Lección 11: Algoritmos distribuidos

- Introducción
- Modelo de sistema distribuido
- Exclusión mutua distribuida: el algoritmo de Ricart-Agrawala
- Exclusión mutua distribuida: algoritmos de paso del testigo (*token-passing*)

Introducción

- Concepto de algoritmo distribuido
  - Requieren un modelo de sistema distribuido
  - Pensamos en sistemas débilmente acoplados
- Interés por:
  - los algoritmos distribuidos de exclusión mutua
    - gestión del acceso a una BBDD compartida por un conjunto de procesos distribuidos
  - los algoritmos distribuidos de consenso

Modelo de sistema distribuido

- Arquitectura
  - conjunto de nodos que comunican enviando y recibiendo mensajes de forma **asíncrona**
  - dentro un nodo, los **procesos** se ejecutan con los mecanismos de memoria compartida estudiados
  - sólo se permite fallos parciales
    - por lo menos la parte de comunicación funciona correctamente

Modelo de sistema distribuido

- Asunciones sobre la comunicación
  - cada par de nodos comparten un canal bidireccional
    - conexión completa
  - los canales distribuyen los mensajes sin error, aunque puede que no en el orden en que se enviaron
  - la latencia es finita, aunque arbitraria
- Asunciones sobre los procesos
  - cada nodo tiene un identificador único **`myID`**

Modelo de sistema distribuido

- Operaciones sobre mensajes
  - **send(message_type, destination *[, parameters]***)**
    - ***message_type***: distinguiremos tipos de mensajes
    - ***destination***: myID del proceso al que se envía el mensaje
    - ***parameters***: información que se desea transmitir. Serán expresiones
  - **receive(message_type *[, var]***)**
    - ***message_type***: para seleccionar el tipo de mensaje que espero
    - lista de variables para recoger la información transmitida
  - atómicas, entre nodos y dentro de un nodo
- ¿Parecidos/diferencias respecto a Linda?

Modelo de sistema distribuido

- Ejemplo de envío y recepción

31

```
string resp

send(request,34,'¿Cine?',7,myID)
receive(reply,resp)
```

34

```
string prop
integer hora,sender

receive(request,prop,hora,sender)
if (prop='¿Cine?') AND (sender=31)
    send(reply,sender,'Tengo que estudiar PSCD')
```

Algoritmo de *Ricart-Agrawala* (solicitud de turno)

- "Dr. Ricart led the team that wrote the code for the first implementation of TCP/IP for the IBM PC"
- "The Ricart-Agrawala Algorithm was the result of his dissertation work at the University of Maryland (1980)"
  - Ricart, Glenn; Ashok K. Agrawala (1981)
    An optimal algorithm for mutual exclusion in computer networks
    Communications of the ACM 24 (1): 9-17

Algoritmo de *Ricart-Agrawala* (solicitud de turno)

- Se basa en un "ticket"
  - pero sin servidor de tickets
- Solicitar permiso:
  - enviar mensaje a **todos los demás** nodos, con mi número de ticket
- Recibir mensajes:
  - Si su ticket es posterior al mío, avisarle cuando **yo salga** de la SC
  - Si su ticket es anterior, responderle inmediatamente
- Sección crítica:
  - Sólo puedo entrar si he recibido mensaje de permiso de **todos** los demás procesos
    - necesidad de establecer el número de procesos participantes

Algoritmo de *Ricart-Agrawala* (solicitud de turno)

- En cada nodo:

```
process MAIN
  while true
    SNC
    protocolo de entrada
    SC
    protocolo de salida
end
```

```
process RECEIVE
  procesa mensajes recibidos
end
```

Algoritmo de *Ricart-Agrawala* (solicitud de turno)

```
integer myNum := 0   -- generará uno
                     -- para cada intento
                     -- entrada en SC
set of integer deferred := {}
```

```
process MAIN
     while true
P1:       SNC
P2:       myNum = choose a ticket number
P3:       for all other nodes N involved
P4:           send(request,N,myID,myNum)
P5:       await reply's from all other nodes
P6:       SC
P7:       for all nodes N in deferred
P8:           remove N from deferred
P9:           send(reply,N,myID)
```

#nodos establecido

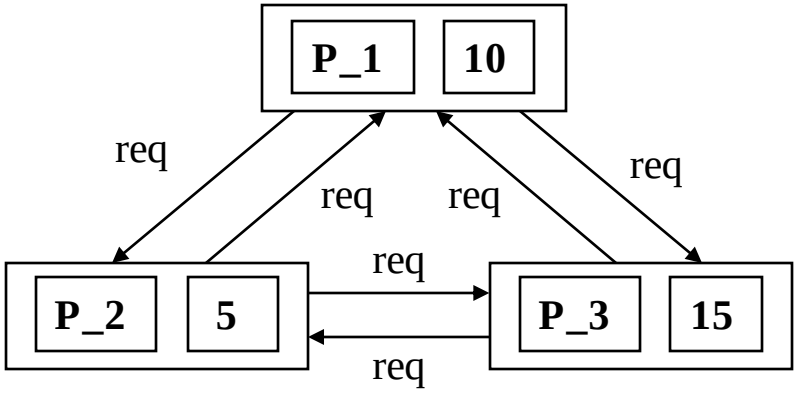
Algoritmo de *Ricart-Agrawala* (solicitud de turno)

```
integer myNum := 0
set of integer deferred := {}
```

```
process RECEIVE
     integer source, requestedNum

     while true
P10:    receive(request,source,requestedNum)
P11:    if (requestedNum < myNum)
P12:       send(reply,source,myID)
P13:    else
           add source to deferred
```

Algoritmo de *Ricart-Agrawala* (solicitud de turno)

- Dado el siguiente estado, ¿cómo el algoritmo gestiona la ejecución del programa?



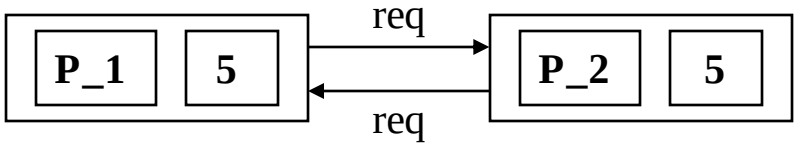
Algoritmo de *Ricart-Agrawala* (solicitud de turno)

- Posibles problemas:
  - ¿qué pasa si dos procesos eligen el mismo ticket?
  - ¿qué pasa si un proceso "rapidillo" tiene tendencia a elegir números de ticket bajos?
  - ¿qué pasa cuando hay procesos poco trabajadores?
    - se quedan tranquilamente en la SNC (*quiescent nodes*)

Algoritmo de *Ricart-Agrawala* (solicitud de turno)

- ¿Qué pasa si dos procesos eligen el mismo ticket?

```
process RECEIVE
     integer source, requestedNum

     while true
P10:    receive(request, source, requestedNum)
P11:    if (requestedNum < myNum)
P12:      send(reply, source, myID)
P13:    else add source to deferred
```



Algoritmo de *Ricart-Agrawala* (solicitud de turno)

- ¿Qué pasa si dos procesos eligen el mismo ticket?

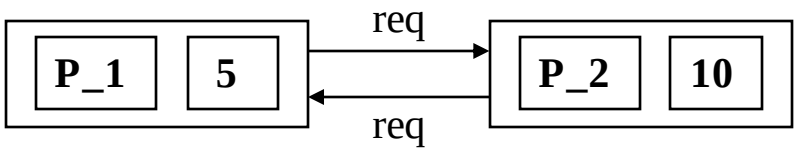
```
process RECEIVE
     integer source, requestedNum

    while true
P10:    receive(request, source, requestedNum)
P11:    if (requestedNum < myNum)
P12:        send(reply, source, myID)
P13:    el          urce to deferred
```

```
P11: if (requestedNum < myNum) or
        ((requestedNum = myNum) and (source < myID))
```

requestedNum << myNum

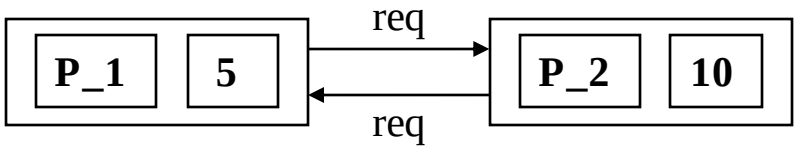
Algoritmo de *Ricart-Agrawala* (solicitud de turno)

- ¿Qué pasa si un proceso "rapidillo" tiene tendencia a elegir números de ticket bajos?



Algoritmo de *Ricart-Agrawala* (solicitud de turno)

- ¿Qué pasa si un proceso "rapidillo" tiene tendencia a elegir números de ticket bajos?




```
P2:       myNum := choose a ticket number
```

```
P2:       myNum := highestNum + 1
```

```
P10:     receive(request, source, requestedNum)
```

```
P10:     receive(request, source, requestedNum)
         highestNum := max(highestNum, requestedNum)
```

```
integer myNum := 0
set of integer deferred := {}
integer highestNum := 0
```

```
process MAIN
    while true
P1:     SNC
P2:     myNum := higeshtNum+1
P3:     for all other nodes N involved
P4:         send(request,N,myID,myNum)
P5:     await reply's from all other nodes
P6:     SC
P7:     for all nodes N in deferred
P8:        remove N from deferred
P9:        send(reply,N,myID)
```

```
process RECEIVE
     integer source, requestedNum

     while true
P10:    receive(request, source, requestedNum)
P11:    highestNum := max(highestNum,requestedNum)
P12:    if requestedNum << myNum
P13:       send(reply,source,myID)
P14:    else
P15:       add source to deferred
```

Algoritmo de *Ricart-Agrawala* (solicitud de turno)

- ¿Qué pasa cuando hay procesos poco trabajadores?

```
integer myNum := 0
set of integer deferred := {}
integer highestNum := 0
boolean requestCS := false
```

```
process MAIN
     while true
P1:       SNC
P2:       requestCS := TRUE
P3:       myNum := higeshtNum+1
P4:       for all other nodes N involved
P5:           send(request,N,myID,myNum)
P6:       await reply's from all other nodes
P7:       SC
P8:       requestCS := FALSE
P9:       for all nodes N in deferred
P10:          remove N from deferred
P11:          send(reply,N,myID)
```

Algoritmo de *Ricart-Agrawala* (solicitud de turno)

```
integer myNum := 0
set of integer deferred := {}
integer highestNum := 0
boolean requestCS := false
```

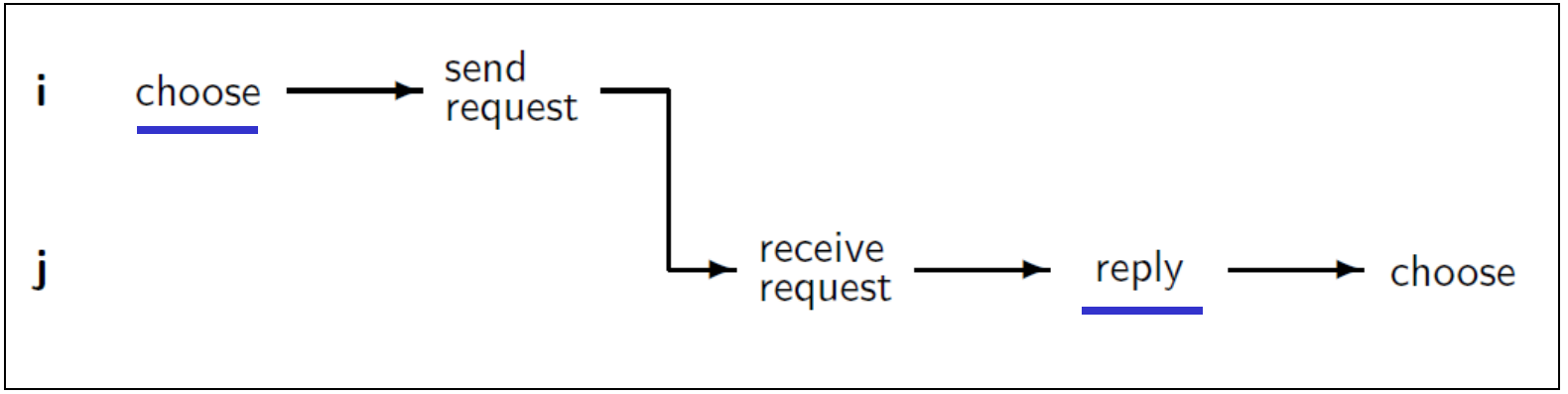
```
process RECEIVE
     integer source, requestedNum

     while true
P12:    receive(request, source, requestedNum)
P13:    highestNum := max(highestNum,requestedNum)
P14:    if NOT requestCS OR requestedNum << myNum
P15:       send(reply,source,myID)
P16:    else
P17:       add source to deferred
```

```
integer myNum := 0
set of integer deferred := {}
integer highestNum := 0
boolean requestCS := false
```

```
process MAIN
    while true
P1:     SNC
P2:     requestCS := TRUE
P3:     myNum := higeshtNum+1
P4:     for all other nodes N involved
P5:         send(request,N,myID,myNum)
P6:     await reply's from all other nodes
P7:     SC
P8:     requestCS := FALSE
P9:     for all nodes N in deferred
P10:        remove N from deferred
P11:        send(reply,N,myID)
```

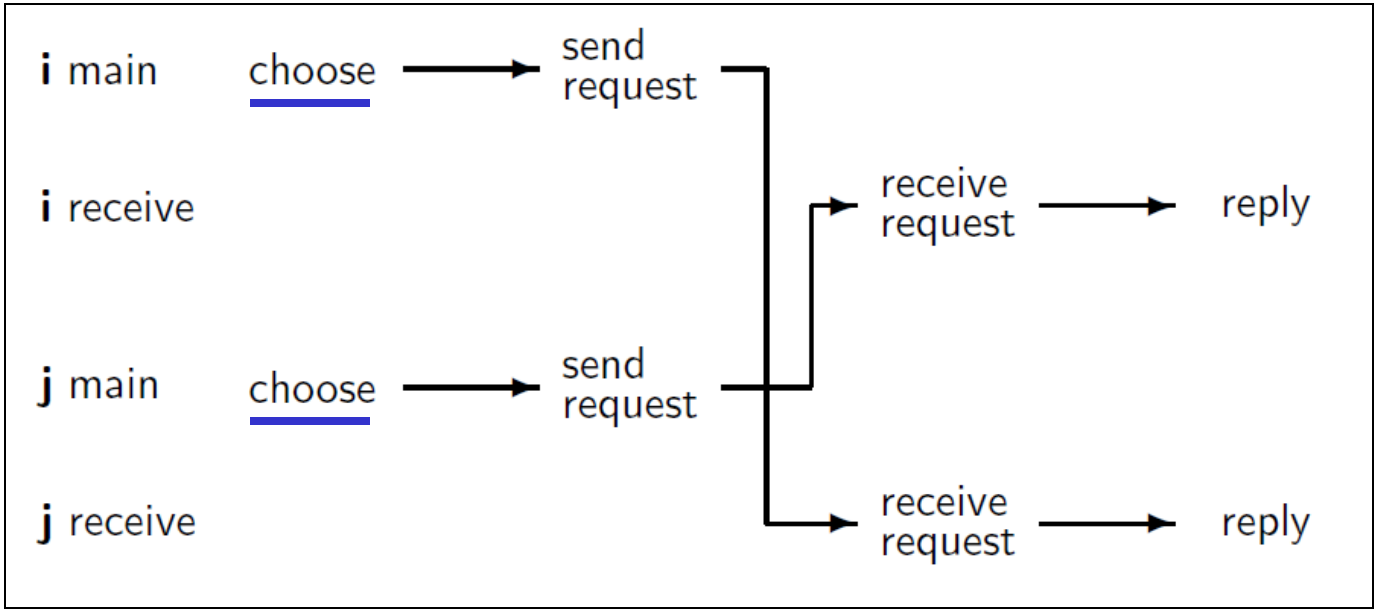
```
process RECEIVE
     integer source, requestedNum

     while true
P12:    receive(request, source, requestedNum)
P13:    highestNum := max(highestNum,requestedNum)
P14:    if NOT requestCS OR requestedNum << myNum
P15:       send(reply,source,myID)
P16:    else
P17:       add source to deferred
```

Algoritmo de *Ricart-Agrawala* (solicitud de turno)



M. Ben-Ari. Principles of Concurrent and Distributed Programming, Second edition © M. Ben-Ari 2006

Algoritmo de *Ricart-Agrawala* (solicitud de turno)



M. Ben-Ari. Principles of Concurrent and Distributed Programming, Second edition © M. Ben-Ari 2006

Algoritmo de *Ricart-Agrawala* (solicitud de turno)

- Cuestiones
  - ¿Tiene problemas de inanición?
  - ¿Cuántos mensajes se envían en una historia en que todos los nodos entran una vez en la SC?
  - ¿Puede el número de ticket crecer indefinidamente?
  - ¿Hay cota máxima para la diferencia entre dos números de ticket?
  - ¿Pueden ser todos los "deferred" no vacíos?
  - ¿Máximo número de elementos de un "deferred"?
  - ¿Máximo número de elementos entre todos los "deferred"?

Algoritmo de *Ricart-Agrawala* (paso de testigo)

- El algoritmo por permiso puede ser "ineficiente"
  - cada paso por la SC requiere n-1 mensajes enviados y recibidos
  - un nodo que quiera entrar dos veces seguidas tiene que ejecutar todo el protocolo
- Algoritmos basados en el "paso del testigo"
  - "token-based" algorithms
  - el sistema dispone de un único testigo, que se pasan entre los procesos
  - sólo el que lo tiene puede entrar en la SC
  - los demás procesos tienen que esperar
  - objetivo: disminuir el número de mensajes intercambiados
    - no nos engañemos, serán mensajes "más gordos"

Algori

```
boolean haveToken := true (node 1), false others
integer array[1..n] requested := (1..n,0)
integer array[1..n] granted := (1..n,0)
integer myNum := 0
boolean inCS := false
```

¿Lo tengo?

el token

veces he pedido entrar

¿Estoy dentro?

```
process MAIN
     while true
P1:       SNC
P2:       if NOT haveToken
P3:            myNum := myNum + 1
P4:            for all other nodes N
P5:               send(request,N,myID,myNum)
P6:            receive(token,granted)
P7:            haveToken := true
P8:       inCS := true
P9:       SC
P10:      granted[myID] := myNum
P11:      inCS := false
P12:      sendToken()
```

Algori

```
boolean haveToken := true (node 1), false others
integer array[1..n] requested := (1..n,0)
integer array[1..n] granted := (1..n,0)
integer myNum := 0
boolean inCS := false
```

¿Lo tengo?

el token

veces he pedido entrar

¿Estoy dentro?

```
process RECEIVE
     integer source,reqNum

     while true
P13:   receive(request,source,reqNum)
P14:   requested[source] := max(requested[source],reqNum)
P15:   if haveToken AND NOT inCS
P16:       sendToken()
```

```
operation sendToken
   if exists N such that requested[N] > granted[N]
      for some such N
          send(token,N,granted)
          haveToken := false
```

Algori

```
boolean haveToken := true (node 1), false others
integer array[1..n] requested := (1..n,0)
integer array[1..n] granted := (1..n,0)
integer myNum := 0
boolean inCS := false
```

¿Lo tengo?

el token

veces he pedido entrar

¿Estoy dentro?

```
process RECEIVE
     integer source,reqNum

     while true
P13:   receive(request,source,reqNum)
P14:   requested[source] := max(requested[source],reqNum)
P15:   sendToken()
```

```
operation sendToken
   if exists N such that requested[N] > granted[N]
      AND haveToken AND NOT inCS
           for some such N
               send(token,N,granted)
               haveToken := false
```

Algori

```
boolean haveToken := true (node 1), false others
integer array[1..n] requested := (1..n,0)
integer array[1..n] granted := (1..n,0)
integer myNum := 0
boolean inCS := false
```

¿Lo tengo?

el token

veces he pedido entrar

¿Estoy dentro?

```
process MAIN
     while true
P1:       SNC
P2:       if NOT haveToken
P3:           myNum := myNum + 1
P4:           for all other nodes N
P5:              send(request,N,myID,myNum)
P6:           receive(token,granted)
P7:           haveToken := true; inCS := true
P8:       inCS := true
P9:       SC
P10:      granted[myID] := myNum
P11:      inCS := false
P12:      sendToken()
```