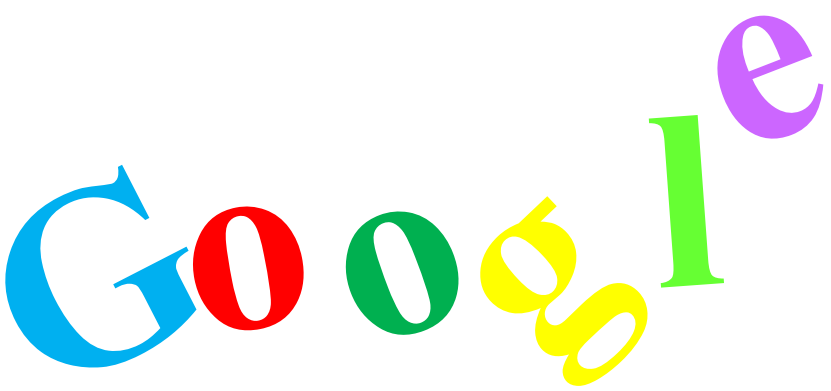
JPU | جامعة الجزيرة الخاصة
ALJAZEERA PRIVATE UNIVERSITY



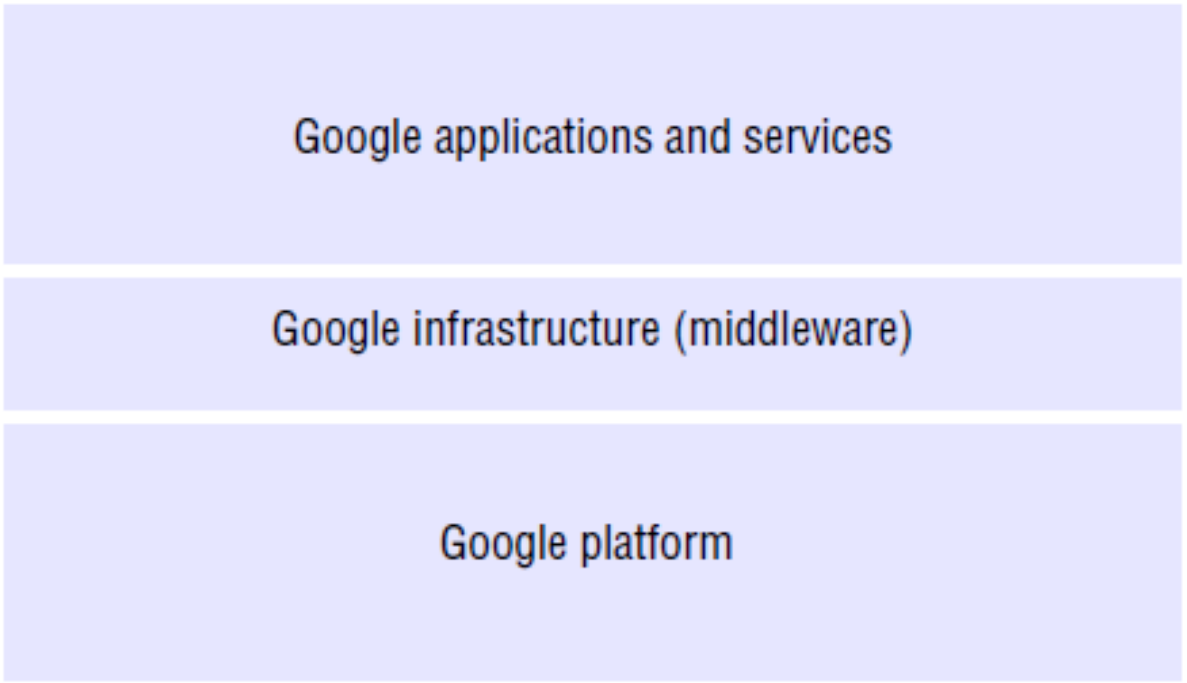
Google

- Google is an open scalable, reliable, high-performing DS.
- Google organizes the world's information and make it universally accessible and useful.
- Google = Search Engine + Gmail + Google Maps, Talk, Docs, Earth, …….
- Google Search Engine = Crawling + Indexing (*inverted index mapping + links + font + position*) + Ranking
- Google = cloud provider as (Software as a service & Platform as a service)
- Google principles :
  - Simplicity : software should do one thing and do it well.
  - Performance : every millisecond counts.
  - Testing : logging and tracing to resolve faults in Sys.

Google design' characteristics

- **Scalable Sys** : more data / more queries <> better results
- **Reliable Sys** : high-available
- **Performing Sys** : crawling, indexing, sorting
- **Open Sys** : middleware & services to support search / cloud computing / new applications

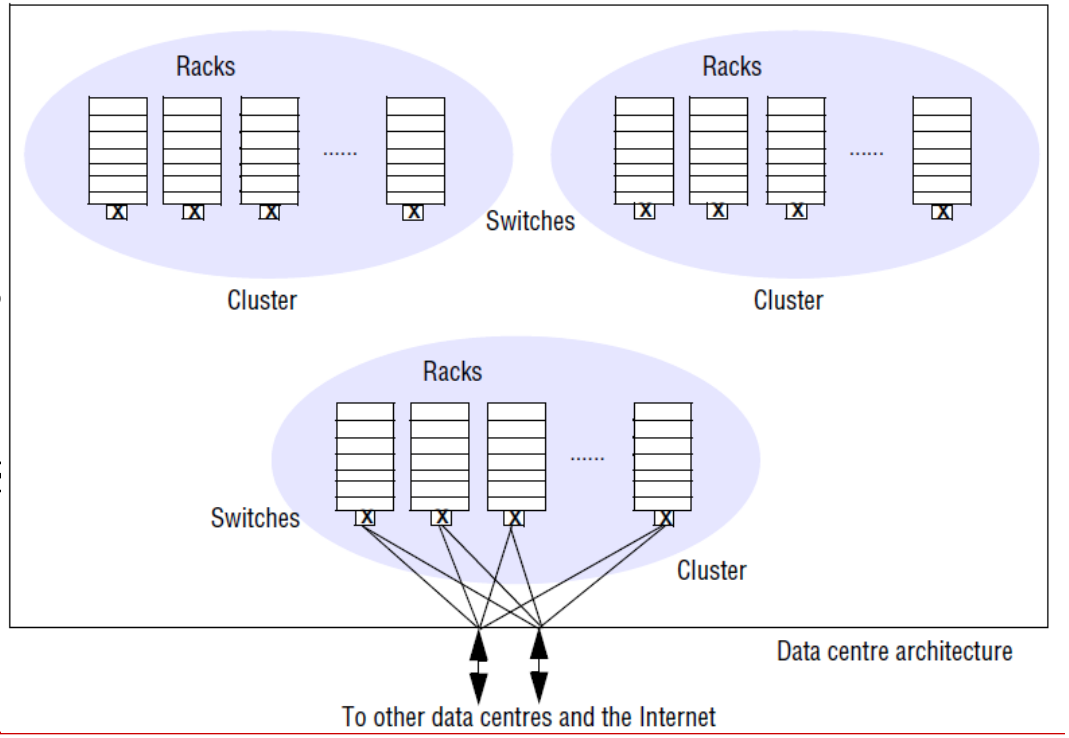
Google Architecture



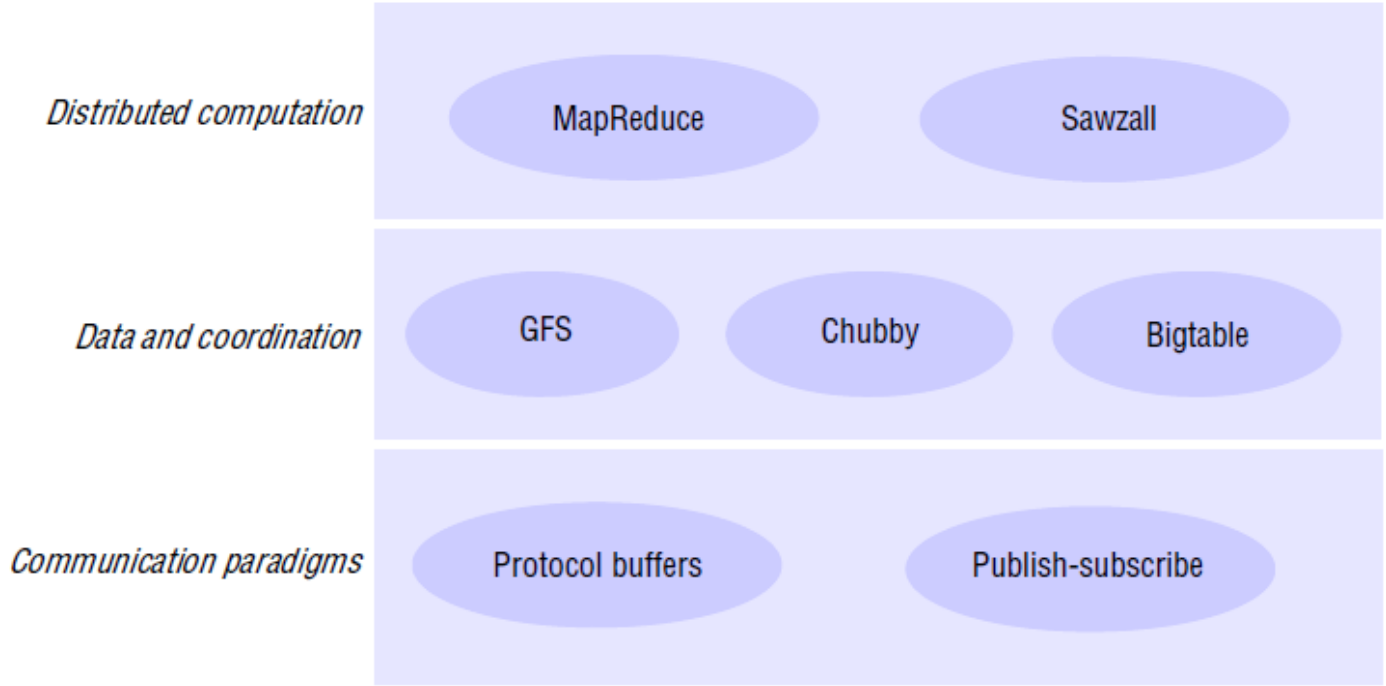
Google physical model

- **Physical model** = very large numbers of PCs to produce a cost-effective environment for distributed storage and computation.

- Data centers
- Clusters = 30 racks
- Racks = 40-80 PCs + Ethernet switches



Google Middleware

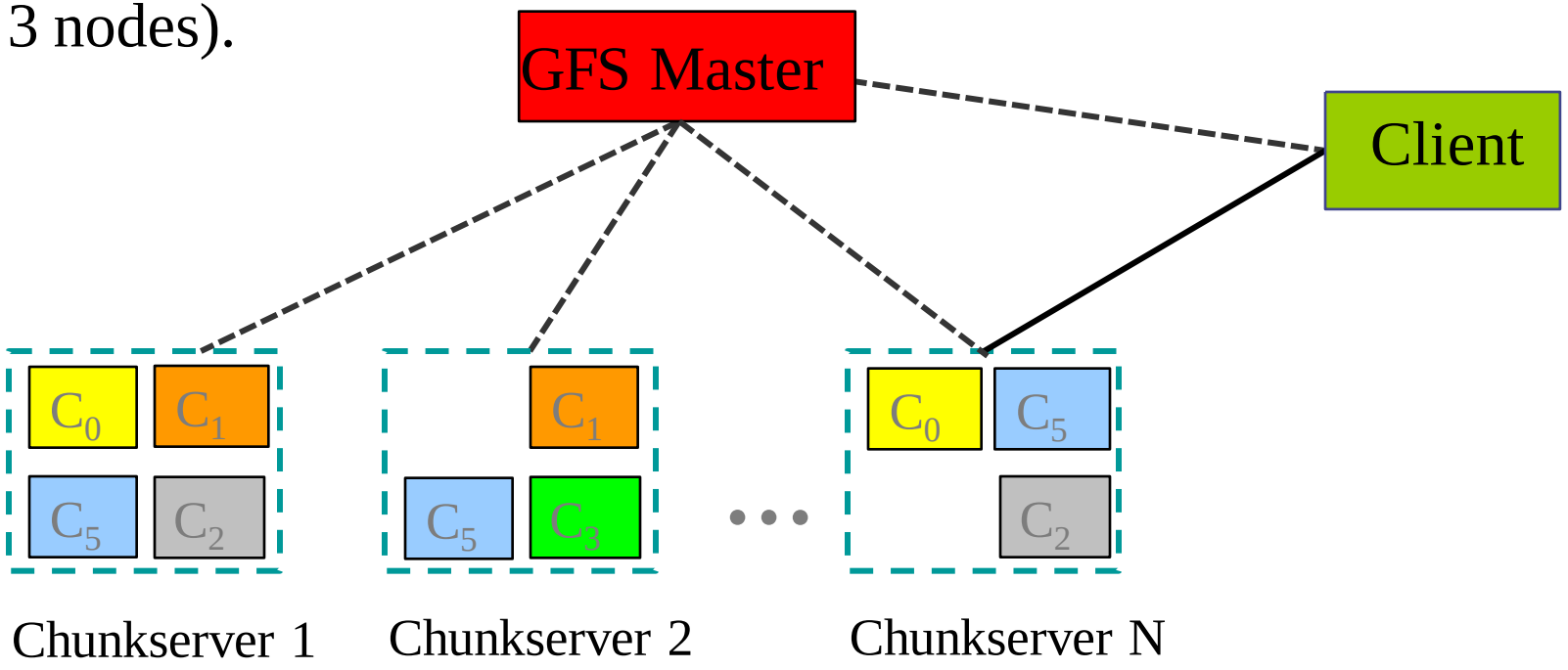


GFS
(Google File System)

- GFS = Distributed File System
- Requirements :
  - To be **reliable** over software/hardware components' failures
  - Files' size is huge ≈ 100 Peta bytes
  - Files' number ≈ 100 Mega files
  - Files' Access ≈ (at most) Sequential reads
  - To be **open**, it must support the development of new applications, by providing services like :
    - Create / open / delete / read / write / close + snapshot / append / record
  - To be **scalable**. So, it must support concurrent updates

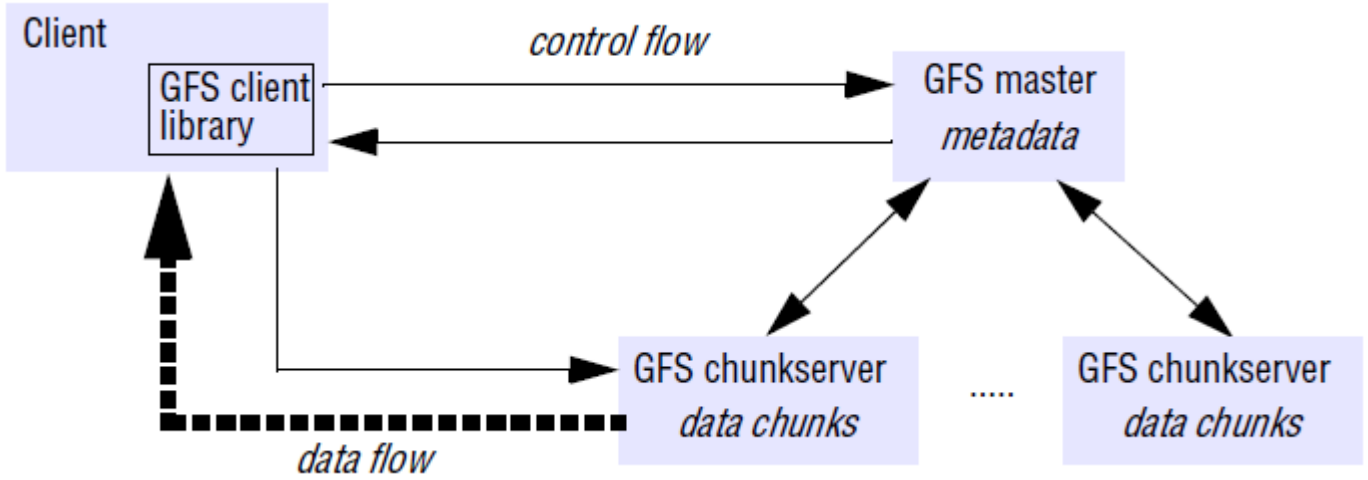
Reliable GFS

- **GFS** is tolerant with component failure.
- Files is divided by chunks of 64 mega-bytes (replicated on 3 nodes).



Optimized GFS

sequential reads/appends through large files



- One master (global view of file sys) + hundreds chunk servers
- Master = manages file namespaces + access control
  - + mapping file to chunk set + (replication 1 chunk to 3 chunk servers) for reliability

GFS

- Results :
  - Minimal involvements of master
  - Large chunk size → less meta-data (in main memory)
  - Separation of control/data flows → performing master
- Consistency Approach :
  - When master recognize an update. Master choose one primary chunk server & it returns to client the primary and secondary IDs.
  - The client sends updates to all replicas (chunk servers).
  - The primary determines the update order to all replicas.
  - When secondary replicas acknowledge to the primary.
  - Primary sends its response to the client.

Chubby

- Chubby = provide storage & coordination services :
  - Locks to synchronize distributed activities.
  - Election service to choose the primary chunk server (Ex).
  - Name service within Google
  - Reliable storage for small files
  - File Event subscribing mechanism (notify by callback)

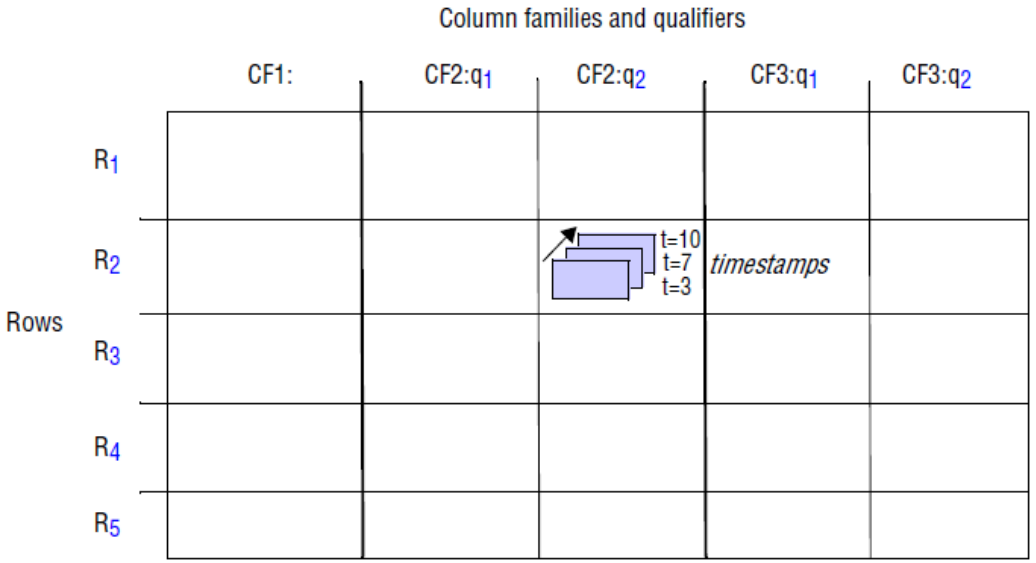
BigTable

- **BigTable** = Scalable, High Available and performing Sys for storing data (≈ peta-bytes) on thousands machines.
- BigTable uses GFS for data storage.
- BigTable uses Chubby for distributed coordination (monitoring + load balancing).
- Used by 60 projects like

  Google Maps,

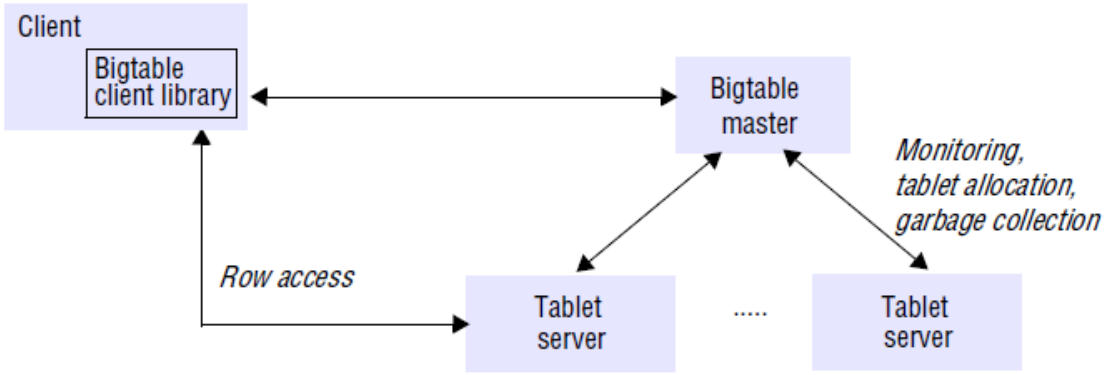
  Google Analytics,

  …..



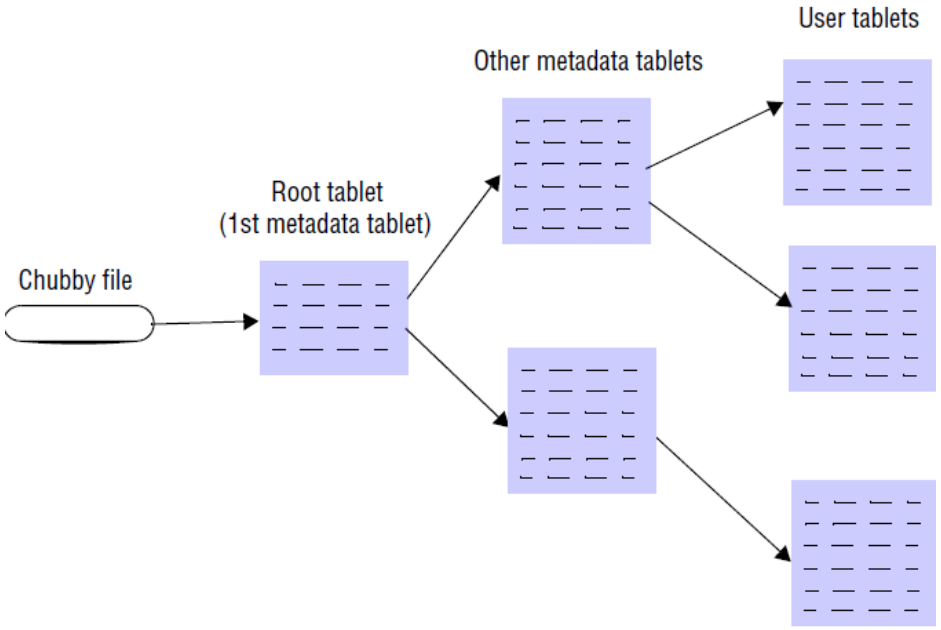
BigTable Architecture

- BigTable is like GFS: one master + several Tablet servers
  - One master = centralized view of sys state → better load balancing decision (Monitoring the tablet servers' state)
  - → separation between control & data
  - A table is divided by rows on tablet servers (as sstable files on GFS)



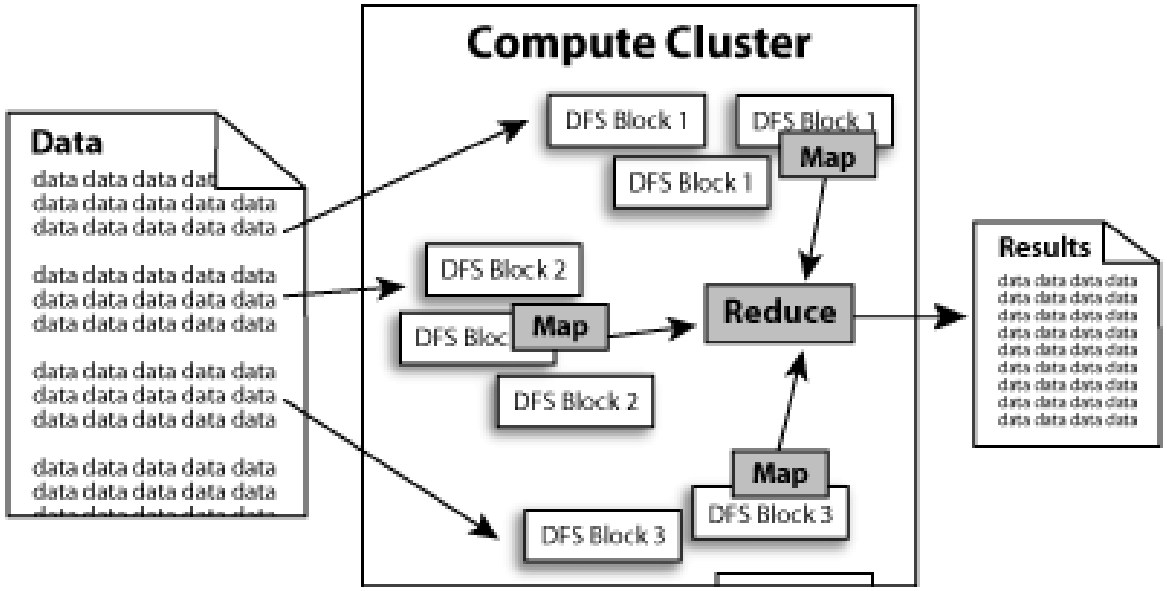
BigTable & GFS & Chubby

- BigTable client uses chubby file (root tablet) to find the tablet location
- Tablet server uses Chubby lock service & GFS to provide its services

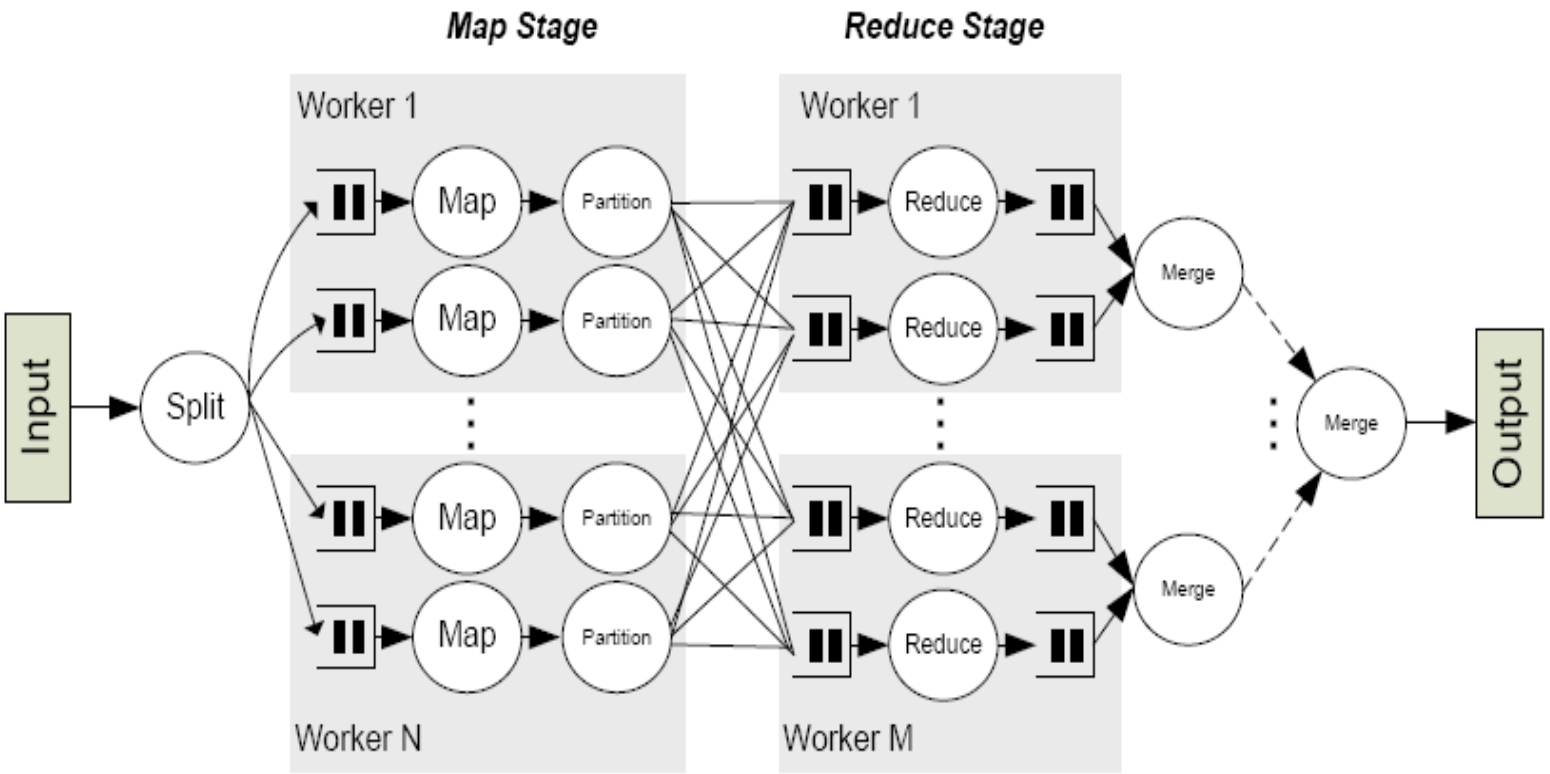


MapReduce

- **Why?** Very large flat data sets (in + RDB over + disks & machines) & regular structure (phone call recs, network logs, web doc repos).



MapReduce

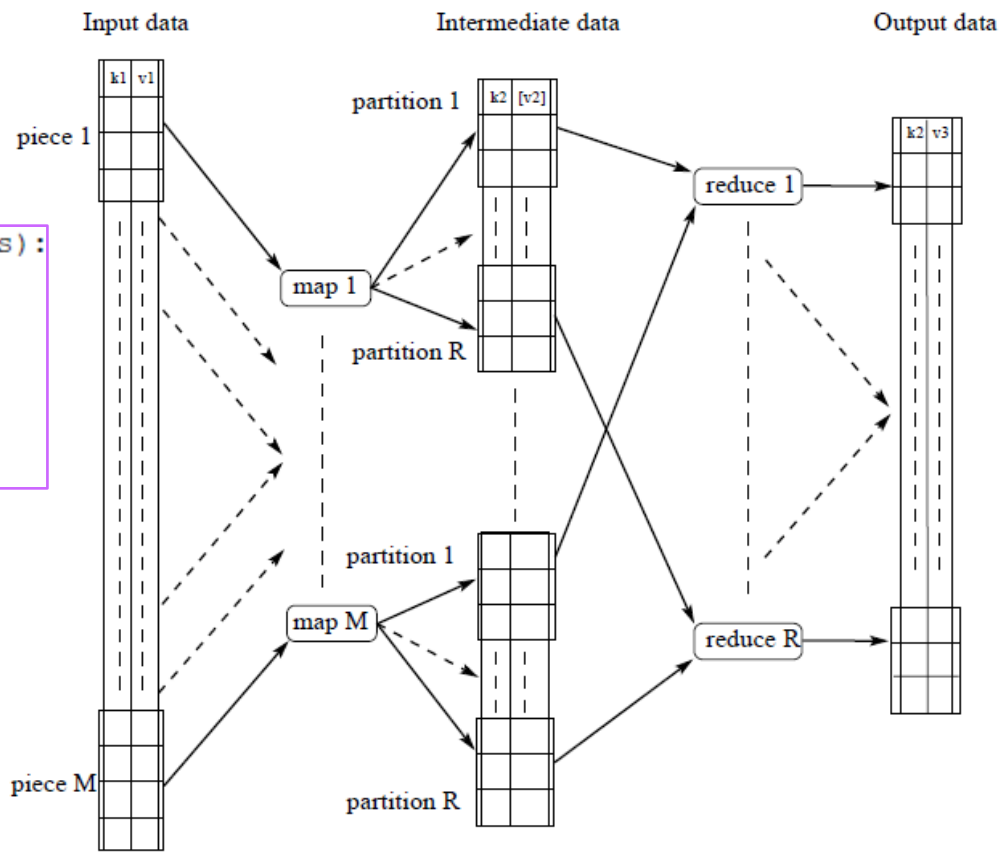

Map Stage
Reduce Stage
Worker 1
Worker 1
Input
Split
Map
Partition
Map
Partition
Map
Partition
Map
Partition
Reduce
Reduce
Reduce
Reduce
Merge
Merge
Merge
Output
Worker N
Worker M

MapReduce

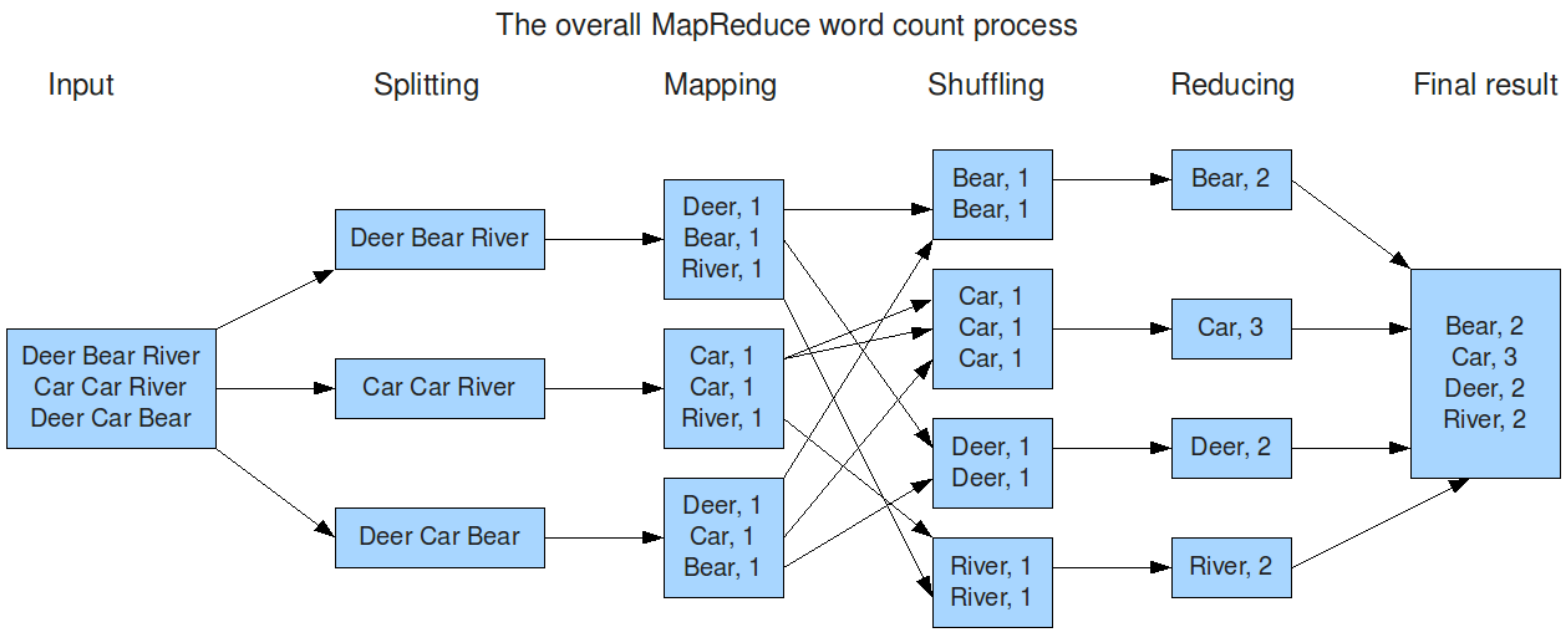
(Ex : word counting)

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

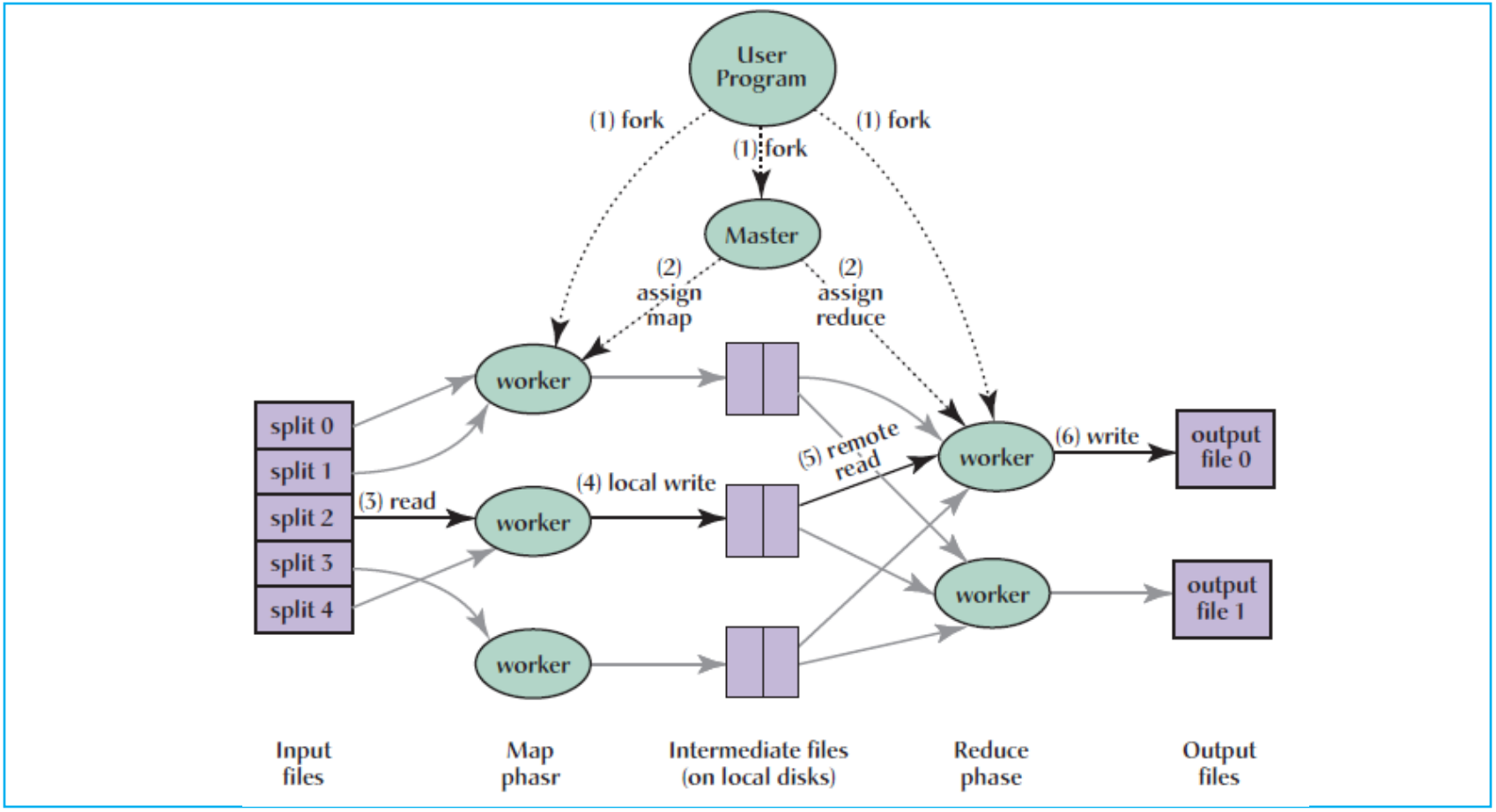
```
reduce(String key, Iterator values):
 // key: a word
 // values: a list of counts
 int result = 0;
 for each v in values:
  result += ParseInt(v);
 Emit(AsString(result));
```



MapReduce
(Ex : word counting)



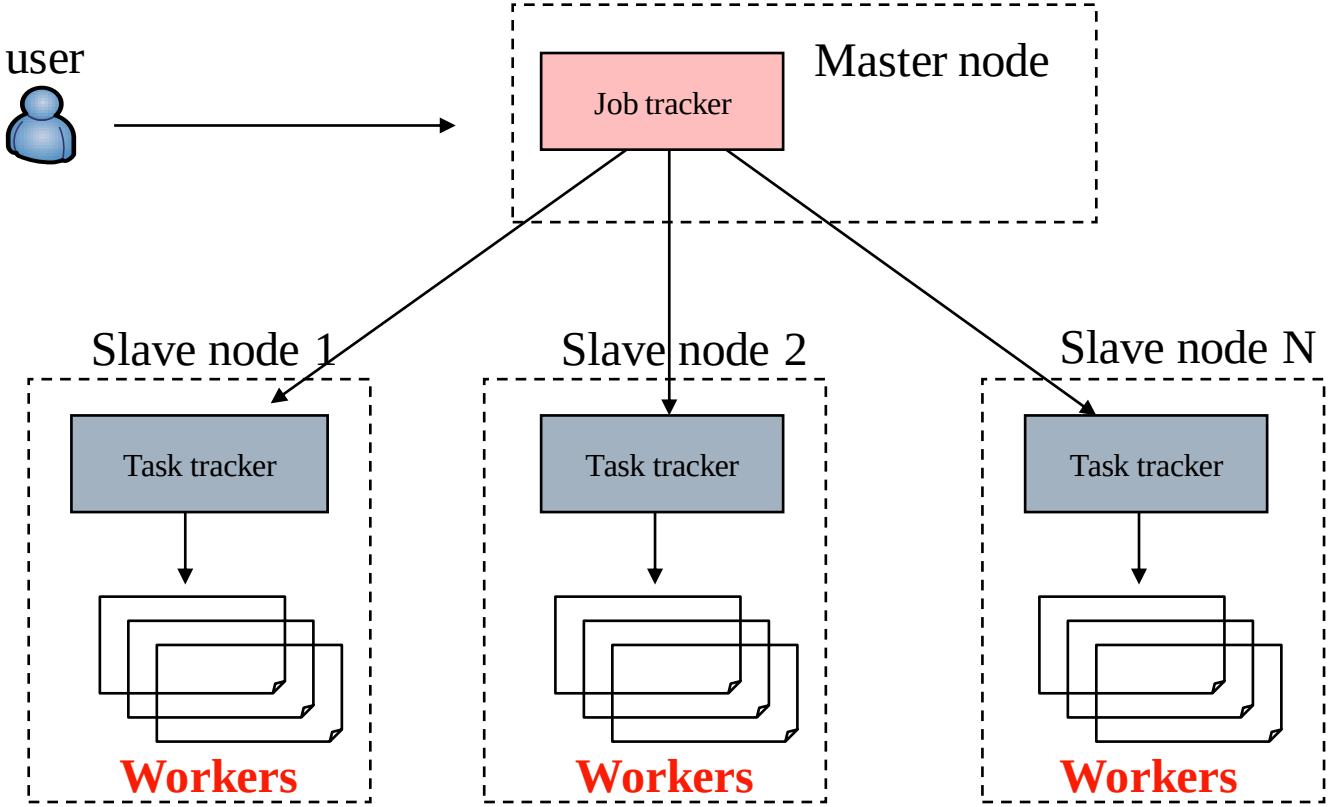
MapReduce (& Management)



MapReduce

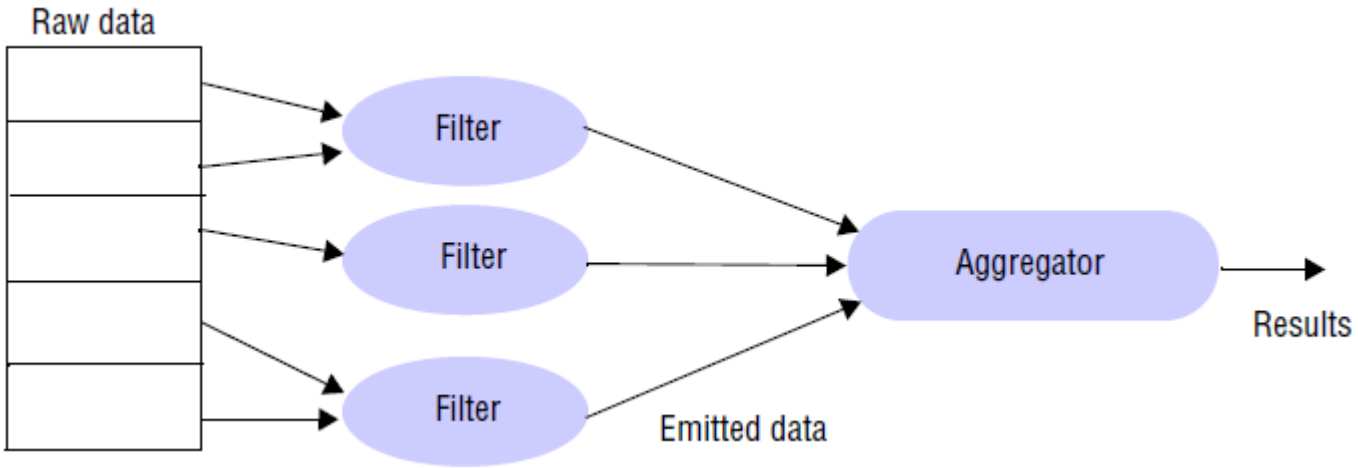
(Fault Tolerance & Load Balancing & ….)

- Bing/load messages to workers



Sawzall

- **Sawzall** = Google's MapReduce programming model
- Filters = Map tasks = commutative operations.
- Aggregators = Reduce tasks = associative operations.



Questions ?