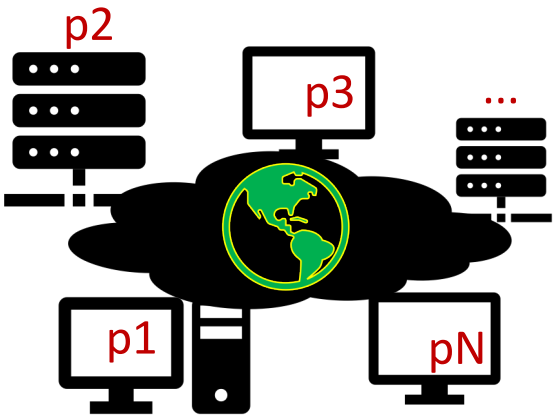
Fundamentos de Processamento Paralelo e Distribuído

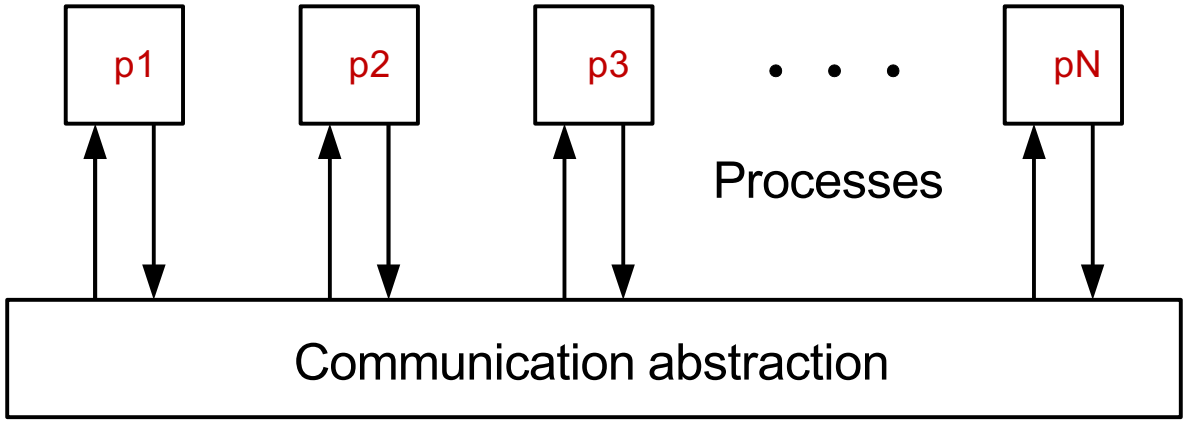
Algoritmo Distribuído de Exclusao Mútua

Fernando Luís Dotti – PUCRS

Material baseado no livro
Introduction to Reliable and Secure Distributed Programming
Christian Cachin, Rachid Gerraoui, Luís Rodrigues
e em material didático disponibilizado pelos autores

Abstrações para Programação Distribuída

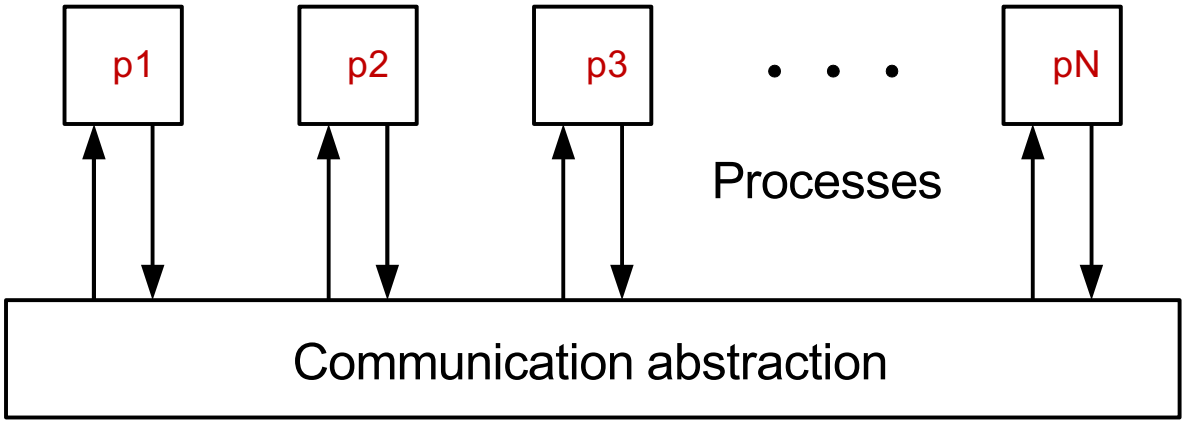




- Sistema com N processos $\Pi = \{p_1, \ldots p_N\}$
- Os processos se conhecem
- Se coordenam para uma tarefa comum

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

Exemplo: acesso a recurso comum

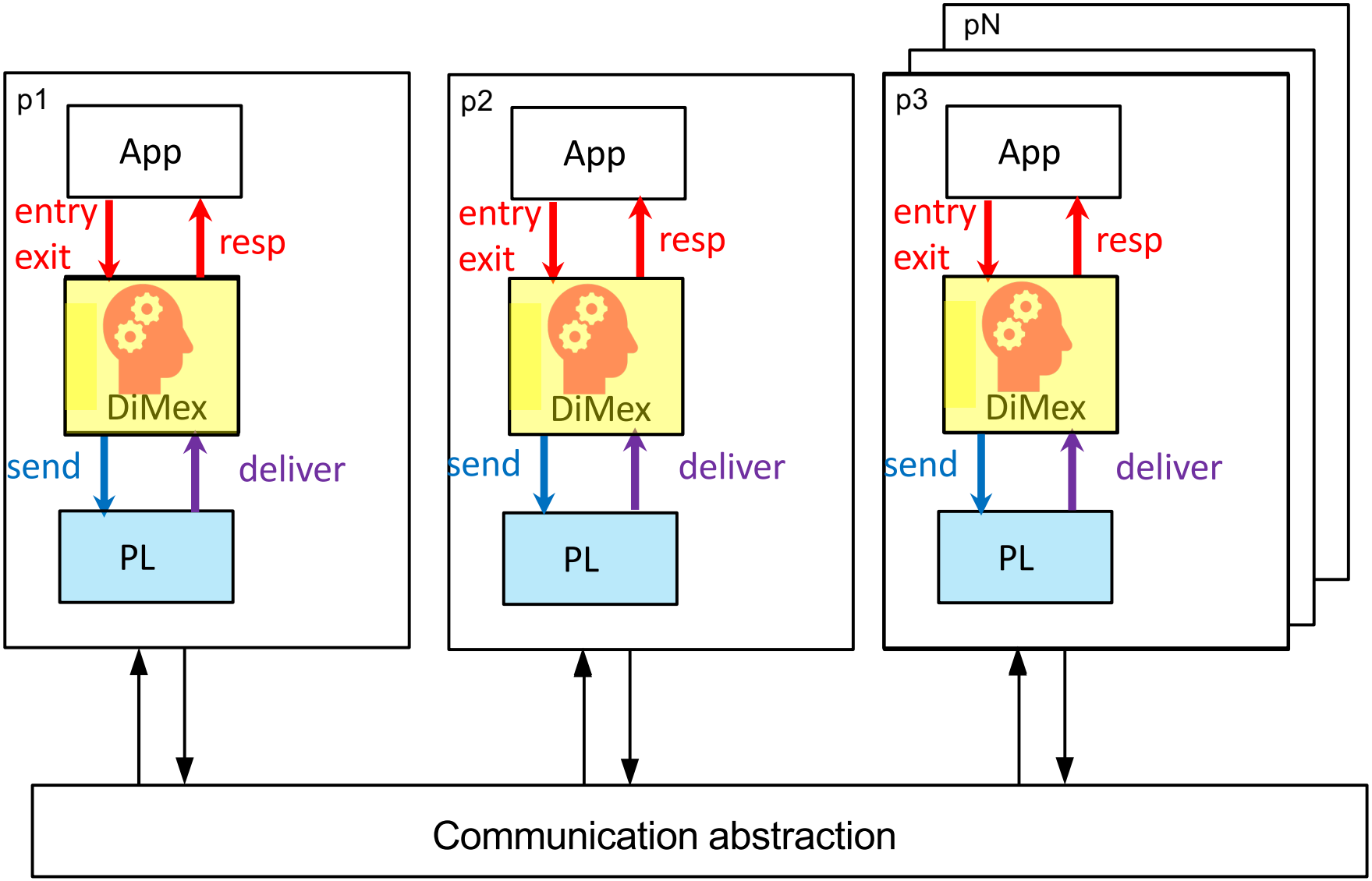


- Os processos tem acesso a um mesmo recurso (por exemplo um arquivo em um sistema de arquivos distribuído)
  e devem utilizá-lo de forma mutuamente exclusiva
- Entretanto a plataforma não provê este tipo de fucionalidade

Algoritmo de Ricart Agrawalla

- Vamos supor
  - links perfeitos ponto a ponto
  - processos não falham
- Algoritmo DiMex (distributed mutual exclusion)

Visão Geral -Exclusão Mútua Distribuída



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

Exclusão Mútua Distribuída - Propriedades

Modulo:

DistributedMutualExclusion, *instância* ***dmx.***

Eventos:

Request: [dmx, Entry | r]: solicita acesso a r

Request: [dmx, Exit | r]: avisa fim de acesso a r

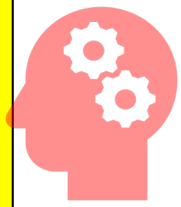
Indicação: [dmx, Resp | r]: permite acesso

Propriedades

DMX1: (não-postergação e não bloqueio) se um processo solicita Entry, decorrido um tempo ele entregará resp

DMX2: (mutex) Se um processo p entregou resp, nenhum outro processo entregará resp antes que p sinalize solicite Exit;

OBS.: Aqui assume-se que o módulo usuário da ExclMútua segue o protocolo: request Entry, aguarda resp, ao final request Exit



Exclusão Mútua Distribuída - Propriedades

Modulo:

DistributedMutualExclusion, *instância* ***dmx.***

Eventos:

Request: [dmx, Entry | r]: solicita acesso a r

Request: [dmx, Exit | r]: avisa fim de acesso a r

Indicação: [dmx, resp | r]: permite acesso

Propriedades

DMX1: (não-postergação e não bloqueio) se um processo solicita Entry, decorrido um tempo ele entregará resp

DMX2: (mutex) Se um processo p entregou resp, nenhum outro processo entregará resp antes que p sinalize solicite Exit;

Modulo:

PerfectPointToPointLinks*, instancia pl*

Eventos de interface (usados pelo módulo superior)**:**

Request: [pl, Send | q, m]: solicita envio de m para q

Indicação: [pl, Deliver | p, m]: entrega mensagem m enviada por p

Propriedades

PL1: Entrega Confiável: se um processo sends uma mensagem m para um processo q, então q entregara (delivers) a mensagem em algum momento;

PL2: Não Duplicação: uma mensagem não é entregue por um processo mais de uma vez

PL3: Não criação: uma mensagem não é entregue se não tiver sido enviada

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

Algoritmo de Ricart / Agrawalla

Communications
of
the ACM

January 1981
Volume 24
Number 1

Operating Systems

R. Stockton Gaines
Editor

An Optimal Algorithm for Mutual Exclusion in Computer Networks

Glenn Ricart
National Institutes of Health

Ashok K. Agrawala
University of Maryland

An algorithm is proposed that creates mutual exclusion in a computer network whose nodes communicate only by messages and do not share memory. The algorithm sends only $2*(N-1)$ messages, where N is the number of nodes in the network per critical section invocation. This number of messages is at a minimum if parallel, distributed, symmetric control is used; hence, the algorithm is optimal in this respect. The time needed to achieve mutual exclusion is also minimal under some general assumptions.

As in Lamport's "bakery algorithm," unbounded sequence numbers are used to provide first-come first-served priority into the critical section. It is shown that the number can be contained in a fixed amount of memory by storing it as the residue of a modulus. The number of messages required to implement the exclusion can be reduced by using sequential node-by-node processing, by using broadcast message techniques, or by sending information through timing channels. The "readers and writers" problem is solved by a simple modification of the algorithm and the modifications necessary to make the algorithm robust are described.

Key Words and Phrases: concurrent programming, critical section, distributed algorithm, mutual exclusion, network, synchronization

Material baseado no livro: *Introduction to Reliable and Secure Distributed Programm*

Algoritmo de Ricart Agrawalla

- ao receber req Entry da aplicação
  - processo pede a todos outros – msg REQUEST
- quando recebeu resposta - msg REPLY - de todos outros
  - deliver resp para aplicação
- ao receber req Exit da aplicação
  - manda REPLY a processos que pediram e estão esperando

reação à aplicação

- ao receber REQUEST de outro processo *p*
  - responde REPLY: se não quer SC ou se quer mas *p* pediu antes
  - posterga (p espera): se está na SC se quer e pediu antes de *p*

reação a outros processos

- *antes ou depois dado por timestamp lógico composto de relógio lógico e identificador do processo.*

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

Exclusão Mútua Distribuída algoritmo mono recurso

Algoritmo para DiMex, no processo p com id:

implementa ***DistributedMutualExclusion,*** *instância* ***dmx***
usa ***PerfectLinks*** instância ***pl;***

Vars

processes: conjunto de processos

pl: referência ao módulo de links perfeitos

upon event [dmx, Init] do

.

quando acontece evento [dmx, Entry] faça

aplicação solicita

...

...

quando acontece evento [dmx, Exit] faça

aplicação sinaliza saída fim

...

...

quando acontece evento de entrega de mensagem [pl, Deliver | q, [reqEntry, rid, rts]

pl entrega mensagem de q, tipo reqEntry com parametros rid,rts

...

.....

quando acontece evento de entrega de mensagem [pl, Deliver | q, [respOk]]

pl entrega mensagem de q, tipo reqEntry

...

...

Exclusão Mútua Distribuída algoritmo mono recurso

Algoritmo para DiMex, no processo p com id:

implementa ***DistributedMutualExclusion,*** *instância* ***dmx***
usa ***PerfectLinks*** instância ***pl;***

Vars

processes: conjunto de processos

pl: referência ao módulo de links perfeitos

upon event [dmx, Init] do

.

quando acontece evento [dmx, Entry] faça

aplicação solicita

…

gera evento [pl , Send | q, [reqEntry, id, reqTs]

usando pl, Send para processo q msg tipo reqEntry com id,reqTs

…

quando acontece evento [dmx, Exit] faça

aplicação sinaliza fim

…

gera evento [pl, Send | q , [respOk]]

usando pl, Send para processo q msg tipo respOk

…

quando acontece evento de entrega de mensagem [pl, Deliver | q, [reqEntry, rid, rts]

pl entrega mensagem de q, tipo reqEntry com parametros rid,rts

…

gera evento [pl, Send | q , [respOk]]

usando pl, Send para processo q msg tipo respOk

…..

quando acontece evento de entrega de mensagem [pl, Deliver | q, [respOk]]

pl entrega mensagem de q, tipo reqEntry

…

gera evento [dmx, Deliver | resp]

sinaliza que aplicação pode entrar

…

Exclusão Mútua Distribuída algoritmo mono recurso

Algoritmo para DiMex, no processo p com id:

implementa ***DistributedMutualExclusion,*** *instância* ***dmx***
usa ***PerfectLinks*** instância ***pl;***

Vars

processes: conjunto de processos

pl: referência ao módulo de links perfeitos

upon event [dmx, Init] do

.

quando acontece evento [dmx, Entry] faça

aplicação solicita

…

gera evento [pl , Send | q, [reqEntry, id, reqTs]

usando pl, Send para processo q msg tipo reqEntry com id,reqTs

…

quando acontece evento [dmx, Exit] faça

aplicação sinaliza fim

…

gera evento [pl, Send | q , [respOk]]

usando pl, Send para processo q msg tipo respOk

…

quando acontece evento de entrega de mensagem [pl, Deliver | q, [reqEntry, rid, rts]

pl entrega mensagem de q, tipo reqEntry com parametros rid,rts

…

gera evento [pl, Send | q , [respOk]]

usando pl, Send para processo q msg tipo respOk

…..

quando acontece evento de entrega de mensagem [pl, Deliver | q, [respOk]]

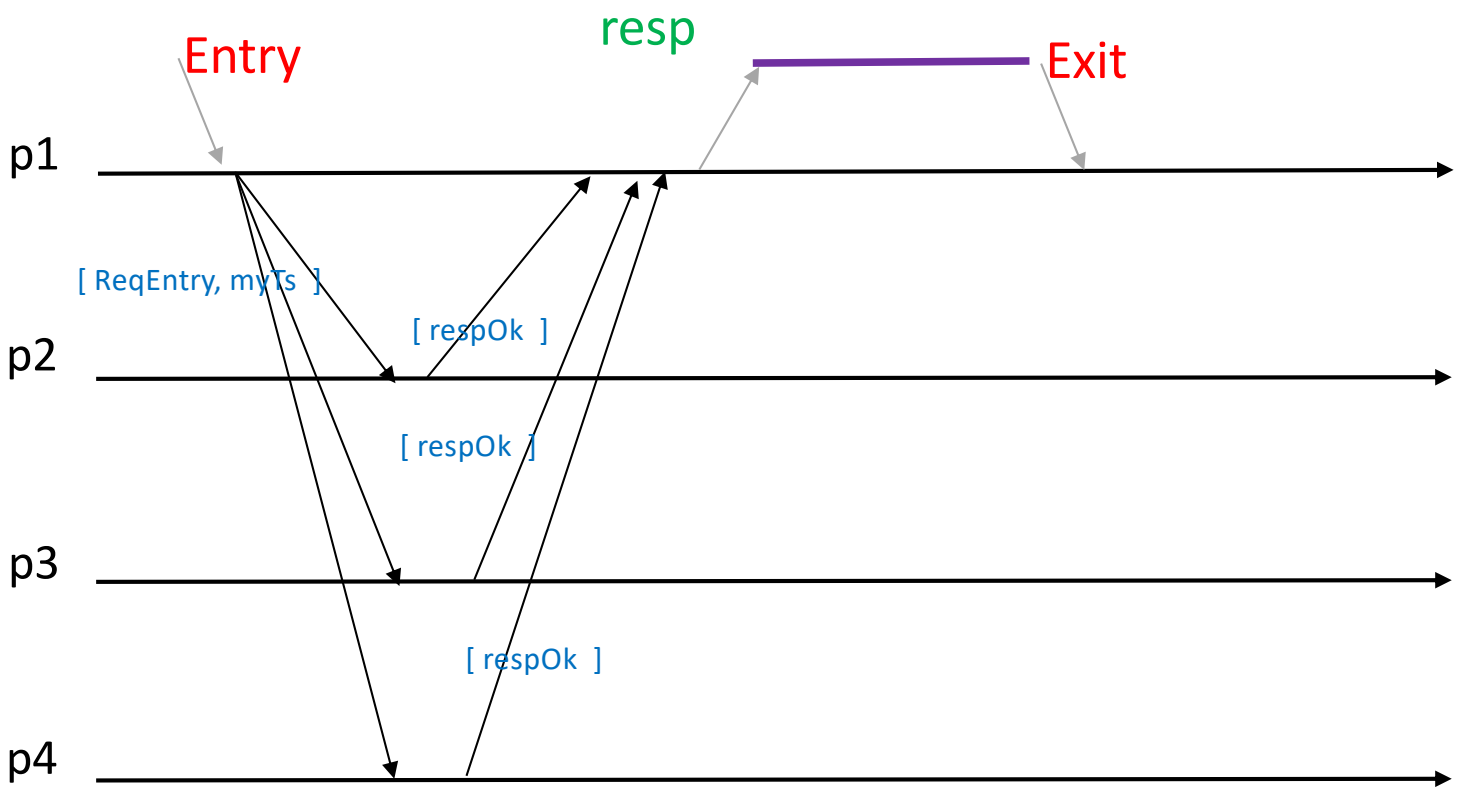
pl entrega mensagem de q, tipo reqEntry

…

gera evento [dmx, Deliver | resp]

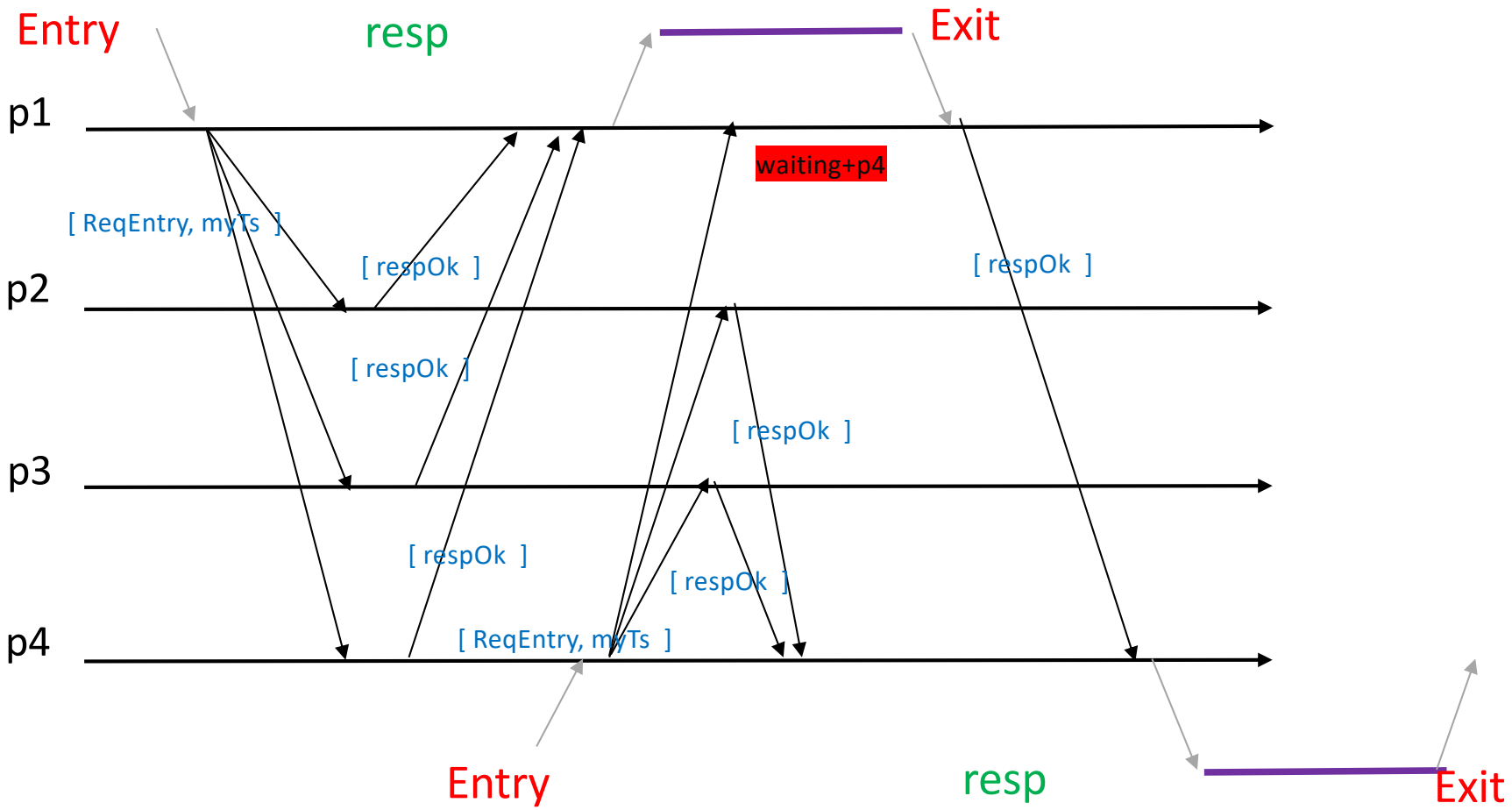
sinaliza que aplicação pode entrar

Algoritmo de Ricart Agrawalla



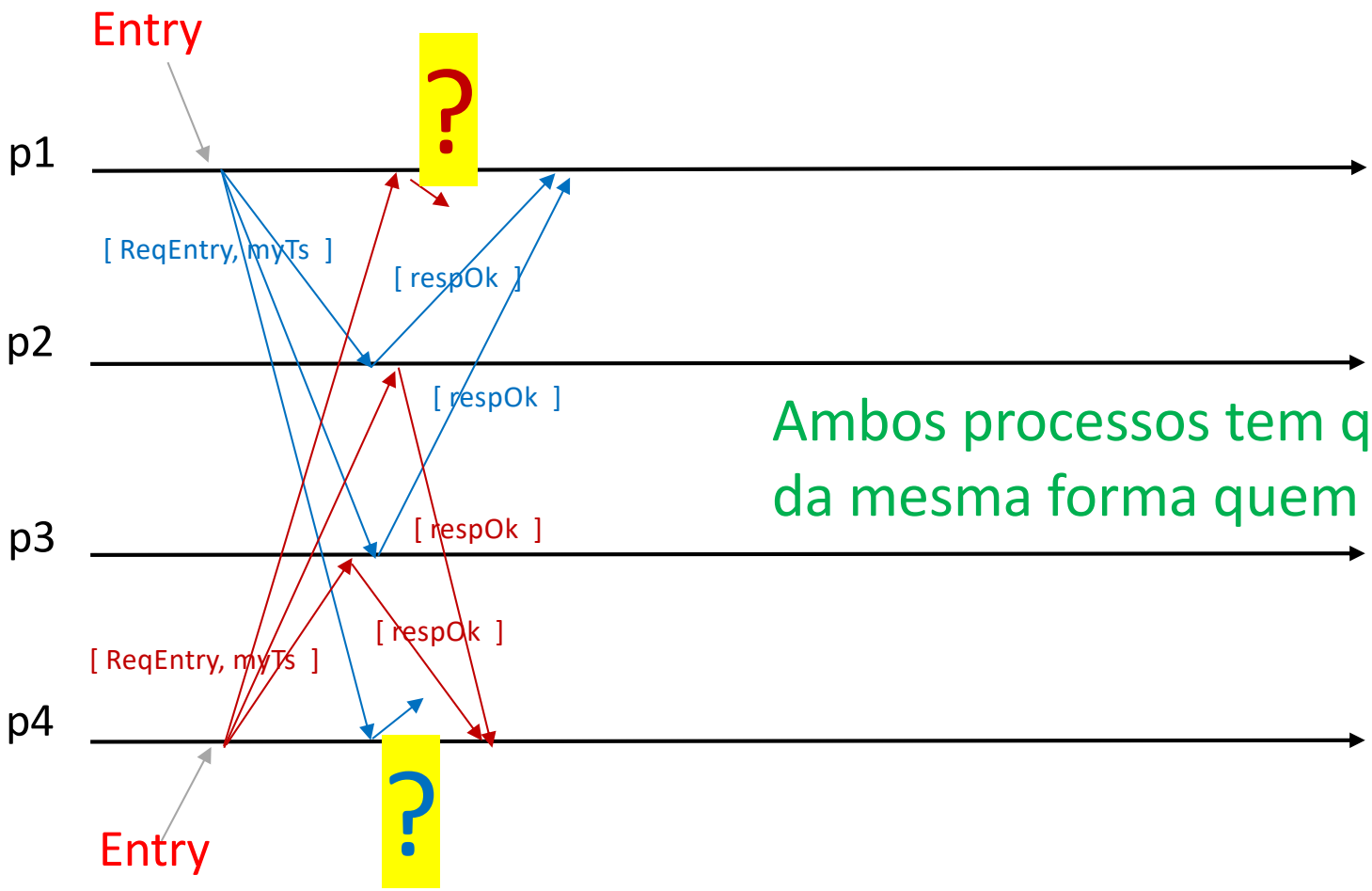
Material baseado no livro: *Introduction to Reliable and Secure Distributed Programming* *Christian Cachin, Rachid Gerraoui, Luís Rodrigues*

Algoritmo de Ricart Agrawalla



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

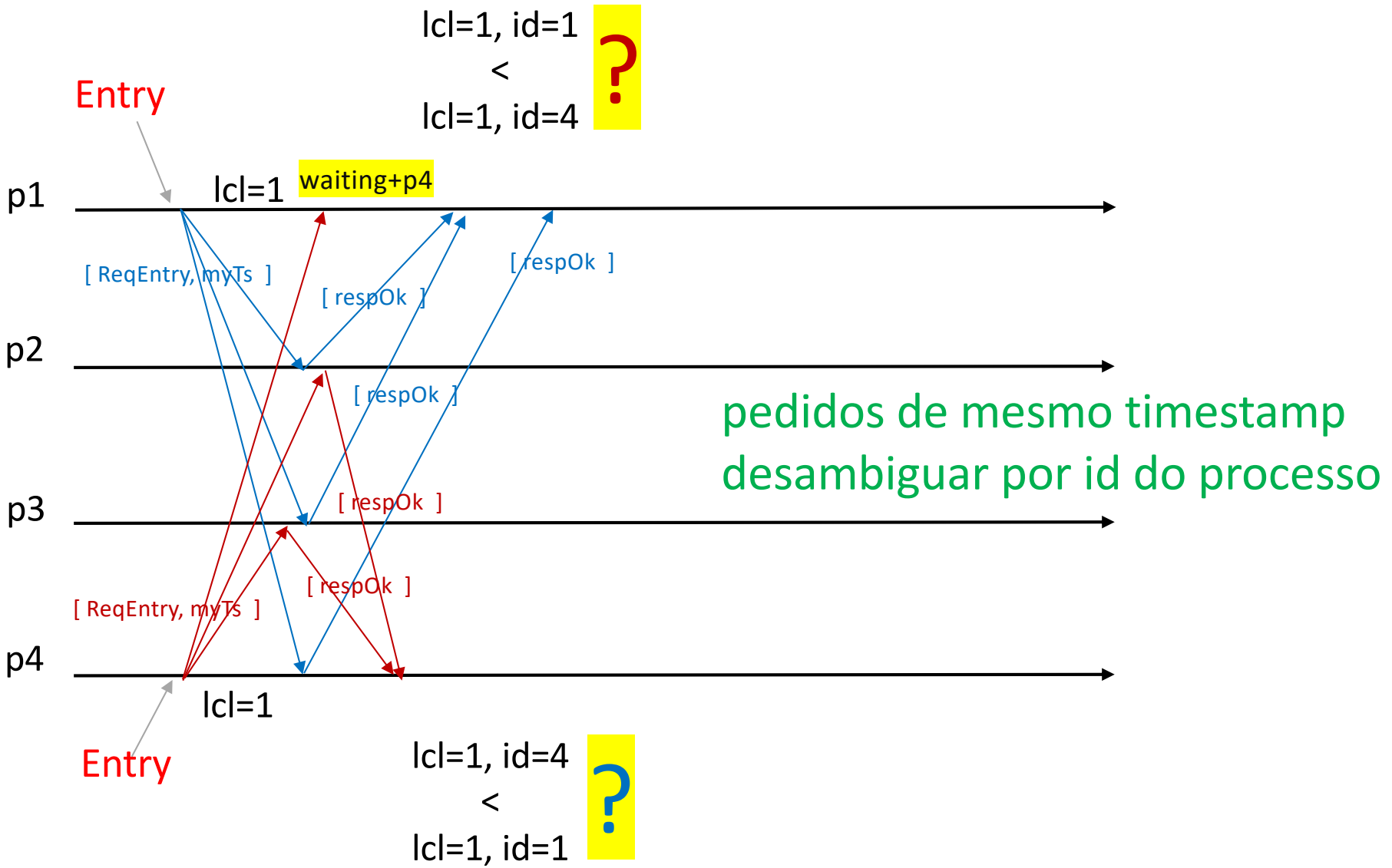
Algoritmo de Ricart Agrawalla



Ambos processos tem que decidir da mesma forma quem tem direito

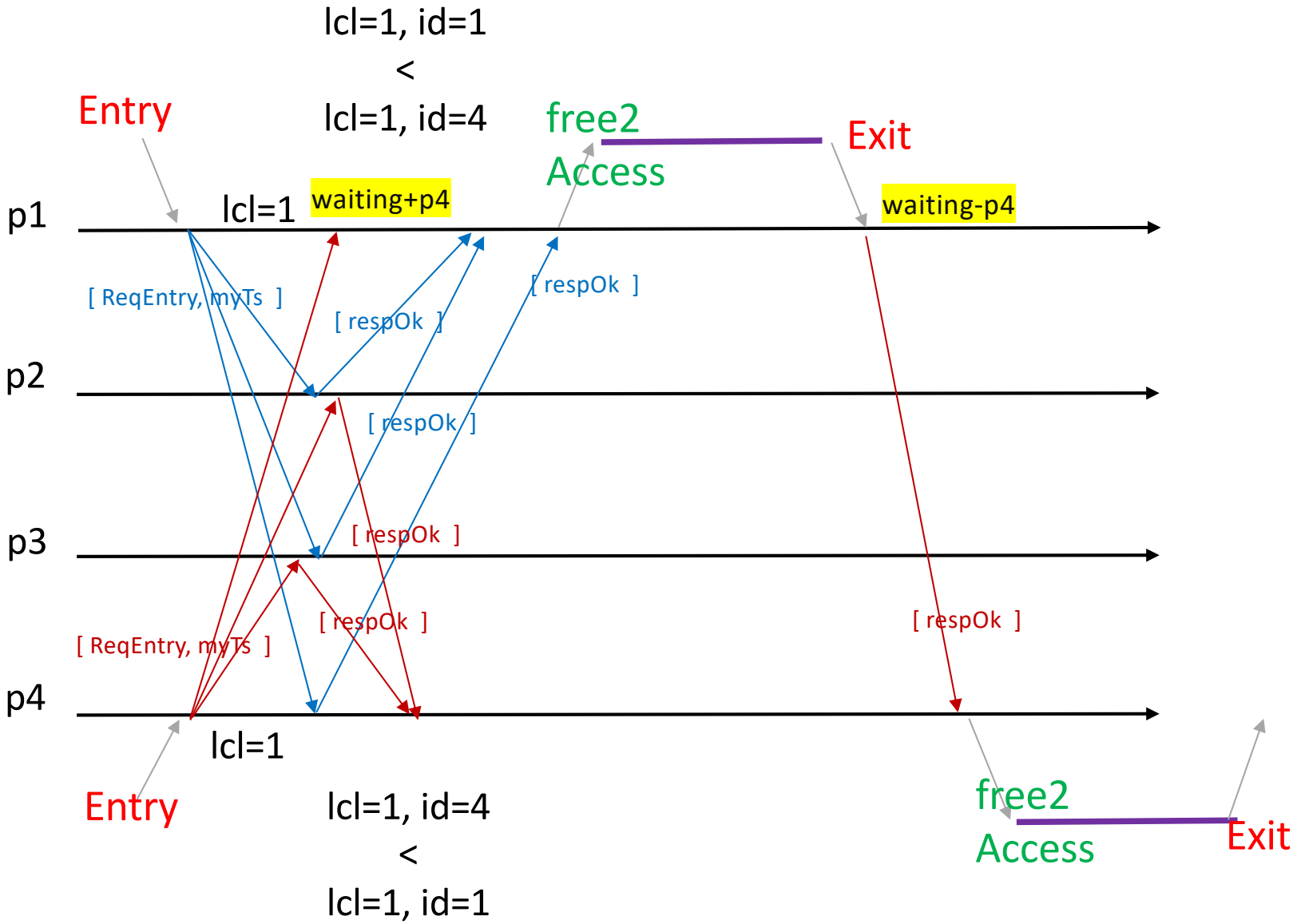
Material baseado no livro: *Introduction to Reliable and Secure Distributed Programming* *Christian Cachin, Rachid Gerraoui, Luís Rodrigues*

Algoritmo de Ricart Agrawalla



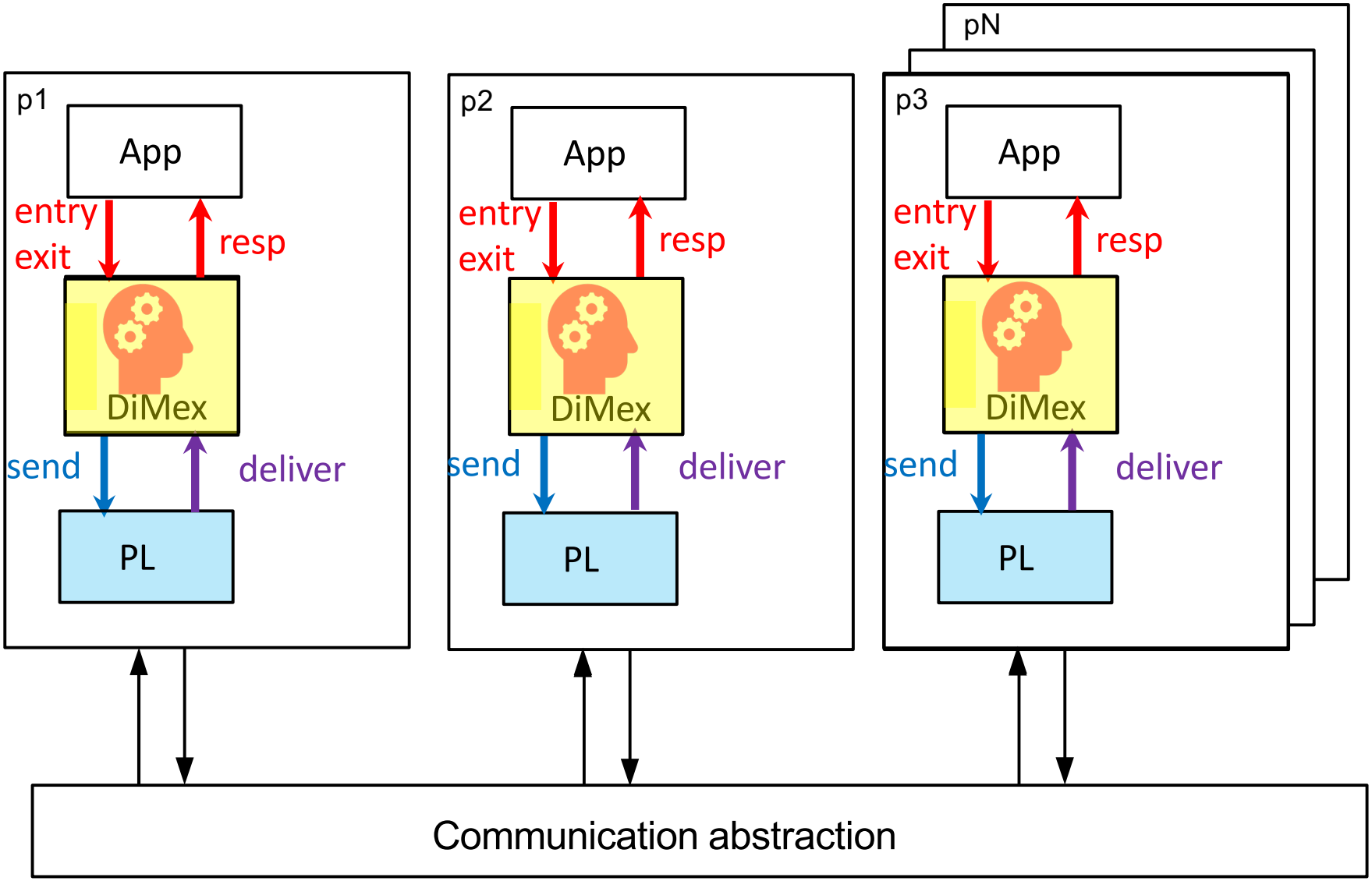
Material baseado no livro: *Introduction to Reliable and Secure Distributed Programming* *Christian Cachin, Rachid Gerraoui, Luís Rodrigues*

Algoritmo de Ricart Agrawalla



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

Visão Geral -Exclusão Mútua Distribuída



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

Exclusão Mútua Distribuída algoritmo mono recurso

Algoritmo para DiMex, no processo p com id:

implementa ***DistributedMutualExclusion,***
instância ***dmx***
usa ***PerfectLinks*** instância ***pl;***

Vars

processes: conjunto de processos
pl: referência ao módulo de links perfeitos
waiting: sub-onjunto de processos esperando
st: [noMX, wantMX, inMX]
lcl,
reqTs: timestamp
nbrResps: int

upon event [dmx, Init] do

st := noMX
lcl := 0
reqTs := 0
nbrResps := 0
waiting :={}
pl := inicializa módulo pl

function after([ts1,id1], [ts2,id2]) :

return (ts1 > ts2) OR (ts1 == ts2 AND id1 > id2)

quando acontece evento [dmx, Entry] faça

aplicação solicita
forall processes q != p
gera evento [pl , Send | q, [reqEntry, id, reqTs]
usando pl, Send para processo q msg tipo reqEntry com id,reqTs
st := wantMX

quando acontece evento [dmx, Exit] faça

aplicação sinaliza fim
...
gera evento [pl, Send | q , [respOk]]
usando pl, Send para processo q msg tipo respOk
st := noMX

quando acontece evento de entrega de mensagem
[pl, Deliver | q, [reqEntry, rid, rts]
pl entrega mensagem de q, tipo reqEntry com parametros rid,rts
...
gera evento [pl, Send | q , [respOk]]
usando pl, Send para processo q msg tipo respOk
.....

quando acontece evento de entrega de mensagem
[pl, Deliver | q, [respOk]]
pl entrega mensagem de q, tipo reqEntry
...
gera evento [dmx, Deliver | resp]
sinaliza que aplicação pode entrar
...

Exclusão Mútua Distribuída
algoritmo mono recurso

Algoritmo para DiMex, no processo p com id:
implementa ***DistributedMutualExclusion,***
instância ***dmx***
usa ***PerfectLinks*** instância ***pl;***

Vars
processes: conjunto de processos
pl: referência ao módulo de links perfeitos
waiting: sub-onjunto de processos esperando
st: [noMX, wantMX, inMX]
lcl,
reqTs: timestamp
nbrResps: int

upon event [dmx, Init] do
st := noMX
lcl := 0
reqTs := 0
nbrResps := 0
waiting :={}
pl := inicializa módulo pl

function after([ts1,id1], [ts2,id2]) :
return (ts1 > ts2) OR (ts1 == ts2 AND id1 > id2)

aplicação aplicação solicita[dmx, Entry] faça
lcl ++
reqTs := lcl
nbrResps := 0
forall processes q != p
manda msg [pl , Send | q, [reqEntry, id, reqTs]
st := wantMX

quando aplicação avisa [dmx, Exit] faça
forall q em waiting
manda msg[pl, Send | q , [respOk]]
waiting := {}
st := noMX

quando pl entregar msg
[pl, Deliver | q, [reqEntry, rid, rts]
if (st == noMX) OR
(st == wantMX AND after([reqTs,id], [rts,rid]))
then gera evento [pl, Send | q , [respOk]]
else waiting := waiting + [q]
// **if** (st == inMX) OR (st == wantMX AND [rts,rid]> [reqTs,id])
// **then** waiting := waiting + [q] **else** // empty
lcl := max(lcl, rts)

quando pl entregar msg [pl, Deliver | q, [respOk]]
nbrResps++
if nbrResps == #processes -1
then gera evento [dmx, Deliver | resp]
st := inMX

Mapeando algoritmo para uma implementação em Go

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

Exclusão Mútua Distribuída algoritmo mono recurso

Algoritmo para DiMex, no processo p com id:

implementa ***DistributedMutualExclusion,***
instância ***dmx***
usa ***PerfectLinks*** instância ***pl;***

Vars

processes: conjunto de processos

pl: referência ao módulo de links perfeitos

waiting: sub-onjunto de processos esperando

st: [noMX, wantMX, inMX]

lcl,

reqTs: timestamp

nbrResps: int

```
type DIMEX_Module struct {
    Req        chan dmxReq  // canal para receber pedidos da aplicacao (REQ e EXIT)
    Ind        chan dmxResp // canal para informar aplicacao que pode acessar
    addresses  []string     // endereco de todos, na mesma ordem
    id         int          // identificador do processo - é o indice no array de enderecos acima
    st         State        // estado deste processo na exclusao mutua distribuida
    waiting    []bool       // processos aguardando tem flag true
    lcl        int          // relogio logico local
    reqTs      int          // timestamp local da ultima requisicao deste processo
    nbrResps   int
    dbg        bool

    Pp2plink *PP2PLink.PP2PLink // acesso aa comunicacao enviar por PP2PLinq.Req  e receber por PP2PLinq.Ind
}
```

Exclusão Mútua Distribuída algoritmo mono recurso

Algoritmo para DiMex, no processo p com id:

implementa ***DistributedMutualExclu***
instância ***dmx***
usa ***PerfectLinks*** instância ***pl;***

Vars

processes: conjunto de processos
pl: referência ao módulo de links perfeit
waiting: sub-onjunto de processos espe
st: [noMX, wantMX, inMX]
lcl,
reqTs: timestamp
nbrResps: int

upon event [dmx, Init] do

st := noMX
lcl := 0
reqTs := 0
nbrResps := 0
waiting :={}
pl := inicializa módulo pl

```
// ------------------------------------------------------------------
// ------- inicializacao
// ------------------------------------------------------------------

func NewDIMEX(_addresses []string, _id int, _dbg bool) *DIMEX_Module {

    p2p := PP2PLink.NewPP2PLink(_addresses[_id], _dbg)

    dmx := &DIMEX_Module{
        Req: make(chan dmxReq, 1),
        Ind: make(chan dmxResp, 1),

        addresses: _addresses,
        id:        _id,
        st:        noMX,
        waiting:   make([]bool, len(_addresses)),
        lcl:       0,
        reqTs:     0,
        dbg:       _dbg,

        Pp2plink: p2p}

    for i := 0; i < len(dmx.waiting); i++ {
        dmx.waiting[i] = false
    }
    dmx.Start()
    dmx.outDbg("Init DIMEX!")
    return dmx
}
```

Exclusão Mútua Distribuída
algoritmo mono recurso

Algoritmo para DiMex, no processo p com id:

implementa ***DistributedMutualExclusion,***
instância ***dmx***
usa ***PerfectLinks*** instância ***pl:***

```
// ------------------------------------------------------------------------------------
// ------- nucleo do funcionamento
// ------------------------------------------------------------------------------------

func (module *DIMEX_Module) Start() {

    go func() {
        for {
            select {
            case dmxR := <-module.Req: // vindo da  aplicação
                if dmxR == ENTER {
                    module.outDbg("app pede mx")
                    module.handleUponReqEntry() // ENTRADA DO ALGORITMO

                } else if dmxR == EXIT {
                    module.outDbg("app libera mx")
                    module.handleUponReqExit() // ENTRADA DO ALGORITMO
                }

            case msgOutro := <-module.Pp2plink.Ind: // vindo de outro processo
                //fmt.Printf("dimex recebe da rede: ", msgOutro)
                if strings.Contains(msgOutro.Message, "respOK") {
                    module.outDbg("         <<<---- responde! " + msgOutro.Message)
                    module.handleUponDeliverRespOk(msgOutro) // ENTRADA DO ALGORITMO

                } else if strings.Contains(msgOutro.Message, "reqEntry") {
                    module.outDbg("          <<<---- pede??  " + msgOutro.Message)
                    module.handleUponDeliverReqEntry(msgOutro) // ENTRADA DO ALGORITMO

                }
            }
        }
    }()
}
```

aplicação aplicação solicita[dmx, Entry] faça
  lcl ++
  reqTs := lcl
  nbrResps := 0
  forall processes q != p
    manda msg [pl , Send | q, [reqEntry, id, reqTs]
  st := wantMX

quando aplicação avisa [dmx, Exit] faça
  forall q em waiting
    manda msg[pl, Send | q , [respOk]]
  waiting := {}
  st := noMX

quando pl entregar msg
  [pl, Deliver | q, [reqEntry, rid, rts]
  if (st == noMX) OR
    (st == wantMX AND after([reqTs,id], [rts,rid]))
  then **gera evento [pl, Send | q , [respOk]]**
  else waiting := waiting + [q]
    // **if** (st == inMX) OR (st == wantMX AND [rts,rid]> [reqTs,id])
    // **then** waiting := waiting + [q] **else** // empty
  lcl := max(lcl, rts)

quando pl entregar msg [pl, Deliver | q, [respOk]]
  nbrResps++
  if nbrResps == #processes -1
  then **gera evento [dmx, Deliver | resp]**
    st := inMX

```
// ------------------------------------------------------------------------------------------
// ------- tratamento de pedidos vindos da aplicacao
// ------- UPON ENTRY
// ------- UPON EXIT
// ------------------------------------------------------------------------------------------

func (module *DIMEX_Module) handleUponReqEntry() {
    /*
            quando aplicação aplicação solicita[ dmx, Entry ]  faça
                lcl ++
                reqTs := lcl
                nbrResps := 0
                forall processes q != p
                    manda msg [ pl , Send | q, [ reqEntry, id, reqTs ]
                st := wantMX
    */
}

func (module *DIMEX_Module) handleUponReqExit() {
    /*
                quando aplicação avisa [ dmx, Exit   ]  faça
                    forall  q  em waiting
                        manda msg[ pl, Send | q , [ respOk ]  ]
                    waiting := {}
                    st := noMX
    */
}

// ------------------------------------------------------------------------------------------
// ------- tratamento de mensagens de outros processos
// ------- UPON respOk
// ------- UPON reqEntry
// ------------------------------------------------------------------------------------------

func (module *DIMEX_Module) handleUponDeliverRespOk(msgOutro PP2PLink.PP2PLink_Ind_Message) {
    /*
                quando pl entregar msg  [ pl, Deliver | q, [ respOk ] ]
                    nbrResps++
                    if nbrResps == #processes -1
                    then gera evento [ dmx, Deliver | resp ]
                        st := inMX
    */
}

func (module *DIMEX_Module) handleUponDeliverReqEntry(msgOutro PP2PLink.PP2PLink_Ind_Message)
    // outro processo quer entrar na SC
    /*
                quando pl entregar msg [ pl, Deliver | q, [ reqEntry, rid, rts ]
                   if  (st == noMX)   OR
                      (st == wantMX  AND  after([reqTs,id], [rts,rid]) )
                   then  gera evento [ pl, Send | q , [ respOk ]  ]
                   else waiting := waiting + [ q ]
                        // if   (st == inMX) OR  (st == wantMX AND [rts,rid]> [reqTs,id])
                        // then   waiting := waiting + [ q ]     else  // empty
                   lcl := max(lcl, rts)
    */
}
```

```
upon event [ dmx, Entry ]  do
    lcl ++
    reqTs := lcl
    nbrResps := 0
    forall processes q != p
        trigger [ pl , Send | q, [ reqEntry, id, reqTs ]
    st := wantMX

upon event [ dmx, Exit   ]  do
    forall  q  em waiting
        trigger [ pl, Send | q , [ respOk ]  ]
    st := noMX
    waiting := {}
upon event [ pl, Deliver | q, [ respOk ] ]
    nbrResps++
    if nbrResps == #processes -1
    then trigger [ dmx, Deliver | resp ]
            st := inMX

upon event [ pl, Deliver | q, [ reqEntry, rid, rts ]  do
    if  (st == noMX)   OR
       (st == wantMX AND  after([reqTs,id], [rts,rid]) )
    then  trigger [ pl, Send | q , [ respOk ]  ]
    else waiting := waiting + [ q ]
            // if   (st == inMX) OR  (st == wantMX AND [rts,rid]> [reqTs,id])
            // then waiting := waiting + [ q ]
            // else  // empty
    lcl := max(lcl, rts)
```

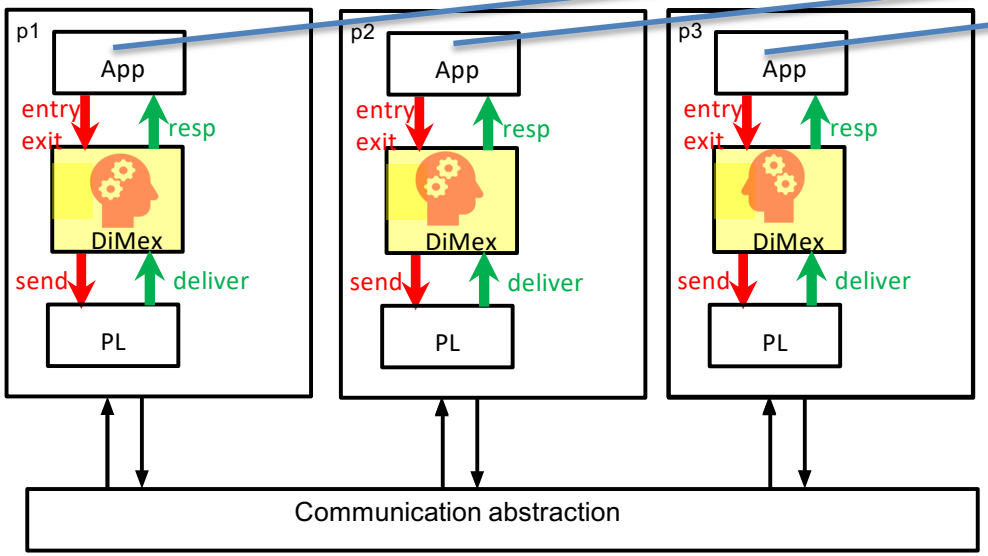
Laboratório

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

Exclusão Mútua Distribuída - Impl

- Encontre no Moodle a pasta de código, abra no local onde guarda seus programas Go
- Dado o DIMEX-Tamplate.go, implementar DiMex
- Acoplar aplicação: useDiMex-f.go - leia os comentarios - gera um arquivo texto para avaliar se ocorre algum erro
- Mínimo 3 processos rodando DiMex: deve poder rodar com mais
- Argumentar que as propriedades da exclusão mútua são garantidas

Exclusão Mútua Distribuída - Impl




```
f: arquivo que todos acessam
cada processo é
  loop {
     entry
     espera resp
     // inicio sc
     abre f
     10 append de linha em f
     fecha f
     // fim sc
     exit
   }
```

useDiMex-f.go faz isto

fazer isto sem mutex para ver o problema

fazer com mutex: arquivo deve ser consistente
("|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|. ....")

→

```
f: arquivo que todos acessam
cada processo é
  loop {
     abre f
     10 append de linha em f
     fecha f
}
```


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

Labora tório

- Voce pode rodar localmente (uma máquina) ou distribuído
- Exemplo de chat em uma máquina
- veja comentários no fonte com os comandos