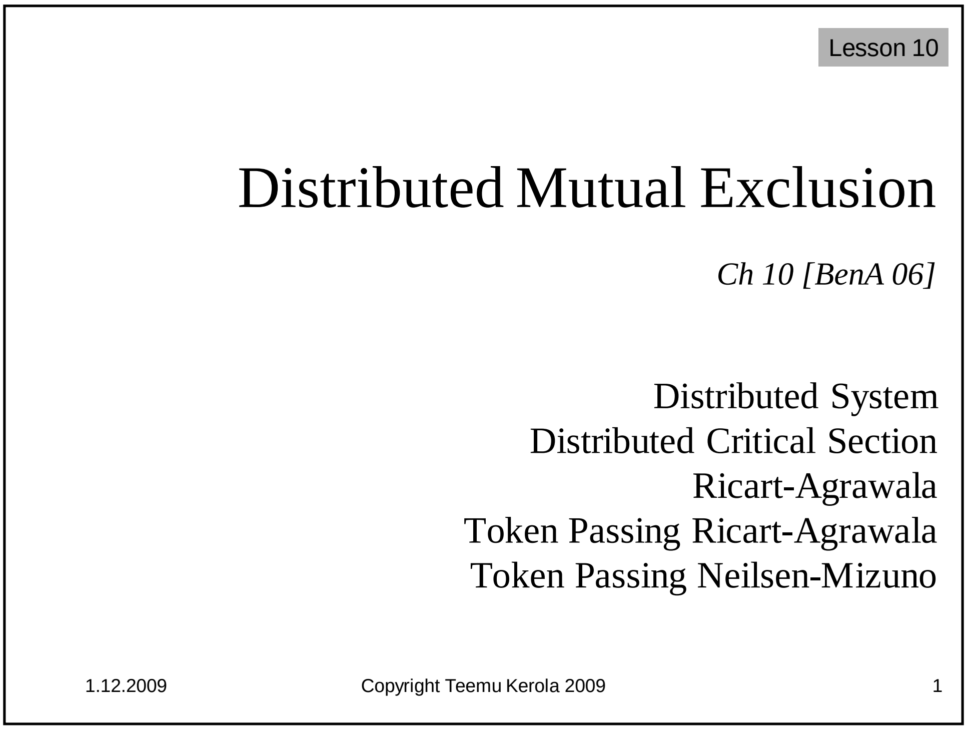


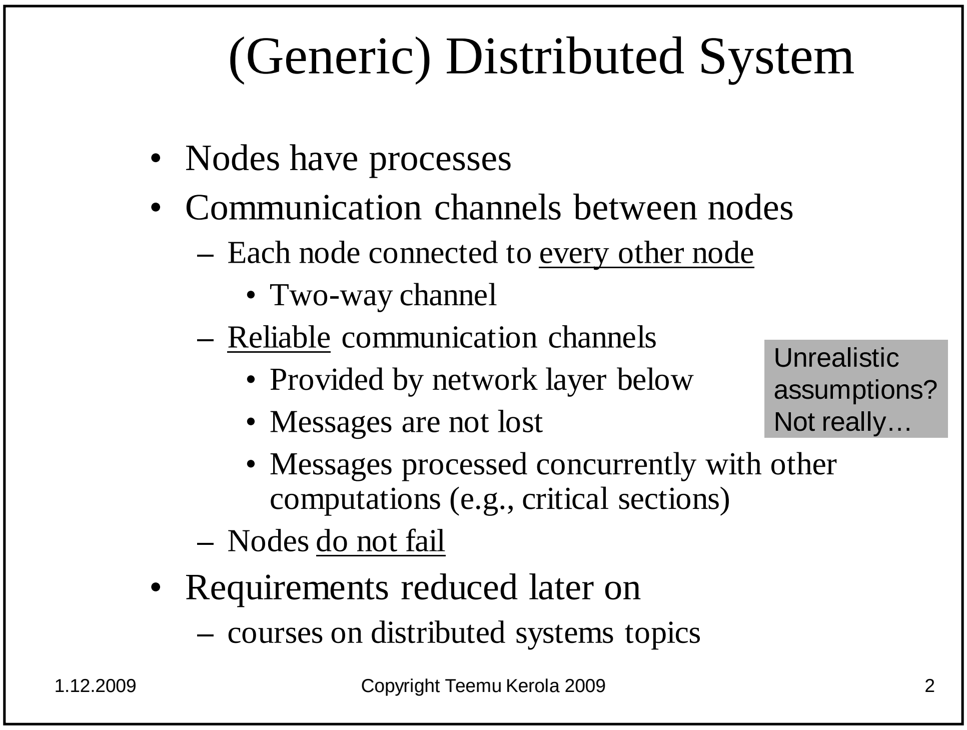
Lesson 10

Distributed Mutual Exclusion

Ch 10 [BenA 06]

Distributed System
Distributed Critical Section
Ricart-Agrawala
Token Passing Ricart-Agrawala
Token Passing Neilsen-Mizuno

1.12.2009 Copyright Teemu Kerola 2009 1

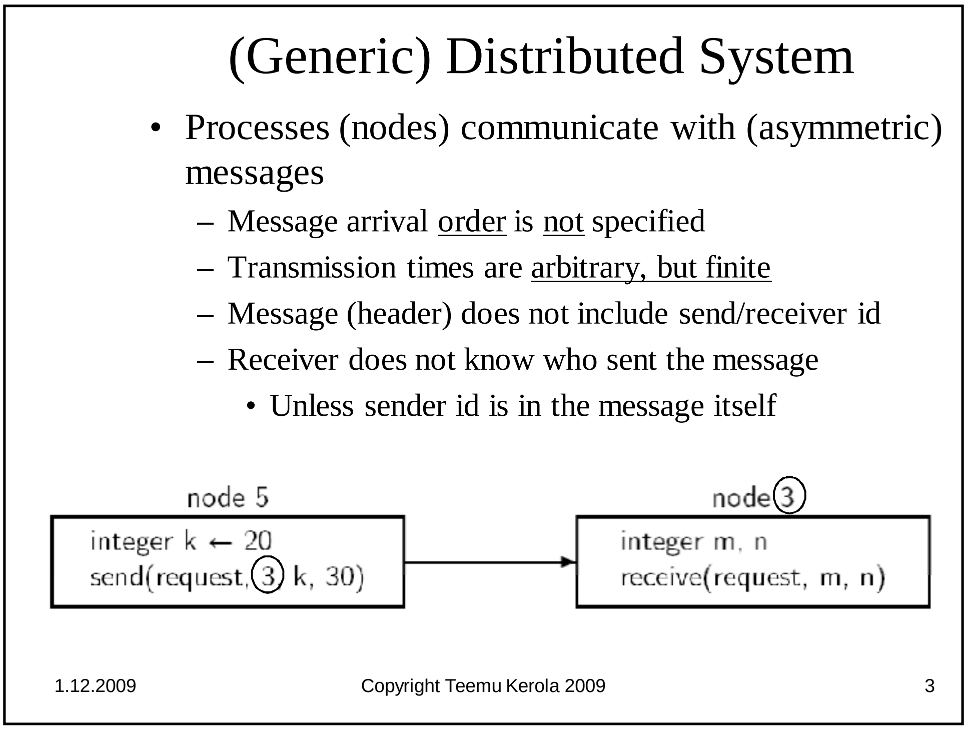


(Generic) Distributed System

- 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…

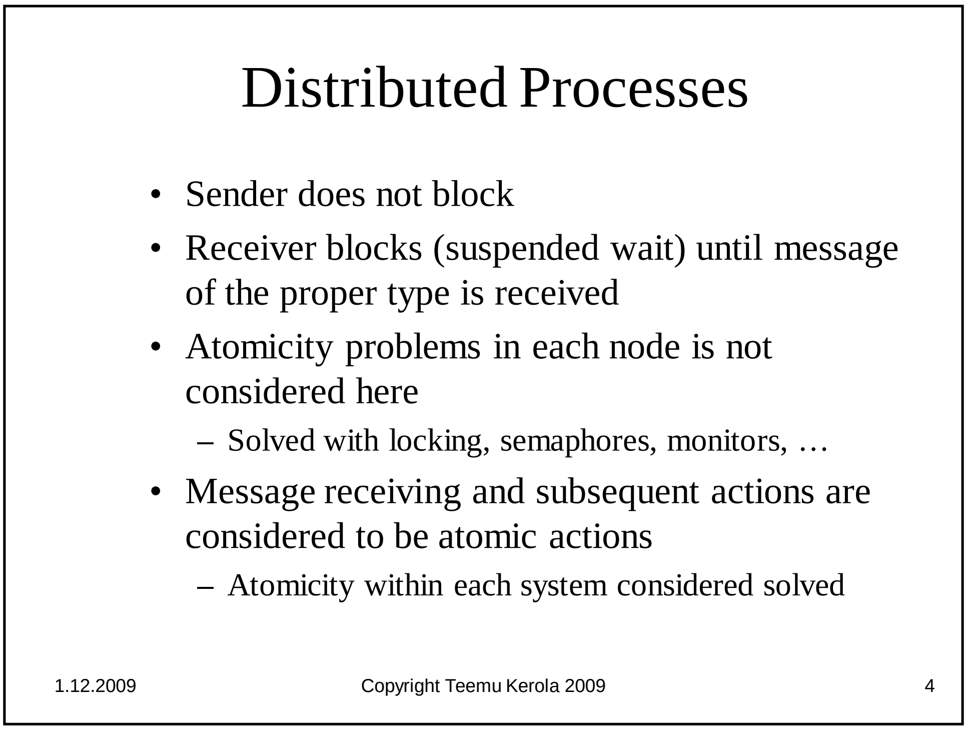
1.12.2009 Copyright Teemu Kerola 2009 2



(Generic) Distributed System

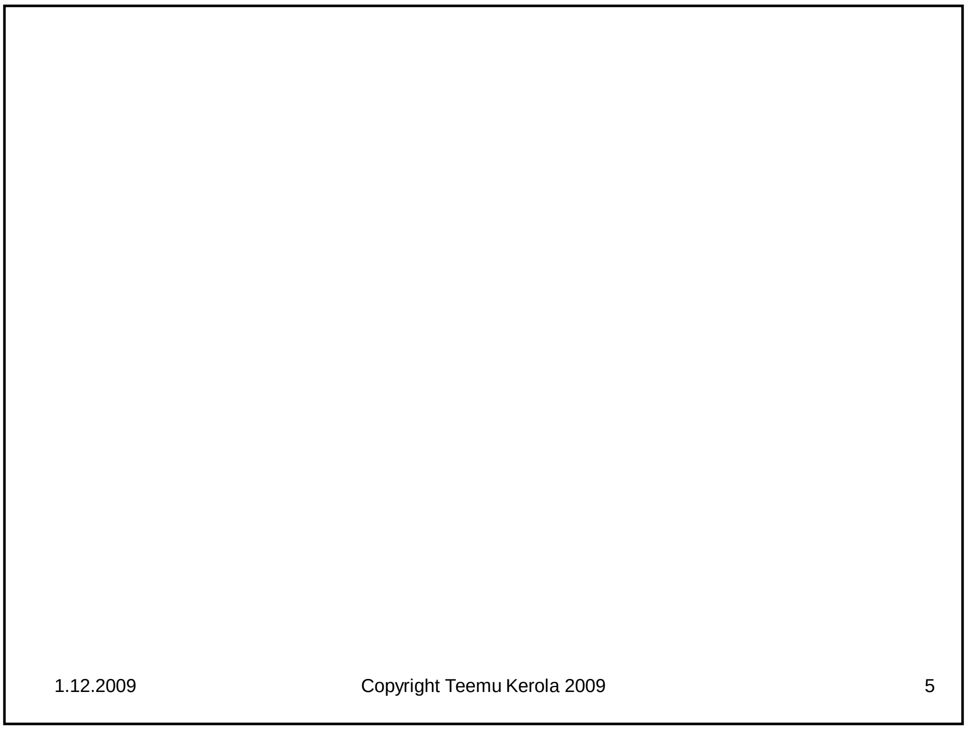
- Processes (nodes) communicate with (asymmetric) messages
  - Message arrival order is not specified
  - Transmission times are arbitrary, but finite
  - Message (header) does not include send/receiver id
  - Receiver does not know who sent the message
    - Unless sender id is in the message itself

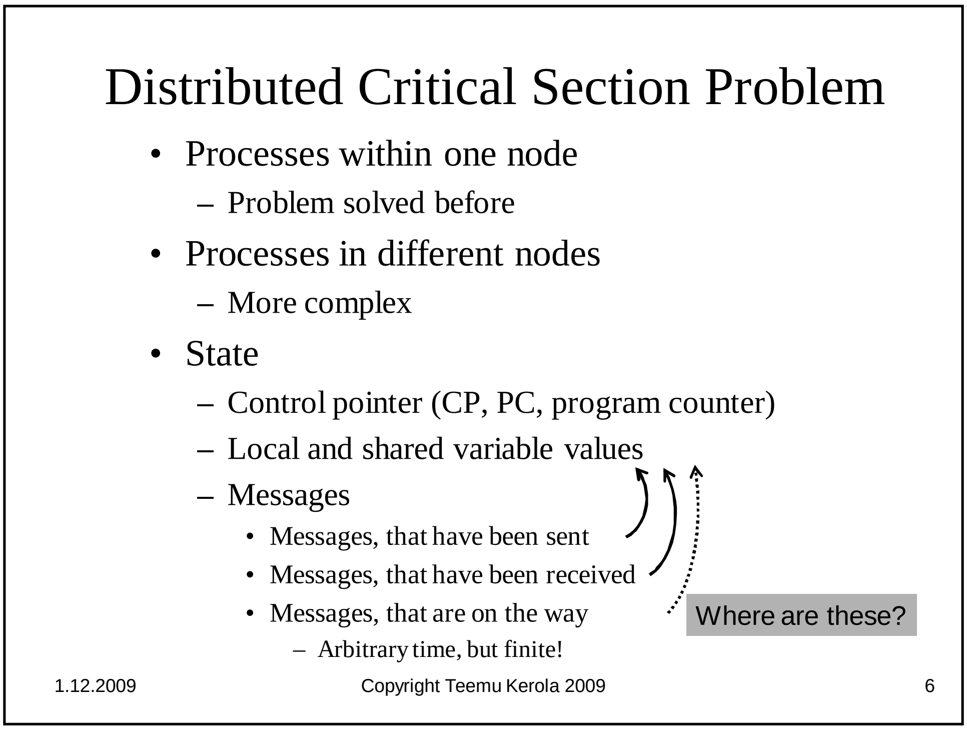
```
node 5                                  node 3
integer k ← 20                 ───▶     integer m, n
send(request, 3  k, 30)                 receive(request, m, n)
```



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

- Ask everybody for <u>permission</u>, if it is my turn now
  - Lots of questions/answers
- I'll wait until I get the <u>token</u>, then it is my turn
  - Pass the token to next one (which one?)
  - 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?
  - What if someone does not talk to me?
  - What if node/network breaks down?
  - What if token is lost?

Do not worry now about the token getting lost …

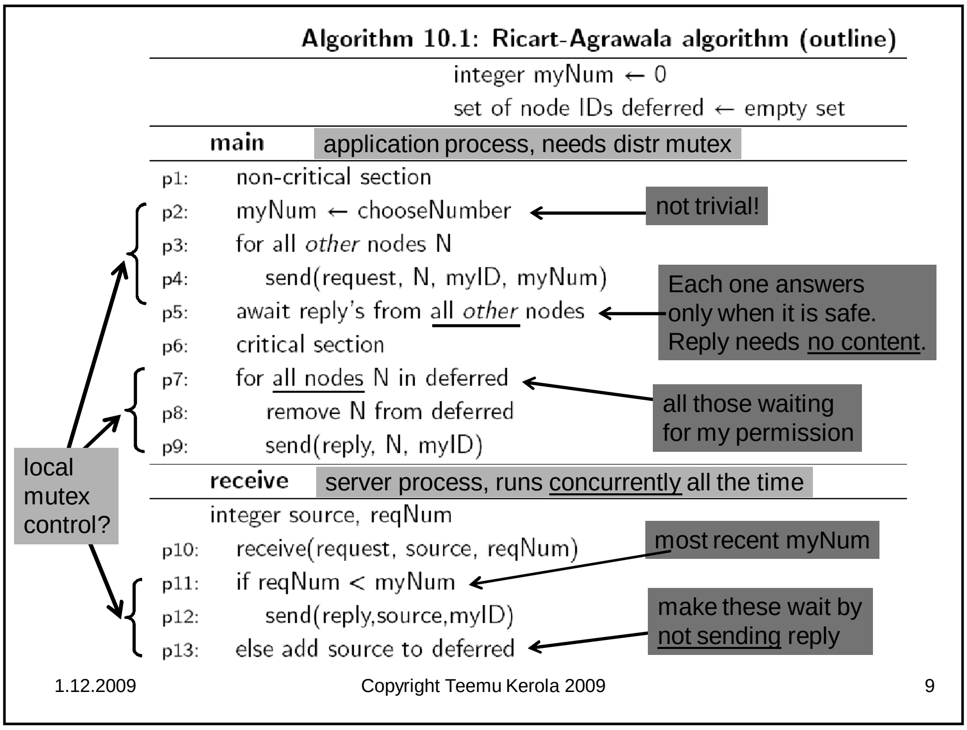
Ricart-Agrawala for Distributed Mutex

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 <u>everybody</u> else
    - Exactly one will always get permission from <u>everybody else</u>?
    - All others will wait
  - Do your CS
  - Give CS permission to <u>everybody else</u> 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
    non-critical section
    myNum ← chooseNumber
    for all other nodes N
        send(request, N, myID, myNum)
    await reply's from all other nodes
    critical section
    for all nodes N in deferred
        remove N from deferred
        send(reply, N, myID)
receive
    integer source, reqNum
    receive(request, source, reqNum)
    if reqNum < myNum
        send(reply,source,myID)
    else add source to deferred
```

application process, needs distr mutex

not trivial!

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

all those waiting for my permission

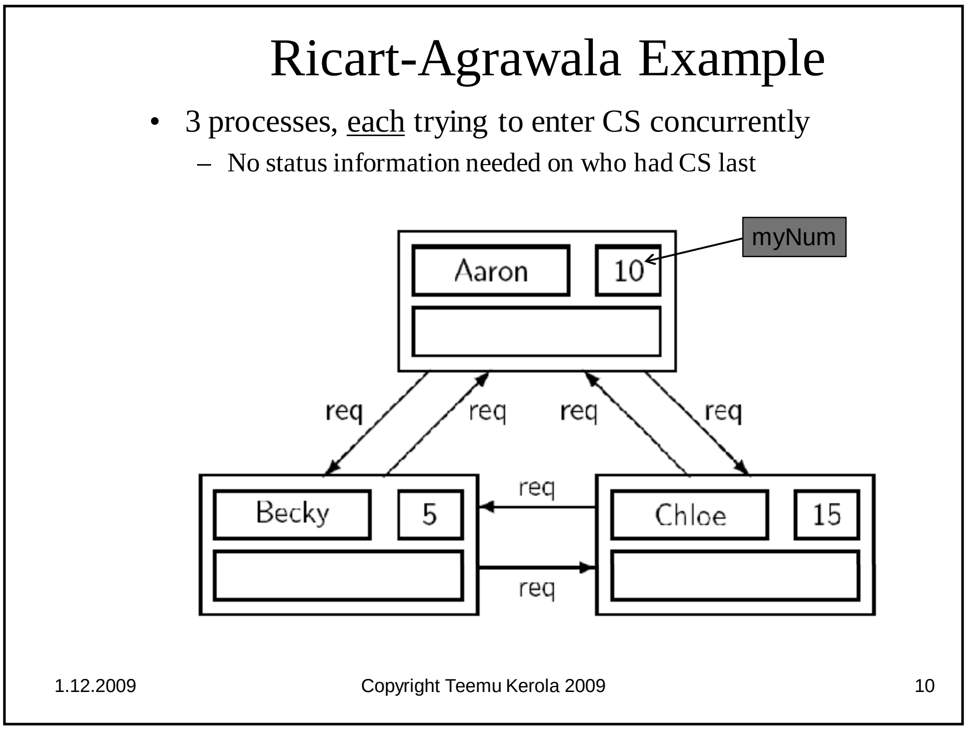
server process, runs concurrently all the time

most recent myNum

make these wait by not sending reply

local mutex control?

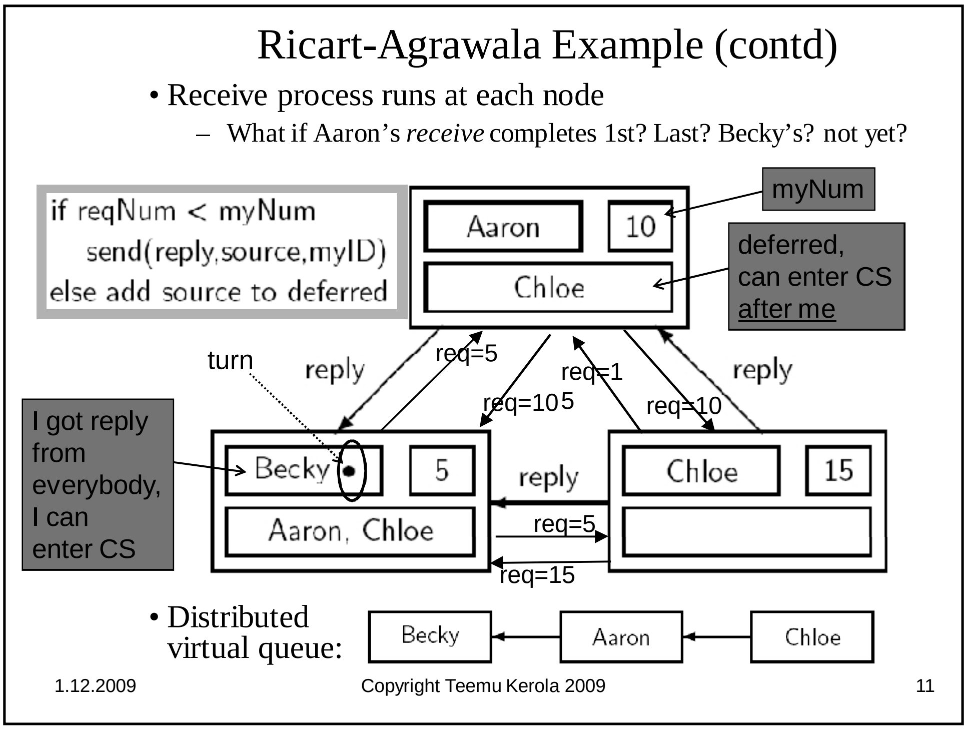
1.12.2009 Copyright Teemu Kerola 2009 9



Ricart-Agrawala Example

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

1.12.2009 Copyright Teemu Kerola 2009 10



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
```

Aaron 10

Chloe

myNum

deferred, can enter CS after me

turn

reply

req=5

req=1

req=105

req=10

reply

I got reply from everybody, I can enter CS

Becky ● 5

Aaron, Chloe

reply

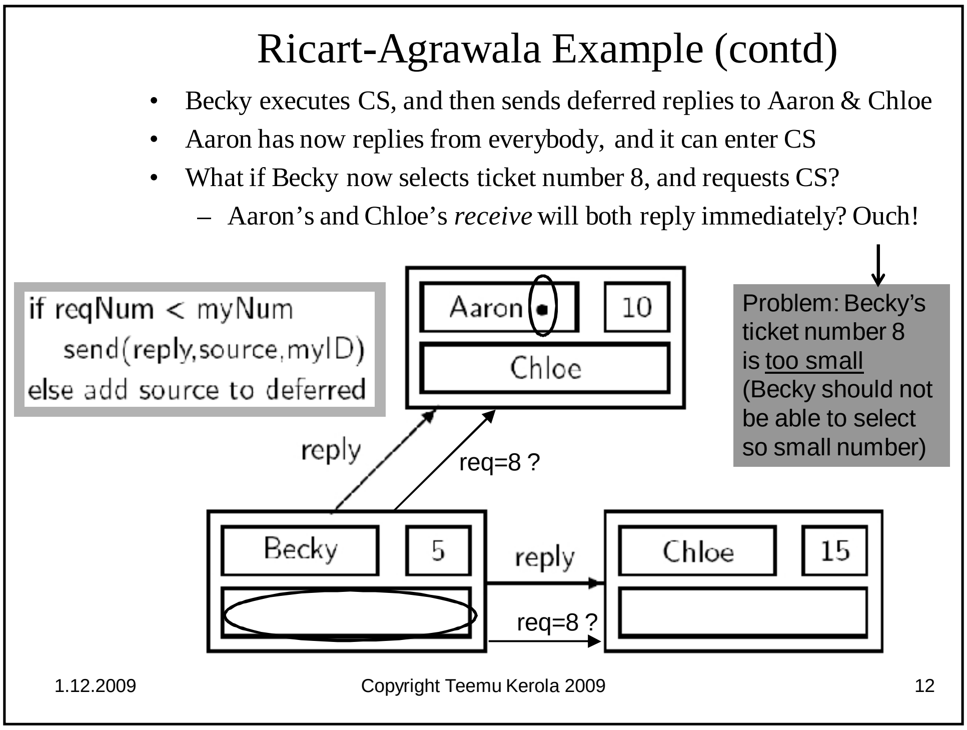
req=5

req=15

Chloe 15

- Distributed virtual queue: Becky ← Aaron ← Chloe

1.12.2009 Copyright Teemu Kerola 2009 11



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
```

Aaron ● 10

Chloe

Problem: Becky’s ticket number 8 is too small (Becky should not be able to select so small number)

reply

req=8 ?

Becky 5

reply

req=8 ?

Chloe 15

1.12.2009 Copyright Teemu Kerola 2009 12

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

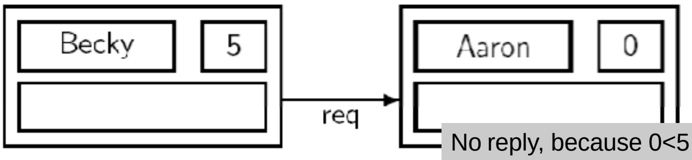
1.12.2009 Discussion A 13

Quiescent Nodes

(hiljaiset solmut)

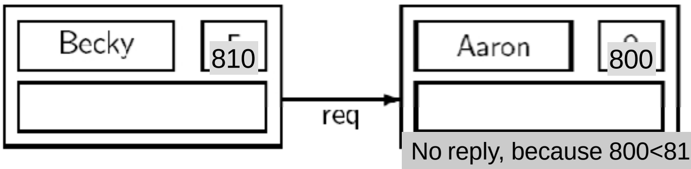
- Nodes that do not try to enter CS (but they could)
  - They are still listed in "all other nodes"
  - Problem with <u>initial value</u> of *myNum*
  - Initial value zero?

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



No reply, because 0<5

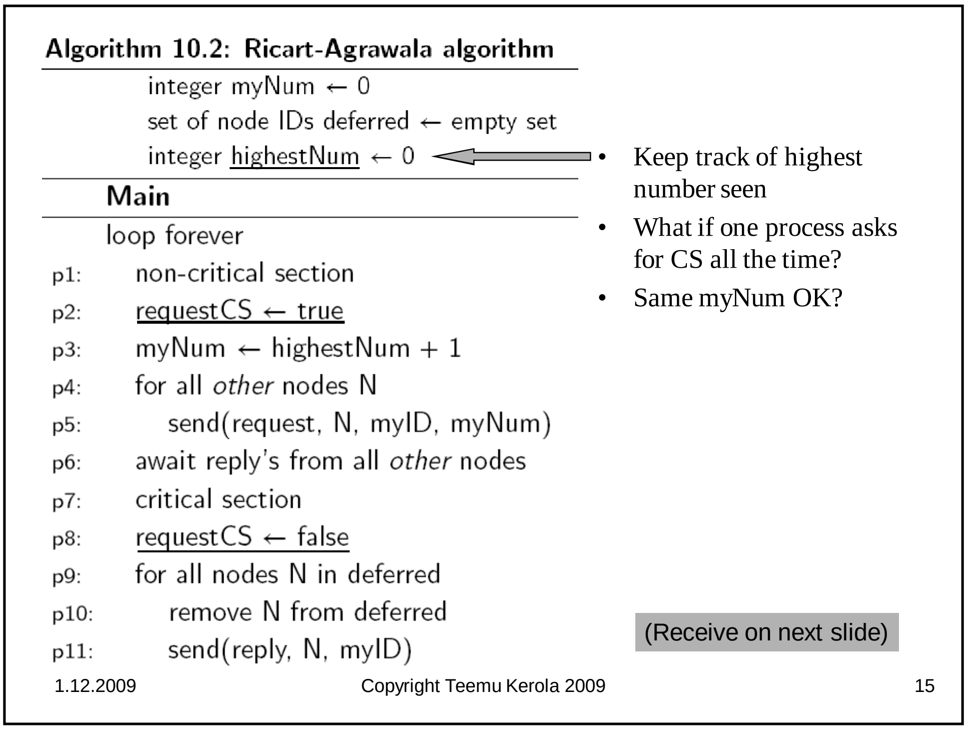
  - Initial value N > 0 ; tickets numbers eventually will reach it



No reply, because 800<810

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

1.12.2009 14



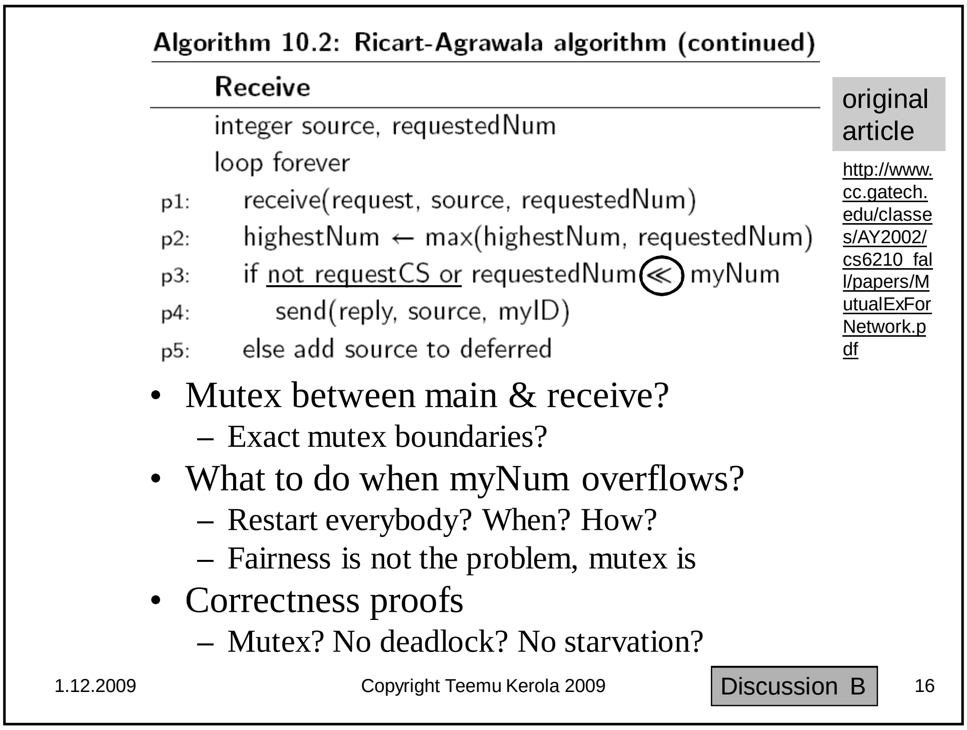
Algorithm 10.2: Ricart-Agrawala algorithm

```
          integer myNum ← 0
          set of node IDs deferred ← empty set
          integer highestNum ← 0
     Main
     loop forever
        non-critical section
        requestCS ← true
        myNum ← highestNum + 1
        for all other nodes N
           send(request, N, myID, myNum)
        await reply's from all other nodes
        critical section
        requestCS ← false
        for all nodes N in deferred
           remove N from deferred
           send(reply, N, myID)
```

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

(Receive on next slide)

1.12.2009 15



Algorithm 10.2: Ricart-Agrawala algorithm (continued)

```
     Receive
     integer source, requestedNum
     loop forever
        receive(request, source, requestedNum)
        highestNum ← max(highestNum, requestedNum)
        if not requestCS or requestedNum ≪ myNum
           send(reply, source, myID)
        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?

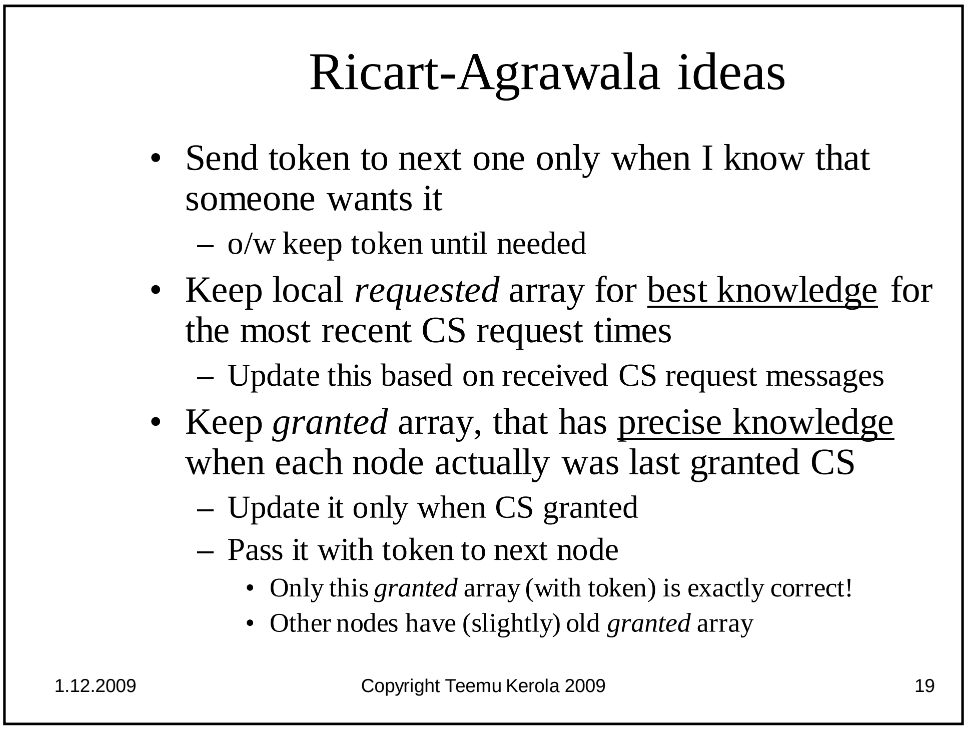
1.12.2009 Discussion B 16

1.12.2009 Copyright Teemu Kerola 2009 17

Token Based Algorithms

- Problems with permission based algorithms
  - Need permission from everybody (very many?)
  - 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?

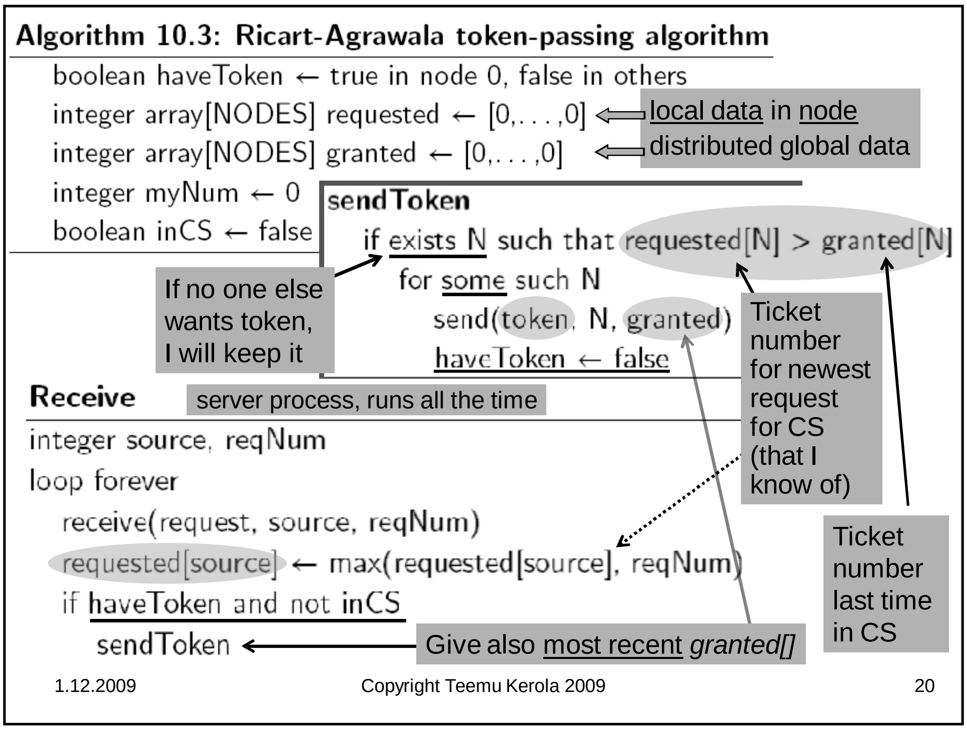
1.12.2009 Copyright Teemu Kerola 2009 18



Ricart-Agrawala 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 *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

1.12.2009 Copyright Teemu Kerola 2009 19



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

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

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

local data in node

distributed global data

If no one else wants token, I will keep it

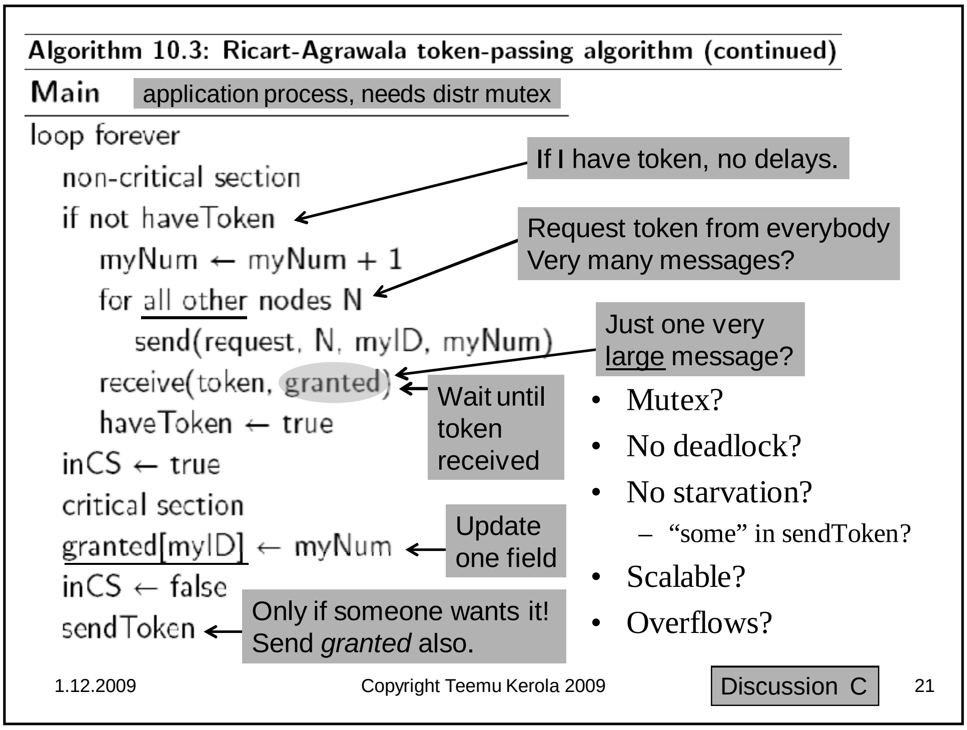
server process, runs all the time

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

Ticket number last time in CS

Give also most recent *granted[]*

1.12.2009 Copyright Teemu Kerola 2009 20



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?

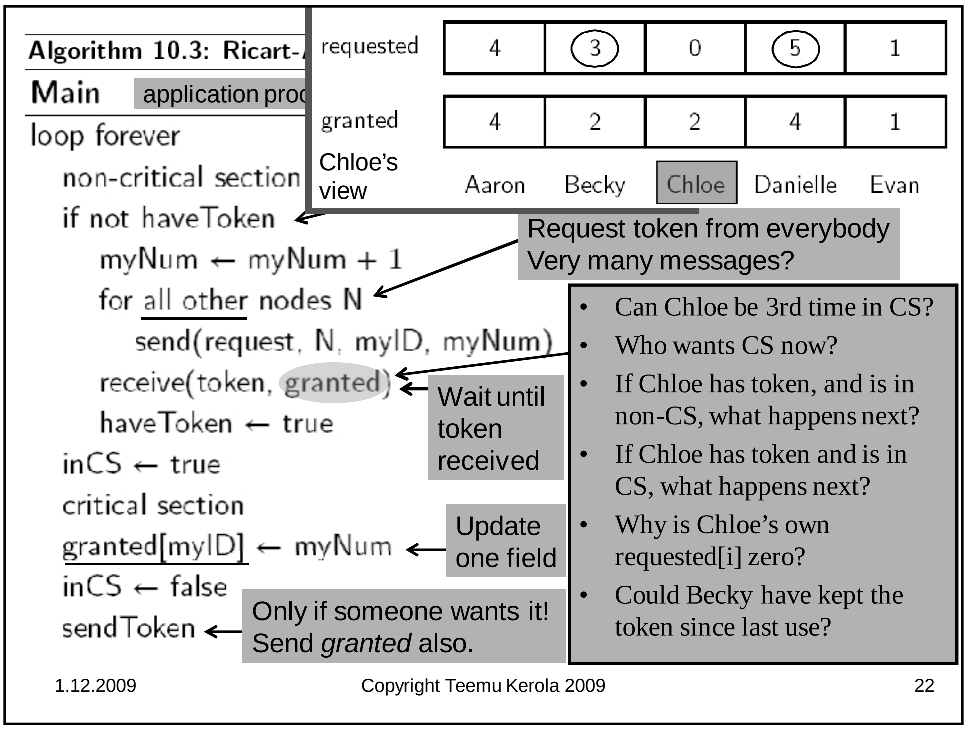
Wait until token received

Update one field

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

- Mutex?
- No deadlock?
- No starvation?
  - “some” in sendToken?
- Scalable?
- Overflows?

1.12.2009 Discussion C 21



Algorithm 10.3: Ricart-/

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

Chloe’s view

```
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?

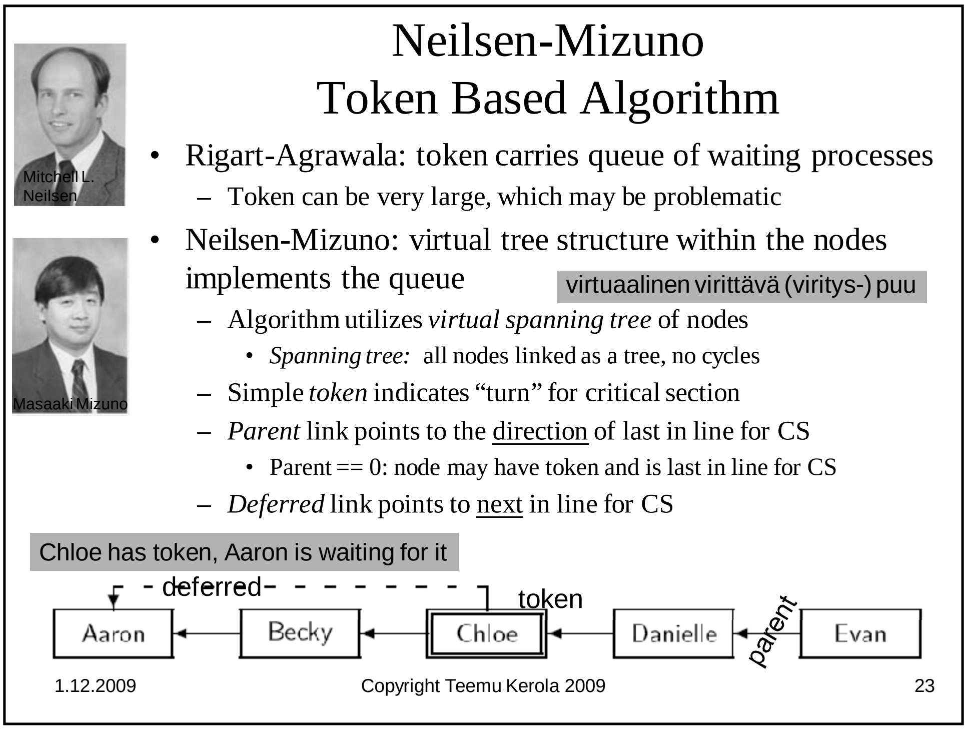
Wait until token received

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?

1.12.2009 22

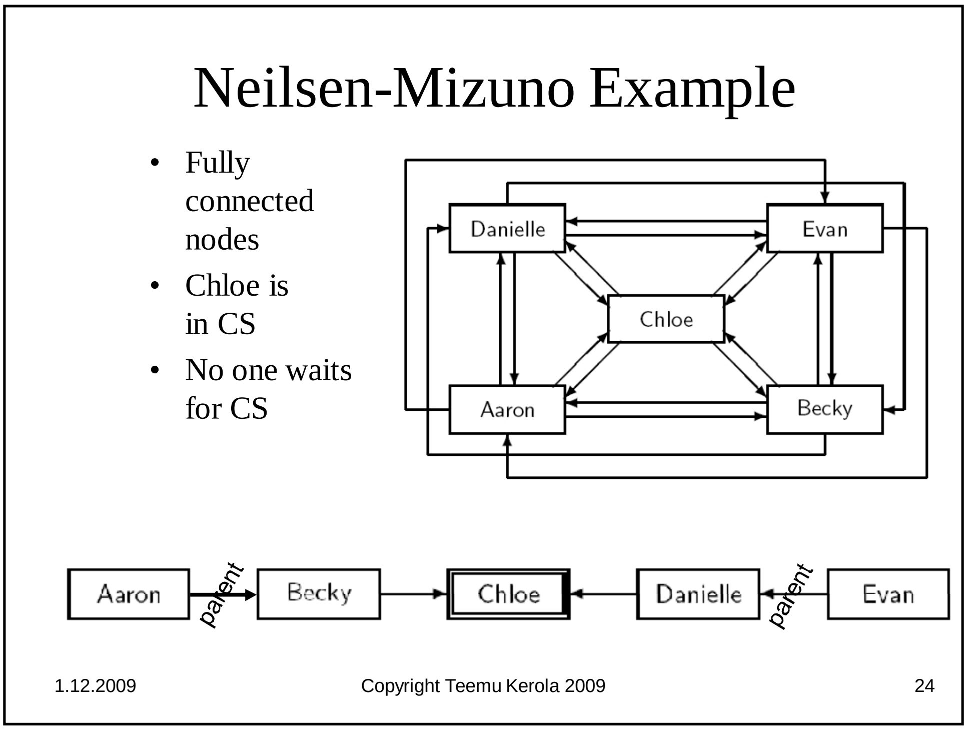


Neilsen-Mizuno Token Based Algorithm

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

Chloe has token, Aaron is waiting for it

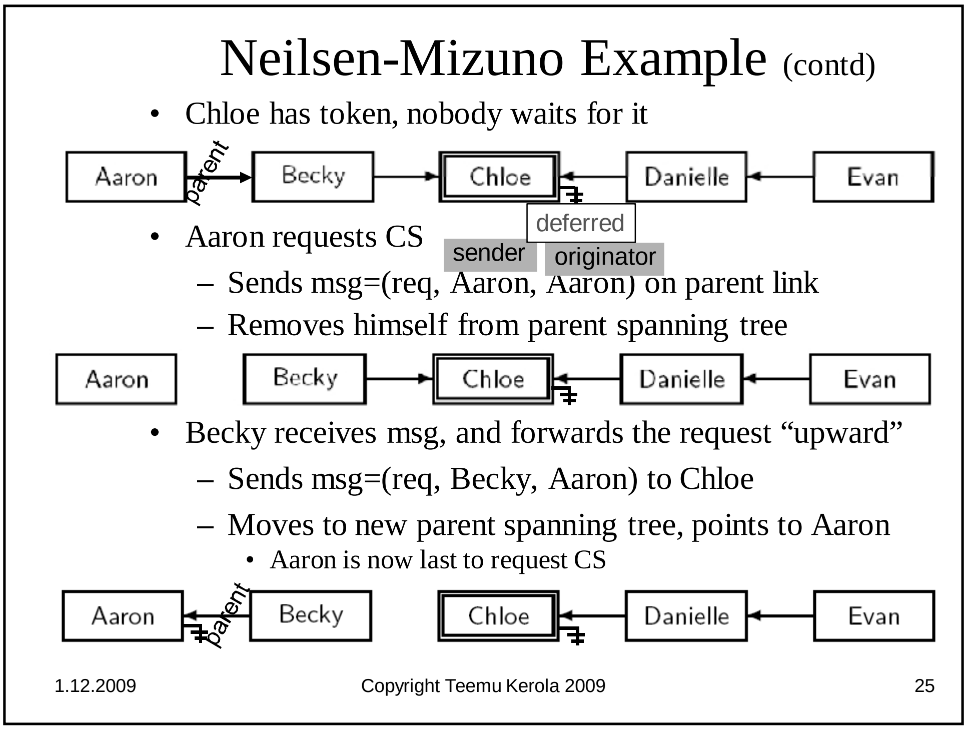
1.12.2009 Copyright Teemu Kerola 2009 23



Neilsen-Mizuno Example

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

1.12.2009 Copyright Teemu Kerola 2009 24



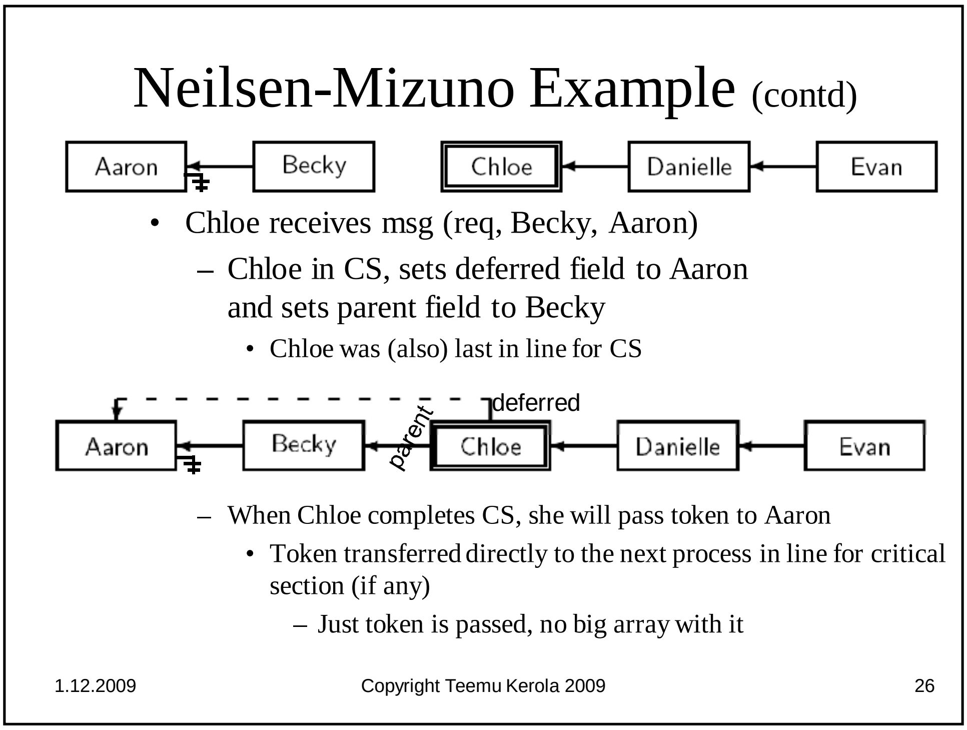
Neilsen-Mizuno Example (contd)

- Chloe has token, nobody waits for it

- Aaron requests CS
  - 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

1.12.2009 Copyright Teemu Kerola 2009 25



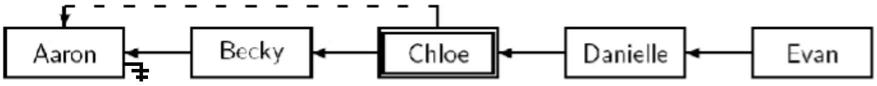
Neilsen-Mizuno Example (contd)

- 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

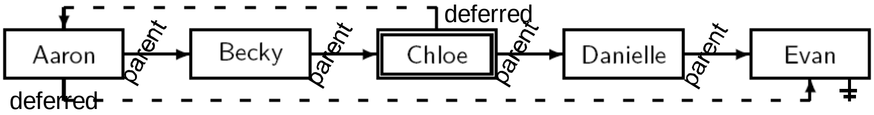
  - When Chloe completes CS, she will pass token to Aaron
    - Token transferred directly to the next process in line for critical section (if any)
      - Just token is passed, no big array with it

1.12.2009 Copyright Teemu Kerola 2009 26

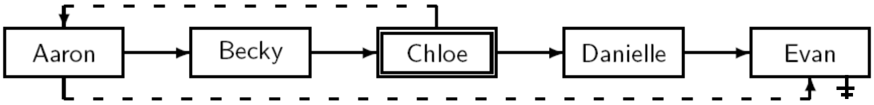
Neilsen-Mizuno Example (contd)



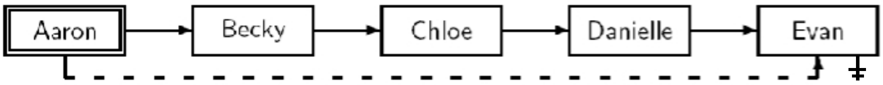
- Chloe still has CS, Evan wants CS
  - Sends (req, Evan, Evan) to Danielle
  - Danielle sends (req, Danielle, Evan) to Chloe
  - Chloe sends (req, Chloe, Evan) to Becky
  - Becky sends (req, Becky, Evan) to Aaron
  - Aaron makes a *deferred* link to Evan



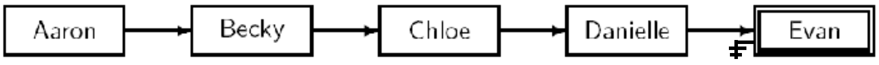
Neilsen-Mizuno Example (contd)



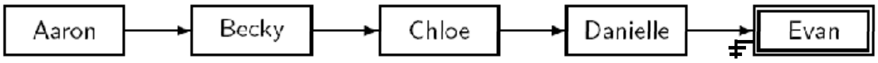
- Chloe completes CS, passes token to Aaron

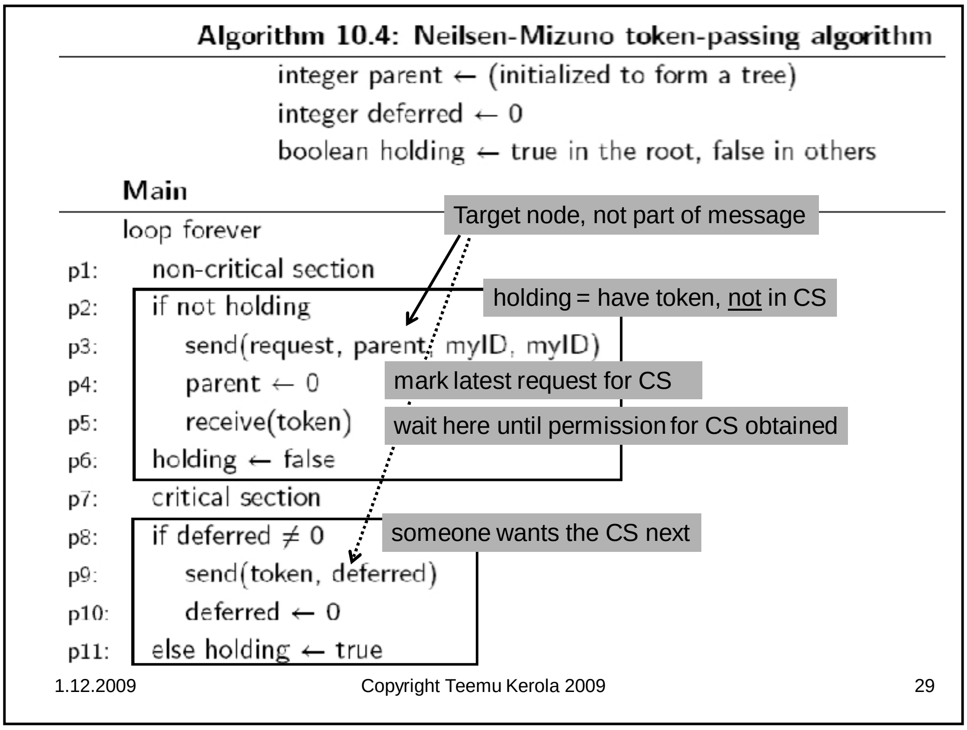


- Aaron completes CS, passes token to Evan



- 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

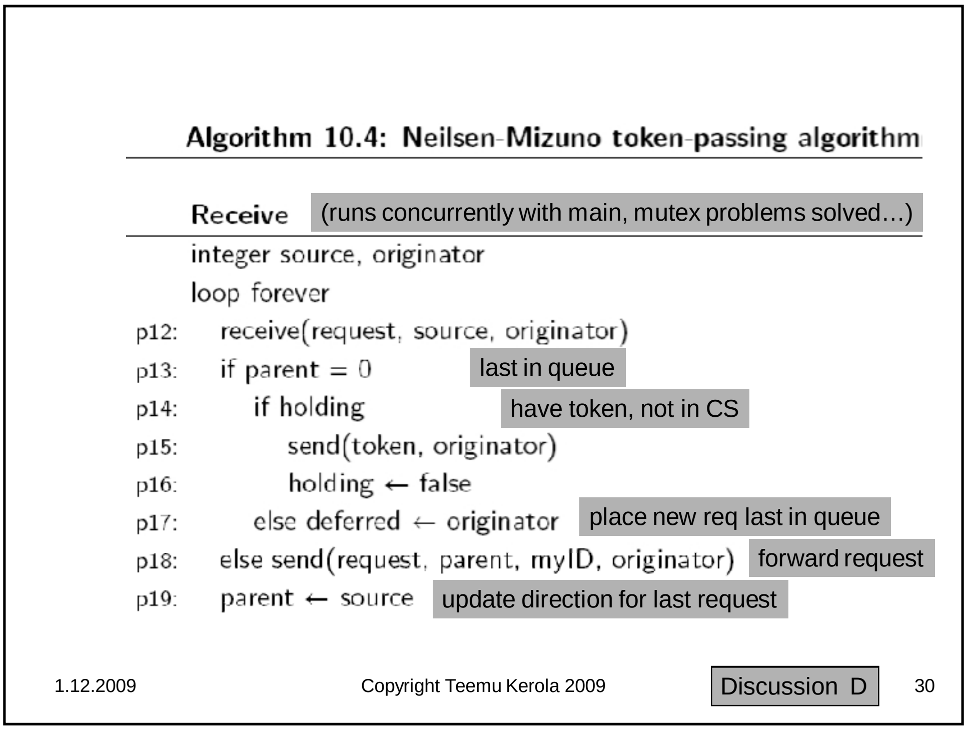
holding = have token, not in CS

mark latest request for CS

wait here until permission for CS obtained

someone wants the CS next

1.12.2009 Copyright Teemu Kerola 2009 29



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
p14:        if holding
p15:           send(token, originator)
p16:           holding ← false
p17:        else deferred ← originator
p18:     else send(request, parent, myID, originator)
p19:     parent ← source
```

last in queue

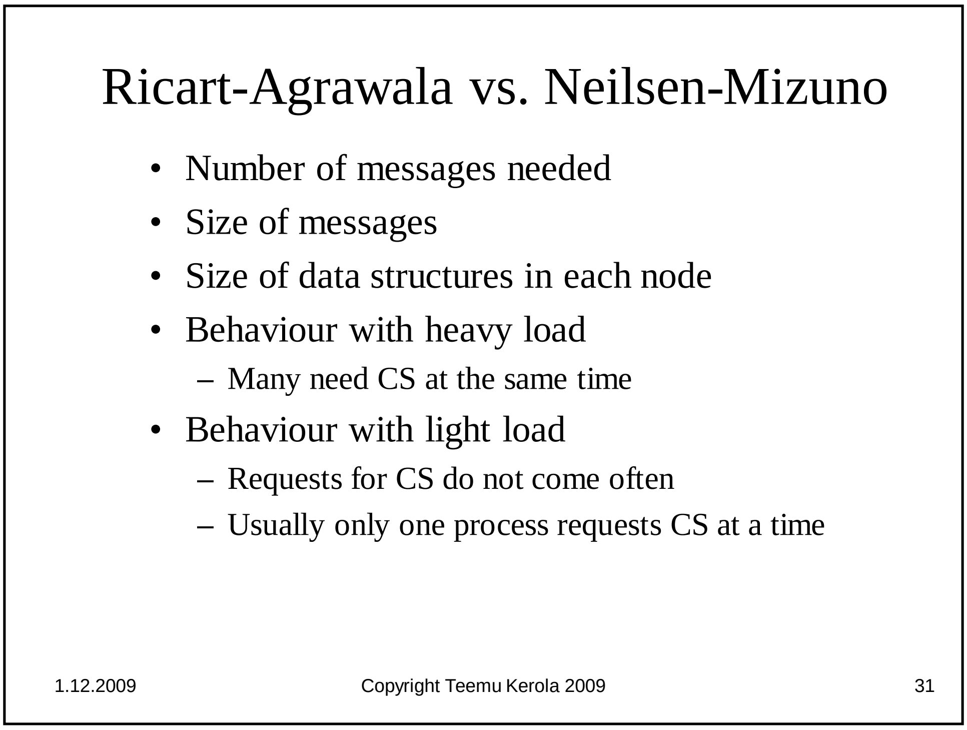
have token, not in CS

place new req last in queue

forward request

update direction for last request

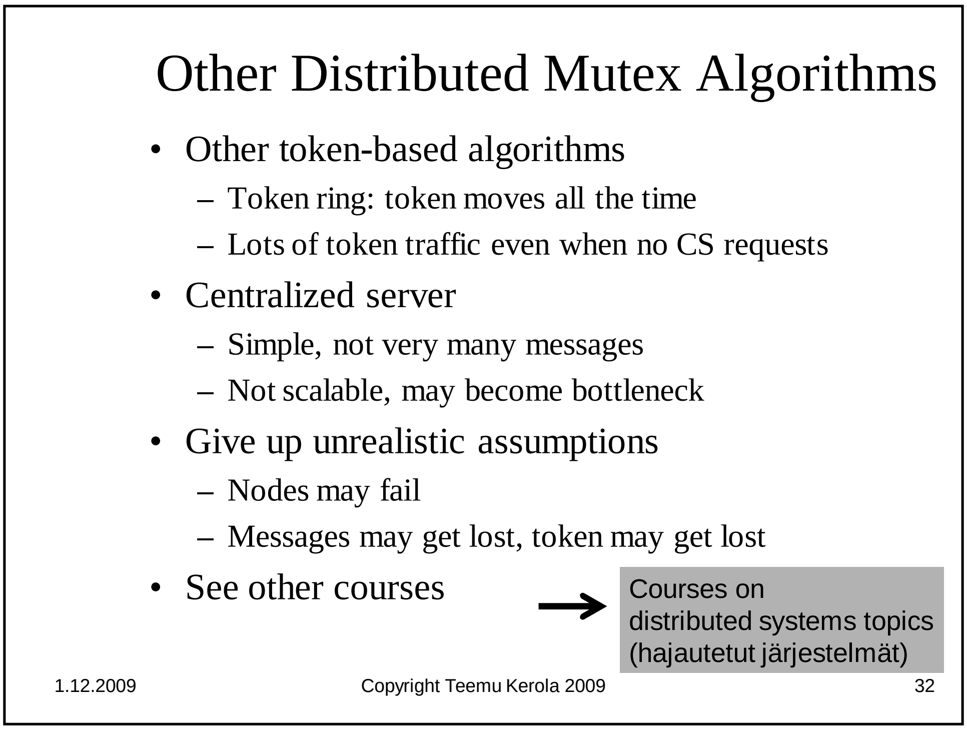
1.12.2009 Copyright Teemu Kerola 2009 Discussion D 30



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

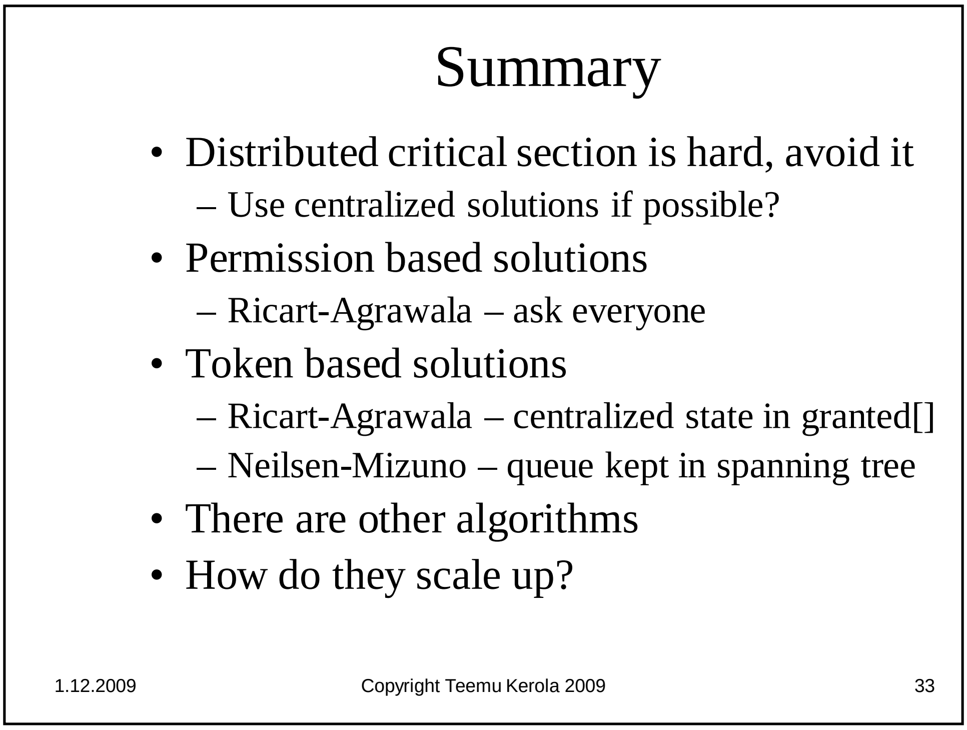
1.12.2009 Copyright Teemu Kerola 2009 31



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)

1.12.2009 Copyright Teemu Kerola 2009 32



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?

1.12.2009 Copyright Teemu Kerola 2009 33