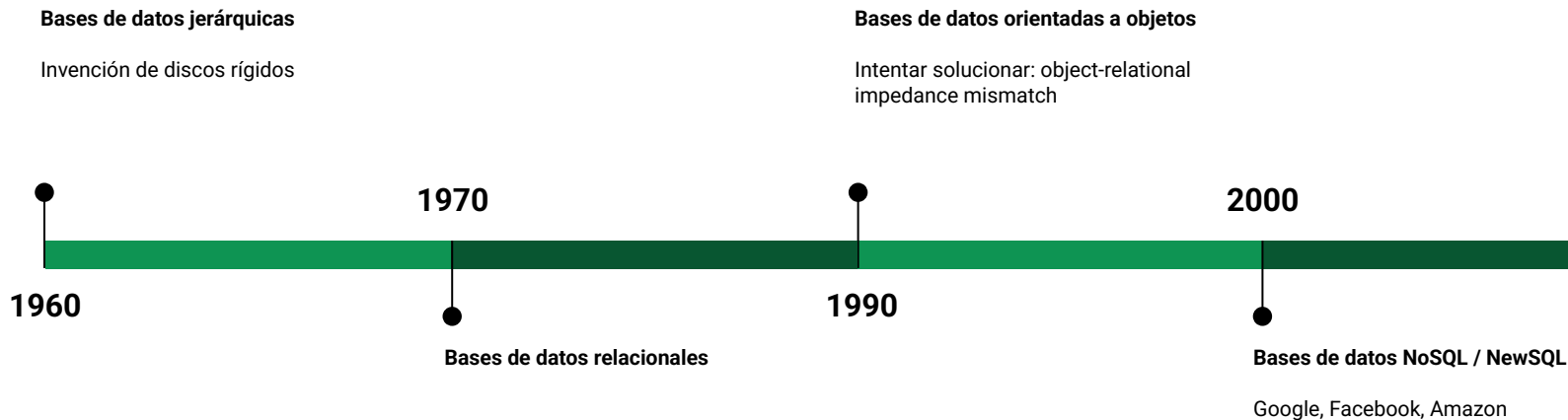
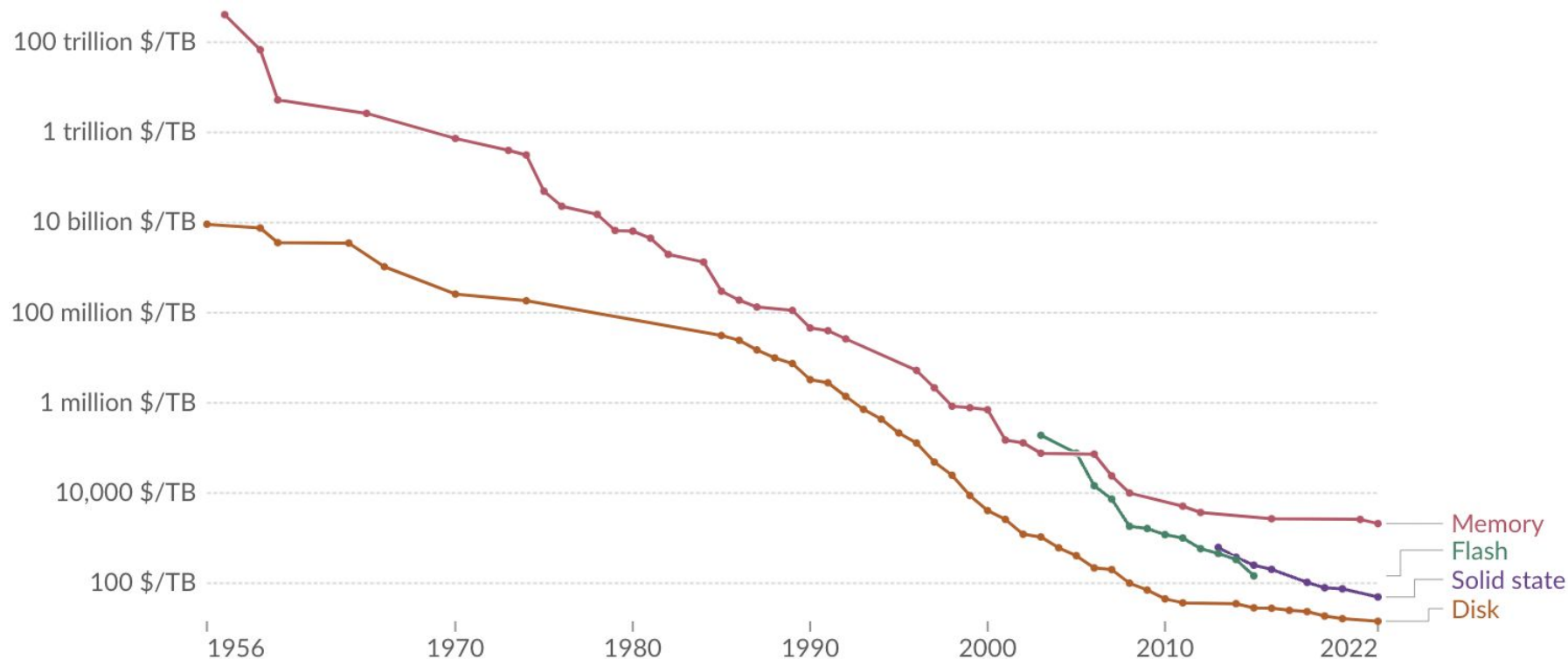


NoSQL

Historia



Costo histórico del almacenamiento



Big data

- Un poco de humo...
- Pero también
- 3 V:
 - Volumen
 - Velocidad
 - Variedad (diversidad)

— — —

Big Data: Volumen + Velocidad

- Sistemas con mucho tráfico y/o volumen
 - Que guardan poco pero de muchos usuarios
 - Que guardan mucho de cada usuario
 - Ej: Comportamiento de usuario

¿Cómo son los requerimientos no funcionales?

Big Data: Volumen + Velocidad (cont.)

- Requerimientos no funcionales
 - Robustez: Soy consistente con el usuario
 - Puedo fallar, pero no silenciosamente
 - Escalabilidad
 - Disponibilidad
 - Performance
- Necesito formas nuevas de almacenar
- Necesito formas nuevas de leer
- Necesito formas nuevas de procesar
- Necesito formas nuevas de escalar
- Necesito high availability
 - Minimizar single point of failures

Escalabilidad

- Se me termina el disco de una máquina
- No doy a basto con el throughput y tiempo de respuesta
- Aumento el hardware (escalabilidad vertical)
 - Es caro
 - Es limitado
 - No resuelve todos los problemas

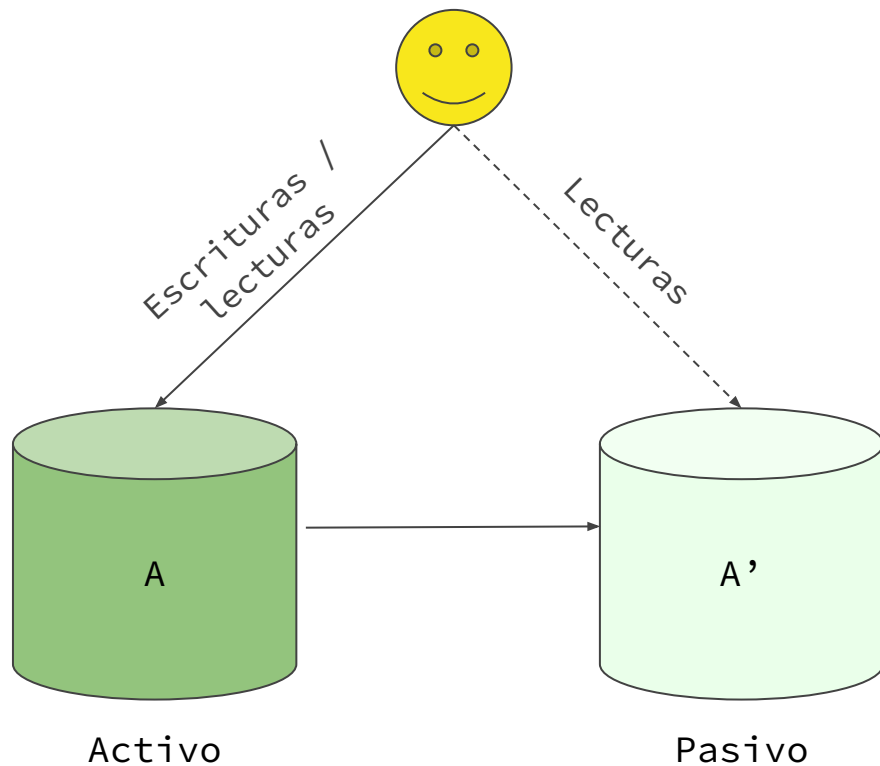
Disponibilidad

- Las instancias se puede caer, no importa que tan buenas sean
- Como hago para no tener single point of failure?
- No puedo escapar de tener más instancias...
- Activo / Pasivo
- Particionamiento
- Sharding
- Replicación

Nota

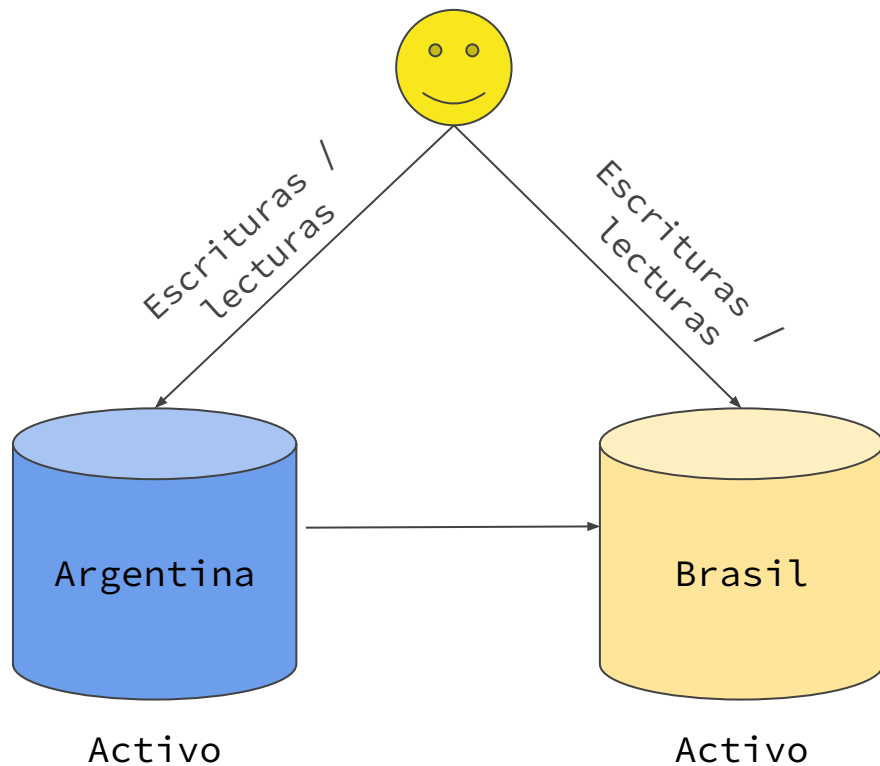
- A partir de acá se mezclan algunos conceptos
- La mayoría de las estrategias mejoran
 - Disponibilidad
 - Escalabilidad

Replicación: Activo pasivo



- Si A se cae o no está disponible puedo switchear a A'
 - Suele hacerse por DNS
- Puedo aprovechar réplicas para lecturas
- Impacto ↓ en escritura
- Impacto ↑ en lectura

Particionamiento



- Puede ser transparente
 - Depende implementación
 - Driver / Cliente
- Suele ir asociado a negocio
- Puedo escalar
 - Procesamiento / Disco
 - Tiempo de respuesta
- Disponibilidad
 - Desacoplo negocios
- Mejora lectura
 - Depende la query
- Approach limitado

Sharding: Hash, Hashmap

- Acceso lineal
- Acceso con hash
- Acceso con hash + módulo N
- Particiones
- Hash distribuído
- Consistent hashing simple
- Consistent hashing + virtual nodes
-

Sharding

- No asociado al negocio
- Permite distribuir la data en N servidores
- Como se implementa?

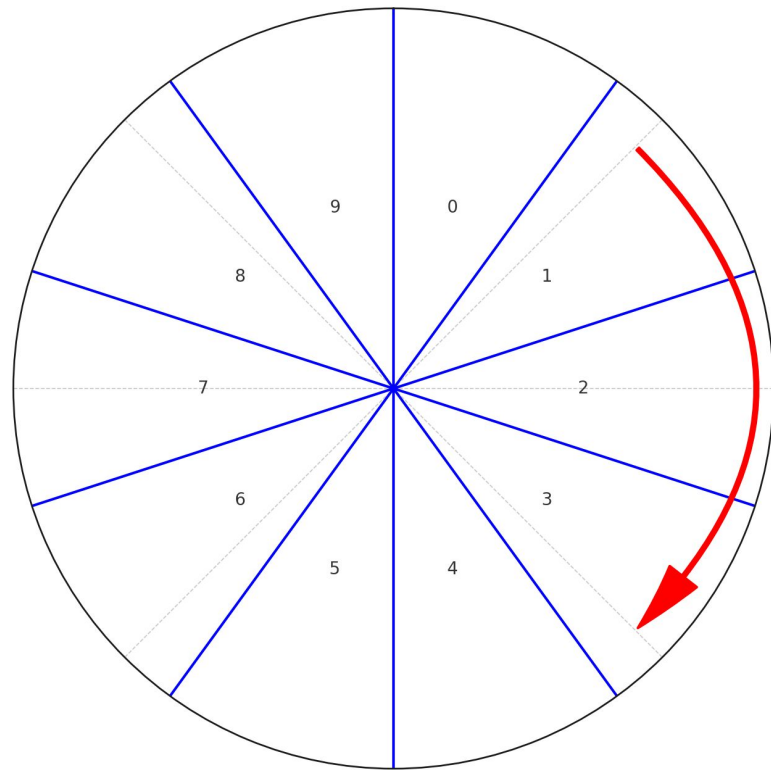
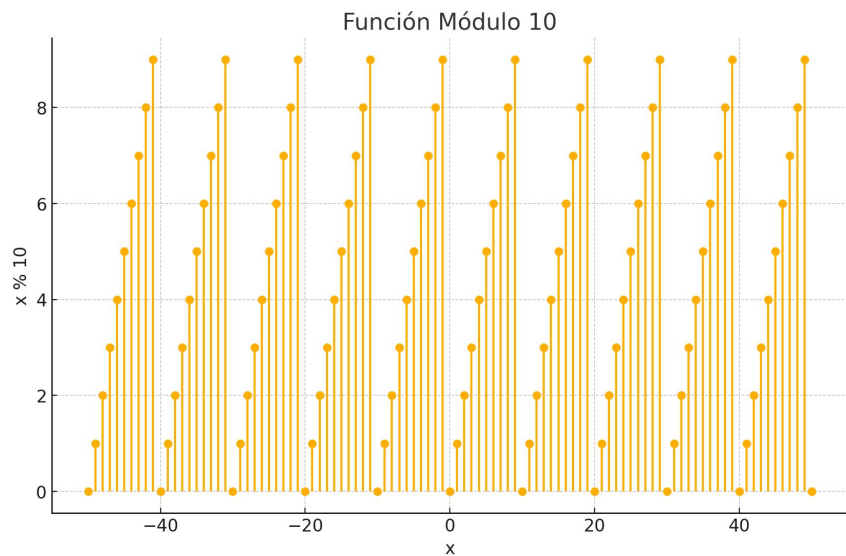
Sharding - 1er approach: hash distribuído

- Cada elemento tiene una key que se puede hashear
 - Ej: último dígito del DNI: 31674167
- Calculo hash módulo N para N servidores
 - Ej: 3 servidores
 - $7 \text{ módulo } 3 = 1$

Servidor 1:	0
Servidor 2:	1
Servidor 3:	2

El elemento va en el servidor 2

Sharding - 1er approach: hash distribuído (cont.)


















Sharding - 1er approach: hash distribuído (cont.)

Servidor 1 (módulo 3 = 0)	Servidor 2 (módulo 3 = 1)	Servidor 3 (módulo 3 = 2)
78461133	14036487	72133562
37179030	76593734	30209598
64597823	92971077	27236772
88450369	94182831	63698738
32921796	52229204	23318035

Sharding - 1er approach: hash distribuído (cont.)

- Que pasa si se cae un servidor?
 - Cambia la función de asignación de servidor
 - Ahora es módulo 2 → Cómo afecta?

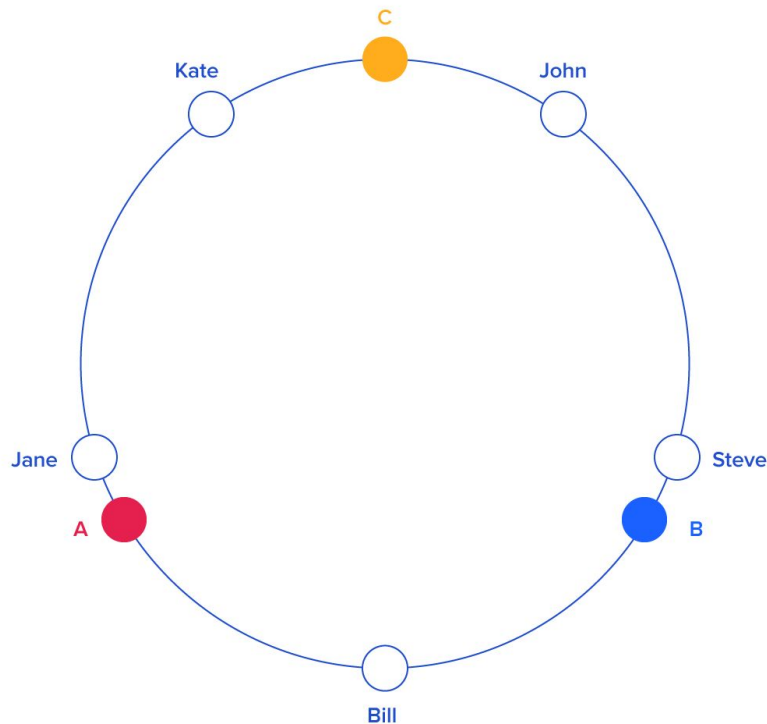
Servidor 1 (módulo 2 = 0)	Servidor 2 (módulo 3 = 1)	Servidor 3 (módulo 2 = 1)
78461133 	14036487 	72133562 
37179030 	76593734 	30209598 
64597823 	92971077 	27236772 
88450369 	94182831 	63698738 
32921796 	52229204 	23318035 

Sharding - 1er approach: hash distribuído (cont.)

- Que pasa si se agrego un servidor?
 - Cambia la función de asignación de servidor
 - Ahora es módulo 4 → Cómo afecta?

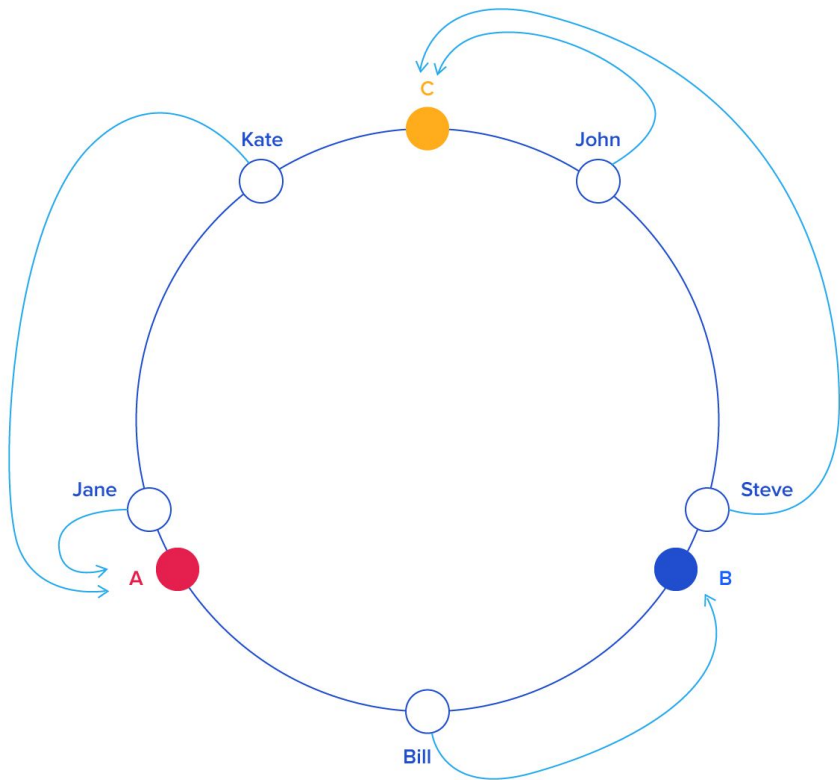
Servidor A (módulo 4 = 0)	Servidor B (módulo 4 = 1)	Servidor C (módulo 4 = 2)	Servidor D (módulo 4 = 3)
78461133 → server D	14036487 → server D	72133562 <input checked="" type="checkbox"/>	78461133
37179030 <input checked="" type="checkbox"/>	76593734 → server A	30209598 → server A	64597823
64597823 → server D	92971077 → server D	27236772 → server B	14036487
88450369 → server B	94182831 <input checked="" type="checkbox"/>	63698738 → server A	92971077
32921796 → server C	52229204 → server A	23318035 → server B	

Sharding: 2do approach: Hash consistente



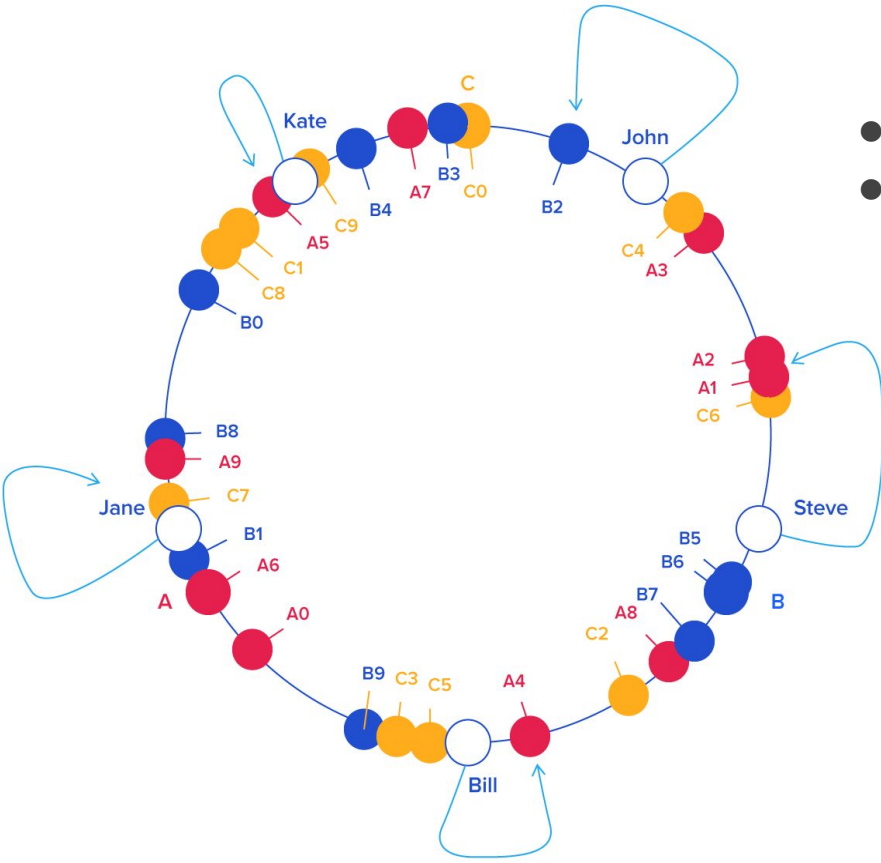
- Ubico no solo los elementos
- Hasheo también los servidores. I.e.
 - `hash(key)`
 - `hash(ip)`
- Cada elemento va al servidor Inmediatamente posterior
 - Sentido horario/antihorario

Sharding: 2do approach: Hash consistente (cont.)



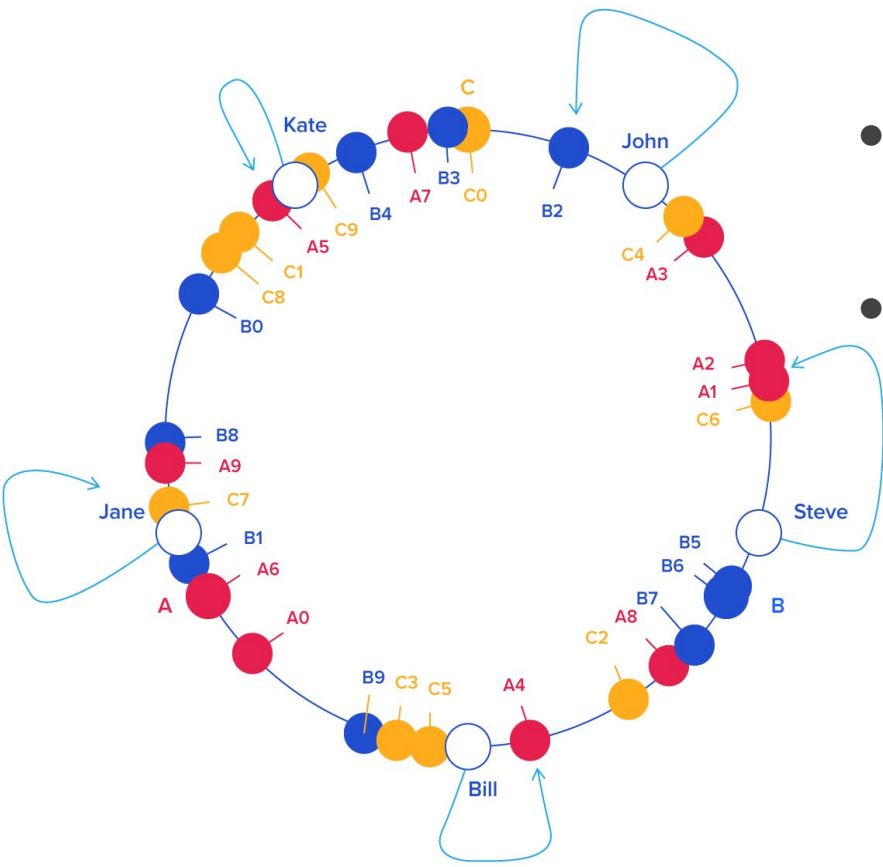
- Cada elemento va al servidor Inmediatamente posterior.
- Si se cae un nodo sólo se afectan los elementos de ese servidor
- Pero todos recaen en el mismo nuevo servidor

Sharding: 3er approach: Hash consistente + virtual nodes



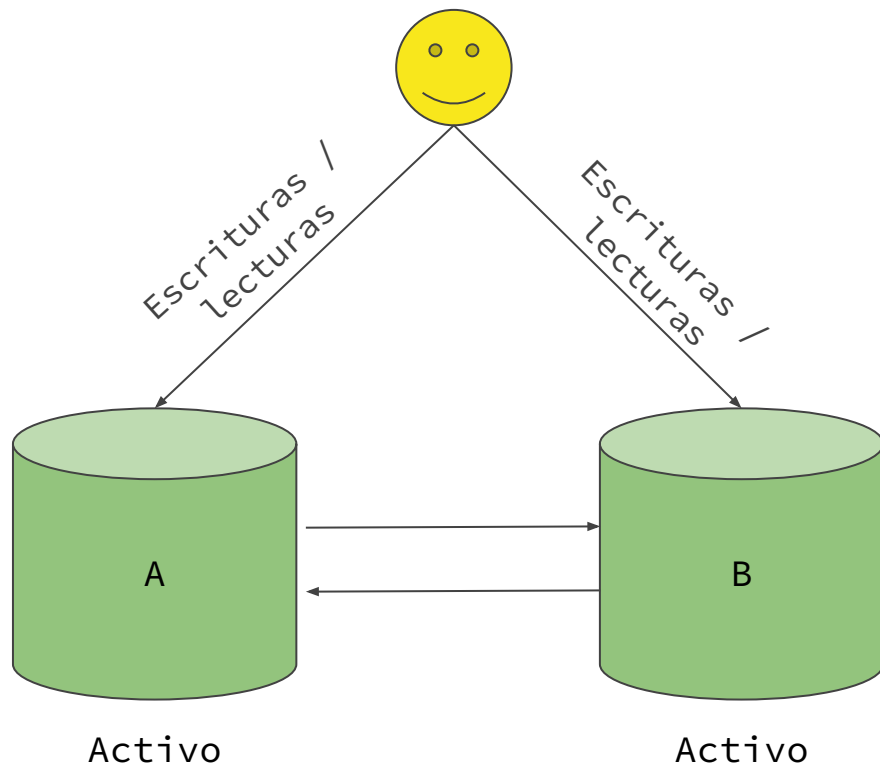
- A cada nodo le doy M virtual nodes
- A cada virtual node le doy un hash

Sharding: 3er approach: Hash consistente + virtual nodes (cont.)



- Si se cae un servidor, distribuirá los elementos entre distintos nodos
- De forma análoga si se agrega

Replicación: Activo - Activo

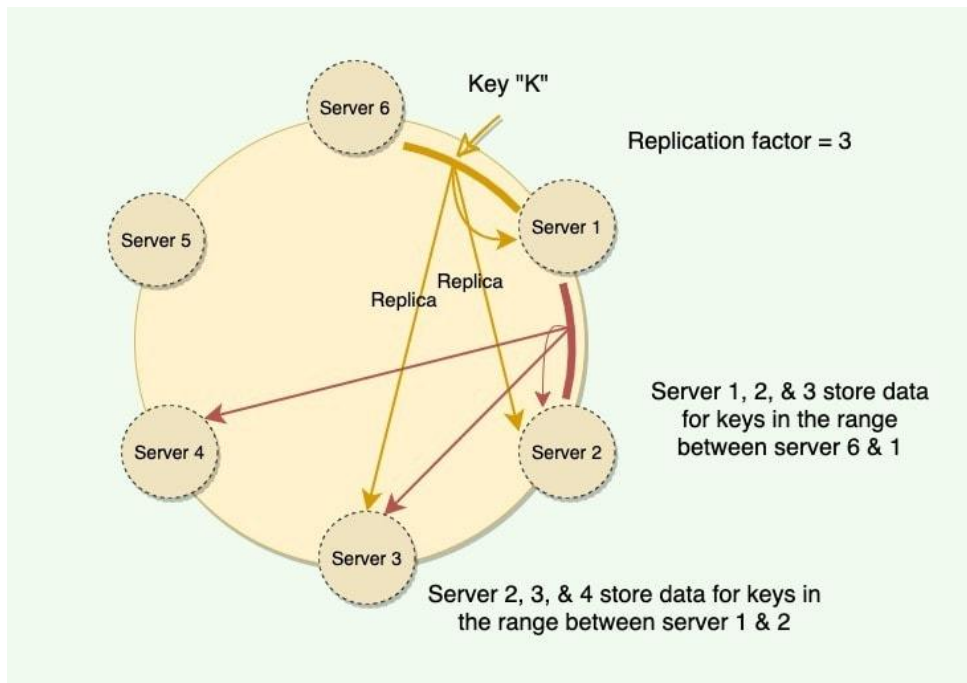


- Puedo escalar y es HA
- Como manejo conflictos?
- A: 23 = Sebastián
- B: 23 = Seba

Replicación: Activo - Activo - Conflictos

- Consenso
 - RAFT
 - Paxos
- Timestamp
 - Guardo y chequeo el más nuevo
- Quorum
 - Necesito $N/2+1$ para dar una respuesta por válida

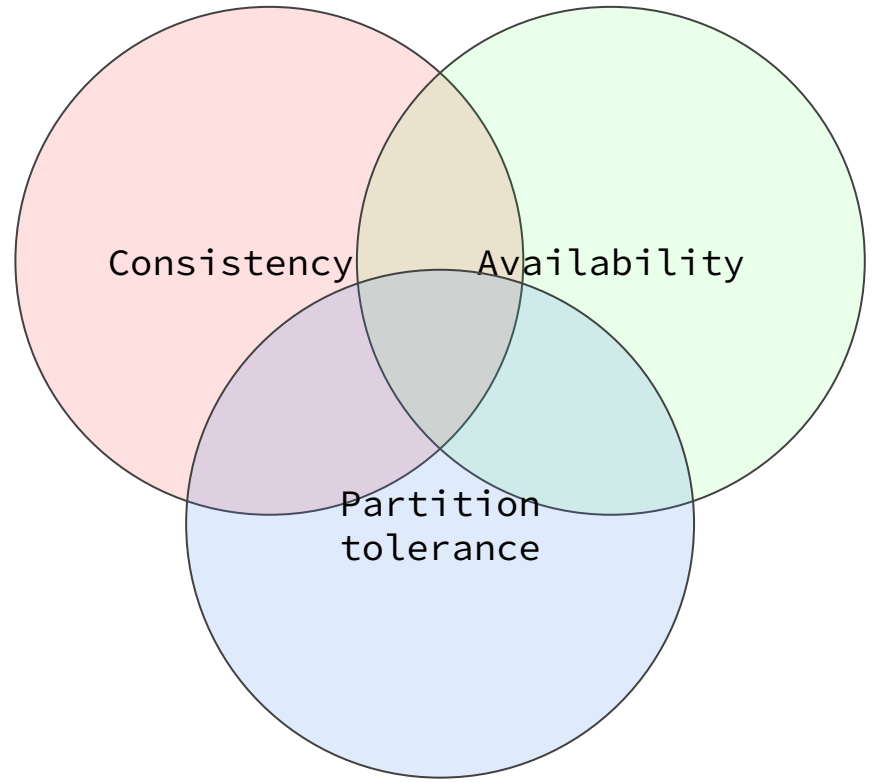
Sharding + replicación



- Un nodo es el coordinador
- Replica en N nodos (replication factor)
- sync o async?
 - Funcionan las 2
 - Con diferente resultado
- De qué nodo puedo leer?

CAP

Brewer's theorem



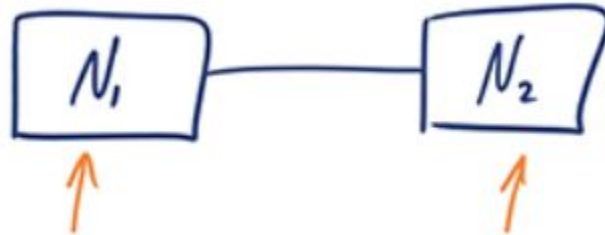
Consistencia

- No tener lecturas fantasmas
- Cada lectura recibe el estado más actualizado o un error



Disponibilidad (Availability)

- Cada request (read or write) recibe una respuesta. No importa que sea con el estado más actualizado o no



Tolerancia a particiones (Partition tolerance)

- El sistema continúa funcionando aunque existan problemas de red (particiones, mensajes perdidos, delay)



ACID vs BASE

ACID

- Atomicity
- Consistency
- Isolation
- Durability

— — —

ACID

— — —

- Atomicidad: La transacción se ejecuta entera o no se ejecuta
- Consistencia: La BD pasa de un estado consistente a otro
- Aislamiento: la transacciones no se afectan entre si
- Durabilidad: una vez que termina una transacción los datos de la misma son almacenados

BASE

- Basic available
- Soft-State
- Eventual consistent

— — —

BASE

— — —

- Basic availability: la BD está disponible la mayor parte del tiempo
- Soft State: el estado de la BD puede cambiar incluso sin entradas
- Eventual consistency: eventualmente los datos son lo mismos en todos los servers

Tipos de bases NoSQL

Key-Value

- Son un mapa. Entro con una key, me da un valor
- Sencillas y de alto rendimiento
 - Suelen almacenar en memoria
- Escalan mucho pero sin mucha flexibilidad
- Ejemplos: Redis, Memcached

ALUMNOS

KEY	VALUE
110354-2	{"nombre":"Rodribuez, Manuel", "anioIngreso":2005, "fechaNacimiento":"01/10/1994"}
110535-5	{"nombre":"Stursi, Marcos", "anioIngreso":2007, "fechaNacimiento":"20/09/1991"}
110536-1	{"nombre":"Scotti, Nahuel", "anioIngreso":2008, "fechaNacimiento":"25/08/1993"}
110537-8	{"nombre":"Martino, Camilo", "anioIngreso":2007, "fechaNacimiento":"26/06/1991"}
110538-3	{"nombre":"Fernandez, Federico", "anioIngreso":2006, "fechaNacimiento":"07/02/1992"}

Wide column

- Los datos se almacenan como un par clave - valor.
 - Al mismo tiempo el valor suele tener otra estructura como un Set, B-Tree, M-Tree, etc.
 - Schemaless en los datos no hasheados/indexados
 - Poca flexibilidad*
 - Alta escalabilidad
-
- Ejemplos: DynamoDB, Cassandra, Bigtable, ScyllaDB

Wide column (cont.)

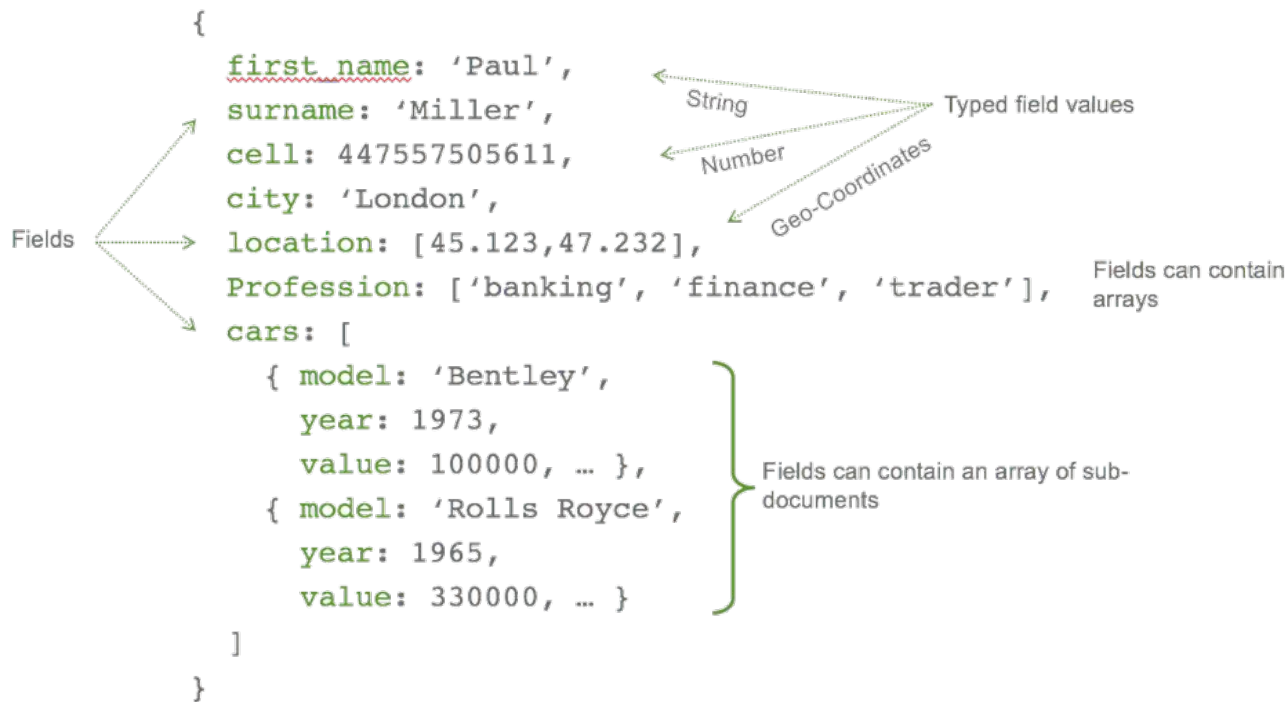
PK (primary/partition key)	SK (sort key)			
SensorID	Timestamp	Temperature	Humidity	Alert
1	4653958801	20	5	
2	1650928601	20	5	
2	1650928602	24	5	
3	5650328211	20	6	
4	7650528471		8	
4	7650528481		9	true

Orientadas a documentos

- Colecciones de documentos
- Estructura jerárquica
- Se permiten hacer queries sobre los datos
- Índices sobre cualquier campo
- Schemaless → Schema on read
- Flexibles → Puedo hacer muchas querys

- Ejemplos: MongoDB, CouchDB, DocumentDB, Elasticsearch

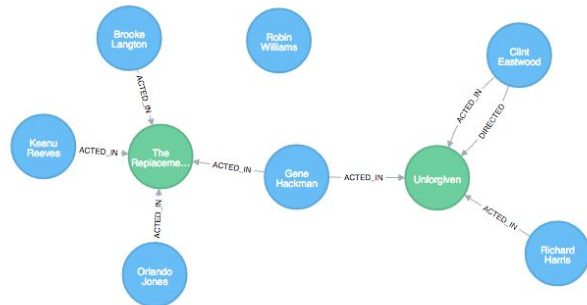
Orientadas a documentos (cont.)



Orientadas a grafos

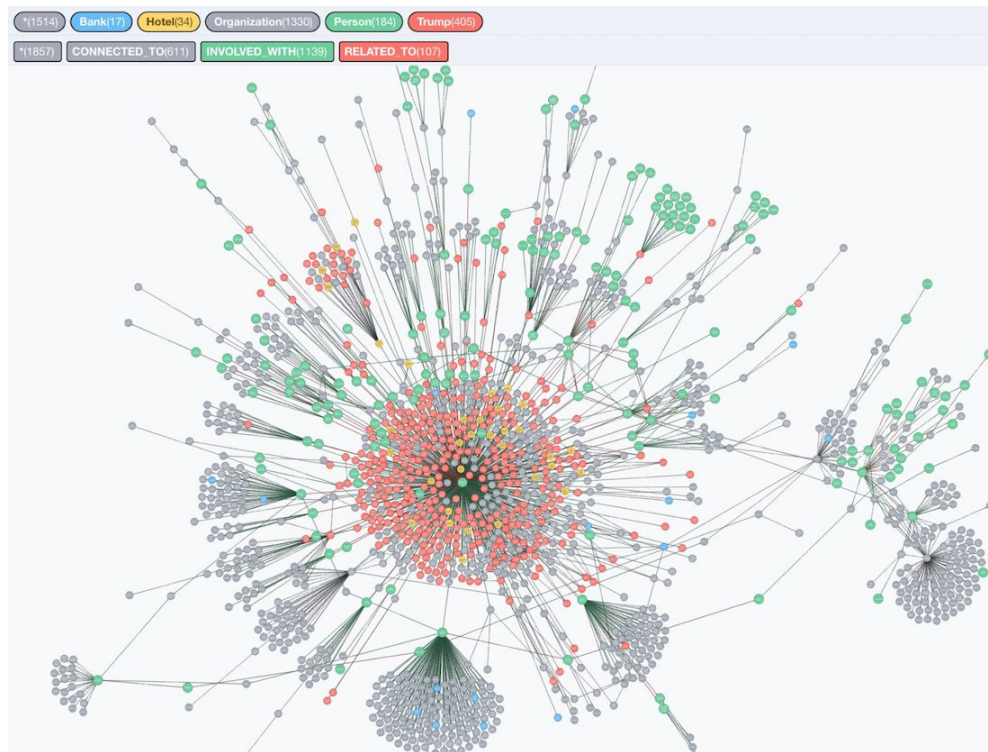
- Las entidades base son nodos y sus relaciones (cada uno con sus propiedades)
- Permite ejecutar queries semánticas
- Altamente flexibles
- Sin sharding
- Ejemplos: Neo4J, Titan

Orientadas a grafos (cont.)



Solution: Gene Hackman and not Robin Williams

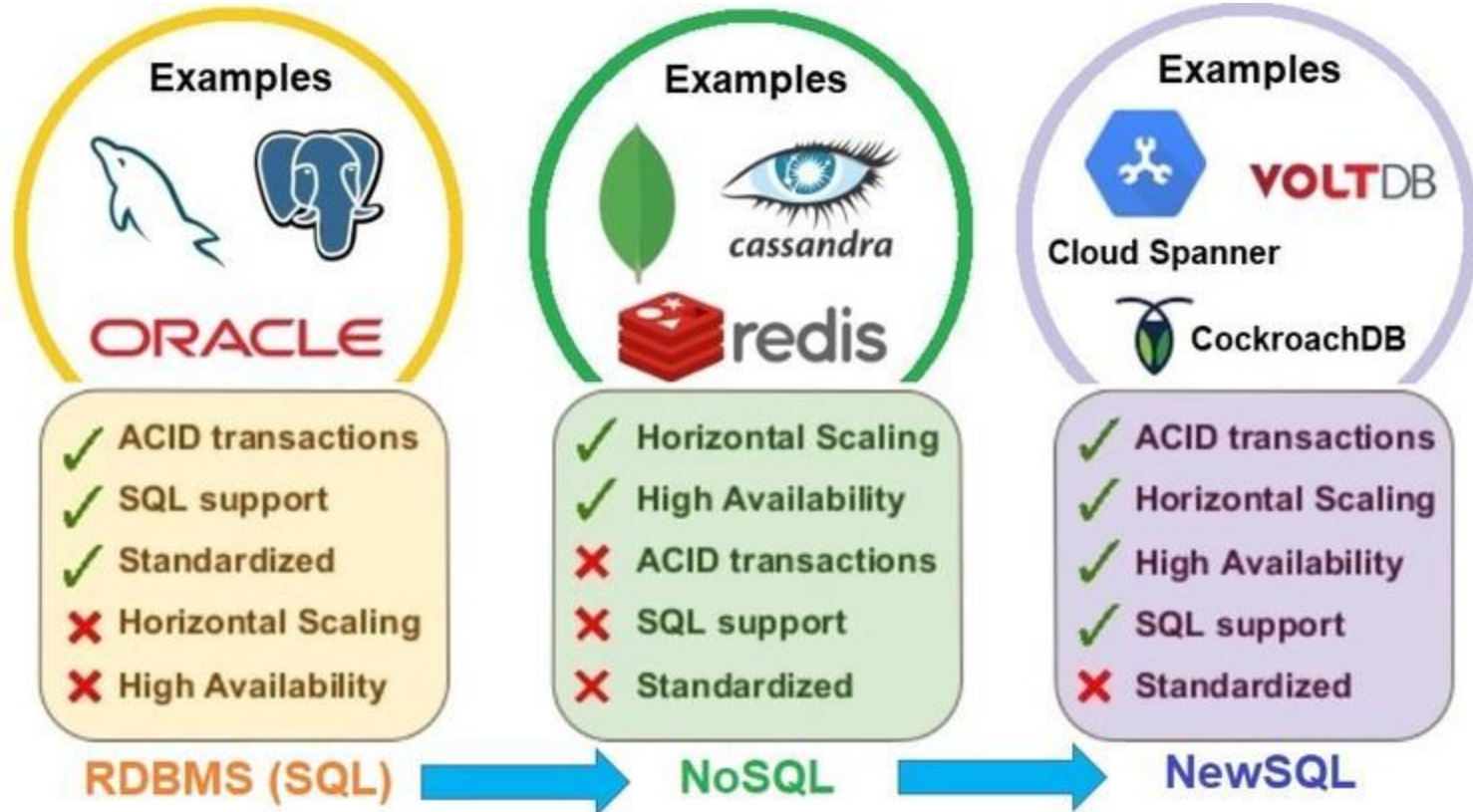
```
MATCH (gene:Person {name:"Gene Hackman"})-[:ACTED_IN]->(movie:Movie),
      (other:Person)-[:ACTED_IN]->(movie),
      (robin:Person {name:"Robin Williams"})
WHERE NOT (robin)-[:ACTED_IN]->(movie)
RETURN DISTINCT other
```



Displaying 1514 nodes, 1857 relationships.

NewSQL

NewSQL



Qué elegir?

- Analizar carga, patrones de acceso.
 - Tiene que justificar sacrificar ACID.
- OLAP vs OLTP
 - No hay una bala de plata para todos los escenarios
 - Priorizar siempre lo transaccional
 - Analytics
 - Queries de mucha data, en general no requiere lo del último día
 - Puedo ir a otra DB
 - Secondary, BigQuery
 - ETL

Preguntas ?

Referencias

- <https://www.toptal.com/big-data/consistent-hashing>
- <https://ourworldindata.org/grapher/historical-cost-of-computer-memory-and-storage>
-