CS 7172
Parallel and Distributed Computation

MapReduce

Kun Suo
Computer Science, Kennesaw State University
https://kevinsuo.github.io/

Outline

- Single server vs multiple server processing
- Divide-and-conquer
- MapReduce
  - Map
  - Reduce
  - How it works
  - Fault-tolerance
  - Limitation

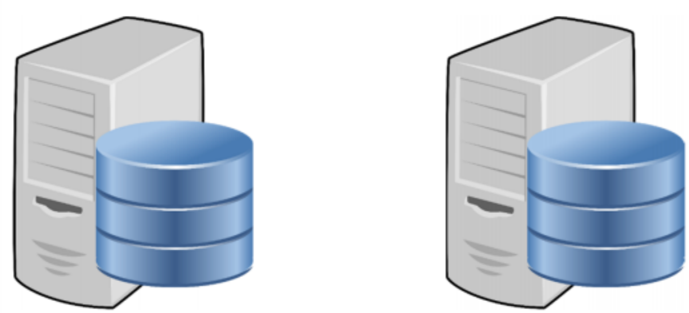
Processing Data on a single Machine

- Data on file system on disk
- Simple processing flow:

1. Read into memory
2. Process data (apply some function)
3. Write back to disk

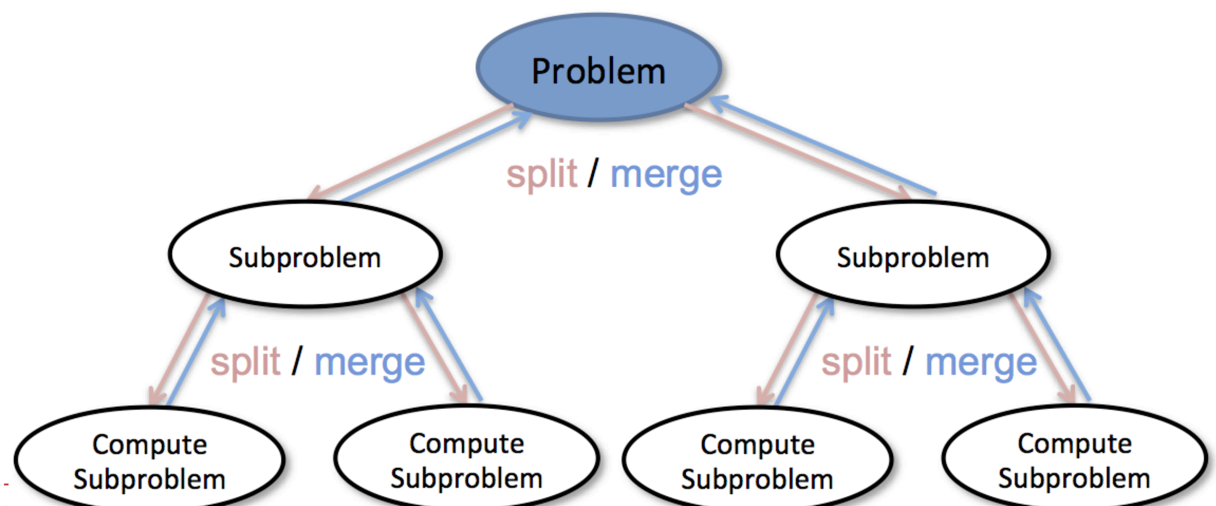
Why Multiple Servers?

- Data size larger than disk capacity
- Sequentially processing data can be slow
  - Limited by disk read/write speeds
  - Parallel processing can significantly reduce time
- Single point of failure with a single server



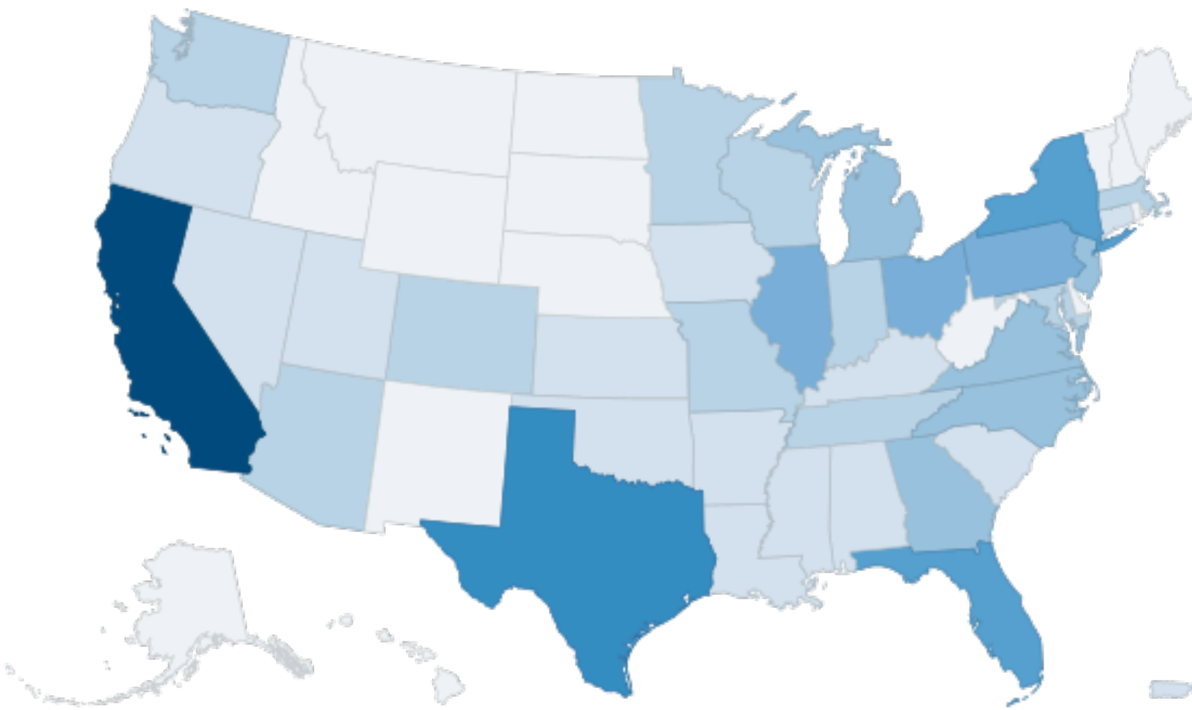
Divide-and-Conquer

- A divide-and-conquer algorithm works by recursively breaking down a problem into two or more sub-problems of the same or related type, until these become simple enough to be solved directly. The solutions to the sub-problems are then combined to give a solution to the original problem.



Divide-and-Conquer

- How to get the total population of USA?



What kind of problem can Divide-and-Conquer solve?

- Problem characteristics:
  - The scale of the problem is large or complex, and the problem can be broken down into several smaller, simple problems of the same type to solve
  - The sub-problems are independent of each other
  - The solutions of the sub-problems can be combined to obtain the solution of original problem

How to Use Divide-and-Conquer

- Decompose the original problem.
  - The original problem is decomposed into several smaller, independent sub-problems that have the same form as the original problem.
- Solve subproblems.
  - If the sub-problems are small and easy to solve, solve them directly;
  - Otherwise, solve the sub-problems recursively.
- Merging solutions
  - Merge the solutions of the sub-problems into solutions of the original problem.

MapReduce

- https://static.googleusercontent.com/media/research.google.com/en//archive/mapreduce-osdi04.pdf

MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

jeff@google.com, sanjay@google.com

Google, Inc.

Abstract

MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a *map* function that processes a key/value pair to generate a set of intermediate key/value

given day, etc. Most such computations are conceptually straightforward. However, the input data is usually large and the computations have to be distributed across hu
a
al

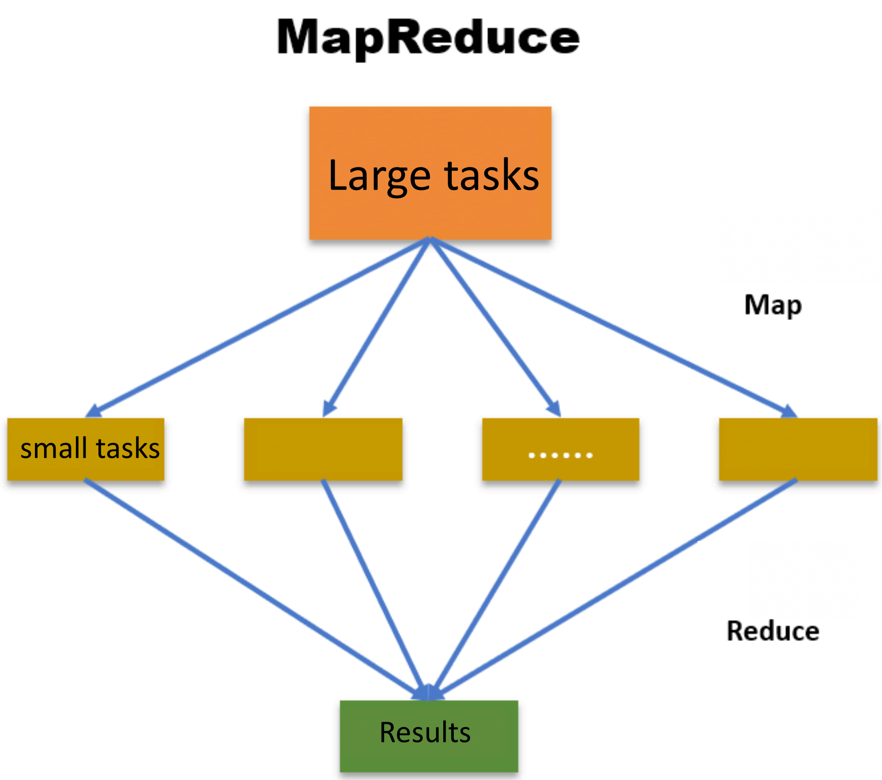
Published at OSDI 2004.
>25,000 citations

Why MapReduce?

- Automated parallelization and distribution of data and processing
- Clean, powerful, well-understood abstraction (Map and Reduce)
- Fault-tolerance
- Scalability: 1000s of servers, TBs of data, …
- Apache Hadoop: widely used open source Java implementation

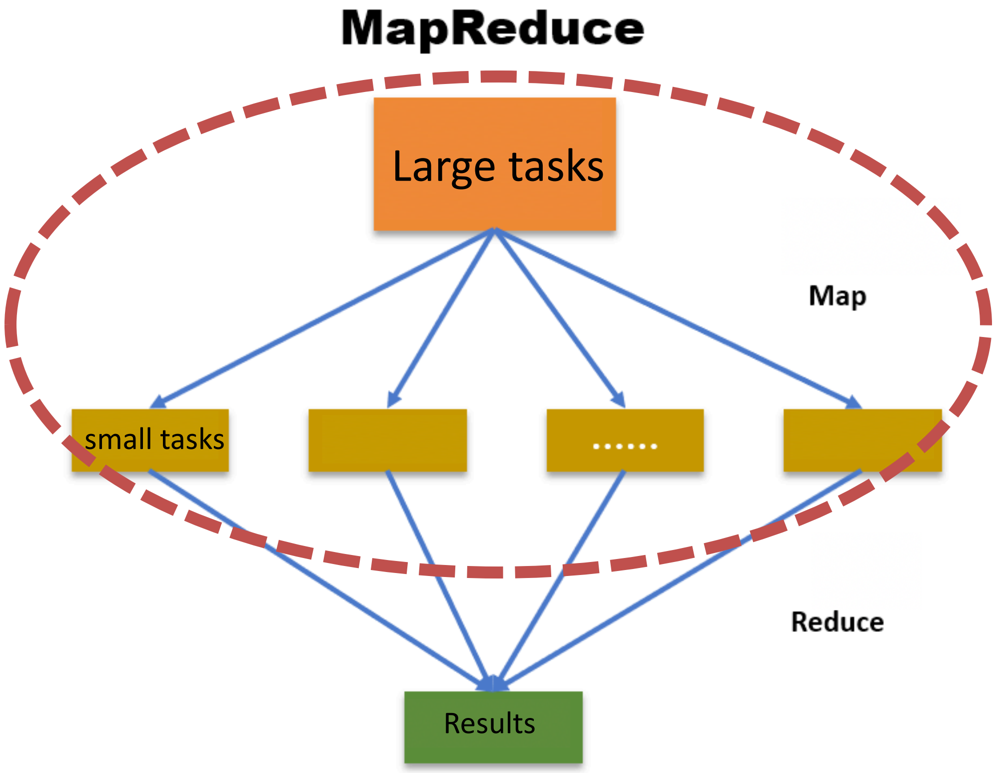
MapReduce

- MapReduce is divided into two core phases: Map and Reduce
- Map corresponds to ”divide”: complex tasks are broken down into several simple tasks for execution
- Reduce corresponds to ”conquer”: the results of the Map phase are summarized.



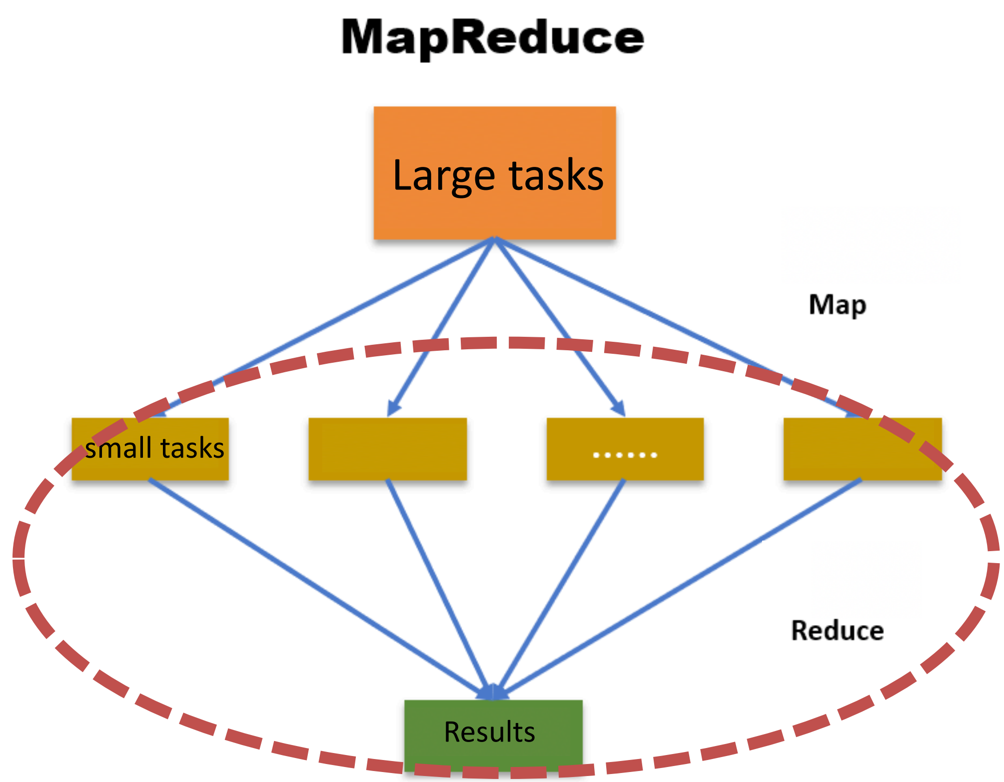
Map

- The small tasks after map have the following characteristics:
    - Compared to the original task, the divided sub-tasks are homogeneous with the original task
    - The sub-task data size and calculation size is much smaller
    - There are no dependencies between different sub-tasks, which can run independently and run in parallel



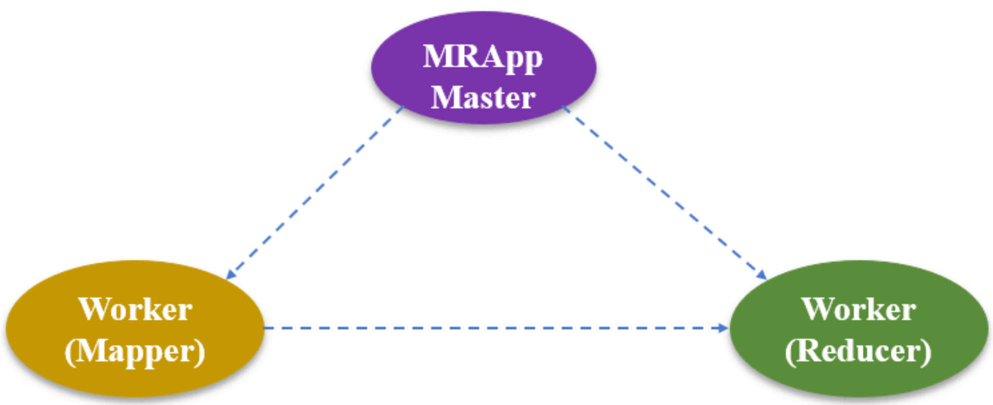
Reduce

- After the division of the sub-tasks in the first stage is completed, the calculation results of all sub-tasks are summarized to obtain the final results.



How the MapReduce works

- MapReduce includes the three components:
  - Master (MRAppMaster): is responsible for assigning tasks, coordinating the operation of tasks, and assigning map function operations to Mappers, and reduce function operations to the Reducers.
  - Mapper worker: is responsible for Map function and performing sub-tasks.
  - Reducer worker: is responsible for Reduce function and summarizing the results of each sub-task

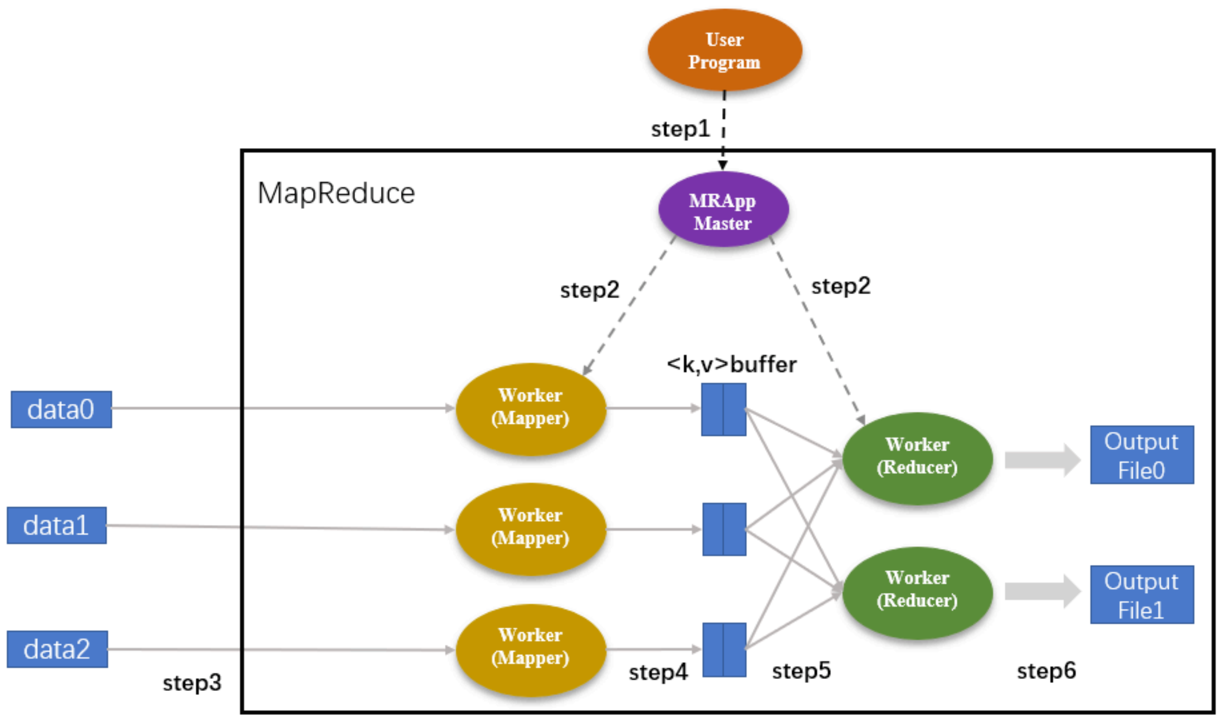


How the MapReduce works

- Workflow (step 1):

  User Program sends the task to MRAppMaster. Then, MRAppMaster splits the task into M subtasks (M is a user-defined value).

  Example: Assume that the MapReduce function divides the task into 5, of which there are 3 Map jobs and 2 Reduce jobs.

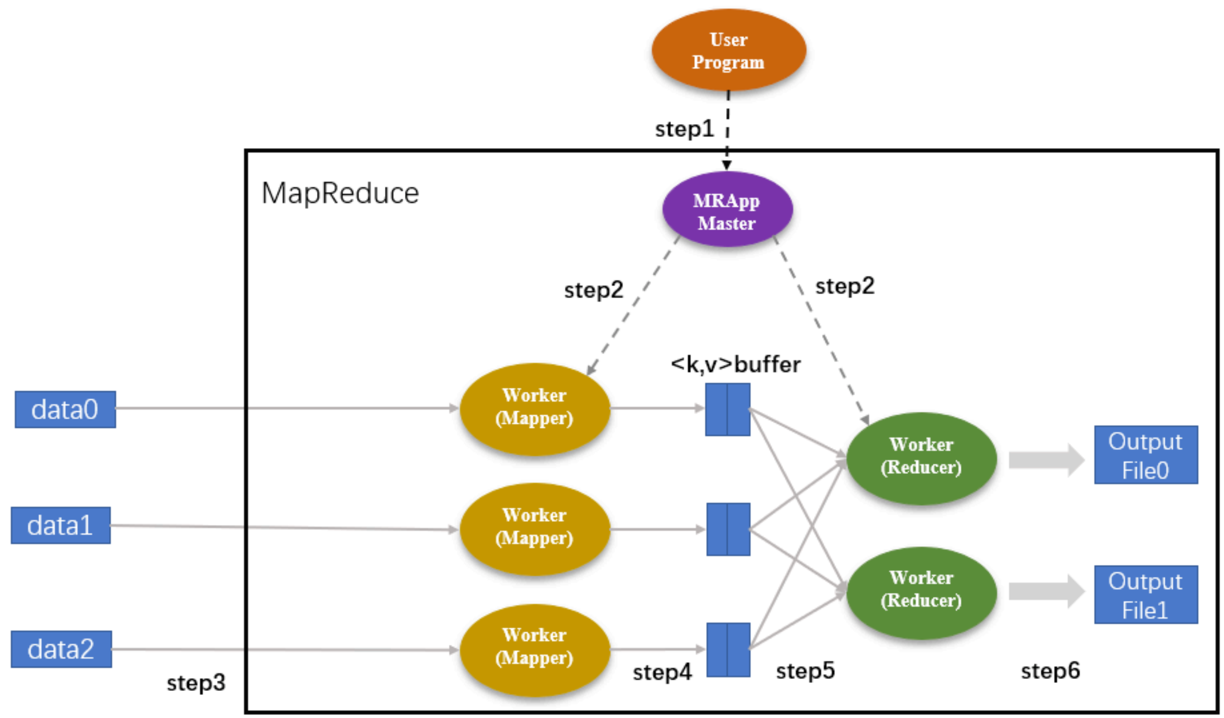


How the MapReduce works

- Workflow (step 2):

  MRAppMaster assigns Map and Reduce jobs to Mapper and Reducer, respectively.

  The number of Map jobs is the number of divided subtasks, which is 3; Reduce jobs are 2

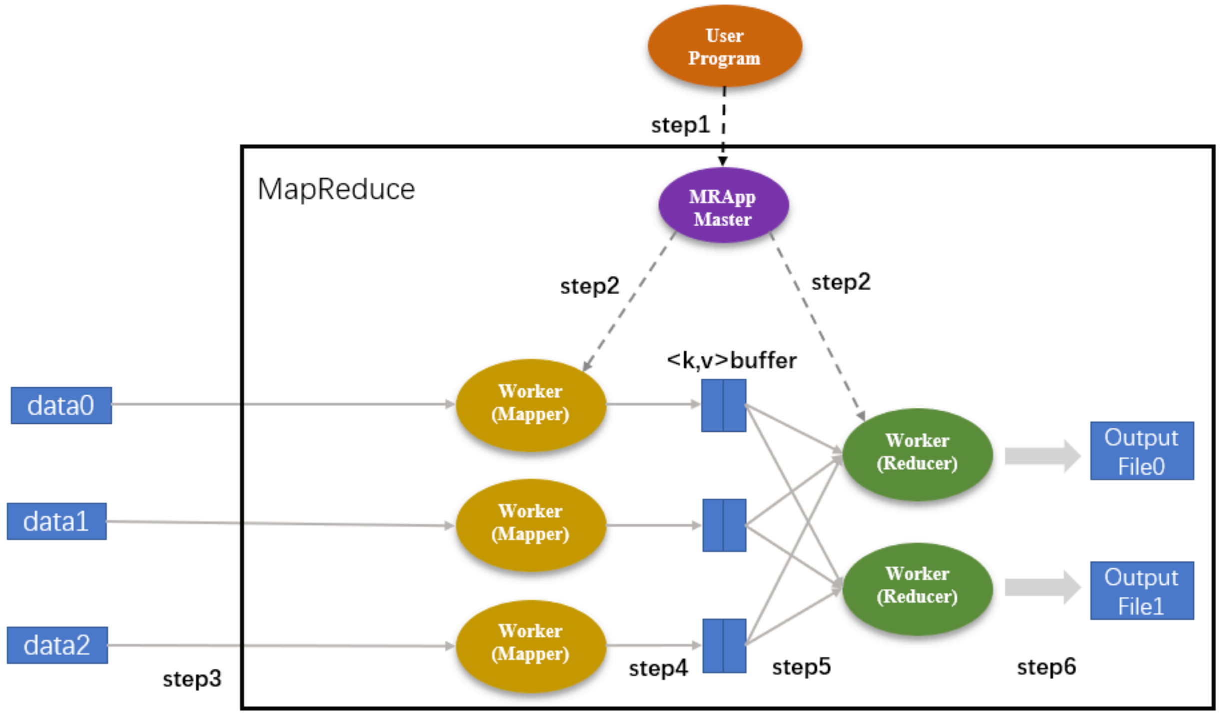


How the MapReduce works

- Workflow (step 3):

  The Mapper worker starts reading the input data of the subtask and extracts the <key, value> pairs from the input data.

  Each key-value pair is passed to the map () function as a parameter.

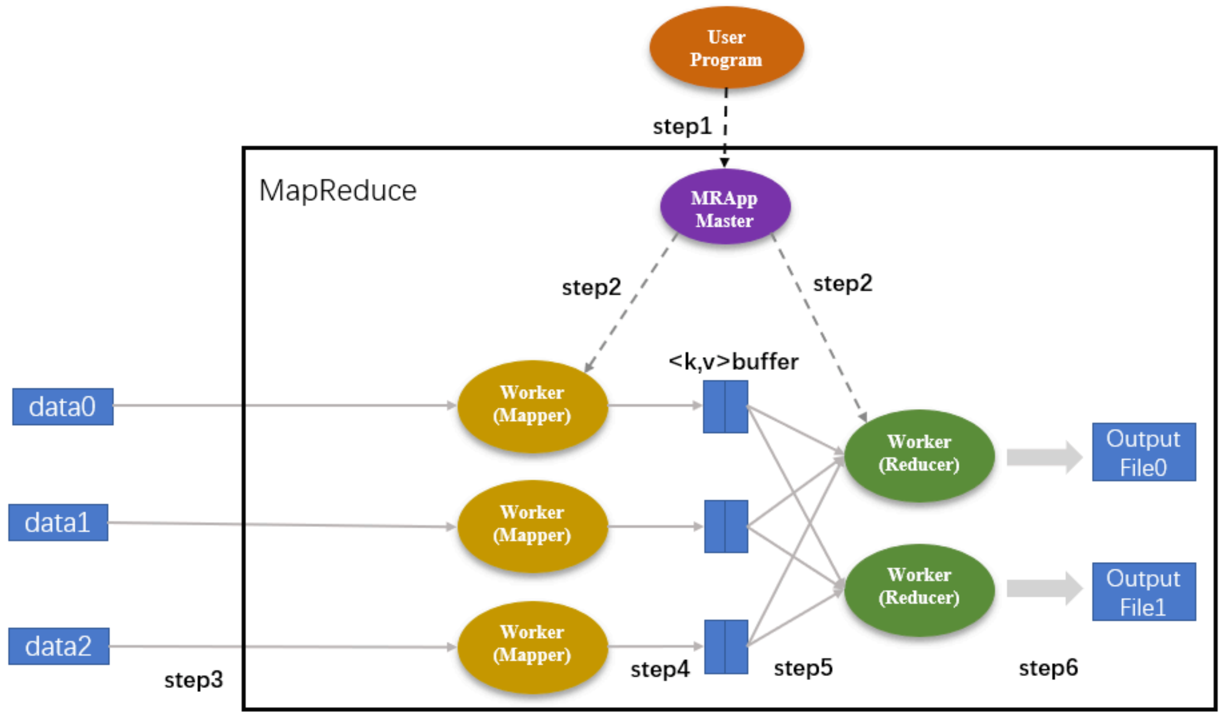


How the MapReduce works

- Workflow (step 4):

The output of the map () function is stored in the ring buffer <k, v>Buffer.

These results are periodically written to the local disk and stored in R different disk areas. Here R represents the number of Reduce jobs. In this case, R = 2. Also, the storage location of each Map result is reported to the MRAppMaster.

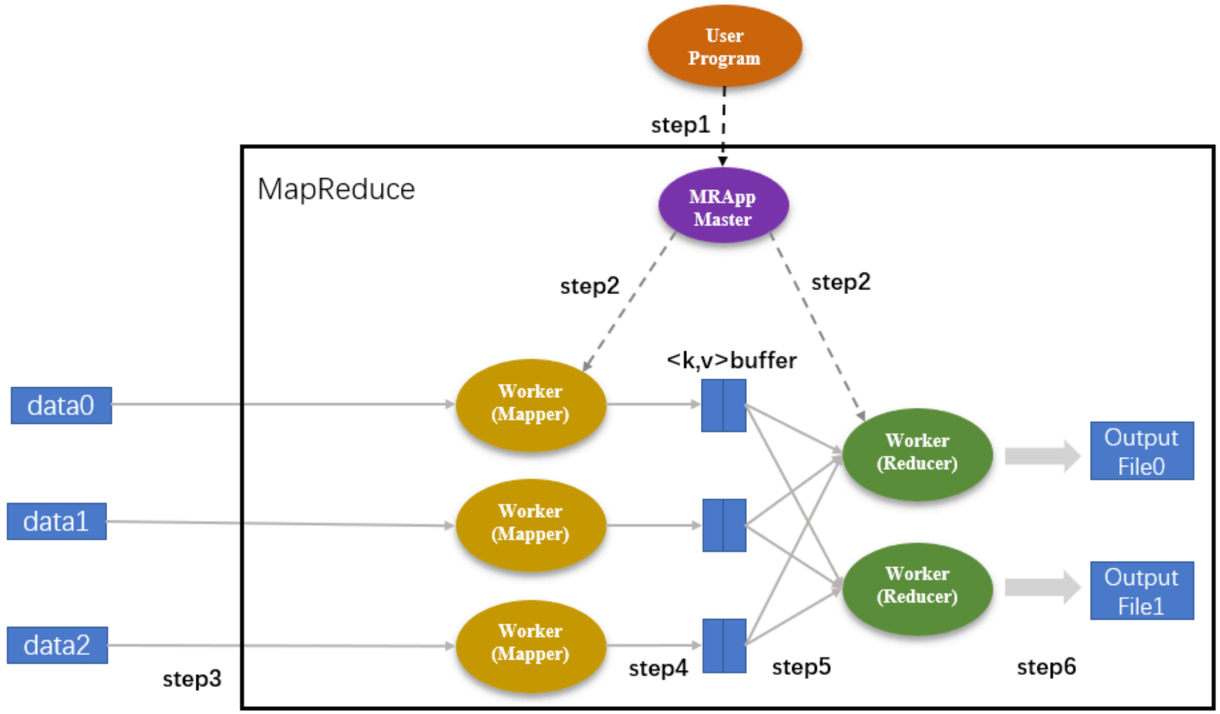


How the MapReduce works

- Workflow (step 5):

  The MRAppMaster informs the Reducer which partition it is responsible for, and the Reducer remotely reads the corresponding Map result, which is the intermediate key-value pair.

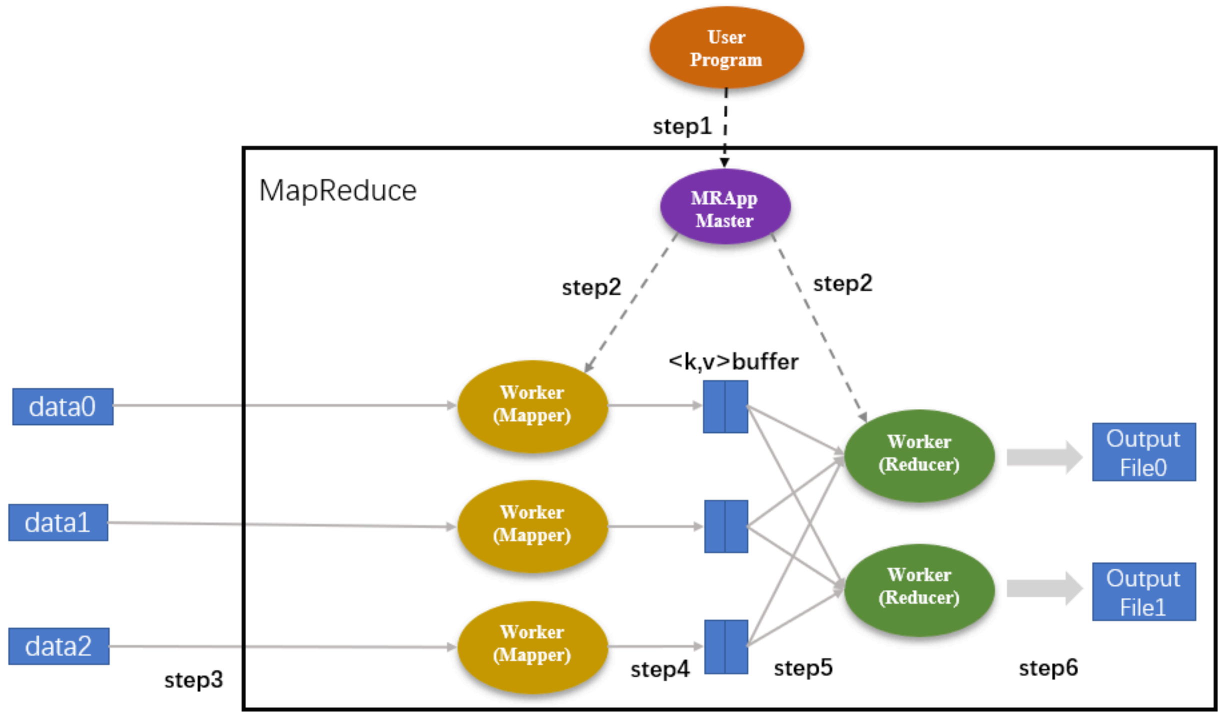
  After the Reducers read all the intermediate key-value pairs it is responsible for, it sorts and aggregates the key-value pairs of the same key value.



How the MapReduce works

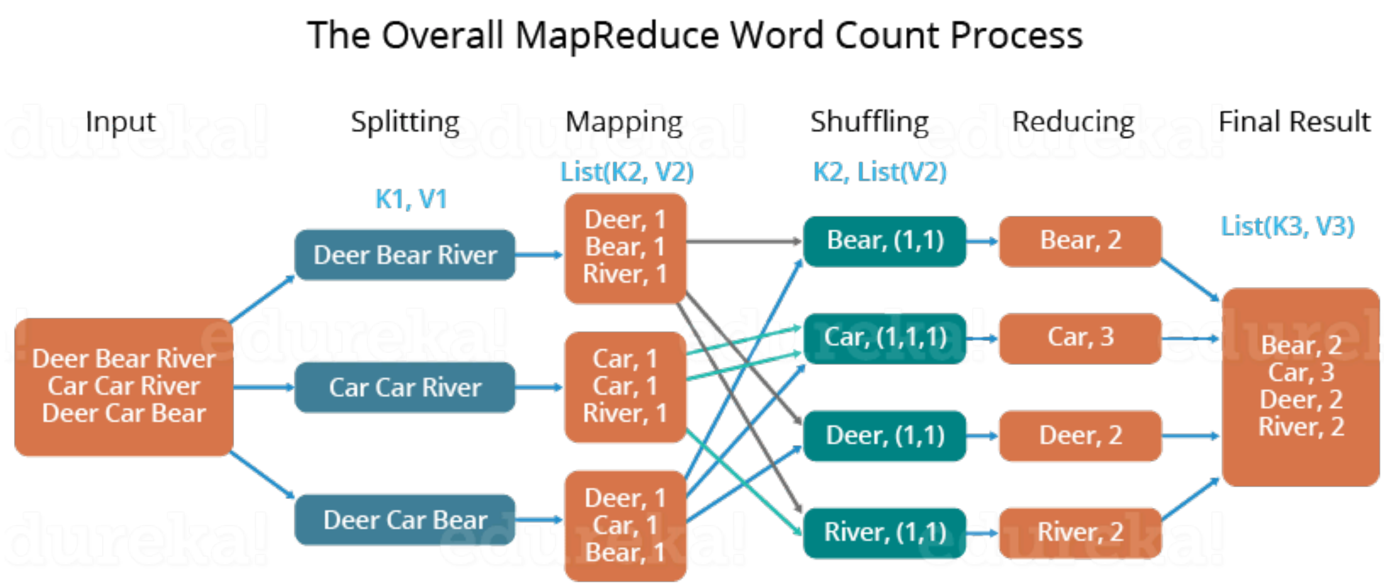
- Workflow (step 6):

  The reducer traverses the sorted key-value pairs, merges the key-value pairs with the same key value, and stores the statistical results as output files in the corresponding partition.



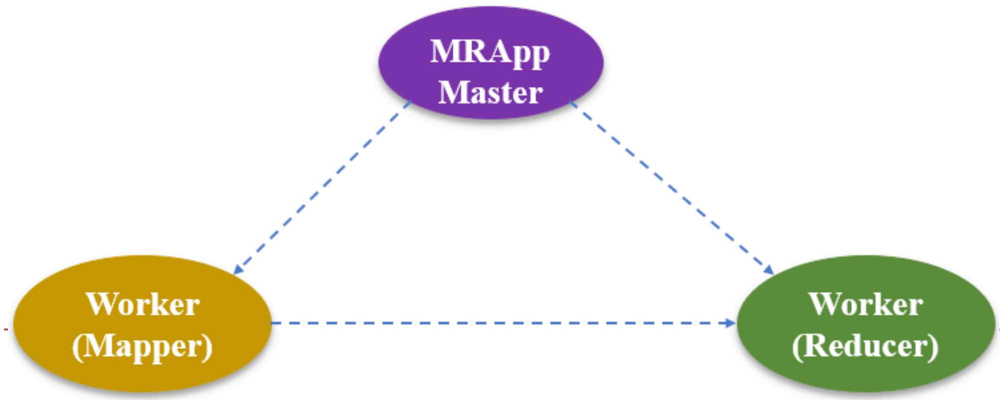
How the MapReduce works

- The entire workflow of MapReduce can be divided into 5 parts:
  - Input, splitting, mapping, reducing and final result



Fault-Tolerance in MapReduce

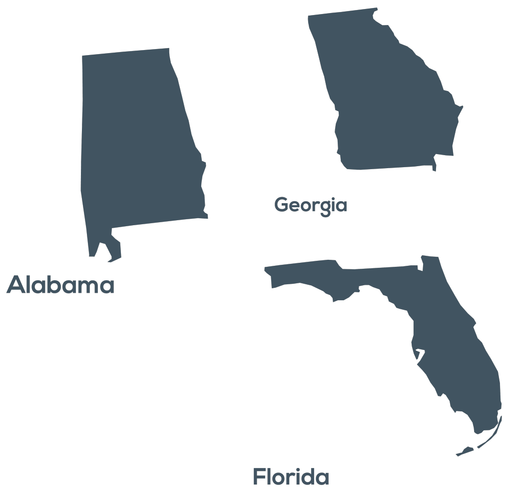
- Master fails
  - Switch to secondary master
  - Restart entire job
- Worker fails
  - If running map: restart map tasks on available/free worker nodes
  - If running reduce: restart reducer tasks



MapReduce Example

- We want to know the top 3 selling smartphones in Alabama, Georgia and Florida area

AL	GA	FL
LG	Sumsang	LG
Sumsang	apple	apple
apple	LG	Sumsang
…	…	…



MapReduce Example

- Splitting: divide the task into 3 sub-tasks

AL
LG
sumsang
…

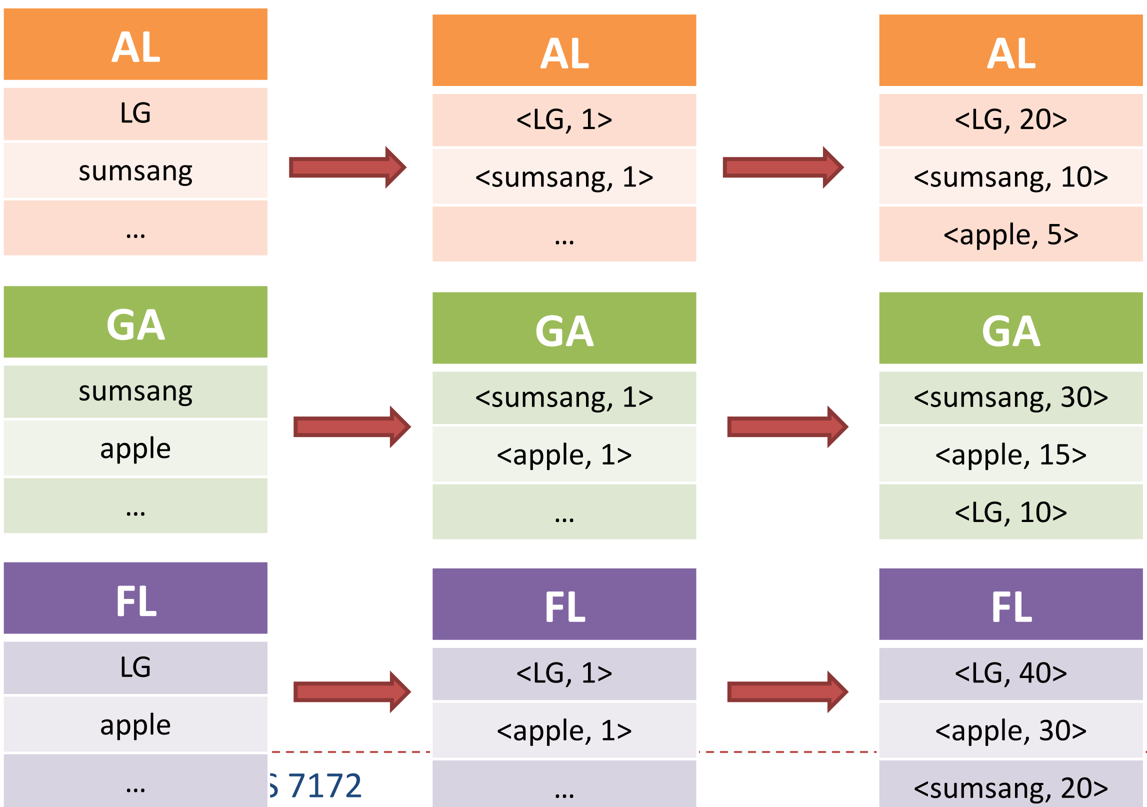
GA
sumsang
apple
…

FL
LG
apple
…

AL	GA	FL
LG	Sumsang	LG
Sumsang	apple	apple
apple	LG	Sumsang
…	…	…

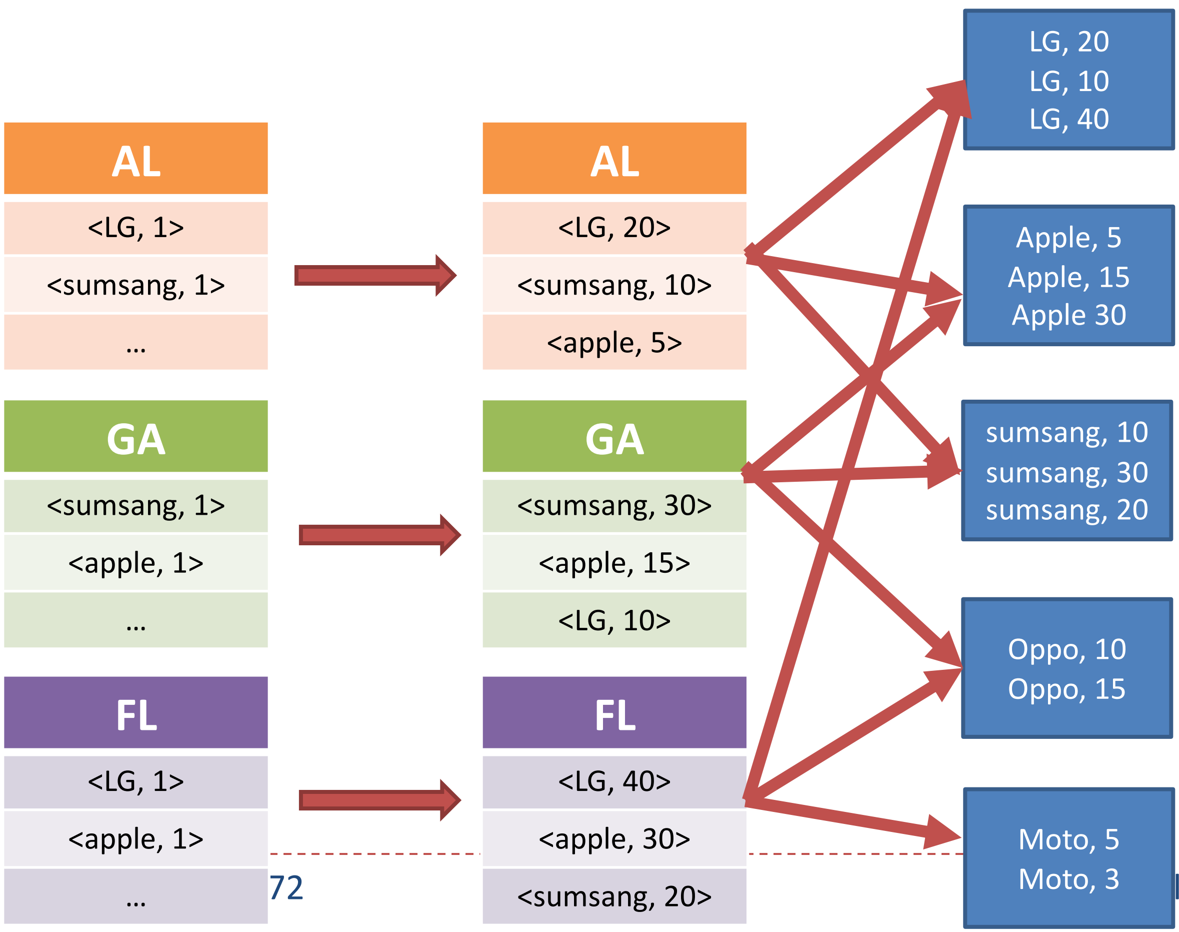
MapReduce Example

- Mapping: keep calling map() function and get the selling statistics of each smartphone brand. Here the key is the smartphone brand and value is the number



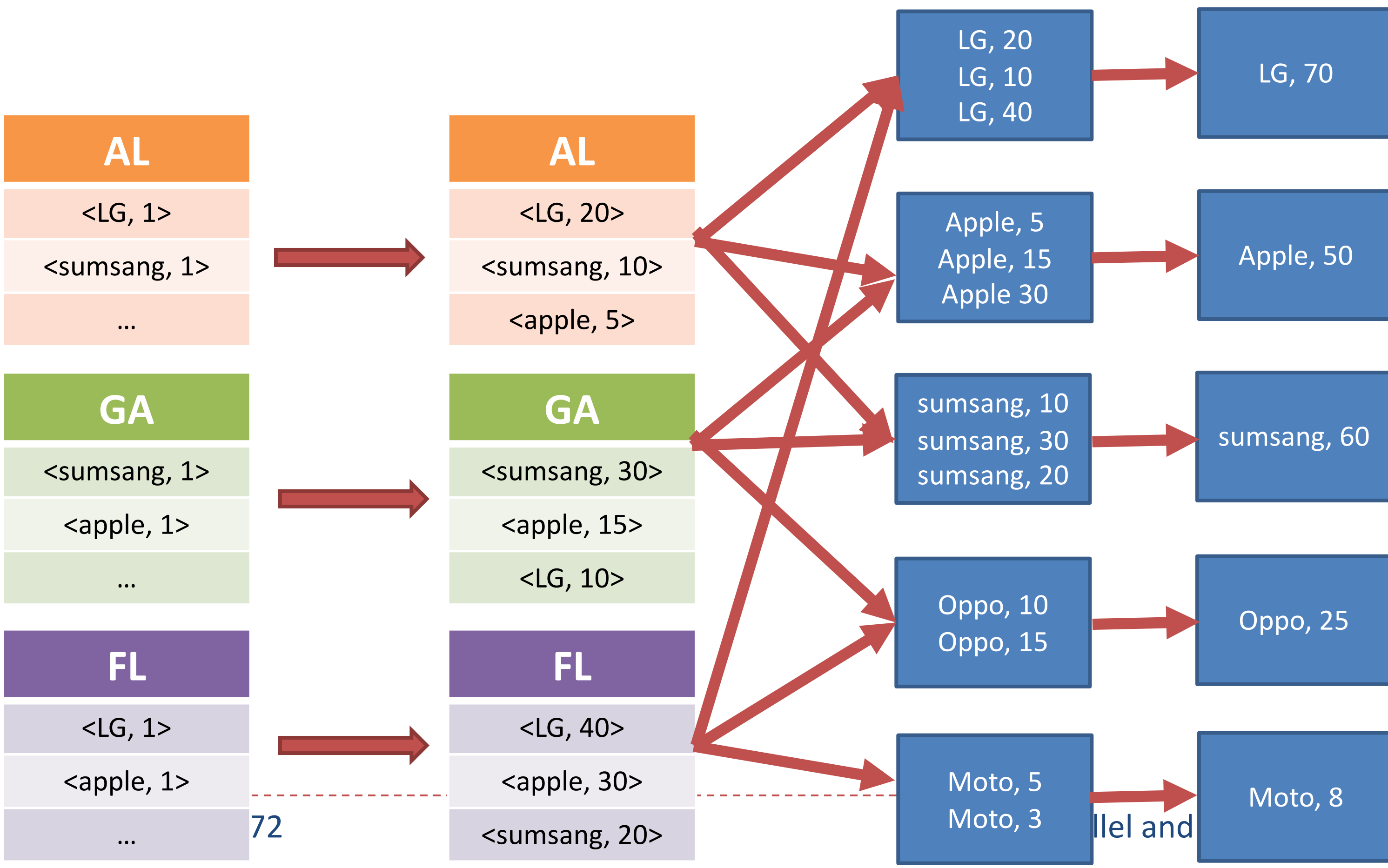
MapReduce Example

- Shuffuling: read the results of the Mapping phase and divide the different results into different areas



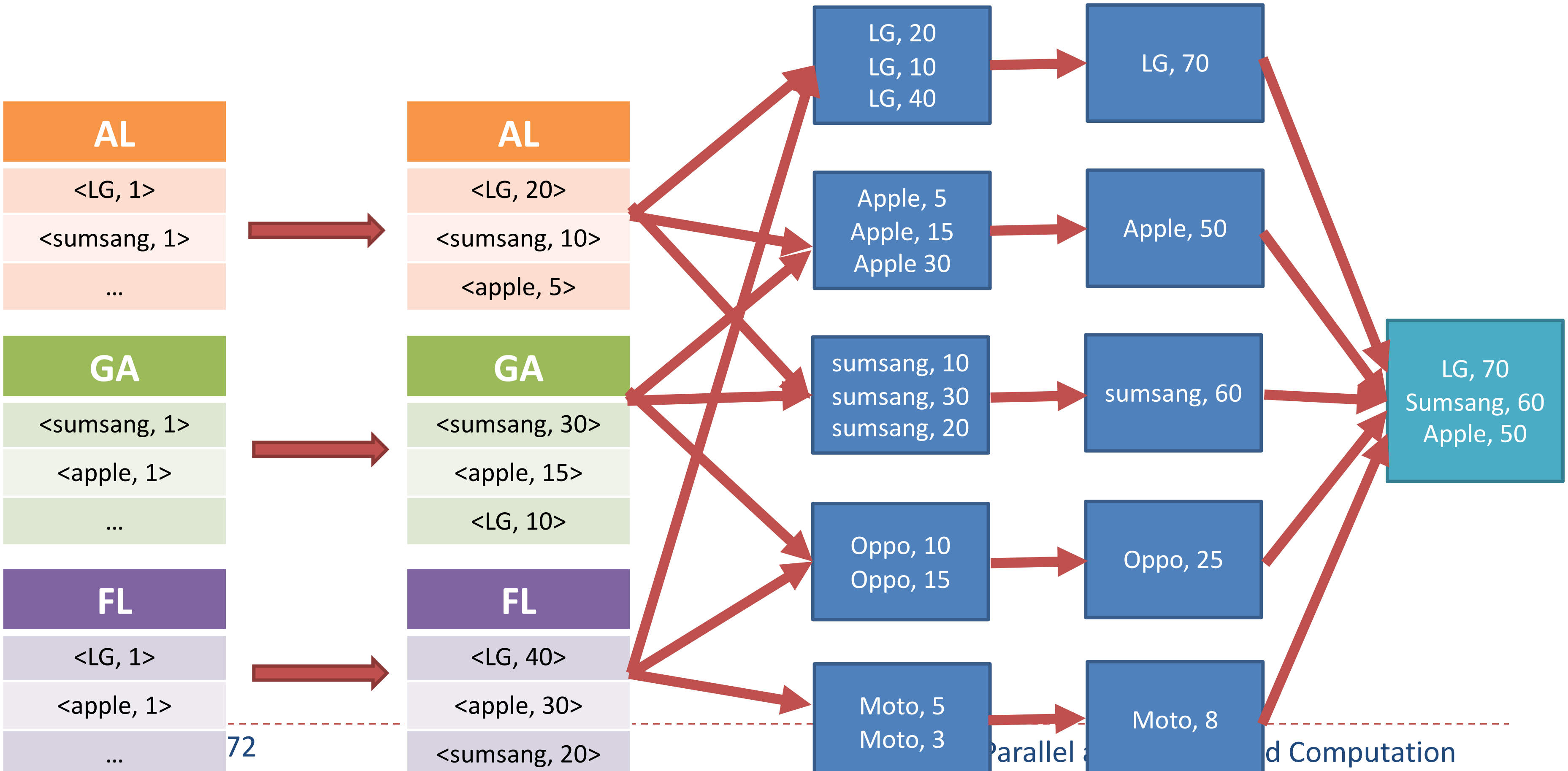
MapReduce Example

- Reducing: merge the results of the same brand



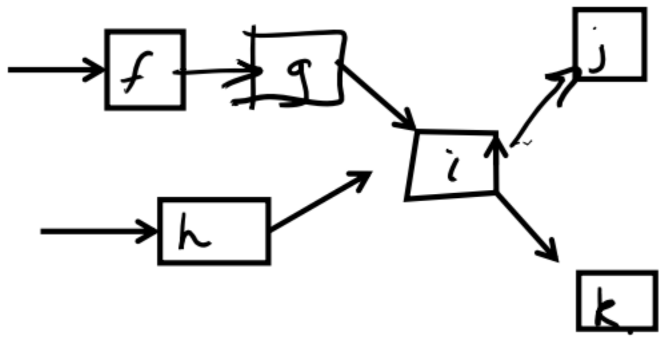
MapReduce Example

- Final results: get the top 3 selling smartphones in Alabama, Georgia and Florida area



MapReduce Limitations

- Static data
  - Restrictive programming model
  - No support for dataflows (not dynamic)
  - Purely batch processing (not flow)
  - Jobs can take hours to complete
  - No streaming, interactive analytics



- How many mappers and reducers?
- How much memory to allocate?

When NOT To Use MapReduce

- Modern computing hardware has plenty of computing power:
  - A typical laptop: 8 cores, 16 GB RAM, 512 GB SSD (usually NVMe)
  - A typical server: 64 cores, 256 GB RAM, 2 TB SSD, …
- Is your dataset really that large?
  - Wikipedia: 30 GB
- In many cases, an optimized non-distributed implementation beats a large cluster (!)

Conclusion

- Single server vs multiple server processing
- Divide-and-conquer
- MapReduce
  - Map
  - Reduce
  - How it works
  - Fault-tolerance
  - Limitation