MapReduce

CS 351

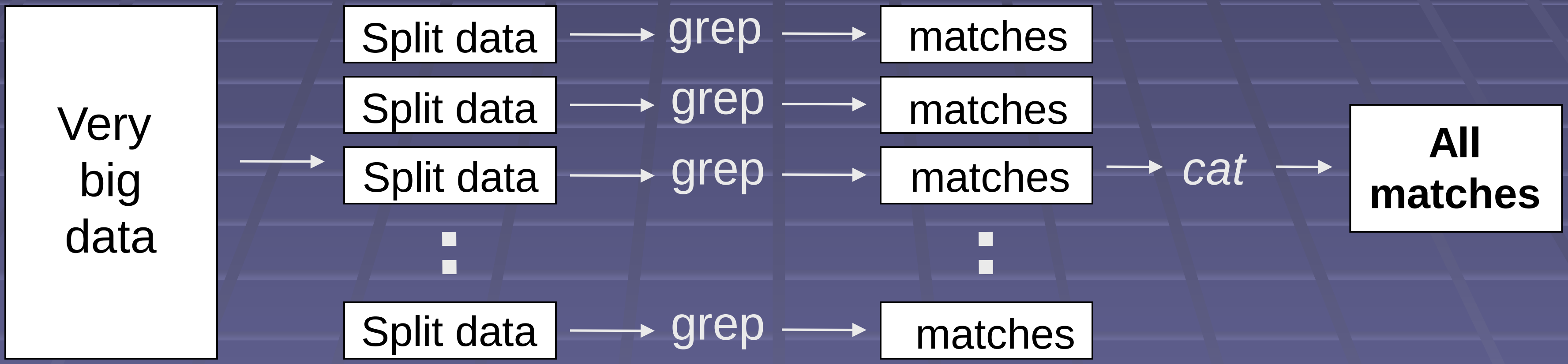
What is MapReduce

- Origin from Google, [OSDI’04]
- A simple programming model
- Functional model
- For large-scale data processing
  - Exploits large set of commodity computers
  - Executes process in distributed manner
  - Offers high availability

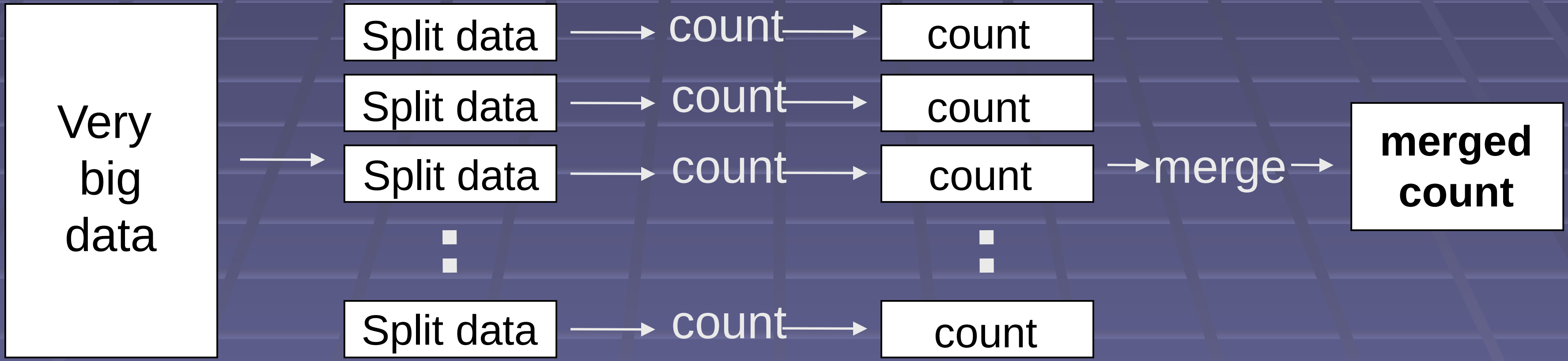
Motivation

- Lots of demands for very large scale data processing
- A certain common themes for these demands
  - Lots of machines needed (scaling)
  - Two basic operations on the input
    - Map
    - Reduce

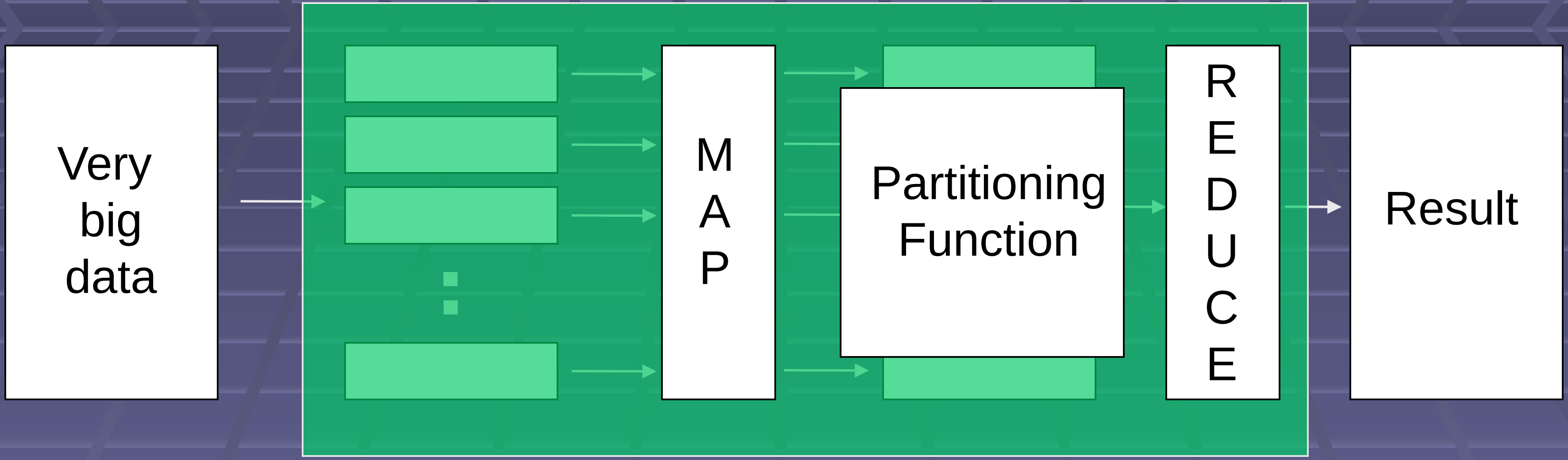
Distributed Grep



Distributed Word Count



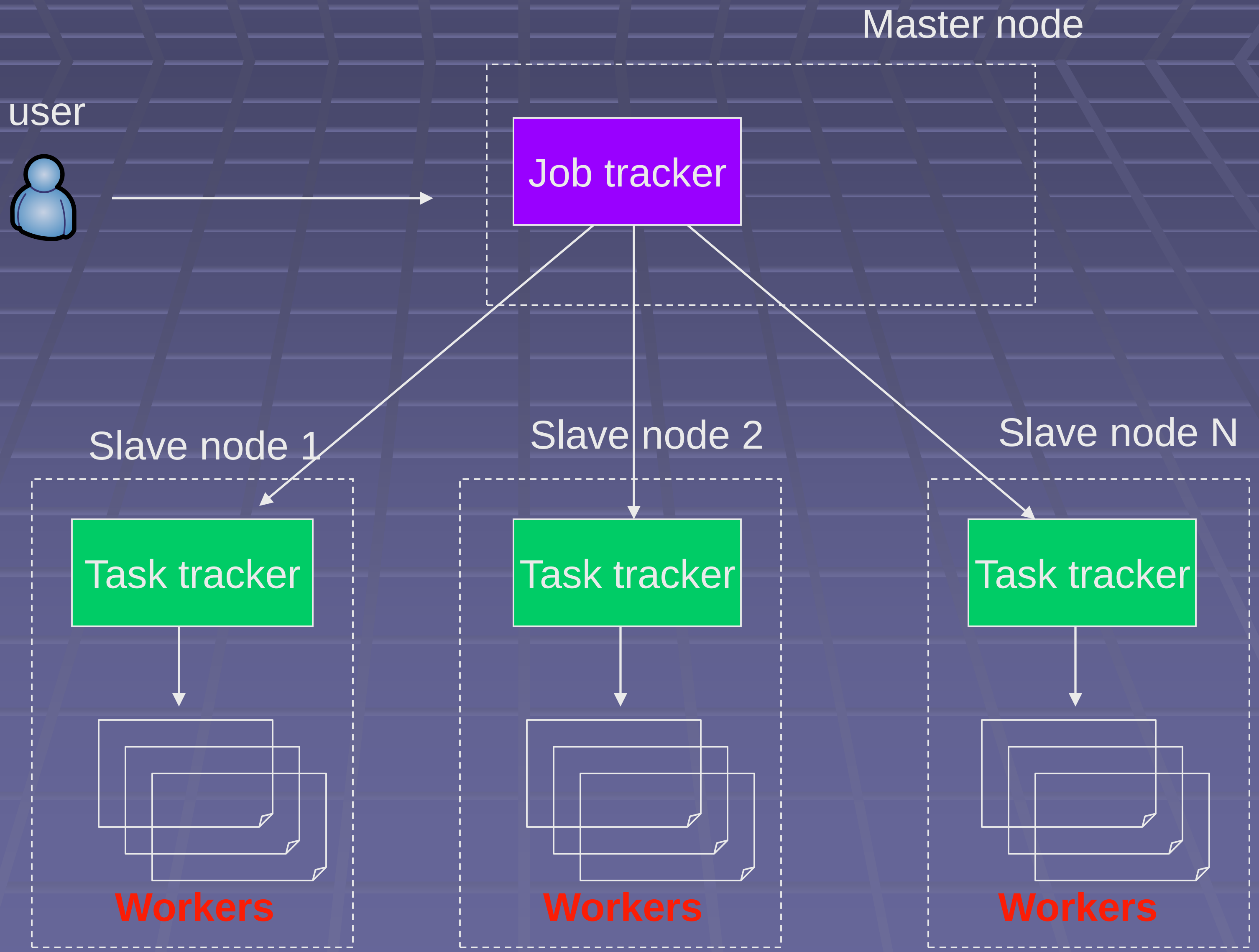
Map+Reduce



- Map:
  - Accepts *input* key/value pair
  - Emits *intermediate* key/value pair
- Reduce :
  - Accepts *intermediate* key/value* pair
  - Emits *output* key/value pair

The design and how it works

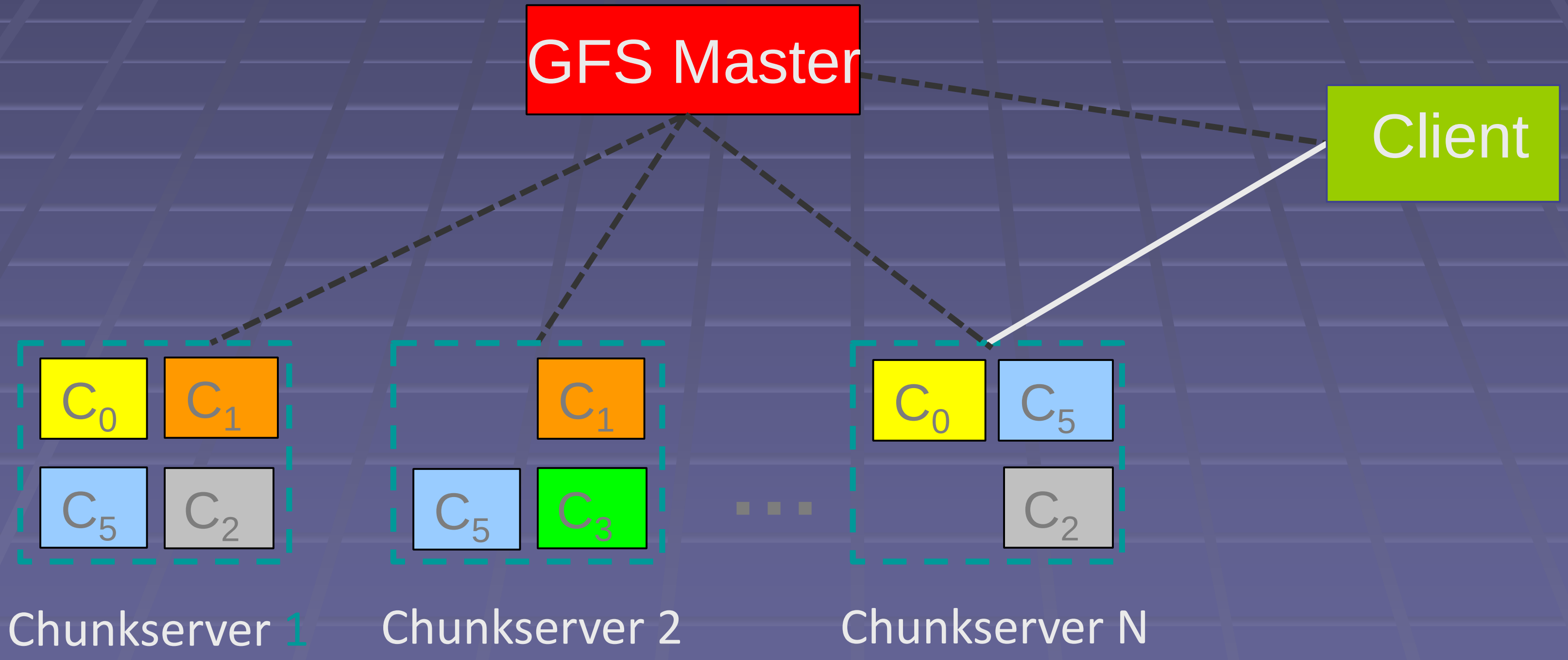
Architecture overview



GFS: underlying storage system

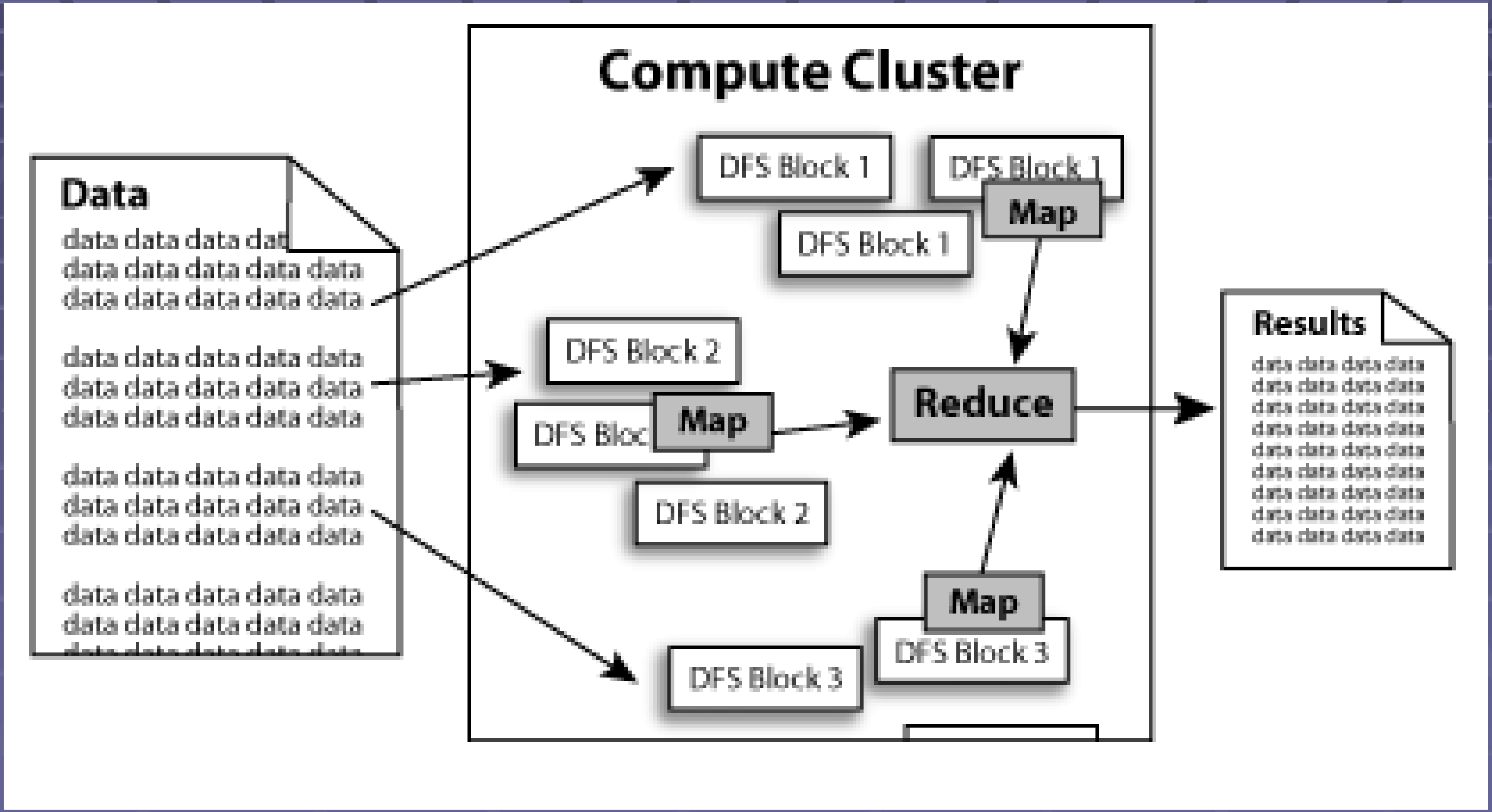
- Goal
  - global view
  - make huge files available in the face of node failures
- Master Node (meta server)
  - Centralized, index all chunks on data servers
- Chunk server (data server)
  - File is split into contiguous chunks, typically 16-64MB.
  - Each chunk replicated (usually 2x or 3x).
    - Try to keep replicas in different racks.

GFS architecture

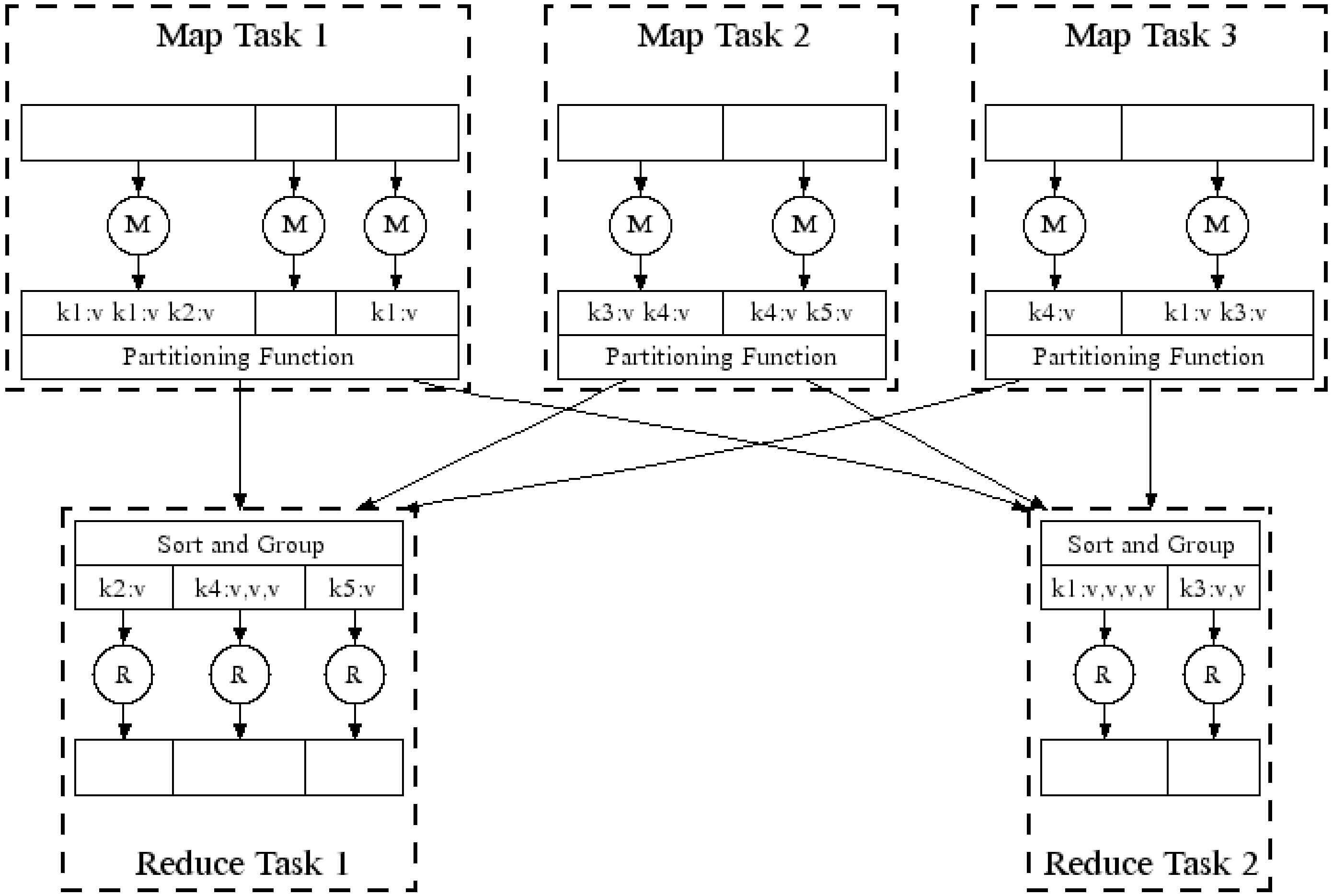


Functions in the Model

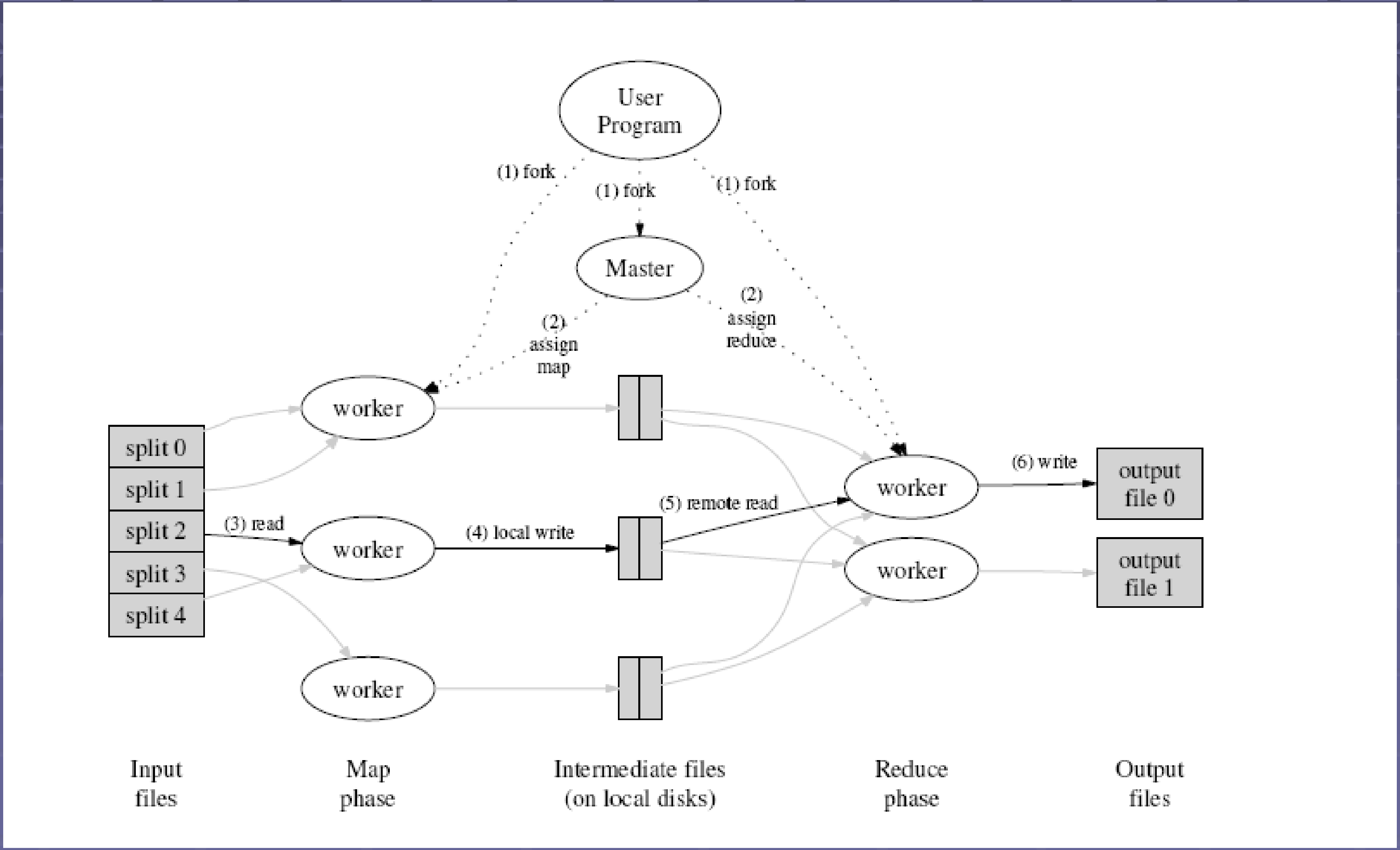
- Map
  - Process a key/value pair to generate intermediate key/value pairs
- Reduce
  - Merge all intermediate values associated with the same key
- Partition
  - By default : `hash(key) mod R`
  - Well balanced


Compute Cluster
Data
data data data data
DFS Block 1
DFS Block 1
DFS Block 1
Map
DFS Block 2
DFS Block 2
DFS Block 2
Map
DFS Block 3
DFS Block 3
Map
Reduce
Results

Parallel Execution



How does it work?



Locality issue

- Master scheduling policy
  - Asks GFS for locations of replicas of input file blocks
  - Map tasks typically split into 64MB (== GFS block size)
  - Map tasks scheduled so GFS input block replica are on same machine or same rack
- Effect
  - Thousands of machines read input at local disk speed
  - Without this, rack switches limit read rate

Fault Tolerance

- Reactive way
  - 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 checkpointed state
    - If eventually the master dies, the job will be aborted

Fault Tolerance

- Proactive way (**Redundant Execution**)
  - The problem of “stragglers” (slow workers)
    - Other jobs consuming resources on machine
    - Bad disks with soft errors transfer data very slowly
    - Weird things: processor caches disabled (!!)

  - When computation almost done, reschedule in-progress tasks
  - Whenever either the primary or the backup executions finishes, mark it as completed

Fault Tolerance

- Input error: bad records
  - Map/Reduce functions sometimes fail for particular inputs
  - Best solution is to debug & fix, but not always possible
  - On segment fault
    - Send UDP packet to master from signal handler
    - Include sequence number of record being processed
  - Skip bad records
    - If master sees two failures for same record, next worker is told to skip the record