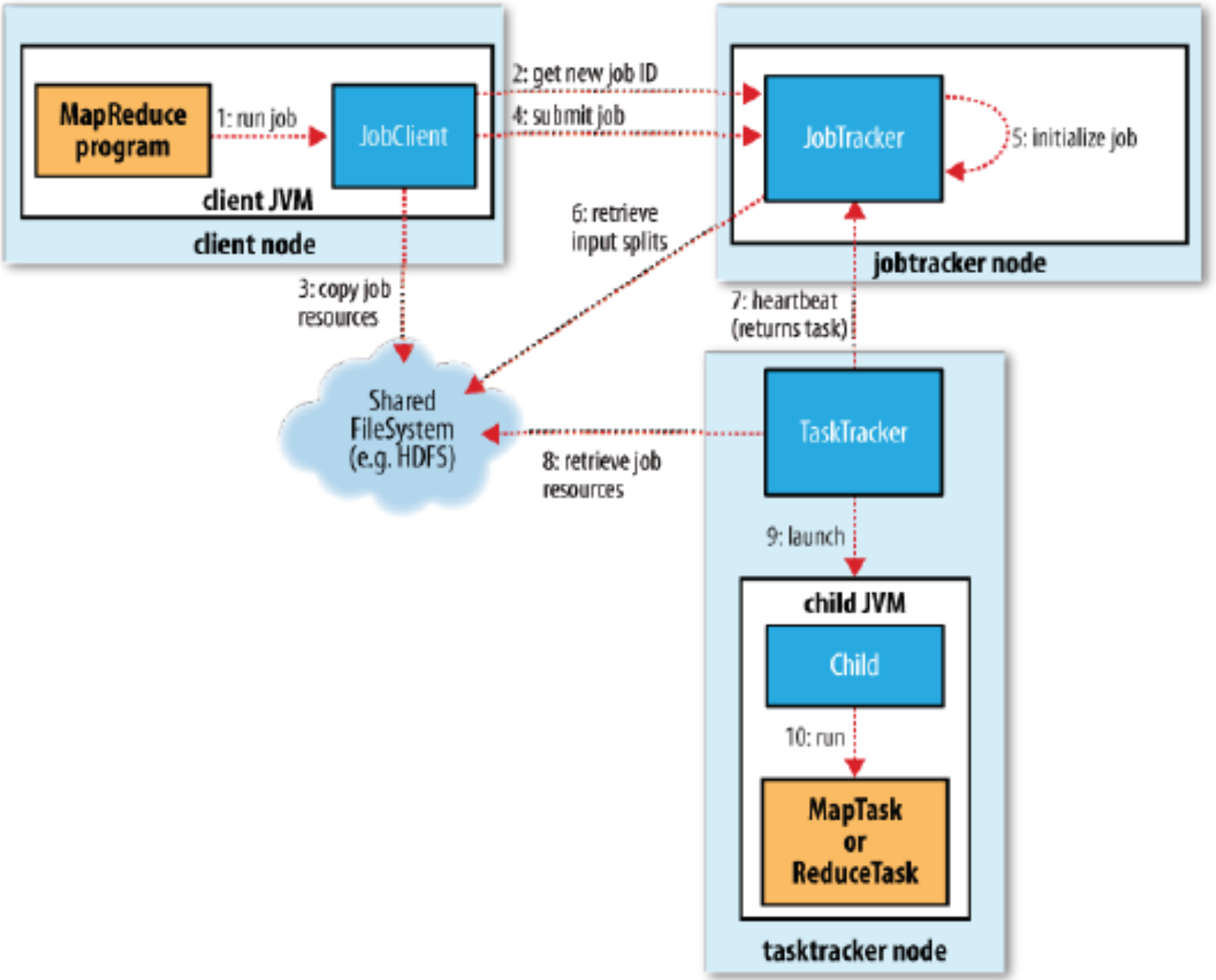
How MapReduce Works

資碩一 戴睿宸

MapReduce Entities

four independent entities:

- The client
- The jobtracker
- The tasktrackers
- The distributed filesystem


MapReduce program
1: run job
JobClient
client JVM
client node
2: get new job ID
4: submit job
JobTracker
5: initialize job
jobtracker node
6: retrieve input splits
3: copy job resources
7: heartbeat (returns task)
Shared FileSystem (e.g. HDFS)
8: retrieve job resources
TaskTracker
9: launch
child JVM
Child
10: run
MapTask or ReduceTask
tasktracker node

Steps

1. Asks the jobtracker for a new job ID
2. Checks the output specification of the job
3. Computes the input splits for the job
4. Copies the resources needed to run the job
5. Tells the jobtracker that the job is ready for execution

Job Submission

`runJob()`

- polls the job's progress once a second
- reports the progress to the console if it has changed since the last report
- When the job is complete, if it was successful, the job counters are displayed.

Job Initialization

- job scheduler first retrieves the input splits computed by the JobClient from the shared filesystem
- It then creates one map task for each split.
- The number of reduce tasks is determined by the `mapred.reduce.tasks` property
- Tasks are given IDs at this point

Task Assignment

- Tasktrackers run a simple loop that periodically sends heartbeat method calls to the jobtracker.
- As a part of the heartbeat, a tasktracker will indicate whether it is ready to run a new task, and if it is, the jobtracker will allocate it a task

Task Assignment

- In the optimal case, the task is *data-local*
- Alternatively, the task may be *rack-local*
- tasks are neither data-local nor rack-local and retrieve their data from a different rack from the one they are running on

Task Execution

- First, it localizes the job JAR by copying it from the shared filesystem
- Second, it creates a local working directory
- Third, it creates an instance of TaskRunner to run the task.rectory for the task
- TaskRunner launches a new Java Virtual Machine to run each task in

Task Execution progress

- When a task is running, it keeps track of its *progress*, that is, the proportion of the task completed.
- For map tasks, this is the proportion of the input that has been processed.
- For reduce tasks, it’s the estimated proportion

Task Execution

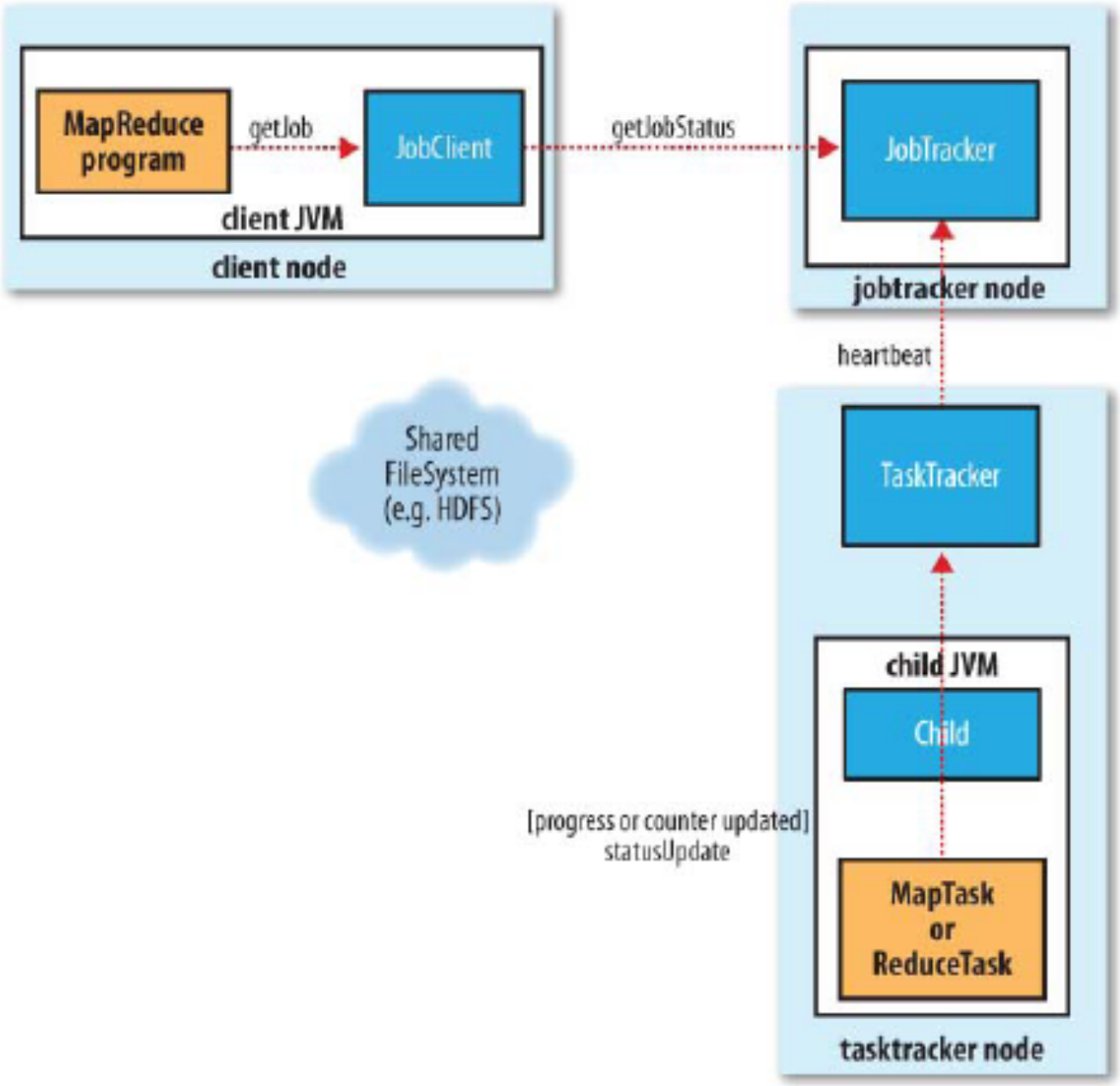
- If a task reports progress, it sets a flag to indicate that the status change should be sent to the tasktracker.

- Meanwhile, the tasktracker is sending heartbeats to the jobtracker every five seconds

Job Completion

- jobtracker receives a notification that the last task for a job is complete, it changes the status for the job to “successful.”

- JobClient polls for status, it learns that the job has completed successfully, so it prints a message to tell the user, and then returns from the runJob() method.


MapReduce program
getJob
JobClient
client JVM
client node
getJobStatus
JobTracker
jobtracker node
heartbeat
Shared FileSystem (e.g. HDFS)
TaskTracker
child JVM
Child
[progress or counter updated]
statusUpdate
MapTask or ReduceTask
tasktracker node

Task Failure

- Wihle the child JVM reports the error back to its parent tasktracker, The tasktracker marks the task attempt as

  failed, freeing up a slot to run another task.

- The tasktracker notices that it hasn’t received a progress update for a while, and proceeds to mark the task as failed. (normally 10 min)

Task Failure

- When the jobtracker is notified of a task attempt that has failed , it will reschedule execution of the task
- The jobtracker will try to avoid rescheduling the task on a tasktracker where it has previously failed
- By default, if any task fails more than four times, the whole job fails (editable)

Tasktracker Failure

- If a tasktracker fails by crashing, or running very slowly, it will stop sending heartbeats to the jobtracker

- The jobtracker arranges for map tasks that were run and completed successfully on that tasktracker to be rerun if they belong to incomplete jobs and reschedule.

Jobtracker Failure

- Hadoop has no mechanism for dealing with failure of the jobtracker
- It is possible that a future releaseof Hadoop will remove this limitation by running multiple jobtrackers, only one of which is the primary jobtracker at any time

Job Scheduling

- Basically FIFO
- setting a job's priority was enabled, which take one of the values VERY_HIGH, HIGH, NORMAL, LOW, VERY_LOW
- priorities do not support *preemption*

The Fair Scheduler

- For Multiuser
- The Fair Scheduler aims to give every user a fair share of the cluster capacity over time.
- If a single job is running, it gets all of the cluster
- It is also possible to define custom pools with guaranteed minimum capacities and to set weighting for each pool
- The Fair Scheduler supports preemption

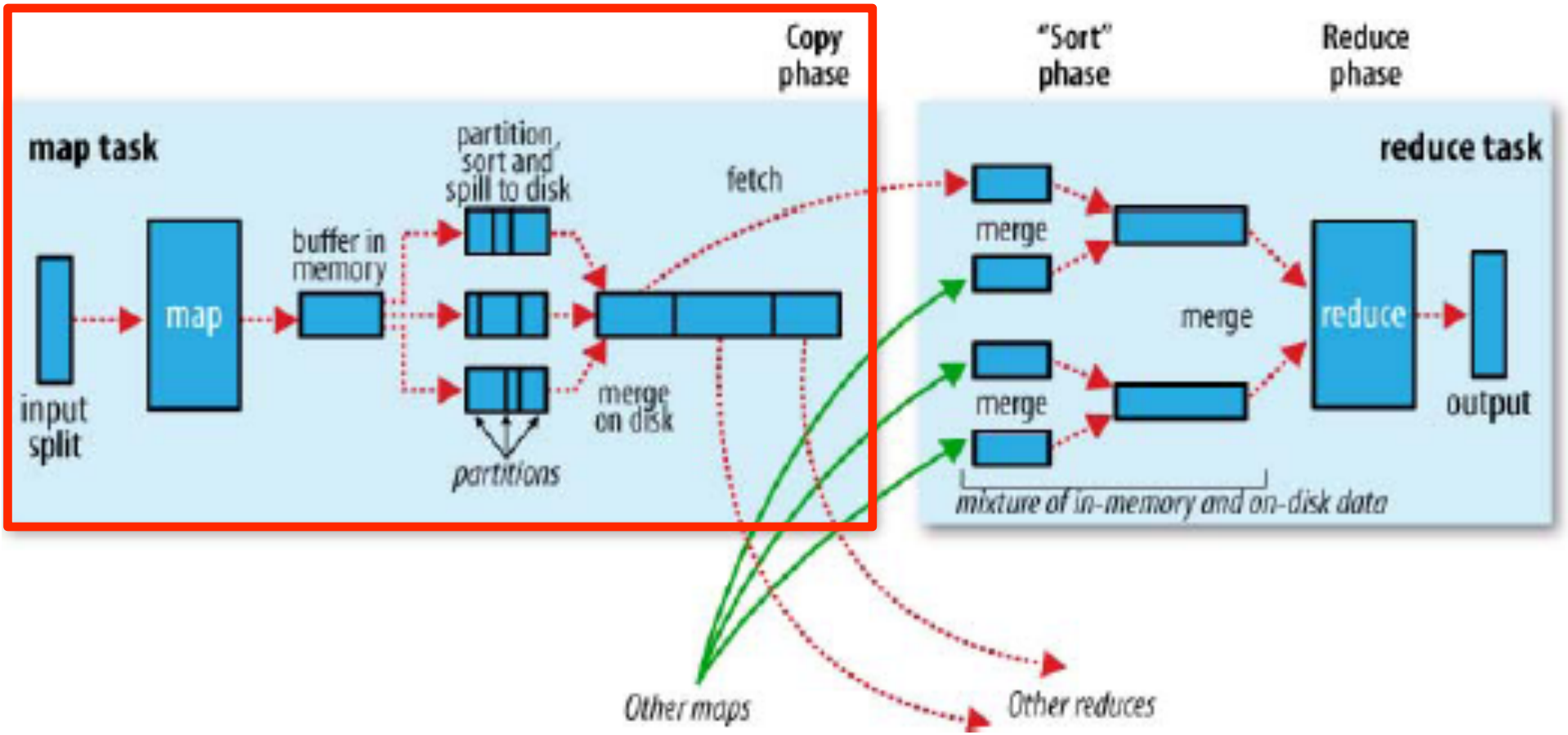
Shuffle and Sort

- MapReduce makes the guarantee that the input to every reducer is sorted by key.

- Theprocess by which the system performs the sort—and transfers the map outputs to thereducers as inputs

—is known as the *shuffle*.†

The Map Side


Copy phase
"Sort" phase
Reduce phase
map task
reduce task
partition, sort and spill to disk
fetch
buffer in memory
map
input split
merge on disk
partitions
merge
merge
merge
reduce
output
mixture of in-memory and on-disk data
Other maps
Other reduces

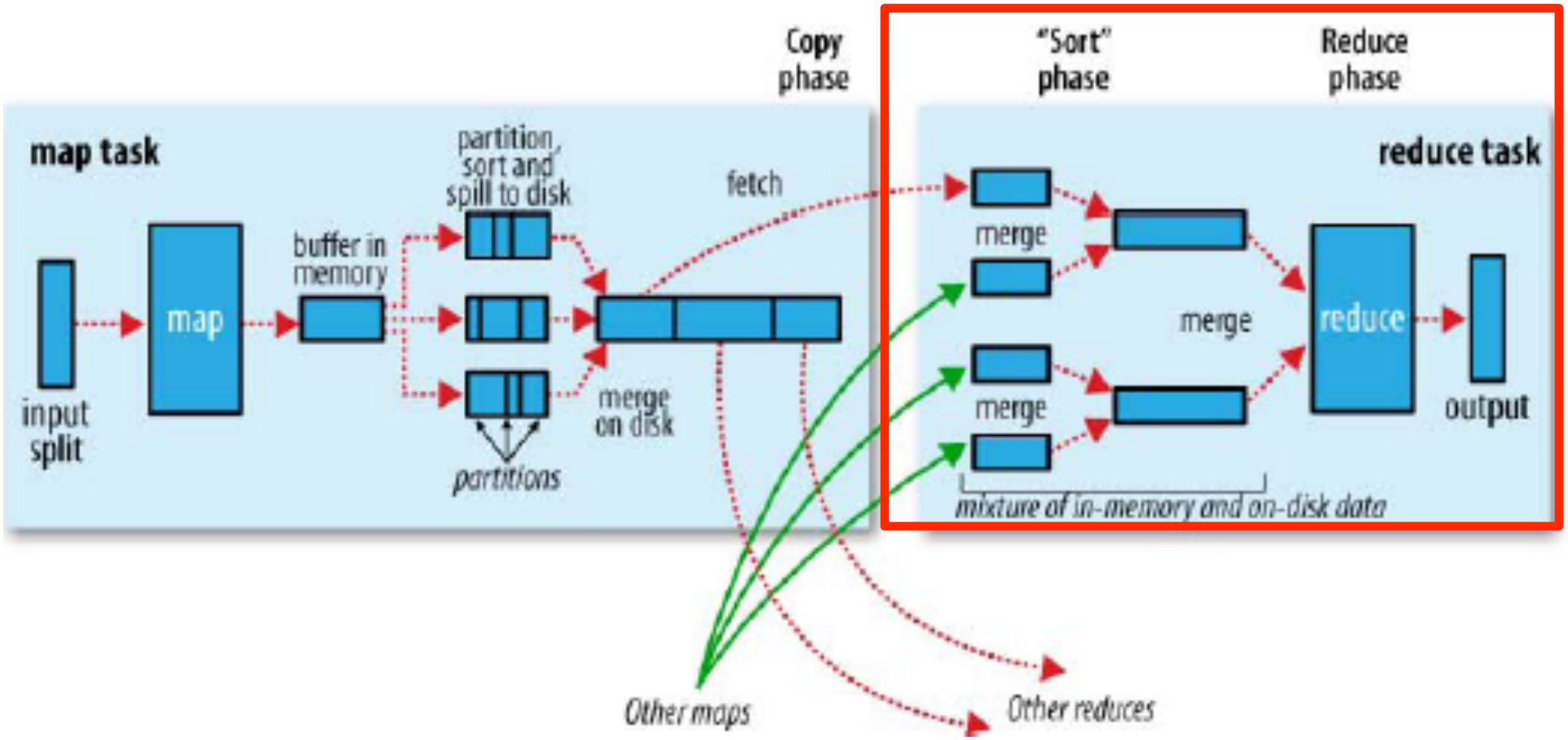
The Map Side

- Spills are written in round-robin fashion to the directories specified by the mapred.local.dir property, in a job-specific subdirectory
- Before it writes to disk, the thread first divides the data into partitions corresponding to the reducers that they will ultimately be sent to.
- The background thread performs an in-memory sort by key

The Map Side

- The spill files are merged into a single partitioned and sorted output file
- It is often a good idea to compress the map output as it is written to disk

The Reduce Side



The Reduce Side

- The map outputs are copied to the reduce tasktracker's memory if they are small enough ;otherwise, they are copied to disk.
- When all the map outputs have been copied, the reduce task moves into the sort phase (merge phase)
- The merge saves a trip to disk by directly feeding the reduce function in what is the last phase: the reduce phase. This final merge can come from a mixture of in-memory and on-disk segments.

Thank you