Big Data Systems

Pramod Bhatotia
http://homepages.inf.ed.ac.uk/pbhatoti/

THE UNIVERSITY *of* EDINBURGH

Big Data Systems

Collect data → Process data → Provide services

E.g. Web-crawl, Click-streams, social network graph

E.g. PageRank, clustering, machine learning algorithms

E.g. search, recommendations, spam detection

Raw data → System → Information

In this course!

Big Data Systems

Data Analytics

- **Batch processing**
  - MapReduce
  - Spark
- **Stream processing**
  - Spark streaming
  - Apache Flink
- **Graph processing**
  - Pregel/Giraph
  - GraphX
- **Query processing**
  - Pig or HIVE

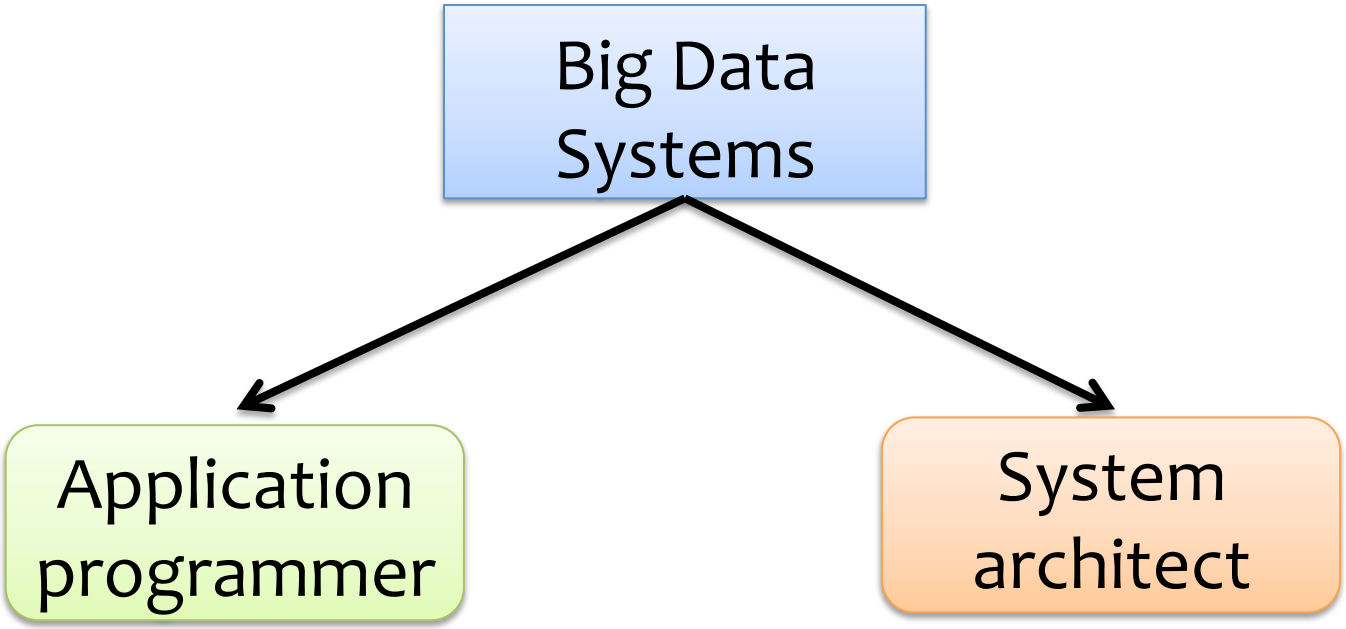
Data Management

- **File-system**
  - GFS/HDFS
- **Distributed database**
  - BigTable or Hbase
  - Spanner

Resource Management

- **Cluster manager**
  - Mesos or YARN
- **Co-ordination service**
  - ZooKeeper or Chubby

My approach


Big Data Systems
Application programmer
System architect

Course schedule

Date	Topic
Mon	• Batch processing (MapReduce) • Query processing (Apache Pig) • Graph processing (Pregel)
Tue	• Distributed file system (GFS) • Distributed structured storage (BigTable)
Wed	• Apache Spark • Stream processing (Spark streaming/Flink)
Thu	• Co-ordination services (Zookeeper) • Cluster manager (Mesos)
Fri	• End-to-end system design (Spanner)

Data-Intensive Computing with MapReduce/Pig

Pramod Bhatotia
http://homepages.inf.ed.ac.uk/pbhatoti/

Credits for the lecture material:
MapReduce OSDI'04 paper and Pig VLDB'09 paper

THE UNIVERSITY *of* EDINBURGH

How much data?

processes 20 PB a day (2008)
crawls 20B web pages a day (2012)

>10 PB data, 75B DB calls per day (6/2012)

>100 PB of user data + 500 TB/day (8/2012)

S3: 449B objects, peak 290k request/second (7/2011)
1T objects (6/2012)

Distributed Systems!

Data-center

Cluster of 100s of thousands of machines

In today's class

How to easily write parallel applications on distributed computing systems?

1. Batch processing framework
   - MapReduce
2. Query processing
   - Pig: A high-level language built on top of MapReduce

Design challenges

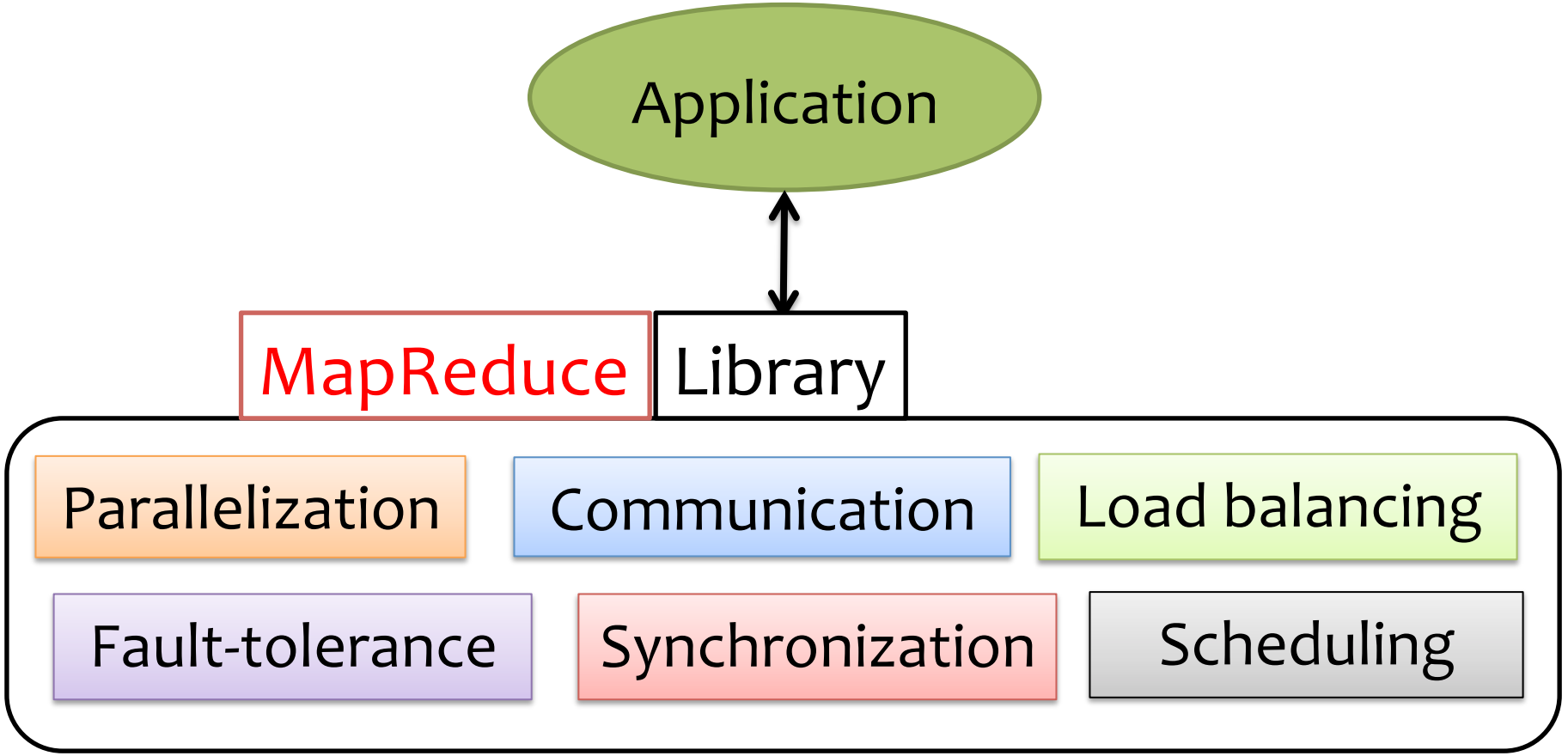
- How to parallelize application logic?
- How to communicate?
- How to synchronize?
- How to perform load balancing?
- How to handle faults?
- How to schedule jobs?

For each and every application!

Design | Implement | Optimize

Debug | Maintain

The power of abstraction



MapReduce

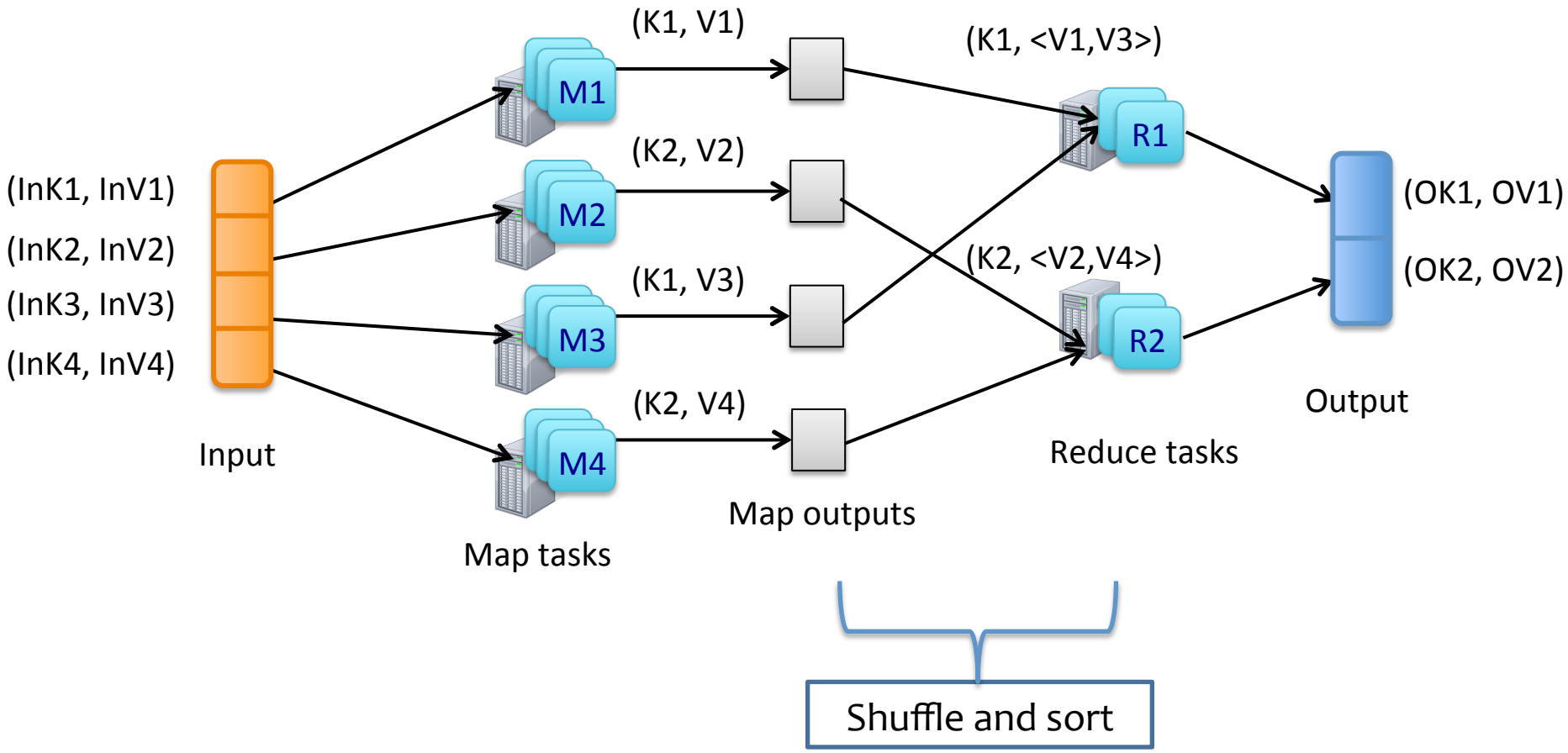
- Programming model
  - Programmer writes two methods: Map & Reduce
- Run-time library
  - Takes care of everything else!

MapReduce programming model

- Inspired from functional programming
  - Data-parallel application logic

- Programmer's interface:
  - Map(key, value) → (key, value)
  - Reduce(key, <value>) → (key,value)

MapReduce run-time system



An example: word-count

- Input:
  - Given a corpus of documents, such as Wikipedia
- Output:
  - Count the frequency of each distinct word

MapReduce for word-count

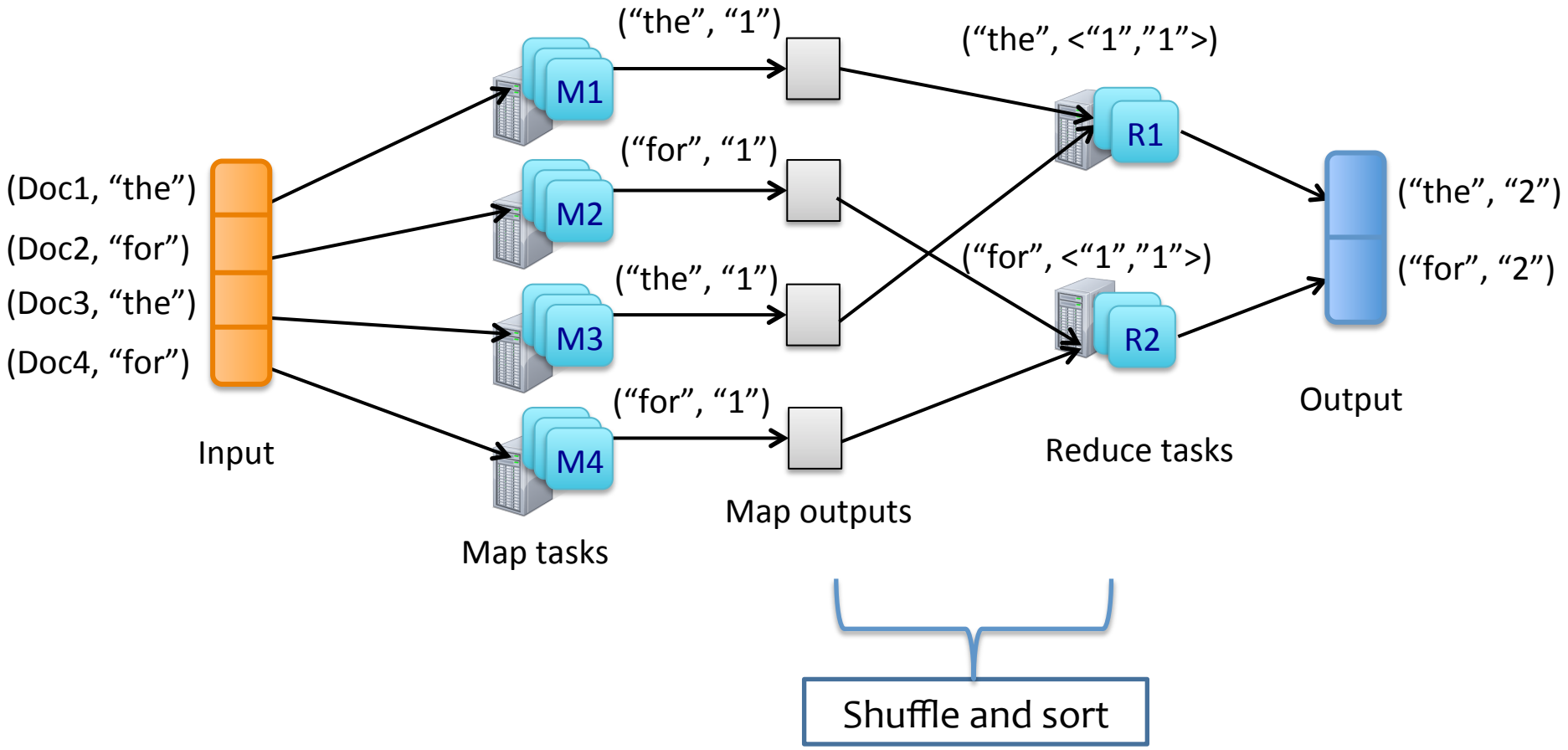
- map(string key, string value)

```
//key: document name
//value: document contents
for each word w in value
   EmitIntermediate(w, “1”);
```

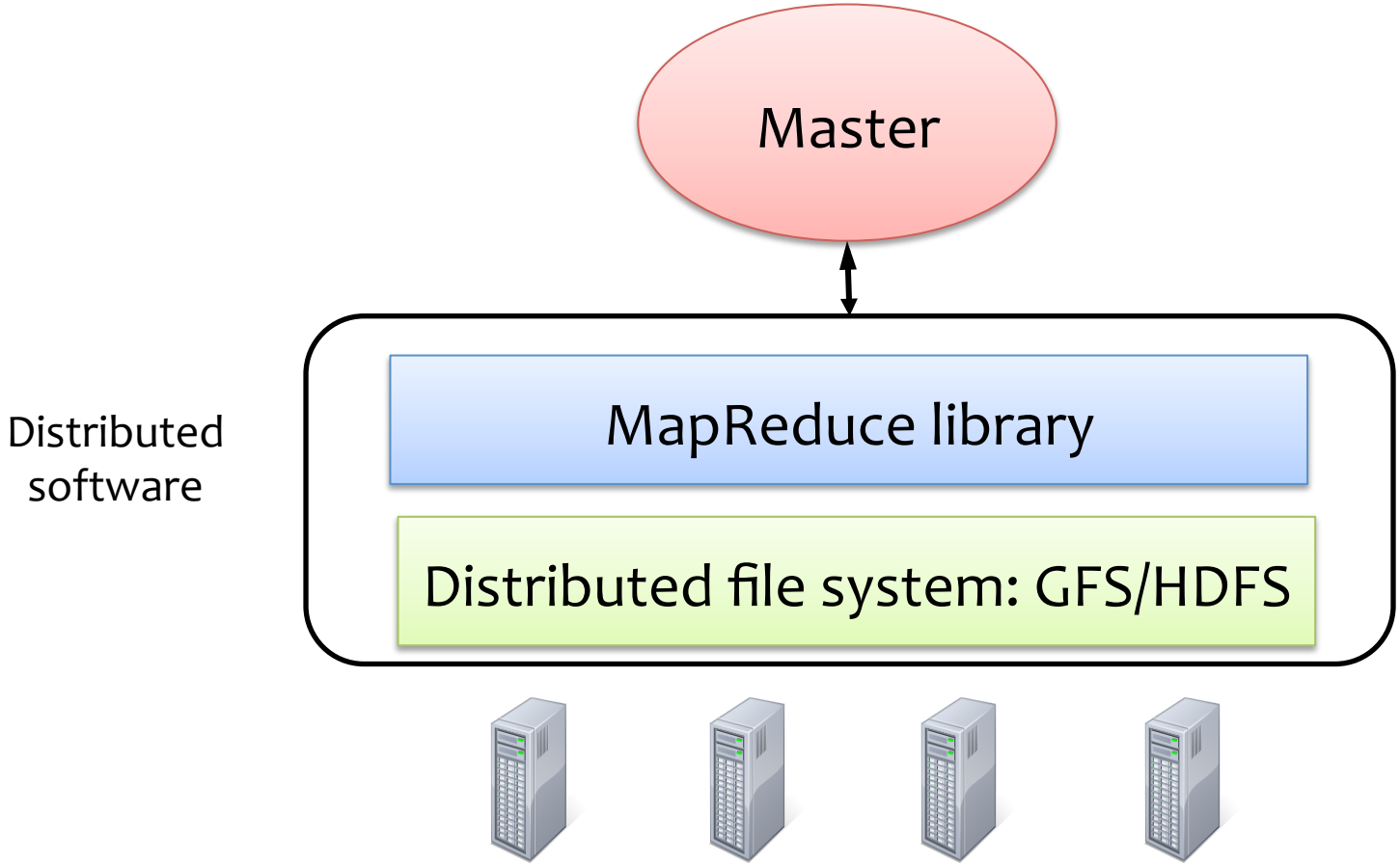
- reduce(string key, iterator values)

```
//key: word
//values: list of counts
int results = 0;
for each v in values
   result += ParseInt(v);
Emit(key, AsString(result));
```

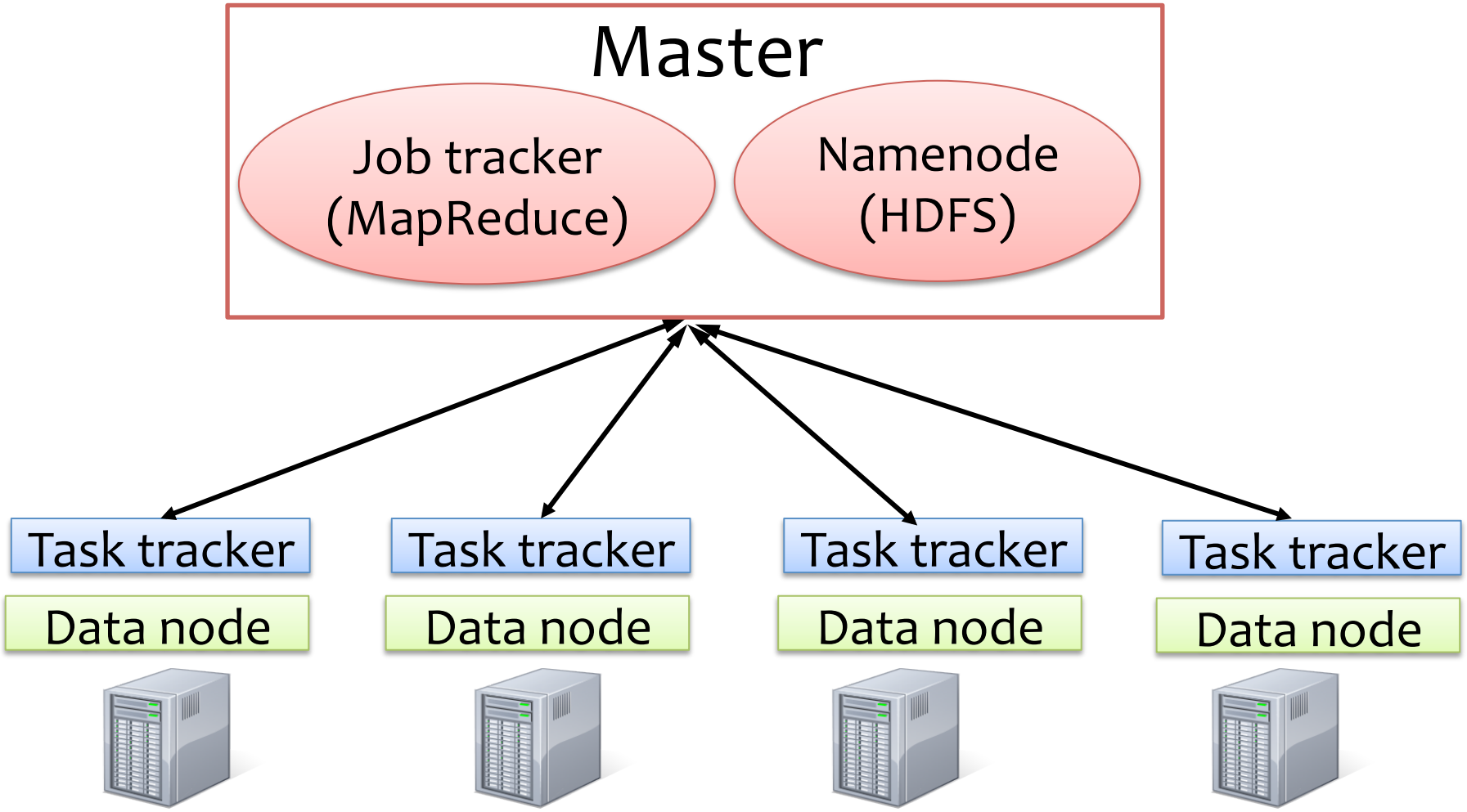
Word-count example



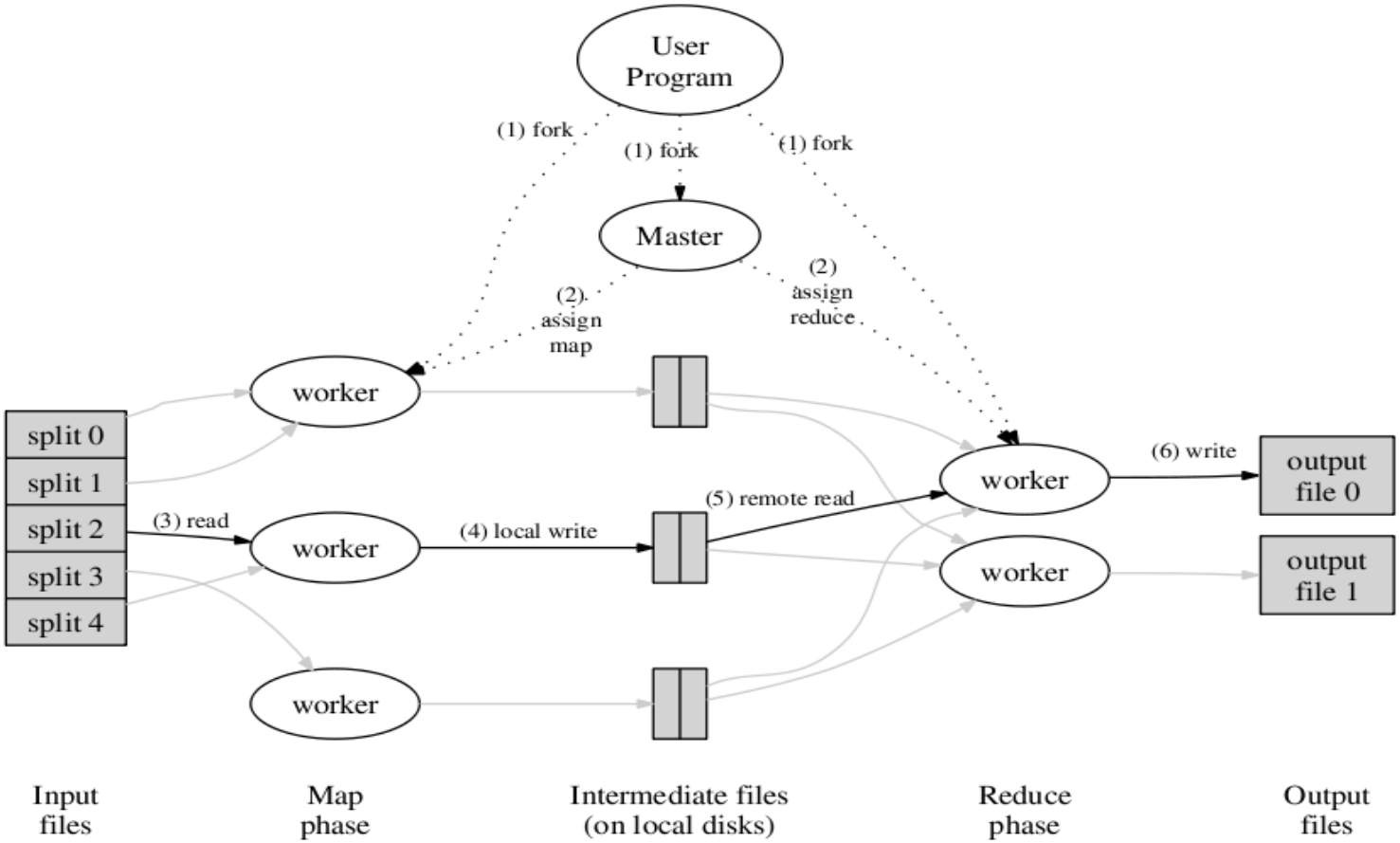
MapReduce software stack



MapReduce software stack



Runtime execution



Design challenges revisited

- Parallelization
- Communication
- Synchronization
- Load balancing
- Faults & semantics
- Scheduling

References

- MapReduce [OSDI'04] and YARN [SoCC'13]
  - Original M/R, and the next generation of M/R
- Dryad [EuroSys'07]
  - Generalized framework for data-parallel computations
- Spark [NSDI'12]
  - In-memory distributed data parallel computing

Limitations of MapReduce

- Graph algorithms
  - Pregel [SIGMOD '10], GraphX [OSDI'14]
- Iterative algorithms
  - Haloop [VLDB'10], CIEL [NSDI '11]
- Stream processing – Low latency
  - D-stream [SOSP'13], Naiad [SOSP'13], Storm, S4
- Low-level abstraction for common data analysis tasks!
  - Pig [SIGMOD'10], Shark [SIGMOD'13], DryadLINQ [OSDI'08]

Motivation for Pig

Programmers are lazy!

(they don't even wish to write Map and Reduce)

Data analysis tasks

- Common operations:
  - Filter, join, group-by, sort, etc.
- MapReduce offers a low-level primitive
  - Requires repeated re-implementation of these operators
- The power of abstraction!
  - Design once and reuse

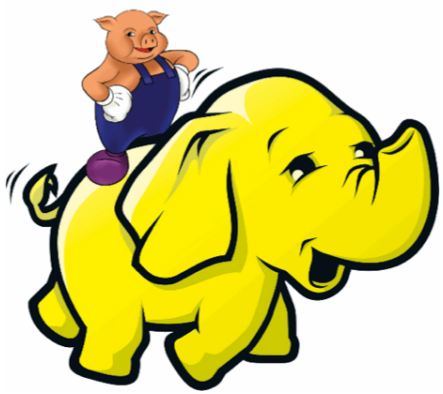
Pig Latin

Distributed dataflow queries

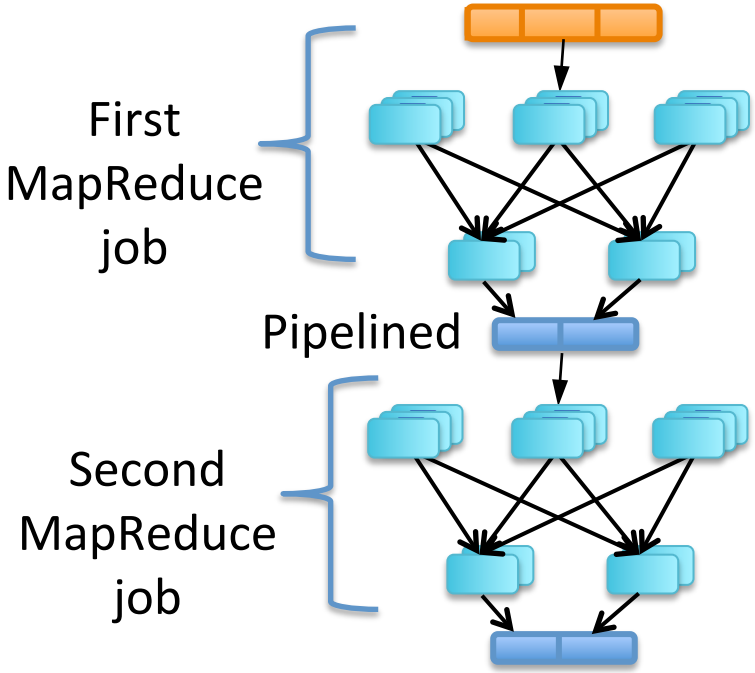
Pig Latin = SQL-kind queries + Distributed execution

Pig architecture

Pig Latin script

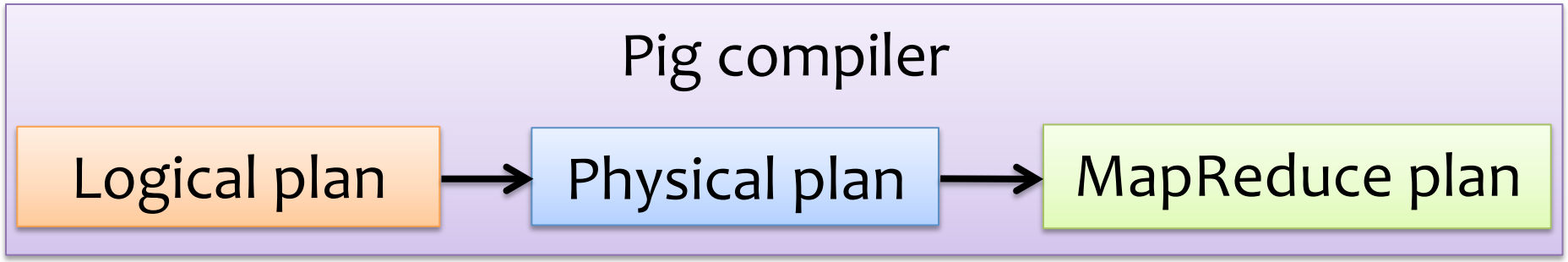


Pig compiler



MapReduce Runtime

Overview of the compilation process

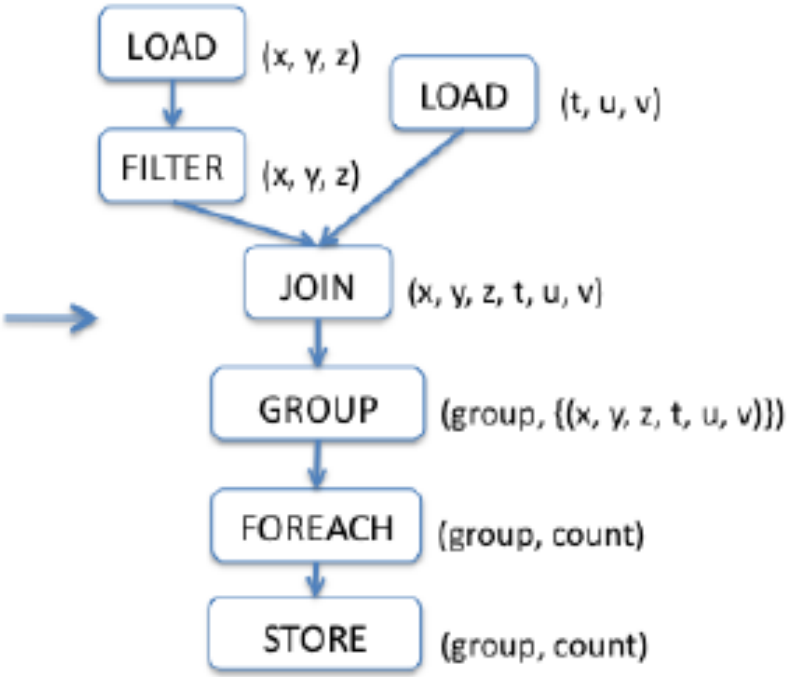


An example

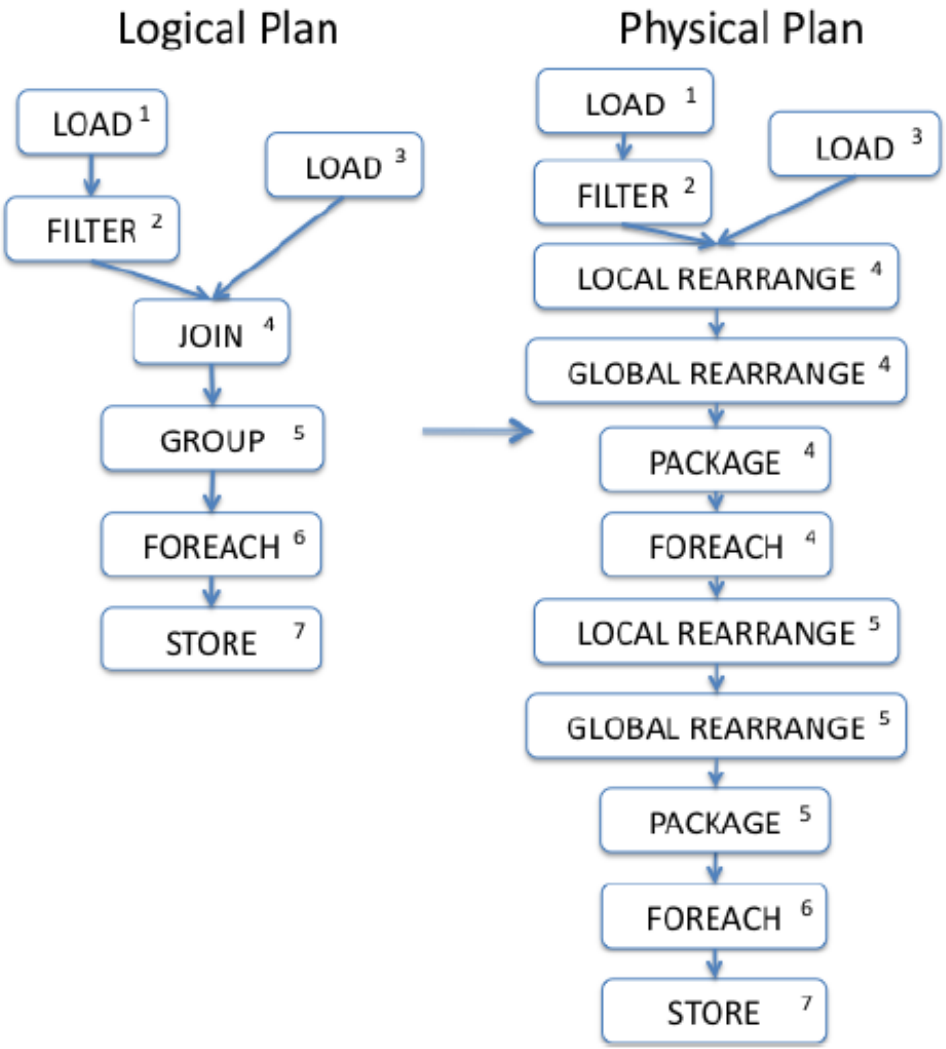
Pig Latin

```
A = LOAD 'file1' AS (x, y, z);
B = LOAD 'file2' AS (t, u, v);
C = FILTER A by y > 0;
D = JOIN C BY x, B BY u;
E = GROUP D BY z;
F = FOREACH E GENERATE
    group, COUNT(D);
STORE F INTO 'output';
```

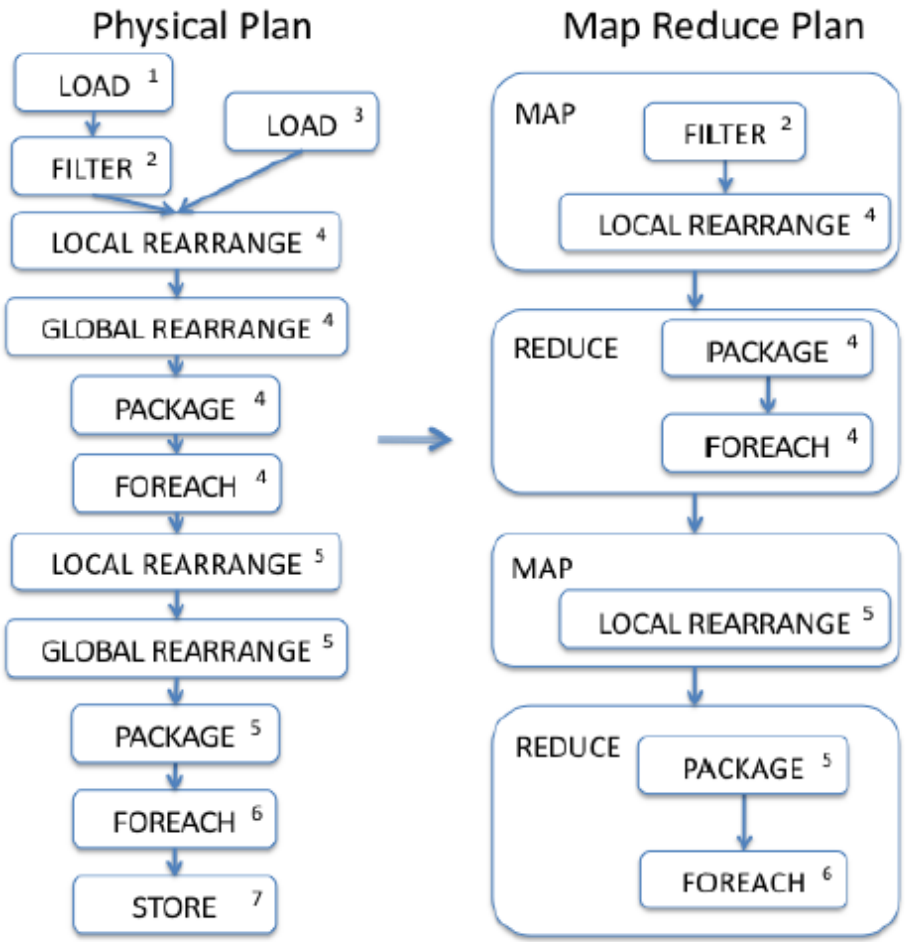
Logical Plan



Example: contd.



Example: contd.



Advantages of staged-compilation

- SQL query optimizations
- MapReduce specific optimizations

Refer Pig papers for details [SIGMOD '08, VLDB'09]

Related systems

- Apache HIVE
  - Built on top of MapReduce
- DryadLINQ [OSDI'08] or SCOPE [VLDB'08]
  - Built on top of Dryad
- Shark [SIGMOD'13]
  - Built on top of Spark

Summary

- Data-intensive computing with MapReduce
  - Data-parallel programming model
  - Runtime library to handle all low-level details
  - Pig: high-level abstraction for common tasks

- Resources:
  - Hadoop: http://hadoop.apache.org/
  - Spark: https://spark.apache.org/
  - Dryad: http://research.microsoft.com/en-us/projects/dryad/

Thanks!

http://homepages.inf.ed.ac.uk/pbhatoti/