User-Defined Functions & Materialized Views

DuyHai DOAN

User Define Functions

- Why ?
- Detailed Impl
- UDAs
- Gotchas

Rationale

- Push computation server-side
  - save network bandwidth (1000 nodes!)
  - simplify client-side code
  - provide standard & useful function (sum, avg …)
  - accelerate analytics use-case (**pre-aggregation** for Spark)

DATASTAX

How to create an UDF ?

```
CREATE [OR REPLACE] FUNCTION [IF NOT EXISTS]
[keyspace.]functionName (param1 type1, param2 type2, ...)
CALL ON NULL INPUT | RETURNS NULL ON NULL INPUT
RETURN returnType
LANGUAGE language
AS '
      // source code here
';
```


DATASTAX

How to create an UDF ?

```
CREATE [OR REPLACE] FUNCTION [IF NOT EXISTS]
[keyspace.]functionName (param1 type1, param2 type2, ...)
CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT
RETURN returnType
LANGUAGE language
AS '
      // source code here
';
```

~~CREATE OR REPLACE FUNCTION IF NOT EXISTS~~

DATASTAX

How to create an UDF ?

```
CREATE [OR REPLACE] FUNCTION [IF NOT EXISTS]
[keyspace.]functionName (param1 type1, param2 type2, ...)
CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT
RETURN re      Type
LANGUA
AS '
      // source code here
';
```

An UDF is keyspace-wide

DATASTAX

How to create an UDF ?

```
CREATE [OR REPLACE] FUNCTION [IF NOT EXISTS]
[keyspace.]functionName (param1 type1, param2 type2, ...)
CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT
RETURN returnType
LANGUAGE language
AS '
      // source code here
';
```

Param name to refer to in the code
Type = CQL3 type

DATASTAX

How to create an UDF ?

```
CREATE [OR REPLACE] FUNCTION [IF NOT EXISTS]
[keyspace.]functionName (param1 type1, param2 type2, ...)
CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT
RETURN returnType
LANGUAGE la
AS '
      // source
';
```

Always called
Null-check mandatory in code

DATASTAX

How to create an UDF ?

```
CREATE [OR REPLACE] FUNCTION [IF NOT EXISTS]
[keyspace.]functionName (param1 type1, param2 type2, ...)
CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT
RETURN returnType
LANGUAGE la
AS '
      // source
';
```

If any input is null, code block is skipped and return null

DATASTAX

How to create an UDF ?

```
CREATE [OR REPLACE] FUNCTION [IF NOT EXISTS]
[keyspace.]functionName (param1 type1, param2 type2, ...)
CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT
RETURN returnType
LANGUAGE languag
AS '
      // source coc
';
```

CQL types

- primitives (boolean, int, ...)
- collections (list, set, map)
- tuples
- UDT

DATASTAX

How to create an UDF ?

```
CREATE [OR REPLACE] FUNCTION [IF NOT EXISTS]
[keyspace.]functionName (param1 type1, param2 type2, ...)
CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT
RETURN returnType
LANGUAGE language
AS '
      // source code here
';
```

JVM supported languages

- Java, Scala
- Javascript (slow)
- Groovy, Jython, JRuby
- ~~Clojure~~ (JSR 223 impl issue)

DATASTAX

How to create an UDF ?

```
CREATE [OR REPLACE] FUNCTION [IF NOT EXISTS]
[keyspace.]functionName (param1 type1, param2 type2, ...)
CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT
RETURN returnType
LANGUAGE language
AS '
    // source code here
';
```

Avoid simple quotes (') or escape them

DATASTAX

Simple UDF Demo
max()

UDF creation for C* 2.2 and 3.0

- UDF definition sent to **coordinator**
- Check for user permissions (**ALTER**, **DROP**, **EXECUTE**)
- Coordinator compiles it using **ECJ compiler** (not Javac because of **licence issue**).
- If language != Java, use other compilers (you should ship the jars on C* servers)
- Announces SchemaChange = **CREATED**, Target = **FUNCTION** to **all nodes**
- Code loaded into **child classloader** of current classloader

UDF creation in C* 3.0

- Compiled byte code is verified
  - static and instance code block
  - declaring methods/constructor → no anonymous class creation
  - declaring inner class
  - use of synchronized
  - accessing forbidden classes, ex: java.net.NetworkInterface
  - accessing forbidden methods, ex: Class.forName()
  - …
- Code loaded into seperated classloader
  - verify loaded classes against whitelist/backlist

UDF execution in C* 2.2

- No safeguard, arbitrary code execution server side …
- Flag "**enable_user_defined_functions**" in **cassandra.yaml** turned **OFF** by default
- Keep your UDFs **pure** (**stateless**)
- Avoid expensive computation (for loop …)

DATASTAX

UDF execution in C* 3.0

- If "**enable_user_defined_functions_threads**" turned **ON** (default)
  - execute UDF in separated **ThreadPool**
  - if duration longer than "**user_defined_function_warn_timeout**", retry with new timeout = **fail_timeout – warn_timeout**
  - if duration longer than "**user_defined_function_fail_timeout**", exception
- Else
  - execute in same thread

UDF creation best practices

- Keep your UDFs **pure** (**stateless**)
- No **expensive computation** (huge for loop)
- Prefer **Java** language for perf

DATASTAX

UDA

- Real use-case for UDF
- Aggregation server-side → huge network bandwidth saving
- Provide similar behavior for Group By, Sum, Avg etc …

DATASTAX

How to create an UDA ?

```
CREATE [OR REPLACE] AGGREGATE [IF NOT EXISTS]
[keyspace.]aggregateName(type1, type2, ...)
SFUNC accumulatorFunction
STYPE stateType
[FINALFUNC finalFunction]
INITCOND initCond;
```

Only type, no param name

State type

Initial state type

DATASTAX

How to create an UDA ?

```
CREATE [OR REPLACE] AGGREGATE [IF NOT EXISTS]
[keyspace.]aggregateName(type1, type2, ...)
SFUNC accumulatorFunction
STYPE stateType
[FINALFUNC finalFunction]
INITCOND initCond;
```

Accumulator function. Signature:
accumulatorFunction(stateType, $type_1$, $type_2$, ...)
RETURNS stateType

DATASTAX

How to create an UDA ?

```
CREATE [OR REPLACE] AGGREGATE [IF NOT EXISTS]
[keyspace.]aggregateName(type1, type2, ...)
SFUNC accumulatorFunction
STYPE stateType
[FINALFUNC finalFunction]
INITCOND initCond;
```

Optional final function. Signature: finalFunction(stateType)

DATASTAX

How to create an UDA ?

```
CREATE [OR REPLACE] AGGREGATE [IF NOT EXISTS]
[keyspace.]aggregateName(type1, type2, ...)
SFUNC accumulatorFunction
STYPE stateType
[FINALFUNC finalFunction]
INITCOND initCond;
```

UDA return type ?

If finalFunction

- return type of finalFunction

Else

- return stateType

DATASTAX

How to create an UDA ?

```
CREATE [OR REPLACE] AGGREGATE [IF NOT EXISTS]
[keyspace.]aggregateName(type1, type2, ...)
SFUNC accumulatorFunction
STYPE stateType
[FINALFUNC finalFunction]
INITCOND initCond;
```

If accumulatorFunction RETURNS NULL ON NULL INPUT
- initCond should be defined

Else
- initCond can be null (default value)

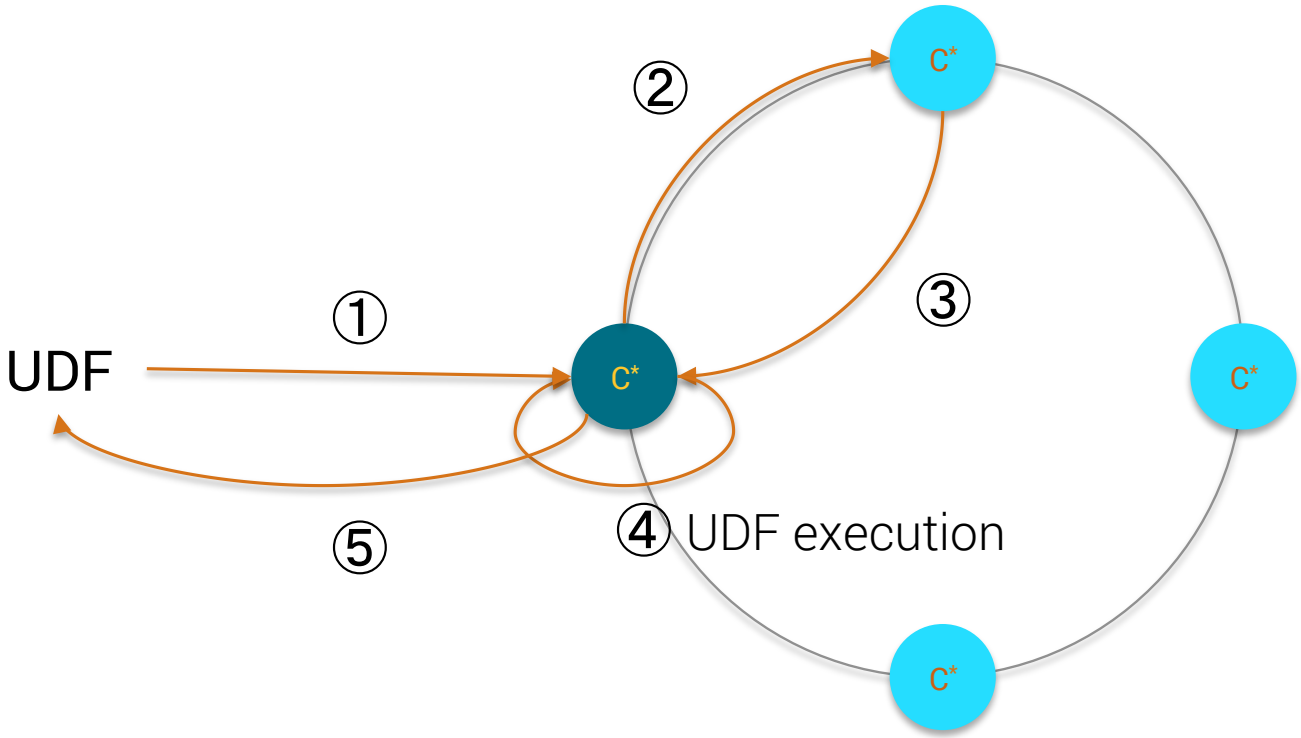
DATASTAX

UDA execution

- Start with **state** = **initCond**
- For each row
  - call accumulator function with **previous state** + **params**
  - update **state** with accumulator function output
- If no **finalFunction**
  - return **last state**
- Else
  - apply **finalFunction** and returns its output

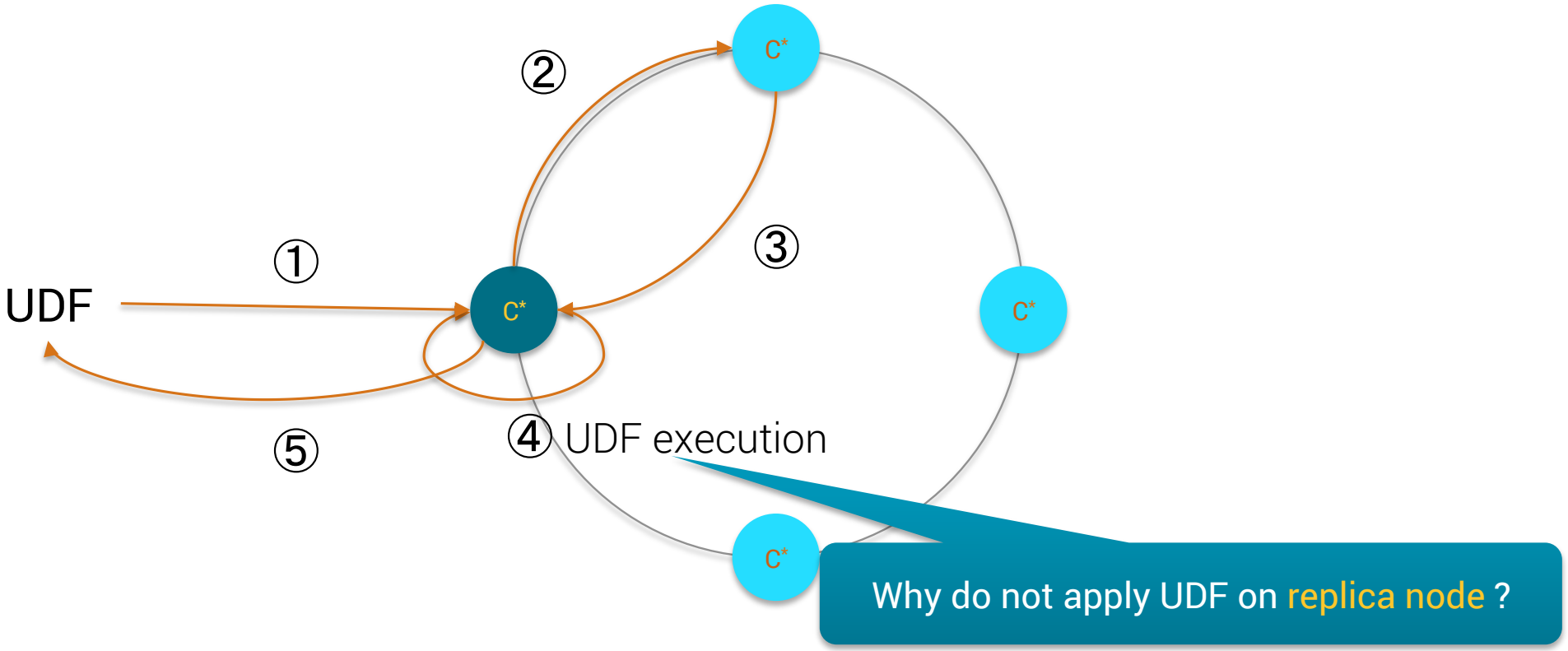
UDA Demo Group By

Gotchas



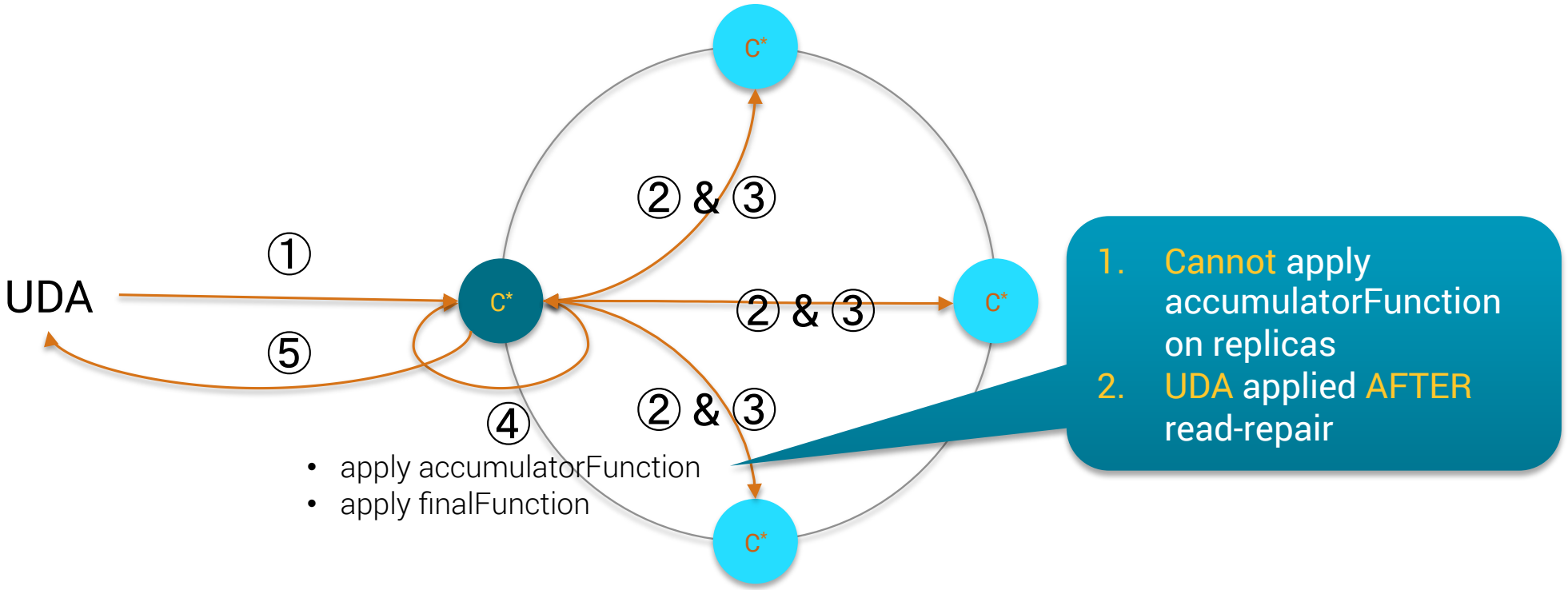
DATASTAX

Gotchas



DATASTAX

Gotchas

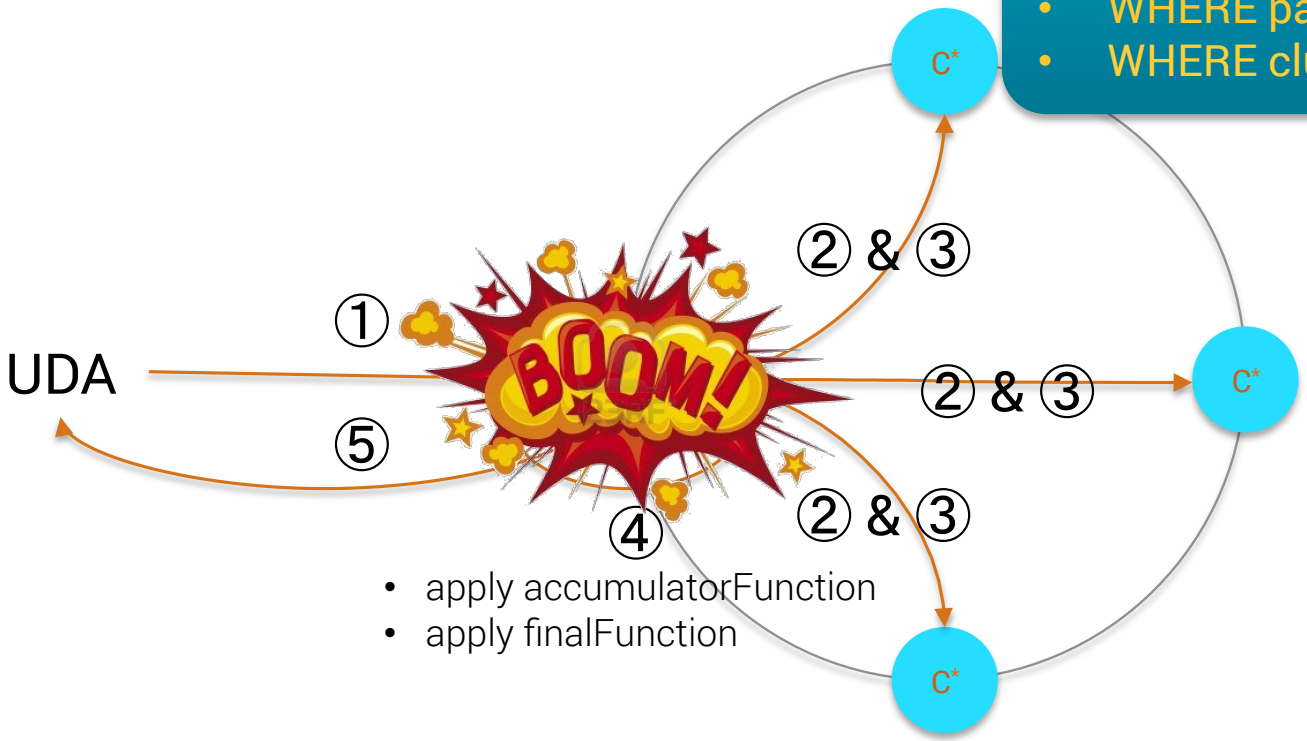


DATASTAX

Gotchas

Avoid UDA on large number of rows

- WHERE partitionKey IN (…)
- WHERE clustering >= AND clusterting <=

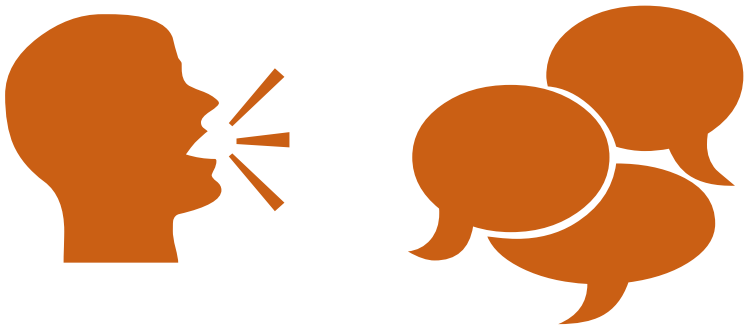


DATASTAX

Future applications of UDFs & UDAs

- Functional index (CASSANDRA-7458)
- Partial index (CASSANDRA-7391)
- WHERE clause filtering with UDF
- …

DATASTAX



Q & A

Materialized Views

- Why ?
- Detailed Impl
- Gotchas

Why Materialized Views ?

- Relieve the pain of manual denormalization

```
CREATE TABLE user(
   id int PRIMARY KEY,
   country text,
   ...
);
CREATE TABLE user_by_country(
   country text,
   id int,
   ...,
   PRIMARY KEY(country, id)
);
```

Materialzed View In Action

```
CREATE MATERIALIZED VIEW user_by_country
AS SELECT country, id, firstname, lastname
FROM  user
WHERE country IS NOT NULL AND id IS NOT NULL
PRIMARY KEY(country, id)
```

```
CREATE TABLE user_by_country (
    country text,
    id int,
    firstname text,
    lastname text,
    PRIMARY KEY(country, id));
```

Materialzed View Syntax

```
CREATE MATERIALIZED VIEW [IF NOT EXISTS]

keyspace_name.view_name

AS SELECT column1, column2, ...

FROM  keyspace_name.table_name

WHERE column1 IS NOT NULL AND column2 IS NOT NULL ...

PRIMARY KEY(column1, column2, ...)
```

Must select all primary key columns of base table

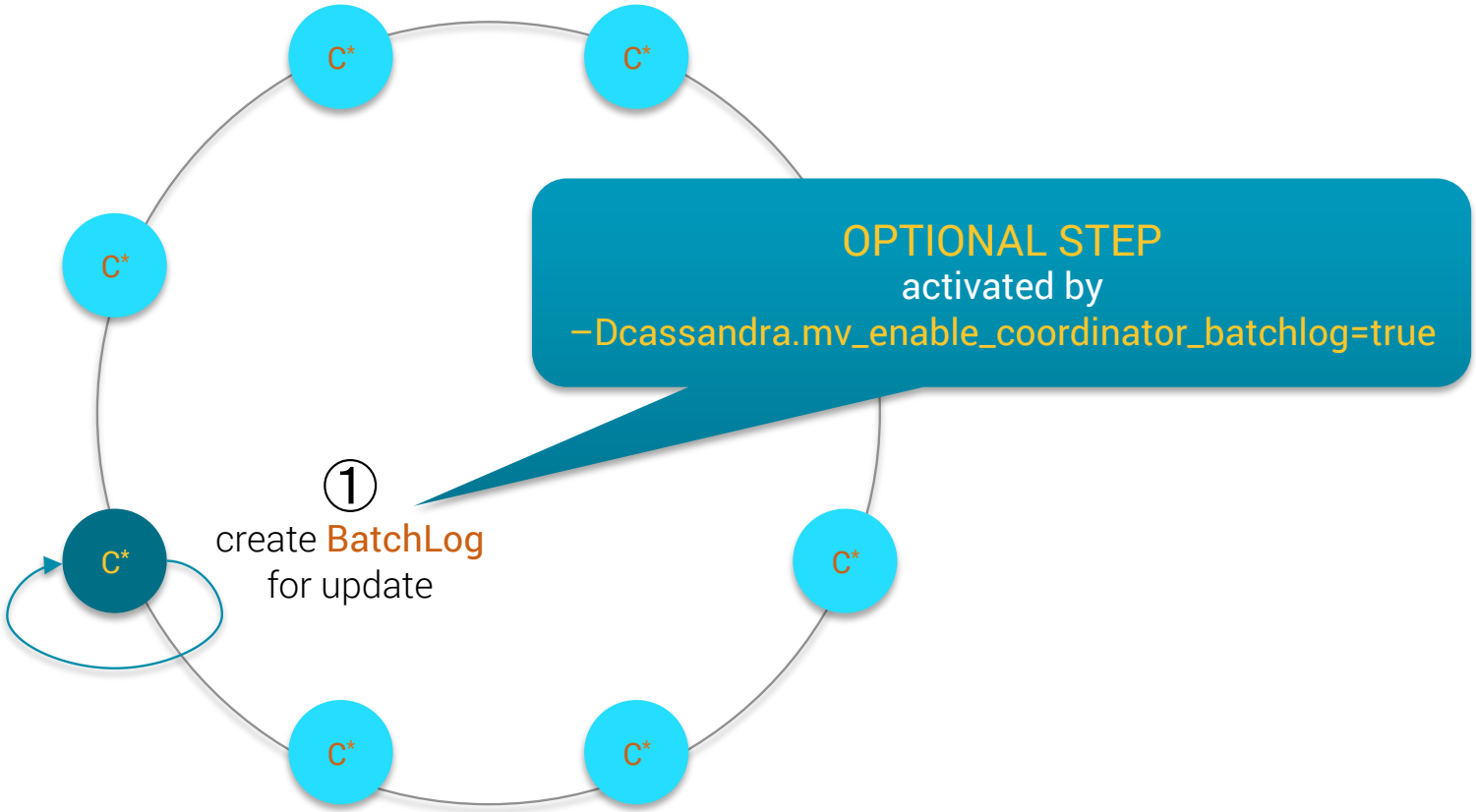
- IS NOT NULL condition for now
- more complex conditions in future

- at least all primary key columns of base table (ordering can be different)
- maximum 1 column NOT pk from base table
- 1:1 relationship between base row & MV row

Materialzed View Demo user_by_country

Materialized View Impl

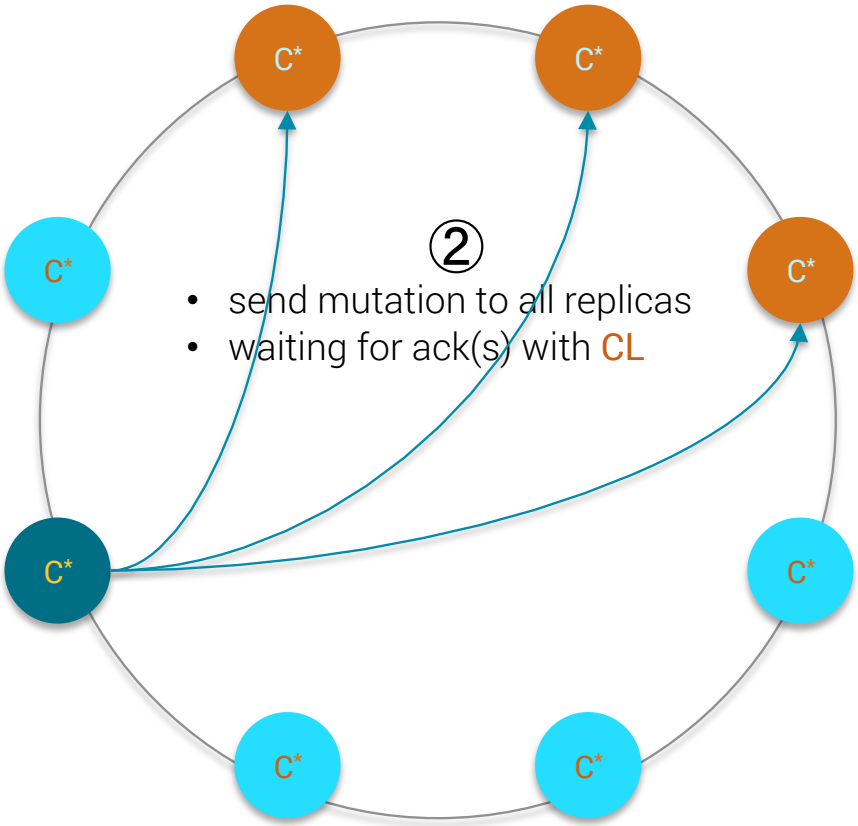
UPDATE user
SET country='FR'
WHERE id=1



DATASTAX

Materialized View Impl

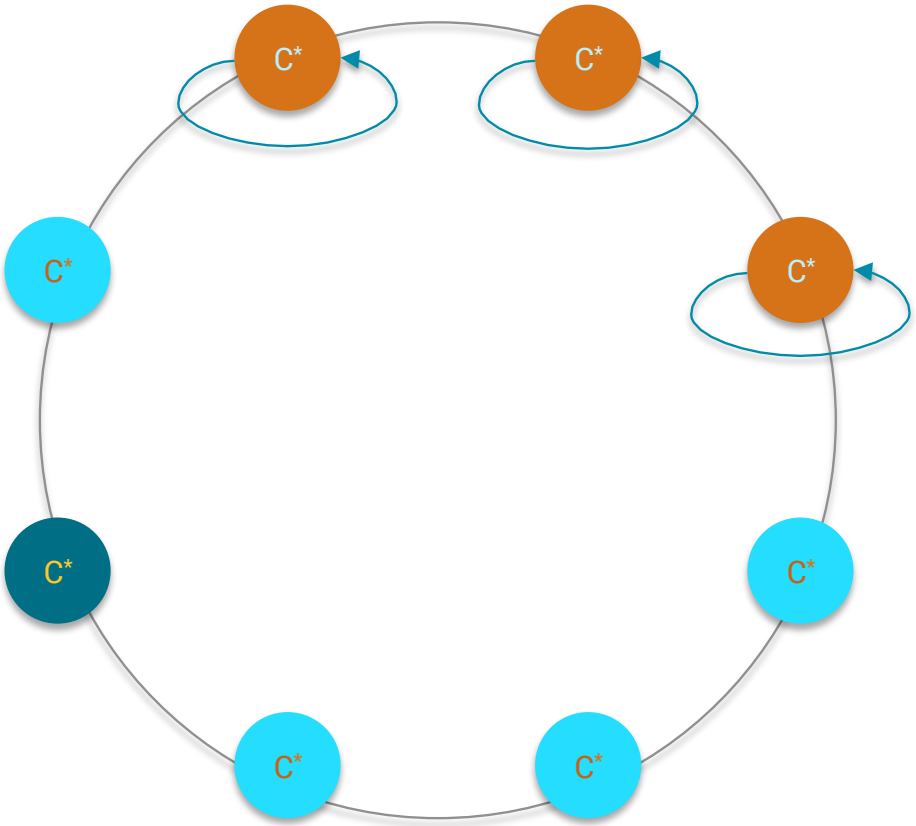
UPDATE user
SET country='FR'
WHERE id=1



DATASTAX

Materialized View Impl

UPDATE user
SET country='FR'
WHERE id=1

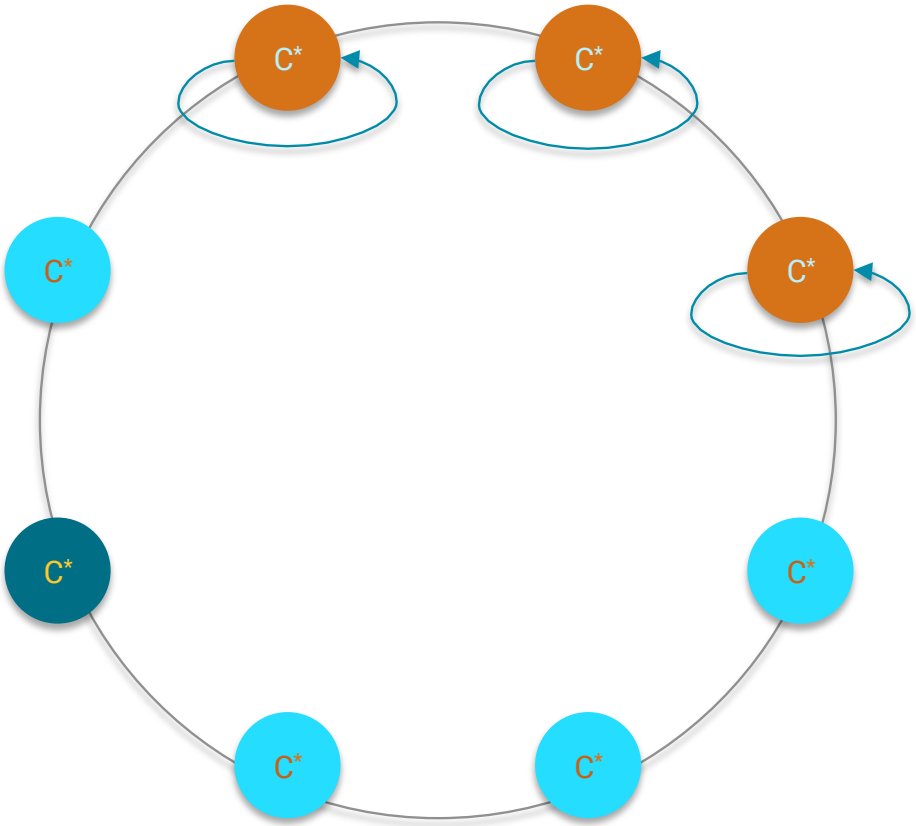


③
Acquire local lock on base table partition

DATASTAX

Materialized View Impl

UPDATE user
SET country='FR'
WHERE id=1

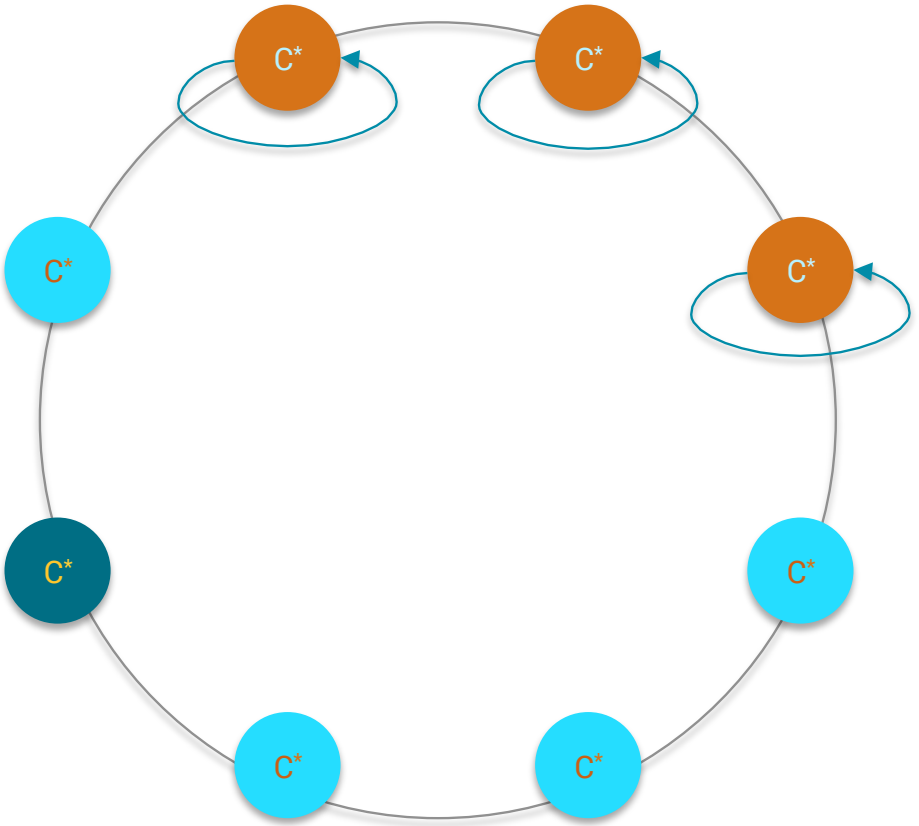


④
Local read to fetch current values
SELECT * FROM user WHERE id=1

DATASTAX

Materialized View Impl

UPDATE user
SET country='FR'
WHERE id=1



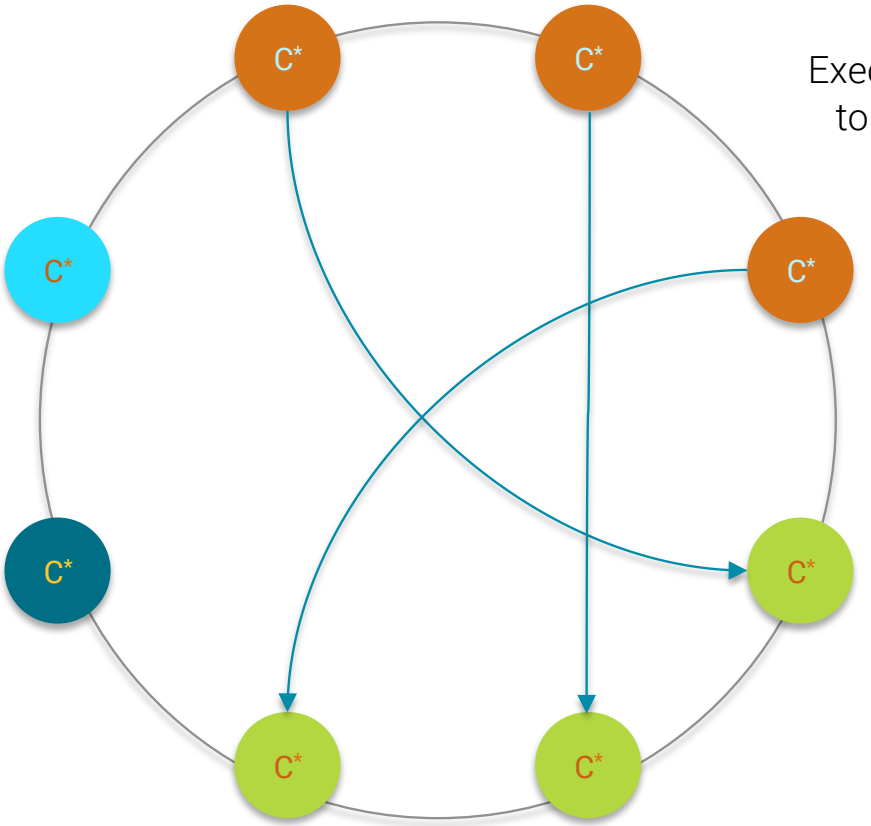
⑤
Create BatchLog with

- DELETE FROM user_by_country WHERE country = 'old_value'
- INSERT INTO user_by_country(country, id, …) VALUES('FR', 1, ...)

DATASTAX

Materialized View Impl

UPDATE user
SET country='FR'
WHERE id=1

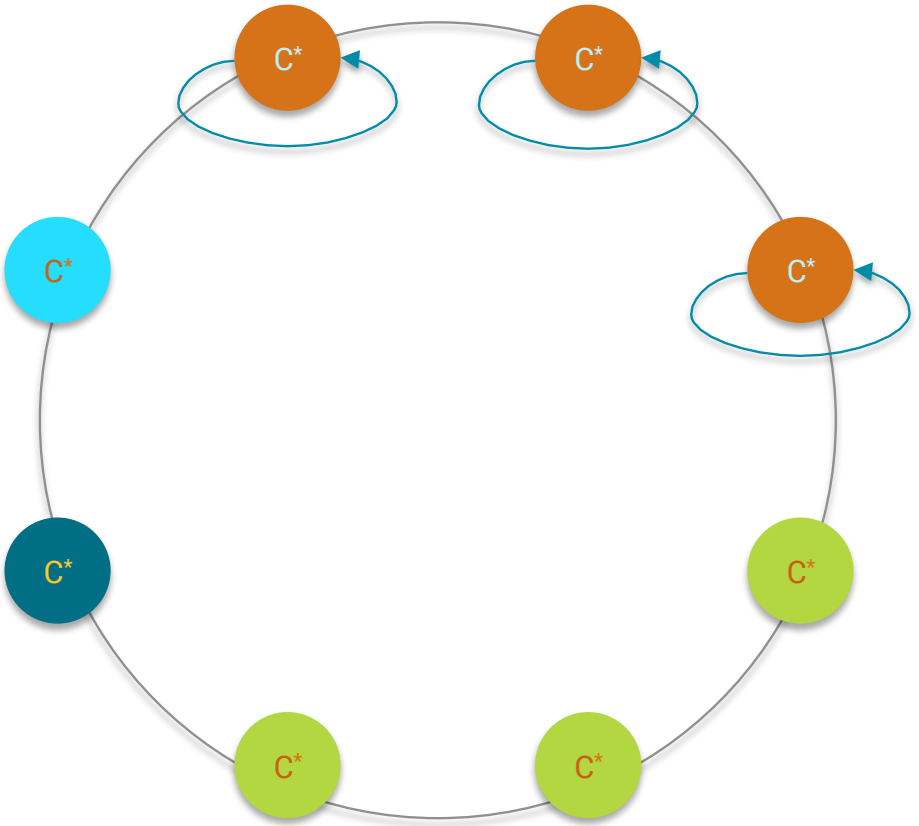


⑥
Execute async BatchLog
to paired view replica
with CL = ONE

DATASTAX

Materialized View Impl

UPDATE user
SET country='FR'
WHERE id=1

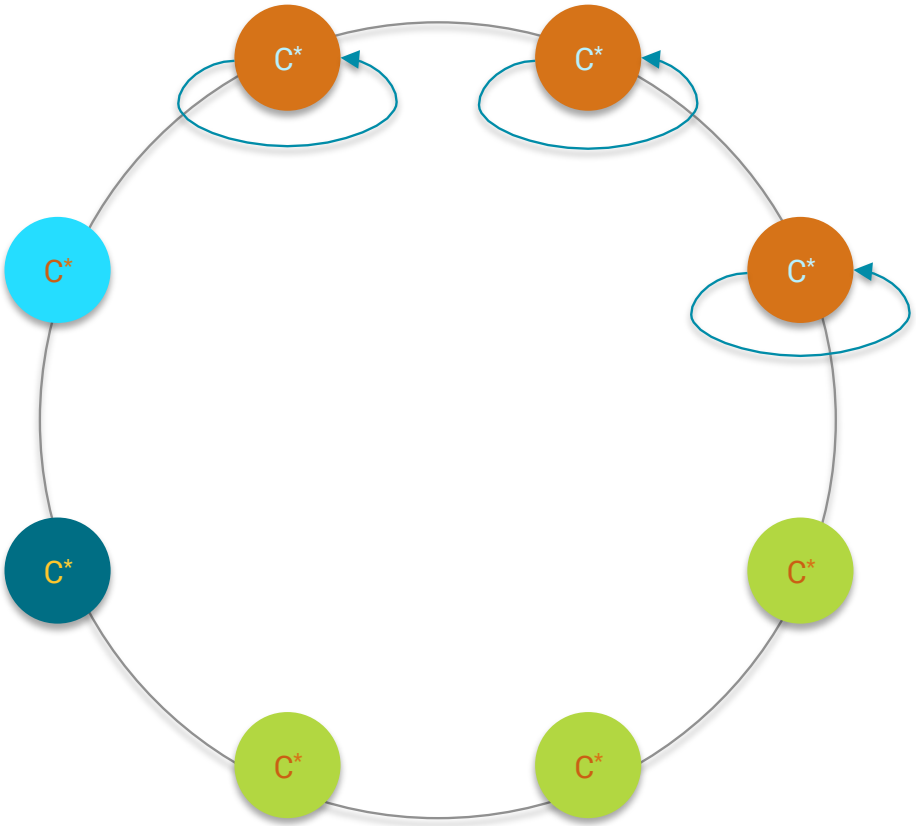


⑦
Apply base table updade locally
SET COUNTRY='FR'

DATASTAX

Materialized View Impl

UPDATE user
SET country='FR'
WHERE id=1

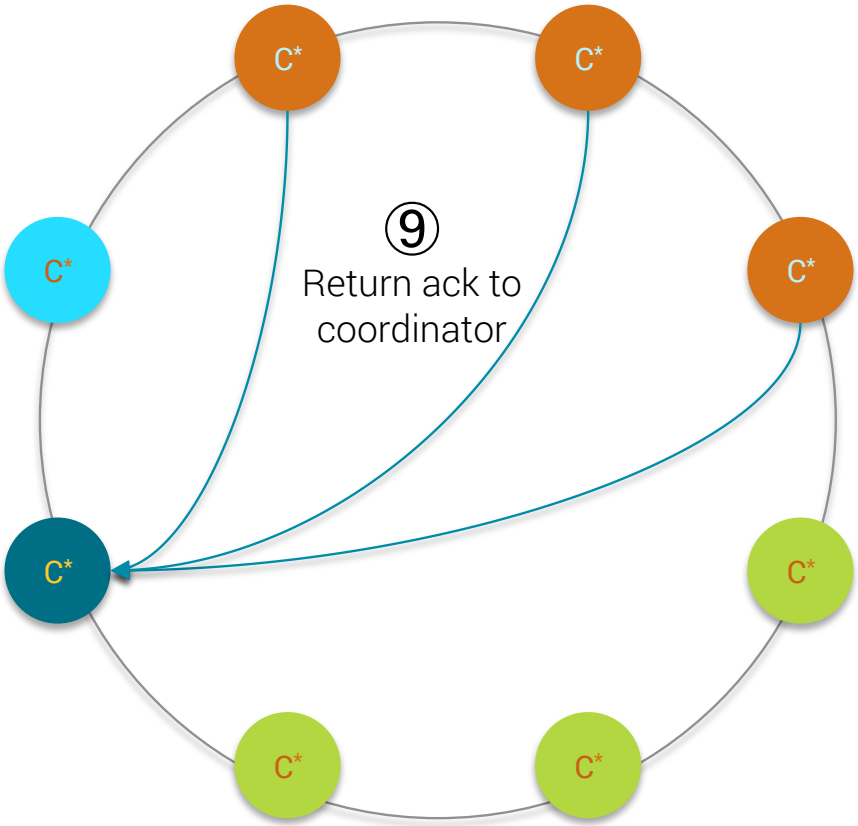


⑧
Release local lock

DATASTAX

Materialized View Impl

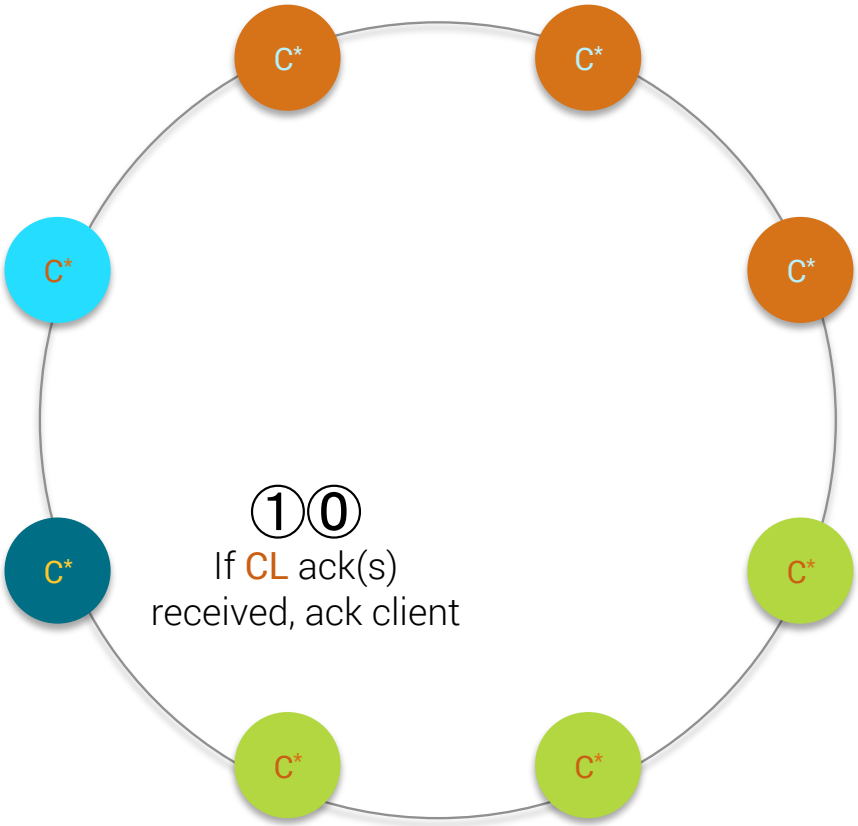
UPDATE user
SET country='FR'
WHERE id=1



DATASTAX

Materialized View Impl

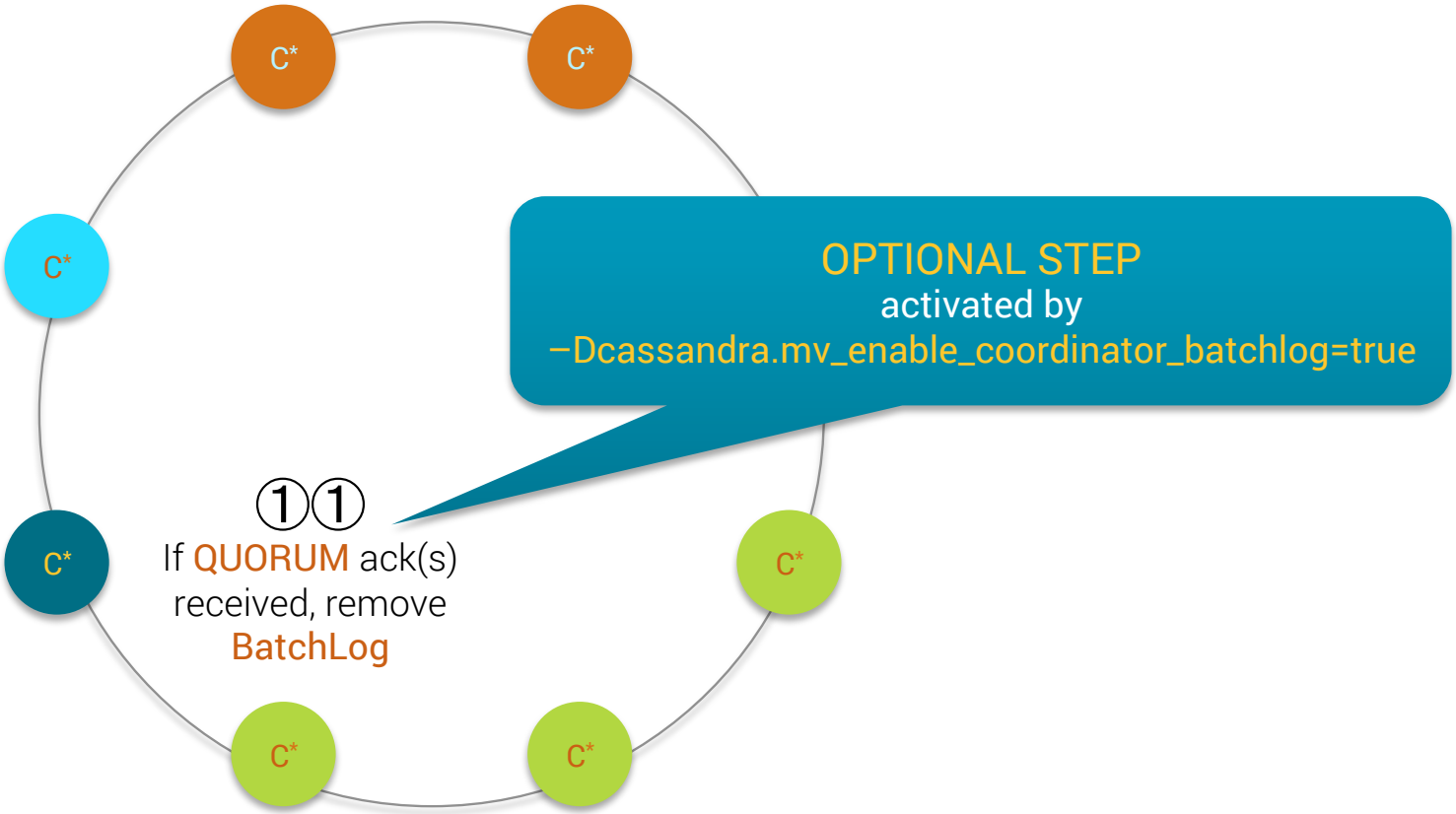
UPDATE user
SET country='FR'
WHERE id=1



DATASTAX

Materialized View Impl

UPDATE user
SET country='FR'
WHERE id=1

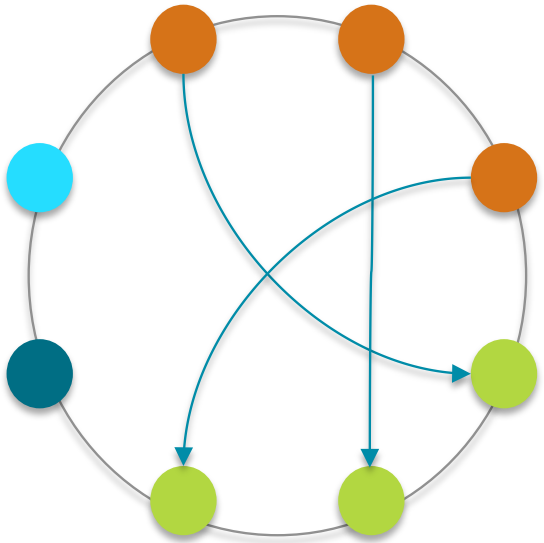


DATASTAX

Materialzed Views impl explained

What is paired replica ?

- Base primary replica for id=1 is paired with MV primary replica for country='FR'
- Base secondary replica for id=1 is paired with MV secondary replica for country='FR'
- etc …



DATASTAX

Materialzed Views impl explained

Why **local lock** on base replica ?

- to provide **serializability** on concurrent updates

Why **BatchLog** on base replica ?

- necessary for **eventual durability**
- replay the MV delete + insert until successful

Why **BatchLog** on base replica uses **CL = ONE** ?

- each base replica is responsible for update of its **paired** MV replica
- CL > ONE will create un-necessary **duplicated mutations**

DATASTAX

Materialzed Views impl explained

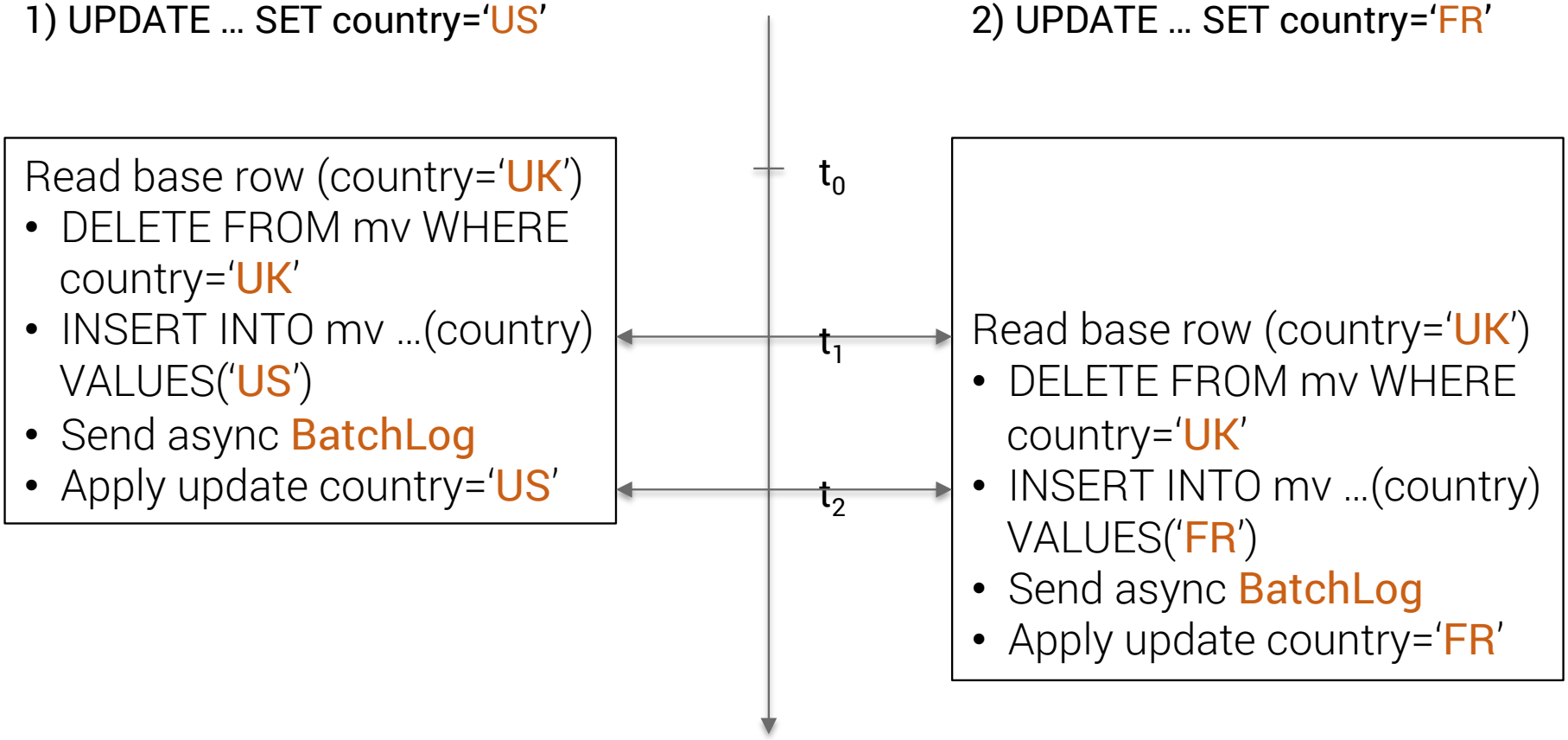
Why **BatchLog** on coordinator with **QUORUM** ?

- necessary to cover some **edge-cases**

DATASTAX

MV Failure Cases: concurrent updates

Without Local Lock



DATASTAX

MV Failure Cases: concurrent updates

With Local Lock

1) UPDATE … SET country='US'

Read base row (country='UK')
- DELETE FROM mv WHERE country='UK'
- INSERT INTO mv …(country) VALUES('US')
- Send async BatchLog
- Apply update country='US'

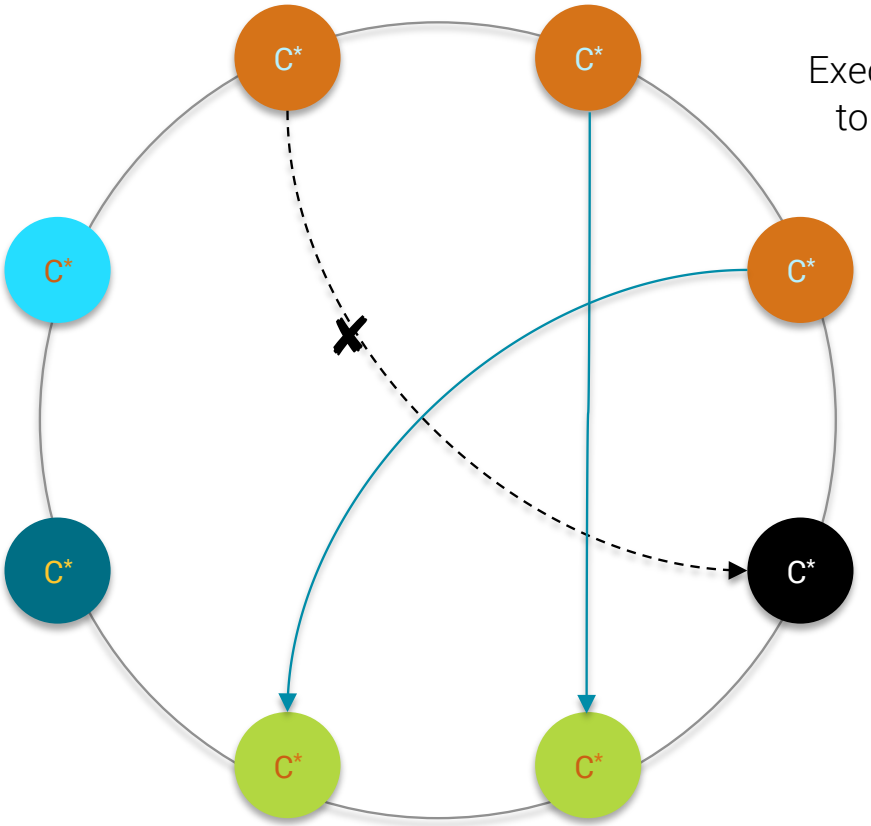
2) UPDATE … SET country='FR'

Read base row (country='US')
- DELETE FROM mv WHERE country='US'
- INSERT INTO mv …(country) VALUES('FR')
- Send async BatchLog
- Apply update country='FR'

DATASTAX

MV Failure Cases: failed updates to MV

UPDATE user
SET country='FR'
WHERE id=1



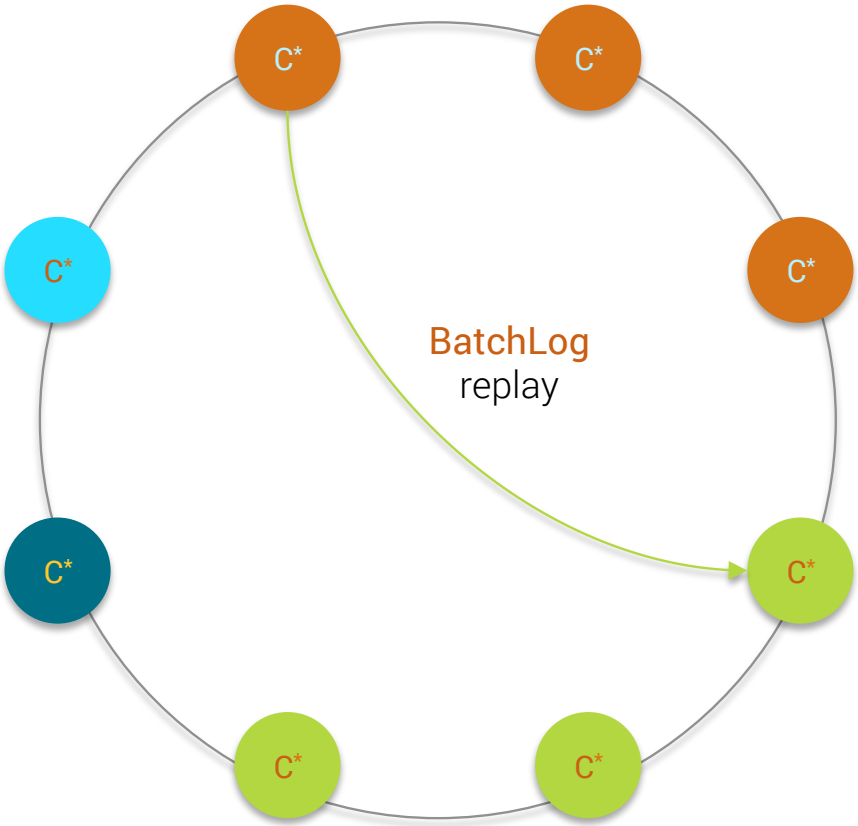
⑥
Execute async BatchLog
to paired view replica
with CL = ONE

MV replica down

DATASTAX

MV Failure Cases: failed updates to MV

UPDATE user
SET country='FR'
WHERE id=1



MV replica up

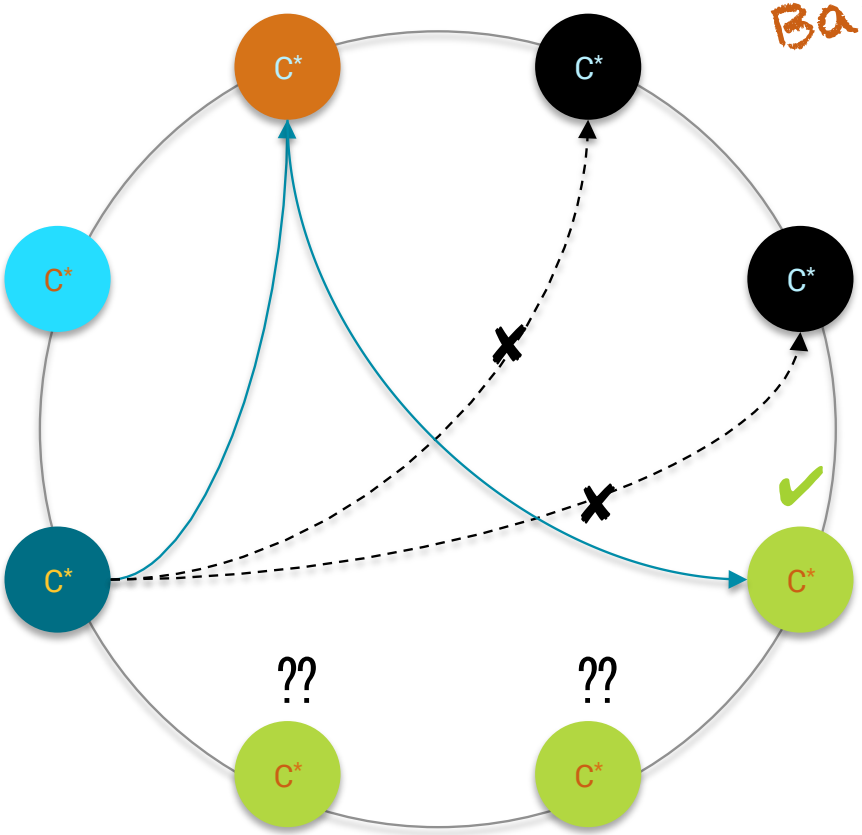
DATASTAX

MV Failure Cases: base replica(s) down

②

- send mutation to all replicas
- waiting for ack(s) with CL

UPDATE user
SET country='FR'
WHERE id=1

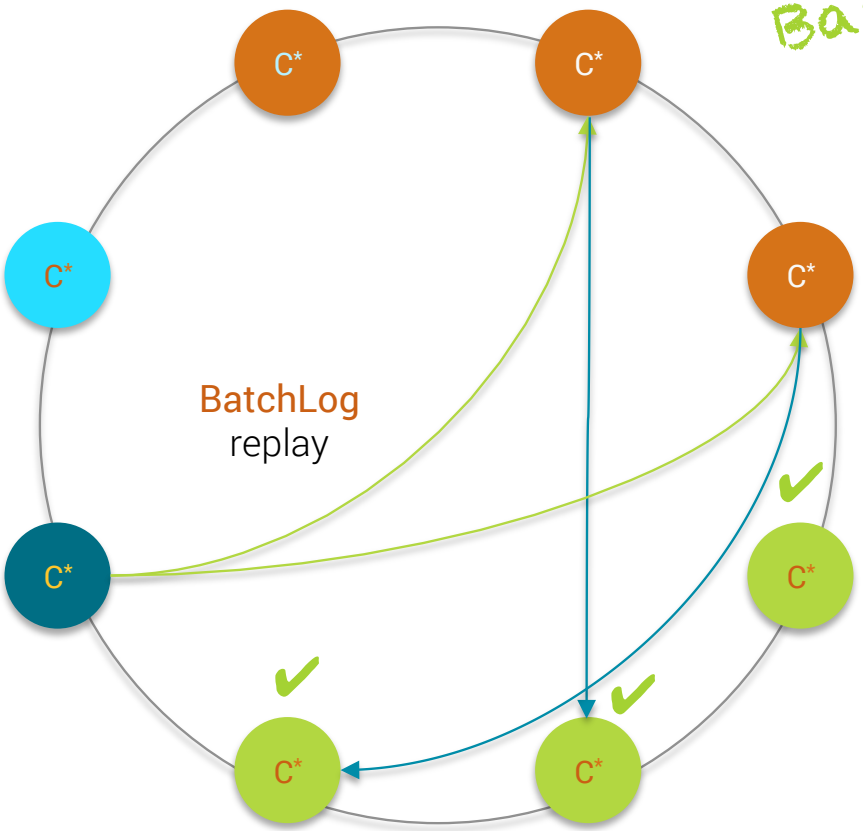


DATASTAX

MV Failure Cases: base replica(s) down

Base replicas up

UPDATE user
SET country='FR'
WHERE id=1



DATASTAX

Materialized Views Gotchas

- Write performance
  - local lock
  - local read-before-write for MV (*most of perf hits here*)
  - local batchlog for MV
  - coordinator batchlog (OPTIONAL)
  - ☞ you only pay this price **once** whatever number of MV
- Consistency level
  - CL honoured for base table, ONE for MV + local batchlog
  - QUORUM for coordinator batchlog (OPTIONAL)

DATASTAX

Materialized Views Gotchas

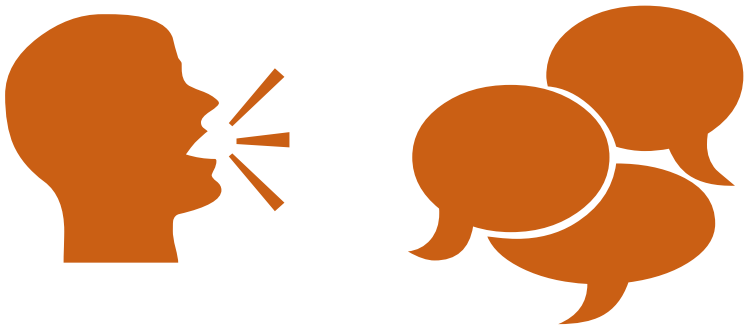
- Beware of hot spots !!!
  - MV **user_by_gender** 
- Repair, read-repair, index rebuild, decomission …
  - repair on base replica → update on MV paired replica
  - repair on MV replica possible
  - read-repair on MV behaves as normal read-repair

DATASTAX

Materialized Views Gotchas

- Schema migration
  - cannot drop column from base table used by MV
  - can add column to base table, initial value = null from MV
  - cannot drop base table, drop all MVs first

DATASTAX



Q & A

Thank You

@doanduyhai

duy_hai.doan@datastax.com

https://academy.datastax.com/