

# TOLERANCIA A FALLAS Y RECUPERACIÓN

Tema # VI

Sistemas de operación II

Abril-Julio 2008

Yudith Cardinale

# INDICE

- ◆ Tolerancia a fallas
- ◆ Conceptos Básicos de Tolerancia a Fallas
- ◆ Redundancia
- ◆ Recuperación de transacciones
- ◆ Listas de Intención
- ◆ Mecanismos de recuperación
  - ◆ Uso de logs
  - ◆ Versiones Sombras
- ◆ Acuerdos en sistemas que fallan

# Tolerancia a fallas

- ♦ En un sistema distribuido pueden ocurrir fallas parciales, en sistemas no distribuidos ocurren fallas totales
- ♦ La falla de un componente puede afectar el funcionamiento de otros componentes
- ♦ Uno de los objetivos de los sistemas distribuidos es recuperarse automáticamente de las fallas parciales sin afectar el rendimiento global
- ♦ Un sistema tolerante a fallas, soporta las fallas. Es imposible evitarlas

# Tolerancia a fallas

- Fallas

**WARNING!**

The system is either busy or has become unstable. You can wait and see if it becomes available again, or you can restart your computer.

- \* Press any key to return to Windows and wait.
- \* Press CTRL+ALT+DEL again to restart your computer. You will lose unsaved information in any programs that are running.

Press any key to continue \_

# Conceptos Básicos de TF

- ♦ Disponibilidad: propiedad del sistema de estar listo para ser usado en cualquier momento
- ♦ Confiabilidad: propiedad del sistema de ejecutar servicios continuamente sin presentar fallas
- ♦ Seguridad: si el sistema falla temporalmente, nada catastrófico suceda.
- ♦ Mantenibilidad: qué tan fácil se repara un sistema cuando falla.

Un sistema altamente disponible no necesariamente es altamente confiable.

La recuperación automática no es tan fácil!!

# Conceptos Básicos de TF

Falla del Sistema <==== Error <==== Falla externa

Un sistema falla (system failure) cuando no cumple con los objetivos, no satisface a sus usuarios

Un error es una parte del estado de un sistema que produce una "falla del sistema"

La causa de un error es llamada "falla externa (fault)"

Determinar una "falla externa (fault)" es importante, pero no siempre ayuda (cambiar las condiciones del tiempo para prevenir un error, no parece una opción!!!!)

# Conceptos Básicos de TF

- ♦ Los Sistemas Tolerantes a fallas no previenen las fallas, más bien las controlan.
- ♦ Los Sistemas Tolerantes a fallas proveen sus servicios aún en presencia de fallas
- ♦ Clasificación de las “Fallas Externas (*Faults*)”:
  - ♦ Transitorias: ocurren una vez y desaparecen
  - ♦ Intermitentes: ocurren, se desvanecen, ocurren nuevamente y así sucesivamente.
  - ♦ Permanentes: la falla permanece.

# Conceptos Básicos de TF

- Modelos de Fallas del Sistema (Failure Models)

Fallas por Muerte (Crash)
Fallas por Omisión: Omisión de recepción Omisión de envío
Fallas Temporales (timeouts)
Fallas por Respuestas: Fallas de valor Fallas de transición de estados
Fallas Arbitrarias (bizantinas)

# Conceptos Básicos de TF

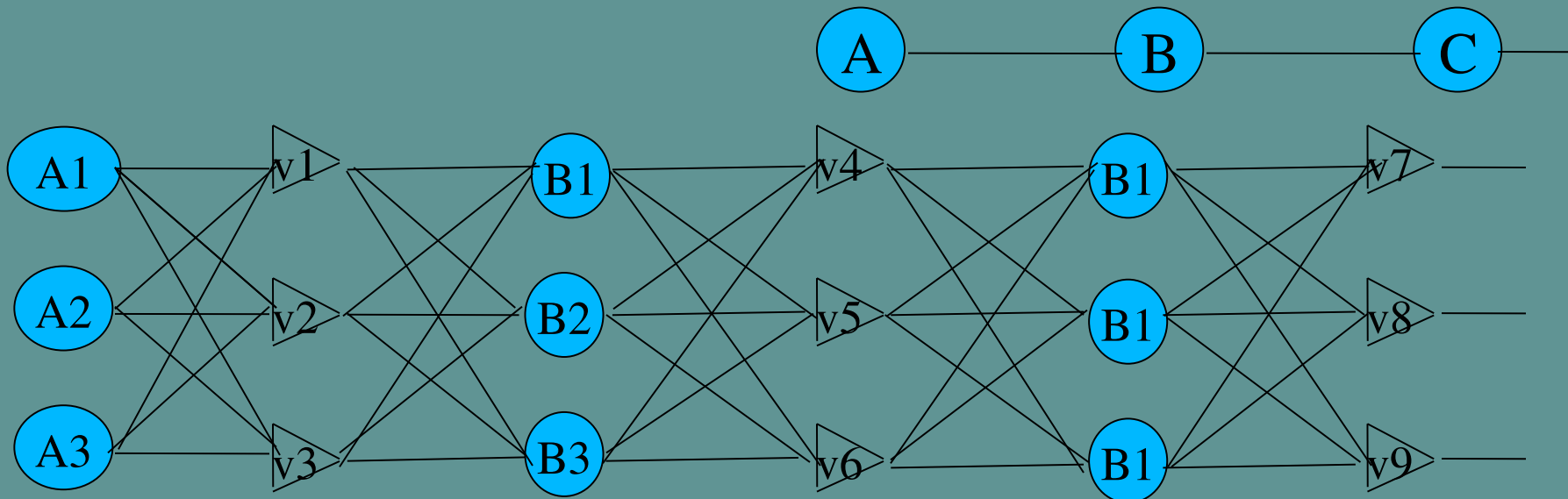
- Modelos de Fallas del sistema (Failure Models)
  - Fail-stop failures: los servidores producen salidas que pueden pronosticar su caída
  - Fail-silent failures: la caída se produce sin anuncios (se puede confundir con servidores lentos)
  - Fail-safe failures: el servidor produce salidas que fácilmente se reconocen como fallas
  - Byzantine failures: el servidor produce salida errónea que se confunde con salida correcta.

# Redundancia

- La técnica más usada para enmascarar las fallas es la Redundancia.
- Hay tres tipos de redundancia:
  - De información: se agrega información adicional para detectar fallas (código de Hamming, bits de paridad, etc.)
  - De tiempo: una acción se puede repetir si es necesario (modelo de transacciones)
  - Física: se adicionan equipos o procesos extras para sustituirlo por el componente que falle

# Redundancia

- La redundancia física es la más usada para proveer tolerancia a fallas: en biología, aviones, deportes y circuitos electrónicos
- El modelo de circuitos electrónicos TMR (Triple Modular Redundancy) es un buen modelo para los sistemas distribuidos



# Redundancia

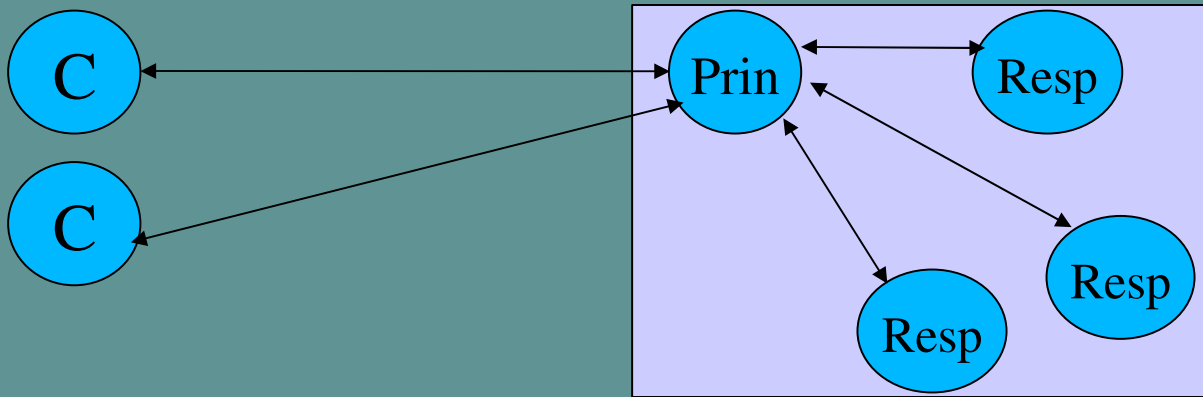
- Los grupos de procesos y la comunicación en grupo pueden usarse para implementar redundancia
- ¿Cuánta replicación es necesaria si se usan grupos de procesos?
- Un sistema es **k tolerante a fallas** si puede continuar funcionando aún si fallan **k** **componentes**
  - **Para soportar fallas silenciosas es necesario ??**
  - **Para soportar fallas bizantinas es necesario ??**
- **Para la redundancia con grupo de procesos es vital el multicast atómico**

# Redundancia

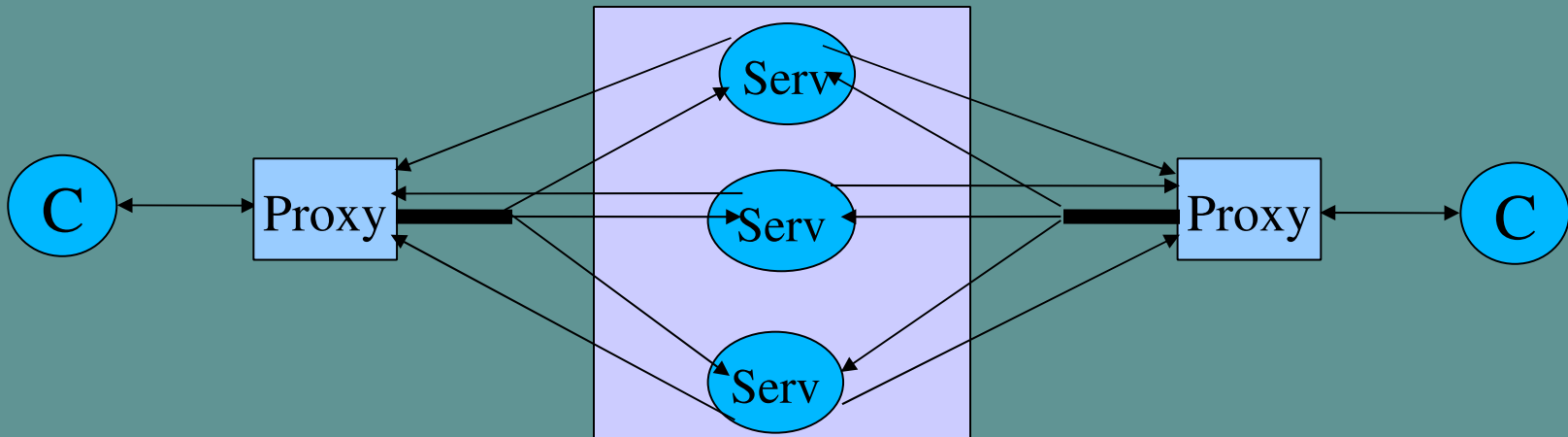
- Existen dos enfoques de replicación por grupos:
  - Protocolo basado en primario (primary based o primary-backup): también se llama replicación pasiva y se implementa con grupos jerárquicos
  - Protocolo de escritura replicada (replicated-write): también se llama replicación activa y se implementa con grupos planos (es el caso de TMR)

# Redundancia

- Replicación Pasiva (principal-respaldos):



- Replicación Activa (escrituras-replicadas)



# Redundancia

- Replicación Pasiva (principal-respaldo):  
Pasos
  - **Petición: va del cliente al principal**
  - **Coordinación: el principal acepta las peticiones en forma atómica y en orden**
  - **Ejecución: el principal ejecuta la petición y guarda la respuesta**
  - **Acuerdo: en caso de actualización se envían los cambios a todos los respaldos (atómicamente)**
  - **Respuesta: el principal responde al cliente**

# Redundancia

## Replicación Pasiva: características

- Requieren algoritmos de elección de coordinador cuando el principal falla
- Dado que el coordinador es quien ejecuta todas las peticiones, es necesario identificar los requerimientos de manera que el nuevo principal pueda detectar retransmisiones
- **La desventaja: sobrecarga relativamente grande para el principal**
- **Para descargar al principal, las lecturas pueden satisfacerlas los respaldos**
- **¿Puede soportar fallas silenciosas y bizantinas?**

# Redundancia

- Replicación Activa (escrituras-replicadas):  
Pasos
  - **Petición:** el cliente/proxy envía el requerimiento al grupo (multicast)
  - **Coordinación:** el sistema de comunicación en grupo reparte la petición (multicast atómico y ordenamiento)
  - **Ejecución:** todos ejecutan la petición
  - **Acuerdo:** no es necesaria si la comunicación es atómica y ordenada
  - **Respuesta:** todos envían la respuesta al cliente/proxy

# Redundancia

- Replicación Activa: características
  - **El cliente/proxy puede obtener la primera y obviar las demás respuestas o puede decidir por la respuesta más común entre las que recibe.**
  - **Provee consistencia secuencial**
  - **¿Puede soportar fallas silenciosas y bizantinas?**

# Recuperación de Transacciones

- ♦ Se refiere a asegurar la *atomicidad de las transacciones* aún en presencia de fallas del servidor.
- ♦ La propiedad de *atomicidad* de las transacciones se expresa en dos aspectos: *durabilidad* y *atomicidad en las fallas*:
  - ♦ *La durabilidad* requiere que los *ítemes* de datos sean salvados en almacenamiento permanente y disponibles indefinidamente.
  - ♦ *La atomicidad en las fallas* requiere que los efectos de las transacciones sean atómicos aún cuando el servidor falle.

# Recuperación de Transacciones

- ♦ Cuando un servidor se está ejecutando mantiene sus *ítemes* de datos en memoria volátil y registra los comprometidos (*committed*) en *archivos de recuperación*.
- ♦ Así, la *recuperación* consiste en restaurar al servidor con las últimas versiones comprometidas de sus *ítemes* de datos, a partir del almacenamiento permanente.
- ♦ Los requerimientos de *durabilidad y atomicidad en las fallas* no son independientes y se tratan con el mismo mecanismo : *El Administrador de Recuperación (AR)*.

# Funciones del *Administrador de Recuperación*

- ♦ Salvar los *ítemes* de datos en almacenamiento permanente, en un *archivo de recuperación*, de las transacciones comprometidas.
- ♦ Restaurar los *ítemes* de datos del servidor después de una caída.
- ♦ Reorganizar el *archivo de recuperación* para mejorar el desempeño en la recuperación.
- ♦ Recobrar espacio de almacenamiento en el *archivo de recuperación*.

# Listas de Intención y Archivos de Recuperación

- ♦ Usadas para que cada servidor mantenga el registro de los *ítemes* de datos accedidos por las transacciones.
- ♦ Durante el progreso de la transacción las operaciones de actualización son aplicadas a un conjunto privado de versiones tentativas.
- ♦ Cada servidor registra una *lista de intenciones* de todas sus transacciones activas.
- ♦ Una *lista de intención* de una transacción particular contiene la lista de nombres y valores de los *ítemes* de datos que son alterados por tal transacción.

# Listas de Intención y Archivos de Recuperación

- ♦ Cuando una transacción se compromete (ejecuta *commit*), el servidor usa la *lista de intenciones* de la transacción para identificar los *ítemes* de datos afectados.
- ♦ La versión comprometida de cada *ítem* de dato es reemplazada por la versión tentativa de tal transacción y el nuevo valor es escrito al *archivo de recuperación* del servidor.
- ♦ Cuando una transacción aborta, el servidor usa la *lista de intención* para borrar todas las versiones tentativas de la transacción.

# Listas de Intención y Archivos de Recuperación

- ◆ Entradas en el *archivo de recuperación* :

<b>Tipo de entrada</b>	<b>Descripción</b>
Item de dato	Valor del ítem de dato
Estado de la Transacción	Tid, estado de la Transacción ( <i>ready, wait, commit, abort</i> ), otros valores.
Lista de intención	Tid y secuencia de intenciones que consisten de: <id del dato>, <posición del valor del ítem en archivo de recuperación>

# Mecanismos de Recuperación: *logging*

- ♦ En esta técnica el *archivo de recuperación* representa un *log* que contiene la historia de todas las transacciones realizadas por un servidor.
- ♦ La historia consiste de valores de *ítems* de datos, entradas de los estados de la transacción y las *listas de intención* de la transacción.
- ♦ El orden de las entradas en el *log* refleja el orden en el cual las transacciones han sido *preparadas, comprometidas o abortadas* en ese servidor.
- ♦ En la práctica el *archivo de recuperación* contendrá una *foto instantánea* de los valores de todos los *ítems* de datos en el servidor, seguido por una historia de transacciones después de la *foto instantánea*.

# Ejemplo del mecanismo de *logging*

## Transaction T

bank&retirar(A,4)  
bank&depositar(B,4)

balance=A.read()      \$100  
A.write(balance-4)      \$96

balance=B.read()      \$200  
B.write(balance+4)      \$204

## Transaction U

bank&retirar(C,3)  
bank&depositar(B,3)

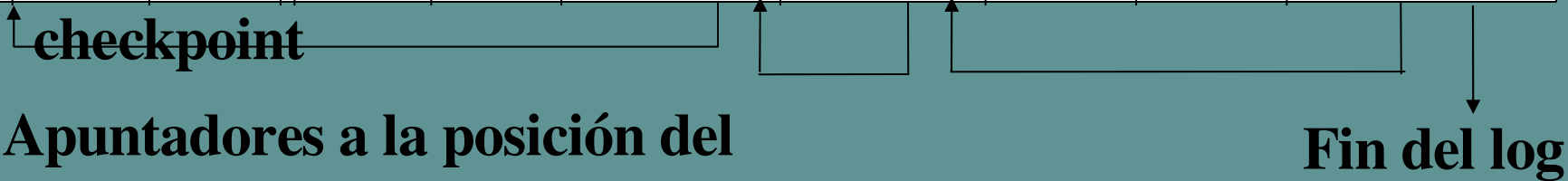
balance=C.read()      \$300  
C.write(balance-3)      \$297

balance=B.read      \$204  
B.write(balance+3)      \$207

# Ejemplo del mecanismo de *logging*

Antes de T y U      T y U comenzaron

P0			P1	P2	P3	P4	P5	P6	P0
Dato A 100	Dato B 200	Dato C 300	Dato A 96	Dato B 204	Trans T prepared <A,P1> <B,P2> P0	Trans T commit P3	Dato C 297	Dato B 207	Trans U prepared <C,P5> <B,P6> P4



Apuntadores a la posición del estado previo de la transacción

# Proceso de recuperación usando *logging*

- Durante la operación normal de un servidor, su *administrador de recuperación* es llamado cuando:
  - Una transacción se prepara para comprometerse, en cuyo caso adiciona todos los *ítemes* de datos en su *lista de intención* al *archivo de recuperación*, seguido por el estado actual (*prepared, ready*) junto con su *lista de intención*.
  - Una transacción se compromete o aborta, en este caso adiciona el estado correspondiente a su *archivo de recuperación*.
  - Se supone que la operación de adicionar una entrada al *archivo de recuperación* es atómica. Si el servidor falla sólo la última escritura puede estar incompleta.

# Proceso de recuperación usando *logging*

- Cuando un servidor es restaurado:
  - a) Coloca los valores iniciales por defecto de sus *ítemes* de datos.
  - b) Llama al *Administrador de Recuperación*.
  - c) El *Administrador de Recuperación* restaura los *ítemes* de datos del servidor de manera que incluyan todos los efectos de las transacciones comprometidas ejecutadas en el orden correcto y ninguno de los efectos de las transacciones incompletas o abortadas.
- La información más reciente sobre las transacciones está al final del *log*, así que el *administrador de recuperaciones* restaurará los *ítemes* de datos recorriendo el log desde el final hacia el principio.

# Reorganización del archivo de recuperación

- El *administrador de recuperaciones* tiene la responsabilidad de organizar su *archivo de recuperación* de manera que el proceso de recuperación sea rápido y reducir el uso de espacio.
- Conceptualmente la única información requerida para la recuperación es una copia de las versiones comprometidas de todos los *ítemes* de datos del servidor.
- ***Checkpointing***: se refiere al proceso de escribir los valores actuales comprometidos de los *ítemes* de datos de un servidor a un nuevo archivo de recuperación, junto con las entradas de los estados de transacción y las *listas de intención* de las transacciones que aún no han sido resueltas completamente.

# Reorganización del archivo de recuperación

- *Checkpoint* : se refiere a la información almacenada por el proceso *checkpointing*.
- El propósito de hacer *checkpointing* es reducir el número de transacciones con las cuales tratar durante la recuperación y recobrar espacio.
- ¿Cuándo hacer *checkpointing*?
  - a) Después de una recuperación
  - b) Antes que una nueva transacción comience
  - c) Periódicamente durante la actividad normal del servidor (si el proceso de recuperación no es muy frecuente)
- El *checkpoint* es escrito a un *archivo de recuperación* futuro y el *archivo de recuperación* actual permanece activo durante el *checkpointing*.

# Proceso de *Checkpointing*

**Archivo de  
recuperación  
actual**

...

Marca de checkpoint

**Pasó durante  
el  
chekpointing**

Estados no comprometidos

*Items de  
datos del  
servidor D*

**Archivo de  
recuperación  
futuro**

**Foto de D**

**Pasó durante  
el  
chekpointing**

## Mecanismos de Recuperación: *versiones sombras*

- Es una forma alterna de organizar el *archivo de recuperación*.
- Usa un *mapa* para localizar versiones de los *ítemes* de datos en un archivo llamado *almacenamiento de versiones* (*version store*)
- El *mapa* asocia los identificadores de los *ítemes* de datos del servidor con la posición de sus versiones actuales en el archivo de *almacenamiento de versiones*.
- Las versiones escritas por cada transacción son llamadas *versiones sombras* mientras la transacción no se comprometa.

## Mecanismos de Recuperación: *versiones sombras*

- Las entradas de los estados de transacción y las *listas de intenciones* son salvadas en el archivo de *estado de transacción*.
- La *lista de intención* representa la parte del mapa que será alterada por una transacción cuando se comprometa.
- Cuando una transacción se compromete, se crea un nuevo mapa copiando el mapa viejo e introduciendo las posiciones de las versiones sombras. Al completarse el compromiso se reemplaza el viejo mapa por el nuevo. Esta operación debe ser atómica.

## *Ejemplo de versiones sombras*

### Transaction T

bank&retirar(A,4)  
bank&depositar(B,4)

balance=A.read()      \$100  
A.write(balance-4)      \$96

balance=B.read()      \$200  
B.write(balance+4)      \$204

### Transaction U

bank&retirar(C,3)  
bank&depositar(B,3)

balance=C.read()      \$300  
C.write(balance-3)      \$297

balance=B.read      \$204  
B.write(balance+3)      \$207

# *Ejemplo de versiones sombras*

Mapa antes de T y U
A --> P0
B --> P1
C --> P2

Mapa después de T commit
A --> P3
B --> P4
C --> P2

## Archivo de almacenamiento de versiones

P0	P1	P2	P3	P4	P5	P6
100	200	300	96	204	296	207

Checkpoint

## Archivo de estados de transacciones

T	T	U
prepared <A,P3> <B,P4>	committed	prepared <B,P6> <C,P5>

# TAREAS

- ¿Cómo es el proceso de recuperación con la técnica de versiones sombras?
- ¿Qué pasaría si el servidor se cae entre adicionar un estado *committed* y actualizar el mapa?
- Ventajas y desventajas del *logging* y las *versiones sombras*.

## *Acuerdos en Sistemas que Fallan*

- Los algoritmos de acuerdo en sistemas distribuidos son muy comunes: Two Phase Commit, elección de coordinador, sincronización y exclusión mutua, etc.
- Las fallas en los sistemas distribuidos pueden afectar las decisiones distribuidas
- El objetivo principal de los algoritmos de acuerdos distribuidos es poder llegar a un consenso con los procesos sobrevivientes (fallas silenciosas) o fieles (fallas bizantinas).
- Se estudiarán los problemas de consenso distribuido en dos escenarios: sólo los procesos son confiables o sólo la comunicación es confiable

## *Acuerdos en Sistemas que Fallan*

- Si los procesos son confiables y las líneas de comunicación no lo son, es imposible llegar a un acuerdo
  - Ilustración: el problema de los dos ejércitos
- Si las líneas de comunicación son confiables y los procesos pueden fallar de manera bizantina, es posible llegar a un acuerdo, aún ante fallas bizantinas:
  - Ilustración: el problema de los generales bizantinos

# *Acuerdos en Sistemas que Fallan*

- El problema de los generales bizantinos
  - Para soportar fallas bizantinas es necesario  $3m + 1$  generales y sólo  $m$  traidores para llegar a un acuerdo
  - En sistemas asíncronos y retardos de comunicación no acotados, no es posible llegar a acuerdos ni con fallas bizantinas ni silenciosas