Distributed Algorithms

Mutual Exclusion

Univ.-Prof. Dr.-Ing. habil. Gero Mühl

Architecture of Application Systems
Faculty for Computer Science and Electrical Engineering
University of Rostock

Overview

- Problem of mutual exclusion
- Algorithm with central coordinator
- Broadcast-based algorithms
- Quorum-based algorithms
- Token-based algorithms
- Comparison of algorithms

Mutual Exclusion

- Is about coordinating the exclusive access on resources such as files, printers, or data structures

- With exclusive access, only 1 process shall get access to the resource at the same time
- Sometimes instead of only one, at most n processes with $n > 1$ are allowed to access at the same time

- Assumption: If a process has the right to access, it voluntarily releases this right after finite time

Default for the lecture

Requirements

- Safety: Something bad that is irreparable shall never happen
  - Here: At no point in time more than one process should have the right to access the resource
- Liveness: Something good that should happen eventually happens
  - Here: If at least one process wants to access the resource, a process gets the right to access it after finite time
- Algorithms must fulfill both safety *and* liveness as often a trivial solution is possible for only *one* of the two

Requirements

- Fairness is often required additionally besides safety and liveness
- Starvation freeness
  - If a process continually wants to access the resource, access is eventually granted to this process
- Stronger fairness
  - The order in which access is granted to the processes takes the order in which access was requested into account

Solutions for Centralized Systems

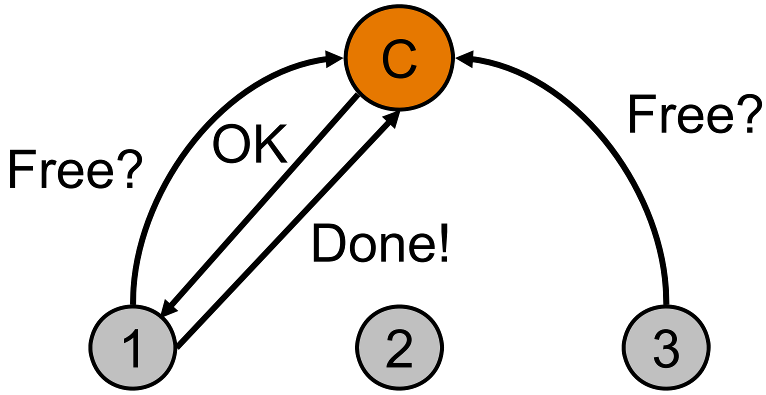
- Examples for mechanisms used to achieve mutual exclusion in centralized systems
  - Busy Waiting
  - Semaphores
  - Monitors
- Those mechanisms base on the fact that processes can atomically test and set a memory cell
- Not given in distributed systems!
- How can mutual exclusion be realized in distributed systems?

Algorithm with Central Coordinator

Centralized Solution for Distributed Systems

- A process is assigned as coordinator of a resource
- Coordinator
  - is informed about all requests / releases
  - grants accesses
- Algorithm easy to implement
- 3 messages per access
  (with blocking operations)

- Disadvantages
  - Single Point of Failure
  - Asymmetric load distribution



Broadcast-Based Algorithms

Broadcast Algorithm (Lamport, 1978)

- Assumptions
  - Reliable FIFO communication channels
  - Messages have unique logical time stamps
- Basic Idea
  - Each process manages a message queue that is ordered according to the messages' time stamps
  - Broadcast is used to send all requests and releases to all processes
- A process is only allowed to access the resource if
  1. its own request is the first request in its own queue
  2. it already received from each other process a message with a larger time stamp (e.g., a request confirmation)

Broadcast Algorithm

- Issue access request
  - Insert request into own queue
  - Send it to all other processes
- Send release after access
  - Remove request from own queue
  - Send release to all other processes

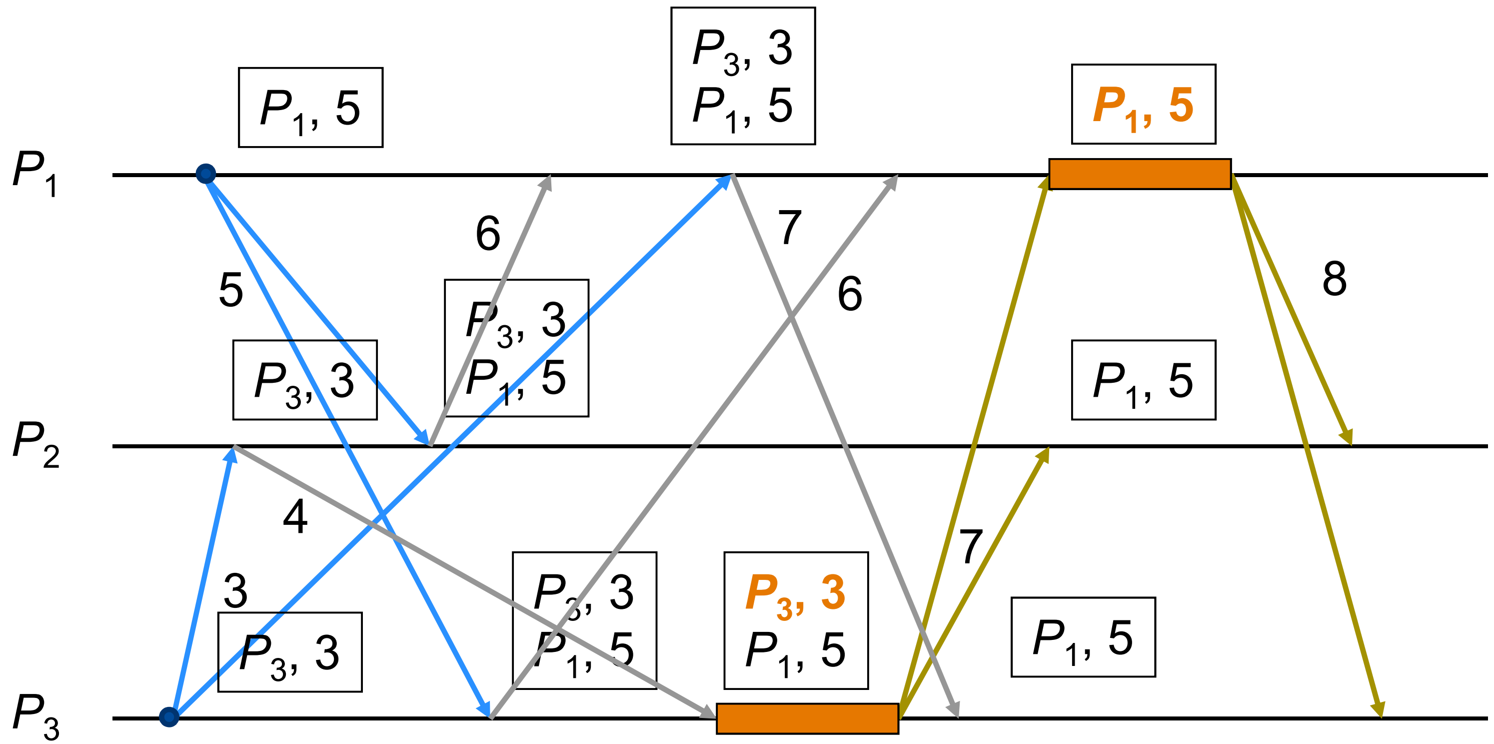
actions executed by process requesting access

- Received access request
  - Insert request into own queue
  - Send request confirmation to requesting process
- Received release
  - Remove request from own queue

actions executed by other processes

Broadcast Algorithm

If time stamps are equal, process-ID is used as tiebreaker



Blue Message: Request
Gray Message: Confirmation
Brown Message: Release

Orange time interval: access

Broadcast Algorithm

> Earliest request is globally unique, after all processes have received a message with a larger time stamp

> Message complexity
  > Sending of request to (n – 1) processes
  > (n – 1) processes send their confirmation
  > Sending of release to (n – 1) processes
  ⇒ 3 (n – 1) messages per access altogether

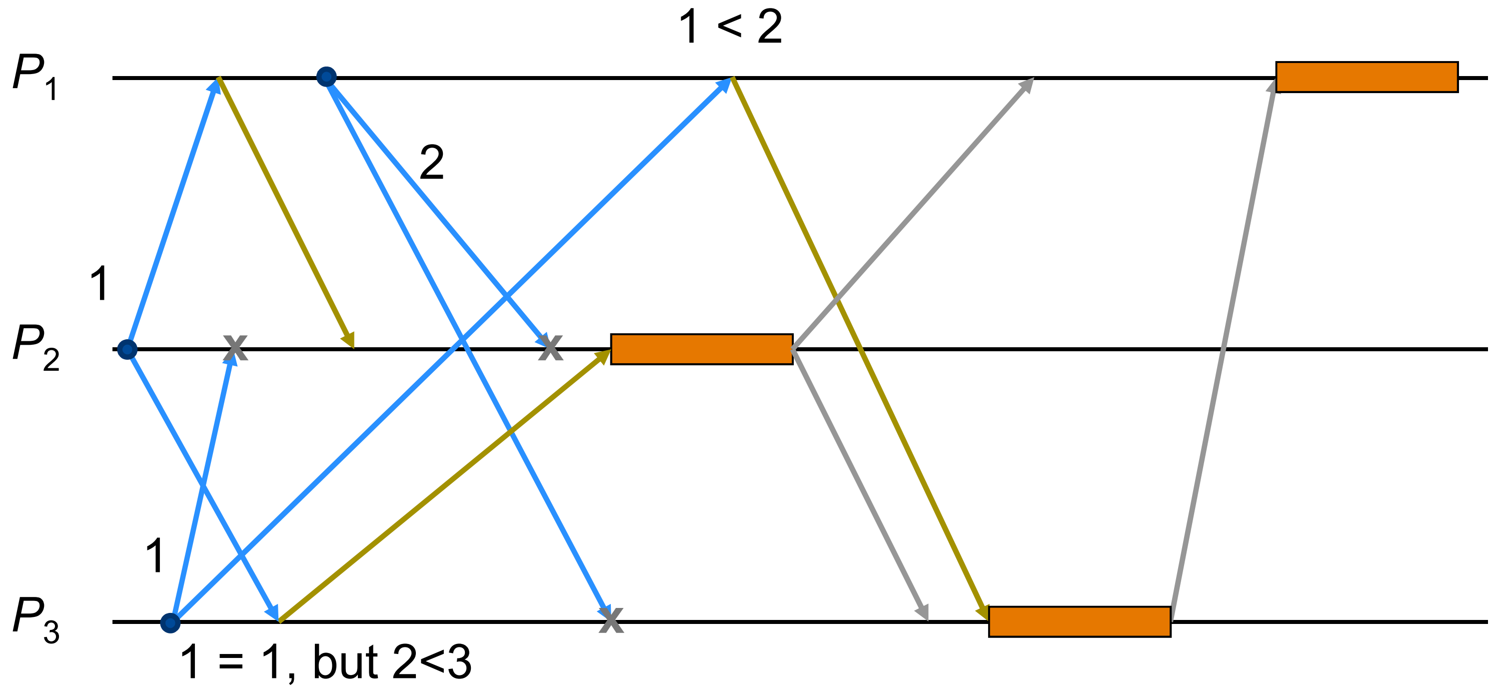
Improvement by Ricart and Agrawala, 1981

- Basic idea is to avoid explicit release messages through delayed confirmation → 2 (*n* – 1) messages per access
- No FIFO channels necessary
- Send request with sequence number that is by 1 larger than all previously received request to all other *n* – 1 processes
- Access after *n* – 1 confirmations have been received

- When a request arrives
  - Send confirmation immediately, if not applied or sender has „older rights“ (recognizable by sequence number)
    - If sequence numbers are identical, node ID ensures uniqueness
  - Otherwise, confirmation is sent only after the own access has granted

Improvement by Ricart and Agrawala, 1981

With the same time stamp process-ID as tiebreaker



Blue Message: Request
Brown Message: Immediate Confirmation
Gray Message: Delayed Confirmation

Orange time interval: access
x Confirmation is delayed

Optimization of Roucairol and Carvalho

- With the algorithm of Ricart and Agrawala a process P_i can access the resource if it has received a confirmation from all other processes
- Optimization: P_i can "reuse" the confirmation received from P_j, until it sends a confirmation to P_j (in response to a new request from P_j)
- This reduces the number of messages from $2\ (n-1)$ to the range of 0 up to $2\ (n-1)$ messages per access
- Optimization is especially useful if a process wants to access the resource several times in a short time interval
- Also if only a small fraction of the processes want access, a substantial amount of messages is saved
- Worst-case message complexity is still the same

Better Algorithms?

- Is a solution possible requiring less messages per access that distributes the load equally?
- Is there a solution not involving *all* processes in *each* coordination that distributes the load equally?

Quorum-Based Algorithms

Quorum-Based Algorithms

- With these algorithms, a process only has to ask a certain subset of the other processes instead of all to gain access
- This usually process-specific subset of the processes R_i is called quorum (or granting set) of the respective process P_i
- The granting sets must be constructed in a way such that mutual exclusion is ensured
- They should be constructed such that
  1. number of messages is minimized and
  2. load is balanced among the processes
- Note: Quorum-based algorithms can also be used for other applications such as achieving consistent replication

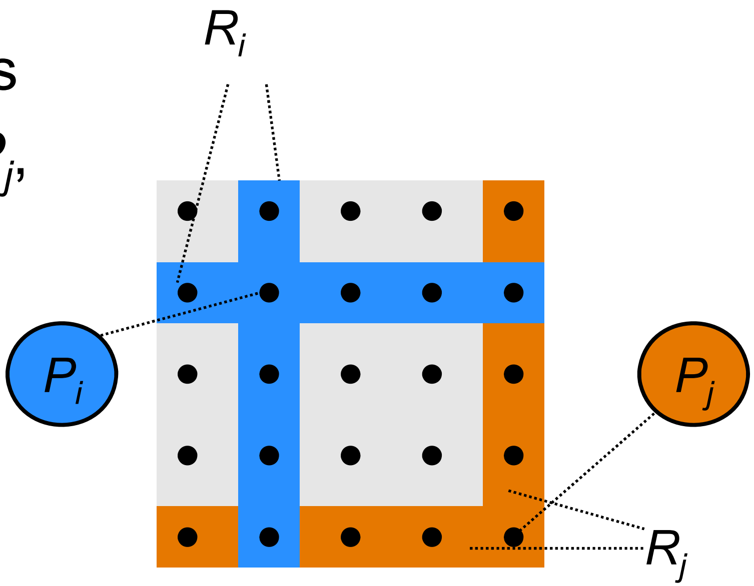
Quorum-Based Algorithms

- To guarantee safety, each pair of granting sets must have at least one process in common, i.e., $\forall i, j : i \neq j \Rightarrow R_i \cap R_j \neq \emptyset$
- This ensures that if one process gets confirmations from all processes in its granting set, no other process can

- Additionally, the following properties are desirable for a truly distributed algorithm
  - The size of all granting sets should be the same, i.e., $\forall i$: $|R_i| = K$
  - Each process should be a member of the same number of granting sets, i.e., $\forall i$: $|\{R_j \mid P_i \in R_j\}| = D$
  - Each granting set R_i should include the process P_i, i.e., $P_i \in R_i$

Mesh Algorithm (Maekawa, 1985)

- n processes are arranged in a quadratic mesh with edge length $\sqrt{n}$
- A process P_i must ask all processes in the same line *and* in the same column of the mesh to gain access
- For all pairs of processes P_i and P_j, their R_i and R_j have at least two processes in common
- Granting sets have the size $(2\sqrt{n}) - 1$
- Each process is a member of $(2\sqrt{n}) - 1$ granting sets



Mesh Algorithm (Maekawa, 1985)

- Message complexity without competing access requests
  - Send request to $(2\sqrt{n}) - 1$ processes (REQUEST)
  - $(2\sqrt{n}) - 1$ processes send confirmation (CONFIRM)
  - Send release to $(2\sqrt{n}) - 1$ processes (RELEASE)
  - $3[(2\sqrt{n}) - 1] = O(\sqrt{n})$ messages per access altogether
- Problem: With competing requests deadlocks may occur
  - Avoidable through the introduction of two additional message types (INQUIRE, RELINQUISH)
  - Requests are totally ordered with logical time stamps and an older request enforces its higher priority
  - Increases the number of messages per access on $5[(2\sqrt{\mathrm{n}}) - 1]$ in the worst-case

Basic Idea of Deadlock Avoidance

- All processes are either in the unlocked or in the locked state
- A process P_i that wants access to the resource
  - sends a REQUEST message to all processes in R_i
  - accesses the resource if it has received a CONFIRM message from all members of R_i
  - sends a RELEASE message to all processes in R_i, when it has finished the access
- A process P_j that is (i) in the unlocked state and (ii) receives a REQUEST message from P_i, sends a CONFIRM message to P_i and enters the locked state
- A process P_j that is (i) in the locked state and (ii) receives a RELEASE message, enters the unlocked state

Basic Idea of Deadlock Avoidance

- Assume P_j is in the locked state and it receives a REQUEST message from a process P_i that precedes the request from process P_k because of which P_j has entered the locked state
- In this case, P_j sends an INQUIRY message to P_k
- Otherwise, the request of P_i is queued

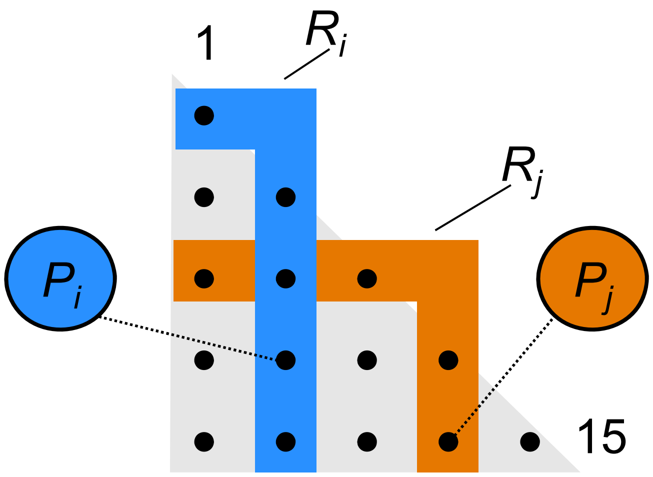
- P_k answers to an INQUIRY with a RELINQUISH message if it has not yet received a CONFIRM message from all processes in R_k
- Otherwise, P_k answers with a RELEASE message after it has finished accessing the resource

- If P_j receives a RELINQUISH message from P_k, it queues the request of P_k and sends a CONFIRM message to P_i
- If P_j receives a RELEASE message from P_k, it sends a CONFIRM message to P_i, because the request of P_k has already been served

- A request that has been queued is served when it is its turn

Triangular Arrangement of Processes

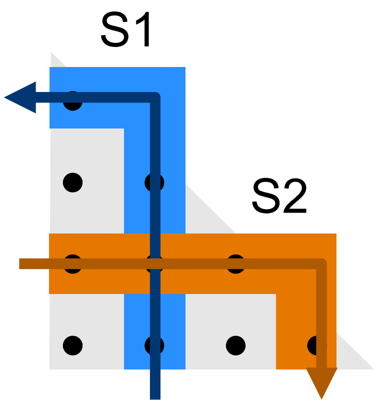
- In a quadratic mesh, two different granting sets have at least two processes in common, although a *single* common process would be sufficient
- Solution: Triangular arrangement
- Granting sets have a size of about $\sqrt{2}\sqrt{n}$
- Problem: The confirmation of some processes is needed more often than that of other processes!
  - Process 15 only occurs in one granting set
  - Process 1 occurs in 9 granting sets
- Solution for load balancing?



Same column and row one further above than the upper column

Solution for Load Balancing

- The solution is to use two different schemes
  - Schema 1: Same column and row one further above than upper column (up and left)
  - Schema 2: Same row and column one further right than the row furthest right (right and down)
- Characteristics
  - Each granting set intersects with each granting set of the same scheme
  - Each granting set of the one scheme intersects with each granting set of the other scheme
  - All processes occur altogether in both schemes equally often in a granting set
- Thus, an alternating (or also random) usage of both schemes can achieve load balancing



Minimal Arrangement

- Let K be the size of the granting set
- A minimal arrangement exists if there is a prime number p and a natural number m with $(K-1) = p^m$
- The arrangement, then, has $n = K(K-1) + 1$ processes
  - $K - 1 = 1 = 1^1$ $\quad n = 3$ (here, we assume 1 as prime)
  - $K - 1 = 2 = 2^1$ $\quad n = 7$
  - $K - 1 = 3 = 3^1$ $\quad n = 13$
  - $K - 1 = 4 = 2^2$ $\quad n = 21$
  - $K - 1 = 5 = 5^1$ $\quad n = 43$
  - $K - 1 = 7 = 7^1$ $\quad n = 57$
  - $K - 1 = 8 = 2^3$ $\quad n = 73$
  - …

For, e.g., $K = 7$ or $K = 10$ there is no minimal arrangement.

- For the size of the granting set holds

$$K = \tfrac{1}{2}\left(1 + \sqrt{4n-3}\right) = \lceil \sqrt{n} \rceil$$

Exemplary Minimal Arrangements

- $K = 2$
  - $B_1 = \{1, 2\}$
  - $B_3 = \{1, 3\}$
  - $B_2 = \{2, 3\}$
- $K = 3$
  - $B_1 = \{1, 2, 3\}$
  - $B_4 = \{1, 4, 5\}$
  - $B_6 = \{1, 6, 7\}$
  - $B_2 = \{2, 4, 6\}$
  - $B_5 = \{2, 5, 7\}$
  - $B_7 = \{3, 4, 7\}$
  - $B_3 = \{3, 5, 6\}$
- $K = 4$
  - $B_1 = \{1, 2, 3, 4\}$
  - $B_5 = \{1, 5, 6, 7\}$
  - $B_8 = \{1, 8, 9, 10\}$
  - $B_{11} = \{1, 11, 12, 13\}$
  - $B_2 = \{2, 5, 8, 11\}$
  - $B_6 = \{2, 6, 9, 12\}$
  - $B_7 = \{2, 7, 10, 13\}$
  - $B_{10} = \{3, 5, 10, 12\}$
  - $B_3 = \{3, 6, 8, 13\}$
  - $B_9 = \{3, 7, 9, 11\}$
  - $B_{13} = \{4, 5, 9, 13\}$
  - $B_4 = \{4, 6, 10, 11\}$
  - $B_{12} = \{4, 7, 8, 12\}$

Majority-Based Approaches

- Simple majority-based approach
  - A process can access the resource if it receives confirmations from at least $\lfloor n / 2 \rfloor + 1$ processes including itself

- Weighted majority-based approach
  - Each process P_i has a weight w_i
  - A process can access the resource if it receives confirmations with an overall weight that is greater than $\sum w_i / 2$
  - Again, the own weight is included

Token-Based Algorithms

Simple Token Ring Solution (Le Lann, 1977)

- Processes are arranged in a (logical) ring
- Access is controlled by circulating token
- Applicant waits for access until token reaches the process
- Accessing process relays the token with the release
- Processes without access intention relay the token directly

- Possible to use separate tokens for coordinating accesses to individual resources

Simple Token Ring-Solution

- Advantages
  - Simple, correct, fair algorithm
  - No deadlocks
  - No starvation
- Disadvantages
  - Token is always on the way, under certain circumstances uselessly
  - Thus, the message number per request is not limited
  - Long waiting time with large number of processes

Token-Based Solution (Suzuki & Kasami, 1985)

- A requesting process sends a request with its sequence number to *all* other processes
  - In a ring, this happens sequentially through a ring circulation
  - In another topology (complete mesh, tree etc.), through broadcast

- Each process P_i stores in a list R_i the highest currently received sequence number from each process
- Token stores
  - in a queue Q, the processes waiting for the token
  - in a list L, for each process the sequence number of the latest fulfilled request
- A process P_i can determine which requests have not yet been served, when receiving the token, by comparing of R_i with L

Token-Based Solution

- If a process P_i receives the token, it
  - Accesses (if it wants to)
  - Sets $L[i] := R_i[i]$
  - Attaches each process P_j (order in increasing sequence numbers) not part of Q to the end of Q for which $R_i[j] > L[j]$
  - Deletes itself from Q
  - If Q is not empty afterwards, the process sends the token
    - to the next process (ring),
    - to the first process in Q (complete meshing) or
    - to the next process in direction of the first process in Q (different topology)
  - Otherwise, it only sends the token on, if it receives a request from a process P_j whose sequence number is larger than $L[j]$

Solution with a Ring

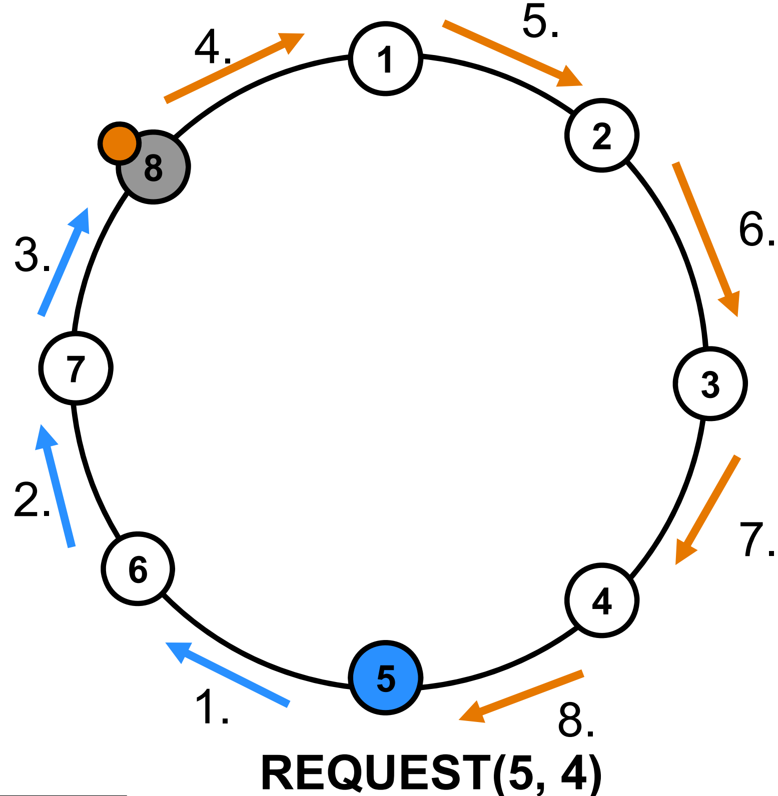
Most current request

Q
5

L	R_8
1, 0	1, 0
2, 0	2, 0
3, 1	3, 1
4, 0	4, 0
5, 3	**5, 4**
6, 0	6, 0
7, 0	7, 0
8, 5	8, 5

Processes waiting for access

Last fulfilled request.



1. A request does not need to be relayed if it meets the *resting* token.
2. The algorithm can be simplified to a great extent if there are no overtakes.
3. Maximal 2*n*-1 messages per access are needed in the physical topology

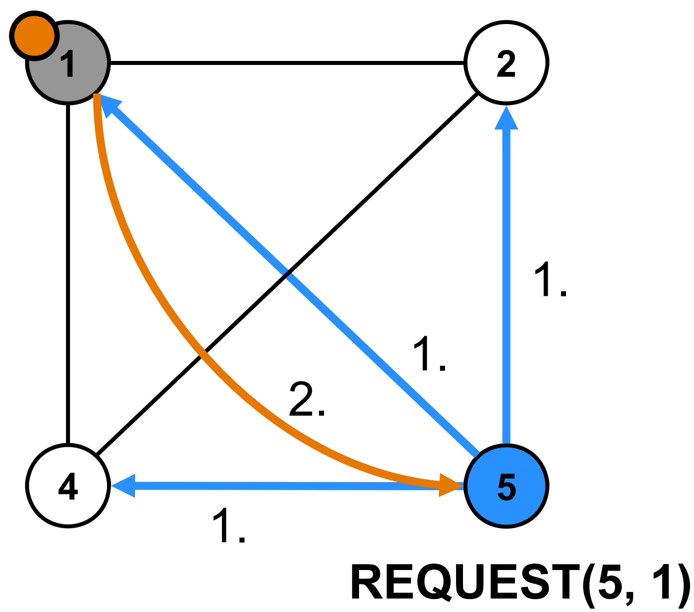
All depicted states after 3.

Solution with Complete Mesh

Exactly *0* or *n* messages are needed in the physical topology.

Q	L
5	1, 1
	2, 0
	3, 0
	4, 0
	5, 0
	6, 0
	7, 0
	8, 0

R_1
1, 1
2, 0
3, 0
4, 0
5, 1
6, 0
7, 0
8, 0



All depicted states after 1.

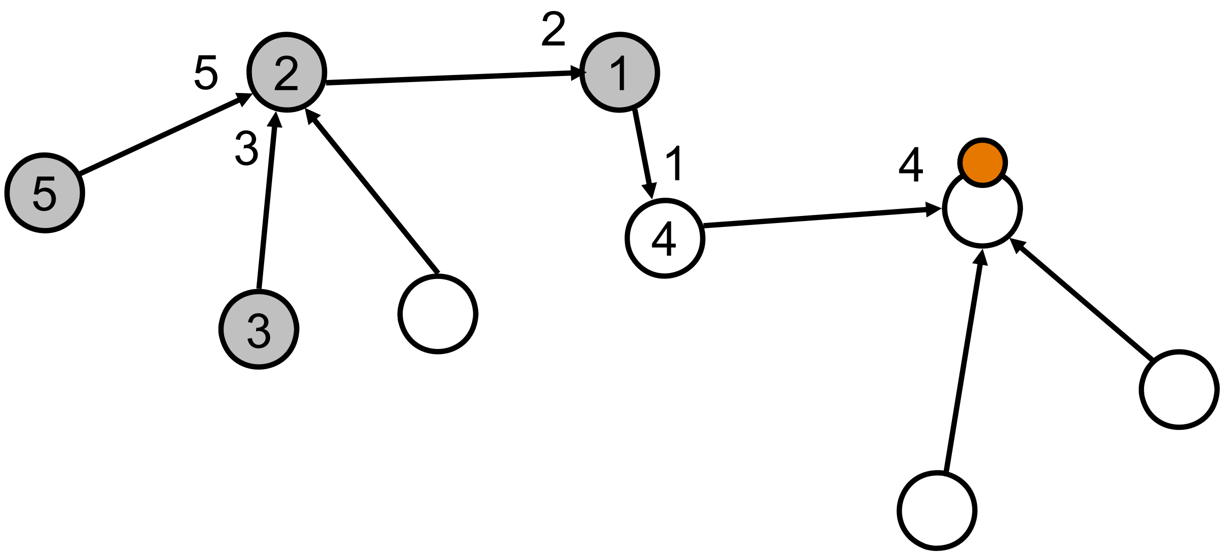
Lift Algorithm (Raymond, 1989)

> Uses a spanning tree to *selectively* forward the request into the direction towards the token (instead of sending it to all processes or to a specific subset)
> The edges of the spanning tree are directed; they always point towards to current position of the token
> The token wanders against the arrow direction and thereby turns around the direction of each edge passed
> A process that wants the token, sends a request over *its* outgoing edge
> If a process has received a request, it once sends a request into the direction of the token

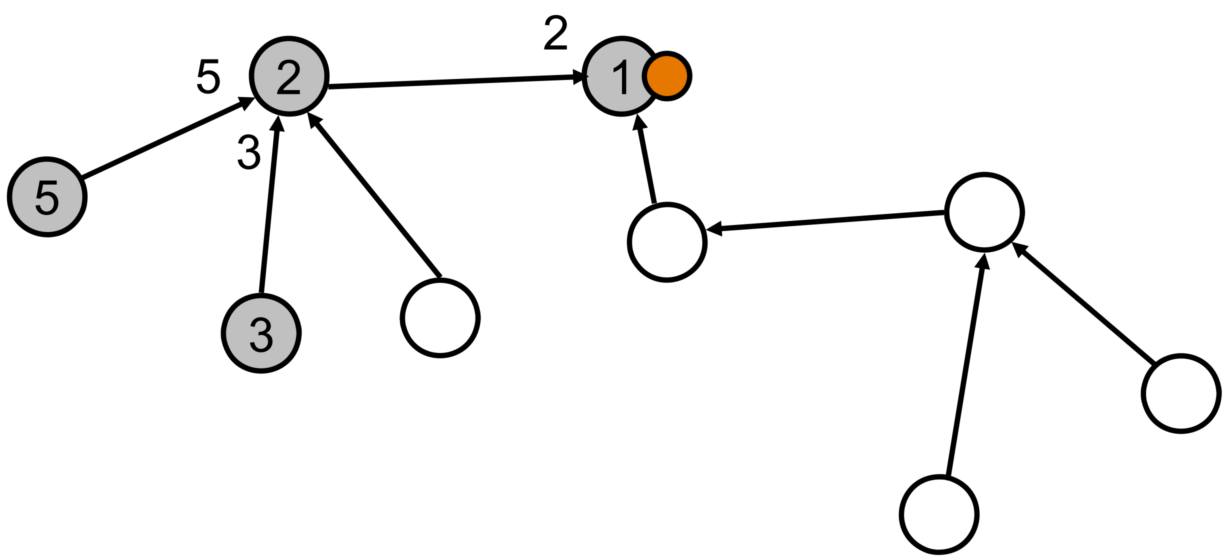
Lift Algorithm

- Each process remembers the processes from which it has received a request
- If a process receives the token
  - it relays it into one of the requesting directions
  - if there are more requests from other directions, it sends a request after the token
- To ensure fairness, a process must not serve a requesting direction arbitrarily often

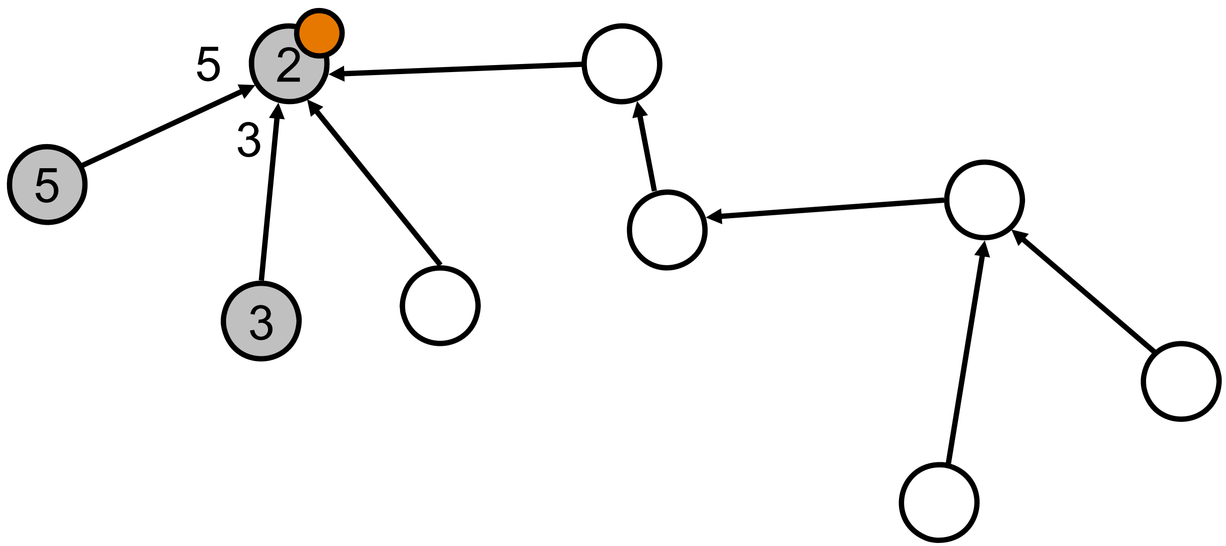
Lift Algorithm



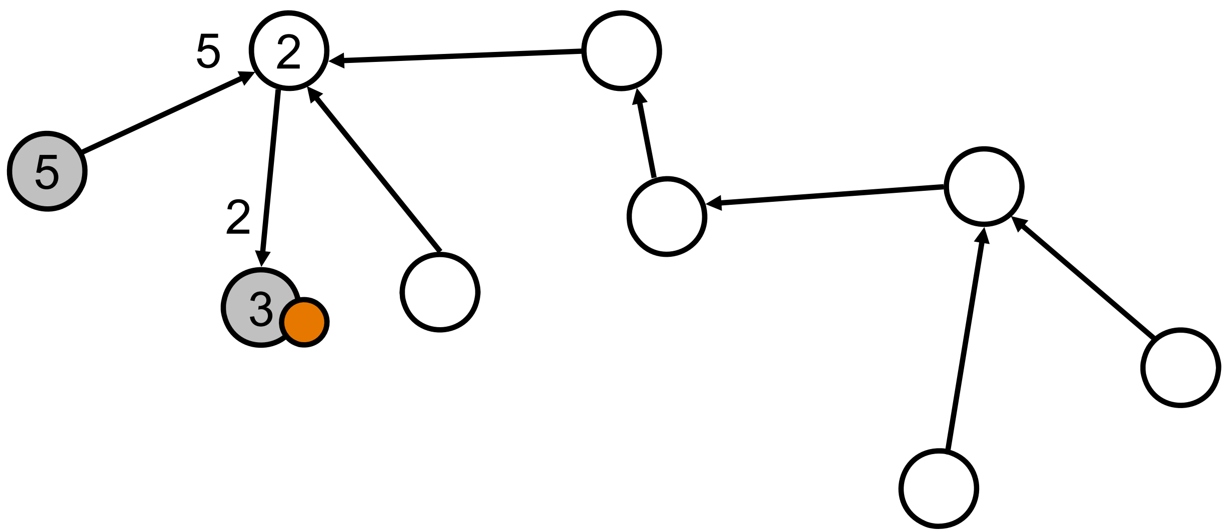
Lift Algorithm



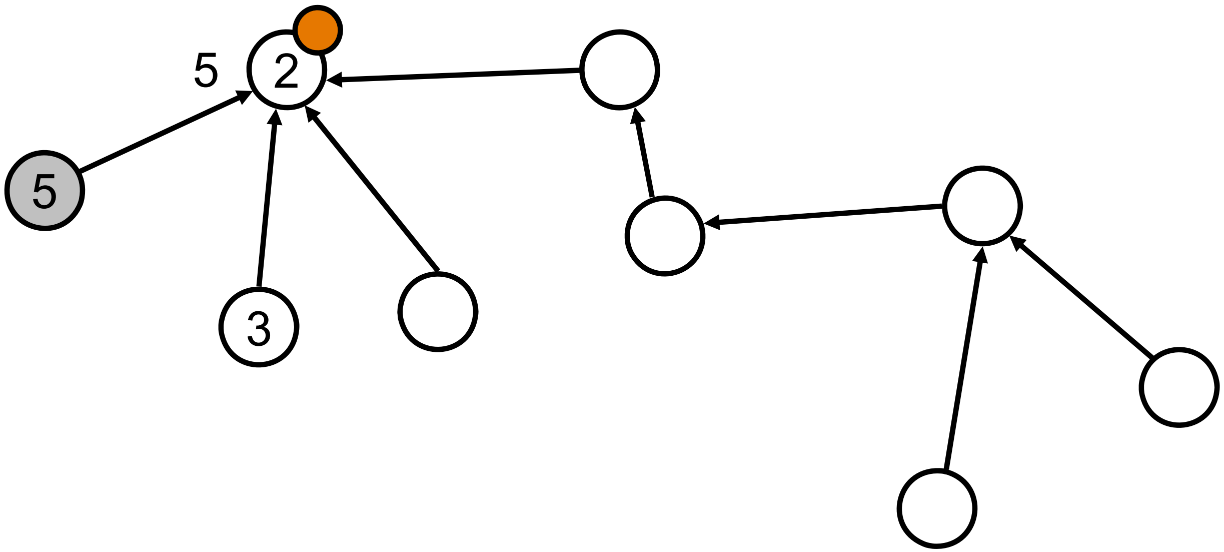
Lift Algorithm



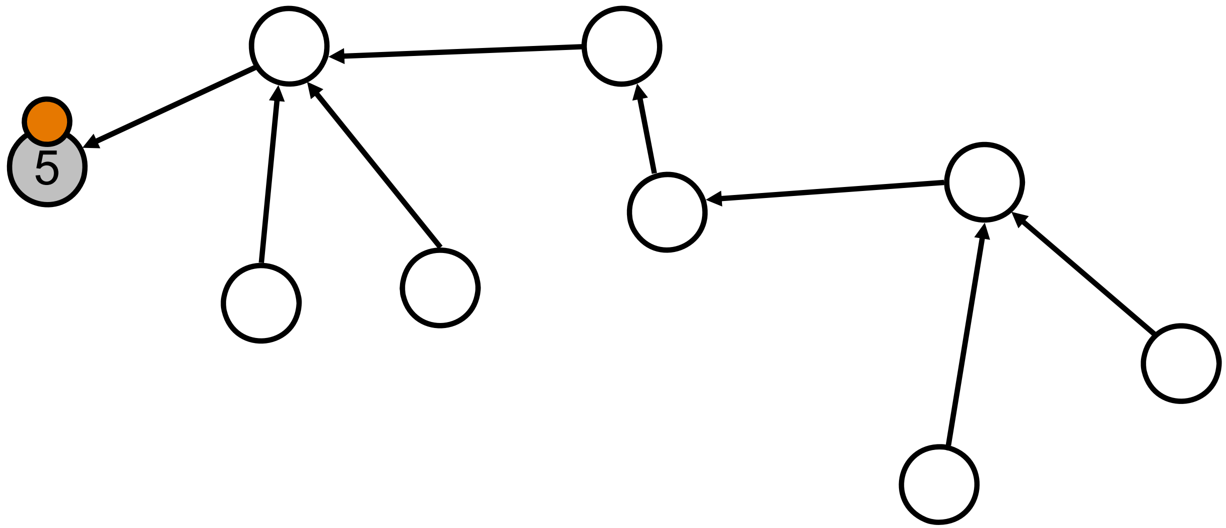
Lift Algorithm



Lift Algorithm



Lift Algorithm



Lift Algorithm

- The algorithm preserves the following invariant:
  from ach process, a directed path leads to the token
- In a *k*-ary balanced tree the maximal path length between arbitrary processes is $O(log_k\ n)$
- Accordingly, only at most $O(log_k\ n)$ messages are needed per access
- Starting state
  - Winner of an election gets the token and creates a spanning tree with edges directed towards it
  - Both can be achieved simultaneously by using the echo algorithm

Comparison of the Algorithms

Message Complexity per Access

Procedure	Message Complexity on Logical Topology
Token Ring	1 ... ∞
Simple Broadcast	$3\,(n-1)$
Improved Broadcast	$2\,(n-1)$
Improved Token Ring	$0 \ldots 2n-1$
Mesh Arrangement	$O(\sqrt{n})$
Lift Algorithm on *k*-ary Tree	$O(log_k\, n)$
Central Manager	3

Exemplary Exam Questions

1. What are the safety and liveness conditions for the problem of mutual exclusion?
2. Describe the broadcast-based algorithms of Lamport as well as of Ricart and Agrawala!
3. Explain the process grid algorithm of Maekawa!
4. Is there any better than the square arrangement?
5. How can load balancing be achieved in a triangular arrangement?
6. Is the triangular arrangement optimal?
7. Explain the Lift Algorithm!
8. What is the message complexity of the discussed algorithms?

Literature

1. L. Lamport. Time, Clocks, and the Ordering of Events in a Distributed Environment. Communications of the ACM, 21:558--564, July 1978.
2. G. Ricart and A. K. Agrawala. An Optimal Algorithm for Mutual Exclusion in Computer Networks. Communications of the ACM, 24(1):9--17, 1981.
3. M. Maekawa. A $\sqrt{N}$ Algorithm for Mutual Exclusion in Decentralized Systems. ACM Transactions on Computer Systems, 3(2):145--159, 1985.
4. K. Raymond. A Tree-Based Algorithm for Distributed Mutual Exclusion. ACM Transactions on Computer Systems, 7(1):61--77, 1989.
5. W. S. Luk and T. T. Wong. Two New Quorum Based Algorithms for Distributed Mutual Exclusion. In Proceedings of the 17th International Conference on Distributed Computing Systems (ICDCS '97), pages 100--107, 1997. IEEE Computer Society.

Literature

6. I. Suzuki and T. Kasami. A distributed mutual exclusion algorithm. ACM Transactions on Computer Systems, 3(4):344--349, 1985.
7. A. S. Tanenbaum and M. van Steen. Distributed Systems: Principles and Paradigms. Prentice Hall, 2002. Chapter 5, pages 262--270
8. N. Lynch. Distributed Algorithms. Morgan Kaufmann, 1996. Chapter 10
9. G. Coulouris, J. Dollimore, and T. Kindberg. Distributed Systems: Concepts and Design. Addison-Wesley, 3rd edition, 2001. Chapter 11.2, pages 423--431

Thank you for your kind attention!

Univ.-Prof. Dr.-Ing. habil. Gero Mühl

`gero.muehl@uni-rostock.de`
`http://wwwava.informatik.uni-rostock.de`