



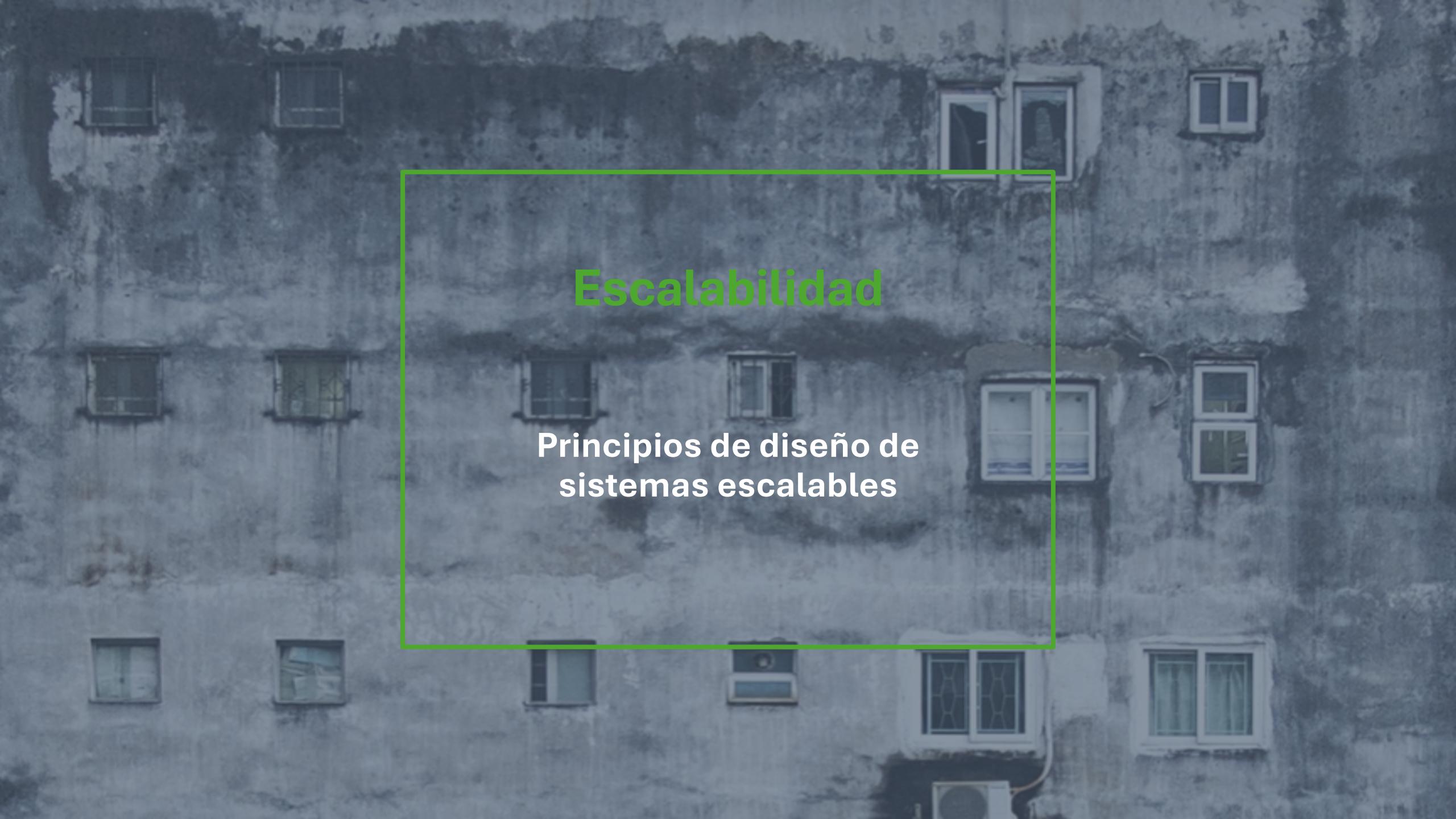
PONTIFICIA
UNIVERSIDAD
CATÓLICA
DE CHILE

IIC3103

Taller de Integración

Profesores

Arturo Tagle / Daniel Darritchon



Escalabilidad

Principios de diseño de
sistemas escalables

TEMARIO

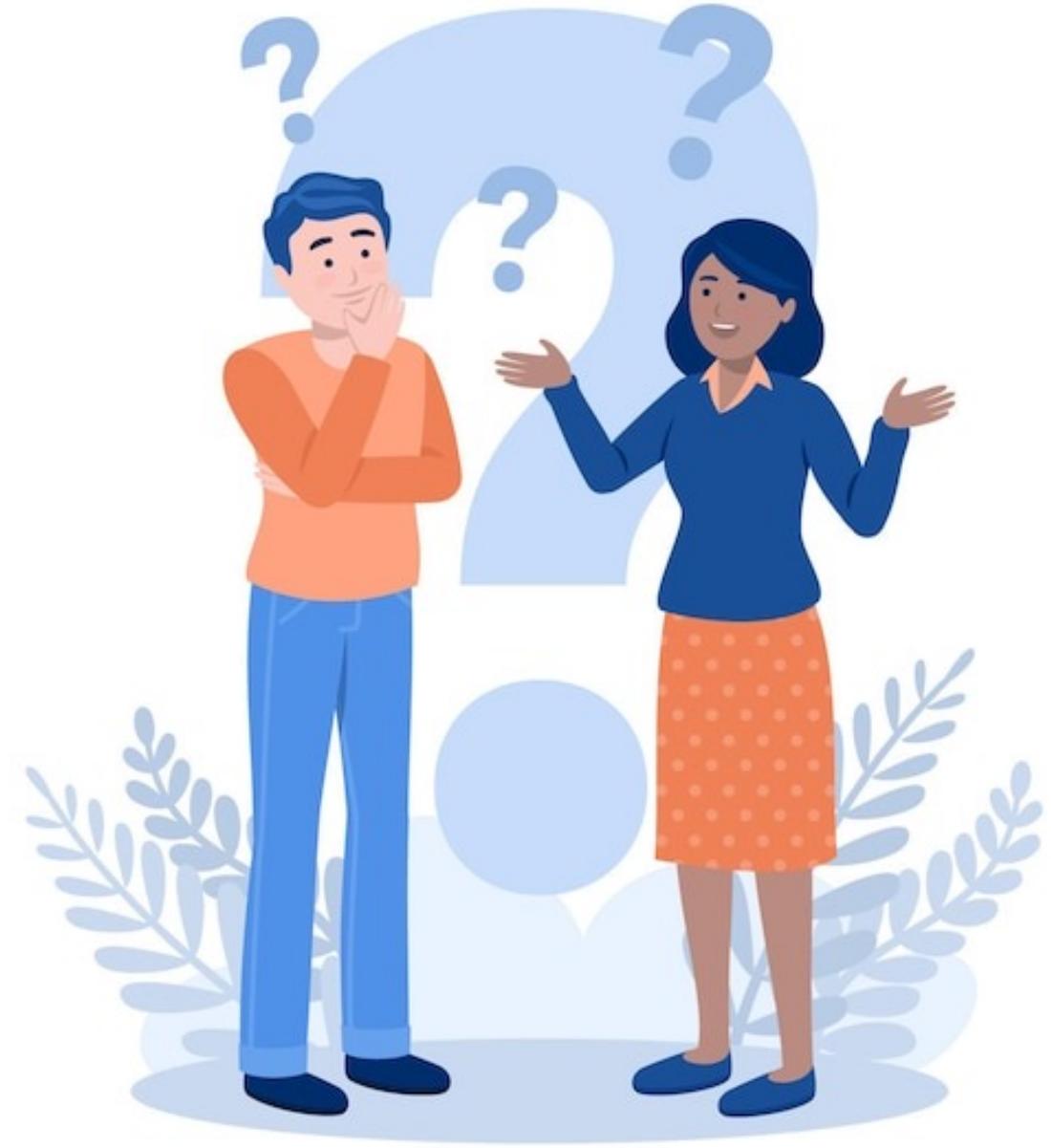
- 1. Introducción**
- 2. Escalabilidad de servidores**
 1. Horizontal
 2. Vertical
 3. Balanceo de carga
- 3. Escalabilidad de datos**
 1. Sharding
 2. CQRS
- 4. Consistencia eventual**
- 5. Teorema CAP y PACELC**



¿Qué es la escalabilidad?



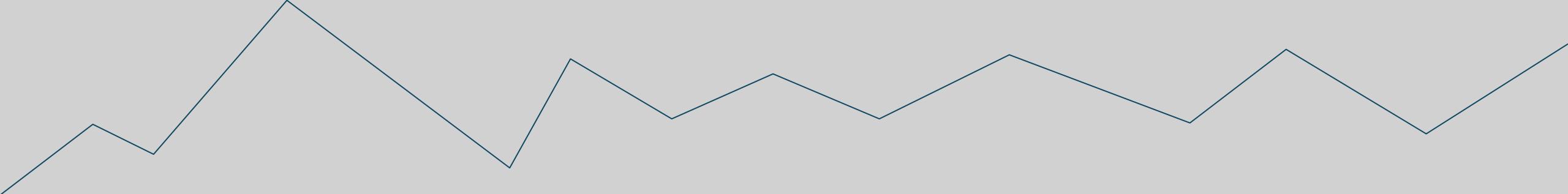
- Es la capacidad de un sistema de agregar recursos a medida que la demanda aumenta sin comprometer el *performance* de este.
- El desafío de la escalabilidad es hacer este proceso eficiente, por medio de estrategias que sea efectivas en costo.



¿Puede ser un sistema
infinitamente
escalable?

Escalando servidores

¿Qué pasa cuando la carga aumenta?





Mauricio Vega



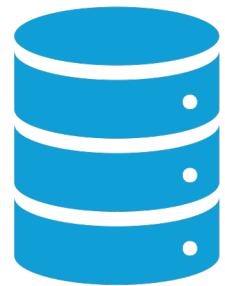
Mauricio Vega



**potencia
máquinas**

máquinas

**potencia
máquinas**



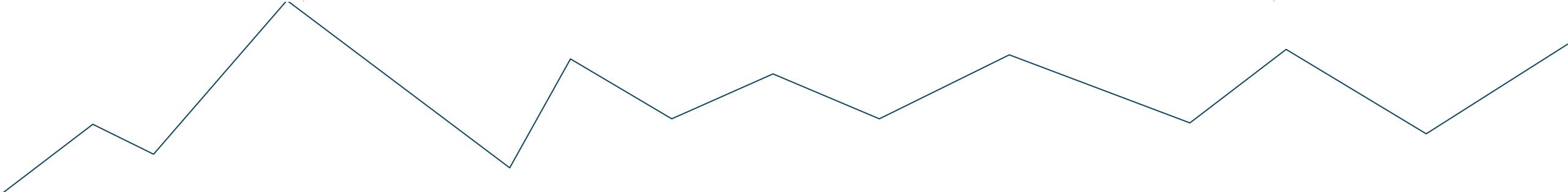
Servidores
bases de
datos

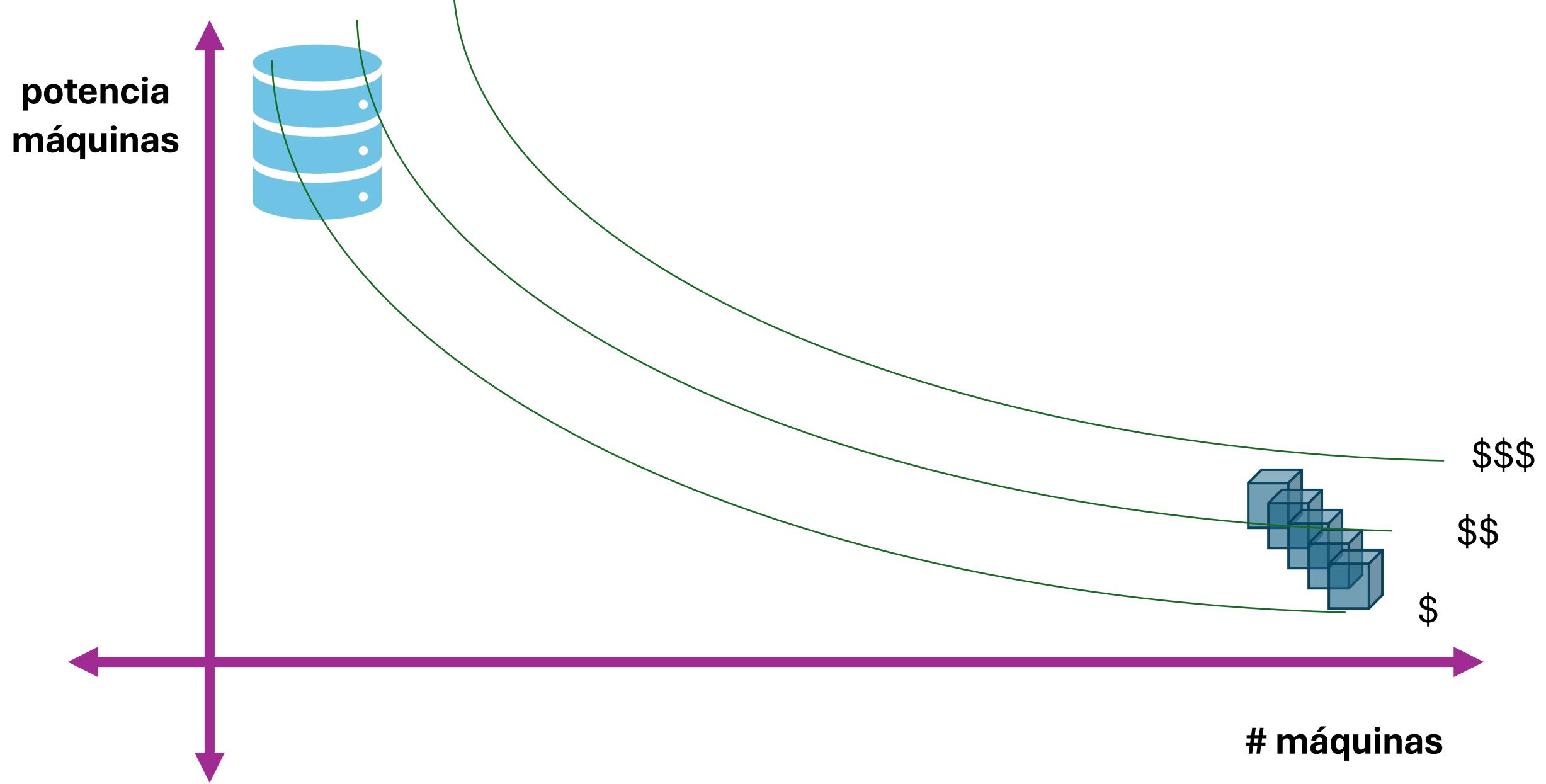


Servidores
web



Nombre de la instancia	Tarifa por hora bajo demanda	vCPU	Memoria	Almacenamiento	Rendimiento de la red
t4g.nano	0,0042 USD	2	0,5 GiB	Solo EBS	Hasta 5 gigabits
t4g.micro	0,0084 USD	2	1 GiB	Solo EBS	Hasta 5 gigabits
t4g.small	0,0168 USD	2	2 GiB	Solo EBS	Hasta 5 gigabits
t4g.medium	0,0336 USD	2	4 GiB	Solo EBS	Hasta 5 gigabits
t4g.large	0,0672 USD	2	8 GiB	Solo EBS	Hasta 5 gigabits
t4g.xlarge	0,1344 USD	4	16 GiB	Solo EBS	Hasta 5 gigabits
t4g.2xlarge	0,2688 USD	8	32 GiB	Solo EBS	Hasta 5 gigabits





Escalabilidad de servidores

Horizontal

- Se requiere un balanceador de carga

Vertical

- Comunicación directa

Escalabilidad de servidores

Horizontal

- Se requiere un balanceador de carga
- Mayor resiliencia

Vertical

- Comunicación directa
- Punto único de falla

Escalabilidad de servidores

Horizontal

- Se requiere un balanceador de carga
- Mayor resiliencia
- Llamadas entre servidores son a través de red

Vertical

- Comunicación directa
- Punto único de falla
- Comunicación entre procesos

Escalabilidad de servidores

Horizontal

- Se requiere un balanceador de carga
- Mayor resiliencia
- Llamadas entre servidores son a través de red
- Inconsistencia de datos (consistencia eventual)

Vertical

- Comunicación directa
- Punto único de falla
- Comunicación entre procesos
- Consistencia de datos

Escalabilidad de servidores

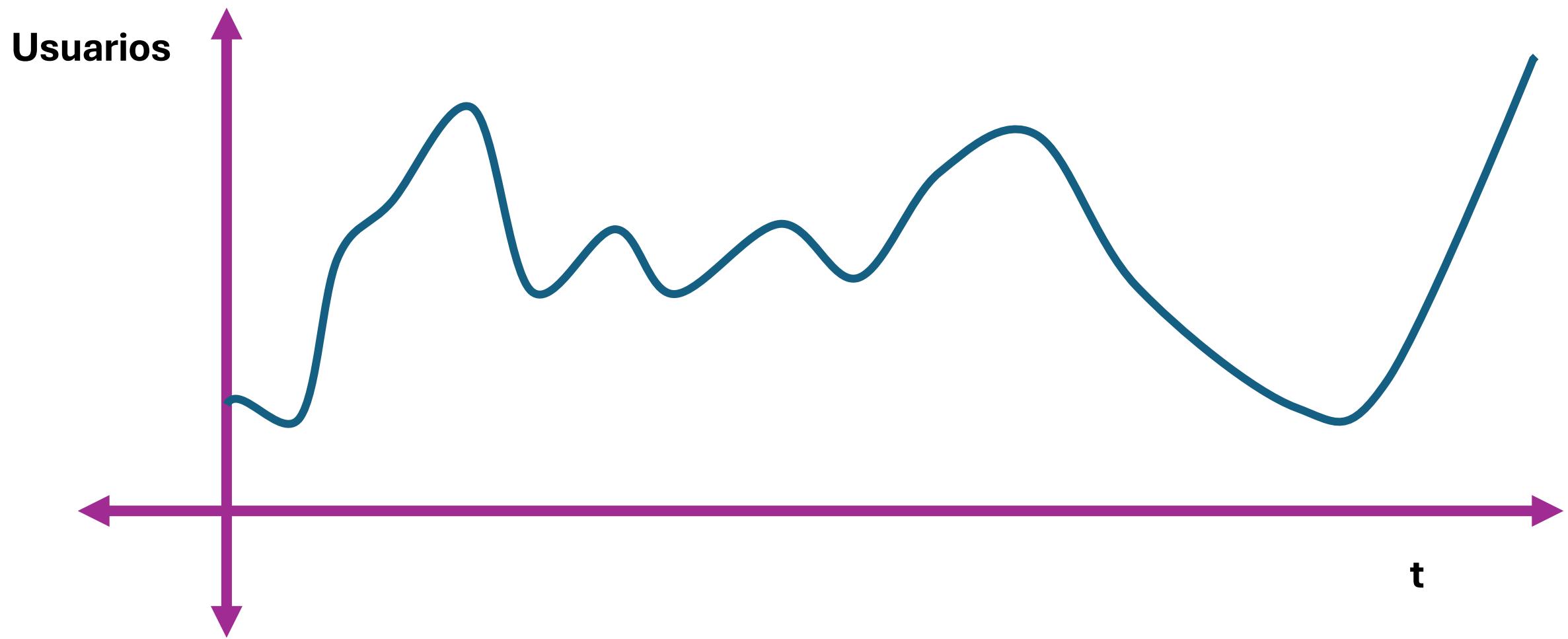
Horizontal

- Se requiere un balanceador de carga
- Mayor resiliencia
- Llamadas entre servidores son a través de red
- Inconsistencia de datos (consistencia eventual)
- Escala bien a medida que aumentan usuarios

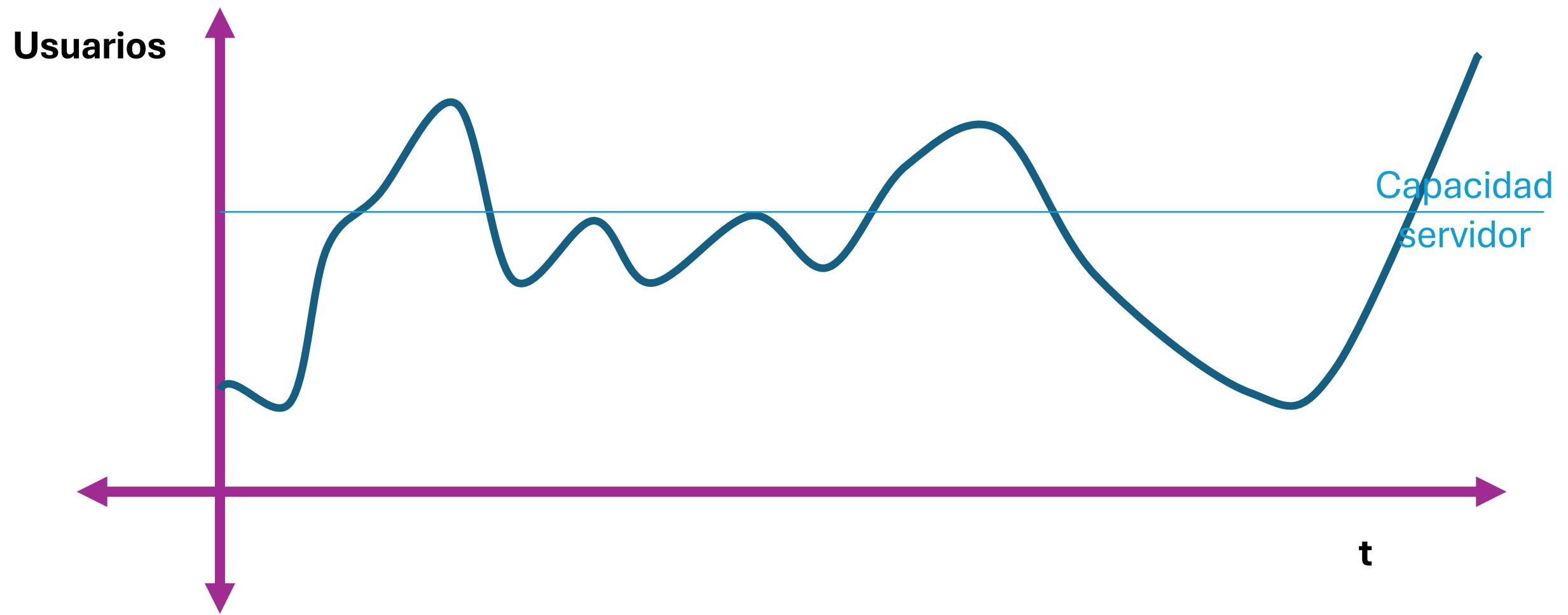
Vertical

- Comunicación directa
- Punto único de falla
- Comunicación entre procesos
- Consistencia de datos
- Límite de hardware

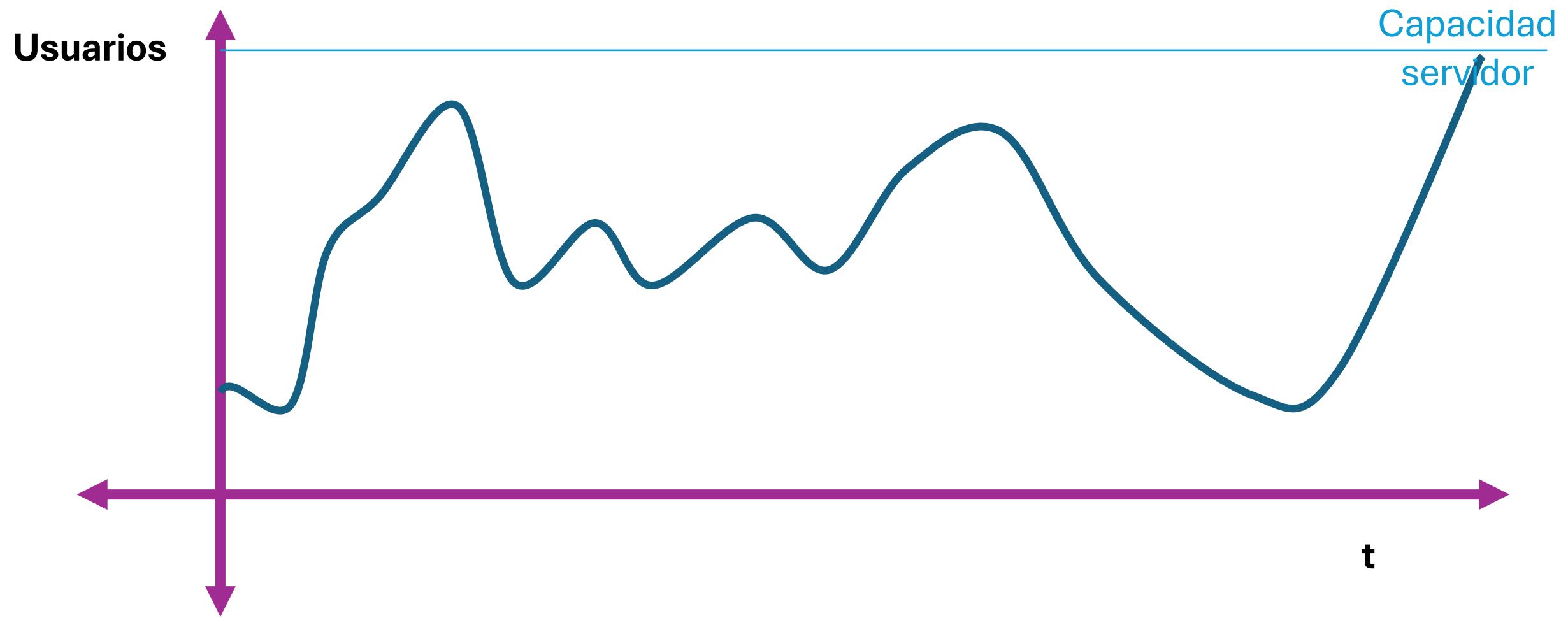
Desafío: Encontrar el óptimo



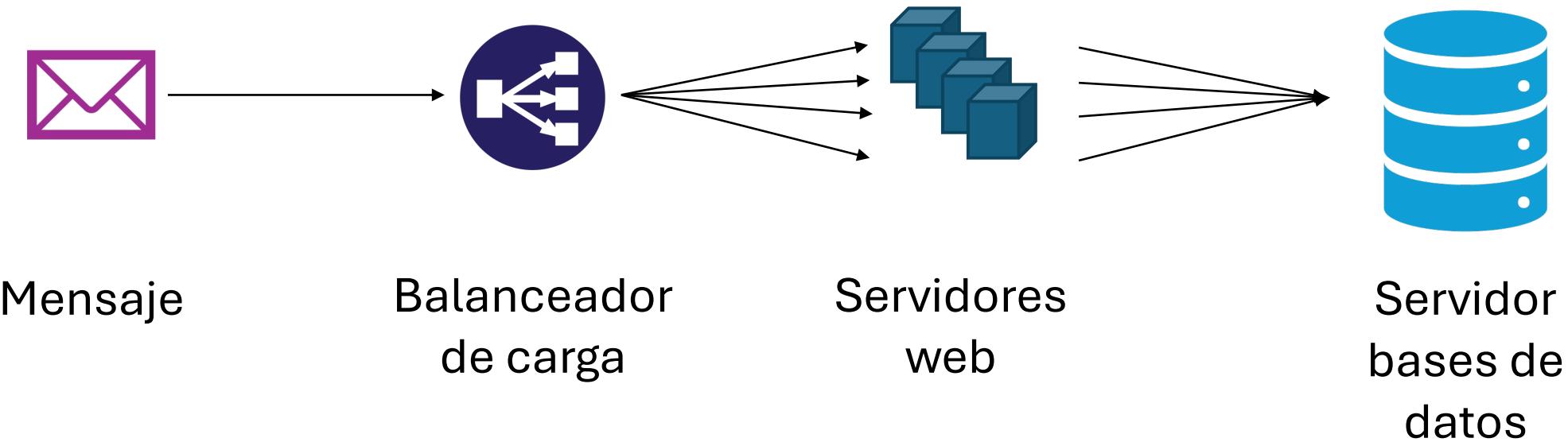
Desafío: Encontrar el óptimo



Desafío: Encontrar el óptimo



Solución común: Arquitectura híbrida



¿A QUÉ SERVIDOR SE VA LA CARGA?

Sticky session

Mismo usuario – Mismo servidor (a través de cookie con id de servidor)

Round-robin

Requests siempre llega al siguiente servidor en la lista del pool de servidores: A -> B -> C -> D -> A ...

Programación según carga

Request se envía al servidor con menor carga

Aleatorio

Request se envía a algún servidor en forma aleatoria



Balanceando el balanceador: DNS rotation

```
➔ integracion git:(develop) ✘ nslookup google.com
Server:      200.30.192.14
Address:     200.30.192.14#53

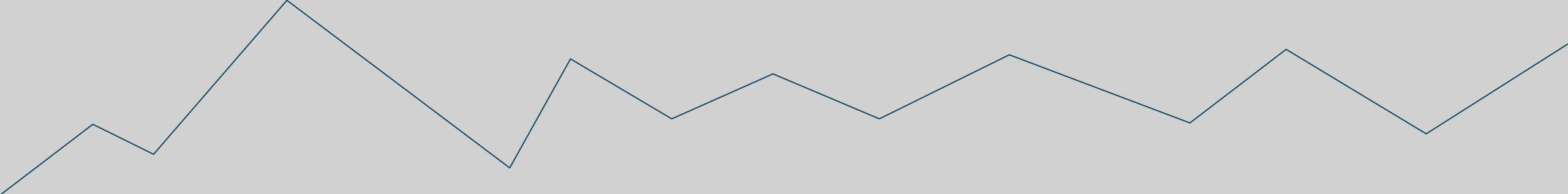
Non-authoritative answer:
Name:   google.com
Address: 64.233.190.113
Name:   google.com
Address: 64.233.190.100
Name:   google.com
Address: 64.233.190.102
Name:   google.com
Address: 64.233.190.101
Name:   google.com
Address: 64.233.190.101
Name:   google.com
Address: 64.233.190.139
Name:   google.com
Address: 64.233.190.138
```

```
➔ integracion git:(develop) ✘ nslookup google.com
Server:      200.30.192.14
Address:     200.30.192.14#53

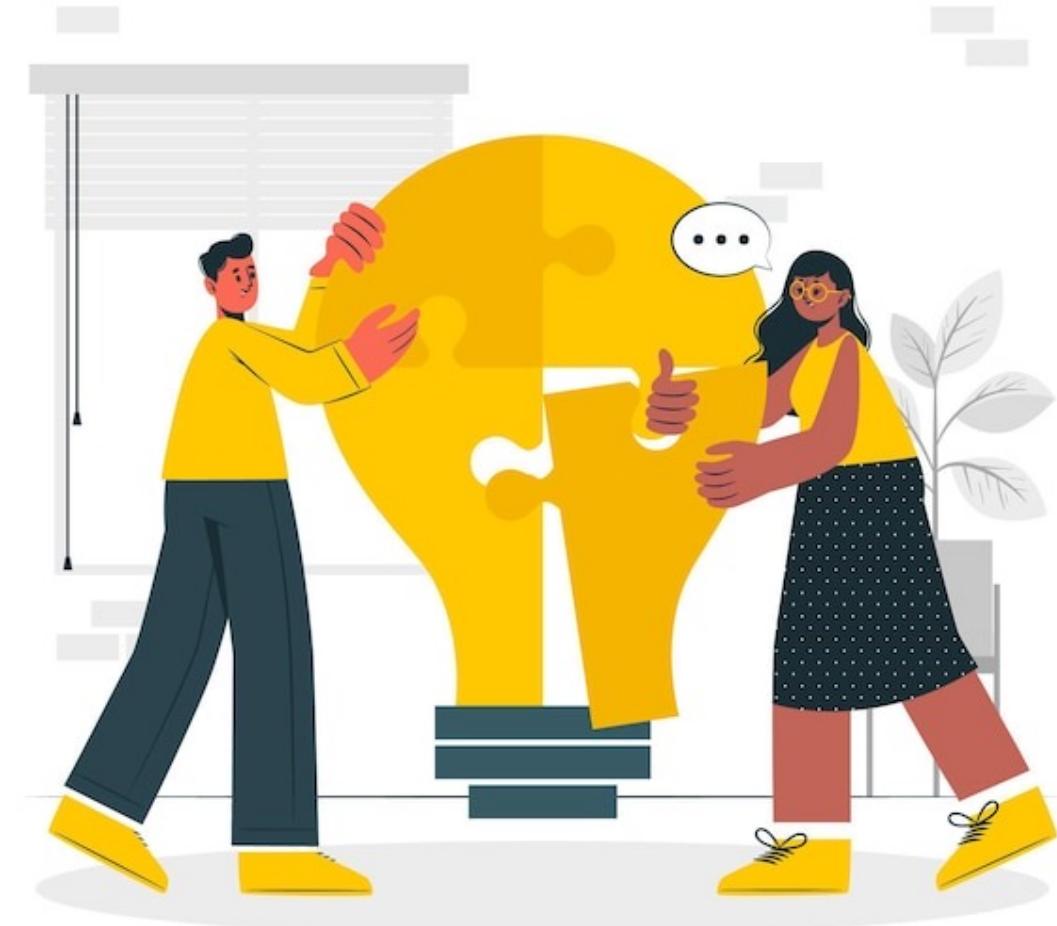
Non-authoritative answer:
Name:   google.com
Address: 64.233.190.102
Name:   google.com
Address: 64.233.190.101
Name:   google.com
Address: 64.233.190.101
Name:   google.com
Address: 64.233.190.139
Name:   google.com
Address: 64.233.190.138
Name:   google.com
Address: 64.233.190.113
Name:   google.com
Address: 64.233.190.100
```

Escalando los datos

¿Qué pasa cuando mi base de datos no puede ser creciendo?



ESCALANDO LA BASE DE DATOS: PARTICIONES

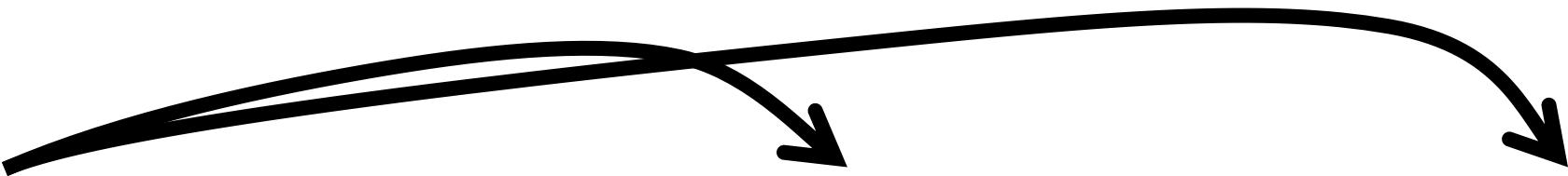


PARTICIONANDO LA BASE DE DATOS

La partición consiste en dividir los datos de una base de datos en dos o más partes de menor tamaño.



Particiones verticales u horizontales



Original Table

CUSTOMER ID	FIRST NAME	LAST NAME	FAVORITE COLOR
1	TAEKO	OHNUKI	BLUE
2	O.V.	WRIGHT	GREEN
3	SELD A	BAĞCAN	PURPLE
4	JIM	PEPPER	AUBERGINE

Vertical Partitions

VP1

CUSTOMER ID	FIRST NAME	LAST NAME
1	TAEKO	OHNUKI
2	O.V.	WRIGHT
3	SELD A	BAĞCAN
4	JIM	PEPPER

VP2

CUSTOMER ID	FAVORITE COLOR
1	BLUE
2	GREEN
3	PURPLE
4	AUBERGINE

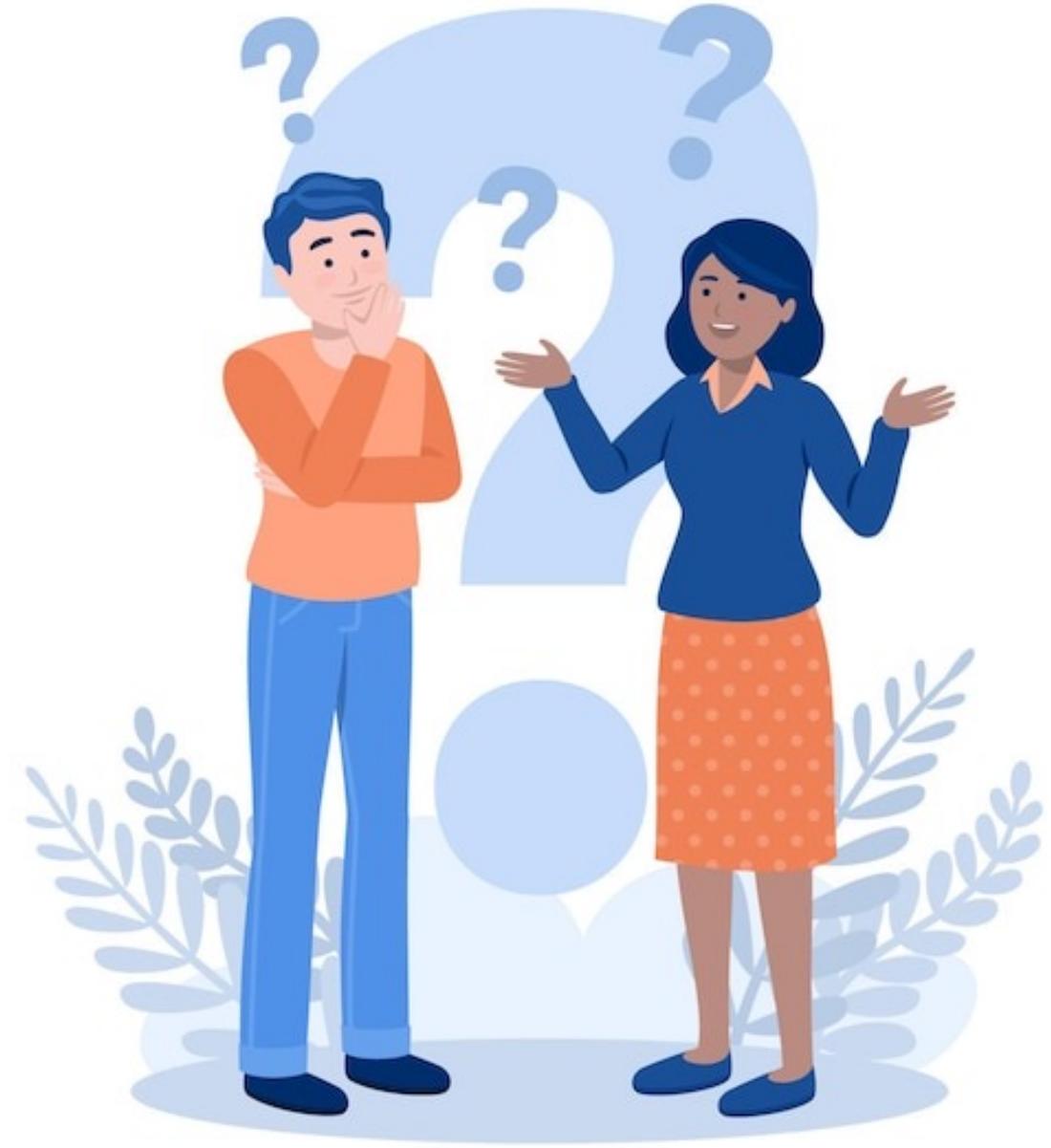
HP1

CUSTOMER ID	FIRST NAME	LAST NAME	FAVORITE COLOR
1	TAEKO	OHNUKI	BLUE
2	O.V.	WRIGHT	GREEN

HP2

CUSTOMER ID	FIRST NAME	LAST NAME	FAVORITE COLOR
3	SELD A	BAĞCAN	PURPLE
4	JIM	PEPPER	AUBERGINE

Sharding



Ventajas y desventajas de participar

VENTAJAS

Queries pueden ser más rápidas (búsquedas en un rango de menor tamaño).

Sistema más resiliente (si se cae una partición, el servicio continúa funcionando, aunque parcialmente).

Mayor escalabilidad, ya que permite escalamiento horizontal de servidores de BD.



DESVENTAJAS

Es muy fácil echar a andar una BD en un servidor, y escalarlo verticalmente.

Algunos motores de BD no soportan estrategias de particiones.

Dificultad de encontrar particiones correctas

Arquitecturas más complejas.

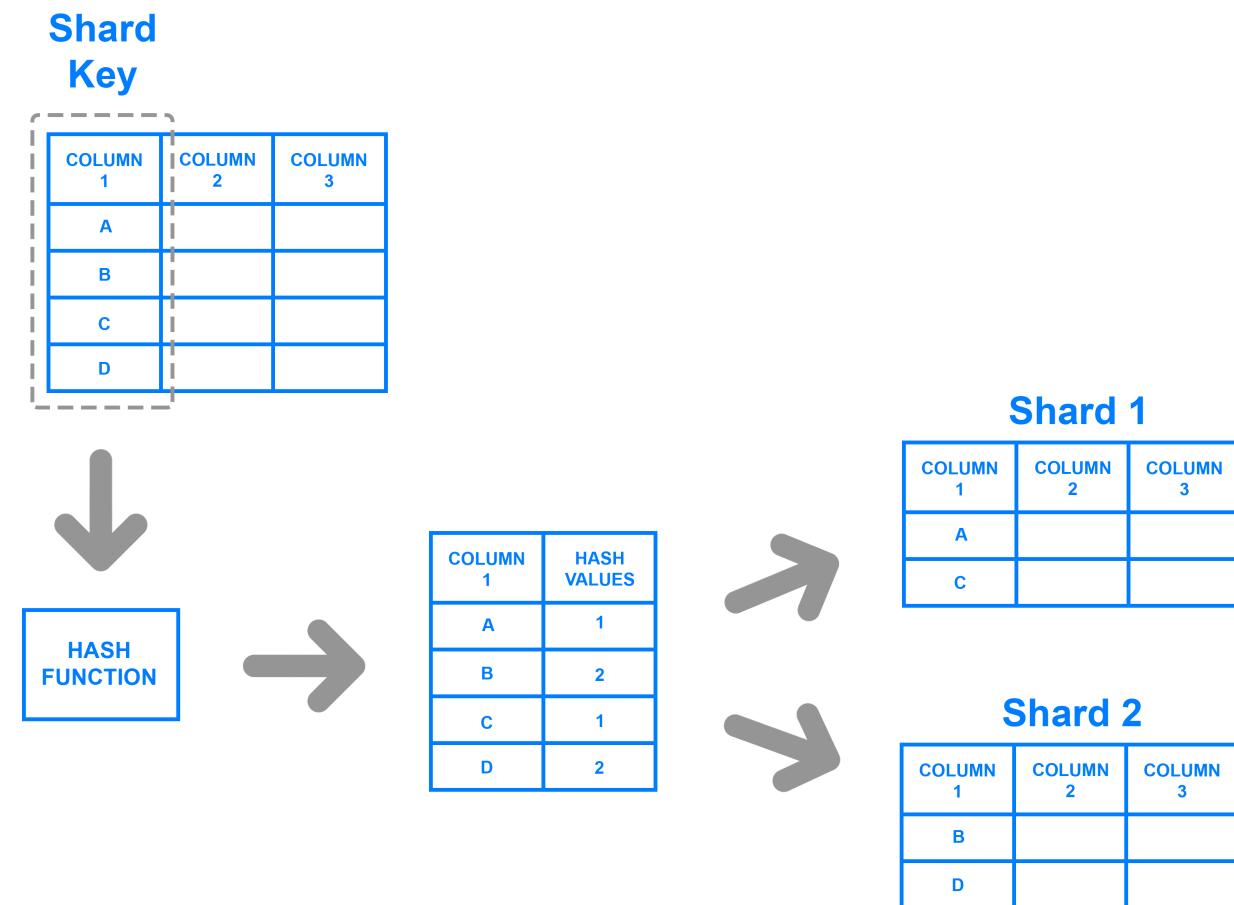
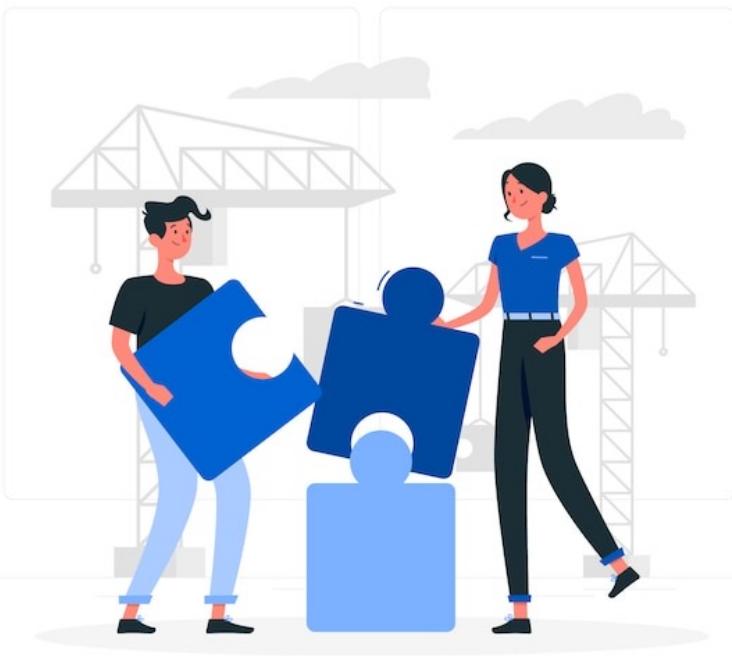
Particiones no balanceadas pueden llevar a problemas de performance.



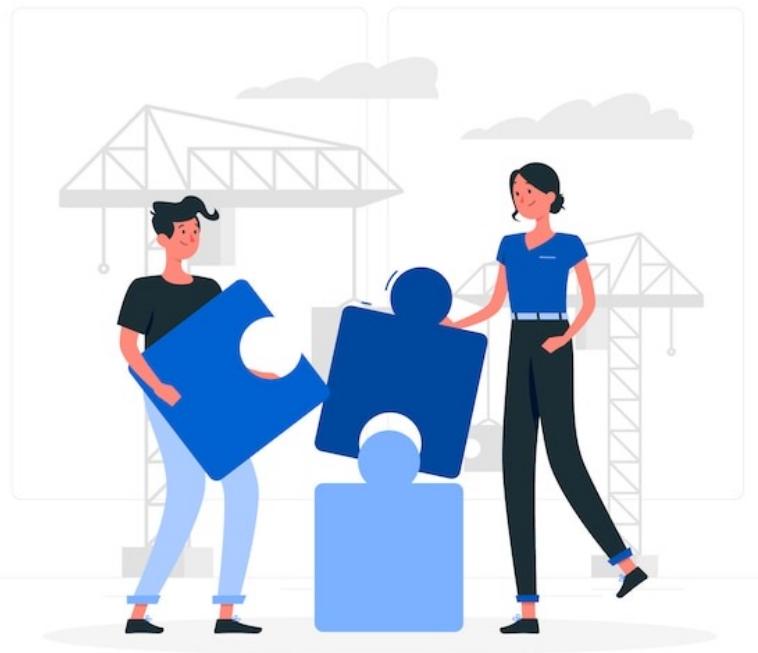
Estrategias de sharding

Haciendo particiones horizontales

SHARDING BASADO EN LLAVES



SHARDING BASADO EN RANGOS



PRODUCT	PRICE
WIDGET	\$118
GIZMO	\$88
TRINKET	\$37
THINGAMAJIG	\$18
DOODAD	\$60
TCHOTCHKE	\$999

(\$0-\$49.99)

PRODUCT	PRICE
TRINKET	\$37
THINGAMAJIG	\$18



(\$50-\$99.99)

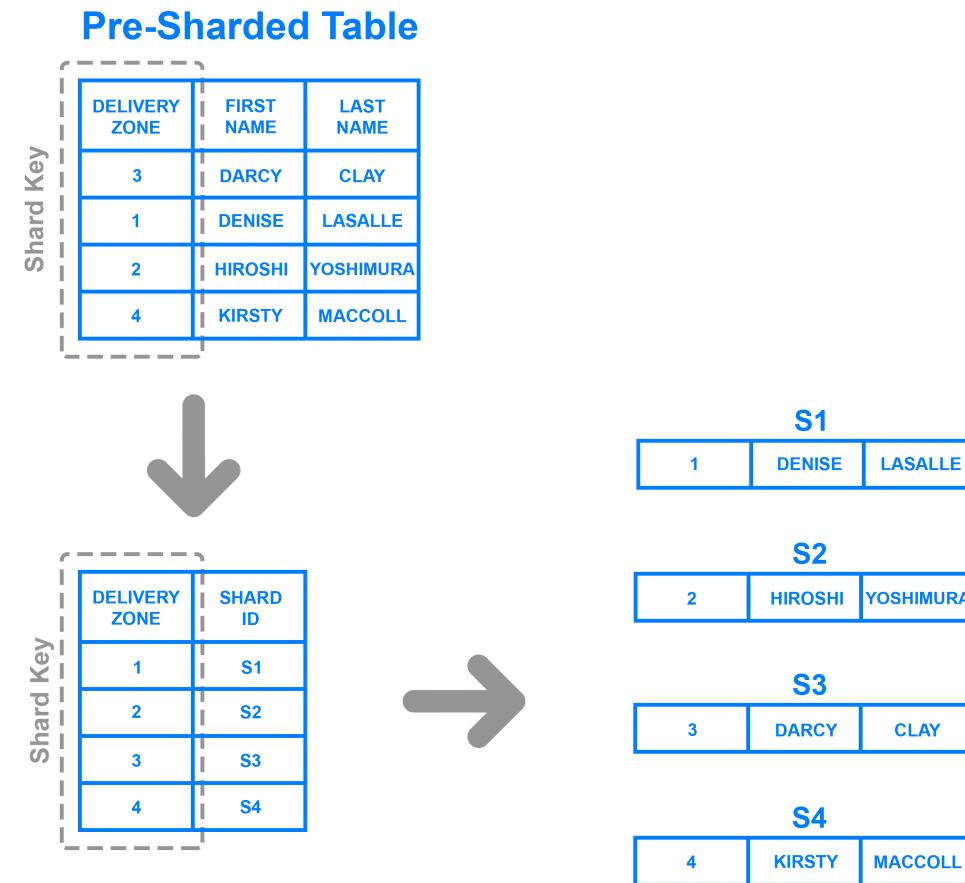
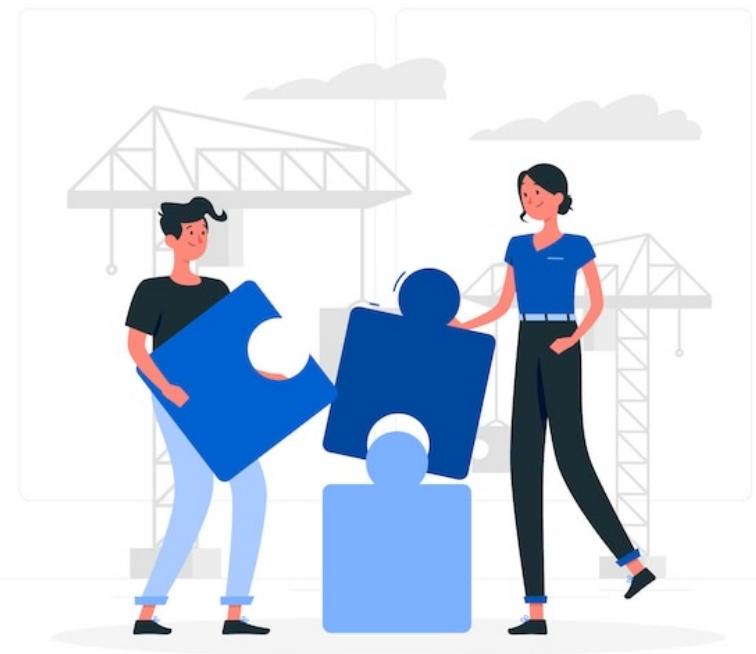
PRODUCT	PRICE
GIZMO	\$88
DOODAD	\$60



(\$100+)

PRODUCT	PRICE
WIDGET	\$118
TCHOTCHKE	\$999

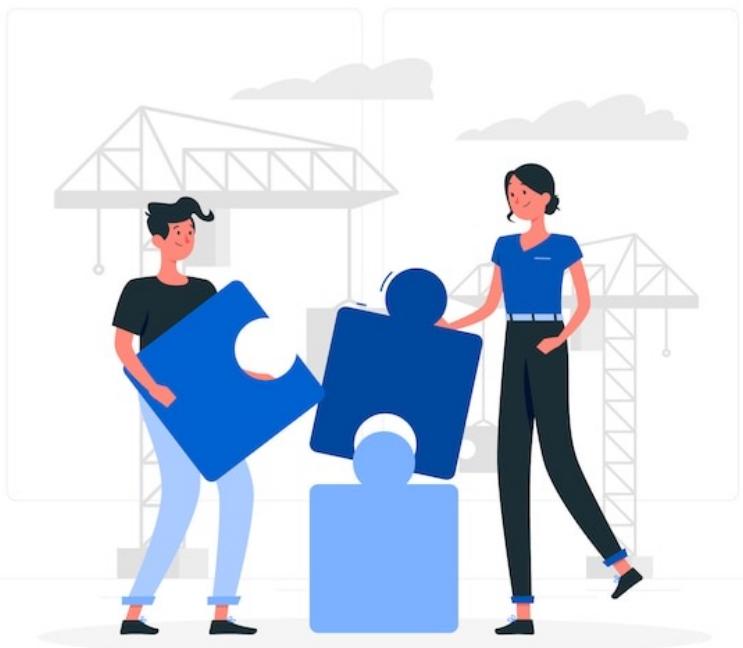
SHARDING BASADO EN LISTAS



SHARDING BASADO EN CICLO DE VIDA

Los datos residen en un shard según su antigüedad (o ciclo de vida)

Tiene la ventaja que la administración se puede realizar de formas diferentes según el dato (por ejemplo, método de almacenamiento)



Checklist antes de realizar particiones



Si se maneja todo en un servidor, mover BD a un servidor propio



Crecer verticalmente el servidor

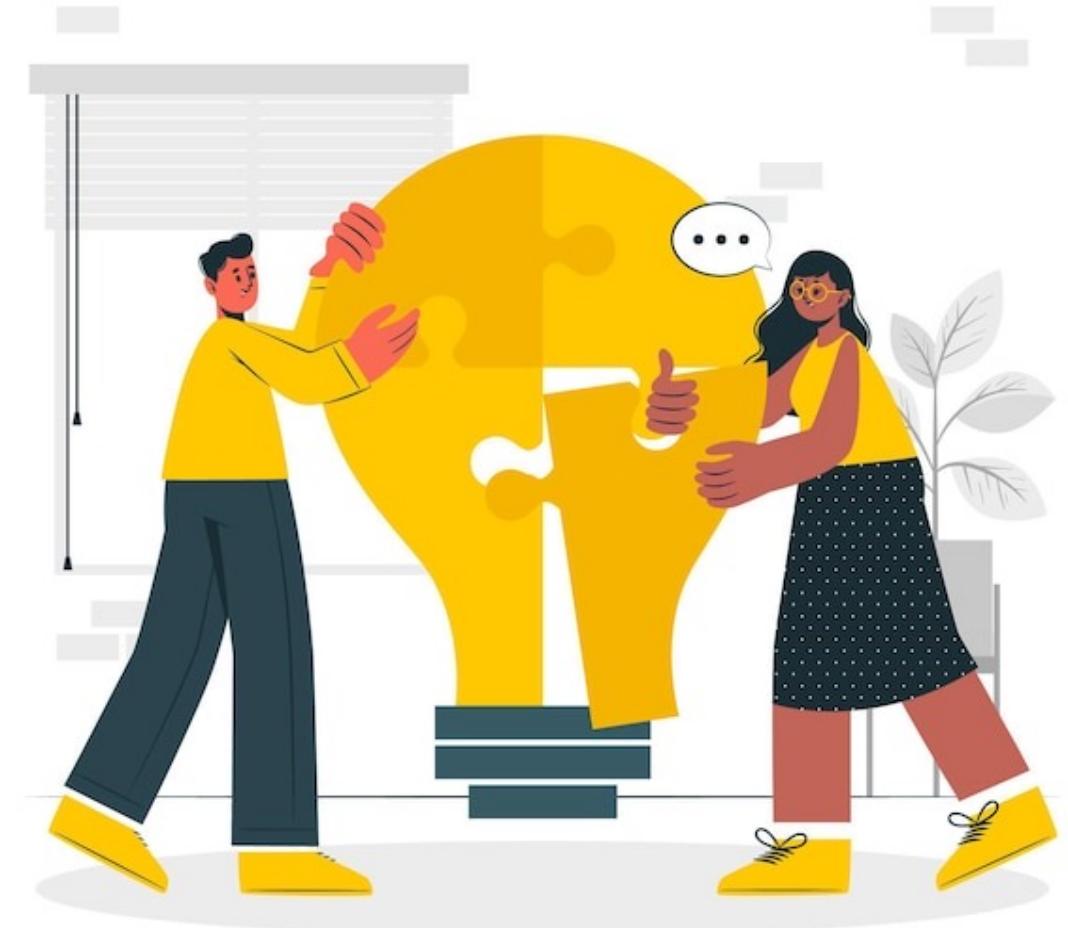


Implementar técnicas de caché



Crear réplicas de lectura y separar lógica de escritura y lectura (CQRS)

ESCALANDO LA BASE DE DATOS: PATRONES



COMMAND QUERY RESPONSABILITY SEGREGATION (CQRS)

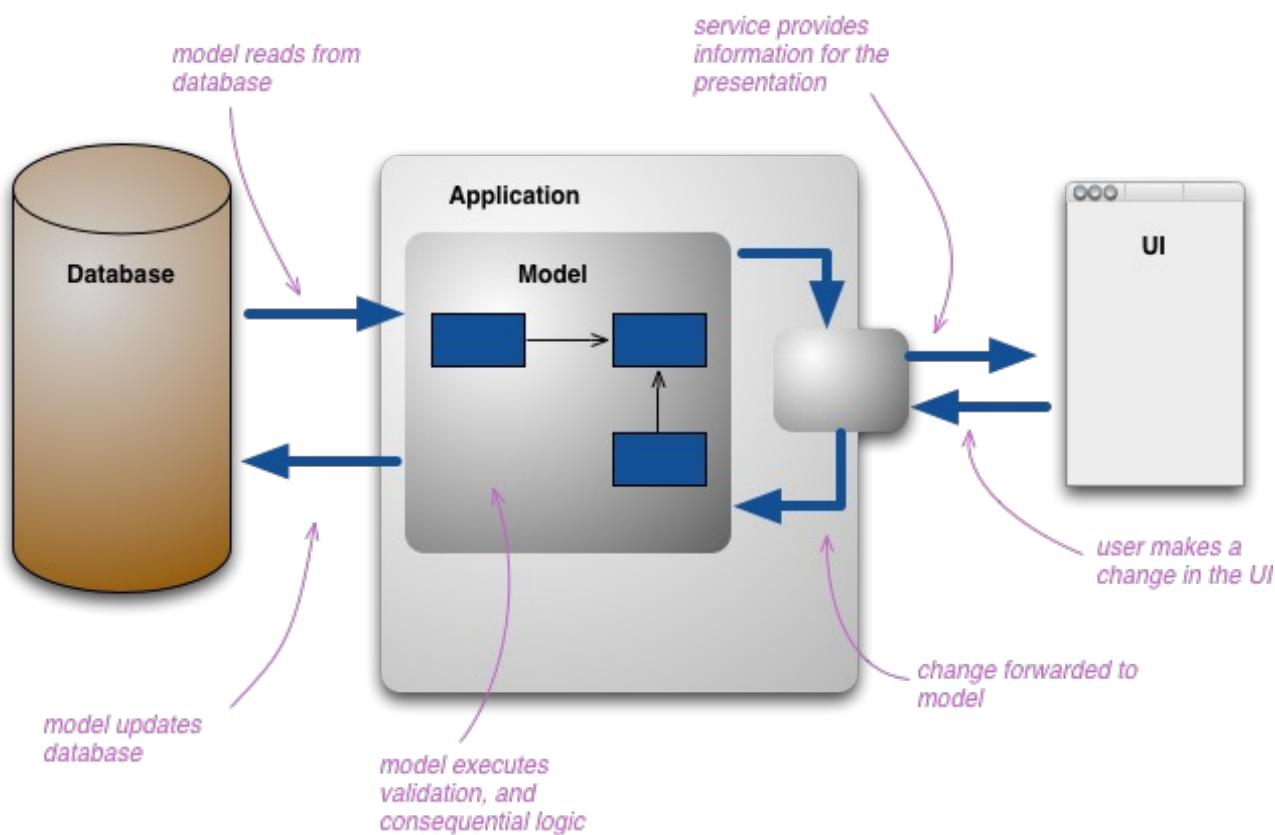
Arquitectura basada en la disociación de las bases de datos de escritura y lectura.

Significa que hay una BD que se preocupa de la escritura, y una segunda BD llamada vista o proyección, que maneja la lectura de datos.

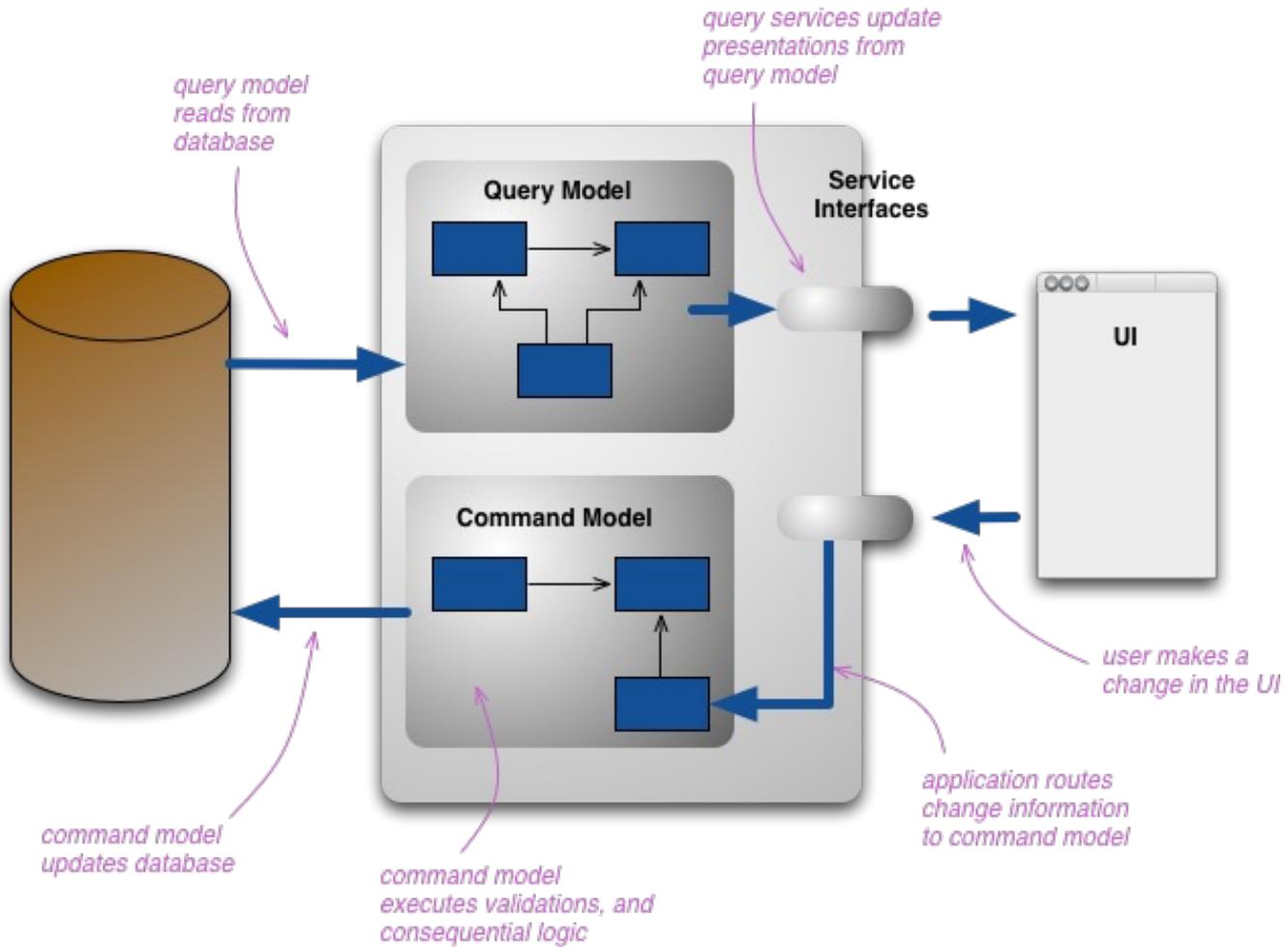
La mayoría de las veces, los datos de lectura se van calculando de forma asíncrona y en segundo plano, lo que implica que no siempre todos los datos son estrictamente consistentes (*eventual consistency*).



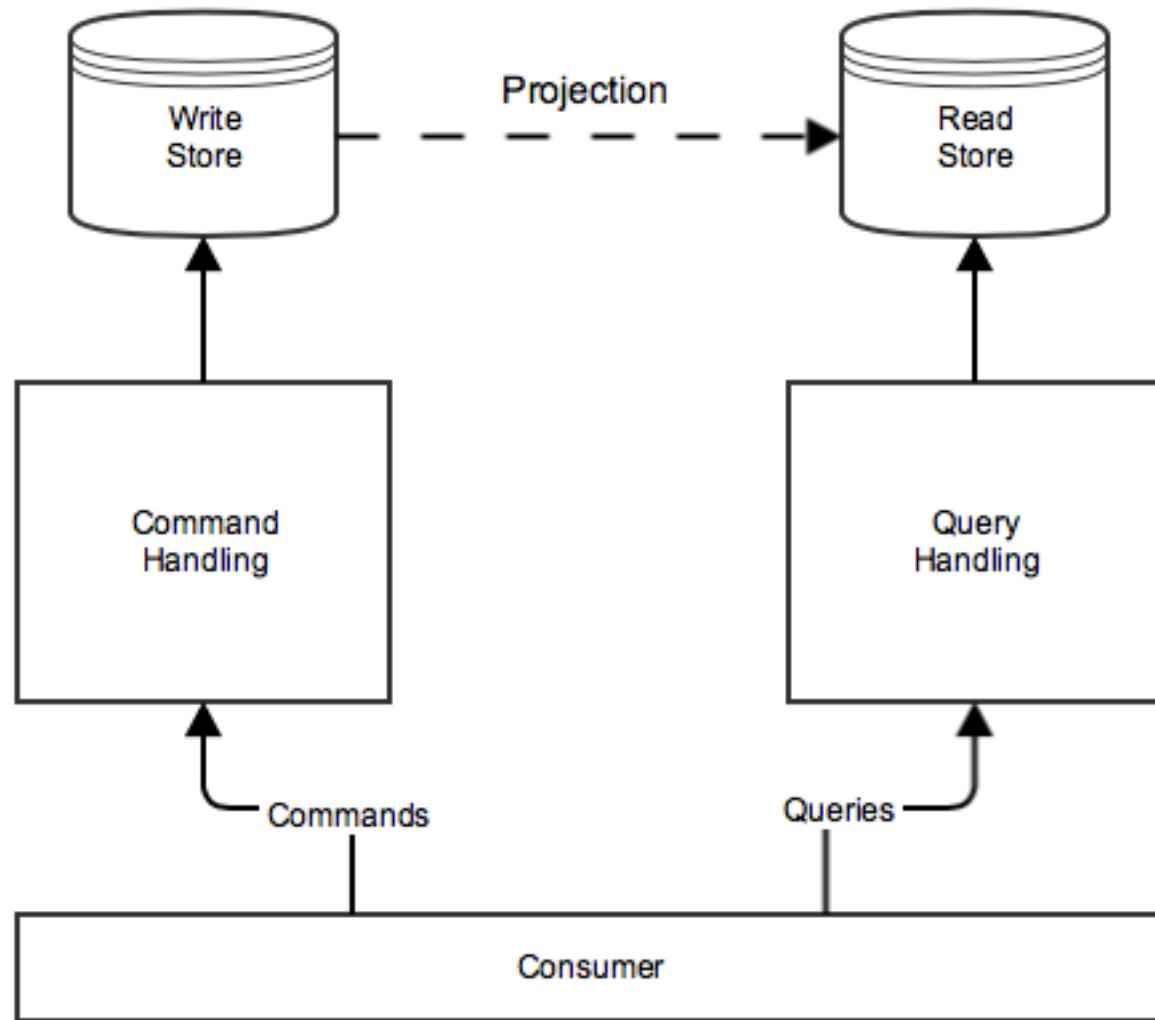
ARQUITECTURA MONOLÍTICA



ARQUITECTURA PRE-CQRS



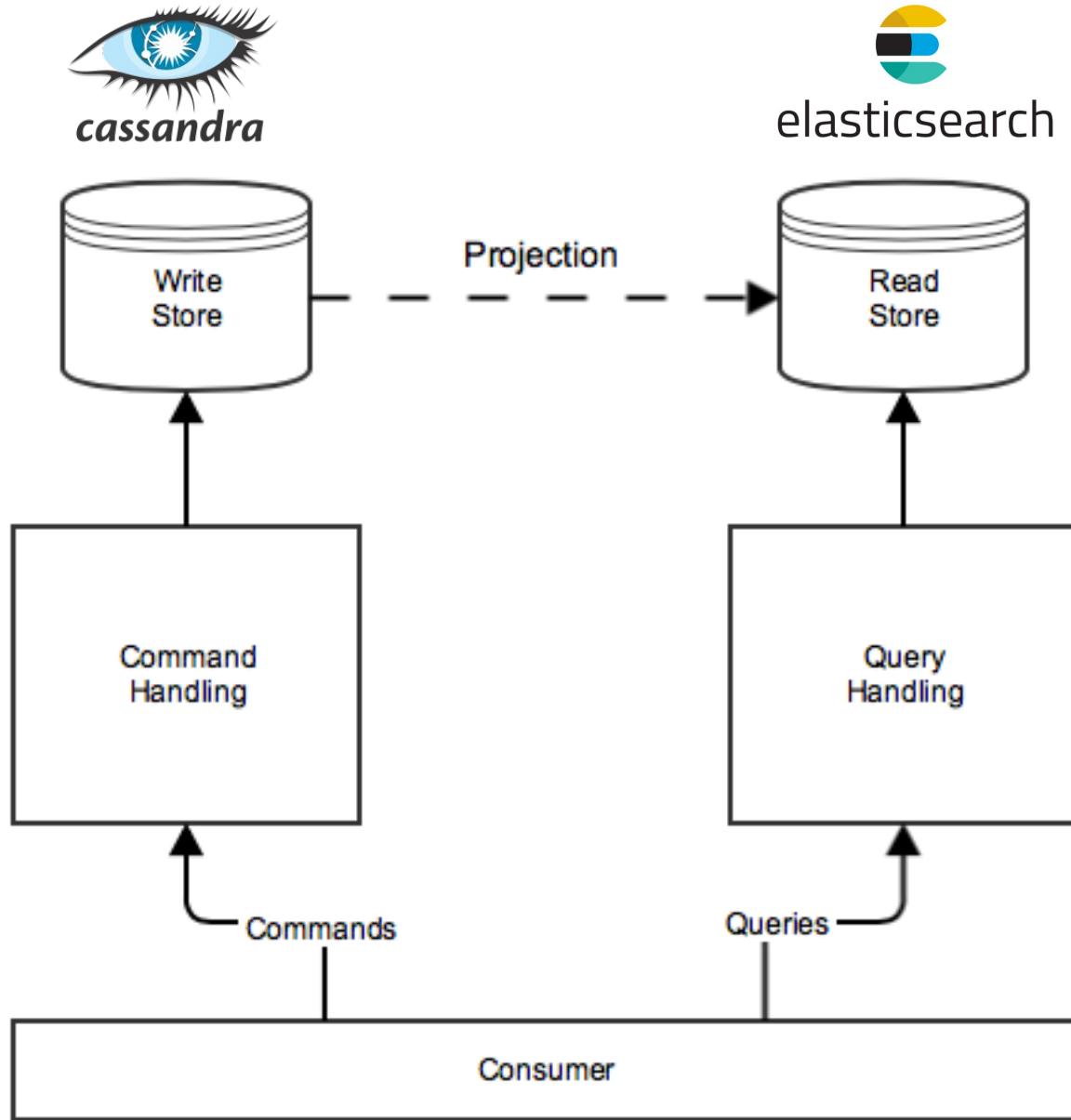
Arquitectura CQRS con 2 bases de datos



Arquitectura CQRS con 2 bases de datos

Cassandra como BD de escritura: muy rápida en escritura de datos.

ElasticSearch para lectura: Lenta en escritura, pero muy eficiente en la búsqueda de datos.



1.087
Invirtiendo

Tesla

US\$1.008,06

↗ 10,81% Variación 1D

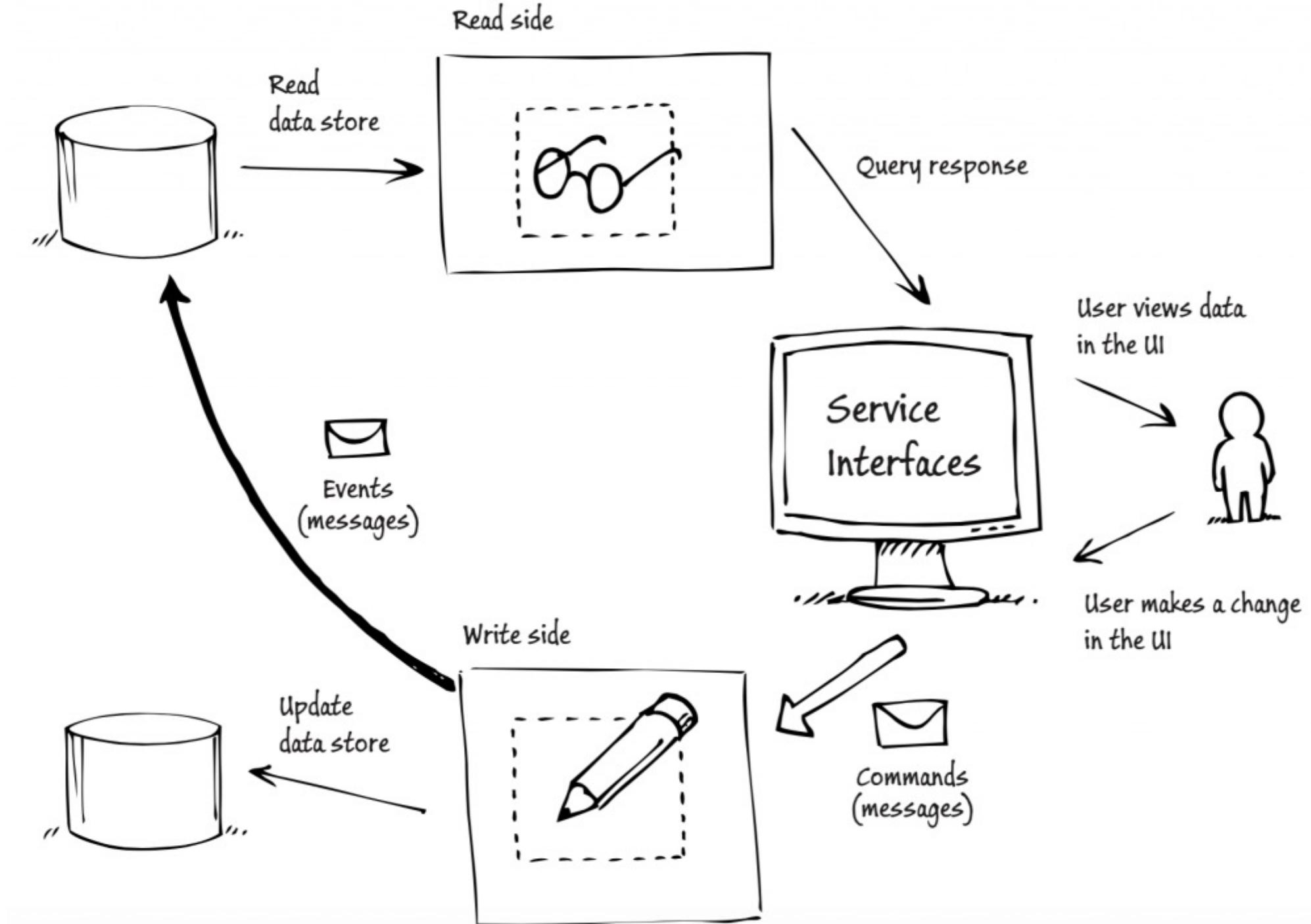


Tu inversión en TSLA

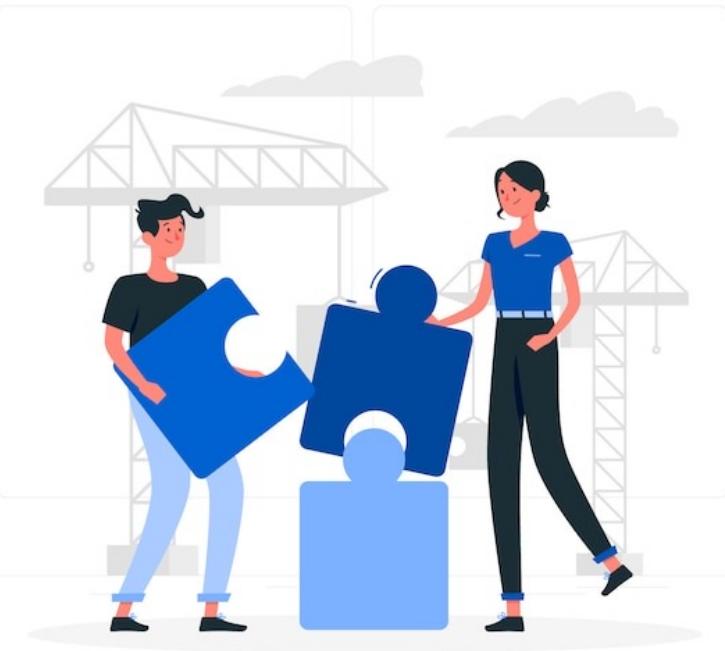
Acciones
1,12208689Valor inversión
US\$1.131,13Ganancias hoy
US\$110,23 (10,80%)Ganancias totales
US\$366,19 (47,87%)

Costo promedio

Comprar



CUANDO ES CONVENIENTE USAR CQRS



- Cuando nos movemos hacia fuera de una arquitectura basada en representación de datos CRUD y avanzamos hacia una arquitectura basada tareas.
- Con arquitecturas basadas en eventos.
- Cuando se necesitan diferentes modelos de escritura y de lectura para los mismos datos.
- Cuando se requieren guardar todas las actualizaciones del modelo, pero las visualizaciones solo requieren el estado final.
- Cuando nos encontramos con múltiples dominios y/o dominios complejos (*Domain-driven design*).

VENTAJAS Y DESVENTAJAS DE CQRS

Escalabilidad es mucho mayor ya que permite manejar de forma independiente las lecturas de las escrituras.

Por ejemplo, casos de alta frecuencia de lectura permite tener una base con mayor capacidad en ese lado.

Agrega flexibilidad en el uso de distintos modelos/motores de BD de acuerdo a los casos de uso.

Es más difícil tener consistencia perfecta de los datos.

Agrega complejidad a la construcción del sistema.



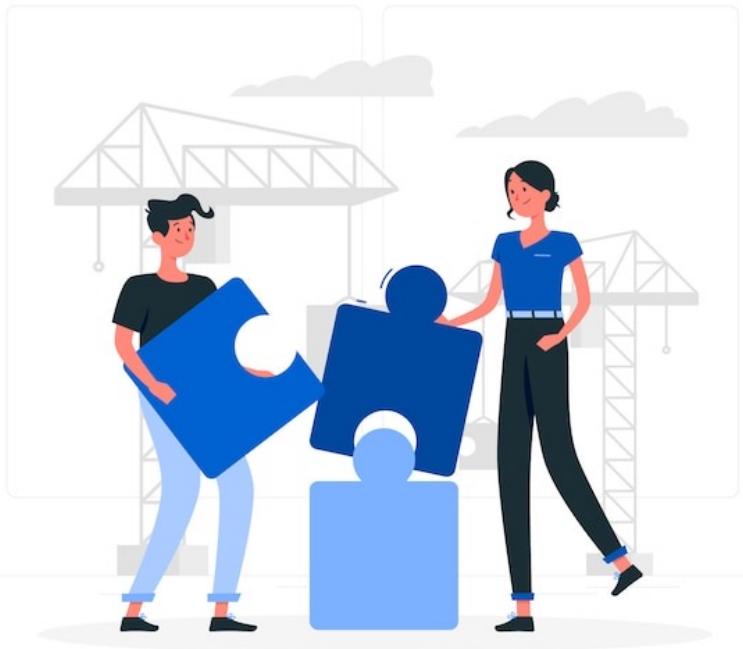
EVENT SOURCING

Se crea un mensaje para cada cambio de estado

Este se almacena para luego ser procesado

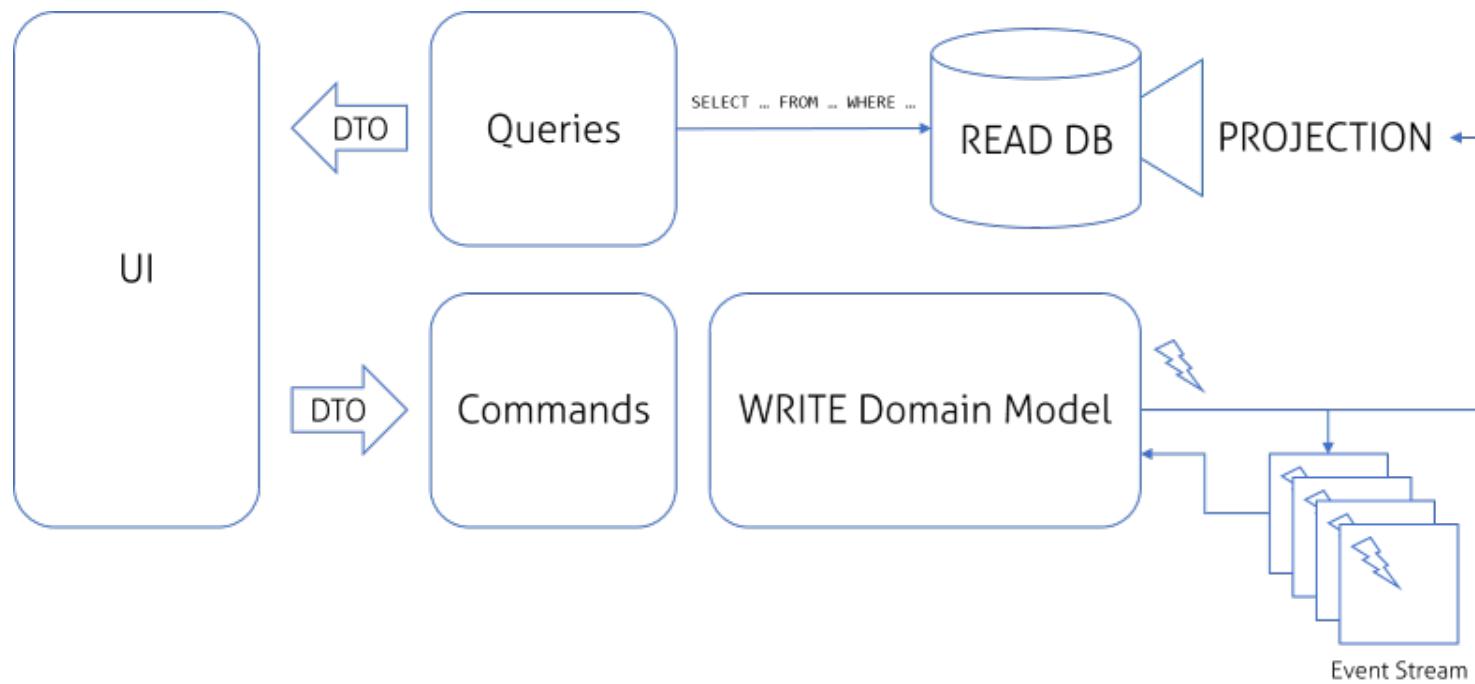


VENTAJAS Y DESVENTAJAS DE EVENT SOURCING



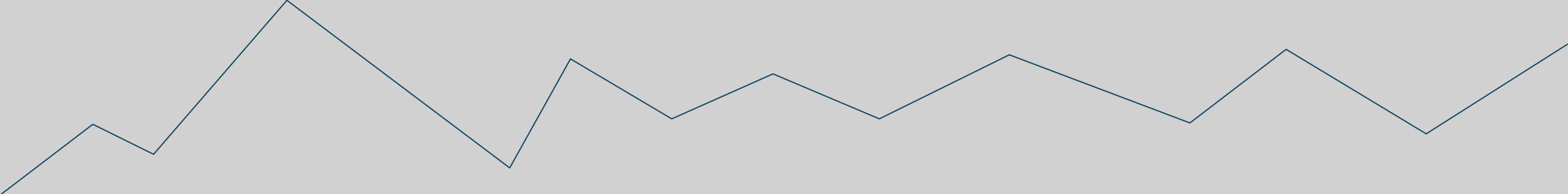
- Se puede reconstruir la historia por completo, ya que se tiene la historia de todos los mensajes
- Se puede tener el estado de una aplicación en cualquier momento del tiempo
- Se pueden repetir los eventos y recalcular el resultado final en el caso de errores en el procesamiento
- No se puede asegurar consistencia (*eventual consistency*)

CQRS + EVENT SOURCING

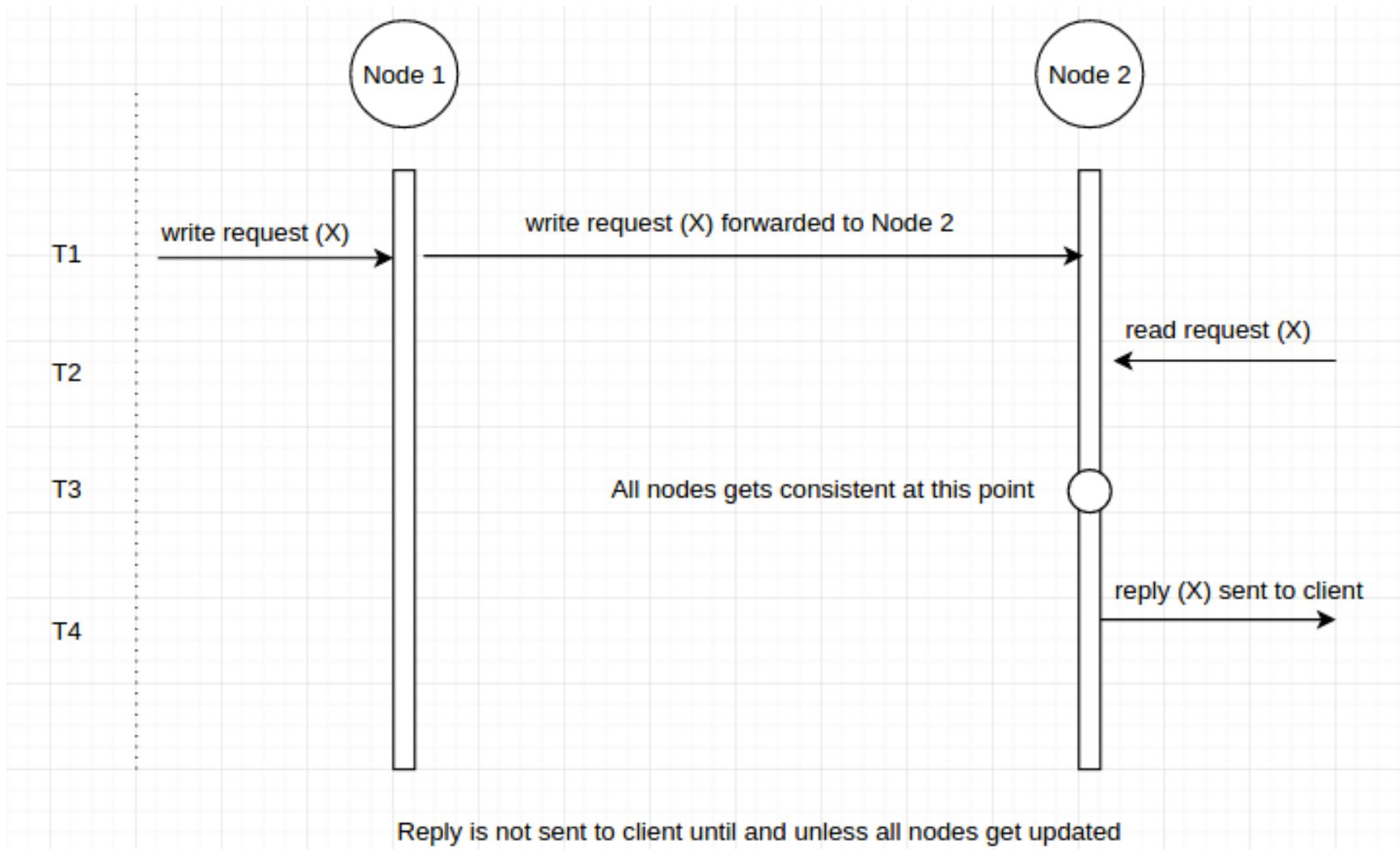


Consistencia Eventual

¿Están los datos sincronizados?



Consistencia dura o fuerte o estricta



No se puede responder a la solicitud hasta que todos los nodos se actualicen.

CONSISTENCIA DURA O FUERTE O ESTRICTA

Ofrece datos siempre actualizados, pero no es gratis:

- Mayor latencia

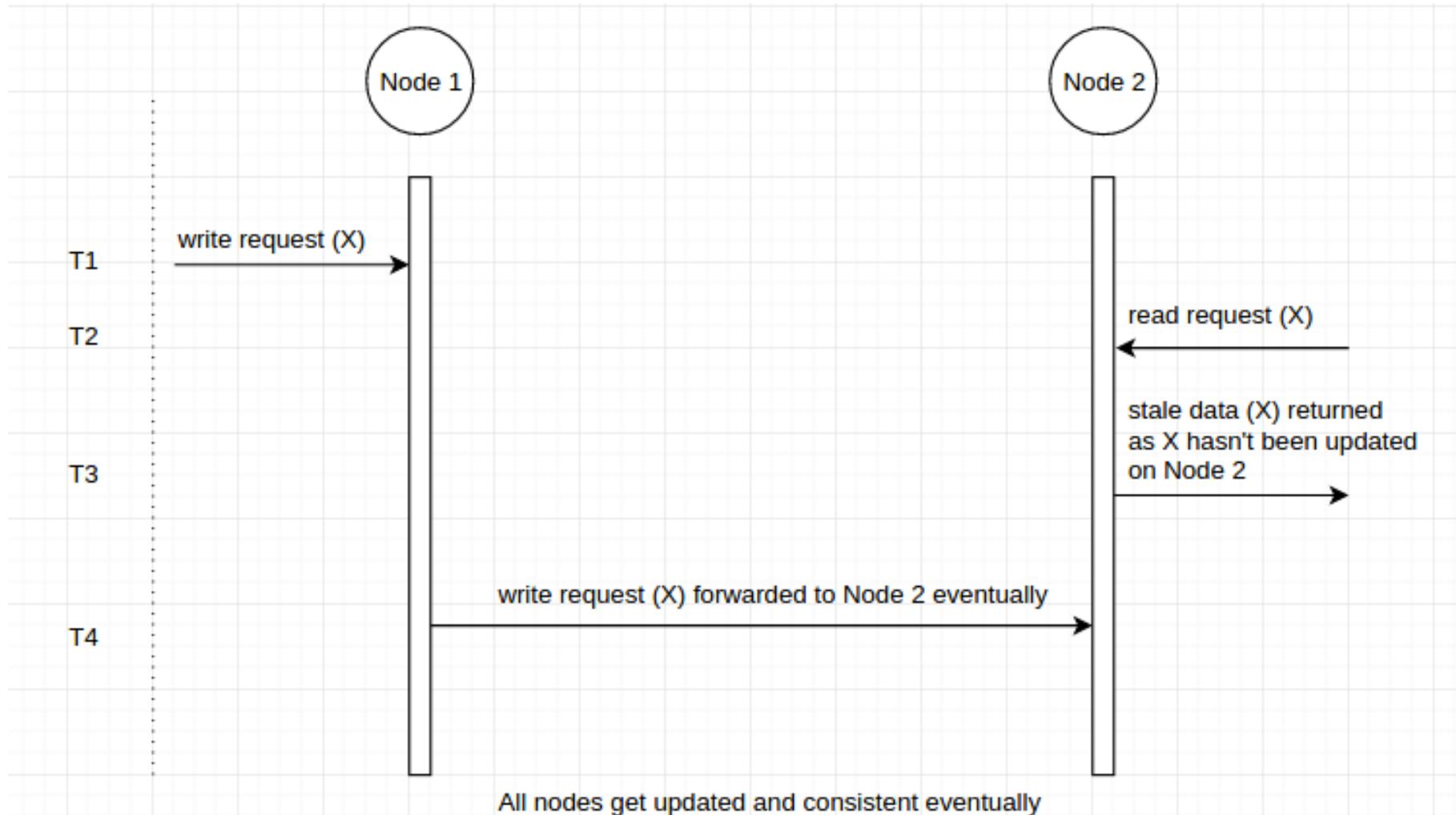
- Potenciales deadlocks



Consistencia fuerte



Consistencia eventual



Se responde a la solicitud, aunque información puede estar desactualizada

CONSISTENCIA EVENTUAL

Los datos pueden no estar siempre actualizados

Baja latencia

En algún punto del tiempo, todos los nodos de datos estarán actualizados

¿Es realmente importante que ese dato esté actualizado?



Consistencia eventual con secuencia



Consistencia eventual



CUANDO USAR CONSISTENCIA EVENTUAL

¿Es realmente importante que ese dato esté actualizado?

Ejemplos:

Un reporte histórico de uso de una app

Información de saldos en cuenta corriente

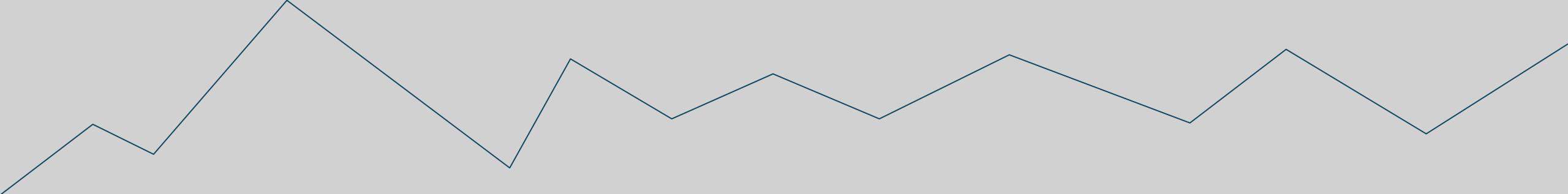
Respaldo de documentos (Google Drive, Dropbox)

Datos de sensores de navegación de un auto que se maneja solo



Teorema CAP

Limitaciones de la escalabilidad



TEOREMA CAP

Un sistema distribuido sólo puede asegurar dos de tres propiedades:

- Consistencia (C)
- Disponibilidad (A)
- Tolerancia a la partición (P)



CONSISTENCIA

Todos los nodos del sistema poseen la misma información al mismo tiempo.

Cuando un cliente lee información, recibe la misma información de cada nodo. Si los nodos no están sincronizados, se genera un error.



DISPONIBILIDAD

Se refiere a la capacidad de un sistema que todos los nodos puedan responder en todo momento, a pesar de fallas o particiones de red.

La disponibilidad no garantiza la entrega de información consistente.



TOLERANCIA A LA PARTICIÓN

Es la habilidad del sistema de continuar funcionando aun cuando existan errores o particiones de red.

Las particiones de red pueden generar que algunos nodos no se puedan comunicar con otros, sin poder haber sincronización en el sistema.

La partición no necesariamente es la pérdida de comunicación. También se puede referir a redes o procesos de sincronización lento por carga u otros motivos.



PRIORIZANDO DOS DE TRES:



C + A

No tolera fallas a las particiones. ¿Es realmente un sistema distribuido?



C + P

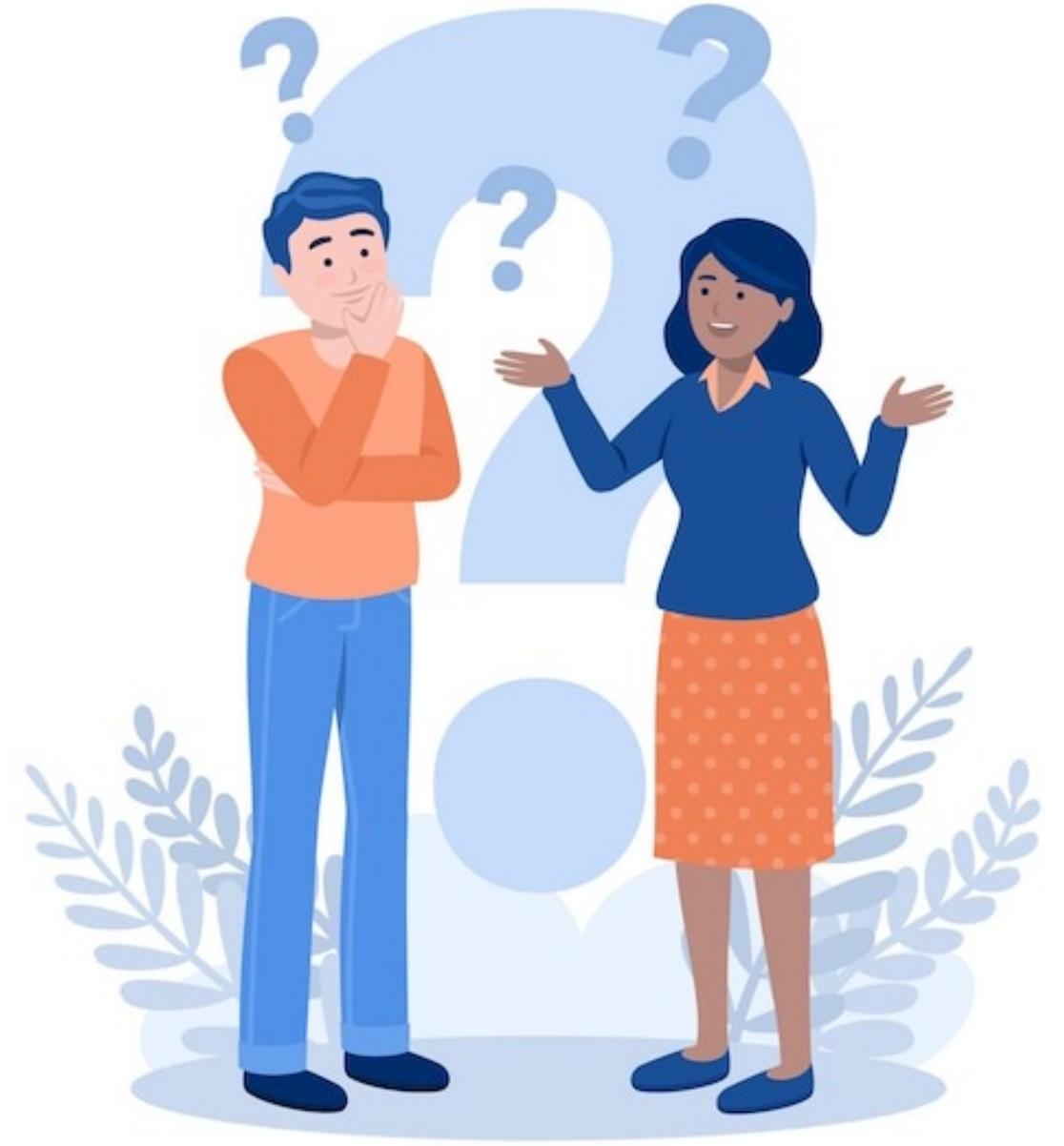
Prioriza consistencia, por lo que no permite tener datos desactualizados entre nodos



A + P

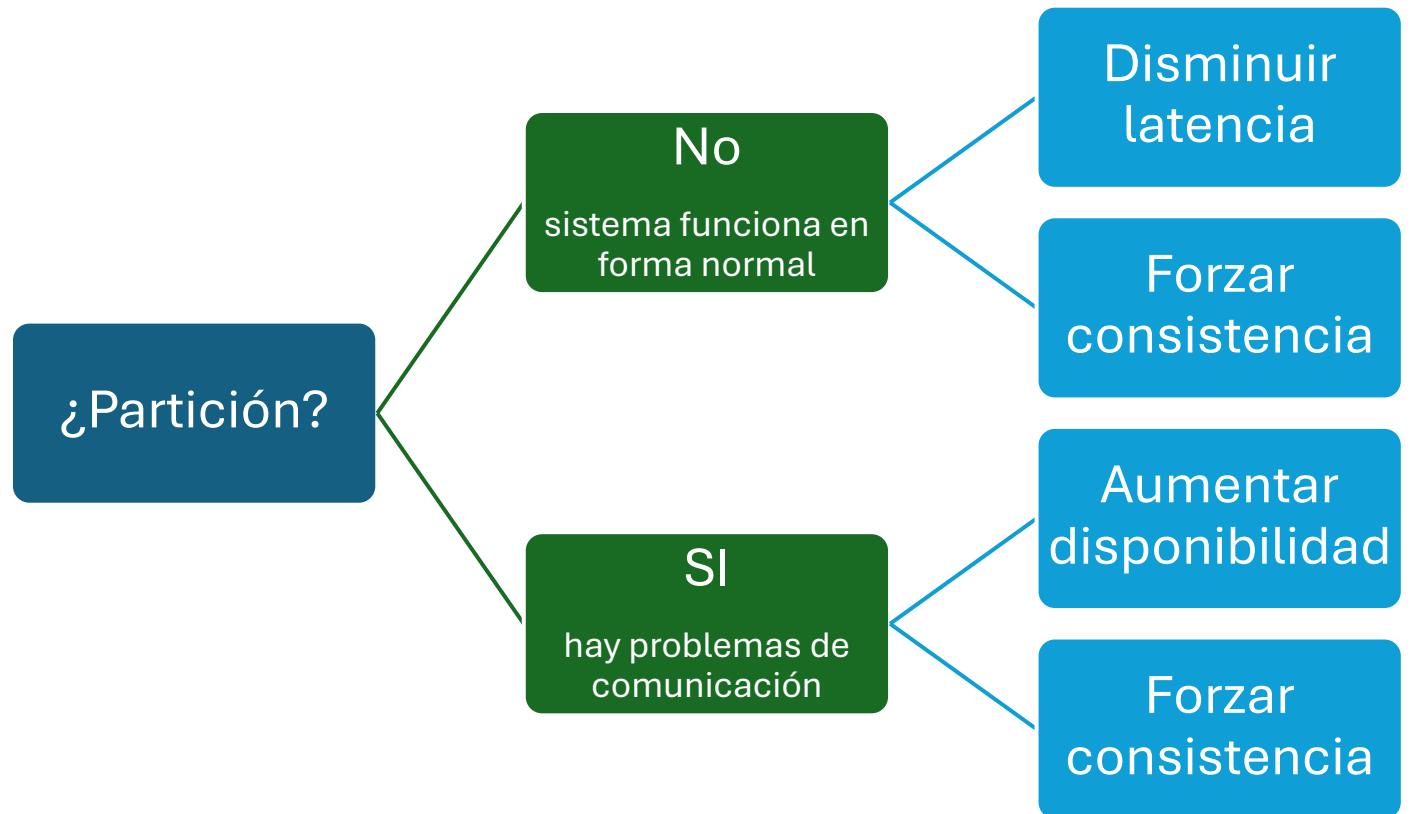
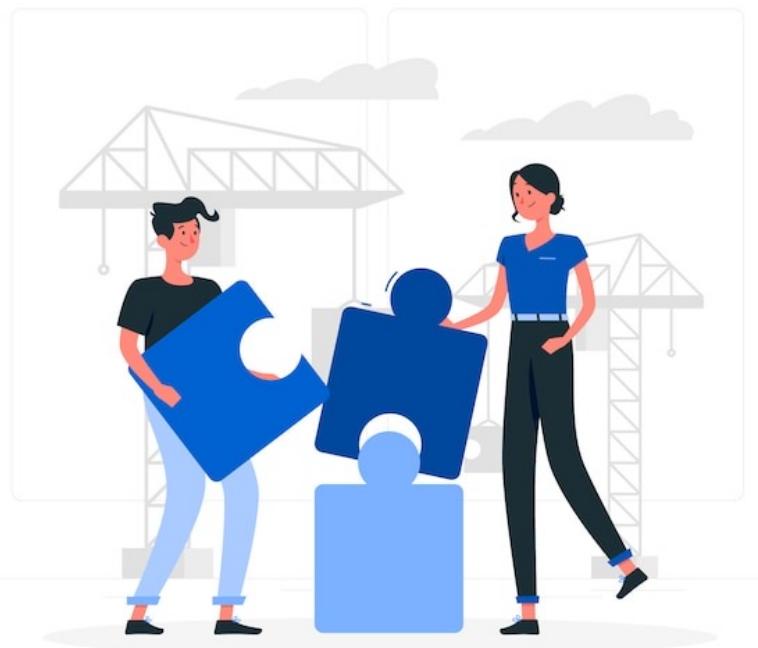
Prioriza disponibilidad, por lo que se permite tener datos desactualizados o no sincronizados entre nodos.





Se ve un poco teórico.
¿Cómo lo aplico?

PACELC



**“If there is a Partition how does the system tradeoff between Availability and Consistency;
Else when the system is running as normal in the absence of partitions, how does the system tradeoff between Latency and Consistency?”**





PONTIFICIA
UNIVERSIDAD
CATÓLICA
DE CHILE

IIC3103

Taller de Integración

Profesores

Arturo Tagle / Daniel Darritchon

BIBLIOGRAFÍA

Teorema CAP

Wikipedia - CAP theorem

https://en.wikipedia.org/wiki/CAP_theorem

CAP Theorem

<http://www.cedanet.com.au/ceda/fallacies/cap-theorem.php>

Understanding of CAP Theorem

<https://www.mysoftkey.com/architecture/understanding-of-cap-theorem/>

Problems with CAP, and Yahoo's little known NoSQL system

<http://dbmsmusings.blogspot.com/2010/04/problems-with-cap-and-yahoos-little.html>

How we used CAP theorem in our system design?

<https://medium.com/@noobj/how-we-used-cap-theorem-in-our-system-design-6f2ea82bd229>



BIBLIOGRAFÍA

Teorema CAP

CAP Theorem Explained: Consistency, Availability, Partition Tolerance

<https://daily.dev/blog/cap-theorem-explained-consistency-availability-partition-tolerance>

System Design Interview Basics: CAP vs. PACELC

<https://www.desingurus.io/blog/system-design-interview-basics-cap-vs-pacelc>

An Illustrated Proof of the CAP Theorem

https://mwhittaker.github.io/blog/an_illustrated_proof_of_the_cap_theorem/



BIBLIOGRAFÍA

Escalabilidad de servidores

System Design Basics: Horizontal vs. Vertical Scaling

<https://www.youtube.com/watch?v=xpDnVSmNFX0>

Basics Of Scaling: Load Balancers

<https://brandonwamboldt.ca/basics-of-scaling-load-balancers-1432/>

DNS Load Balancing and Using Multiple Load Balancers in the Cloud

<https://blogs.flexera.com/cloud/enterprise-cloud-strategies/dns-load-balancing-and-using-multiple-load-balancers-in-the-cloud/>



BIBLIOGRAFÍA

Escalabilidad de datos

Particiones

Partitioning Concepts

https://docs.oracle.com/cd/B28359_01/server.111/b32024/partition.htm

Understanding Database Sharding

<https://www.digitalocean.com/community/tutorials/understanding-database-sharding>

CQRS

Some thoughts on using CQRS without Event Sourcing

<https://medium.com/@mbue/some-thoughts-on-using-cqrs-without-event-sourcing-938b878166a2>

CQRS (1): ¿Qué es?

<https://eamodeorubio.wordpress.com/2012/09/03/cqrs-1-que-es/>

1 Year of Event Sourcing and CQRS

<https://hackernoon.com/1-year-of-event-sourcing-and-cqrs-fb9033ccd1c6>

CQRS

<https://martinfowler.com/bliki/CQRS.html>

CQRS Pattern

<https://medium.com/eleven-labs/cqrs-pattern-c1d6f8517314>



BIBLIOGRAFÍA

Escalabilidad de mensajes

What Starbucks can teach us about software scalability

<https://particular.net/blog/what-starbucks-can-teach-us-about-software-scalability>

Event Sourcing

<https://martinfowler.com/eaaDev/EventSourcing.html>

Event sourcing, CQRS, stream processing and Apache Kafka:
What's the connection?

<https://www.confluent.io/blog/event-sourcing-cqrs-stream-processing-apache-kafka-whats-connection/>



BIBLIOGRAFÍA

Consistencia eventual

<https://twitter.com/gregyoung/status/1101642600342265857>

Eventual vs Strong Consistency in Distributed Databases

<https://hackernoon.com/eventual-vs-strong-consistency-in-distributed-databases-282fdad37cf7>

What is Strict Consistency vs Eventual Consistency?

<https://www.cohesity.com/blog/strict-vs-eventual-consistency/>

Consistency levels in Azure Cosmos DB

<https://docs.microsoft.com/en-us/azure/cosmos-db/consistency-levels>

Eventual Consistency: What, How, and Why

<https://levelup.gitconnected.com/eventual-consistency-what-how-and-why-50c942472a0c>



MATERIAL ADICIONAL

Videos sobre escalabilidad

Scalability Simply Explained in 10 Minutes

https://www.youtube.com/watch?v=EWS_ClxttVw

CAP Theorem Simplified

<https://www.youtube.com/watch?v=BHqjEjzAicA>

Vertical Vs Horizontal Scaling: Key Differences You Should Know

<https://www.youtube.com/watch?v=dvRFHG2-uYs>

