

Grupo ARCOS



**uc3m** | Universidad **Carlos III** de Madrid

# Tema 9: Tolerancia a fallos

## **Sistemas Distribuidos**



Grado en Ingeniería Informática  
Grado en Matemática aplicada y Computación  
Doble Grado en Ingeniería Informática y Administración de Empresas

# Ejemplos de sistemas que precisan ser tolerantes a fallos

- ▶ Sistemas con una vida larga.
- ▶ Sistemas de difícil mantenimiento:
  - ▶ Satélites, cohetes, etc.
- ▶ Aplicaciones críticas:
  - ▶ Aviones, telemedicina, etc.
- ▶ Sistemas de alta disponibilidad:
  - ▶ Sistemas bancarios, etc.

# Contenidos

1. Introducción a la tolerancia a fallos
2. Tolerancia a fallos software
3. Tolerancia a fallos en sistemas distribuidos

# Contenidos

1. **Introducción a la tolerancia a fallos**
2. Tolerancia a fallos software
3. 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**

# 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

# 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**



- 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.

# 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**



- La **fiabilidad** (*reliability*) de un sistema como medida global o como función de la fiabilidad de cada componente del sistema:
  - Analizar cada componente: **tipo de fallos + fiabilidad + impacto.**
  - Aplicar **técnicas para aumentar la fiabilidad.**



# 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



Origen de los fallos

- **Fallos hardware**
  - Fallos {permanentes o transitorios} x {componentes hardware o subsistemas de comunicación}
- **Fallos software**
  - **Especificación** inadecuada
  - Fallos introducidos por errores en **diseño y programación** de componentes software

# 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**

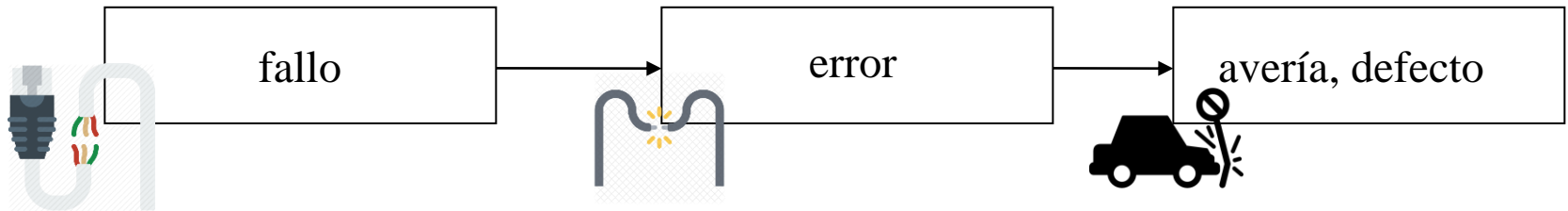
## ► Objetivo

- Conseguir que un sistema sea altamente

Fiabilidad + impacto

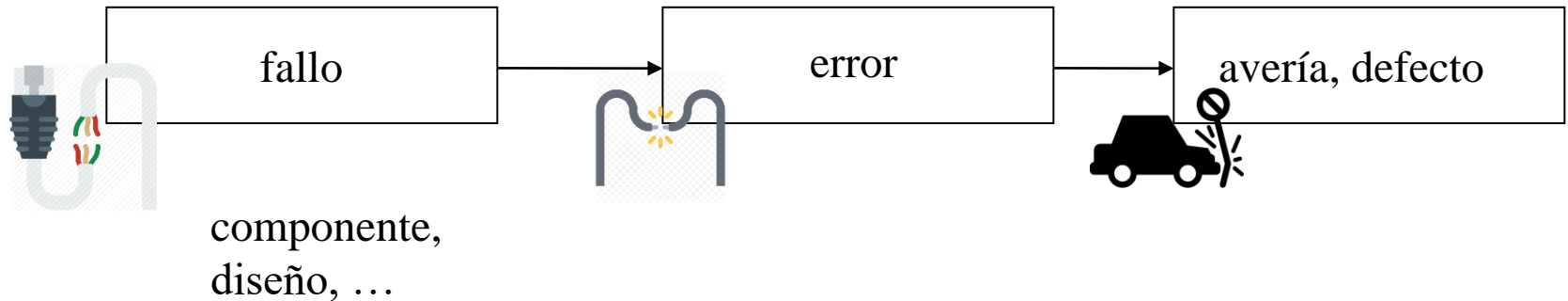


# Conceptos básicos



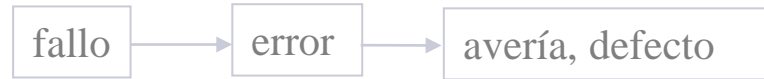
- ▶ Los **fallos** (*faults*)
  - ▶ Son las **causas mecánicas/algorítmicas de los errores**.
  - ▶ Pueden ser consecuencias de averías en los componentes del sistema.
- ▶ Un **error** (*errors*)
  - ▶ Se manifiesta dentro de los **valores internos del estado** del sistema como valores **distintos a los deseados**.
- ▶ Una **avería o defecto** (*failure*)
  - ▶ Es una **desviación** del comportamiento de un sistema **respecto de su especificación**.
  - ▶ Se manifiesta en el comportamiento externo del sistema, pero son el resultado de **errores internos**.

# Conceptos básicos



- ▶ Los fallos pueden ser pequeños, pero los defectos muy grandes (tener un gran impacto):
  - ▶ 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 (1/3)

- ▶ Explosión del Ariane 5 en 1996
  - ▶ Enviado por la ESA en junio de 1996 (fue su primer viaje)
  - ▶ Coste del desarrollo: 10 años y 7 000 millones de dólares.
  - ▶ Explotó 40 seg. después del despegue a 3 700 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 32 767 (el mayor entero con signo de 16 bits) y se produjo un fallo de conversión y una excepción.

# Ejemplos de fallos (2/3)

- ▶ 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 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 pérdida de precisión correspondiente.
  - ▶ Esta pérdida de precisión es la que provoca un fallo en la interceptación

# Ejemplos de fallos (3/3)

- ▶ 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



# Más ejemplos de fallos...

## ► Historias sobre fallos en:

- <http://www.cs.tau.ac.il/~nachumd/verify/horror.html>
- <https://rollbar.com/blog/10-developer-horror-stories-to-keep-you-up-at-night/#>
- <https://worthwhile.com/blog/2017/03/09/software-development-horror-stories/>
- ...

# 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

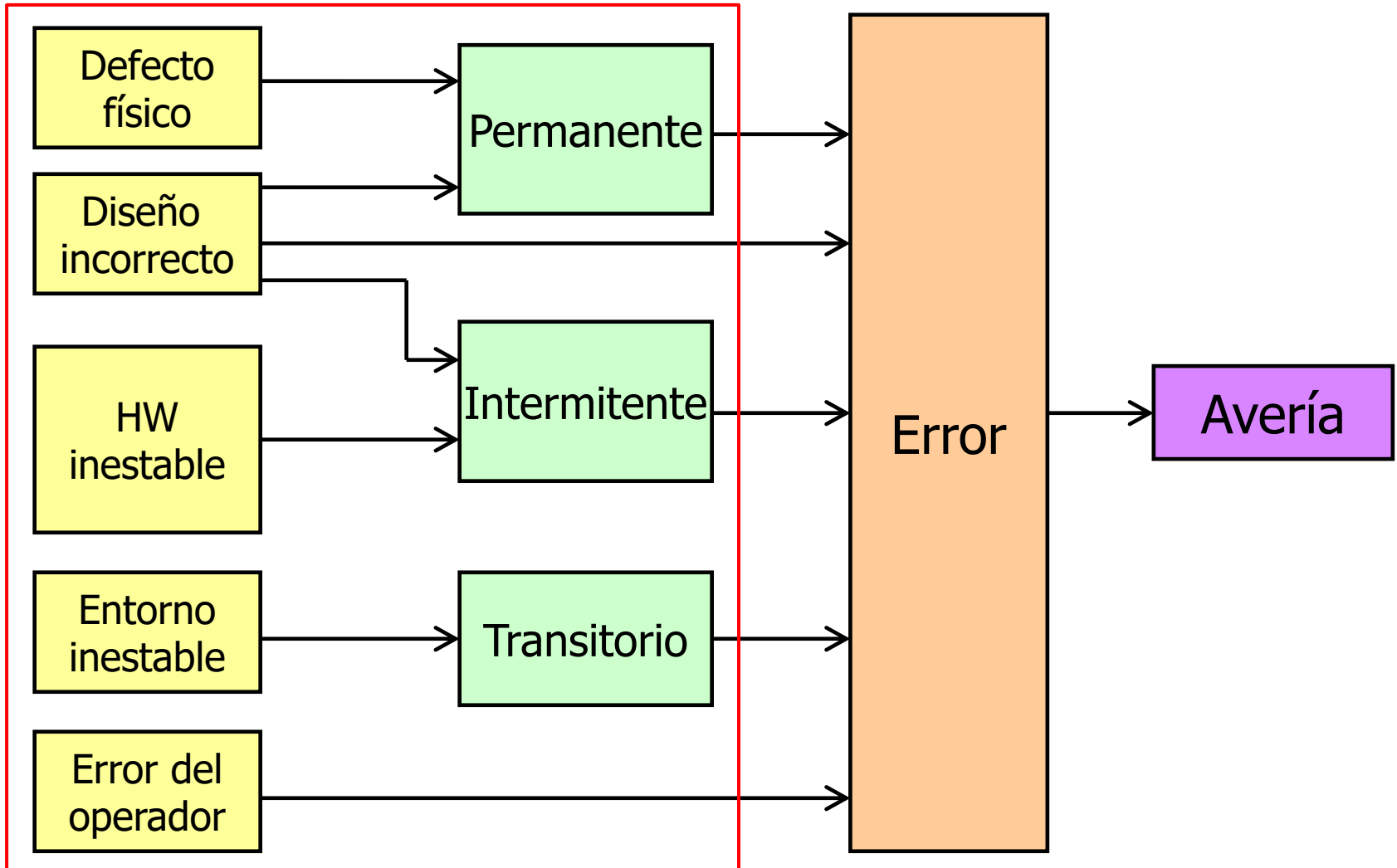
- ▶ Desaparecen solos al cabo de un cierto tiempo.
- ▶ **Ejemplo:** interferencias en comunicaciones, fallos transitorios en los enlaces de comunicación.

## ▶ 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



# 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

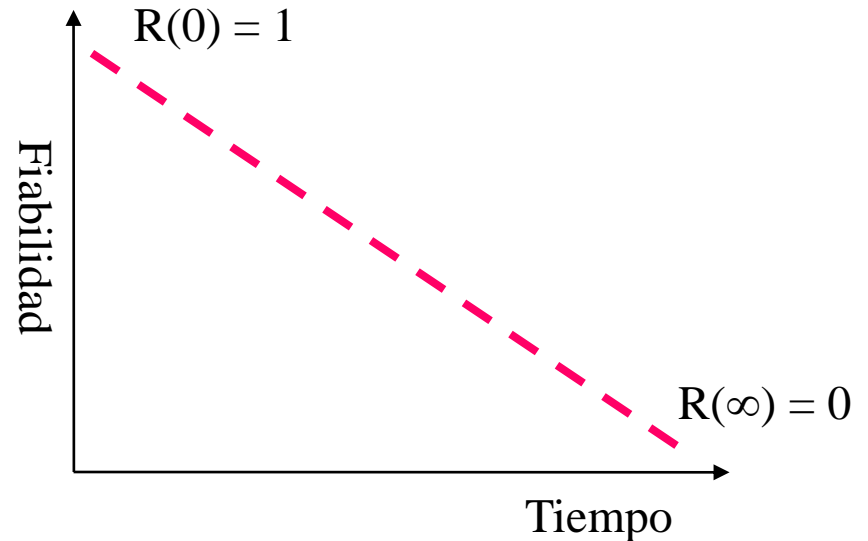
- Conseguir que un sistema sea altamente **fiable**



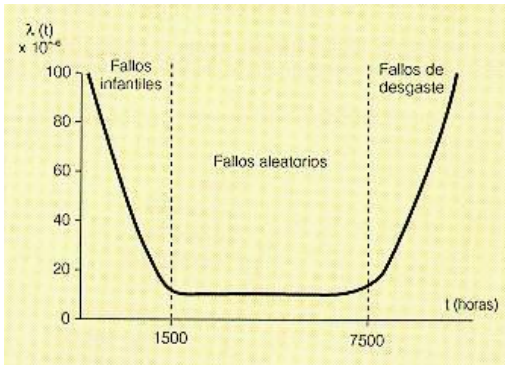
- La **fiabilidad** (*reliability*) de un sistema **es una medida** de su conformidad con una especificación autorizada de su comportamiento.
- Un sistema es **fiable** si cumple sus especificaciones.

# Fiabilidad

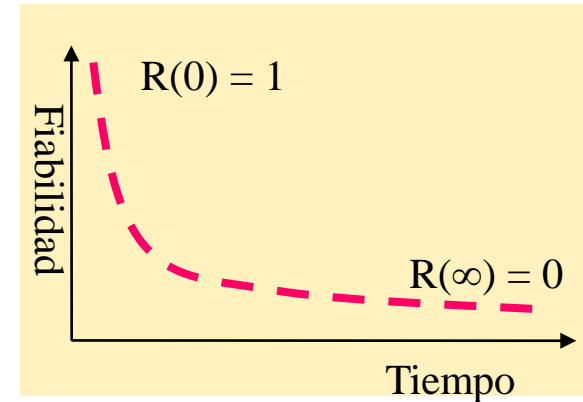
- El tiempo de vida de un sistema se representa mediante una variable aleatoria  $X$
- Se define la **fiabilidad** del sistema como una función  $R(t)$ 
  - $R(t) = P(X > t)$
  - De forma que:
    - $R(0) = 1$  y  $R(\infty) = 0$



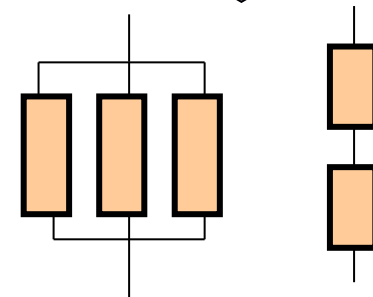
# Fiabilidad de un sistema a partir de la fiabilidad de sus componentes...



**1. Modelar fiabilidad de componente i**

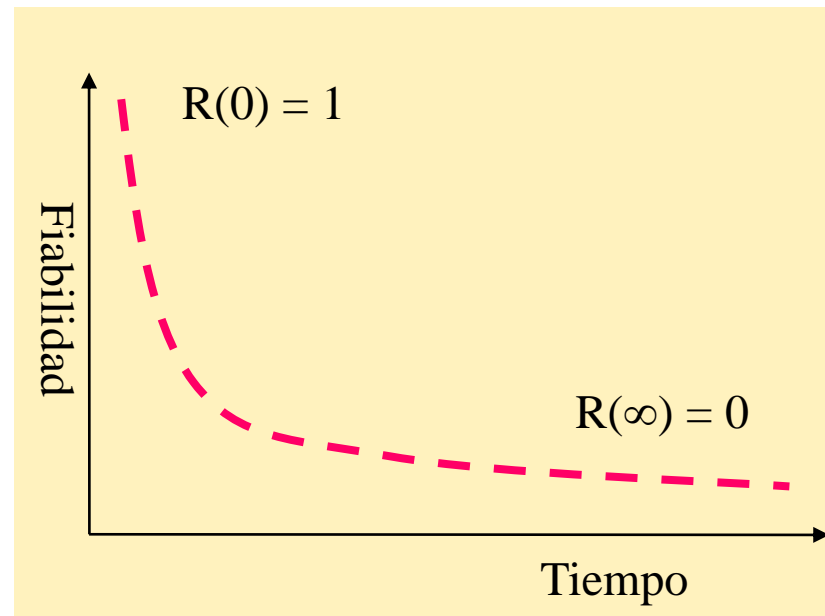
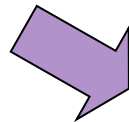
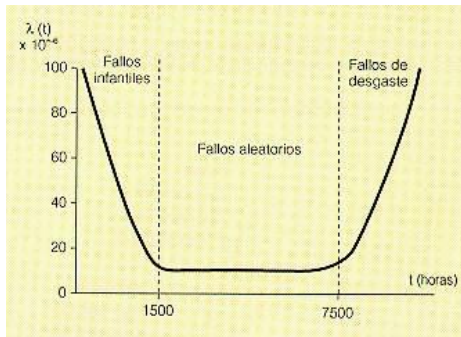


**2. Componer fiabilidad (serie | paralelo | k-de-n)**



# Fiabilidad: modelar (1/2)

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



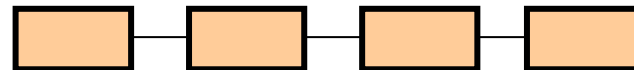
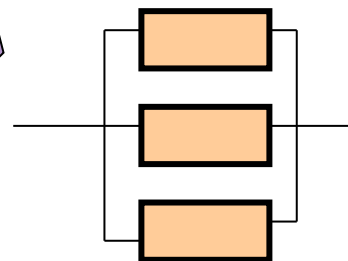
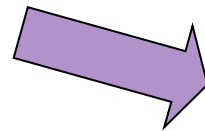
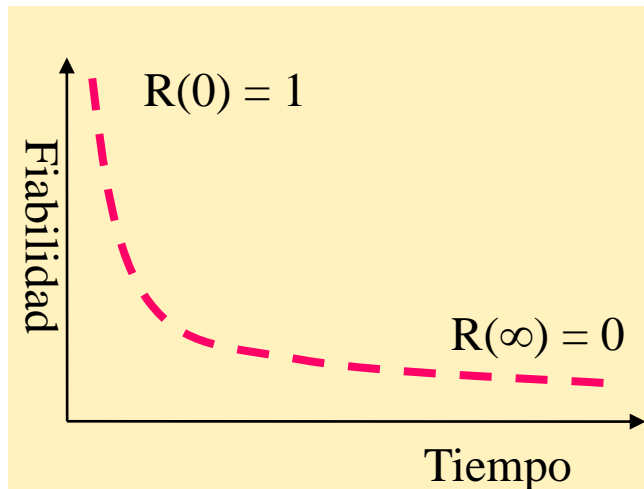
# Ejemplos de distribuciones

Nombre	Descripción	Gráfica
Exponencial	Usada si la tasa de errores es constante (generalmente verdadero para componentes electrónicos)	<p> <math>\lambda = 0.5</math>  <math>f(t) = \lambda \cdot \exp^{-(\lambda \cdot t)}</math>  <math>MTTF = 20</math> </p>
Normal	Usada para describir los equipos con una tasa de errores que se incrementa con el paso del tiempo	<p> <math>f(t) = \left( \frac{1}{\sigma \sqrt{2 \cdot \pi}} \right) \exp \left[ \frac{-1}{2} \left( \frac{t - \mu}{\sigma} \right)^2 \right]</math>  <math>Mean = \mu = 20.0</math>  <math>StdDev = \sigma = 8.5</math>  <math>MTTF = 20.0</math> </p>
Normal logarítmica	Se encuentra cuando los tiempos de fallo o reparación dependen de factores que contribuyen de forma acumulativa (fatiga)	<p> <math>f(t) = \left[ \frac{1}{(\sqrt{2 \cdot \pi}) \cdot \sigma \cdot t} \right] \exp \left[ \frac{-1}{2} \left( \frac{\ln(t) - \mu}{\sigma} \right)^2 \right]</math>  <math>Mean = \mu = 3.60</math>  <math>StdDev = \sigma = 0.89</math>  <math>MTTF = 20.0</math> </p>
Weibull	Vida característica $\eta$ (tiempo en el que el 63,2% de población falla) y factor de forma $\beta$ (asociado a la tasa de error, siendo $b=1 \rightarrow$ tasa de error constante)	<p> <math>f(t) = \left( \frac{\beta}{\eta} \right) \left( \frac{t}{\eta} \right)^{\beta-1} \exp \left[ - \left( \frac{t}{\eta} \right)^\beta \right]</math>  <math>CharLife = \eta = 22.2</math>  <math>ShpFct = \beta = 1.5</math>  <math>MTTF = 20.0</math> </p>

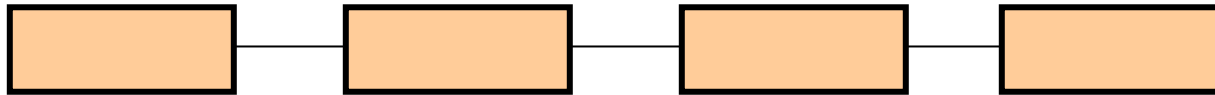


# Fiabilidad: componer (2/2)

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



# Sistema serie



- ▶ El sistema falla cuando algún componente falla

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

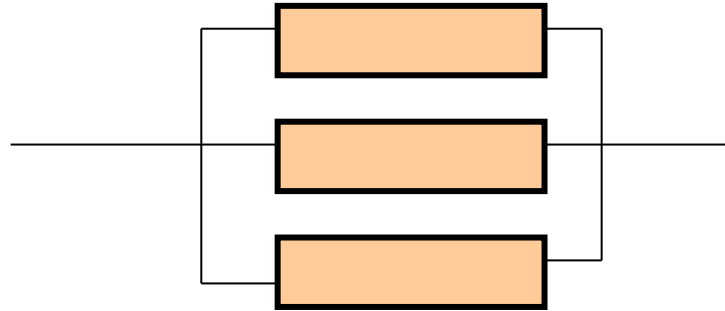
- ▶ Sea  $R_i(t)$  la fiabilidad del componente  $i$
- ▶ Son fallos independientes (entre sí)

- ▶ Se cumple que:

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

- ▶ La fiabilidad del sistema es menor

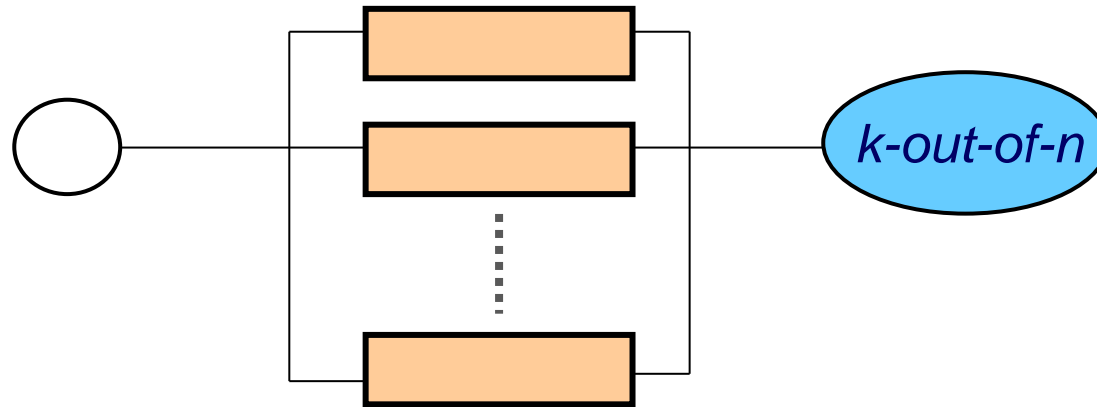
# Sistema paralelo



- El sistema falla cuando fallan todos los componentes

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

# Sistema *k-out-of-n*

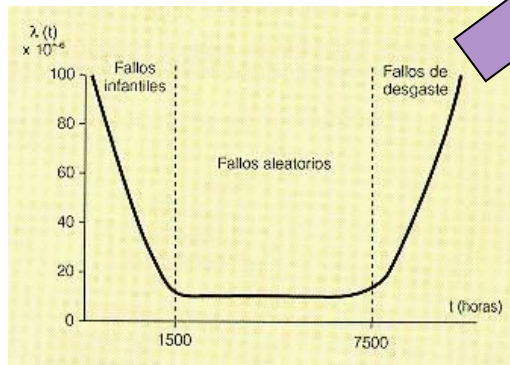
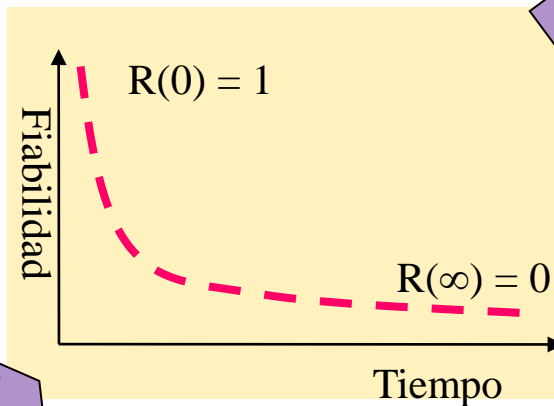
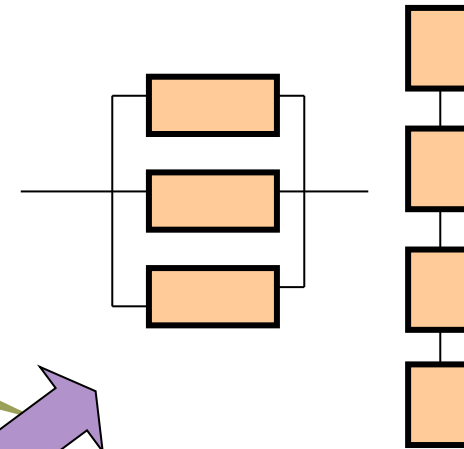


- El sistema funciona cuando funcionan al menos k de n:

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

# Fiabilidad: resumen

## 2. Componer fiabilidad (serie | paralelo | k-de-n)



## 1. Modelar fiabilidad de componente

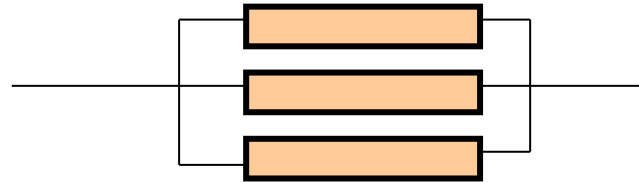
# Ejemplo



$$R_i(t) = 0.9$$



$$R(t) = 0.9 * 0.9 * 0.9 = 0.729$$



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

# Ejemplo

- ▶ ¿Cuál es la fiabilidad de un RAID0 con 5 discos, si la fiabilidad de cada disco es  $R=0.99$ ?
- ▶ ¿Cuál es la fiabilidad de un RAID4 con 5 discos, si la fiabilidad de cada disco es  $R=0.99$ ?

# Ejemplo

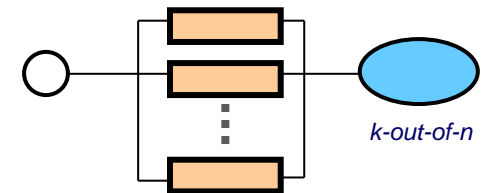
- ▶ ¿Cuál es la fiabilidad de un RAID0 con 5 discos, si la fiabilidad de cada disco es  $R=0.99$ ?



$$R = R_i^5 = 0.9039$$

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

$$R(4,5) = \sum_{r=4}^{r=5} \binom{5}{r} R_i^r (1 - R_i)^{5-r} = 0.99989$$





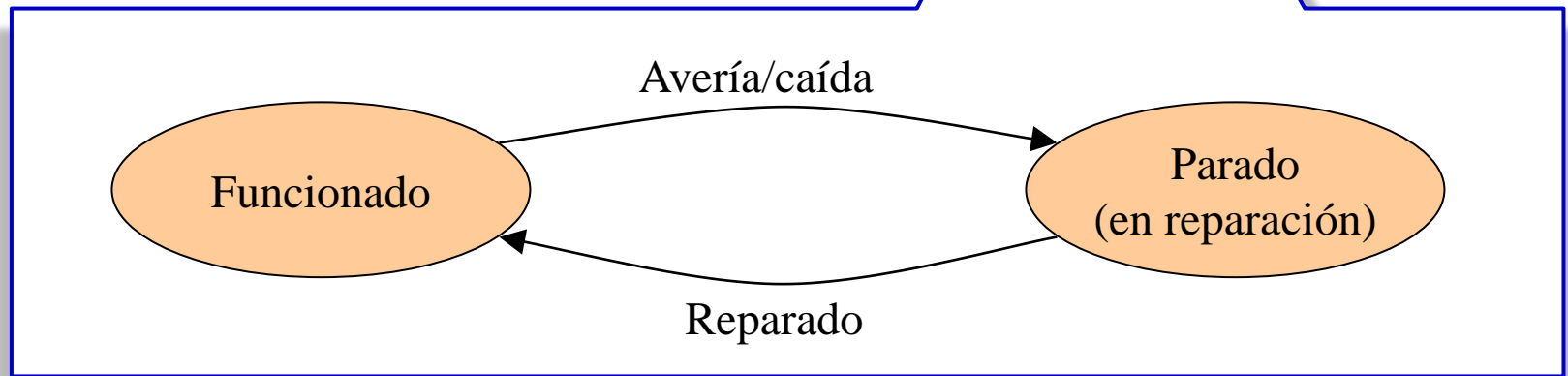
# Tolerancia a fallos

## ► 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**

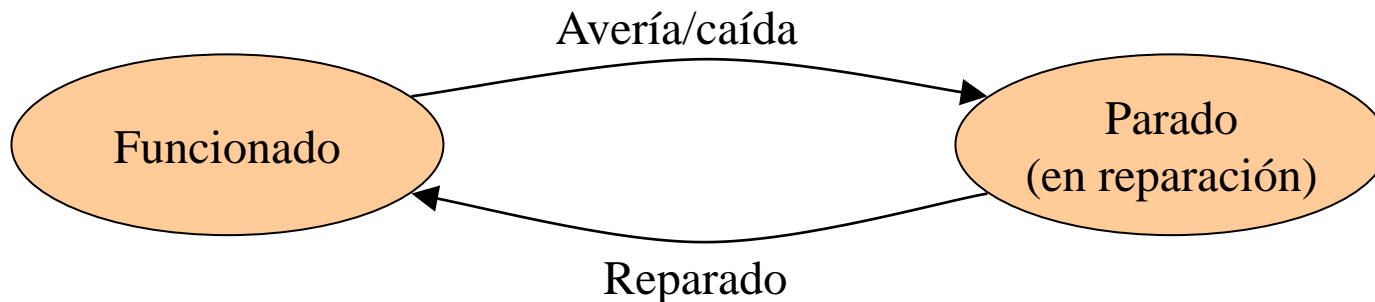
## ► Objetivo

- Conseguir que un sistema sea altamente



# Disponibilidad

- ▶ En muchos casos es más interesantes conocer la disponibilidad
- ▶ Se define la **disponibilidad de un sistema  $A(t)$**  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:



# Medida de la disponibilidad

- ▶ Sea T<sub>MF</sub> el tiempo medio hasta el fallo
- ▶ Sea T<sub>MR</sub> el tiempo medio de reparación
- ▶ Se define la disponibilidad de un sistema como:

$$\text{Disponibilidad} = \frac{T_{MF}}{T_{MF} + T_{MR}}$$

- ▶ ¿Qué significa una **disponibilidad del 99%**?
  - ▶ En **365** días funciona correctamente  $99 \cdot 365 / 100 = 361,3$  días
  - ▶ Está sin servicio **3,65** días

# Tiempo anual sin servicio

<b>Disponibilidad (%)</b>	<b>Tiempo sin servicio <u>al año</u></b>
98%	7,3 días
99%	3,65 días
99.8%	17 horas y 30 minutos
99.9%	8 horas y 45 minutos
99.99%	52 minutos y 30 segundos
99.999%	5 minutos y 15 segundos
99.9999%	31,5 segundos

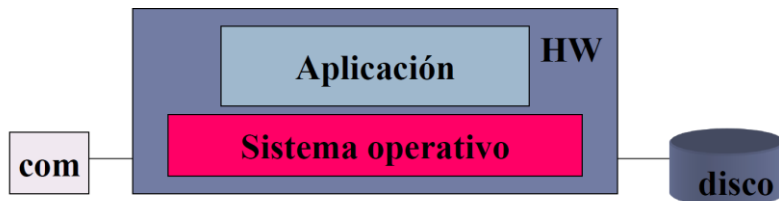
# Tipos de paradas

- ▶ **Mantenimiento correctivo:**
  - ▶ Debido a fallos (reactivo)
  - ▶ No planificados (normalmente)
  - ▶ Ej.: cambiar las bombillas al dejar de funcionar
- ▶ **Mantenimiento preventivo:**
  - ▶ Para prevenir fallos (proactivo)
  - ▶ Pueden planificarse
  - ▶ Ej.: cambiar las bombillas al 90% de su vida media

# Cálculo de la disponibilidad

## composición

$$A(t) = \prod_{i=1}^N A_i(t)$$



### ► Disponibilidad de los elementos:

- Hw.: 99.99 %
- Disco: 99.9 %
- S.O.: 99.99 %
- Aplicación: 99.9 %
- Comunicación 99.9

### ► Disponibilidad del sistema:

- 99.6804 % -> 1,17 días sin servicio

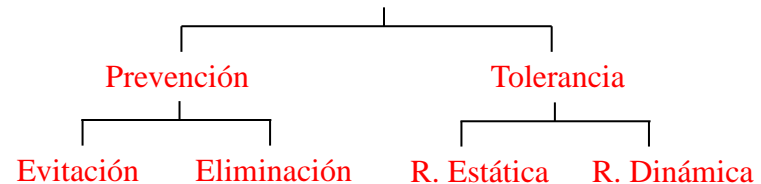
# Tolerancia a fallos

## ► 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**



# Técnicas para aumentar la fiabilidad

## ▶ **Prevención de fallos**

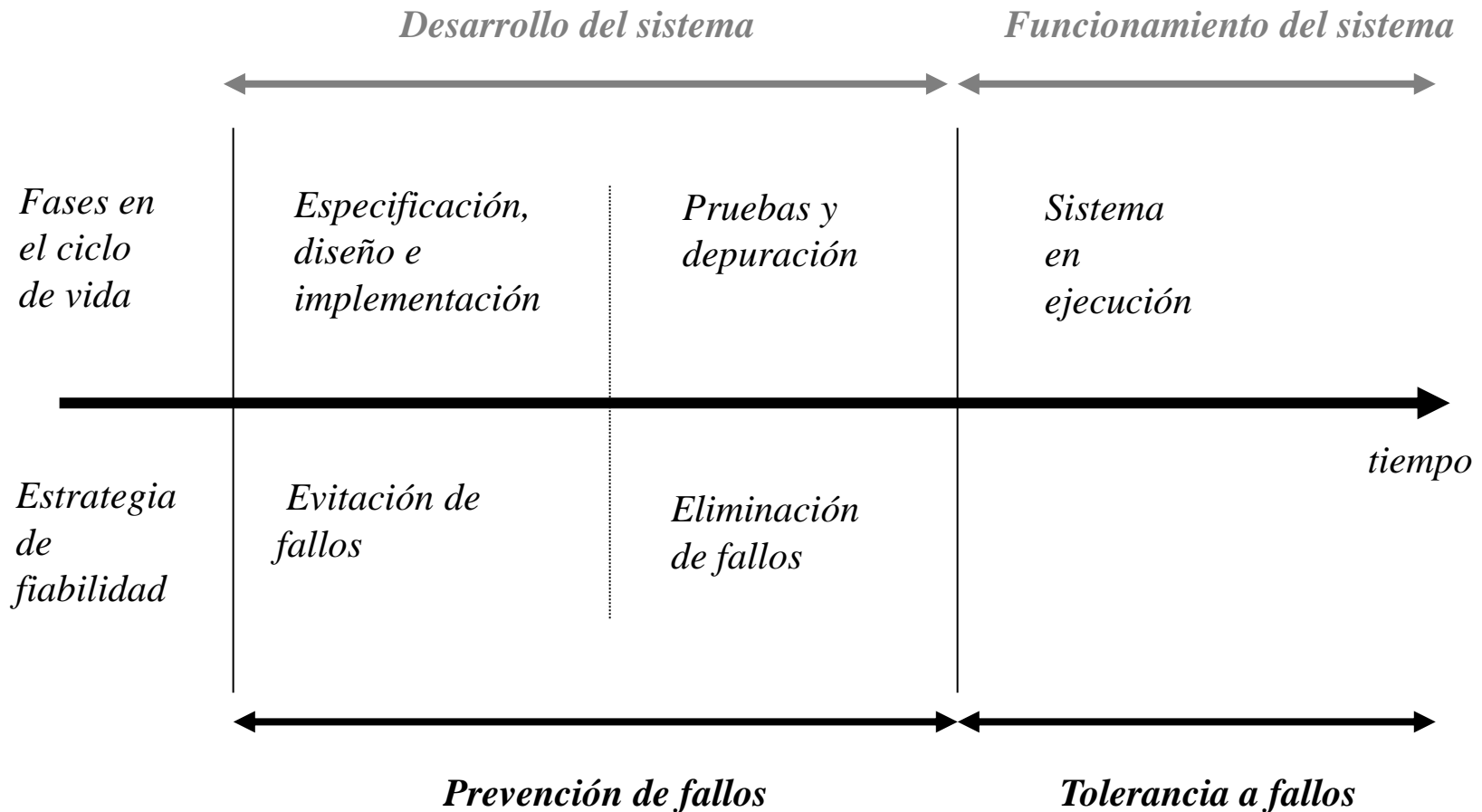
- ▶ Evitar que se introduzcan fallos en el sistema antes que entre en funcionamiento.
- ▶ Se utilizan en la fase de desarrollo del sistema.
  - ▶ Evitar fallos.
  - ▶ Eliminar fallos.

## ▶ **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 y herramientas 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 (además) 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.
- ▶ Precaución:
  - ▶ El empleo de redundancia aumenta la complejidad del sistema y puede introducir fallos adicionales si no se gestiona de forma correcta.
  - ▶ Los métodos y técnicas son sensibles a los errores en los requisitos (si está mal descrito el sistema...)

# Redundancia

- ▶ Todas las técnicas de tolerancia a fallos se basan en el uso de la **redundancia**.

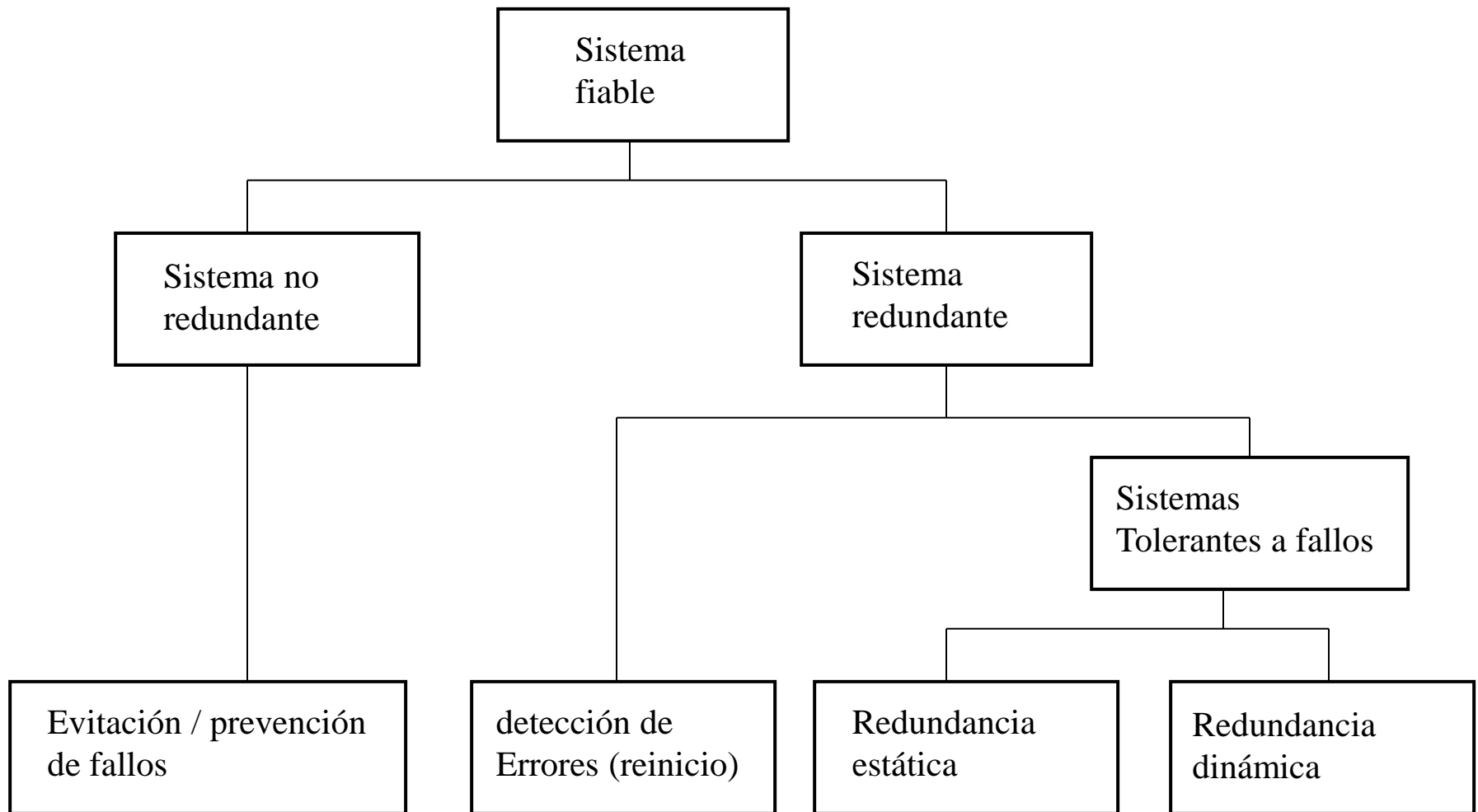
- ≡ *Redundancia **estática**:*

Los componentes redundantes se utilizan dentro del sistema para enmascarar los efectos de los componentes con defectos.

- ≡ *Redundancia **dinámica**.*

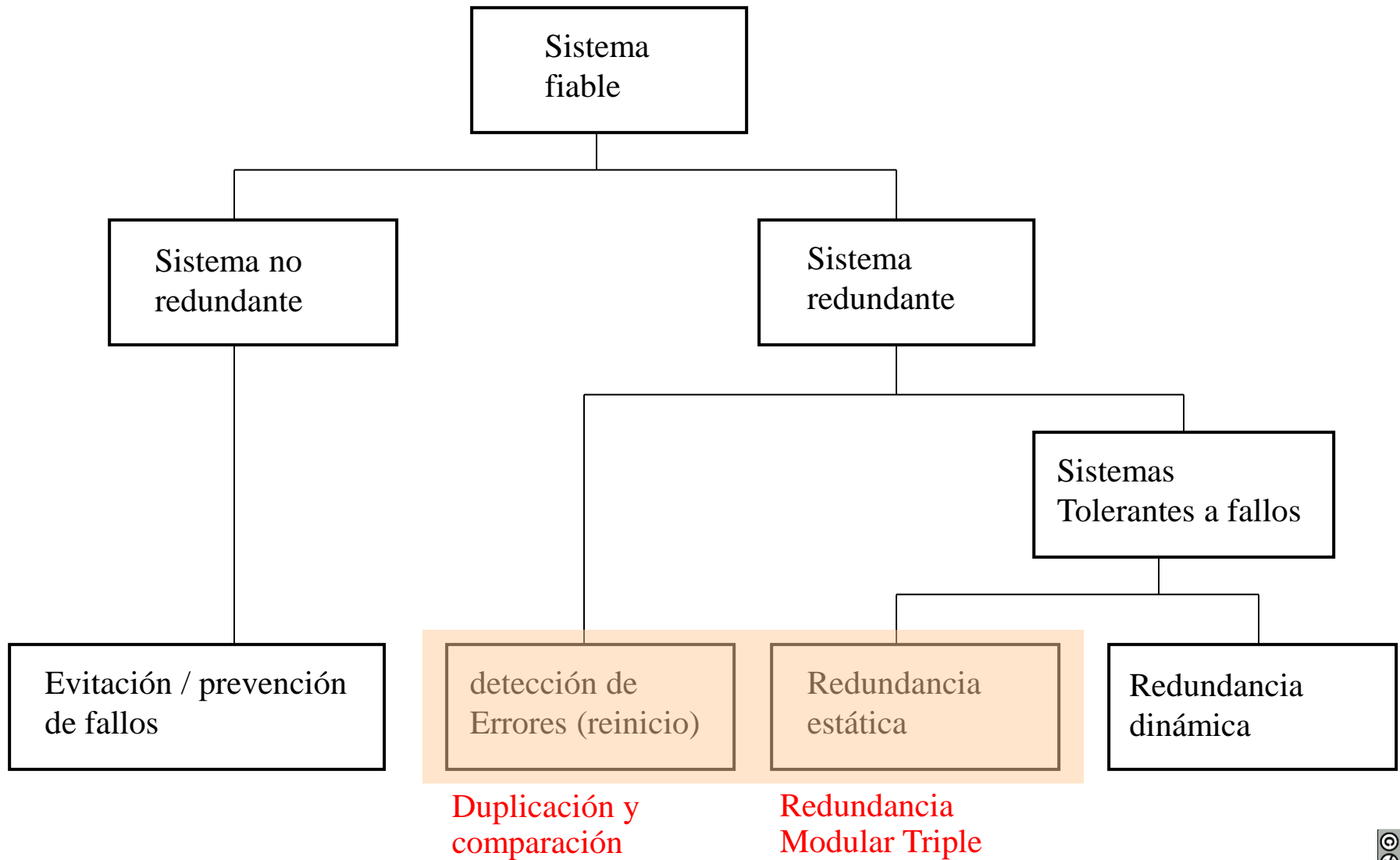
La redundancia se utiliza sólo para la detección de errores.  
La recuperación debe realizarla otro componente.

# Estrategias para diseñar un sistema fiable



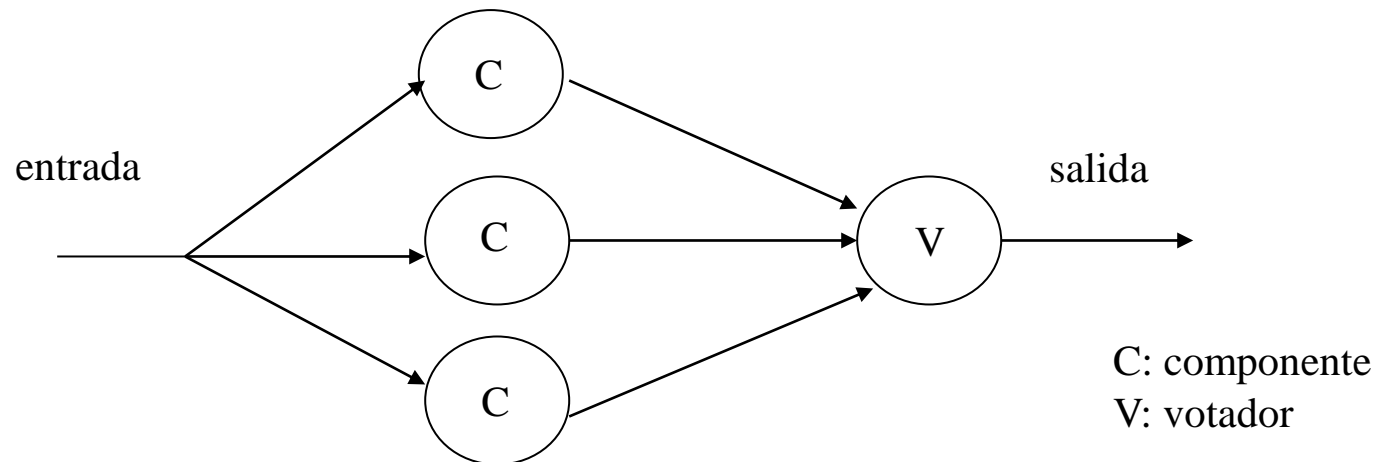
# Estrategias para diseñar un sistema fiable

## hardware



# Redundancia modular triple (TMR)

- ▶ Ejemplo de redundancia estática.



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

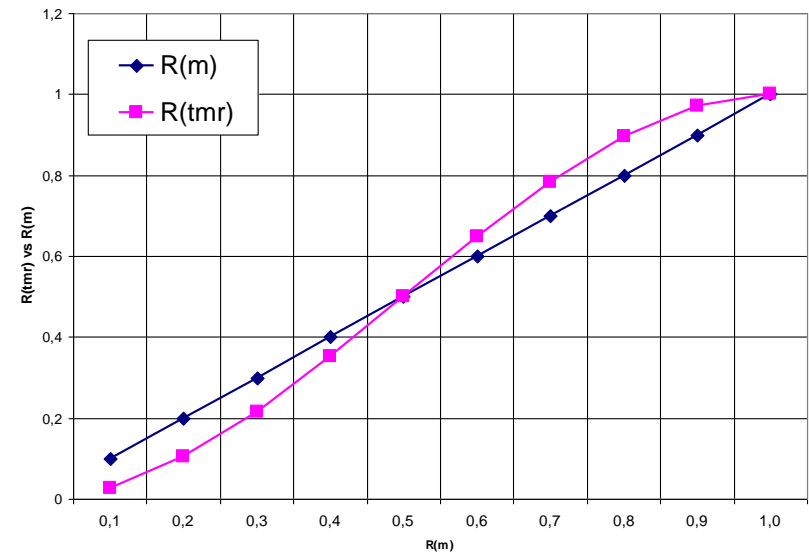


# Fiabilidad de un sistema TMR

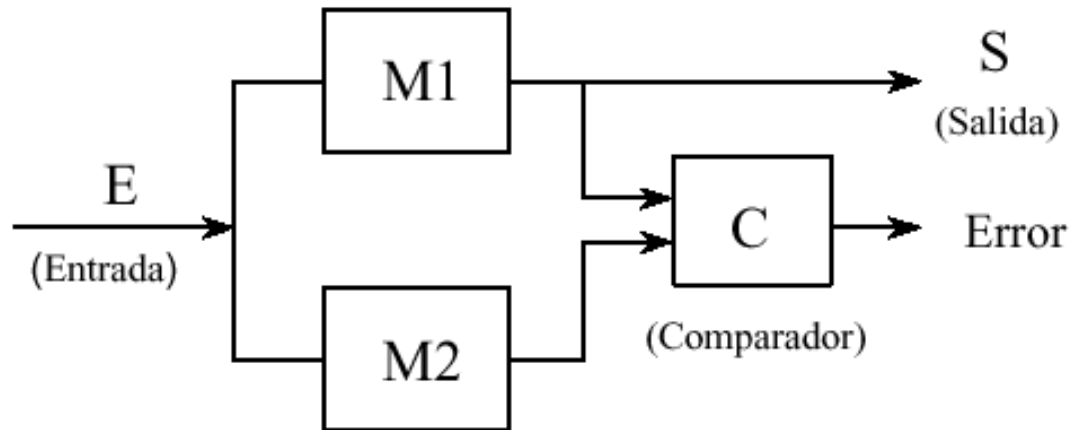
- Fiabilidad de un sistema TMR es:

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

- 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



M1, M2: Módulos con igual función

- 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)

# 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.



# Fases en la tolerancia a fallos

## 1. Detección de errores

- ▶ El **punto de partida es detectar** los **efectos de los errores**
  - No se puede detectar un fallo directamente. Su efecto darán lugar a errores en algún lugar del sistema.
- ▶ Hay que detectar el estado erróneo en el funcionamiento del sistema.

## 2. Confinamiento y diagnostico de daños

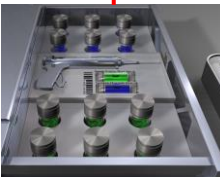
- ▶ Posible retraso entre la manifestación de un fallo y su detección:
  - El fallo puede provocar errores en otras partes del sistema.
- ▶ Antes de hacer frente al error detectado **es necesario**:
  - **Valorar alcance de los fallos** que pueden generarse.
  - **Limitar la propagación confinando los daños.**

## 3. Recuperación de errores

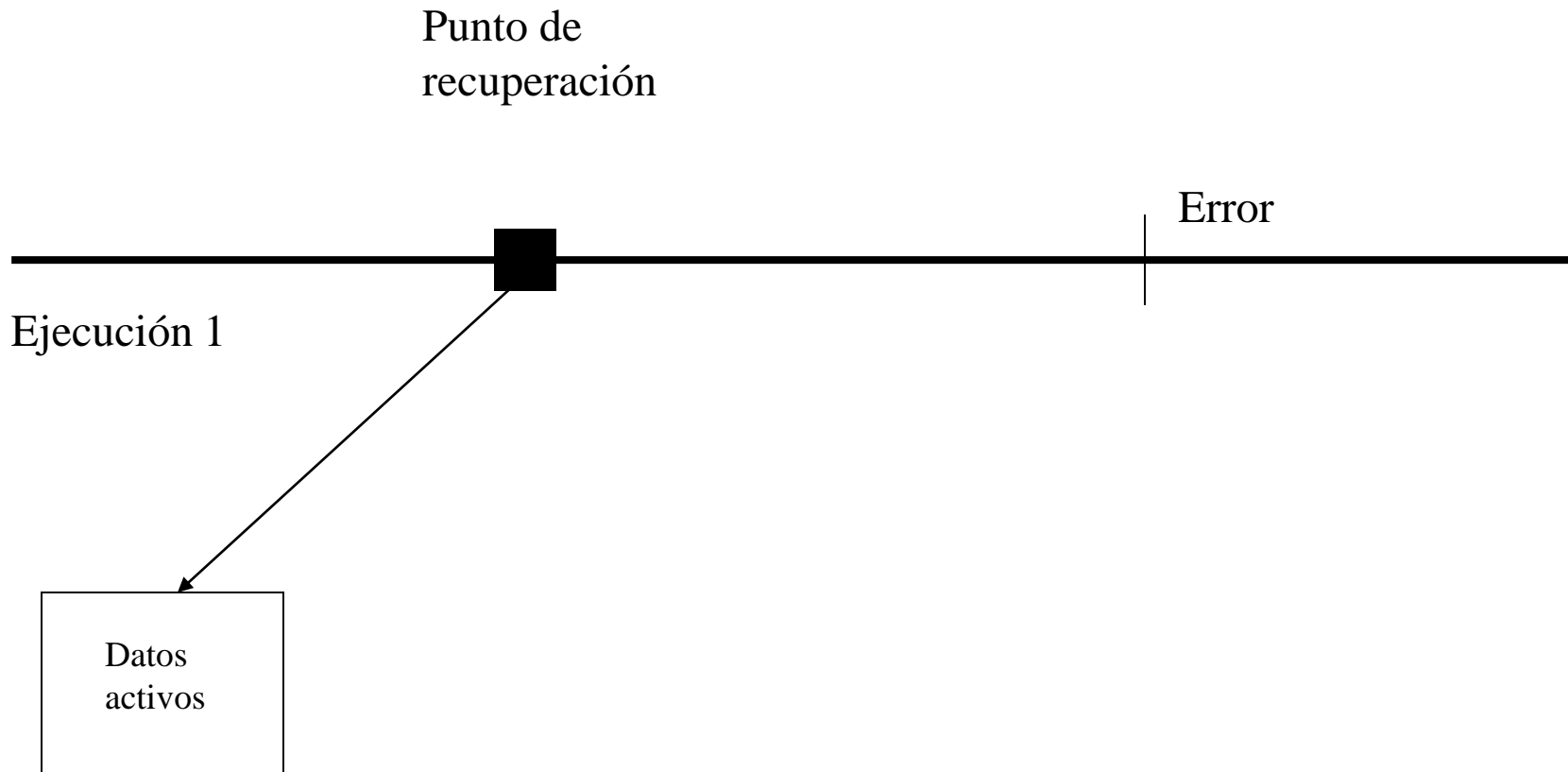
- ▶ Tras detectar y confinar el error **es necesario recuperar al sistema del error.**
- ▶ Uso de técnicas que transformen el estado erróneo en otro libre de errores:
  - A. **Recuperación hacia atrás**: volver a un estado anterior sin errores (**checkpoints, n-versiones**)
  - B. **Recuperación hacia delante**: llevar al sistema a un estado sin errores (**código autocorrector**).

## 4. Tratamiento de fallos y servicio continuado

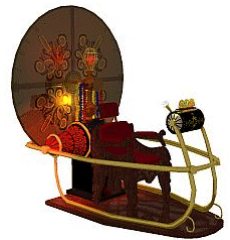
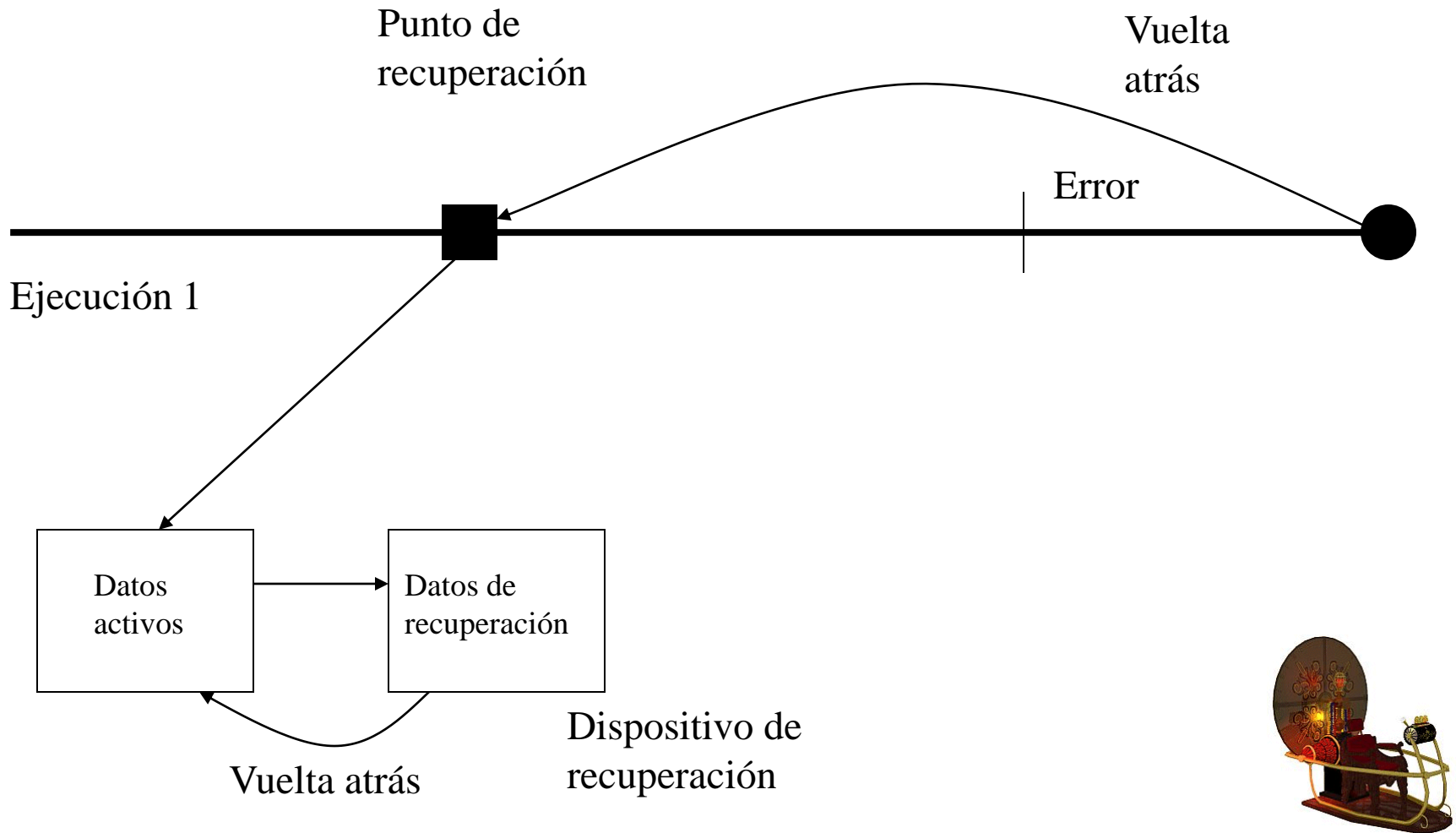
- ▶ Una vez detectado un error se repara el fallo.
- ▶ **Se reconfigura el sistema para evitar** que **el fallo vuelva a generar errores.**
  - Cuando los errores fueron transitorios no es necesario realizar ninguna acción.



## 3.a) Recuperación hacia atrás

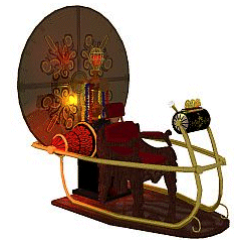
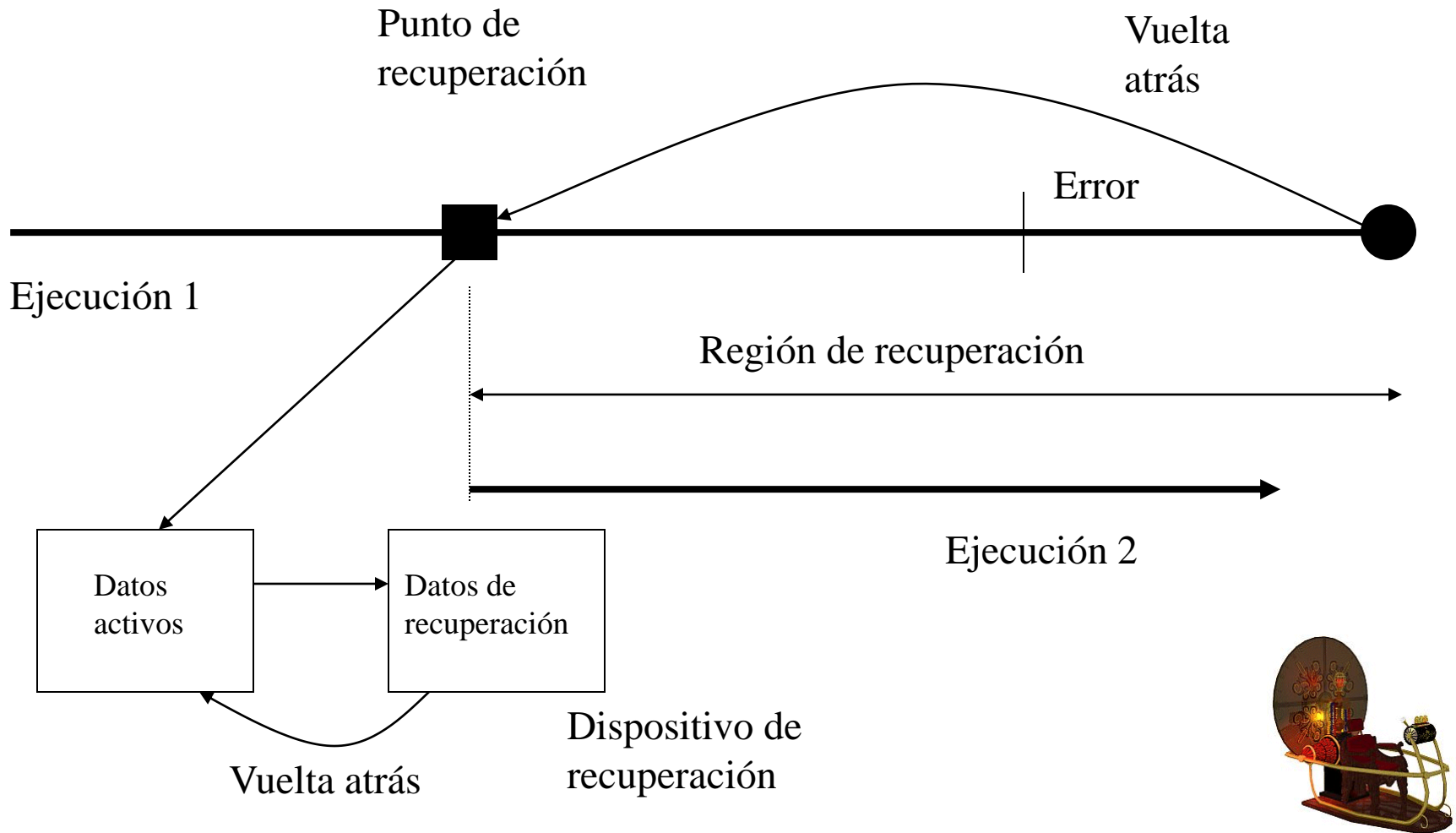


## 3.a) Recuperación hacia atrás





## 3.a) Recuperación hacia atrás



## 3.a) 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.

## 3.b) 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.

# Contenidos

1. Introducción a la tolerancia a fallos
2. **Tolerancia a fallos software**
3. Tolerancia a fallos en sistemas distribuidos

# Tolerancia a fallos

## repaso

### ► 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**



- 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.

# Tolerancia a fallos

## repaso

### ► 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



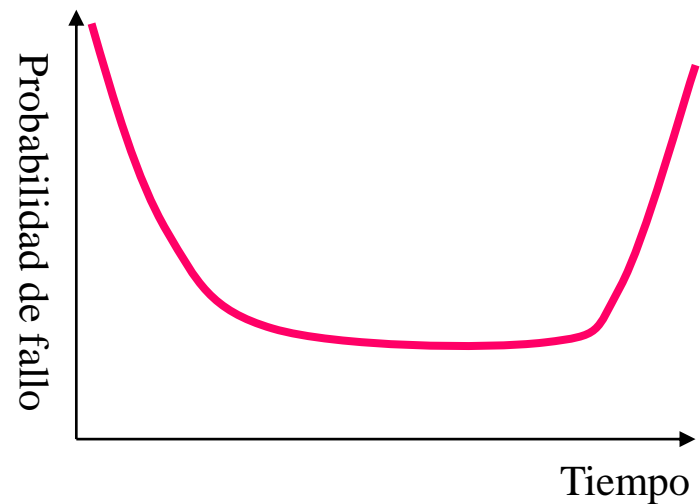
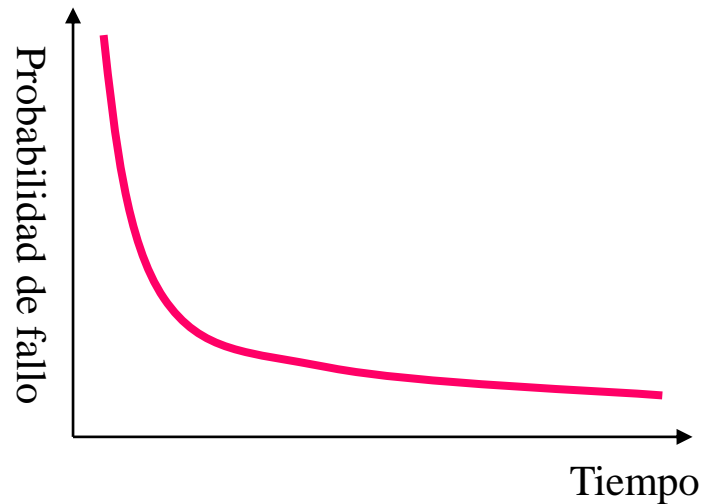
Origen de los fallos

- **Fallos hardware**
  - Fallos {permanentes o transitorios} x  
{componentes hardware o subsistemas de comunicación}
- **Fallos software**
  - **Especificación** inadecuada
  - Fallos introducidos por errores en **diseño y programación** de componentes software

# Fiabilidad

## fallos de software

- ▶ Los fallos en SW se deben a fallos en el diseño/implementación.
- ▶ 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

# Fases en la tolerancia a fallos de software...

- ▶ Las técnicas de tolerancia a fallo de software tienen como base en su diseño e implementación las siguientes cuatro fases:
  1. Detección de errores
  2. Confinamiento y diagnostico de daños
  3. Recuperación de errores
  4. Tratamiento de fallos y servicio continuado
  
- ▶ Las cuatro fases están presentes en las técnicas de tolerancia a fallos tanto en SW como en HW



# 1) Detección de errores

- ▶ Lo primero es necesario detectar los efectos de los errores.
  - ▶ No se puede detectar un fallo directamente. El efecto de los fallos darán lugar a errores en algún lugar del sistema.
- ▶ El punto de partida de cualquier técnica de tolerancia a fallos es la detección de un estado erróneo en el funcionamiento del sistema.

- **Por el entorno de ejecución:**

- *Señales:* sistema operativo (ej.: uso de puntero nulo, ...).
- *Excepciones:* error hardware (ej.: instrucción ilegal, 0/0, ...).

- **Por el software de aplicación:**

- Duplicación (redundancia con 2 versiones).
- Códigos detectores de error.
- Validación estructural (*asserts*).
  - Comprobar integridad de listas, colas, etc. (# de elementos, punteros redundantes).

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

## 2) Confinamiento y diagnóstico de fallos

- ▶ Cuando se detecta un error en el sistema, éste puede haber pasado por un cierto número de estados erróneos antes.
  - ▶ Posible retraso entre la manifestación de un fallo y su detección.
  - ▶ El fallo puede provocar errores en otras partes del sistema.
- ▶ Antes de hacer frente al error detectado es necesario:
  - ▶ Valorar alcance de los fallos que pueden generarse.
  - ▶ Limitar la propagación confinando los daños.

- Estructurar el sistema para minimizar daños causados por los componentes defectuosos mediante distintas técnicas:
  - **Descomposición modular:** confinamiento estático.
  - Acciones atómicas: confinamiento dinámico.
    - Mueven al sistema entre estados consistentes.

### 3) 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.

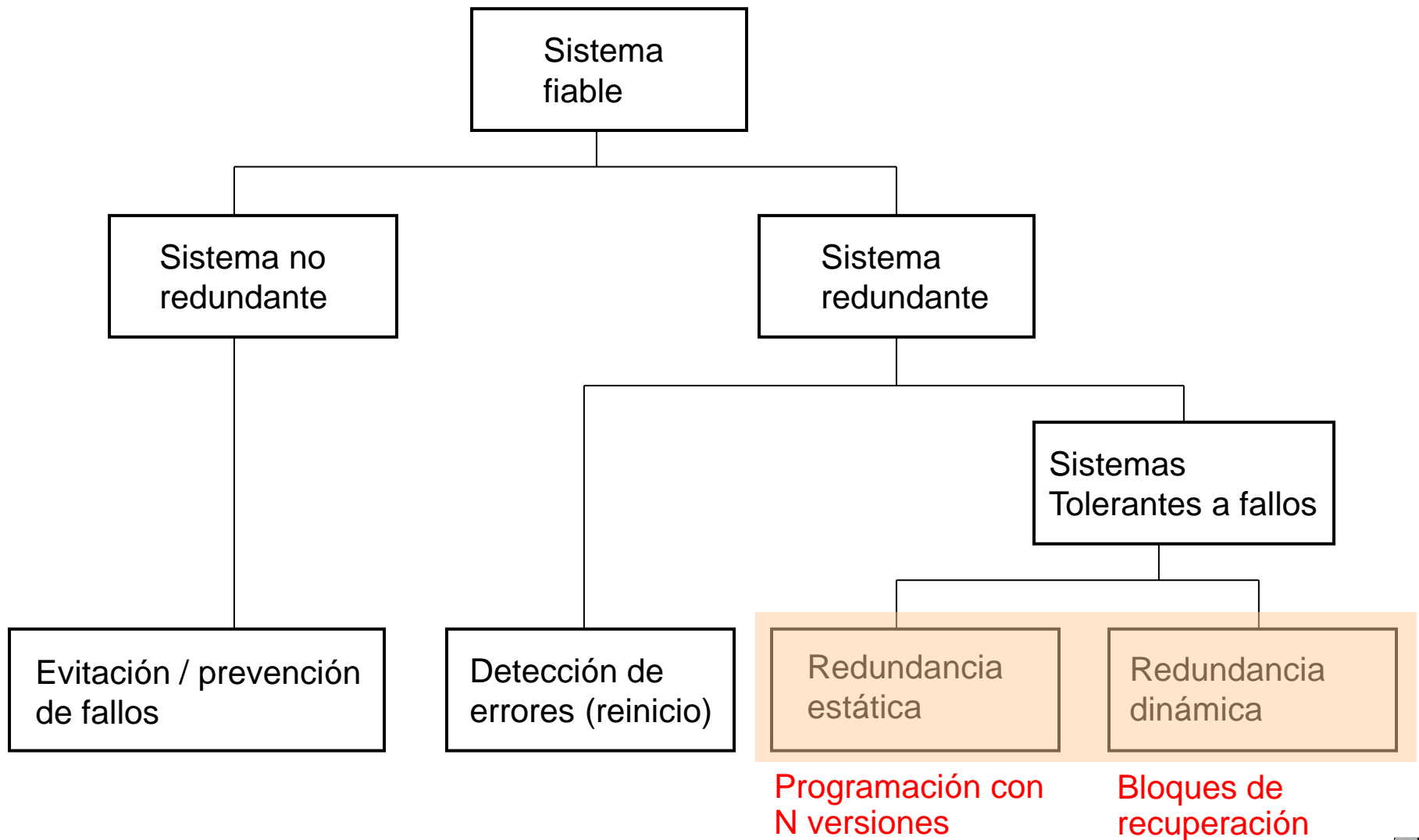
- Prepara el software para poder saltar a un estado sin error:
  - Redundancia estática
    - **Programación con N versiones**
  - Redundancia dinámica
    - **Puntos de recuperación o Checkpoints** (volver atrás)
    - Programación con códigos autocorrectores (recuperación hacia adelante).
- Todos los métodos son sensibles a los errores en los requisitos

## 4) 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.
- **Prepara el software para poder actualizarse:**
  - **Actualización del software** y reiniciar sistema
  - Carga de componentes software actualizados dinámicamente

# Estrategias para diseñar un sistema fiable

## software

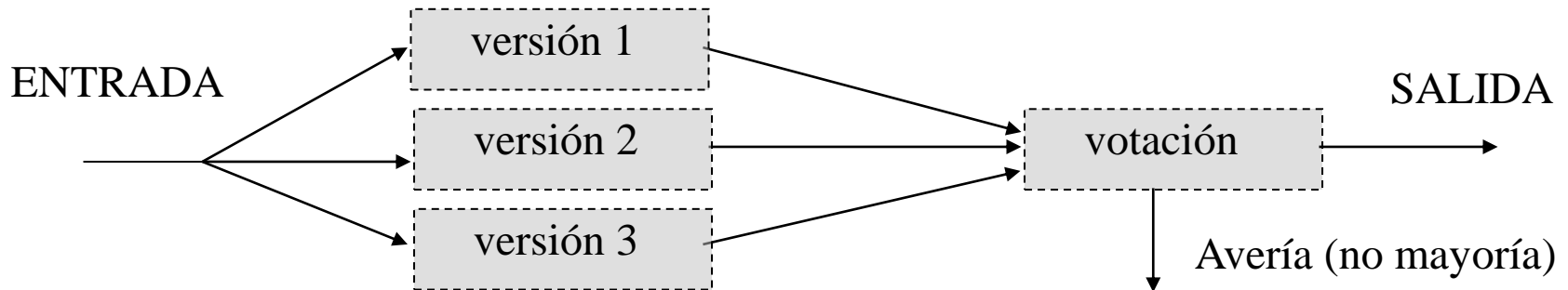


# Redundancia estática

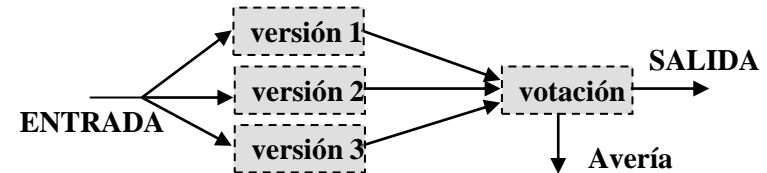
- ▶ Redundancia estática en el software:
  - ▶ Los componentes redundantes se utilizan dentro del sistema para enmascarar los efectos de los componentes con defectos
- ▶ Se aplican las cuatro fases:
  1. Detección de errores
  2. Confinamiento y diagnóstico de daños
  3. Recuperación de errores
  4. Tratamiento de fallos y servicio continuado
- ▶ Técnicas principales:
  - ▶ **Programación con N versiones**

# Programación con N versiones

- ▶ La programación N-versión se define como la generación independiente de  $N$  ( $N \geq 2$ ) 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.



# Aplicación de las cuatro fases



- ▶ *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.

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

NOTA: Para permitir F fallos se necesitan  $2 \cdot F + 1$  módulos



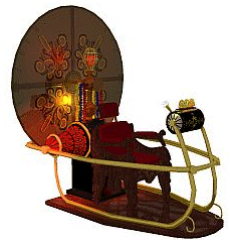
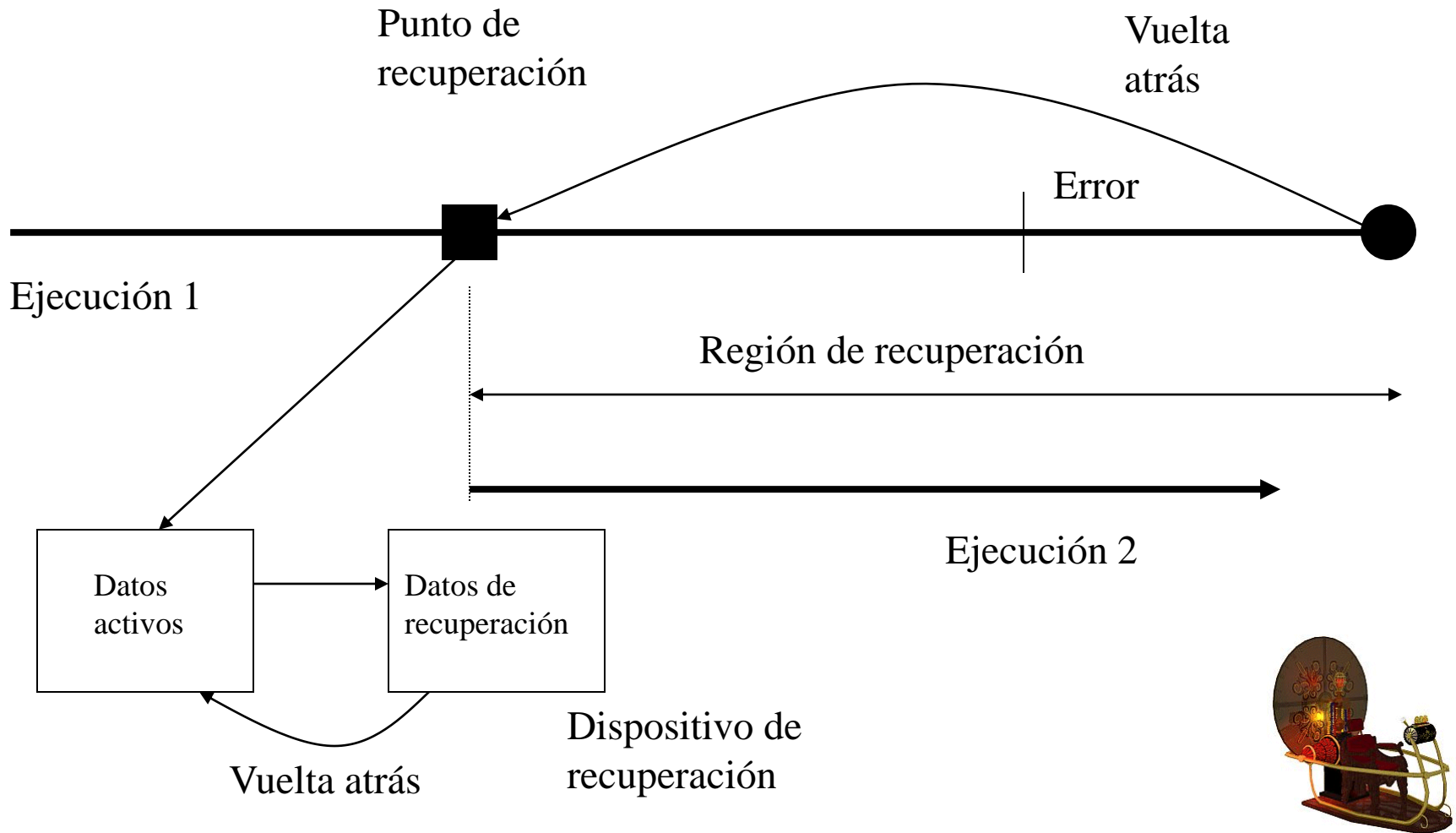
# La programación con N versiones depende de...

- ▶ 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 diagnóstico de daños
  3. Recuperación de errores
  4. Tratamiento de fallos y servicio continuado
- ▶ Técnicas principales:
  - ▶ **Bloques de recuperación**

# 3.a) Recuperación hacia atrás repaso



## 3.a) Recuperación hacia atrás

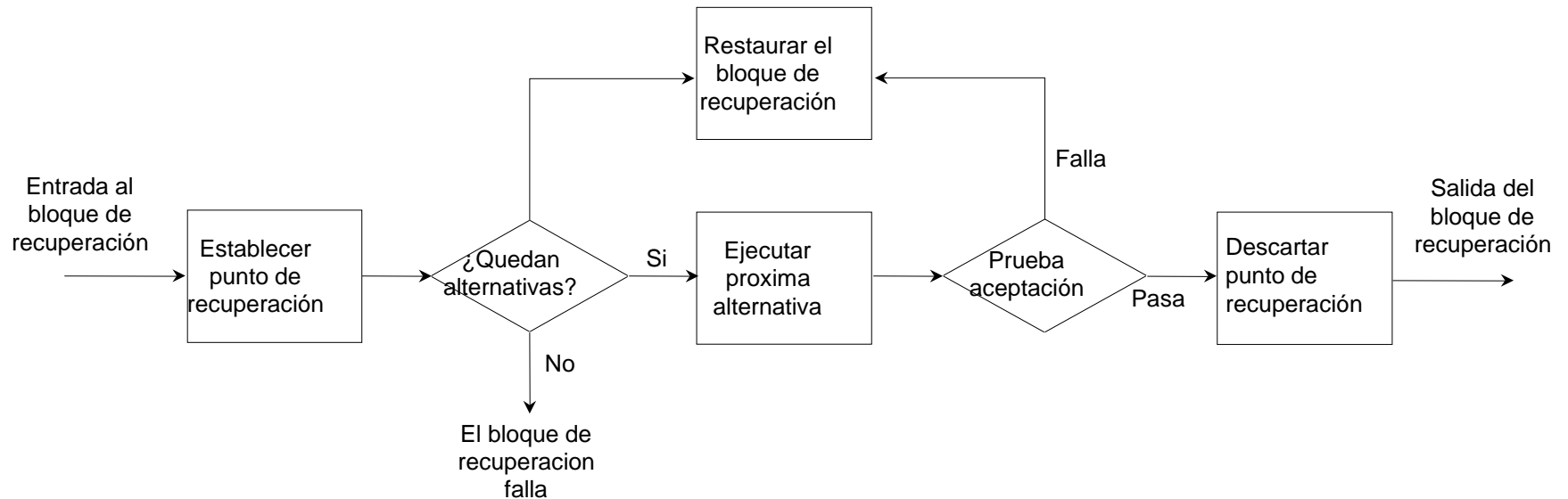
### repaso

- ▶ **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.

# 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.

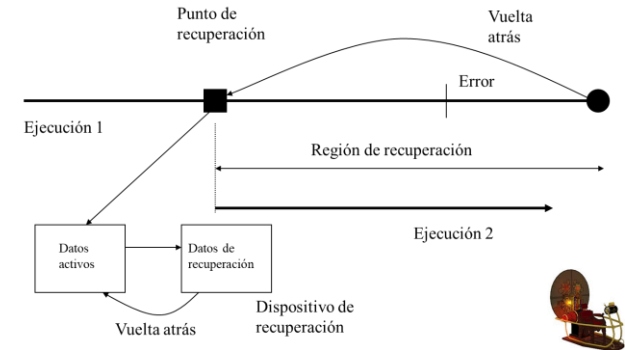
# Prueba de aceptación

- ▶ 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



# Primitivas necesarias

**ejemplo**



## ► ***Establecer punto de recuperación:***

- Salvaguarda los registros y las páginas modificadas por el proceso desde el último punto de recuperación.

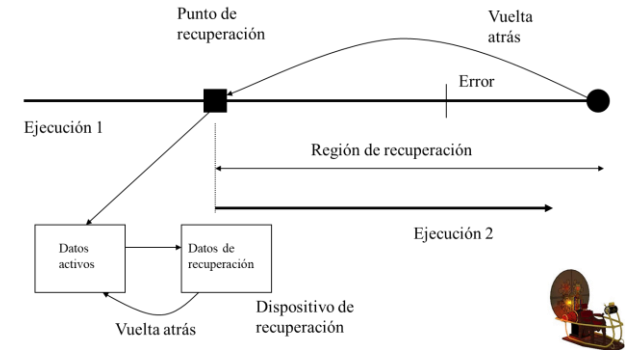
## ► ***Anular punto de recuperación:***

- 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.

# Aplicación de las cuatro fases



- ▶ *Detección de errores:*
  - ▶ La realiza la prueba de aceptación.
- ▶ *Confinamiento y diagnostico de daños:*
  - ▶ Se hace al diseñar el bloque de recuperación.
- ▶ *Recuperación de errores:*
  - ▶ Se consigue volviendo atrás y ejecutando otro código.
- ▶ *Tratamiento de fallos y servicio continuado:*
  - ▶ Volviendo al estado inicial del bloque de recuperación.

# 3.a) Recuperación hacia atrás

## Tipos de sistemas

### ▶ ***Transparentes a la aplicación:***

- ▶ El establecimiento de los puntos de recuperación y la vuelta atrás queda bajo el control del hardware o del sistema operativo.
- ▶ **Ventaja:** transparencia.
- ▶ **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.
- ▶ **Ventajas:** se establece en el momento adecuado y permite minimizar el conjunto de datos a salvaguardar.
- ▶ **Problema:** falta de transparencia.

# Programación con N versiones autocomprobantes

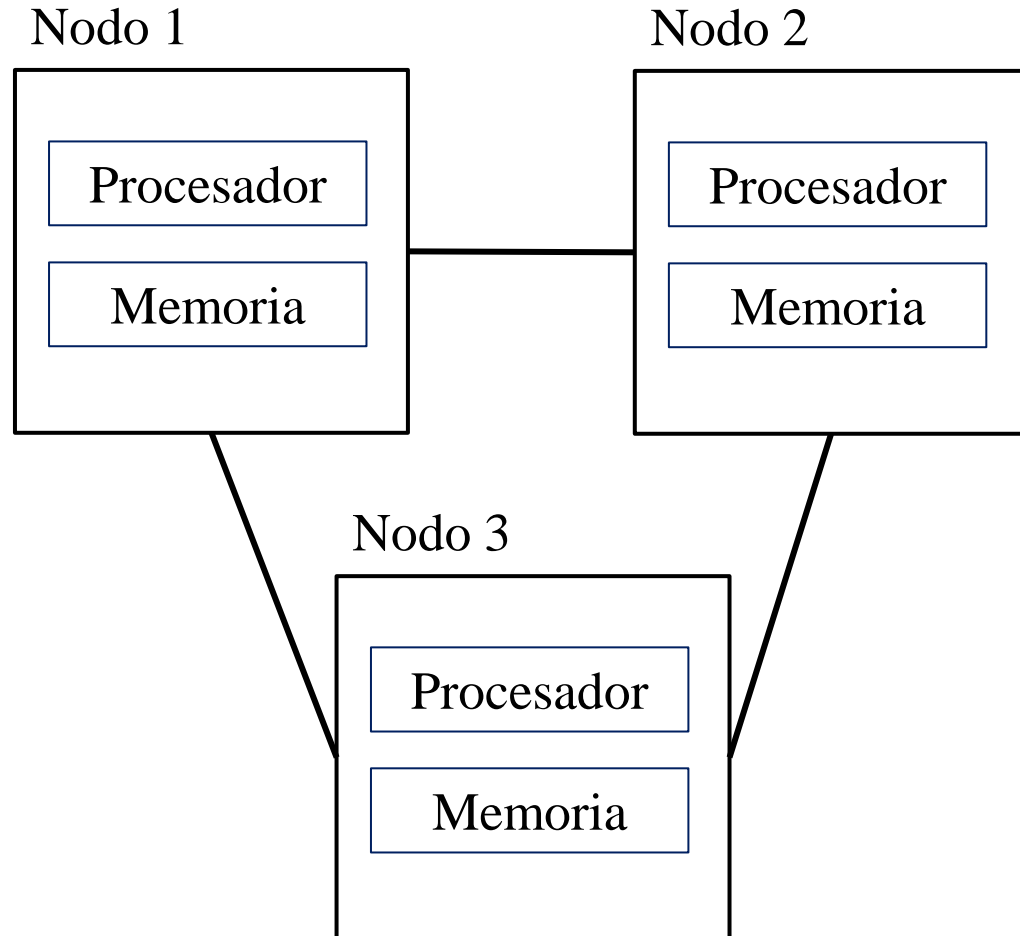
- ▶ Similar a la programación con N versiones pero con redundancia dinámica.  
Se basa en la diversidad de diseño.
- ▶ Cada componente realiza su propia comprobación.
- ▶ Dos tipos de componentes autocomprobantes:
  - ▶ Una variante y una prueba de aceptación.
  - ▶ Dos variantes y un algoritmo de comparación.
- ▶ 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

1. Introducción a la tolerancia a fallos
2. Tolerancia a fallos software
3. **Tolerancia a fallos en sistemas distribuidos**
  - ▶ **Procesamiento:** N-versiones, *checkpoint*, ...
  - ▶ **Almacenamiento:** replicación y consistencia, *snapshots*, ...
  - ▶ **Comunicación:** CRC, número de secuencia, retransmisión, ...

# Sistema distribuido

## repaso



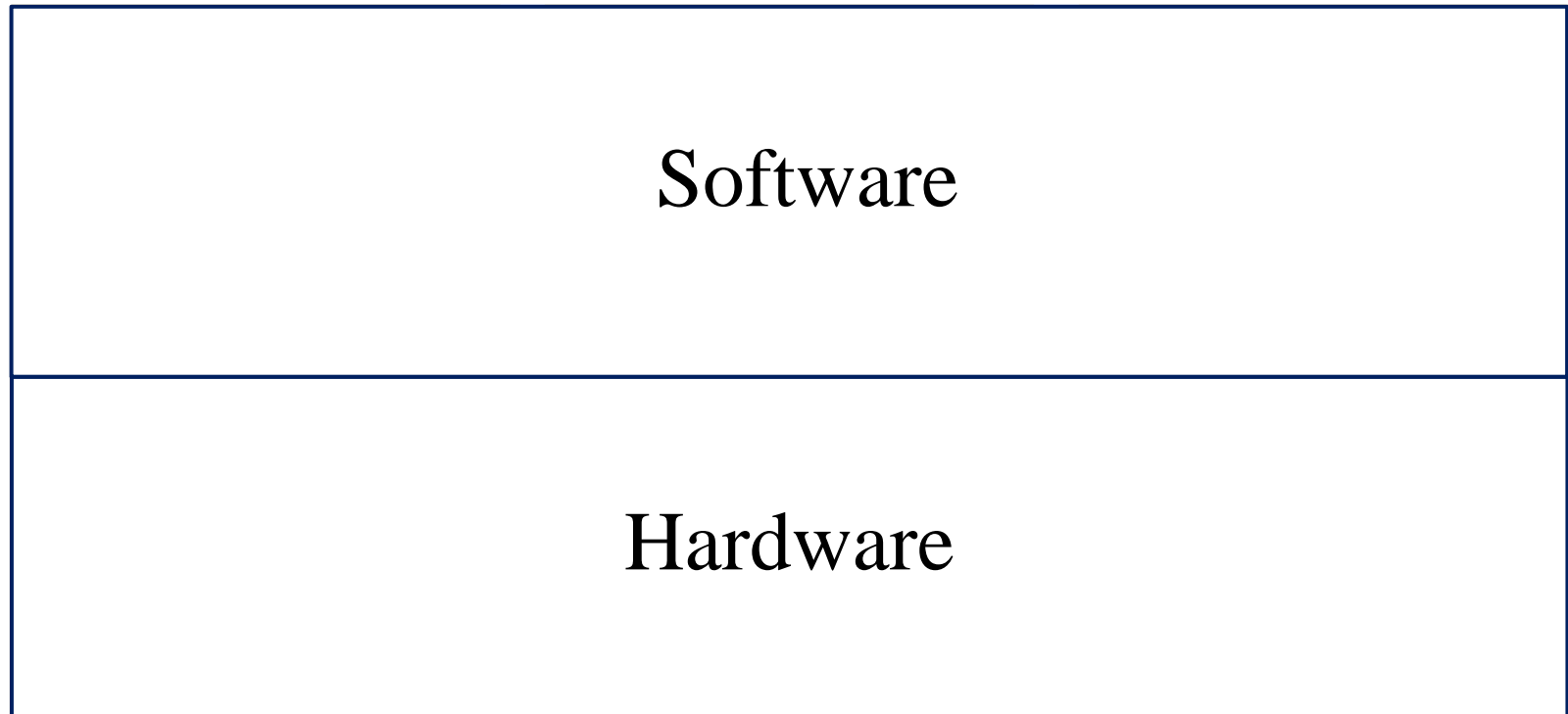
# Sistema distribuido

## repaso

- ▶ Un **sistema distribuido** es una **colección de ordenadores independientes** que aparecen a sus usuarios como un **único sistema coherente**.
  - ▶ Cada sistema tiene su **propia memoria** (y recursos).
  - ▶ Los sistemas se organizan para ocultar la existencia al usuario final: **transparencia**.
  - ▶ Se utiliza **primitivas de paso de mensaje** o **llamada a procedimiento/método remoto** a través de protocolos de comunicación de red como TCP/IP.
- ▶ Los sistemas distribuidos se hacen de un gran número de componentes, lo que dispara la probabilidad de fallo.
  - ▶ Un fallo crítico hace que todo el sistema distribuido deje de funcionar.

# Sistema distribuido

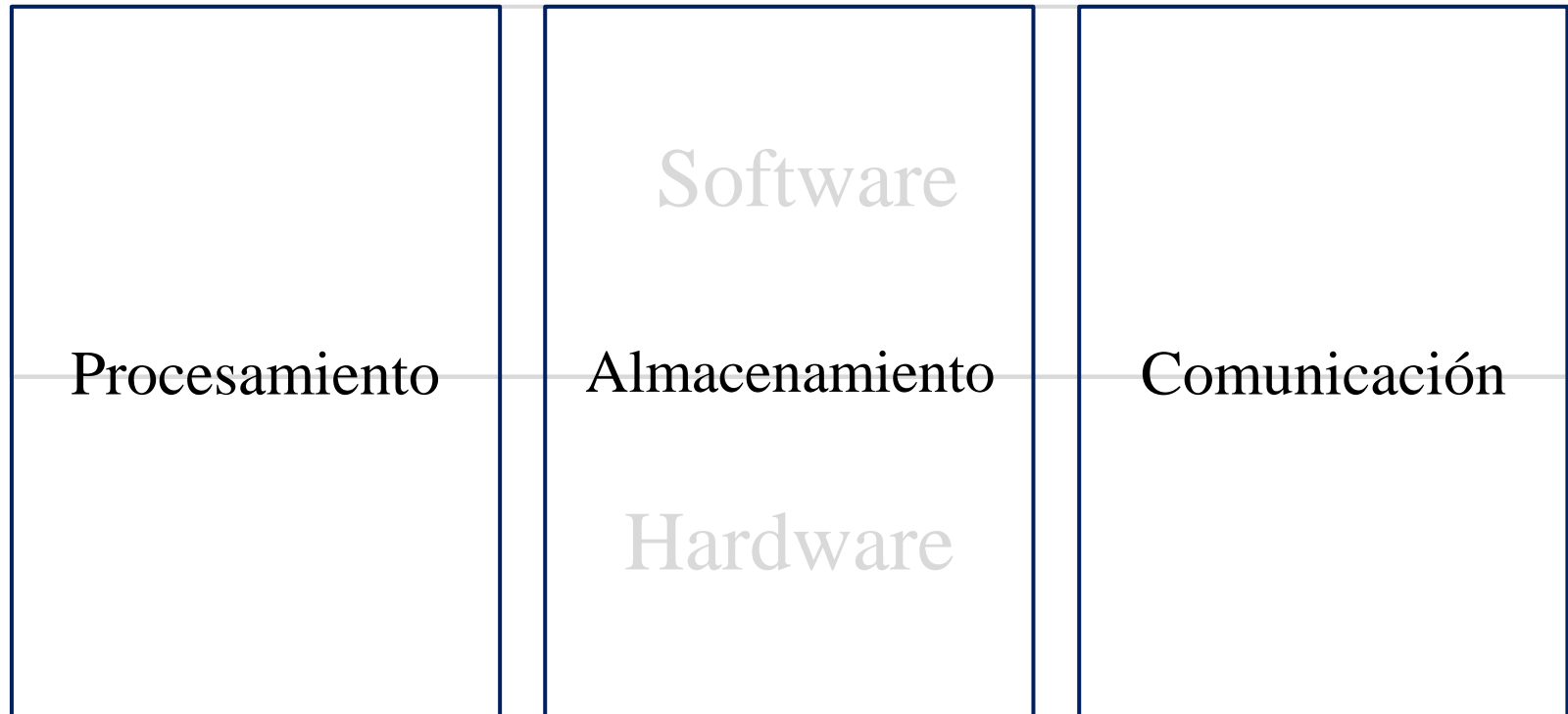
## repaso



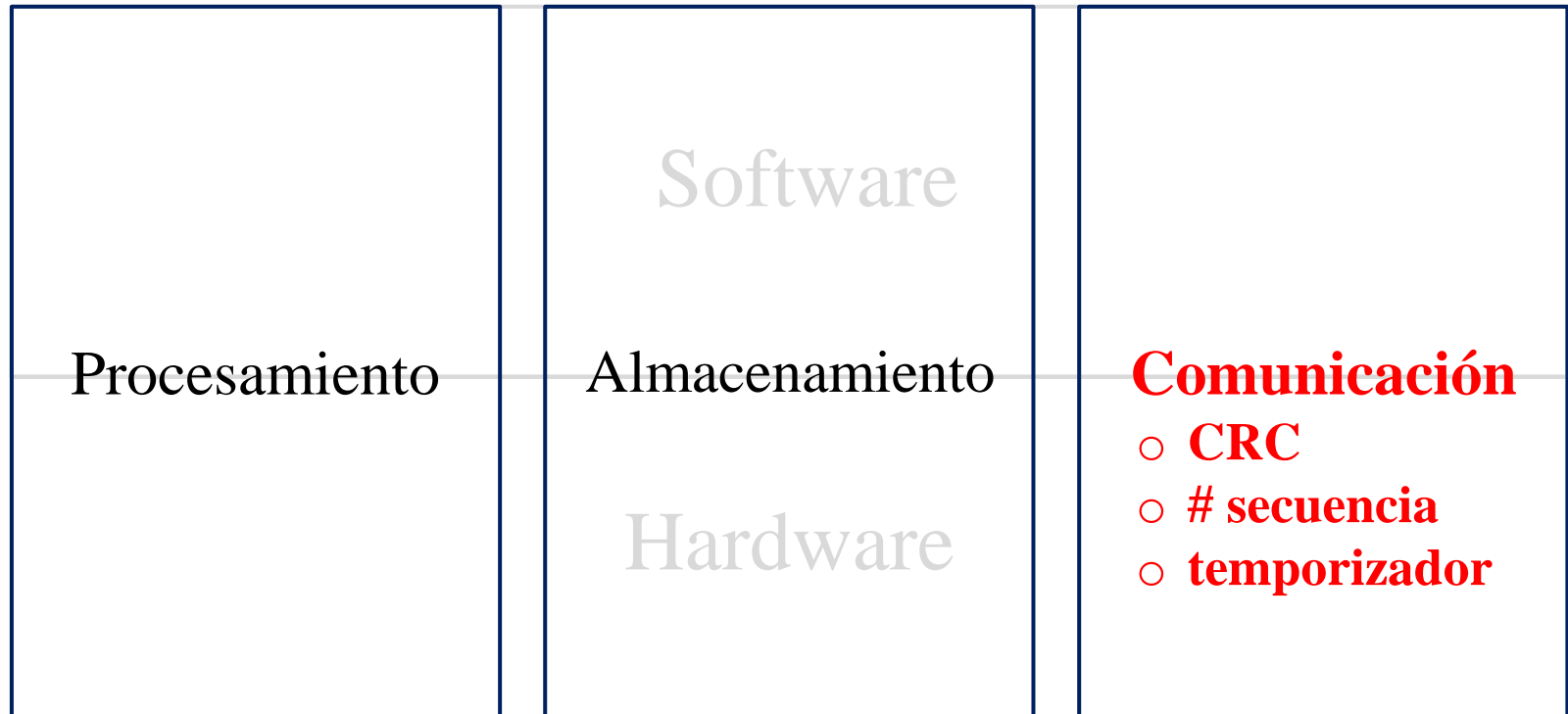


# Sistema distribuido

## repaso



# Tolerancia a fallos en **comunicación**



# Entrega de mensajes fiable

- ▶ Posibles **problemas** durante el envío:
  - ▶ **Corrupción del mensaje:**
  - ▶ **Duplicación de mensajes:**
  - ▶ **Perdida de mensaje:**

# Entrega de mensajes fiable

- ▶ Posibles ***soluciones*** para ellos:
  - ▶ ***Corrupción del mensaje:***
    - ▶ *El uso de CRC hace que se transforme en mensaje perdido*
  - ▶ ***Duplicación de mensajes:***
    - ▶ *El uso de número de secuencias para descartar*
  - ▶ ***Perdida de mensaje:***
    - ▶ *Temporizador y retransmisión de mensaje perdido*
    - ▶ *Es posible redirigir los mensajes por diferentes caminos*

# Necesidades adicionales


- ▶ *¿Son solo estos los únicos problemas?*


- ▶ **No**

*Ejemplo: puede que se envíe de forma correcta  
un mensaje incorrecto (fallo en la aplicación)*


- ▶ *Necesarias técnicas para ayudar a solucionarlos...*

# Necesidades principales 😊

 **Tweet**



**Mathias Verraes**  
@mathiasverraes



There are only two hard problems in distributed systems: 2. Exactly-once delivery 1. Guaranteed order of messages 2. Exactly-once delivery

[Traducir Tweet](#)

8:40 p. m. · 14 ago. 2015

7.133 Retweets

189 Citas

6.861 Me gusta

95 Elementos guardados

# Tolerancia a fallos en **software**



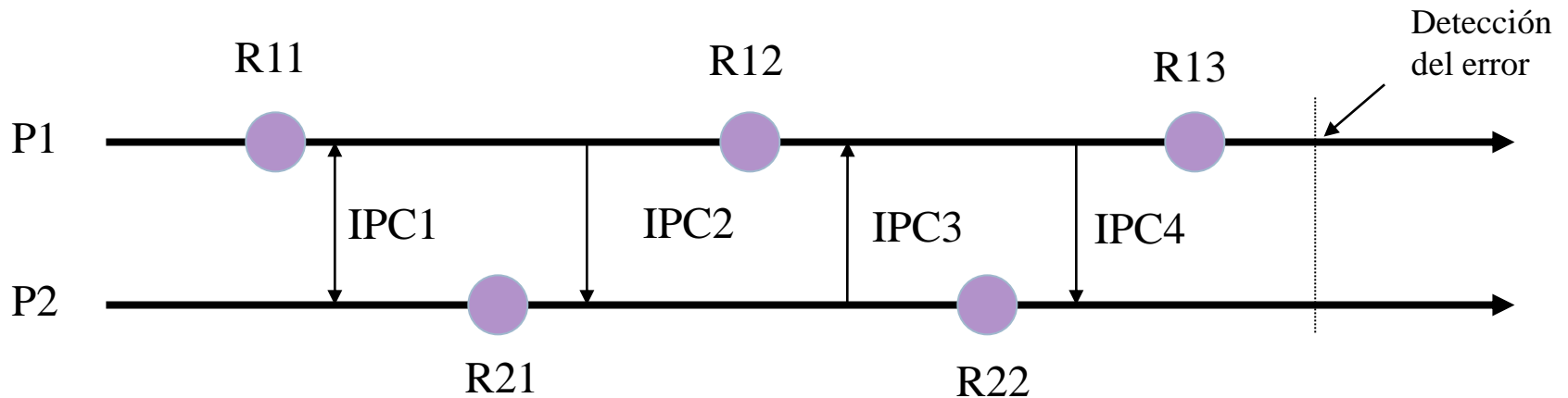
# Puntos de recuperación en sistemas concurrentes

- ▶ 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.

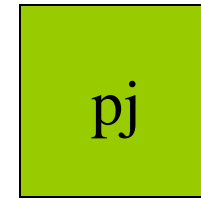
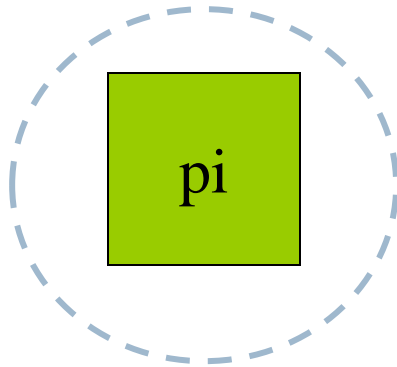


# Efecto dominó

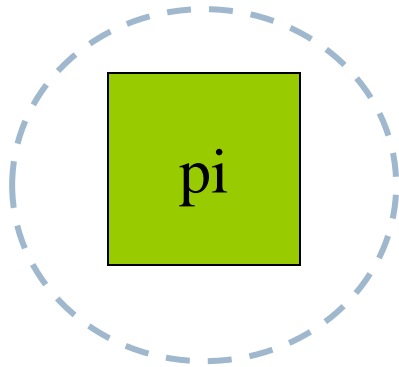
- ▶ Se produce un conjunto de vuelta atrás no acotado que puede llegar a reiniciar el sistema concurrente.
- ▶ Solución: ***líneas de recuperación***
  - ▶ Objetivo: acotar el efecto dominó en caso de realizar una vuelta atrás encontrando un conjunto de procesos y de puntos de recuperación que permita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



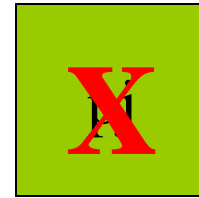
# Detectores de fallos



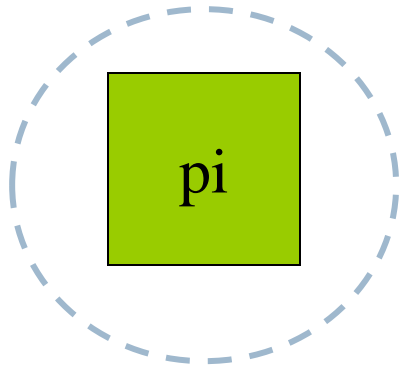
# Detectores de fallos



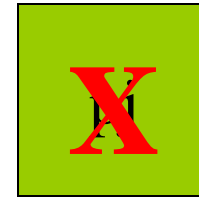
Proceso  $p_j$  falla



# Detectores de fallos

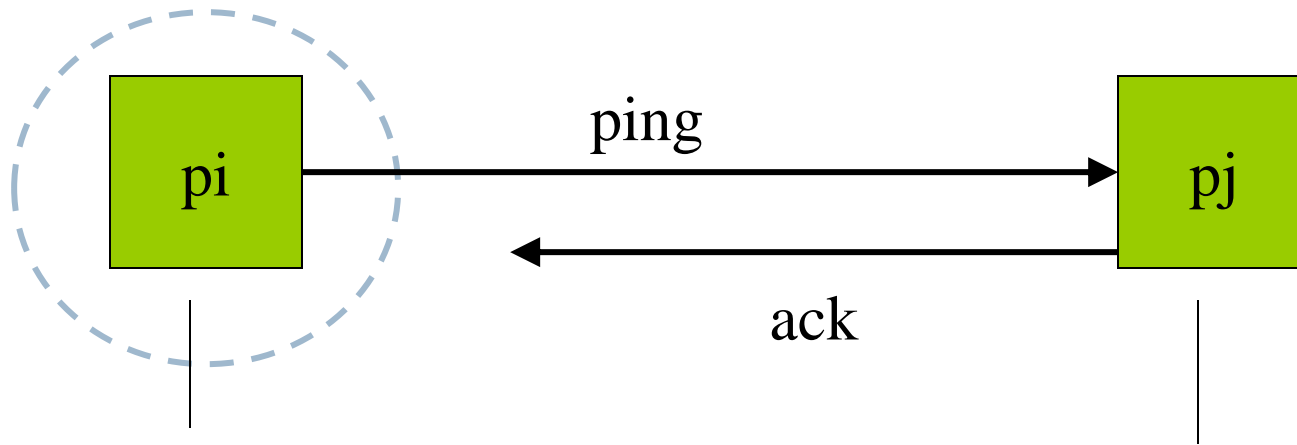


Proceso  $p_j$  falla



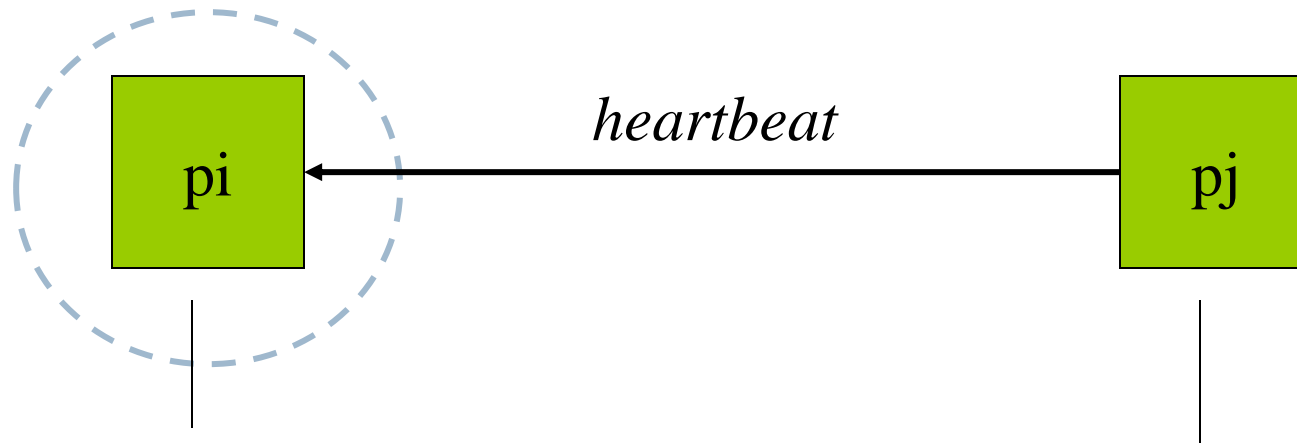
$P_i$  es un proceso sin fallo que necesita conocer el estado de  $P_j$

# Protocolo basado en ping



- De forma periódica  $p_i$  interroga a  $p_j$

# Protocolo basado en latido



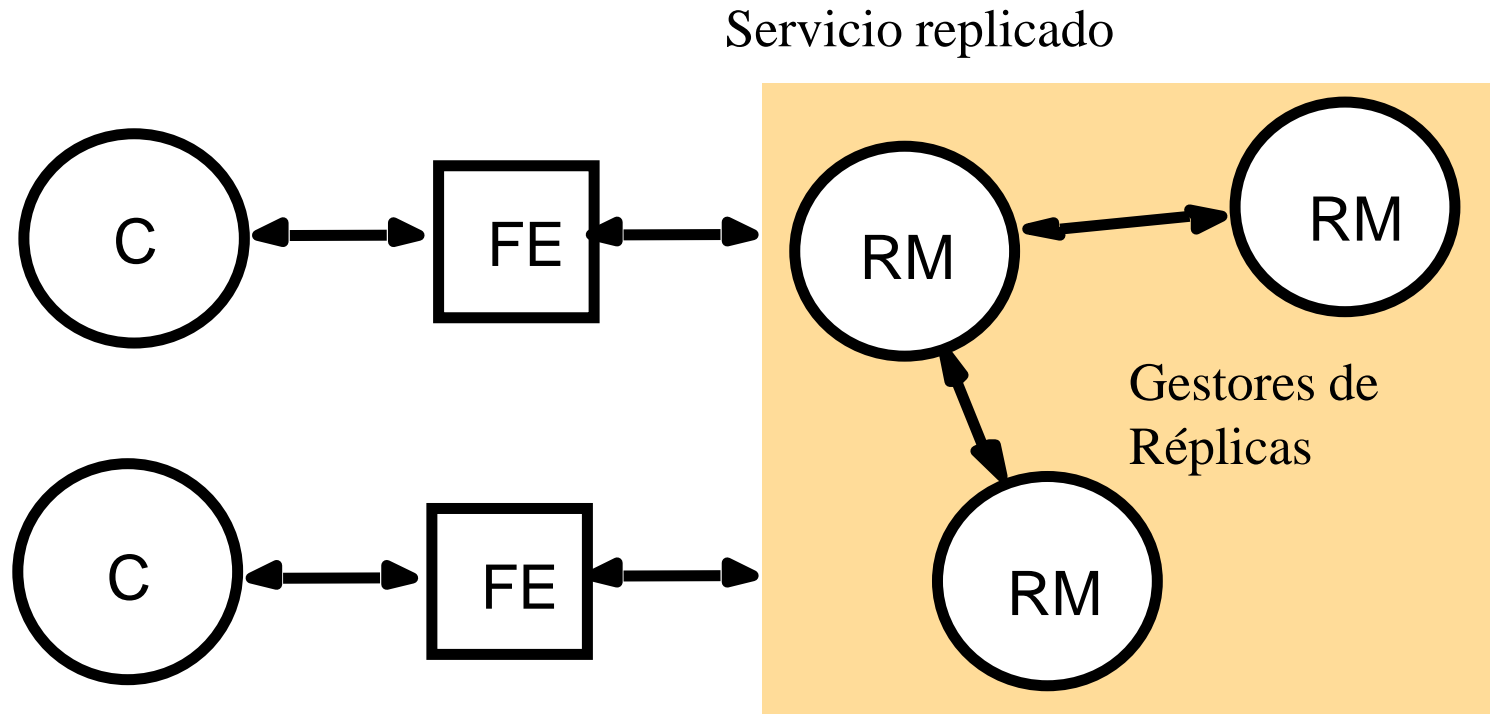
- ▶  $p_j$  mantiene un número de secuencia
- ▶  $p_j$  envía a  $p_i$  a latido con un nº de sec. incrementado cada  $T$  unidades de  $t$ .

# Tolerancia a fallos en **almacenamiento**

Replicación (N-copias) e instantáneas



# Arquitectura básica de replicación



Front-end: gestiona la replicación haciéndola transparente



# 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

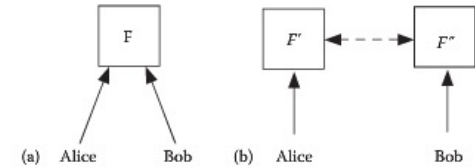
- ▶ Permite mantener los datos cerca de los/as usuarios/as
- ▶ Mejora la disponibilidad
  - ▶ El sistema puede continuar funcionando aunque algunas partes fallen
- ▶ Mejora la escalabilidad
  - ▶ Al permitir atender a más clientes

# Geo-replicación

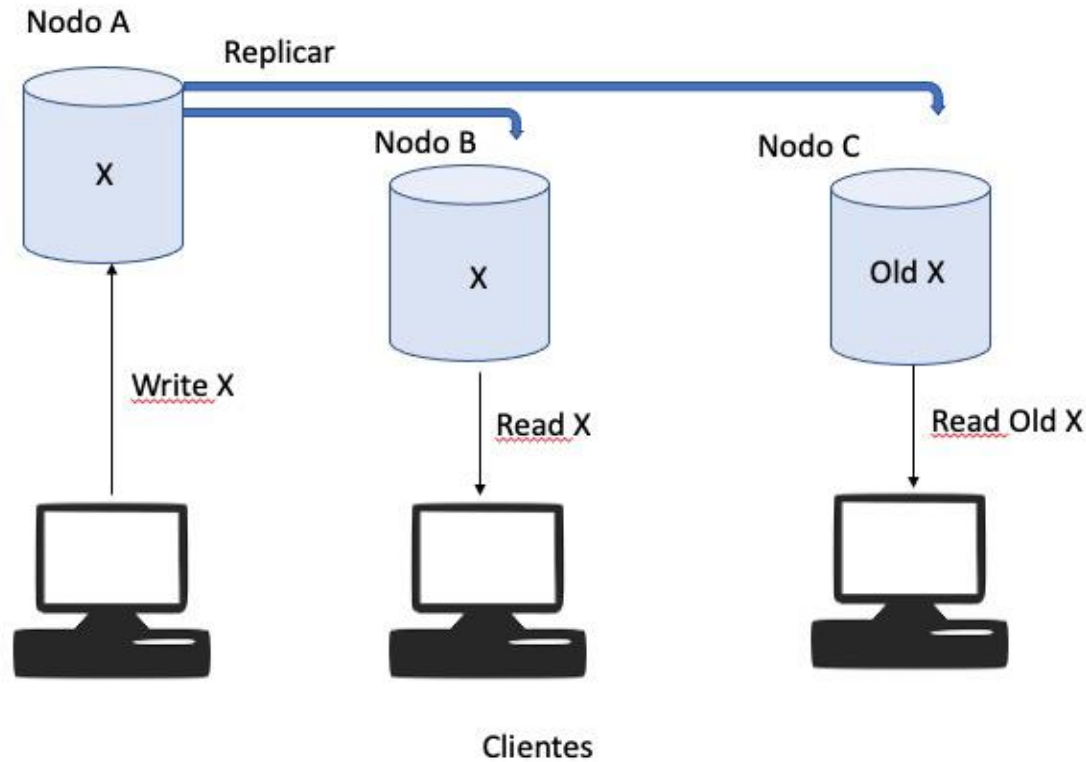
- ▶ Replicación entre centros conectados con redes de alta latencia
- ▶ Objetivos:
  - ▶ Clientes acceden a centros cercanos con baja latencia
  - ▶ Distribuye la carga
  - ▶ Escalabilidad: facilidad para atender a miles de usuarios
  - ▶ Disponibilidad, tolerancia a fallos
- ▶ Ejemplos:
  - ▶ Google,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

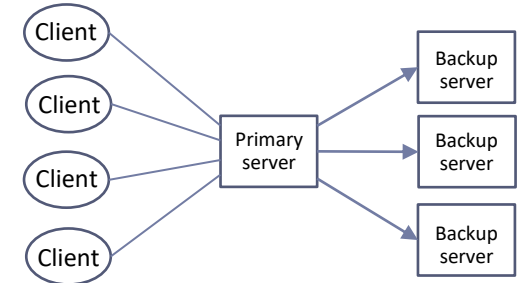


# Métodos de replicación

- ▶ **Métodos pesimistas:** imponen restricciones en los accesos cuando hay particiones
  - ▶ Copia **primaria** (replicación pasiva)
  - ▶ 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

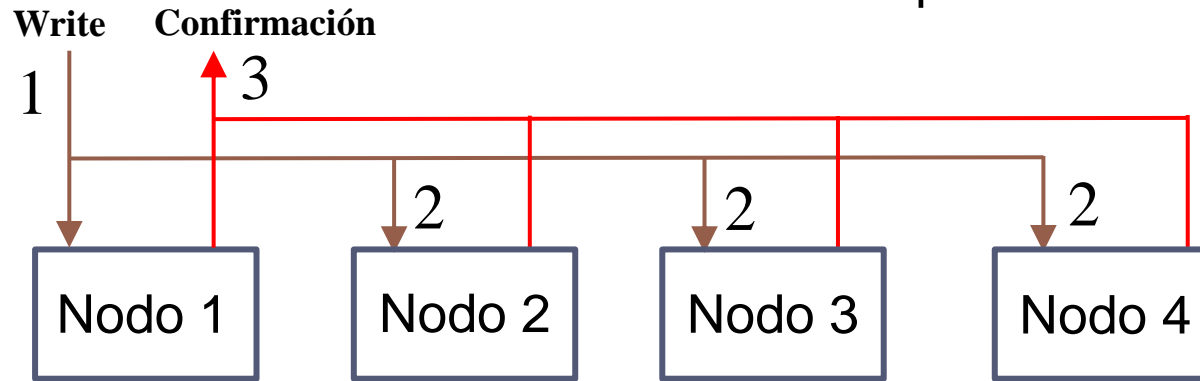
# Copia primaria (replicación pasiva)

- ▶ Para hacer frente a  $k$  fallos, se necesitan  $k+1$  copias
  - ▶ Un nodo **primario**
  - ▶  $K$  nodos de respaldo
- ▶ Funcionamiento SIN fallo:
  - ▶ **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

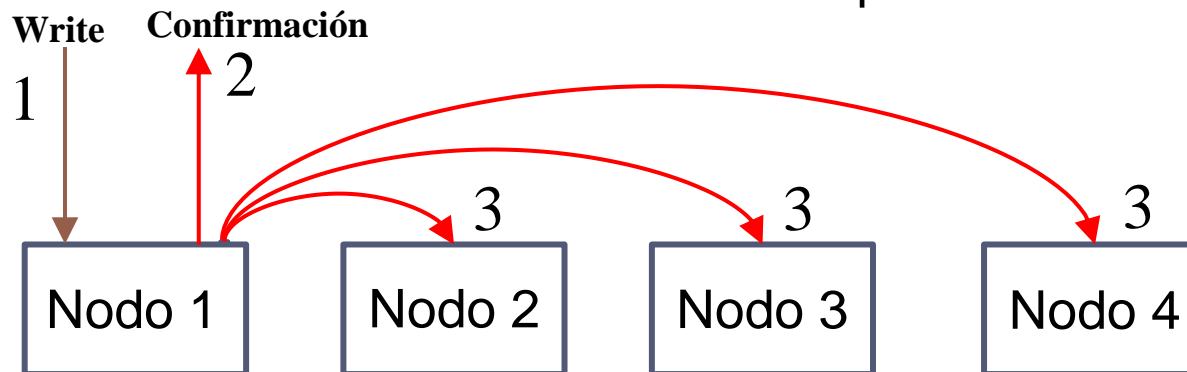


# Sincronización de réplicas

Replicación síncrona

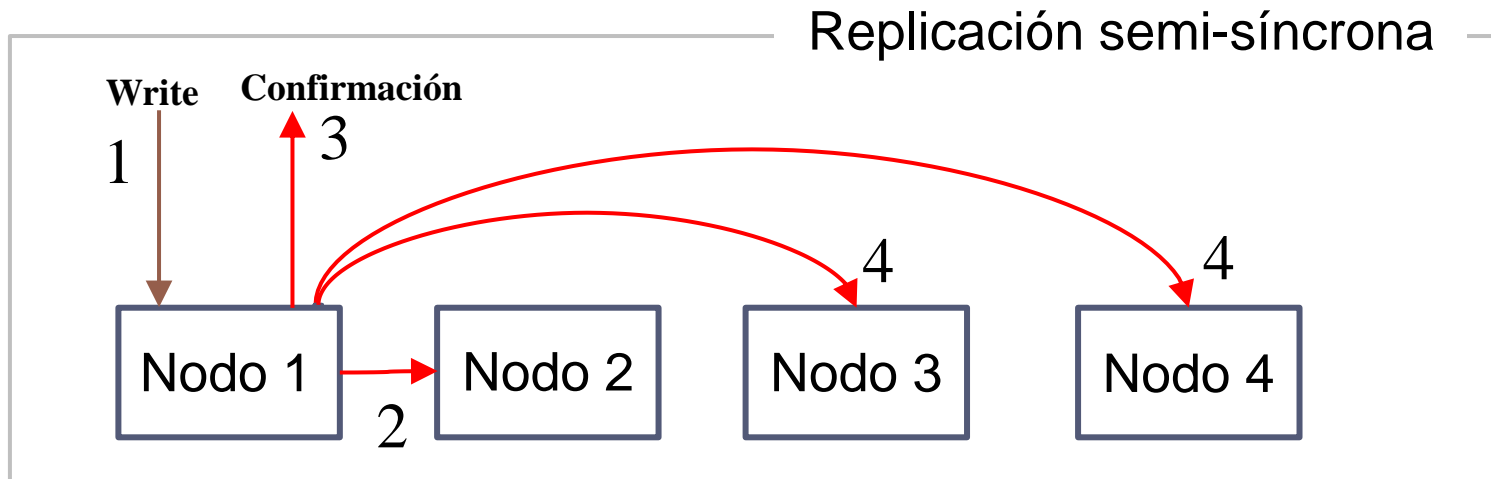


Replicación asíncrona





# Sincronización de réplicas



# Copia primaria (replicación pasiva)

- ▶ Para hacer frente a  $k$  fallos, se necesitan  $k+1$  copias

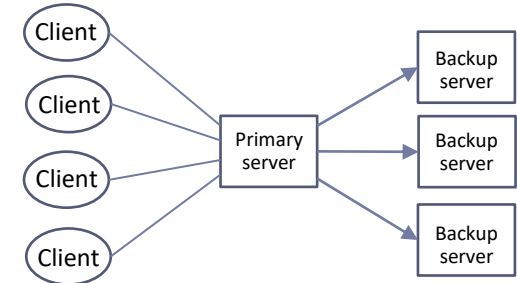
- ▶ Un nodo **primario**
- ▶  $K$  nodos de respaldo

- ▶ Funcionamiento SIN fallo:

- ▶ **Lecturas**: se envían a cualquier servidor

- ▶ **Escrituras**: se envían al primario

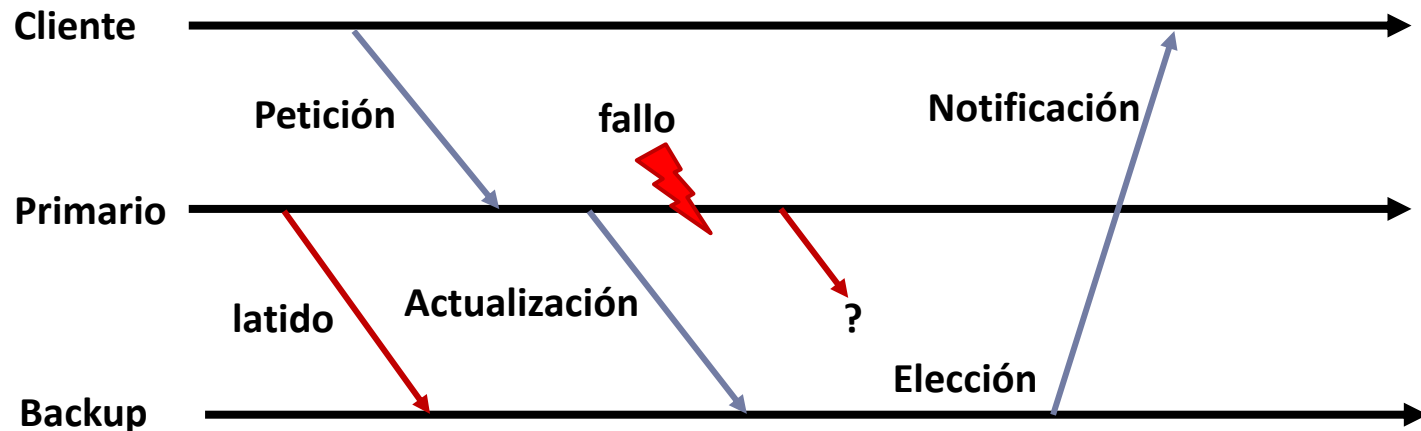
- ▶ El primario realiza la actualización y guarda el resultado
- ▶ El primario actualiza el resto de copias
- ▶ El primario responde al cliente
- ▶ Las escrituras solo son atendidas por el nodo primario



- ▶ Funcionamiento CON fallo:

- ▶ Falla primario: un nodo secundario lo releva (algoritmo de elección)
- ▶ Falla secundario: primario guarda los cambios, secundario tras arrancar le pide dichos cambios.

# Implementación con mensajes *heartbeat*



# Copia primaria (replicación pasiva)

## ► Fallo en un nodo primario

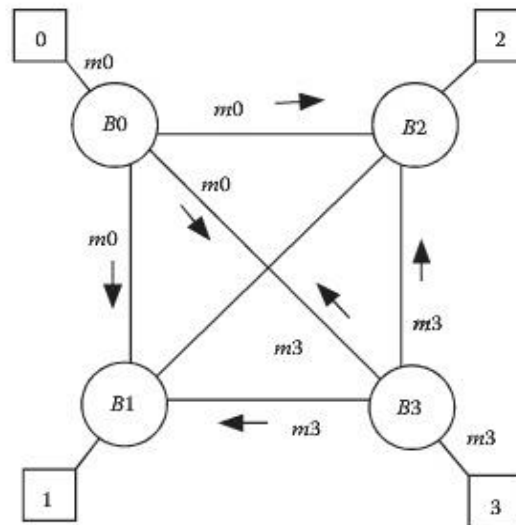
- Uno de los secundarios tiene que convertirse en primario:
  - Detectar el fallo en el nodo primario
  - Elegir un nuevo primario: algoritmo de elección
- Tienen que reconfigurarse los clientes para que envíen las nuevas escrituras al nuevo primario
- A este proceso se le denomina *failover*
- NOTA: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

## ► 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

# Réplicas activas

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

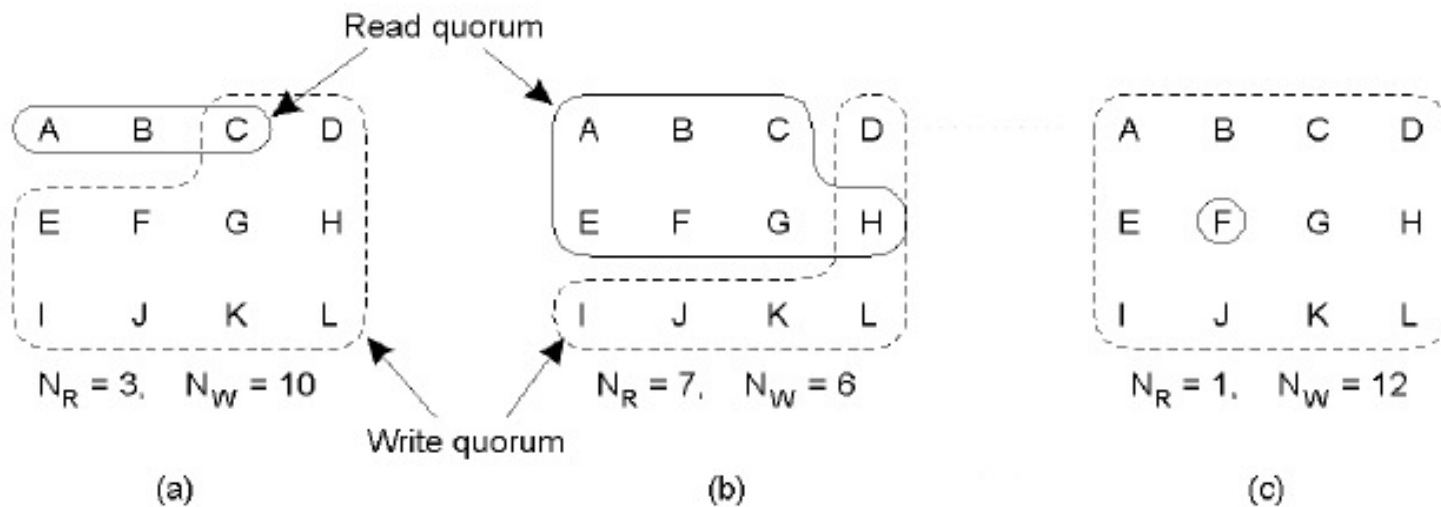


# Método de votación (quorum)

- ▶ Se definen dos operaciones **READ** y **WRITE**
- ▶ Hay un conjunto de nodos **N**, 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

/



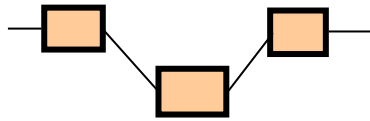
# ¿Cómo elegir W y R?

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



# Sistemas serie, paralelo y *k-out-of-n*

## repaso

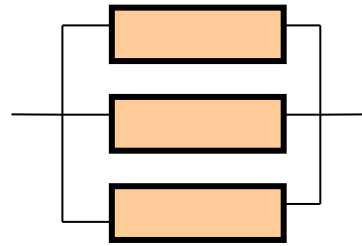


- ▶ El sistema falla cuando algún componente falla

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

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

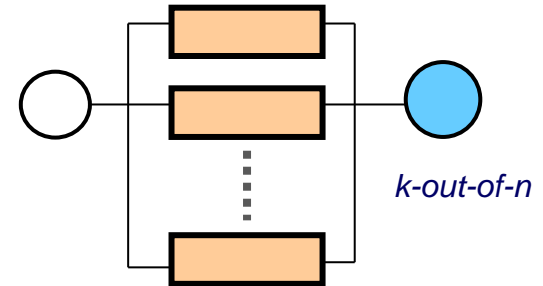
- ▶  $R_i(t)$  es la fiabilidad del componente  $i$
- ▶ Son fallos independientes (entre sí)
- ▶ La fiabilidad del sistema es menor



- ▶ El sistema falla cuando fallan todos los componentes

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

$$\text{donde } Q_i(t) = 1 - R_i(t)$$



- ▶ El sistema funciona cuando funcionan al menos  $k$  de  $n$

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

# Solución

- ▶  $N=7$
- ▶ 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

- ▶  $N=7$
- ▶ 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	$(0,05)^7$	$1-(0,95)^7$	$9,05 \cdot 10^{-02}$
2	6	5,0	$1-R(2,7)$	$1-R(6,7)$	$1,33 \cdot 10^{-02}$
3	5	5,1	$1-R(3,7)$	$1-R(5,7)$	$1,13 \cdot 10^{-03}$
4	4	5,2	$1-R(4,7)$	$1-R(4,7)$	$1,94 \cdot 10^{-04}$

¿Y para  $P_R = 60\%$ ?

$$\begin{aligned} \text{Coste} &= R * 1 * 0,7 + W * 2 * 0,3 \\ &= 2 * 0,7 + 6 * 0,6 = 5 \end{aligned}$$

# 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 (la última versión)
- ▶ 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  $W$

# 2PC: Two-phase commit

- Denominamos **coordinador** al proceso que realiza la operación

Coordinador:

multicast: *ok to commit?*

recoger las respuestas:

todos ok => *send(commit)*

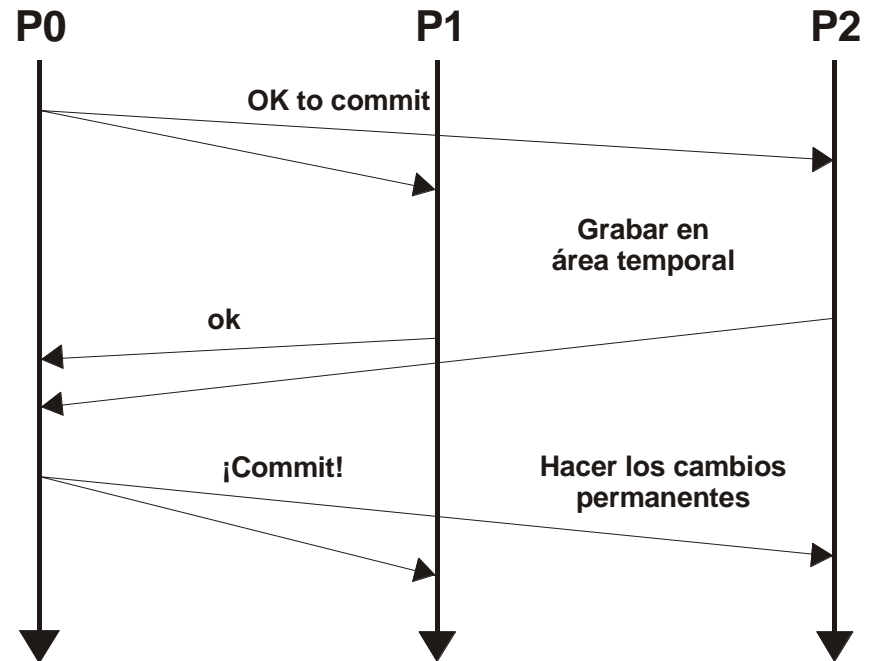
else => *send(abort)*

Procesos:

*ok to commit* => guardar la petición  
en un área temporal y  
responder *ok*

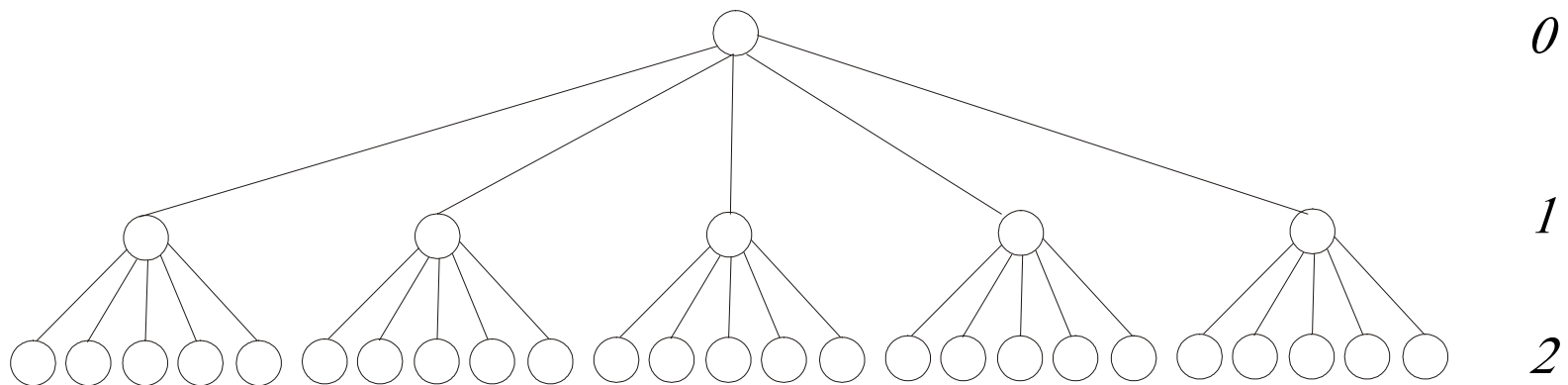
*commit* => hacer los cambios  
permanentes

*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 =  $5 \times 5 = 25$  (nodos hoja)



# Votación jerárquica

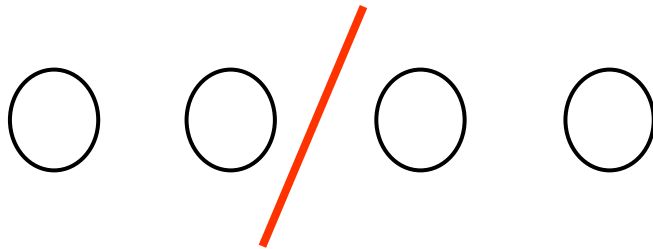
- ▶ Se realiza el **quorum** por niveles

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 (1/2)

- ▶ Cada dato  $d$  está soportado por  $N$  réplicas  $\{d_1 \dots d_n\}$
- ▶ Cada dato  $d_i$  en el nodo  $i$  tiene un **número de versión**  $VN_i$  (inicialmente 0)
- ▶ Se denomina VN actual o  $AVN(d) = \max\{VN_i\} \forall i$
- ▶ Una réplica  $d_i$  es actual si  $VN_i = AVN$
- ▶ Un grupo constituye una **partición mayoritaria** si contiene una mayoría de copias actuales de  $d$

# Método de votación dinámica (2/2)

- ▶ Cada copia  $d_i$  tiene asociado un número entero denominado **cardinalidad de actualizaciones**  $SC_i =$  número de nodos que participaron en la actualización
- ▶ Inicialmente  $SC_i = N$
- ▶ Cuando se actualiza  $d_i$ 
  - ▶  $SC_i =$  número 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

$I = \{i \text{ tal que } VNi = M\}$

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

**if**  $|I| \leq N/2$

then {

se rechaza la operación

(el nodo no pertenece a una partición mayoritaria)

}

else {

$\forall \text{ nodos } \in I$

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

Partición 1

Partición 2

# 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      Partición 2      Partición 3

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

# Ejemplo

- Tras actualización:

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

Partición 1      Partición 2      Partición 3

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

# Unión de un nodo a un grupo

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

$M = \max\{VN_i\}$

$I = \{A_j, \text{ tal que } M = VN_j\}$

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

**if**  $|I| \leq N/2$

then {

no se puede unir

}

else {

Actualiza su estado

$VN_i = M$

$SC_i = N + 1$

}



# Ejemplo

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

Partición 1      Partición 2      Partición 3

- ▶ 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

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

Partición 1      Partición 2      Partición 3

- 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

- ▶ Se une la partición 1 y 2

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

# Replicación basada en vectores de versiones

sistema de ficheros CODA

- ▶ Método de replicación **optimista**
- ▶ Cada réplica lleva asociado **un vector de versiones**  $V$  con  $n$  componentes = grado de replicación
- ▶ En el nodo  $j$ ,  $V[j]$  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 con  $V1$
- ▶  $V1$  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

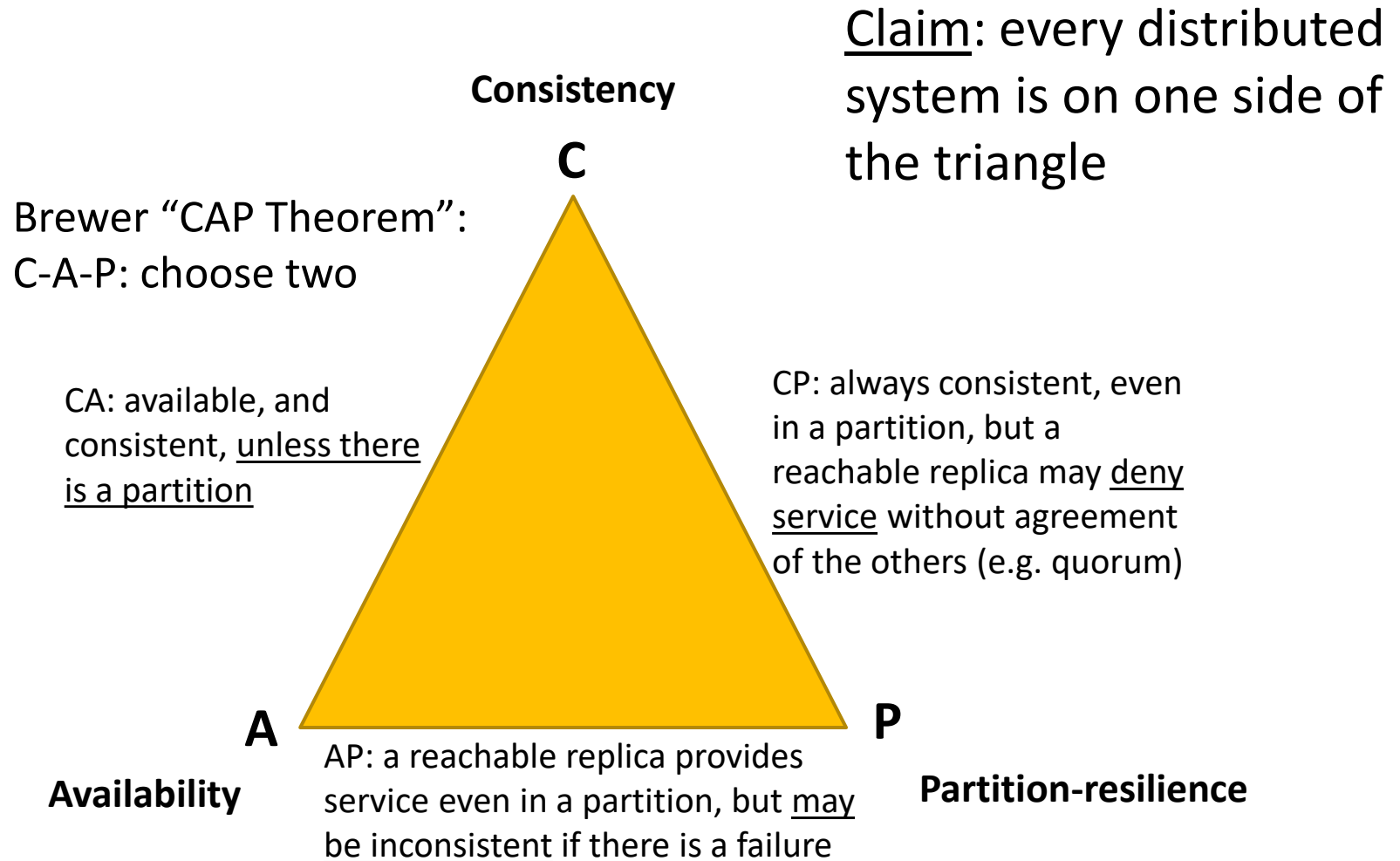
- ▶ Tres servidores = {A, B, C}
- ▶ Inicialmente  $V = (0,0,0)$  en los tres
- ▶ Cuando se realiza una actualización:  $V=(1,1,1)$  en los tres servidores
- ▶ Se produce un fallo de red:
  - ▶ Grupo 1: {A,B}
  - ▶ Grupo 2: {C}
- ▶ Se produce una actualización sobre el grupo 1
  - ▶  $V=(2,2,1)$  para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

# Ejemplo

- ▶ Se produce una actualización sobre el grupo 2
  - ▶  $V=(2,3,2)$  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





Grupo ARCOS



**uc3m** | Universidad **Carlos III** de Madrid

# Tema 9: Tolerancia a fallos

## **Sistemas Distribuidos**



Grado en Ingeniería Informática  
Grado en Matemática aplicada y Computación  
Doble Grado en Ingeniería Informática y Administración de Empresas