GBC074 - Sistemas Distribuídos

Difusão tolerante a falhas

Paulo Coelho
Universidade Federal de Uberlândia

Especificação

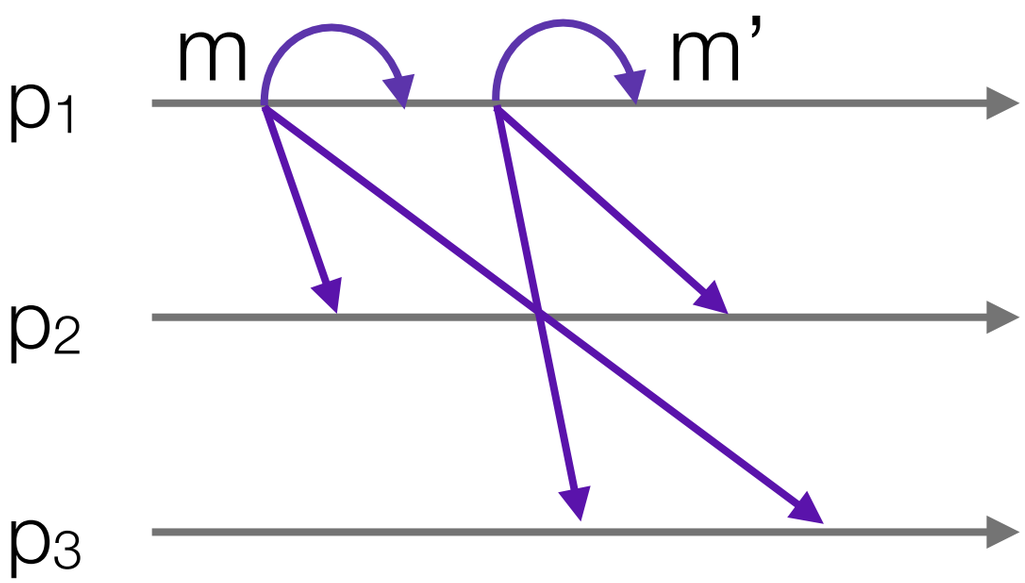
- Modelo
  - ✦ Considere um conjunto $\Sigma = \{ p_1, \ldots, p_n \}$ com n processos
  - ✦ Assuma o modelo *crash failure*
    - Um processo *correto* nunca falha
    - Um processo *faltoso* eventualmente falha

Difusão confiável

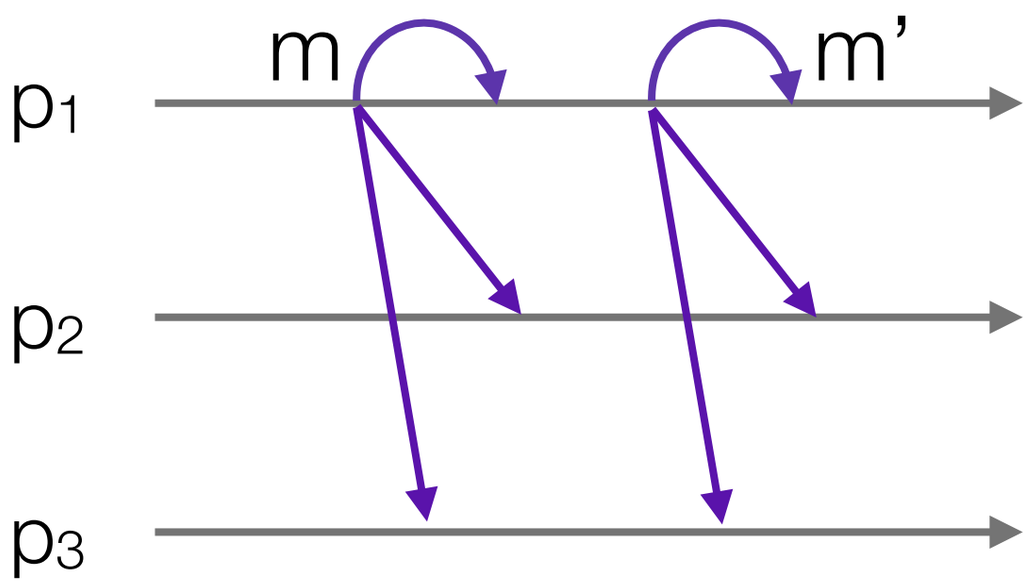
- Difusão confiável (*Reliable broadcast*)
  - ✦ Validade: se um processo correto faz a difusão de uma mensagem **m** (*broadcast(m)*), então todos os processos corretos eventualmente entregam **m** (*deliver(m)*)
  - ✦ Acordo: se um processo correto entrega **m**, então eventualmente todos os processos corretos entregam **m**
  - ✦ Integridade: para qualquer mensagem **m**, todo processo correto entrega **m** no máximo uma vez, e somente se **m** já foi previamente difundida

Difusão FIFO

- Difusão FIFO (*FIFO broadcast*)
  - ✦ Ordem FIFO: se um processo correto difunde **m** antes de **m'**, então nenhum processo correto entrega **m'** a menos que tenha feito a entrega de **m**



Non-FIFO Reliable Broadcast



FIFO Reliable Broadcast

Difusão causal

- Difusão causal (*Causal broadcast*)
  - ✦ Exemplo:
    - Em uma aplicação de envio de notícias, se usuários difundem seus artigos com difusão FIFO, o seguinte pode ocorrer:

A. Usuário **A** difunde um artigo

B. Usuário **B** entrega o artigo e difunde uma resposta ao artigo

C. Usuário **C** entrega a resposta de **B** antes do artigo de **A**

Difusão causal

- *Causal broadcast (happens-before, Lamport '78)*
  - ✦ Uma execução da primitiva *broadcast(m)* ou *deliver(m)* é denominado um **evento**

    Dizemos que evento *e* precede (causalmente) *f*, $e \rightarrow f$, iff

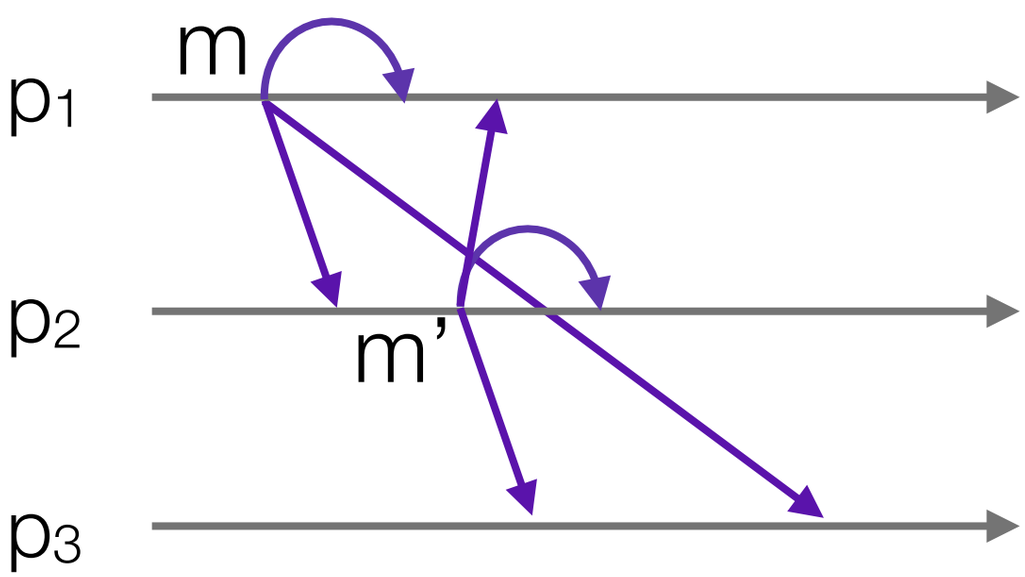
    (a) o mesmo processo executa *e* e *f* nesta ordem, ou

    (b) *e* é a difusão de **m** e *f* é sua entrega, ou ainda

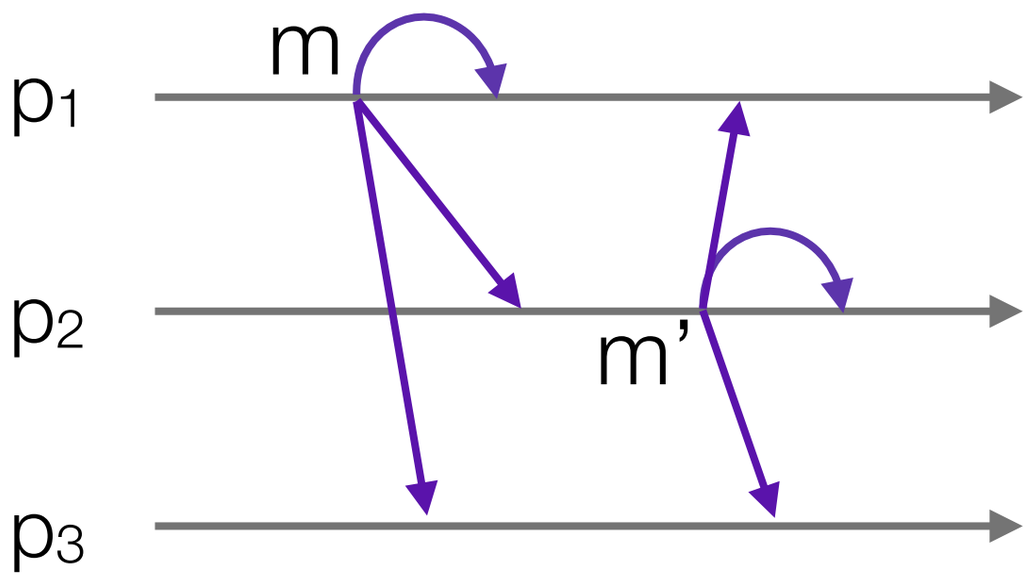
    (c) existe um evento *h* tal que $e \rightarrow h$ e $h \rightarrow f$

Difusão causal

- Difusão causal
  - ✦ Ordem causal: se a difusão de uma mensagem **m** precede a difusão de uma mensagem **m'**, então nenhum processo correto entrega **m'** a menos que tenha feito a entrega de **m**



Non-Causal Reliable Broadcast



Causal Reliable Broadcast

Difusão confiável *uniforme*

- Difusão confiável uniforme (*Uniform reliable broadcast*)
  - ✦ Validade: se um processo correto faz a difusão de uma mensagem **m** (*broadcast(m)*), então todos os processos corretos eventualmente entregam **m** (*deliver(m)*)
  - ✦ Acordo ***uniforme***: se um processo ~~*correto*~~ entrega **m**, então eventualmente todos os processos corretos entregam **m**
  - ✦ Integridade: para qualquer mensagem **m**, todo processo correto entrega **m** no máximo uma vez, e somente se **m** já foi previamente difundida

Difusão *uniforme*

- Difusão FIFO uniforme (*Uniform FIFO broadcast*)
  - ✦ Ordem FIFO ***uniforme***: se um processo ~~*correto*~~ difunde **m** antes de **m'**, então nenhum processo ~~*correto*~~ entrega **m'** a menos que tenha feito a entrega de **m**
- Difusão causal uniforme (*Uniform causal broadcast*)
  - ✦ Ordem causal ***uniforme***: se a difusão de uma mensagem **m** precede a difusão de uma mensagem **m'**, então nenhum processo ~~*correto*~~ entrega **m'** a menos que tenha feito a entrega de **m**

Difusão confiável - implementação

broadcast(R, m) works as follows:
    send(m) to all

upon receive(m) for the first time do
    deliver(R, m)
    send(m) to all

Difusão confiável *uniforme* - implementação

```
broadcast(UR, m) works as follows:
    send(m) to all

upon receive(m)
    if [for f+1 processes q: received(m) from q] then
        deliver(UR, m)
    if [didn't send m yet] then send(m) to all
```

Difusão confiável uniforme vs. não-uniforme

	Número de mensagens	Latência
Reliable Broadcast	N^2	δ
Uniform Reliable Broadcast	N^2	2δ

Difusão confiável FIFO - implementação

- *FIFO reliable broadcast*
  - ✦ Como transformar *Reliable Broadcast* em *FIFO Reliable Broadcast*?
  - ✦ Considerações
    - Se **m** é a i-ésima mensagem difundida por **p**, então **m** é rotulada com *sender(m) = p* e *seq#(m) = i*
    - Cada processo mantém um vetor local de contadores *next[1..n]*

Difusão confiável FIFO - implementação

- Algorithm for a process *p*:

Initialization:
msgSet ← ∅
next[q] ← 1, for each process q
seq ← 1

To execute broadcast(F, m):
sender(m) ← p
seq#(m) ← seq
seq← seq + 1
broadcast(R, m)

deliver(F,–) occurs as follows:
upon deliver(R, m) do
q ← sender(m)
msgSet ← msgSet ∪ { m }
while (∃m’ ∈ msgSet: sender(m’) = q and next[q] = seq#(m’))
deliver(F, m’)
next[q] ← next[q] + 1

Difusão confiável causal - implementação

- *Causal reliable broadcast*
  - ✦ Como transformar *Reliable Broadcast* em *Causal Reliable Broadcast*?
  - ✦ Notação
    - Sequências de mensagens: $\langle m_1, m_2, \ldots \rangle$
    - $\perp$ é a sequência vazia
    - $\oplus$ é o operador de concatenação

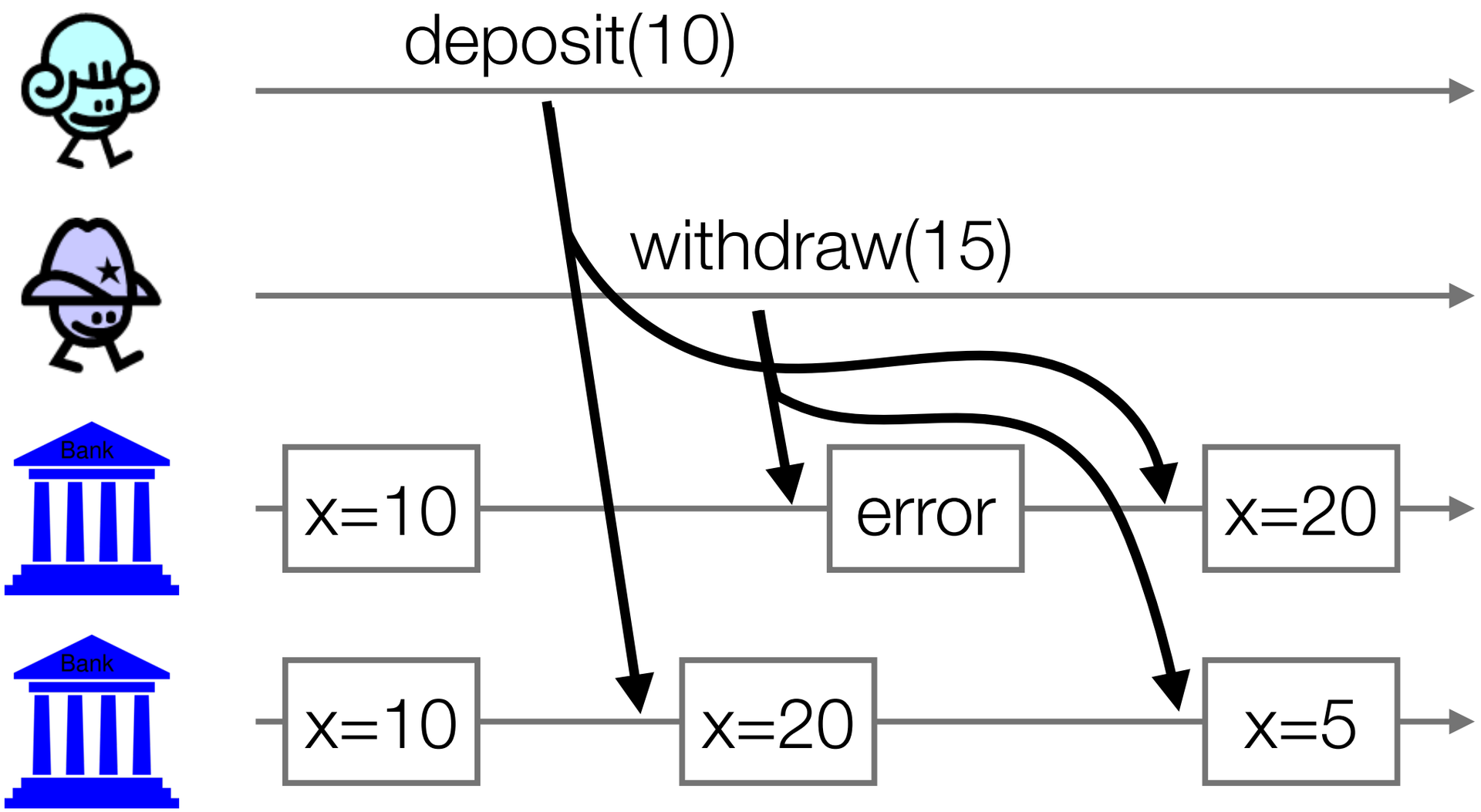
Difusão confiável causal - implementação

Initialization:
    rcntDlvrs ← ⊥

To execute broadcast(C, m):
    broadcast(F, rcntDlvrs ⊕ ⟨ m ⟩)
    rcntDlvrs ← ⊥

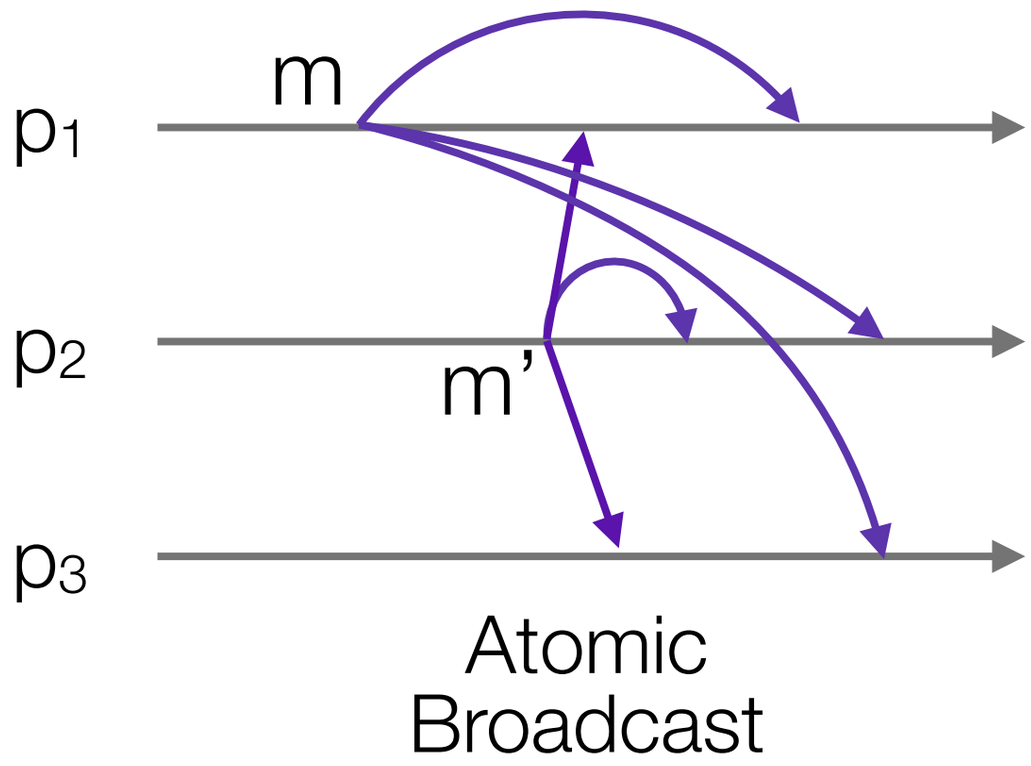
deliver(C,–) occurs as follows:
    upon deliver(F, ⟨ m_1, m_2, …, m_l ⟩) do
        for i from 1 to l do
            if p has not previously executed deliver(C, m_i) then
                deliver(C, m_i)
                rcntDlvrs ← rcntDlvrs ⊕ ⟨ m_i ⟩

Difusão - especificações



Difusão atômica

- Difusão atômica (*Atomic broadcast*)
  - ✦ Ordem uniforme total: se processos **p** e **q** ambos entregam as mensagens **m** e **m'**, então **p** entrega **m** antes de **m'** se e somente se **q** entrega **m** antes de **m'**



Difusão atômica

- *FIFO Atomic broadcast*
- *Causal Atomic broadcast*
  - ✦ *FIFO atomic broadcast* não garante ordem causal

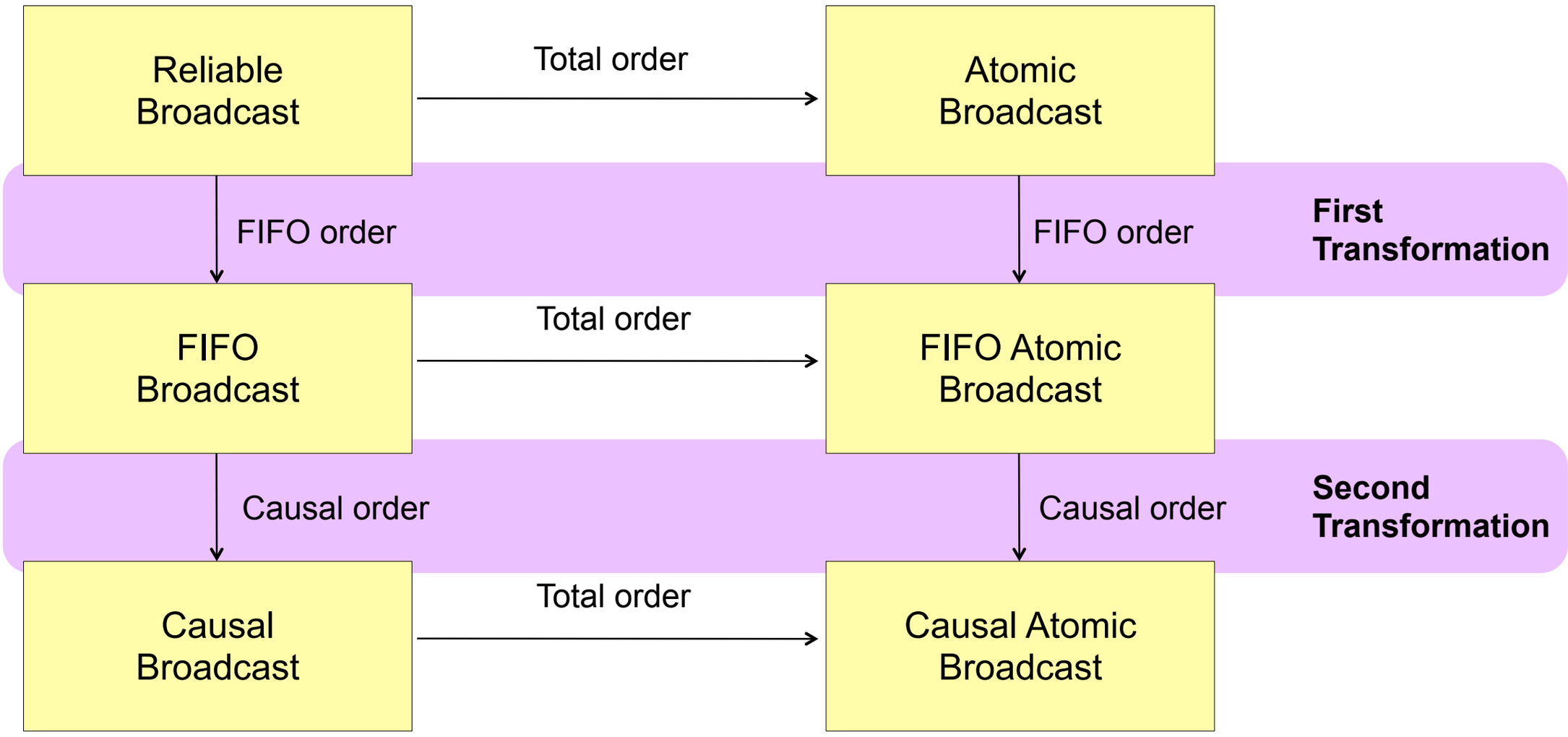
- Reconsidere a aplicação de envio de notícias, se usuários difundem seus artigos com difusão FIFO, o seguinte pode ocorrer:

  A. Usuário faltoso **A** difunde um artigo

  B. Usuário faltoso **B** é o único a entregar o artigo e difunde uma resposta ao artigo

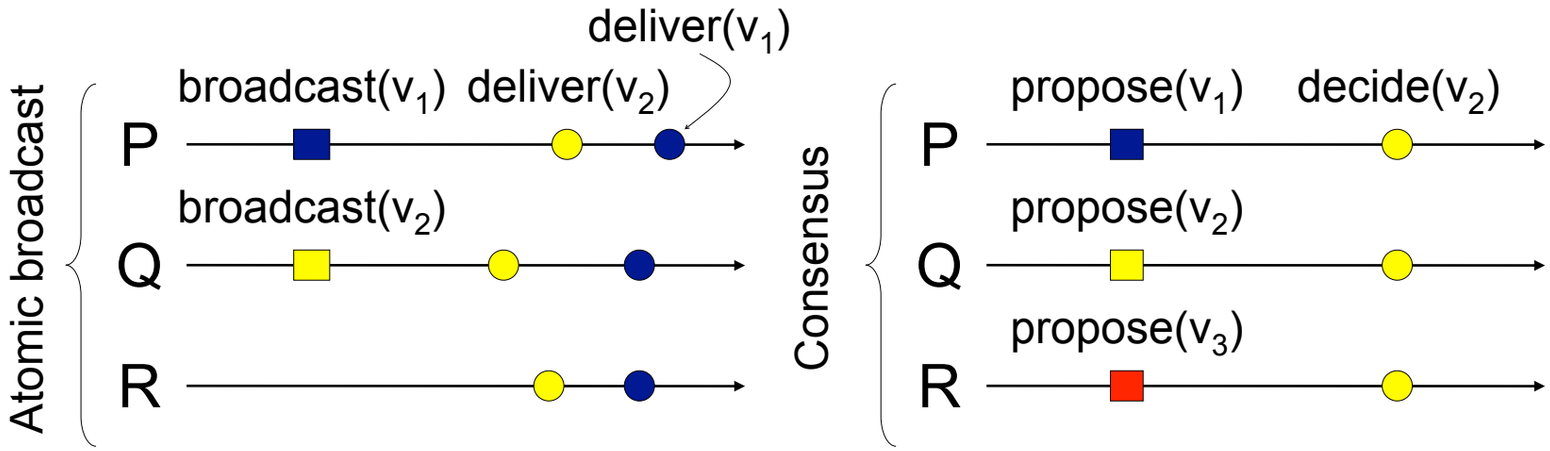
  C. Usuário **C** entrega a resposta de **B** sem nunca entrega o artigo de **A**

Relação entre tipos de difusão



Difusão atômica - implementação

- From atomic broadcast to consensus



To execute propose(v):
broadcast(v)

Let v be the first value delivered
decide(v)

Difusão atômica - implementação

Initialization
    R-delivered ← Ø
    A-delivered ← Ø
    k ← 0
To execute broadcast(A, m):
    broadcast(R, m)
deliver(A,–) occurs as follows:
    upon deliver(R, m) do
        R-delivered ← R-delivered ∪ { m }
    upon R-delivered \ A-delivered ≠ Ø
        k ← k + 1
        A-undelivered ← R-delivered \ A-delivered
        propose(k, A-undelivered)
        wait until decide(k, $msgSet_k$)
        $A\text{-deliver}_k$ ← $msgSet_k$ \ A-delivered
        deliver all messages in $A\text{-deliver}_k$ in some deterministic order
        A-delivered ← A-delivered ∪ $A\text{-deliver}_k$

Difusão atômica - implementação