DASCI

Instituto Andaluz Interuniversitario
en Ciencia de Datos e
Inteligencia Computacional

Ciencia de Datos a través del Big Data

Diego García (djgarcia@ugr.es)
Isaac Triguero (isaaktriguero@ugr.es)

Financiado por
la Unión Europea
NextGenerationEU

GOBIERNO
DE ESPAÑA

MINISTERIO
PARA LA TRANSFORMACIÓN DIGITAL
Y DE LA FUNCIÓN PÚBLICA

SECRETARÍA DE ESTADO
DE DIGITALIZACIÓN
E INTELIGENCIA ARTIFICIAL

Plan de
Recuperación,
Transformación
y Resiliencia

UNIVERSIDAD
DE GRANADA

UNIMORE
UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

REPSOL

CAMBRIDGE UNIVERSITY PRESS

Chapter 2

MapReduce I

Learning Outcomes

- Learn a functional programming paradigm for distributed computing with Big Data [KU]
- Understand why transparency and fault-tolerance are desirable in Big Data [KU]
- How to use map reduce in practice for problem solving [IS]
- How to exploit map reduce to preserve data locality principles, minimizing data movement [IS]

Recap

- Big data – Mining and Analytics
- Objective: apply operation(s) on all data
- The principle of data locality
- Move computation is cheaper than moving data and computation!

Recap: Distributed Systems in Big Data

- We will need a new programming model: **MapReduce**
  - *"Moving computation is cheaper than moving computation and data at the same time"*
- **Idea**
  - Data is distributed among nodes (distributed file system)
  - Functions/operations to process data are distributed to all the computing nodes
  - Each computing node works with the data stored in it
  - Only the necessary data is moved across the network

Objective for today

- What is MapReduce?
- Basic functions: Map and Reduce (first in Python)
- How do we split the work across computers?

What is MapReduce?

- Parallel Programming model
- **Divide & conquer strategy**
  - **divide**: partition dataset into smaller, independent chunks to be processed in parallel (map)
  - **conquer**: combine, merge or otherwise aggregate the results from the previous step (reduce)
- Based on **simplicity** and **transparency** to the programmers, and assumes **data locality**
- Became popular thanks to the open-source project Hadoop! (Used by Google, Facebook, Amazon, …)

Ghemawat, S., Gobioff, H., and Leung, S.-T. 2003. The Google file system. Pages 29–43 of: *Proceedings of the nineteenth ACM symposium on Operating systems principles*. SOSP '03. New York, NY, USA: ACM. Link
Dean, Jeffrey, and Ghemawat, Sanjay. 2004. MapReduce: Simplified Data Processing on Large Clusters. Pages 137–150 of: *OSDI'04: Sixth Symposium on Operating System Design and Implementation*. Link

Map and Reduce functions in Python

- Functional programming in Python
- Higher-order functions
  - Receive other functions as parameters

Map and Reduce functions in Python

- map function
- Exercises:
  - Create a list and square its elements
  - Do the same creating a function named square

Map and Reduce functions in Python

CAMBRIDGE UNIVERSITY PRESS

- `map` function

```
lst = [1, 2, 3, 4]


list(map(lambda x: x*x, lst))
```

```
[1, 4, 9, 16]
```

Map and Reduce functions in Python

- `map` function

```
lst = [1, 2, 3, 4]

list(map(lambda x: x*x, lst))
```

```
[1, 4, 9, 16]
```

```
def square(x):
    return x*x
list(map(square, lst))
```

```
[1, 4, 9, 16]
```

Map and Reduce functions in Python

- reduce function
- Belongs to the *functools* module
- Exercises:
  - Add all elements from the previous list
  - Do the same creating a function named *add_reduce*

The reduce function must be **commutative** and **associative**

Map and Reduce functions in Python

CAMBRIDGE UNIVERSITY PRESS

- reduce function

```
from functools import reduce

reduce(lambda x, y: x + y, lst)
```

```
10
```

The reduce function must be **commutative** and **associative**

Map and Reduce functions in Python

CAMBRIDGE UNIVERSITY PRESS

- reduce function

```
from functools import reduce

reduce(lambda x, y: x + y, lst)
```

```
10
```

```
def add_reduce(x, y):
    out = x + y
    print(f"{x}+{y}-->{out}")
    return out
reduce(add_reduce,lst)
```

```
1+2-->3
3+3-->6
6+4-->10
```

```
10
```

The reduce function must be **commutative** and **associative**

Map and Reduce functions in Python

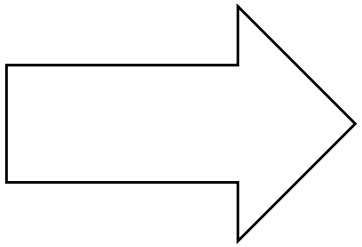
- map & reduce functions
- Exercises:
  - Given a list of names:
    1. Make them uppercase (classic and MapReduce solution)
    2. Add “Hello, --” to each of them
    3. Calculate the length of each of them
    4. Calculate how many characters there are in total
    5. Count the number of times the letter “a” appears
  - Given a list of 6 decimal numbers (positive and negative):
    1. Round them to 1 6 decimals
    2. Calculate their absolute values
    3. Add all of them together
    4. Calculate the maximum and minimum

The reduce function must be **commutative** and **associative**

'Hello World' in Big Data: Word Count

Task: count the word frequency in a document.

```
"Welcome to the
Big World of Big
Big Data Welcome
World bye
World Hello
MapReduce GoodBye
MapReduce
This Book on Big
Data is fun"
```



```
{'Welcome': 2,
 'to': 1,
 'the': 1,
 'Big': 4,
 'World': 3,
 'of': 1,
 'Data': 2,
 'bye': 1,
 'Hello': 1,
 'MapReduce': 2,
 …}
```

Example inspired by BerkeleyX: CS110x Big Data Analysis with Apache Spark

'Hello World' in Big Data: Word Count

Using **hash tables or a dictionary?**

```
"Welcome to the
Big World of Big
Big Data Welcome
World bye
World Hello
MapReduce GoodBye
MapReduce
This Book on Big
Data is fun"
```

```
'Welcome': 1
```

'Hello World' in Big Data: Word Count

Using **hash tables or a dictionary?**

```
“Welcome to the
Big World of Big
Big Data Welcome
World bye
World Hello
MapReduce GoodBye
MapReduce
This Book on Big
Data is fun”
```

```
'Welcome’: 1,
 'to': 1
```

'Hello World' in Big Data: Word Count

Using **hash tables or a dictionary?**

```
“Welcome to the
Big World of Big
Big Data Welcome
World bye
World Hello
MapReduce GoodBye
MapReduce
This Book on Big
Data is fun”
```

```
'Welcome’: 1,
 'to': 1,
 'the': 1
```

'Hello World' in Big Data: Word Count

CAMBRIDGE UNIVERSITY PRESS

Using **hash tables or a dictionary?**

```
text = "Welcome to the Big World of Big
Big Data Welcome World bye\
World Hello MapReduce GoodBye MapReduce\
This Book on Big Data is fun"
```

'Hello World' in Big Data: Word Count

Using **hash tables or a dictionary?**

```
text = "Welcome to the Big World of Big
Big Data Welcome World bye\
World Hello MapReduce GoodBye MapReduce\
This Book on Big Data is fun"
```

Code your solution!

'Hello World' in Big Data: Word Count

Using **hash tables or a dictionary?**

```
text = "Welcome to the Big World of Big
Big Data Welcome World bye\
World Hello MapReduce GoodBye MapReduce\
This Book on Big Data is fun"

dic = {}
for word in text.split(" "):
    if word not in dic.keys():
        dic[word]=1
    else:
        dic[word]+=1
```

'Hello World' in Big Data: Word Count

Using **hash tables or a dictionary?**

```
text = "Welcome to the Big World of Big
Big Data Welcome World bye\
World Hello MapReduce GoodBye MapReduce\
This Book on Big Data is fun"

dic = {}
for word in text.split(" "):
    if word not in dic.keys():
        dic[word]=1
    else:
        dic[word]+=1
```

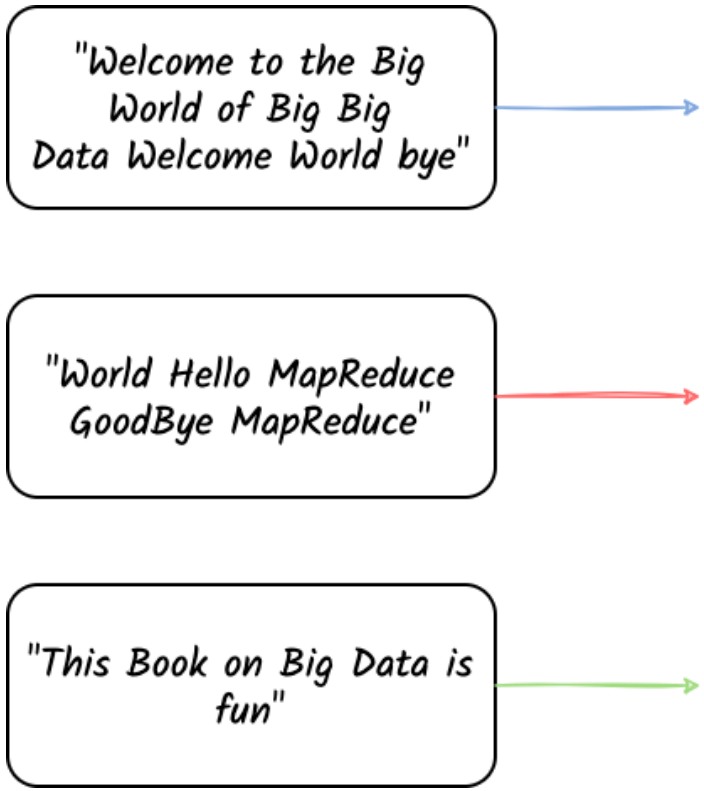
A solution in Python

```
{'Welcome': 2, 'to': 1,
 'the': 1, 'Big': 4,
 'World': 3, 'of': 1,
 'Data': 2, 'bye': 1,
 'Hello': 1, MapReduce':
2, 'GoodBye': 1, 'This':
1, 'Book': 1, 'on': 1,
 'is': 1, 'fun': 1}
```

'Hello World' in Big Data: Word Count

- Let's try with a bigger file:
  - https://mattmahoney.net/dc/enwik8.zip

What if the document is really BIG?



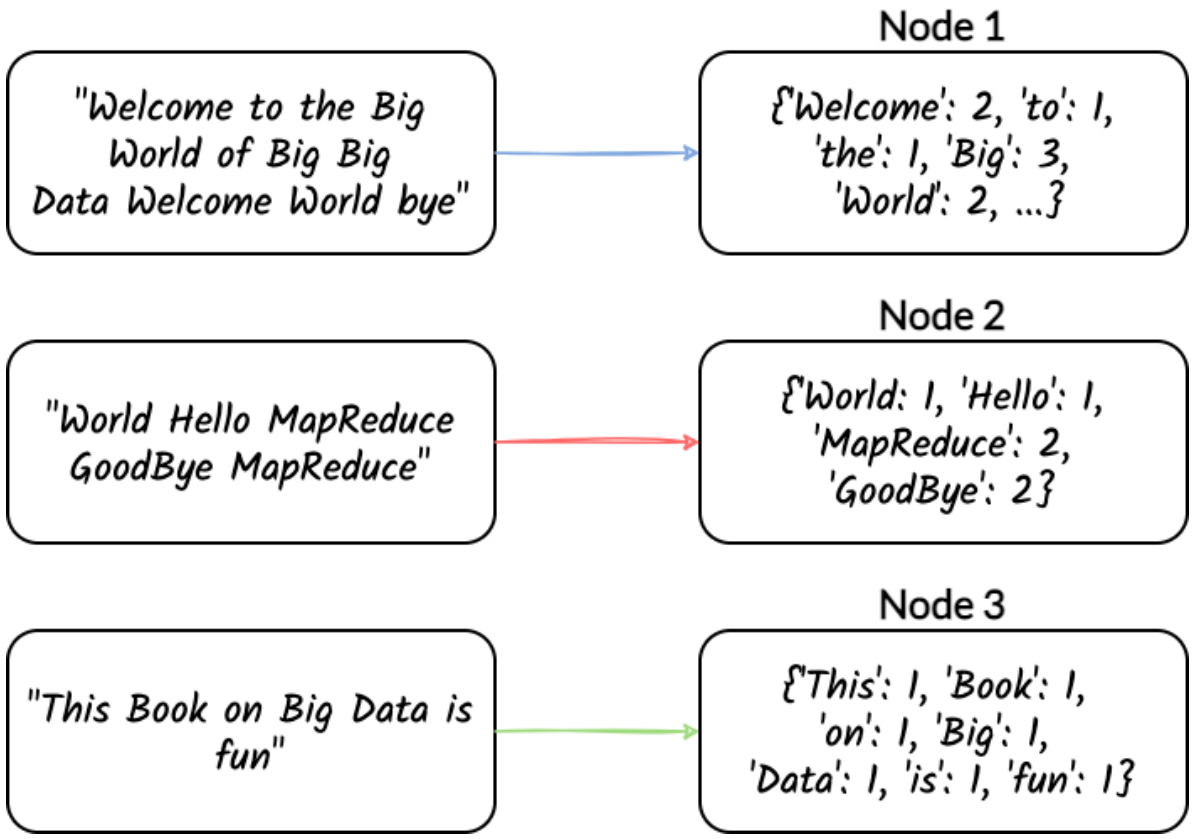
Let's parallelise this!

Divide

Step 1: Split the data into different chunks

'Hello World' in Big Data: Word Count

What if the document is really BIG?



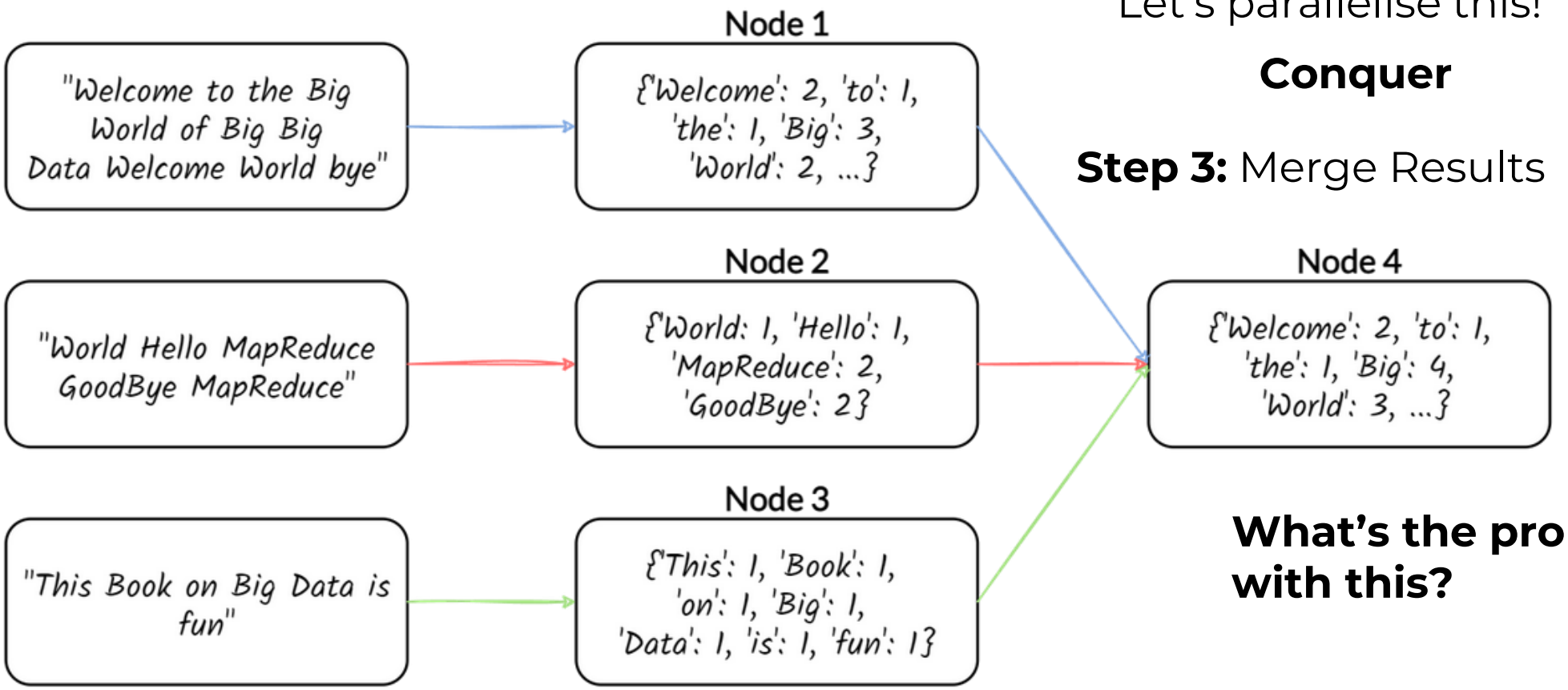
Let's parallelise this!

Divide

Step 2: Each machine could handle one of those chunks

'Hello World' in Big Data: Word Count

What if the document is really BIG?



Let's parallelise this!

Conquer

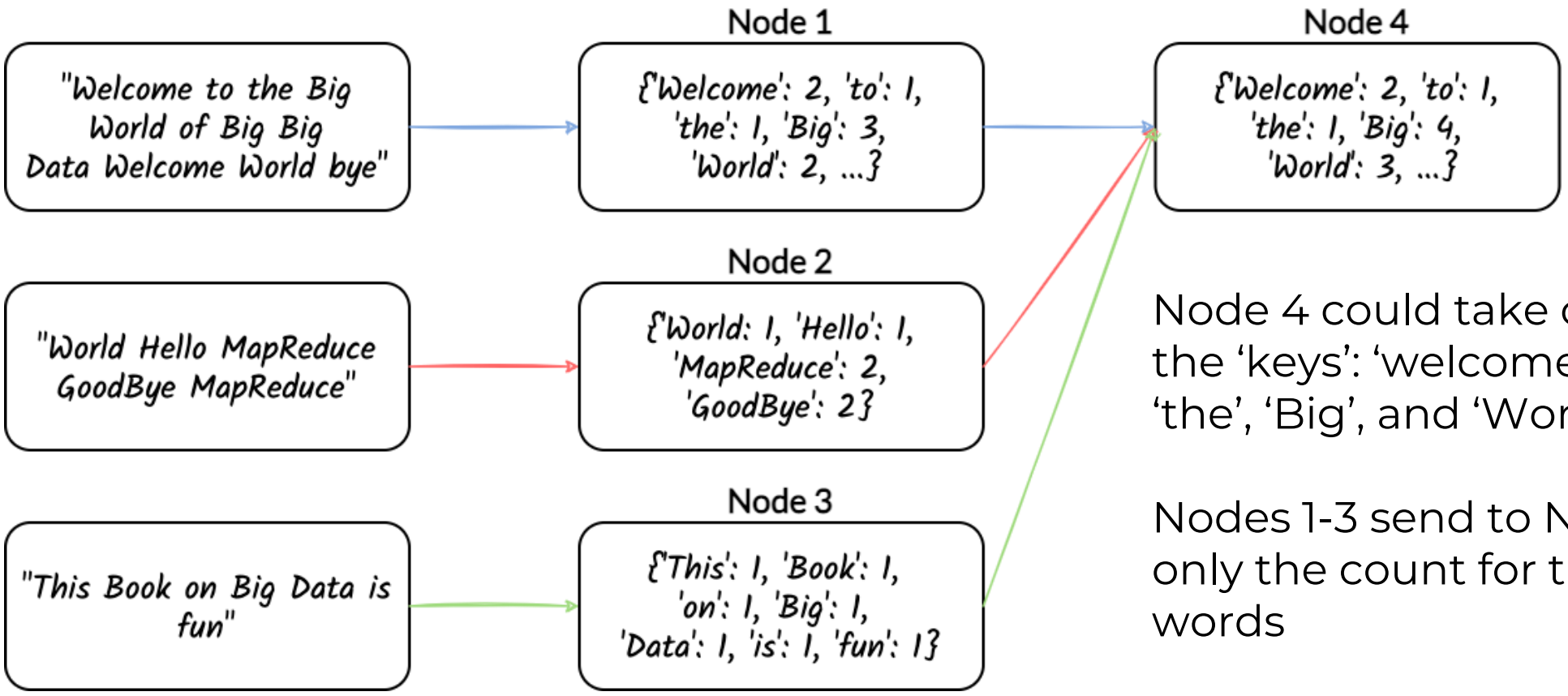
Step 3: Merge Results

What's the problem with this?

'Hello World' in Big Data: Word Count

What if the document is really BIG?

Let's 'MapReduce' this!



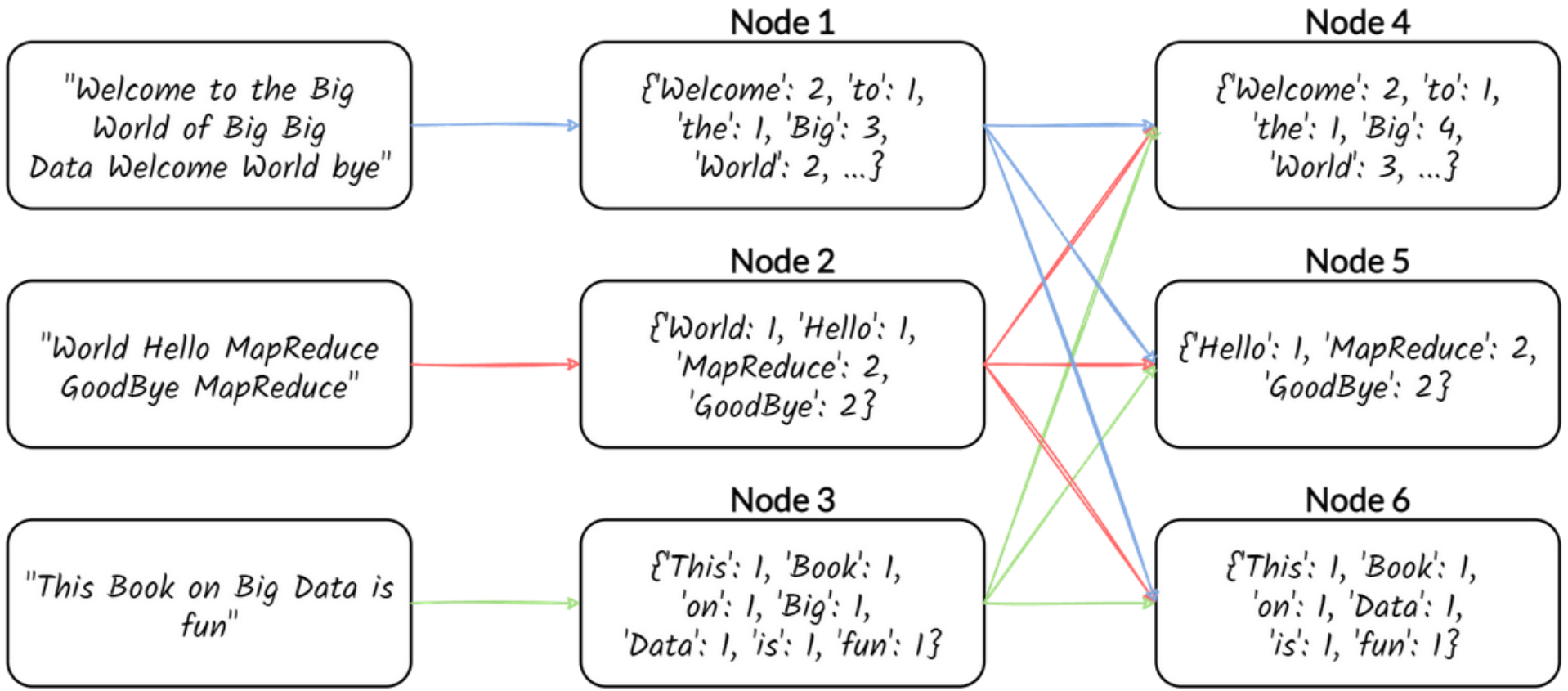
Node 4 could take care of the 'keys': 'welcome', 'to', 'the', 'Big', and 'World'.

Nodes 1-3 send to Node 4 only the count for these words

'Hello World' in Big Data: Word Count

What if the document is really BIG?

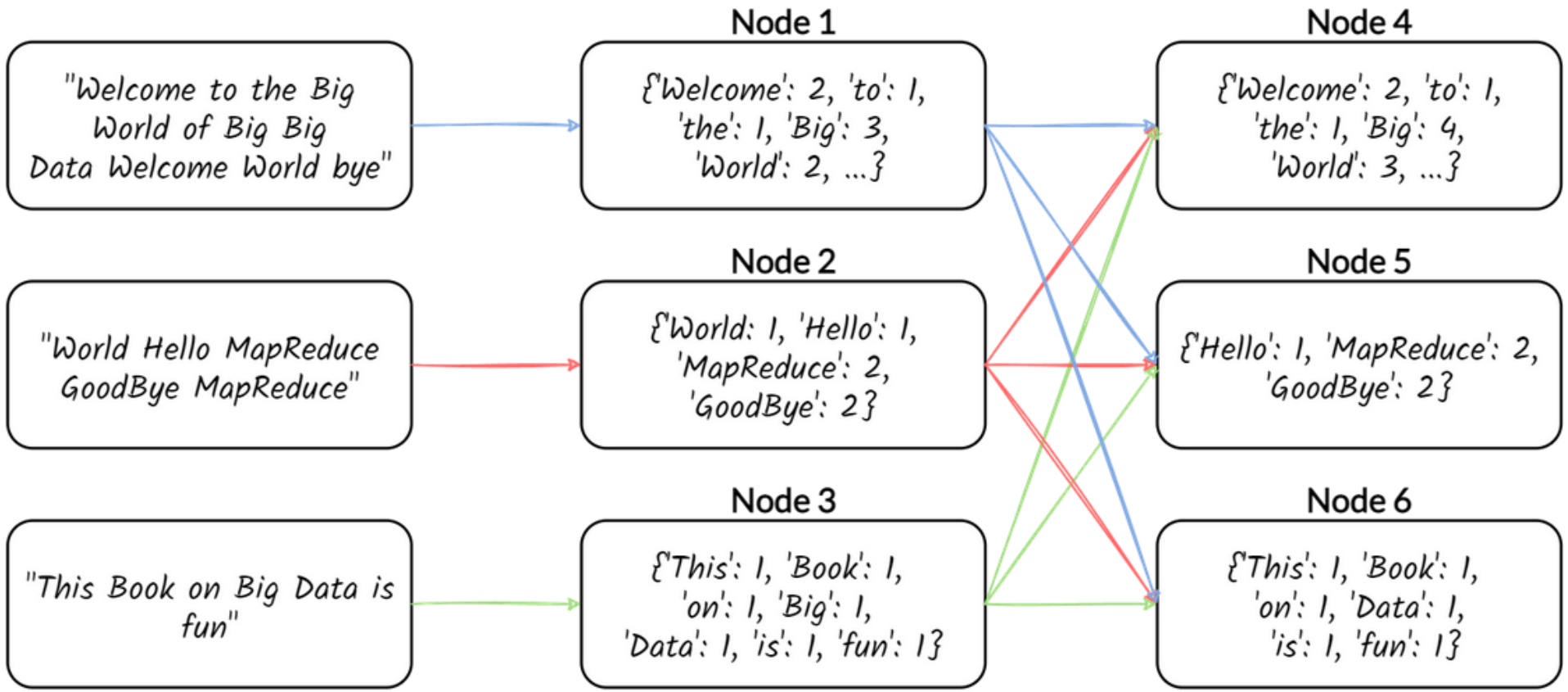
Let's 'MapReduce' this!



'Hello World' in Big Data: Word Count

What if the document is really BIG?

Let's 'MapReduce' this!

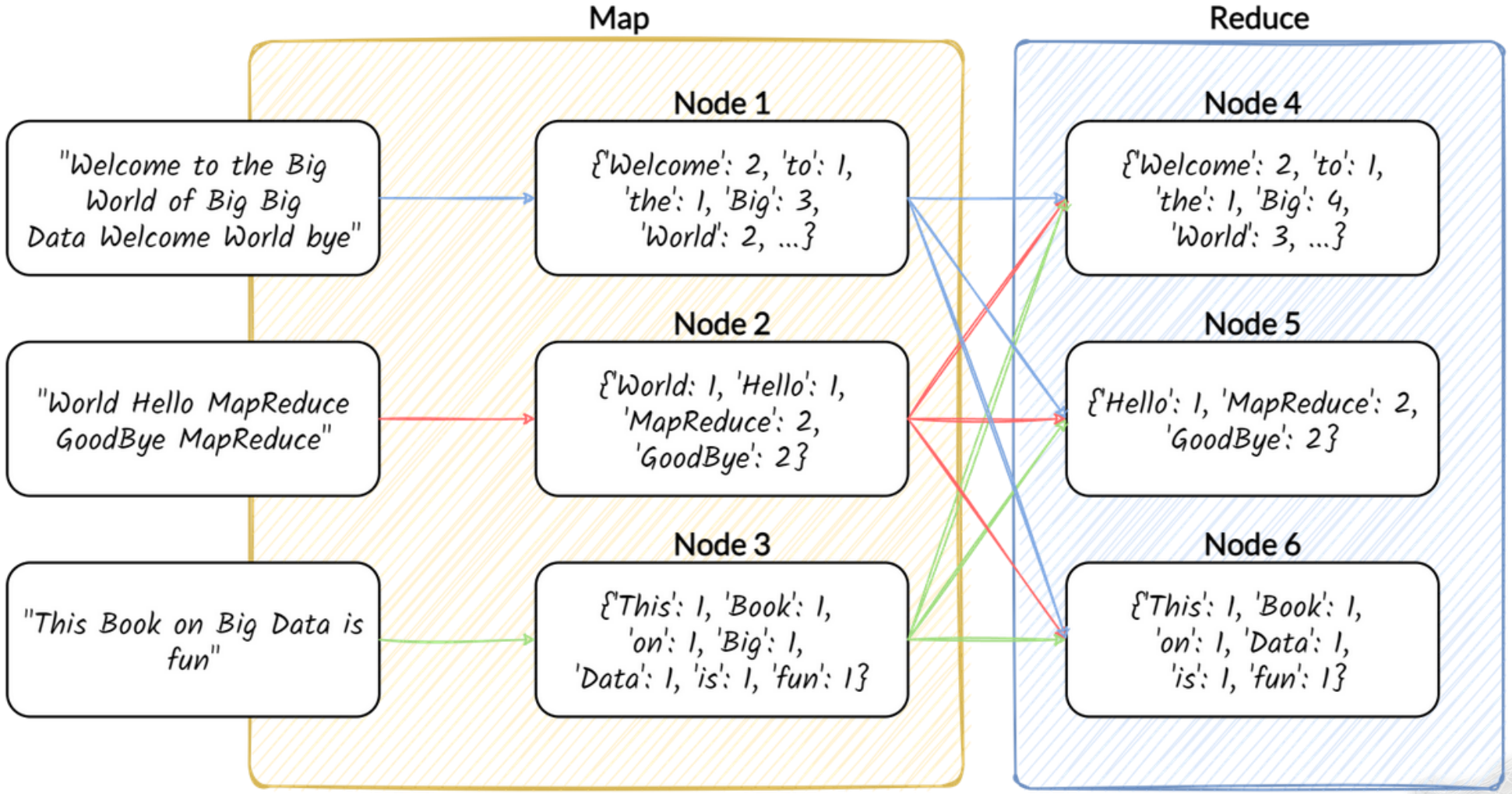


We could use the same nodes!

'Hello World' in Big Data: Word Count

What if the document is really BIG?

Let's 'MapReduce' this!



Challenge #1

- How would you create a ranking of the words to determine which word is the most frequently used?

Challenge #1

- How would you create a ranking of the words to determine which word is the most frequently used?

- For example:
  - *Node1* would keep World, Hello, and MapReduce because they are the most popular ones (3,2 and 2 repetitions, respectively)
  - *Nodes 2* and *3* would keep the other ones... or maybe we don't even need a 3 node in this case.

Challenge #1

- How would you create a ranking of the words to determine which word is the most frequently used?

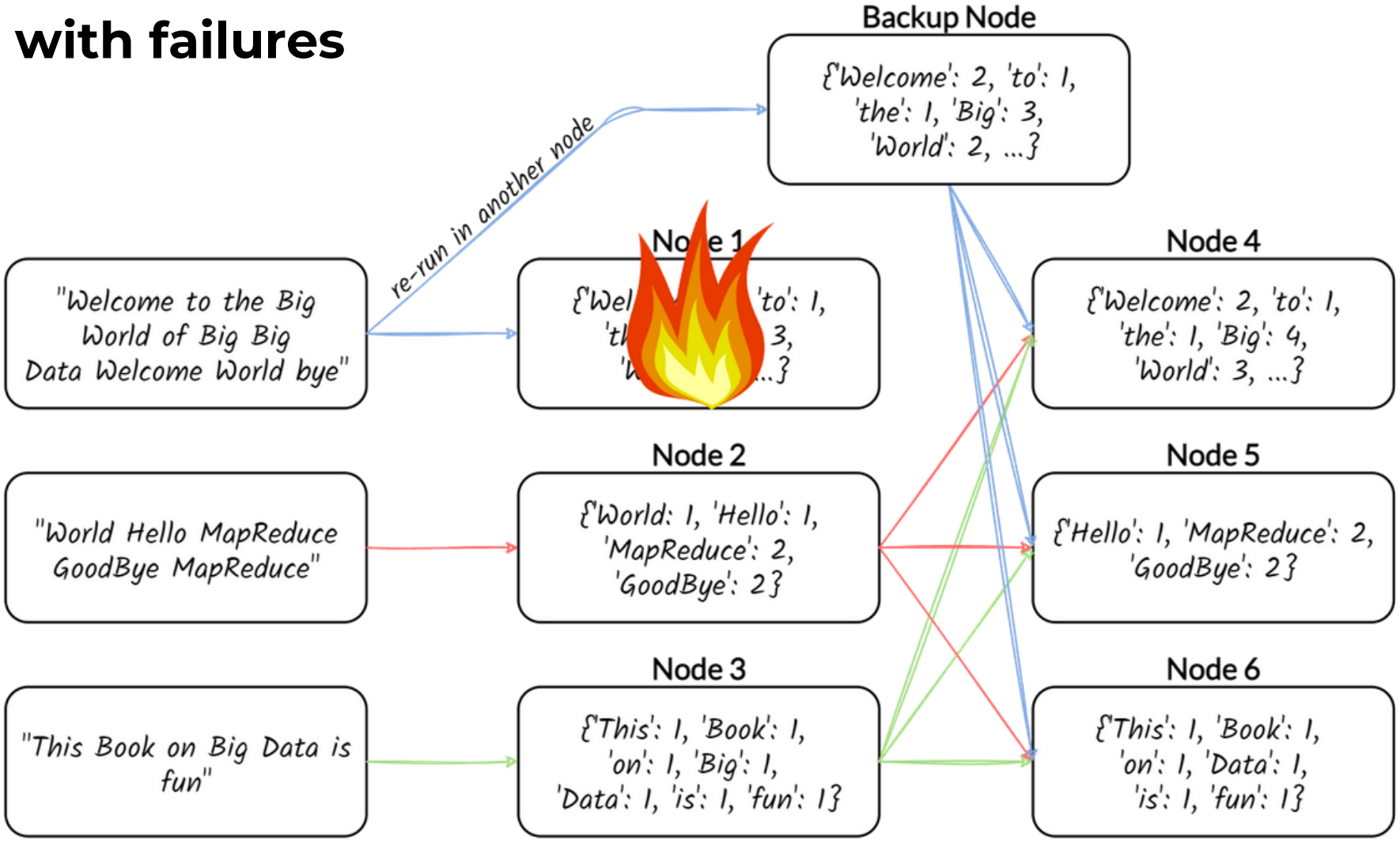
- For example:
  - *Node1* would keep World, Hello, and MapReduce because they are the most popular ones (3,2 and 2 repetitions, respectively)
  - *Nodes 2* and *3* would keep the other ones... or maybe we don't even need a 3 node in this case.

- Can you do this in a single MapReduce step?

Challenges in distributed systems

- How do we split the work across machines?
  - Don’t forget about network, data locality
- How do we deal with hardware failures?
  - 1 server fails every 3 years → 10.000 nodes, 10 failure/day
  - Even worse: those nodes could just be slow :-(, and that’s more difficult to detect!

Dealing with failures



Take-home message

- MapReduce – parallel programming model for Big Data
- Two main functions, `map` and `reduce`
- The challenges of distributed systems
  - (Transparent) Data locality
  - Hardware Failures

Next Lecture:

- More MapReduce!

CAMBRIDGE UNIVERSITY PRESS

Chapter 2

MapReduce I