Column-family Stores, Cassandra

Lecture 7 of *NoSQL Databases* (PA195)

David Novak & Vlastislav Dohnal
Faculty of Informatics, Masaryk University, Brno

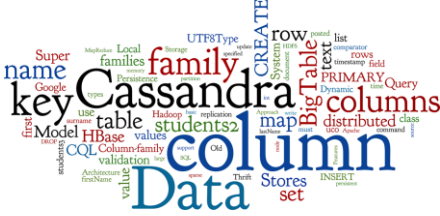
NoSQL Databases and Data Types

1. Key-value stores:
   - Can store any (text or binary) data
     - often, if using JSON data, additional functionality is available

2. Document databases
   - Structured text data - Hierarchical tree data structures
     - typically JSON, XML

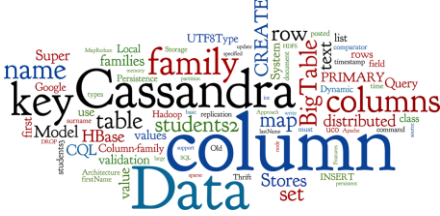
3. Column-family stores
   - Rows that have many columns associated with a row key
     - can be written as JSON

Agenda



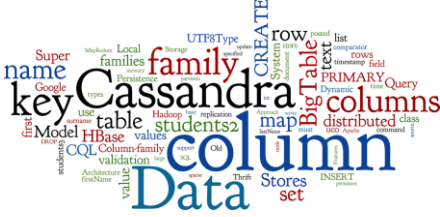
- Data Model
  - Column families, super columns, two points of view
- Column-family Stores
  - Google BigTable, Cassandra, HBase
- Cassandra in Detail
  - Cassandra data model 1.0 vs. 2.0
  - Cassandra Query Language (CQL)
  - Data partitioning, replication
  - Local Data Persistence
  - Query processing, Indexes, Lightweight Transactions

Agenda



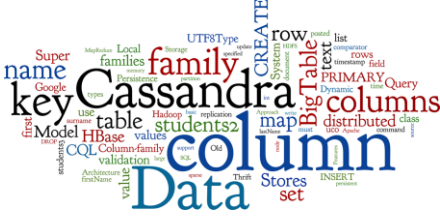
- Data Model
  - Column families, super columns, two points of view
- Column-family Stores
  - Google BigTable, Cassandra, HBase
- Cassandra in Detail
  - Cassandra data model 1.0 vs. 2.0
  - Cassandra Query Language (CQL)
  - Data partitioning, replication
  - Local Data Persistence
  - Query processing, Indexes, Lightweight Transactions

Column-family Stores: Basics

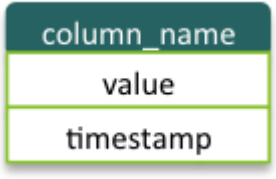


- AKA: wide-column, columnar
  - not to confuse with column-oriented RDBMS
- Data model: rows that have many columns associated with a row key
- Column families are groups of related data (columns) that are often accessed together
  - e.g., for a customer we typically access all profile information at the same time, but not customer's orders

Data Model: Column



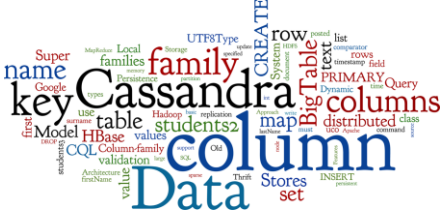
- **Column =** the basic data item
  - a 3-tuple consisting of
    - column **name**
    - **value**
    - **timestamp**
  - Can be modeled as follows




```
{ name: "firstName",
  value: "Martin",
  timestamp: 12345667890 }
```

- In the following, we will ignore the timestamp

Data Model: Row



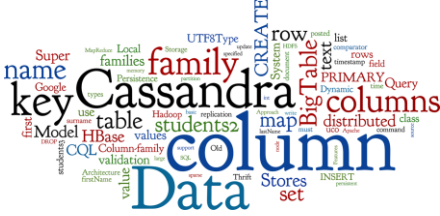
- **Row:** a collection of columns attached to row key
  - Columns can be added to any row at any time
    - without having to add it to other rows

```
// row
"martin-fowler" : {
    firstName: "Martin",
    lastName: "Fowler",
    location: "Boston"
 }
```

Row key1	Column Key1	Column Key2	Column Key3	...
	Column Value1	Column Value2	Column Value3	
⋮				

http://www.ebaytechblog.com/2012/07/16/cassandra-data-modeling-best-practices-part-1/

Data Model: Column Family

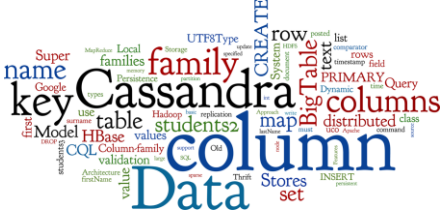


- **CF =** Set of columns containing “related” data

user_id (*row key*)	*column key*	*column key*	...
	column value	*column value*	...
1	login	first_name	...
	honza	Jan	...
4	login	age	...
	david	35	...
5	first_name	last_name	...
	Irena	Holubová	...
...			

I. Holubová, J. Kosek, K. Minařík, D. Novák. Big Data a NoSQL databáze. Praha: Grada Publishing, 2015. 288 p.

Data Model: Column Family (2)



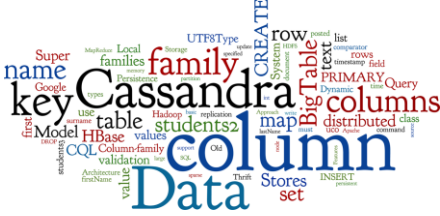
- Column family - example as JSON

```
{ // row (columns from a CF)
  "pramod-sadalage" : {
    firstName: "Pramod",
    lastName: "Sadalage",
    lastVisit: "2012/12/12"
}
```

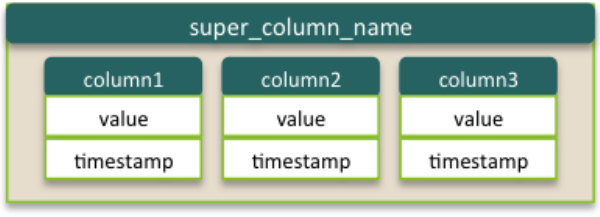
```
// row (cols from the same CF)
  "martin-fowler" : {
    firstName: "Martin",
    lastName: "Fowler",
    location: "Boston",
    active: "true" }
}
```

source: Sadalage & Fowler: NoSQL Distilled, 2012

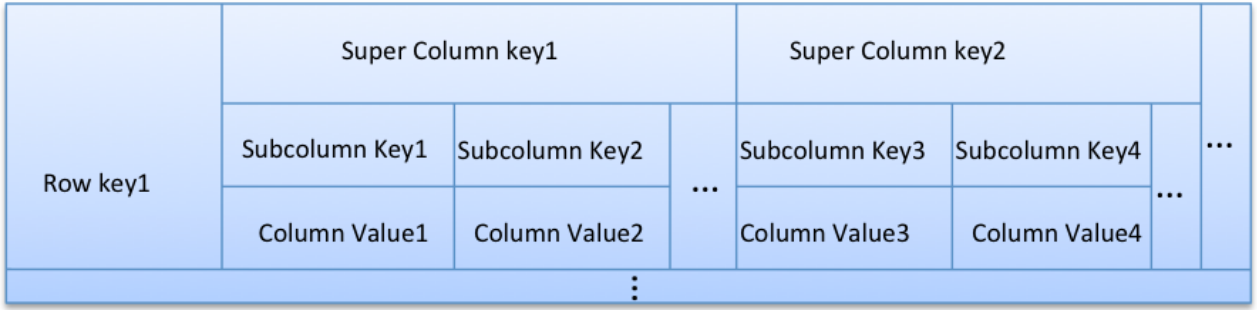
Data Model: Super Column Family



- **Super column**
  - A column whose value is composed of a map of columns
  - Used in some column-family stores (Cassandra 1.0)

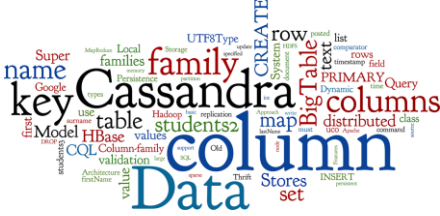


- **Super column family**
  - A column family consisting of super columns



http://www.ebaytechblog.com/2012/07/16/cassandra-data-modeling-best-practices-part-1/

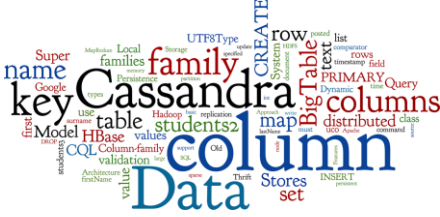
Super Column Family: Addresses



user_id (*row key*)	*super column key*			*super column key*			...
	subcolumn key	*subcolumn key*	...	*subcolumn key*	*subcolumn key*	...	...
	subcolumn value	*subcolumn value*	...	*subcolumn value*	*subcolumn value*	...	
1	home_address			work_address			
	city	street	...	city	street	...	
	Brno	Krásná 5	...	Praha	Pracovní 13	...	
4	home_address			temporary_address			
	city	street	...	city	PSČ		
	Plzeň	sady Pětatřicátníků 35	...	Praha	111 00		
...							

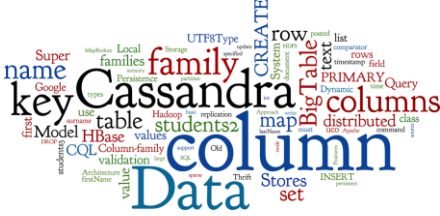
I. Holubová, J. Kosek, K. Minařík, D. Novák. Big Data a NoSQL databáze. Praha: Grada Publishing, 2015. 288 p.

Super Column Family: Generic




```
{ // row
  ”Cath”: {
    ”username”: { ”firstname”: ”Cath”, ”lastname”: ”Yoon” },
    ”address”: { ”city”: ”Seoul”, ”postcode”: ”1234” }
    }
  // row
  ”Terry”: {
    ”username”: { ”firstname”: ”Terry”, ”lastname”: ”Cho” },
    ”account”: { ”bank”: ”Hana”, ”accounted”: 1234 },
    ”preferences”: { ”color”: ”blue”, ”style”: ”simple” }
   }
 }
```

Data Model: Interpretation 1



1. Each column family = a relational table
   - with (a lot of) **null** values

row key | columns …

jbellis	name	email	address	state
	jonathan	jb@ds.com	123 main	TX

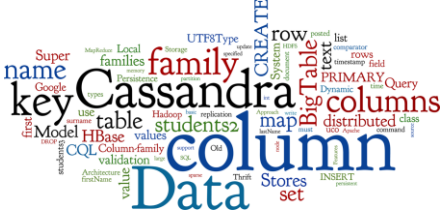
dhutch	name	email	address	state
	daria	dh@ds.com	45 2nd St.	CA

egilmore	name	email
	eric	eg@ds.com

Relational Model	Cassandra Model
Database	Keyspace
Table	Column Family (CF)
Primary key	Row key
Column name	Column name/key
Column value	Column value

http://www.ebaytechblog.com/2012/07/16/cassandra-data-modeling-best-practices-part-1/

Data Model: Interpretation 2



2. Column family = a map of maps (nested map)

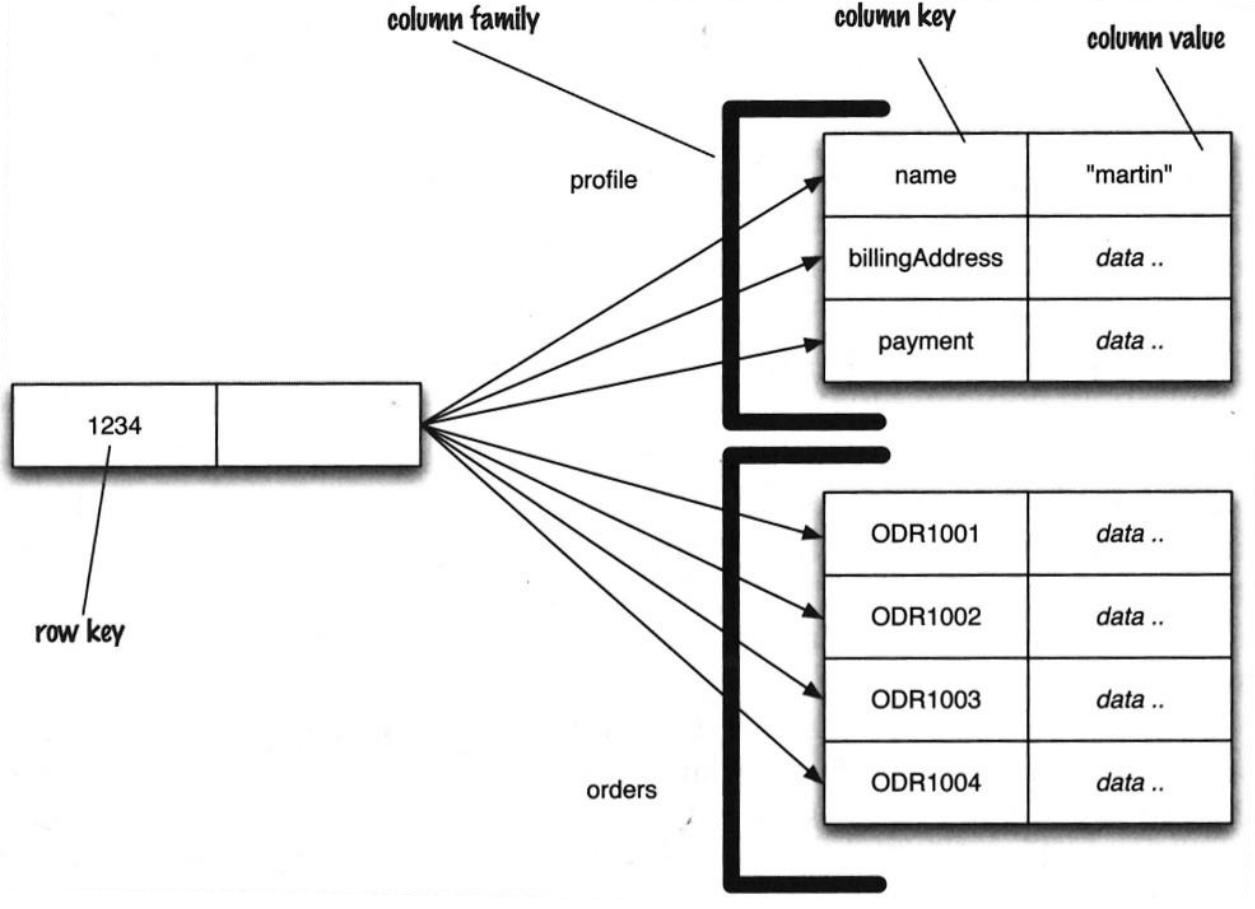
```
Map<RowKey, Map<ColumnKey, ColumnValue>>
```

- Super column family:

```
Map<RowKey, Map<SuperColumnKey,
                Map<ColumnKey, ColumnValue>>>
```

- The column-family data model can be viewed as *JSON-like documents with restrictions on the format*

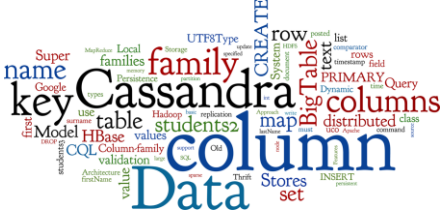
Example: Visualization



source: Sadalage & Fowler: NoSQL Distilled, 2012

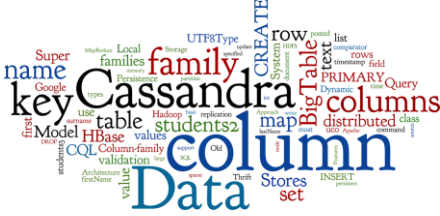
Column Family Stores

Agenda



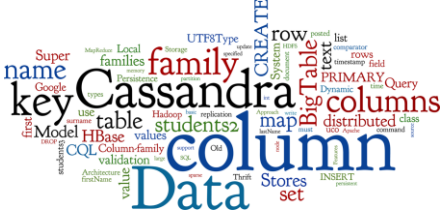
- Data Model
  - Column families, super columns, two points of view
- Column-family Stores
  - Google BigTable, Cassandra, HBase
- Cassandra in Detail
  - Cassandra data model 1.0 vs. 2.0
  - Cassandra Query Language (CQL)
  - Data partitioning, replication
  - Local Data Persistence
  - Query processing, Indexes, Lightweight Transactions

Column Family Stores: Features



- Data model: Column families
- System architecture
  - Data partitioning
- Local persistence
  - update log, memory, disk...
- Data replication
  - balancing of the data
- Query processing
  - query language
- Indexes

Representatives



Google BigTable

Cassandra

H·BASE

SCYLLA

accumulo™

Ranked list: http://db-engines.com/en/ranking/wide+column+store

BigTable

- Google's paper:
  - Chang, F. et al. (2008). Bigtable: A Distributed Storage System for Structured Data. ACM TOCS, 26(2), pp 1–26.
- Proprietary, not distributed outside Google
  - used in Google Cloud Platform

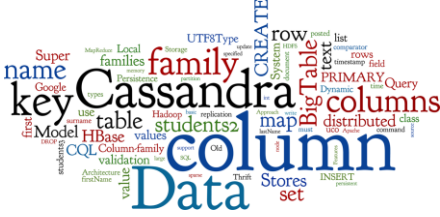
- Data model: column families as defined above

  "A table in Bigtable is a sparse, distributed, persistent multidimensional sorted map."

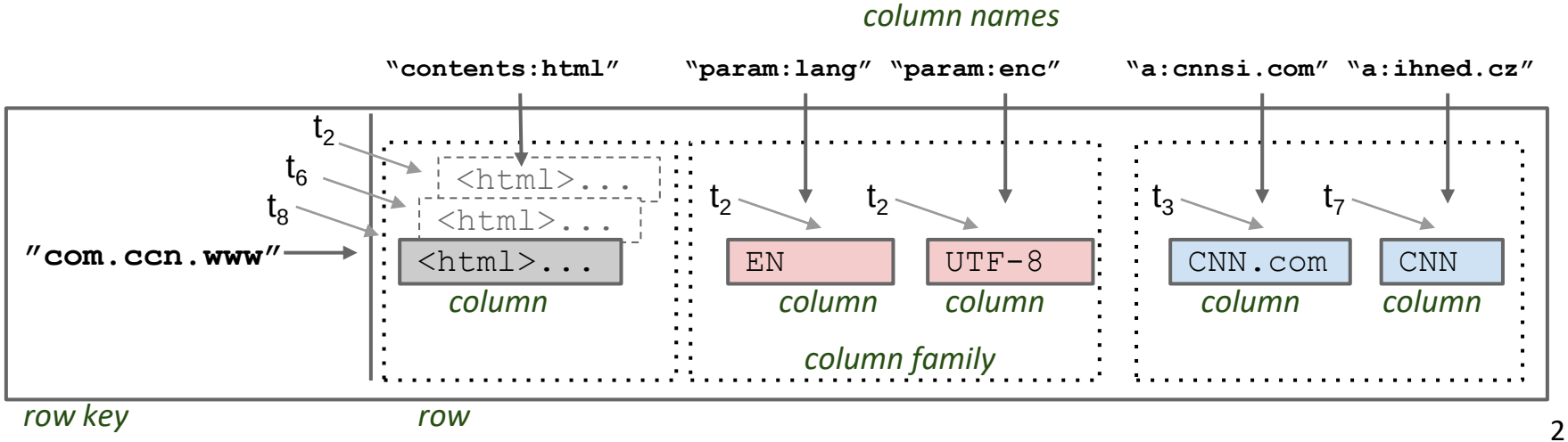
`(row:string, column:string, time:int64) → string`

http://en.wikipedia.org/wiki/BigTable

BigTable: Example



- "BigTable = sparse, distributed, persistent, multi-dimensional sorted map indexed by *(row_key, column_key, timestamp)*"



HBase

"Open source, non-relational, distributed database modeled after Google's BigTable. "

- Initial release: 2008
- Implementation: Java
  - Based on Apache Hadoop (HDFS)
- Open source: Apache Software License 2.0
- Systems: Linux, Unix, Windows (only via Cygwin)

"If you have hundreds of millions or billions of rows, then HBase is a good candidate. "

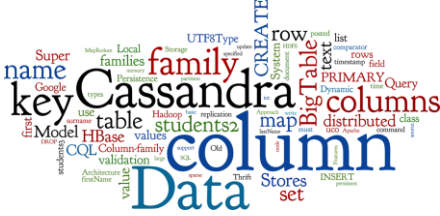
http://hbase.apache.org/

Cassandra

- Developed at Facebook
  - now under Apache Software License 2.0
- Initial release: 2008 (stable release 3.11 in 2017)
- Written in: Java
- OS: cross-platform
- Operations:
  - CQL (Cassandra Query Language)
  - MapReduce support (can cooperate with Hadoop)
- Professional support by DataStax
  - http://www.datastax.com/

http://cassandra.apache.org/

Agenda



- Data Model
  - Column families, super columns, two points of view
- Column-family Stores
  - Google BigTable, Cassandra, HBase
- Cassandra in Detail
  - Cassandra data model 1.0 vs. 2.0
  - Cassandra Query Language (CQL)
  - Data partitioning, replication
  - Local Data Persistence
  - Query processing, Indexes, Lightweight Transactions

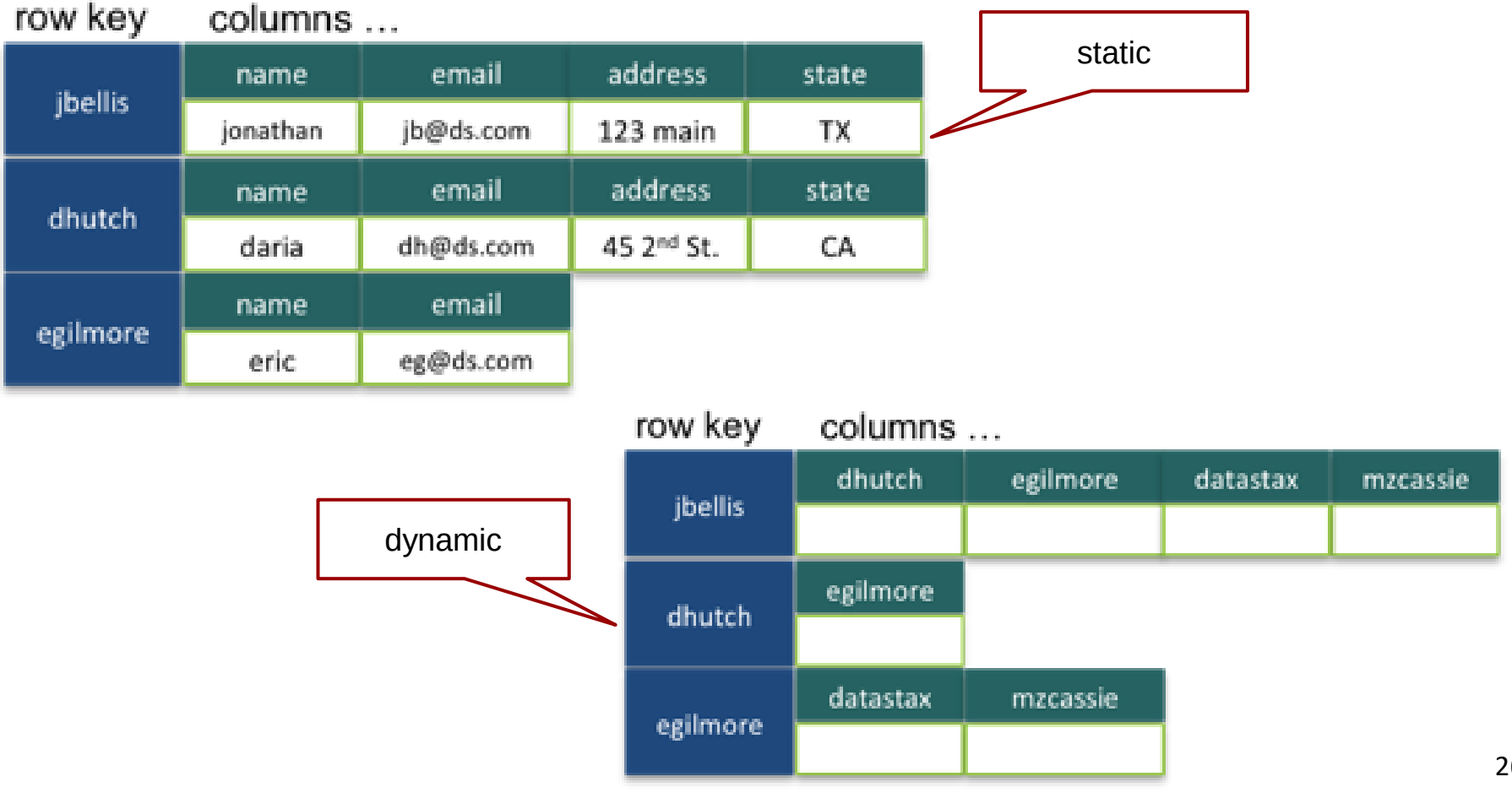
Cassandra 1.0: Data Model

- Column families, super column families
  - Can define metadata about columns
  - Now denoted as: Thrift API

- **Static** – similar to a relational database table
  - Rows have the same set of columns
    - Not required to have all of the columns set

- **Dynamic** – takes advantage of Cassandra's ability to use arbitrary column names

Cassandra 1.0: Data Model (2)



Cassandra 1.0: Column Families

- A key *must* be specified
- Data types for columns *can* be specified
- Options *can* be specified

```
CREATE COLUMNFAMILY Fish (key blob PRIMARY KEY);
CREATE COLUMNFAMILY FastFoodEatings (user text PRIMARY KEY)
    WITH comparator=timestamp AND default_validation=int;
CREATE COLUMNFAMILY MonkeyTypes (
    key uuid PRIMARY KEY,
    species text, alias text,
    population varint
) WITH comment='Important biological records'
  AND read_repair_chance = 1.0;
```

Cassandra 1.0: Column Families (2)

- **Comparator** = data type for a column name
- **Validator** = data type of a column value
  - or content of a row key

- Data types do not need to be defined
  - Default: BytesType, i.e., arbitrary hexadecimal bytes

- Basic operations: GET, SET, DEL

Cassandra 1.0: Data Manipulation


```
create column family users
    with key_validation_class = Int32Type
    and comparator = UTF8Type
    and default_validation_class = UTF8Type;


// set column values in row with key 7
set users[7]['login'] = utf8('honza');
set users[7]['name'] = utf8('Jan Novák');
set users[7]['email'] = utf8('jan@novak.name');

set users[13]['login'] = utf8('fantomas');
set users[13]['name'] = utf8('incognito');
```

Cassandra 1.0: Data Manipulation (2)


```
get users[7]['login'];
=> (name=login, value=honza, timestamp=142968223462000)

get users[13];
=> (name=login, value=fantomas, timestamp=142968224554000)
=> (name=name, value=incognito, timestamp=142968224555000)

list users;
RowKey: 7
=> (name=email, value=jan@novak.name, timestamp=14292682...)
=> (name=login, value=honza, timestamp=142968223462000)
=> (name=name, value=Jan Novák, timestamp=142968223471000)
-------------------
RowKey: 13
=> (name=login, value=fantomas, timestamp=142968224554000)
```

Cassandra: Sparse Tables

- CQL: Cassandra Query Language
  - SQL-like commands
    - CREATE, ALTER, UPDATE, DROP, DELETE, TRUNCATE, INSERT, …
  - Simpler than SQL

- Since CQL 3 (Cassandra 1.2)
  - Column -> cell
  - Column family -> table
- Dynamic columns (wide rows) still supported
  - CQL supports everything that was possible before
  - "Old" approach (Thrift API) can be used as well

http://www.datastax.com/documentation/cql/3.1/

Working with Tables


```
CREATE TABLE users (
    user_id int PRIMARY KEY,
    login text,
    name text,
    email text );

INSERT INTO users (user_id, login, name)
VALUES (3, 'honza', 'Jan Novák');

SELECT * FROM users;
 user_id | email | login | name
---------+-------+-------+-----------
       3 | null  | honza | Jan Novák
```

Tables: Dynamic Columns

- Values can use “collection” types:
  - **set** – unordered unique values
  - **list** – ordered list of elements
  - **map** – name + value pairs
    - a way to realize super-columns
- Realization of the original idea of free columns
  - Internally, all values in collections as individual columns
  - Cassandra can handle “unlimited” number of columns well

Tables: Dynamic Columns (2)


```
CREATE TABLE users (
    login text PRIMARY KEY,
    name text,
    emails set<text>, // column of type “set”
    profile map<text, text> // column of type “map”
)

INSERT INTO users (login, name, emails, profile)
VALUES ( 'honza', 'Jan Novák', { 'honza@novak.cz' },
    { 'colorschema': 'green', 'design': 'simple' }
);

UPDATE users
SET emails = emails + { 'jn@firma.cz' }
WHERE login = 'honza';
```

Dynamic Columns: Another Way

- Compound primary key

```
CREATE TABLE mytable (
    row_id int, column_name text, column_value text,
    PRIMARY KEY (row_id, column_name)
);

INSERT INTO mytable (row_id, column_name, column_value)
VALUES ( 3, 'login', 'honza');
INSERT INTO mytable (row_id, column_name, column_value)
VALUES ( 3, 'name', 'Jan Novák');
INSERT INTO mytable (row_id, column_name, column_value)
VALUES ( 3, 'email', 'honza@novak.cz');
```

Cassandra: Working with Data

Data Sharding in Columnar Systems

System	Terminology
BigTable	tablets
HBase	regions
Cassandra	partitions

část 1
(partition 1)

část 2
(partition 2)

user_id *(row key)*	login	name
1	honza	Jan...
4	david	David...
...		
1000	karel	Karel...
1001	irena	Irena...
1003	jirka	Jiří...
...		
2000		
....		

I. Holubová, J. Kosek, K. Minařík, D. Novák. Big Data a NoSQL databáze. Praha: Grada Publishing, 2015. 288 p.

Data Sharding in Cassandra

- Entries in each table are split by partition key
  - Which is a selected column (or a set of columns)
  - Specifically, the first column (or columns) from the primary key is the partition key of the table

```
CREATE TABLE tab ( a int, b text, c text, d text,
    PRIMARY KEY ( a, b, c)
);
```

```
CREATE TABLE tab ( a int, b text, c text, d text,
    PRIMARY KEY ( (a, b), c)
);
```

Data Sharding in Cassandra (2)

- All entries with the same partition key
  - Will be stored on the same physical node
  - => efficient processing of queries on one partition key

```
CREATE TABLE mytable (
    row_id int, column_name text, column_value text,
    PRIMARY KEY (row_id, column_name) );
```

- The rest of the columns in the primary key
  Are so called clustering columns
  - Rows are locally sorted by values in the clustering columns
    - the order for physical storing rows

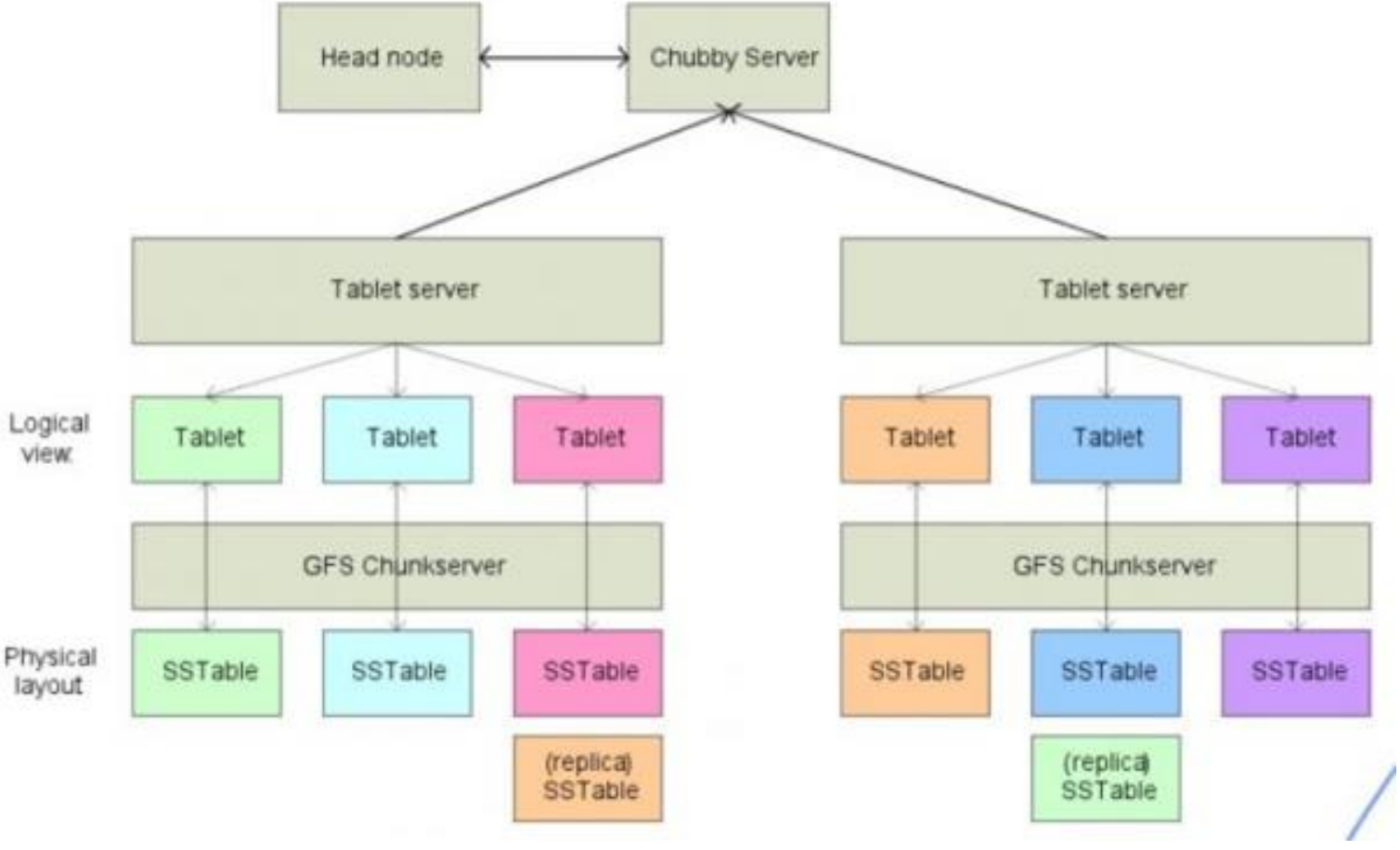
Data Replication

- Cassandra adopts peer-to-peer replication
  - The same principles like in key-value stores & document DB
  - Read/Write quora to balance between availability and consistency guarantees

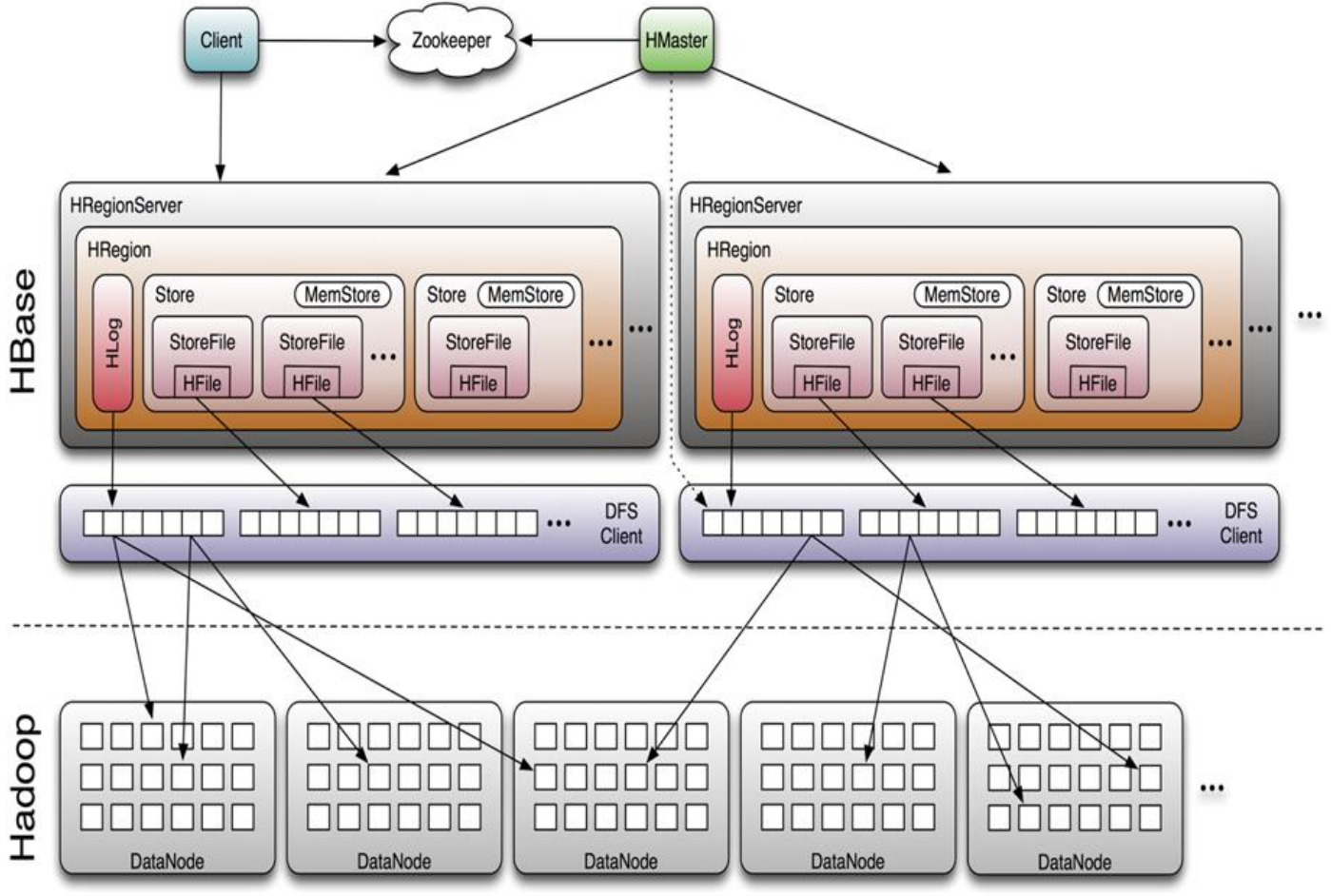
- HBase (and Google BigTable)
  - Physical data distribution & replication is done by the underlying distributed file system
  - HDFS, GFS (see below)

BigTable: Architecture

Google BigTable

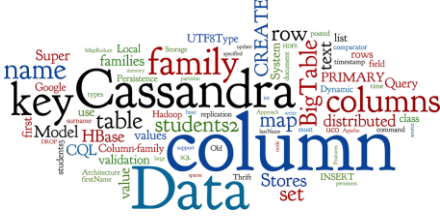


HBase: Architecture



http://bigdatariding.blogspot.cz/2013/10/hbase-overview-of-architecture-and-data.html

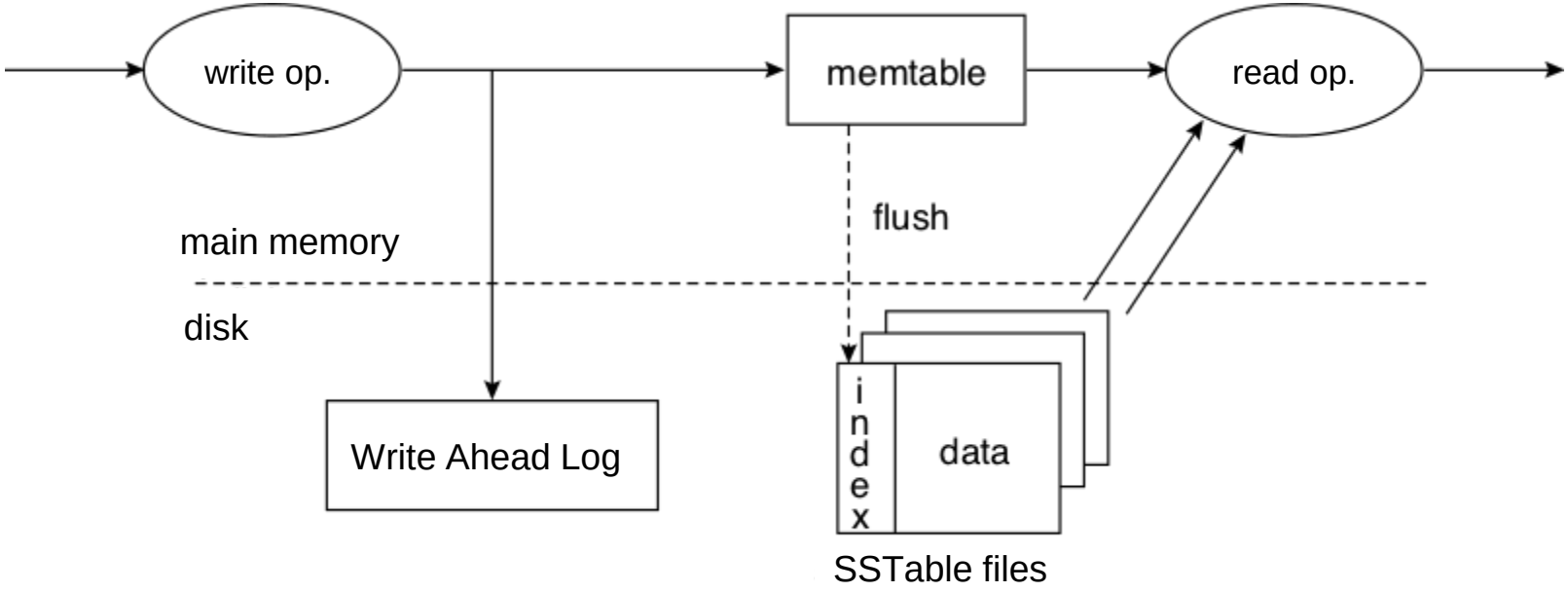
Local Persistence



- Organization of local data store at nodes
- Objectives:
  - Persistent, durable (ensure persistence after commit)
  - High performance of reads & writes

- Approach:
  - Memory tables
  - Append-only update log
  - SSTable disk-storage format: immutable
  - Compaction

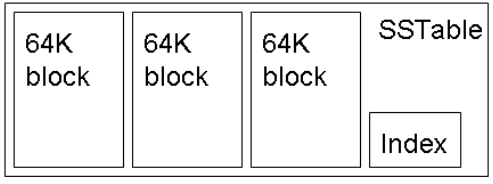
Local Persistence

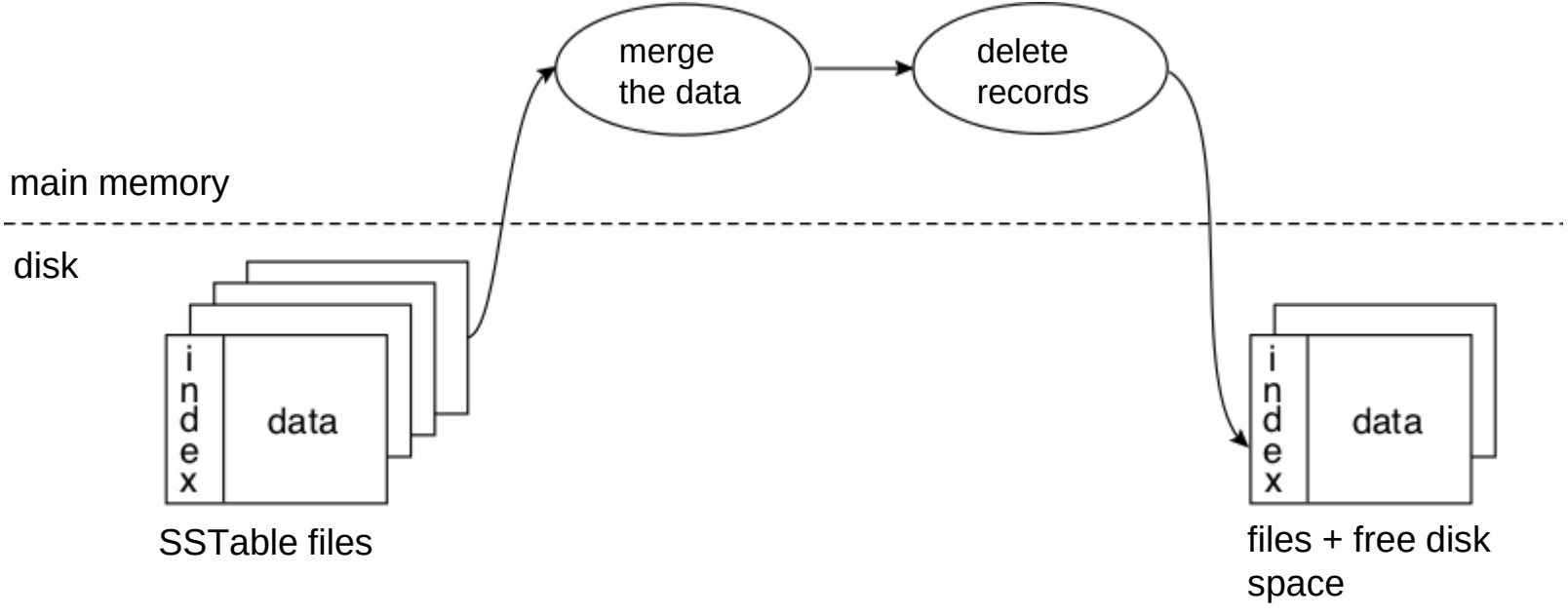


- Persistency + durability
- but also high throughput of write operations

Compaction, Consolidation

- Data in SSTables are immutable
- A regular process of data compaction





Cassandra: Querying

Cassandra Query Language (CQL)

- The syntax of CQL is similar to SQL
  - But search just in one table (no joins)

```
SELECT <selectExpr>
FROM [<keyspace>.]<table>
[WHERE <clause>]
[ORDER BY <clustering_colname> [DESC]]
[LIMIT m];
```

```
SELECT column_name, column_value
FROM mytable
WHERE row_id=3
ORDER BY column_name;
```

```
CREATE TABLE mytable (
     row_id int,
     column_name text,
     column_value text,
     PRIMARY KEY (row_id, column_name)
);
```

CQL: Limitations on "Where" Part

- The search condition can be:
  - on columns in the partition key
    - And only using operators = and IN

```
... WHERE row_id IN (3, 4, 5)
```

    - Therefore, the query hits only one or several physical nodes (not all)

```
CREATE TABLE mytable (
  row_id int,
  column_name text,
  column_value text,
  PRIMARY KEY
       (row_id, column_name)
);
```

  - on columns from the clustering key
    - Especially, if there is also condition on the partitioning key

```
... WHERE row_id=3 AND column_name='login'
```

    - If it is not, the system must filter all entries

```
SELECT * FROM mytable
WHERE column_name IN ('login', 'name') ALLOW FILTERING;
```

CQL: Limitations on “Where” Part (1)

- Other columns can be queried
  - If there is an index built on the column

- Indexes can be built also on collection columns (set, list, map)

```
CREATE INDEX ON users (emails);
```

  - And then queried by `CONTAINS` like this

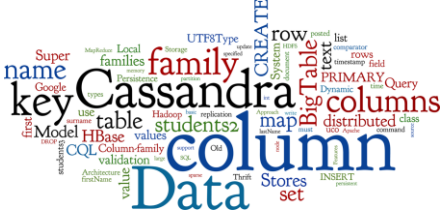
```
SELECT login FROM users
    WHERE emails CONTAINS 'jn@firma.cz';
SELECT * FROM users
    WHERE profile CONTAINS KEY 'colorschema';
```

Indexes

- Secondary indexes on any column
  - B^+-Tree indexes
  - User-defined implementation of indexes

```
CREATE INDEX ON users (emails);
```

Queries: HBase and Cassandra



- **Cassandra**: Direct support for secondary indexes
- **HBase**: Indirect ways to build secondary indexes

- In general, all systems support MapReduce
  - HBase & Cassandra: Hadoop MapReduce

- Comparison of HBase and Cassandra:

http://www.infoworld.com/article/2610656/database/big-data-showdown--cassandra-vs--hbase.html

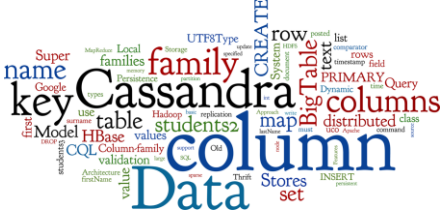
Transactions

- Cassandra 2.x supports “lightweight transactions”
  - compare and set operations
  - using Paxos consensus protocol
    - nodes agree on proposed data additions/modifications
    - faster than Two-phase commit protocol (P2C)

```
INSERT INTO users (login, name, emails)
VALUES ('honza', 'Jan Novák', { 'honza@novak.cz' })
IF NOT EXISTS;

UPDATE mytable SET column_value = 'honza@firma.cz'
WHERE row_id = 3 AND column_name = 'email'
IF column_value = 'honza@firm.cz';
```

Summary



- Column-family stores
  - are worth only for large data and large query throughput
  - two ways to see the data model:
    - large sparse tables or multidimensional (nested) maps
  - data distribution is via row key
    - analogue of document ID or key in document or key-value stores
  - efficient disk + memory local data storage

- Cassandra
  - CQL: structured after SQL, easy transition from RDBMS

Questions?

Please, ANY questions?

References

- I. Holubová, J. Kosek, K. Minařík, D. Novák. Big Data a NoSQL databáze. Praha: Grada Publishing, 2015. 288 p.
- RNDr. Irena Holubova, Ph.D. MMF UK course NDBI040: Big Data Management and NoSQL Databases
- Chang, F. et al. (2008). Bigtable: A Distributed Storage System for Structured Data. ACM TOCS, 26(2), pp 1–26.
- http://www.datastax.com/documentation/cassandra/1.2/
- http://www.datastax.com/documentation/cassandra/2.0/
- http://wiki.apache.org/cassandra/
- http://hbase.apache.org/