Modèle clé-valeur

- **Principes**
- **Redis**
- **HSTORE sous PostgreSQL**
- **Dynamo**

Base de données Clé-Valeur

- Similaire à table de hachage : Stockage des paires clé-valeur, identifiables par leur clé
- Similaire à une table relationnelle avec deux colonnes
- Très bonnes performances grâce à l'indexation sur la clé
- Le plus simple espace de stockage de type NoSQL
- 3 opérations principales : Récupérer/définir une valeur pour une clé, supprimer une clé

Clé-Valeur : cas d'utilisation

- **Cas d'utilisation :**
  - Stockage des informations d'une session pour un site web
  - Profils et préférences d'un utilisateur
  - Panier d'achats sur un site de e-commerce : Stockage du panier d'achats actuel d'un utilisateur
- **Non applicable aux:**
  - Données liées à différentes clés
  - Requêtes sur les valeurs

Principaux moteurs Clé - Valeur

☐ include secondary database models

71 systems in ranking, March 2025

Rank Mar 2025	Rank Feb 2025	Rank Mar 2024	DBMS	Database Model	Score Mar 2025	Score Feb 2025	Score Mar 2024
1.	1.	1.	Redis	Key-value, Multi-model			
2.	2.	2.	Amazon DynamoDB	Multi-model			
3.	3.	3.	Microsoft Azure Cosmos DB	Multi-model			
4.	4.	4.	Memcached	Key-value			
5.	5.	↑ 6.	etcd	Key-value			
6.	6.	↓ 5.	Hazelcast	Key-value, Multi-model			
7.	7.	7.	Aerospike	Multi-model			
8.	8.	8.	Ehcache	Key-value	4.22	-0.10	-1.33
9.	9.	9.	Riak KV	Key-value	3.39	+0.06	-1.56
10.	10.	↑ 17.	Oracle NoSQL	Multi-model	3.35	+0.06	+0.09
11.	11.	↓ 10.	RocksDB	Key-value	3.15	+0.09	-1.37
12.	↑ 13.	12.	Apache Ignite	Multi-model	2.99	+0.11	-1.04

Key-value store
Document store
Graph DBMS
Spatial DBMS
Search engine
Time Series DBMS
Vector DBMS

Redis (*REmote DIctionary Server*)

- Projet open-source écrit en ANSI C jusqu'en 2024 (avant Redis 7.4)
- Lancé en 2009, puis développé par *Vmware* à partir de mars 2010 et par Redis Labs depuis juin 2015
- Similaire à une gigantesque *Hash Map* ou un dictionnaire Python
- Stocke les données en mémoire (*in memory DB*) - peut également servir de cache de données
- Possibilité de stocker des données sur disque (*snapshot*) : Au démarrage du service, dernier snapshot mis en mémoire
- Souvent utilisé pour augmenter la vitesse de réponse des sites web (Facebook, Twitter, Wikipedia) créés à partir de bases de données

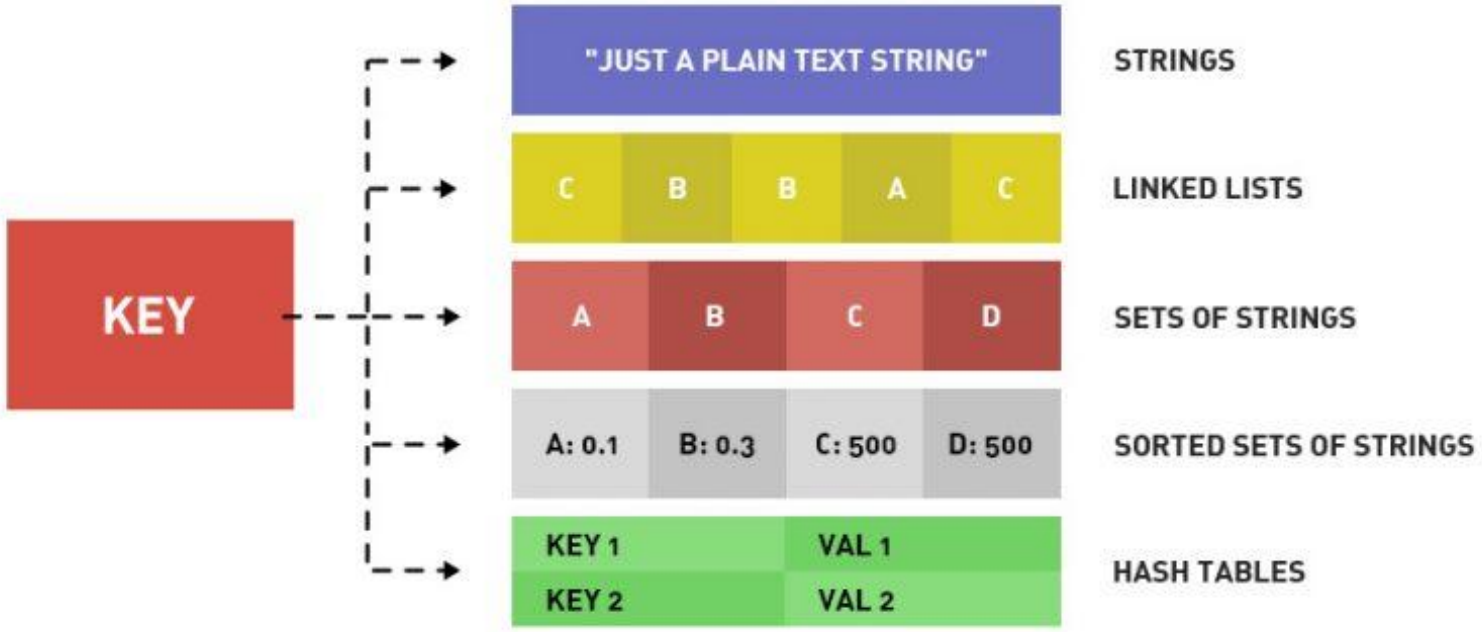
Adapté de perso.isep.fr/rchiky/nosql/TelecomParisTechCours2-NoSQL.pdf et http://sametmax.com/redis-pourquoi-et-comment/

Redis : liens utiles

- Site web officiel Redis : https://redis.io/
- Liste des commandes Redis : https://redis.io/commands/
- Console : https://onecompiler.com/redis
- Vidéo sur les bases de Redis et sur son installation : https://grafikart.fr/tutoriels/redis
- *Redis Explained* (2022) : https://architecturenotes.co/redis/
- Sur AWS : https://aws.amazon.com/fr/redis/
- Sur Microsoft Azure : https://azure.microsoft.com/fr-fr/services/cache/

Redis : types de données

- Clé unique, *binary-safe*, pouvant aller jusqu'à 512MB – d'une chaîne de caractères, de préférence ASCII à une image JPEG
- Possibilité de donner une durée de vie à une clé
- Plusieurs types de valeurs possibles: chaîne (*string*), liste (*list*), ensemble non ordonné *(set)* , ensemble ordonné (*Zset*) et table de hachage (*hash*),

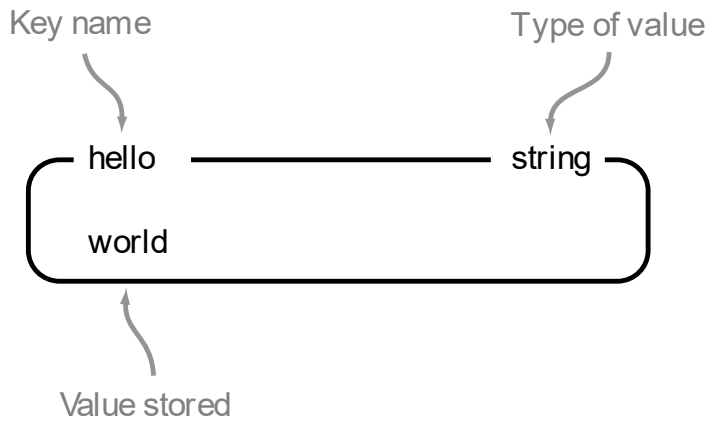


Redis : valeurs de type chaîne

Commandes pour manipuler les chaînes :

- SET ajoute une nouvelle chaine
- GET récupère la valeur associée à une clé
- DEL supprime une clé du *store*

```
> SET ma:clé "ma valeur"
OK
> GET ma:clé
"ma valeur"
```



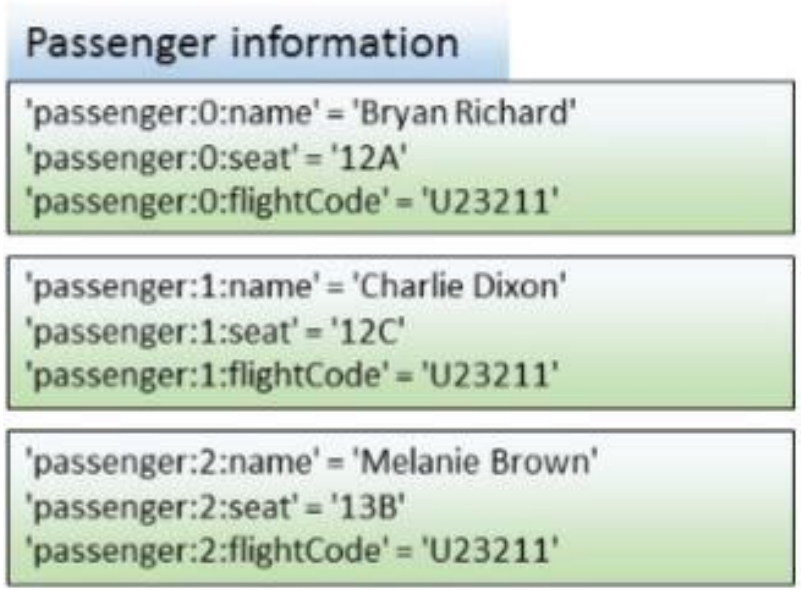
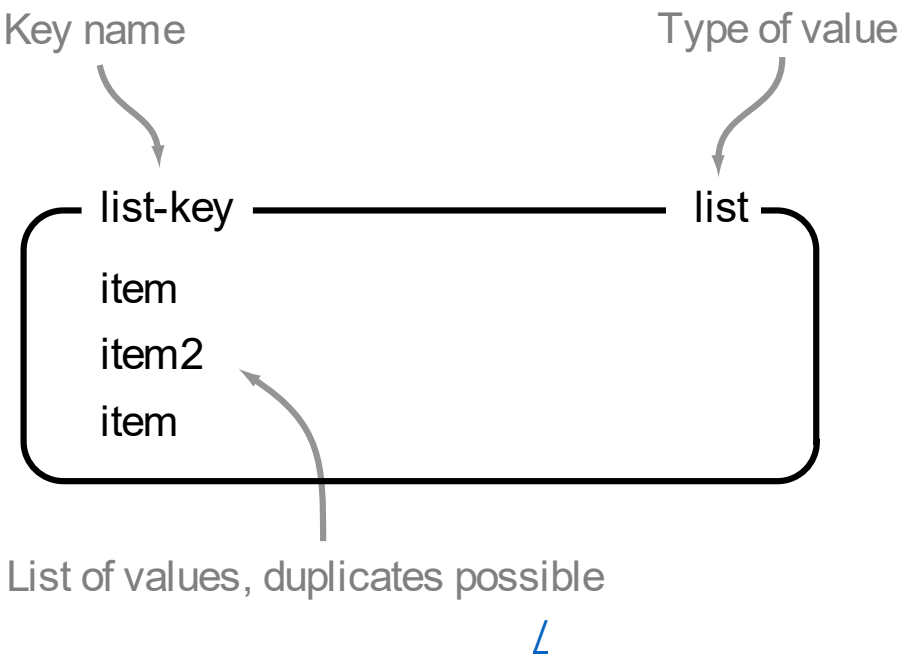


Image reprise de https://techannotation.wordpress.com/2018/02/15/data-model-in-redis/ et https://redis.com/fr/redis-enterprise/structures-de-donnees/

Redis : listes

Commandes pour manipuler les listes :

- LPUSH ajoute une entrée en début de la liste
- LPOP supprime le premier élément de la liste
- RPUSH ajoute une entrée en fin de la liste
- RPOP supprime de dernier élément de la liste
- LRANGE extrait une sous-liste depuis une liste

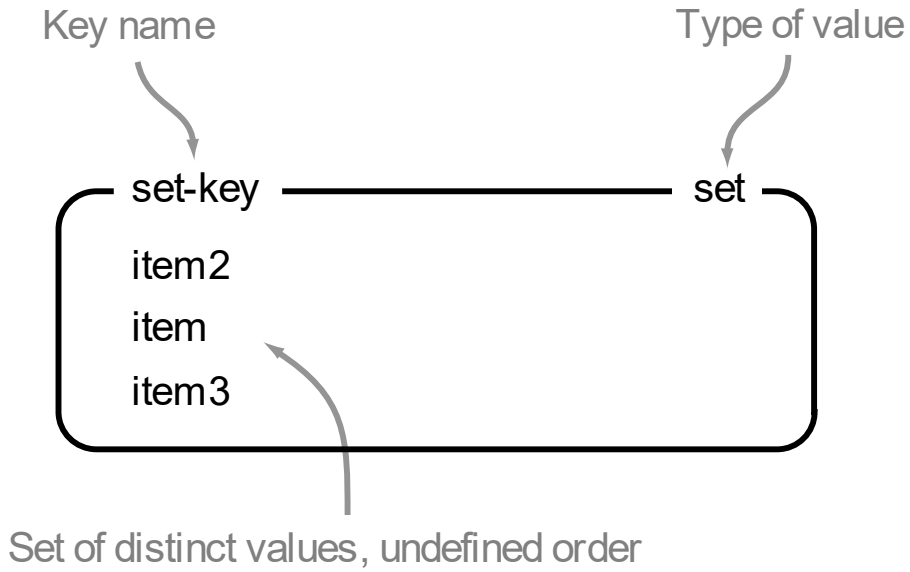



```
redis 127.0.0.1:6379> LPUSH LastWebVisitors Webuser:1
(integer) 1
redis 127.0.0.1:6379> LPUSH LastWebVisitors Webuser:2
(integer) 2
redis 127.0.0.1:6379> LPUSH LastWebVisitors Webuser:Temp
(integer) 3
redis 127.0.0.1:6379> LRANGE LastWebVisitors 0 -1
1) "Webuser:Temp"
2) "Webuser:2"
3) "Webuser:1"
```

Redis : ensembles non ordonnés

Commandes pour manipuler les ensembles :

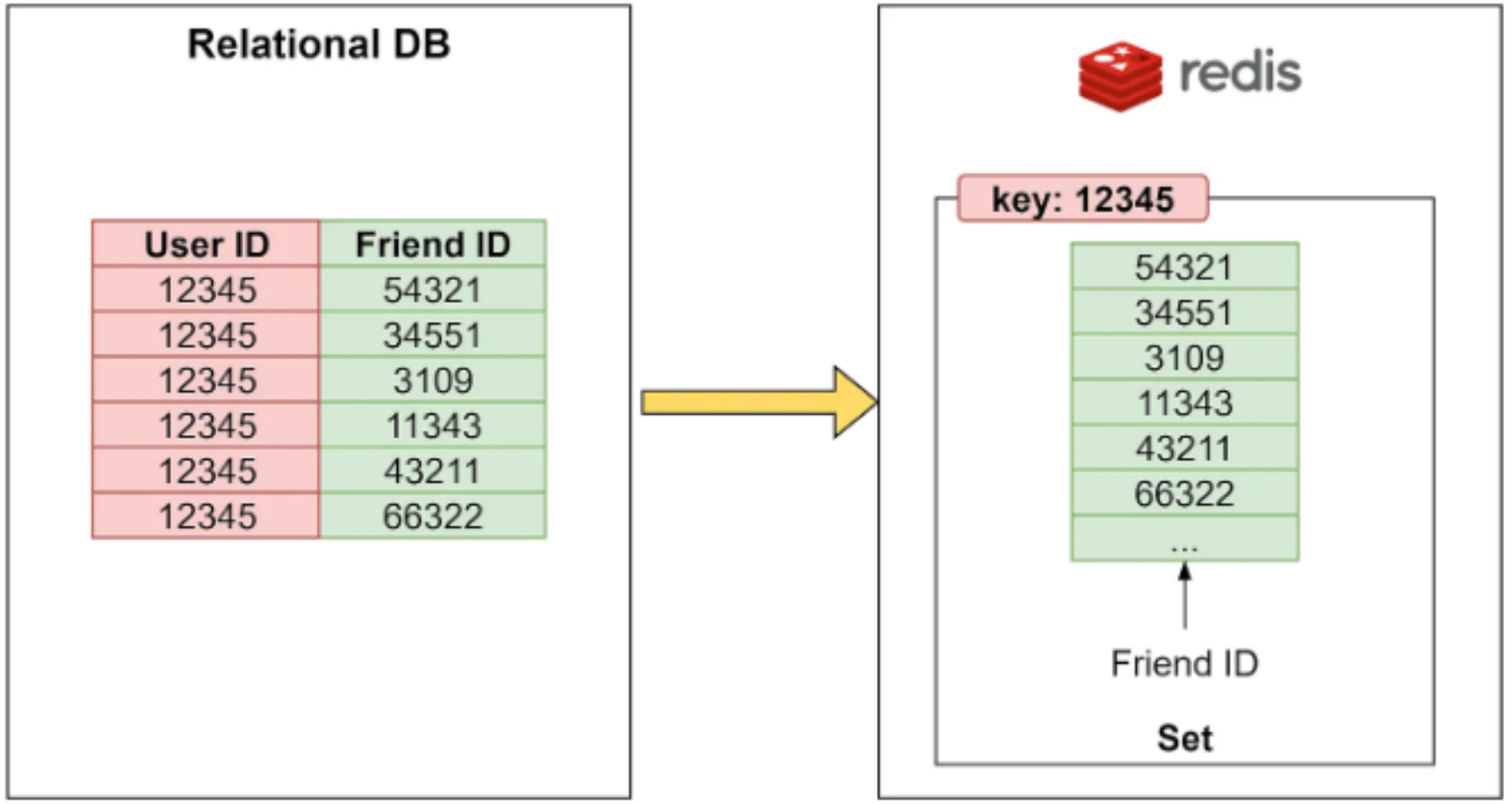
- SADD ajoute un élément dans l'ensemble
- SMEMBERS renvoie les éléments de l'ensemble
- SISMEMBER teste l'appartenance
- SREM supprime un élément
- SINTER, SUNION, SDIFF opérations ensemblistes




```
redis 127.0.0.1:6379> SADD Webuser:HasGMail Webuser:3
(integer) 1
redis 127.0.0.1:6379> SADD Webuser:HasGMail Webuser:2
(integer) 1
redis 127.0.0.1:6379> SADD Webuser:HasGMail Webuser:5
(integer) 1
redis 127.0.0.1:6379> SMEMBERS Webuser:HasGMail
1) "Webuser:2"
2) "Webuser:5"
3) "Webuser:3"
```

```
redis 127.0.0.1:6379> SMEMBERS Webuser:IsFrench
1) "Webuser:2"
2) "Webuser:9"
3) "Webuser:7"
4) "Webuser:5"

# Donne moi les utilisateurs Français avec GMail
redis 127.0.0.1:6379> SINTER Webuser:IsFrench Webuser:HasGMail
1) "Webuser:2"
2) "Webuser:5"
```

Redis: exemple d'ensemble non ordonné



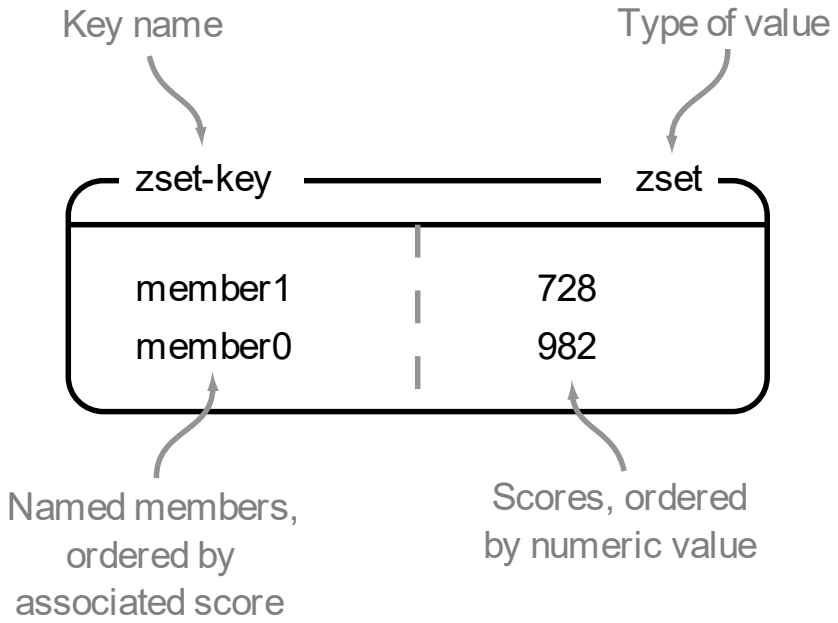

```
SELECT friend_id
FROM relationship_table
WHERE user_id = 12345
```

```
SMEMBERS 12345
```

Redis : ensembles ordonnés

Commandes pour manipuler les ZSET :

- ZADD ajoute un élément dans l'ensemble avec un certain score
- ZRANGE renvoie les éléments de l'ensemble
- ZRANGEBYSCORE renvoie les éléments de l'ensemble par score
- ZREM supprime un élément



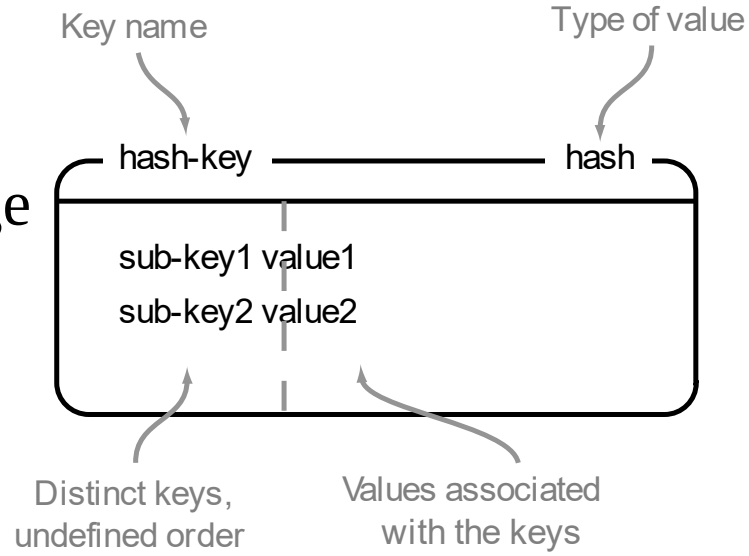

```
redis rio.zaptravel.com:6379> ZADD DealsFranceByPrice 59 Trip:Lyon
(integer) 1
redis rio.zaptravel.com:6379> ZADD DealsFranceByPrice 259 Trip:Venise
(integer) 1
redis rio.zaptravel.com:6379> ZADD DealsFranceByPrice 130 Trip:Bordeaux
(integer) 1
redis rio.zaptravel.com:6379> ZADD DealsFranceByPrice 990 Trip:Paris
(integer) 1
```

```
redis rio.zaptravel.com:6379> ZRANGEBYSCORE DealsFranceByPrice 0 200 WITHSCORES LIMIT 0 10
1) "Trip:Lyon"
2) "59"
3) "Trip:Bordeaux"
4) "130"
```

Redis : table de hachage

Commandes pour manipuler les tables de hachage :

- HSET ajoute une entrée dans la table de hachage d'une clé
- HVALS récupère la table de hachage complète d'une clé
- HGET récupère la valeur d'une entrée d'une table de hachage
- HGETALL récupère les données associées à une clé
- HDEL supprime une entrée d'une table de hachage

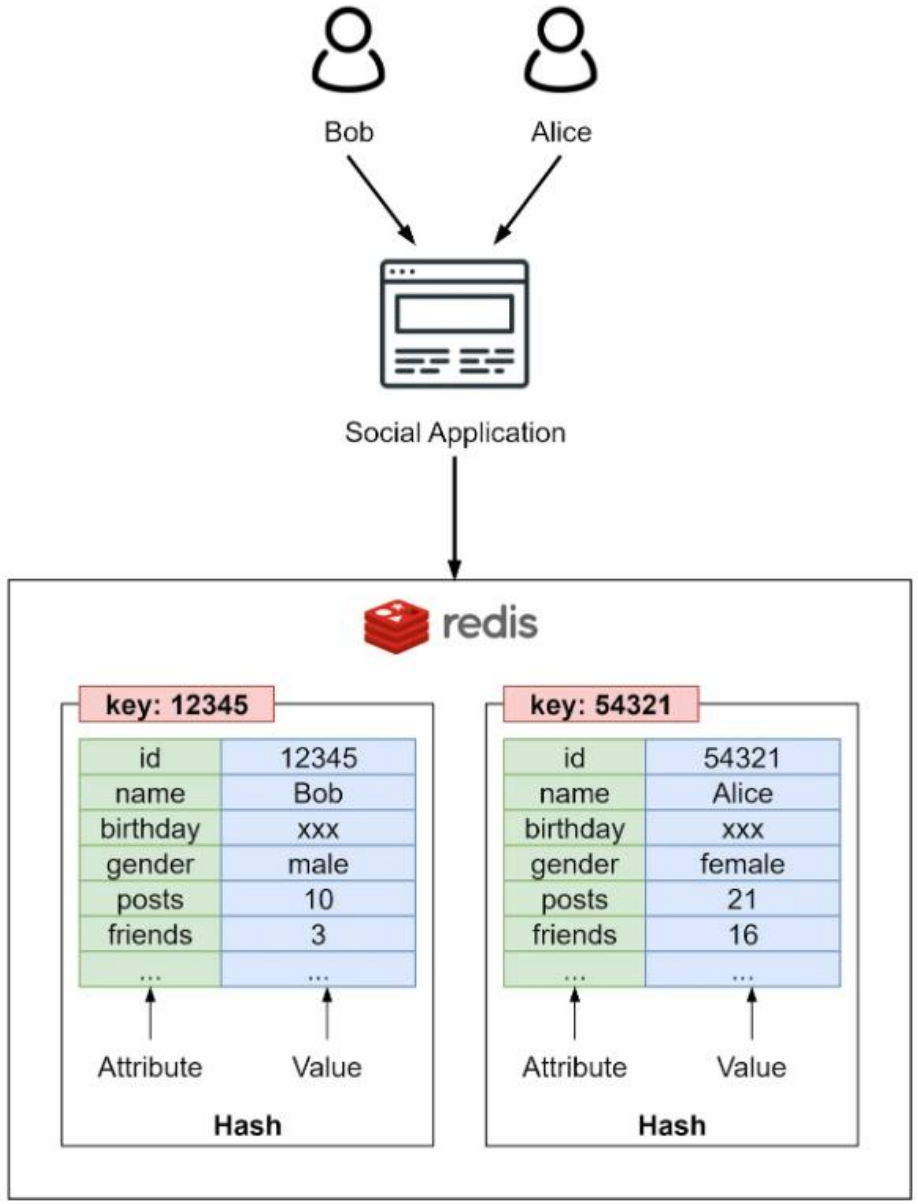



```
> HSET utilisateur:1 nom "Thomas Bourgier"
> HSET utilisateur:1 email "t.bourgier@exemple.com"
> HSET utilisateur:1 mot_de_passe "1234"
> HSET utilisateur:2 nom "Myriam Fois" email "m.fois@exemple.com"
```

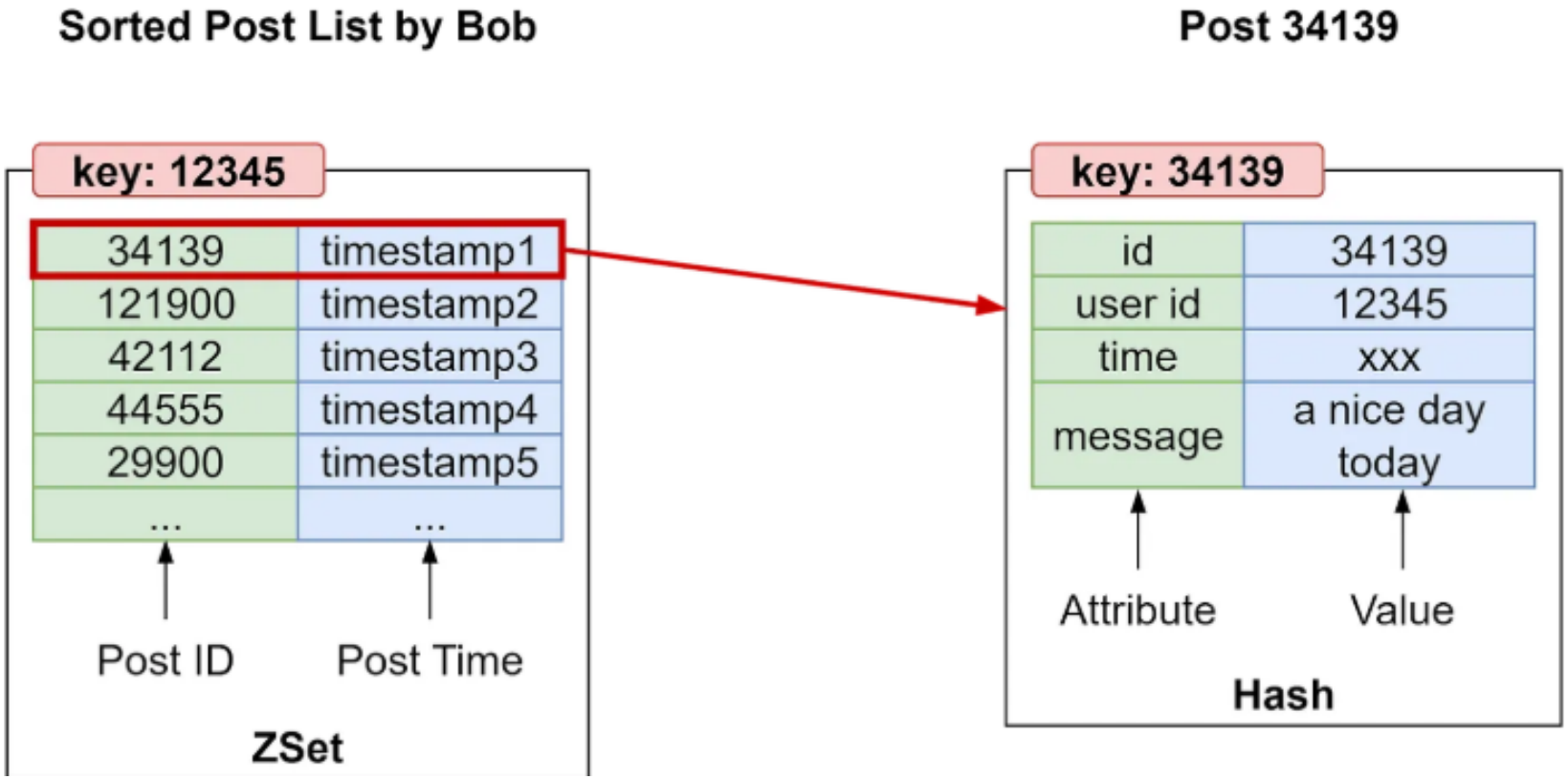
Redis: exemple d'instructions pour les tables de hachage

```
> HGET ALL utilisateur:1
1) "nom"
2) "Thomas Bourgier"
3) "email"
4) "t.bourgier@exemple.com"
5) "mot_de_passe"
6) "1234"

> HGET utilisateur:1 nom
"Thomas Bourgier"

> HSET utilisateur:1 visites 10
> HINCRBY utilisateur:1 visites 1
(integer) 11
```

Redis: exemple de tables de hachage



Redis: exemple de bases de données



Redis : opération arithmétique

- Possibilité d'appliquer des opérations arithmétiques sur des entiers :

```
> SET mon:entier 1
OK
> INCR mon:entier
(integer) 2
> INCRBY mon:entier 10
(integer) 12
```

- Garantie par Redis que plusieurs appels simultanés à la fonction INCR seront exécutés de manière séquentielle

Redis : extension JSON (1/2)

Possibilité de gérer des documents JSON avec le module *RedisJSON*

Store (or update) a JSON document associated with a key in Redis

```
.cloud:6379> JSON.SET myDoc . '{"title": "css", "colors": ["green"]}'
```

Extract the whole document

```
redis.cloud:6379> JSON.GET myDoc .
"{\"title\":\"css\",\"colors\":[\"green\"]}"
```

Extract part of it using a subset of JSONPath

```
redis.cloud:6379> JSON.GET myDoc colors[0]
"\"green\""
```

Redis : extension JSON (2/2)

Possibilité de gérer des documents JSON avec le module *RedisJSON*

Replace a subpart (eg. the string value of a key)

```
redis.cloud:6379> JSON.SET myDoc title '"style"'
OK
```

Add an item to a collection or a map

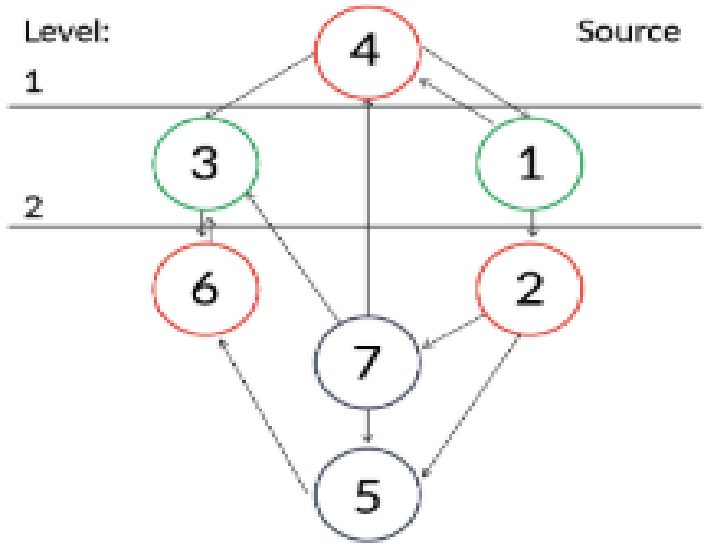
```
redis.cloud:6379> JSON.ARRAPPEND myDoc colors '"red"' '"blue"'
(integer) 3
```

Utilisation de RediSearch pour faire de l'indexation et de la recherche *full-text*

Redis : extension Graphe

Possibilité de gérer des graphes : module RedisGraph (abandonné en 2025)

Destination Nodes \ Source Nodes	1	2	3	4	5	6	7
1	0	0	0	1	0	0	0
2	1	0	0	0	0	0	0
3	0	0	0	1	0	1	1
4	1	0	0	0	0	0	1
5	0	1	0	0	0	0	1
6	0	0	1	0	1	0	0
7	0	1	0	0	0	0	0

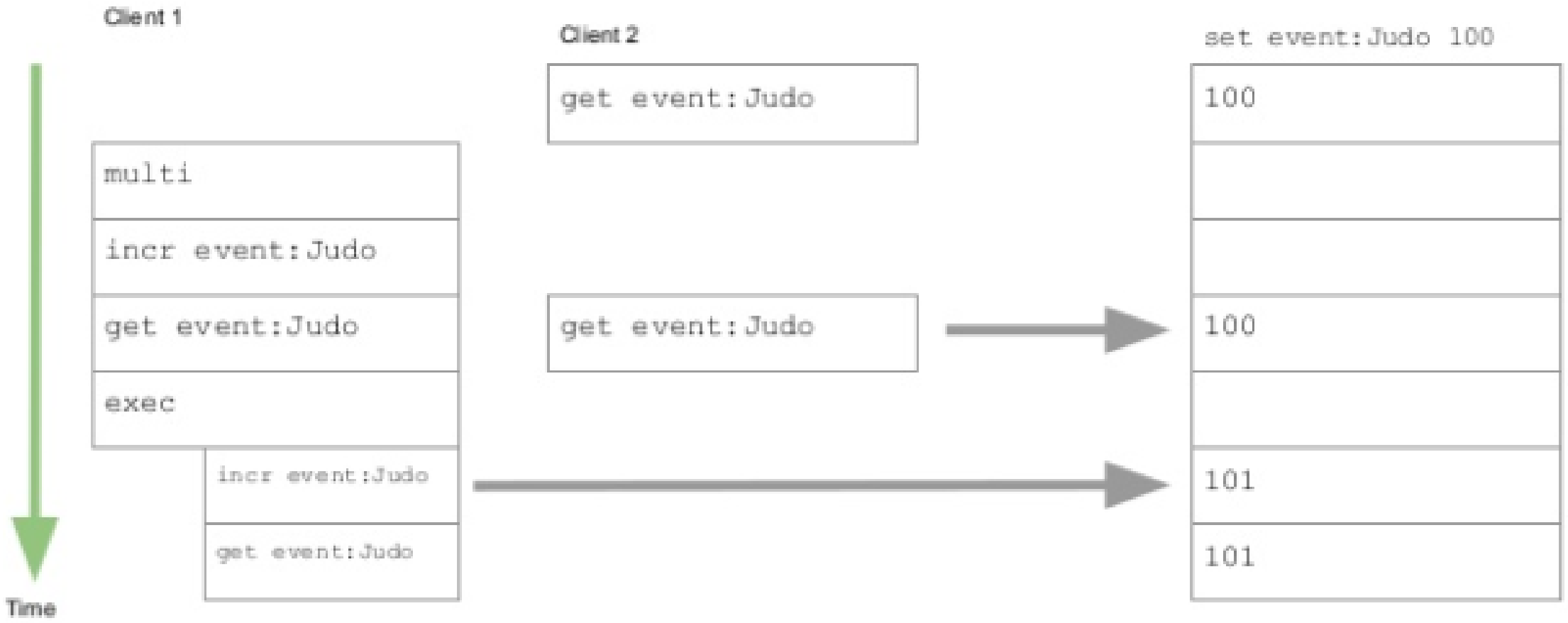



```
$ redis-cli
127.0.0.1:6379> GRAPH.QUERY MotoGP "CREATE (:Rider {name:'Valentino
Rossi'})-[:rides]->(:Team {name:'Yamaha'}), (:Rider {name:'Dani Pedrosa'})-
[:rides]->(:Team {name:'Honda'}), (:Rider {name:'Andrea Dovizioso'})-
[:rides]->(:Team {name:'Ducati'})"
1) 1) Labels added: 2
   2) Nodes created: 6
   3) Properties set: 6
   4) Relationships created: 3
   5) "Query internal execution time: 0.399000 milliseconds"
```

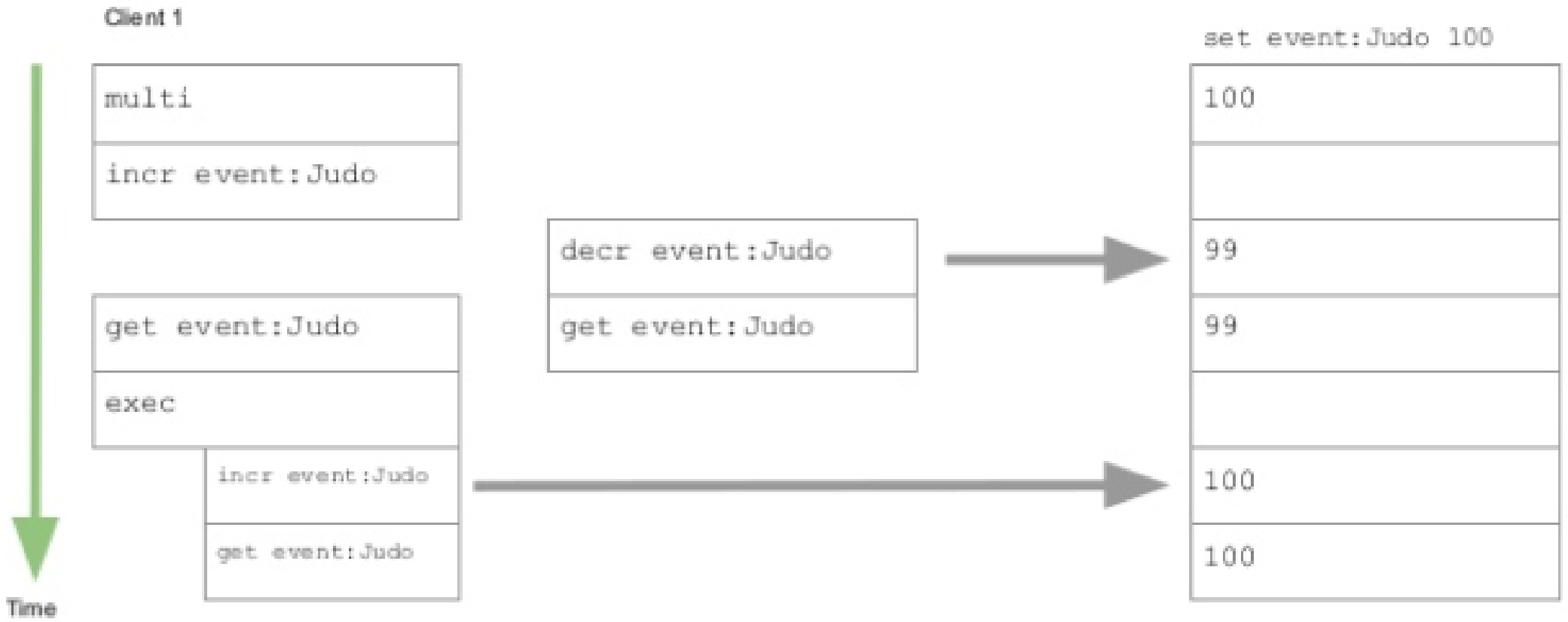
Redis : « transactions »

- Transactions dans Redis non équivalentes aux transactions au sens bases de données SQL
- “Transaction” Redis : bloc d’instructions placées entre les commandes `MULTI` and `EXEC` (ou `DISCARD` pour *cancel*) = bloc atomique
- Commande `MULTI` ⇒ stockage des instructions dans une queue –
- Exécution des instructions stockées dans la queue (QUEUE) lors de la commande `EXEC`.
- Commande `WATCH` / `UNWATCH` ⇒ verrouillage optimiste.

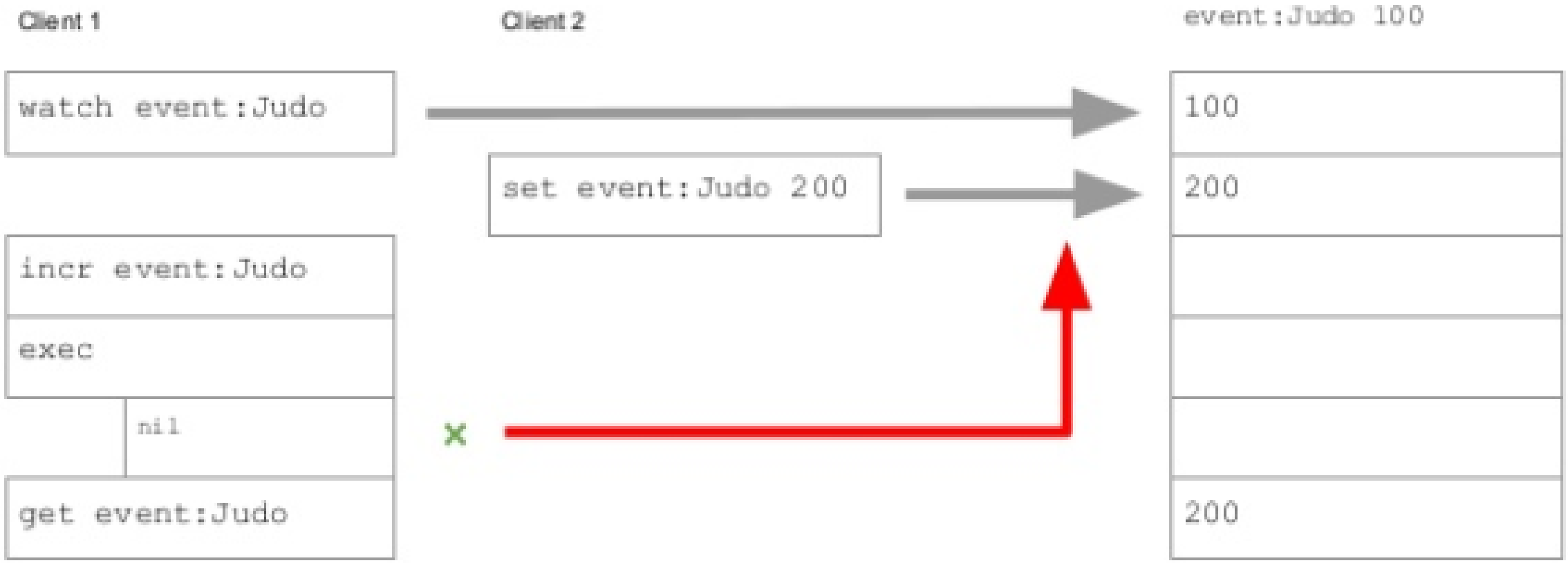
Redis : instruction `multi exec` => « atomicité »



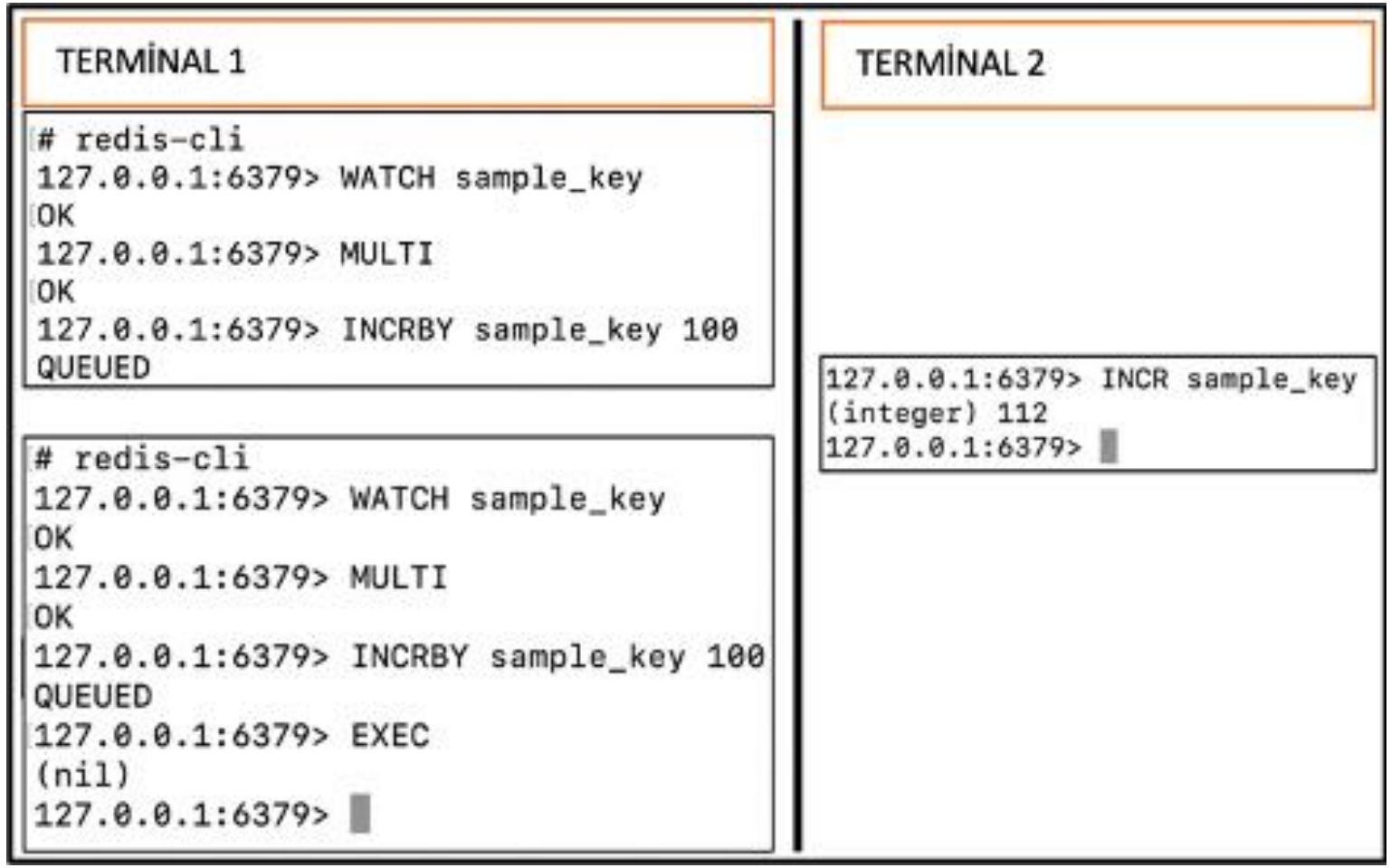
Redis : instruction `multi exec` => pas de « verrouillage »



Redis : gestion optimiste de la concurrence

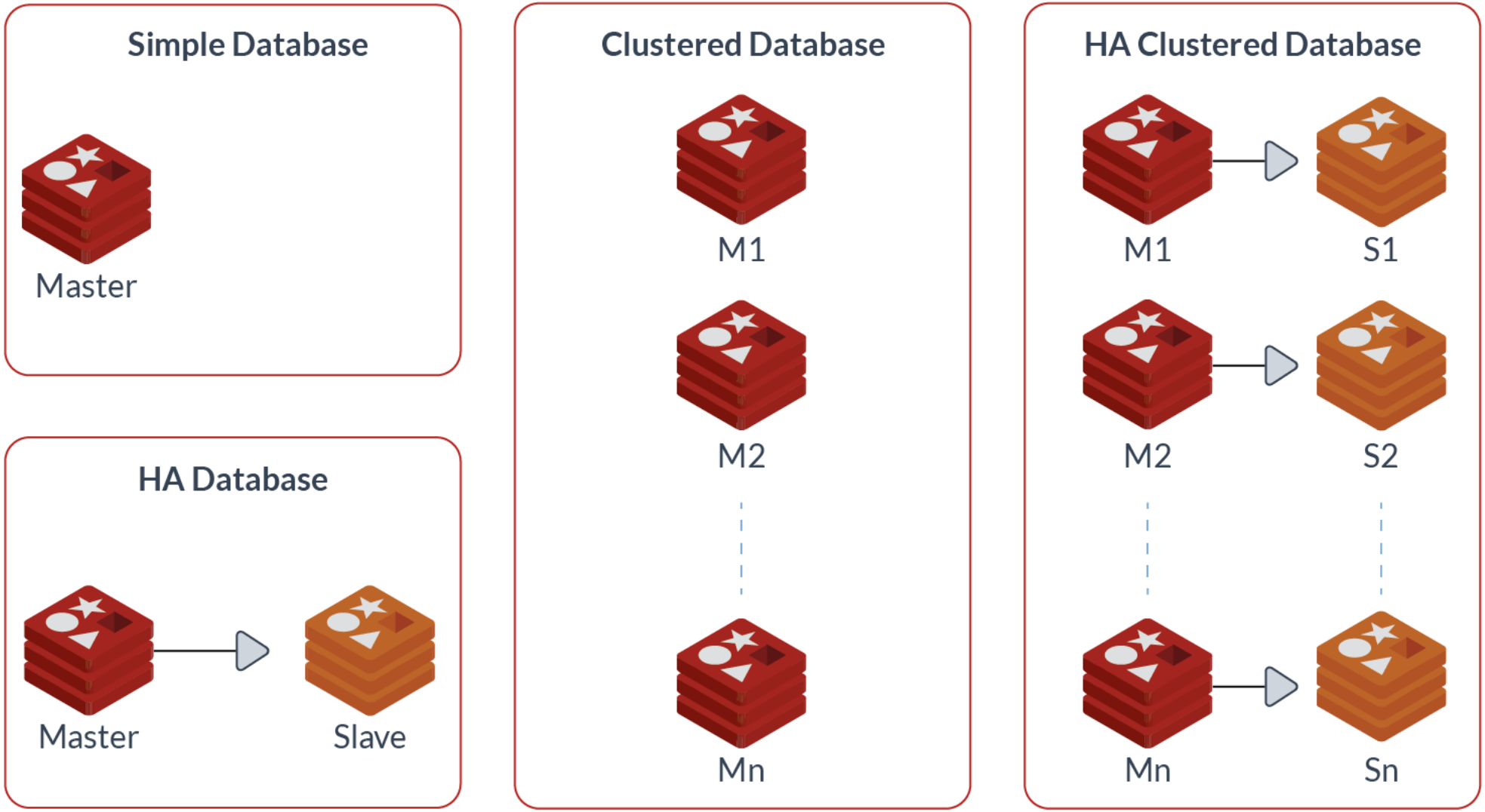


Redis : exemple d'utilisation de MULTI ET WATCH

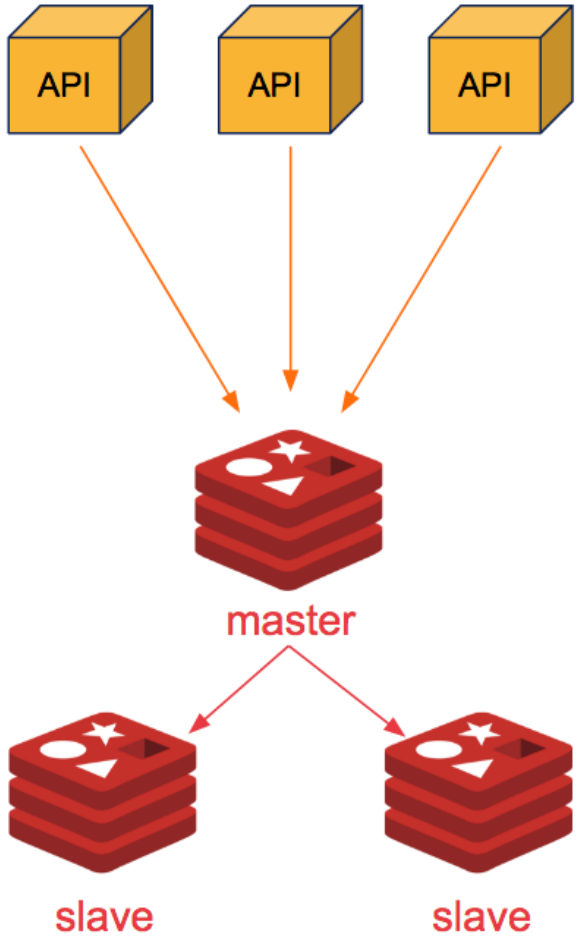


1. `sample_key` a pour valeur 111
2. Dans le Terminal 1 :
   a. Exécution de `WATCH`
   b. Exécution de `MULTI`
   c. Stockage dans la QUEUE de l'incrémentation de `sample_key` dont la valeur passerait à 211
3. Dans le Terminal 2 : exécution de l'incrémentation de `sample_key` dont la valeur passe à 112
4. Dans le Terminal 1 : Exécution de la commande `EXEC` qui crée (nil) car la valeur de 211 ne peut pas être stockée

Redis : Système de réplication basé sur différentes architectures maître-esclave



Redis : 1 master et 3 réplicas



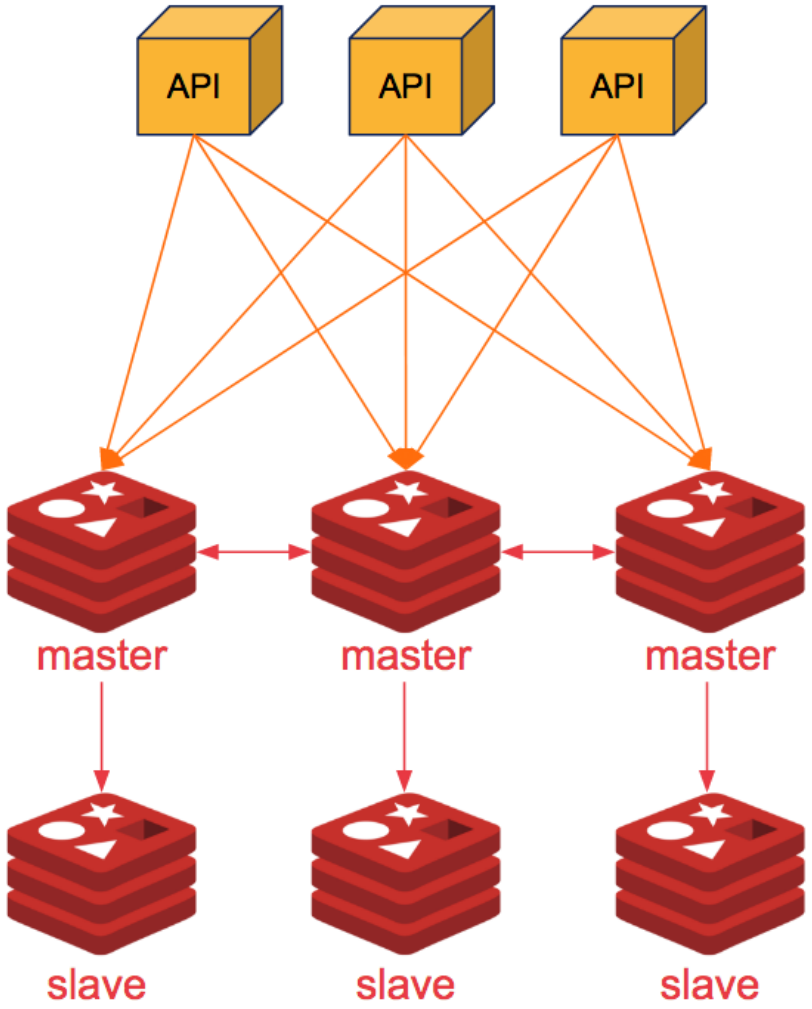
Redis : communication et synchronisation entre le maitre et ses esclaves

- Envoi d'un flux de commandes du maître aux esclaves pour répliquer les opérations effectuées sur le jeu de données du maître, quasiment en temps réel.
- En cas de rupture de la connexion entre maître et esclave est rompue ou de connexion trop longue : reconnexion de l'esclave, tentative de synchronisation partielle afin de ne récupérer que la partie du flux de commandes qu'il n'a pas pu obtenir lors de la coupure.
- En cas d'impossibilité de synchronisation partielle : demande de synchronisation complète par l'esclave demande une, une sauvegarde de toutes ses données par le maître, envoie à l'esclave, et reprise de l'envoi du flux de commande comme précédemment.

Redis : réplication, sauvegardes périodiques/incrémentales et système de surveillance Sentinel

- Réplication asynchrone et donc non bloquante pour le maître et dans la plupart des cas elle est non bloquante également pour les esclaves.
- Possibilité pour les esclaves d'avoir des réplications en cascade.
- Possibilité de combiner la réplication avec les sauvegardes périodiques (RDB) et les sauvegardes incrémentales (AOF).
- Recommandation de Redis : activer les deux modes de sauvegarde sur les maîtres et les esclaves afin d'assurer la meilleure reprise sur panne possible.
- Système Sentinel : pour surveiller les instances de Redis, et, en cas de panne d'un serveur maître, élire un nouveau serveur maître parmi les esclaves encore disponibles

Redis : Plusieurs maîtres

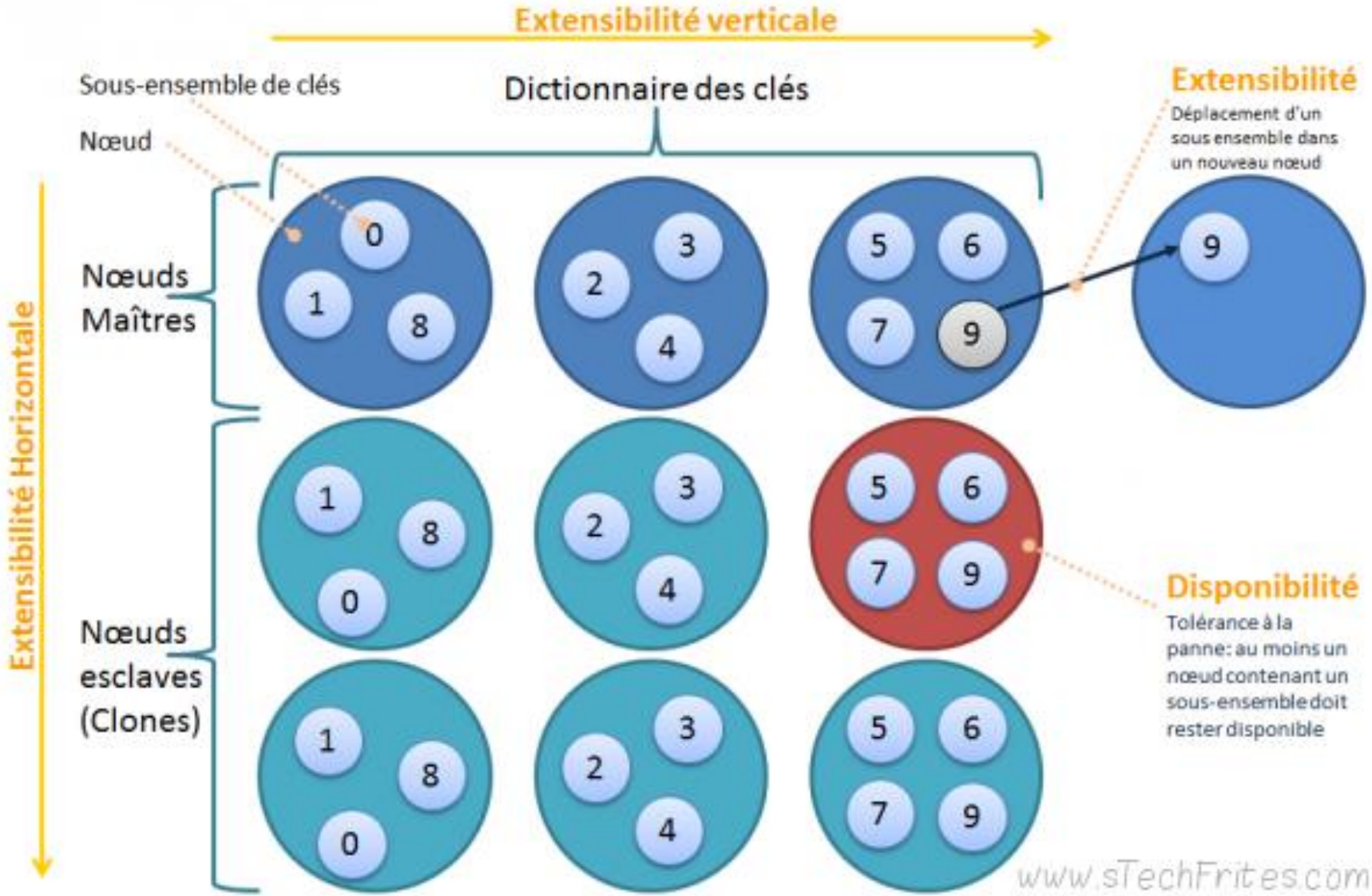


Redis : Partitionnement horizontal / vertical

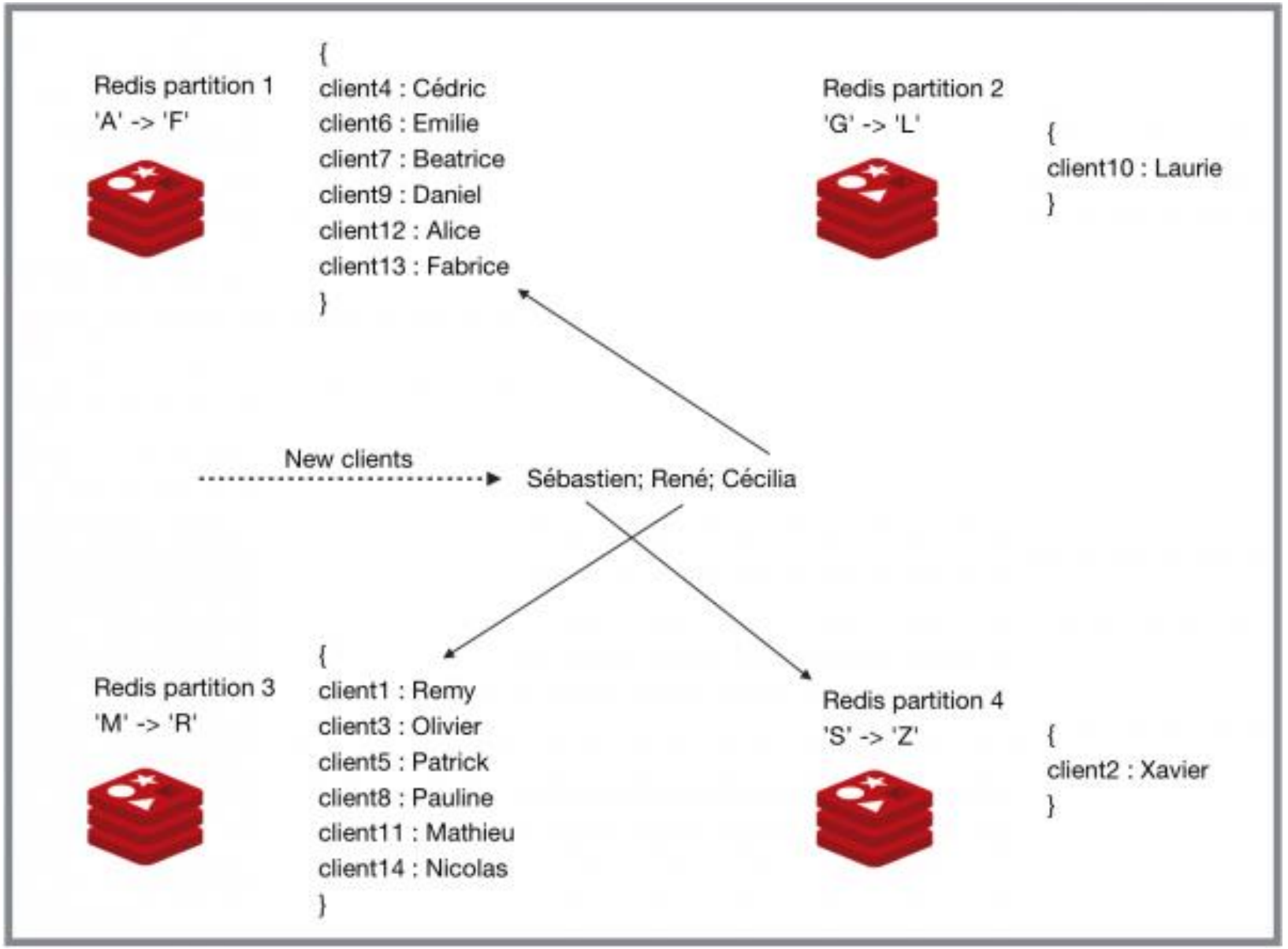
- Partitionnement horizontal : Distribution des clés à travers les différentes instances de Redis (aussi connu sous le nom de '*Sharding*') – la plus utilisée

- Partitionnement vertical : Distribution des valeurs des clés à travers les différentes instances de Redis

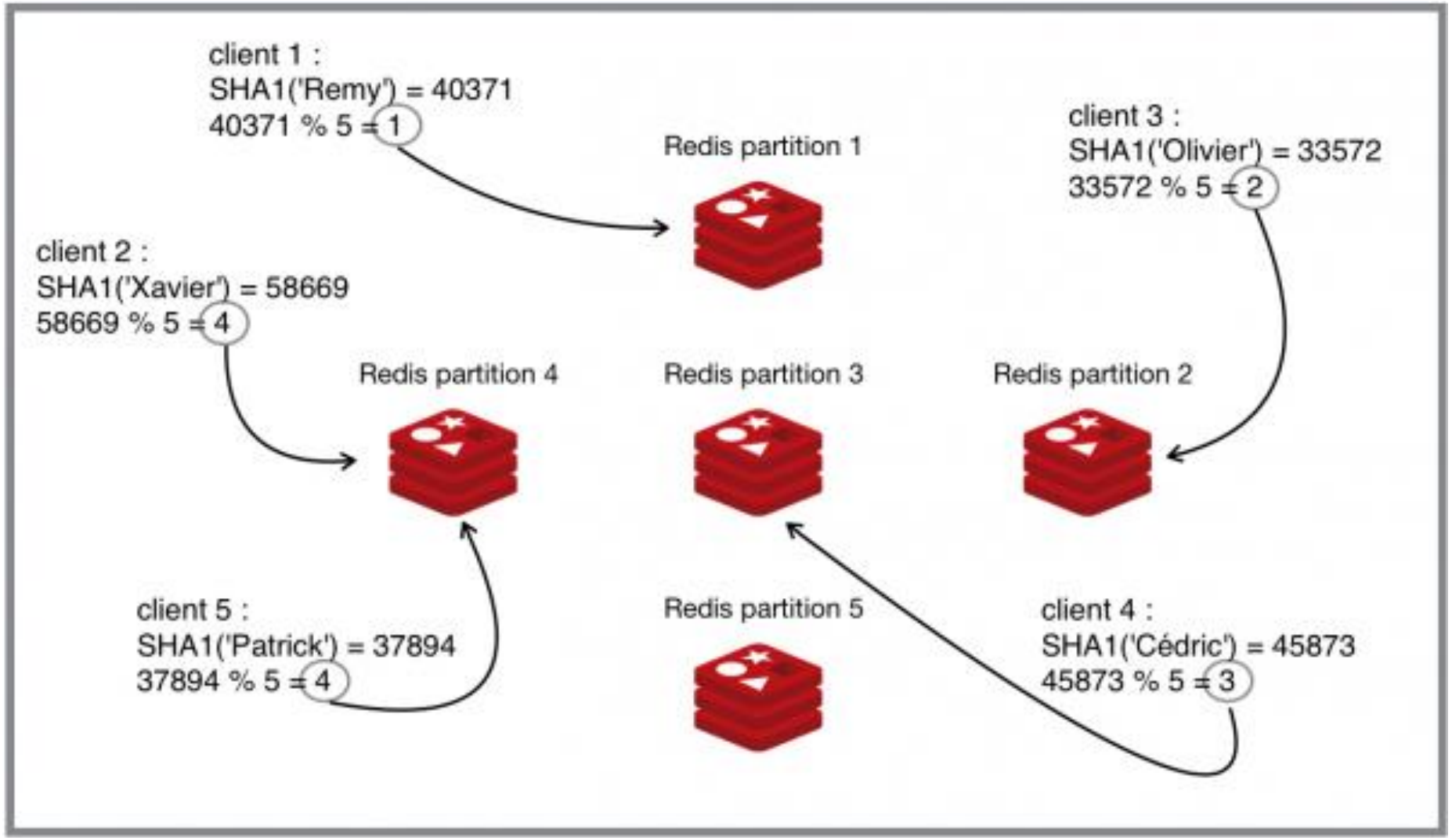
Redis : Partitionnement



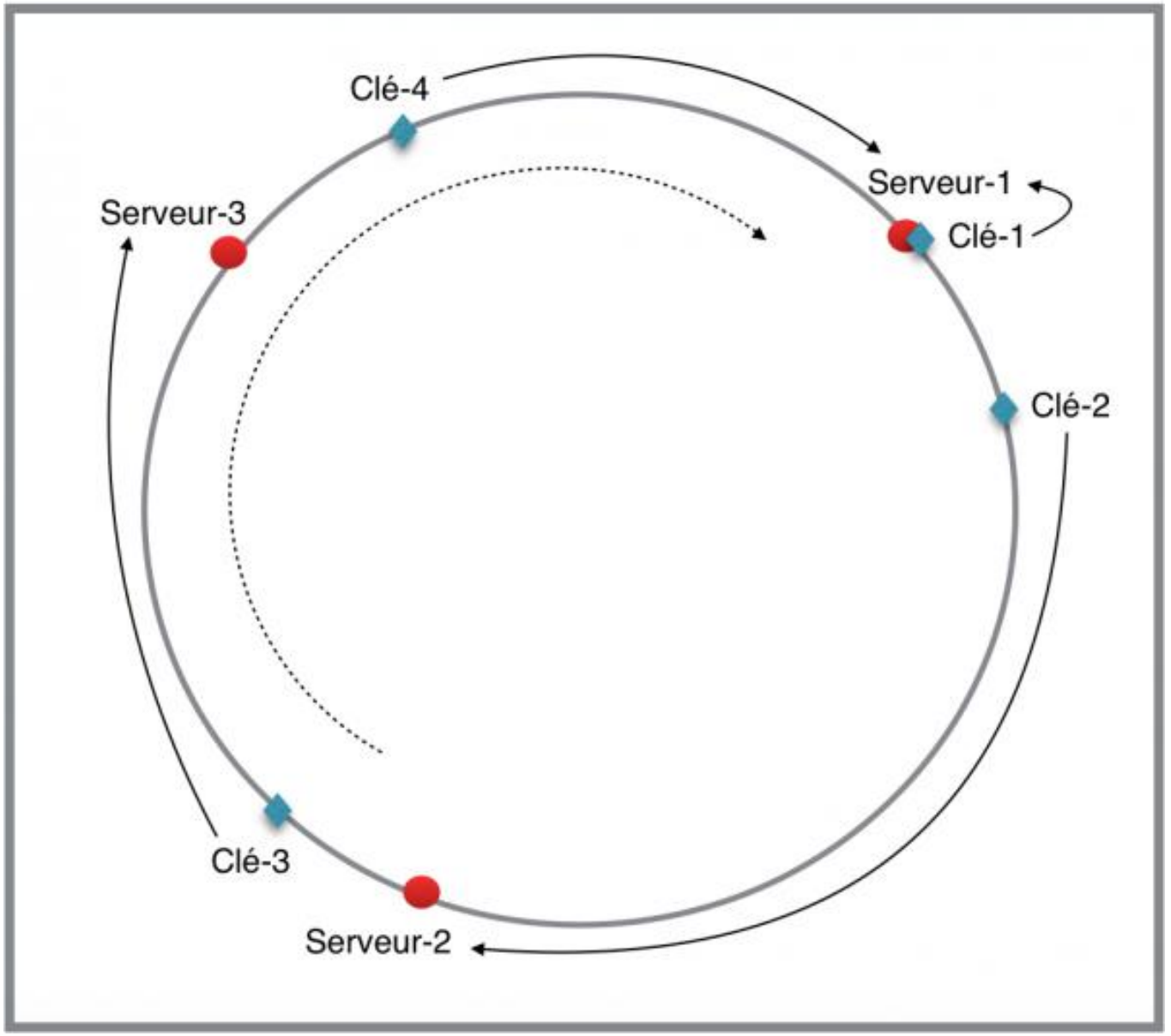
Redis : *Range Partitioning*



Redis : *Hash Partitioning*



Redis : *Consistent hashing*



Hachage cohérent (*Consistent hashing)* : pourquoi ?

- Hachage : basé sur une fonction *h()* qui distribue les valeurs de clé vers un intervalle *[0, n-1]*, *n* correspondant au nombre de fragments

modification de la fonction rend invalide la distribution existante

- **Hachage cohérent :** maintien de la même fonction tout en adaptant la règle d'affectation d'une donnée à un serveur selon l'évolution (ajout / suppression) de la grappe
- Principe du hachage cohérent : considérer dès le départ un intervalle immuable *D = [0, n-1]* pour le domaine d'arrivée de la fonction de hachage, où *n* est choisi assez grand pour réduire le nombre de collisions.

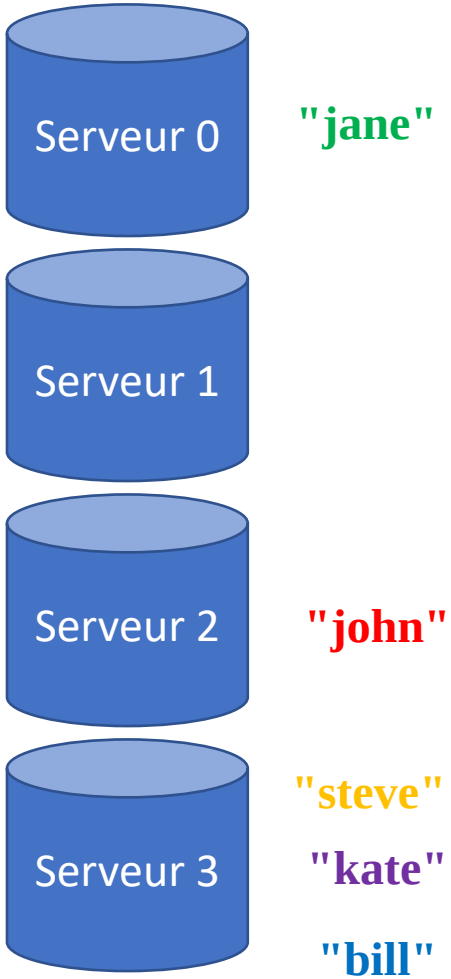
Hachage cohérent (*Consistent hashing)* : principe

- Objectif : répartir les données de manière la plus équitable possible
- Système réparti vu comme un anneau (*hash ring*), où chaque nœud occupe une position dans l'anneau et se voit attribuer une valeur de hachage (rangée de valeurs hébergées par le nœud)
- *Hash code* calculé à partir de la clé et associé au nœud dont la valeur de hachage est immédiatement supérieure
- Chaque nœud contient toutes les clés inférieures à sa valeur de hachage et supérieures à celle de nœud précédent

Hachage cohérent *versus* Hachage (1/8)

KEY	HASH	HASH mod 4
"john"	1633428562	2
"bill"	7594634739	3
"jane"	5000799124	0
"steve"	9787173343	3
"kate"	3421657995	3

HASH mod 5 renvoie le numéro de la machine qui va stocker la donnée

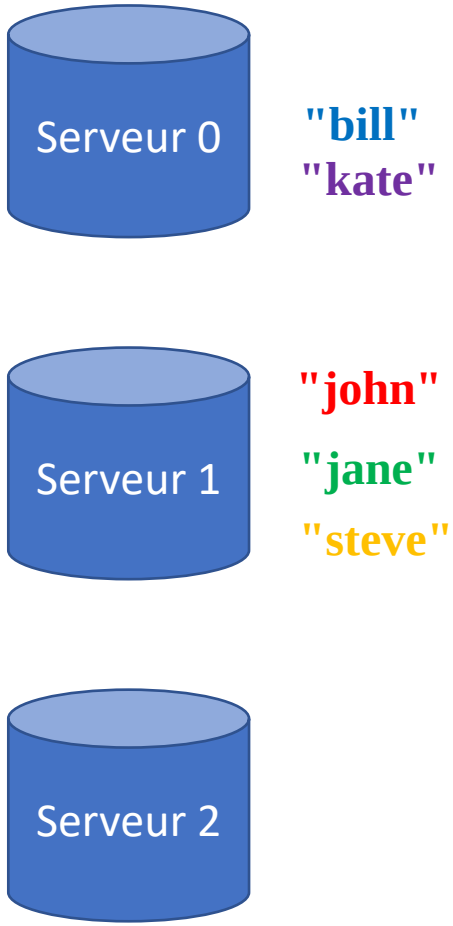


Hachage cohérent *versus* Hachage (2/8)

Suppression d'une machine

KEY	HASH	HASH mod 3
"john"	1633428562	1
"bill"	7594634739	0
"jane"	5000799124	1
"steve"	9787173343	1
"kate"	3421657995	0

La suppression (suite à une panne par exemple) d'une machine impose de refaire complétement la partition des données

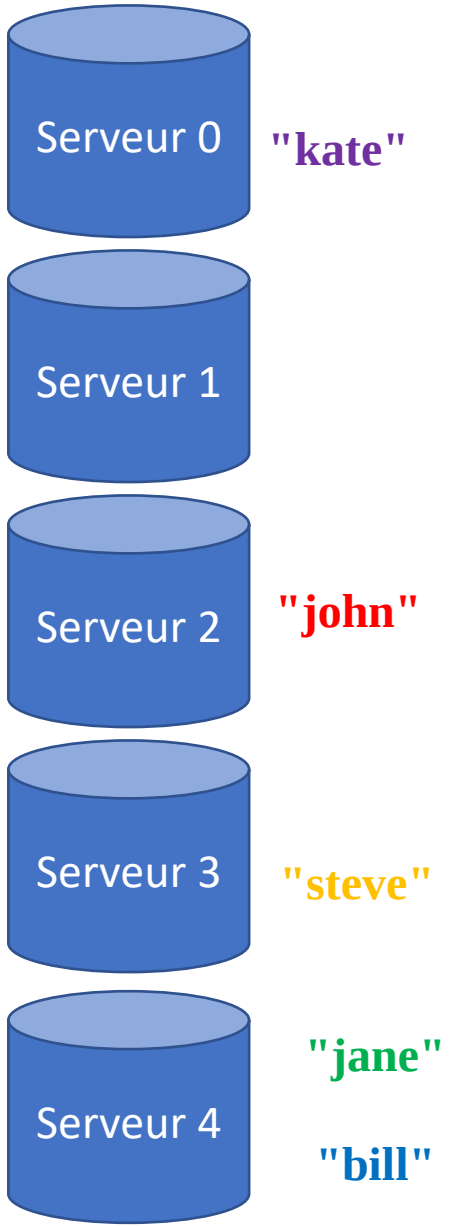


Hachage cohérent *versus* Hachage (3/8)

Ajout d'une machine (en repartant du transparent 1)

KEY	HASH	HASH mod 5
"john"	1633428562	2
"bill"	7594634739	4
"jane"	5000799124	4
"steve"	9787173343	3
"kate"	3421657995	0

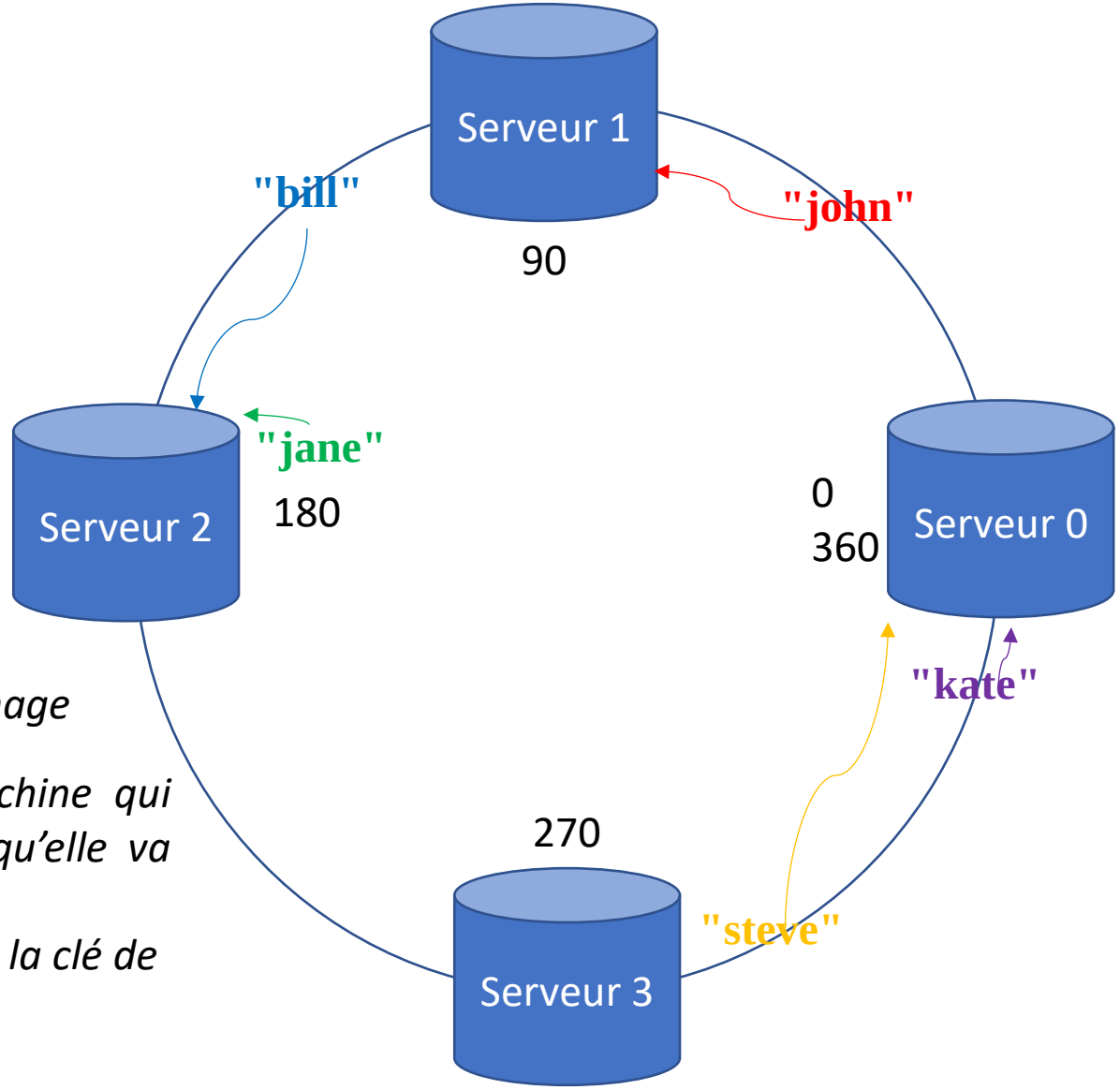
L'ajout d'une machine (pour alléger la charge d'une machine) impose de refaire complétement la partition des données



Hachage cohérent *versus* Hachage (4/8)

Hachage cohérent

KEY	HASH	ANGLE (DEG)
"john"	1633428562	58.7
"bill"	7594634739	123.1
"jane"	5000799124	180
"steve"	9787173343	273.4
"kate"	3421657995	352.3



On associe un angle sur l'anneau à chaque clé de hachage

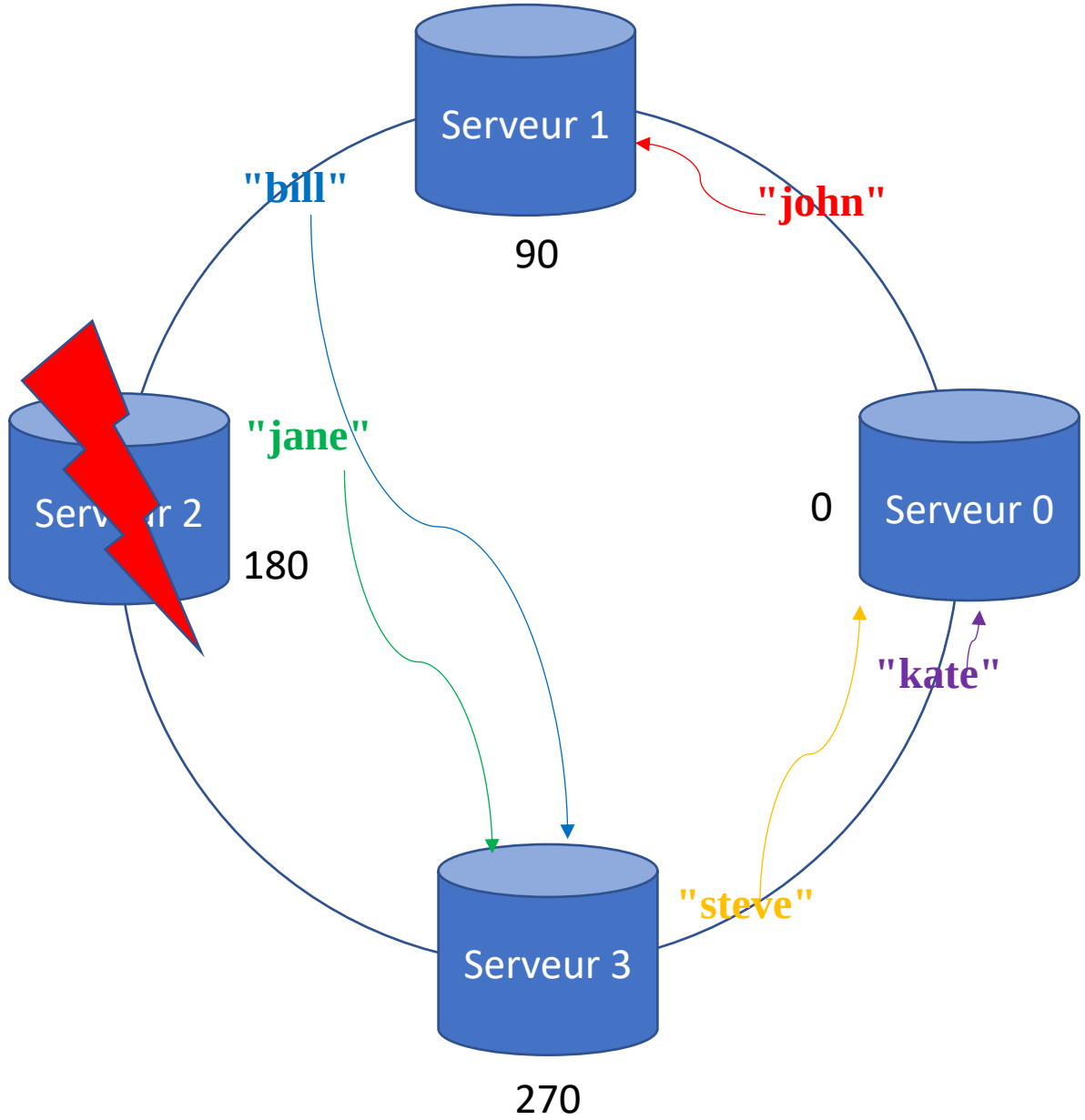
Une valeur de hachage est associée à chaque machine qui détermine l'espace des valeurs de clé de hachage qu'elle va gérer.
Par exemple, le serveur 1 gère toutes les données dont la clé de hachage appartient à]0,90], le serveur 0 tous les clés appartenant]270,360]

Hachage cohérent *versus* Hachage (5/8)

Suppression d'une machine

KEY	HASH	ANGLE (DEG)
"john"	1633428562	58.7
"bill"	7594634739	123.1
"jane"	5000799124	180
"steve"	9787173343	273.4
"kate"	3421657995	352.3

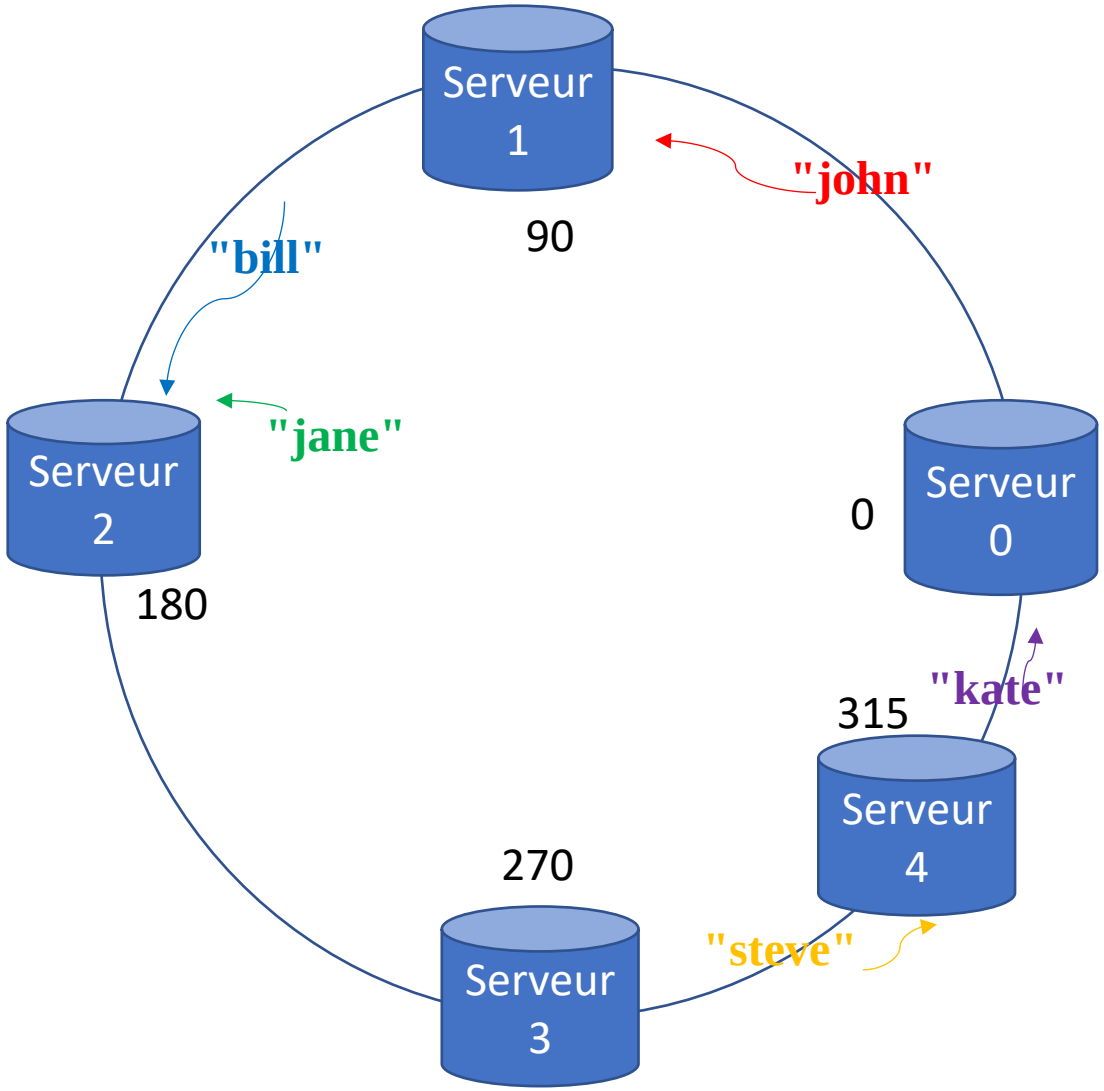
La suppression (suite à une panne par exemple) d'une machine ne modifie que la répartition d'un sous-ensemble des données (ceux de la machine en panne).



Hachage cohérent *versus* Hachage (6/8)

Ajout d'une machine

KEY	HASH	ANGLE (DEG)
"john"	1633428562	58.7
"bill"	7594634739	123.1
"jane"	5000799124	180
"steve"	9787173343	273.4
"kate"	3421657995	352.3



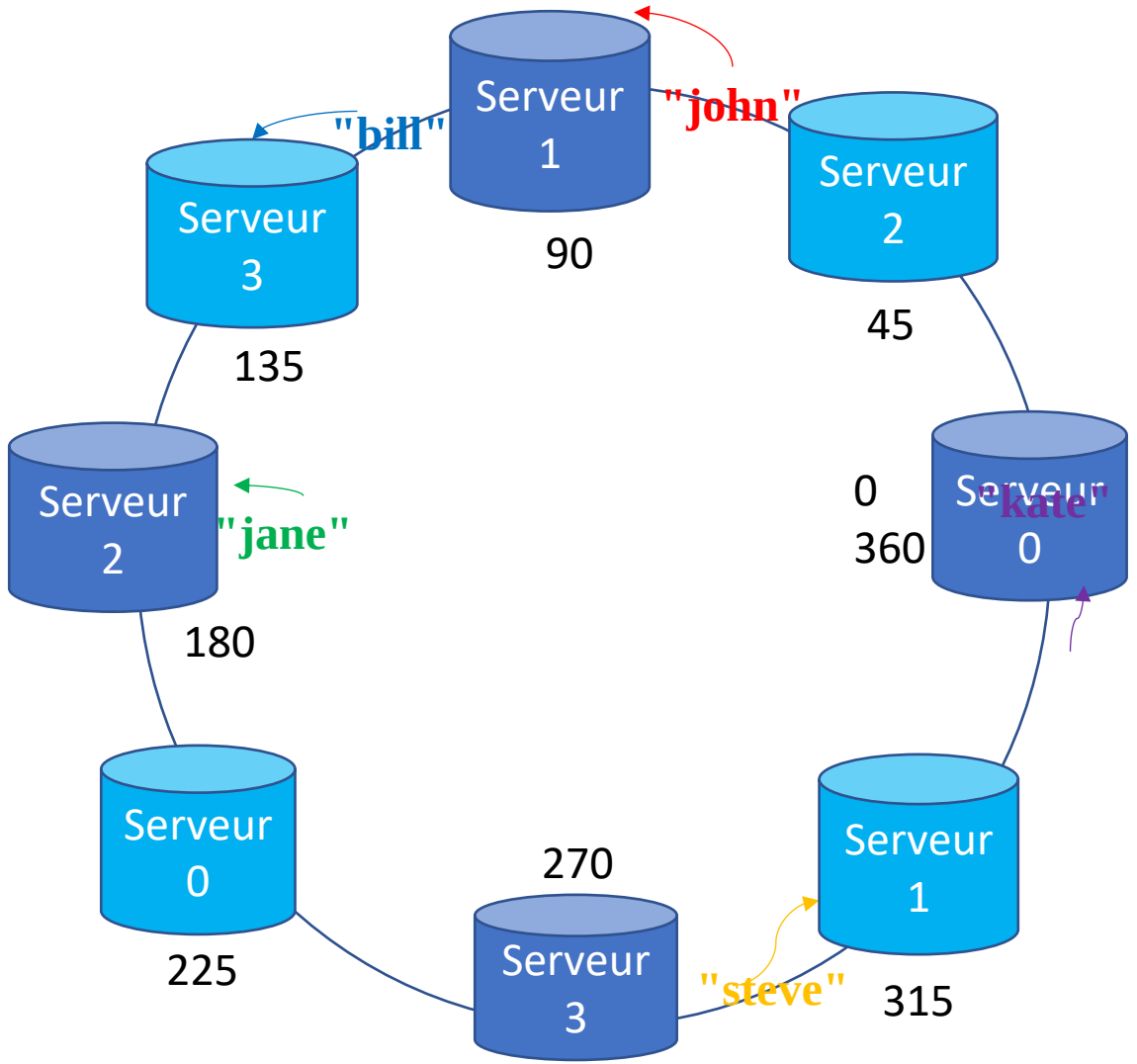
L'ajout d'une machine (pour alléger la charge d'une machine) ne touche que la répartition des données des machines voisines au nouveau nœud.

Hachage cohérent *versus* Hachage (7/8)

Hachage cohérent avec nœuds virtuels

KEY	HASH	ANGLE (DEG)
"john"	1633428562	58.7
"bill"	7594634739	123.1
"jane"	5000799124	180
"steve"	9787173343	273.4
"kate"	3421657995	352.3

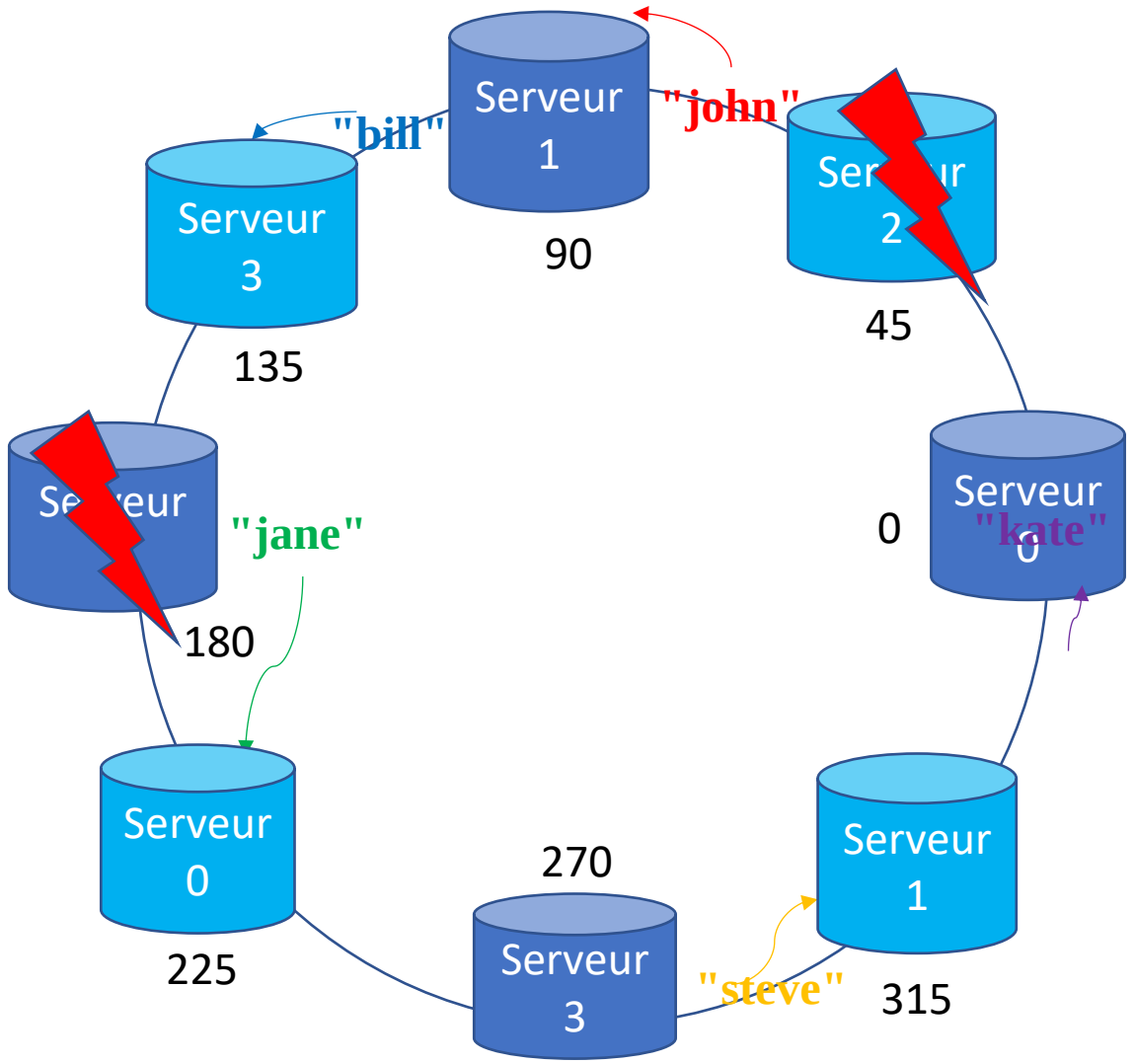
Une même machine apparait plusieurs fois sur l'anneau, donc chaque machine va gérer plusieurs intervalles de valeurs.
Ex. le serveur 1 gère toutes les données dont la clé de hachage appartient à]45,90] et]270,315]



Hachage cohérent *versus* Hachage (8/8)

Hachage cohérent avec nœuds virtuels

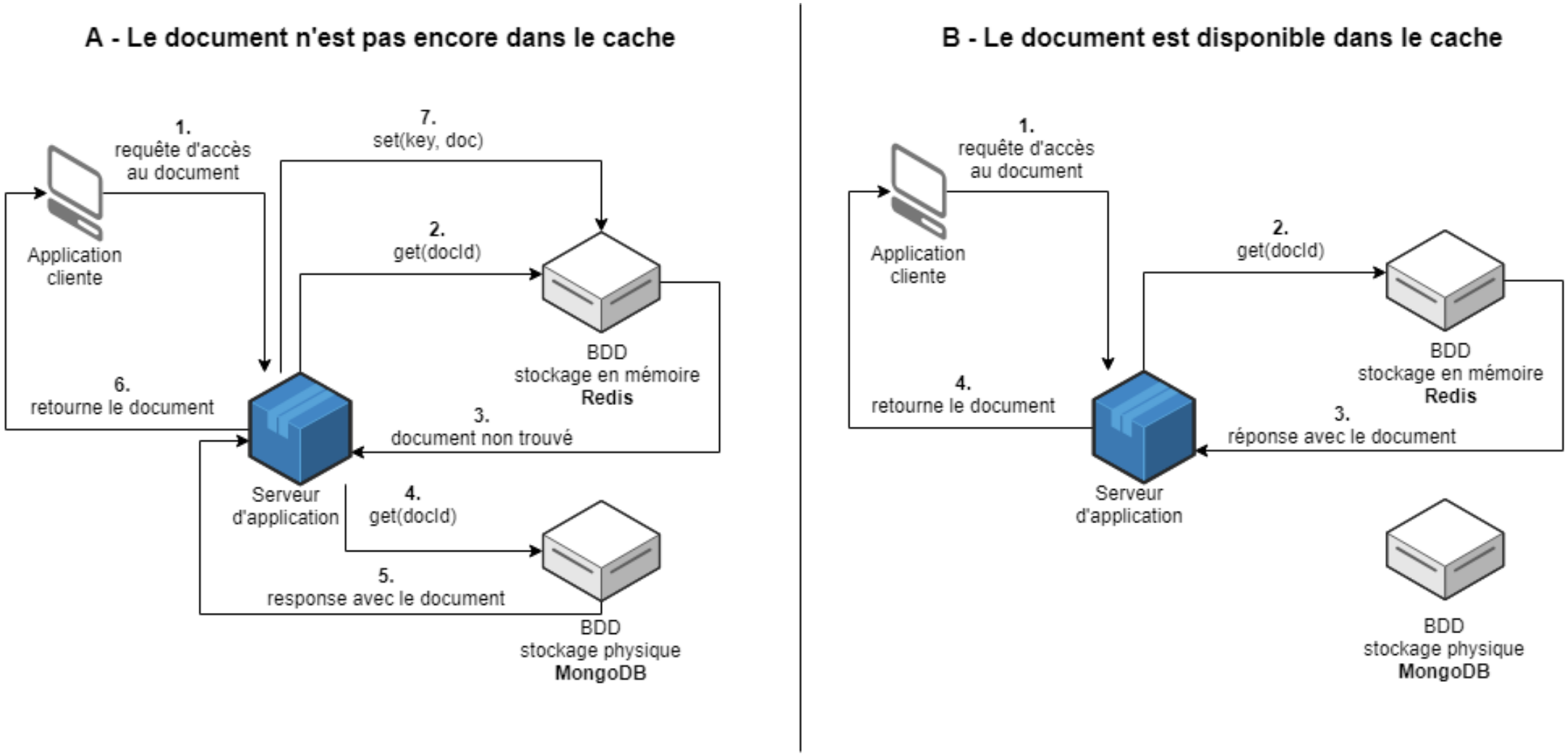
KEY	HASH	ANGLE (DEG)
"john"	1633428562	58.7
"bill"	7594634739	123.1
"jane"	5000799124	180
"steve"	9787173343	273.4
"kate"	3421657995	352.3



La suppression (suite à une panne par exemple) d'une machine ne modifie que la répartition des sous-ensembles des données (ceux de la machine en panne).

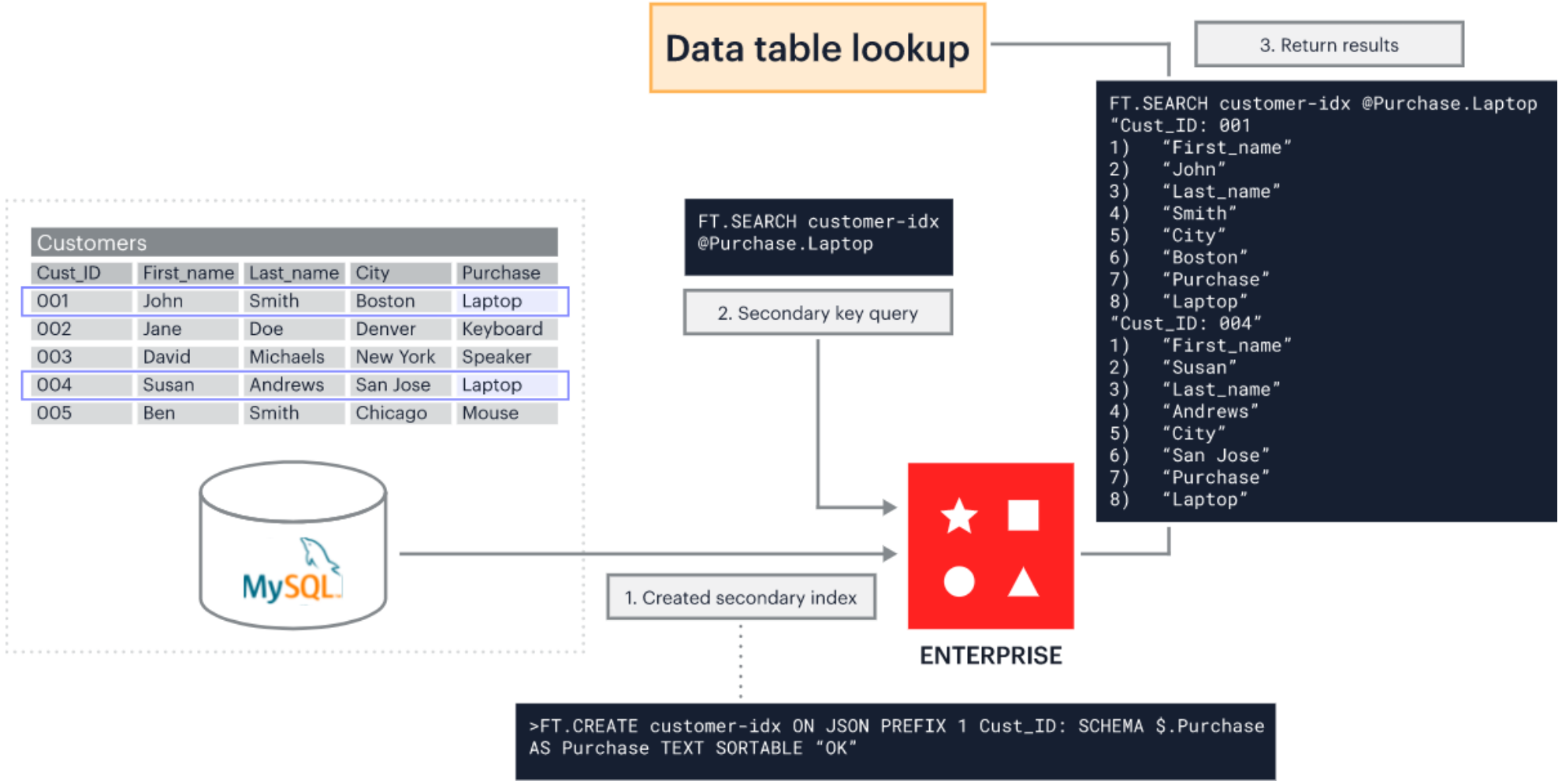
De même lors de l'ajout d'une machine, seuls des sous-ensembles de données sont impactés.

Redis : exemple d'utilisation comme système de cache



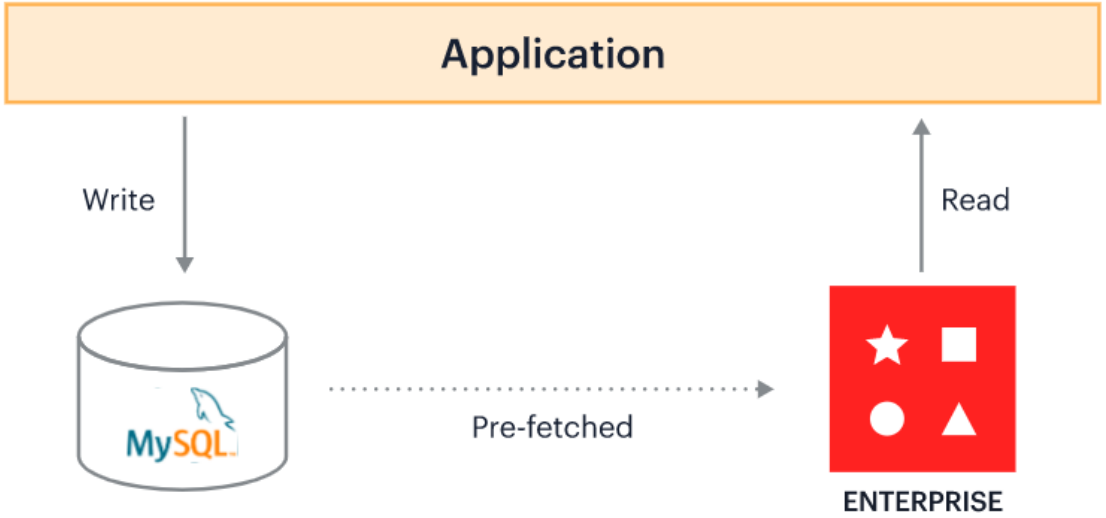
Redis : exemple d'utilisation avec un SGBD relationnel

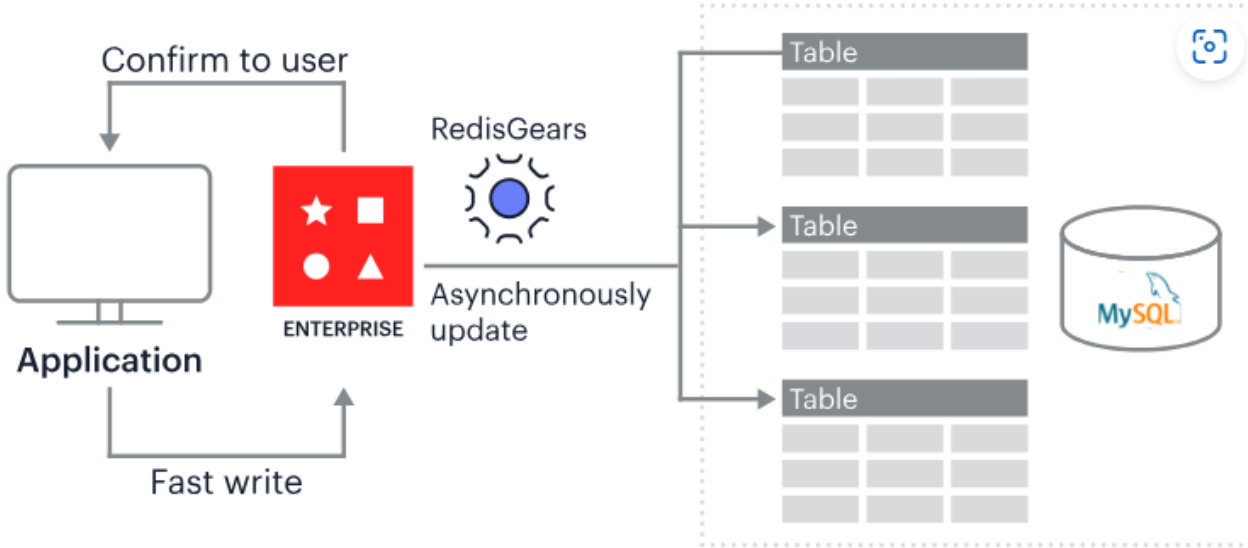
Utilisation de Redis comme index secondaire



Redis : exemple d'utilisation avec un SGBD relationnel

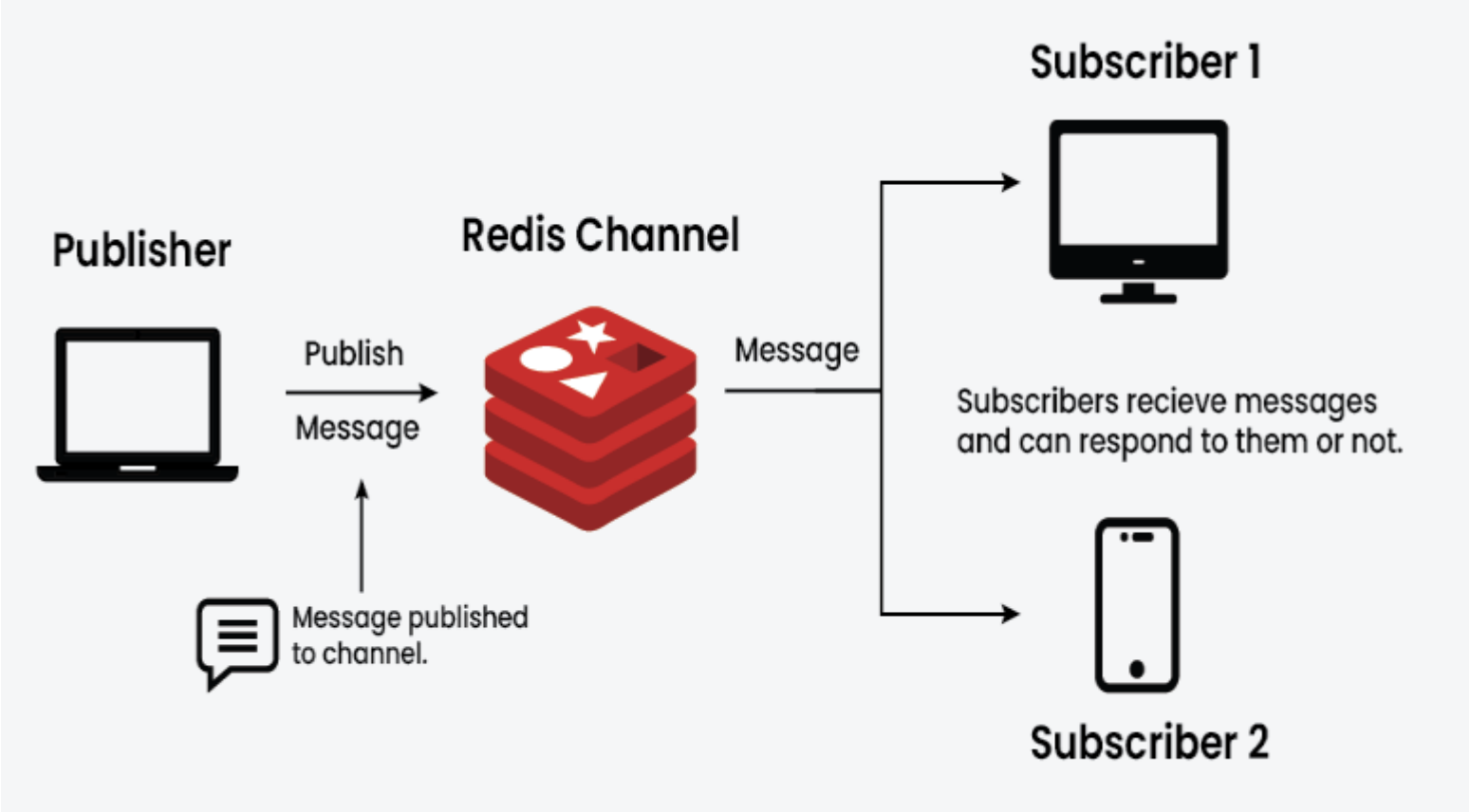
Utilisation de Redis comme cache





Redis : publication/abonnement

Pour implémenter des architectures de publication/abonnement ou de file d'attente cf. https://www.geeksforgeeks.org/redis-publish-subscribe/



Redis : conclusion


Accessibilité et performance


Types de données avancées


Sécuriser la persistance des données


Gestion d'écritures concurrentes


Sécurisation relative des échanges


Gestion des erreurs


pas une solution adaptée pour stocker les données de manière durable

Clé-valeur : dans un moteur SQL

- HSTORE sous PostgreSQL
  (cf. doc https://www.postgresql.org/docs/current/hstore.html)

```
1  CREATE EXTENSION hstore;
2  CREATE TABLE mytable (h hstore);
3
4  INSERT INTO mytable VALUES ('key1=>value1, key2=>value2');
5
6  SELECT * FROM mytable;
```

	h hstore
1	"key1"=>"value1", "key2"=>"value2"

```
8  SELECT h->'key1' AS valeur FROM mytable;
9
```

Data Output | Explain | Messages | History

	valeur text
1	value1

Query | Query History

```
1  SELECT h['key1'] FROM mytable;
```

	h text
1	value1

Conversion d'une relation en ensemble de clé-valeur

```
CREATE EXTENSION HSTORE;

CREATE TABLE test (col1 integer, col2 text, col3 text);

INSERT INTO test VALUES (123, 'foo', 'bar');
```

```
SELECT * FROM test;
```

col1	col2	col3
123	foo	bar

Transformation de la relation en couple (clé, valeur) :

```
SELECT hstore(t) FROM test AS t;
```

hstore
"col1"=>"123", "col2"=>"foo", "col3"=>"bar"

Exemple d'attributs de type clé-valeur sous PostgreSQL

```sql
CREATE TABLE books (
    id serial primary key,
    title VARCHAR (255),
    attr hstore
);

INSERT INTO books (title, attr) VALUES
    (   'PostgreSQL Tutorial',
        '"paperback" => "243",
        "publisher" => "postgresqltutorial.com",
        "language"  => "English",
        "ISBN-13"   => "978-1449370000",
        "weight"    => "11.2 ounces"'
    );
```

```sql
INSERT INTO books (title, attr) VALUES
    (
        'PostgreSQL Cheat Sheet',
        '"paperback" => "5",
        "publisher" => "postgresqltutorial.com",
        "language"  => "English",
        "ISBN-13"   => "978-1449370001",
        "weight"    => "1 ounces"'
    );
```

Requête sur un attribut de type clé-valeur sous PostgreSQL

```
1  SELECT attr FROM books;
```

	attr hstore
1	"weight"=>"11.2 ounces", "ISBN-13"=>"978-1449370000", "language"=>"English", "paperback"=>"243", "publisher"=>"postgresqltutorial.com"
2	"weight"=>"1 ounces", "ISBN-13"=>"978-1449370001", "language"=>"English", "paperback"=>"5", "publisher"=>"postgresqltutorial.com"

```
1  SELECT attr -> 'ISBN-13' AS isbn
2   FROM books;
```

	isbn text
1	978-1449370000
2	978-1449370001

Requête sur la valeur d'une clé particulière sous PostgreSQL

```
1  SELECT title, attr -> 'weight' AS weight
2   FROM books
3    WHERE attr -> 'ISBN-13' = '978-1449370000';
```

	title character varying (255)	weight text
1	PostgreSQL Tutorial	11.2 ounces

```
1  SELECT title, attr -> 'publisher' AS weight
2   FROM books
3    WHERE attr -> 'publisher' LIKE 'postgres%';
```

	title character varying (255)	weight text
1	PostgreSQL Tutorial	postgresqltutorial.com
2	PostgreSQL Cheat Sheet	postgresqltutorial.com

Ajout d'une paire clé-valeur sous PostgreSQL

```
1  UPDATE books
2    SET attr = attr || '"freeshipping"=>"yes"' :: hstore;
3
4  SELECT title, attr -> 'freeshipping' AS freeshipping
5   FROM books;
```

	title character varying (255)	freeshipping text
1	PostgreSQL Tutorial	yes
2	PostgreSQL Cheat Sheet	yes

```
1  UPDATE books
2  SET attr = delete(attr, 'freeshipping');
3
4  SELECT title, attr -> 'freeshipping' AS freeshipping
5   FROM books;
```

	title character varying (255)	freeshipping text
1	PostgreSQL Tutorial	[null]
2	PostgreSQL Cheat Sheet	[null]

Test d'existence d'une clé PostgreSQL

```
1  SELECT
2    title,
3    attr->'publisher' as publisher,
4    attr
5   FROM books
6    WHERE attr ? 'publisher';
```

	title character varying (255)	publisher text	attr hstore
1	PostgreSQL Tutorial	postgresqltutorial.com	"weight"=>"11.2 ounces", "ISBN-13"=>"978-14493...
2	PostgreSQL Cheat Sheet	postgresqltutorial.com	"weight"=>"1 ounces", "ISBN-13"=>"978-14493700...

```
1  SELECT title, attr->'publisher' as publisher, attr
2   FROM books
3    WHERE attr ? 'author';
```

	title character varying (255)	publisher text	attr hstore

Test sur des paires clé-valeur PostgreSQL

```
1  SELECT title
2   FROM books
3    WHERE attr @> '"weight"=>"11.2 ounces"' :: hstore;
```

	title character varying (255)
1	PostgreSQL Tutorial

```
1  SELECT title
2    FROM books
3     WHERE attr ?& ARRAY [ 'language', 'weight' ];
```

	title character varying (255)
1	PostgreSQL Tutorial
2	PostgreSQL Cheat Sheet

```
1  SELECT akeys (attr)
2    FROM books;
```

	akeys text[]
1	weight,ISBN-13,language,paperback,publisher
2	weight,ISBN-13,language,paperback,publisher

Conversion des paires clé-valeur PostgreSQL

```
1  SELECT title, hstore_to_json (attr) json
2  FROM books;
```

	title character varying (255)	json json
1	PostgreSQL Tutorial	{"language":"English","paperback":"243","ISBN-13":"978-1449370000","weight":"11.2 ounces","publisher":"postgresqltutorial.com...
2	PostgreSQL Cheat Sheet	{"language":"English","paperback":"5","ISBN-13":"978-1449370001","weight":"1 ounces","publisher":"postgresqltutorial.com"}

```
1  SELECT title, (EACH(attr) ).*
2   FROM books;
```

	title character varying (255)	key text	value text
1	PostgreSQL Tutorial	weight	11.2 ounces
2	PostgreSQL Tutorial	ISBN-13	978-1449370000
3	PostgreSQL Tutorial	language	English
4	PostgreSQL Tutorial	paperba...	243
5	PostgreSQL Tutorial	publisher	postgresqltutorial.com
6	PostgreSQL Cheat Sheet	weight	1 ounces
7	PostgreSQL Cheat Sheet	ISBN-13	978-1449370001
8	PostgreSQL Cheat Sheet	language	English
9	PostgreSQL Cheat Sheet	paperba...	5
10	PostgreSQL Cheat Sheet	publisher	postgresqltutorial.com

Hstore sous PostgreSQL : conclusion


Combinaison relationnel et clé-valeur


Gestion des transactions


Persistance des données


Pas fait pour gérer les données en mémoire


Pas les mêmes types de données et le même usage

Performance par rapport à Redis ?

cf. ex. de résultats expérimentations contradictoires https://www.cybertec-postgresql.com/en/postgresql-vs-redis-vs-memcached-performance/, https://www.peterbe.com/plog/redis-vs-postgres-blob-of-json **et** https://profil-software.com/blog/development/database-comparison-sql-vs-nosql-mysql-vs-postgresql-vs-redis-vs-mongodb/

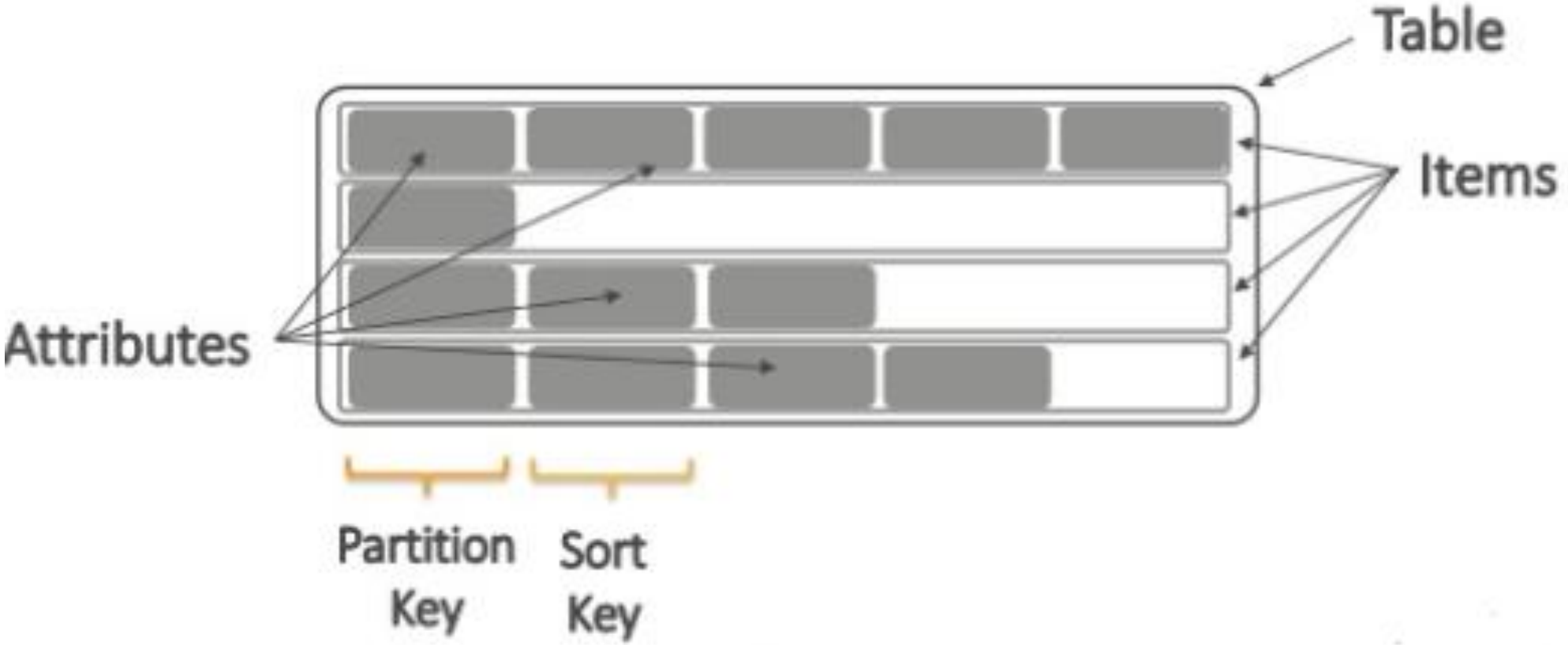
DynamoDB

- Créée par Amazon en 2004
- Implémentation des modèles clés-valeurs et documents, en mode totalement distribué
- Article de recherche publié en 2007, « **Dynamo Paper** », qui a inspiré de nombreux moteur NoSQL (ex. Riak, Cassandra ou Voldemort) : https://www.allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf
- **Amazon DB** (2012) : base de données NoSQL sur Amazon Web Services
- Accès gratuit jusqu'à 100 Mo de stockage et une capacité en I/O de 5 écritures et 10 lectures par seconde – cf. https://www.lemondeinformatique.fr/actualites/lire-dynamodb-une-base-de-donnees-nosql-sur-amazon-web-services-47471.html
- Utilisée par Netflix, SnapShat, Nike etc.

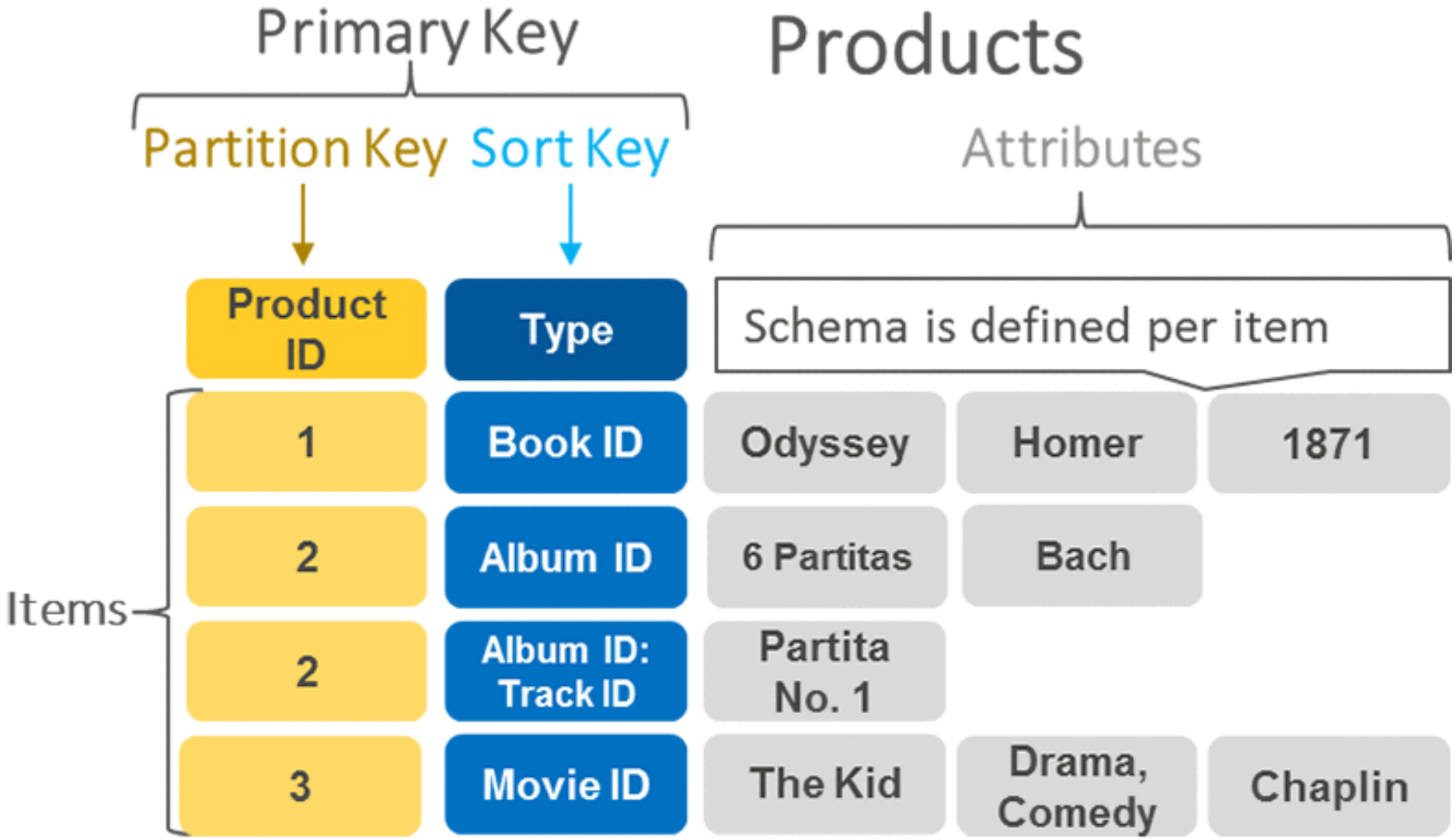
DynamoDB : liens utiles

- Site officiel : https://aws.amazon.com/fr/dynamodb/
- Doc AmazonDB : https://docs.aws.amazon.com/fr_fr/amazondynamodb/latest/developerguide/Introduction.html
- Article annonçant DynamoDB en 2012 : https://www.allthingsdistributed.com/2012/06/amazon-dynamodb-growth.html
- Transparents en ligne de Nicolas Travers, inspiré de Advait Deo : http://chewbii.com/wp-content/uploads/2016/10/handout_Dynamo.pdf

DynamoDB : modèle de données



DynamoDB : exemple



DynamoDB : exemple de table

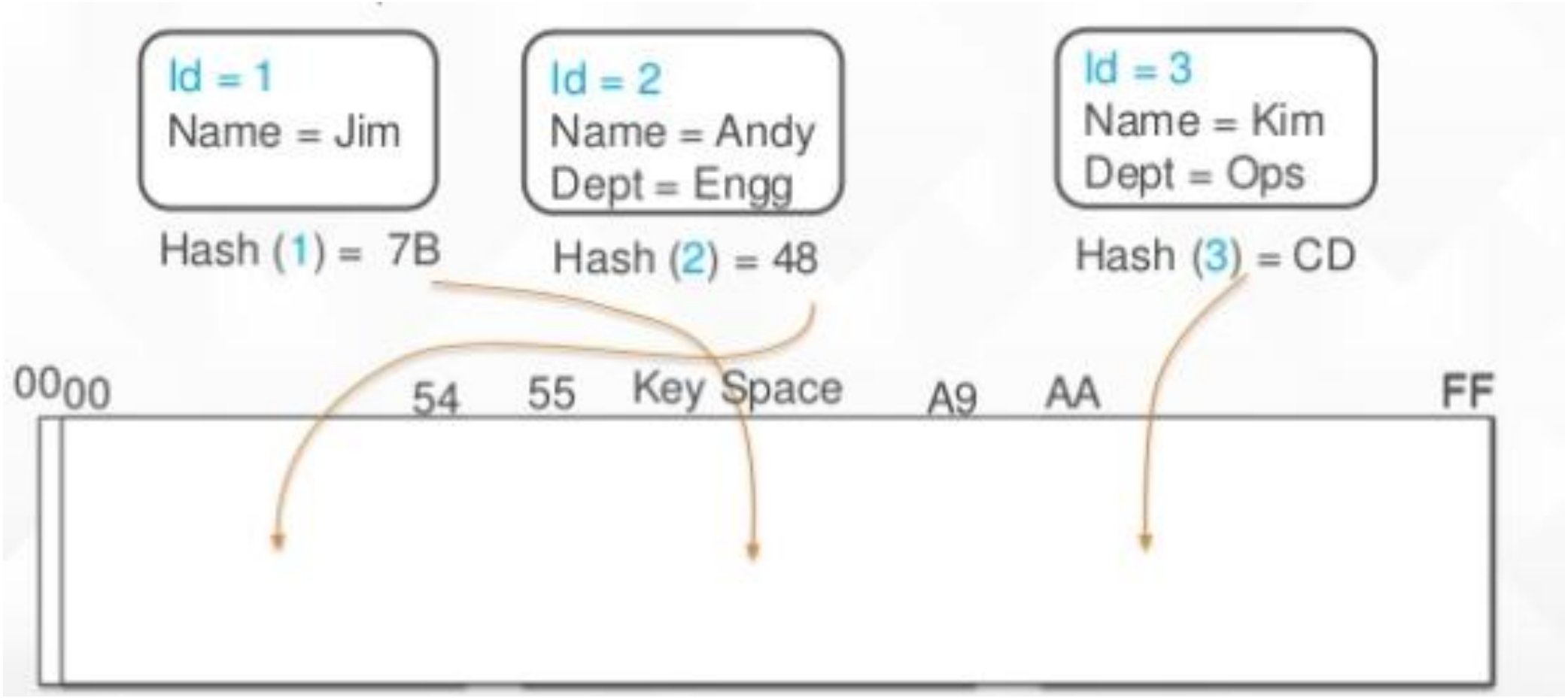
Primary Key		Attributes		
Actor (PARTITION)	**Movie (SORT)**			
Tom Hanks	Cast Away	**Role**	**Year**	**Genre**
		Chuck Noland	2000	Drama
	Toy Story	**Role**	**Year**	**Genre**
		Woody	1995	Children's
Tim Allen	Toy Story	**Role**	**Year**	**Genre**
		Buzz Lightyear	1995	Children's
Natalie Portman	Black Swan	**Role**	**Year**	**Genre**
		Nina Sayers	2010	Drama

DynamoDB : clé de partition / clé de tri

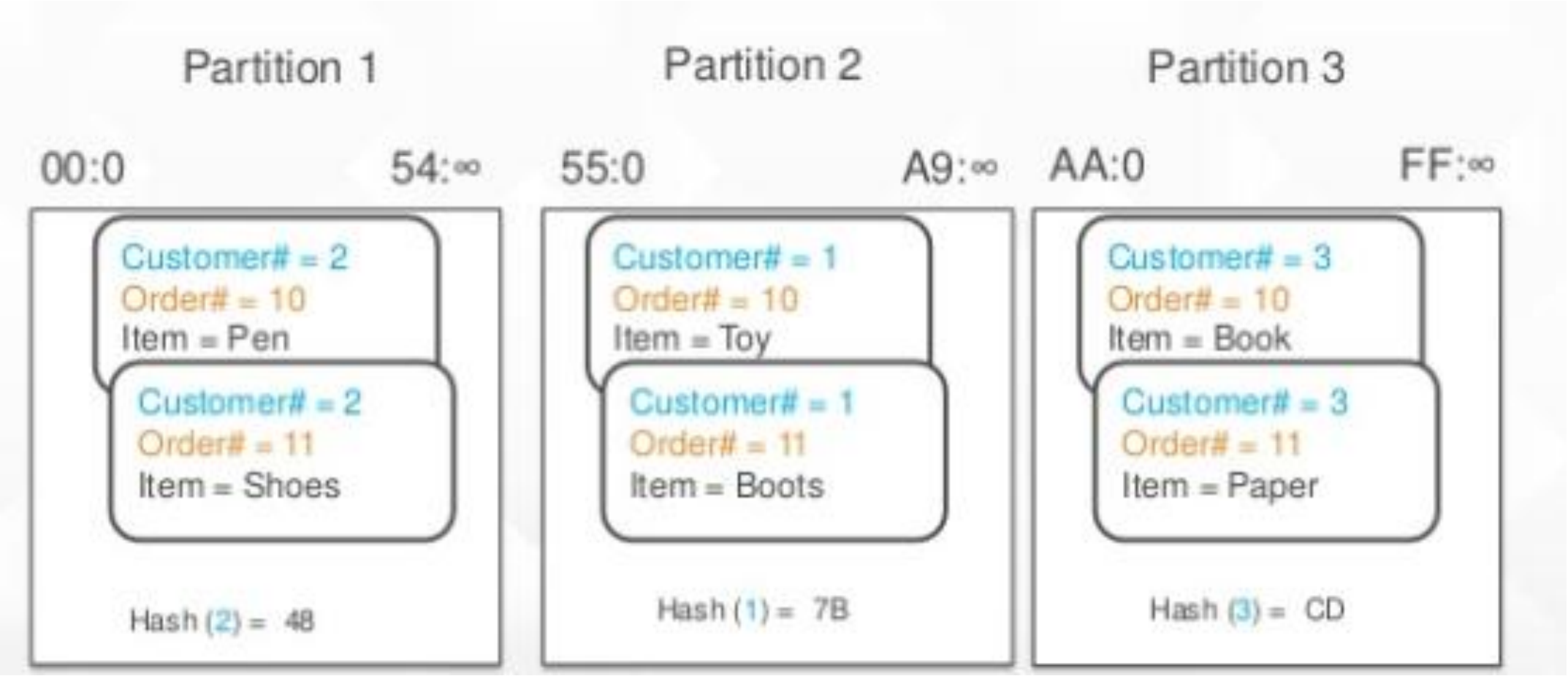
2 différents types de clés primaires :

- **Clé de partition** : clé primaire simple, composée d'un attribut
- **Clé de partition et clé de tri** : clé primaire composite, composée de deux attributs
- Valeur de la clé de partition utilisée comme entrée d'une fonction de hachage interne : *attribut de hachage*
- Chaque attribut de clé primaire doit être un scalaire et de type *string*, *number* ou *binary*

DynamoDB : exemple de clé de partition



DynamoDB : exemple de clé de partition et de clé de tri



DynamoDB : principaux composants

- **Table** : ensemble d'éléments / données
- **Elément** : ensemble d'**attributs** (généralement scalaires)
- **Identifiant** unique pour chaque élément de la table
- Table **sans schéma**: pas de définition préalable des attributs ni des types de données
- Possibilité d'attributs imbriqués, jusqu'à 32 niveaux

DynamoDB : identificateur à plusieurs attributs

- Possibilité d'avoir des identificateurs à plusieurs attributs

```
{
    "Artist": "No One You Know",
    "SongTitle": "My Dog Spot",
    "AlbumTitle": "Hey Now",
    "Price": 1.98,
    "Genre": "Country",
    "CriticRating": 8.4
}
```

```
{
    "Artist": "No One You Know",
    "SongTitle": "Somewhere Down The Road",
    "AlbumTitle": "Somewhat Famous",
    "Genre": "Country",
    "CriticRating": 8.4,
    "Year": 1984
}
```

```
{
    "Artist": "The Acme Band",
    "SongTitle": "Still in Love",
    "AlbumTitle": "The Buck Starts Here",
    "Price": 2.47,
    "Genre": "Rock",
    "PromotionInfo": {
        "RadioStationsPlaying": [
            "KHCR",
            "KQBX",
            "WTNR",
            "WJJH"
        ],
        "TourDates": {
            "Seattle": "20150625",
            "Cleveland": "20150630"
        },
        "Rotation": "Heavy"
    }
}
```

DynamoDB : dénormalisation par duplication

Authors		
AuthorId	**AuthorName**	**AuthorBirthdate**
1	John Grisham	February 8, 1955
2	Stephen King	September 21, 1947
3	J.K. Rowling	July 31, 1965

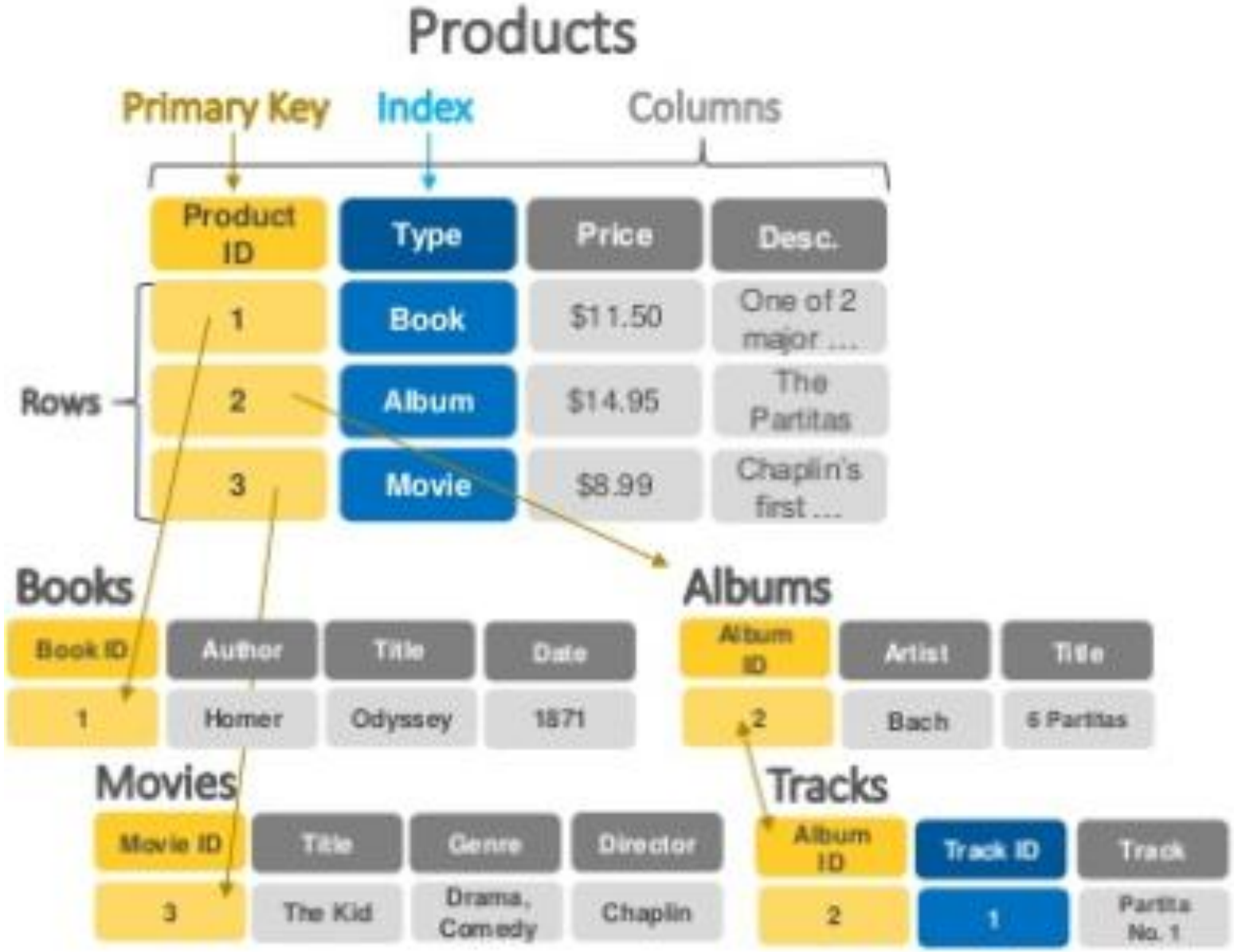
Books			
BookId	**AuthorId**	**BookTitle**	**ReleaseYear**
1	2	The Shining	1977
2	2	It	1986
3	3	Harry Potter and the Sorcerer's Stone	1997

Modèle relationnel modélisant des livres et des auteurs (chaque livre ayant un seul auteur)

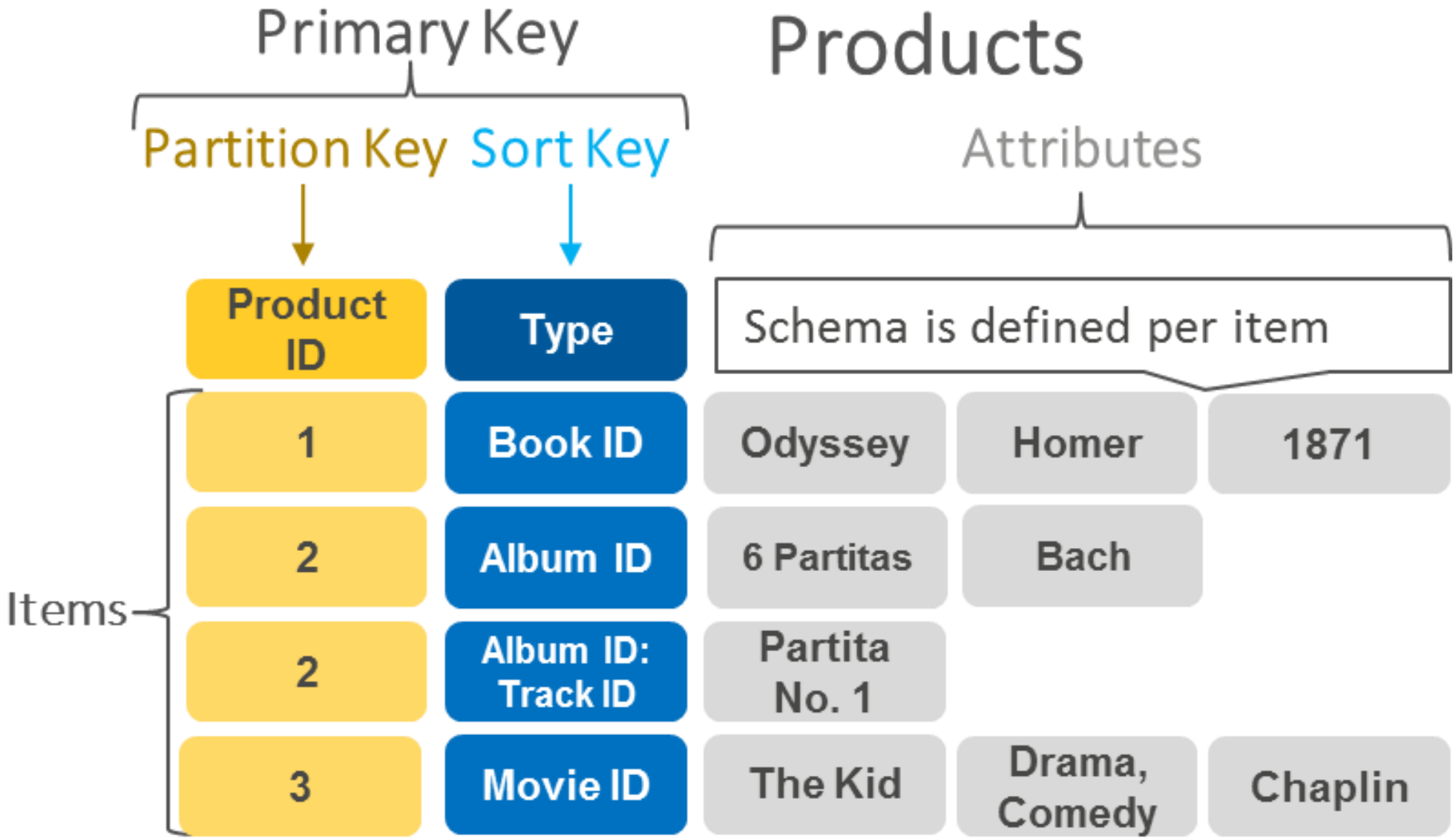
Primary key		Attributes	
Partition key: AuthorName	**Sort key: BookName**		
Stephen King	It	AuthorBirthdate	ReleaseYear
		September 21, 1947	1986
	The Shining	AuthorBirthdate	ReleaseYear
		September 21, 1947	1977
J.K. Rowling	Harry Potter and the Sorcerer's Stone	AuthorBirthdate	ReleaseYear
		July 31, 1965	1997

Collection d'*items* sous DynamoDB : avec duplication de données

Exemple d'un modèle relationnel



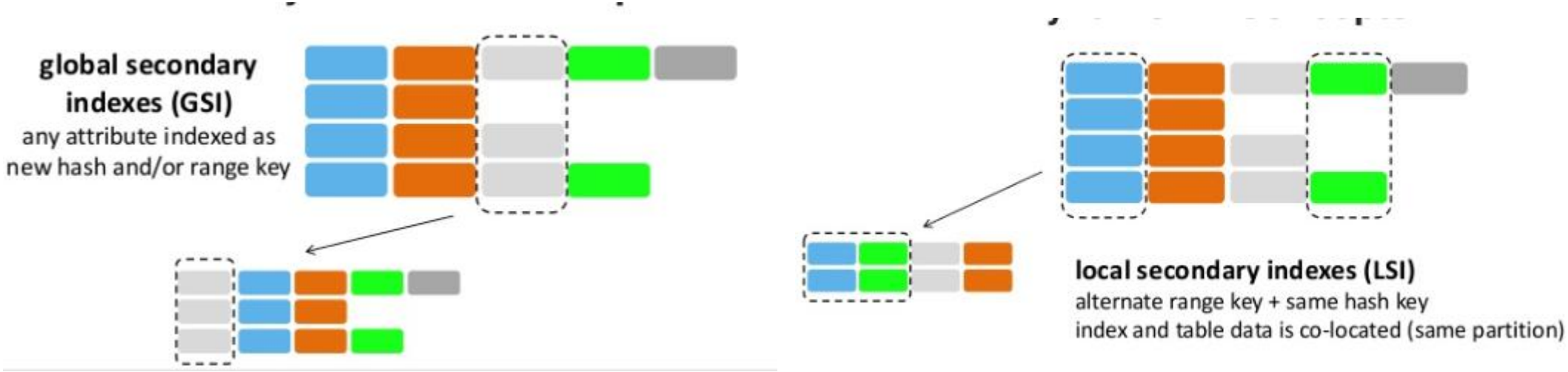
Exemple du même modèle sous DynamoDB



DynamoDB : indexation

Possibilité de créer des index secondaires :

- ***Global secondary index*** : index avec une clé de partition et une clé de tri qui peuvent être différentes de celles de la table
- ***Index secondaire local*** **:** index avec la même clé de partition que la table, mais une clé de tri différente



DynamoDB : exemple d'index global

Primary Key		Attributes		
Actor (PARTITION)	**Movie (SORT)**			
Tom Hanks	Cast Away	**Role**	**Year**	**Genre**
		Chuck Noland	2000	Drama
	Toy Story	**Role**	**Year**	**Genre**
		Woody	1995	Children's
Tim Allen	Toy Story	**Role**	**Year**	**Genre**
		Buzz Lightyear	1995	Children's
Natalie Portman	Black Swan	**Role**	**Year**	**Genre**
		Nina Sayers	2010	Drama

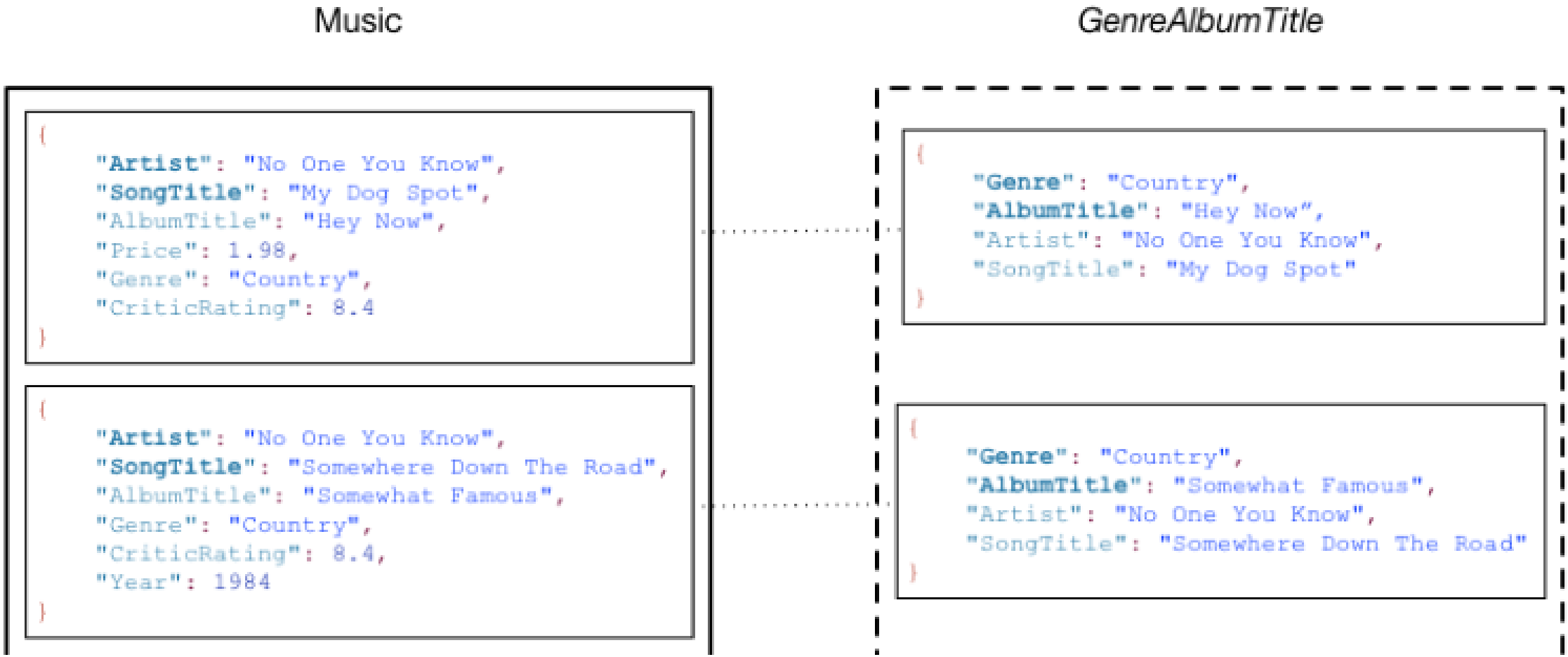
GSI (MoviesActorsIndex)		Attributes		
Movie (PARTITION)	**Actor (SORT)**			
Cast Away	Tom Hanks	**Role**	**Year**	**Genre**
		Chuck Noland	2000	Drama
Toy Story	Tom Hanks	**Role**	**Year**	**Genre**
		Woody	1995	Children's
Toy Story	Tim Allen	**Role**	**Year**	**Genre**
		Buzz Lightyear	1995	Children's
Black Swan	Natalie Portman	**Role**	**Year**	**Genre**
		Nina Sayers	2010	Drama

DynamoDB : exemple d'index local

Primary Key		Attributes		
Actor (PARTITION)	**Movie (SORT)**			
Tom Hanks	Cast Away	**Role**	**Year**	**Genre**
		Chuck Noland	2000	Drama
	Toy Story	**Role**	**Year**	**Genre**
		Woody	1995	Children's
Tim Allen	Toy Story	**Role**	**Year**	**Genre**
		Buzz Lightyear	1995	Children's
Natalie Portman	Black Swan	**Role**	**Year**	**Genre**
		Nina Sayers	2010	Drama

LSI (ActorYearIndex)		Attributes		
Actor (PARTITION)	**Year (SORT)**			
Tom Hanks	2000	**Role**	**Movie**	**Genre**
		Chuck Noland	Cast Away	Drama
	1995	**Role**	**Movie**	**Genre**
		Woody	Toy Story	Children's
Tim Allen	1995	**Role**	**Movie**	**Genre**
		Buzz Lightyear	Toy Story	Children's
Natalie Portman	2010	**Role**	**Movie**	**Genre**
		Nina Sayers	Black Swan	Drama

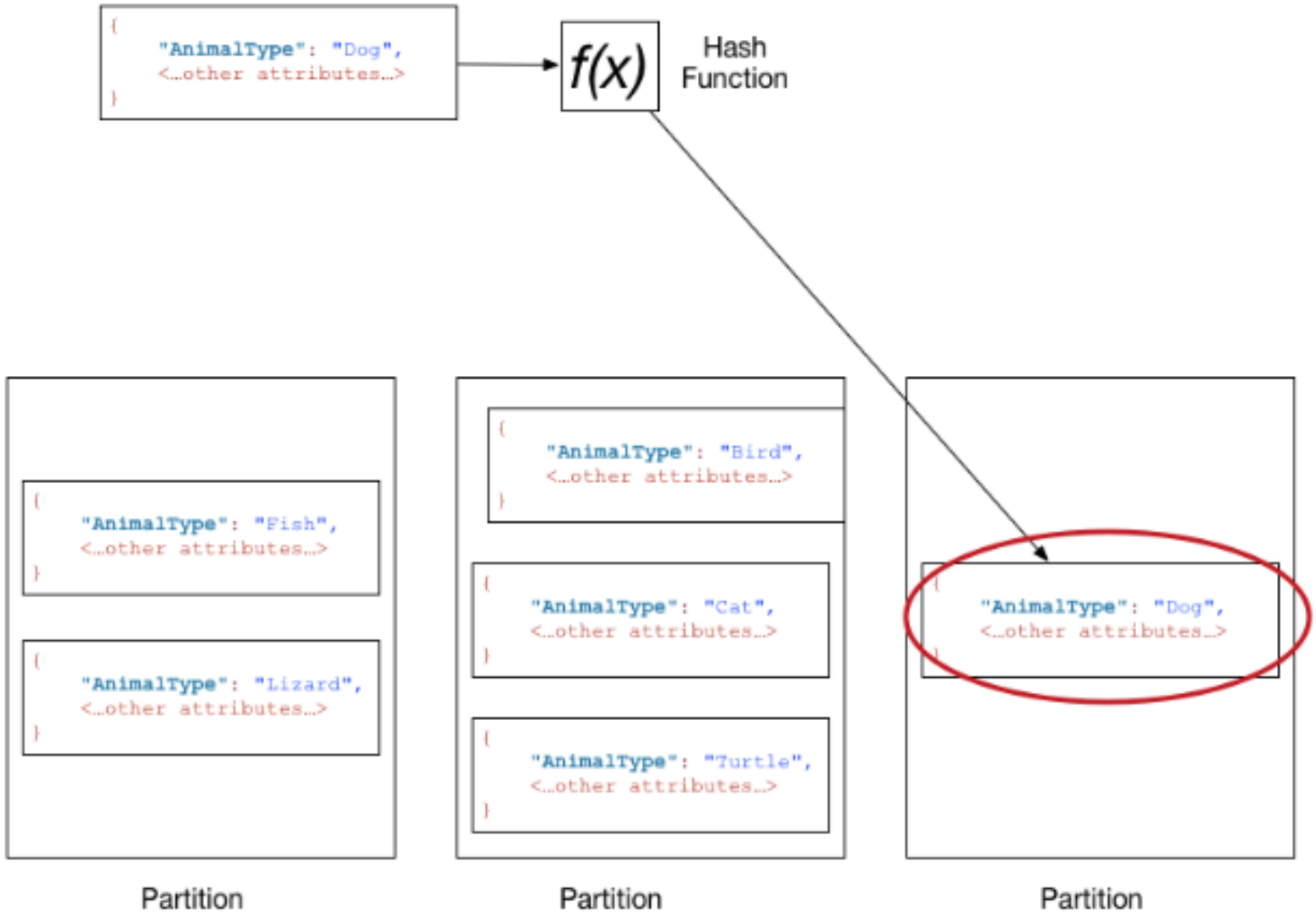
DynamoDB : autre exemple d'index global



DynamoDB : partionnement

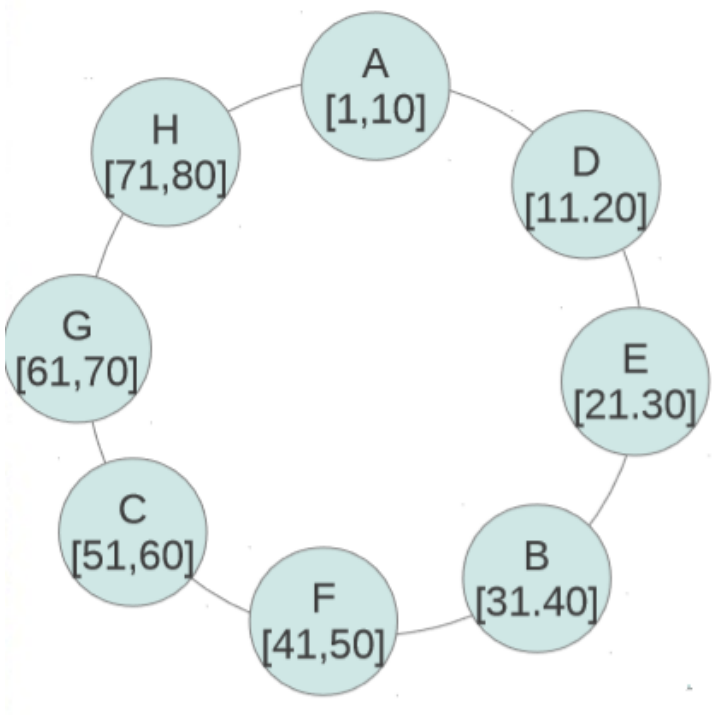
- Stocke des données dans des partitions
- Partition : allocation de stockage pour une table, automatiquement répliquée sur plusieurs zones de disponibilité
- Partition déterminée à partir valeur de sortie de la fonction de hachage appliquée à la valeur de la clé de partition
- Stockage de tous les éléments avec la même valeur de clé de partition physiquement proches les uns des autres, triés par la valeur de clé de tri

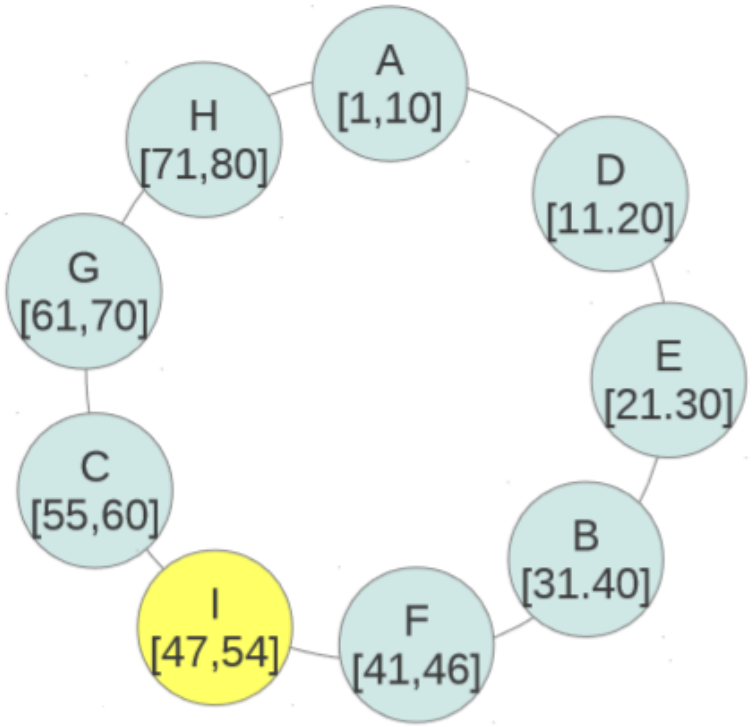
DynamoDB : exemple de partionnement



DynamoDB : hachage cohérent

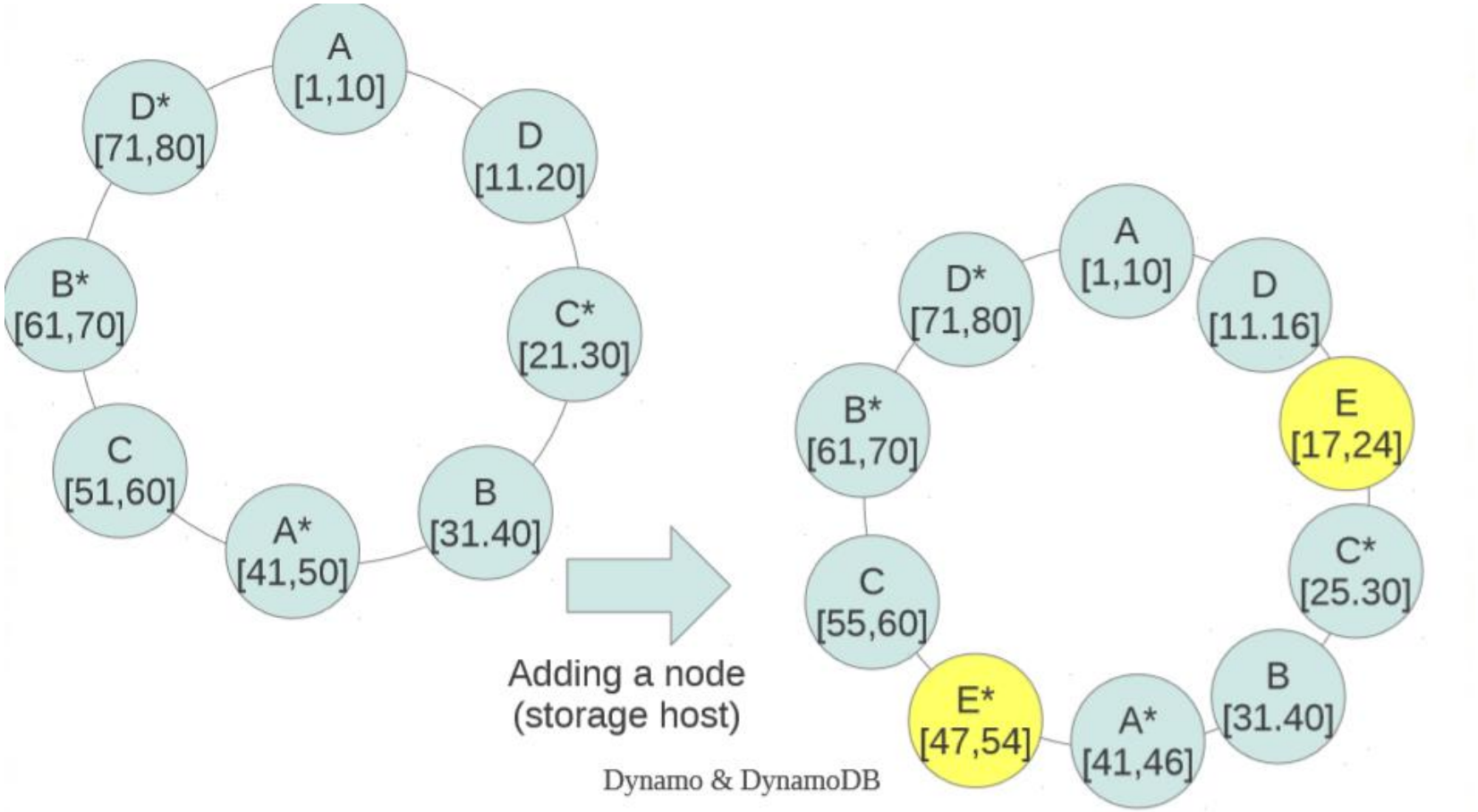
- Données partitionnées sur différents nœuds répartis sur un anneau (*ring*)





DynamoDB : hachage cohérent avec nœuds virtuels

Utilisation du hachage cohérent avec des nœuds virtuels pour homogénéiser la répartition des données



DynamoDB : réplication

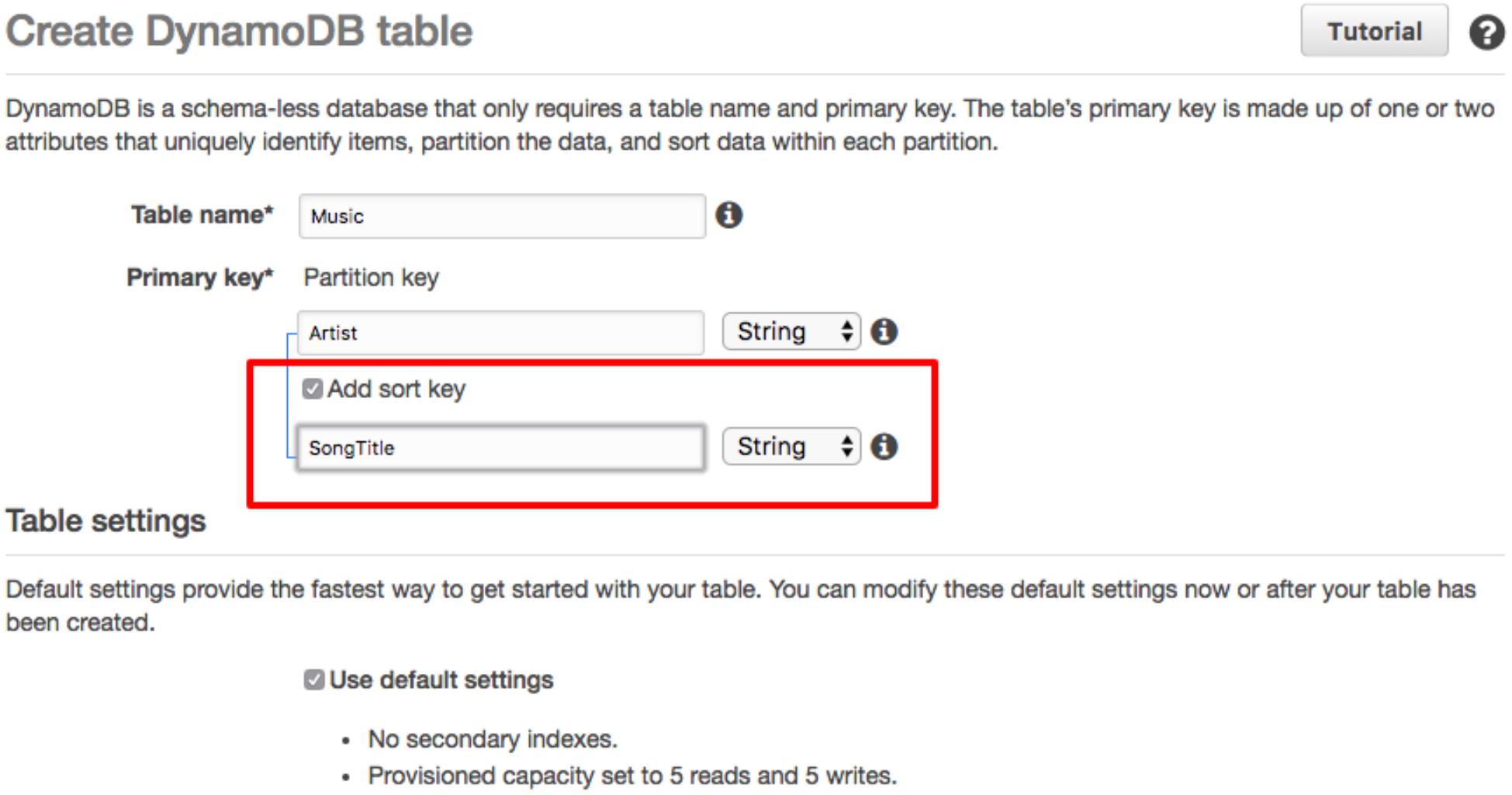
- Réplication de chaque item sur *N* (*replication factor*) nœud physique (pas virtuel)
- Association de chaque clé à un nœud coordinateur
  - Responsable des opérations de lecture et d'écriture pour la clé
  - Responsable du stockage en local des items associés à la clé et de leur réplication sur *N-1* nœuds voisins (enregistrés dans une liste de préférence)
- Facteur de réplication par défaut : 3

Modèle de données DynamoDB : différence avec SQL

```sql
CREATE TABLE Music (
    Artist VARCHAR(20) NOT NULL,
    SongTitle VARCHAR(30) NOT NULL,
    AlbumTitle VARCHAR(25),
    Year INT,
    Price FLOAT,
    Genre VARCHAR(10),
    Tags TEXT,
    PRIMARY KEY(Artist, SongTitle)
);
```

```
{
    TableName : "Music",
    KeySchema: [
        {
            AttributeName: "Artist",
            KeyType: "HASH", //Partition key
        },
        {
            AttributeName: "SongTitle",
            KeyType: "RANGE" //Sort key
        }
    ],
    AttributeDefinitions: [
        {
            AttributeName: "Artist",
            AttributeType: "S"
        },
        {
            AttributeName: "SongTitle",
            AttributeType: "S"
        }
    ],
    ProvisionedThroughput: {       // Only specified if using provisioned mode
        ReadCapacityUnits: 1,
        WriteCapacityUnits: 1
    }
}
```

DynamoDB : création d'une table via l'interface



Requêtes d'interrogation DynamoDB : différence avec SQL (1/4)

```
/* Return all of the data in the table */
SELECT * FROM Music;
```

```
// Return all of the data in the table
{
    TableName:  "Music"
}
```

Requêtes d'interrogation DynamoDB : différence avec SQL (2/4)

```sql
/* Return all of the values for Artist and Title */
SELECT Artist, Title FROM Music;
```

```
// Return all of the values for Artist and Title
{
    TableName:  "Music",
    ProjectionExpression: "Artist, Title"
}
```

Requêtes d'interrogation DynamoDB : différence avec SQL (3/4)

```sql
/* Return a single song, by primary key */

SELECT * FROM Music
WHERE Artist='No One You Know' AND SongTitle = 'Call Me Today';
```

```
// Return a single song, by primary key

{
    TableName: "Music",
    KeyConditionExpression: "Artist = :a and SongTitle = :t",
    ExpressionAttributeValues: {
        ":a": "No One You Know",
        ":t": "Call Me Today"
    }
}
```

Requêtes d'interrogation DynamoDB : différence avec SQL (4/4)

```sql
/* Return all of the songs by an artist, with a particular word in the title...
...but only if the price is less than 1.00 */

SELECT * FROM Music
WHERE Artist='No One You Know' AND SongTitle LIKE '%Today%'
AND Price < 1.00;
```

```
// Return all of the songs by an artist, with a particular word in the title...
// ...but only if the price is less than 1.00

{
    TableName: "Music",
    KeyConditionExpression: "Artist = :a and contains(SongTitle, :t)",
    FilterExpression: "price < :p",
    ExpressionAttributeValues: {
        ":a": "No One You Know",
        ":t": "Today",
        ":p": 1.00
    }
}
```

DynamoDB + PartiQL

- Annonce en 2019 d'un un nouveau langage de requête compatible avec SQL : **PartiQL** – compatible avec DynamoDB depuis nov. 2020 https://aws.amazon.com/fr/about-aws/whats-new/2020/11/you-now-can-use-a-sql-compatible-query-language-to-query-insert-update-and-delete-table-data-in-amazon-dynamodb/
- **PartiQL : «** Langage entièrement open source sous licence Apache2.0, présenté comme « hautement flexible », censé faciliter la recherche et l'extraction efficace de grandes quantités et variétés de données, quel que soit le lieu ou le format dans lequel elles sont stockées »
- https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-reference.html
- https://partiql.org/
- https://medium.com/@jvroig/dynamodb-partiql-is-fun-but-dangerous-fec0f2803220

DynamoDB : conclusion

 **Gestion clé-valeur et document**

 **Scalabilité**

 **Certaine « gestion de transactions ACID »**

 **Pas d'infrastructure à gérer**

 **Taille limitée pour les items et les requêtes**

 **Plus on veut de fonctionnalité plus on paye**

 **Pas de jointure entre tables**