Computação Distribuída

Odorico Machado Mendizabal

Universidade Federal de Santa Catarina – UFSC
Departamento de Informática e Estatística – INE

Algoritmos de Difusão (*Multicast*)

Primitivas de comunicação *multicast*

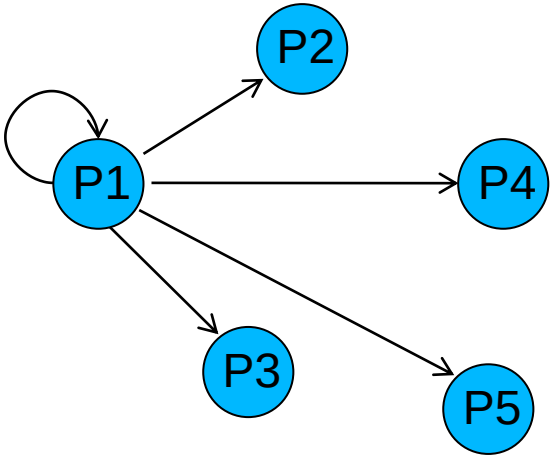
- Primitiva de comunicação que permite comunicação confiável entre um grupo de processos:
  - ***Multicast***: Uma mensagem é enviada para um grupo de processos de uma aplicação

  - ***Broadcast***: Uma mensagem é enviada para todos os processos de uma aplicação

- O grupo de processos participantes da aplicação pode ser estático ou dinâmico
  - No momento vamos nos concentrar em grupos estáticos

Primitivas de comunicação *multicast*

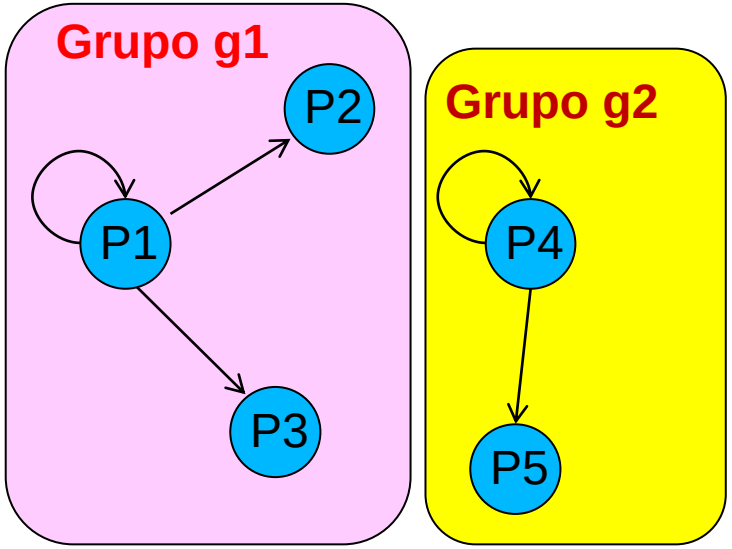
Broadcast



Primitivas
Envio:
broadcast(m)

Recebimento:
deliver(m)

Multicast



Primitivas
Envio:
multicast(g1,m)
multicast(g2,m)

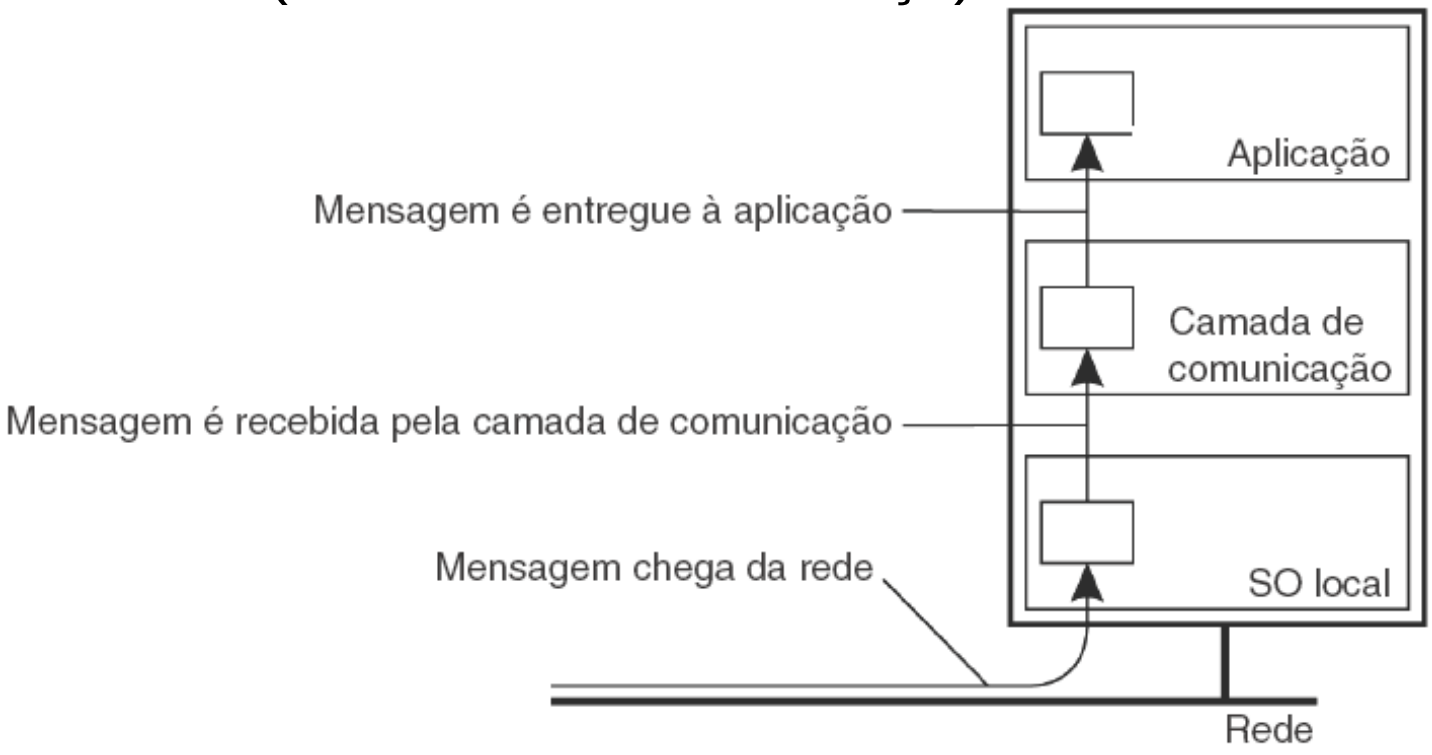
Recebimento:
deliver(g1,m)
deliver(g2,m)

ou deliver(m) (recebe de qualquer grupo ao qual faz parte)

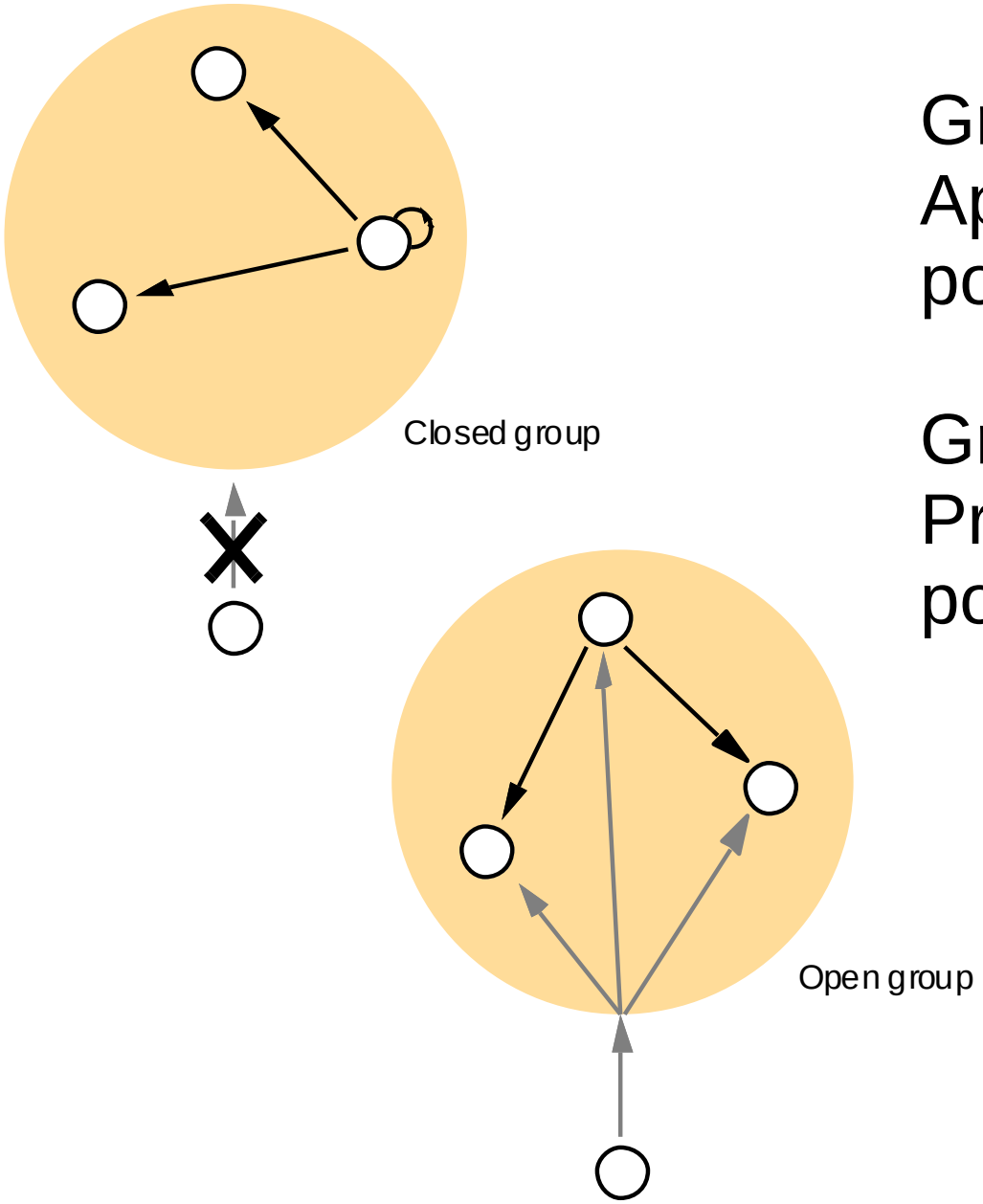
Primitivas de comunicação *multicast*

- Em primitivas de entrega (*deliver(m)*) há diferenciação entre o recebimento e a entrega de uma mensagem para o destinatário

- Uma camada de comunicação é responsável por receber a mensagem e entregá-la apenas quando as garantias necessárias forem satisfeitas (ex. ordem ou confiança)



Grupo *multicast* abertos e fechados



Grupo fechado:
Apenas os processos do grupo podem enviar mensagens

Grupo aberto:
Processos de fora do grupo podem enviar mensagens

- Pode ser implementado permitindo que um membro do grupo recebe mensagens externas e repasse as mensagens no grupo através de *multicast*

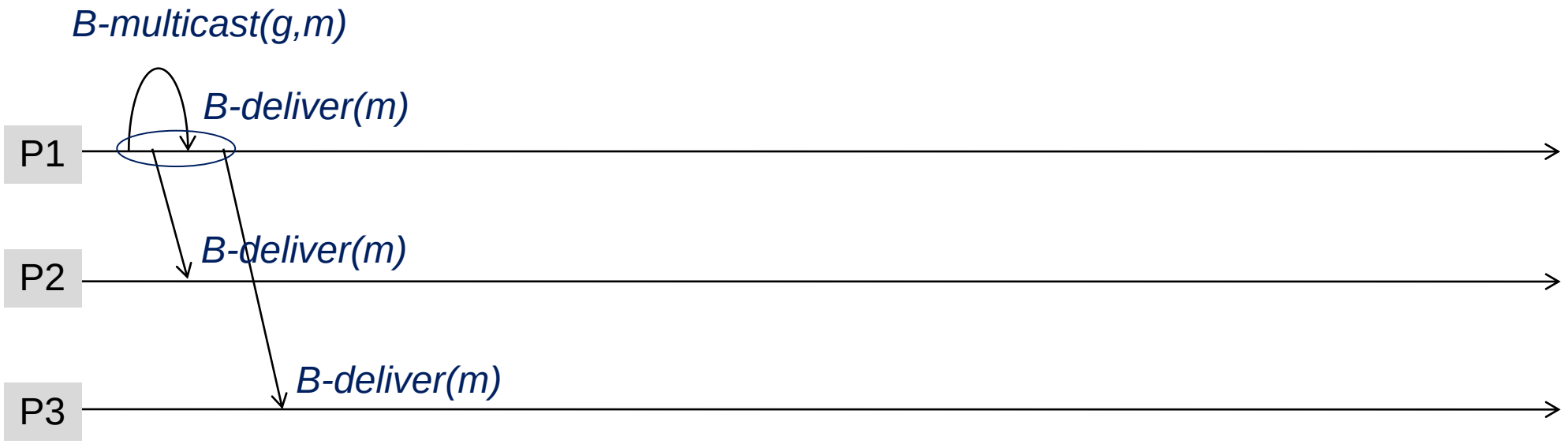
Difusão (*multicast)*

- *Multicast* Básico (*B-multicasting*)
  - Se um processo correto envia m, então todos os processos corretos entregam a mensagem m

  - Algoritmo utilizando as primitivas *send* e *receive* confiáveis:

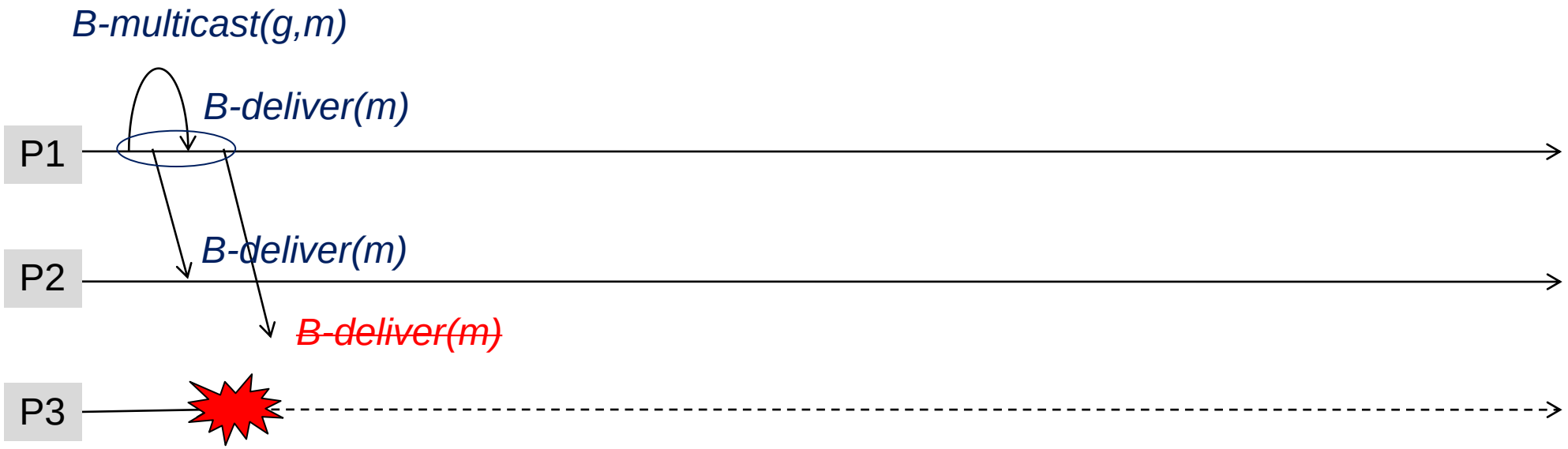
```
B-multicast(g,m):
para cada  processo p ∈ g, send(p,m)

B-deliver (m):
receive(m)
```

Difusão (*multicast*) – *Exemplo 1*



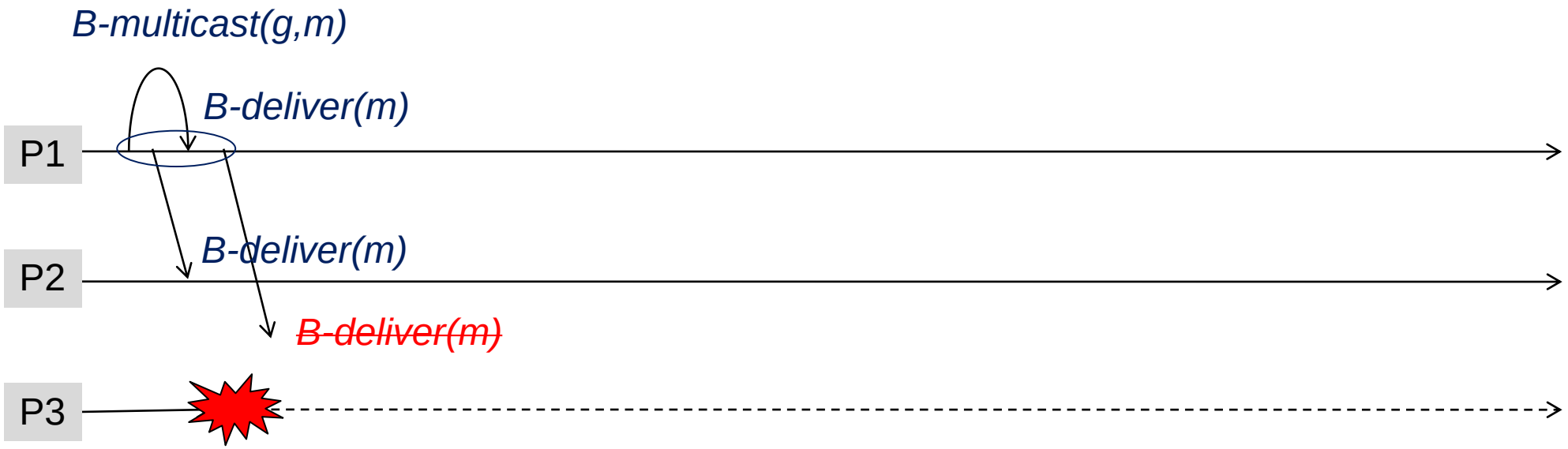
O traço de execução acima viola as propriedades de *multicast*?

Difusão (*multicast*) – *Exemplo 1*



O traço de execução acima viola as propriedades de *multicast*?

Difusão (*multicast*) – *Exemplo 1*

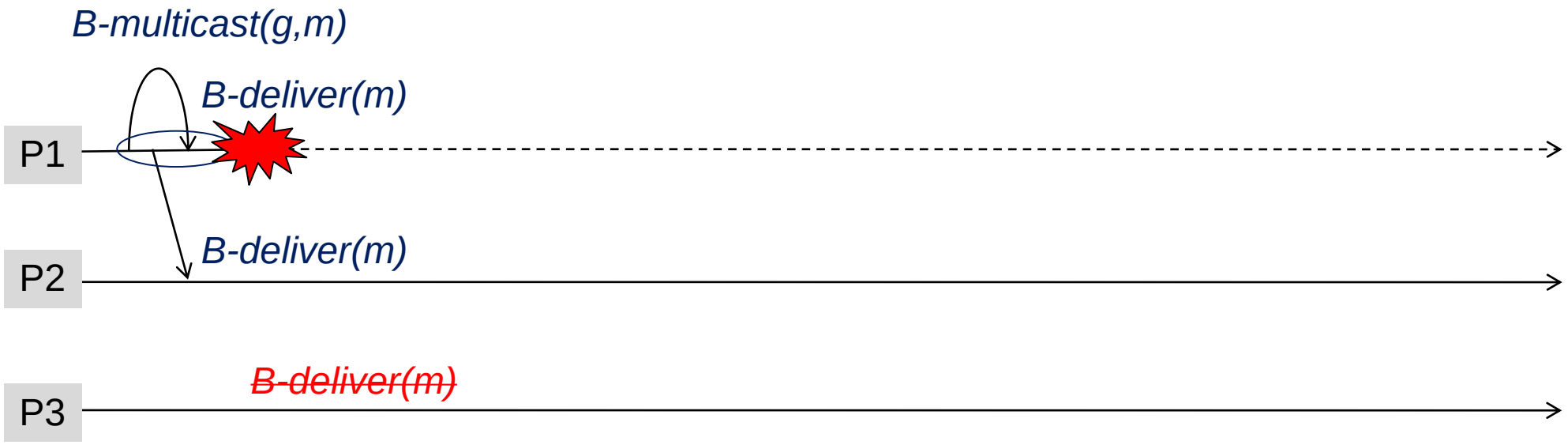


O traço de execução acima viola as propriedades de *multicast*?

Não, o processo que não entrega m (ou seja, P3) não é um processo correto.

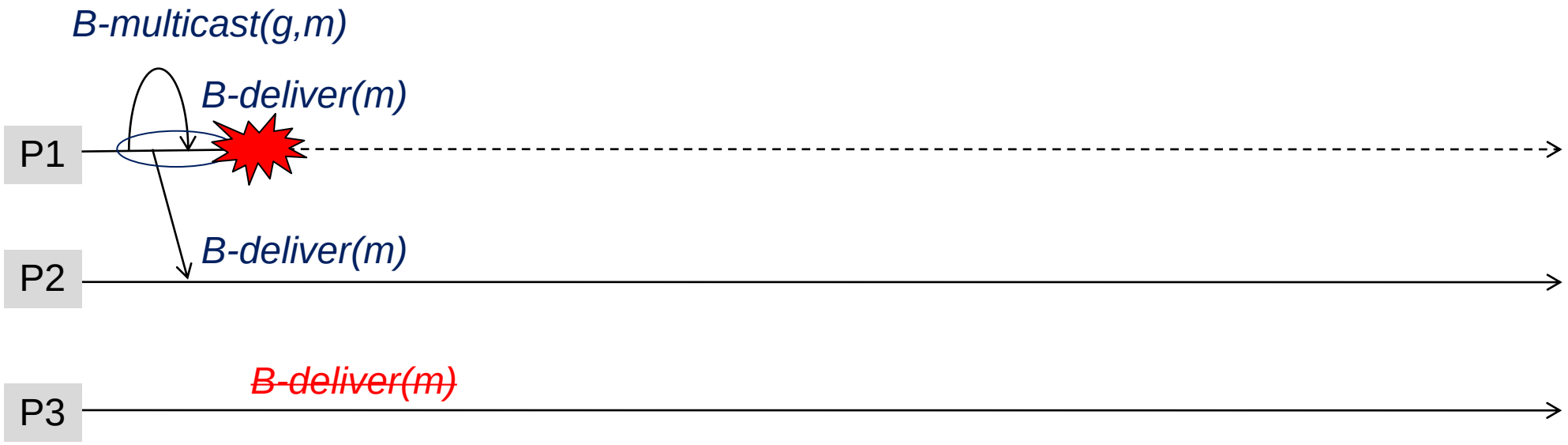
Todos os processos corretos pertencentes à g entregam m

Difusão (*multicast*) – *Exemplo 2*



O traço de execução acima viola as propriedades de *multicast*?

Difusão (*multicast*) – *Exemplo 2*



O traço de execução acima viola as propriedades de *multicast*?

Não, a implementação é baseada em comunicação confiável um-a-um.

P1 não é um processo correto, então não há qualquer garantia de que mensagens enviadas por ele sejam entregues à todos os processos

Difusão (*multicast)*

- Considerações
  - Para reduzir o tempo de envio, múltiplas threads podem ser usadas para paralelizar o envio para diferentes processos

  - Suscetível à *explosão de confirmações*
    - Mensagens de confirmação podem chegar quase ao mesmo tempo, ocupando o buffer de recebimento dos processos
    - Ocasiona perde de confirmações, forçando retransmissões

    - *Uma alternativa é o uso de IP multicast*

Difusão Confiável (*Reliable Multicast)*

- Propriedades (*R-multicasting*)

  - **Integridade**: um processo correto $p \in g$ entrega uma mensagem *m* no máximo uma vez (além disso, *m* foi enviada: *R-multicast(g,m)*)

  - **Validade**: se um processo correto *p* executa *R-multicast(g,m),* então *p* entregará *m*

  - **Acordo**: Se um processo correto $p \in g$ entrega *m*, então cada processo $q \in g$ entrega *m*

Difusão Confiável (*Reliable Multicast)*

On initialization

Received := {};

For process p to R-multicast message m to group g

B-multicast(*g*, *m*); // $p \in g$ is included as a destination

On B-deliver(*m*) *at process q with g = group*(*m*)

if ($m \notin$ *Received*)

then

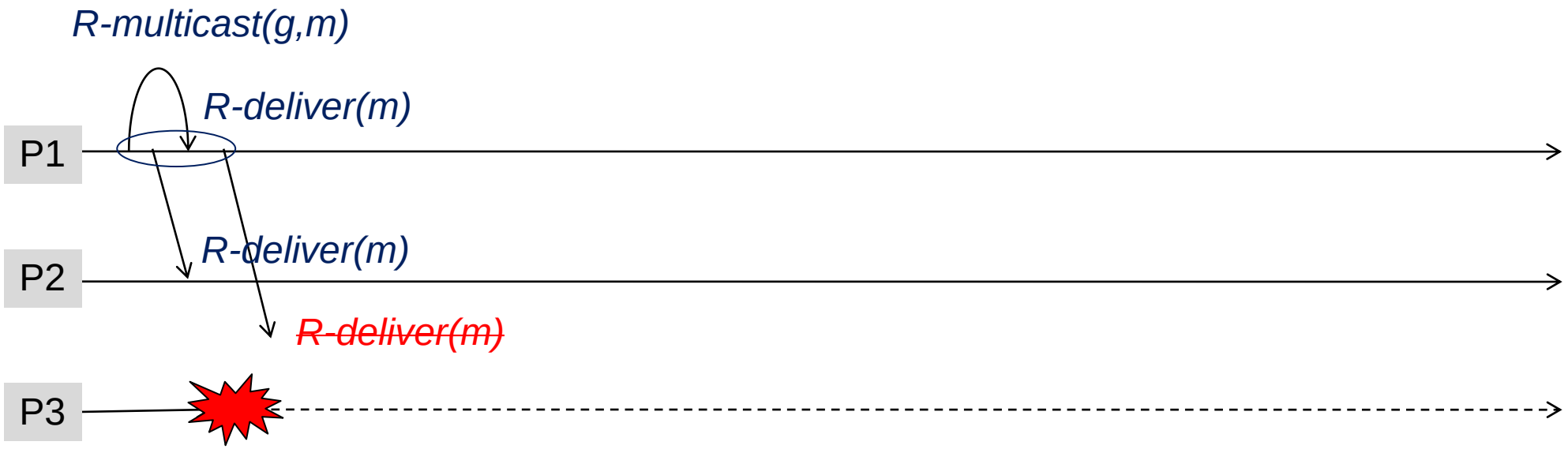
Received := *Received* $\cup$ {*m*};

if ($q \neq p$) *then B-multicast*(*g*, *m*); *end if*

R-deliver m;

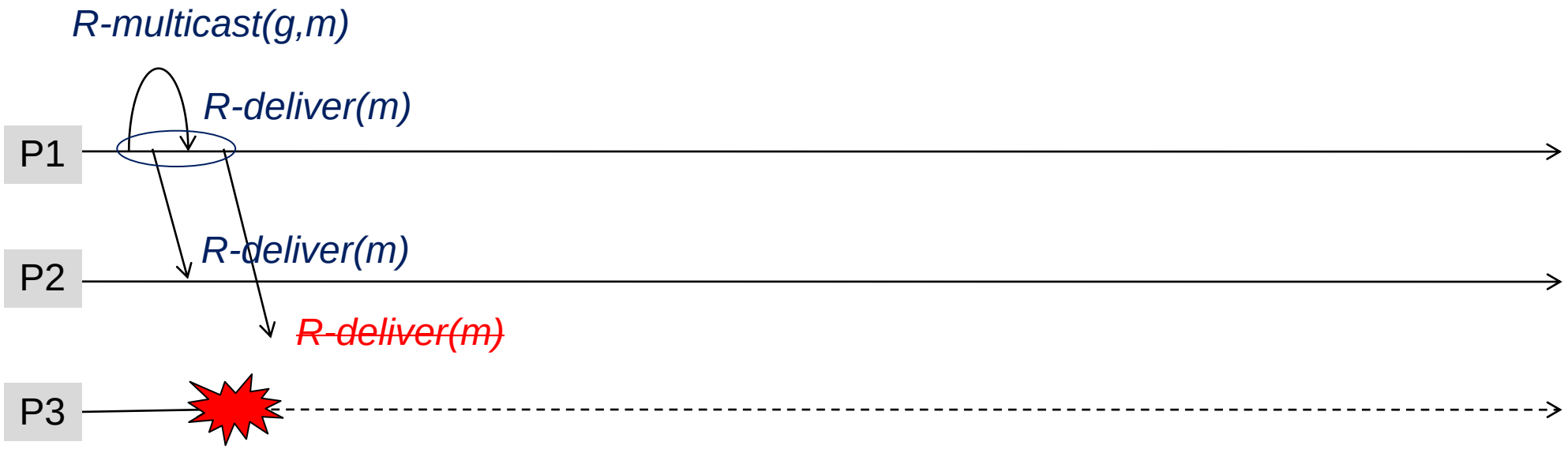
end if

Difusão Confiável (*Reliable Multicast*) – *Exemplo 1*



O traço de execução acima viola as propriedades de *multicast*?

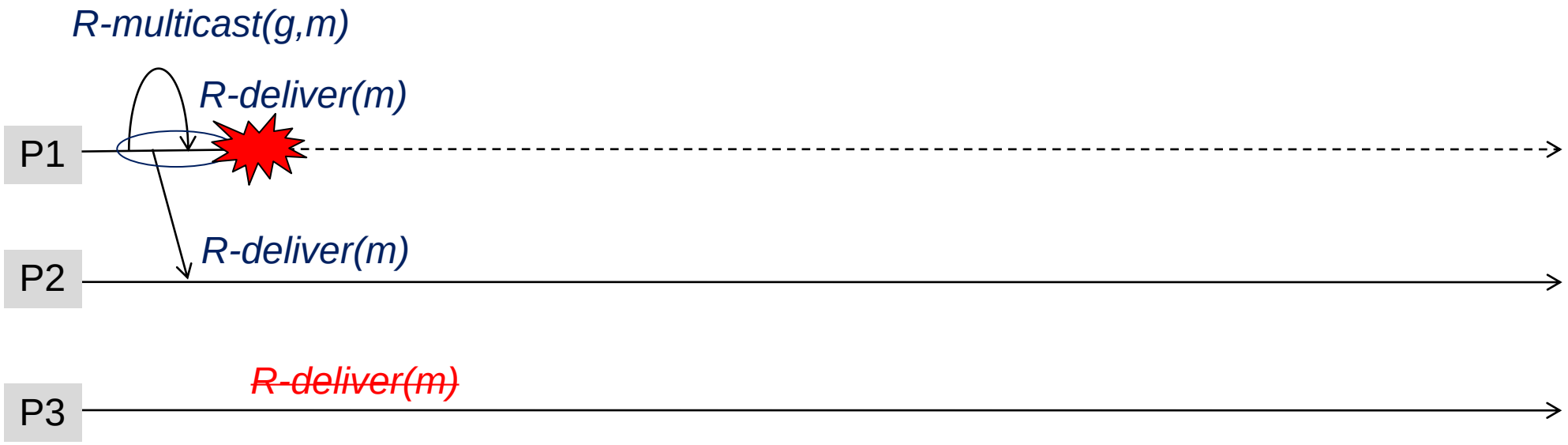
Difusão Confiável (*Reliable Multicast*) – *Exemplo 1*



O traço de execução acima viola as propriedades de *multicast*?

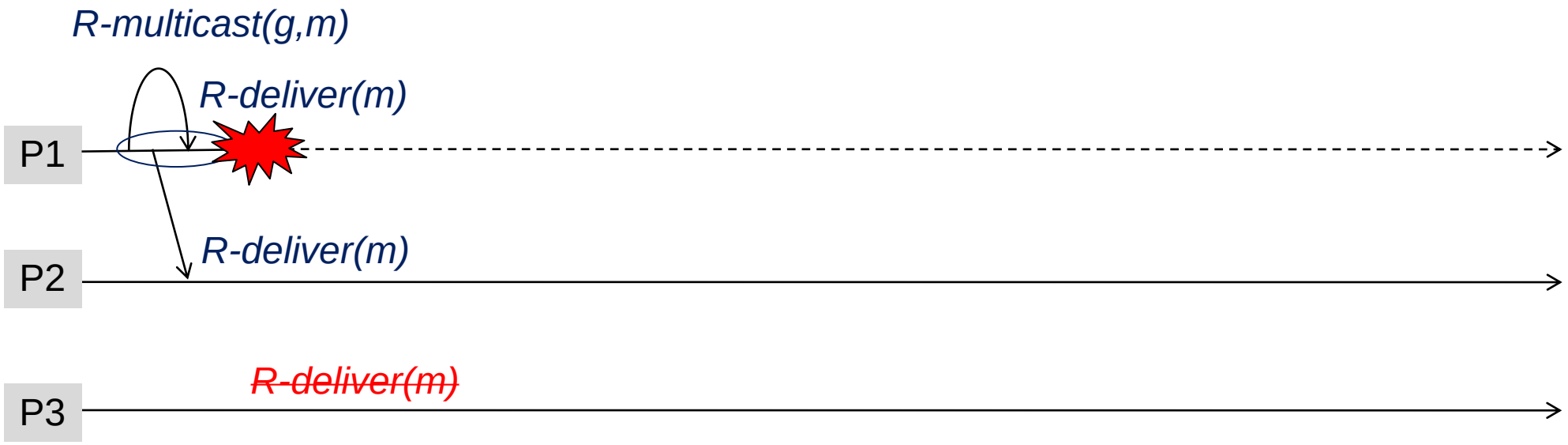
Não, o processo que não entrega m não é um processo correto.
Todos os processos corretos pertencentes à g entregam m

Difusão Confiável (*Reliable Multicast*) – *Exemplo 2*



O traço de execução acima viola as propriedades de *multicast*?

Difusão Confiável (*Reliable Multicast*) – *Exemplo 2*

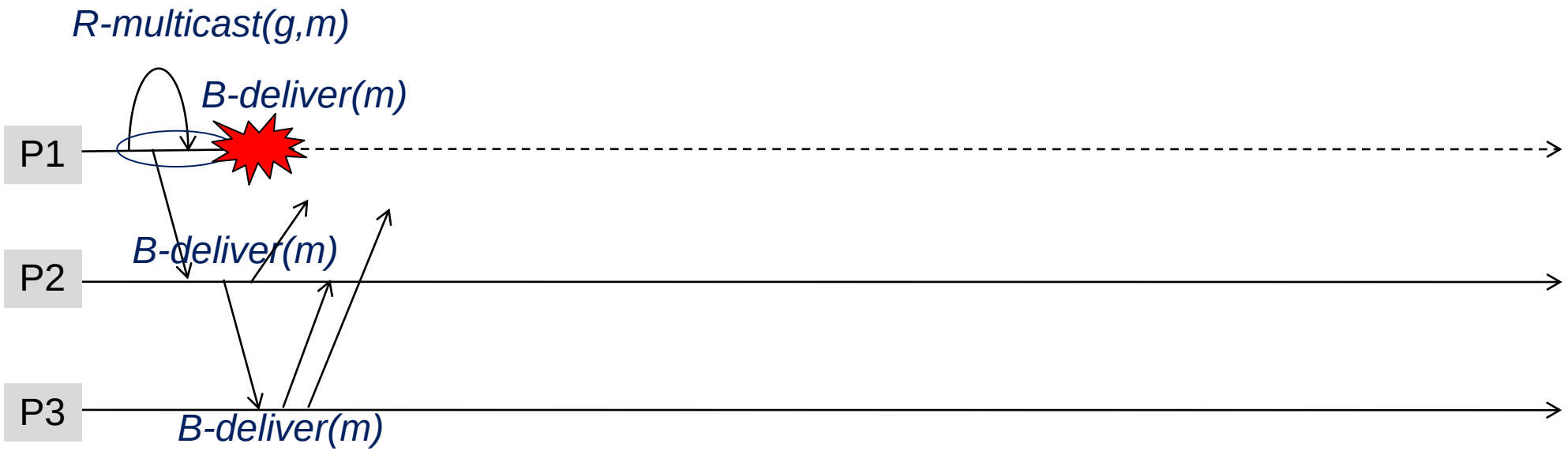


O traço de execução acima viola as propriedades de *multicast*?

Sim, se pelo menos um processo correto (P2) entrega m, então todos os processos corretos (inclusive P3) deveriam entregar m

Este traço não está correto!

Difusão Confiável (*Reliable Multicast*) – *Exemplo 2*



Observe que o processo P2, ao receber *m* executa um *B-multicast(m)*, garantindo que P3 recebe e entregue *m*

Difusão Confiável (*Reliable Multicast)*

- Propriedades Uniformes (*R-multicasting*)

  - **Integridade**: um processo correto $p \in g$ entrega uma mensagem *m* no máximo uma vez (além disso, *m* foi enviada: *R-multicast(g,m)*).

  - **Validade**: se um processo correto *p* executa *R-multicast(g,m),* então *p* entregará *m*

  - **Acordo Uniforme**: Se um processo (correto ou falho) $p \in g$ entrega *m*, então cada processo $q \in g$ entrega *m*

Difusão Confiável (*Reliable Multicast)*

On initialization
    Received := {};

For process p to R-multicast message m to group g
    B-multicast(*g*, *m*); // $p \in g$ is included as a destination

On B-deliver(*m*) *at process q with g = group*(*m*)
    if ($m \notin$ *Received*)
    then
        Received := *Received* $\cup$ {*m*};
        if ($q \neq p$) *then B-multicast*(*g*, *m*); *end if*
        R-deliver m;
    end if

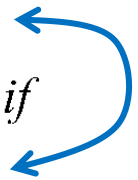
O algoritmo apresentado anteriormente (e descrito acima), implementa o acordo uniforme

Difusão Confiável (*Reliable Multicast)*

On initialization
    Received := {};

For process p to R-multicast message m to group g
    B-multicast(*g*, *m*); // $p \in g$ is included as a destination

On B-deliver(*m*) *at process q with g = group*(*m*)
    if ($m \notin$ *Received*)
    then
        Received := *Received* $\cup$ {*m*};
        if ($q \neq p$) *then B-multicast*(*g*, *m*); *end if*
        R-deliver m;
    end if



Se inverter estas linhas o acordo uniforme não é garantido

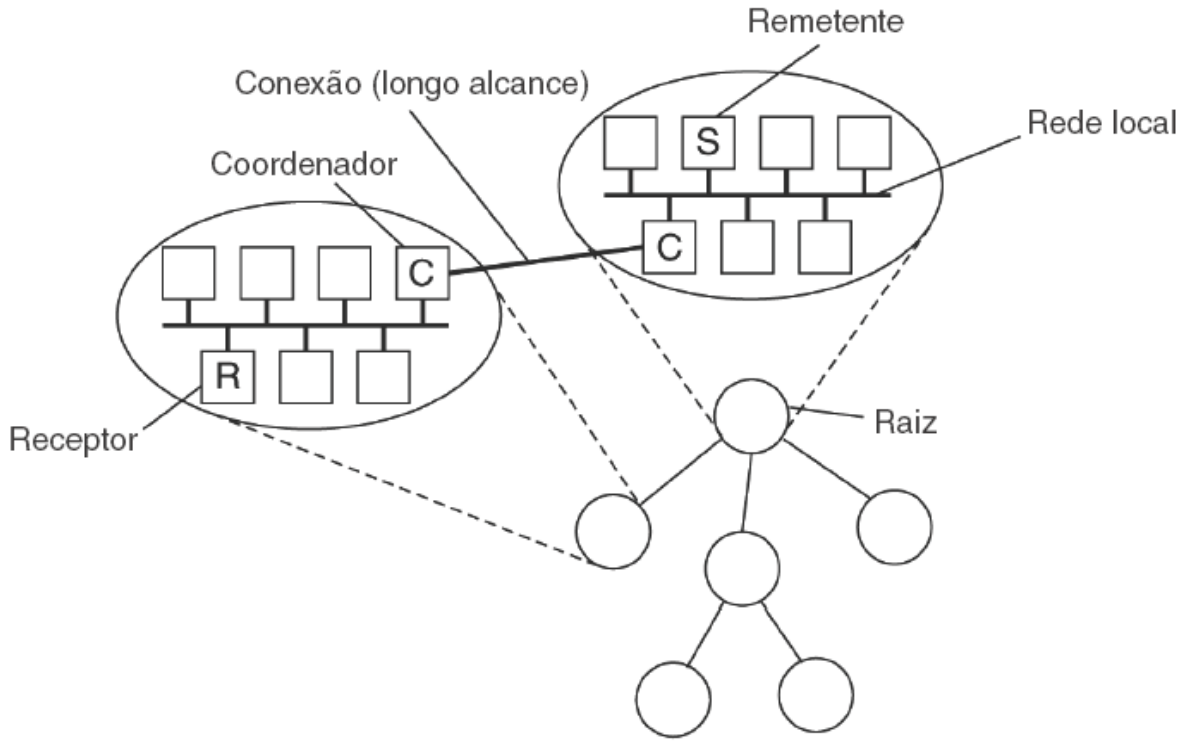
Algoritmos de Difusão (*multicast)* hierárquico

- Quando o grupo de processos é muito grande, o uso de algoritmos de difusão tradicional é desaconselhável (muitas trocas de mensagem)

- Abordagem hierárquica:
  - Grupos são particionados em sub-grupos
  - Um processo envia uma mensagem apenas para os coordenadores de cada partição
  - Os coordenadores efetuam a difusão apenas entre os membros de sua partição

Algoritmos de Difusão (*multicast)* hierárquico

Controle de recebimento de mensagens hierárquico



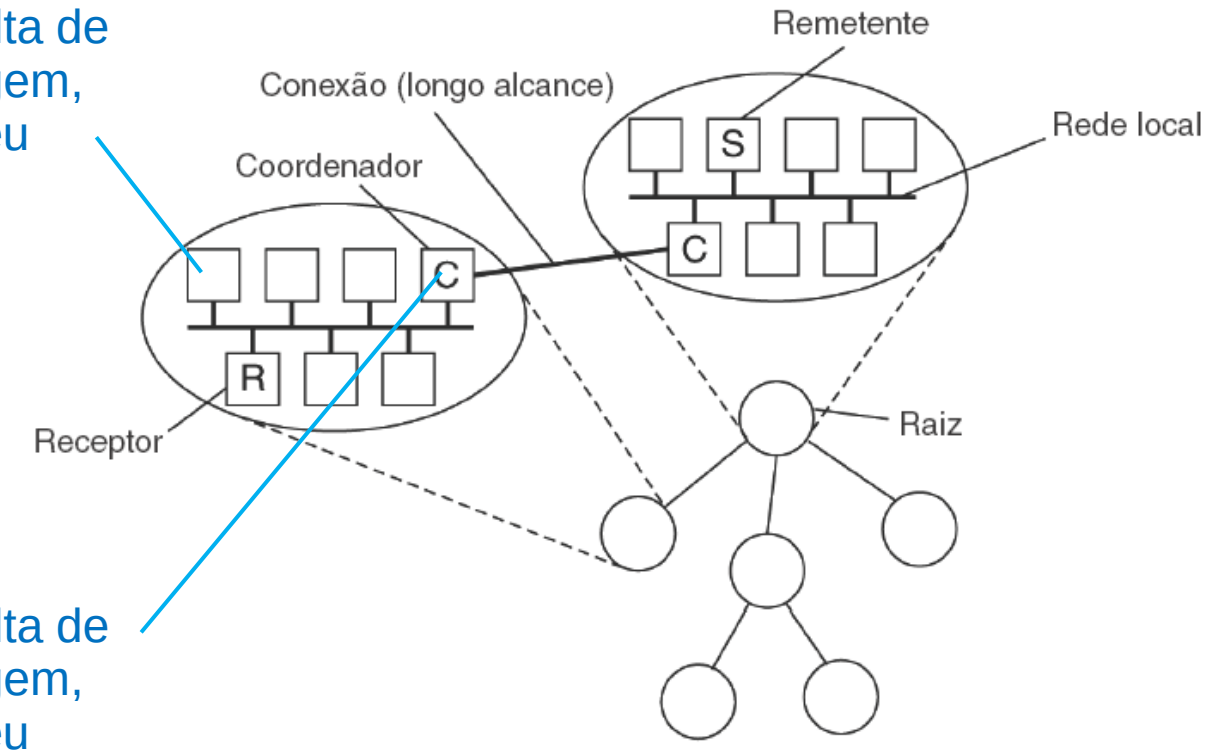
O grupo de receptores é organizado na forma de árvore.
Cada partição é um vértice da árvore

a) Cada coordenador local repassa a mensagem aos seus filhos
b) Um coordenador local trata as requisições de retransmissão

Algoritmos de Difusão (*multicast)* hierárquico

Controle de recebimento de mensagens hierárquico

Se um receptor percebe a falta de uma mensagem, solicita ao seu coordenador



Se um coordenador percebe a falta de uma mensagem, solicita ao seu coordenador pai

O grupo de receptores é organizado na forma de árvore.
Cada partição é um vértice da árvore

a) Cada coordenador local repassa a mensagem aos seus filhos
b) Um coordenador local trata as requisições de retransmissão

Difusão Confiável (*Reliable Multicast)* Ordenado

- É comum oferecer garantias sobre a ordem de entrega de mensagens *multicast*

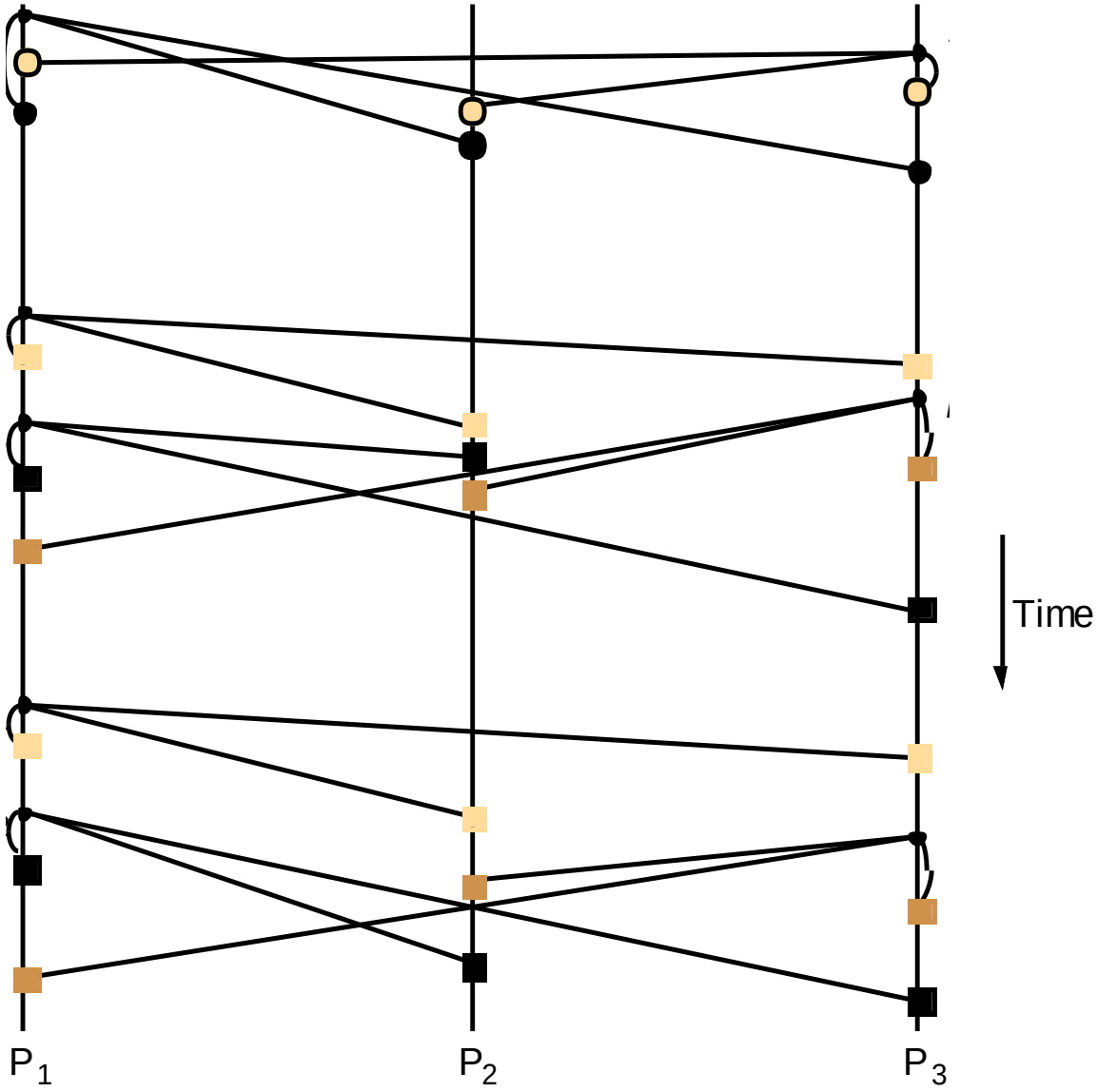
  - **FIFO**: se um processo correto executa *multicast(g,m)* e depois *multicast(g,m'),* então todo processo correto entregará *m* antes de *m'*

  - **CAUSAL**: se *multicast(g,m)* "acontece antes" de *multicast(g,m'),* então todo processo correto que entregar *m'* entregará *m* antes

  - **TOTAL**: Se um processo correto entregar *m* antes de *m'*, então todos os processos corretos entregarão *m* antes de *m'*

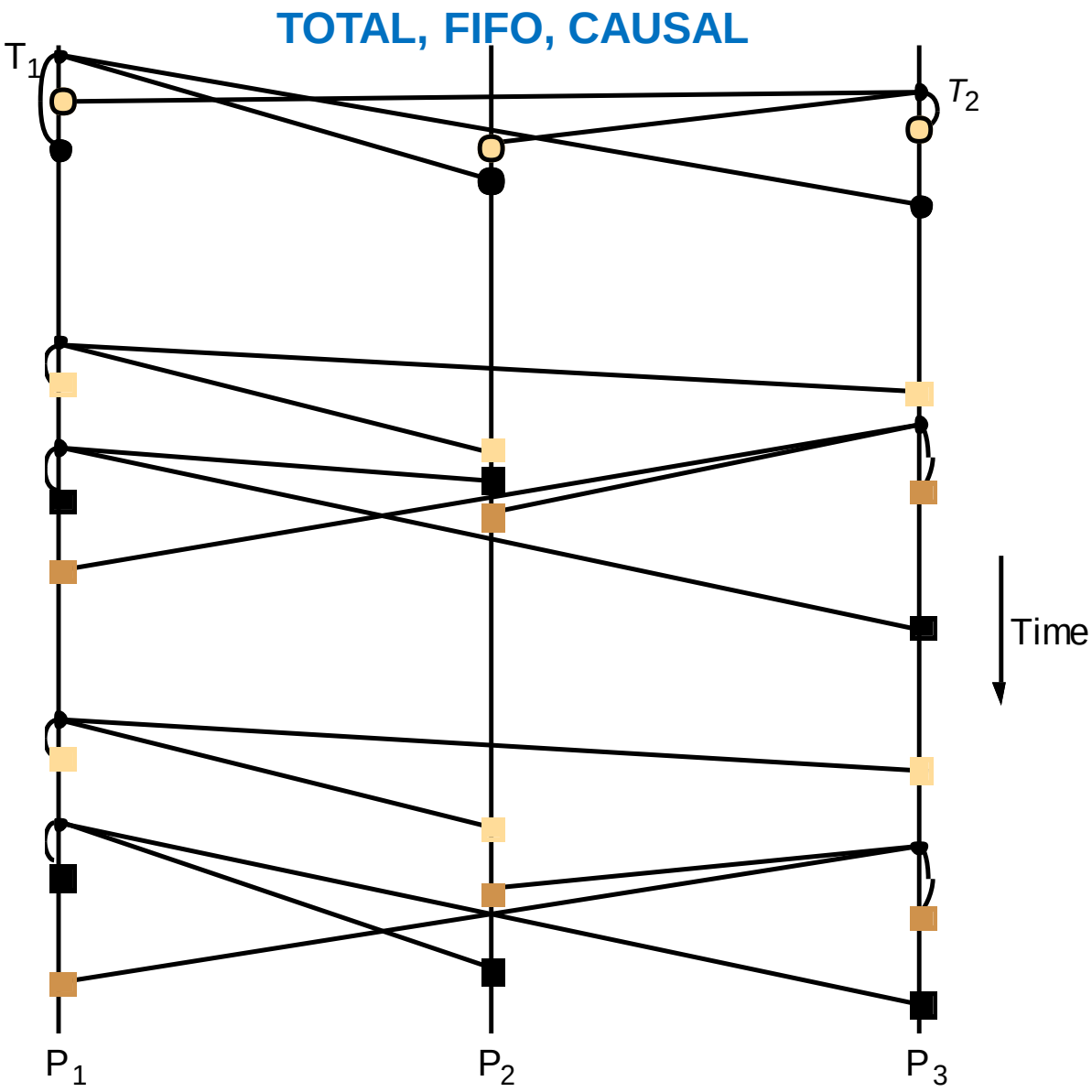
Difusão Confiável (*Reliable Multicast)* Ordenado

Quais as garantias na ordem de entrega implementadas no diagrama ao lado?



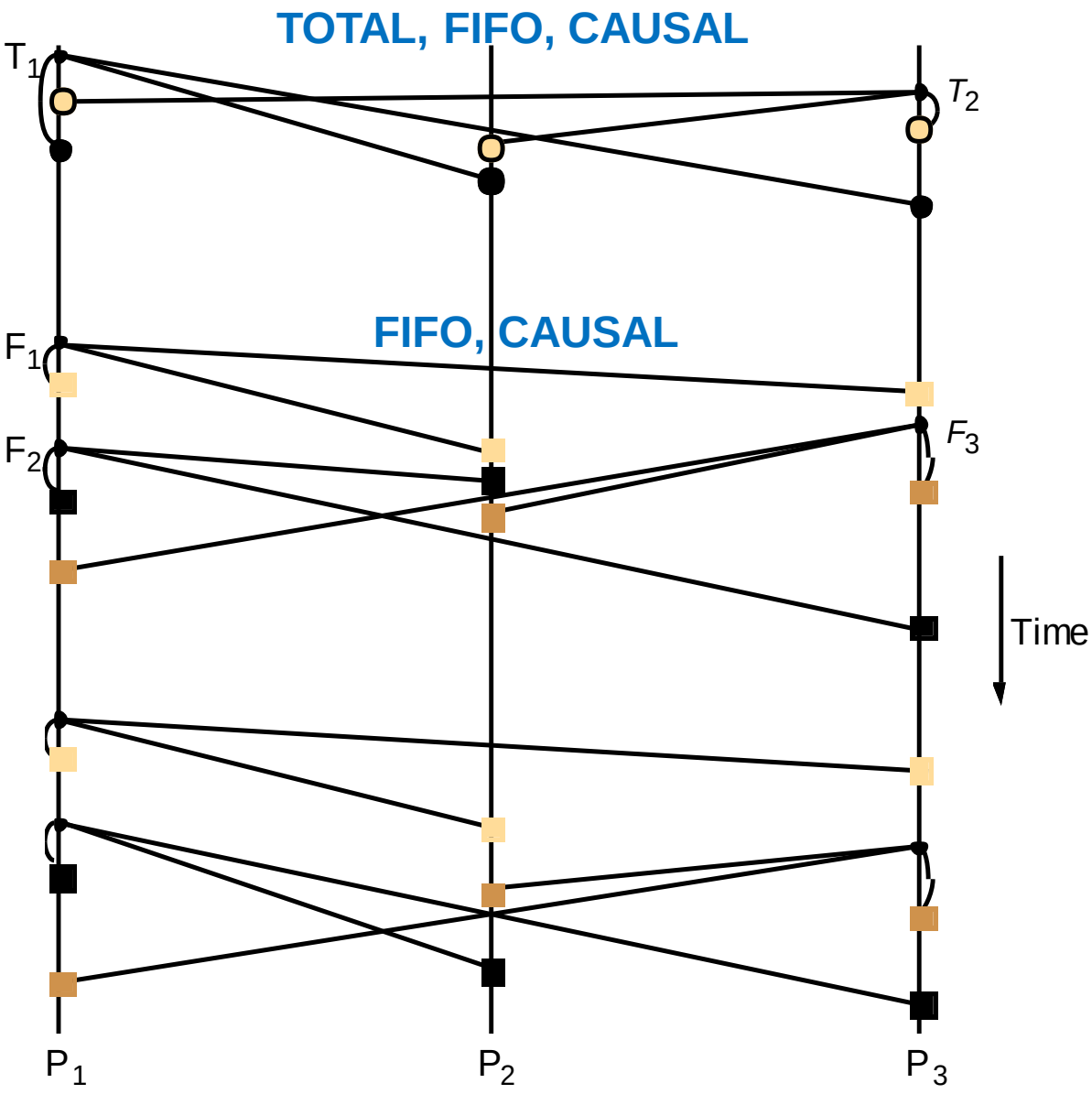
Difusão Confiável (*Reliable Multicast)* Ordenado

Quais as garantias na ordem de entrega implementadas no diagrama ao lado?



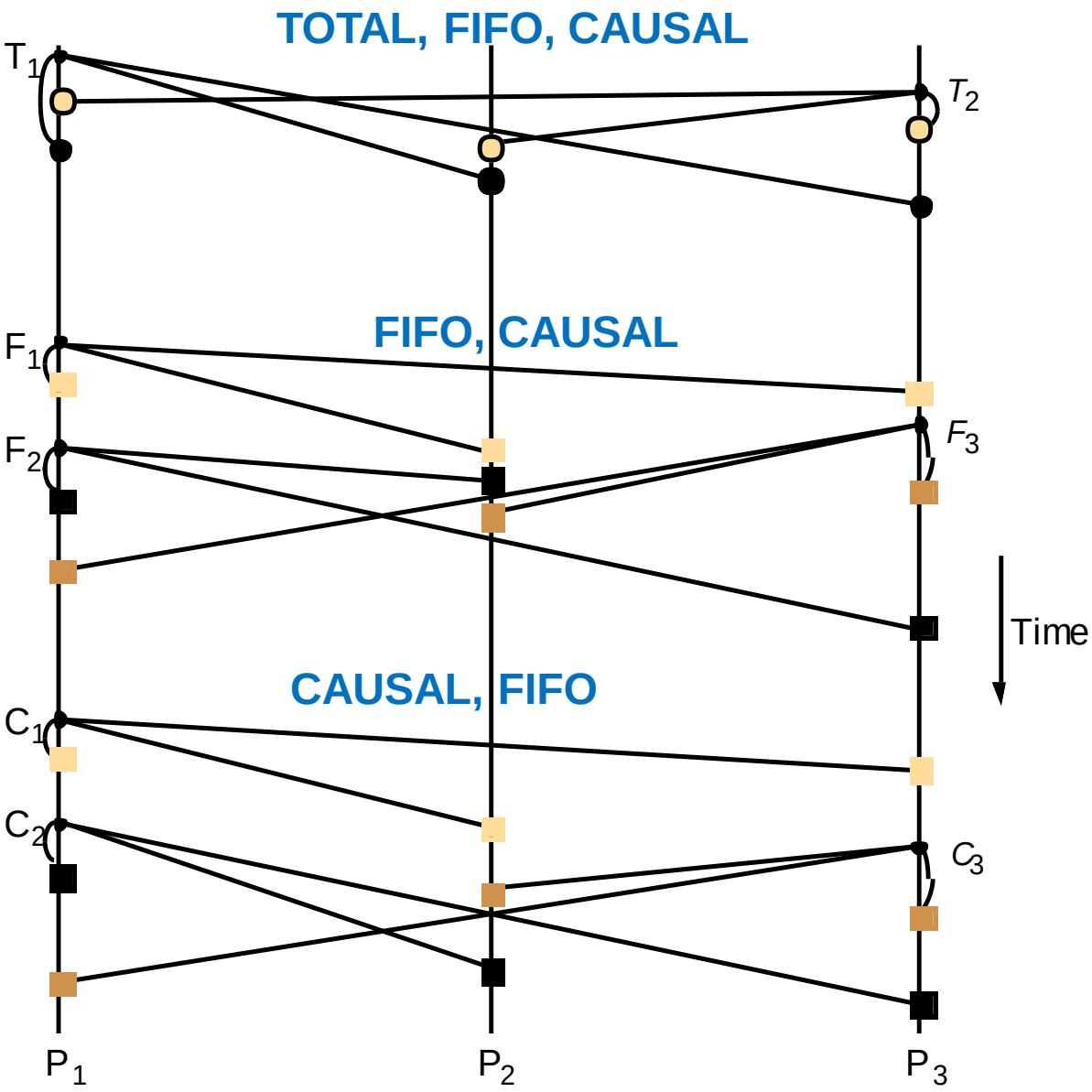
Difusão Confiável (*Reliable Multicast)* Ordenado

Quais as garantias na ordem de entrega implementadas no diagrama ao lado?



Difusão Confiável (*Reliable Multicast)* Ordenado

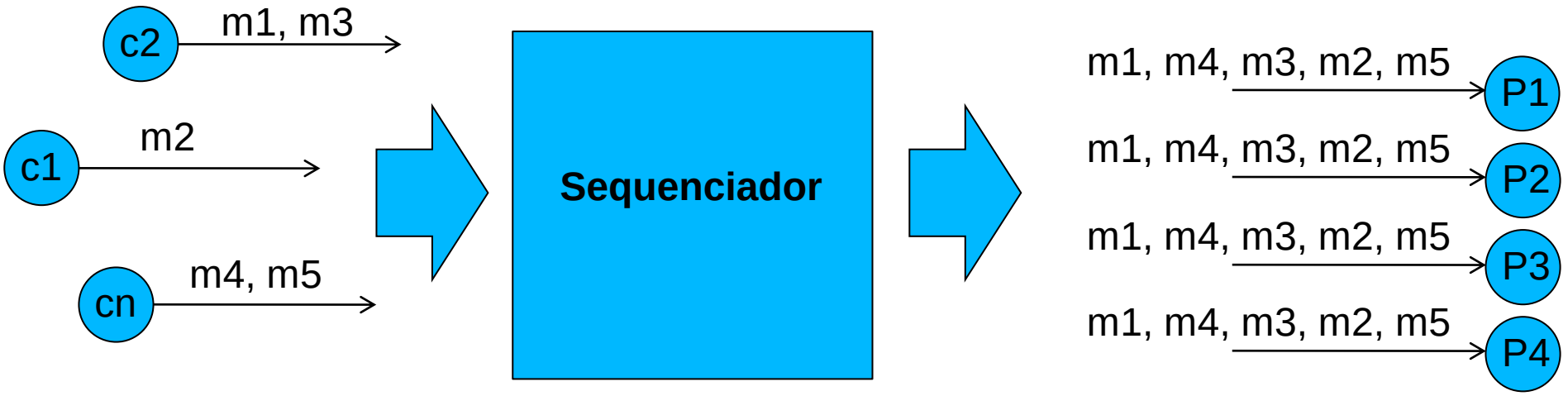
Quais as garantias na ordem de entrega implementadas no diagrama ao lado?



Difusão Confiável com Ordem Total (*Total Order Reliable Multicast)*

Implementação de ordem total com entrega confiável é comumente chamado de **difusão atômica** (*Atomic Multicast*)

Implementação pode utilizar um processo sequenciador

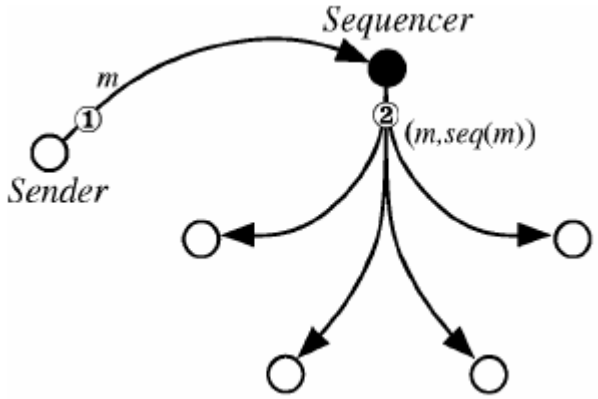


Difusão Confiável com Ordem Total (*Total Order Reliable Multicast)*

Abordagem 1:

Processos enviam a mensagem para o sequenciador

O sequenciador encaminha a mensagem com uma ordem associada



Sender:

procedure *TO-broadcast*(m) { To *TO-broadcast* a message m }
send (m) to sequencer

Sequencer:

Initialization:
$seqnum := 1$
when receive (m)
$sn(m) := seqnum$
send $(m, sn(m))$ to all
$seqnum := seqnum + 1$

Destinations (code of process p_i):

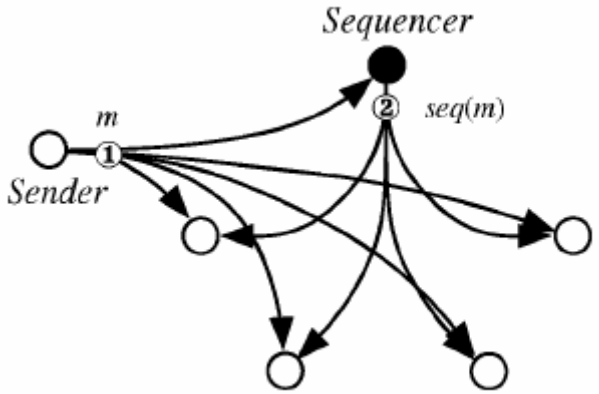
Initialization:
$nextdeliver_{p_i} := 1$
$pending_{p_i} := \emptyset$
when receive $(m, seqnum)$
$pending_{p_i} := pending_{p_i} \cup \{(m, seqnum)\}$
while $\exists (m', seqnum') \in pending_{p_i} : seqnum' = nextdeliver_{p_i}$ **do**
deliver (m')
$nextdeliver_{p_i} := nextdeliver_{p_i} + 1$

Difusão Confiável com Ordem Total (*Total Order Reliable Multicast)*

Abordagem 2:

Processos enviam a mensagem para membros do grupo e para o sequenciador

Mensagens só são entregues quando uma ordem é atribuída a elas



1. Algorithm for group member p

On initialization: $r_g := 0$;

To TO-multicast message m to group g
B-multicast($g \cup \{sequencer(g)\}$, <m, i>);

On B-deliver(<m, i>) *with* $g = group(m)$
Place <m, i> in hold-back queue;

On B-deliver(m_{order} = <"order", i, S>) *with* $g = group(m_{order})$
wait until <m, i> in hold-back queue and $S = r_g$;
TO-deliver m; // (after deleting it from the hold-back queue)
$r_g = S + 1$;

2. Algorithm for sequencer of g

On initialization: $s_g := 0$;

On B-deliver(<m, i>) *with* $g = group(m)$
B-multicast(g, <"order", i, s_g>);
$s_g := s_g + 1$;

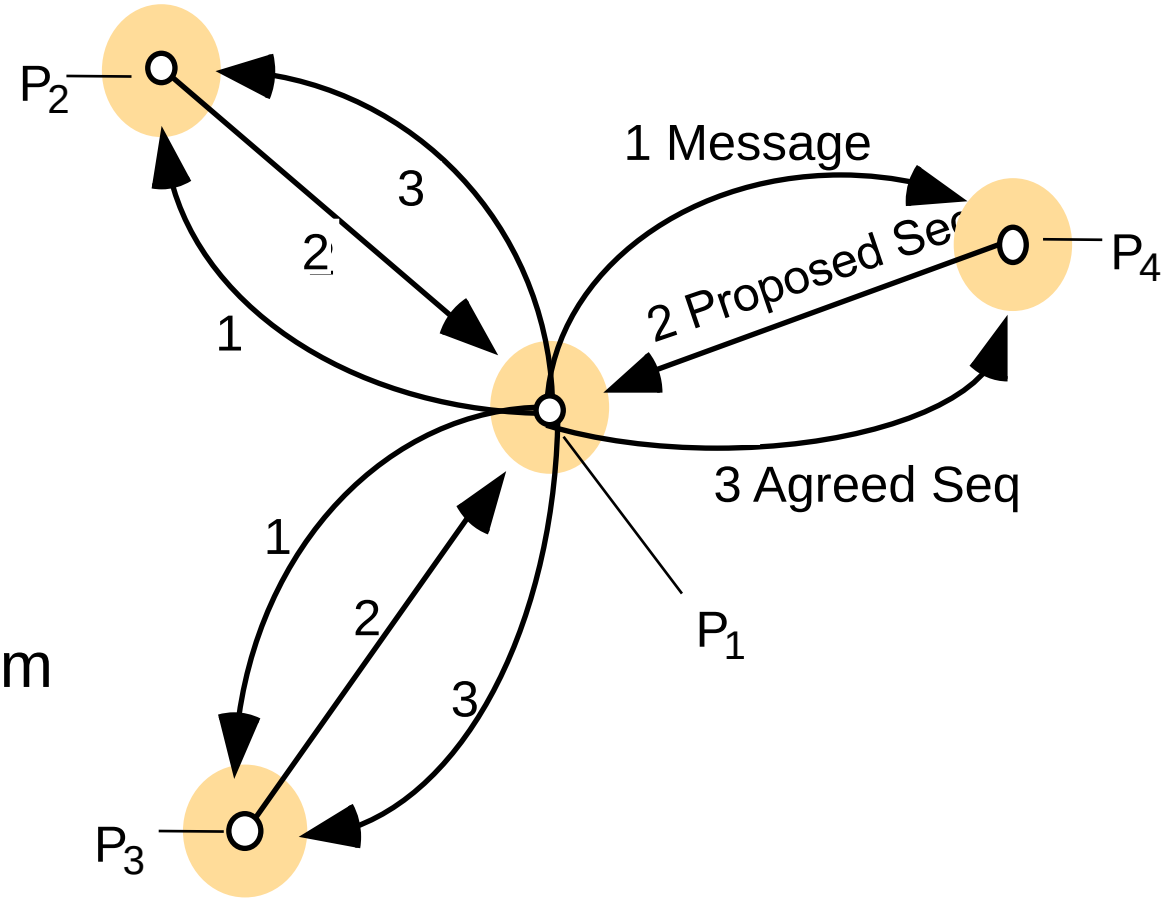
Difusão Confiável com Ordem Total (*Total Order Reliable Multicast)*

Implementação proposta no sistema ISIS (proposto por K. Birman) – sem sequenciador

Processo envia multicast(g,<m,i>), onde i é um identificador único para m

Processos propõe o seu valor máximo aceito para entrega de m

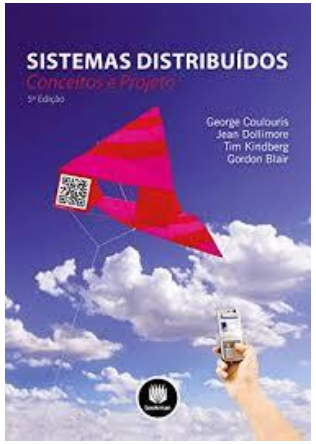
O proponente identifica o maior valor proposto e o envia aos demais processos

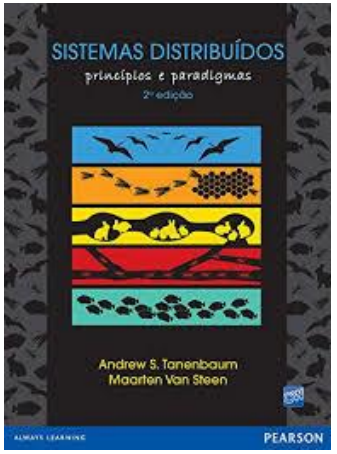


Processos agora sabem a ordem que devem entregar m

Referências

- *Coulouris, George; Dollimore, Jean; Kindberg, Tim; Blair, Gordon. Sistemas Distribuídos: Conceitos e Projetos. Bookman; 5ª edição. 2013.*
- *Tanenbaum, Andrew S.; Van Steen, Maarten. Sistemas Distribuídos: Princípios e Paradigmas. 2007. Pearson Universidades; 2ª edição.*
- *"Protocolos Fundamentais para o Desenvolvimento de Aplicações Robustas". Fabíola Greve. Mini-curso SBRC 2005*
- *- "Total Order Broadcast and Multicast Algorithms: Taxonomy and Survey". Xavier Défago, André Schiper, Péter Urbán. ACM Computing Surveys, Vol. 36, No. 4*





- *Imagens e clip arts diversos: https://free-icon-rainbow.com/ , https://www.gratispng.com/*