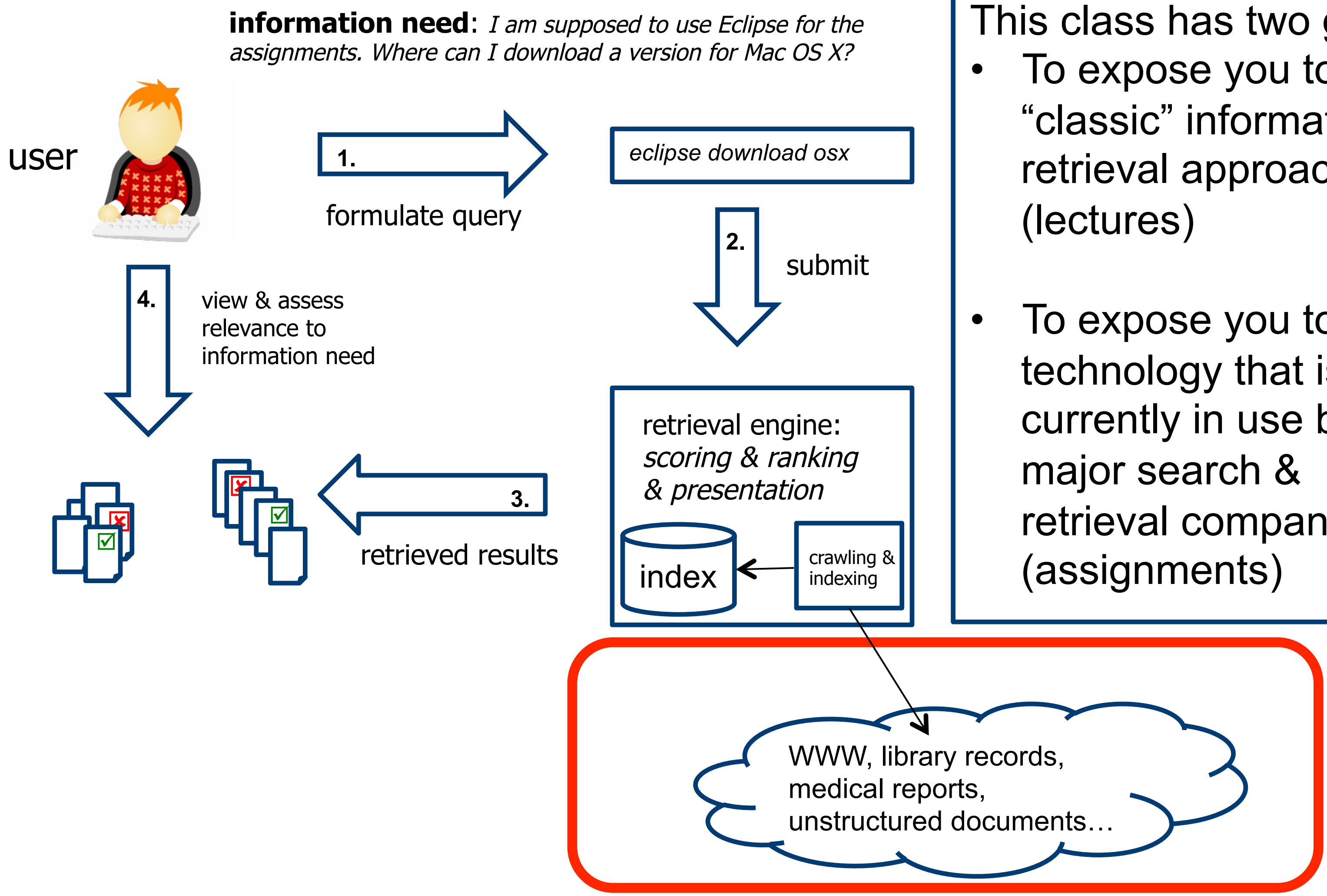
Big Data

IN4325 – Information Retrieval

Organization matters

- The deadline for the final assignment (“first week of Q4”)
  - Teaching week 4.1: April 23 to April 29
  - Thus, **final (hard) deadline: April 29, 2012**
- SEPAM students can opt for less programming-intensive assignments (though Hadoop is really fun!)

Today: How to process big data!

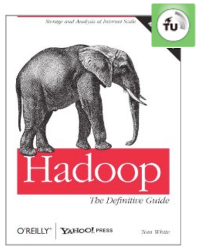

information need: *I am supposed to use Eclipse for the assignments. Where can I download a version for Mac OS X?*
user
1.
formulate query
eclipse download osx
2.
submit
retrieval engine:
scoring & ranking
& presentation
index
crawling & indexing
WWW, library records, medical reports, unstructured documents…
3.
retrieved results
4.
view & assess relevance to information need

This class has two goals:

- To expose you to "classic" information retrieval approaches (lectures)
- To expose you to technology that is currently in use by all major search & retrieval companies (assignments)

Reading material for the assignments

- Hadoop programming
  - *Hadoop: The Definitive Guide* by Tom White, O'Reilly Media, 2011.
  - Many tutorials exist online (e.g. by Yahoo!, Cloudera)



- MapReduce algorithm design
  - *Data-Intensive Text Processing with MapReduce* by Lin, Dyer and Hirst, Morgan and Claypool Publishers, 2010
    - Available online: http://www.umiacs.umd.edu/~jimmylin/book.html
    - Take a look at the rest of Jimmy Lin's website too (many Hadoop examples, source code, lectures)
  - This lecture is largely based on the book.

What is 'Big data'?

- "Big data refers to enormous amounts of unstructured data produced by high-performance applications"
  - Scientific computing applications
  - Social networks
  - E-government applications
  - Medical information systems
- Issues
  - Scalability
  - Heterogeneity
  - Data analysis

How big is big?

- Data repositories will only grow bigger with time.
- More data usually translates into more effective algorithms.

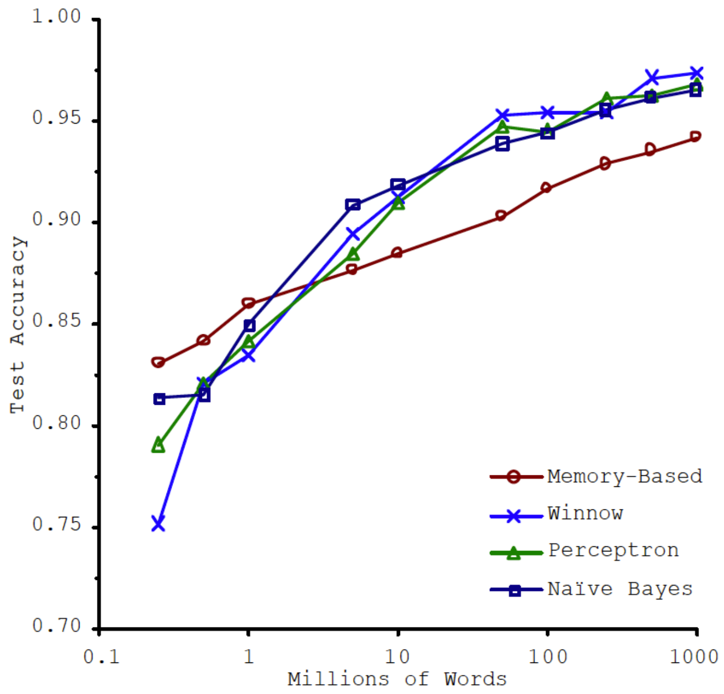
- YouTube: 4 billion views a day, one hour of video upload per second
- Facebook: 483 million daily active users (December 2011)
- Twitter: 140 million tweets on average per day (March 2011)
- Google: >1 billion searches per day (March 2011)

- Google processed 100 TB of data per day in 2004 and 20 PB data per day in 2008
- Large Hadron Collider at CERN: when fully functional, it will generate 15 petabytes of data per year
- Internet Archive: contains 2 petabytes of data, grows 20 terabytes per month (2011)

The more data, the better

The unreasonable effectiveness of data. A. Halevy, P. Norvig and F. Pereira, 2009.

“So, follow the data.”



Confusion set disambiguation

- *then* vs. *than*
- *to* vs. *two* vs. *too*
- ….

Scaling to very very large corpora for natural language disambiguation. M. Banko and E. Brill, 2001.

Cloud computing

- “Anything running inside a browser that gathers and stores user-generated content” (Jimmy Lin)
- Utility computing
  - A computing resource as a metered service
  - A “cloud user” buys any amount of computing power from a “cloud provider” (pay-per-use)
    - Virtual machine instances
  - IaaS: infrastructure as a service
  - Amazon Web Services (EC2: elastic compute cloud, S3: simple storage service) is the dominant provider

MapReduce

Clear separation between ***what*** to compute and ***how*** to compute it on the cluster.

① Programming model for distributed computations on **large-scal**e data, inspired by the functional programming paradigm

② Execution framework for clusters of commodity hardware

- Developed by researchers at Google in 2003
  - Built on principles in parallel and distributed processing

- "MapReduce is used for the generation of data for Google's production web search service, for sorting, for data mining, for machine learning and many other systems" [12]

- Designed for **batch** processing over large data sets

Ideas behind MapReduce I

- Scale “out”, not “up”
  - Many commodity servers are more cost effective than few high-end servers
- Assume failures are common
  - A 10,000-server cluster with a mean-time between failures of 1000 days experiences on average 10 failures a day.

- Move processes to the data
  - Moving the data around is expensive
  - Data locality awareness
- Process data sequentially and avoid random access
  - Data sets do not fit in memory, disk-based access (slow)
  - Sequential access is orders of magnitude faster

Ideas behind MapReduce II

- Hide system-level details from the application developer
  - Frees the developer to think about the task at hand only (no need to worry about deadlocks, …)
  - MapReduce takes care of the system-level details (separation of what and how to compute)

- Seamless scalability
  - Data scalability (given twice as much data, the ideal algorithm runs twice as long)
  - Resource scalability (given a cluster twice the size, the ideal algorithm runs in half the time)

Ideas behind MapReduce II

- Hide system-level details
  - Frees the developer to th
    need to worry about dea
  - MapReduce takes care o
    what and how to compu

System-level details:
•Data partitioning
•Scheduling, load balancing
•Fault tolerance
    •Machine failures are common in clusters of thousands of commodity machines
•Inter-machine communication

- Seamless scalability
  - Data scalability (given twice as much data, the ideal algorithm runs twice as long)
  - Resource
    algorithm

"… MapReduce is not the final word, but rather the first in a new class of programming models that will allow us to more effectively organize computations on a massive scale." (Jimmy Lin)

Hadoop: an open-source implementation

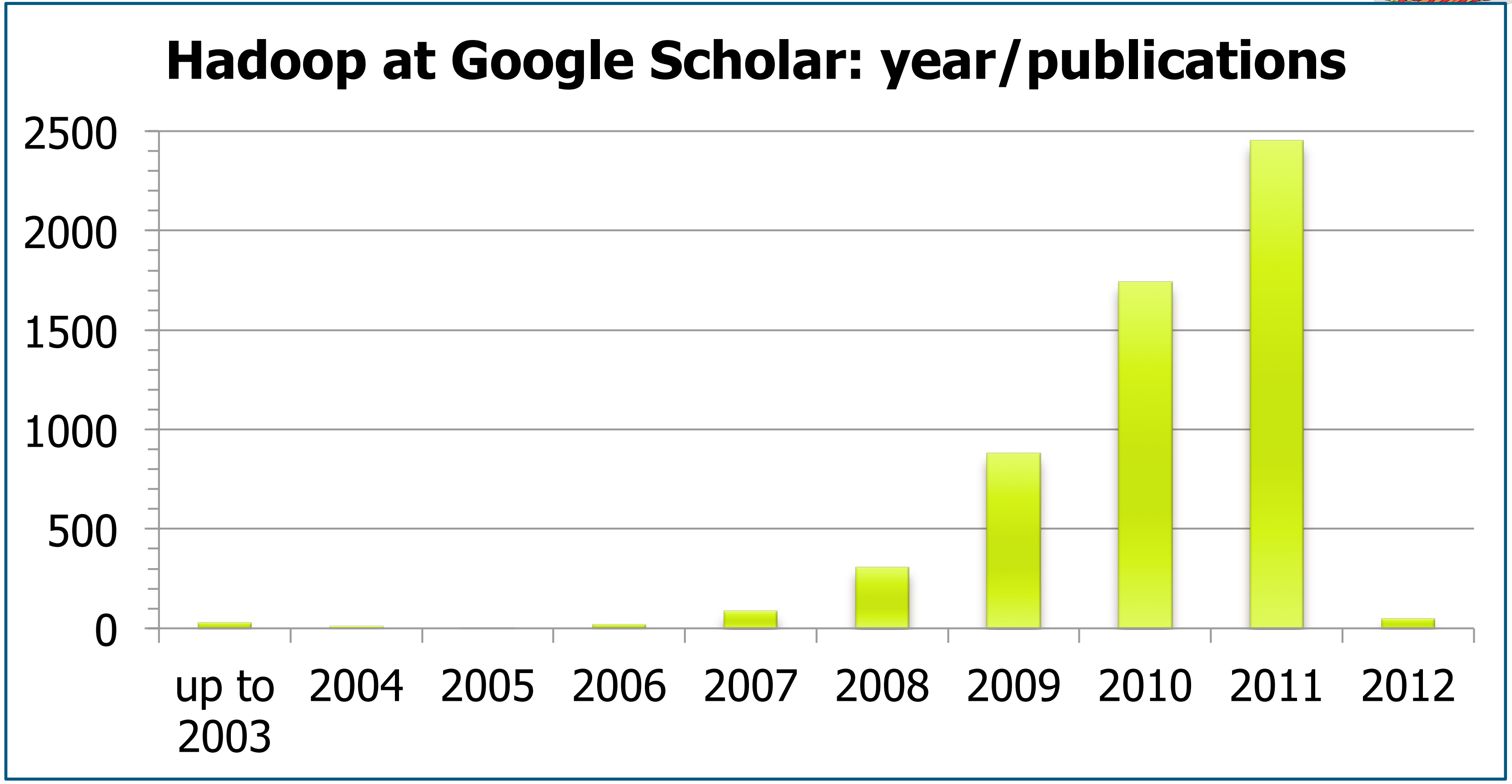
- De facto MapReduce standard, employed by Amazon, Adobe, Ebay, Facebook, IBM, Last.fm, LinkedIn, StumbleUpon, Twitter, Yahoo!, … [13]

- An Apache project (originally developed by Doug Cutting) which has quickly spawned additional Hadoop-related top-level projects

- You will be using it for your assignments
  - We focus on core Hadoop functionality

- Important to know about, Hadoop is here to stay
  - "Swiss army knife of the 21st century" [14]

Hadoop: an open-source implementation



MapReduce

- Divide & conquer: partition a large problem into smaller sub-problems
  - Independent sub-problems can be executed in parallel by **workers** (anything from threads to clusters)
  - **Intermediate** results from each worker are combined to get the **final** result

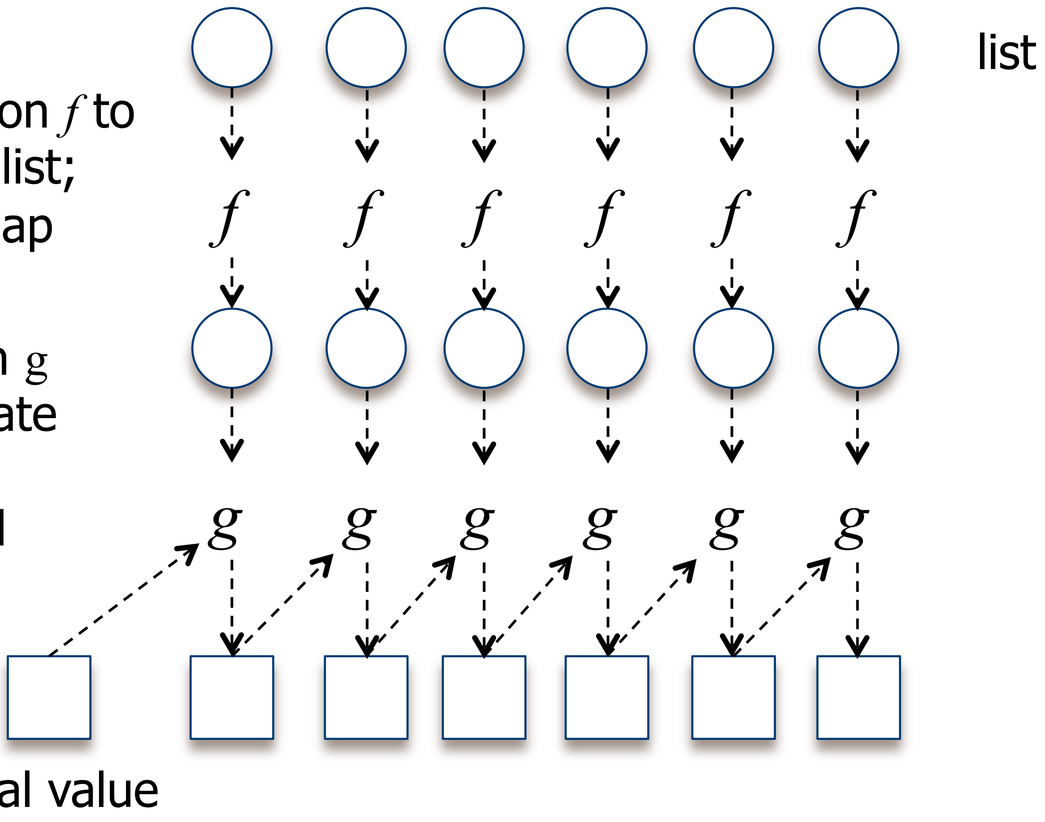
- Issues:
  - Problem ➔ sub-problems
  - Worker assignment & synchronization
  - How do the workers get the required data?
  - Sharing of intermediate results

Map & fold

Two higher-order functions

map: applies function *f* to every element in a list; *f* is argument for map

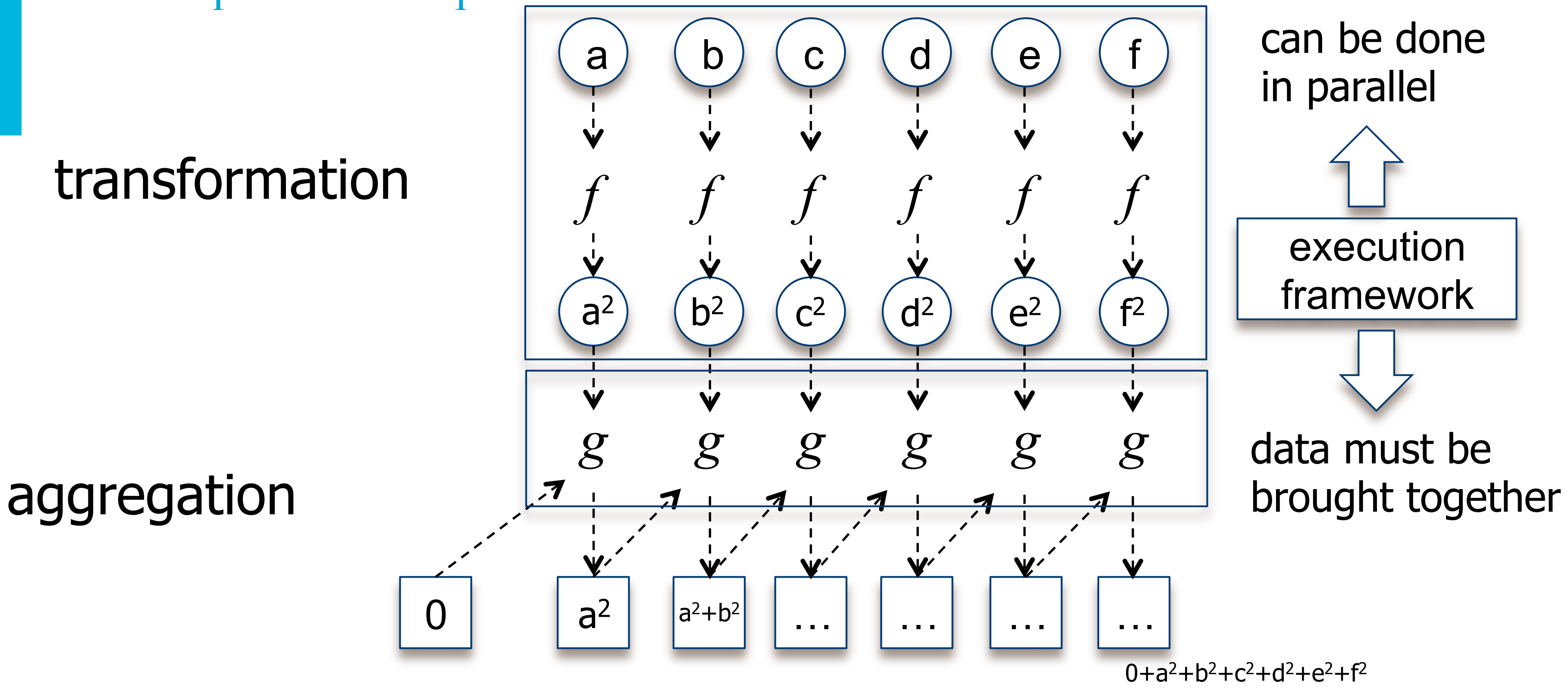
fold: applies function g iteratively to aggregate the results; g is argument of fold plus an initial value



Adapted from *Data-Intensive Text Processing with MapReduce*, Figure 2.1 (page 20).

Map & fold

Example: sum of squares



Adapted from *Data-Intensive Text Processing with MapReduce*, Figure 2.1 (page 20).

Map & reduce

Key/value pairs form the basic data structure

- Apply a map operation to each record in the input to compute a set of intermediate key/value pairs

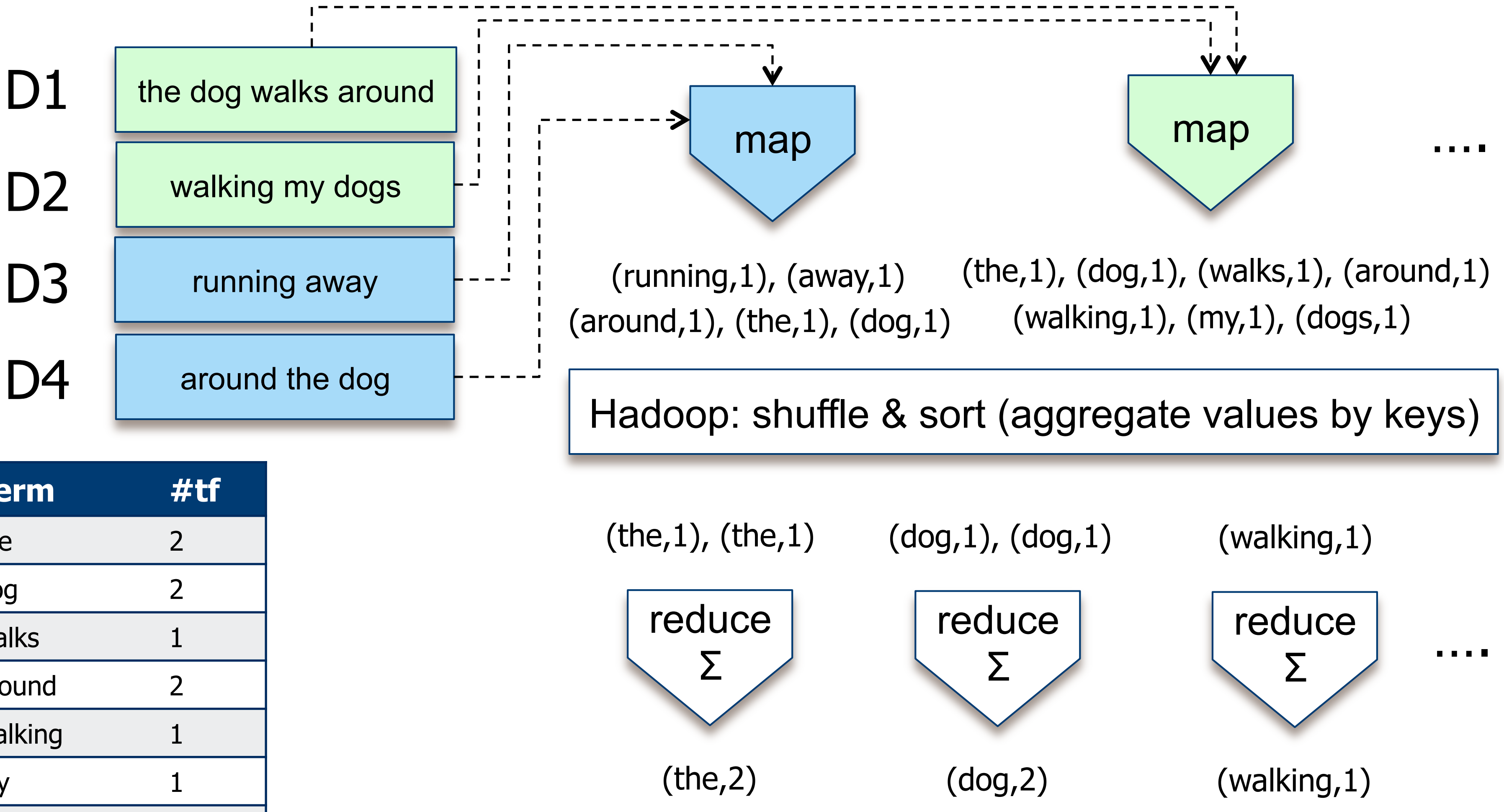
map `(k1,v1)` -> `[(k2,v2)]`

- Apply a reduce operation to all values that share the same key

reduce `(k2,[v2])` -> `[(k3,v3)]`

- The user of a MapReduce library specifies only the map and reduce functions

Example: word count I



Term	#tf
the	2
dog	2
walks	1
around	2
walking	1
my	1
….	…

Example: word count II

- Problem: count the number of occurrenes of each word w in a large corpus of documents

docid

doc content

```
map(String key, String value):
        foreach word w in value:
                EmitIntermediate(w,1);
```

intermediate key/ value pairs

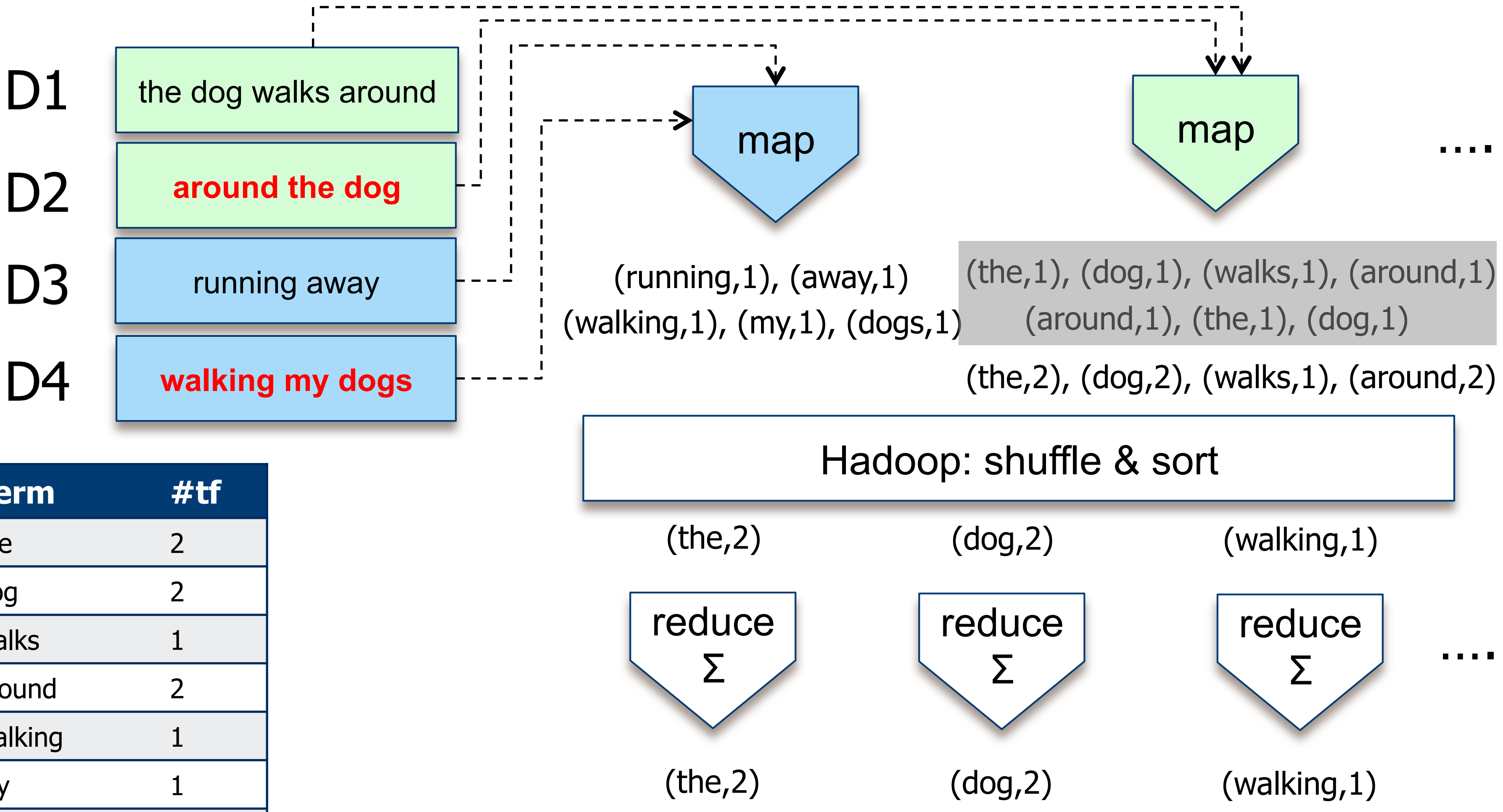
all values with the same key

word

```
reduce(String key, Iterator values):
        int res = 0;
        foreach int v in values:
                res += v;
        Emit(key, res)
```

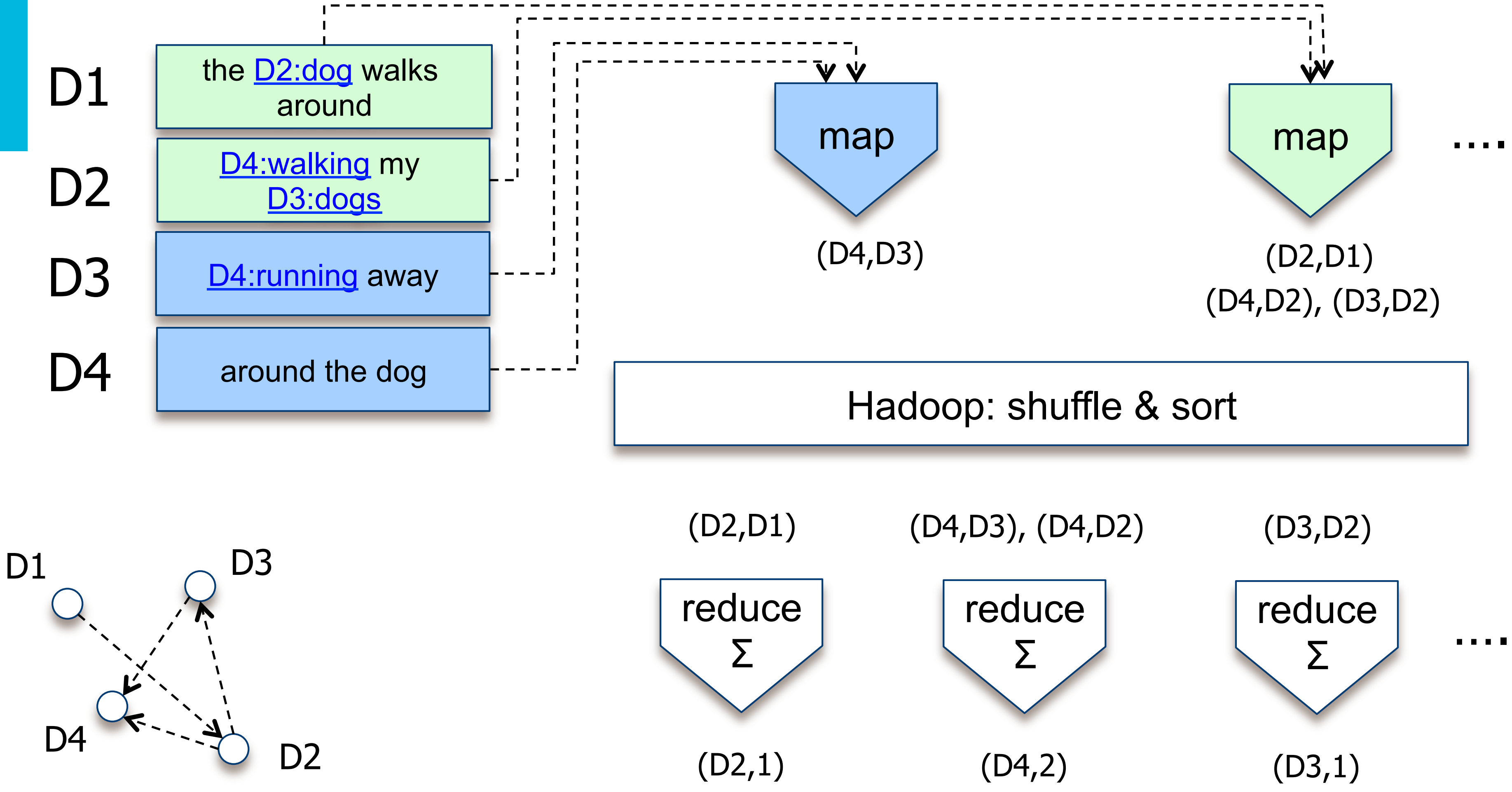
count of `key` in the corpus

Example: word count III



Term	#tf
the	2
dog	2
walks	1
around	2
walking	1
my	1
….	…

Example: inlink count I



Example: inlink count II

- Problem: collect all Web pages (`sources`) that are pointing to a Web page (`target`)

source

doc content

```
map(String key, String value):
        foreach link target t in value:
                EmitIntermediate(t,key);
```

intermediate key/ value pairs

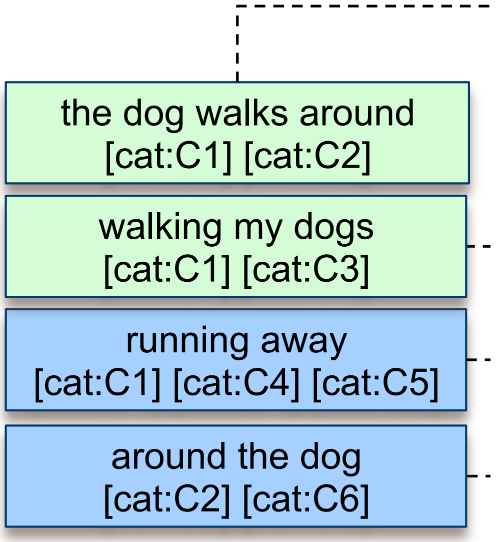
target

```
reduce(String key, Iterator values):
        int res = 0;
        foreach source s in values:
                res++;
        Emit(key,res)
```

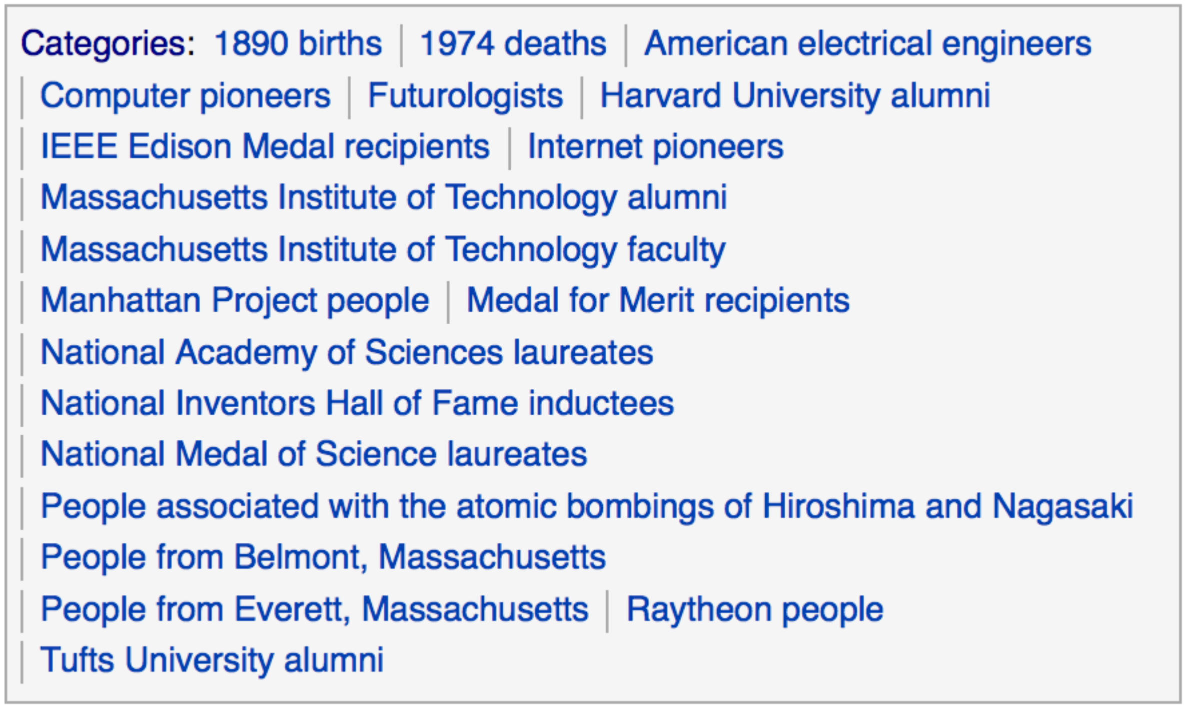
all sources (pages pointing to target)

Number of pages linking to `key`

Example: list doc categories (min. freq. 2)

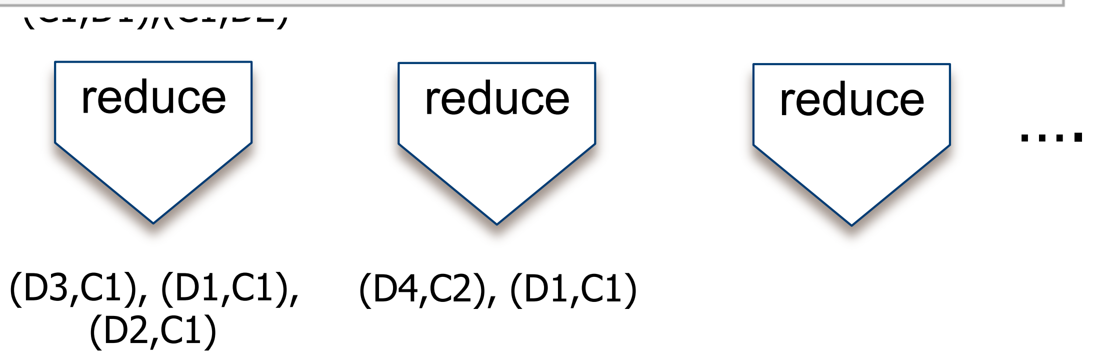


D1
the dog walks around [cat:C1] [cat:C2]
D2
walking my dogs [cat:C1] [cat:C3]
D3
running away [cat:C1] [cat:C4] [cat:C5]
D4
around the dog [cat:C2] [cat:C6]

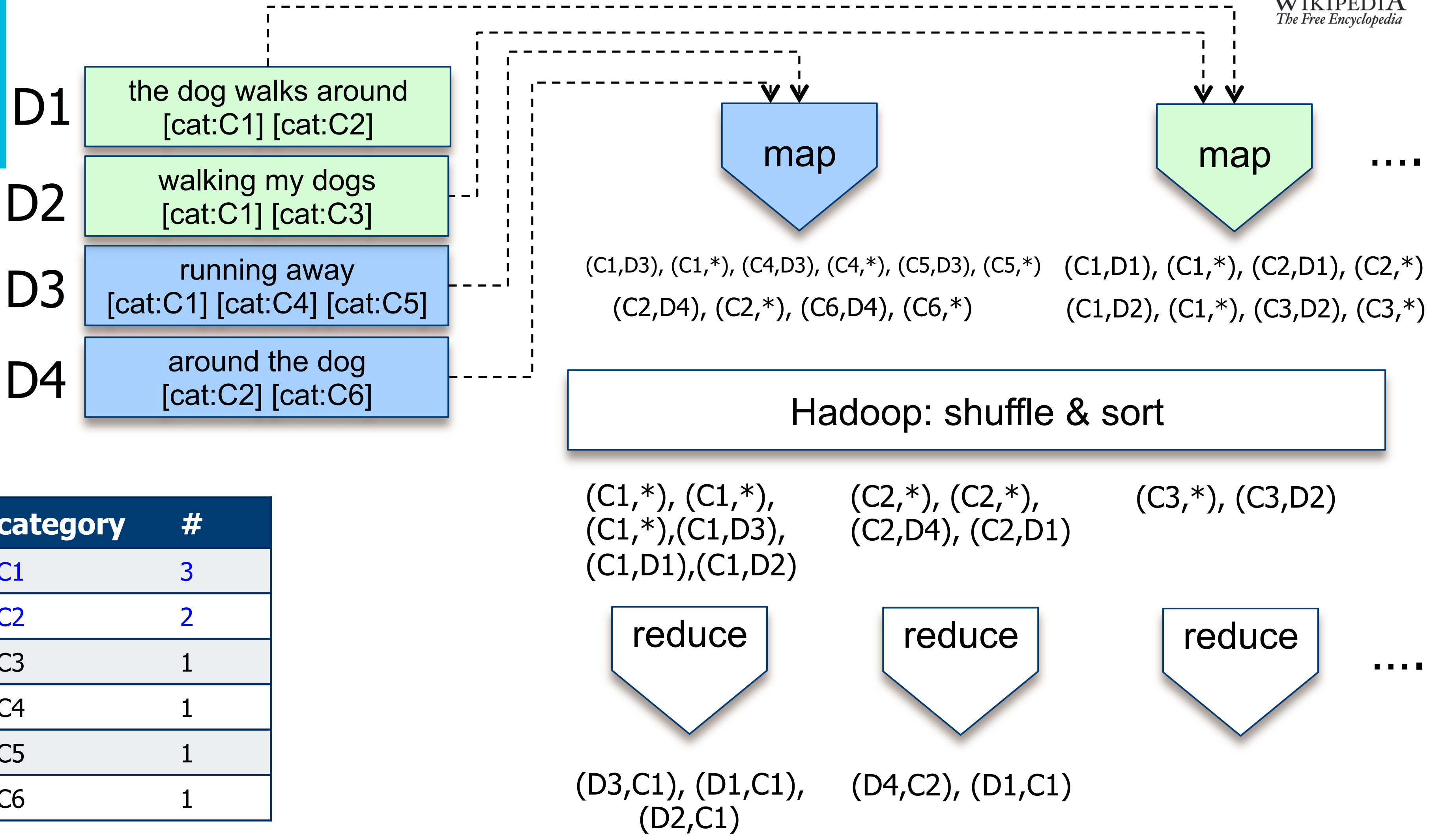


Categories: 1890 births | 1974 deaths | American electrical engineers | Computer pioneers | Futurologists | Harvard University alumni | IEEE Edison Medal recipients | Internet pioneers | Massachusetts Institute of Technology alumni | Massachusetts Institute of Technology faculty | Manhattan Project people | Medal for Merit recipients | National Academy of Sciences laureates | National Inventors Hall of Fame inductees | National Medal of Science laureates | People associated with the atomic bombings of Hiroshima and Nagasaki | People from Belmont, Massachusetts | People from Everett, Massachusetts | Raytheon people | Tufts University alumni

category	#
C1	3
C2	2
C3	1
C4	1
C5	1
C6	1



Example: list doc categories (min. freq. 2)



D1
the dog walks around [cat:C1] [cat:C2]
D2
walking my dogs [cat:C1] [cat:C3]
D3
running away [cat:C1] [cat:C4] [cat:C5]
D4
around the dog [cat:C2] [cat:C6]
map
map
....
(C1,D3), (C1,*), (C4,D3), (C4,*), (C5,D3), (C5,*)
(C2,D4), (C2,*), (C6,D4), (C6,*)
(C1,D1), (C1,*), (C2,D1), (C2,*)
(C1,D2), (C1,*), (C3,D2), (C3,*)
Hadoop: shuffle & sort
(C1,*), (C1,*), (C1,*),(C1,D3), (C1,D1),(C1,D2)
(C2,*), (C2,*), (C2,D4), (C2,D1)
(C3,*), (C3,D2)
reduce
reduce
reduce
....
(D3,C1), (D1,C1), (D2,C1)
(D4,C2), (D1,C1)

category	#
C1	3
C2	2
C3	1
C4	1
C5	1
C6	1

Example: list doc categories (min. freq. 2) II

- Problem: list all categories of a Wikipedia page that occur at least 2 times in the corpus

docid

doc content

```
map(String key, String value):
        foreach category c in value:
                EmitIntermediate(c,key);
                EmitIntermediate(c,*);
```

intermediate key/value pairs

We can emit more than one key/value pair!

category

```
reduce(String key, Iterator values):
        int total = 0;
        foreach * in values:
                total++;
        foreach docid d in values:
                if total>1
                        Emit(d,key)
```

*'s and docids

The iterator can only be used once!

Assumption: the values are sorted in a particular order (*'s before anything else)

d's category with min. freq. 2 the corpus

Example: list doc categories (min. freq. 2) III

- Problem: list all categories of a Wikipedia page that occur at least 2 times in the corpus

```
reduce(String key, Iterator values):
        List list = copyFromIterator(values);
        int total = 0;
        foreach l in list:
                if(l eq *)
                        total++;

        foreach l in list:
                if(l neq * && total>1)
                        Emit(d,key)
```

We don't assume a particular sorting.

What if there are 10GB of values for `key`? Do they fit into memory?

Zipf's law

- Term frequencies: *The Count of Monte Christo*

	Term	#tf
1.	the	28388
2.	to	12841
3.	of	12834
4.	and	12447
5.	a	9328
6.	i	8174
7.	you	8128

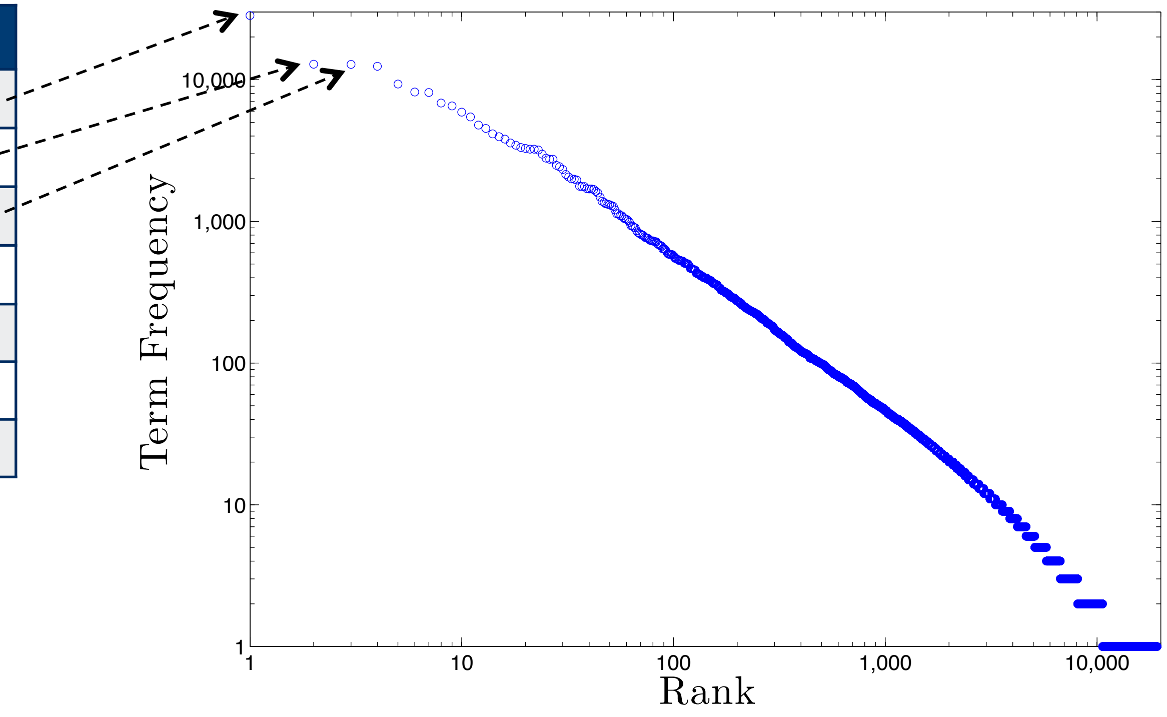
	Term	#tf
1001.	arranged	46
1002.	eat	46
1003.	terms	46
1004.	majesty	46
1005.	rising	46
1006.	satisfied	46
1007.	useless	46

	Term	#tf
19001.	calaseraigne	1
19002.	jackals	1
19003.	sorti	1
19004.	meyes	1
19005.	bets	1
19006.	pistolshots	1
19007.	francsah	1

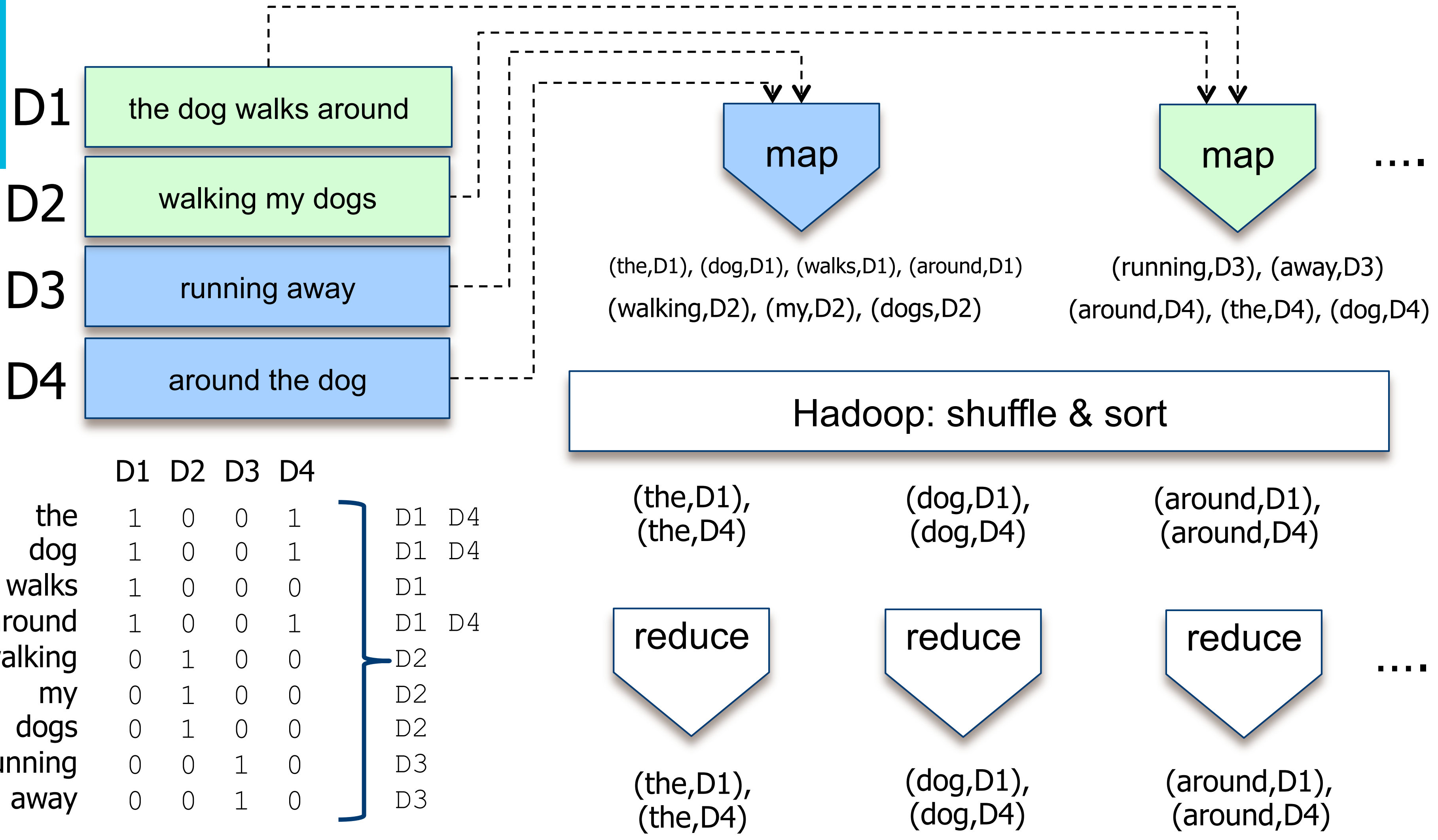
Zipf's law

- Term frequencies: *The Count of Monte Christo*

	Term	#tf
1.	the	28388
2.	to	12841
3.	of	12834
4.	and	12447
5.	a	9328
6.	i	8174
7.	you	8128



Example: a simple inverted index I



	D1	D2	D3	D4		
the	1	0	0	1	D1	D4
dog	1	0	0	1	D1	D4
walks	1	0	0	0	D1	
around	1	0	0	1	D1	D4
walking	0	1	0	0	D2	
my	0	1	0	0	D2	
dogs	0	1	0	0	D2	
running	0	0	1	0	D3	
away	0	0	1	0	D3	

Example: a simple inverted index II

- Problem: create an inverted index, i.e. for each term, list the document the term appears in

docid   doc content

```
map(String key, String value):
      foreach term t in value:
            EmitIntermediate(t,key);
```

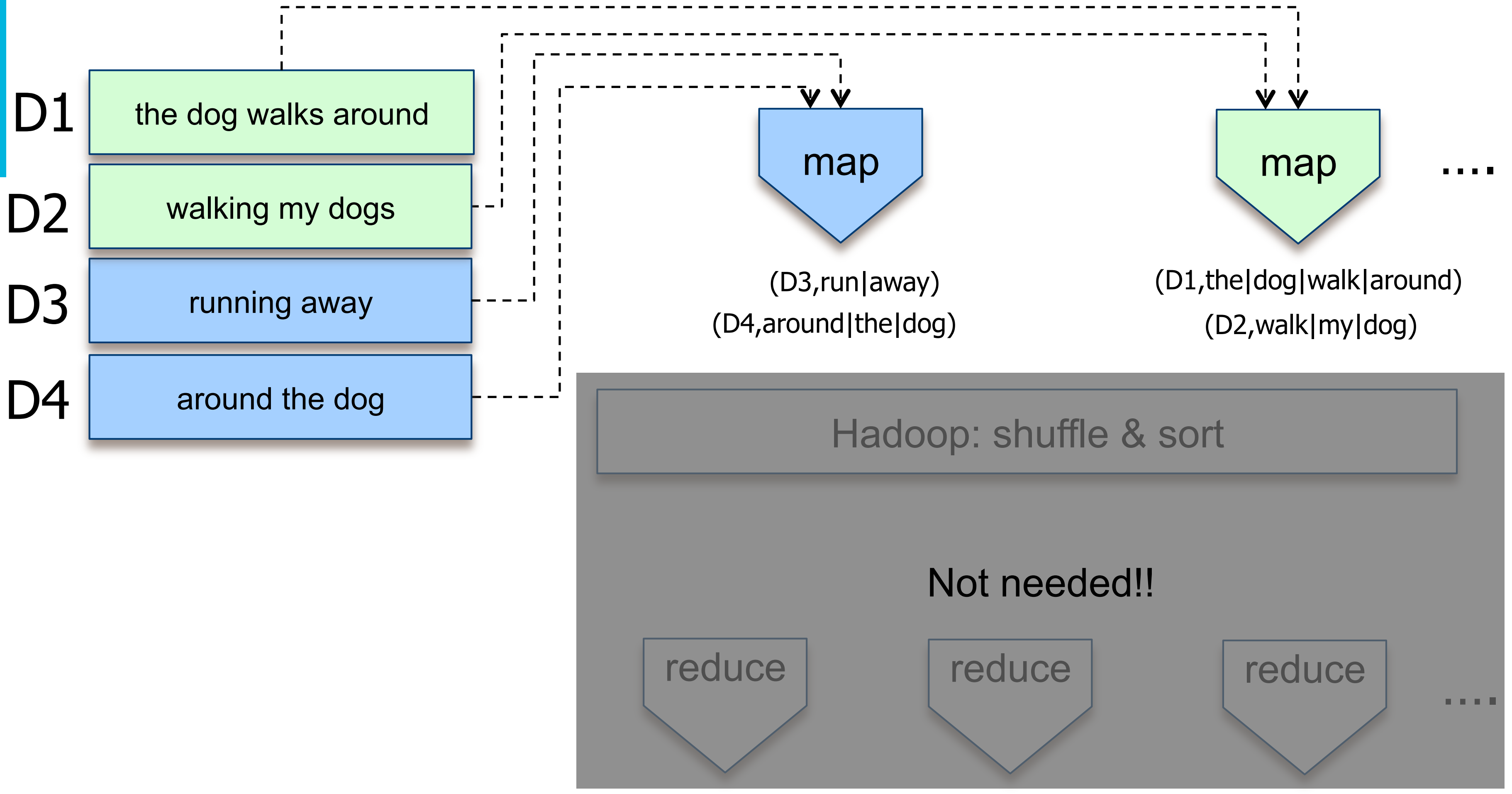
term

```
reduce(String key, Iterator values)
      foreach docid d in values:
            Emit(key,d)
```

All documents with term `key`

Not much to be done in the reducer.

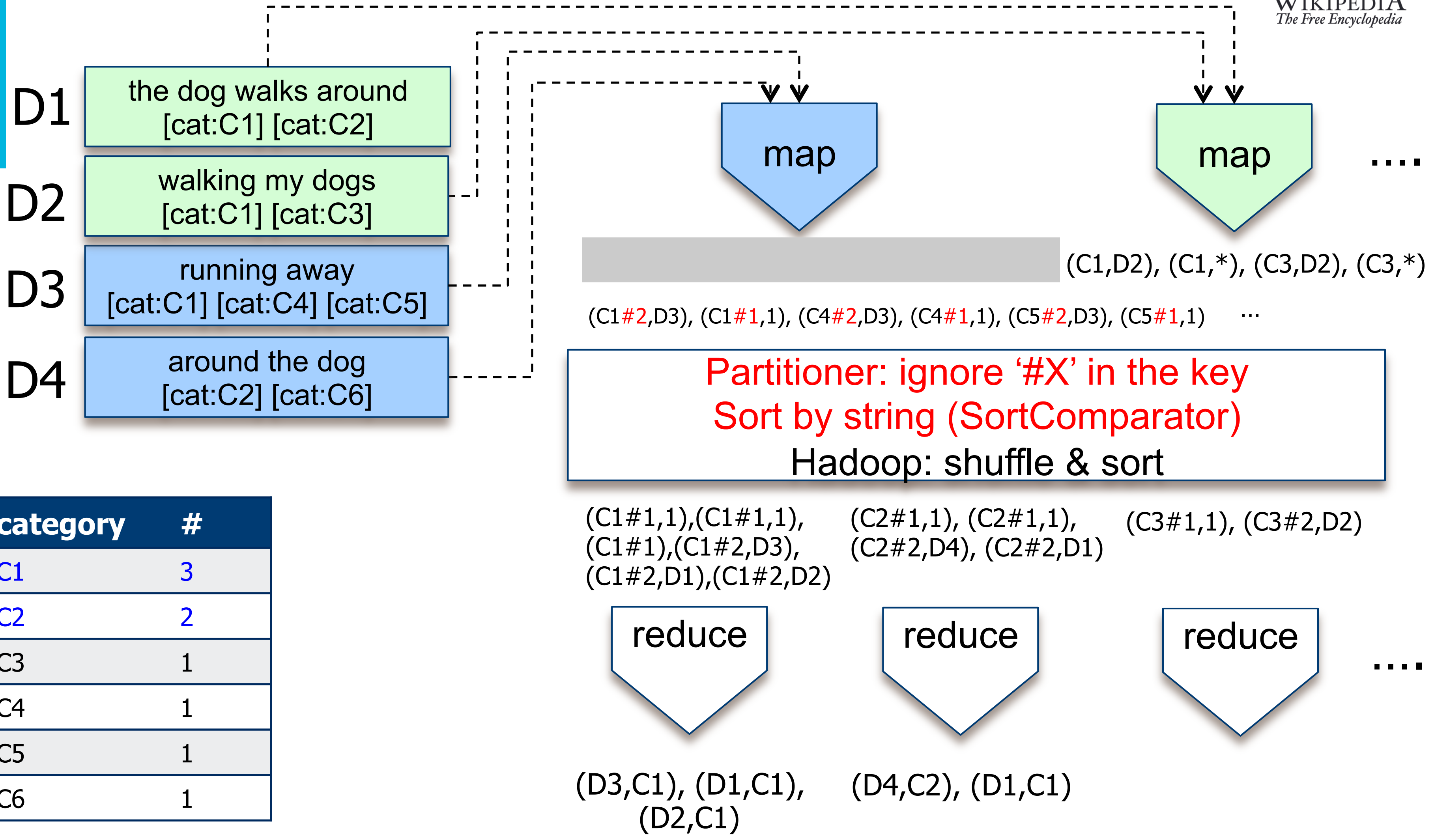
Example: parsing



There is more: combiner & partitioner I

- Partitioner: responsible for dividing the intermediate key space and assigning intermediate key/value pairs to reducers (within each reducer, keys are processed in sorted order)
    - Default: `hash(key)%NR` (Number of reducers `NR`)

Example: list doc categories (min. freq. 2)



D1
the dog walks around [cat:C1] [cat:C2]
D2
walking my dogs [cat:C1] [cat:C3]
D3
running away [cat:C1] [cat:C4] [cat:C5]
D4
around the dog [cat:C2] [cat:C6]
map
map
....
(C1,D2), (C1,*), (C3,D2), (C3,*)
(C1#2,D3), (C1#1,1), (C4#2,D3), (C4#1,1), (C5#2,D3), (C5#1,1) …
Partitioner: ignore '#X' in the key
Sort by string (SortComparator)
Hadoop: shuffle & sort
(C1#1,1),(C1#1,1), (C1#1),(C1#2,D3), (C1#2,D1),(C1#2,D2)
(C2#1,1), (C2#1,1), (C2#2,D4), (C2#2,D1)
(C3#1,1), (C3#2,D2)
reduce
reduce
reduce
....
(D3,C1), (D1,C1), (D2,C1)
(D4,C2), (D1,C1)

category	#
C1	3
C2	2
C3	1
C4	1
C5	1
C6	1

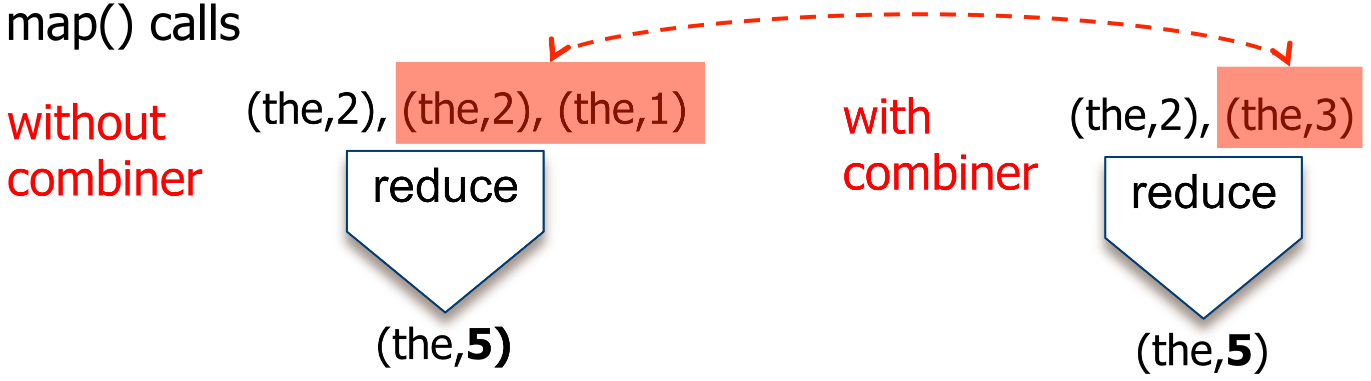
There is more: combiner & partitioner I

- Partitioner: responsible for dividing the intermediate key space and assigning intermediate key/value pairs to reducers (within each reducer, keys are processed in sorted order)
  - Default: `hash(key)%NR` (Number of reducers `NR`)
  - Custom: `hash(key.substring(0,key.lastIndexOf('#'))%NR`

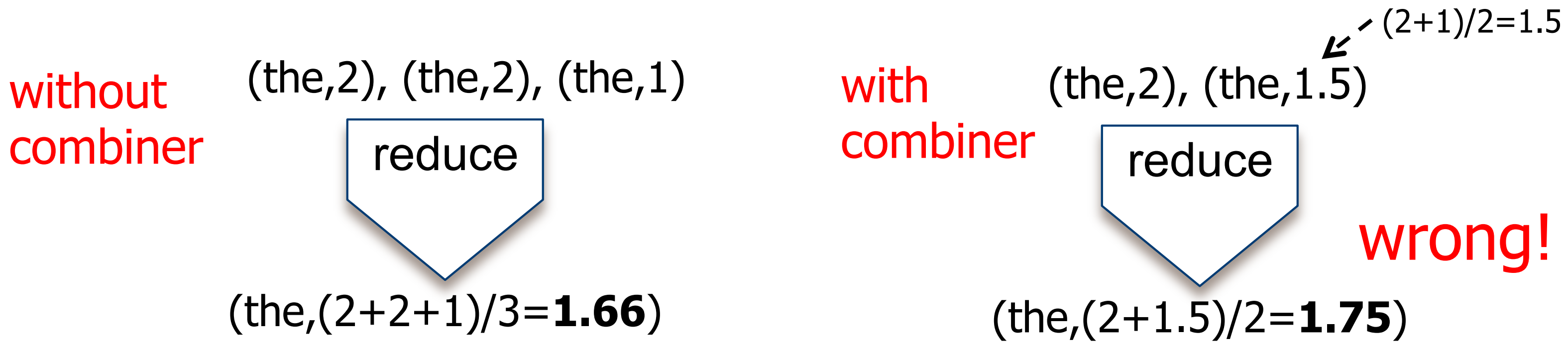
- Combiner ("mini-reducer"): local aggregation before the shuffle & sort phase
  - Instead of emitting 100 times *(the,1*), the combiner can emit *(the,100)*
  - Can lead to great speed-ups
  - Needs to be employed with care
    - If the reduce operation is associative & commutative (e.g. multiplication), the reducer code can serve as combiner

There is more: combiner & partitioner II

- Task: determine the term frequencies in a corpus
  - Assume a mapper that outputs `(term,termFreqInDoc)` pairs
  - Assume the combiner is a reducer 'copy' that aggregates the results for 2 map() calls



- Average frequency of terms in documents (combiner as reducer 'copy')



There is more: combiner & partitioner II

- Combiner cont.
  - Each combiner operates in isolation, has no access to other mapper's key/value pairs
  - A combiner cannot be assumed to process all values associated with the same key
  - Emitted key/value pairs must be the same as those emitted by the mapper
  - Most often, combiner code != reducer code

Hadoop in practice I

- Specified by the user
  - Mapper
  - Reducer
  - Combiner (optional)
  - Partitioner (optional)
  - Job configuration

Hadoop in practice II

Mapper (counting inlinks)

input key/value: sourceUrl, content
output key/value: targetUrl, 1

```
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;

public class InlinkCount extends Mapper<Object,Text,Text,IntWritable>
{
        IntWritable one = new IntWritable(1);
        Pattern linkPattern = Pattern.compile("\\[\\[.+?\\]\\]");

        public void map(Object key, Text value, Context con) throws Exception
        {
                String page = value.toString();
                Matcher m = linkPattern.matcher(page);
                while(m.find())
                {
                        String match = m.group();
                        con.write(new Text(match),one);
                }
        }
}
```

Hadoop in practice III

Reducer (counting inlinks)

input key/value: targetUrl, 1
output key/value: targetUrl, count

```
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;

public class SumReducer extends Reducer<Text,IntWritable,Text,IntWritable>
{
        public void reduce(Text key,Iterable<IntWritable> values,Context con)
                throws Exception
        {
                int sum = 0;
                for(IntWritable iw : values)
                        sum += iw.get();

                con.write(key, new IntWritable(sum));
        }
}
```

Hadoop in practice IV

Driver (counting inlinks)

```
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;
…
public class InlinkCountDriver
{
        public static void main(String[] args) throws Exception
        {
                Configuration conf = new Configuration();
                String[] otherArgs = new GenericOptionsParser
                        (conf,args).getRemainingArgs();
                Job job = new Job(conf, "InlinkAccumulator");
                job.setMapperClass(InlinkCountMapper.class);
                job.setCombinerClass(SumUpReducer.class);
                job.setReducerClass(SumUpReducer.class);
                job.setOutputKeyClass(Text.class);
                job.setOutputValueClass(IntWritable.class);

                FileInputFormat.addInputPath(job,new Path("/user/in/"));
                FileOutputFormat.setOutputPath(job,new Path("/user/out/"));
                job.waitForCompletion(true);
        }
}
```

Hadoop in practice V

Driver (job chaining)

```
import …
public class InlinkCountDriver
{
          public static void main(String[] args) throws Exception
          {
                    Configuration conf = new Configuration();
                    Job job1 = new Job(conf, “Categories Per Page");
                    job1.setMapperClass(CategoryPerPageMapper.class);
                    job1.setReducerClass(AggregateAndPruningReducer.class);
                    …
                    FileInputFormat.addInputPath(job1,new Path("/user/in/"));
                    FileOutputFormat.setOutputPath(job1,new Path("/user/out1/"));
                    job.waitForCompletion(true);

                    Job job2 = new Job(conf, “Reformatter”);
                    job2.setMapperClass(LineToRecordStringStringMapper.class);
                    job2.setReducerClass(WritePairsReducer.class);
                    …
                    FileInputFormat.addInputPath(job2, new Path(“/user/out1/”));
                    FileInputFormat.setOutputPath(job2, new Path(“/user/out2/”));

                    job.waitForCompletion(true);
                    job2.waitForCompletion(true);
          }
}
```

Hadoop in practice VI

Partitioner (Wikipedia category example)

```
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;
…
public class CustomPartitioner extends Partitioner
{
        public int getPartition(Object key, Object value, int numPartitions)
        {
                String s = ((Text)key).toString();
                String newKey = s.substring(0,s.lastIndexOf('#'));

                return newKey.hashCode() % numPartitions;
        }

}
```

Hadoop in practice VII

//create a directory with content on the hadoop distributed file system

```
$ hadoop dfs -copyFromLocal /Users/ch/gutenbergBooks gutenbergBooks
```

//deploy the created jar

```
$ hadoop jar wordcount.jar wordcount /user/ch/gutenbergBooks /user/ch/out
```

//have a look at the reducer's output

```
$ hadoop dfs -cat /user/claudiahauff/gutenbergBooks-output/part-r-00000
```

Hadoop's architecture

- NameNode
- SecondaryNamenode
- DataNode
- JobTracker
- TaskTracker

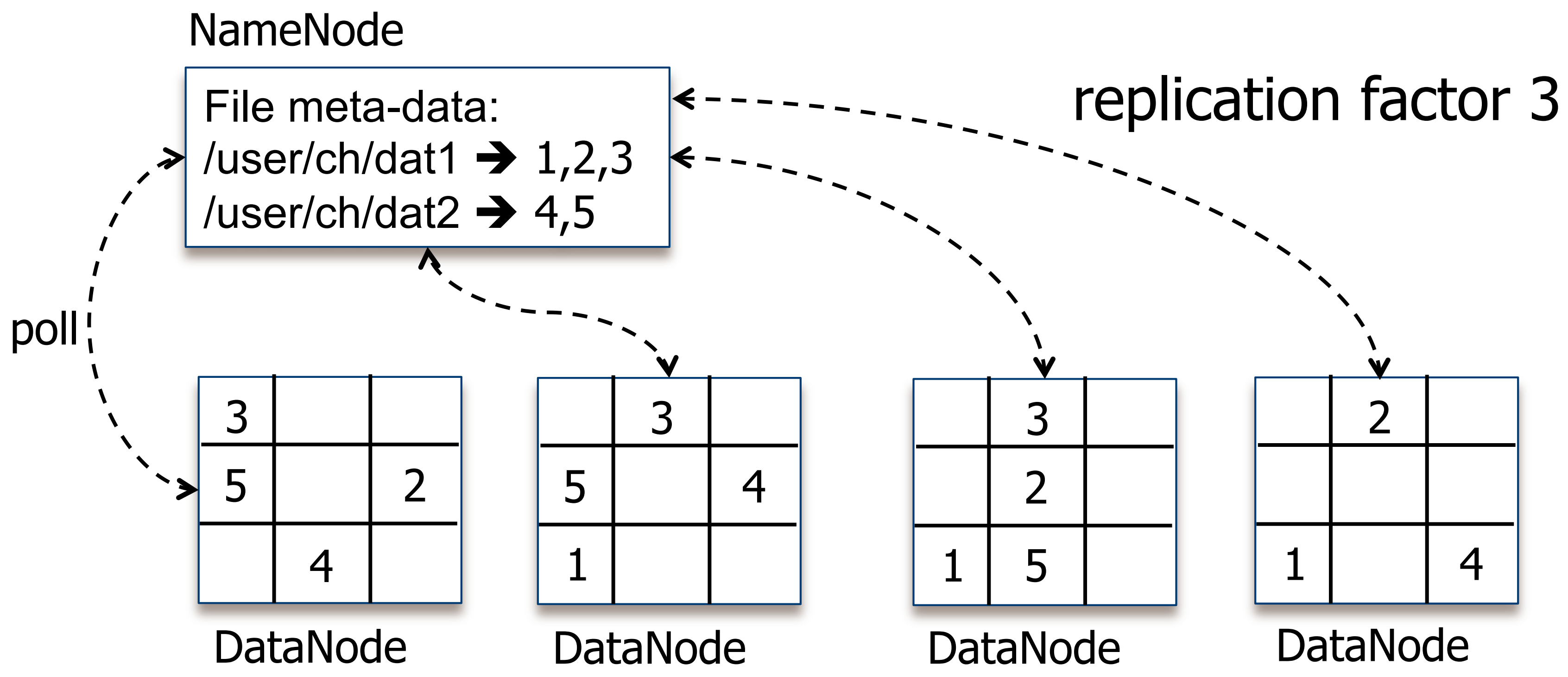
Hadoop's architecture

- NameNode
  - Master of HDFS, directs the slave DataNode daemons to perform low-level I/O tasks
  - Keeps track of file splitting into blocks (usually 64M), replication, which nodes store which blocks, etc.
  - Keeps track of the health of the distributed file system
  - Memory and I/O intensive task

- SecondaryNameNode: takes snapshots of the NameNode

- DataNode
  - Each slave machine hosts a DataNode daemon

NameNode and DataNodes

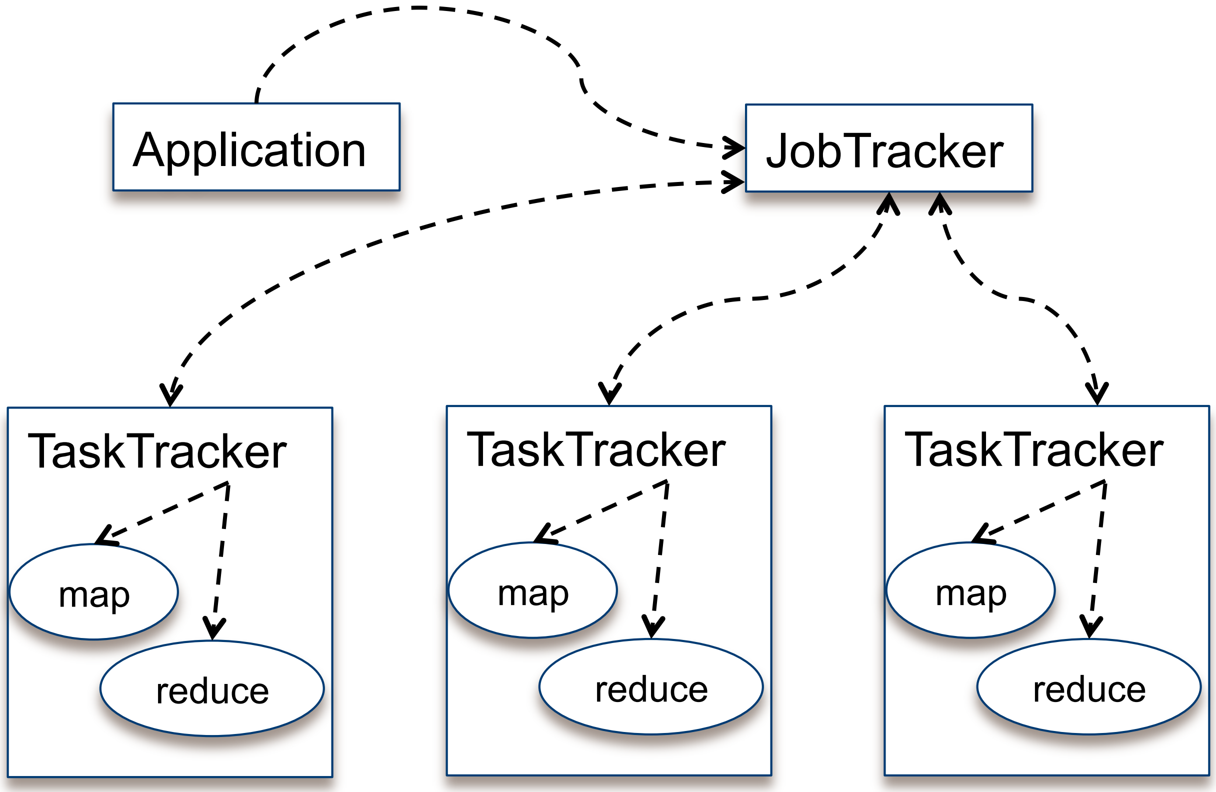


Adapted from *Hadoop In Action* by Chuck Lam, 2011, Figure 2.1 (page 23).

JobTracker and TaskTracker

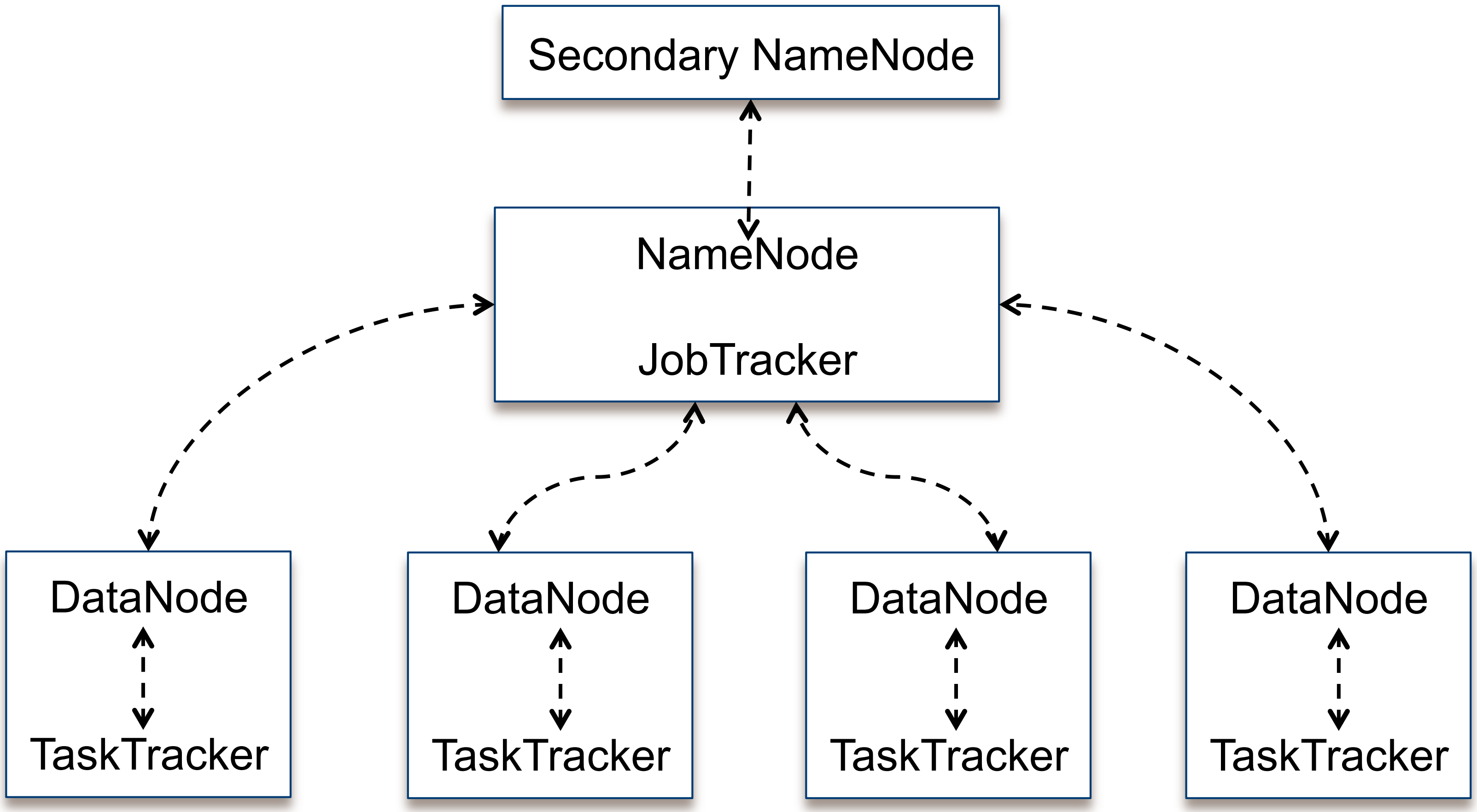
- JobTracker
  - One JobTracker per hadoop cluster
  - Middleman between your application and hadoop
  - Determines the execution plan for the application (files to process, assignment of nodes to tasks, task monitoring)
  - Takes care of task failures
- TaskTracker
  - One TaskTracker per slave node
  - Manages individual tasks
  - Keeps in touch with the JobTracker (heartbeat)

JobTracker and TaskTracker



Adapted from *Hadoop In Action* by Chuck Lam, 2011, Figure 2.2 (page 24).

Hadoop cluster topology



Adapted from *Hadoop In Action* by Chuck Lam, 2011, Figure 2.3 (page 25).

Hadoop related projects

Apache Pig

- Initially developed by Yahoo! research in 2006, incubation at Apache in 2007 [16]
- High-level platform for large-scale data set analysis
- Contains a compiler that produces MapReduce programs
    - Ease of use, quick prototyping is possible

- Relational data-flow language: Pig Latin [17]

```
Users                   = load 'users' as (name, age, ipaddr);
Clicks                  = load 'clicks' as (user, url, value);
ValuableClicks          = filter Clicks by value > 0;
UserClicks              = join Users by name, ValuableClicks by user;
```

Hadoop related projects

HBase

- Hadoop database for random, realtime read/write access to big data
- Similar to Google's Bigtable [18]

- ……

Assignment 0

If you search for answers on the Web, always include your Hadoop version! Code changes are very frequent!

- Install Hadoop
  - Many versions, newest one is 1.0.0 (December 2011)
  - Mine is 0.20.2 → if you think you will need a lot of practical help, please install this one (installation README on blackboard)
  - Hadoop can be installed from source/binaries or as a virtual machine (Cloudera, Yahoo!)

- Run the ubiquitous WordCount example provided by Hadoop
- Learn how to upload/download/view data on the HDFS

- Learn how to compile your own map/reduce code

- Read chapters 1 & 2 of *Data-Intensive Text Processing ….*
- Have a look at Amazon EC2

Sources

① Data-Intensive Text Processing with MapReduce (manuscript). Jimmy Lin and Chris Dyer. 2010.
② http://youtube-global.blogspot.com/2012/01/holy-nyans-60-hours-per-minute-and-4.html
③ http://newsroom.fb.com/content/default.aspx?NewsAreaId=22
④ http://blog.twitter.com/2011/03/numbers.html
⑤ http://public.web.cern.ch/public/en/LHC/Computing-en.html
⑥ Analytics over Large-Scale Multidimensional Data: The Big Data Revolution! A. Cuzzocrea, I.Y. song and K.C. Davis, 2011
⑦ http://www.nytimes.com/2011/03/06/weekinreview/06lohr.html
⑧ MapReduce: Simplified Data Processing on Large Clusters http://research.google.com/archive/mapreduce.htmll
⑨ http://www.archive.org/about/faqs.php
⑩ http://www.archive.org/web/petabox.php
11 http://www.cs.nott.ac.uk/~gmh/book.html
12 xxxx
13 http://wiki.apache.org/hadoop/PoweredBy
14 http://www.guardian.co.uk/technology/2011/mar/25/media-guardian-innovation-awards-apache-hadoop
15 Data-Intensive Text Processing with MapReduce, Jimmy Lin and Chris Dyer, 2010.
16 http://developer.yahoo.com/blogs/hadoop/posts/2008/10/pig_-_the_road_to_an_efficient_high-level_language_for_hadoop/
17 http://developer.yahoo.com/blogs/hadoop/posts/2010/01/comparing_pig_latin_and_sql_fo/
18 http://research.google.com/archive/bigtable.html