Sistemas Distribuídos
Os Diversos Tipos de Broadcast

Prof. Elias P. Duarte Jr.
Universidade Federal do Paraná (UFPR)
Departamento de Informática
www.inf.ufpr.br/elias/sisdis

Sumário

- Inicialmente vamos definir a difusão – em inglês: *broadcast*
- Em seguida vamos estudar os tipos clássicos de broadcast:
  - *Reliable Broadcast*
  - FIFO *Broadcast*
  - *Causal Broadcast*
  - *Atomic Broadcast*
- Vamos definir o broadcast uniforme
- Por fim vamos estudar um tipo (muito simples!) adicional: *Best-Effort Broadcast*

Difusão = *Broadcast*

- Em diversas situações, é necessário que um processo transmita uma mensagem para **todos** os processos de um sistema distribuído
- É exatamente esta a funcionalidade do
  - Broadcast: comunicação 1-para-todos
- Lembrando que temos outras alternativas:
  - Unicast: comunicação 1-para-1
  - Multicast: comunicação 1-para-Grupo
  - Anycast: comunicação 1-para um membro qq do grupo

Primitivas de Broadcast

- Três primitivas são utilizadas:
  - *broadcast(msg)*: executada pelo processo que envia a mensagem para todos os processos do grupo
  - *receive(msg):* executada pelo processo que recebe a mensagem enviada por broadcast
  - *deliver(msg):* executada pelo processo que recebeu a mensagem enviada por broadcast para entregá-la à aplicação, de acordo com critérios definidos (ex. só entrega uma msg 1 única vez)
- Duas primitivas são iguais às que vimos para unicast: apenas *send(msg)* pode ser utilizada por ou trocada por *broadcast(msg)*
- É comum usar a abreviação *bcast(msg)*

Identificação Única de Mensagens

- Vamos relembrar que é simples identificar cada mensagem que trafega no sistema distribuído
- Basta que a origem inclua um contador local de mensagens
- O identificador da origem mais o contador identificam univocamente cada mensagem

Os 4 Tipos Clássicos de Broadcast

- Nesta aula vamos começar estudando os 4 tipos clássicos de broadcast, que são:
  - Broadcast Confiável → Reliable Broadcast
  - Broadcast FIFO
  - Broadcast Causal
  - Broadcast Atômico
- Existem outros (menos clássicos ;-) um dos quais vamos estudar hoje: *Best-Effort Broadcast*

Reliable Broadcast

- *Reliable Broadcast* = Difusão Confiável
- Primitiva de transmissão: *rbcast(msg)*
- *Modelo de falhas de processos: crash*
- Canal de comunicação: perfeito
- Informalmente: se uma mensagem é transmitida por reliable broadcast e é entregue por algum processo que não falha, então ela é entregue por todos os processos corretos
  - várias sutilezas para discutir!

Reliable Broadcast

- Considere que um processo transmitiu uma mensagem por reliable broadcast e falhou! :-0
- Agora considere que a mensagem foi entregue por algum(ns) processos mas não todos

Reliable Broadcast

- Como resolver o problema?
- Veja, não tem saída: os processos que entregaram a mensagem vão ter seguir com o broadcast!
- Como fazer isso? Uma alternativa clássica é o seguinte algoritmo

O Algoritmo Básico de Reliable Broadcast

Algoritmo Difusão Confiável

Delivered: conjunto de mensagens entregues

Init: Delivered ← Φ;

Upon rbcast(msg): Delivered ← Delivered **U** {msg};
deliver(msg);
for all pi **∈** S do: send(msg) to pi;

Upon receive(msg): if msg **not** in Delivered
then delivered ← delivered U {msg};
deliver(msg);
for all pi **∈** S do: send(msg) to pi;

Algoritmo Básico RBCast

- Este algoritmo é **muito** importante
- Funciona mesmo em sistemas assíncronos sujeitos a falhas crash
- Qual o número de mensagens que utiliza?
- Cada processo correto do sistema que recebe a mensagem a transmite para todos os processos do sistema: N^2 mensagens
- Em um sistema com N=100 processos, são 10 mil mensagens...

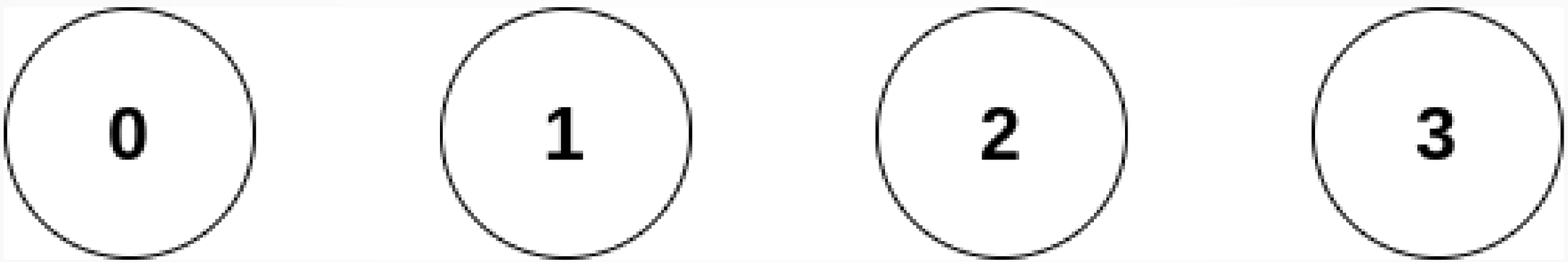
Propriedades do Reliable Broadcast

- Acordo (*Agreement*): se um processo correto entrega a mensagem *msg*, então todo processo correto *eventually* entrega *msg*
- Validade (*Validity*): se um processo correto envia a mensagem msg usando *rbcast(msg)*, então todos os processos corretos *eventually* entregam *msg*
- Integridade *(Integrity)*: um processo correto entrega a mensagem *msg* uma única vez e apenas se ela foi transmitida por um processo correto que a transmitiu usando *rbcast(msg)*

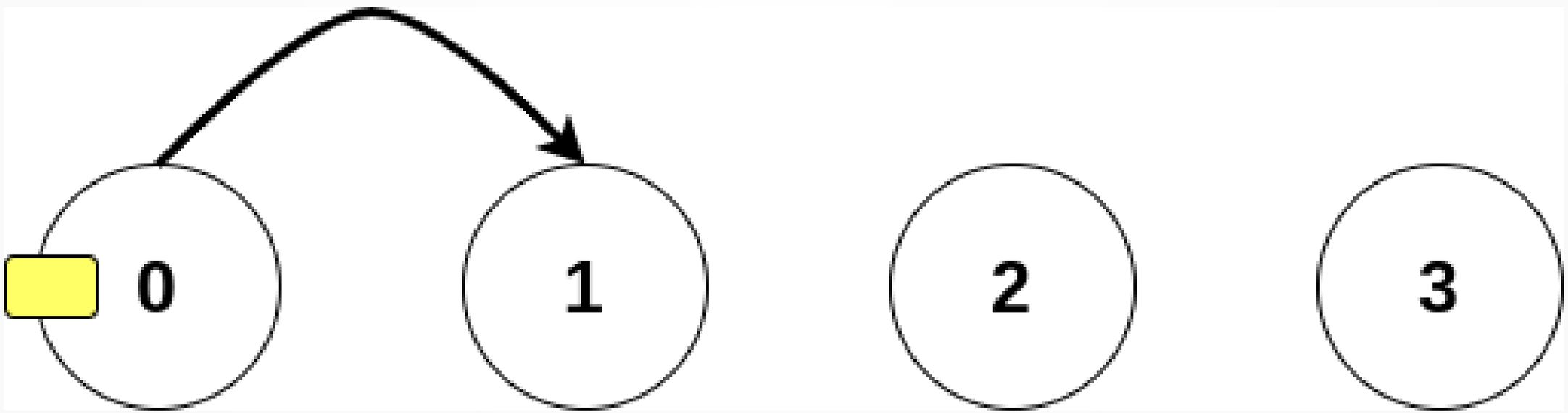
Por que o algoritmo é correto?

- Considere que o processo origem envia a mensagem para 1 único processo e falha
- Sem problemas! Este processo que recebeu vai enviar para todos os outros
- Mas e se envia para apenas 1 processo e falha?
- Sem problemas! Este processo que recebeu vai enviar para todos os outros
- E assim por diante

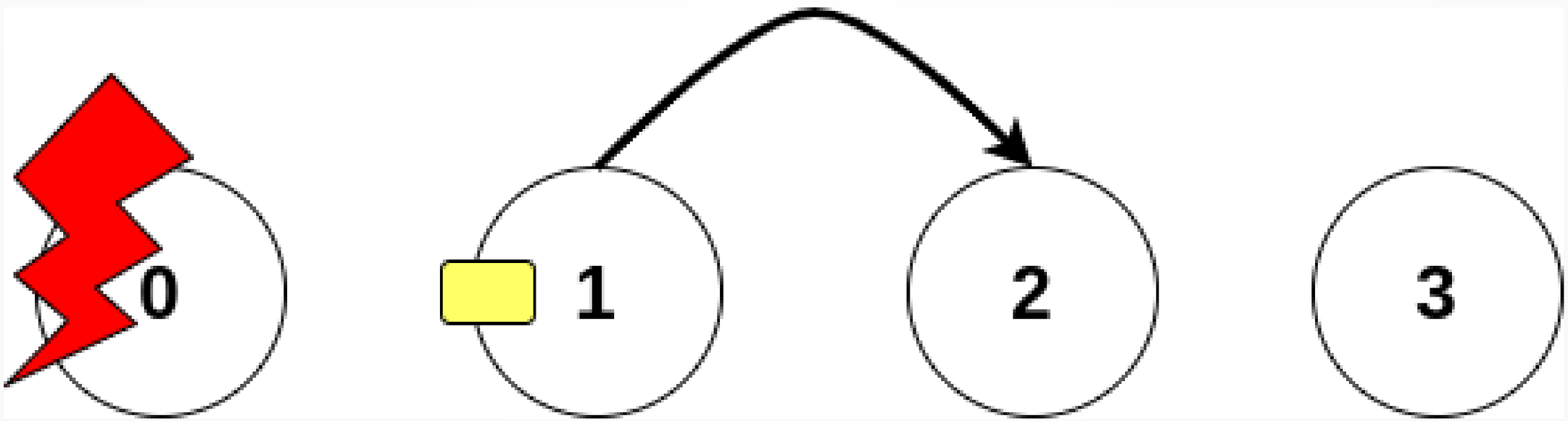
Reliable Broadcast no Extremo



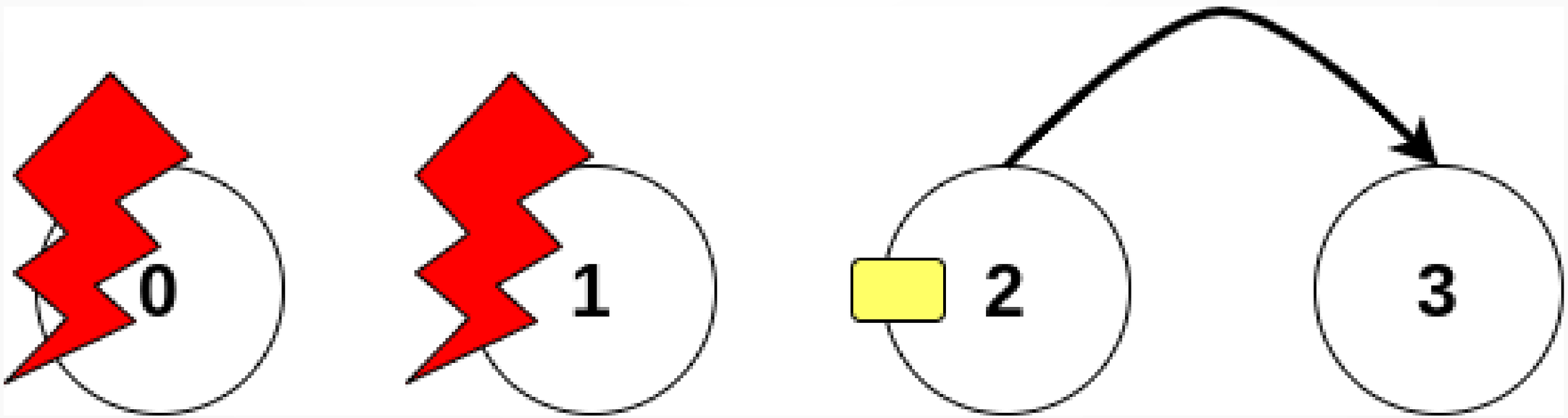
Reliable Broadcast no Extremo



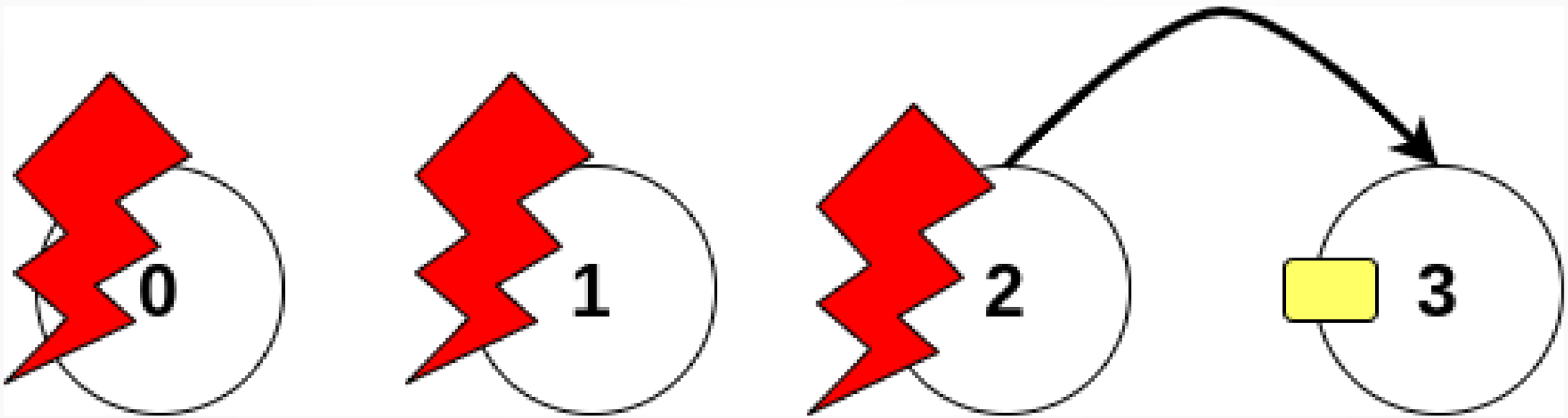
Reliable Broadcast no Extremo



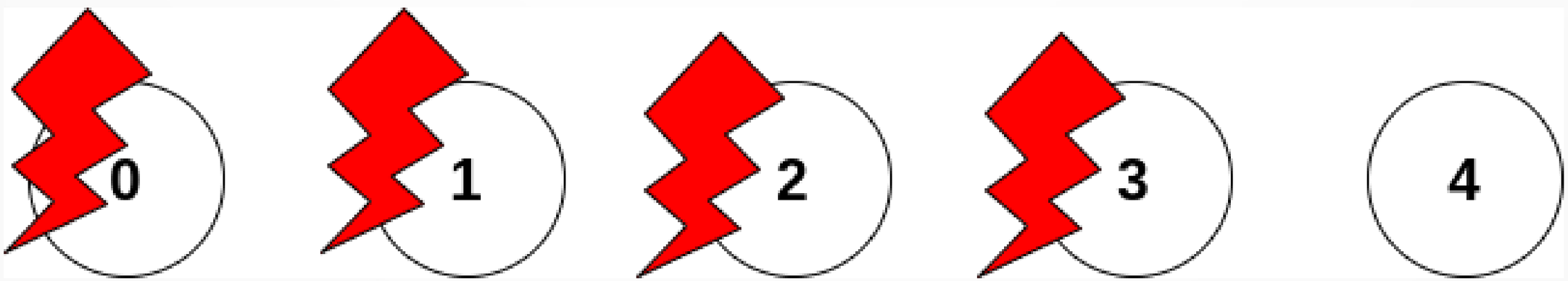
Reliable Broadcast no Extremo



Reliable Broadcast no Extremo



Reliable Broadcast no Extremo



Entrega Ordenada de Mensagens

- A partir de agora os 3 broadcasts clássicos que vamos estudar todos eles aplicam uma ordem de entrega de mensagens
- Têm todas as propriedades do Reliable Broadcast, mais a ordem!
- Observe que as mensagens podem ser recebidas em ordens diferentes pelos processos...
- ...mas todos vão entregar para a aplicação de acordo com a ordem específica que o tipo de broadcast demanda: FIFO, Causal ou Atômica
- OBS.: O Best-Effort Broadcast não tem ordem ;-)

FIFO Broadcast

- FIFO: First In First Out
- O broadcast FIFO trata da ordem de mensagens transmitidas por *um mesmo processo origem*
- Informalmente: não entrega mensagens vindas de um processo “embaralhadas” e sim na ordem em que foram transmitidas

Ordem FIFO

- Uma forma ingênua de definir a ordem FIFO é:

  → Se um processo i transmite a mensagem msg antes de transmitir a mensagem msg’, então todos os processos entregam msg antes de entregar msg’

Ordem FIFO

- Uma forma ingênua de definir a ordem FIFO é:

  → Se um processo i transmite a mensagem msg antes de transmitir a mensagem msg', então todos os processos entregam msg antes de entregar msg'
- Nesta definição uma mensagem pode não ser entregue!
- Por exemplo se um processo transmite por broadcast FIFO as mensagens m1, m2 e m3 e todos menos um processo entregam nesta ordem...
- ...o outro entrega apenas m1 e m3 (por algum motivo m2 desapareceu para este processo): a definição tá valendo!

Ordem FIFO

- A definição correta da ordem FIFO é:

  → Se um processo i transmite a mensagem msg por FIFO broadcast antes de transmitir msg’, então todos os processos corretos só entregam msg’ se antes tiverem entregado msg

Causal Broadcast

- O exemplo clássico da ordem causal é a transmissão de uma piada em um grupo de rede social, seguida das manifestações geradas (i.e., “kkkkkkk”)
- Há uma “relação causal” entre a piada e a gargalhada
- No broadcast causal a ordem causal é respeitada, assim, no exemplo acima, a gargalhada nunca é entregue antes que a piada

Causal Broadcast

- A ordem causal foi definida quando estudamos relógios lógicos
- Ela é exatamente a ordem que é definida pela relação “aconteceu-antes-de”
- No contexto do broadcast causal ela é aplicada da seguinte forma:

1) Um processo qualquer ordena usando seu relógio local deliver(msg) → cbcast(msg’)
2) Em qualquer processo: cbcast(msg) → deliver(msg)
3) Transitividade: se x→w e w→z então x->z

Ordem Causal

- Podemos definir a ordem causal da seguinte maneira:

  Se o cbcast de uma mensagem msg *aconteceu-antes-do* cbcast de uma mensagem msg’ e há uma precedência causal entre as transmissões de duas mensagens msg → msg’, então nenhum processo entrega msg’ sem antes ter entregue msg

Ordem Causal: Ingênua

- Se pensarmos nas transmissões de 3 mensagens m1 → m2 → m3 com relação de causalidade entre elas, nesta ordem
- A definição ingênua da ordem causal é:
  - Se há uma precedência causal entre as transmissões de duas mensagens msg e msg', então todo processo correto entrega msg antes de entregar msg'
- No exemplo acima permitiria a entrega da gargalhada sem entregar a piada

Atomic Broadcast

- Este é o mais importante dos broadcasts ordenados: usado na replicação distribuída
- Por exemplo, considere que um banco de dados está replicado em diversos locais
- Considere que diversos usuários enviam operações para serem realizadas sobre o banco de dados
- É essencial que todas as réplicas executem as operações na mesma ordem, caso contrário o banco de dados pode ficar inconsistente!

Ordem Atômica

- No broadcast atômico todas as mensagens são entregues por todos os processos na mesma ordem

Ordem Atômica

- No broadcast atômico todas as mensagens são entregues por todos os processos na mesma ordem
- Qual ordem?

Ordem Atômica ou Total

- No broadcast atômico todas as mensagens são entregues por todos os processos na mesma ordem
- Qual ordem?
- Resposta: *qualquer ordem!* Desde que seja a mesma em todos os processos

Ordem Total

- Podemos definir a ordem total (*total order*) da seguinte maneira:

  → Se dois processos x e z corretos entregam as mensagens msg e msg', então x entrega msg antes de msg' se e somente se z entrega msg antes de msg'

- Todos os processos entregam para a aplicação a mesma sequência de mensagens

É Possível Misturar

- A ordem total é *qualquer* ordem
- É possível definir tipos híbridos de broadcast: FIFO-atomic, Causal-atomic etc.
- Neste caso deve ser explícita a especificação das ordens desejadas

Broadcast Uniforme

- Os processos podem falhar
- Considere que um processo entregou uma mensagem [para a aplicação] e falhou
- Pode ser que esta aplicação seja uma réplica de um banco de dados e garantir o que o processo que falhou fez pode ser essencial
- Neste caso usamos o broadcast uniforme

Difusão Confiável Uniforme

- *Uniform Reliable Broadcast*
- Há variações uniformes para todos os broadcasts!
- *Se um processo [correto ou falho] entregou uma mensagem, então todos os processos eventually entregam aquela mensagem*
- Mais a frente na nossa disciplina vamos estudar a implementação de algoritmos distribuídos uniformes
- A ideia genérica é que um processo só entrega uma mensagem se sabe que outros processos sabem que está fazendo isso

Best-Effort Broadcast

- Primitiva: BEBcast(msg)
- Propriedades:
  - Validade: se os processos x e z são corretos e x faz o BEBcast de uma mensagem msg, então z eventualmente entrega msg
  - Observe que não fala nada sobre x falhar ;-)
  - Outras propriedades: não entrega mensagens duplicadas, não entrega mensagens espúrias

Best-Effort Broadcast

- Se o processo que executa o BEBcast(msg) não falhar, então todos os processos corretos do sistema entregam msg
- Entretanto: se o processo origem falha tendo transmitido a mensagem para alguns processos e não para outros, é isso mesmo
- Se a origem falha, pode ser que alguns processos não recebam e portanto não entreguem a mensagem :-P

Referências:

A melhor referência para os diversos tipos de broadcast ainda é o Capítulo 5 do Sape Mullender:

V. Hadzilacos, S. Toueg, “Fault-Tolerant Broadcasts and Related Problems,” **in** Sape Mullender, *Distributed Systems*, Addison-Wesley, 1993.

O BEBcast é definido nosso livro texto.

Conclusão

- Nesta aula primeiro definimos o que é broadcast
- Depois definimos 4 tipos de difusão:
  - Confiável (*Reliable Broadcast*)
  - FIFO (FIFO *Broadcast*)
  - Causal (*Causal Broadcast*)
  - Atômico (*Atomic Broadcast*)
- Além das definições, hoje vimos que um algoritmo de Reliable Broadcast funciona corretamente mesmo em sistemas assíncronos sujeitos a falhas crash
- Definimos o broadcast uniforme
- Definimos o *Best-Effort Broadcast*

Obrigado!
Página da Disciplina
Sistemas Distribuídos:
www.inf.ufpr.br/elias/sisdis