

---

# IIC2523

# Sistemas Distribuidos

— Hernán F. Valdivieso López —  
(2025 - 2 / Clase 11)

---

# Avisos importantes

1. **Respondan**, por favor, la ETC (Encuesta Temprana de Curso) 🧐 🧐
  - a. Es la primera vez que me enfrento a un curso 100% nuevo, puedo equivocarme pero si no me lo dicen, no puedo mejorar.
  - b. Esta encuesta no provocará que me echen del trabajo, pero si me dará *feedback* que puedo intentar aplicar para lo que queda del curso.



← Este seré yo si no responden la ETC o solo me llegan insultos.

# Tolerancia a fallos

¿Qué es un fallo? ¿Cómo detectarlo? ¿Qué hacer?

# Temas de la clase

1. Conceptos fundamentales y tipos de fracasos
2. *Failure Detection* y *Failure masking*
3. Teorema de Imposibilidad de Fischer, Lynch, Patterson
4. Semántica de RPC en Presencia de Fallos

# Conceptos fundamentales

Confiabilidad

Fallo, fracaso y error ¿son lo mismo?

Tipos de fallos

---

# Conceptos fundamentales

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

# 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.*

# 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 - Propiedades Confiabilidad

## Propiedad de **Disponibilidad**



- ◆ Un sistema esté listo para ser usado inmediatamente, o la probabilidad de que esté operando correctamente en un momento dado.
- ◆ Por ejemplo, si un servidor tiene un 5% de probabilidad de fallar, dos servidores replicados aumentan la disponibilidad al 99.75%.

## Propiedad de **Fiabilidad**



- ◆ Un sistema puede funcionar continuamente sin fallos durante un período de tiempo considerable.
- ◆ A diferencia de la disponibilidad, la fiabilidad mide el rango de tiempo que se puede ocupar sin fallos.

# Conceptos fundamentales - Propiedades Confiabilidad

Propiedad de **Seguridad** 

- ◆ Si un sistema falla, no se producirán consecuencias catastróficas.

Propiedad de **Mantenibilidad** 

- ◆ La facilidad con la que un sistema fallido puede ser reparado después de una falla.

# Conceptos fundamentales - Propiedades Confiabilidad

## Ejemplos

- ◆ Un sistema que administra una central nuclear exige una alta \_\_\_\_\_.
- ◆ Un código que ocupa varias librerías, pero está todo modularizado para poder cambiar una librería si es que falla. Eso garantiza una alta \_\_\_\_\_.
- ◆ Un sistema que funciona perfecto durante el día, pero de 2 a 6 de la madrugada entra en mantención y nadie puede acceder tiene una alta \_\_\_\_\_.
- ◆ Un aplicación para ver el clima debería tener una alta \_\_\_\_\_.
- ◆ Una plataforma para rendir una prueba, donde tienes 1 semana para hacerla, debería tener una alta \_\_\_\_\_.

# Conceptos fundamentales - Propiedades Confiabilidad

## Ejemplos

- ◆ Un sistema que administra una central nuclear exige una alta Seguridad.
- ◆ Un código que ocupa varias librerías, pero está todo modularizado para poder cambiar una librería si es que falla. Eso garantiza una alta Mantenibilidad.
- ◆ Un sistema que funciona perfecto durante el día, pero de 2 a 6 de la madrugada entra en mantención y nadie puede acceder tiene una alta Fiabilidad.
- ◆ Un aplicación para ver el clima debería tener una alta Disponibilidad.
- ◆ Una plataforma para rendir una prueba, donde tienes 1 semana para hacerla, debería tener una alta Fiabilidad.

# Conceptos fundamentales - Propiedades Confiabilidad

## Ejemplos

- ◆ Un sistema que administra una central nuclear exige una alta Seguridad.
- ◆ Un código que ocupa varias librerías, pero está todo modularizado para poder cambiar una librería si es que falla. Eso garantiza una alta Mantenibilidad.
- ◆ Un sistema que funciona perfecto durante el día, pero de 2 a 6 de la madrugada entra en mantención y nadie puede acceder tiene una alta Fiabilidad.
- ◆ Un aplicación para ver el clima debería tener una alta Disponibilidad.
- ◆ Una plataforma para rendir una prueba, donde tienes 1 semana para hacerla, debería tener una alta Fiabilidad.

! Se espera que un sistema cumpla la mayor cantidad de estas propiedades !

# Conceptos fundamentales - Métricas

Existen 3 métricas que se tienden a ocupar para medir la confiabilidad de un sistema.

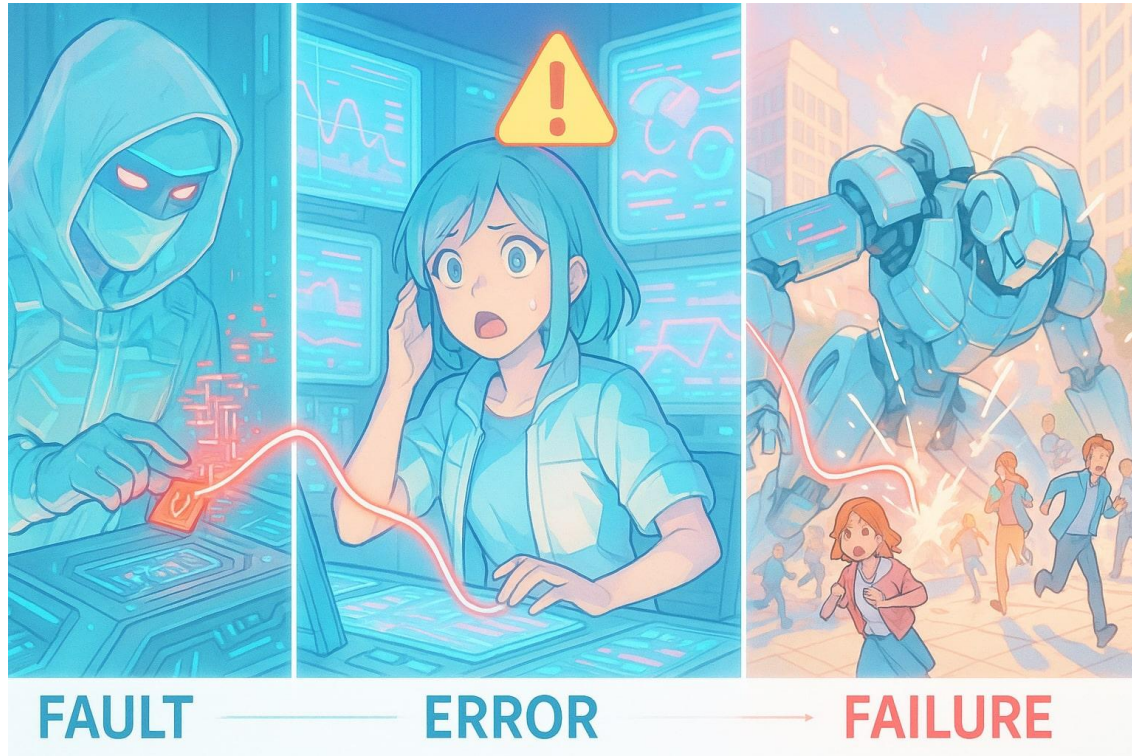
- ◆ *Mean Time To Failure* (**MTTF**): Tiempo promedio hasta que algún componente/proceso/nodo falle.
- ◆ *Mean Time To Repair* (**MTTR**): Tiempo promedio que se necesita para reparar el elemento fallado.
- ◆ *Mean Time Between Failures* (**MTBF**): Tiempo promedio entre fallas. Simplemente  $MTTF + MTTR$ .

# Conceptos fundamentales - Falla, Error y Fracaso

¿Son 3 conceptos para describir lo mismo?

# Conceptos fundamentales - Falla, Error y Fracaso

¿Son 3 conceptos para describir lo mismo? No.



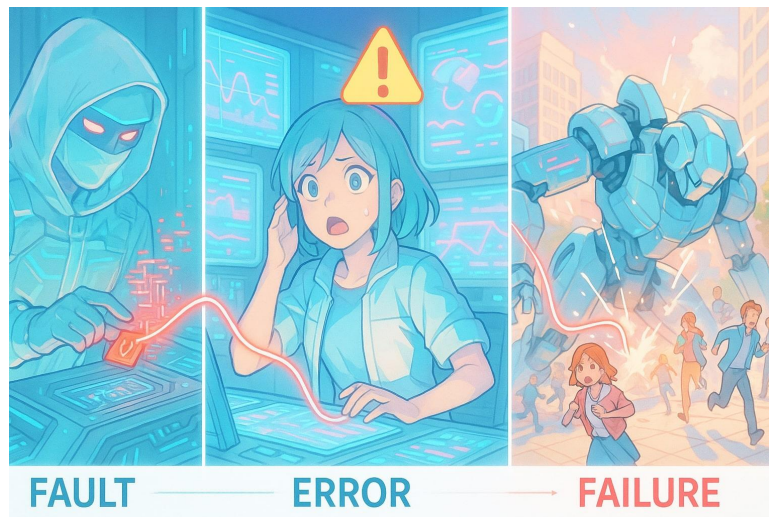


# Conceptos fundamentales - Falla, Error y Fracaso

¿Son 3 conceptos para describir lo mismo? No.

- ◆ Cada concepto describe un aspecto distinto de una operación que no logró ser ejecutada con éxito en el sistema. Se define la siguiente cadena de causalidad:

- ◆ **Fallo (*Fault*) → Error → Fracaso (*Failure*)**



# Conceptos fundamentales - Falla, Error y Fracaso

¿Son 3 conceptos para describir lo mismo? No.

- ◆ Cada concepto describe un aspecto distinto de una operación que no logró ser ejecutada con éxito en el sistema. Se define la siguiente cadena de causalidad:
  - ◆ **Fallo (*Fault*) → Error → Fracaso (*Failure*)**
- ◆ En términos simples, el fallo es la causa de un error. El error es justamente lo que no está ocurriendo y el fracaso es el estado final de la operación.
  - ◆ Por ejemplo, un código espera un número y se le entrega un *string*.
    - ◆ Fallo (*Fault*): entregar un *string*.
    - ◆ Error: no se puede procesar el dato entregado.
    - ◆ Fracaso (*Failure*): el código no se ejecutó.

# Conceptos fundamentales - Falla, Error y Fracaso

¿Son 3 conceptos para describir lo mismo? No.

- ◆ Cada concepto describe un aspecto distinto de una operación que no logró ser ejecutada con éxito en el sistema. Se define la siguiente cadena de causalidad:
  - ◆ **Fallo (*Fault*) → Error → Fracaso (*Failure*).**
- ◆ En términos simples, el fallo es la causa de un error. El error es justamente lo que no está ocurriendo y el fracaso es el estado final de la operación.
- ◆ **Un error no necesariamente lleva al fracaso** si un sistema posee mecanismos para tolerar la falla. Por ejemplo, servidores de respaldo, algoritmos que logran finalizar incluso si se cae el nodo, etc.

# Conceptos fundamentales - Falla, Error y Fracaso

## Fallo (*Fault*)

- ◆ Es la **causa** de un error.
- ◆ Puede ser un defecto de *hardware*, *software*, o un error humano.
- ◆ Se pueden categorizar según su ocurrencia como: transitorios, intermitentes o permanentes.

# Conceptos fundamentales - Falla, Error y Fracaso

## Fallo (*Fault*)

- ◆ Es la **causa** de un error.
- ◆ Puede ser un defecto de *hardware*, *software*, o un error humano.
- ◆ Se pueden categorizar según su ocurrencia como: transitorios, intermitentes o permanentes.
  - ◆ **Transitorios**: Ocurren una vez y luego desaparecen.
    - ◆ El usuario ingresa mal un dato. Se perdió un paquete en TCP.
  - ◆ **Intermitentes**: Aparecen, luego desaparece por sí misma y reaparecen sin intervención.
    - ◆ Un contacto eléctrico flojo.
  - ◆ **Permanentes**: Persisten hasta que el componente defectuoso es reemplazado.
    - ◆ Bugs de *software* o discos duros dañados.

# Conceptos fundamentales - Falla, Error y Fracaso

## Error

- ◆ Es la manifestación interna de una falla en un sistema.

## Fracaso (*Failure*)

- ◆ Es el instante cuando un sistema no puede cumplir sus promesas; el servicio no puede ser provisto como se esperaba.

# Conceptos fundamentales - Falla, Error y Fracaso

## Error

- ◆ Es la manifestación interna de una falla en un sistema.

## Fracaso (*Failure*)

- ◆ Es el instante cuando un sistema no puede cumplir sus promesas; el servicio no puede ser provisto como se esperaba.

El objetivo principal es la tolerancia a fallos, que significa que un sistema puede proporcionar sus servicios incluso en presencia de fallos. Nunca llega al fracaso (*failure*) incluso si ocurre 1 o más fallos (*fault*)

# Conceptos fundamentales - Tipos de Fracasos

- ◆ En un sistema distribuido, los fracasos son parciales, es decir, algunos componentes fallan mientras otros continúan funcionando.
- ◆ Hay diferentes tipos de fracasos que puede tener un sistema distribuido:
  - ◆ Caída de uno o más nodos
  - ◆ Omisión de mensaje
  - ◆ Tiempo de respuesta
  - ◆ Respuesta incorrecta
  - ◆ Falla bizantina
  - ◆ Partición de red



# Conceptos fundamentales - Tipos de Fracasos

- ◆ Caída de uno o más nodos (*crash failure*)
  - ◆ El nodo funcionaba correctamente hasta que en algún momento se apaga o se cuelga y deja de ejecutar su programa.
- ◆ Omisión de mensaje (*omission failure*)
  - ◆ El nodo no responde a solicitudes entrantes. Se divide en 2 tipos de omisión
    - ◆ *Receive omission*: No recibe mensajes entrantes.
    - ◆ *Send omission*: No logra enviar mensajes.
- ◆ Tiempo de respuesta (*timing failure*)
  - ◆ El servidor responde, pero lo hace fuera del intervalo de tiempo esperado.

# Conceptos fundamentales - Tipos de Fracasos

- ◆ Respuesta incorrecta (*response failure*)
  - ◆ El servidor responde, pero de forma incorrecta a lo esperado. Puede ser:
    - ◆ *Value failure*: El valor de la respuesta está mal.
    - ◆ *State-transition failure*: El nodo sigue un flujo de control incorrecto, es decir, actúa fuera del orden esperado.
- ◆ Falla bizantina (*arbitrary failure*)
  - ◆ El nodo se comporta de manera impredecible.
- ◆ Particiones de Red (*network partitions*)
  - ◆ La red puede dividirse en subgrupos donde la comunicación entre ellos es imposible, incluso si los nodos individuales dentro de los subgrupos siguen funcionando.

# *Failure Detection y Failure masking*

Mecanismos para detectar un  
fracaso

Mecanismos para ser tolerante  
a fallos

---

# Failure Detection

- ◆ Existen diferentes mecanismos para detectar el fallo de un sistema. Aunque no pueden precisar el tipo de fallo.
  - ◆ **Sondas (*probes*)**: mecanismo activo donde se envían mensajes del tipo "¿sigues vivo?" a los diferentes nodos.
  - ◆ **Latidos (*heartbeats*)**: mecanismo pasivo donde se espera mensajes de otro nodo para entender que "sigue vivo". Funciona solo si se garantiza un buen canal de comunicación.
  - ◆ **Tiempo de espera (*timeouts*)**: las mensajes poseen un tiempo máximo de respuesta, si no se cumple se sospecha que el nodo ha fallado. Incluso si un mensaje no espera una respuesta, se puede obligar al nodo a responder con un "ACK" (confirmar que recibió el mensaje).

# Failure Detection

## Desafíos

- ◆ En sistemas donde no hay límites en las velocidades de ejecución o los retrasos de mensajes, un *timeout* solo indica que un proceso no responde, pero...
  - ◆ ¿Se bloqueó? ¿está lento? ¿el mensaje se perdió?
  - ◆ Esto puede generar falsos positivos (creer que el nodo falló cuando fue la red).

# Failure Detection

## Desafíos

- ◆ En sistemas donde no hay límites en las velocidades de ejecución o los retrasos de mensajes, un *timeout* solo indica que un proceso no responde, pero...
  - ◆ ¿Se bloqueó? ¿está lento? ¿el mensaje se perdió?
  - ◆ Esto puede generar falsos positivos (creer que el nodo falló cuando fue la red).
- ◆ Por lo mismo, existen los mecanismos "eventualmente perfectos" que intentan adaptarse al sistema y sus posibles fallas.
  - ◆ Ajustar *timeout* para los distintos nodos o enviar más de una petición para asegurar que al menos 1 mensaje llegue o para contrastar respuestas.

# *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.

# 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.
  - ◆ Sin embargo, la redundancia puede impactar el rendimiento, tamaño, peso y consumo de energía de un sistema.



# 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.
  - ◆ Sin embargo, la redundancia puede impactar el rendimiento, tamaño, peso y consumo de energía de un sistema.
  - ◆ Se distinguen 4 tipos de redundancia: *Hardware*, *Información*, *Tiempo*, *Software*.

# Failure masking - Redundancia

## Redundancia de *hardware* (enfoque pasivo)

- ◆ Se ignora la ocurrencia de una falla.
- ◆ **TMR - Triple Modular Redundancy**: se triplica el *hardware* y luego se hace votación para determinar la salida del sistema.
  - ◆ No se preocupa por determinar la falla y corregir, solo deja que el Quórum escoja, idealmente, la respuesta correcta.
  - ◆ Se puede extender a N *hardware* (con N impar) para detectar más fallas simultáneas.

# Failure masking - Redundancia

## Redundancia de *hardware* (enfoque activo)

- ◆ Detectan el componente fallado y lo reparan o reemplazan.
- ◆ **Standby Sparing**: Uno o más módulos están operativos y hay otros de respaldo (*spares*). Si se detecta y localiza un fallo, el módulo defectuoso se retira y se reemplaza por un repuesto. El defectuoso luego se intenta reparar.
  - ◆ **Hot Standby Sparing**: Los repuestos operan en sincronía con los módulos en línea, listos para tomar el control de inmediato.
  - ◆ **Cold Standby Sparing**: Los repuestos están apagados hasta que se necesitan.

# Failure masking - Redundancia

## Redundancia de información

- ◆ Se añade información adicional a los datos para detectar y/o reconstruir el dato.
- ◆ Algunos ejemplos son:
  - ◆ **Código duplicado**: toda la información está duplicada N veces.
  - ◆ **Checksum**: bloque con la suma de todos los *bytes*. Existen diferentes fórmulas para sumar.
  - ◆ **Código Hamming**: se agregan *bits* extras de paridad para detectar si un *bit* de la información original falló y corregirlo.
  - ◆ **Códigos Reed-Solomon**: Los datos se tratan como polinomios y se agregan símbolos (redundantes) que permiten reconstruir los datos originales si parte de ellos se pierde o daña.

# Failure masking - Redundancia

## Redundancia de tiempo

- ◆ Reducir el *hardware* extra a expensas de usar tiempo adicional.
- ◆ Se realiza la misma operación dos o más veces y se comparan los resultados. Si hay discrepancia, se repite la computación para ver si el error desaparece.
- ◆ Se incluye concepto de votación entre ejecuciones para determinar el resultado a enviar.
- ◆ Para detectar fallos permanentes, se realizan operaciones con resultados esperados.


# Failure masking - Redundancia

## Redundancia de *software*

- ◆ Un nodo posee más programas/procesos para asegurar el correcto funcionamiento.
- ◆ Incluir programas para verificar información con conocimiento previo o verifican el estado del sistema (pruebas de memoria, prueba a la ALU, etc).
- ◆ También se pueden recurrir a técnicas a la redundancia de *hardware*, pero ahora con programas/procesos.
  - Varios procesos concurrentes y luego se vota por la respuesta.
  - Disponer de procesos de respaldo.

# Teorema de Imposibilidad de Fischer, Lynch, Patterson

# Teorema de Imposibilidad de Fischer, Lynch, Patterson

- ◆ Teorema creado en 1985 por Michael J. Fischer, Nancy A. Lynch y Michael S. Paterson.
- ◆ Resultado fundamental en la teoría de los sistemas distribuidos.
- ◆ Ganó el premio *Dijkstra* el 2001.
  - ◆ El premio se otorga a trabajos destacados sobre los principios de la computación distribuida, cuya importancia e impacto en la teoría y/o práctica de la computación distribuida haya sido evidente durante al menos una década.
  - ◆ El premio incluye una dotación de \$2000 dólares 



# Teorema de Imposibilidad de Fischer, Lynch, Patterson

*En un sistema completamente asíncrono, es **imposible garantizar** el consenso si incluso un solo proceso es propenso a fallos.*

La razón: no se puede distinguir entre un proceso muy lento y un proceso que se ha bloqueado.

La implicación: en un sistema asíncrono es imposible garantizar la propiedad de ser tolerante a fallos.

# Teorema de Imposibilidad de Fischer, Lynch, Patterson

- ◆ El teorema ofrece una cota superior al momento de establecer algoritmos de consenso tolerantes a fallas.
- ◆ Debemos ser rigurosos con los supuestos que hagamos en algoritmos distribuidos.

# Teorema de Imposibilidad de Fischer, Lynch, Patterson

- ◆ El teorema ofrece una cota superior al momento de establecer algoritmos de consenso tolerantes a fallas.
- ◆ Debemos ser rigurosos con los supuestos que hagamos en algoritmos distribuidos.

## Estrategias para la Tolerancia a Fallas considerando teorema:

1. **Asumir Sincronía Parcial**: incorporar límites de tiempo al sistema asíncrono.
2. **Optar por consenso parcialmente correcto**: se alcanza si se recurre a *Quorum*.
3. **Relajar las garantías**: En lugar de soluciones que "garantizan un consenso en sistemas asíncronos", optar por "se llega a consenso con alta probabilidad".

# RPC en Presencia de Fallos

# RPC en Presencia de Fallos

## Clases de Fallos en RPC:

- ◆ El cliente no puede localizar al RPC por fallo de la red o del mismo RPC.
- ◆ El mensaje de solicitud (*request*) se pierde. El cliente no recibe respuesta.
- ◆ El RPC falla después de recibir una solicitud. El cliente no recibe respuesta, y no puede saber si el RPC ejecutó o no la operación antes de fallar.
- ◆ El mensaje de respuesta (*reply*) se pierde: Similar al caso anterior, el cliente no sabe si la operación se ejecutó.
- ◆ El cliente falla después de enviar una solicitud: La operación puede quedar como un "cálculo huérfano" (*orphan computation*) en el RPC, consumiendo recursos o causando problemas si el cliente se reinicia y vuelve a intentar la operación.

# RPC en Presencia de Fallos

## Semántica de Llamada (*Delivery Guarantees*):

- ◆ **Semántica *Maybe***: El RPC puede ejecutarse una vez o no ejecutarse en absoluto. Ocurre cuando no se aplican medidas de tolerancia a fallos. Útil solo cuando fallos ocasionales son aceptables.
- ◆ **Semántica *At-least-once***: EL RPC se ejecuta al menos una vez, posiblemente más. El cliente retransmite la solicitud hasta recibir una respuesta. Para evitar resultados incorrectos por duplicación, las operaciones deben ser idempotentes.
- ◆ **Semántica *At-most-once***: EL RPC se ejecuta como máximo una vez o ninguna. Implica filtrado de duplicados en el RPC y retransmisiones de resultados almacenados.
- ◆ **Semántica *Exactly-once***: El RPC se ejecuta si o si una vez. Una mezcla de retransmisiones de operaciones por parte del cliente y que el RPC se comporta con semántica *At-most-once*.

# Poniendo a prueba lo que hemos aprendido 🧐

¿Cuáles de las siguientes afirmaciones sobre tolerancia a fallos en sistemas distribuidos es **correcta**?

- a. Un sistema con alta disponibilidad necesariamente tiene alta fiabilidad.
- b. Un sistema tolerante a fallos se asegura que no ocurra ningún fallo.
- c. Se pueden reparar mensajes dañados aplicando correctamente redundancia de información.
- d. Una falla de tiempo se refiere a que el nodo no responderá aunque se le de todo el tiempo posible.
- e. Es requisito primordial que un sistema tolerante a fallo pueda detectar los fallos que ocurren.

# Poniendo a prueba lo que hemos aprendido 🙄

¿Cuáles de las siguientes afirmaciones sobre tolerancia a fallos en sistemas distribuidos es **correcta**?

- a. Un sistema con alta disponibilidad necesariamente tiene alta fiabilidad.
- b. Un sistema tolerante a fallos se asegura que no ocurra ningún fallo.
- c. Se pueden reparar mensajes dañados aplicando correctamente redundancia de información.**
- d. Una falla de tiempo se refiere a que el nodo no responderá aunque se le de todo el tiempo posible.
- e. Es requisito primordial que un sistema tolerante a fallo pueda detectar los fallos que ocurren.



# Próximos eventos

## Próxima clase

- ◆ Terminar lo que faltó de esta clase (en caso de no alcanzar a terminar).
- ◆ Sesión para explicar y resolver dudas de la Tarea 2.
- ◆ Subsiguiente clase: replicación de datos para tolerar fallos

## Evaluación

- ◆ Mañana se publica la tarea 2 que va a simular, en Python, algunos de los algoritmos vistos antes de la semana de receso.

---

# IIC2523

# Sistemas Distribuidos

— Hernán F. Valdivieso López —  
(2025 - 2 / Clase 11)

---