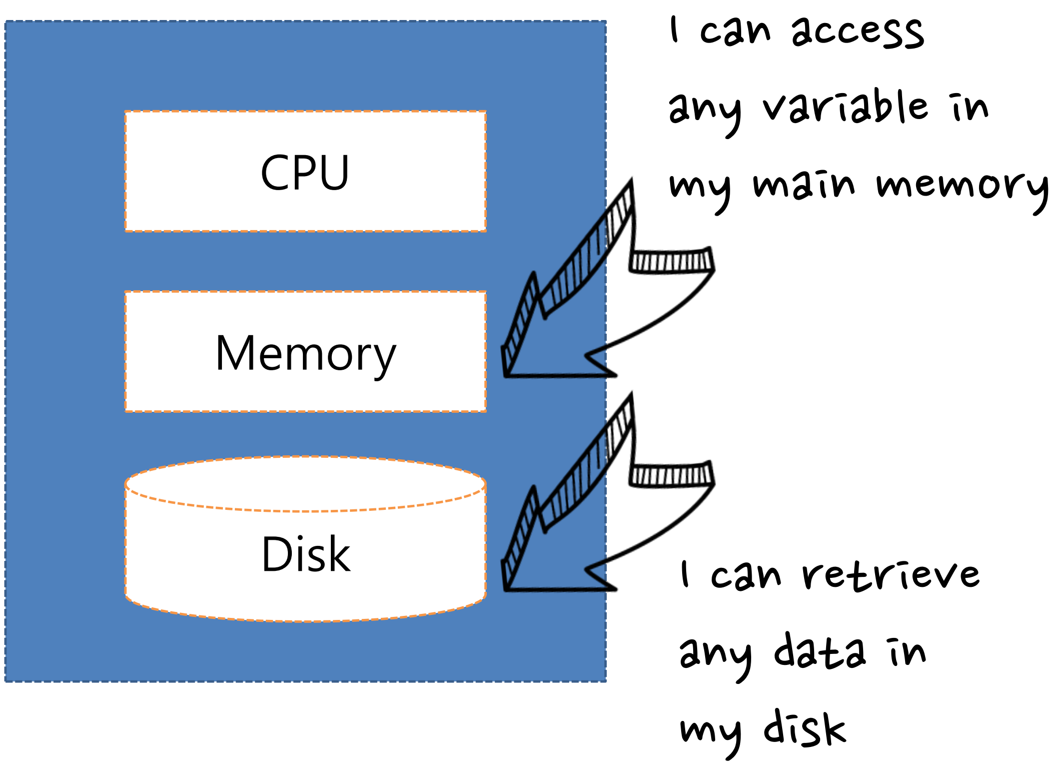
Parallel Indexing & MapReduce

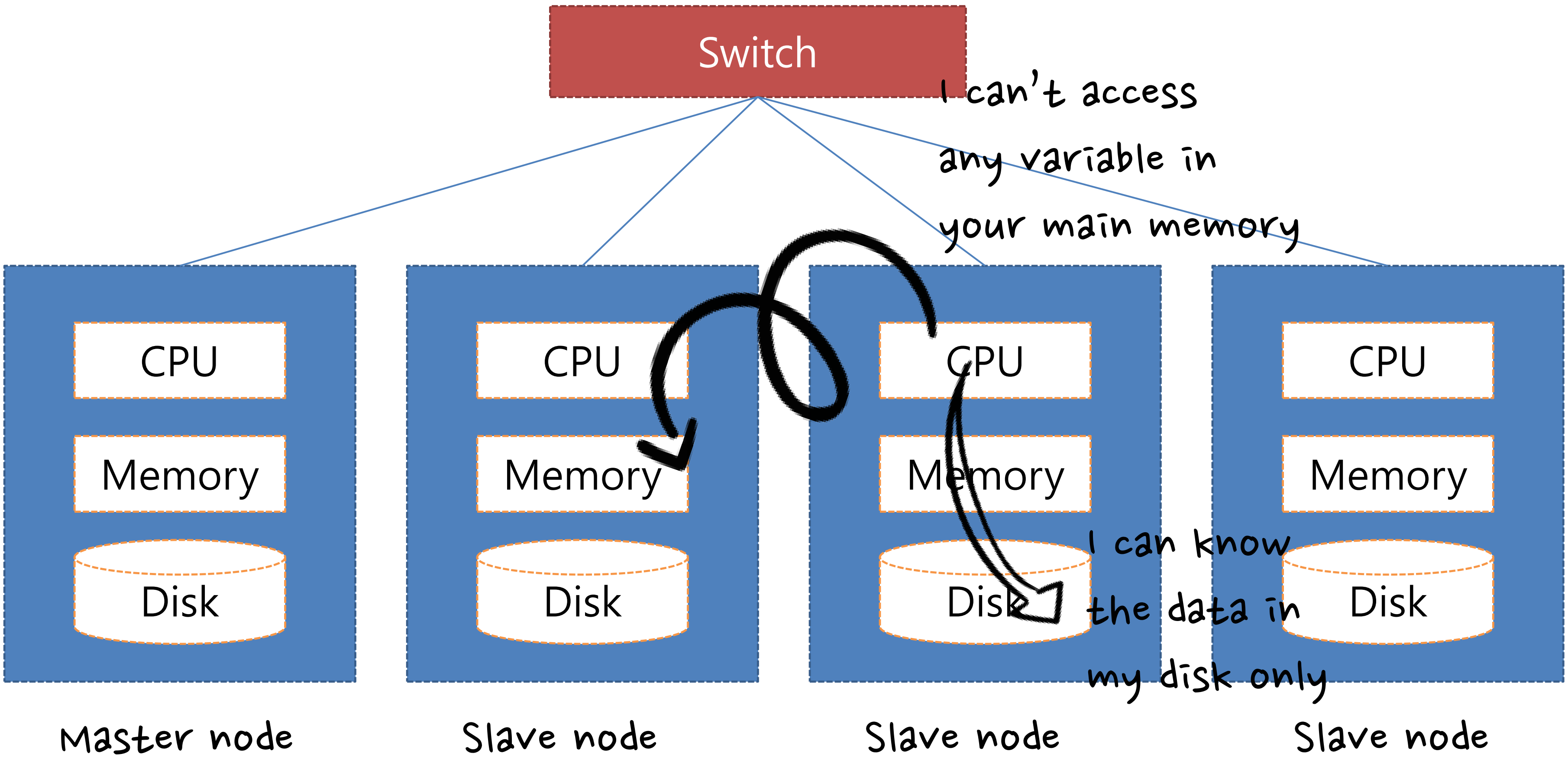
Younghoon Kim
Hanyang University ERICA
(nongaussian@hanyang.ac.kr)

PARALLEL PROGRAMMING USING MAPREDUCE

Single Node Architecture



Shared-Nothing Cluster Architecture



Programming Model

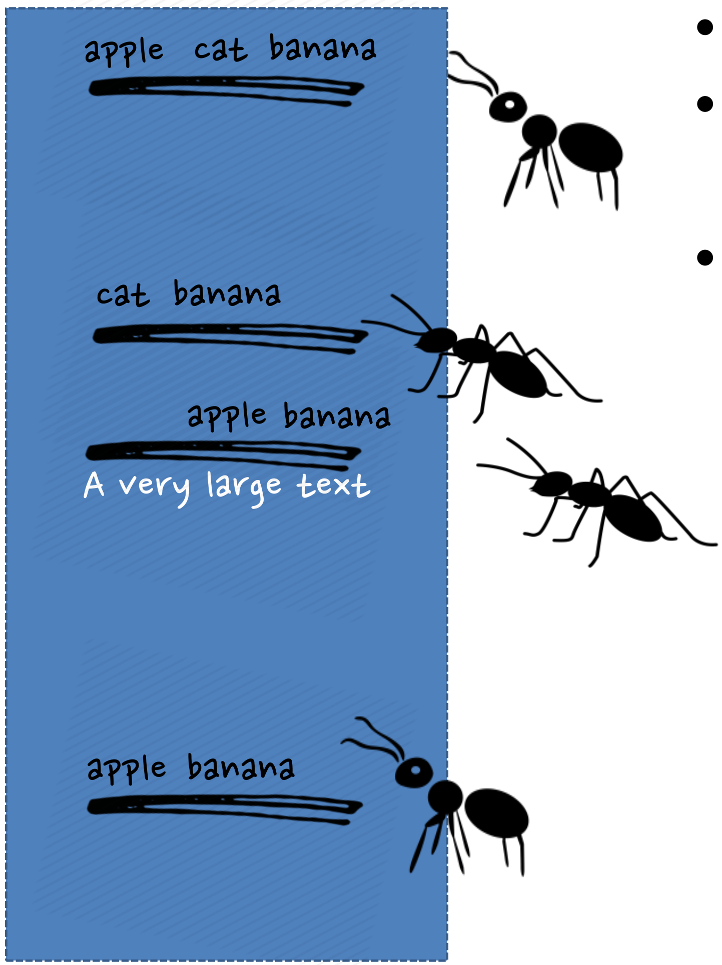
- Functional programming
- Users implement interface of two functions:
  - map (in_key, in_value) -> (out_key, intermediate_value) list
  - reduce (out_key,intermediate_value list) -> out_value list

MAP/REDUCE EXAMPLE #1 (WORD COUNTING)

Word Counting

```
main () {
        fd = open file ('big text file');
        cnt = initialize a hash table;
        while ( (line = read_a_line (fd)) != null) {
                tokens = tokenize (line);
                foreach (word in tokens) {
                        if (cnt[word] is defined) {
                                cnt[word] += 1;
                        }
                        else {
                                cnt[word] = 1;
                        }
                }
        }
}
```

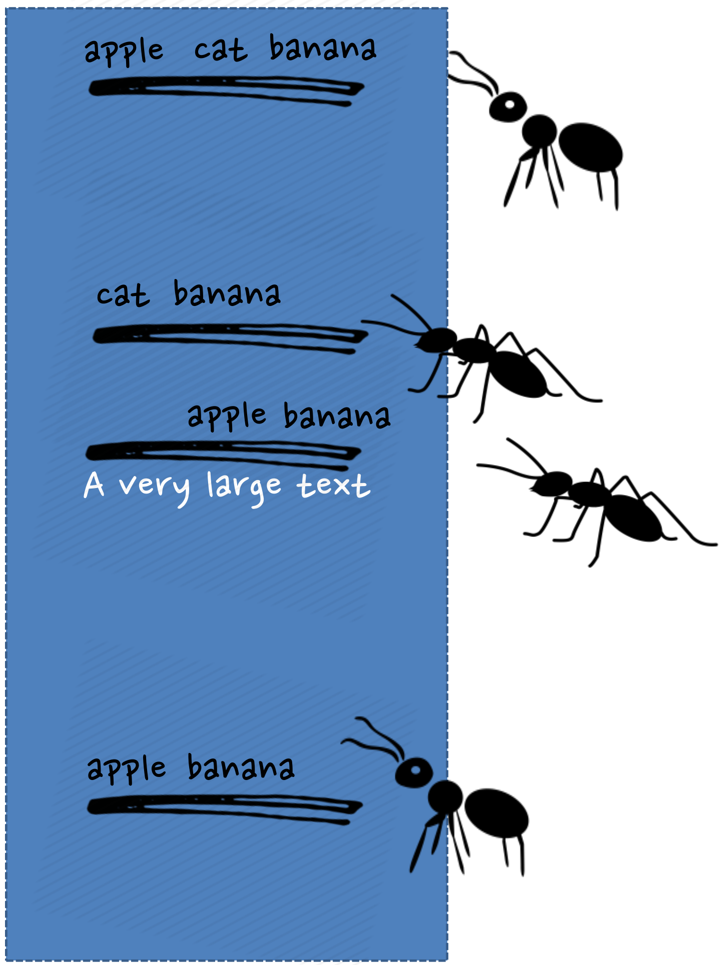
Word Counting with MapReduce



- I can read only a line
- we cannot use any hash table
- what I can do is

```
tokens ← tokenize (line);
foreach (word in tokens)
{
    emit(word, 1);
}
```

Word Counting with MapReduce



<apple, 1>

<apple, 1>

<apple, 1>

<banana, 1>

<banana, 1>

<banana, 1>

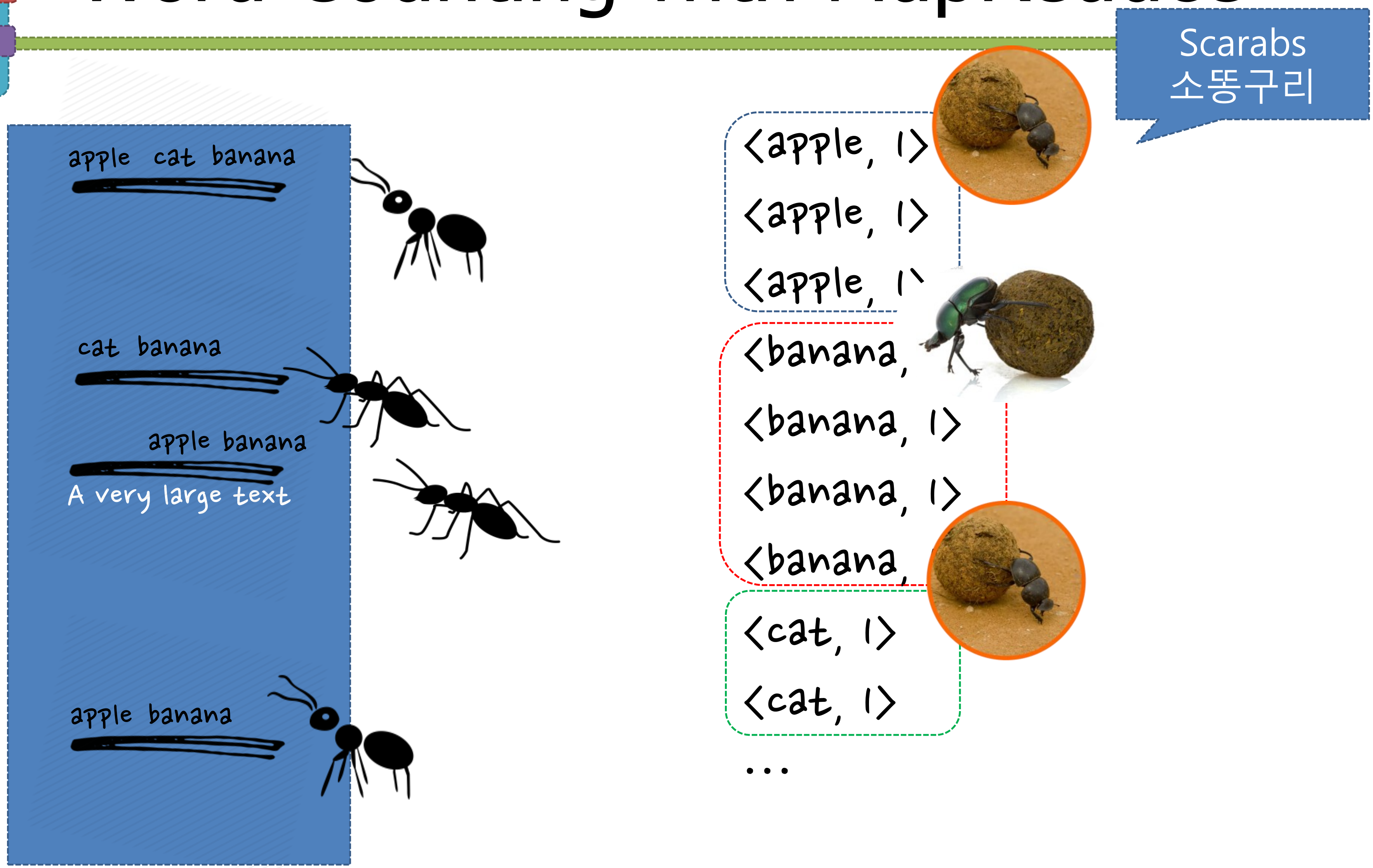
<banana, 1>

<cat, 1>

<cat, 1>

...

Word Counting with MapReduce



Word Counting with MapReduce

Documents(t)
Financial, IMF, Economics, Crisis
Financial, IMF, Crisis

Documents(t)
Economics, Harry
Financial, Harry, Potter, Film
Crisis, Harry, Potter

Map

Key	Value
Financial	1
IMF	1
Economics	1
Crisis	1
Financial	1
IMF	1
Crisis	1

Word Counting with MapReduce

Documents(t)
Financial, IMF, Economics, Crisis
Financial, IMF, Crisis

Documents(t)
Economics, Harry
Financial, Harry, Potter, Film
Crisis, Harry, Potter

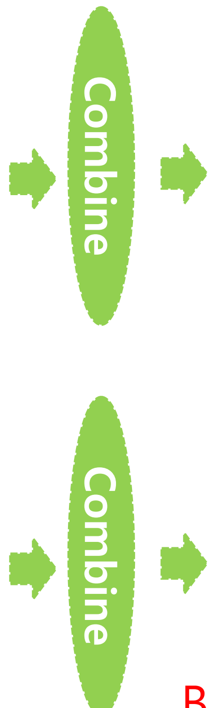
Map

Key	Value
Economics	1
Harry	1
Financial	1
Harry	1
Potter	1
Film	1
Crisis	1
Harry	1
Potter	1

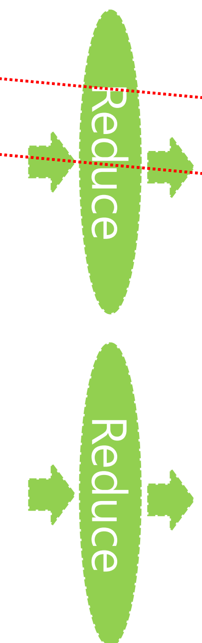
Word Counting with MapReduce

Documents(t)
Financial, IMF, Economics, Crisis
Financial, IMF, Crisis

Documents(t)
Economics, Harry
Financial, Harry, Potter, Film
Crisis, Harry, Potter



Key	Value list
Financial	1,1, 1
IMF	1,1
Economics	1, 1
Crisis	1,1, 1
Harry	1,1,1
Film	1
Potter	1,1



Key	Value

Before reduce functions are called,
for each distinct key,
the list of its values are generated

Hadoop MapReduce Programming in Java

```
public static class Map extends MapReduceBase implements
              Mapper<LongWritable, Text, Text, IntWritable> {
  private final static IntWritable one = new IntWritable(1);
  private Text word = new Text();

  public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable>
                  output, Reporter reporter) throws IOException {
    String line = value.toString();
    StringTokenizer tokenizer = new StringTokenizer(line);
    while (tokenizer.hasMoreTokens()) {
      word.set(tokenizer.nextToken());
      output.collect(word, one);
}}}
```

Hadoop MapReduce Programming in Java

```
public static class Reduce 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 sum = 0;
    while (values.hasNext()) { sum += values.next().get(); }
    output.collect(key, new IntWritable(sum));
}}
```

Hadoop MapReduce Programming in Java

```
public static void main(String[] args) throws Exception {
  JobConf conf = new JobConf(WordCount.class);
  conf.setJobName("wordcount");
  conf.setOutputKeyClass(Text.class);
  conf.setOutputValueClass(IntWritable.class);
  conf.setMapperClass(Map.class);
  conf.setCombinerClass(Reduce.class);
  conf.setReducerClass(Reduce.class);
  conf.setInputFormat(TextInputFormat.class);
  conf.setOutputFormat(TextOutputFormat.class);
  FileInputFormat.setInputPaths(conf, new Path(args[0]));
  FileOutputFormat.setOutputPath(conf, new Path(args[1]));

  JobClient.runJob(conf);
}}
```

MAP/REDUCE EXAMPLE #2 (BUILDING AN INVERTED INDEX)

An Example of Indexing

Documents(t)
Financial, IMF, Economics, Crisis
Financial, IMF, Crisis

Documents(t)
Economics, Harry
Financial, Harry, Potter, Film
Crisis, Harry, Potter

Map

Key	Value
Financial	1,1
IMF	1,2
Economics	1,3
Crisis	1,4
Financial	2,1
IMF	2,2
Crisis	2,3

An Example of Indexing

Documents(t)
Financial, IMF, Economics, Crisis
Financial, IMF, Crisis

Documents(t)
Economics, Harry
Financial, Harry, Potter, Film
Crisis, Harry, Potter

Map

Key	Value
Economics	3,1
Harry	3,2
Financial	4,1
Harry	4,2
Potter	4,3
Film	4,4
Crisis	5,1
Harry	5,2
Potter	5,3

An Example of Indexing

Documents(t)
Financial, IMF, Economics, Crisis
Financial, IMF, Crisis

Documents(t)
Economics, Harry
Financial, Harry, Potter, Film
Crisis, Harry, Potter

Combine

Combine

Key	Value list
Financial	(1,1),(2,1),(4,1)
IMF	(1,2),(2,2)
Economics	(1,3),(3,1)
Crisis	(1,4),(2,3),(5,1)
Harry	(3,2),(4,2),(5,2)
Film	(4,4)
Potter	(4,3),(5,3)

Reduce

Reduce

Key	Value
Financial	(1,1),(2,1),(4,1)
IMF	(1,2),(2,2)
Economics	(1,3),(3,1)
Crisis	(1,4),(2,3),(5,1)
Harry	(3,2),(4,2),(5,2)
Film	(4,4)
Potter	(4,3),(5,3)

Before reduce functions are called,
for each distinct key,
the list of its values are generated

MAP/REDUCE EXAMPLE #3 (AGGREGATION IN NOSQL)

Practice with

mongoDB

Characteristics

Short for Binary JSON: Binary-encoded serialization

	mongoDB
Data Model	Document-oriented (based on BSON)
Interface	Custom protocol over TCP/IP
Object Storage	Database contains **collections (=tables)** Collections contains **documents (=rows)**
Query Method	**MapReduce (javascript)** creating collections + Object-cased query language
Replication	Master-Slave
Concurrency	Update in-place
Written In	C++

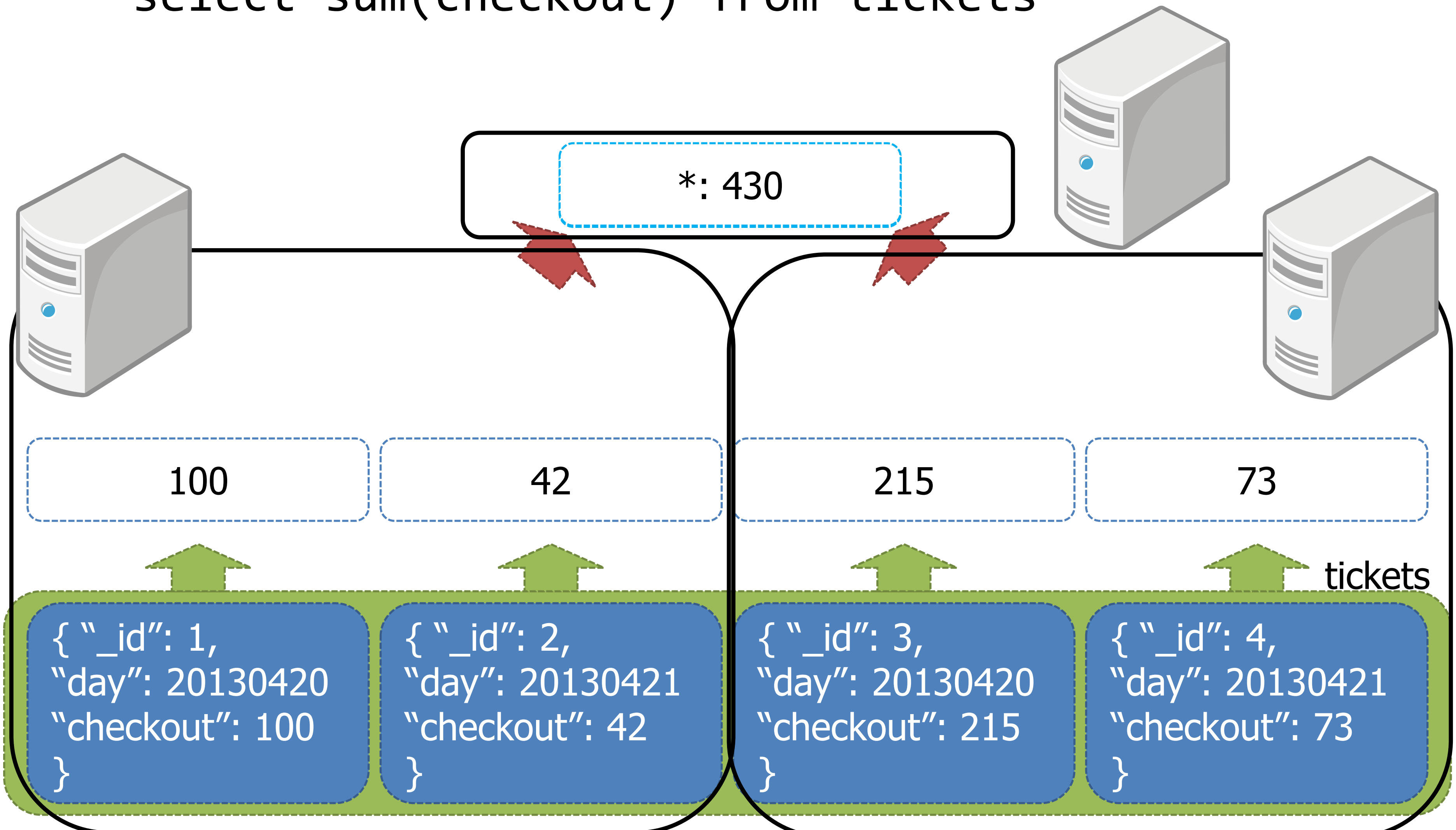
Select-Where Query

- Example:
  - select * from colors where name=‘green’;
    → db.colors.find({**name:’green’**})
  - select * from people where age <= 27;
    → db.people.find({**age:{$lte:27}**})

연산자
$gt
$gte
$lt
$lte
$ne
$in
…

Summation with MapReduce

- `select sum(checkout) from tickets`



Example

```
db.tickets.mapReduce(
  function() {
    emit( ‘*’  , this.checkout);
  },
  function(key, values) {
    return Array.sum(values);
  },
  { out:'postout', query:{status:'active'} }
).find();
```

```
→ { "_id" : “*”, "value" : 430 }
```

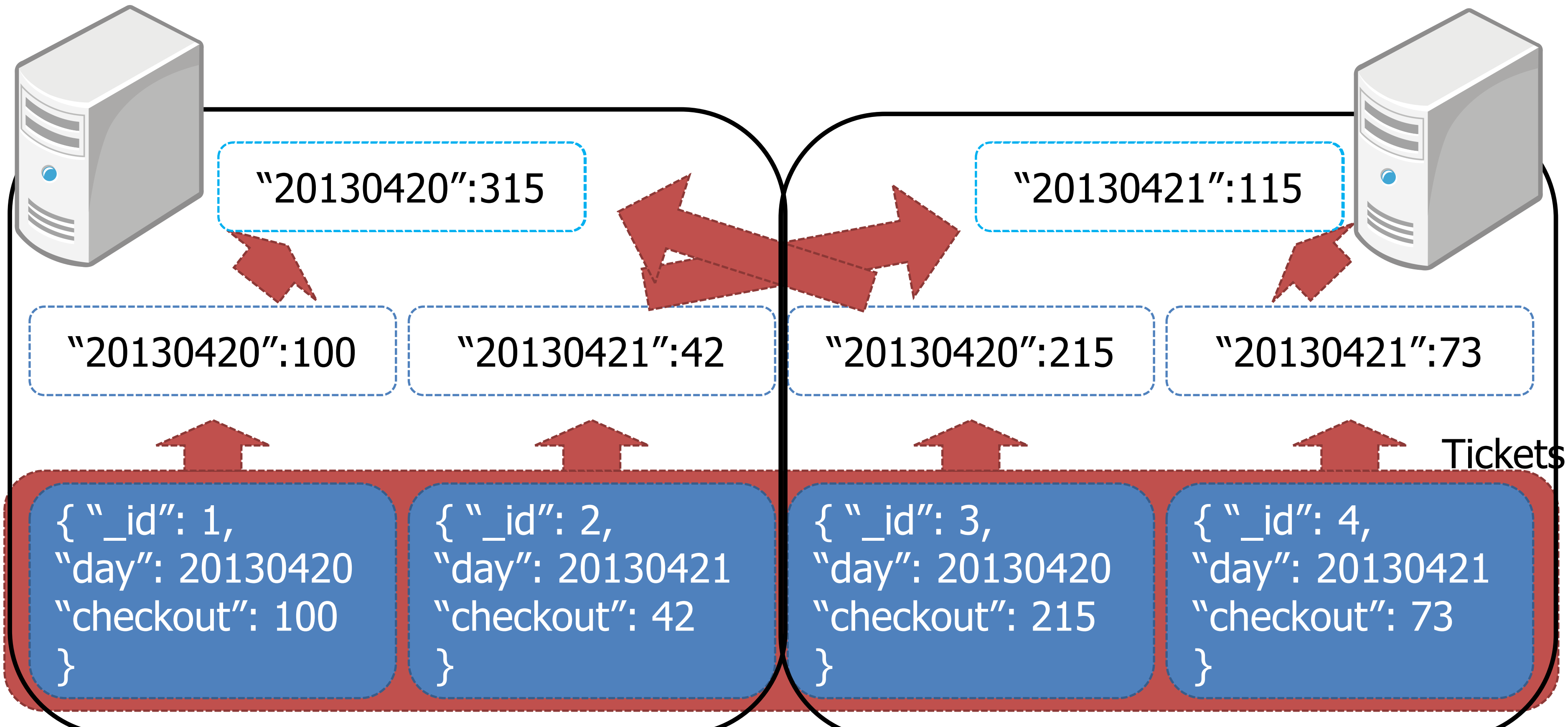
MapReduce Commend

- `> db.collection.mapReduce(`
- `function() { … emit(key,value); … },`
- `function(key,values) {return output_value;},`
- `{`
  - `out: collection,`
  - `query: document,`
  - `sort: document,`
  - `limit: number`
- `})`

> - map is a javascript function that maps a value with a key and emits a key-value pair
> - reduce is a javascript function that reduces or groups all the documents having the same key
> - out specifies the location of the map-reduce query result
> - query specifies the optional selection criteria for selecting documents
> - sort specifies the optional sort criteria
> - limit specifies the optional maximum number of documents to be returned

Groupby with MapReduce

- `select sum(checkout) from tickets group by day`



Groupby with MapReduce

- ```
  db.tickets.mapReduce(
    function() {
      emit( this.day, this.checkout);
    },
    function(key, values) {
      return Array.sum(values);
    },
    {out:'groupby'}
  ).find()
  ```
- `{ "_id" : 20130420, "value" : 315 }`
- `{ "_id" : 20130421, "value" : 115 }`
  ```

MAP/REDUCE EXAMPLE #4 (LOGISTIC REGRESSION WITH SPARK)

Practice with

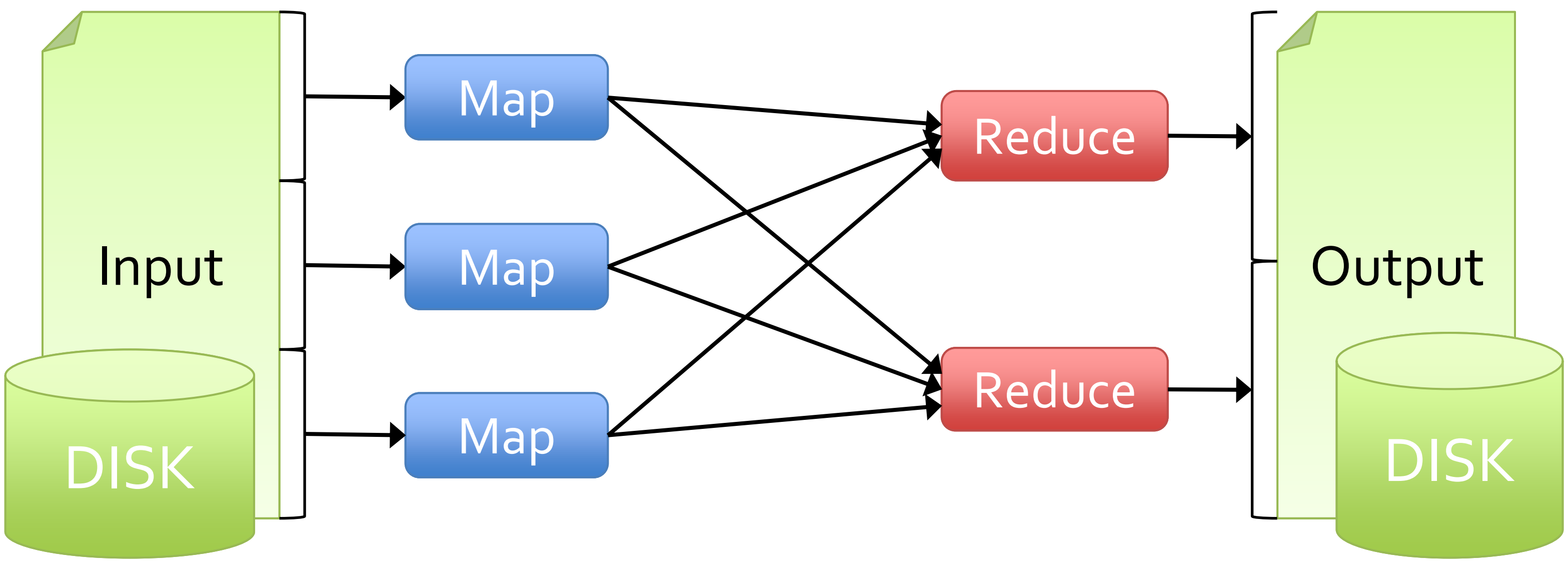
APACHE Spark™

In-Memory cluster computing for

Iterative and Interactive Applications

Motivation

- Popular MapReduce implementations such as Hadoop transform data flowing from <u>stable storage</u> to <u>stable storage</u>



[Harold Liu with UC, Berkeley]

Motivation

- Acyclic data flow is a powerful abstraction, but is not efficient for applications that repeatedly reuse a working set of data:
  - Iterative algorithms (many in machine learning, e.g., PageRank, EM algorithms)
  - Interactive data mining tools (R, Excel, Python)
- Spark introduces augment data flow model with "resilient distributed datasets" (RDDs)

~~Resilient distributed datasets (RDDs)~~

- An RDD is an immutable, partitioned, logical collection of records
  - Need not be materialized, but rather contains information to rebuild a dataset from stable storage
- Partitioning can be based on a key in each record (using hash or range partitioning)
- Built using bulk transformations on other RDDs
- Can be cached for future reuse

[Harold Liu with UC, Berkeley]

Simple Spark Apps: Word Counting

Scala:

RDD

```
val f = sc.textFile("README.md")
val wc = f.flatMap(l => l.split(" ")).map(word => (word, 1)).reduceByKey(_ + _)
wc.saveAsTextFile("wc_out")
```

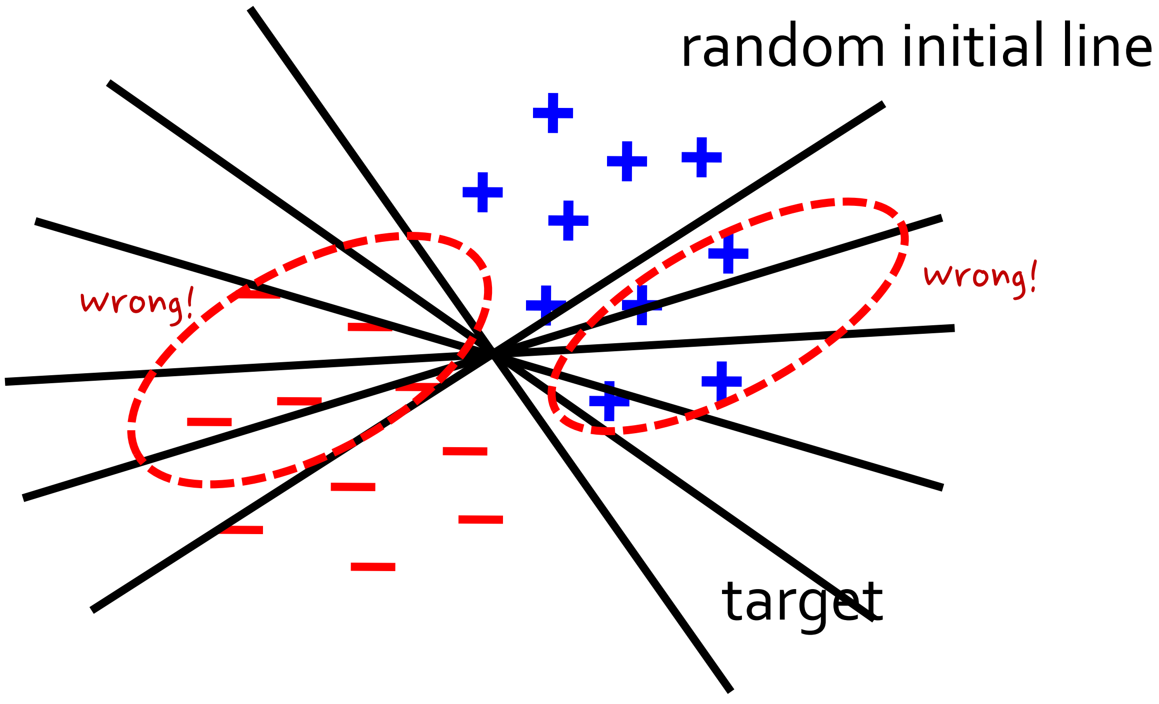
Python:

```
from operator import add
f = sc.textFile("README.md")
wc = f.flatMap(lambda x: x.split(' ')).map(lambda x: (x, 1)).reduceByKey(add)
wc.saveAsTextFile("wc_out")
```

[Harold Liu with UC, Berkeley]

Logistic Regression

- Goal: find the **best line separating** two sets of points



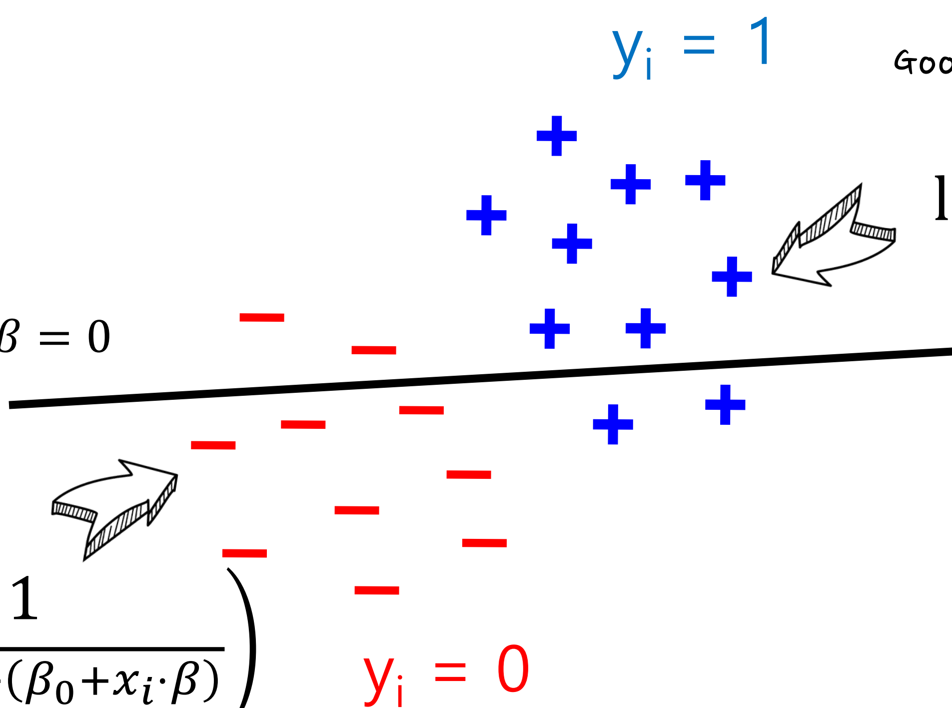
[Harold Liu with UC, Berkeley]

Logistic Regression

- Goal: find the **best line separating** two sets of points

Hyperplane:

$$\beta_0 + x_i \cdot \beta = 0$$



Good as much as

$$\log \frac{1}{1 + e^{-(\beta_0 + x_i \cdot \beta)}}$$

Good as much as

$$\log \left(1 - \frac{1}{1 + e^{-(\beta_0 + x_i \cdot \beta)}} \right)$$

Optimization Problem

- Maximize

$$\sum_{i=1}^{n} y_i \cdot \log\left(\frac{1}{1+e^{-(\beta_0 + x_i \cdot \beta)}}\right) + \sum_{i=1}^{n} (1-y_i) \cdot \log\left(1 - \frac{1}{1+e^{-(\beta_0 + x_i \cdot \beta)}}\right)$$

- Gradient descent method

Sum of values calculated with each data points

$$\beta^{(t+1)} \leftarrow \beta^{(t)} + \alpha \sum_{i=1}^{n} \boxed{\left(y_i - \frac{1}{1+e^{-(\beta_0 + x_i \cdot \beta)}}\right) x_i}$$

Logistic Regression Code

RDD

```
val data = spark.textFile(...).map(readPoint).cache()

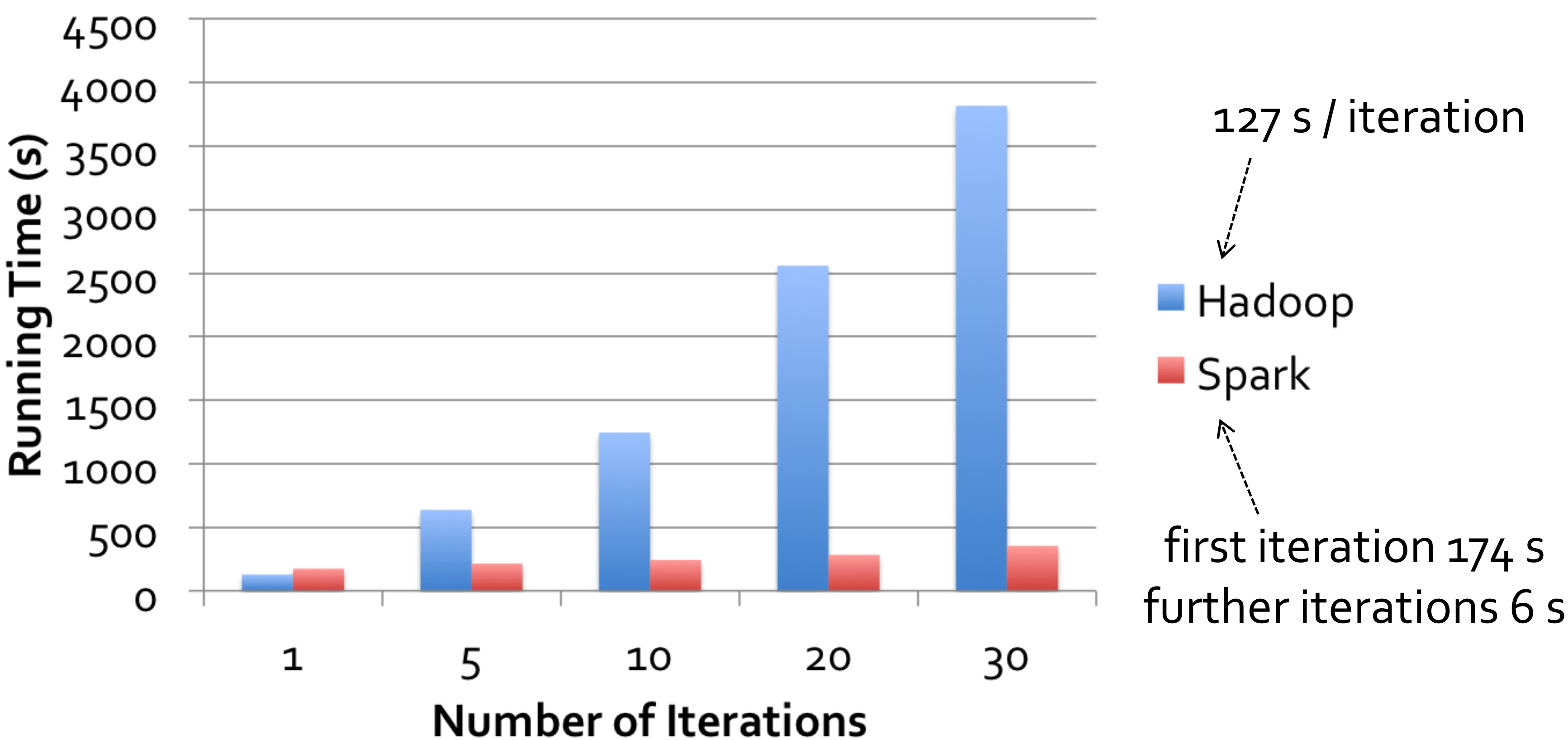
var w = Vector.random(D)

for (i <- 1 to ITERATIONS) {
   val gradient = data.map(p =>
       (p.y - 1 / (1 + exp(-(w dot p.x)))) * p.x
   ).reduce(_ + _)
   w += gradient
}

println("Final w: " + w)
```

[Harold Liu with UC, Berkeley]

Logistic Regression Performance



[Harold Liu with UC, Berkeley]

Summary

Programming with MapReduce is not a choice, but a necessity.

Don't worry. It is fun!