Big Data Processing, 2014/15

Lecture 10: HBase

Claudia Hauff (Web Information Systems)
ti2736b-ewi@tudelft.nl

Course content

- Introduction
- Data streams 1 & 2
- The MapReduce paradigm
- Looking behind the scenes of MapReduce: HDFS & Scheduling
- Algorithm design for MapReduce
- A high-level language for MapReduce: Pig Latin 1 & 2
- **MapReduce is not a database, but HBase nearly is**
- Lets iterate a bit: Graph algorithms & Giraph
- How does all of this work together? ZooKeeper/Yarn

Learning objectives

- **Explain** how HBase differs from Hadoop and RDBMSs
- **Decide** for which use cases HBase is suitable
- **Explain** the HBase organisation
- **Use** HBase from the terminal

Lets start with the last point …

HBase from the shell

- `hbase shell`
- `create ‘food’, ‘local’`
- `put ‘food’, ‘row1’, ‘local:source’, ‘Delft’`
- `put ‘food’, ‘row1’, ‘local:name’, ‘tomato’`
- `put ‘food’, ‘row1’, ‘local:price’, ‘1.20’`
- `put ‘food’, ‘row1’, ‘local:price’, ‘1.29’`
- `put ‘food’, ‘row2’, ‘local:name’, ‘pineapple’`
- `scan ‘food’`
- `scan ‘food’, {COLUMNS => [‘local:source’]}`
- `get ‘food’, ‘row1’`
- `count ‘food’`
- `describe ‘food’`
- `disable ‘food’; drop ‘food’`

Introduction to HBase

HBase

HBase is not ACID compliant

"HBase is a **distributed column-oriented** *database* built on top of HDFS. HBase is the Hadoop application to use when you require **real-time** read/write **random** access to very **large** datasets." (Tom White)

"HBase tables are like those in an RDBMS, only cells are **versioned**, rows are **sorted**, and columns can be **added on the fly**…" (Tom White)

HBase in the Hadoop ecosystem

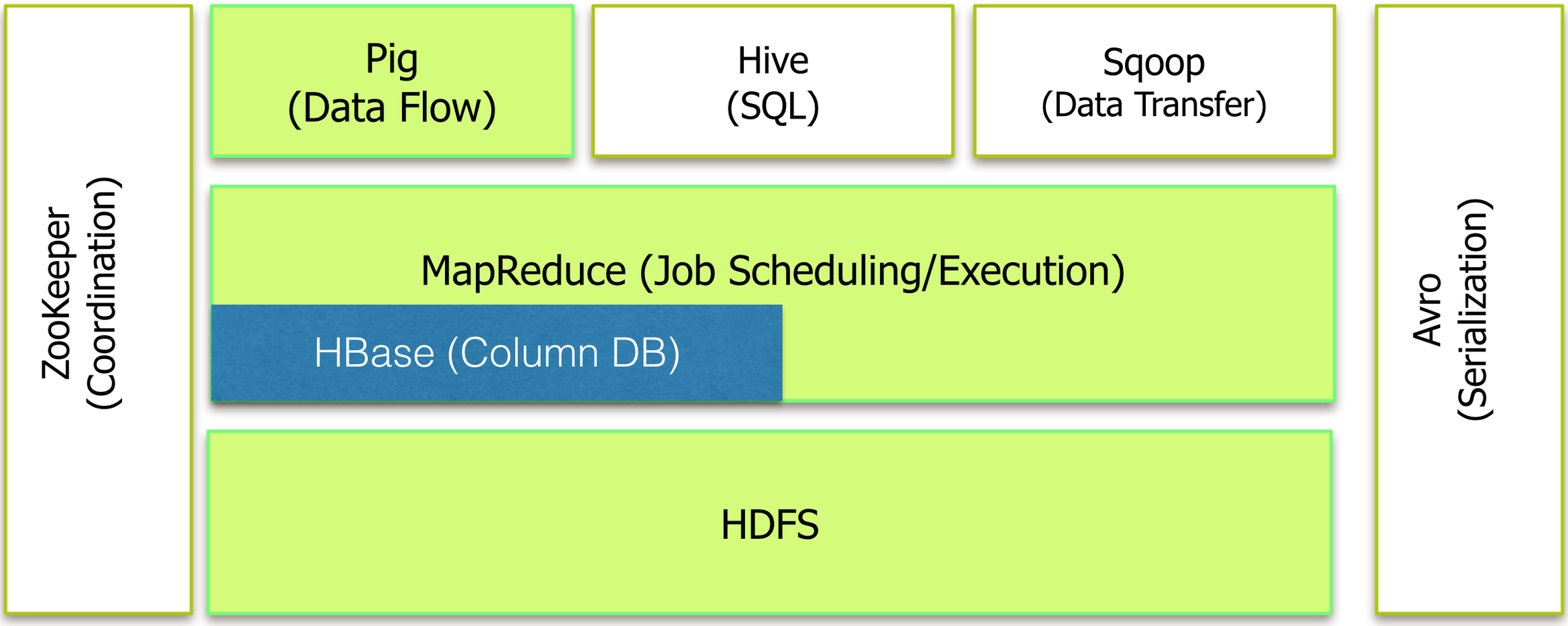


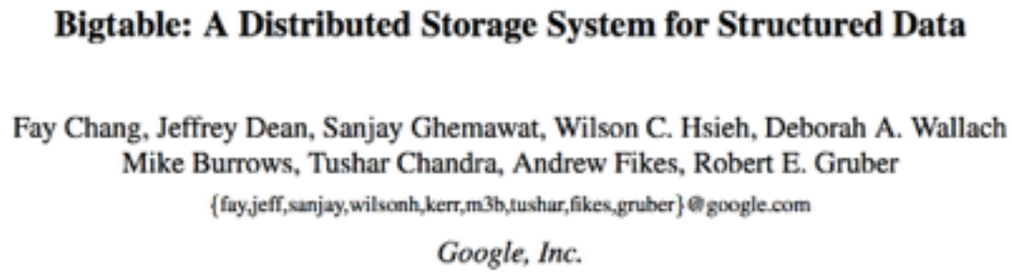
Image source: Cloudera

Main points

- HBase is **not** an ACID-compliant database
- HBase does **not** support a full relational model
- HBase provides clients with a **simple data model**
- Clients have **dynamic control** over data layout and format
- Clients can control the **locality of their data** by creating appropriate schemas

History of HBase

- Started at the end of **2006**
- Modelled after **Google's Bigtable paper** (2006)



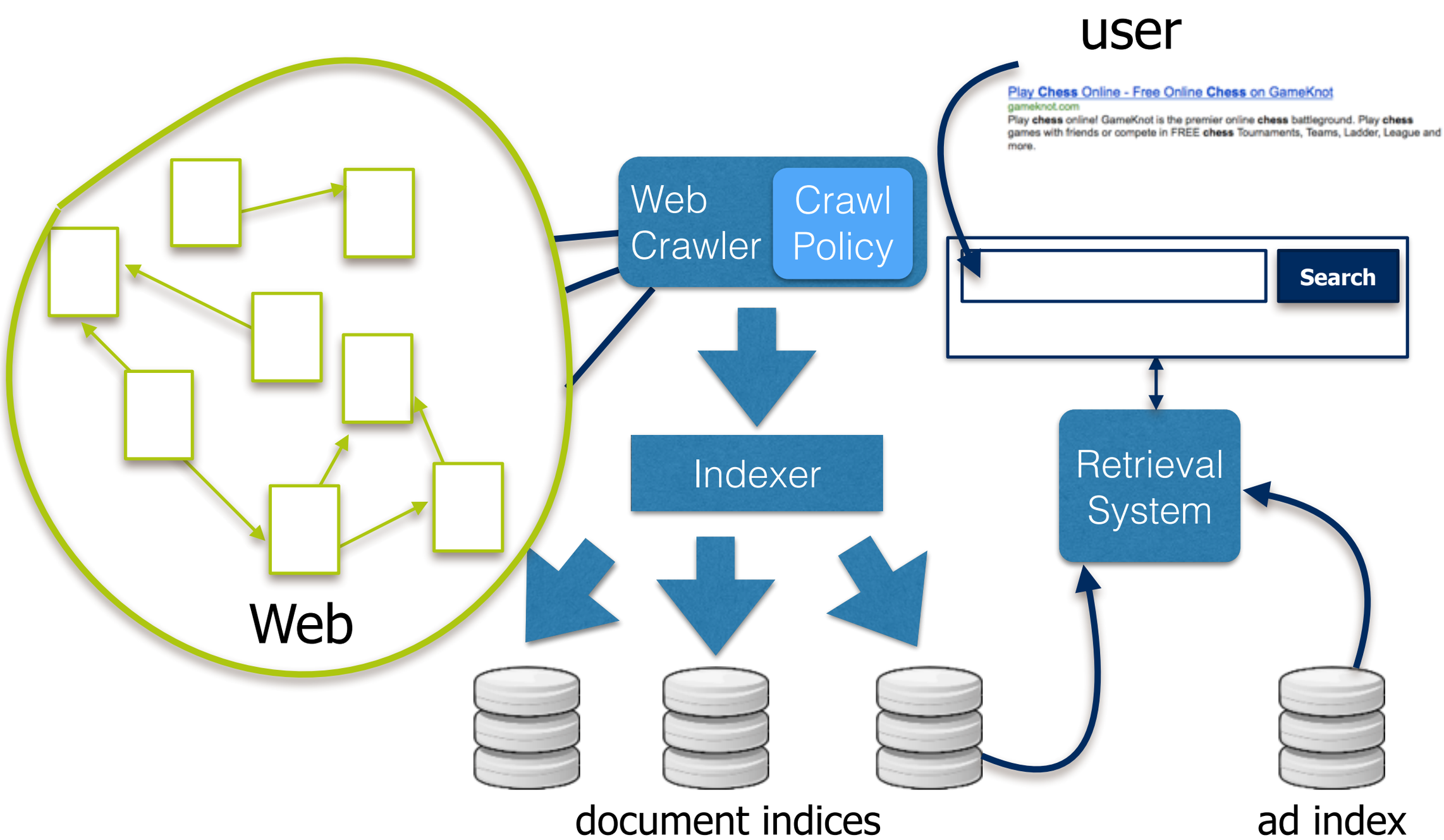
Bigtable: A Distributed Storage System for Structured Data

Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach
Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber
{fay,jeff,sanjay,wilsonh,kerr,m3b,tushar,fikes,gruber}@google.com
Google, Inc.

highly recommended read!

- **January 2008:** Hadoop becomes Apache top level project, HBase becomes subproject
- **May 2010:** HBase becomes an Apache top level project
- Contributors from Cloudera, Facebook, Intel, Hortonworks, etc.

Use case: Web tables



Use case: Web tables

- Table of **crawled web pages** and their **attributes** (content, language, anchor text, inlinks, …) with **web page URL as row key**
  - **Retrieval**: [first round] simple approach, [second round] subset of pages ranked by complex machine learning algorithms

- Webtable contains **billions of rows** (URLs)

- **Batch processes** are running against Webtable, deriving statistics (PageRank, etc.) and adding new columns for indexing, ranking, …

- Webtable is **randomly accessed** by many crawlers concurrently running at various rates and **updating random rows**

- **Cached web pages** are served to users in **real-time**

- **Different versions of a web page** are used to compute **crawler frequency**

Demands for HBase

- **Structured data**, scaling to petabytes
- **Efficient** handling of **diverse** data
  - Wrt. data size (URLs, web pages, satellite imagery)
  - Wrt. latency (backend bulk processing vs. real-time data serving)
- **Efficient** read and write of **individual records**

HBase vs. Hadoop

- Hadoop's use case is **batch processing**
  - Not suitable for a single record lookup
  - Not suitable for adding **small amounts** of data at all times
  - Not suitable for making **updates** to existing records

- HBase addresses Hadoop's weaknesses
  - Provides fast lookup of **individual** records
  - Supports **insertion** of **single** records
  - Supports record **updating**
  - Not all columns are of interest to everyone; each client only wants a particular subset of columns (**column-based storage**)

HBase vs. Hadoop

HBase is built on top of HDFS!

	Hadoop	HBase
writing	file append only, no updates	random write, updating
reading	sequential	random read, small range scan, full scan
structured storage	up to the user	sparse column family data model

HBase vs. RDBMS

small to medium-volume applications

use when scaling up in terms of dataset size, read/write concurrency

	RDBMS	HBase
schema	fixed	random write, updating
orientation	row-oriented	column-oriented
query language	SQL	simple data access model
size	terabytes (at most)	billions of rows, millions of columns
scaling up	difficult (workarounds)	add nodes to a cluster

Question: which tool is best suited for which use case?

- Data generated by the Large Hadron Collider is stored and analysed by researchers
- All pages crawled from the Web are stored by a search engine and served to clients via its search interface
- Data generated by the Hubble telescope is used in the SETI@Home project (served at request to users)
- Data generated by the Dutch tax office about tax payers is used to send warning letters (“you are late with your taxes”)

How does a RDBMS scale up in practice?

RDBMS scaling story

- Initial public launch of service
  - Remotely hosted **MySQL** instance with well-defined schema

- Service becomes popular; too many reads hitting the database
  - Add **memcached** to cache common queries

- Service grows, too many writes are hitting the database
  - **Scale MySQL vertically** by buying a new (larger) server

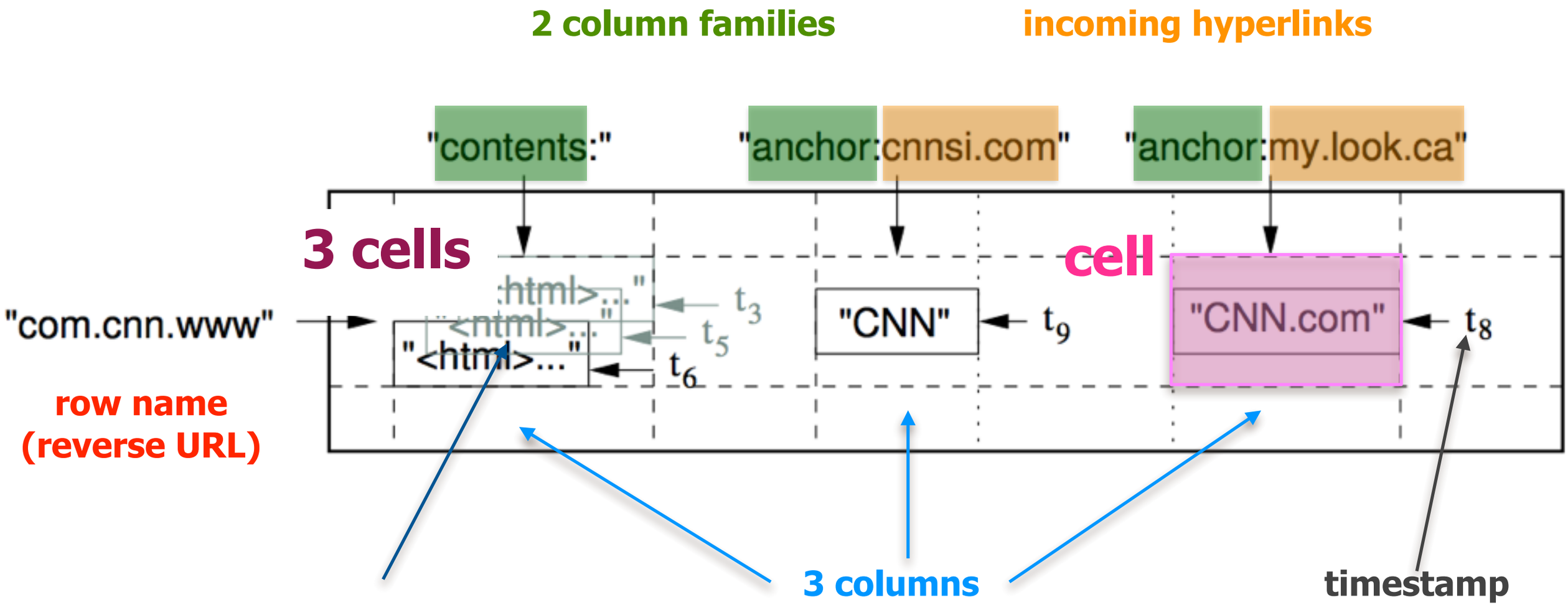
RDBMS scaling story

- New features of the service increase query complexity; now there are too many joins
  - **Denormalize data** to reduce joins (not good DB practice)
- Rising popularity swamps the server; things are too slow
  - **Stop** doing any **server-side computations**
- Some queries are still too slow
  - Periodically **prematerialise** the most **complex queries**
- Reads are OK, but writes are getting slower and slower
  - **Drop** secondary **indexes**

What now?

HBase data model

A row

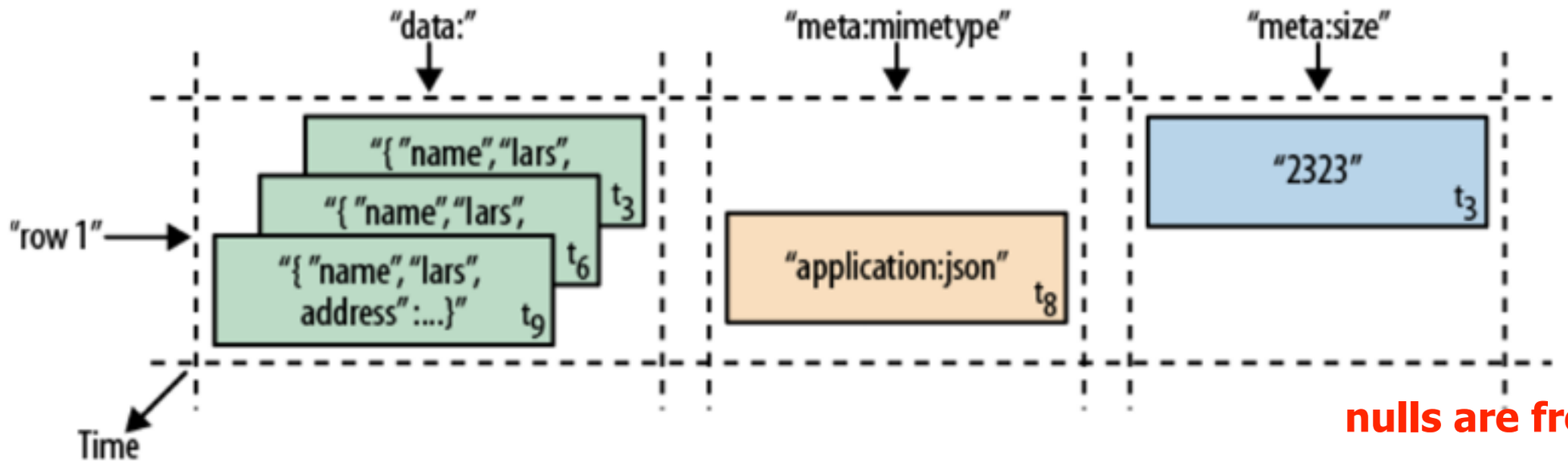


3 versions of
the web page
stored: table cells are versioned

different columns written at different times

HBase vs. RDBMS

sparse, distributed,
persistent multi-dimensional
sorted map



nulls are free of cost

Row Key	Time Stamp	Column "data:"	Column "meta:" "mimetype"	Column "meta:" "size"	Column "counters:" "updates"
"row1"	t_3	"{ "name" : "lars", "address" : ...}"		"2323"	"1"
	t_6	"{ "name" : "lars", "address" : ...}"			"2"
	t_8		"application/json"		
	t_9	"{ "name" : "lars", "address" : ...}"			"3"

Image source: Tom White's Hadoop The Definite Guide

Table rows

- Row keys are **byte arrays**
  - i.e. anything can be a row key

- Row keys are **unique**

- Rows are **sorted lexicographically** by row key (similar to a primary key index in RDBMS)

```
row-1
row-11
row-111
row-2
row-22
row-3
```

- Rows are composed of **columns**

- Read/write access to row data is **atomic**

Table columns & column families

- Columns are **grouped into column families**
  - Semantic or topical boundaries
  - Useful for compression, caching

millions of columns in a column family

- Column family members have a **common prefix**
  - E.g. `anchor:cnnsi.com`, `anchor:bbc.co.uk/sports`

- A table's **column families** must be specified as part of the table schema definition
  - **Few families, few updates**
  - Column family members can be **added on demand**, e.g. `anchor:ww.twitter.com/bbc_sports` can be added to a table having column family `anchor`

Table columns & column families

- **Tuning and storage specifications** happen at the **column-family** level

- For performance reasons column family's members should **share** the same general **access patterns** and **size characteristics**

- All columns in a column family are **stored together** in the same low-level storage file: HFile

- Columns can be written to at different times

Table cells

```
(Table, RowKey, Family, Column, Timestamp) → Value
```

table column families columns

```
SortedMap<RowKey, List<SortedMap<Column,
List<Value, Timestamp>>>>
```

cells

- Cells are indexed by a **row key**, a **column key** and a **timestamp**
- Table cells are "**versioned**"
  - Default: auto-assigned timestamp (insertion time)
  - Timestamp can be set explicitly by the user
- Cells are stored in decreasing timestamp order
- User can specify how many versions to keep
- Cell values are uninterpreted array of bytes - clients need to know what to do with the data

Table cells

- API provides a coherent view of rows as a combination of all columns and their most current versions

- By **default** API returns the value with the **most recent timestamp**

- User can query HBase for values before/after a specific timestamp or more than one version at a time

Summary

- HBase shell
- HBase vs. RDBMS/Hadoop
- HBase organisation

THE END