MapReduce

Knowledge Objectives

1. Explain the main design assumptions behind MapReduce
2. Enumerate the main declarative languages proposed as alternatives to MapReduce
3. Identify when the combine function is really useful

Understanding Objectives

1. Simulate the internal MapReduce algorithm
2. Elaborate on 6 improvements for MapReduce
3. Explain the 4 main drawbacks of MapReduce

Application Objectives

1. Write a simple program (less than a hundred lines) benefitting from MapReduce and HBase libraries

MAPREDUCE

USAGES

Typical Uses

- Find which source pages link to a target page
- Count the number of accesses to each Web page
- Count the number of accesses to each domain
- Create an index structure that maps search terms to document IDs
- Retrieve introductory paragraph of all Web pages so that “x”
- Find all pairs of users accessing the same URL
- Find the average age of users accessing a given URL
- Find all friends of a given user
- Find all friends of friends of a given user
- Find all women friends of men friends of a given user
- Grouping different manifestations of the same real world object

Not So Typical Uses

- Mobile Commerce
- Electricity
- Agricultural Planning
- Fuel Conservation
- National Intelligence
- Drug Development and Personalization
- Financial Service Security

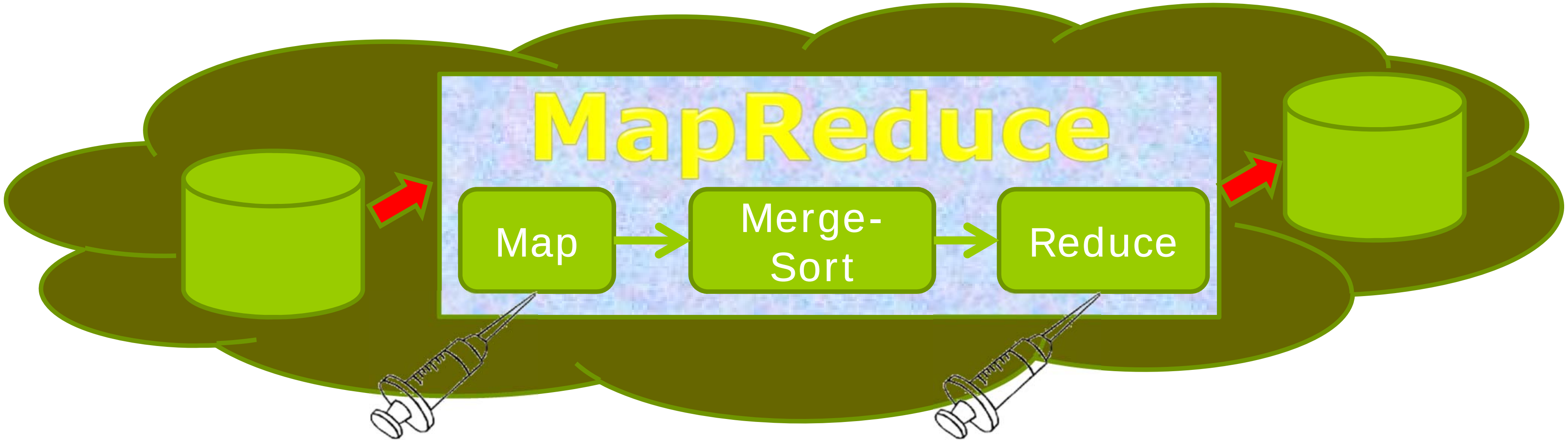
MAPREDUCE

THE ALGORITHM UNDERNEATH

MapReduce

- Apache MapReduce
  - Based on Google File System (GFS)
  - http://hadoop.apache.org/docs/r1.0.4/mapred_tutorial.html
- Designed to meet the following requirements
  - Exploit distributed systems and provide **full distributed-transparency** for the end-user
    - Send the queries to data (i.e., query-shipping instead of data-shipping for exploiting the data locality principle)
    - Support parallelism and hide its complexity
      - Independent data (typically collected from the web)
        - Without references to other pieces of data
        - No joins
      - Exploit petabytes of data in batch mode
        - No transactions
    - Failure resilience
      - Cope with failures without aborting
- Inspired in functional programming
  - On top of Hadoop

MapReduce Basics



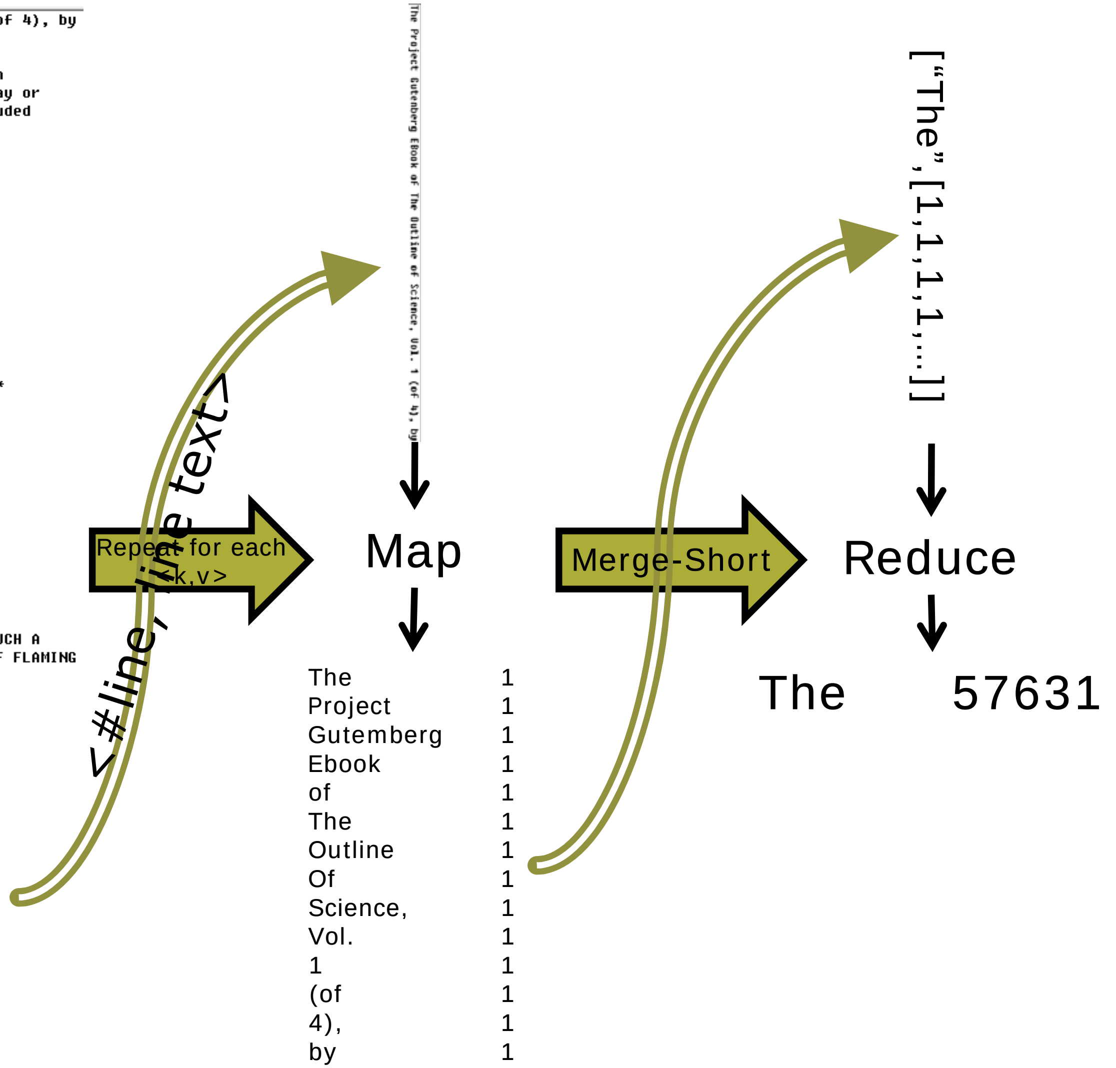
- Simple model to express relatively sophisticated distributed programs
  - Processes pairs [key, value]
  - Signature:

$$\mathtt{map}(\text{key } k, \text{value } v) \mapsto [(ik_1, iv_1), \ldots, (ik_{m(k,v)}, iv_{m(k,v)})]$$

$$\mathtt{reduce}(\text{key } ik, \text{vset } ivs) \mapsto [ov_1, .., ov_{r(ik,ivs)}]$$

WordCount Execution Example

```
The Project Gutenberg EBook of The Outline of Science, Vol. 1 (of 4), by
J. Arthur Thomson

This eBook is for the use of anyone anywhere at no cost and with
almost no restrictions whatsoever.  You may copy it, give it away or
re-use it under the terms of the Project Gutenberg License included
with this eBook or online at www.gutenberg.org


Title: The Outline of Science, Vol. 1 (of 4)
       A Plain Story Simply Told

Author: J. Arthur Thomson

Release Date: January 22, 2007 [EBook #20417]

Language: English

Character set encoding: ASCII

*** START OF THIS PROJECT GUTENBERG EBOOK OUTLINE OF SCIENCE ***




Produced by Brian Janes, Leonard Johnson and the Online
Distributed Proofreading Team at http://www.pgdp.net






[Illustration: THE GREAT SCARLET SOLAR PROMINENCES, WHICH ARE SUCH A
NOTABLE FEATURE OF THE SOLAR PHENOMENA, ARE IMMENSE OUTBURSTS OF FLAMING
HYDROGEN RISING SOMETIMES TO A HEIGHT OF 500,000 MILES]




THE
OUTLINE OF SCIENCE

A PLAIN STORY SIMPLY TOLD



EDITED BY
J. ARTHUR THOMSON
REGIUS PROFESSOR OF NATURAL HISTORY IN THE
UNIVERSITY OF ABERDEEN




WITH OVER 800 ILLUSTRATIONS
OF WHICH ABOUT 40 ARE IN COLOUR
```



WordCount Code Example

public void map(**Key** , **Value**) {

Blackbox

Key , **Value**

}

}

public void reduce(**Key** , **Value<u>s</u>**) {

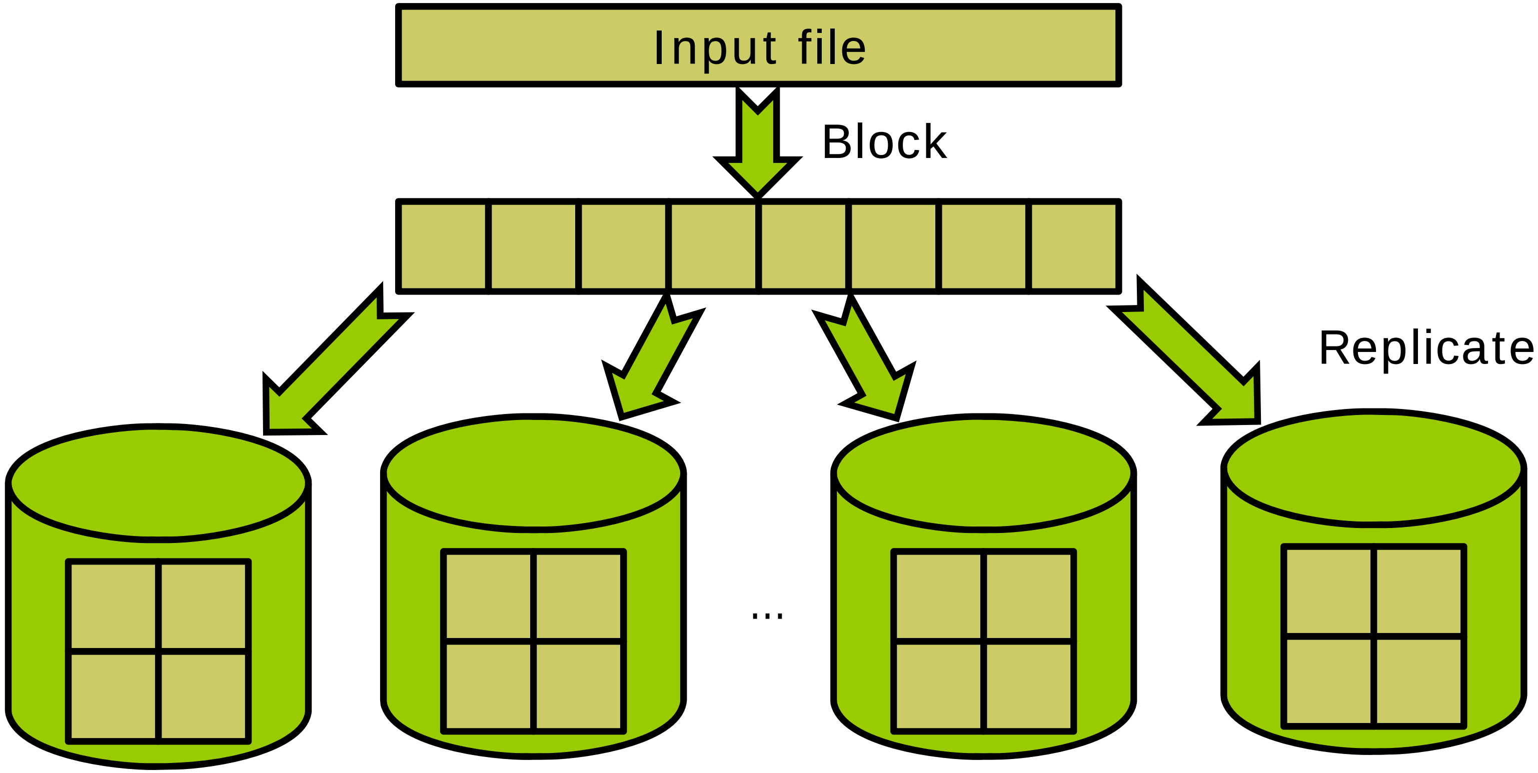
Blackbox

Key , **Value**

}

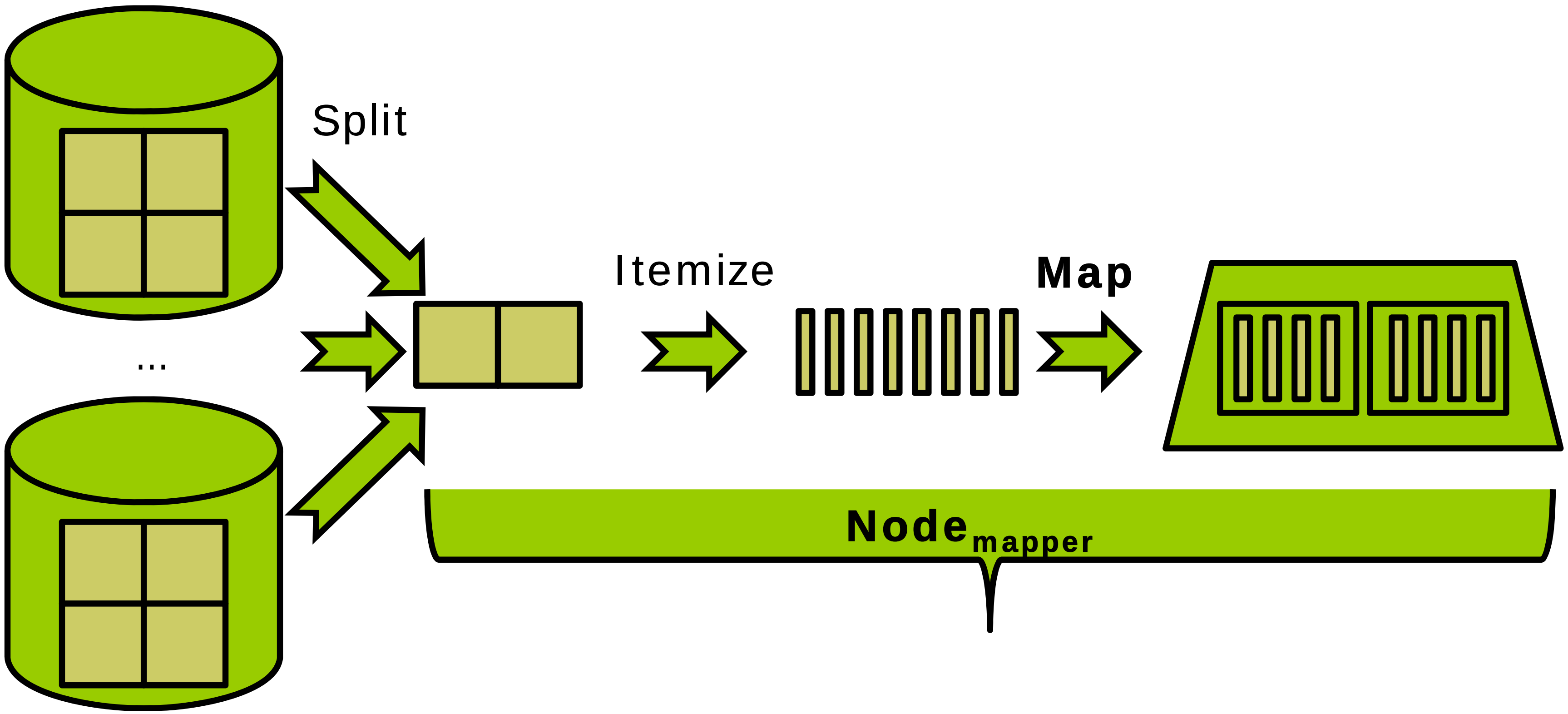
MapReduce Algorithm: Data Load

1. The input data is partitioned into blocks
   - It can be done by using HDFS or any other storage (e.g., HadoopDB, MongoDB, Cassandra, CouchDB, etc.)
2. Replicate them in different nodes



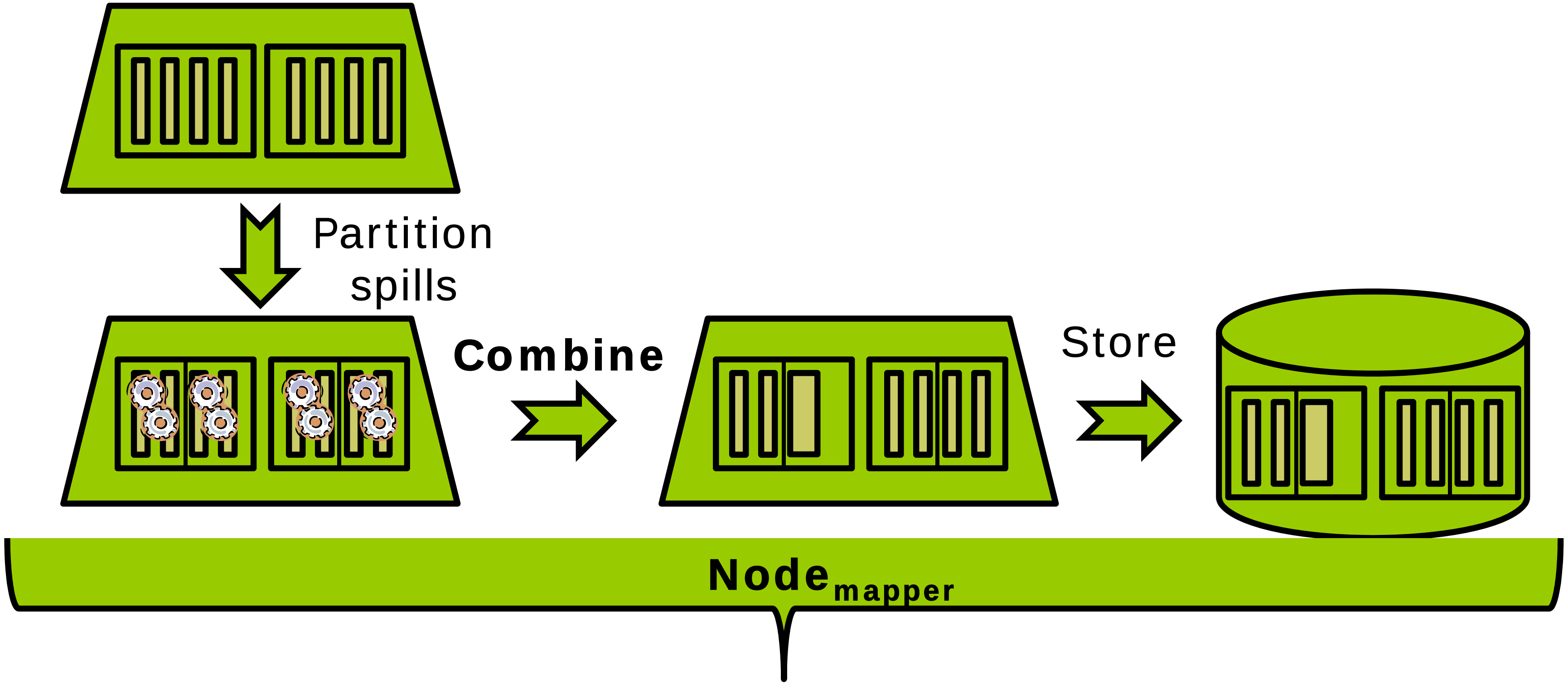
MapReduce Algorithm: Map Phase (I)

3. Each map subplan reads a subset of blocks (i.e., split)
   - Ideally, to exploit data locality, 10 to 100 mappers per node
4. Divides it into records
5. Executes the map for each record and leaves them in memory divided into spills



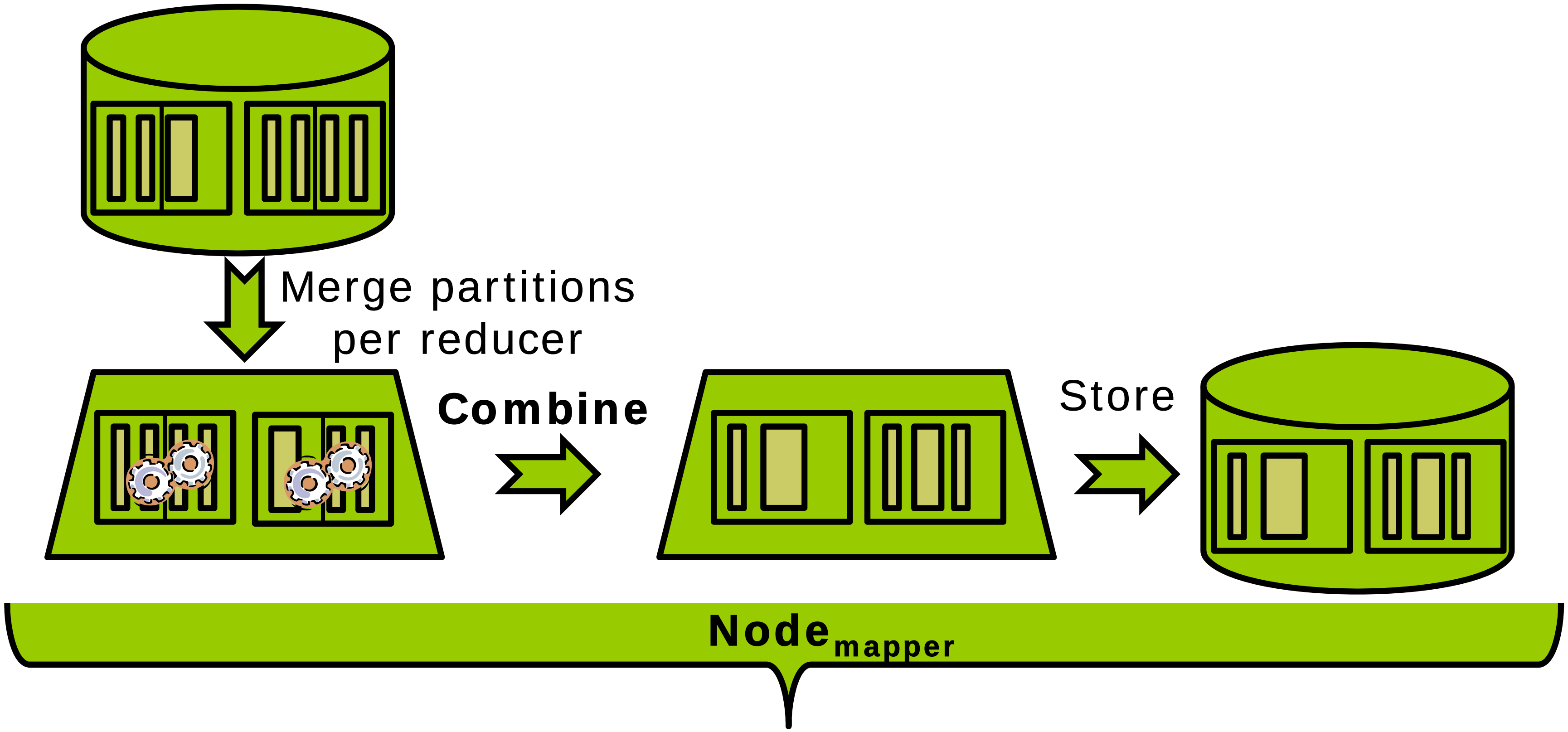
MapReduce Algorithm: Map Phase (II)

6. Each spill is then partitioned per reducers
   - Using a hash function f over the key, according to the number of reducers R
     - Both can be parametrized
7. Each spill partition is sorted independently
   - If a combine is defined, it is executed locally after sorting
8. Store the spill partitions into disk (massive writing)



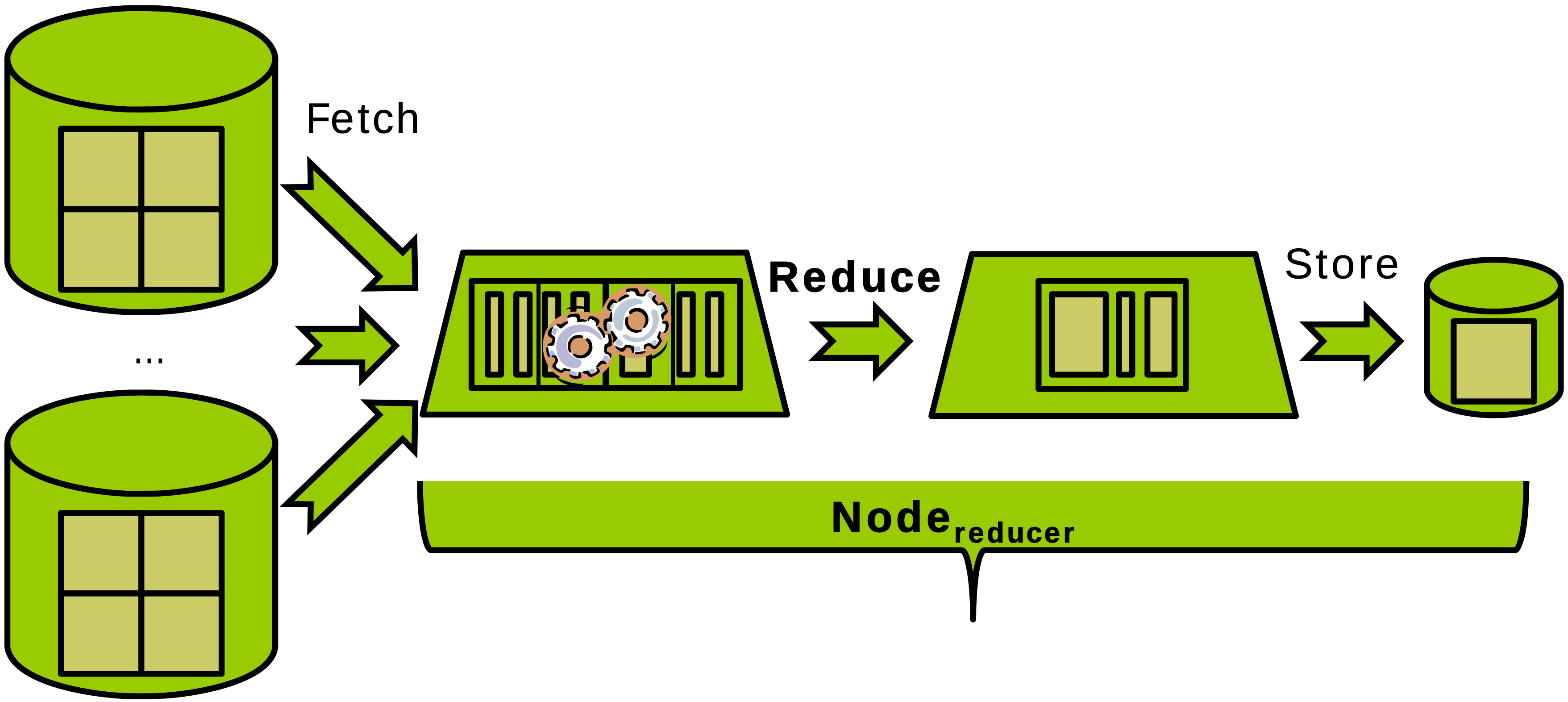
Algorithm: Map phase (III)

9. Spill partitions are merged and sorted independently
   - Combine function is applied to the merges
10. Store the result into disk

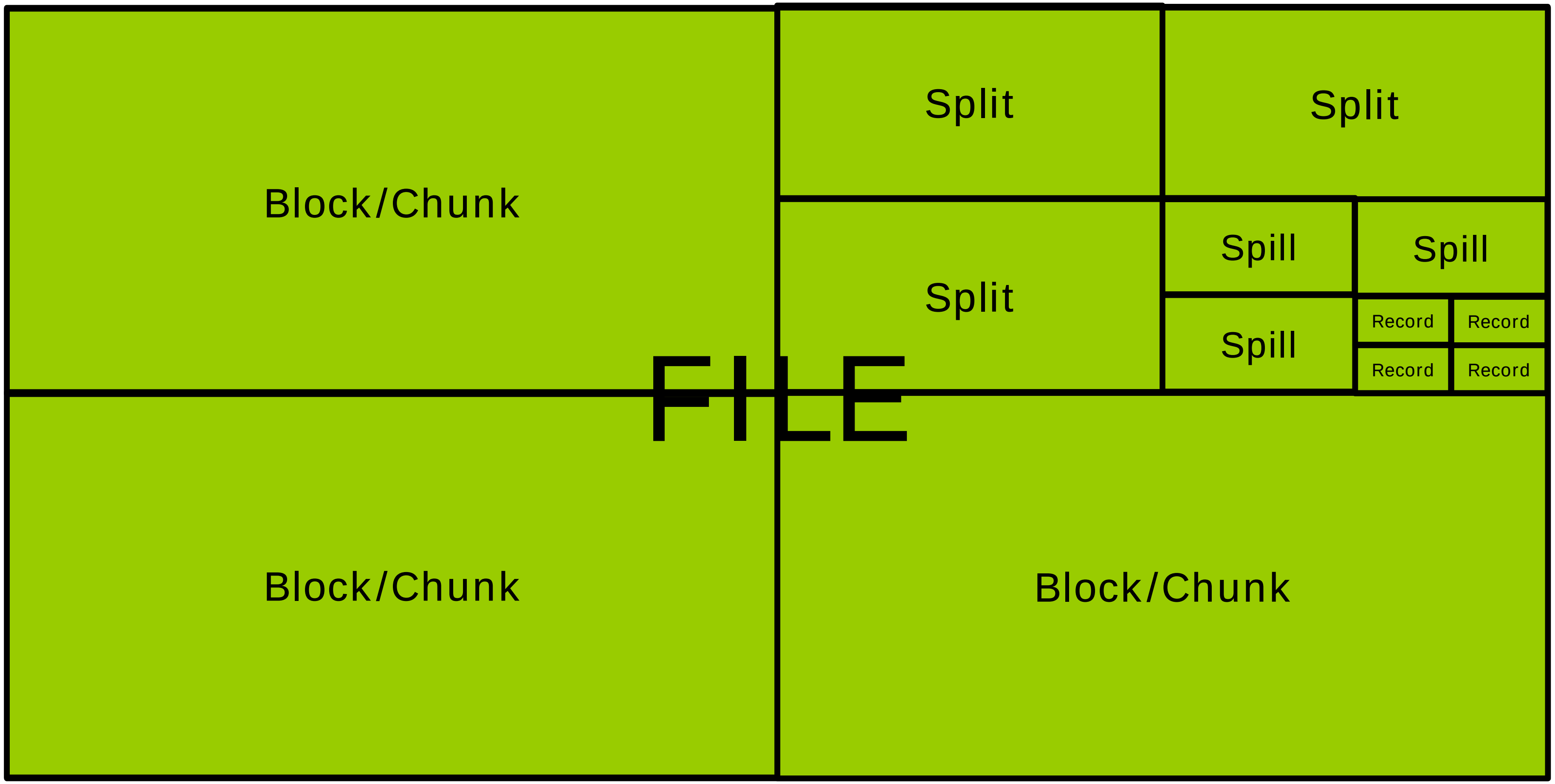


MapReduce algorithm: Shuffle and Reduce

11. Reducers fetch data from mappers (<u>massive data transfer!</u>)
12. Mappers output is merged and sorted
13. Reduce function is executed per key
14. Store the result into disk



MapReduce objects



Record=Key-Value pair

Local-Global Aggregation: Combine

- Combine is executed locally
  - Assumes uniform random distribution of input
  - Reduces the number of tuples sent to reducers
- Only possible when the reducer function is:
  - Commutative
  - Associative
- Only makes sense if $|I|/|O| >> \#CPU$

Activity: MapReduce

- *Objective: Understand the algorithm underneath MapReduce*
- *Tasks:*
  1. *(40') Reproduce step by step the MapReduce execution*
     - Consider the following data set:
       - Block0: "a b b a c | c d c a e"
       - Block1: "a b d d a | b b c c f"
     - Simulate the execution of the MapReduce code given the following configuration:
       - The map and reduce functions are those of the wordcount
         - The combine function shares the implementation of the reduce
       - There is one block per split
       - The "|" divides the records inside each block
         - We have two records per block
       - We can keep four pairs [key,value] per spill
       - We have two mappers and two reducers
         - Machine0, contains block0, runs mapper0 and reducer0
         - Machine1, contains block1, runs mapper1 and reducer1
       - The hash function used to shuffle data to the reducers uses the correspondence:
         - {b,d,f}->0
         - {a,c,e}->1

MAPREDUCE

DRAWBACKS

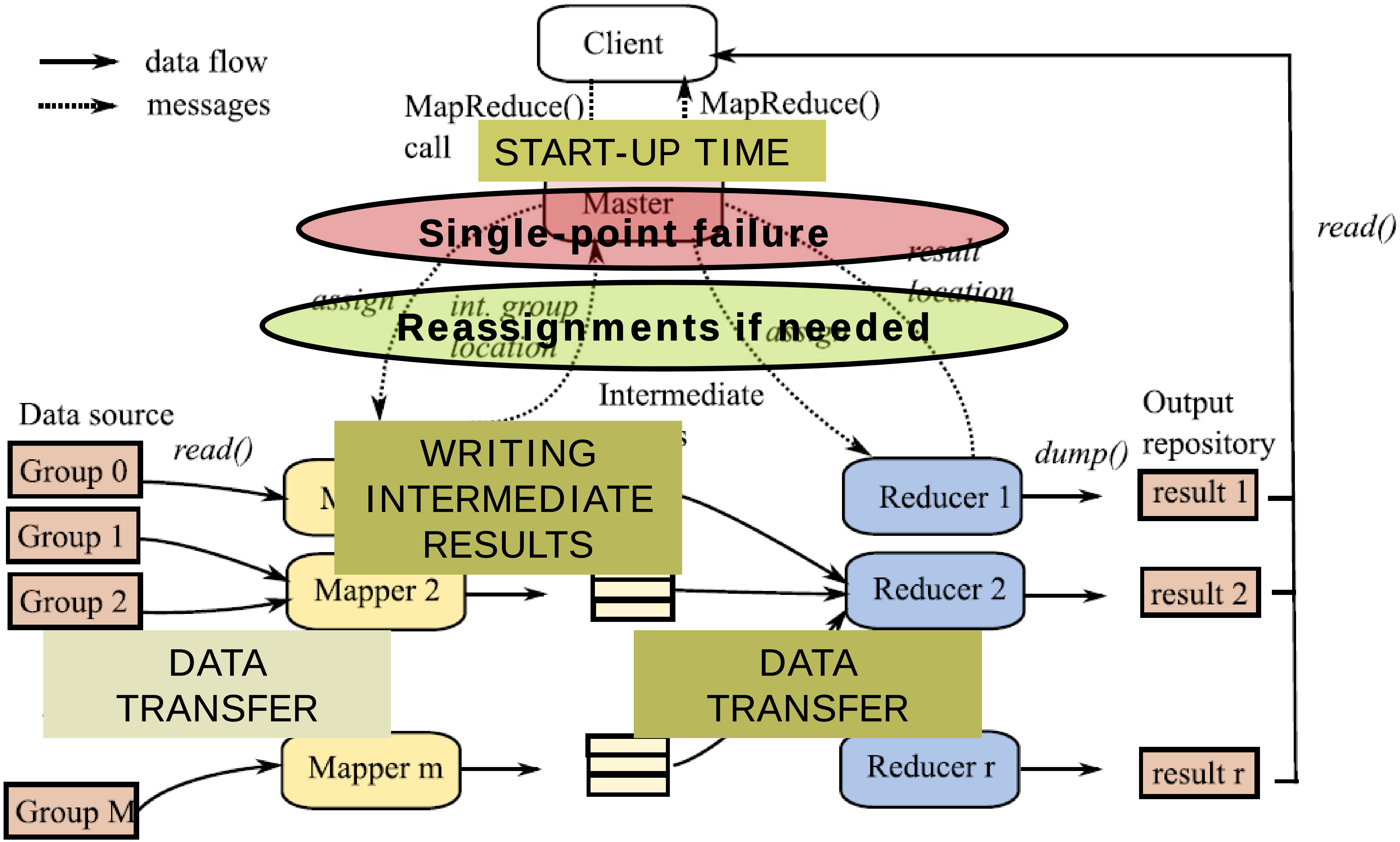
High Level Languages

- MapReduce is the de facto standard for robust execution of large data-oriented tasks
  - Support in HBase, MongoDB, CouchDB, etc.
- MapReduce has been massively criticized for being too low-level
- However, other NOSQL databases do not provide better solutions
  - APIs for Ruby, Python, Java, C++, etc.
- But something is changing...
  - Attempts to build declarative languages on top of MapReduce
    - Hive
    - Pig Latin
  - Cassandra Query Language (CQL)
    - Resembles SQL

MapReduce at First Sight

- The MapReduce paradigm program is computationally **complete** and **<u>ANY</u>** program can be adapted to it
- Furthermore, MapReduce's signature **<u>is closed</u>**
  - For example, map-reduce iterations can be nested
- However, some tasks better adapt to it than others
  - Easily adaptable:
    - Aggregations
    - Selections
    - Projections
    - Set operators
    - Sorting
  - Difficult for:
    - Joins
    - Any other operation referring to other data

MapReduce: Tasks and Data Flows



Contributions of Spark

- Immutable storage of arbitrary records across a cluster
- Low latency
  - Prioritizes in-memory processing
- Palette of coarse-grained operators
  - … beyond map and reduce
- Control over data partitioning

Summary

- MapReduce phases
  - Combine function
- MapReduce drawbacks

Bibliography

- S. Abiteboul et al. Web Data Management, 2012
- J. Dittrich et al. Hadoop++: Making a yellow elefant run like a cheetah (without it even noticing)
- D. Jiang et al. *The performance of MapReduce: An In-depth Study*. VLDB'10
- E. Brewer, "Towards Robust Distributed Systems," *Proc. 19th Ann. ACM Symp.Principles of Distributed Computing* (PODC 00), ACM, 2000, pp. 7-10.
- F. Chang et all. *Bigtable: A Distributed Storage System for Structured Data*. OSDI'06
- Sanjay Ghemawat et al. *The Google File System*. OSDI'03
- Jeffrey Dean et al. *MapReduce: Simplified Data Processing on Large Clusters*. OSDI'04
- D. Battre et al. Nephele/PACTs: A Programming Model andExecution Framework forWebScale Analytical Processing. SoCC'10
- L. Liu and M.T. Özsu (Eds.). Encyclopedia of Database Systems. Springer, 2009