Large-scale Data Mining: MapReduce and beyond
Part 1: Basics

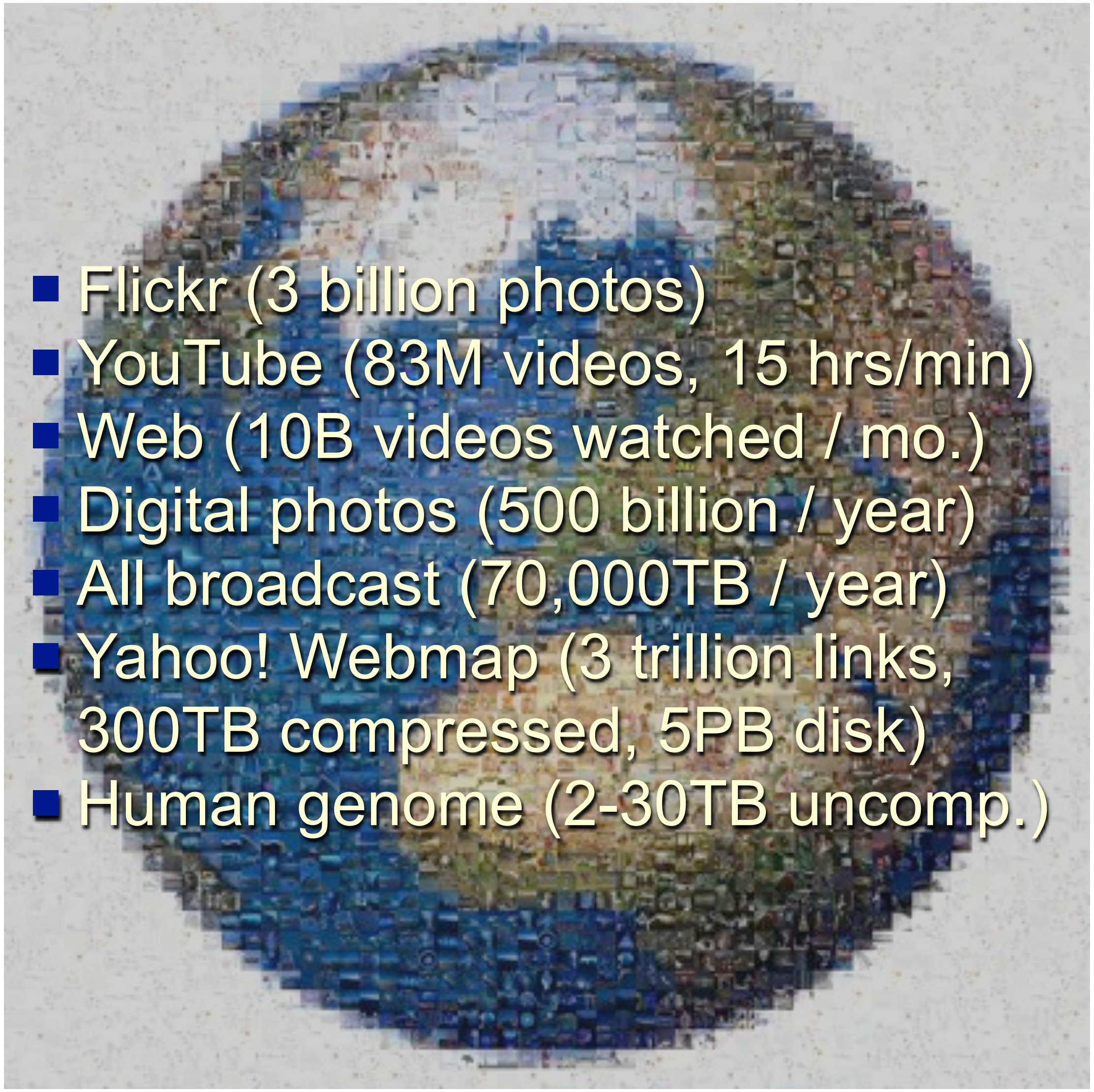
Spiros Papadimitriou, Google

Jimeng Sun, IBM Research

Rong Yan, Facebook

Data everywhere

Data everywhere



- Flickr (3 billion photos)
- YouTube (83M videos, 15 hrs/min)
- Web (10B videos watched / mo.)
- Digital photos (500 billion / year)
- All broadcast (70,000TB / year)
- Yahoo! Webmap (3 trillion links, 300TB compressed, 5PB disk)
- Human genome (2-30TB uncomp.)

Data everywhere

- Flickr (3 billion photos)
- YouTube (83M videos, 15 hrs/min)
- Web (10B videos watched / mo.)
- Digital photos (500 billion / year)
- All broadcast (70,000TB / year)
- Yahoo! Webmap (3 trillion links, 300TB compressed, 5PB disk)
- Human genome (2-30TB uncomp.)

So what ??

Data everywhere

- Flickr (3 billion photos)
- YouTube (83M videos, 15 hrs/min)
- Web (10B videos watched / mo.)
- Digital photos (500 billion / year)
- All broadcast (70,000TB / year)
- Yahoo! Webmap (3 trillion links, 300TB compressed, 5PB disk)
- Human genome (2-30TB uncomp.)

more is:

So what ??

Data everywhere

- Flickr (3 billion photos)
- YouTube (83M videos, 15 hrs/min)
- Web (10B videos watched / mo.)
- Digital photos (500 billion / year)
- All broadcast (70,000TB / year)
- Yahoo! Webmap (3 trillion links, 300TB compressed, 5PB disk)
- Human genome (2-30TB uncomp.)

more is:
more …

So what ??

Data everywhere

- Flickr (3 billion photos)
- YouTube (83M videos, 15 hrs/min)
- Web (10B videos watched / mo.)
- Digital photos (500 billion / year)
- All broadcast (70,000TB / year)
- Yahoo! Webmap (3 trillion links, 300TB compressed, 5PB disk)
- Human genome (2-30TB uncomp.)

more is:
more …

more is:
different!

So what ??

Data everywhere

- Opportunities
  - Real-time access to content
  - Richer context from users and hyperlinks
  - Abundant training examples
  - “Brute-force” methods may suffice
- Challenges
  - “Dirtier” data
  - Efficient algorithms
  - Scalability (with reasonable cost)

Data everywhere

- Opportunities
  - Real-time access to content
  - Richer context from users and hyperlinks
  - Abundant training examples
  - "Brute-force" methods may suffice
- Challenges
  - "Dirtier" data
  - Efficient algorithms
  - Scalability (with reasonable cost)

Data everywhere

- Opportunities
  - Real-time access to content
  - Richer context from users and hyperlinks
  - Abundant training examples
  - “Brute-force” methods may suffice
- Challenges
  - “Dirtier” data
  - Efficient algorithms
  - Scalability (with reasonable cost)

“The Google Way”

Chris Anderson, Wired (July 2008)

“The Google Way”

“All models are wrong, but some are useful”
– George Box

Chris Anderson, Wired (July 2008)

“The Google Way”

“All models are wrong, but some are useful”
– George Box

“All models are wrong, and increasingly you can succeed without them.” – Peter Norvig

- Google PageRank
- Shotgun gene sequencing

Chris Anderson, Wired (July 2008)

“The Google Way”

“All models are wrong, but some are useful”
– George Box

“All models are wrong, and increasingly you can succeed without them.” – Peter Norvig

- Google PageRank
- Shotgun gene sequencing
- Language translation

Chris Anderson, Wired (July 2008)

"The Google Way"

"All models are wrong, but some are useful"
– George Box

"All models are wrong, and increasingly you can succeed without them." – Peter Norvig

- Google PageRank
- Shotgun gene sequencing
- Language translation
- …

Chris Anderson, Wired (July 2008)

Getting over the marketing hype…

Cloud Computing

=

Getting over the marketing hype…

Cloud Computing

=

Internet

Getting over the marketing hype…

Cloud Computing

=

Internet

+

Commoditization/
standardization

Getting over the marketing hype…

Cloud Computing

=

Internet

+

Commoditization/

‘It’s what I and many others have worked towards our entire careers. It’s just happening *now*.’

– Eric Schmidt

This tutorial

- Is **not** about cloud computing
- **But** about large scale data processing

This tutorial

- Is **not** about cloud computing
- **But** about large scale data processing

Data + Algorithms

Tutorial overview

- Part 1 (Spiros): Basic concepts & tools
  - MapReduce & distributed storage
  - Hadoop / HBase / Pig / Cascading / Hive
- Part 2 (Jimeng): Algorithms
  - Information retrieval
  - Graph algorithms
  - Clustering (k-means)
  - Classification (k-NN, naïve Bayes)
- Part 3 (Rong): Applications
  - Text processing
  - Data warehousing
  - Machine learning

Outline

- Introduction
- **MapReduce & distributed storage**
- Hadoop
  - HBase
  - Pig
  - Cascading
  - Hive
- Summary

What is MapReduce?

- Programming model?
- Execution environment?
- Software package?

What is MapReduce?

- Programming model?
- Execution environment?
- Software package?

It’s all of those things, depending who you ask…

What is MapReduce?

- Programming model?
- Executi
- Software

"MapReduce" (this talk)
==
Distributed **computation** +
distributed **storage** +
scheduling / fault tolerance

It's all of those things, depending who you ask…

Example – Programming model

employees.txt

# LAST	FIRST	SALARY
Smith	John	$90,000
Brown	David	$70,000
Johnson	George	$95,000
Yates	John	$80,000
Miller	Bill	$65,000
Moore	Jack	$85,000
Taylor	Fred	$75,000
Smith	David	$80,000
Harris	John	$90,000
...	...	...
...	...	...

Q: "What is the frequency of each first name?"

Example – Programming model

employees.txt

# LAST	FIRST	SALARY
Smith	John	$90,000
Brown	David	$70,000
Johnson	George	$95,000
Yates	John	$80,000
Miller	Bill	$65,000
Moore	Jack	$85,000
Taylor	Fred	$75,000
Smith	David	$80,000
Harris	John	$90,000
...	...	...
...	...	...

```
def getName (line):
```

Q: "What is the frequency of each first name?"

Example – Programming model

employees.txt

# LAST	FIRST	SALARY
Smith	John	$90,000
Brown	David	$70,000
Johnson	George	$95,000
Yates	John	$80,000
Miller	Bill	$65,000
Moore	Jack	$85,000
Taylor	Fred	$75,000
Smith	David	$80,000
Harris	John	$90,000
...	...	...
...	...	...

mapper

```
def getName (line):
    return  line.split('\t')[1]

def addCounts (hist,  name):
```

Q: "What is the frequency of each first name?"

Example – Programming model

employees.txt

# LAST	FIRST	SALARY
Smith	John	$90,000
Brown	David	$70,000
Johnson	George	$95,000
Yates	John	$80,000
Miller	Bill	$65,000
Moore	Jack	$85,000
Taylor	Fred	$75,000
Smith	David	$80,000
Harris	John	$90,000
...	...	...
...	...	...

mapper

```
def getName (line):
    return  line.split('\t')[1]

def addCounts (hist,  name):
    hist[name] = \
    hist.get(name,default=0) + 1
    return hist

input = open('employees.txt', 'r')
```

Q: "What is the frequency of each first name?"

Example – Programming model

employees.txt

# LAST	FIRST	SALARY
Smith	John	$90,000
Brown	David	$70,000
Johnson	George	$95,000
Yates	John	$80,000
Miller	Bill	$65,000
Moore	Jack	$85,000
Taylor	Fred	$75,000
Smith	David	$80,000
Harris	John	$90,000
...	...	...
...	...	...

mapper

```
def getName (line):
    return  line.split('\t')[1]

def addCounts (hist,  name):
    hist[name] = \
    hist.get(name,default=0) + 1
    return hist

input = open('employees.txt', 'r')
```

Q: "What is the frequency of each first name?"

Example – Programming model

employees.txt

# LAST	FIRST	SALARY
Smith	John	$90,000
Brown	David	$70,000
Johnson	George	$95,000
Yates	John	$80,000
Miller	Bill	$65,000
Moore	Jack	$85,000
Taylor	Fred	$75,000
Smith	David	$80,000
Harris	John	$90,000
...	...	...
...	...	...

mapper

```
def getName (line):
   return  line.split('\t')[1]

def addCounts (hist,  name):
   hist[name] = \
   hist.get(name,default=0) + 1
   return hist

input = open('employees.txt', 'r')

intermediate = map(getName, input)
```

Q: "What is the frequency of each first name?"

Example – Programming model

employees.txt

# LAST	FIRST	SALARY
Smith	John	$90,000
Brown	David	$70,000
Johnson	George	$95,000
Yates	John	$80,000
Miller	Bill	$65,000
Moore	Jack	$85,000
Taylor	Fred	$75,000
Smith	David	$80,000
Harris	John	$90,000
...	...	...
...	...	...

mapper

```
def getName (line):
   return  line.split('\t')[1]

def addCounts (hist,  name):
   hist[name] = \
   hist.get(name,default=0) + 1
   return hist

input = open('employees.txt', 'r')

intermediate = map(getName, input)
```

Q: "What is the frequency of each first name?"

Example – Programming model

employees.txt

# LAST	FIRST	SALARY
Smith	John	$90,000
Brown	David	$70,000
Johnson	George	$95,000
Yates	John	$80,000
Miller	Bill	$65,000
Moore	Jack	$85,000
Taylor	Fred	$75,000
Smith	David	$80,000
Harris	John	$90,000
...	...	...
...	...	...

mapper

```
def getName (line):
    return  line.split(‘\t’)[1]
```

reducer

```
def addCounts (hist,  name):
    hist[name] = \
    hist.get(name,default=0) + 1
    return hist

input = open(‘employees.txt’, ‘r’)

intermediate = map(getName, input)

result = reduce(addCounts, \
    intermediate, {})
```

Q: “What is the frequency of each first name?”

Example – Programming model

employees.txt

# LAST	FIRST	SALARY
Smith	John	$90,000
Brown	David	$70,000
Johnson	George	$95,000
Yates	John	$80,000
Miller	Bill	$65,000
Moore	Jack	$85,000
Taylor	Fred	$75,000
Smith	David	$80,000
Harris	John	$90,000
...	...	...
...	...	...

Q: “What is the frequency of each first name?”

mapper

```
def getName (line):
    return (line.split('\t')[1], 1)
```

reducer

```
def addCounts (hist, (name, c)):
    hist[name] = \
    hist.get(name,default=0) + c
    return hist

input = open('employees.txt', 'r')

intermediate = map(getName, input)

result = reduce(addCounts, \
    intermediate, {})
```

Example – Programming model

employees.txt

# LAST	FIRST	SALARY
Smith	John	$90,000
Brown	David	$70,000
Johnson	George	$95,000
Yates	John	$80,000
Miller	Bill	$65,000
Moore	Jack	$85,000
Taylor	Fred	$75,000
Smith	David	$80,000
Harris	John	$90,000
...	...	...
...	...	...

Q: "What is the frequency of each first name?"

mapper

```
def getName (line):
   return (line.split('\t')[1], 1)
```

reducer

```
def addCounts (hist, (name, c)):
   hist[name] = \
   hist.get(name,default=0) + c
   return hist

input = open('employees.txt', 'r')

intermediate = map(getName, input)

result = reduce(addCounts, \
```

Key-value iterators

Example – Programming model

Hadoop / Java

```
public class HistogramJob extends Configured implements Tool {

  public static class FieldMapper extends MapReduceBase
      implements Mapper<LongWritable,Text,Text,LongWritable> {

    private static LongWritable ONE = new LongWritable(1);
    private static Text firstname = new Text();

    @Override
    public void map (LongWritable key, Text value,
        OutputCollector<Text,LongWritable> out, Reporter r) {
      firstname.set(value.toString().split("\t")[1]);
      output.collect(firstname, ONE);
    }
  } // class FieldMapper
```

Example – Programming model

Hadoop / Java

```
public class HistogramJob extends Configured implements Tool {

  public static class FieldMapper extends MapReduceBase
      implements Mapper<LongWritable,Text,Text,LongWritable> {

    private static LongWritable ONE = new LongWritable(1);
    private static Text firstname = new Text();

    @Override
    public void map (LongWritable key, Text value,
        OutputCollector<Text,LongWritable> out, Reporter r) {
      firstname.set(value.toString().split("\t")[1]);
      output.collect(firstname, ONE);
    }
  } // class FieldMapper
```

non-boilerplate

Example – Programming model

Hadoop / Java

```
public class HistogramJob extends Configured implements Tool {

  public static class FieldMapper extends MapReduceBase
      implements Mapper<LongWritable,Text,Text,LongWritable> {
                                                      typed...

    private static LongWritable ONE = new LongWritable(1);
    private static Text firstname = new Text();

    @Override
    public void map (LongWritable key, Text value,
        OutputCollector<Text,LongWritable> out, Reporter r) {
      firstname.set(value.toString().split("\t")[1]);
      output.collect(firstname, ONE);
    }
                                             non-boilerplate
  } // class FieldMapper
```

Example – Programming model

Hadoop / Java

```
public static class LongSumReducer extends MapReduceBase
    implements Mapper<LongWritable,Text,Text,LongWritable> {

  private static LongWritable sum = new LongWritable();

  @Override
  public void reduce (Text key, Iterator<LongWritable> vals,
      OutputCollector<Text,LongWritable> out, Reporter r) {
    long s = 0;
    while (vals.hasNext())
      s += vals.next().get();
    sum.set(s);
    output.collect(key, sum);
  }
} // class LongSumReducer
```

Example – Programming model

Hadoop / Java

```
public static class LongSumReducer extends MapReduceBase
    implements Mapper<LongWritable,Text,Text,LongWritable> {

  private static LongWritable sum = new LongWritable();

  @Override
  public void reduce (Text key, Iterator<LongWritable> vals,
      OutputCollector<Text,LongWritable> out, Reporter r) {
    long s = 0;
    while (vals.hasNext())
      s += vals.next().get();
    sum.set(s);
    output.collect(key, sum);
  }
} // class LongSumReducer
```

Example – Programming model

Hadoop / Java

```
  public int run (String[] args) throws Exception {
    JobConf job = new JobConf(getConf(), HistogramJob.class);
    job.setJobName("Histogram");
    FileInputFormat.setInputPaths(job, args[0]);
    job.setMapperClass(FieldMapper.class);
    job.setCombinerClass(LongSumReducer.class);
    job.setReducerClass(LongSumReducer.class);
    // ...
    JobClient.runJob(job);
    return 0;
  } // run()

  public static main (String[] args) throws Exception {
    ToolRunner.run(new Configuration(), new HistogramJob(), args);
  } // main()
} // class HistogramJob
```

Example – Programming model

Hadoop / Java

```
  public int run (String[] args) throws Exception {
    JobConf job = new JobConf(getConf(), HistogramJob.class);
    job.setJobName("Histogram");
    FileInputFormat.setInputPaths(job, args[0]);
    job.setMapperClass(FieldMapper.class);
    job.setCombinerClass(LongSumReducer.class);
    job.setReducerClass(LongSumReducer.class);
    // ...
    JobClient.runJob(job);
    return 0;
  } // run()

  public static main (String[] args) throws Exception {
    ToolRunner.run(new Configuration(), new HistogramJob(), args)
  } // main()
} // class HistogramJob
```

~ 30 lines = 25 boilerplate (Eclipse) + 5 actual code

MapReduce for…

- Distributed clusters
  - Google's original
  - Hadoop (Apache Software Foundation)
- Hardware
  - SMP/CMP: Phoenix (Stanford)
  - Cell BE
- Other
  - Skynet (in Ruby/DRB)
  - QtConcurrent
  - BashReduce
  - …many more

MapReduce for…

- Distributed clusters
  - Google's original
  - Hadoop (Apache Software Foundation)
- Hardware
  - SMP/CMP: Phoenix (Stanford)
  - Cell BE
- Other
  - Skynet (in Ruby/DRB)
  - QtConcurrent
  - BashReduce
  - …many more

Recap

Quick-n-dirty script	vs	Hadoop
~5 lines of (non-boilerplate) code		
Single machine, local drive		Up to *thousands* of machines and drives

Recap

Quick-n-dirty script	vs	Hadoop
~5 lines of (non-boilerplate) code		
Single machine, local drive		Up to *thousands* of machines and drives

What is hidden to achieve this:

- Data partitioning, placement and replication
- Computation placement (and replication)
- Number of nodes (mappers / reducers)
- …

Recap

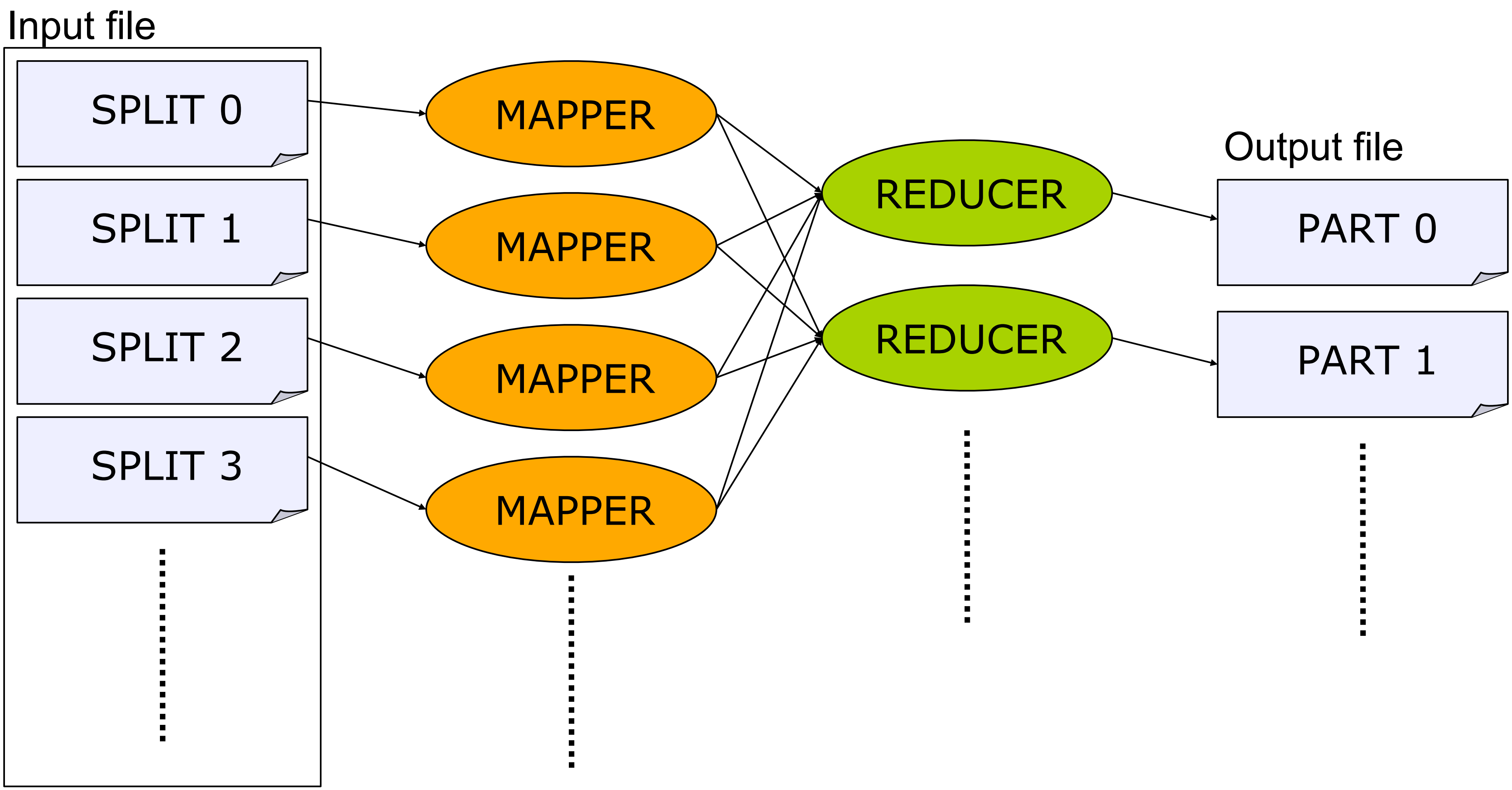
Quick-n-dirty script	vs	Hadoop
~5 lines of (non-boilerplate) code		
Single machine, local drive		Up to *thousands* of machines and drives

What is hidden to achieve this:

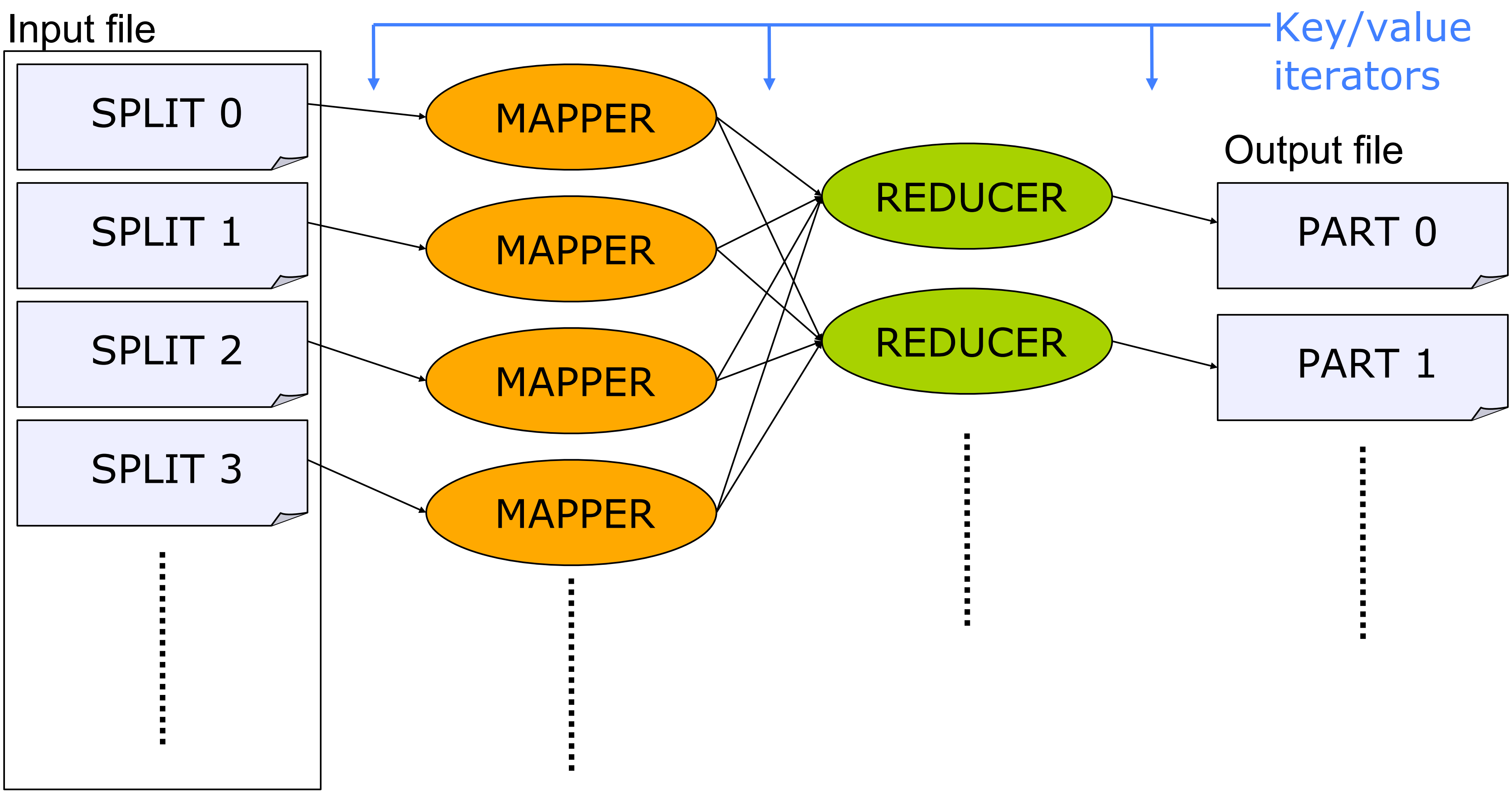
- Data partitioning, placement and replication
- Computation placement (and replication)
- Number of nodes (mappers / reducers)

As a programmer, you don't *need* to know what I'm about to show you next…

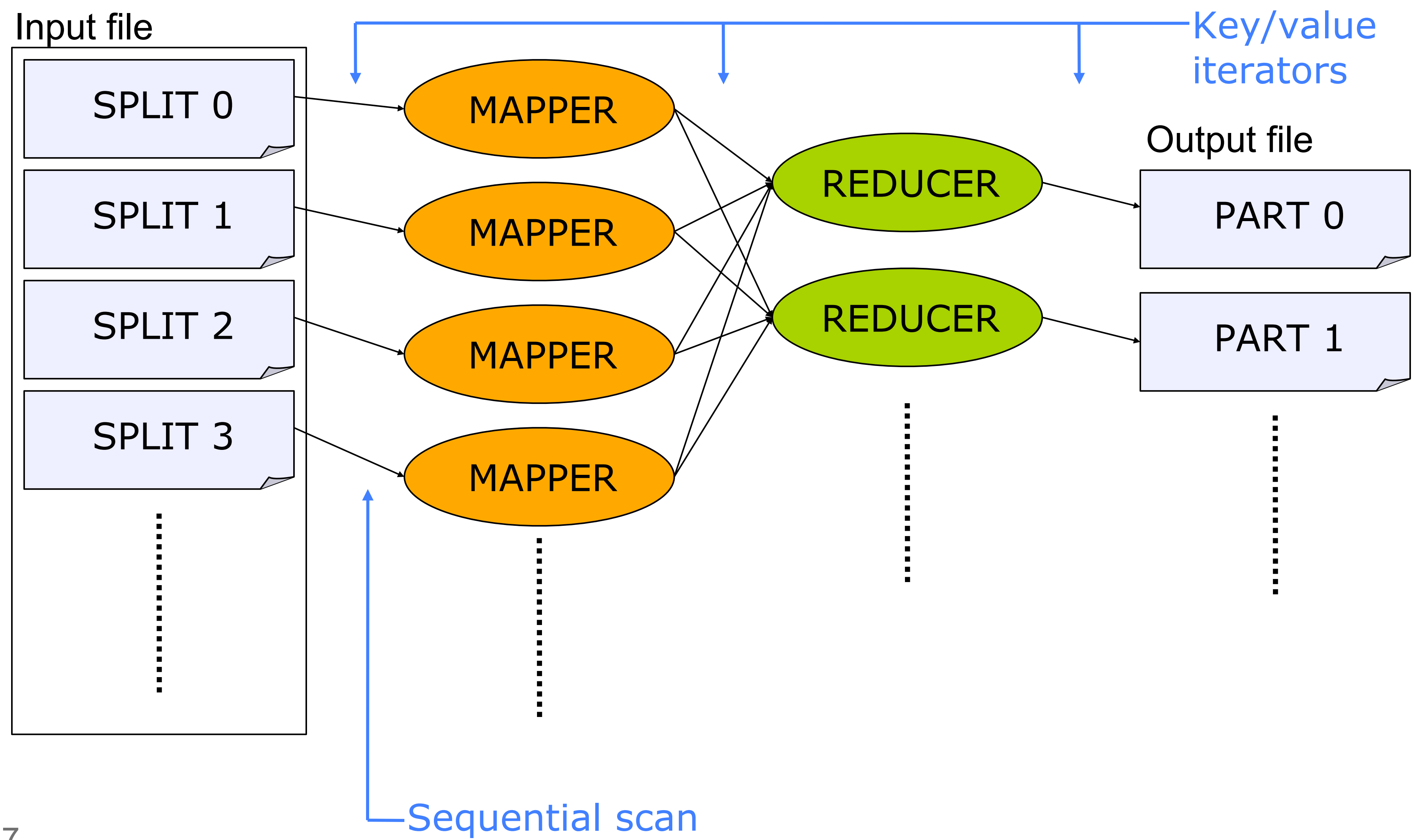
Execution model: Flow



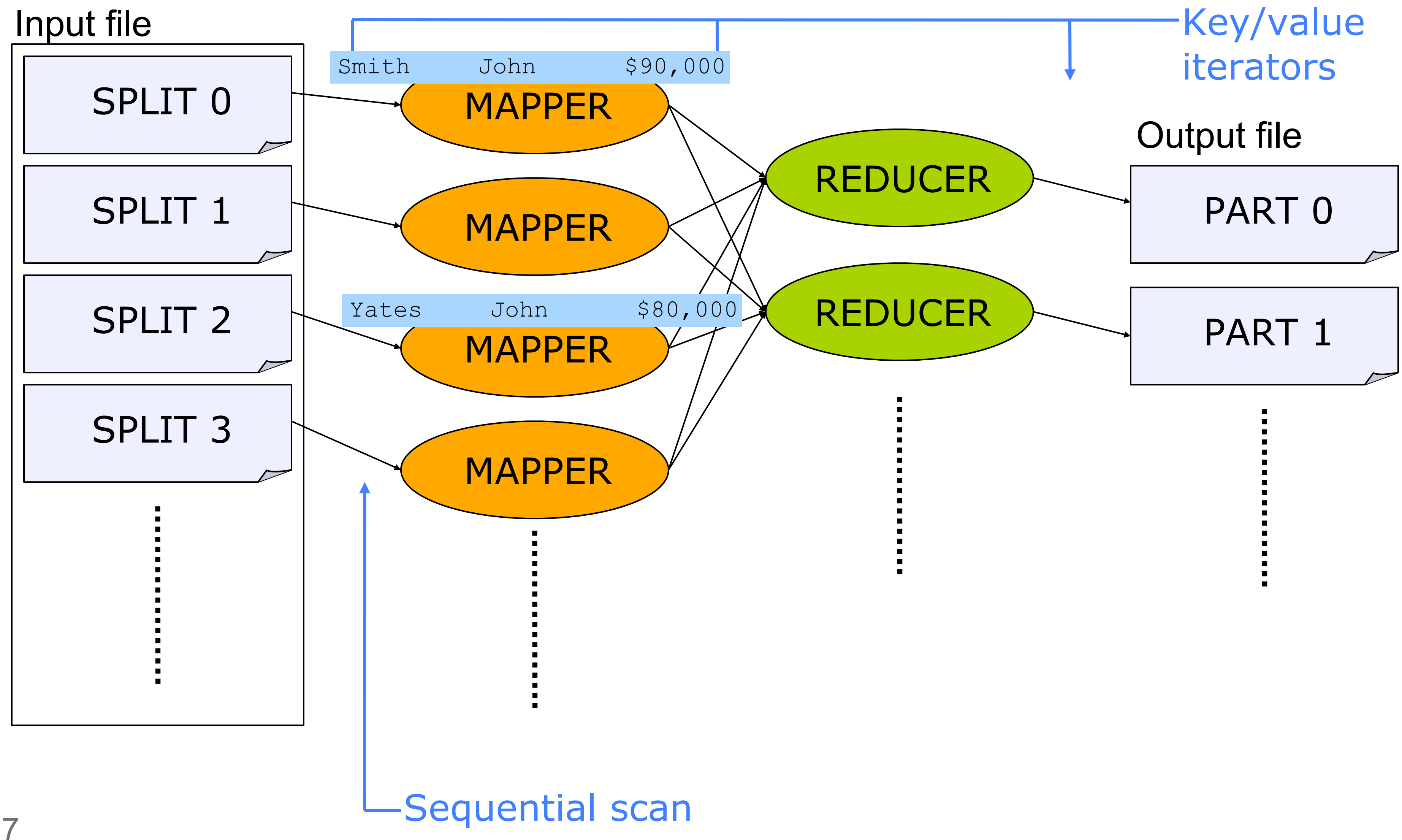
Execution model: Flow



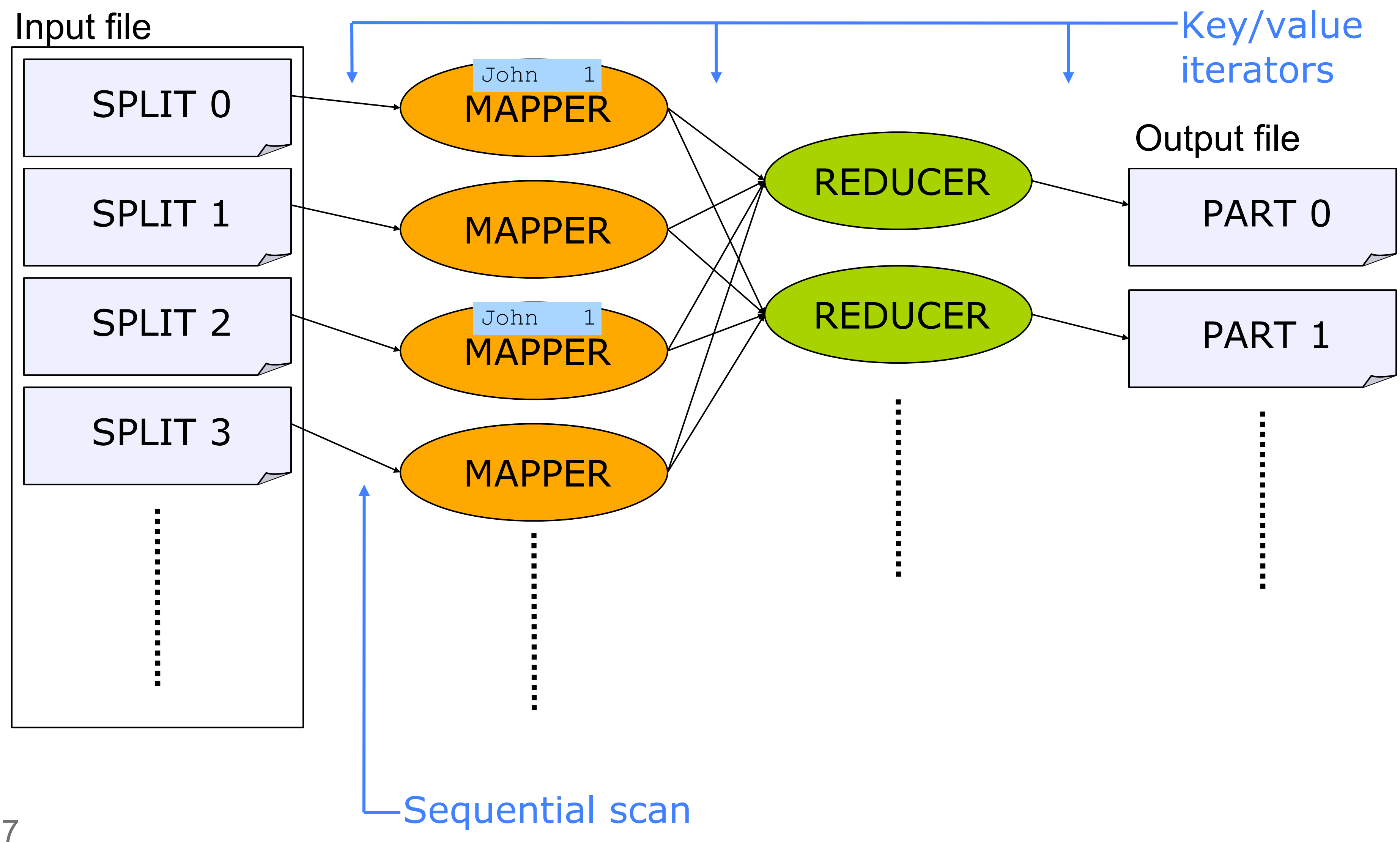
Execution model: Flow



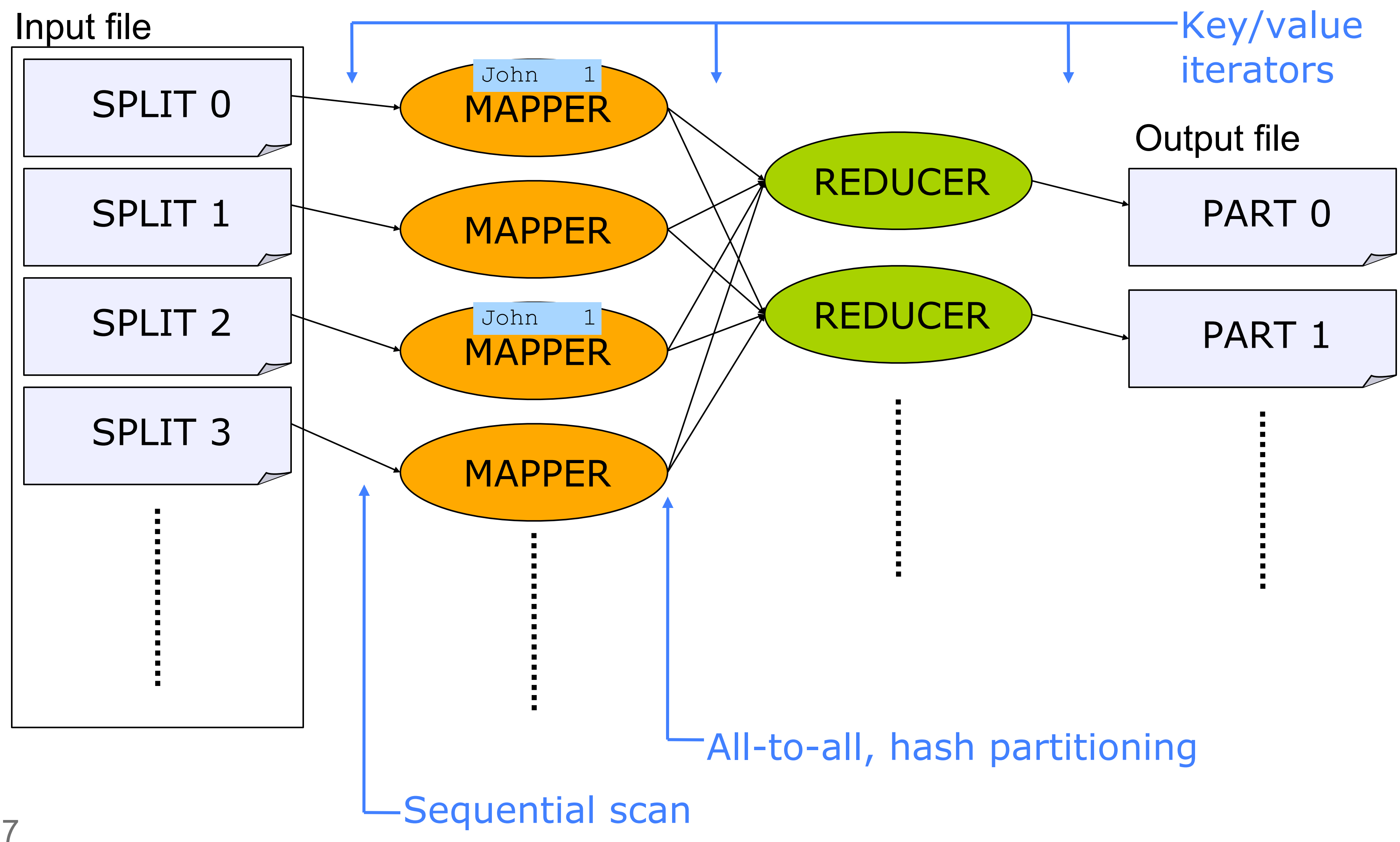
Execution model: Flow



Execution model: Flow

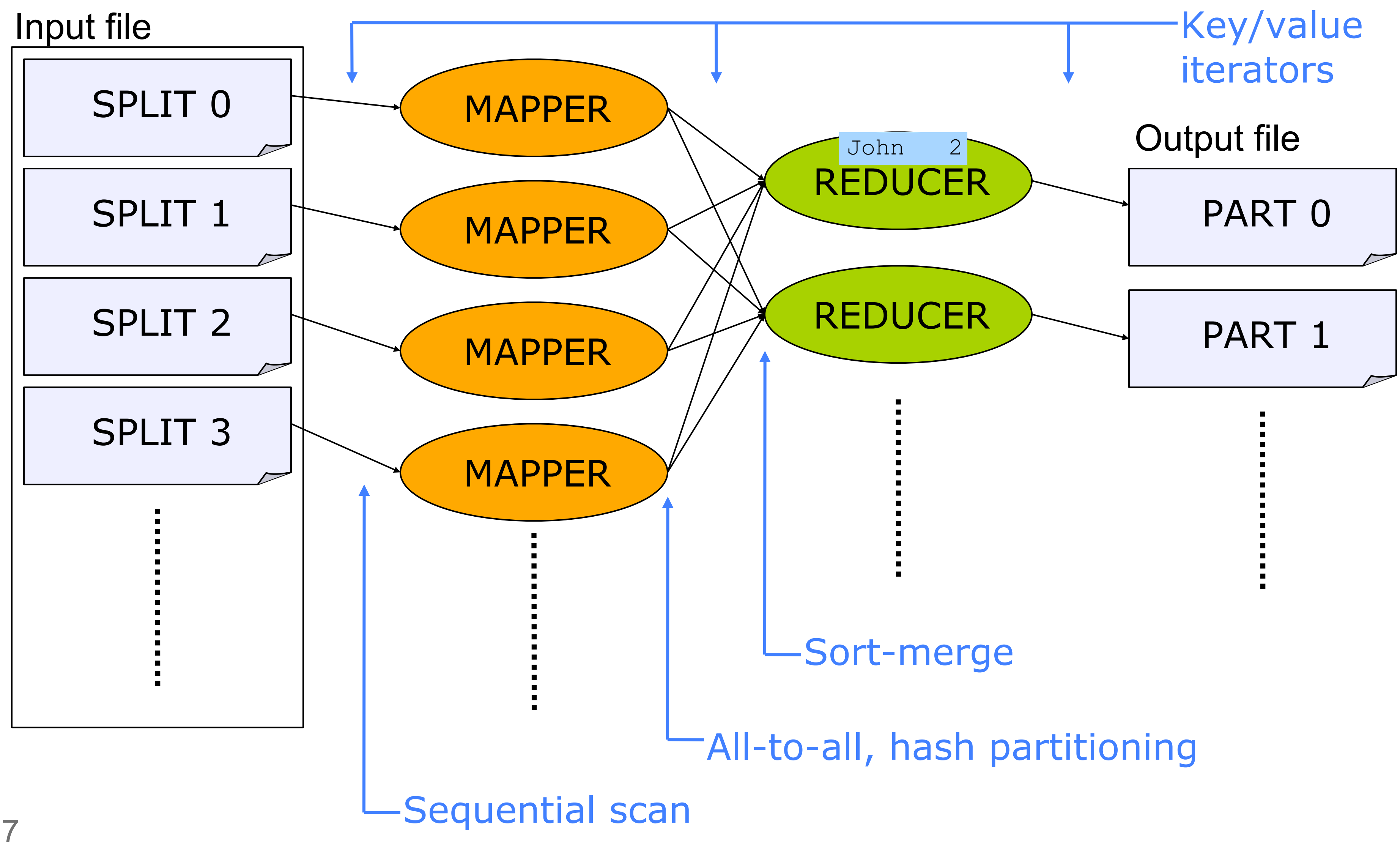


Execution model: Flow

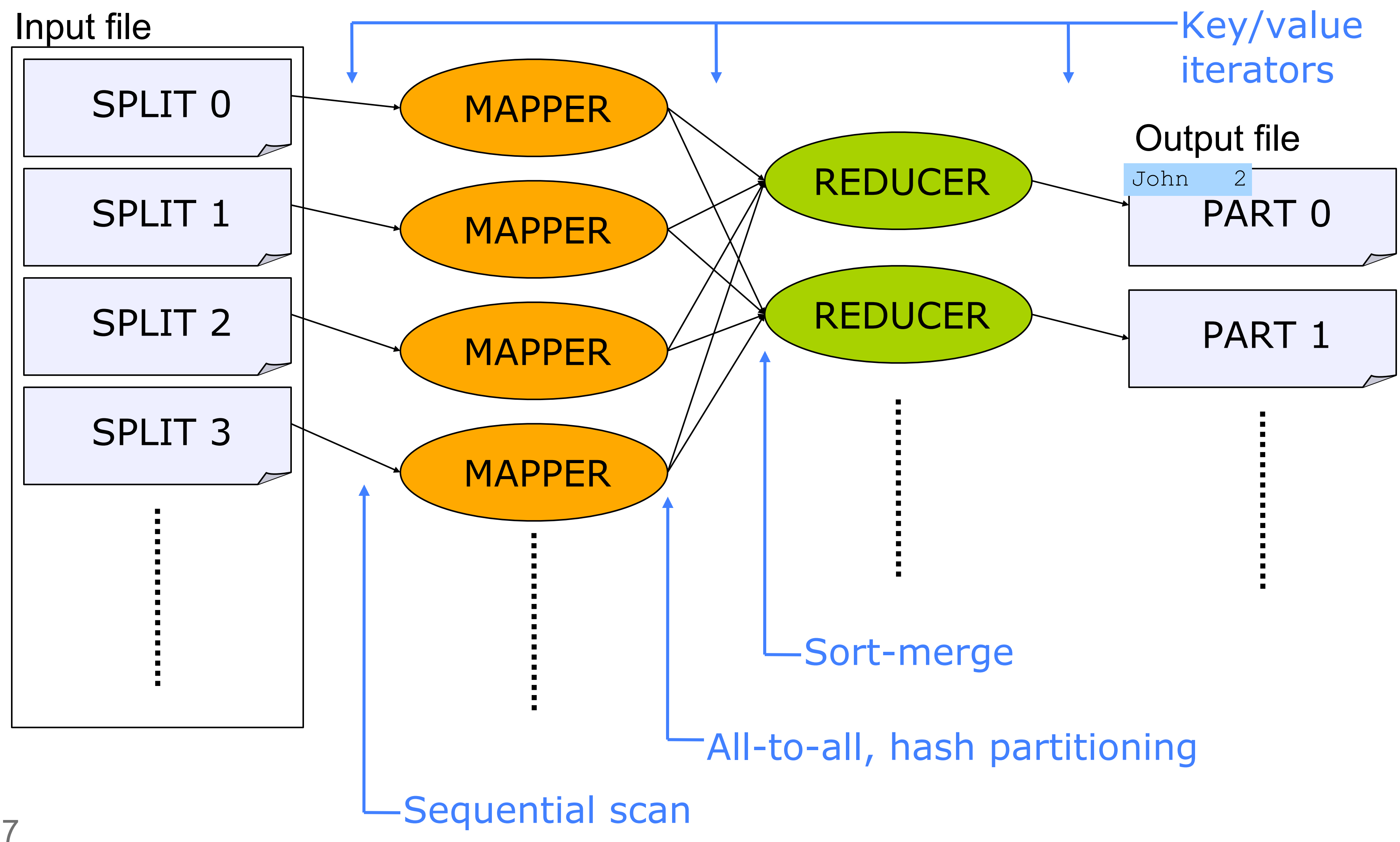


Monday, August 23, 2010

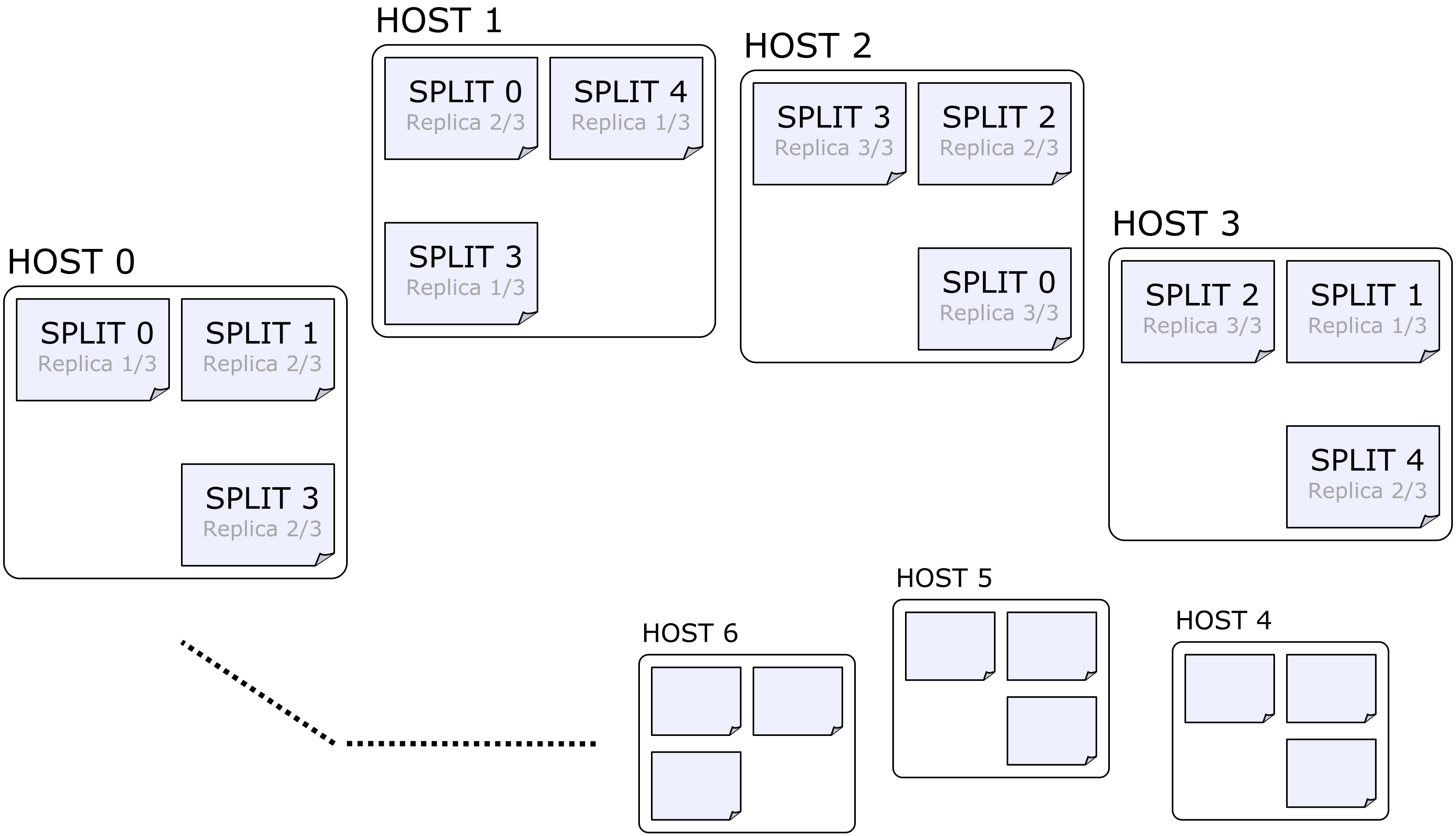
Execution model: Flow



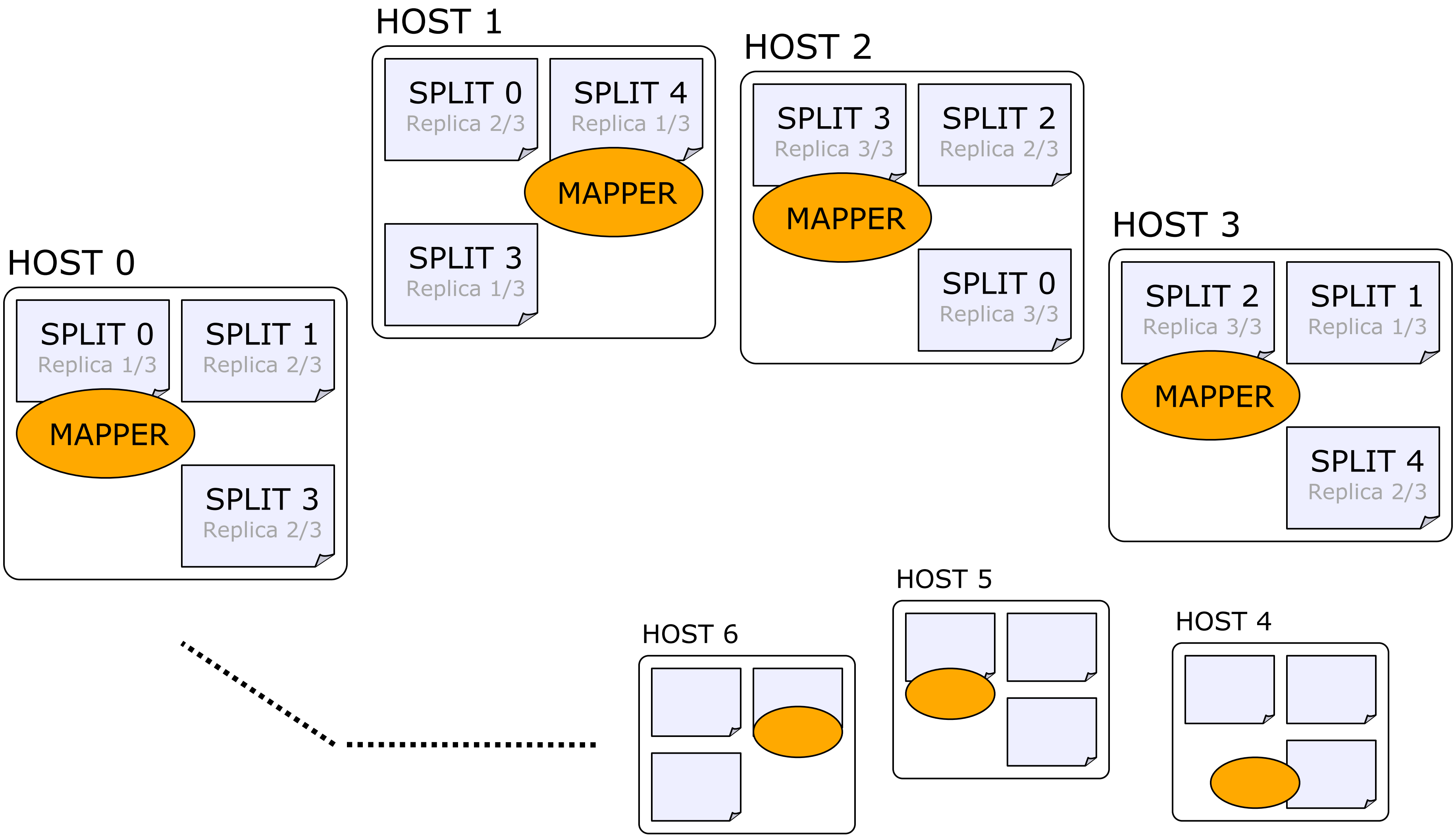
Execution model: Flow



Execution model: Placement

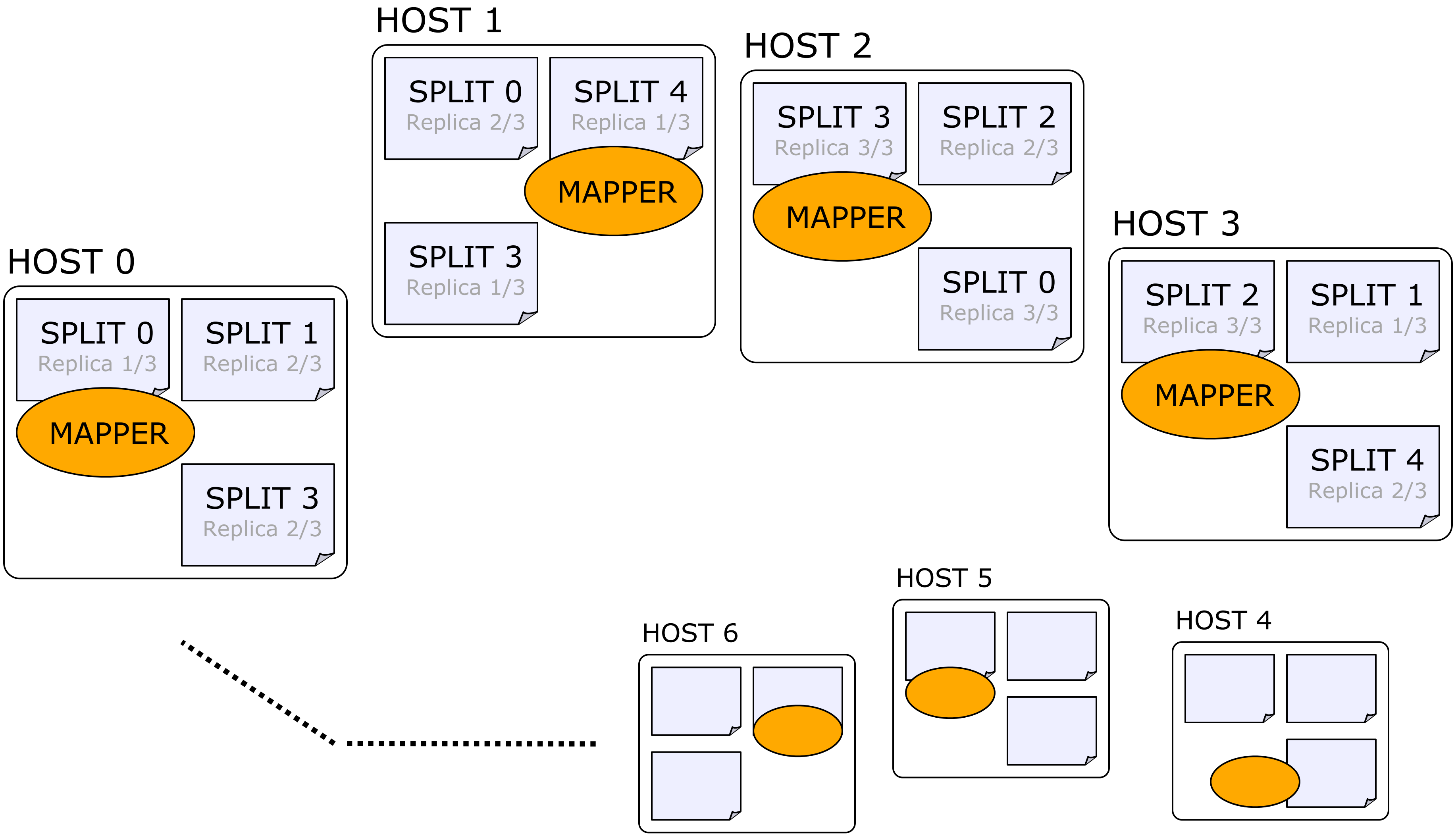


Execution model: Placement

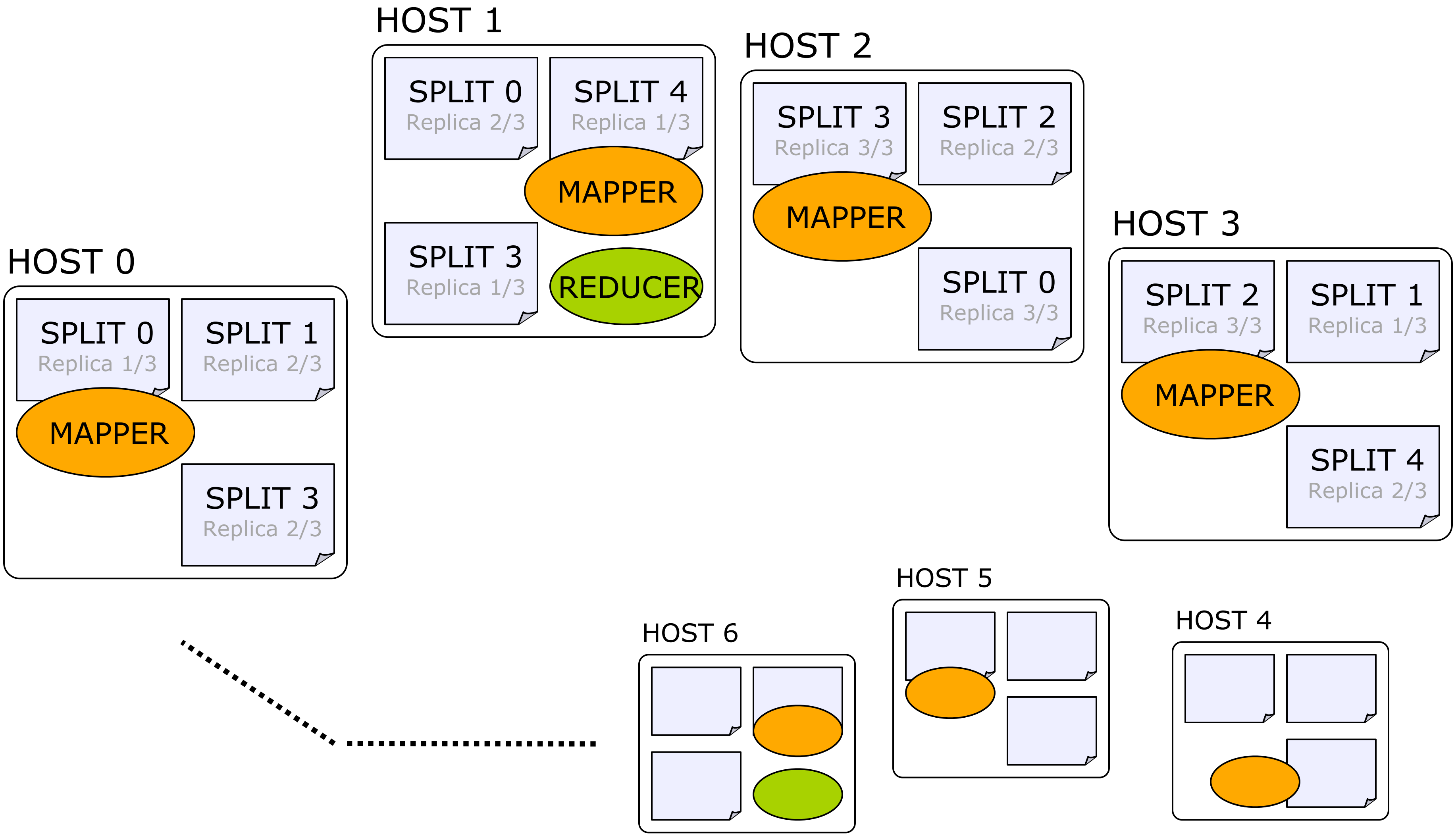


Computation *co-located* with data (as much as possible)

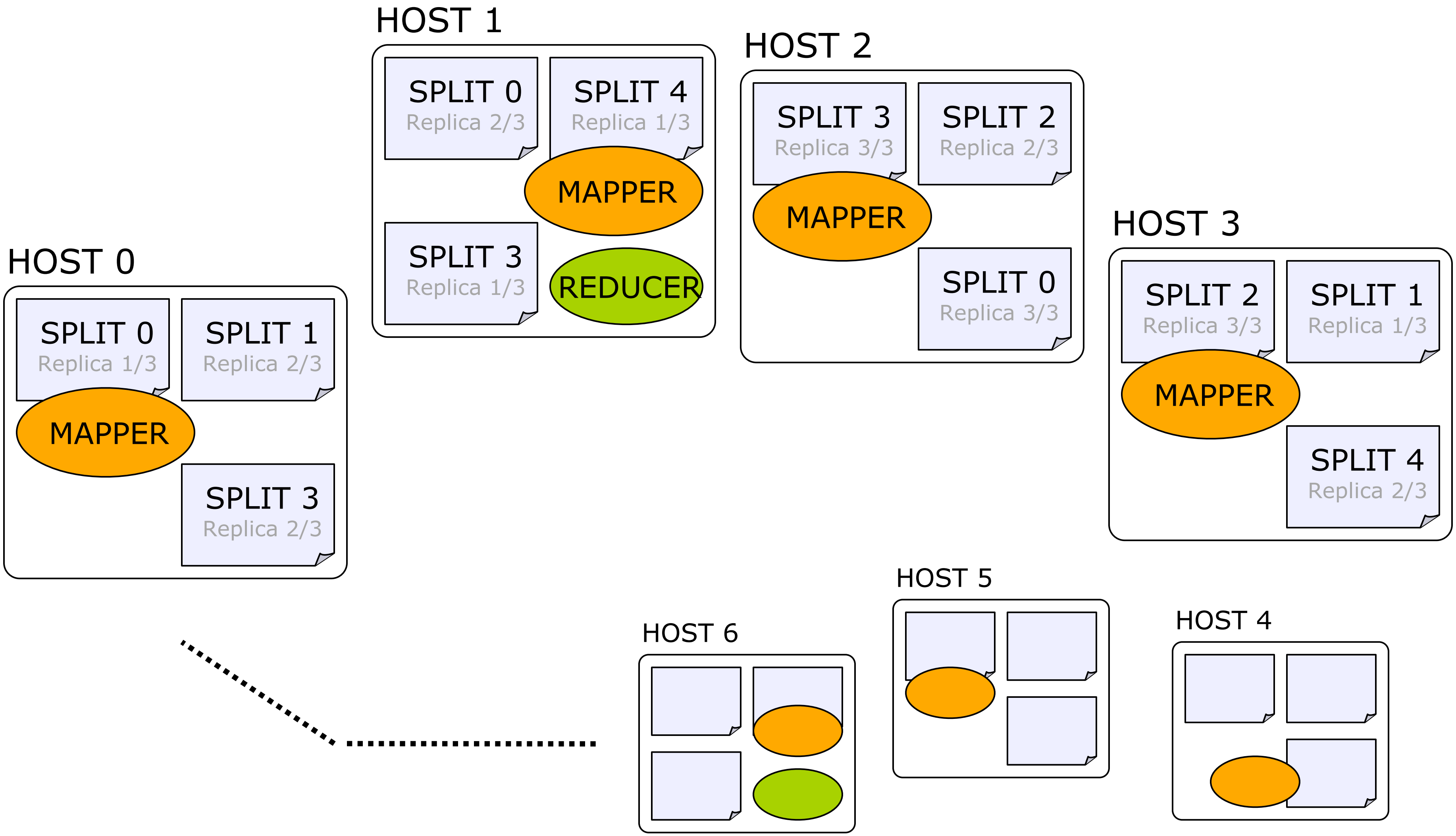
Execution model: Placement



Execution model: Placement

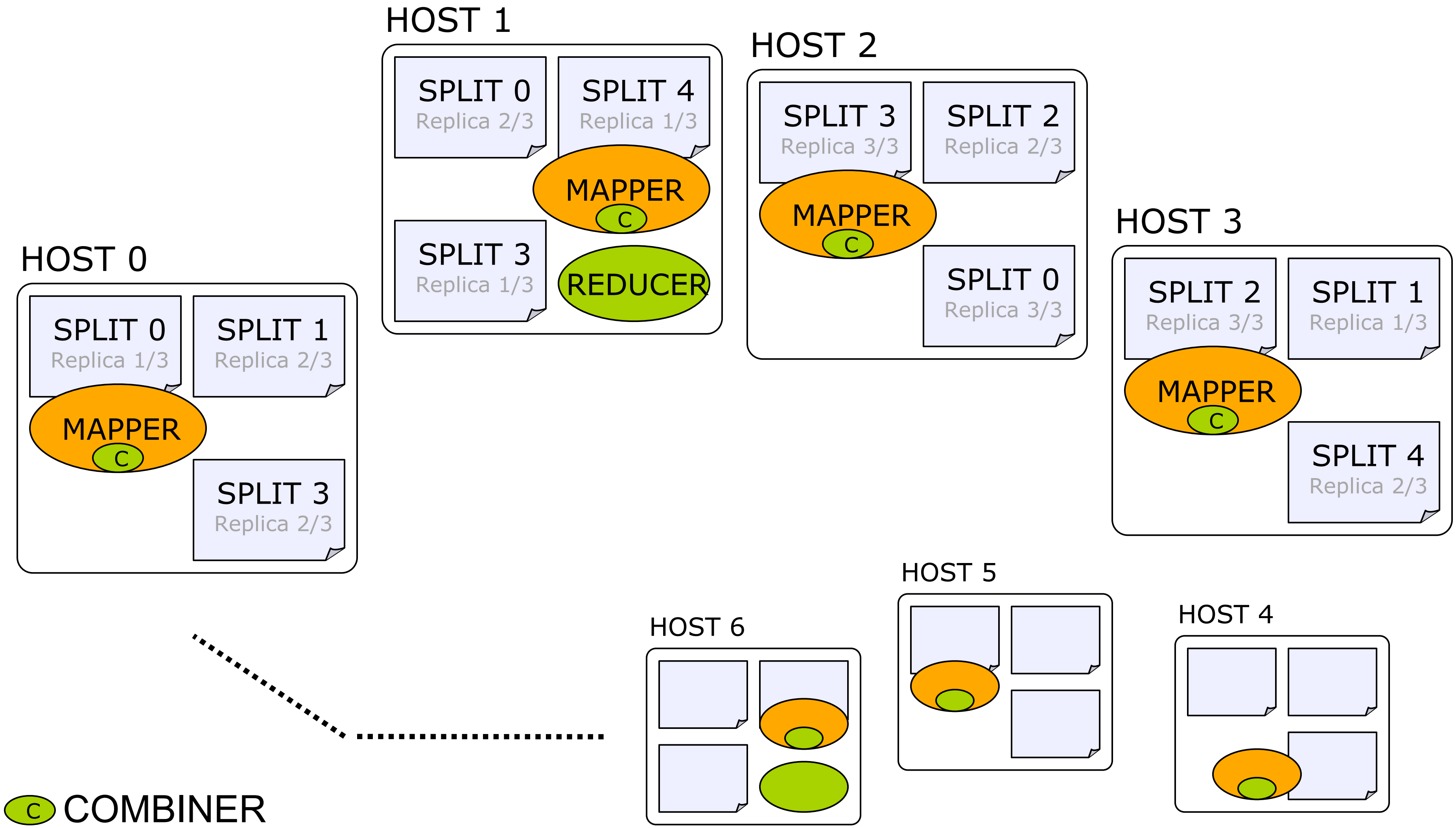


Execution model: Placement



Rack/network-aware

Execution model: Placement



Rack/network-aware

MapReduce Summary

MapReduce Summary

- Simple programming model
- Scalable, fault-tolerant
- Ideal for (pre-)processing large volumes of data

MapReduce Summary

- Simple programming model
- Scalable, fault-tolerant
- Ideal for (pre-)processing large volumes of

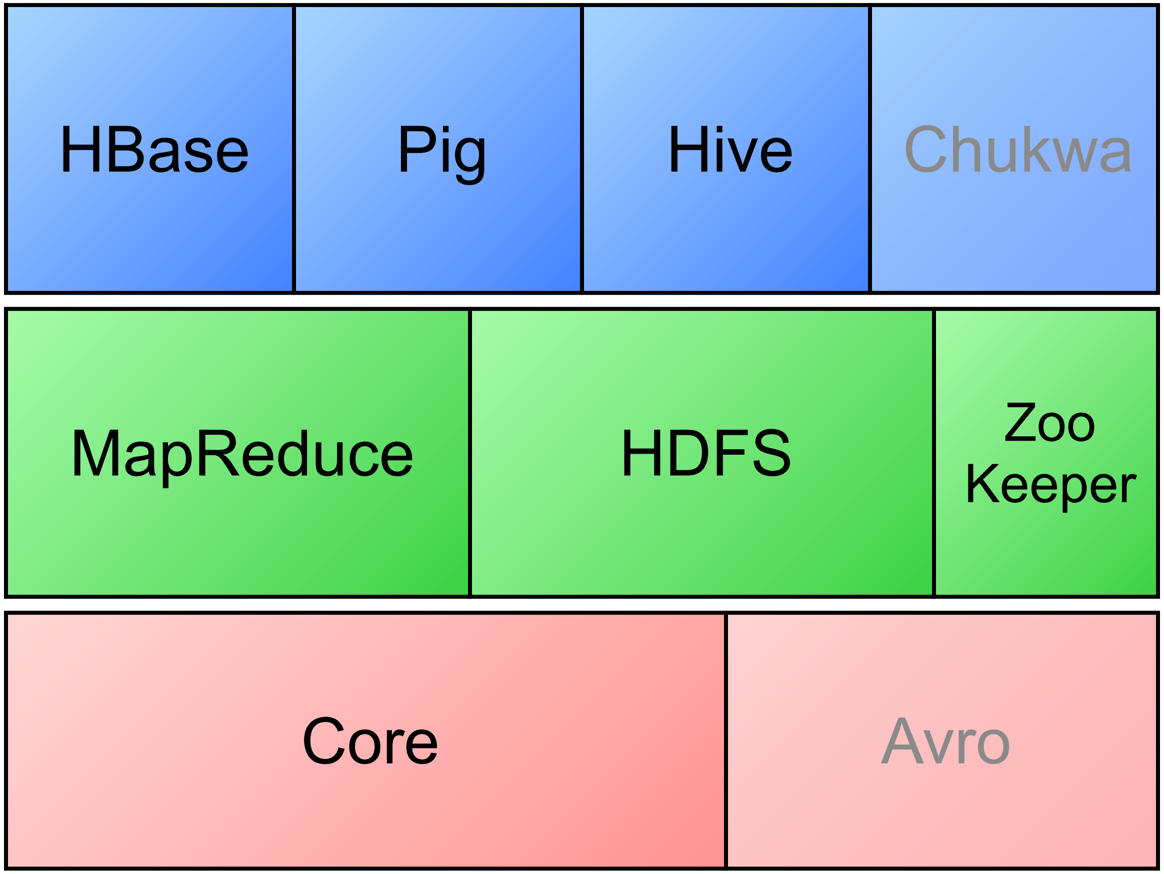
'However, if the data center is the computer, it leads to the even more intriguing question "What is **the equivalent of the ADD instruction** for a data center?" [...] If **MapReduce is the first instruction of the "data center computer"**, I can't wait to see the rest of the instruction set, as well as the data center programming language, the data center operating system, the data center storage systems, and more.'

– David Patterson, "*The Data Center Is The Computer*", CACM, Jan. 2008

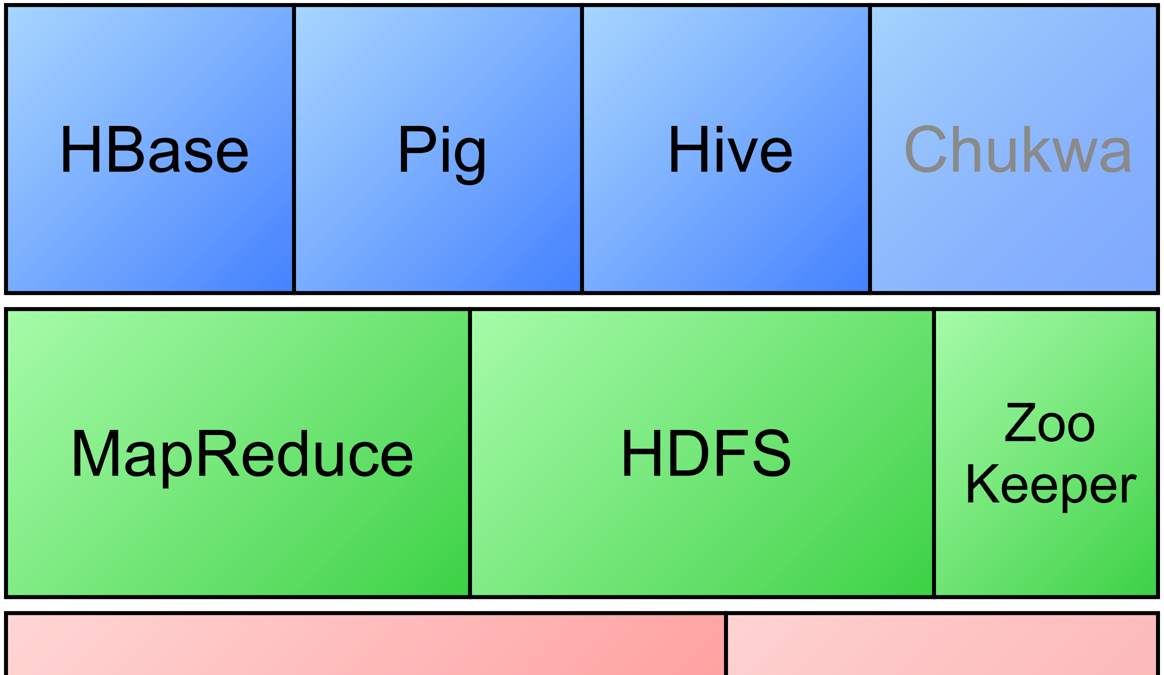
Outline

- Introduction
- MapReduce & distributed storage
- **Hadoop**
  - HBase
  - Pig
  - Cascading
  - Hive
- Summary

Hadoop



Hadoop



Hadoop’s stated mission (Doug Cutting interview):

Commoditize infrastructure for web-scale, data-intensive applications

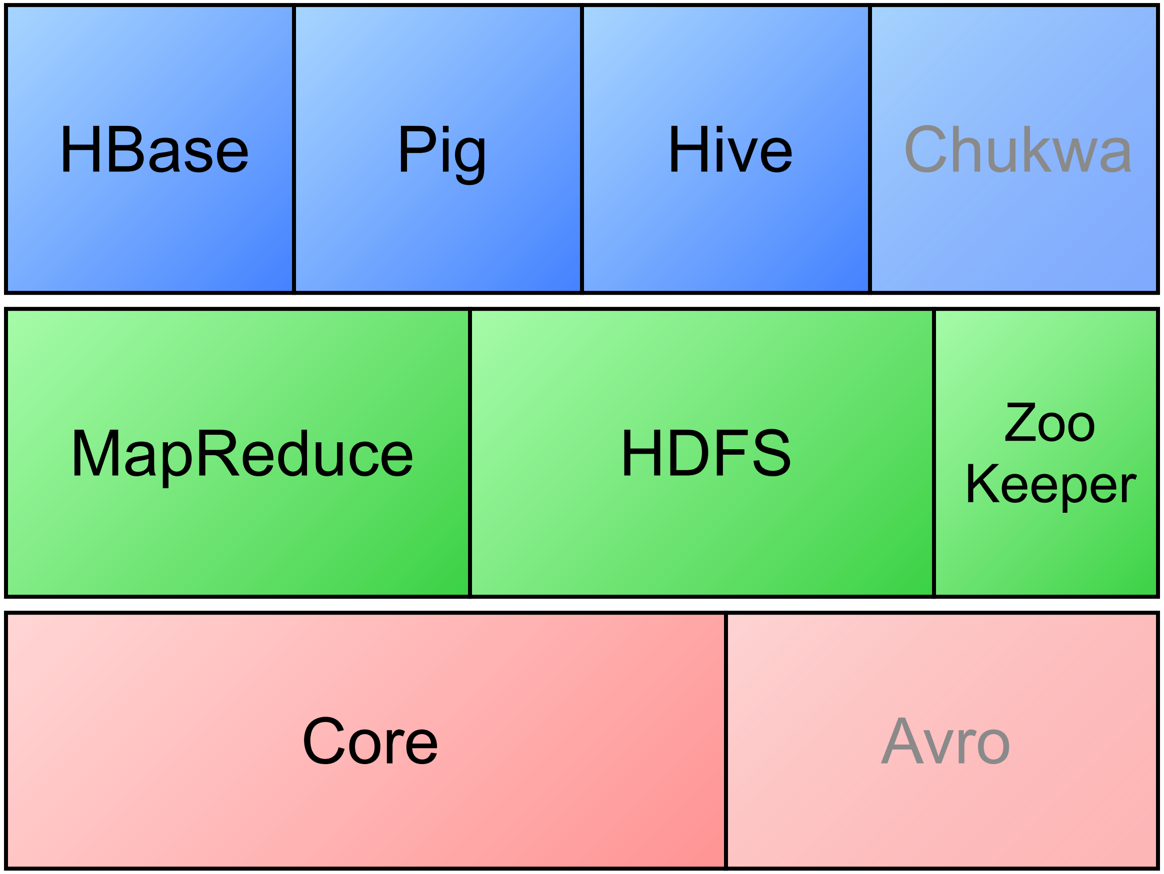
Who uses Hadoop?

- Yahoo!
- Facebook
- Last.fm
- Rackspace
- Digg

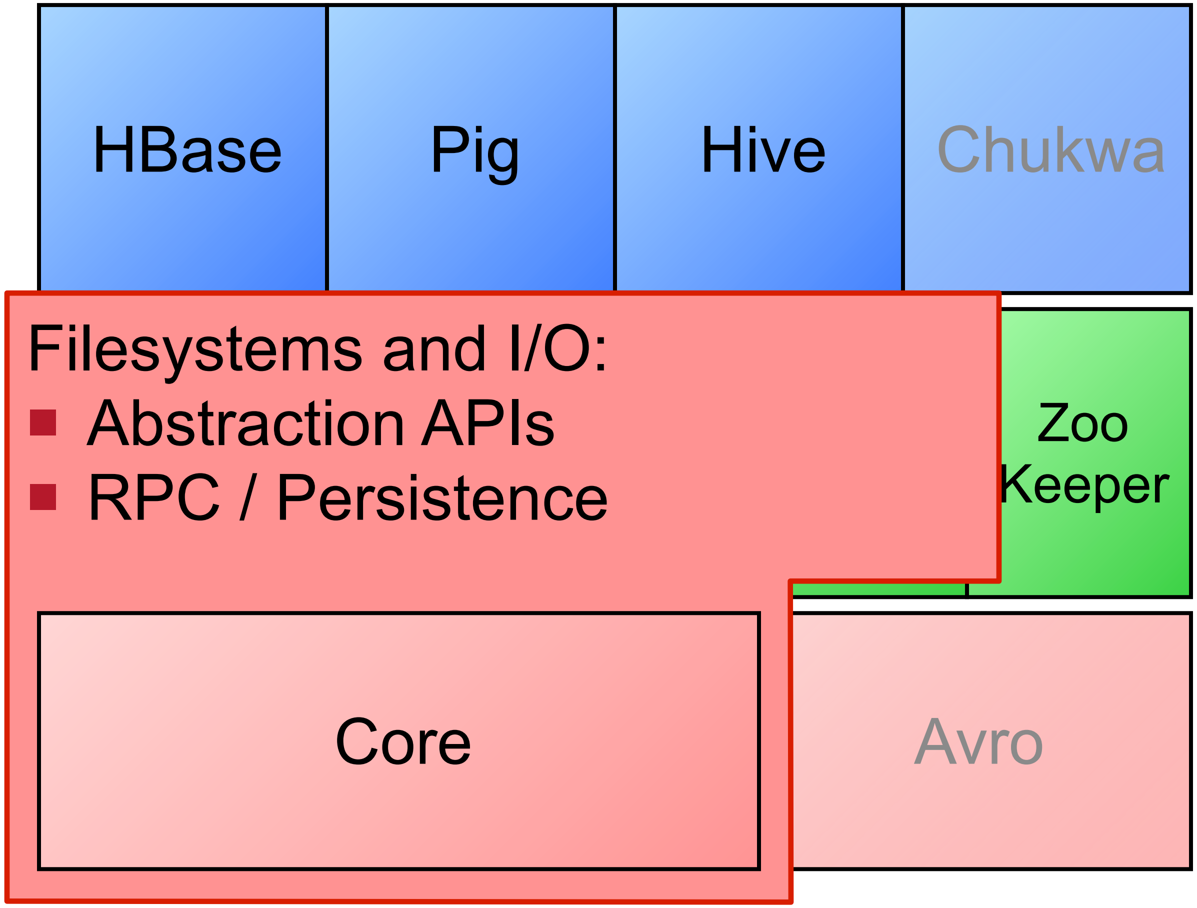
- Apache Nutch

- … more in part 3

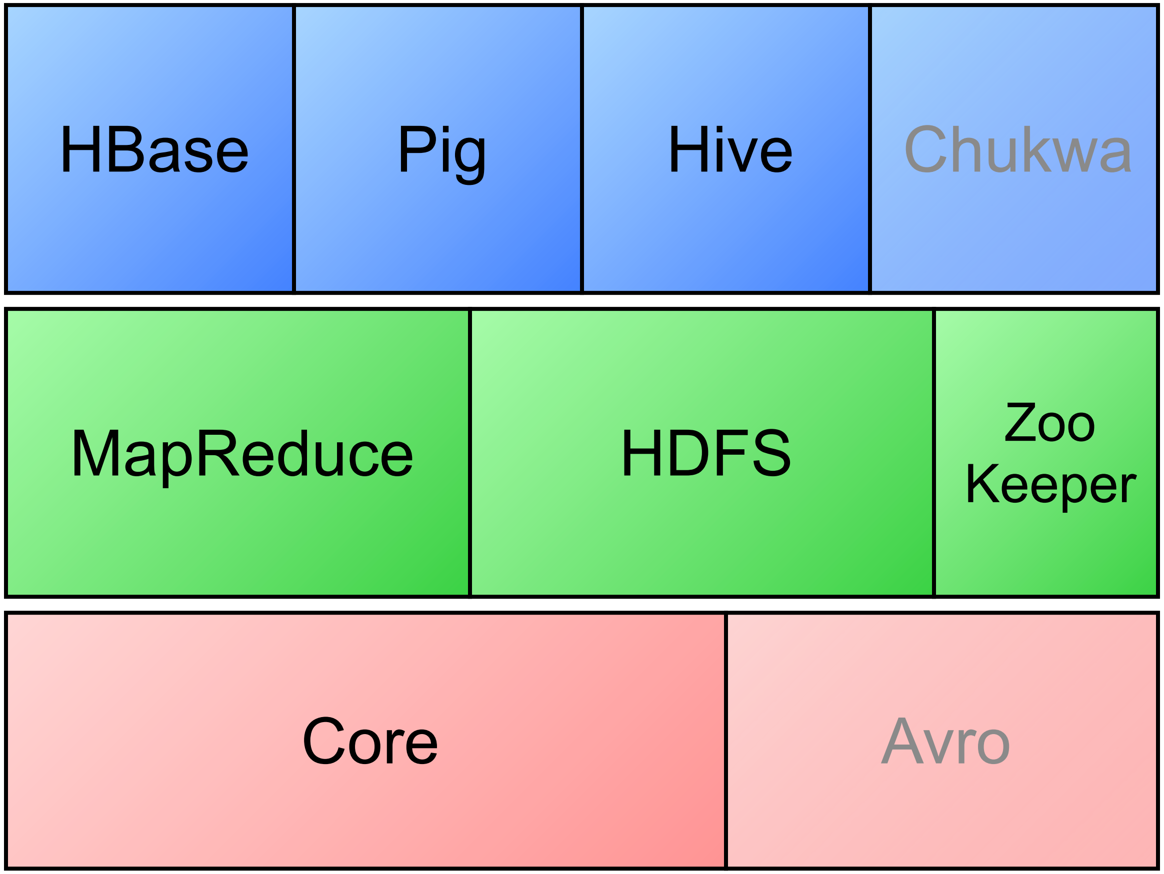
Hadoop



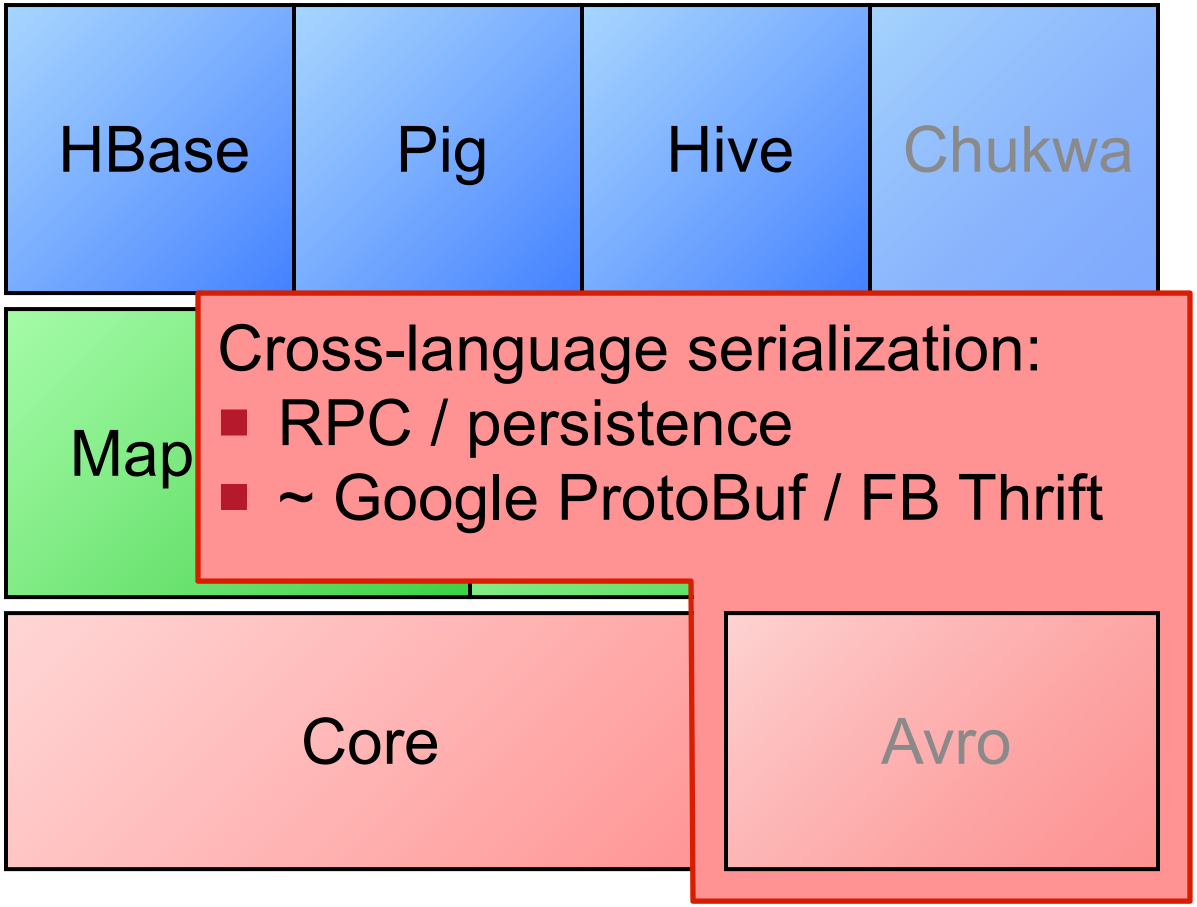
Hadoop


HBase
Pig
Hive
Chukwa
Filesystems and I/O:
Abstraction APIs
RPC / Persistence
Zoo Keeper
Core
Avro

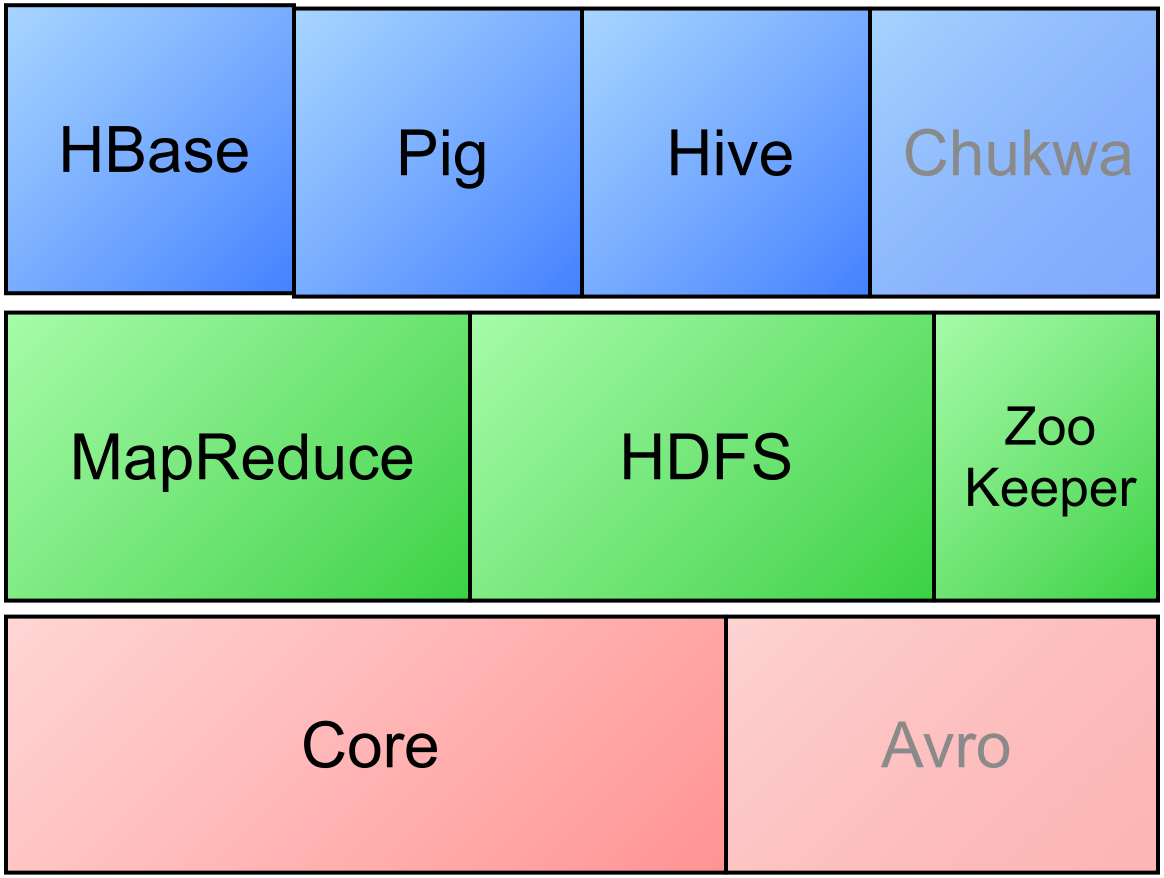
Hadoop



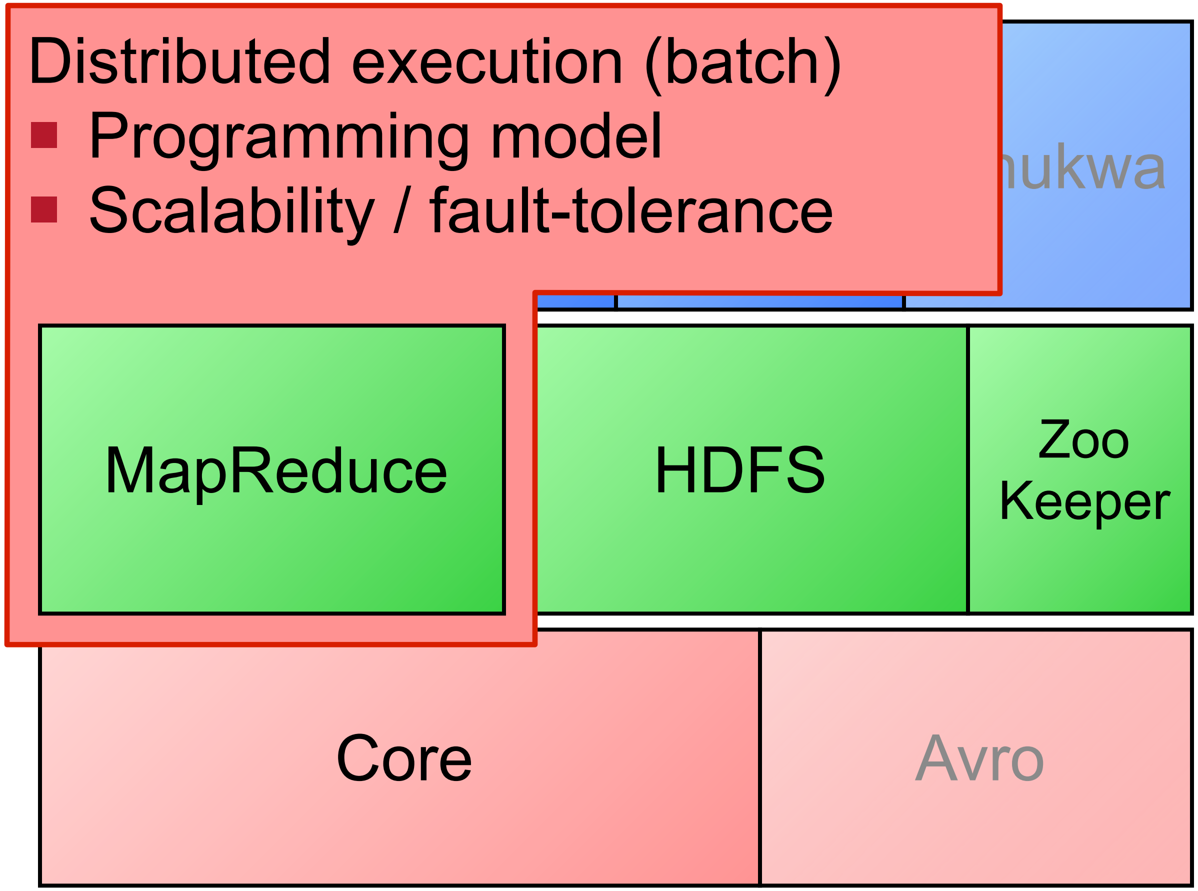
Hadoop


HBase
Pig
Hive
Chukwa
Map
Cross-language serialization:
RPC / persistence
~ Google ProtoBuf / FB Thrift
Core
Avro

Hadoop

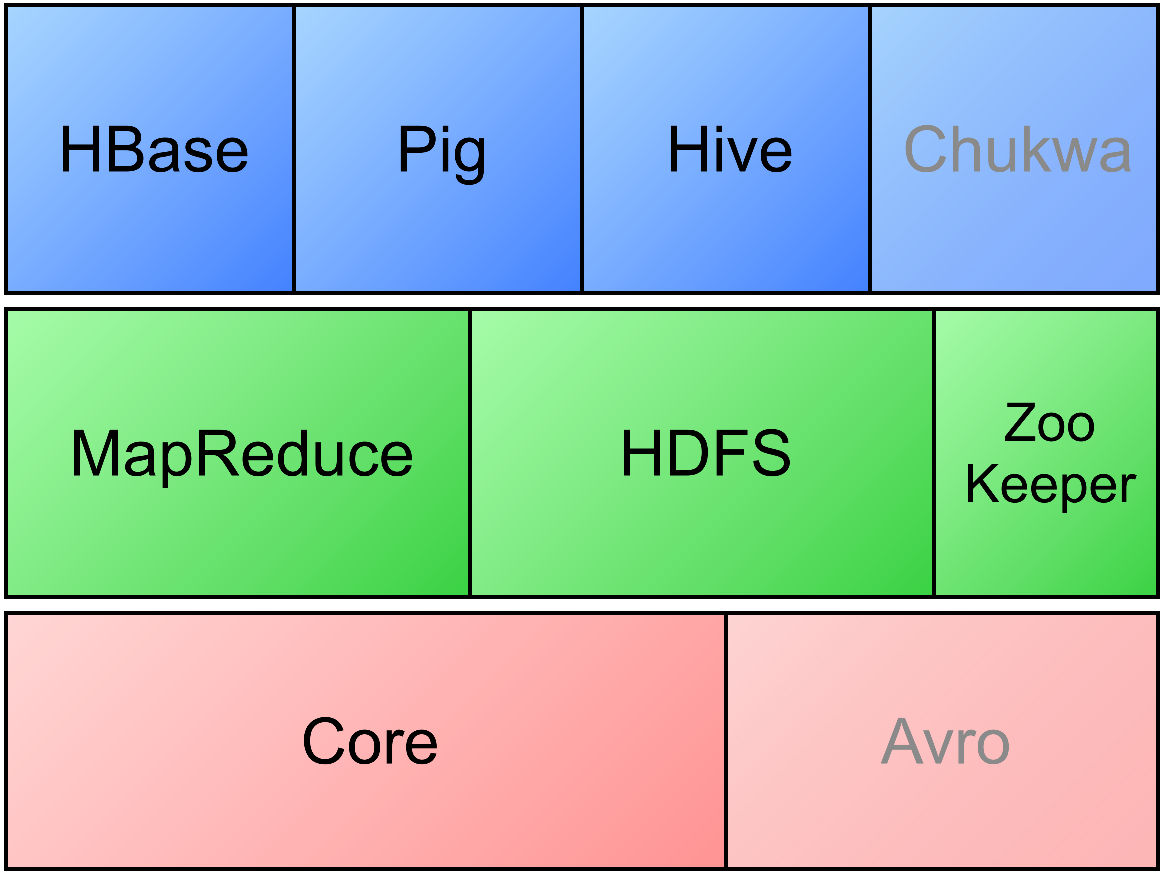


Hadoop

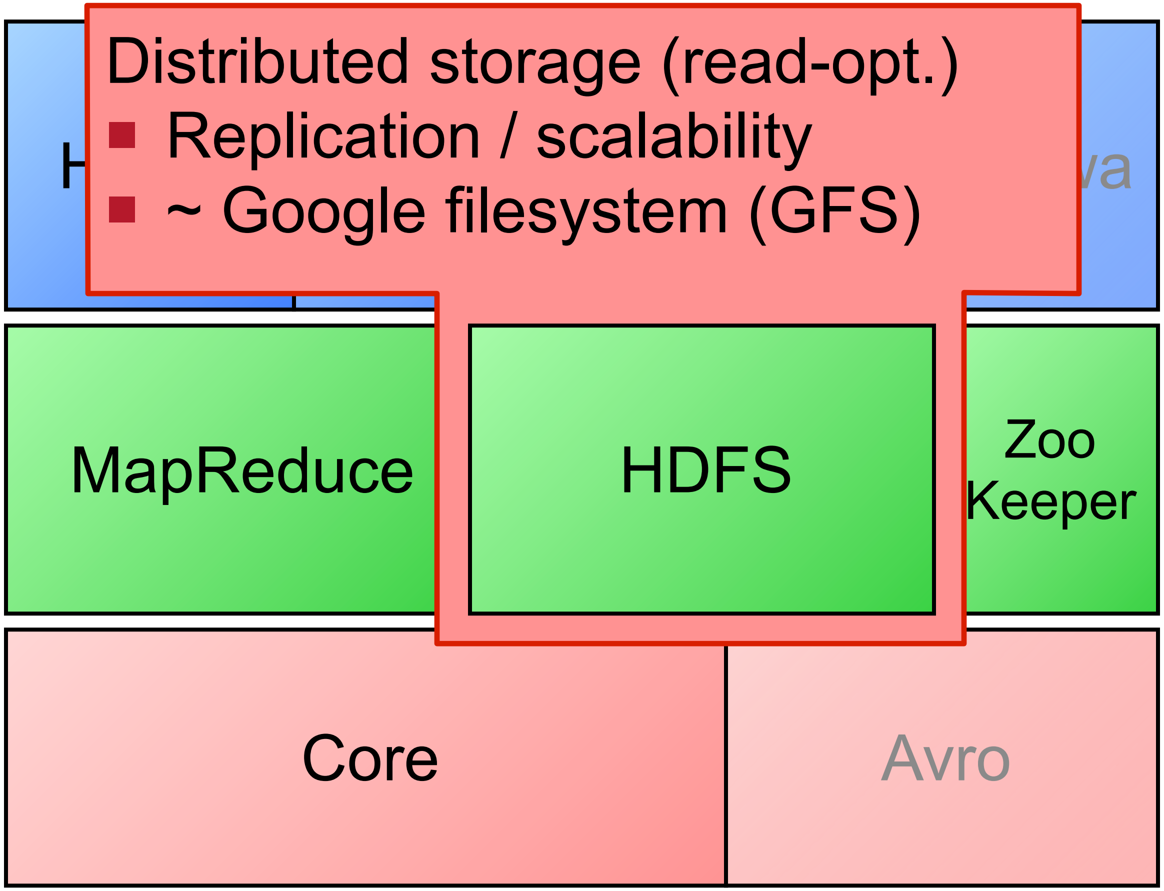


Distributed execution (batch)
Programming model
Scalability / fault-tolerance
ukwa
MapReduce
HDFS
Zoo
Keeper
Core
Avro

Hadoop

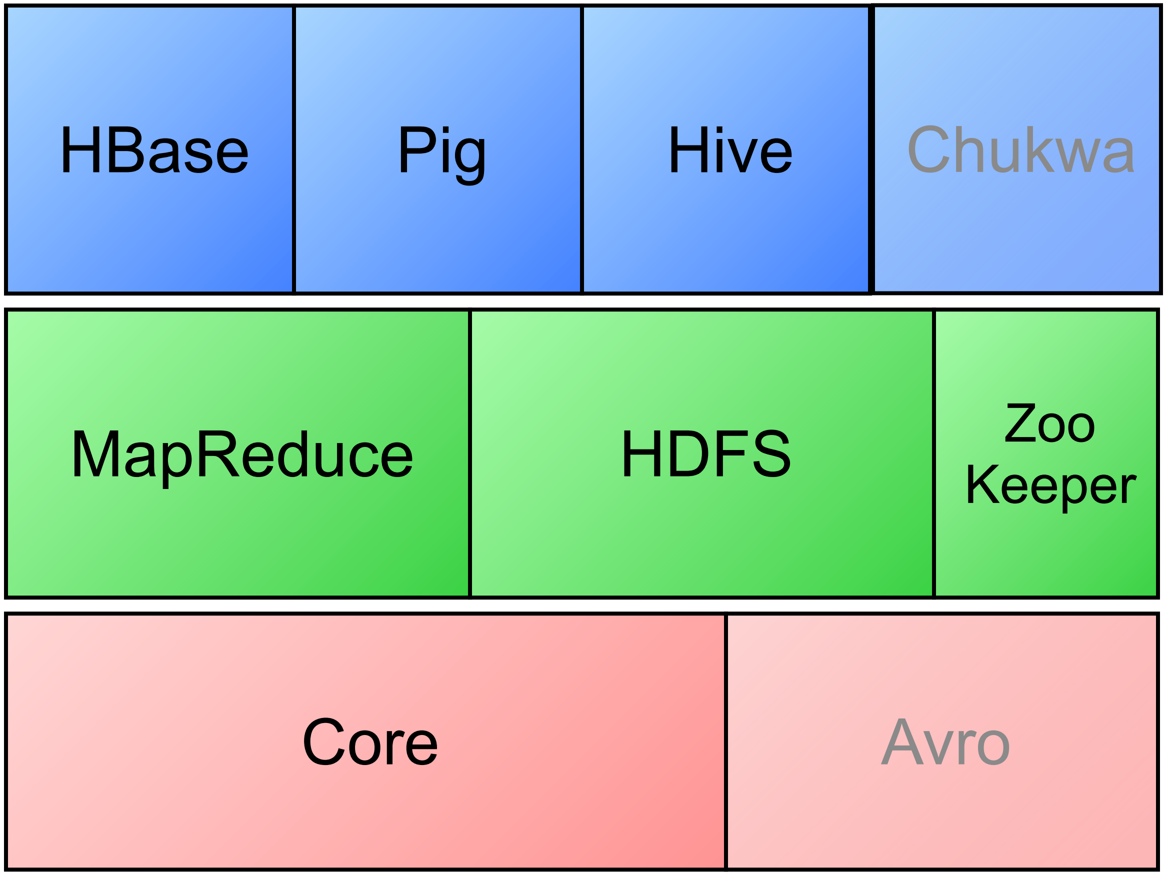


Hadoop

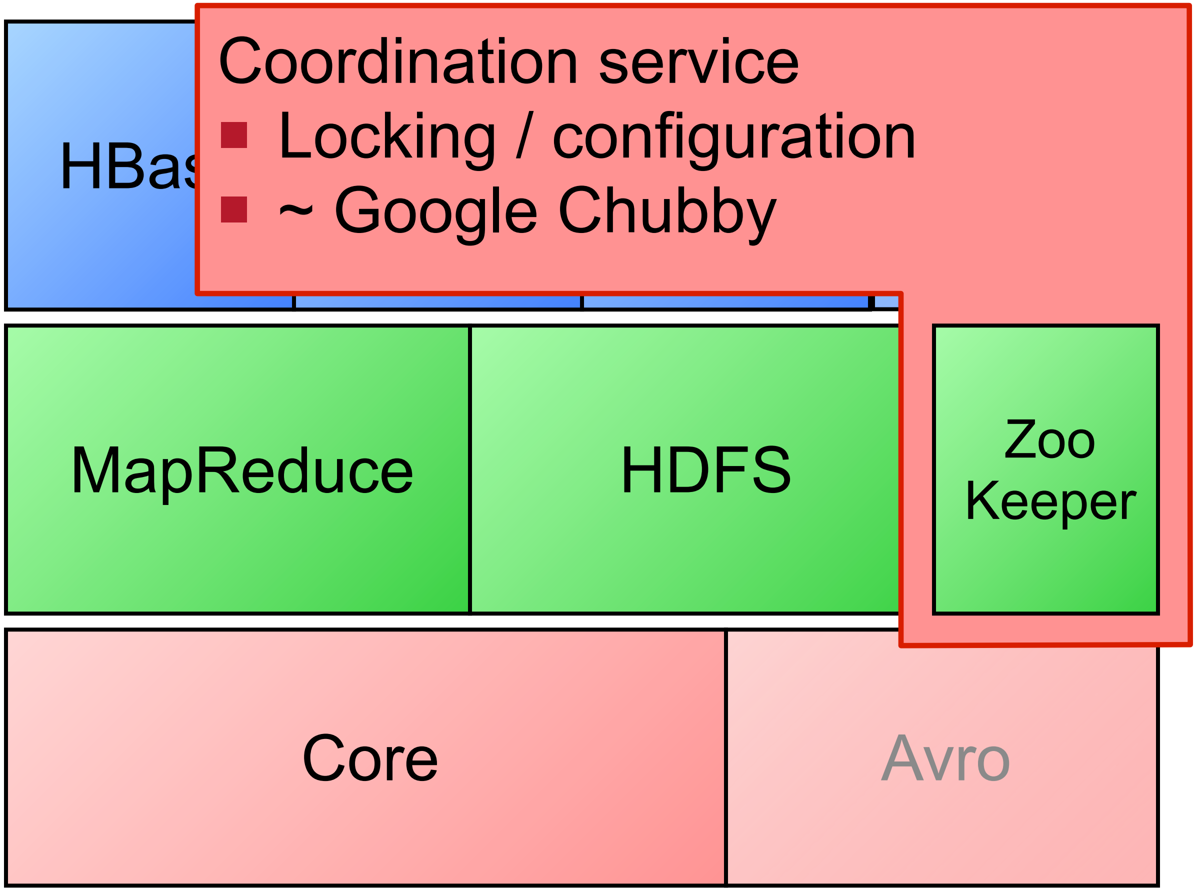


Distributed storage (read-opt.)
Replication / scalability
~ Google filesystem (GFS)
MapReduce
HDFS
Zoo Keeper
Core
Avro

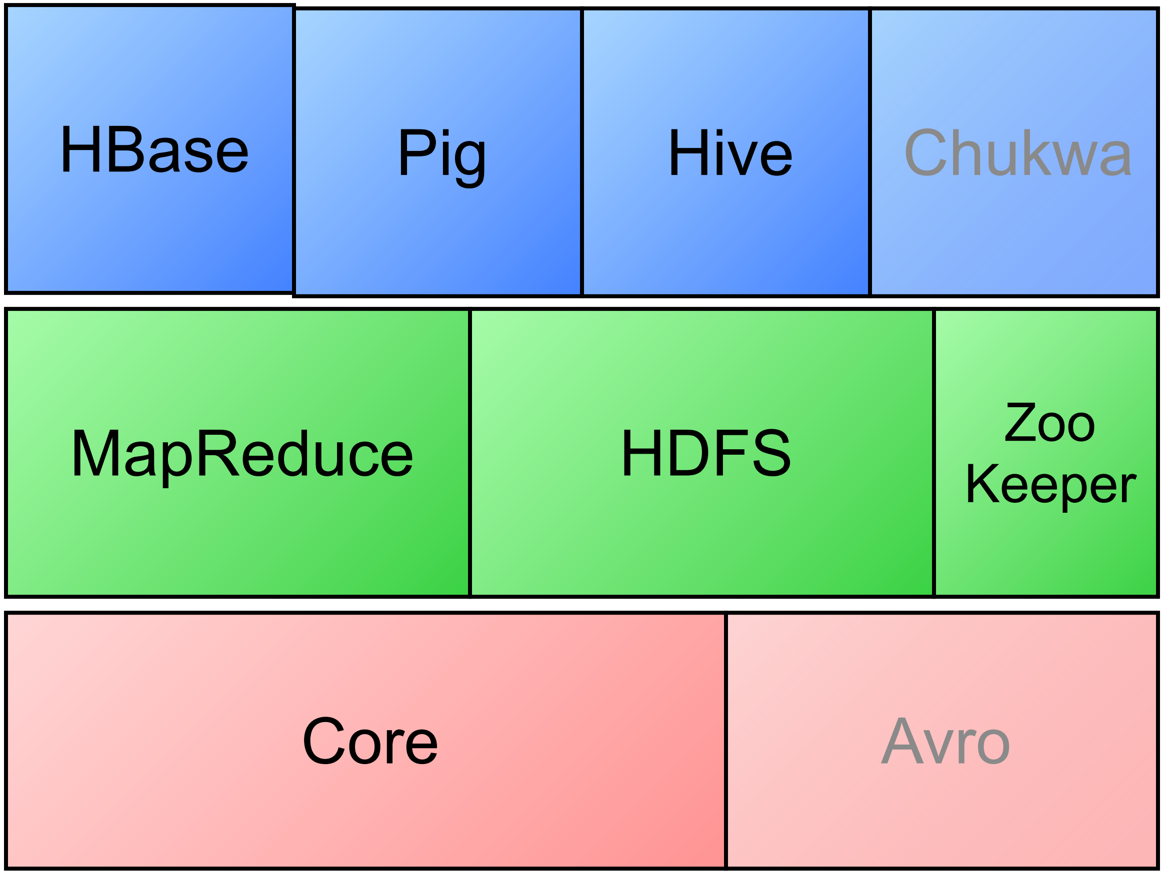
Hadoop



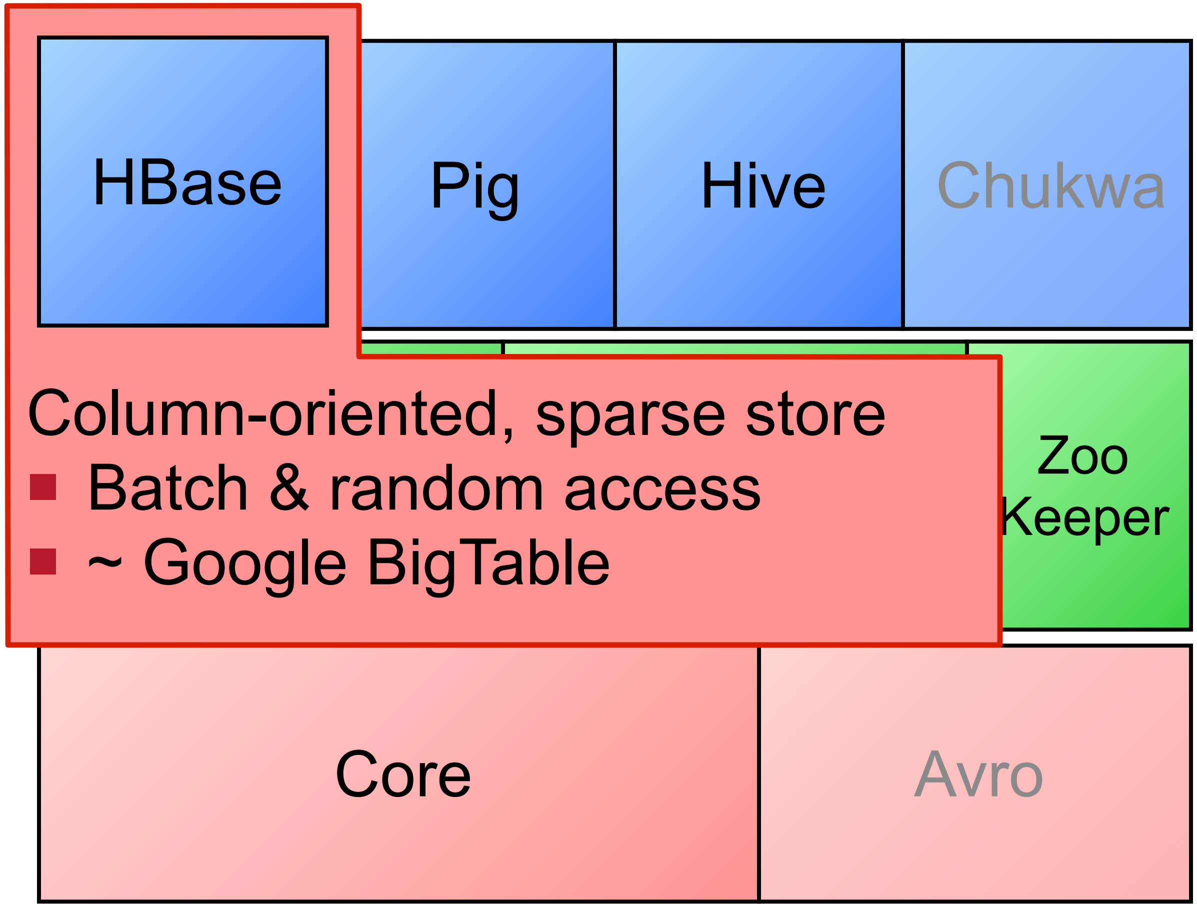
Hadoop


HBas
Coordination service
Locking / configuration
~ Google Chubby
MapReduce
HDFS
Zoo
Keeper
Core
Avro

Hadoop



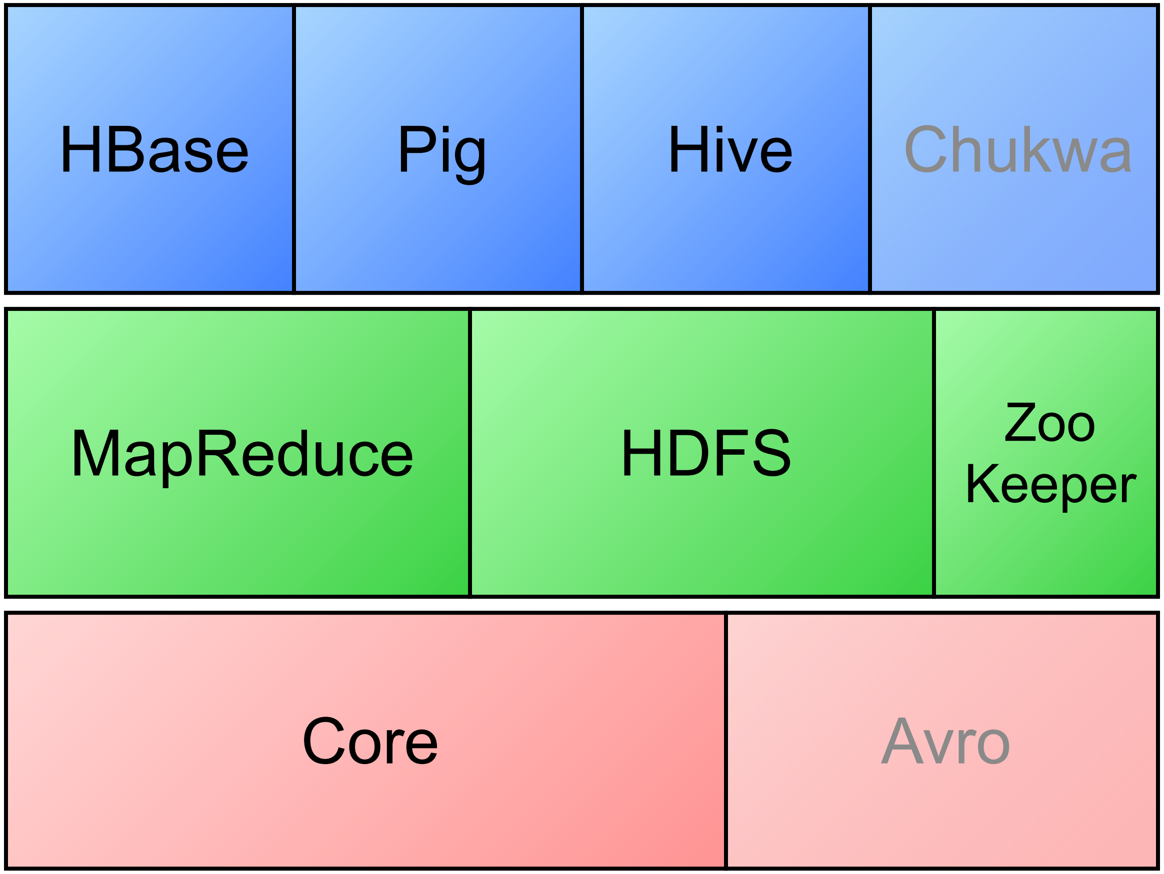
Hadoop



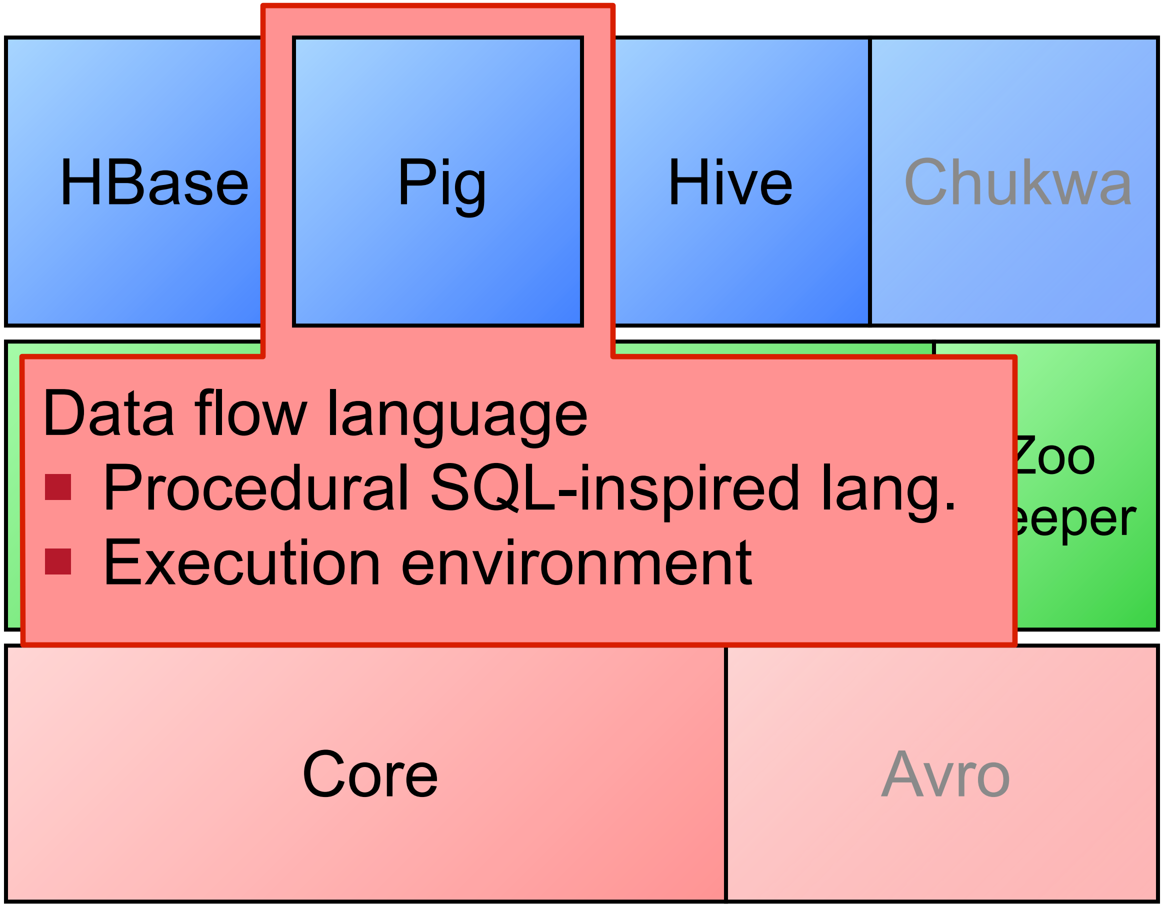
Column-oriented, sparse store

- Batch & random access
- ~ Google BigTable

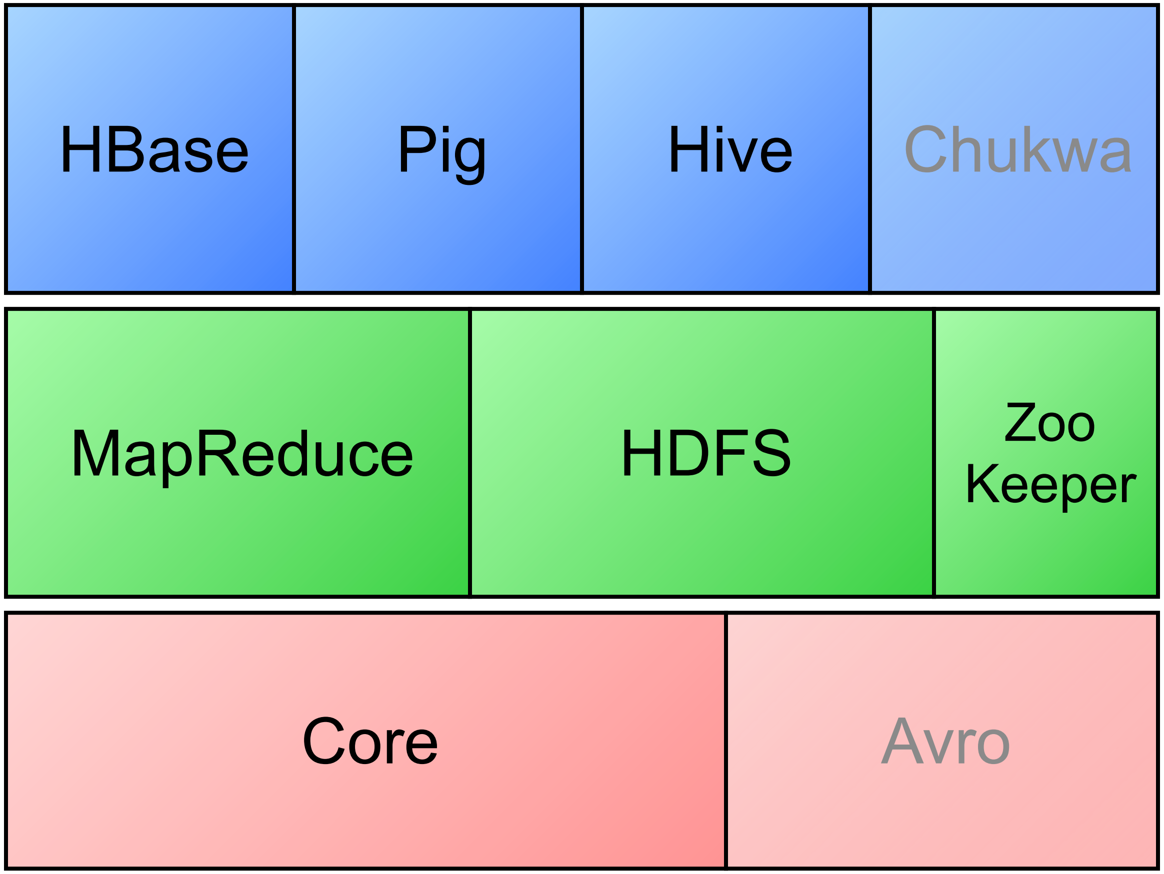
Hadoop



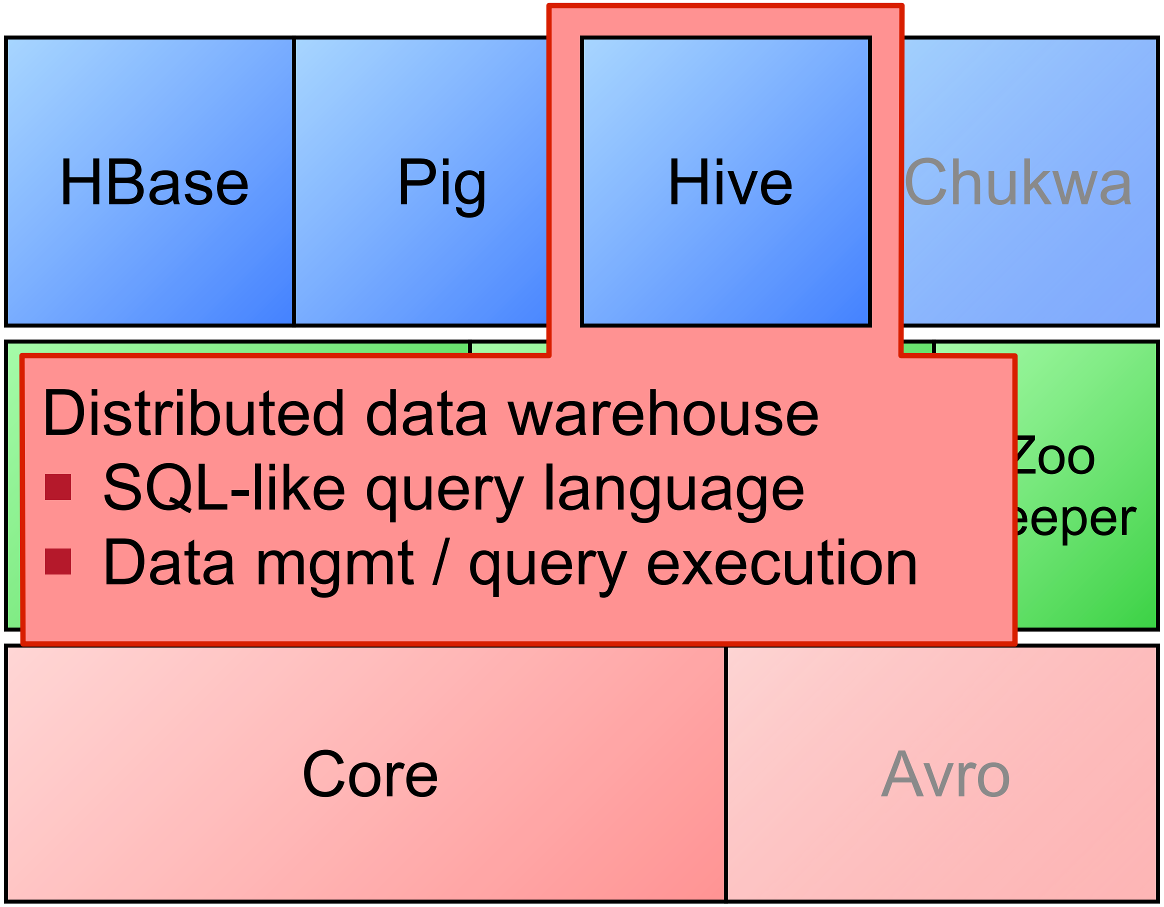
Hadoop


HBase
Pig
Hive
Chukwa
Zoo
eeper
Data flow language
Procedural SQL-inspired lang.
Execution environment
Core
Avro

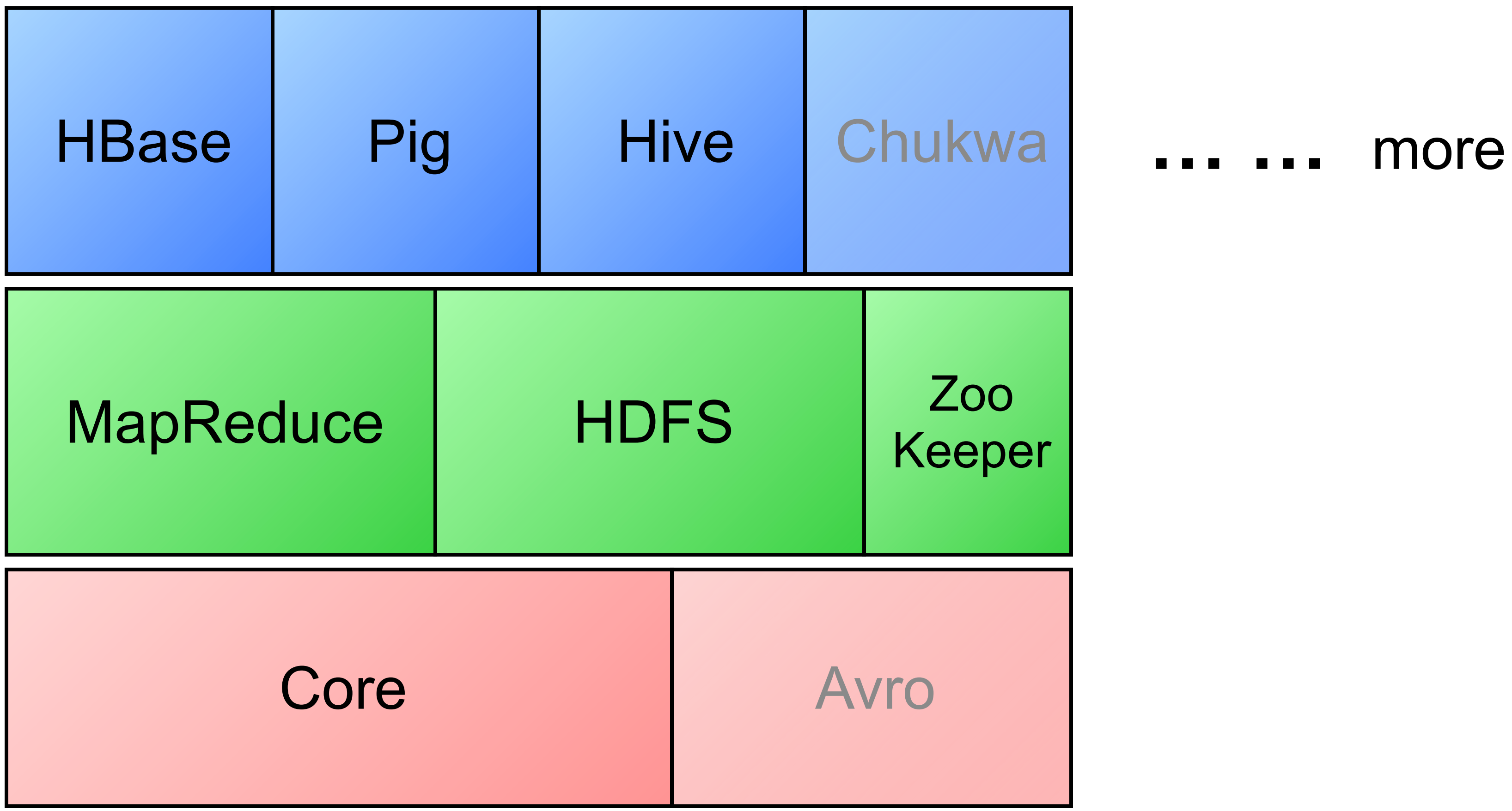
Hadoop



Hadoop


HBase
Pig
Hive
Chukwa
Zoo
eeper
Distributed data warehouse
SQL-like query language
Data mgmt / query execution
Core
Avro

Hadoop


HBase
Pig
Hive
Chukwa
… … more
MapReduce
HDFS
Zoo Keeper
Core
Avro

MapReduce

- Mapper: `(k1, v1) → (k2, v2)[]`
  - E.g., `(void, textline : string)`

    `→ (first : string, count : int)`
- Reducer: `(k2, v2[]) → (k3, v3)[]`
  - E.g., `(first : string, counts : int[])`

    `→ (first : string, total : int)`

- Combiner: `(k2, v2[]) → (k2, v2)[]`
- Partition: `(k2, v2) → int`

Mapper interface

```
interface Mapper<K1, V1, K2, V2> {
❶ void configure (JobConf conf);
❷ void map (K1 key, V1 value,
      OutputCollector<K2, V2> out,
      Reporter reporter);
❸ void close();
}
```

- Initialize in `configure()`
- Clean-up in `close()`
- Emit via `out.collect(key,val)` any time

Reducer interface

```
interface Reducer<K2, V2, K3, V3> {
❶ void configure (JobConf conf);
❷ void reduce (
      K2 key, Iterator<V2> values,
      OutputCollector<K3, V3> out,
❸     Reporter reporter);
  void close();
}
```

- Initialize in `configure()`
- Clean-up in `close()`
- Emit via `out.collect(key,val)` any time

Some canonical examples

- Histogram-type jobs:
  - Graph construction (bucket = edge)
  - K-means et al. (bucket = cluster center)
- Inverted index:
  - Text indices
  - Matrix transpose
- Sorting
- Equi-join
- **More details in part 2**

Equi-joins

“Reduce-side”

(Smith, 7)

(Jones, 7)

(Brown, 7)

(Davis, 3)

(Dukes, 5)

(Black, 3)

(Gruhl, 7)

MAP

(Sales, 3)

(Devel, 7)

(Acct., 5)

MAP

Equi-joins

“Reduce-side”

(Smith, 7)
(Jones, 7)
(Brown, 7)
(Davis, 3)
(Dukes, 5)
(Black, 3)
(Gruhl, 7)

MAP

7: (■, (Smith))

(Sales, 3)
(Devel, 7)
(Acct., 5)

MAP

Equi-joins

"Reduce-side"

```
(Smith, 7)    MAP  →  7: (■, (Smith))
(Jones, 7)
(Brown, 7)
(Davis, 3)
(Dukes, 5)
(Black, 3)
(Gruhl, 7)

(Sales, 3)
(Devel, 7)    MAP  →  7: (■, (Devel))
(Acct., 5)
```

Equi-joins

"Reduce-side"

`(Smith, 7)`
`(Jones, 7)`
`(Brown, 7)`
`(Davis, 3)`
`(Dukes, 5)`
`(Black, 3)`
`(Gruhl, 7)`

MAP

`7: (■, (Smith))`

-OR-

`(7,■): (Smith)`

`(Sales, 3)`
`(Devel, 7)`
`(Acct., 5)`

MAP

`7: (■, (Devel))`

-OR-

`(7,■): (Devel)`

Equi-joins

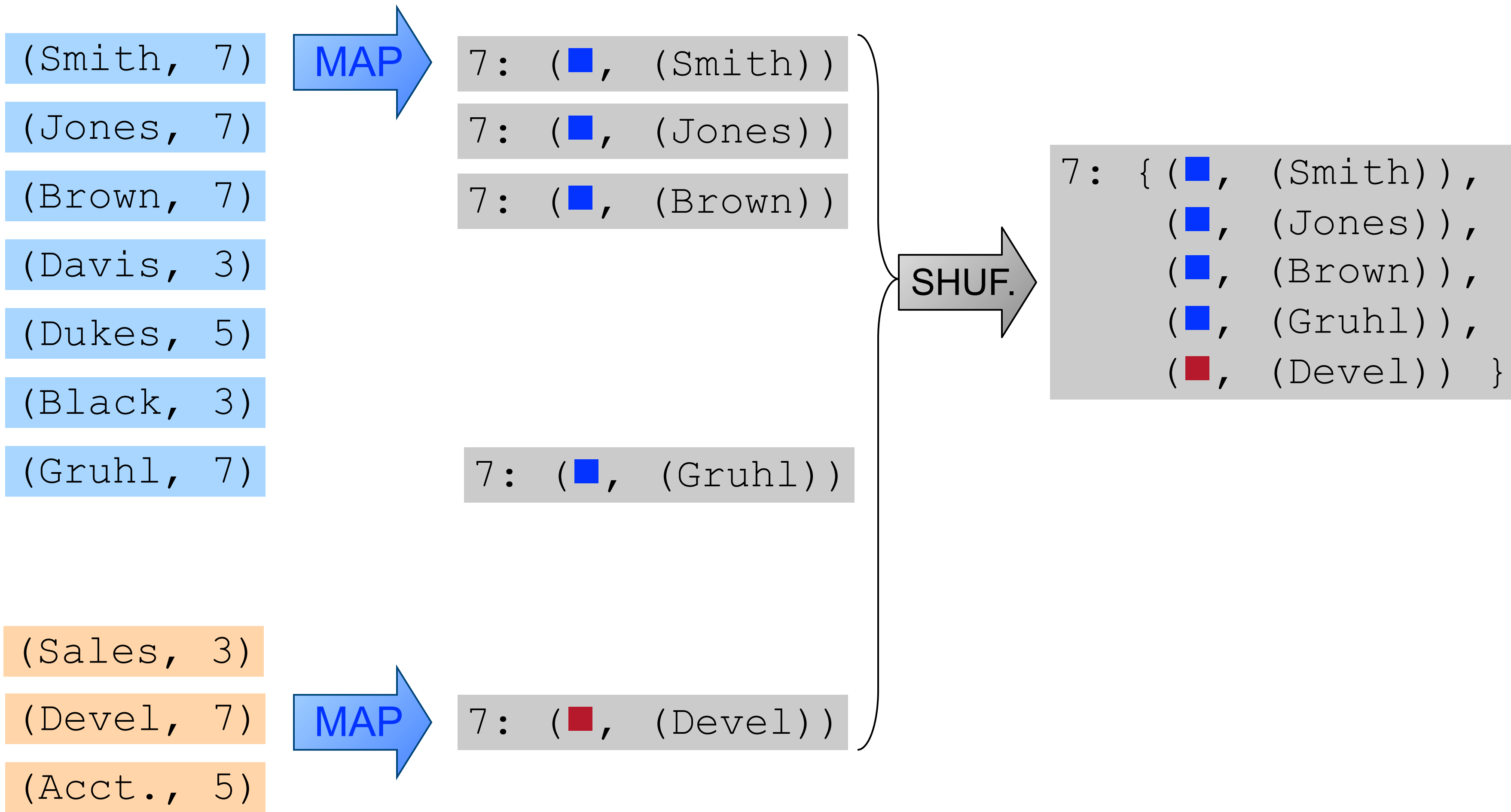
“Reduce-side”

```
(Smith, 7)                     7: ( , (Smith))
(Jones, 7)        MAP          7: ( , (Jones))
(Brown, 7)                     7: ( , (Brown))
(Davis, 3)
(Dukes, 5)
(Black, 3)
(Gruhl, 7)                     7: ( , (Gruhl))

(Sales, 3)
(Devel, 7)        MAP          7: ( , (Devel))
(Acct., 5)
```

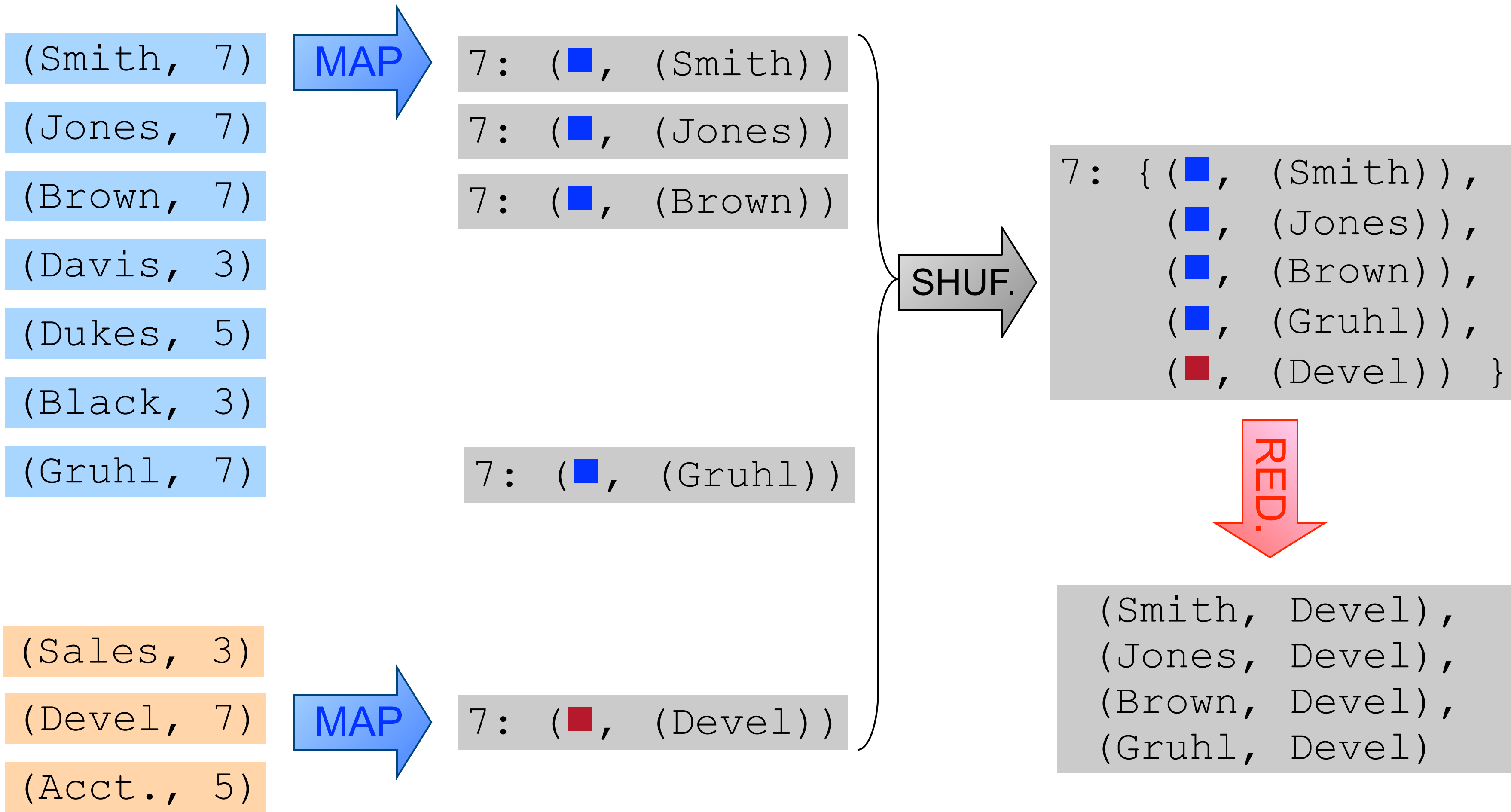
Equi-joins

“Reduce-side”

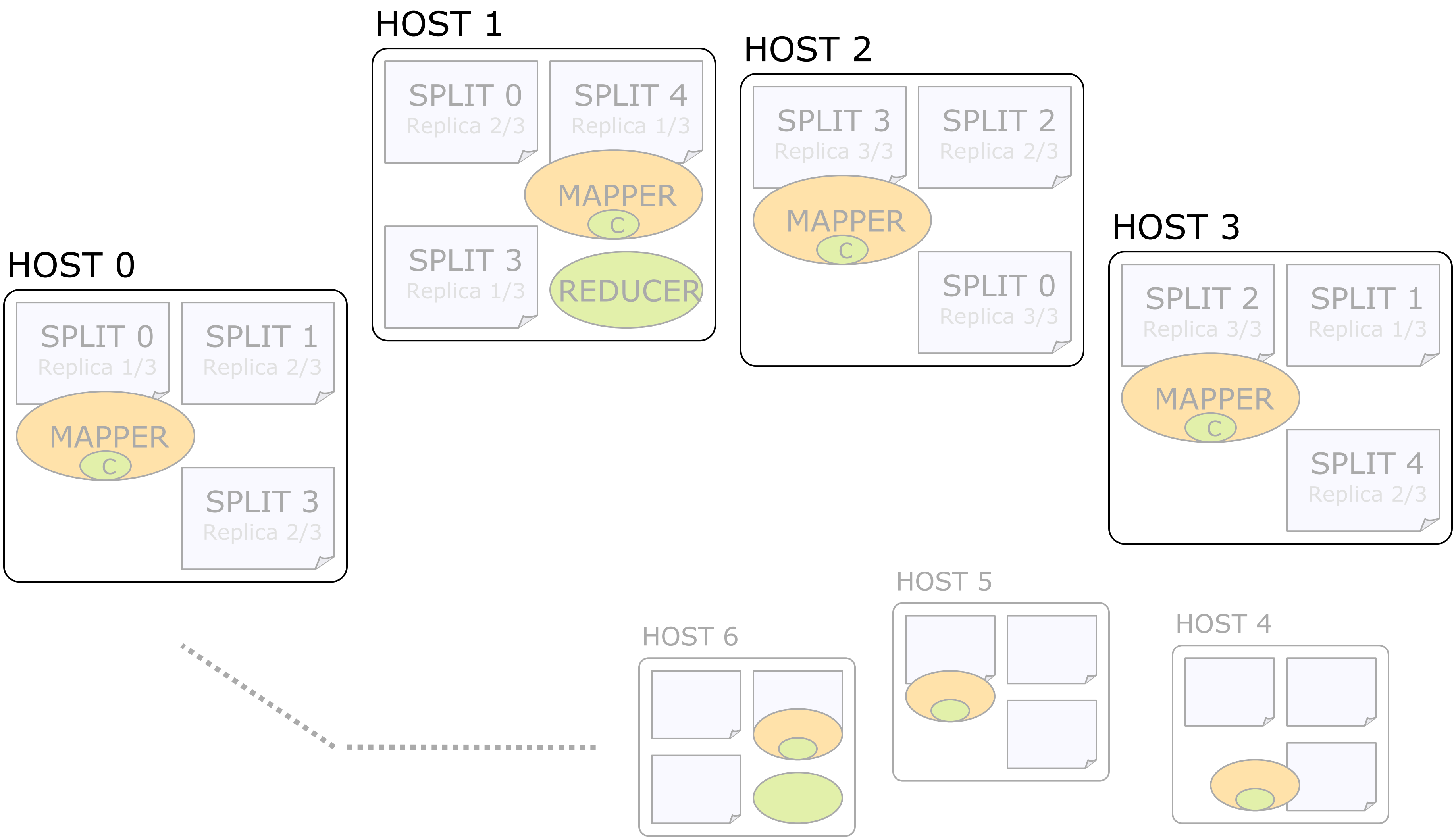


Equi-joins

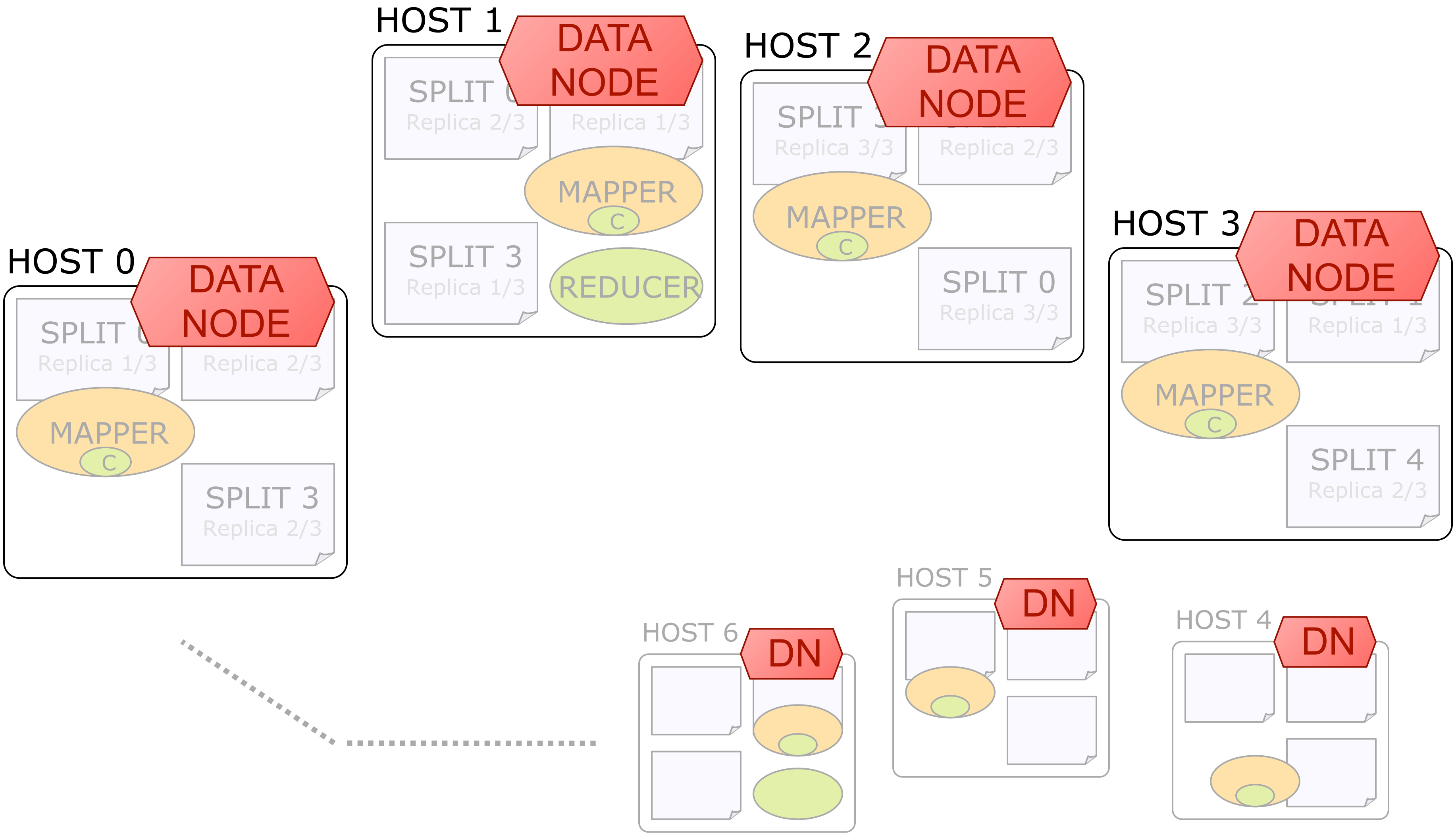
“Reduce-side”



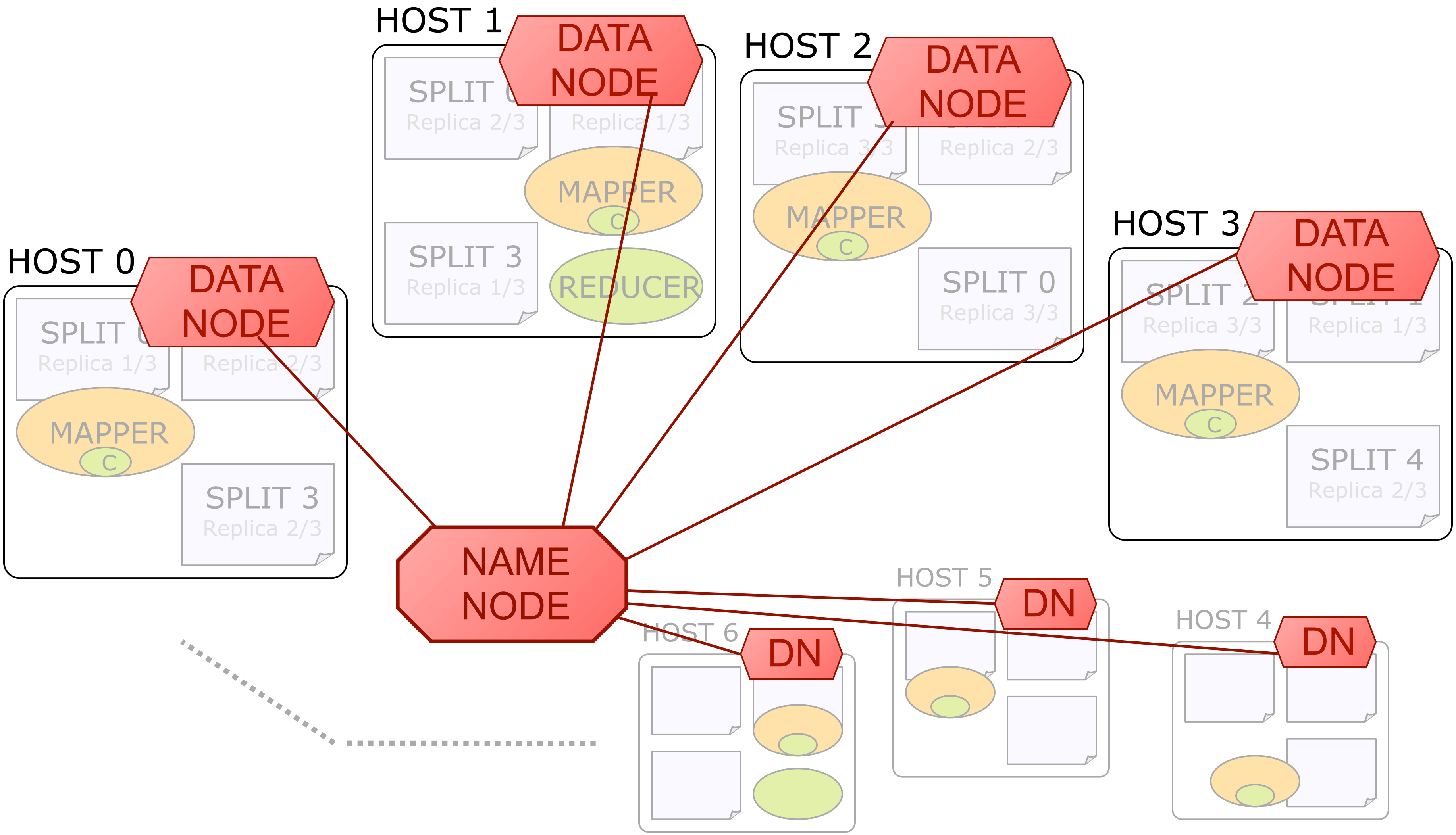
HDFS & MapReduce processes



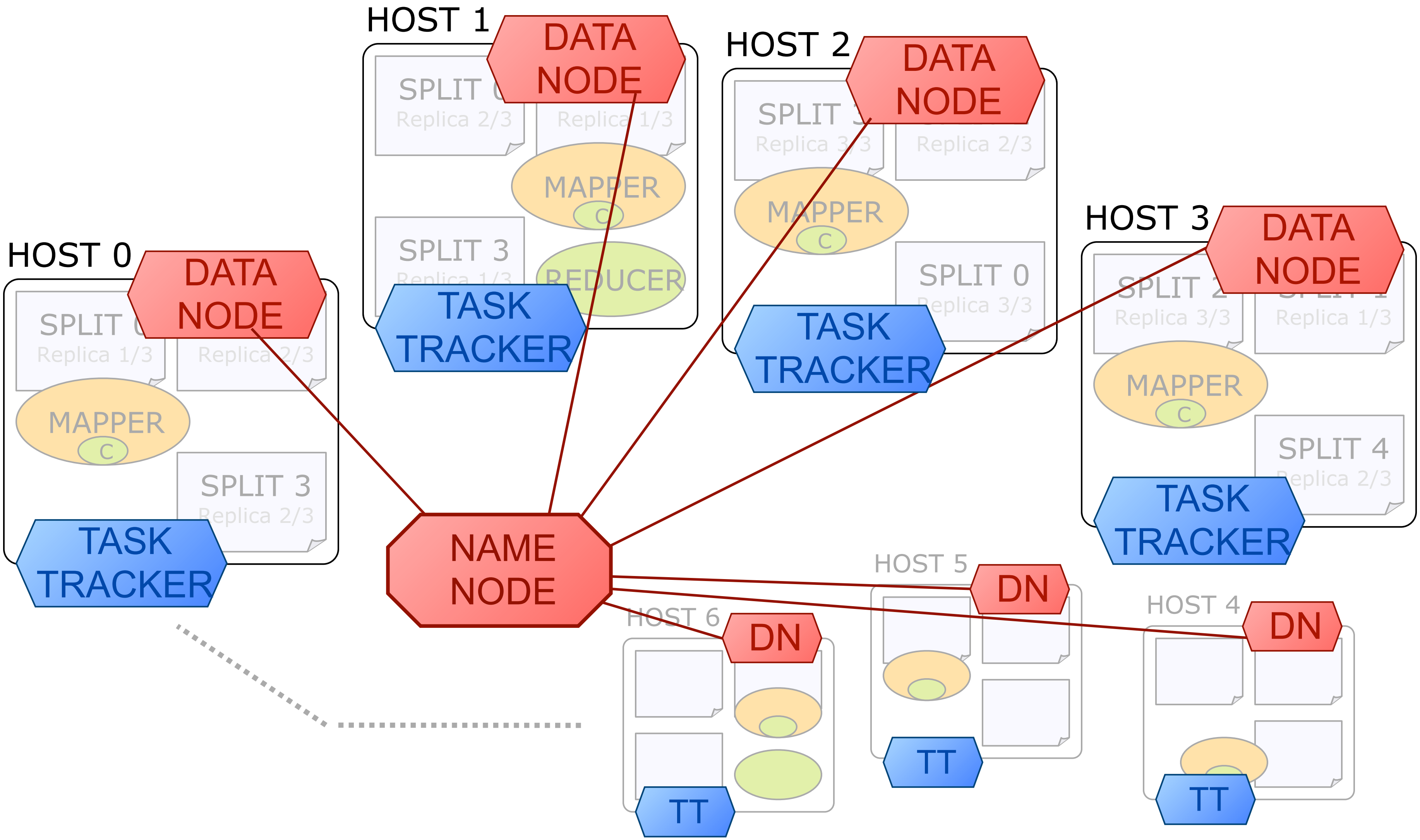
HDFS & MapReduce processes



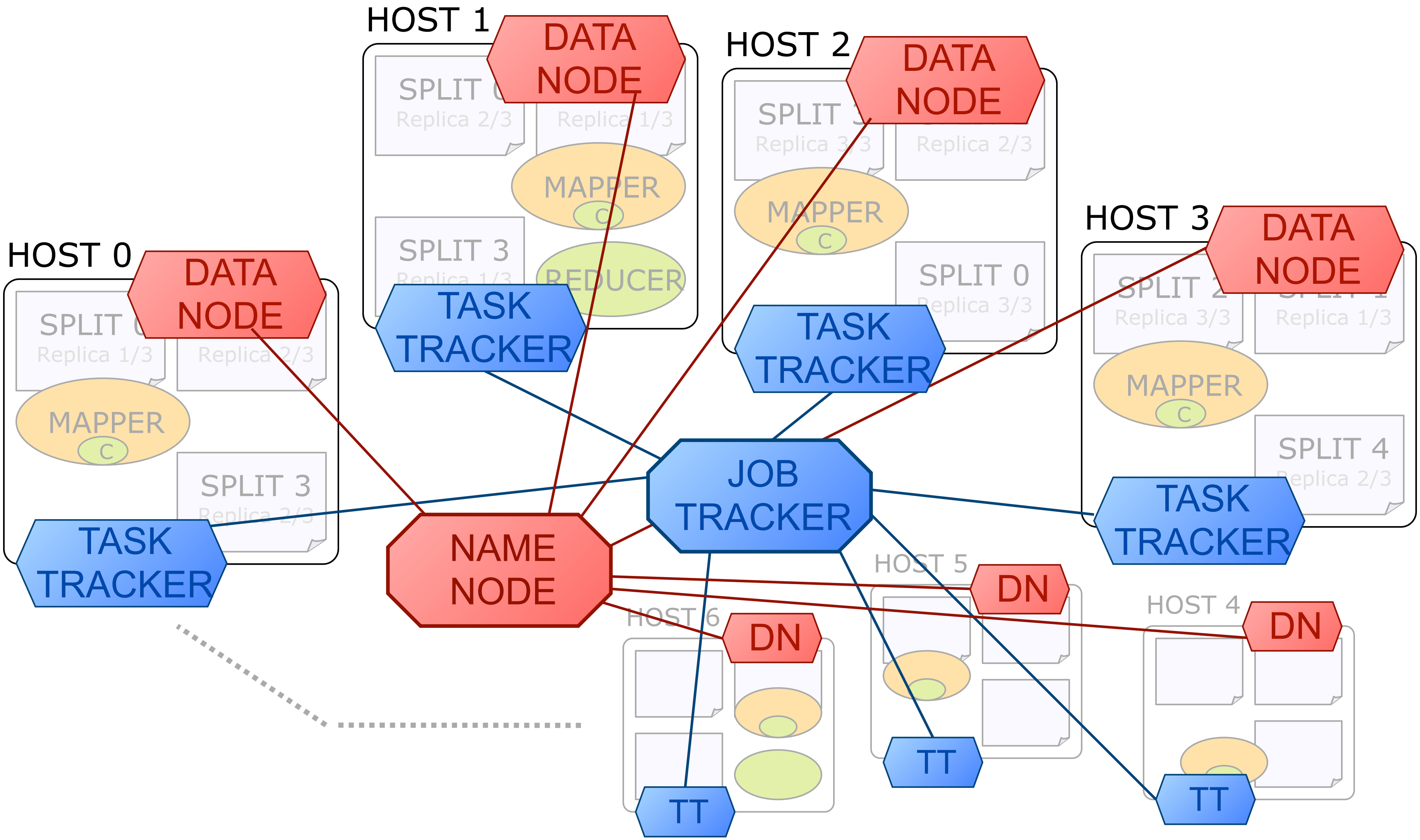
HDFS & MapReduce processes



HDFS & MapReduce processes



HDFS & MapReduce processes



Hadoop Streaming & Pipes

- Don't have to use Java for MapReduce

- Hadoop Streaming:
  - Use stdin/stdout & text format
  - Any language (C/C++, Perl, Python, shell, etc)

- Hadoop Pipes:
  - Use sockets & binary format (more efficient)
  - C++ library required

Outline

- Introduction
- MapReduce & distributed storage
- Hadoop
  - **HBase**
  - Pig
  - Cascading
  - Hive
- Summary

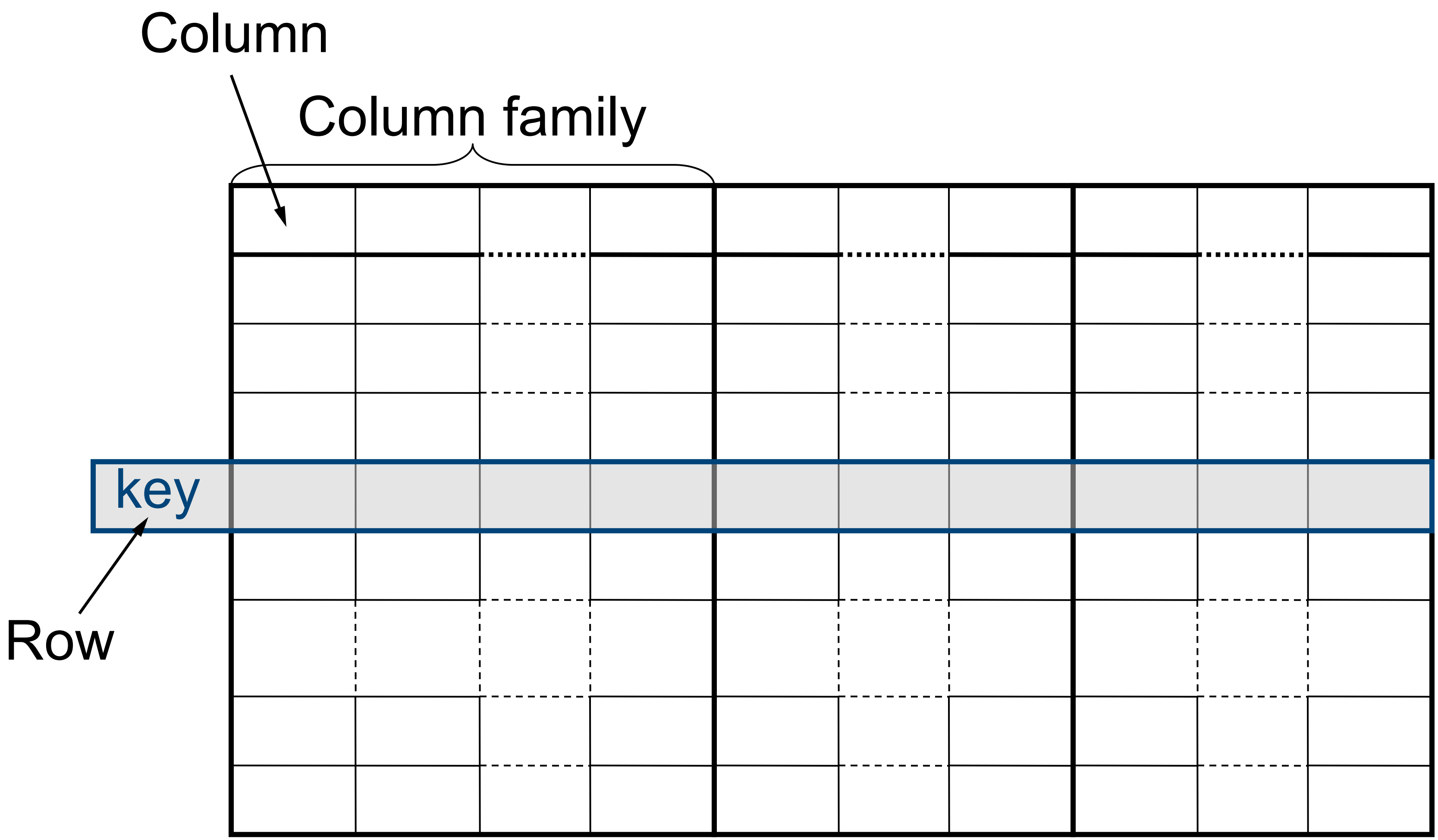
HBase introduction

- MapReduce canonical example:
  - Inverted index (more in Part 2)
- Batch computations on large datasets:
  - Build static index on crawl snapshot
- However, in reality crawled pages are:
  - Updated by crawler
  - Augmented by other parsers/analytics
  - Retrieved by cache search
  - Etc…

HBase introduction

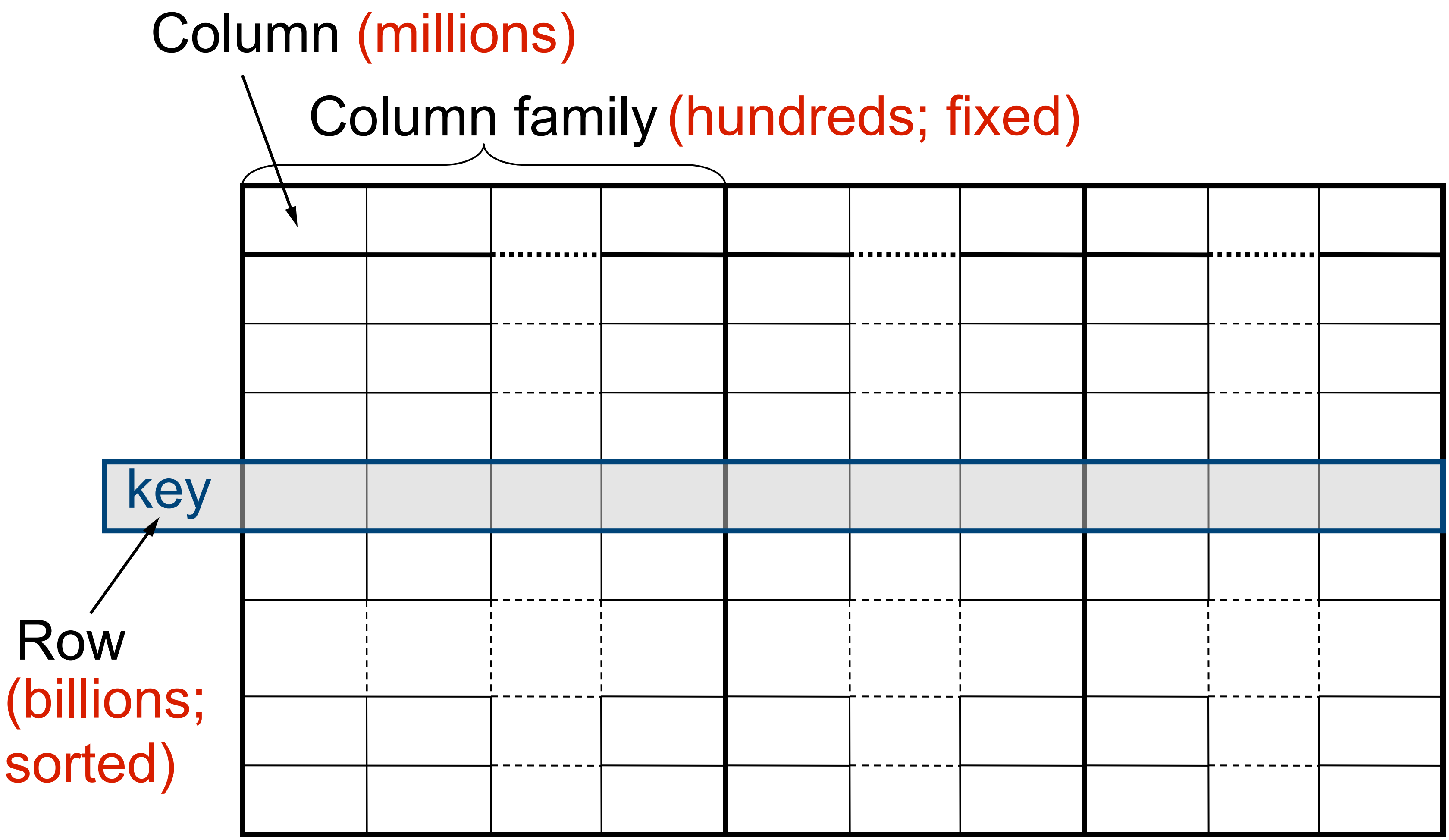
- MapReduce & HDFS:
  - Distributed storage + computation
  - Good for batch processing
  - But: no facilities for accessing or updating individual items
- HBase:
  - Adds random-access read / write operations
  - Originally developed at Powerset
  - Based on Google's Bigtable

HBase data model



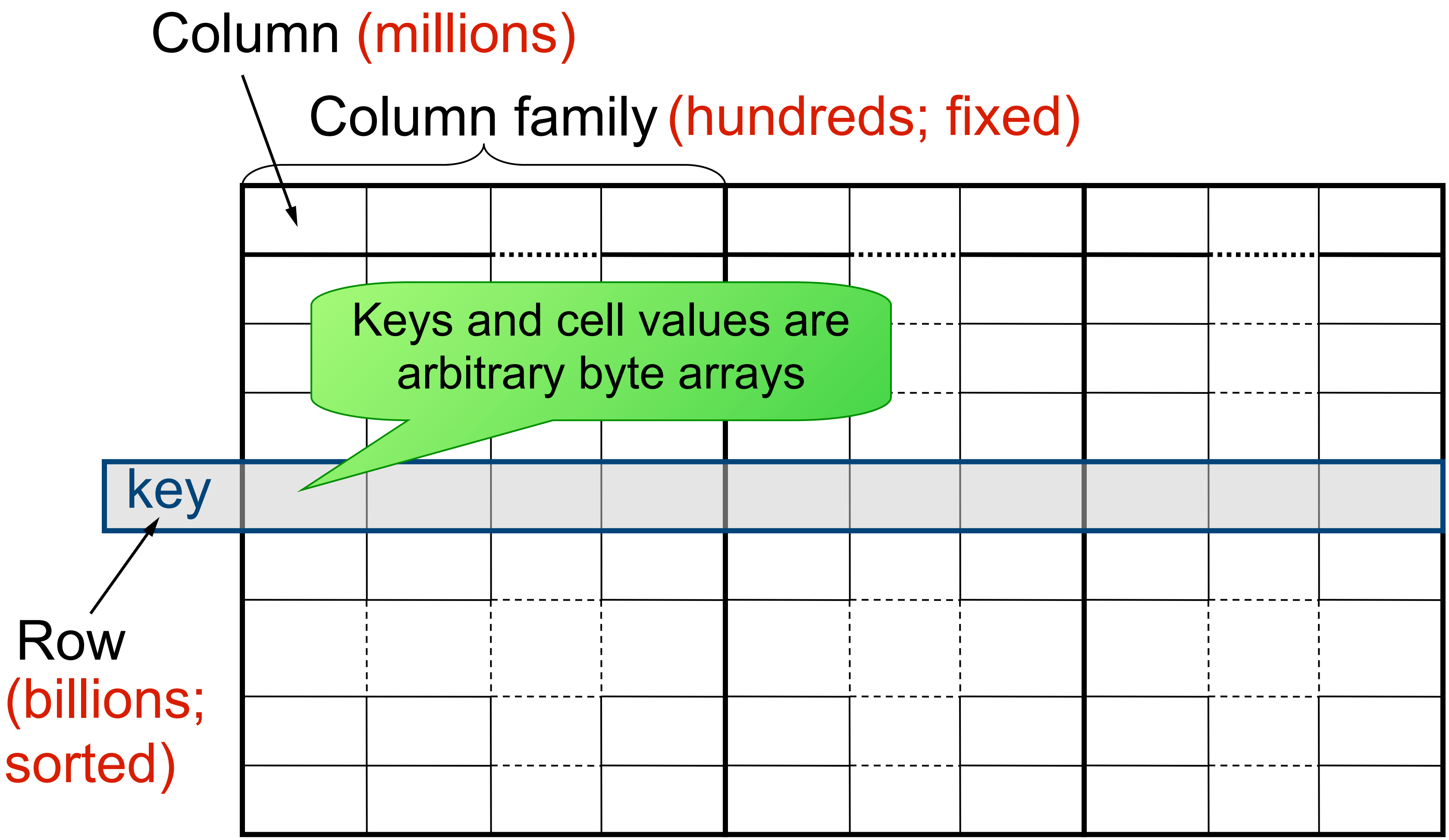
Partitioned over many nodes

HBase data model



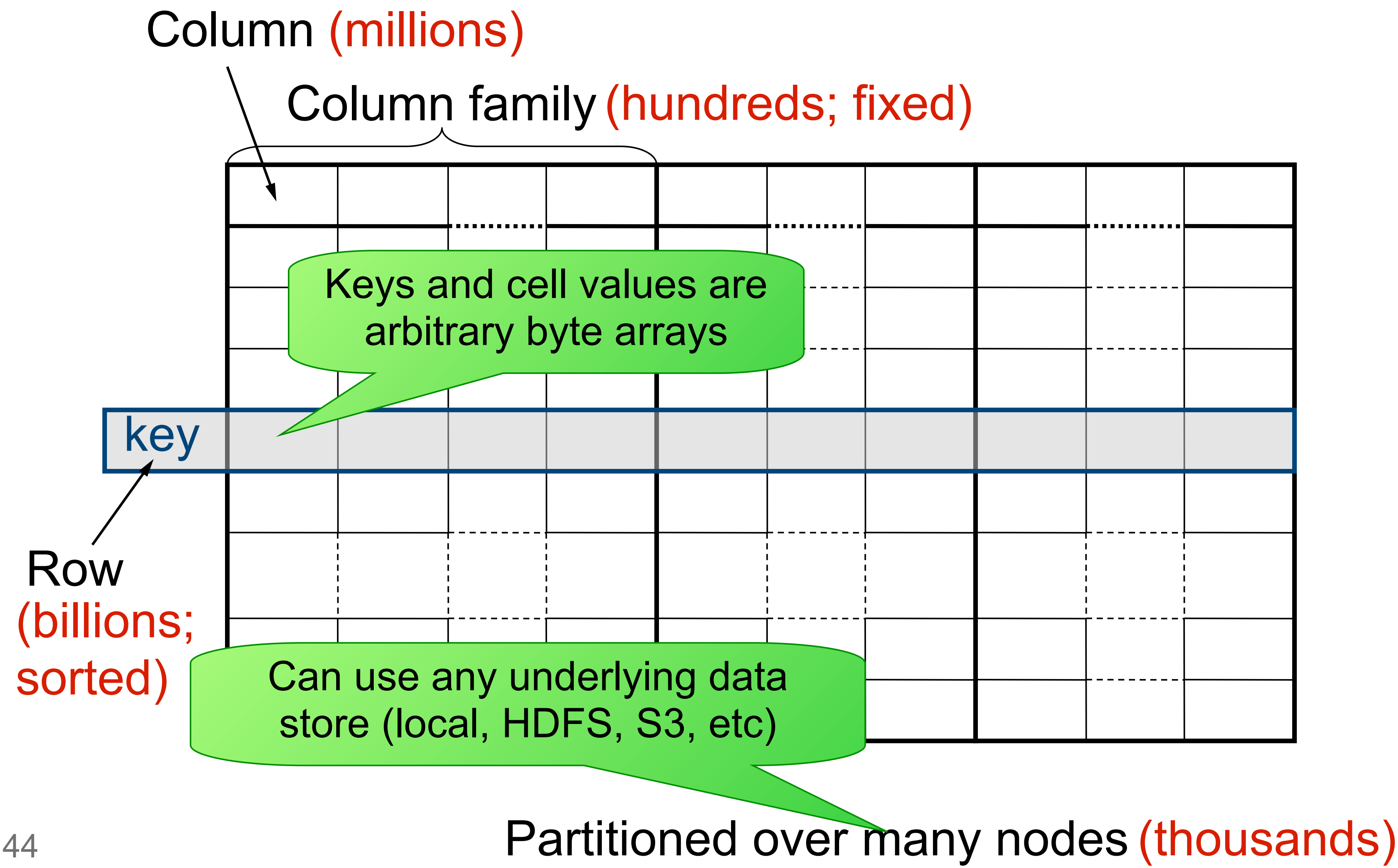
Partitioned over many nodes (thousands)

HBase data model



Partitioned over many nodes (thousands)

HBase data model



Data model example

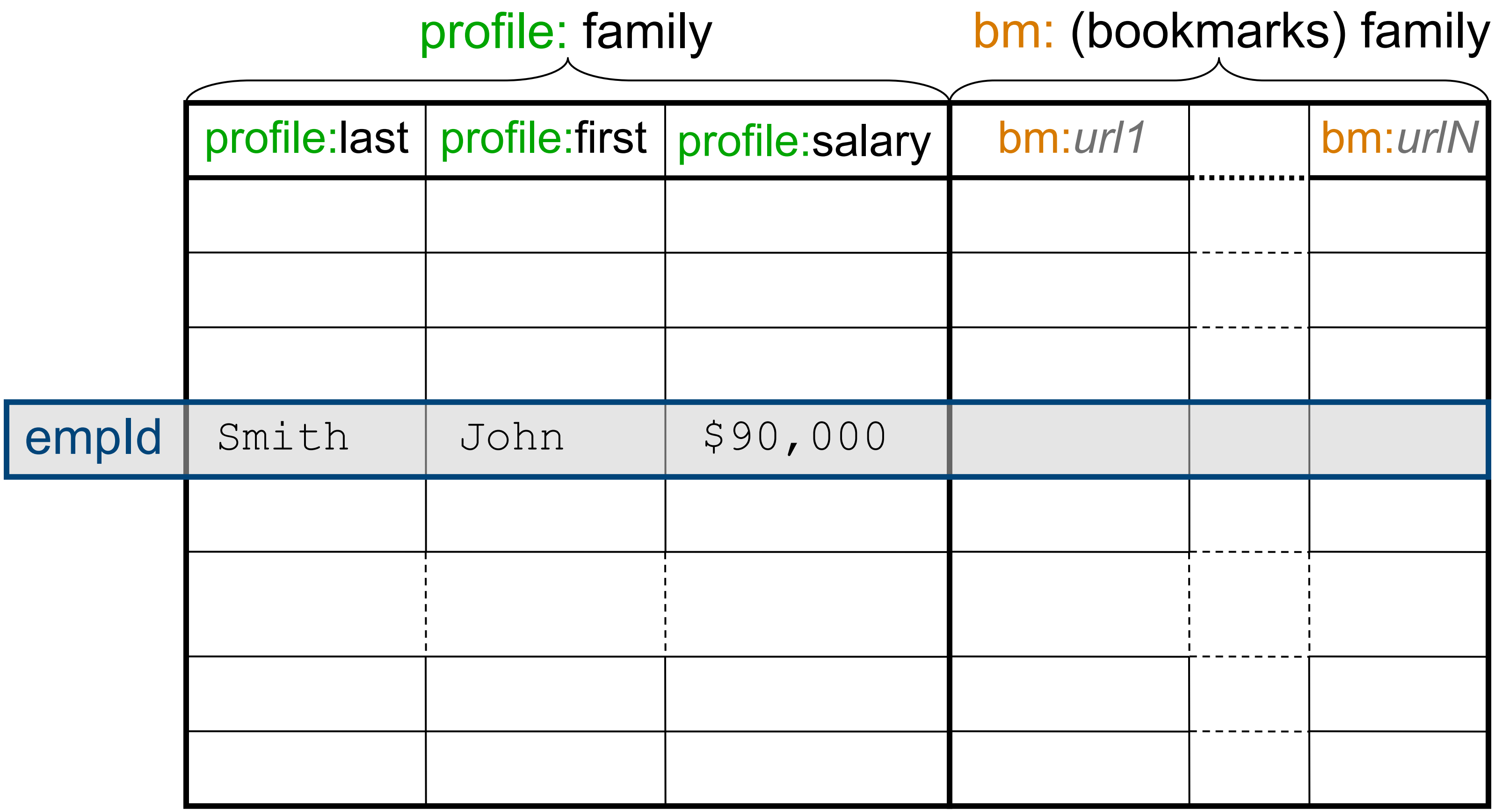
profile: family

	profile:last	profile:first	profile:salary
empId	Smith	John	$90,000

Data model example

	profile: family			bm: (bookmarks) family		
	profile:last	profile:first	profile:salary	bm:*url1*		bm:*urlN*
empId	Smith	John	$90,000			

Data model example



Always access via primary key

HBase vs. RDBMS

- Different solution, similar problems
- RDBMSes:
  - Row-oriented
  - Fixed-schema
  - ACID
- HBase et al.:
  - Designed from ground-up to scale out, by adding commodity machines
  - Simple consistency scheme: atomic row writes
  - Fault tolerance
  - Batch processing
  - No (real) indexes

Outline

- Introduction
- MapReduce & distributed storage
- Hadoop
  - HBase
  - **Pig**
  - Cascading
  - Hive
- Summary

Pig introduction

- “ ~5 lines of non-boilerplate code ”
- Writing a single MapReduce job requires significant gruntwork
  - Boilerplates (mapper/reducer, create job, etc)
  - Input / output formats
- Many tasks require more than one MapReduce job

Pig main features

- Data structures (multi-valued, nested)
- Pig-latin: data flow language
  - SQL-inspired, but imperative (not declarative)

Pig example

```
records = LOAD filename
   AS (last: chararray, first: chararray, salary:int);
grouped = GROUP records BY first;
counts = FOREACH grouped
   GENERATE group, COUNT(records.first);
DUMP counts;
```

employees.txt

# LAST	FIRST	SALARY
Smith	John	$90,000
Brown	David	$70,000
Johnson	George	$95,000
Yates	John	$80,000
Miller	Bill	$65,000
Moore	Jack	$85,000
Taylor	Fred	$75,000
Smith	David	$80,000
Harris	John	$90,000
...	...	...
...	...	...

Q: "What is the frequency of each first name?"

Pig schemas

- Schema = tuple data type

- Schemas are optional!
  - Data-loading step is not required
  - "Unknown" schema: similar to AWK ($0, $1, ..)

- Support for most common datatypes
- Support for nesting

Pig Latin feature summary

- Data loading / storing
  - LOAD / STORE / DUMP
- Filtering
  - FILTER / DISTINCT / FOREACH / STREAM
- Group-by
  - GROUP
- Join & co-group
  - JOIN / COGROUP / CROSS
- Sorting
  - ORDER / LIMIT
- Combining / splitting
  - UNION / SPLIT

Outline

- Introduction
- MapReduce & distributed storage
- Hadoop
  - HBase
  - Pig
  - **Cascading**
  - Hive
- Summary

Cascading introduction

- Provides higher-level abstraction
  - Fields, Tuples
  - Pipes
  - Operations
  - Taps, Schemes, Flows
- Ease composition of multi-job flows

Cascading introduction

- Provides higher-level abstraction
  - Fields, Tuples
  - Pipes
  - Operations
  - Taps, Schemes, Flows
- Ease composition of multi-job flows

Library, not a new language

Cascading example

employees.txt

# LAST	FIRST	SALARY
Smith	John	$90,000
Brown	David	$70,000
...	...	...

Q: "What is the frequency of each first name?"

```
Scheme srcScheme = new TextLine();
Tap source = new Hfs(srcScheme, filename);
Scheme dstScheme = new TextLine();
Tap sink = new Hfs(dstScheme, filename, REPLACE);

Pipe assembly = new Pipe("lastnames");

Function splitter = new RegexSplitter(
      new Fields("last", "first", "salary"), "\t");
assembly = new Each(assembly, new Fields("line"), splitter);

assembly = new GroupBy(assembly, new Fields("first"));

Aggregator count = new Count(new Fields("count"));
assembly = new Every(assembly, count);

FlowConnector flowConnector = new FlowConnector();
Flow flow = flowConnector.connect("last-names",
                    source, sink, assembly);
flow.complete();
```

Cascading example

employees.txt

# LAST	FIRST	SALARY
Smith	John	$90,000
Brown	David	$70,000
...	...	...

Q: “What is the frequency of each first name?”

```
Scheme srcScheme = new TextLine();
Tap source = new Hfs(srcScheme, filename);
Scheme dstScheme = new TextLine();
Tap sink = new Hfs(dstScheme, filename, REPLACE);

Pipe assembly = new Pipe("lastnames");

Function splitter = new RegexSplitter(
      new Fields("last", "first", "salary"), "\t");
assembly = new Each(assembly, new Fields("line"), splitter);

assembly = new GroupBy(assembly, new Fields("first"));

Aggregator count = new Count(new Fields("count"));
assembly = new Every(assembly, count);

FlowConnector flowConnector = new FlowConnector();
Flow flow = flowConnector.connect("last-names",
                    source, sink, assembly);
flow.complete();
```

Cascading feature summary

- Pipes: transform streams of tuples
  - Each
  - GroupBy / CoGroup
  - Every
  - SubAssembly
- Operations: what is done to tuples
  - Function
  - Filter
  - Aggregator / Buffer

Outline

- Introduction
- MapReduce & distributed storage
- Hadoop
  - HBase
  - Pig
  - Cascading
  - **Hive**
- Summary

Hive introduction

- Originally developed at Facebook
  - Now a Hadoop sub-project
- Data warehouse infrastructure
  - Execution: MapReduce
  - Storage: HDFS files
- ***Large*** datasets, e.g. Facebook daily logs
  - 30GB (Jan'08), 200GB (Mar'08), 15+TB (2009)
- Hive QL: SQL-like query language

Hive example

```
CREATE EXTERNAL TABLE records
   (last STRING, first STRING, salary INT)
ROW FORMAT DELIMETED
   FIELDS TERMINATED BY '\t'
STORED AS TEXTFILE
LOCATION filename;

SELECT records.first, COUNT(1)
FROM records
GROUP BY records.first;
```

employees.txt

# LAST	FIRST	SALARY
Smith	John	$90,000
Brown	David	$70,000
Johnson	George	$95,000
Yates	John	$80,000
Miller	Bill	$65,000
Moore	Jack	$85,000
Taylor	Fred	$75,000
Smith	David	$80,000
Harris	John	$90,000
...	...	...
...	...	...

Q: "What is the frequency of each first name?"

Hive schemas

- Data should belong to tables
  - But can also use pre-existing data
  - Data loading optional (like Pig) but encouraged
- Partitioning columns:
  - Mapped to HDFS directories
  - E.g., (date, time) → `datadir/2009-03-12/18_30_00`
- Data columns (the rest):
  - Stored in HDFS files
- Support for most common data types
- Support for pluggable serialization

Hive QL feature summary

- Basic SQL
  - FROM subqueries
  - JOIN (only equi-joins)
  - Multi GROUP BY
  - Multi-table insert
  - Sampling
- Extensibility
  - Pluggable MapReduce scripts
  - User Defined Functions
  - User Defined Types
  - SerDe (serializer / deserializer)

Outline

- Introduction
- MapReduce & distributed storage
- Hadoop
  - HBase
  - Pig
  - Cascading
  - Hive
- **Summary**

Recap

Recap

- Scalable: all

Recap

- Scalable: all
- High(-er) level: all except MR

Recap

- Scalable: all
- High(-er) level: all except MR
- Existing language: MR, Cascading

Recap

- Scalable: all
- High(-er) level: all except MR
- Existing language: MR, Cascading
- “Schemas”: HBase, Pig, Hive, (Casc.)
  - Pluggable data types: all

Recap

- Scalable: all
- High(-er) level: all except MR
- Existing language: MR, Cascading
- "Schemas": HBase, Pig, Hive, (Casc.)
  - Pluggable data types: all
- Easy transition: Hive, (Pig)

Related projects

Higher level—computation:

- Dryad & DryadLINQ (Microsoft) [EuroSys 2007]
- Sawzall (Google) [Sci Prog Journal 2005]

Higher level—storage:

- Bigtable [OSDI 2006] / Hypertable

Lower level:

- Kosmos Filesystem (Kosmix)
- VSN (Parascale)
- EC2 / S3 (Amazon)
- Ceph / Lustre / PanFS

- Sector / Sphere (http://sector.sf.net/)
- …

Summary

MapReduce:

- Simplified parallel programming model

Hadoop:

- Built from ground-up for:
  - Scalability
  - Fault-tolerance
  - Clusters of commodity hardware
- Growing collection of components and extensions (HBase, Pig, Hive, etc)

Tutorial overview

- Part 1 (Spiros): Basic concepts & tools
  - MapReduce & distributed storage
  - Hadoop / HBase / Pig / Cascading / Hive

NEXT:

- Part 2 (Jimeng): Algorithms
  - Information retrieval
  - Graph algorithms
  - Clustering (k-means)
  - Classification (k-NN, naïve Bayes)
- Part 3 (Rong): Applications
  - Text processing
  - Data warehousing
  - Machine learning

Large-scale Data Mining: MapReduce and beyond
Part 1: Basics

Spiros Papadimitriou, Google
Jimeng Sun, IBM Research
Rong Yan, Facebook