Modèle colonne

- **Principes**
- **HBase**
- **Cassandra**

Base de données colonnes

- Proches des bases de données relationnelles
- Données logiquement structurées en tables, lignes et colonnes
- Nombre de colonnes dynamique et pouvant varier d'un enregistrement à un autre
- Pas de stockage des valeurs NULL
- Stockage uniquement des données avec de valeurs
- Suit l'approche *BigTable* apportée par *Google* dont *HBase* est une implémentation open source

Colonnes et familles de colonnes

- **Colonne**
  - Couple clé/valeur
  - Représentation d'un champ de données
  - **Super Colonne** : colonne contenant d'autres colonnes
- **Famille de colonnes**
  - Regroupement de plusieurs colonnes ou super-colonnes
  - Colonnes sont regroupées en lignes
  - Identifiant unique pour chaque ligne

SuperColumns	
Key	**Value**
person1	Column **Name** : **Value** firstName : John lastName : Calagan
person2	Column **Name** : **Value** firstName : George lastName : Truffe

ColumnFamily	
Key	**Value**
AddressBook	SuperColumns **Key** : **Value** person1 : Column **Name** : **Value** firstName : John lastName : Calagan person2 : Column **Name** : **Value** firstName : George lastName : Truffe

Base de données colonnes : cas d'utilisation

- **Cas d'utilisation :**
  - Journalisation d'évènements ou comptage et catégorisation des visiteurs d'une page web
  - Utile pour les données éparses
  - Utile pour les tâches d'analyses sur des colonnes et dans les traitements massifs (via des opérations de *MapReduce*)
- **Non applicable aux:**
  - Applications avec des besoins de transactions ACID
  - Données interconnectés (distance, trajectoire)
  - Agrégation (effectuée par l'application client)

Principaux moteurs orientés colonne

☐ include secondary database models

13 systems in ranking, February 2023

Rank			DBMS	Database Model	Score		
Feb 2023	Jan 2023	Feb 2022			Feb 2023	Jan 2023	Feb 2022
1.	1.	1.	Cassandra	Wide column	116.22	-0.09	-7.76
2.	2.	2.	HBase	Wide column	38.41	-0.93	-5.21
3.	3.	3.	Microsoft Azure Cosmos DB	Multi-model	36.51	-1.45	-3.45
4.	4.	4.	Datastax Enterprise	Wide column, Multi-model	8.20	+0.51	-2.54
5.	5.	5.	Microsoft Azure Table Storage	Wide column	5.93	+0.11	+0.28
6.	↑ 7.	6.	Google Cloud Bigtable	Multi-model	5.92	+0.45	+1.75
7.	↓ 6.	7.	Accumulo	Wide column	5.75	+0.05	+1.82
8.	8.	8.	ScyllaDB	Wide column, Multi-model	5.63	+0.30	+1.75
9.	9.	9.	HPE Ezmeral Data Fabric	Multi-model	1.29	+0.04	+0.45
10.	10.	10.	Amazon Keyspaces	Wide column	0.91	+0.06	+0.35
11.	11.	11.	Elassandra	Wide column, Multi-model	0.63	+0.11	+0.12
12.	12.	12.	Alibaba Cloud Table Store	Wide column	0.23	-0.05	-0.23
13.	13.	13.	SWC-DB	Wide column, Multi-model	0.02	-0.01	-0.03

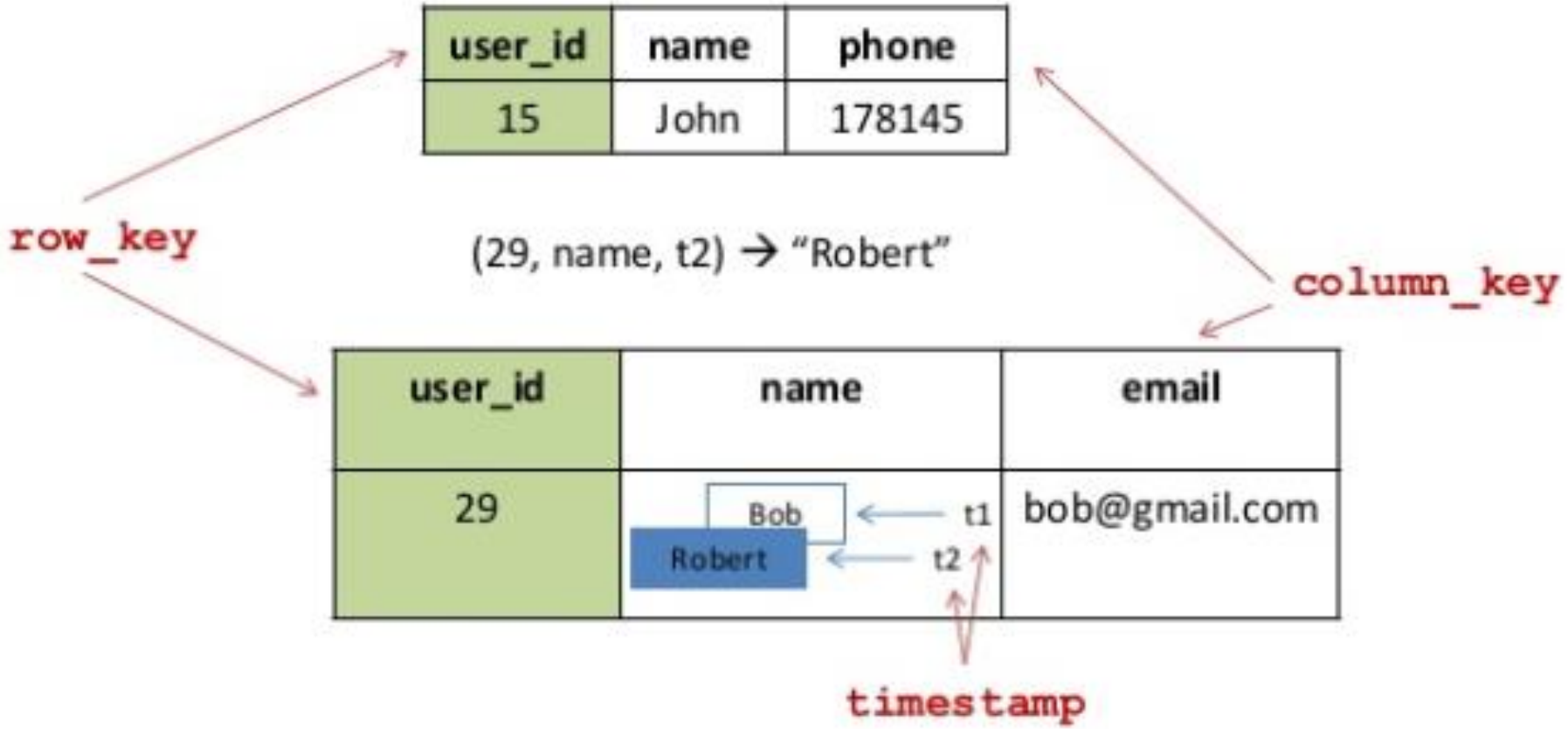
BigTable

- Base de données propriétaire gérée en interne chez Google
- Accessible au public uniquement via Google App Engine
- Ecrite en C++
- Cohérence forte (*strong consistency*)
- Stockage de données basé sur le système de fichiers distribués GFS (*Google File Systems*)
- Site officiel : https://cloud.google.com/bigtable/
- Article d'origine : http://static.googleusercontent.com/media/research.google.com/fr//archive/bigtable-osdi06.pdf

BigTable : concepts de base

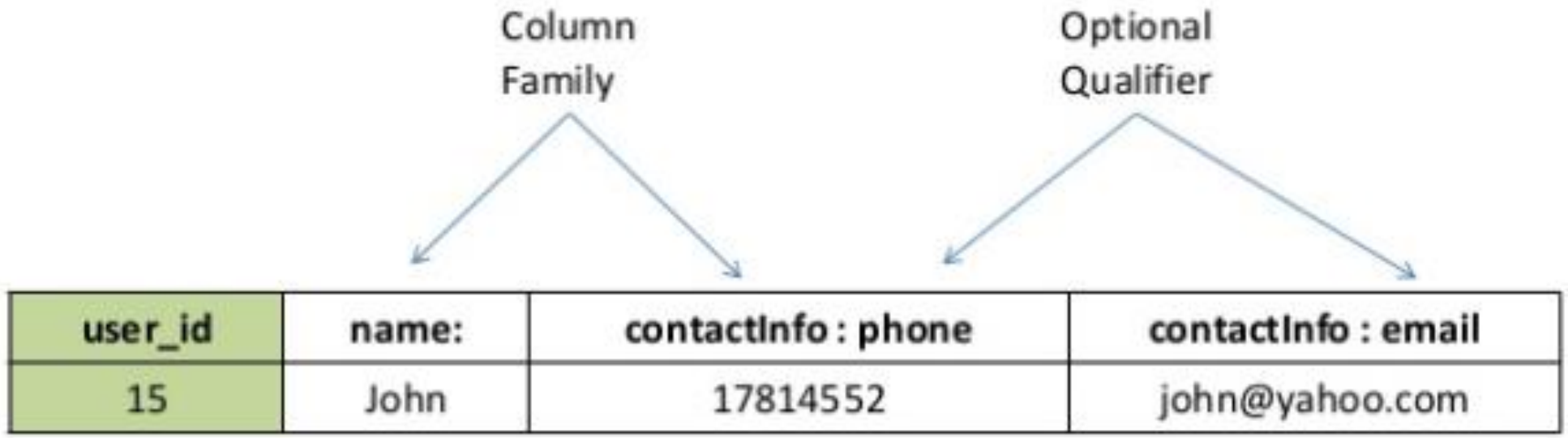
- *Map* multi-dimensionnelle
- *Row key* : clé unique d'une entité (*string* de 64KB)
- *Column family* : groupe d'attributs (colonnes) reliés
- *Column* : comporte différentes versions de la donnée (ordonnées par *timestamp* décroissant)
- Accès à une donnée : *Row key* $\rightarrow$ *column family* $\rightarrow$ *column* $\rightarrow$ *timestamp*

BigTable : modèle de données (clés et estampilles)



RDBMS Approach	user_id	name	phone	email
	15	John	178145	null
	29	Bob	null	bob@gmail.com

BigTable : modèle de données (famille de colonnes et *qualifier*)



RDBMS Approach	user_id	name
	15	John

user_id	type	value
15	phone	178145
15	email	john@yahoo.com

HBase

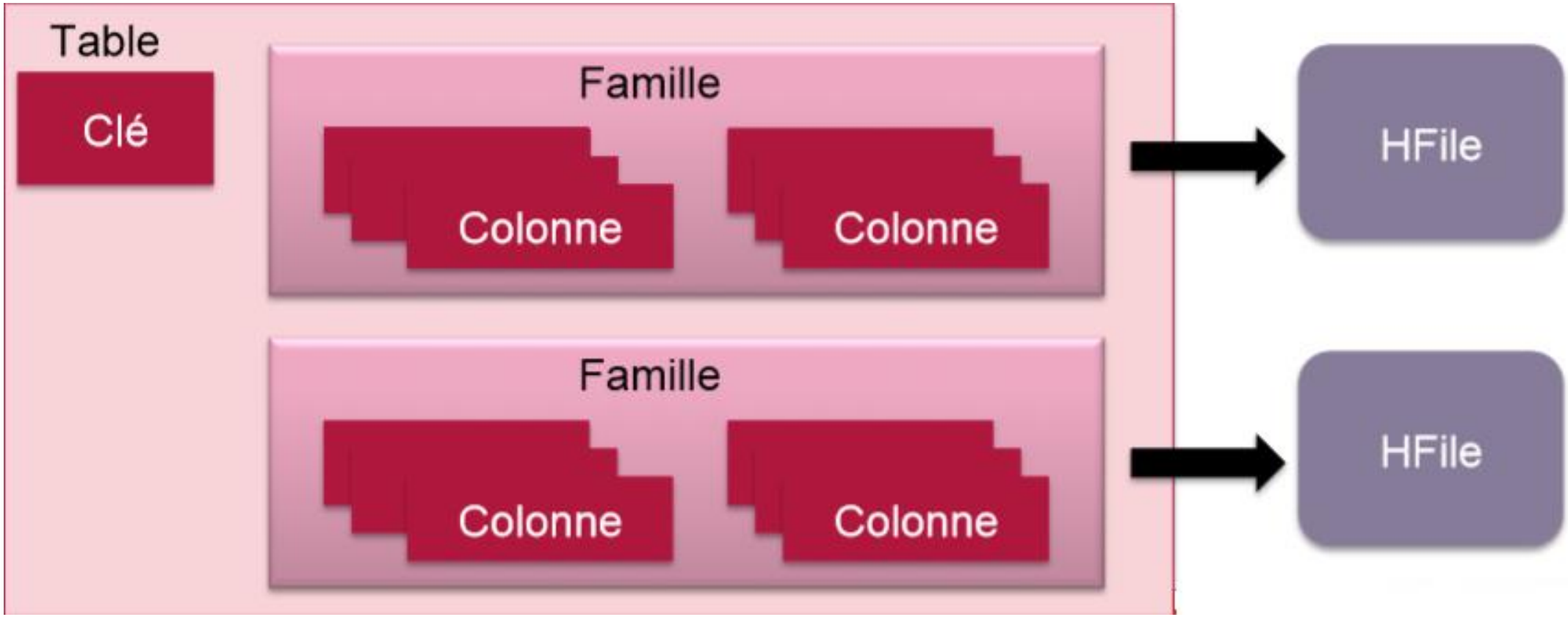
- Implémentation libre du moteur *BigTable* de *Google*
- Partie du projet Hadoop de Apache
- Écrit en Java
- Architecture Maître/Esclave
- Utilise HDFS (*Hadoop Distributed File System*) et ZooKeeper (système *open-source* de synchronisation et de coordination des systèmes distribués)

HBase : liens utiles

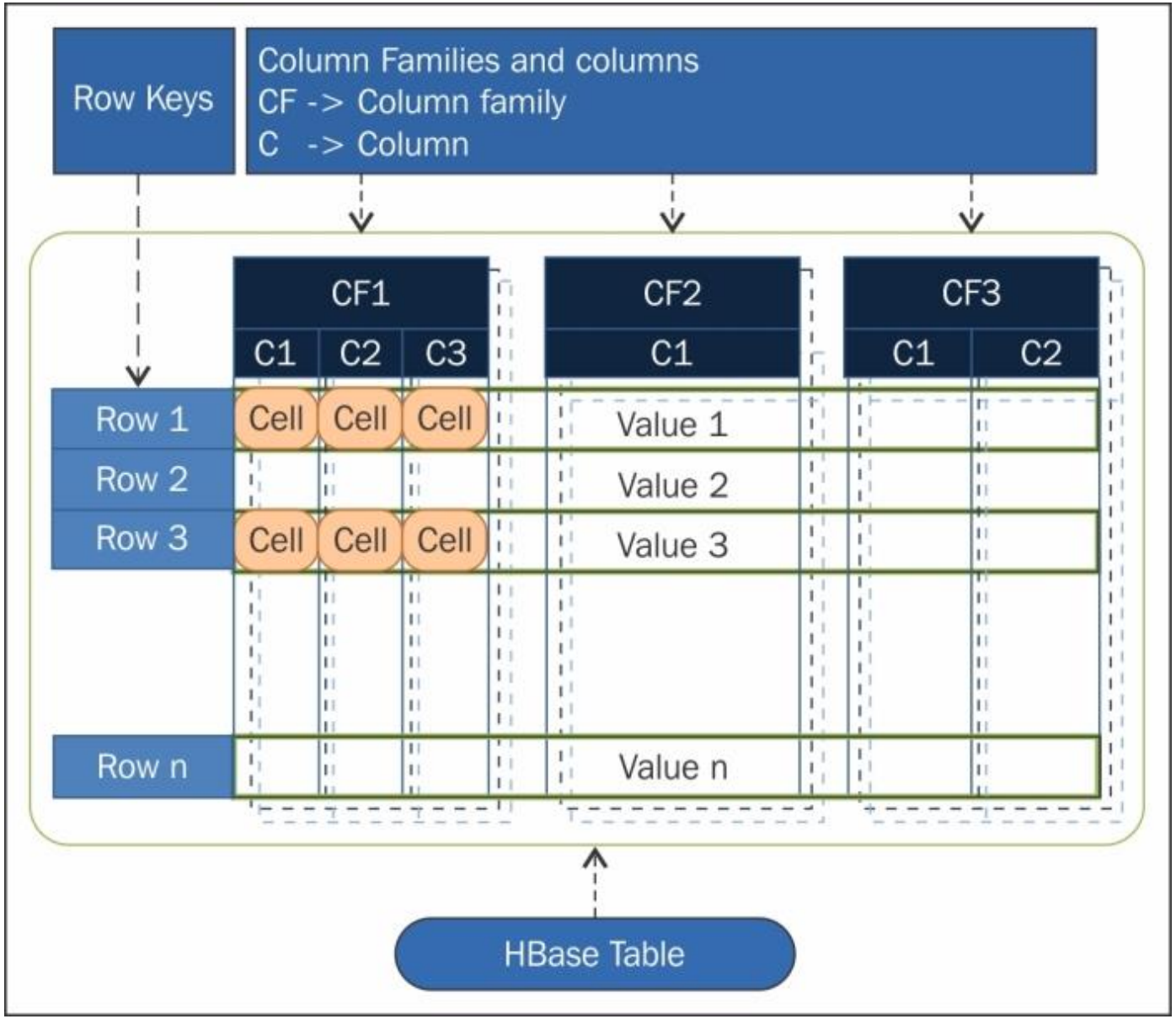
- Site officiel : https://hbase.apache.org/
- Tutoriels et cours en ligne :
  - https://juvenal-chokogoue.developpez.com/tutoriels/apprendre-travailler-hbase
  - https://www.tutorialspoint.com/hbase/
  - https://www.linkedin.com/learning/l-essentiel-de-nosql/decouvrir-hbase?autoplay=true&u=74606242

Hbase : Architecture

Stockage de chaque famille de colonnes dans un HFile



Hbase : Modèle de données



Hbase : Exemple de table

	Personnel		Contact	
Row Key	**Prénom**	**Age**	**téléphone**	**ville**
00001	Juvénal ***timestamp***: 10/09/2011 13:05:17:09	22 ***timestamp***: 10/09/2011 13:05:17:09	06 90 98 76 52 ***timestamp***: 10/09/2011 13:05:17:09	Douala ***timestamp***: 10/09/2011 13:05:17:09
		25 ***timestamp***: 20/09/2013 15:00:05:09		Lille ***timestamp***: 20/09/2013 15:00:05:09
				Paris ***timestamp***: 30/10/2016 16:18:50:10
00002	Paul ***timestamp***: 10/09/2011 13:10:05:09	30 ***timestamp***: 10/09/2011 13:10:05:09	07 90 94 86 52 ***timestamp***: 10/09/2011 13:10:05:09	Nancy ***timestamp***: 10/09/2011 13:10:05:09
00003	Jean ***timestamp***: 12/09/2011 11:30:20:09	34 ***timestamp***: 12/09/2011 11:30:20:09	06 74 98 76 25 ***timestamp***: 12/09/2011 11:30:20:09	Marseille ***timestamp***: 12/09/2011 11:30:20:09

- 2 familles : Personnel (de colonnes Prénom et Age) et Contact (de colonnes Téléphone et Ville)
- Pour la clé 0001 : 3 versions (suite au changement d'âge et de ville du contact)

Hbase : Modèle de données

Stockage des données sous forme de *HFiles*, par colonnes, dans HDFS.

une *column family* particulière par *HFile*

- **Table**
  - Contains column-families
- **Column family**
  - Logical and physical grouping of columns
- **Column**
  - Exists only when inserted
  - Can have multiple versions
  - Each row can have different set of columns
  - Each column identified by it's key
- **Row key**
  - Implicit primary key
  - Used for storing ordered rows
  - Efficient queries using row key

HBTABLE

Row key	Value
11111	cf_data: {'cq_name' : 'name1' , 'cq_val' : 1111} cf_info: {'cq_desc' : 'desc11111'}
22222	cf_data: {'cq_name' : 'name2' , 'cq_val' : 2013 @ ts = 2013, 'cq_val' : 2012 @ ts = 2012 }

HFile

```
11111 cf_data cq_name name1  @ ts1
11111 cf_data cq_val  1111   @ ts1
22222 cf_data cq_name name2  @ ts1
22222 cf_data cq_val  2013   @ ts1
22222 cf_data cq_val  2012   @ ts2
```

HFile

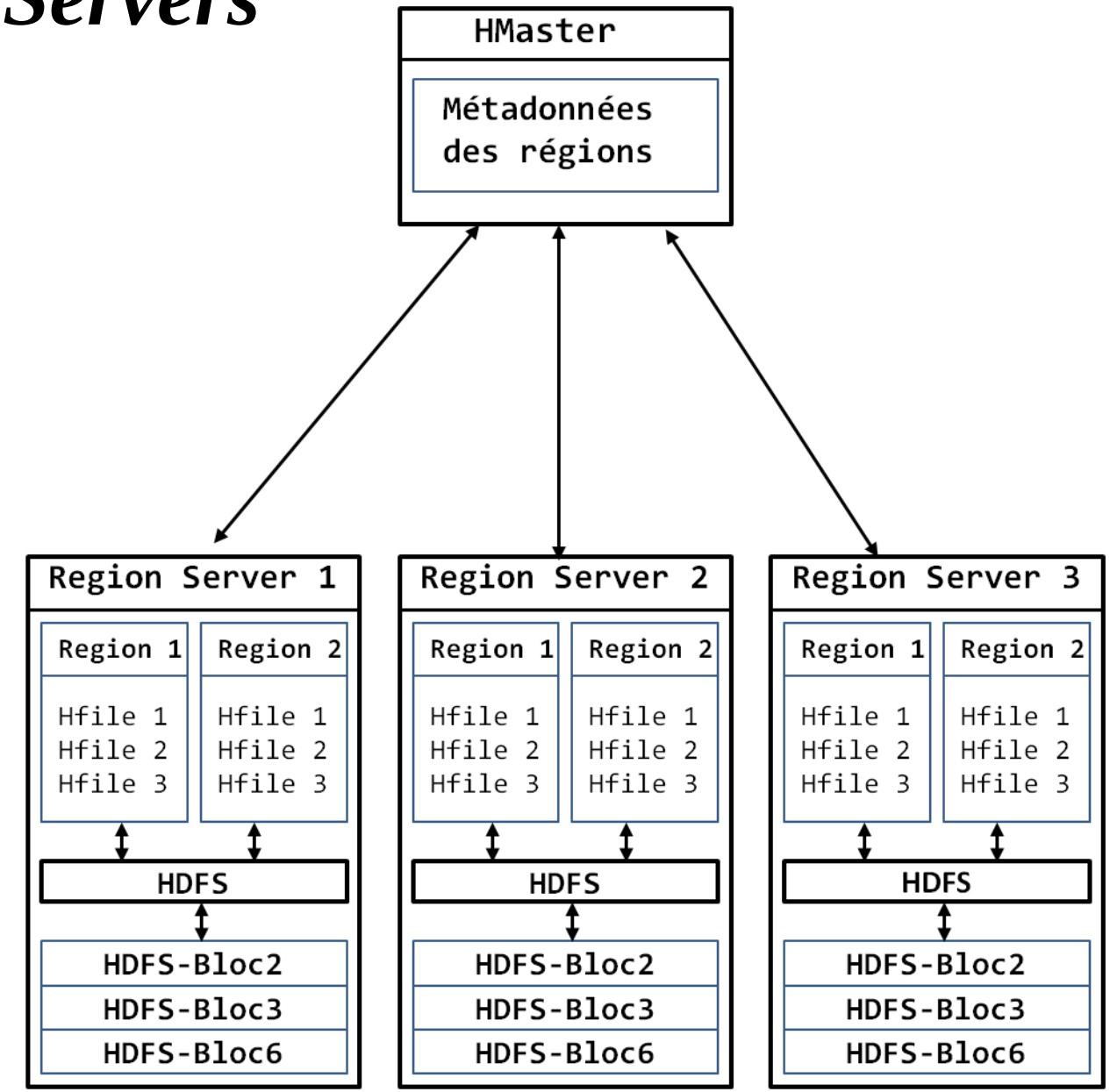
```
11111 cf_info cq_desc desc11111  @ ts1
```

Hbase : gestion des données

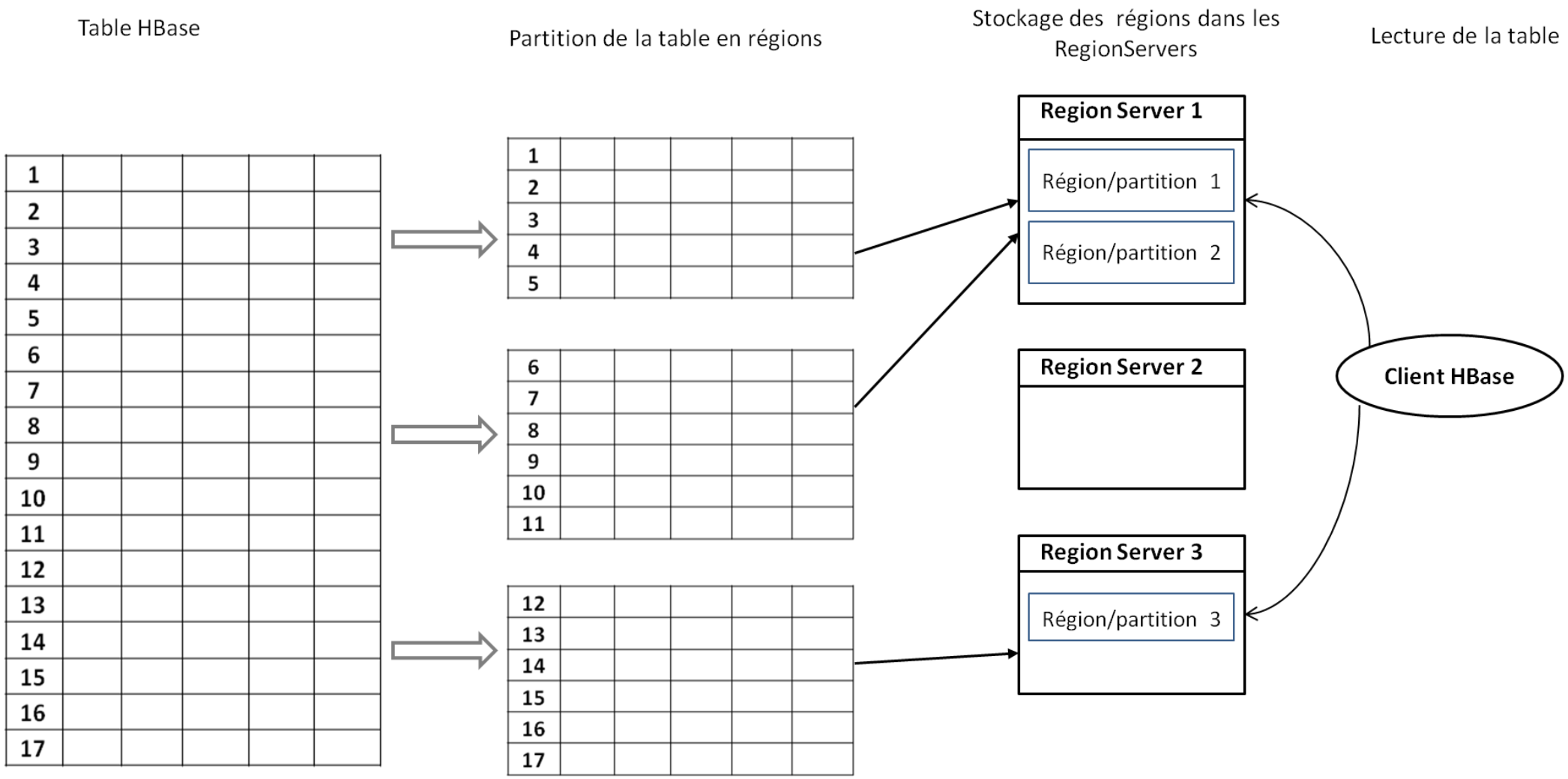
- Plusieurs étapes d'écriture :
  - Ecriture dans un *WAL* (*Write-Ahead Log*)
  - Placement des données dans un *buffer* nommé *memstore*
  - *Memstore* écrit dans un *HFile* sur le HDFS lorsque trop gros
- Suppression :
  - Gérées à l'aide d'un marqueur *tombstone*
  - Réelle au moment d'un compactage

Hbase : *HMaster* et *RegionServers*

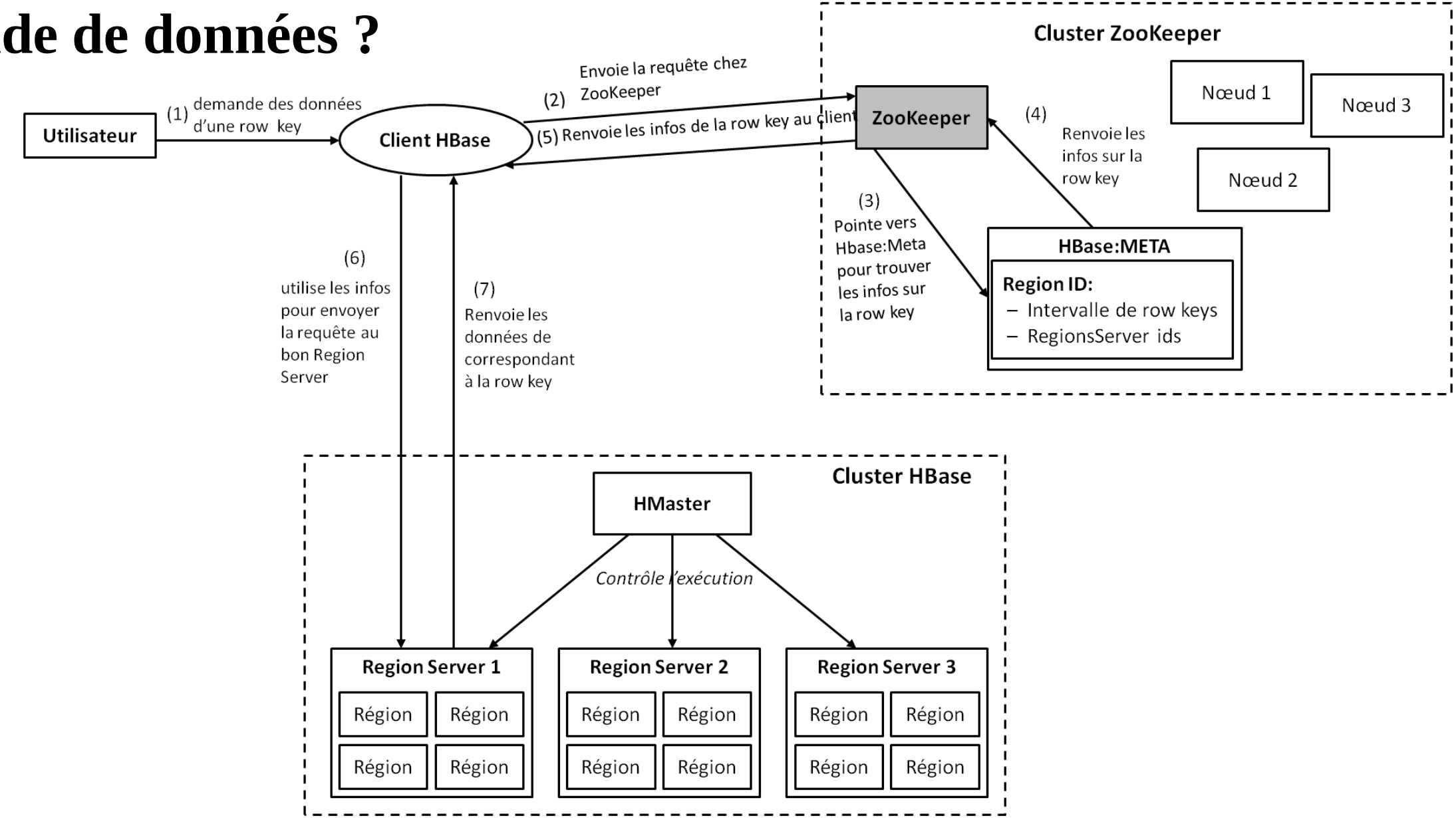
- *HMaster* : nœud maître
- *RegionsServers* **:** nœuds esclaves
- Stockage des données distribué sur les *RegionsServers* gérés par le *Hmaster*
- Tables *HBase* rendues persistantes sur le disque sous forme de fichiers HDFS appelés *HFiles*



Hbase : partition d'une table



Hbase : que se passe-t-il lors d'une demande de données ?



Hbase : Les bonnes questions à se poser pour la modélisation

- Quelle est la structure des valeurs de la *row key* ?
- Combien de familles de colonnes la table doit-elle avoir ?
- Quelles données vont dans quelle famille de colonnes ?
- Combien de colonnes y'a t-il dans chaque famille ?
- Quel est le label ou titre de chaque colonne ?
- Quelles données seront stockées dans les cellules ?
- Combien de versions de chaque cellule *HBase* devra-t-elle historiser ?

Hbase : "*Best practices*" pour la modélisation

- Regrouper les colonnes utilisées selon le même schéma dans la même famille de colonnes
- Limiter autant que possible à 10 ou 15 le nombre de familles de colonnes par table
- Donner le nom de toutes les colonnes à la création de la table même si les colonnes elles-mêmes n'existent pas encore
- Dénormaliser au maximum les tables modélisées

Hbase : opérations CRUD (création d'une table)

Création d'une nouvelle table : commande `create`

```
hbase(main):002:0> create 'students', {NAME => 'infos', VERSIONS
=> 1}, {NAME => 'registrations', VERSIONS => 2}
0 row(s) in 1.2230 seconds

=> Hbase::Table - students

hbase(main):003:0> list
TABLE
students
1 row(s) in 0.0630 seconds

=> ["students"]
```

Hbase : opérations CRUD (*get* et *put*)

- `get` : équivalent à `SELECT` en SQL
- `put` : équivalent à `INSERT` en SQL

```
hbase(main):004:0> put 'students', '16139', 'infos:firstname', '
Alexis'
0 row(s) in 0.1350 seconds

hbase(main):005:0> put 'students', '16139', 'infos:age', '22'
0 row(s) in 0.0120 seconds

hbase(main):006:0> put 'students', '16139', 'registrations:class
', '3BE'
0 row(s) in 0.0110 seconds

hbase(main):007:0> get 'students', '16139'
COLUMN                          CELL
 infos:age                      timestamp=1477172359150, value=22
 infos:firstname                timestamp=1477172339414, value=Alexis
 registrations:class            timestamp=1477172463762, value=3BE
3 row(s) in 0.0750 seconds
```

Hbase : exemples de *get* et *put*

```
hbase(main):008:0> put 'students', '16139', 'registrations:note',
 'Pasterelle 4M'
0 row(s) in 0.0030 seconds

hbase(main):009:0> put 'students', '16139', 'registrations:note',
 'Passerelle 4M'
0 row(s) in 0.0030 seconds

hbase(main):010:0> get 'students', '16139', {COLUMN => '
registrations:note', VERSIONS => 2}
COLUMN                          CELL
 registrations:note             timestamp=1477173105470, value=
 Passerelle 4M
 registrations:note             timestamp=1477173102196, value=
 Pasterelle 4M
2 row(s) in 0.0110 seconds
```

Hbase : opérations CRUD (*scan* et *delete*)

- `scan` : lecture séquentielle
- `delete` : équivalent à `DELETE` en SQL mais création par HBase de marqueurs sur chaque valeur à supprimer et suppression de celles-ci massivement lorsqu'HBase effectue un compactage de la table
- Possibilité d'utiliser un langage d'abstraction comme HiveQL ou Pig Latin, pour créer et manipuler les tables HBase

HBase : avantages et inconvénients


Tolérance aux pannes


Gestion de gros volumes de données


Ecosystème Hadoop


Pas de support pour les jointures (à faire au niveau applicatif)


***Single-row* ACID**


Ecosystème Hadoop

HBase vs relationnel

HBase vs. RDBMS

	HBase	RDBMS
Hardware architecture	Similar to Hadoop. Clustered commodity hardware. Very affordable.	Typically large scalable multiprocessor systems. Very expensive.
Fault Tolerance	Built into the architecture. Lots of nodes means each is relatively insignificant. No need to worry about individual node downtime.	Requires configuration of the HW and the RDBMS with the appropriate high availability options.
Typical Database Size	Terabytes to Petabytes - hundred of millions to billions of rows.	Gigabytes to Terabytes – hundred of thousands to millions of rows.
Data Layout	A sparse, distributed, persistent, multidimensional sorted map.	Rows or column oriented.
Data Types	Bytes only.	Rich data type support.
Transactions	ACID support on a single row only	Full ACID compliance across rows and tables
Query Language	API primitive commands only, unless combined with Hive or other technology	SQL
Indexes	Row-Key only unless combined with other technologies such as Hive or IBM's BigSQL	Yes
Throughput	Millions of queries per second	Thousands of queries per second

Cassandra

- Conçu à l'origine par Facebook
- Actuellement projet de la fondation Apache (2008)
- Distribué par la société *Datastax*
- Initialement basé sur le système *BigTable* de Google (stockage orienté colonnes)
- Maintenant basé sur le système *DynamoDB* (hachage) ⇒ stockage orienté "clé/valeur"
- Un des rares systèmes NoSQL à proposer un typage fort
- Ecrit en Java

Cassandra : modèle relationnel étendu

- Evolution vers un modèle relationnel étendu
- Langage d'interrogation : *Cassandra Query Language* (CQL)
- Pas d'architecture maître/esclave – Architecture pair à pair
- Système *scalable* et distribué :
  - Méthodes de passage à l'échelle inspirées du système Dynamo (Amazon)
  - Distribution par hachage (*consistent hashing*)
  - Tolérance aux pannes par réplication en mode multi-nœuds
  - Cohérence réglable (*tunable consistency*)

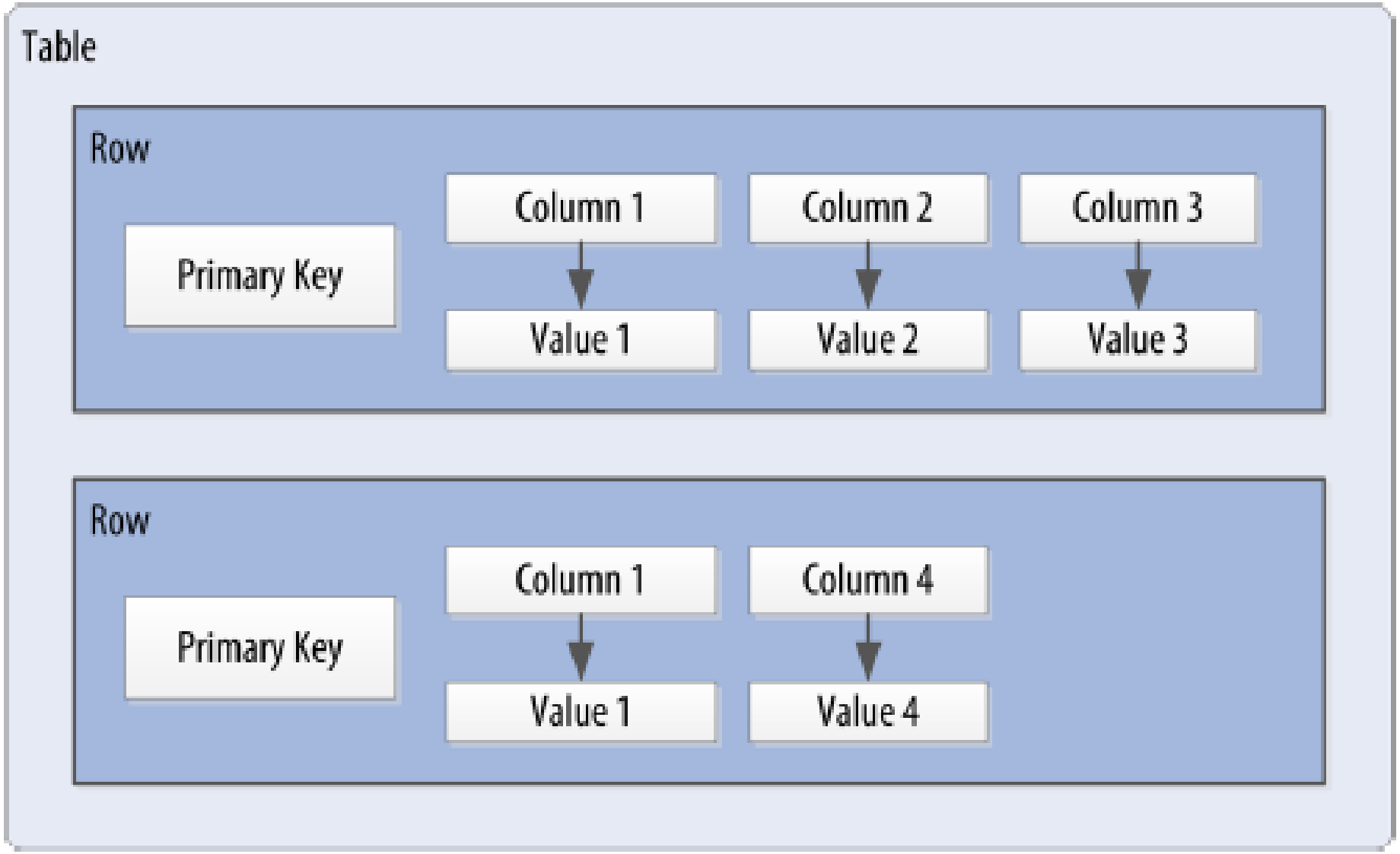
Cassandra : liens utiles

- Site officiel : http://cassandra.apache.org/
- Documentation en ligne : https://cassandra.apache.org/doc/latest/
- Version entreprise : *Datasax AstraDB* (multi-cloud DBaaS based on Apache Cassandra) https://www.datastax.com/products/datastax-astra
- Cours en ligne :
  - http://www-igm.univ-mlv.fr/~dr/XPOSE2010/Cassandra/cassandra.html
  - http://chewbii.com/cassandra-slides/
  - http://b3d.bdpedia.fr/cassandra_tp.html
  - https://stph.scenari-community.org/contribs/nos/Cassandra1/co/cours.html

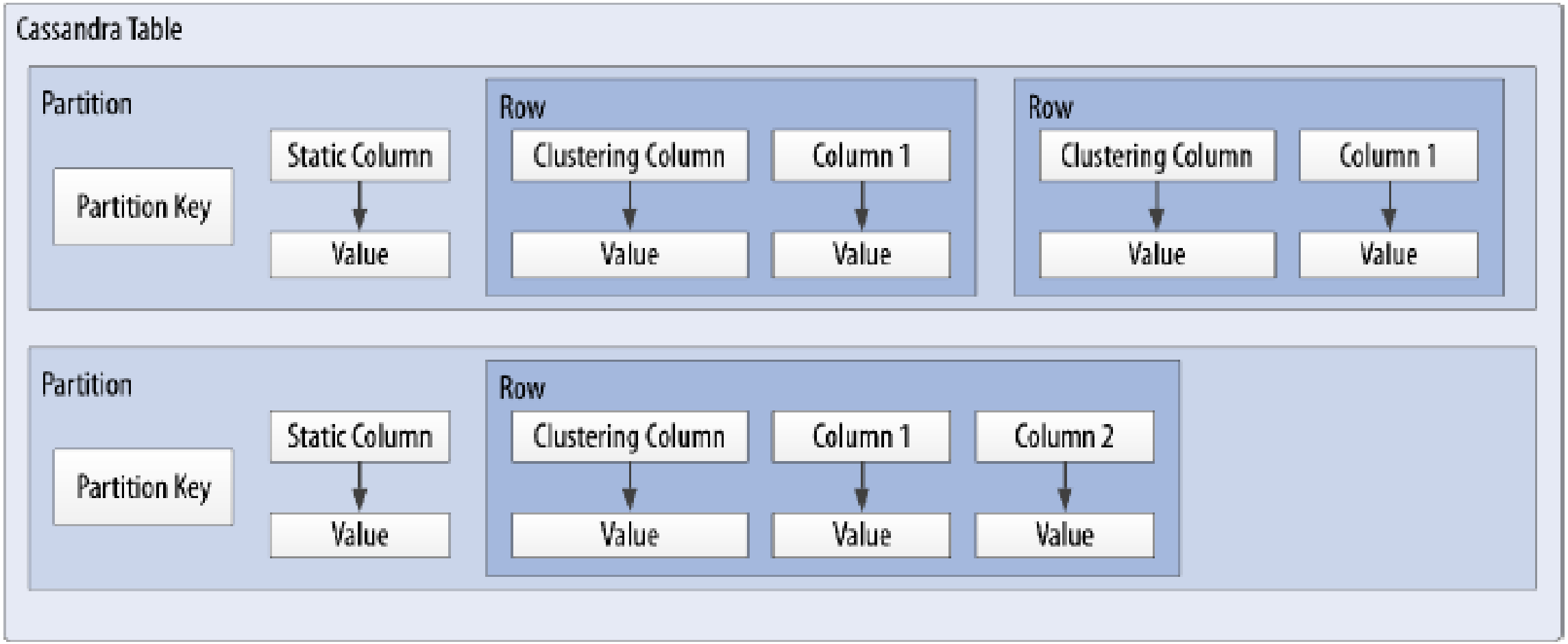
Cassandra : modèle de données

- Bases de données : *Keyspace*
- Tables : *Table* ou *Column Family* (lignes organisées en partition)
- Partition : regroupement de lignes stockées sur le même nœud
- Lignes : *Row* (valeurs simples ou complexes)
- Relationnel étendu en rompant la première règle de normalisation (type atomique).
- Ligne (*row*) : document structuré (imbrication) avec identifiant (*row key*) associé à un ensemble de paires (clé, valeur).
- Lignes typées par un schéma, y compris les données imbriquées
- Pas d'insertion de données ne respectant pas le schéma

Cassandra : nombre de colonnes flexible



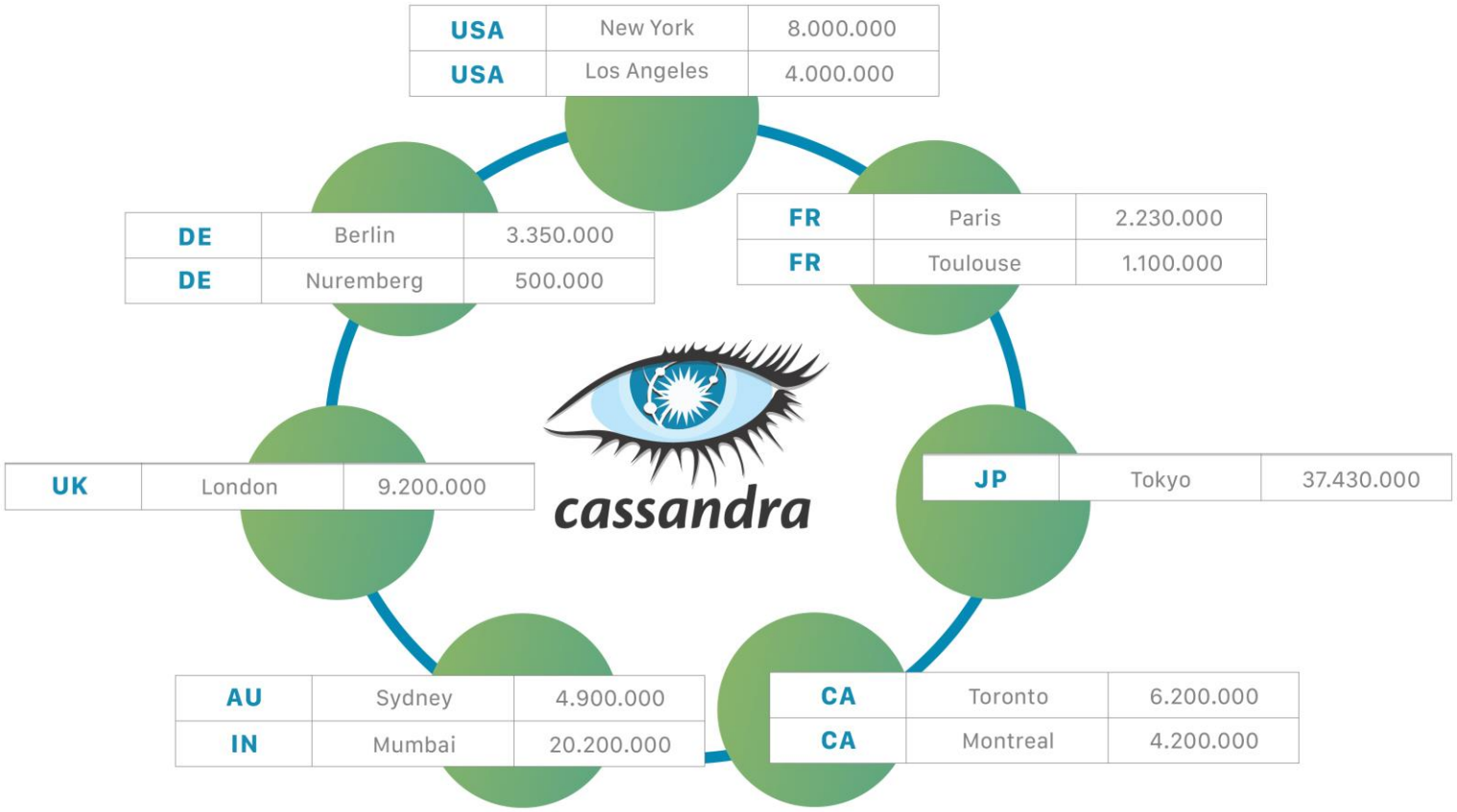
Cassandra : clé de partition



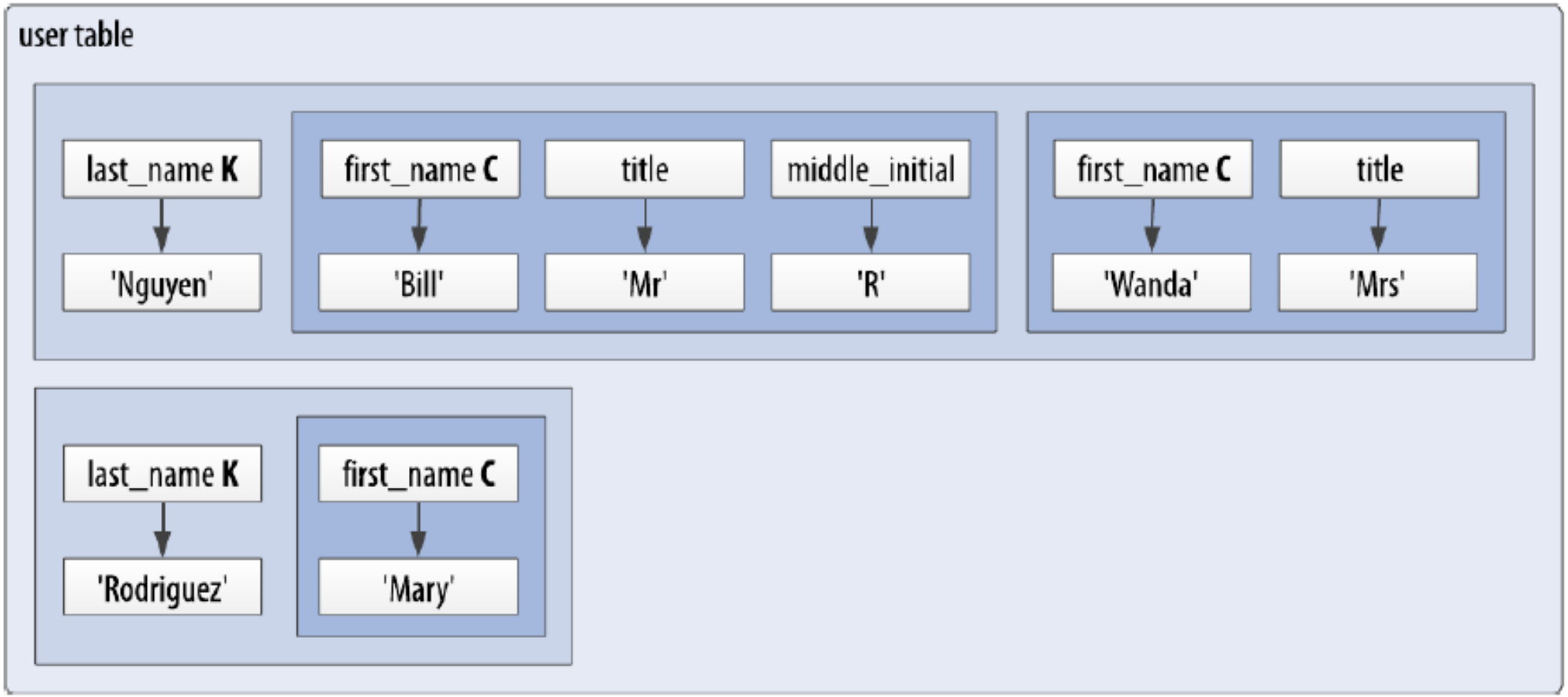
Cassandra : répartition des données selon la clé de partition

COUNTRY	CITY	POPULATION
USA	New York	8.000.000
USA	Los Angeles	4.000.000
FR	Paris	2.230.000
DE	Berlin	3.350.000
UK	London	9.200.000
AU	Sydney	4.900.000
DE	Nuremberg	500.000
CA	Toronto	6.200.000
CA	Montreal	4.200.000
FR	Toulouse	1.100.000
JP	Tokyo	37.430.000
IN	Mumbai	20.200.000

Partition Key



Cassandra : exemple de table



Cassandra : pas de stockage des valeurs `null`

Name	Mobile Number	Address
Alice	01-789-456-123	101 Main Street

Name	Mobile Number	Address
Bob	<empty> *Affiché `null` sous `cqlsh`*	98 7th Avenue

Ce qui est réellement stocké :

```
| name: Alice | mobile_number: 01-789-456-123 | address: 101 Main Street |
```

```
| name: Bob | address: 98 7th Avenue |
```

Cassandra : principes

- Pas de jointure : duplication
- Dénormalisation et possibilité de gérer des vues matérialisées au dessus d'une même table
- Pas d'intégrité référentielle (pas de clé étrangère)
- Modélisation orientée requêtes (*Query-first design*)
- Prise en compte du stockage : stockage de chaque table dans un fichier séparé – nécessité de conserver les colonnes allant ensemble au sein d'une même table et de minimiser le nombre de partitions
- Pas de requête de tri – gestion du tri par l'ordre ascendant ou descendant des clés de regroupement (*clustering columns*)

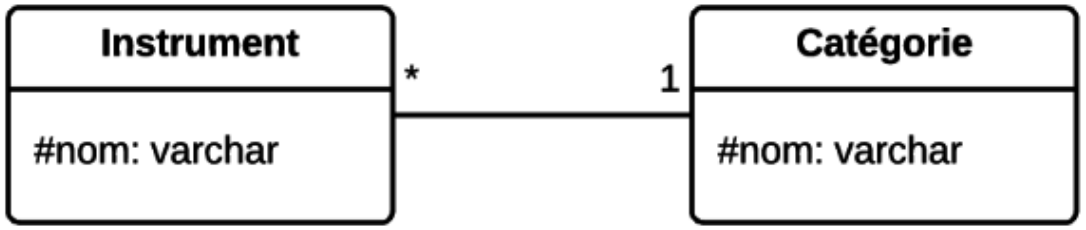
Cassandra : dénormalisation

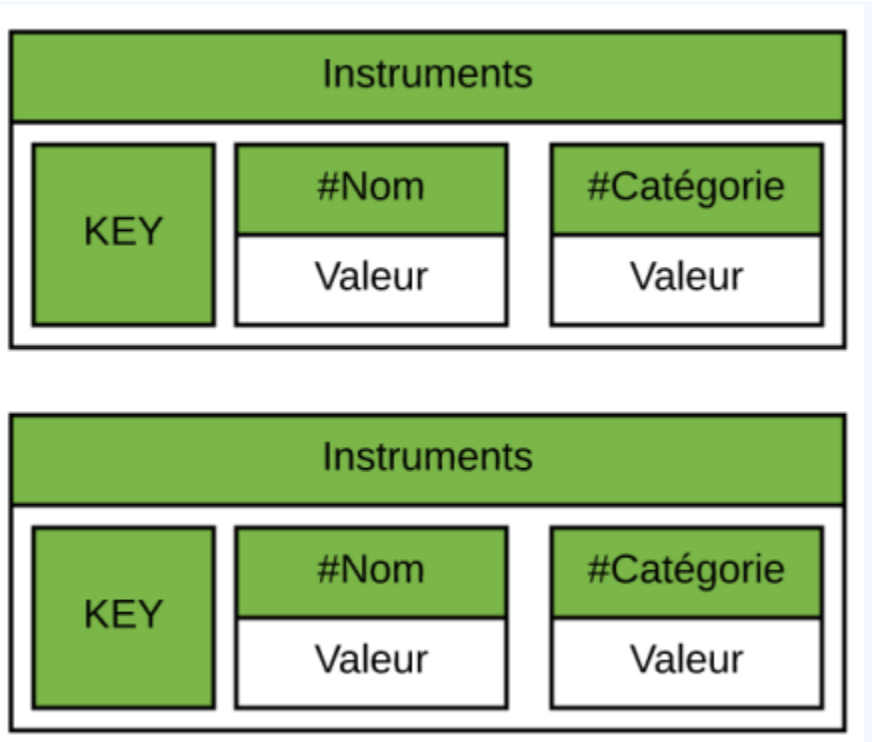
- Jointures inexistante ⇒ dénormalisation du modèle
- Regrouper des données le plus possible dans des lignes
- Conception orientée sur les requêtes les plus fréquentes de l'application
- Dimension la plus grande souvent favorisée (meilleure distribution)

⇒ redondance et possibilités d'incohérence

Cassandra : exemple de dénormalisation

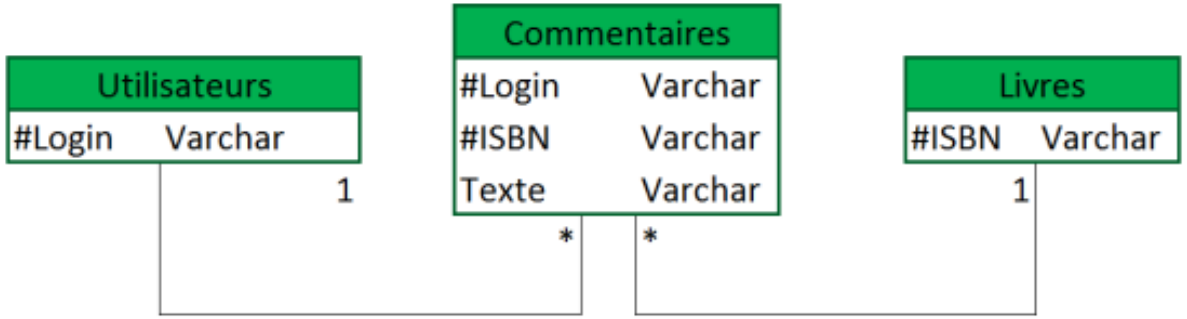
Jointures inexistante ⇒ dénormalisation du modèle

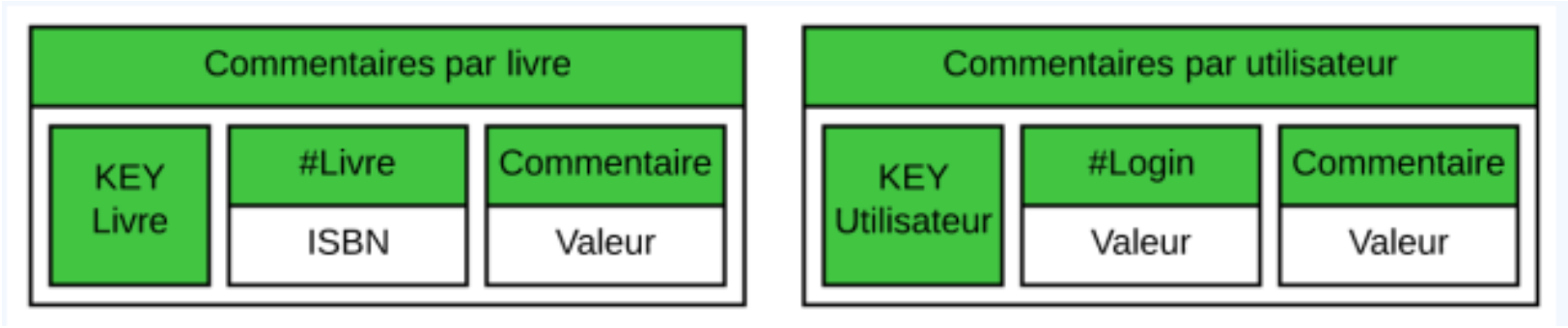




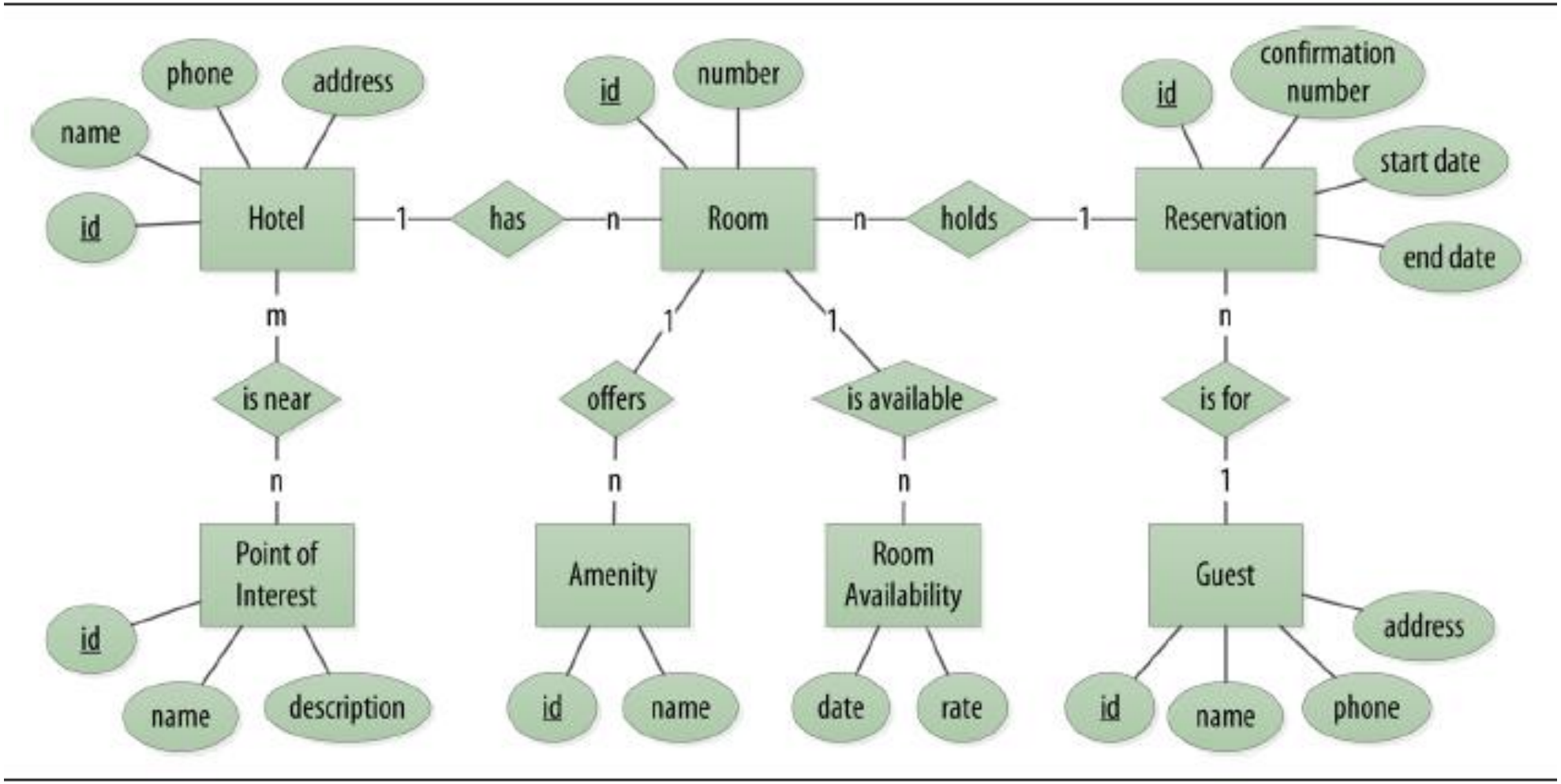
Cassandra : une table par requête

Jointures inexistante ⇒ dénormalisation du modèle



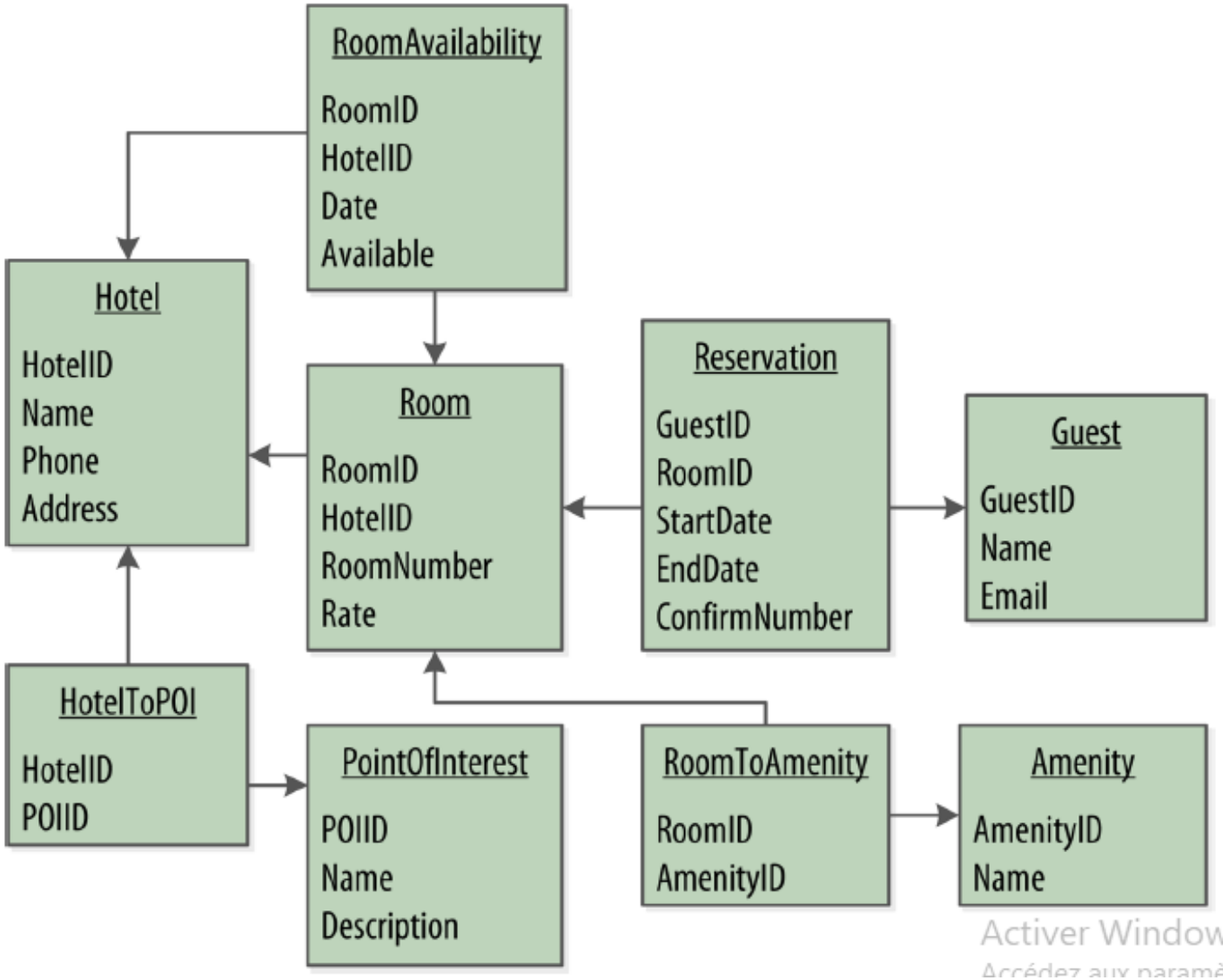


Cassandra : exemple de la documentation

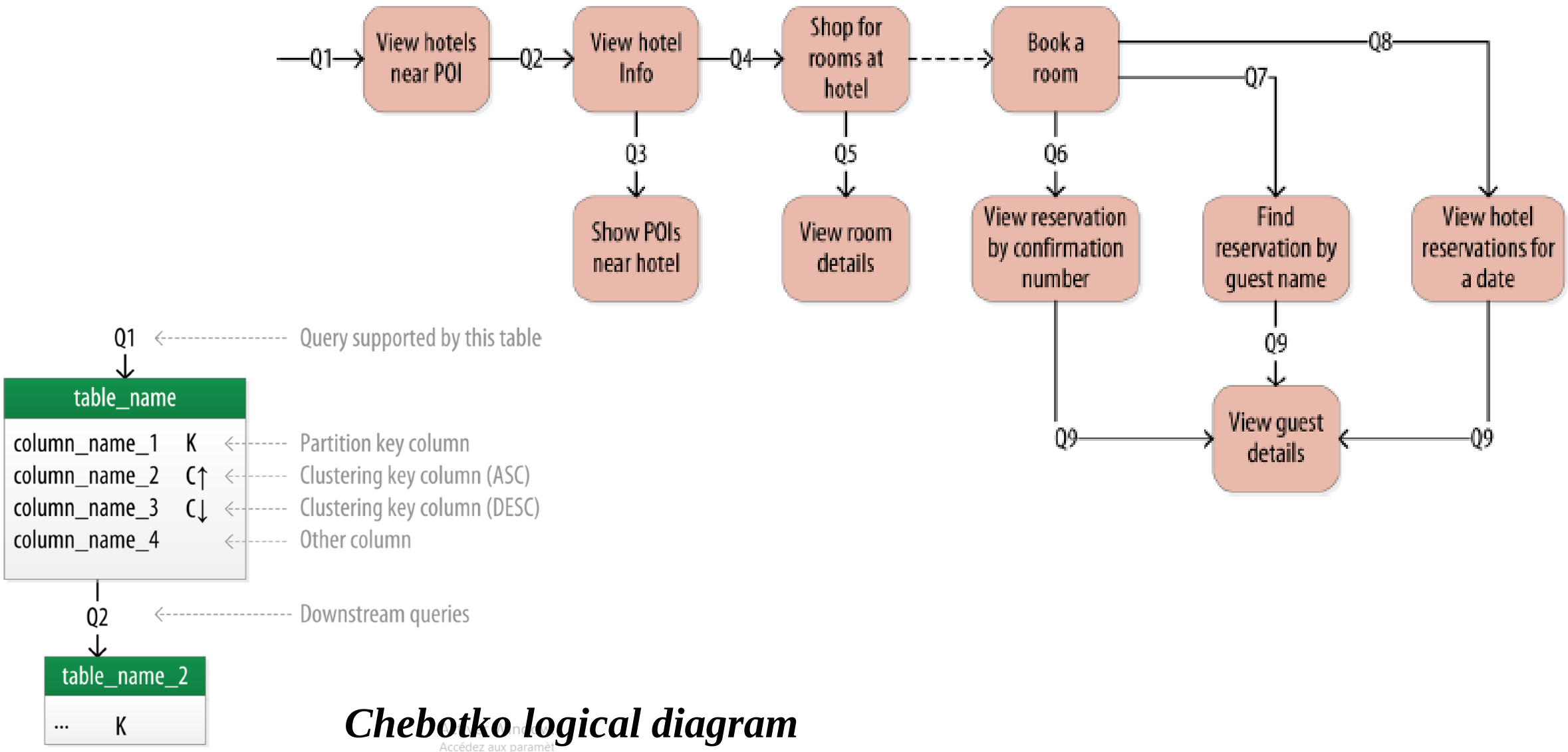


Cassandra : modèle relationnel de l'exemple de la documentation

Modèle relationnel orienté donnée

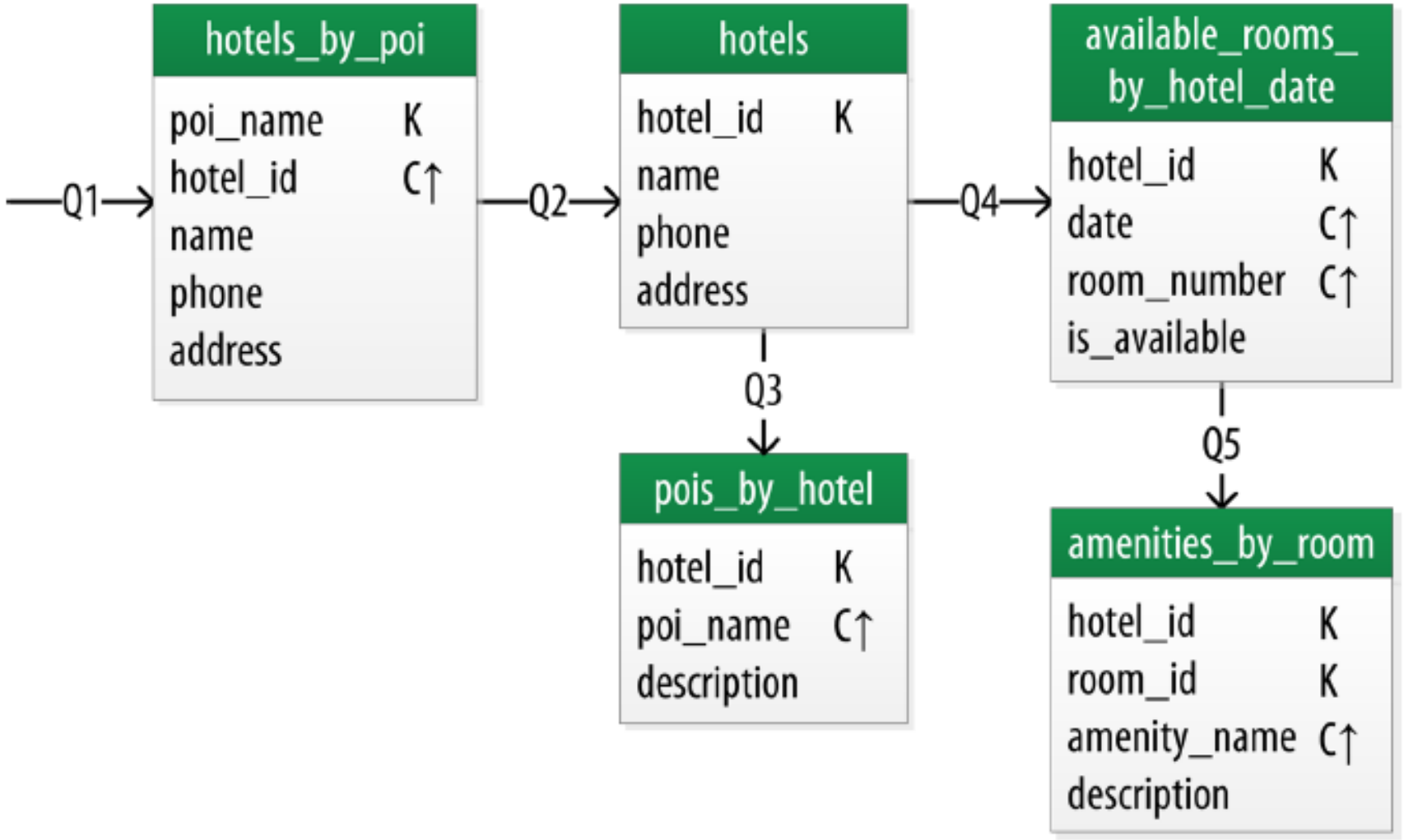


Cassandra : Modèle orienté par les requêtes (1/3)

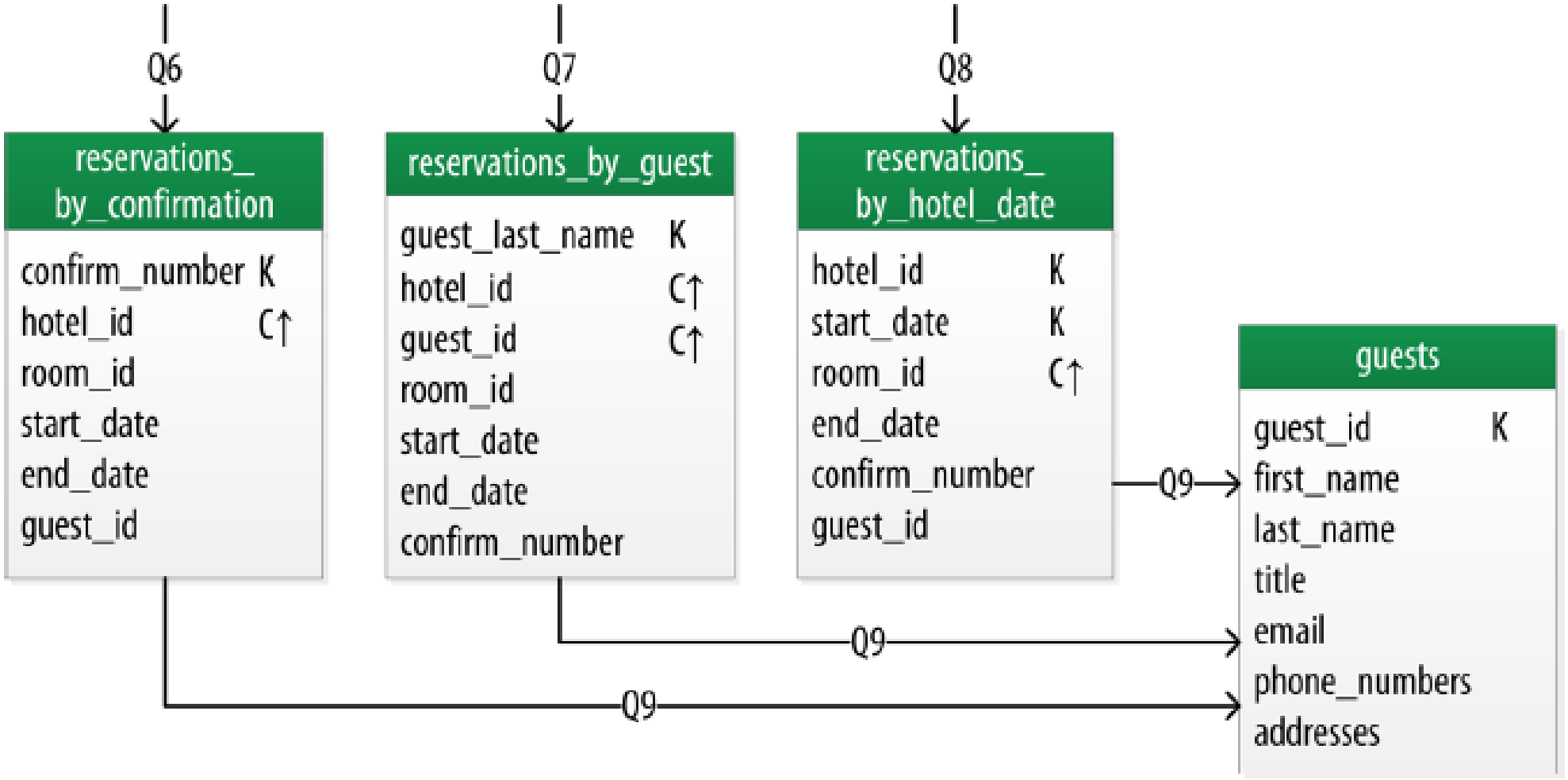


Chebotko logical diagram

Cassandra : Modèle orienté par les requêtes (2/3)



Cassandra : Modèle orienté par les requêtes (3/3)



Cassandra : langage CQL (création)

Création d'un **keyspace** :

```
CREATE KEYSPACE IF NOT EXISTS Compagnie
    WITH REPLICATION={'class':'SimpleStrategy','replication_factor':3};
```

Création d'une **Column Family** / table (sans imbrication) :

```
CREATE TABLE Vols (
    idVol INT, dateDepart DATE, distance INT,  duree FLOAT,
    depart CHAR(3), arrivee CHAR(3), pilote INT,
    copilote INT, officier INT, ChefCabine1 INT, ChefCabine2 INT,
    primary key (idVol)
);

CREATE INDEX vol_pilote ON Vols (pilote);
```

Cassandra : langage CQL (insertion)

A la SQL

```
INSERT INTO Vols (idVol, dateDepart, distance,  duree, depart,
        arrivee, pilote, copilote, officier, ChefCabine1, ChefCabine2)
    VALUES (1, '2016-10-15', 344, 1.3, 'CDG', 'LCY', 1, 2, 3, 4, 5);
```

Avec JSon

```
INSERT INTO Vols JSON '{
    "idVol" : 1, "dateDepart" : "2016-10-15", "distance":344,
    "duree" : 1.3, "depart" : "CDG", "arrivee" : "LCY",
    "pilote" : 1, "copilote" : 2, "officier" : 3,
    "ChefCabine1" : 4, "ChefCabine2" : 5}';
```

Cassandra : langage CQL (sélection)

```
SELECT * FROM Vols WHERE idVol = 1;
```

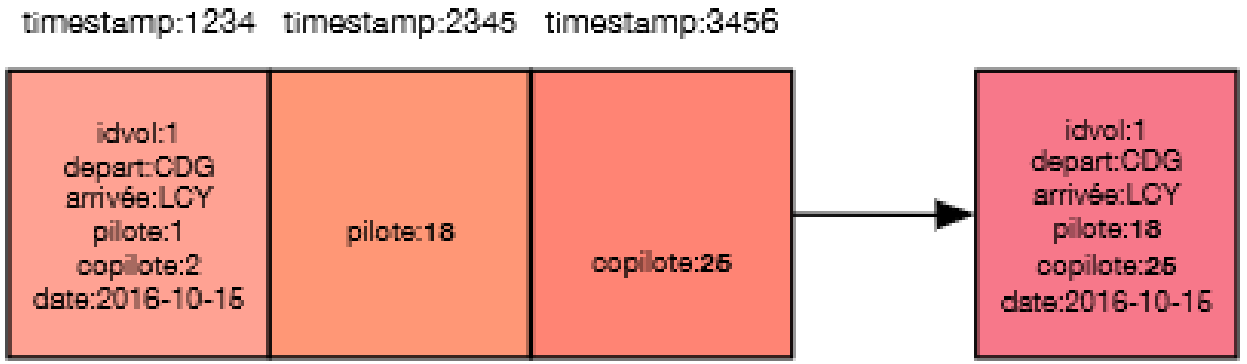
Résultat :

```
idVol | dateDepart  | distance |  duree | depart | arrivee | pilote | copilote | officier
-------------------------------------------------------------------------------------------
    1 |   2016-10-15|      344 |    1.3 |    CDG |     LCY |      1 |        2 |
```

Cassandra : langage CQL (utilisation des estampilles)

- Toute valeur est associée à un *TIMESTAMP*
- Estampillage automatique (ms) lors de la mise à jour
- Possible de spécifier l'estampille dans les requêtes

```
UPDATE Vols USING TIMESTAMP 2345
    SET pilote=18 WHERE idVol = 1;
```



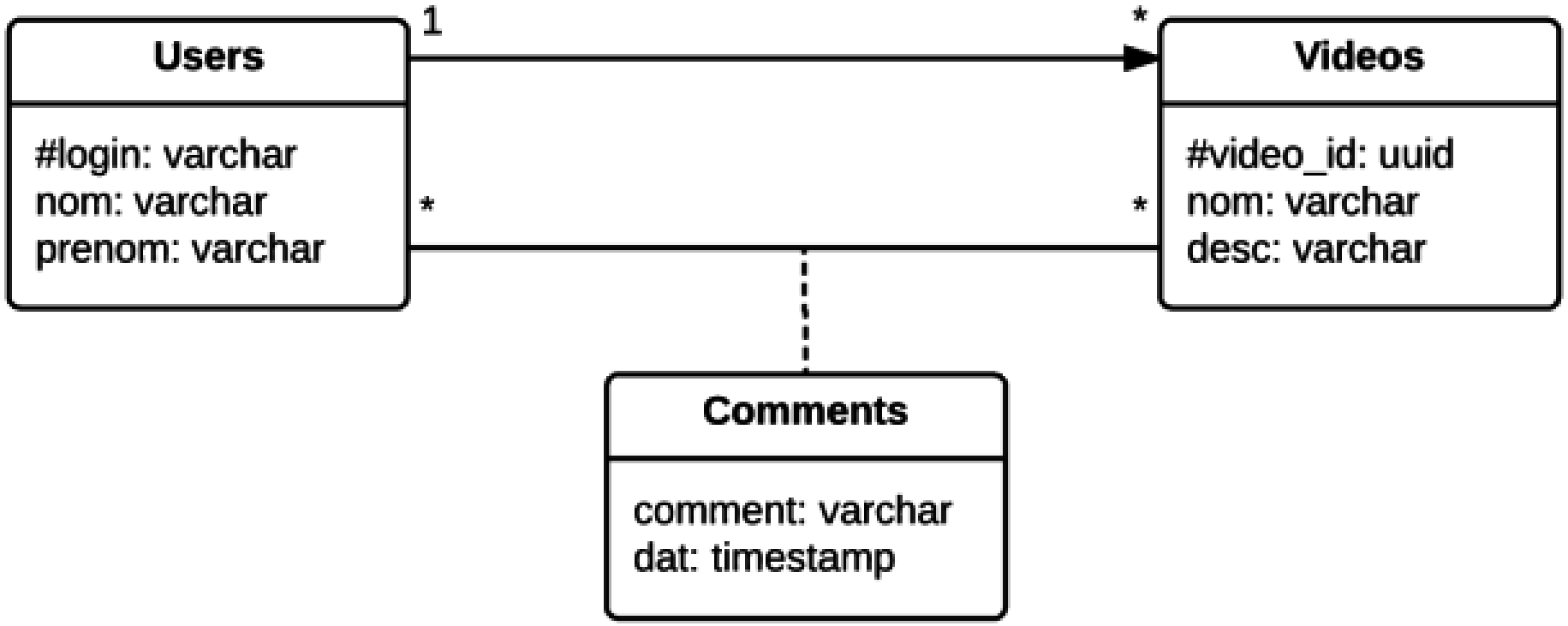
Cassandra : exemples de requêtes CQL

```
SELECT writetime(depart), writetime(pilote), writetime(copilote)
FROM Vols WHERE idVol=1;

writetime(depart) | writetime(pilote) | writetime(copilote)
------------------------------------------------------------
             1234 |              2345 |                3456
```

```
DELETE pilote USING TIMESTAMP 1234 FROM Vols WHERE idVol=1;
```

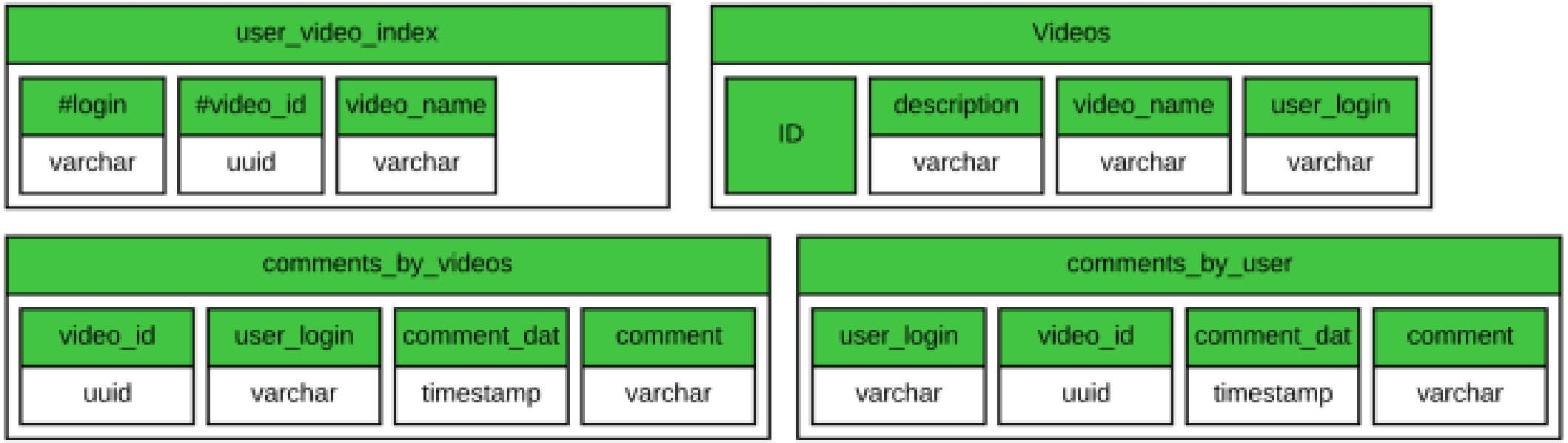
Cassandra : Exemple (1/5)



Cassandra : Exemple (2/5)

On ne souhaite pas faire des requêtes sur les utilisateurs mais on souhaite :

- Connaître les vidéos insérées par un utilisateur dont on connaît le login
- Accéder à toutes les informations d'une vidéo
- Connaître les commentaires par vidéo et par utilisateur



Cassandra : Exemple (3/5)

DATASTAX

One-to-Many

```
CREATE TABLE videos (
  videoid uuid,
  videoname varchar,
  username varchar,
  description varchar,
  tags varchar,
  upload_date timestamp,
  PRIMARY KEY(videoid)
);
```

- Static table to store videos
- UUID for unique video id
- Add username to denormalize

```
CREATE TABLE username_video_index (
   username varchar,
   videoid uuid,
   upload_date timestamp,
   video_name varchar,
   PRIMARY KEY (username, videoid)
);
```

```
SELECT video_name FROM username_video_index
WHERE username = 'tcodd' AND videoid =
'99051fe9'
```

- Lookup video by username

Write in two tables at once for fast lookups

Cassandra : Exemple (4/5)

- Model both sides of the view
- Insert both when comment is created

```
CREATE TABLE comments_by_video (
   videoid uuid,
   username varchar,
   comment_ts timestamp,
   comment varchar,
   PRIMARY KEY (videoid,username)
);
```

```
CREATE TABLE comments_by_user (
   username varchar,
   videoid uuid,
   comment_ts timestamp,
   comment varchar,
   PRIMARY KEY (username,videoid)
);
```

DON'T BE AFRAID OF WRITES

Cassandra : Exemple (5/5)

Keyword index example

DATASTAX

- Using the previous video example, users want to tag videos.
- Video table defined as:

```
CREATE TABLE videos (
  videoid uuid,
  videoname varchar,
  username varchar,
  description varchar,
  tags varchar,
  upload_date timestamp,
  PRIMARY KEY(videoid)
);
```

- Now we can define an index for tagging videos

```
CREATE TABLE video_tag_index (
  tag varchar,
  videoid uuid,
  timestamp timestamp
  PRIMARY KEY(tag, videoid)
);
```

Cassandra : Clé de partition et de regroupement

Table avec une clé de partition à 1 attribut

```
CREATE TABLE users (
  email TEXT,
  name TEXT,
  age INT,
  date_joined DATE,
  PRIMARY KEY ((email))
); ↩
```

Table avec une clé de partition à 2 attributs

```
CREATE TABLE movies (
  title TEXT,
  year INT,
  duration INT,
  avg_rating FLOAT,
  PRIMARY KEY ((title, year))
); ↩
```

Table avec une clé de partition à 1 attribut et une clé de regroupement à 2 attributs

```
CREATE TABLE ratings_by_user (
  email TEXT,
  title TEXT,
  year INT,
  rating INT,
  PRIMARY KEY ((email), title,
year)
); ↩
```

Table avec une clé de partition à 2 attributs et une clé de regroupement à 2 attributs

```
CREATE TABLE movies_by_actor (
  first_name TEXT,
  last_name TEXT,
  title TEXT,
  year INT,
  PRIMARY KEY ((first_name,
last_name), title, year)
); ↩
```

Cassandra : attributs complexes

Pour un attribut peut être associé à un type complexe :

- *SET* : ensemble valeurs (non ordonnées)
- *LIST* : liste de valeurs (ordonnées)
- *MAP* : *hashMap* clé/valeur
- *SubType* : ligne imbriquée

```
CREATE TABLE Vols (
    idVol INT, dateDepart DATE, distance INT,  duree FLOAT,
    depart CHAR(3), arrivee CHAR(3), pilote INT,
    copilote INT, officier INT, ChefCabine1 INT, ChefCabine2 INT,
    hotesses XXX,
    primary key (idVol)
);
```

Cassandra : attribut de type SET

- *Insertion* :

  INSERT INTO Vols (..., hotesses) VALUES (..., {6, 7, 8}) ;

- *Mise à jour* :

  UPDATE Vols SET hotesses = hotesses + {9} WHERE idVol = 1 ;
  UPDATE Vols SET hotesses = hotesses - {8} WHERE idVol = 1 ;
  UPDATE Vols SET hotesses = {10} WHERE idVol = 1 ;

  DELETE hotesses FROM Vols WHERE idvol = 1 ;

- *Liste des hotesses* :

  SELECT idVol, hotesses FROM Vols WHERE idVol = 1 ;

```
idVol |  hotesses
------------------
    1 | {6, 7, 8}
```

Cassandra : attribut de type LIST

- *Insertion* :

  INSERT INTO Vols (..., hotesses) VALUES (..., [6, 7, 8]) ;

- *Mise à jour* :

  UPDATE Vols SET hotesses = hotesses + [9] WHERE idVol = 1 ;
  UPDATE Vols SET hotesses[1] = 8 WHERE idVol = 1 ;
  UPDATE Vols SET hotesses = [10] WHERE idVol = 1 ;

  DELETE hotesses[0] FROM Vols WHERE idvol = 1 ;

- *Liste des hotesses* :

  SELECT idVol, hotesses FROM Vols WHERE idVol = 1 ;

```
idVol | hotesses
-------------------
    1 | [6, 7, 8]
```

Cassandra : attribut de type MAP

- *Insertion* :
  INSERT INTO Vols (..., hotesses) VALUES (..., {"h1" : 6, "h2" : 7, "h3" : 8}]);
- *Mise à jour* :
  UPDATE Vols SET hotesses = hotesses + {"h4" : 9} WHERE idVol = 1;
  UPDATE Vols SET hotesses["h1"] = 10 WHERE idVol = 1;
  UPDATE Vols SET hotesses = {"h1" : 9} WHERE idVol = 1;
  DELETE hotesses["h2"] FROM Vols WHERE idvol = 1;
- *Liste des hotesses* :
  SELECT idVol, hotesses FROM Vols WHERE idVol = 1;

```
idVol |                          hotesses
-------------------------------------------
    1 | {"h1" : 6, "h2" : 7, "h3" : 8}
```

Cassandra : attribut d'un type imbriqué

- Création du *type* à imbriquer :

  ```
  CREATE TYPE hotesseType (ID int, nom text, prenom text);
  ```

  Il faut "*figer*" le type : (frozen<hotesseType>).

- *Insertion* :

  ```
  INSERT INTO Vols (..., hotesses) VALUES (..., {"ID" : 6,
                                  "nom" : "Walthéry", "prenom" : "Natacha"});
  ```

- *Mise à jour* :

  ```
  UPDATE Vols SET hotesse["nom"] = "Walter" WHERE idVol = 1;
  ```

  ```
  DELETE hotesse FROM Vols WHERE idvol = 1;
  ```

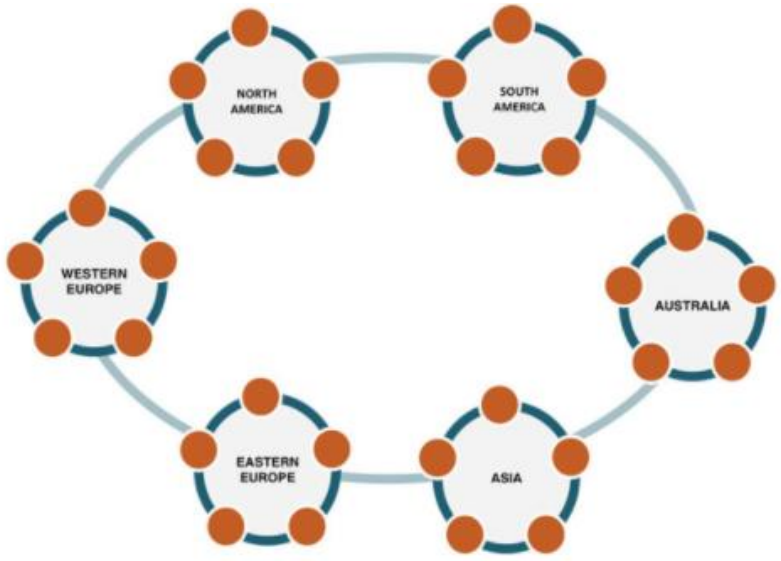
- *Liste des hotesses* :

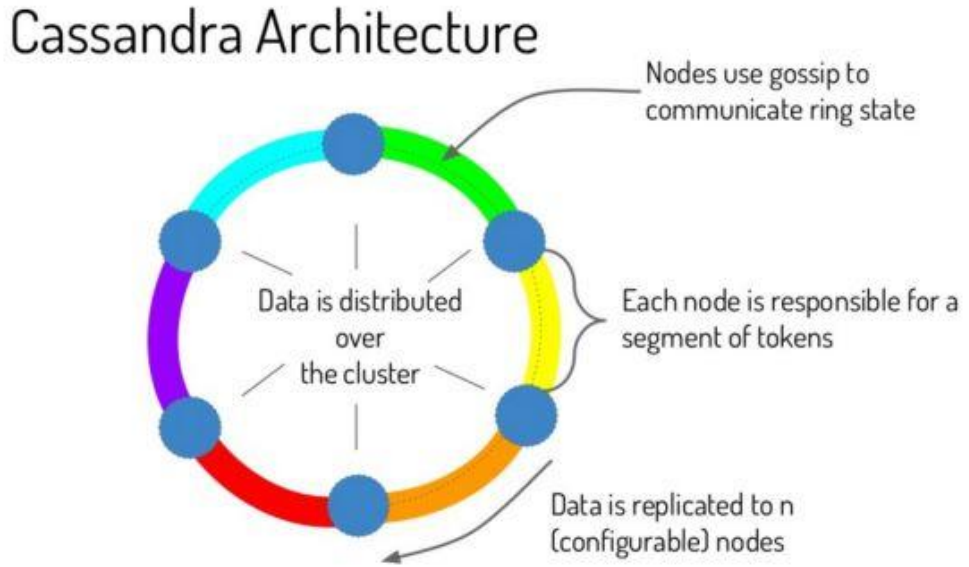
  ```
  SELECT idVol, hotesse.nom, hotesse.prenom FROM Vols WHERE
  idVol=1;
  ```

  ```
  idVol | hotesse.nom | hotesse.prenom
  ------------------------------------
      1 |    Walthéry |        Natacha
  ```

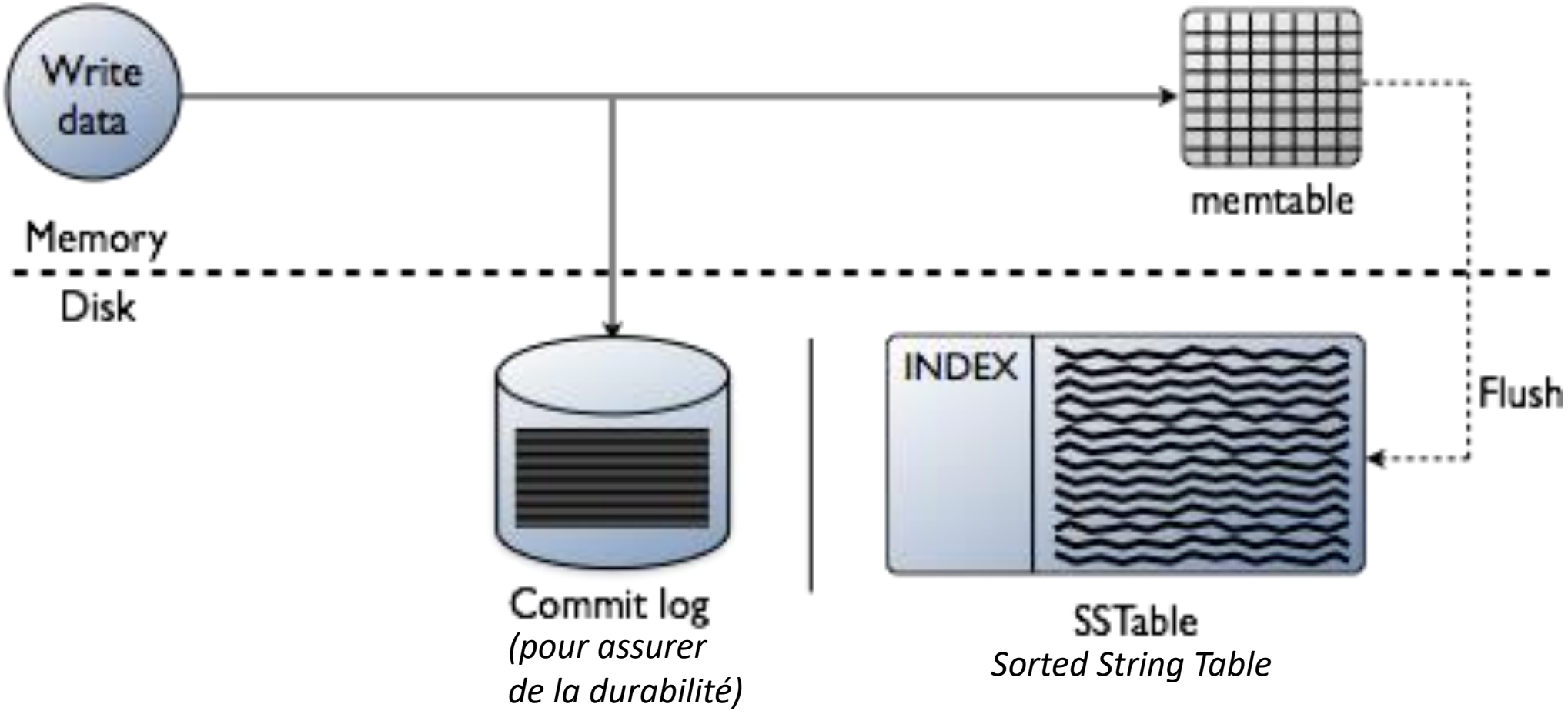
Cassandra : architecture

- Base de donnée contenue dans un cluster ou ensemble de *data-centers* (ensemble de nœuds qui sont dans un environnement géographique proche)
- Données réparties sur plusieurs nœuds et éventuellement répliquées sur 1 à N nœuds
- Possibilité pour un utilisateur de se connecter sur n'importe quel nœud et accéder à l'ensemble des données.

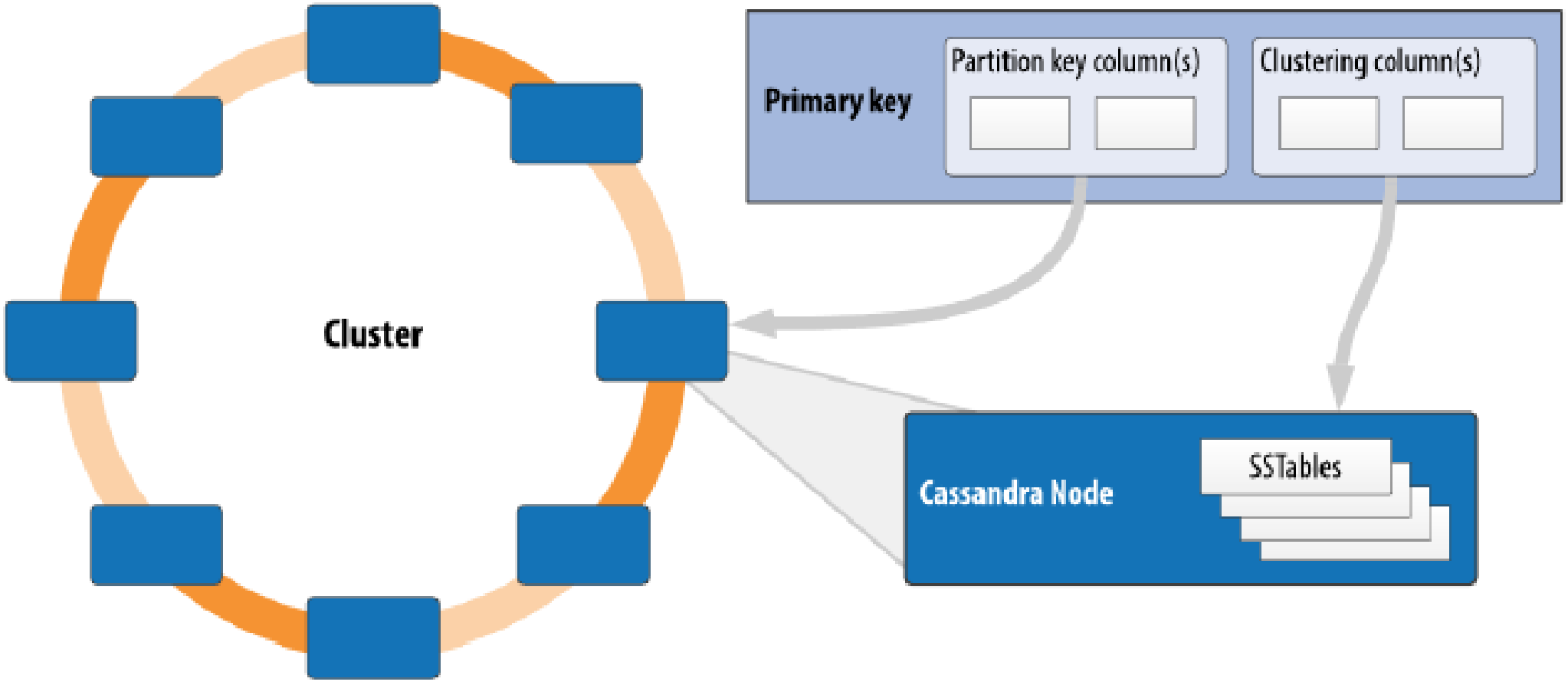




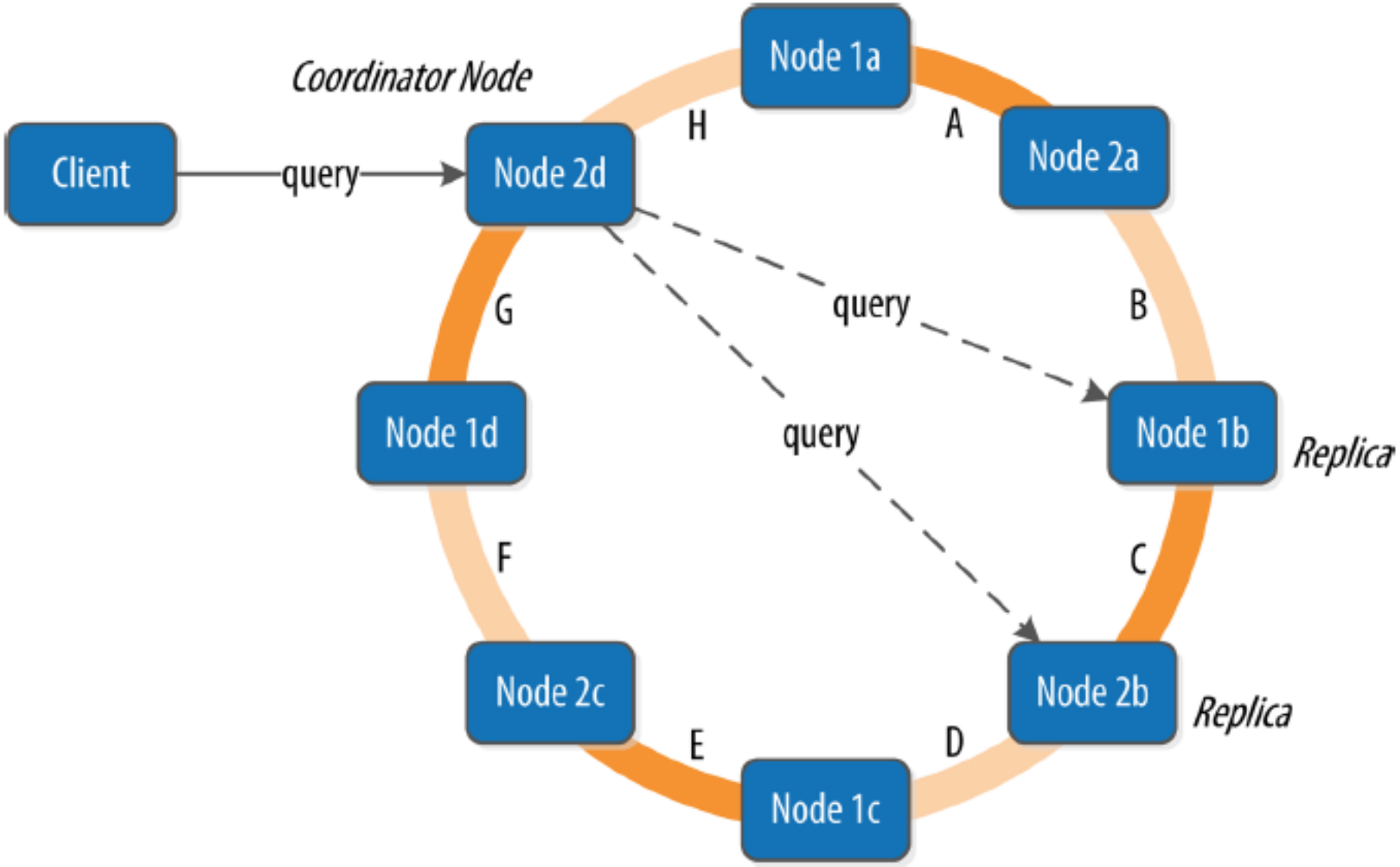
Cassandra : *memtable*, *commitlog* et *SSTable*



Cassandra : contenu d'un nœud



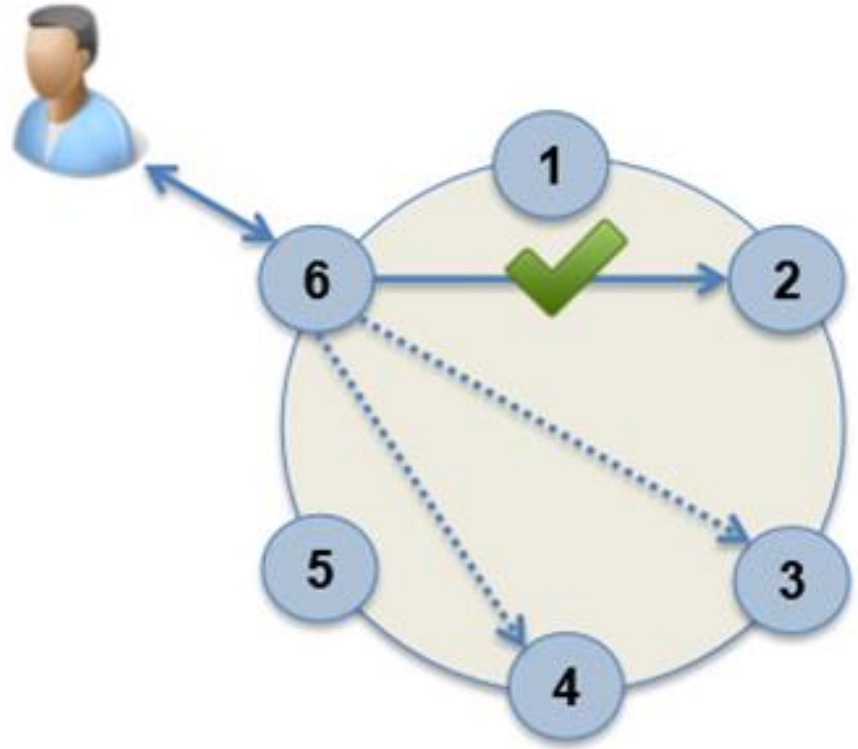
Cassandra : nœud coordinateur



Cassandra : cohérence

- Extension de la notion de **cohérence éventuelle** à une **cohérence ajustable**
- **Choix du niveau de cohérence défini par requête** : clause `USING CONSISTENCY` (niveau `ONE` par défaut)
- **Ecriture** :
  - Niveau `ONE` : écriture effectuée sur au moins un nœud
  - Niveau `ALL` : écriture effectuée sur tous les N nœuds (avec $N=ReplicationFactor$)
  - Niveau `QUORUM` : écriture effectuée sur M nœuds (avec $M=|ReplicationFactor/2| + 1$)
- **Lecture** :
  - Niveau `ONE` : enregistrement retourné par le premier nœud qui répond
  - Niveau `ALL` : Requête sur tous les nœuds et sélection de l'enregistrement avec l'estampille (*timestamp)*la plus récente (Si un nœud ne répond pas, l'opération échoue)
  - Niveau `QUORUM` :Requête sur M nœuds et sélection de l'enregistrement avec le l'estampille (*timestamp)* la plus récents retournée par la majorité des nœuds

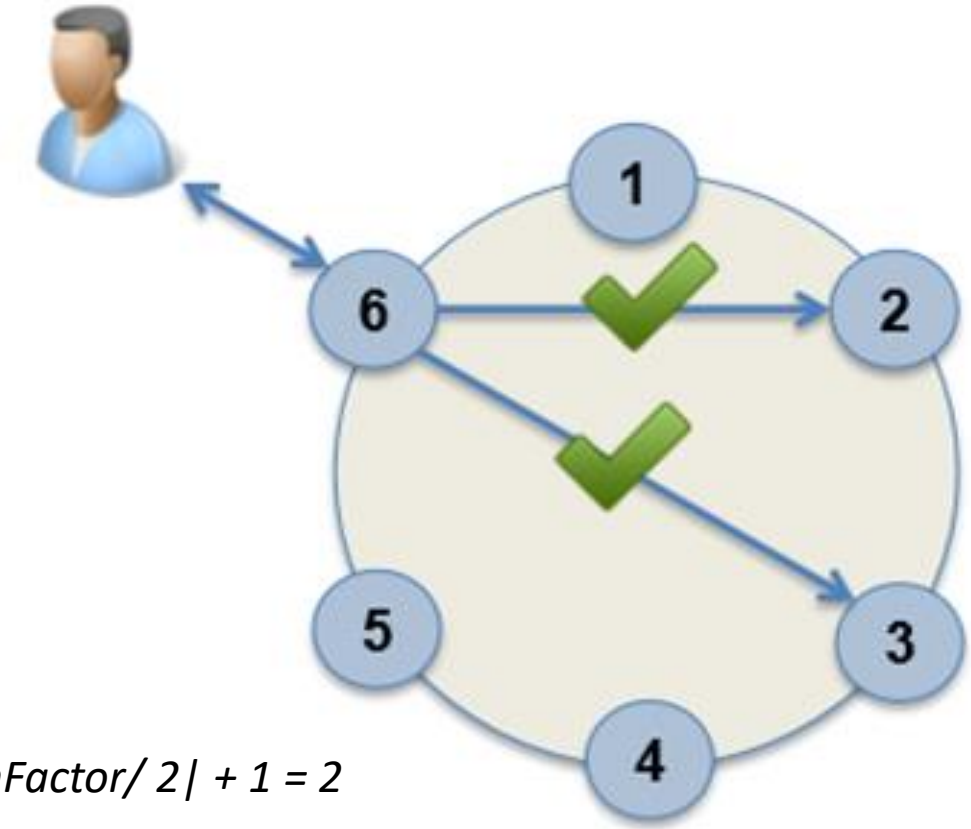
Cassandra : niveau de cohérence ONE en écriture avec un facteur de réplication de 3



Opération réussie

Écriture asynchrone

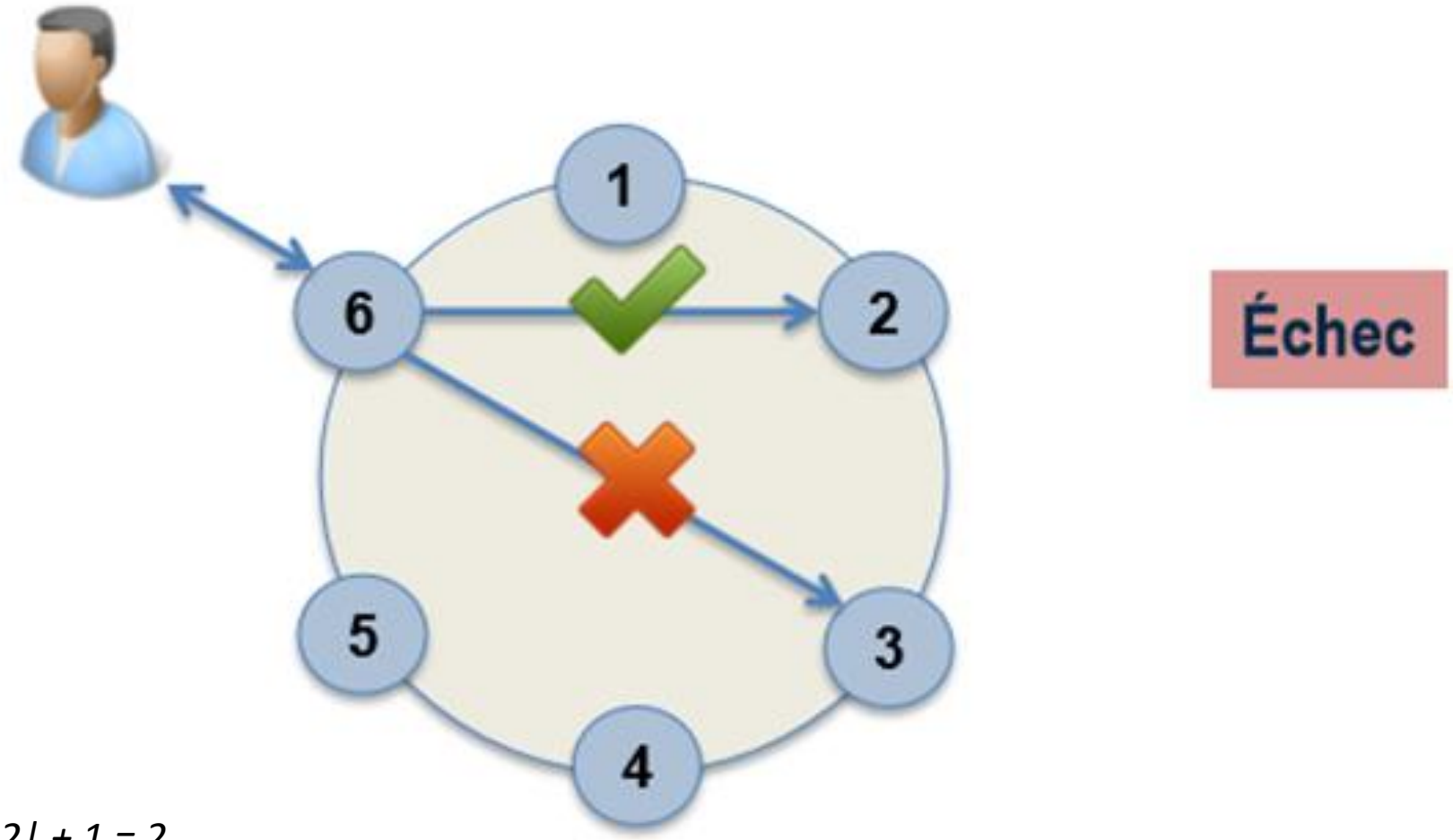
Cassandra : niveau de cohérence QUORUM en écriture avec un facteur de réplication de 2



Opération réussie

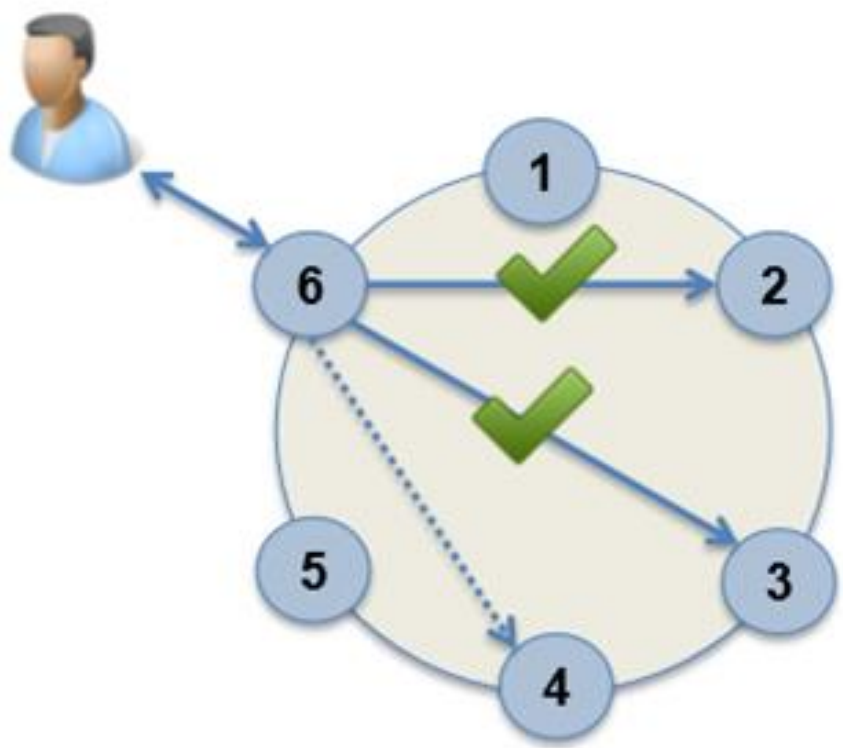
M=|ReplicationFactor/ 2| + 1 = 2

Cassandra : niveau de cohérence QUORUM
Exemple d'échec en écriture avec un facteur de rép. de 2



M=|ReplicationFactor/ 2| + 1 = 2

Cassandra niveau de cohérence QUORUM en écriture avec un facteur de réplication de 3

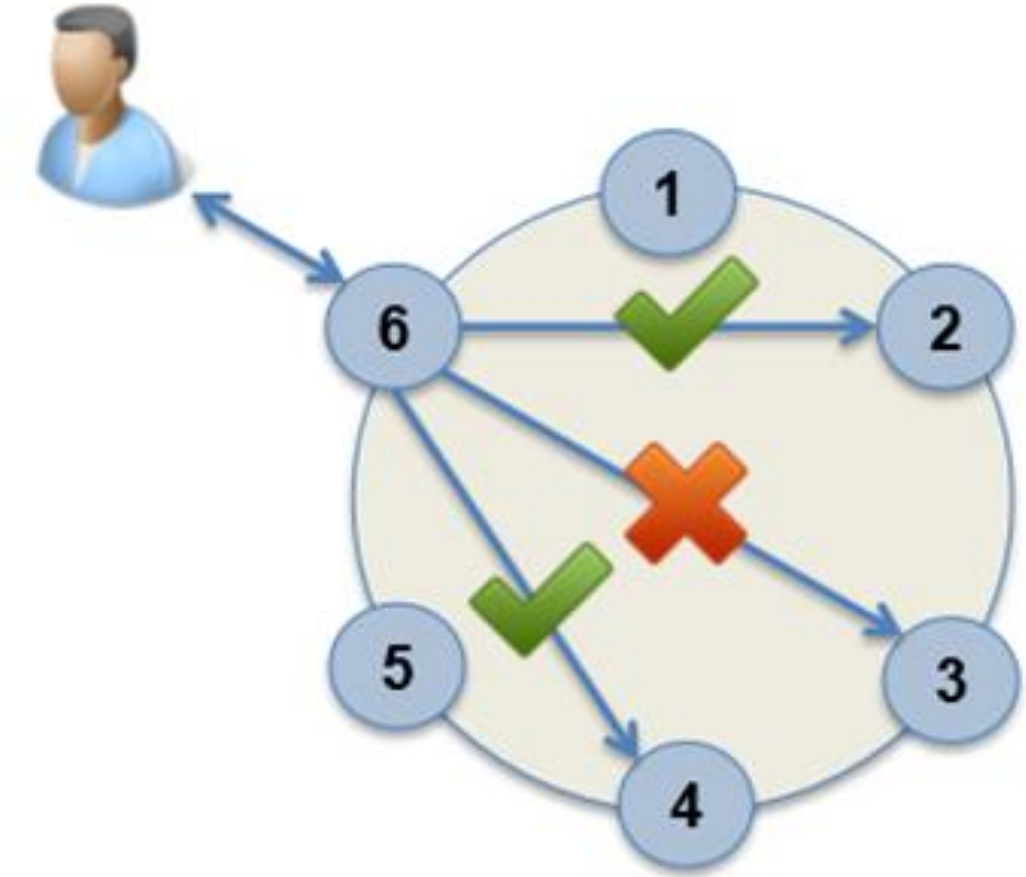


Opération réussie

M=|ReplicationFactor/ 2| + 1 = 2

Écriture asynchrone

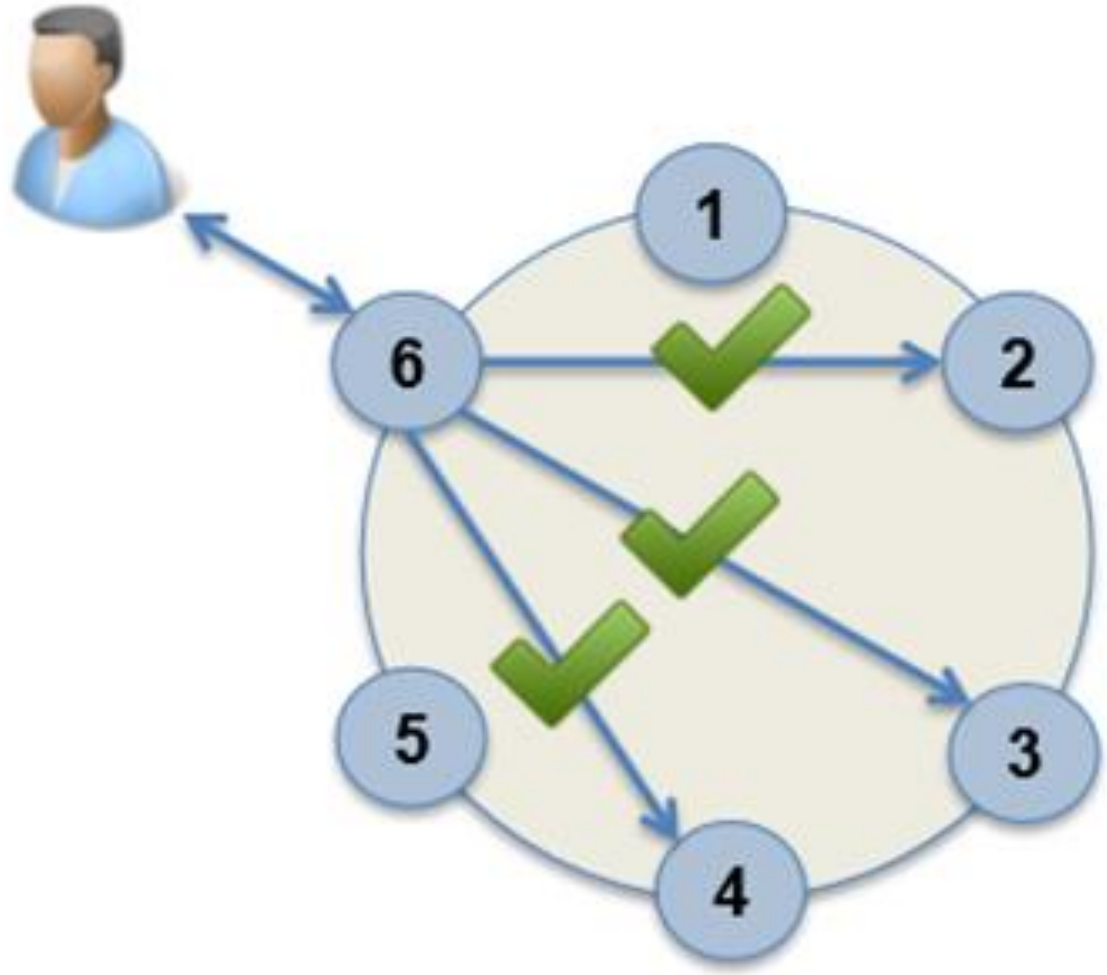
Cassandra : niveau de cohérence QUORUM
Exemple d'échec en écriture avec un facteur de rép. de 3



Opération réussie

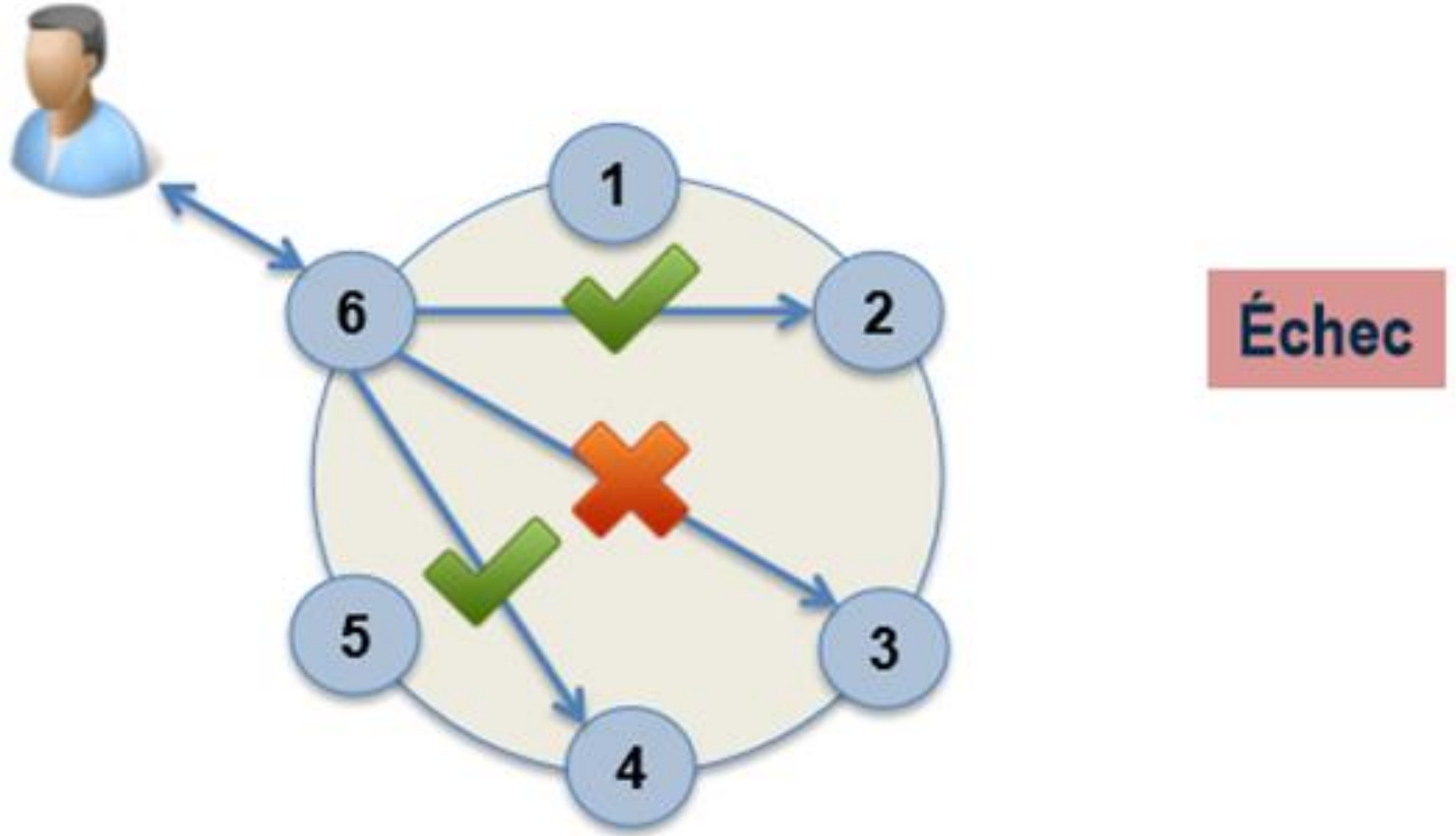
M=|ReplicationFactor/ 2| + 1 = 2

Cassandra niveau de cohérence ALL en écriture avec un facteur de réplication de 3



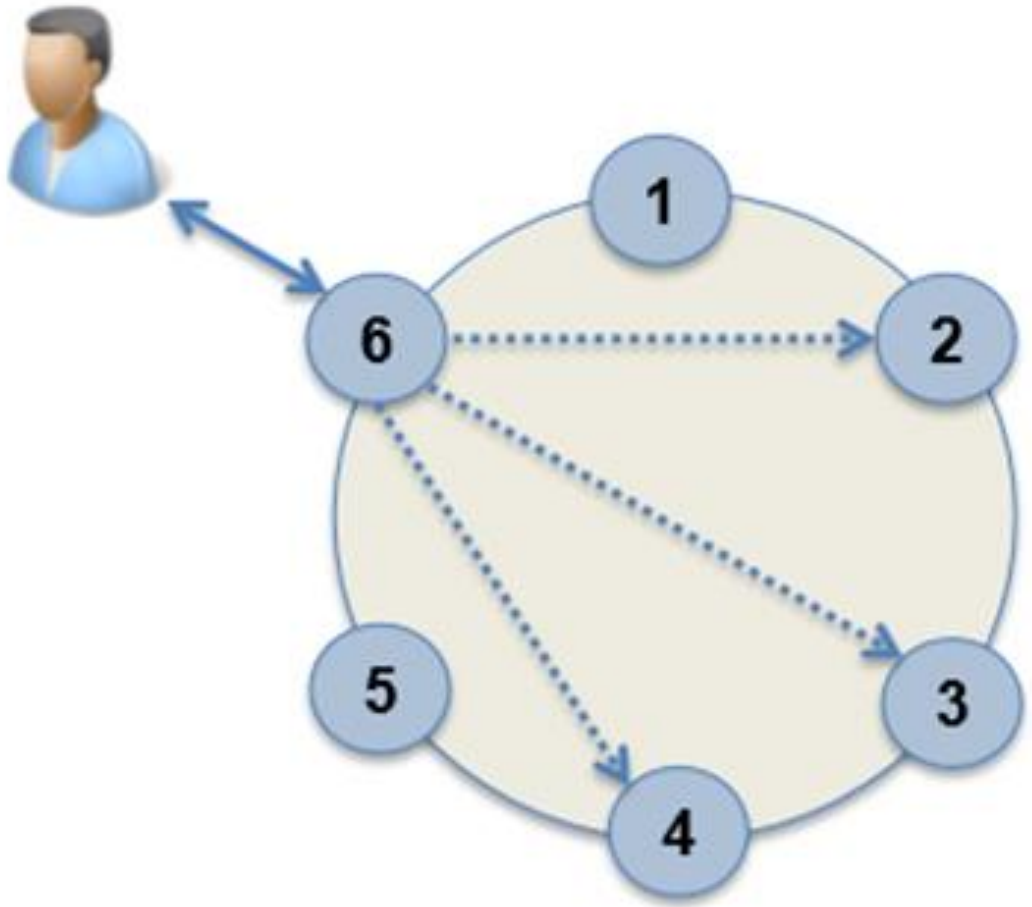
Opération réussie

Cassandra : niveau de cohérence ALL
Exemple d'échec en écriture avec un facteur de rép. de 3

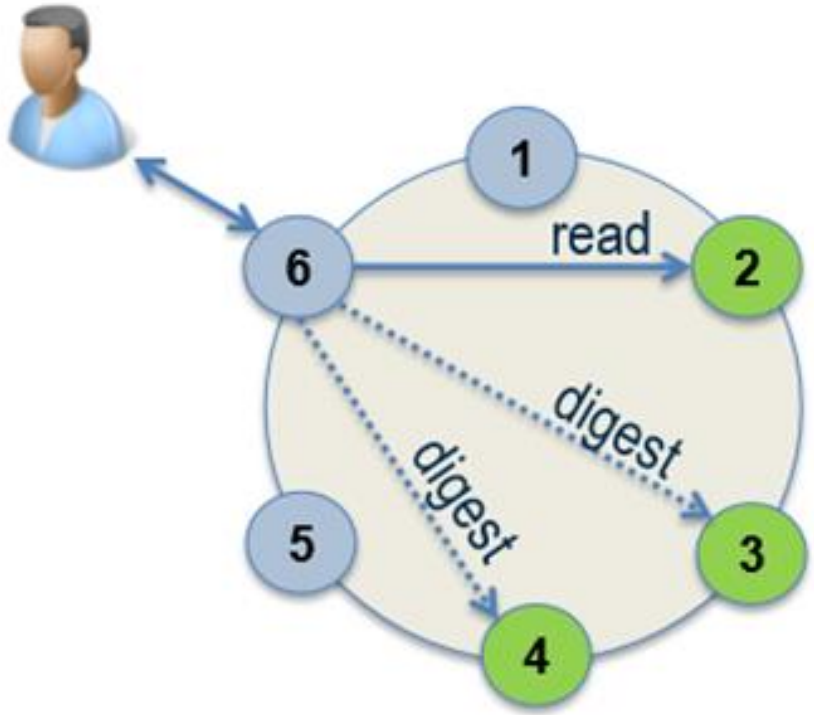


Cassandra : niveau de cohérence ONE en lecture avec un facteur de réplication de 3 - lecture sur n'importe quel nœud

Le client ne sait où la donnée sera lue.



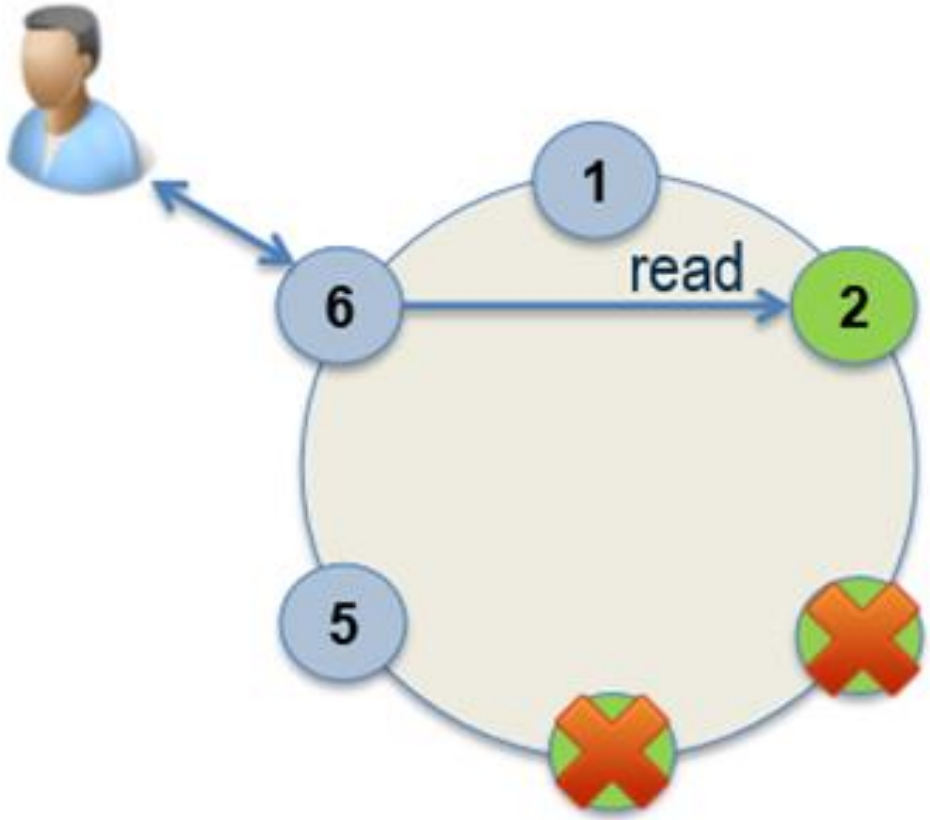
Cassandra : niveau de cohérence ONE en lecture avec un facteur de réplication de 3 – possibilité de « réparer » la lecture



Opération réussie

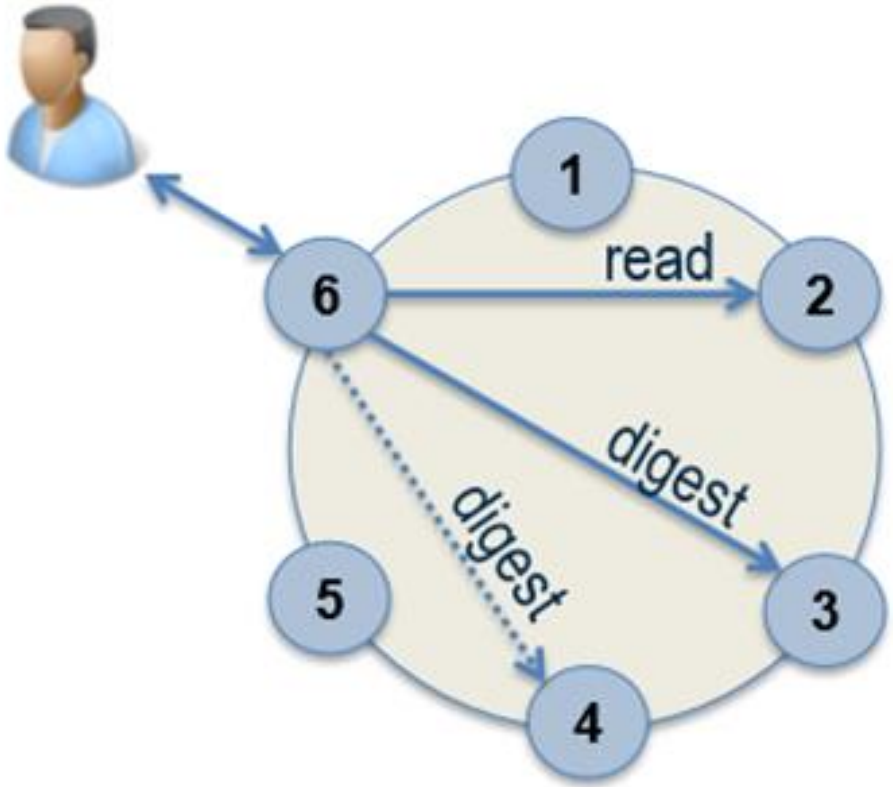
lecture asynchrone , puis « read repair » si besoin et si configuré

Cassandra : niveau de cohérence ONE en lecture avec un facteur de réplication de 3 – réussite même en cas de panne d'un ou plusieurs nœuds



Opération réussie

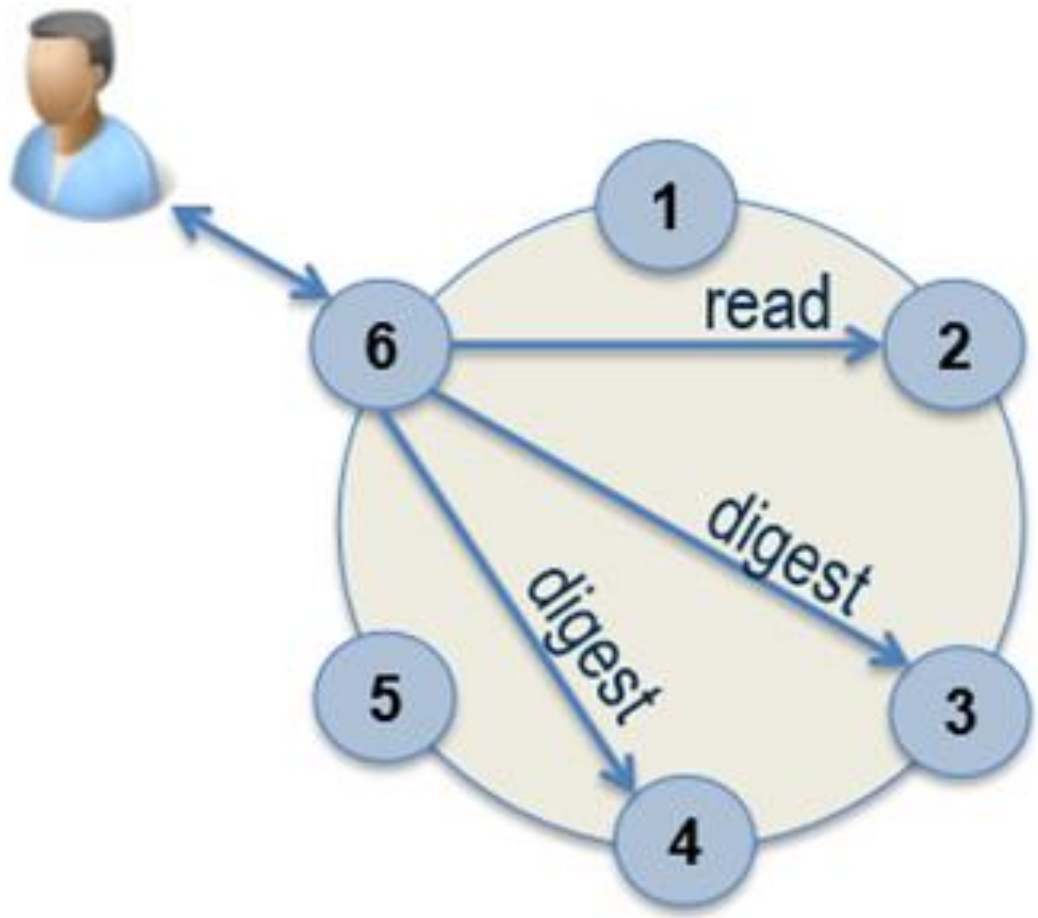
Cassandra : niveau de cohérence QUORUM en lecture avec un facteur de réplication de 3



Opération réussie

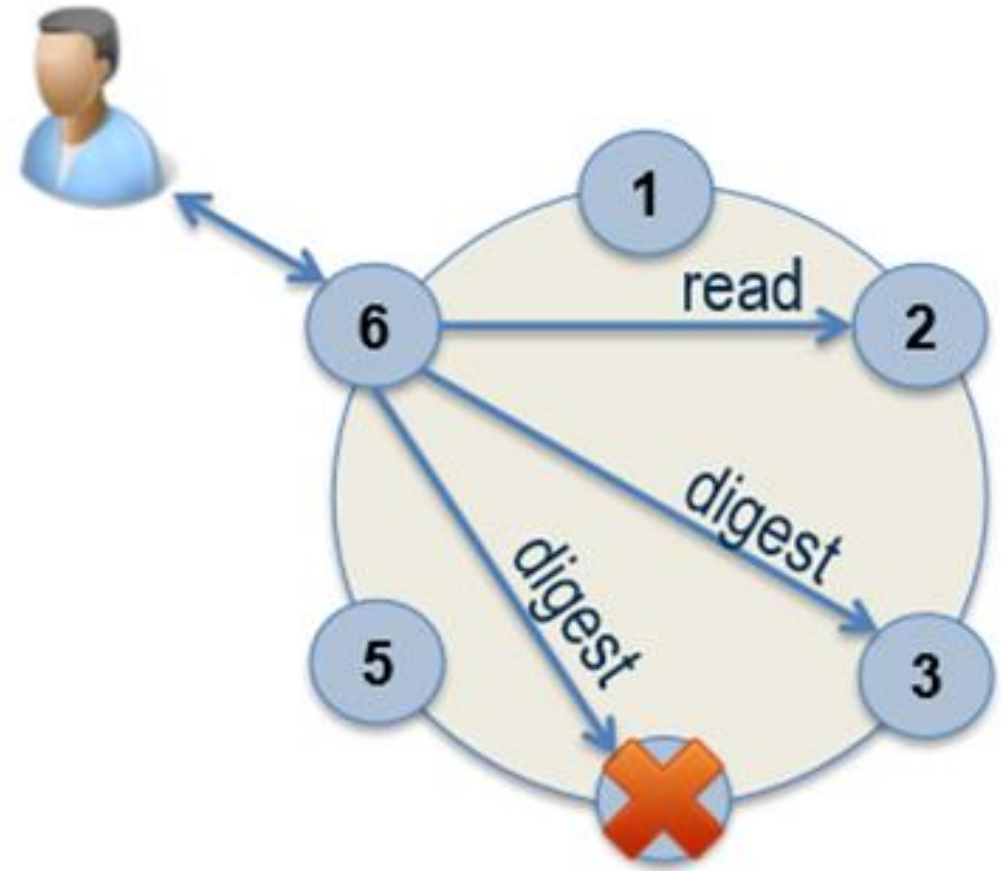
M=|ReplicationFactor/ 2| + 1 = 2

Cassandra : niveau de cohérence ALL en lecture avec un facteur de réplication de 3



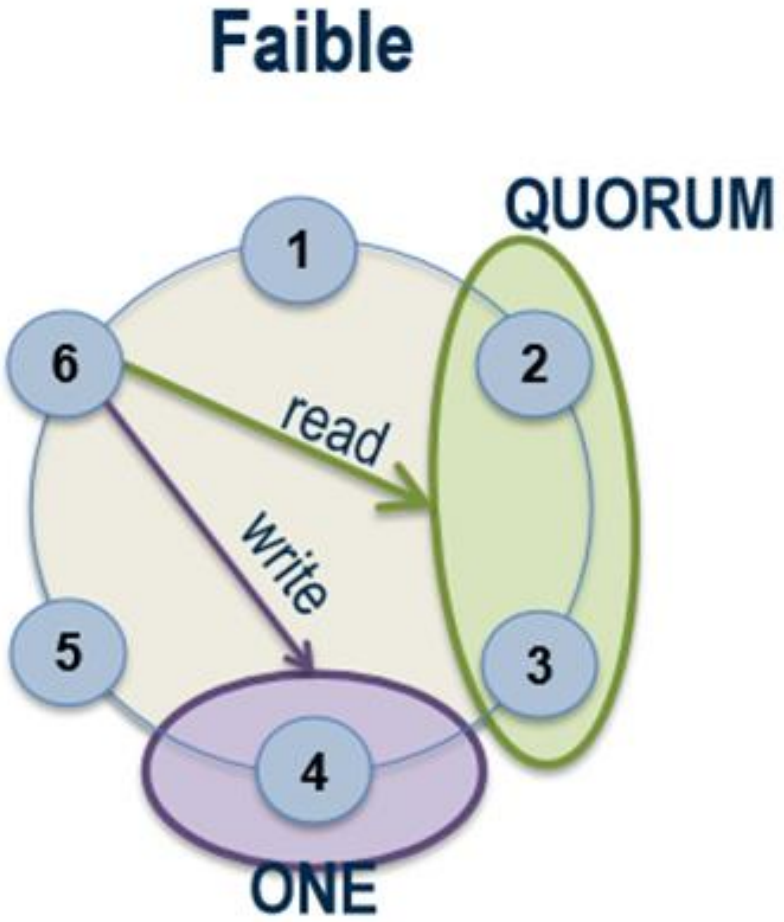
Opération réussie

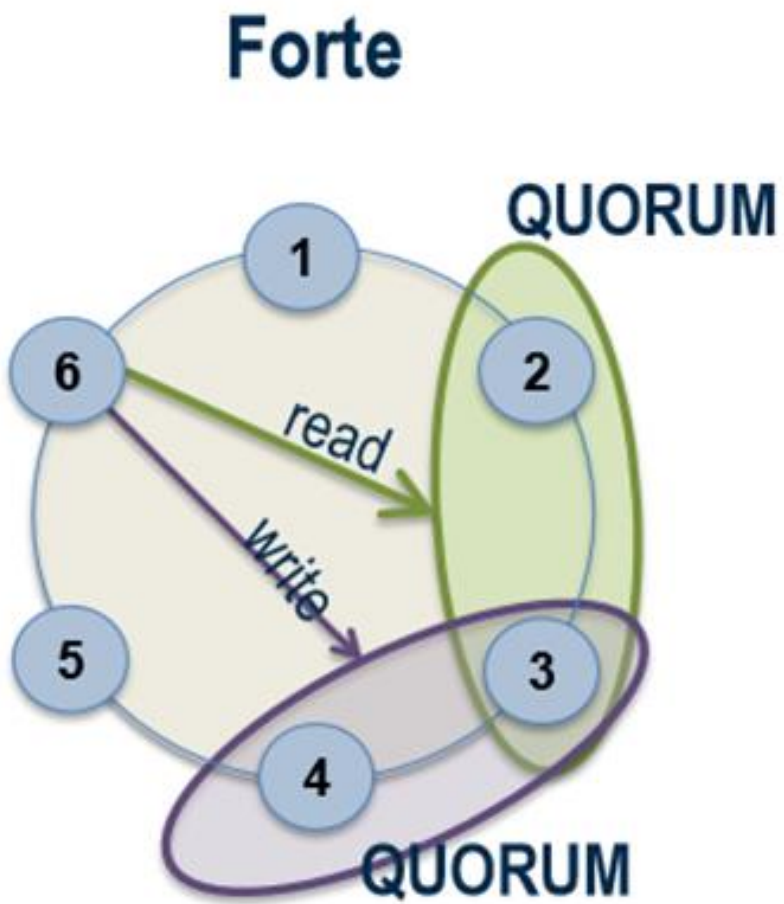
Cassandra : niveau de cohérence ALL en lecture Exemple d'échec en écriture avec un facteur de rép. de 3



Échec

Cassandra : Moduler le niveau de cohérence
Exemple avec un facteur de réplication de 3





Cassandra : conclusion


Rapidité d'écriture


Haute performance, Décentralisation, Support de réplication


Cohérence réglable


Performances de lecture dépendant du modèles de données


Pas d'intégrité référentielle


Limitation des tailles des colonnes et des super-colonnes

Exemple d'applications polyglottes utilisant Cassandra et Redis