Big Data Computing

Master's Degree in Computer Science

2019-2020

Gabriele Tolomei

Department of Computer Science

Sapienza Università di Roma

tolomei@di.uniroma1.it

SAPIENZA
UNIVERSITÀ DI ROMA

Recap from Last Lecture

- Large-scale data analysis poses new challenges on traditional single-node architecture
  - Cluster computing architecture (scaling out)
- Need for novel frameworks supporting clustered architectures:
  - Reliability
  - Network communication
  - Distributed programming model

MapReduce

- A **programming model** (and an associated implementation) for processing big data sets with parallel, distributed algorithms on a cluster

MapReduce

- A **programming model** (and an associated implementation) for processing big data sets with parallel, distributed algorithms on a cluster
- It addresses the **3** main **challenges** of cluster architecture described
  - Stores data redundantly on multiple nodes to ensure data/computation availability
  - Moves computation close to data to minimize network data transfers
  - Provides a simple computational model to hide all the complexities of the distributed environment

MapReduce: Distributed File System

- Redundant storage infrastructure

MapReduce: Distributed File System

- Redundant storage infrastructure
- Provides global file namespace and availability across nodes in a cluster

MapReduce: Distributed File System

- Redundant storage infrastructure
- Provides global file namespace and availability across nodes in a cluster
- Well-known implementations:
  - Google GFS
  - Hadoop HDFS

MapReduce: Distributed File System

- Redundant storage infrastructure
- Provides global file namespace and availability across nodes in a cluster
- Well-known implementations:
  - Google GFS
  - Hadoop HDFS
- Usage pattern:
  - Large files (100s GB ÷ 10s TB)
  - Many "read" operations vs. few "updates" (append)

MapReduce: Distributed File System

- **3** main **components**:
  - Chunk Servers
  - Master Nodes
  - Client API

MapReduce: Distributed File System

- 3 main components:
  - **Chunk Servers**
  - Master Nodes
  - Client API

Distributed File System: Chunk Servers

- Large data files are split into contiguous "**chunks**" of fixed size
  - e.g., 16÷64 MB

Distributed File System: Chunk Servers

- Large data files are split into contiguous "**chunks**" of fixed size
  - e.g., 16÷64 MB
- Each chunk is **replicated** across multiple nodes (chunk servers)
  - 2 or 3 replicas per chunk
  - Each replica on a different node
  - At least, one replica on a different rack

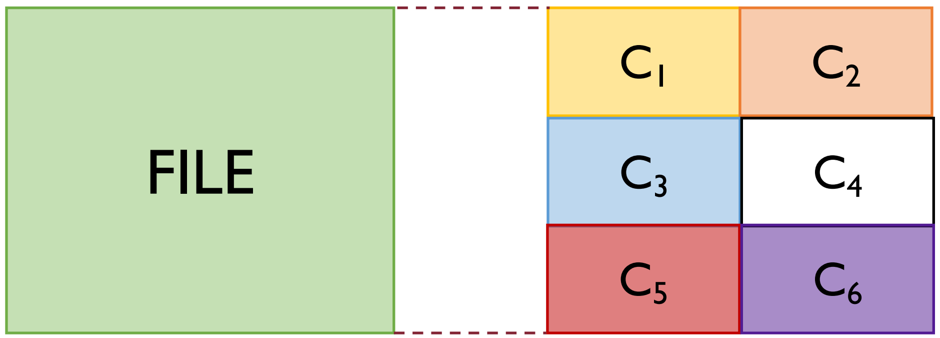
Distributed File System: Chunk Servers

- Large data files are split into contiguous "**chunks**" of fixed size
  - e.g., 16÷64 MB
- Each chunk is **replicated** across multiple nodes (chunk servers)
  - 2 or 3 replicas per chunk
  - Each replica on a different node
  - At least, one replica on a different rack
- Chunk servers act also as **computational servers**
  - move computation to data

Distributed File System: Chunk Servers

FILE

Distributed File System: Chunk Servers



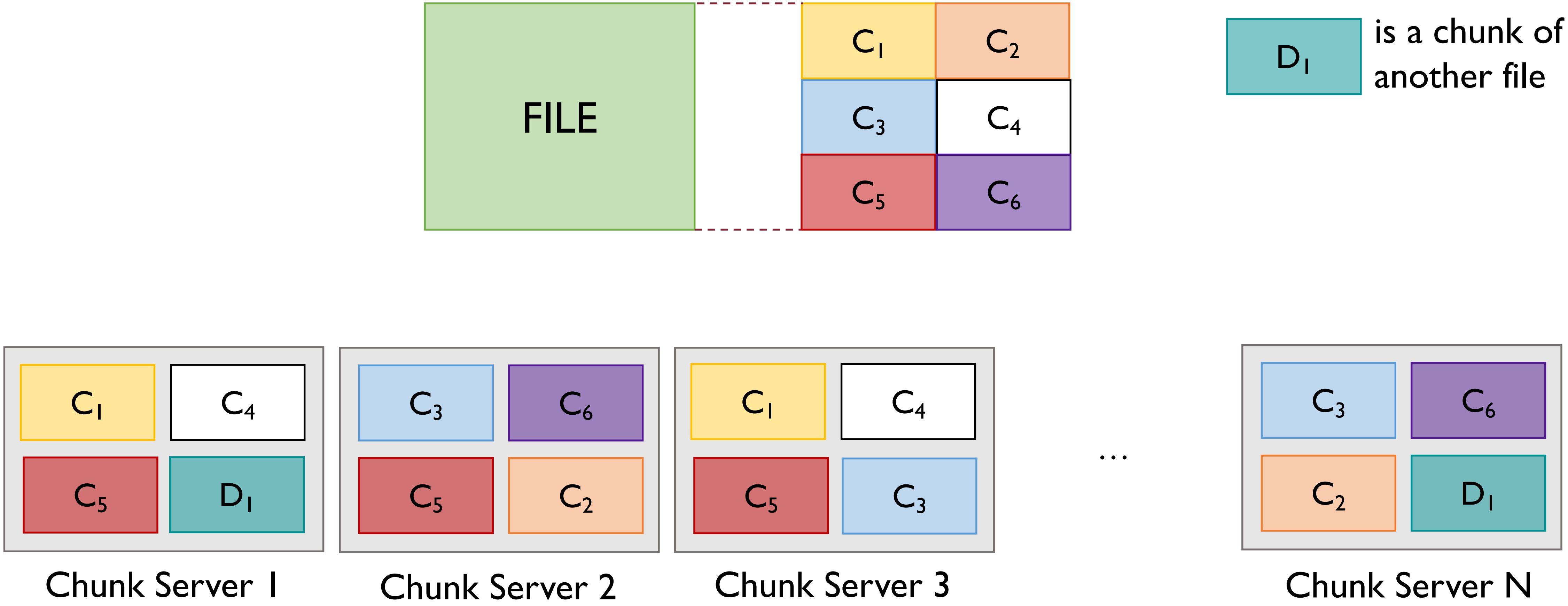
FILE

C_1 C_2

C_3 C_4

C_5 C_6

Distributed File System: Chunk Servers



MapReduce: Distributed File System

- 3 main components:
  - Chunk Servers
  - **Master Nodes**
  - Client API

Distributed File System: Master Node

- Stores **metadata** about files in the distributed filesystem
  - How many chunks each file is split into
  - Where each of those chunks are located

Distributed File System: Master Node

- Stores **metadata** about files in the distributed filesystem
  - How many chunks each file is split into
  - Where each of those chunks are located
- Possibly **replicated** to avoid single-point of failure

MapReduce: Distributed File System

- 3 main components:
  - Chunk Servers
  - Master Nodes
  - **Client API**

Distributed File System: Client API

- Allows clients to **access data** stored on chunk servers

Distributed File System: Client API

- Allows clients to **access data** stored on chunk servers
- Client asks the Master Node through the API where a particular chunk is located

Distributed File System: Client API

- Allows clients to **access data** stored on chunk servers
- Client asks the Master Node through the API where a particular chunk is located
- The Master Node replies with the information needed

Distributed File System: Client API

- Allows clients to **access data** stored on chunk servers
- Client asks the Master Node through the API where a particular chunk is located
- The Master Node replies with the information needed
- Afterwards, any communication between the client and the chunk server storing the data happens directly (i.e., without the Master Node)

MapReduce: Programming Model

- MapReduce is a **style of programming** designed for:
  - Easy parallel programming
  - Invisible management of hardware and software failures
  - Easy management of very-large-scale data

MapReduce: Programming Model

- MapReduce is a **style of programming** designed for:
  - Easy parallel programming
  - Invisible management of hardware and software failures
  - Easy management of very-large-scale data
- It has **several implementations**, including
  - Hadoop, Spark (used in this class), Flink, and the original Google implementation just called "MapReduce"

MapReduce: Intuition through an Example

- Suppose you are given a very large text document (e.g., 10s of TB)
  - The text document clearly does not fit into main memory!

MapReduce: Intuition through an Example

- Suppose you are given a very large text document (e.g., 10s of TB)
  - The text document clearly does not fit into main memory!
- **Word Counting Task**: compute how many times each individual word appears in the document

MapReduce: Intuition through an Example

- Suppose you are given a very large text document (e.g., 10s of TB)
  - The text document clearly does not fit into main memory!
- **Word Counting Task**: compute how many times each individual word appears in the document
- Possible applications:
  - Analysis of web/query logs
  - Statistical language modeling

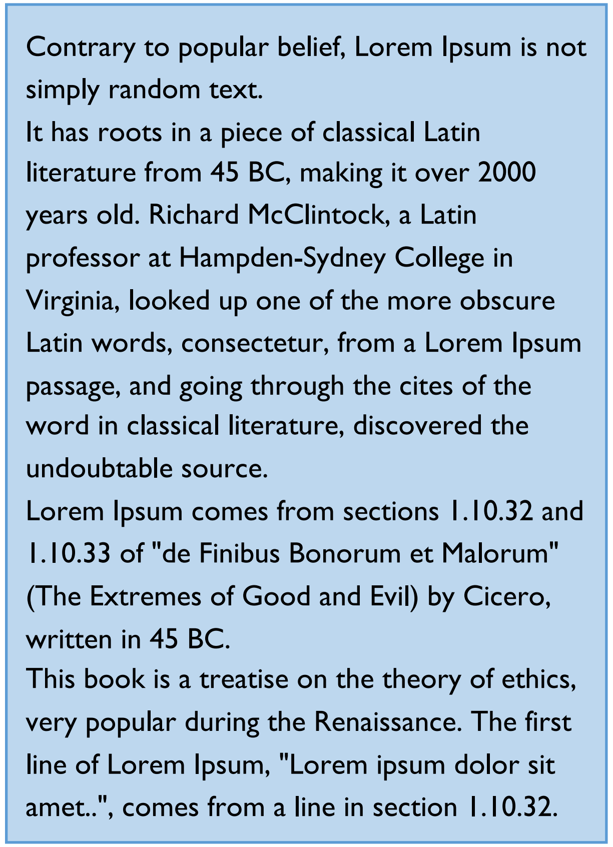
MapReduce: Intuition through an Example

- The result of the task will be a list of (word, count) pairs

MapReduce: Intuition through an Example

- The result of the task will be a list of (word, count) pairs
- 2 possible scenarios:
  - The total number of (word, count) pairs fit into main memory
  - The total number of (word, count) pairs **does not** fit into main memory

Word Counting: Result Fits into Main Memory



Contrary to popular belief, Lorem Ipsum is not simply random text.
It has roots in a piece of classical Latin literature from 45 BC, making it over 2000 years old. Richard McClintock, a Latin professor at Hampden-Sydney College in Virginia, looked up one of the more obscure Latin words, consectetur, from a Lorem Ipsum passage, and going through the cites of the word in classical literature, discovered the undoubtable source.
Lorem Ipsum comes from sections 1.10.32 and 1.10.33 of "de Finibus Bonorum et Malorum" (The Extremes of Good and Evil) by Cicero, written in 45 BC.
This book is a treatise on the theory of ethics, very popular during the Renaissance. The first line of Lorem Ipsum, "Lorem ipsum dolor sit amet..", comes from a line in section 1.10.32.

`doc.txt`

Word Counting: Result Fits into Main Memory

Contrary to popular belief, Lorem Ipsum is not simply random text.
It has roots in a piece of classical Latin literature from 45 BC, making it over 2000 years old. Richard McClintock, a Latin professor at Hampden-Sydney College in Virginia, looked up one of the more obscure Latin words, consectetur, from a Lorem Ipsum passage, and going through the cites of the word in classical literature, discovered the undoubtable source.
Lorem Ipsum comes from sections 1.10.32 and 1.10.33 of "de Finibus Bonorum et Malorum" (The Extremes of Good and Evil) by Cicero, written in 45 BC.
This book is a treatise on the theory of ethics, very popular during the Renaissance. The first line of Lorem Ipsum, "Lorem ipsum dolor sit amet..", comes from a line in section 1.10.32.

`doc.txt`

Initialize an empty hash map/table

word	count

Word Counting: Result Fits into Main Memory

Contrary to popular belief, Lorem Ipsum is not simply random text.

It has roots in a piece of classical Latin literature from 45 BC, making it over 2000 years old. Richard McClintock, a Latin professor at Hampden-Sydney College in Virginia, looked up one of the more obscure Latin words, consectetur, from a Lorem Ipsum passage, and going through the cites of the word in classical literature, discovered the undoubtable source.

Lorem Ipsum comes from sections 1.10.32 and 1.10.33 of "de Finibus Bonorum et Malorum" (The Extremes of Good and Evil) by Cicero, written in 45 BC.

This book is a treatise on the theory of ethics, very popular during the Renaissance. The first line of Lorem Ipsum, "Lorem ipsum dolor sit amet..", comes from a line in section 1.10.32.

Process one line at a time

word	count
Lorem	1
…	…

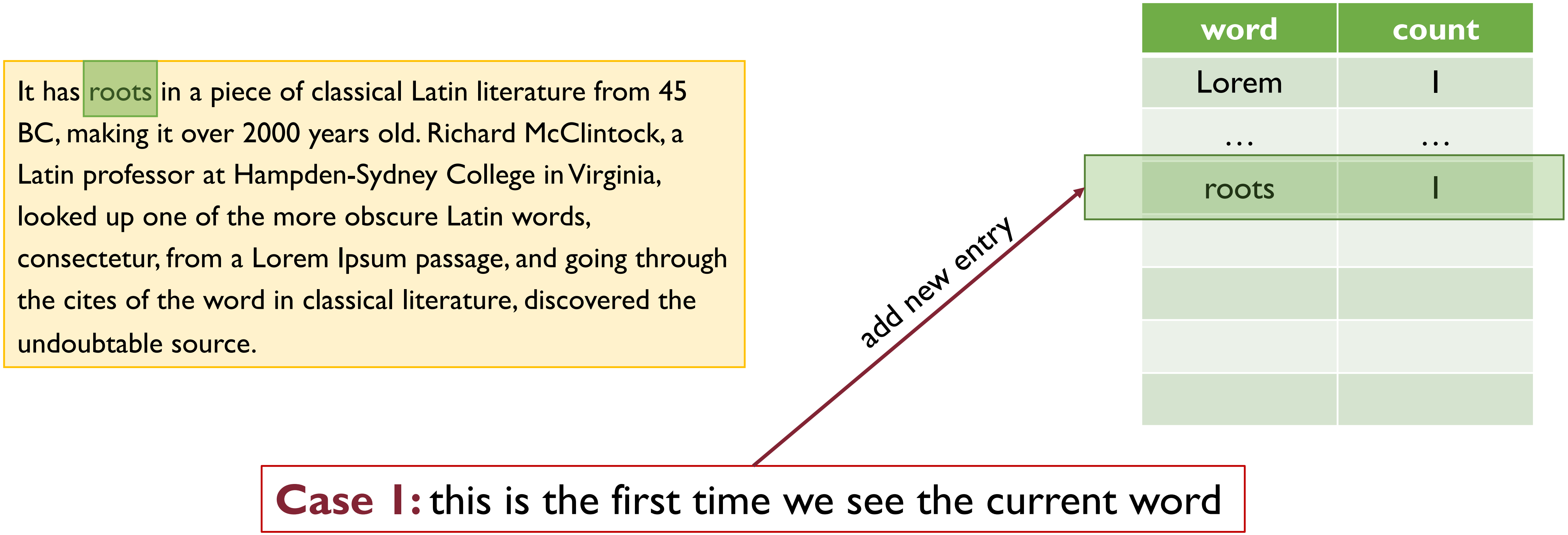
Word Counting: Result Fits into Main Memory

It has roots in a piece of classical Latin literature from 45 BC, making it over 2000 years old. Richard McClintock, a Latin professor at Hampden-Sydney College in Virginia, looked up one of the more obscure Latin words, consectetur, from a Lorem Ipsum passage, and going through the cites of the word in classical literature, discovered the undoubtable source.

word	count
Lorem	1
…	…

Extract each individual word from a line and update the hash map

Word Counting: Result Fits into Main Memory

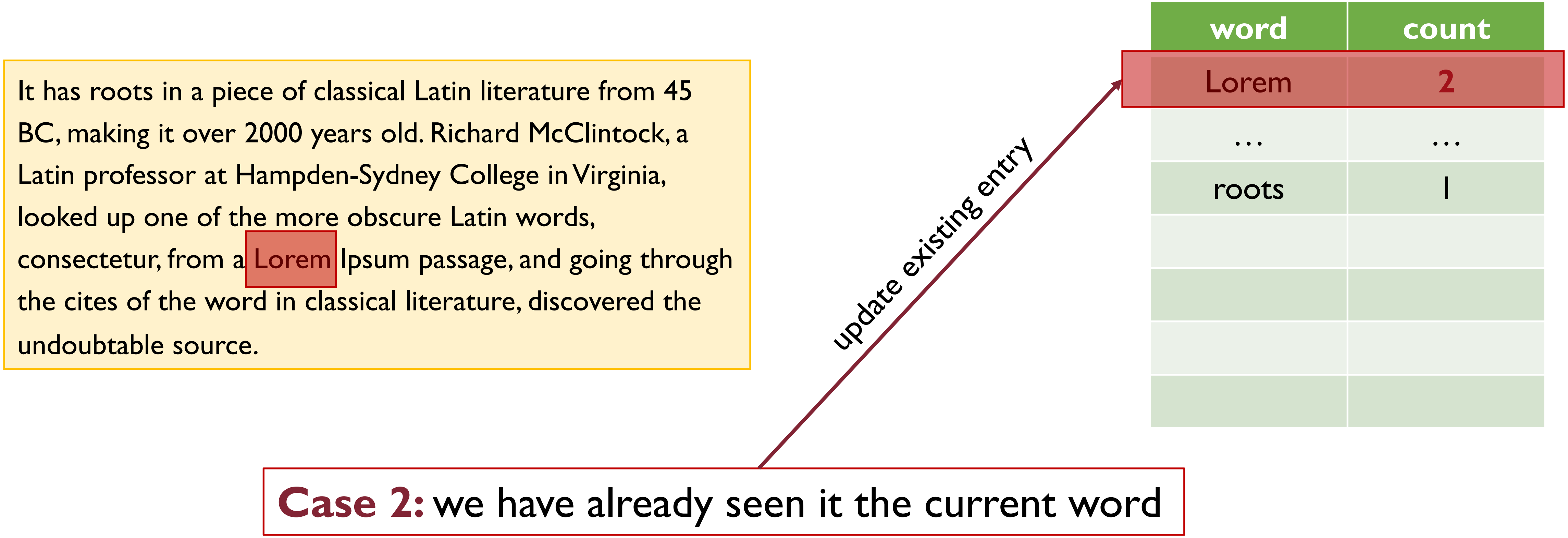


It has roots in a piece of classical Latin literature from 45 BC, making it over 2000 years old. Richard McClintock, a Latin professor at Hampden-Sydney College in Virginia, looked up one of the more obscure Latin words, consectetur, from a Lorem Ipsum passage, and going through the cites of the word in classical literature, discovered the undoubtable source.

word	count
Lorem	1
…	…
roots	1

Case 1: this is the first time we see the current word

Word Counting: Result Fits into Main Memory



It has roots in a piece of classical Latin literature from 45 BC, making it over 2000 years old. Richard McClintock, a Latin professor at Hampden-Sydney College in Virginia, looked up one of the more obscure Latin words, consectetur, from a Lorem Ipsum passage, and going through the cites of the word in classical literature, discovered the undoubtable source.

update existing entry

word	count
Lorem	**2**
…	…
roots	1

Case 2: we have already seen it the current word

Word Counting: Result Does Not Fit into Main Memory

- Use a mixture of simple scripting and UNIX command line tools

```
> print_words(doc.txt) | sort | uniq -c
```

`print_words` is a simple script which just prints each word of **`doc.txt`** to **`stdout`**, one per line

Word Counting: Result Does Not Fit into Main Memory

- Use a mixture of simple scripting and UNIX command line tools

```
> print_words(doc.txt) | sort | uniq -c
```

`print_words` is a simple script which just prints each word of **`doc.txt`** to **`stdout`**, one per line

- This solution nicely fits the MapReduce philosophy! We'll see how

Word Counting: Result Does Not Fit into Main Memory

- Use a mixture of simple scripting and UNIX command line tools

```
> print_words(doc.txt) | sort | uniq -c
```

`print_words` is a simple script which just prints each word of **`doc.txt`** to **`stdout`**, one per line

- This solution nicely fits the MapReduce philosophy! We'll see how

Note:

UNIX **`sort`** utility uses an external merge sorting algorithm
and therefore it doesn't require the data to be sorted to fit entirely in main memory

MapReduce: Steps

- **Input:** a set of (key, value) pairs

MapReduce: Steps

- **Input:** a set of (key, value) pairs
- **Output:** another set of (key, value) pairs

MapReduce: Steps

- **Input:** a set of (key, value) pairs
- **Output:** another set of (key, value) pairs
- Programmer defines **2 methods**:
  - `map`
  - `reduce`

MapReduce: Steps

- **Input:** a set of (key, value) pairs
- **Output:** another set of (key, value) pairs
- Programmer defines **2 methods**:
  - **`map`**
  - **`reduce`**
- An intermediate **`shuffle`** step is implicitly provided by the framework

MapReduce: Steps (More Formally)

- Input key-value pairs: $\{(k_1, v_1), (k_2, v_2), \ldots, (k_M, v_M)\}$

MapReduce: Steps (More Formally)

- Input key-value pairs: $\{(k_1, v_1), (k_2, v_2), \ldots, (k_M, v_M)\}$
- **`map`**(k_i, v_i) → $\{(k_i', v_i')\}^*$
  - Takes an input key-value pair and outputs a set of 0 or more new, intermediate key-value pairs
  - One **`map`** function call for each input key-value pair (k_i, v_i)
    - **map task** → multiple map calls executed in parallel on a subset of the input key-value pairs

MapReduce: Steps (More Formally)

- Input key-value pairs: $\{(k_1, v_1), (k_2, v_2), \ldots, (k_M, v_M)\}$
- **`map`**$(k_i, v_i) \rightarrow \{(k_i', v_i')\}^*$
  - Takes an input key-value pair and outputs a set of 0 or more new, intermediate key-value pairs
  - One **`map`** function call for each input key-value pair (k_i, v_i)
    - **map task** → multiple map calls executed in parallel on a subset of the input key-value pairs
- **`reduce`**$(k_i', \{v_i'\}^*) \rightarrow \{(k_i', v_i'')\}^*$
  - All values v_i' associated with the same key k_i' are reduced together
  - One **`reduce`** function call for each unique key k_i'

Word Counting: Map (`print_words`)

```
> print_words(doc.txt)
```

- Resembles the role of **`map`** function in MapReduce paradigm

Word Counting: Map (`print_words`)

```
> print_words(doc.txt)
```

- Resembles the role of **`map`** function in MapReduce paradigm
- A **`map`** function:
  - takes as input the original data (e.g., a chunk of the whole **`doc.txt`** file)
  - produces as output something out of the data called **intermediate keys** (e.g., a word for each line in the chunk)

Word Counting: Shuffle (`sort`)

```
> print_words(doc.txt) | sort
```

- The intermediate keys generated by the map function are sorted and shuffled

Word Counting: Shuffle (`sort`)

```
> print_words(doc.txt) | sort
```

- The intermediate keys generated by the map function are sorted and shuffled
- Note that intermediate keys are not unique!

Word Counting: Shuffle (sort)

```
> print_words(doc.txt) | sort
```

- The intermediate keys generated by the map function are sorted and shuffled
- Note that intermediate keys are not unique!
- For example, **print_words** may print out the same word multiple times

Word Counting: Reduce (`uniq -c`)

```
> print_words(doc.txt) | sort | uniq -c
```

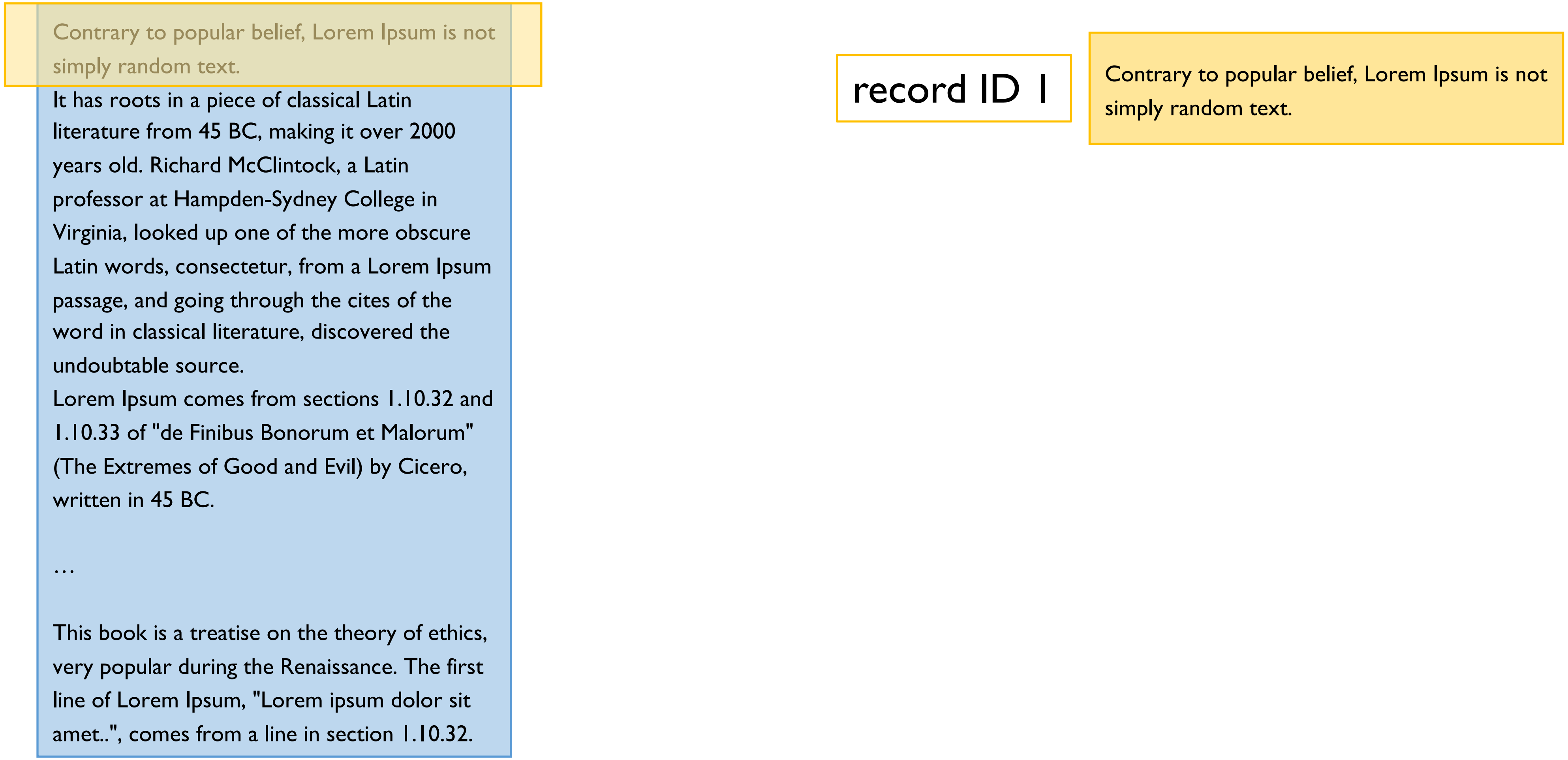
- Resembles the role of `reduce` function in MapReduce paradigm

Word Counting: Reduce (uniq -c)

```
> print_words(doc.txt) | sort | uniq -c
```

- Resembles the role of reduce function in MapReduce paradigm
- A reduce function:
  - takes as input the groups of intermediate keys
  - computes an aggregating/filtering/transforming function over those keys
  - persists out the result

MapReduce: Input Key-Value Pairs



Contrary to popular belief, Lorem Ipsum is not simply random text.

It has roots in a piece of classical Latin literature from 45 BC, making it over 2000 years old. Richard McClintock, a Latin professor at Hampden-Sydney College in Virginia, looked up one of the more obscure Latin words, consectetur, from a Lorem Ipsum passage, and going through the cites of the word in classical literature, discovered the undoubtable source.

Lorem Ipsum comes from sections 1.10.32 and 1.10.33 of "de Finibus Bonorum et Malorum" (The Extremes of Good and Evil) by Cicero, written in 45 BC.

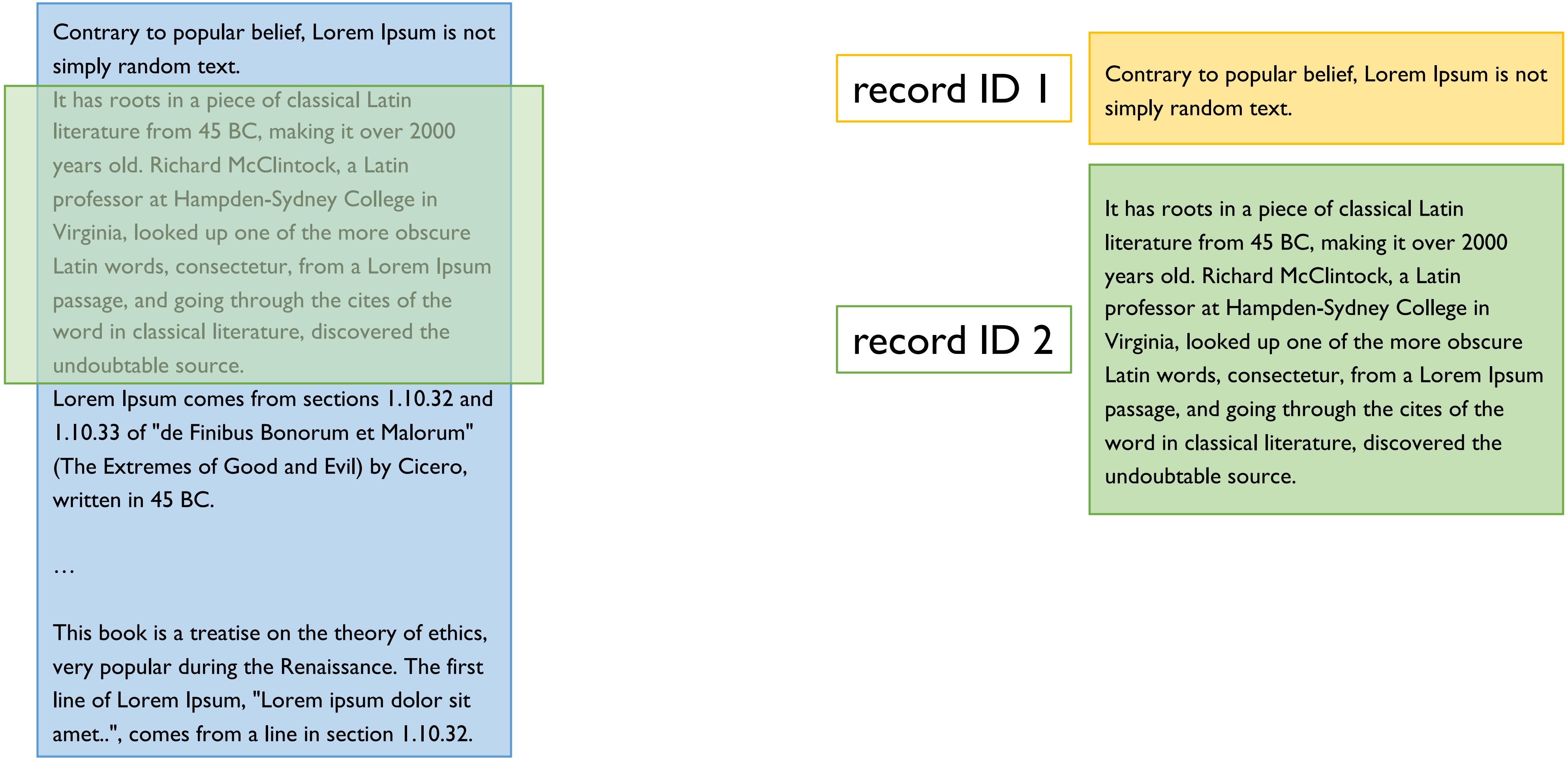
…

This book is a treatise on the theory of ethics, very popular during the Renaissance. The first line of Lorem Ipsum, "Lorem ipsum dolor sit amet..", comes from a line in section 1.10.32.

record ID 1

Contrary to popular belief, Lorem Ipsum is not simply random text.

MapReduce: Input Key-Value Pairs



Contrary to popular belief, Lorem Ipsum is not simply random text.

It has roots in a piece of classical Latin literature from 45 BC, making it over 2000 years old. Richard McClintock, a Latin professor at Hampden-Sydney College in Virginia, looked up one of the more obscure Latin words, consectetur, from a Lorem Ipsum passage, and going through the cites of the word in classical literature, discovered the undoubtable source.

Lorem Ipsum comes from sections 1.10.32 and 1.10.33 of "de Finibus Bonorum et Malorum" (The Extremes of Good and Evil) by Cicero, written in 45 BC.

…

This book is a treatise on the theory of ethics, very popular during the Renaissance. The first line of Lorem Ipsum, "Lorem ipsum dolor sit amet..", comes from a line in section 1.10.32.

record ID 1

Contrary to popular belief, Lorem Ipsum is not simply random text.

record ID 2

It has roots in a piece of classical Latin literature from 45 BC, making it over 2000 years old. Richard McClintock, a Latin professor at Hampden-Sydney College in Virginia, looked up one of the more obscure Latin words, consectetur, from a Lorem Ipsum passage, and going through the cites of the word in classical literature, discovered the undoubtable source.

MapReduce: Input Key-Value Pairs

Contrary to popular belief, Lorem Ipsum is not simply random text.

It has roots in a piece of classical Latin literature from 45 BC, making it over 2000 years old. Richard McClintock, a Latin professor at Hampden-Sydney College in Virginia, looked up one of the more obscure Latin words, consectetur, from a Lorem Ipsum passage, and going through the cites of the word in classical literature, discovered the undoubtable source.

Lorem Ipsum comes from sections 1.10.32 and 1.10.33 of "de Finibus Bonorum et Malorum" (The Extremes of Good and Evil) by Cicero, written in 45 BC.

…

This book is a treatise on the theory of ethics, very popular during the Renaissance. The first line of Lorem Ipsum, "Lorem ipsum dolor sit amet..", comes from a line in section 1.10.32.

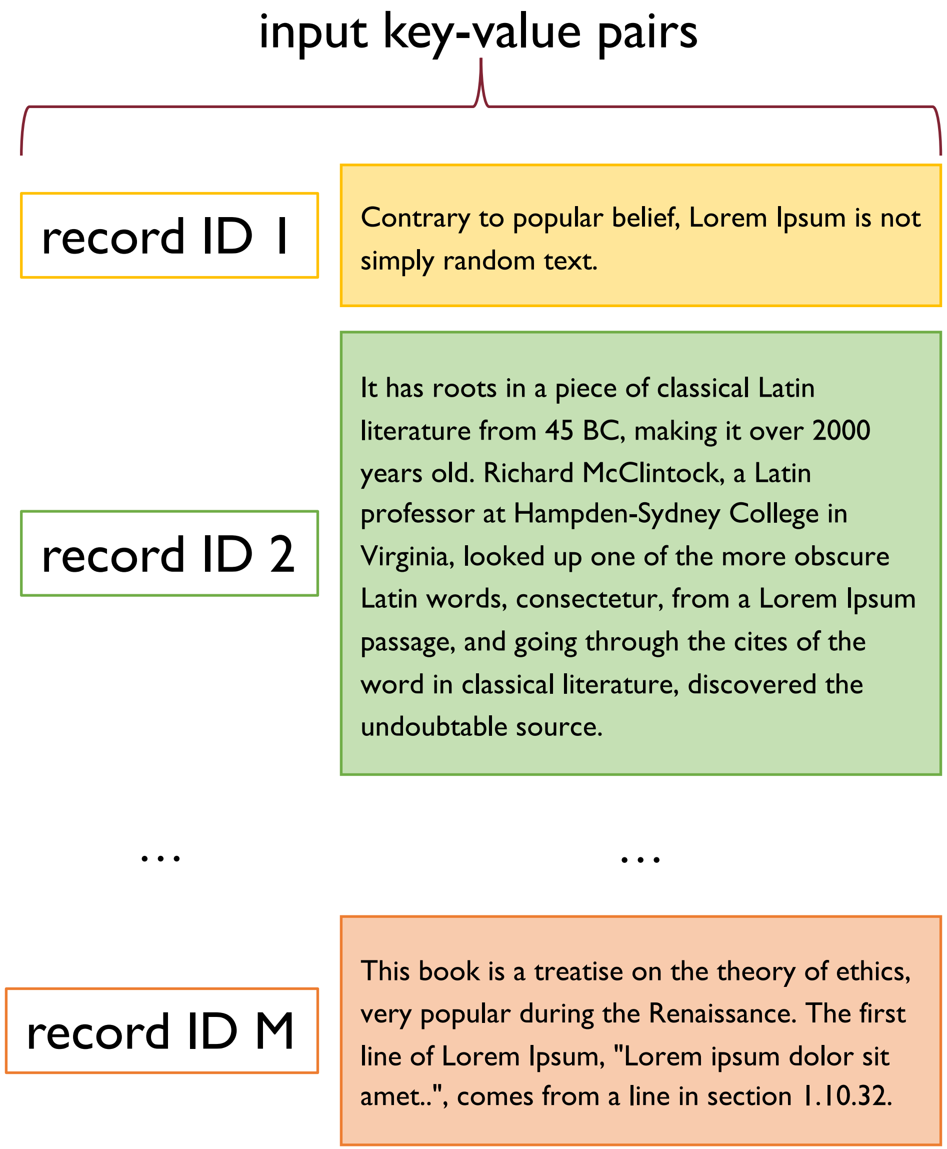
record ID 1	Contrary to popular belief, Lorem Ipsum is not simply random text.
record ID 2	It has roots in a piece of classical Latin literature from 45 BC, making it over 2000 years old. Richard McClintock, a Latin professor at Hampden-Sydney College in Virginia, looked up one of the more obscure Latin words, consectetur, from a Lorem Ipsum passage, and going through the cites of the word in classical literature, discovered the undoubtable source.
…	…
record ID M	This book is a treatise on the theory of ethics, very popular during the Renaissance. The first line of Lorem Ipsum, "Lorem ipsum dolor sit amet..", comes from a line in section 1.10.32.

MapReduce: Input Key-Value Pairs

Contrary to popular belief, Lorem Ipsum is not simply random text.
It has roots in a piece of classical Latin literature from 45 BC, making it over 2000 years old. Richard McClintock, a Latin professor at Hampden-Sydney College in Virginia, looked up one of the more obscure Latin words, consectetur, from a Lorem Ipsum passage, and going through the cites of the word in classical literature, discovered the undoubtable source.
Lorem Ipsum comes from sections 1.10.32 and 1.10.33 of "de Finibus Bonorum et Malorum" (The Extremes of Good and Evil) by Cicero, written in 45 BC.

…

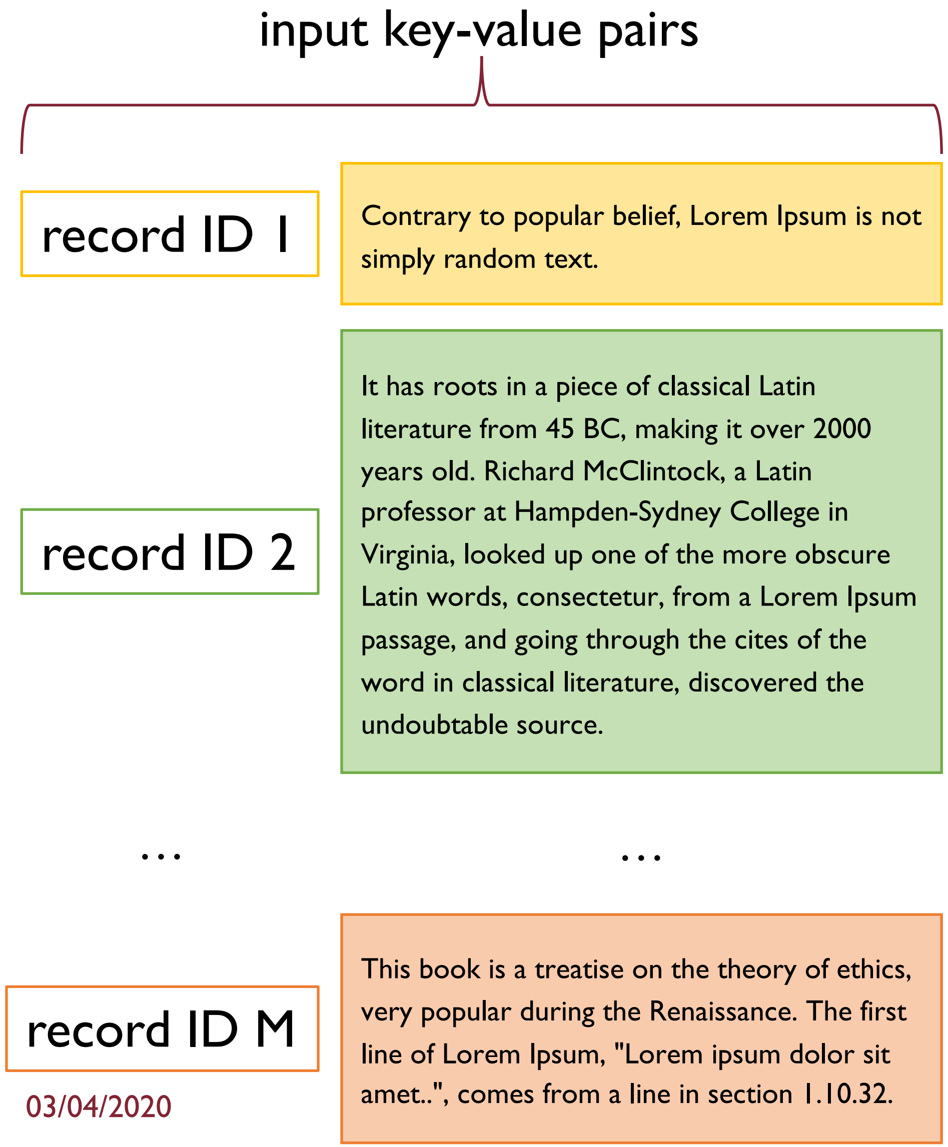
This book is a treatise on the theory of ethics, very popular during the Renaissance. The first line of Lorem Ipsum, "Lorem ipsum dolor sit amet..", comes from a line in section 1.10.32.



input key-value pairs

record ID 1	Contrary to popular belief, Lorem Ipsum is not simply random text.
record ID 2	It has roots in a piece of classical Latin literature from 45 BC, making it over 2000 years old. Richard McClintock, a Latin professor at Hampden-Sydney College in Virginia, looked up one of the more obscure Latin words, consectetur, from a Lorem Ipsum passage, and going through the cites of the word in classical literature, discovered the undoubtable source.
…	…
record ID M	This book is a treatise on the theory of ethics, very popular during the Renaissance. The first line of Lorem Ipsum, "Lorem ipsum dolor sit amet..", comes from a line in section 1.10.32.

MapReduce: The Map Step



input key-value pairs

record ID 1

Contrary to popular belief, Lorem Ipsum is not simply random text.

record ID 2

It has roots in a piece of classical Latin literature from 45 BC, making it over 2000 years old. Richard McClintock, a Latin professor at Hampden-Sydney College in Virginia, looked up one of the more obscure Latin words, consectetur, from a Lorem Ipsum passage, and going through the cites of the word in classical literature, discovered the undoubtable source.

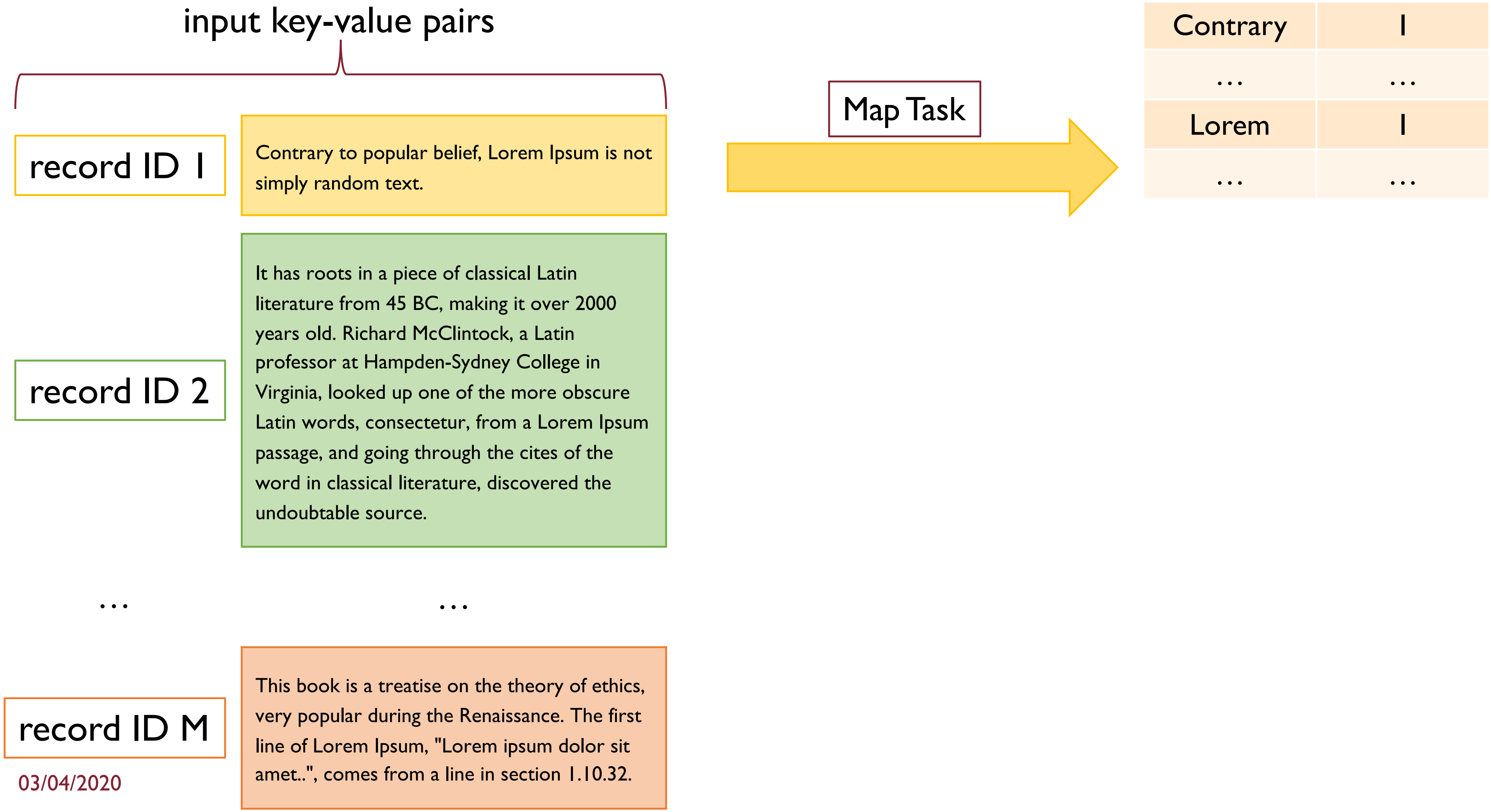
…

…

record ID M

This book is a treatise on the theory of ethics, very popular during the Renaissance. The first line of Lorem Ipsum, "Lorem ipsum dolor sit amet..", comes from a line in section 1.10.32.

MapReduce: The Map Step



input key-value pairs

record ID 1

Contrary to popular belief, Lorem Ipsum is not simply random text.

record ID 2

It has roots in a piece of classical Latin literature from 45 BC, making it over 2000 years old. Richard McClintock, a Latin professor at Hampden-Sydney College in Virginia, looked up one of the more obscure Latin words, consectetur, from a Lorem Ipsum passage, and going through the cites of the word in classical literature, discovered the undoubtable source.

…

…

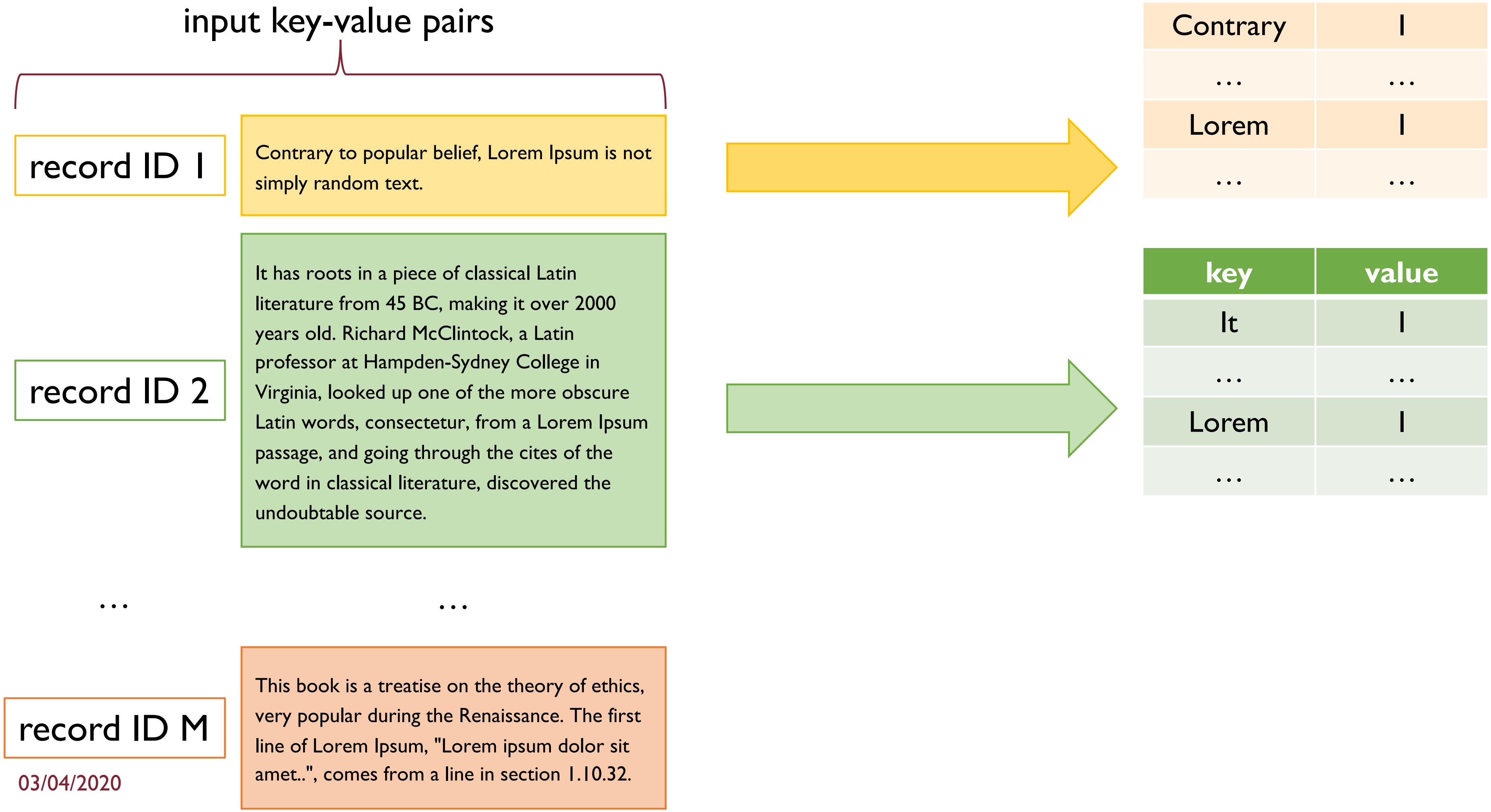
record ID M

This book is a treatise on the theory of ethics, very popular during the Renaissance. The first line of Lorem Ipsum, "Lorem ipsum dolor sit amet..", comes from a line in section 1.10.32.

Map Task

key	value
Contrary	1
…	…
Lorem	1
…	…

MapReduce: The Map Step



input key-value pairs

record ID 1

Contrary to popular belief, Lorem Ipsum is not simply random text.

record ID 2

It has roots in a piece of classical Latin literature from 45 BC, making it over 2000 years old. Richard McClintock, a Latin professor at Hampden-Sydney College in Virginia, looked up one of the more obscure Latin words, consectetur, from a Lorem Ipsum passage, and going through the cites of the word in classical literature, discovered the undoubtable source.

… …

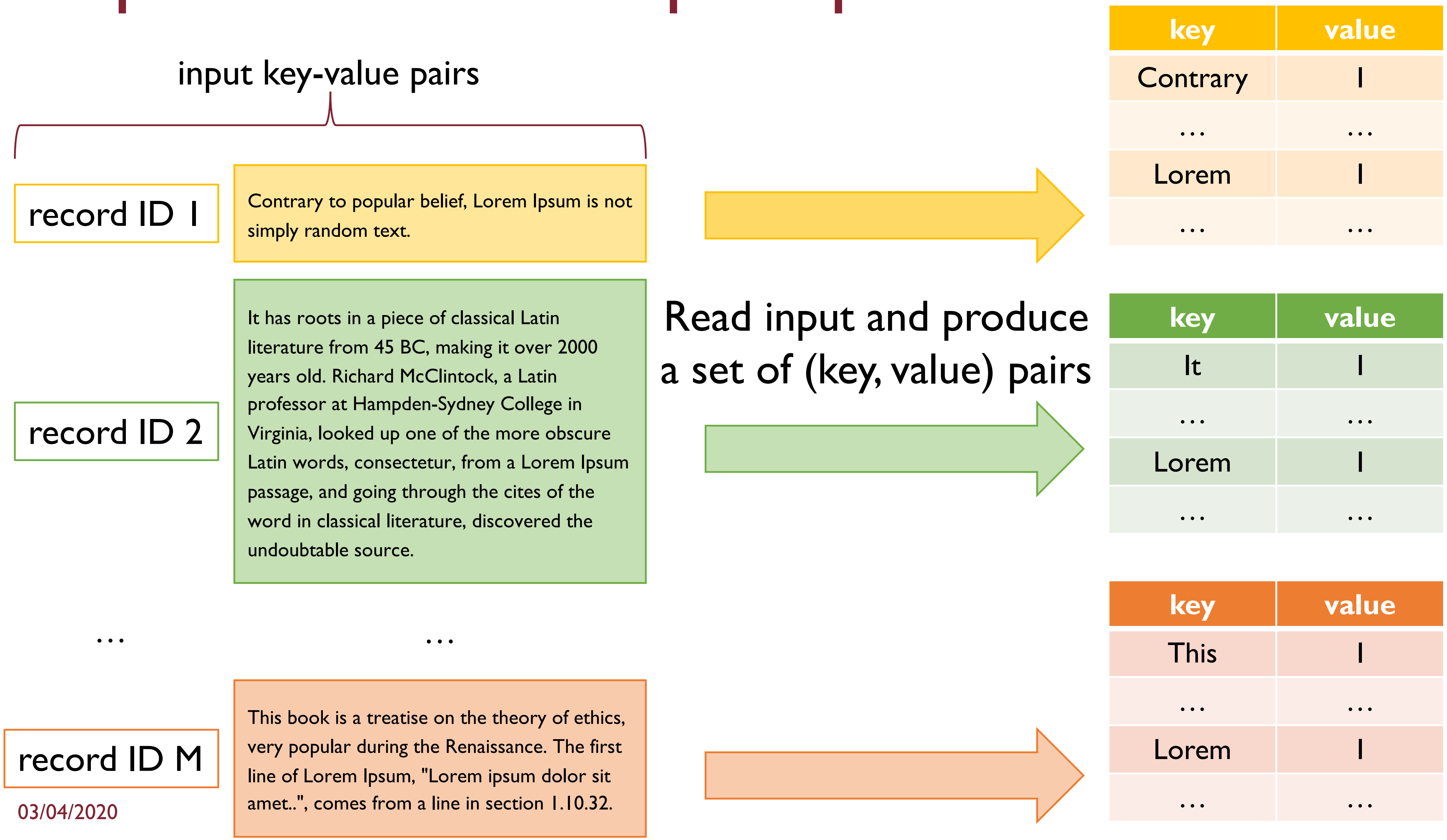
record ID M

This book is a treatise on the theory of ethics, very popular during the Renaissance. The first line of Lorem Ipsum, "Lorem ipsum dolor sit amet..", comes from a line in section 1.10.32.

key	value
Contrary	1
…	…
Lorem	1
…	…

key	value
It	1
…	…
Lorem	1
…	…

MapReduce: The Map Step



input key-value pairs

record ID 1

Contrary to popular belief, Lorem Ipsum is not simply random text.

record ID 2

It has roots in a piece of classical Latin literature from 45 BC, making it over 2000 years old. Richard McClintock, a Latin professor at Hampden-Sydney College in Virginia, looked up one of the more obscure Latin words, consectetur, from a Lorem Ipsum passage, and going through the cites of the word in classical literature, discovered the undoubtable source.

…

…

record ID M

This book is a treatise on the theory of ethics, very popular during the Renaissance. The first line of Lorem Ipsum, "Lorem ipsum dolor sit amet..", comes from a line in section 1.10.32.

Read input and produce a set of (key, value) pairs

key	value
Contrary	1
…	…
Lorem	1
…	…

key	value
It	1
…	…
Lorem	1
…	…

key	value
This	1
…	…
Lorem	1
…	…

MapReduce: The Shuffle Step

key	value
Contrary	1
…	…
Lorem	1
…	…

key	value
It	1
…	…
Lorem	1
…	…

key	value
This	1
…	…
Lorem	1
…	…

MapReduce: The Shuffle Step

key	value
Contrary	1
…	…
Lorem	1
…	…

key	value
It	1
…	…
Lorem	1
…	…

key	value
This	1
…	…
Lorem	1
…	…

Collect (i.e., group) all pairs with the same key

key	value
A	1
A	1
…	…
Lorem	1
Lorem	1
Lorem	1

key	value
the	1
the	1
…	…
Ipsum	1
Ipsum	1
Ipsum	1

MapReduce: The Reduce Step

key	value
A	1
A	1
…	…
Lorem	1
Lorem	1
Lorem	1

key	value
the	1
the	1
…	…
Ipsum	1
Ipsum	1
Ipsum	1

Process all values belonging to a given key and output the result

key	value
A	2
…	…
Ipsum	3
…	…
Lorem	3
…	…
the	2
…	…
undoubtable	1

MapReduce: Word Counting Pseudocode

```
map(key, value):
# key: docID; value: text
    foreach word in value:
        emit(word, 1)
```

```
reduce(key, values):
# key: word; values: iterator
    result = 0
    foreach v in values:
        result += v
    emit(key, result)
```

MapReduce: Word Counting Pseudocode

```
map(key, value):
# key: docID; value: text
    foreach word in value:
        emit(word, 1)
```

Note:
input (key, value) can be just a single pair as the actual split of the input is done transparently by the framework

```
reduce(key, values):
# key: word; values: iterator
    result = 0
    foreach v in values:
        result += v
    emit(key, result)
```

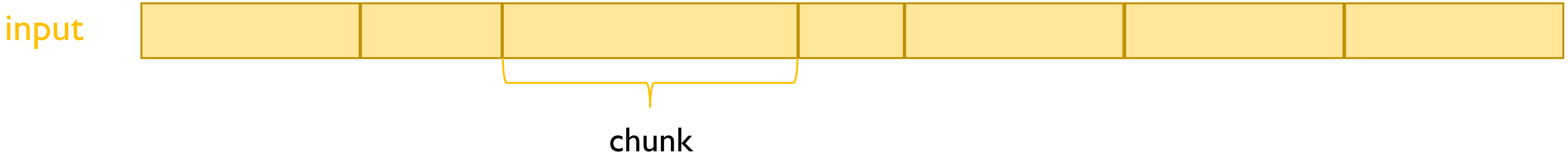
MapReduce: PROs and CONs

- MapReduce is **great** for:
  - Problems that require many sequential data access (from disk)
  - Large batch jobs (i.e., not interactive nor real time)

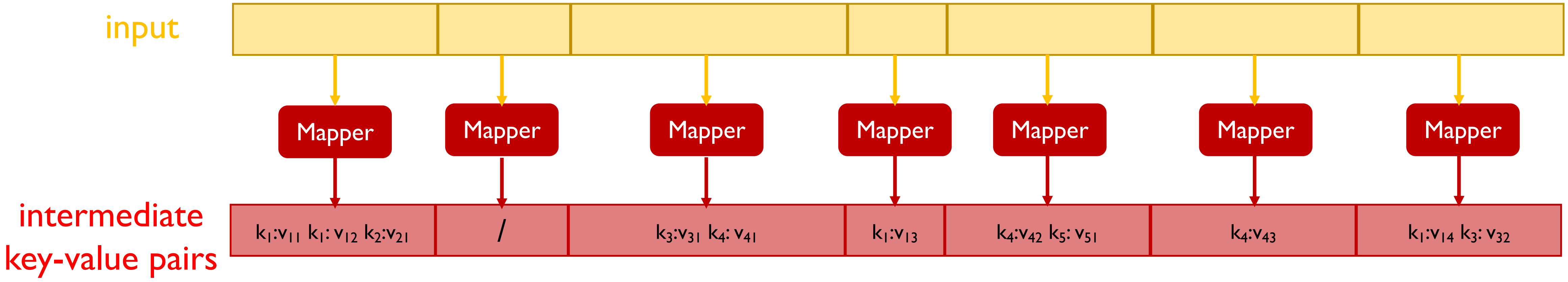
MapReduce: PROs and CONs

- MapReduce is **great** for:
  - Problems that require many sequential data access (from disk)
  - Large batch jobs (i.e., not interactive nor real time)
- MapReduce is **not suitable** for:
  - Problems that require random access to data
  - Working with graphs
  - Interdependent data

MapReduce on a Single-Node



MapReduce on a Single-Node



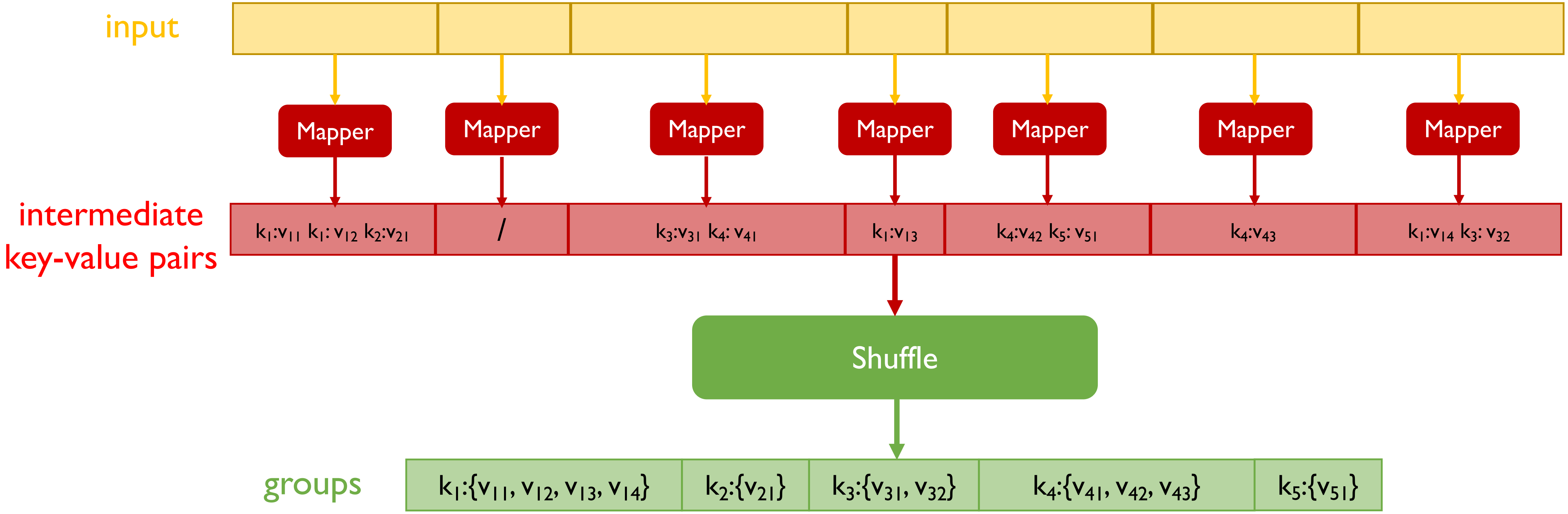
input

Mapper Mapper Mapper Mapper Mapper Mapper Mapper

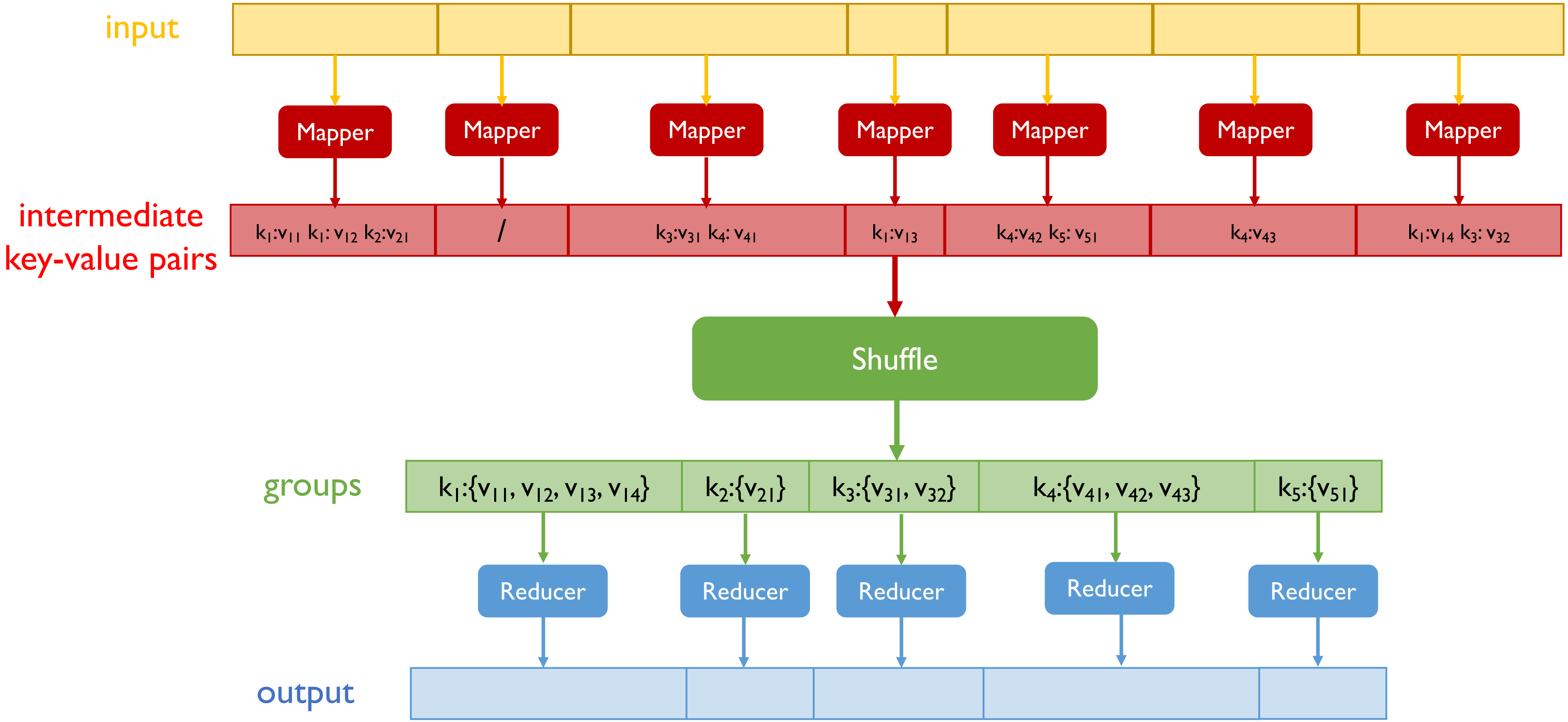
intermediate key-value pairs

k_1:v_{11} k_1: v_{12} k_2:v_{21}	/	k_3:v_{31} k_4: v_{41}	k_1:v_{13}	k_4:v_{42} k_5: v_{51}	k_4:v_{43}	k_1:v_{14} k_3: v_{32}

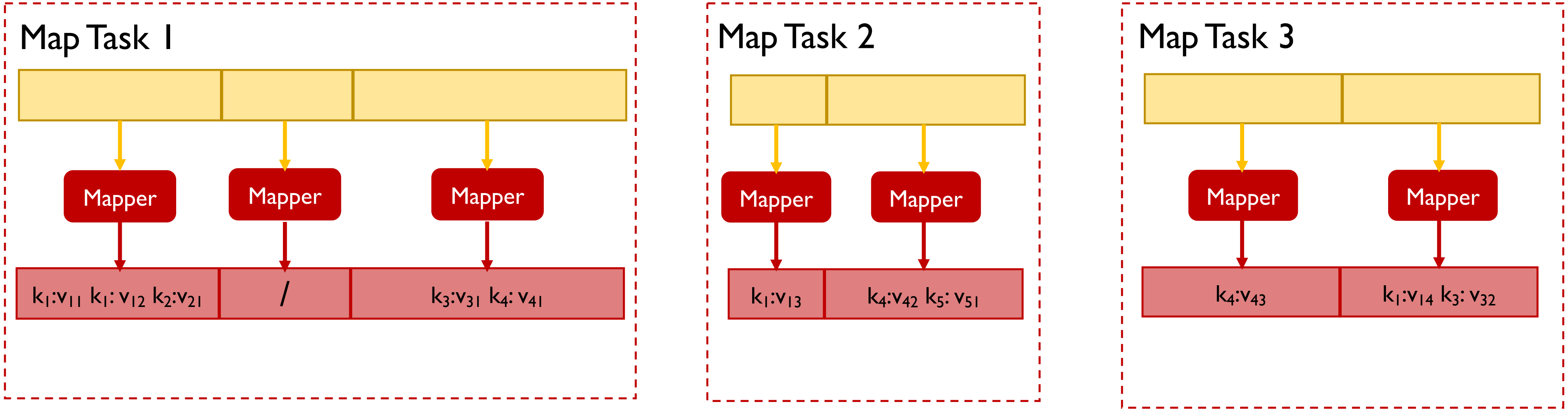
MapReduce on a Single-Node


input
Mapper
Mapper
Mapper
Mapper
Mapper
Mapper
Mapper
intermediate key-value pairs
k_1:v_{11} k_1: v_{12} k_2:v_{21}
/
k_3:v_{31} k_4: v_{41}
k_1:v_{13}
k_4:v_{42} k_5: v_{51}
k_4:v_{43}
k_1:v_{14} k_3: v_{32}
Shuffle
groups
k_1:{v_{11}, v_{12}, v_{13}, v_{14}}
k_2:{v_{21}}
k_3:{v_{31}, v_{32}}
k_4:{v_{41}, v_{42}, v_{43}}
k_5:{v_{51}}

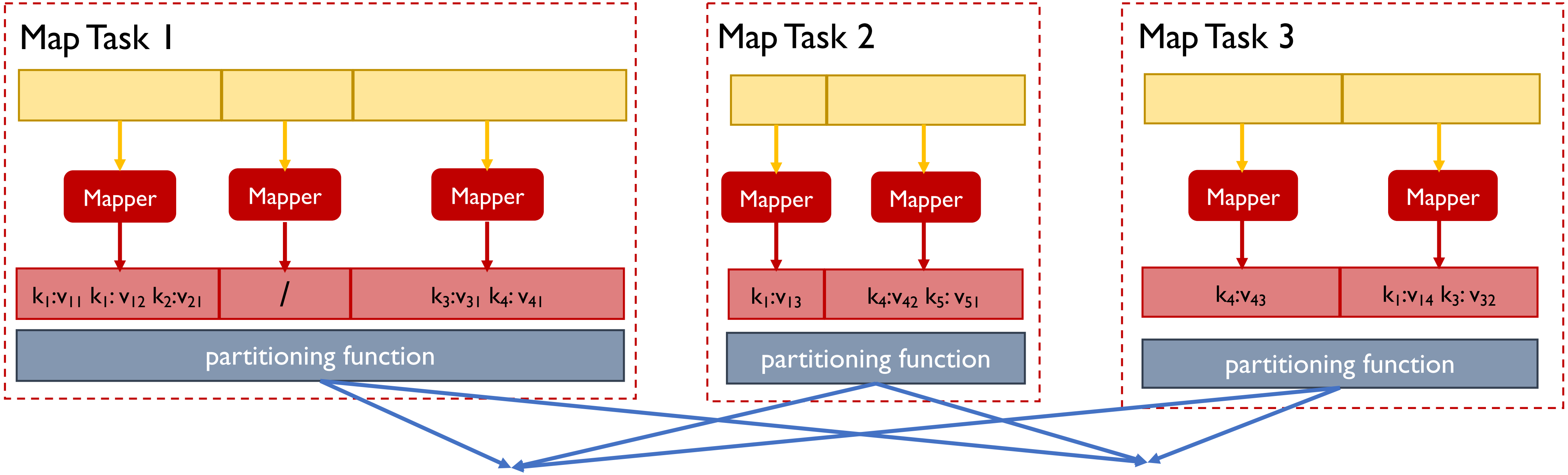
MapReduce on a Single-Node


input
Mapper
Mapper
Mapper
Mapper
Mapper
Mapper
Mapper
intermediate key-value pairs
k_1:v_{11} k_1: v_{12} k_2:v_{21}
/
k_3:v_{31} k_4: v_{41}
k_1:v_{13}
k_4:v_{42} k_5: v_{51}
k_4:v_{43}
k_1:v_{14} k_3: v_{32}
Shuffle
groups
k_1:{v_{11}, v_{12}, v_{13}, v_{14}}
k_2:{v_{21}}
k_3:{v_{31}, v_{32}}
k_4:{v_{41}, v_{42}, v_{43}}
k_5:{v_{51}}
Reducer
Reducer
Reducer
Reducer
Reducer
output

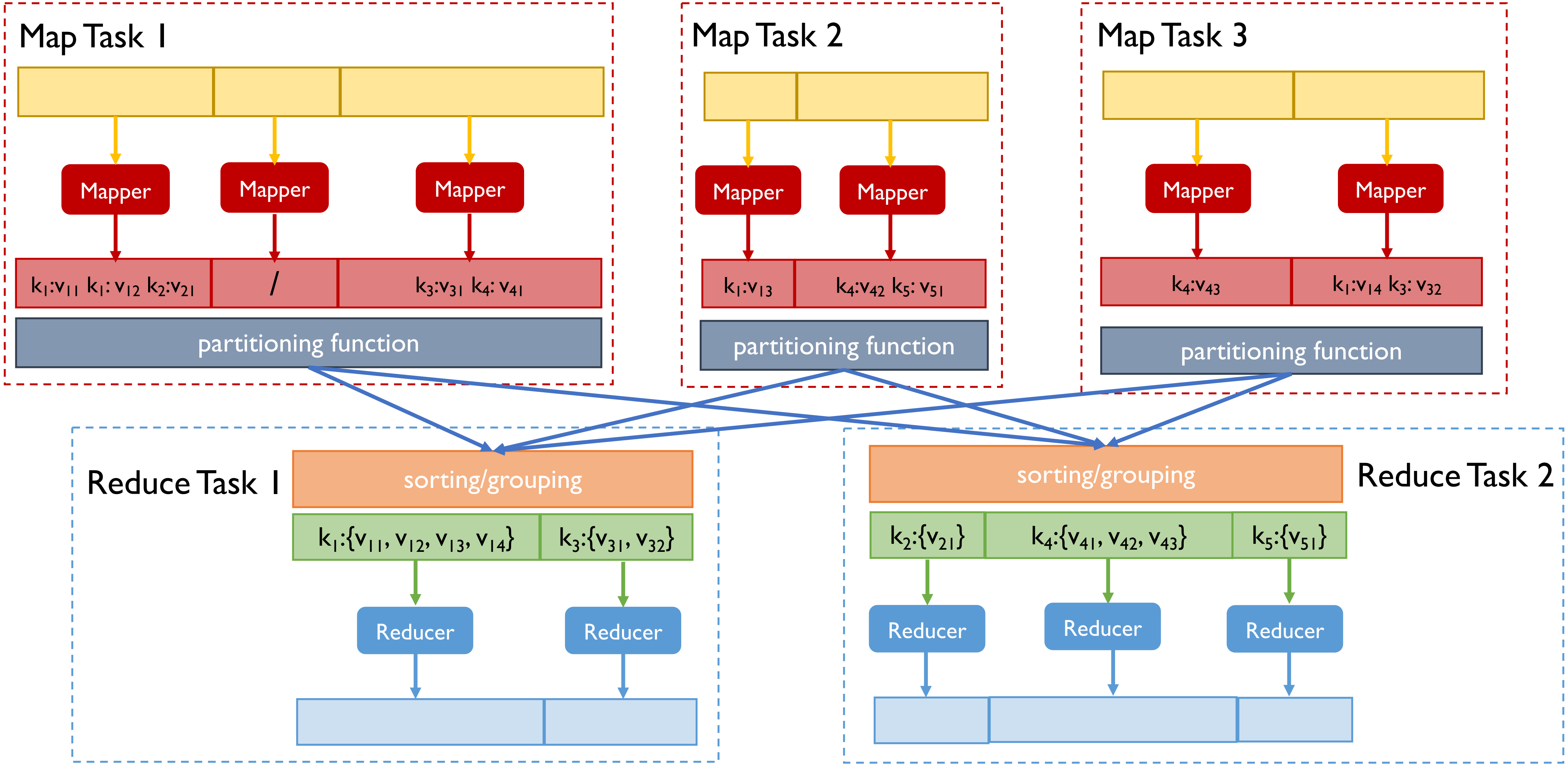
MapReduce on a Cluster


Map Task 1
Mapper
Mapper
Mapper
k_1:v_{11} k_1: v_{12} k_2:v_{21}
/
k_3:v_{31} k_4: v_{41}
Map Task 2
Mapper
Mapper
k_1:v_{13}
k_4:v_{42} k_5: v_{51}
Map Task 3
Mapper
Mapper
k_4:v_{43}
k_1:v_{14} k_3: v_{32}

MapReduce on a Cluster



MapReduce on a Cluster



MapReduce: The Infrastructure

- Remember! Programmer needs only to specify **`map`** and **`reduce`** functions

MapReduce: The Infrastructure

- Remember! Programmer needs only to specify `map` and `reduce` functions
- Everything else is managed by the infrastructure
  - Input data partitioning (physical = chunk/block and logical = split)
  - Scheduling tasks across nodes of the cluster
  - Shuffling/group by of intermediate keys output by mappers
  - Handling node failures
  - Managing inter-node communications

Data Flow

- Both input and output are stored on the distributed file system
  - MapReduce scheduler tries to allocate map tasks "close" to data
  - Each map task running on a node will be using the chunks of data that are stored on that node (chunk server)

Data Flow

- Both input and output are stored on the distributed file system
  - MapReduce scheduler tries to allocate map tasks "close" to data
  - Each map task running on a node will be using the chunks of data that are stored on that node (chunk server)
- Intermediate results of map/reduce tasks are stored on local filesystem of each node
  - This is to avoid copies/replica of useless files across the cluster (DFS)

The Master Node

- Takes care of node coordination/orchestration

The Master Node

- Takes care of node coordination/orchestration
  - Status associated with each task (either map or reduce):
    - idle, in-progress, completed

The Master Node

- Takes care of node coordination/orchestration
    - Status associated with each task (either map or reduce):
        - idle, in-progress, completed
    - Idle tasks are eligible to be executed as soon as a worker node becomes available

The Master Node

- Takes care of node coordination/orchestration
  - Status associated with each task (either map or reduce):
    - idle, in-progress, completed
  - Idle tasks are eligible to be executed as soon as a worker node becomes available
  - When a map task completes it sends notification of that to the master node who propagates that information to the reducers

The Master Node

- Takes care of node coordination/orchestration
  - Status associated with each task (either map or reduce):
    - idle, in-progress, completed
  - Idle tasks are eligible to be executed as soon as a worker node becomes available
  - When a map task completes it sends notification of that to the master node who propagates that information to the reducers
  - The master node periodically pings mappers/reducers to detect failures

Failure Detection

- Map worker node fails
  - All the map tasks completed or in-progress at the (failed) worker node are reset to idle
  - Idle map tasks will be eventually rescheduled later on other worker node(s)

Failure Detection

- Map worker node fails
  - All the map tasks completed or in-progress at the (failed) worker node are reset to idle
  - Idle map tasks will be eventually rescheduled later on other worker node(s)
- Reduce worker node fails
  - Only in-progress tasks are reset to idle (completed ones have already output to the DFS)
  - Idle reduce tasks will be eventually rescheduled later on other worker node(s)

Failure Detection

- Map worker node fails
  - All the map tasks completed or in-progress at the (failed) worker node are reset to idle
  - Idle map tasks will be eventually rescheduled later on other worker node(s)
- Reduce worker node fails
  - Only in-progress tasks are reset to idle (completed ones have already output to the DFS)
  - Idle reduce tasks will be eventually rescheduled later on other worker node(s)
- Master node fails → The whole MapReduce job is aborted

How Many Map/Reduce Tasks?

- N = # nodes of the cluster; M = # map tasks; R = # reduce tasks

How Many Map/Reduce Tasks?

- N = # nodes of the cluster; M = # map tasks; R = # reduce tasks
- Again, mostly transparent to the programmer

How Many Map/Reduce Tasks?

- N = # nodes of the cluster; M = # map tasks; R = # reduce tasks
- Again, mostly transparent to the programmer
- Rule of thumb:
    - M >> N (in fact, one map task per DFS chunk is pretty common)
    - Having M >> N speeds up recovery from node failures (what if M = N?)
    - R < M (convenient to have the output spread across a limited number of nodes)

Another Example of MapReduce Task: Join

- Suppose we have two (very large) tables R(A, B) and S(B, C) below
- Both tables are stored in files
- We want to compute the **natural join** T(A, C) = R(A, B) $\bowtie$ S(B, C)

Another Example of MapReduce Task: Join

- Suppose we have two (very large) tables R(A, B) and S(B, C) below
- Both tables are stored in files
- We want to compute the **natural join** T(A, C) = R(A, B) ⋈ S(B, C)

R

A	B
a_1	b_1
a_2	b_1
a_3	b_2
a_4	b_3

⋈

S

B	C
b_2	c_1
b_2	c_2
b_3	c_3

=

T

A	C
a_3	c_1
a_3	c_2
a_4	c_3

Another Example of MapReduce Task: Join

- Assume the set of possible values of column B are $\{b_1, b_2, \ldots, b_k\}$

Another Example of MapReduce Task: Join

- Assume the set of possible values of column B are $\{b_1, b_2, \ldots, b_k\}$
- Associate a hash function *h* which maps each b_i to an integer in $[1, \ldots, k]$

Another Example of MapReduce Task: Join

- Assume the set of possible values of column B are $\{b_1, b_2, \ldots, b_k\}$
- Associate a hash function *h* which maps each b_i to an integer in $[1, \ldots, k]$
- Map task:
  - For each input tuple R(a, b) output an intermediate key-value pair (b, (a, R))
  - For each input tuple S(b, c) output an intermediate key-value pair (b, (c, S))

Another Example of MapReduce Task: Join

- Assume the set of possible values of column B are $\{b_1, b_2, \ldots, b_k\}$
- Associate a hash function *h* which maps each b_i to an integer in $[1, \ldots, k]$
- Map task:
  - For each input tuple R(a, b) output an intermediate key-value pair (b, (a, R))
  - For each input tuple S(b, c) output an intermediate key-value pair (b, (c, S))
- All the intermediate key-value pairs with the same *h*(b) are sent to the same reducer

Another Example of MapReduce Task: Join

- Assume the set of possible values of column B are $\{b_1, b_2, \ldots, b_k\}$
- Associate a hash function *h* which maps each b_i to an integer in $[1, \ldots, k]$
- Map task:
  - For each input tuple R(a, b) output an intermediate key-value pair (b, (a, R))
  - For each input tuple S(b, c) output an intermediate key-value pair (b, (c, S))
- All the intermediate key-value pairs with the same *h*(b) are sent to the same reducer
- Reduce task:
  - Match all the (b, (a, R)) pairs with (b, (c, S)) ones and output (a, b, c)

Same Key-Value Pairs

- A map task may produce many pairs of the form (k, v_1), (k, v_2), … all sharing the same key k

Same Key-Value Pairs

- A map task may produce many pairs of the form (k, v_1), (k, v_2), … all sharing the same key k
- For example, consider again the word counting task
  - A word w may appear several times in the input chunk associated with a mapper
  - Still, the mapper will ouptut the same key-value pair (w, 1) every time it will find an occurrence of w
  - Eventually, all these (same) key-value pairs must be transferred to a reducer

Same Key-Value Pairs

- A map task may produce many pairs of the form (k, v_1), (k, v_2), … all sharing the same key k
- For example, consider again the word counting task
  - A word w may appear several times in the input chunk associated with a mapper
  - Still, the mapper will ouptut the same key-value pair (w, 1) every time it will find an occurrence of w
  - Eventually, all these (same) key-value pairs must be transferred to a reducer
- Can we do any better?

Solution: Combiners

- Combiners can save network transfers by pre-aggregating values at the mapper end

Solution: Combiners

- Combiners can save network transfers by pre-aggregating values at the mapper end
- `combine`(k, $\{v_1, v_2, \ldots, v_m\}$) → (k, v')
  - where v' is the result of an aggregating function computed on $\{v_1, \ldots, v_m\}$

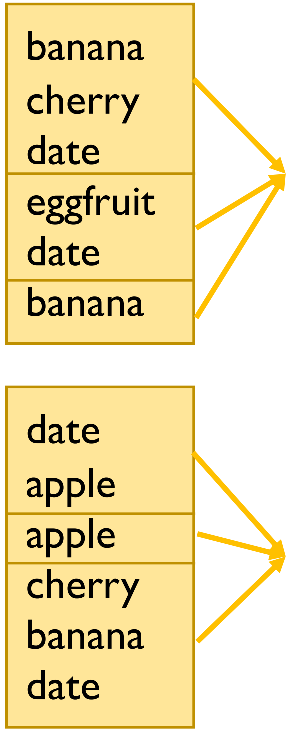
Solution: Combiners

- Combiners can save network transfers by pre-aggregating values at the mapper end
- `combine`(k, $\{v_1, v_2, \ldots, v_m\}$) → (k, v')
    - where v' is the result of an aggregating function computed on $\{v_1, \ldots, v_m\}$
- Usually, combiner computes the same aggregating function of reducer

Solution: Combiners

- Combiners can save network transfers by pre-aggregating values at the mapper end
- `combine`(k, {v_1, v_2, …, v_m}) → (k, v')
  - where v' is the result of an aggregating function computed on {v_1, …, v_m}
- Usually, combiner computes the same aggregating function of reducer
- In the word counting example, at each mapper:
  - `combine`("apple", {1, 1, 1}) → ("apple", 3)

Combiners



banana
cherry
date

eggfruit
date

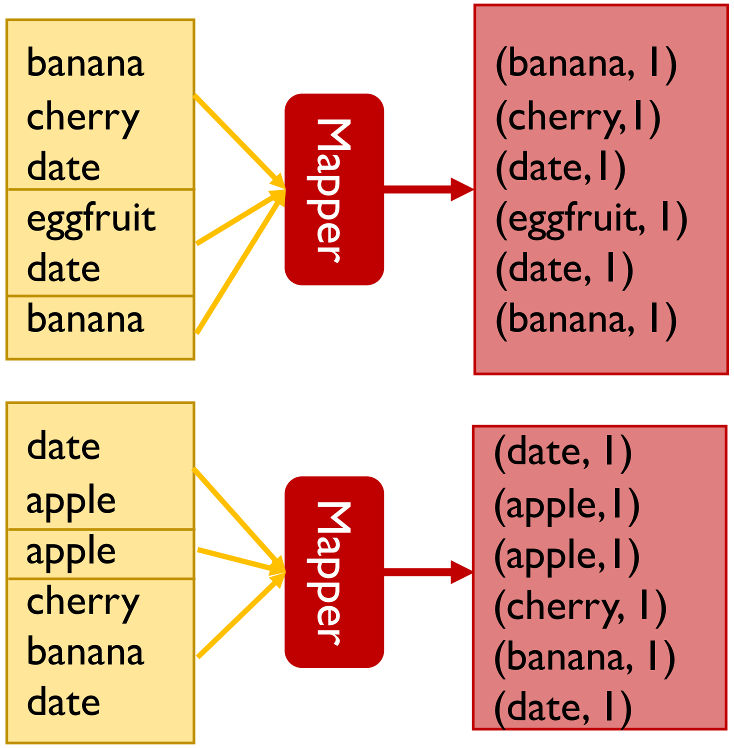
banana

date
apple

apple

cherry
banana
date

Combiners



banana
cherry
date
eggfruit
date
banana

Mapper

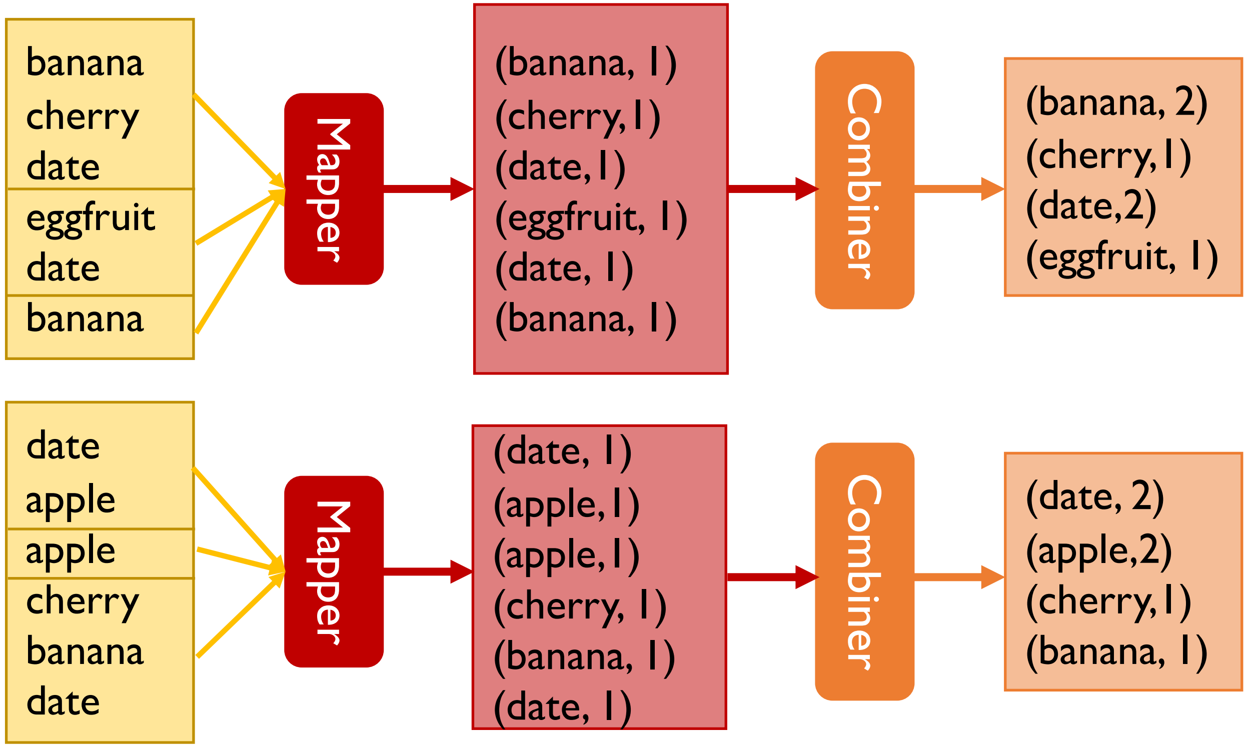
(banana, 1)
(cherry,1)
(date,1)
(eggfruit, 1)
(date, 1)
(banana, 1)

date
apple
apple
cherry
banana
date

Mapper

(date, 1)
(apple,1)
(apple,1)
(cherry, 1)
(banana, 1)
(date, 1)

Combiners



banana
cherry
date
eggfruit
date
banana

Mapper

(banana, 1)
(cherry,1)
(date,1)
(eggfruit, 1)
(date, 1)
(banana, 1)

Combiner

(banana, 2)
(cherry,1)
(date,2)
(eggfruit, 1)

date
apple
apple
cherry
banana
date

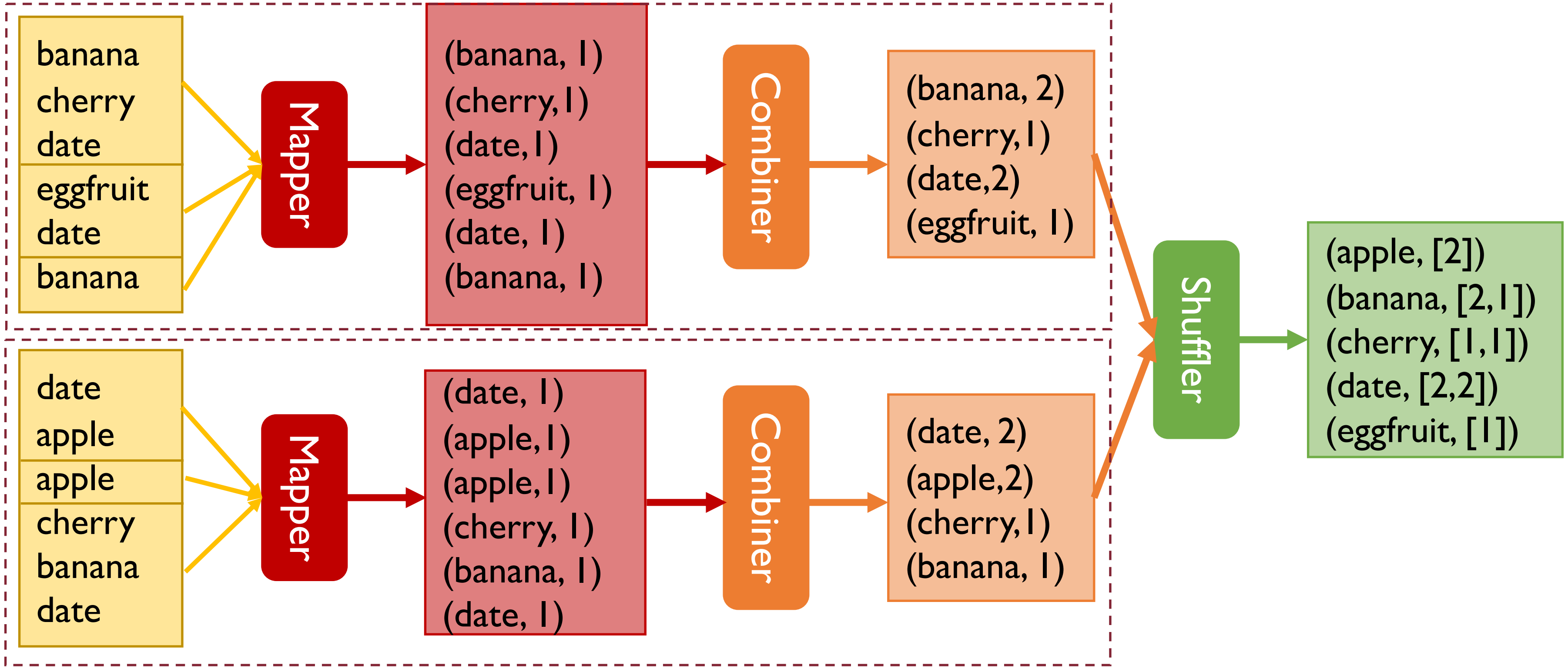
Mapper

(date, 1)
(apple,1)
(apple,1)
(cherry, 1)
(banana, 1)
(date, 1)

Combiner

(date, 2)
(apple,2)
(cherry,1)
(banana, 1)

Combiners



banana
cherry
date
eggfruit
date
banana

(banana, 1)
(cherry,1)
(date,1)
(eggfruit, 1)
(date, 1)
(banana, 1)

(banana, 2)
(cherry,1)
(date,2)
(eggfruit, 1)

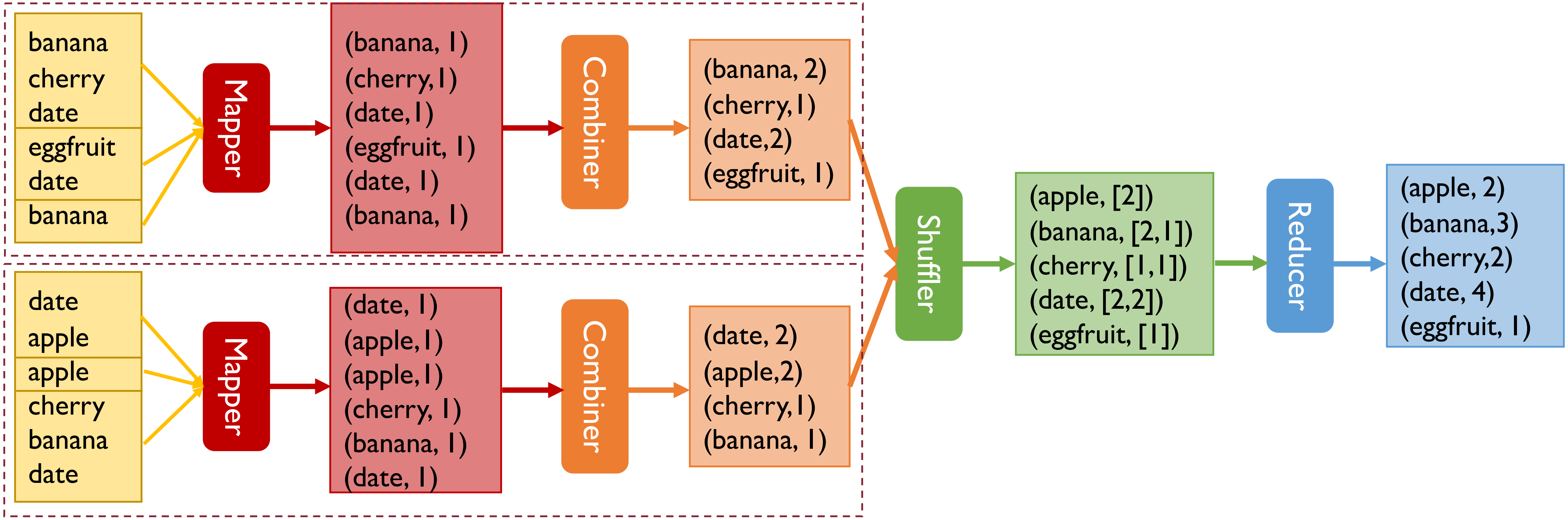
date
apple
apple
cherry
banana
date

(date, 1)
(apple,1)
(apple,1)
(cherry, 1)
(banana, 1)
(date, 1)

(date, 2)
(apple,2)
(cherry,1)
(banana, 1)

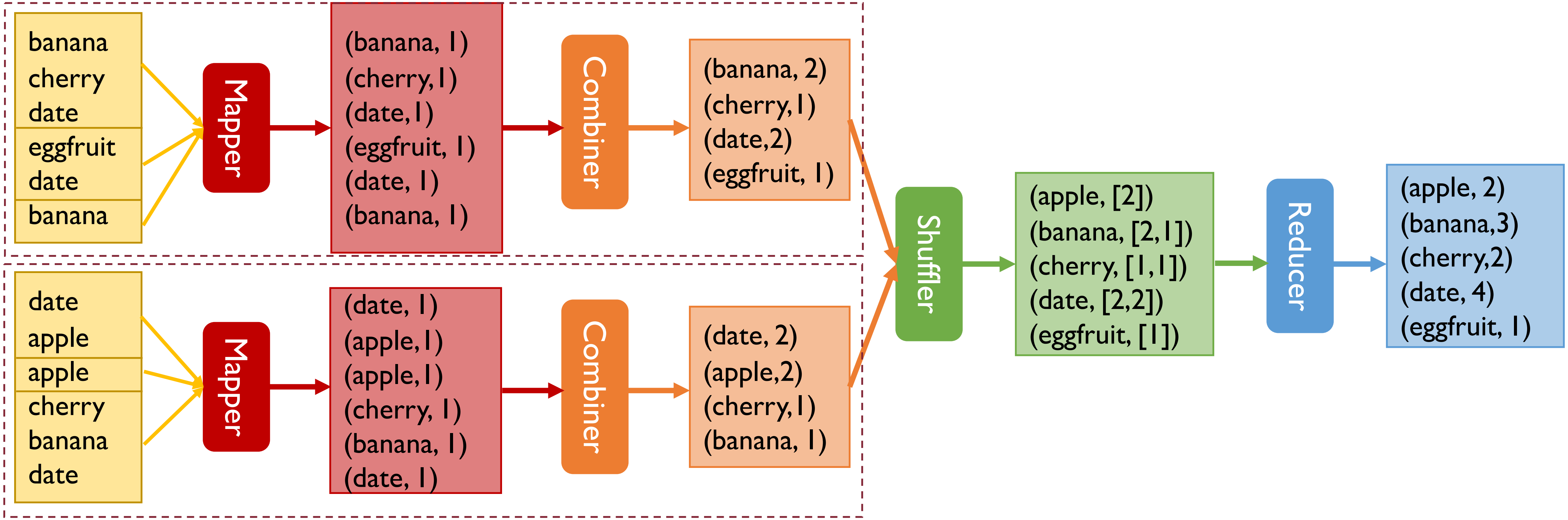
(apple, [2])
(banana, [2,1])
(cherry, [1,1])
(date, [2,2])
(eggfruit, [1])

Combiners


banana
cherry
date
eggfruit
date
banana
Mapper
(banana, 1)
(cherry,1)
(date,1)
(eggfruit, 1)
(date, 1)
(banana, 1)
Combiner
(banana, 2)
(cherry,1)
(date,2)
(eggfruit, 1)
date
apple
apple
cherry
banana
date
Mapper
(date, 1)
(apple,1)
(apple,1)
(cherry, 1)
(banana, 1)
(date, 1)
Combiner
(date, 2)
(apple,2)
(cherry,1)
(banana, 1)
Shuffler
(apple, [2])
(banana, [2,1])
(cherry, [1,1])
(date, [2,2])
(eggfruit, [1])
Reducer
(apple, 2)
(banana,3)
(cherry,2)
(date, 4)
(eggfruit, 1)

Combiners

Combiner combines values associated with the same key yet coming from a single mapper (i.e., 1 mapper : 1 combiner)



Combiners: Drawbacks

- Combiners can be used only on a limited number of situations
- Only when the reduce function is **commutative** and **associative**
  - sum → ok
  - product → ok
  - average → not ok as the local average output by each combiner cannot be used to compute the overall average at the reducer's end

Combiners: Computing Average (Trick)

- Sometimes workarounds exist to take benefit from combiners even if the reduce function is not commutative and associative

Combiners: Computing Average (Trick)

- Sometimes workarounds exist to take benefit from combiners even if the reduce function is not commutative and associative
- Take again the example of the average
  - Instead of letting each combiner output the local average from its own input data
  - Make the combiner output the pair (k_i, (sum_i, $count_i$)) where:
    - sum_i is the sum of the values associated with the key k_i
    - $count_i$ is the total number of values with that key k_i
  - In this way, the reducer can compute the average associated with the key k_i by simply doing $[(sum_i)_1 + \ldots + (sum_i)_m]/[(count_i)_1 + \ldots + (count_i)_m]$

Combiner Trick

- The combiner trick seen before is not applicable to every function
- It works only for those functions which can be expressed as the composition of commutative and associative operators
- There exist functions which cannot be decomposed in such a way (e.g., median)
- When the combiner trick cannot be used, the aggregating function must be computed at the reducer

Partition Function

- Controls how intermediate key-value pairs produced by mappers are distributed across (i.e., sent over) reducers

Partition Function

- Controls how intermediate key-value pairs produced by mappers are distributed across (i.e., sent over) reducers
- Assuming R reducer nodes, default partition function is as simple as

```
hash(key) mod R
```

Partition Function

- Controls how intermediate key-value pairs produced by mappers are distributed across (i.e., sent over) reducers
- Assuming R reducer nodes, default partition function is as simple as

```
hash(key) mod R
```

- Sometimes may be useful to override the default partition function with a custom one

Implementations

- **Google MapReduce**
  - Uses Google File System (GFS) for redundant storage
  - Not available outside Google

Implementations

- **Google MapReduce**
  - Uses Google File System (GFS) for redundant storage
  - Not available outside Google
- **Hadoop**
  - Apache's open-source implementation of MapReduce
  - Uses Hadoop Distributed File System (HDFS)
  - Terminology: Master → NameNode, Chunk Server → DataNode
  - Hive/Pig → SQL-like abstractions on top of Hadoop MapReduce

MapReduce as a Service

- Allows to rent computing by the hour along with other services like persistent storage
- Amazon's "Elastic Computing Cloud" (EC2) provides:
  - Stable Storage (S3)
  - Elastic MapReduce (EMR)

MapReduce: Criticisms

- **2** major **limitations** of MapReduce paradigm:
  - Hard to program directly
    - many problems are not easily described as map-reduce
  - I/O communication bottlenecks cause performance issues
    - persistence to disk slower than in-memory computation

MapReduce: Criticisms

- **2** major **limitations** of MapReduce paradigm:
  - Hard to program directly
    - many problems are not easily described as map-reduce
  - I/O communication bottlenecks cause performance issues
    - persistence to disk slower than in-memory computation
- In short, MapReduce is **not suitable** for large applications composed of several map-reduce steps

Take-Home Message of Today

- MapReduce is a full-fledged framework for distributed computing

Take-Home Message of Today

- MapReduce is a full-fledged framework for distributed computing
- Typical implementations come with a suite of tools/services for reliably storing and processing large volumes of data

Take-Home Message of Today

- MapReduce is a full-fledged framework for distributed computing
- Typical implementations come with a suite of tools/services for reliably storing and processing large volumes of data
- Useful in all those situations where data need to be accessed sequentially

Take-Home Message of Today

- MapReduce is a full-fledged framework for distributed computing
- Typical implementations come with a suite of tools/services for reliably storing and processing large volumes of data
- Useful in all those situations where data need to be accessed sequentially
- May be hard to program and does not support well multiple map-reduce rounds