Big Data Processing, 2014/15

Lecture 4: MapReduce & Hadoop

Claudia Hauff (Web Information Systems)
ti2736b-ewi@tudelft.nl

Lab organisation

First lab session

- Two lab assistants

Robert Carosi

Kilian

- “Sign in” with them at each lab session

Software

- Virtual machine-based: **Cloudera CDH 4.7**, based on CentOS
- Saves us from a “manual” Hadoop installation (especially difficult on Windows) — but if you want to install Hadoop ‘by hand’ you can do that as well
- Ensures that everyone has the same setup

“As part of the boot process, the VM automatically launches Cloudera Manager and configures **HDFS, Hive, Hue, MapReduce, Oozie, ZooKeeper, Flume, HBase, Cloudera Impala, Cloudera Search, and YARN**.
Only the ZooKeeper, **HDFS**, **MapReduce**, Hive, and Hue services are started automatically.”

Cloudera

Hadoop runs in “**pseudo-distributed**” mode on a single machine (yours).
Hadoop: write once, run on one machine or a cluster of 20,000 machines.

Course content

- Introduction
- Data streams 1 & 2
- **The MapReduce paradigm**
- Looking behind the scenes of MapReduce: HDFS & Scheduling
- Algorithm design for MapReduce
- A high-level language for MapReduce: Pig 1 & 2
- MapReduce is not a database, but HBase nearly is
- Lets iterate a bit: Graph algorithms & Giraph
- How does all of this work together? ZooKeeper/Yarn

Learning objectives

- **Explain** the difference between MapReduce and Hadoop
- **Explain** the difference between the MapReduce paradigm and related approaches (RDMBS, HPC)
- **Transform** simple problem statements into map/reduce functions
- **Employ** Hadoop's combiner and partitioner functionality effectively

MapReduce & Hadoop

"MapReduce is a programming model for expressing **distributed** computations on **massive amounts of data** and an execution framework for large-scale data processing on clusters of **commodity servers**."

-Jimmy Lin

Hadoop is an open-source implementation of the MapReduce framework.

MapReduce characteristics

- **Batch** processing
- **No limits** on #passes over the data or time
- **No memory constraints**

History of MapReduce

- Developed by researchers at **Google** around **2003**
  - Built on principles in parallel and distributed processing
- **Seminal papers:**
  - *The Google file system* by Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung (2003)
  - *MapReduce: Simplified Data Processing on Large Clusters.* by Jeffrey Dean and Sanjay Ghemawat (2004)
- "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" (2004)

Provides a clear separation between <u>what</u> to compute and <u>how</u> to compute it on a cluster.

History of Hadoop

Apache project Web crawler

- Created by Doug Cutting as solution to Nutch's scaling problems, inspired by Google's GFS/MapReduce papers
- 2004: Nutch Distributed Filesystem written (based on GFS)
- Middle 2005: all important parts of Nutch ported to MapReduce and NDFS
- February 2006: code moved into an independent subproject of Lucene called Hadoop
- In early 2006 Doug Cutting joined Yahoo! which contributed resources and manpower

Apache Software Foundation

- January 2008: Hadoop became a top-level project at Apache

Hadoop versioning [warning]

Hadoop Releases

- Download
- News
  - 12 September, 2014: Relase 2.5.1 available
  - 11 August, 2014: Release 2.5.0 available
  - 30 June, 2014: Release 2.4.1 available
  - 27 June, 2014: Release 0.23.11 available
  - 07 April, 2014: Release 2.4.0 available
  - 20 February, 2014: Release 2.3.0 available
  - 11 December, 2013: Release 0.23.10 available
  - 15 October, 2013: Release 2.2.0 available
  - 23 September, 2013: Release 2.1.1-beta available
  - 25 August, 2013: Release 2.1.0-beta available
  - 23 August, 2013: Release 2.0.6-alpha available
  - 1 Aug, 2013: Release 1.2.1 (stable) available
  - 8 July, 2013: Release 0.23.9 available
  - 6 June, 2013: Release 2.0.5-alpha available
  - 5 June, 2013: Release 0.23.8 available
  - 13 May, 2013: Release 1.2.0 available
  - 25 April, 2013: Release 2.0.4-alpha available
  - 18 April, 2013: Release 0.23.7 available
  - 15 February, 2013: Release 1.1.2 available
  - 14 February, 2013: Release 2.0.3-alpha available
  - 7 February, 2013: Release 0.23.6 available

http://hadoop.apache.org/releases.html

Hadoop versioning [warning]

Hadoop Releases

- Download
- News
  - 12 September, 2014: Relase 2.5.1 available
  - 11 August, 2014: Release 2.5.0 available
  - 30 June, 2014: Release 2.4.1 available

- Frequent API changes
- When searching for help online, include the Hadoop version you are working with
- What is deprecated in one version might become de-deprecated in the next one
- Hadoop is still under heavy development

  - 13 May, 2013: Release 1.2.0 available
  - 25 April, 2013: Release 2.0.4-alpha available
  - 18 April, 2013: Release 0.23.7 available
  - 15 February, 2013: Release 1.1.2 available
  - 14 February, 2013: Release 2.0.3-alpha available
  - 7 February, 2013: Release 0.23.6 available

http://hadoop.apache.org/releases.html

Hadoop versioning [warning]

Feature	1.x	0.22	2.x
Secure authentication	Yes	No	Yes
Old configuration names	Yes	Deprecated	Deprecated
New configuration names	No	Yes	Yes
Old MapReduce API	Yes	Yes	Yes
New MapReduce API	Yes (with somemissing libraries)	Yes	Yes
MapReduce 1 runtime (Classic)	Yes	Yes	No
MapReduce 2 runtime (YARN)	No	No	Yes
HDFS federation	No	No	Yes
HDFS high-availability	No	No	Yes

http://hadoop.apache.org/releases.html

Ideas behind MapReduce

- **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 programs/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

- **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

- **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

- **Hide system-level details** from the application developer
  - Frees the developer to thin
    only (no need to worry abo
  - MapReduce takes care of t

System-level details:
- data partitioning
- scheduling, load balancing
- fault tolerance
- inter-machine communication

- **Seamless scalability**
  - Data scalability (given
    algorithm runs twice as
  - Resource scalability (g
    the ideal algorithm runs in half the time)

"… 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)

MapReduce vs. RDBMS

Trend: disk seek times are improving more slowly than the disk transfer rate (i.e. it is faster to stream all data than to make seeks to the data)

	RDBMS	MapReduce
Data size	Gigabytes (mostly)	Petabytes
Access	interactive & batch	batch
Updates	many reads & writes	write once, read a lot (**the entire data**)
Structure	static schema	data interpreted at processing time
Redundancy	low (normalized data)	high (unnormalized data)
Scaling	nonlinear	linear

MapReduce vs. RDBMS

```
fcrawler.looksmart.com - - [26/Apr/2000:00:00:12 -0400] "GET /contacts.html HTTP/1.0" 200
fcrawler.looksmart.com - - [26/Apr/2000:00:17:19 -0400] "GET /news/news.html HTTP/1.0" 200

123.123.123.123 - - [26/Apr/2000:00:23:48 -0400] "GET /pics/wpaper.gif HTTP/1.0"     200
123.123.123.123 - - [26/Apr/2000:00:23:47 -0400] "GET /asctortf/ HTTP/1.0"           200
123.123.123.123 - - [26/Apr/2000:00:23:48 -0400] "GET /pics/5star2000.gif HTTP/1.0"  200
123.123.123.123 - - [26/Apr/2000:00:23:50 -0400] "GET /pics/5star.gif HTTP/1.0"      200
```

sk
eks to

	RDBMS	**MapReduce**
Data size	Gigabytes (mostly)	Petabytes
Access	interactive & batch	batch
Updates	many reads & writes	write once, read a lot (**the entire data**)
Structure	static schema	data interpreted at proce
Redundancy	low (normalized data)	high
Scaling	nonlinear	linea

Blurring the lines: MapReduce moves into the direction of RDBMs (Hive, Pig) and RDBMs move into the direction of MapReduce (NoSQL).

MapReduce vs. High Performance Computing (HPC)

- HPC works well for **computationally intensive problems** with low to medium data volumes
  - Bottleneck: network bandwidth, leading to idle compute nodes
- MapReduce: **moves the computation to the data**, conserving network bandwidth

- HPC gives a lot of control to the programmer, requires handling of low-level aspects (data flow, failures, etc.)
- MapReduce requires programmer to only provide map/reduce code, takes care of low-level details
  - **But**: everything needs to be pressed into the map/reduce framework

MapReduce paradigm

- **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**:
  - How to transform a problem into sub-problems?
  - How to assign workers and synchronise the intermediate results?
  - How do the workers get the required data?
  - How to handle failures in the cluster?

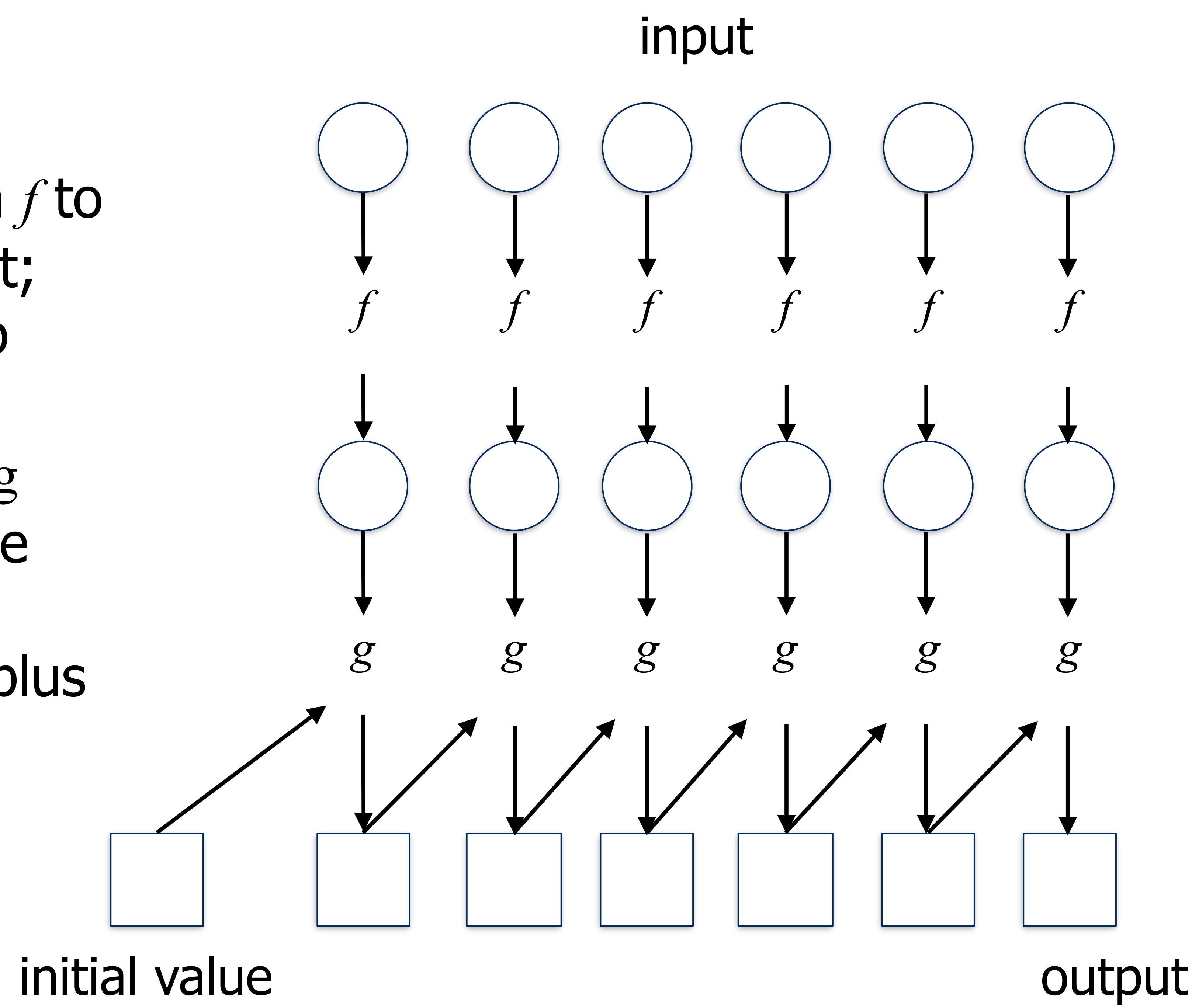
MapReduce in brief

- Define the `map()` function
- Define the input to `map()` as key/value pair
- Define the output of `map()` as key/value pair
- Define the `reduce()` function
- Define the input to `reduce()` as key/value pair
- Define the output of `reduce()` as key/value pair

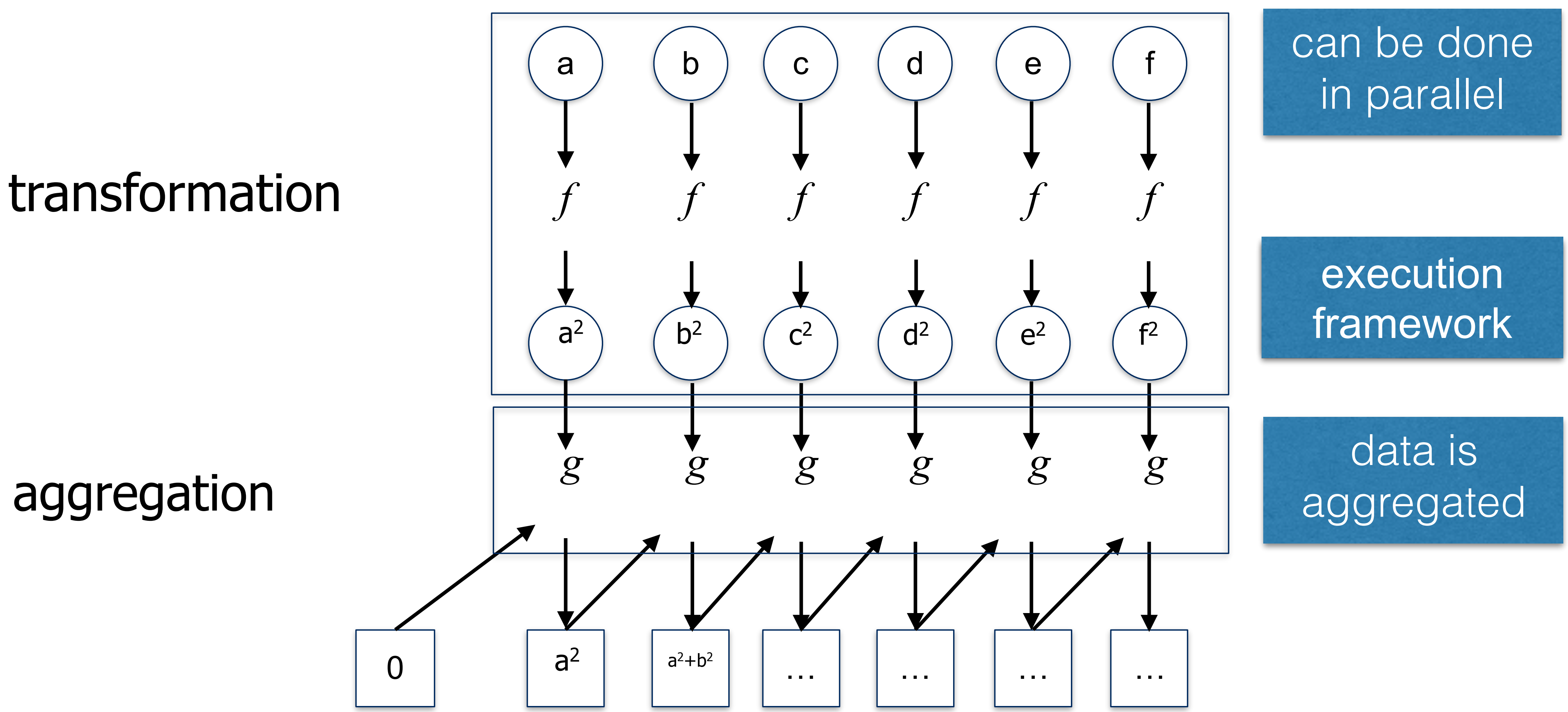
Map & fold: two higher order functions

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

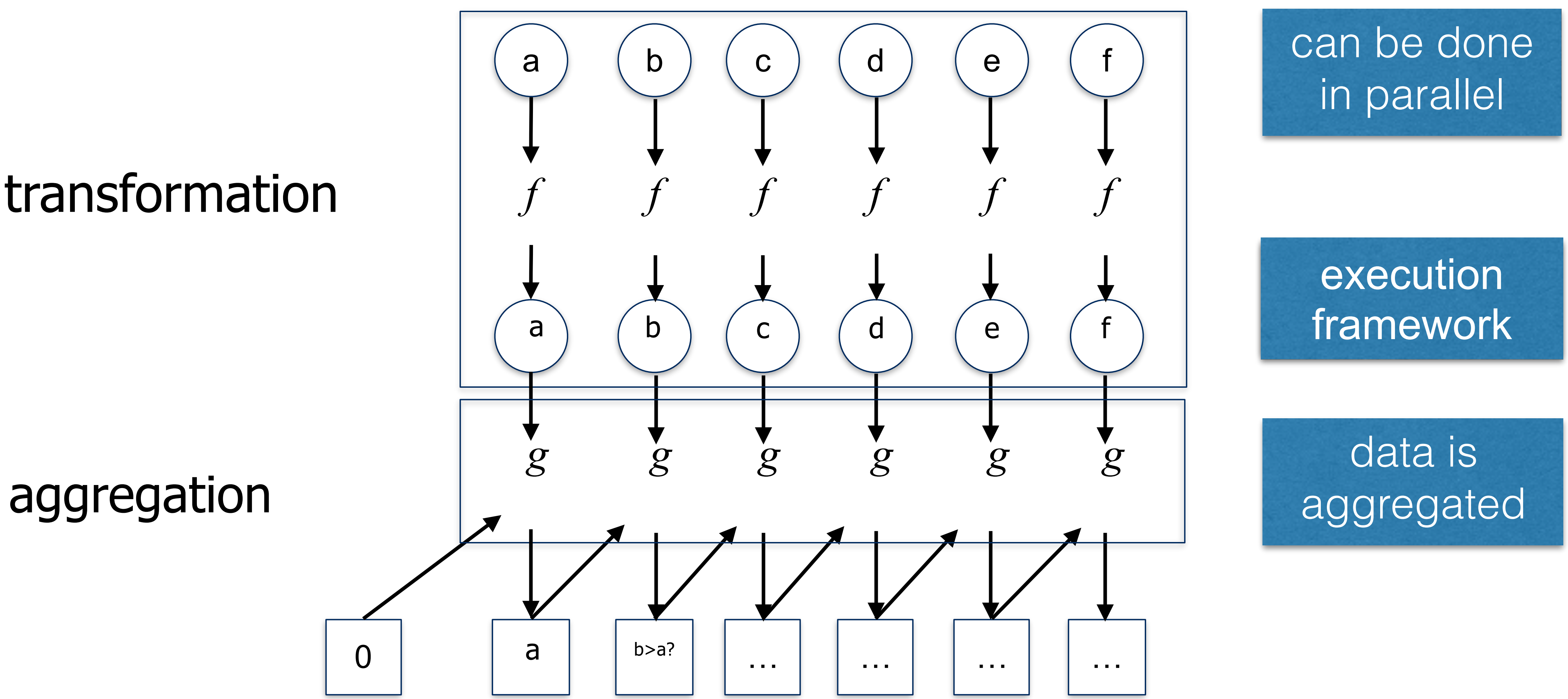


Map & fold example: sum of squares



$$0 + a^2 + b^2 + c^2 + d^2 + e^2 + f^2$$

Map & fold example: maximum (pos. numbers)



$$max(a, b, c, d, e, f)$$

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

$$\textit{map:}\ (k_i, v_i) \rightarrow [(k_j, v_j)]$$
$$\textit{map:}\ (k_i, v_i) \rightarrow [\boxed{(k_j, v_x),}\, (k_m, v_y), \boxed{(k_j, v_n),} \ldots]$$

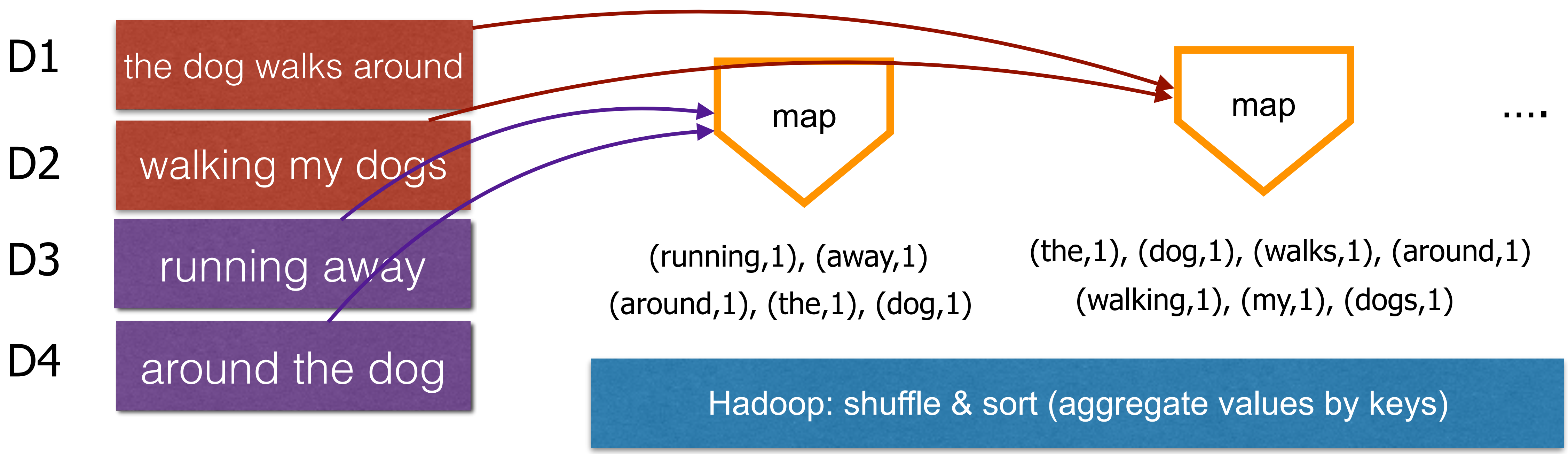
- Apply a reduce operation to all values that share the same key

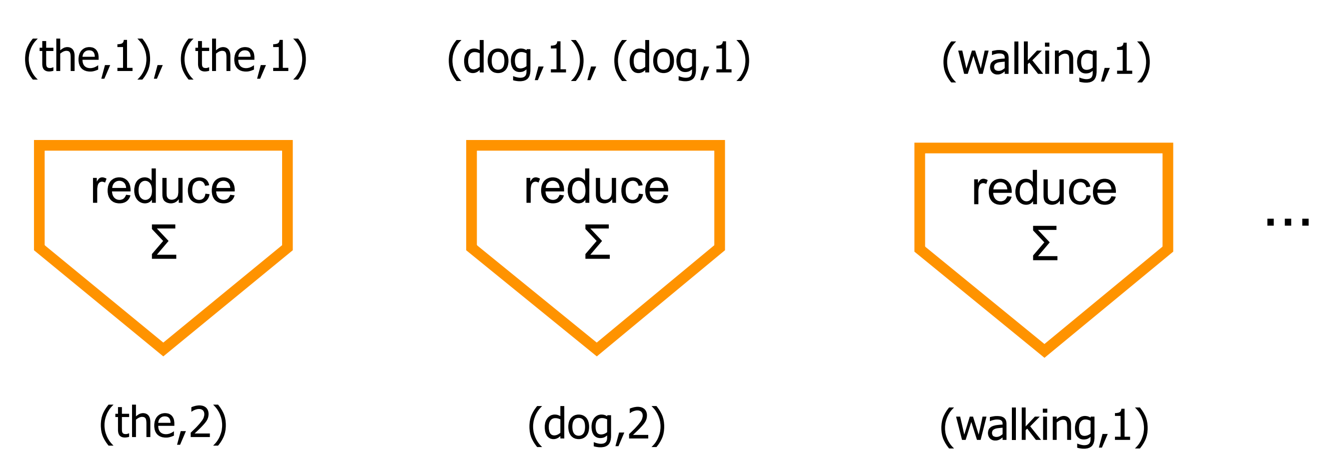
$$\textit{reduce:}\ (k_j, [v_x, v_n]) \rightarrow [(k_h, v_a), (k_h, v_b), (k_l, v_a)]$$

Map & reduce: developer focus

- **Divide** the data into appropriate key/value pairs
- Make sure that the **memory footprint** of the map/reduce functions is limited
- Think about the **number of key/value pairs** to be **sent over the network**

Example: word count





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

Example: word count

Problem: compute the frequency of every term in the corpus.

docid

document content

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

intermediate key/value pairs

term

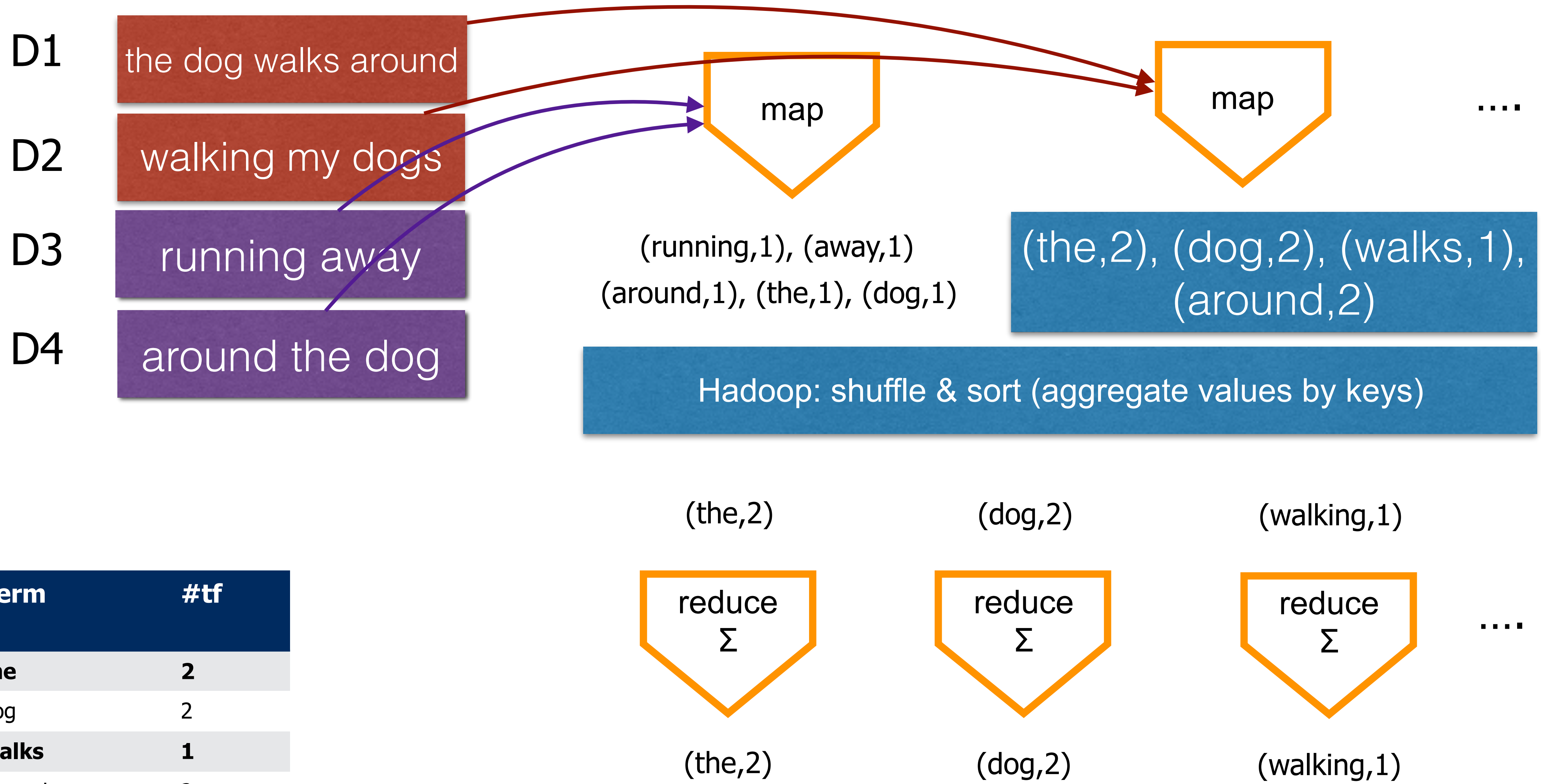
```
reduce(String key, Iterator values):
    int res = 0;
    foreach int v in values:
        res += v;
    Emit(key, res)
```

all values with the same key

count of 'key' in the corpus

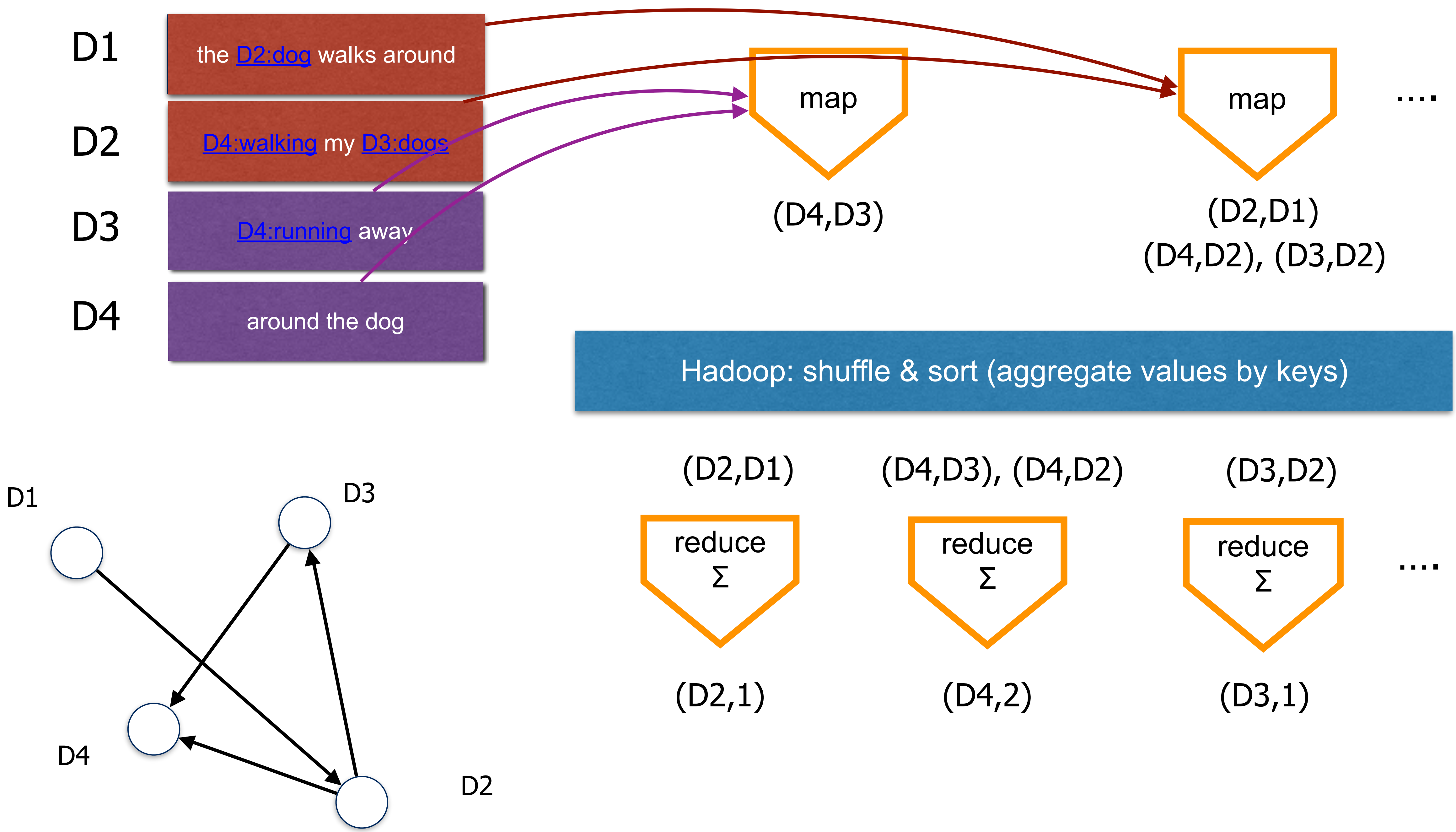
Important: the iterator in the reducer can only be used once!
There is no looking back!
There is no restart option.

Example: word count



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

Example: inlink count



Example: inlink count

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

source

document content

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

target

intermediate key/value pairs

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

all sources pointing to target

#pages linking to 'key'

Important: the iterator in the reducer can only be used once!
There is no looking back!
There is no restart option.

Example: list documents and their categories occurring 2+ times

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]

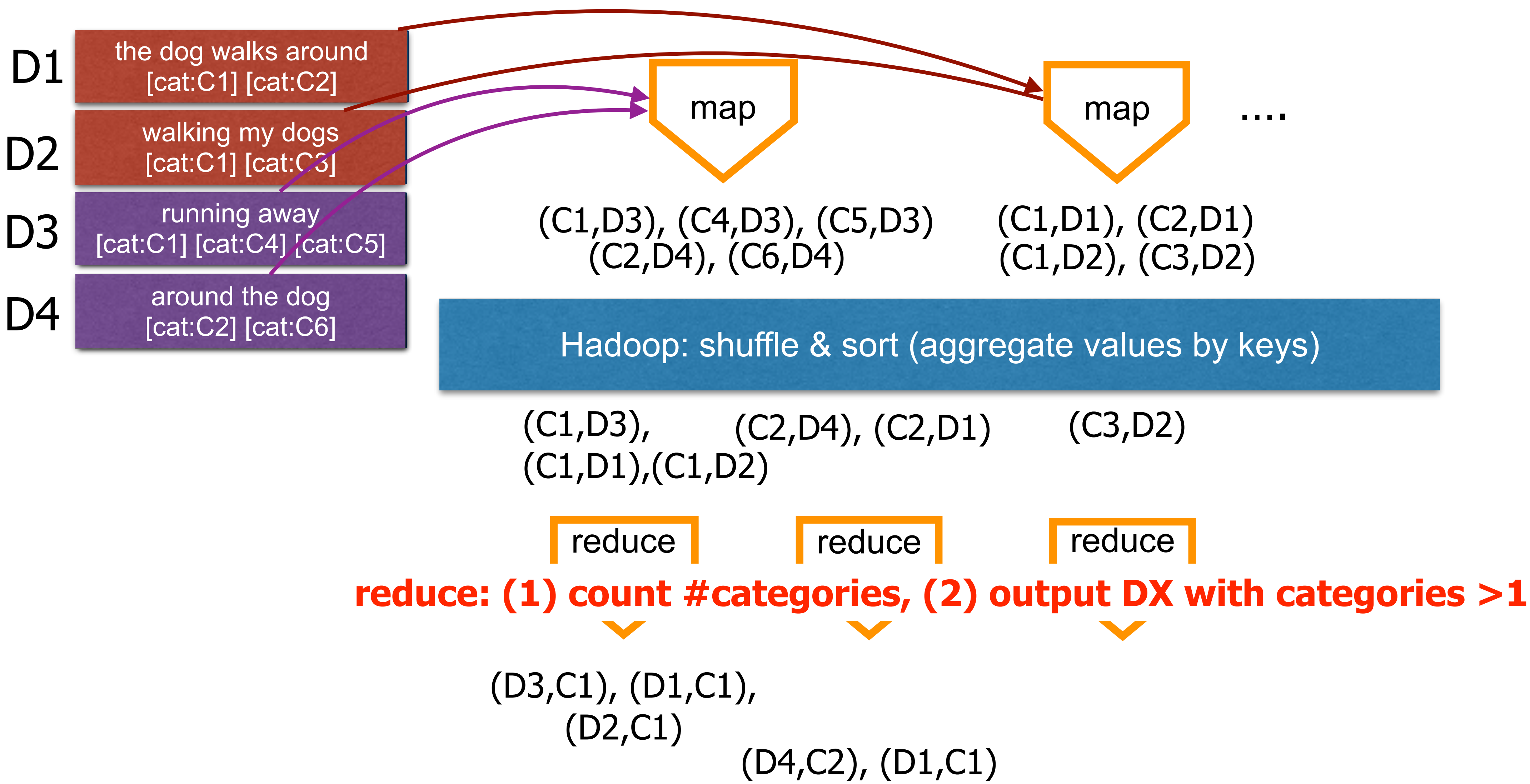
D4 — around the dog [cat:C2] [cat:C6]

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

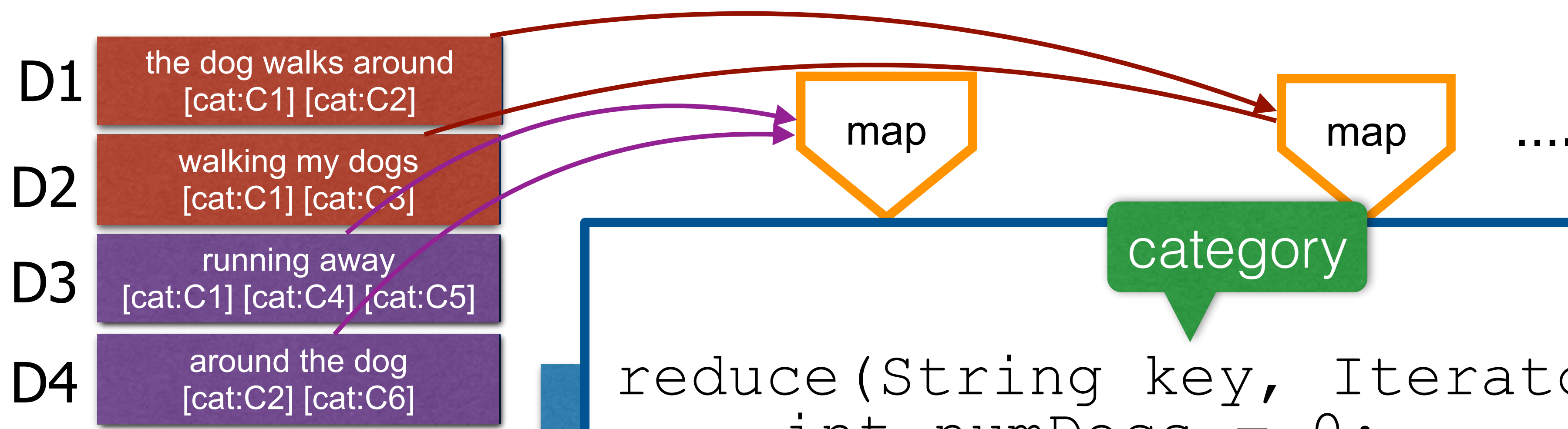
Categories: 1890 births | 1974 deaths | American electrical engineers | Computer pioneers | Futurologists | Harvard University alumni | IEEE Edison Medal recipients | Internet pioneers | Massachusetts Institute of Technology alumni | Massachusetts Institute of Technology faculty | Manhattan Project people | 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 | Peopl | Peopl | Tufts

```
…
{{DEFAULTSORT:Bush, Vannevar}}
[[Category:1890 births]]
[[Category:1974 deaths]]
[[Category:American electrical engineers]]
[[Category:Computer pioneers]]
[[Category:Futurologists]]
[[Category:Harvard University alumni]]
[[Category:IEEE Edison Medal recipients]]
[[Category:Internet pioneers]]
…
```

Example: list documents and their categories occurring 2+ times



Example: list documents and their categories occurring 2+ times



category

documents

reduce

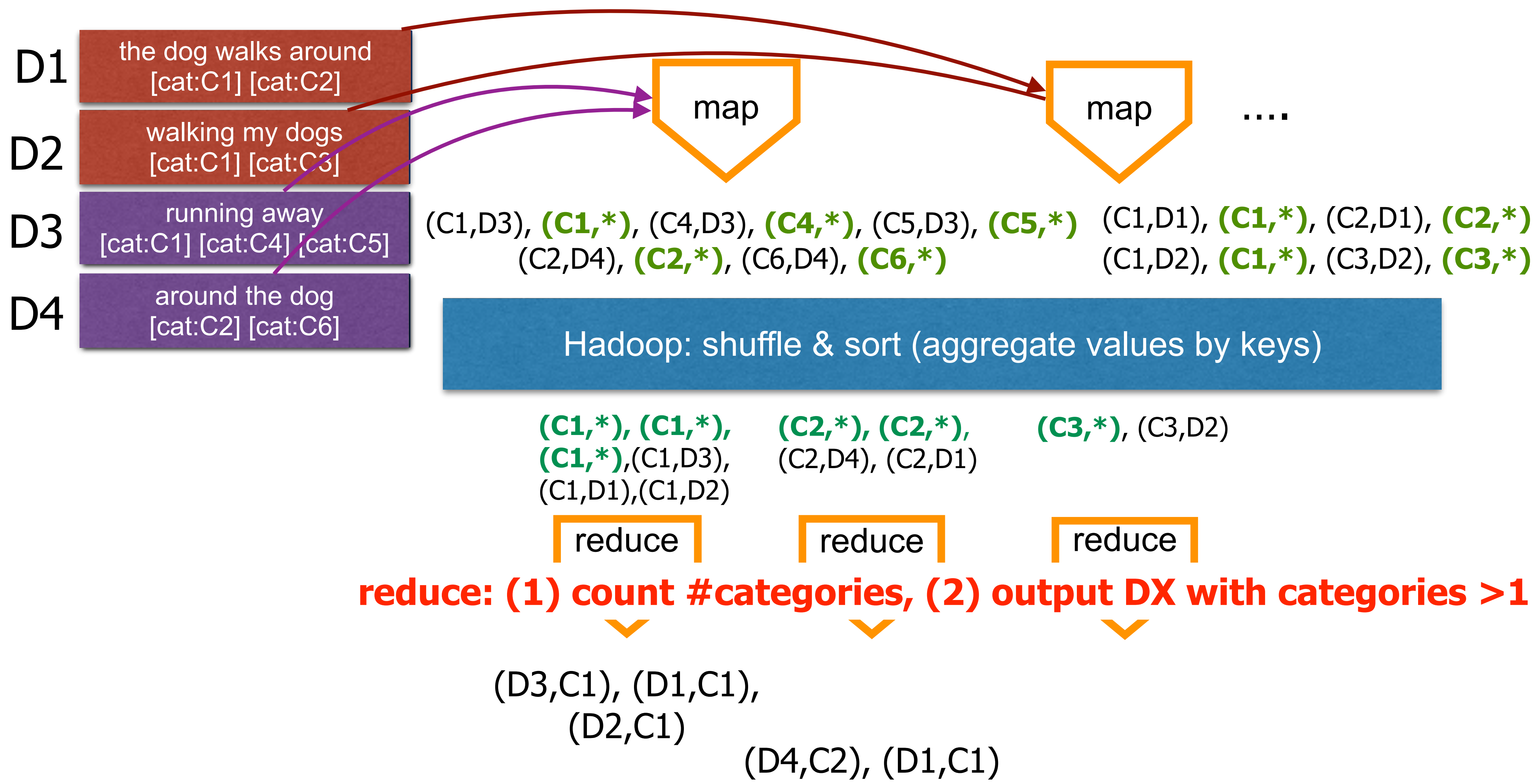
```
reduce(String key, Iterator values):
     int numDocs = 0;
     foreach v in values:
          numDocs += v;

     if(numDocs<2)
          return;

     foreach v in values:
          Emit(key, res)
```

No looking back!

Example: list documents and their categories occurring 2+ times



Example: list documents and their categories occurring 2+ times

docid

document content

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

we can emit more than 1 key/value pair

category

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

*'s and docids

Assumption: the values are sorted in a particular order (* first).

document's category with min freq. 2

Example: list documents and their categories occurring 2+ times

docid

document content

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

we can emit more than 1 key/value pair

category

```
reduce(String key, Iterator values):
          List list = copyFromIterator(values)

          int numDocs = 0;
          foreach l in list:
               if(l==*)
                    numDocs ++;
          if(numDocs<2)
               return;
          foreach l in list:
               Emit(d,key)
```

We assume no particular sorting of values.

What if there are 100GB of values for key? Do they fit into memory?

Zipf's law

Term frequencies: the Count of Monte Christo

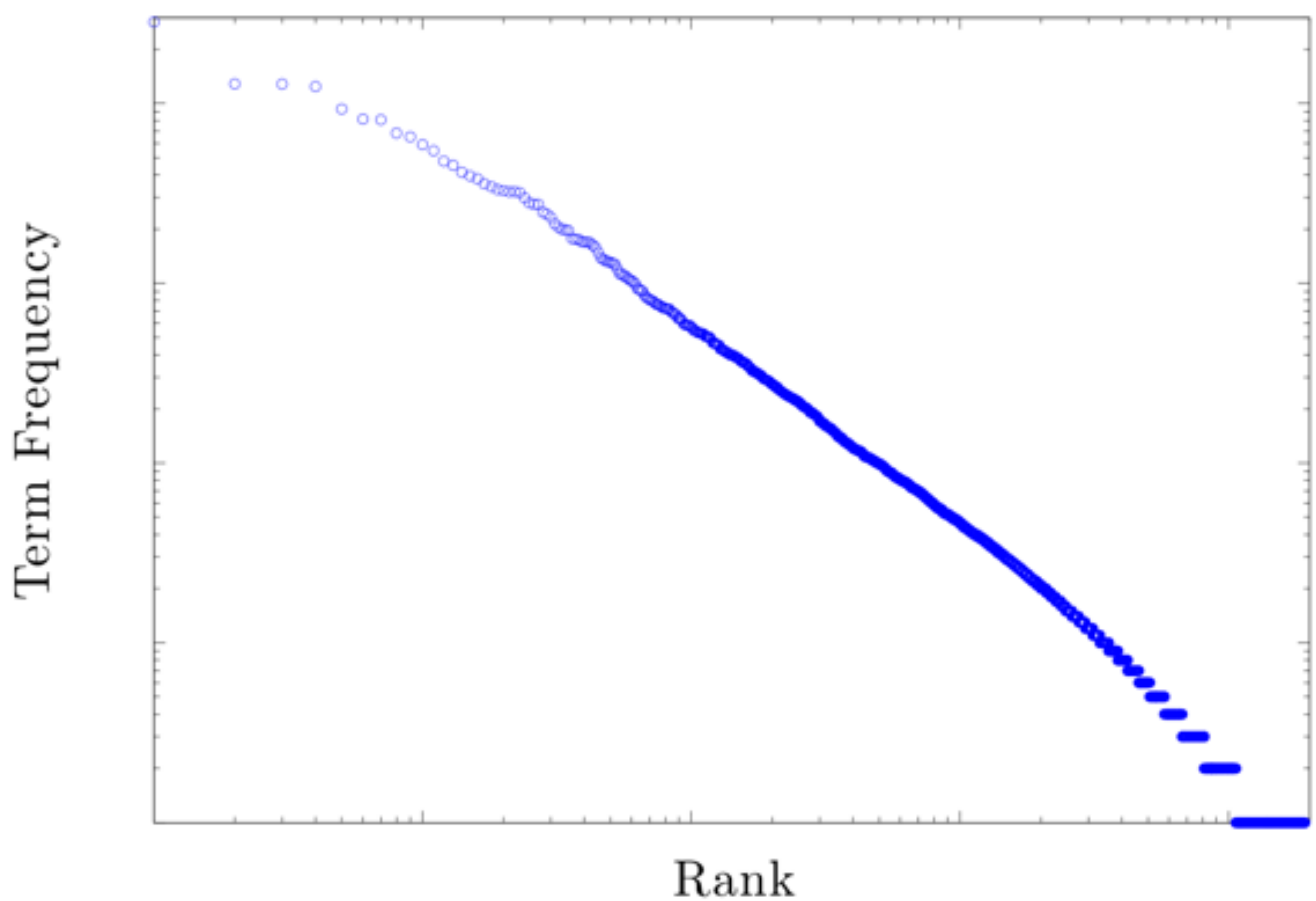
	Term	#tf
1.	**the**	**28388**
2.	to	12841
3.	**of**	**12834**
4.	and	12447
5.	**a**	**9328**
6.	i	8174
7.	**you**	**8128**

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

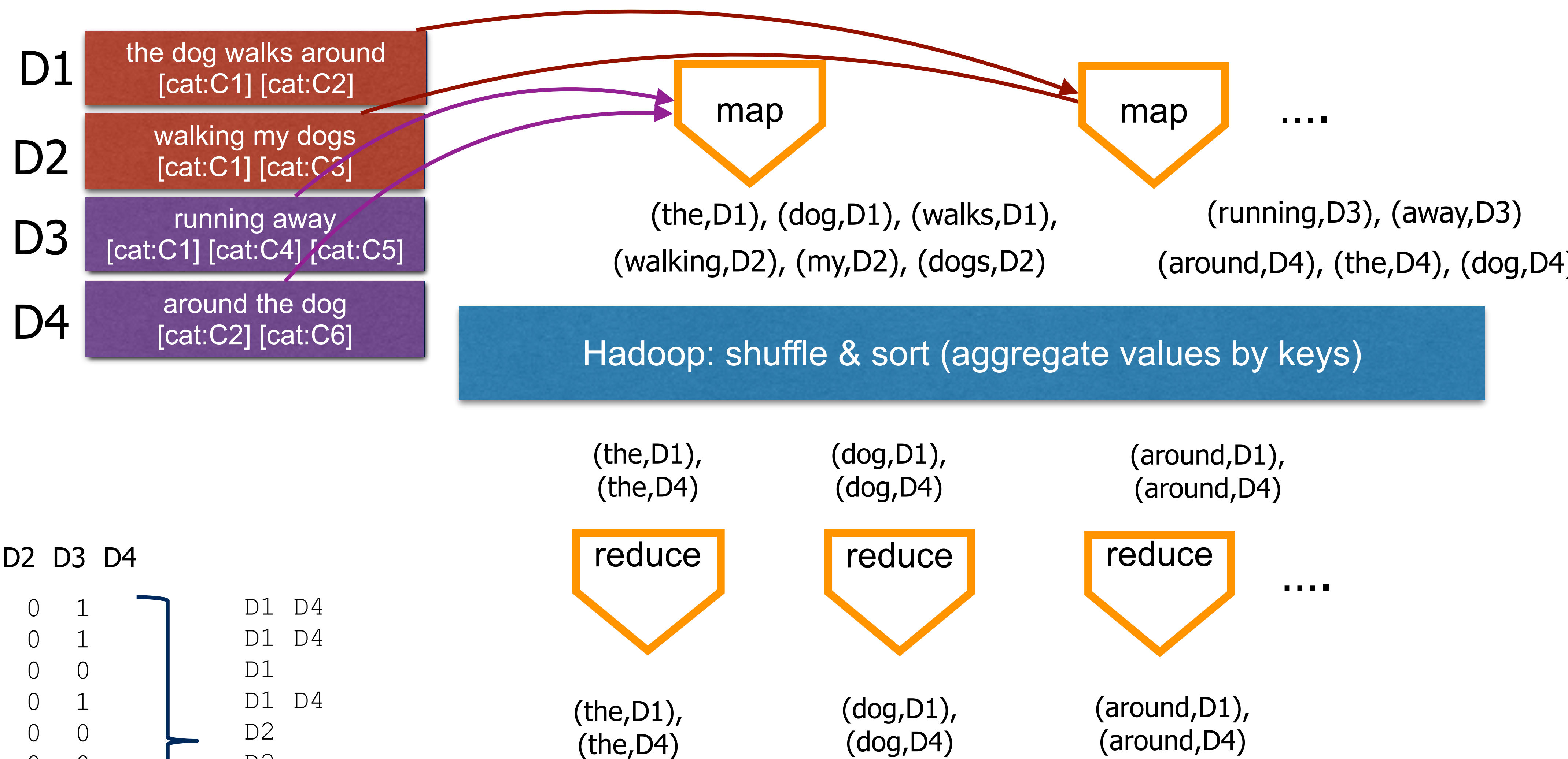
	Term	#tf
19001.	**calaseraigne**	**1**
19002.	jackals	1
19003.	**sorti**	**1**
19004.	meyes	1
19005.	**bets**	**1**
19006.	pistolshots	1
19007.	**francsah**	**1**

Zipf's law

Term frequencies: the Count of Monte Christo



Example: a simple inverted index



	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

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

docid

document 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)
```

Not much to be done in the reducer. (IdentityReducer)

all documents with term 'key'

Example: parsing

But: you cannot create a Hadoop job without a Mapper.

There is more: the 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 key-to-reducer assignment:
  `hash(key) modulus num_reducers`

There is more: the combiner

- Combiner: local aggregation of key/value pairs after map() and before the shuffle & sort phase (occurs on the same machine as map())

- Also called "mini-reducer"

Sometimes the reducer code can be used.

- Instead of emitting 100 times (the,1), the combiner emits (the,100)

- Can lead to great speed-ups

- Needs to be employed with care

There is more: the combiner

Setup: a mapper which outputs (term,termFreqInDoc) and a combiner which is simply a copy of the reducer.

Task 1: total term frequency of a term in the corpus

without combiner

(the,2), (the,2), (the,1)

reduce

(the,**5**)

with combiner

(the,2), (the,3)

reduce

(the,**5**)

correct!

Task 2: average term frequency of a term in the corpus

without combiner

(the,2), (the,2), (the,1)

reduce

(the,(2+2+1)/3=**1.66**)

with combiner

(the,2), (the,1.5) ← (2+1)/2=1.5

reduce

(the,(2+1.5)/2=**1.75**)

wrong!

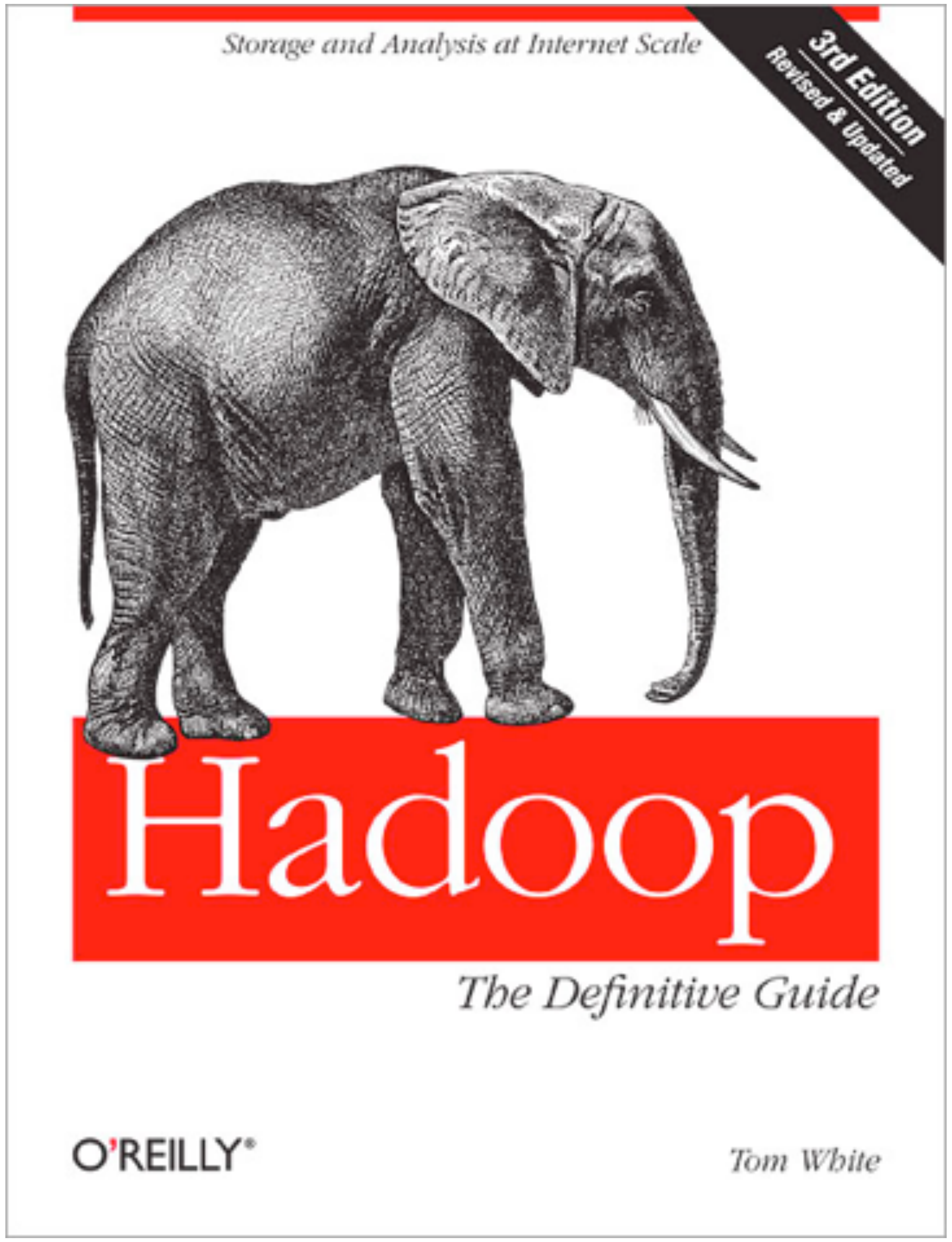
There is more: the combiner

- 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 (may not run at all! Hadoop's decision)
- Emitted key/value pairs **must be the same** as those emitted by the mapper
- Most often, combiner code != reducer code
    - Exception: Associative & commutative reduce operations

Summary

- MapReduce vs. Hadoop
- MapReduce vs. RDBMS/HPC
- Problem transformation into MapReduce programs
- Combiner & partitioner

Recommended reading



Chapter 1, 2 and 3.

A warning: coding takes time. More time than usual.
MapReduce is not difficult to understand, but different templates, different advice on different sites (of widely different quality).
Small errors are disastrous.

THE END