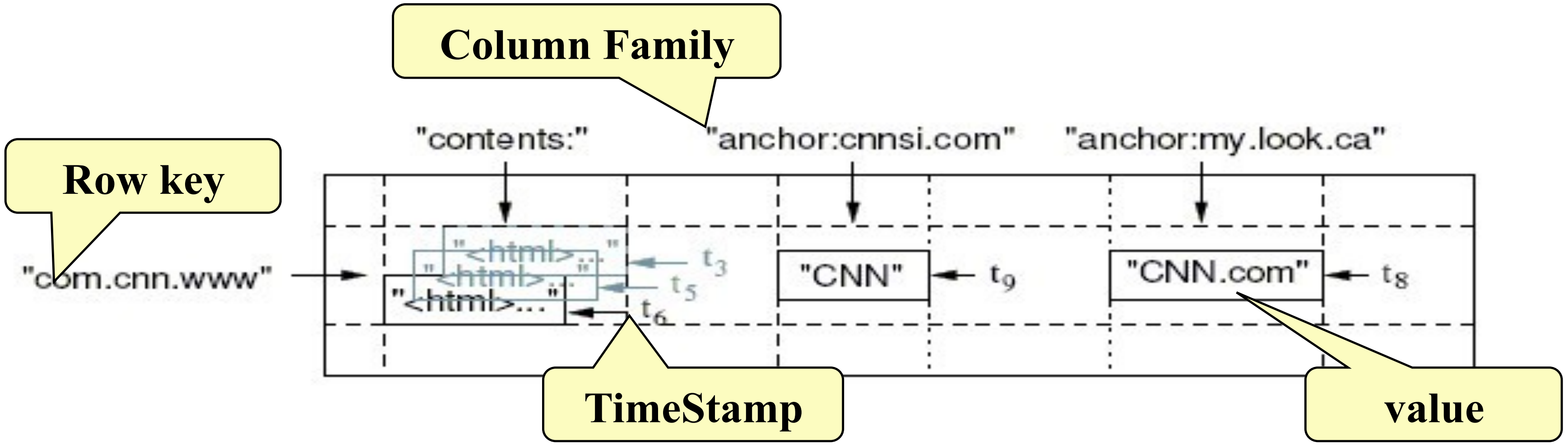
Data Model and Storage in NoSQL Systems (Bigtable, HBase)

Slides from Mohamed Eltabakh

Bigtable Data Model

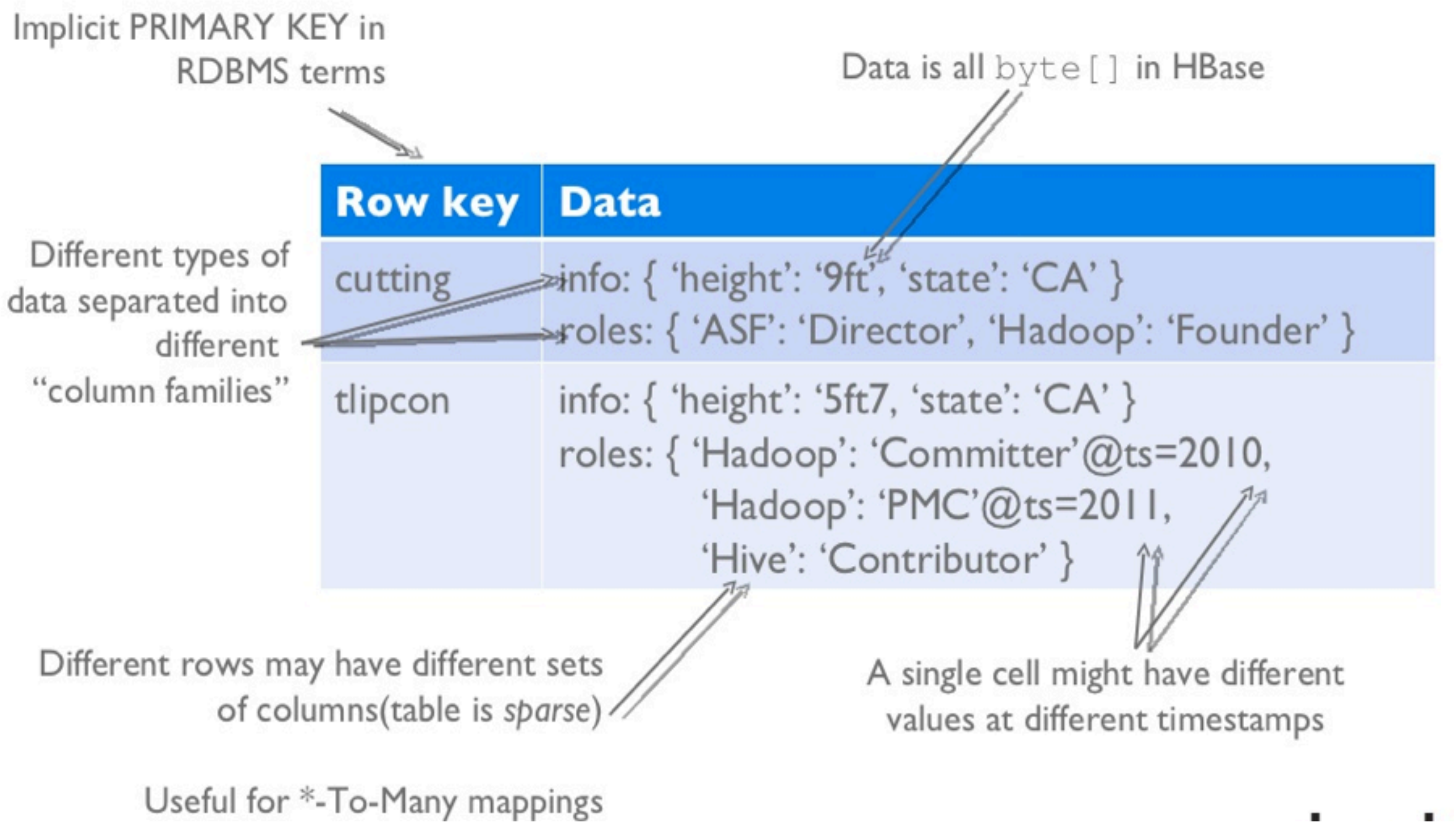


HBase’s Data Model is similar

Data Organization in HBase

- All data is stored in **Tables**
- Table **rows** have exactly one Key, and all rows in a table are *physically ordered by key*
- Tables have a fixed number of **Column Families**
- Each row can have many **Columns** in each column family
- Each column has a set of values, each with a **timestamp**
- Each rowkey:columnfamily:column:timestamp combination represents coordinates for a **Cell**

HBase Logical View



Implicit PRIMARY KEY in RDBMS terms

Data is all `byte[]` in HBase

Row key	Data

cutting	info: { 'height': '9ft', 'state': 'CA' }
	roles: { 'ASF': 'Director', 'Hadoop': 'Founder' }

tlipcon	info: { 'height': '5ft7, 'state': 'CA' }
	roles: { 'Hadoop': 'Committer'@ts=2010,
	'Hadoop': 'PMC'@ts=2011,
	'Hive': 'Contributor' }

Different types of data separated into different "column families"

Different rows may have different sets of columns(table is *sparse*)

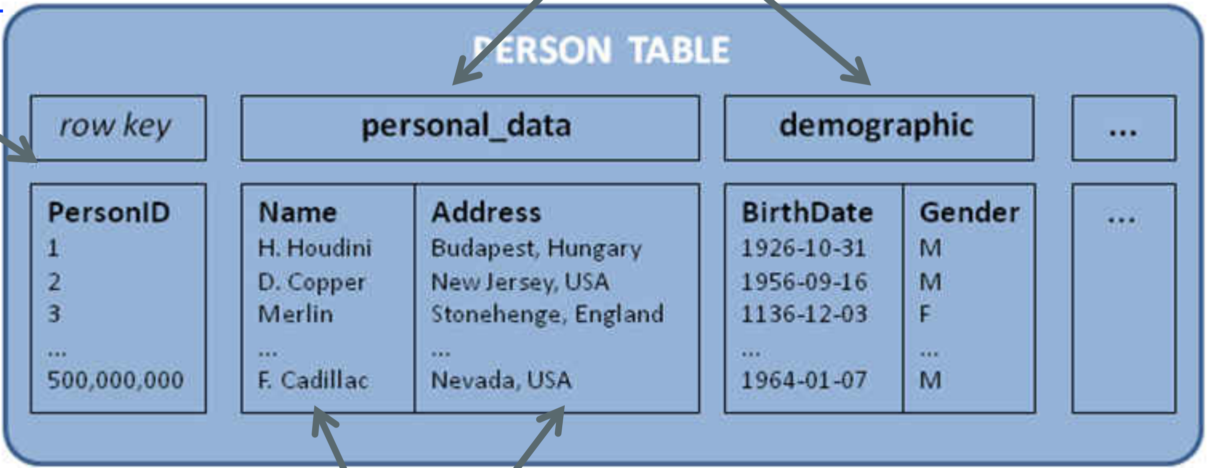
A single cell might have different values at different timestamps

Useful for *-To-Many mappings

HBase: Keys and Column Families

Each record is divided into *Column Families*

Each row has a *Key*



PERSON TABLE

row key	personal_data		demographic		...
PersonID	**Name**	**Address**	**BirthDate**	**Gender**	...
1	H. Houdini	Budapest, Hungary	1926-10-31	M	
2	D. Copper	New Jersey, USA	1956-09-16	M	
3	Merlin	Stonehenge, England	1136-12-03	F	
...	...	...	...	...	
500,000,000	F. Cadillac	Nevada, USA	1964-01-07	M	

Figure 2 Census Data in Column Families

Each column family consists of one or more *Columns*

What is a Column Family?

- A Column Family is a group of related columns
- All columns must be in a column family
- Each row can have a completely different set of columns for a column family

Row:	**Column Family:**	**Columns:**	
Chris	Friends	Friends:Bob	
Bob		Friends:Chris	
James		Friends:Jane	

Rows and Cells

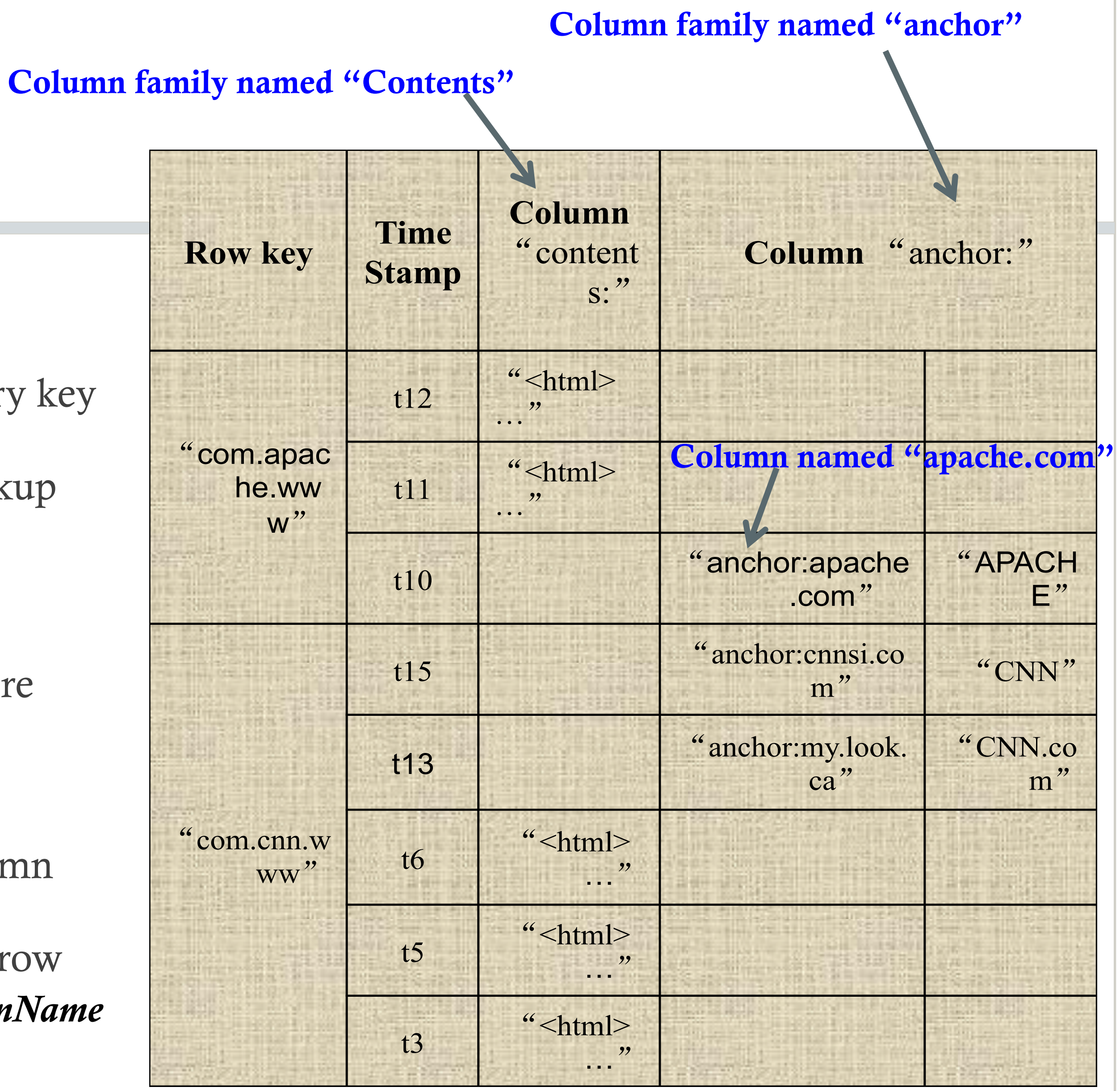
- Not exactly the same as rows in a traditional RDBMS
  - Key: a byte array
  - Data: Cells, qualified by column family, column, and timestamp (not shown here)

Row Key:	**Column Families : (Defined by the Table)**	**Columns: (Defined by the Row) (May vary between rows)**	**Cells: (Created with Columns)**
Chris	Attributes	Attributes:Age	30
		Attributes:Height	68
	Friends	Friends:Bob	1 (Bob's a cool guy)
		Friends:Jane	0 (Jane and I don't get along)

Cell Timestamps

- All cells are created with a timestamp
- Column family defines how many versions of a cell to keep
- Updates always create a new cell
- Deletes create a tombstone
- Queries can include an "as-of" timestamp to return point-in-time values

- **Key**
  - Byte array
  - Serves as the primary key for the table
  - Indexed for fast lookup
- **Column Family**
  - Has a name (string)
  - Contains one or more related columns
- **Column**
  - Belongs to one column family
  - Included inside the row
    - *familyName:columnName*


Column family named "Contents"
Column family named "anchor"
Column named "apache.com"

Row key	Time Stamp	Column "contents:"	Column "anchor:"	
"com.apache.www"	t12	"<html> …"		
	t11	"<html> …"		
	t10		"anchor:apache.com"	"APACHE"
"com.cnn.www"	t15		"anchor:cnnsi.com"	"CNN"
	t13		"anchor:my.look.ca"	"CNN.com"
	t6	"<html> …"		
	t5	"<html> …"		
	t3	"<html> …"		

- **Version Number**
  - Unique within each key
  - By default→ System's timestamp
  - Data type is Long
- **Value (Cell)**
  - Byte array

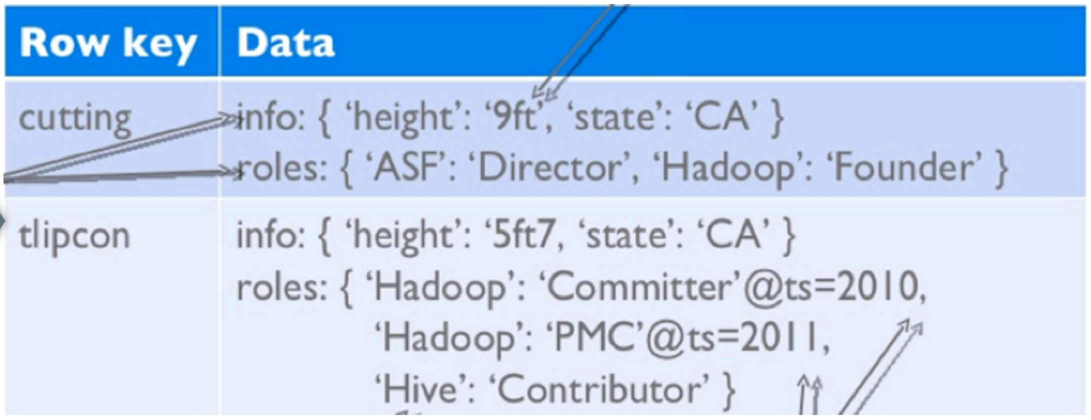
Version number for each row

Row key	Time Stamp	Column "contents:"	Column "anchor:"	
"com.apache.www"	t12	"<html>…"		value
	t11	"<html>…"		
	t10		"anchor:apache.com"	"APACHE"
"com.cnn.www"	t15		"anchor:cnnsi.com"	"CNN"
	t13		"anchor:my.look.ca"	"CNN.com"
	t6	"<html>…"		
	t5	"<html>…"		
	t3	"<html>…"		

Notes on Data Model

- HBase schema consists of several ***Tables***
- Each table consists of a set of ***Column Families***
  - Columns are not part of the schema
- HBase has ***Dynamic Columns***
  - Because column names are encoded inside the cells
  - Different cells can have different columns

“Roles” column family has different columns in different cells



Row key	Data

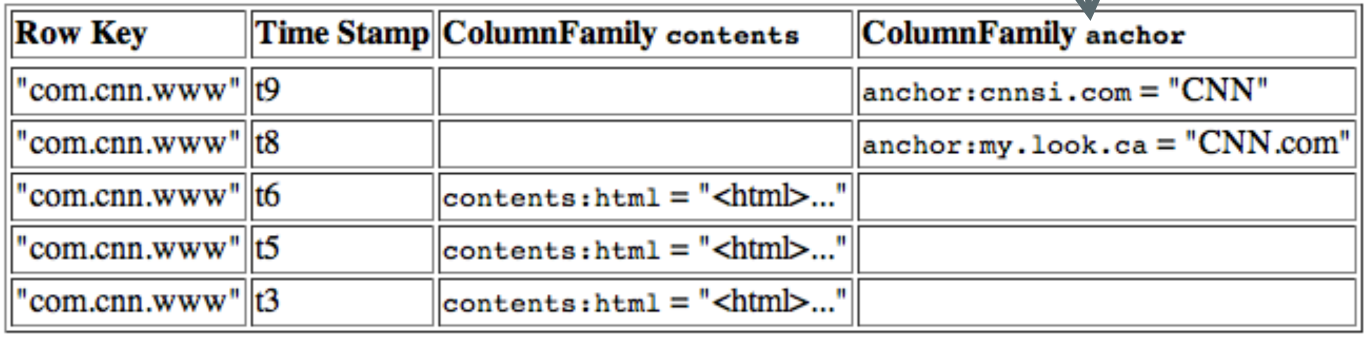
cutting	info: { ‘height’: ‘9ft’, ‘state’: ‘CA’ }
	roles: { ‘ASF’: ‘Director’, ‘Hadoop’: ‘Founder’ }

tlipcon	info: { ‘height’: ‘5ft7, ‘state’: ‘CA’ }
	roles: { ‘Hadoop’: ‘Committer’@ts=2010,
	‘Hadoop’: ‘PMC’@ts=2011,
	‘Hive’: ‘Contributor’ }

Notes on Data Model (Cont'd)

- The ***version number*** can be user-supplied
  - Even does not have to be inserted in increasing order
  - Version numbers are unique within each key
- Table can be very sparse
  - Many cells are empty
- ***Keys*** are indexed as the primary key

Has two columns
[cnnsi.com & my.look.ca]



Row Key	**Time Stamp**	**ColumnFamily** `contents`	**ColumnFamily** `anchor`
"com.cnn.www"	t9		`anchor:cnnsi.com` = "CNN"
"com.cnn.www"	t8		`anchor:my.look.ca` = "CNN.com"
"com.cnn.www"	t6	`contents:html` = "<html>..."	
"com.cnn.www"	t5	`contents:html` = "<html>..."	
"com.cnn.www"	t3	`contents:html` = "<html>..."	

HBase Physical Model

HBase Physical Model

- Each column family is stored in a separate file
- Key & Version numbers are replicated with each column family
- Empty cells are not stored

HBase maintains a multi-level index on values:
<key, column family, column name, timestamp>

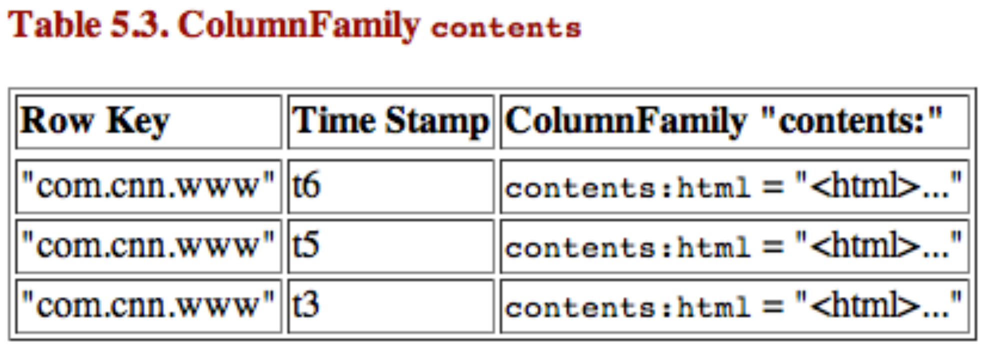


Table 5.3. ColumnFamily `contents`

Row Key	Time Stamp	ColumnFamily "contents:"
"com.cnn.www"	t6	`contents:html` = "<html>..."
"com.cnn.www"	t5	`contents:html` = "<html>..."
"com.cnn.www"	t3	`contents:html` = "<html>..."

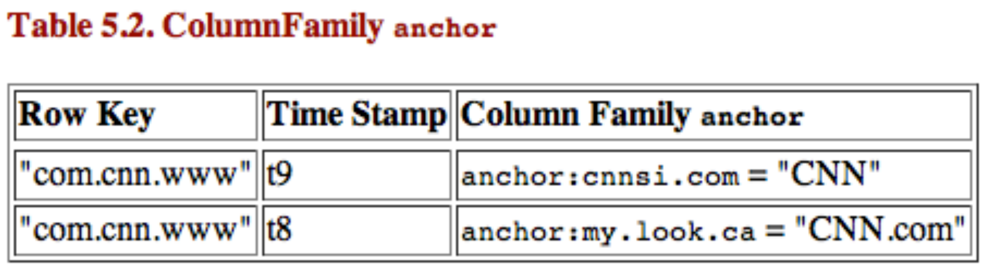
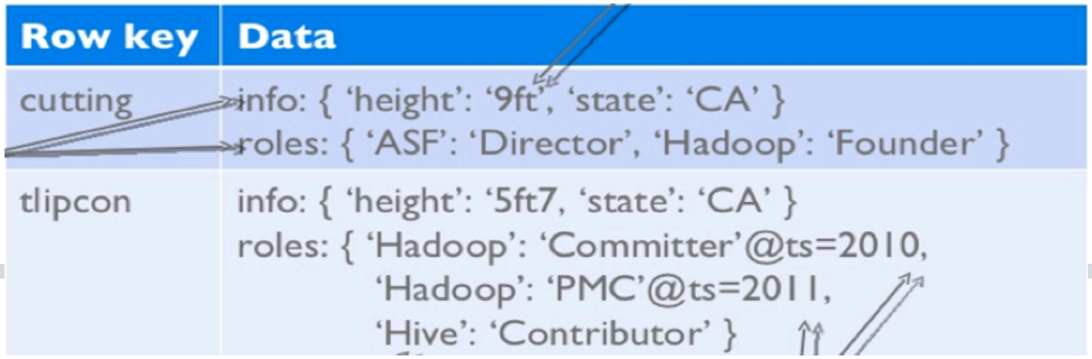


Table 5.2. ColumnFamily `anchor`

Row Key	Time Stamp	Column Family `anchor`
"com.cnn.www"	t9	`anchor:cnnsi.com` = "CNN"
"com.cnn.www"	t8	`anchor:my.look.ca` = "CNN.com"

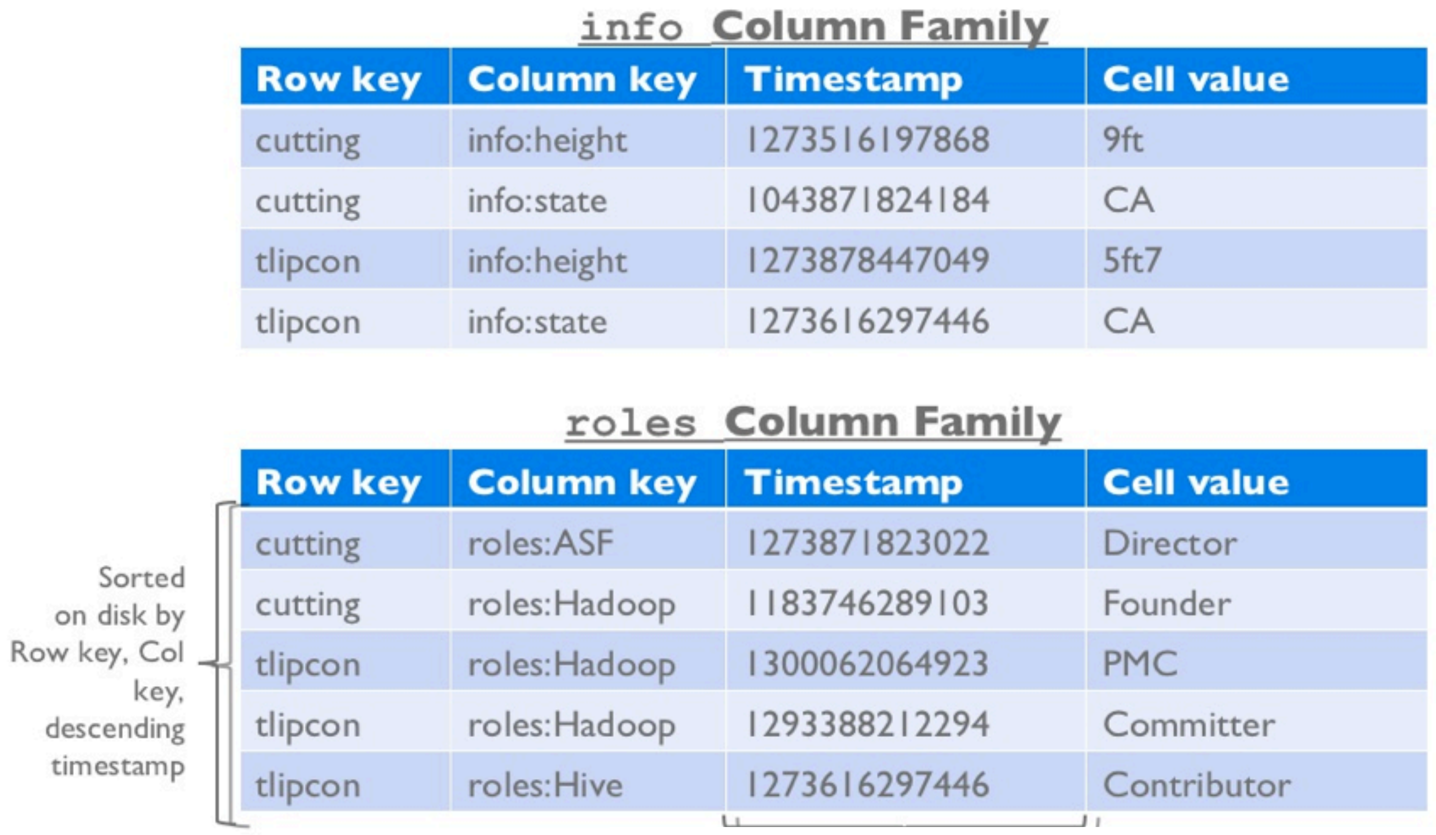
Example



Row key	Data

cutting	info: { 'height': '9ft', 'state': 'CA' }
	roles: { 'ASF': 'Director', 'Hadoop': 'Founder' }

tlipcon	info: { 'height': '5ft7, 'state': 'CA' }
	roles: { 'Hadoop': 'Committer'@ts=2010,
	'Hadoop': 'PMC'@ts=2011,
	'Hive': 'Contributor' }



info Column Family

Row key	Column key	Timestamp	Cell value
cutting	info:height	1273516197868	9ft
cutting	info:state	1043871824184	CA
tlipcon	info:height	1273878447049	5ft7
tlipcon	info:state	1273616297446	CA

roles Column Family

Row key	Column key	Timestamp	Cell value
cutting	roles:ASF	1273871823022	Director
cutting	roles:Hadoop	1183746289103	Founder
tlipcon	roles:Hadoop	1300062064923	PMC
tlipcon	roles:Hadoop	1293388212294	Committer
tlipcon	roles:Hive	1273616297446	Contributor

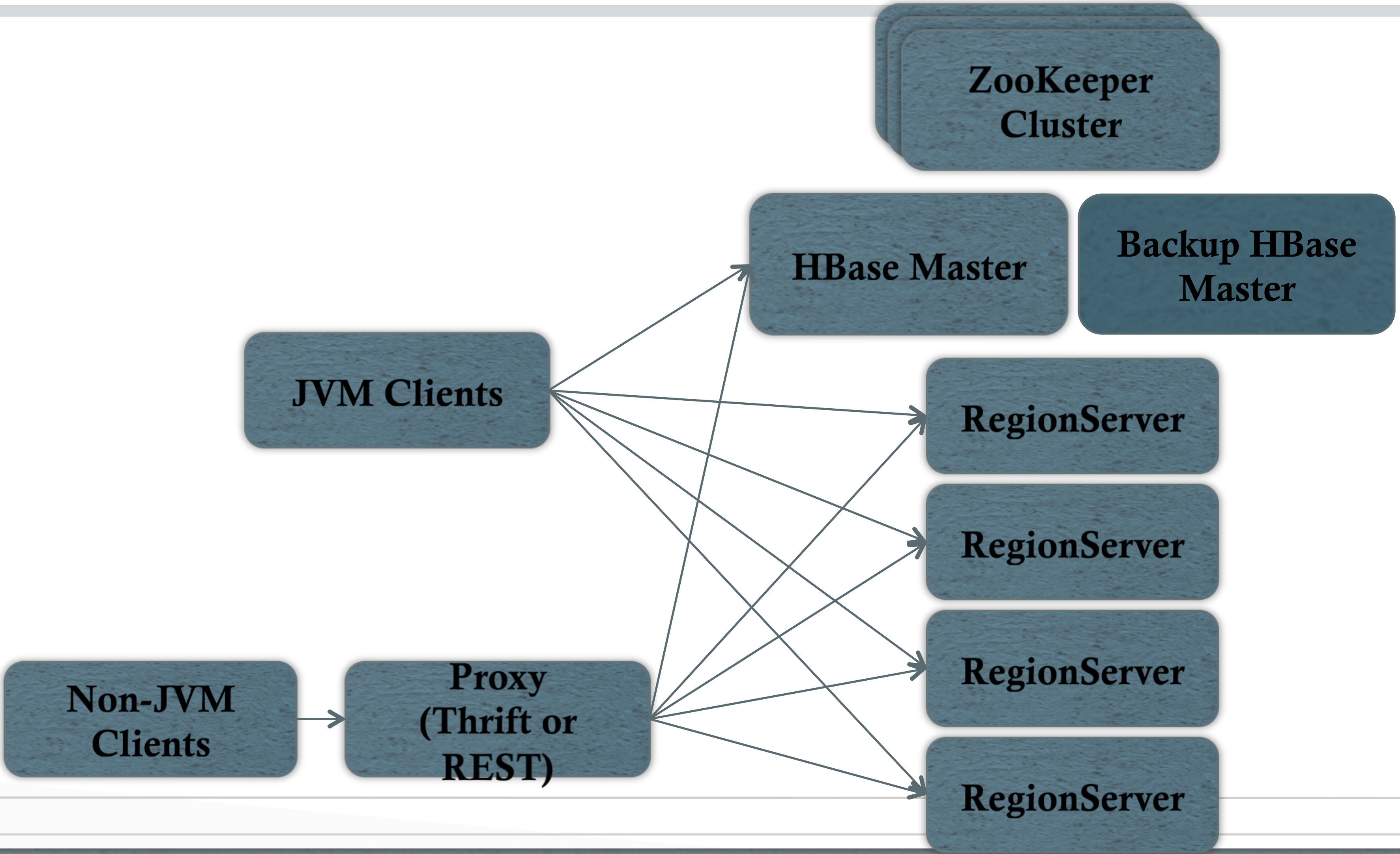
Sorted on disk by Row key, Col key, descending timestamp

Number of Milliseconds since Epoch

HBase Data Partitioning

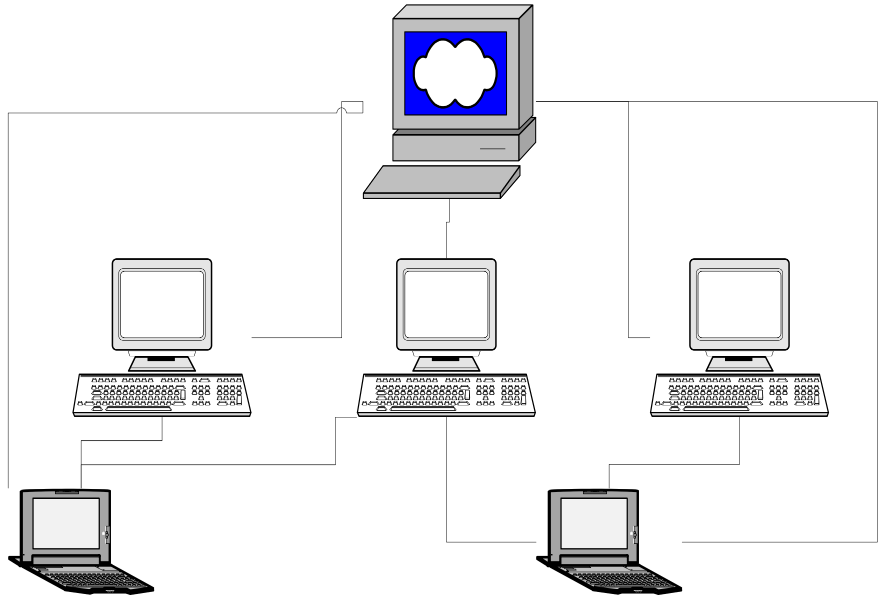
- HBase scales horizontally
- Needs to split data over many RegionServers
- Regions are the unit of scale

HBase Architecture



Three Major Components

- The HBaseMaster
  - One master
- The HRegionServer
  - Many region servers
- The HBase client



HBase Components

- **Region**
  - A subset of a table's rows, like horizontal range partitioning
  - Automatically done
- **RegionServer (many slaves)**
  - Manages data regions
  - Serves data for reads and writes
- **Master**
  - Responsible for coordinating the slaves
  - Assigns regions, detects failures
  - Admin functions

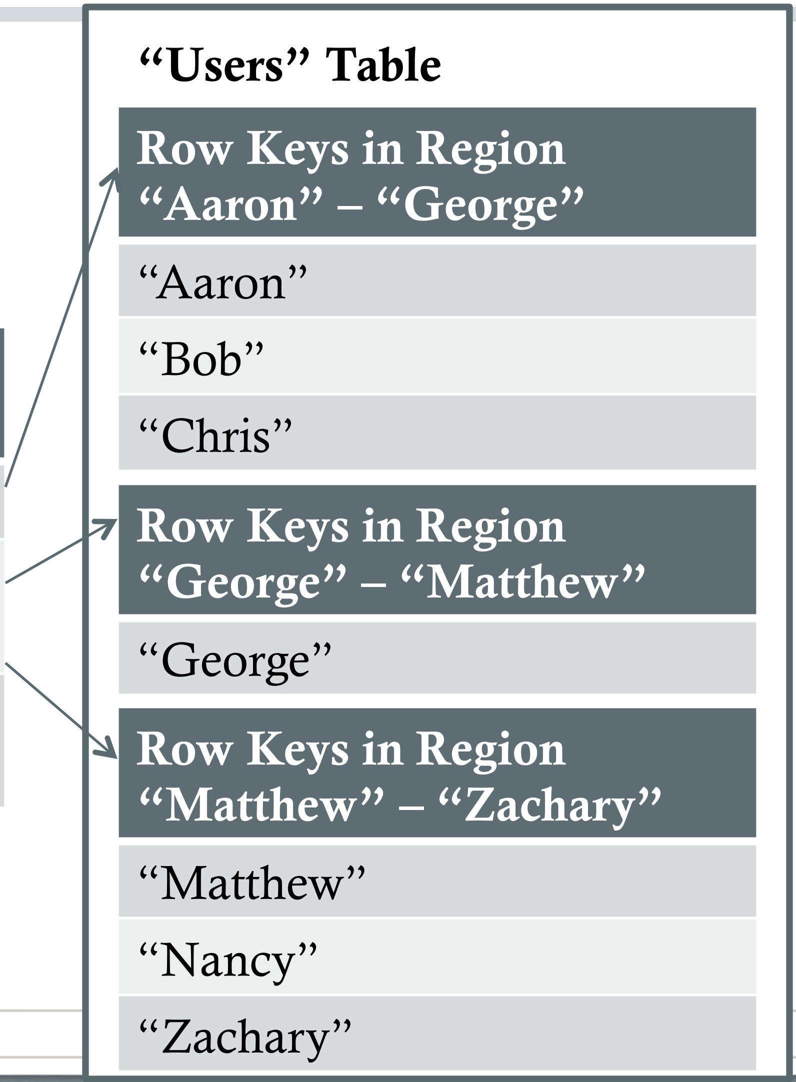
Regions & RegionServers

- All HBase tables are broken into 1 or more regions
- Regions have a start row key and an end row key
- Each Region lives on exactly one RegionServer
- RegionServers may host many Regions
- When RegionServers die, Master detects this and assigns Regions to other RegionServers

Region Distribution

-META- Table

Table	Region	Region Server
Users	"Aaron" – "George"	Node01
	"George" – "Matthew"	Node02
	"Matthew" – "Zachary"	Node01



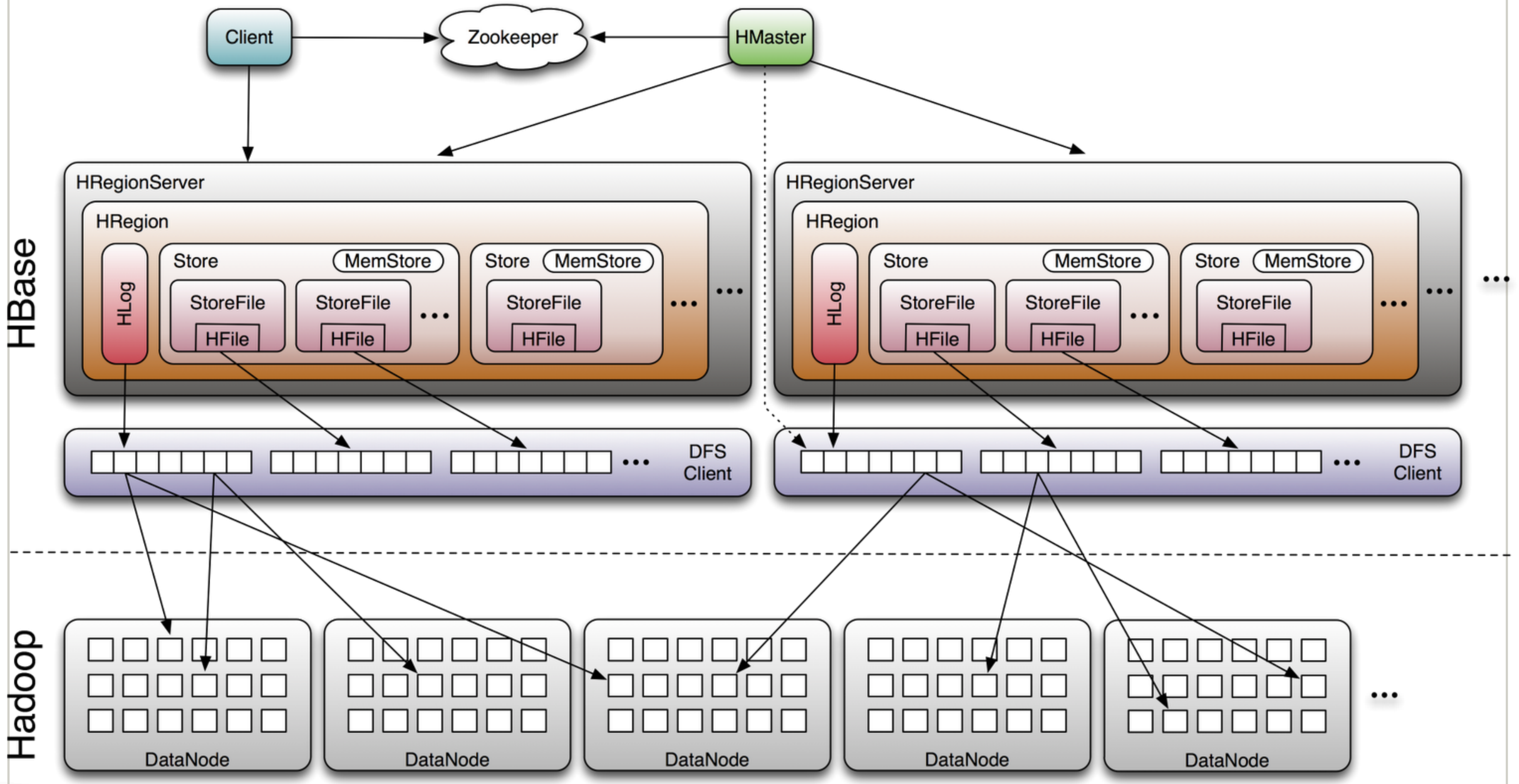
"Users" Table

Row Keys in Region "Aaron" – "George"
"Aaron"
"Bob"
"Chris"

Row Keys in Region "George" – "Matthew"
"George"

Row Keys in Region "Matthew" – "Zachary"
"Matthew"
"Nancy"
"Zachary"

Big Picture


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

Use Case – Time Series

- **Requirement**: Store real-time stock tick data

Ticker	Timestamp	Sequence	Bid	Ask
IBM	09:15:03:001	1	179.16	179.18
MSFT	09:15:04:112	2	28.25	28.27
GOOG	09:15:04:114	3	624.94	624.99
IBM	09:15:04:155	4	179.18	179.19

- **Requirement**: Accommodate many simultaneous readers & writers
- **Requirement**: Allow for reading of current price for any ticker at any point in time

Time Series Use Case – RDBMS Solution

Historical Prices:

Keys	Column	DataType
Primary Key	Ticker	Varchar
	Timestamp	DateTime
	Sequence_Number	Integer
	Bid_Price	Decimal
	Ask_Price	Decimal

Latest Prices:

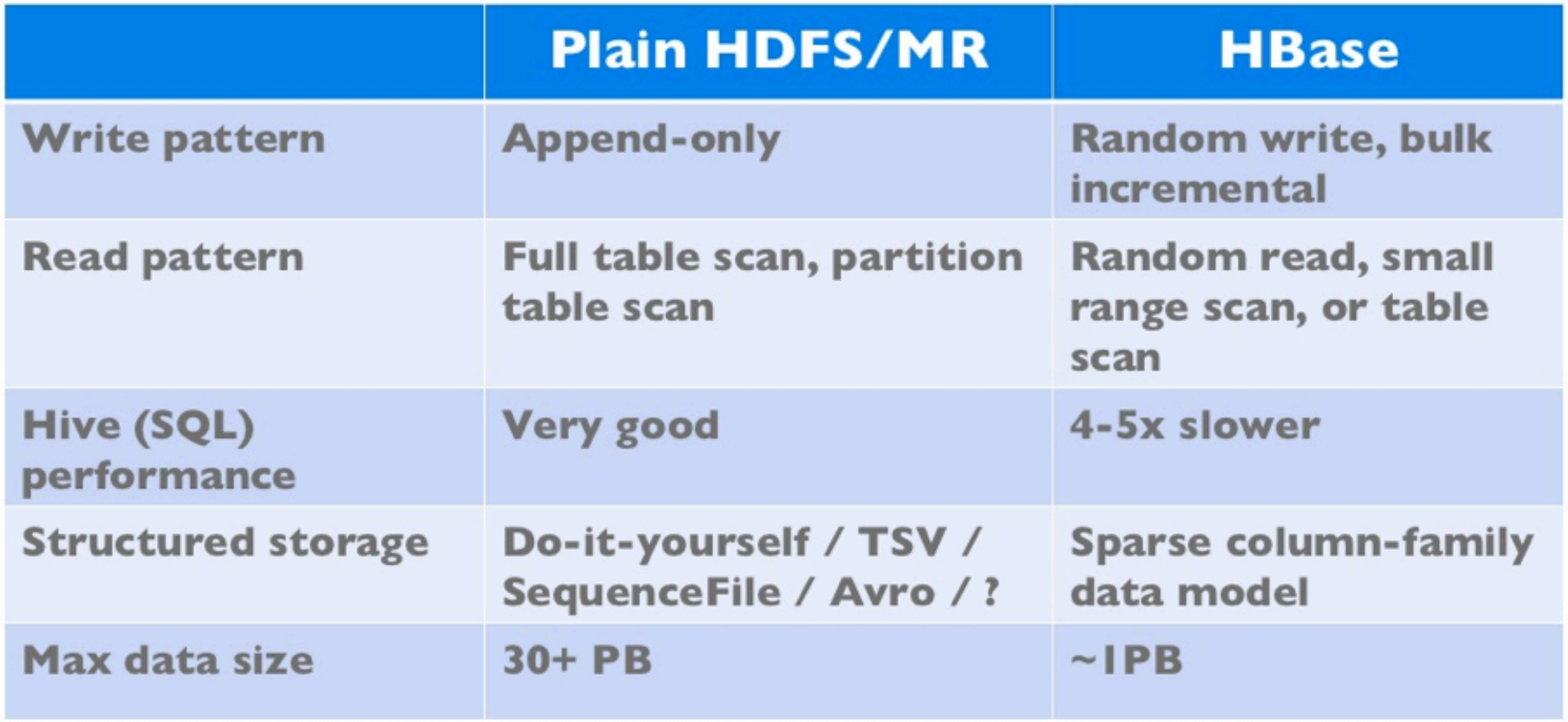
Keys	Column	DataType
Primary Key	Ticker	Varchar
	Bid_Price	Decimal
	Ask_Price	Decimal

Time Series Use Case – HBase Solution

Row Key	Family:Column
[Ticker].[Reverse_Timestamp].[Reverse_Sequence_Number]	Prices:Bid
	Prices:Ask

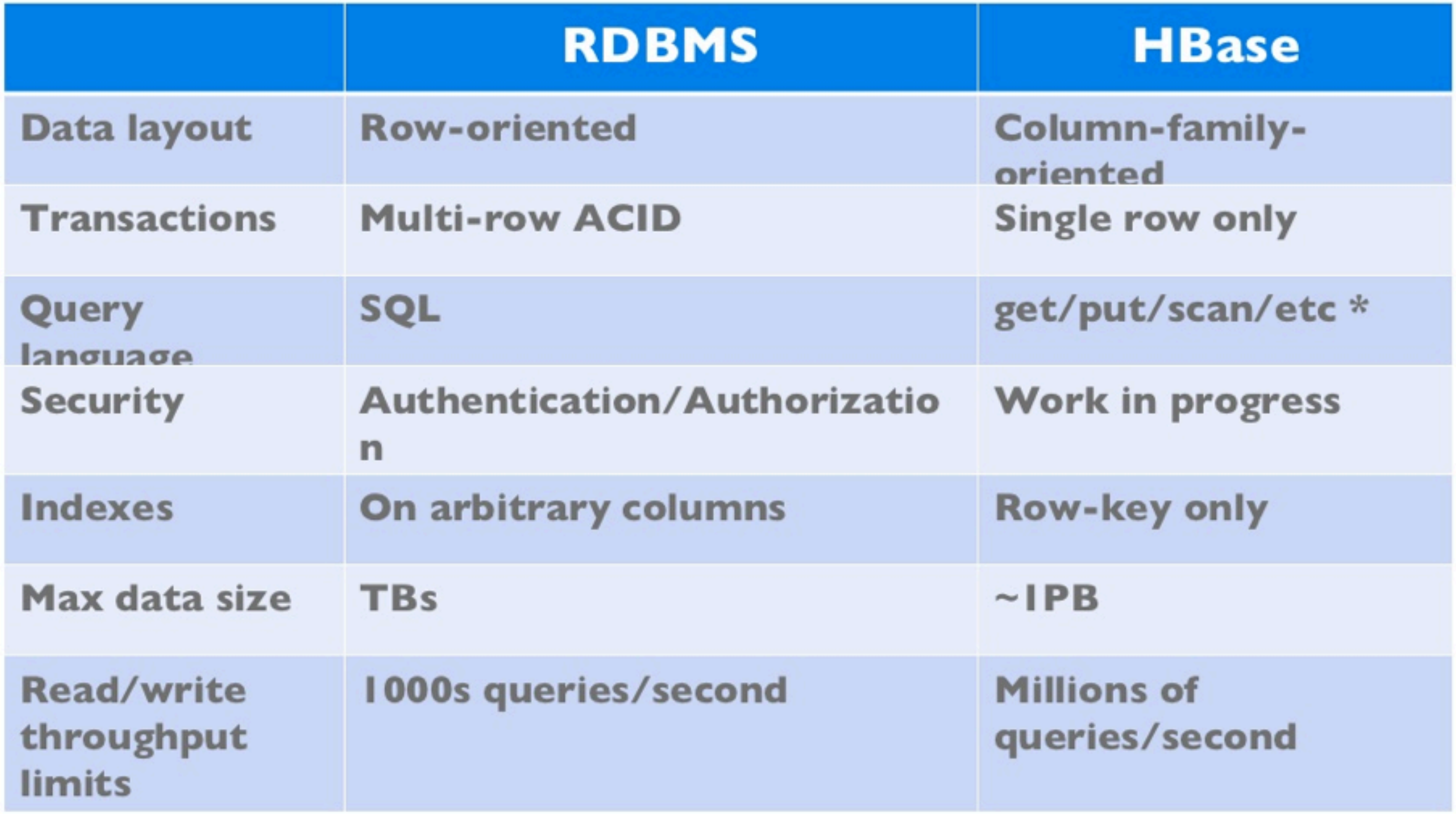
- No need to keep separate "latest price" table
  - A scan starting at "ticker" will always return the latest price row
- **Let us analyze this solution further (reverse timestamps are explained at the link below):**
  http://hbase.apache.org/0.94/book/rowkey.design.html

HBase vs. HDFS



	Plain HDFS/MR	HBase
Write pattern	Append-only	Random write, bulk incremental
Read pattern	Full table scan, partition table scan	Random read, small range scan, or table scan
Hive (SQL) performance	Very good	4-5x slower
Structured storage	Do-it-yourself / TSV / SequenceFile / Avro / ?	Sparse column-family data model
Max data size	30+ PB	~1PB

HBase vs. RDBMS



	RDBMS	HBase
Data layout	Row-oriented	Column-family-oriented
Transactions	Multi-row ACID	Single row only
Query language	SQL	get/put/scan/etc *
Security	Authentication/Authorizatio n	Work in progress
Indexes	On arbitrary columns	Row-key only
Max data size	TBs	~1PB
Read/write throughput limits	1000s queries/second	Millions of queries/second