IIC2523
Sistemas Distribuidos

Hernán F. Valdivieso López
(2025 - 2 / Clase 20)

Fiabilidad en Sistemas Distribuidos + Seguridad
Resumen *Express*

Temas de la clase

1. Tolerancia a Fallas
2. Consistencia Centrada en Datos y Cliente
3. Replicación de Datos (protocolos)
4. Transacciones Distribuidas
5. Control de Concurrencia
6. Teorema PAC y PACELC
7. Introducción a seguridad
8. Cifrado de mensaje

Tolerancia a fallas

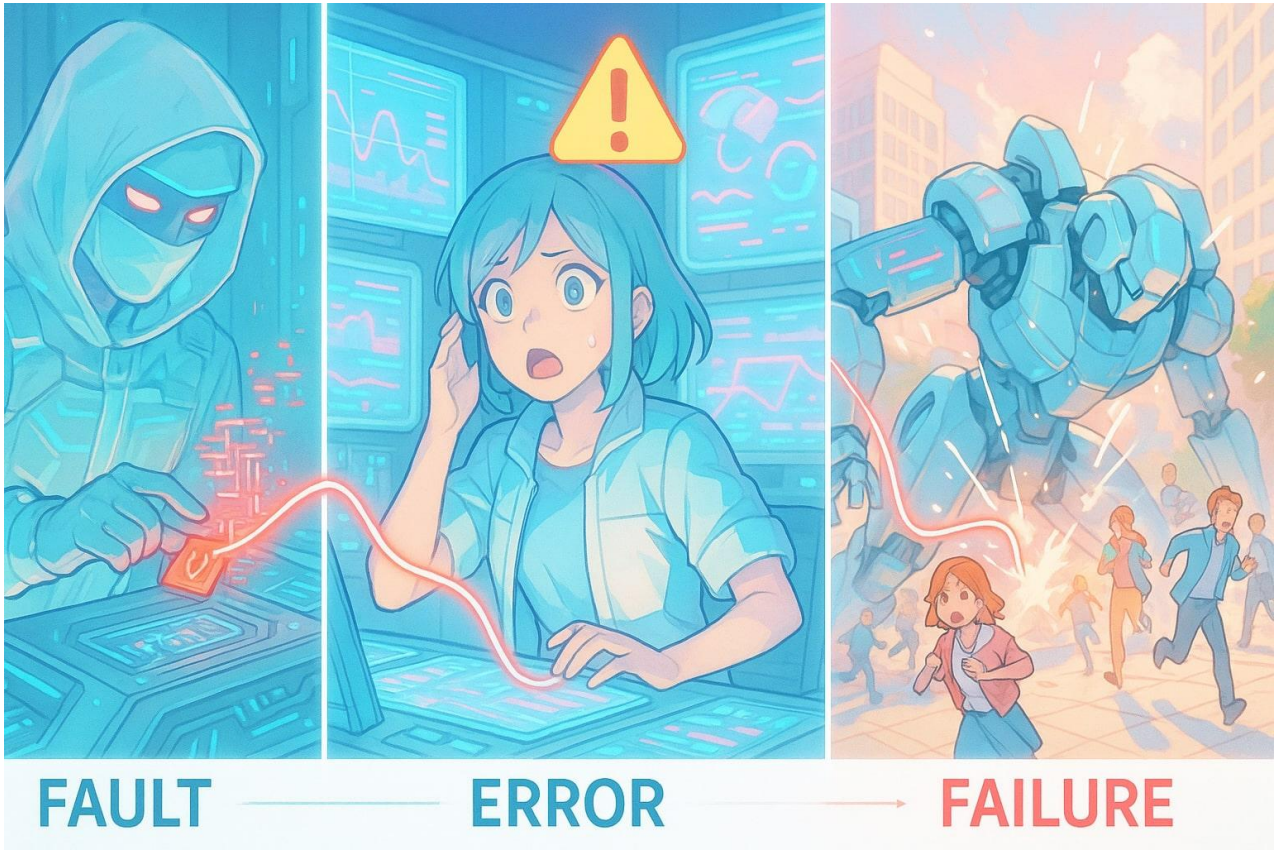
Conceptos fundamentales

- Un sistema distribuido tolerante a fallos es un sistema con una alta **confiabilidad**.

  El grado en que un sistema informático puede ser utilizado de forma esperada.

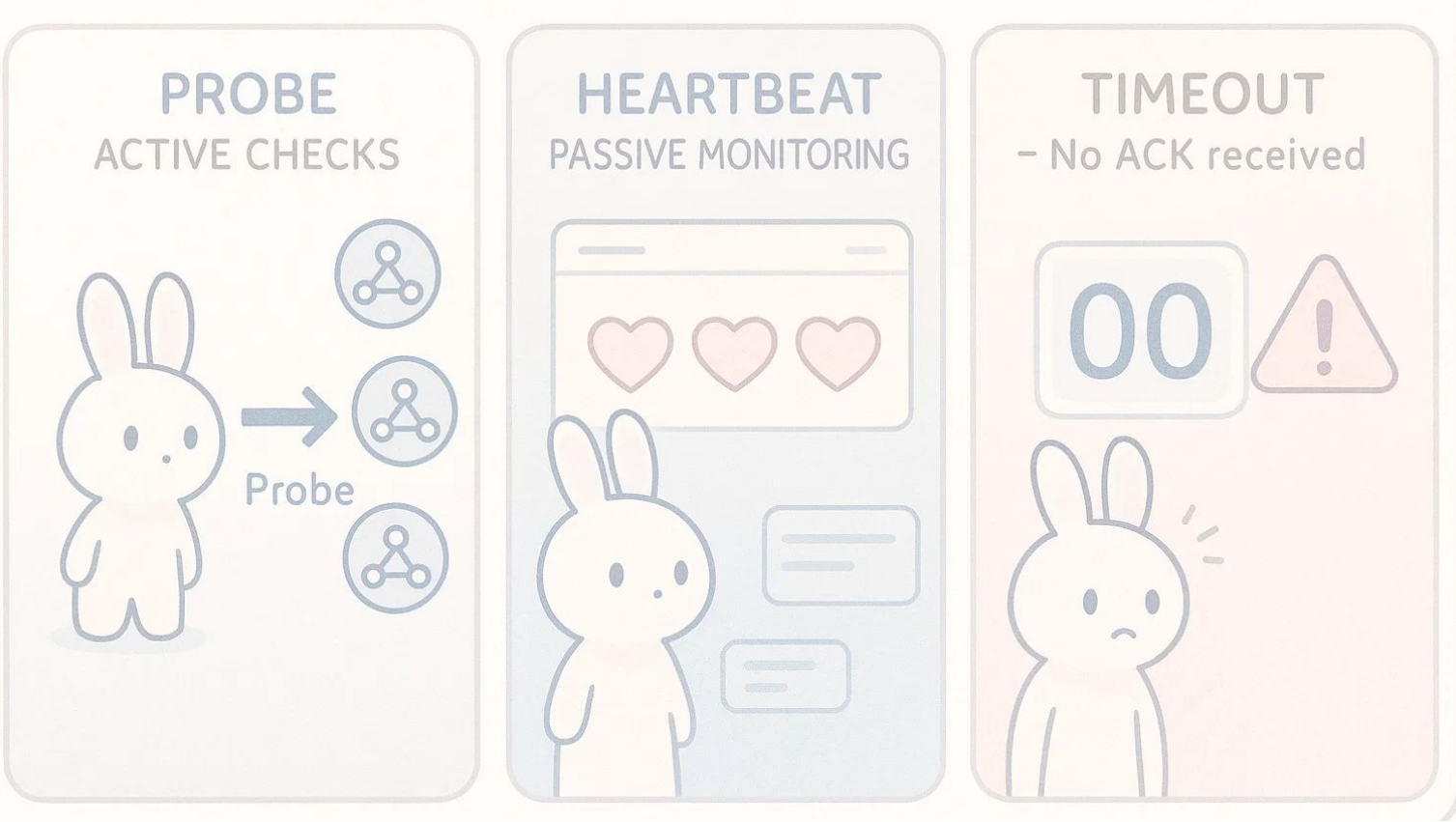
- Para lograr una alta confiabilidad. Hay varias propiedades que se desean cumplir:
  - ✓  Disponibilidad
  - ✓  Fiabilidad
  - ✓  Seguridad
  - ✓  Mantenibilidad

Conceptos fundamentales - Falla, Error y Fracaso



Fuente: OpenAI. (2025). [Diferencia Fault, Error y Failure] [Imagen generada por inteligencia artificial con Sora]. https://openai.com/sora

Failure Detection



Fuente: OpenAI. (2025). [Mecanismos de *Failure Detection*] [Imagen generada por inteligencia artificial con Sora]. https://openai.com/sora

Failure Masking

- Un sistema tolerante a fallos es aquel que puede continuar el rendimiento correcto de sus tareas especificadas en presencia de fallos de hardware y/o software.
- Su objetivo principal es evitar que los fallos generen un fracaso del sistema.
- La **redundancia** es un concepto clave para lograr este objetivo.
  - Implica la **adición** de información, recursos o tiempo **más allá de lo necesario** para el funcionamiento normal del sistema.
- Se distinguen 4 tipos de redundancia: *Hardware*, Información, Tiempo, *Software*.

Fuente: An Introduction to the Design and Analysis of Fault-Tolerant Systems (Johnson B), 1995.

Failure Masking

- *Hardware*: más elementos físicos.
- Información: agregar más datos al mensaje.
- Tiempo: repetir operaciones.
- *Software*: incluir otros programas que monitoreen o hagan de soporte.

Fuente: An Introduction to the Design and Analysis of Fault-Tolerant Systems (Johnson B), 1995.

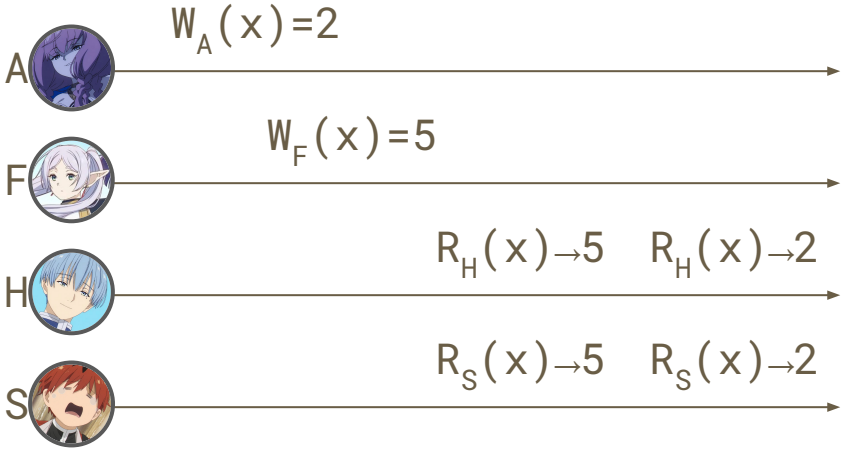
Consistencia de Datos

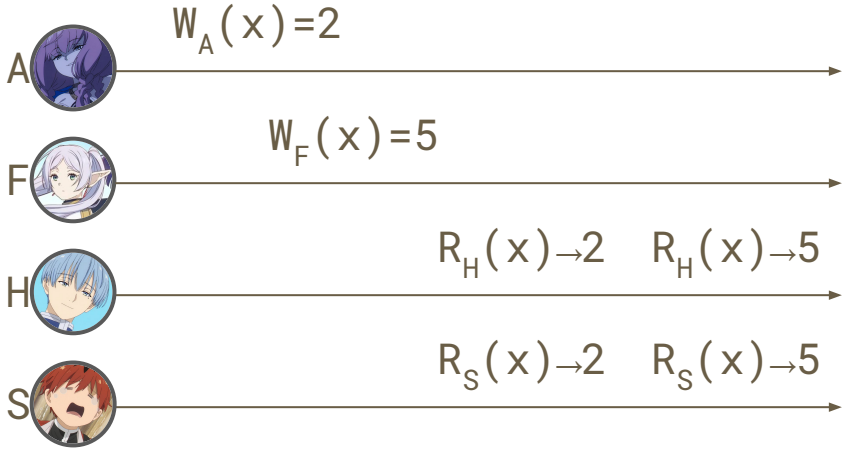
Centradas en los datos

Centradas en los clientes

Consistencia Secuencial [Lamport, 1979]

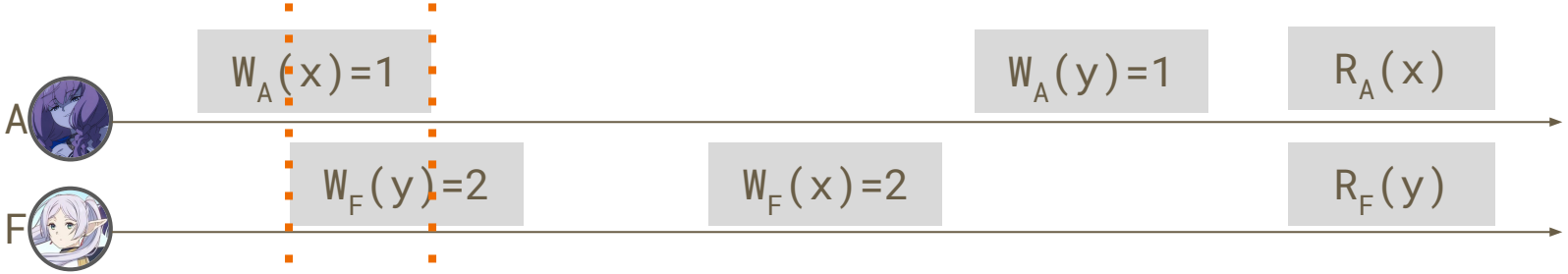
- El resultado de cualquier ejecución es el mismo que si las operaciones de todos los procesos se ejecutarán **en algún orden secuencial** ... y cada proceso individual respeta dicho orden.
  - Cualquier intercalación válida de las operaciones entre nodos es aceptable... pero todos los nodos ven la misma intercalación.





Consistencia Fuerte [Herlihy & Wing, 1986]

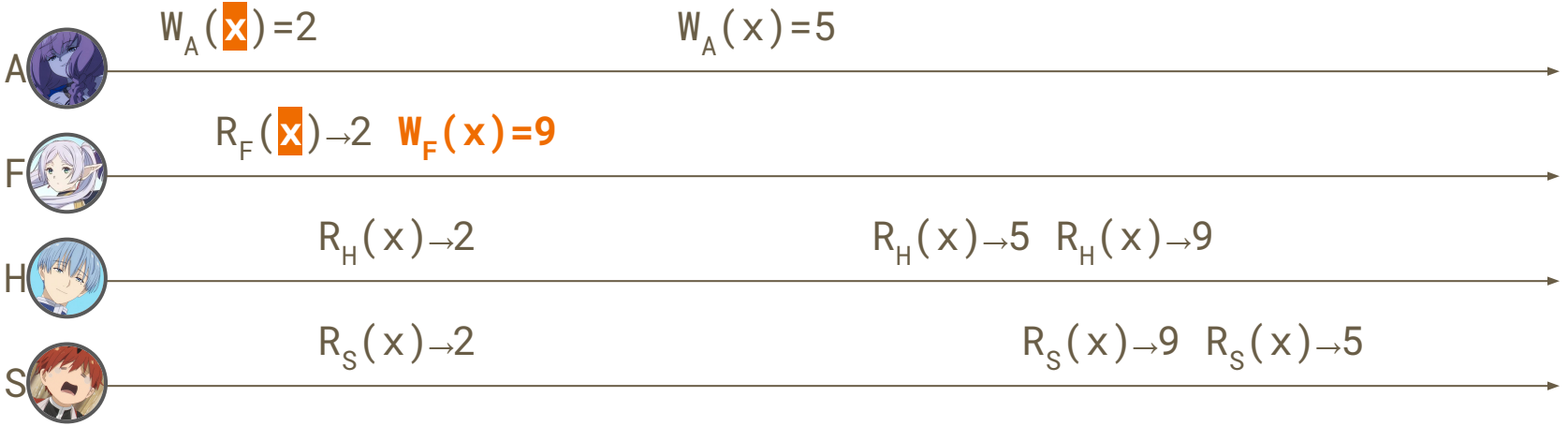
- Consistencia más estricta que la secuencia. Llamada también linealizable.
- Cada operación debe aparecer como que tiene efecto instantáneo en **algún momento entre su inicio y su finalización**.
  - La consistencia fuerte está **totalmente ligado al tiempo real de las operaciones**.
- Al momento que se completa un *write*, esta operación se propaga inmediato a las demás réplicas.



Solo 2 secuencia válida de *writes*

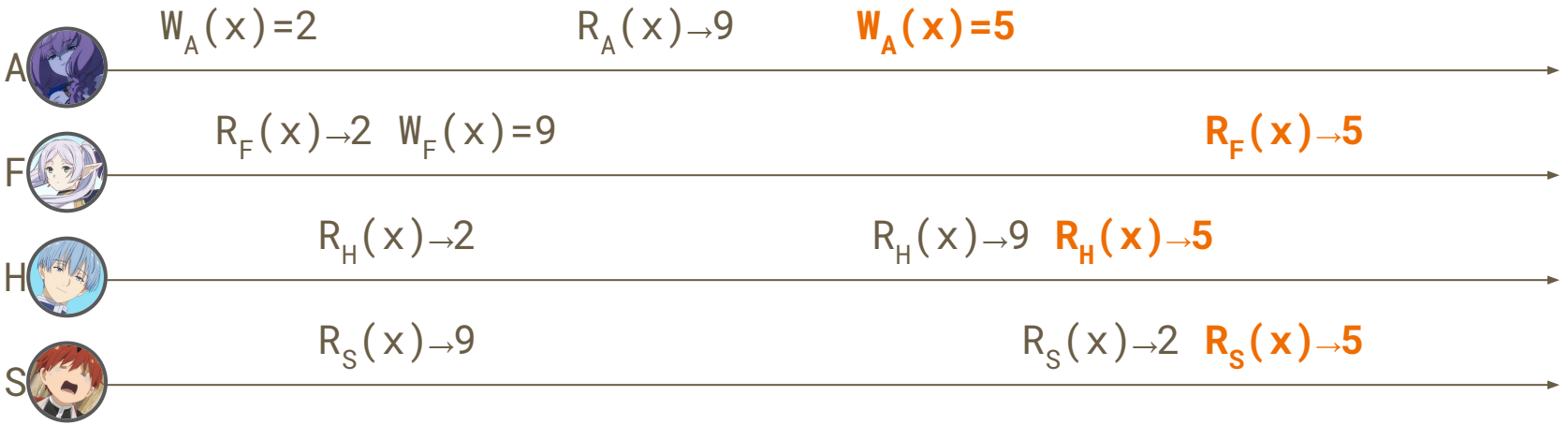
Partir con $W_a(x)=1$ o partir con $W_f(y)=2$

Consistencia Causal [Huto & Ahamad, 1990]



- Nodo F escribe 9 después de leer 2.
  - → Ningún nodos **puede** leer 9 antes de 2.

Consistencia Eventual [Vogels, 2009]



- Todos los nodos tienen finalmente el mismo valor (x=5).

Consistencia Centrada en el Cliente

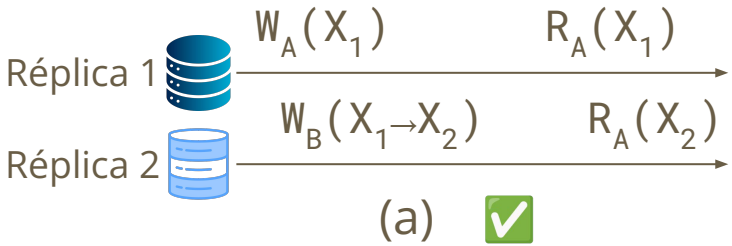
- Se centran en la experiencia consistente de **un cliente individual**, no en una vista global del sistema.
- Incluyen cuatro propiedades fundamentales:
  - *Monotonic Reads*
  - *Read Your Writes*
  - *Monotonic Writes*
  - *Writes Follow Reads*

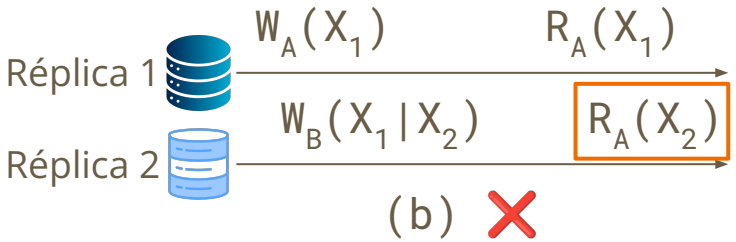
Fuente: Session Guarantees for Weakly Consistent Replicated Data [Terry et al., 1994]

Session Guarantees - Monotonic Reads

> *Si un cliente lee X_z , cualquier lectura realizada por ese mismo cliente a X devolverá X_z o alguna versión X_d tal que $W_q(X_z \rightarrow X_d)$*

Intuición: Una vez que has visto algo, no esperarías ver una versión anterior o inconsistente lo que viste recién.

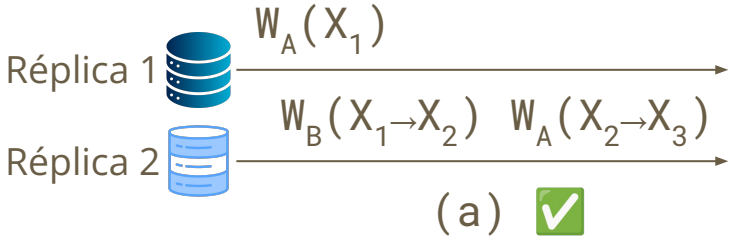


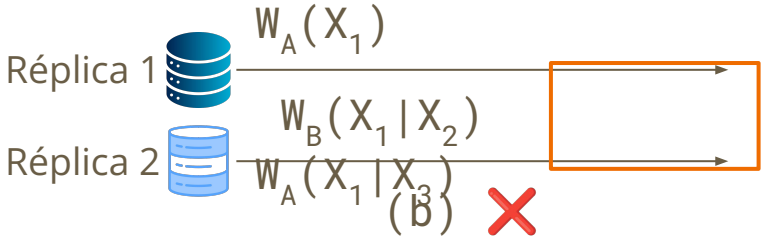


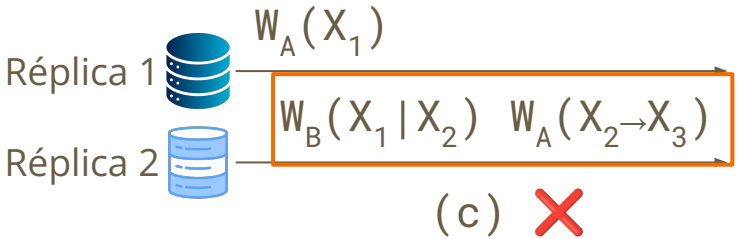
Session Guarantees - Monotonic Writes

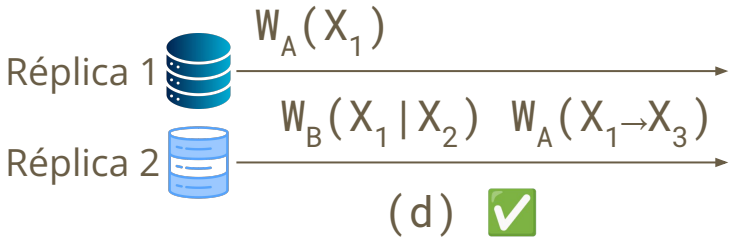
Si un cliente escribe X_z , cualquier escritura posterior realizada por ese mismo cliente a X debe ser alguna versión X_d tal que $W(X_z \rightarrow X_d)$

Intuición: Tus propias escrituras deben reflejar una progresión lógica.





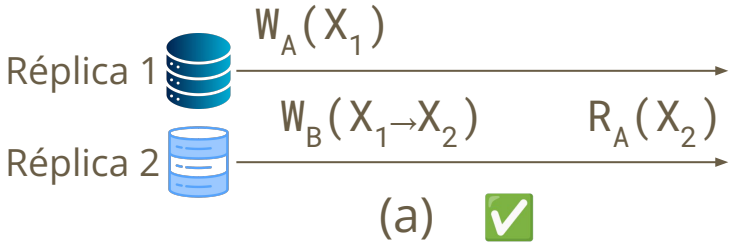


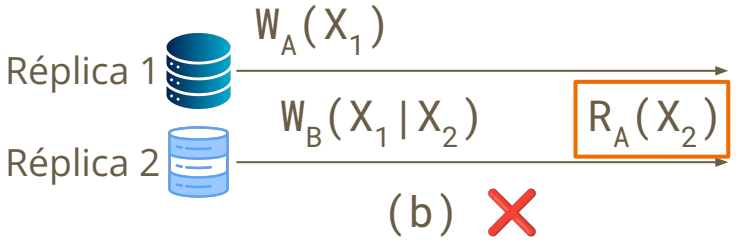


Session Guarantees - Read your writes

> *Si un cliente escribe X_z , cualquier lectura posterior realizada por ese mismo cliente a X debe ser X_z o alguna versión X_d tal que $W(X_z \rightarrow X_d)$*

Intuición: Si escribes algo, deberías poder leerlo inmediatamente después.

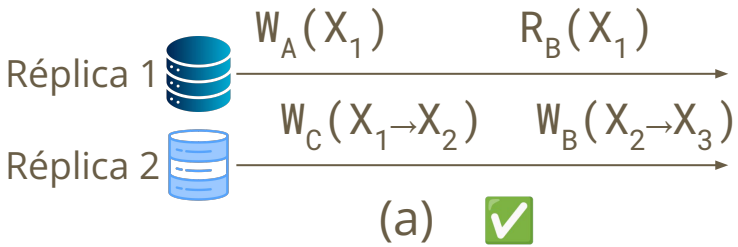


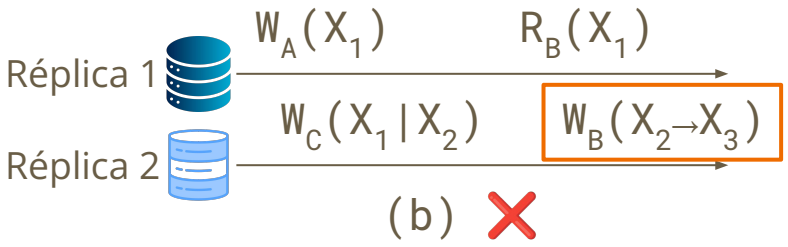


Session Guarantees - Writes Follow Reads

Si un cliente lee X_z , cualquier escritura posterior realizada por ese mismo cliente a X debe ser alguna versión X_d tal que $W(X_z \rightarrow X_d)$

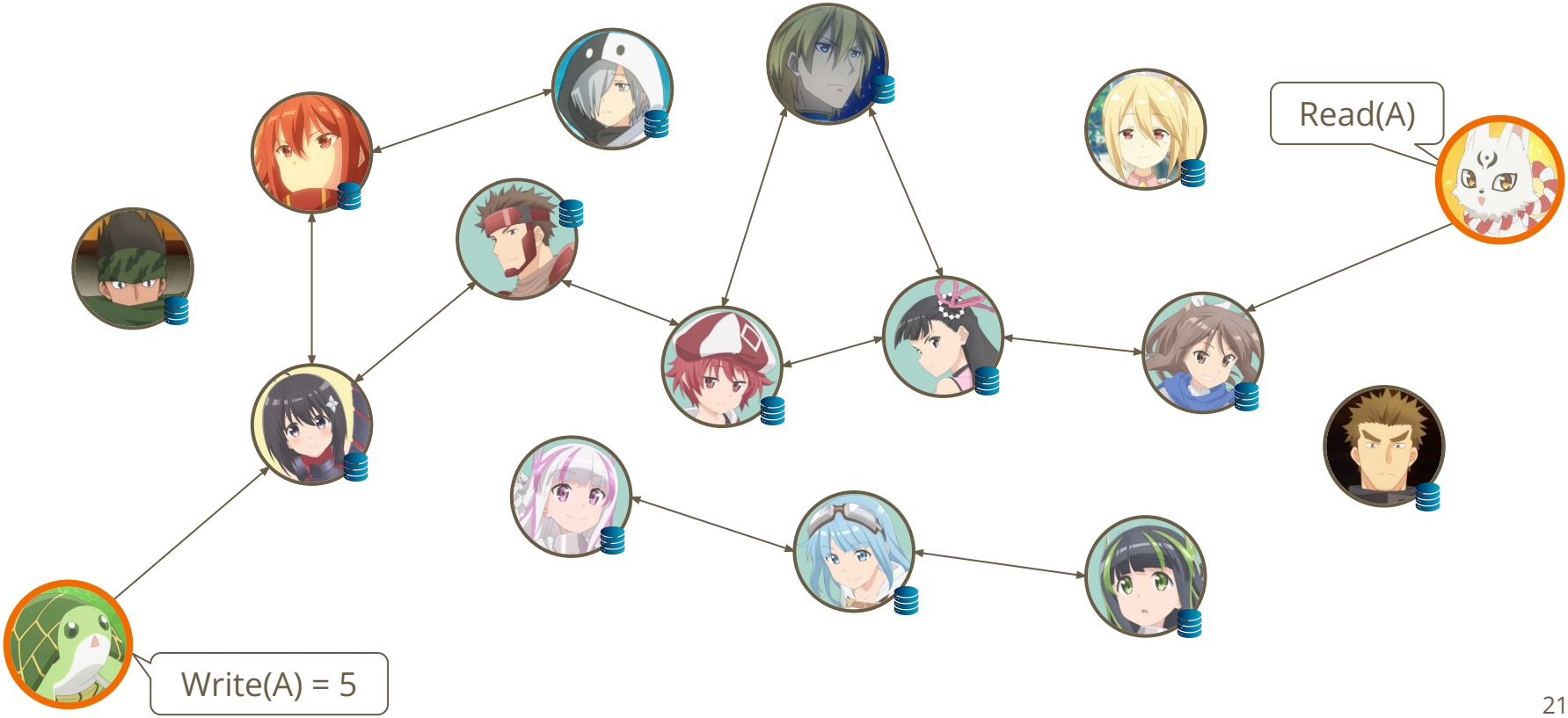
Intuición: Si lees algo y luego lo modificas, tu modificación debe basarse en la versión que leíste.



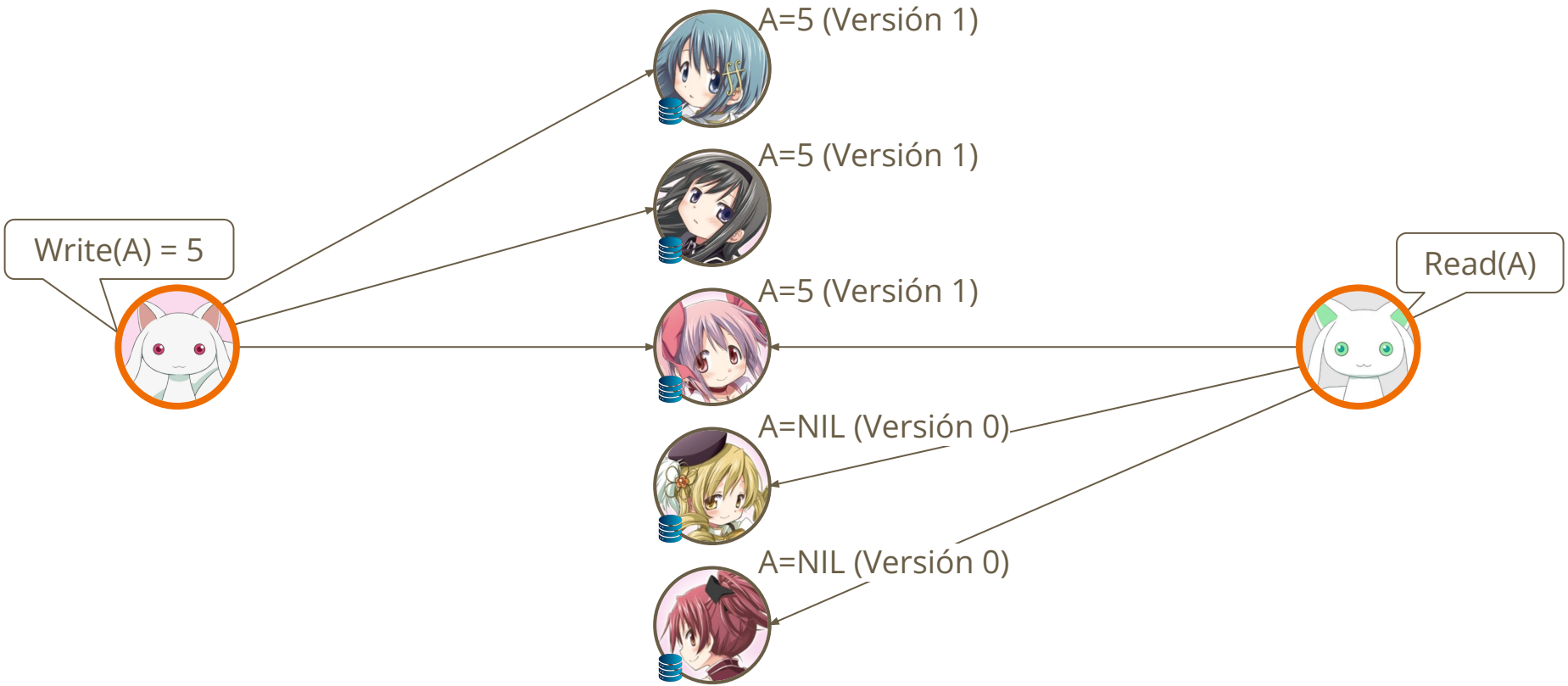


Replicación de Datos

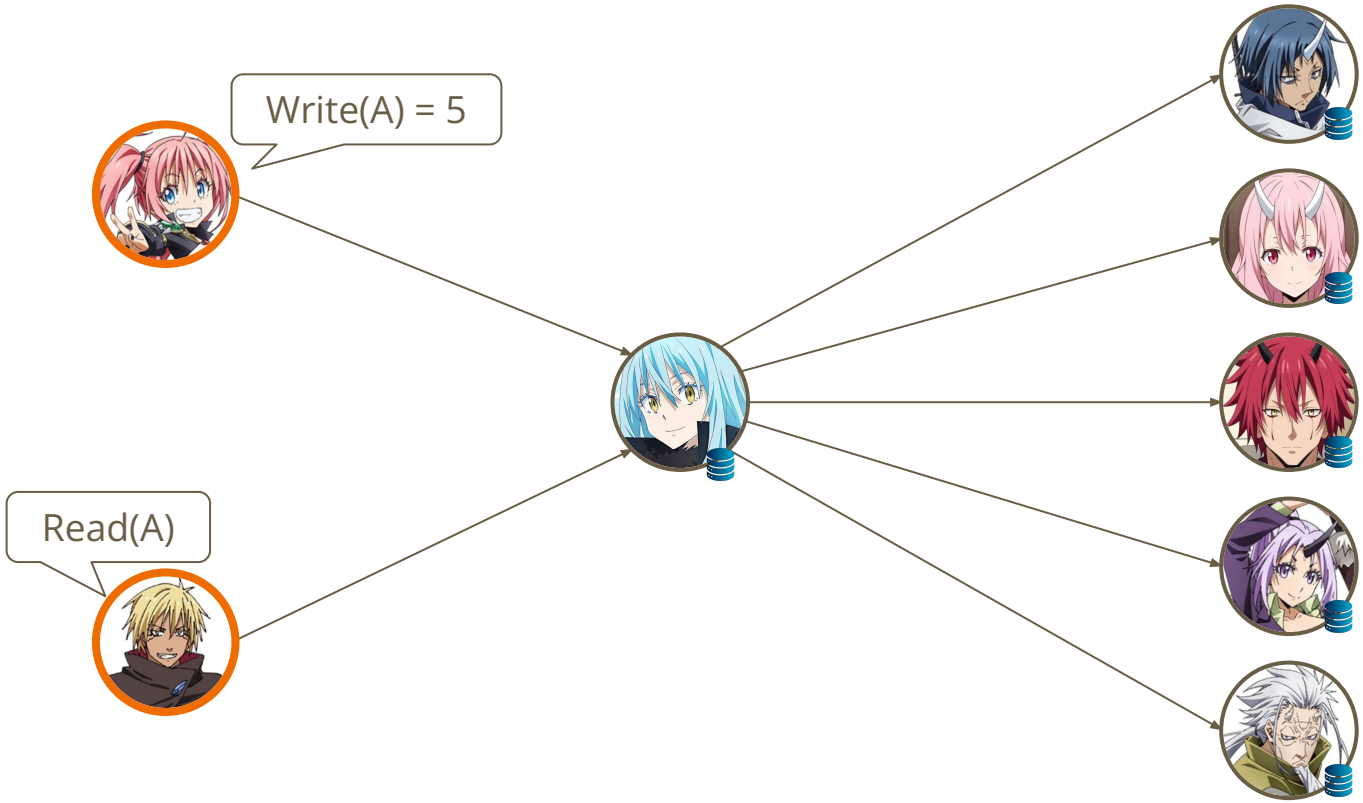
Protocolos de Replicación - *Gossip*



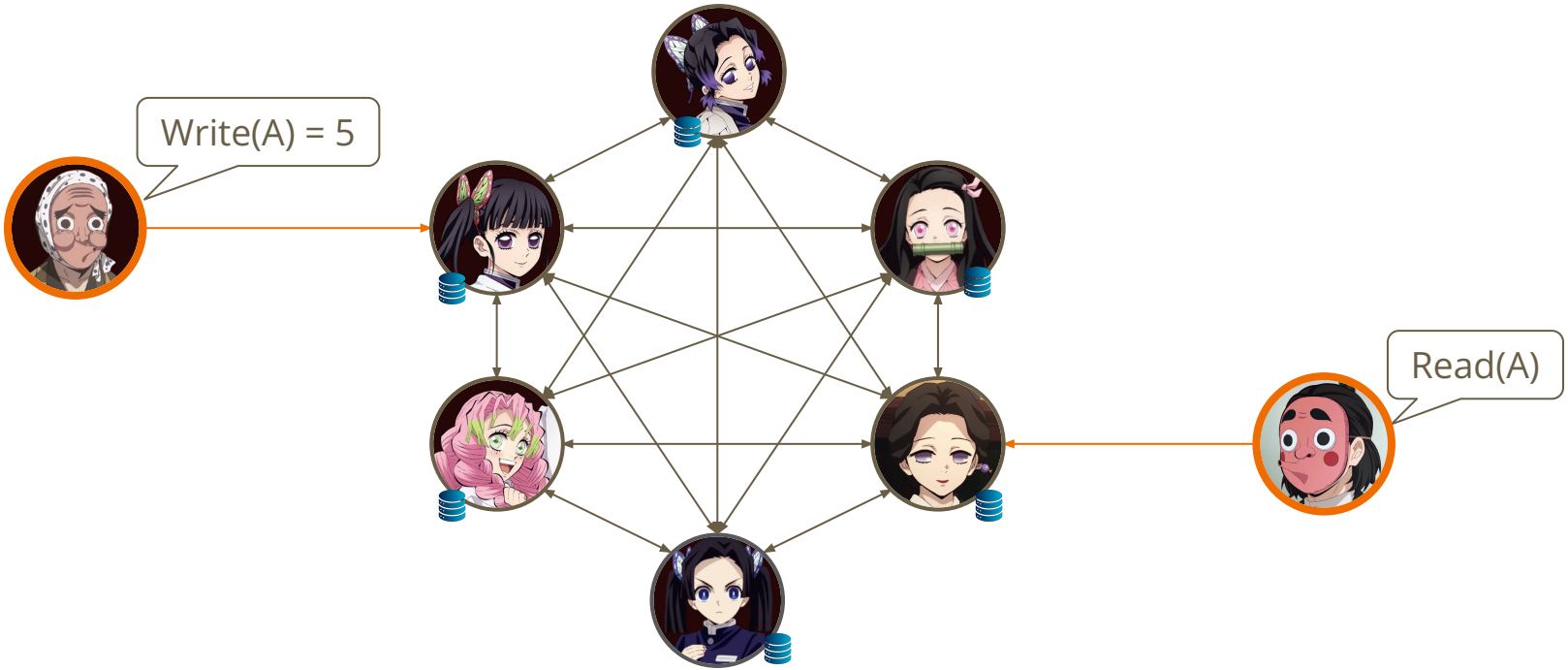
Protocolos de Replicación - *Quorum-Based*



Protocolos de Replicación - *Primary-Based Protocols*



Protocolos de Replicación - *Replicated-Write Protocols*

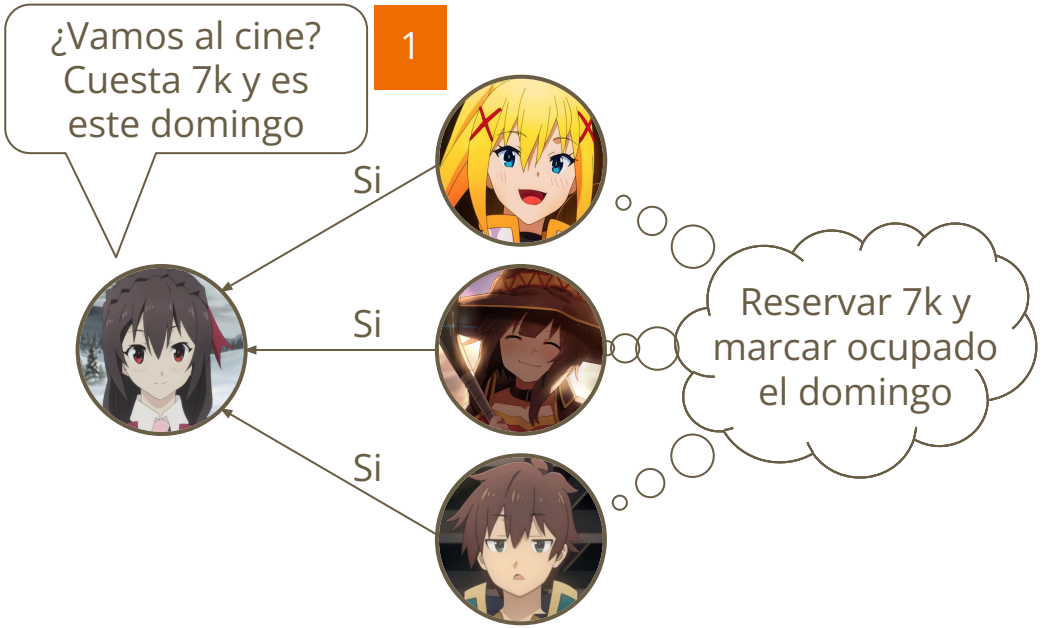


Transacciones Distribuidas

Transacción Distribuida y Propiedades ACID

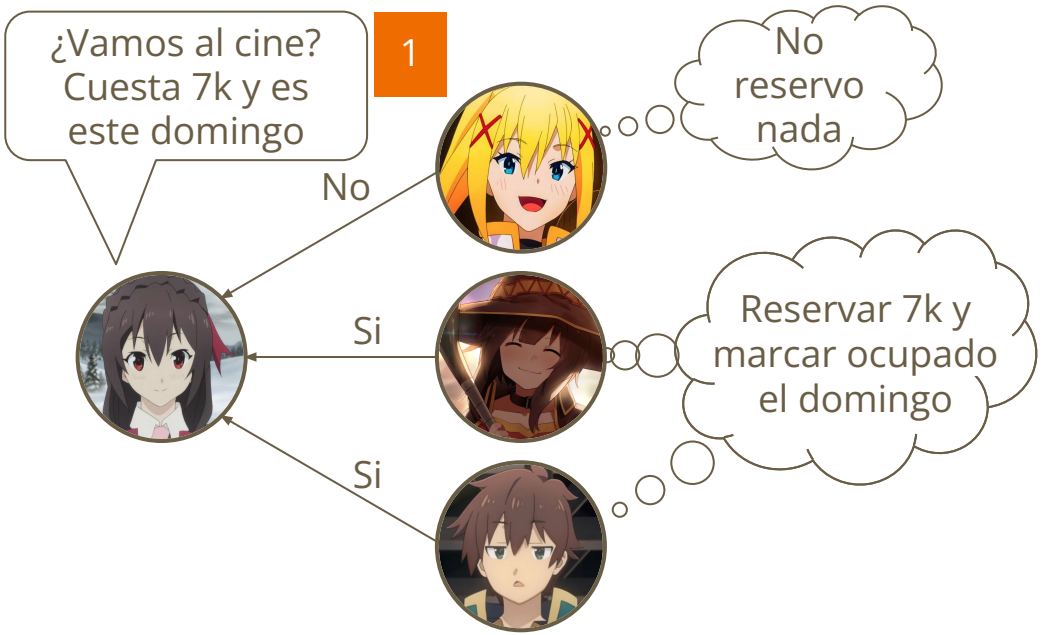
- Las propiedades ACID son los pilares de la fiabilidad de las transacciones:
  - A - *Atomicity* (Atomicidad)
  - C - *Consistency* (Consistencia/Coherencia)
  - I - *Isolation* (Aislamiento)
  - D - *Durability* (Durabilidad)

Commit de 2 fases (2PC)





Commit de 2 fases (2PC)

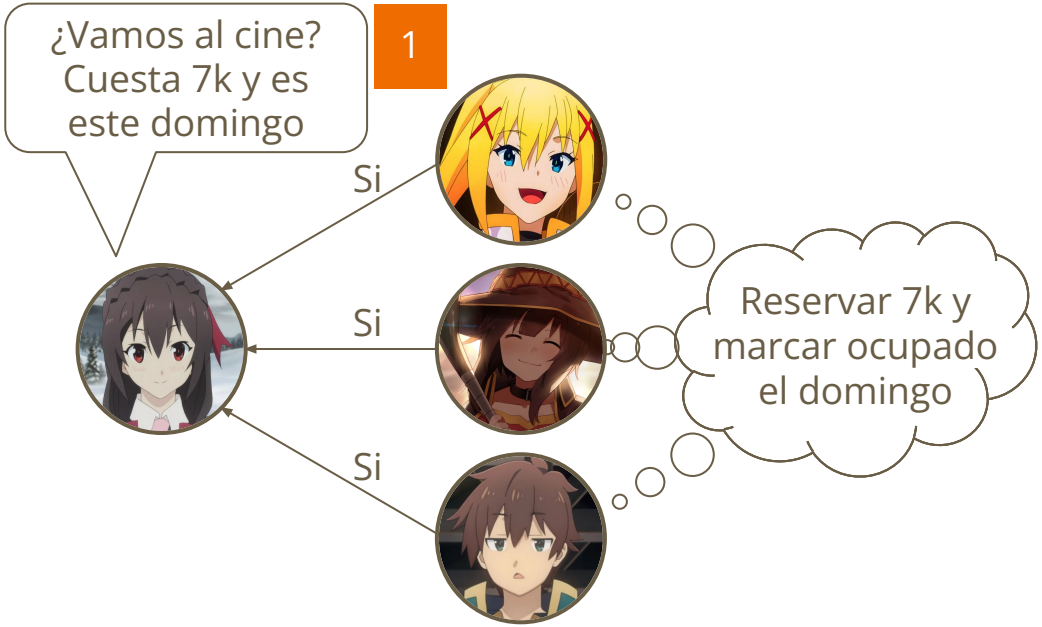




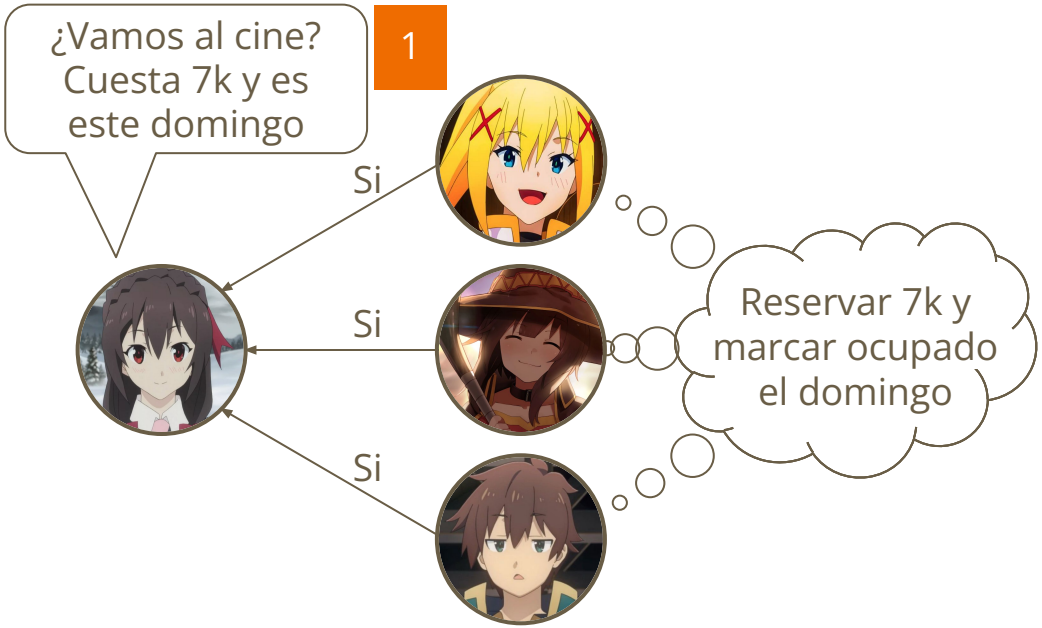
Commit de 2 fases (2PC)



Commit de 3 fases (3PC)



Commit de 3 fases (3PC)





Commit de 2 o 3 fases (2PC o 3PC)

- 2PC
  - Fase 1: Se pregunta y reservan variables. Se recopilan las respuestas.
  - Fase 2: Commitear si todos dicen que si, abortar si al menos 1 dice que no.
- 3PC
  - Fase 1: Se pregunta y reservan variables. Se recopilan las respuestas.
  - Fase 2: Avisar que todos dijeron que si.
  - Fase 3: Commitear si todos dijeron que si o abortar si no existe nadie que recibió el aviso de la fase 2.

Importante: al momento de reservar variables. Estas pueden, por ejemplo, usando Lock para que nadie acceda, o protegiendo (como en la tarea) para evitar ciertos conflictos.

Control de Concurrencia

Estrategias de Control de Concurrencia - Pesimista



Estrategias de Control de Concurrencia - Optimista



Estrategias de Control de Concurrencia - Optimista

- Existen 2 tipos de validación para detectar el conflicto *Read-Write.*
- Sea T_v la transacción a validar

Aspecto	***Forward* Validation**	***Backward* Validation**
T_v se valida contra	Transacciones **activas**	Transacciones **ya consolidadas**
Operación a validar	**Escrituras** de T_v	**Lecturas** de T_v
Intención	No invalidar lecturas futuras	Asegurar que las lecturas fueron válidas
Riesgo de abortos	Depende del solapamiento con transacciones activas	Depende del solapamiento con *commits* previos
Complejidad	Requiere seguimiento de transacciones activas	Mirar historial de *commits* consolidadas desde que empezó T_v

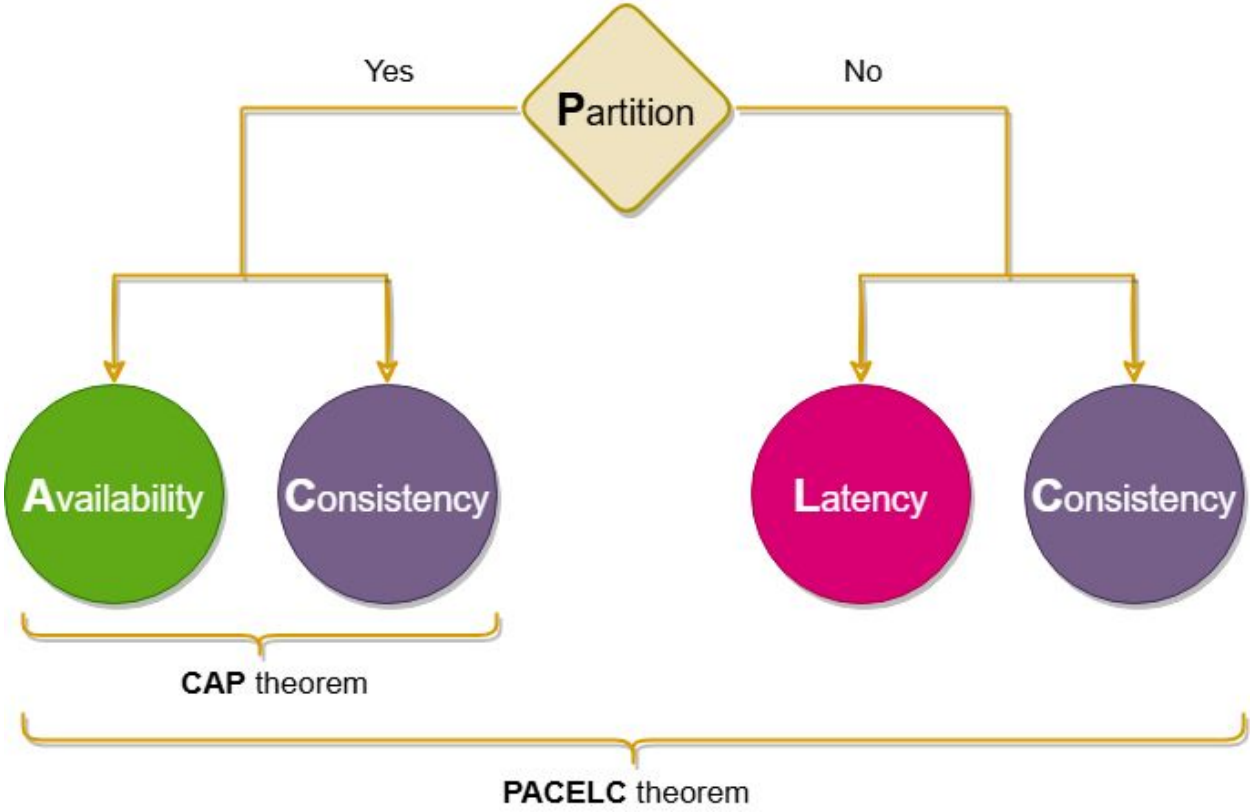
Resolución de Conflictos Concurrentes

- En caso de permitir que existan los conflictos, existen diversas formas de resolverlo

Protocolo	Cómo resuelve
Last Write Wins	Se queda con el valor con *timestamp* mayor
Versiones	Genera más de un recurso
***Merge* manual**	Un usuario decide qué versión conservar
***Merge* semántico**	Usa lógica del dominio para combinar versiones
Operational Transformation	Reescribe cambios como operaciones para que puedan aplicarse en cualquier orden
Conflict-free Replicated Data Types	Usa estructuras especiales y operaciones algebraicas para combinar sin conflicto.

Teorema PAC y PACELC

Teorema PAC y PACELC

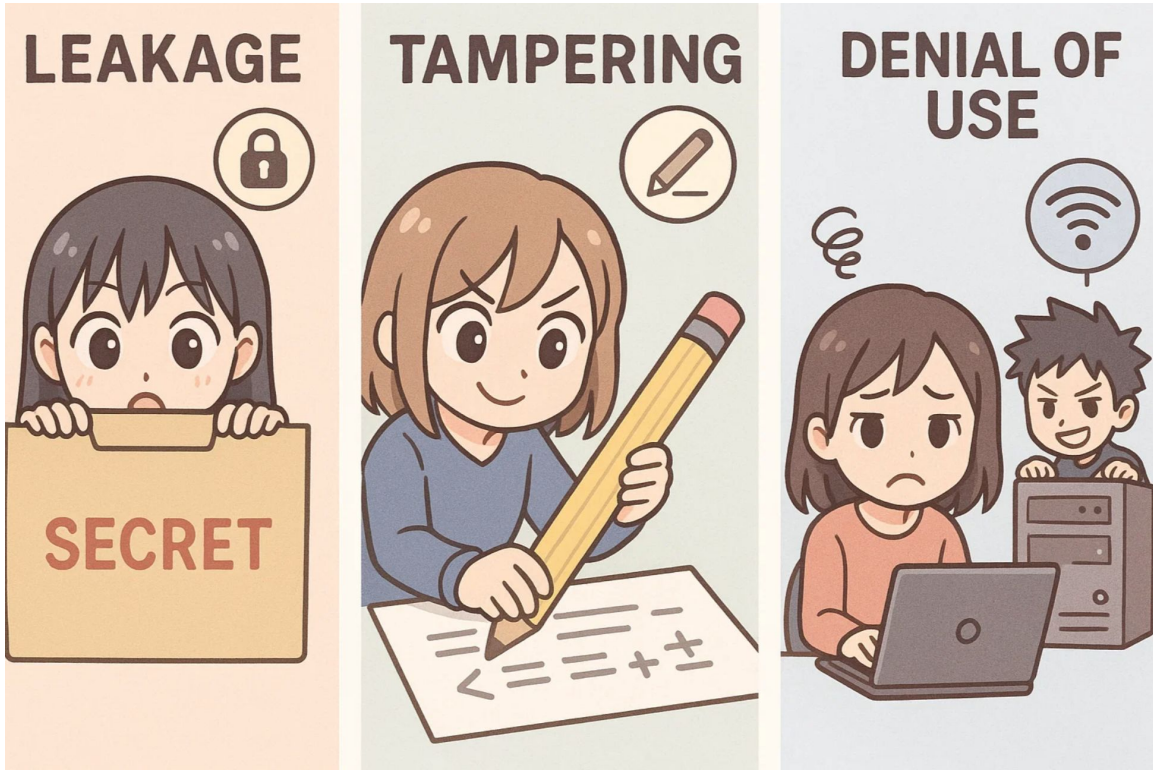


Introducción a ataques

Ataque a un sistema

- En sistemas distribuidos, comprender las amenazas y ataques es fundamental para diseñar sistemas seguros y confiables.
- Los ataques a un sistema buscan comprometer alguno de los tres pilares de la seguridad.
  - **Confidencialidad** → Garantiza que no se divulgue la información privada.
  - **Integridad** → Garantiza que los datos se envían y almacenan correctamente.
  - **Disponibilidad** → Garantiza que el sistema esté disponible cuando corresponda.

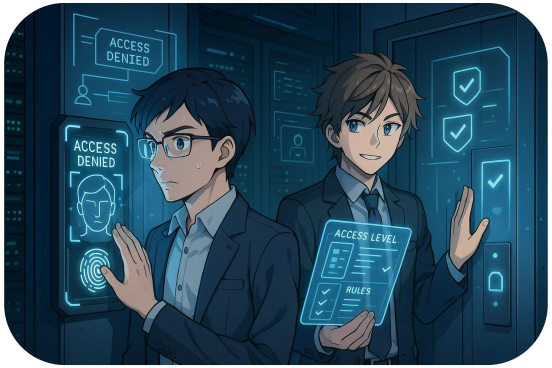
Ataque a un sistema - Tipos de Amenazas



Fuente: OpenAI. (2025). [Representación de tipos de amenazas] [Imagen generada por inteligencia artificial con Sora]. https://openai.com/sora

Ataque a un sistema - Medidas de seguridad

Autenticación y autorización



Cifrado



Canales seguros



Monitoreo

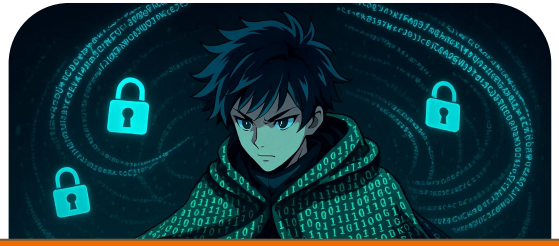


Ataque a un sistema - Medidas de seguridad

- Autenticación y autorización



- Cifrado



¿Qué gano y qué pierdo con estas medidas?

- Canales seguros



- Monitoreo



Cifrado de mensajes

Llave simétrica o Asimétrica

- Simétrica: ambas partes ocupan la misma llave para encriptar y desencriptar los mensajes.
- Asimétrica: cada parte tiene 2 llaves, una pública y otra privada.
  - Enviar mensaje protegido:
    - A cifra con la llave pública de B
    - B descifra con su llave privada.
  - Firmar mensaje para garantizar autenticidad
    - A cifra el mensaje con su llave privada
    - Solo con la llave pública de A se logra acceder al mensaje y se puede validar que la "firma" fue de A.

Firma digital con llave asimétrica y funciones de hash.

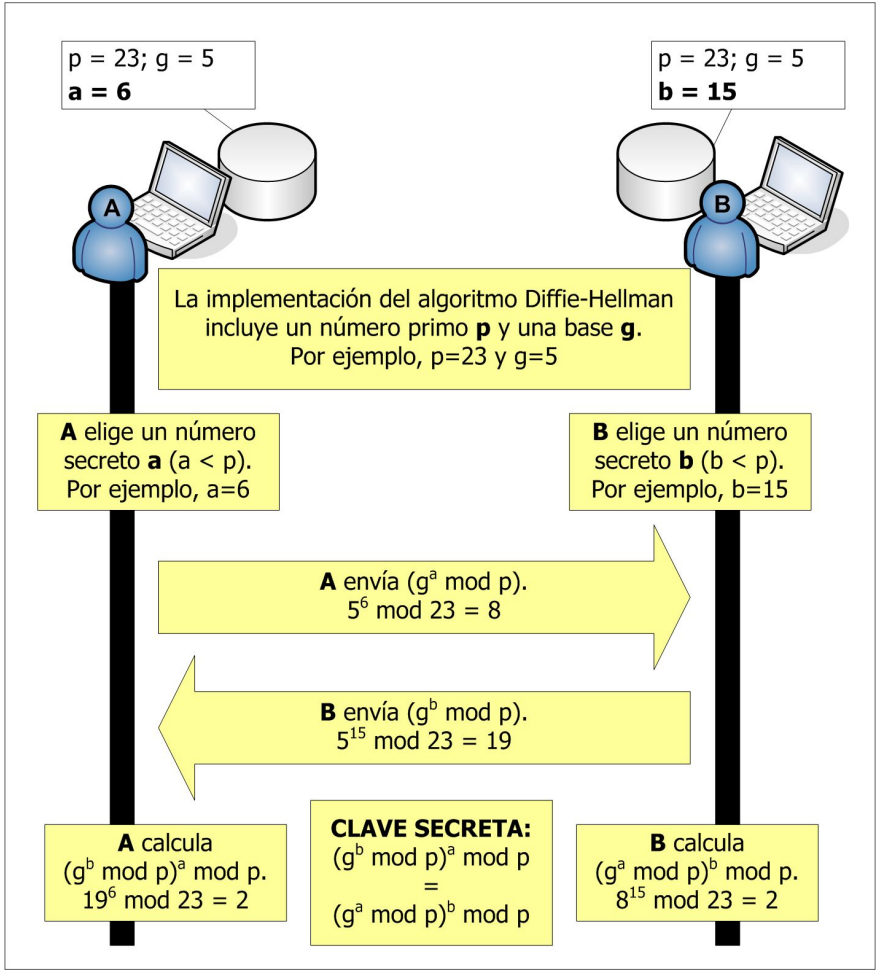
- Asimétrica: cada parte tiene 2 llaves, una pública y otra privada.
  - Firmar mensaje para garantizar autenticidad
    - A cifra el mensaje con su llave privada
    - Solo con la llave pública de A se logra acceder al mensaje y se puede validar que la "firma" fue de A.

Firma digital con llave asimétrica y funciones de hash.

- Asimétrica: cada parte tiene 2 llaves, una pública y otra privada.
  - Firmar mensaje para garantizar autenticidad
    - A cifra el mensaje con su llave privada
    - Solo con la llave pública de A se logra acceder al mensaje y se puede validar que la "firma" fue de A.
  - **Con funciones de hash**
    - A *hashea* el mensaje original y cifra el *hash* con su llave privada.
    - A envía mensaje original y el hash cifrado.
    - B descifra el hash con la llave pública de A (valida al emisor).
    - B aplica hash al mensaje original y lo compara con el hash descifrado.
    - Si son iguales, confía que el documento fue efectivamente enviado por A.

Diffie hellman

- A y B definen, cada uno, un número secreto.
- A y B se comparten parámetros públicos.
- Cada uno aplica una operación matemática específica usando su número secreto + los parámetros públicos.
  - Esto genera un número igual para ambos que usan como llave secreta (cifrado simétrico)



Híbridos

- Usan ambos tipos de cifrado
  - Cifrado asimétrico para el envío de una llave secreta
  - Usan la llave secreta para comunicarse mediante cifrado simétrico
- Ejemplo: TLS 1.3
  - Cliente comparte llave pública al servidor. Responde con llave pública de servidor y certificado firmado por él.
  - Cliente y servidor utilizan la llave pública del otro + la llave privada de cada uno para generar una llave secreta en común.
  - Desde ahora se usa la llave secreta para comunicarse.

IIC2523
Sistemas Distribuidos

Hernán F. Valdivieso López
(2025 - 2 / Clase 20)