Tema 5 – Gestión de fallos

Tecnologías de los Sistemas de Información en la Red

Índice

1. Introducción
2. Replicación
3. Consistencia
4. Resultados de aprendizaje

Objetivos

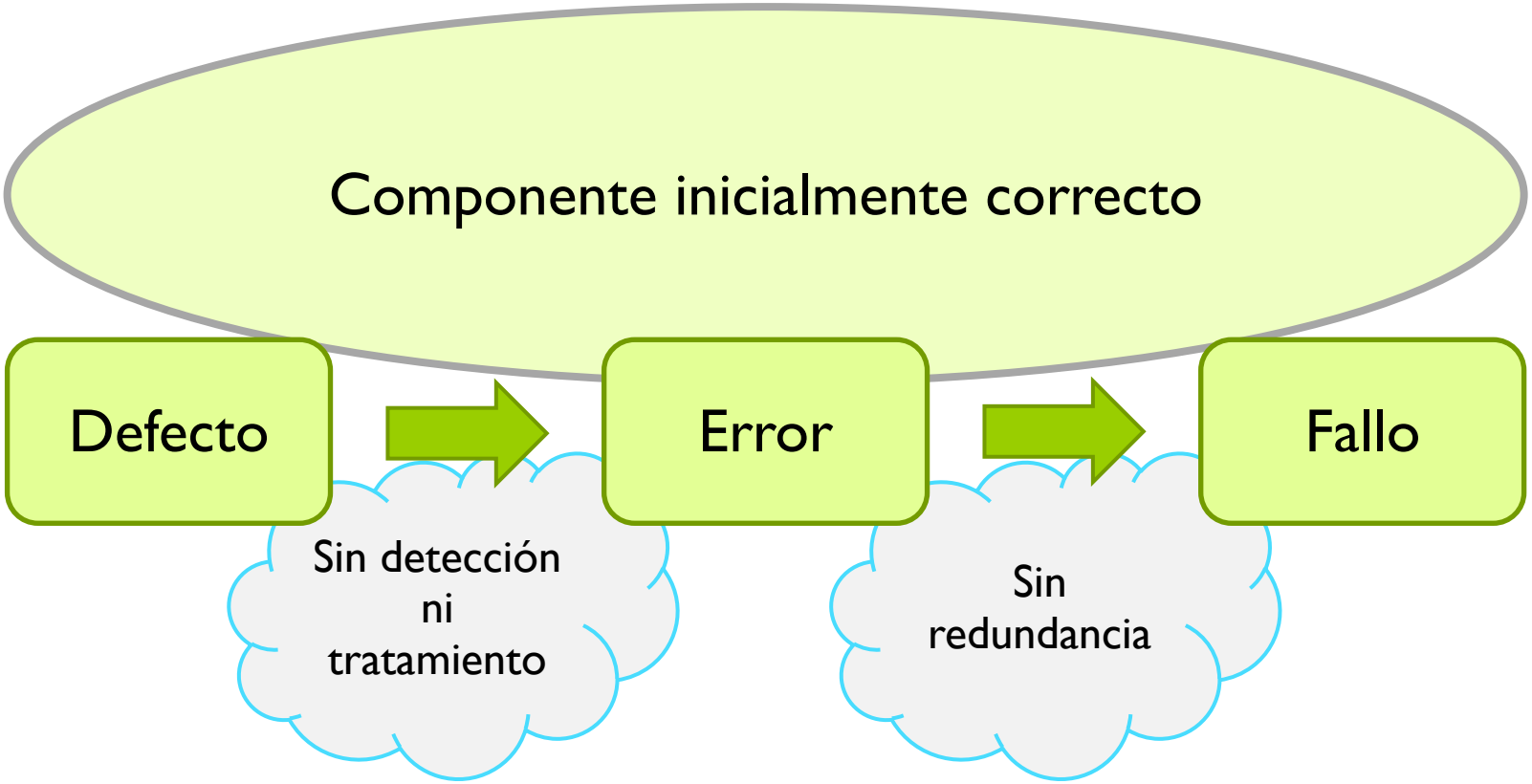
- Caracterizar adecuadamente las situaciones de fallo en un sistema distribuido.
- Estudiar los modelos de replicación.
- Identificar los modelos de consistencia.

Índice

1. Introducción

- Fallo. Definición intuitiva:
  - Se produce un **fallo** cuando algún componente del sistema es incapaz de comportarse de acuerdo con su especificación.
- Definición precisa:
  - Debemos distinguir entre defectos, errores y fallos.
    - **Defecto** ("*fault*"): Condición anómala.
      - Ejemplos: Error de diseño, interferencias, entrada imprevista, corte del suministro eléctrico…
    - **Error**: Manifestación de un defecto en un sistema. El estado de algún componente diferirá de su estado previsto.
    - **Fallo** ("*failure*"): Incapacidad para que un elemento desarrolle aquellas funciones para las que fue diseñado debido a errores en el propio elemento o en su entorno, que han sido causados por diferentes defectos.

1. Introducción



1. Introducción

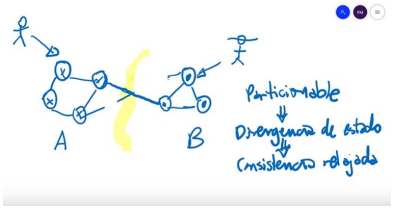
- Todo sistema distribuido debe proporcionar transparencia de fallos.
  - Los errores en algún componente, en caso de producirse, no deben ser percibidos por los usuarios.
    - Solución: Replicación:
      - La réplica con errores se aísla, se repara y se reincorpora.
      - Las demás ocultan esa situación.
- También se denomina "*fault tolerance*":
  - Un **sistema** tolera defectos cuando exhibe un comportamiento correcto en caso de que haya defectos.
    - Un **servicio** tolera defectos si se diseña adecuadamente y, además, también toleran defectos todos aquellos servicios de los que dependa.

1. Introducción

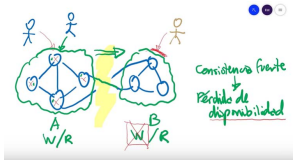
- Los fallos pueden tener múltiples causas.
  - Depende de los defectos que haya y a qué componentes afecten.
  - Se pueden distinguir múltiples tipos de fallos.
- Cuando se diseñan algoritmos distribuidos conviene asumir algún modelo de fallos.
  - Ningún modelo puede reflejar con precisión todas las situaciones de fallo.
  - Mayor abstracción.
    - No se consideran los detalles concretos.
  - El middleware tendrá que "aproximar" esas situaciones a lo que asuma el modelo.

1. Introducción

- Fallos de la red
  - Importa la conectividad.
    - Problema de **particionado**.
  - *Partición*: Cuando un grupo de nodos queda aislado del resto del sistema.
  - Opciones:



1. Sistema particionable:
   - Cada uno de los grupos aislados puede continuar con su trabajo.
   - Se necesitará algún protocolo de reconciliación cuando se recupere la conectividad.
2. Modelo de partición primaria:
   - Sólo se admite que continúe aquel grupo que tenga una mayoría de nodos.
   - No siempre existirá tal grupo.



Índice

1. Fallos: Concepto y tipos
2. Replicación
3. Consistencia
4. Resultados de aprendizaje

2. Replicación

- La replicación es un mecanismo básico para asegurar la disponibilidad de un componente.
  - Cada réplica del componente se ubica en una máquina distinta.
  - Las máquinas utilizadas no deben depender de una misma fuente de fallos.
    - Red eléctrica, SAI, red local…
  - Cuando falle una réplica las demás no tienen por qué fallar.
    - Las operaciones en curso en la réplica caída podrán completarse en una de las demás.
  - La replicación también facilita la recuperación tras un fallo.
    - Las réplicas activas se toman como fuente para reconfigurar la réplica caída tras su reparación.

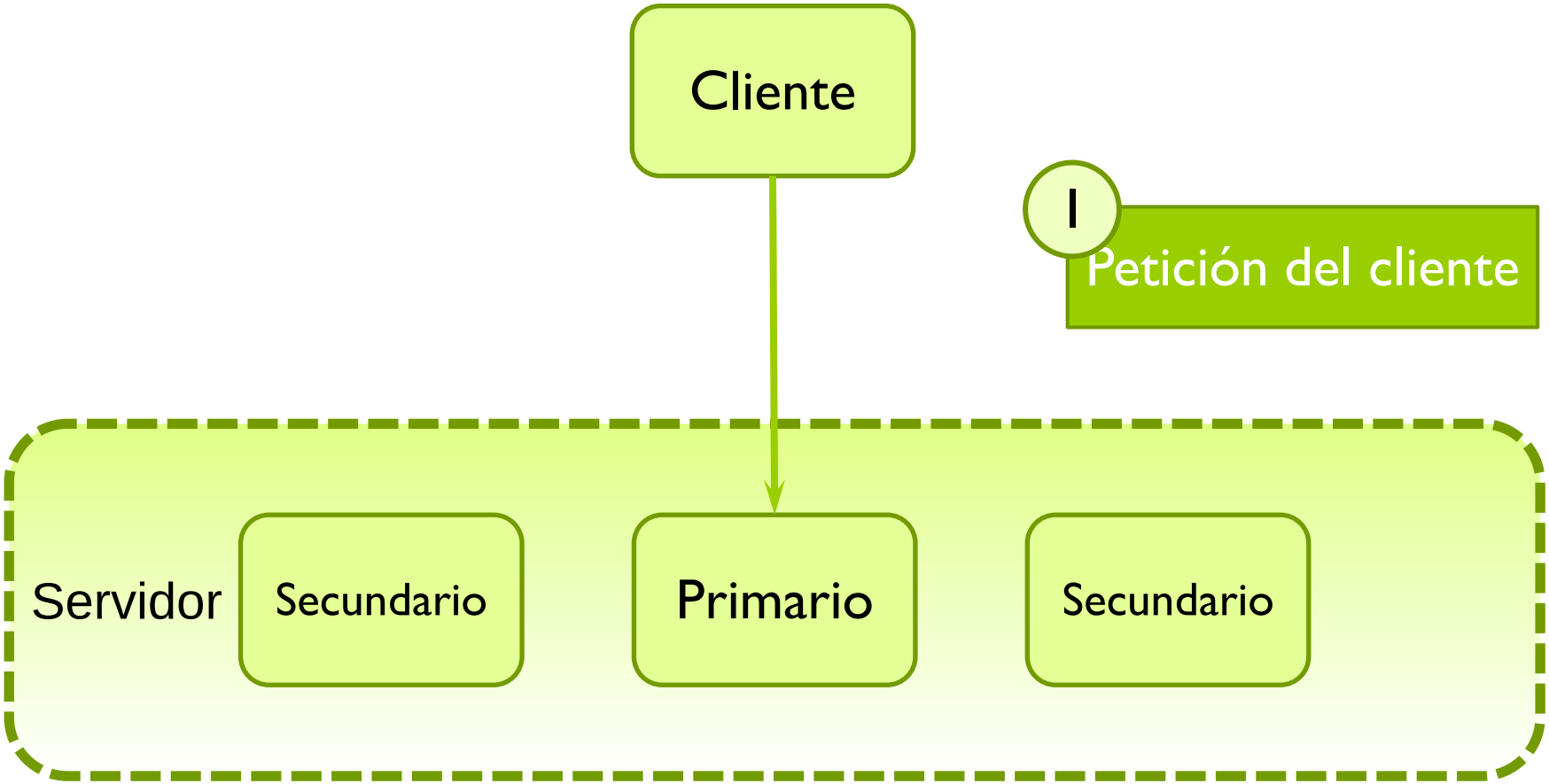
2. Replicación

- La replicación también mejora la rendimiento:
  - Las operaciones de sólo lectura pueden ser ejecutadas por una sola réplica.
    - Rendimiento lineal respecto al número de réplicas.
    - Excelente escalabilidad.
  - Las operaciones de escritura deben ser aplicadas en todas las réplicas.
    - Esto conlleva retardos.
    - Si la operación es breve, puede ser ejecutada por todas las réplicas.
      - Habrá que propagar la petición a todas.
    - Si su ejecución es costosa pero modifica unos pocos datos interesa:
      - Ejecutarla en una sola réplica.
      - Propagar las modificaciones posteriormente al resto de réplicas.
    - Según cómo se intercalen o propaguen las operaciones en cada réplica podrá haber diferencias en sus estados.
      - El grado de divergencia determina el modelo de consistencia.

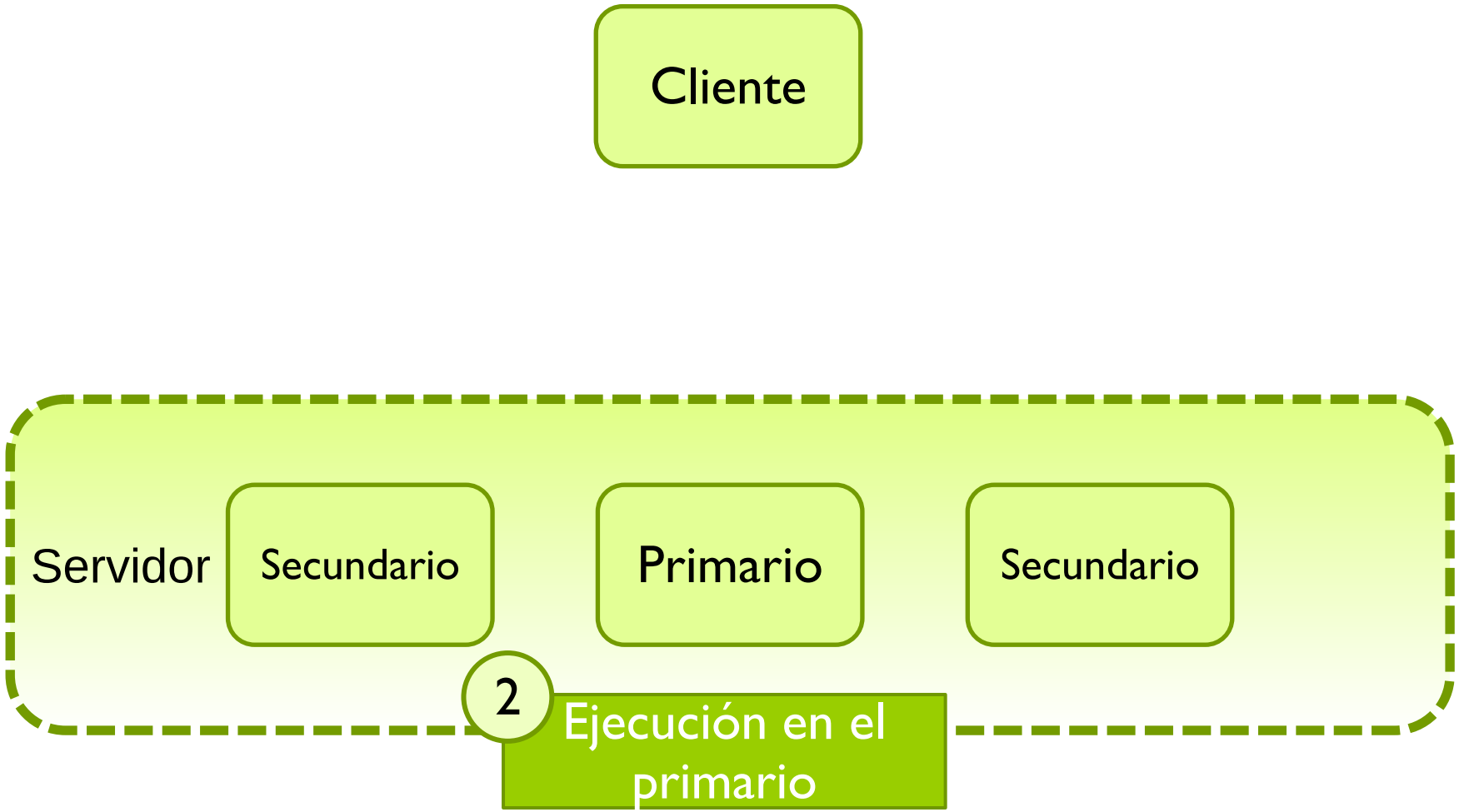
2. Replicación

- Dos modelos de replicación clásicos:
  1. Pasivo
     - El cliente envía su petición a la réplica primaria.
       - Una misma réplica primaria para todos los clientes y todas las peticiones.
     - Esta réplica ejecuta la operación.
     - Al terminar:
       1. Propaga las modificaciones a las réplicas secundarias.
       2. Responde al cliente.
  2. Activo (o "máquina de estados")
     - El cliente propaga su petición a todas las réplicas servidoras.
     - Cada réplica servidora ejecuta directamente la operación.
     - Cuando una réplica termina, responde al cliente.

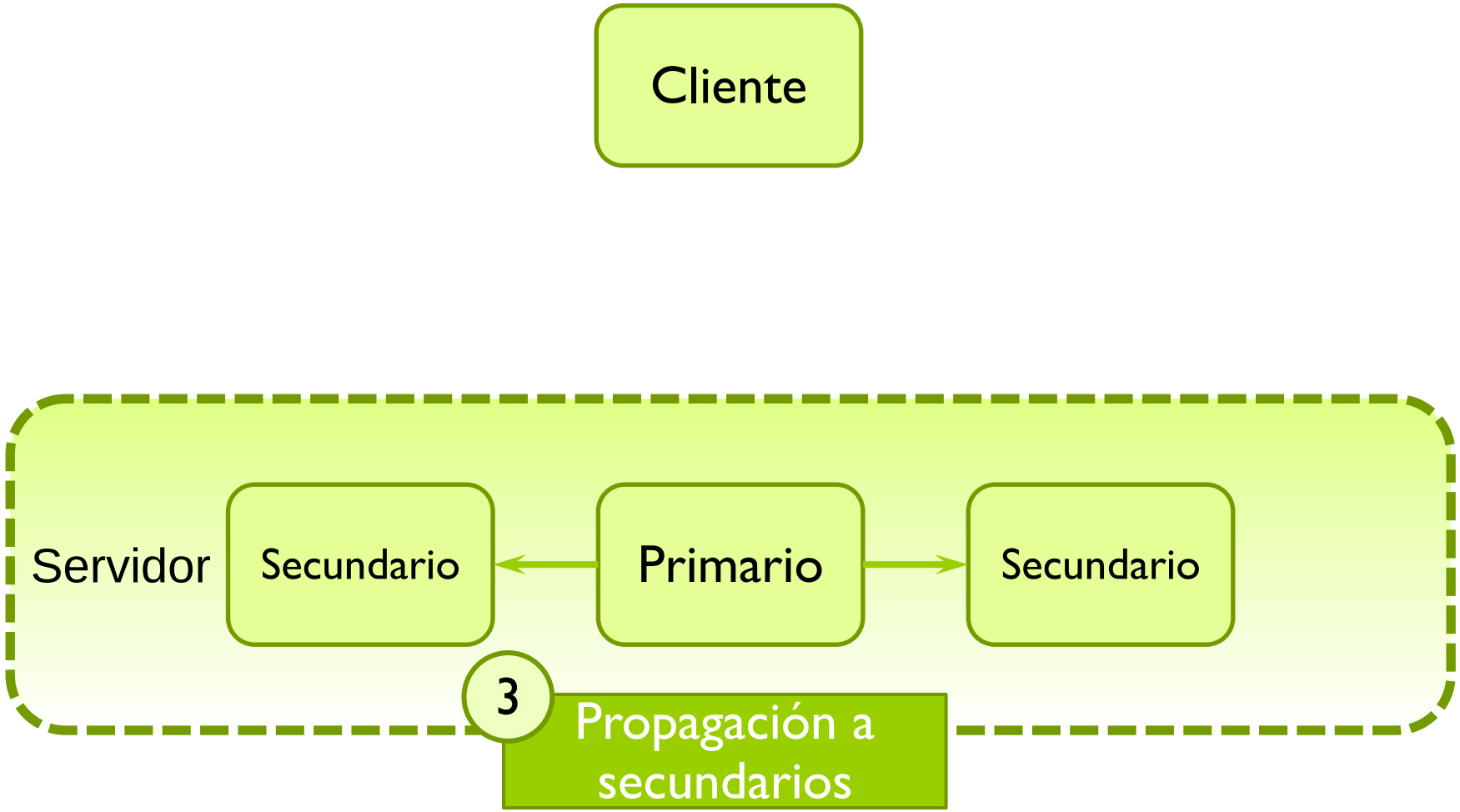
2.1. Replicación pasiva



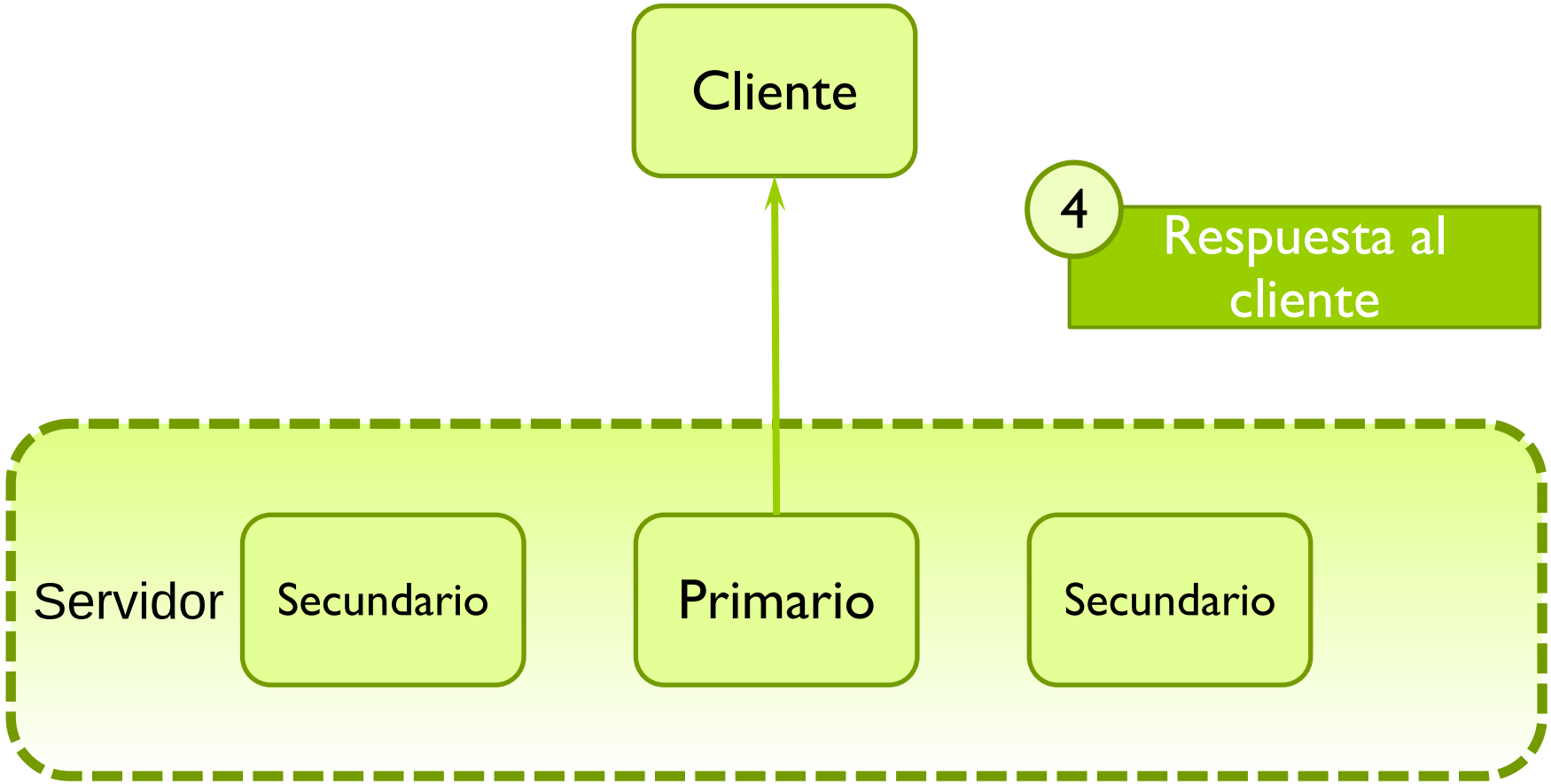
2.1. Replicación pasiva



2.1. Replicación pasiva



2.1. Replicación pasiva

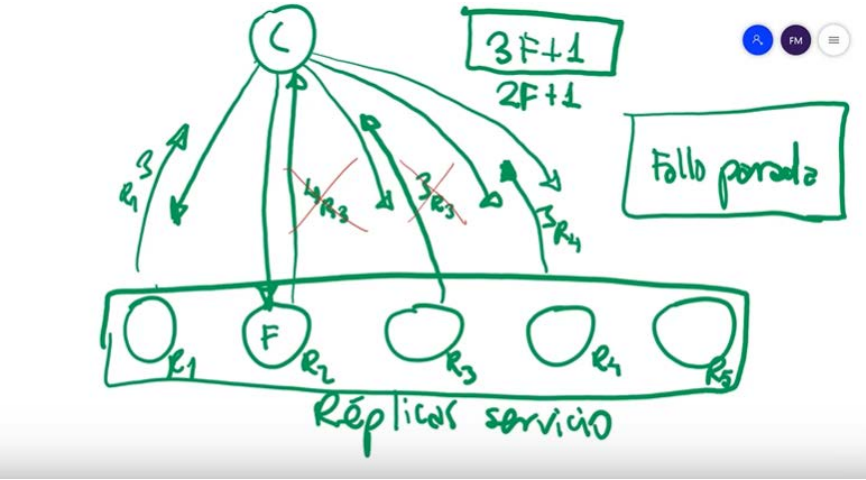


2.1. Replicación pasiva

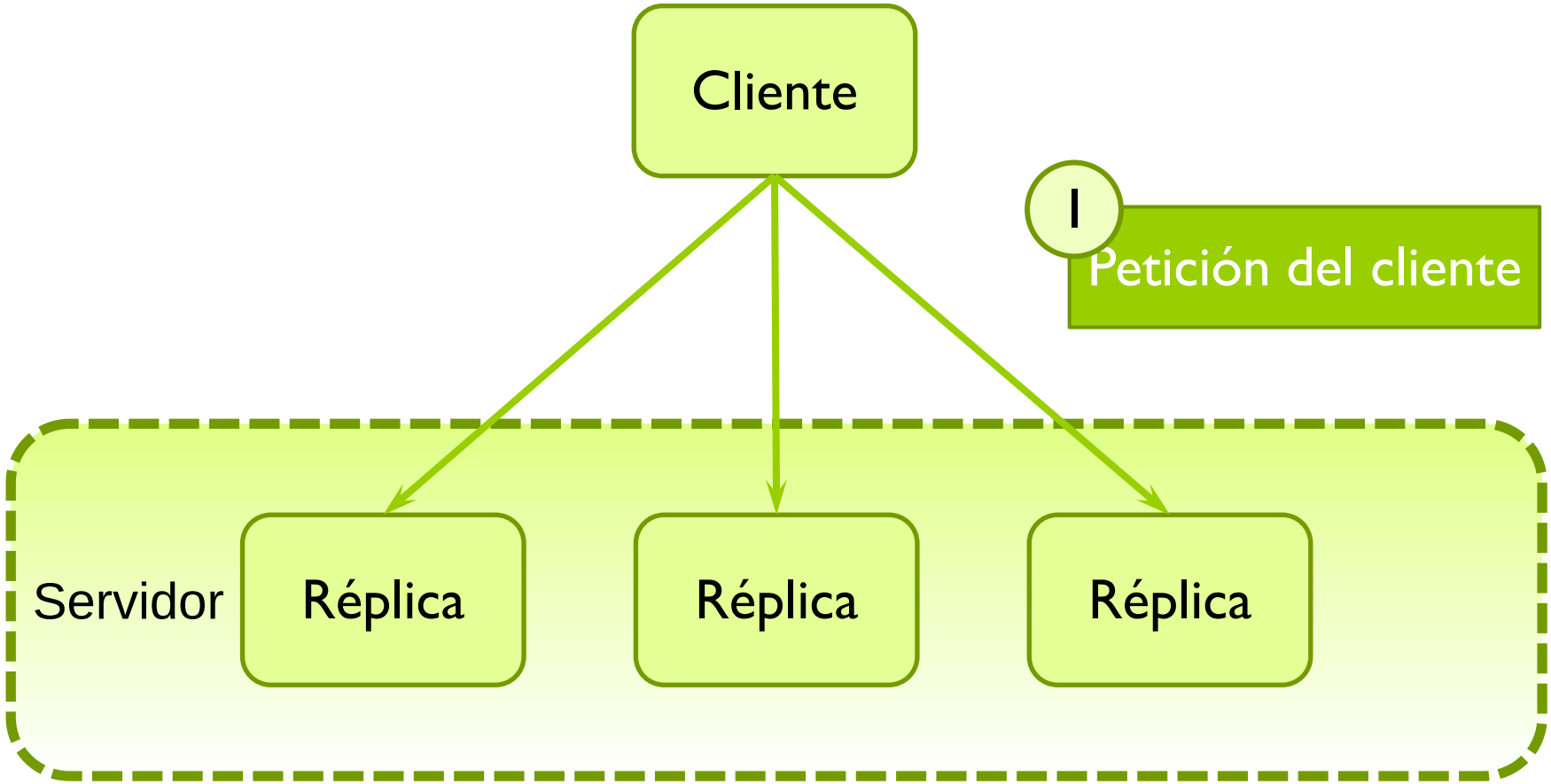
- Ventajas
  - **Mínima carga**.
    - Solo una réplica procesa cada petición.
    - Las peticiones de solo lectura pueden ser atendidas por los secundarios.
      - Así se reparte mejor la carga.
  - **Orden** fácil de establecer.
    - Basta con numerar los mensajes difundidos por el primario.
  - Control de **concurrencia local**.
    - No se necesitan algoritmos distribuidos para ello.
  - Admite **operaciones no deterministas**.
    - Solo las ejecuta el primario.
    - No pueden generar inconsistencias.

2.1. Replicación pasiva

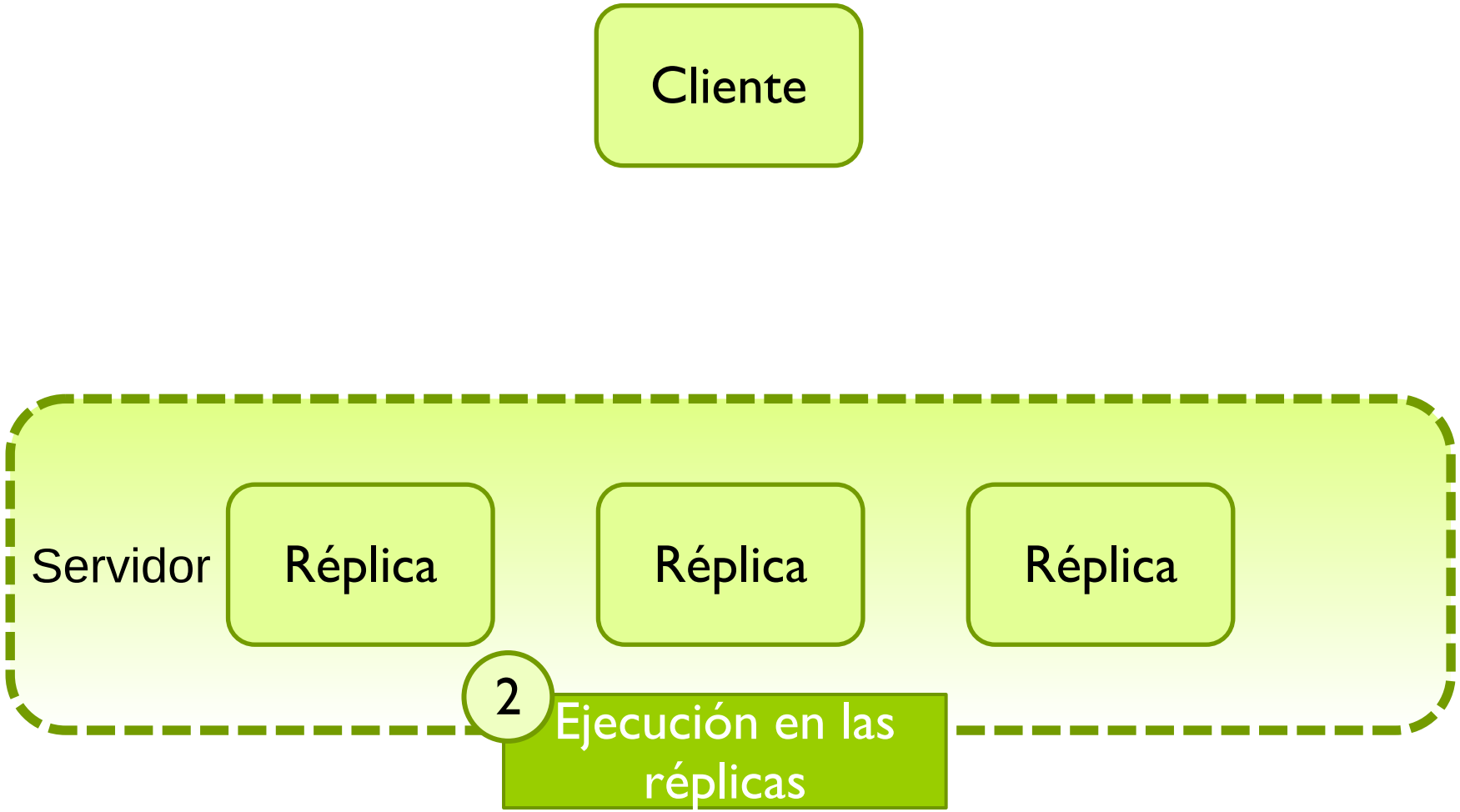
- Inconvenientes
  - **Reconfiguración pesada** cuando falle el primario.
    - Hay que seleccionar un secundario y promoverlo a primario.
    - Pueden llegar a perderse peticiones en curso.
  - **No** se **soporta** el modelo de **fallos bizantinos**.



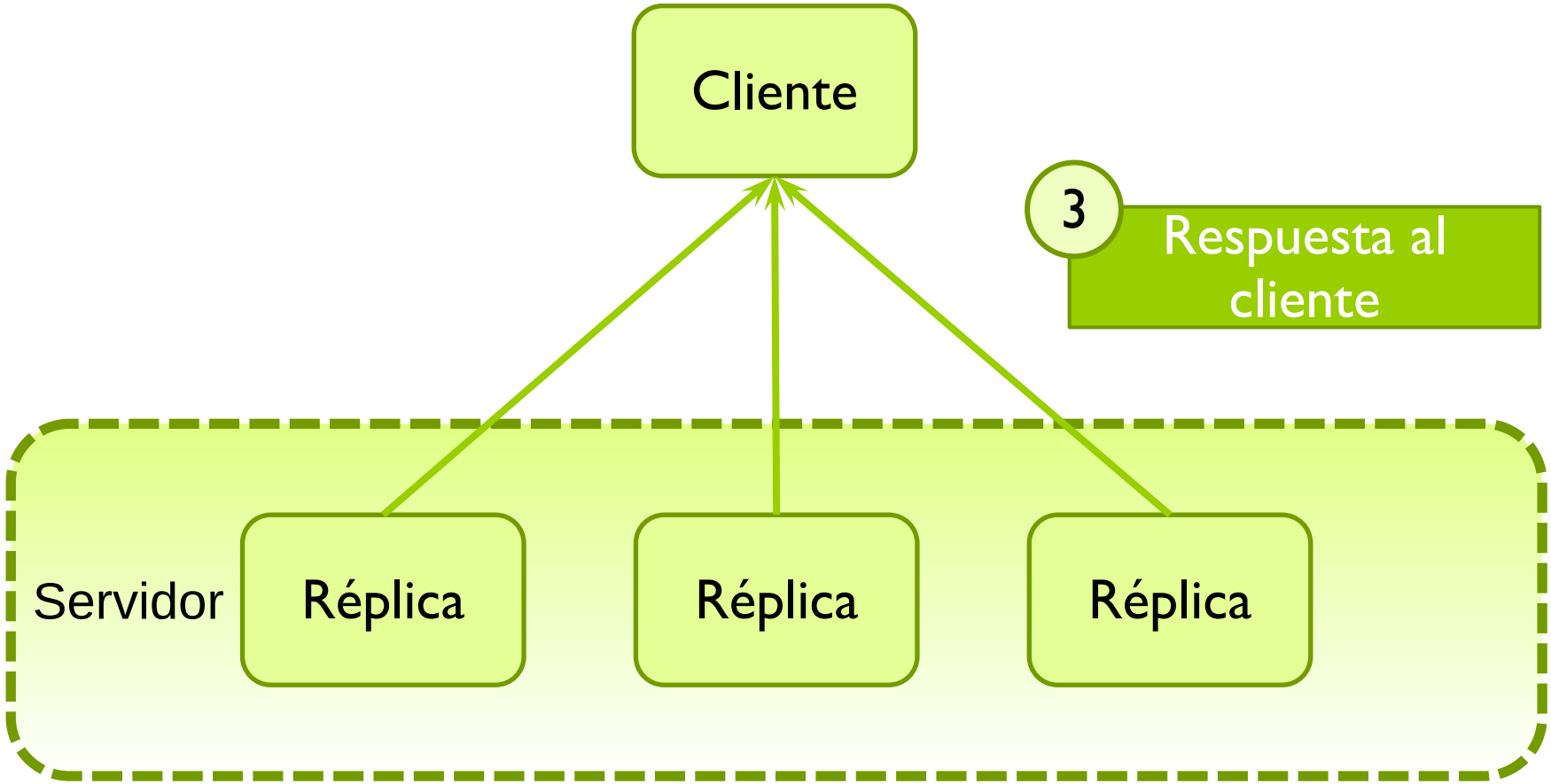
2.2. Replicación activa



2.2. Replicación activa



2.2. Replicación activa



2.2. Replicación activa

- Ventajas
  - **Reconfiguración trivial** en caso de fallo.
    - No se necesita realizar nada especial.
      - Siempre y cuando quede alguna réplica ;-)
  - Se **soporta** el modelo de **fallos bizantinos**.
    - Asumiendo "f" fallos simultáneos, en el caso más favorable, se necesitan 2f + 1 réplicas y, en el caso más habitual, se necesitarían 3F + 1 réplicas.
    - El cliente seleccionará la respuesta mayoritaria.

2.2. Replicación activa

- Inconvenientes
  - Si se necesita **consistencia fuerte**, las peticiones deben difundirse a todas las réplicas en **orden total**.
    - Eso requiere consenso: protocolo pesado.
  - No se toleran las **operaciones** no **deterministas**.
    - Cada réplica podría obtener un resultado diferente.
    - Generaría inconsistencias.
  - Cuando interactúen servicios replicados bajo este modelo hay que **filtrar** las **peticiones**.
    - Ejemplo: El servicio A invoca al servicio B. Ambos utilizan replicación activa.
      - B tendrá que filtrar las múltiples peticiones recibidas.
        - Para evitar que B ejecute N veces la misma operación.
      - Habrá que propagar las respuestas a todas las réplicas de A.

2.3.1. Comparativa. Aspectos generales

Aspecto	Modelo pasivo	Modelo activo
Réplicas procesadoras	**1**	Todas
Evita la ordenación distribuida de las peticiones	**Sí**	No
Evita propagación de modificaciones	No	**Sí**
Admite indeterminismo	**Sí**	No
Tolera fallos arbitrarios	No	**Sí** (Para ello, necesita que todas las réplicas procesen también las lecturas)
Consistencia	Al menos, secuencial	Al menos, secuencial
Recuperación en caso de fallo	Necesita etapas de elección y reconfiguración cuando falla el primario	**Inmediata**

2.3.2. Comparativa. Caso 1. Escenario 1

- Supongamos el despliegue de un servicio S1 con estas características:
  - 5 réplicas
  - Ancho de banda de la red: 1 Gbps
  - Tiempo de propagación de los mensajes: 2 ms
  - Tiempo medio de respuesta para las operaciones: 200 ms
    - No asume concurrencia
    - Cuando se procesan N peticiones concurrentemente por los hilos de un mismo proceso, entonces el tiempo de respuesta pasa a ser 200*N
  - Tamaño medio de las modificaciones por operación: 10 KB
    - i.e., 80 Kb → 0.08 Mb → 0.00008 Gb
      - Con ello, el tiempo de transferencia de las modificaciones pasa a ser:
        - propagación + transmisión (ancho de banda)
          - 0.002 + 0.00008 s → 2.08 ms
  - Tiempo de aplicación de modificaciones en las réplicas secundarias: 3 ms
  - ¿Qué modelo de replicación tendrá menor tiempo de respuesta?
    - Calculémoslo en la hoja siguiente...

2.3.2. Comparativa. Caso 1. Escenario 1

Aspecto	Modelo pasivo		Modelo activo
	Réplica primaria	Réplica secundaria	
Entrega de la petición	2 ms	--	4 - 6 ms (con un algoritmo de difusión de orden total basado en secuenciador)
Procesamiento de la petición	200 ms	--	200 ms
Transferencia de modificaciones	2.08 ms		0 ms
Aplicación de modificaciones	--	3 ms	0 ms
Respuesta a modificaciones	2 ms		0 ms
Respuesta al cliente	2 ms	--	2 ms
TOTAL:	211.08 ms		206 – 208 ms

2.3.2. Comparativa. Caso 1. Escenario 2

- Discusión:
  - A primera vista, los resultados son similares en ambos modelos.
  - Sin embargo, podemos considerar que...
    - Cada réplica ha sido desplegada en un ordenador diferente
    - Si necesitáramos desplegar cinco servicios similares a S1 en esos ordenadores:
      - Con replicación pasiva, cada servicio desplegaría su réplica primaria en un ordenador diferente
      - En ese escenario, los resultados obtenidos se muestran en la hoja siguiente
        - El modelo pasivo ofrece mejores resultados

2.3.2. Comparativa. Caso 1. Escenario 2

Aspecto	Modelo pasivo		Modelo activo
	Réplica primaria	Réplica secundaria	
Entrega de la petición	2 ms	--	4 - 6 ms (con un algoritmo de difusión de orden total basado en secuenciador)
Procesamiento de la petición	200+4*3=212ms	--	200*5=1000 ms
Transferencia de modificaciones	2.08 ms		0 ms
Aplicación de modificaciones	--	3 ms	0 ms
Respuesta a modificaciones	2 ms		0 ms
Respuesta al cliente	2 ms	--	2 ms
TOTAL:	223.08 ms		1006 – 1008 ms

2.3.3. Comparativa. Caso 2. Escenario 2

- Supongamos ahora el despliegue de un servicio S2 con estas características:
  - 5 réplicas
  - Ancho de banda de la red: 1 Gbps
  - Tiempo de propagación de los mensajes: 2 ms
  - Tiempo medio de respuesta para las operaciones: 5 ms
    - No asume concurrencia
    - Cuando se procesan N peticiones concurrentemente por los hilos de un mismo proceso, entonces el tiempo de respuesta pasa a ser 5*N
  - Tamaño medio de las modificaciones por operación: 10 MB
    - i.e., 80 Mb → 0.08 Gb
      - Con ello, el tiempo de transferencia de las modificaciones pasa a ser:
        - propagación + transmisión (ancho de banda)
          - 0.002 + 0.08 s → 82 ms
  - Tiempo de aplicación de modificaciones en las réplicas secundarias: 3 ms
  - ¿Qué modelo de replicación tendrá menor tiempo de respuesta?
    - Calculémoslo en la hoja siguiente...

2.3.3. Comparativa. Caso 2. Escenario 2

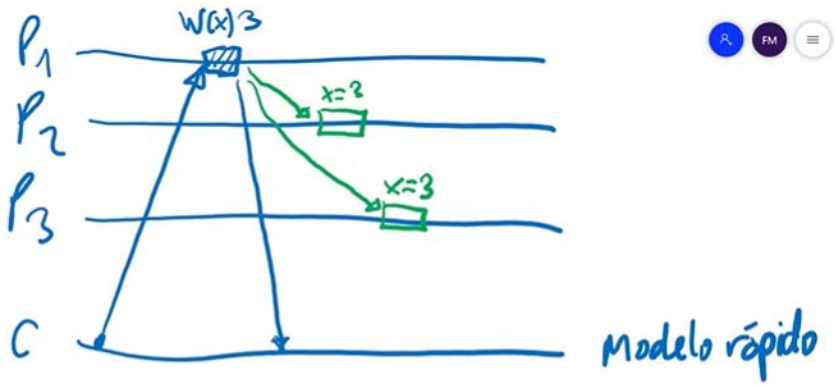
Aspecto	Modelo pasivo		Modelo activo
	Réplica primaria	Réplica secundaria	
Entrega de la petición	2 ms	--	4 - 6 ms (con un algoritmo de difusión de orden total basado en secuenciador)
Procesamiento de la petición	5+4*3=17ms	--	5*5=25 ms
Transferencia de modificaciones	82 ms		0 ms
Aplicación de modificaciones	--	3 ms	0 ms
Respuesta a modificaciones	2 ms		0 ms
Respuesta al cliente	2 ms	--	2 ms
TOTAL:	108 ms		31 – 33 ms

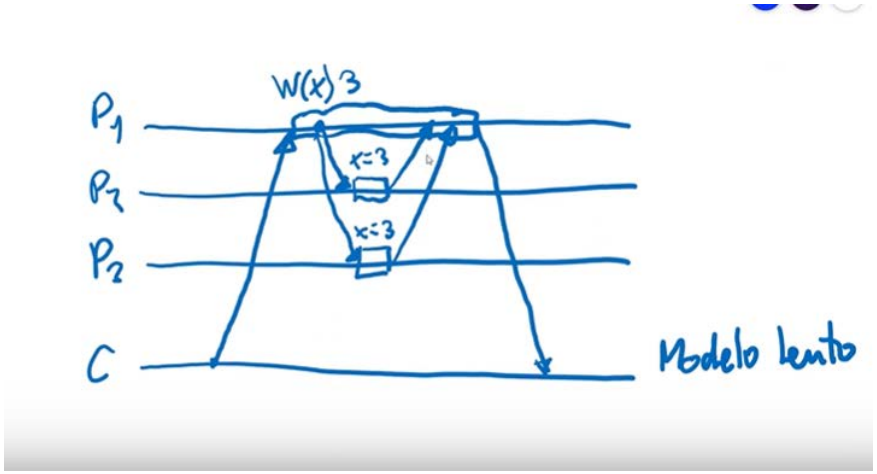
2.3.4. Comparativa. Análisis

- El modelo de replicación elegido depende principalmente de:
  - El tiempo promedio de procesamiento para las peticiones
    - El modelo pasivo es apropiado para procesamiento prolongado
      - Porque esa carga solo afecta a la réplica primaria
    - El modelo activo es apropiado para procesamiento breve
  - El tamaño de las modificaciones
    - El modelo pasivo es apropiado para modificaciones pequeñas
    - El modelo activo es apropiado para modificaciones grandes
      - Porque no necesita transferirlas a otras réplicas

Índice

1. Fallos: Concepto y tipos
2. Replicación
3. Consistencia
4. Resultados de aprendizaje



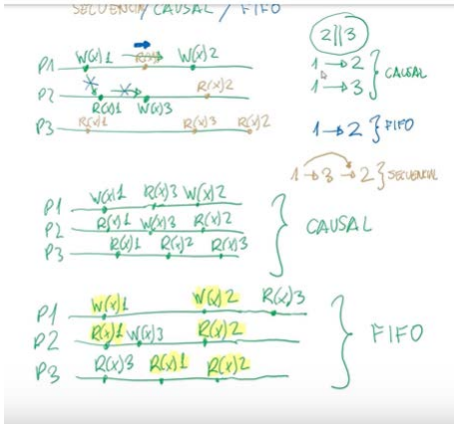


3. Consistencia

- Consistencia centrada en datos
  - Cuando se replica información en múltiples nodos, un modelo de consistencia especifica qué divergencias se admiten entre los valores de las réplicas de un mismo elemento.
  - Los clientes realizan la escritura inicialmente en un nodo.
  - Ese nodo propaga posteriormente el resultado a las demás réplicas.
    - La consistencia obtenida depende del retardo de esa propagación...
      - ... y las esperas que introduzca ese retardo en otros procesos.
    - Para controlar esas acciones de escritura y las acciones de lectura, se utiliza un algoritmo de consistencia.
  - Si el algoritmo de consistencia permite que tanto escrituras como lecturas retornen el control sin esperar a que se haya transmitido algún mensaje...
    - Tendremos un modelo de consistencia rápido.
    - En otro caso, tendremos un modelo de consistencia lento.
  - Cada proceso lee en un solo nodo: lecturas locales.

3. Consistencia

- Modelos de consistencia
  - Si no se asume el uso de herramientas de sincronización, los modelos más importantes son:
    - Estricto (lento)
    - Secuencial (lento)
    - Procesador (lento)
    - Caché (lento)
    - Causal (rápido)
    - FIFO (rápido)
  - Solo estudiaremos algunos de estos modelos.

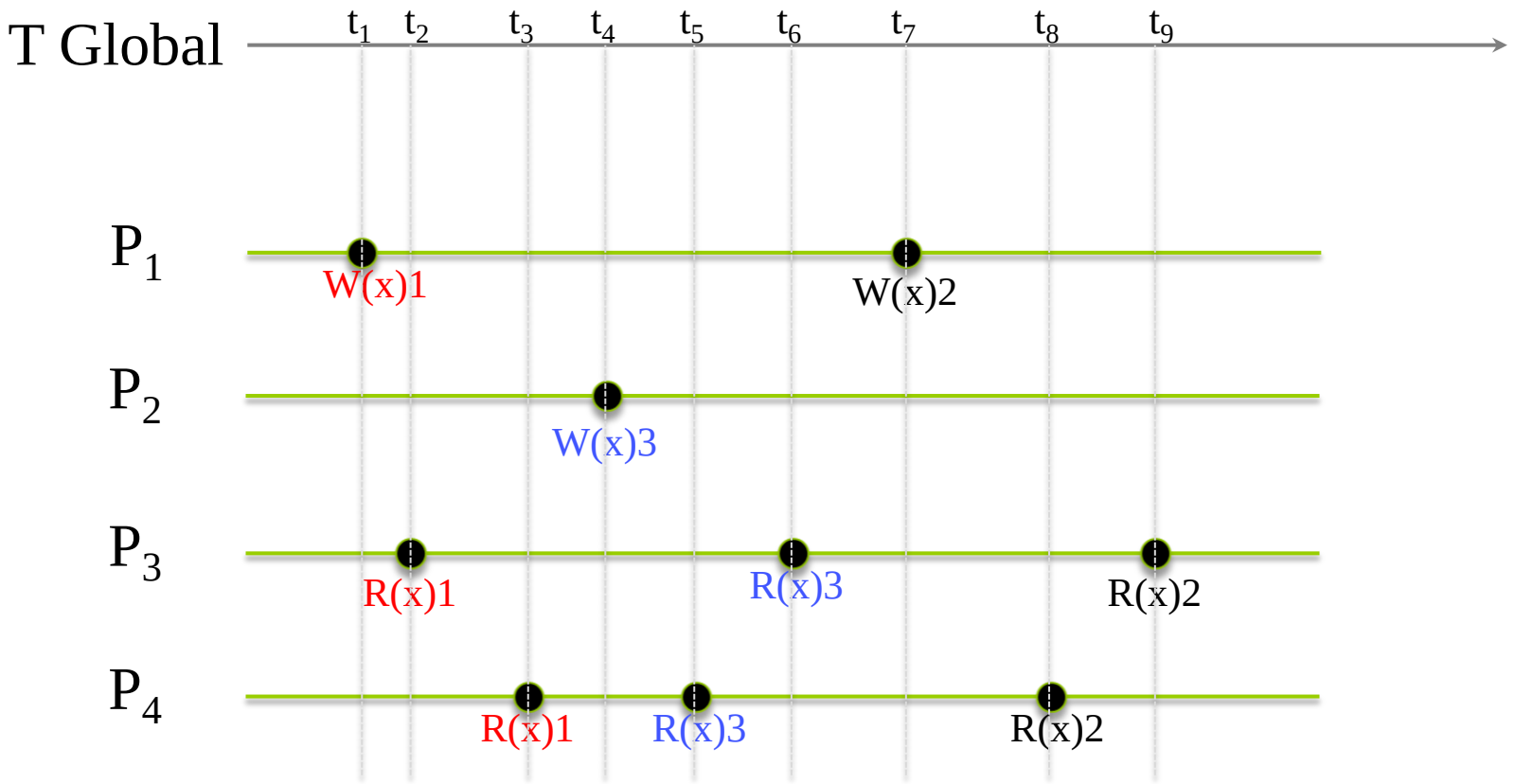


- Consistencia final ("*eventual consistency*")
  - En los entornos escalables solo se exige que las réplicas converjan cuando haya intervalos prolongados sin escrituras.
  - Mientras haya escrituras, cada réplica las admitirá sin preocuparse por lo que hagan las demás.
  - Posteriormente se decide cómo intercalarlas y se acuerda qué valor final adoptar.
    - Sencillo si las operaciones son conmutativas.

3.1. Consistencia estricta

- Noción intuitiva de cómo trabaja una consistencia fuerte
- Asume un reloj global, capaz de etiquetar cada evento.
  - No pueden suceder dos escrituras a la vez en todo el sistema.
  - Asume propagación inmediata de las escrituras.
  - Cada lectura siempre devuelve el valor de la última escritura realizada sobre esa variable.
- Imposible de implantar.

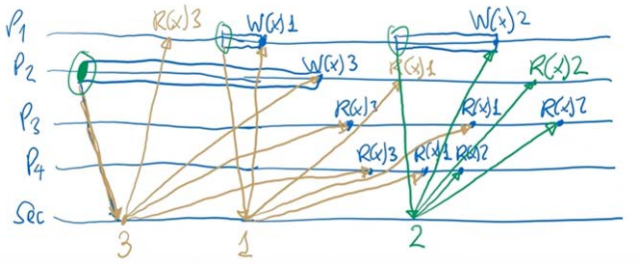
3.1. Consistencia estricta. Ejemplo



- Problema
  - No tiene sentido hablar de tiempo global en un sistema distribuido

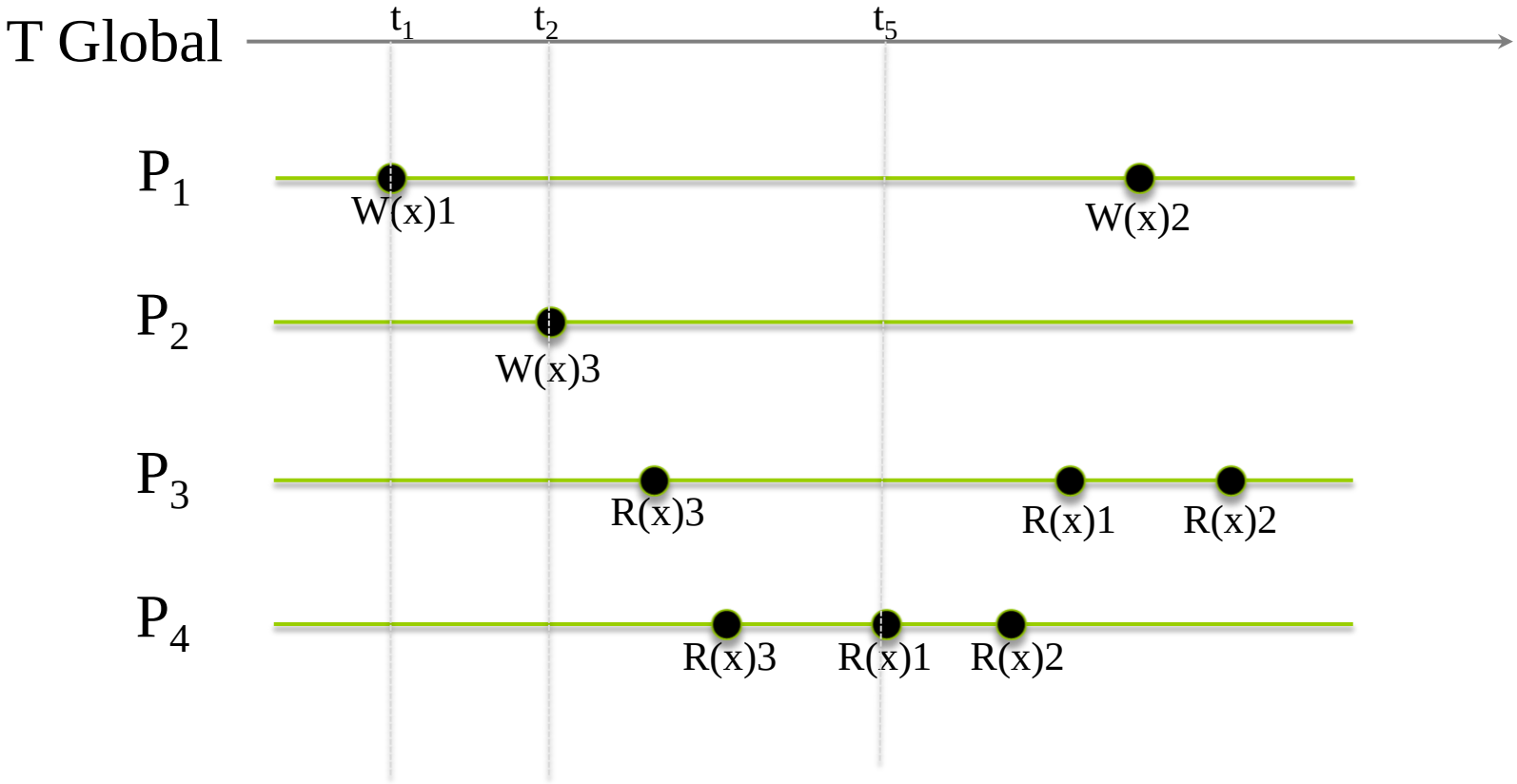
3.2. Consistencia secuencial

- Especificación informal:
  - "El resultado de una ejecución es el mismo que si las operaciones de todos los procesos se ejecutaran en algún orden secuencial, y las operaciones de cada proceso aparecieran en dicha secuencia en el orden que dicta su respectivo programa".
    - Todos los procesos llegan a un acuerdo sobre el orden en que se han llegado a aplicar las escrituras sobre todas las variables.
      - Además, ese orden cuadra con el utilizado al escribir en cada proceso.
    - Pero cada proceso puede avanzar "a su ritmo".



Todos los procesos ven lo mismo. En este caso, se ha decidido ver 3-1-2

3.2. Consistencia secuencial. Ejemplo



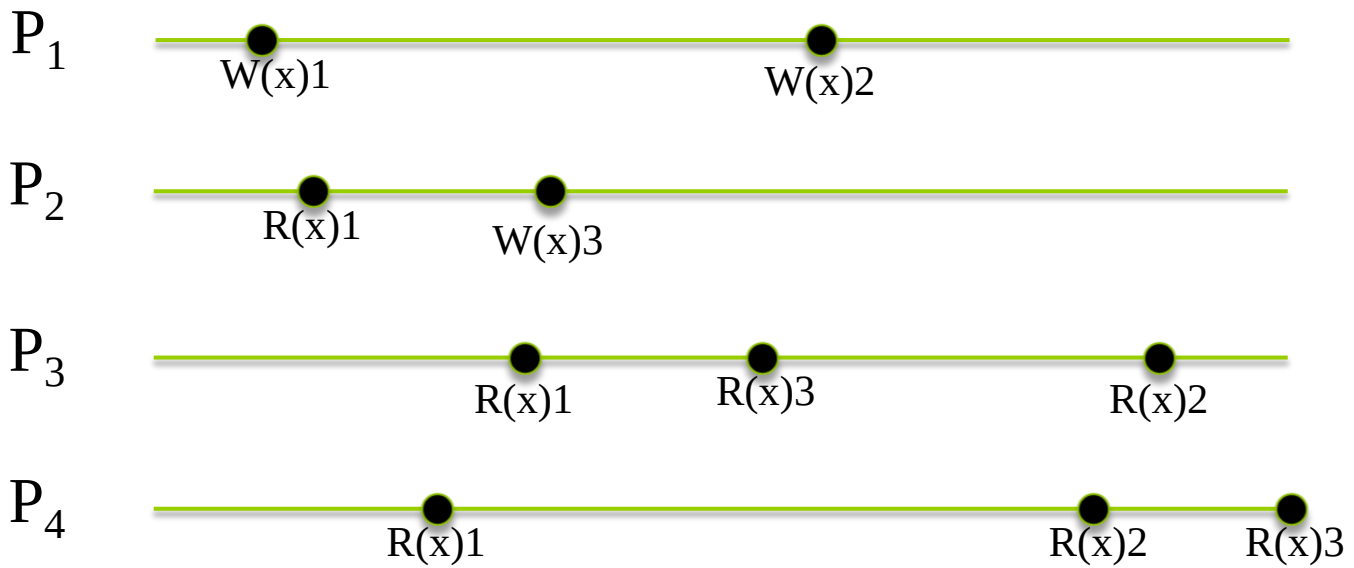
- Evidentemente, esta ejecución no satisface los requisitos de la consistencia estricta asumiendo un reloj global
  - $t_1 < t_2$ pero en t_5 se obtiene R(x)1 en P4, en lugar de R(x)3, como se esperaba, contradiciendo las condiciones de la consistencia estricta

3.3. Consistencia causal

- Se respeta la relación de orden "*happens before*" definida por Lamport
  - Utilizada para definir los relojes lógicos
  - En la consistencia causal se cumple
    - W(x)a → R(x)a.

3.3. Consistencia causal

1 va antes que 2 y
1 va antes que 3 (se escribe y lee antes)
EN TODOS



3 y 2
son
concurrentes

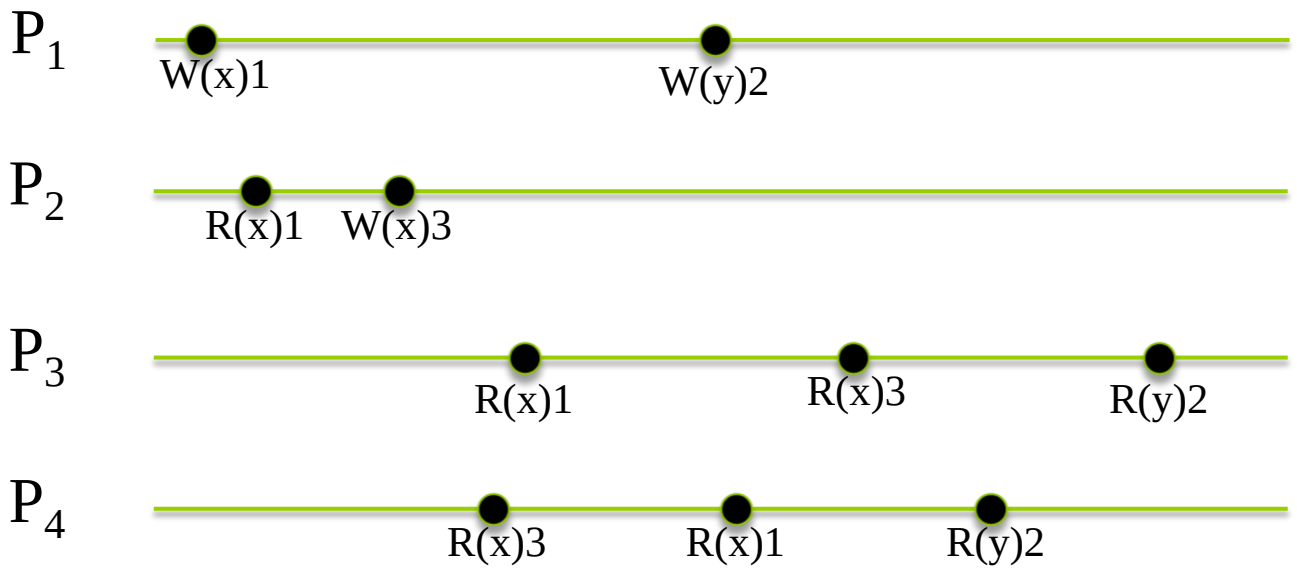
- La escritura del valor 1 precede causalmente a la escritura del valor 2.
- La escritura del valor 1 precede causalmente a la escritura del valor 3.
- Las escrituras de los valores 2 y 3 son concurrentes.
  - Los procesos P3 y P4 pueden ordenarlas como quieran.

3.4. Consistencia FIFO

- Requiere que las escrituras realizadas por un proceso sean leídas en orden de escritura por todos los demás procesos.
- Pero no impone ninguna restricción a la hora de "mezclar" lo que han hecho diferentes escritores.

3.4. Consistencia FIFO

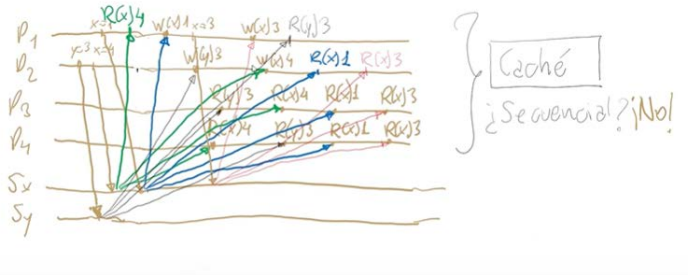
escrituras de un proceso leídas en orden (1 va antes que 2 en TODOS)



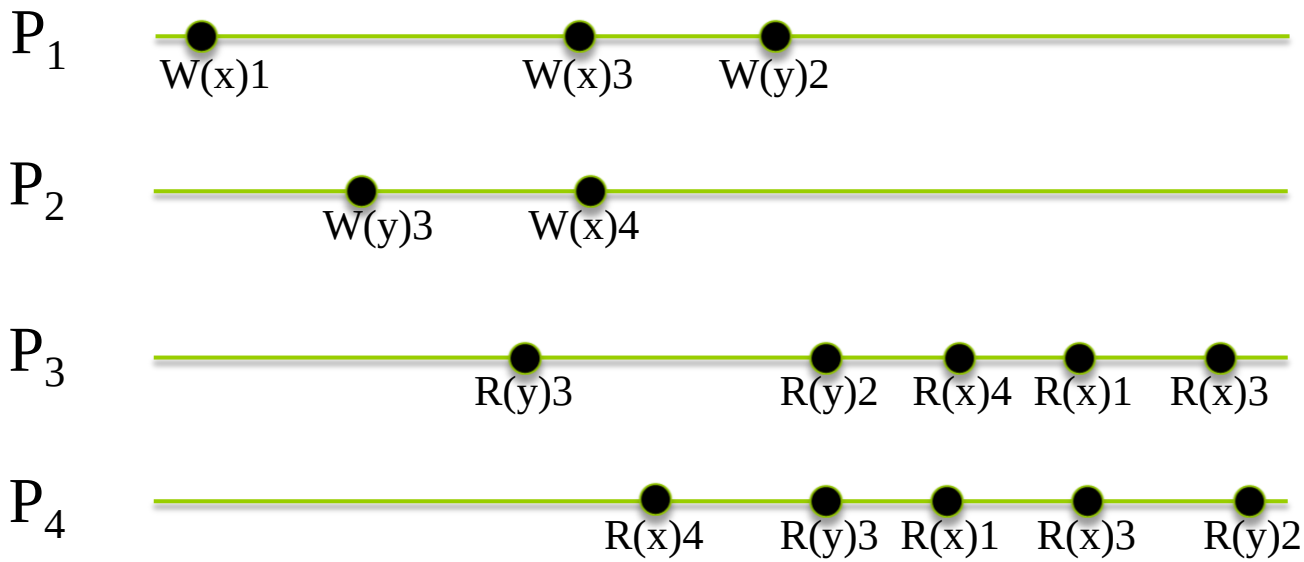
- En este ejemplo, la única restricción a respetar es que todos los lectores obtienen antes el valor 1 que el 2.
  - Pues ambos valores fueron escritos por P1.
  - Pero el valor 3 puede obtenerse en cualquier punto de la secuencia.

3.5. Consistencia caché

- Requiere que las escrituras realizadas sobre una misma variable sean vistas en el mismo orden por todos los procesos.
  - Además, si múltiples escrituras sobre una misma variable han sido realizadas por un mismo proceso, estas escrituras están ordenadas tal como las ejecutó el proceso.
- Pero no impone ninguna restricción a la hora de "mezclar" lo que se haya hecho sobre diferentes variables.



3.5. Consistencia caché

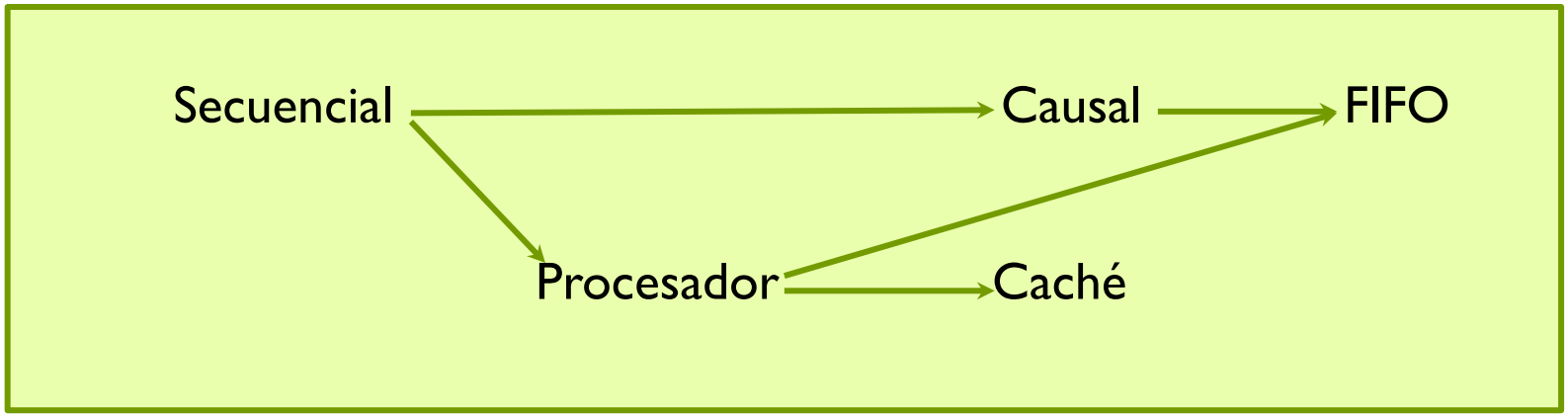


- En este ejemplo se ha decidido que:
  - El orden de lectura de las escrituras sobre "x" sea 4, 1, 3.
    - El valor 1 debe estar antes que el 3 puesto que ambos fueron escritos por P1.
  - El orden de lectura de las escrituras sobre "y" sea 3, 2.
- Cómo se intercalen las lecturas de "x" e "y" no es relevante.

- Tenemos **consistencia "procesador"** cuando se cumplen simultáneamente los modelos caché y FIFO.

3.6. Consistencia

- Jerarquía de modelos



  - Las flechas indican que el modelo origen es más estricto que el modelo destino.
    - Si se cumple el modelo origen, también se cumple el modelo destino.
  - Algunos modelos no son comparables.
    - Causal y procesador.
    - FIFO y caché.
    - Causal y caché.

Índice

1. Fallos: Concepto y tipos
2. Replicación
3. Consistencia
4. Resultados de aprendizaje

4. Resultados de aprendizaje

- Al finalizar este tema, el alumno debe ser capaz de:
  - Distinguir entre defectos, errores y fallos.
  - Identificar a la replicación como el mecanismo a adoptar para superar las situaciones de fallo y mejorar la escalabilidad de un sistema.
  - Conocer los modelos de replicación clásicos y seleccionar el adecuado para cada tipo de componente en una aplicación escalable.
  - Conocer los diferentes modelos de consistencia e intuir su influencia en la escalabilidad de un sistema.