Tema 9
Tolerancia a fallos

Sistemas Distribuidos
Grado en Ingeniería Informática
Universidad Carlos III de Madrid

Contenidos

- Introducción a la tolerancia a fallos
- Tolerancia a fallos software
- Tolerancia a fallos en sistemas distribuidos

Tolerancia a fallos

- Sistema tolerante a fallos
  - Sistema que posee la capacidad interna para asegurar la ejecución correcta y continuada de un sistema a pesar de la presencia de fallos HW o SW
- Objetivo
  - Conseguir que un sistema sea altamente fiable

Origen de los fallos

- Fallos hardware
  - Fallos permanentes o transitorios en los componentes hardware.
  - Fallos permanentes o transitorios en los subsistemas de comunicación.
- Fallos software
  - Especificación inadecuada.
  - Fallos introducidos por errores en el diseño y programación de componentes software.

Conceptos básicos

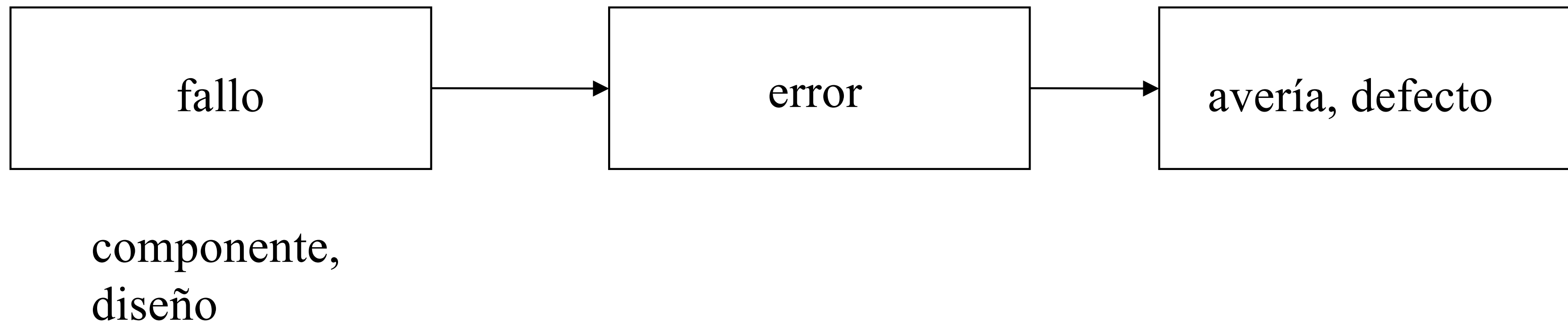
- La **fiabilidad** (*reliability*) de un sistema es una medida de su conformidad con una especificación autorizada de su comportamiento.

- Un sistema es ***fiable*** si cumple sus especificaciones.

Conceptos básicos

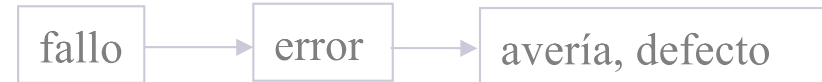
- Una **averia o defecto** (*failure*) es una desviación del comportamiento de un sistema respecto de su especificación.
- Las averías se manifiestan en el comportamiento externo del sistema, pero son el resultado de **errores** (*errors*) internos.
- Las causas mecánicas o algorítmicas de los errores se denominan **fallos** (*faults*).
  - Los fallos pueden ser consecuencia de averías en los componentes del sistema.

Relaciones causa-efecto



- Los fallos pueden ser pequeños, pero los defectos muy grandes
  - Un simple bit puede convertir el saldo de una cuenta bancaria de positivo a negativo

Ejemplo



- Fichero corrupto almacenado en el disco.
- Consecuencia: avería en el sistema que utiliza el fichero.
- ¿Qué provocó el fallo?
  - Error en el programa que escribió el fichero (fallo de diseño).
  - Problema en la cabeza del disco (fallo en el componente).
  - Problema en la transmisión del fichero por la red (fallo HW)
- El error en el sistema podría ser corregido (cambiando el fichero) pero los fallos podrían permanecer.
- Importante distinguir entre fallos y errores.

Ejemplos de fallos

- Explosión del Ariane 5 en 1996
  - Enviado por la ESA en junio de 1996 (fue su primer viaje)
  - Coste del desarrollo: 10 años y 7000 millones de dólares.
  - Explotó 40 seg. después del despegue a 3700 metros de altura.

  - El fallo se debió a la pérdida total de la información de altitud.
  - Causa: error del diseño software.
  - El SW del sistema de referencia inercial realizó la conversión de un valor real en coma flotante de 64 bits a un valor entero de 16 bits. El número a almacenar era mayor de 32767 (el mayor entero con signo de 16 bits) y se produjo un fallo de conversión y una excepción que no se controló

Ejemplos de fallos

- Fallo de los misiles Patriot
  - Misiles utilizados en la guerra del golfo en 1991 para interceptar los misiles iraquíes Scud
  - Fallo en la interceptación debido a errores de precisión en el cálculo del tiempo.
  - El reloj interno del sistema proporciona décimas de segundo que se expresan como un entero
  - Este entero se convierte a un real de 24 bits con la perdida de precisión correspondiente.
  - Esta pérdida de precisión es la que provoca un fallo en la interceptación

Ejemplos de fallos

- Fallo en la sonda Viking enviada a Venus
  En lugar de escribir en Fortran:

  DO 20 I = 1,100

  que es un bucle de 100 iteraciones sobre la etiqueta 20, se escribió:

  DO 20 I = 1.100

  y como los blancos no se tienen en cuenta el compilador

  lo interpretó como:

  DO20I = 1.100

  es decir, la declaración de una variable (O20I) con valor 1.100.

  D indica un identificador real

Ejemplos de fallos

- Fallo del robot Spirit (2003)
  - El Spirit está equipado con un RAD6000 similar a un IBM RS6000 (con protección contra la radiación) equipado con un procesador predecesor de un PowerPC, 128 MB de RAM y 256 MB de memoria flash
  - Utiliza el SOTR VxWorks ubicado en 32 MB de RAM
  - Cada fichero necesita cierta información controlada por el SO.
  - En la flash se acumularon miles de ficheros y las estructuras de datos asociadas desbordaron los 32 MB asignados al SO reiniciándose.
  - Cada vez que se reiniciaba se desbordaba la memoria y se volvía a reiniciar dando la sensación de que estaba parado.

Más ejemplos de fallos…

- Historias sobre fallos en:

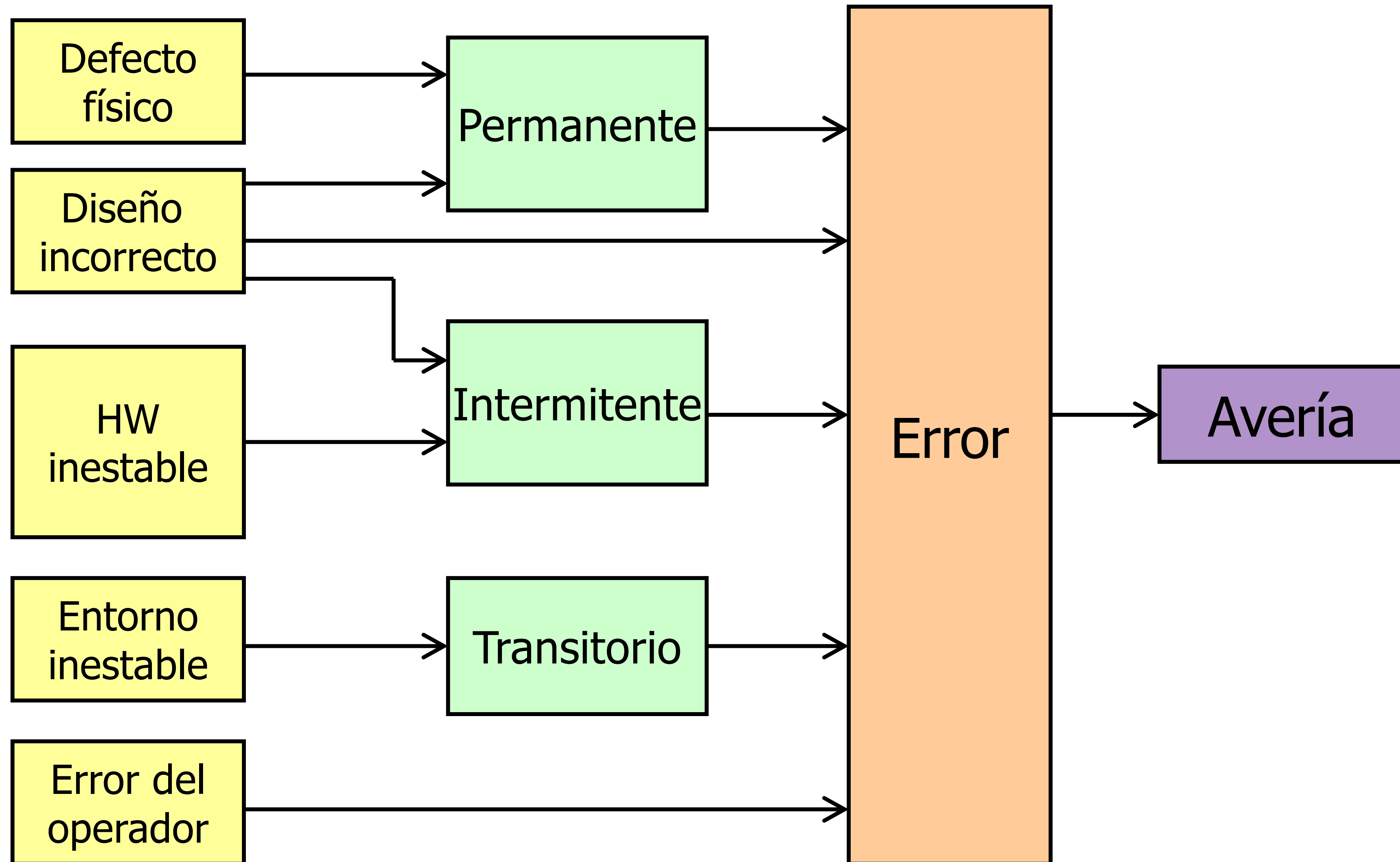
    http://www.cs.tau.ac.il/~nachumd/verify/horror.html

Tipos de fallos

- Fallos permanentes
  - Permanecen hasta que el componente se repara o sustituye.
  - Ejemplo: roturas en el hardware, errores de software.
- Fallos (temporales) transitorios
  - Desaparecen solos al cabo de un cierto tiempo.
  - Ejemplo: interferencias en comunicaciones, fallos transitorios en los enlaces de comunicación.
- Fallos (temporales) intermitentes:
  - Fallos transitorios que ocurren de vez en cuando.
  - Ejemplo: calentamiento de un componente hardware.

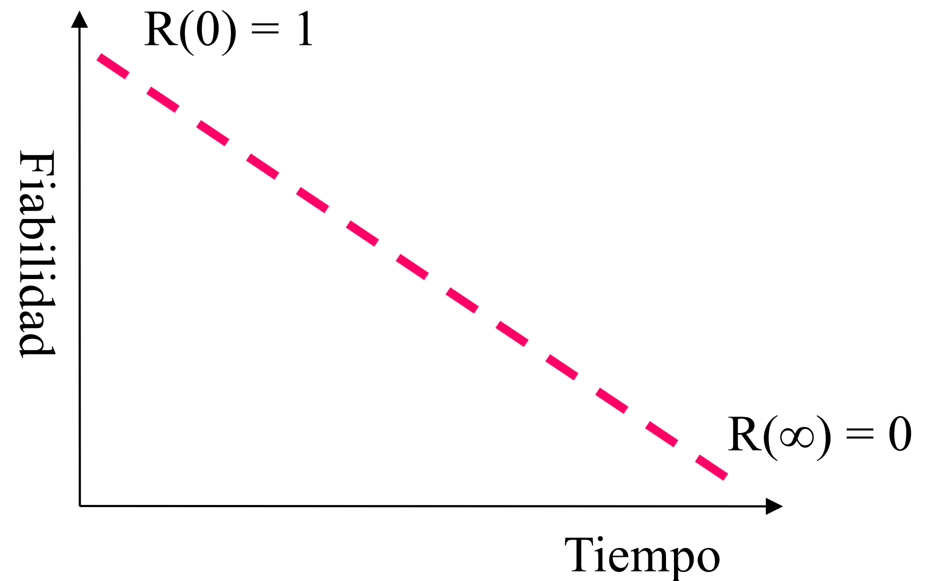
- **Objetivo**: evitar que los fallos produzcan averías.

Tipos de fallos



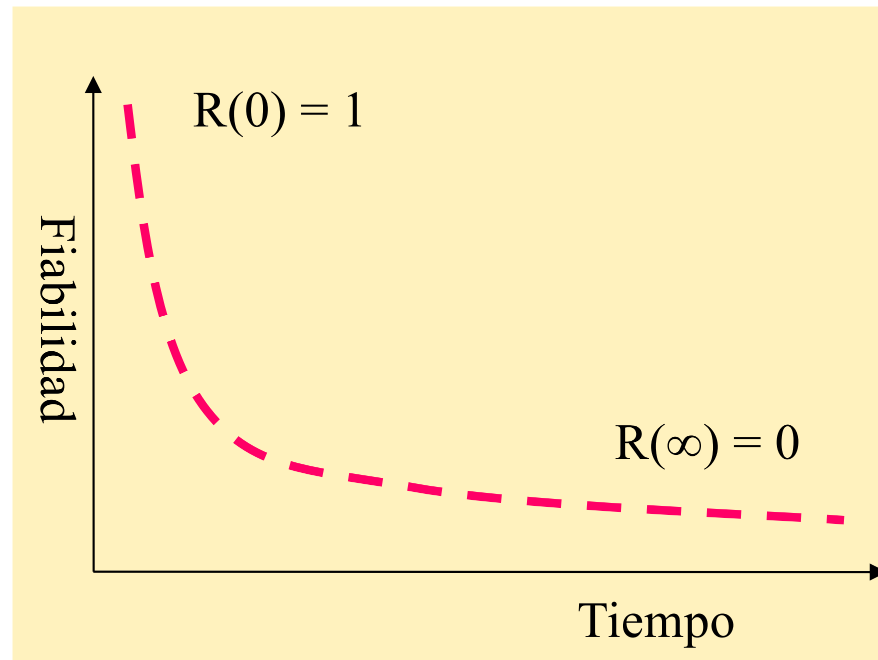
Fiabilidad

- El tiempo de vida de un sistema se representa mediante una variable aleatoria X
- Se define la **fiabilidad** del sistema como una función R(t)
  - R(t) = P(X > t)
  - De forma que:
    - R(0) = 1 y R(∞) = 0



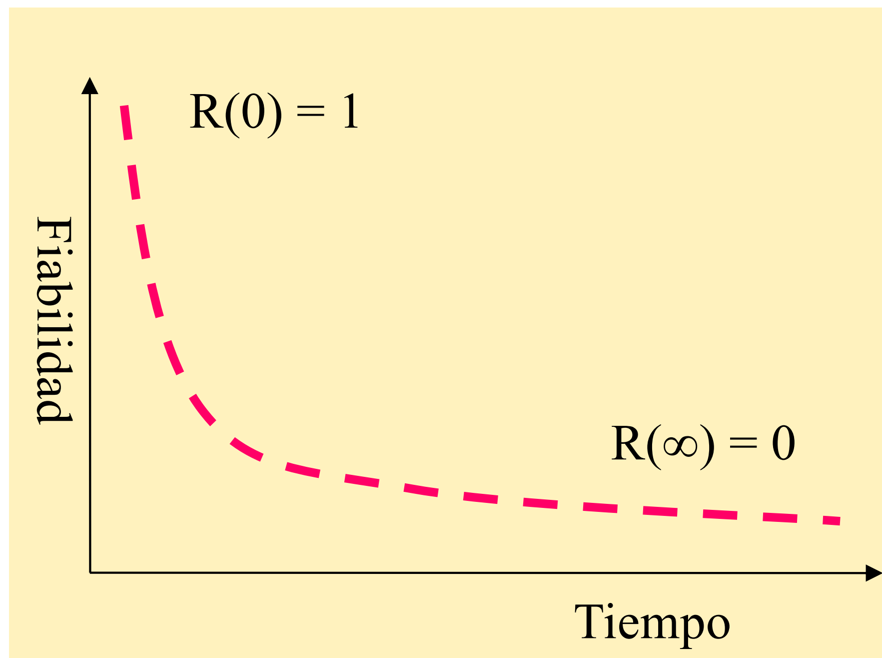
Fiabilidad

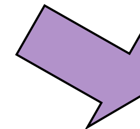
- A partir del estudio de los fallos de los componentes se obtiene la fiabilidad

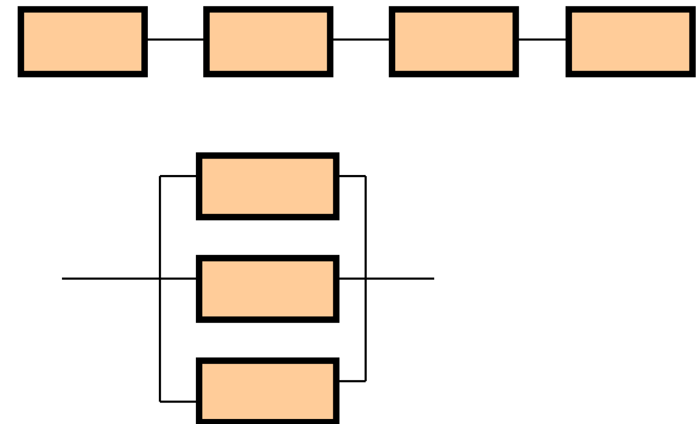


Fiabilidad

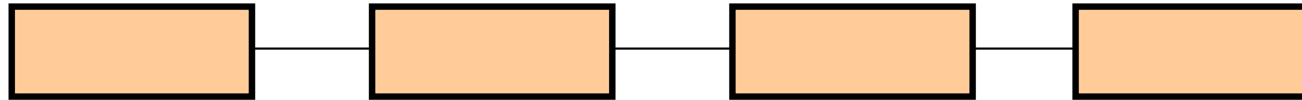
- A partir de la fiabilidad de los componentes es posible obtener la fiabilidad del sistema







Sistema serie



- Sea $R_i(t)$ la fiabilidad del componente **i**
- El sistema falla cuando algún componente falla
- Si los fallos son independientes entonces

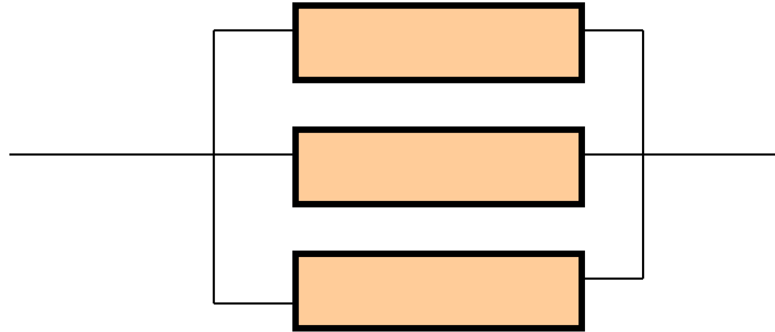
$$R(t) = \prod_{i=1}^{N} R_i(t)$$

- Se cumple que:

$$R(t) < R_i(t) \qquad \forall i$$

- La fiabilidad del sistema es menor

Sistema paralelo

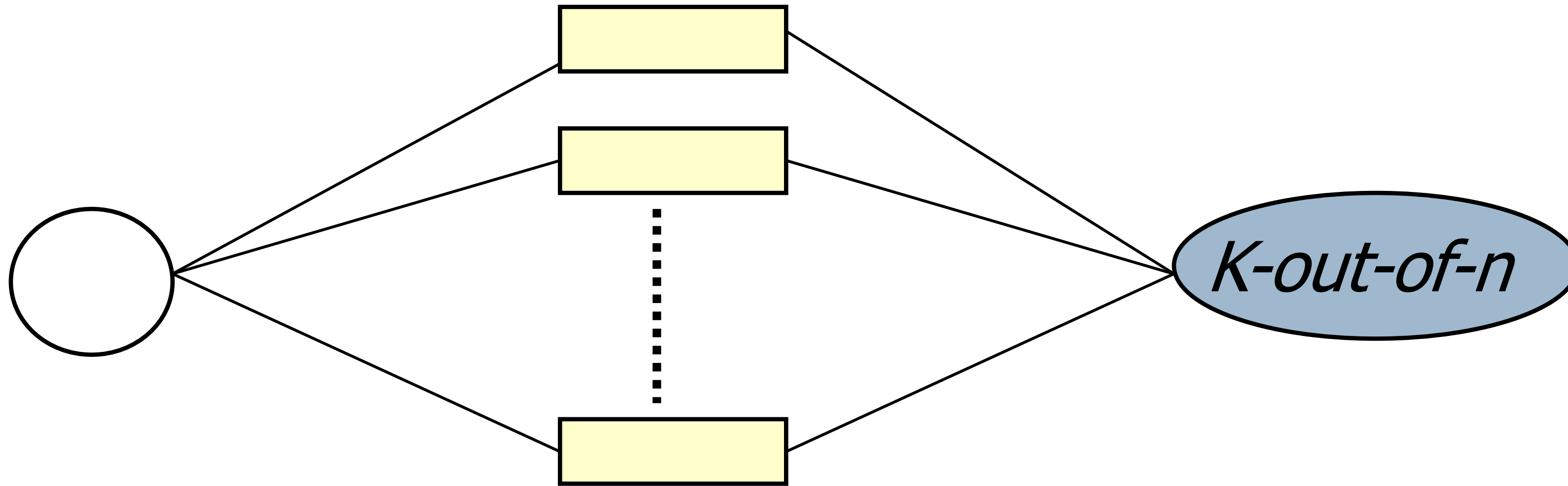


- El sistema falla cuando fallan todos los componentes

$$R(t) = 1 - \prod_{i=1}^{N} Q_i(t) \quad \text{donde} \quad Q_i(t) = 1 - R_i(t)$$

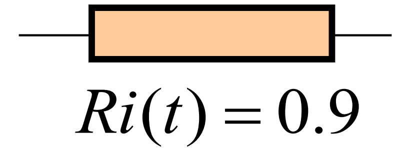
Fiabilidad de un sistema *k-out-of-n*

El sistema funciona cuando funcionan al menos k de n

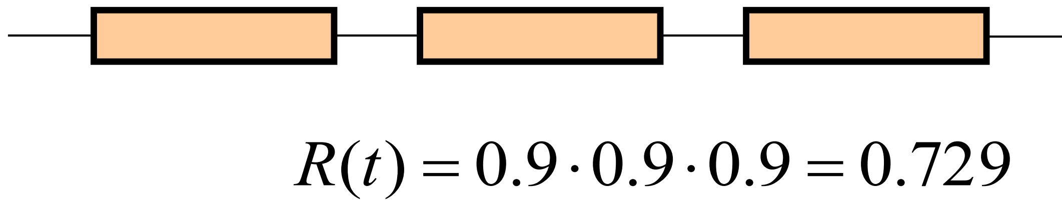


$$R(k,n) = \sum_{r=k}^{r=n} \binom{n}{r} R^r (1-R)^{n-r}$$

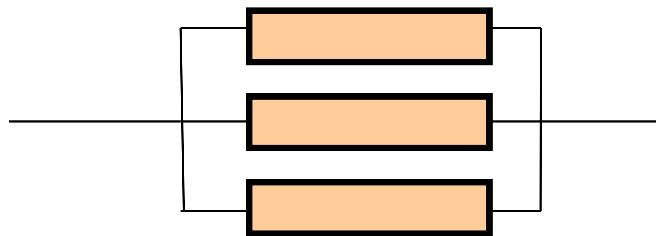
Ejemplo



$$Ri(t) = 0.9$$



$$R(t) = 0.9 \cdot 0.9 \cdot 0.9 = 0.729$$



$$R(t) = 1 - (1 - 0.9)^3 = 0.999$$

Ejemplo

- ¿Cuál es la fiabilidad de un RAID0 con 5 discos, si la fiabilidad de cada disco es R_i=0.99?

- ¿Cuál es la fiabilidad de un RAID4 con 5 discos, si la fiabilidad de cada disco es R_i=0.99?

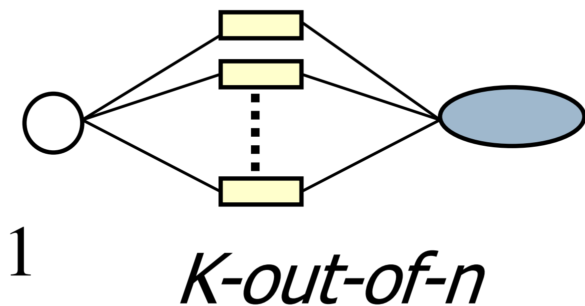
Ejemplo

- ¿Cuál es la fiabilidad de un RAID0 con 5 discos, si la fiabilidad de cada disco es R=0.99?

$$R = R_i^5 = 0.9039$$

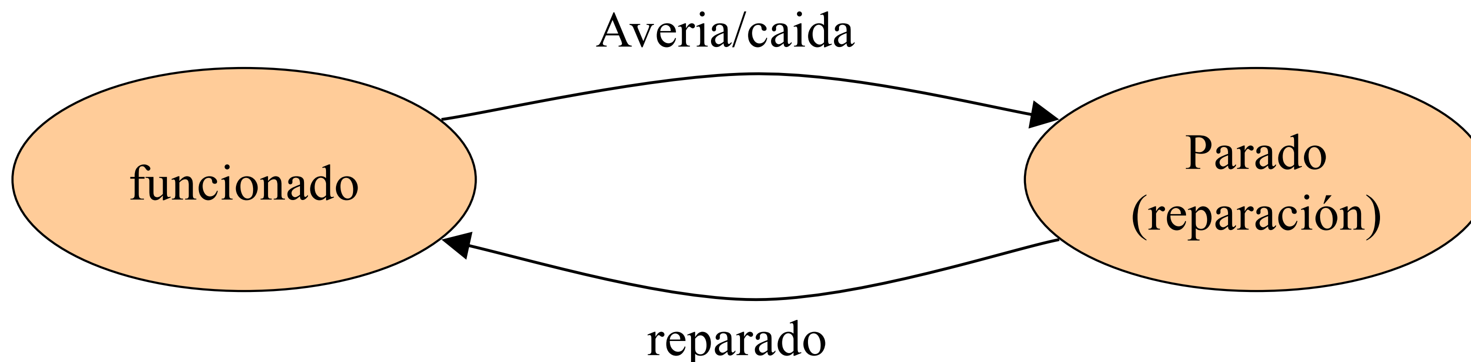
- ¿Cuál es la fiabilidad de un RAID4 con 5 discos, si la fiabilidad de cada disco es R=0.99?

$$R(4,5) = \sum_{r=4}^{r=5} \binom{5}{r} R_i^r \left(1 - R_i\right)^{5-r} = 0.9961$$



Disponibilidad

- En muchos casos es más interesantes conocer la disponibilidad
- Se define la disponibilidad de un sistema A(t) se define como la probabilidad de que el sistema esté funcionando correctamente en el instante t
  - **La fiabilidad considera el intervalo [0,t]**
  - **La disponibilidad considera un instante concreto de tiempo**
- Un sistema se modeliza según el siguiente diagrama de estados



Tipos de paradas

- Mantenimiento correctivo
  - Debido a fallos
- Mantenimiento preventivo
  - Para prevenir fallos
  - Pueden planificarse

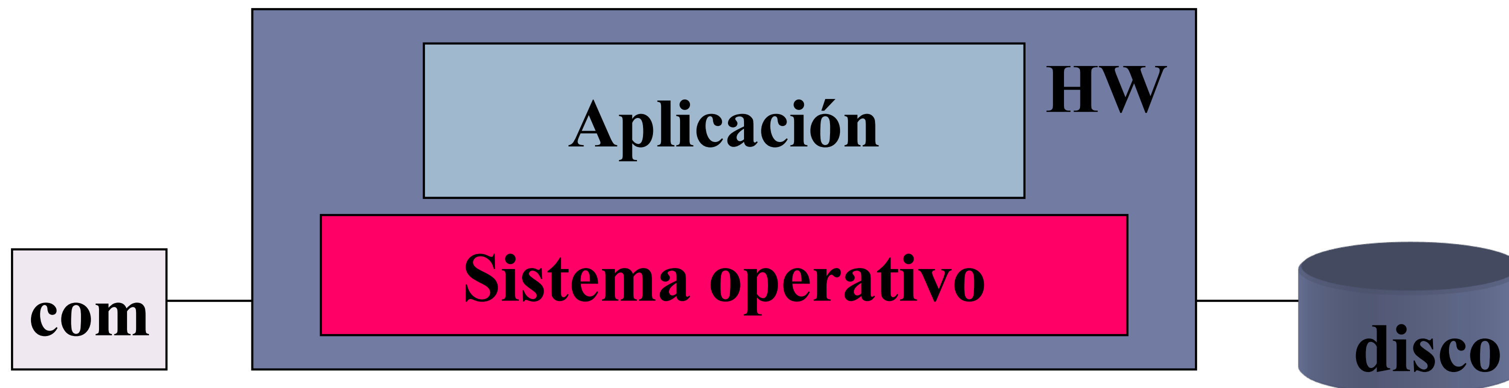
Medida de la disponibilidad

- Sea TMF el tiempo medio hasta el fallo
- Sea TMR el tiempo medio de reparación
- Se define la disponibilidad de un sistema como:

$$disponibilidad = \frac{TMF}{TMF + TMR}$$

- ¿Qué significa una **disponibilidad** del **99%**?
  - En **365** días funciona correctamente 99*365/100 = 361.3 días
  - **Está sin servicio 3.65 días**

Cálculo de la disponibilidad



- Disponibilidad de los elementos:
  - HW: 99.99 %
  - Disco: 99.9 %
  - SO: 99.99 %
  - Aplicación: 99.9 %
  - Comunicación 99.9
- Disponibilidad del sistema: 99.6804 % (1.17 días sin servicio)

$$A(t) = \prod A_i(t)$$

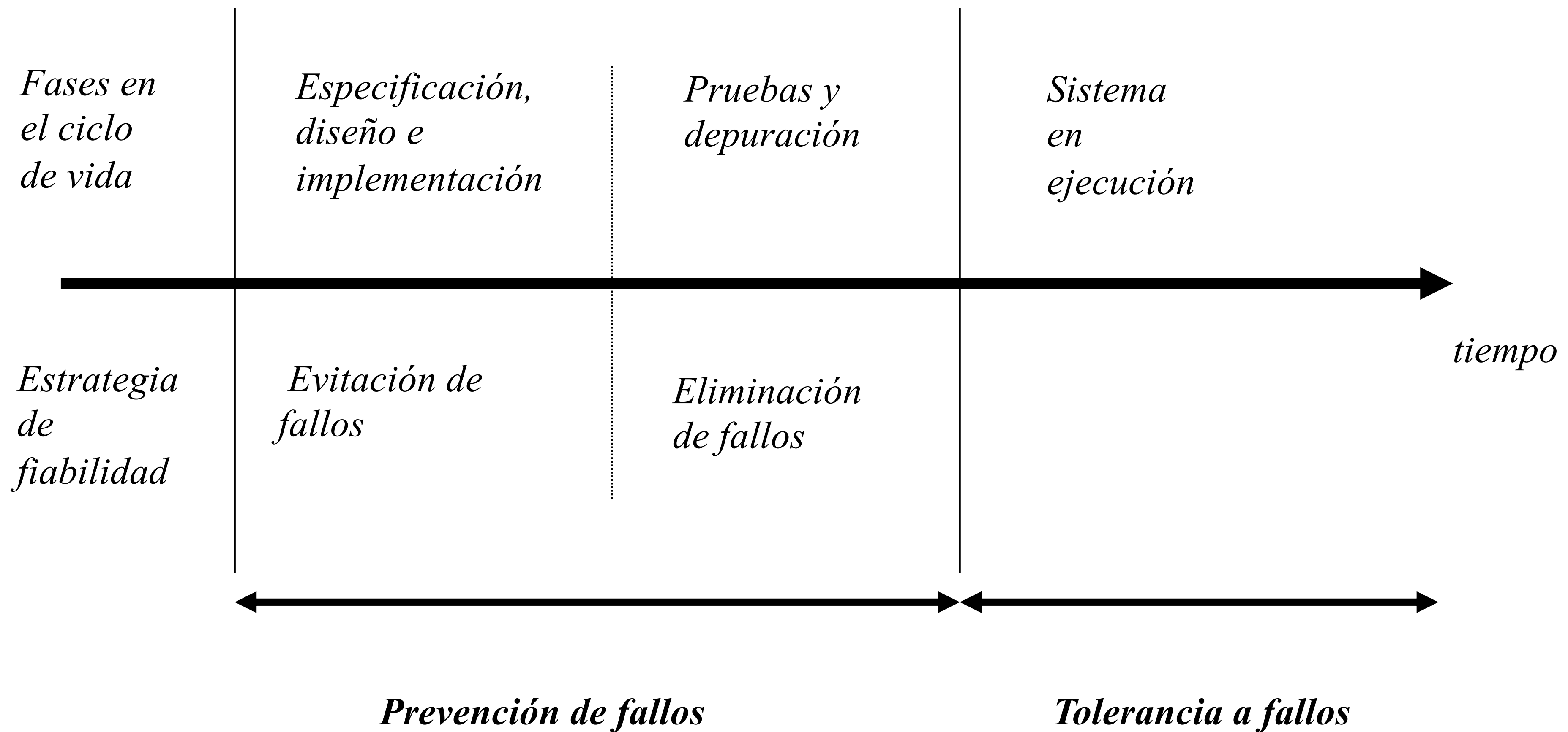
Tiempo anual sin servicio

Disponibilidad (%)	Días sin servicio al año
98%	7.3 días
99%	3.65 días
99.8%	17 horas, 30 minutos
99.9%	8 horas, 45 minutos
99.99%	52 minutos, 30 segundos
99.999%	5 minutos, 15 segundos
99.9999%	31.5 segundos

Técnicas para aumentar la fiabilidad

- **Prevención de fallos**
  - Evitar que se introduzcan fallos en el sistema antes que entre en funcionamiento.
  - Se utilizan en la fase de desarrollo del sistema.
    - Evitar fallos.
    - Eliminar fallos.
- **Tolerancia a fallos**
  - Conseguir que el sistema continúe funcionando aunque se produzcan fallos.
  - Se utilizan en la etapa de funcionamiento del sistema.
  - Es necesario saber los posibles tipos de fallos, es decir, anticiparse a los fallos.

Técnicas para obtener fiabilidad



Prevención de fallos

- **Evitación de fallos**: evitar la introducción de fallos en el desarrollo del sistema.
  - Uso de componentes muy fiables.
  - Especificación rigurosa, métodos de diseño comprobados.
  - Empleo de técnicas, herramientas y personal adecuadas.
- **Eliminación de fallos**: eliminar los fallos introducidos durante la construcción del sistema.
  - No se puede evitar la introducción de fallos en el sistema (errores en el diseño, programación).
  - Revisiones del diseño.
  - Pruebas del sistema.

Limitaciones de la prevención de fallos

- Los componentes hardware se deterioran y fallan.
  - La sustitución de componentes no siempre es posible:
    - No se puede detener el sistema.
    - No se puede acceder al sistema.
- Deficiencias en las pruebas
  - No pueden ser nunca exhaustivas.
  - Sólo sirven para mostrar que hay errores, pero no permiten demostrar que no los hay.
  - A veces es imposible reproducir las condiciones reales de funcionamiento del sistema.
  - Los errores de especificación no se detectan.

Solución: utilizar técnicas de **tolerancia a fallos**.

Tolerancia a fallos

- La tolerancia a fallos se basa en la **redundancia**.
- Se utilizan componentes adicionales para **detectar** los fallos, **enmascararlos** y **recuperar** el comportamiento correcto del sistema.
- El empleo de redundancia aumenta la complejidad del sistema y puede introducir fallos adicionales si no se gestiona de forma correcta.

Grados de tolerancia a fallos

- ***Tolerancia completa***: el sistema continúa funcionando, al menos durante un tiempo, sin pérdida de funcionalidad ni de prestaciones.
- ***Degradación aceptable***: el sistema sigue funcionando en presencia de errores pero con una pérdida de funcionalidad o de prestaciones hasta que se repare el fallo.
- ***Parada segura***: el sistema se detiene en un estado que asegura la integridad del entorno hasta que el fallo sea reparado.
  - Trenes
  - Airbus A320

- El nivel de tolerancia a fallos depende de cada aplicación.

Fases en la tolerancia a fallos

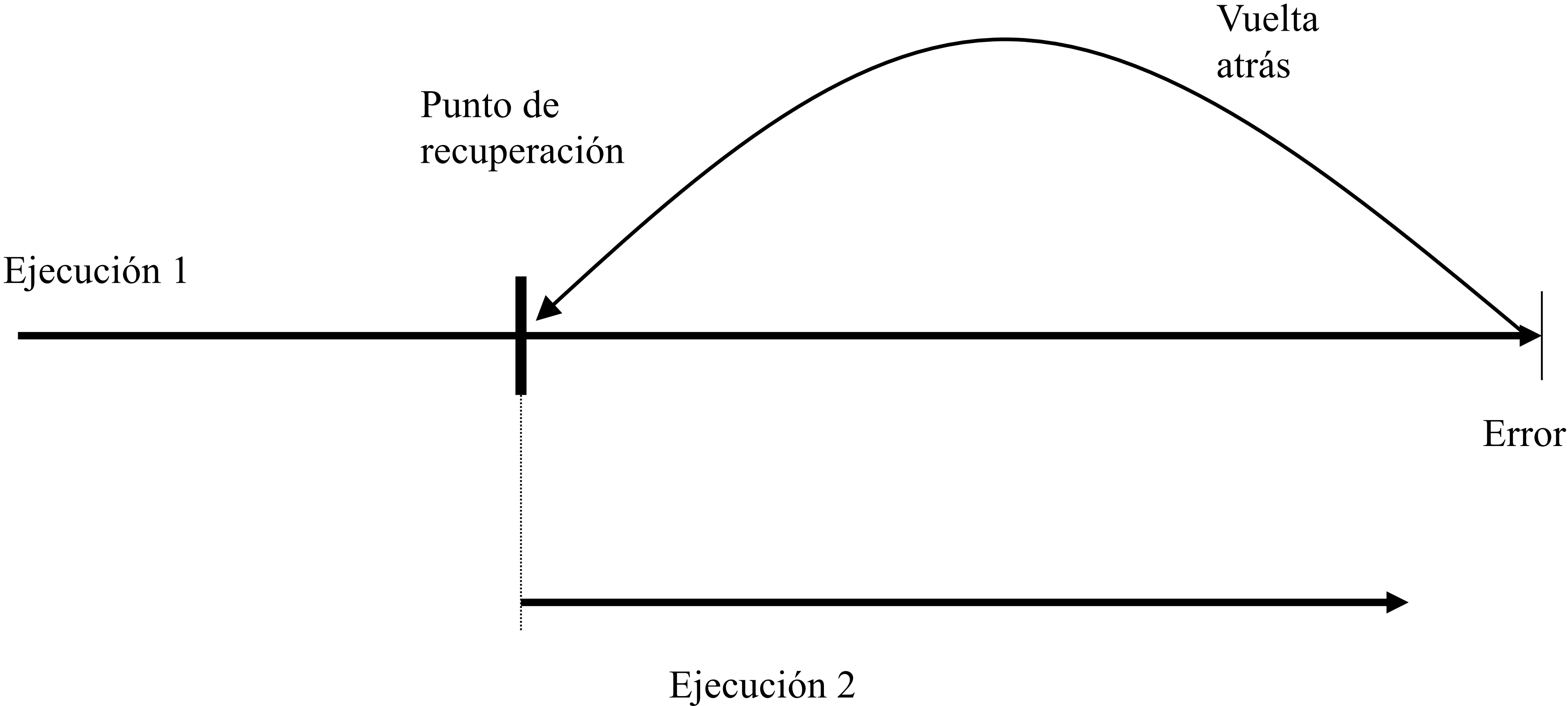
1. Detección de errores
2. Confinamiento y diagnostico de daños
3. Recuperación de errores
4. Tratamiento de fallos y servicio continuado

- Estas cuatro fases constituyen la base de todas las técnicas de tolerancia a fallos y deberían estar presentes en el diseño e implementación de un sistema tolerante a fallos.

Recuperación de errores

- Una vez detectado el error es necesario recuperar al sistema del error.
- Es necesario utilizar técnicas que transformen el estado erróneo del sistema en otro estado bien definido y libre de errores.
- Dos técnicas básicas:
  - ***Recuperación hacia atrás***. Cuando se detecta un error en el sistema se vuelve a un estado anterior libre de errores.
    - Puntos de recuperación o *Checkpoints*
  - ***Recuperación hacia adelante.*** Llevar al sistema a un estado libre de errores (no volviendo hacia atrás).

Recuperación hacia atrás



Tratamiento de fallos y servicio continuado

- Una vez que el sistema se encuentra libre de errores es necesario que siga ofreciendo el servicio demandado.
- Una vez detectado un error:
  - Se repara el fallo.
  - Se reconfigura el sistema para evitar que el fallo pueda volver a generar errores.
  - Cuando los errores fueron transitorios no es necesario realizar ninguna acción.

Redundancia

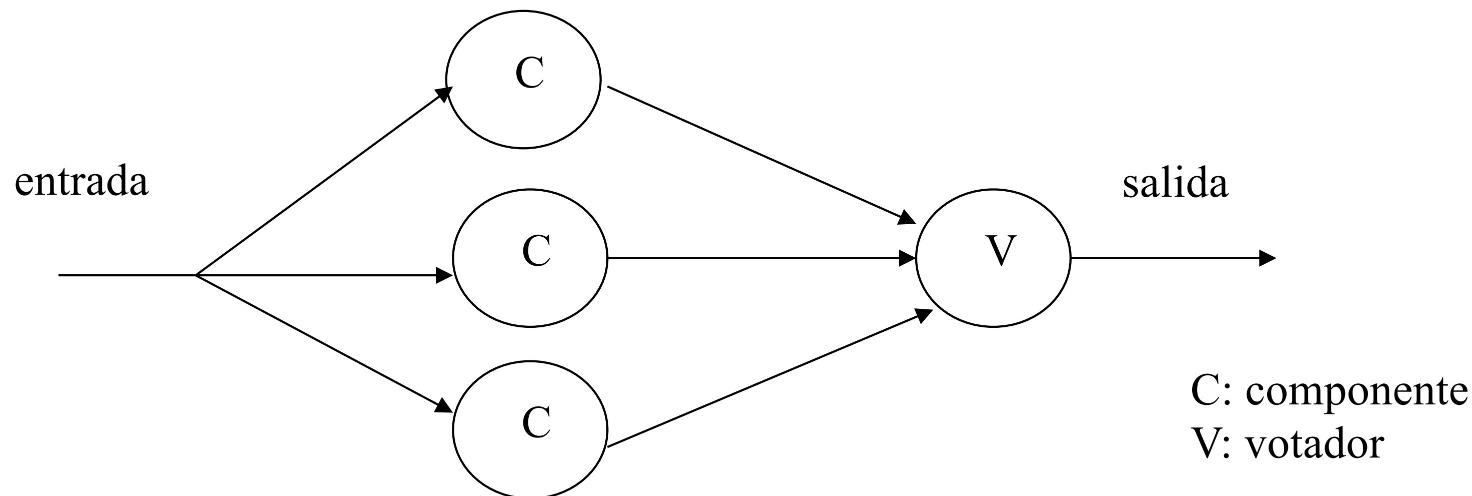
- Todas las técnicas de tolerancia a fallos se basan en el uso de la **redundancia**.
- *Redundancia estática*: Los componentes redundantes se utilizan dentro del sistema para enmascarar los efectos de los componentes con defectos.
- *Redundancia dinámica*. La redundancia se utiliza sólo para la detección de errores. La recuperación debe realizarla otro componente.
- *Redundancia en el diseño*. Partes de un diseño pueden ser redundantes.
- *Redundancia temporal*. Mediante el uso repetido de un componente en presencia de fallos. Adecuado para fallos transitorios.

Estrategias en hardware

- Sistema triple modular redundante
- Duplicación y comparación
- Sistemas mixtos (TMR+DC)
- Ejemplos de sistemas hardware con estas técnicas

Redundancia modular triple (TMR)

- Ejemplo de redundancia estática.



- NMR: redundancia con N componentes redundantes
  - Para permitir F fallos se necesitan N módulos, con **N = 2F+1**

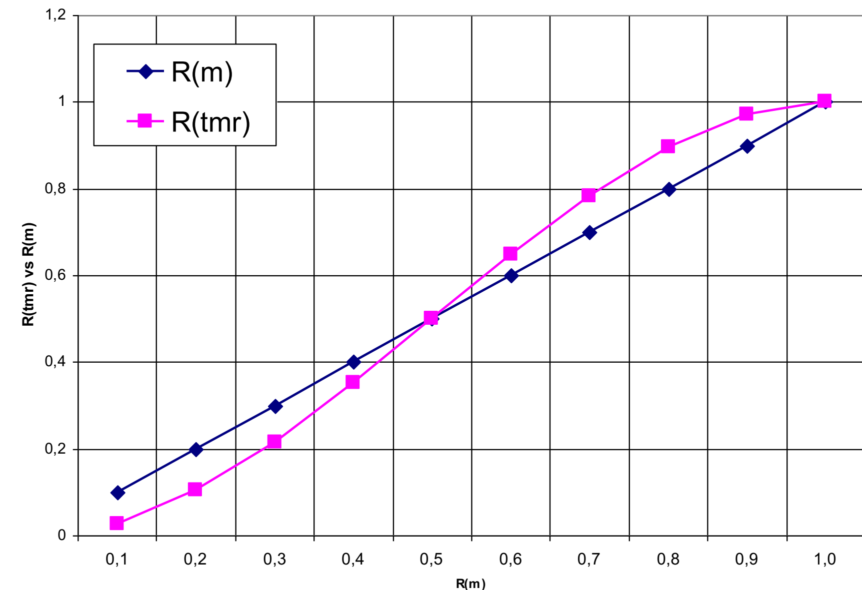
Fiabilidad de un sistema TMR

- Fiabilidad de un sistema TMR es:

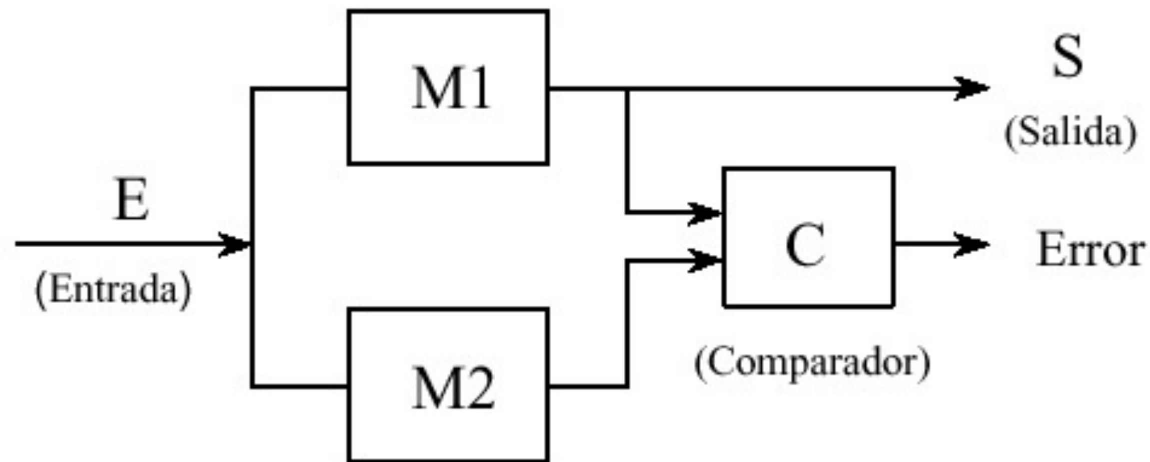
$$R_{TMR} = R_m^{\,3} + 3R_m^{\,2}(1 - R_m) = 3R_m^{\,2} - 2R_m^{\,3}$$

  - Donde R_m es la fiabilidad de un componente

- No siempre es mejor un TMR:
  - $R_{TMR} < R_m$ si $R_m < 0.5$
    - Cuando la fiabilidad del componente es muy baja la redundancia no mejora la fiabilidad
  - Para $R_m = 0.9$, $R_{TMR} = 0.972$



Duplicación y comparación

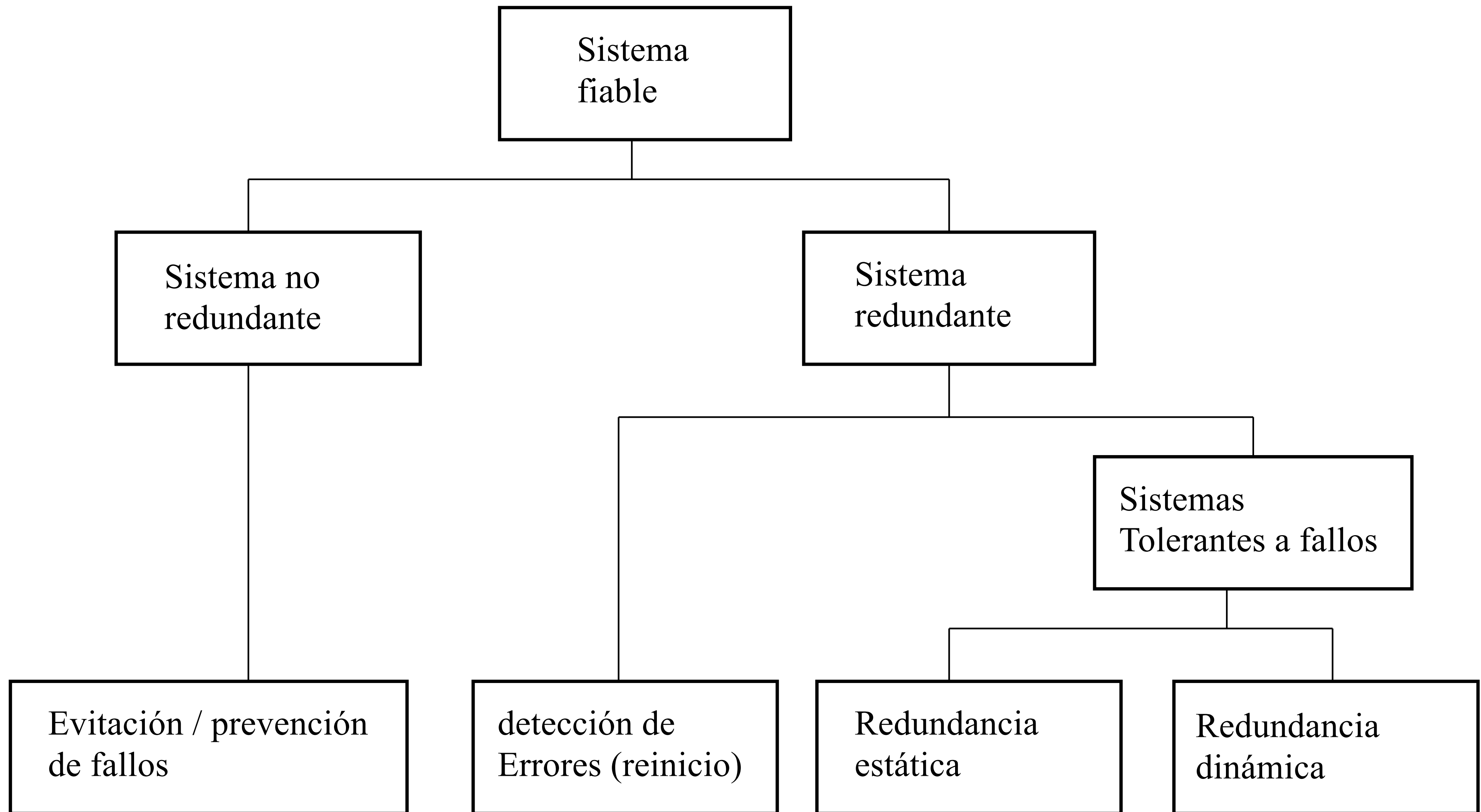


- Duplicación y comparación
  - Ejemplo de detección de errores (reinicio).
- Códigos detectores y correctores
  - Ejemplo de redundancia dinámica.

Diseño de sistemas tolerantes a fallos

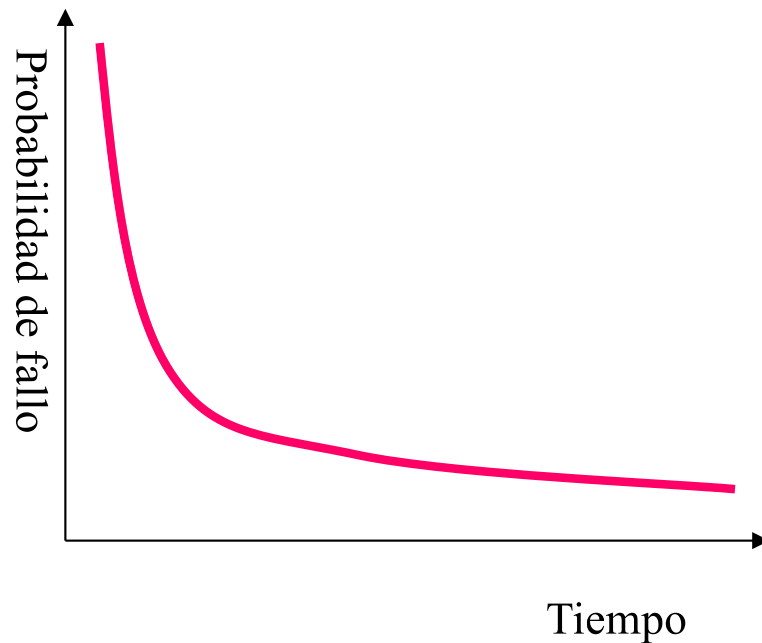
- Para diseñar un sistema tolerante a fallos sería ideal **identificar** todos los posibles fallos y evaluar las técnicas adecuadas de tolerancia a fallos.
  - Sin embargo:
    - Hay fallos que se pueden anticipar (fallos en el HW).
    - Hay fallos que no se pueden anticipar (fallos en el SW).
  - Los errores surgen por:
    - Fallos en los componentes.
    - Fallos en el diseño.
- Objetivo:
  - **Maximizar** la **fiabilidad** del sistema.
  - **Minimizar** la **redundancia**
    (↑ Redundancia → ↑ Complejidad → ↑ Probabilidad errores)

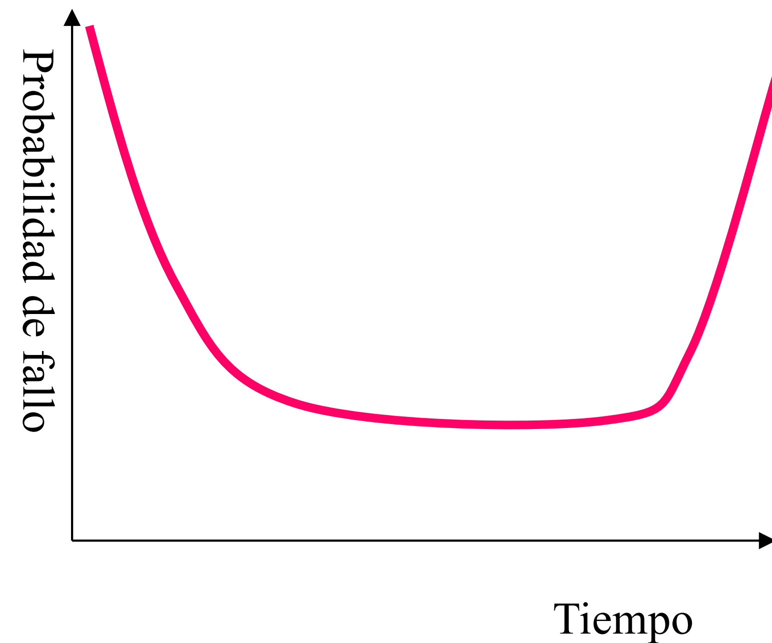
Estrategias para diseñar un sistema fiable



Tolerancia a fallos de software

- Los fallos en el software se deben a fallos en el diseño.
- Funciones típicas de fallos aplicadas al software.



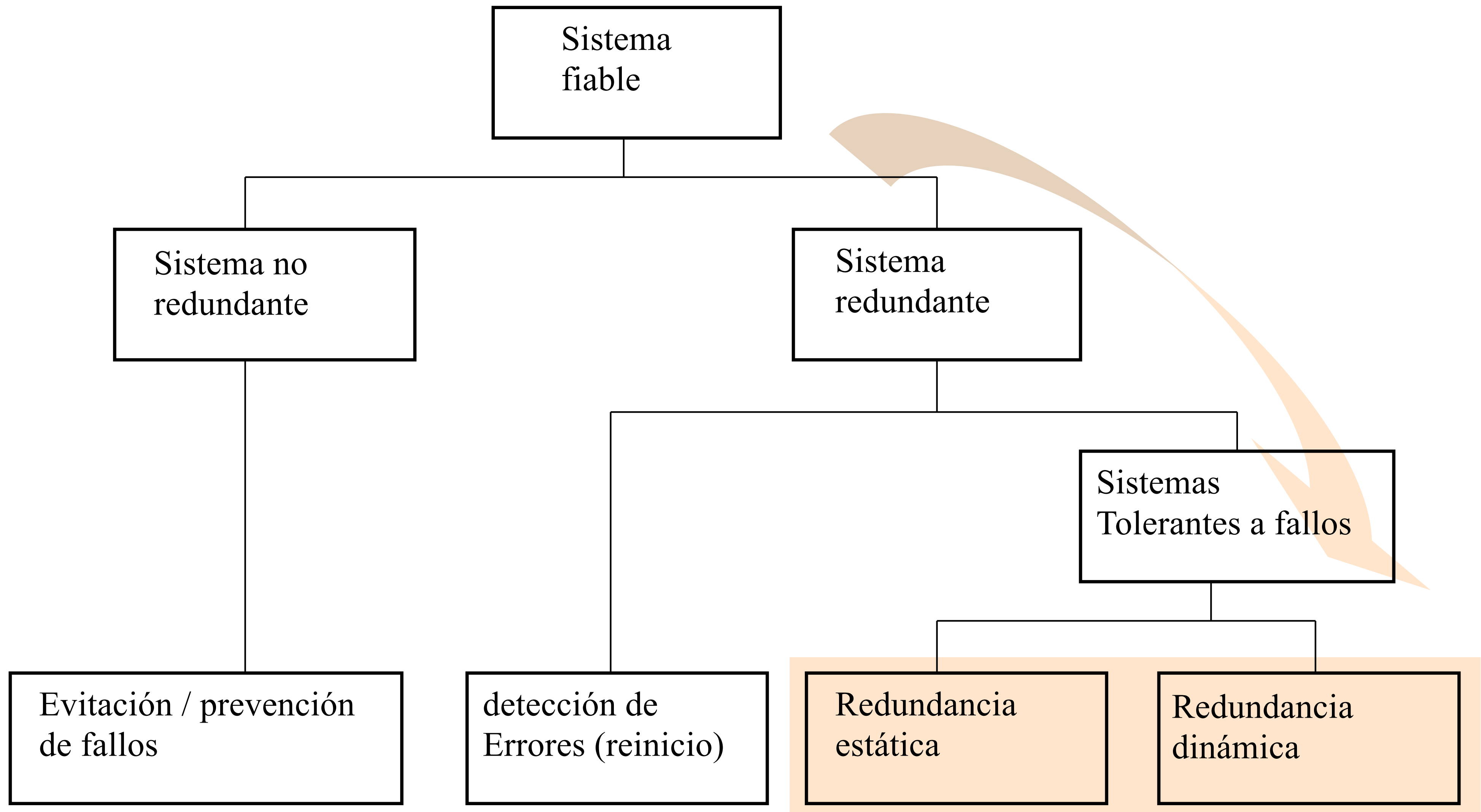


- Las técnicas de tolerancia a fallos de SW permiten obtener una alta fiabilidad a partir de componentes de menor fiabilidad

Ejemplos de fallos software

- Especificaciones inadecuadas
- Errores de codificación
- Errores de diseño
- Pérdidas de memoria que degeneran el sistema

Redundancia en el software



Redundancia en el software

```
try {
  // instrucciones;
  // throw
}
cath (exception E) {
  // manejador E
}
```

- Soporte para tolerancia a fallos:
  - Señales
  - Excepciones
    - Facilitan **la detección** y **recuperación** de errores.
    - Una excepción es una manifestación de un cierto tipo de error.
    - Cuando se produce un error en el sistema se genera (eleva) la excepción correspondiente en el contexto donde se ha invocado la actividad que ha dado lugar al error.
    - De esta forma se puede manejar la excepción en este contexto.
  - *Checkpoint*
  - *Asserts*

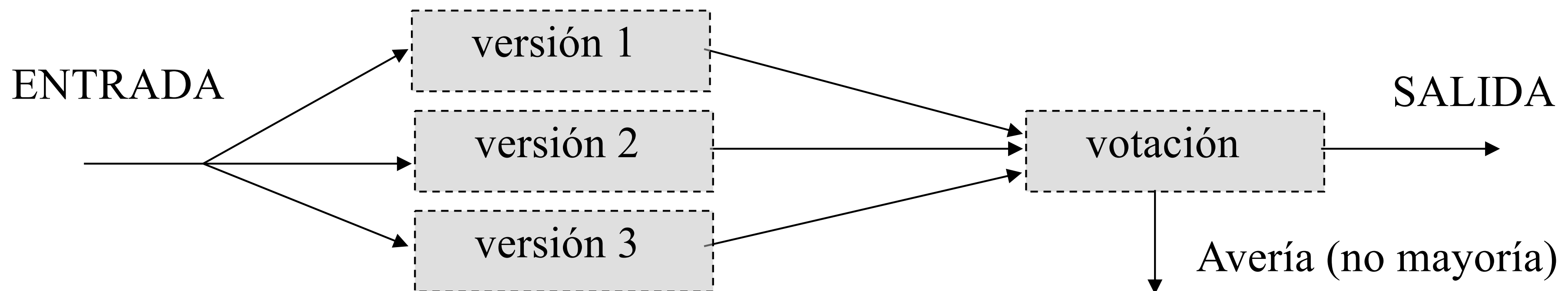
Redundancia en el software

- Técnicas para detectar y corregir errores de diseño.
  - Redundancia estática
    - **Programación con N versiones**
  - Redundancia dinámica en el software:
    - **Bloques de recuperación**
      - Proporcionan recuperación hacia atrás (y adelante)
    - **Programación con N versiones autocomprobantes**

- ***Todos los métodos son sensibles a los errores en los requisitos***

Redundancia estática

- Programación con N versiones:
  - La programación N-versión se define como la generación independiente de N (N>=2) programas a partir de una misma especificación.
  - Los programas se ejecutan **concurrentemente, con la misma entrada** y sus resultados son comparados por un proceso **coordinador**.
  - El resultado han de ser el mismo.
    Si hay discrepancia, se realiza una votación.



Tolerancia a fallos con N versiones

- *Detección de errores:*
  - La realiza el programa votador.
- *Confinamiento y diagnostico de daños:*
  - No es necesaria ya que las versiones son independientes.
- *Recuperación de errores:*
  - Se consigue descartando los resultados erróneos.
- *Tratamiento de fallos y servicio continuado:*
  - Se consigue ignorando el resultado de la versión errónea.

Si todas las versiones producen valores diferentes se detecta el error pero no se ofrece recuperación.

Redundancia estática

- La programación con N versiones depende de:
  - Una *especificación inicial correcta.*
    - Un error de especificación aparece en todas las versiones.
  - Un *desarrollo independiente*
    - No debe haber interacción entre equipos de desarrollo.
    - Uso incluso de lenguajes de programación distintos.
    - No está claro que programadores distintos cometan errores independientes.
  - Disponer de un *presupuesto suficiente*
    - Los costes de desarrollo se multiplican.
    - El mantenimiento también es más costosa.
    - Para N versiones no está claro si el presupuesto será N veces el presupuesto necesario para una versión.

Redundancia dinámica

- Redundancia dinámica en el software:
  - Los componentes redundantes sólo se ejecutan cuando se detecta un error.
- Se aplican las cuatro fases:
  1. Detección de errores
  2. Confinamiento y diagnostico de daños
  3. Recuperación de errores
  4. Tratamiento de fallos y servicio continuado
- Técnicas principales:
  - Bloques de recuperación
  - Programación con N versiones autocomprobantes

1) Detección de errores

- ***Por el entorno de ejecución:***
  - Error de hardware (ej. instrucción ilegal, división por cero).
  - Sistema operativo (ej: referencia a un puntero nulo).
- ***Por el software de aplicación:***
  - Duplicación (redundancia con dos versiones).
  - Códigos detectores de error.
  - Validación del estado del sistema.
  - Validación estructural.
    - Comprueban la integridad de objetos como listas o colas (número de elementos, punteros redundantes).

2) Confinamiento y diagnóstico de daños

- Importante confinar los daños provocados por un fallo a una parte limitada del sistema.
- Para ello se estructura el sistema de forma que se minimice el daño causado por los componentes defectuosos.
- Técnicas:
  - Descomposición modular: confinamiento estático.
  - Acciones atómicas: confinamiento dinámico.
    - Mueven al sistema de un estado consistente a otro estado consistente.

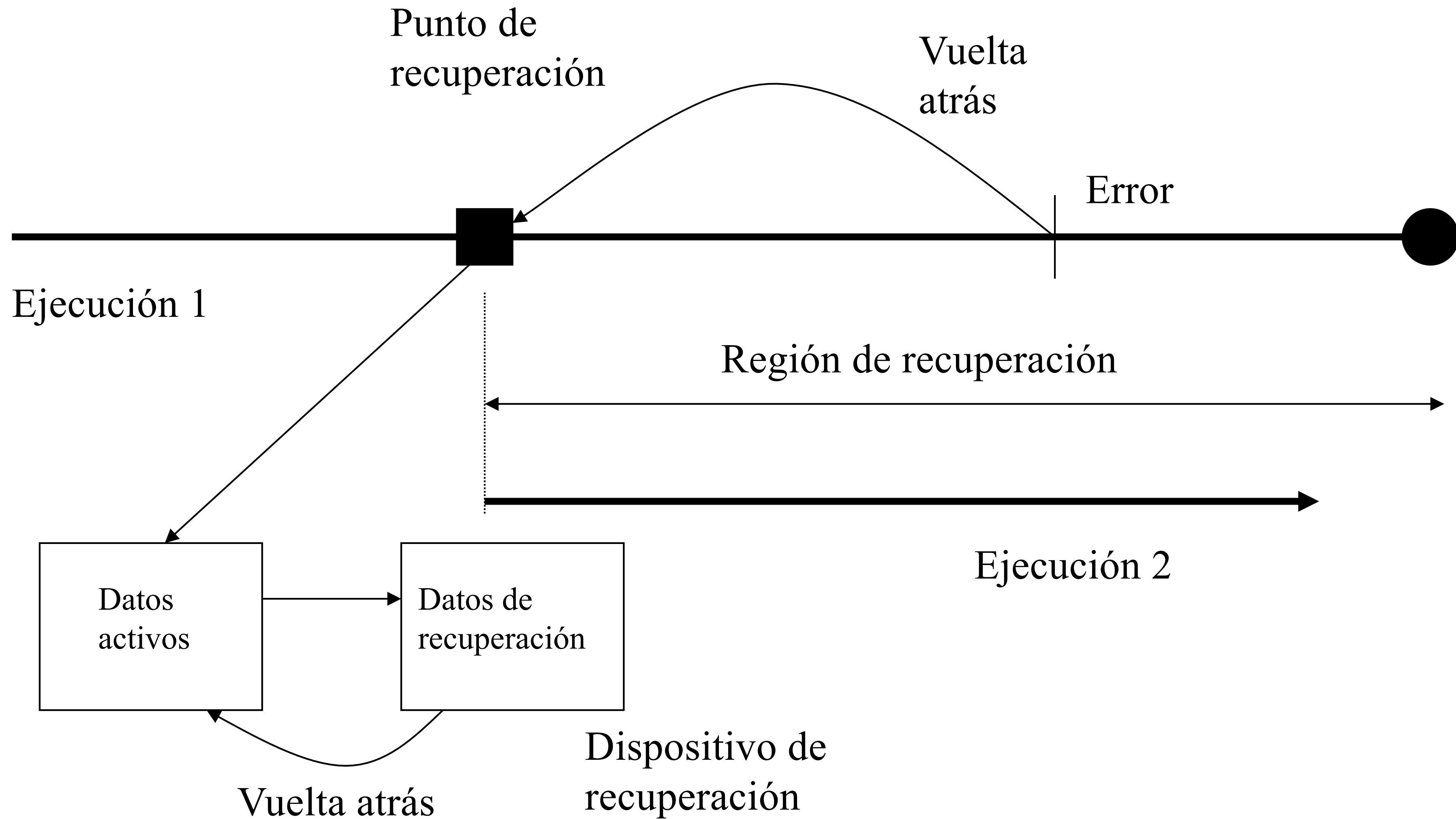
3) Recuperación de errores

- Situar el sistema en un estado correcto desde el cual se puede seguir funcionando.
- Etapa más importante.
- Dos formas:
  - ***Recuperación hacia adelante:***
    - Se avanza desde un estado erróneo haciendo correcciones sobre partes del estado.
  - ***Recuperación hacia atrás:***
    - Se retrocede a un estado anterior correcto que se ha guardado previamente.

3.a) Recuperación hacia adelante

- Toma como punto de partida los datos erróneos que sometidos a determinadas transformaciones permiten alcanzar un estado libre de errores.
- Depende de una predicción correcta de los posibles fallos y de su situación.
- Ejemplos:
  - Códigos autocorrectores que emplean bits de redundancia.

3.b) Recuperación hacia atrás



Conceptos

- ***Punto de recuperación*** (*checkpoint):* instante en el que se salvaguarda el estado del sistema.
- ***Datos de recuperación***: datos que se salvaguardan.
  - Registros de la máquina.
  - Datos modificados por el proceso (variables globales y pila).
    - Páginas del proceso modificadas desde el último punto de recuperación.
- ***Datos activos***: conjunto de datos a los que accede el sistema después de establecer un punto de recuperación.
- ***Vuelta atrás***: proceso por el cual los datos salvaguardados se restauran para restablecer el estado.
- ***Región de recuperación***: periodo de tiempo en el que los datos de recuperación de un punto de recuperación están activos y se pueden restaurar en caso de detectarse un fallo.

Tipos de sistemas

- ***Transparentes a la aplicación***:
  - El establecimiento de los puntos de recuperación y la vuelta atrás queda bajo el control del HW o del sistema operativo.
  - Ventaja: transparencia.
    - Las aplicaciones pueden transportarse sin problemas.
  - Inconveniente: pueden establecerse puntos de recuperación en momentos que no son necesarios (posibles sobrecargas).
- ***Controlados por la aplicación***:
  - El diseñador de la aplicación establece los puntos de recuperación.
    - Momento adecuado.
    - Permite minimizar el conjunto de datos a salvaguardar.
  - Problema: falta de transparencia.

Primitivas necesarias

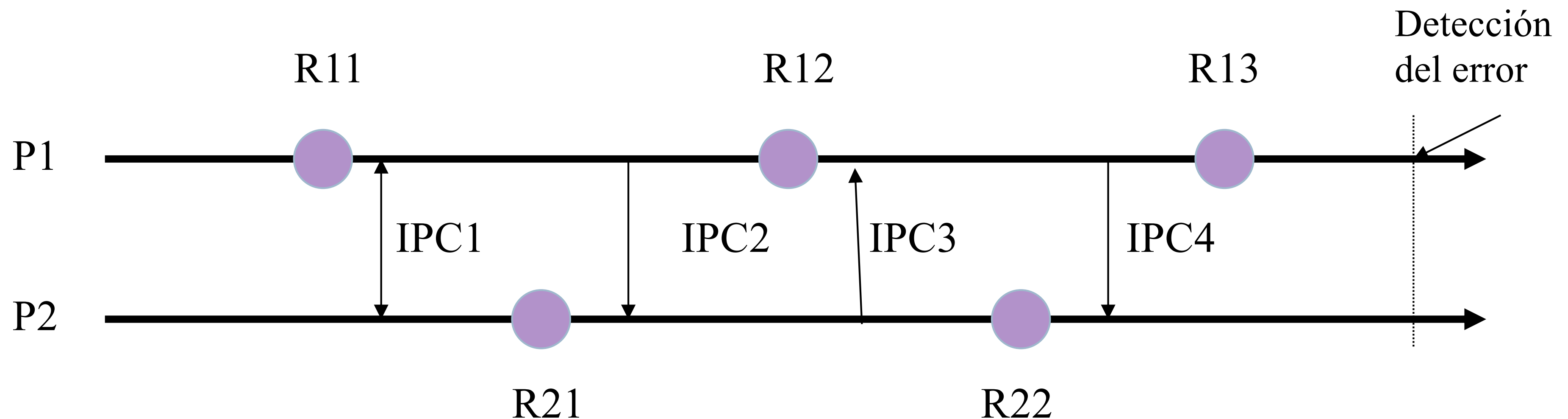
- ***Establecer punto de recuperación***:
  - Salvaguarda los registros y las páginas modificadas por el proceso desde el último punto de recuperación.
- ***Anular punto de recuperación***:
  - Se anulan los datos correspondientes a un punto de recuperación y se libera el espacio ocupado por éstos en el dispositivo de recuperación.
- ***Restaurar punto de recuperación***:
  - Se copian los datos salvaguardados en el dispositivo de recuperación sobre las copias activas.

Puntos de recuperación en sistemas concurrentes

- Tipos de procesos concurrentes:
  - ***Independientes***: la ejecución de un proceso no afecta a otros.
    - La recuperación se realiza como se ha descrito hasta ahora.
  - ***Competitivos***: los procesos comparten recursos del sistema.
    - No comparten datos y se tratan como los procesos independientes.
  - ***Cooperantes (dependientes):*** cooperan e intercambian información entre ellos.
    - Una vuelta atrás en un proceso puede provocar estados inconsistentes en otros.

Efecto dominó

- Se produce un conjunto de vuelta atrás no acotado que puede llegar a reiniciar el sistema concurrente.
- Solución: ***líneas de recuperación***
  - Objetivo: acotar el efecto dominó en caso de realizar una vuelta atrás encontrando un conjunto de procesos y de puntos de recuperación que permita hacer volver al sistema a un estado consistente.



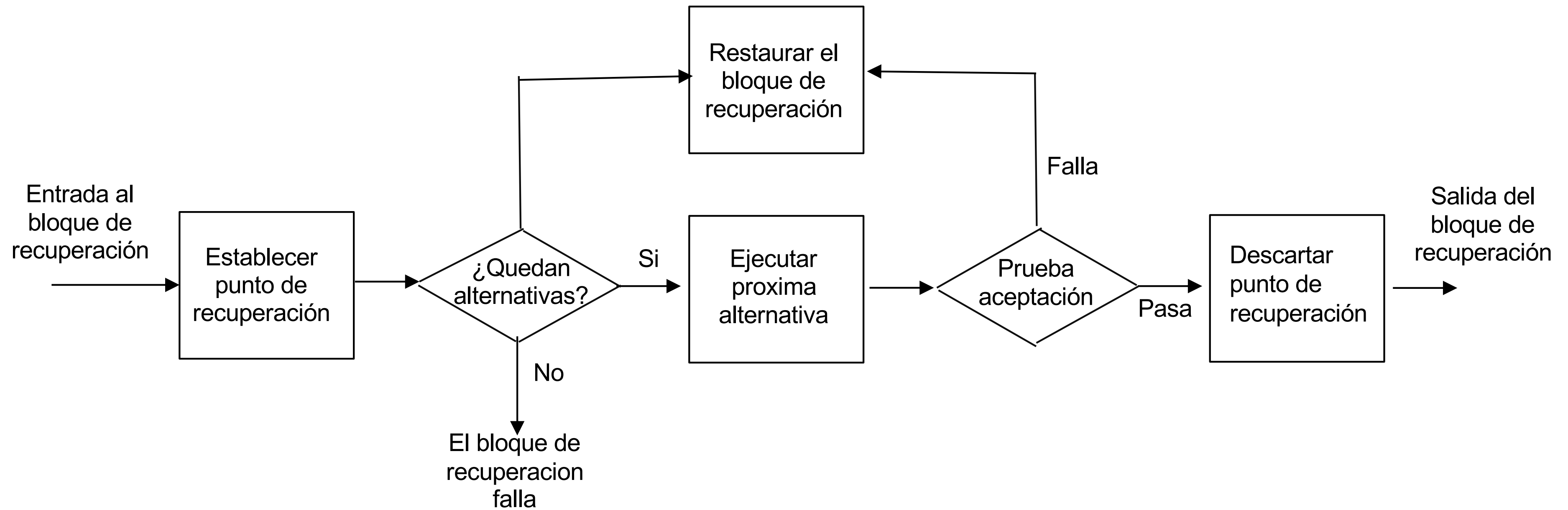
Redundancia dinámica

- Redundancia dinámica en el software:
  - Los componentes redundantes sólo se ejecutan cuando se detecta un error.
- Se aplican las cuatro fases:
  1. Detección de errores
  2. Confinamiento y diagnostico de daños
  3. Recuperación de errores
  4. Tratamiento de fallos y servicio continuado
- Técnicas principales:
  - Bloques de recuperación
  - Programación con N versiones autocomprobantes

Bloques de recuperación

- Técnica de recuperación hacia atrás.
- Un ***bloque de recuperación*** es un bloque tal que:
  - Su entrada es un ***punto de recuperación***.
  - A su salida se realiza una ***prueba de aceptación***
    - Sirve para comprobar si el ***módulo primario*** del bloque termina en un estado correcto.
  - Si la prueba de aceptación falla
    - Se restaura el estado inicial en el punto de recuperación.
    - Se ejecuta un ***módulo alternativo*** del mismo bloque.
  - Si vuelve a fallar, se intenta con otras alternativas.
  - Cuando no quedan módulos alternativos el bloque falla y la recuperación debe realizarse en un nivel más alto.

Esquema de recuperación



Posible sintaxis para bloques de recuperación

```
ensure    < condición  de aceptación >
by        < módulo primario >
else by   < módulo alternativo 1 >
else by   < módulo alternativo 2 >
...
else by   < módulo alternativo N >
else error;
```

- Puede haber bloques anidados
  - Si falla el bloque interior, se restaura el punto de recuperación del bloque exterior.

Prueba de aceptación

- La prueba de aceptación proporciona el mecanismo de detección de errores que activa la redundancia en el sistema.
- El diseño de la prueba de aceptación es crucial para el buen funcionamiento de los bloques de recuperación.
- Hay que buscar un compromiso entre detección exhaustiva de fallos y eficiencia de ejecución.
- No es necesario que todos los módulos produzcan el mismo resultado sino resultados ***aceptables.***
- Los módulos alternativos pueden ser más simples aunque el resultado sea peor para evitar que contengan errores.
- Sobrecarga en aplicaciones de tiempo real

Programación con N versiones autocomprobantes

- Similar a la programación con N versiones pero con redundancia dinámica. Se basa en la diversidad de diseño.
- Cada componente realiza su propia comprobación.
- Dos tipos de componentes autocomprobantes:
  - Una variante y una prueba de aceptación.
  - Dos variantes y un algoritmo de comparación.
- La tolerancia a fallos se consigue por medio de la ejecución paralela de al menos dos componentes:
  - Uno es el activo.
  - Otros de reserva.
- Ejemplo: El Airbus A-320 utiliza un sistema basado en dos componentes autocomprobantes cada uno basado en la ejecución paralela de dos variantes cuyos resultados se comparan.

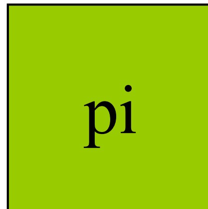
Contenidos

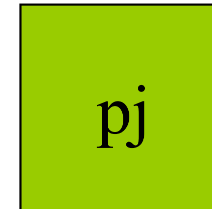
- Introducción a la tolerancia a fallos
- Tolerancia a fallos software
- Tolerancia a fallos en sistemas distribuidos

Clasificación de los fallos en sistemas distribuidos

- Fallo parada: el componente falla de forma permanente y deja de ejecutar acciones
- Fallos por omisión: fallo en la recepción de un mensaje enviado desde un nodo
- Fallos transitorios
- Fallos bizantinos: fallos arbitrarios, un componente con un fallo de este tipo puede ofrecer resultados diferentes bajo las mismas entradas
- Fallos software

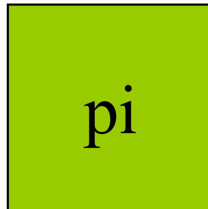
Detectores de fallos





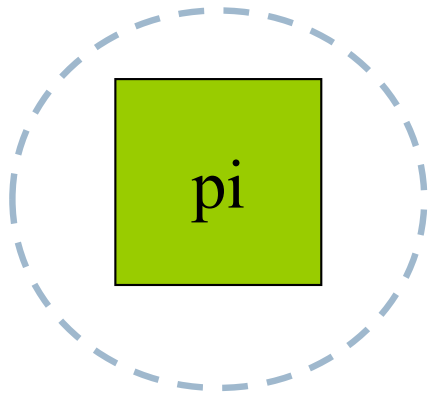
Detectores de fallos

Proceso pj falla



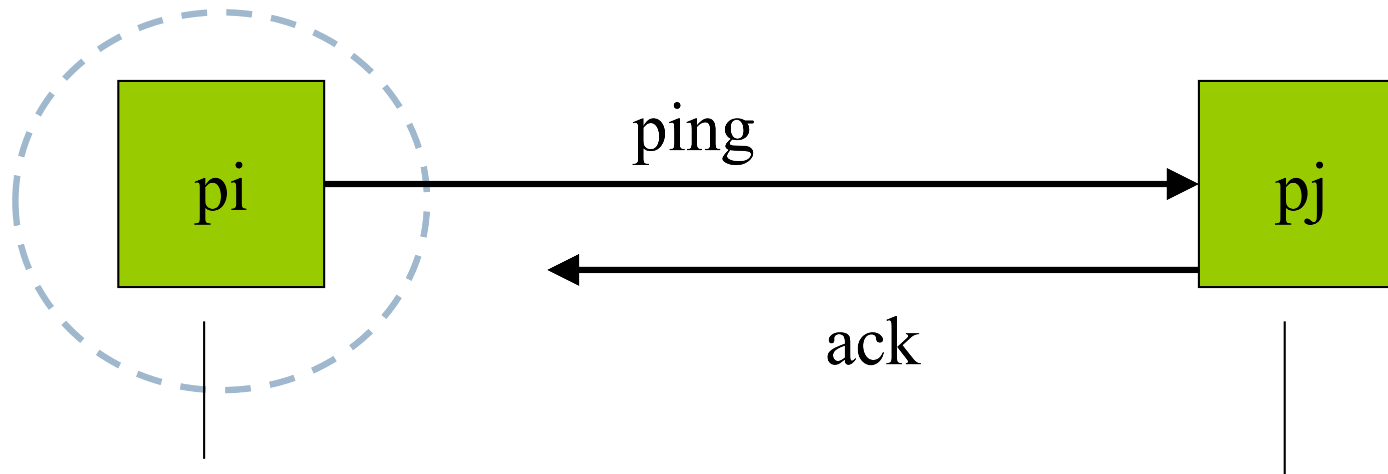
Detectores de fallos

Proceso pj falla



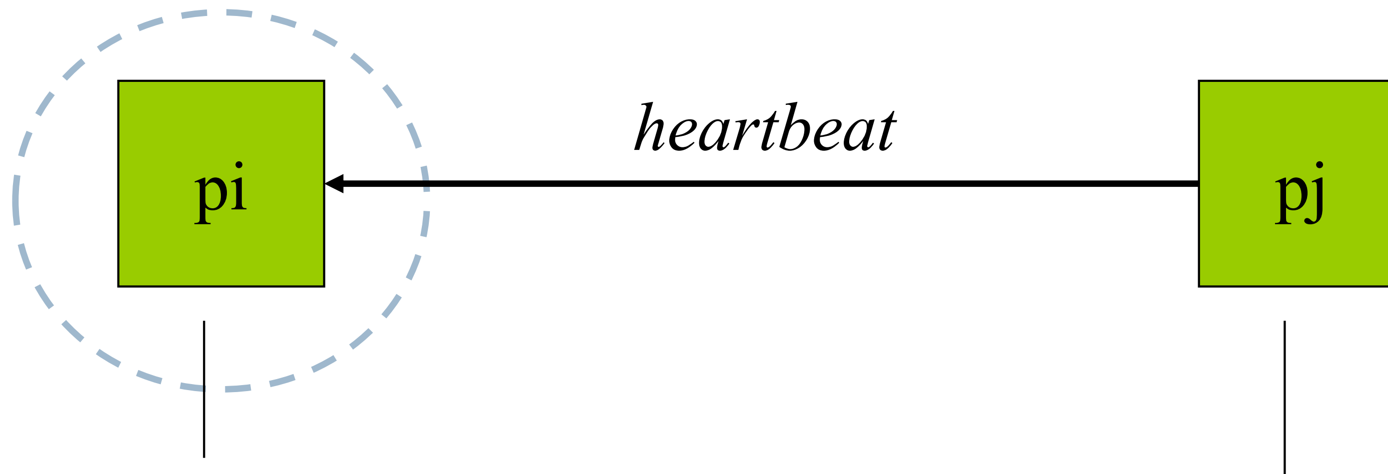
Pi es un proceso sin fallo que necesita conocer el estado de pj

Protocolo basado en ping



- De forma periódica pi interroga a pj

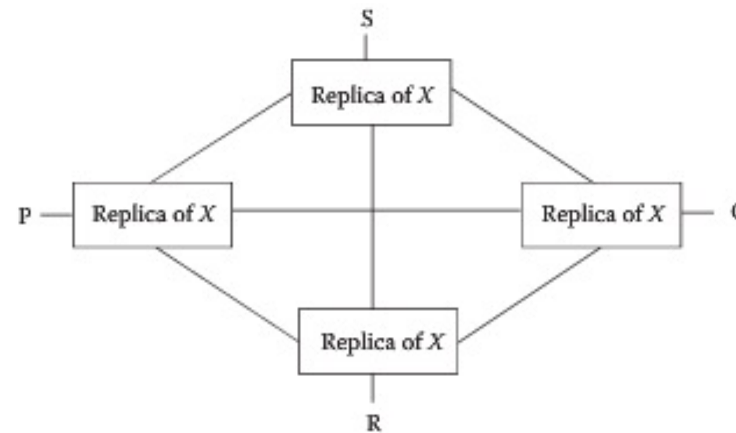
Protocolo basado en latido



- pj mantiene un número de secuencia
  pj envía a pi a latido (mensaje) con un nº de sec. incrementado cada T unidades de t.

Replicación

- Objetivos
  - Mejorar el rendimiento (caché)
  - Mejorar la disponibilidad
    - Si p es la probabilidad de fallo de un servidor
    - Con n servidores la probabilidad de fallo del sistema será p^n
- Tipos de replicación
  - De datos
  - De procesos
- Problemas que introduce
  - Consistencia
- Requisitos
  - Transparencia
  - Consistencia
  - Rendimiento



Ventajas de la replicación de datos

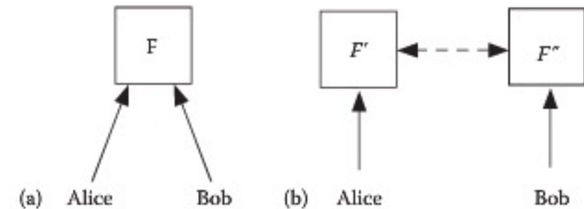
- Permite mantener los datos cercas de los usuarios
- Mejora la disponibilidad, el sistema puede continuar funcionando aunque algunas partes fallen
- Mejora la escalabilidad al permitir atender a más clientes

Geo-replicación

- Replicación entre centros conectados con redes de alta latencia
- Objetivos:
  - Clientes acceden a centros cercanos con baja latencia
  - Distribuye la carga
  - Escalabilidad: facilidad para atender a miles de usuarios
  - Disponibilidad, tolerancia a fallos
- Ejemplos:
  - Google, Facebook, Amazon Web servcies, Azure

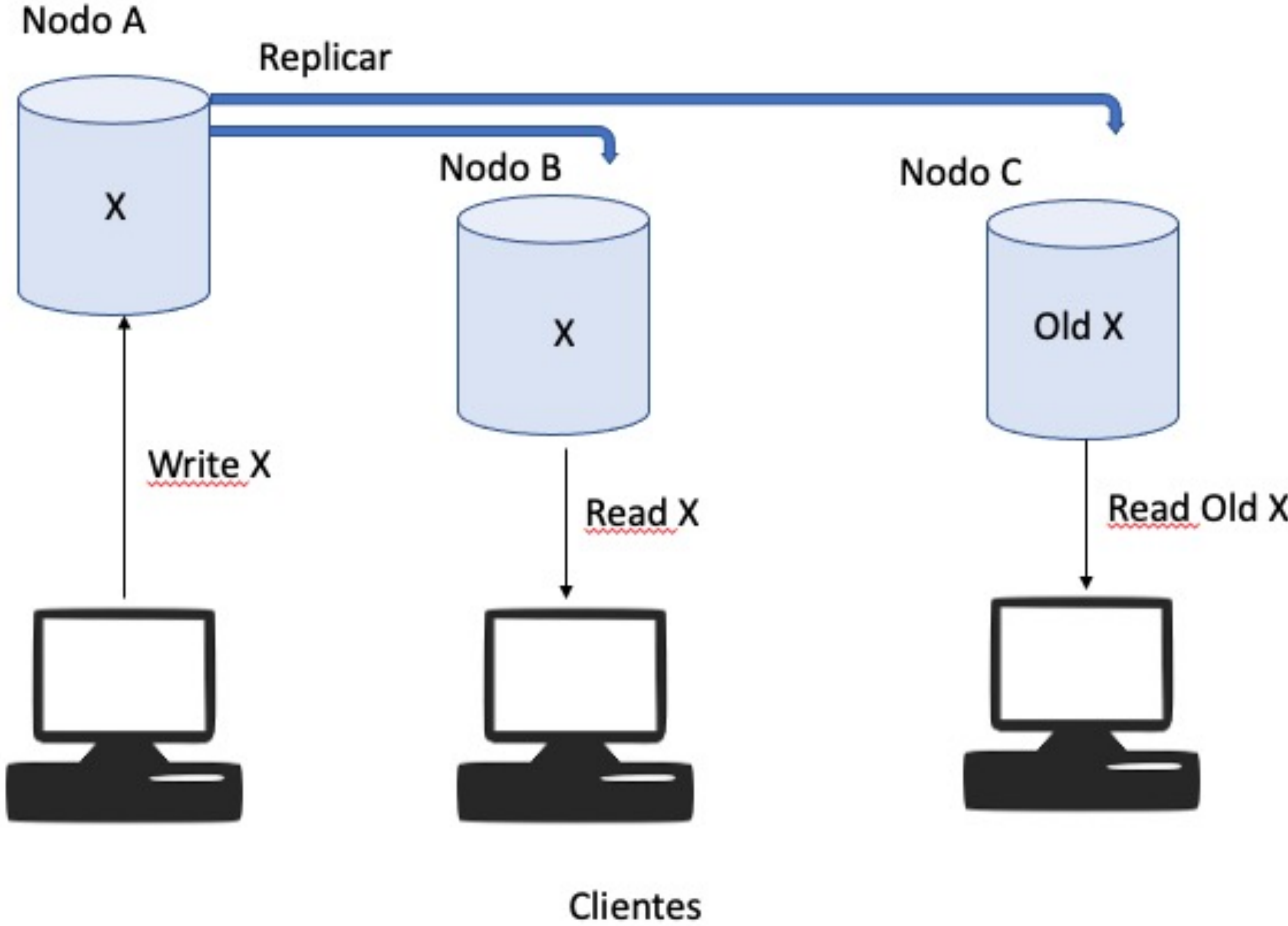
Problemas que introduce la replicación

- ¿Cómo mantener la consistencia de las réplicas?
  - En un esquema basado en replicación las réplicas pueden tener un estado inconsistente
  - Particiones de red
  - Caídas de nodos que gestionan réplicas



- Resolución de conflictos: procedimiento para reconciliar el estado de diferentes réplicas
  - Automático sin intervención manual
  - Intervención manual
- Modelos de consistencia de datos
  - Describen el comportamiento de las operaciones READ y WRITE sobre objetos replicados

Inconsistencias

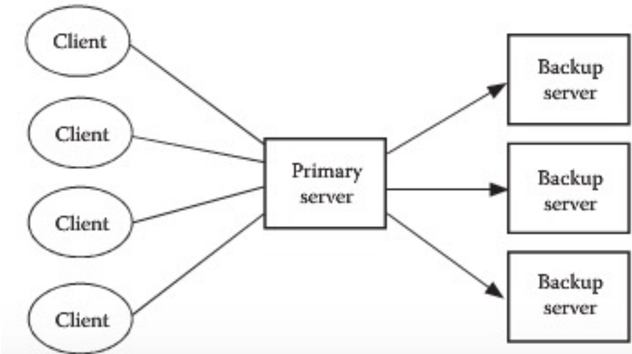


Métodos de replicación

- **Métodos pesimistas**: imponen restricciones en los accesos cuando hay particiones
  - Copia primaria
  - Réplicas activas
  - Esquemas de votación (quorum)
    - Estáticos
    - Dinámicos
- **Métodos optimistas**: no se imponen restricciones en los accesos cuando hay particiones
  - No imponen limitaciones
  - Ejemplo: sistemas basados en vectores de versiones

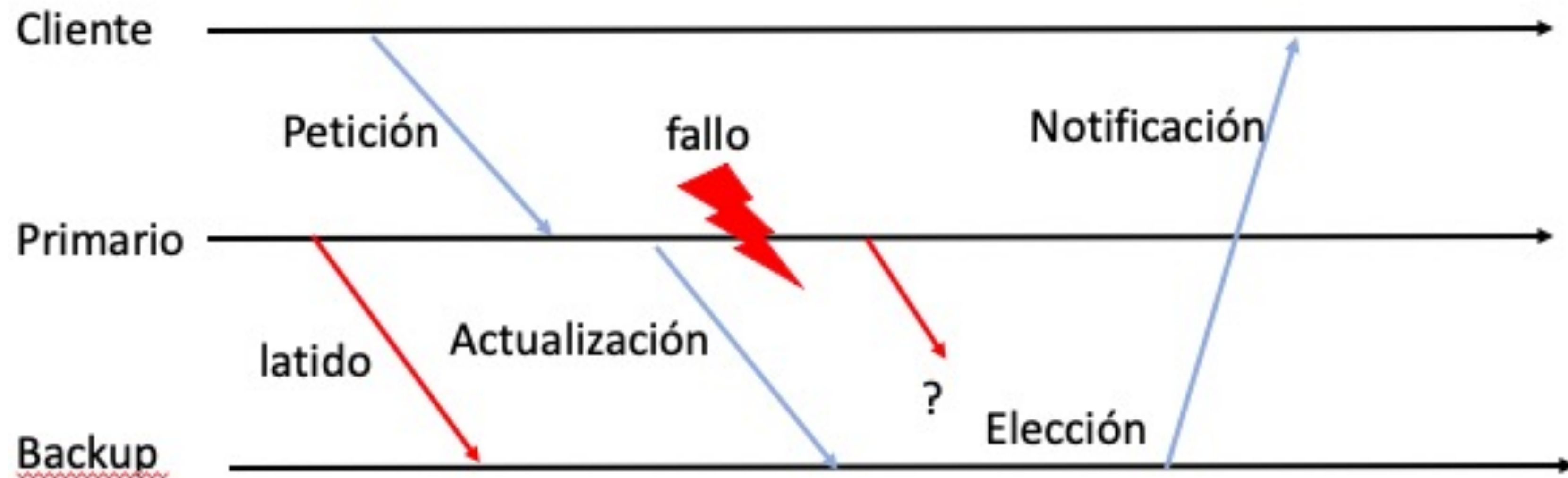
Copia primaria (replicación pasiva)

- Para hacer frente a k fallos, se necesitan k+1 copias
  - Un nodo primario
  - K nodos de respaldo



- Lecturas: se envían a cualquier servidor
- Escrituras: se envían al primario
  - El primario realiza la actualización y guarda el resultado
  - El primario actualiza el resto de copias
  - El primario responde al cliente
  - Las escrituras solo son atendidas por el nodo primario
- Cuando falla el primario un nodo secundario toma su papel (algoritmo de elección)

Implementación con mensajes *hearbeat*



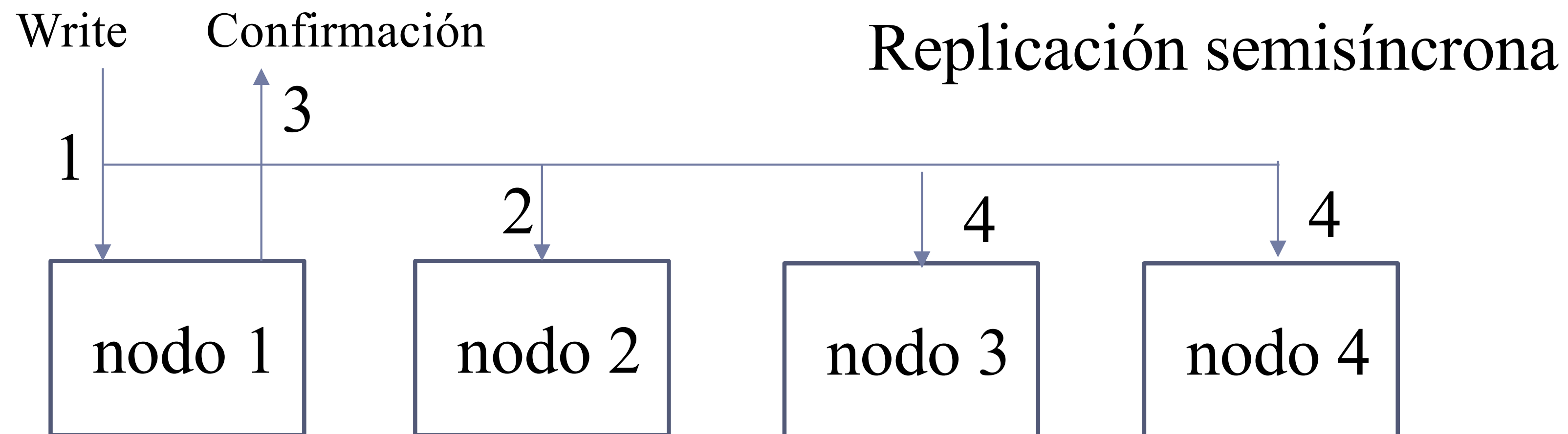
Sincronización de réplicas

Replicación síncrona

Write — 1 → nodo 1; 2 → nodo 2, nodo 3, nodo 4; 3 → Confirmación

Replicación asíncrona

Sincronización de réplicas



Fallo en un nodo secundario

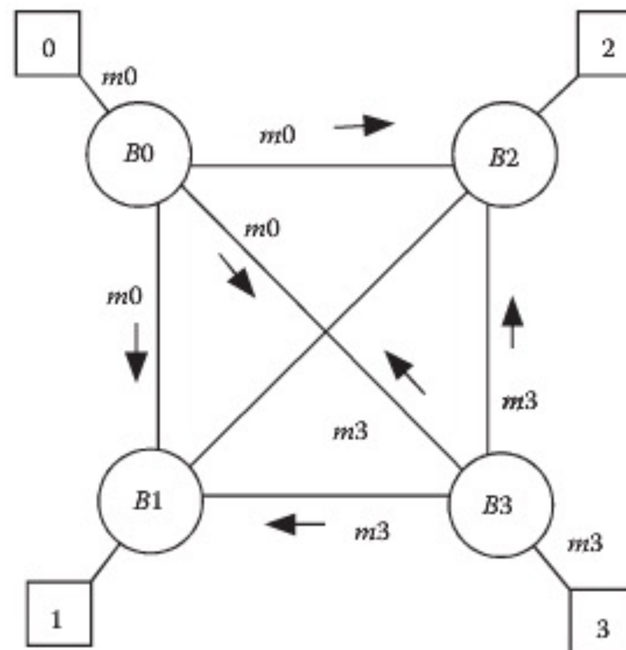
- El nodo primario mantiene en su disco local un registro de los últimos cambios realizados.
- Cuando un nodo secundario que ha fallado se reinicia contacta con el primario para obtener todos los cambios ocurridos en el último periodo

Fallo en el nodo primario

- Uno de los secundarios tiene que convertirse en primario:
  - Detectar el fallo en el nodo primario
  - Elegir un nuevo primario: algoritmo de elección
- Tienen que reconfigurarse los clientes para que envíen las nuevas escrituras al nuevo primario
- A este proceso se le denomina *failover*
- En modelos asíncronos se pueden perder las últimas escrituras realizadas

Réplicas activas

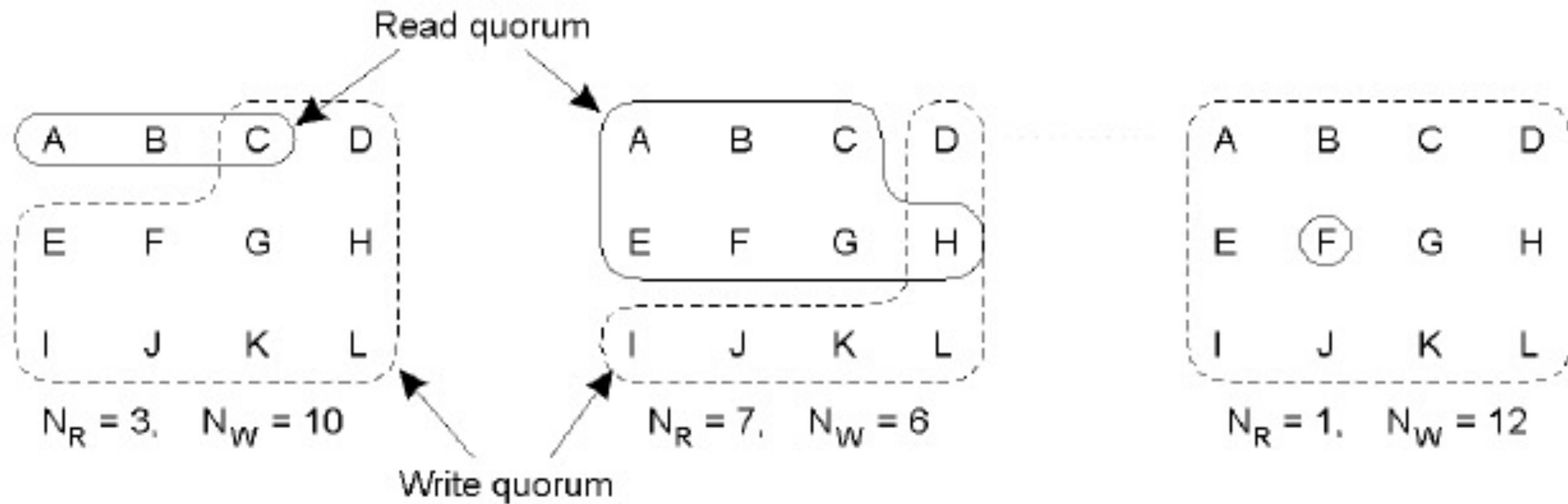
- Todas los nodos sirven peticiones
  - Mejor rendimiento en lecturas
- En escrituras se utiliza un multicast atómico
  - Se asegura el orden de las escrituras



Método de votación (quorum)

- Se definen dos operaciones READ y WRITE
- Hay un conjunto de N nodos, que sirven peticiones
  - Un READ debe realizarse sobre R copias
  - Un WRITE debe realizarse sobre W copias
  - Cada réplica tiene un número de versión V
  - Debe cumplirse que:
    - **R + W > N**
    - **W + W > N**
    - **R, W < N**

Ejemplos de quorums

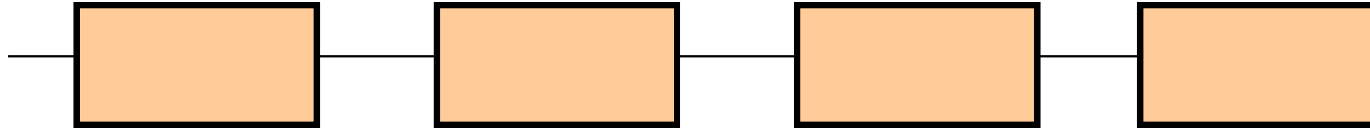


Quorum no válido

¿Cómo elegir W y R?

- Se analizan dos factores:
  - Rendimiento: depende del % de lecturas y escrituras y su coste
    - Coste total = coste L * P_R * R + coste E * (1-P_R) * W
  - Tolerancia a fallos: depende de la probabilidad con la que ocurren los fallos
    - Probabilidad fallo = Probabilidad de fallo L + Probabilidad de fallo E
- Ejemplo:
  - N=7
  - Coste de W = 2 veces el coste de R (K)
  - Porcentaje de lecturas (P_R) = 70%
  - Probabilidad de fallo = 0.05

Sistema serie

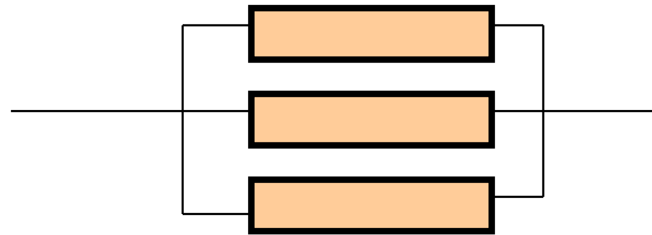


- Sea Ri(t) la fiabilidad del componente i
- El sistema falla cuando algún componente falla
- Si los fallos son independientes entonces

$$R(t) = \prod_{i=1}^{N} R_i(t)$$

- Se cumple que:
  - La fiabilidad del sistema es menor

Sistema paralelo

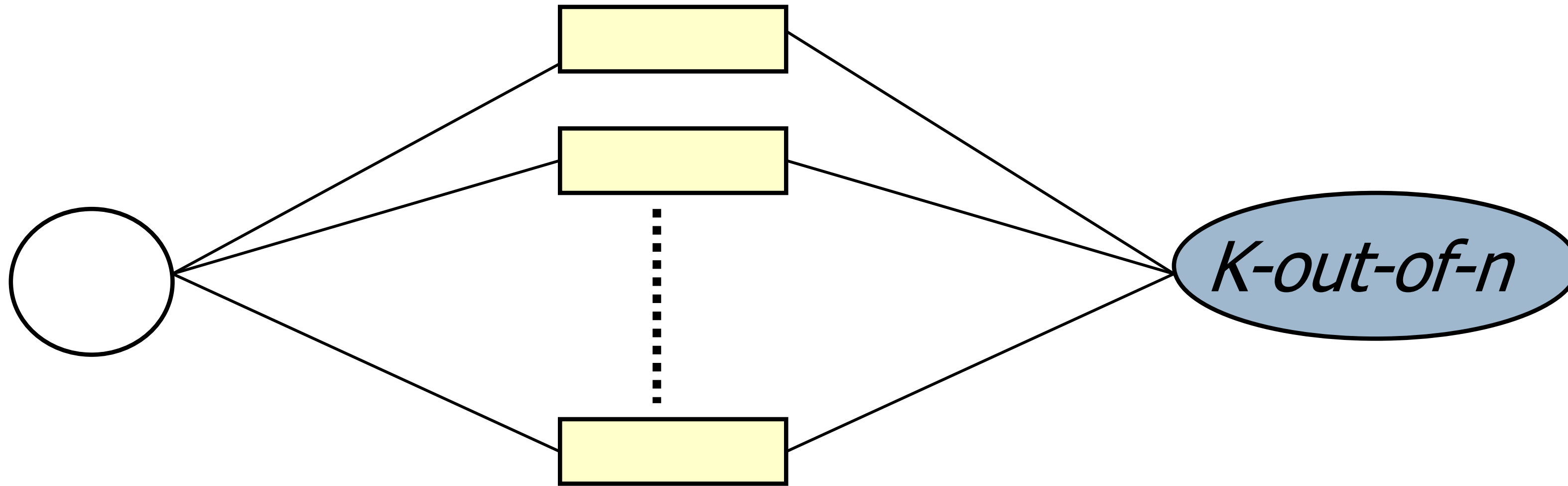


- El sistema falla cuando fallan todos los componentes

$$R(t) = 1 - \prod_{i=1}^{N} Q_i(t) \quad \text{donde} \quad Q_i(t) = 1 - R_i(t)$$

Fiabilidad de un sistema *k-out-of-n*

El sistema funciona cuando funcionan al menos k de n



$$R(k,n) = \sum_{r=k}^{r=n} \binom{n}{r} R^{r} (1-R)^{n-r}$$

Solución

Coste de W = 2 veces el coste de R (K)

Porcentaje de lecturas (P_R) = 70%

Probabilidad de fallo = 0.05

R	W	Coste	Probabilidad de fallo en R	Probabilidad de fallo en W	Probabilidad de fallo
1	7				
2	6				
3	5				
4	4				

Solución

Coste de W = 2 veces el coste de R (K)

Porcentaje de lecturas (P_R) = 70%

Probabilidad de fallo = 0.05

R	W	Coste	Probabilidad de fallo en R	Probabilidad de fallo en W	Probabilidad de fallo
1	7	4,9	$(0.05)^7$	$1-(0.95)^7$	9,05E-02
2	6	5	1-R(2,7)	1-R(6,7)	1,33E-02
3	5	5,1	1-R(3,7)	1-R(5,7)	1,13E-03
4	4	5,2	1-R(4,7)	1-R(4,7)	1,94E-04

¿y para P_R = 60%?

Coste = R*1 *0,7 + W * 2 *0.3=
0,7+2*6*0,3 =5

Operaciones en el método de votación

- Cada réplica tiene un número de versión V
- READ:
  - Se lee de R réplicas (X_i, V_i), se queda con la copia que tiene la versión V_i mayor
- WRITE:
  - Se realiza en primer lugar una operación READ para determinar el número de versión actual (V)
  - Se calcula el nuevo número de versión (V = V + 1)
  - Se actualiza de forma atómica W réplicas con el nuevo valor y número de versión
    - Se inicia un protocolo 2PC para actualizar el valor y el número de versión en W réplicas.

Two-phase commit

- Two-phase-commit (2PC)
- Denominamos coordinador al proceso que realiza la operación

Coordinador:

multicast: *ok to commit?*

recoger las respuestas

todos ok => *send(commit)*

else => *send(abort)*

Procesos:

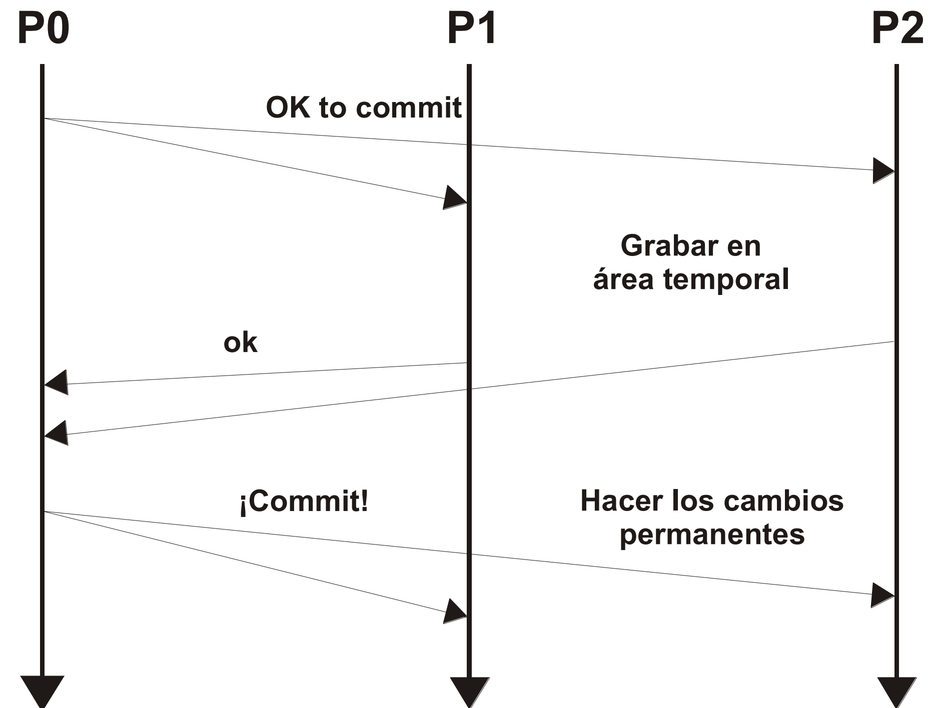
ok to commit=> guardar

la petición en un

área temporal y responder *ok*

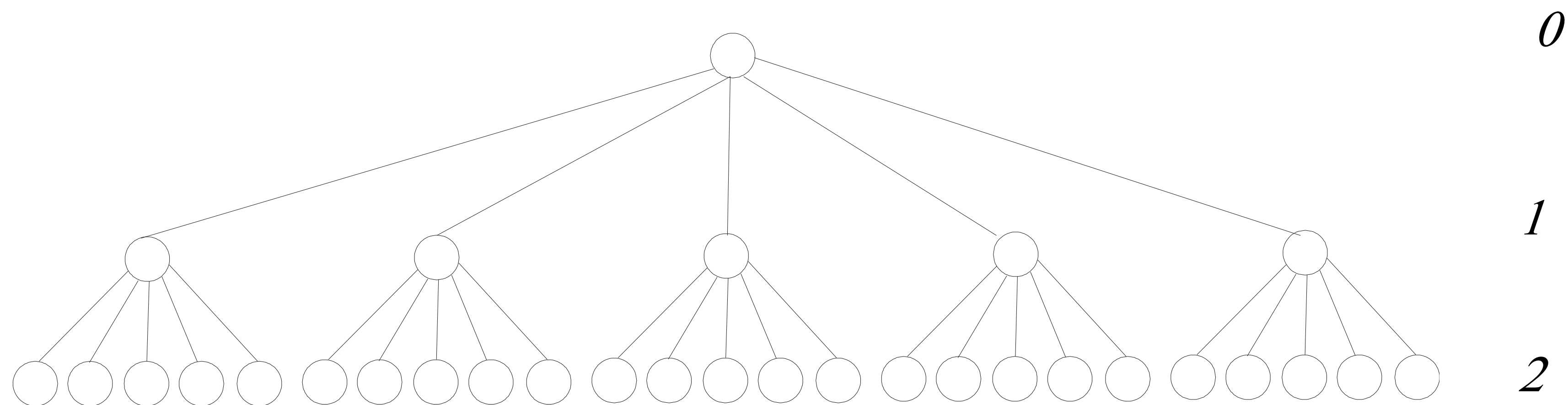
commit => hacer los cambios permanentes

abort => borrar los datos temporales



Votación jerárquica

- El problema del método anterior es que W aumenta con el número de réplicas
- Solución: **quorum jerárquico**
  - Ej: número de réplicas = 5 x 5 = 25 (nodos hoja)



Votación jerárquica

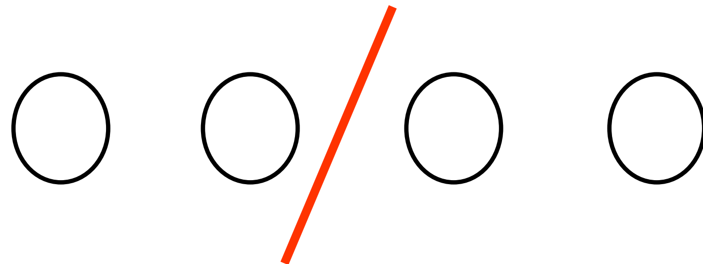
- Se realiza el **quorum** por niveles

R1	W1	R2	W2	RT	WT
1	5	1	5	1	25
1	5	2	4	2	20
1	5	3	3	3	15
2	4	2	4	4	16
2	4	3	3	6	12
3	3	3	3	9	9

¿Cuál se elige?

Métodos adaptativos dinámicos

- Los métodos anteriores (estáticos) no se adaptan a los cambios que ocurren cuando hay fallos
- Ejemplo:
  - Dado un esquema de votación para 4 réplicas con
    - R=2 y W=3
  - Si se produce una partición



  - No se pueden realizar escrituras

Método de votación dinámica

- Cada dato d está soportado por *N* réplicas $\{d1..dn\}$
- Cada dato di en el nodo i tiene un número de versión VNi (inicialmente 0)
- Se denomina NV actual $NVA(d) = \max\{VNi\} \quad \forall\, i$
- Una réplica di es actual si $VNi = VNA$
- Un grupo constituye una partición mayoritaria si contiene una mayoría de copias actuales de *d*
- Cada copia di tiene asociado un número entero denominado cardinalidad de actualizaciones SCi = número de nodos que participaron en la actualización

Método de votación dinámica

- Inicialmente $SC_i = N$
- Cuando se actualiza d_i
  - SC_i = nº de copias de d modificadas durante esta actualización
- Un nodo puede realizar una actualización si pertenece a una partición mayoritaria

Algoritmo de escritura

$\forall$ *i* accesible solicita *NVi* y *SCi*

M = max{*NVi*} incluido él

I = {*i* tal que *VNi* = *M*}

N = max{*SCi*, $i \in I$}

if $|I| \leq N/2$

then

el nodo no pertenece a una partición mayoritaria, se rechaza la operación

else {

$\forall$ nodos $\in$ *I*

Actualizar

VNi = *M*+1

SCi = $/I/$

}

Ejemplo

- N= 5
- Inicialmente:

	A	B	C	D	E
VN	9	9	9	9	9
SC	5	5	5	5	5

- Ocurre una partición:

	A	B	C	D	E
VN	9	9	9	9	9
SC	5	5	5	5	5

Ejemplo

- ¿Escritura en partición 2?
  - M= max{9, 9} = 9
  - I={D, E}
  - N= 5 , $|I|$ = 2 $\leq$ 5/2 $\Rightarrow$ No se puede realizar

	D	E
	9	9
	5	5

- ¿Escritura en partición 1?
  - M= max{9, 9, 9} = 9
  - I={A, B, C}
  - N=5
  - $|I|$ = 3 > 5/2 $\Rightarrow$ Se puede actualizar

	A	B	C
VN	9	9	9
SC	5	5	5

	A	B	C	D	E
VN	10	10	10	9	9
SC	3	3	3	5	5

Ejemplo

- Nueva partición

	A	B	C	D	E
VN	10	10	10	9	9
SC	3	3	3	5	5

Partición 1 (A, B) | Partición 2 (C) | Partición 3 (D, E)

- ¿Escritura en partición 1?
  - N=max{10,10} = 10
  - I = {A, B}
  - N= 3
  - $|I|$ = 2 > 3/2 $\Rightarrow$ Se puede actualizar

Ejemplo

	A	B	C	D	E
VN	11	11	10	9	9
SC	2	2	3	5	5

Partición 1 Partición 2 Partición 3

Unión de un nodo a un grupo

- Cuando un nodo se une a un grupo tiene que actualizar su estado:

  $M = \max\{VNi\}$

  $I = \{Aj, \text{tal que } M = VNj\}$

  $N = \max\{SCk, k \in I\}$

  if $|I| \leq N/2$

  then

  no se puede unir

  else {

  Actualiza su estado

  $VNi = M$

  $SCi = N + 1$

  }

Ejemplo

- Se une la partición 2 y 3

	A	B	C	D	E
VN	11	11	10	9	9
SC	2	2	3	5	5

Partición 1 Partición 2

- ¿Se puede unir C a la partición 2?
  - M= max{10, 9, 9} = 10
  - I={C}
  - N=3
  - $|I| = 1 \leq 3/2 \Rightarrow$ se rechaza, no se puede unir

Ejemplo

- Se une la partición 1 y 2

	A	B	C	D	E
VN	11	11	10	9	9
SC	2	2	3	5	5

Partición 1 — Partición 2

- ¿Se puede unir C a la partición 1?
  - M= max{11, 11, 10} = 11
  - I={A,B}
  - N=2
  - $|I| = 2 > 2/2 \Rightarrow$ Se puede unir, se actualiza

Ejemplo

	A	B	C	D	E
VN	11	11	11	9	9
SC	3	3	3	5	5

Partición 1 Partición 2

Replicación basada en vectores de versiones

- Método de replicación **optimista**
- Cada réplica lleva asociado un vector de versiones V con n componentes = grado de replicación
- En el nodo *i*, *Vi[j]* representa el número de actualizaciones realizadas en la réplica de *j*
- Cuando no hay fallos de red todos los vectores son iguales en todas las réplicas
- Cuando hay fallos de red los vectores difieren
- Dados *V1* y *V2*, ***V1* domina a *V2*** sii $V1(i) \geq V2(i)\ \forall\ i$
- Si *V1* domina a *V2* hay más actualizaciones en la copia de *V1*
- *V1* y *V2* **están en conflicto** si ninguno domina al otro

Replicación del sistema de ficheros CODA

- Cuando dos grupos se juntan
  - Se comparan los vectores
  - Si el vector de un grupo domina al vector del otro se copia la copia del primero en el segundo
  - Si hay conflictos el archivo se marca como *inoperable* y se informa al propietario para que resuelva el conflicto.

Ejemplo

- Tres servidores = {A, B, C}
- Inicialmente *V* = (0,0,0) en los tres
- Cuando se realiza una actualización: *V*=(1,1,1) en los tres servidores
- Se produce un fallo de red:
  - Grupo 1: {A,B}
  - Grupo 2: {C}
- Se produce una actualización sobre el grupo 1
  - *V*=(2,2,1) para el grupo 1
- Se produce un fallo de red:
  - Grupo 1: {A}, *V*=(2,2,1)
  - Grupo 2: {B, C}
    - (2,2,1) ≥ (1,1,1) ⇒ se actualiza la copia de C y *V* = (2,2,2) en B y C

Ejemplo

- Se produce una actualización sobre el grupo 2
  - *V*=(2,3,3) en {B,C}
- Situación 1: se une {A} a {B,C}
  - (2,2,1) $\leq$ (2,3,3) $\Rightarrow$ se actualiza la copia de {A} y *V* =(3,3,3)
- Situación 2:
  - Se modifica la versión de {A} $\Rightarrow$ en A, *V*= (3,2,1)
  - Se une A con *V*=(3,2,1) a {B,C} con *V*=(2,3,2)
  - Se comparan (3,2,1) y (2,3,2), ninguno domina $\Rightarrow$ **conflicto**

El teorema CAP

Brewer, PODC 2000