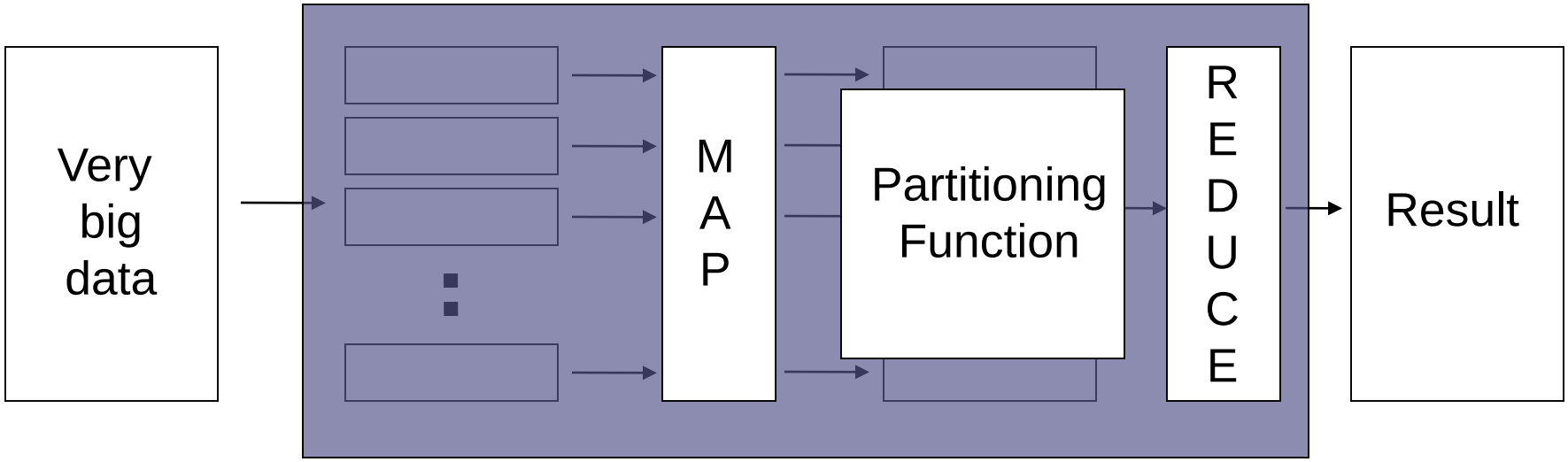
Take a Close Look at MapReduce

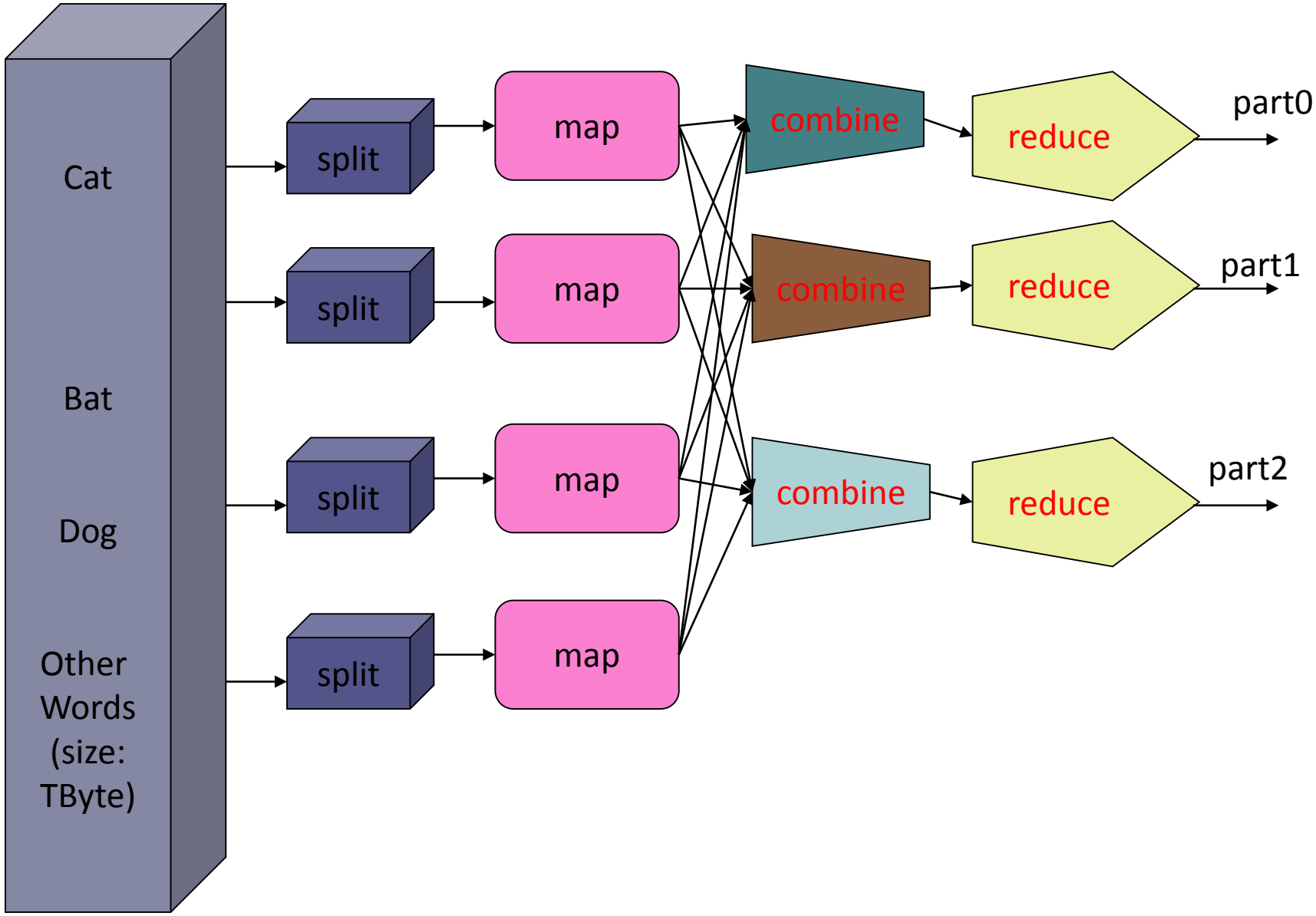
What is MapReduce

- Origin from Google
- 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

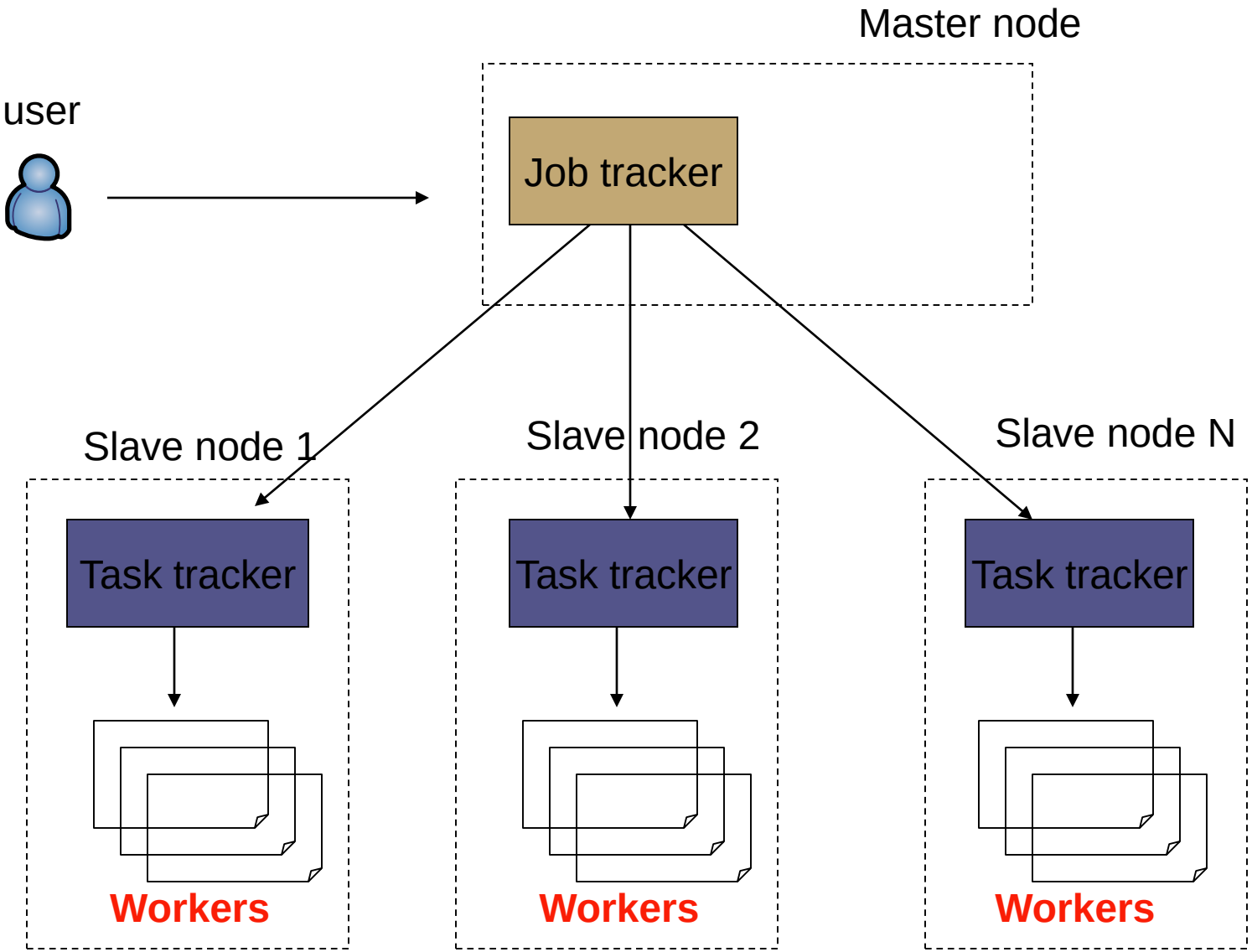
Map+Reduce



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


Cat
Bat
Dog
Other Words (size: TByte)
split
split
split
split
map
map
map
map
combine
combine
combine
reduce
reduce
reduce
part0
part1
part2

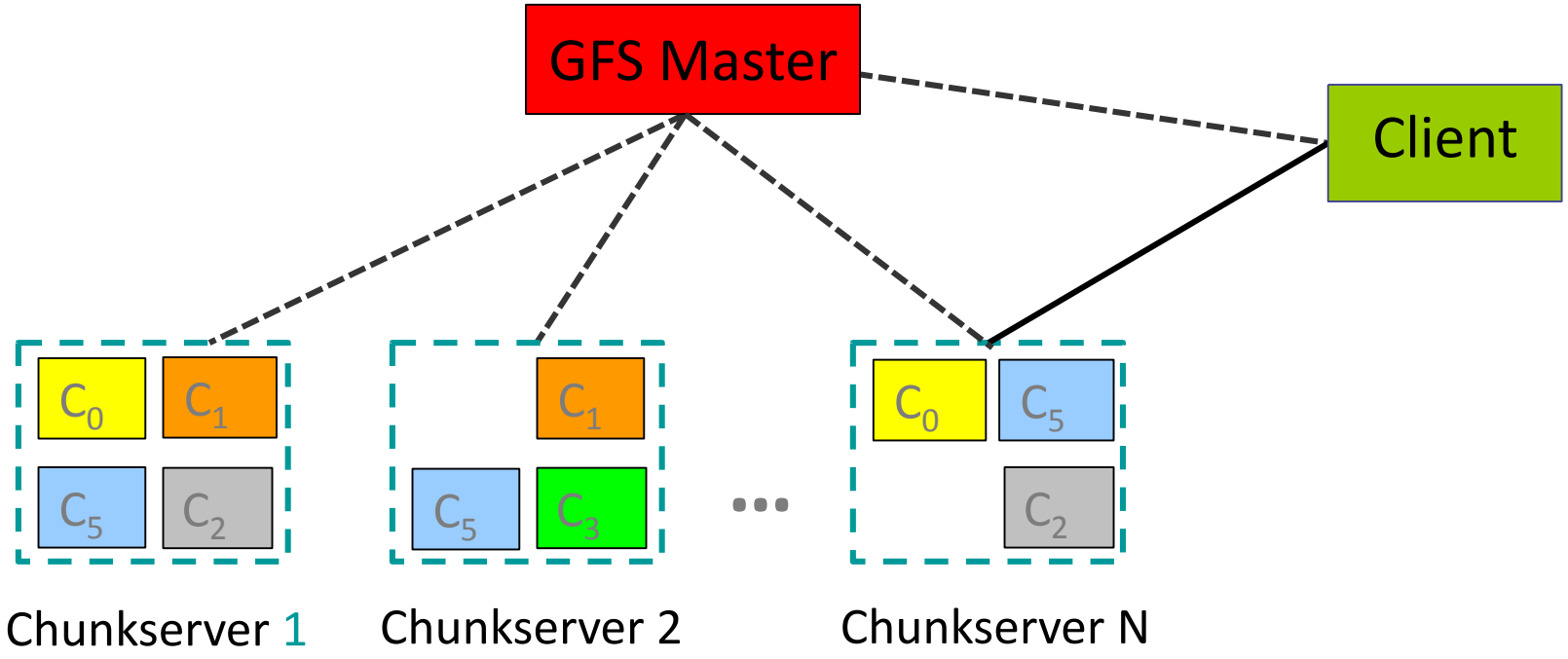
Architecture overview



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

A Simple Example

- Counting words in a large set of documents

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

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

Mapper

- Reads in
- Outputs a pair
  - Let's count number of each word in user queries (or Tweets/Blogs)
  - The input to the mapper will be <queryID, QueryText>:

    <Q1,"The teacher went to the store. The store was closed; the store opens in the morning. The store opens at 9am." >
  - The output would be:

    <The, 1> <teacher, 1> <went, 1> <to, 1> <the, 1> <store,1> <the, 1> <store, 1> <was, 1> <closed, 1> <the, 1> <store,1> <opens, 1> <in, 1> <the, 1> <morning, 1> <the 1> <store, 1> <opens, 1> <at, 1> <9am, 1>

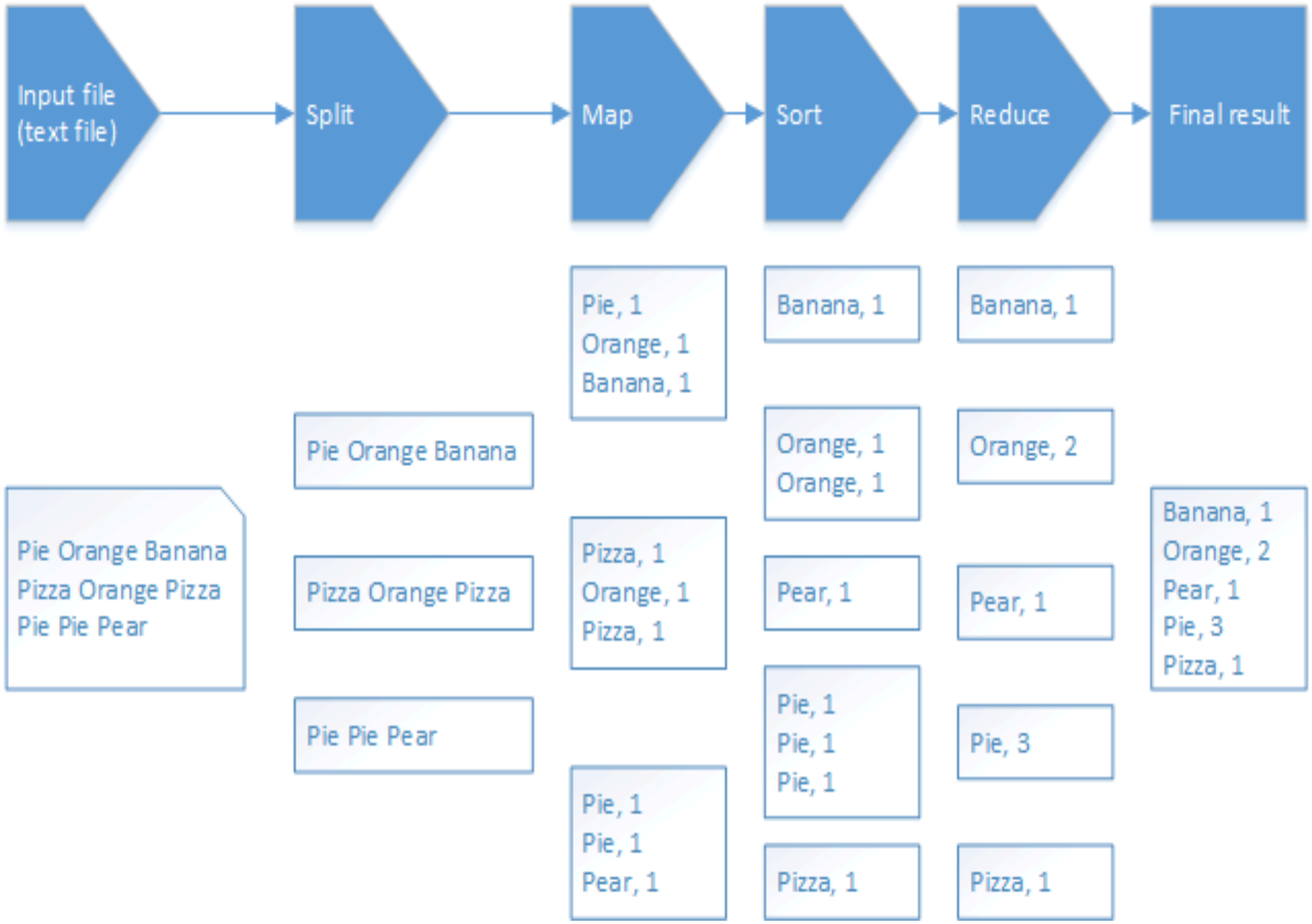
Reducer

- Accepts the Mapper output, and aggregates values on the key
  - For our example, the reducer input would be:

    <The, 1> <teacher, 1> <went, 1> <to, 1> <the, 1> <**store**, 1> <the, 1> <store, 1> <was, 1> <closed, 1> <the, 1> <**store**, 1> <opens,1> <in, 1> <the, 1> <morning, 1> <the 1> <**store**, 1> <opens, 1> <at, 1> <9am, 1>

  - The output would be:

    <The, 6> <teacher, 1> <went, 1> <to, 1> **<store, 3>** <was, 1> <closed, 1> <opens, 1> <morning, 1> <at, 1> <9am, 1>


Input file (text file)
Split
Map
Sort
Reduce
Final result
Pie Orange Banana
Pizza Orange Pizza
Pie Pie Pear
Pie Orange Banana
Pizza Orange Pizza
Pie Pie Pear
Pie, 1
Orange, 1
Banana, 1
Pizza, 1
Orange, 1
Pizza, 1
Pie, 1
Pie, 1
Pear, 1
Banana, 1
Orange, 1
Orange, 1
Pear, 1
Pie, 1
Pie, 1
Pie, 1
Pizza, 1
Banana, 1
Orange, 2
Pear, 1
Pie, 3
Pizza, 1
Banana, 1
Orange, 2
Pear, 1
Pie, 3
Pizza, 1

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

Points to be emphasized

- No *reduce* can begin until *map* is complete
- Master must communicate locations of intermediate files
- Tasks scheduled based on location of data
- If *map* worker fails any time before *reduce* finishes, task must be completely rerun.

How to use it

- User to do list:
  - indicate:
    - Input/output files
    - **M**: number of map tasks
    - **R**: number of reduce tasks
    - **W**: number of machines
  - Write *map* and *reduce* functions
  - Submit the job

Applications

- String Match, such as Grep
- Reverse index
- Count URL access frequency
- Lots of examples in data mining

Conclusion

- Provide a general-purpose model to simplify large-scale computation
- Allow users to focus on the problem without worrying about details