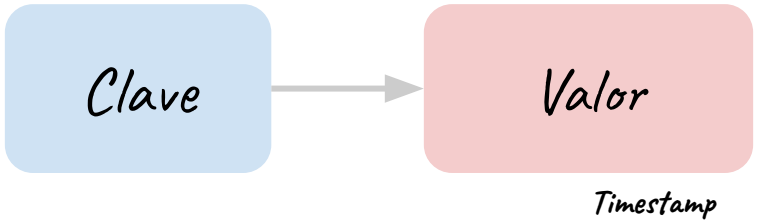

# NoSQL - Modelado

Rafael Garrote Hernández
*Profesor en NoSQL*

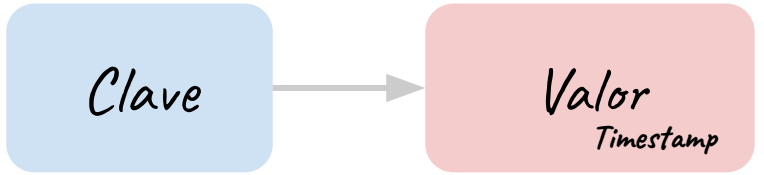
# Clave Valor

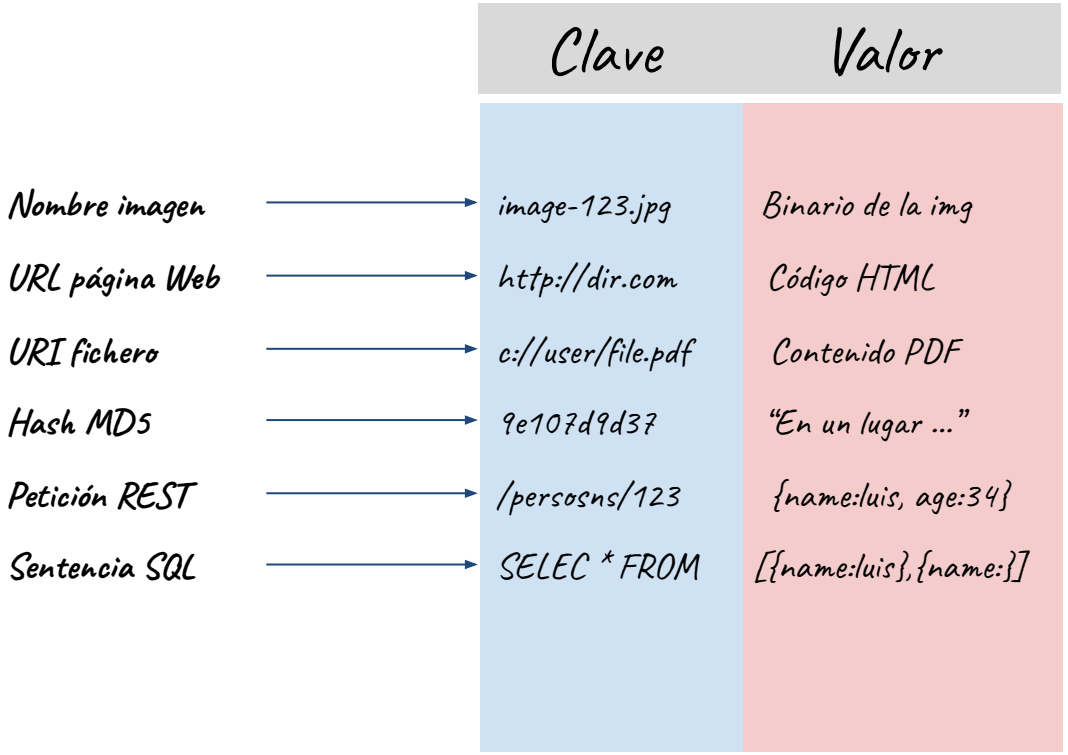
# Clave Valor




- La clave es única.
- Tanto la clave como el valor son un array de bytes.
- El valor tiene asociado el timestamp de la operación.
- Trabajan en memoria.

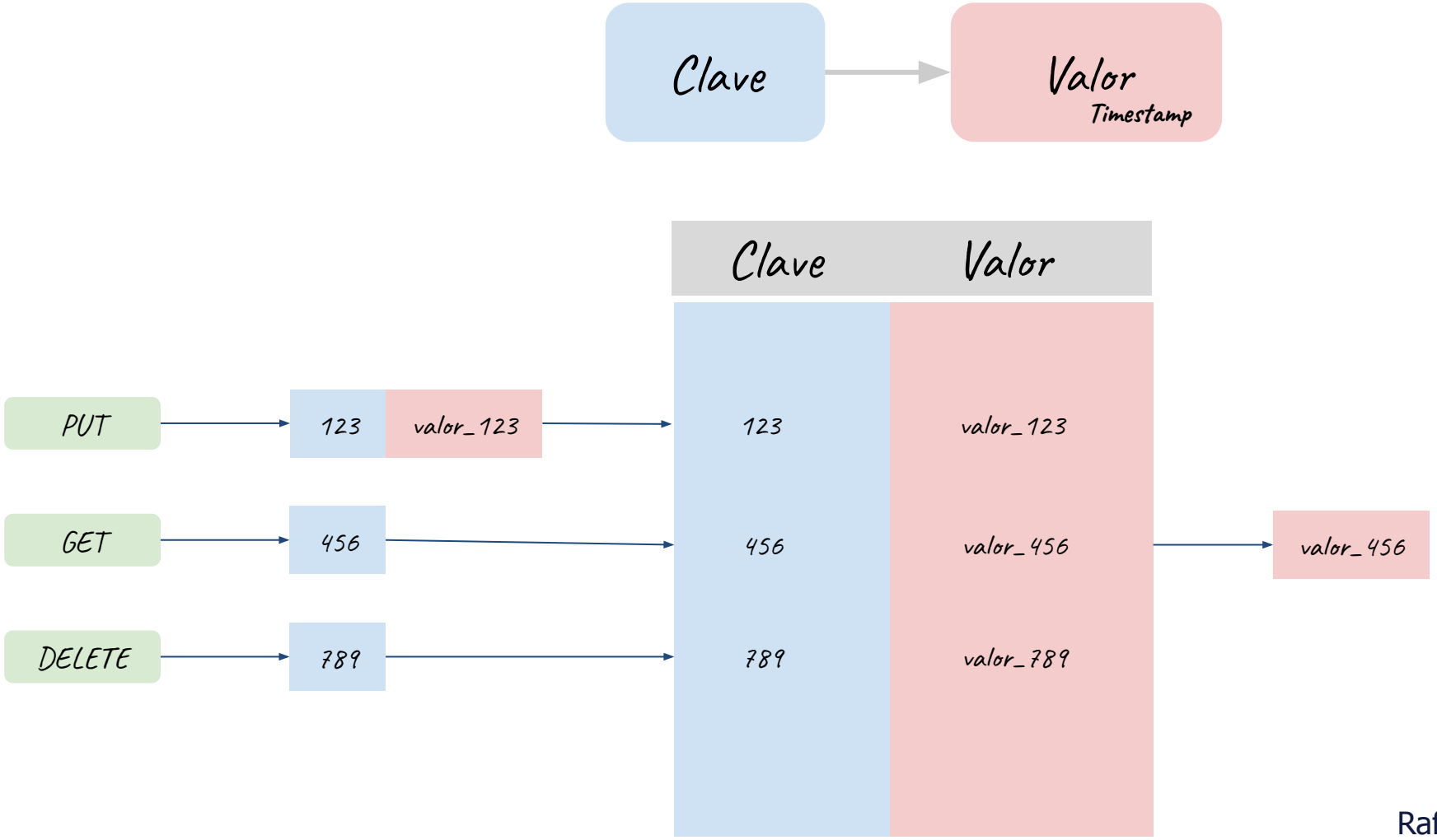
# Clave Valor




Rafael Garrote Hernández

# Clave Valor




Rafael Garrote Hernández

# Clave Valor

Consistent Hashing

$f(X) \rightarrow Y$

$f(Z) \rightarrow W$

$Y$ distinto $W$

f(00001A) → a12e03c0

f(99999B) → b32ac12b

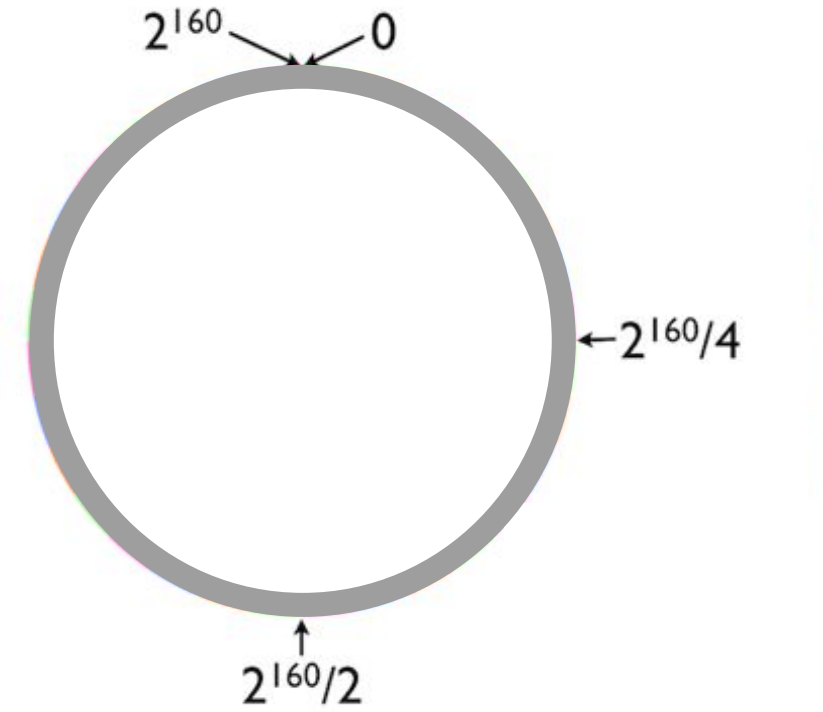
f(00001A) → a12e03c0



# Clave Valor

Consistent Hashing

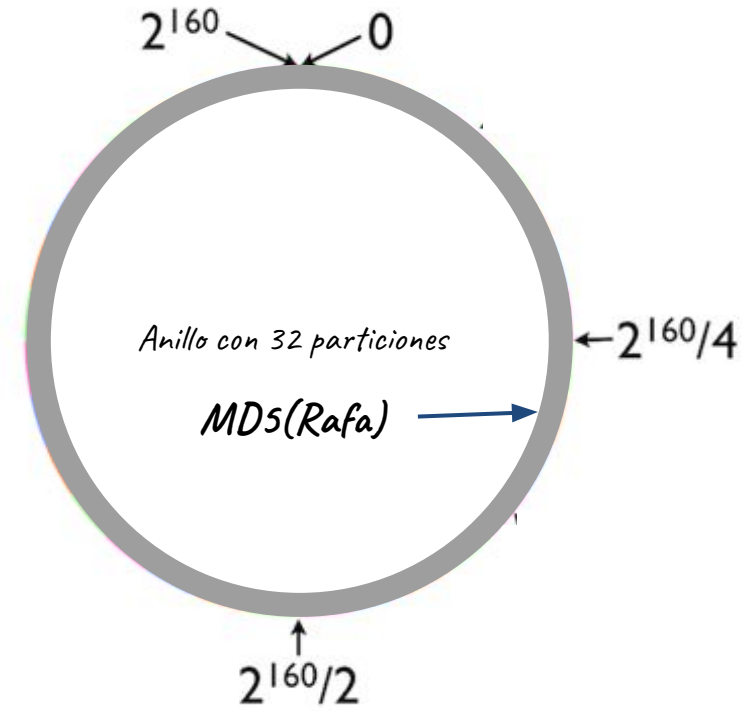
MD5(X) ⟶ (2^160) - 1



# Clave Valor

Consistent Hashing

MD5(X) → (2^160) - 1



# Clave Valor

## Consistent Hashing

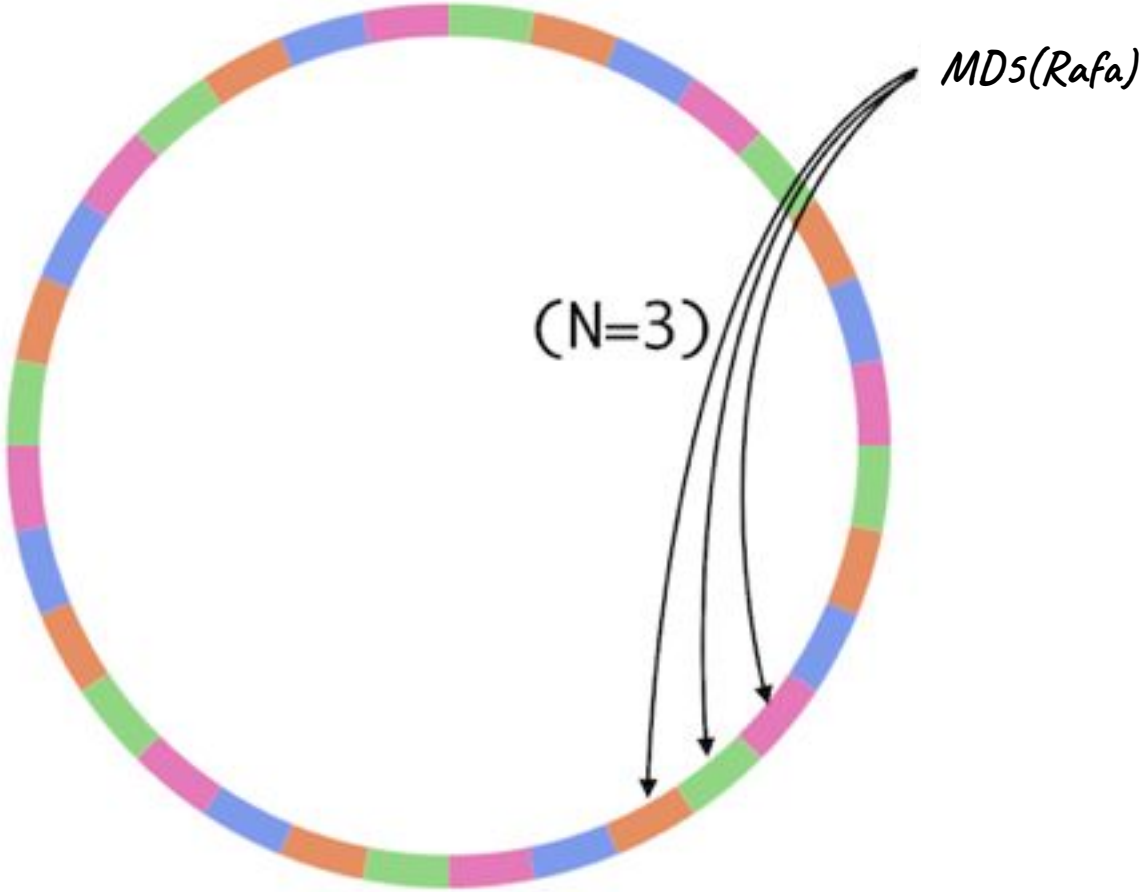
Dividir el anillo en particiones:

- Una vez dividido no se puede volver a dividir con más o menos particiones. Escoger un valor alto.
- Cada nodo se divide en subnodos y cada subnodo se hace coincidir con una partición.
- Inicialmente los datos quedan repartidos de forma equitativa entre las claves.
- Añadir un nodo supone que todos los nodos ceden un subnodo al nodo nuevo.
- Eliminar un nodo supone que cada nodo acoge un subnodo del nodo eliminado.
- Si se añade un nuevo nodo, habrá que calcular la claves que están contenidas en cada nodo para añadir el nuevo nodo donde más concentración de claves haya para "aliviar" la parte del cluster donde haya más claves.
- Para saber en qué subnodo se encuentra una clave se aplica la función hash.



# Clave Valor

Consistent Hashing



# Clave Valor

Consistent Hashing

- Las bases de datos NoSQL para garantizar la alta disponibilidad y un rendimiento alto de operaciones utilizan **replicación de las particiones por el cluster**.
- Cada partición es almacenada por más de un nodo del cluster.
- Número mínimo de replicación para garantizar la disponibilidad 3.
- Todos los **nodos son equivalentes**, no hay replicación maestro esclavo:
  - No SPOF
  - Consistencia eventual



# Clave Valor

- No tiene esquema asociado al valor. El valor es un array de bytes.
- Las únicas operaciones que permiten son:
  - Añadir un par clave/valor.
  - Eliminar un par clave/valor.
  - Modificar el valor asociado a una clave.
  - Obtener el valor asociado a una clave.
- No permite búsquedas sobre el valor asociado a la clave, para el datastore es un array de bytes.
- No permite filtros, agrupaciones, agregaciones…
- No permite Joins entre datos.



# Clave Valor

- Si no permite filtros (where ...), ¿cómo podemos sustituir esta funcionalidad?
- Algunas bases de datos clave - valor permiten el uso de índices secundarios.
- Un índice es un nuevo conjunto clave-valor donde la clave es el valor por el que queremos filtrar y el valor el conjunto de claves que cumplen la condición del filtro.



# Clave Valor

- Si no permite Joins ¿cómo relacionamos datos?
  - Deja en el valor todos los datos que necesites al acceder a la clave. ¿Duplicidad de información? La coherencia es responsabilidad tuya. ¿Puedes manejarla?
  - Utiliza índices secundarios para relacionar grupos.
- Si se complica mucho, plantéate si es el datastore más adecuado para tu caso de uso.



# Clave Valor

**Pinterest**

Es una base de datos que permite a sus usuarios colgar imágenes, vídeos, textos. etc, y compartirlo con el resto de la comunidad, permitiendo crear tablones temáticos donde poder colgar y clasificar más fácilmente el contenido.

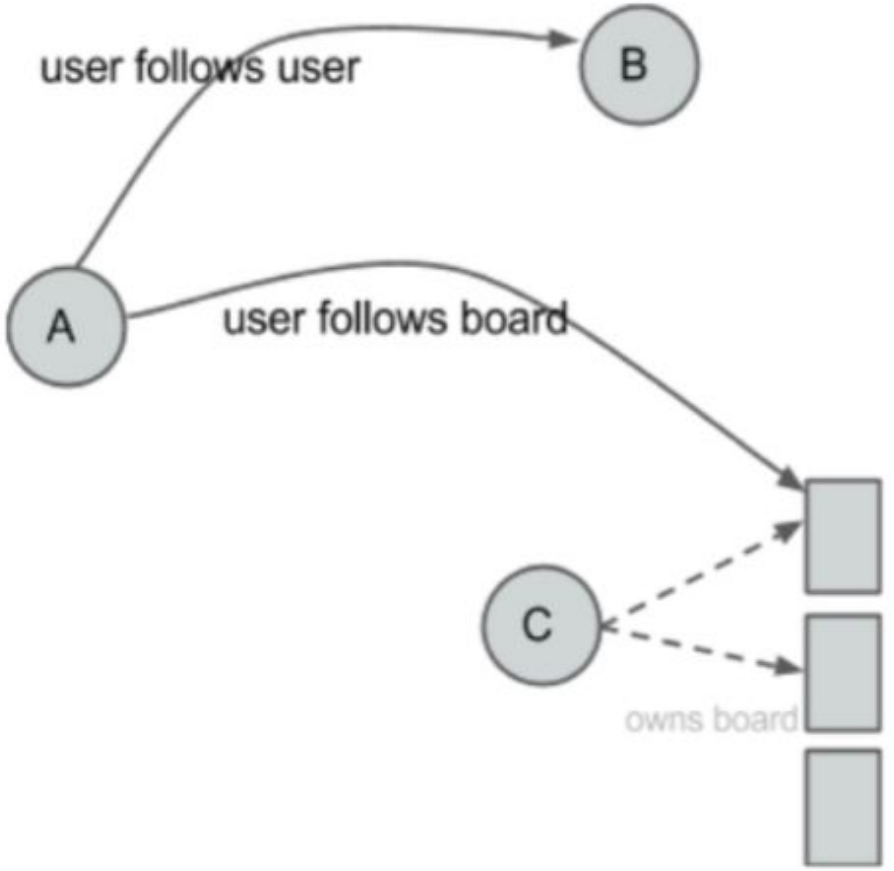
Para ello Pinterest mantienen el modelo de datos que ellos llaman **Follower Graph**:

- Pinterest permite que sus usuarios sigan a otros usuarios, lo que significa que, por extensión, seguirán a todos los tablones que haya generado ese usuario.
- También permite que los usuarios sigan tablones concretos de usuario.



# Clave Valor

- Según Pinterest, el tamaño del grafo no es excesivo, por lo que almacenarlo en memoria es una posibilidad.
- Para ello utilizan Redis.
- El grafo está particionado por UserID.
- Persiste en disco cada segundo para reducir la posibilidad de pérdida de información.
- Cada instancia de Redis mantiene una partición diferente, lo que facilita la división y el reparto de particiones cuando la máquinas alcanzan sobrecargas.



# Clave Valor

**Tweeter**

Es una de las redes sociales más utilizadas del mundo, por tanto con más carga del mundo.

Los usuarios de Twitter pueden seguir a otros usuarios, y por cada usuario Twitter mantiene un timeline con todos los tweets que ha escrito la gente a la que sigue ese usuario.

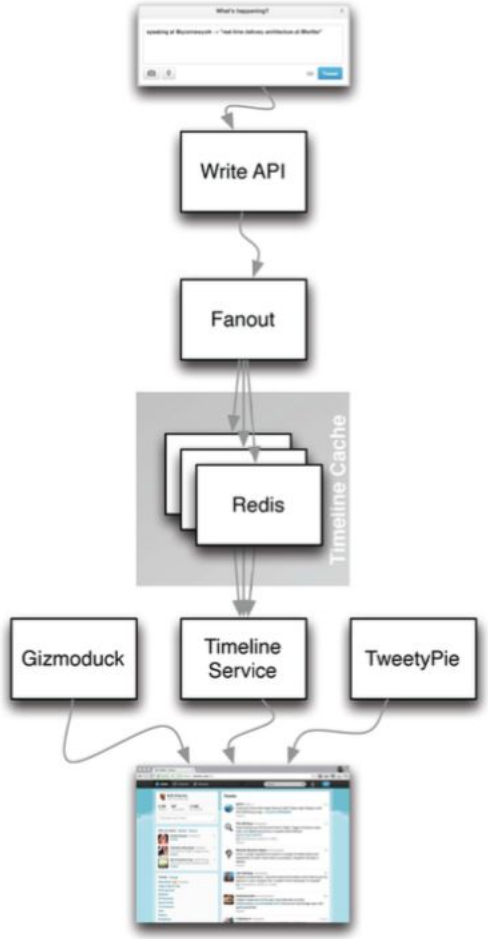
Cada vez que el usuario accede a Twitter a través de la web o de una aplicación externa, el timeline suele ser la primera pantalla que se muestra.

Cada vez que un usuario escribe un nuevo tweet, un proceso lee los seguidores de ese usuario y escribe en sus timelines una referencia al tweet que acaba de ser escrito.



# Clave Valor

- Twitter mantiene en un clúster de Redis en el que se gestionan los timelines de los usuarios del sitio en memoria.
- Cada vez que un usuario escribe un tweet, se realiza una inserción en Redis en los timelines de cada uno de sus followers.
- Twitter recibe un promedio escrituras de 5.000 tweets/s, con picos de 12.000, y un ratio de lecturas de 300.000 tweets/s.
- Personalidades como Justin Bieber o Barack Obama, que cuentan con decenas de millones de seguidores cada uno. Cada tweet que escribe una de estas personas, provoca unos 50 millones de actualizaciones en Redis, una por cada seguidor.



# Clave Valor

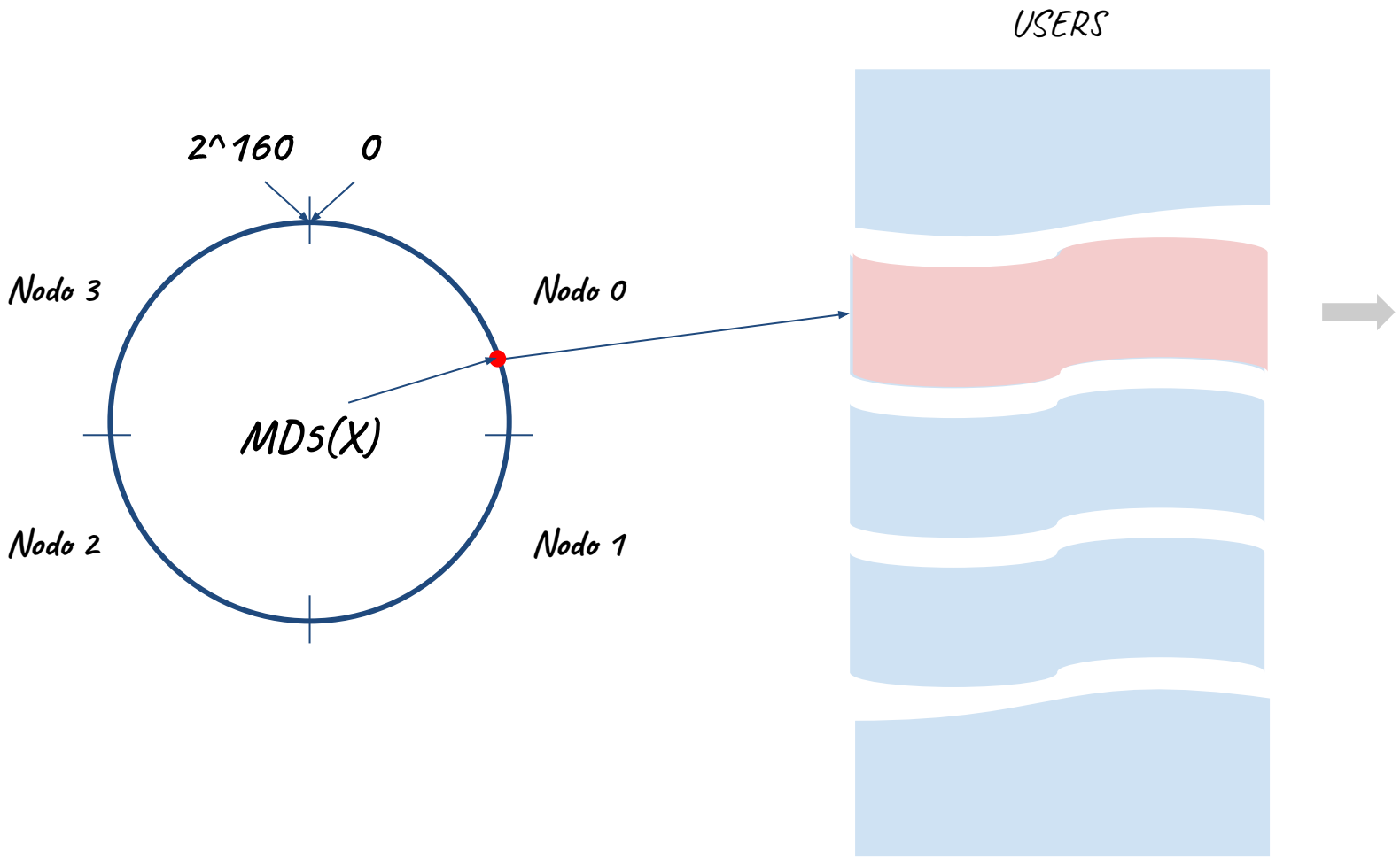
## Otros casos de uso

- Aplicaciones simples que requieren alto rendimiento en operaciones de lectura / escritura.
- Aplicaciones que necesiten que la base de datos esté siempre disponible.
- Servir anuncios a aplicaciones web / móvil .
- Enormes cantidades de información (Clicks /Sensores).
- Preferencias de usuarios.
- Almacenamiento de sesión.



# Familias de Columnas

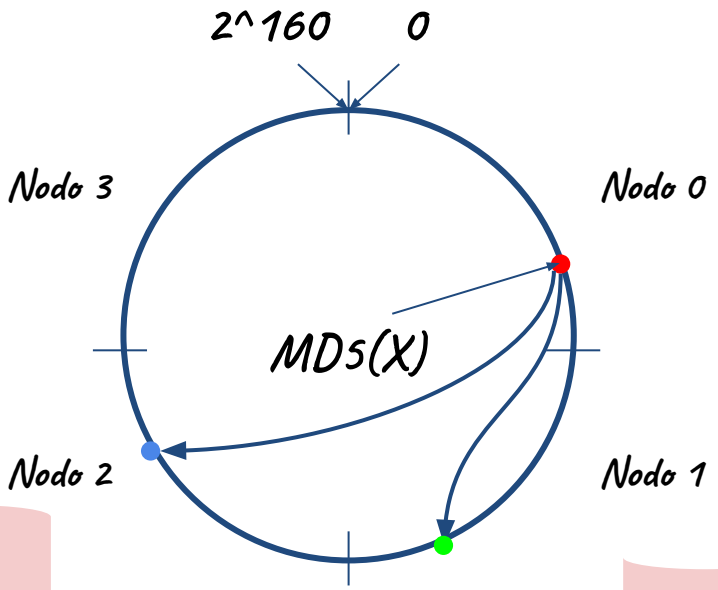
# Familia de columnas - Particionado




| Col 1 | Col 2 | Col 3 | Col 4 |
|---|---|---|---|
| | | | |
| | | | |
| | | | |

Rafael Garrote Hernández

# Familia de columnas - Particionado

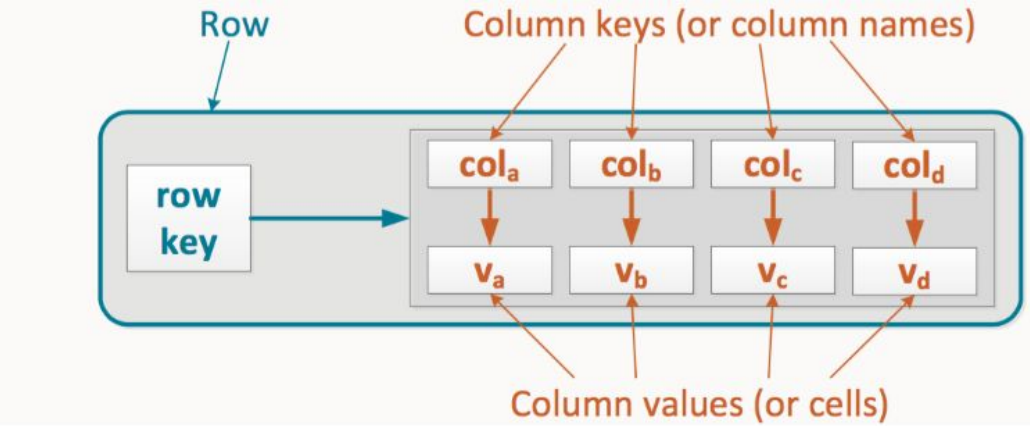



| Col 1 | Col 2 | Col 3 | Col 4 |
|---|---|---|---|
| | | | |
| | | | |
| | | | |

| Col 1 | Col 2 | Col 3 | Col 4 |
|---|---|---|---|
| | | | |
| | | | |
| | | | |

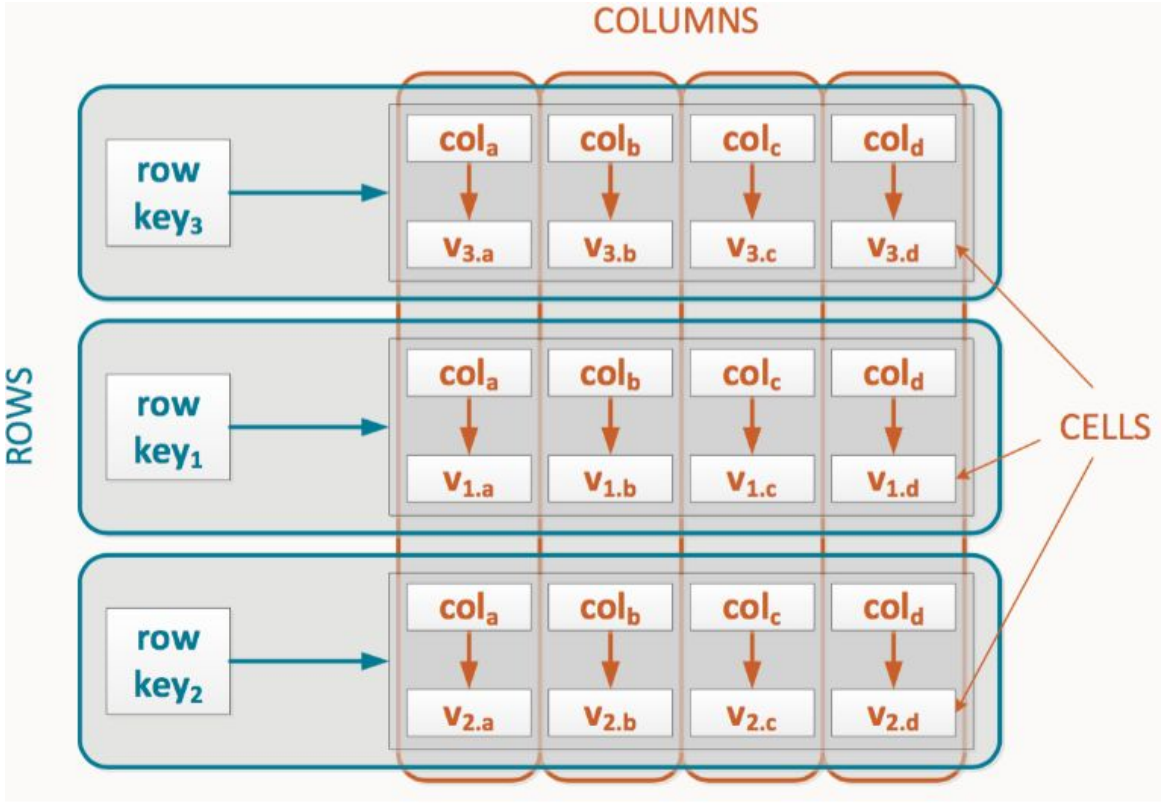
| Col 1 | Col 2 | Col 3 | Col 4 |
|---|---|---|---|
| | | | |
| | | | |
| | | | |

# Familia de columnas - Modelado



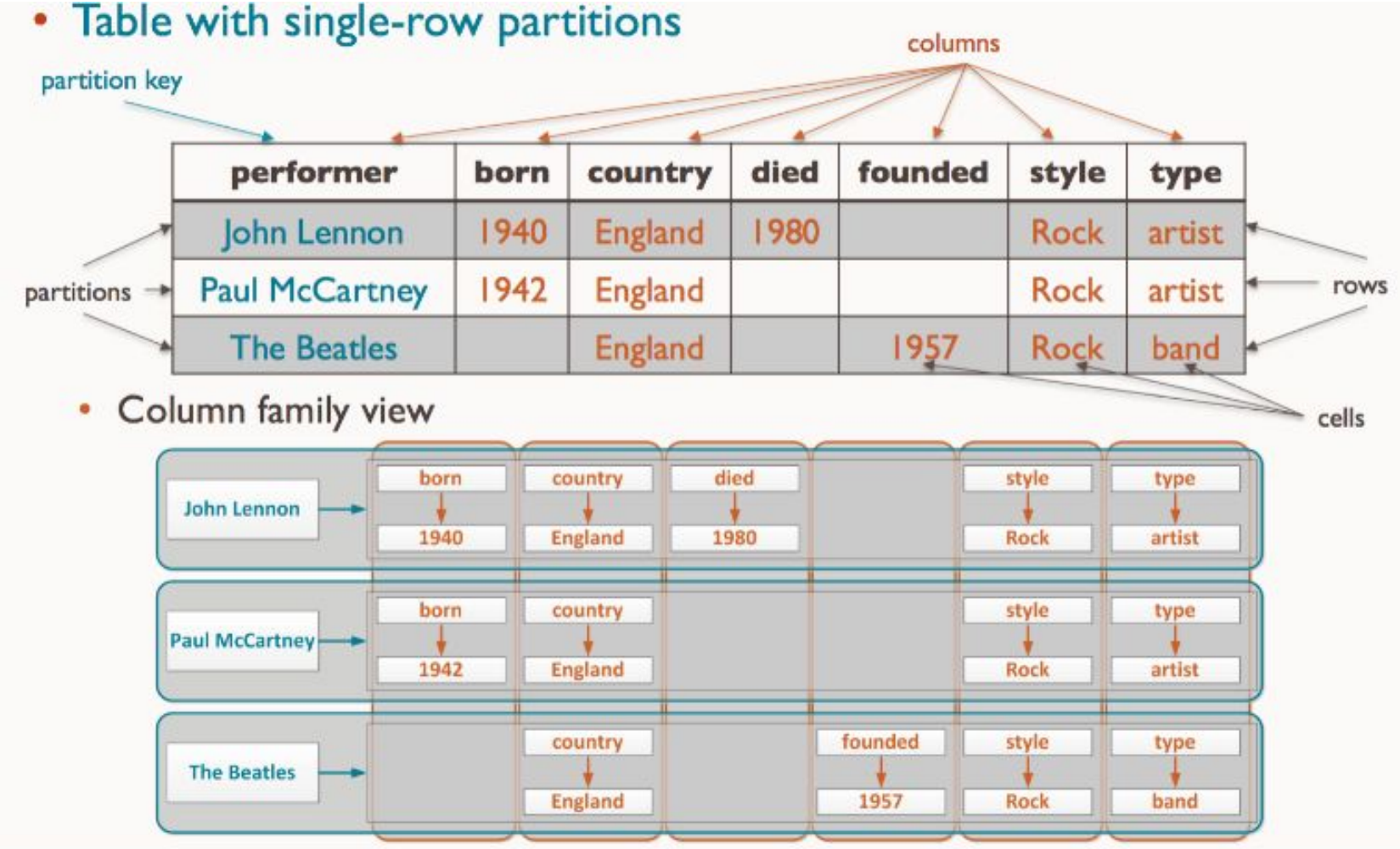

- Una **partición** o fila (**row**) es la unidad mínima de acceso de información.
- Una partición almacena columnas. Y una columna es un registro clave / valor.
- Una partición tiene un **row key** que identifica a la partición de forma unívoca.
- Un **row key** es una o un conjunto de columnas del conjunto de datos que queremos almacenar.
- La **column key** identifica una columna en una partición.
- El **column value** almacena el valor de una columna en una partición.

# Familia de columnas - Modelado

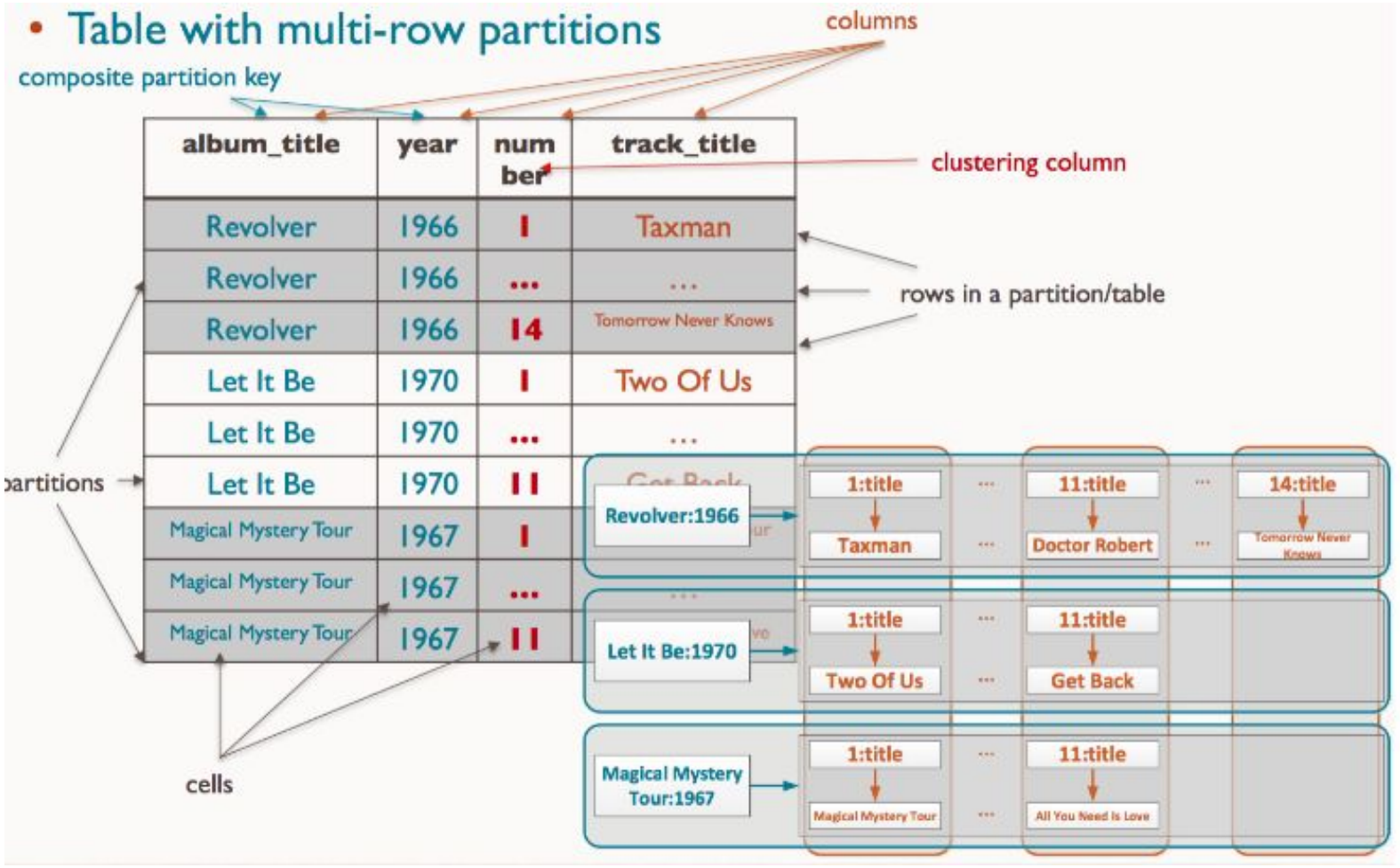



- Un conjunto de particiones constituye una **familia de comunas** que se puede identificar con una tabla en una base de datos relacional.

# Familia de columnas - Modelado



# Familia de columnas - Modelado



# Familia de columnas - Primary Key

```
Primary key = (ParitionKey, ColumnKey1, ColumnKey2, ... , ColumnKeyN)
```

La **Primary Key** identifica de forma unívoca cada uno de los registros de una familia de columnas.

- Se compone de dos partes:
  - **Partition Key** (row key): Una o más columnas que identifican de forma unívoca una partición dentro de la base de datos. Se utiliza para particionar los datos.
  - **Clustering Key**: Una o más column keys que añadimos a la clave de primaria para hacer única a la clave y poder identificar un registro de forma unívoca dentro de una partición.



# Familia de columnas - Primary Key

- El **orden** de las colum keys que forman las clustering keys de la primary key importa.
- El orden permite **agrupar** la información para cumplir los criterios de búsqueda de los datos dentro de la partición.
- La base de datos almacena los datos de cada una de las columnas de forma **ordenada** dentro de cada partición. Por defecto los almacena de forma ascendente.



# Familia de columnas - Primary Key

**Ejemplo1:**

Queremos saber cuántos usuarios de twitter hay por ciudad en España.

Los datos que disponemos en nuestro data lake por cada usuario son los siguientes:

- Nickname
- Nombre completo de usuario
- País
- Ciudad
- tweets del usuario (tweet_id, tweet_text, retweets)



PrimaryKey = (**país**, **ciudad**, **nickname**, **tweet_id**)

```
SELECT ciudad, count(nickname) AS num_users
FROM users
WHERE país = 'es'
GROUPBY ciudad;
```

| país | ciudad | nickname | tweet_id | nombre_de_usuario | tweet_text | retweets |
|---|---|---|---|---|---|---|
| es | Barcelona | pepe | 1182 | Pepe Hernández | Lorem ipsum…. | 400 |
| es | Barcelona | pepe | 1267 | Pepe Hernández | Lorem ipsum…. | 5 |
| es | Barcelona | maría | 1673 | María González | Lorem ipsum…. | 12 |
| es | Madrid | pepe | 3894 | José Fernández | Lorem ipsum…. | 76 |
| es | Madrid | luis | 6787 | Luis Gómez | Lorem ipsum…. | 3 |
| es | Salamanca | ana | 8345 | Ana Fernández | Lorem ipsum…. | 0 |
| es | Salamanca | pepe | 9834 | Jose Hernández | Lorem ipsum…. | 10 |
| fr | | | | | | |
| fr | | | | | | |
| uk | | | | | | |
| uk | | | | | | |



# Familia de columnas - Primary Key

**Ejemplo2:**

Queremos saber cual es el tweet con más retweets en España y los datos del usuario que lo ha generado.

Los datos que disponemos en nuestro data lake por cada usuario son los siguientes:

- Nickname
- Nombre completo de usuario
- País
- Ciudad
- tweets del usuario (tweet_id, tweet_text, retweets)



PrimaryKey = (**país**, **retweets**, **tweet_id**)

```
SELECT nickname, nombre_de_usuario, tweet_text, retweets
FROM users
WHERE país = 'es'
Limit 1;
```

| país | retweets | tweet_id | nickname | nombre_de_usuario | tweet_text | ciudad |
|---|---|---|---|---|---|---|
| es | 400 | 1182 | pepe | Pepe Hernández | Lorem ipsum…. | Barcelona |
| es | 76 | 3894 | pepe | José Fernández | Lorem ipsum…. | Madrid |
| es | 12 | 1673 | maria | María González | Lorem ipsum…. | Barcelona |
| es | 10 | 9834 | pepe | Jose Hernández | Lorem ipsum…. | Salamanca |
| es | 5 | 1267 | pepe | Pepe Hernández | Lorem ipsum…. | Barcelona |
| es | 3 | 6787 | luis | Luis Gómez | Lorem ipsum…. | Madrid |
| es | 0 | 8345 | ana | Ana Fernández | Lorem ipsum…. | Salamanca |
| fr | | | | | | |
| fr | | | | | | |
| uk | | | | | | |
| uk | | | | | | |



# Familia de columnas - Query Driven

- Estas bases de datos **no soportan Joins**. Supondría una penalización en el rendimiento y en la escalabilidad ya que tendría que gestionar la comparación de tablas en más de una partición.
- Solución: **Desnormalizar y duplicar información**. Persistir todos los datos que se necesitan para la consulta juntos en la misma tabla.
- **Implica conocer las consultas previo al modelado.**
- Estas bases de datos **no soportan funciones de agrupación**. Supondría una penalización en el rendimiento por cómo están distribuidas las particiones en el cluster.
  - Solución: Persistir la información ya agrupada valiéndonos del diseño de la arquitectura.
  - **Implica conocer las consultas previo al modelado.**



# Familia de columnas - Query Driven

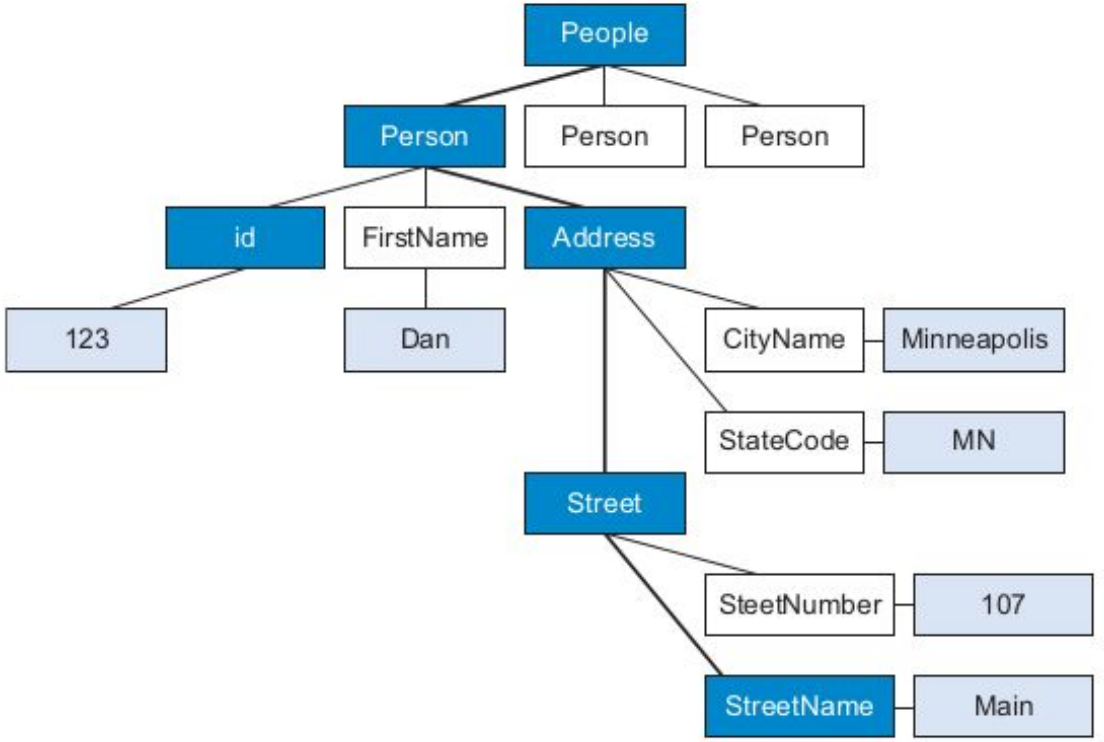
- Query Driven, idealmente **existirá una tabla por consulta**.
- Se **desnormaliza** y se **redunda** la información tanto como sea necesario.
- Se debe optimizar para un **alto rendimiento de lectura**.
- La repartición y ordenación de los datos en la tabla se hará en función de las necesidades de la consulta, ordenación, agrupación...



# Orientadas a Documentos

# Orientado a documentos

- Un documento es la unidad principal de almacenamiento.
- Un documento almacena la información en estructura de árbol.
- Un gestor de documentos indexa automáticamente todos los valores de un documento al ser insertado.
- Permite búsquedas sobre cualquier elemento indexado.



# Orientado a documentos

- Las codificaciones más habituales de estos documentos suelen ser JSON, XML, YAML.
- También pueden almacenar Word o PDF.
- Los documentos se estructuran dentro de una serie de contenedores llamados habitualmente colecciones.
- Los documentos son almacenados asociados a una clave única que es la que se utiliza posteriormente para recuperar los datos.
- No admiten joins entre las colecciones.



# Orientado a documentos

# Orientado a documentos

- Características de un documento:
  - Son jerárquicos.
  - Los documentos tienen un esquema dinámico.
  - Los documentos son tipos nativos en muchas bases de datos.
  - Al poder incluir documentos y arrays dentro de los documentos reduce la necesidad de hacer joins.
  - Los esquemas dinámicos permiten soportar cualquier estructura de datos de una colección.



# Orientado a documentos

- El reto principal que tienen las bases de datos orientadas a documentos es suplir la carencia de expresar relaciones.
- Para ello existen dos herramientas:
  - Referencias a documentos.
  - Embeber documentos en documentos.



# Orientado a documentos

## Documentos embebidos

# Orientado a documentos

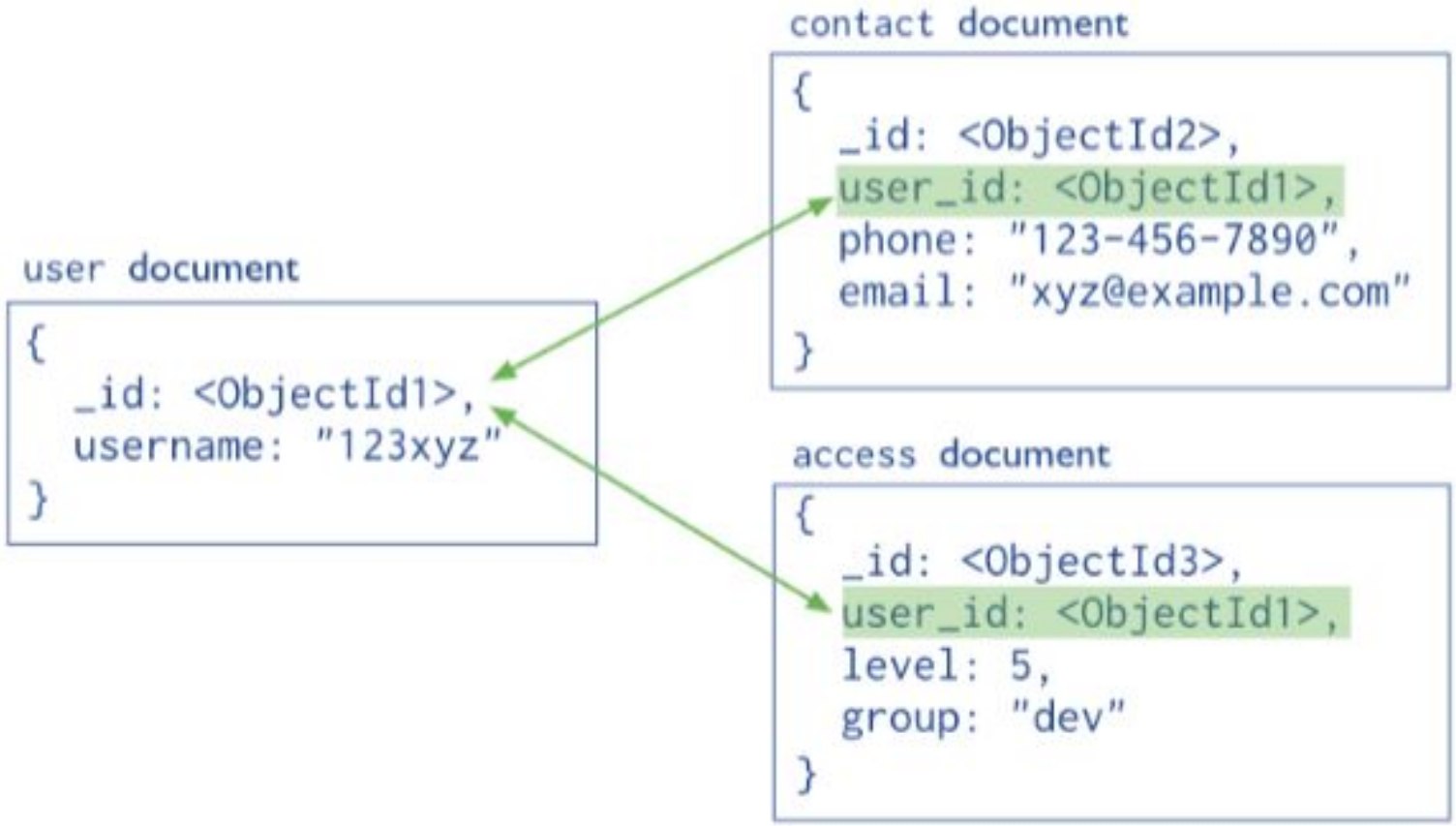
## Documentos embebidos

- Representan relaciones entre los datos almacenando la información relacionada bajo el mismo documento.
- Se pueden embeber bajo un documento o bajo un array.
- Cuándo utilizarlos:
  - Existen relaciones contenidas entre dos entidades. Cuando dos entidades independientes sean accedidas a través sólo de una de ellas. Ejemplo *Persona*, *Dirección*. (Relación 1:1)
  - Relaciones 1:N. En este caso la parte de varios suele ir dentro de la de uno.



# Orientado a documentos

## Referencias



# Orientado a documentos

## Referencias

- Almacenan relaciones entre los datos mediante enlaces entre documentos.
- Cuándo utilizarlas:
  - Cuando embeber los datos produzca una redundancia de la información y los costes de mantenimiento sean mayores que las ganancias en lecturas.
  - Es necesario representar un modelo varios a varios con cierta complejidad.
  - Para modelar conjuntos de datos con mucho tamaño.



# Orientado a documentos

**Foursquare**

Red social cuya actividad consiste en permitir a sus usuarios realizar *check-in* en lugares concretos (marcar lugares como visitados a medida que el usuario los visita y compartir la localización con amigos y contactos).

Cuando Foursquare decidió migrar a MongoDB, su capa de backend consistía en una única base de datos relacional.

Debido al crecimiento exponencial que experimentó desde su creación en 2009 hasta pocos años después, los ingenieros de Foursquare decidieron evaluar, entre otras soluciones, bases de datos NoSQL. Finalmente encontraron en MongoDB la base de datos que les permitía solucionar tanto sus necesidades más inmediatas, como las , previsiblemente, pudieran surgirles más adelante.



# Orientado a documentos

## Que les aporta MongoDB

- El particionado automático, mediante el cual simplemente tienes que añadir nodos al clúster a medida que se van necesitando y el software se encarga de lo demás.
- La indexación geográfica de la que dispone MongoDB, la cual les permitía realizar búsquedas basadas en una ubicación espacial concreta.
- Los ReplicaSet, mediante los cuales se consigue alta disponibilidad y redundancia de nodos en caso de caídas.
- Su modelo de datos dinámico y adaptable a las necesidades en cada punto del proyecto.



# Orientado a documentos

**MTV Networks**

MTV Networks, aparte del canal de TV y de las webs de cada país (mtv.com, mtv.es, etc) es propietaria de un conjunto de sitios tales como spike.tv, gametrailers.com, entre otros, los cuales tienen unas estadísticas de tráfico y accesos incluso mayores que la web de la propia MTV.

El equipo de la web de MTV evaluó migrar su base de datos relacional a un modelo NoSQL de cara a facilitar el escalado y un posible cambio de modelo de datos rápido de cara a cambiar ciertos aspectos del desarrollo de la web.



# Orientado a documentos

## Que les aporta MongoDB

- El modelo de datos flexible les permitía almacenar información jerárquica de forma sencilla y sin necesidad de consultas pesadas sobre la base de datos.
- Las querys y el indexado de información es potente y completo, y la MTV necesitaba poder ofrecer estas dos características de cara al público.
- Facilidad de operaciones, en el sentido de la escalabilidad y la alta disponibilidad. Es relativamente sencillo levantar un nodo y hacerlo formar parte de un ReplicaSet ya existente.



# Orientado a documentos

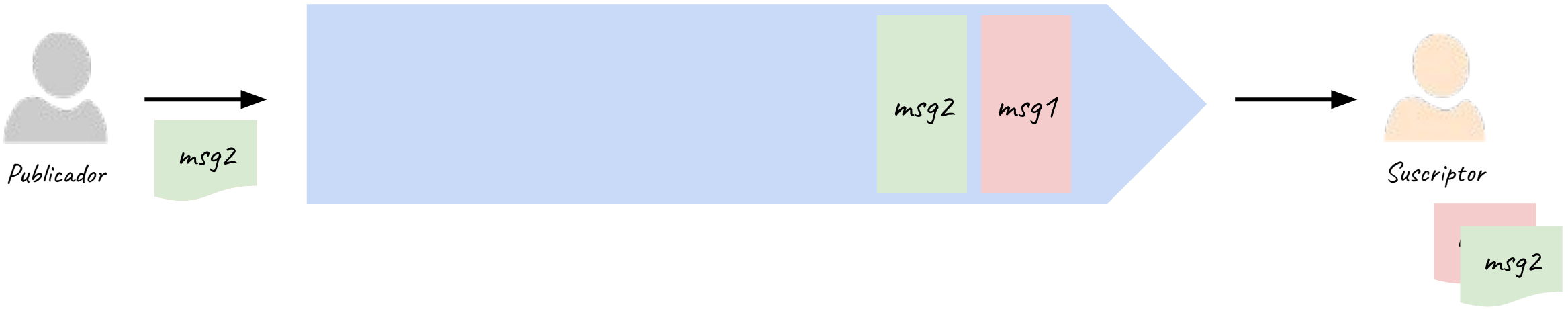
**Otros casos de uso**

- Cualquier aplicación que necesite utilizar datos semiestructurados.
- Aplicaciones con alto volumen de información.
- Sistemas de manejo de documentos y contenido.
- Desarrollo rápido / Metodologías ágiles.
- Datos generados por máquinas (logs, sensores, etc.).



# Colas de Mensajes

# Colas de mensajes



# Colas de mensajes

- Son un mecanismo de comunicación asíncrona entre servicios.
- Son un buffer ligero, que almacena temporalmente mensajes.
- Puntos de enlace que permiten a los componentes de software producir y consumir mensajes.
- Los mensajes suelen ser pequeños y pueden ser cosas como solicitudes, respuestas, mensajes de error o sencillamente información.



# Colas de mensajes

*Publicador / Suscriptor*

- ***Publicador*** (sender), publica o envía un dato o *mensaje*.
- No se envía directamente a *ningún destinatario* y tampoco espera confirmación de lectura.
- El publicador utiliza algún mecanismo para *clasificar* los mensajes que publica.



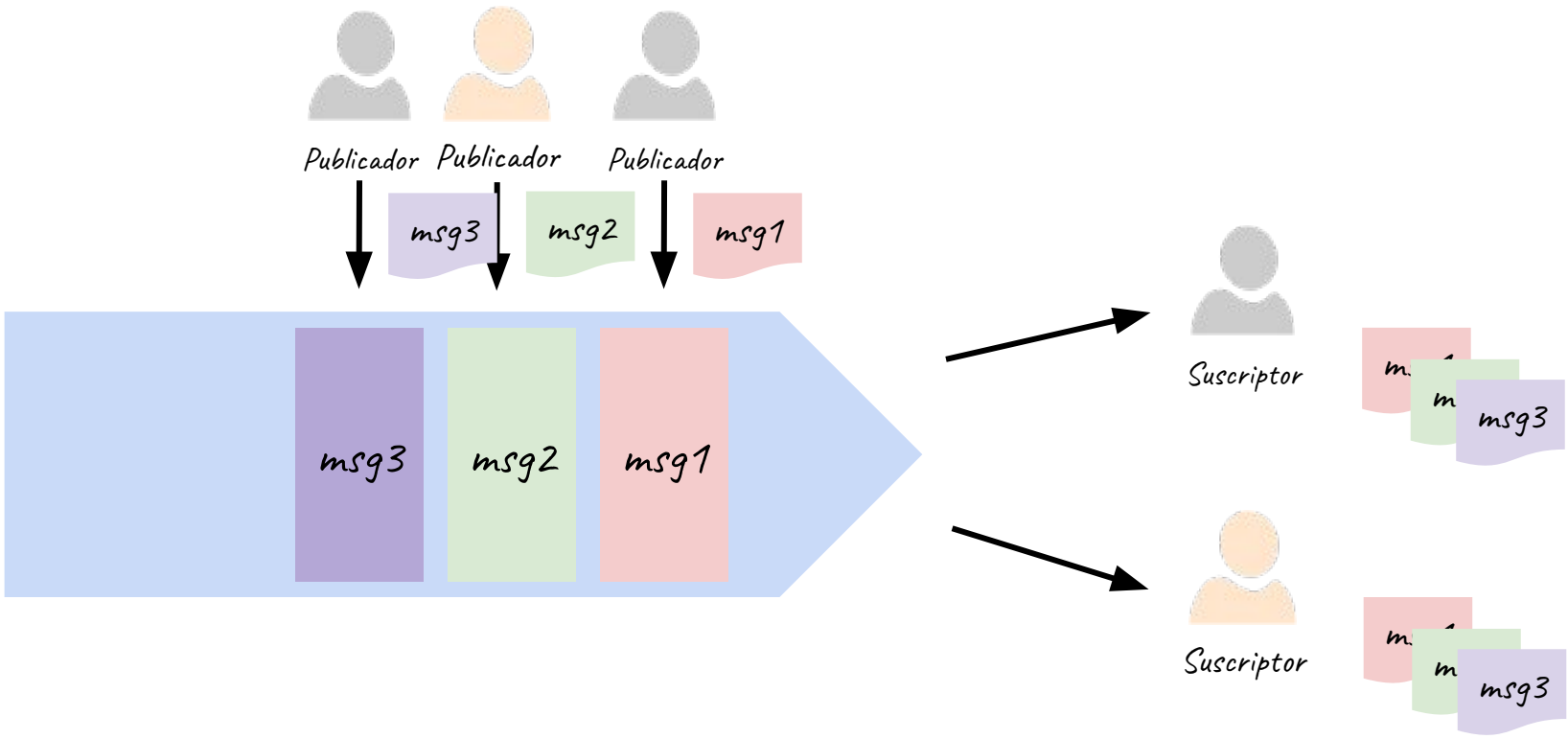
# Colas de mensajes

*Publicador / Suscriptor*

- ***Suscriptor*** (destinatario), se suscribe para recibir ciertos tipos de mensajes y no informa al suscriptor de su lectura.
- Para facilitar la comunicación se dispone de un punto central de comunicación o broker donde se publican los mensajes.
- Comunicación asíncrona.

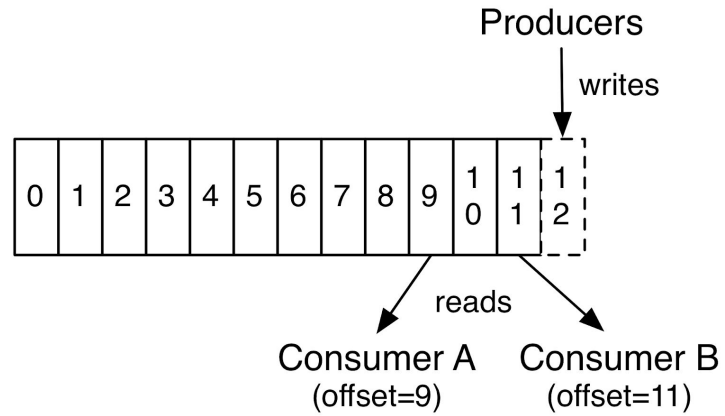


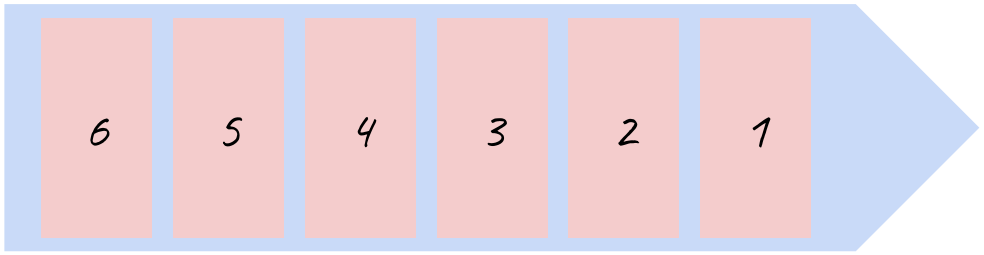
# Colas de mensajes

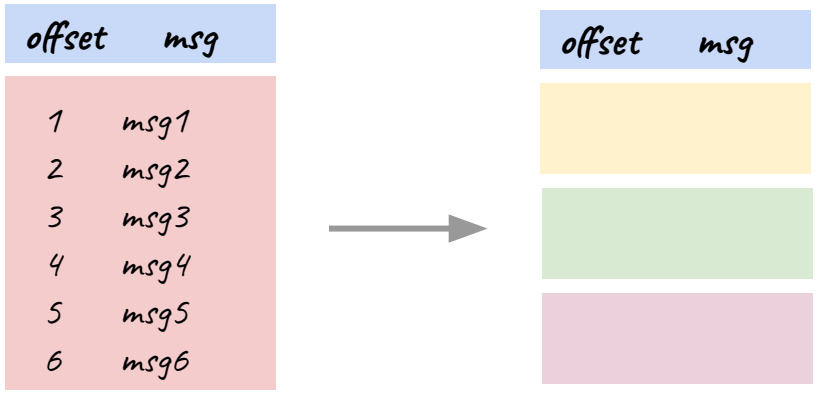


# Colas de mensajes

- Puede haber tantos productores como se quiera escribiendo mensajes en una cola.
- Puede haber tantos consumidores como se quiera leyendo de la cola.
- La cola garantiza que un consumidor sólo lea una vez el mismo mensaje.

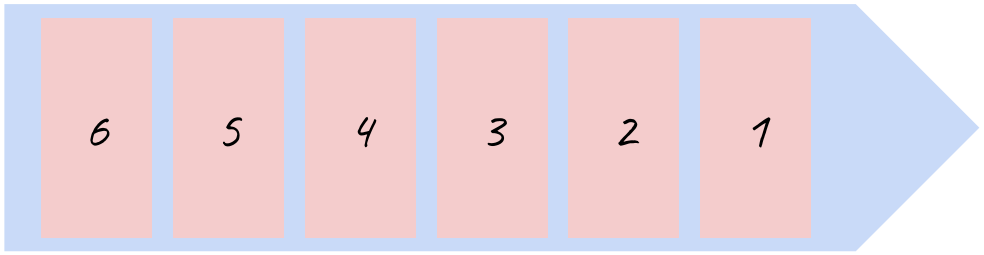


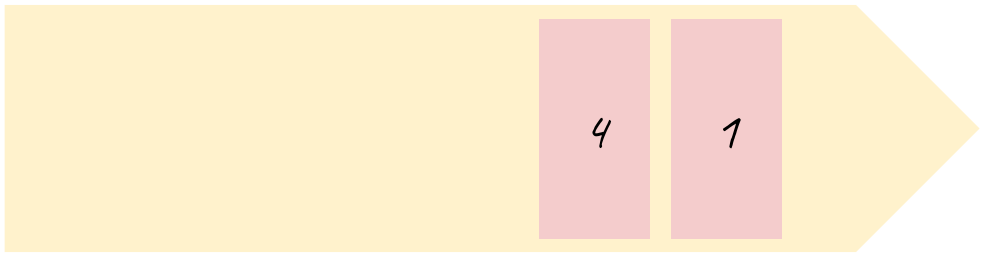

6
5
4
3
2
1

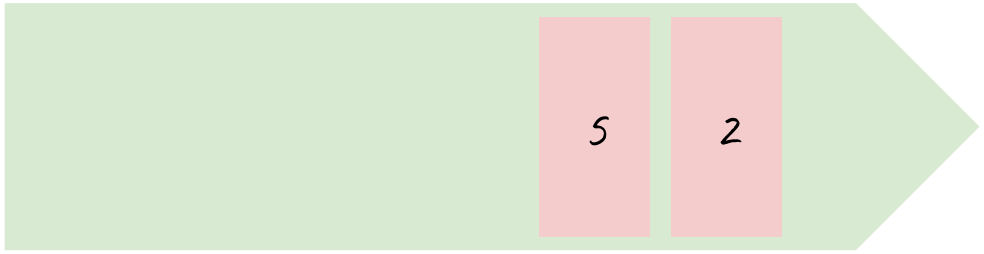

offset msg
1 msg1
2 msg2
3 msg3
4 msg4
5 msg5
6 msg6
offset msg

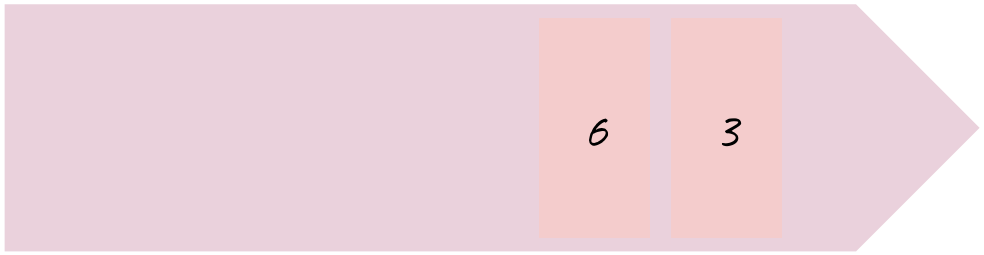
¿Qué pasa con el orden de los mensajes?

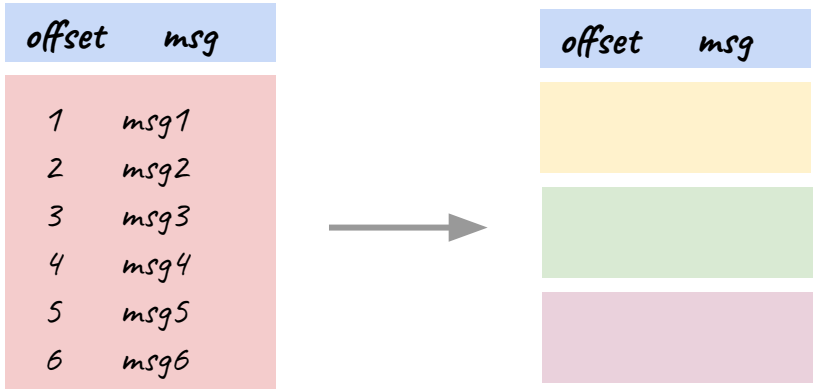
¿Consistencia Secuencial?

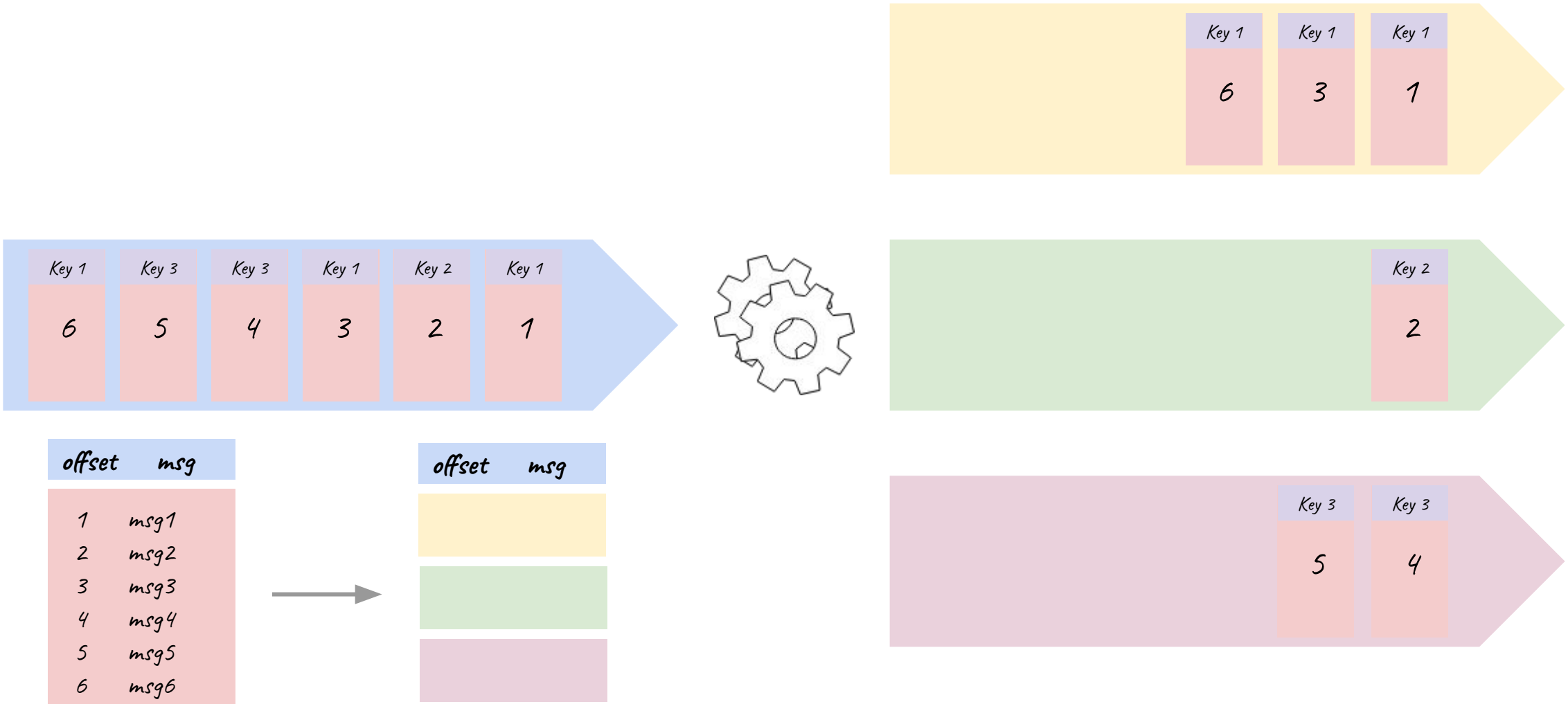





Rafael Garrote Hernández

# Colas de mensajes

*Aplicaciones Distribuidas*

- **Desacoplamiento simplificado**: Los componentes se pueden desarrollar para realizar una función de negocio concreta y sin necesidad de añadir la lógica de comunicación entre servicios, lo que facilita desacoplar servicios de una forma organizada.
- **Escalabilidad detallada**: Al estar todos los servicios desacoplados se puede escalar cada uno por separado según sus necesidades.



# Colas de mensajes

*Aplicaciones Distribuidas*

- **Mejor desempeño**: Permite comunicación asíncrona entre servicios, sin tener que comunicarse entre sí. Ningún servicio queda esperando a que otro consuma el dato.
- **Mayor fiabilidad**: Las colas hacen que los datos sean persistentes, facilitan que si una parte de los servicios se caen, el resto pueda seguir funcionando, lo que aumenta el nivel de tolerancia a fallos.



# Colas de mensajes

*Streaming Platforms*

- **Data stream**: Flujo *continuo* e *infinito* de datos y no en lotes finitos.
- Ofrece la capacidad de publicar y suscribirse a streams de datos.
- Permite procesar los datos *según ocurren*.



# Colas de mensajes

*Distributed Streaming Platforms*

- **Distribuida**: permite procesar los streams de datos de forma paralela en varias máquinas a la vez.
- Almacena los streams de datos de forma duradera y tolerante a fallos.
- Permite la gestión y tratamiento de grandes volúmenes de datos en tiempo real.



# Colas de mensajes

*Fast Data Pipelines*

- Construir pipelines de streaming en tiempo real que transmitan datos entre sistemas o aplicaciones de forma confiable.
- Construir aplicaciones de streaming en tiempo real que transformen o reaccionen a los streams de datos.
- Construir pipelines que consoliden el dato en los diferentes niveles del dato.




# Muchas gracias

datahack
INSPIRING BIG DATA