Concepts and Models of Parallel and Data-centric Programming

MapReduce – Parallelizing MapReduce

Lecture, Summer 2020

Simon Schwitanski
Dr. Christian Terboven

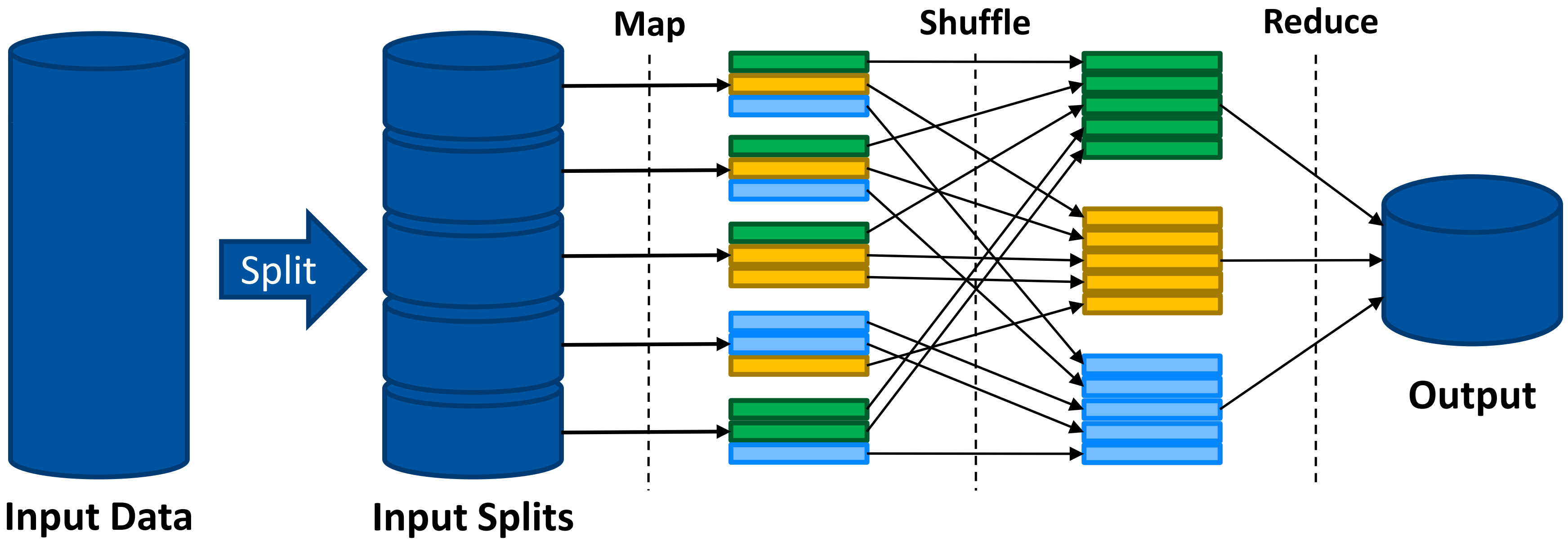
High Performance Computing

RWTH AACHEN UNIVERSITY

Outline

Recap: MapReduce in a Nutshell

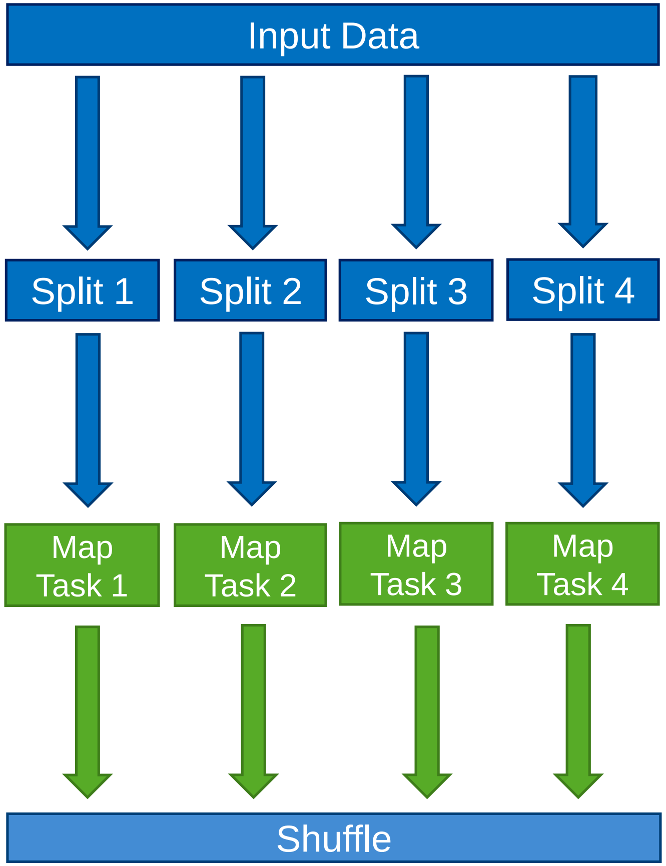
- Two essential functions *Map* and *Reduce* are defined by developer
- Work on data as key-value (KV) pairs, types chosen by developer
- Rest implicitly provided by framework
- Three execution steps: Map, Shuffle, Reduce



Opportunities for Parallelism (1)

Parallel map over input

- Map function can be performed independently on each element
  - Total data parallelism
  - “Embarrassingly parallel”
- Divide input into equally sized “input splits”
- Assign each input split to a *Map task*
  - Performs Map function(s) on input split
- Data access, splitting and distribution handled by distributed file system



Opportunities for Parallelism (2)

Parallel reduction over grouped keys

- *Reduce* function can be performed independently on each group
  - Again: Data parallelism
- Assign groups to *partitions*
- Each *Reduce task* processes typically one partition (consisting of several groups)
- Important: Reduce tasks can only start after all Map tasks are finished
  - Implicit barrier
  - Reason: Each Map task can potentially produce output for any arbitrary Reduce task.

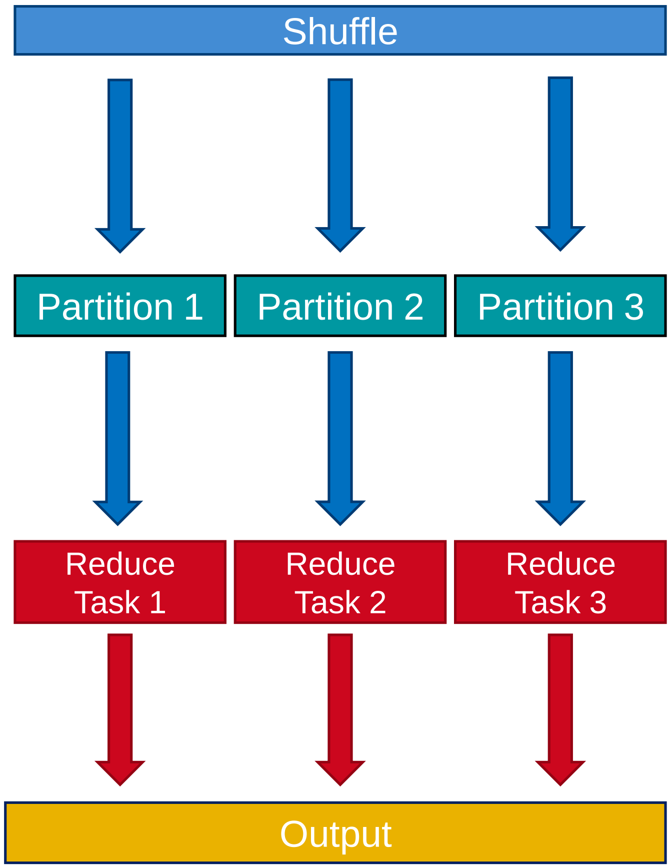
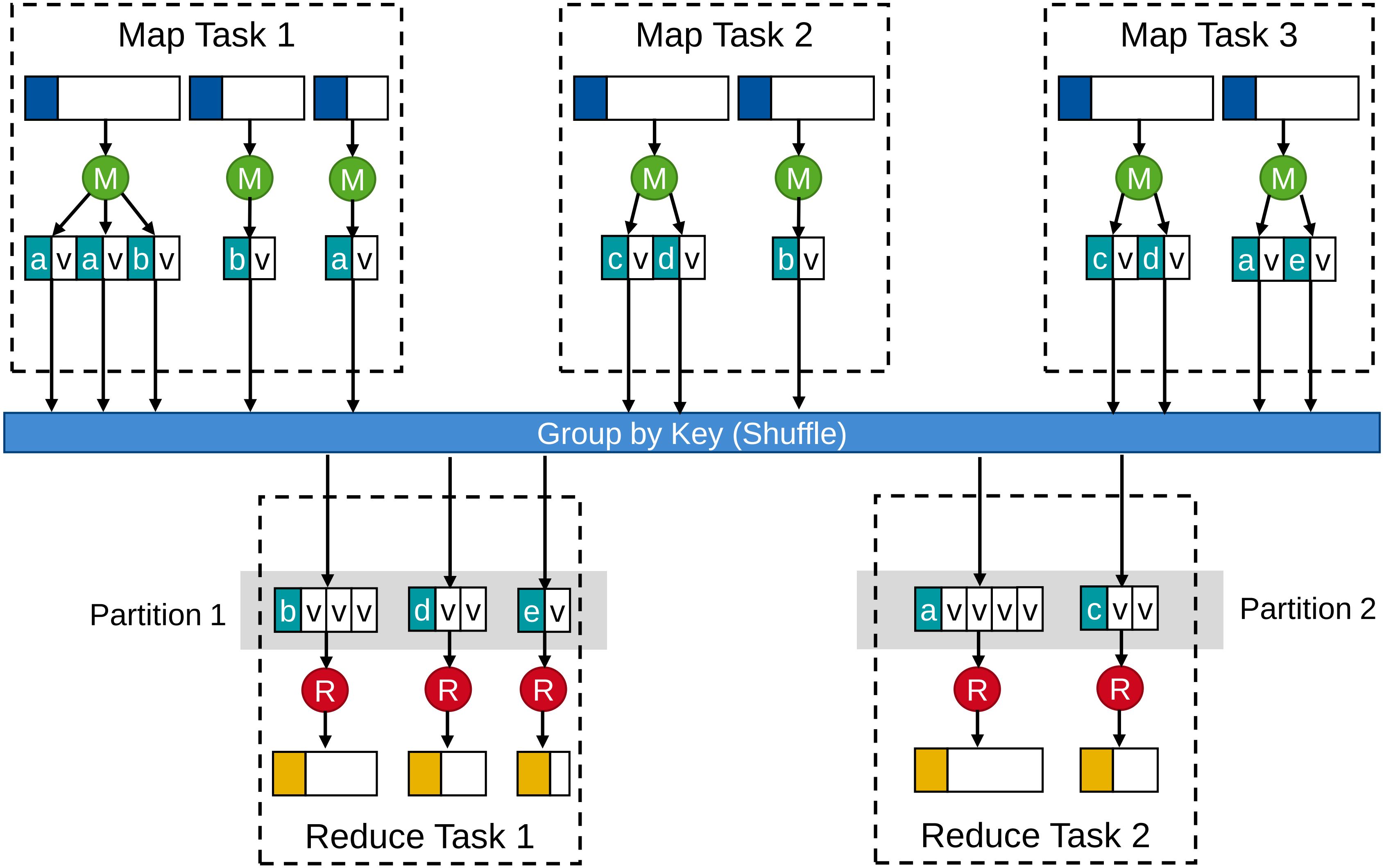


Illustration adapted from Dean, Jeffrey, and Sanjay Ghemawat. “MapReduce: simplified data processing on large clusters.” *Communications of the ACM* 51.1 (2008): 107-113.

MapReduce in Parallel (1)

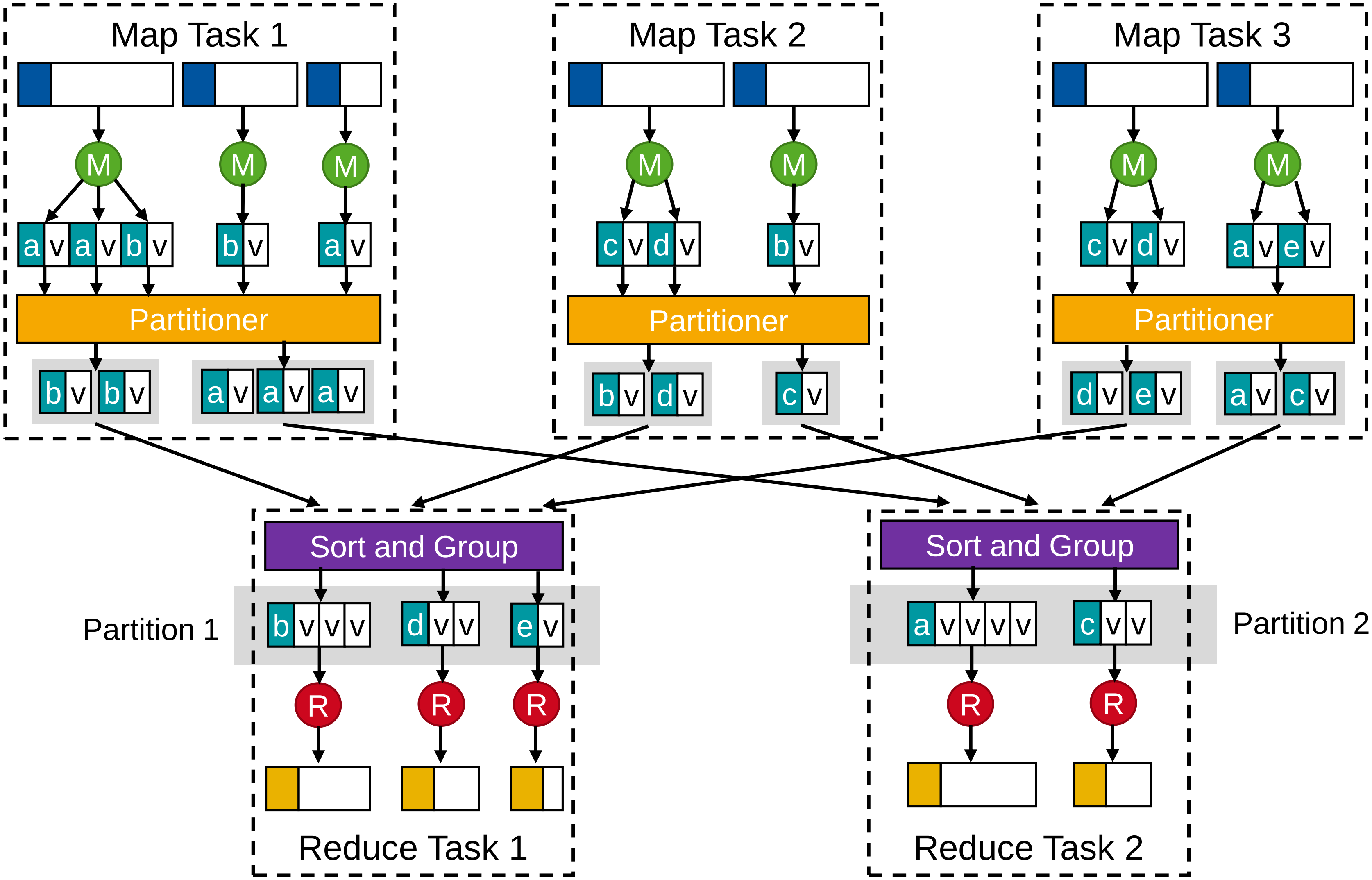


What about the Shuffle?

- So far: Considered Shuffle step as black box
- Shuffle is critical point for MapReduce computation
  - Transfer results of Map tasks to Reduce tasks
  - Typical: High network I/O during shuffle step
- *Partitioner* runs on each Map task after performing Map function
  - **Partitions** intermediate KV pairs
  - Partitioning provided by a hash function (can be user-defined)
  - Determines which group goes to which Reduce task
  - Typically one partition per Reduce task
  - Keys are **sorted per partition**
- Reduce task retrieves, **merge-sorts** and groups partitions from different Map tasks in single partition

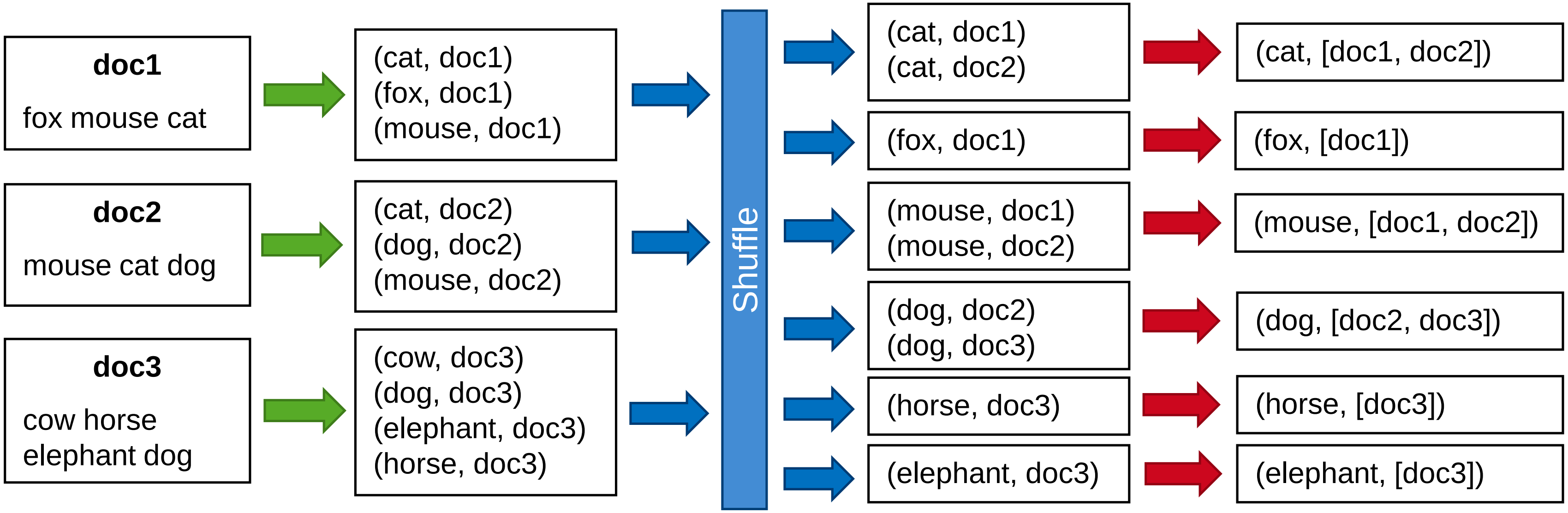
Illustration adapted from Dean, Jeffrey, and Sanjay Ghemawat. “MapReduce: simplified data processing on large clusters.” *Communications of the ACM* 51.1 (2008): 107-113.

MapReduce in Parallel (2)



Example Applications – Inverted Index

- Inverted index: Lookup all matching documents for a given search word
  - *Map*: Parse each document, emit sequence of $(word, document\ ID)$
  - *Reduce*: For all pairs for a given word emit $(word, list(document\ ID))$
  - Result: Given some search word, all matching documents can be returned



Example Applications – Reverse Web-Link Graph

- Web-link graph
  - Nodes: URLs of web pages
  - Edge from node u to node v if web page of u points to web page of v
- Reverse web-link graph: Edges reversed
- Computation with MapReduce framework
  - *Map*: Output $(target, source)$ for each link to target URL found on source
  - *Reduce*: Concatenate all source URLs associated with a target URL, emit $(target, list(sources))$