

Fundamentos de Processamento Paralelo e Distribuído

Algoritmo Distribuído de Exclusao Mútua

Fernando Luís Dotti – PUCRS

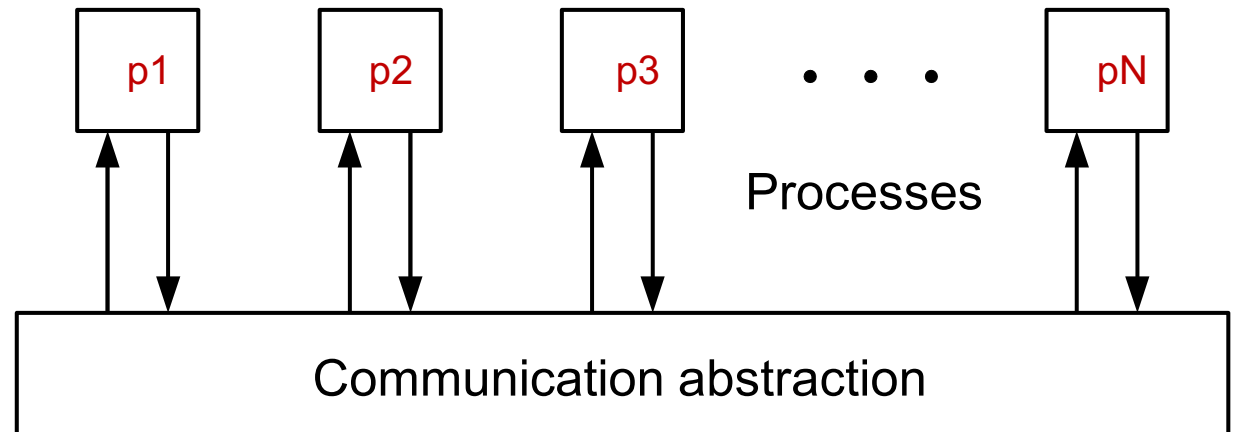
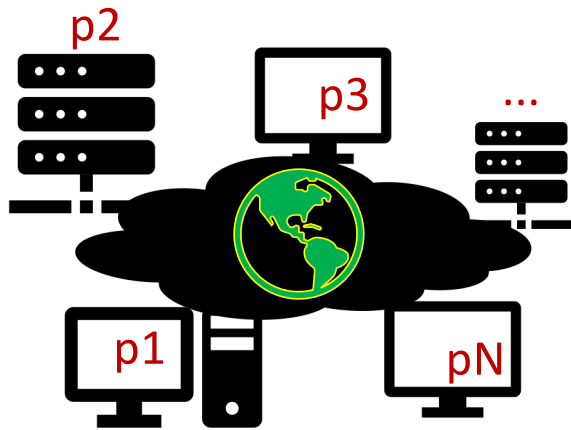
Material baseado no livro

Introduction to Reliable and Secure Distributed Programming

Christian Cachin, Rachid Gerraoui, Luís Rodrigues

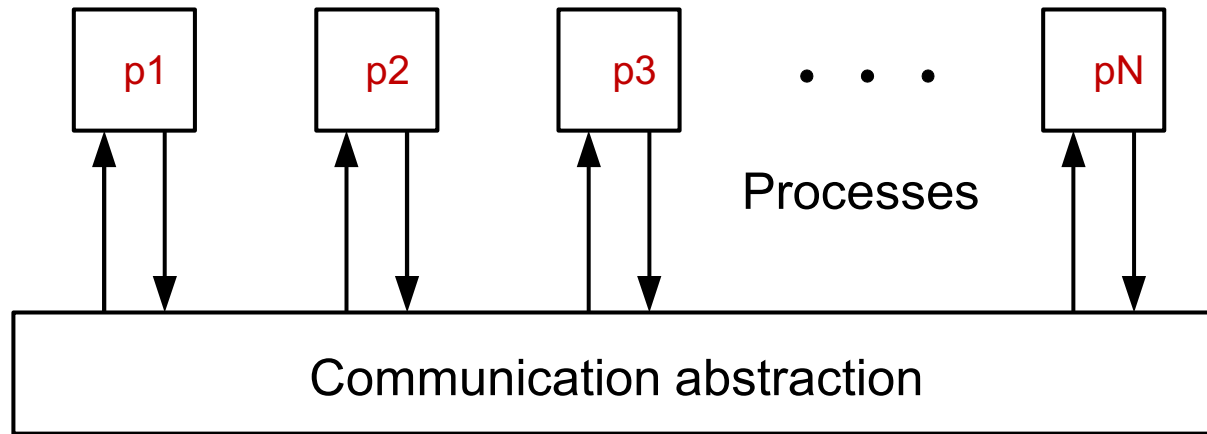
e em material didático disponibilizado pelos autores

Abstrações para Programação Distribuída



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

Exemplo: acesso a recurso comum

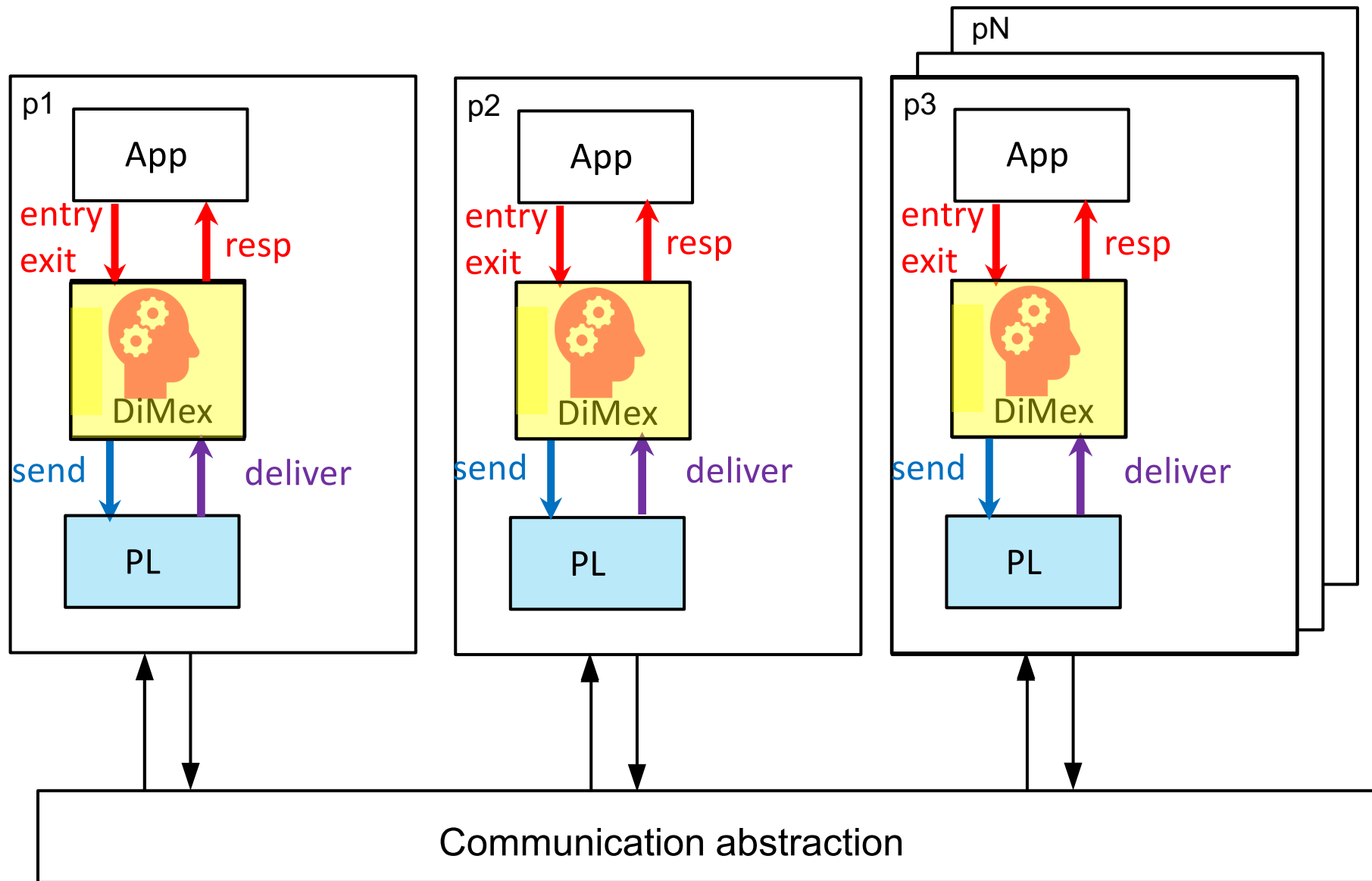


- Os processos tem acesso a um mesmo recurso (por exemplo um arquivo em um sistema de arquivos distribuído) e devem utilizá-lo de forma mutuamente exclusiva
- Entretanto a plataforma não provê este tipo de funcionalidade

Algoritmo de Ricart Agrawalla

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

Visão Geral -Exclusão Mútua Distribuída



Exclusão Mútua Distribuída - Propriedades

Modulo:

DistributedMutualExclusion, instância dmx .

Eventos:

Request: [$dmx, Entry \mid r$]: solicita acesso a r

Request: [$dmx, Exit \mid r$]: avisa fim de acesso a r

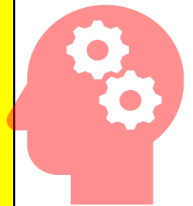
Indicação: [$dmx, resp \mid r$]: permite acesso

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

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

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 esperandoreação 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 preação a outros processos
- *antes ou depois dado por timestamp lógico composto de relógio lógico e identificador do processo.*

Exclusão Mútua Distribuída algoritmo mono recurso

Algoritmo para DiMex, no processo p com id:
implementa *DistributedMutualExclusion*

Vars

processes: conjunto de processos

upon event [pl, Init] do

.

upon event [dmX, Entry] do

...

trigger [pl , Send | q, [reqEntry, id, reqTs]

...

upon event [dmX, Exit] do

....

trigger [pl, Send | q , [respOk]]

...

upon event [pl, Deliver | q, [reqEntry, rid, rts] do

....

trigger [pl, Send | q , [respOk]]

.....

upon event [pl, Deliver | q, [respOk]]

...

trigger [dmX, Deliver | resp]

Exclusão Mútua Distribuída algoritmo mono recurso

Algoritmo para DiMex, no processo p com id:
implementa *DistributedMutualExclusion*

Vars

processes: conjunto de processos

upon event [pl, Init] do

upon event [dmX, Entry] do

...

trigger [pl , Send | q, [reqEntry, id, reqTs]

...

upon event [dmX, Exit] do

....

trigger [pl, Send | q , [respOk]]

...

upon event [pl, Deliver | q, [reqEntry, rid, rts] do

....

trigger [pl, Send | q , [respOk]]

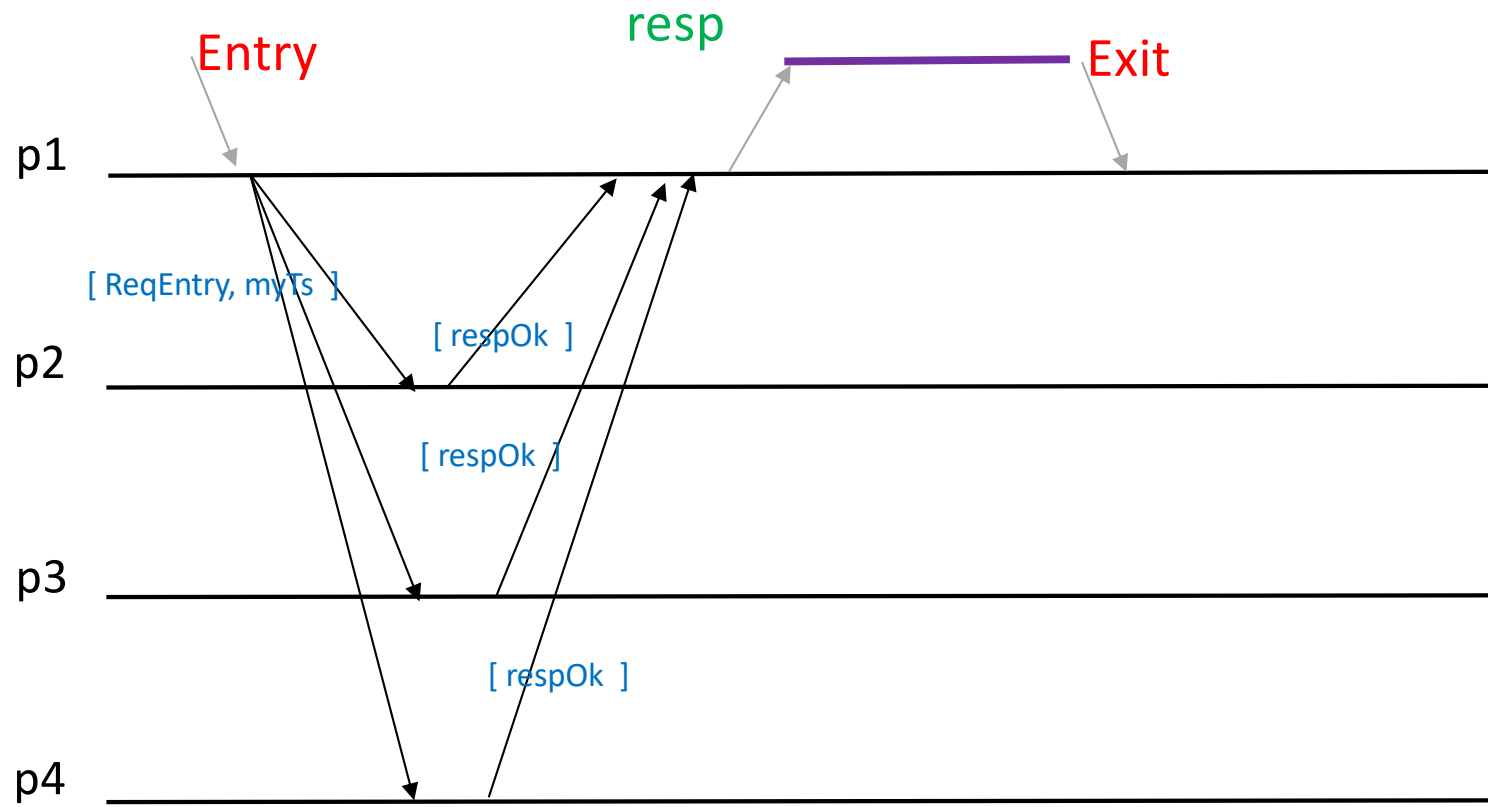
.....

upon event [pl, Deliver | q, [respOk]]

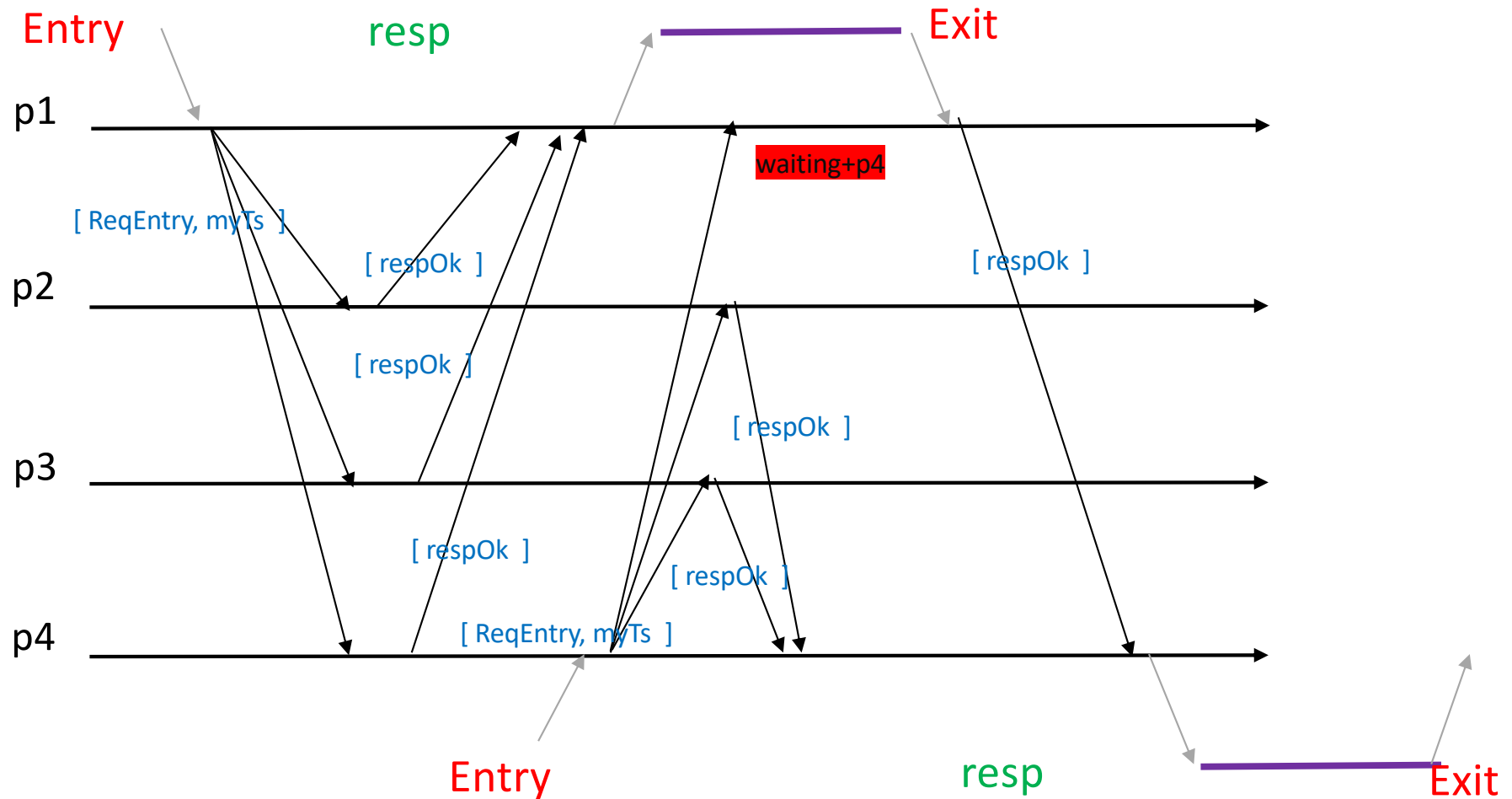
...

trigger [dmX, Deliver | resp]

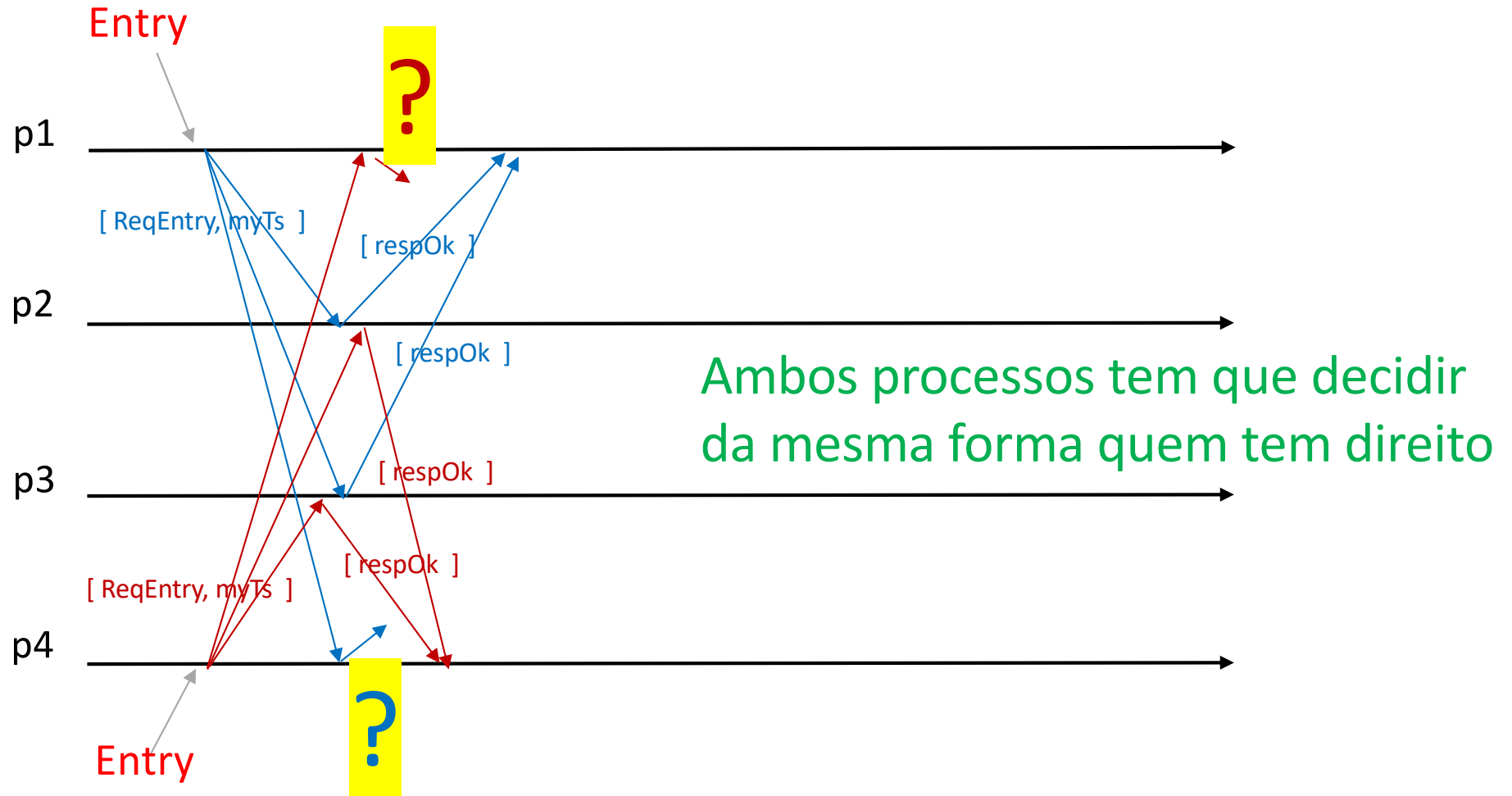
Algoritmo de Ricart Agrawalla



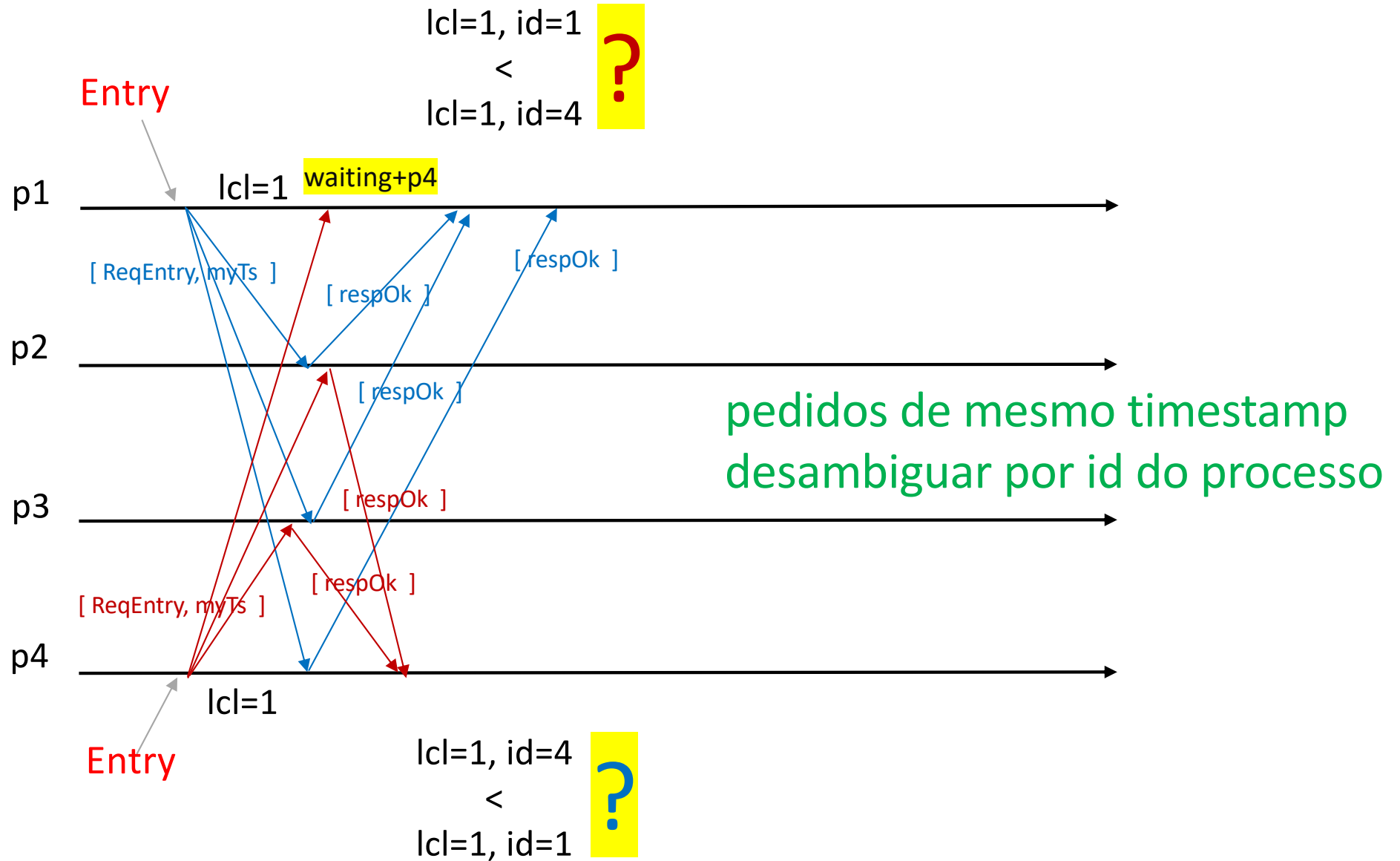
Algoritmo de Ricart Agrawalla



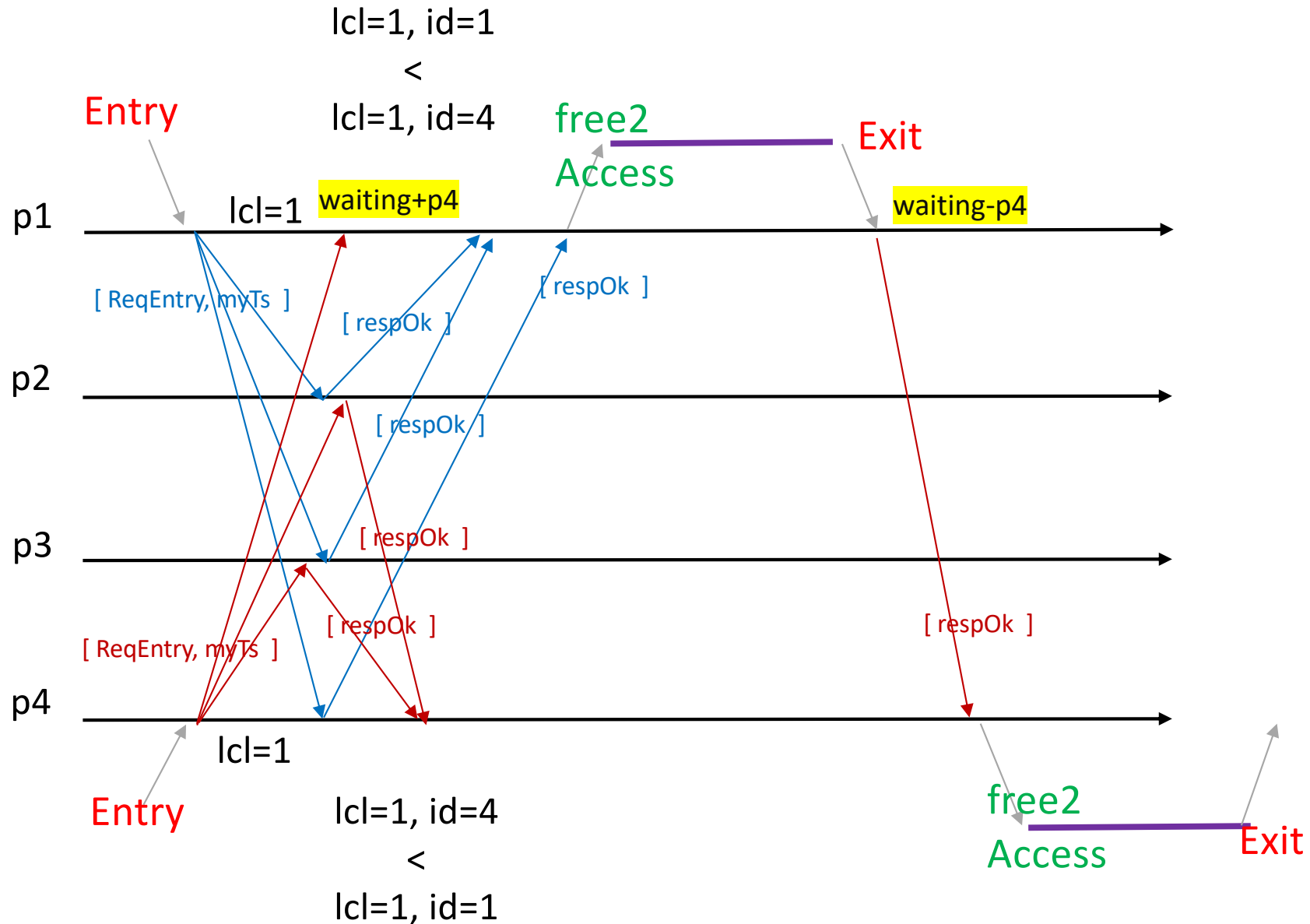
Algoritmo de Ricart Agrawalla



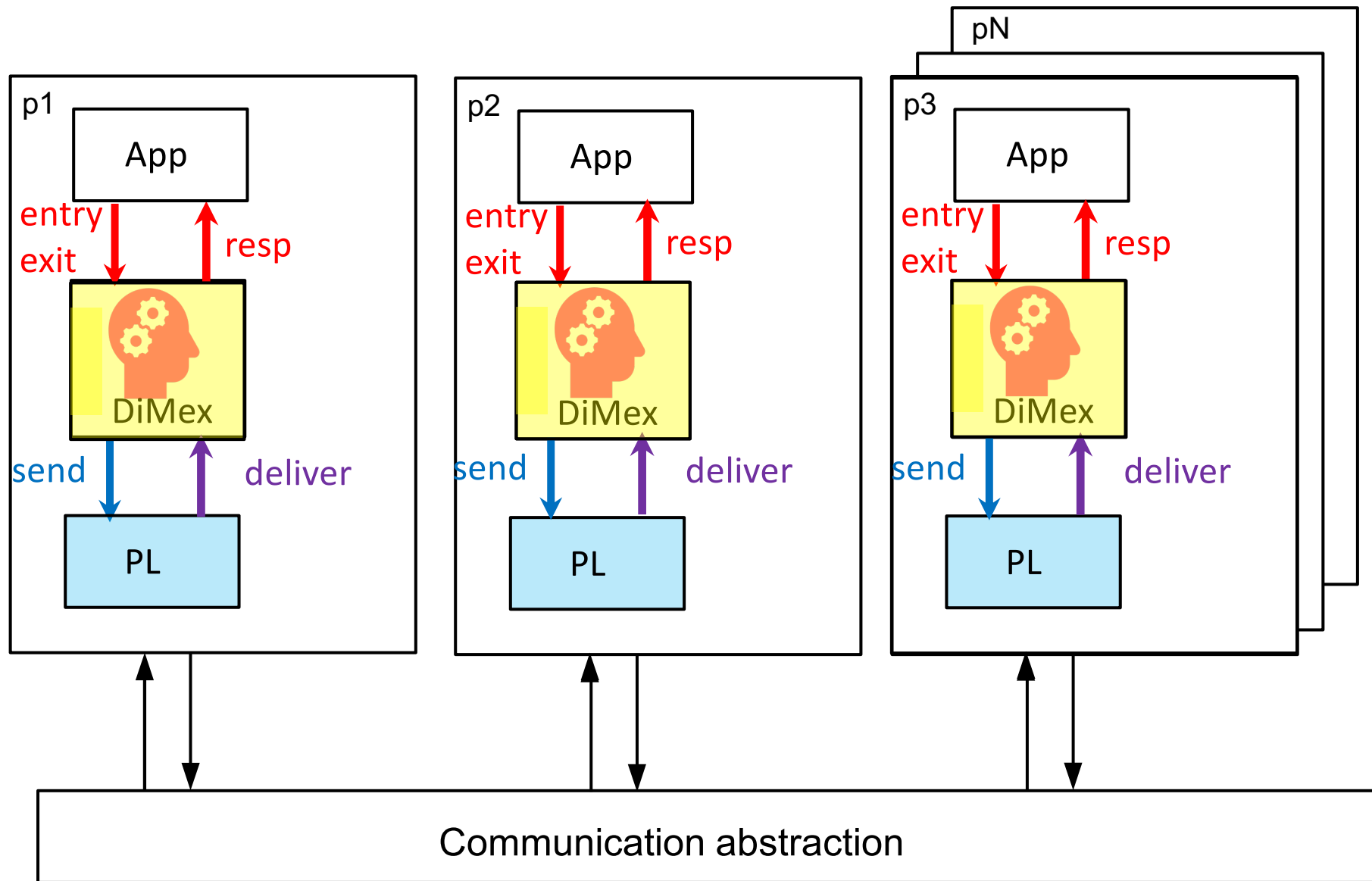
Algoritmo de Ricart Agrawalla



Algoritmo de Ricart Agrawalla



Visão Geral -Exclusão Mútua Distribuída



Exclusão Mútua Distribuída algoritmo mono recurso

Algoritmo para DiMex, no processo p com id:
implementa *DistributedMutualExclusion*

Vars

processes: conjunto de processos

upon event [pl, Init] do

upon event [dmX, Entry] do

...

trigger [pl, Send | q, [reqEntry, id, reqTs]

...

upon event [dmX, Exit] do

....

trigger [pl, Send | q, [respOk]]

...

upon event [pl, Deliver | q, [reqEntry, rid, rts] do

....

trigger [pl, Send | q, [respOk]]

.....

upon event [pl, Deliver | q, [respOk]]

...

trigger [dmX, Deliver | resp]

Exclusão Mútua Distribuída algoritmo mono recurso

Algoritmo para DiMex, no processo p com id:
implementa *DistributedMutualExclusion*

Vars

processes: conjunto de processos
waiting: sub-onjunto de processos esperando
st: [noMX, wantMX, inMX]
lcl,
reqTs: timestamp
nbrResps: int

upon event [pl, Init] do

st := noMX
lcl := 0
reqTs := 0
nbrResps := 0
waiting := {}

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

upon event [dmX, Entry] do

...
trigger [pl , Send | q, [reqEntry, id, reqTs]
...

upon event [dmX, Exit] do

....
trigger [pl, Send | q , [respOk]]
...

upon event [pl, Deliver | q, [reqEntry, rid, rts] do

....
trigger [pl, Send | q , [respOk]]
.....

upon event [pl, Deliver | q, [respOk]]

...
trigger [dmX, Deliver | resp]

Exclusão Mútua Distribuída algoritmo mono recurso

Algoritmo para DiMex, no processo p com id:
implementa *DistributedMutualExclusion*

Vars

processes: conjunto de processos
waiting: sub-onjunto de processos esperando
st: [noMX, wantMX, inMX]
lcl,
reqTs: timestamp
nbrResps: int

upon event [pl, Init] do

st := noMX
lcl := 0
reqTs := 0
nbrResps := 0
waiting := {}

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

upon event [dmX, Entry] do

forall processes q != p

trigger [pl, Send | q, [reqEntry, id, reqTs]

st := wantMX

upon event [dmX, Exit] do

trigger [pl, Send | q, [respOk]]

st := noMX

upon event [pl, Deliver | q, [reqEntry, rid, rts] do

if

then trigger [pl, Send | q, [respOk]]

else waiting := waiting + [q]

upon event [pl, Deliver | q, [respOk]]

nbrResps++

if nbrResps == #processes -1

then trigger [dmX, Deliver | resp]

st := inMX

Exclusão Mútua Distribuída algoritmo mono recurso

Algoritmo para DiMex, no **processo p** com **id**:
implementa *DistributedMutualExclusion*

Vars

processes: conjunto de processos (**addresses**)

waiting: sub-conjunto de processos esperando

st: [noMX, wantMX, inMX]

lcl,

reqTs: timestamp

nbrResps: int

upon event [pl, Init] do

st := noMX

lcl := 0

reqTs := 0

nbrResps := 0

waiting := {}

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

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

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, [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)

upon event [pl, Deliver | q, [respOk]]

nbrResps++

if nbrResps == #processes -1

then trigger [dmX, Deliver | resp]

st := inMX

Exclusão Mútua Distribuída algoritmo mono recurso

Algoritmo para DiMex, no processo p com id:
implementa *DistributedMutualExclusion*

Vars

processes: conjunto de processos

waiting: sub-onjunto de processos esperando

st: [noMX, wantMX, inMX]

lcl,

reqTs: timestamp

nbrResps: int

upon event [pl, Init] do

st := noMX

lcl := 0

reqTs := 0

nbrResps := 0

waiting := {}

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

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

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, [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)

upon event [pl, Deliver | q, [respOk]]

nbrResps++

if nbrResps == #processes -1

then trigger [dmX, Deliver | resp]

st := inMX

Mapeando algoritmo para uma implementação em Go

Exclusão Mútua Distribuída algoritmo mono recurso

Algoritmo para DiMex, no **processo p** com **id**:
implementa *DistributedMutualExclusion*

Vars

processes: conjunto de processos (**addresses**)

waiting: sub-conjunto de processos esperando

st: [noMX, wantMX, inMX]

lcl,

reqTs: timestamp

nbrResps: int

```
type DIMEX_Module struct {
    Req      chan dmxFreq // canal para receber pedidos da aplicacao (REQ e EXIT)
    Ind      chan dmxFreq // 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          // relógio 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** c
implementa *DistributedMutualE*

Vars

processes: conjunto de processos

waiting: sub-conjunto de processos

st: [noMX, wantMX, inMX]

lcl,

reqTs: timestamp

nbrResps: int

upon event [pl, Init] do

st := noMX

lcl := 0

reqTs := 0

nbrResps := 0

waiting := {}

```
// ----- inicializacao
// -----
func NewDIMEX(_addresses []string, _id int, _dbg bool) *DIMEX_Module {
    p2p := PP2PLink.NewPP2PLink(_addresses[_id], _dbg)

    dmex := &DIMEX_Module{
        Req: make(chan dmexReq, 1),
        Ind: make(chan dmexResp, 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(dmex.waiting); i++ {
        dmex.waiting[i] = false
    }

    dmex.Start()
    dmex.outDbg("Init DIMEX!")
    return dmex
}
```

Exclusão Mútua Distribuída algoritmo mono recurso

Algoritmo para DiMex, no **processo p** com **id**:
implementa *DistributedMutualExclusion*

```
// ----- 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
                }
            }
        }
    }()
}
```

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

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
```

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)
```

```

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

func (module *DIMEX_Module) handleUponReqEntry() {
    /*
    upon event [ dmX, Entry | r ] do
        lts.ts++
        myTs := lts
        resps := 0
        para todo processo p
            trigger [ pl, Send | [ reqEntry, r, myTs ]
        estado := queroSC
    */
}

func (module *DIMEX_Module) handleUponReqExit() {
    /*
    upon event [ dmX, Exit | r ] do
        para todo [p, r, ts] em waiting
            trigger [ pl, Send | p, [ respOk, r ]
        estado := naoQueroSC
        waiting := {}
    */
}

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

func (module *DIMEX_Module) handleUponDeliverRespOk(msgOutro PP2PLink.PP2PLink_Ind_Message) {
    /*
    upon event [ pl, Deliver | p, [ respOk, r ]
        resps++
        se resps = N
            então trigger [ dmX, Deliver | free2Access ]
            estado := estouNaSC
    */
}

func (module *DIMEX_Module) handleUponDeliverReqEntry(msgOutro PP2PLink.PP2PLink_Ind_Message) {
    // outro processo quer entrar na SC
    /*
    upon event [ pl, Deliver | p, [ reqEntry, r, rts ] do
        se (estado == naoQueroSC) OR
        | (estado == QueroSC AND myTs > ts)
        então trigger [ pl, Send | p, [ respOk, r ]
        senão
        | se (estado == estouNaSC) OR
        | | (estado == QueroSC AND myTs < ts)
        então postergados := postergados + [p, r]
        lts.ts := max(lts.ts, rts.ts)
    */
}

```

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

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 comentários - 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

Labora tório

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

```
30-SD — chatBEB • go run chatBEB.go 127.0.0.1:5001 127.0.0.1:6001 127.0.0.1:7001 — 113x17
Last login: Mon May 29 15:20:35 on ttys001

The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
MacBook-Pro-de-Fernando:~ fldotti$ cd /Users/fldotti/Library/Mobile\ Documents/com~apple~CloudDocs/Tudo/Docenci
aPesquisa/EnsinoGradu/G-FPPD/Versao2023-1/programasGo/30-SD
MacBook-Pro-de-Fernando:30-SD fldotti$ go run chatBEB.go 127.0.0.1:5001 127.0.0.1:6001 127.0.0.1:7001
[[127.0.0.1:5001 127.0.0.1:6001 127.0.0.1:7001]
aaaa
Message from 127.0.0.1:5001: aaaa
Message from 127.0.0.1:6001: bbbb
Message from 127.0.0.1:7001: ccc

30-SD — chatBEB • go run chatBEB.go 127.0.0.1:6001 127.0.0.1:5001 127.0.0.1:7001 — 113x16
Last login: Wed May 31 10:49:14 on ttys003

The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
MacBook-Pro-de-Fernando:~ fldotti$ cd /Users/fldotti/Library/Mobile\ Documents/com~apple~CloudDocs/Tudo/Docenci
aPesquisa/EnsinoGradu/G-FPPD/Versao2023-1/programasGo/30-SD
MacBook-Pro-de-Fernando:30-SD fldotti$ go run chatBEB.go 127.0.0.1:6001 127.0.0.1:5001 127.0.0.1:7001
[[127.0.0.1:6001 127.0.0.1:5001 127.0.0.1:7001]
Message from 127.0.0.1:5001: aaaa
bbbb
Message from 127.0.0.1:6001: bbbb
Message from 127.0.0.1:7001: ccc

30-SD — chatBEB • go run chatBEB.go 127.0.0.1:7001 127.0.0.1:6001 127.0.0.1:5001 — 113x15
Last login: Wed May 31 10:49:09 on ttys002

The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
MacBook-Pro-de-Fernando:~ fldotti$ cd /Users/fldotti/Library/Mobile\ Documents/com~apple~CloudDocs/Tudo/Docenci
aPesquisa/EnsinoGradu/G-FPPD/Versao2023-1/programasGo/30-SD
MacBook-Pro-de-Fernando:30-SD fldotti$ go run chatBEB.go 127.0.0.1:7001 127.0.0.1:6001 127.0.0.1:5001
[[127.0.0.1:7001 127.0.0.1:6001 127.0.0.1:5001]
Message from 127.0.0.1:5001: aaaa
Message from 127.0.0.1:6001: bbbb
ccc
Message from 127.0.0.1:7001: ccc
```