Sistemas Distribuidos I
(75.74)

Tolerancia a Fallos

Confiabilidad. Coordinación y Acuerdo. Transacciones distribuidas.

Docentes

- Pablo D. Roca
- Ezequiel Torres Feyuk
- Guido Albarello
- Ana Czarnitzki
- Cristian Raña

Agenda

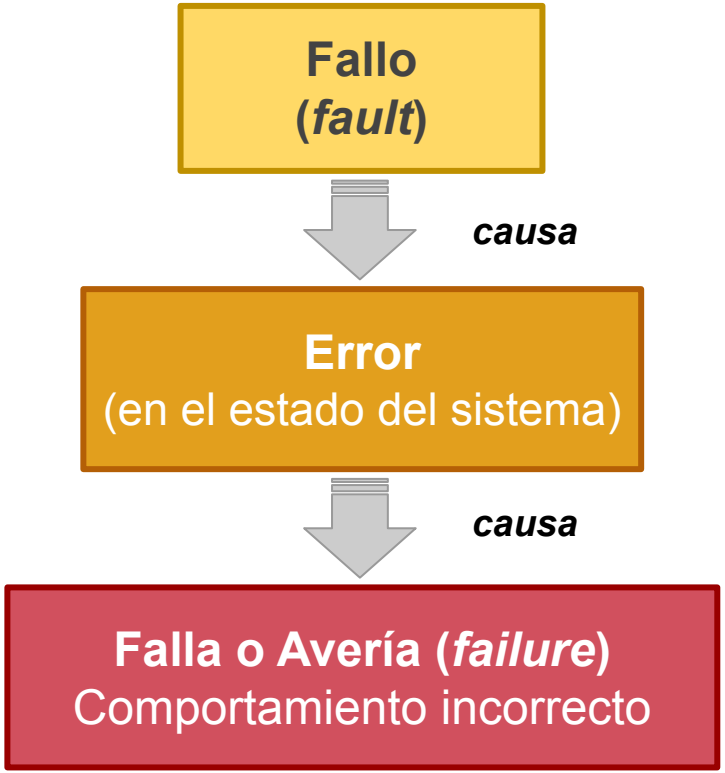
- **Tolerancia a Fallos**
- Confiabilidad
- Coordinación y Acuerdo
  - Exclusión Mutua Distribuida

Tolerancia a Fallos | Introducción

- Estudia las necesidades de sistemas confiables (*dependable systems*):
  - Garantizar que se ejecuten y comporten de acuerdo a lo esperado por el usuario en distintas condiciones (de uso y del entorno de ejecución del sistema)
  - Prevenir la aparición de fallas o averías de cara al usuario, tanto normales como excepcionales
  - Alternativas para prevenir como tolerar cada situación
- Permite definir la inversión y el nivel de tolerancia para cada tipo de sistema.
- Herramientas: replicación, votación, recuperación.

Fallo parcial:

- Cuando un componente de un sistema distribuido incurre en error.
- Característica distintiva de los sistemas distribuidos.
- Puede generar una reacción en cadena que afecta al comportamiento del sistema total.



Evento N°1:

- Fault: Rayo cósmico cambia estado bit en memoria RAM
- Error: Variable en heap apunta a dirección inválida
- Failure: Programa X accede a sección de memoria corrupta y muere

Evento N°2:

- Fault: Programa Y no puede comunicarse con programa X
- Error: Programa Y hace pooling infinito con un timeout pequeño a programa X que no responde dejando conexiones abiertas
- Failure: Programa Y llega al límite de conexiones abiertas y muere

"En presencia de fallos, el sistema distribuido continúa operando en forma aceptable"

Tolerancia a Fallos | Clasificación

- **Fallos** se clasifican en**:**
  - **Transientes**: ocurren una vez y luego desaparecen; si se repite la operación, el fallo desaparece
  - **Intermitentes**: ocurren en forma intermitente; difíciles de diagnosticar
  - **Permanentes:** existen hasta que el componente defectuoso se reemplaza
- Es importante entender la diferencia entre fallos improbables vs imposibles

Tolerancia a Fallos | Clasificación II

Crash. El servicio se detiene.

Timing. Respuesta fuera de los tiempos aceptables.

Omisión. El servicio falla al responder solicitudes entrantes.

Respuesta. La respuesta es incorrecta (valor incorrecto, desvío en el flujo).

Arbitraria o Bizantina. Arbitraria en tiempos y respuesta; diferente información para diferentes consumidores de esa información.

Tolerancia a Fallos | Condiciones

Para definir el nivel de tolerancia a fallos de un sistema, es necesario indicar en qué condiciones trabaja

Condiciones del entorno

- Entorno físico del hardware (temperatura, resistencia a vibraciones y polvo, ubicación)
- Interferencia y ruido
- *Clock drift*

Condiciones operacionales

- Especificaciones, valores límites y tiempos de respuesta (ej. de sensores)
- *Networking* (ancho de banda, latencia)
- Protocolos soportados

Fault Removal

Remover errores antes de que estos sucedan. Ej.: Detección de hardware errores antes que sucedan (ECC Memory)

Fault forecasting

Determinar la probabilidad de que un componente pueda llegar a fallar. Ej.: Reemplazo de piezas en un avión por tiempo de uso

Fault Prevention/Avoidance

Evitar las condiciones que llevan a la generación de errores. Ej.: Componentes que impidan que haya fallos (relojes atómicos, *military grade*)

Fault Tolerance

Procesar errores en el sistema y tratar los mismos en vez de evitar que sucedan

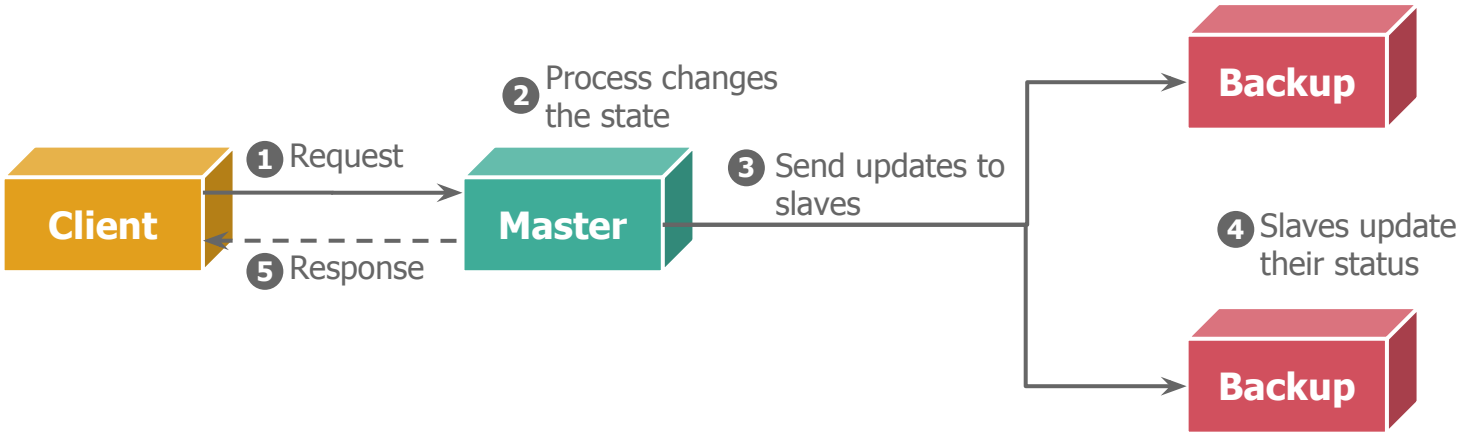
Tolerancia a Fallos | Resiliencia

- **Resiliencia:** mantener un nivel aceptable de servicio en presencia de fallos y desafíos a la operación normal:
  - Errores de configuración u operacionales
  - Desastre naturales (terremotos, inundaciones, huracanes)
  - Factores políticos, económicos, sociales, del negocio
  - Ataques maliciosos
- **Degradación suave (*graceful degradation*):** El comportamiento difiere del comportamiento en ausencia de fallos pero continúa siendo aceptable

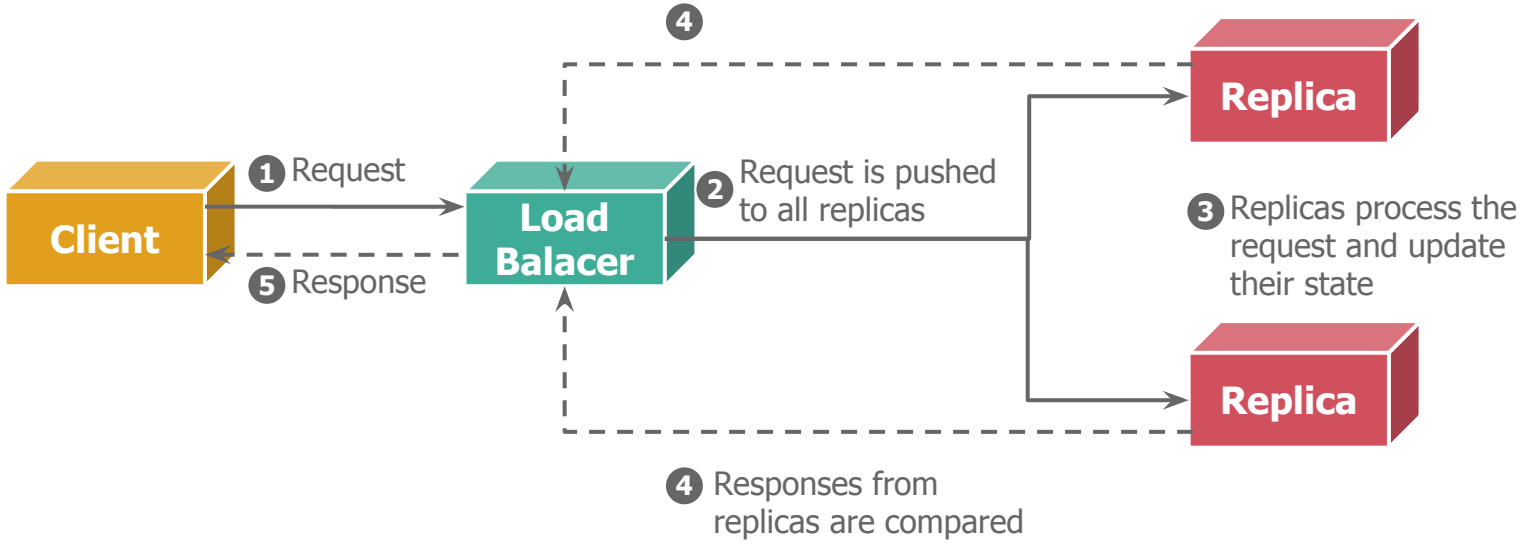
- **Se toleran fallos mediante redundancia:**
  - física (replicación), de información (valor), de tiempo (reintentos)
  - Ejemplo de la vida real: Rueda de auxilio. a) en la ciudad, b) en ruta, c) en el desierto, d) para un camión de carga
- **Punto único de fallo => Replicación:**
  - Varios tipos: activa, semiactiva (*leader-follower*), pasiva
  - Algoritmos de consenso permiten elegir componentes activos
  - Particionamiento de la red => muy dificultoso reconciliar estado automáticamente

- **Recuperarse de un error y llevar el sistema a un estado correcto**
  - **Almacenamiento estable**: almacenamiento seguro de la información
  - ***Checkpointing***: se guarda periódicamente el estado completo del sistema en almacenamiento estable
  - ***Message logging***: se parte de un checkpoint y se repiten todos los mensajes intercambiados desde ese checkpoint
  - **Consenso**: acordar el estado correcto

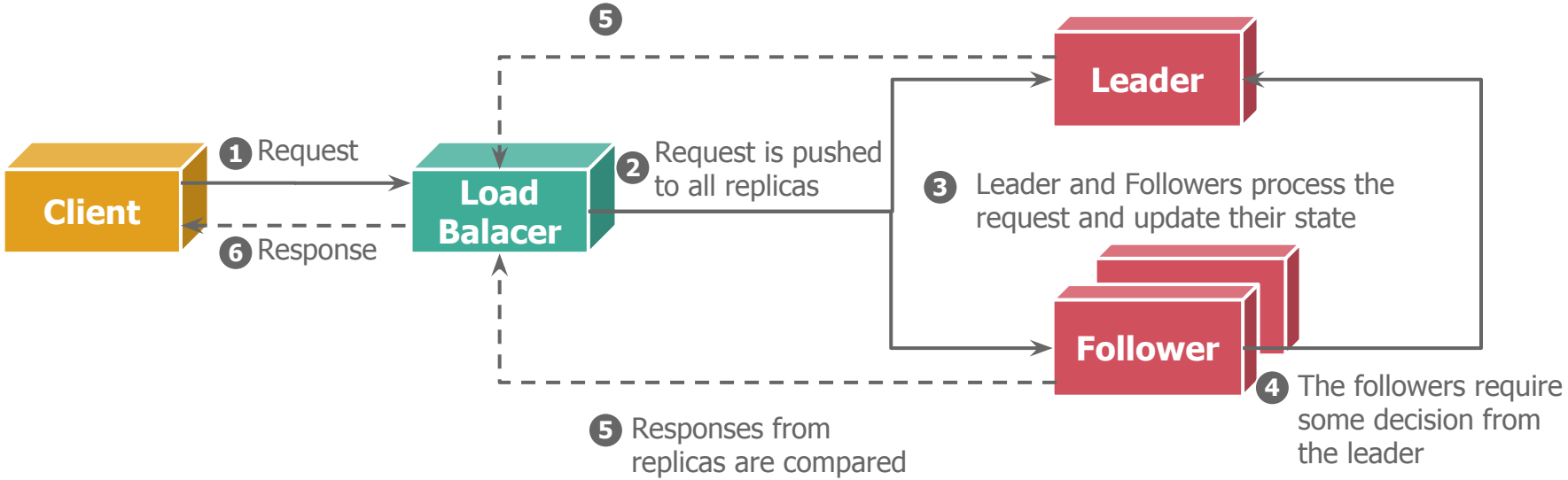
Pasiva: Una réplica primaria y varias secundarias o de backup



Activa: Múltiples réplicas de la misma máquina de estado que ejecutan las mismas operaciones en el mismo orden (por lo tanto: orden total)



Semi-activa (leader-follower): Todas las réplicas ejecutan los comandos pero una sola, el líder, toma las decisiones no determinísticas



Agenda

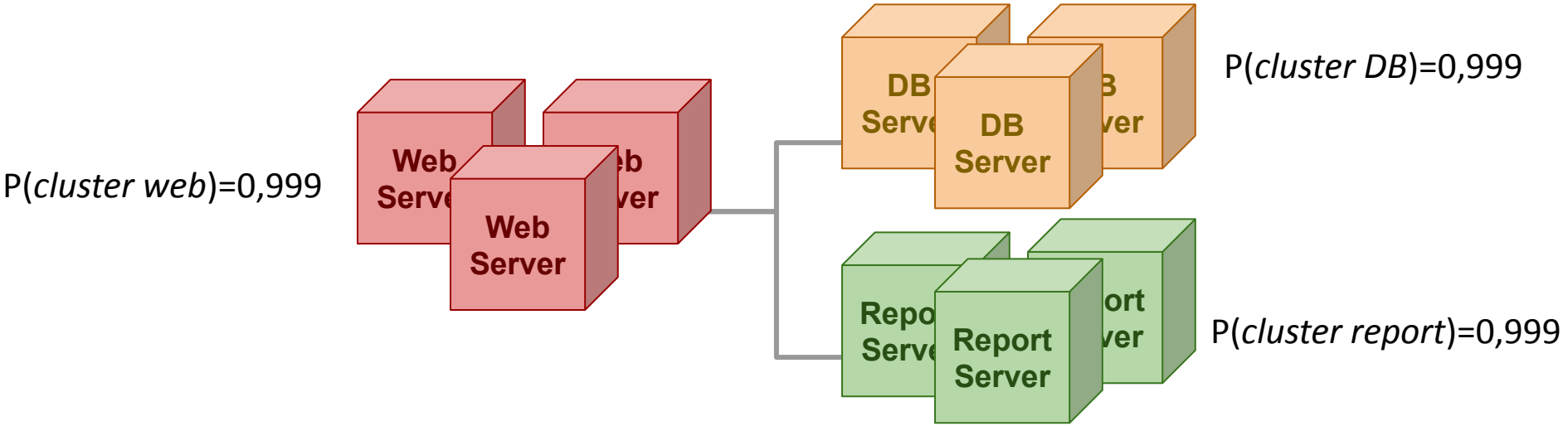
- Tolerancia a Fallos
- **Confiabilidad**
- Coordinación y Acuerdo
  - Exclusión Mutua Distribuida

Confiabilidad | Introducción

- **Dependability (*confianza*):** medida de la confianza en el sistema.
  - **Availability (*disponibilidad*)**: la probabilidad de que el sistema esté operando correctamente.
  - **Reliability (*fiabilidad*)**: capacidad del sistema para dar servicio correcto en forma continuada.
  - **Safety (*seguridad*)**: en presencia de fallos no ocurre nada catastrófico.
  - **Maintainability (*mantenibilidad*)**: la cantidad de tiempo que se requiera para actualizar/reparar el sistema.

Confiabilidad | Availability y Reliability

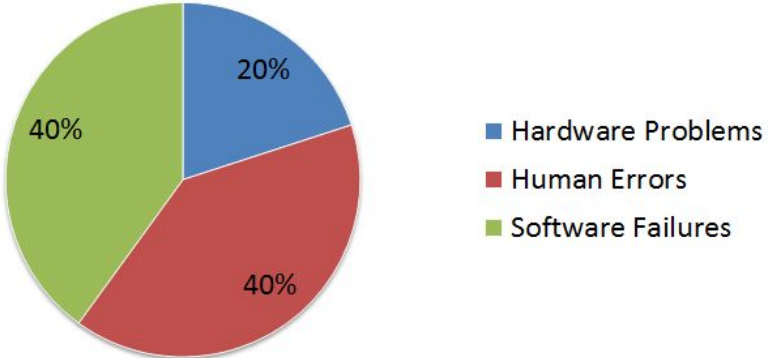
Escenario: Sistema distribuido con clusters para cada componente



P(*available*) = P(*cluster web*) . P(*cluster DB*) . P(*cluster report*)
= 0,999 . 0,999 . 0,999 = **0,997**

Confiabilidad | Availability y Reliability

- La mejor opción depende de:
  - Costos y presupuesto disponible
  - Necesidades de performance y escalabilidad
  - Necesidades heterogéneas de cada componente
- No todo es hardware
- Pensar en el origen de los errores



Fuente : http://www.cs.cmu.edu/~priya/PDL-CMU-05-109.pdf

Confiabilidad | Maintainability

- Creación de imágenes ante cada cambio a deployar (e.j. AMI, docker image)
  - Nueva versión de un paquete => Nueva imagen
  - Automatizado, no hay humanos "tocando" el server
- Testing de Images antes de deploy
- Configuración e imagen en ambiente de test es igual a la que es ejecutada en producción
- Imágenes viejas almacenadas ante posible rollback

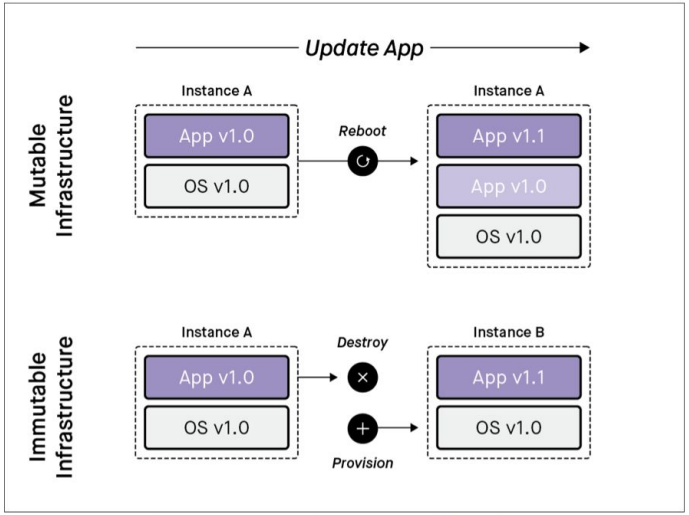
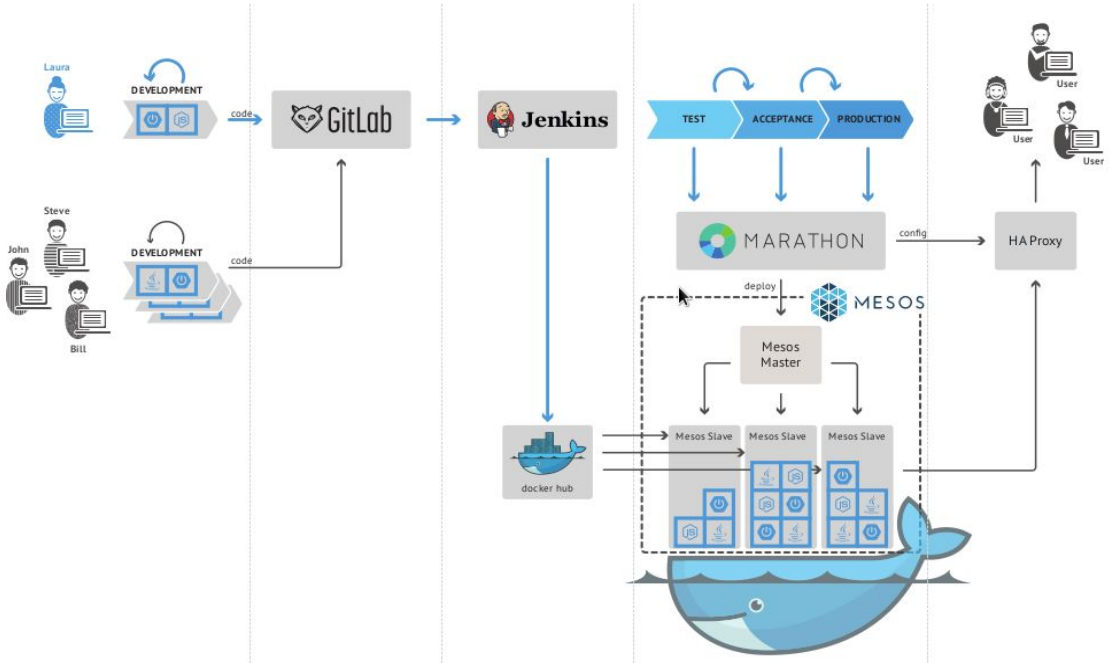


Figure 1-1. Updating via mutable vs. immutable infrastructures

Confiabilidad | Maintainability

- **Inmutabilidad**
  - Trazabilidad
  - Upgrades inmediatos
- **Resiliencia**
  - Alta Disponibilidad (HA)
  - Health Checks
- **Desacoplamiento**
  - Servicios (SaaS)
  - Plataforma (PaaS)
  - Infraestructura (IaaS)



Confiabilidad | Safety

"El sistema debe poder ser recuperado automática o manualmente ante cualquier tipo de falla"

- Qué sucede con las fallas catastróficas?
  - Necesidad de *Disaster Recovery Process* para infraestructura y datos
  - Tiempo debe ser conocido para poder garantizar SLAs
  - Escenarios catastróficos deben ser testeados
    - ¿Qué sucede con nuestro sistema si cada uno de los componentes cae (tenga o no replicación)?
    - ¿Qué sucede con la integridad de los datos?

Agenda

- Tolerancia a Fallos
- Confiabilidad
- **Coordinación y Acuerdo**
  - Exclusión Mutua Distribuida

Coordinación y Acuerdo | Introducción

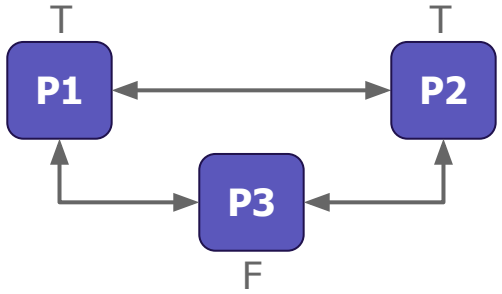
- **Objetivo**
  - Lograr que un conjunto de procesos pueda realizar un conjunto de tareas siguiendo una secuencia
  - Permitir la replicación de información
  - Evitar puntos únicos de fallo
- **Problemas a resolver**
  - Sincronización entre diferentes procesos
    - Un proceso debe esperar a otro para continuar
    - Recurso compartido requiere acceso exclusivo
  - Elección de un proceso coordinador (Líder)
  - Determinación del valor correcto de una propiedad

- Dado un conjunto de procesos distribuidos y un punto de decisión, todos los procesos deben acordar en el mismo valor

- **Problema complejo, require acotar variables:**
  - Canales de comunicación son *reliables*
  - Todos los procesos pueden comunicarse entre sí
  - Única falla a considerar es la caída de un proceso
  - Caída de un proceso no puede ocasionar la caída de otro

Proceso i

```
Init
  Values(i, 0) = {}
  Values(i, 1) = { <ValueProposed> }

For each round r, 1 < r <= f+1
  broadcast Values(i, r) - Values(i, r-1)
  Values(i, r+1) = Values(i, r)
  While round r still open
    receive Values from j in Values(j, r)
    Values(i, r+1) = Values(i, r+1) + Values(j,r)

//After f+1 rounds
decide d = aggregation function over Values(i, f+1)
```



Ejemplo ejecución:

P1:
Values(1)={T}
Values(2)={T,F,T}

P2:
Values(1)={T}
Values(2)={T,T}
Values(3)={T,T,F}

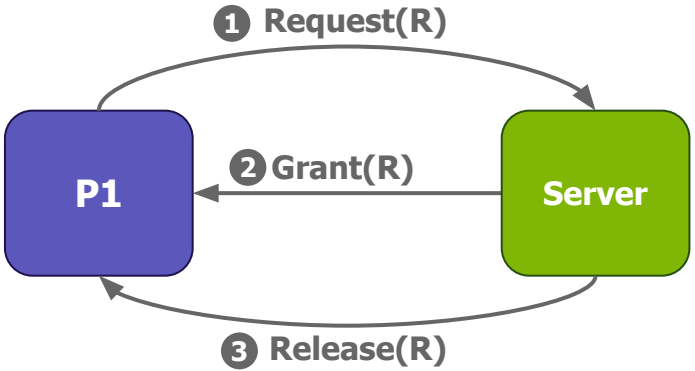
P3:
Values(1)={F}
Values(2)={F,T}
Values(3)={F,T,T}

Exclusión Mutua Distribuida | Introducción

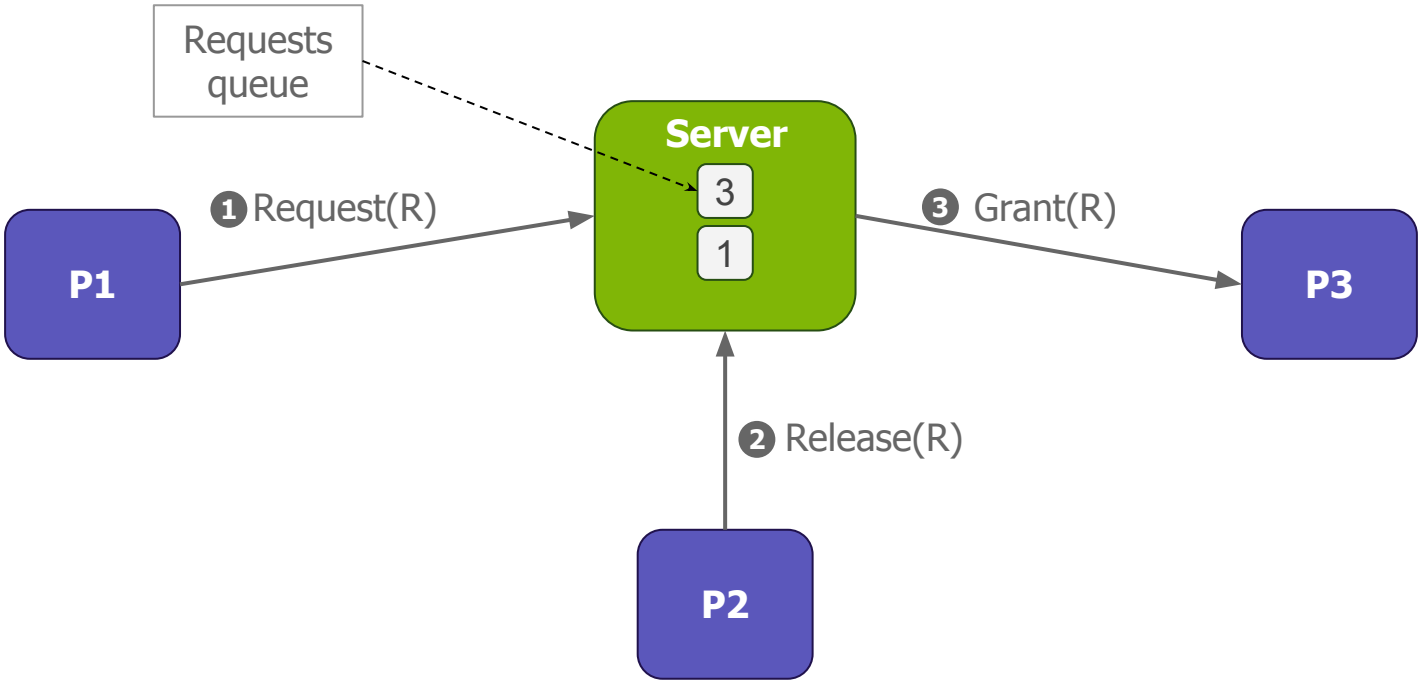
- **Objetivo**
  - Crear un algoritmo que permita pedir y obtener acceso exclusivo a un recurso que se encuentra disponible en la red
  - Utilizar **pasaje de mensajes** para lograr nuestro objetivo
- **Propiedades requeridas**
  - ***Safety***: Sólo un proceso puede obtener el recurso en todo momento
  - ***Liveness***: Procesos no deben esperar eternamente por mensajes que nunca van a llegar. *Starvation* debe ser evitada
  - ***Fairness***: Cada proceso posee la misma prioridad para obtener el recurso. Una vez obtenido el recurso, el mismo debe ser liberado luego de un tiempo conocido. *In-order processing*

Exclusión Mutua Distribuida | Servidor Central

- Alguno de los procesos en el sistema es elegido como el coordinador de la sección crítica
- Se sabe de antemano la forma en la que el recurso será identificado
  - Ej. string utilizado como ID
- Si el recurso se encuentra tomado, request son encolados (FIFO)
  - Procesos esperan a que el server les de acceso a la sección crítica
  - Acceso a sección crítica *time-bounded*



Exclusión Mutua Distribuida | Servidor Central



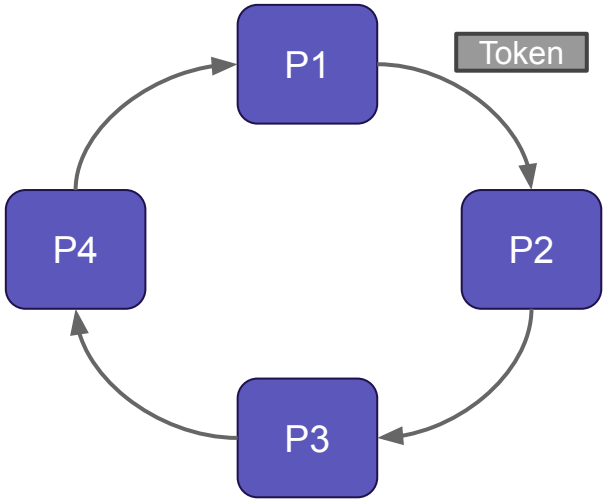
Exclusión Mutua Distribuida | Servidor Central

- **Ventajas**
  - *Requests* procesados en orden
  - Fácil de entender e implementar
  - Procesos sólo necesitan conocer al Server
  - Cantidad de conexiones y pasaje de mensajes mínimos

- **Desventajas**
  - Server es un punto único de falla
  - Procesos no pueden distinguir entre si el Server está caído o no responsivo
  - Cuello de botella

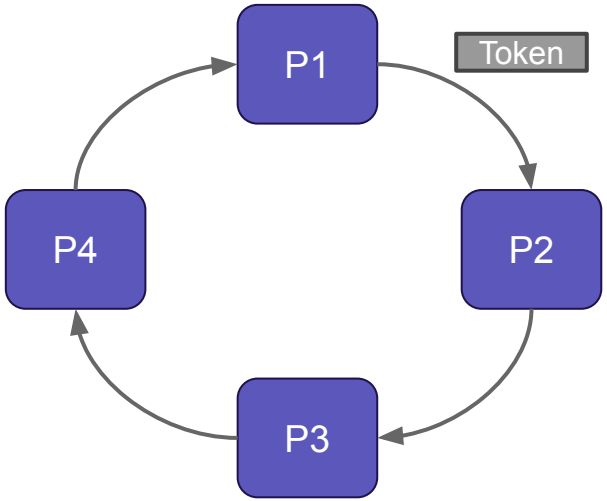
Exclusión Mutua Distribuida | Token Ring

- Anillo es construído ordenando a los procesos involucrados por algún atributo (MAC, IP, PID)
- Inicialización
  - Proceso 0 crea un token
- Token circula alrededor del anillo
  - De P_i a $P_{(i+1)}$ mod N
- Cuando un proceso recibe el token
  - Proceso chequea si requiere sección crítica
  - En caso negativo, pasa el token a su vecino
  - En caso afirmativo, accede al recurso reteniendo el token



Exclusión Mutua Distribuida | Token Ring

- **Ventajas**
  - Fácil de implementar y entender
  - No es necesario elegir un coordinador
  - Exclusión mutua garantizada (token)
- **Desventajas**
  - Creación del anillo de forma dinámica Requiere un protocolo en sí mismo
  - Caída de un proceso implica regeneración del anillo
  - Token puede ser perdido



Exclusión Mutua Distribuida | Ricart & Agrawala

- Algoritmo distribuido que utiliza *reliable multicast* y relojes lógicos
- **Cuando un proceso intenta acceder a la sección crítica**
  - Crea un *request* con un *timestamp* asociado al proceso, un ID y el nombre del recurso (<T, P_i, R>
  - Envía el *request* a todos los procesos en el grupo
  - Espera hasta que todos los procesos le den permiso de ingresar a la sección crítica (OK *response*)
  - Entra a la sección crítica

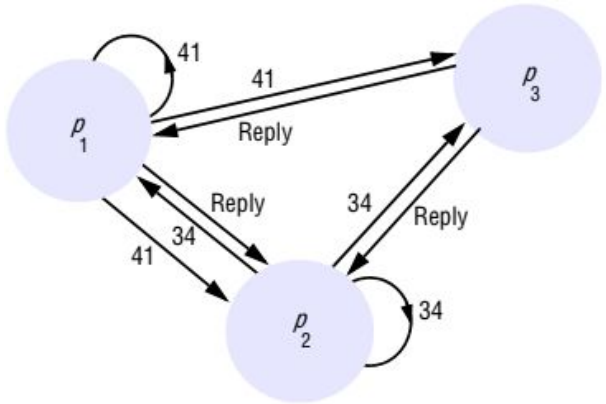
Exclusión Mutua Distribuida | Ricart & Agrawala

- **Cuando un proceso recibe un request**
  - Si el *receiver* no está interesado, envía un OK al *Sender*
  - Si el *receiver* posee la sección crítica, no responde al proceso *Sender y* encola el mensaje
  - Si el *receiver* envió también un request para acceder a la sección crítica
    - Se deben comparar los timestamp del mensaje enviado y recibido
    - Aquel que tenga un timestamp menor **gana**
    - Si el *receiver* es el perdedor, envía un OK al Sender
    - Si el *receiver* es el ganador, encola *request*
- **Si el *receiver* poseía la sección crítica**
  - Envía un OK a todos los requests encolados

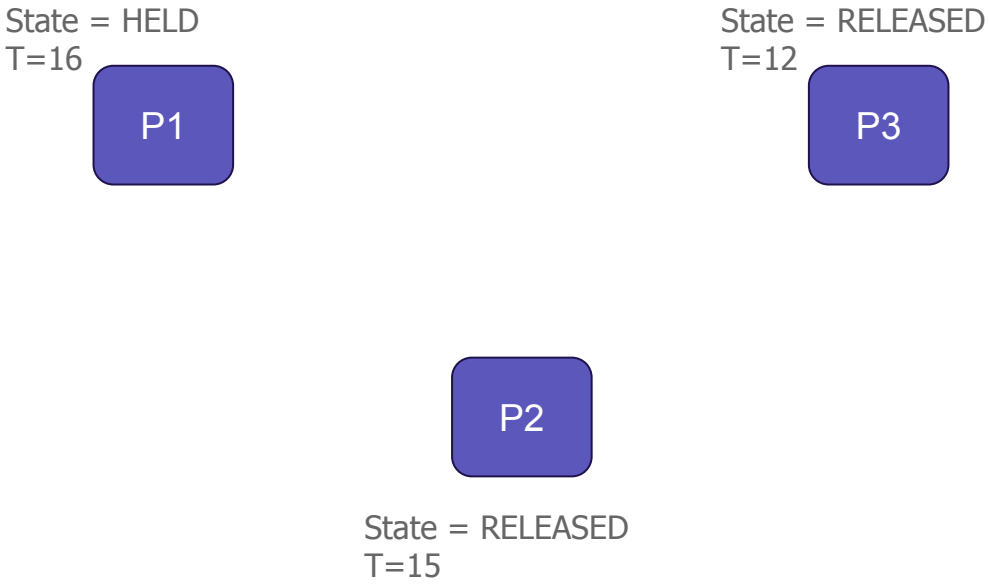
Exclusión Mutua Distribuida | Ricart & Agrawala

Proceso i

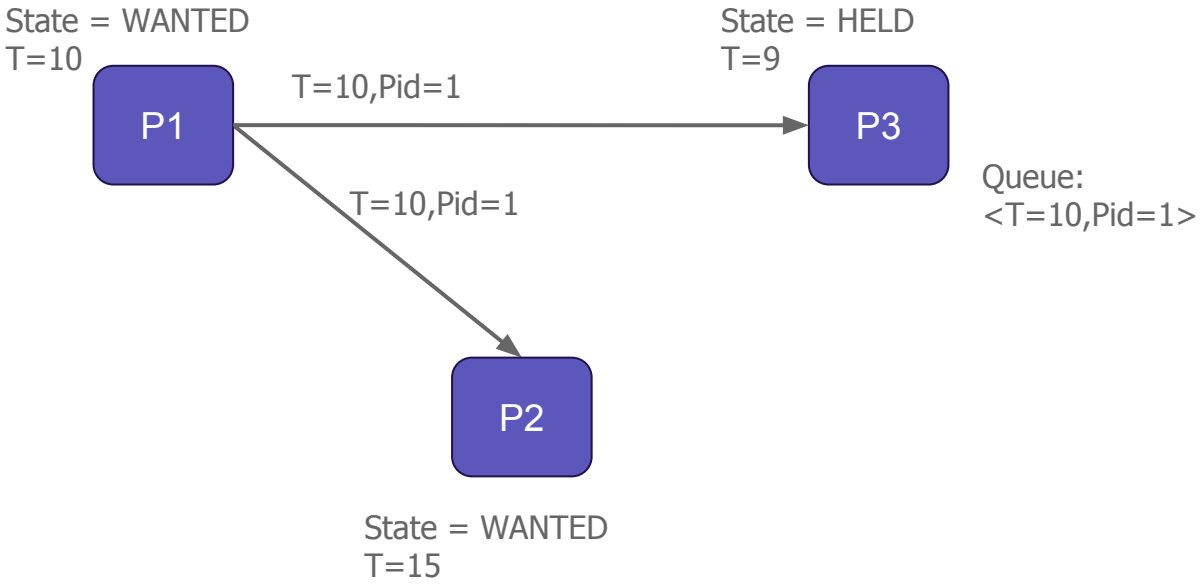
```
// On initialization
state := RELEASED;

// To enter critical section (P_i)
State := WANTED;
T := request's timestamp;
multicast_request(T,P_i);
state := HELD;

// On message Receipt <T_i, P_i> at P_j (i != j)
if (state == HELD OR (state == WANTED AND (T,P_j) < (T_i,P_i))):
    Queue request from P_i without replying
else:
     Reply immediately to P_i

// To exit critical section
State := RELEASED
for request in queue:
   send_OK
```



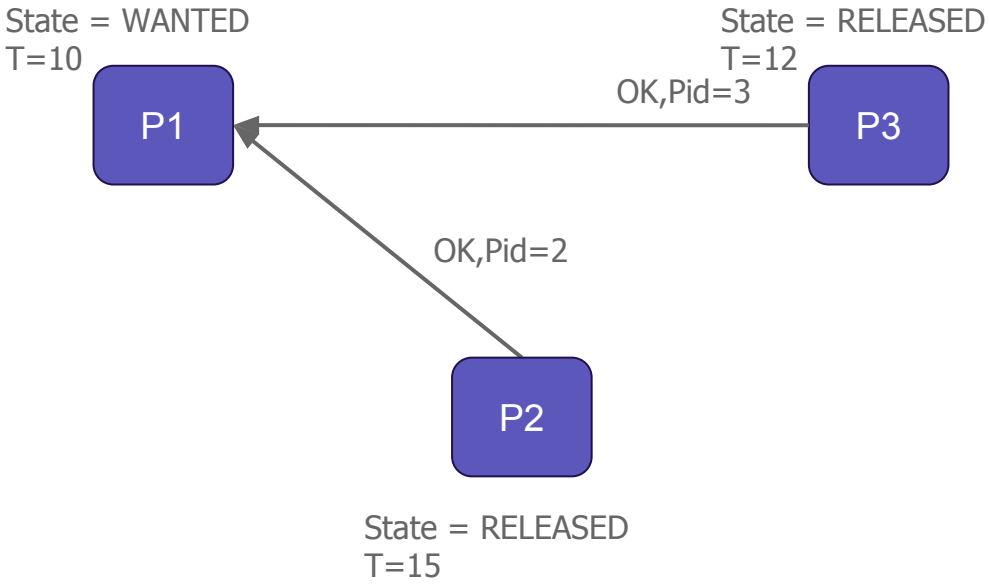
Exclusión Mutua Distribuida | Ricart & Agrawala



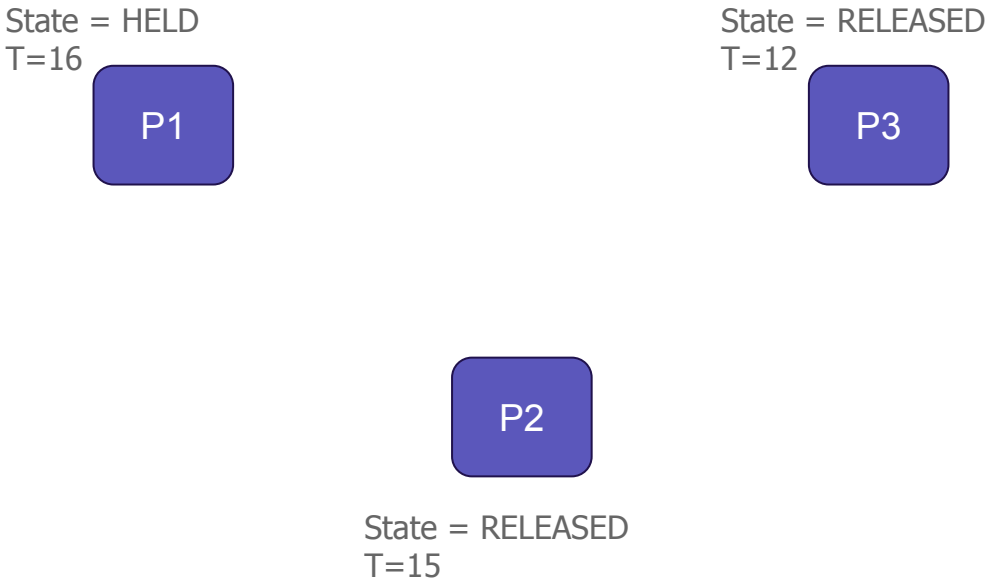
Exclusión Mutua Distribuida | Ricart & Agrawala



Exclusión Mutua Distribuida | Ricart & Agrawala

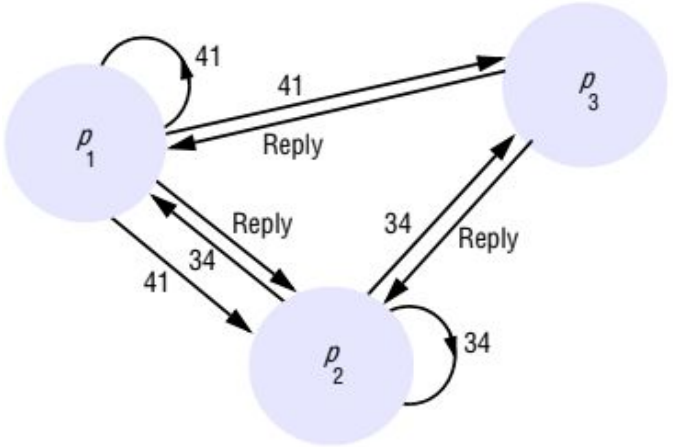


Exclusión Mutua Distribuida | Ricart & Agrawala



Exclusión Mutua Distribuida | Ricart & Agrawala

- **Ventajas**
  - N puntos de falla
  - No hay necesidad de tener un coordinador
- **Desventajas**
  - *Mesh* de conexiones: Todos los procesos deben conocerse
  - Cantidad de paquetes enviados para obtener sección crítica alta [2 (N-1)]
  - Imposible detectar entre proceso caído o proceso en sección crítica



Bibliografía

- P. Verissimo, L. Rodriguez: Distributed Systems for Systems Architects, Kluwer Academic Publishers, 2001.
  - Capítulo 6: Fault-Tolerant System Foundations
  - Capítulo 7: Paradigms for Distributed Fault Tolerance
  - Capítulo 8: Models of Distributed Fault-Tolerant Computing
- G. Coulouris, J. Dollimore, t. Kindberg, G. Blair: Distributed Systems. Concepts and Design, 5th Edition, Addison Wesley, 2012.
  - Capítulo 15: Coordination And Agreement
  - Capítulo 17: Replication