# Google Bigtable

## (Bigtable: A Distributed Storage System for Structured Data)

Komadinovic Vanja, Vast Platform team

# Presentation overview

- introduction
- design
- basic implementation
- GFS - HDFS introduction
- MapReduce introduction
- implementation
- HBase - Apache Bigtable solution
- performances and usage case
- some thoughts for discussion

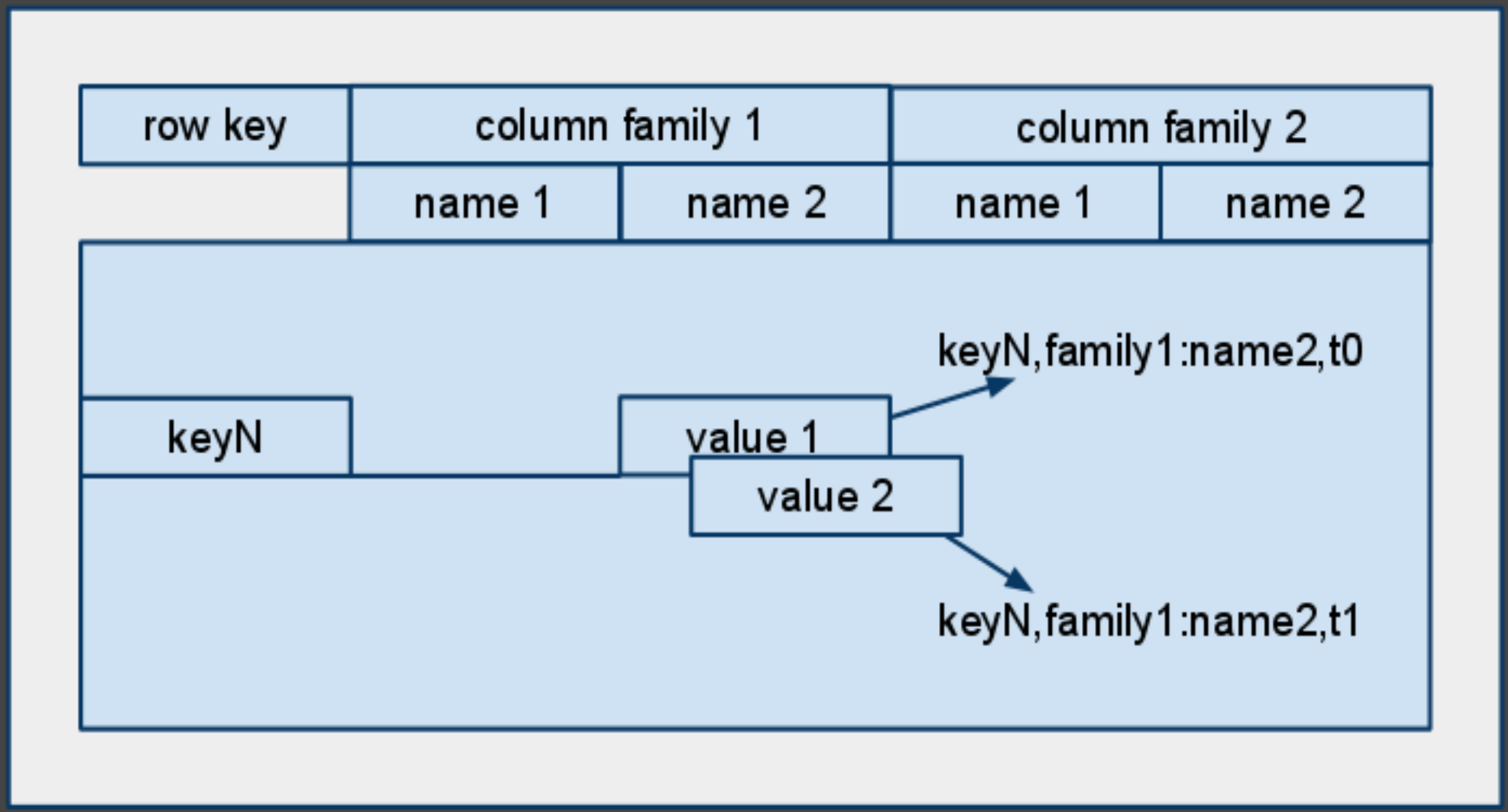
# Introduction

- what is bigtable?
- why need for something more than SQL?
- machines number increase - scaling
- fault tolerance
- row, column, cell
- versioning
- storage on distributed file system
- read, write operations on row or set of rows
- scanner - iterator
- no need for indexing

# Design

- column family basic unit of access control
- column family must be created before data insertion
- column labels number is unlimited
- row key is arbitrary string, max. size 64Kb
- every cell has copies in time - configurable

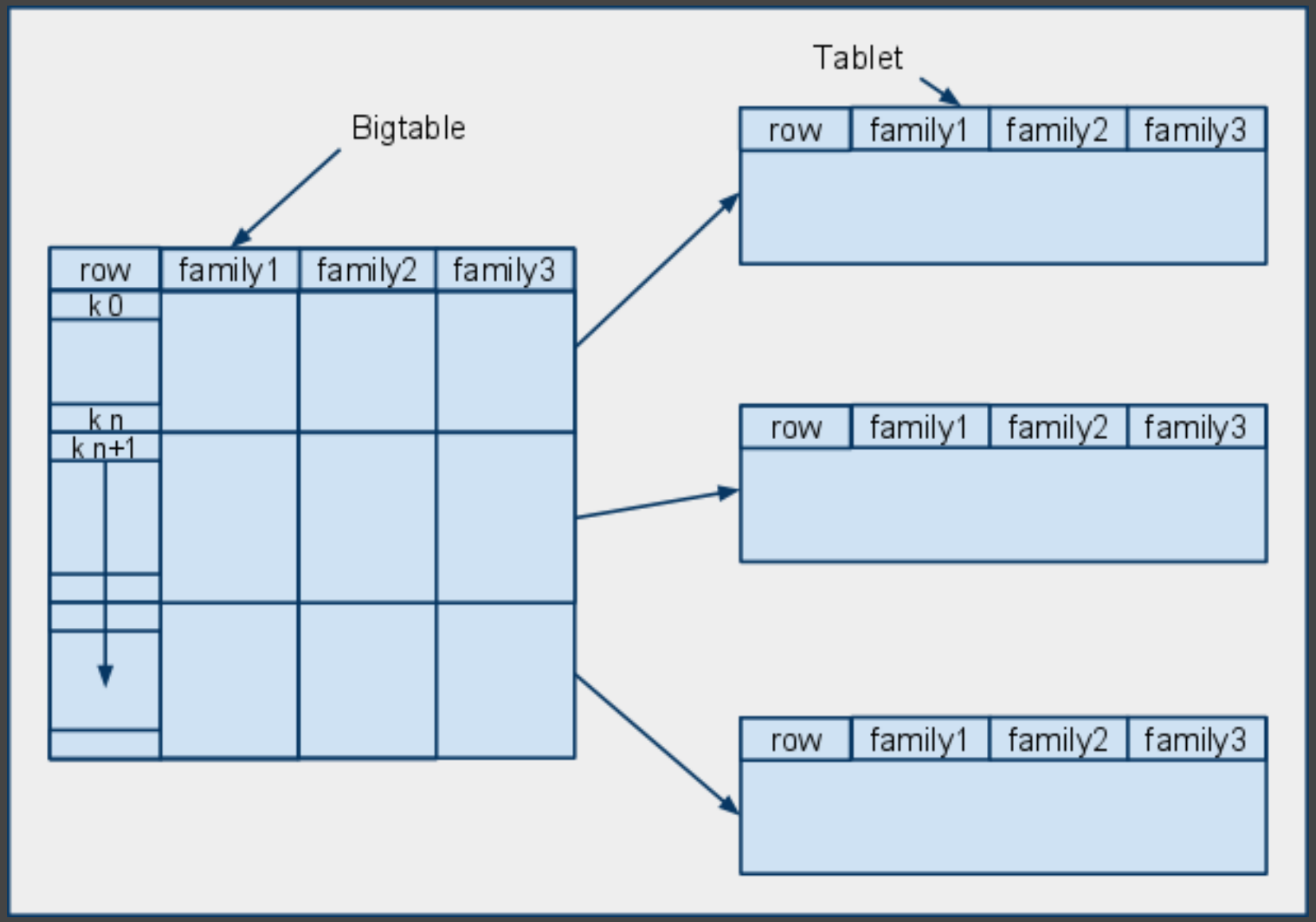
# Design


row key
column family 1
column family 2
name 1
name 2
name 1
name 2
keyN,family1:name2,t0
keyN
value 1
value 2
keyN,family1:name2,t1

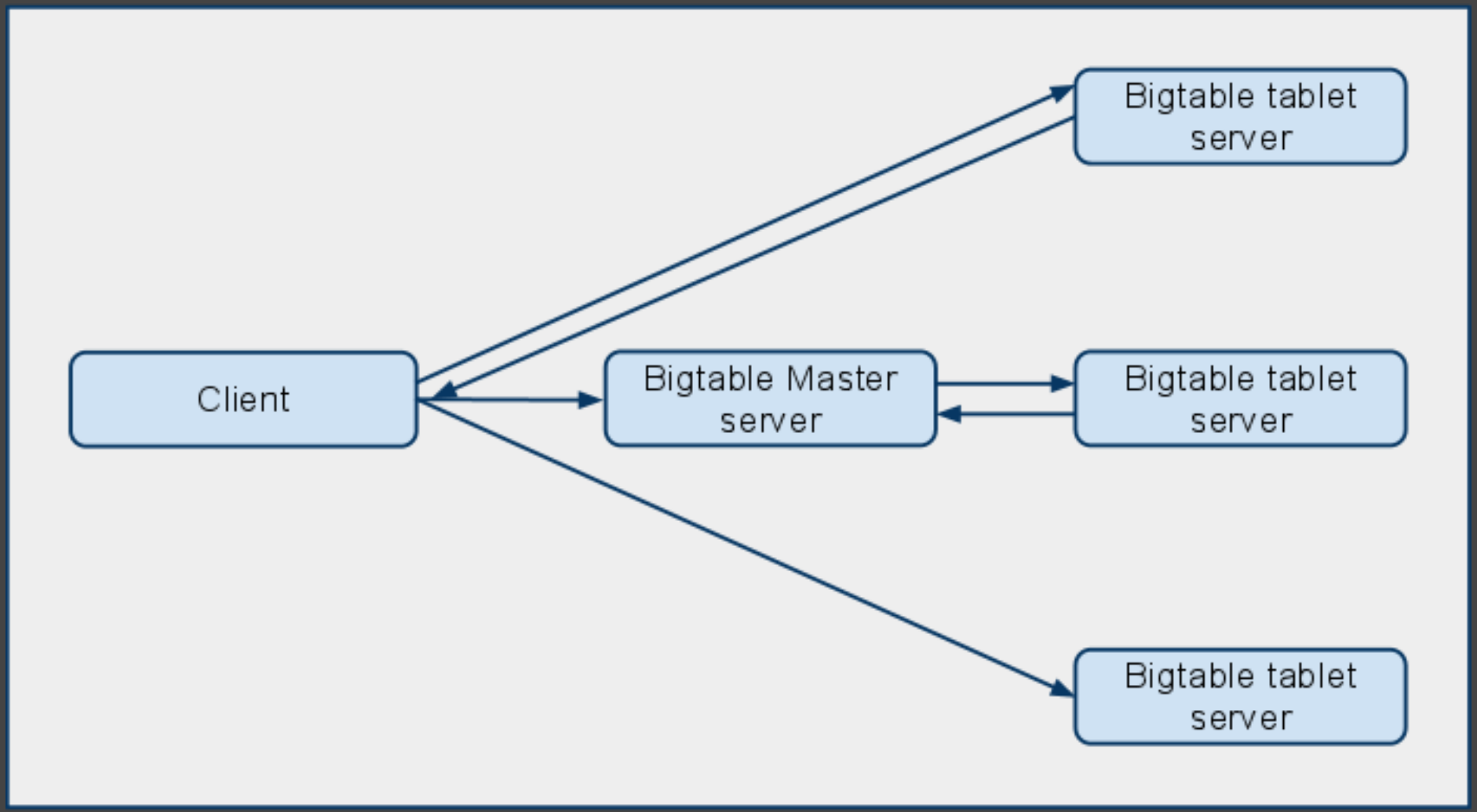
# Design

- table is divided in tablets
- tablets are distributed on machines
- each tablet contains N rows
- tablet size 100 - 200 Mb
- tablet server splits tablet when it grows, only operation initiated by tablet server
- tablets are stored on GFS

# Design


Tablet
Bigtable
row
family1
family2
family3
k 0
k n
k n+1
row
family1
family2
family3
row
family1
family2
family3
row
family1
family2
family3

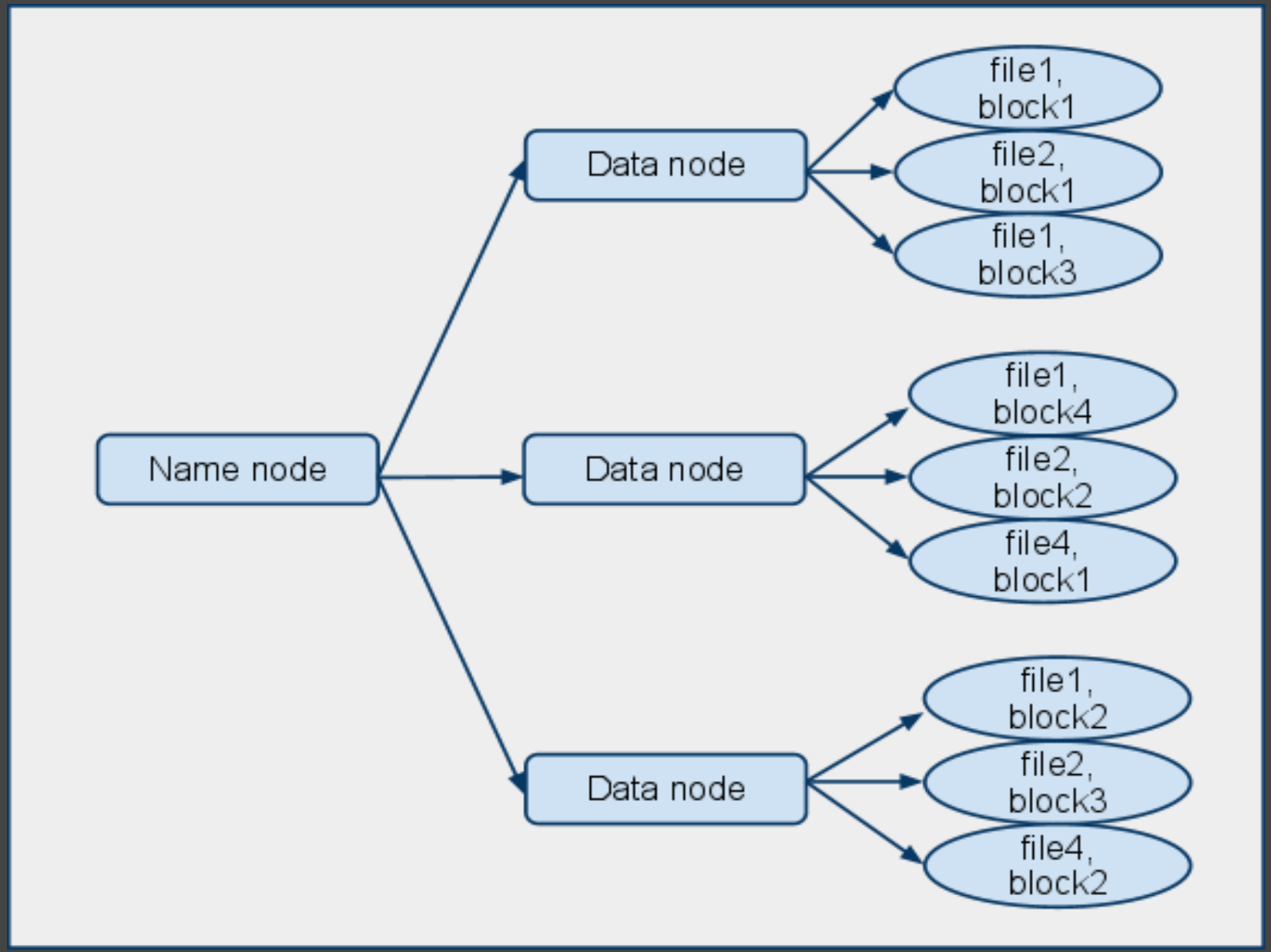
# Basic implementation



# GFS - HDFS introduction

- distributed file system
- files are divided in relatively large blocks, 64Mb
- blocks of one file are stored on multiple machines
- every block is replicated in configurable number of copies
- fault tolerance guaranteed
- Master - Slaves architecture

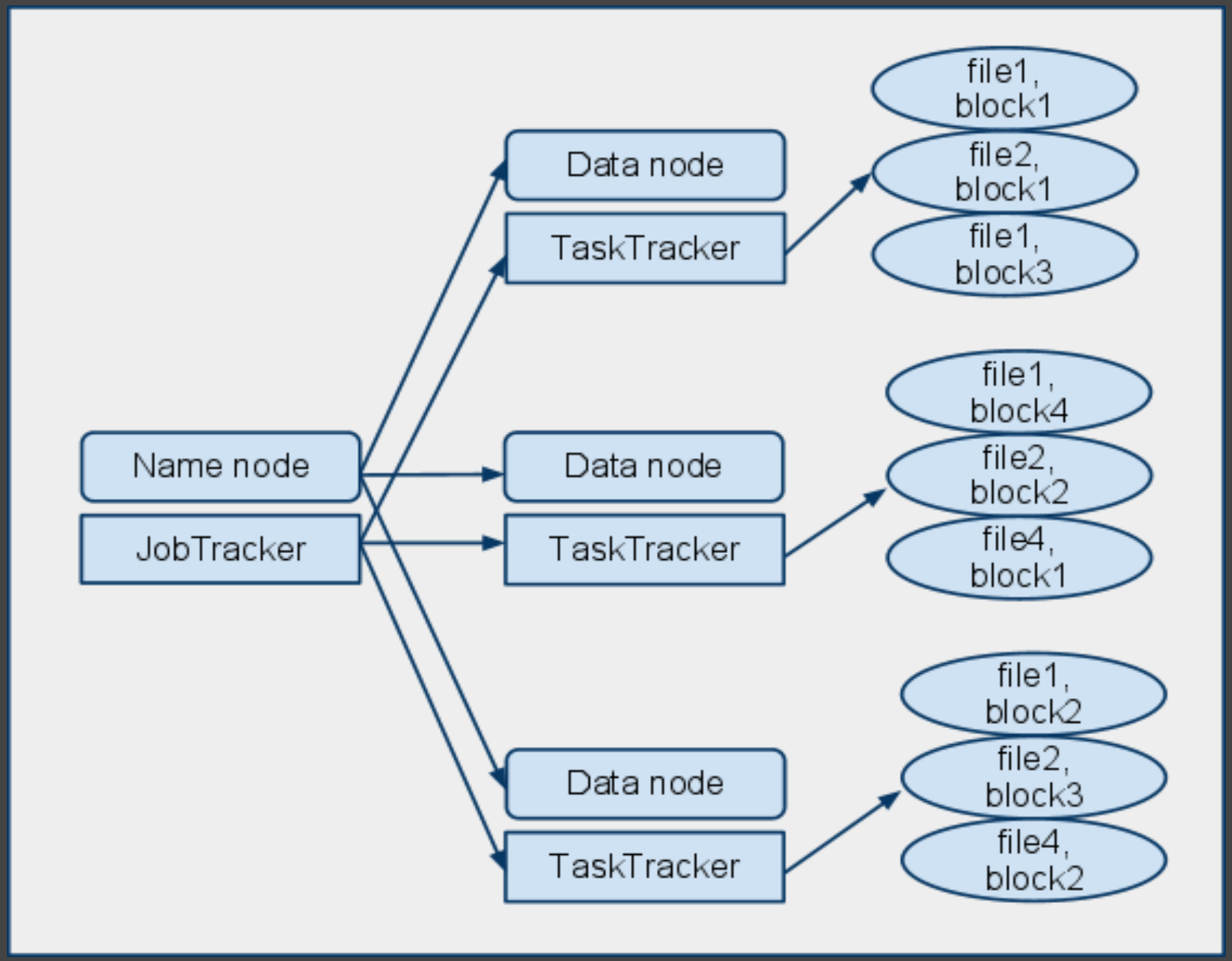
# GFS - HDFS introduction: overview



# MapReduce introduction

- framework build on top of (GFS) HDFS
- used for distributed data processing
- Master - Slaves architecture
- between data elements there is no dependency
- everything is executed in parallel
- two parts: Mapper and Reducer
- Mapper reads input data and emits ( key, value pairs ) to Reducer
- Data on Reducer is grouped by key
- Reducer produces output data
- Combiner and Partitioner - who are they, and what they want?

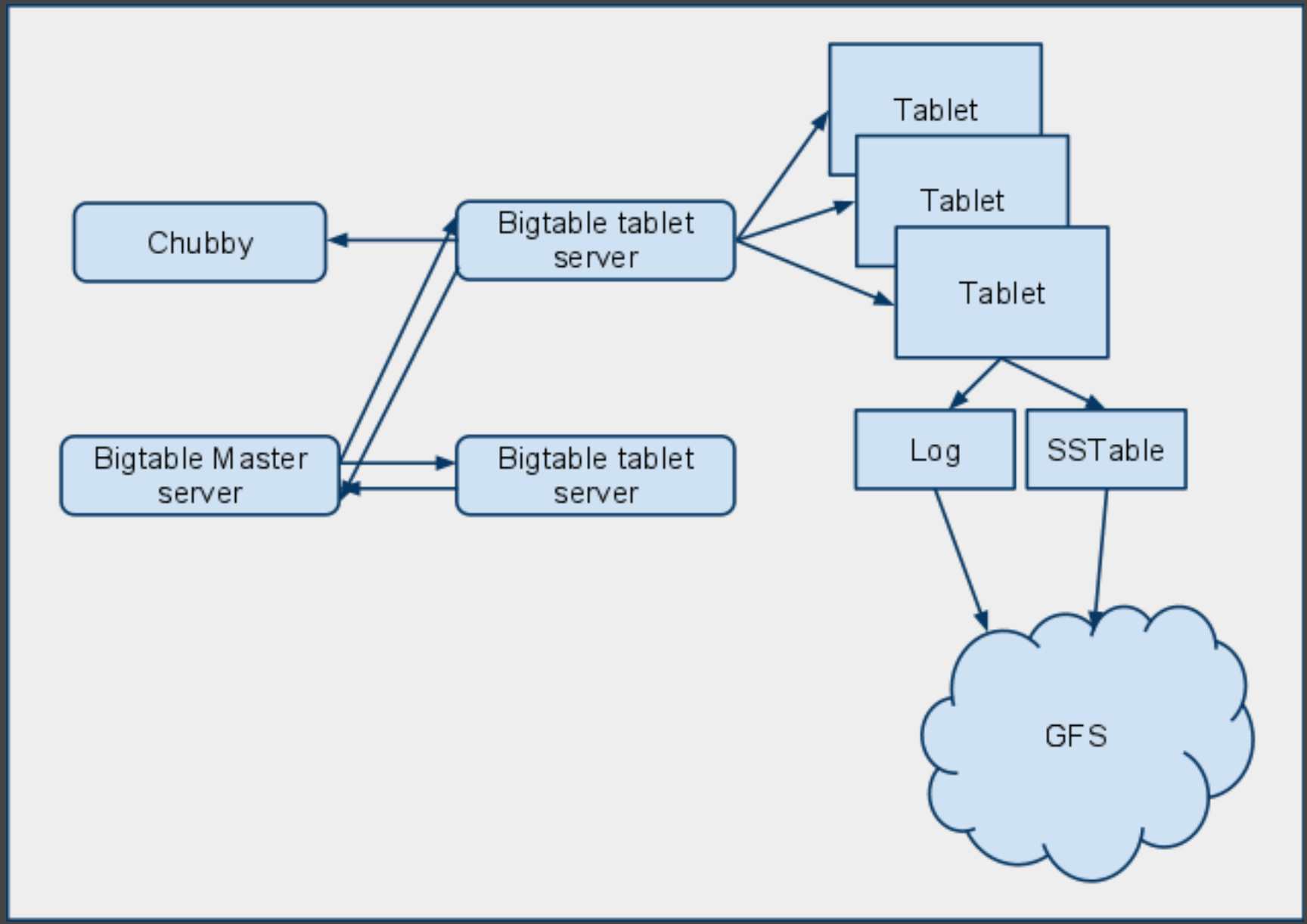
# MapReduce introduction: overview



# Implementation

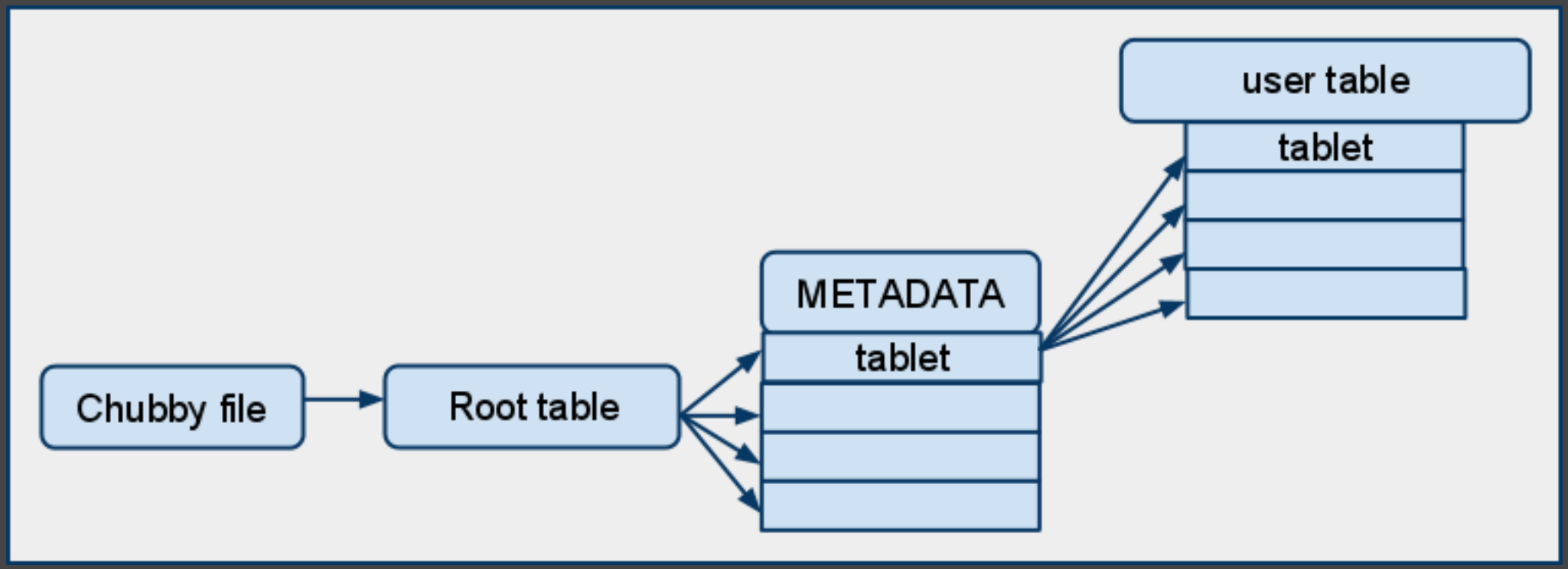
- Chubby - distributed lock service, synchronization point, start point for master, tablet server and even clients
- Master server, only one, guaranteed by Chubby
- in case of Master tragical death tablet servers continue to serve client
- Master decides who serves each tablet ( including ROOT and METATABLE )
- tablets are saved on GFS
- all trafic between clients and Bigtable is done directly within client and tablet server ( Master is not required )
- client caches METATABLE, on first fault recache is done

# Implementation: overview



# Implementation: oragnization

- two special tables: ROOT and METADATA
- ROOT never gets splitted
- location of ROOT is in Chubby file
- estimated ROOT size 128Mb => 2^34 tablets
- METADATA contains all user tablets, row key is an encoding of the tablet's table identifier and its end row



# Implementation

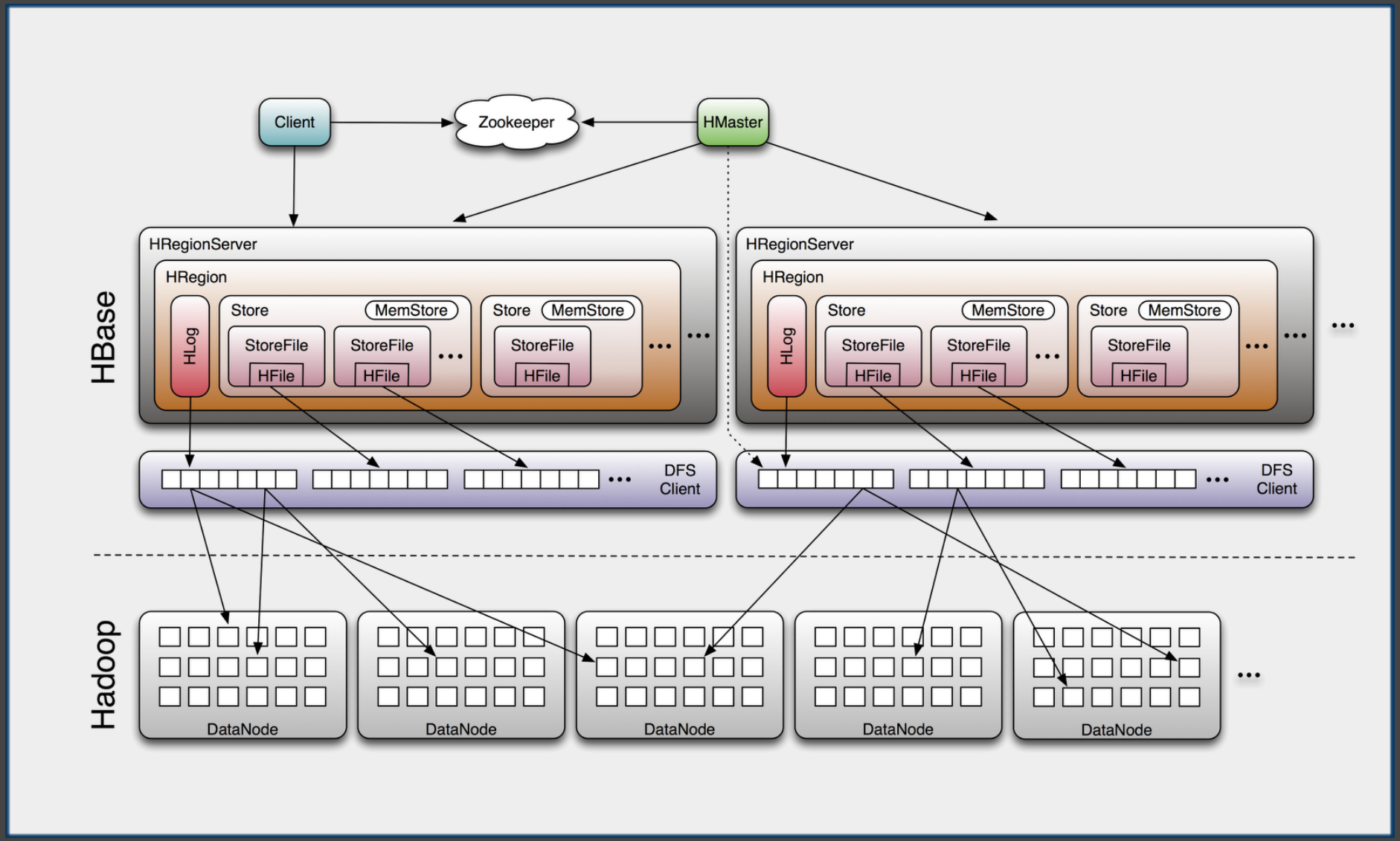
- SSTable - file representation of tablet
  - consist of blocks
  - column family labels are stored side by side
  - index of blocks at end of file

- compaction
  - minor
  - major

- locality groups

- caching

# HBase - Apache Bigtable solution



# HBase - Bigtable synonyms

- Bigtable ~ HBase
- tablet ~ region
- Master server ~ HBase master
- Tablet server ~ HBase Region server
- Chubby ~ Zookeeper ???
- GFS ~ HDFS
- MapReduce ~ MapReduce :)
- SSTable file ~ MapFile file

# HBase differences

- MapFiles index is stored in separate file instead at end of file as in SSTable

- Unlike Bigtable which identifies a row range by the table name and end-key, HBase identifies a row range by the table name and start-key

- when the HBase master dies, the cluster will shut down

# Performance and usage case

- number of 1000-byte values read/written per second
- table shows the rate per tablet server

| Experiment | 1 TS | 50 TS | 250 TS | 500 TS |
|---|---|---|---|---|
| random reads | 1212 | 593 | 479 | 241 |
| random reads mem | 10811 | 8511 | 8000 | 6250 |
| random writes | 8850 | 3745 | 3425 | 2000 |
| sequential reads | 4425 | 2463 | 2625 | 2469 |
| sequential writes | 8547 | 3623 | 2451 | 1905 |
| scans | 15385 | 10526 | 9524 | 7843 |

# Performances and usage case

Scaling:

- performance of random reads from memory increases by almost a factor of 300 as the number of tablet server increases by a factor of 500

- significant drop in per-server throughput when going from 1 to 50 tablet servers

- the random read benchmark shows the worst scaling (an increase in aggregate throughput by only a factor of 100 for a 500-fold increase in number of servers)

# Performances and usage case

Real users:

- Google Analytics:
    - It provides aggregate statistics, such as the number of unique visitors per day and the page views per URL per day
    - two tables - raw user clicks (~200Tb, 14% compression) and summary (~20Tb, 29% compression)

- Google Earth:
    - Google operates a collection of services that provide users with access to high-resolution satellite imagery of the world's surface
    - two tables - (~70Tb, imagery, no compression), (~500Gb, cache, served on hundreds of tablet servers

# Cooked by Vast: RecordStack

Introduction:

- long-term archival storage for vertical data to HDFS
- optimized for later analytical processing.
- stores:
    - listings data
    - transactional log data
    - user clicks
    - leads
    - other reporting stuff

# Cooked by Vast: RecordStack

*Goals:*

*- Efficient archival of records to HDFS - conserving space*
*- Checksuming records and tracking change*
*- Optmization of HDFS - stored structure by periodical data compacting*
*- Simple access from Hadoop tasks for analytical processing*
*- Simple access via client application - Remote console*
*- generation of reports*

# Some thoughts for discussion

- doing Bigtable with some number of "classic" databases?

- expose of Chubby to client, is it required?

- why not open source Bigtable?

- run of tablet servers on data nodes - closer to data?

# Useful links

- Scalability: http://scalability.rs/

- NoSQL Summer: http://nosqlsummer.org/

- Google Bigtable paper: http://tinyurl.com/37rlevv

- HBase Architecture: http://tinyurl.com/3y3fhbk

- HBase related blog: http://www.larsgeorge.com/

- Hadoop: The Definitive guide: http://tinyurl.com/36ponz3

# If you, maybe, want to contact me?

e-mail: vanjakom@gmail.com, vanja@vast.com

skype: komadinovic.vanja

mobile: =381 (64) 296 03 43

twitter: vanjakom

Thanks !!!