

# Initiation à Apache Cassandra

...

Devoxx France 2018

# Qui sommes nous ?

Alexander Dejanovski

@alexanderDeja

Consultant Cassandra

@ The Last Pickle

Maxence Lecointe

@maxospiquante

Architecte Java - Manager technique

@ Ippon



# Agenda

- Partitionnement et Réplication
- Syntaxe CQL (Cassandra Query Language)
- Tombstones
- LightWeight Transactions (LWT)
- Opérations
- Niveaux de cohérence
- Découverte et utilisation du driver Java et requêtes asynchrones
- Compaction
- Simulation de pannes et reprise sur incident

# Tools

**ccm** : Création de clusters multi noeuds/multi DC en local

```
ccm create my_cluster_3016 -v binary:3.0.16 -n 3:3
```

```
ccm node1 cqlsh
```

```
ccm node1 nodetool status
```

```
ccm start/stop
```

```
ccm switch another_cluster
```

# Tools

`pip install ccm`

Or

`brew install ccm`

## Basics - CQL = Cassandra Query Language

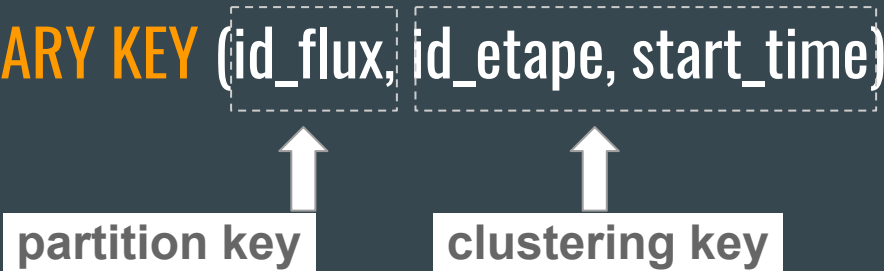
```
CREATE KEYSPACE myks WITH REPLICATION =  
    {'class': 'SimpleStrategy', 'replication_factor': 3}
```

## Basics - CQL = Cassandra Query Language

```
CREATE KEYSPACE myks WITH REPLICATION =  
    {'class': 'NetworkTopologyStrategy', 'dc1': 3, 'dc2': 3}
```

## Basics - CQL : DDL

```
CREATE TABLE IF NOT EXISTS myks.workflow(  
    id_flux varchar,  
    id_etape varchar,  
    start_time timestamp,  
    end_time timestamp,  
    status int,  
    PRIMARY KEY (id_flux, id_etape, start_time)  
);
```



partition key

clustering key

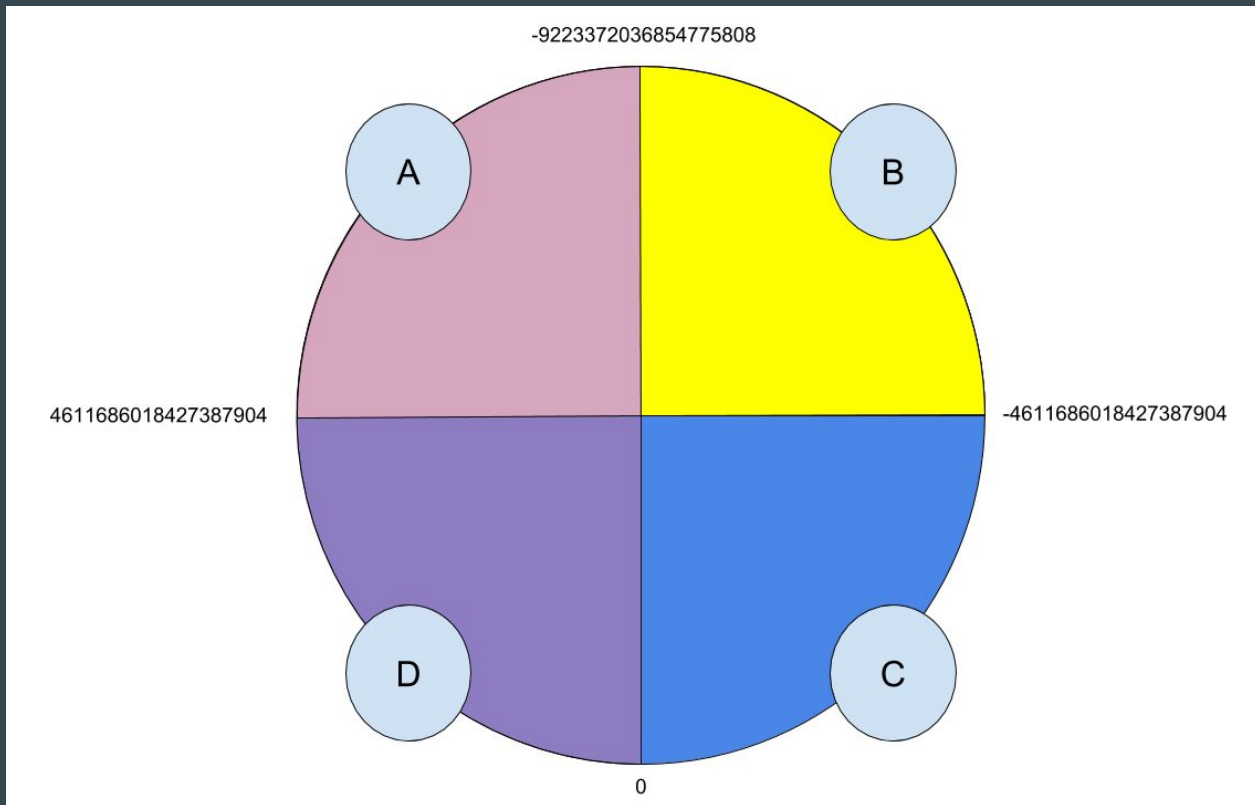


## Consistent hashing - les tokens

Murmur3Partitioner : tokens de  $-2^{63}$  à  $2^{63}-1$

$-9,223,372,036,854,775,808$  to  $9,223,372,036,854,775,807$

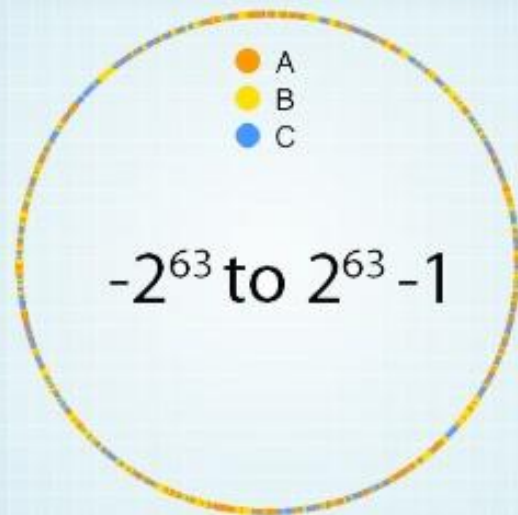
# Consistent hashing - sans vnodes



## Consistent hashing avec vnodes

# CONSISTENT HASHING - 3

num\_tokens: 256



virtual nodes

Basics - CQL = Cassandra Query Language

`hash(id_flux) = token`

## Basics - CQL = Cassandra Query Language

**INSERT INTO** myks.workflow

(id\_flux, id\_etape, start\_time)

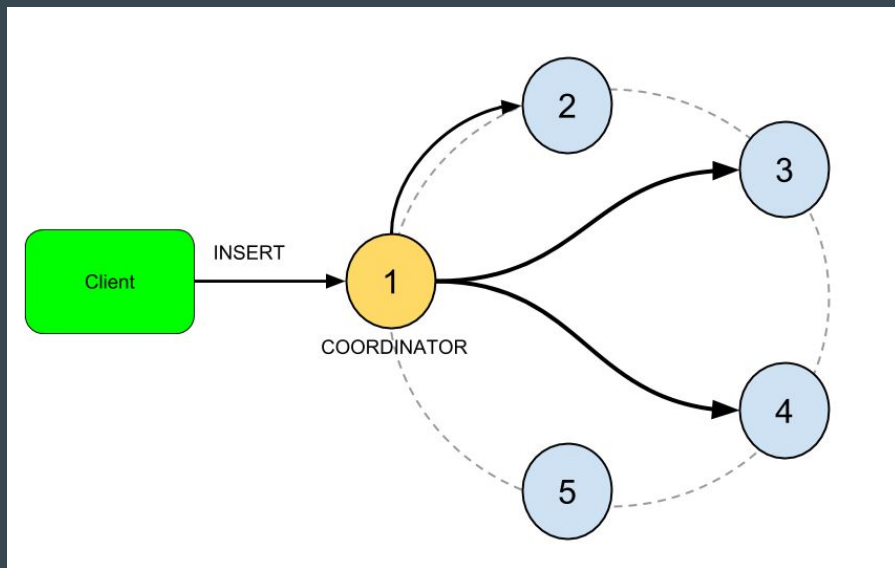
**VALUES** ('7172A', '345-131', '2018-03-19 15:32')

**hash(7172A) = 253**

Replica 1 for token 253 = Node 2

Replica 2 for token 253 = Node 4

Replica 3 for token 253 = Node 3



# Basics - CQL = Cassandra Query Language

**INSERT INTO** myks.workflow

(id\_flux, id\_etape, start\_time)

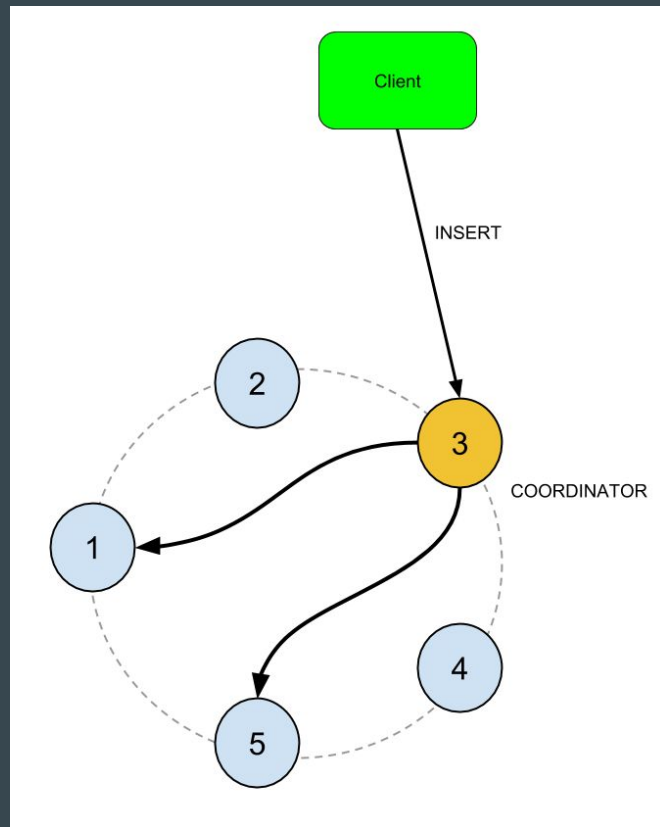
**VALUES** ('4864C', '345-131', '2018-03-19 19:45')

**hash(4864C) = 10325**

Replica 1 for token 10325 = Node 1

Replica 2 for token 10325 = Node 3

Replica 3 for token 10325 = Node 5



# Storage engine

## 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	ceo	37	dev

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

# Storage engine

## The CQL/Cassandra Mapping

```
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

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

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

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



## Basics - CQL : types

text, varchar, ascii

tinyint, bigint, decimal, double, float, int, smallint, varint

blob

boolean

counter

inet

list, map, set

tuple

time, timestamp, timeuuid, date

uuid

## Basics - CQL : ce n'est pas du SQL

- Pas de jointures
- Pas de OR
- Limitations sur les champs dans la clause WHERE
- Pas d'INSERT/SELECT
- Pas de vues
- Index peu performants
- GROUP BY introduit en version 3.10 (par partition uniquement)

Basics - CQL : Comment peut-on requêter cette table ?

```
CREATE TABLE myks.workflow(  
    id_flux varchar,  
    id_etape varchar,  
    start_time timestamp,  
    end_time timestamp,  
    status int,  
    PRIMARY KEY (id_flux, id_etape, start_time)  
) WITH CLUSTERING ORDER BY (id_etape ASC, start_time DESC);
```

## Basics - CQL : Comment peut-on requêter cette table ?

Lire toutes les étapes d'un flux spécifique

```
SELECT * FROM myks.workflow  
WHERE id_flux = 'value'
```

Lire une étape spécifique d'un flux spécifique

```
SELECT * FROM myks.workflow  
WHERE id_flux = 'value' AND id_etape = 'step'
```

Lire une étape donnée d'un flux spécifique si son heure de début est comprise entre deux bornes

```
SELECT * FROM myks.workflow  
WHERE id_flux = 'value' AND id_etape = 'step'  
AND start_time >= '2018-03-19 15:00' AND start_time < '2018-03-19 16:00';
```

## Basics - CQL : Comment peut-on requêter cette table ?

Par contre on ne peut pas :

Lire toute les étapes qui ont un id\_etape donné

**SELECT \* FROM** myks.workflow  La restriction sur la clé de partition manque  
**WHERE** id\_etape = 'step'

Lire toutes les étapes d'un flux dont l'heure de début est comprise entre deux bornes

**SELECT \* FROM** myks.workflow  
**WHERE** id\_flux = 'value'  
**AND** start\_time >= '2018-03-19 15:00' **AND** start\_time < '2018-03-19 16:00';

 id\_etape apparaît avant start\_time dans la clustering key

Basics - CQL : échangeons l'ordre des champs de la clé de clustering

```
CREATE TABLE myks.workflow_by_start_time(  
    id_flux varchar,  
    id_etape varchar,  
    start_time timestamp,  
    end_time timestamp,  
    status int,  
    PRIMARY KEY (id_flux, start_time, id_etape)  
) WITH CLUSTERING ORDER BY (start_time DESC, id_etape ASC);
```

## Basics - CQL : Maintenant on peut

Lire toutes les étapes d'un flux dont l'heure de début est comprise entre deux bornes

```
SELECT * FROM myks.workflow_by_start_time
```

```
WHERE id_flux = 'value'
```

```
AND start_time >= '2018-03-19 15:00' AND start_time < '2018-03-19 16:00';
```

## Basics - CQL : Mais on ne peut plus

Lire une étape précise d'un flux

```
SELECT * FROM myks.workflow_by_start_time  
WHERE id_flux = 'value' AND id_etape = 'step'
```

Nous aurions besoin d'avoir la valeur précise de start\_time (à la milliseconde)  
pour lire une étape

```
SELECT * FROM myks.workflow_by_start_time  
WHERE id_flux = 'value' AND start_time = 1520958714000 AND id_etape = 'step'
```



## Basics - CQL : Idempotence

Cassandra ne lit pas avant d'écrire

Les deux requêtes suivantes peuvent être exécutées à la suite, sans erreur :

```
INSERT INTO myks.workflow  
(id_flux, id_etape, start_time)  
VALUES ('flow1', 'step1', '2018-03-19 12:00');
```

```
INSERT INTO myks.workflow  
(id_flux, id_etape, start_time)  
VALUES ('flow1', 'step1', '2018-03-19 12:00');
```

Last Write Wins : le dernier insert écrase les précédents

## Basics - CQL : Idempotence

Aussi cette requête :

```
INSERT INTO myks.workflow  
(id_flux, id_etape, start_time, end_time)  
VALUES ('flow1', 'step1', '2018-03-19 12:00', '2018-03-19 12:10');
```

Est équivalente à celle-ci :

```
UPDATE myks.workflow  
SET end_time = '2018-03-19 12:10'  
WHERE id_flux='flow1' AND id_etape = 'step1' AND start_time = '2018-03-19 12:00';
```

Si l'enregistrement n'existe pas, un UPDATE le crée.

## Basics - CQL : DELETE

Un DELETE ne supprime pas réellement les données sur disque.

Il écrit une donnée spéciale appelée **TOMBSTONE**.

## Basics - CQL : DELETE statements

Donc supprimer des données crée **plus de données** sur disque.

Lire un grand nombre de tombstones génère beaucoup d'I/O et de **pression du GC**.

## Basics - CQL : Collections

Lists : permet les doublons - éléments ordonnés par date d'insertion

```
CREATE TABLE table_with_list (id int PRIMARY KEY, my_list list<text>);
```

## Basics - CQL : Collections

```
UPDATE table_with_list SET my_list = my_list + ['value1','value3']  
WHERE id = 1;
```

```
UPDATE table_with_list SET my_list = my_list + ['value2','value3']  
WHERE id = 1;
```

## Basics - CQL : Collections

```
select * from table_with_list;
```

```
id | my_list
```

```
-----+-----
```

```
1 | ['value1', 'value3', 'value2', 'value3']
```

## Basics - CQL : Collections

Sets : fusionne les doublons - éléments ordonnés alphabétiquement

```
CREATE TABLE table_with_set (id int PRIMARY KEY, my_set set<text>);
```



## Basics - CQL : Collections

```
UPDATE table_with_set SET my_set = my_set + {'value3','value1'}  
WHERE id = 1;
```

```
UPDATE table_with_set SET my_set = my_set + {'value2','value3'}  
WHERE id = 1;
```

## Basics - CQL : Collections

```
select * from table_with_set;
```

id	my_set
1	{'value1', 'value2', 'value3'}

## Basics - CQL : Collections

Maps : un key/value store.

```
CREATE TABLE table_with_map  
(id int PRIMARY KEY, my_map map<text, int>);
```

## Basics - CQL : Collections

```
UPDATE table_with_map SET my_map['key1'] = 1  
WHERE id = 1;
```

```
UPDATE table_with_map SET my_map['key2'] = 2  
WHERE id = 1;
```

## Basics - CQL : Collections

```
select * from table_with_map;
```

id	my_map
1	{ 'key1': 1, 'key2': 2 }

## Basics - CQL : Counters

Mise à jour par incréments - **non idempotent**

```
CREATE TABLE table_with_counter  
(id int PRIMARY KEY, my_counter counter);
```

## Basics - CQL : Counters

```
UPDATE table_with_counter SET my_counter = my_counter + 10  
WHERE id = 1;
```

```
UPDATE table_with_counter SET my_counter = my_counter + 15  
WHERE id = 1;
```

## Basics - CQL : Counters

```
select * from table_with_counter;
```

id	my_counter
----	------------

1	25
---	----



## Basics - CQL : TTL

Les insertions peuvent contenir une durée de vie, après quoi les cellules deviennent des **tombstones**.

## Basics - CQL : TTL

```
INSERT INTO myks.workflow (id_flux, id_etape, start_time, end_time)  
VALUES ('flow1', 'step1', '2018-03-19 12:00', '2018-03-19 12:10')  
USING TTL 86400
```

## Basics - CQL : TTL

UPDATE myks.workflow

USING TTL 86400

SET end\_time = '2018-03-19 12:10'

WHERE id\_flux='flow1' AND id\_etape = 'step1'

AND start\_time = '2018-03-19 12:00';

## Basics - CQL : Bucketing

```
CREATE TABLE IF NOT EXISTS myks.workflow_par_jour(  
    id_flux varchar,  
    jour varchar, // 2018-04-18  
    id_etape varchar,  
    start_time timestamp,  
    end_time timestamp,  
    status int,  
    PRIMARY KEY ((id_flux, jour), id_etape, start_time)  
);
```

Lire tous les enregistrements d'un flux nécessite d'exécuter une requête par jour

## Basics - CQL : Index secondaires

```
CREATE TABLE IF NOT EXISTS myks.workflow(  
    id_flux varchar,  
    id_etape varchar,  
    start_time timestamp,  
    end_time timestamp,  
    status int,  
    PRIMARY KEY (id_flux, id_etape, start_time)  
);  
  
CREATE INDEX workflow_status ON myks.workflow (status);
```

## Basics - CQL : Secondary indexes

```
select * from myks.workflow where status = 1;
```

id_flux	id_etape	start_time	end_time	status
flow1	step1	2018-03-19 12:00:00+0100	2018-03-19 12:10:00+0100	1

## Basics - CQL : Secondary indexes

Évitez les au possible :

- Index locaux : il ne “scalent” pas avec le cluster
- Non adaptés aux hautes/basses cardinalités
- Utilisables uniquement avec égalités strictes

# Basics - CQL : DDL

```
CREATE TABLE IF NOT EXISTS myks.workflow(
```

```
...
```

```
    PRIMARY KEY (id_flow, id_step, start_time)
```

```
) WITH bloom_filter_fp_chance = 0.01
```

```
    AND caching = {'keys': 'ALL', 'rows_per_partition': 'NONE'}
```

```
    AND comment = "
```

```
    AND compaction = {'unchecked_tombstone_compaction': 'true', 'tombstone_compaction_interval': '86400', 'tombstone_threshold': '0.5', 'class':  
'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy'}
```

```
    AND compression = {'chunk_length_kb': '64', 'sstable_compression': 'org.apache.cassandra.io.compress.LZ4Compressor'}
```

```
    AND dclocal_read_repair_chance = 0.1
```

```
    AND default_time_to_live = 0
```

```
    AND gc_grace_seconds = 864000
```

```
    AND max_index_interval = 2048
```

```
    AND memtable_flush_period_in_ms = 0
```

```
    AND min_index_interval = 128
```

```
    AND read_repair_chance = 0.0
```

```
    AND speculative_retry = '99.0PERCENTILE';
```



# Hands on

Créez un cluster Cassandra 3.0.16 :

```
ccm create devoxx2018 -v binary:3.0.16 -n 3
```

## Hands on

Démarrez le cluster :

```
sudo ifconfig lo0 alias 127.0.0.2
```

```
sudo ifconfig lo0 alias 127.0.0.3
```

```
sudo ifconfig lo0 alias 127.0.0.4
```

**ccm start**

# Hands on

## ccm status

```
Cluster: 'devovx2018'
```

```
-----
```

```
node1: UP
```

```
node3: UP
```

```
node2: UP
```

# Hands on

## Configuration des noeuds :

~/.ccm/devoxx2018/node1/conf/cassandra.yaml

~/.ccm/devoxx2018/node2/conf/cassandra.yaml

~/.ccm/devoxx2018/node3/conf/cassandra.yaml

# Hands on

## Seed nodes :

seed\_provider:

- class\_name: org.apache.cassandra.locator.SimpleSeedProvider

parameters:

- seeds: 127.0.0.1

# Tools

nodetool

info

status

cfstats / tablestats

tpstats

netstats

# Tools

## nodetool cfhistograms ks table

Percentile	SSTables	Write Latency (micros)	Read Latency (micros)	Partition Size (bytes)	Cell Count
50%	4.00	20.50	20924.30	24601	215
75%	60.00	24.60	43388.63	105778	924
95%	72.00	42.51	107964.79	545791	5722
98%	86.00	61.21	129557.75	1131752	9887
99%	86.00	88.15	155469.30	1629722	14237
Min	0.00	3.31	51.01	104	0
Max	86.00	962624.93	962624.93	20924300	219342

# Hands on

## ccm node1 nodetool status

```
Datacenter: datacenter1
```

```
=====
```

```
Status=Up/Down
```

```
|/ State=Normal/Leaving/Joining/Moving
```

--	Address	Load	Tokens	Owns (effective)	Host ID	Rack
UN	127.0.0.1	98,98 KiB	1	66,7%	728c1207-6fd8-4f9a-818c-7216222495a3	rack1
UN	127.0.0.2	98,96 KiB	1	66,7%	953024b8-3ffc-453e-8680-ec25d5d41be5	rack1
UN	127.0.0.3	98,97 KiB	1	66,7%	d74db85d-fc3e-460c-8701-a4b0b8a85d0b	rack1



## Hands on

Utilisation de **Cassandra Dataset Manager (CDM)** pour créer un schéma et l'alimenter automatiquement.

**movielens** : base de données de films et ratings, en général utilisé pour des exercices de filtrage collaboratif.

# Hands on

## Installation de CDM :

```
wget https://github.com/rustyrazorblade/cdm/releases/download/v0.2.0/cdm  
chmod 755 cdm
```

## Chargement des données de movielens :

```
./cdm install movielens
```

# movielens

```
CREATE TABLE movielens.movies (  
  id uuid PRIMARY KEY,  
  avg_rating float,  
  genres set<text>,  
  name text,  
  release_date date,  
  url text,  
  video_release_date date  
)
```

```
CREATE TABLE movielens.users (  
  id uuid PRIMARY KEY,  
  address text,  
  age int,  
  city text,  
  gender text,  
  name text,  
  occupation text,  
  zip text  
)
```

# movielens

```
CREATE TABLE movielens.ratings_by_user (  
  user_id uuid,  
  movie_id uuid,  
  name text,  
  rating int,  
  ts int,  
  PRIMARY KEY (user_id, movie_id)  
)
```

```
CREATE TABLE movielens.ratings_by_movie (  
  movie_id uuid,  
  user_id uuid,  
  rating int,  
  ts int,  
  PRIMARY KEY (movie_id, user_id)  
) WITH CLUSTERING ORDER BY  
  (user_id ASC)
```

## Shell CQL

```
ccm node1 cqlsh
```

```
DESCRIBE KEYSPACES
```

```
DESCRIBE KEYSPACE movielens
```

movielens

Impossible de rechercher par titre partiel ou par première lettre...

# movielens

```
CREATE TABLE movielens.movies_by_first_letter (  
  first_letter text,  
  first_word text,  
  id uuid,  
  avg_rating float,  
  genres set<text>,  
  name text,  
  release_date date,  
  url text,  
  video_release_date date,  
  PRIMARY KEY (first_letter, first_word, id)  
) WITH CLUSTERING ORDER BY (first_word ASC, id ASC);
```

# movielens

Clonez le repo du hands on :

```
git clone https://github.com/thelastpickle/devoxxfr2018.git  
cd devoxxfr2018 && git checkout part1-movielens
```

Compilez et exécutez le loader de movies\_by\_first\_letter:

```
cd devoxx2018
```

```
mvn package
```

```
java -jar target/devoxx2018-0.0.1-SNAPSHOT.jar
```



## Hands on : opérations

Ajouter un noeud au cluster existant :

```
ccm add node4 -i 127.0.0.4 -j 7400 -b
```

```
ccm node4 start
```

## Hands on : opérations

Retirer un noeud du cluster existant :

```
ccm node2 nodetool decommission
```

```
ccm node2 remove
```

## Basics - Consistency levels

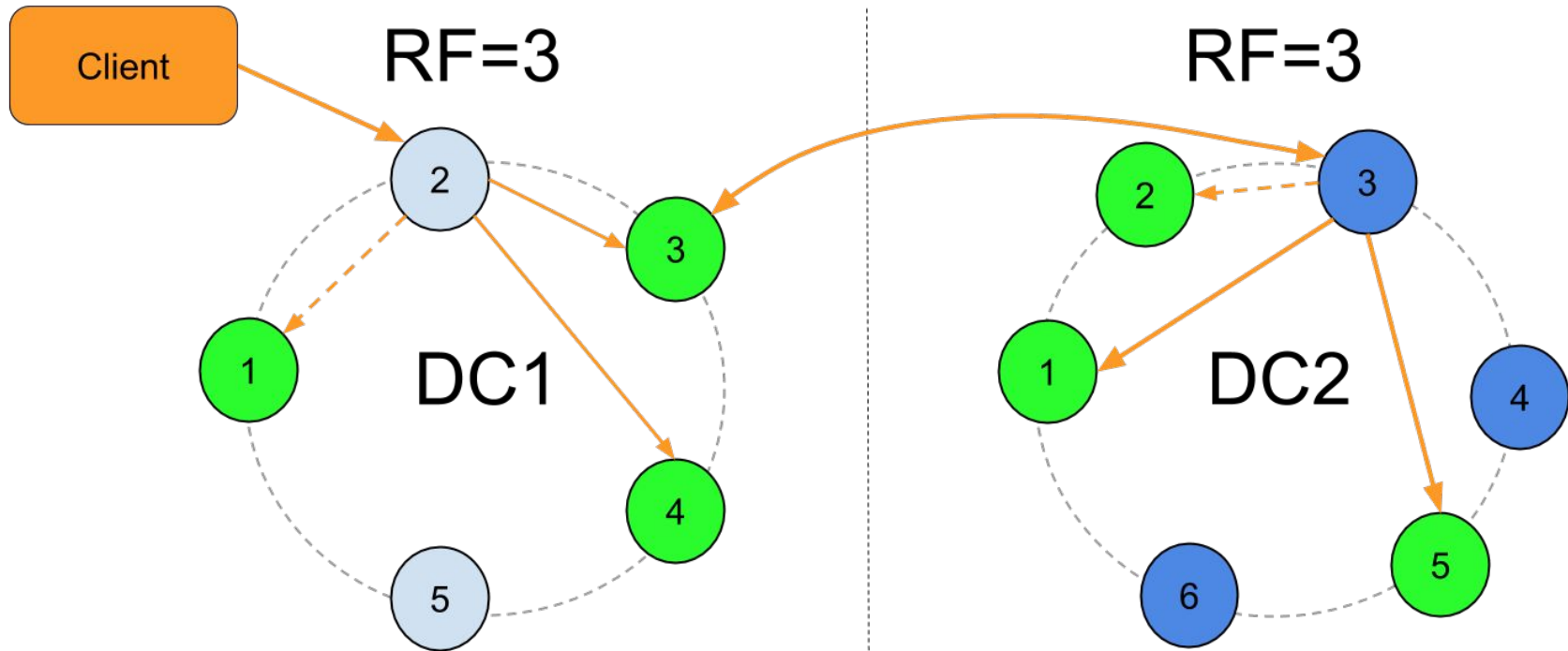
ONE / LOCAL\_ONE >> Cohérence in fine

QUORUM / LOCAL\_QUORUM / EACH\_QUORUM >> Cohérence forte

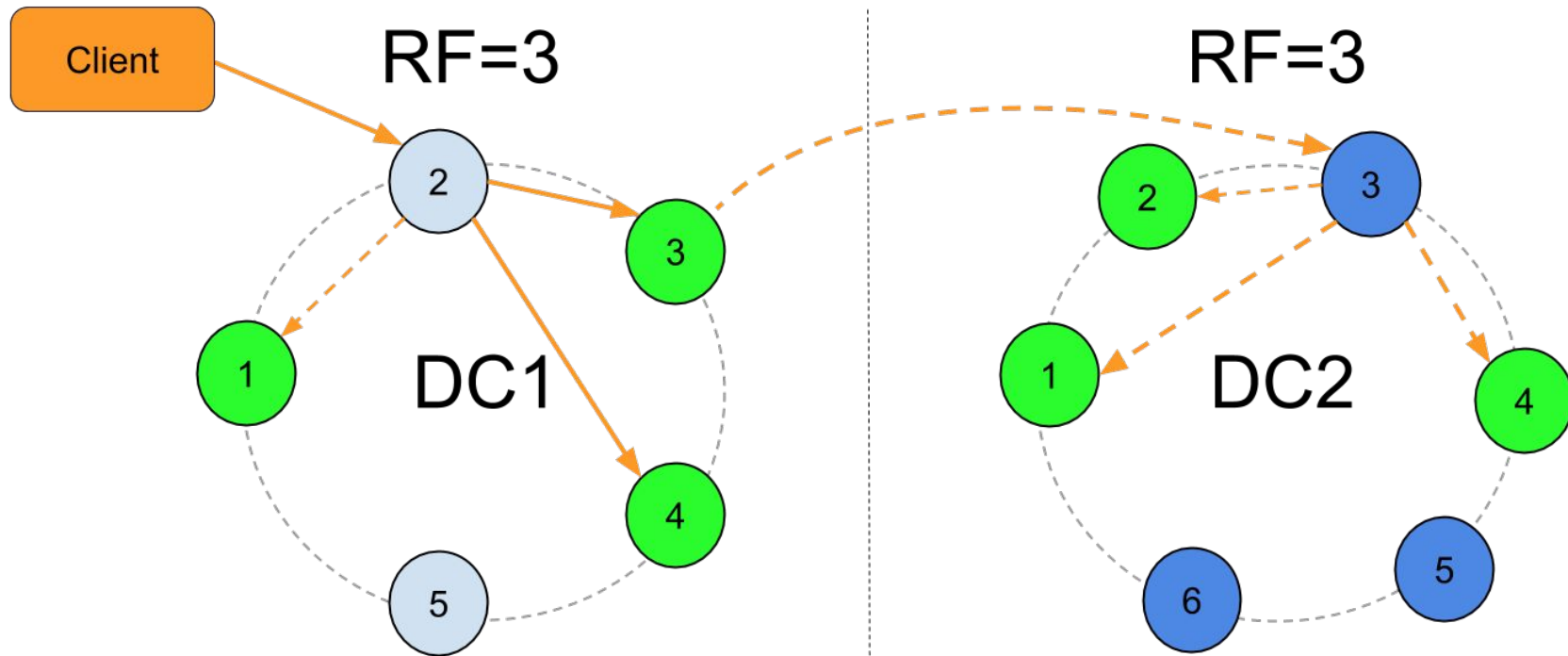
ALL >> Personne ne s'en sert...

ANY >> Les gens qui veulent se faire virer s'en servent...

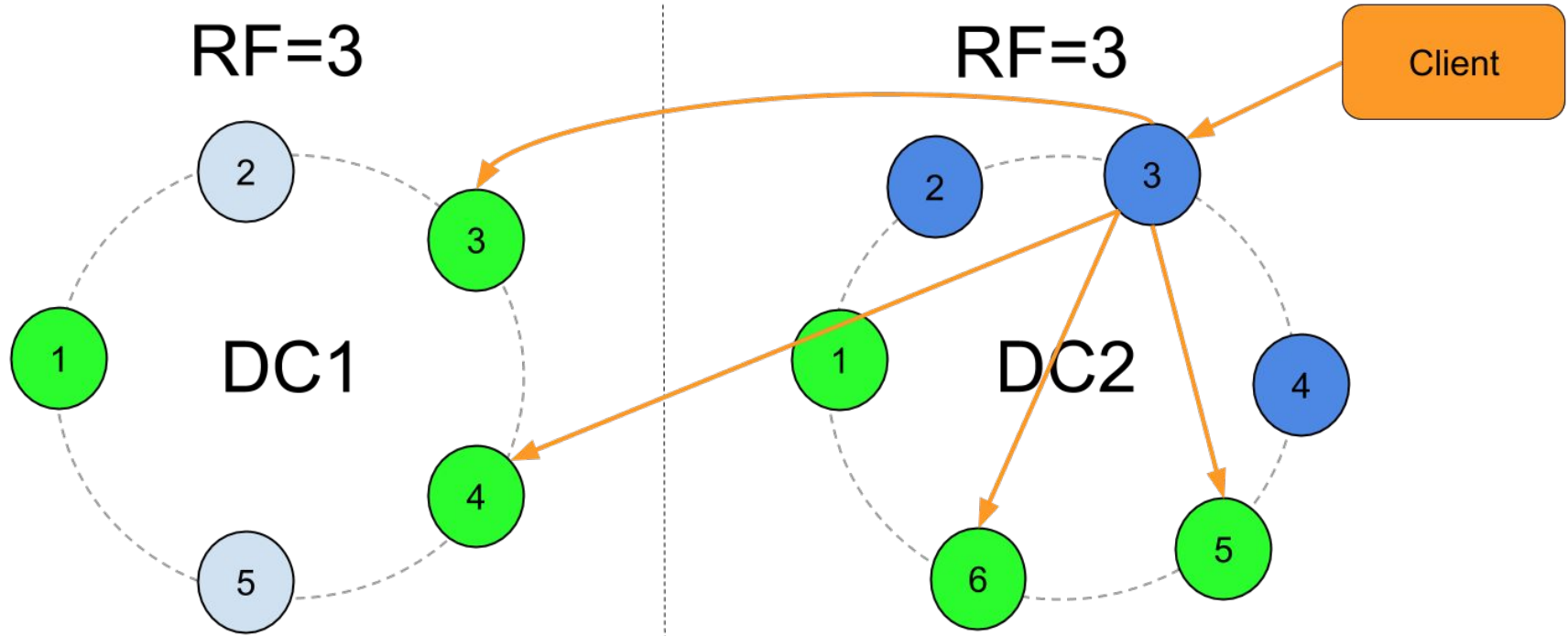
## Basics - Consistency levels - écriture au QUORUM



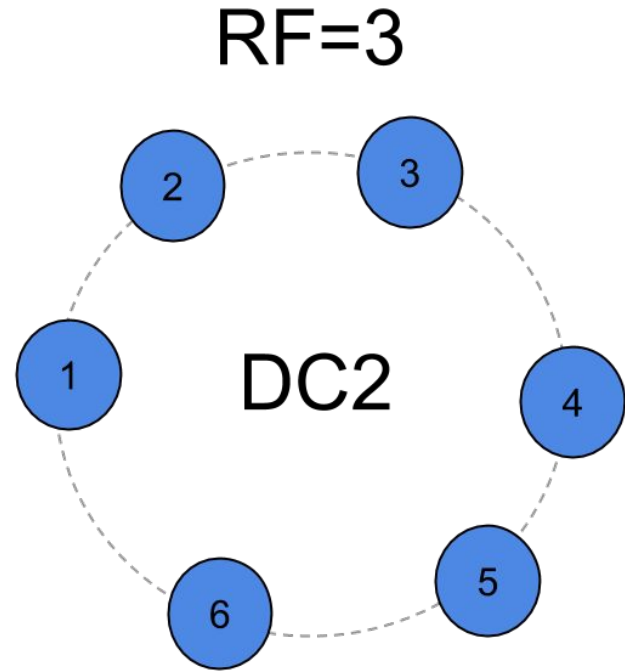
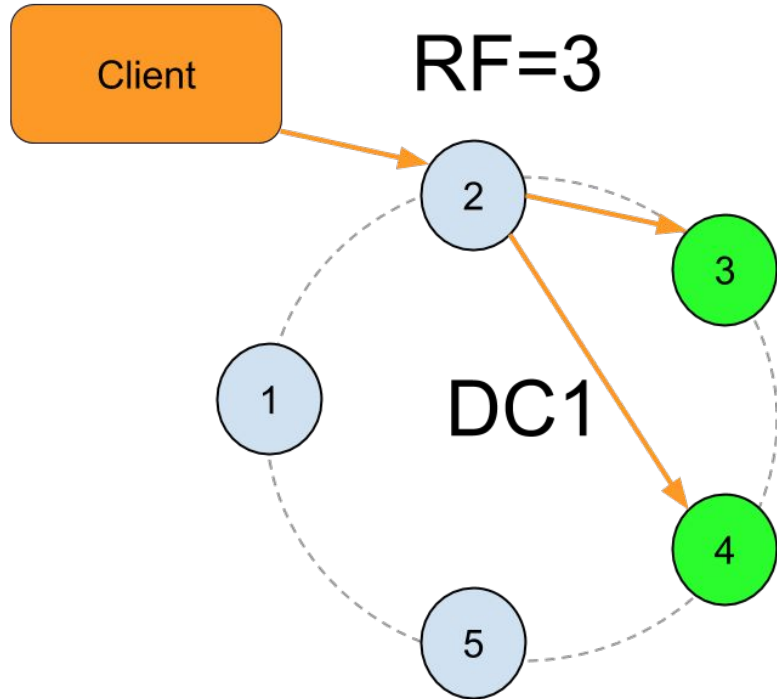
## Basics - Consistency levels - écriture au LOCAL\_QUORUM



## Basics - Consistency levels - lecture au QUORUM



## Basics - Consistency levels - lecture au LOCAL\_QUORUM



## CQL : **LightWeight Transactions**

Implémentation de l'algorithme Paxos.

Sérialisation des requêtes.

Effectue une lecture avant écriture.

4 aller-retours réseau au lieu d'un pour chaque écriture.



## CQL : LightWeight Transactions

```
CREATE TABLE users(  
  username text,  
  email text,  
  name text,  
  age int,  
  PRIMARY KEY(username));
```

## CQL : LightWeight Transactions

```
INSERT INTO users (username, email, name, age)
VALUES ('adejanovski', 'alex@thelastpickle.com', 'Alexander Dejanovski', 40)
IF NOT EXISTS
```

## CQL : LightWeight Transactions

Si l'insertion réussit :

```
[applied]
```

```
-----
```

```
True
```

Sinon Cassandra retourne l'enregistrement en conflit :

```
[applied] | username          | email                  | age | name
```

```
-----+-----+-----+-----+-----
```

```
False | adejanovski | alex@gmail.com | 40 | Alexander Dejanovski
```

## CQL : LightWeight Transactions

UPDATE users

SET age = 41

WHERE username = 'adejanovski'

IF age = 40

## CQL : **LightWeight Transactions**

Si la condition de la LWT n'est pas réalisée, Cassandra retourne la valeur actuelle :

```
[applied] | age
-----+-----
      False |  41
```

Client

Drivers Datastax pour Python, Java, etc...

Pas de pool de connexion ou de broker

Un seul objet Cluster par process et cluster physique

# Client

## Load balancing policies :

- RoundRobinPolicy
- DCAwareRoundRobinPolicy
- TokenAwarePolicy
- LatencyAwarePolicy
- WhiteListPolicy

Décide à chaque requête quel noeud va la recevoir

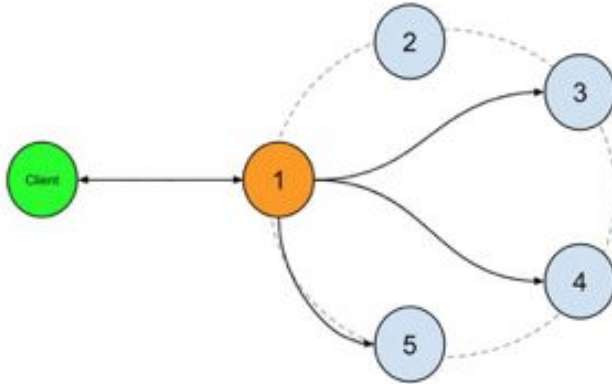
# Client

Par défaut, vous devriez utiliser :

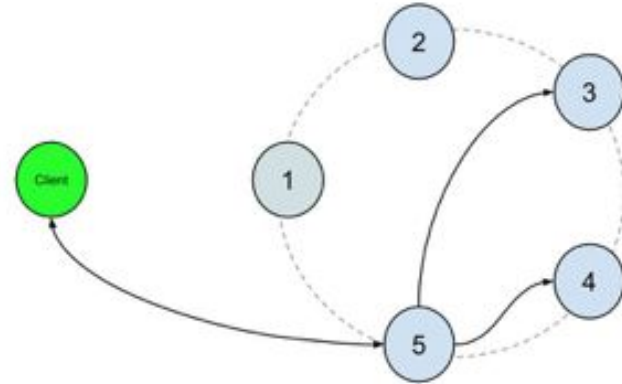
**TokenAwarePolicy(DCAwareRoundRobinPolicy())**



## Client - TokenAwarePolicy

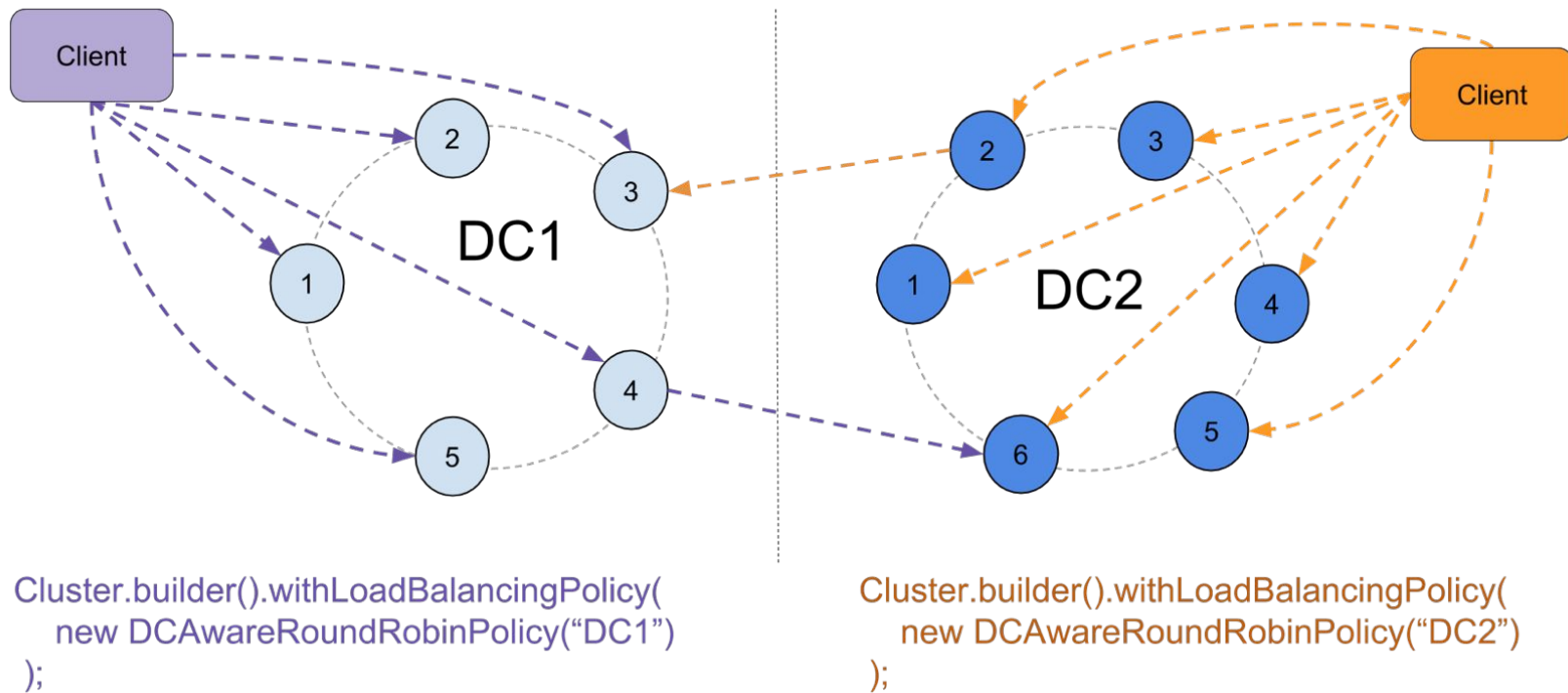


Sans Token Aware Policy  
*Sollicitation noeud 1 inutile*



Avec Token Aware Policy  
*Pas de sollicitation inutile*

# Client - DCAwareRoundRobinPolicy



# Client

## Retry policies :

- onUnavailable
- onReadTimeout
- onWriteTimeout
- onRequestError

Décide du comportement à adopter selon la cause d'échec d'une requête

Client

Query idempotence

# Java client

```
<dependency>  
  <groupId>com.datastax.cassandra</groupId>  
  <artifactId>cassandra-driver-core</artifactId>  
  <version>3.4.0</version>  
</dependency>
```

# Java client

```
Cluster cluster =  
    Cluster.builder()  
        .addContactPoint("127.0.0.1")  
        .withLoadBalancingPolicy(  
            new TokenAwarePolicy(DCAwareRoundRobinPolicy.builder().build())  
        )  
        .withRetryPolicy(DowngradingConsistencyRetryPolicy.INSTANCE)  
        .withQueryOptions(  
            new QueryOptions().setConsistencyLevel(ConsistencyLevel.LOCAL_QUORUM))  
        .build();
```

# Java client

```
Cluster cluster =  
    Cluster.builder()  
        .addContactPoint("127.0.0.1")  
        .withLoadBalancingPolicy(  
            new TokenAwarePolicy(DCAwareRoundRobinPolicy.builder().build())  
        )  
        .withRetryPolicy(DowngradingConsistencyRetryPolicy.INSTANCE)  
        .withQueryOptions(  
            new QueryOptions().setConsistencyLevel(ConsistencyLevel.LOCAL_QUORUM))  
        .build();
```

# Java client

```
Cluster cluster =  
    Cluster.builder()  
        .addContactPoint("127.0.0.1")  
        .withLoadBalancingPolicy(  
            new TokenAwarePolicy(DCAwareRoundRobinPolicy.builder().build())  
        )  
        .withRetryPolicy(DowngradingConsistencyRetryPolicy.INSTANCE)  
        .withQueryOptions(  
            new QueryOptions().setConsistencyLevel(ConsistencyLevel.LOCAL_QUORUM))  
        .build();
```



# Java client

```
Cluster cluster =  
    Cluster.builder()  
        .addContactPoint("127.0.0.1")  
        .withLoadBalancingPolicy(  
            new TokenAwarePolicy(DCAwareRoundRobinPolicy.builder().build())  
        )  
        .withRetryPolicy(DowngradingConsistencyRetryPolicy.INSTANCE)  
        .withQueryOptions(  
            new QueryOptions().setConsistencyLevel(ConsistencyLevel.LOCAL_QUORUM))  
        .build();
```

# Java client

```
Cluster cluster =  
    Cluster.builder()  
        .addContactPoint("127.0.0.1")  
        .withLoadBalancingPolicy(  
            new TokenAwarePolicy(DCAwareRoundRobinPolicy.builder().build())  
        )  
        .withRetryPolicy(DowngradingConsistencyRetryPolicy.INSTANCE)  
        .withQueryOptions(  
            new QueryOptions().setConsistencyLevel(ConsistencyLevel.LOCAL_QUORUM))  
        .build();
```

# Java client

```
Session session = cluster.connect("movielens");
```

```
ResultSet results = session.execute("SELECT * FROM movies");
```

```
for (Row row:results) {
```

```
    UUID id = row.getUUID("id");
```

```
    String name = row.getString("name");
```

```
    // ...
```

```
}
```

# Java client

```
Session session = cluster.connect("movielens");
```

```
ResultSet results = session.execute("SELECT * FROM movies");
```

```
for (Row row:results) {
```

```
    UUID id = row.getUUID("id");
```

```
    String name = row.getString("name");
```

```
    // ...
```

```
}
```

# Java client

```
Session session = cluster.connect("movielens");
```

```
ResultSet results = session.execute("SELECT * FROM movies");
```

```
for (Row row:results) {
```

```
    UUID id = row.getUUID("id");
```

```
    String name = row.getString("name");
```

```
    // ...
```

```
}
```

# Java client

```
Session session = cluster.connect("movielens");
```

```
ResultSet results = session.execute("SELECT * FROM movies");
```

```
for (Row row:results) {
```

```
    UUID id = row.getUUID("id");
```

```
    String name = row.getString("name");
```

```
    // ...
```

```
}
```

# Java client

```
PreparedStatement insert = session.prepare("INSERT INTO movies_by_first_letter "  
+      " (first_letter, first_word, id) VALUES (?, ?, ?)");
```

```
...
```

```
List<ResultSetFuture> futures = new ArrayList<>();  
for (Row row:results) {  
    futures.add(  
        session.executeAsync(  
            insert.bind(  
                row.getString("name").substring(0,1),  
                row.getString("name").split(" ")[0],  
                row.getUUID("id")  
            )  
        )  
    )  
}  
}
```

# Java client

```
PreparedStatement insert = session.prepare("INSERT INTO movies_by_first_letter "  
+      " (first_letter, first_word, id) VALUES (?, ?, ?)");
```

```
...
```

```
List<ResultSetFuture> futures = new ArrayList<>();  
for (Row row:results) {  
    futures.add(  
        session.executeAsync(  
            insert.bind(  
                row.getString("name").substring(0,1),  
                row.getString("name").split(" ")[0],  
                row.getUUID("id")  
            );  
        );  
    };  
};  
}
```



# Java client

```
PreparedStatement insert = session.prepare("INSERT INTO movies_by_first_letter "  
+      " (first_letter, first_word, id) VALUES (?, ?, ?)");
```

```
...
```

```
List<ResultSetFuture> futures = new ArrayList<>();  
for (Row row:results) {  
    futures.add(  
        session.executeAsync(  
            insert.bind(  
                row.getString("name").substring(0,1),  
                row.getString("name").split(" ")[0],  
                row.getUUID("id")  
            );  
        );  
    };  
};  
}
```

## Java client

```
for (ResultSetFuture future:futures) {  
    ResultSet result = future.getUninterruptibly();  
    ...  
}
```

Or

```
ListenableFuture<List<ResultSet>> results = Futures.successfulAsList(futures);  
for (ResultSet result:results.get()) {  
    ...  
}
```

# Java client

Pour commencer à traiter les résultats par ordre d'arrivée :

```
ImmutableList<ListenableFuture<ResultSet>> results = Futures.inCompletionOrder(readFutures);
for (ListenableFuture<ResultSet> result : results) {
    for (Row row : result.get()) {
        ...
    }
}
```

## Partie 2 : Coder une queue avec élection de leader

- Créez le keyspace devoxx dans votre cluster CCM :

```
CREATE KEYSPACE devoxx
```

```
WITH REPLICATION = {'class':'SimpleStrategy', 'replication_factor':3}
```

## Partie 2 : Coder une queue de messages

Table source

```
CREATE TABLE IF NOT EXISTS devoxx.messages(  
    id_queue int,  
    id_message timeuuid,  
    payload varchar,  
    published_by text,  
    PRIMARY KEY (id_queue, id_message)  
) WITH CLUSTERING ORDER BY (id_message ASC);
```

## Partie 2 : Coder une queue de messages

Table de sortie

```
CREATE TABLE IF NOT EXISTS devoxx.messages_ack(  
    id_queue int,  
    id_message timeuuid,  
    payload varchar,  
    published_by text,  
    processed_by list<text>,  
    PRIMARY KEY (id_queue, id_message)  
) WITH CLUSTERING ORDER BY (id_message DESC);
```

## Partie 2 : Coder une queue

Vous devez concevoir 2 classes :

- Un producteur qui écrit dans la table `devoxx.messages`
- Un consommateur qui :
  - lit le plus ancien enregistrement de la table
  - écrit un acquittement dans `devoxx.messages_ack` en ajoutant le nom du consommateur dans `processed_by`
  - supprime l'enregistrement de la table `devoxx.messages`

## Partie 2 : Coder une queue

**A vous de jouer !**

**git checkout part2-first-design-squelette**

**Le schéma de la base est dans src/main/resources**



## Partie 2 : Coder une queue

Un exemple d'implémentation :

`git checkout part2-first-design`

## Partie 2 : Coder une queue

2 problèmes avec ce design :

- On lit de plus en plus de tombstones au fil du temps
- Les messages peuvent être consommés par plusieurs participants

## Partie 2 : Coder une queue avec un design approprié

**Tombstones** : utiliser un partitionnement temporel  
(on ne s'intéressera qu'aux messages de moins de 2 minutes)

**Unicité de traitement** : Mettre en place un lock sur les enregistrements (type élection de master)  
grâce aux LWTs et au champ additionnel “processed\_by”

## Partie 2 : Coder une queue avec élection de leader

```
CREATE TABLE IF NOT EXISTS devoxx.messages_good(  
    id_queue int,  
    time_bucket bigint, // timestamp - résolution à la minute  
    id_message timeuuid,  
    payload varchar,  
    published_by text,  
    processed_by text, // mettre 'nobody' par défaut  
    PRIMARY KEY ((id_queue, time_bucket), id_message)  
);
```

## Partie 2 : Coder une queue avec élection de leader

**A vous de jouer !**

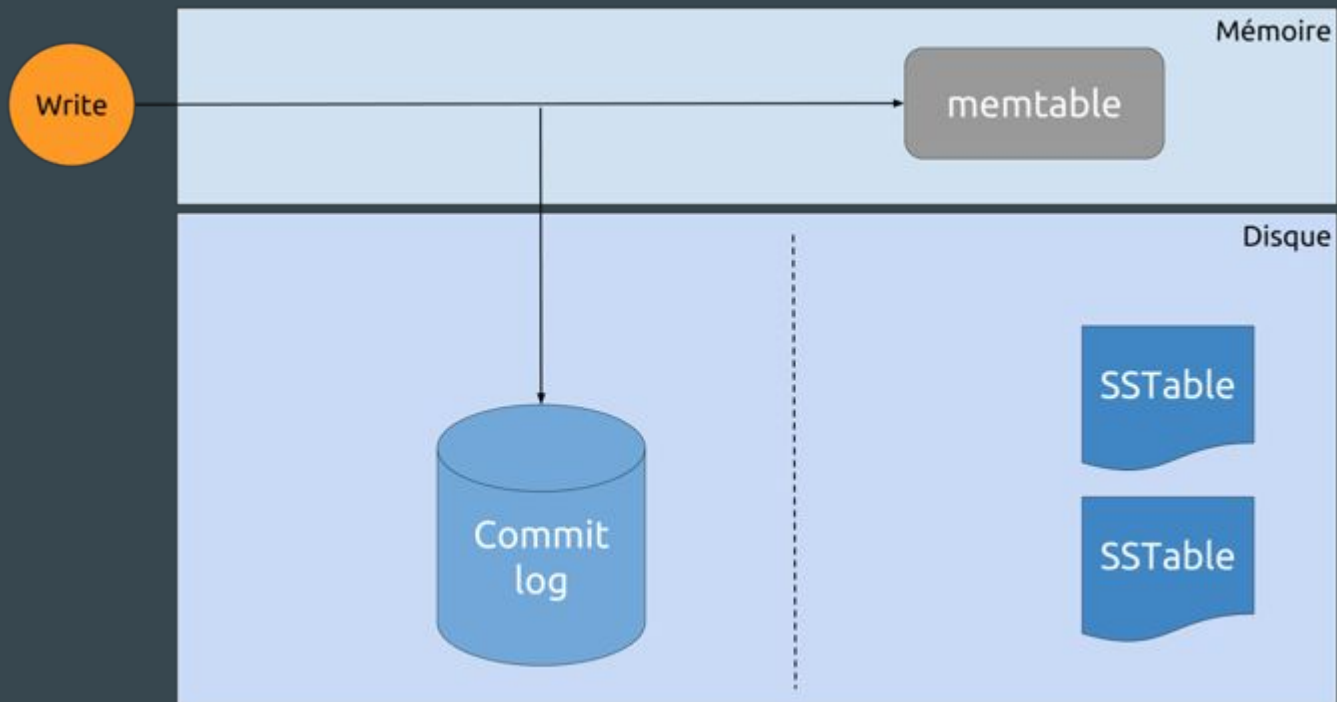
`git checkout part2-second-design-squelette`

## Partie 2 : Coder une queue avec élection de leader

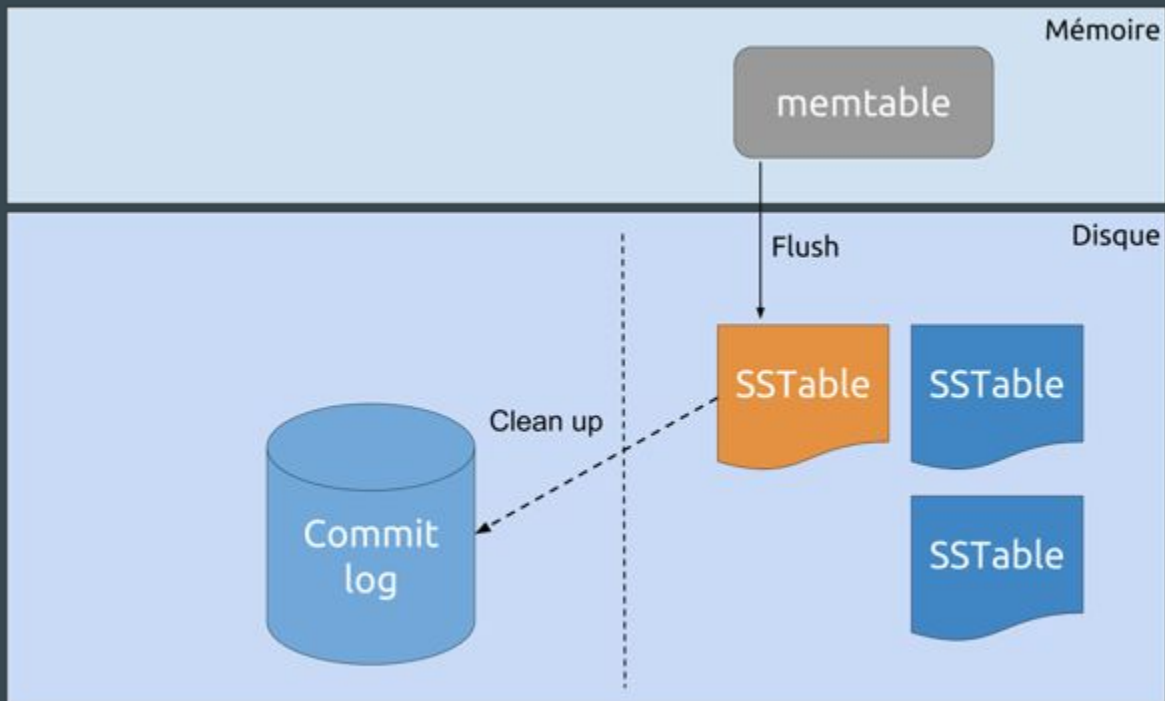
**L'implémentation complète :**

**`git checkout part2-second-design`**

## Basics - Write path

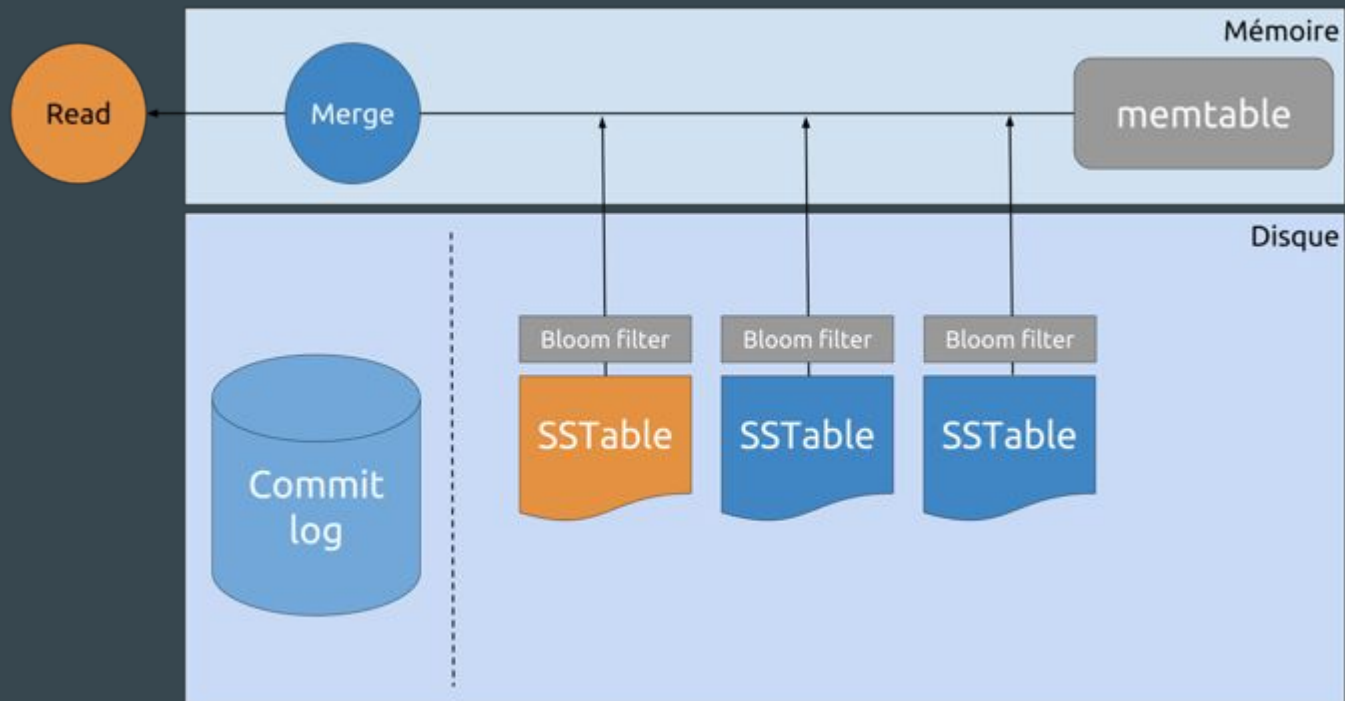


# Basics - Flush





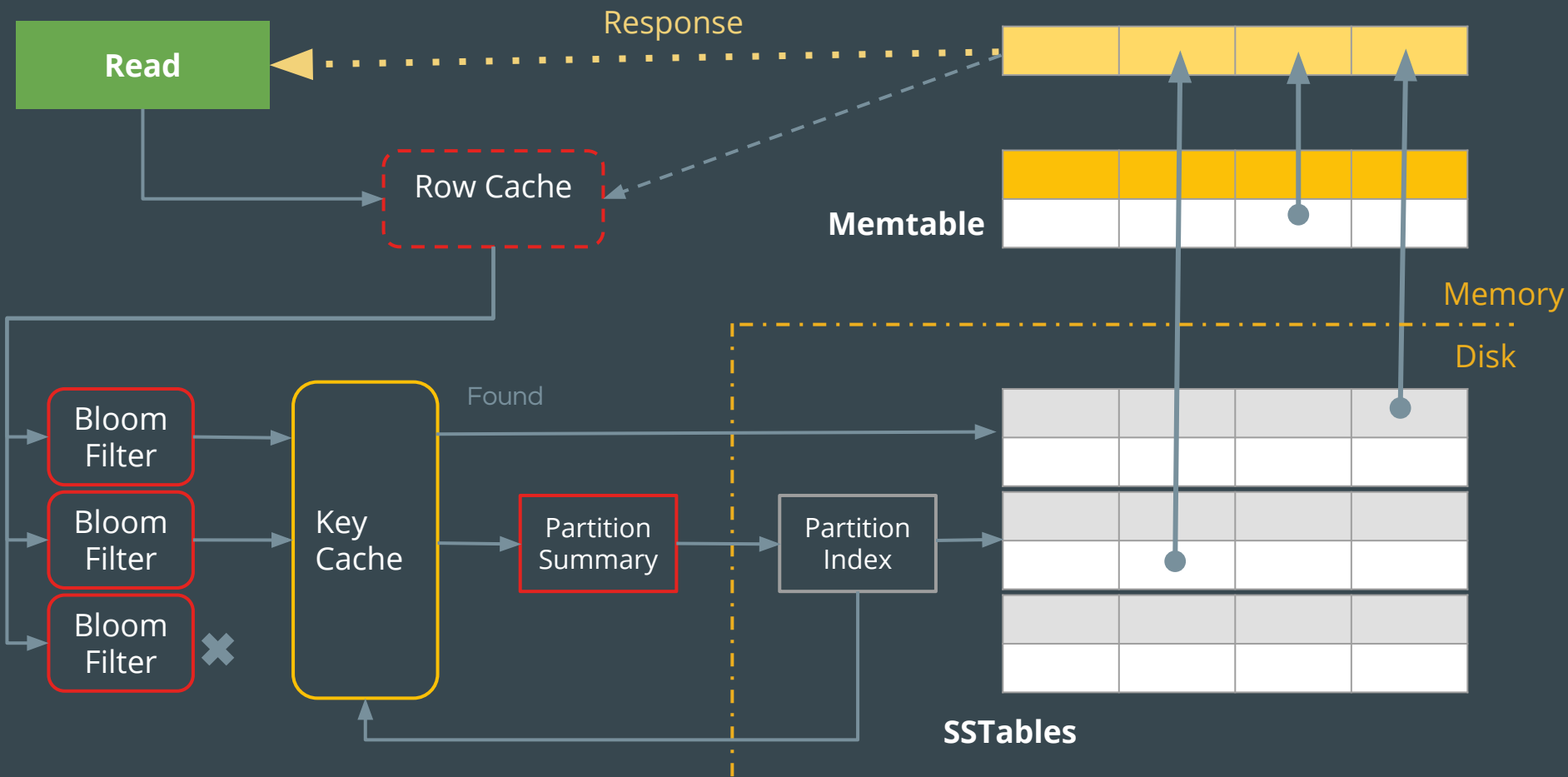
## Basics - Read Path



# Basics - Read Path (full)

On heap

Off heap



# Compaction

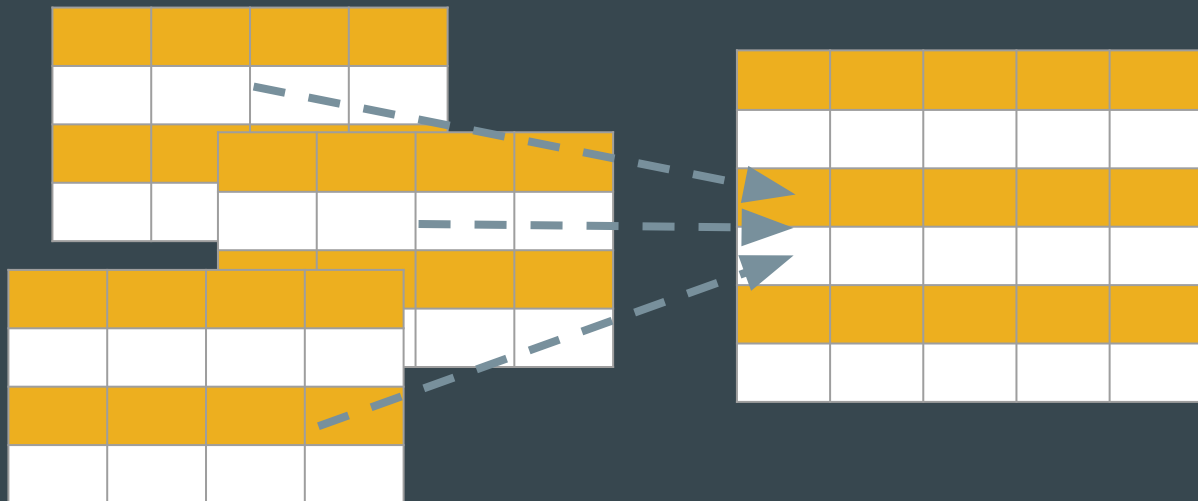
## Combine les SSTables ensemble

But:

Réduire le nombre de fichier à lire

Purge les données :

- obsolètes
- supprimées
- mises à jour



# Compaction

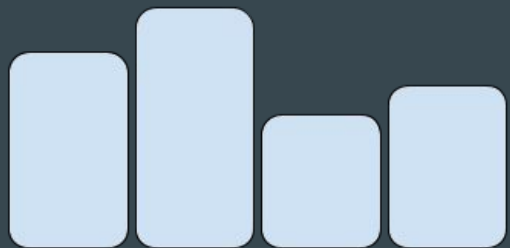
SizeTieredCompactionStrategy

LeveledCompactionStrategy

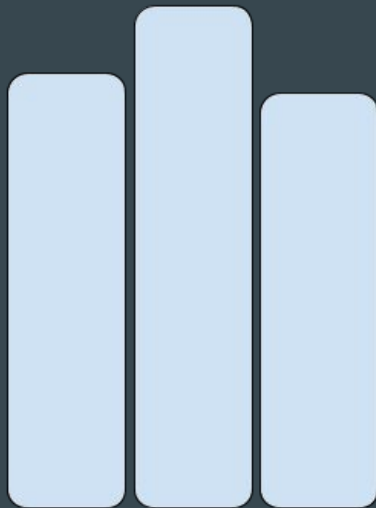
~~DateTieredCompactionStrategy~~

TimeWindowCompactionStrategy

## Compaction - STCS



Tier 0  
< 50MB

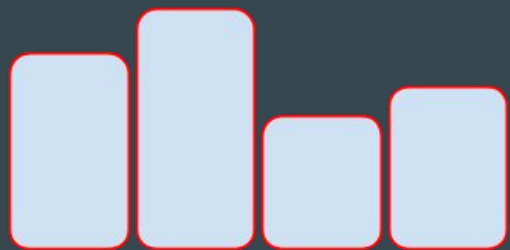


Tier 1  
~ 200MB



Tier 1  
~ 800MB

## Compaction - STCS



Tier 0  
< 50MB

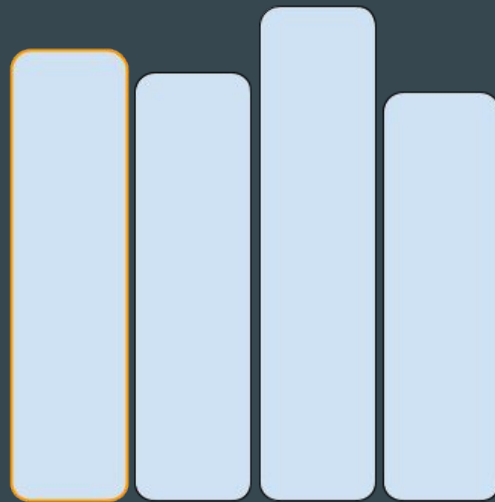


Tier 1  
~ 200MB



Tier 1  
~ 800MB

## Compaction - STCS



---

Tier 0  
**< 50MB**

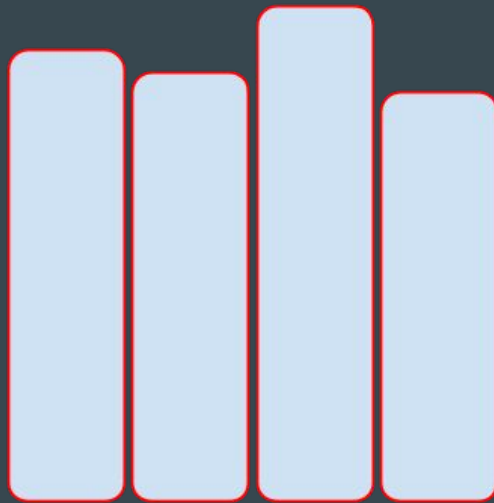
---

Tier 1  
**~ 200MB**

---

Tier 1  
**~ 800MB**

## Compaction - STCS



---

Tier 0  
< 50MB

---

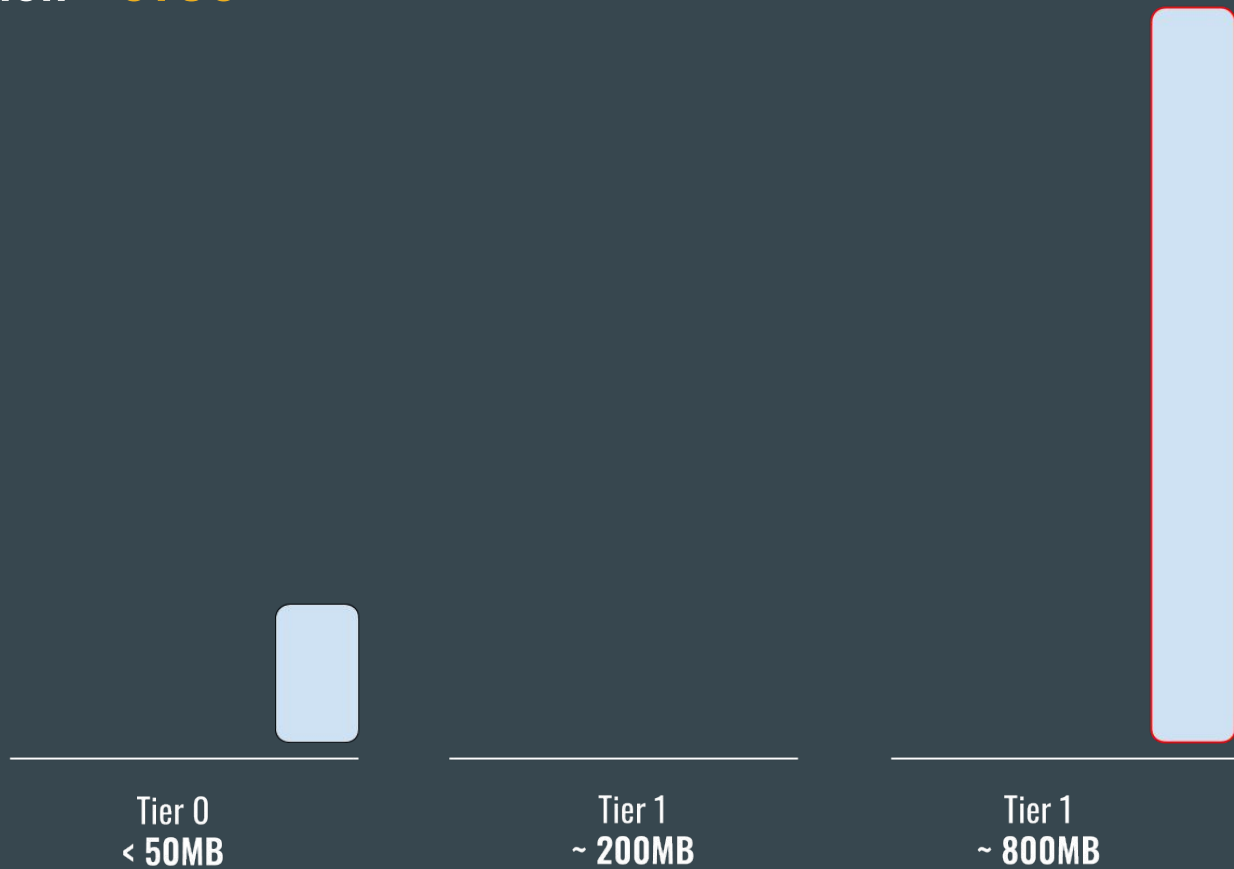
Tier 1  
~ 200MB

---

Tier 1  
~ 800MB



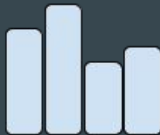
# Compaction - STCS



# Compaction - LCS

L0

Pas de limites



---

L1

10 SSTables de 160Mo

---

L2

100 SSTables de 160Mo

---

L3

1000 SSTables de 160Mo

---

# Compaction - LCS

L0

Pas de limites



---

L1

10 SSTables de 160Mo

---

L2

100 SSTables de 160Mo

---

L3

1000 SSTables de 160Mo

---

# Compaction - LCS

L0

Pas de limites

---

L1

10 SSTables de 160Mo



L2

100 SSTables de 160Mo

---

L3

1000 SSTables de 160Mo

---

# Compaction - LCS

L0

Pas de limites



L1

10 SSTables de 160Mo



L2

100 SSTables de 160Mo

L3

1000 SSTables de 160Mo

# Compaction - LCS

L0

Pas de limites

L1

10 SSTables de 160Mo



L2

100 SSTables de 160Mo

L3

1000 SSTables de 160Mo

# Compaction - LCS

L0

Pas de limites

L1

10 SSTables de 160Mo



L2

100 SSTables de 160Mo



L3

1000 SSTables de 160Mo

# Compaction - LCS

L0

Pas de limites



L1

10 SSTables de 160Mo



L2

100 SSTables de 160Mo



L3

1000 SSTables de 160Mo



# Compaction - LCS

L0

Pas de limites

L1

10 SSTables de 160Mo



L2

100 SSTables de 160Mo



L3

1000 SSTables de 160Mo

# Compaction - LCS

L0

Pas de limites

L1

10 SSTables de 160Mo



L2

100 SSTables de 160Mo



L3

1000 SSTables de 160Mo

# Compaction - LCS

L0

Pas de limites

L1

10 SSTables de 160Mo



L2

100 SSTables de 160Mo



L3

1000 SSTables de 160Mo

# Compaction - LCS

L0

Pas de limites

L1

10 SSTables de 160Mo



L2

100 SSTables de 160Mo



L3

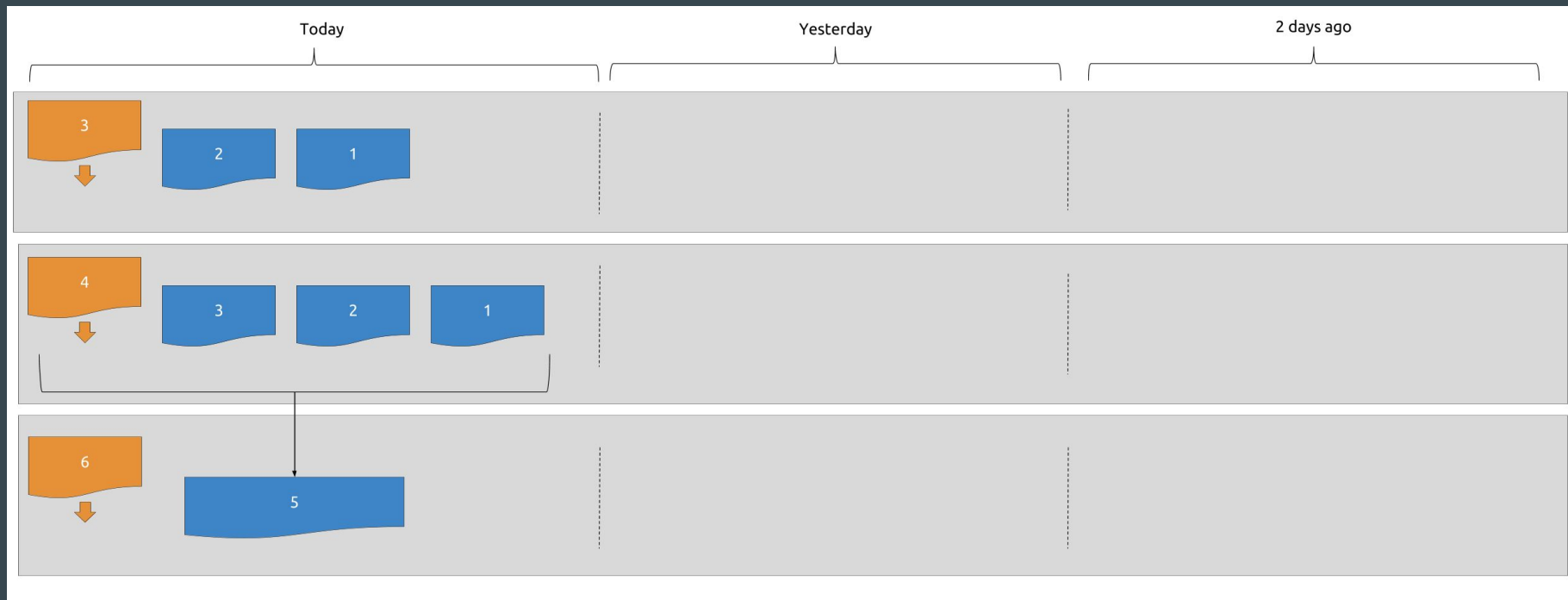
1000 SSTables de 160Mo



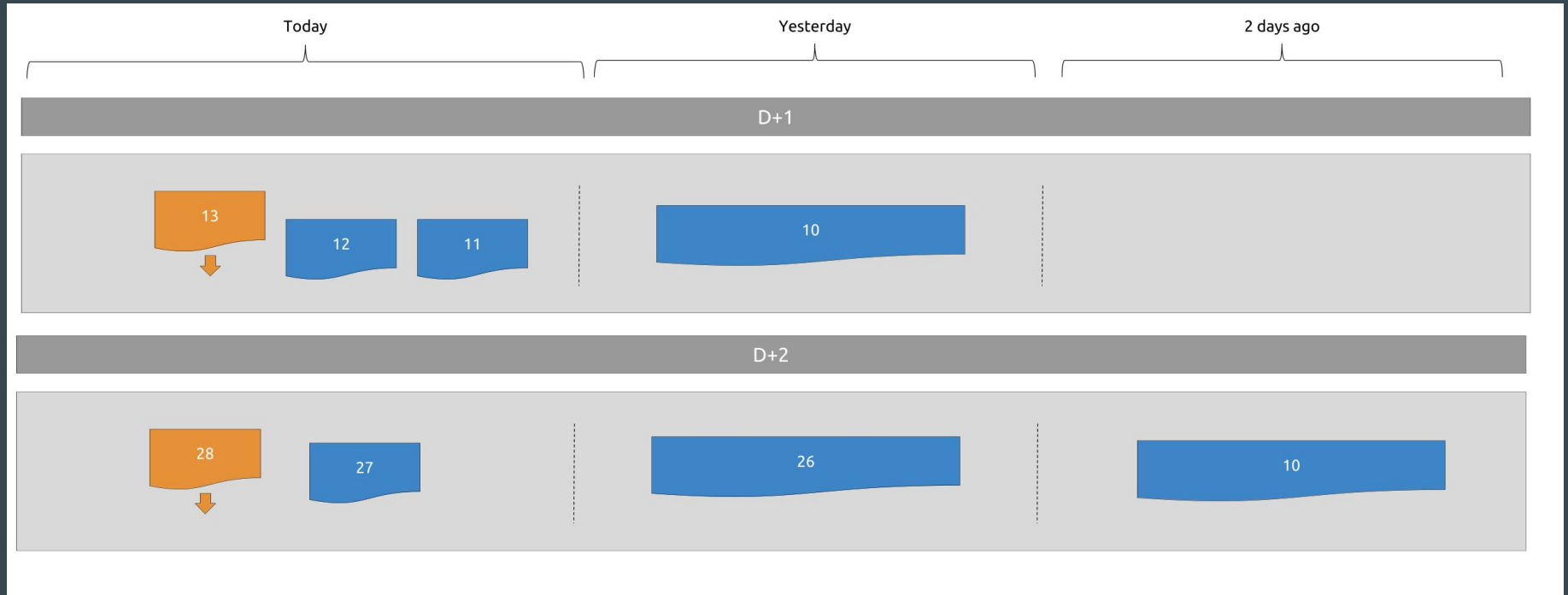
## Compaction - TWCS

```
CREATE TABLE test.my_table (  
    id int,  
    value int,  
    text_value text,  
    PRIMARY KEY (id, value)  
) WITH CLUSTERING ORDER BY (value ASC)  
    AND compaction = {  
        'compaction_window_size': '1',  
        'compaction_window_unit': 'DAYS'  
        'class': 'TimeWindowCompactionStrategy'  
    }
```

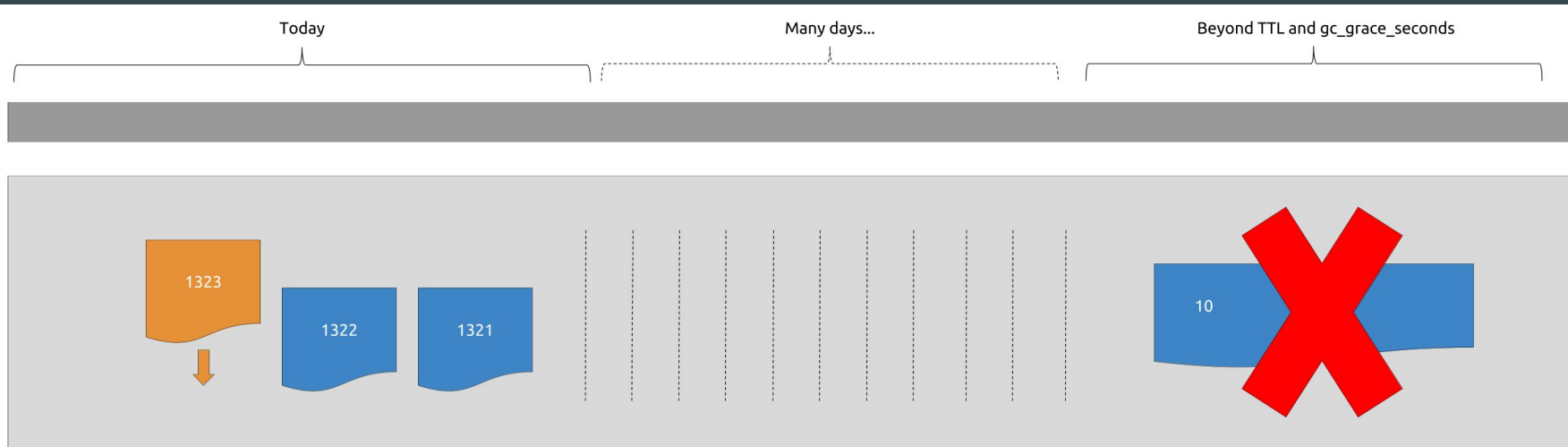
# Compaction - TWCS



# Compaction - TWCS

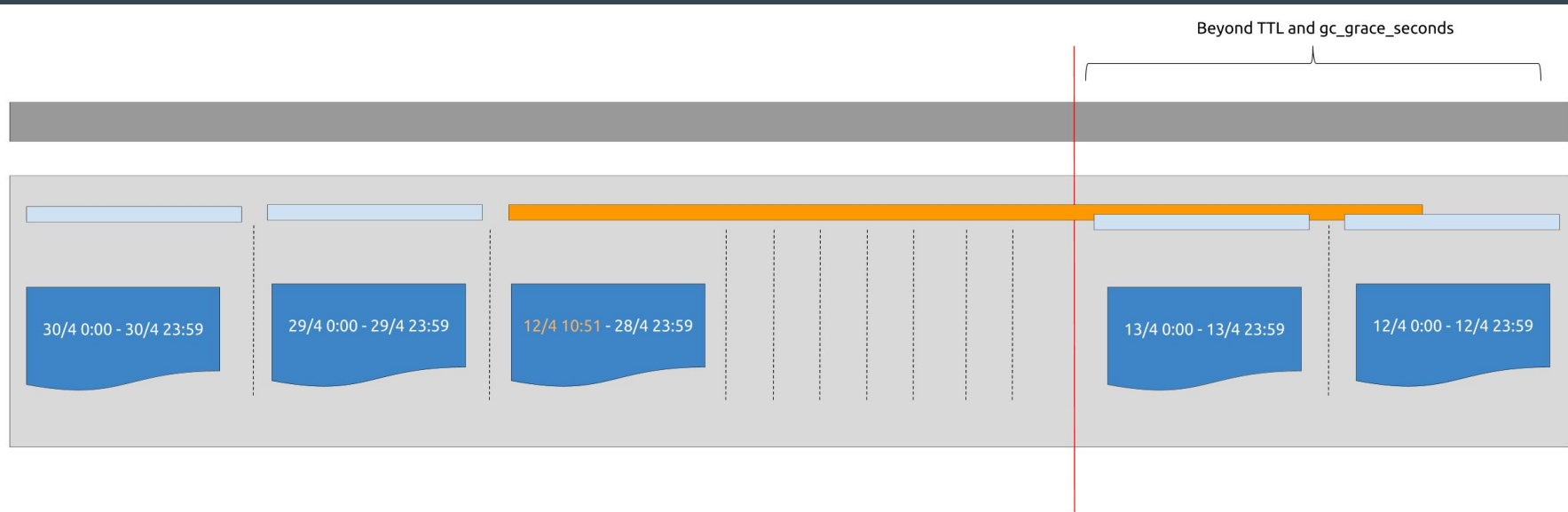


# Compaction - TWCS





# Compaction



# Réparation

Opération de maintenance remettant à niveau toutes les répliques.

# Réparation

Pour lancer une réparation :

**nodetool repair**

En vrai c'est une chouille plus compliqué...

Hands on : on casse et on répare

Allez dans :

`~/.ccm/devoxx2018/node1/data0/devoxx/messages***`

Et tapez :

`ccm node1 nodetool flush`

`rm -f *`

Hands on : on casse et on répare

Ensuite :

```
ccm node1 stop
```

```
ccm node1 start
```

```
ccm node1 cqlsh
```

```
> SELECT * FROM devoux.messages limit 1;
```

Hands on : on casse et on répare

Réparation :

```
ccm node1 nodetool repair
```

## Les hints

Dès qu'un noeud est DOWN, les noeuds actifs conservent pendant 3h les écritures manquées.

Cela s'appelle le **hinted handoff**.

Dès que le noeud revient, les hints lui sont envoyés avec un débit limité.