Presenter: Nian Ke
David R . Cheriton School of Computer Science
University of Waterloo

MapReduce: Simplified Data Processing on Large Clusters

Outline

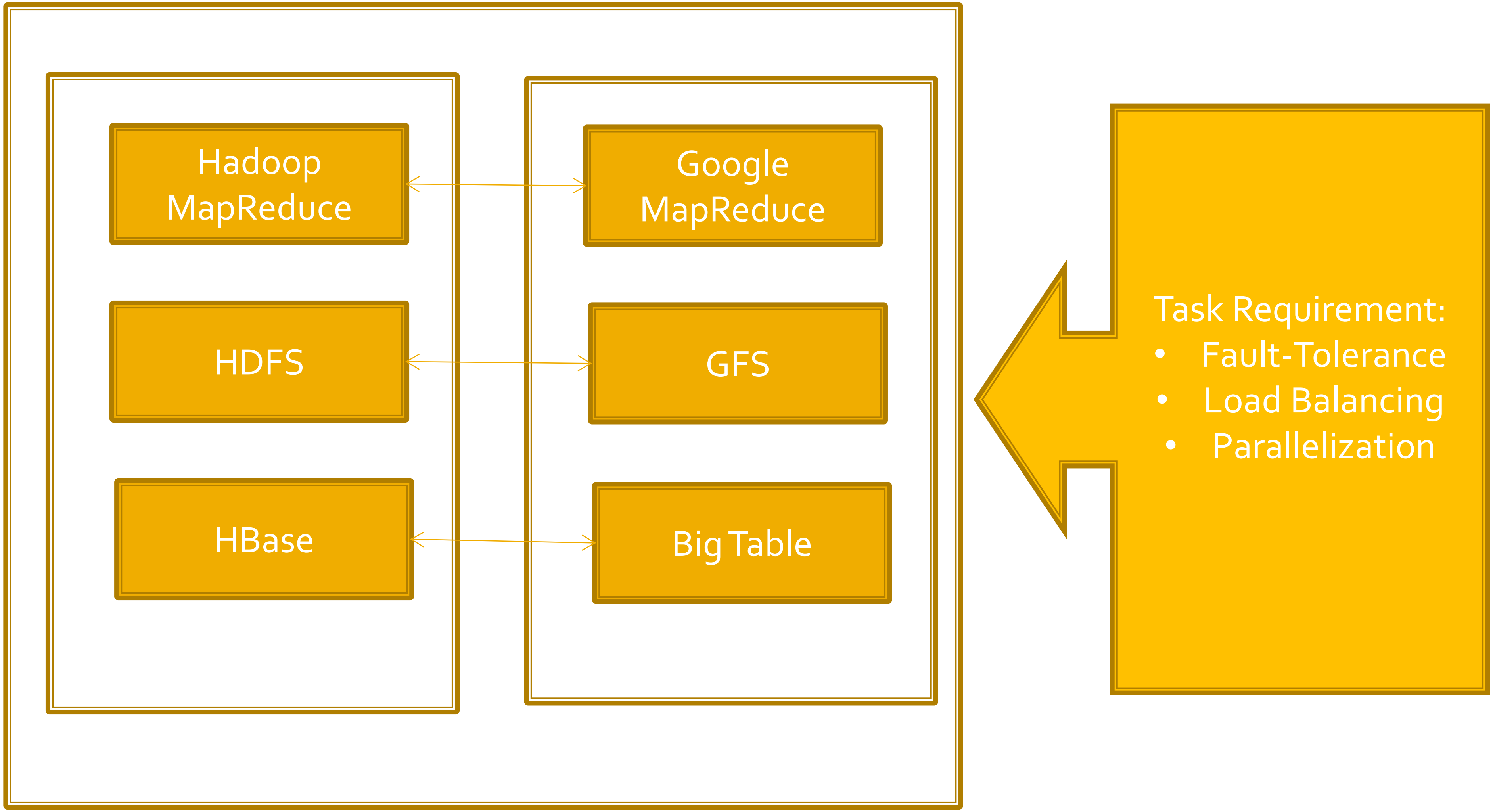
- Background
- Overview of MapReduce Framework
- Implementation Details
- Extending MapReduce: Spark Project

Background

How everything starts?

- Problems for Google:
  - How to store the big data set over commodity computers
    - Google File System(GFS)
    - BigTable
    - Megastore
  - How to efficiently compute the results with distributed file systems
    - MapReduce Framework

Computational Framework



Hadoop MapReduce

HDFS

HBase

Google MapReduce

GFS

Big Table

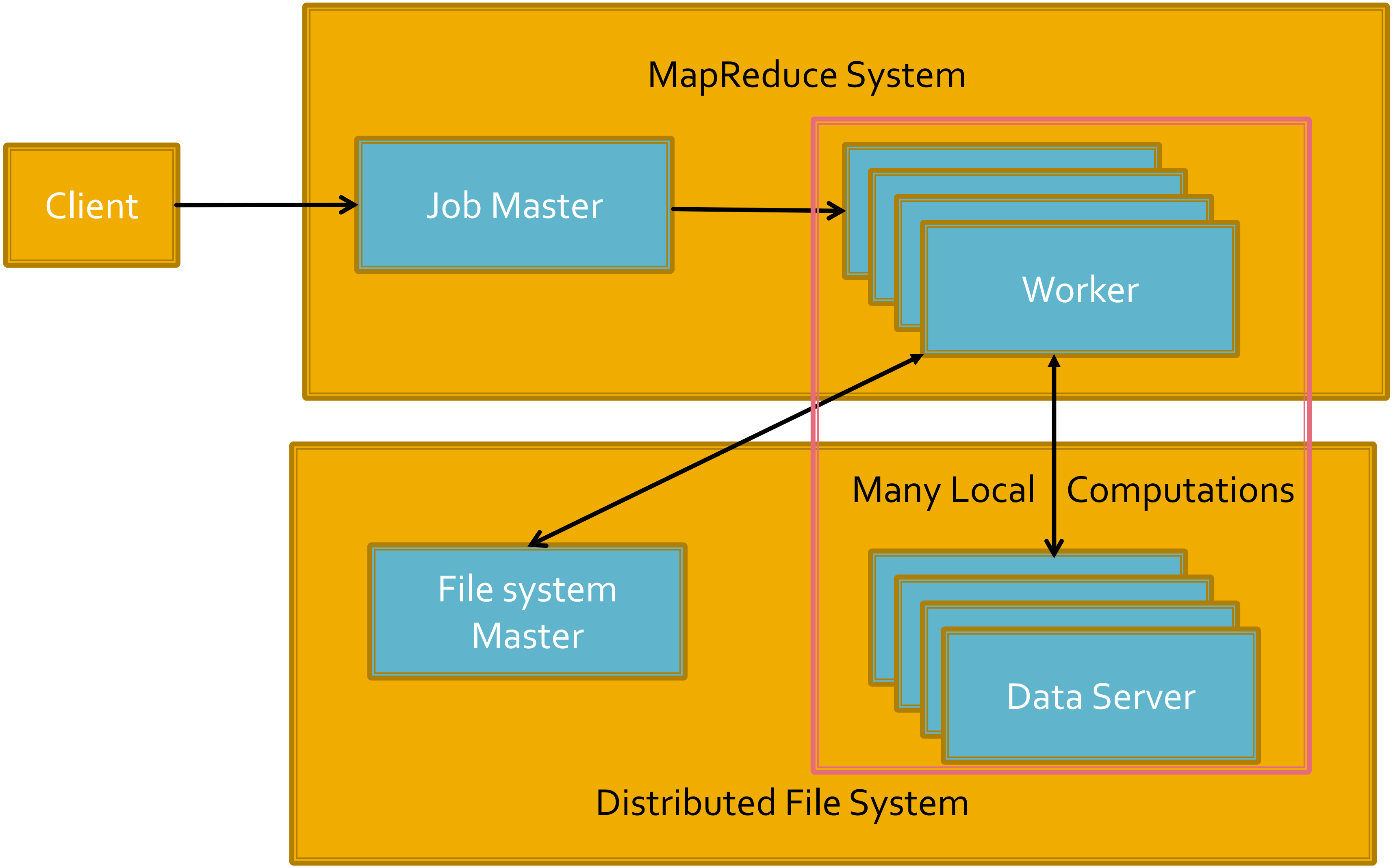
Task Requirement:

- Fault-Tolerance
- Load Balancing
- Parallelization

Advantages of MapReduce

- Automatic parallelization & distribution
- Fault tolerance
- Automatic scheduling and load balancing

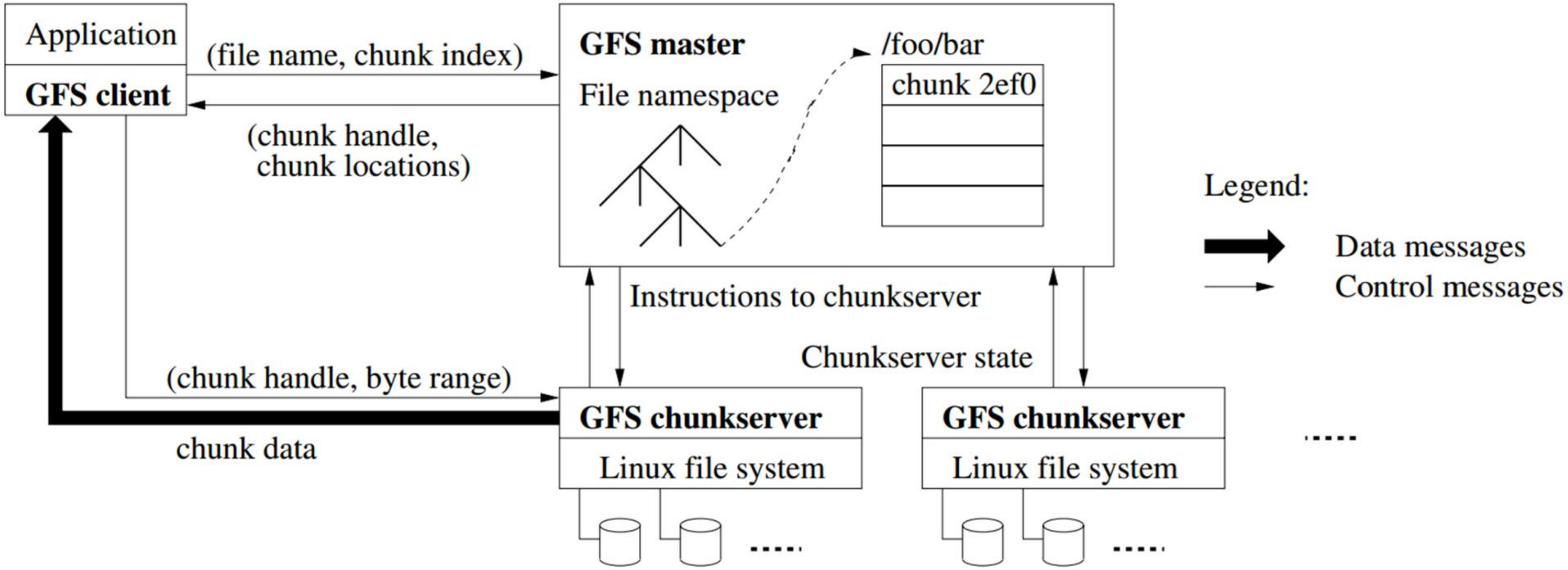
Overall Framework



Overview of MapReduce Framework

Distributed File Systems

- Master/ Slaves architecture
- Files are divided into chunks (small size)
- Chunks are distributed over multiple computers
- Replication of chunks is adopted

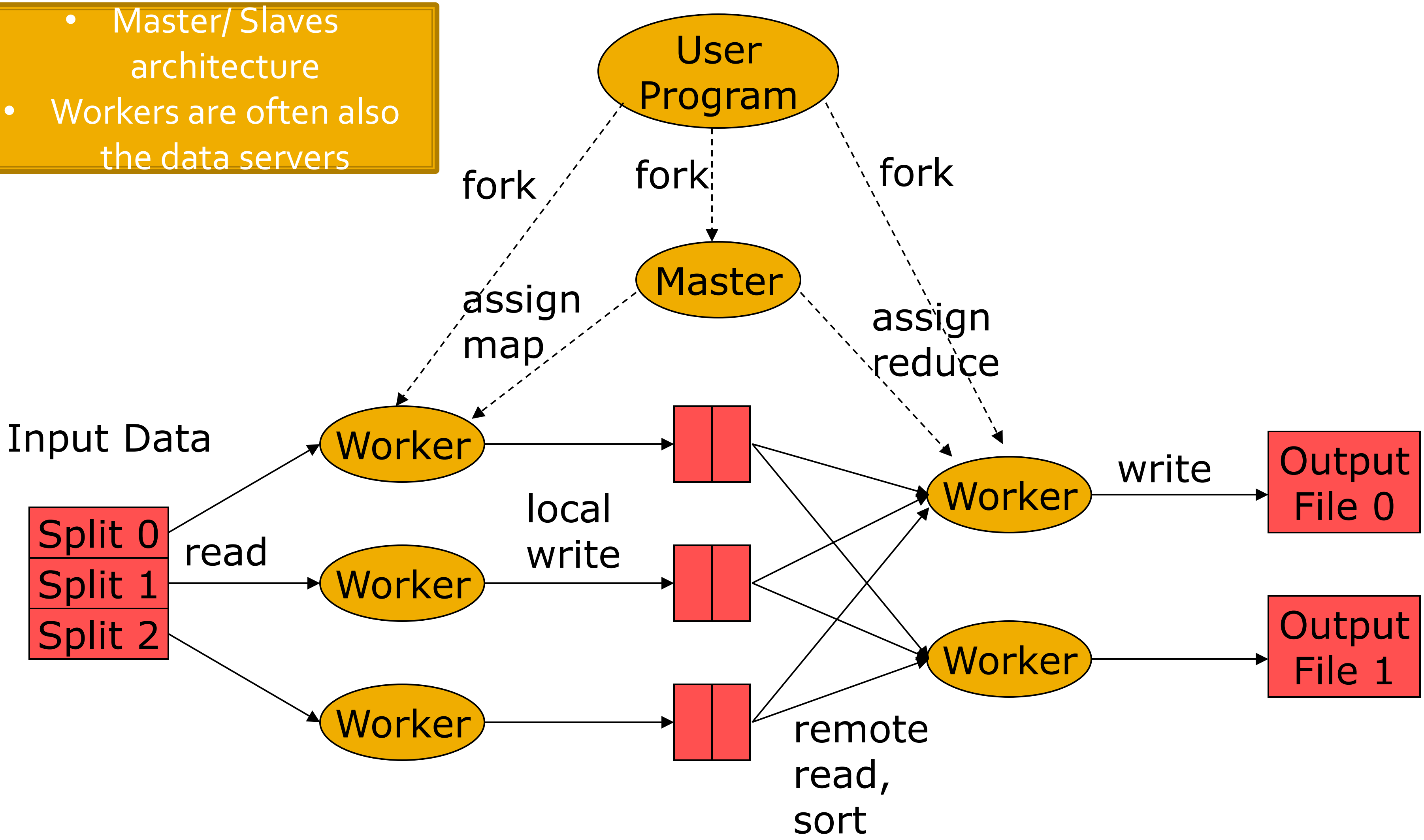


Functions in the Model

- Map
  - Process a key/value pair to generate intermediate key/value pairs
- Reduce
  - Merge all intermediate values associated with the same key
- Partition
  - By default : `hash(key) mod R`
  - Well balanced

MapReduce Framework

- Master/ Slaves architecture
- Workers are often also the data servers

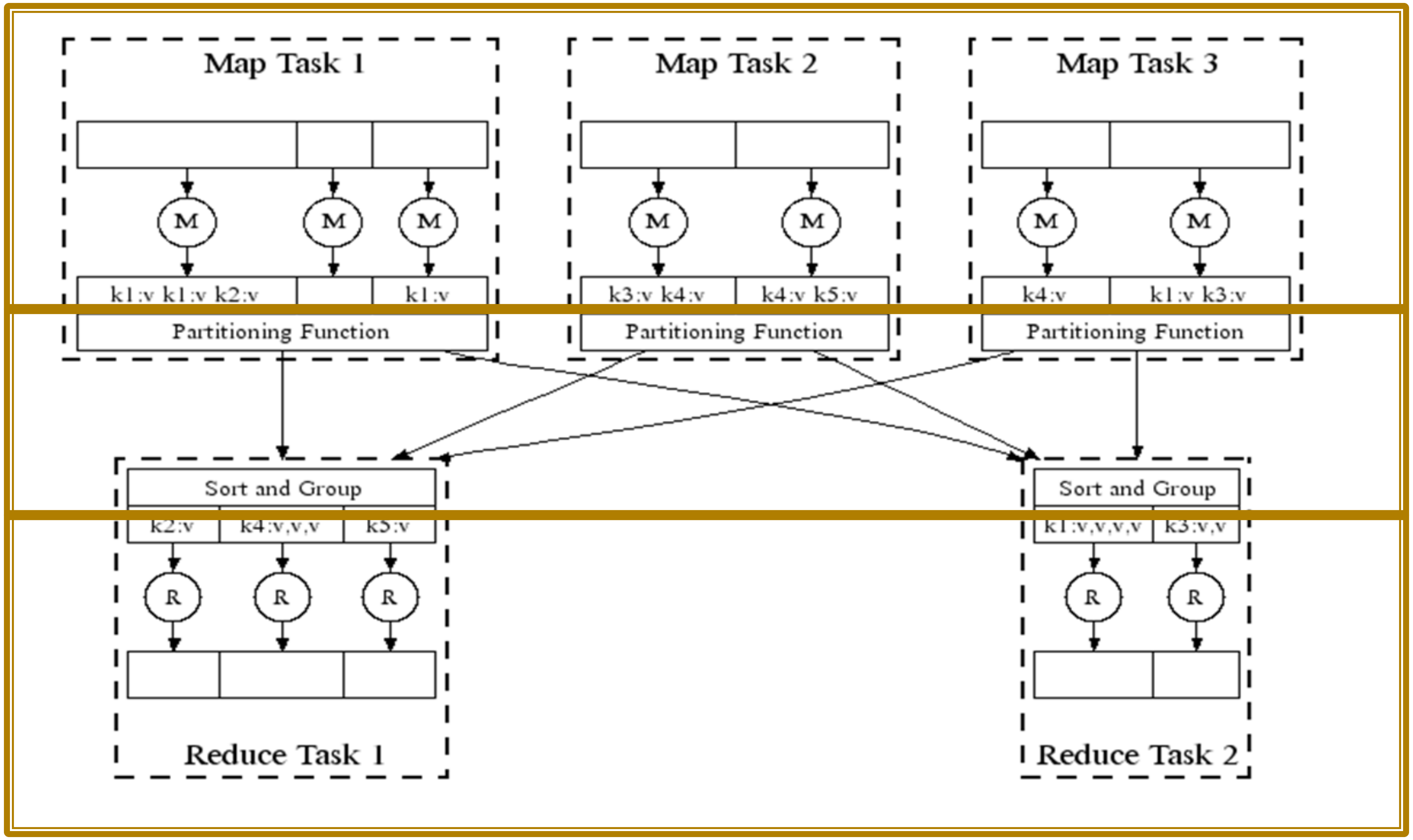


MapReduce Framework

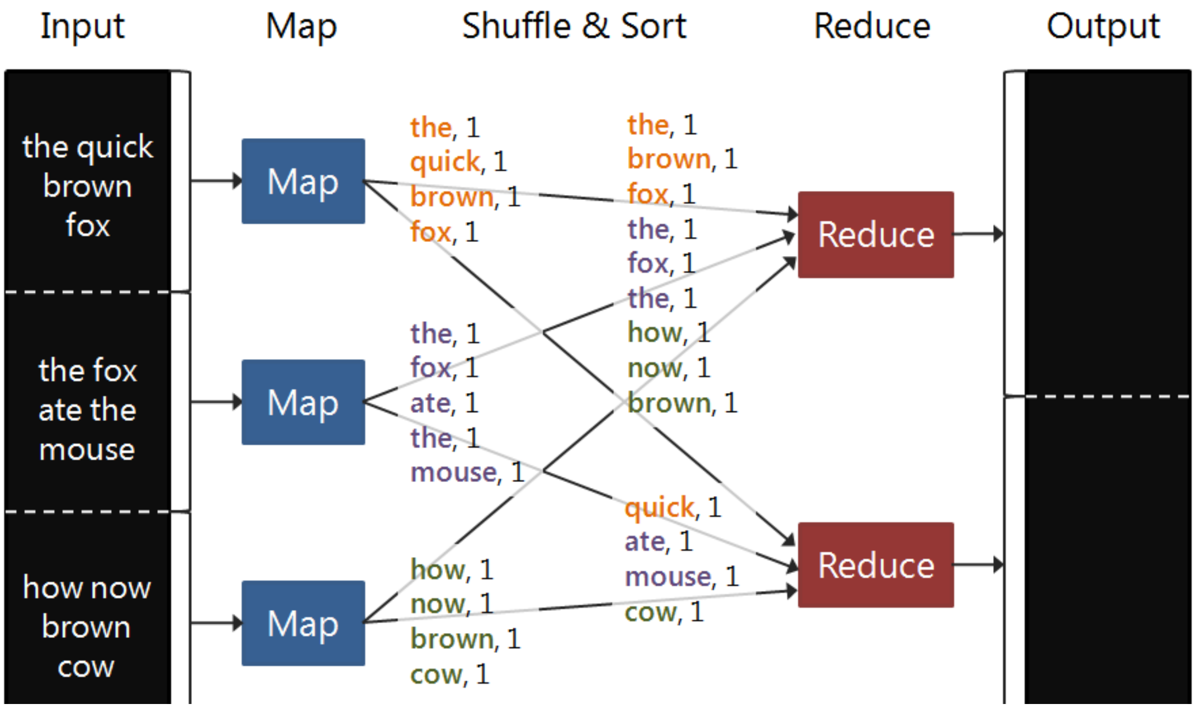
Map Stage

Shuffle Stage

Reduce Stage



Example: Word Count



Input

Map

Shuffle & Sort

Reduce

Output

the quick brown fox

the fox ate the mouse

how now brown cow

Map

Map

Map

the, 1
quick, 1
brown, 1
fox, 1

the, 1
fox, 1
ate, 1
the, 1
mouse, 1

how, 1
now, 1
brown, 1
cow, 1

the, 1
brown, 1
fox, 1
the, 1
fox, 1
the, 1
how, 1
now, 1
brown, 1

quick, 1
ate, 1
mouse, 1
cow, 1

Reduce

Reduce

```
map(String key, String value):
  // key: document name
  // value: document contents
  for each word w in value:
    EmitIntermediate(w, "1");
```

```
reduce(String key, Iterator values
  // key: a word
  // values: a list of counts
  int result = 0;
  for each v in values:
    result += ParseInt(v);
  Emit(AsString(result));
```

Implementation Details

Coordination And Scheduling

- Master data structures
  - Task status: (idle, in-progress, completed)
  - Finished map tasks send the info of the intermediate files to master
  - The info of the intermediate files are pushed to the in-progress reduce tasks.

Coordination And Scheduling

- Master scheduling policy
  - Read local replicas, if possible
  - Map tasks scheduled so GFS input block replicas are on same machine or close to the machine
- Effect
  - Thousands of machines read input at local disk

How many Map and Reduce jobs?

- Make M and R much larger than the number of nodes in cluster
- One GFS chunk per map is common
- Improves dynamic load balancing and speeds up recovery from worker failure
- Usually R is smaller than M

Combiners

- Too many key/value pairs for the same key k
  - E.g., popular words in Word Count
- Pre-aggregating at mapper-Combiners
  - Usually same as reduce function
- Works only if reduce function is commutative and associative

Fault Tolerance (1)

- Handling Failures
  - Worker failure
    - Heartbeat, Workers are periodically pinged by master
      - NO response = failed worker
    - If the processor of a worker fails, the tasks of that worker are reassigned to another worker.
  - Master failure
    - Master writes periodic checkpoints
    - Another master can be started from the last state
    - If eventually the master dies, the job will be aborted

Fault Tolerance (2)

- Handling Stragglers
  - Slow workers
    - Other jobs consuming resources on machine
    - Bad disks, software errors and so on

  - When computation almost done, reschedule in-progress tasks

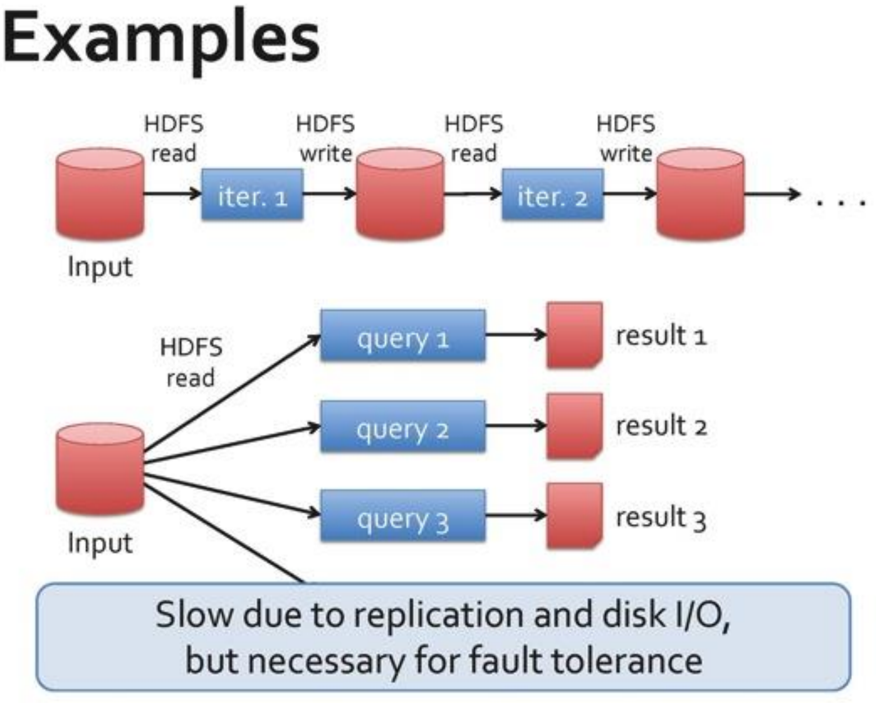
  - Whenever either the primary or the backup executions finishes, mark it as completed

Extending MapReduce

Spark Project

Disadvantages of MapReduce

- Limited choice of transformation.
- Not suitable for interactive task.
- Not suitable for iterative multi-stage process.



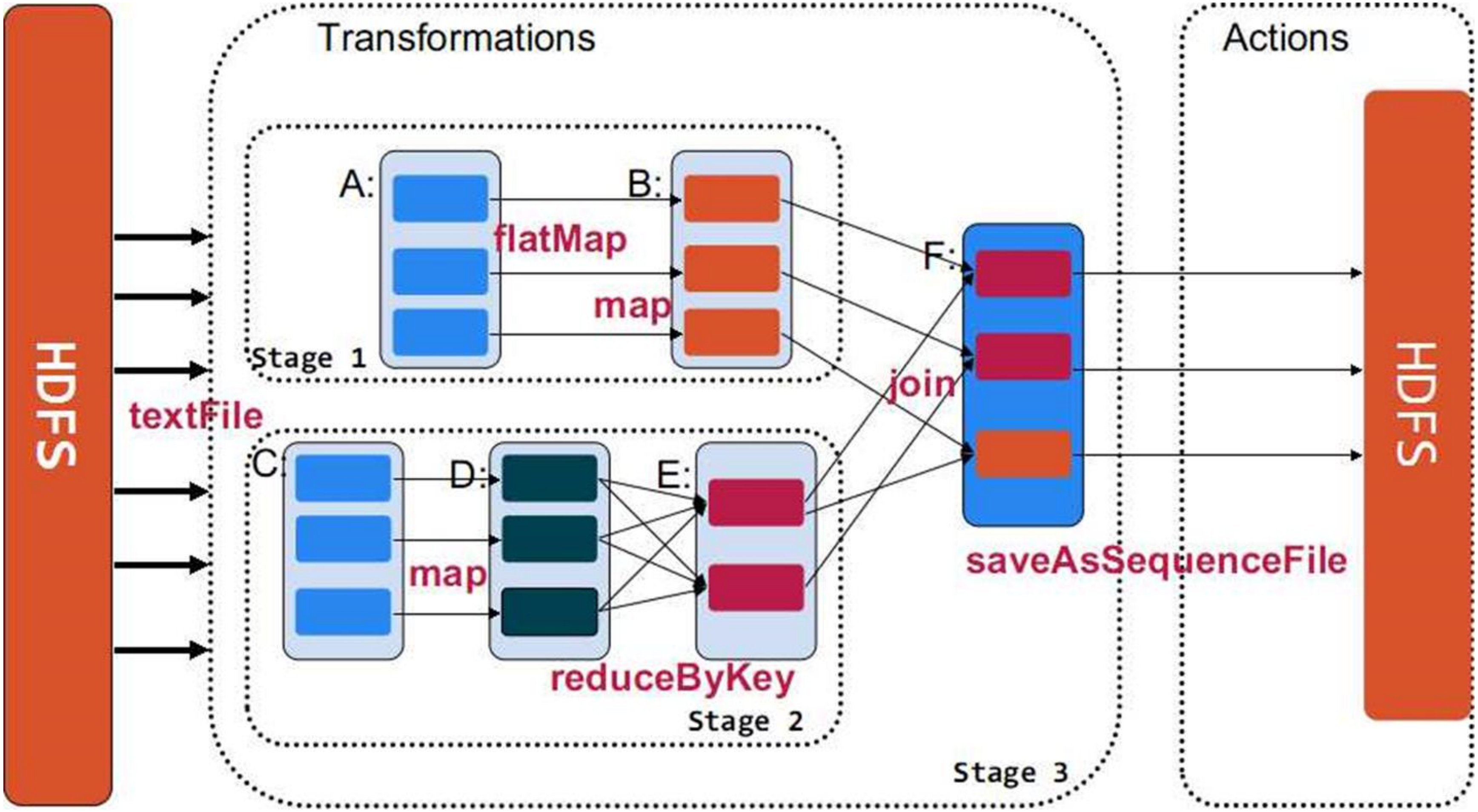
Examples

Slow due to replication and disk I/O, but necessary for fault tolerance

Spark: iterative MapReduce

- In memory computation:
  - Resilient Distributed Dataset(RDD)
- Support more transformation models than MapRduece
- Give users more control over the intermediate process
- Can coexist with Hadoop framework

Spark: iterative MapReduce



Conclusion

- MapReduce: powerful abstraction for parallel computation
  - Elicit many similar framework for parallel computation such as Spark project.

- Spark try to overcome the drawbacks of original MapReduce framework

Q&A