Fundamentos de Processamento Paralelo e Distribuído

modelos de comunicação

Fernando Luís Dotti

PUCRS | ESCOLA POLITÉCNICA

Como se especifica e como é construída toda esta funcionalidade ?

Abstrações para a Comunicação ponto a ponto

Material baseado no *livro e em slides* de: *Introduction to Reliable and Secure Distributed Programming Christian Cachin, Rachid Gerraoui, Luís Rodrigues*

foi exemplo no conjunto anterior de slides

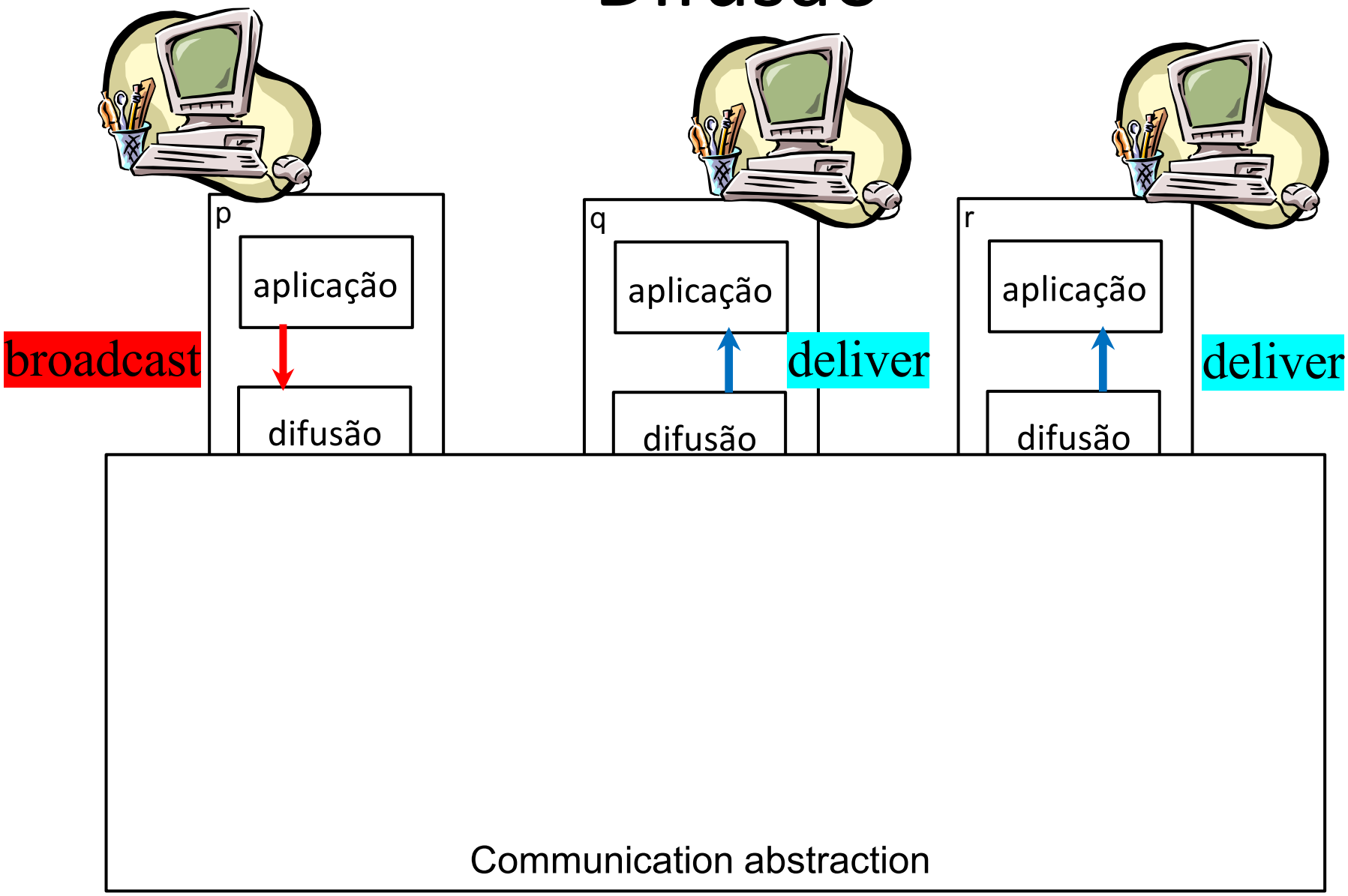
Comunicação em um grupo de processos - Difusão (broadcast)

Material baseado no *livro e em slides* de: *Introduction to Reliable and Secure Distributed Programming Christian Cachin, Rachid Gerraoui, Luís Rodrigues*

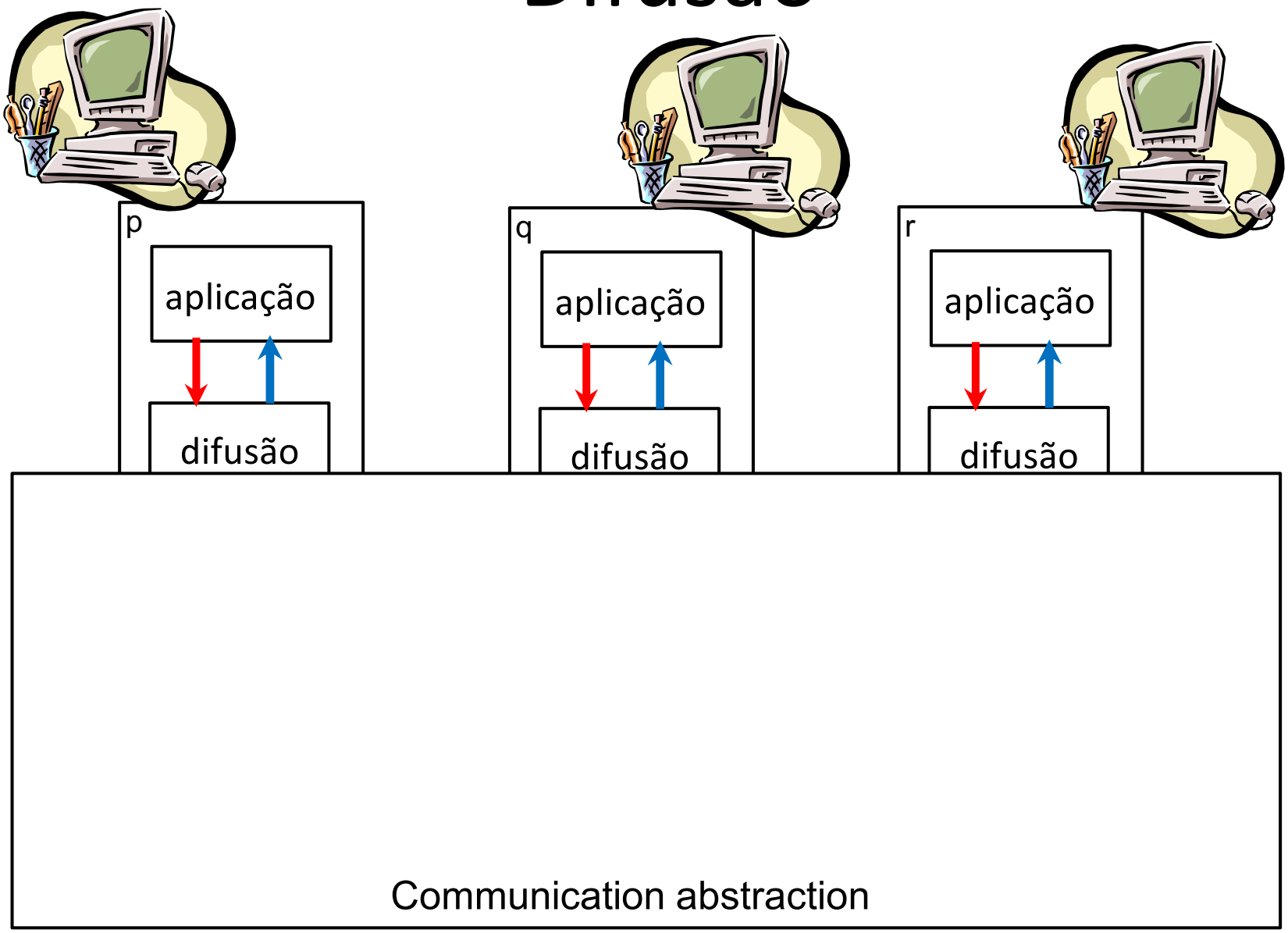
Tópicos

- Broadcast
  - Confiabilidade
    - Best effort
    - Reliable
      - Regular
      - Uniforme
  - Ordem
    - FIFO
    - Causal
    - (total – consenso)

Difusão



Difusão



Intuição

- Broadcast é util em diversas aplicações, e.g.:
  - onde processos (subscribers) desejam receber eventos de outros processos
  - em que processos colaboram para manter estado replicado consistente
  - em que processos representam diferentes partes em um diálogo único
    - jogos
    - edição colaborativa de documentos
  - ...

Níveis de Confiabilidade

- Três formas da primitiva de broadcast em relação à confiabilidade

(1) Best-effort broadcast
    na ocorrência de falha, nada é feito
(2) (Regular) reliable broadcast
    tolera falha do processo originador
(3) Uniform (reliable) broadcast
    tolera falha do originador dando maior garantia aos recebedores

Best Effort Broadcast

Material baseado no *livro e em slides* de: *Introduction to Reliable and Secure Distributed Programming Christian Cachin, Rachid Gerraoui, Luís Rodrigues*

Best-effort broadcast (beb)

Module 3.1: Interface and properties of best-effort broadcast

Module:

Name: BestEffortBroadcast, **instance** *beb*.

Events:

Request: $\langle$ *beb*, *Broadcast* | m $\rangle$: Broadcasts a message m to all processes.

Indication: $\langle$ *beb*, *Deliver* | p, m $\rangle$: Delivers a message m broadcast by process p.

Properties:

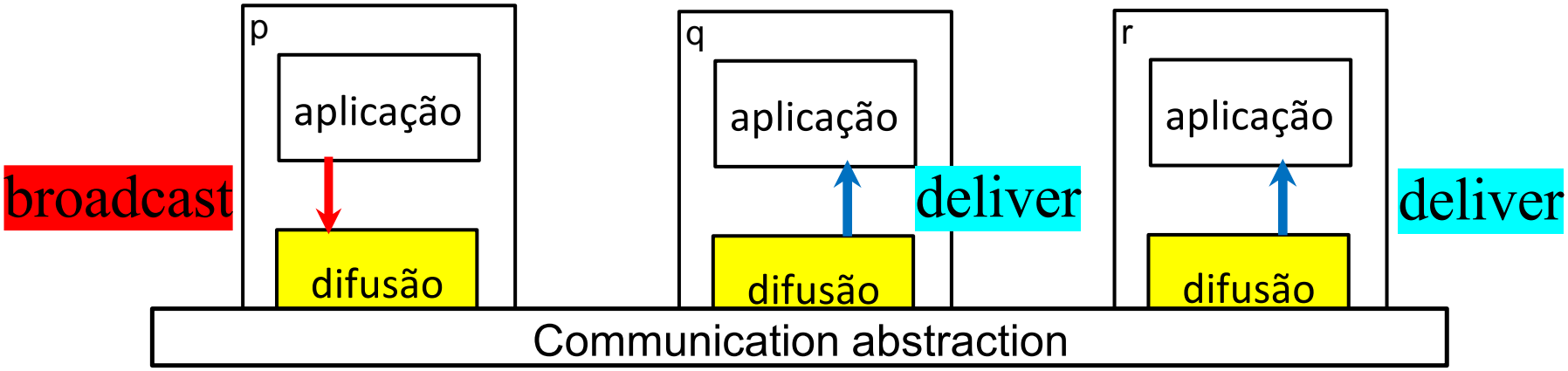
==>

BEB1: *Validity:* If a correct process broadcasts a message m, then every correct process eventually delivers m.

BEB2: *No duplication:* No message is delivered more than once.

BEB3: *No creation:* If a process delivers a message m with sender s, then m was previously broadcast by process s.

Difusão – Best Effort



Module 3.1: Interface and properties of best-effort broadcast

Module:

Name: BestEffortBroadcast, **instance** *beb*.

Events:

Request: ⟨ *beb*, *Broadcast* | m ⟩: Broadcasts a message m to all processes.

Indication: ⟨ *beb*, *Deliver* | p, m ⟩: Delivers a message m broadcast by process p.

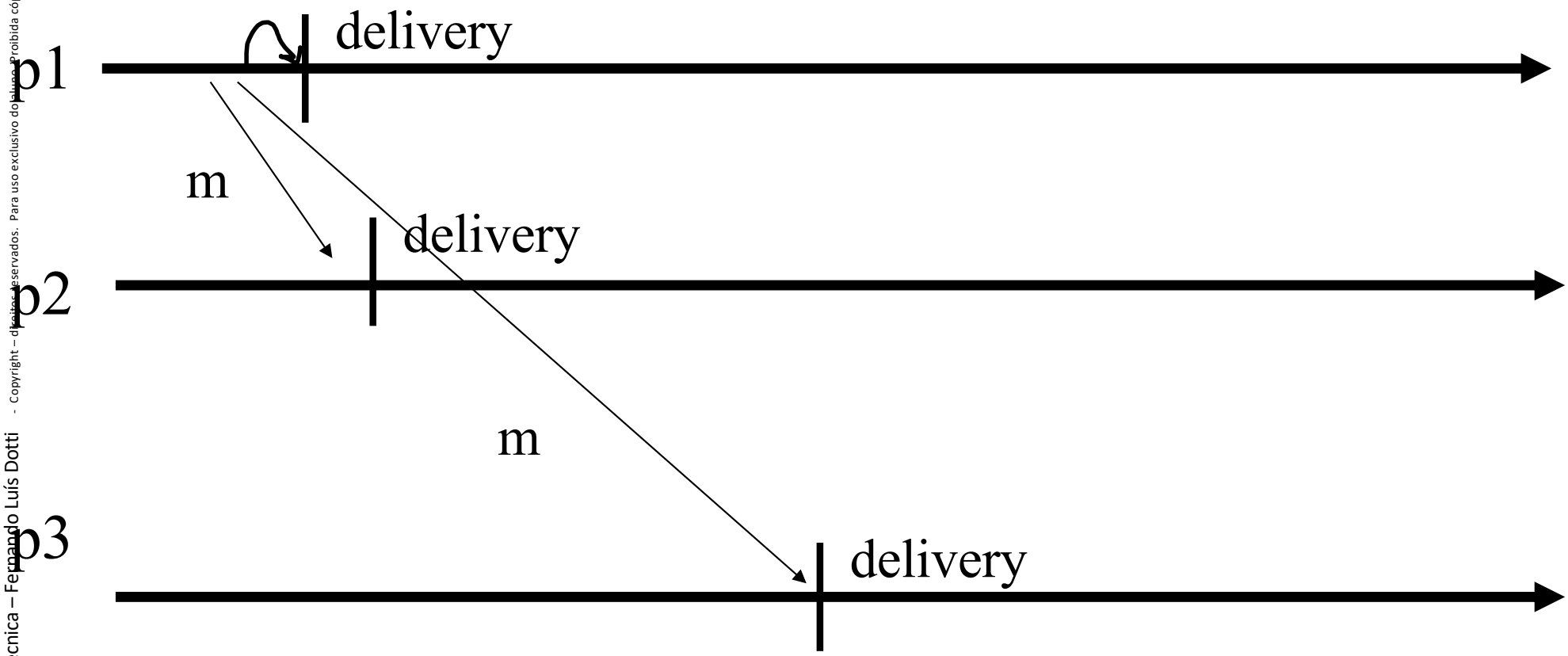
Properties:

BEB1: *Validity:* If a correct process broadcasts a message m, then every correct process eventually delivers m.

BEB2: *No duplication:* No message is delivered more than once.

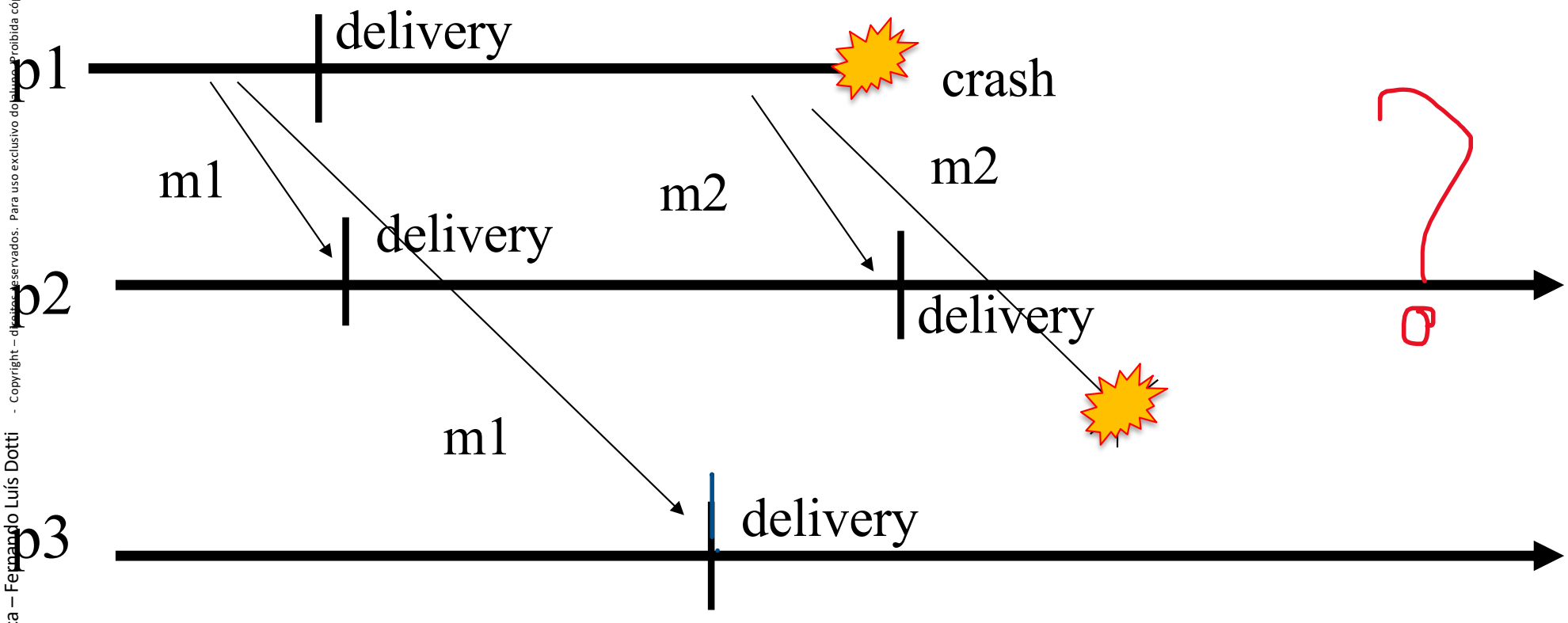
BEB3: *No creation:* If a process delivers a message m with sender s, then m was previously broadcast by process s.

Best-effort broadcast



Material baseado no *livro e em slides* de: *Introduction to Reliable and Secure Distributed Programming Christian Cachin, Rachid Gerraoui, Luís Rodrigues*

Best-effort broadcast

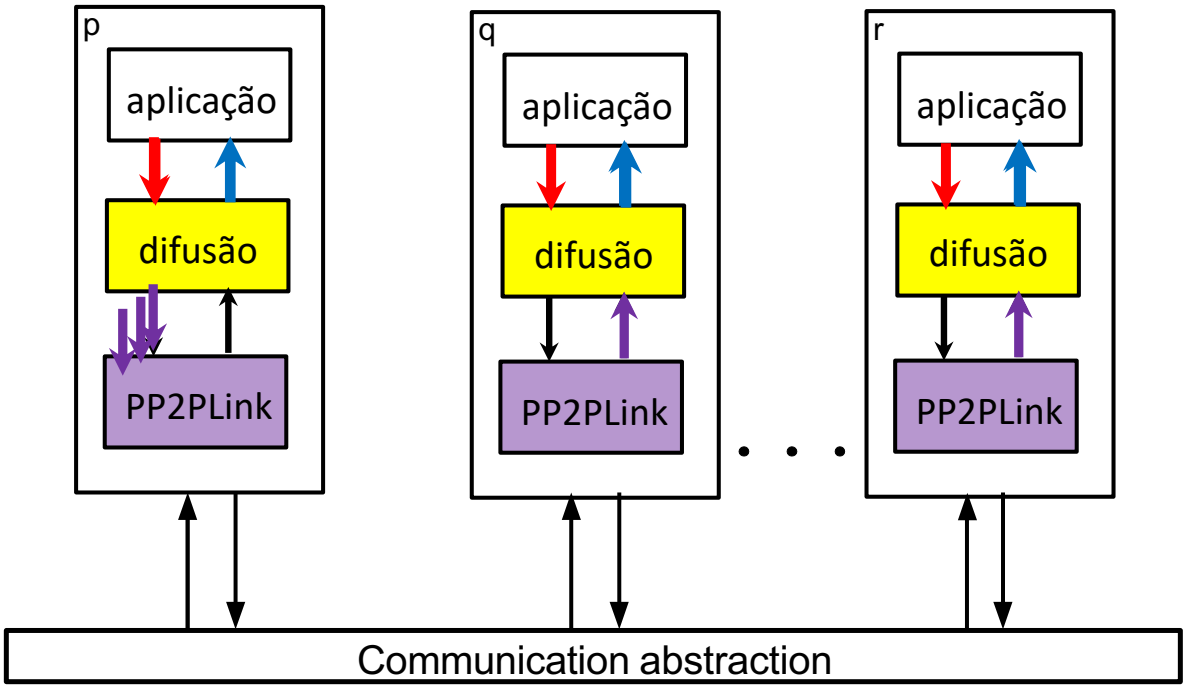


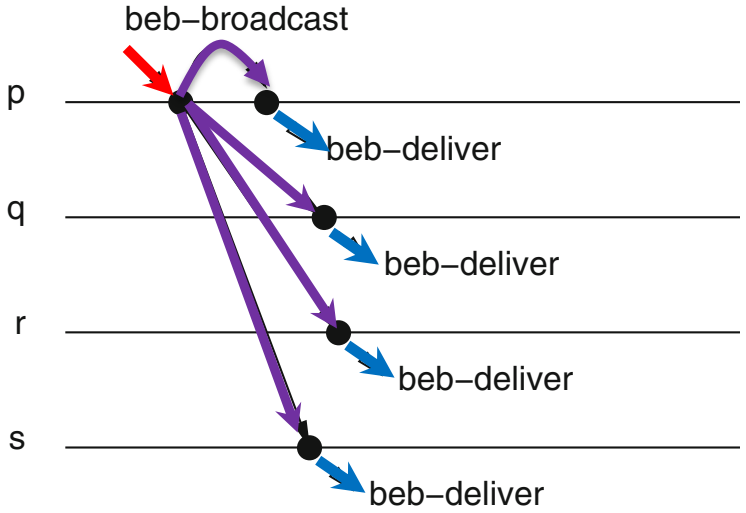
Material baseado no *livro e em slides* de: *Introduction to Reliable and Secure Distributed Programming Christian Cachin, Rachid Gerraoui, Luís Rodrigues*

Algoritmo (beb)

Material baseado no *livro e em slides* de: *Introduction to Reliable and Secure Distributed Programming Christian Cachin, Rachid Gerraoui, Luís Rodrigues*

Difusão – Best Effort





Material baseado no *livro e em slides* de: *Introduction to Reliable and Secure Distributed Programming Christian Cachin, Rachid Gerraoui, Luís Rodrigues*

Algoritmo (beb)

- Fail-silent: Basic Broadcast

Algorithm 3.1: Basic Broadcast

Implements:

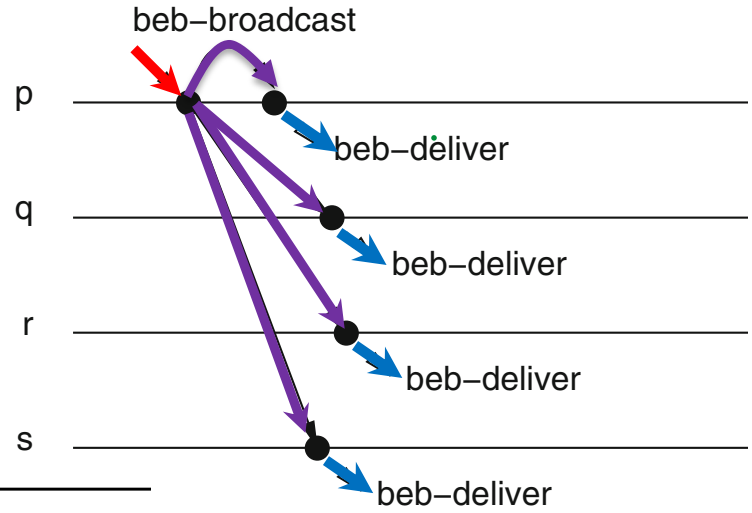
BestEffortBroadcast, **instance** *beb*.

Uses:

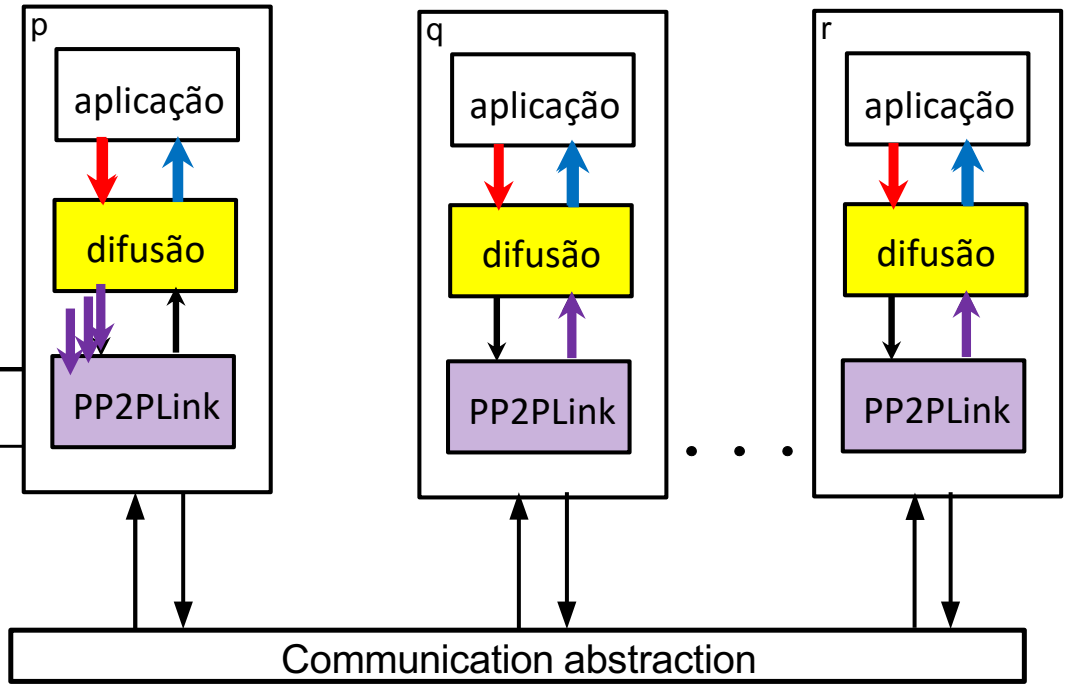
PerfectPointToPointLinks, **instance** *pl*.

upon event $\langle\ beb, Broadcast \mid m\ \rangle$ **do**
  forall $q \in \Pi$ **do**
    trigger $\langle\ pl, Send \mid q, m\ \rangle$;

upon event $\langle\ pl, Deliver \mid p, m\ \rangle$ **do**
  trigger $\langle\ beb, Deliver \mid p, m\ \rangle$;



Algoritmo (beb)



Algorithm 3.1: Basic Broadcast

Implements:

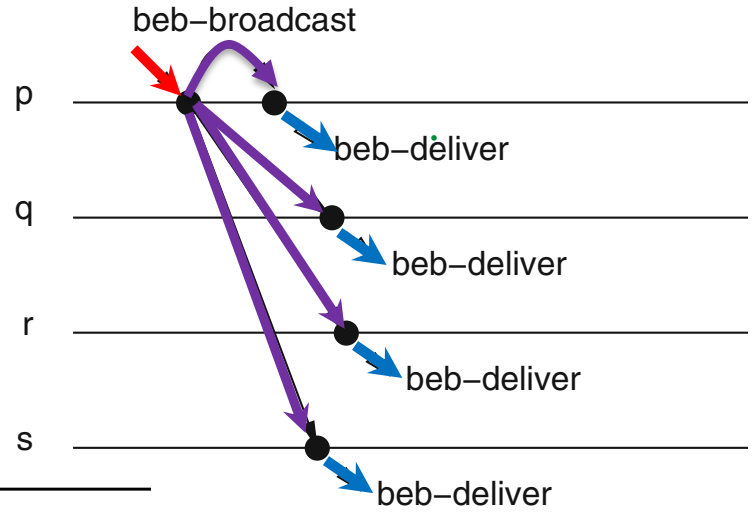
BestEffortBroadcast, **instance** *beb*.

Uses:

PerfectPointToPointLinks, **instance** *pl*.

upon event $\langle$ *beb*, *Broadcast* | m $\rangle$ **do**
  forall $q \in \Pi$ **do**
    trigger $\langle$ *pl*, *Send* | q, m $\rangle$;

upon event $\langle$ *pl*, *Deliver* | p, m $\rangle$ **do**
  trigger $\langle$ *beb*, *Deliver* | p, m $\rangle$;



Material baseado no *livro e em slides* de: *Introduction to Reliable and Secure Distributed Programming Christian Cachin, Rachid Gerraoui, Luís Rodrigues*

Algoritmo (beb)

- *Algoritmo implementa as propriedades ?*

Algoritmo (beb)

implementa as propriedades ?

Algorithm 3.1: Basic Broadcast

Implements:

BestEffortBroadcast, **instance** *beb*.

Uses:

PerfectPointToPointLinks, **instance** *pl*.

upon event $\langle$ *beb*, *Broadcast* | m $\rangle$ **do**
  forall $q \in \Pi$ **do**
    trigger $\langle$ *pl*, *Send* | q, m $\rangle$;

upon event $\langle$ *pl*, *Deliver* | p, m $\rangle$ **do**
  trigger $\langle$ *beb*, *Deliver* | p, m $\rangle$;

Argumentação (prova)

Module 3.1: Interface and properties of best-effort broadcast

Module:

Name: BestEffortBroadcast, **instance** *beb*.

Events:

Request: $\langle$ *beb*, *Broadcast* | m $\rangle$: Broadcasts a message m to all processes.

Indication: $\langle$ *beb*, *Deliver* | p, m $\rangle$: Delivers a message m broadcast by process p.

Properties:

BEB1: *Validity:* If a correct process broadcasts a message m, then every correct process eventually delivers m.

BEB2: *No duplication:* No message is delivered more than once.

BEB3: *No creation:* If a process delivers a message m with sender s, then m was previously broadcast by process s.

Module 2.3: Interface and properties of perfect point-to-point links

Module:

Name: PerfectPointToPointLinks, **instance** *pl*.

Events:

Request: $\langle$ *pl*, *Send* | q, m $\rangle$: Requests to send message m to process q.

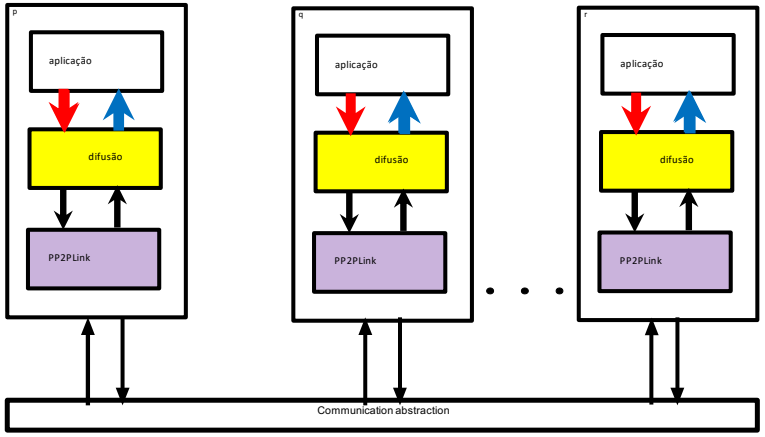
Indication: $\langle$ *pl*, *Deliver* | p, m $\rangle$: Delivers message m sent by process p.

Properties:

PL1: *Reliable delivery:* If a correct process p sends a message m to a correct process q, then q eventually delivers m.

PL2: *No duplication:* No message is delivered by a process more than once.

PL3: *No creation:* If some process q delivers a message m with sender p, then m was previously sent to q by process p.



Material baseado no *livro e em slides* de: *Introduction to Reliable and Secure Distributed Programming* *Christian Cachin, Rachid Gerraoui, Luís Rodrigues*

Best-effort broadcast (beb)

- *... Falha no processo origem, durante o envio ...*
  - *Alguns processos entregam, outros não, uma mesma mensagem*
  - *Eles nao estão em « acordo » com relação à entrega da mensagem*

Reliable Broadcast

Material baseado no *livro e em slides* de: *Introduction to Reliable and Secure Distributed Programming Christian Cachin, Rachid Gerraoui, Luís Rodrigues*

Reliable broadcast (rb)

Module 3.2: Interface and properties of (regular) reliable broadcast

Module:

Name: ReliableBroadcast, **instance** *rb*.

Events:

Request: $\langle$ *rb*, *Broadcast* | m $\rangle$: Broadcasts a message m to all processes.

Indication: $\langle$ *rb*, *Deliver* | p, m $\rangle$: Delivers a message m broadcast by process p.

Properties:

=beb-broadcast

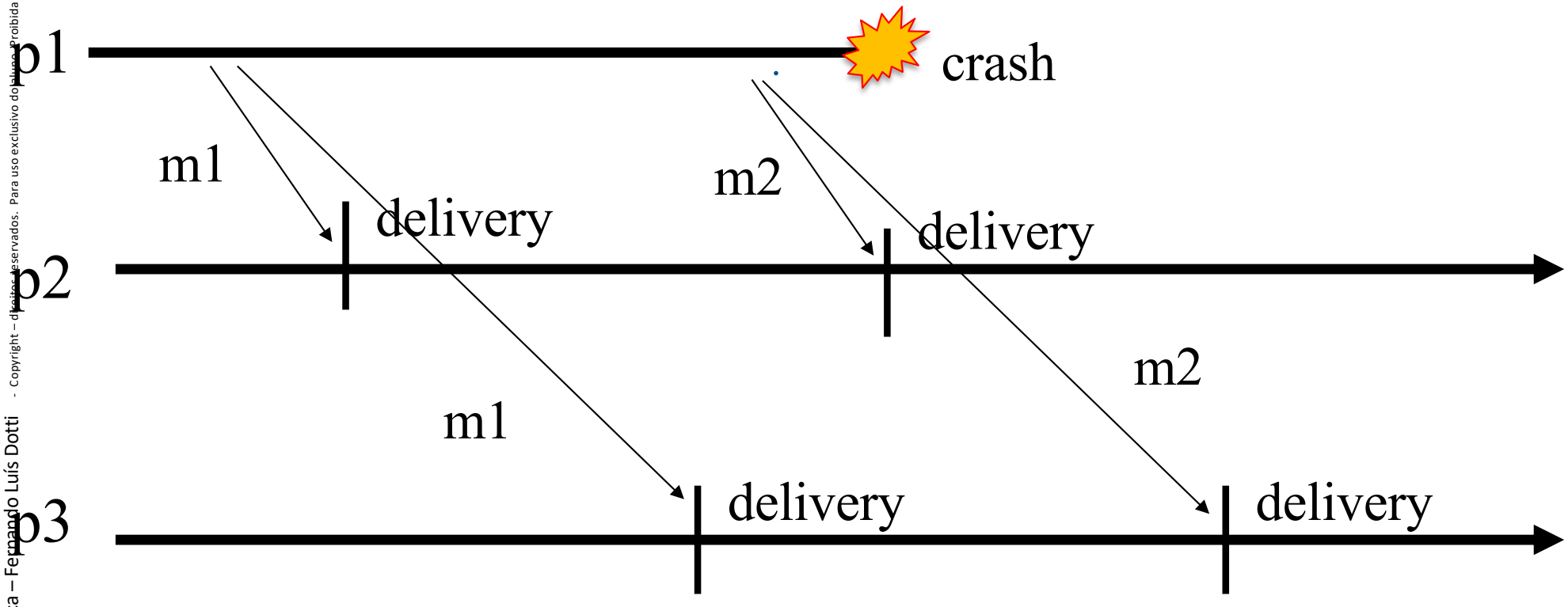
RB1: *Validity:* If a correct process p broadcasts a message m, then p eventually delivers m.

RB2: *No duplication:* No message is delivered more than once.

RB3: *No creation:* If a process delivers a message m with sender s, then m was previously broadcast by process s.

RB4: *Agreement:* If a message m is delivered by some correct process, then m is eventually delivered by every correct process.

Reliable broadcast



Material baseado no *livro e em slides* de: *Introduction to Reliable and Secure Distributed Programming Christian Cachin, Rachid Gerraoui, Luís Rodrigues*

Algorithm (rb)

Algorithm 3.3: Eager Reliable Broadcast

Implements:
ReliableBroadcast, **instance** *rb*.

Uses:
BestEffortBroadcast, **instance** *beb*.

upon event ⟨ *rb*, *Init* ⟩ **do**
delivered := ∅;

upon event ⟨ *rb*, *Broadcast* | m ⟩ **do**
trigger ⟨ *beb*, *Broadcast* | [DATA, *self*, m] ⟩;

upon event ⟨ *beb*, *Deliver* | p, [DATA, s, m] ⟩ **do**
if $m \notin$ *delivered* **then**
delivered := *delivered* ∪ {m};
trigger ⟨ *rb*, *Deliver* | s, m ⟩;
trigger ⟨ *beb*, *Broadcast* | [DATA, s, m] ⟩;

- Fail-silent alg: *eager* reliable
  - Retransmite sempre

Algorithm (rb)

Algorithm 3.3: Eager Reliable Broadcast

Implements:

ReliableBroadcast, **instance** *rb*.

Uses:

BestEffortBroadcast, **instance** *beb*.

upon event ⟨ *rb*, *Init* ⟩ **do**

delivered := ∅;

upon event ⟨ *rb*, *Broadcast* | *m* ⟩ **do**

trigger ⟨ *beb*, *Broadcast* | [DATA, *self*, *m*] ⟩;

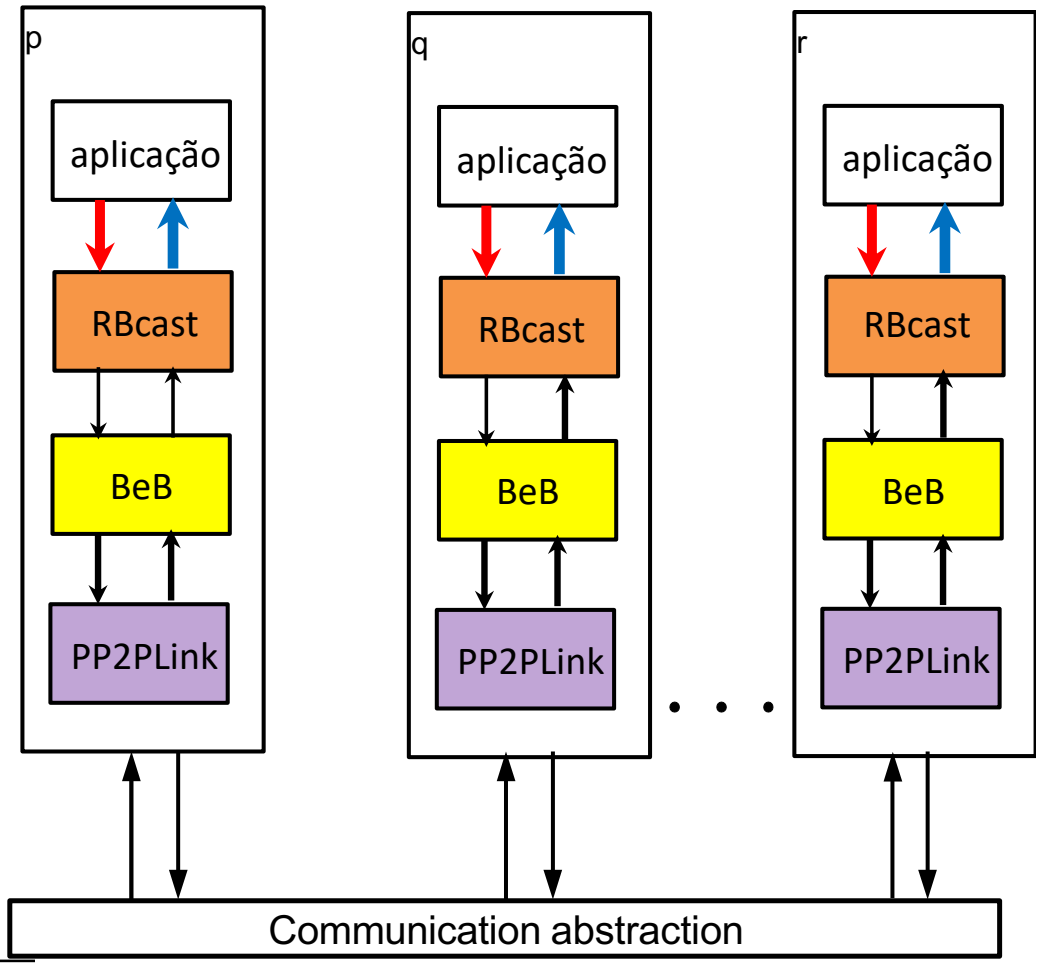
upon event ⟨ *beb*, *Deliver* | *p*, [DATA, *s*, *m*] ⟩ **do**

if *m* ∉ *delivered* **then**

delivered := *delivered* ∪ {*m*};

trigger ⟨ *rb*, *Deliver* | *s*, *m* ⟩;

trigger ⟨ *beb*, *Broadcast* | [DATA, *s*, *m*] ⟩;



Algoritmo (rb)

- Garantia de acordo mesmo quando sender falha:

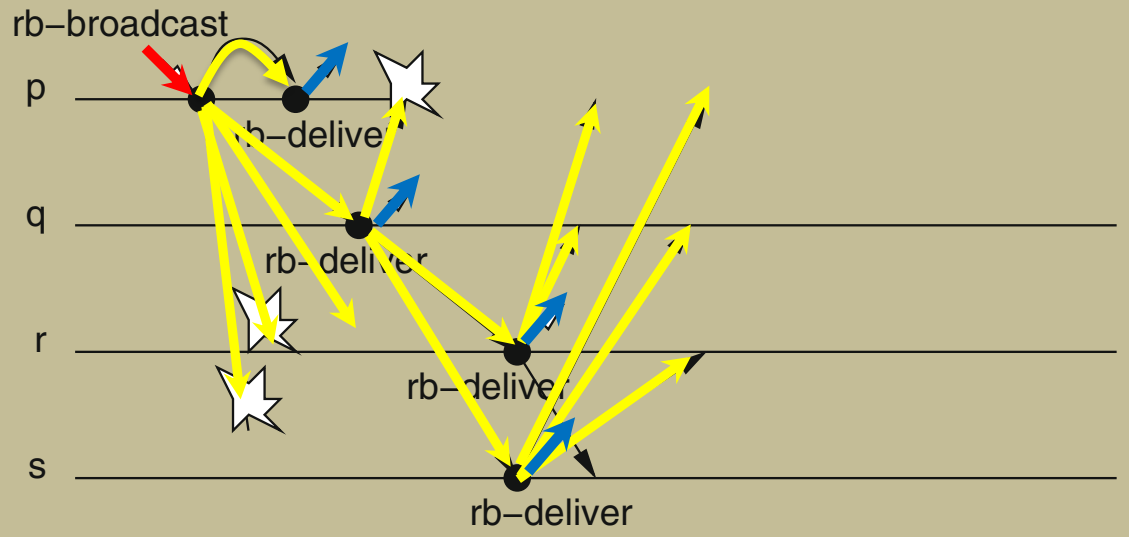


Figure 3.2: Sample execution of reliable broadcast with faulty sender

Material baseado no *livro e em slides* de: *Introduction to Reliable and Secure Distributed Programming Christian Cachin, Rachid Gerraoui, Luís Rodrigues*

Algoritmo (rb)

- Garantia de acordo mesmo quando sender falha:

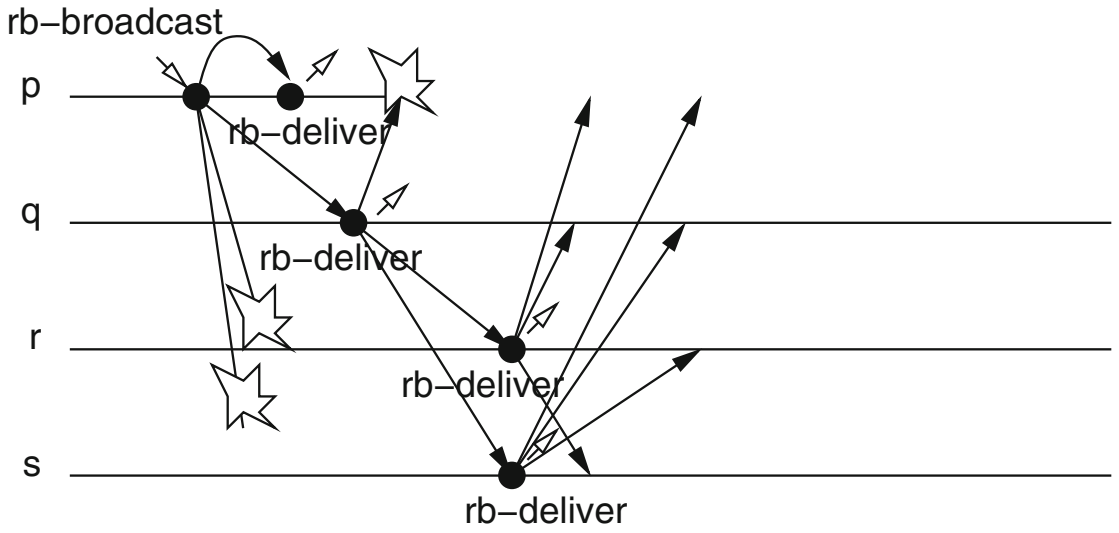


Figure 3.2: Sample execution of reliable broadcast with faulty sender

O(N^2): para cada RBroadcast,
N^2 mensagens ponto a ponto, onde N é o nro de processos

Algorithm (rb)

Algorithm 3.2: Lazy Reliable Broadcast

Implements:
ReliableBroadcast, **instance** *rb*.

Uses:
BestEffortBroadcast, **instance** *beb*;
PerfectFailureDetector, **instance** $\mathcal{P}$.

upon event $\langle$ *rb*, *Init* $\rangle$ **do**
correct := Π;
from[*p*] := $[\emptyset]^N$;

upon event $\langle$ *rb*, *Broadcast* | m $\rangle$ **do**
trigger $\langle$ *beb*, *Broadcast* | [DATA, *self*, m] $\rangle$;

upon event $\langle$ *beb*, *Deliver* | p, [DATA, s, m] $\rangle$ **do**
if $m \notin$ *from*[s] **then**
trigger $\langle$ *rb*, *Deliver* | s, m $\rangle$;
from[s] := *from*[s] $\cup$ {m};
if $s \notin$ *correct* **then**
trigger $\langle$ *beb*, *Broadcast* | [DATA, s, m] $\rangle$;

upon event $\langle$ $\mathcal{P}$, *Crash* | p $\rangle$ **do**
correct := *correct* \ {p};
forall $m \in$ *from*[p] **do**
trigger $\langle$ *beb*, *Broadcast* | [DATA, p, m] $\rangle$;

- Fail-stop alg: *lazy* reliable
  - Retransmite somente se detecta falho

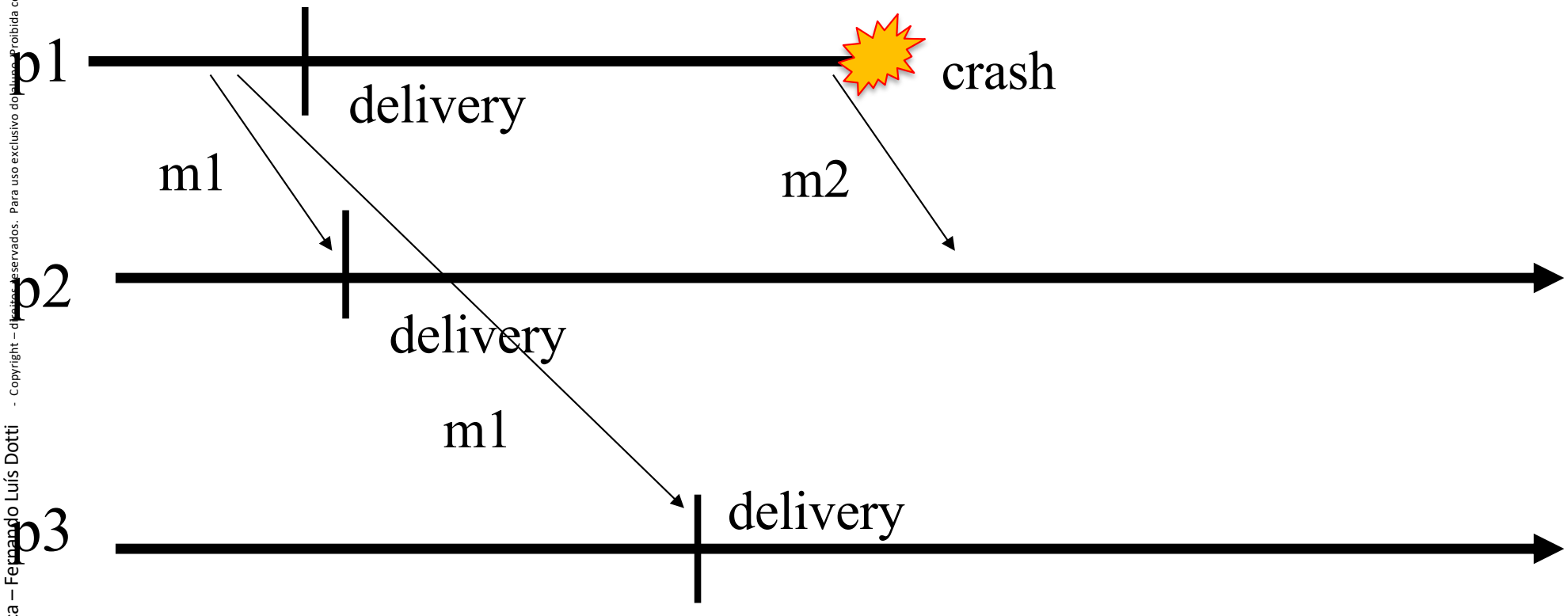
[DATA, *s*, *m*] :
DATA = DESCRITOR DA MENSAGEM
s = *fonte*
m = mensagem

Material baseado no *livro e em slides* de: *Introduction to Reliable and Secure Distributed Programming* *Christian Cachin, Rachid Gerraoui, Luís Rodrigues*

Uniform R-Broadcast

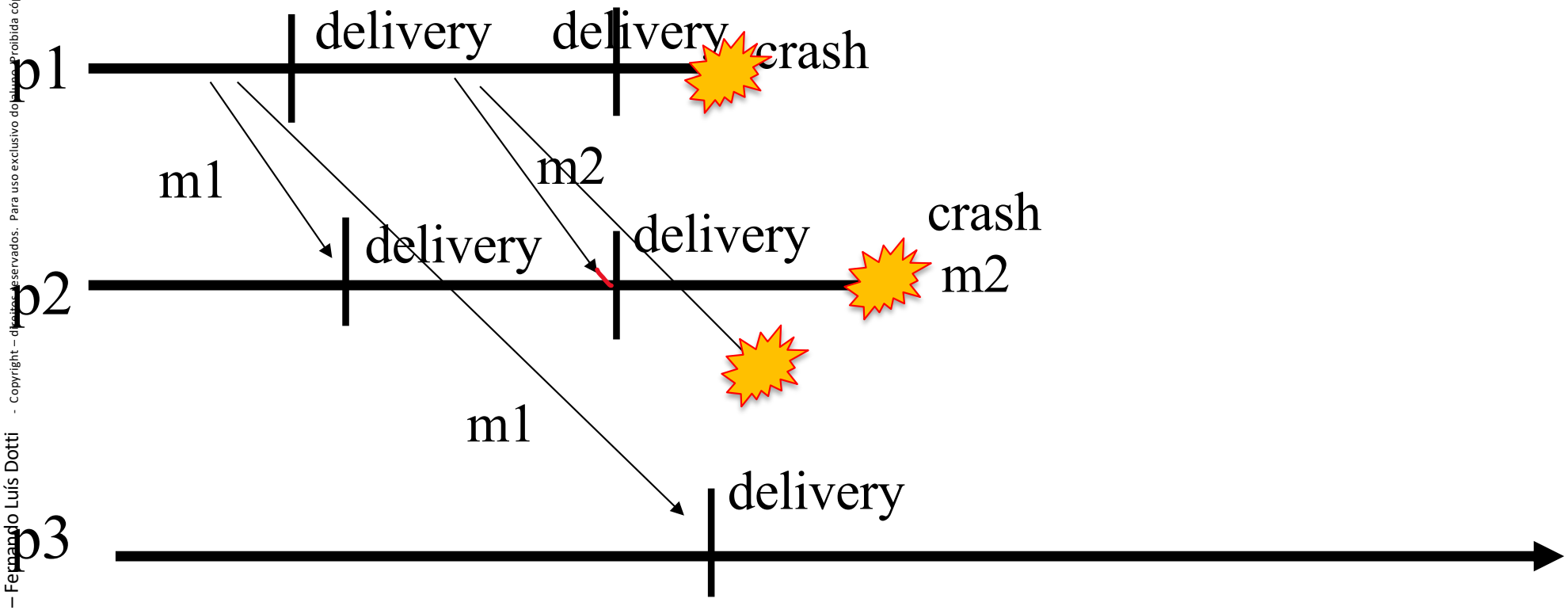
Material baseado no *livro e em slides* de: *Introduction to Reliable and Secure Distributed Programming Christian Cachin, Rachid Gerraoui, Luís Rodrigues*

Reliable broadcast



Material baseado no *livro e em slides* de: *Introduction to Reliable and Secure Distributed Programming Christian Cachin, Rachid Gerraoui, Luís Rodrigues*

Regular Reliable broadcast



Material baseado no *livro e em slides* de: *Introduction to Reliable and Secure Distributed Programming Christian Cachin, Rachid Gerraoui, Luís Rodrigues*

Reliable broadcast (rb)

- Processo que rb-deliver uma mensagem e posteriormente falha pode deixar aplicação em estado inconsistente
- Considere que a mensagem entregue significa alguma ação com efeito externo ao sistema
  - persistência de dados
  - atuação em uma infraestrutura física
- Os demais processos vivos não estao sincronizados com esta ação!!!

Uniform broadcast (urb)

Module 3.3: Interface and properties of uniform reliable broadcast

Module:

Name: UniformReliableBroadcast, **instance** *urb*.

Events:

Request: $\langle$ *urb*, *Broadcast* | m $\rangle$: Broadcasts a message m to all processes.

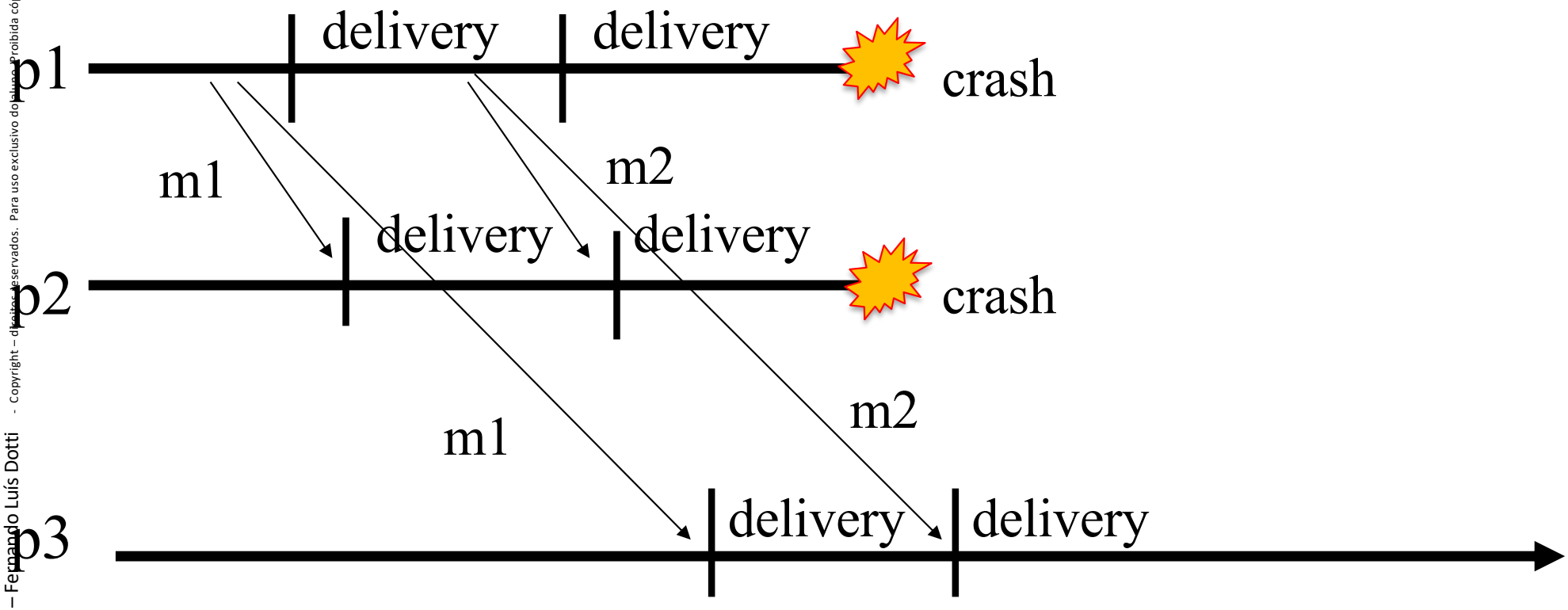
Indication: $\langle$ *urb*, *Deliver* | p, m $\rangle$: Delivers a message m broadcast by process p.

Properties:

URB1–URB3: Same as properties RB1–RB3 in (regular) reliable broadcast (Module 3.2).

URB4: *Uniform agreement:* If a message m is delivered by some process (whether correct or faulty), then m is eventually delivered by every correct process.

Uniform reliable broadcast



Material baseado no *livro e em slides* de: *Introduction to Reliable and Secure Distributed Programming Christian Cachin, Rachid Gerraoui, Luís Rodrigues*

Algorithm (urb)

Algorithm 3.4: All-Ack Uniform Reliable Broadcast

Implements:
UniformReliableBroadcast, **instance** *urb*.

Uses:
BestEffortBroadcast, **instance** *beb*.
PerfectFailureDetector, **instance** $\mathcal{P}$.

upon event $\langle$ *urb*, *Init* $\rangle$ **do**
delivered := $\emptyset$;
pending := $\emptyset$;
correct := Π;
forall m **do** *ack*$[m]$:= $\emptyset$;

upon event $\langle$ *urb*, *Broadcast* | m $\rangle$ **do**
pending := *pending* $\cup$ $\{(self, m)\}$;
trigger $\langle$ *beb*, *Broadcast* | [DATA, *self*, m] $\rangle$;

upon event $\langle$ *beb*, *Deliver* | p, [DATA, s, m] $\rangle$ **do**
ack$[m]$:= *ack*$[m]$ $\cup$ $\{p\}$;
if $(s, m) \notin$ *pending* **then**
pending := *pending* $\cup$ $\{(s, m)\}$;
trigger $\langle$ *beb*, *Broadcast* | [DATA, s, m] $\rangle$;

upon event $\langle$ $\mathcal{P}$, *Crash* | p $\rangle$ **do**
correct := *correct* $\setminus$ $\{p\}$;

function *candeliver*(m) **returns** Boolean **is**
return $(correct \subseteq ack[m])$;

upon exists $(s, m) \in$ *pending* such that *candeliver*$(m) \wedge m \notin$ *delivered* **do**
delivered := *delivered* $\cup$ $\{m\}$;
trigger $\langle$ *urb*, *Deliver* | s, m $\rangle$;

Fail-stop alg

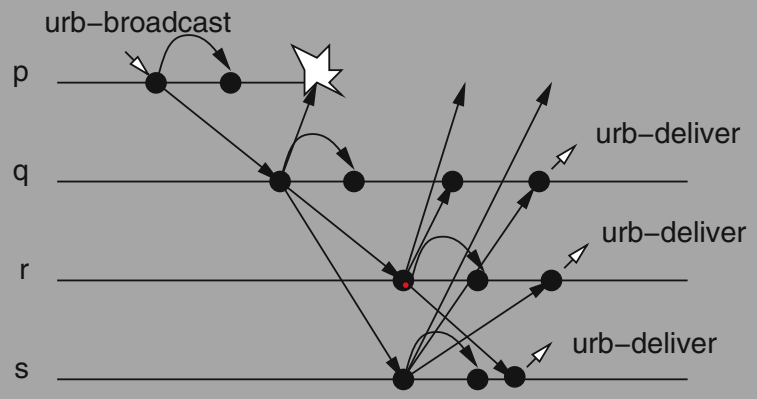


Figure 3.4: Sample execution of all-ack uniform reliable broadcast

Atenção para semântica das linhas

Ordenação

Material baseado no *livro e em slides* de: *Introduction to Reliable and Secure Distributed Programming Christian Cachin, Rachid Gerraoui, Luís Rodrigues*

FIFO Broadcast

Proprieadade de Entrega FIFO:
se um processo difunde m1 e depois m2, todo outro processo entrega m2 somente depois de entregar m1

Causal Order -Broadcast

Entrega Causalmente Ordenada:
para qualquer mensagem m1 que potencialmente causa m2, nenhum processo entrega m2 a não ser que entruegue m1

Existe relação de causa sempre que:

um processo envia m1 e depois m2

um processo recebe m1 e depois envia m2

por transitividade

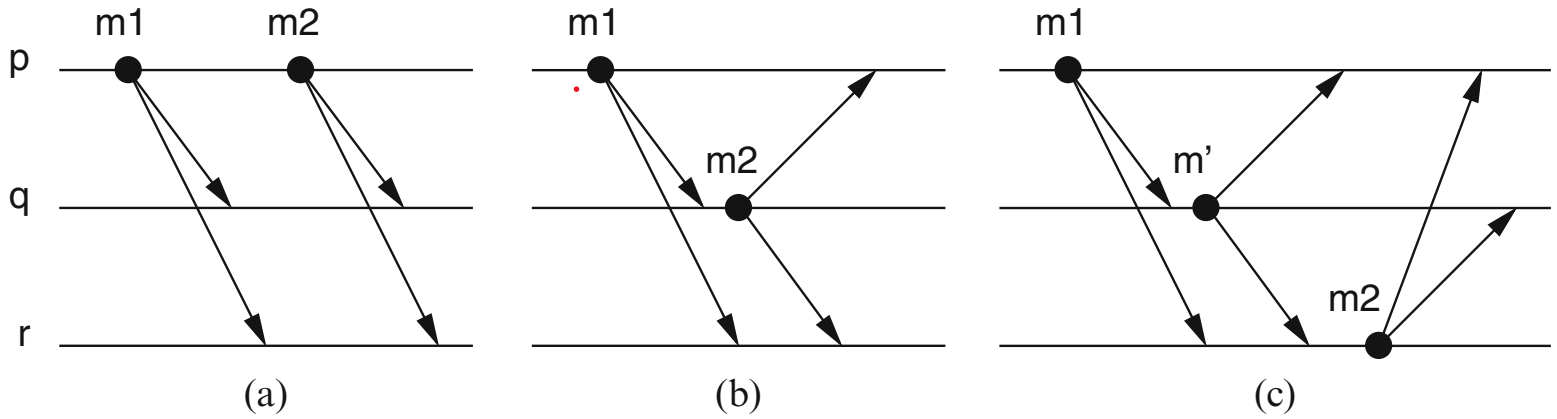
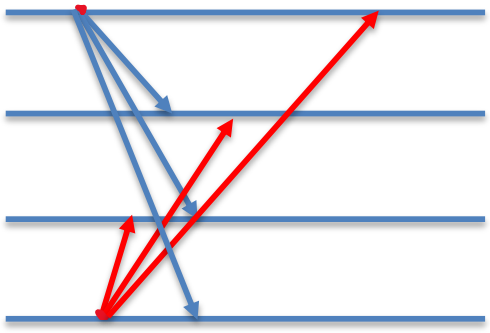


Figure 3.8: Causal order of messages

Material baseado no *livro e em slides* de: *Introduction to Reliable and Secure Distributed Programming Christian Cachin, Rachid Gerraoui, Luís Rodrigues*

Causal Order -Broadcast



Mensagens independentes podem ser entregues em qualquer ordem!

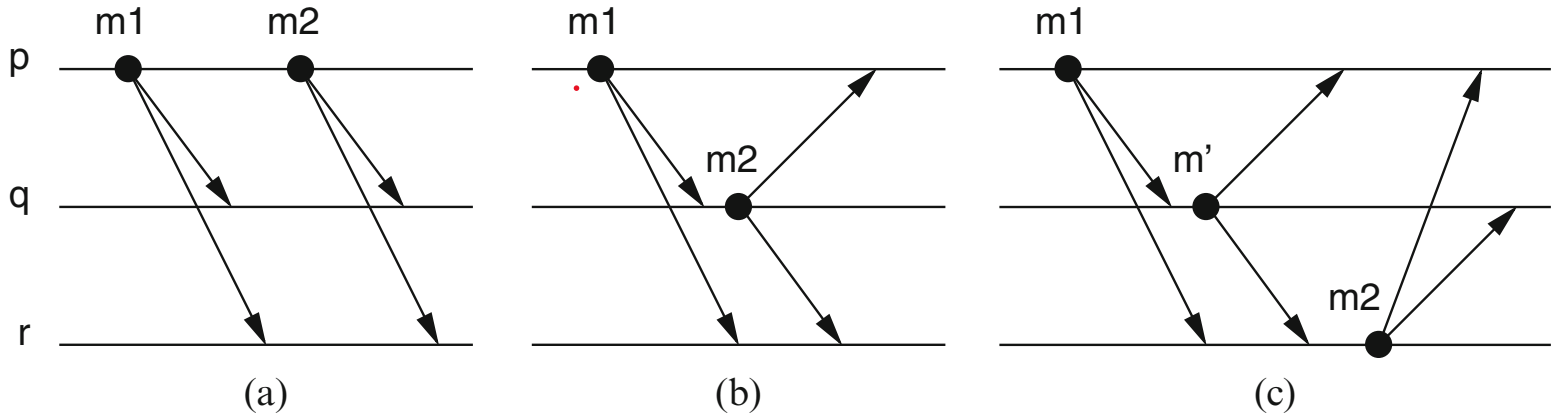


Figure 3.8: Causal order of messages

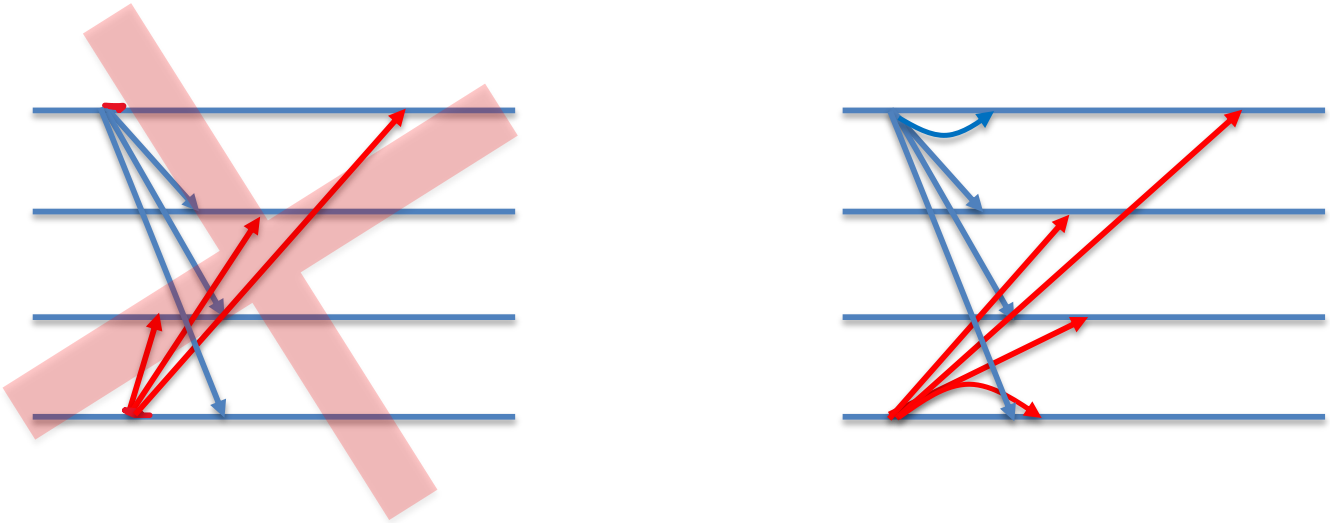
CO- Broadcast

- Causal order
  - Mensagens respeitam relação de causa e efeito, ou ordem causal
  - Ordem causal aplicada a mensagens trocadas entre processos
    - Expressão através de eventos de broadcast e deliver
  - Uma mensagem m1 (potencialmente) causa m2, denotado aqui m1-> m2 se qualquer de abaixo é verdade:
    - Um processo p broadcast m1 antes de (o mesmo p) broadcast m2
    - Algum processo p deliver m1 e subsequentemente broadcast m2
    - Existe alguma mensagem m' tal que m1 -> m' e m'->m2 (relação transitiva)

Total Order - Broadcast

- Total order
  - Todas mensagens são entregues na mesma ordem em seus destinatários
  - Não necessariamente a ordem do tempo absoluto do envio (pois isso não é implementável)
  - A ordem de entrega em todos processos é uma ordem "acordada" entre eles -> Consenso
  - Forma de maior nível de abstração e necessária em aplicações de alto nível de consistência, como dados replicados

Total Order - Broadcast



TO-Broadcast ou Atomic Broadcast

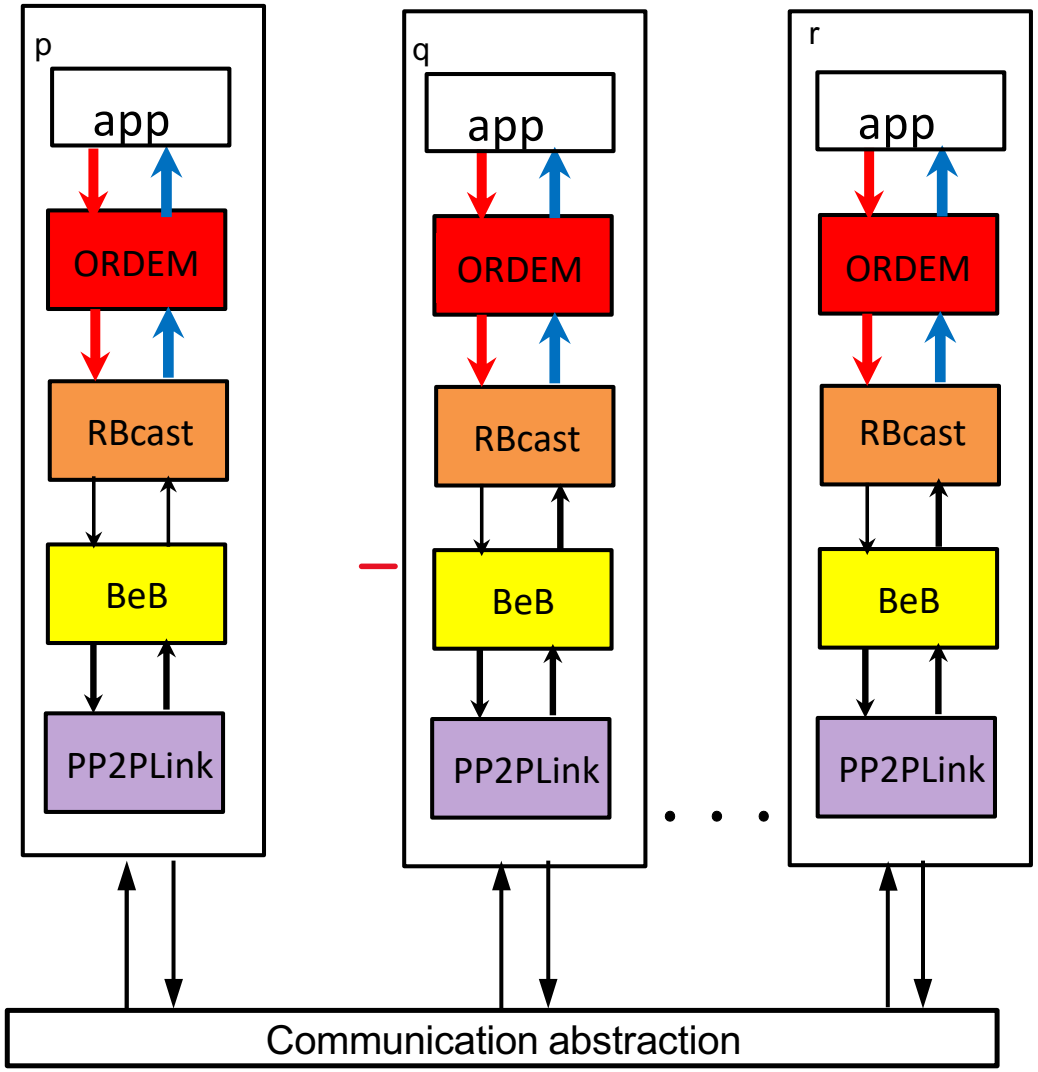
todo processo receptor processa todas mensagens na mesma ordem

Isto gera funcionalidades importantes

por exemplo, em bases de dados replicadas, seja em qual réplica for, a versão do dado obtida é a mesma

Ordenação – Implementação

- Assim como já explanado
  - em geral
    as ordens podem ser implementadas adicionando módulos responsáveis aos já existentes
  - ao lado, no módulo ORDEM, pode se implementar FIFO, CAUSAL, TOTAL,



Material baseado no *livro e em slides* de: *Introduction to Reliable and Secure Distributed Programming* *Christian Cachin, Rachid Gerraoui, Luís Rodrigues*

FIFO - Broadcast

- Propriedade FIFO
  - Mensagens do mesmo originador, na ordem de envio

Algorithm 3.12: Broadcast with Sequence Number

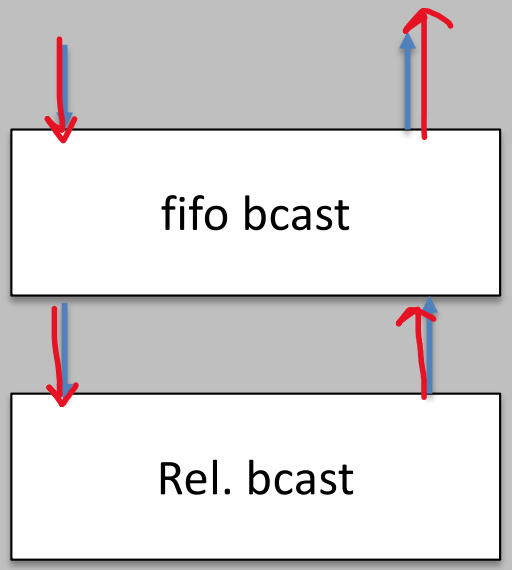
Implements:
FIFOReliableBroadcast, **instance** *frb*.

Uses:
ReliableBroadcast, **instance** *rb*.

upon event $\langle$ *frb*, *Init* $\rangle$ **do**
lsn := 0;
pending := $\emptyset$;
next := $[1]^N$;

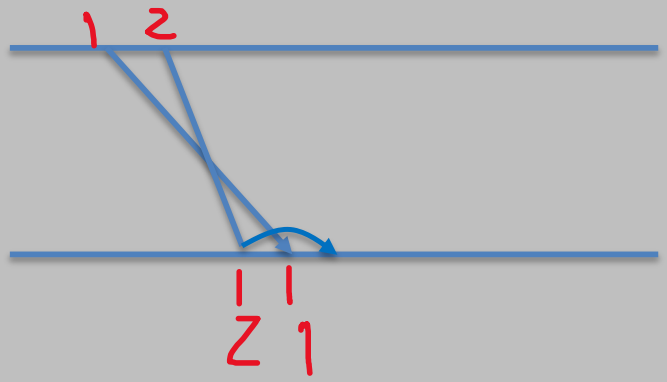
upon event $\langle$ *frb*, *Broadcast* | m $\rangle$ **do**
lsn := *lsn* + 1;
trigger $\langle$ *rb*, *Broadcast* | [DATA, *self*, m, *lsn*] $\rangle$;

upon event $\langle$ *rb*, *Deliver* | p, [DATA, s, m, sn] $\rangle$ **do**
pending := *pending* $\cup$ $\{(s, m, sn)\}$;
while exists $(s, m', sn') \in$ *pending* such that $sn' =$ *next*$[s]$ **do**
next$[s]$:= *next*$[s]$ + 1;
pending := *pending* $\setminus$ $\{(s, m', sn')\}$;
trigger $\langle$ *frb*, *Deliver* | s, m' $\rangle$;



ordem

confiabilidade



Material baseado no *livro e em slides* de: *Introduction to Reliable and Secure Distributed Programming* *Christian Cachin, Rachid Gerraoui, Luís Rodrigues*

CO- Broadcast

Algorithm 3.14: Garbage-Collection of Causal Past (extends Algorithm 3.13)

Implements:
CausalOrderReliableBroadcast, **instance** *crb*.

Uses:
ReliableBroadcast, **instance** *rb*;
PerfectFailureDetector, **instance** $\mathcal{P}$.

// Except for its ⟨ *Init* ⟩ event handler, the pseudo code of Algorithm 3.13 is also
// part of this algorithm.

upon event ⟨ *crb*, *Init* ⟩ **do**
delivered := ∅;
past := [];
correct := Π;
forall m **do** *ack*$[m]$:= ∅;

upon event ⟨ $\mathcal{P}$, *Crash* | p ⟩ **do**
correct := *correct* \ $\{p\}$;

upon exists $m \in$ *delivered* such that *self* $\notin$ *ack*$[m]$ **do**
ack$[m]$:= *ack*$[m] \cup \{$*self*$\}$;
trigger ⟨ *rb*, *Broadcast* | [ACK, m] ⟩;

upon event ⟨ *rb*, *Deliver* | p, [ACK, m] ⟩ **do**
ack$[m]$:= *ack*$[m] \cup \{p\}$;

upon *correct* $\subseteq$ *ack*$[m]$ **do**
forall $(s', m') \in$ *past* such that $m' = m$ **do**
remove$(past, (s', m))$;

CO- Broadcast

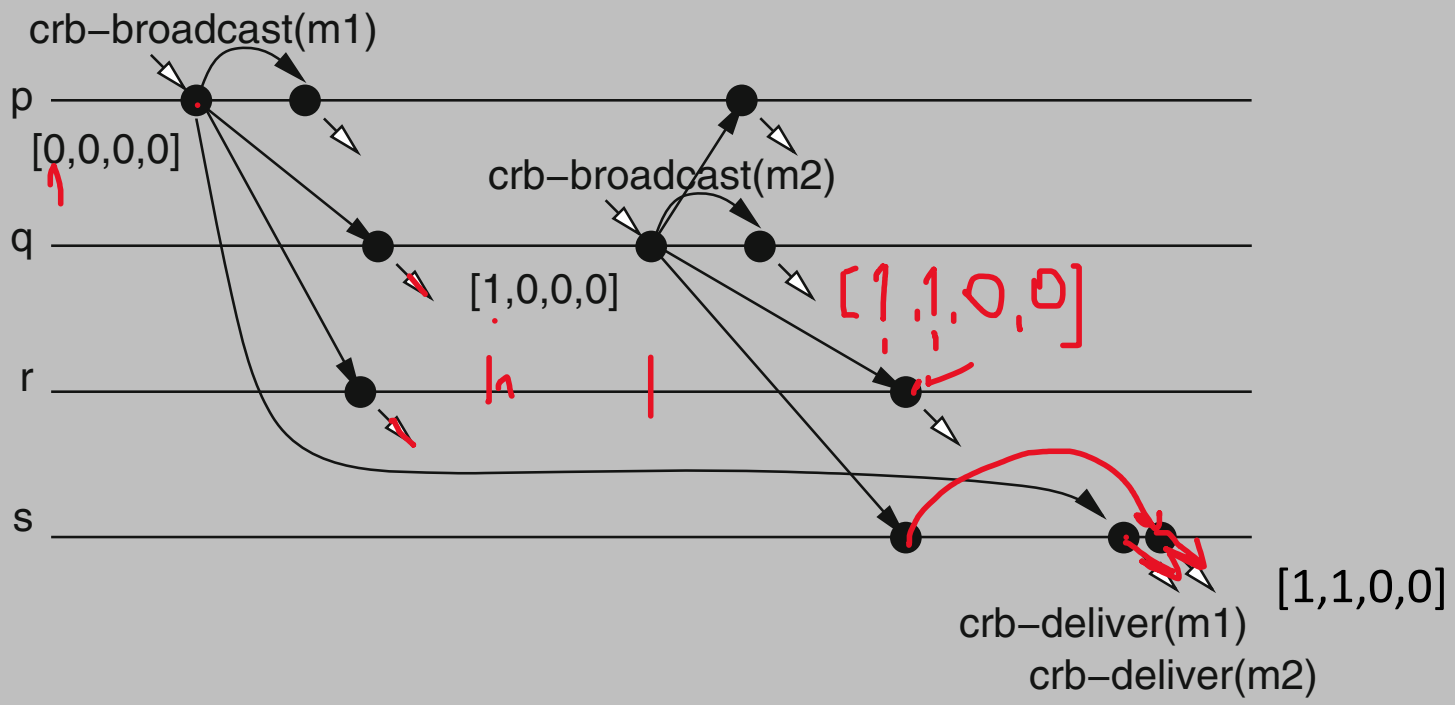


Figure 3.10: Sample execution of waiting causal broadcast

Material baseado no *livro e em slides* de: *Introduction to Reliable and Secure Distributed Programming Christian Cachin, Rachid Gerraoui, Luís Rodrigues*

Difusão Probabilística ou Epidêmica

Material baseado no *livro e em slides* de: *Introduction to Reliable and Secure Distributed Programming Christian Cachin, Rachid Gerraoui, Luís Rodrigues*

Difusão Probabilística

- voltadas a aplicações com alto número de processos – mas que possam conviver com não determinismo na entrega
- considerando grupos de processos com *milhares ou milhões de nodos*, a gerência da comunicação com os protocolos anteriores não escala
- protocolos determinísticos, com difusão e acks, sofrem do problema da *implosão de acks*

Difusão Probabilística

- algoritmos randomizados
  - comportamento parcialmente determinado por experimento randômico controlado
  - não provê garantias *determinísticas*, mas *probabilísticas*
  - para aplicações que não precisam de garantias determiníticas ("full reliability")
  - confiabilidade total é muito custosa (nro msgs), especialmente com muitos nodos

Difusão Probabilística

- pode-se construir sistemas que *escalam* com número de nodos

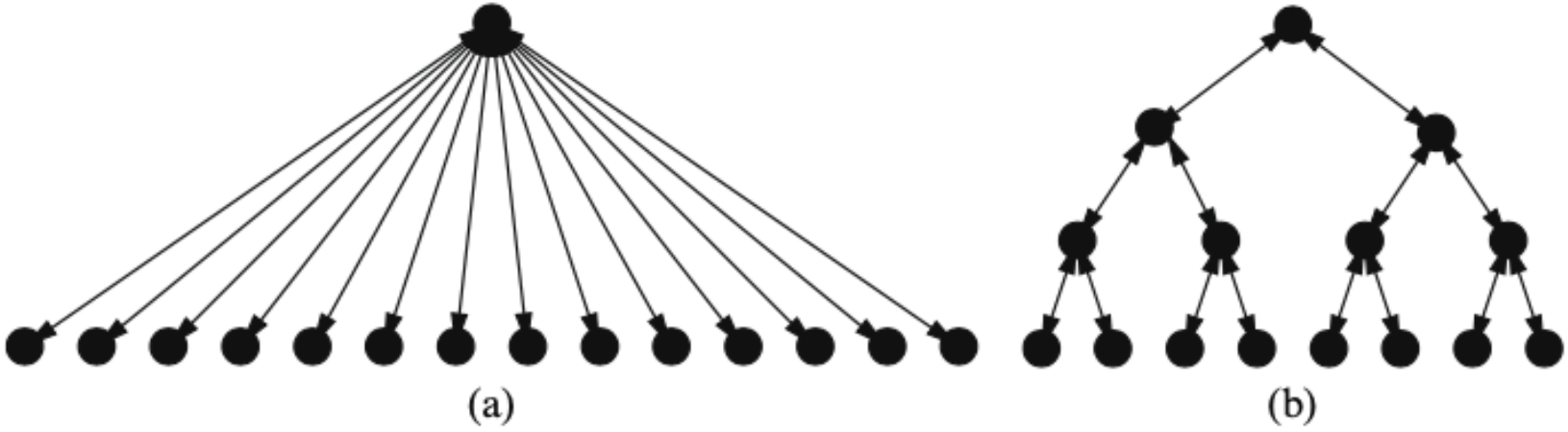


Figure 3.5: Direct vs. hierarchical communication for sending messages and receiving acknowledgments

Disseminação Epidêmica

- inspirado em como epidemias se disseminam em uma população
  - inicialmente algum indivíduo *infectado*
  - cada indivíduo infectado irá infectar *alguns* outros
  - *após um período*, toda população está infectada
- diversos algoritmos
  - também chamados: emidêmicos, *rumor mongering* (espalhamento de rumor), *gossip* (fofoca), *ou probabilistic broadcast* (difusão probabilística)

Difusão Probabilística

Module 3.7: Interface and properties of probabilistic broadcast

Module:

Name: ProbabilisticBroadcast, **instance** *pb*.

Events:

Request: ⟨ *pb*, *Broadcast* | m ⟩: Broadcasts a message m to all processes.

Indication: ⟨ *pb*, *Deliver* | p, m ⟩: Delivers a message m broadcast by process p.

Properties:

PB1: *Probabilistic validity:* There is a positive value ε such that when a correct process broadcasts a message m, the probability that every correct process eventually delivers m is at least $1 - \varepsilon$.

PB2: *No duplication:* No message is delivered more than once.

PB3: *No creation:* If a process delivers a message m with sender s, then m was previously broadcast by process s.

Material baseado no *livro e em slides* de: *Introduction to Reliable and Secure Distributed Programming* *Christian Cachin, Rachid Gerraoui, Luís Rodrigues*

Eager Probabilistic Broadcast

- processo seleciona k outros processos aleatoriamente e manda a mensagem
- cada um destes repete o comportamento
- k chamado *fanout* (fan : ventilador)
- cada passo de recebimento e reenvio da mensagem é chamado *round of gossiping* (rodada de fofofca)
- o algoritmo faz até R rodadas para cada mensagem (profundidade)

Eager Probabilistic Broadcast

- k chamado *fanout* (fan : ventilador)
  - escolha de k impacta desempenho
  - impacta na propriedade de validade probabilistica

PB1: *Probabilistic validity:* There is a positive value ε such that when a correct process broadcasts a message m, the probability that every correct process eventually delivers m is at least $1 - \varepsilon$.

  - k alto: aumenta a probabilidade de atingir toda população e diminui número de rounds necessários
  - custo: aumento de informação redundante na rede

Eager Probabilistic Broadcast

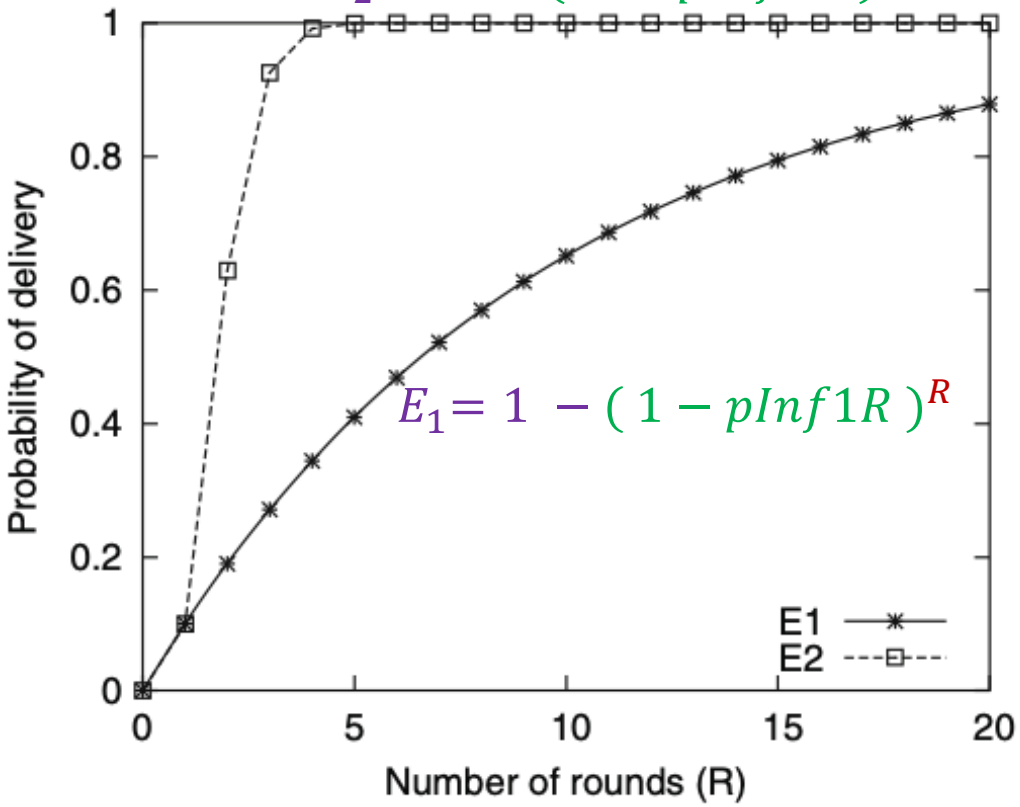
- entrega
  - pode existir algum processo nunca escolhido para ser infectado pelos demais
  - escolha de k e R podem reduzir esta probabilidade, mas nunca zerada

Difusão Probabilística

- exemplo com N=100, R = 20, k=10, f = 25

$E_2 = 1 - (1 - pInf1R)^{\sum_{r=0}^{R-1} I_r}$

$I_R = I_{R-1} + ((N - I_{R-1}) \times d \times pInf1R)$



probabilidade de entrega em um processo correto,

conforme estimativas E1 e E2, com Eager Probabilistic Broadcast

Material baseado no *livro e em slides* de: *Introduction to Reliable and Secure Distributed Programming* *Christian Cachin, Rachid Gerraoui, Luís Rodrigues*

Eager Probabilistic Broadcast

Algorithm 3.9: Eager Probabilistic Broadcast

Implements:
ProbabilisticBroadcast, **instance** *pb*.

Uses:
FairLossPointToPointLinks, **instance** *fll*.

upon event $\langle\ pb, Init\ \rangle$ **do**
$delivered := \emptyset$;

procedure *gossip*(msg) **is**
forall $t \in picktargets(k)$ **do trigger** $\langle\ fll, Send \mid t, msg\ \rangle$;

upon event $\langle\ pb, Broadcast \mid m\ \rangle$ **do**
$delivered := delivered \cup \{m\}$;
trigger $\langle\ pb, Deliver \mid self, m\ \rangle$;
gossip([GOSSIP, *self*, m, R]);

upon event $\langle\ fll, Deliver \mid p, [\text{GOSSIP}, s, m, r]\ \rangle$ **do**
if $m \notin delivered$ **then**
$delivered := delivered \cup \{m\}$;
trigger $\langle\ pb, Deliver \mid s, m\ \rangle$;
if $r > 1$ **then** *gossip*([GOSSIP, $s, m, r-1$]);

function *picktargets*(k) **returns** set of processes **is**
$targets := \emptyset$;
while $\#(targets) < k$ **do**
$candidate :=$ random$(\Pi \setminus \{self\})$;
if $candidate \notin targets$ **then**
$targets := targets \cup \{candidate\}$;
return *targets*;

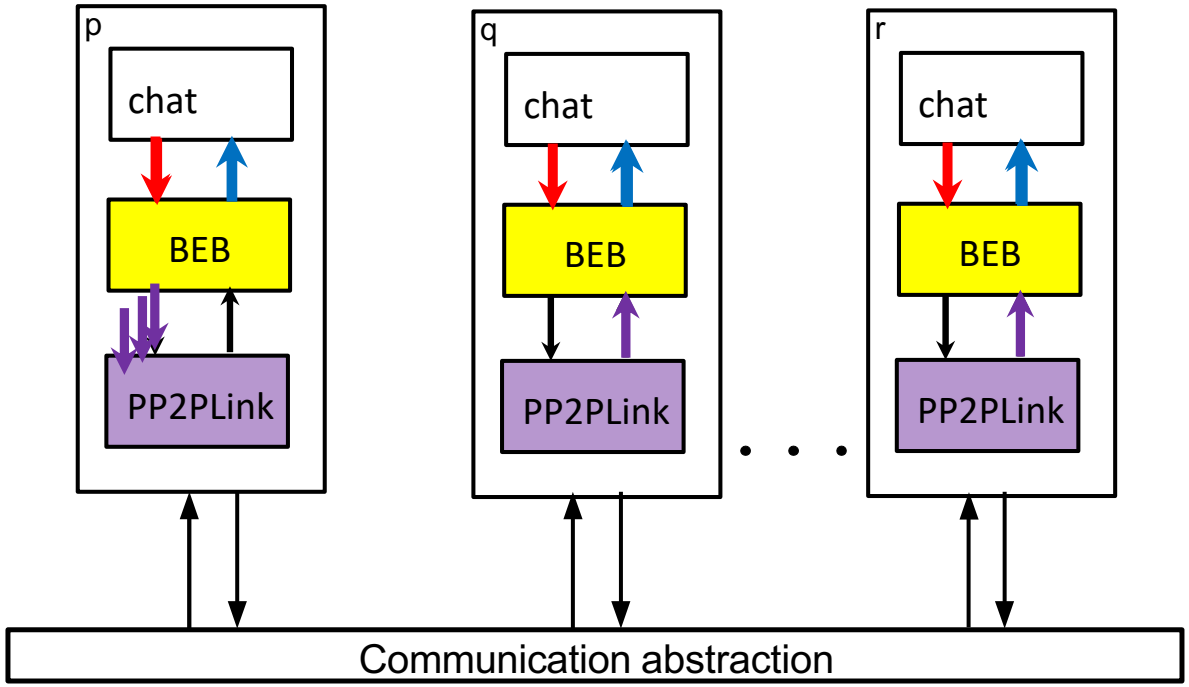
Material baseado no *livro e em slides* de: *Introduction to Reliable and Secure Distributed Programming Christian Cachin, Rachid Gerraoui, Luís Rodrigues*

Implementação

Implementação

veja módulos no moodle

Implementação



Material baseado no *livro e em slides* de: *Introduction to Reliable and Secure Distributed Programming Christian Cachin, Rachid Gerraoui, Luís Rodrigues*

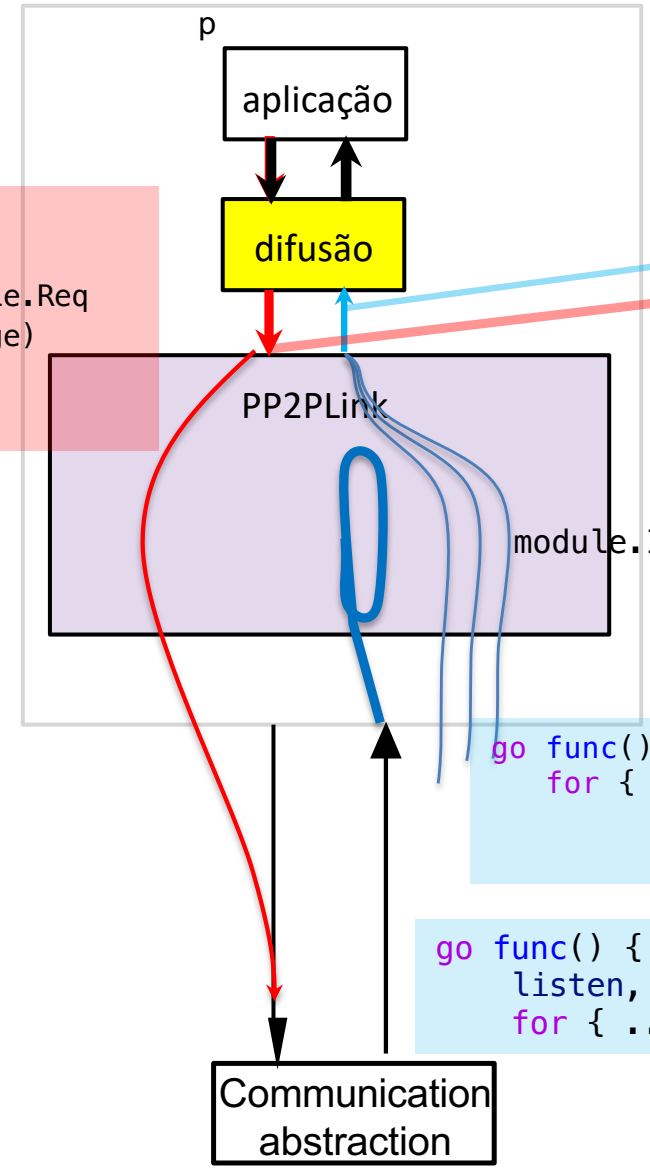
Implementação PP2PLink

```
type PP2PLink_Req_Message struct {
    To      string
    Message string
}

type PP2PLink_Ind_Message struct {
    From    string
    Message string
}

type PP2PLink struct {
    Ind   chan PP2PLink_Ind_Message
    Req   chan PP2PLink_Req_Message
    Run   bool
    dbg   bool
    Cache map[string]net.Conn // cache de conexo
}
```

```
go func() {
        for {
            message := <-module.Req
            module.Send(message)
        }
    }()
```



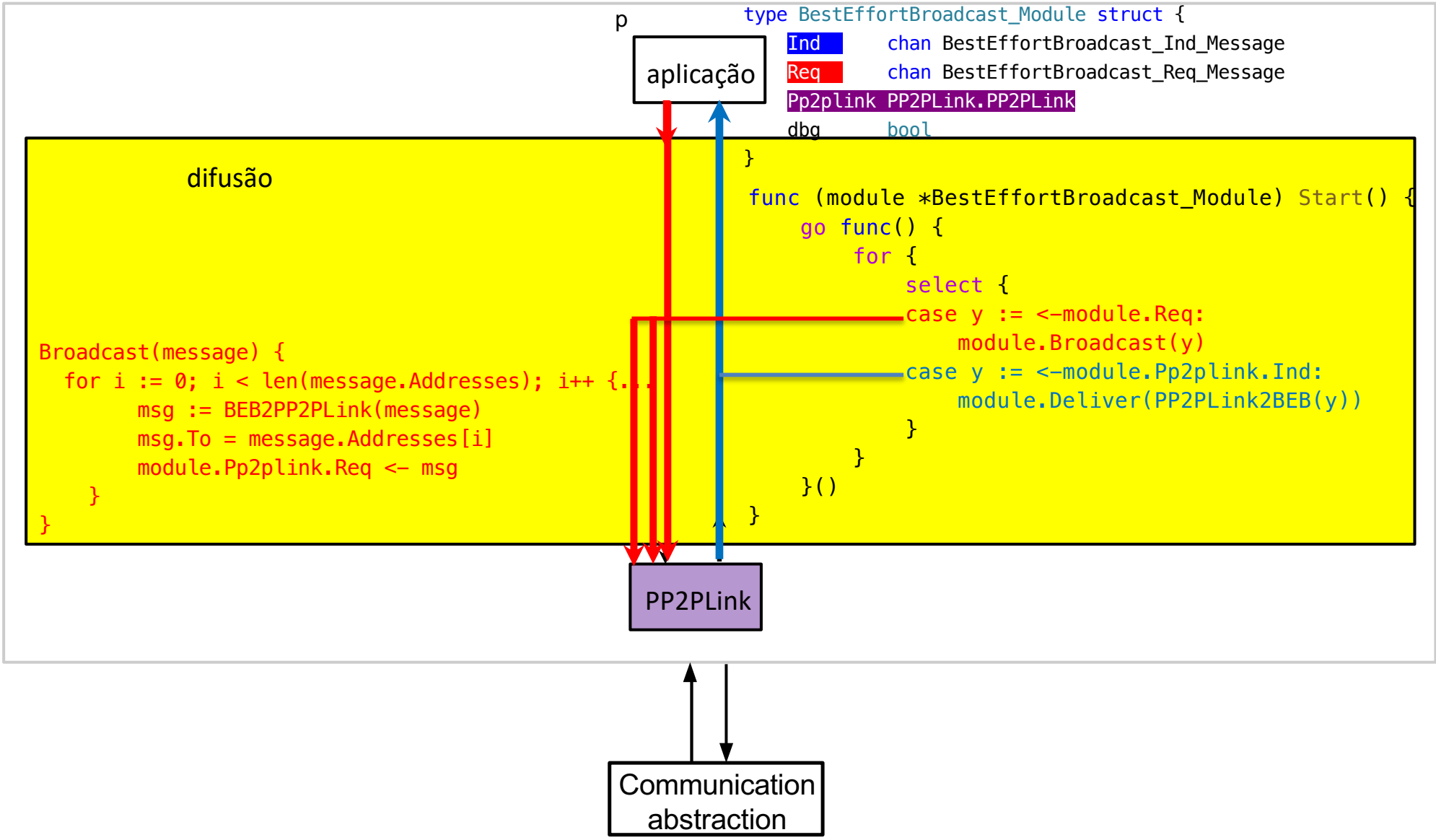

```
go func() {
    for { // // enquanto conexao aberta
            _, err := io.ReadFull(conn, bufTam)
            _, err = io.ReadFull(conn, bufMsg)
```

```
go func() {
    listen, _ := net.Listen("tcp4", address)
    for { ...
```


Material baseado no *livro e em slides* de: *Introduction to Reliable and Secure Distributed Programming Christian Cachin, Rachid Gerraoui, Luís Rodrigues*

Implementação BEB

```
type BestEffortBroadcast_Req_Message struct {
    Addresses []string
    Message   string}
type BestEffortBroadcast_Ind_Message struct {
    From    string
    Message string}
type BestEffortBroadcast_Module struct {
    Ind      chan BestEffortBroadcast_Ind_Message
    Req      chan BestEffortBroadcast_Req_Message
    Pp2plink PP2PLink.PP2PLink
    dbg      bool
}
```



Material baseado no *livro e em slides* de: *Introduction to Reliable and Secure Distributed Programming Christian Cachin, Rachid Gerraoui, Luís Rodrigues*

Material baseado no *livro e em slides* de: *Introduction to Reliable and Secure Distributed Programming* *Christian Cachin, Rachid Guerraoui, Luís Rodrigues*