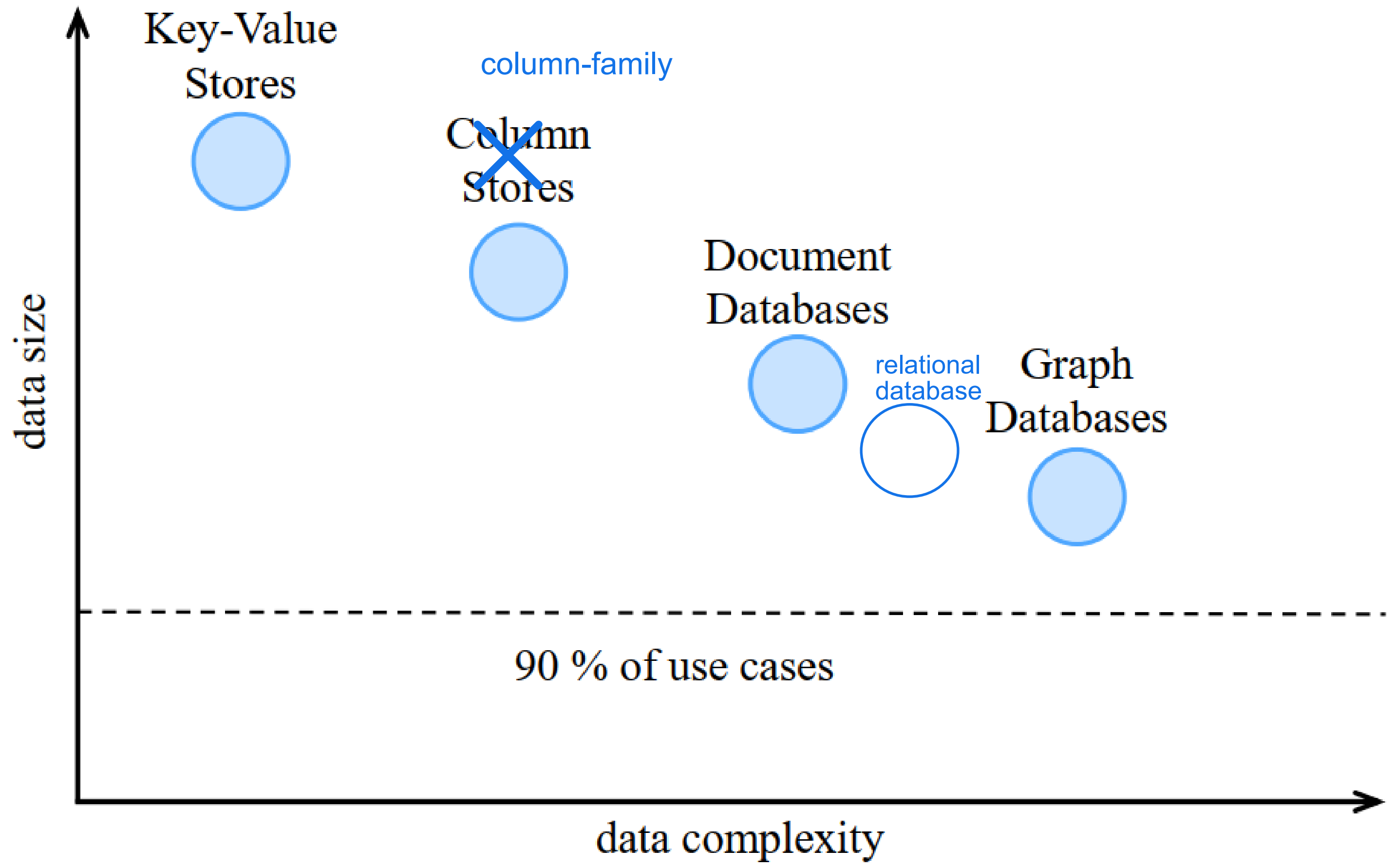
KUTAISI INTERNATIONAL UNIVERSITY

8
Column Store Databases
Column-Family Databases

Readings: [Me] chapter 7.3, [Ha] chapter 10

Data Size - Data Complexity



Column-Family Databases

What is in a name?

Column Database / Column Store column store is a variation of relational database

Column-Oriented Database / Columnar Database

Column-Family Database / Wide-Column Database

NoSQL database,
P2P distributed database

Relational Column Store versus Column-Family Databases Fundamentally Different

SQL Relational Databases with Column Store Option	NoSQL Column Familiy Databases
Relational databases that offer a column-by-column storage in addition to the traditional row-by-row storage.	groups and stores data in column families. Column families are physical storage units.
Oracle, MS SQL, MySQL, MariaDB, SAP Hana, MonetDB (column store only)	Cassandra, Hbase, ScyllaDB, Google BigTable
Relational data model (relational scheme) and SQL query language	
Relational database used for analytical, read-heavy purposes	P2P distributed database, heavy write-workload possible

Relational Database Column Storage

Row Format

- RDBMS are –logically - two-dimensional: columns (attributes) and rows (tuples).
- However, the storage of the data happens one-dimensionally – in a file (heap file or B-Tree).
  → how to map from two dimensions to one dimension for storage?

By placing all the elements of the first row, then the second row and so on in a single sequence.

teacherID	name	postal	email	DoB	gender	education	remark
41	Krawinkel	10045	krawinkel@wir_wissen_alles.xx	1978-09-15	f	Master	
42	Kaiser	10045	kaiser@zirkus.xx	2001-11-30	f	Ph.D	
43	Kern	10045	kern@immer_da.xx	1977-06-11	m	Ph.D	
44	Schuster	10004	schusterl@flieg.xx	1975-03-27	f	Master	
45	Newman	10010	new@kriech.xx	1995-11-20	m	Ph.D	

heap file storage

Header, 41, Krawinkel, 10045, krawinkel@wir_wissen_alles.xx ,1978-09-15,f,Master, Header, 42, Kaiser, 10045, kaiser@zirkus.xx ,2001-11-30,f,Ph.D, Header, 43, Kern 4,3, 10045, kern@immer_da.xx, 1977-06-11, m, Ph.D., Header, 44, Schuster 10004, schusterl@flieg.xx , 1975-03-27,f,Master, Header, 45, Newman, 10010, new@kriech.xx, 1995-11-20,m, Ph.D.

RDBMS Row-Store (row-oriented storage)

- Traditionally, RDBMS store data row-oriented - that is, complete tuples (row-store).
- Row-oriented storage means:
    - All values of one tuple are stored together.
- It is optimized for operations that read all or most information of a row:
    - select * from teacher where t_name = "Summer"
- It is optimized for operations that write all or most information of a row:
    - INSERT INTO `lesson` (`teacherID`, `lessonDate`, `userName`, `subject`, `lessonDone`) VALUES ('3', '2023-03-26 18:45:30.000000', 'Wolf', 'CH', '0');

- NoSQl document databases store whole documents one after the other – somewhat similar to RDBMS row-oriented store.

Column Store

teacherID
41
42
43
44
45

name
Krawinkel
Kaiser
Kern
Schuster
Neumann

zip
10045
10045
10045
10004
10010

rating
4
5
3
5
5

41,42,43,44,45;
Krawinkel,Kaiser,Kern,Schuster,Neumann;
10045,10045,10045,10004,10010;
4,5,3,5,5

- In recent years, more and more RDBMS also offer to store data column-wise - that is, all values of one column are stored together.
- Column-wise storage means:
  - All values of one column are stored together (mariadb: one file per column)
  - A query does not load / read whole rows into memory but only queried columns.
- Use Case?

  statistics, analysis
- Give a query example that a relational column store is optimized for: when we query one or two columns

  select count(*) from teacher group by rating

  if we had for example millions of rows and 20 columns in teacher column store would only load 1 column and row-store would load all 20 columns

KUTAISI INTERNATIONAL UNIVERSITY

Column Store

	teacherID
1	41
2	42
3	43
4	44
5	45

name
Krawinkel
Kaiser
Kern
Schuster
Neumann

zip
10045
10045
10045
10004
10010

rating
4
5
3
5
5

[41,42,43,44,45]
[Krawinkel,Kaiser,Kern,Schuster,Neumann]
[10045,10045,10045,10004,10010]
[4,5,3,5,5]

4

select * from teacher where teacherID = 44

How is this query executed?

by array position

teacherID=44 has array position 4, then we go to name and look for array position 4(Schuster), then same position in zip(10004) and same position in rating(5)

Why not store a TID with each column value?

it takes up space

usually we have a lot of columns that store few values such as gender column, rating column, etc. and these kind of columns are great for compression. And then with compression they become really fast again and if we had TID then it would kill compression idea. for example if we had in rating column [5,5,5,5,5,5,5,5...] this can be compressed easily. But if we add TID and we then have for TID 1 value 5, for TID 2 value 5, for TID 3 value 5,... then these tuples are not identical anymore and can't be compressed.

in cocurrent writes row-store is preferred because instead of locking many columns it's easier to lock entire row

Column Compression

profID	name	zip	rating
41	Krawinkel	10045	4
42	Kaiser	10045	5
43	Kern	10045	3
44	Schuster	10004	5
45	Neumann	10010	5

41,42,43,44,45;
Krawinkel,Kaiser,Kern,Schuster,Neumann;
10045,10045,10045,10004,10010;
4,5,3,5,5

- Columns often hold redundant values and / or a very limited number of different values. Those columns are suitable for compression. rating is good candidate for compression
- Popular compression method for columnar storage are bitmaps: each possible value gets a bitmap assigned. Each bitmap has same length as the column vector. if we have 1000 rows in a column then bitmap is length of 1000
- Run-length encoding is used to further compact the bitmaps.

Bitmaps + Run-Length Encoding

- bitmap index on column rating with values {1,2,3,4,5}.
- The column vector has length 5 (5 tuples, Ids 41 - 45).
- Each rating value {1,2,3,4,5} gets a bitmap with length 5
- For each row, a 1 is filled into the bitmap if the row has that value; a 0 if it does not have the value.

	1	2	3	4	5
41	0	0	0	1	0
42	0	0	0	0	1
43	0	0	1	0	0
44	0	0	0	0	1
45	0	0	0	0	1

rating
4
5
3
5
5

Run_length Encoding

this line is stored	5	5	2,1	0,1	1,1,1,2
	5 zeros	5 zeros	2 zeros, 1 one, rest zero	0 zero, 1 one, rest zero	1 zero, 1 one, 1 zero, 2 ones

What advantge does compression have?

increases throughput

Compressed data = fewer bytes read = faster input

Column-oriented storage and column families

"... column families Cassandra and HBase have a concept of column families, which they inherited from Bigtable [9]. However, it is very misleading to call them column-oriented: within each column family, they store all columns from a row together, along with a row key, and they do not use column compression. ..."

[Kl, p. 99]

KUTAISI INTERNATIONAL UNIVERSITY

NoSQL

Column-Family Databases

Create Database Command for RDBMS (Postgres) and Column-Family (Cassandra)

database=keyspace

Postgres:

```
CREATE DATABASE lesson
    WITH
    OWNER = postgres
    ENCODING = 'UTF8'
    LC_COLLATE = 'C'
    LC_CTYPE = 'C'
    LOCALE_PROVIDER = 'libc'
    TABLESPACE = pg_default
    CONNECTION LIMIT = -1
    IS_TEMPLATE = False;
```

Cassandra:

```
CREATE KEYSPACE lesson
WITH replication = {
  'class': 'SimpleStrategy',
  'replication_factor': 3
};
```

Create Table Command for RDBMS (Postgres) and Column-Family (Cassandra)

Postgres:

```
CREATE TABLE IF NOT EXISTS public.teacher ((
   t_id integer NOT NULL DEFAULT nextval('"teacher_tID_seq"'::regclass),
   t_name character varying(30) COLLATE pg_catalog."default" NOT NULL,
   t_mail character varying(50) COLLATE pg_catalog."default" NOT NULL,
   t_postalcode integer NOT NULL,
   t_gender gender,
   t_education education NOT NULL,
   t_payment integer NOT NULL DEFAULT 0,
CONSTRAINT teacher_pkey PRIMARY KEY (t_id)
)
```

Cassandra:

```
CREATE TABLE lesson.teachers (
  t_email text,
  t_name text,
  t_phone text,
  address frozen<address>,
  t_subjects set<text> ,
  t_education text,
  PRIMARY KEY (t_email)
  WITH comment = 'Q1. Find information about a teacher';
```

Table definition in Postgres and Cassandra looks similar. That is the problem.

Concepts and behavior are very different → data design needs to be approached differently,
application development is different.

RDBMS versus Column-Family Concepts

RDBMS:

schema

referential integrity(PKs, FKs)

normalization --> minimize redundency--> keep consistency

join

ACID

Column-Family:

schema

no referential integrity

denormalized, redundant data

no join --> query needs to be answered by one table because it is stored on one node

CAP

Designing for optimal storage
Sorting is a design decision

https://cassandra.apache.org/doc/latest/cassandra/developing/data-modeling/data-modeling_rdbms.html

Summary Data Model Cassandra

- Query-first model, starting point is query which is the opposite of row-store systems which are often Data-First meaning They load rows and extract what the query needs later.
- de-normalized, redundancy accepted
- no joins → design has to consider storage model, in order "to minimize the number of partitions that must be searched in order to satisfy a given query. Because the partition is a unit of storage that does not get divided across nodes, a query that searches a single partition will typically yield the best performance." https://cassandra.apache.org/doc/latest/cassandra/developing/data-modeling/data-modeling_rdbms.html
- Sorting is part of the design
  "The sort order available on queries is fixed, and is determined entirely by the selection of clustering columns you supply in the CREATE TABLE"

Defining Application Queries

Q1: Return all information about a given, specific teacher

cassandra table from slide 15 answers this Q1

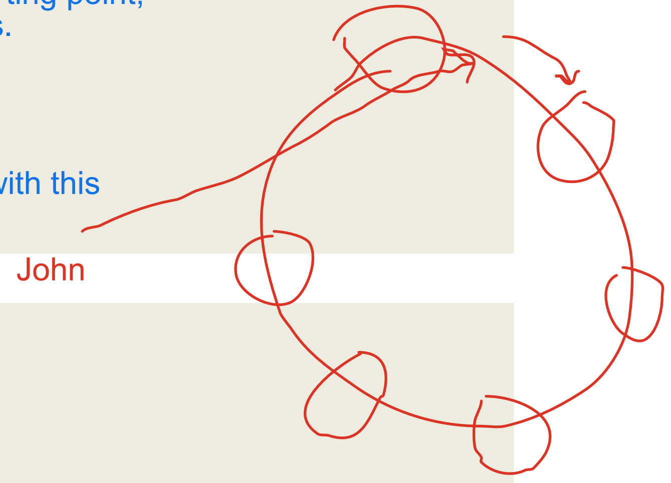
Q2: Return the students of a given, specific teacher

table on slide 20 answers Q2

Q3: Return a teacher list by given, specific postalcode

these given specific ... refer to starting point, meaning it relates to partition keys.

and whatever we want to return with this key John must be on that node.



How are the queries mapped on the components of the Cassandra data model:
→ a table for each query

- Rowkeys (=partition keys) controls distribution onto the rind
- column-families (attribute group)
- Columns (attribute – value)

Cassandra Data Model

Keyspace (=database)

→ replication factor

→ replication strategy

→ column-families (tables)

→ A column family consists of a collection of columns in rows

→ each column family must have a partition key (row-key)

→ each column family may have a clustering key in addition to the partition key.

for distribution

Create Table Command Column-Family (Cassandra)

```
CREATE TABLE lesson.teachers (
  t_email text,
  t_fname text
  t_lname text,
  t_phone text,
  t_address frozen<address>,
  t_subjects set<text> ),
  t_education text,
  PRIMARY KEY (t_email) )
  WITH comment = 'Q1. Find information about a teacher';
```

```
CREATE TABLE lesson.students_by_teacher (
  t_email text,
  s_email text,
  t_fname text,
  t_lname text,
  s_fname text,
  s_lname text,
  s_phone text
  s_subjects set<text> ,
  PRIMARY KEY ((t_email), s_email) )
  WITH comment = 'Q2. Find the students of a given teacher'
  AND CLUSTERING ORDER BY (s_email DESC) ;
```

t_email is partition key
s_email is clustering key

we have a lot of redundent information here because teacher info (t_fname, t_lname) is repeated for every student, but it's there because we want to retrieve all the info in one query.

The empty column-family schema (table) is replicated to all nodes. When a data object is assigned to a specific node, the schema is used and the storage starts. (How is the storing done?) LSM - storage starts with memtable and when it reaches certain size it will be flushed.

Column-Family Data Model

Figure 10-5 illustrates such a wide column family. In the FRIENDS column family, we have a variable number of columns, each corresponding to a specific friend. The name of the column corresponds to the name of the friend, while the value of the column is the friend's email. In this example, both Guy and Joanna have a common friend John, so each share that column. But other columns that represent friends who are not shared, and those columns appear only in the row required.

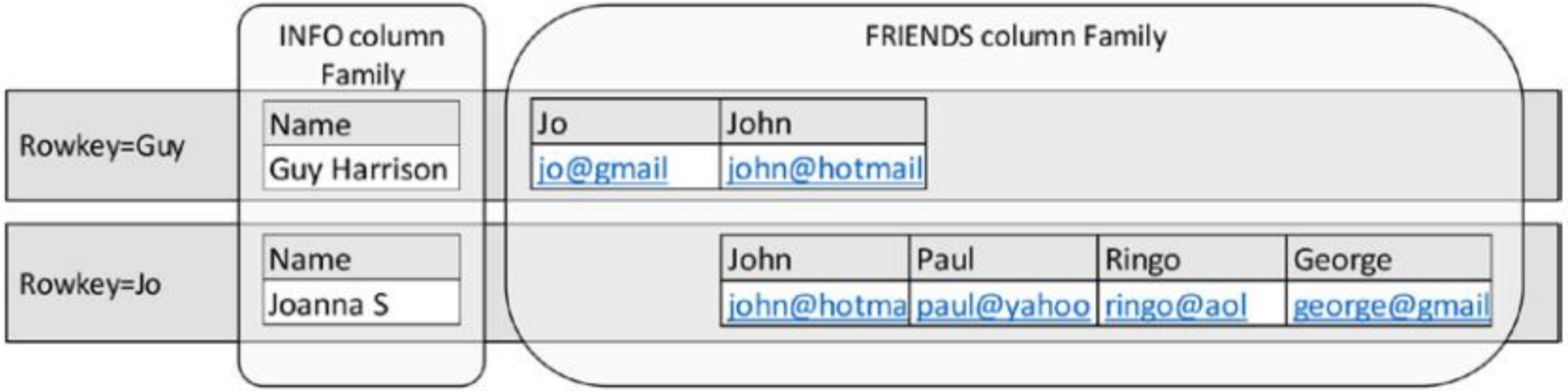


Figure 10-5. *Wide column family structure*

[Ha], page 152]

	Partition key	Info	Students
summer is stored on n2	summer@kiu.edu.ge	fname: sally, lname:summer, DoB: 1990-05-05, … … subjects [EN, GE,DE, MA]	s_email: s1@kiu.edu.ge, t_fname: sally, t_lname:summer, s_fname: giorgi, s_lname: daravadze, s_phone: 599-123-456, s_subjects: [EN,MA]
doe is stored on n10	doe@kiu.edu.ge	fname: john lname:doe DoB: 1999-12-02 … … subjects [CS, PH, MA]	s_email: s2@kiu.edu.ge, t_fname: john, t_lname: doe, s_fname: mariam, s_lname: chikhvadze s_phone: 599-123-654, s_subjects: [CS,MA]

Partition key

summer@kiu.edu.ge

Info

fname: sally,
lname:summer,
DoB: 1990-05-05,
…
…
subjects [EN, GE,DE, MA]

Students

s_email: s1@kiu.edu.ge,
t_fname: sally,
t_lname:summer,
s_fname: giorgi,
…
s_subjects: [EN,MA]

s_email: s5@kiu.edu.ge,
t_fname: sally,
t_lname:summer,
s_fname: ana,
…
s_subjects: [EN,DE]

s_email: s10@kiu.edu.ge,
t_fname: sally,
t_lname:summer,
s_fname: levan
…
s_subjects: [EN,DE]

clustering key sorts so we have sorted s_email:
s1@...,s2@...,s3@...

in each student column we also have teacher
name which is redundant

Cassandra Column-Family Keys

- Each Cassandra column family (table) has a primary key.
- The primary key can be a compound key:
    - 1st part of the primary key: partition key, which is hashed to determine storage node on the token ring - required
    - 2nd part of the primary key: clustering key that sorts the data within the partition key optional
    - Both, partition and clustering key can – again - be composite keys.
    - Sorting order of the memtable: partition key – clustering key

Examples:
Primary key for column family teachers: atomic or compound?
it is atomic because it only has partition key
Primary key for column family students_by_teacher: atomic or compound?
it is compound because it has both partition and clustering keys

Addressing is done by using a combination of column_family: partition key: clustering column key.

Partition key

Info

Students

summer@kiu.edu.ge

winter@kiu.edu.ge

angel@kiu.edu.ge

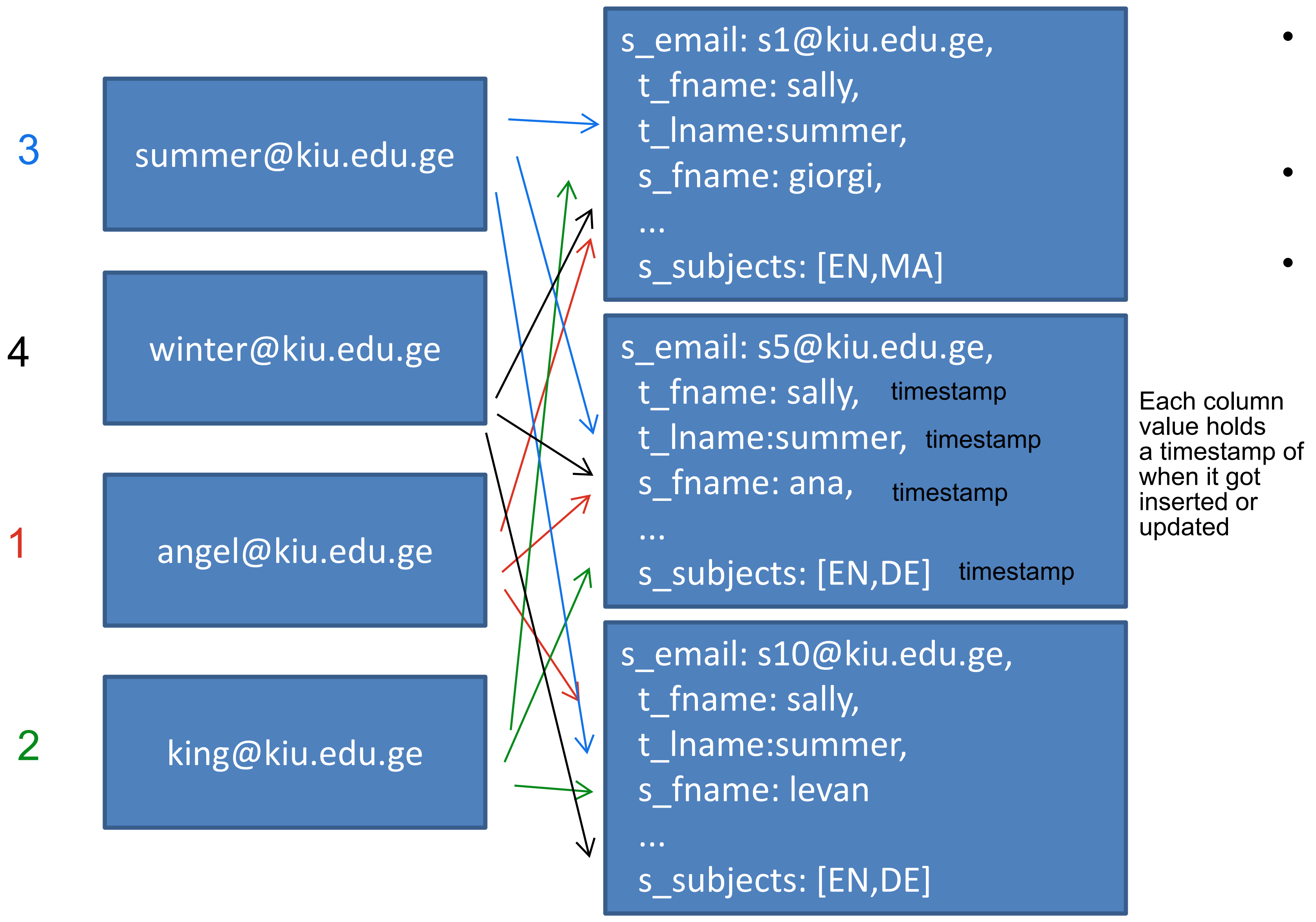
king@kiu.edu.ge

fname: sally,
lname:summer,
DoB: 1990-05-05,
…
…
subjects [EN, GE,DE, MA]

Assume that these partiton keys all are hashed to the same node

then on the node it will all be stored sorted:

angel
king
summer
winter



- Assume that these partiton keys all are hashed to the same node
- Sssume that the teachers have the same students
- What would the sorting in the memtable be?

angel:
s1
s5
s10
king:
s1
s5
s10
summer:
s1
s5
s10
winter:
s1
s5
s10

Cassandra Column-Family Data Model

- All column families need to be predefined
- Each column family is created as a table.
- The table (column family) needs to hold all columns (data) to answer a query
  Why? because we have it distributed on token ring so we need to have them all in one table to answer the query
- Column-families are the units by which data is stored → each column family results in a SSTable .
- A query does not need to load the whole row, that is all column families, if a rowkey (partition key) controls multiple column families. we only need the “Students” part of the partition. we do not need to load the teacher info block. In our case we are not loading the entire row (the whole partition). we are just reading the "student" column family.
- All column families of the same partition key (rowkey) are stored on the same node (why?) but not in the same SSTable because they have same hash but maybe not in the same segment
- Each column value holds a timestamp. What for?

  for concurrent writes

Summary Column-Family Database (Cassandra)

- distibruted "token ring" databases with no single point of failure
- AP systems with eventual consistency
- limited querying possibilities
- LSM storage structure

Use Case Examples

Storing user history of a service (e.g. streaming service, driving service)

```
CREATE TABLE history_by_user (
    user_id UUID,
    usage_timestamp TIMESTAMP, show_id TEXT,
    title TEXT,
    genre TEXT,
    duration_watched INT,
    location TEXT,                    usage_timestamp
    PRIMARY KEY ((user_id), ~~view_timestamp~~)
) WITH CLUSTERING ORDER BY (usage_timestamp DESC);
```

for example netflix keeps information of timestamps of when did you watch something, what did you watch, duration how long did you watch, etc.

Other use cases:
- Global Press Agencies storing their articles / press releases by date / topic
- Social media: feeds by followed persons per user