Lesson 10

Distributed Mutual Exclusion

Ch 10 [BenA 06]

Distributed System

Distributed Critical Section

Ricart-Agrawala

Token Passing Ricart-Agrawala

Token Passing Neilsen-Mizuno

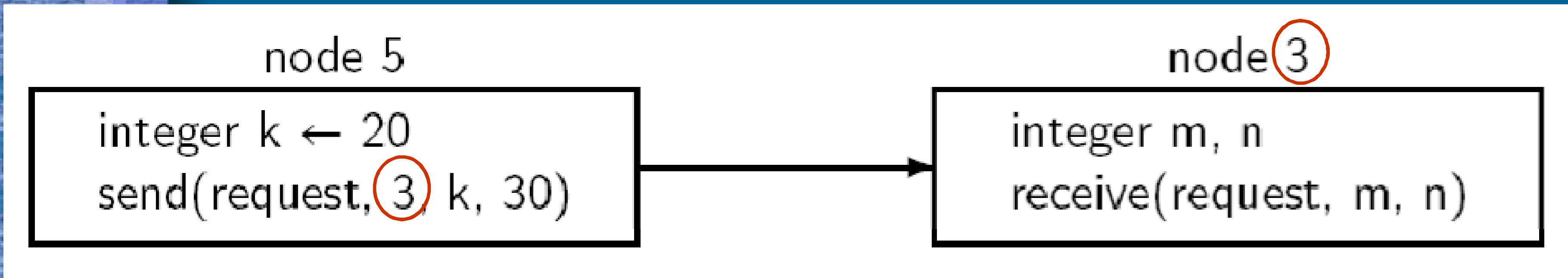
(Generic) Distributed System

- Nodes have processes
- Communication channels between nodes
  - Each node connected to every other node
    - Two-way channel
  - Reliable communication channels
    - Provided by network layer below
    - Messages are not lost
    - Messages processed concurrently with other computations (e.g., critical sections)
  - Nodes do not fail
- Requirements reduced later on
  - courses on distributed systems topics

Unrealistic assumptions? Not really…

(Generic) Distributed System

- Processes (nodes) communicate with (asymmetric) messages
  - Message arrival order is not specified
  - Transmission times are arbitrary, but finite
  - Message (header) does not include send/receiver id
  - Receiver does not know who sent the message
    - Unless sender id is in the message itself



Distributed Processes

- Sender does not block
- Receiver blocks (suspended wait) until message of the proper type is received
- Atomicity problems in each node is not considered here
  - Solved with locking, semaphores, monitors, …
- Message receiving and subsequent actions are considered to be atomic actions
  - Atomicity within each system considered solved

Distributed Critical Section Problem

- Processes within one node
  - Problem solved before
- Processes in different nodes
  - More complex
- State
  - Control pointer (CP, PC, program counter)
  - Local and shared variable values
  - Messages
    - Messages, that have been sent
    - Messages, that have been received
    - Messages, that are on the way
      - Arbitrary time, but finite!

Where are these?

Two Approaches for Crit. Section

- A) Ask everybody for permission to see, if it is my turn now
  - Lots of questions/answers
- B) I'll wait until I get the token, then it is my turn
  - Pass the token to next one (which one?), or keep it?
  - Wait until I get the token
  - Token (turn) goes around all the time
    - Moves only when needed?
- Both approaches have advantages/disadvantages
  - Who is "everybody"? How do I know them?
  - How do I know who has the token?
  - What if node/network breaks down?
  - What if token is lost?

Do not worry now about the token getting lost …

Ricart-Agrawala for Distributed Mutex

G. Ricart

A. K. Agrawala

- Distributed Mutex, 1981 (Lamport, 1978)
- Modification of Bakery algorithm with ticket numbers
- Idea
  - Must know all other processes/nodes competing for CS
  - Choose own ticket number, "larger than previous"
  - Send it to everybody else
  - Wait until permission from everybody else
    - Exactly one will always get permission from everybody else?
    - All others will wait
  - Do your CS
  - Give CS permission to everybody who was waiting for you

mutex,
no deadlock,
no starvation?

Algorithm 10.1: Ricart-Agrawala algorithm (outline)

integer myNum ← 0
set of node IDs deferred ← empty set

main application process, needs distr mutex

```
p1:  non-critical section
p2:  myNum ← chooseNumber
p3:  for all other nodes N
p4:     send(request, N, myID, myNum)
p5:  await reply's from all other nodes
p6:  critical section
p7:  for all nodes N in deferred
p8:     remove N from deferred
p9:     send(reply, N, myID)
```

receive server process, runs concurrently all the time

```
     integer source, reqNum
p10: receive(request, source, reqNum)
p11: if reqNum < myNum
p12:    send(reply,source,myID)
p13: else add source to deferred
```

not trivial!

Each one answers only when it is safe. Reply needs no content.

all those waiting for my permission

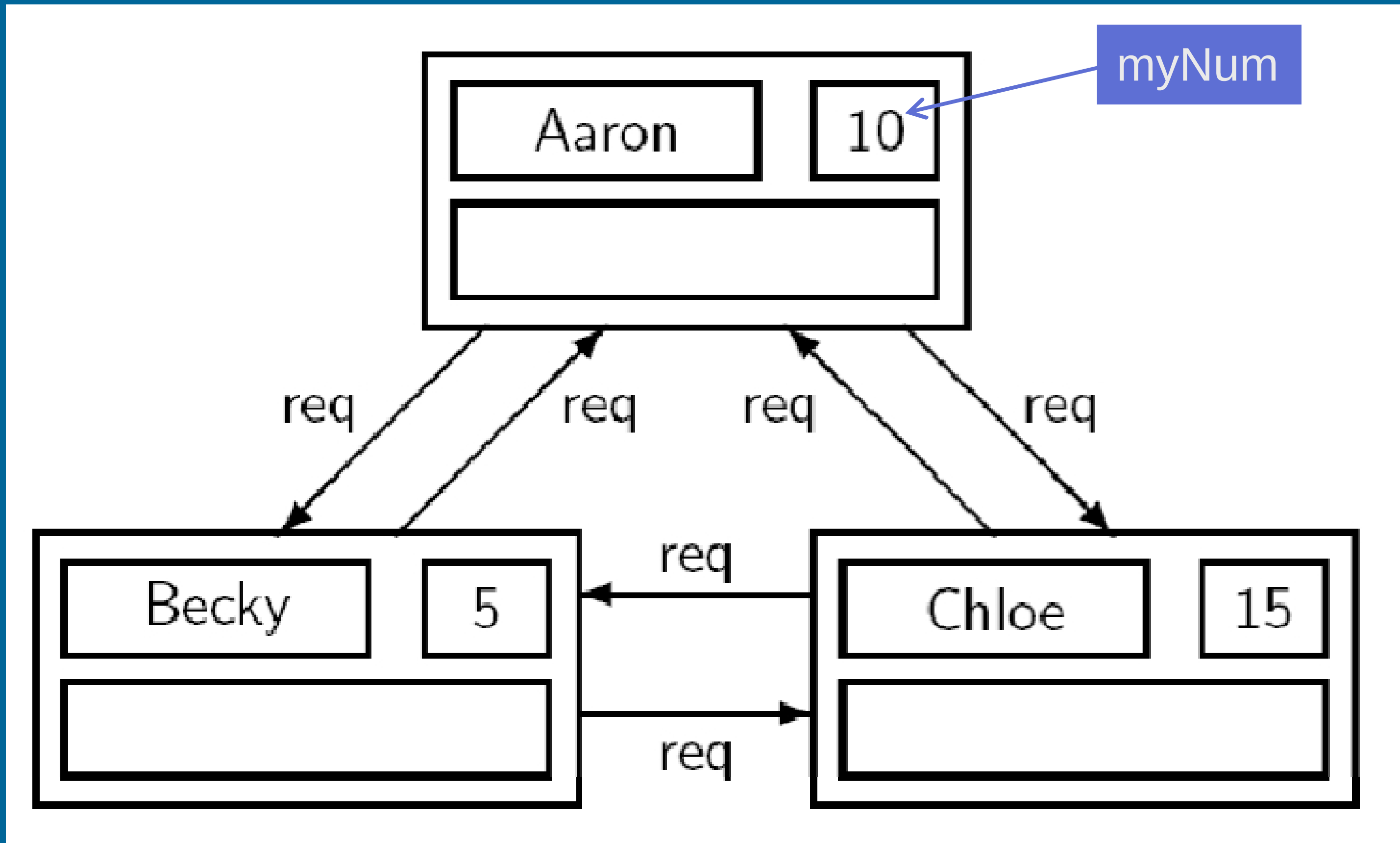
most recent myNum

make these wait by not sending reply

local mutex control?

Ricart-Agrawala Example

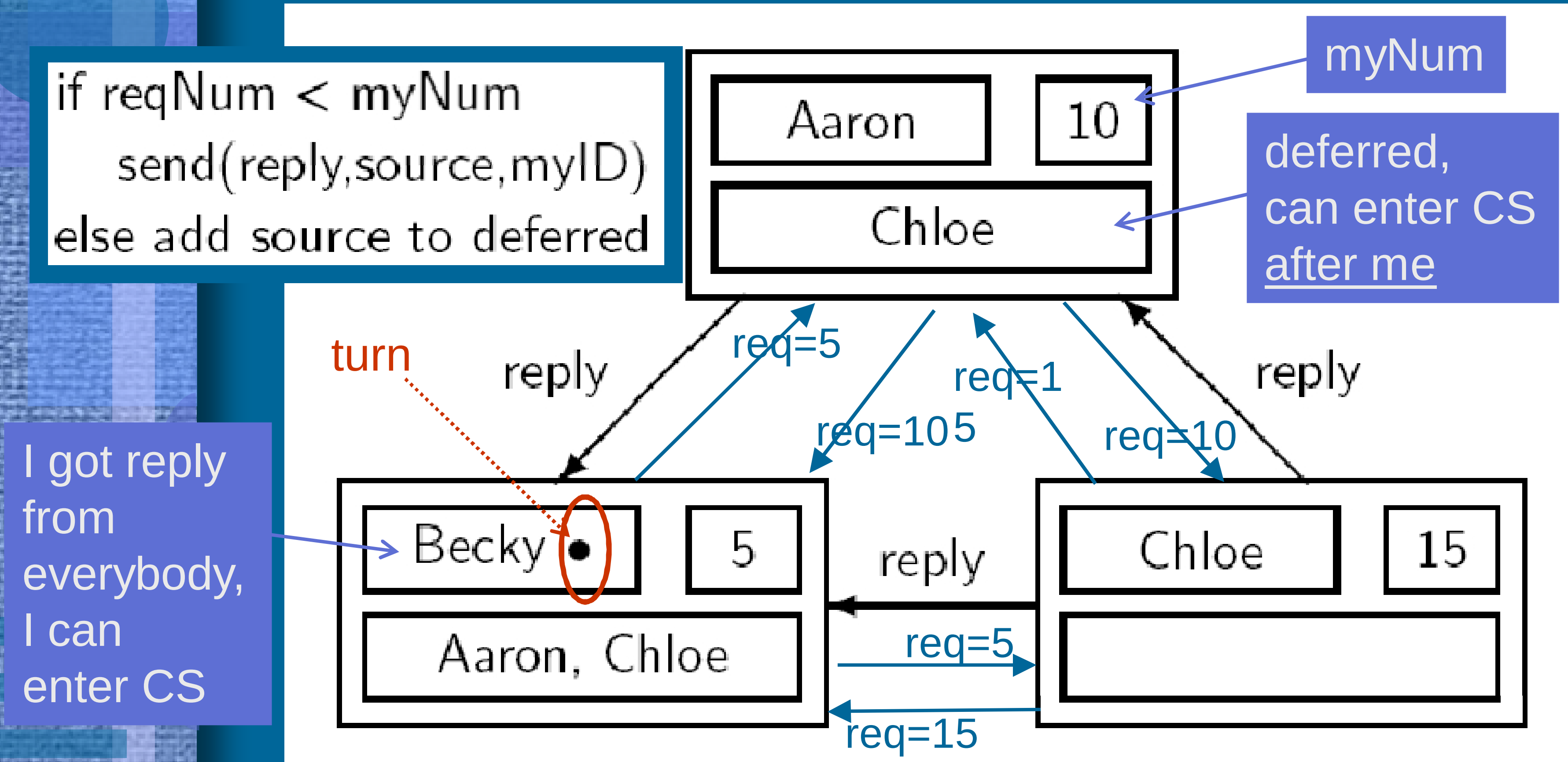
- 3 processes, <u>each</u> trying to enter CS concurrently
  - No status information needed on who had CS last



Ricart-Agrawala Example (contd)

- Receive process runs at each node
  - What if Aaron's *receive* completes 1st? Last? Becky's? not yet?

```
if reqNum < myNum
    send(reply,source,myID)
else add source to deferred
```

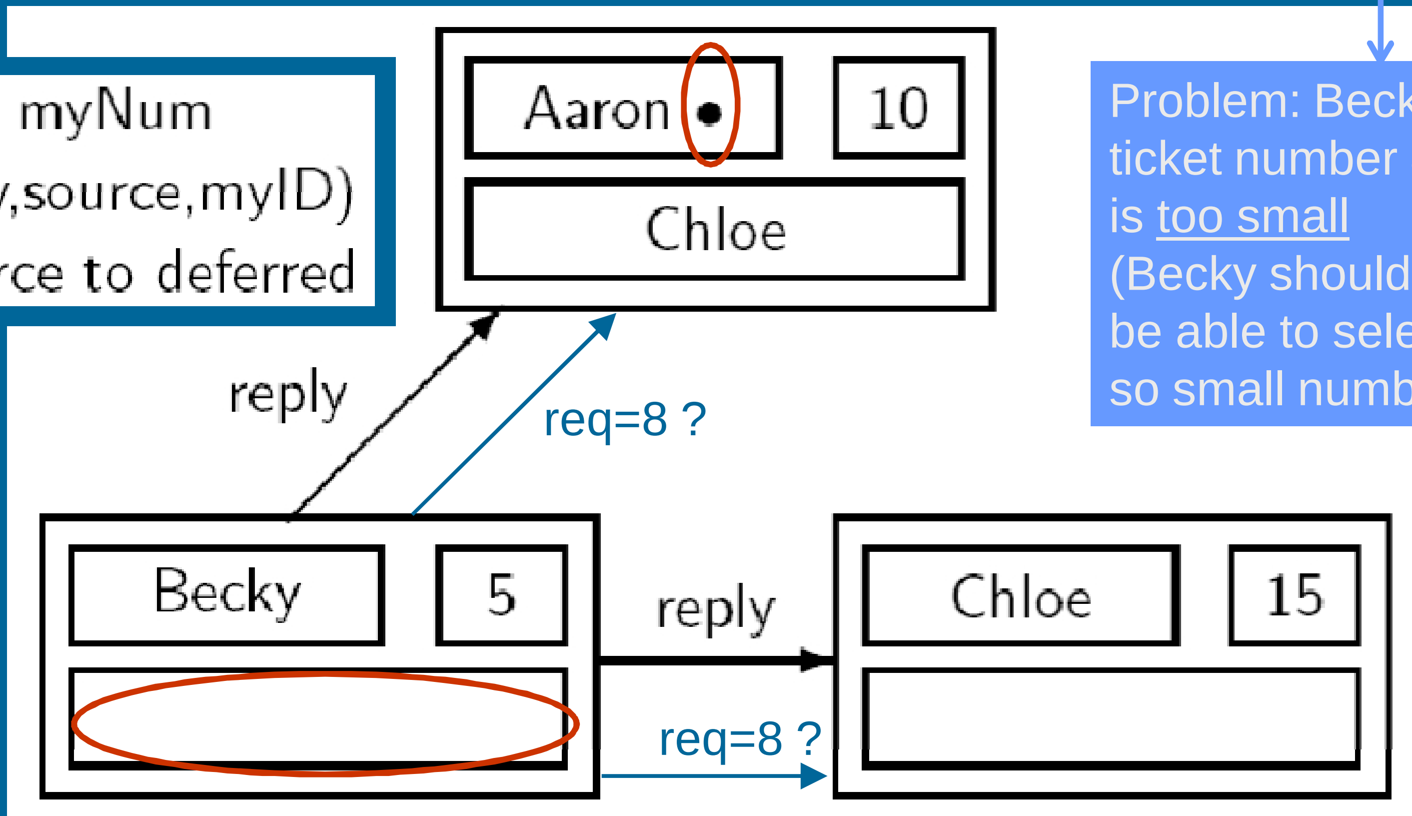


- Distributed virtual queue:

Ricart-Agrawala Example (contd)

- Becky executes CS, and then sends deferred replies to Aaron & Chloe
- Aaron has now replies from everybody, and it can enter CS
- What if Becky now selects ticket number 8, and requests CS?
  - Aaron's and Chloe's *receive* will both reply immediately? Ouch!

```
if reqNum < myNum
    send(reply,source,myID)
else add source to deferred
```



Problem: Becky's ticket number 8 is <u>too small</u> (Becky should not be able to select so small number)

How to select ticket numbers

- Select always larger one than you have <u>seen</u> before
  - Larger than your previous *myNum*
  - Larger than any *requestedNum* that <u>you have seen</u>
    - They all came before you, and you should not try to get ahead of them
- What if equal ticket numbers?
  - Fixed priority, based on node/process id numbers
  - Used only with equal ticket numbers to avoid deadlock
    - Just like in Bakery algorithm

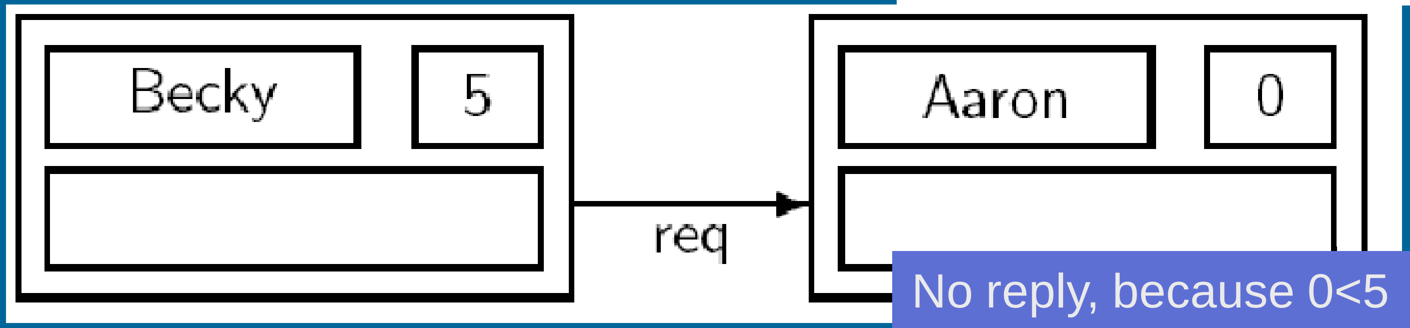
Discuss

Quiescent Nodes

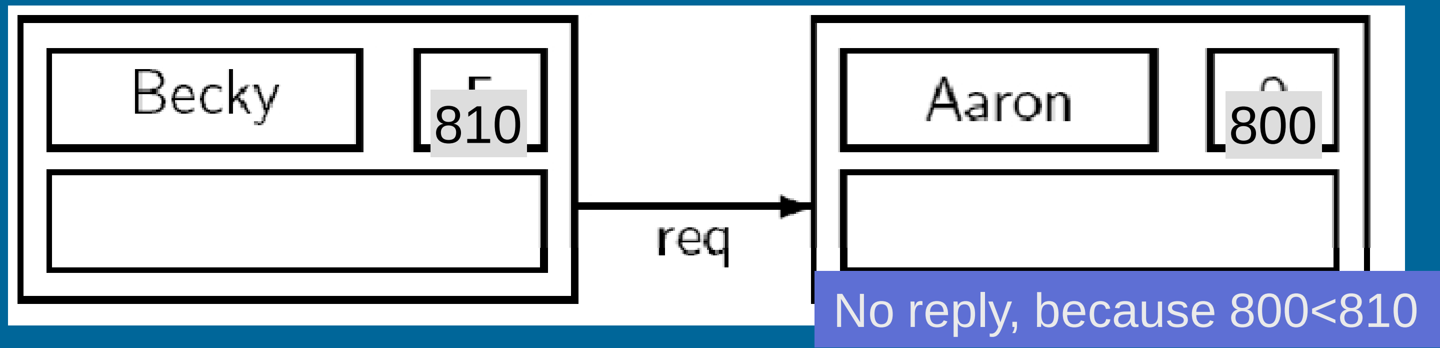
(hiljaiset solmut)

- Nodes that do not try to enter CS (but they could)
  - They are still listed in "all other nodes"
  - Problem with <u>initial value</u> of *myNum*
  - Initial value zero?

```
if reqNum < myNum
   send(reply,source,myID)
else add source to deferred
```



  - Initial value N > 0 ; tickets numbers eventually will reach it



  - Cure: *receive* checks for tickets numbers only if *main* wants CS

Algorithm 10.2: Ricart-Agrawala algorithm

```
        integer myNum ← 0
        set of node IDs deferred ← empty set
        integer highestNum ← 0
    Main
    loop forever
p1:     non-critical section
p2:     requestCS ← true
p3:     myNum ← highestNum + 1
p4:     for all other nodes N
p5:         send(request, N, myID, myNum)
p6:     await reply's from all other nodes
p7:     critical section
p8:     requestCS ← false
p9:     for all nodes N in deferred
p10:        remove N from deferred
p11:        send(reply, N, myID)
```

requestCS ← false

Algorithm 10.2: Ricart-Agrawala algorithm (continued)

```
    Receive
    integer source, requestedNum
    loop forever
p1:     receive(request, source, requestedNum)
p2:     highestNum ← max(highestNum, requestedNum)
p3:     if not requestCS or requestedNum ≪ myNum
p4:         send(reply, source, myID)
p5:     else add source to deferred
```

- Keep track of highest number seen
- What if one process asks for CS all the time?
- Same myNum OK?

Algorithm 10.2: Ricart-Agrawala algorithm (continued)

```
        Receive
        integer source, requestedNum
        loop forever
p1:        receive(request, source, requestedNum)
p2:        highestNum ← max(highestNum, requestedNum)
p3:        if not requestCS or requestedNum ≪ myNum
p4:           send(reply, source, myID)
p5:        else add source to deferred
```

original article

http://www.cc.gatech.edu/classes/AY2002/cs6210_fall/papers/MutualExForNetwork.pdf

- Mutex between main & receive?
  - Exact mutex boundaries?
- What to do when myNum overflows?
  - Restart everybody? When? How?
  - Fairness is not the problem, mutex is
- Correctness proofs
  - Mutex? No deadlock? No starvation?

Discuss

Token Based Algorithms

- Problems with permission based algorithms
  - Need permission from everybody (very many?)
    - At least everybody active
  - Inactive participants (those not wanting in CS) slow you down
    - Need reply from all of them!
    - Lots of synchronization even if only one tries to get into CS
    - →→→ Lots of communication (many messages)
- Token based algorithms
  - Have token, that is enough
    - No synchronization with everybody else needed
  - Get token, send token is simple
    - Communicate only with a few (fewer) nodes
    - Scalable?
  - Mutex is trivial, how about deadlock and starvation?

Ricart-Agrawala Token-Pass Ideas

- Send token to next one only when I know that someone wants it
  - o/w keep token until needed
- Keep local *requested* array for best knowledge for the most recent CS request times
  - Update this based on received CS request messages
- Keep local *granted* array, that has precise knowledge when each node actually was last granted CS
  - Update it only when CS granted
  - Pass it with token to next node
    - Only this *granted* array (with token) is exactly correct!
    - Other nodes have (slightly) old *granted* array

Algorithm 10.3: Ricart-Agrawala token-passing algorithm

```
boolean haveToken ← true in node 0, false in others
integer array[NODES] requested ← [0,...,0]
integer array[NODES] granted ← [0,...,0]
integer myNum ← 0
boolean inCS ← false
```

local data in node

distributed global data

sendToken

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

If no one else wants token, I will keep it

Ticket number for newest request for CS (that I know of)

Ticket number last time in CS

Receive server process, runs all the time

```
integer source, reqNum
loop forever
   receive(request, source, reqNum)
   requested[source] ← max(requested[source], reqNum)
   if haveToken and not inCS
      sendToken
```

Give also most recent *granted[]*

Algorithm 10.3: Ricart-Agrawala token-passing algorithm (continued)

Main application process, needs distr mutex

```
loop forever
   non-critical section
   if not haveToken
      myNum ← myNum + 1
      for all other nodes N
         send(request, N, myID, myNum)
      receive(token, granted)
      haveToken ← true
   inCS ← true
   critical section
   granted[myID] ← myNum
   inCS ← false
   sendToken
```

If I have token, no delays.

Request token from everybody
Very many messages?

Just one very
large message?

Wait until
token
received

Update
one field

Only if someone wants it!
Send *granted* also.

- Mutex?
- No deadlock?
- No starvation?
  - "some" in sendToken?
- Scalable?
- Overflows?

Discuss

	Aaron	Becky	Chloe	Danielle	Evan
requested	4	3	0	5	1
granted	4	2	2	4	1

Chloe's view

Algorithm 10.3: Ricart-

Main application proc

```
loop forever
  non-critical section
  if not haveToken
    myNum ← myNum + 1
    for all other nodes N
      send(request, N, myID, myNum)
    receive(token, granted)
    haveToken ← true
  inCS ← true
  critical section
  granted[myID] ← myNum
  inCS ← false
  sendToken
```

Request token from everybody
Very many messages?

Wait until token received

Update one field

Only if someone wants it!
Send *granted* also.

- Can Chloe be 3rd time in CS?
- Who wants CS now?
- If Chloe has token, and is in non-CS, what happens next?
- If Chloe has token and is in CS, what happens next?
- Why is Chloe's own requested[i] zero?
- Could Becky have kept the token since last use?

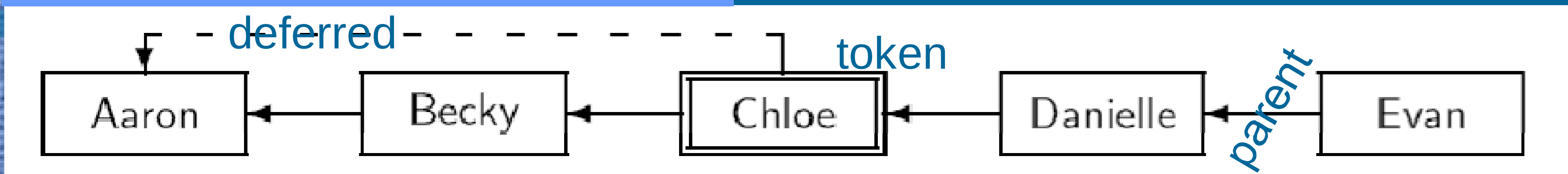
Neilsen-Mizuno
Token Based Algorithm

- Rigart-Agrawala: token carries queue of waiting processes
  - Token can be very large, which may be problematic
- Neilsen-Mizuno: virtual tree structure within the nodes implements the queue

virtuaalinen virittävä (viritys-) puu

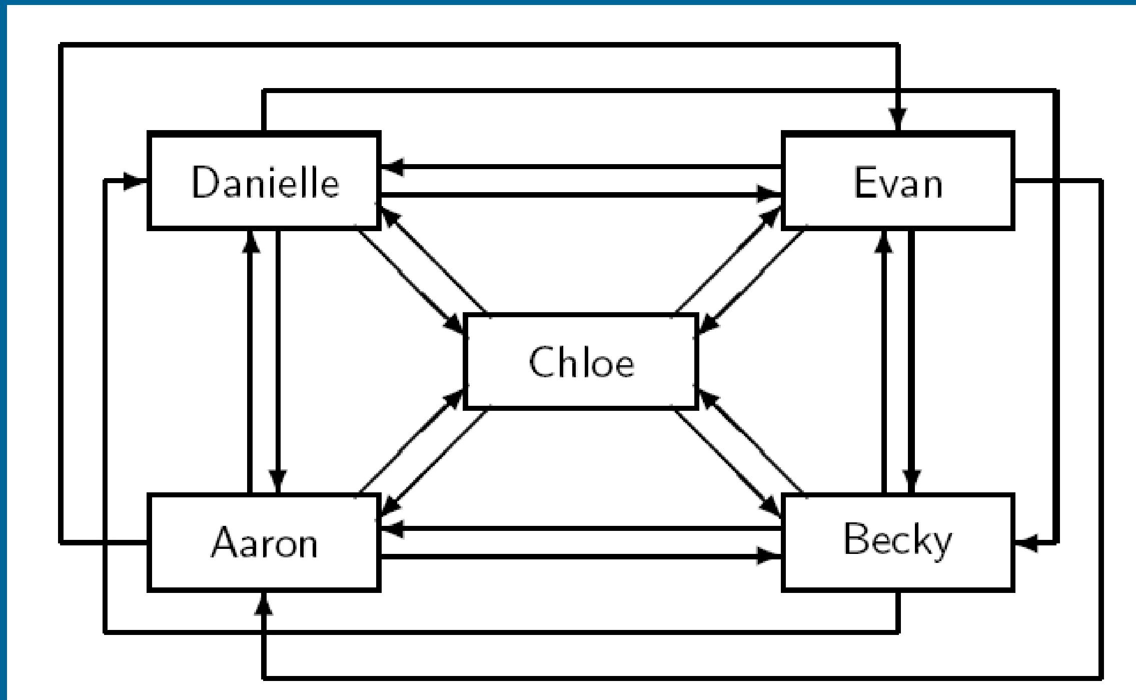
  - Algorithm utilizes <u>*virtual spanning tree*</u> of nodes
    - *Spanning tree:* all nodes linked as a tree, no cycles
  - Simple *token* indicates "turn" for critical section
  - *Parent* link points to the <u>direction</u> of last in line for CS
    - Parent == 0: node may have token and is last in line for CS
  - *Deferred* link points to <u>next</u> in line for CS

Chloe has token, Aaron is waiting for it



Neilsen-Mizuno Example

- Fully connected nodes
- Chloe is in CS
- No one waits for CS



Neilsen-Mizuno Example (contd)

- Chloe has token, nobody waits for it

deferred

- Aaron requests CS

sender originator

  - Sends msg=(req, Aaron, Aaron) on parent link
  - Removes himself from parent spanning tree

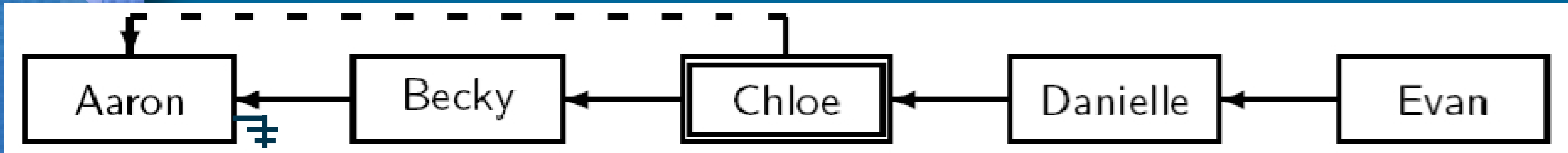
- Becky receives msg, and forwards the request “upward”
  - Sends msg=(req, Becky, Aaron) to Chloe
  - Moves to new parent spanning tree, points to Aaron
    - Aaron is now last to request CS

Neilsen-Mizuno Example (contd)

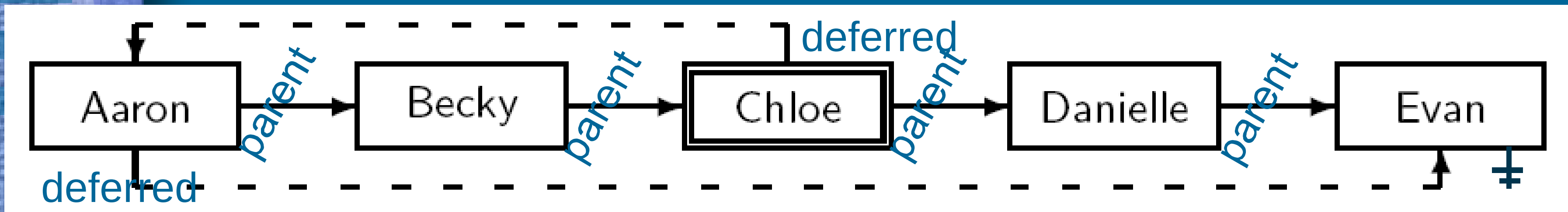
- Chloe receives msg (req, Becky, Aaron)
  - Chloe in CS, sets deferred field to Aaron and sets parent field to Becky
    - Chloe was (also) last in line for CS

  - When Chloe completes CS, she will pass token to Aaron
    - Token transferred directly to the next process in line for critical section (if any)
      - Just token is passed, <u>no big array with it</u>

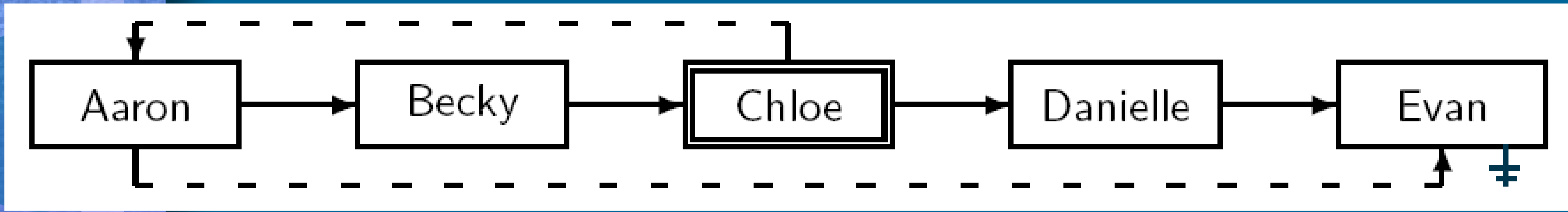
Neilsen-Mizuno Example (contd)



- Chloe still has CS, Evan wants CS
  - Sends (req, Evan, Evan) to Danielle
  - Danielle sends (req, Danielle, Evan) to Chloe
  - Chloe sends (req, Chloe, Evan) to Becky
  - Becky sends (req, Becky, Evan) to Aaron
  - Aaron makes a *deferred* link to Evan



Neilsen-Mizuno Example (contd)



- Chloe completes CS, passes token to Aaron

- Aaron completes CS, passes token to Evan

- Evan completes CS, keeps token

Algorithm 10.4: Neilsen-Mizuno token-passing algorithm

```
integer parent ← (initialized to form a tree)
integer deferred ← 0
boolean holding ← true in the root, false in others
```

Main

```
      loop forever
p1:      non-critical section
p2:      if not holding
p3:         send(request, parent, myID, myID)
p4:         parent ← 0
p5:         receive(token)
p6:      holding ← false
p7:      critical section
p8:      if deferred ≠ 0
p9:         send(token, deferred)
p10:        deferred ← 0
p11:     else holding ← true
```

Target node, not part of message

holding = have token, <u>not</u> in CS

mark latest request for CS

wait here until permission for CS obtained

someone wants the CS next

Algorithm 10.4: Neilsen-Mizuno token-passing algorithm

Receive (runs concurrently with main, mutex problems solved…)

```
      integer source, originator
      loop forever
p12:    receive(request, source, originator)
p13:    if parent = 0                               last in queue
p14:       if holding                               have token, not in CS
p15:          send(token, originator)
p16:          holding ← false
p17:       else deferred ← originator               place new req last in queue
p18:    else send(request, parent, myID, originator)   forward request
p19:    parent ← source                             update direction for last request
```

Discuss

Ricart-Agrawala vs. Neilsen-Mizuno

- Number of messages needed?
- Size of messages?
- Size of data structures in each node?
- Behaviour with heavy load?
  - Many need CS at the same time
- Behaviour with light load?
  - Requests for CS do not come often
  - Usually only one process requests CS at a time

Other Distributed Mutex Algorithms

- Other token-based algorithms
  - Token ring: token moves all the time
  - Lots of token traffic even when no CS requests
- Centralized server
  - Simple, not very many messages
  - Not scalable, may become bottleneck
- Give up unrealistic assumptions
  - Nodes may fail
  - Messages may get lost, token may get lost
- See other courses → Courses on distributed systems topics (hajautetut järjestelmät)

Summary

- Distributed critical section is hard, avoid it
  - Use centralized solutions if possible?
- Permission based solutions
  - Ricart-Agrawala – ask everyone
- Token based solutions
  - Ricart-Agrawala – centralized state in granted[]
  - Neilsen-Mizuno – queue kept in spanning tree
- There are other algorithms
- How do they scale up?