Inspire…Educate…Transform.

Engineering Big Data

Processing Frameworks on Clusters: Map-Reduce

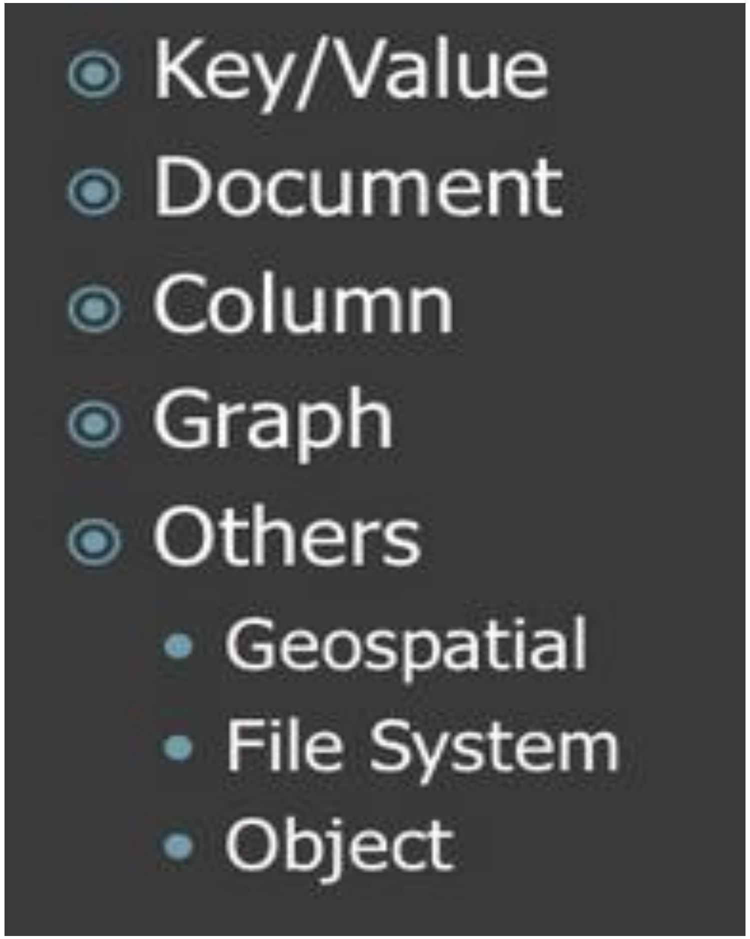
Dr. Sreerama KV Murthy
CEO, Quadratyx

August 8, 2015

Wake-Up Quiz

NO-SQL (CONTINUED)

NoSQL Taxonomy



- Key/Value
- Document
- Column
- Graph
- Others
  - Geospatial
  - File System
  - Object

Key-Value NoSQL Stores

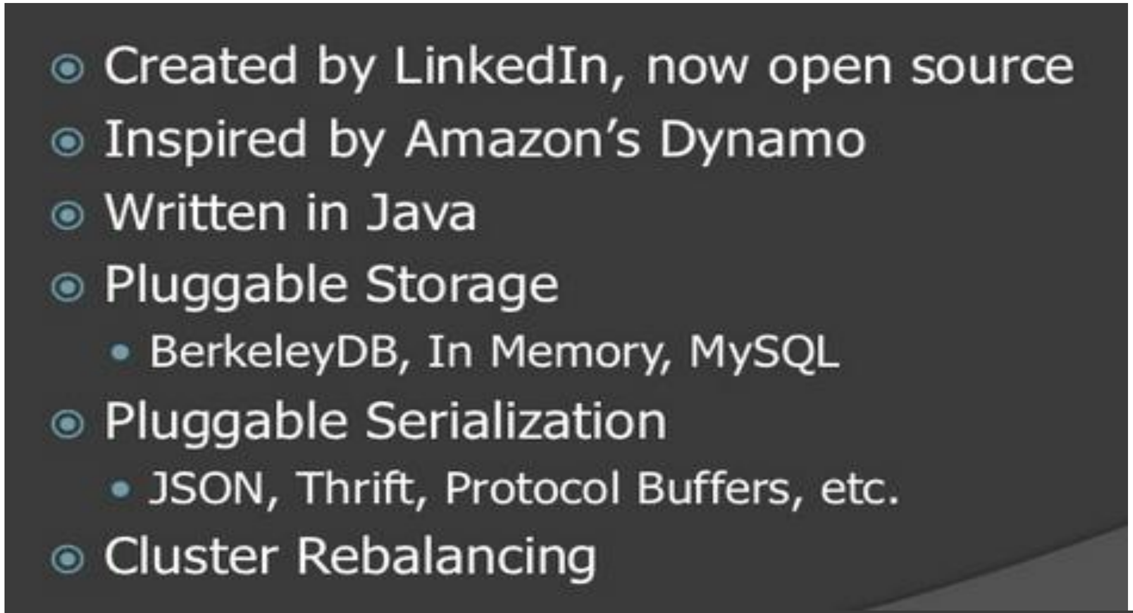
Extremely simple interface

- Data model: (key, value) pairs
- Operations: Insert(key,value), Fetch(key), Update(key), Delete(key)
- Some allow (non-uniform) columns within value
- Some allow Fetch on range of keys

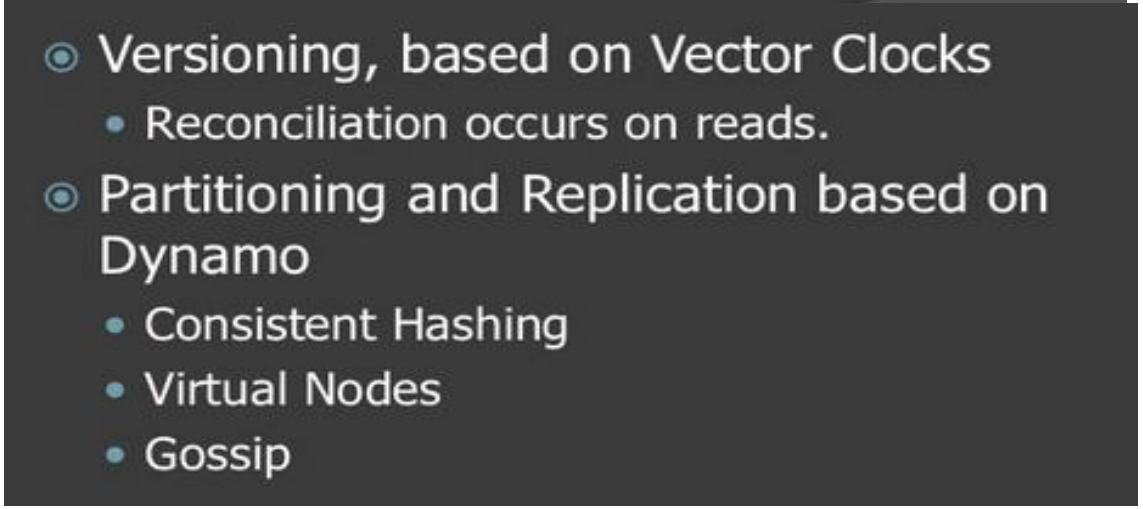
Implementation: efficiency, scalability, fault-tolerance

- Records distributed to nodes based on key
- Replication
- Single-record transactions, “eventual consistency”

Key-Value NoSQL: Voldemart



- Created by LinkedIn, now open source
- Inspired by Amazon’s Dynamo
- Written in Java
- Pluggable Storage
  - BerkeleyDB, In Memory, MySQL
- Pluggable Serialization
  - JSON, Thrift, Protocol Buffers, etc.
- Cluster Rebalancing



- Versioning, based on Vector Clocks
  - Reconciliation occurs on reads.
- Partitioning and Replication based on Dynamo
  - Consistent Hashing
  - Virtual Nodes
  - Gossip

Document Stores

Like Key-Value Stores except value is document

- Data model: (key, document) pairs
- Document: JSON, XML, other semistructured formats
- Basic operations: Insert(key,document), Fetch(key), Update(key), Delete(key)
- Also Fetch based on document contents

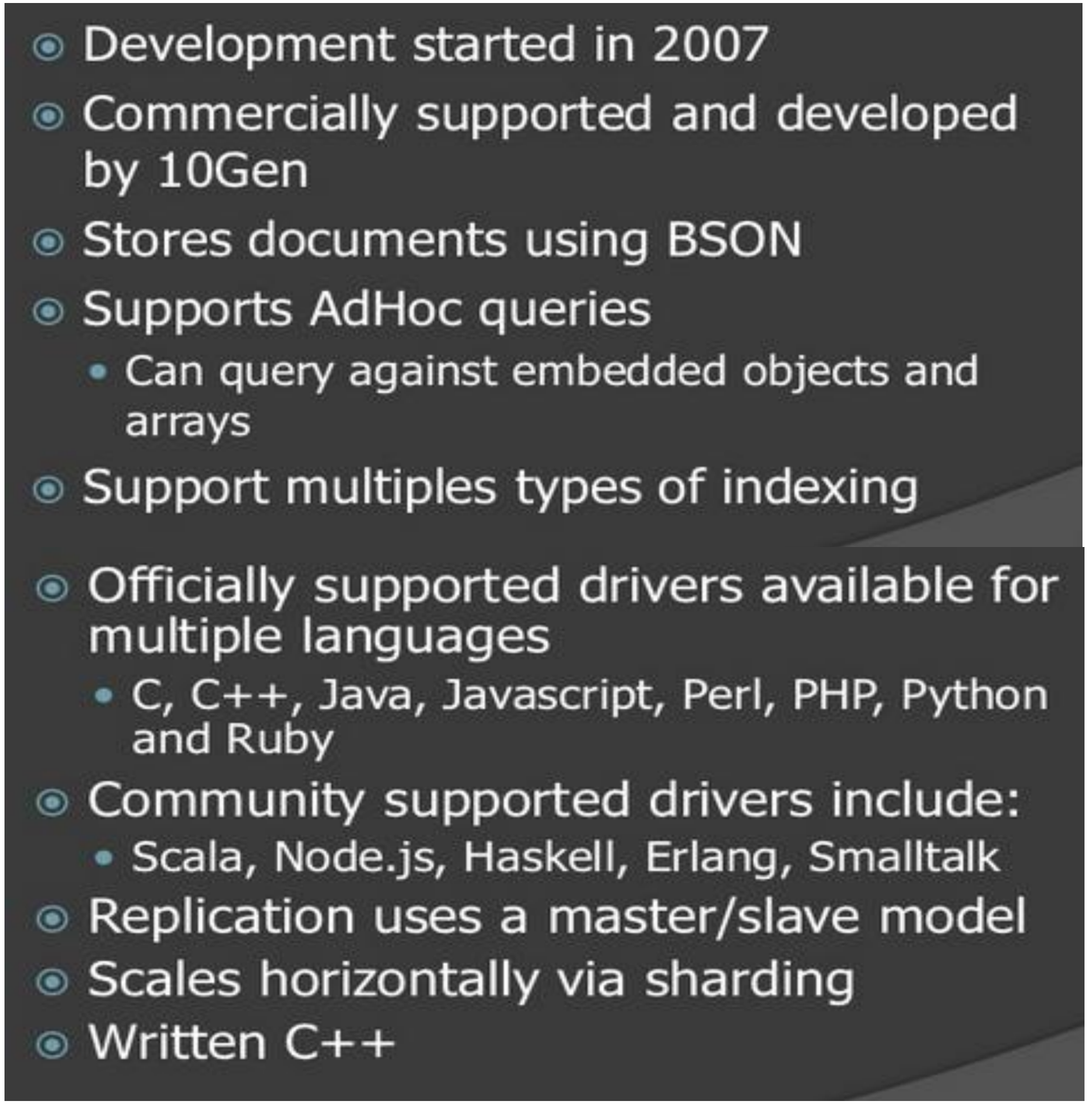
Example systems

- CouchDB, MongoDB, SimpleDB, …

CouchDB JSON Example

```
{
  "_id": "guid goes here",
  "_rev": "314159",

  "type": "abstract",

  "author": "Keith W. Hare"

  "title": "SQL Standard and NoSQL Databases",

  "body": "NoSQL databases (either no-SQL or Not Only SQL)
           are currently a hot topic in some parts of
           computing.",
  "creation_timestamp": "2011/05/10 13:30:00 +0004"
}
```

Document NoSQL: MongoDB



- Development started in 2007
- Commercially supported and developed by 10Gen
- Stores documents using BSON
- Supports AdHoc queries
  - Can query against embedded objects and arrays
- Support multiples types of indexing

- Officially supported drivers available for multiple languages
  - C, C++, Java, Javascript, Perl, PHP, Python and Ruby
- Community supported drivers include:
  - Scala, Node.js, Haskell, Erlang, Smalltalk
- Replication uses a master/slave model
- Scales horizontally via sharding
- Written C++

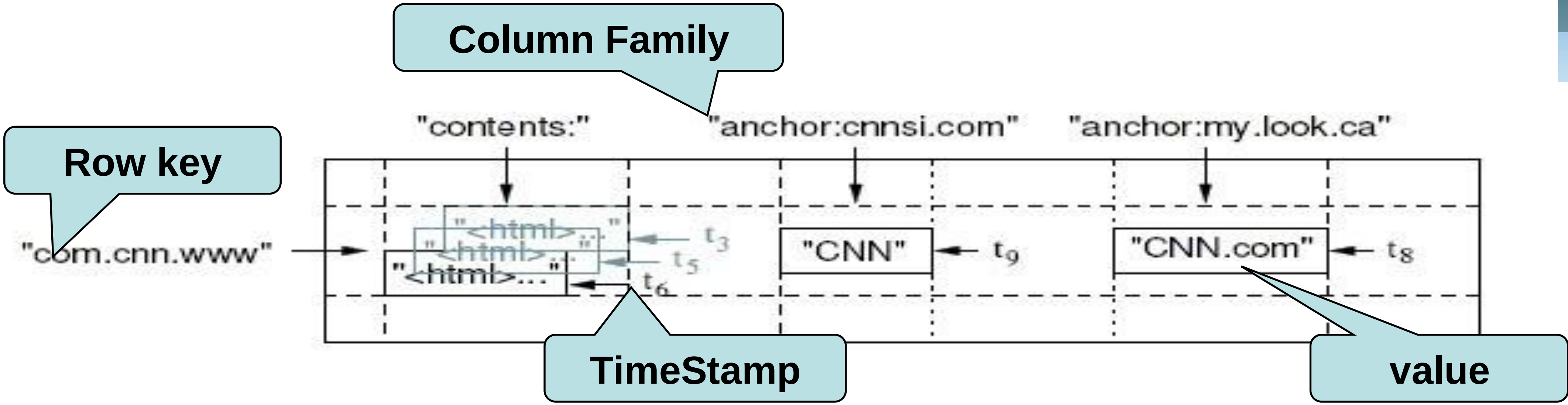
No-SQL

Google

An **open-source**, **distributed**, **column-oriented** database built on top of HDFS based on **BigTable**!

Big Table: Data Model

- Tables are sorted by Row
- Table schema only defines its *column families.*
  - Each family consists of any number of columns
  - Each column consists of any number of versions
  - Columns only exist when inserted, NULLs are free.
  - Columns within a family are sorted and stored together
- Everything except table names are byte[]
- (Row, Family: Column, Timestamp) → Value



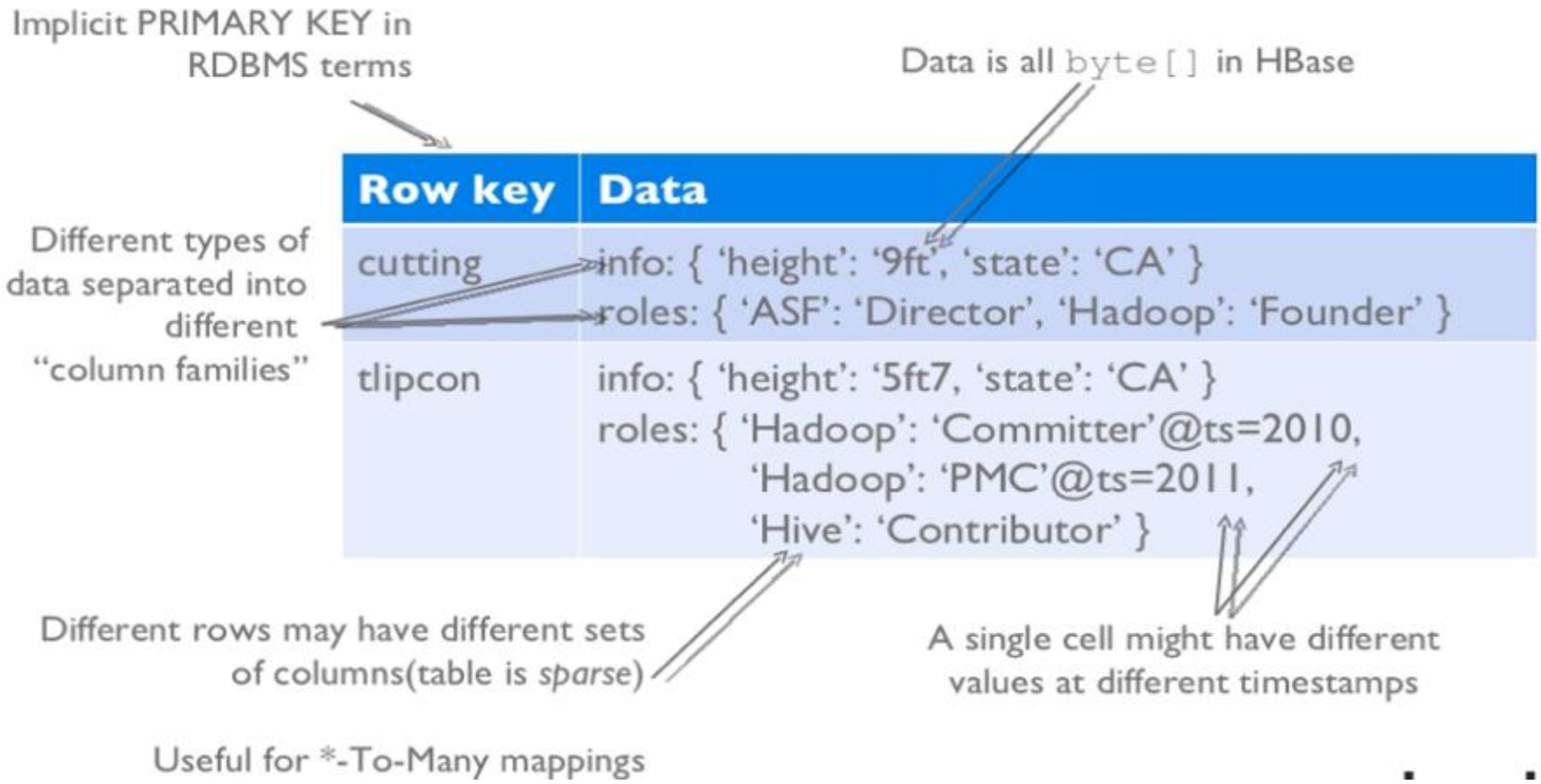
Column Family

Row key

TimeStamp

value

HBase Logical View



Implicit PRIMARY KEY in RDBMS terms

Data is all byte[] in HBase

Row key	Data

cutting	info: { 'height': '9ft', 'state': 'CA' }
	roles: { 'ASF': 'Director', 'Hadoop': 'Founder' }

tlipcon	info: { 'height': '5ft7, 'state': 'CA' }
	roles: { 'Hadoop': 'Committer'@ts=2010,
	'Hadoop': 'PMC'@ts=2011,
	'Hive': 'Contributor' }

Different types of data separated into different "column families"

Different rows may have different sets of columns(table is *sparse*)

A single cell might have different values at different timestamps

Useful for *-To-Many mappings

Physical distribution of the table

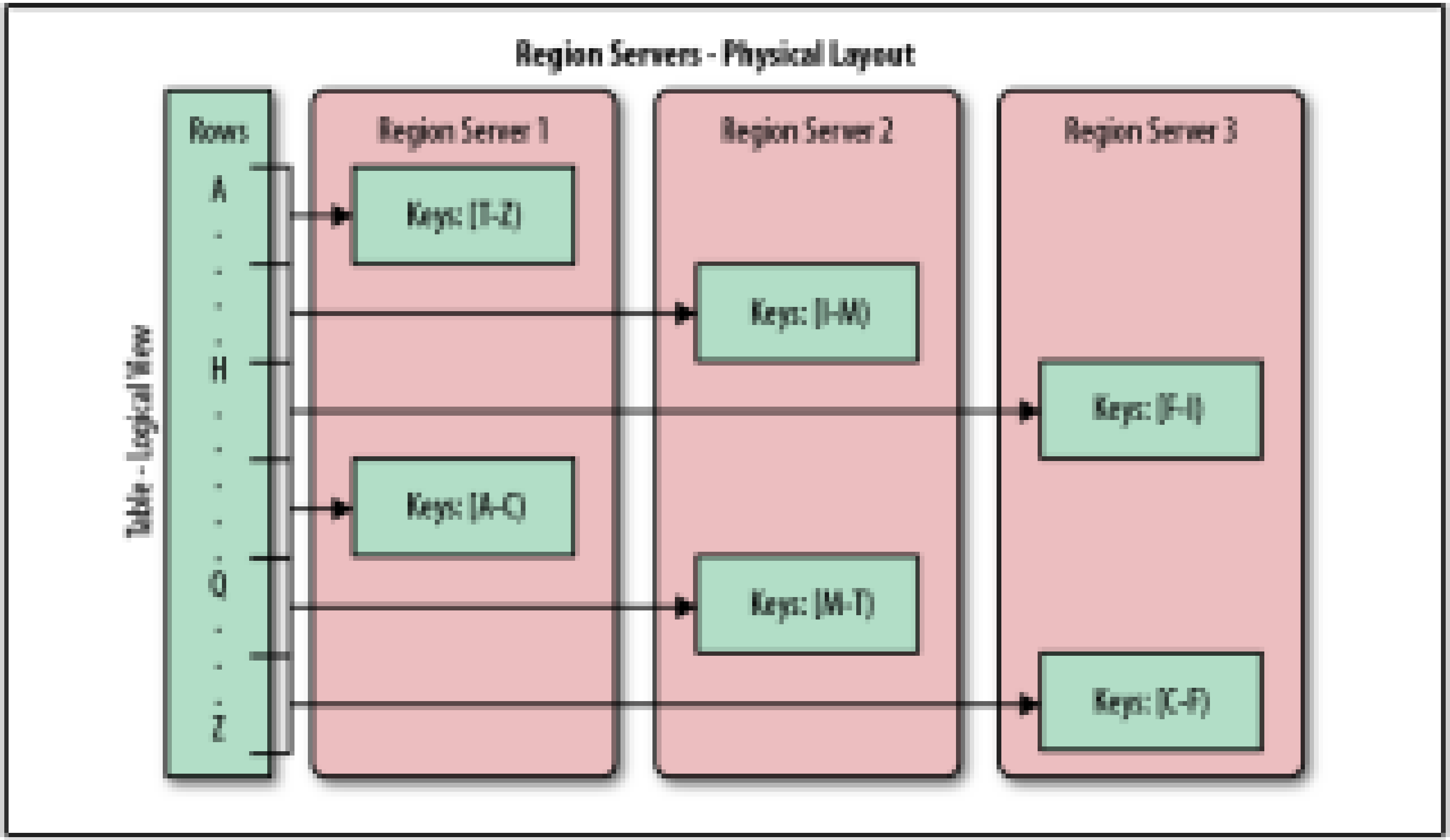
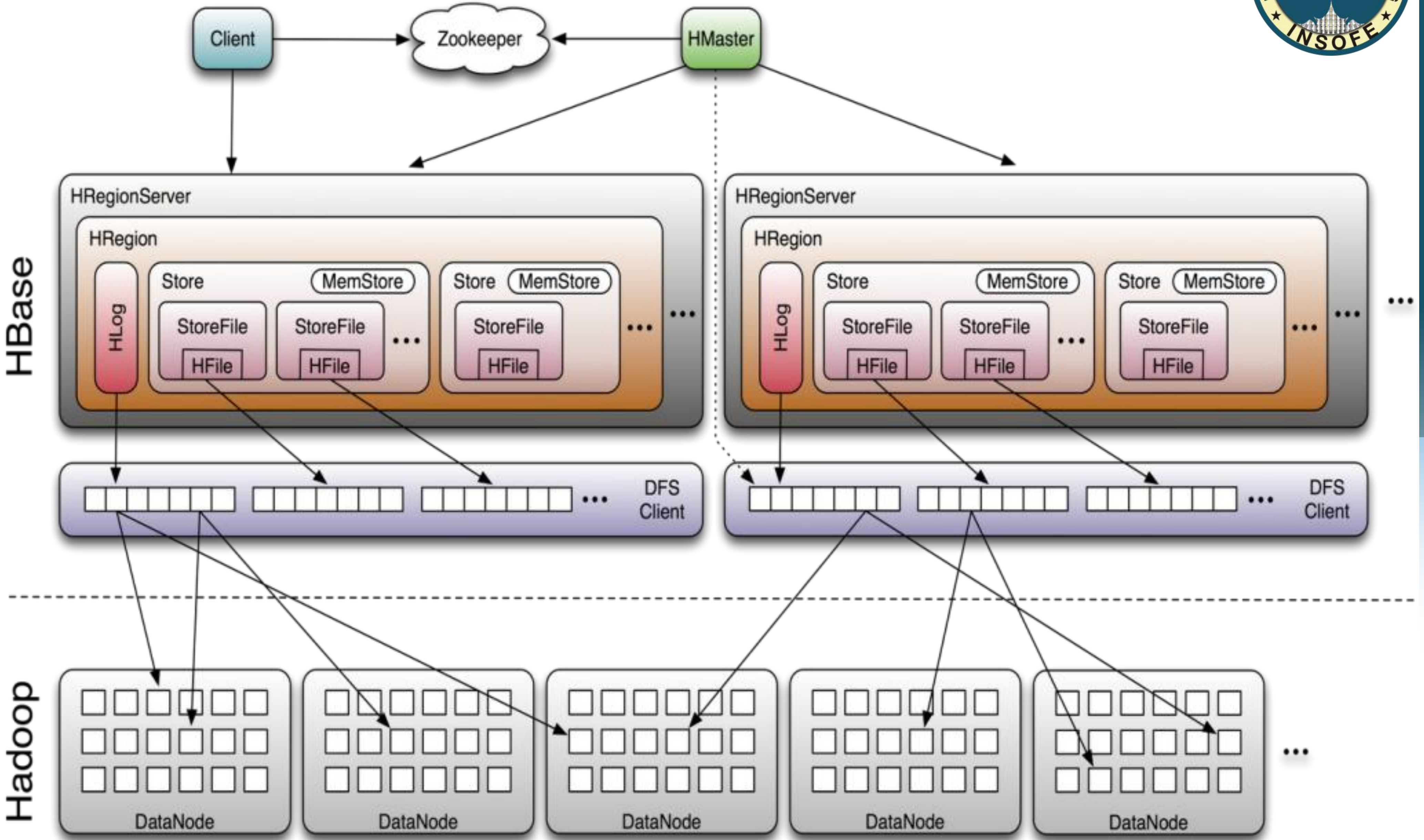


Figure 1-7. Rows grouped in regions and served by different servers

HBase implements BigTable on Hadoop



Write path of HBase

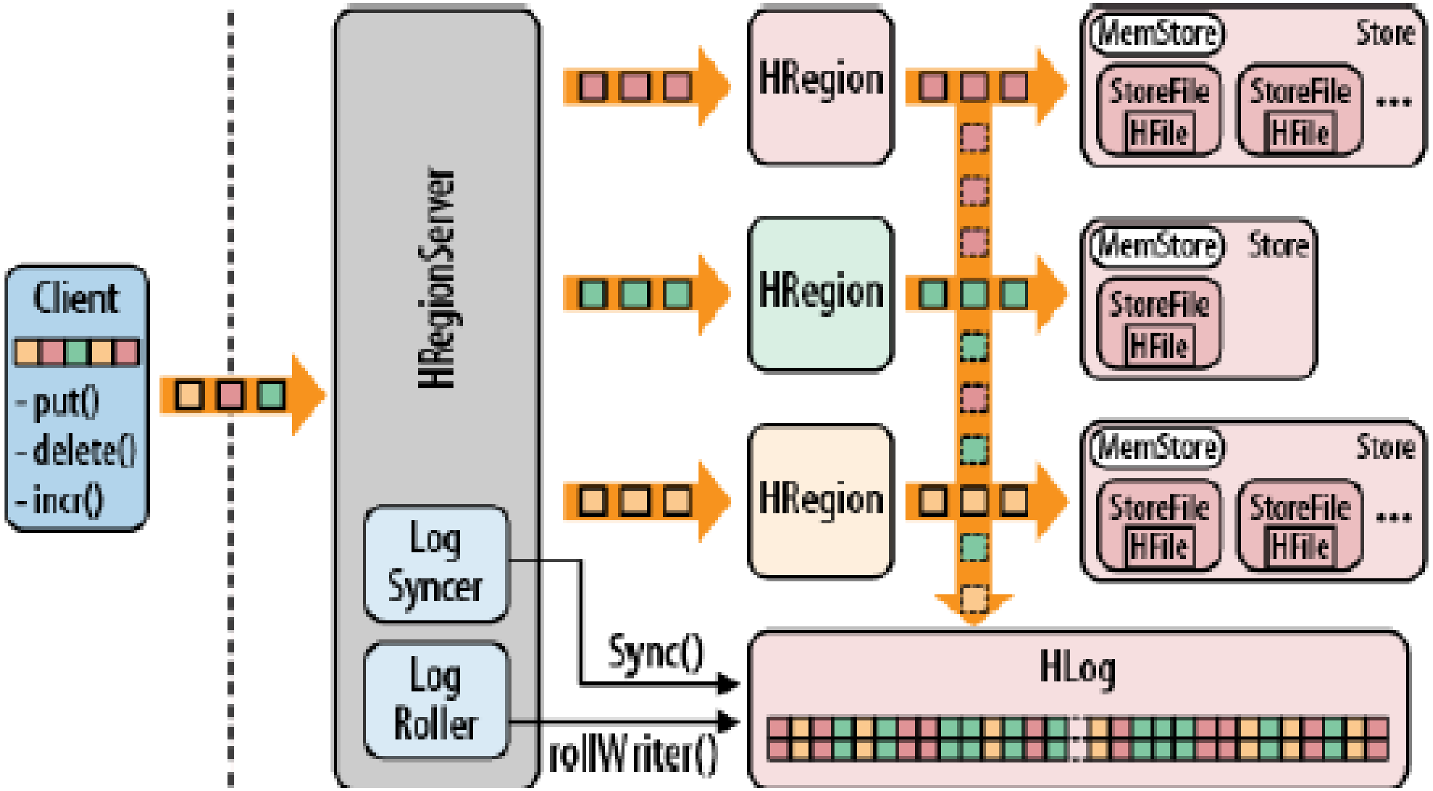
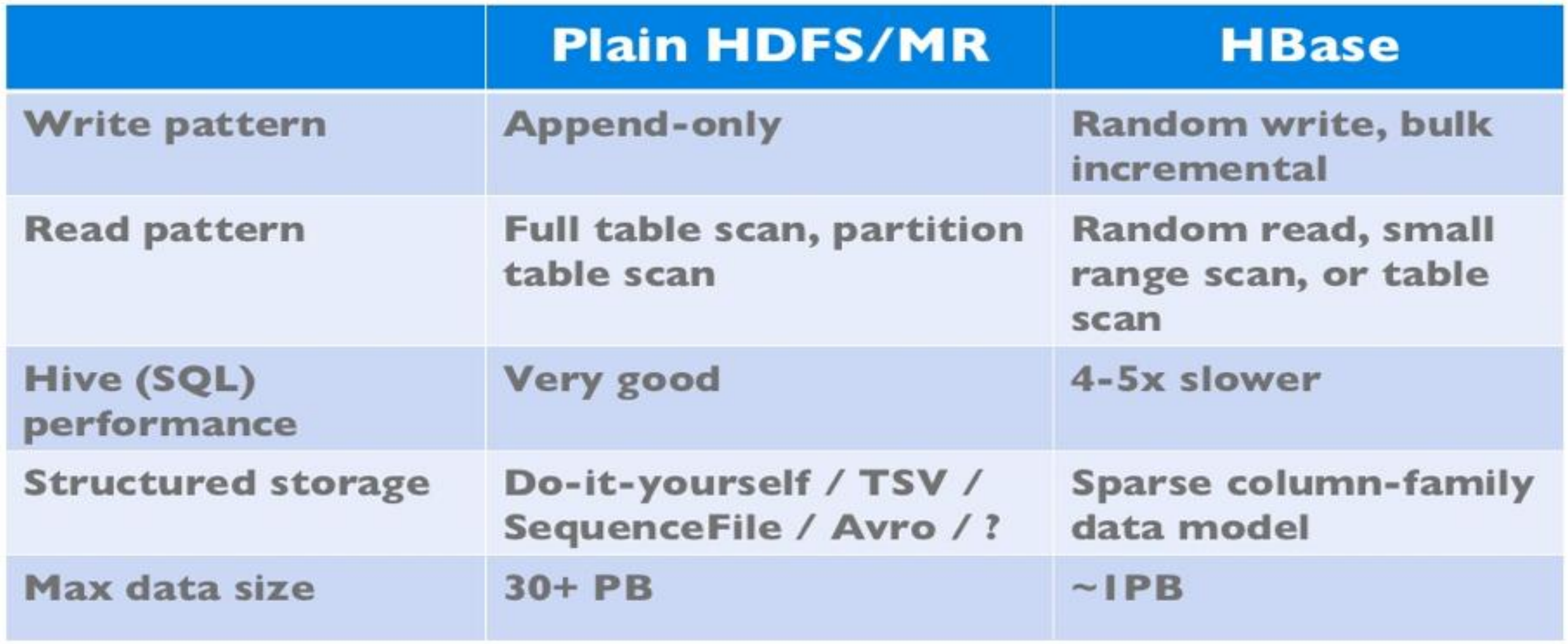


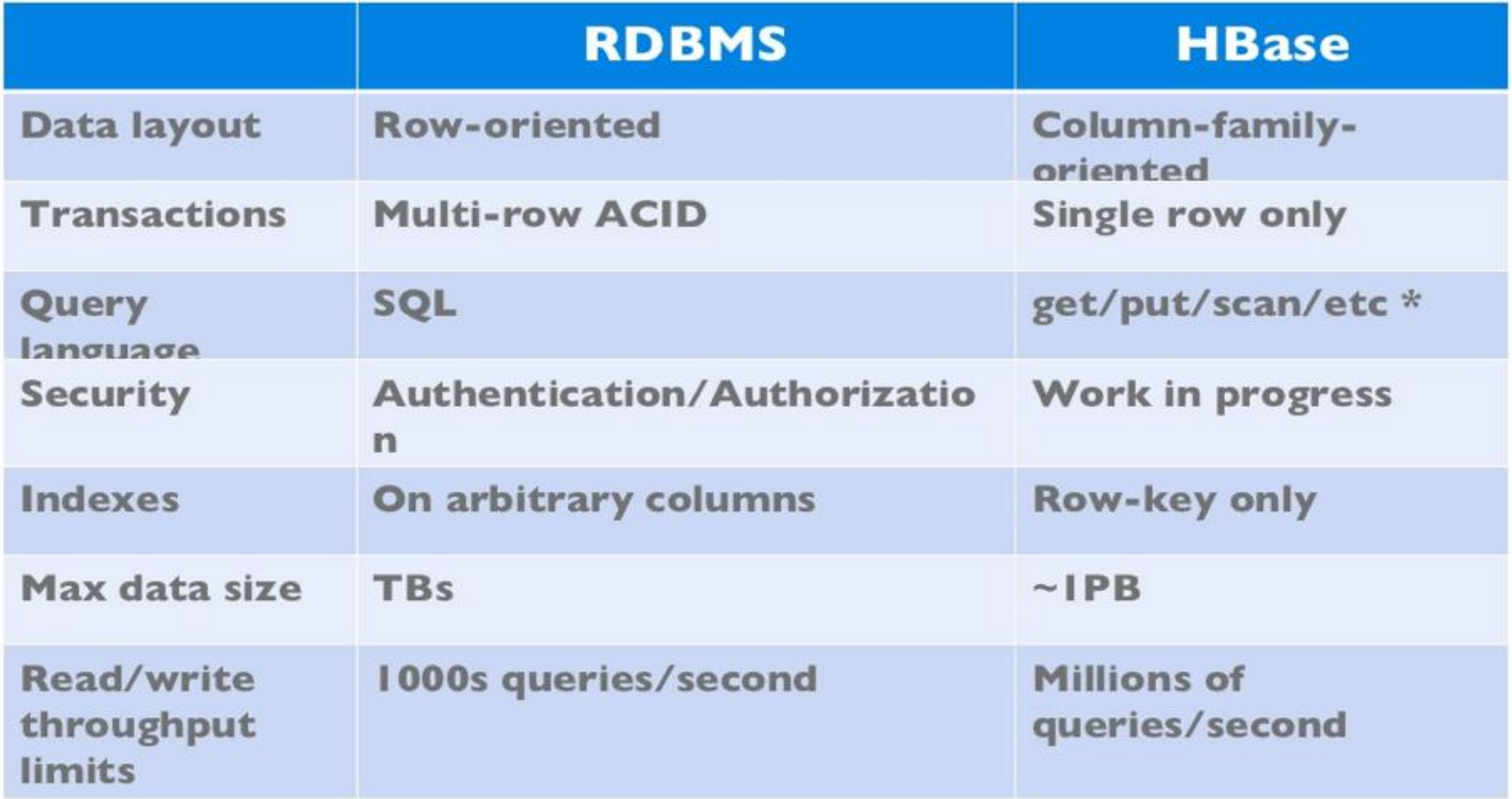
Figure: The write path of HBase

HBase vs. HDFS



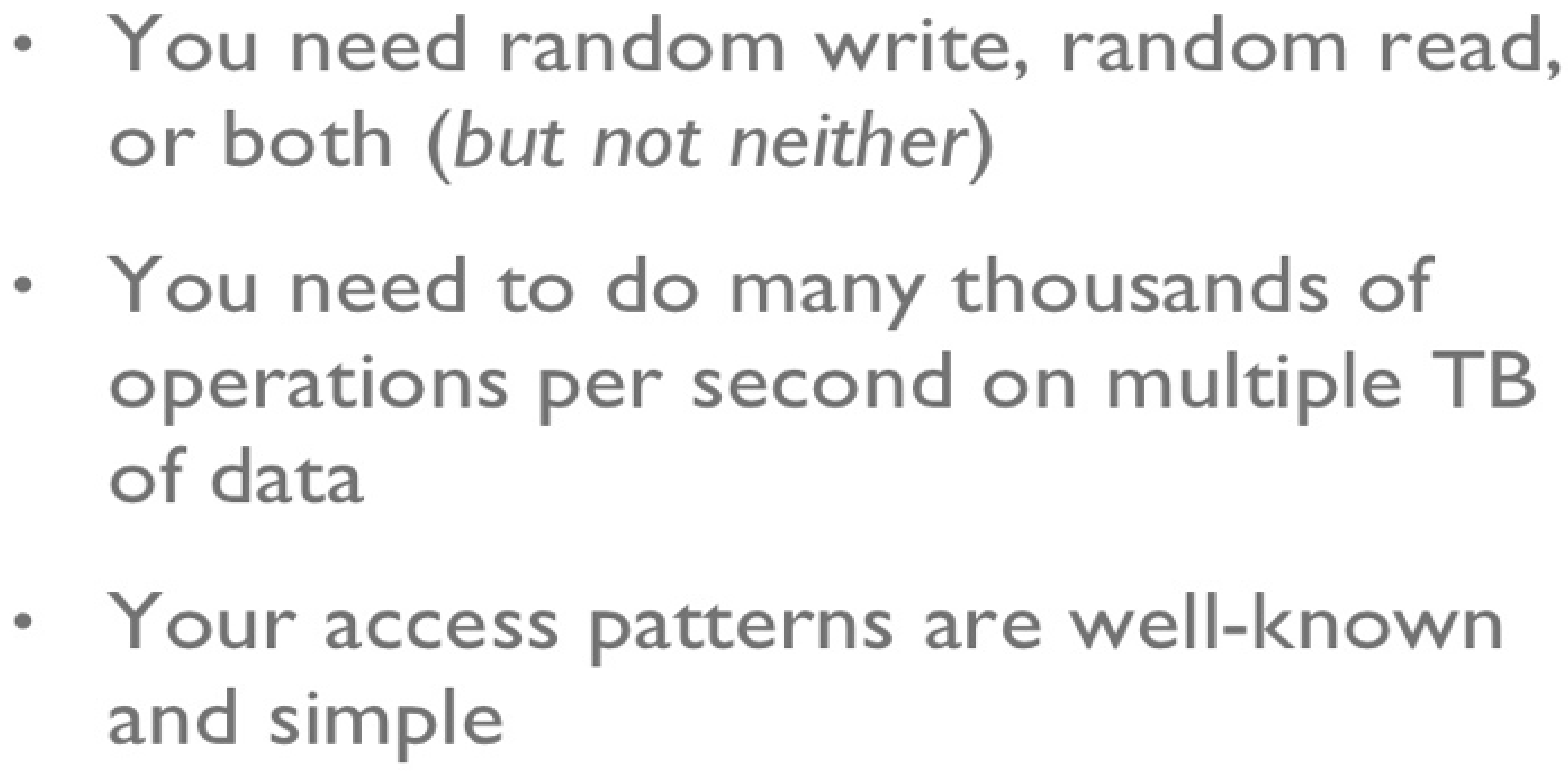
	Plain HDFS/MR	HBase
Write pattern	Append-only	Random write, bulk incremental
Read pattern	Full table scan, partition table scan	Random read, small range scan, or table scan
Hive (SQL) performance	Very good	4-5x slower
Structured storage	Do-it-yourself / TSV / SequenceFile / Avro / ?	Sparse column-family data model
Max data size	30+ PB	~1PB

HBase vs. RDBMS



	RDBMS	HBase
Data layout	Row-oriented	Column-family-oriented
Transactions	Multi-row ACID	Single row only
Query language	SQL	get/put/scan/etc *
Security	Authentication/Authorization	Work in progress
Indexes	On arbitrary columns	Row-key only
Max data size	TBs	~1PB
Read/write throughput limits	1000s queries/second	Millions of queries/second

When to use HBase



- You need random write, random read, or both (*but not neither*)
- You need to do many thousands of operations per second on multiple TB of data
- Your access patterns are well-known and simple

NoSQL: Performance Comparison

- Facebook Search

- MySQL > 50 GB Data
  - Writes Average : ~300 ms
  - Reads Average : ~350 ms

- Rewritten with Cassandra > 50 GB Data
  - Writes Average : 0.12 ms
  - Reads Average : 15 ms

Summary

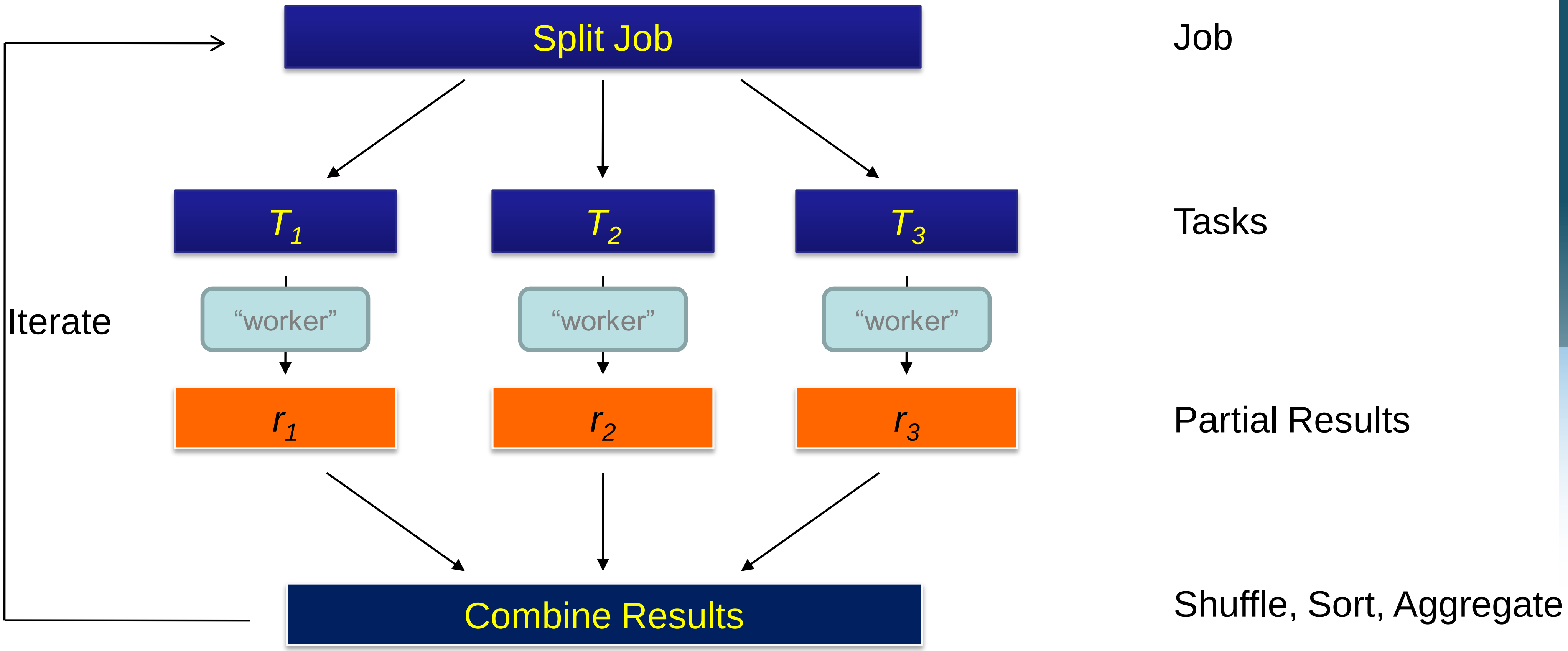
- SQL Databases
  - Predefined Schema
  - Standard definition and interface language
  - Tight consistency
  - Well defined semantics
- NoSQL Database
  - No predefined Schema
  - Per-product definition and interface language
  - Getting an answer quickly is more important than getting a correct answer

An excellent summary book on select NOSQL databases

WWW.CHRISTOF-STRAUCH.DE/NOSQLDBS.PDF

MAP REDUCE

Processing Frameworks 1: Map-Reduce



If not done right…

Source: Ricardo Guimarães Herrmann

Dean & Ghemawat, 2004

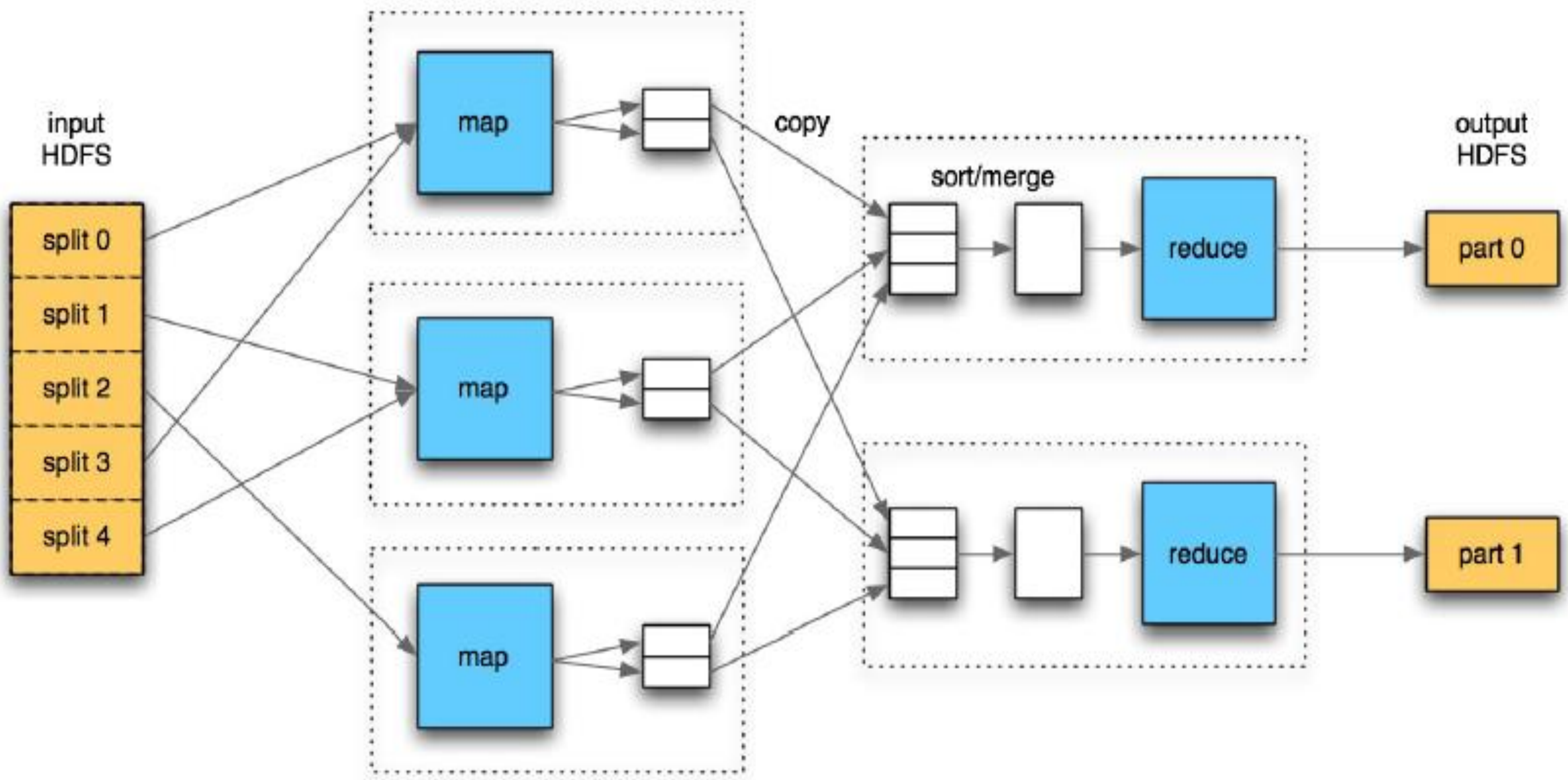
Map-Reduce: Solving large data problems

Map

- Iterate over a large number of records
- Extract something of interest from each
- Shuffle and sort intermediate results
- Aggregate intermediate results Reduce
- Generate final output

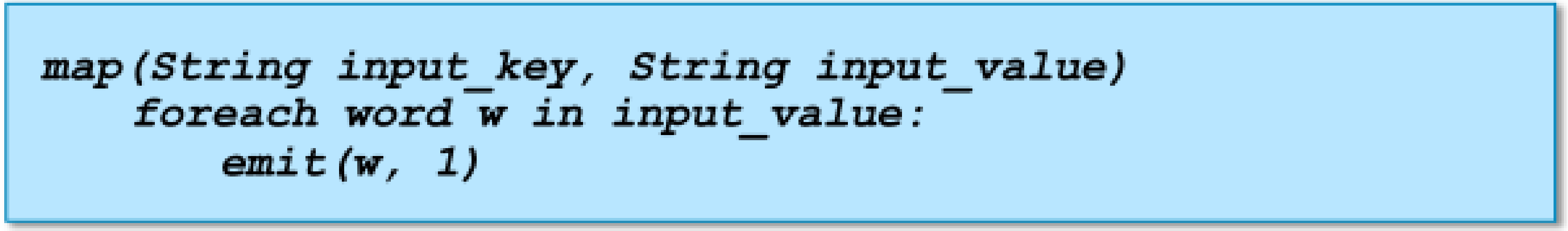
Key idea: provide a functional abstraction for these two operations

Map-Reduce: Physical Flow

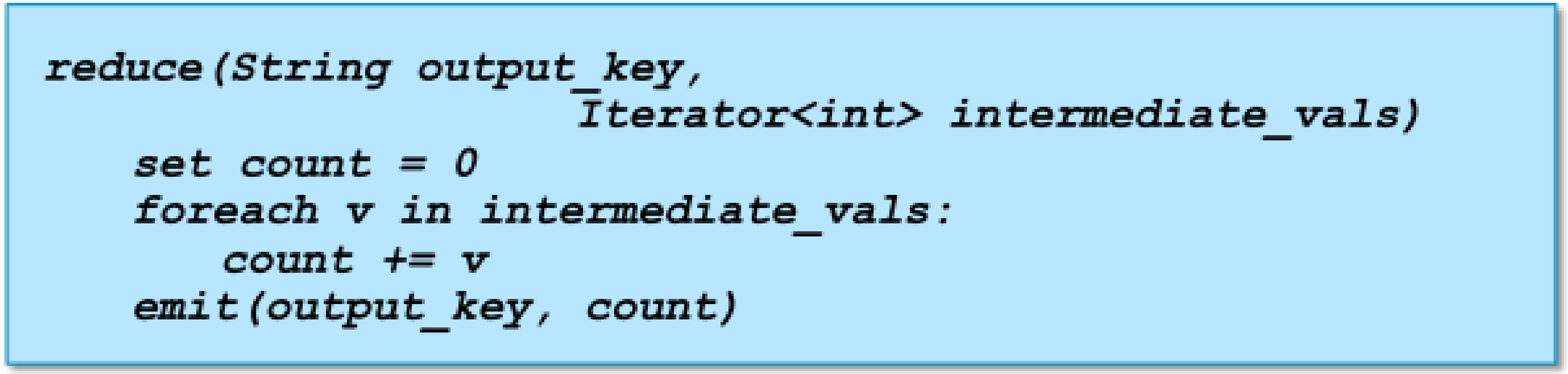


The Hello World of MapReduce

- **Count the number of occurrences of each word in a large amount of input data**
  - This is the ‘hello world’ of MapReduce programming



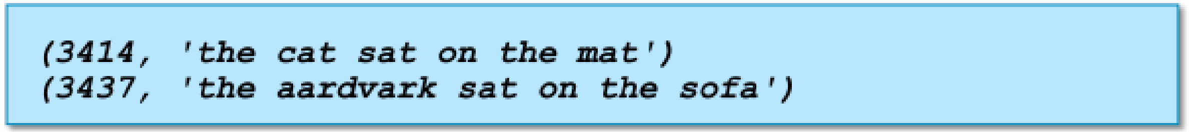
```
map(String input_key, String input_value)
    foreach word w in input_value:
        emit(w, 1)
```



```
reduce(String output_key,
                       Iterator<int> intermediate_vals)
    set count = 0
    foreach v in intermediate_vals:
        count += v
    emit(output_key, count)
```

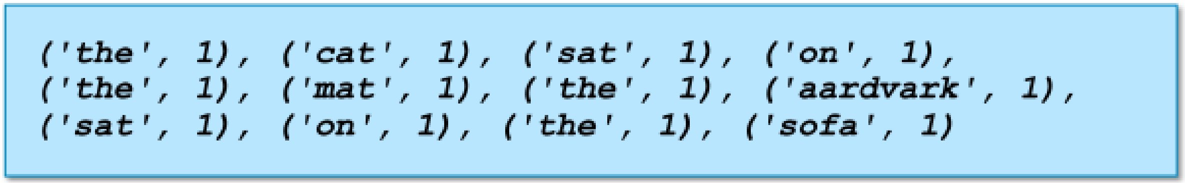
Hello World - continued

- Input to the Mapper:



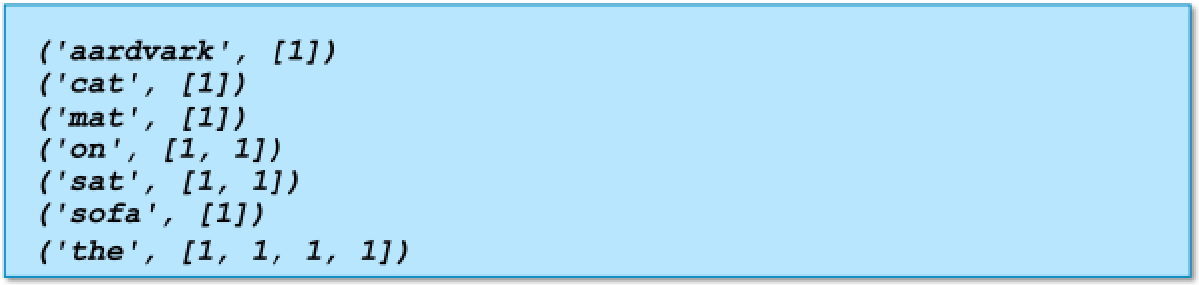
```
(3414, 'the cat sat on the mat')
(3437, 'the aardvark sat on the sofa')
```

- Output from the Mapper:



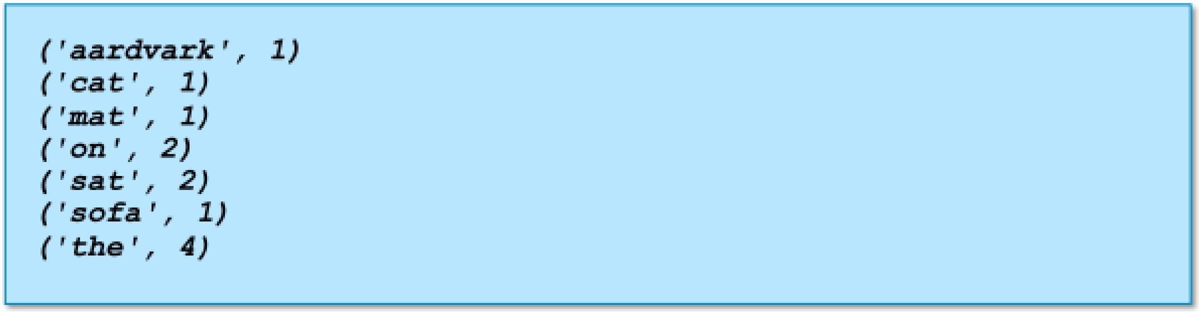
```
('the', 1), ('cat', 1), ('sat', 1), ('on', 1),
('the', 1), ('mat', 1), ('the', 1), ('aardvark', 1),
('sat', 1), ('on', 1), ('the', 1), ('sofa', 1)
```

- Intermediate data sent to the Reducer:



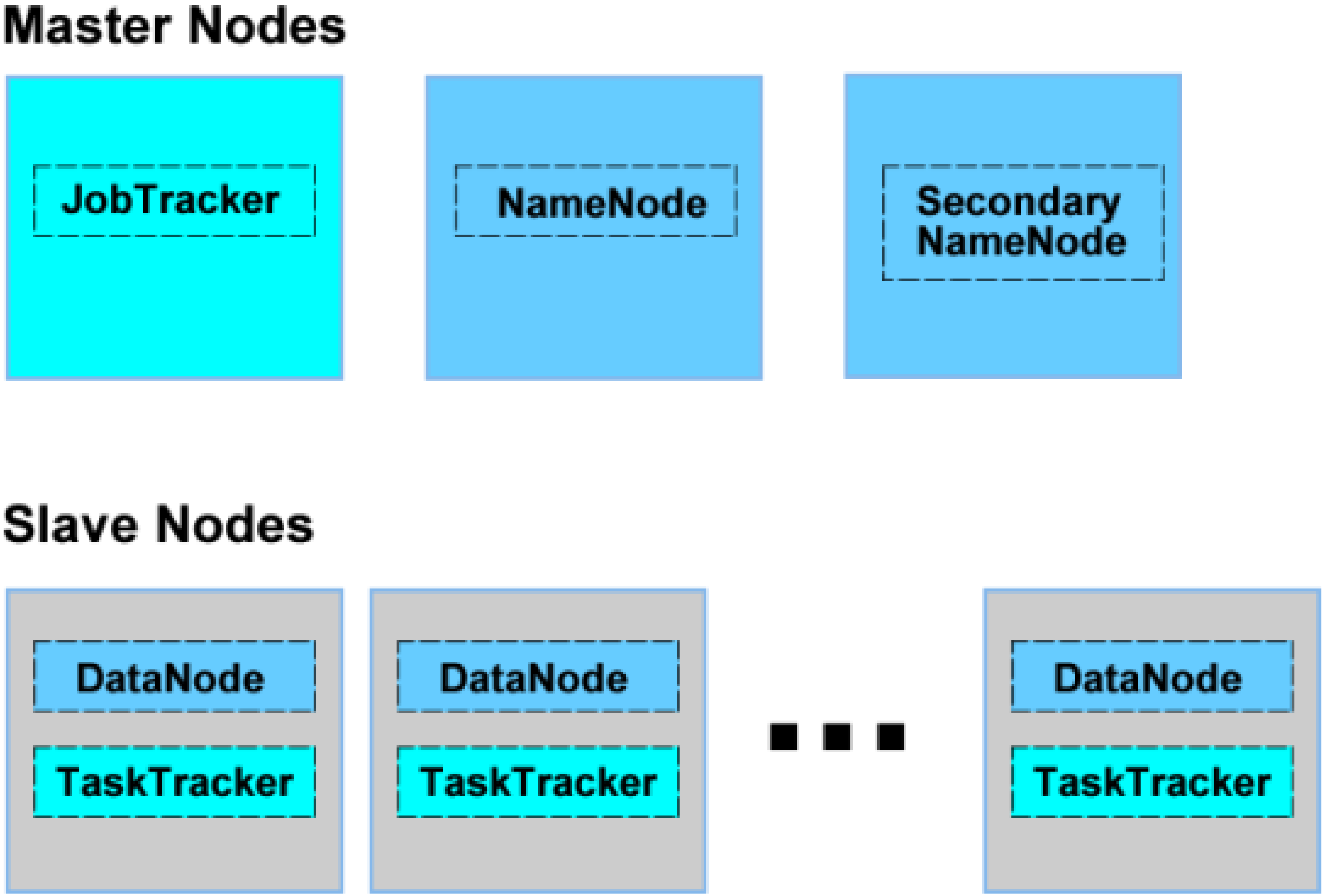
```
('aardvark', [1])
('cat', [1])
('mat', [1])
('on', [1, 1])
('sat', [1, 1])
('sofa', [1])
('the', [1, 1, 1, 1])
```

- Final Reducer output:



```
('aardvark', 1)
('cat', 1)
('mat', 1)
('on', 2)
('sat', 2)
('sofa', 1)
('the', 4)
```

Five Daemons of MapReduce



Master Nodes

JobTracker

NameNode

Secondary NameNode

Slave Nodes

DataNode

TaskTracker

DataNode

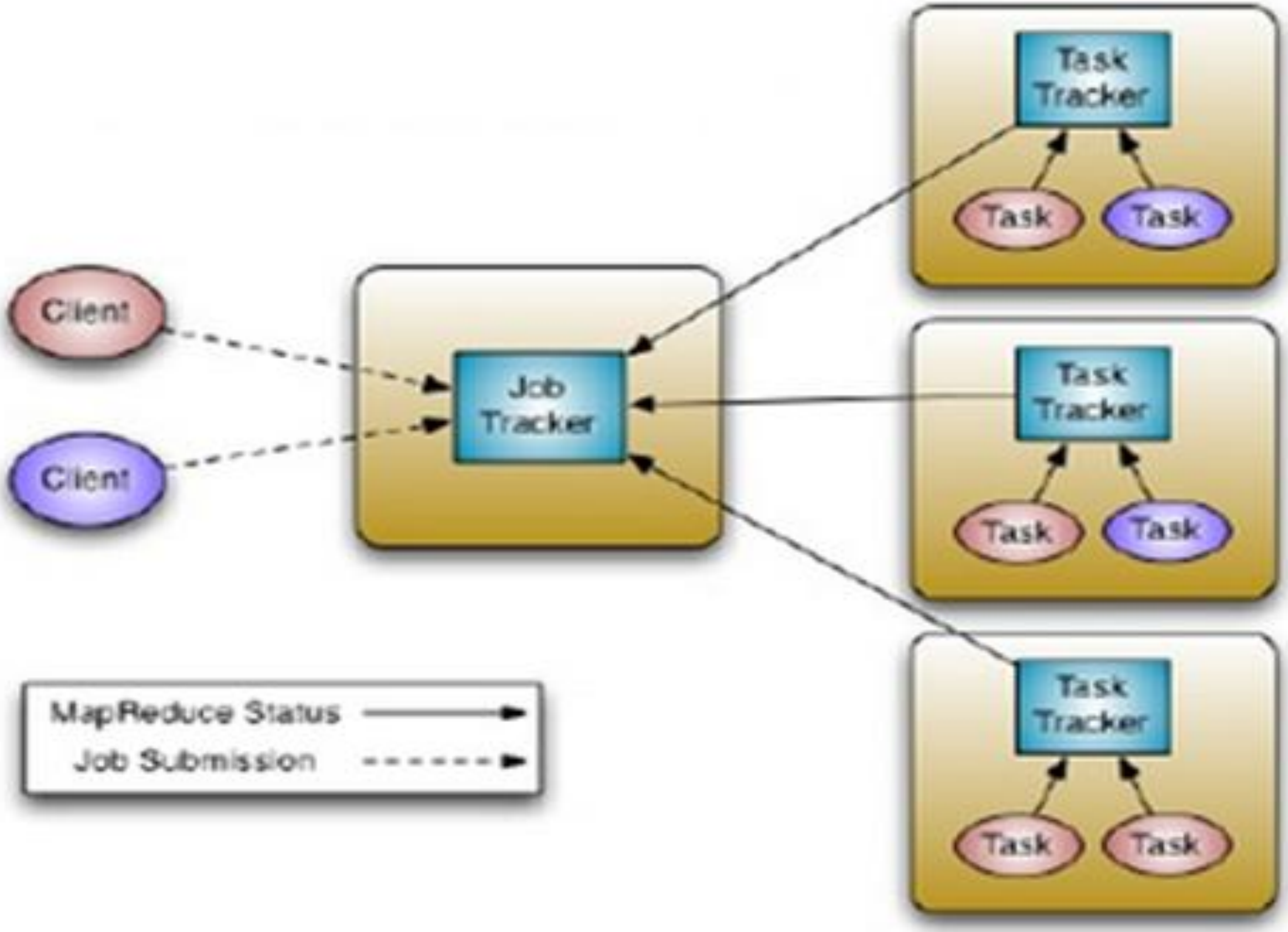
TaskTracker

...

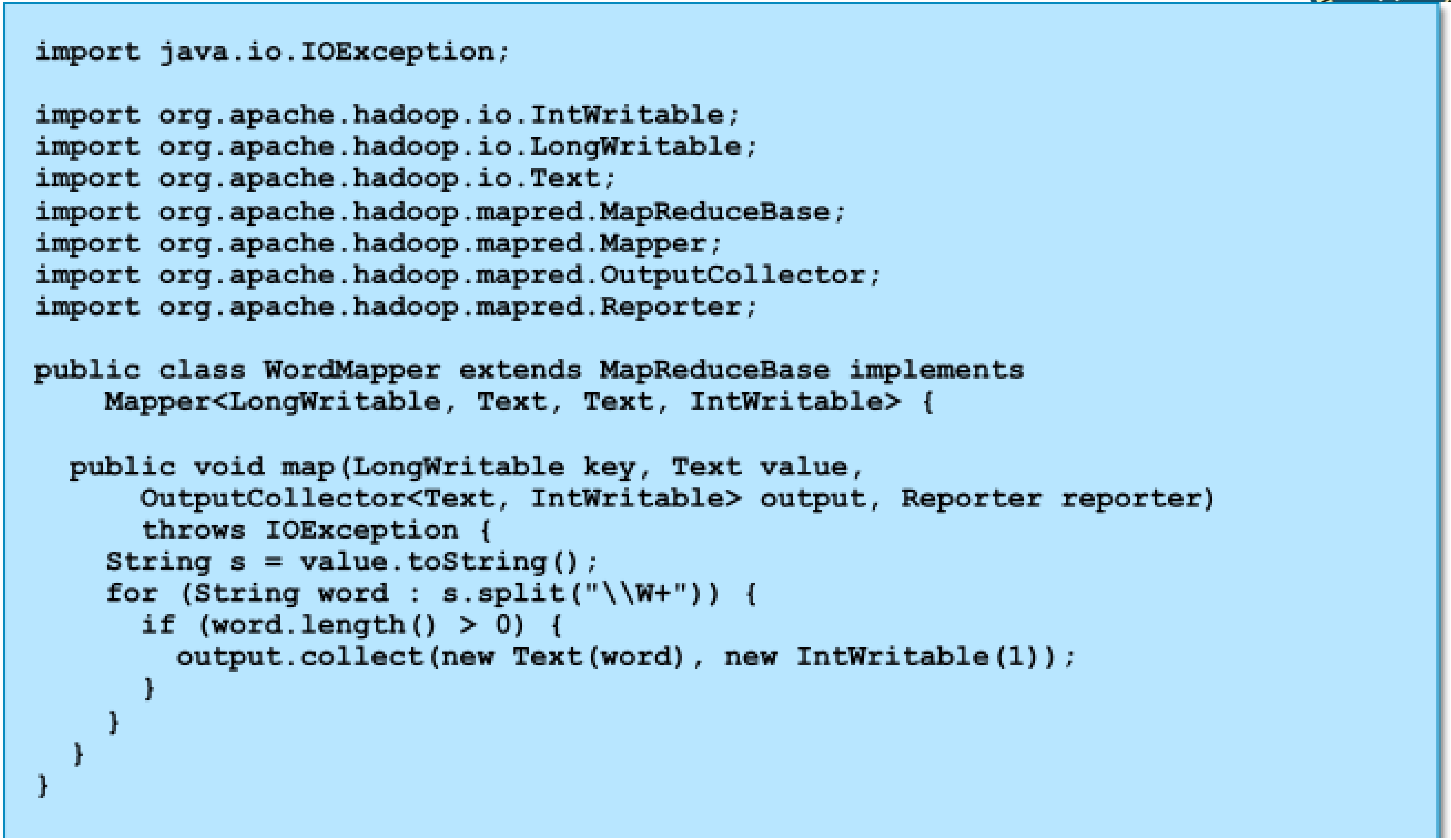
DataNode

TaskTracker

Map Reduce Daemons (circa 2011)

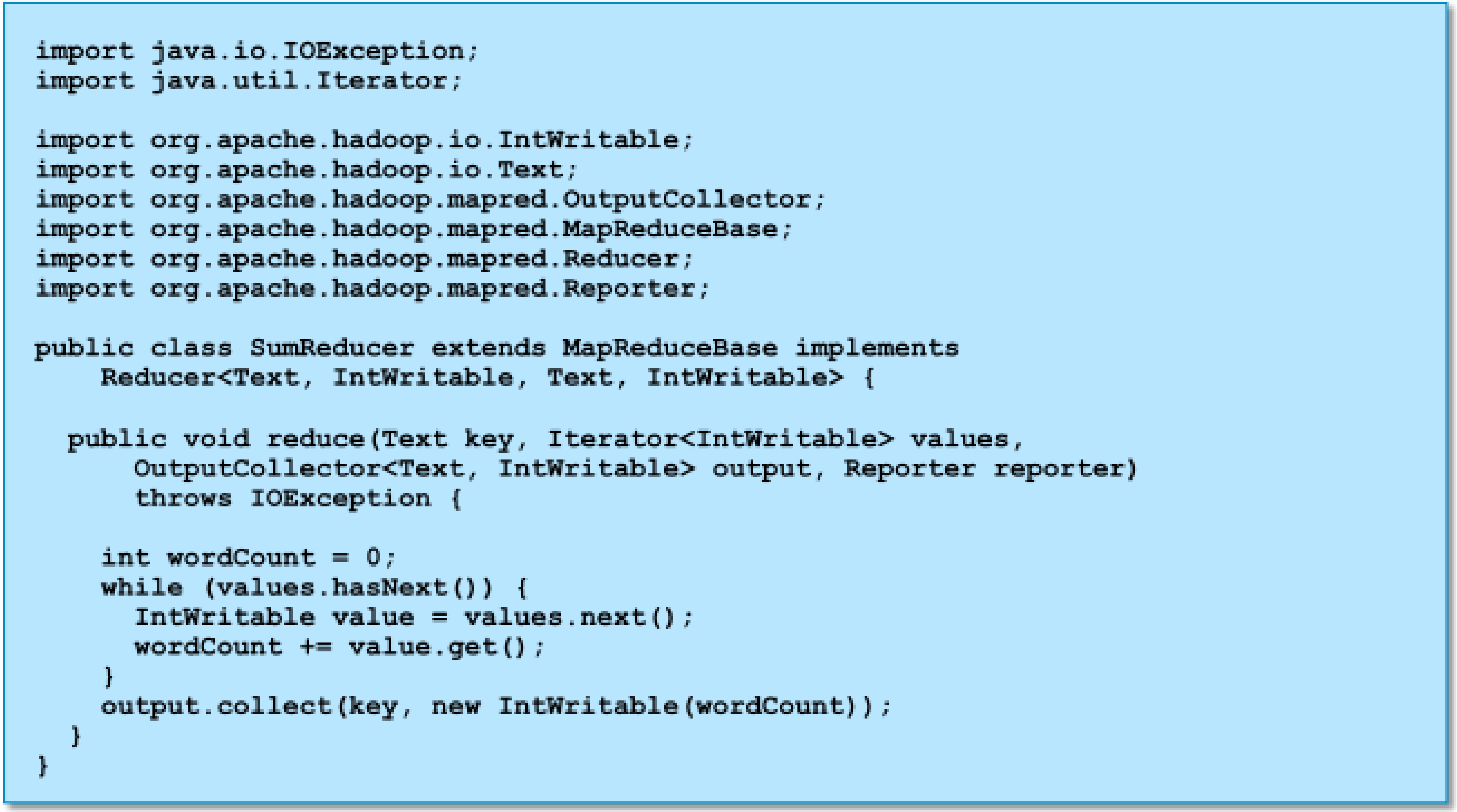


Mapper



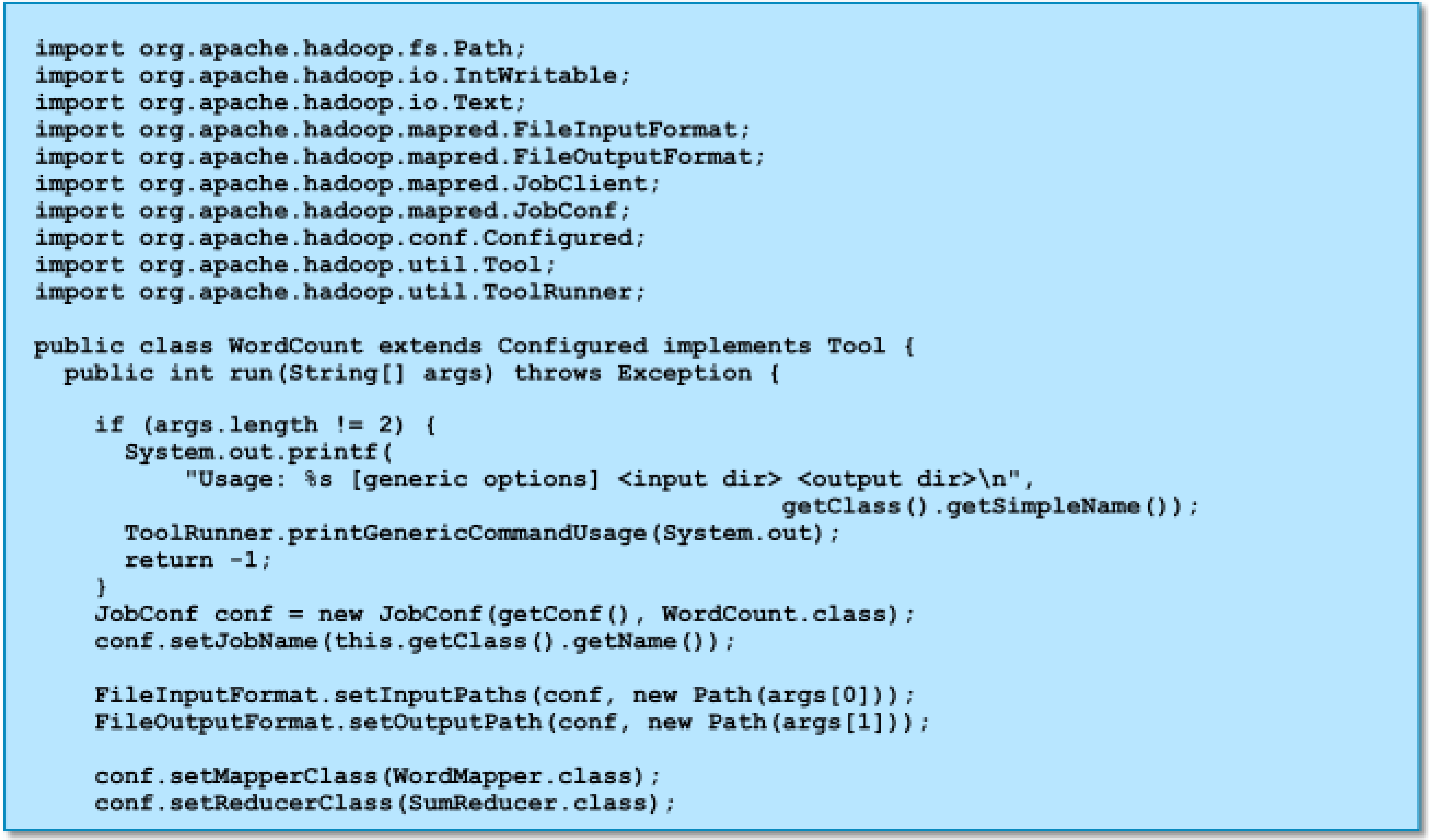
```
import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reporter;

public class WordMapper extends MapReduceBase implements
    Mapper<LongWritable, Text, Text, IntWritable> {

  public void map(LongWritable key, Text value,
      OutputCollector<Text, IntWritable> output, Reporter reporter)
      throws IOException {
    String s = value.toString();
    for (String word : s.split("\\W+")) {
      if (word.length() > 0) {
        output.collect(new Text(word), new IntWritable(1));
      }
    }
  }
}
```

Reducer



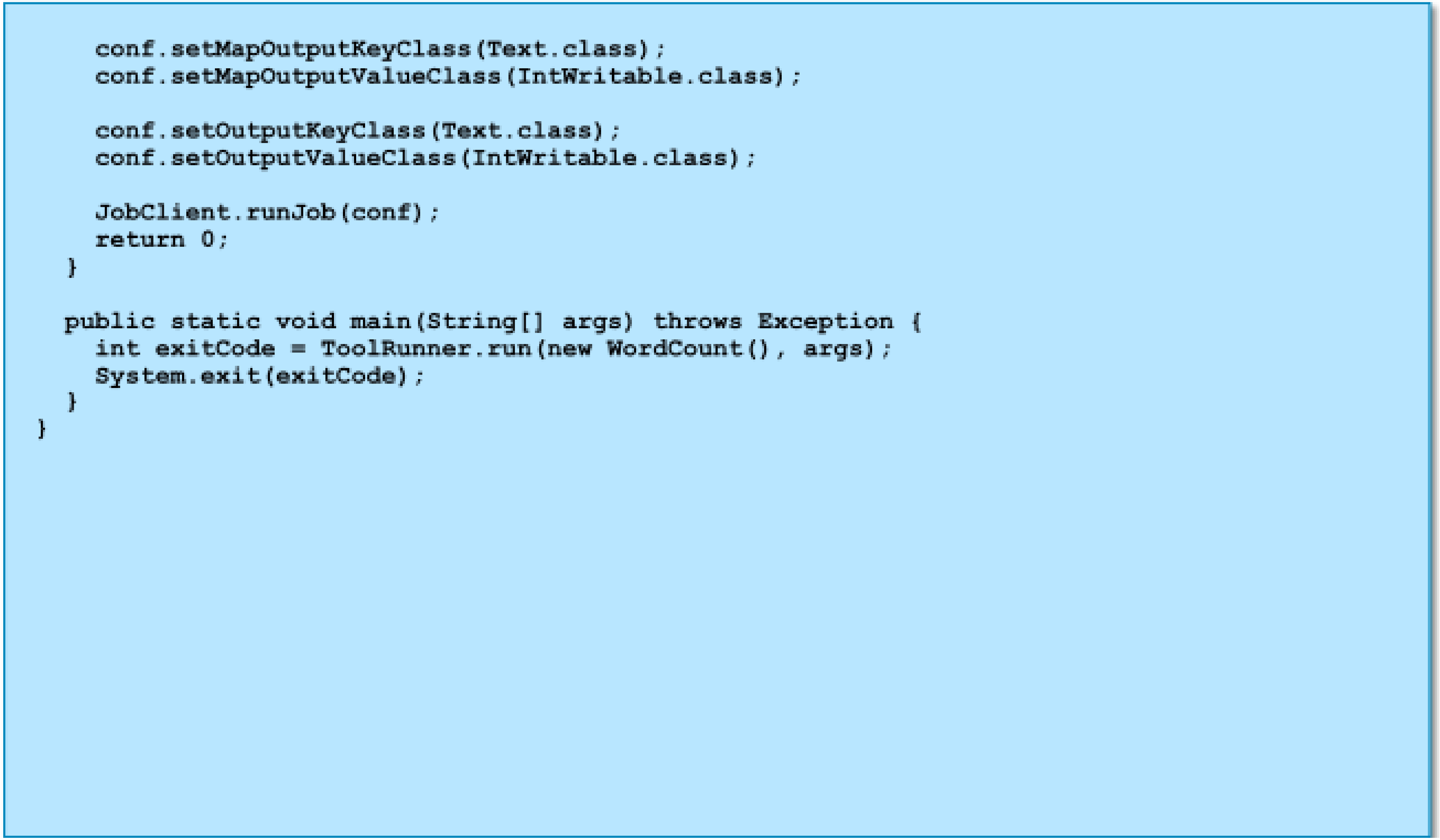
```
import java.io.IOException;
import java.util.Iterator;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;

public class SumReducer extends MapReduceBase implements
    Reducer<Text, IntWritable, Text, IntWritable> {

  public void reduce(Text key, Iterator<IntWritable> values,
      OutputCollector<Text, IntWritable> output, Reporter reporter)
      throws IOException {

    int wordCount = 0;
    while (values.hasNext()) {
      IntWritable value = values.next();
      wordCount += value.get();
    }
    output.collect(key, new IntWritable(wordCount));
  }
}
```

The Driver Code



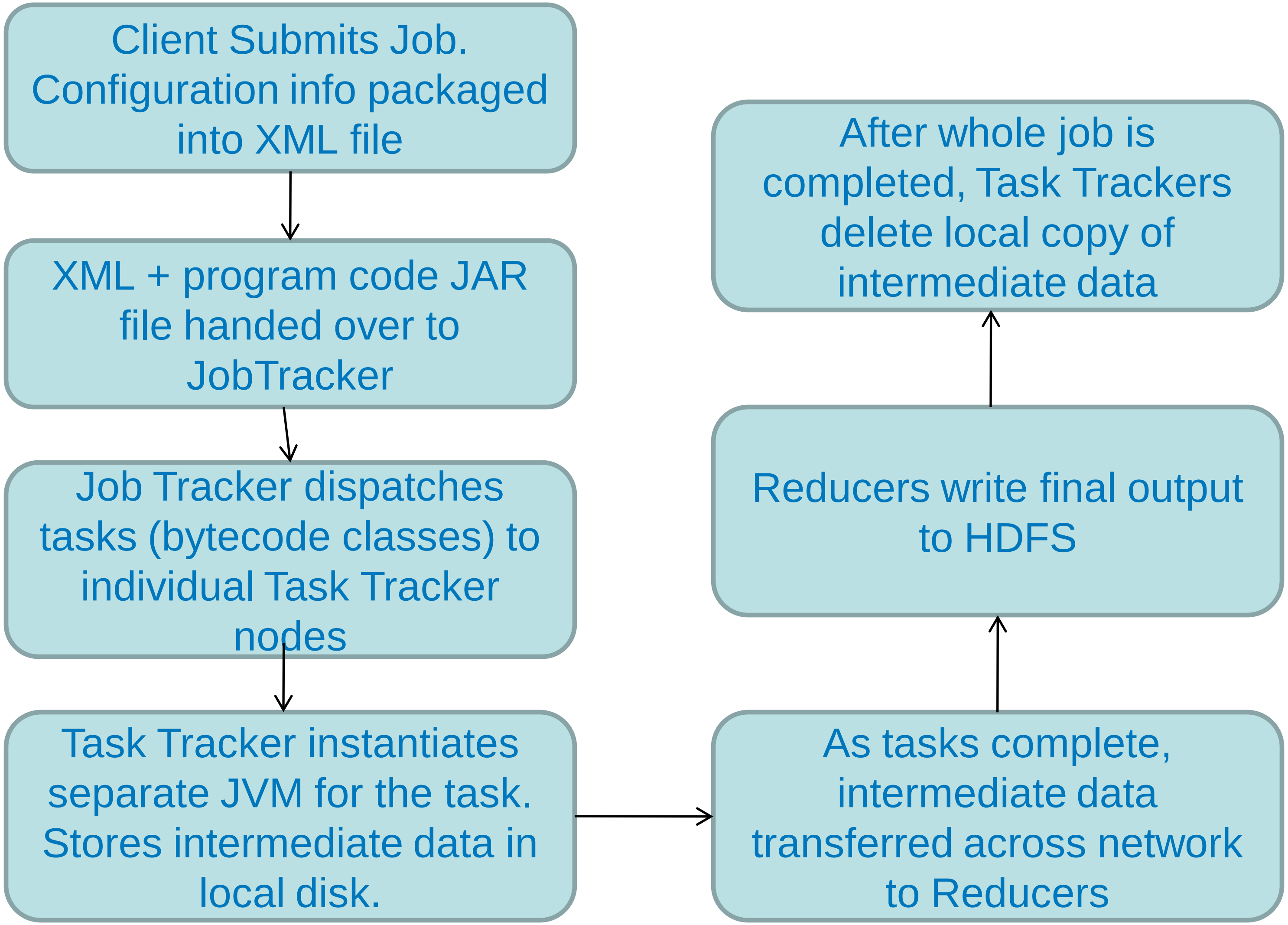
```
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

public class WordCount extends Configured implements Tool {
  public int run(String[] args) throws Exception {

    if (args.length != 2) {
      System.out.printf(
          "Usage: %s [generic options] <input dir> <output dir>\n",
                                                  getClass().getSimpleName());
      ToolRunner.printGenericCommandUsage(System.out);
      return -1;
    }
    JobConf conf = new JobConf(getConf(), WordCount.class);
    conf.setJobName(this.getClass().getName());

    FileInputFormat.setInputPaths(conf, new Path(args[0]));
    FileOutputFormat.setOutputPath(conf, new Path(args[1]));

    conf.setMapperClass(WordMapper.class);
    conf.setReducerClass(SumReducer.class);
```

Driver code… contd



```
    conf.setMapOutputKeyClass(Text.class);
    conf.setMapOutputValueClass(IntWritable.class);

    conf.setOutputKeyClass(Text.class);
    conf.setOutputValueClass(IntWritable.class);

    JobClient.runJob(conf);
    return 0;
  }

  public static void main(String[] args) throws Exception {
    int exitCode = ToolRunner.run(new WordCount(), args);
    System.exit(exitCode);
  }
}
```

Workflow



Client Submits Job. Configuration info packaged into XML file

XML + program code JAR file handed over to JobTracker

Job Tracker dispatches tasks (bytecode classes) to individual Task Tracker nodes

Task Tracker instantiates separate JVM for the task. Stores intermediate data in local disk.

As tasks complete, intermediate data transferred across network to Reducers

Reducers write final output to HDFS

After whole job is completed, Task Trackers delete local copy of intermediate data

International School of Engineering

Plot 63/A, 1st Floor, Road # 13, Film Nagar, Jubilee Hills, Hyderabad - 500 033

For Individuals:	+91-9502334561/63 or 040-65743991
For Corporates:	+91-9618483483
Web:	http://www.insofe.edu.in
Facebook:	https://www.facebook.com/insofe
Twitter:	https://twitter.com/Insofeedu
YouTube:	http://www.youtube.com/InsofeVideos
SlideShare:	http://www.slideshare.net/INSOFE
LinkedIn:	http://www.linkedin.com/company/international-school-of-engineering

This presentation may contain references to findings of various reports available in the public domain. INSOFE makes no representation as to their accuracy or that the organization subscribes to those findings.