Apache Hbase

Viet-Trung Tran

School of Information and Communication Technology

Outline

- Introduction
- Why use Hbase?
- What is Hbase?
- Hbase Data Model
- Hbase Architecture
- Acid properties in hbase
- Hbase vs. RDBMS
- Hbase vs. HDFS

Introduction

- HBase is developed as part of **Apache Software Foundation's Apache Hadoop project** and runs on top of **HDFS** (Hadoop Distributed Filesystem), providing **BigTable**-like capabilities for Hadoop.
- Open-source implementation of Google's BigTable
  - Lots of semi-structured data
  - Commodity Hardware
  - Horizontal Scalability
  - Tight integration with MapReduce
  - Hbase is a part in Hadoop ecosystem

History

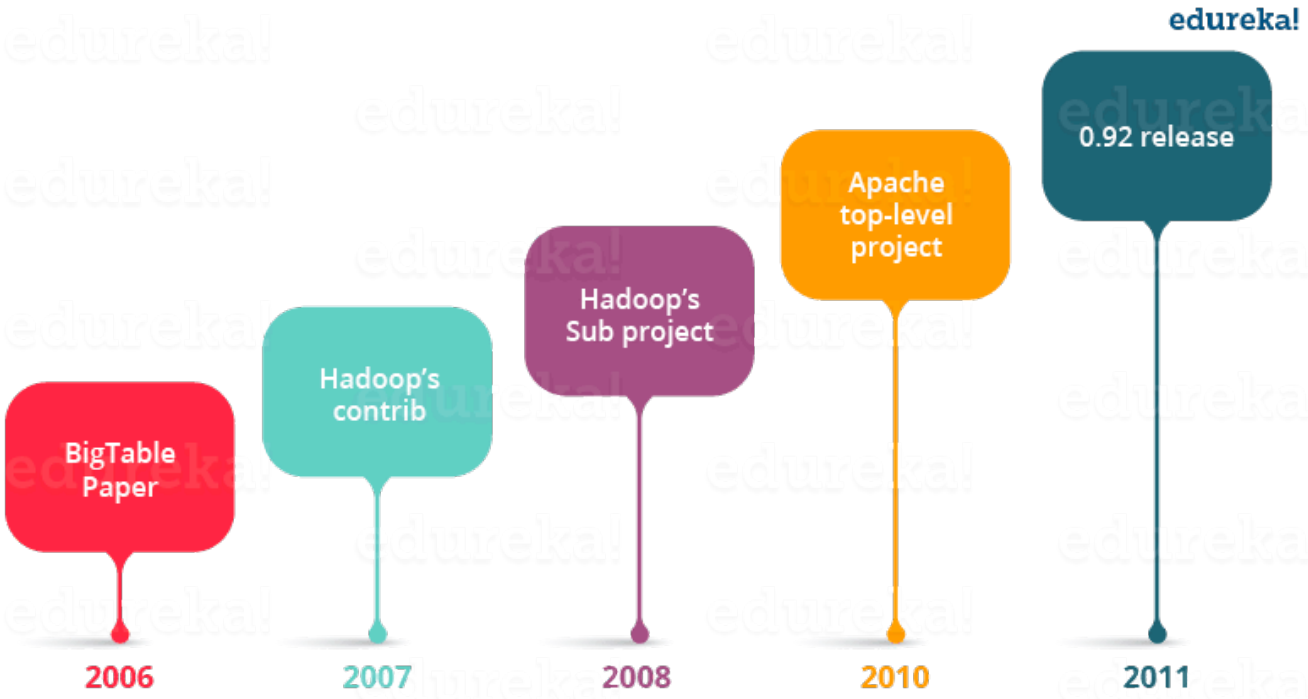
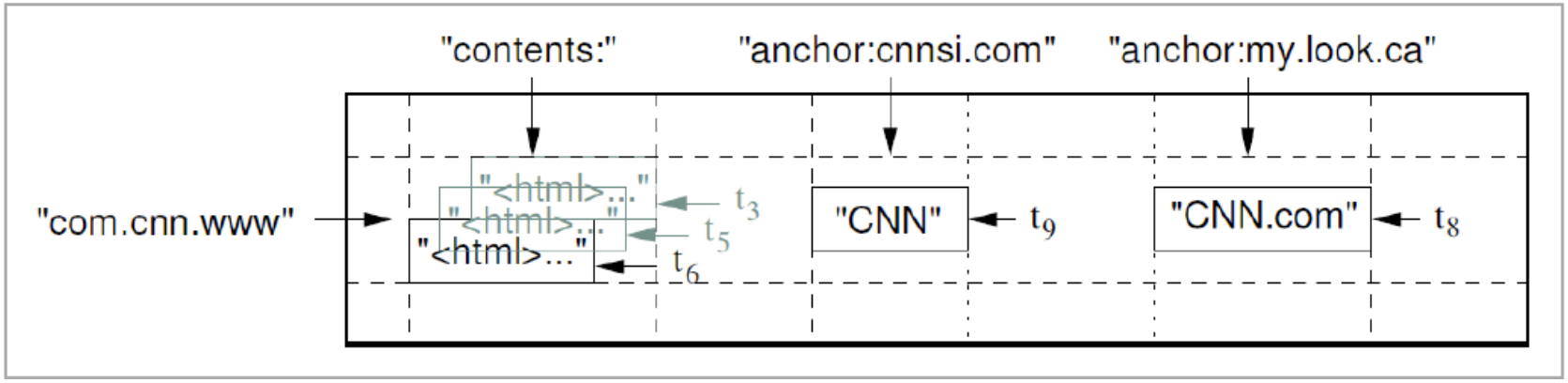


Figure: History of HBase

Hbase data model

- a “sparse, distributed, persistent multidimensional sorted map.”
  - {row, column, timestamp} -> cell
  - Column = Column Family: Column Qualifier
  - Rows are sorted lexicographically based on row key
  - Region: contiguous set of sorted rows
- HBase: a large number of columns, a low number of column families

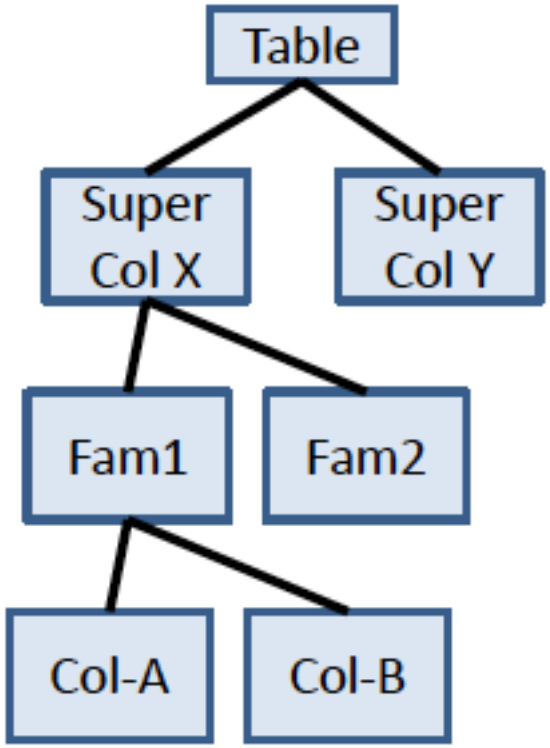


Sparse, distributed, persistent properties

- Spare
  - Sparse data is supported with no waste of costly storage space
  - Rows can be empty or NULL
  - New fields can be added dynamically without having to redesign the schema or disrupt operations
  - HBase as a schema-less data store
- Persistent
  - HBase leverages HDFS to persist its data to disk storage.
- Distributed
  - Hbase supports partitioning

Hbase column family

- HBase data stores consist of one or more tables, which are indexed by row keys.
  - Data versioning for rows is implemented with time stamps
- Columns are grouped into column families
  - Column families must be defined up front during table creation
- Column families are grouped together on disk
  - Grouping data with similar access patterns reduces overall disk access and increases performance



Column family example

Table 12-2 Logical View of Customer Contact Information in HBase

Row Key	*Column Family: {Column Qualifier:Version:Value}*
00001	CustomerName: {'FN': 1383859182496:'John', 'LN': 1383859182858:'Smith', 'MN': 1383859183001:'Timothy', 'MN': 1383859182915:'T'} ContactInfo: {'EA': 1383859183030:'John.Smith@xyz.com', 'SA': 1383859183073:'1 Hadoop Lane, NY 11111'}
00002	CustomerName: {'FN': 1383859183103:'Jane', 'LN': 1383859183163:'Doe', ContactInfo: { 'SA': 1383859185577:'7 HBase Ave, CA 22222'}

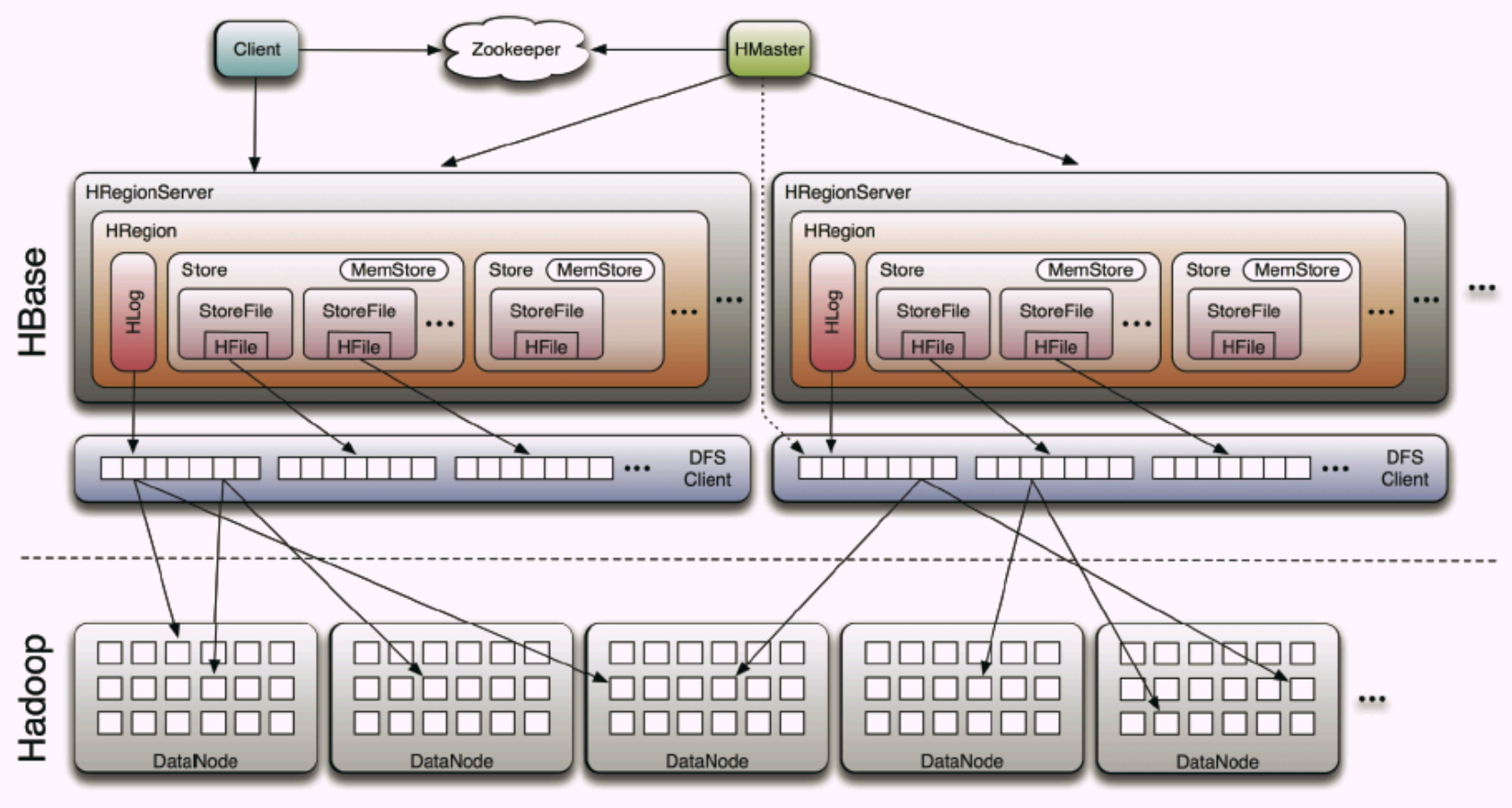
Hbase column

- Column qualifiers are specific names assigned to data values
- Column qualifiers can be virtually unlimited in content, length and number.
  - New data can be added to column families on the fly, making HBase flexible and highly scalable
- HBase stores the column qualifier within the value
  - long column qualifiers can be quite costly in terms of storage
- Values stored in HBase are time stamped by default
  - The versioned data is stored in decreasing order, so that the most recent value is returned by default unless a query specifies a particular timestamp

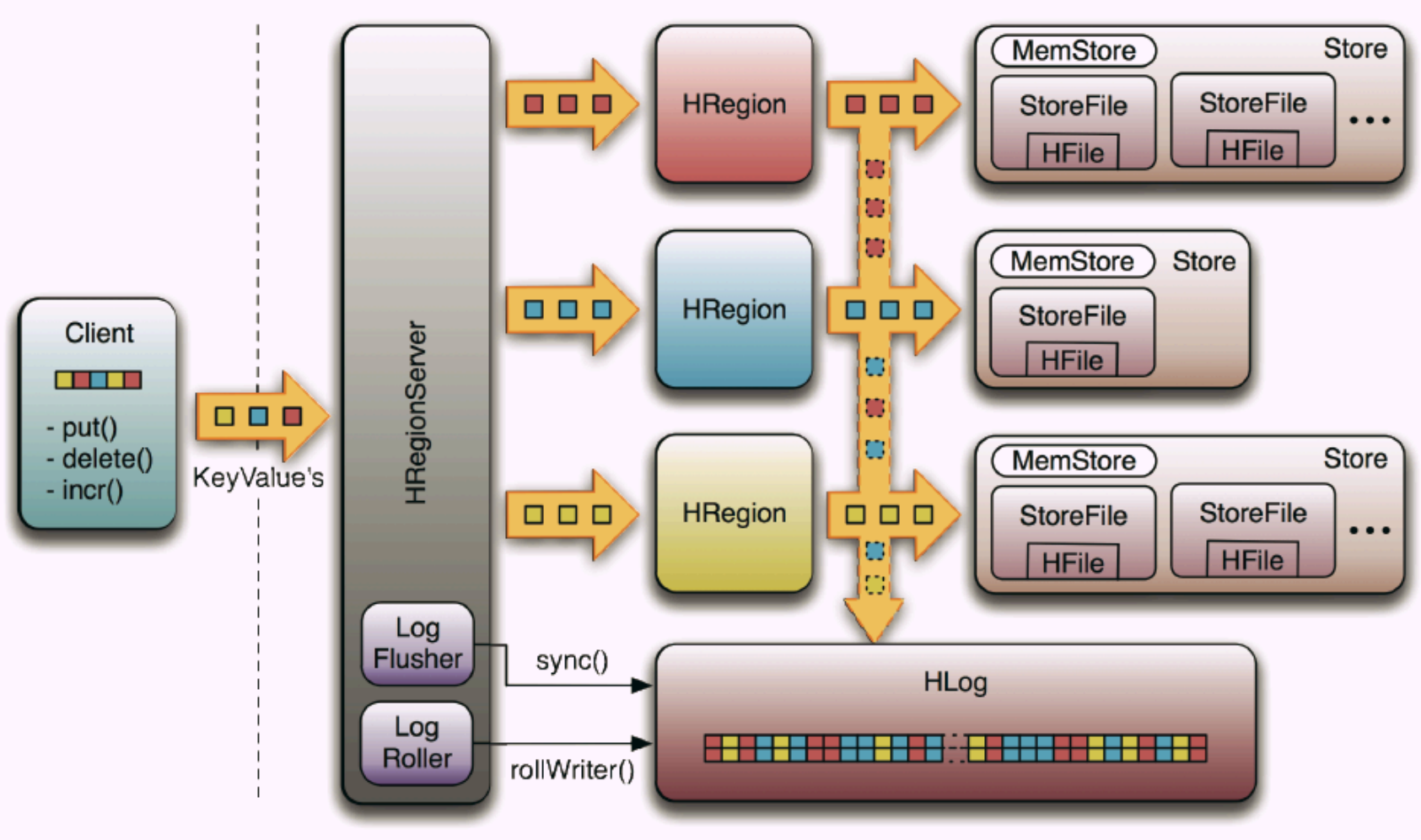
Hbase operators

- Operations are based on row keys
- Single-row operations
  - Put
  - Get
  - Scan
- Multi-row operations
  - Scan
  - MultiPut
- No built-in joins (leverage MapReduce)

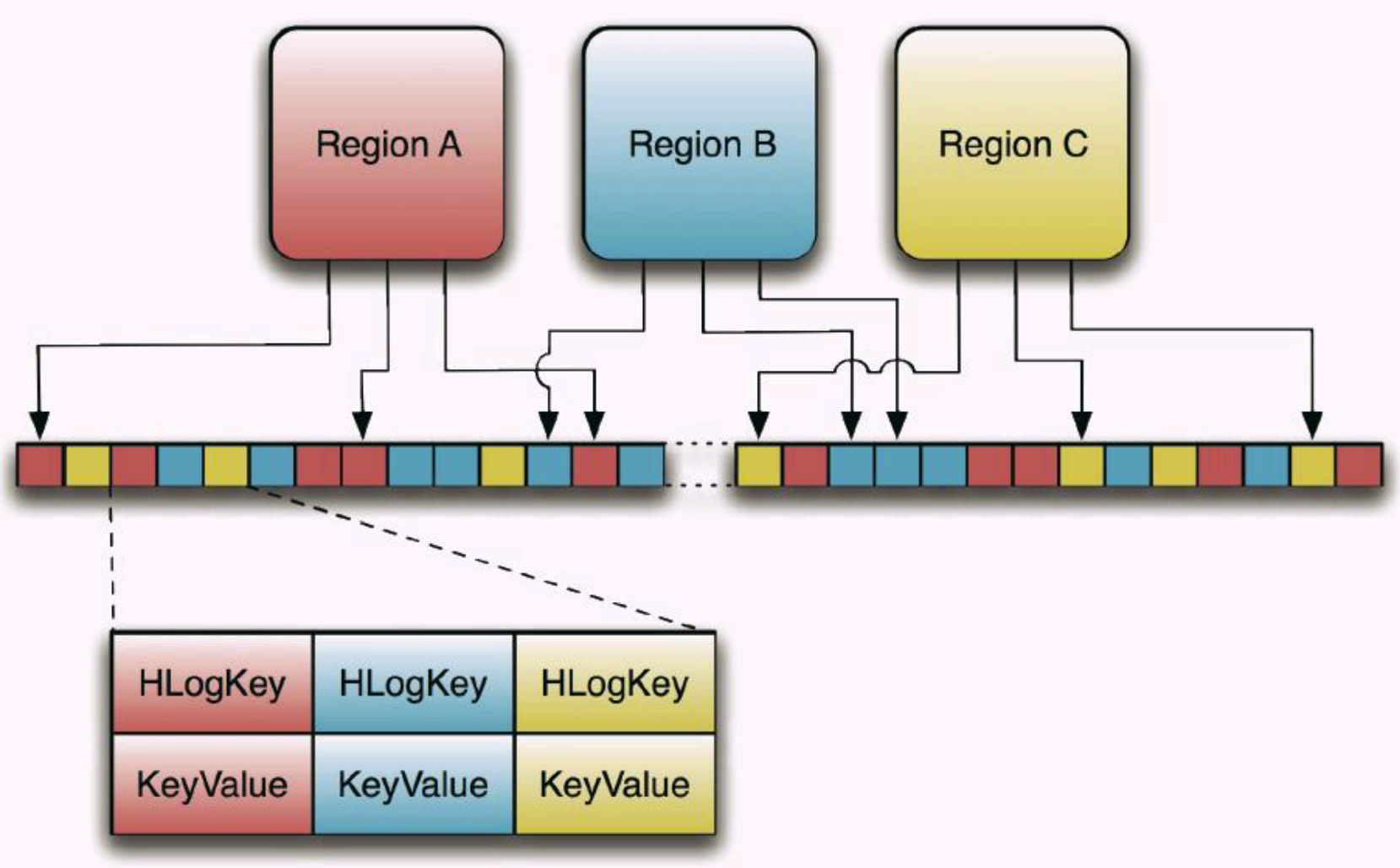
Hbase architecture



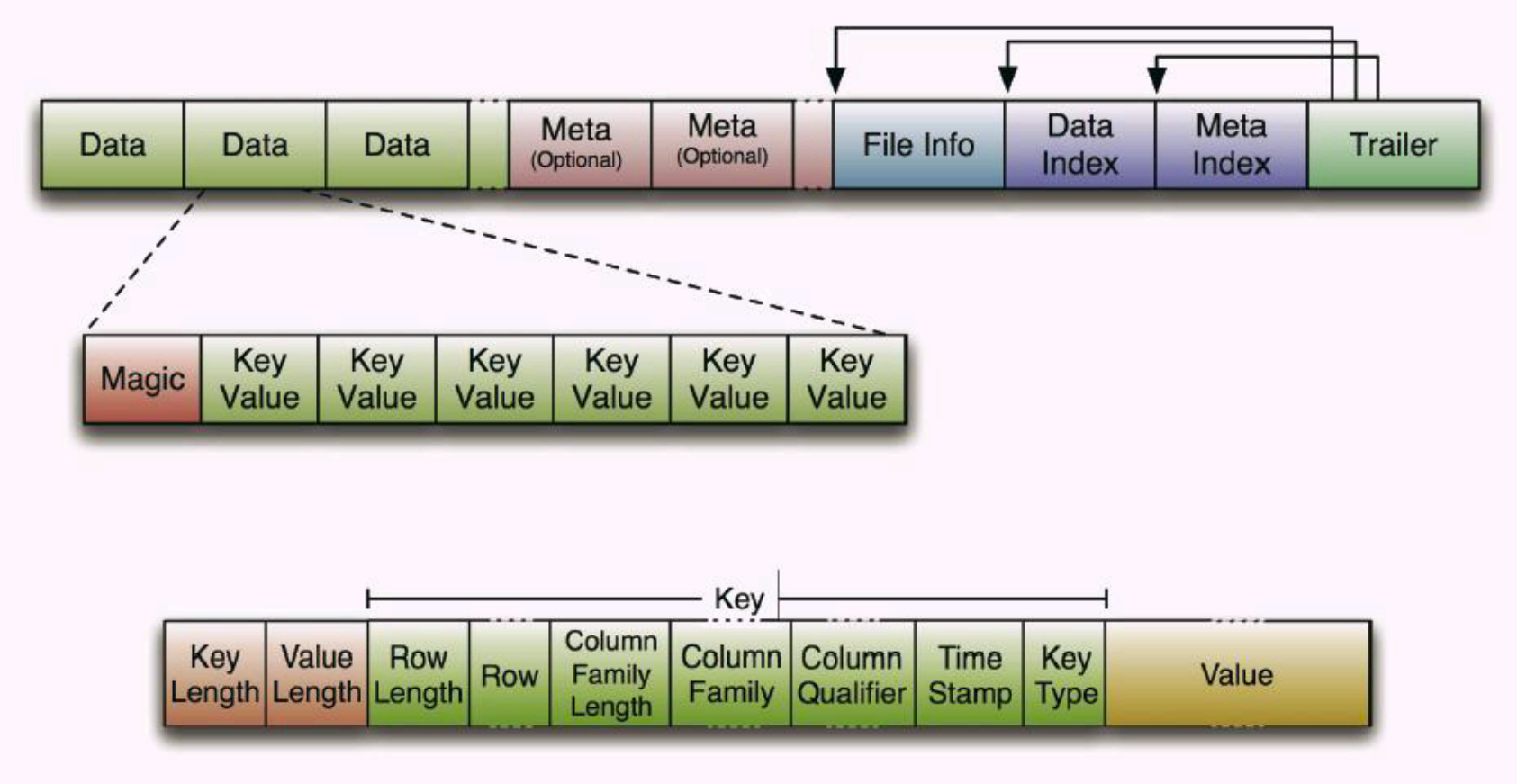
Architecture: Write-Ahead-Log flow



Write-Ahead-Log flow



HFile and Key/Value

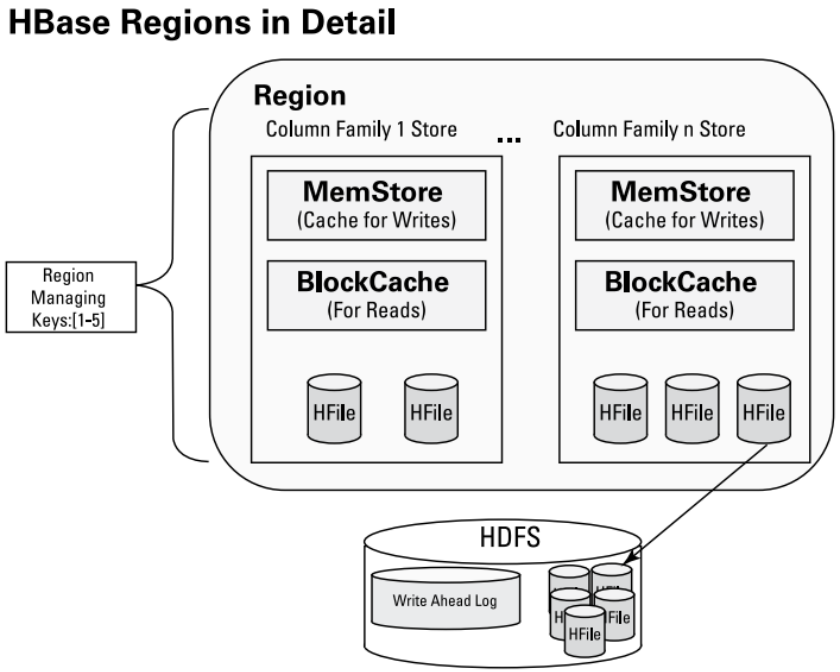


Hbase region servers

- RegionServers store and retrieve data in HBase.
  - Each RegionServer is deployed on a dedicated compute node
- Auto-sharding
  - Upon a table grows beyond a configurable limit, the table is splitted and distributes the load to another RegionServer
- Regions
  - As tables are split, the splits become regions
  - Regions store a range of key-value pairs, and each RegionServer manages a configurable number of regions

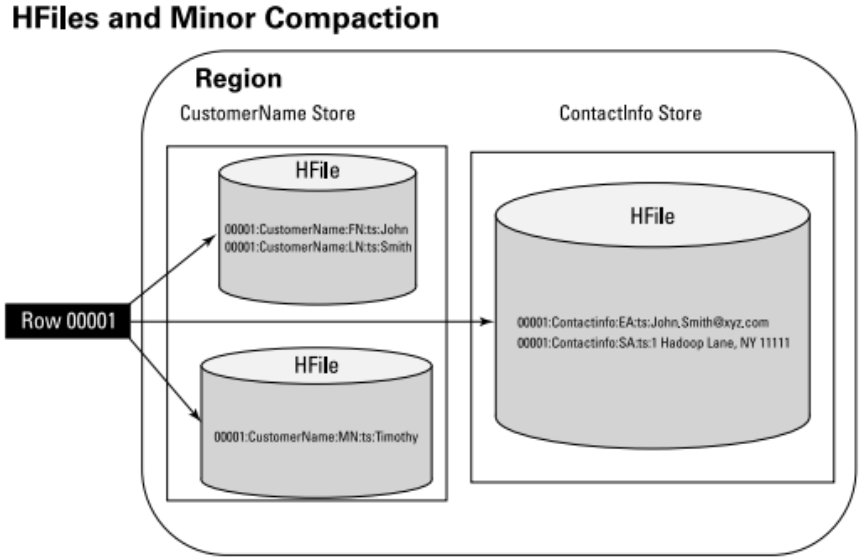
Hbase region servers

- Each column family store object has a read cache called the BlockCache and a write cache called the MemStore.
- Hbase flushs column family data stored in the MemStore to one HFile per flush



Hbase compaction

- Minor compaction
  - At configurable intervals, HFiles are combined into larger HFiles
  - Important because reading a particular row can require many disk reads and cause slow overall performance.
- Major compaction
  - Combine *all* HFiles into one large Hfile
  - A major compaction does the cleanup work after a user deletes a record.



Hbase master

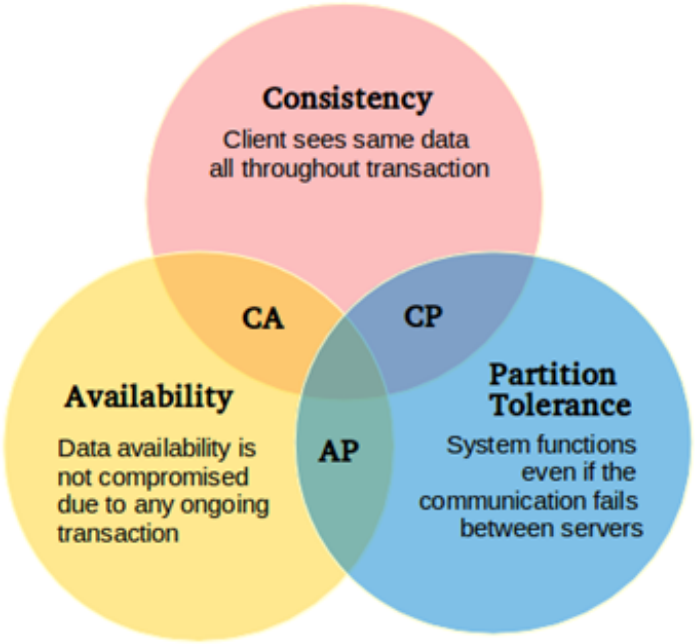
- Monitor the region servers in the Hbase clusters
  - Assign regions
  - Manage region server failover
  - Oversee load balancing of regions across all available region servers.
- Handle metadata operations
  - Manage and clean catalog tables
  - Clear the WAL
- Fault tolerance
  - Maintain a backup MasterServer in any HBase cluster

Zookeeper in a Hbase cluster

- Zookeeper is a distributed cluster of servers that collectively provides reliable coordination and synchronization services for clustered applications
- HBase leverages Zookeeper to coordinate the operations of the MasterServers, RegionServers, and clients

Hbase and CAP theorem

- HBase provides “Consistency” and “Partition Tolerance” but is not always “Available”
  - High degree of reliability
  - Tolerate any failure and still function properly



ACID properties

- HBase is not ACID-compliant, but does guarantee certain specific properties
  - Atomicity
    - All mutations are atomic within a row. Any put will either wholely succeed or wholely fail.
    - APIs that mutate several rows will not be atomic across the multiple rows
    - The order of mutations is seen to happen in a well-defined order for each row, with no interleaving
  - Consistency and Isolation
    - All rows returned via any access API will consist of a complete row that existed at some point in the table's history.

Acid properties (2)

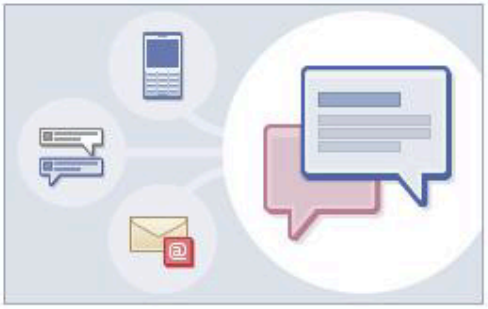
- Consistency of Scans
  - A scan is not a consistent view of a table
  - Scans do not exhibit snapshot isolation.
- Durability
  - All visible data is also durable data
  - Read will never return data that has not been made durable on disk.
  - Any operation that returns a "success" code (e.g. does not throw an exception) will be made durable
  - Any operation that returns a "failure" code will not be made durable (subject to the Atomicity guarantees above)
  - All reasonable failure scenarios will not affect any of the listed ACID guarantees

Usercase at Facebook: Messaging system

The New Messages

Texts, chat and email together in one simple conversation.

All your messages together



Get Facebook messages, chats and texts all in the same place.

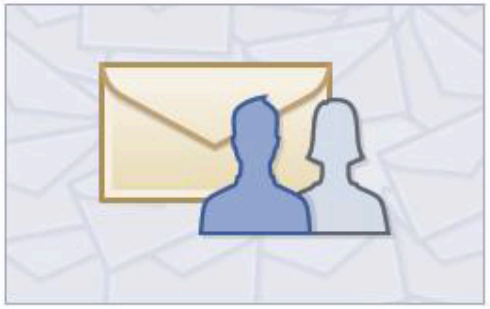
- Include email by activating your optional Facebook email address
- Control who can send you messages through your privacy settings

Full conversation history

See everything you've ever discussed with each friend as a single conversation.

- No need for subject lines or other formalities
- Easily leave large conversations that no longer interest you

The messages you want



Focus on messages from your friends.

- Messages from unknown senders and bulk email go into the Other folder
- Spam is hidden from view automatically

For more information, read the top questions about the new Messages.

Request an Invitation

HBase vs. RDBMS

HBase	RDBMS
Column-oriented	Row oriented (mostly)
Flexible schema, add columns on the fly	Fixed schema
Good with sparse tables	Not optimized for sparse tables
No query language	SQL
Wide tables	Narrow tables
Joins using MR –not optimized	Optimized for joins (small, fast ones too!)
Tight integration with MR	Not really…
De-normalize your data	Normalize as you can
Horizontal scalability –just add hardware	Hard to shard and scale
Consistent	Consistent
No transactions	Transactional
Good for semi-structured data as well as structured data	Good for structured data

HBase vs. HDFS

- Both are distributed systems that scale to hundreds or thousands of nodes
- HDFS is good for batch processing (scans over big files)
  - Not good for record lookup
  - Not good for incremental addition of small batches
  - Not good for updates
- HBase is designed to efficiently address the below points
  - Fast record lookup
  - Support for record-level insertion
  - Support for updates
- HBase updates are done by creating new versions of values

Thank you for your attention!

Q&A

Why use Hbase?

- Storing large amounts of data.
- High throughput for a large number of requests.
- Storing unstructured or variable column data.
- Big data with random read writes.