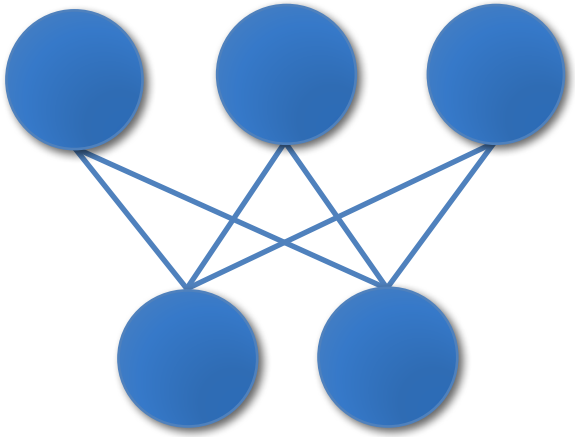
deep se
dependable evolvable pervasive software engineering group



Big Data Technologies and Applications

Danilo Ardagna

Politecnico di Milano

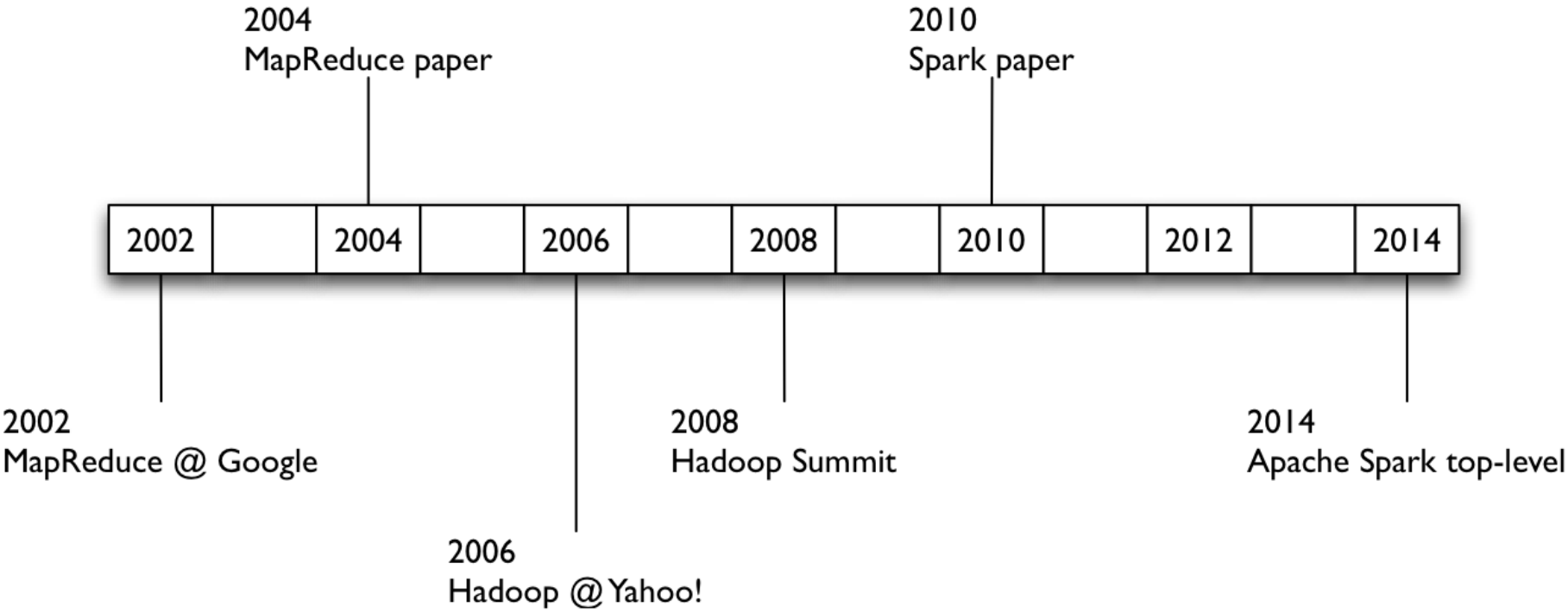
danilo.ardagna@polimi.it

POLITECNICO DI MILANO

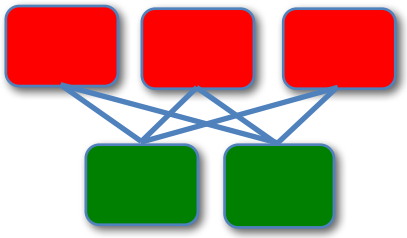
Content

- Big Data Analytics and Data Science
- Map-reduce and Hadoop fundamentals
- Map-reduce and Hadoop pros and cons
- Hadoop Success Stories
- Hadoop Evolution and Hadoop eco-system
- Pig and Hive
- Map Reduce Cloud based solutions
- Where is the world going? Spark Apache project

A Brief History



Map-reduce

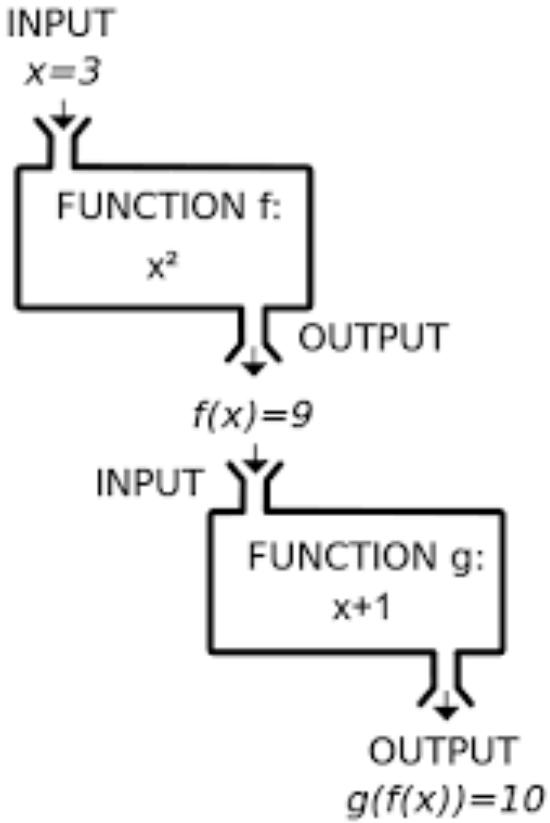


- Map-reduce: A general **algorithm**, and is prevalent in **functional programming** languages, which supports the notion of **map** and **reduce functions**

- MapReduce: The **patented software framework** from **Google** that the company applies in the realm of managing large datasets over clusters or other distributed topologies

- Hadoop: **Apache project open source** implementation of the MapReduce framework

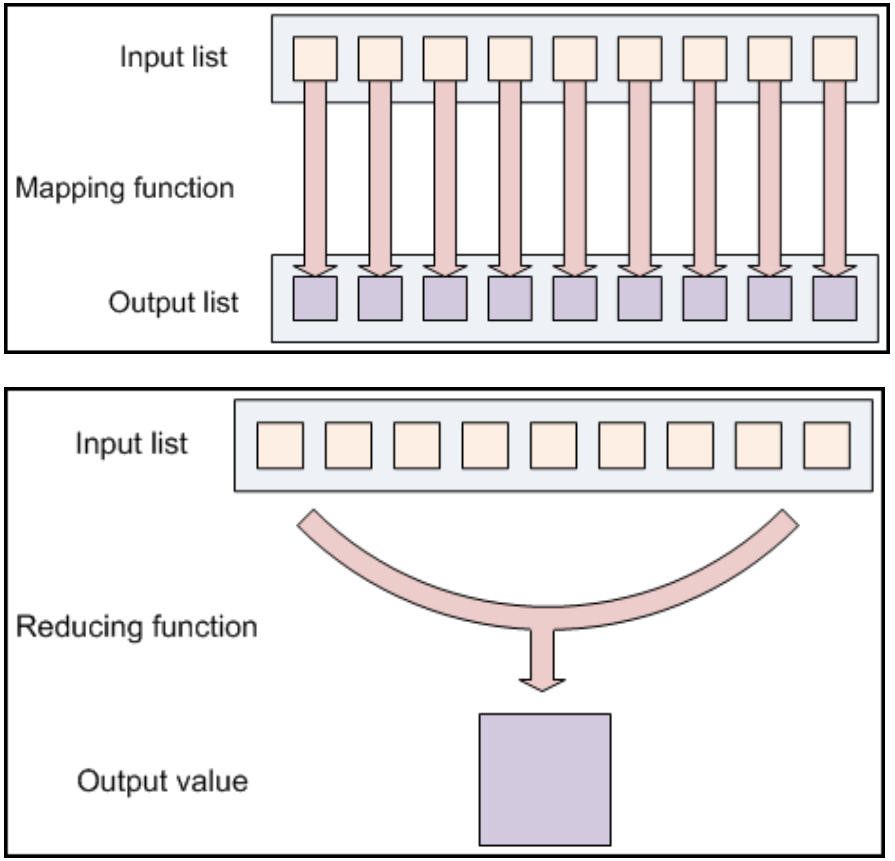
Functional programming

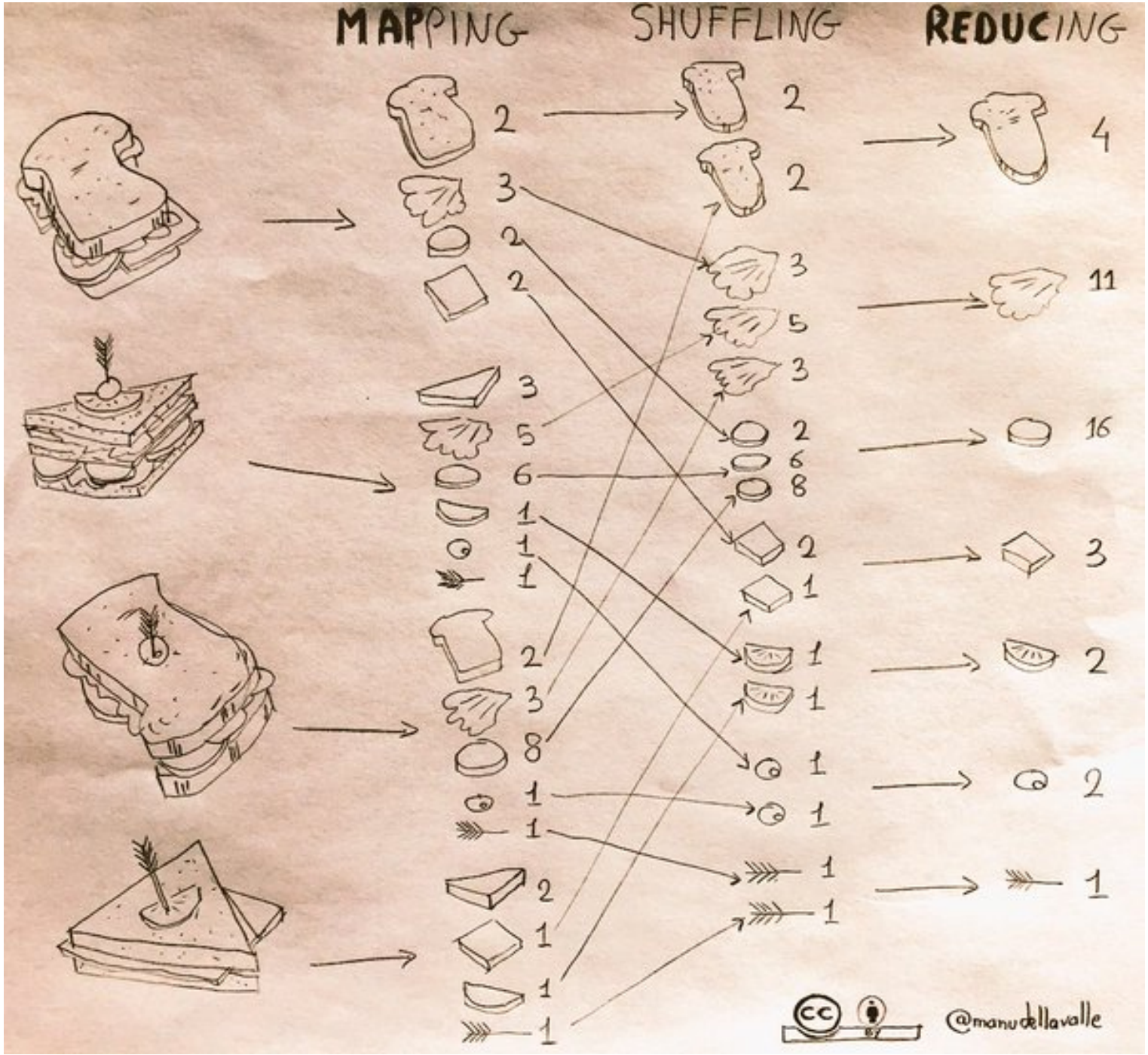


- Computation as **evaluation of** mathematical **functions avoiding changing-state and mutable data**
  - functions are **expressions** and running a program means evaluating such expressions to get a **value**
- Mathematical functions have no side effects
  - output value depends only on input arguments
  - calling a ***f*** twice with the same value for an argument ***x*** will produce the same result **f(x)**
  - eliminating side effects, i.e. changes in state that do not depend on the function inputs, can make much easier to understand and predict the behavior of a program
- Examples: **Scala**, Clojure, Haskell, Lisp

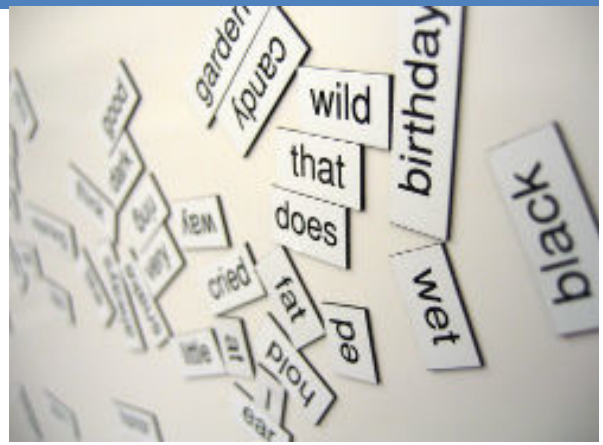
Function view

- Input: a **set** of **key/value pairs**
- User supplies two functions:
  - **map(k,v) → list(k1,v1)**
  - **reduce(k1, list(v1)) → v2**
- **(k1,v1)** is an **intermediate** key/value pair
- **Output** is the **set** of **(k1,v2)** pairs

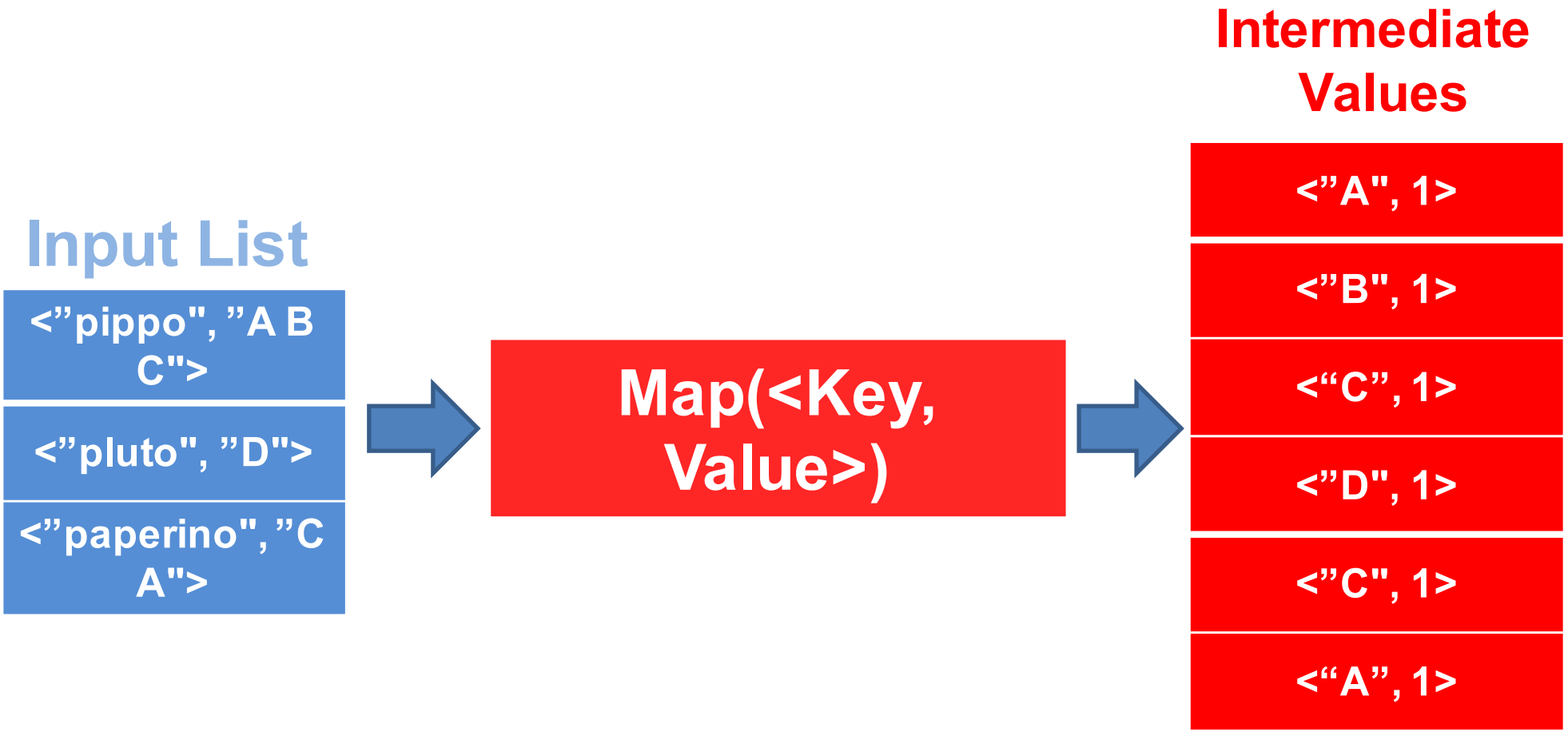



MAPPING
SHUFFLING
REDUCING
@manudellavalle

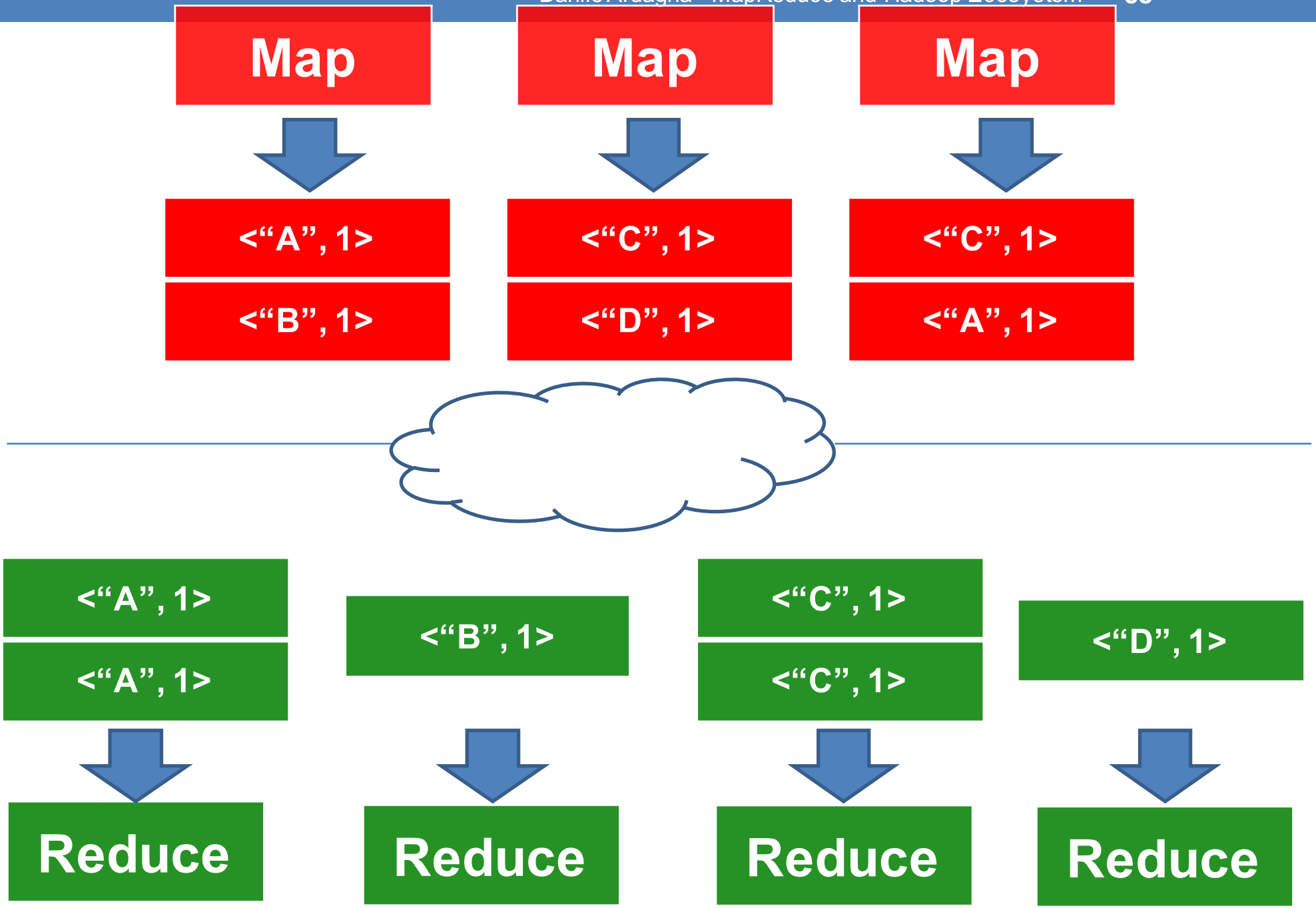
An example: Word Count

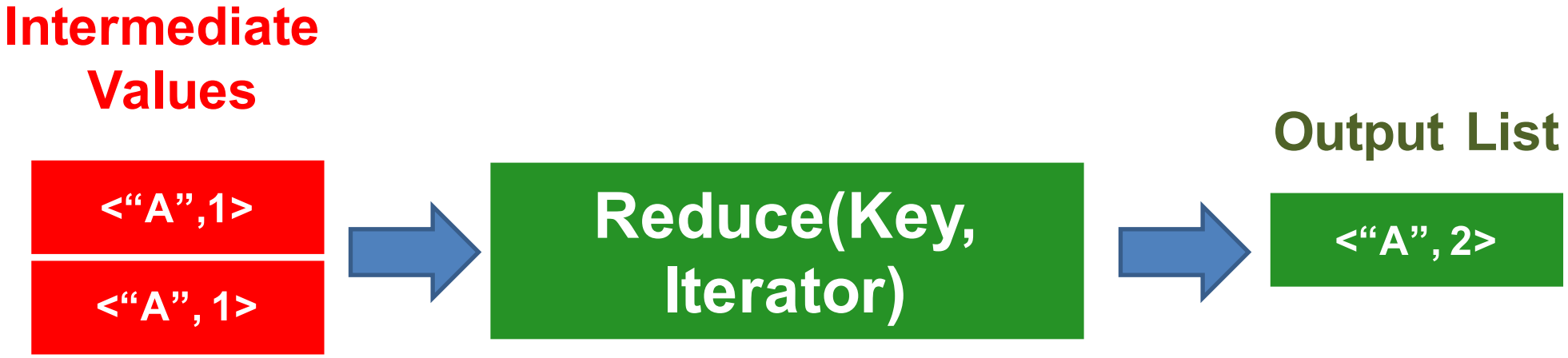


- We have many large files of words
- **Count** the number of times each **distinct word** appears in the files
- *Sample application*: analyze web server logs to find **popular URLs**




```
let map(String document_name, String document_content)=
foreach Word word in document_content :
emit(word, 1)
```


Map
Map
Map
<“A”, 1>
<“B”, 1>
<“C”, 1>
<“D”, 1>
<“C”, 1>
<“A”, 1>
<“A”, 1>
<“A”, 1>
<“B”, 1>
<“C”, 1>
<“C”, 1>
<“D”, 1>
Reduce
Reduce
Reduce
Reduce

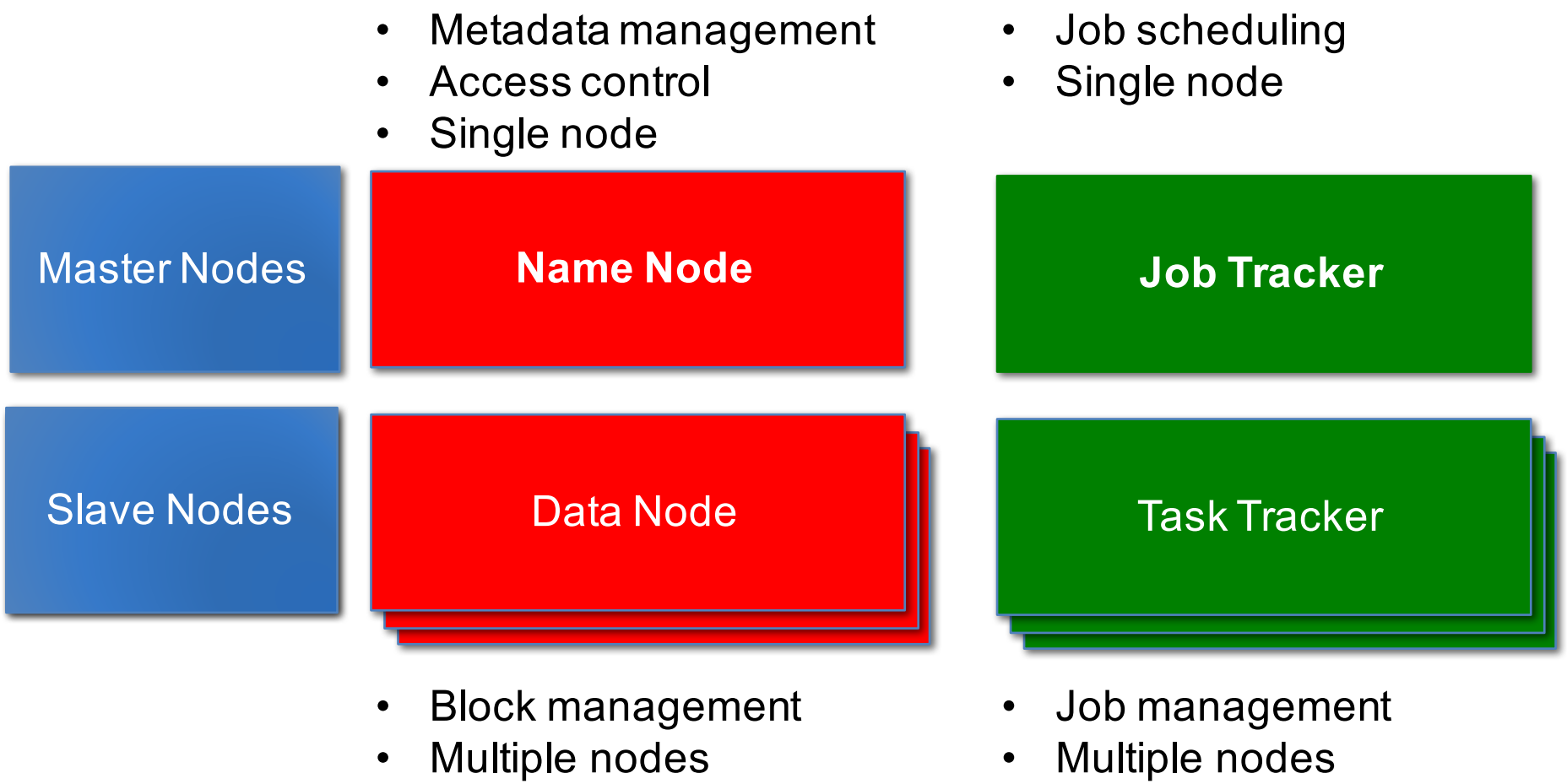



```
let reduce(Word word, Iterator<int> occurences) =
int total_occurences = 0;
foreach int o in occurences : total_occurences += o;
emit(word, total_occurences);
```

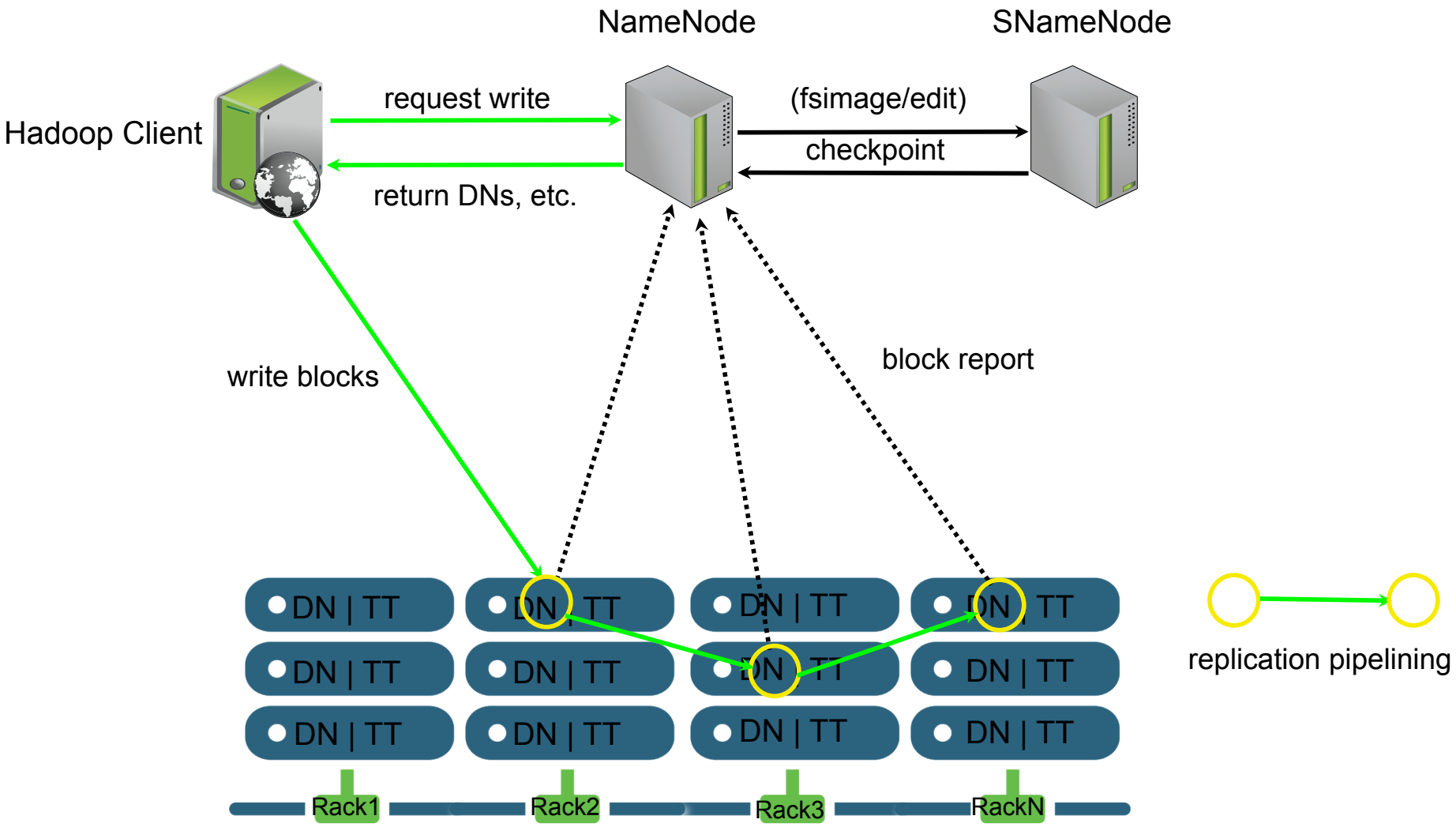
Motivations for MapReduce

- Large-Scale Data Processing:
  - Want to use 1000s of CPUs
  - But don't want hassle of ***managing*** things
- Map-reduce Architecture provides:
  - **Automatic** parallelization & distribution
  - **Fault tolerance**
  - I/O **scheduling**
  - **Monitoring** & status updates

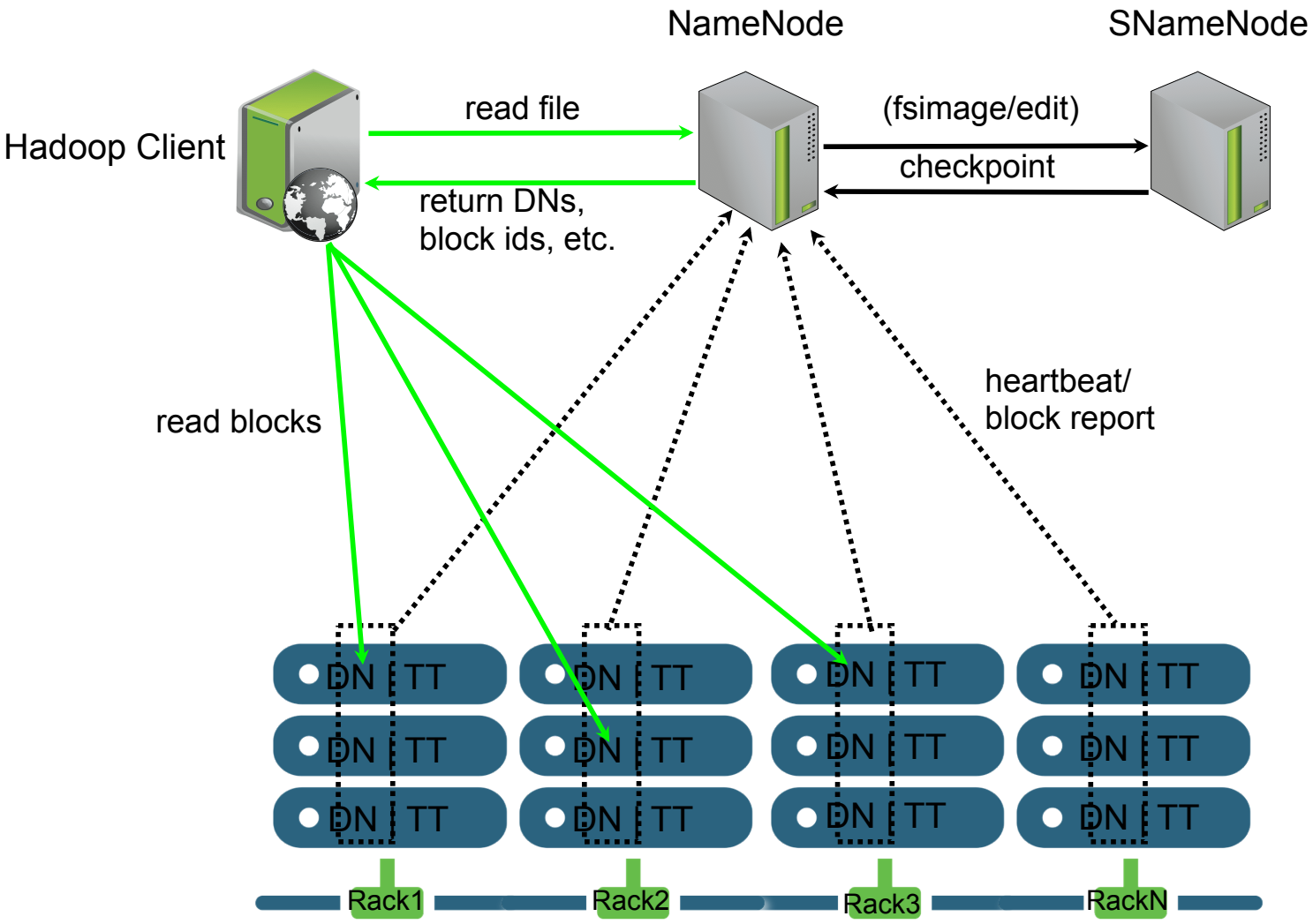
System view: Hadoop 1.0 implementation



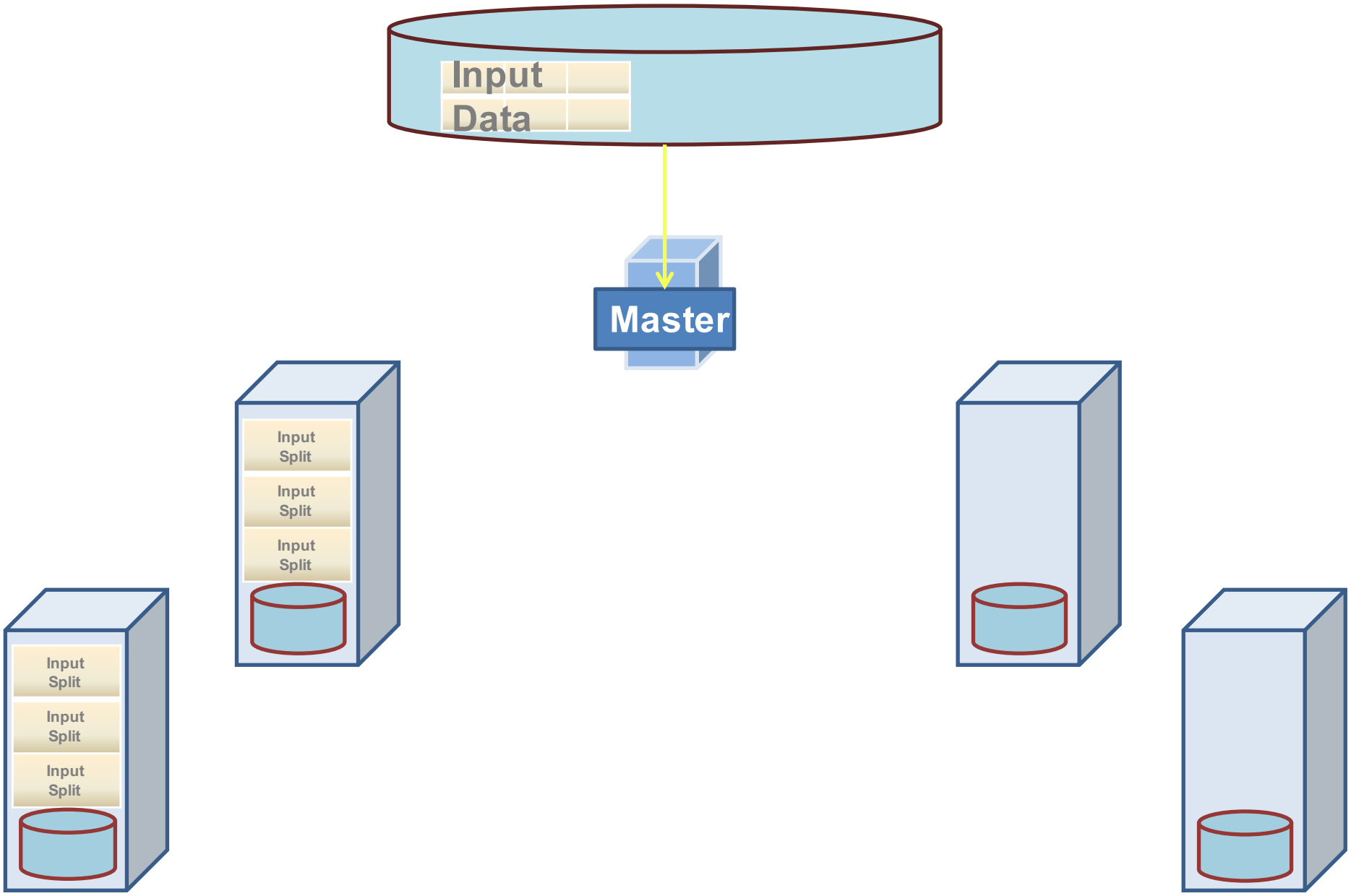
HDFS – Writing files



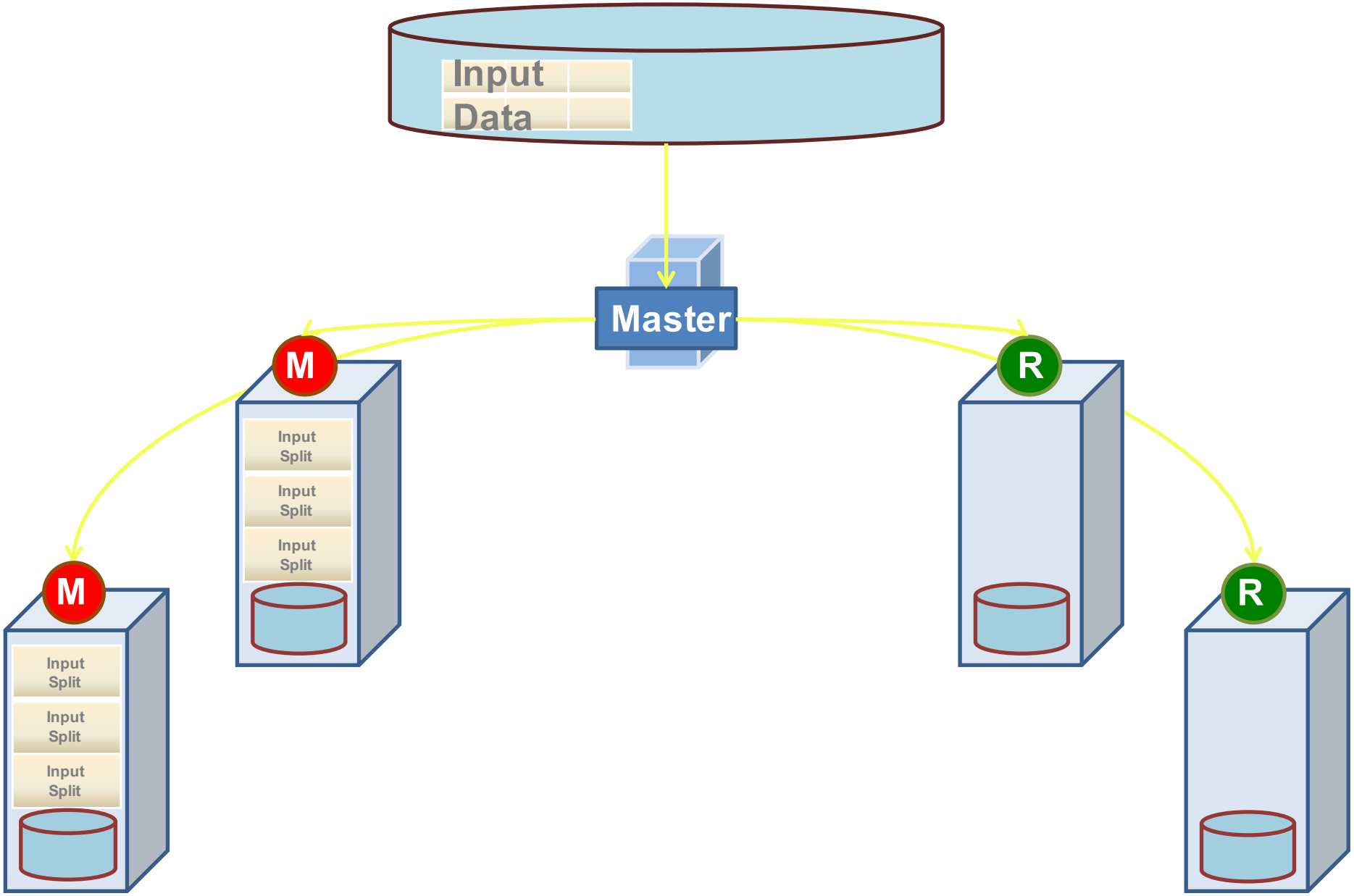
HDFS – Reading files



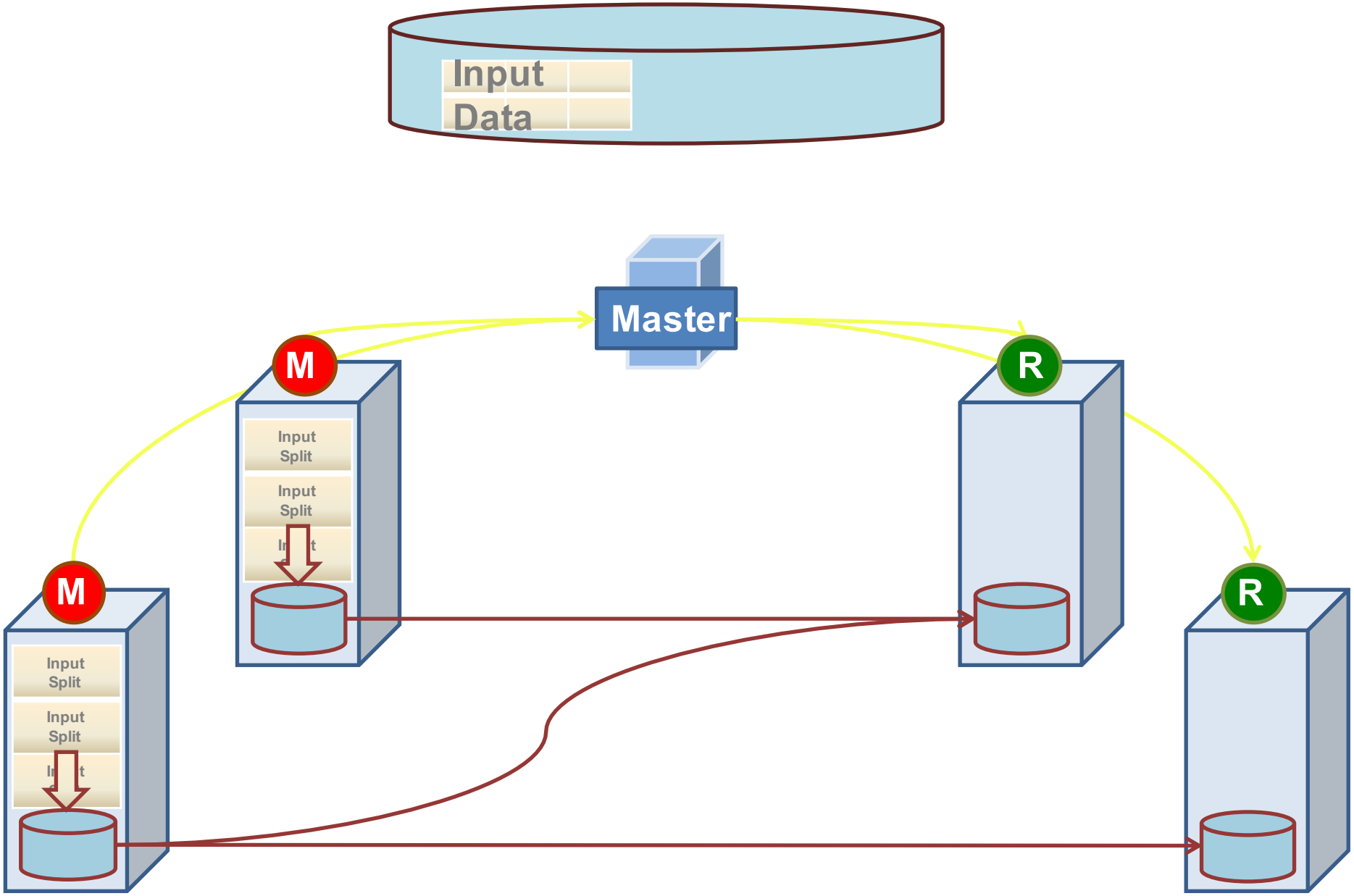
System view – Execution (Hadoop 1.0)



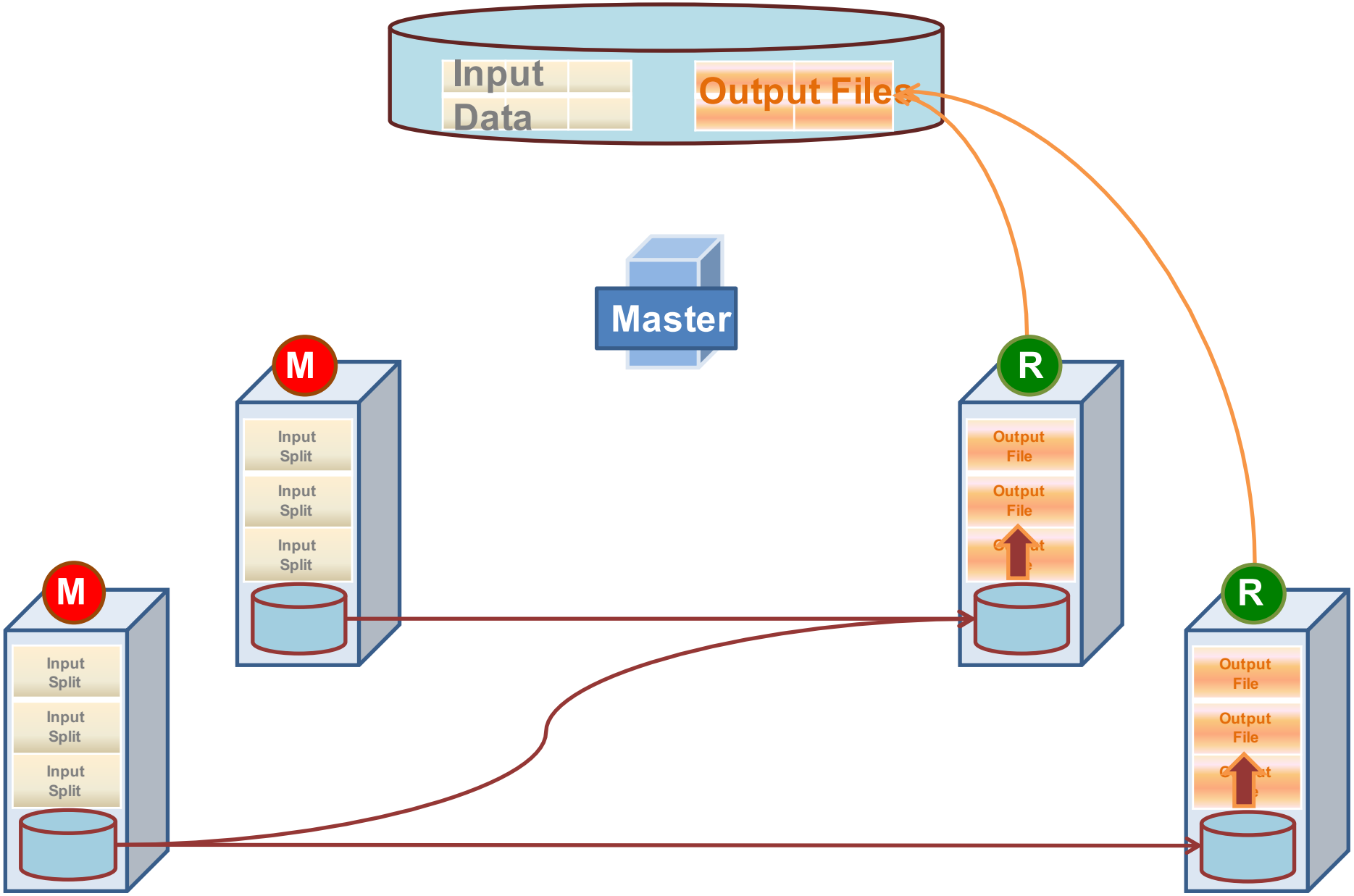
System view – Execution (Hadoop 1.0)



System view – Execution (Hadoop 1.0)



System view – Execution (Hadoop 1.0)



Coordination

- Master data structures
  - **Task status**: idle, in-progress, completed
  - **Idle tasks** get scheduled as workers become available
  - When a **map** task **completes**, it **sends** the **master** the **location** and sizes of its R **intermediate files**, one for each reducer
  - **Master pushes** this **info** to **reducers**
- Master **pings workers** periodically to detect failures

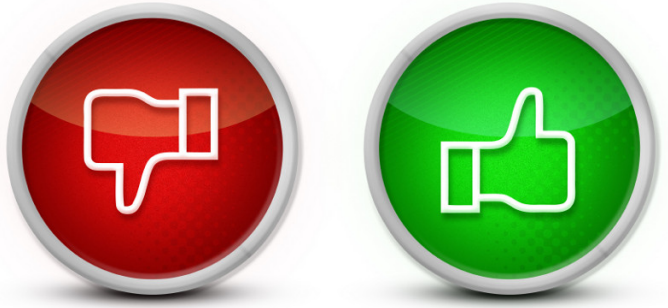
Failures

- Map worker failure
  - **Map** tasks **completed or in-progress** at worker are **reset** to idle
  - Reduce workers are notified when task is rescheduled on another worker
- Reduce worker failure
  - **Only in-progress tasks** are **reset** to idle
- Master failure
  - **MapReduce job** is **aborted** and client is notified

Content

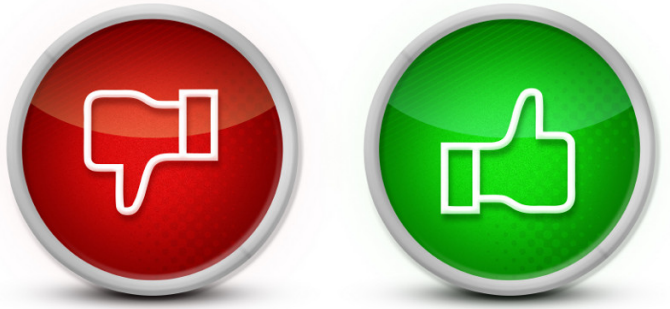
- Big Data Analytics and Data Science
- Big Data Analytics Applications
- Map-reduce and Hadoop fundamentals
- Map-reduce and Hadoop pros and cons
- Hadoop Success Stories
- Hadoop Evolution and Hadoop eco-system
- Pig and Hive
- Map Reduce Cloud based solutions
- Where is the world going? Spark Apache project

Hadoop 1.0 pros



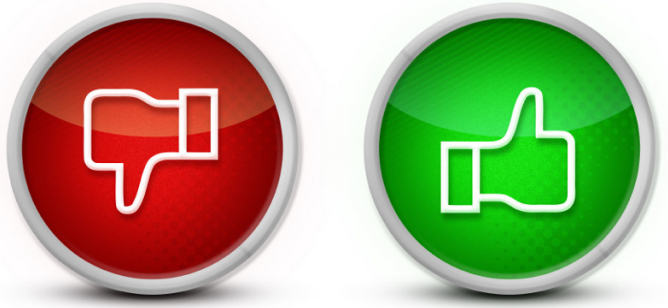
- "Simple" and easy to use
  - A programmer defines his job with only map and reduce functions, **without** having to **specify physical distribution** of his **job** across nodes
- Fault tolerance
  - MapReduce is **highly fault-tolerant** (continue to work in spite of an average of 1.2 failures per analysis job at Google)
- Flexible
  - **No dependency on data model** and schema (good for irregular or unstructured data)

Hadoop 1.0 cons



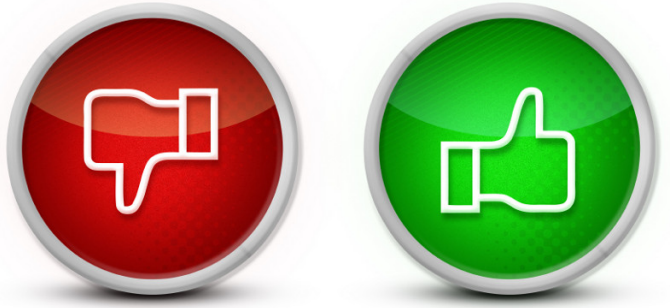
- A **single fixed dataflow**
  - The dataflow is fixed, many **complex algorithms** are **hard** to **implement** in a single job
  - Some algorithms that **require multiple inputs are not well supported** since the dataflow of MapReduce is originally designed to read a single input and generate a single output
- No high-level language
  - **No declarative language** like SQL in DBMS and any query optimization technique
- No schema and **no index**
  - Each item is parsed at reading input and transform it into data objects for data processing, causing performance degradation

Hadoop 1.0 cons



- Low efficiency
  - With fault-tolerance and scalability as its primary goals, MapReduce operations are **not** always **optimized for I/O efficiency**. In addition, Map and Reduce are **blocking** operations
  - **No** specific **execution** plans and **does not optimize** plans like DBMS does to minimize **data transfer** across nodes

Comparing Hadoop with commercial data warehouses



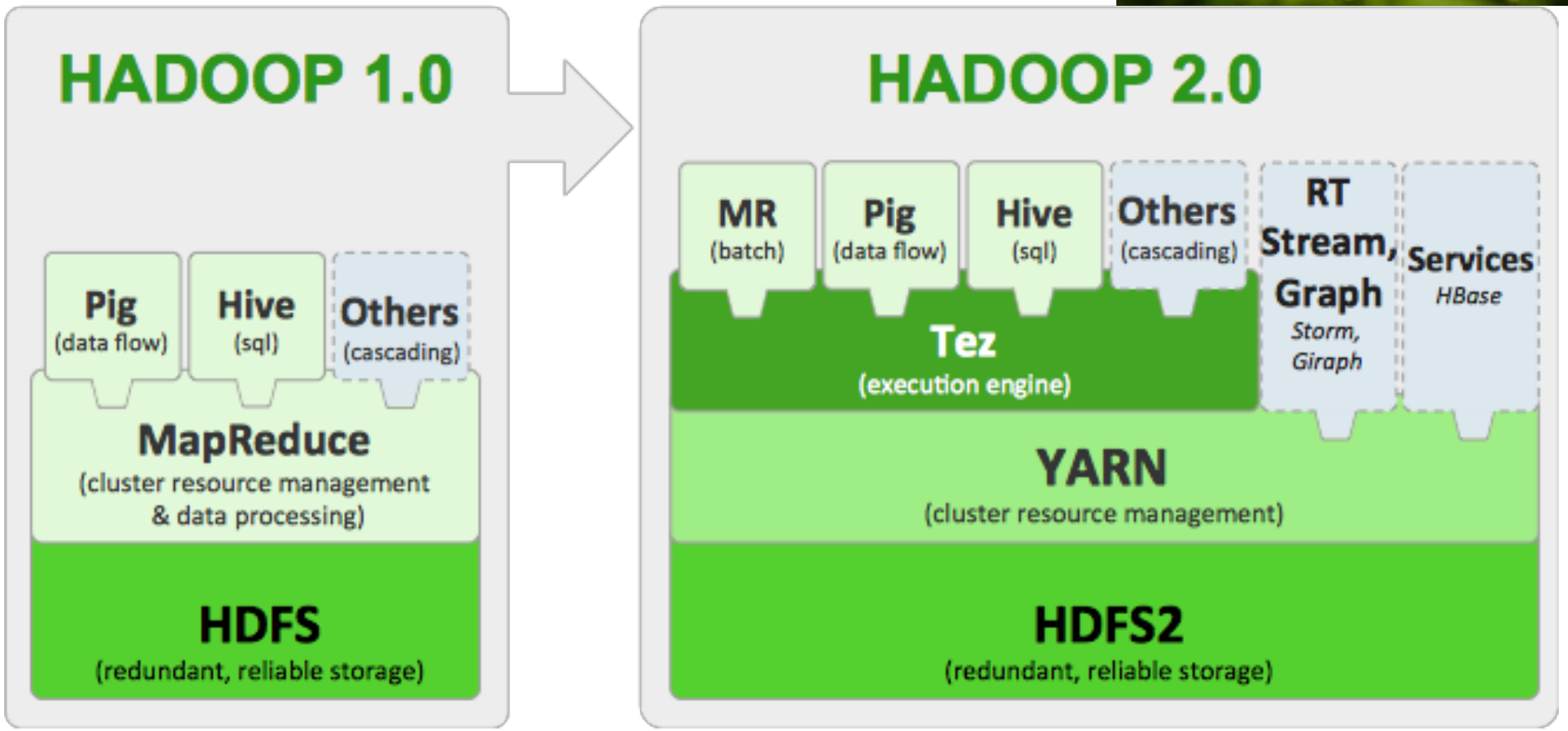
- Hadoop **2~50 times slower**, except in data loading

in 2011 was achieved

- ~~Current~~ Hadoop system ~~is~~ scalable but ~~achieves~~ **very low efficiency per node**, around 5MB/s processing rate, repeating a mistake that previous studies on HPC did *focusing on scalability but missing efficiency*

Source: K. H. Lee, Y. J. Lee, H. Choi, Y. D. Chung, and B. Moon. Parallel data processing with MapReduce: a survey. *SIGMOD Rec.* 40(4), 2011.

Hadoop Eco-system



https://www.youtube.com/watch?v=Z5kQR71yJpE

http://www.tableausoftware.com