tables:
  - Table of word counts from Shakespeare collection
  - Table of word counts from the bible

```
SELECT s.word, s.freq, k.freq FROM shakespeare s
  JOIN bible k ON (s.word = k.word) WHERE s.freq >= 1 AND k.freq >= 1
  ORDER BY s.freq DESC LIMIT 10;
```

the	25848	62394
I	23031	8854
and	19671	38985
to	18038	13526
of	16700	34654
a	14170	8057
you	12702	2720
my	11297	4135
in	10797	12445
is	8882	6884

Source: Material drawn from Cloudera training VM via Lin et al. Big Data Infrastructure, UMD Spring 2015.

Hive: Behind the Scenes

```
SELECT s.word, s.freq, k.freq FROM shakespeare s
  JOIN bible k ON (s.word = k.word) WHERE s.freq >= 1 AND k.freq >= 1
  ORDER BY s.freq DESC LIMIT 10;
```

(Abstract Syntax Tree)

```
(TOK_QUERY (TOK_FROM (TOK_JOIN (TOK_TABREF shakespeare s) (TOK_TABREF bible k) (= (. (TOK_TABLE_OR_COL s) word) (.
(TOK_TABLE_OR_COL k) word)))) (TOK_INSERT (TOK_DESTINATION (TOK_DIR TOK_TMP_FILE)) (TOK_SELECT (TOK_SELEXPR (.
(TOK_TABLE_OR_COL s) word)) (TOK_SELEXPR (. (TOK_TABLE_OR_COL s) freq)) (TOK_SELEXPR (. (TOK_TABLE_OR_COL k) freq))) (TOK_WHERE
(AND (>= (. (TOK_TABLE_OR_COL s) freq) 1) (>= (. (TOK_TABLE_OR_COL k) freq) 1))) (TOK_ORDERBY (TOK_TABSORTCOLNAMEDESC (.
(TOK_TABLE_OR_COL s) freq))) (TOK_LIMIT 10)))
```

(one or more of MapReduce jobs)

Source: Lin et al. Big Data Infrastructure, UMD Spring 2015.

Hive: Behind the Scenes

```
STAGE DEPENDENCIES:
 Stage-1 is a root stage
 Stage-2 depends on stages: Stage-1
 Stage-0 is a root stage

STAGE PLANS:
 Stage: Stage-1
  Map Reduce
   Alias -> Map Operator Tree:
    s
     TableScan
      alias: s
      Filter Operator
       predicate:
          expr: (freq >= 1)
          type: boolean
       Reduce Output Operator
        key expressions:
             expr: word
             type: string
        sort order: +
        Map-reduce partition columns:
             expr: word
             type: string
        tag: 0
        value expressions:
             expr: freq
             type: int
             expr: word
             type: string
    k
     TableScan
      alias: k
      Filter Operator
       predicate:
          expr: (freq >= 1)
          type: boolean
       Reduce Output Operator
        key expressions:
             expr: word
             type: string
        sort order: +
        Map-reduce partition columns:
             expr: word
             type: string
        tag: 1
        value expressions:
             expr: freq
             type: int
```

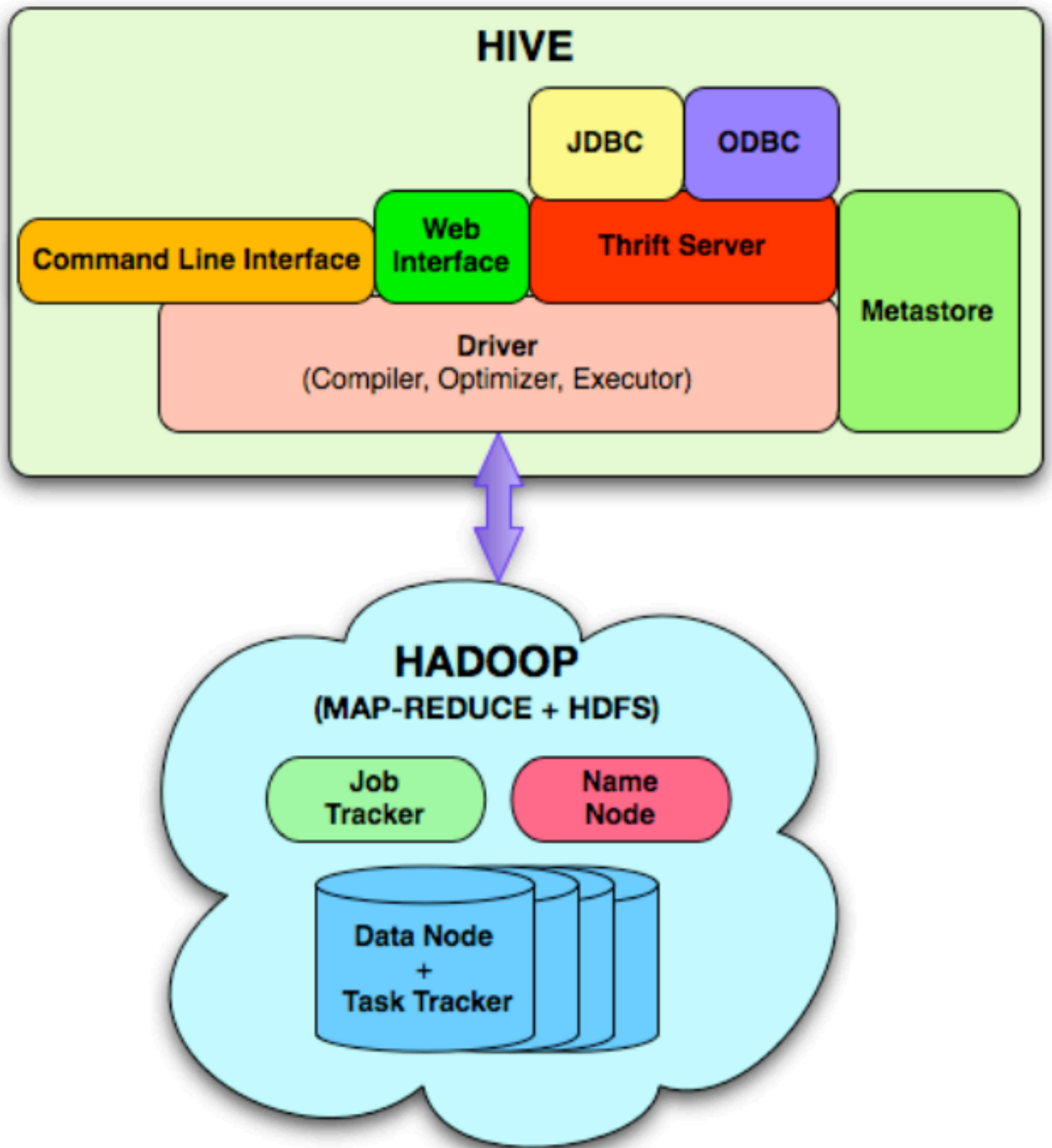
```
Reduce Operator Tree:
   Join Operator
    condition map:
       Inner Join 0 to 1
    condition expressions:
     0 {VALUE._col0} {VALUE._col1}
     1 {VALUE._col0}
    outputColumnNames: _col0, _col1, _col2
    Filter Operator
     predicate:
        expr: ((_col0 >= 1) and (_col2 >= 1))
        type: boolean
     Select Operator
      expressions:
          expr: _col1
          type: string
          expr: _col0
          type: int
          expr: _col2
          type: int
      outputColumnNames: _col0, _col1, _col2
      File Output Operator
       compressed: false
       GlobalTableId: 0
       table:
          input format: org.apache.hadoop.mapred.SequenceFileInputFormat
          output format: org.apache.hadoop.hive.ql.io.HiveSequenceFileOutputFormat
```

```
Stage: Stage-2
  Map Reduce
   Alias -> Map Operator Tree:
    hdfs://localhost:8022/tmp/hive-training/364214370/10002
      Reduce Output Operator
       key expressions:
          expr: _col1
          type: int
       sort order: -
       tag: -1
       value expressions:
          expr: _col0
          type: string
          expr: _col1
          type: int
          expr: _col2
          type: int
   Reduce Operator Tree:
    Extract
     Limit
      File Output Operator
       compressed: false
       GlobalTableId: 0
       table:
          input format: org.apache.hadoop.mapred.TextInputFormat
          output format: org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat

Stage: Stage-0
  Fetch Operator
   limit: 10
```

Source: Lin et al. Big Data Infrastructure, UMD Spring 2015.

Hive Architecture



Source: Lin et al. Big Data Infrastructure, UMD Spring 2015.

Hive Implementation

- Metastore holds metadata
  - Databases, tables
  - Schemas (field names, field types, etc.)
  - Permission information (roles and users)
- Hive data stored in HDFS
  - Tables in directories
  - Partitions of tables in sub-directories
  - Actual data in files

Source: Lin et al. Big Data Infrastructure, UMD Spring 2015.

Pig!

Source: Wikipedia (Pig) via Lin et al. Big Data Infrastructure, UMD Spring 2015.

Pig: Example

Task: Find the top 10 most visited pages in each category

Visits

User	Url	Time
Amy	cnn.com	8:00
Amy	bbc.com	10:00
Amy	flickr.com	10:05
Fred	cnn.com	12:00

Url Info

Url	Category	PageRank
cnn.com	News	0.9
bbc.com	News	0.8
flickr.com	Photos	0.7
espn.com	Sports	0.9

Pig Slides adapted from Olston et al. (SIGMOD 2008) via Lin et al. Big Data Infrastructure, UMD Spring 2015.

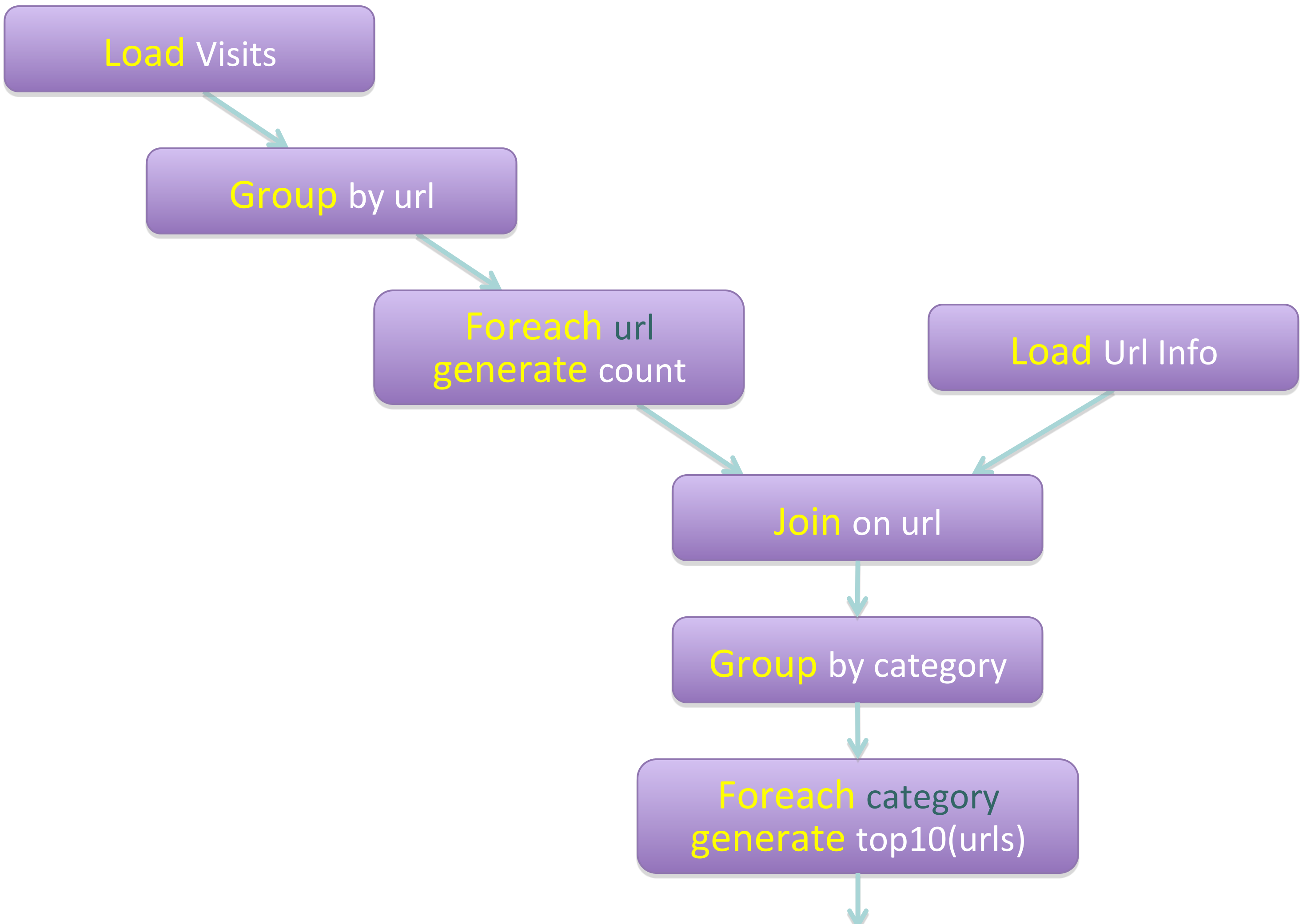
Pig Script

```
visits = load '/data/visits' as (user, url, time);
gVisits = group visits by url;
visitCounts = foreach gVisits generate url, count(visits);
urlInfo = load '/data/urlInfo' as (url, category, pRank);
visitCounts = join visitCounts by url, urlInfo by url;
gCategories = group visitCounts by category;
topUrls = foreach gCategories generate top(visitCounts,10);

store topUrls into '/data/topUrls';
```

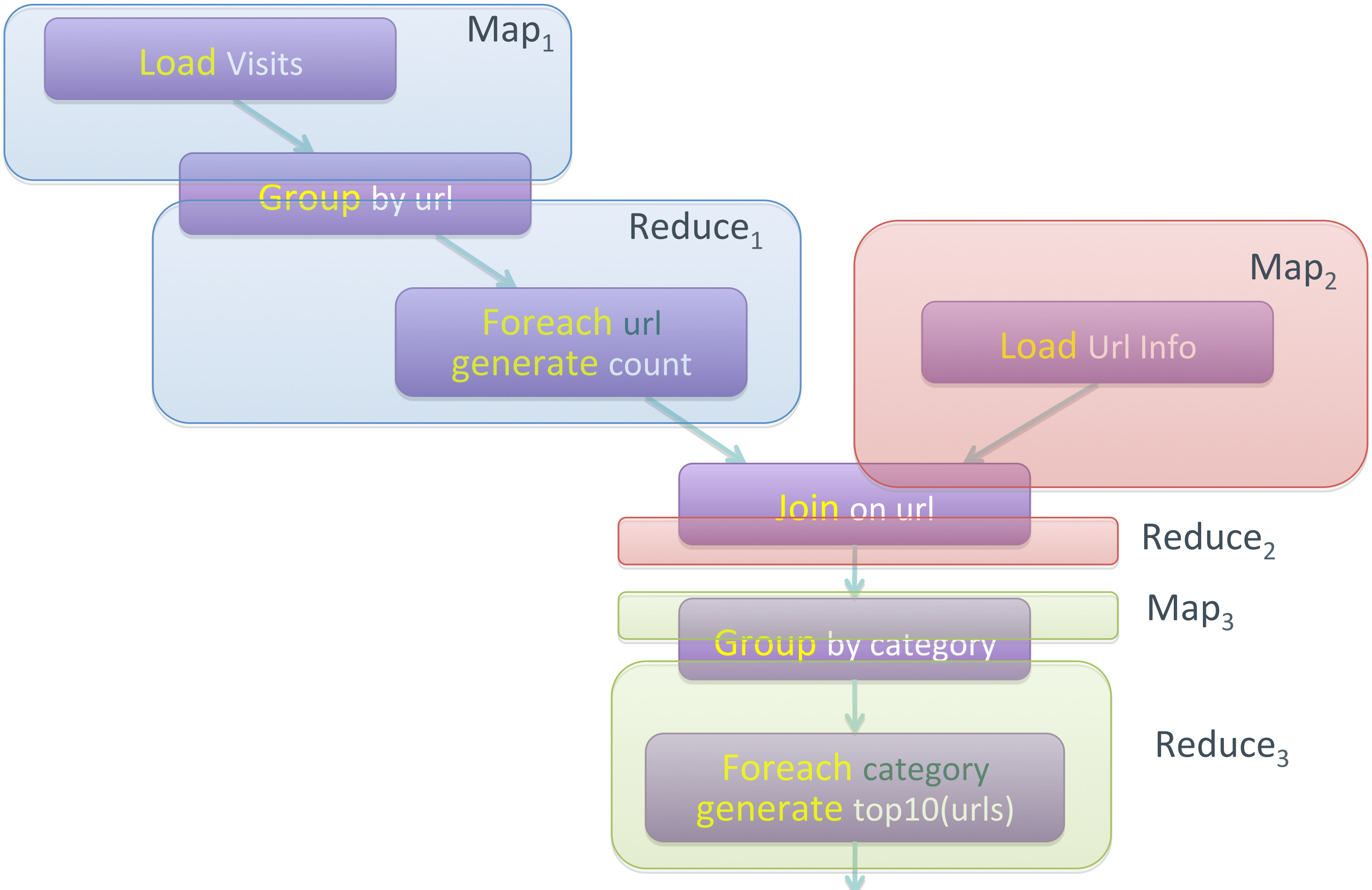
Pig Slides adapted from Olston et al. (SIGMOD 2008) via Lin et al. Big Data Infrastructure, UMD Spring 2015.

Pig Query Plan



Pig Slides adapted from Olston et al. (SIGMOD 2008) via Lin et al. Big Data Infrastructure, UMD Spring 2015.

Pig Script in Hadoop



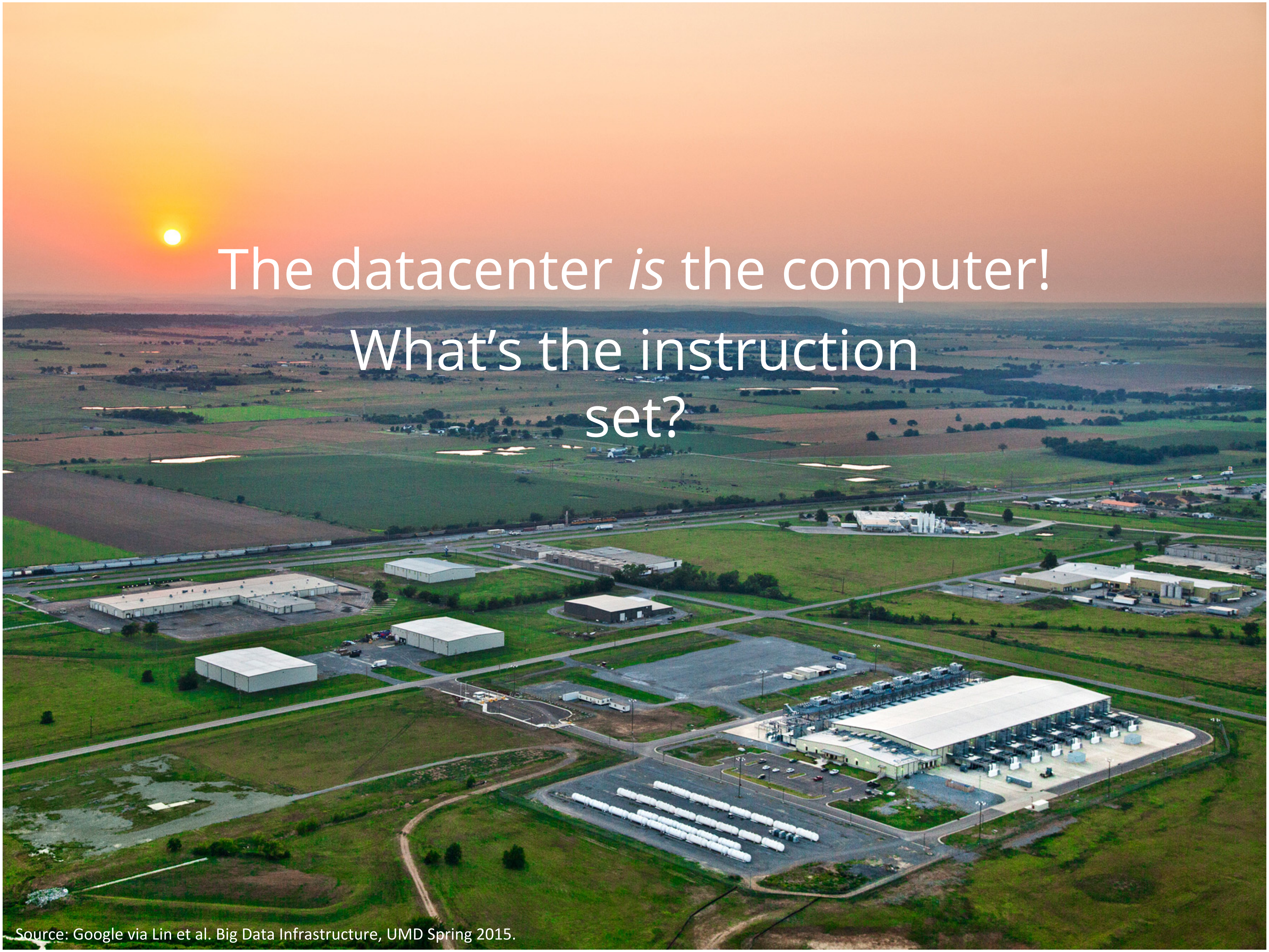
Pig Slides adapted from Olston et al. (SIGMOD 2008) via Lin et al. Big Data Infrastructure, UMD Spring 2015.

What’s next?

Source: Lin et al. Big Data Infrastructure, UMD Spring 2015.

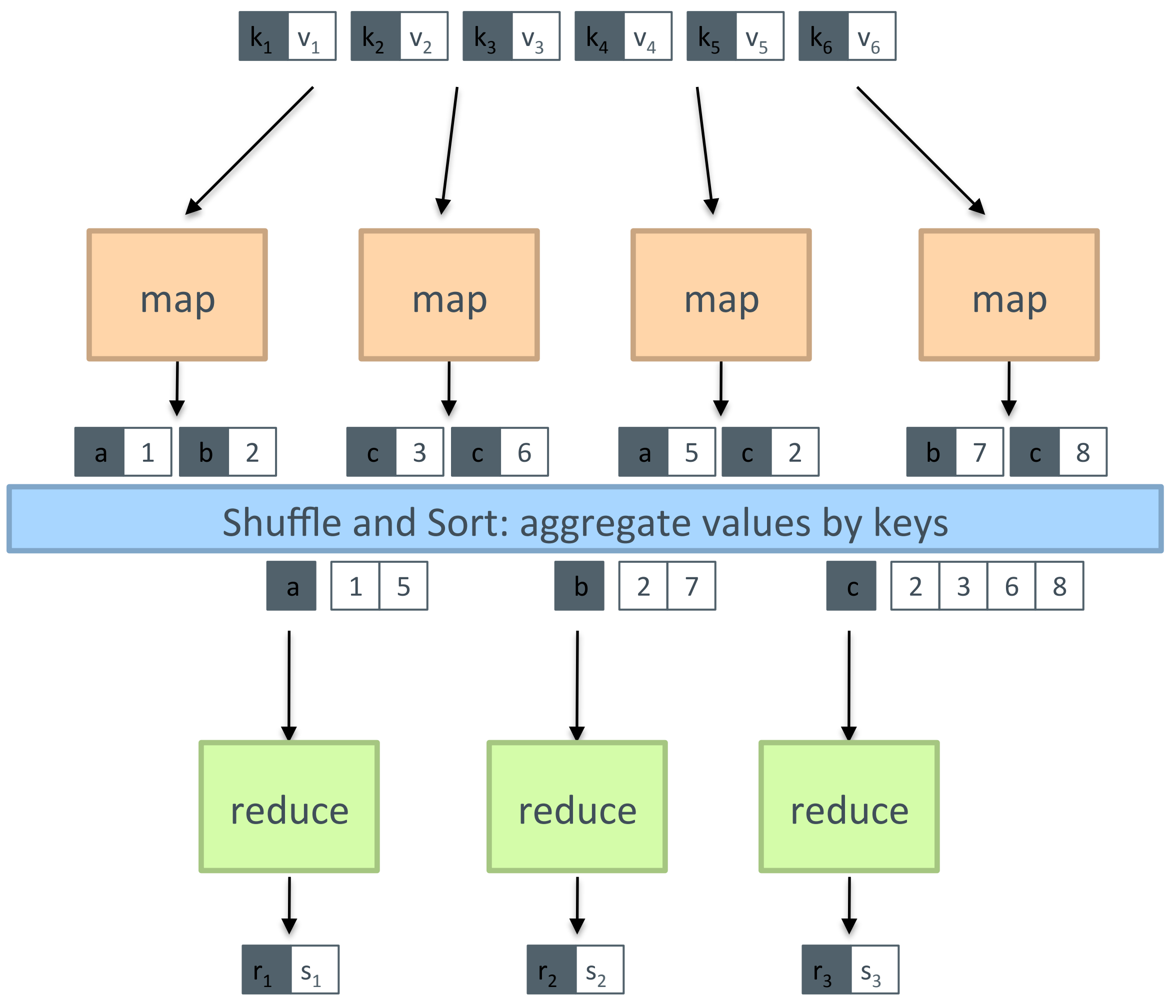
The datacenter *is* the computer!

What's the instruction set?



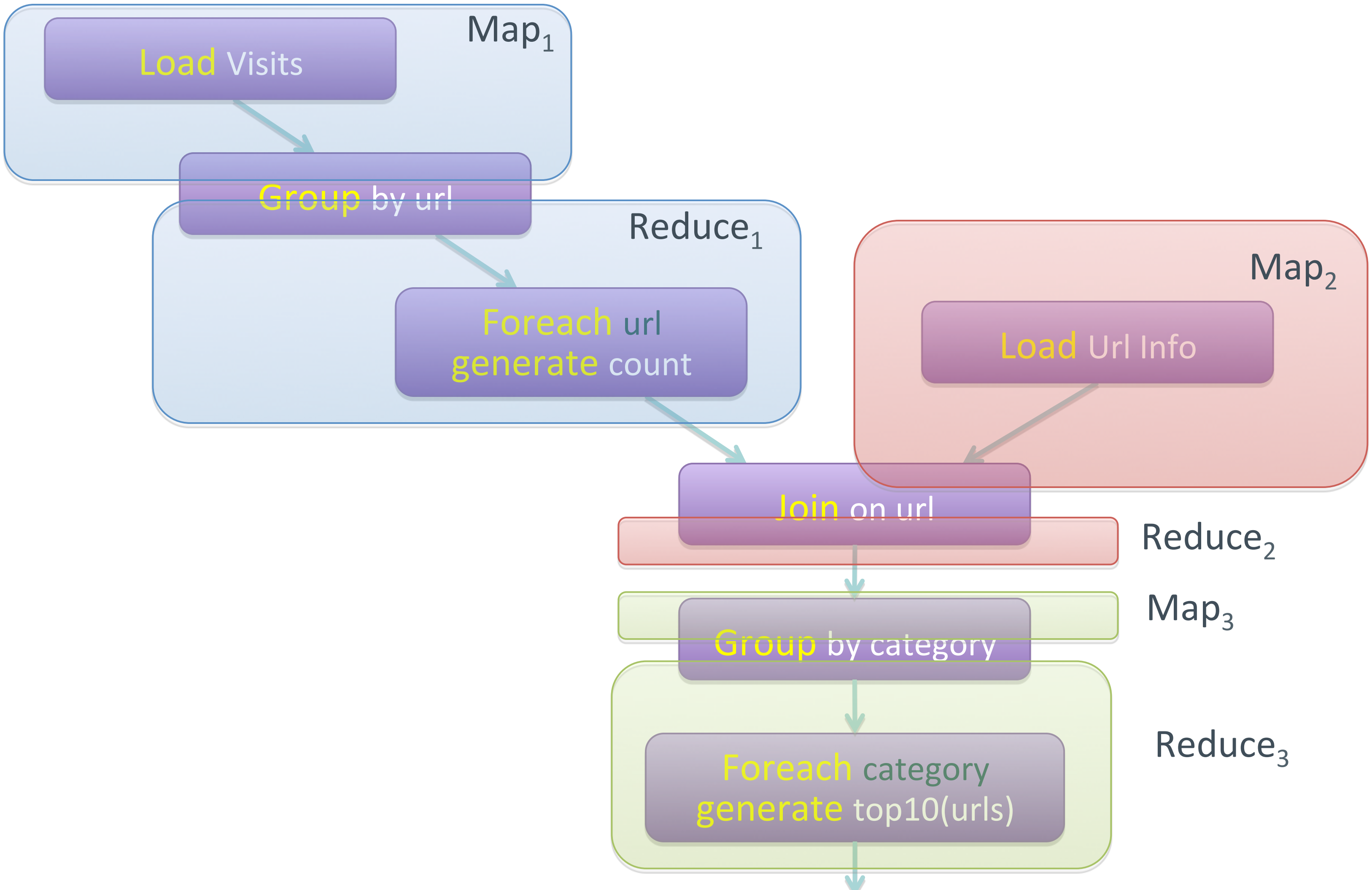
Source: Google via Lin et al. Big Data Infrastructure, UMD Spring 2015.

Answer?



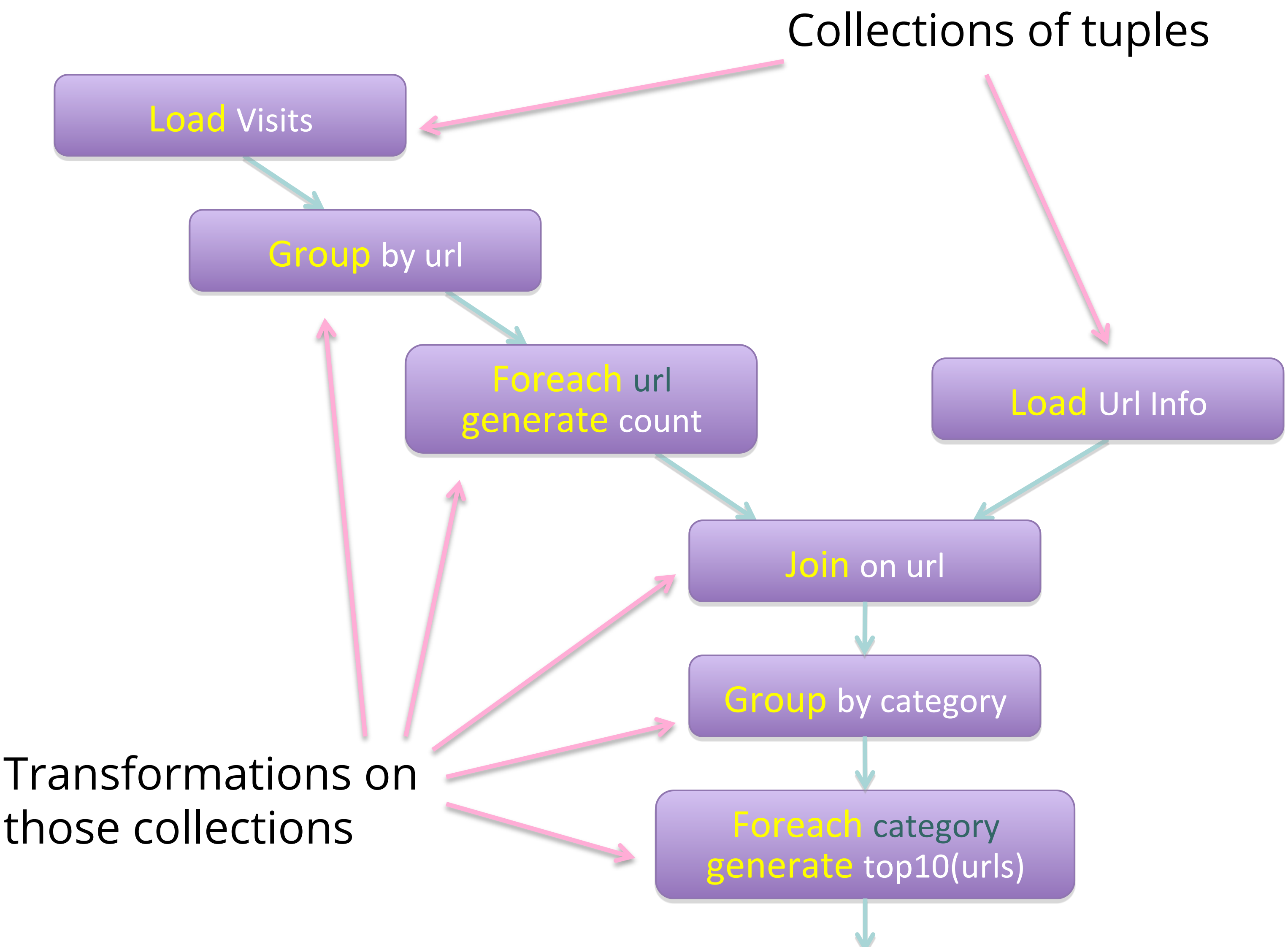
Source: Lin et al. Big Data Infrastructure, UMD Spring 2015.

Answer?



Pig Slides adapted from Olston et al. (SIGMOD 2008) via Lin et al. Big Data Infrastructure, UMD Spring 2015.

Generically, what is this?



Pig Slides adapted from Olston et al. (SIGMOD 2008) via Lin et al. Big Data Infrastructure, UMD Spring 2015.

Dataflows

- Comprised of:
  - Collections of records
  - Transformations on those collections
- Two important questions:
  - What are the logical operators?
  - What are the physical operators?

Source: Lin et al. Big Data Infrastructure, UMD Spring 2015.

Spark

- One popular answer to “What’s beyond MapReduce?”
- Open-source engine for large-scale batch processing
  - Supports generalized dataflows
  - Written in Scala, with bindings in Java, Python, R
- Brief history:
  - Developed at UC Berkeley AMPLab in 2009
  - Open-sourced in 2010
  - Became top-level Apache project in February 2014
  - Commercial support provided by DataBricks

Source: Lin et al. Big Data Infrastructure, UMD Spring 2015.

Resilient Distributed Datasets

- RDD: Spark “primitive” representing a collection of records
  - Immutable
  - Partitioned (the *D* in RDD)
- Transformations operate on an RDD to create another RDD
  - Coarse-grained manipulations only
  - RDDs keep track of *lineage*
- Persistence
  - RDDs can be materialized in memory or on disk
  - OOM or machine failures: What happens?
- Fault tolerance (the *R* in RDD):
  - RDDs can *always* be recomputed from stable storage (disk)

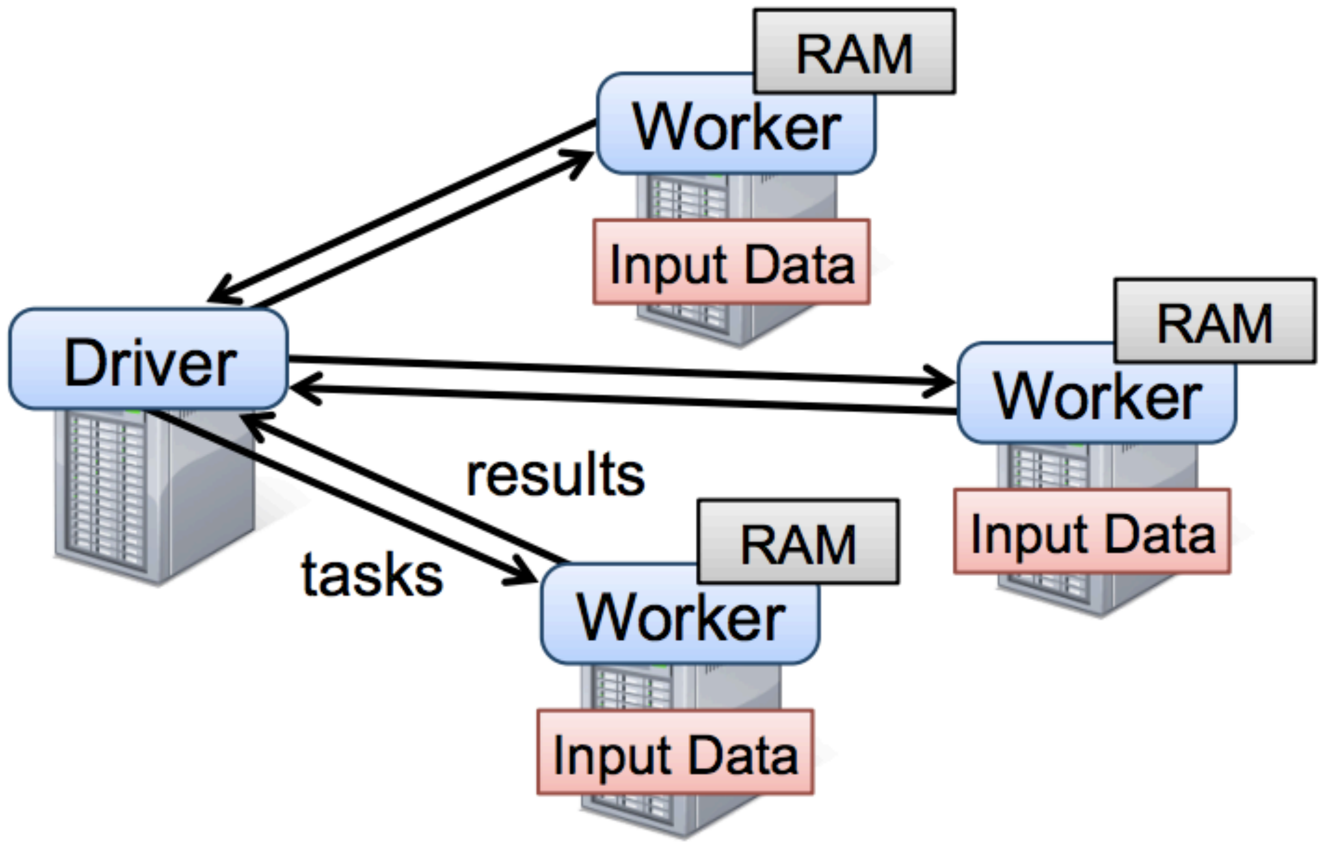
Source: Lin et al. Big Data Infrastructure, UMD Spring 2015.

Operations on RDDs

- Transformations (lazy):
  - map
  - flatMap
  - filter
  - union/intersection
  - join
  - reduceByKey
  - groupByKey
  - …
- Actions (actually trigger computations)
  - collect
  - saveAsTextFile/saveAsSequenceFile
  - …

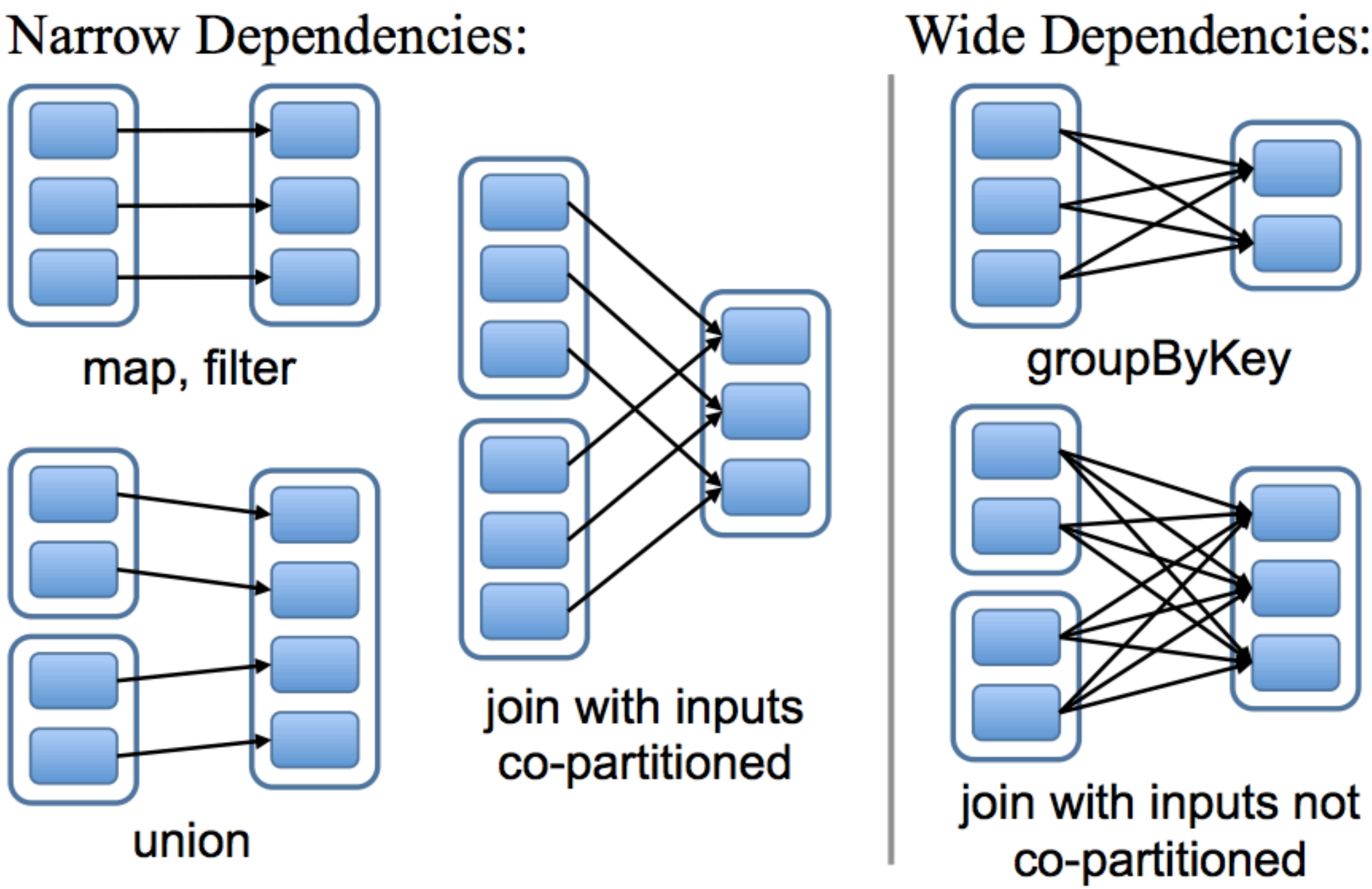
Source: Lin et al. Big Data Infrastructure, UMD Spring 2015.

Spark Architecture



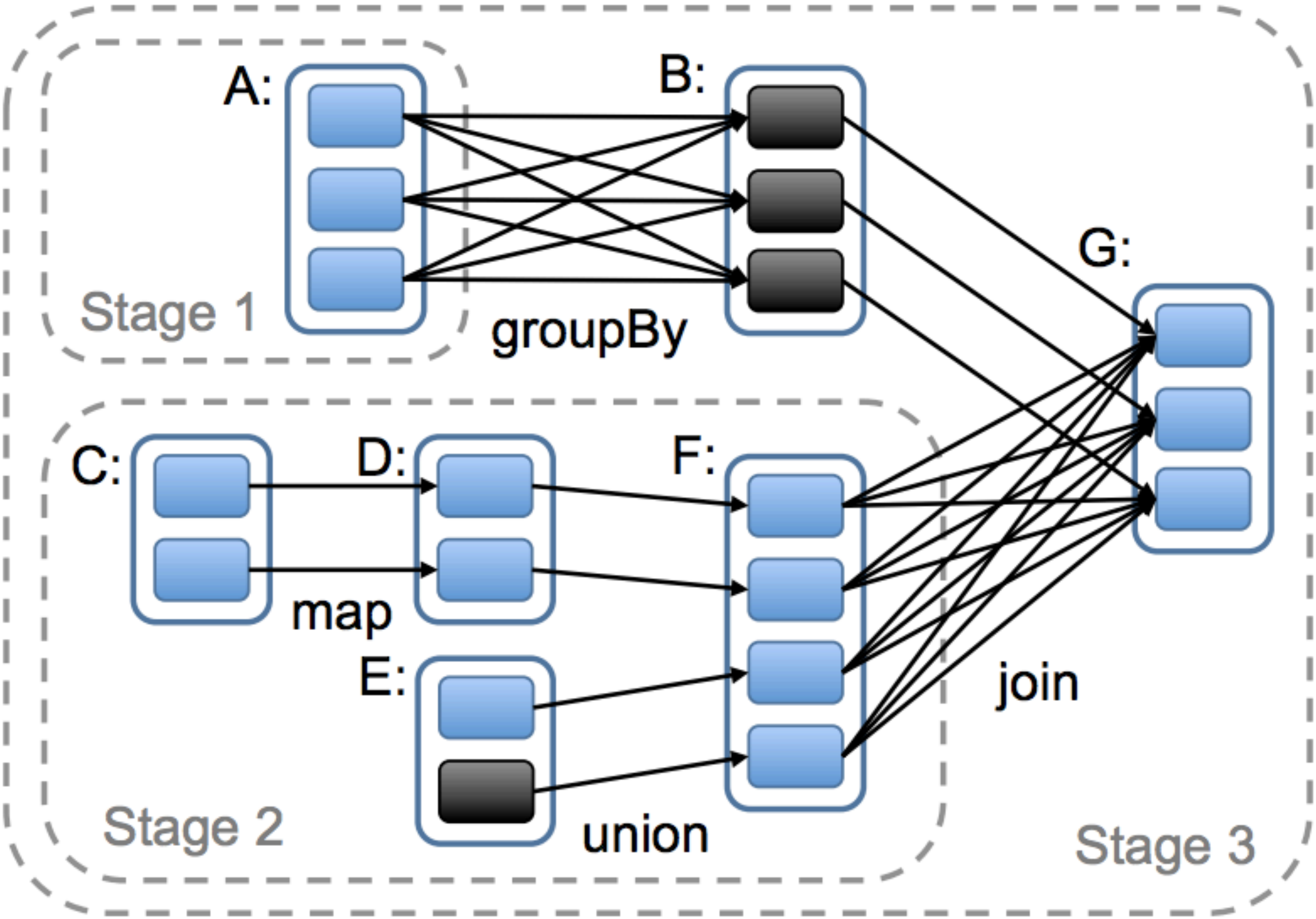
Source: Lin et al. Big Data Infrastructure, UMD Spring 2015.

Spark Physical Operators



Source: Lin et al. Big Data Infrastructure, UMD Spring 2015.

Spark Execution Plan



Source: Lin et al. Big Data Infrastructure, UMD Spring 2015.

Spark DataFrames

- The hot new RDD (built on RDDs)
  - Column-oriented, schemas, …
  - Datasets – efficient ORM
- Spark SQL
  - The hot new Shark
  - Tight integration between procedural and relational processing
  - Catalyst optimizer – “don’t bet against the compiler”
  - IndexedRDD – you can see where this is going
- GraphFrames
  - The hot new GraphX
- MLlib
  - Machine learning/fast vector math over DFs
- SparkNet …

Spark MLlib

- Machine learning/fast vector math over dataframes
- “We observe that often a simple idea is enough: separating matrix operations from vector operations and shipping the matrix operations to be run on the cluster, while keeping vector operations local to the driver.” (Zadeh et al. 2016)

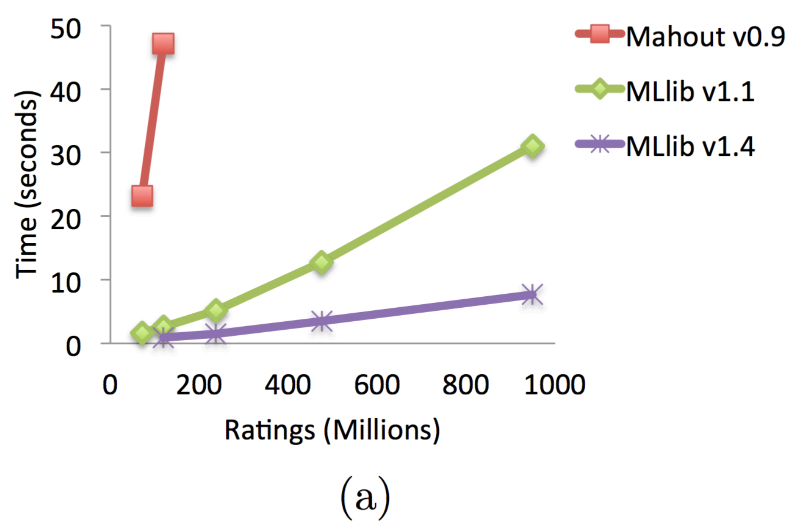


Figure 2: (a) Benchmarking results for ALS.

Questions?