Gestion de Données à Grande Échelle

MapReduce and Hadoop

Francieli ZANON BOITO

francieli.zanon-boito@inria.fr

November 2018

References

- Slides by Thomas Ropars
- Coursera – Big Data, University of California San Diego
- The lecture notes of V. Leroy
- Designing Data-Intensive Applications by Martin Kleppmann
- Mining of Massive Datasets by Leskovec et al.

In today's class

- The MapReduce paradigm for big data processing, and its most popular implementation (Apache Hadoop)
- Main ideas and how it works
- In the TP: put it to practice

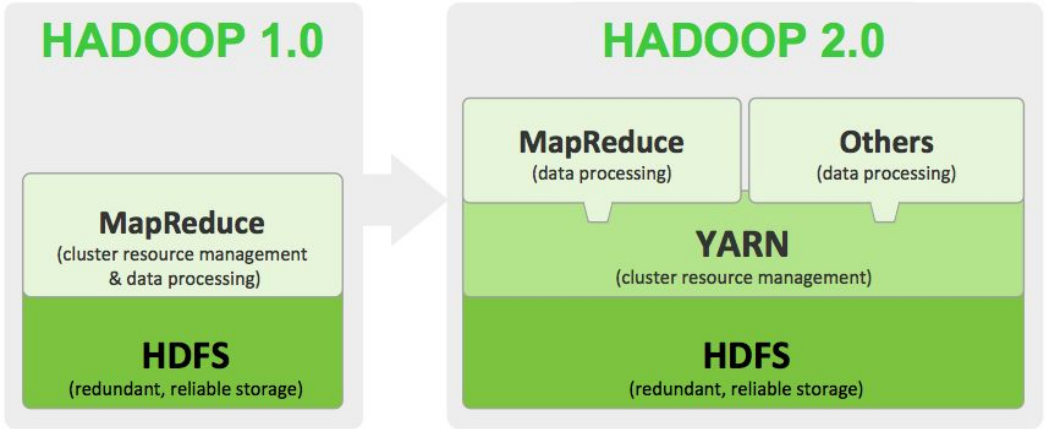
History

- First publications
  - "The Google File System", S. Ghemawat et al. 2003
  - "MapReduce: simplified data processing on large clusters", D. Jeffrey and S. Ghemawat 2004
- Used to implement several tasks:
  - Building the indexing system for Google Search
  - Extracting properties of web pages
  - Graph processing, etc
- Google does not use MapReduce anymore*
  - The amount of data they handle increased too much
  - Moved on to more efficient technologies

* https://www.datacenterknowledge.com/archives/2014/06/25/google-dumps-mapreduce-favor-new-hyper-scale-analytics-system

History

- Apache Hadoop: open source MapReduce framework
  - Implemented by people working at Yahoo!, released in 2006
- Now it is a full ecosystem, used by many companies
  - Notably, Facebook *
  - HDFS @ Yahoo!: 600PB on 35K servers **
  - Criteo (42k cores, 150PB, 300k jobs per day) ***



* https://dzone.com/articles/how-is-facebook-deploying-big-data

** http://yahoohadoop.tumblr.com/post/138739227316/hadoop-turns-10

*** http://labs.criteo.com/about-us/

Main ideas

- A distributed computing execution framework
- Data represented as key-value pairs
- A distributed file system
- Two main operations on data: Map and Reduce

Map and Reduce

- The Map operation
  - Transformation operation
  - A function is applied to each element of the input set
  - map(f)[x_0, ..., x_n] = [f (x_0), ..., f (x_n)]
  - map(∗2)[2, 3, 6] = [4, 6, 12]
- The Reduce operation
  - Aggregation operation (fold)
  - reduce(f)[x_0, ..., x_n] = [f (x_0, f (x_1, ..., f (x_{n-1} , x_n)))]
  - reduce(+)[2, 3, 6] = (2 + (3 + 6)) = 11
  - In MapReduce, Reduce is applied to all the elements with the same key

Why is it so popular?

Code once (expert), benefit to all

- “Simple” to program and execute
  - Handles distribution of data and the computation
  - Detects failure and automatically takes corrective actions
- Scale to large number of nodes
  - Data parallelism (as opposed to task parallelism): running the same task on different data pieces in parallel
  - Move the computation instead of the data
    - Distributed file system is central
    - Execute tasks where their data is

Why is it so popular?

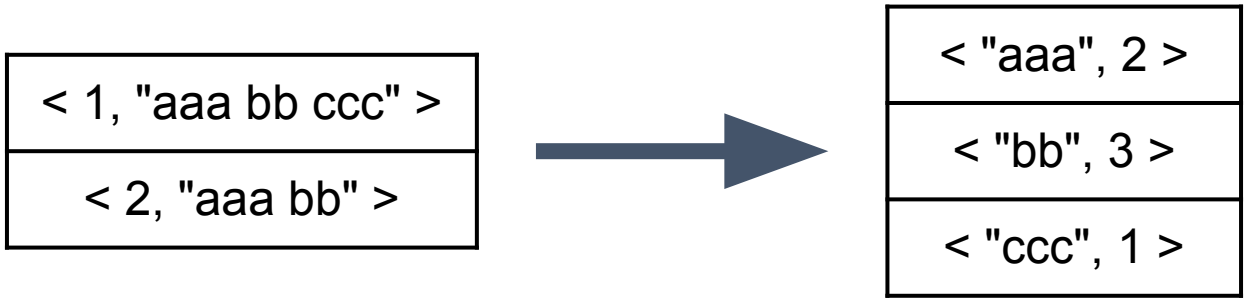
- Fault tolerance
  - Data replication by the distributed file system
  - Intermediate results are written to disk
  - Failed tasks are re-executed on other nodes
  - Tasks can be executed multiple times in parallel to deal with stragglers (slow nodes)

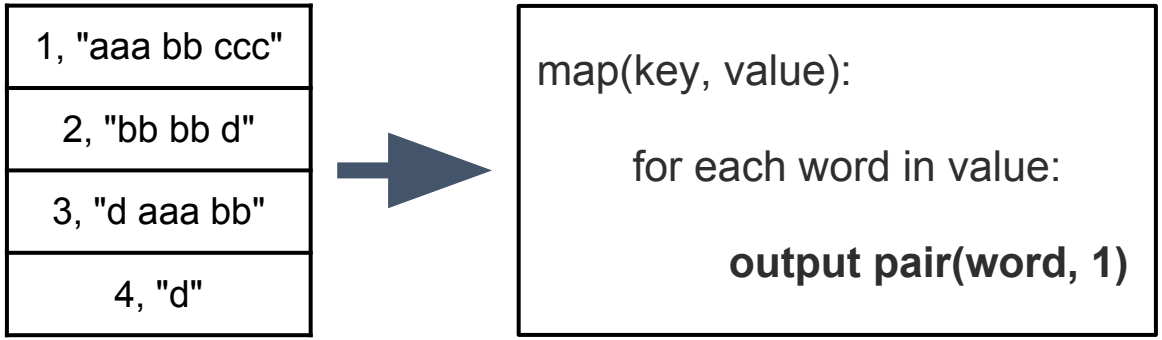
Agenda

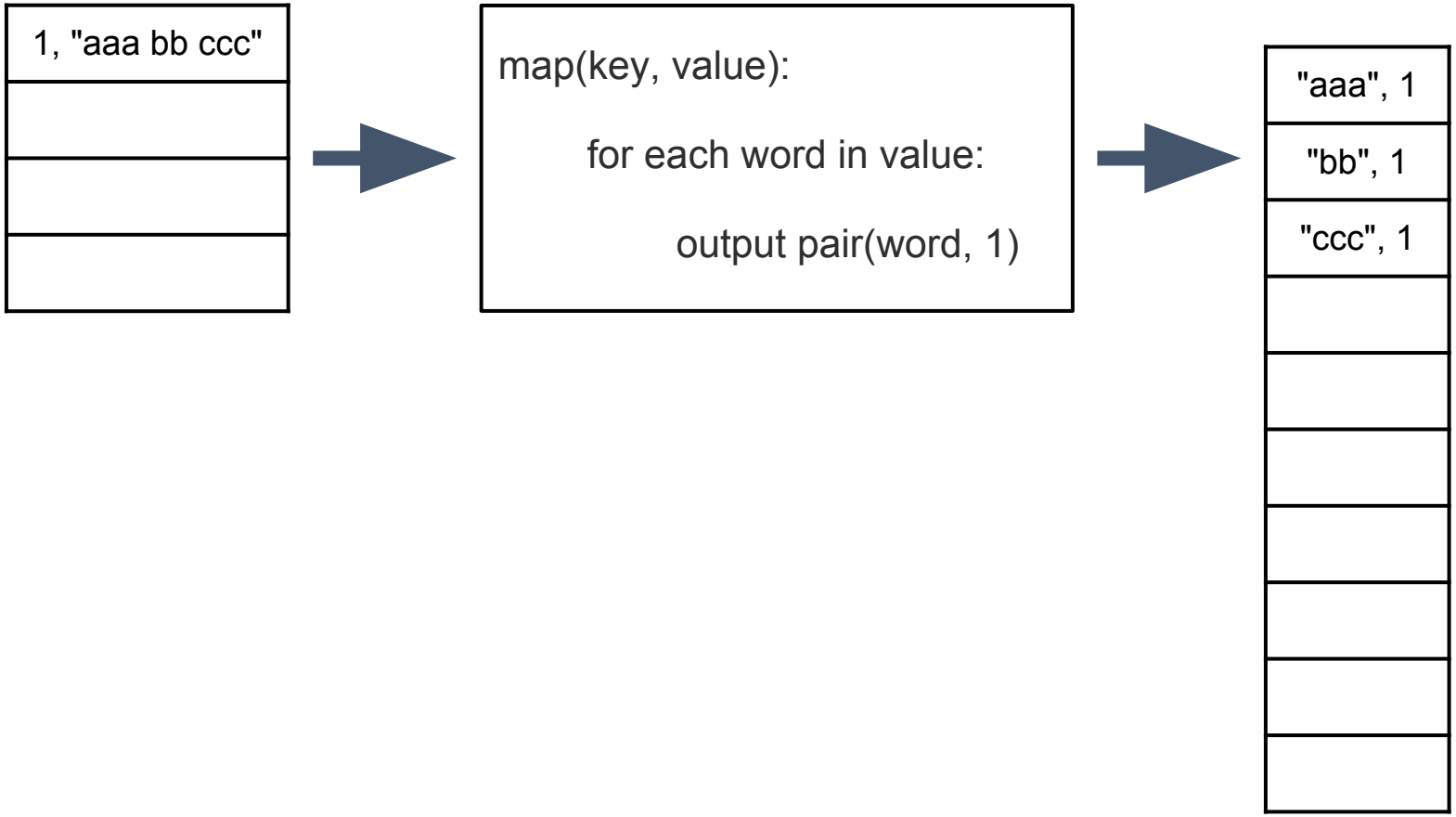
- Introduction
- **A first MapReduce program**
- Apache Hadoop
  - MapReduce
  - HDFS
  - Yarn
- Combiners

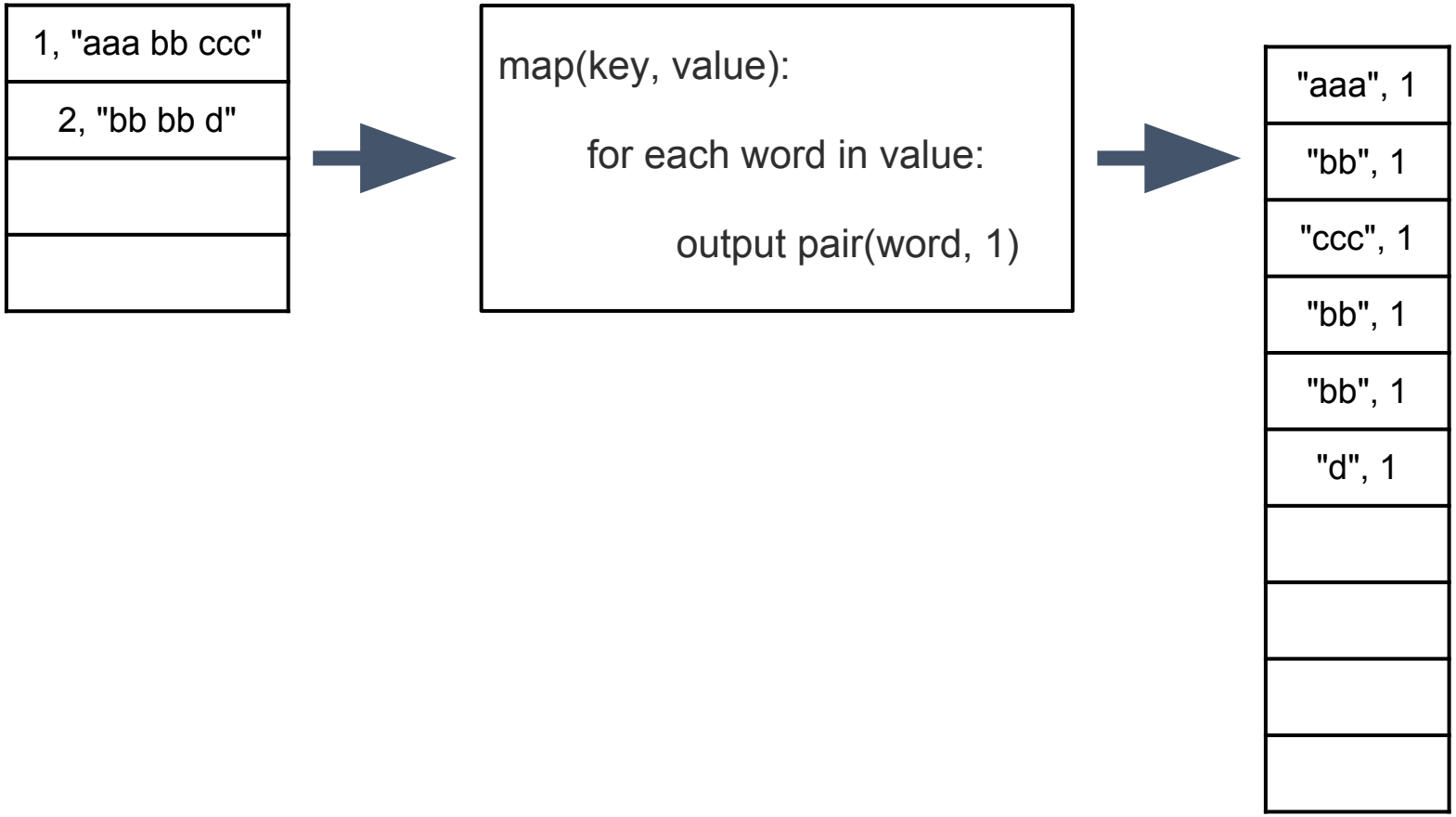
A first MapReduce program: word counter

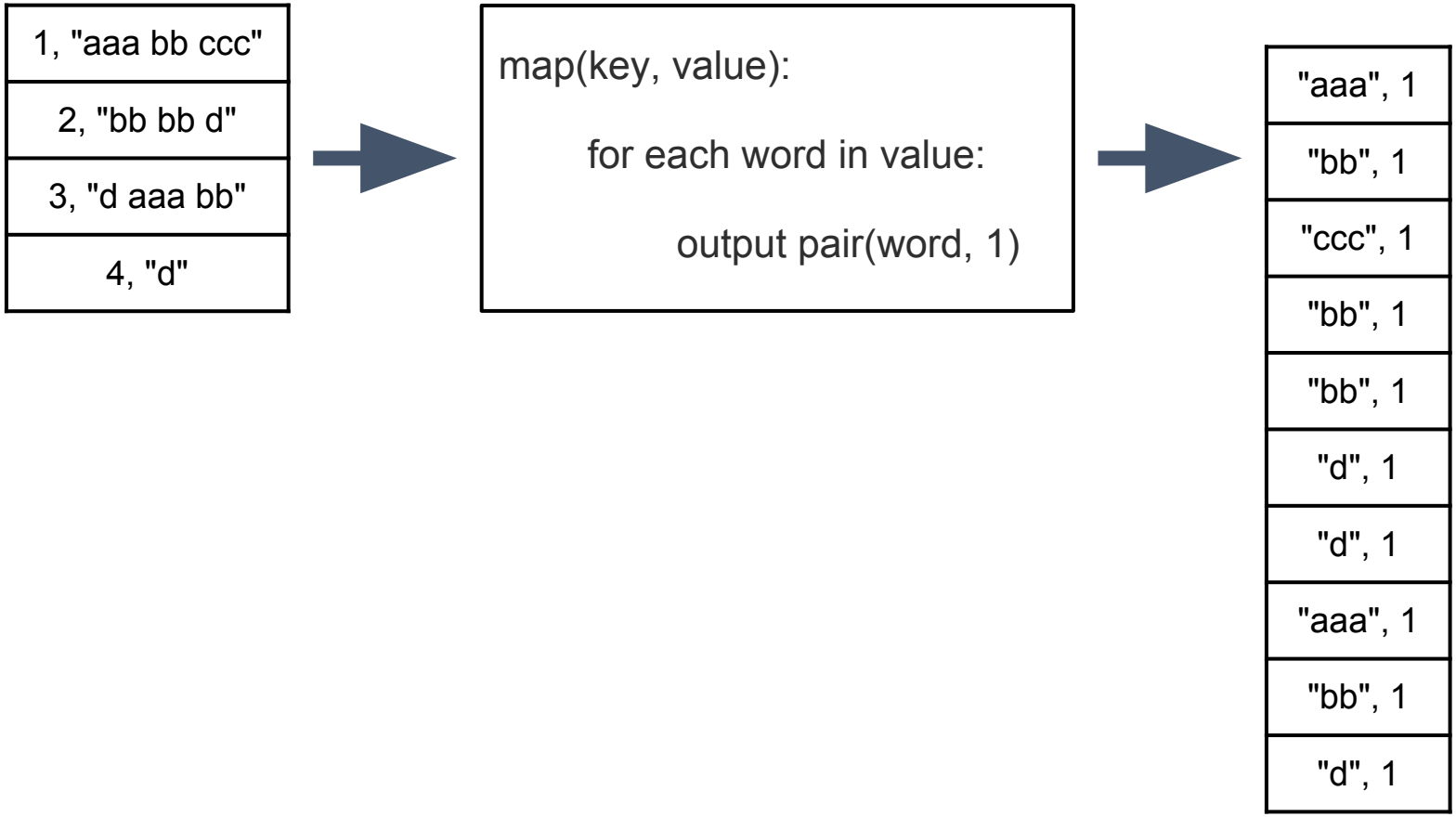
- We want to count the occurrences of words in a text
- Input: A set of lines, each line is a pair < line number, line content >
- Output: A set of pairs < word, number of occurrences >

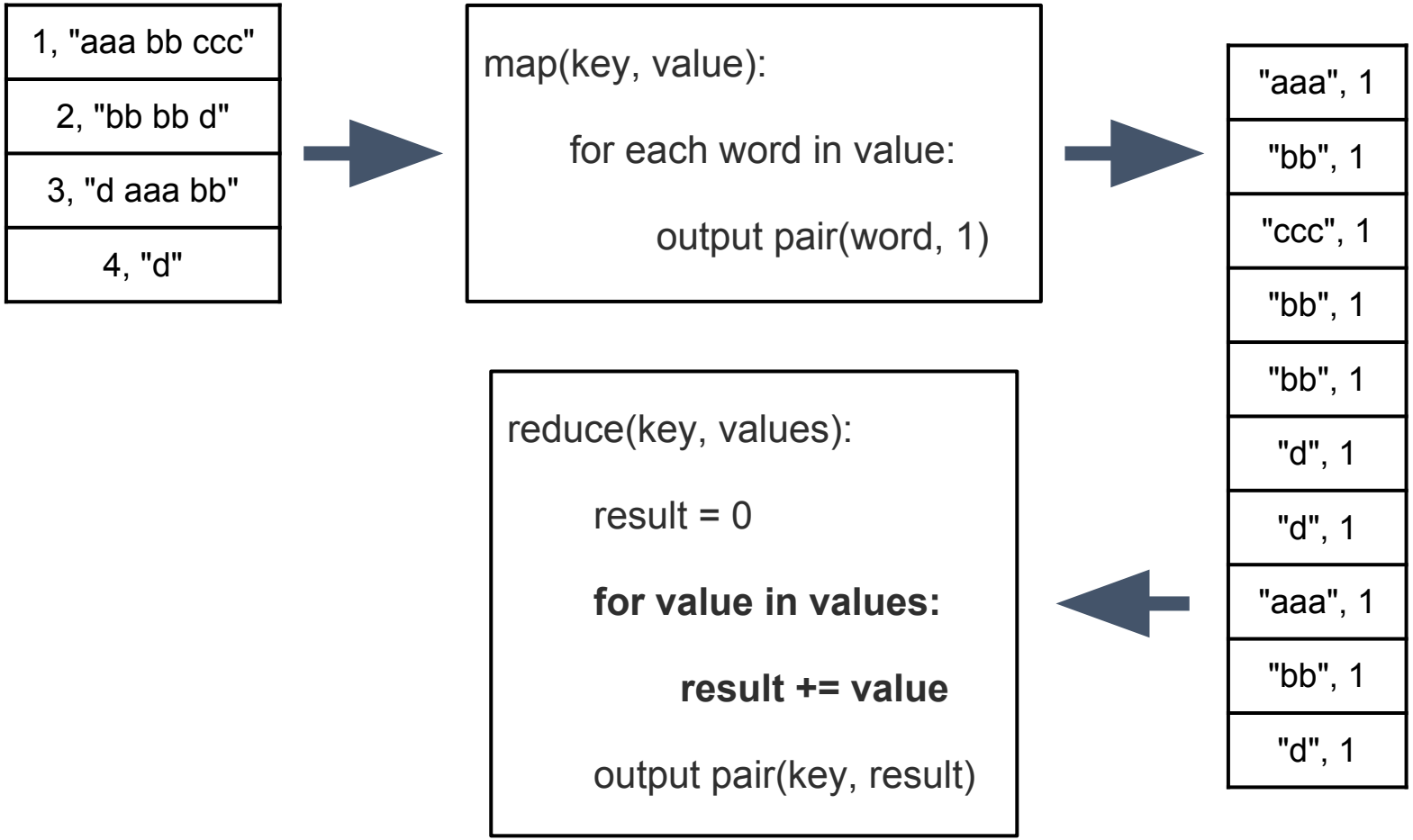


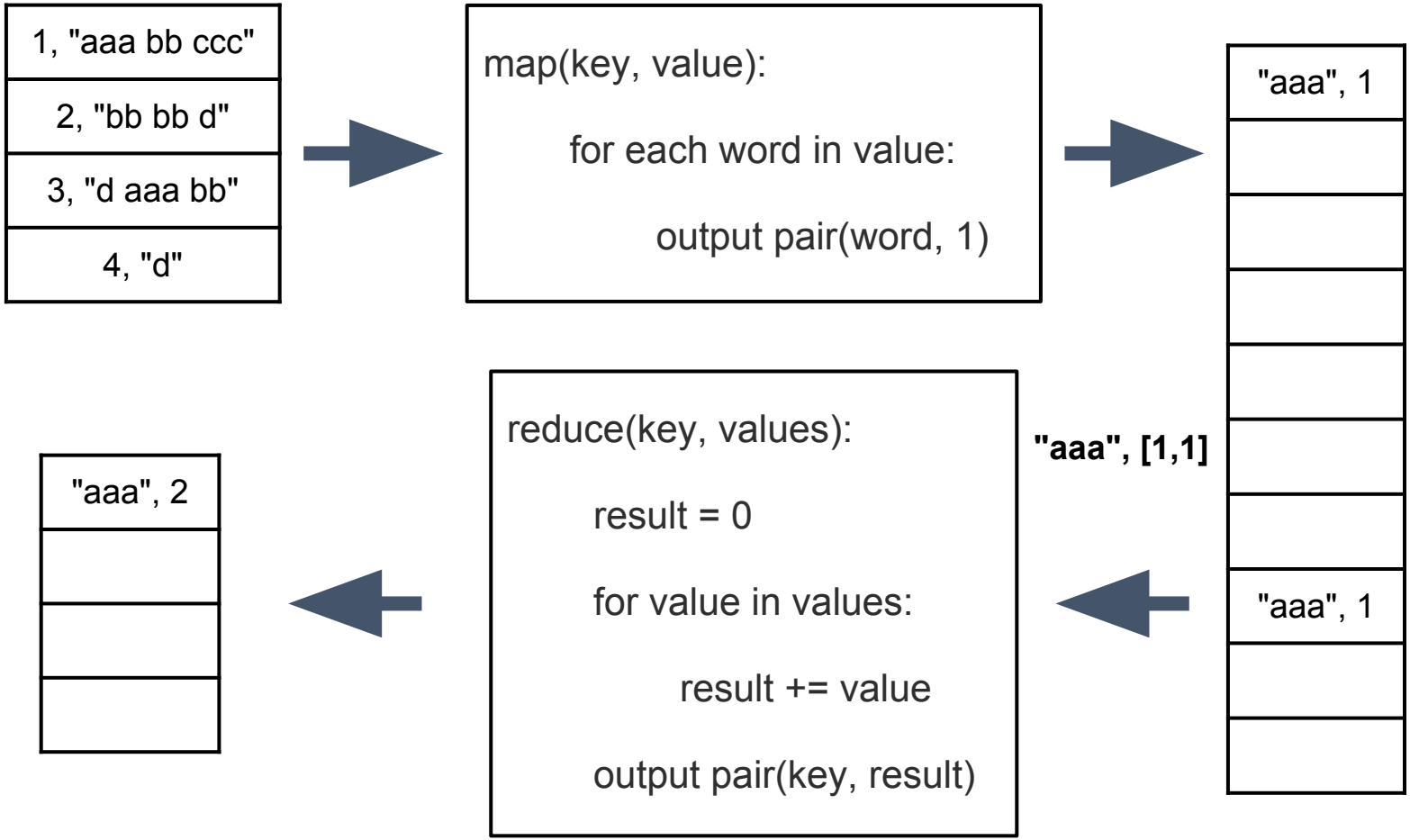

1, "aaa bb ccc"
2, "bb bb d"
3, "d aaa bb"
4, "d"
map(key, value):
for each word in value:
output pair(word, 1)

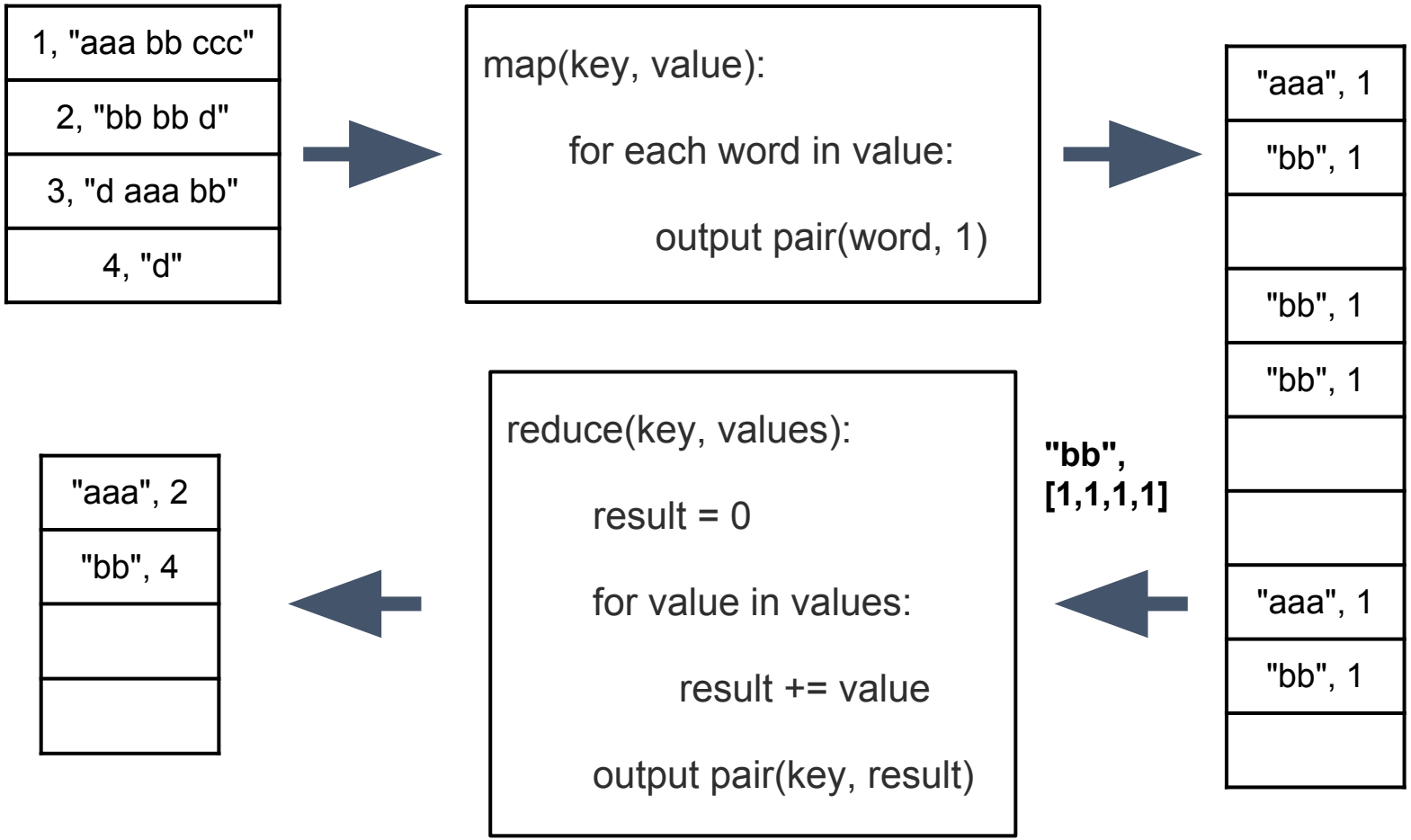

1, "aaa bb ccc"
map(key, value):
for each word in value:
output pair(word, 1)
"aaa", 1
"bb", 1
"ccc", 1

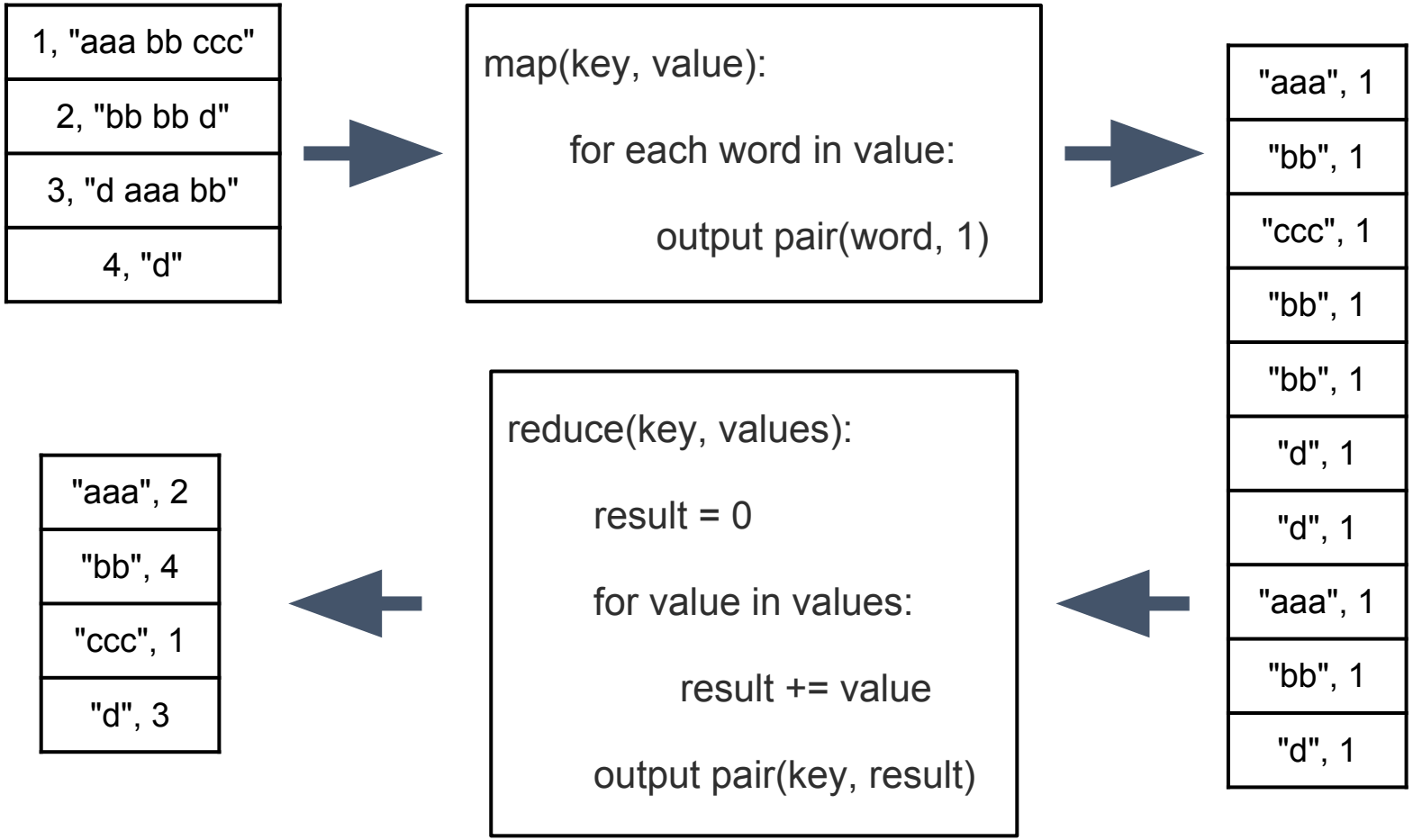

1, "aaa bb ccc"
2, "bb bb d"
map(key, value):
for each word in value:
output pair(word, 1)
"aaa", 1
"bb", 1
"ccc", 1
"bb", 1
"bb", 1
"d", 1

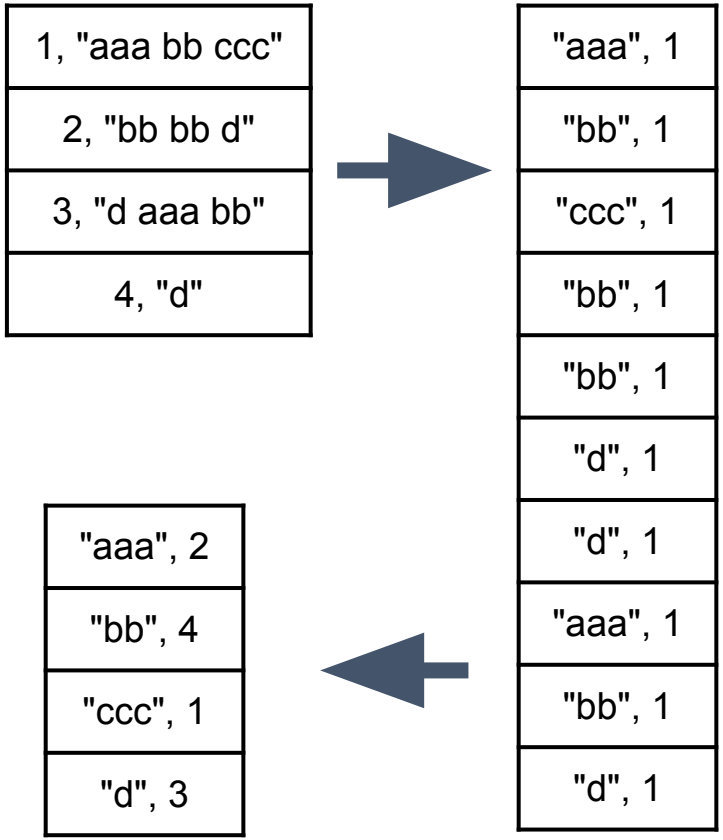

1, "aaa bb ccc"
2, "bb bb d"
3, "d aaa bb"
4, "d"
map(key, value):
for each word in value:
output pair(word, 1)
"aaa", 1
"bb", 1
"ccc", 1
"bb", 1
"bb", 1
"d", 1
"d", 1
"aaa", 1
"bb", 1
"d", 1


1, "aaa bb ccc"
2, "bb bb d"
3, "d aaa bb"
4, "d"
map(key, value):
for each word in value:
output pair(word, 1)
"aaa", 1
"bb", 1
"ccc", 1
"bb", 1
"bb", 1
"d", 1
"d", 1
"aaa", 1
"bb", 1
"d", 1
reduce(key, values):
result = 0
for value in values:
result += value
output pair(key, result)


1, "aaa bb ccc"
2, "bb bb d"
3, "d aaa bb"
4, "d"
map(key, value):
for each word in value:
output pair(word, 1)
"aaa", 1
"aaa", 1
"aaa", [1,1]
reduce(key, values):
result = 0
for value in values:
result += value
output pair(key, result)
"aaa", 2


1, "aaa bb ccc"
2, "bb bb d"
3, "d aaa bb"
4, "d"
map(key, value):
for each word in value:
output pair(word, 1)
"aaa", 1
"bb", 1
"bb", 1
"bb", 1
"aaa", 1
"bb", 1
"bb",
[1,1,1,1]
reduce(key, values):
result = 0
for value in values:
result += value
output pair(key, result)
"aaa", 2
"bb", 4


1, "aaa bb ccc"
2, "bb bb d"
3, "d aaa bb"
4, "d"
map(key, value):
for each word in value:
output pair(word, 1)
"aaa", 1
"bb", 1
"ccc", 1
"bb", 1
"bb", 1
"d", 1
"d", 1
"aaa", 1
"bb", 1
"d", 1
reduce(key, values):
result = 0
for value in values:
result += value
output pair(key, result)
"aaa", 2
"bb", 4
"ccc", 1
"d", 3



But we generate a lot of intermediate data!

Why not keep a centralized counter per word?

That's the price we pay for scalability!

Let's see how it works.

Agenda

- Introduction
- A first MapReduce program
- **Apache Hadoop**
  - MapReduce
  - HDFS
  - Yarn
- Combiners

MapReduce

- The developer defines:
  - map and reduce functions to manipulate key-value pairs
  - key and value types (map output needs to match reduce input)
- The map function will be executed once per input pair
- The reduce function will be executed once per existing key (with all the values associated with that key)

We start with the input separated in blocks and distributed over the nodes

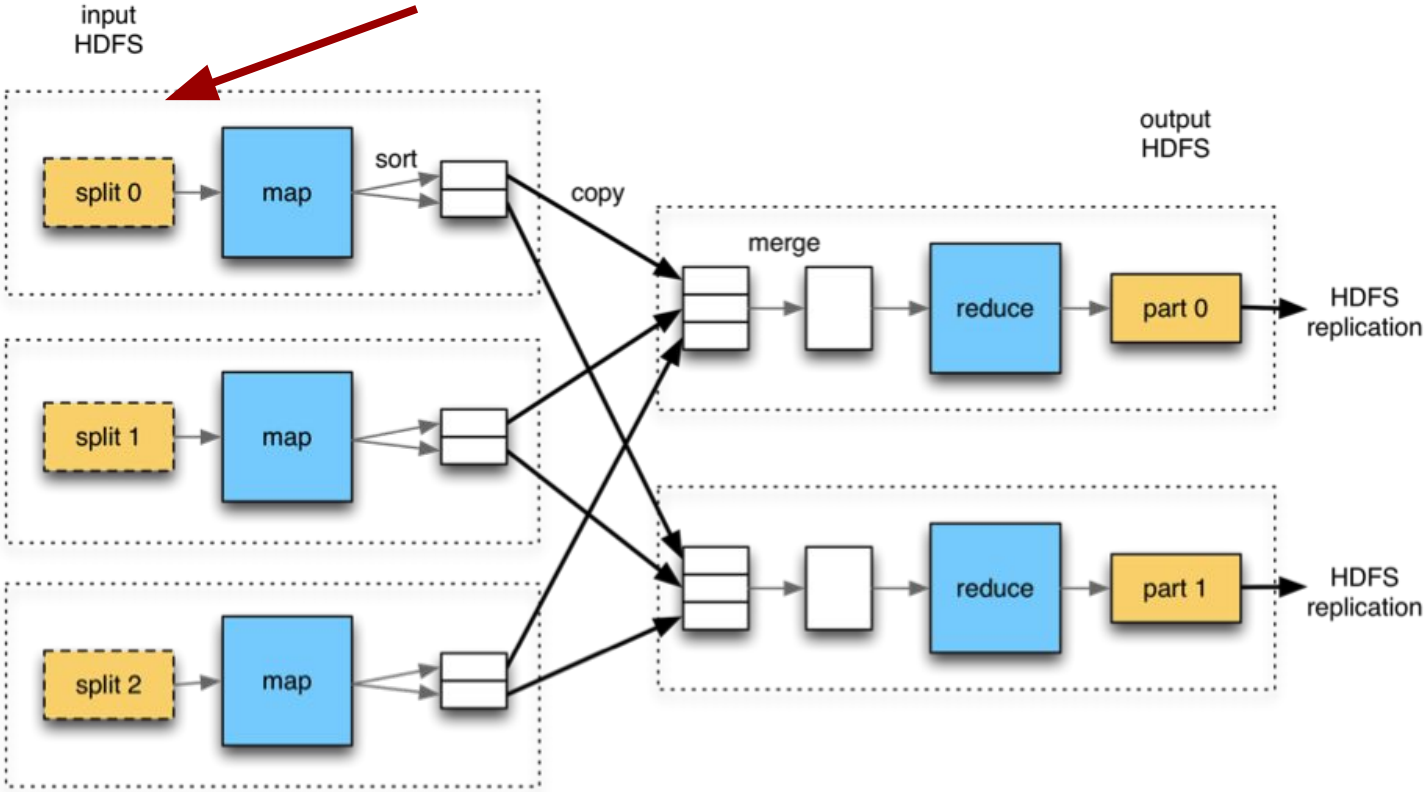


Figure from https://www.supinfo.com/articles/single/2807-introduction-to-the-mapreduce-life-cycle

We have one map task per input block (each task executes the map function multiple times)
In the same node to avoid data movement!

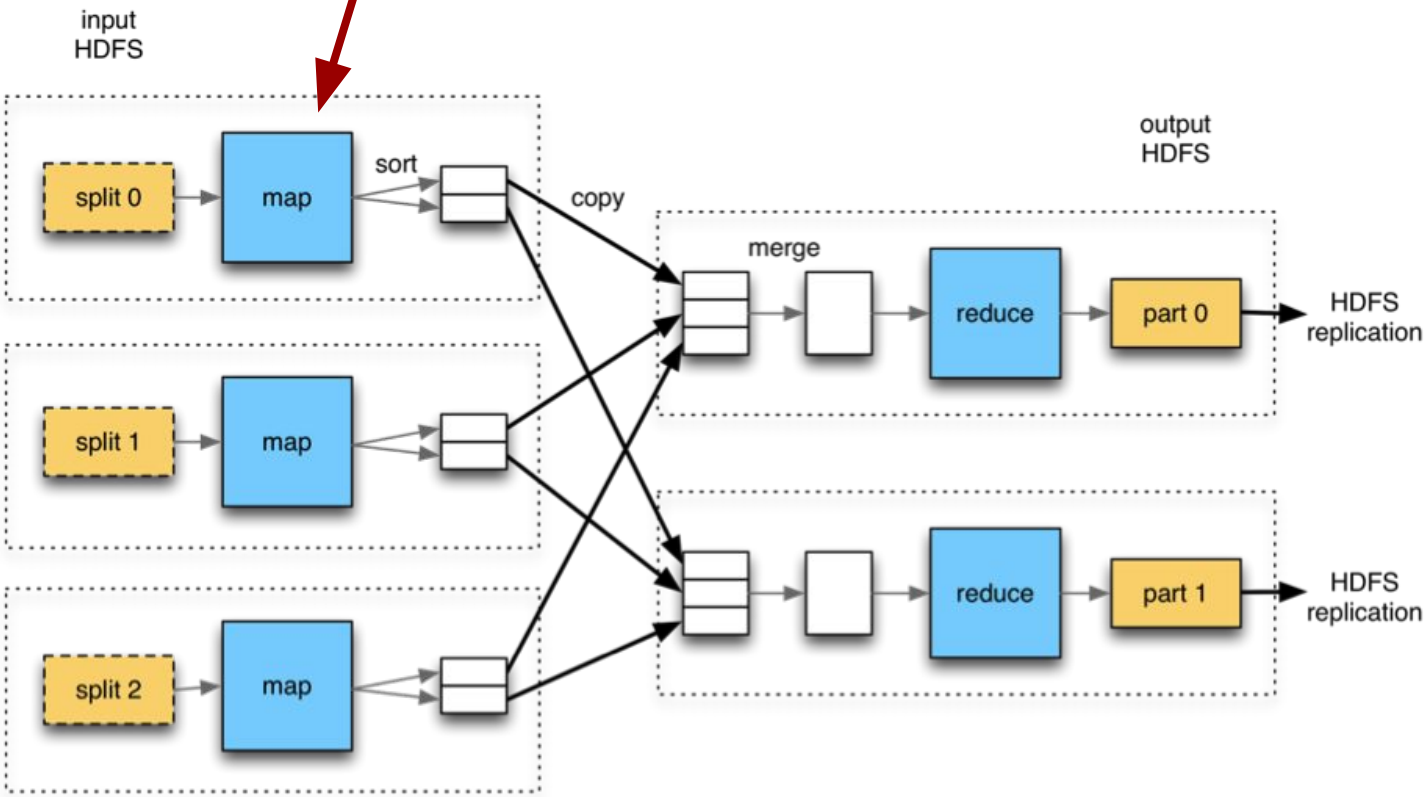


Figure from https://www.supinfo.com/articles/single/2807-introduction-to-the-mapreduce-life-cycle

Now we have the Shuffle & Sort phase
First sort each map task output by key

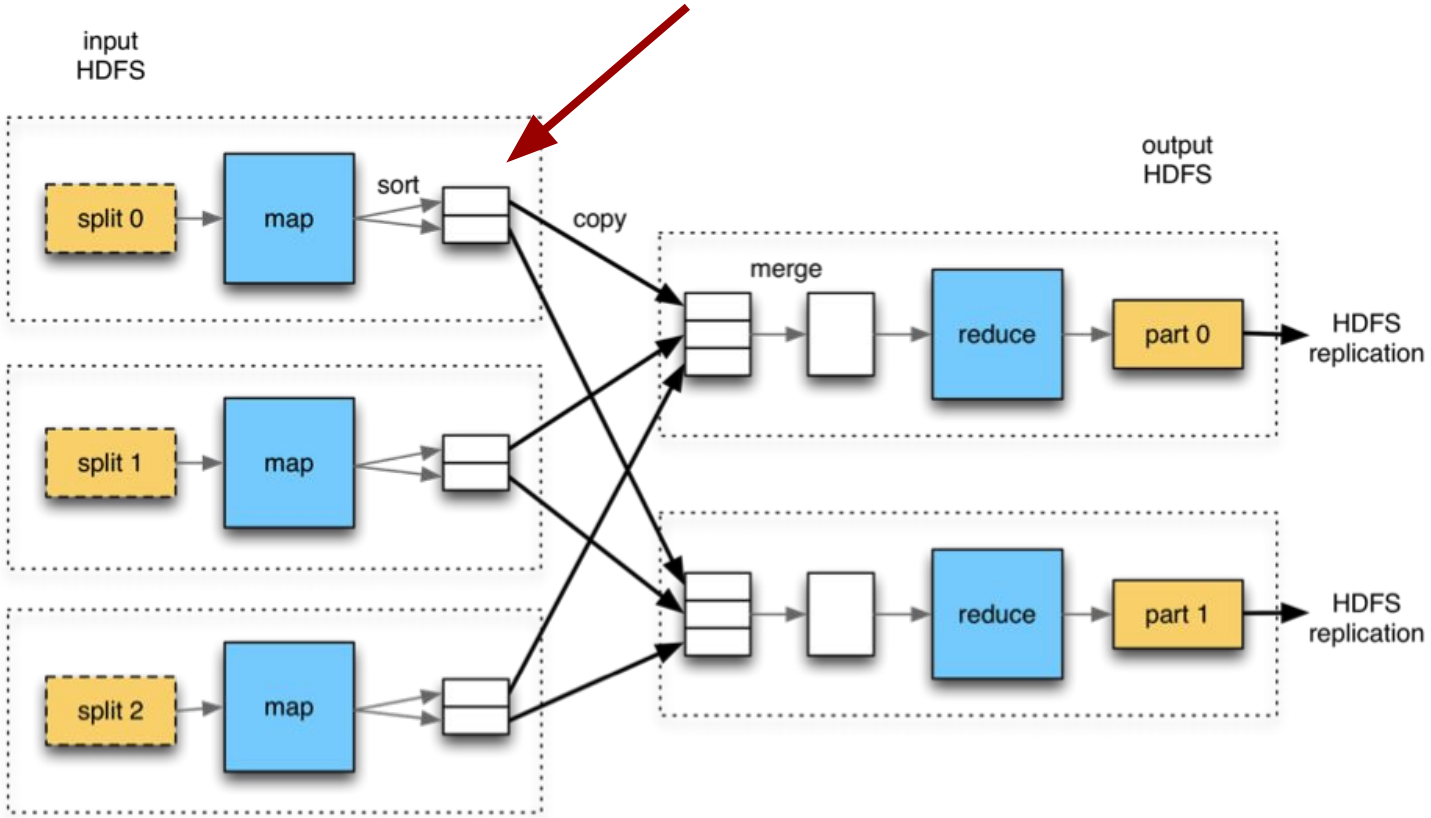


Figure from https://www.supinfo.com/articles/single/2807-introduction-to-the-mapreduce-life-cycle

Send the pairs to the adequate reduce task (hashing)
The number of reduce tasks is configurable

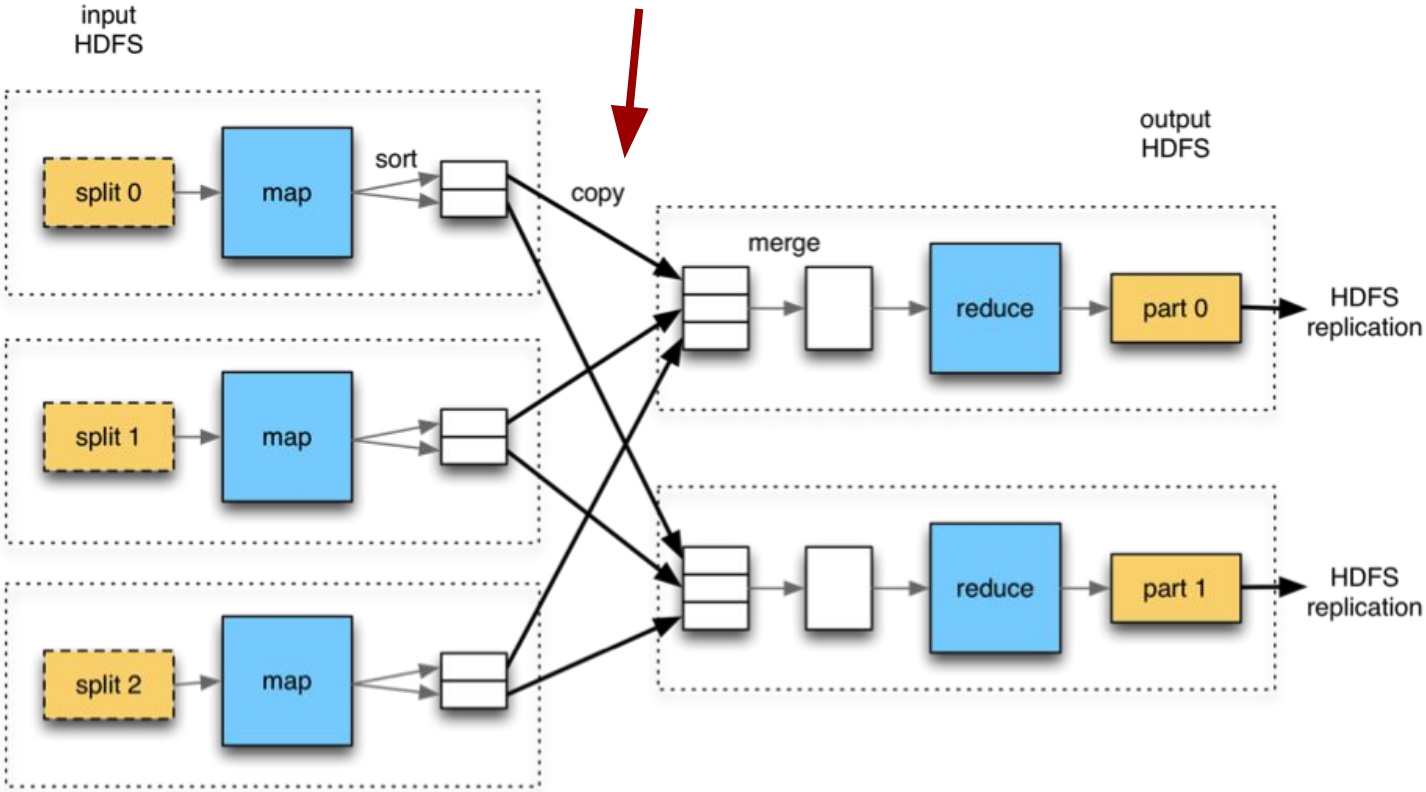


Figure from https://www.supinfo.com/articles/single/2807-introduction-to-the-mapreduce-life-cycle

Combine the pairs that have the same key

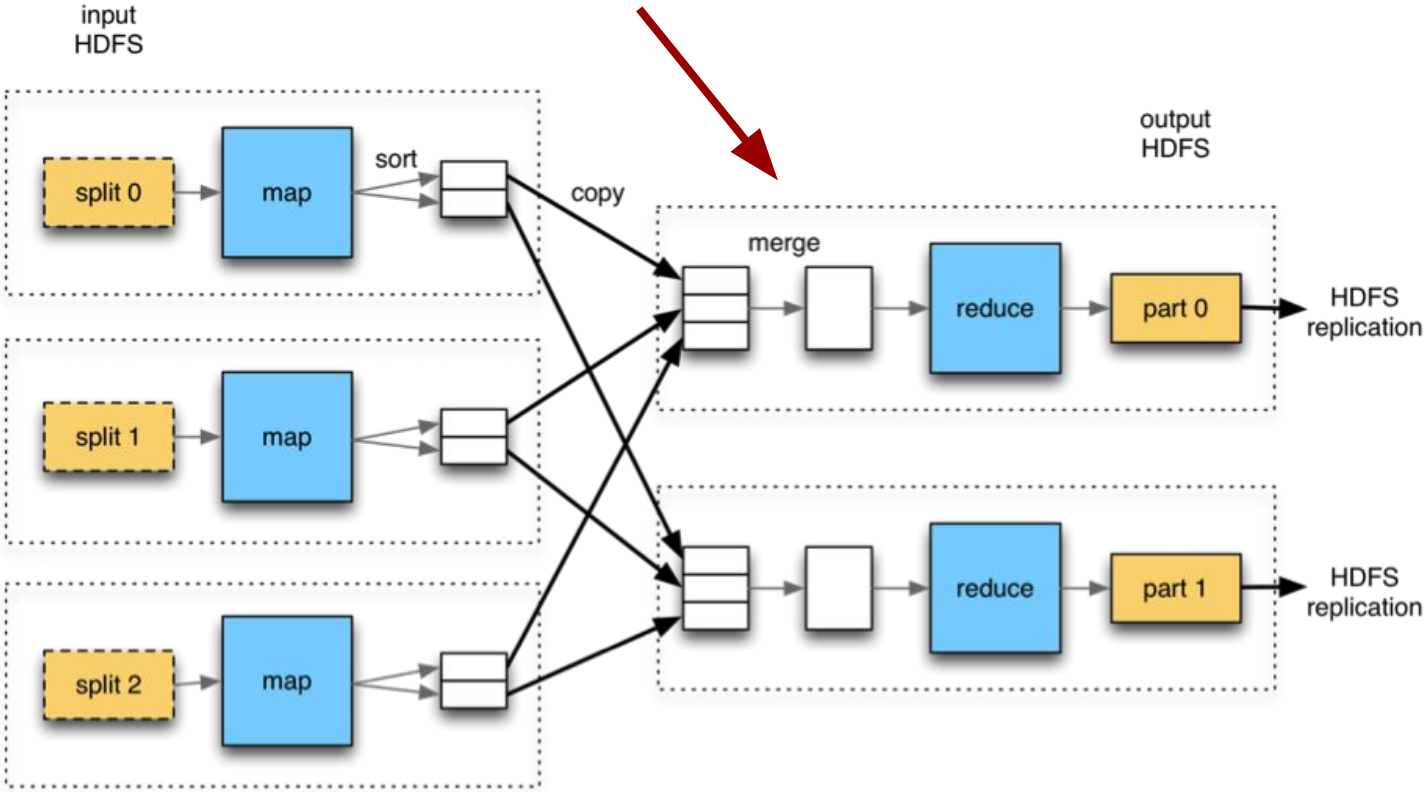


Figure from https://www.supinfo.com/articles/single/2807-introduction-to-the-mapreduce-life-cycle

Run the reduce tasks

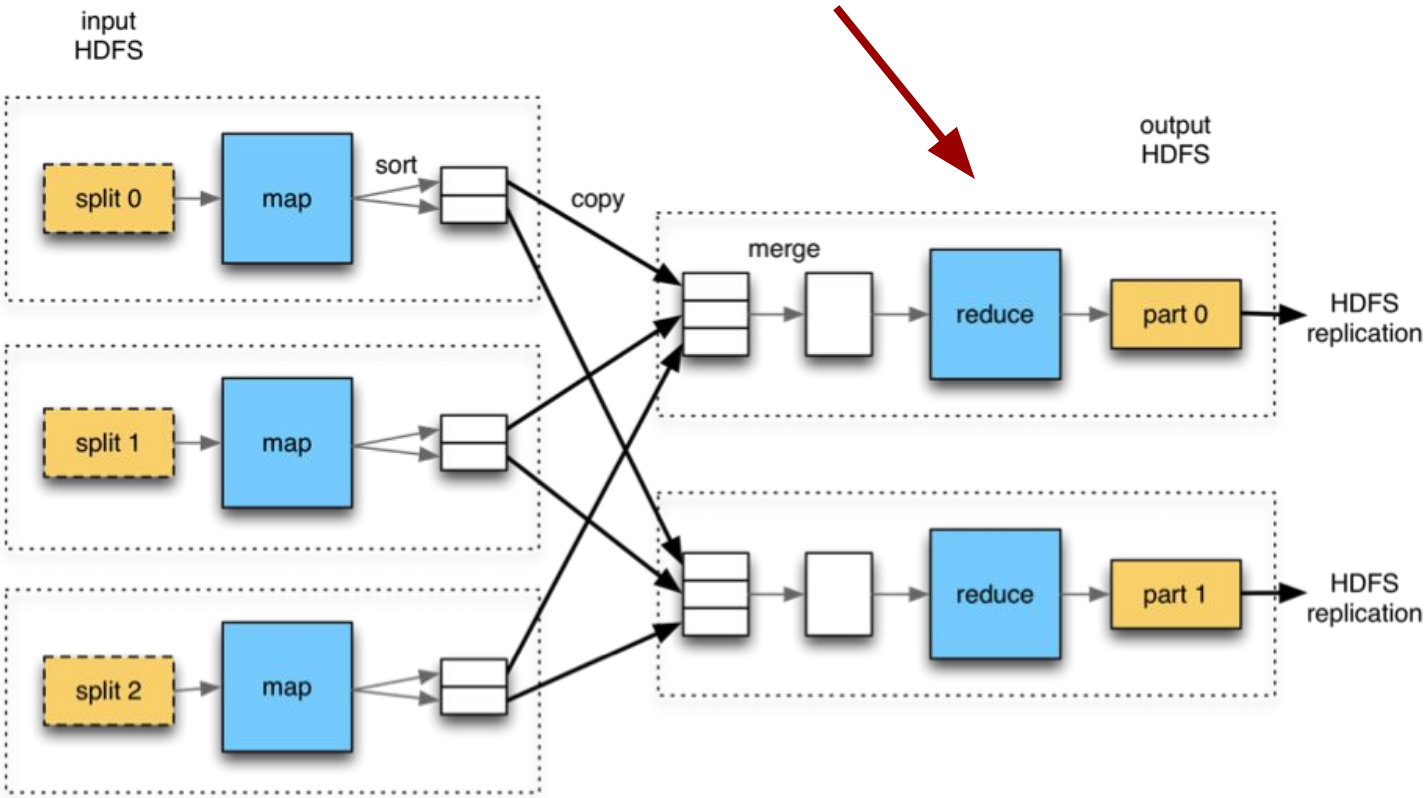


Figure from https://www.supinfo.com/articles/single/2807-introduction-to-the-mapreduce-life-cycle

Now we have (unsorted) output that is distributed over some nodes

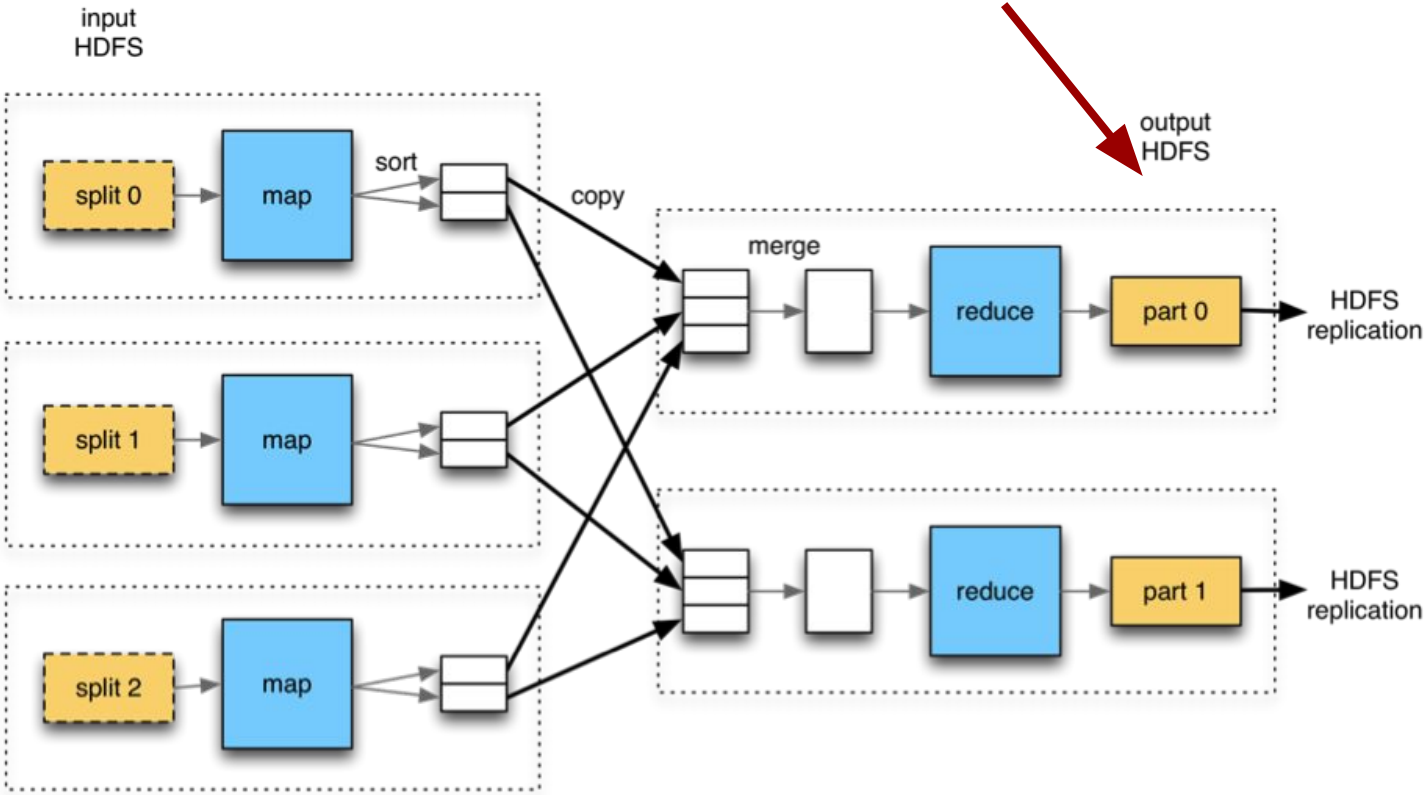


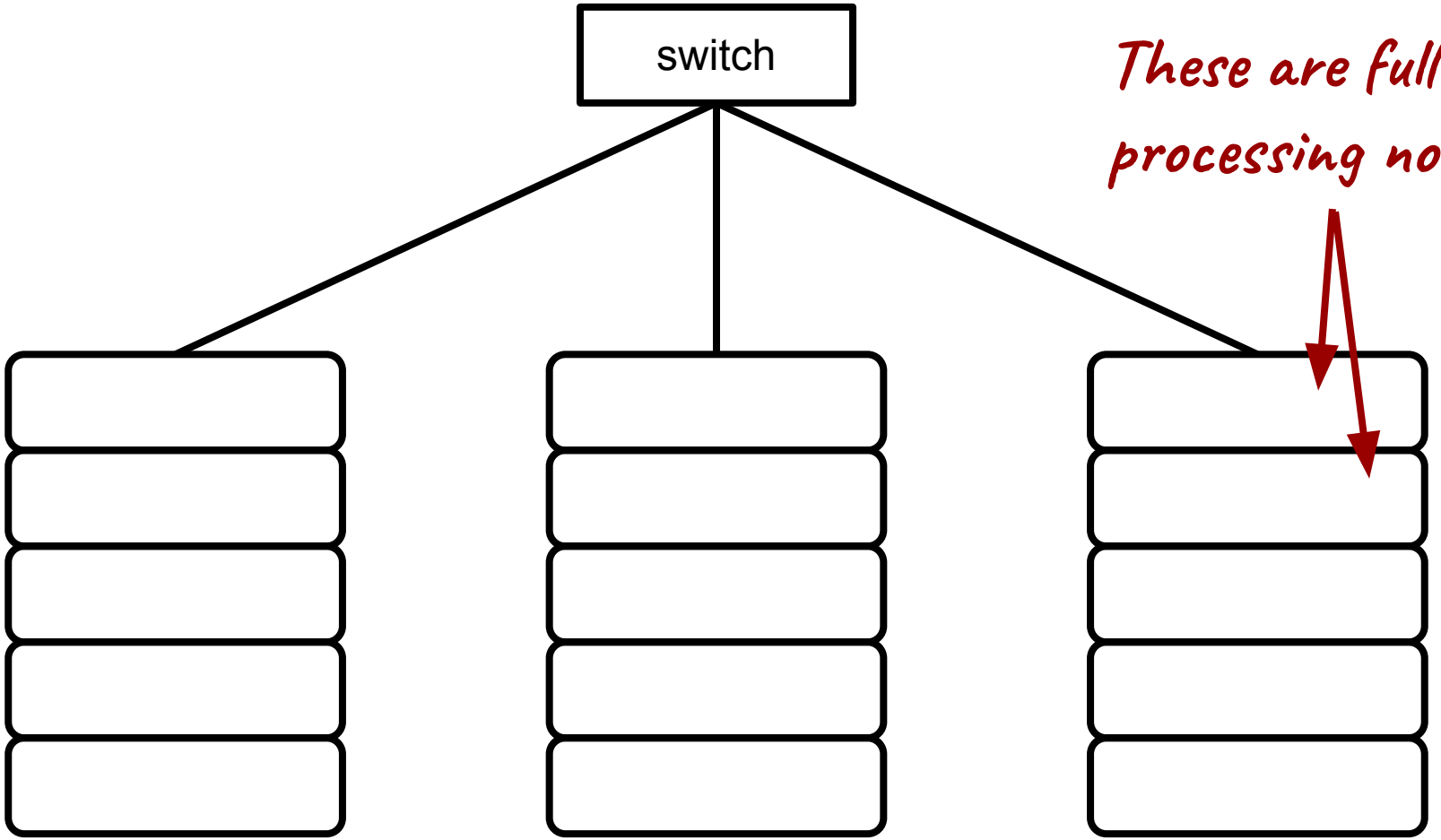
Figure from https://www.supinfo.com/articles/single/2807-introduction-to-the-mapreduce-life-cycle

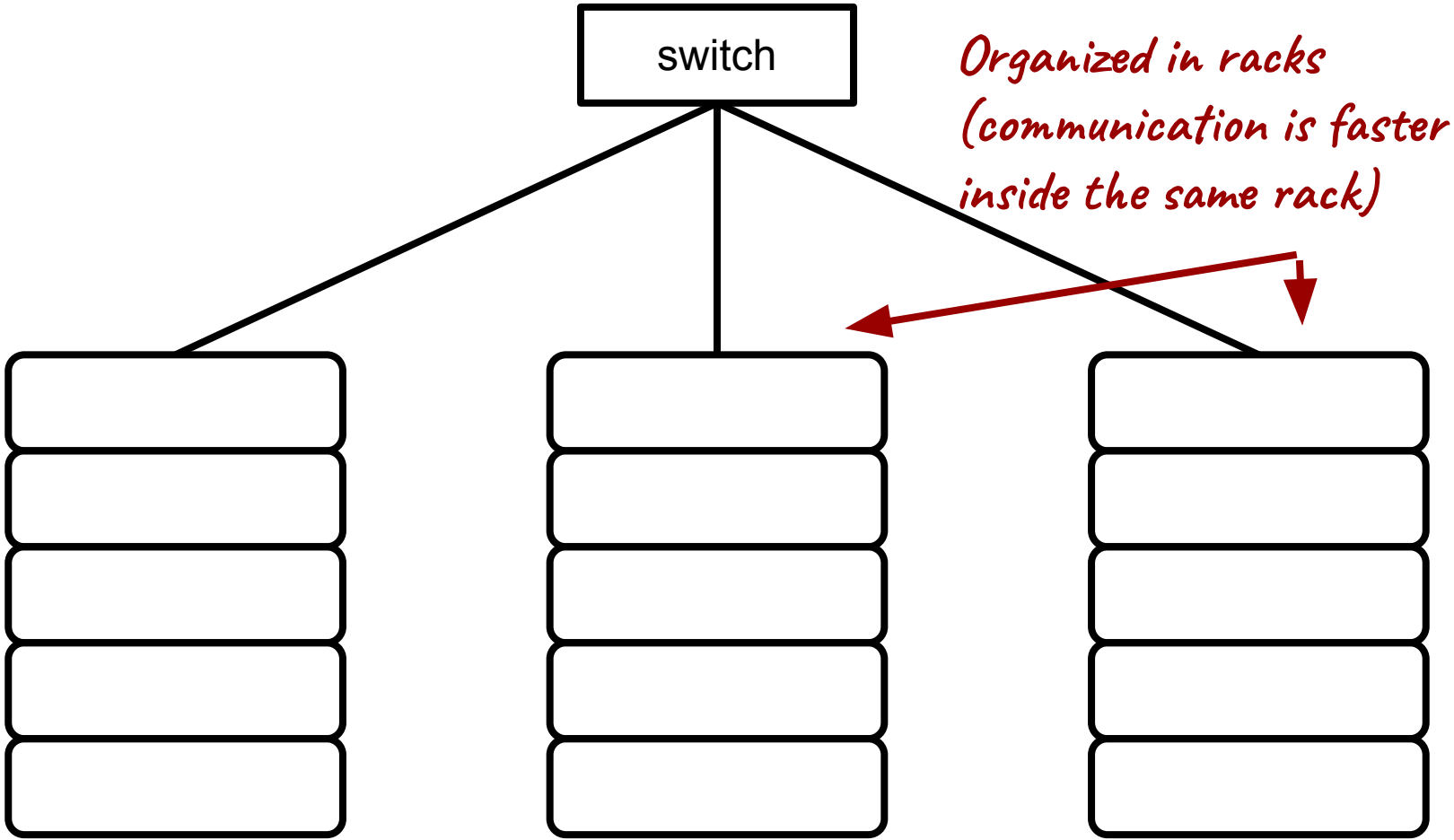
HDFS

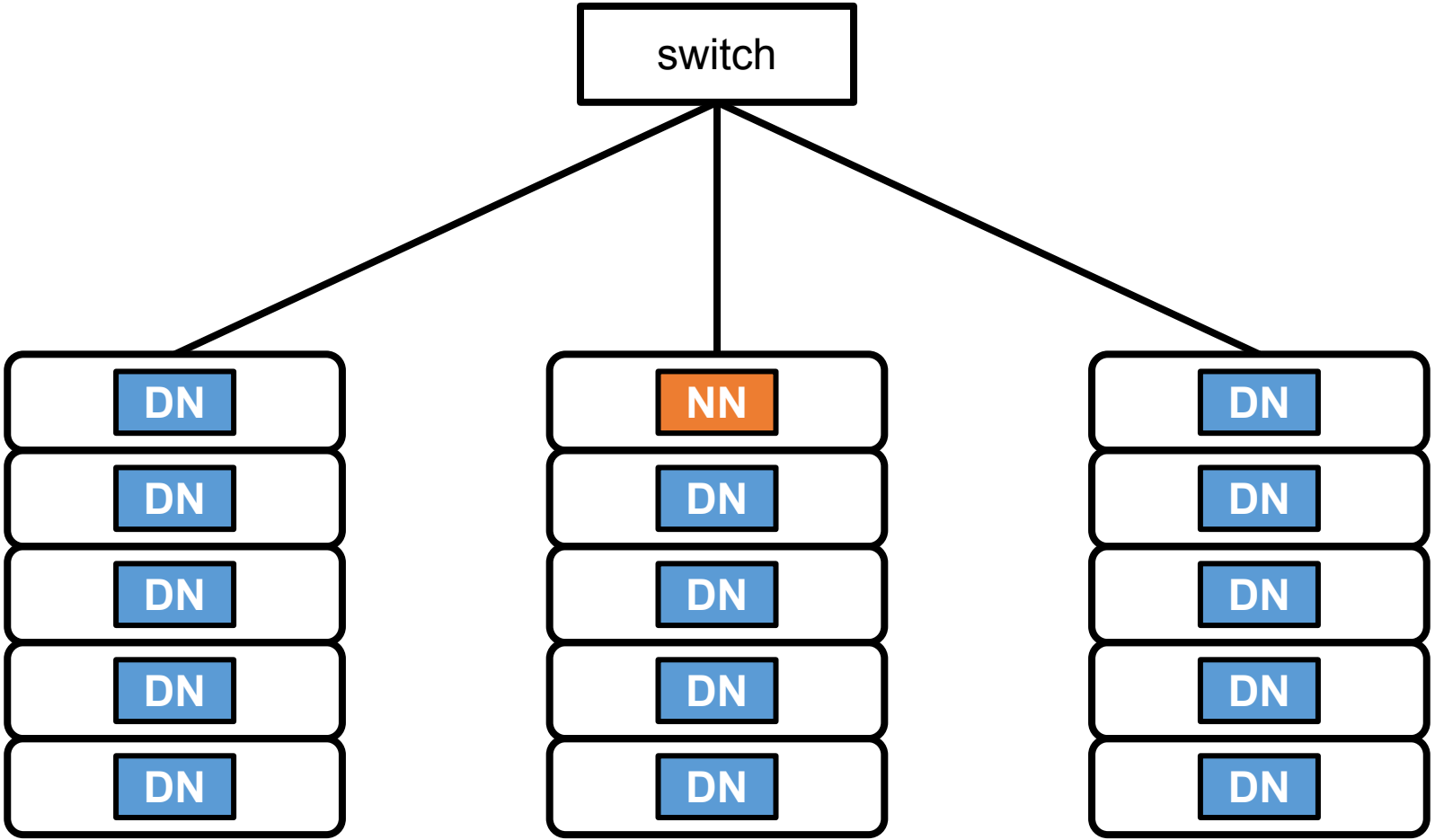
- Distributed file system for shared-nothing infrastructures
- Main goals: scalability and fault tolerance, optimized for throughput
- It is not POSIX-compliant
  - Sequential read and writes only
  - Write-once-read-many file access (supports append and truncate)

HDFS

- Files are partitioned into blocks and distributed over nodes
  - Recently: 128MB blocks
- Replicas are topology aware (rack awareness)
  - Default replication factor is 3
- Architecture:
  - NameNode: clients' entry point
  - one DataNode per node

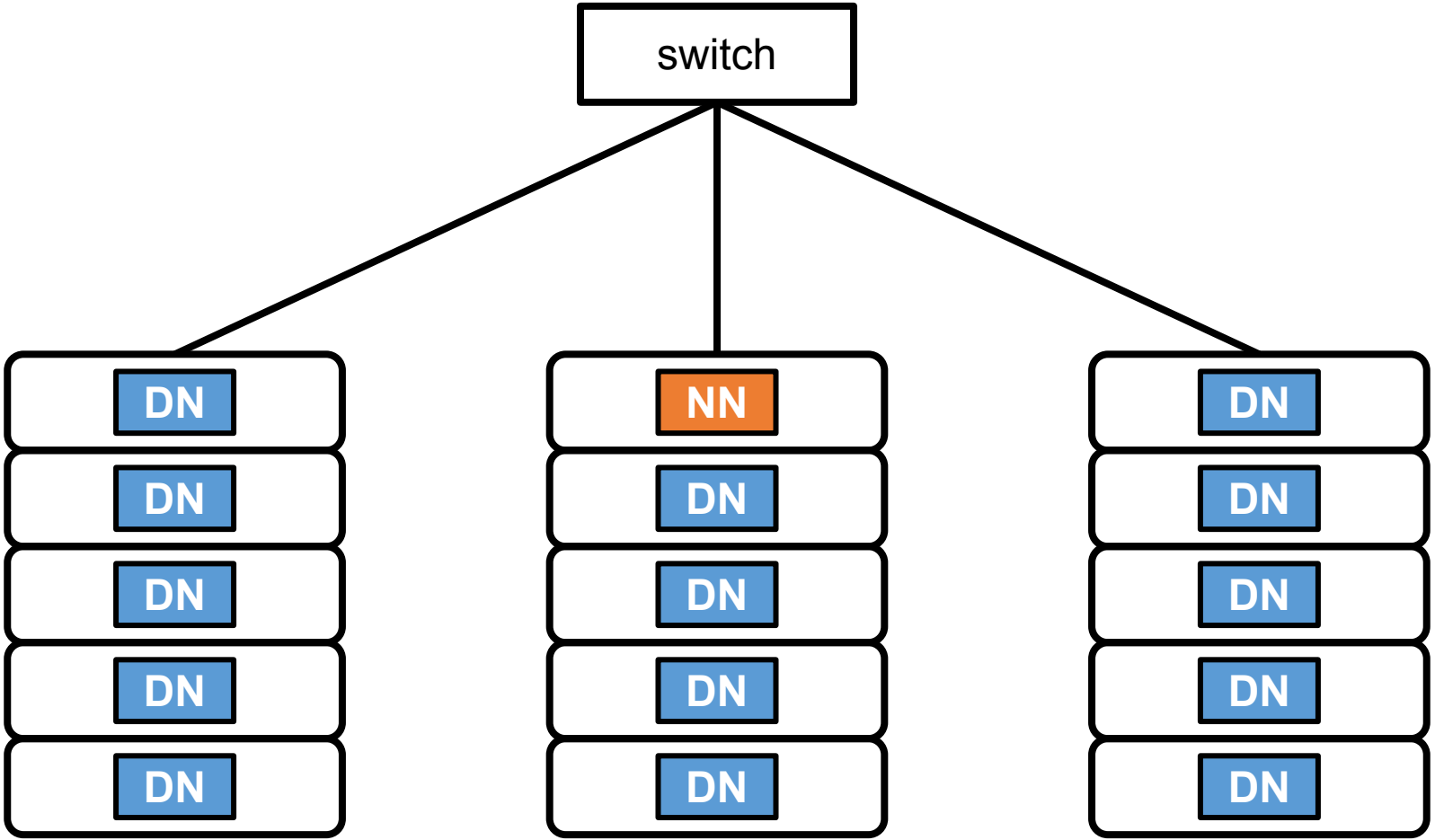

switch
These are full processing nodes

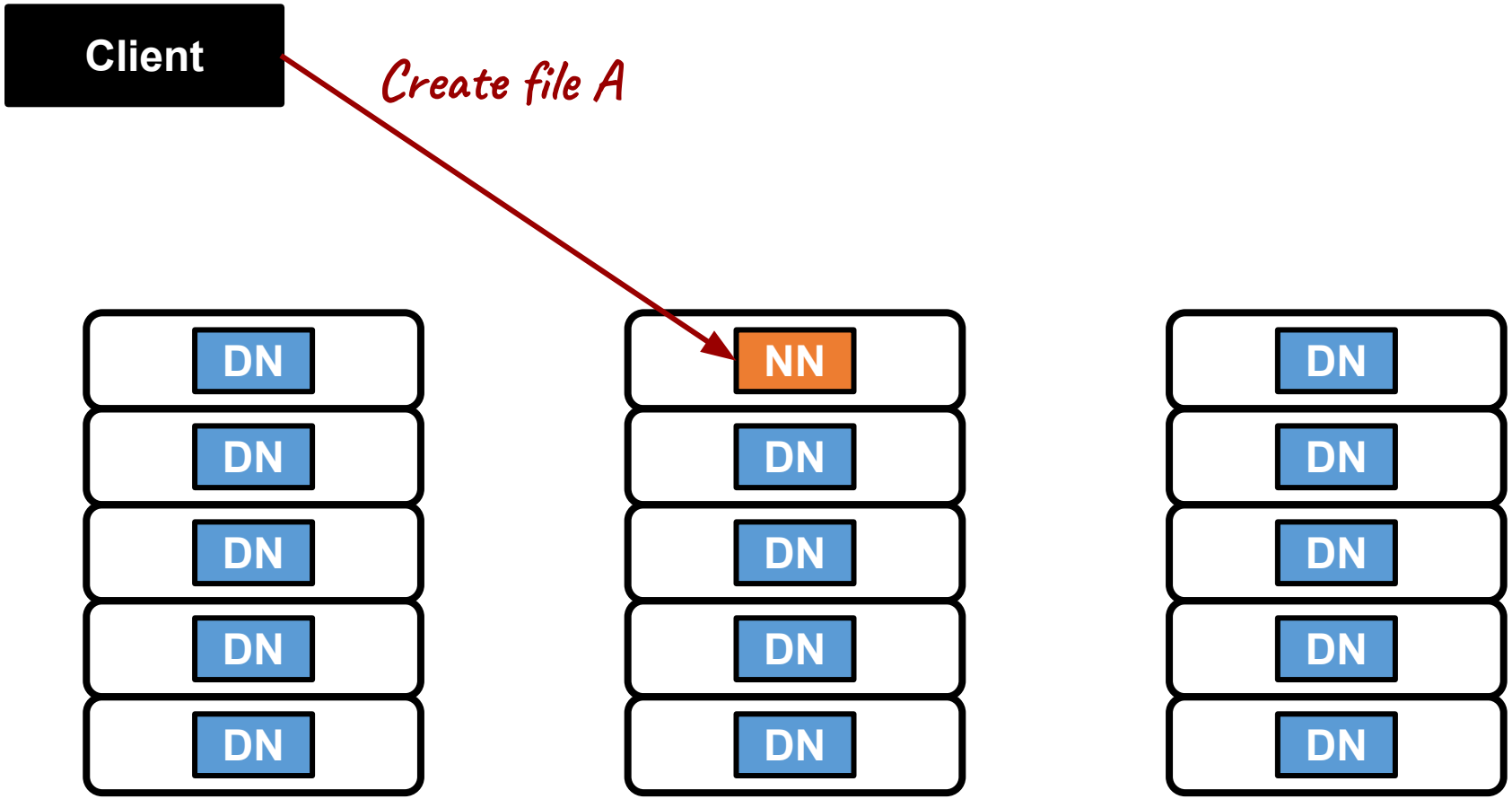

switch
Organized in racks
(communication is faster
inside the same rack)

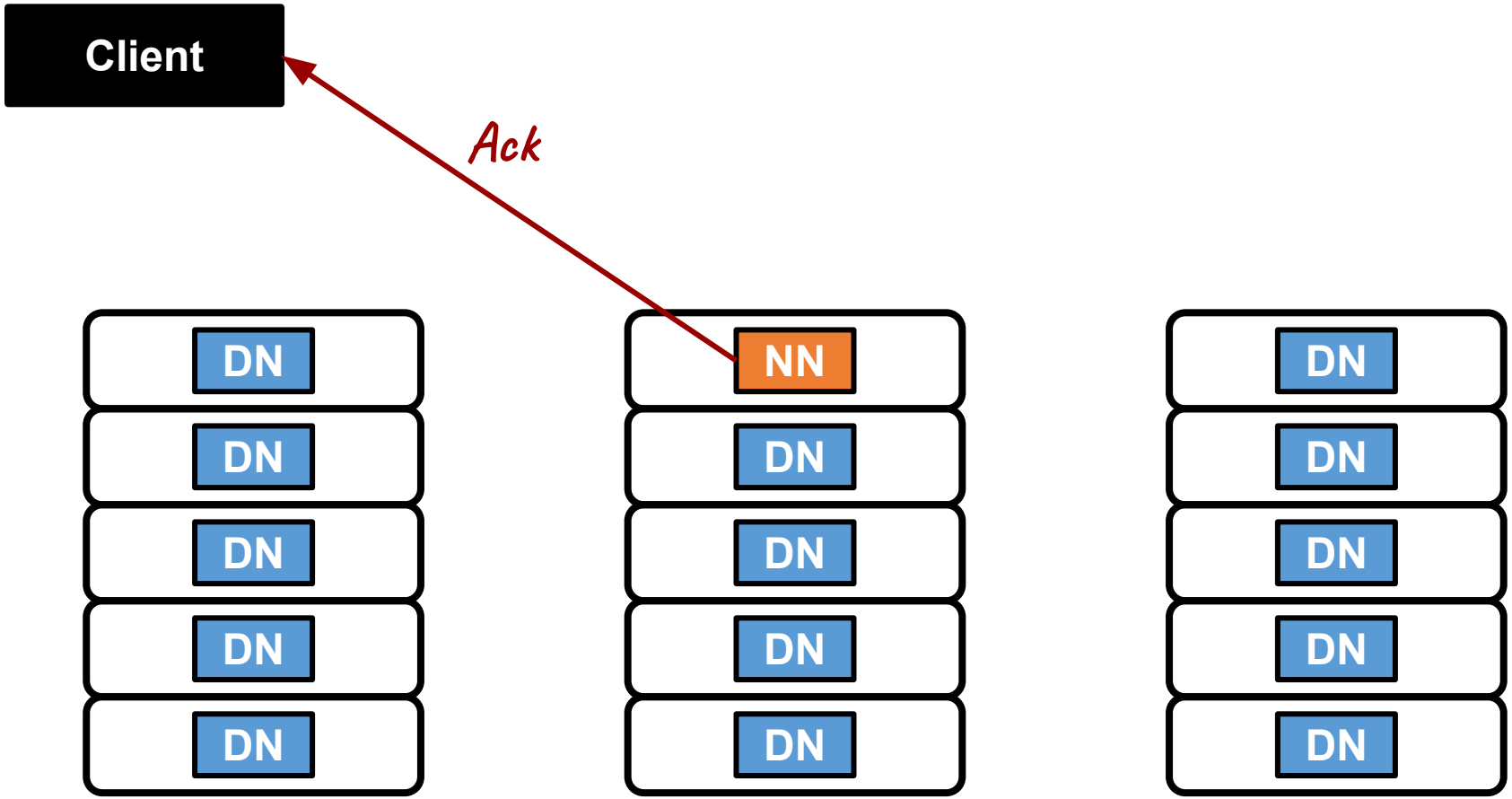

switch
DN
DN
DN
DN
DN
NN
DN
DN
DN
DN
DN
DN
DN
DN
DN

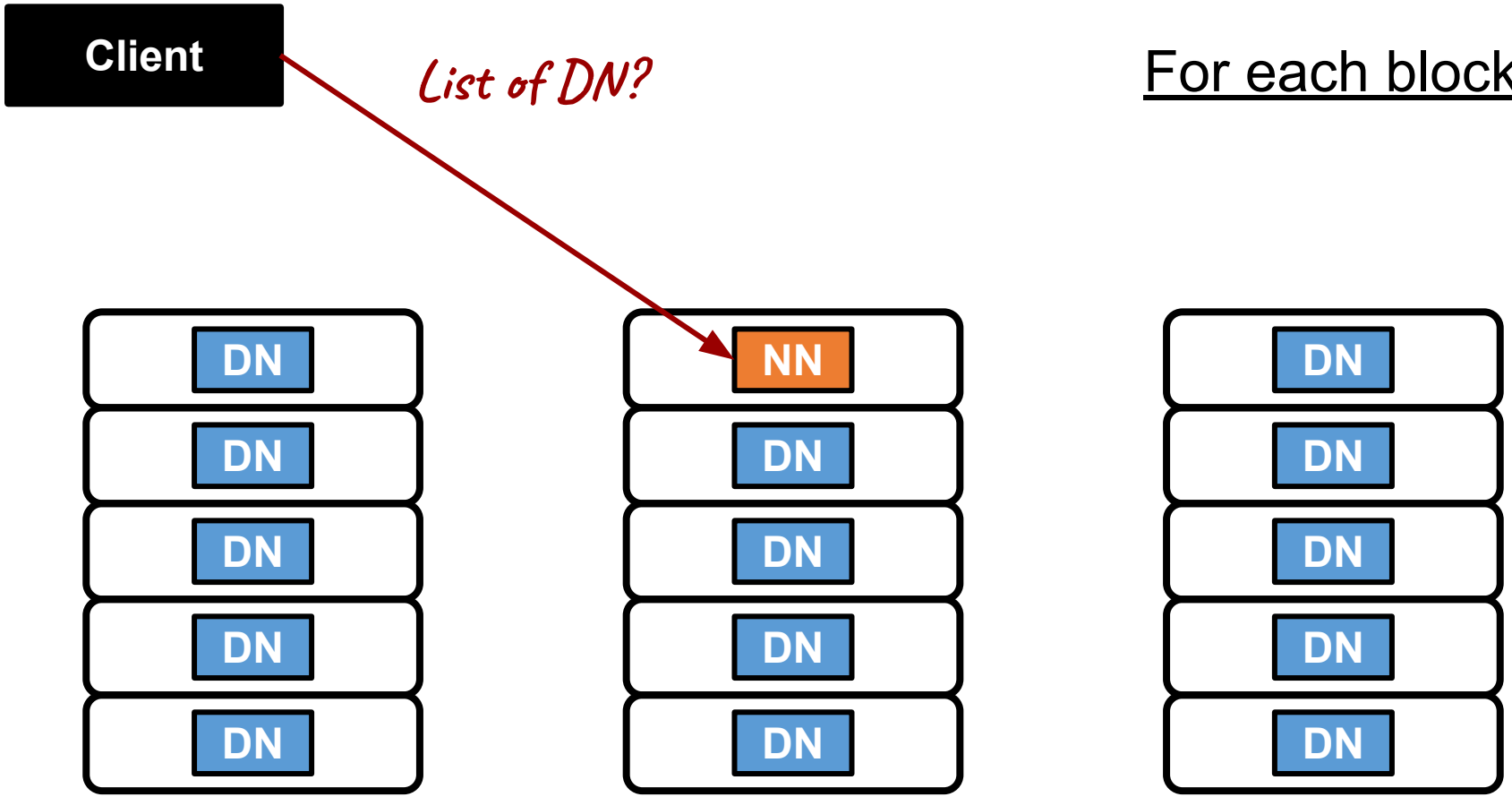
Writing a file

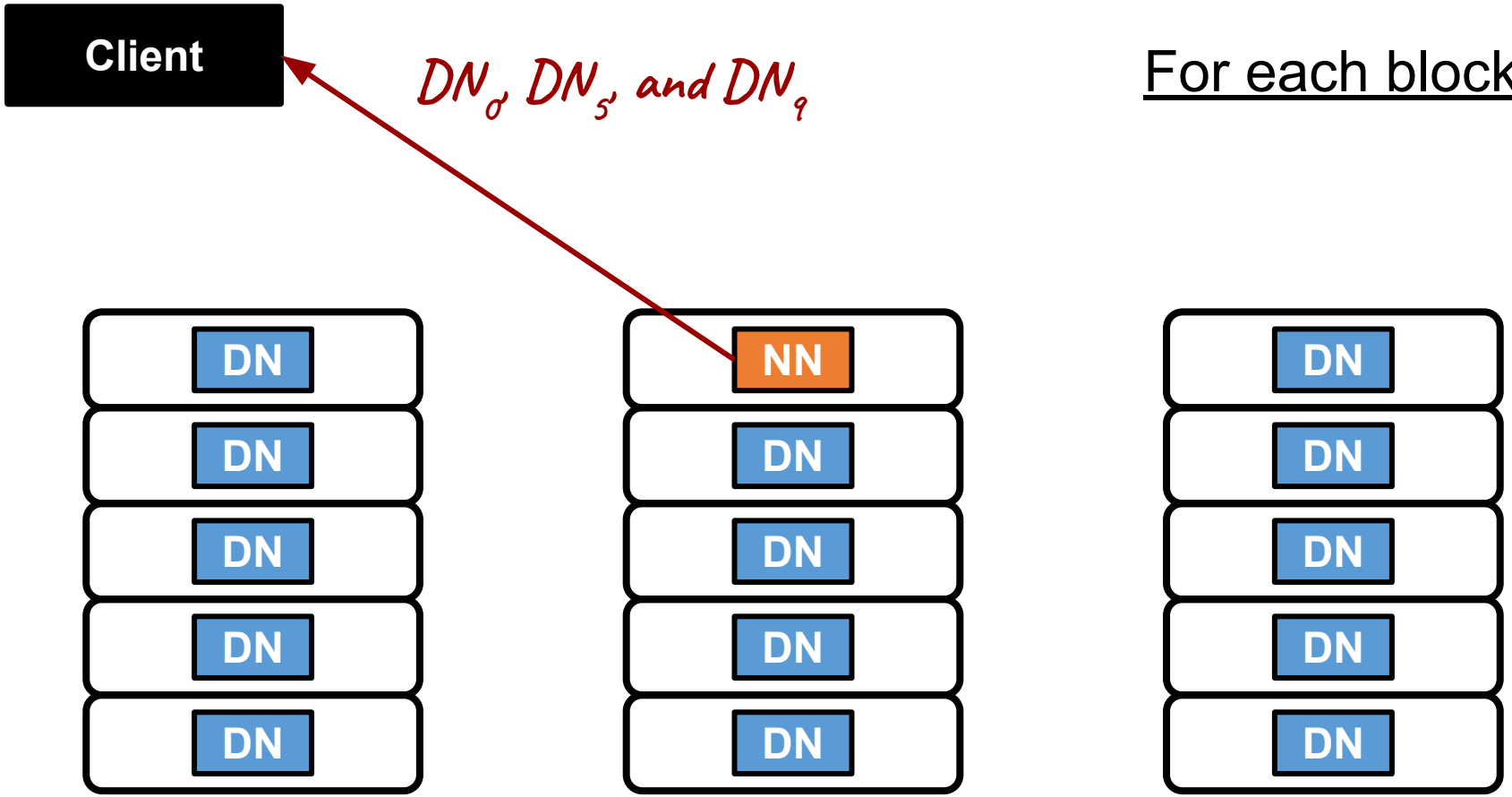
- The client asks the NameNode
  - The NameNode checks permissions, etc
- The NameNode allows the client to proceed
- The client breaks data into blocks
  - To each block, the client asks the NameNode for a list of destination DataNodes
  - The NameNode returns a list sorted by distance to the client
- Each block is written:
  - The client sends it to the first (closest) DataNode
  - Each DataNode forwards it to the next DataNode in the list (to create the replicas)
- When it is all written, the client acknowledges the file creation to the NameNode
- The NameNode saves information about the file (metadata) to disk

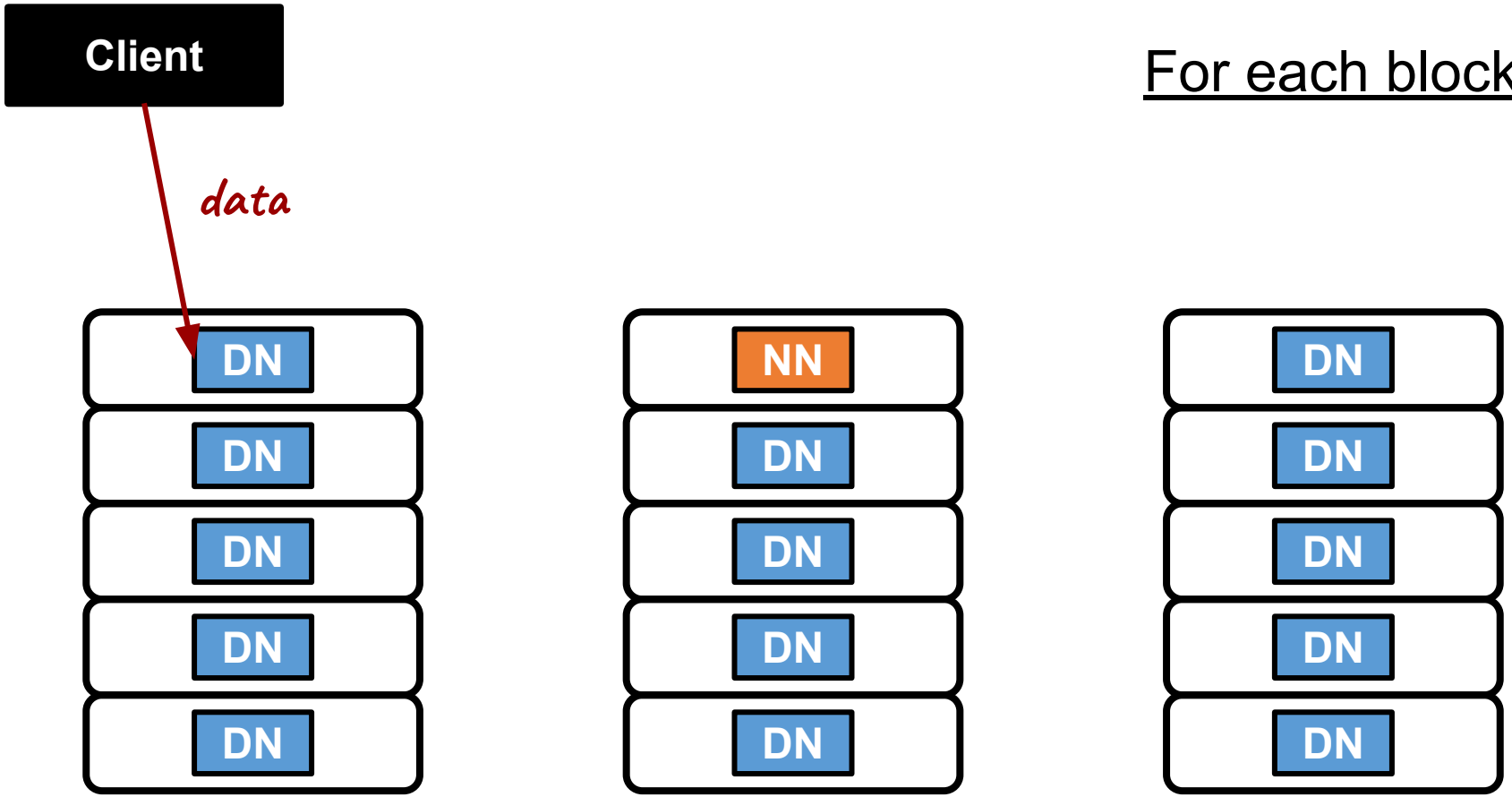

switch
DN
DN
DN
DN
DN
NN
DN
DN
DN
DN
DN
DN
DN
DN
DN

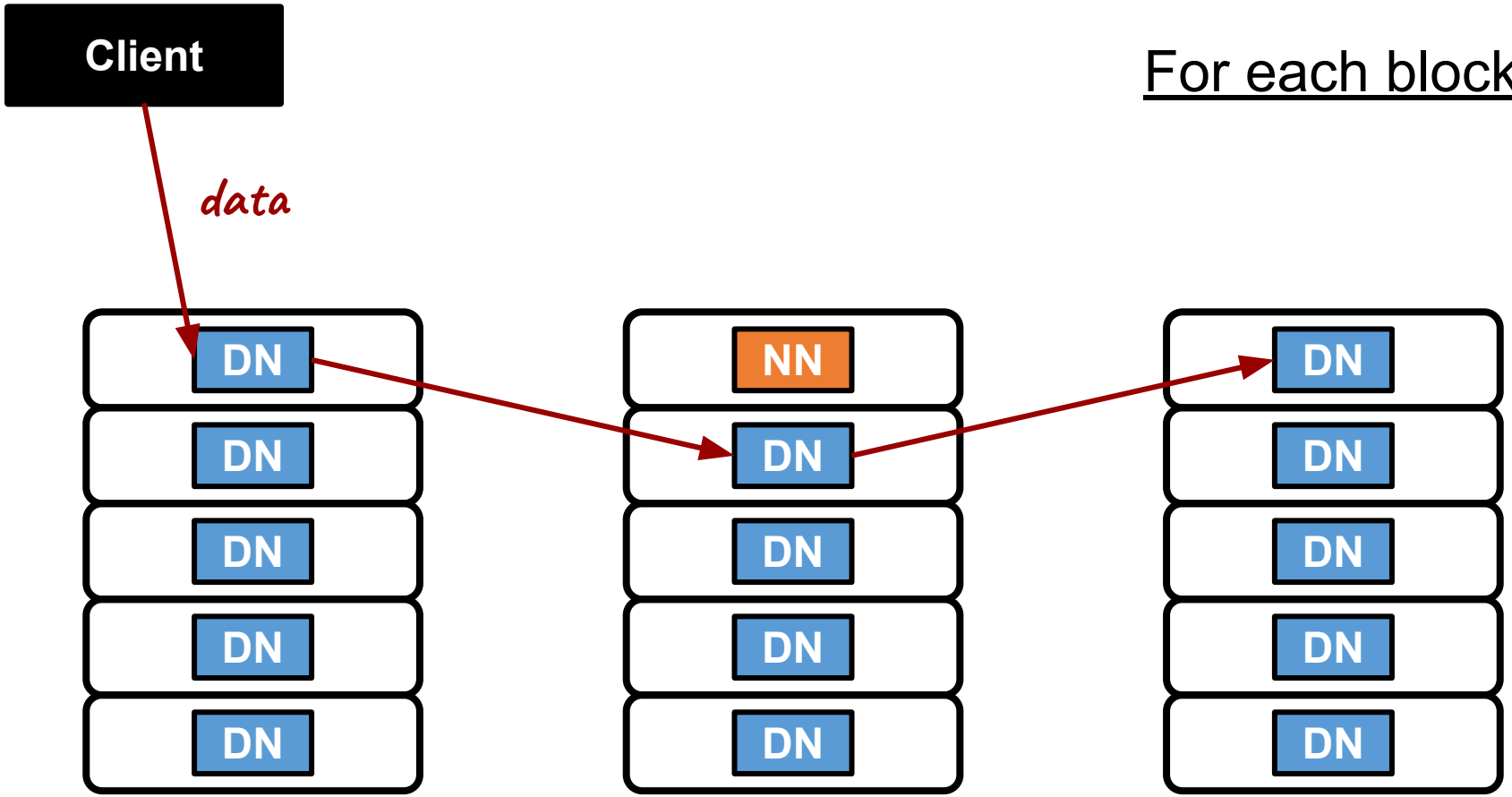

Client
Create file A
DN
DN
DN
DN
DN
NN
DN
DN
DN
DN
DN
DN
DN
DN
DN

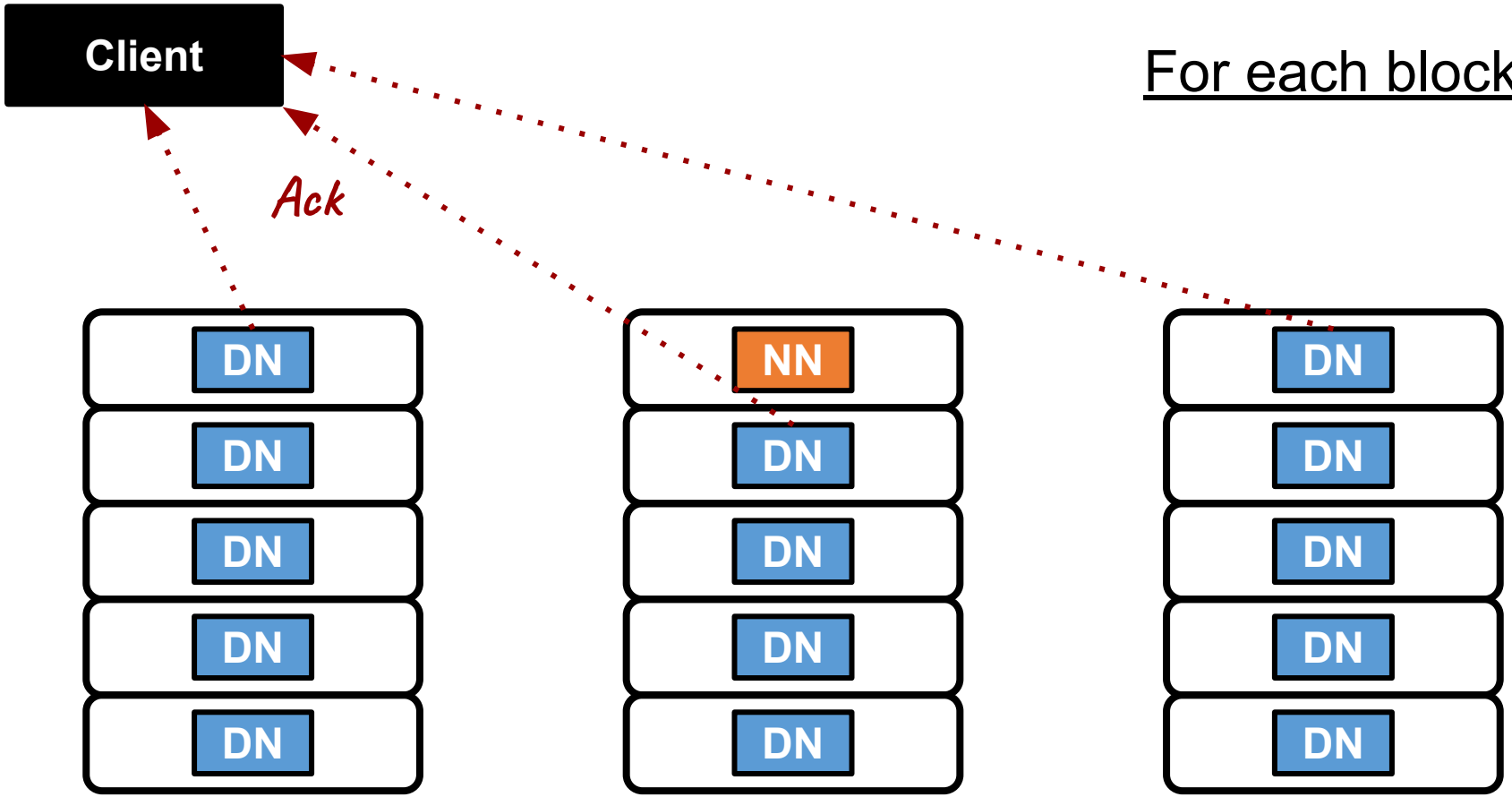

Client
Ack
DN
DN
DN
DN
DN
NN
DN
DN
DN
DN
DN
DN
DN
DN
DN

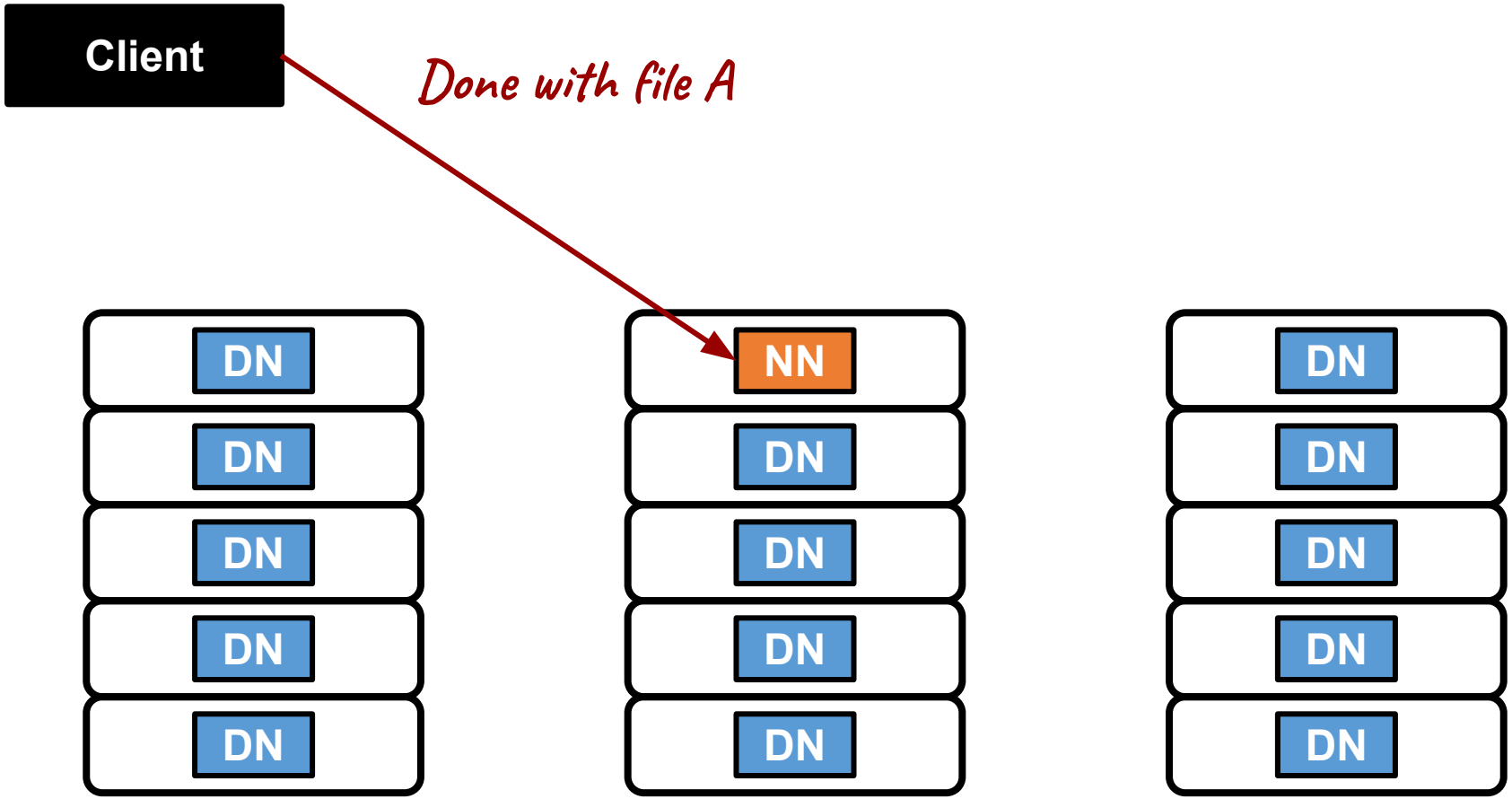

Client
List of DN?
For each block!
DN
DN
DN
DN
DN
NN
DN
DN
DN
DN
DN
DN
DN
DN
DN


Client
DN_0, DN_5, and DN_9
For each block!
DN
DN
DN
DN
DN
NN
DN
DN
DN
DN
DN
DN
DN
DN
DN


Client
data
DN
DN
DN
DN
DN
NN
DN
DN
DN
DN
DN
DN
DN
DN
DN
For each block!

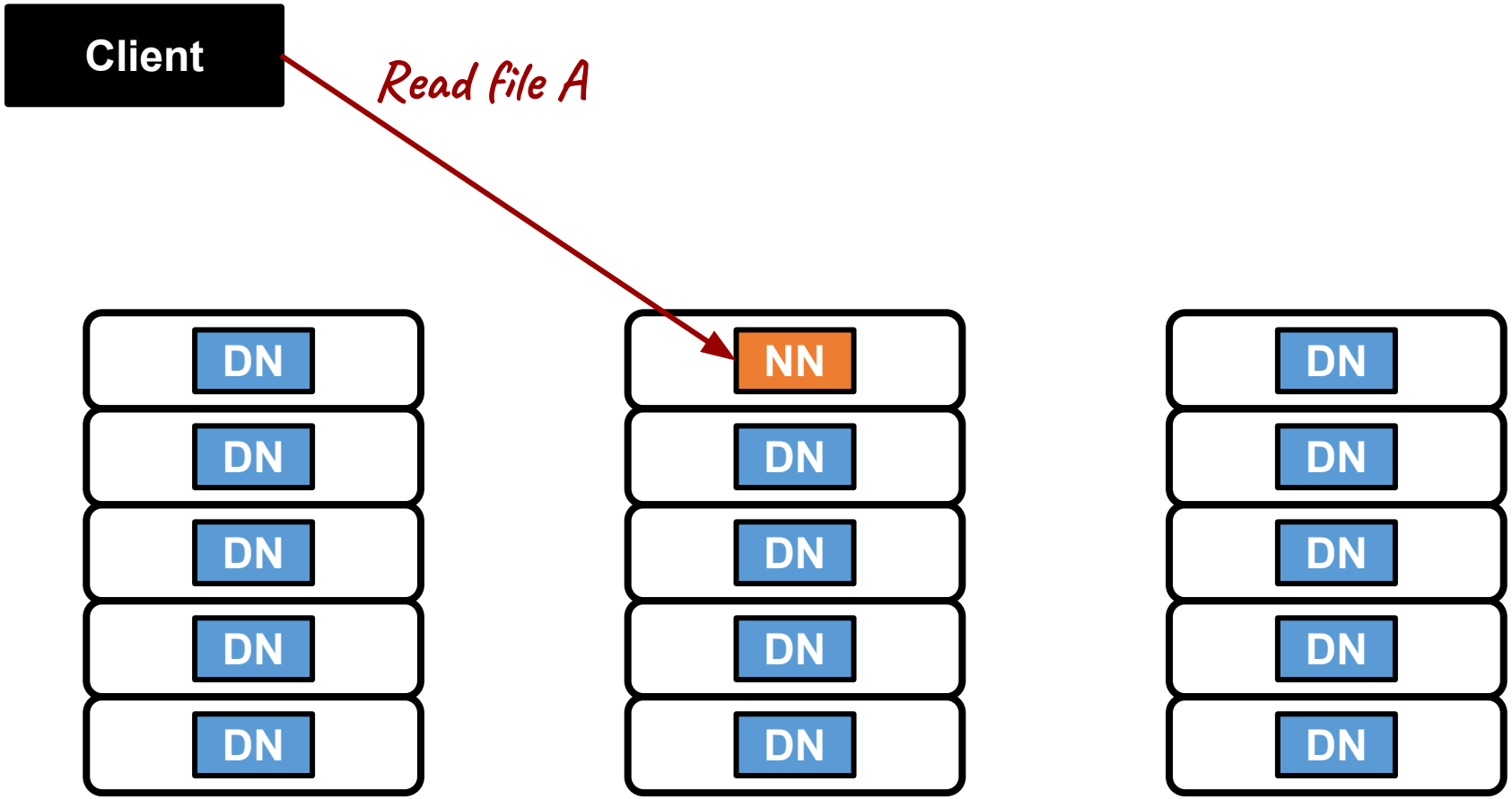

Client
data
DN
DN
DN
DN
DN
NN
DN
DN
DN
DN
DN
DN
DN
DN
DN
For each block!

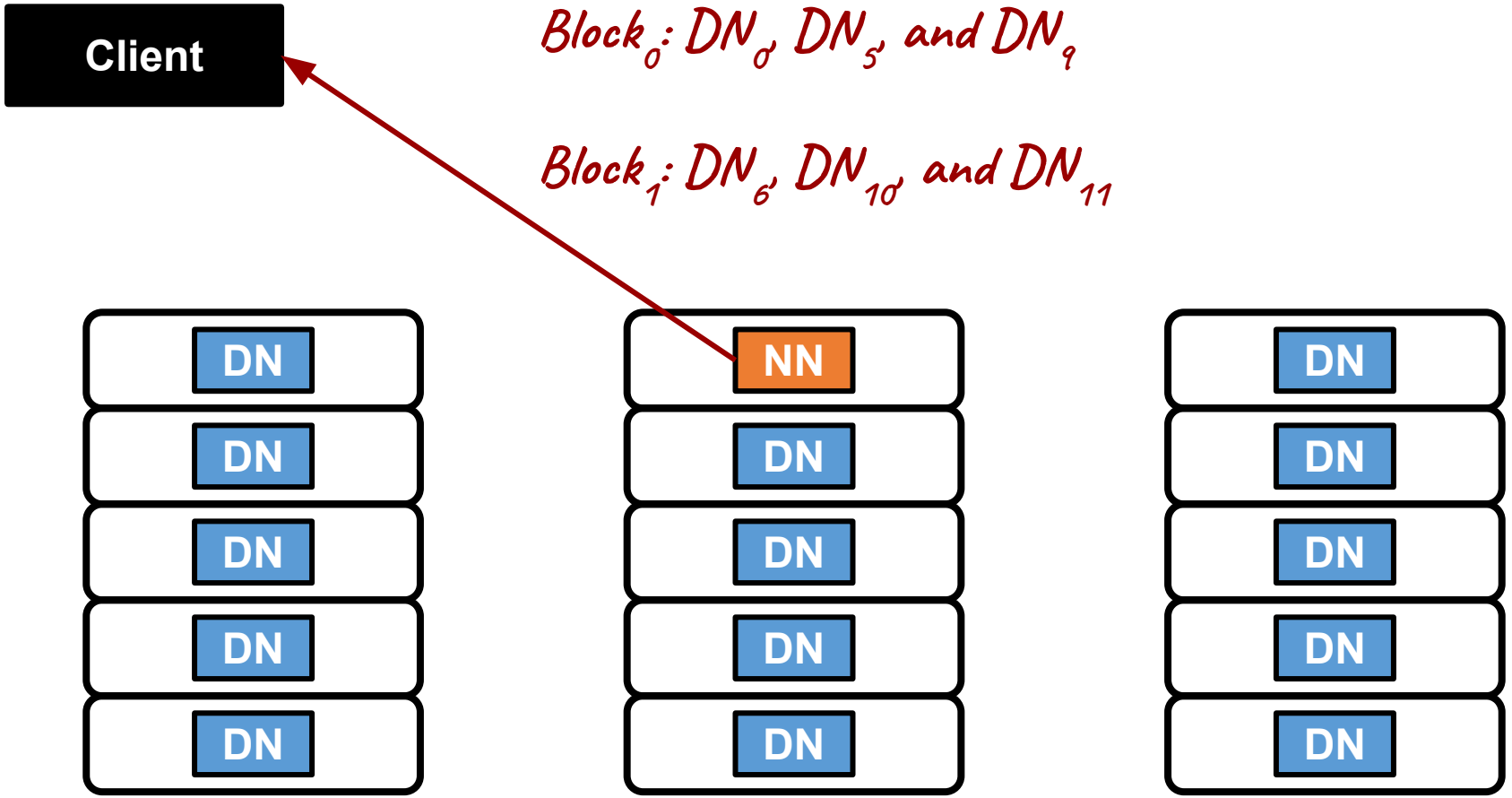

Client
Ack
For each block!
DN
DN
DN
DN
DN
NN
DN
DN
DN
DN
DN
DN
DN
DN
DN

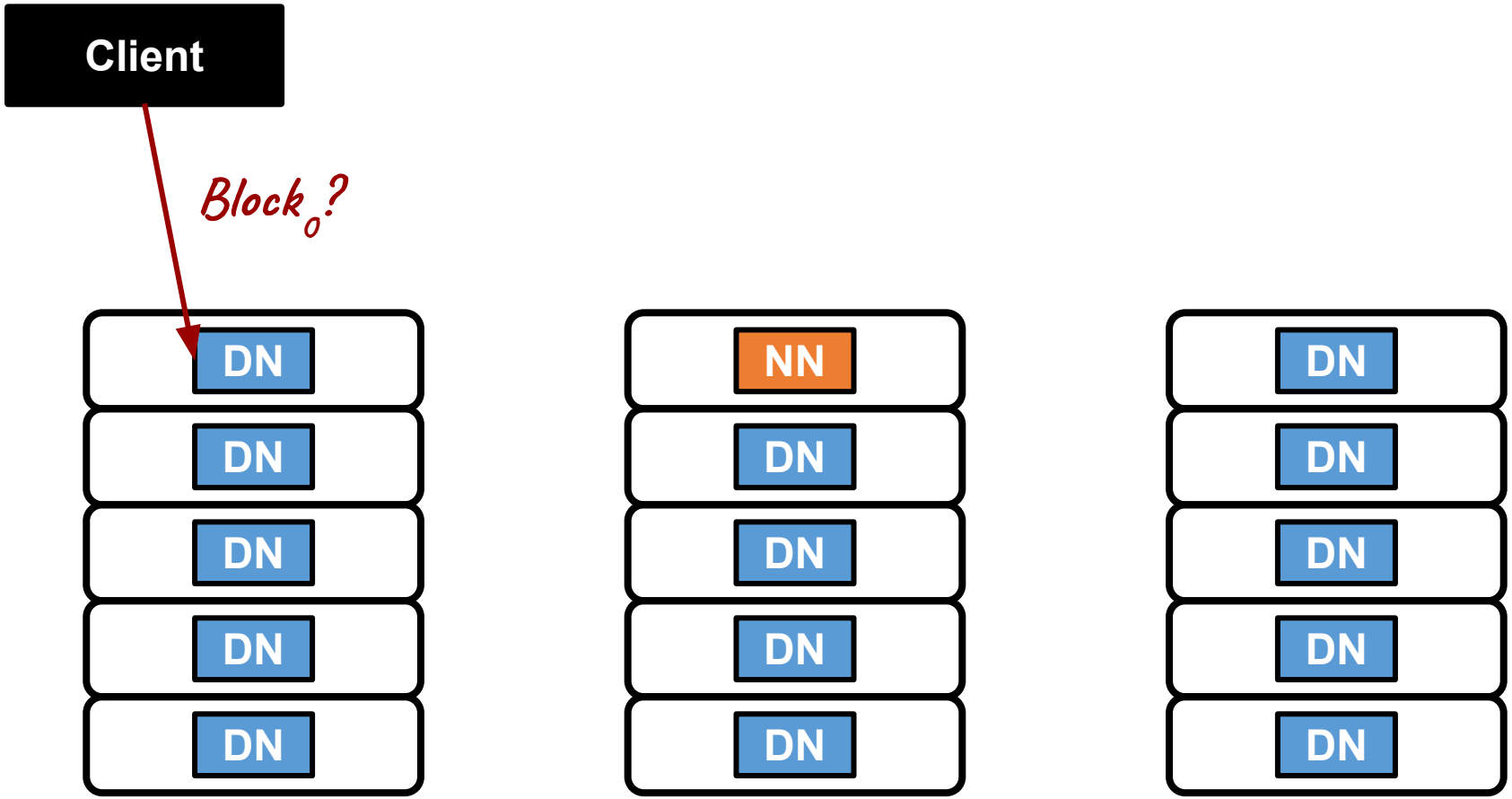

Client
Done with file A
DN
DN
DN
DN
DN
NN
DN
DN
DN
DN
DN
DN
DN
DN
DN

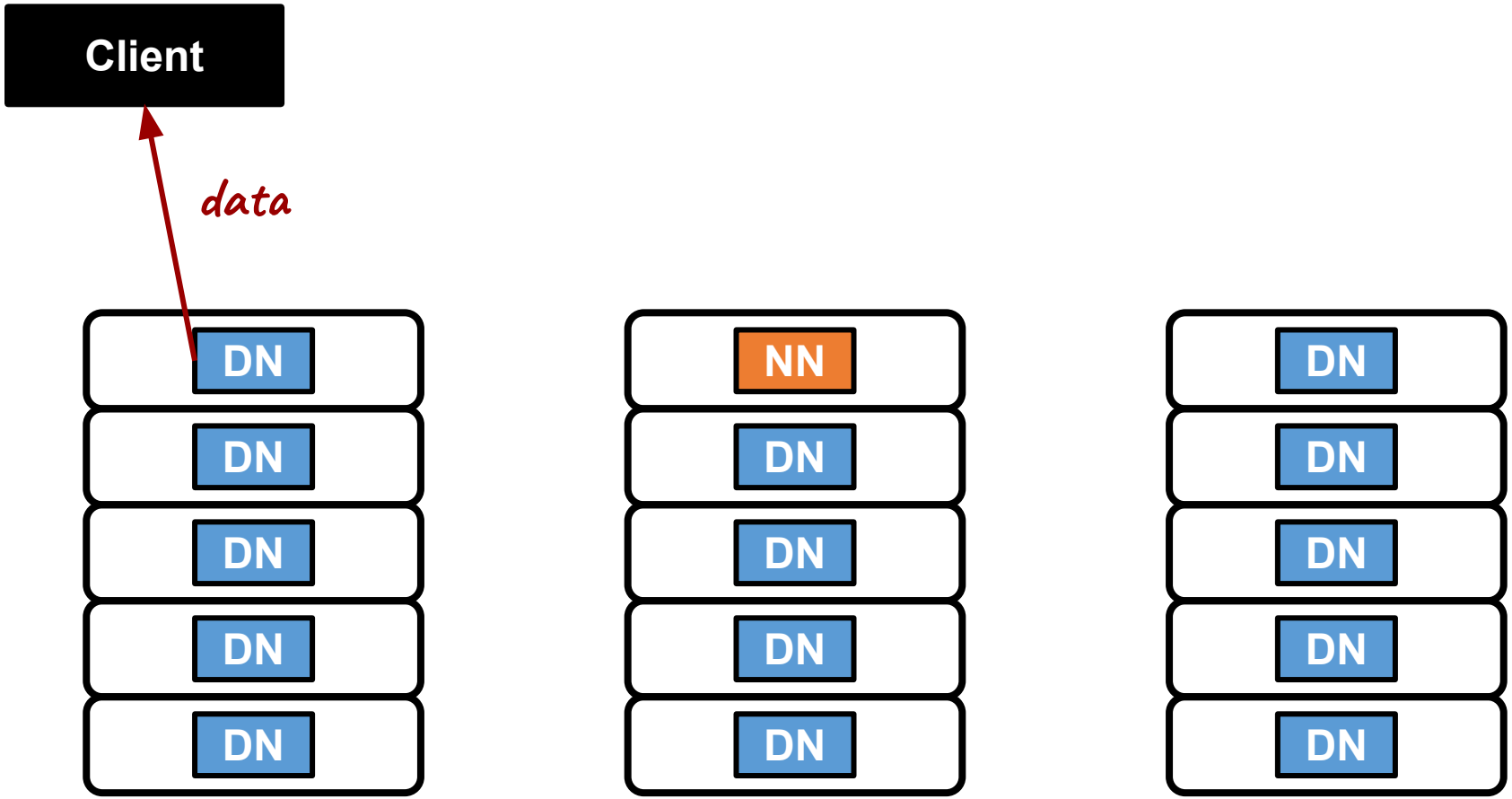
Reading a file

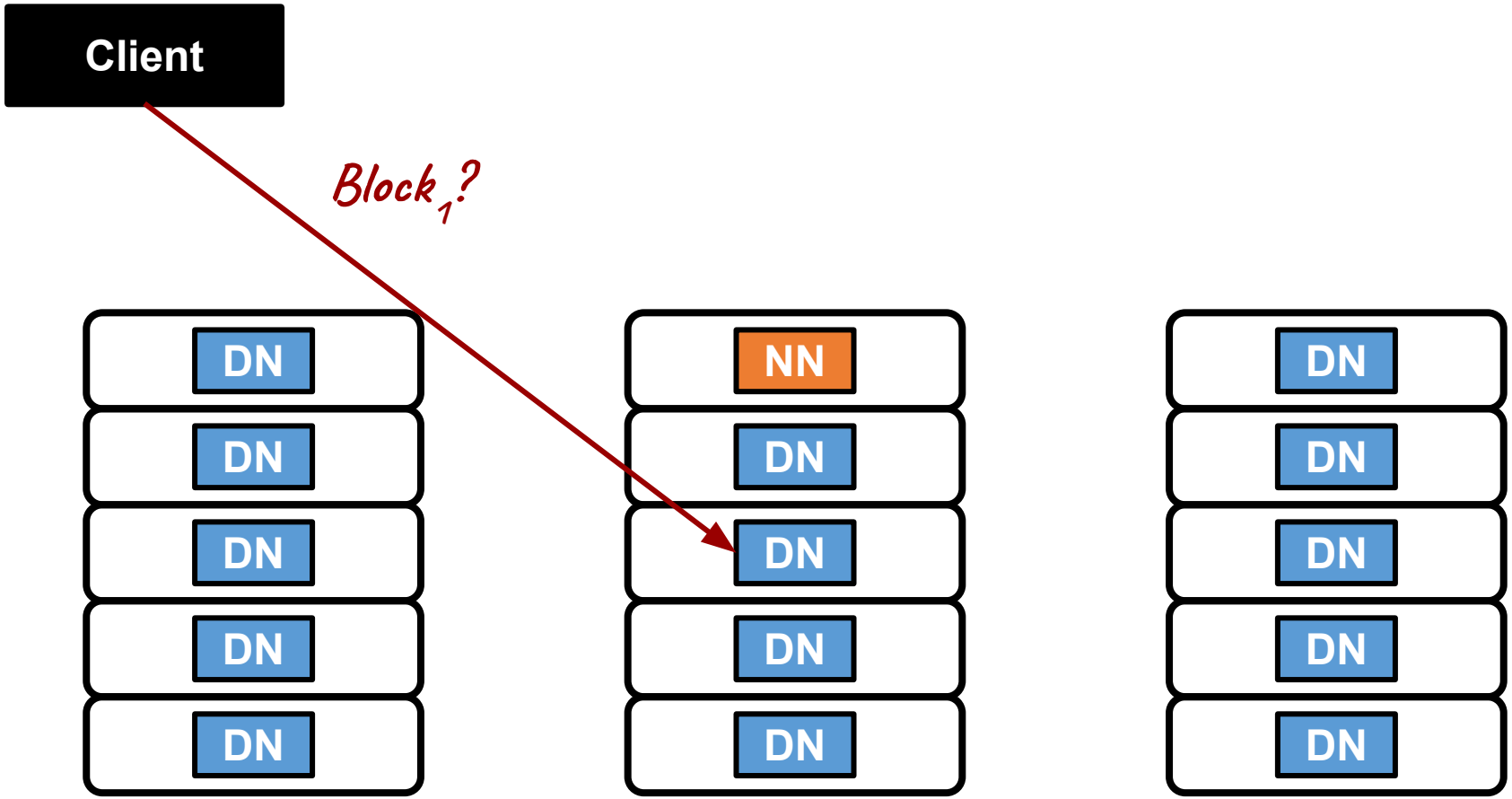
- The client asks the NameNode for information about the file
- The NameNode gives the client a list of blocks
  - To each block, a list of DataNodes that have that block, sorted by distance to the client
- The client reads the blocks sequentially
  - Tries to read each block from the closest DataNode, if it is not available try the others

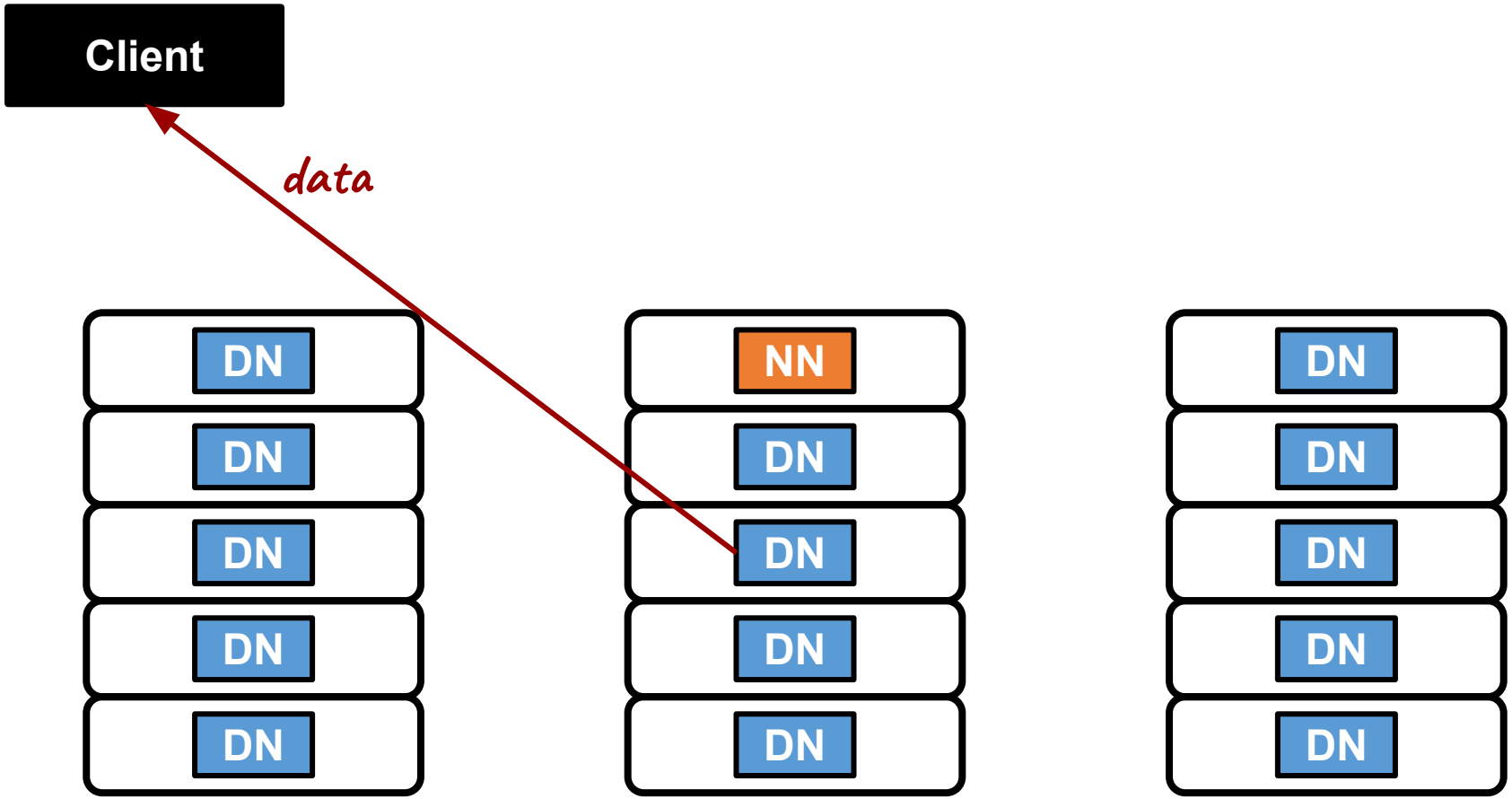

Client
Read file A
DN
DN
DN
DN
DN
NN
DN
DN
DN
DN
DN
DN
DN
DN
DN


Client
Block0: DN0, DN5, and DN9
Block1: DN6, DN10, and DN11
DN
DN
DN
DN
DN
NN
DN
DN
DN
DN
DN
DN
DN
DN
DN


Client
Block0?
DN
DN
DN
DN
DN
NN
DN
DN
DN
DN
DN
DN
DN
DN
DN


Client
data
DN
DN
DN
DN
DN
NN
DN
DN
DN
DN
DN
DN
DN
DN
DN

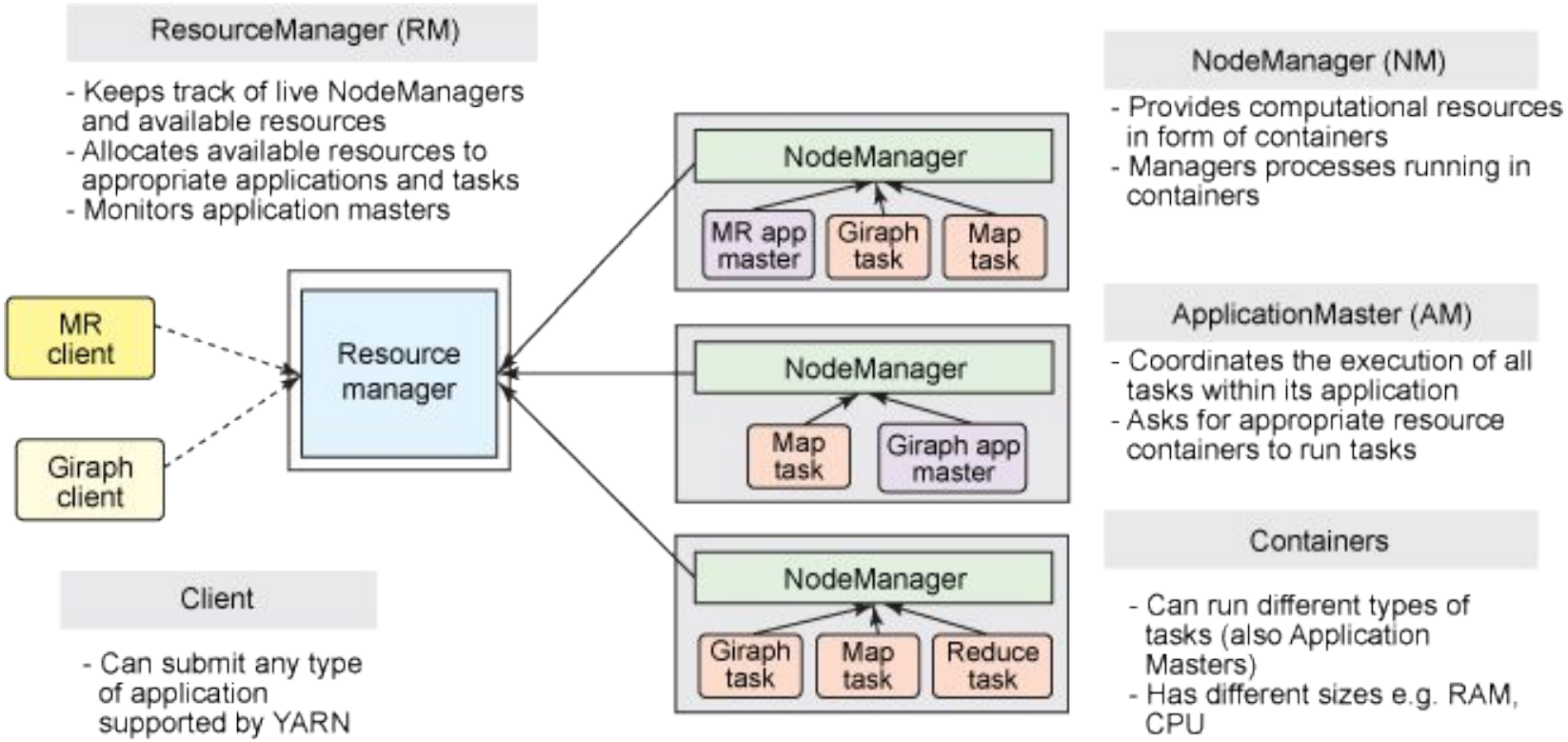

Client
Block 1?
DN
DN
DN
DN
DN
NN
DN
DN
DN
DN
DN
DN
DN
DN
DN


Client
data
DN
DN
DN
DN
DN
NN
DN
DN
DN
DN
DN
DN
DN
DN
DN

Yarn

- The cluster resource manager: dynamically allocates resources to jobs
- Multiple engines (in addition to MapReduce) run in parallel on the cluster
- Hierarchical architecture for scalability

Yarn architecture



Source: https://www.ibm.com/developerworks/library/bd-yarn-intro/index.html

Agenda

- Introduction
- A first MapReduce program
- Apache Hadoop
  - MapReduce
  - HDFS
  - Yarn
- **Combiners**

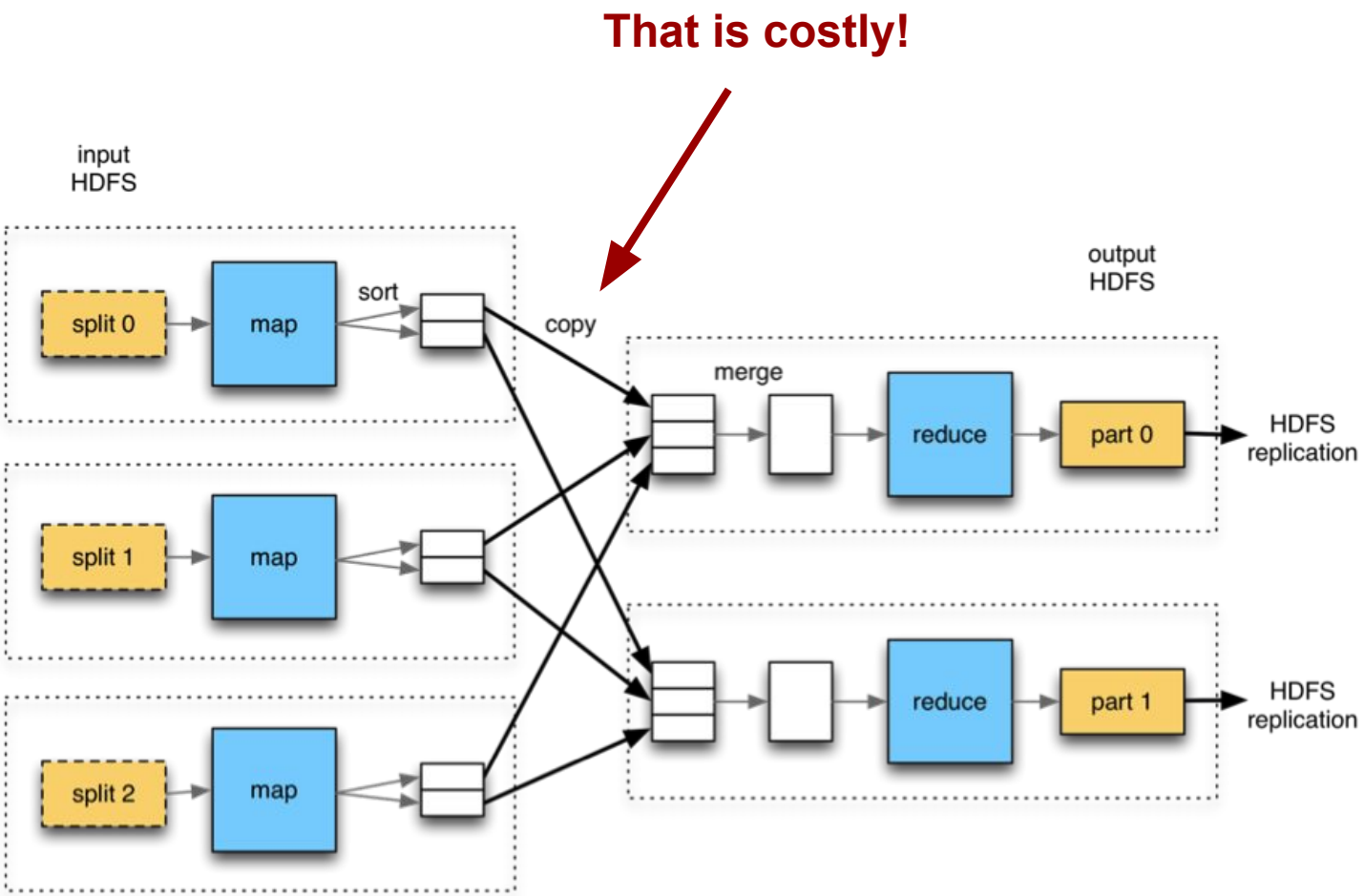
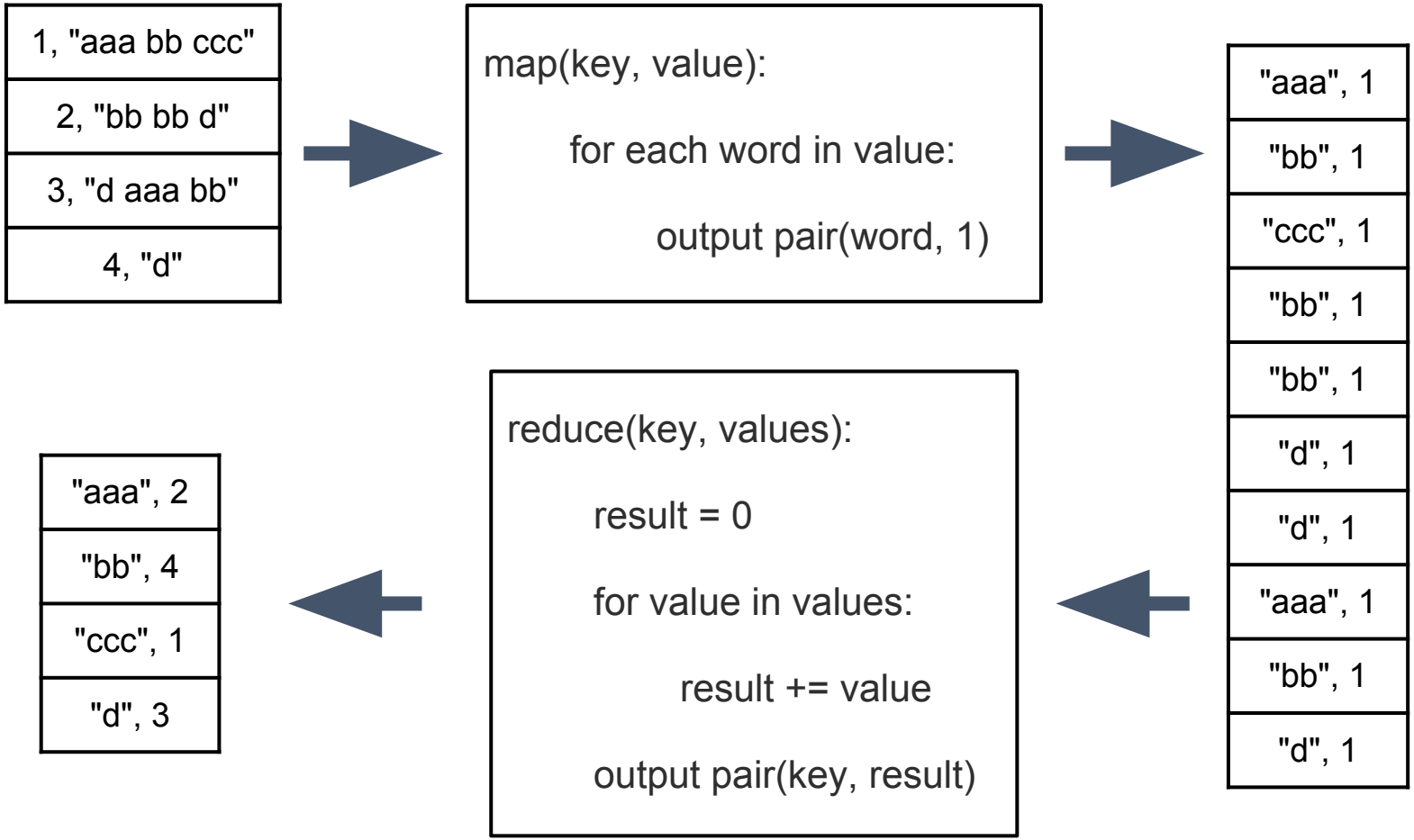
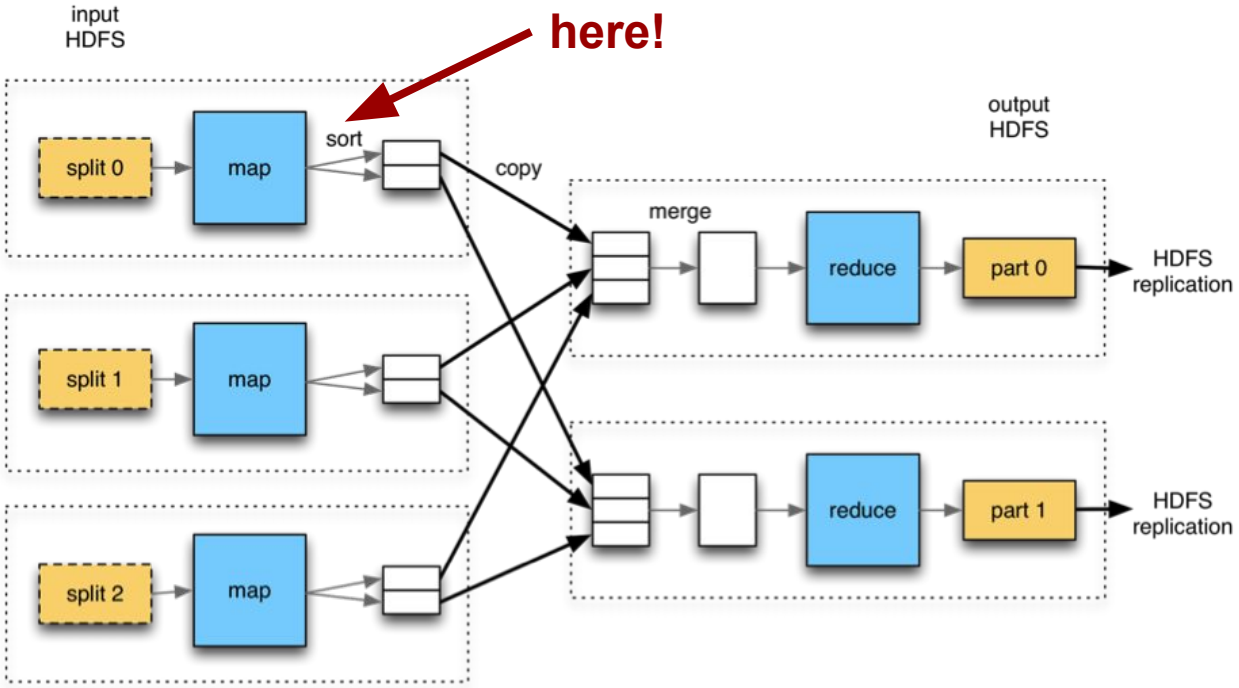


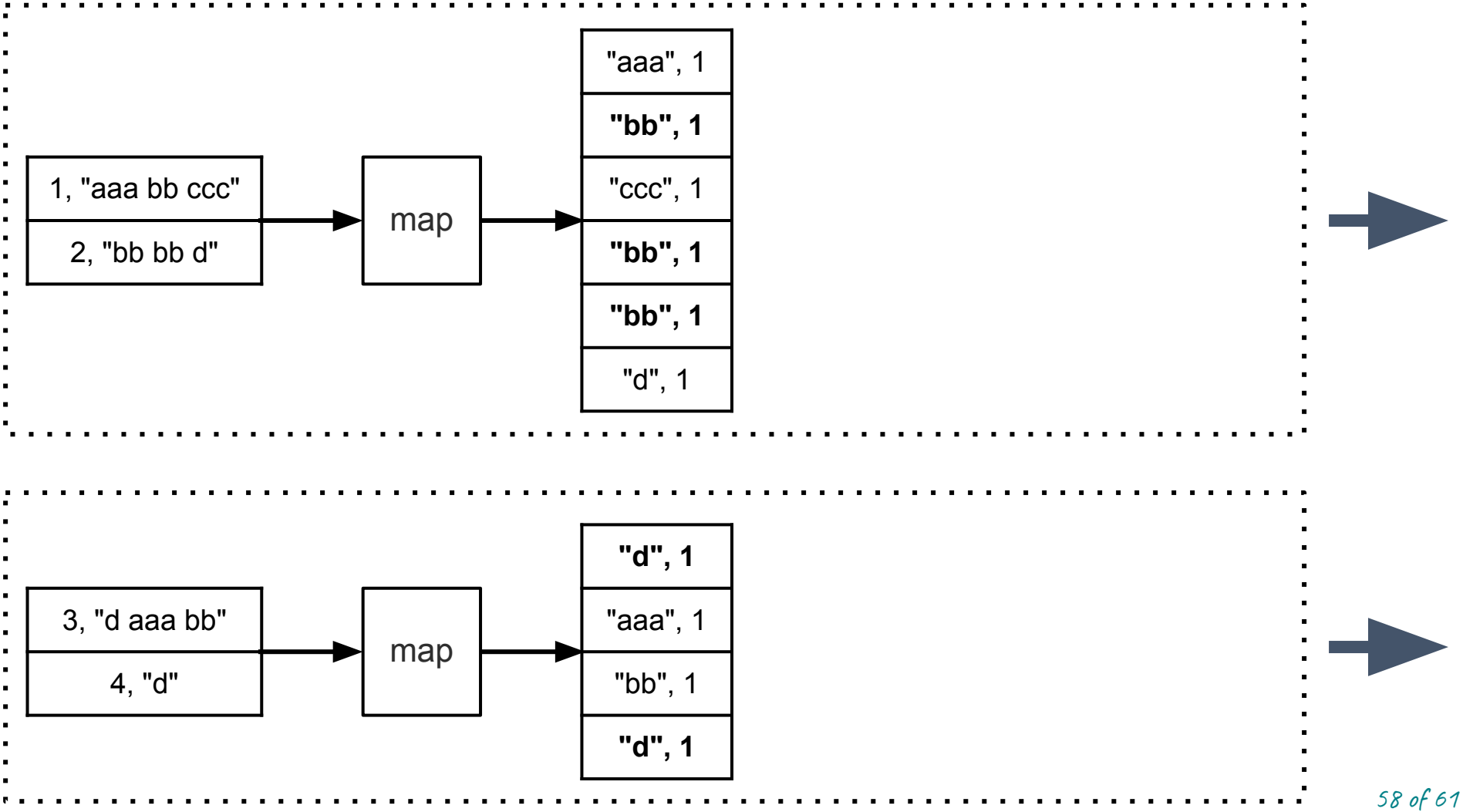
Figure from https://www.supinfo.com/articles/single/2807-introduction-to-the-mapreduce-life-cycle

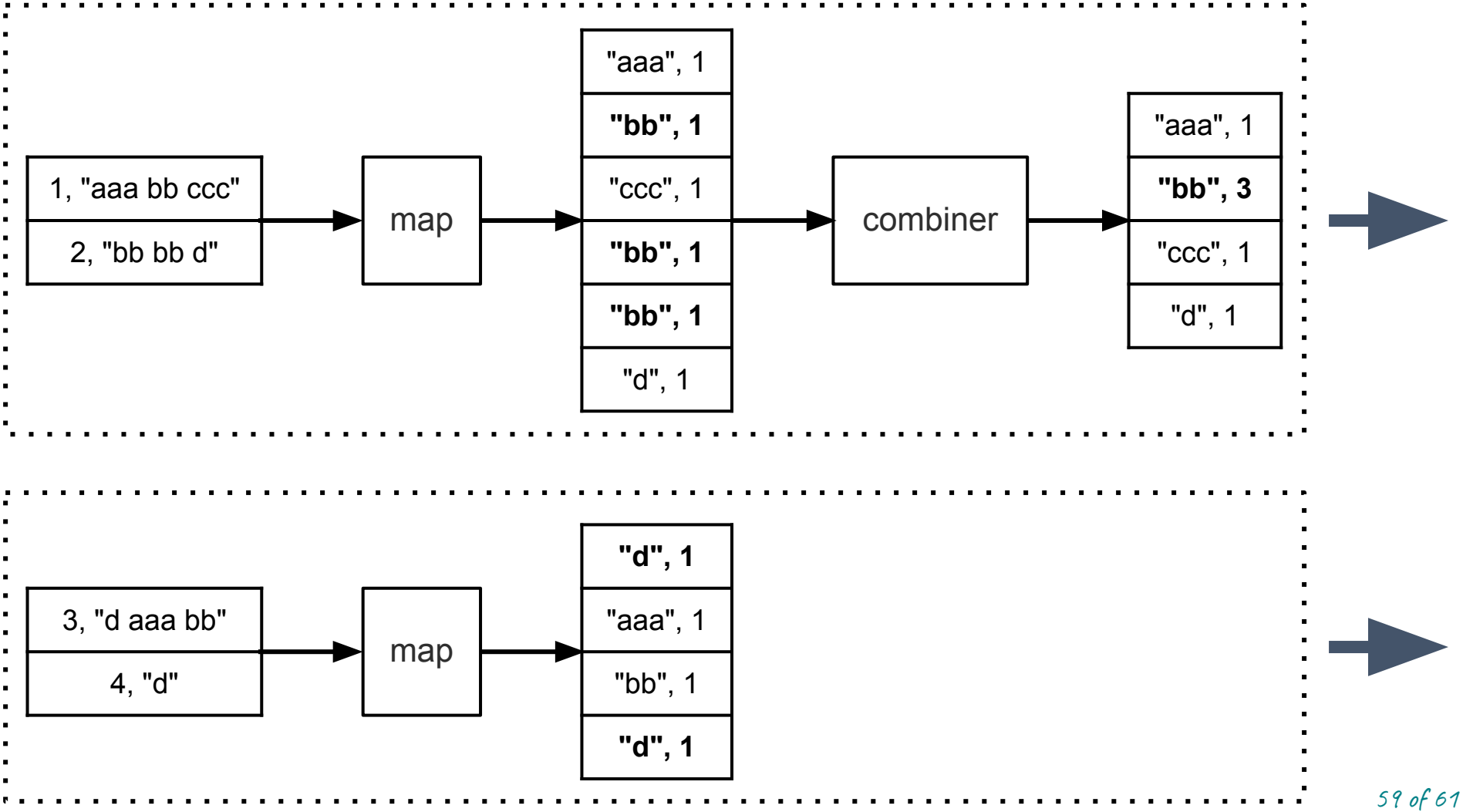

1, "aaa bb ccc"
2, "bb bb d"
3, "d aaa bb"
4, "d"
map(key, value):
for each word in value:
output pair(word, 1)
"aaa", 1
"bb", 1
"ccc", 1
"bb", 1
"bb", 1
"d", 1
"d", 1
"aaa", 1
"bb", 1
"d", 1
reduce(key, values):
result = 0
for value in values:
result += value
output pair(key, result)
"aaa", 2
"bb", 4
"ccc", 1
"d", 3

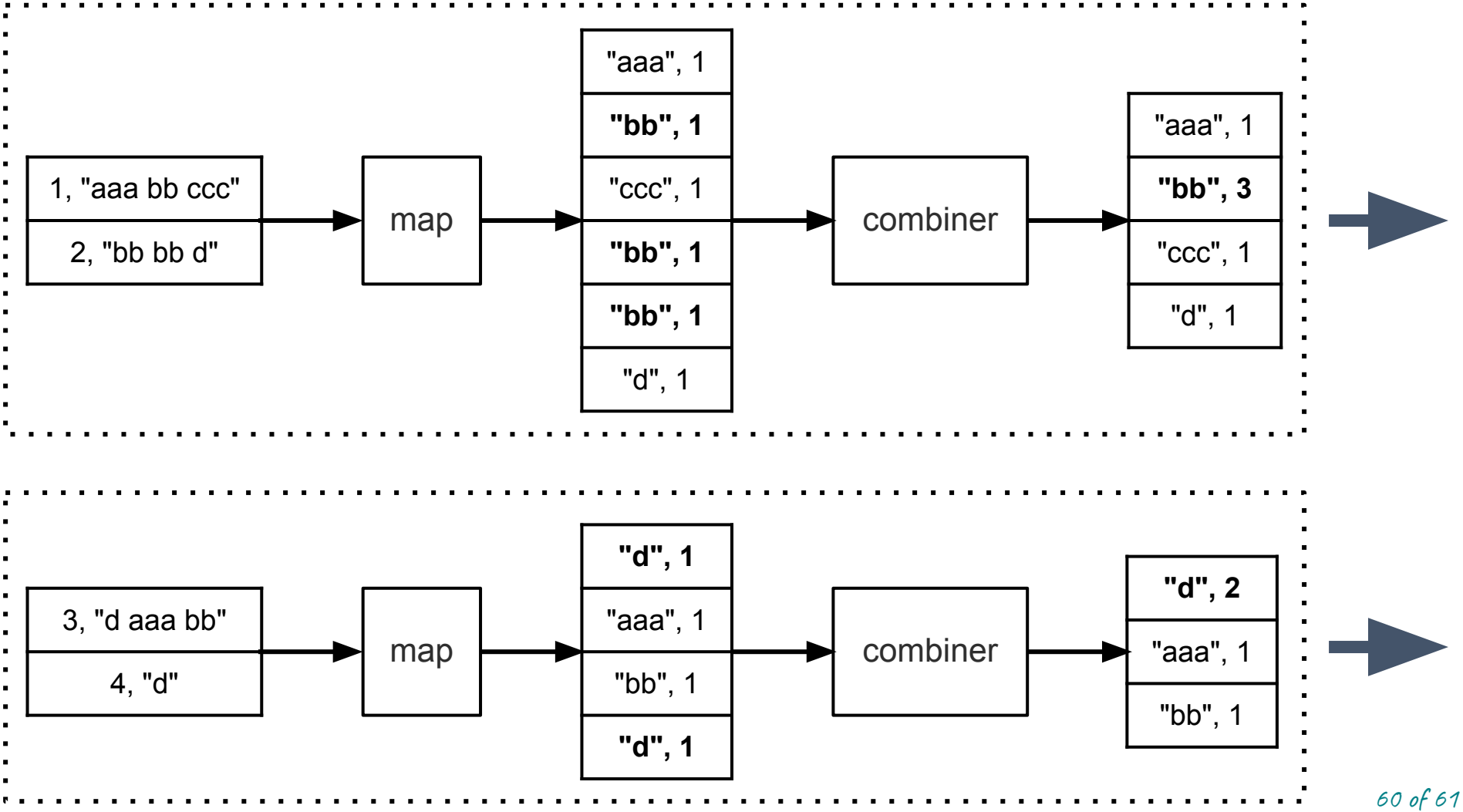
Combiner

User-defined function for local aggregation on the map tasks




1, "aaa bb ccc"
2, "bb bb d"
map
"aaa", 1
"bb", 1
"ccc", 1
"bb", 1
"bb", 1
"d", 1
3, "d aaa bb"
4, "d"
map
"d", 1
"aaa", 1
"bb", 1
"d", 1


1, "aaa bb ccc"
2, "bb bb d"
map
"aaa", 1
"bb", 1
"ccc", 1
"bb", 1
"bb", 1
"d", 1
combiner
"aaa", 1
"bb", 3
"ccc", 1
"d", 1
3, "d aaa bb"
4, "d"
map
"d", 1
"aaa", 1
"bb", 1
"d", 1


1, "aaa bb ccc"
2, "bb bb d"
map
"aaa", 1
"bb", 1
"ccc", 1
"bb", 1
"bb", 1
"d", 1
combiner
"aaa", 1
"bb", 3
"ccc", 1
"d", 1
3, "d aaa bb"
4, "d"
map
"d", 1
"aaa", 1
"bb", 1
"d", 1
combiner
"d", 2
"aaa", 1
"bb", 1

Additional references

- MapReduce: Simplified Data Processing on Large Clusters, by J. Dean and S. Ghemawat.
- Suggested reading
  - Chapter 10 of Designing Data-Intensive Applications by Martin Kleppmann
  - HDFS Carton: https://wiki.scc.kit.edu/gridkaschool/upload/1/18/Hdfs-cartoon.pdf
  - MapReduce illustration: https://words.sdsc.edu/words-data-science/mapreduce