第六讲 云存储 之
以存储为中心的计算架构

陈华钧

浙江大学计算机科学与技术学院 教授

提纲

一、以存储为中心的计算架构

二、可扩展的半结构化数据存储

以存储为中心的计算架构

什么是Hadoop和MapReduce?

- MapReduce最早由Google提出，用于处理云中P级的大数据
  - Processes 20 PB of data per day
- MapReduce是一种专用于大规模数据的并行编程框架；
- MapReduce依赖于底层的文件系统，MapReduce程序可以方便的在万级以上的大规模廉价集群中部署和运行。
  - Data-parallel programming model for clusters of commodity machines
- Hadoop是支持MapReduce的最大开源平台
  - Used by Yahoo!, Facebook, Amazon, …

MapReduce能够做什么？

- Google:
  - Index building for Google Search
  - Article clustering for Google News
  - Statistical machine translation
- Yahoo!:
  - Index building for Yahoo! Search
  - Spam detection for Yahoo! Mail
- Facebook:
  - Data mining
  - Ad optimization
  - Spam detection

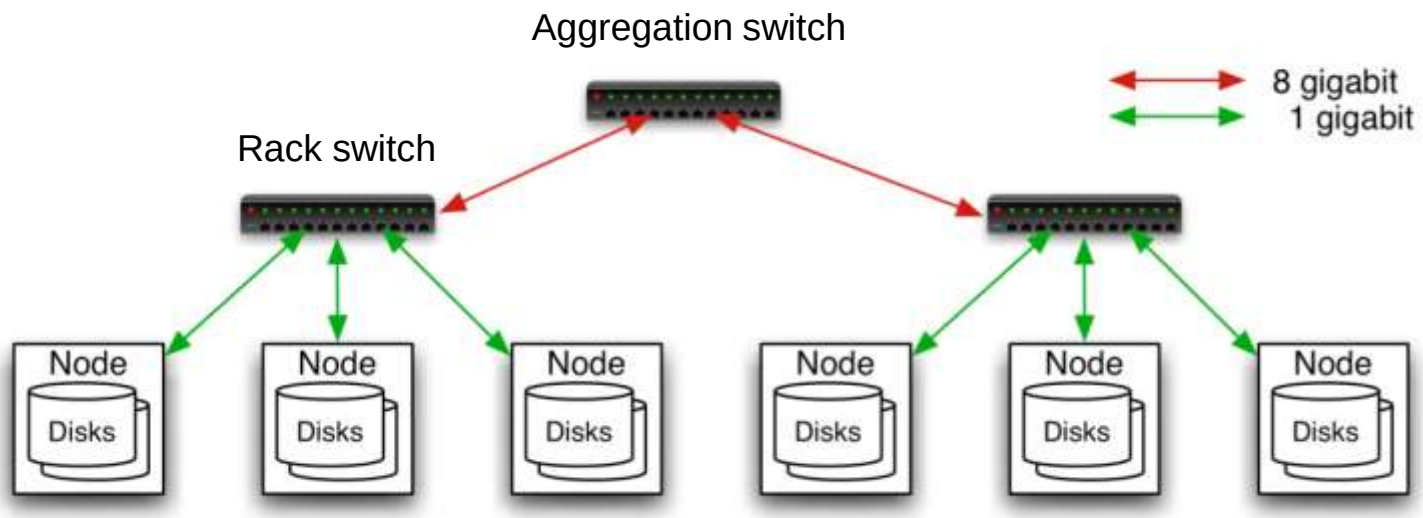
MapReduce能够做什么？

- In research:
  - Analyzing Wikipedia conflicts (PARC)
  - Natural language processing (CMU)
  - Bioinformatics (Maryland)
  - Astronomical image analysis (Washington)
  - Ocean climate simulation (Washington)
  - <Your application here>

MapReduce的设计目标

1. 可扩展性——Scalability
   - Scan 100 TB on 1 node @ 50 MB/s = 23 days
   - Scan on 1000-node cluster = 33 minutes
2. 省钱——Cost-efficiency:
   - Commodity nodes (cheap, but unreliable)
   - Commodity network
   - Automatic fault-tolerance (fewer admins)
   - Easy to use (fewer programmers)

典型的Hadoop集群



- 40 nodes/rack, 1000-4000 nodes in cluster
- 1 GBps bandwidth in rack, 8 GBps out of rack
- Node specs (Yahoo terasort):
  8 x 2.0 GHz cores, 8 GB RAM, 4 disks (= 4 TB?)

典型的Hadoop集群

Image from http://wiki.apache.org/hadoop-data/attachments/HadoopPresentations/attachments/aw-apachecon-eu-2009.pdf

主要针对的挑战

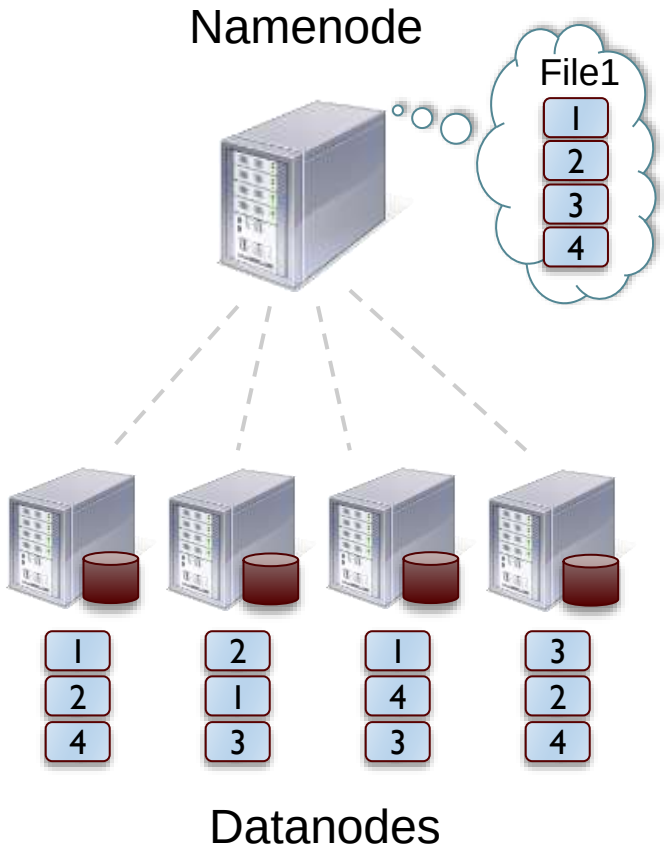
- 容错：**Cheap nodes fail, especially if you have many**
  - Mean time between failures for 1 node = 3 years
  - MTBF for 1000 nodes = 1 day
  - Solution: Build fault-tolerance into system
- 低带宽：**Commodity network = low bandwidth**
  - Solution: Push computation to the data
- 分布式系统编程：**Programming distributed systems is hard**
  - Solution: Data-parallel programming model: users write "map" and "reduce" functions, system handles work distribution and fault tolerance

Hadoopde的核心构成

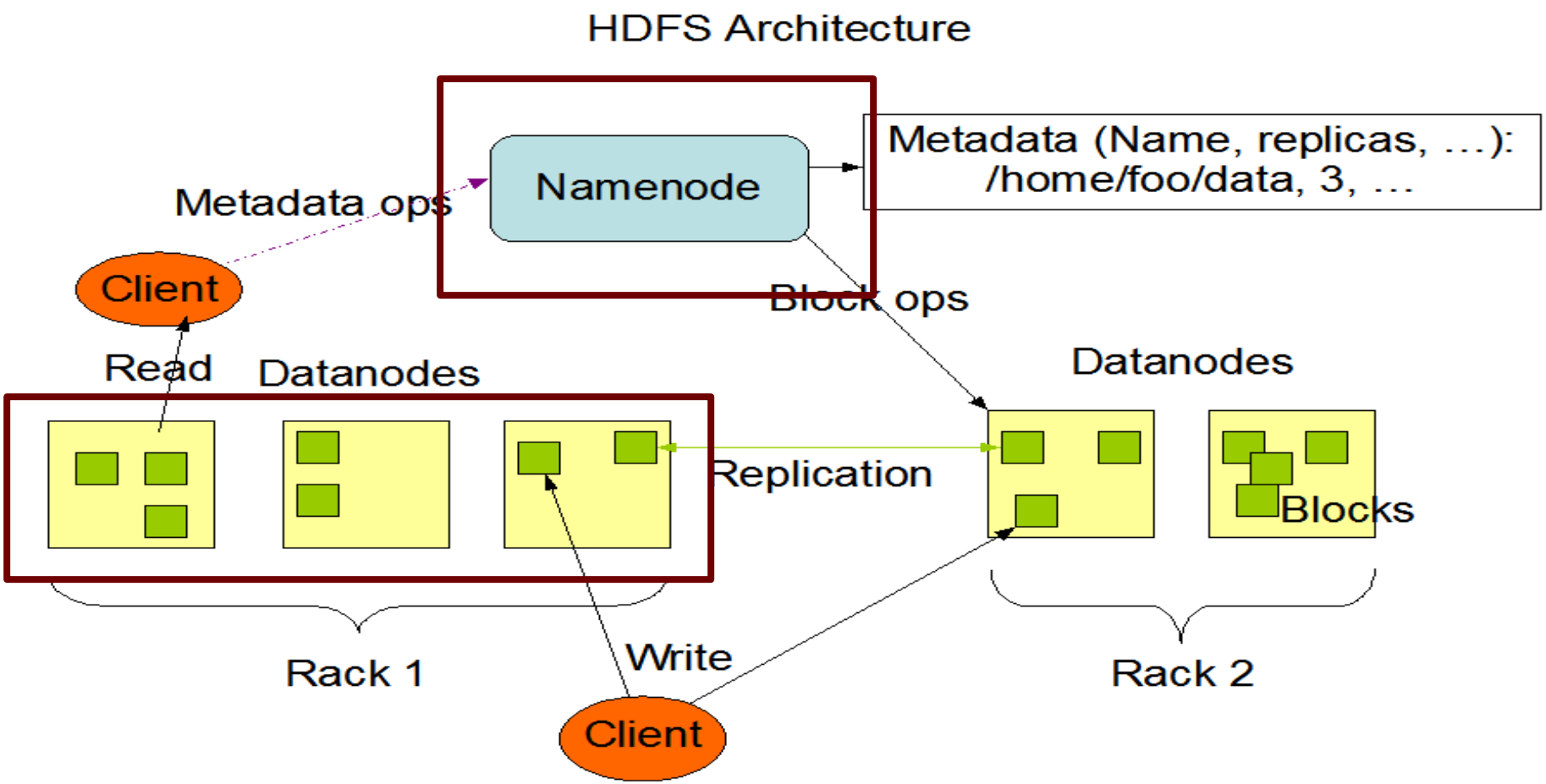
- Distributed file system (HDFS)
  - Single namespace for entire cluster
  - Replicates data 3x for fault-tolerance
- MapReduce implementation
  - Executes user jobs specified as “map” and “reduce” functions
  - Manages work distribution & fault-tolerance

HDFS: Hadoop分布式文件系统

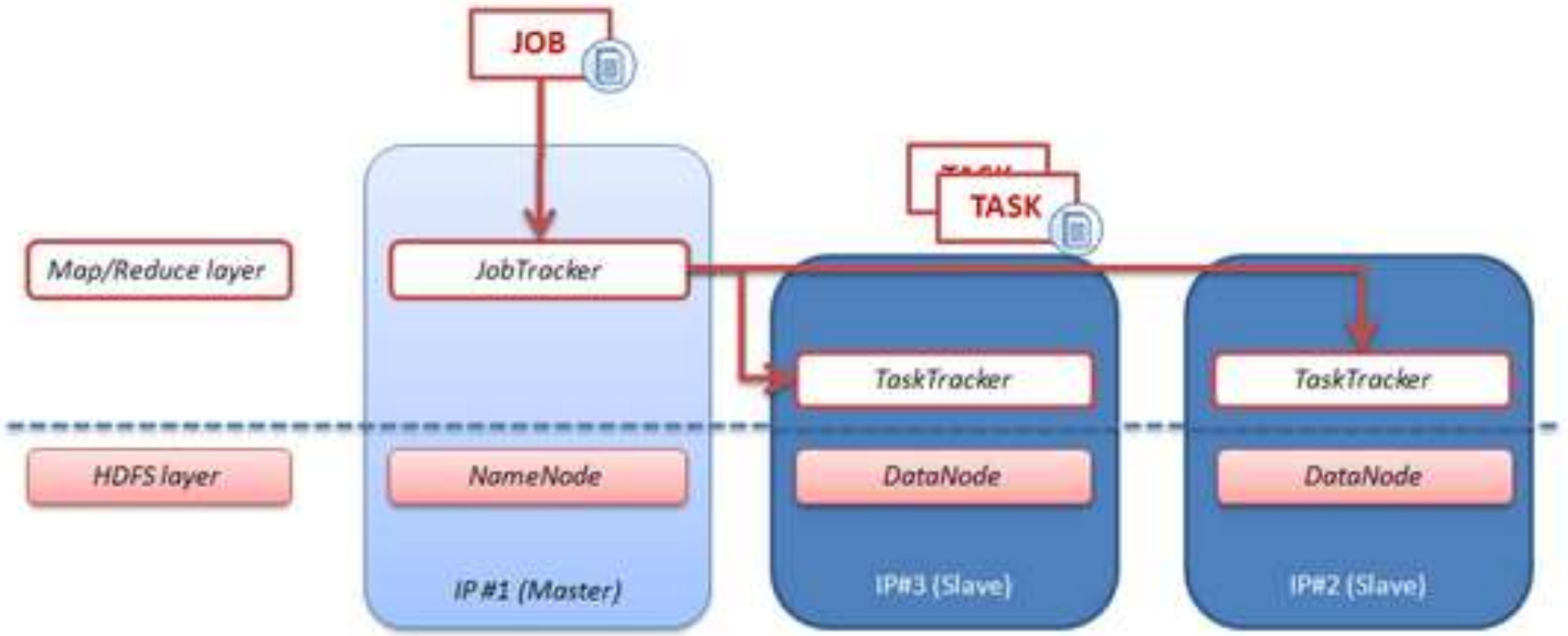
- Hadoop Distributed File System
  - Files split into 128MB *blocks*
  - Blocks replicated across several *datanodes* (usually 3)
  - Single *namenode* stores metadata (file names, block locations, etc)
  - Optimized for large files, sequential reads
  - Files are append-only



Hadoop架构-HDFS



Hadoop架构-MapReduce



MapReduce 编程模型

- Data type: key-value *records*

- Map function:

$$(K_{in}, V_{in}) \rightarrow list(K_{inter}, V_{inter})$$

- Reduce function:

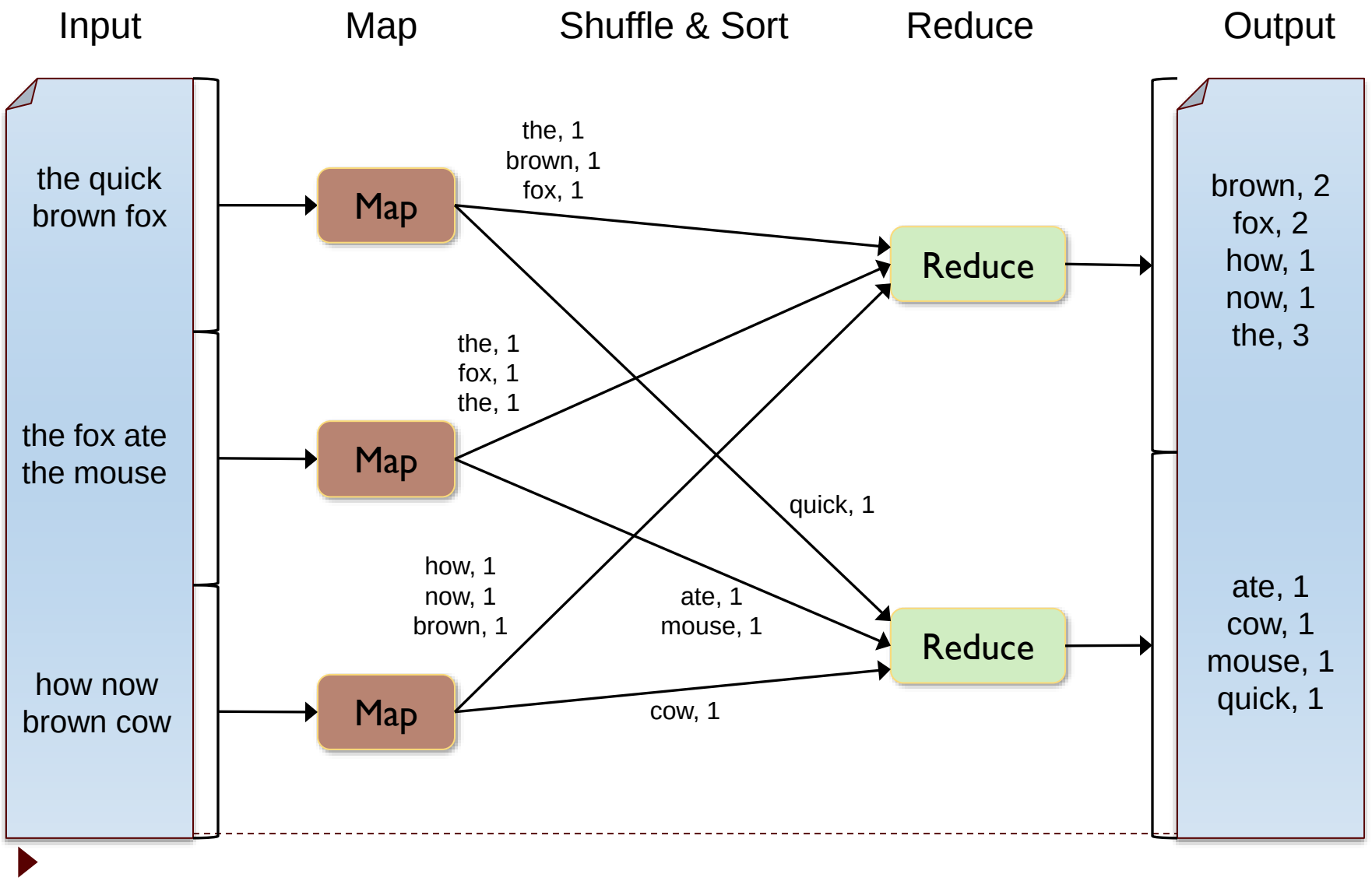
$$(K_{inter}, list(V_{inter})) \rightarrow list(K_{out}, V_{out})$$

举例: Word Count

```
def mapper(line):
  foreach word in line.split():
    output(word, 1)

def reducer(key, values):
  output(key, sum(values))
```

Word Count Execution



MapReduce程序的执行过程

- Master节点控制着多个Salve节点上的任务执行，并负责用户段的调度
- Mappers优先部署于与输入数据相同的节点或机架上。
  - Push computation to data, minimize network use
- Mappers 将结果直接保存于本地磁盘，而不是推送给Reducers
- Reducers继续处理这些分布于集群各处的中间结果，通常Reducers比节点数要多，并且允许Reducer在执行失败的情况下自动重启。

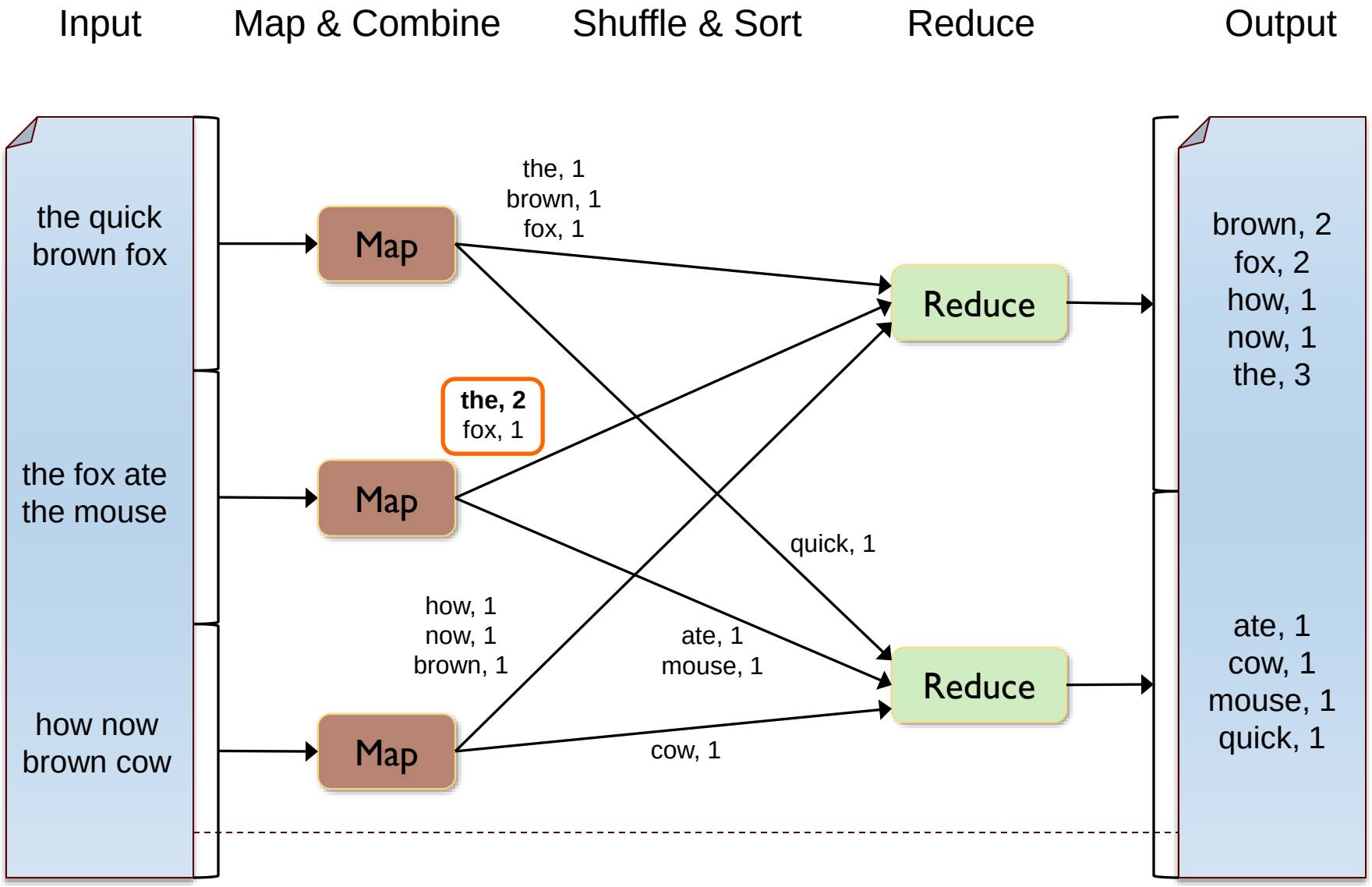
增加优化步骤: The Combiner

- A combiner is a local aggregation function for repeated keys produced by same map
- For associative ops. like sum, count, max
- Decreases size of intermediate data

- Example: local counting for Word Count:

```
def combiner(key, values):
  output(key, sum(values))
```

Word Count with Combiner



MapReduce容错机制

1. If a task crashes:
   - Retry on another node
     - Okay for a map because it had no dependencies
     - Okay for reduce because map outputs are on disk
   - If the same task repeatedly fails, fail the job or ignore that input block (user-controlled)

➢ Note: For this and the other fault tolerance features to work, *your map and reduce tasks must be side-effect-free*

MapReduce容错机制

2. If a node crashes:
   - Relaunch its current tasks on other nodes
   - Relaunch any maps the node previously ran
     - Necessary because their output files were lost along with the crashed node

MapReduce容错机制

3. If a task is going slowly (straggler):
   - Launch second copy of task on another node
   - Take the output of whichever copy finishes first, and kill the other one

- Critical for performance in large clusters: stragglers occur frequently due to failing hardware, bugs, misconfiguration, etc

更多例子： Search

- **Input:** (lineNumber, line) records
- **Output:** lines matching a given pattern

- **Map:**

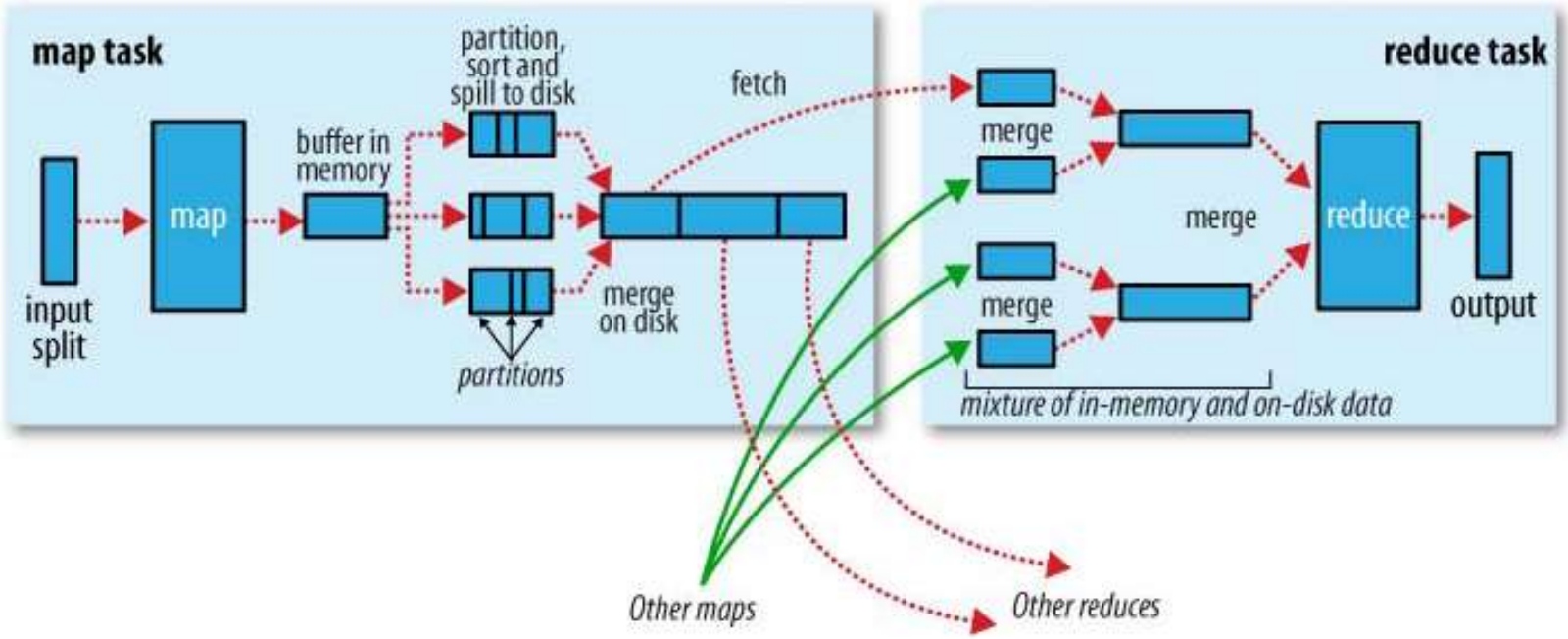
```
if(line matches pattern):
   output(line)
```

- **Reduce:** identify function
  - Alternative: no reducer (map-only job)

更多例子：Sort

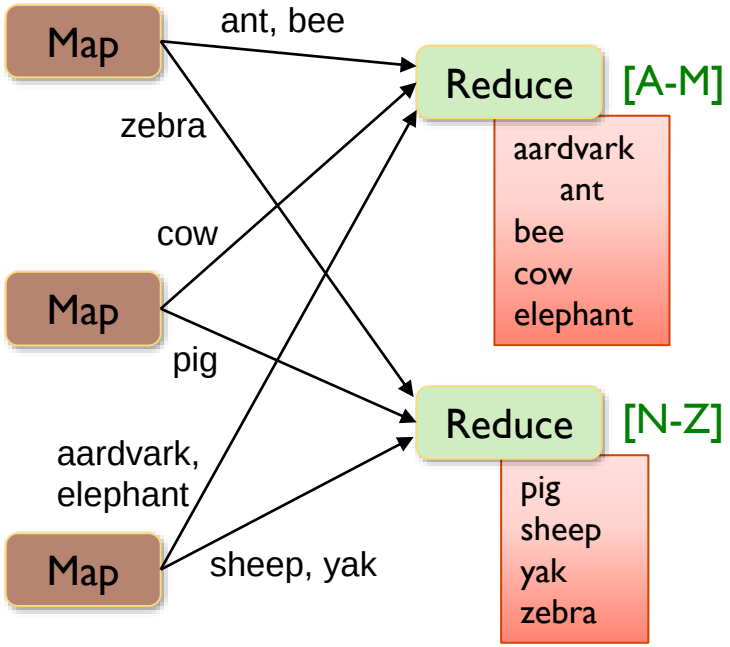
- **Input:** (key, value) records
- **Output:** same records, sorted by key

- **Map:** 先做局部排序
- **Reduce:** 再负责Merge Map阶段的成果，生成全局排序



更多例子： Sort

- 优化手段：多个Reduce分工合作
  - **Trick:** Pick partitioning function h such that $k_1<k_2 \Rightarrow h(k_1)<h(k_2)$



更多例子： Inverted Index

- **Input:** (filename, text) records
- **Output:** list of files containing each word

- **Map:**

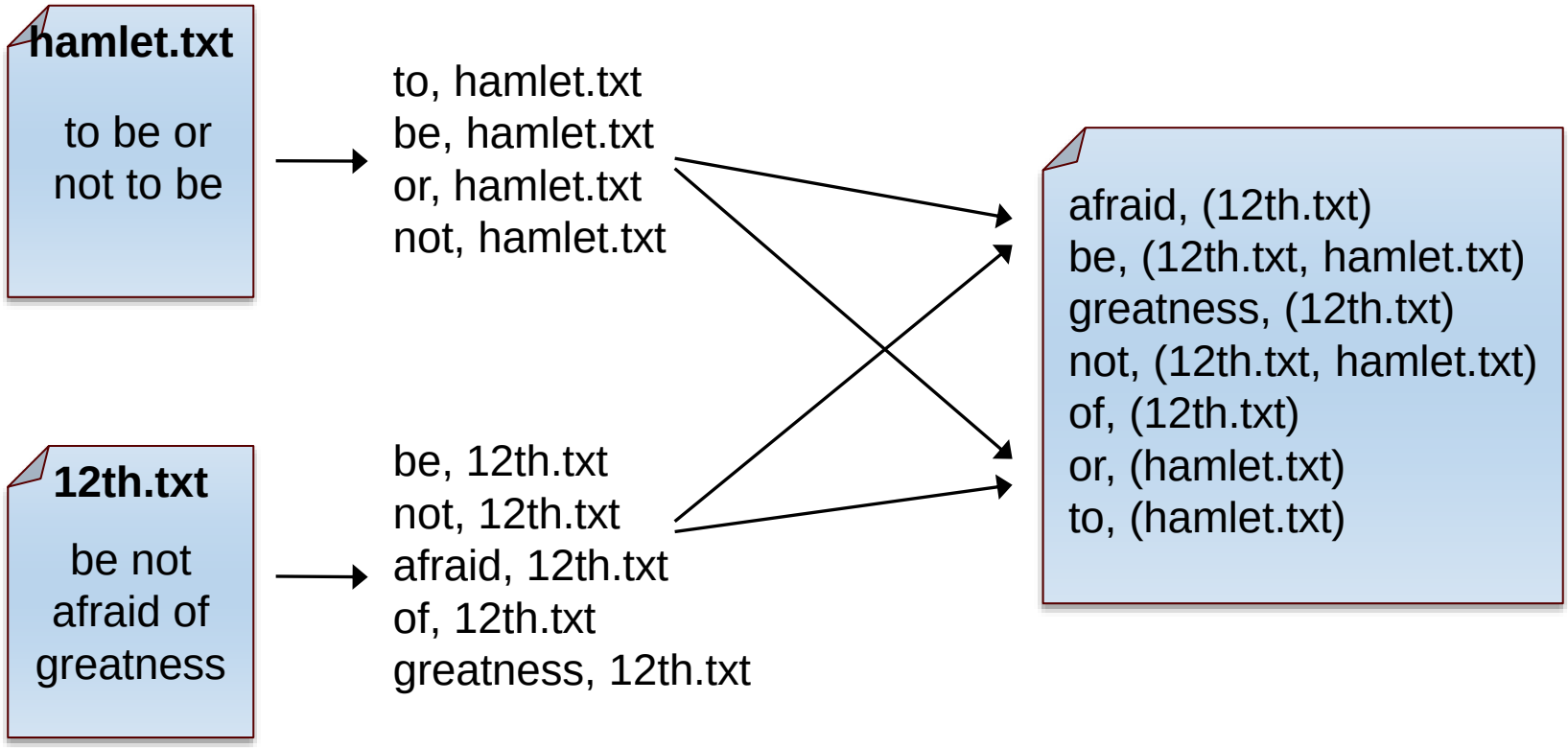
```
foreach word in text.split():
    output(word, filename)
```

- **Combine:** uniquify filenames for each word

- **Reduce:**

```
def reduce(word, filenames):
    output(word, sort(filenames))
```

更多例子： Inverted Index Example



更多例子： Most Popular Words

- **Input:** (filename, text) records
- **Output:** the 100 words occurring in most files

- Two-stage solution:
  - **Job 1:**
    - Create inverted index, giving (word, list(file)) records
  - **Job 2:**
    - Map each (word, list(file)) to (count, word)
    - Sort these records by count as in sort job

- Optimizations:
  - Map to (word, 1) instead of (word, file) in Job 1
  - Estimate count distribution in advance by sampling

Hadoop安装

- Download from hadoop.apache.org
- To install locally, unzip and set JAVA_HOME
- Details: hadoop.apache.org/core/docs/current/quickstart.html

- Three ways to write jobs:
  - Java API
  - Hadoop Streaming (for Python, Perl, etc)
  - Pipes API (C++)

Word Count in Java

```
public static class MapClass extends MapReduceBase
    implements Mapper<LongWritable, Text, Text, IntWritable> {

    private final static IntWritable ONE = new IntWritable(1);

    public void map(LongWritable key, Text value,
                    OutputCollector<Text, IntWritable> output,
                    Reporter reporter) throws IOException {
      String line = value.toString();
      StringTokenizer itr = new StringTokenizer(line);
      while (itr.hasMoreTokens()) {
        output.collect(new text(itr.nextToken()), ONE);
      }
    }
  }
```

Word Count in Java

```
public static class Reduce extends MapReduceBase
    implements Reducer<Text, IntWritable, Text, IntWritable> {

    public void reduce(Text key, Iterator<IntWritable> values,
                       OutputCollector<Text, IntWritable> output,
                       Reporter reporter) throws IOException {
      int sum = 0;
      while (values.hasNext()) {
        sum += values.next().get();
      }
      output.collect(key, new IntWritable(sum));
    }
  }
```

Word Count in Java

```
public static void main(String[] args) throws Exception {
    JobConf conf = new JobConf(WordCount.class);
    conf.setJobName("wordcount");

    conf.setMapperClass(MapClass.class);
    conf.setCombinerClass(Reduce.class);
    conf.setReducerClass(Reduce.class);

    FileInputFormat.setInputPaths(conf, args[0]);
    FileOutputFormat.setOutputPath(conf, new Path(args[1]));

    conf.setOutputKeyClass(Text.class); // out keys are words (strings)
    conf.setOutputValueClass(IntWritable.class); // values are counts

    JobClient.runJob(conf);
  }
```

Word Count in Python with Hadoop Streaming

Mapper.py:

```
import sys
for line in sys.stdin:
  for word in line.split():
    print(word.lower() + "\t" + 1)
```

Reducer.py:

```
import sys
counts = {}
for line in sys.stdin:
  word, count = line.split("\t")
    dict[word] = dict.get(word, 0) + int(count)
for word, count in counts:
  print(word.lower() + "\t" + 1)
```

MapReduce的问题

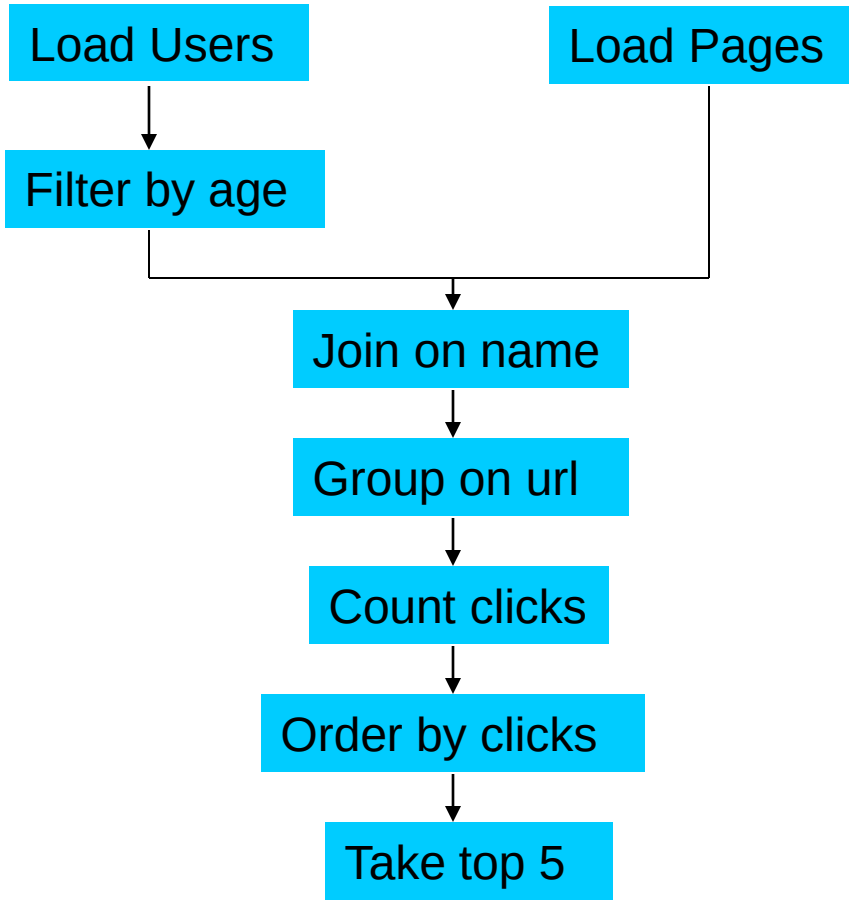
- MapReduce is great, as many algorithms can be expressed by a series of MR jobs
- But it's low-level: must think about keys, values, partitioning, etc
- Can we capture common "job patterns"?

Pig简介

- Started at Yahoo! Research
- Now runs about 30% of Yahoo!'s jobs
- Features:
  - Expresses sequences of MapReduce jobs
  - Data model: nested "bags" of items
  - Provides relational (SQL) operators (JOIN, GROUP BY, etc)
  - Easy to plug in Java functions
  - Pig Pen dev. env. for Eclipse

一个简单的例子

Suppose you have user data in one file, website data in another, and you need to find the top 5 most visited pages by users aged 18 - 25.



Example from http://wiki.apache.org/pig-data/attachments/PigTalksPapers/attachments/ApacheConEurope09.ppt

复杂的MapReduce程序

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.Writable;
import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.KeyValueTextInputFormat;
import org.a  pache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.RecordReader;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;
import org.apache.hadoop.mapred.SequenceFileInputFormat;
import org.apache.hadoop.mapred.SequenceFileOutputFormat;
import org.apache.hadoop.mapred.TextInputFormat;
import org.apache.hadoop.mapred.jobcontrol.Job;
import org.apache.hadoop.mapred.jobcontrol.JobC      ontrol;
import org.apache.hadoop.mapred.lib.IdentityMapper;

public class MRExample {
    public static class LoadPages extends MapReduceBase
        implements Mapper<LongWritable, Text, Text, Text> {

        public void map(LongWritable k, Text val,
                        OutputCollector<Text, Text> oc,
                        Reporter reporter) throws IOException {
            // Pull the key out
            String line = val.toString();
            int firstComma = line.indexOf(',');
            String key = line.sub     string(0, firstComma);
            String value = line.substring(firstComma + 1);
            Text outKey = new Text(key);
            // Prepend an index to the value so we know which file
            // it came from.
            Text outVal = new Text("1     " + value);
            oc.collect(outKey, outVal);
        }
    }
    public static class LoadAndFilterUsers extends MapReduceBase
        implements Mapper<LongWritable, Text, Text, Text> {

        public void map(LongWritable k, Text val,
                        OutputCollector<Text, Text> oc,
                        Reporter reporter) throws IOException {
            // Pull the key out
            String line = val.toString();
            int firstComma = line.indexOf(',');
            String value = line.substring(     firstComma + 1);
            int age = Integer.parseInt(value);
            if (age < 18 || age > 25) return;
            String key = line.substring(0, firstComma);
            Text outKey = new Text(key);
            // Prepend an index to the value so w      e know which file
            // it came from.
            Text outVal = new Text("2" + value);
            oc.collect(outKey, outVal);
        }
    }
    public static class Join extends MapReduceBase
        implements Reducer<Text, Text, Text, Text> {

        public void reduce(Text key,
                Iterator<Text> iter,
                OutputCollector<Text, Text> oc,
                Reporter reporter) throws IOException {
            // For each value, figure out which file it's from and
store it
            // accordingly.
            List<String> first = new ArrayList<String>();
            List<String> second = new ArrayList<String>();

            while (iter.hasNext()) {
                Text t = iter.next();
                String value = t.to    String();
                if (value.charAt(0) == '1')
first.add(value.substring(1));
                else second.add(value.substring(1));
                reporter.setStatus("OK");
            }

            // Do the cross product and collect the values
            for (String s1 : first) {
                for (String s2 : second) {
                    String outval = key + "," + s1 + "," + s2;
                    oc.collect(null, new Text(outval));
                    reporter.setStatus("OK");
                  }
            }
        }
    }
    public static class LoadJoined extends MapReduceBase
        implements Mapper<Text, Text, Text, LongWritable> {

        public void map(
                Text k,
                Text val,
                OutputColle   ctor<Text, LongWritable> oc,
                Reporter reporter) throws IOException {
            // Find the url
            String line = val.toString();
            int firstComma = line.indexOf(',');
            int secondComma = line.indexOf(',', first      Comma);
            String key = line.substring(firstComma, secondComma);
            // drop the rest of the record, I don't need it anymore,
            // just pass a 1 for the combiner/reducer to sum instead.
            Text outKey = new Text(key);
            oc.collect(outKey, new LongWritable(1L));
        }
    }
    public static class ReduceUrls extends MapReduceBase
        implements Reducer<Text, LongWritable, WritableComparable,
Writable> {

        public void reduce(
                Text ke   y,
                Iterator<LongWritable> iter,
                OutputCollector<WritableComparable, Writable> oc,
                Reporter reporter) throws IOException {
            // Add up all the values we see

            long sum = 0;
            wh  ile (iter.hasNext()) {
                sum += iter.next().get();
                reporter.setStatus("OK");
            }

            oc.collect(key, new LongWritable(sum));
        }
    }
    public static class LoadClicks extends MapReduceBase
        i mplements Mapper<WritableComparable, Writable, LongWritable,
Text> {

        public void map(
                WritableComparable key,
                Writable val,
                OutputCollector<LongWritable, Text> oc,
                Reporter reporter)     throws IOException {
            oc.collect((LongWritable)val, (Text)key);
        }
    }
    public static class LimitClicks extends MapReduceBase
        implements Reducer<LongWritable, Text, LongWritable, Text> {

        int count = 0;
        public   void reduce(
            LongWritable key,
            Iterator<Text> iter,
            OutputCollector<LongWritable, Text> oc,
            Reporter reporter) throws IOException {

            // Only output the first 100 records
            while (count    < 100 && iter.hasNext()) {
                oc.collect(key, iter.next());
                count++;
            }

        }
    }
    public static void main(String[] args) throws IOException {
        JobConf lp = new JobConf(MRExample.class);
        lp.se  tJobName("Load Pages");
        lp.setInputFormat(TextInputFormat.class);
        lp.setOutputKeyClass(Text.class);
        lp.setOutputValueClass(Text.class);
        lp.setMapperClass(LoadPages.class);
        FileInputFormat.addInputPath(lp, new
Path("/ user/gates/pages"));
        FileOutputFormat.setOutputPath(lp,
            new Path("/user/gates/tmp/indexed_pages"));
        lp.setNumReduceTasks(0);
        Job loadPages = new Job(lp);

        JobConf lfu = new JobConf(MRExample.class);
        lfu.s  etJobName("Load and Filter Users");
        lfu.setInputFormat(TextInputFormat.class);
        lfu.setOutputKeyClass(Text.class);
        lfu.setOutputValueClass(Text.class);
        lfu.setMapperClass(LoadAndFilterUsers.class);
        FileInputFormat.add   InputPath(lfu, new
Path("/user/gates/users"));
        FileOutputFormat.setOutputPath(lfu,
            new Path("/user/gates/tmp/filtered_users"));
        lfu.setNumReduceTasks(0);
        Job loadUsers = new Job(lfu);

        JobConf join = new JobConf(    MRExample.class);
        join.setJobName("Join Users and Pages");
        join.setInputFormat(KeyValueTextInputFormat.class);
        join.setOutputKeyClass(Text.class);
        join.setOutputValueClass(Text.class);
        join.setMapperClass(IdentityMap     per.class);
        join.setReducerClass(Join.class);
        FileInputFormat.addInputPath(join, new
Path("/user/gates/tmp/indexed_pages"));
        FileInputFormat.addInputPath(join, new
Path("/user/gates/tmp/filtered_users"));
        FileOutputFormat.se    tOutputPath(join, new
Path("/user/gates/tmp/joined"));
        join.setNumReduceTasks(50);
        Job joinJob = new Job(join);
        joinJob.addDependingJob(loadPages);
        joinJob.addDependingJob(loadUsers);

        JobConf group = new JobConf(MRE     xample.class);
        group.setJobName("Group URLs");
        group.setInputFormat(KeyValueTextInputFormat.class);
        group.setOutputKeyClass(Text.class);
        group.setOutputValueClass(LongWritable.class);
        group.setOutputFormat(SequenceFi     leOutputFormat.class);
        group.setMapperClass(LoadJoined.class);
        group.setCombinerClass(ReduceUrls.class);
        group.setReducerClass(ReduceUrls.class);
        FileInputFormat.addInputPath(group, new
Path("/user/gates/tmp/joined"));
        FileOutputFormat.setOutputPath(group, new
Path("/user/gates/tmp/grouped"));
        group.setNumReduceTasks(50);
        Job groupJob = new Job(group);
        groupJob.addDependingJob(joinJob);

        JobConf top100 = new JobConf(MRExample.class);
        top100.setJobName("Top 100 sites");
        top100.setInputFormat(SequenceFileInputFormat.class);
        top100.setOutputKeyClass(LongWritable.class);
        top100.setOutputValueClass(Text.class);
        top100.setOutputFormat(SequenceFileOutputF      ormat.class);
        top100.setMapperClass(LoadClicks.class);
        top100.setCombinerClass(LimitClicks.class);
        top100.setReducerClass(LimitClicks.class);
        FileInputFormat.addInputPath(top100, new
Path("/user/gates/tmp/grouped"));
        FileOutputFormat.setOutputPath(top100, new
Path("/user/gates/top100sitesforusers18to25"));
        top100.setNumReduceTasks(1);
        Job limit = new Job(top100);
        limit.addDependingJob(groupJob);

        JobControl jc = new JobControl("Find top      100 sites for users
18 to 25");
        jc.addJob(loadPages);
        jc.addJob(loadUsers);
        jc.addJob(joinJob);
        jc.addJob(groupJob);
        jc.addJob(limit);
        jc.run();
    }
}
```

Example from http://wiki.apache.org/pig-data/attachments/PigTalksPapers/attachments/ApacheConEurope09.ppt

In Pig Latin

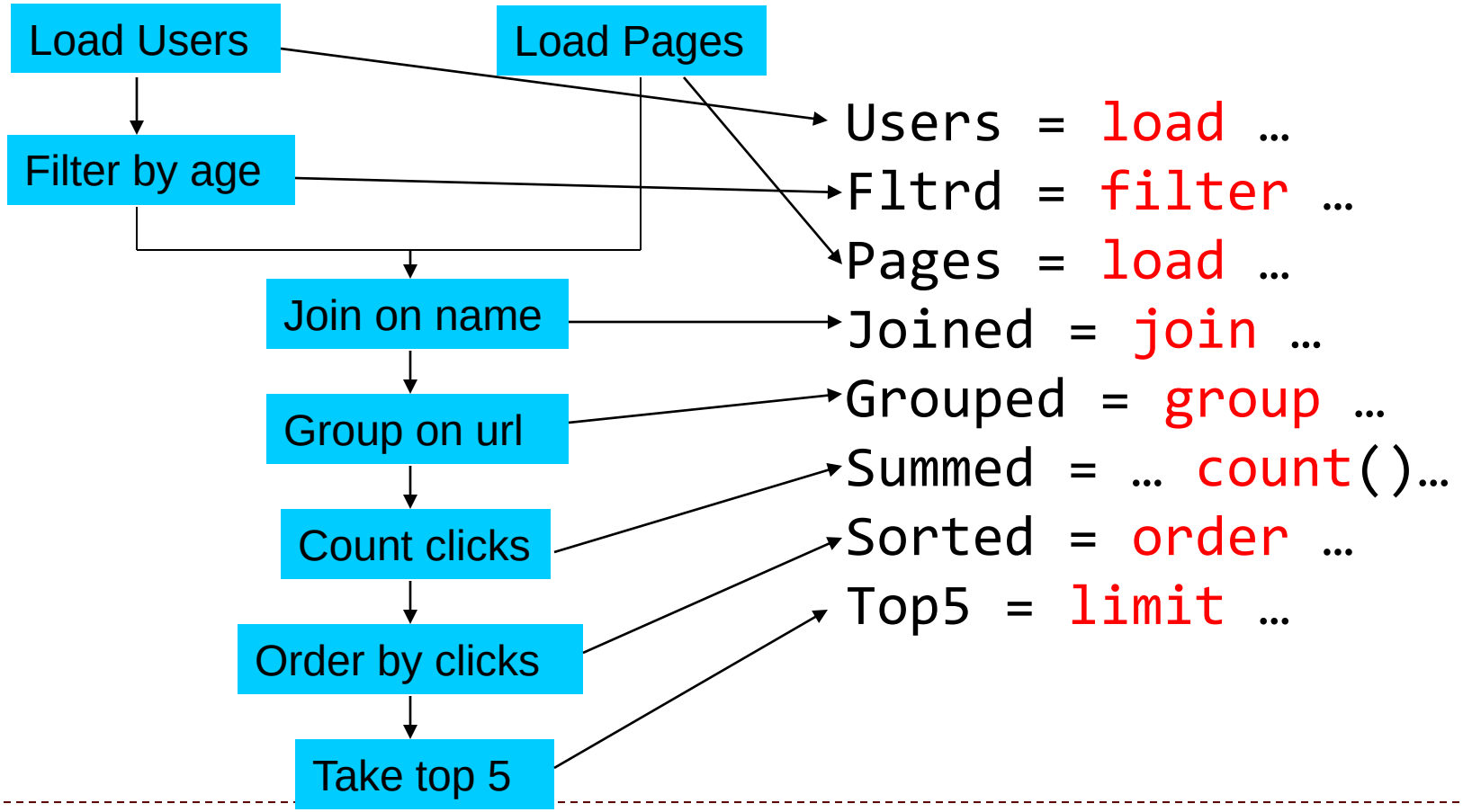
```
Users    = load ‘users’ as (name, age);
Filtered = filter Users by
                  age >= 18 and age <= 25;
Pages    = load ‘pages’ as (user, url);
Joined   = join Filtered by name, Pages by user;
Grouped  = group Joined by url;
Summed   = foreach Grouped generate group,
                   count(Joined) as clicks;
Sorted   = order Summed by clicks desc;
Top5     = limit Sorted 5;

store Top5 into ‘top5sites’;
```

Example from http://wiki.apache.org/pig-data/attachments/PigTalksPapers/attachments/ApacheConEurope09.ppt

PIG脚本的翻译

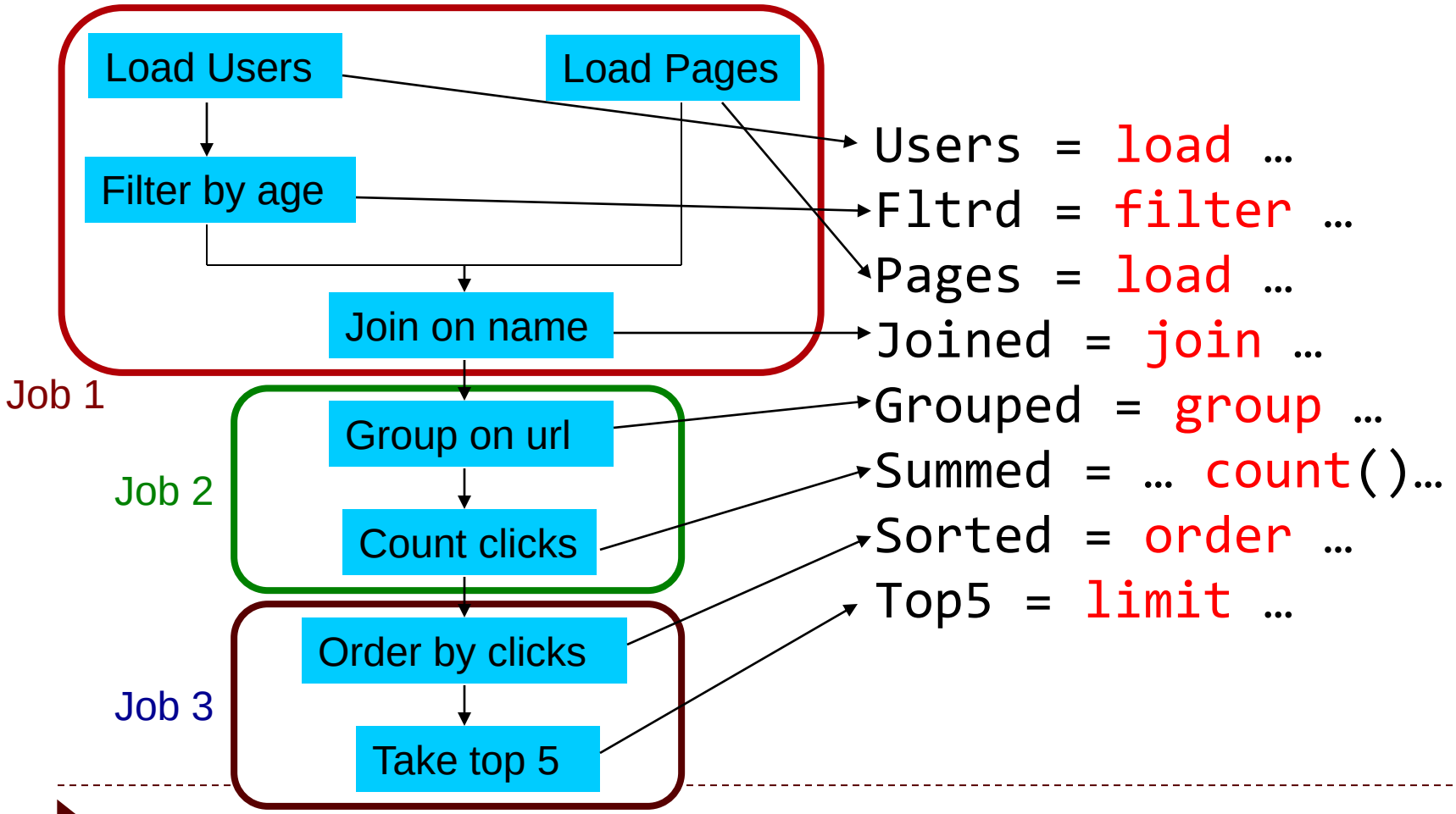
Notice how naturally the components of the job translate into Pig Latin.



Example from http://wiki.apache.org/pig-data/attachments/PigTalksPapers/attachments/ApacheConEurope09.ppt

PIG脚本的翻译

Notice how naturally the components of the job translate into Pig Latin.



Example from http://wiki.apache.org/pig-data/attachments/PigTalksPapers/attachments/ApacheConEurope09.ppt

Hive简介

- Developed at Facebook
- Used for majority of Facebook jobs
- "Relational database" built on Hadoop
  - Maintains list of table schemas
  - SQL-like query language (HQL)
  - Can call Hadoop Streaming scripts from HQL
  - Supports table partitioning, clustering, complex data types, some optimizations

创建Hive表

```
CREATE TABLE page_views(viewTime INT, userid BIGINT,
                        page_url STRING, referrer_url STRING,
                        ip STRING COMMENT 'User IP address')
COMMENT 'This is the page view table'
PARTITIONED BY(dt STRING, country STRING)
STORED AS SEQUENCEFILE;
```

- Partitioning breaks table into separate files for each (`dt, country`) pair

  Ex: `/hive/page_view/dt=2008-06-08,country=US`
  `/hive/page_view/dt=2008-06-08,country=CA`

简单的Hive查询

- Find all page views coming from xyz.com on March 31st:

```
SELECT page_views.*
FROM page_views
WHERE page_views.date >= '2008-03-01'
AND page_views.date <= '2008-03-31'
AND page_views.referrer_url like '%xyz.com';
```

- Hive only reads partition `2008-03-01,*` instead of scanning entire table

聚合和连接计算

- Count users who visited each page by gender:

```
SELECT pv.page_url, u.gender, COUNT(DISTINCT u.id)
FROM page_views pv JOIN user u ON (pv.userid = u.id)
GROUP BY pv.page_url, u.gender
WHERE pv.date = '2008-03-03';
```

- Sample output:

page_url	gender	count(userid)
home.php	MALE	12,141,412
home.php	FEMALE	15,431,579
photo.php	MALE	23,941,451
photo.php	FEMALE	21,231,314

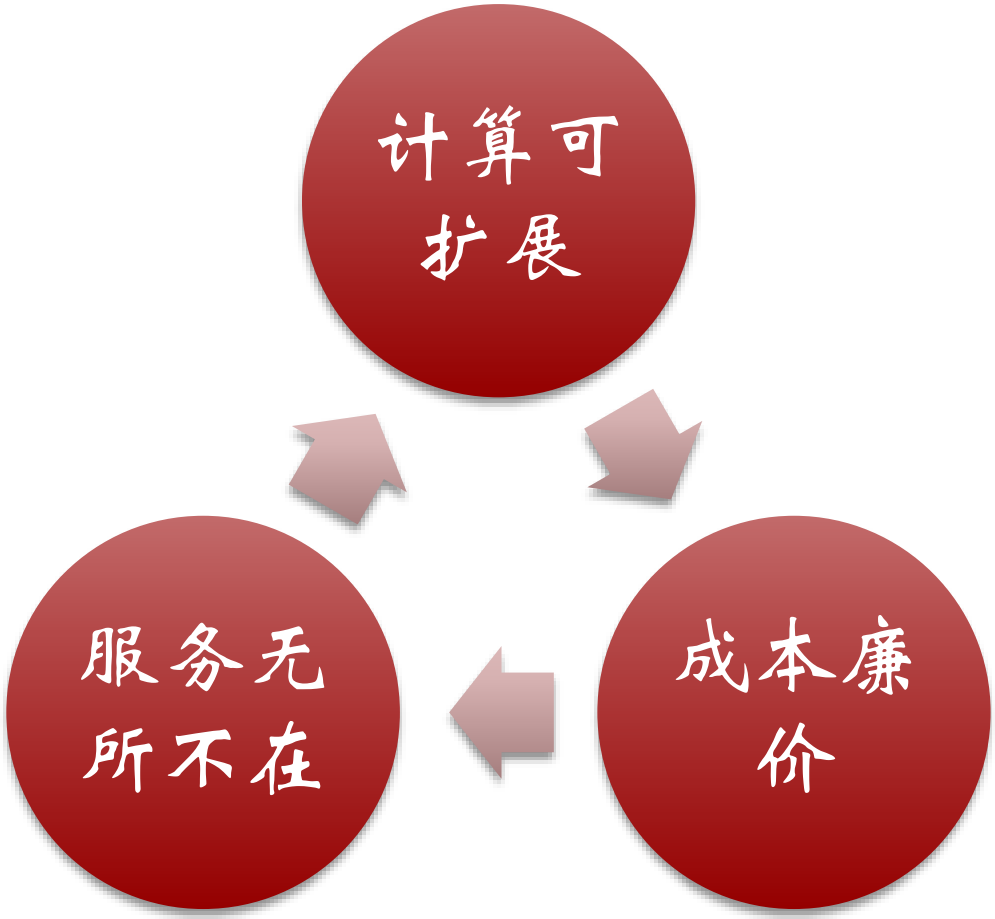
小节

- MapReduce提供了一个简单的编程框架，简化了在大规模数据集群上编写数据处理程序的复杂性，并提供了优秀的容错机制。
- 核心的原则
  - *Make it scale*, so you can throw hardware at problems
  - *Make it cheap*, saving hardware, programmer and administration costs (but requiring fault tolerance)

- PIG和Hive的目的是进一步简化和抽象编写MapReduce程序，类似于高级编程语言与汇编语言的关系。
- MapReduce并非万能，但对于一大类云中数据的处理问题，非常好用。

可扩展的半结构化数据存储

云的三要素



云时代关系模型不再是主角

- 传统关系模型在扩展性方面存在瓶颈
  - Required DB with wide scalability, wide applicability, high performance and high availability
- 相对比较昂贵
  - Most DBMSs require very expensive infrastructure
- 关系模型提供的很多特性对于很多应用是多余的
  - E.g., full transactions, SQL
- 新系统的新特性
  - GFS, Chubby, MapReduce, Job scheduling

NoSQL的兴起

- *Wikipedia*上定义：NoSQL是一种打破了关系型数据库长久以来占主导地位的快速成长起来的**非关系松散数据存储类型**，这种数据存储**不需要事先设计好的表结构**，它也不会出现表之间的连接操作和水平分割，学术界称这种数据库为结构化存储
- 产生的需求背景
  - 海量数据存储，SQL数据库可扩展性差，应对这类业务，代价很大
  - 很多非结构化的数据并不似乎和用RDBMS的表结构来建模
  - NoSQL舍弃RDBMS中的很多限制，从而满足高性能的需求
- 产品实现：Google的BigTable、Amazon的Dynamo、Apache的Hbase和Cassandra、Zvents的Hypertable

BigTable是什么？

- BigTable是Google实现的分布式大表模型系统-发表于OSDI2006
  - 存储海量非结构化数据
  - 只支持部分关系数据模型
  - 高可扩展
  - 数据的自动化管理
- Google超过60个产品使用它
  - Google Analytics
  - Google Finance
  - Personalized Search
  - Google Documents
  - Google Earth
  - Google Fusion Tables
  - …
- 很好的应对各类需求
  - 低延时的业务
  - 批处理业务

Bigtable模型的本质

A BigTable is a sparse, distributed, persistent multidimensional sorted map. The map is indexed by a row key, a column key, and a timestamp; each value in the map is an uninterpreted array of bytes.”

BigTable是一个稀疏、分布式的、持久存储、多维、有序图。

BigTable vs 关系模型

- 相比起传统DBMS, BigTable 提供的能力…
  - Simplified data retrieval mechanism
    - A map
    - <Row, Column, Timestamp> -> string
    - No relational operators
  - Atomic updates only possible at row level
  - Arbitrary number of columns per row
  - Arbitrary data type for each column
  - Provides extremely large scale (data, throughput) at extremely small cost

BigTable模型: Row

- Row keys are arbitrary strings
- Row is the unit of transactional consistency
  - Every read or write of data under a single row is atomic
- Data is maintained in lexicographic order by row key
- Rows with consecutive keys (Row Range) are grouped together as "*tablets*".
  - Unit of distribution and load-balancing
  - reads of short row ranges are efficient and typically require communication with only a small number of machines

BigTable模型: Column

- Column keys are grouped into sets called "*column families*", which form the unit of access control.
- Data stored under a column family is usually of the same type
- A column family must be created before data can be stored in a column key
- Column key is named using the following syntax: *family :qualifier*
- Access control and disk/memory accounting are performed at column family level

BigTable模型: timestamps

- Each cell in Bigtable can contain multiple versions of data, each indexed by timestamp
- Timestamps are 64-bit integers
- Assigned by:
  - Bigtable: real-time in microseconds
  - Client application: when unique timestamps are a necessity
- Data is stored in decreasing timestamp order, so that most recent data is easily accessed
  - Application specifies how many versions (n) of data items are maintained in a cell
  - Bigtable garbage collects obsolete versions

BigTable模型举例

Example: Zoo

BigTable模型举例

Example: Zoo

row key col. key timestamp

BigTable模型举例

Example: Zoo

row key col. key timestamp

- (zebras, length, 2006) --> 7 ft
- (zebras, weight, 2007) --> 600 lbs
- (zebras, weight, 2006) --> 620 lbs

BigTable模型举例

Example: Zoo

row key col. key timestamp

- (zebras, length, 2006) --> 7 ft
- (zebras, weight, 2007) --> 600 lbs
- (zebras, weight, 2006) --> 620 lbs

Each key is sorted in Lexicographic order

BigTable模型举例

Example: Zoo

row key col. key timestamp

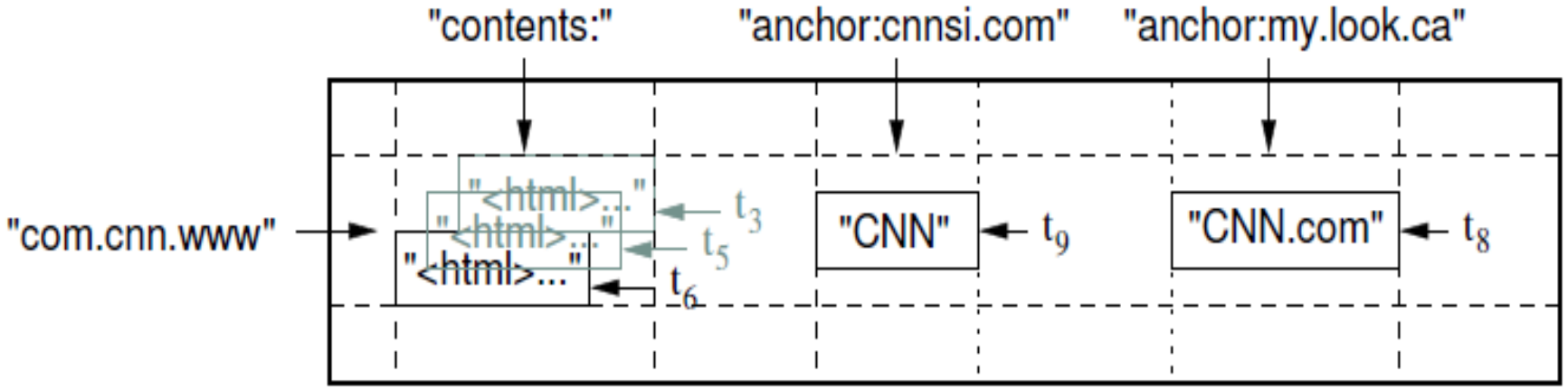
- (zebras, length, 2006) --> 7 ft
- (zebras, weight, 2007) --> 600 lbs
- (zebras, weight, 2006) --> 620 lbs

Timestamp ordering is defined as "most recent appears first"

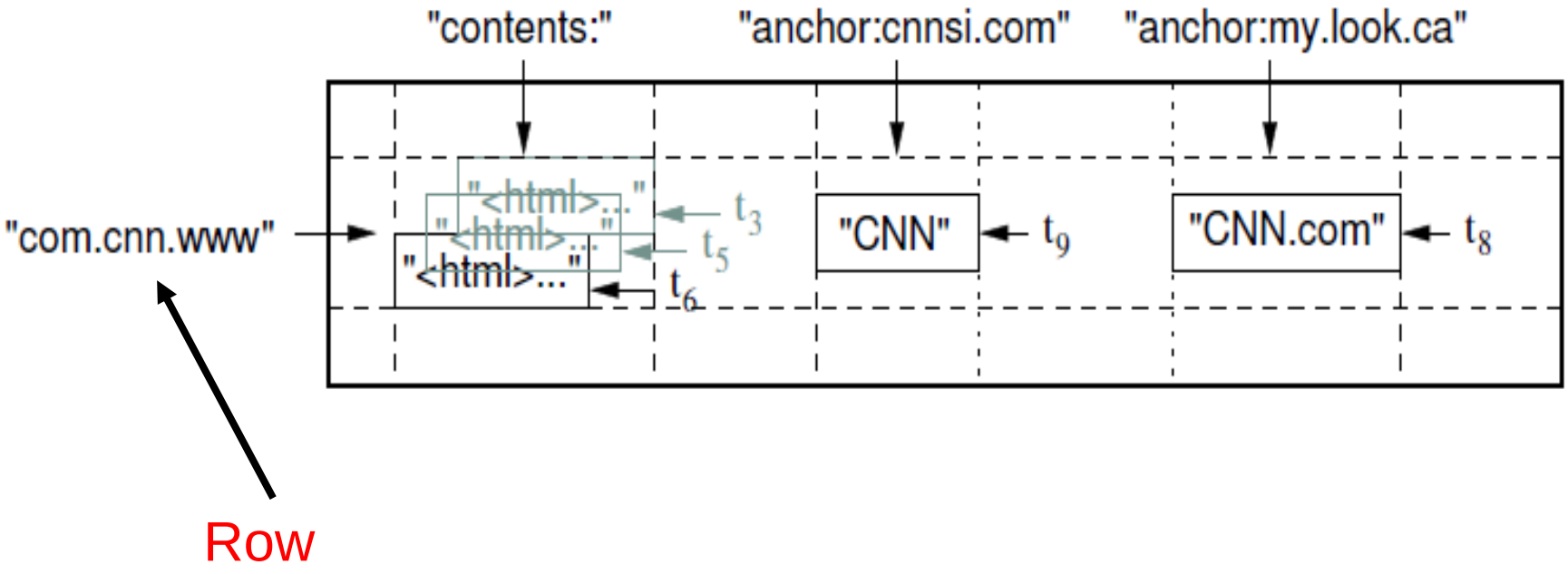
BigTable模型举例

Example: Web Indexing

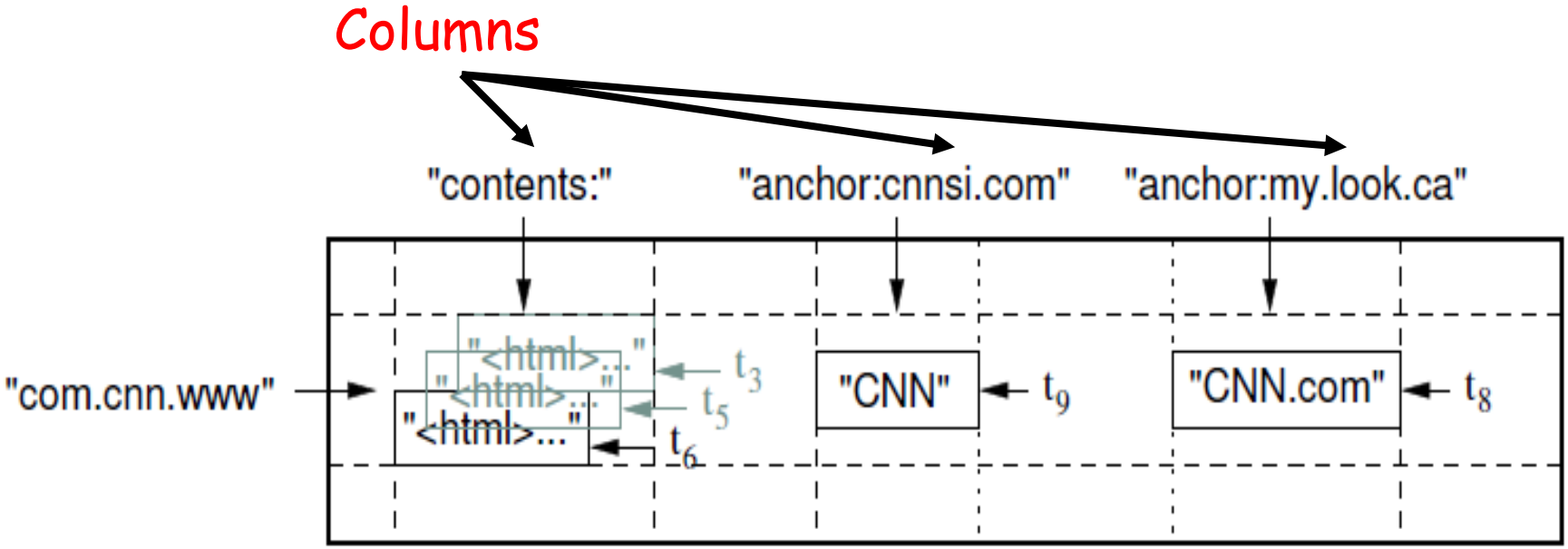
BigTable模型举例



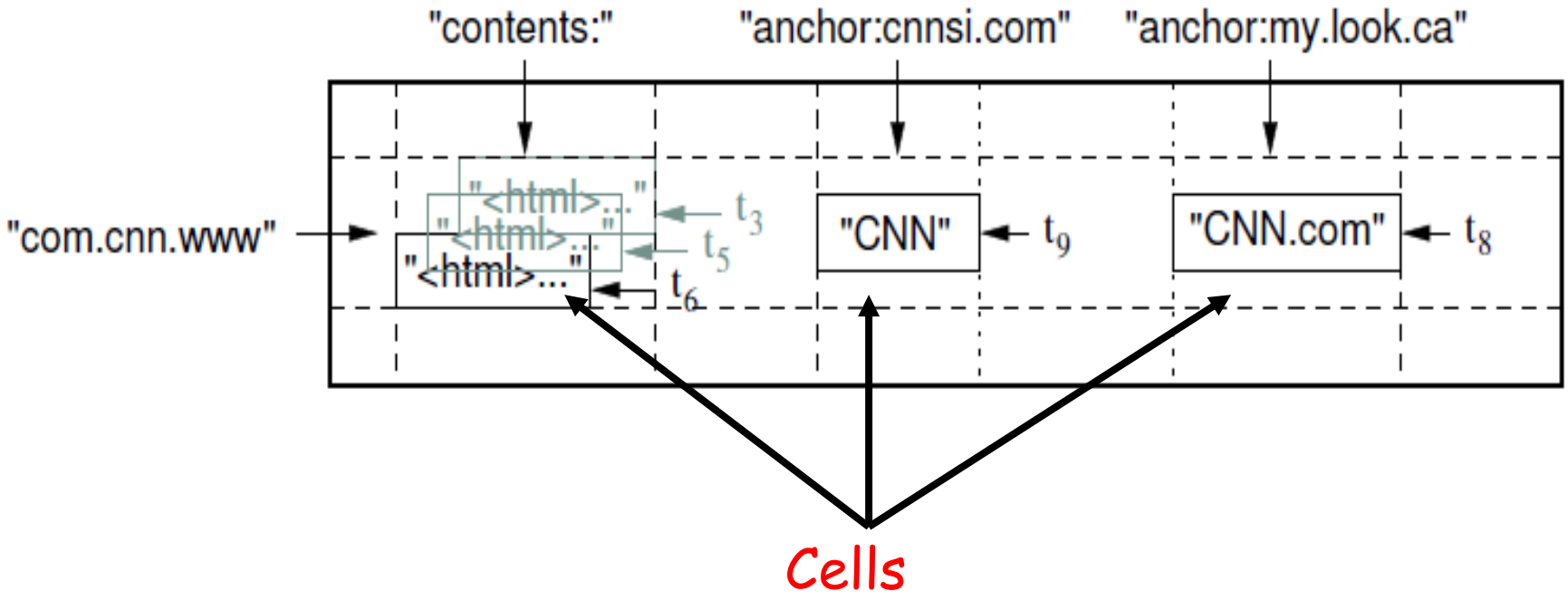
BigTable模型举例



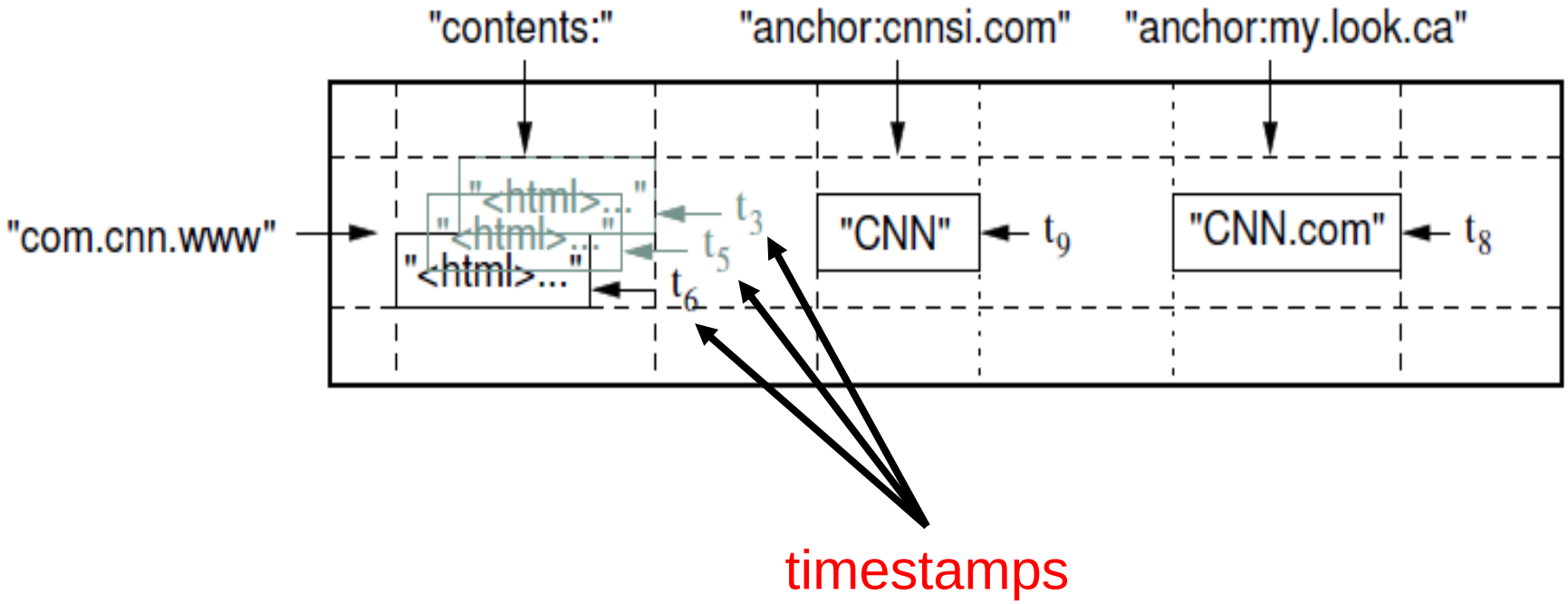
BigTable模型举例



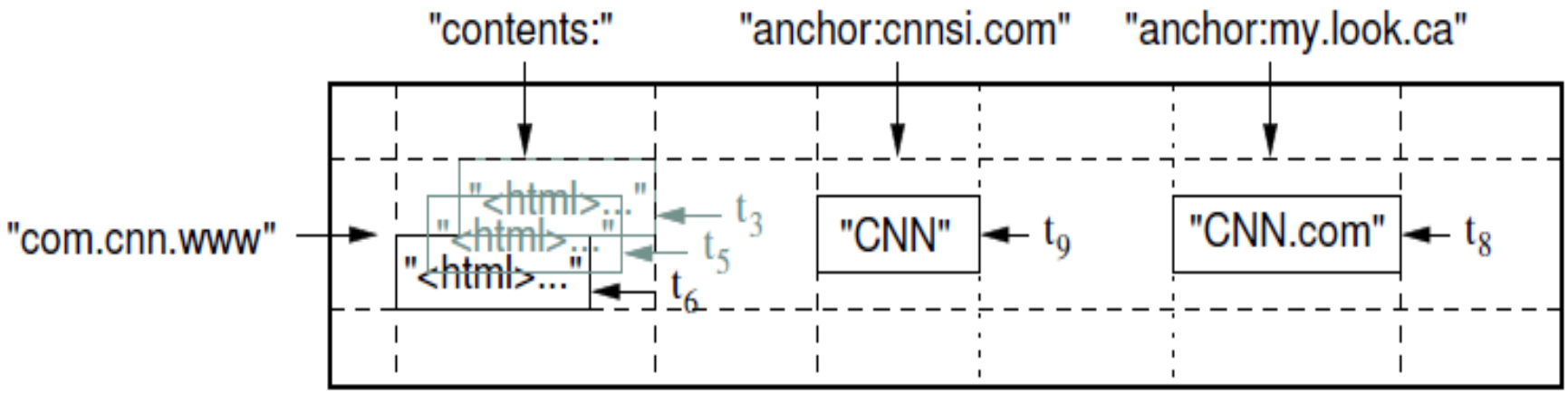
BigTable模型举例



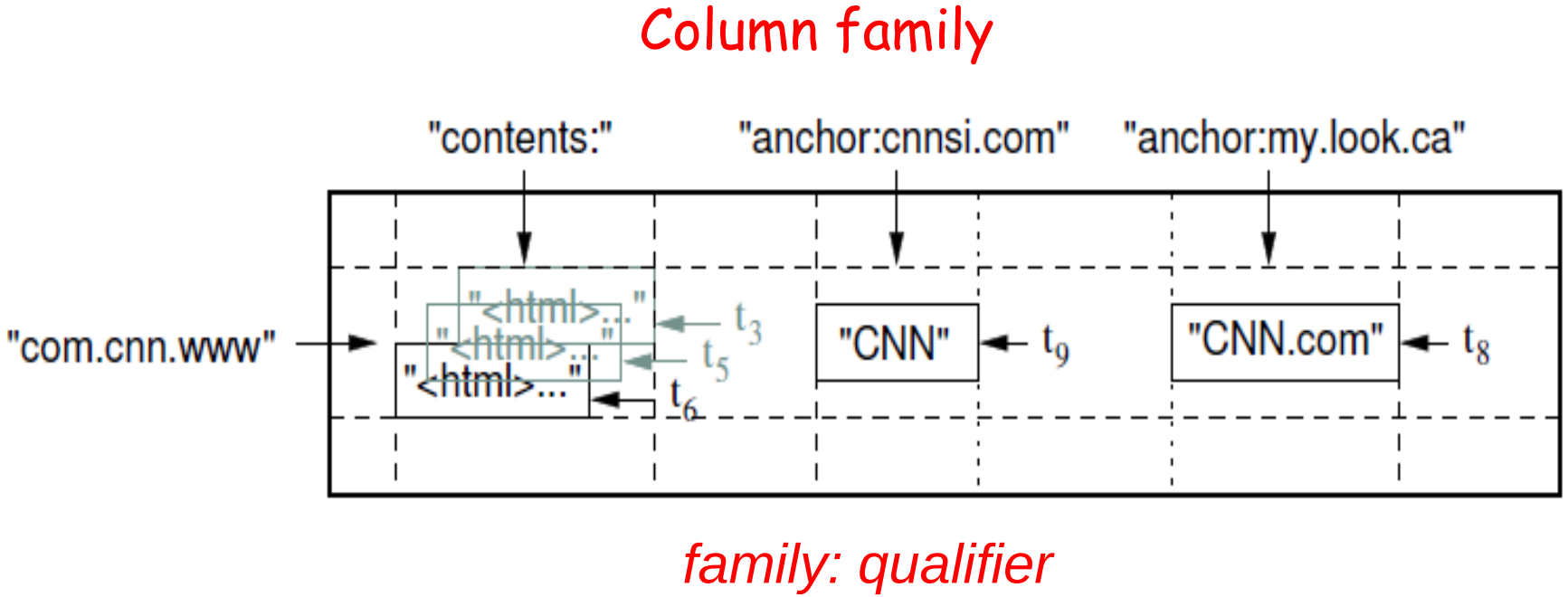
BigTable模型举例



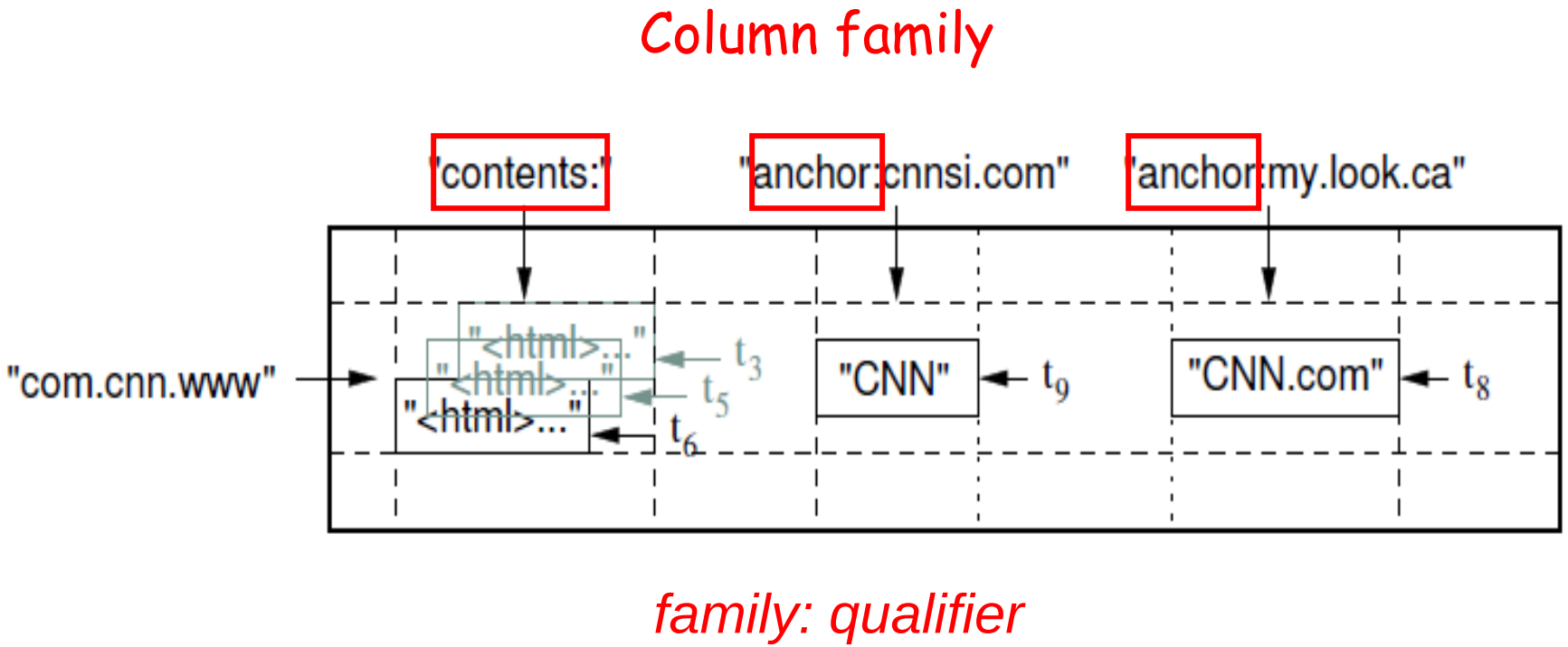
BigTable模型举例



BigTable模型举例



BigTable模型举例



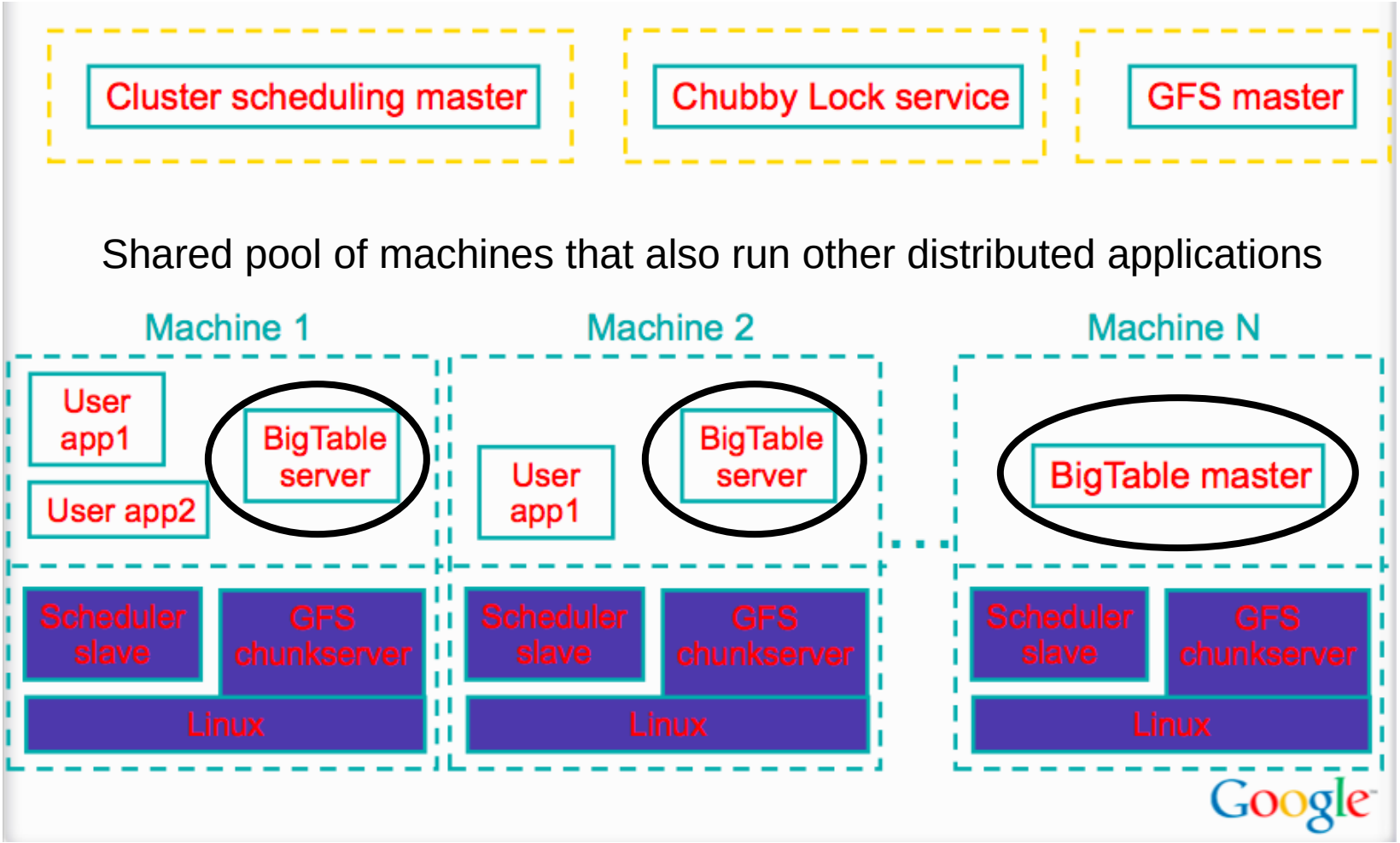
BigTable的物理存储

- Bigtable uses the distributed Google File System (GFS) to store log and data files
  - The Google SSTable file format is used internally to store Bigtable data
  - An SSTable provides a persistent , ordered immutable map from keys to values
    - Operations are provided to look up the value associated with a specified key, and to iterate over all key/value pairs in a specified key range

BigTable的分布式锁管理

- Bigtable relies on a highly-available and persistent distributed lock service called Chubby
  - Chubby provides a namespace that consists of directories and small files. Each directory or file can be used as a lock
    - Consists of 5 active replicas, one replica is the master and serves requests
    - Service is functional when majority of the replicas are running and in communication with one another – when there is a quorum

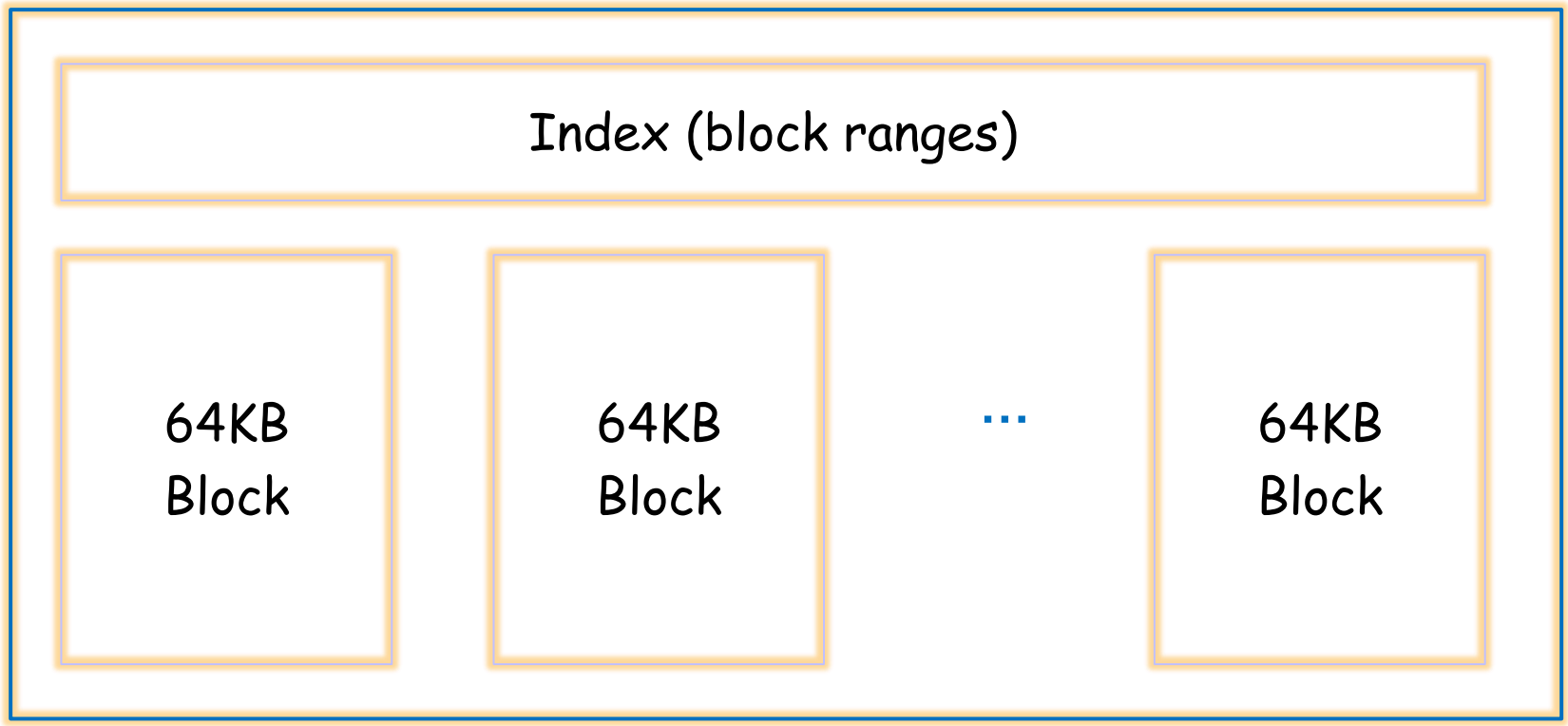
BigTable的实现体系架构



SSTable

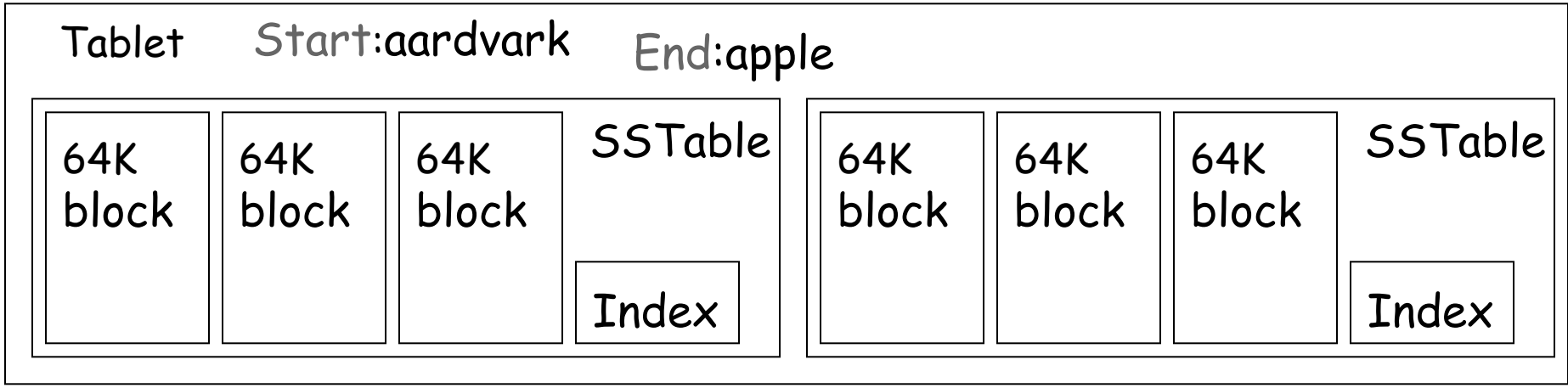
- An SSTable provides a persistent , ordered immutable map from keys to values
- Each SSTable contains a sequence of blocks
- A block index (stored at the end of SSTable) is used to locate blocks
- The index is loaded into memory when the SSTable is open

SSTable的结构



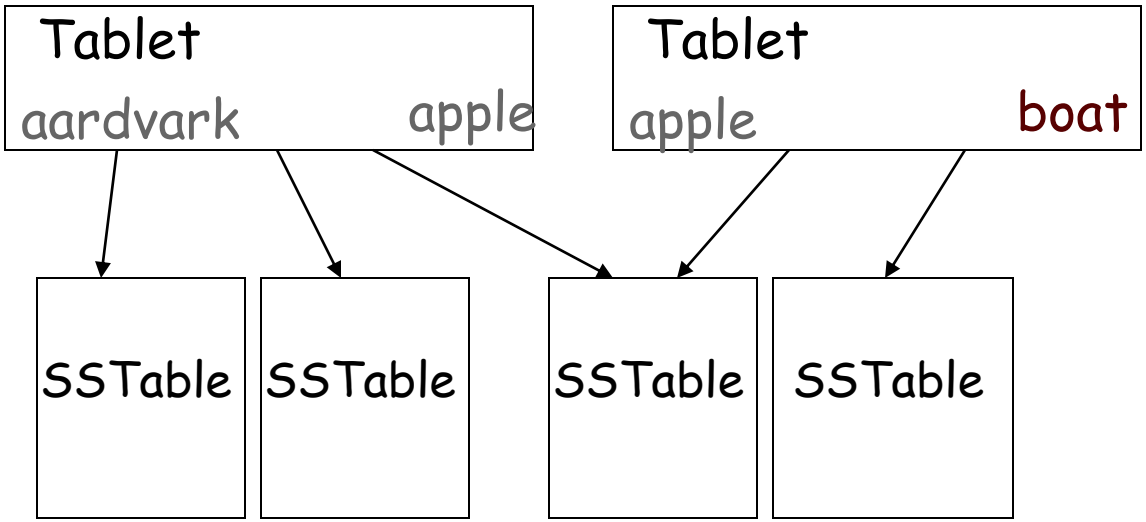
Tablet

- Contains some range of rows of the table
- Built out of multiple SSTables



Table

- Multiple tablets make up the table
- SSTables can be shared
- Tablets do not overlap, SSTables can overlap



Bigtable的组织管理

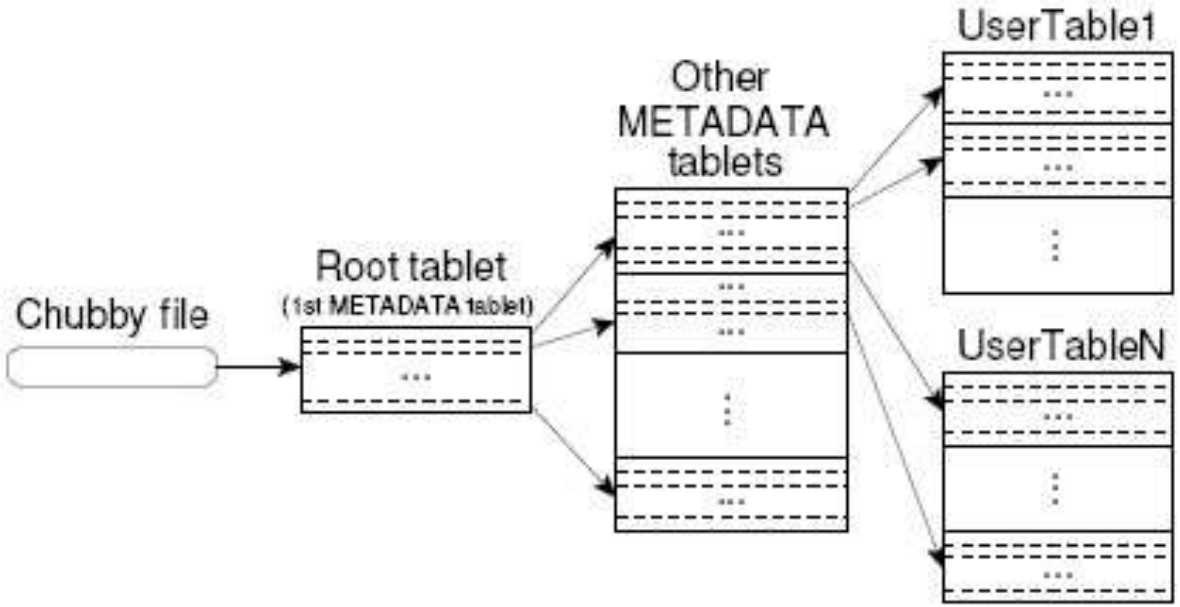
- A Bigtable cluster stores tables
- Each table consists of tablets
  - Initially each table consists of one tablet
  - As a table grows it is automatically split into multiple tablets
- Tablets are assigned to tablet servers
  - Multiple tablets per server. Each tablet is 100-200 MB
  - Each tablet lives at only one server

Bigtable的组织管理

- One master server
  - Assigning tablets to tablet servers
  - Detecting the addition and deletion of tablet servers
  - Balancing tablet-server load
  - Garbage collection of files in GFS
- Many tablet servers
  - Tablet servers manage tablets
  - Tablet server splits tablets that get too big
- Client communicates directly with tablet server for reads/writes.

Tablet Location

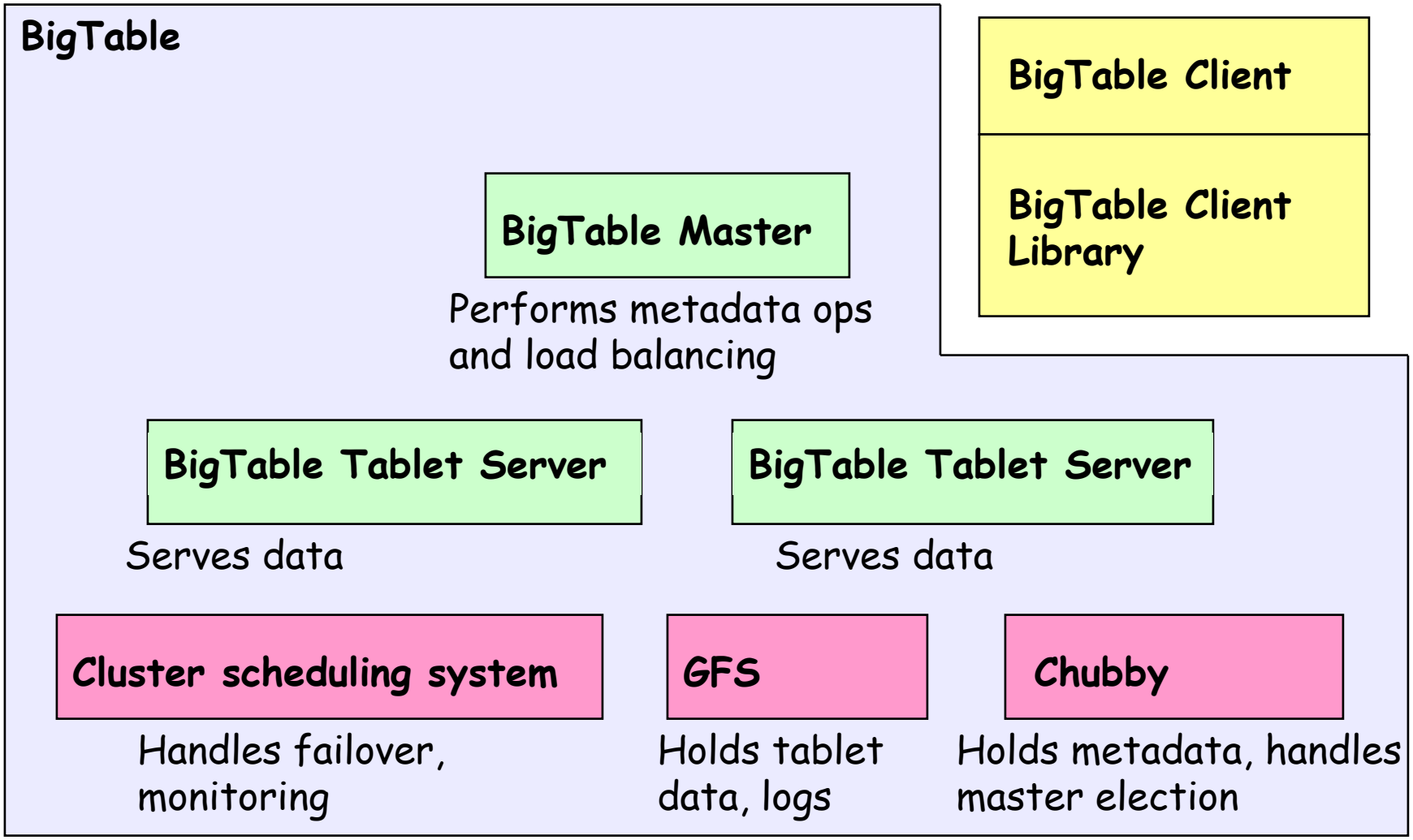
- Since tablets move around from server to server, given a row, how do clients find the right machine?
  - Need to find a tablet whose row range covers the target row



Tablet Location

- A 3-level hierarchy analogous to that of a B+-tree to store tablet location information :
  - A file stored in chubby contains location of the root tablet
  - Root tablet contains location of *Metadata tablets*
    - The root tablet never splits
  - Each meta-data tablet contains the locations of a set of user tablets
- Client reads the Chubby file that points to the root tablet
  - This starts the location process
- Client library caches tablet locations
  - Moves up the hierarchy if location N/A

Tablet 管理体系架构



Tablet Server

- When a tablet server starts, it creates and acquires exclusive lock on, a uniquely-named file in a specific Chubby directory
  - Call this servers directory
- A tablet server stops serving its tablets if it loses its exclusive lock
  - This may happen if there is a network connection failure that causes the tablet server to lose its Chubby session

Tablet Server

- A tablet server will attempt to reacquire an exclusive lock on its file as long as the file still exists
- If the file no longer exists then the tablet server will never be able to serve again
  - Kills itself
  - At some point it can restart; it goes to a pool of unassigned tablet servers

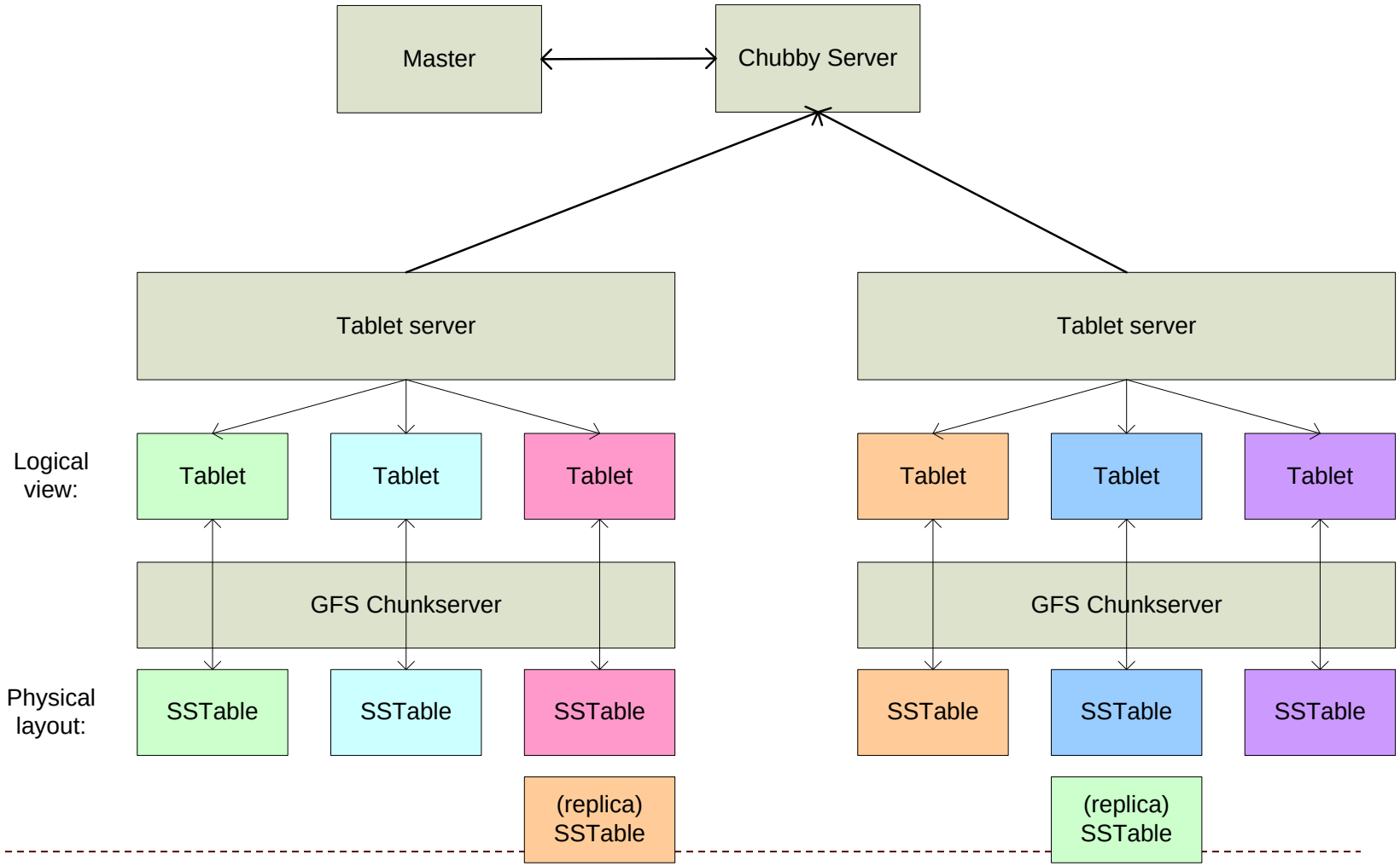
Master Startup Operation

- Upon start up the master needs to discover the current tablet assignment.
  - Grabs unique master lock in Chubby
    - Prevents concurrent master instantiations
  - Scans servers directory in Chubby for live servers
  - Communicates with every live tablet server
    - Discover all tablets
  - Scans METADATA table to learn the set of tablets
    - Unassigned tablets are marked for assignment

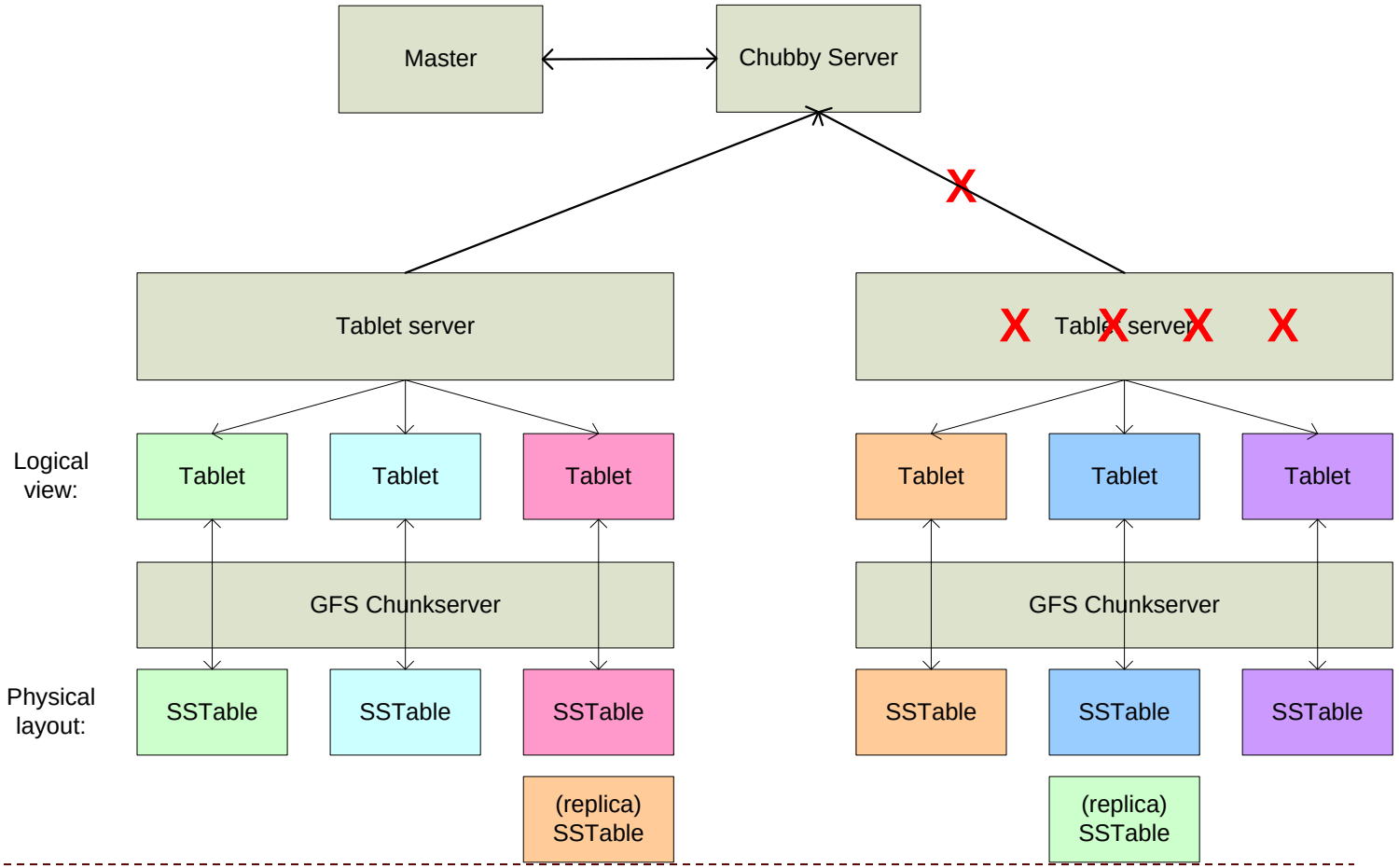
Master Operation

- Detect tablet server failures/resumption
- Master periodically asks each tablet server for the status of its lock

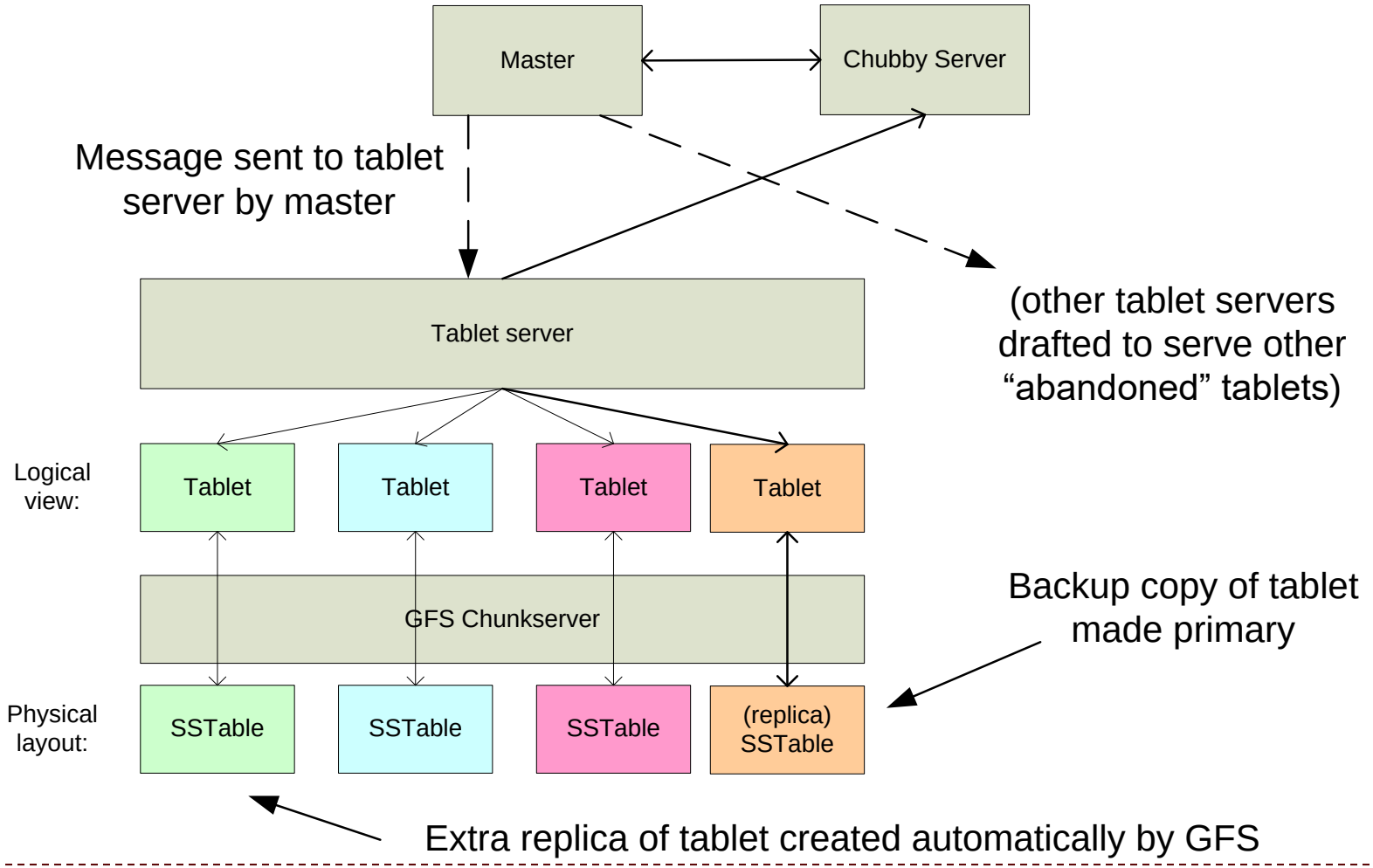
Tablet Server容错



Tablet Server容错



Tablet Server 容错



HBase

- BigTable的开源社区实现
  - 动态表结构(“NoSQL”)
  - 自动管理数据的划分，易负载均衡
  - 高可扩展，高容错
  - 属于Hadoop生态系统，大量的开源工具
- 使用场景举例
  - eBay：Hadoop集群监控日志的存储，BI用户行为日志存储
  - 搜狐视频：存储用户的行为数据，作为各类数据挖掘和推荐算法的数据基础
  - FaceBook：Message存储请求系统
- 特点
  - 高写入
  - 查询模式相对简单
  - 数据海量（搜狐1分钟1million）

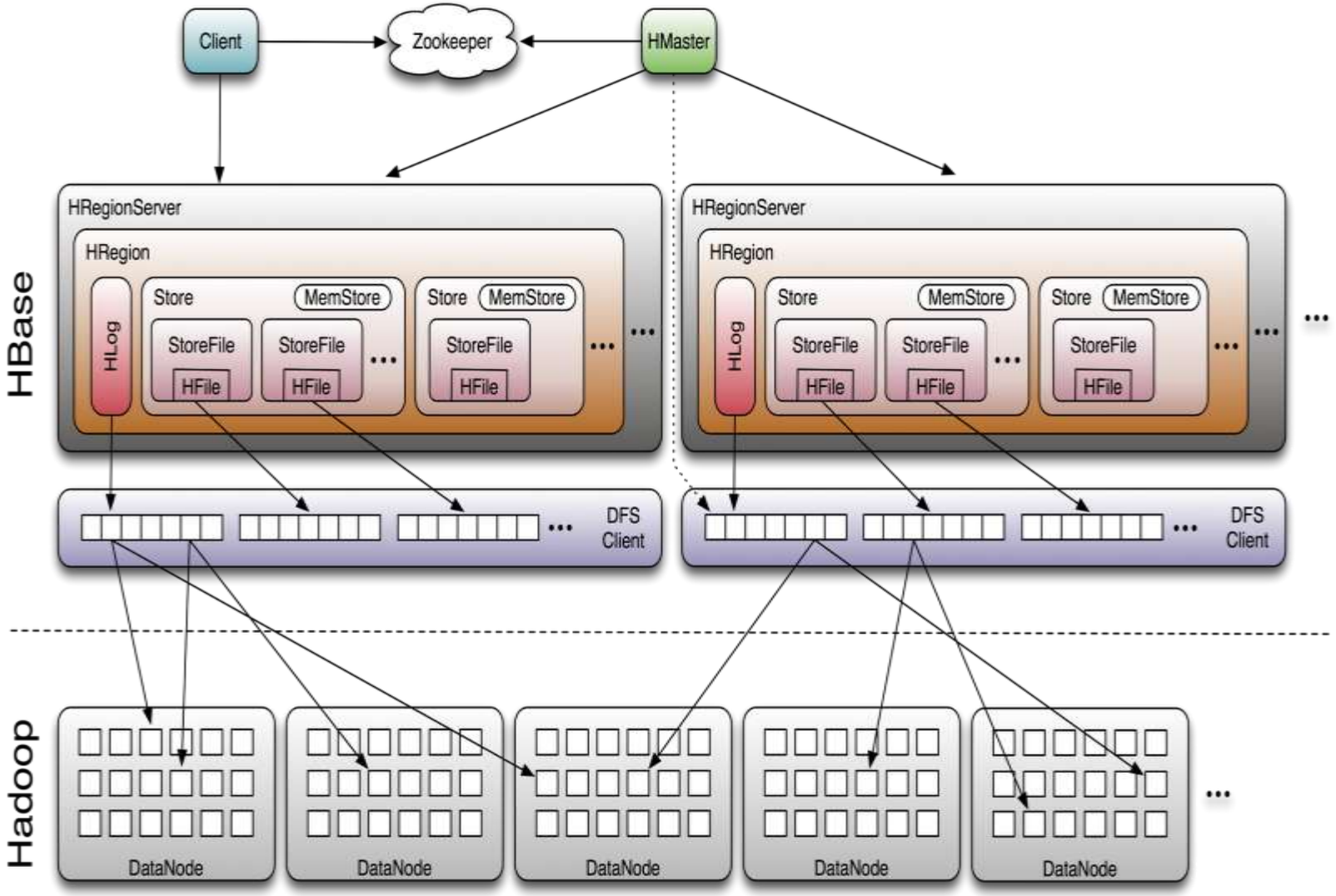
HBase与BigTable的对应关系

Hbase	BigTable
RegionServer	TabletServer
Region	Tablet
HFile	sstable
Memstore	Memtable
Zookeeper	Chubby
HDFS	GFS

Hbase的组成

- *Master*
  - Responsible for monitoring region servers
  - Load balancing for regions
  - Redirect client to correct region servers
- *regionserver* slaves
  - Serving requests(Write/Read/Scan) of Client
  - Send HeartBeat to Master
  - Throughput and Region numbers are scalable by region servers

Hbase体系架构


Client
Zookeeper
HMaster
HBase
HRegionServer
HRegion
HLog
Store
MemStore
StoreFile
HFile
DFS Client
Hadoop
DataNode

Hbase使用场景

- 适合
  - 存储海量数据，对可扩展性要求较高
  - 高效的随机访问性能
  - 高效的写性能
  - 一次写入多次读取的场景
- 不适合
  - 不适合read-modify-write的请求模式
  - 不适合复杂的查询请求（NoSQL）

Hbase使用

- 接口
  - Java API
  - REST/HTTP
  - Apache Thrift（任何编程语言）
  - Hive/Pig 用于数据分析
- 功能
  - Get(row)
  - Put（row，Map<列，列值>）
  - Scan（主键范围，filter）
  - CheckAndPut，Delete等等
  - MapReduce/Hive

Hbase是什么

A sparse, multi-dimensional, sorted map

逻辑结构：稀疏的，多维且有序的Map结构

主键	文章信息		作者信息		
1	内容	Hbase是基于Hadoop的分布式数据库	ID	linfly	
	标签	Hbase,Hadoop,NoSQL	昵称	T1	小熊
	标题	Hbase技术分享	昵称	T2	大雄
2	内容	MongoDB是最接近SQL的NoSQL文档数据库	ID	wenyu	
	标题	MongoDB技术分享			

Hbase是什么

A sparse, multi-dimensional, sorted map

逻辑结构：稀疏的，多维且有序的Map结构

主键	文章信息		作者信息		
1 第一行	内容	Hbase是基于Hadoop的分布式数据库	ID	linfly	
	标签	Hbase,Hadoop,NoSQL	昵称	T1	小熊
	标题	Hbase技术分享	昵称	T2	大雄
2	内容	MongoDB是最接近SQL的NoSQL文档数据库	ID	wenyu	
	标题	MongoDB技术分享			

Hbase是什么

A sparse, multi-dimensional, sorted map

逻辑结构：稀疏的，多维且有序的Map结构

主键	文章信息		作者信息		
1	内容	Hbase是基于Hadoop的分布式数据库	ID	linfly	
	标签	Hbase,Hadoop,NoSQL	昵称	T1	小熊
	标题	Hbase技术分享	昵称	T2	大雄
2 第二行	内容	MongoDB是最接近SQL的NoSQL文档数据库	ID	wenyu	
	标题	MongoDB技术分享			

Hbase是什么

A sparse, multi-dimensional, sorted map

逻辑结构：稀疏的，多维且有序的Map结构

主键	文章信息列族		作者信息列族		
1	内容	Hbase是基于Hadoop的分布式数据库	ID	linfly	
	标签	Hbase,Hadoop,NoSQL	昵称	T1	小熊
	标题	Hbase技术分享	昵称	T2	大雄
2	内容	MongoDB是最接近SQL的NoSQL文档数据库	ID	wenyu	
	标题	MongoDB技术分享			

Hbase是什么

A sparse, multi-dimensional, sorted map

逻辑结构：稀疏的，多维且有序的Map结构

主键	文章信息		作者信息		
1	内容	Hbase是基于Hadoop的分布式数据库	ID	linfly	
	标签	Hbase,Hadoop,NoSQL	昵称	T1	小熊
	标题	Hbase技术分享	昵称	T2	大雄
2 第二行	内容	MongoDB是最接近SQL的NoSQL文档数据库	ID	wenyu	
	标题	MongoDB技术分享			

Hbase是什么

A sparse, multi-dimensional, sorted map

逻辑结构：稀疏的，多维且有序的Map结构

主键	文章信息		作者信息		
1	内容	Hbase是基于Hadoop的分布式数据库	ID	linfly	
	标签	Hbase,Hadoop,NoSQL	昵称	T1	小熊
	标题	Hbase技术分享	昵称	T2	大雄
2	内容	MongoDB是最接近SQL的NoSQL文档数据库	ID	wenyu	
	标题	MongoDB技术分享			

时间戳

Hbase是什么

A sparse, multi-dimensional, sorted map

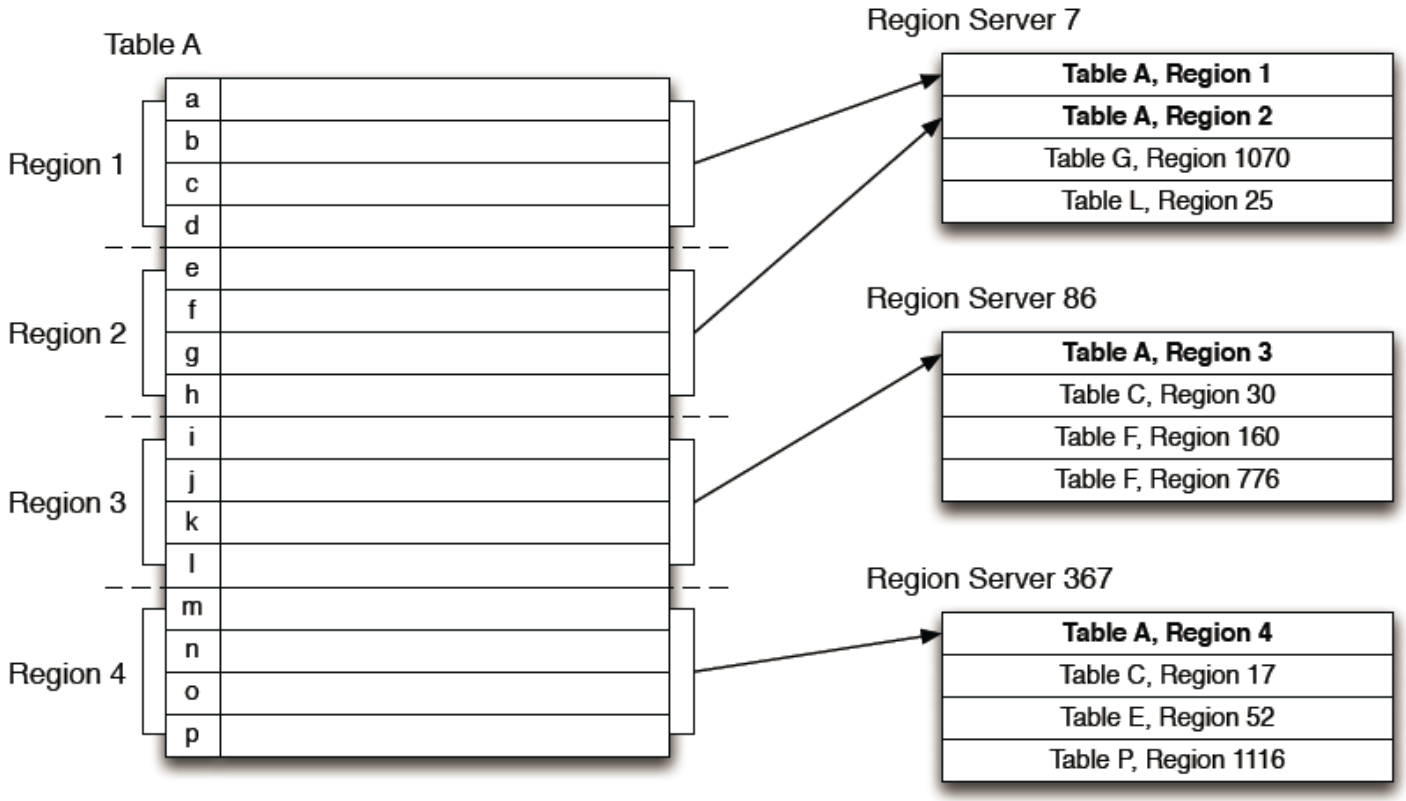
逻辑结构：稀疏的，多维且有序的Map结构

主键	文章信息		作者信息		
1	内容	Hbase是基于Hadoop的分布式数据库	ID	linfly	
	标签	Hbase,Hadoop,NoSQL	昵称	T1	小熊
	标题	Hbase技术分享	昵称	T2	大雄
2	内容	MongoDB是最接近SQL的NoSQL文档数据库	ID	wenyu	
	标题	MongoDB技术分享			

(表明, 主键, 列族名, 列名, 时间戳) —>列值
(blog, 1, 作者信息，昵称， T1) —>小熊
(blog, 1, 作者信息，昵称， T2) —>大雄

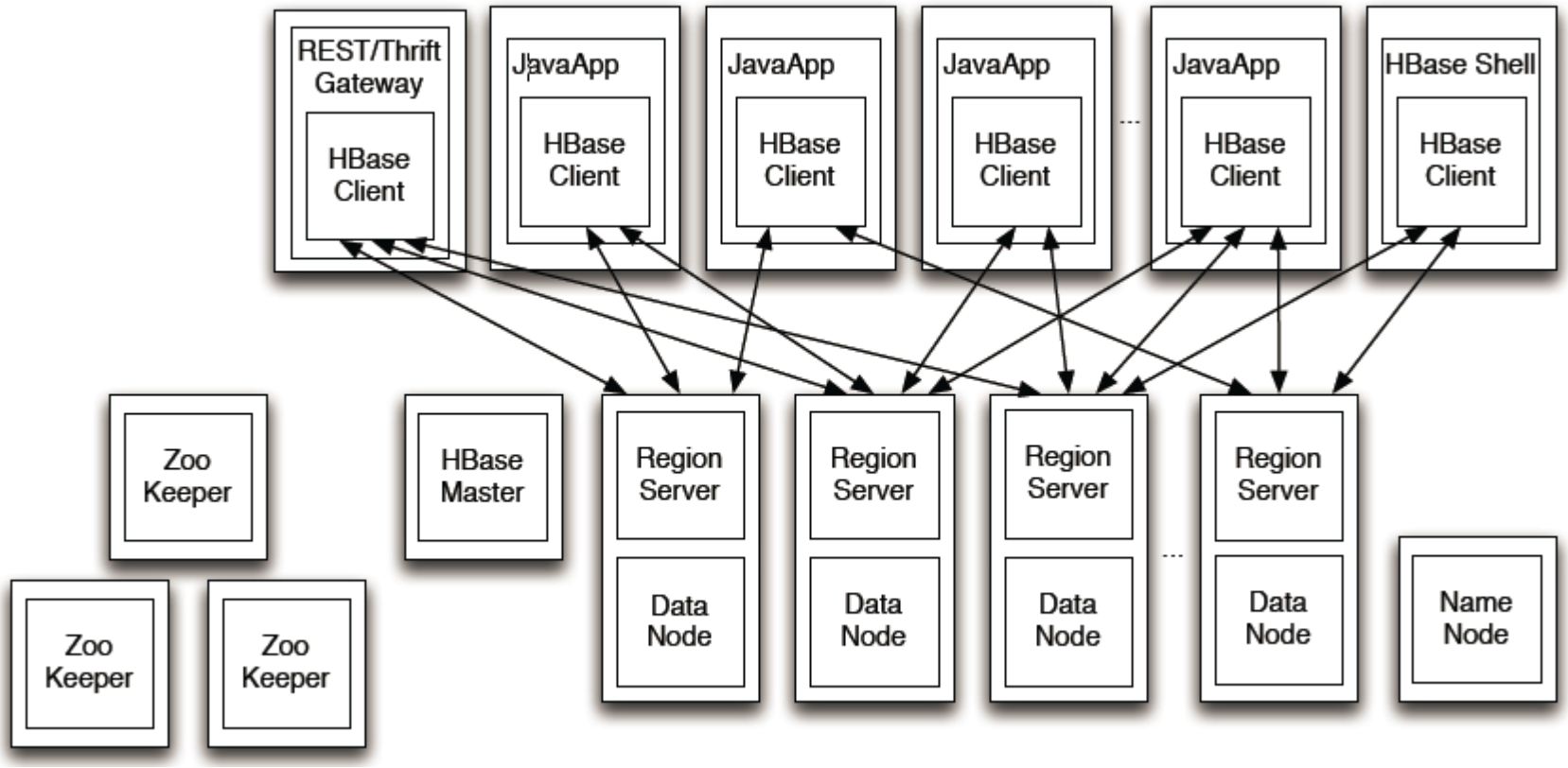
KeyValue结构

系统架构-Hbase集群逻辑结构



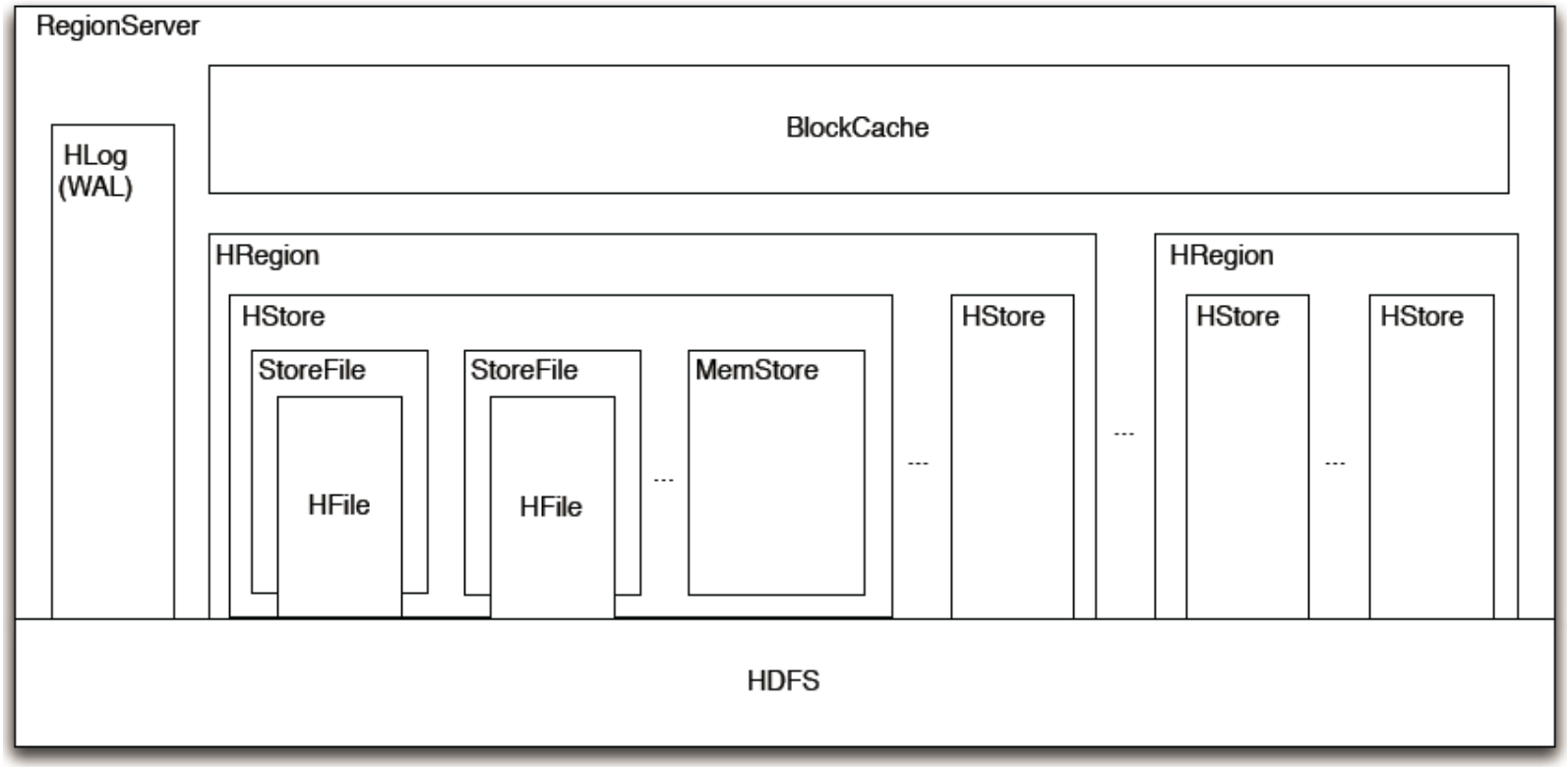
- 一个表会被划分成很多regions，每个region大小相当
- Region会被分配给region server，region server负载处理client的数据读写请求
- 每个region server所拥有的region数目大致相当

系统架构-Hbase集群物理结构



HBase的Region Server跟HDFS的DataNode是一对一的关系
HBase的客户端读写数据时直接与RegionServer通信
HMaster负责Region管理，例如新建，split等
在进行数据传输的时候HMaster和Zookeeper不会参与

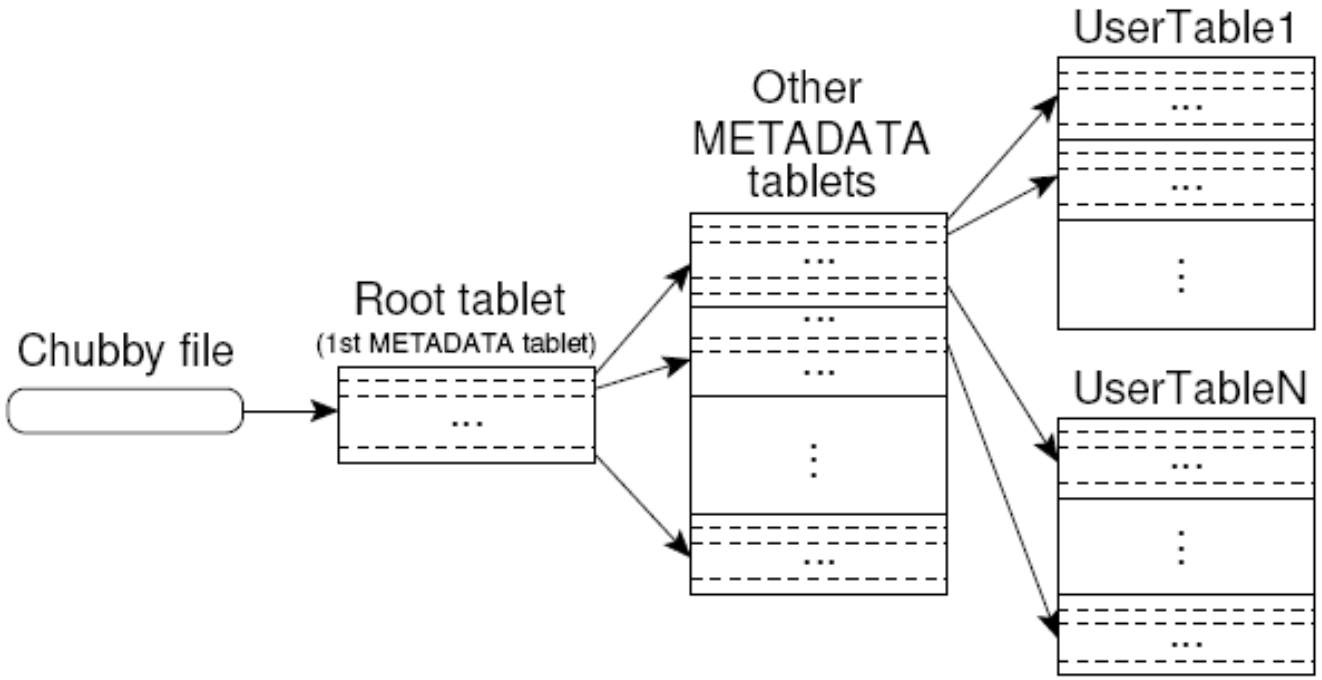
系统架构-Hbase集群物理结构



- 一个RegionServer包含了一个WAL，一个Block Cache以及多个Regions
- 一个Region包含多个HStore，每个HStore对应表中的一个column family
- 一个Hstore包含多个StoreFile和一个MemStore
- 一个StoreFile对应一个Hfile
- HFile和WAL是持久化存在HDFS中的

Region/Tablet的定位

使用三级目录树结构，类似于B+树，但是树固定只有三层



Tablet/Region的管理

- 分配，通过Hbase-Master与Zookeeper的配合
- Region的恢复：region的存的只是索引，真正数据在HDFS，恢复快，通过Region的备份或者Commit Log恢复
- Compaction
  - Minor：将内存的memstore持久化到disk中
  - Major：将多个Hfile/SSTable合并为一个

谢谢！