Middleware and Distributed Systems

Coordination and Consensus

Peter Tröger

Coordination

- Collection of processes in a distributed system need to agree on something
  - Mutual exclusion for shared resource access
    - Concurrent access to shared resource in distributed system
    - Critical section problem - like in OS, but no shared memory
  - Election of leader
    - Choose a process to play a particular role
    - Difference to mutual exclusion - leader must be known to everybody, and does not give away leadership explicitly after some time
  - Ordered multicast
    - Group of processes should receive copies of message sent to the group, sometimes with delivery and order guarantees

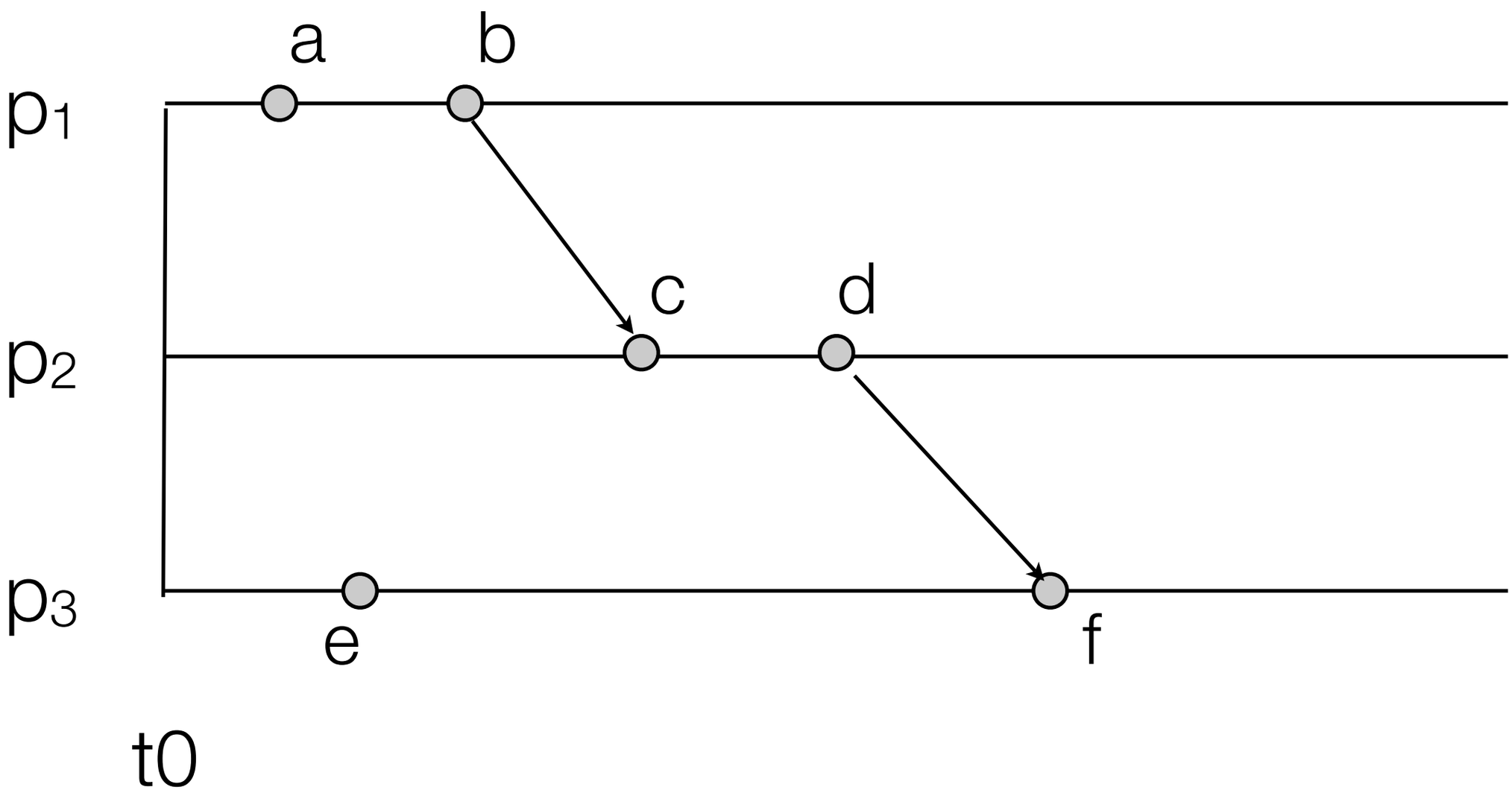
Reminder: Logical Time

- No perfect clock synchronization in a distributed system
  - Physical time unsuitable for global event ordering
- Partial ordering with *happened-before* relation "->" (causal ordering):
  - Two events e and e' in the same process p_i occure in the order p_i observes them, event of sending a message is before the event of receiving the message:
    - If exists a process p_i: e $->_i$ e' , then e -> e'
    - For any message m, send(m) -> receive(m)
    - If e, e' and e" are events with e->e' and e'->e", then e->e"
- Without happenend-before, events are *concurrent* - e||e'

Logical clocks

- Problems with happened-before relation
  - Captures data flow, but demands network message
  - Only potential causality (e.g. periodic message sending)
- Logical clock L_i for each process - capture ordering numerically
  - Li is increments before each event is issued
  - Sent message contains current L_i,
    receiver sets receive event timestamp to $L_j=\max(L_j, L_i) + 1$
- Events on different processes might have same timestamp
  - Totally ordered logical clocks by considering process identifiers
  - $(T_i, i) < (T_j, j)$ if either $T_i < T_j$, or $T_i = T_j$ and $i<j$ - no physical significance
- e -> e' leads to L(e) < L(e'), but L(e) < L(e') leads not to e -> e'

Reminder: Happened-before Example



Case 1 - Mutual Exclusion

- Assumptions: Processes do not fail, reliable message delivery, at-most-once
- Algorithms should ensure exclusive access in critical section for exactly one processor, and should avoid deadlocks and starvation
  - Useful fairness requirement is happened-before relation for requests
- Central server algorithm
  - Central server grants permission to enter critical section by providing token
  - Messages: Request token, release token, grant token
  - Requests are queued on the central server, clients are therefore blocked
  - Another example: NFS locking

NFS lockd

- RPC-based Network Lock Manager (NLM) service, *rpc.lockd* daemon on both client and server side
  - Application issues fctl() command for file locking, kernel sends request to local lock daemon (Kernel Lock Manager protocol)
  - Forwarded to remote host, 5 asynchronous operations
    - Test for lock (returns conflicting lock, or LCK_GRANTED)
    - Claim a lock (LCK_GRANTED or LCK_DENIED / LCK_BLOCKED)
    - Unlock a lock
    - Cancel a blocked lock request (client is no longer interested)
    - Grant a blocked lock (notification of server to client)

NFS statd

- Network Status Monitor (NSM) protocol for status change notification, implemented by *rpc.statd* daemon
  - When lock server grants lock request, it registers a callback in *statd* - invoked when the client has a status change
  - Client does the same when registering a server
  - On client crash and reboot, client informs *statd* about issue
    - Information to *lockd*, which frees all the locks
  - Same mechanism for server crash
    - Client *lockd* gets information, now tries to re-allocate all locks

Ring-based Mutual Exclusion

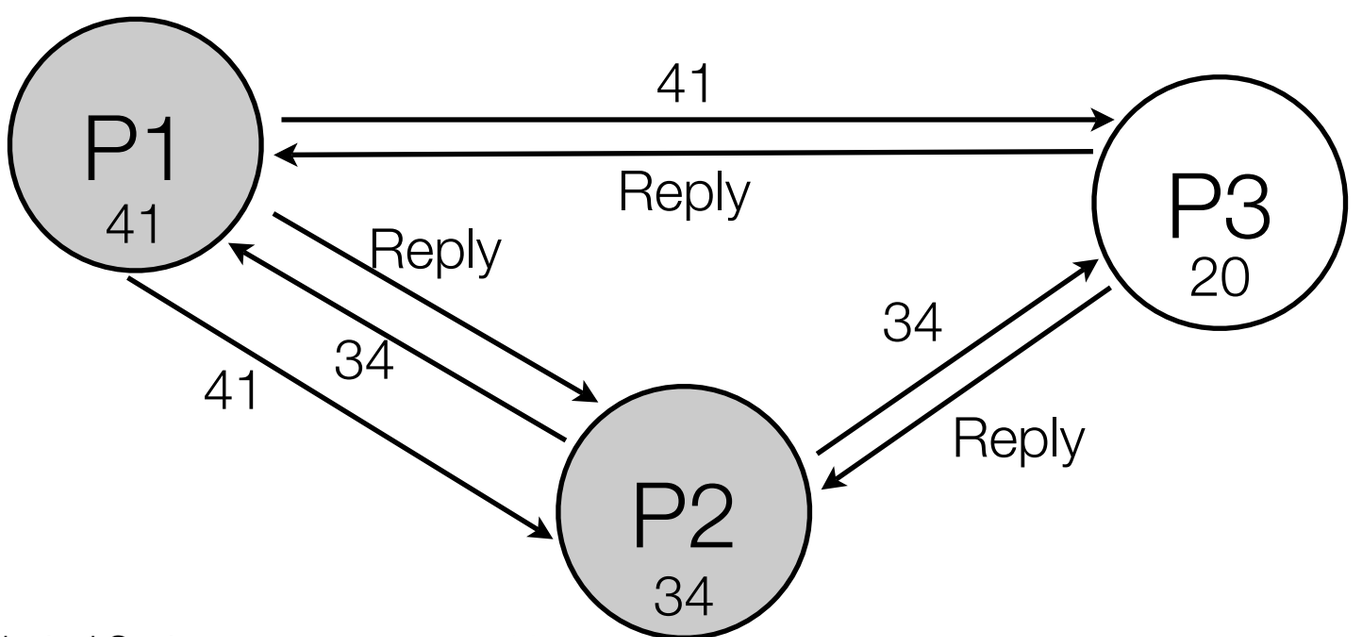
- Ring-based algorithm
  - Arrange agreement on mutual exclusion without additional node
  - Unidirectional ring topology, unrelated to physical interconnection
  - If process does not require to enter critical section, it forwards the token to the next one in the ring
  - Performance comparison
    - Consumed bandwidth as number of messages for one reservation
    - Client delay for critical section entry and exit
    - Synchronization delay between one process leaving and the next one entering the critical section

Coordination in Practice - Token Ring Network

- From late 1960's, by Olof Söderblom (IBM); IEEE 802.5
- Switched 10/BASE-T is cheaper, performance and reliability no longer an issue for Ethernet
- Token frame is repeated from station to station
  - Data is attached to token, marked as used
  - Receiver node copies data and forwards packet
  - Original sender removes data and marks token as free again
- Each station is either active or standby monitor, only one active monitor elected
  - Highest MAC address wins the election
  - Triggered when active monitor is lost or token receive timeout expired

Multicast-based Mutual Exclusion

- Ricart and Agrawala, 1981
- Process demanding critical section entry multicasts a request message to N-1 other processes, can only enter after it collected N-1 replies
  - Each process keeps logical Lamport clock, has either critical section RELEASED (answers immediately), critical section HELD (delays replies) or WANTED (delays replies if own request timestamp was earlier)
  - Example: Concurrent request by P1 and P2



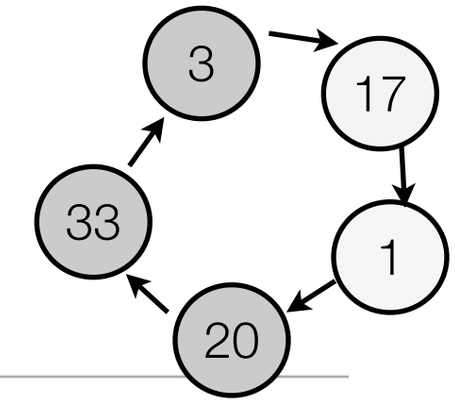
Case 2 - Election Algorithms

- Choose an unique process to play a particular role
  - Election is needed when the current leader fails, one node could be a better leader, or new nodes enter the system
    - One process calls the election
    - Unique choice needed, even with multiple parallel elections
  - Example: Time server in Berkeley clock synchronization algorithm
  - Example: IEEE1394 FireWire Tree Identify Protocol
    - Leader election after bus reset (node added or removed), root node acts as manager of the bus communication
    - Includes cycle detection (see „Distributed Algorithms“, N. Lynch 96)

Assumptions for Leader Election

- Elected process should be the one with the largest identifier
  - Unique identifiers, totally ordered
  - Algorithm should tell all participants the leader with largest identifier
  - Processes marked as participants or non-participants
- Assumptions (Molina)
  - All nodes rely on the same algorithm
  - No bugs, no message spoofing, non-volatile node memory
  - At-most-once transfer of messages, reliable network
  - Maximum response time for nodes, otherwise failed

Ring-based Election (Chang & Roberts ‘79)

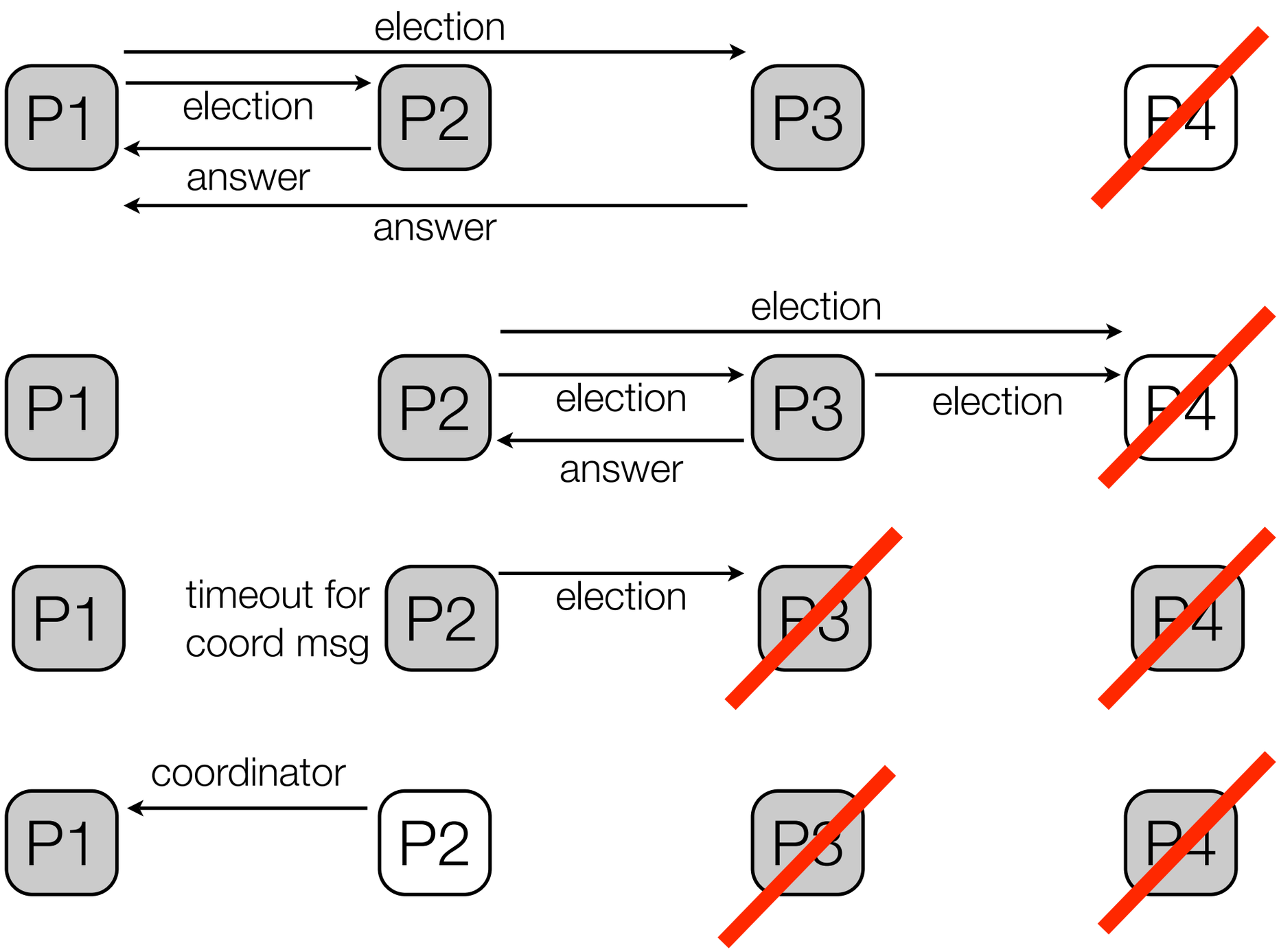


- Everybody marked as *none-participant*, one node changes its status and sends *election* message with its own id to clockwise neighbor
- On receipt of an *election* message
  - If the contained node ID is greater than the own, become a *participant* and forward message
  - If smaller and node is *non-participant*, exchange contained number, become a *participant* and the forward message
  - If smaller and node is already *participant*, don‘t forward message
  - If contained node ID is own ID and node is already *participant*, become the leader, change to *non-participant* and send *elected* message
- On receipt of an elected message, become *none-participant* and forward
- No fault tolerance, since ring is not allowed to break

Bully Algorithm (Garcia-Molina ‘82)

- Processes are allowed to crash during election (but: reliable message delivery)
  - Timeout used to detect system crash
  - Each process knows processes with higher numbers
- *Election* message (announcement), *answer* message, *coordinator* message
  - Process receiving an *election* message sends back an *answer* message and starts an own election with the higher nodes
  - Highest number elects itself by sending *coordinator* to lower numbers
  - Process with lower number sends *election* message to higher numbers and waits for *answer* message
    - If *answer* is received, waits for *coordinator* message, then starts new election
    - When timeout occurs, sends *coordinator* message to lower numbers

Bully Algorithm Example



Leader Election in Practice - SMB Election

- Computer Browser Service: Central list of resources in the network
  - Netware: Periodic announcement messages, high traffic
  - Windows: Elected master browser, used by all other resources
- Domain Master Browser: Receives browser lists from master browser
  - Typically PDC, every subnet has own master browser
- Master Browser: Master copy of network ressources, periodically updated by all available machines in the network
- Backup Browser: Fetches copy of resource list, offers it to clients
- Browser Clients: Use special mailslot BROWSE for queries
- Role of computer depends on operating system type and version

SMB Election

- Election process triggered by *RequestElection* to “\MAILSLOT\MSBROWSE“
  - If browser recently lost a round, it looses again
  - Browser determines if it has won the round by comparing senders election version with own number (or uptime, or lexical name comparison)
- If a browser wins, it sends out a new *RequestElection* message after some delay based on the current role
  - Master Browsers and Domain Master Browsers (100ms), Backup Browsers (200-600ms), all other (800-3000ms)
- If a browser wins 4 rounds in a row, it becomes the master browser, and sends out some announcement frame to the remaining machines

SMB Problems

- Worst case machine removal time of 51 minutes
  - Backup browser fetches updated list every 15 minutes
  - Computer announces itself in intervals (1 - 12 minutes)
    - After three left out intervals, computer is removed
- Too many election processes (e.g. by faulty Win95 clients) lead to unavailability of master browser
- Problem with broadcast messages over subnets - WINS server
  - Own naming services for NETBIOS networks
  - Meanwhile ActiveDirectory / DNS-based approaches

Demo

Case 3: Multicast Communication

- Group communication
  - Group of processes should receive copies of message sent to the group, sometimes with delivery and order guarantees
  - Process may join and leave group at arbitrary times
  - Many research implementations: V-System (1985), Chorus (1988), Amoeba (1991), Isis (1993), Horus (1996), Totem (1996), Transis (1996)
- Example: Database replication across several sites, query always routed to the nearest copy
  - Two concurrent updates, events arrive at different order at two sites (add $100 to account - increase account by 1% interest)
  - Demands totally-ordered multicast - all events should happen everywhere in the same order

Group Communication

- System model: reliable communication, processes may crash, messages carry sender ID, closed groups vs. open groups
  - *multicast (group, message)*
  - *deliver (message)*
- Delivery to application separated from receive event
  - Re-ordering of incoming messages to fullfil some order guarantees
- Different implementation strategies
  - Hardware support (multicast addresses)
  - Broadcast communication with filtering
  - Point-to-point to all members

Message Ordering

- FIFO Order
  - If correct member executes multicast(m_1) before multicast(m_2), then no other correct member delivers m_2 before m_1
- Causal Order
  - If correct member executes multicast(m_2) after receipt of m_1, then no other correct member delivers m_2 before m_1
  - If multicast(m_1)->multicast(m_2), then deliver(m_1)->deliver(m_2) (Happens-before)
- Total (Agreed) Order
  - Message delivery order is the same at all members in the group
- Total FIFO and total causal order

Fine, What About Fault Tolerance ?

- Mutual exclusion
  - Lost messages - not tolerated by algorithms
  - Crashed process - not tolerated by ring-based approach, server-based approach can tolerate client crash if it neither holds or requested the token
  - Even if faulty process can be detected, system state must be re-established (e.g. current owner of the token in server-based approach)
- Election
  - Timeout as failure detector might lead to wrong results
- Multicast communication
  - Assumption of crash-fault semantics

Fault - Error - Failure

- A system **failure** is an *event* that occurs when the delivered service deviates from correct service.
- An **error** is that part of the system *state* that may cause a subsequent failure: a failure occurs when an error reaches the service interface and alters the service.
- A **fault** is the adjudged or hypothesized *cause* of an error.
- A fault originally causes an error within the state of one (or more) components, but system failure will not occur as long as the error does not reach the service interface of the system.
  - System failure can be a fault to high layers.

„Fundamental Concepts of Dependability“
(Algirdas Avizienis, Jean-Claude Laprie, Brian Randell)

Example

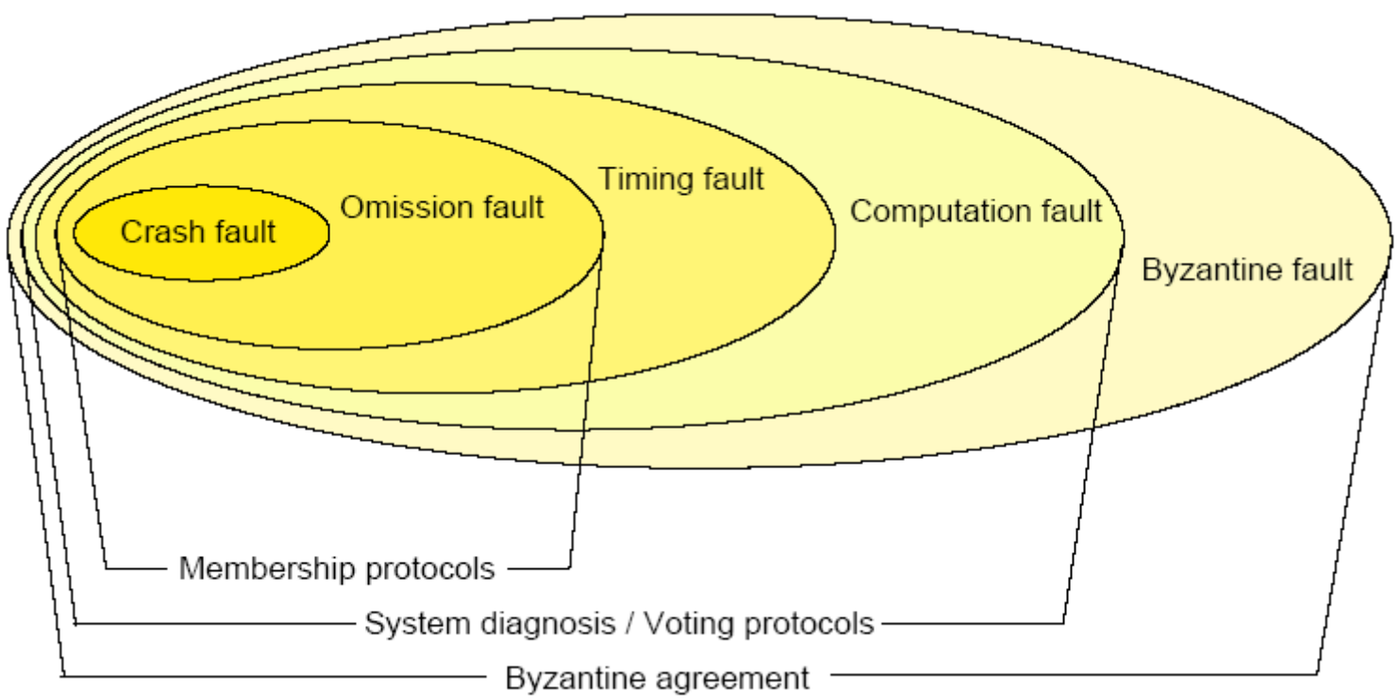
The result of an **error** by a programmer leads to a **failure** to write the correct instruction or data, that in turn results in a (dormant) **fault** in the written software (faulty instruction(s) or data).

Upon activation (invoking the component where the fault resides and triggering the faulty instruction) the **fault** becomes **active** and produces an **error**; if and when the **error** affects the delivered service (in information content and/or in the timing of delivery), a **failure** occurs.

- Failure assumptions for simplicity of theoretical work
  - Reliable communication protocol, but messages might be delayed
  - Node communication capabilities independent from other nodes failure
  - Failures might lead to *asymmetric communication* paths, and *network partitioning*

Fault Model (Flaviu Cristian)

- Originates from hardware background, meanwhile adopted to software
  - How many faults of different classes can occur
- Process as black box, only look on input and output messages
- Link faults are mapped to the participating nodes
- Timing of faults: Fault delay, repeat time, recovery time, reboot time, ...



Fault Types

- Fail Stop Fault : Processor stops all operations, notifies the other ones
- Crash Fault : Processor looses internal state or stops without notification
- Omission Fault : Processor will break a deadline or cannot start a task
  - Send / Receiver Omission Fault: Necessary message was not not sent / not received in time
- Timing Fault / Performance Fault : Processor stops a task before its time window, after its time window, or never
- Incorrect Computation Fault : No correct output on correct input
- Byzantine Fault / Arbitrary Fault : Every possible fault
  - Authenticated Byzantine Fault : Every possible fault, but authenticated messages cannot be tampered

Failure Detectors

- Coordination research typically concentrates on crash or byzantine failures
  - Detection of process crash by *failure detector* [Chandra and Toueg 96]
  - Processes query if a particular other process failed
  - Typically local objects to the supervised process
- Unreliable failure detector states *unsuspected* or *suspected*
  - Decision maybe based on message arrival rate
  - Example implementation: pair-wise heartbeat
- Reliable failure detector states *unsuspected* or *failed*
  - Implementation demands upper limit for network transfer

Consensus

- Processes in a distributed system need to agree on something, even in case of any kind of failure - Byzantine Generals problem (Lamport 1982)
  - Byzantine Empire's army must decide unanimously whether to attack some enemy army - only concerted effort results in victory
  - Geographic separation of the generals, who must communicate by sending messengers to each other; commander and lieutenants
  - Presence of traitors amongst the generals
    - Trick some generals into attacking
    - Force a decision that is not consistent with the generals' desires
    - Confusing some generals so that they are unable to make up their minds
  - Loyal (non-faulty) generals need unanimous agreement on their strategy
  - Commander issues order, lieutenants must decide to attack or retreat

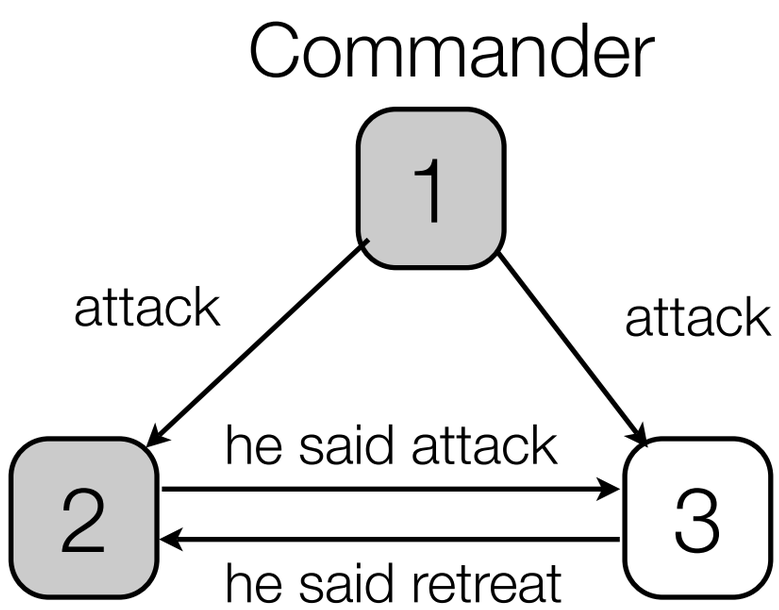
Byzantine Generals Problem

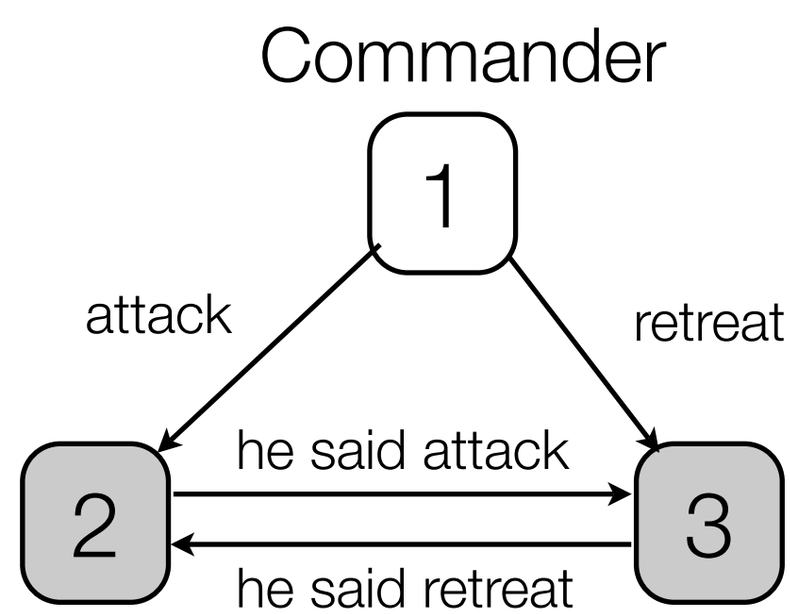
- Faulty ‚treacherous‘ generals
  - Commander might tell different things to different people
  - Lieutenant tells only some peers that commander told to attack
- Termination condition: Each correct process decides on something
- Agreement condition: Decision value of all correct processes is the same
- Integrity condition: If the commander is correct, all processes have agreement
- Generalized impossibility result by Pease et.al.
  - Let f be the maximum number of faulty processes in a system of n processes. As long as n<=3f, there is no algorithm to solve the byzantine generals problem.

Three Generals

- n=2 - clear
- For n=3, f=1, it is impossible to reach consensus
  - P2 cannot figure out who the traitor is, so he obeys the attack order
  - P3 cannot figure out who the traitor is, so he obeys the retreat order

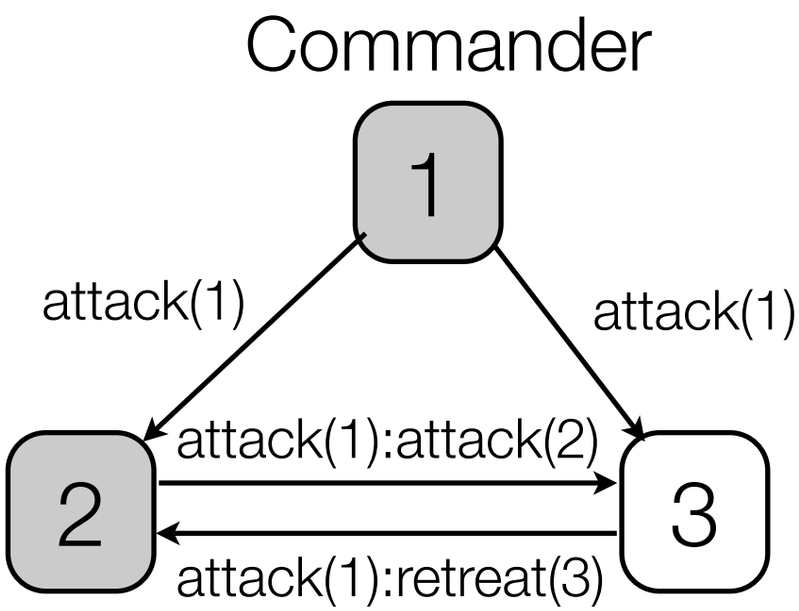
    -> violates agreement condition, since both correct nodes should do the same

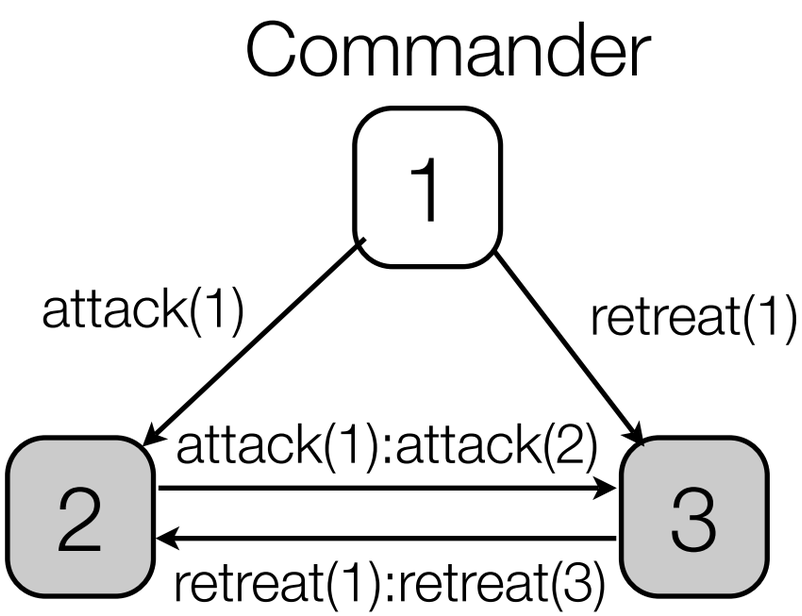




Signed Messages

- A loyal general's signature cannot be forged, and any alteration of the contents of his signed messages can be detected
- Anyone can verify the authenticity of a general's signature.

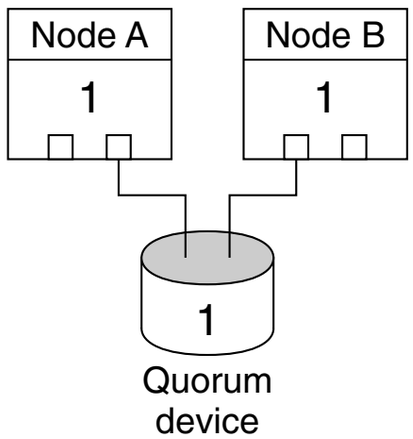




Real-World Consensus Examples

- Cluster environments support multi-host devices for single-node fault tolerance
  - Coordinated concurrent access to storage device (e.g. database)
- Active split might cause data corruption
  - Split brain: Every partition thinks it is the only one
  - Amnesia: Start cluster on node which has old configuration available
- Each node gets vote, partition with vote majority is allowed to operate
  - Static setting for number of configured votes, depends on node count
  - Two node cluster needs additional external vote - quorum device
    - Device is connected to N nodes, gets N-1 votes
    - Contributes votes if at least one of its nodes is a cluster member

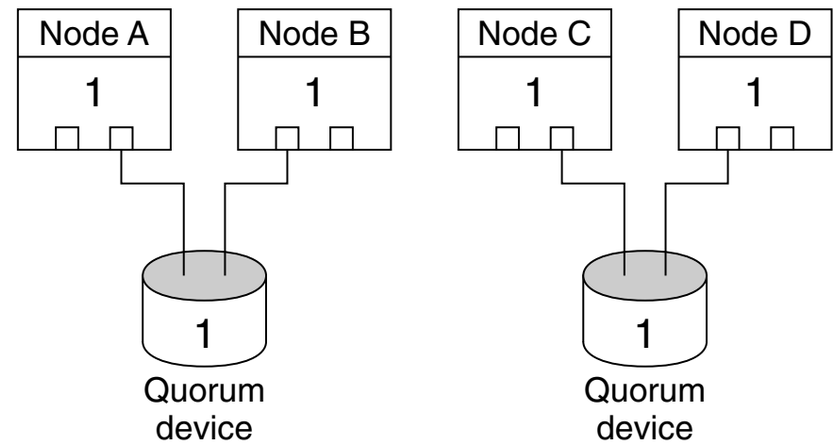
Quorum Device in SUN Cluster Environment



Any node can act as cluster

Total configured votes: 3

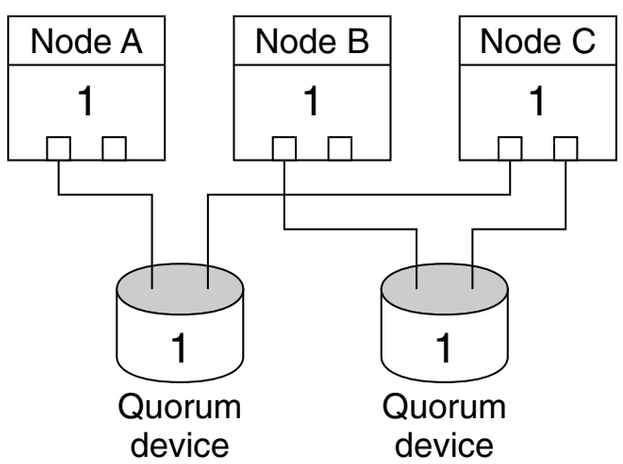
Votes required for quorum: 2



Each pair must be available

Total configured votes: 6

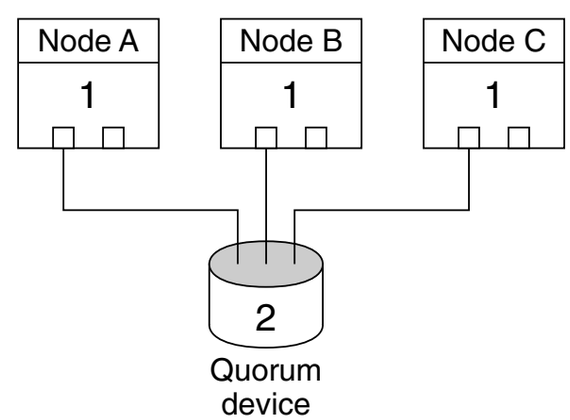
Votes required for quorum: 4



Node C as hot spare

Total configured votes: 5

Votes required for quorum: 3



Any node can act as cluster

Total configured votes: 5

Votes required for quorum: 3

Cluster Quorum

- Quorum device must be fault-tolerant, and reachable from all nodes
  - Oracle RAC: Voting disk, typically by using a SAN system
  - Sun Cluster: SCSI disk
- Concurrent writing to voting disk
  - In case of lost interconnect, last writer wins
- Usually also parallel storage of cluster status on quorum device
  - Allows faster node recovery after restart

Summary

- More distributed algorithms
  - Breadth-first search, finding shortest path or minimum spanning tree
  - Find maximum independent set of nodes
  - Atomic objects
- More consensus problems
  - Agreement on small set of values
  - Approximate agreement on real value
  - Distributed database commit
  - Global snapshot