BIG DATA MANAGEMENT

CE/CZ4123

DISTRIBUTED SYSTEMS AND MAP-REDUCE

MapReduce:
A general distributed paradigm

DISTRIBUTED COMPUTATION PARADIGM

"The issues of how to parallelize the computation, distribute the data, and handle failures conspire to obscure the original simple computation with large amounts of complex code to deal with these issues."

Jeff Dean

Lead of Google AI

PROGRAMING WITH DISTRIBUTED MACHINES

- Programing with distributed machines is hard
- Compared with the codes to run in a single machine, coding for distributed systems requires
  - Handling communication between machines
  - Coping with machine failures
  - Data replicas
  - Data partitioning
  - …

Coding from scratch for distributed Machines is painful, even for simple computation.

People start to develop ideas to ease the process:

- OpenMPI
- Hadoop

Realization of MapReduce

One popular programming model

Enhancement

- Spark
- ...

GOOGLE’S MAPREDUCE

MapReduce is now one of the most widely used programming paradigm in distributed processing.

Hides details such as parallelization, fault-tolerance, data distribution and load balancing

Realize the computation using *map* and *reduce* interfaces

PROGRAMMING MODEL OF MAPREDUCE

map(String key, String value)

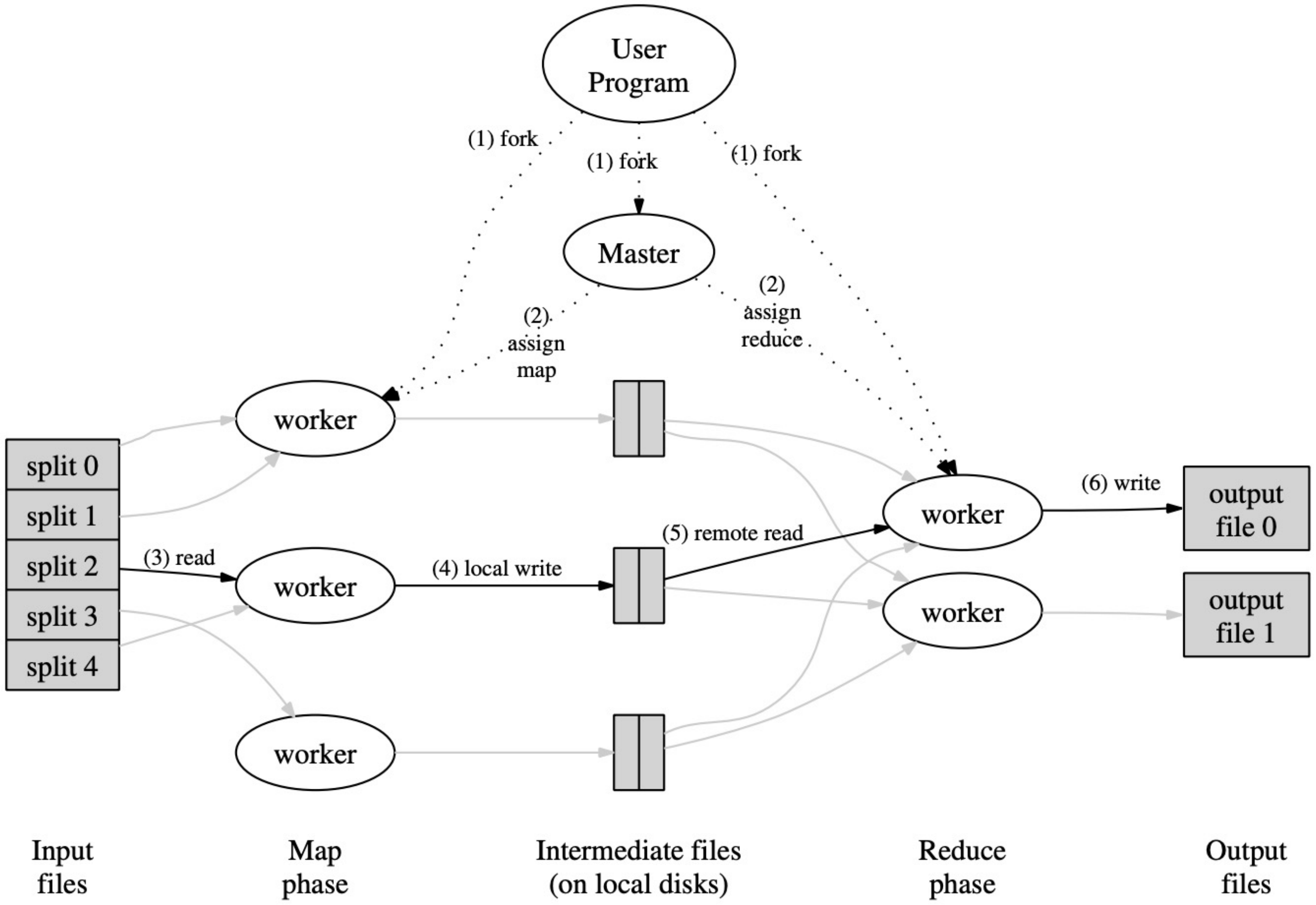
reduce(String key, Iterator values)

map $(k_1,v_1) \rightarrow$ list(k_2,v_2) Defined by you

Same k_2 will be gathered together and passed to the reduce workers (the system does it automatically)

reduce $(k_2,$list$(v_2)) \rightarrow$ list(k_3, v_3)

EXECUTION OVERVIEW



Reading: [MapReduce: Simplified Data Processing on Large Clusters]

EXECUTION OVERVIEW

- The map invocations are distributed across machines by partitioning the input data into M splits (each split: 16-64MB).

One split ⟶ one map task (conducted in one map worker)

↓ ↓

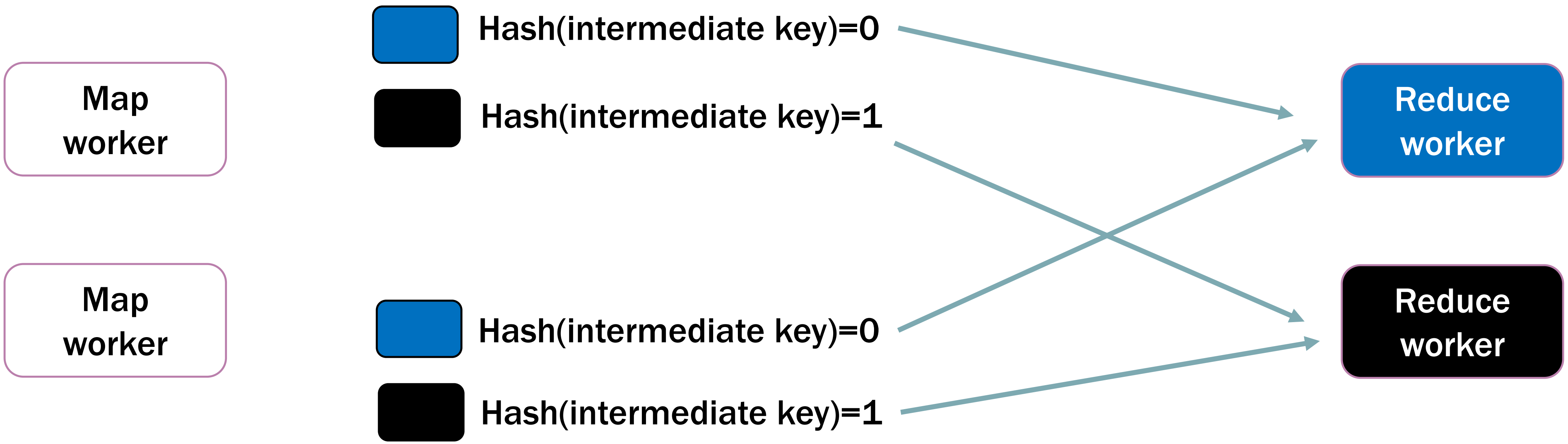
A set of key-value pairs. ⟶ Map function

- The reduce invocations are distributed by partitioning the intermediate key space into R pieces using hash(key) mod R

- There are M map tasks and R reduce tasks.

EXECUTION OVERVIEW

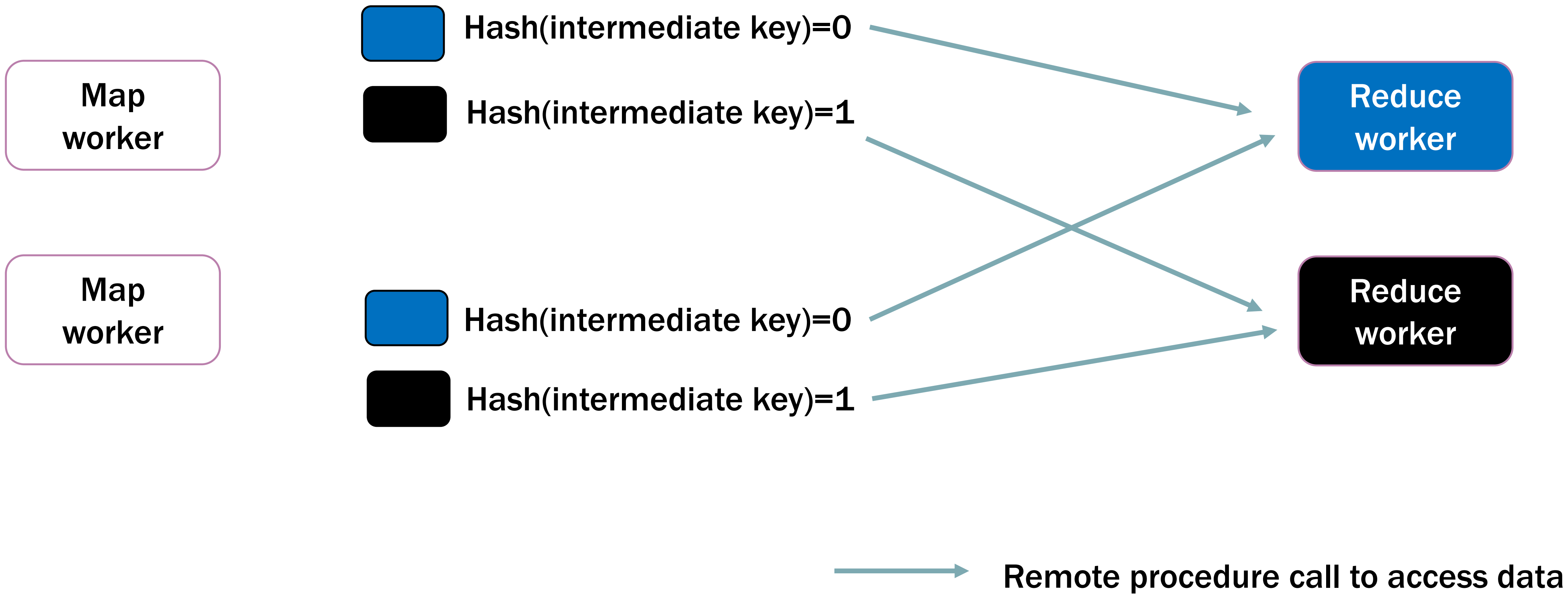
- Intermediate map output is stored locally (in disk). The output is divided into R parts (each for one reduce worker to read)



Remote procedure call to access data

EXECUTION OVERVIEW

- Each reduce worker output results in local files



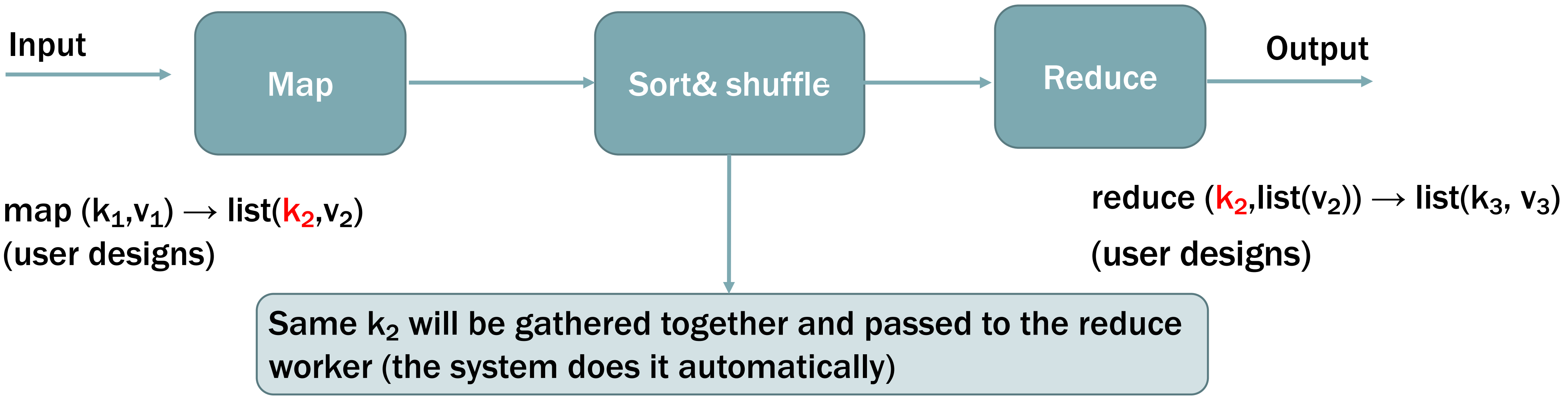
FROM THE PERSPECTIVE OF ALGORITHMIC DESIGNER

map $(k_1,v_1) \longrightarrow$ list(k_2,v_2)

reduce $(k_2,$list$(v_2)) \longrightarrow$ list (k_3, v_3)

Same k_2 will be gathered together and passed to the reduce worker (the system does it automatically)

FROM THE PERSPECTIVE OF ALGORITHMIC DESIGNER



EXAMPLE: COMPUTING FREQUENCIES

(User, product) data

100 Billion items

Alex, bag

Bob, shoes

...

Which item has been sold the most?

MAPREDUCE

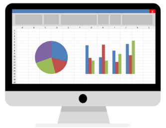
Data sets (100B)

(u1, p1)

(u2, p2)

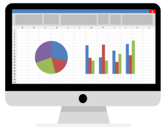
...

A1



A2

...



A100

MAPREDUCE

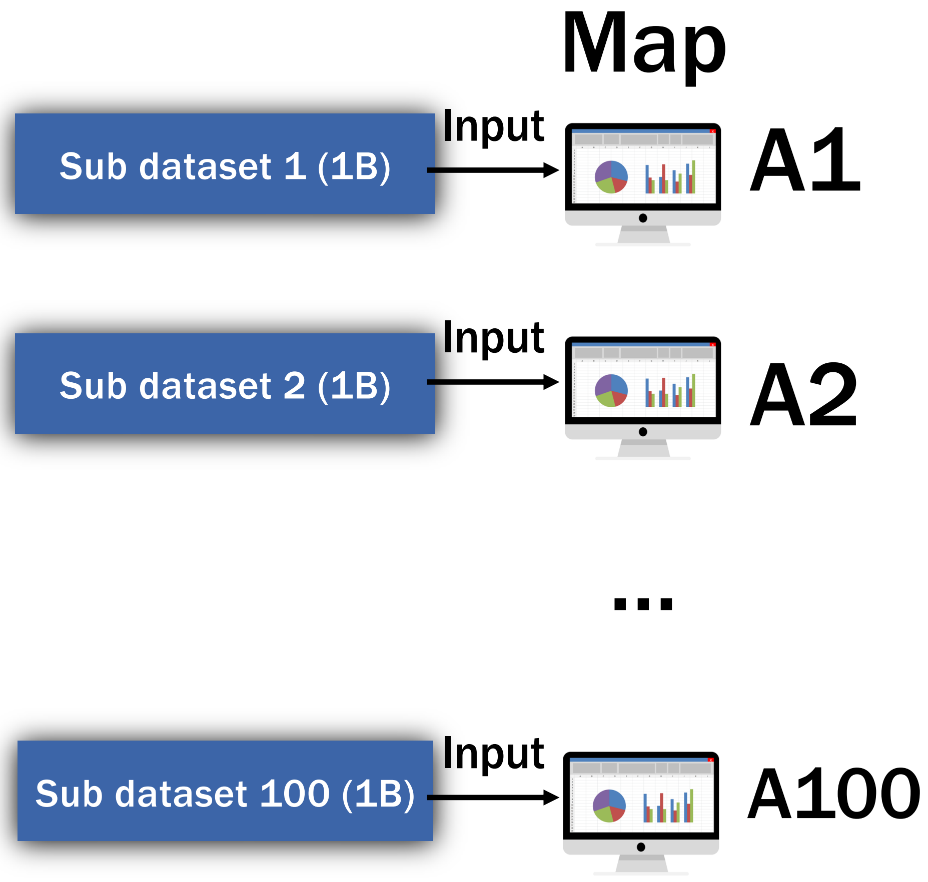
Sub dataset 1 (1B)  A1

Sub dataset 2 (1B)  A2

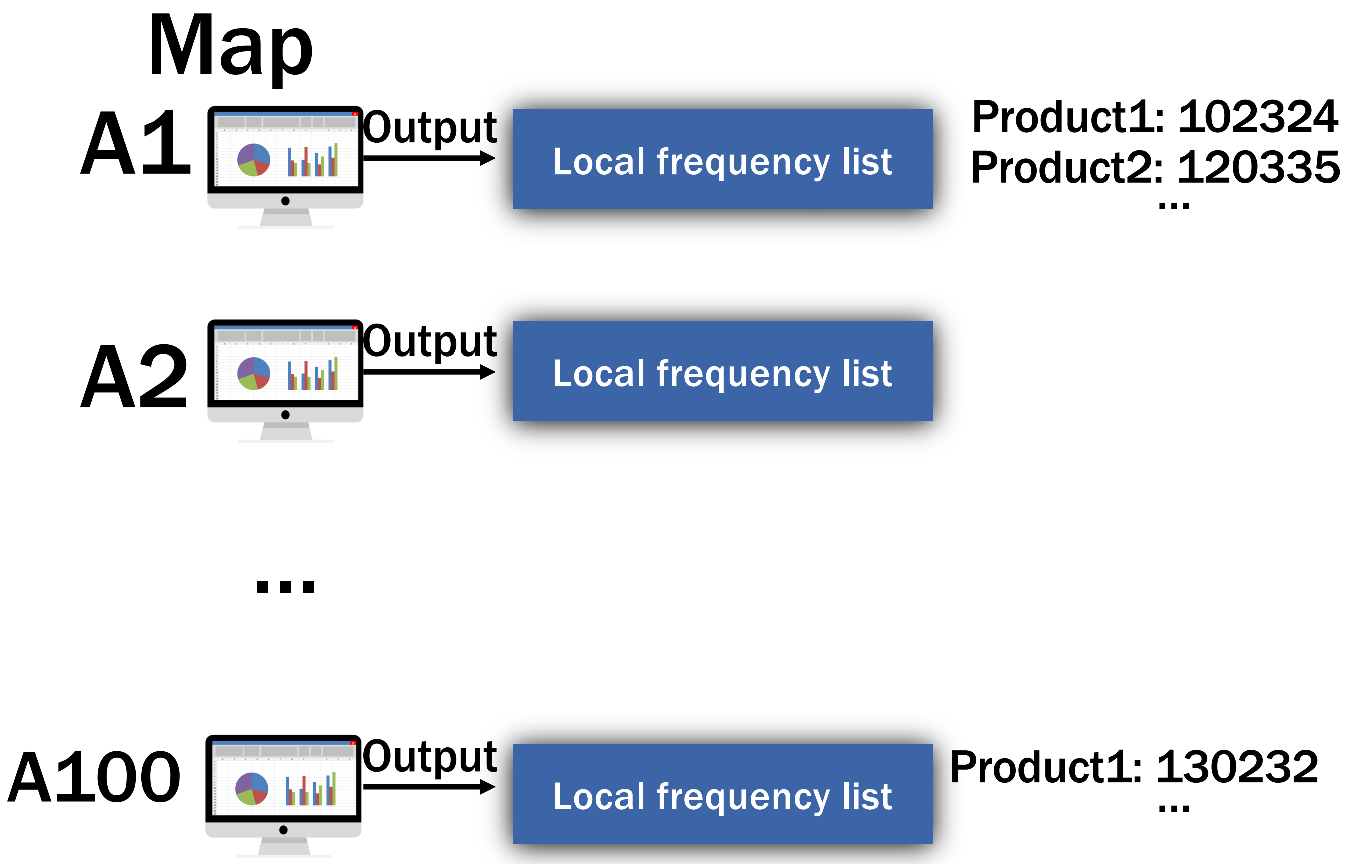
...

Sub dataset 100 (1B)  A100

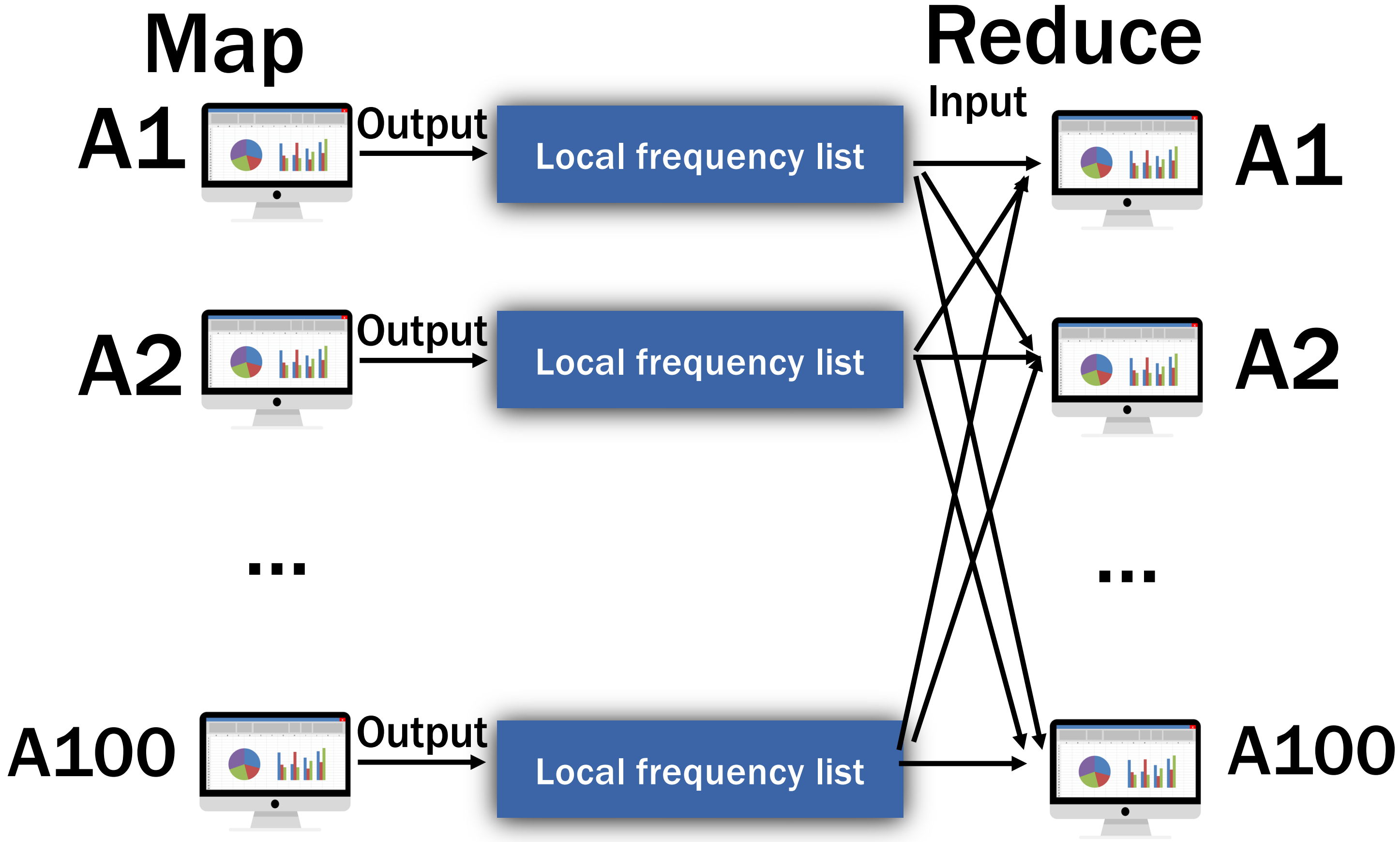
MAPREDUCE


Map
Sub dataset 1 (1B)
Input
A1
Sub dataset 2 (1B)
Input
A2
...
Sub dataset 100 (1B)
Input
A100

MAPREDUCE


Map
A1
Output
Local frequency list
Product1: 102324
Product2: 120335
...
A2
Output
Local frequency list
...
A100
Output
Local frequency list
Product1: 130232
...

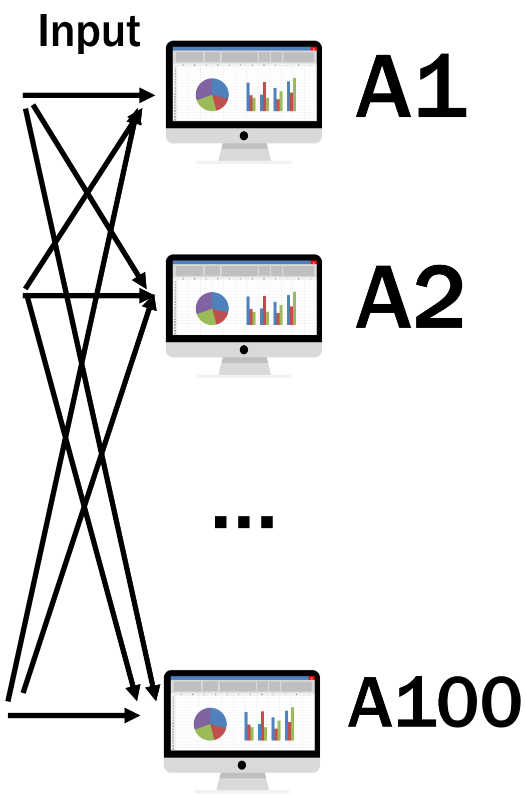
MAPREDUCE



Map
Reduce
A1
Output
Local frequency list
Input
A1
A2
Output
Local frequency list
A2
...
...
A100
Output
Local frequency list
A100

MAPREDUCE

Reduce

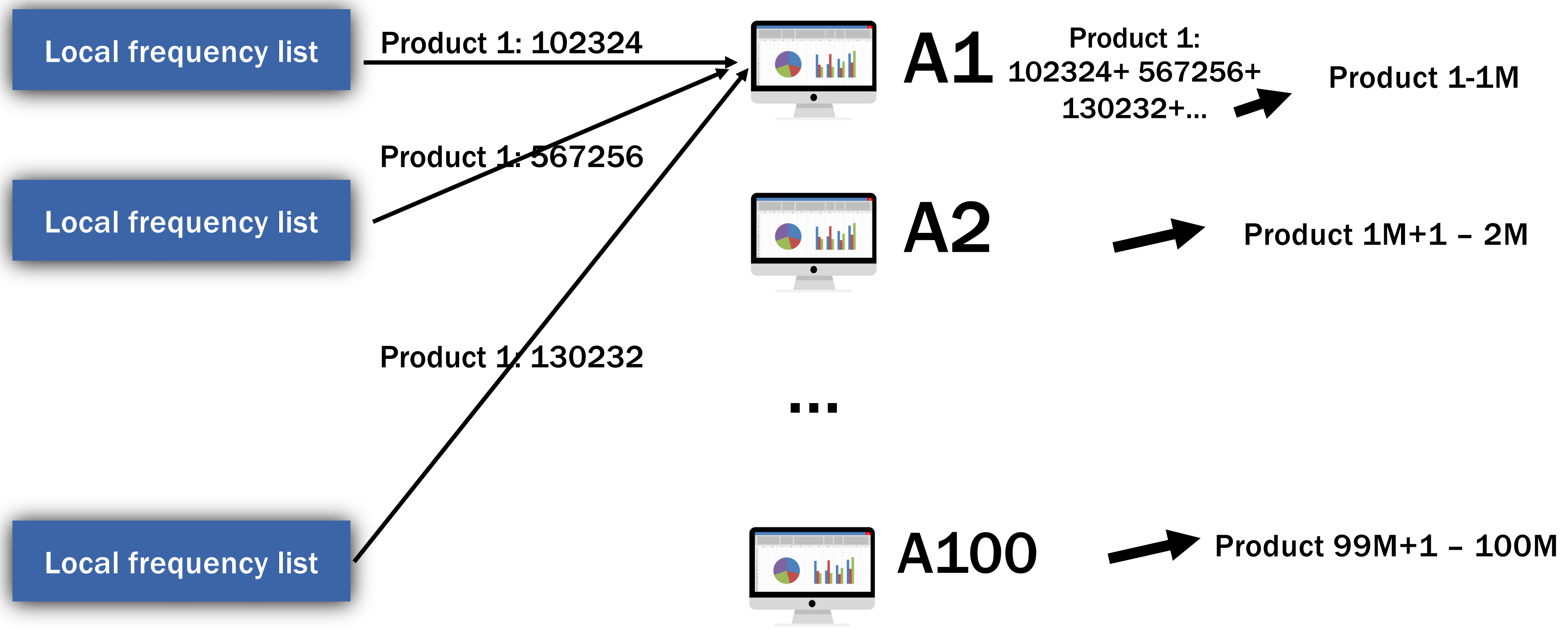


Each reduce worker processes a subset of products.

For example, if there are 100M products, then
A1 handles the Product 1 - 1M,
A2 handles Product 1M+1 – 2M,
...

MAPREDUCE

Reduce



MAPREDUCE PSEUDOCODE

```
map(String key, String value) // key: user // value: product id
{
	Emit-Intermediate(value, "1");
}
```

Note: Map() can be implemented in a simple way that it does not even need to compute the local frequency list for each subset.

MAPREDUCE PSEUDOCODE

```
reduce(String key, Iterator<String> values)
// key: a product
// values: a list of counts
{
        int result = 0;
        for (each v in values){
             result += ParseInt(v);
        }
        Emit(Key, AsString(result));
}
```

INPUT/OUTPUT IN EACH PHASE

Input

(key, value) pairs

Map Output

(key, value) pairs

Input of Reduce

(key, value-list) pairs

Reduce Output

(key, value) pairs

EXAMPLE INPUT/OUTPUT

Input

(Alex, product 1) (Alex, product 2) (Bob, product 1) (Bob, product 3)

Map Output

(key, value) pairs

Input of Reduce

(key, value-list) pairs

Reduce Output

(key, value) pairs

EXAMPLE INPUT/OUTPUT

Input

(“Alex”, “bag001”) (“Alex”, “bag002”) (“Bob”, “bag001”) (“Bob”, “bag003”)

Map Output

(“bag001”,”1”) (“bag002”, “1”) (“bag001”, “1”) (“bag003”, “1”)

Input of Reduce

?

Reduce Output

?

EXAMPLE INPUT/OUTPUT

Input

(“Alex”, “bag001”) (“Alex”, “bag002”) (“Bob”, “bag001”) (“Bob”, “bag003”)

Map Output

(“bag001”,”1”) (“bag002”, “1”) (“bag001”, “1”) (“bag003”, “1”)

Input of Reduce

?

Reduce Output

?

EXAMPLE INPUT/OUTPUT

Input

(“Alex”, “bag001”) (“Alex”, “bag002”) (“Bob”, “bag001”) (“Bob”, “bag003”)

Map Output

(“bag001”,”1”) (“bag002”, “1”) (“bag001”, “1”) (“bag003”, “1”)

Input of Reduce

(“bag001”,{“1”, “1”}) (“bag002”, “1”) (“bag003”, “1”)

Reduce Output

?

EXAMPLE INPUT/OUTPUT

Input

("Alex", "bag001") ("Alex", "bag002") ("Bob", "bag001") ("Bob", "bag003")

Map Output

("bag001","1") ("bag002", "1") ("bag001", "1") ("bag003", "1")

Input of Reduce

("bag001",{"1", "1"}) ("bag002", "1") ("bag003", "1")

Reduce Output

("bag001","2") ("bag002", "1") ("bag003", "1")

ANOTHER EXAMPLE INPUT/OUTPUT

Input

(“A”, “001”) (“A”, “001”) (“A”, “002”) (“A”, “003”) (“B”, “004”)

Map Output

(“001”,”1”) (“001”, “1”) (“002”, “1”) (“003”, “1”) (“004”, “1”)

Input of Reduce

(“001”, {“1”, “1”}) (“002”, “1”) (“003”, “1”) (“004”, “1”)

Reduce Output

(“001”,”2”) (“002”, “1”) (“003”, “1”) (“004”, “1”)

EXAMPLE: INVERTED INDEX

Doc 1

NTU's 30th anniversary marks a history of sparking innovative ideas and addressing global challenges. Since its inauguration in 1991, NTU has been home to an exceptional community of faculty, students and staff who have helped propel the university to excellence. To celebrate our birthday year we have created a digital time capsule that contains a collection of 30 past and present memories contributed by members of the NTU community. Sealed on 9 December 2021 to commemorate NTU's momentous achievements, the time capsule will be opened in 2041 at NTU's Golden Jubilee. View the artefacts at NTU's 30th anniversary exhibition, which is now open to public, or visit the time capsule website to learn more about the university's milestone moments.

EXAMPLE: INVERTED INDEX

Doc 2

The NTU School of Computer Science and Engineering (SCSE) was established in November 1988 as the School of Applied Science (SAS) offering Bachelor Degree in Computer Technology, the first of its kind in Singapore. We were part of NTU's predecessor, Nanyang Technological Institute (NTI) that was set up in August 1981 with a charter to train three-quarters of Singapore's engineers. Within 4 years of operation, NTI was singled out as one of the best engineering institutions in the world by the Commonwealth Engineering Council. The Council reached this verdict after an extensive 4-year study of the courses offered by engineering institutions worldwide.

A TYPICAL QUERY

Given a word, find out which documents contain that word.

Example:

“NTU”: Doc 1, Doc 2

“August”: Doc 2

“achievements”: Doc 1

A TYPICAL QUERY

Given a word, find out which documents contain that word.

Example:

"NTU": Doc 1, Doc 2

"August": Doc 2

"achievements": Doc 1

Extremely inefficient!

A TYPICAL QUERY

Given a word, find out which documents contain that word.

How about precompute all the document lists for each possible word in the corpus?

This is called inverted index.

EXAMPLE: INVERTED INDEX

Suppose we have a list of documents, each of which contains a set of words. We need to construct the inverted index, such that given a word, we can directly extract the list of documents (by Ids) containing the word. Please describe the algorithm using the MapReduce model.

Example input:

(“doc1”, “I study at NTU”)

(“doc2”, “NTU is famous”)

Example output:

(“I”, {“doc1”}), (“study”, {“doc1”}), (“at”, {”doc1”}), (“NTU”, {“doc1, doc2”}), (“is”, {“doc2”}), (“famous”, {“doc2”})

DESIGN: THE INPUT TO MAP

Key: document ID (string)

Value: document content (string)

DESIGN: MAP FUNCTION

Key: document ID (string)

Value: document content (string)

```
map(String docID, String docs): {
{
        Iterator<string> words = split-to-words(docs);
        for(string word in words){
            Emit-Intermediate(word, docID);
        }
}
```

DESIGN: REDUCE FUNCTION

Key: document ID (string)

Value: document content (string)

```
reduce(String word, Iterator<String> docIDs):
// key: a word // values: a list of document ids
{
        Emit(word, ToString(docIDs));
}
```

EXAMPLE INPUT/OUTPUT

Input

(“Doc1”, “Big data management is a course.”)

(“Doc2”, “Introduction to databases is a course.”)

Map Output

?

Input of Reduce

?

Reduce Output

?

EXAMPLE INPUT/OUTPUT

Input

(“Doc1”, “Big data management is a course.”)

(“Doc2”, “Introduction to databases is a course.”)

Map Output

(“Big”, “Doc1”) (“data”, “Doc1”) (“management”, “Doc1”) (“is”, “Doc1”) (“a”, “Doc1”) (“course”, “Doc1”) (“Introduction”, “Doc2”) (“to”, “Doc2”) (“databases”, “Doc2”) (“is”, “Doc2”) (“a”, “Doc2”) (“course”, “Doc2”)

Input of Reduce

?

Reduce Output

?

EXAMPLE INPUT/OUTPUT

Input

(“Doc1”, “Big data management is a course.”)

(“Doc2”, “Introduction to databases is a course.”)

Map Output

(“Big”, “Doc1”) (“data”, “Doc1”) (“management”, “Doc1”) (“is”, “Doc1”) (“a”, “Doc1”) (“course”, “Doc1”) (“Introduction”, “Doc2”) (“to”, “Doc2”) (“databases”, “Doc2”) (“is”, “Doc2”) (“a”, “Doc2”) (“course”, “Doc2”)

Input of Reduce

(“Big”, {“Doc1”}) (“data”, {“Doc1”}) (“management”, {“Doc1”}) (“is”, {“Doc1”, “Doc2”}) (“a”, {“Doc1”, “Doc2”}) (“course”, {“Doc1”, “Doc2”})

Reduce Output

?

EXAMPLE INPUT/OUTPUT

Input

(“Doc1”, “Big data management is a course.”)

(“Doc2”, “Introduction to databases is a course.”)

Map Output

(“Big”, “Doc1”) (“data”, “Doc1”) (“management”, “Doc1”) (“is”, “Doc1”) (“a”, “Doc1”) (“course”, “Doc1”) (“Introduction”, “Doc2”) (“to”, “Doc2”) (“databases”, “Doc2”) (“is”, “Doc2”) (“a”, “Doc2”) (“course”, “Doc2”)

Input of Reduce

(“Big”, {“Doc1”}) (“data”, {“Doc1”}) (“management”, {“Doc1”}) (“is”, {“Doc1”, “Doc2”}) (“a”, {“Doc1”, “Doc2”}) (“course”, {“Doc1”, “Doc2”})

Reduce Output

(“Big”, “Doc1”) (“data”, “Doc1”) (“management”, “Doc1”) (“is”, “Doc1;Doc2”) (“a”, “Doc1;Doc2”) (“course”, “Doc1;Doc2”)

Looks like most of the tasks can be simply finished using one MapReduce job

There are more complicated cases where we need multiple jobs