CQL
(Cassandra Query Language)

Dr. Rajiv Misra
Dept. of Computer Science & Engg.
Indian Institute of Technology Patna
rajivm@iitp.ac.in

Preface

Content of this Lecture:

- In this lecture, we will discuss CQL (Cassandra Query Language) Mapping to Cassandra's Internal Data Structure.

What Problems does CQL Solve?

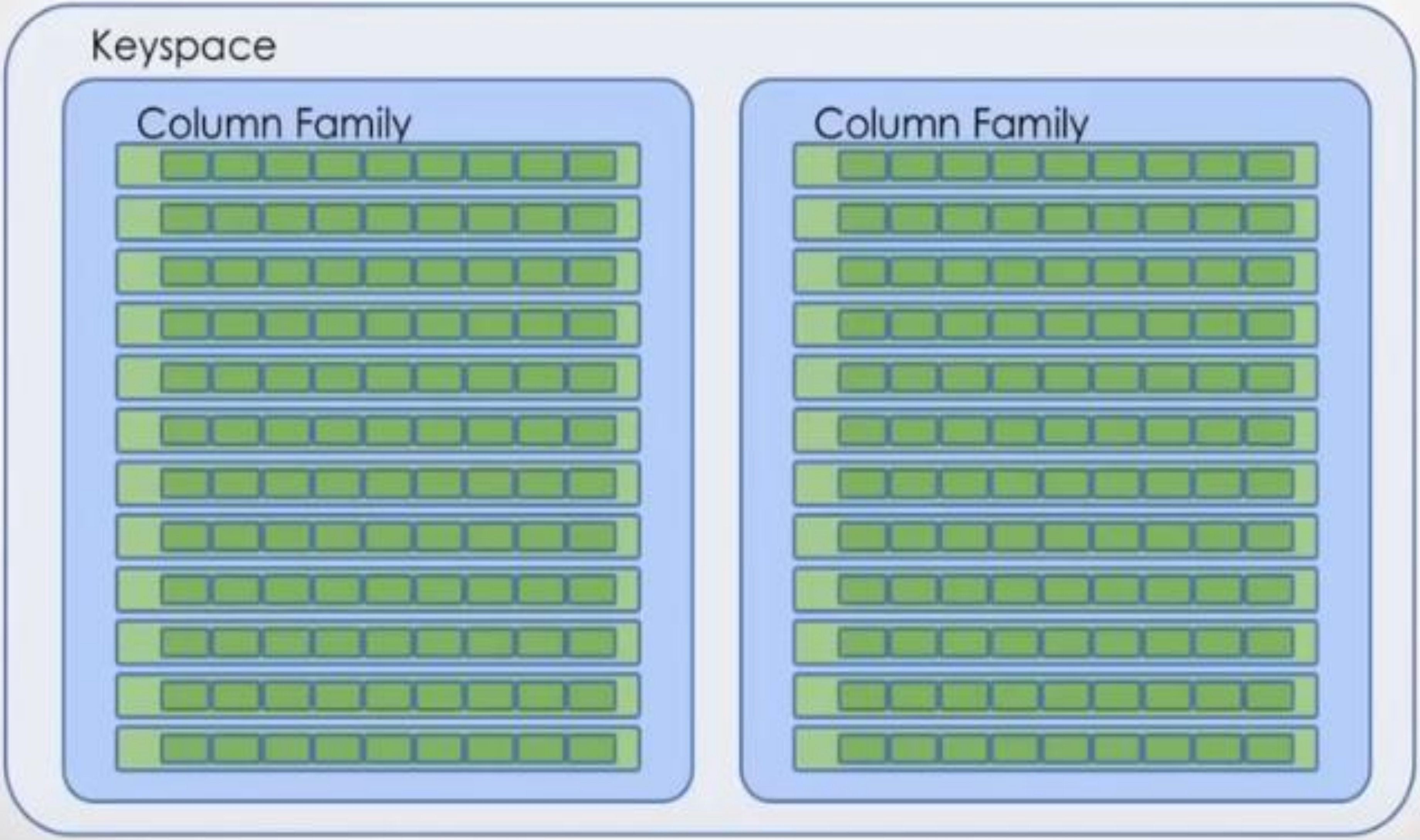
- **The Awesomeness that is Cassandra:**
  - Distributed columnar data store
  - No single point of failure
  - Optimized for availability (through "Tunably" consistent
  - Optimized for writes
  - Easily maintainable
  - Almost infinitely scalable

What Problems does CQL Solve? (Contd.)

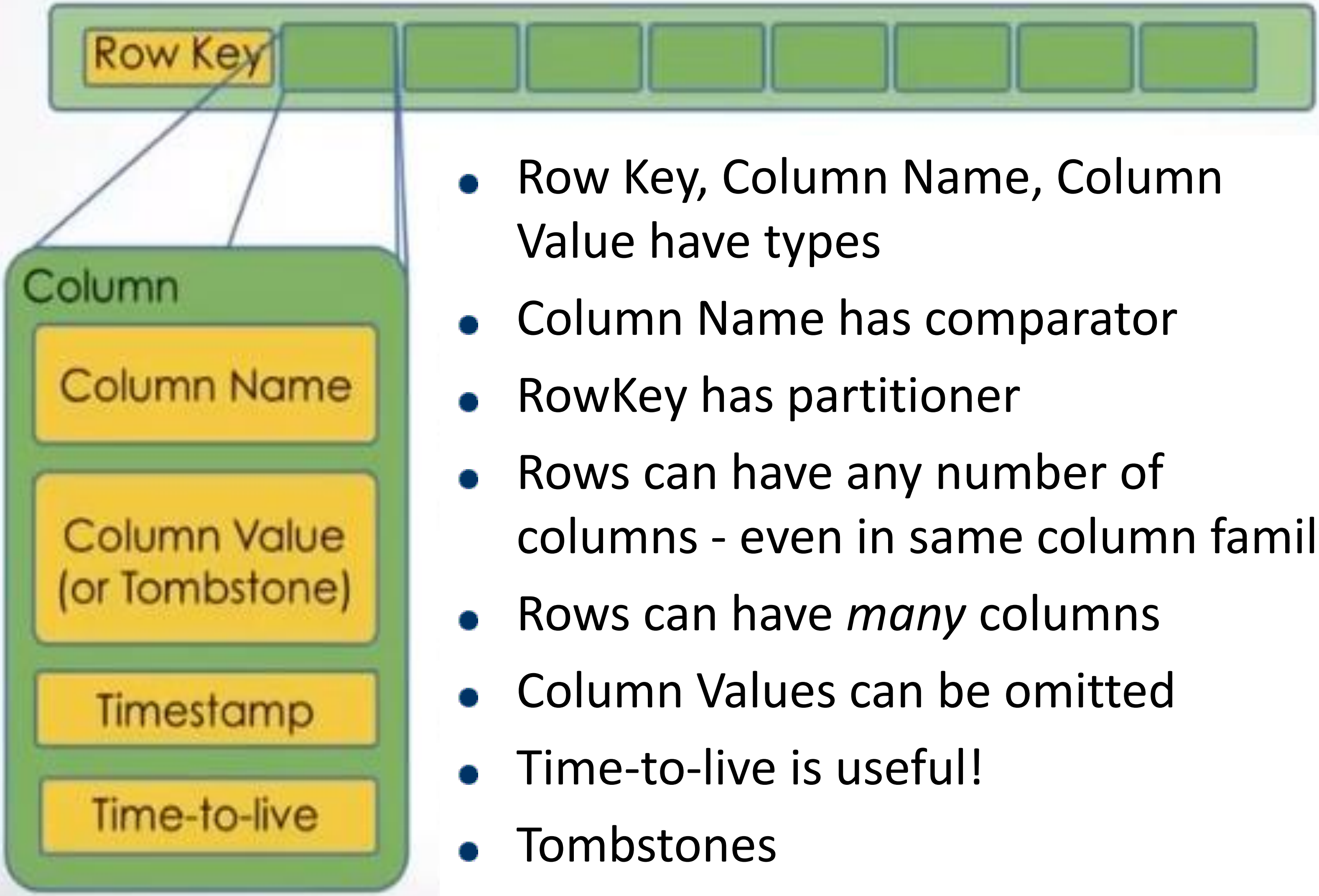
- **Cassandra's usability challenges**
  - NoSQL: "Where are my JOINS? No Schema? De-normalize!?"
  - BigTable: "Tables with millions of columns!?"

- **CQL Saves the day!**
  - A *best-practices* interface to Cassandra
  - Uses familiar SQL-Like language

C* Data Model

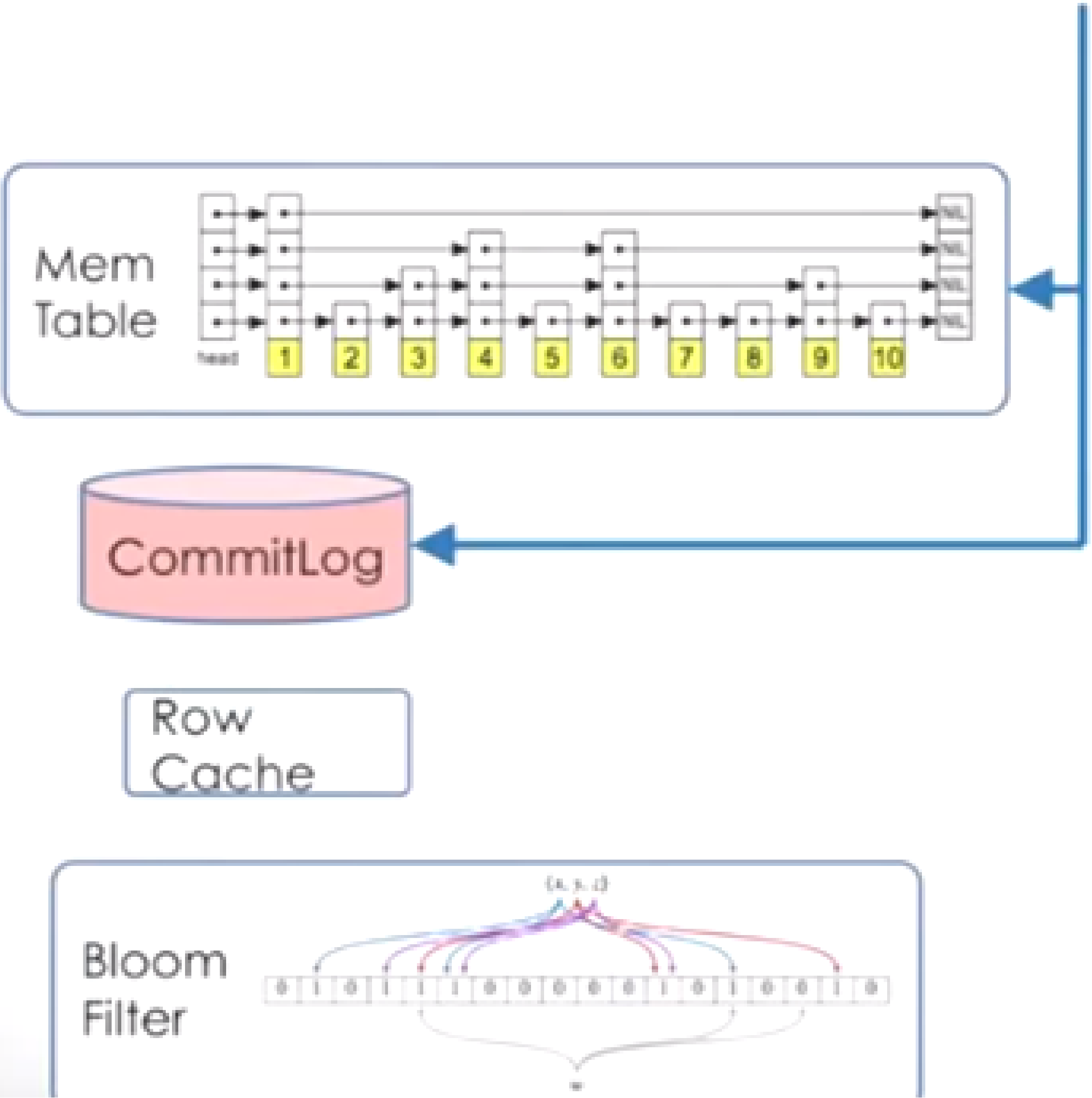


C* Data Model (Contd.)



- Row Key, Column Name, Column Value have types
- Column Name has comparator
- RowKey has partitioner
- Rows can have any number of columns - even in same column family
- Rows can have *many* columns
- Column Values can be omitted
- Time-to-live is useful!
- Tombstones

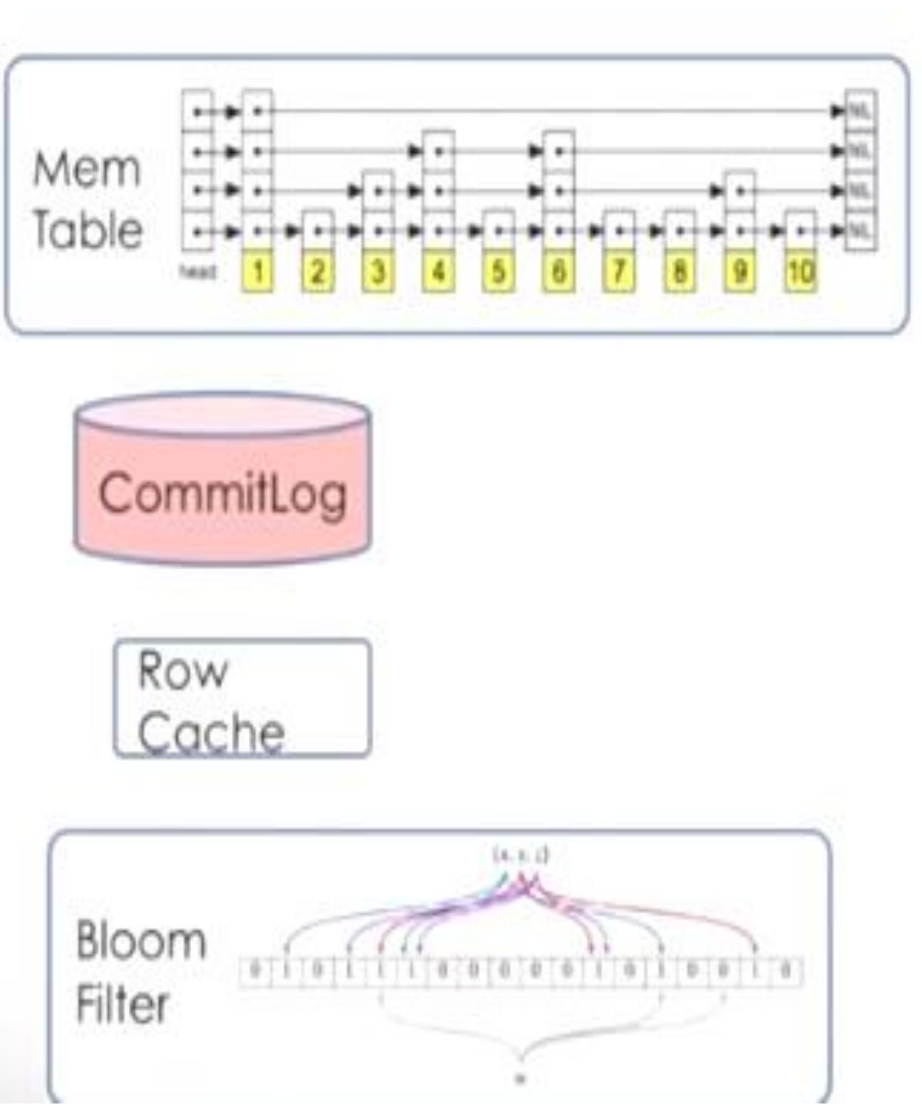
C* Data Model: Writes



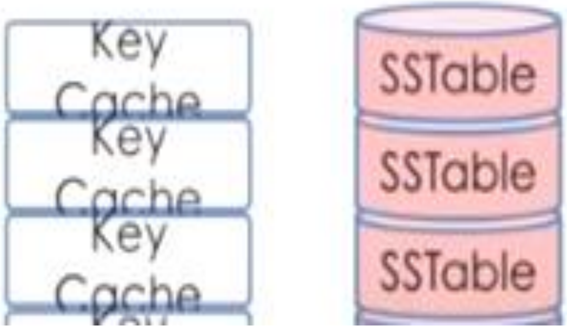
- Insert into MemTable
- Dump to CommitLog
- No read
- Very Fast!
- Blocks on CPU before O/I!

Key Cache
Key Cache
Key Cache
SSTable
SSTable
SSTable

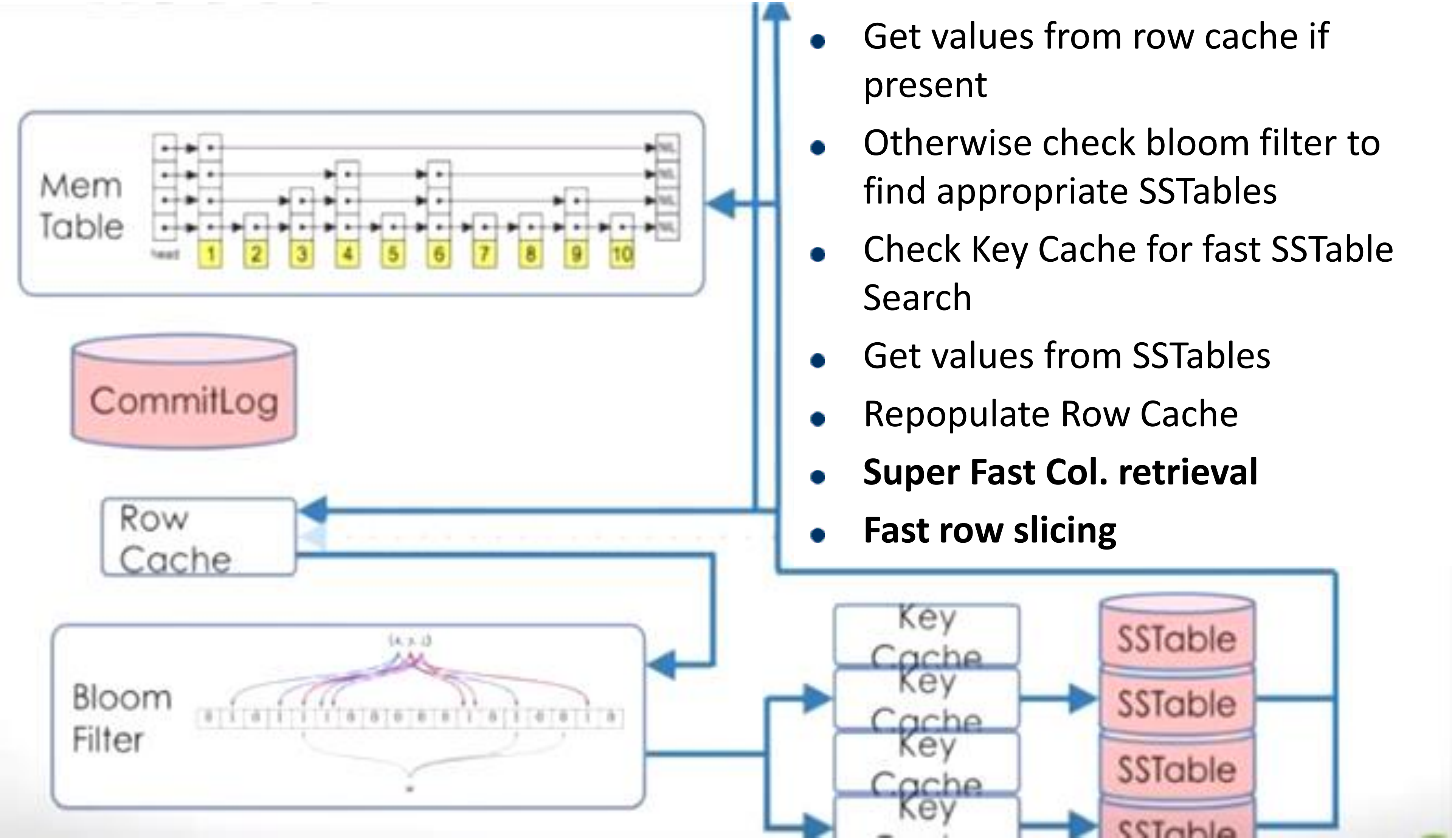
C* Data Model: Reads



- Get values from Memtable
- Get values from row cache if present
- Otherwise check bloom filter to find appropriate SSTables
- Check Key Cache for fast SSTable Search
- Get values from SSTables
- Repopulate Row Cache
- **Super Fast Col. retrieval**
- **Fast row slicing**



C* Data Model: Reads (Contd.)



- Get values from Memtable
- Get values from row cache if present
- Otherwise check bloom filter to find appropriate SSTables
- Check Key Cache for fast SSTable Search
- Get values from SSTables
- Repopulate Row Cache
- **Super Fast Col. retrieval**
- **Fast row slicing**

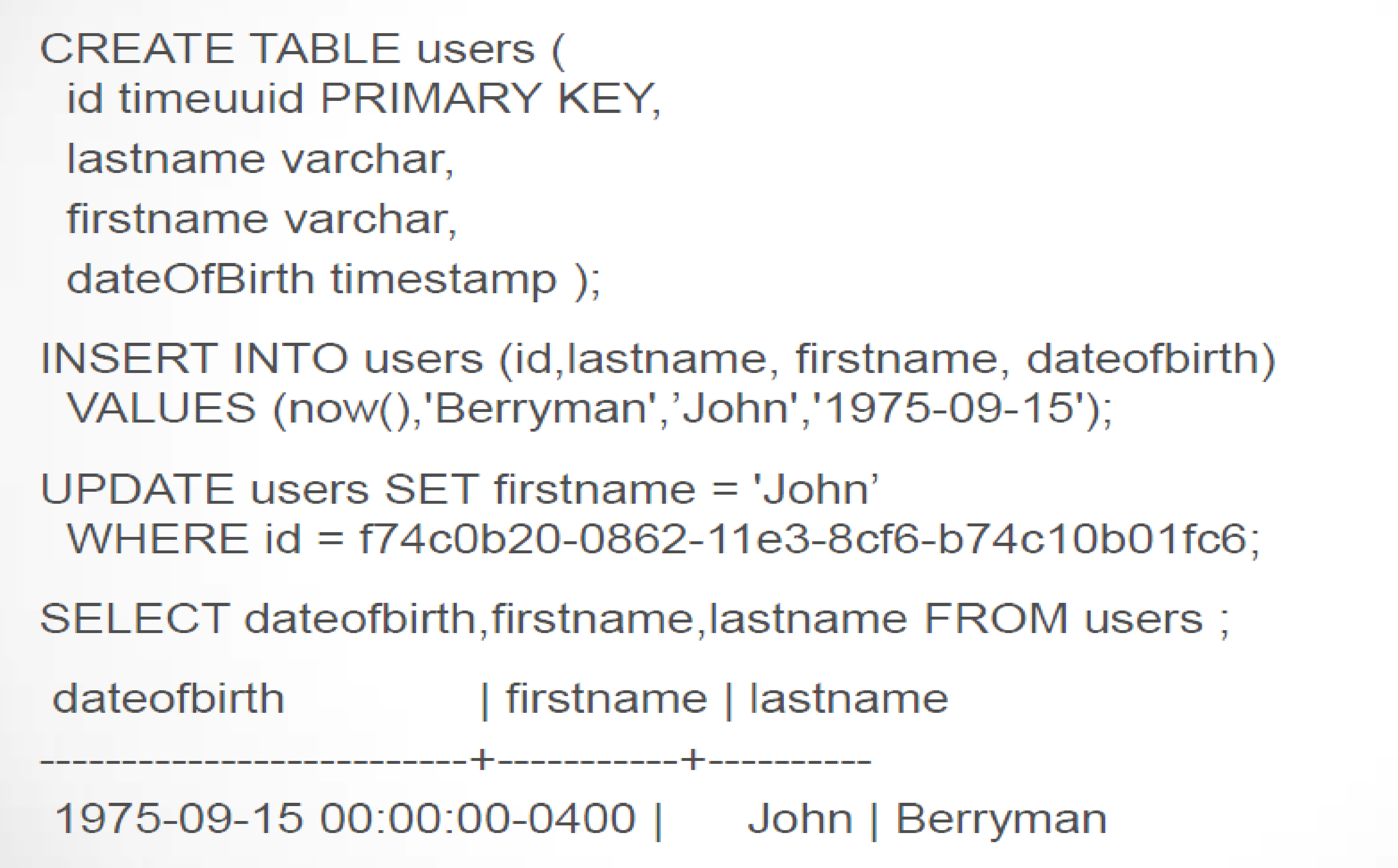
Introducing CQL

- **CQL is a reintroduction of schema** so that you don’t have to read code to understand the data model.

- **CQL creates a common language** so that details of the data model can be easily communicated.

- **CQL is a best-practices Cassandra interface** and hides the messy details.

Introducing CQL (Contd.)

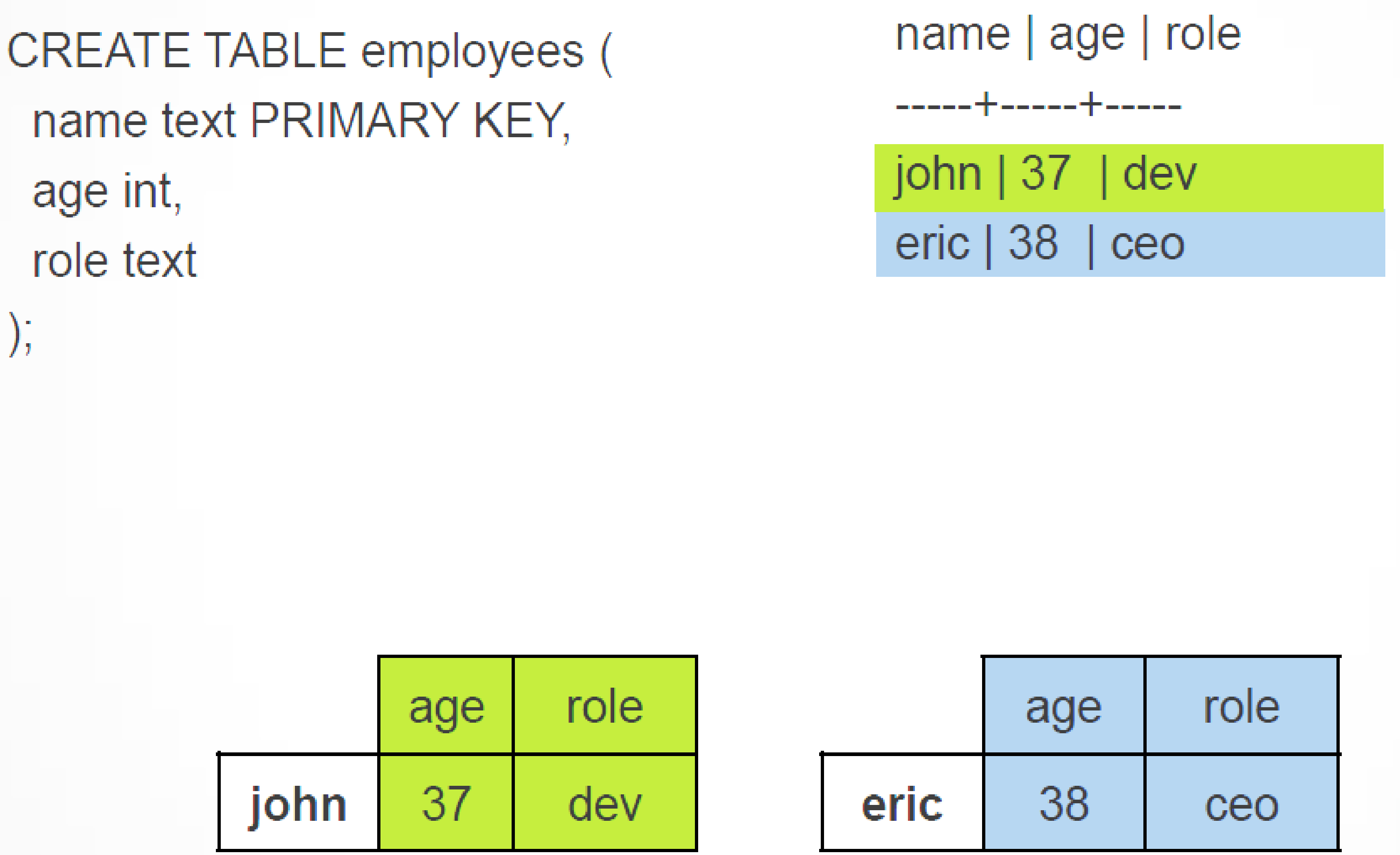


```
CREATE TABLE users (
  id timeuuid PRIMARY KEY,
  lastname varchar,
  firstname varchar,
  dateOfBirth timestamp );

INSERT INTO users (id,lastname, firstname, dateofbirth)
  VALUES (now(),'Berryman',’John','1975-09-15');

UPDATE users SET firstname = 'John’
  WHERE id = f74c0b20-0862-11e3-8cf6-b74c10b01fc6;

SELECT dateofbirth,firstname,lastname FROM users ;

 dateofbirth                | firstname | lastname
--------------------------+-----------+----------
 1975-09-15 00:00:00-0400 |      John | Berryman
```

Remember this:

- Cassandra finds rows fast
- Cassandra scans columns fast
- Cassandra *does not* scan rows

The CQL/Cassandra Mapping



```
CREATE TABLE employees (
  name text PRIMARY KEY,
  age int,
  role text
);
```

```
name | age | role
-----+-----+-----
john | 37  | dev
eric | 38  | ceo
```

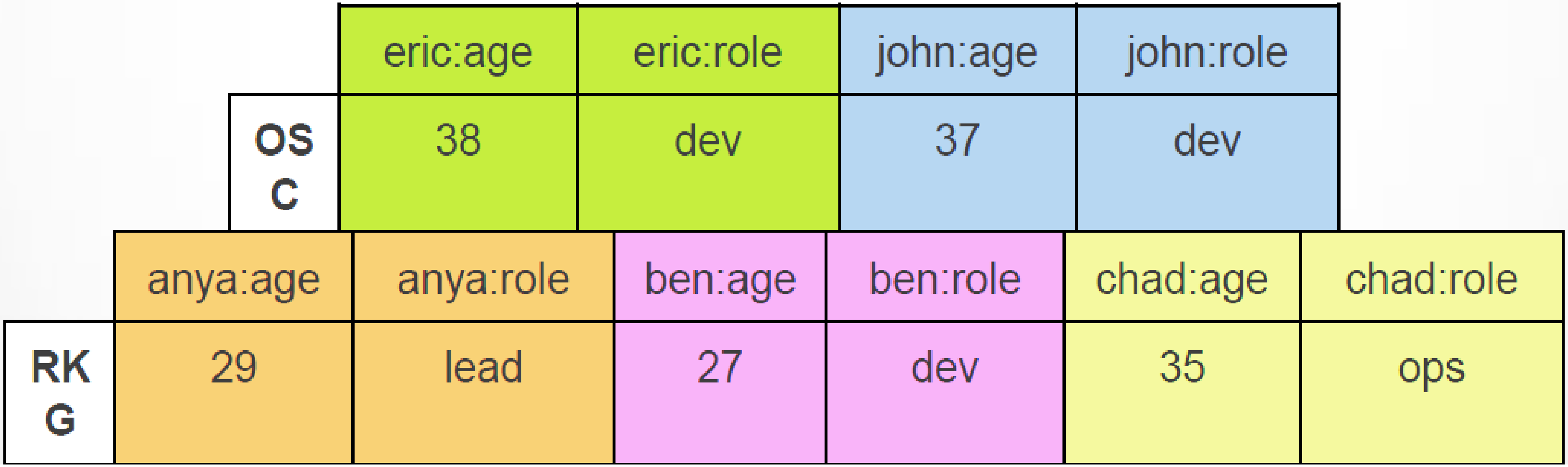
	age	role
john	37	dev

	age	role
eric	38	ceo

The CQL/Cassandra Mapping

```
CREATE TABLE employees (
  company text,
  name text,
  age int,
  role text,
  PRIMARY KEY (company,name)
);
```

```
company | name | age | role
--------+------+-----+-----
OSC     | eric | 38  | ceo
OSC     | john | 37  | dev
RKG     | anya | 29  | lead
RKG     | ben  | 27  | dev
RKG     | chad | 35  | ops
```



	eric:age	eric:role	john:age	john:role
OSC	38	dev	37	dev

	anya:age	anya:role	ben:age	ben:role	chad:age	chad:role
RKG	29	lead	27	dev	35	ops

The CQL/Cassandra Mapping (Contd.)

```
CREATE TABLE example (
  A text,
  B text,
  C text,
  D text,
  E text,
  F text,
  PRIMARY KEY ((A,B),C,D)
);
```

```
A | B | C | D | E | F
--+---+---+---+---+---
a | b | c | d | e | f
a | b | c | g | h | i
a | b | j | k | l | m
a | n | o | p | q | r
s | t | u | v | w | x
```

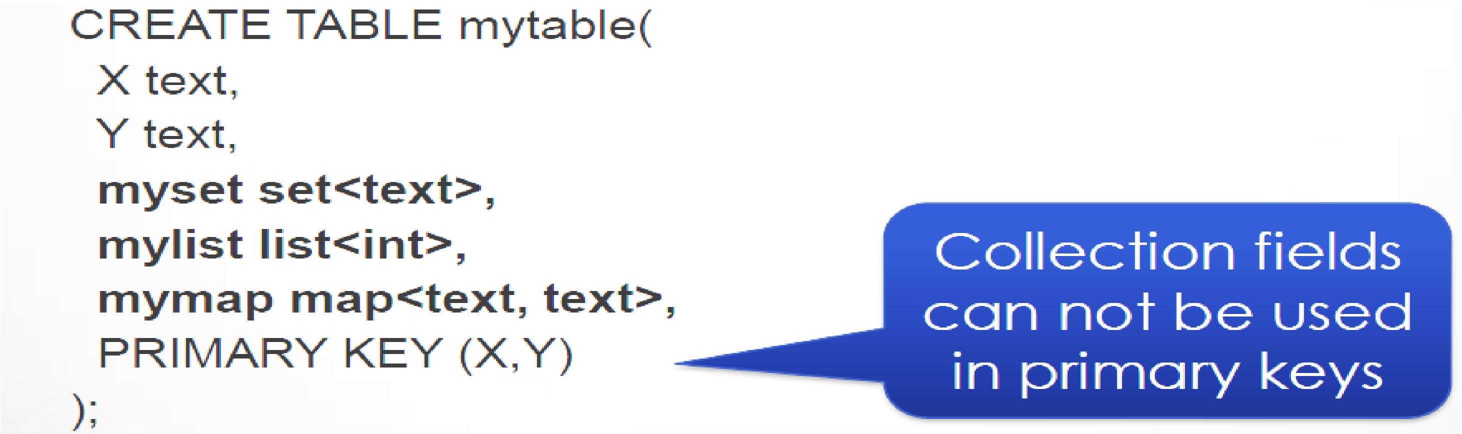
	c:d:E	c:d:F	c:g:E	c:g:F	j:k:E	j:k:F
a:b	e	f	h	i	l	m

	o:p:E	o:p:F
a:n	q	r

	u:v:E	u:v:F
s:t	w	x

CQL for Sets, Lists and Maps

- Collection Semantics
  - Sets hold list of unique elements
  - Lists hold ordered, possibly repeating elements
  - Maps hold a list of key-value pairs
- Uses same old Cassandra datastructure
- Declaring



```
CREATE TABLE mytable(
  X text,
  Y text,
  myset set<text>,
  mylist list<int>,
  mymap map<text, text>,
  PRIMARY KEY (X,Y)
);
```

Collection fields can not be used in primary keys

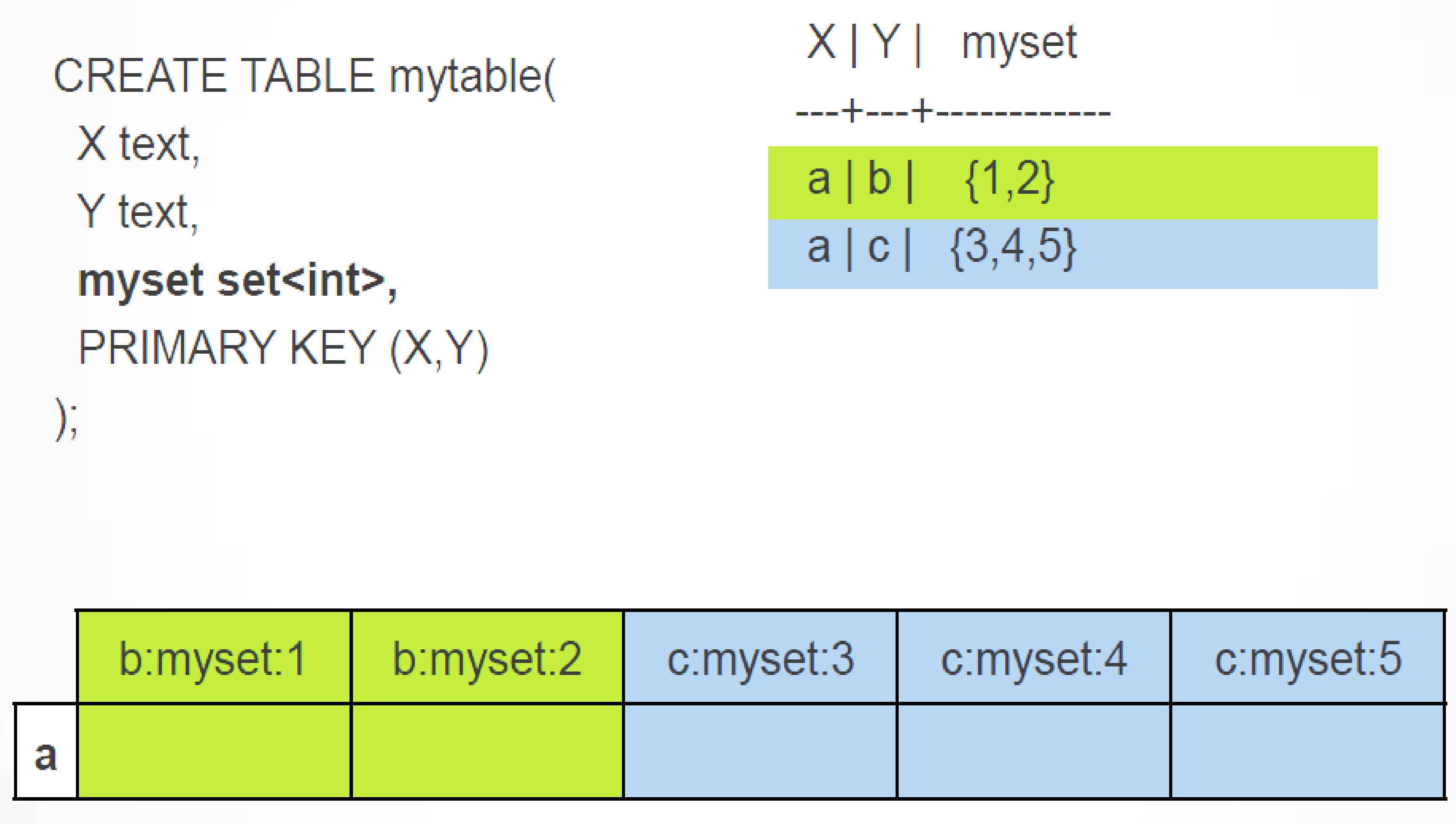
Inserting

```
INSERT INTO mytable (row, myset)
   VALUES (123, { ‘apple’, ‘banana’});

INSERT INTO mytable (row, mylist)
   VALUES (123, [‘apple’,’banana’,’apple’]);

INSERT INTO mytable (row, mymap)
    VALUES (123, {1:’apple’,2:’banana’})
```

Updating

```
UPDATE mytable SET myset = myset + {‘apple’,‘banana’}
WHERE row = 123;
UPDATE mytable SET myset = myset - { ‘apple’ }
WHERE row = 123;

UPDATE mytable SET mylist = mylist + [‘apple’,‘banana’]
WHERE row = 123;
UPDATE mytable SET mylist = [‘banana’] + mylist
WHERE row = 123;

UPDATE mytable SET mymap[‘fruit’] = ‘apple’
WHERE row = 123
UPDATE mytable SET mymap = mymap + { ‘fruit’:‘apple’}
WHERE row = 123
```

SETS



```
CREATE TABLE mytable(
  X text,
  Y text,
  myset set<int>,
  PRIMARY KEY (X,Y)
);
```

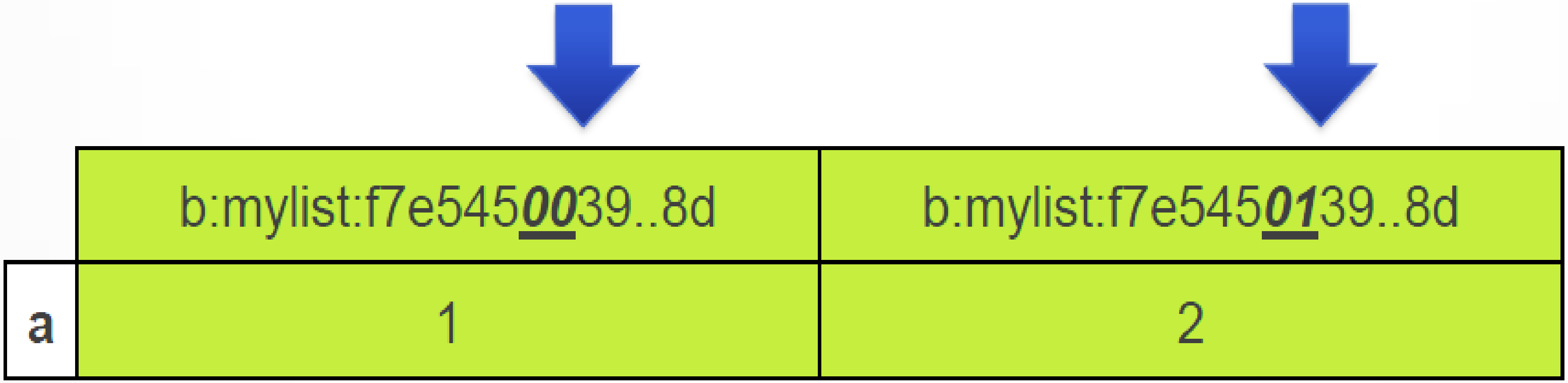
```
 X | Y |   myset
---+---+------------
 a | b |   {1,2}
 a | c |  {3,4,5}
```

	b:myset:1	b:myset:2	c:myset:3	c:myset:4	c:myset:5
a					

LISTS

```
CREATE TABLE mytable(
 X text,
 Y text,
 mylist list<int>,
 PRIMARY KEY (X,Y)
);
```

```
 X | Y |   mylist
---+---+------------
 a | b |  [1,2]
```



	b:mylist:f7e545***00***39..8d	b:mylist:f7e545***01***39..8d
a	1	2

MAPS

```
CREATE TABLE mytable(
  X text,
  Y text,
  mymap map<text,int>,
  PRIMARY KEY (X,Y)
);
```

```
 X | Y |   mymap
---+---+------------
 a | b |  {m:1,n:2}
 a | c |{n:3,p:4,q:5}
```

	b:mymap:m	b:mymap:n	c:mymap:n	c:mymap:p	c:mymap:q
a	1	2	3	4	5

Example

(in cqlsh)

```
CREATE KEYSPACE test WITH replication =
    {'class': 'SimpleStrategy', 'replication_factor': 1};
USE test;
CREATE TABLE stuff ( a int, b int, myset set<int>,
    mylist list<int>, mymap map<int,int>, PRIMARY KEY (a,b));
UPDATE stuff SET myset = {1,2}, mylist = [3,4,5], mymap = {6:7,8:9} WHERE a = 0
AND b = 1;
SELECT * FROM stuff;
```

(in cassandra-cli)

```
use test;
list stuff ;
```

(in cqlsh)

```
SELECT key_aliases,column_aliases from system.schema_columnfamilies WHERE
keyspace_name = 'test' AND columnfamily_name = 'stuff';
```

Conclusion

- CQL is a reintroduction of schema
- CQL creates a common data modeling language
- CQL is a best-practices Cassandra interface
- CQL let’s you take advantage of the C* Data structure
- CQL protocol is binary and therefore interoperable with any language
- CQL is asynchronous and fast (Thrift transport layer is synchronous)
- CQL allows the possibility for prepared statements