Sistemas Distribuídos

José Orlando Pereira

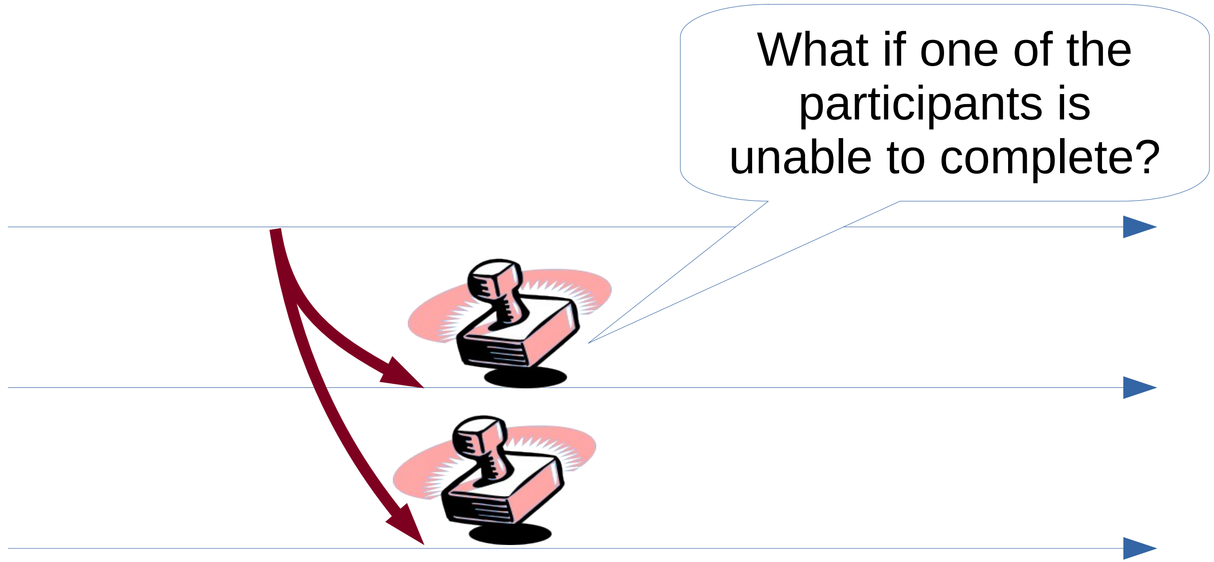
Departamento de Informática
Universidade do Minho

Fault tolerance

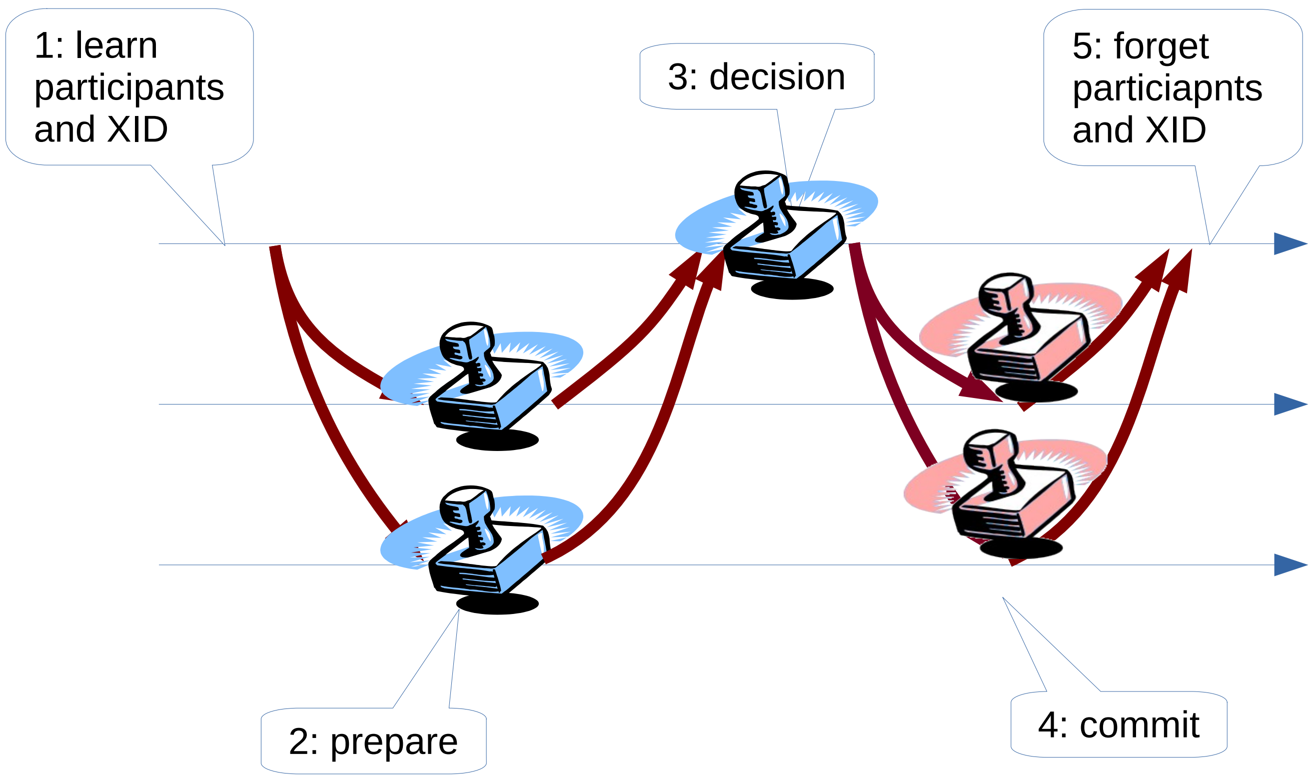
- A distributed system is composed by autonomous computing elements and can tolerate faults to avoid failure
- Fault model describes types of faults:
  - Omissive: process crash, lost message, …
  - Assertive (a.k.a. Byzantine): corrupted messages, …
- Fault model describes the number of faults:
  - Example: Number of processes that can crash

Transactional commit

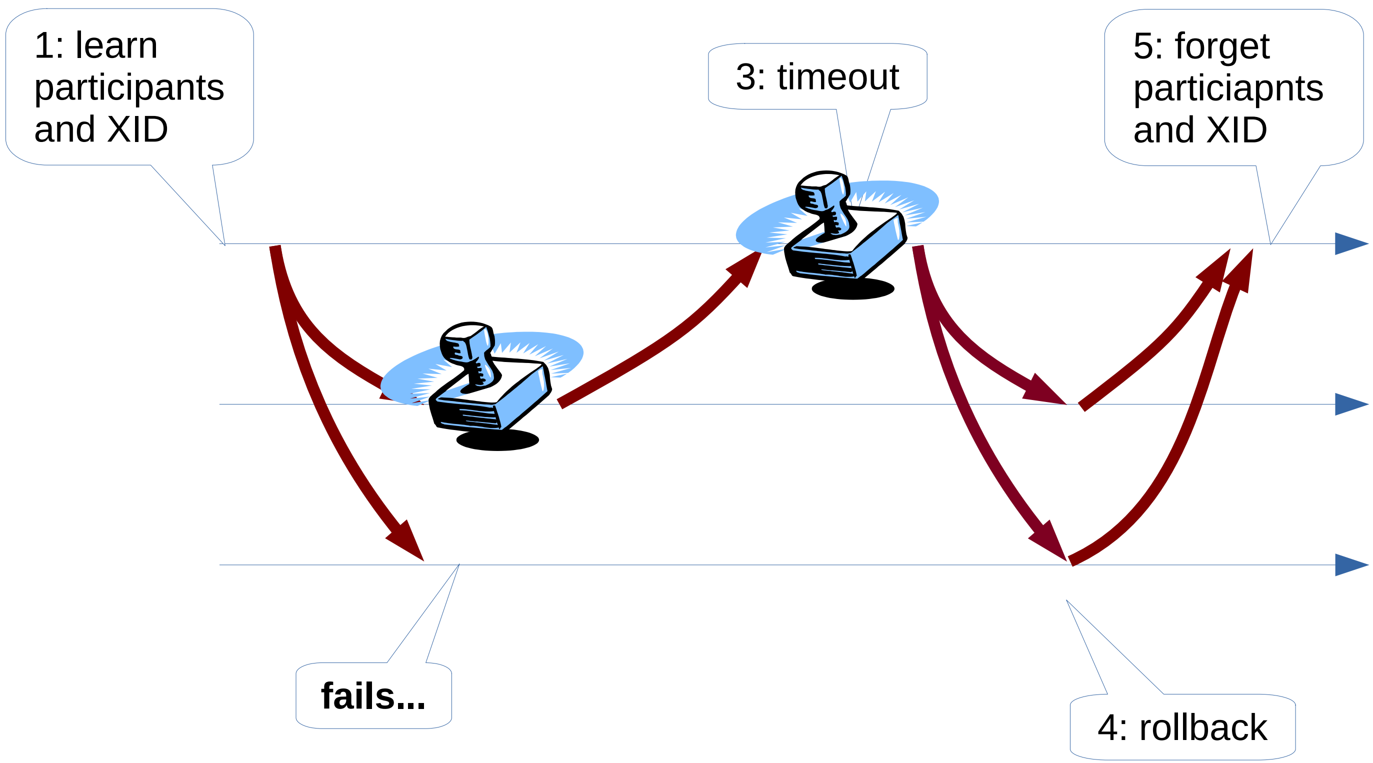
- Coordinate multiple irreversible actions across a distributed system:



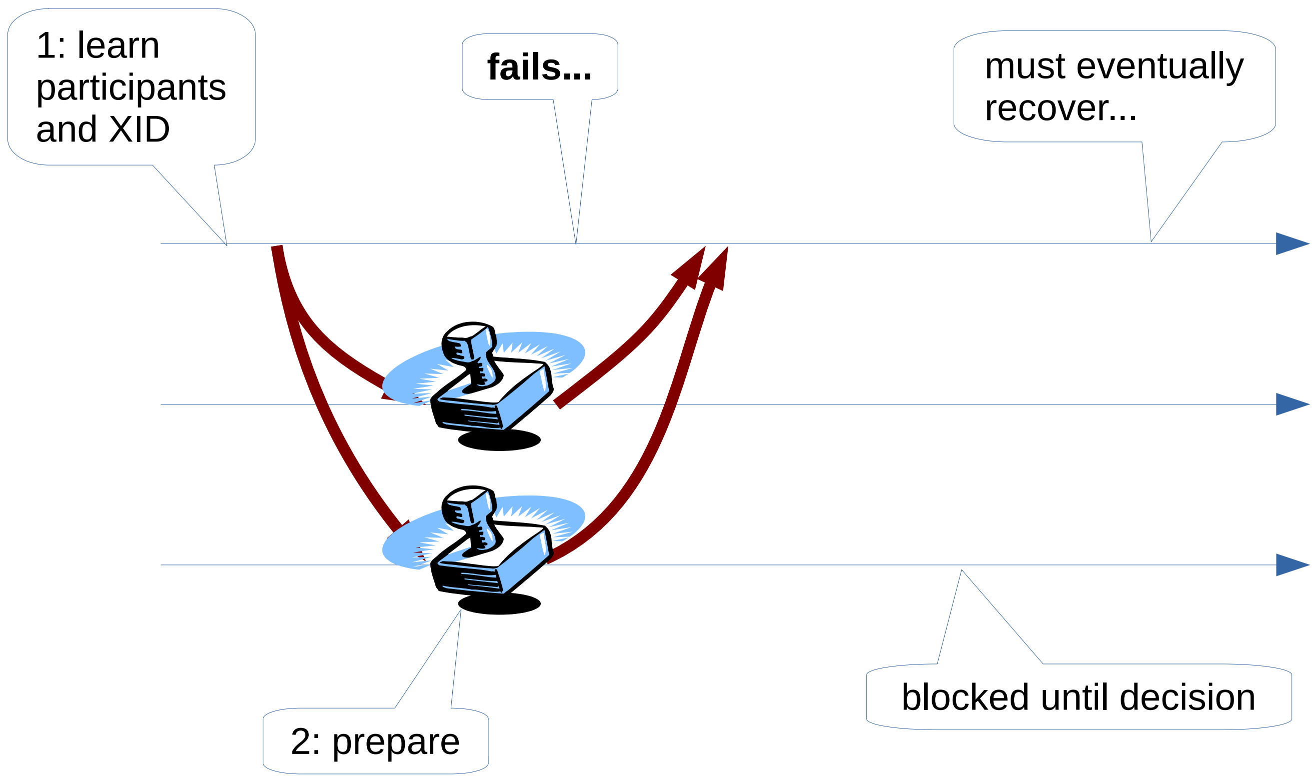
2-phase commit (2PC)



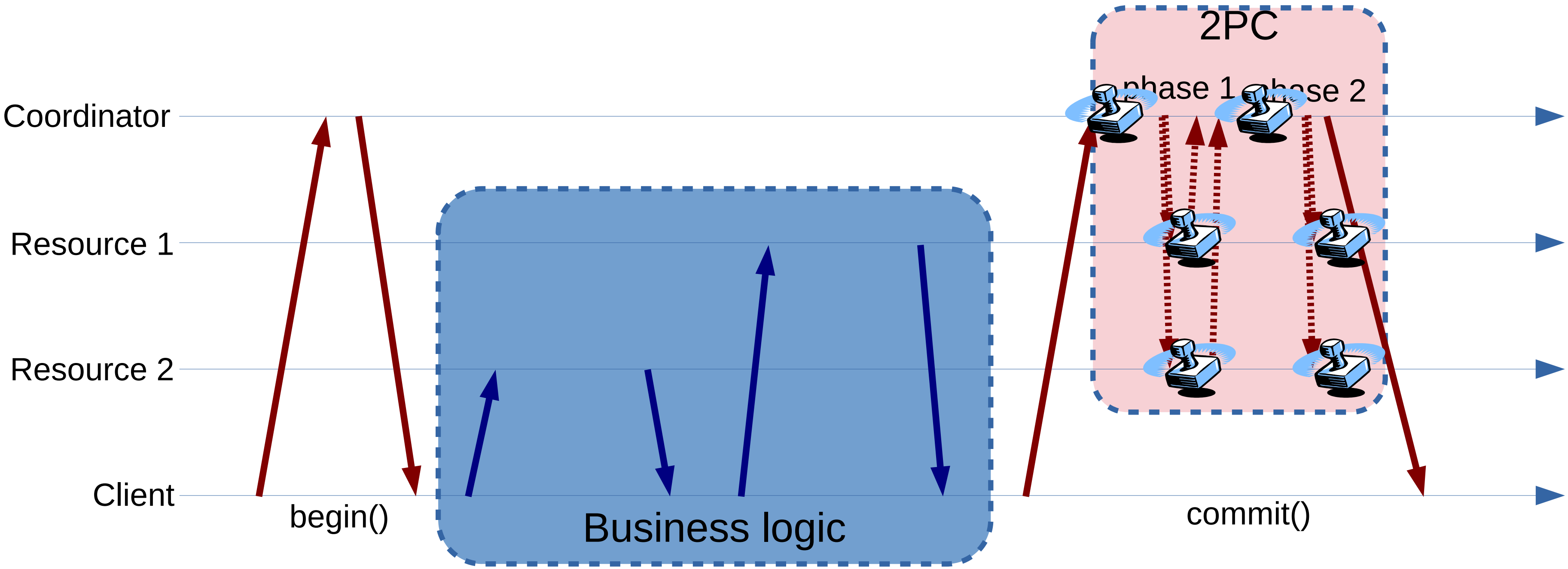
2PC: Participant failure



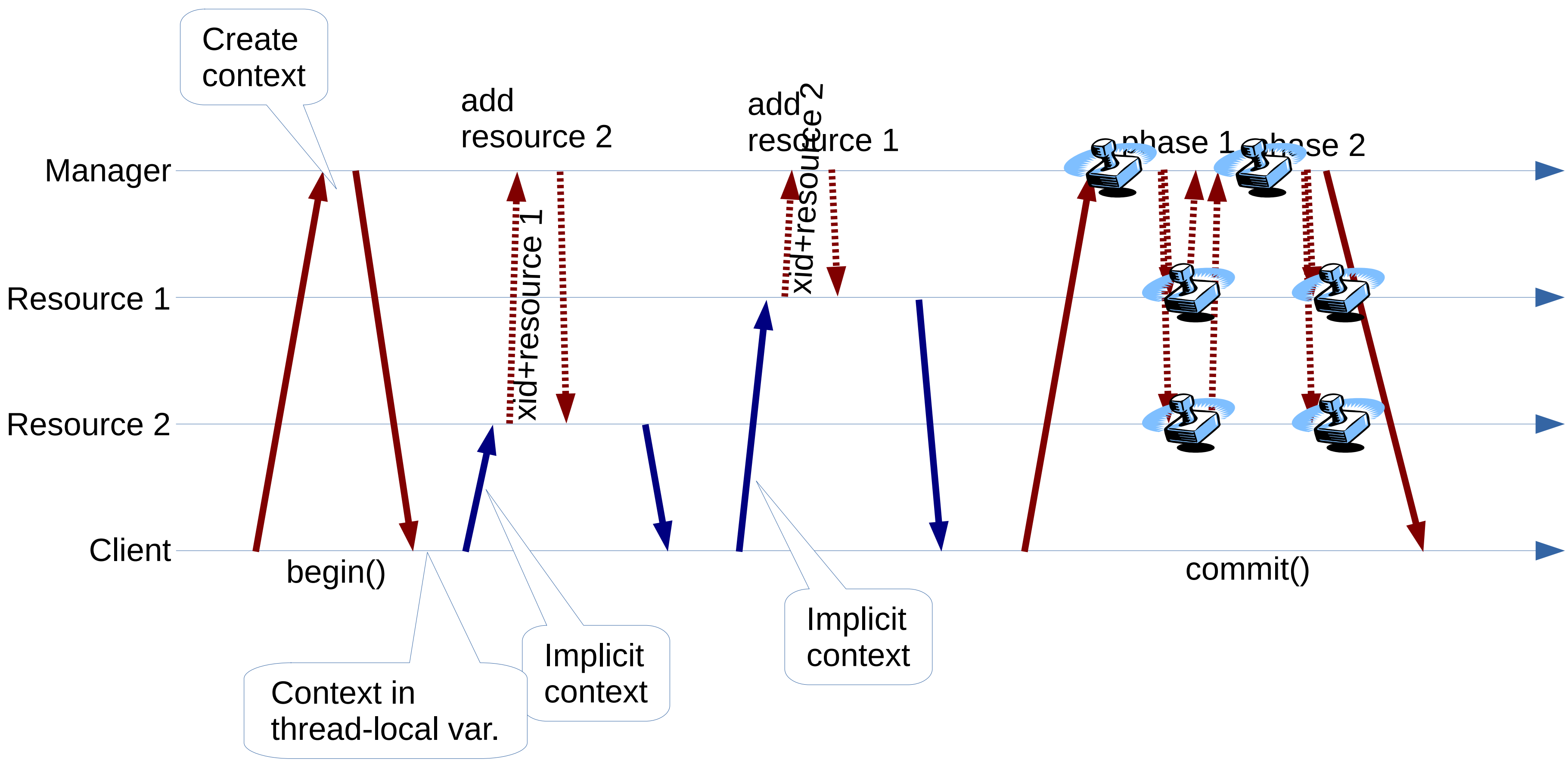
2PC: Coordinator failure



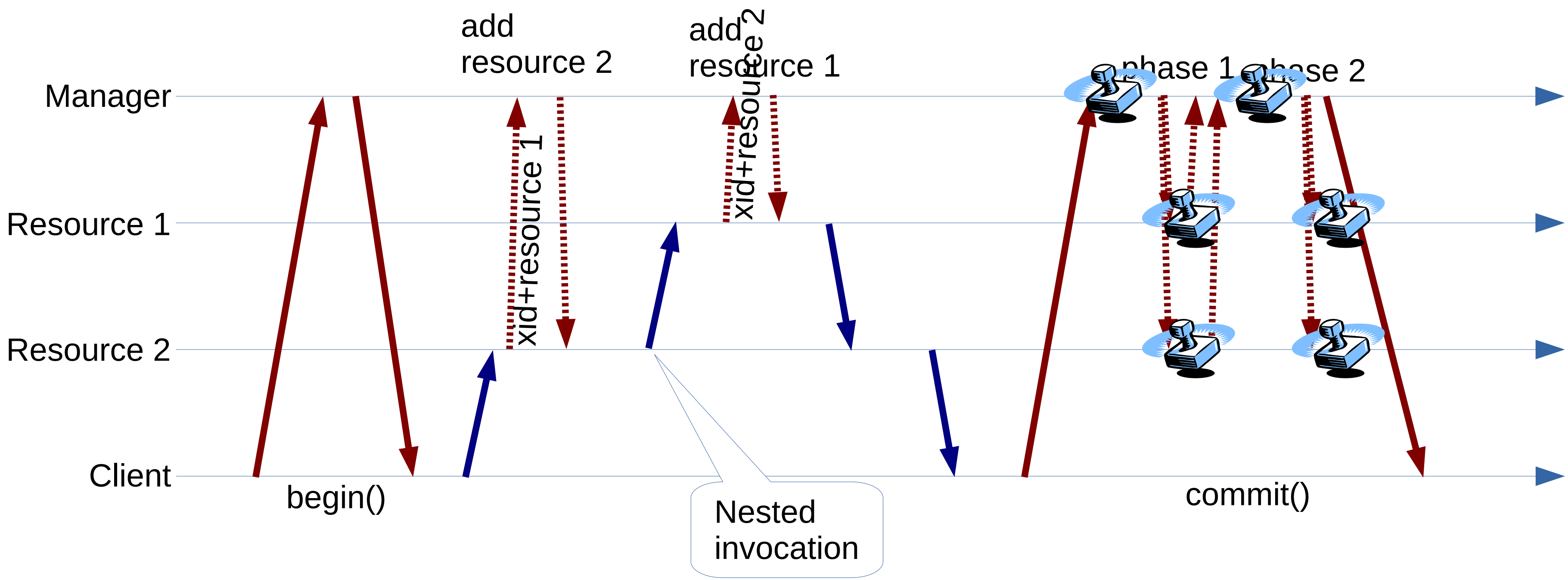
2PC in systems



Transactional RPC



Transactional RPC

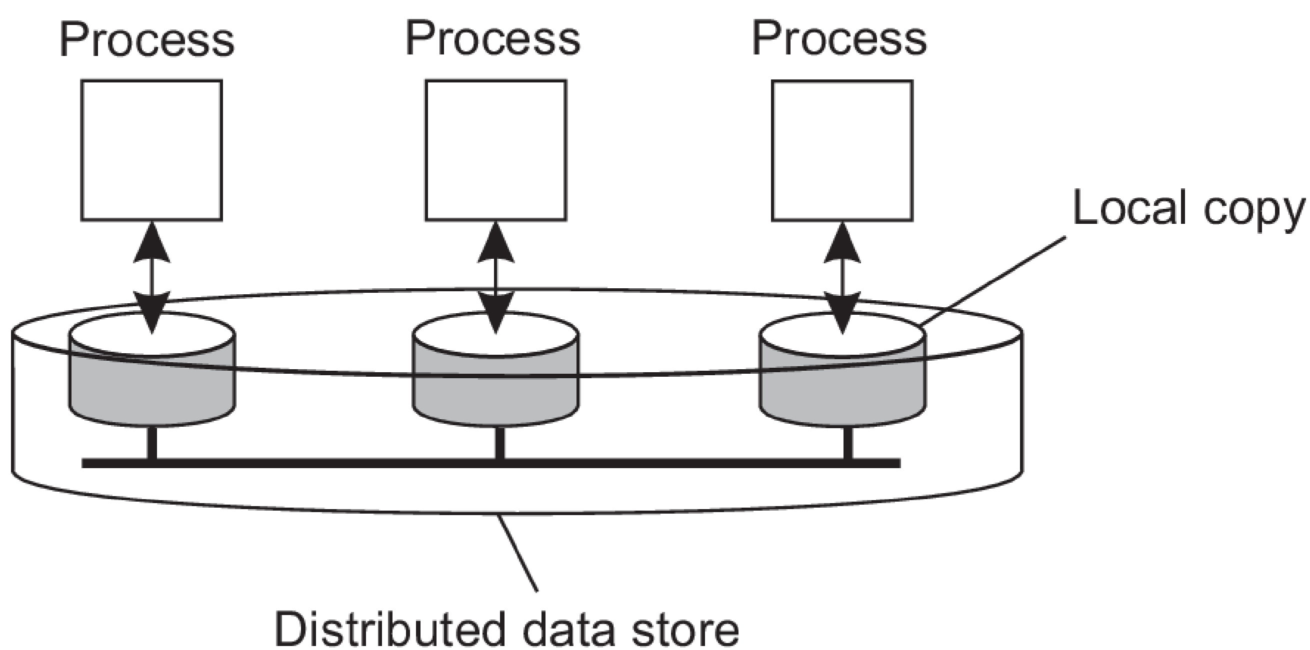


Summary

- Distributed transactions with 2-phase commit (2PC) support agreement in systems with faults
  - Limited to crash-recovery of the coordinator
- Is widely used in enterprise middleware for application integration

Replication

- Keep multiple copies of the same data or service
  - Distribute the load for scalability
  - Tolerate server faults

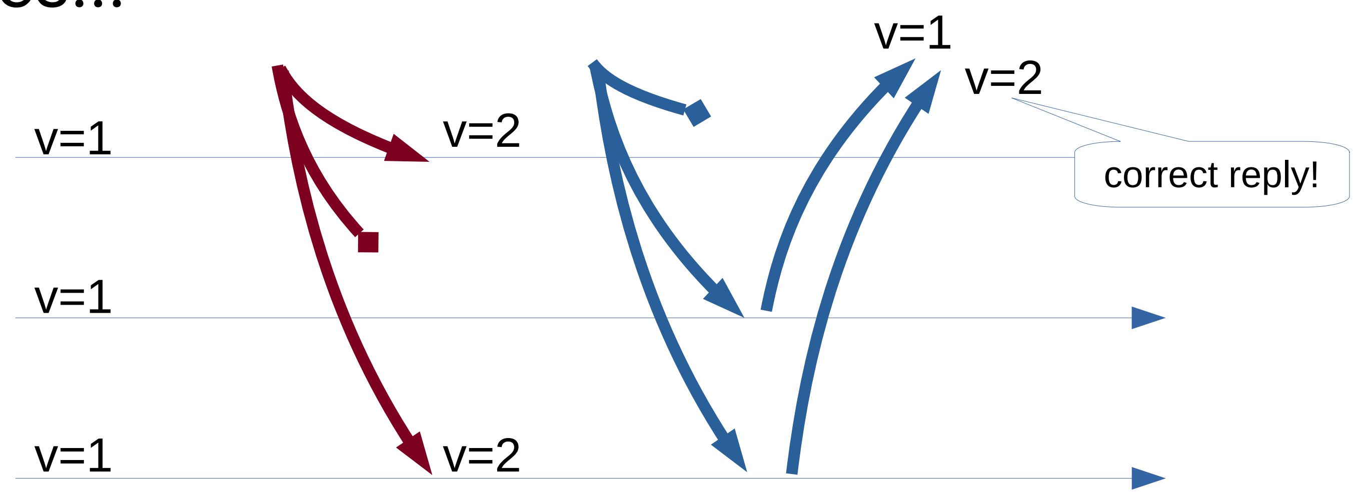


Replication

- Naive solution: write then propagate
  - state may diverge
  - clients observe paradoxes when reading
  - not fault-tolerant
- 2PC: Correct, but progress only with all up (tolerates reboots, not crashes)

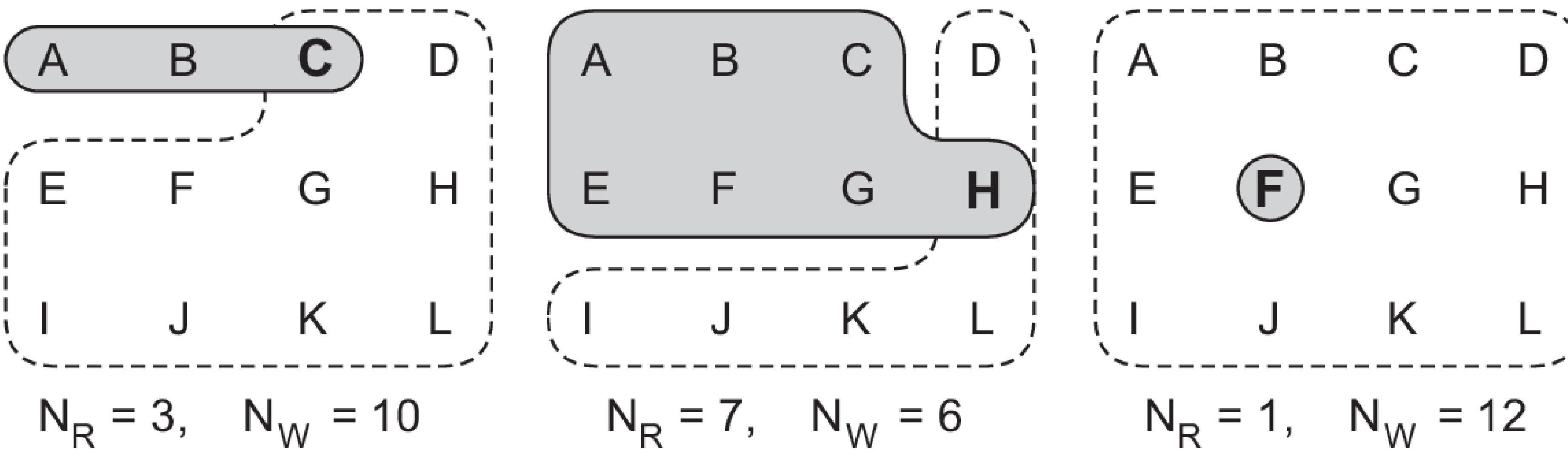
Replication

- Assume that:
  - operations are reads and writes
  - we keep a timestamp with each item
- It might be possible to read and write from fewer processes...



Quorum

- Assume 2-phase protocol for writing (phase 1 == read)
- Quorum rules for replicated data:
  - $N_R + N_W > N$ → readers get the latest value
  - $N_W > N/2$ → concurrent writers conflict



Quorum

- Additional rules for fault-tolerance when assuming at most *f* faults:
  - $N_R + f \leq N \rightarrow$ readers never block
  - $N_W + f \leq N \rightarrow$ writers never block
- Can be configured to ensure both or either of them
- Typical solution is having a majority:
  - $N_R = N_W = f + 1$
  - $N = 2f + 1$
- Examples: N=3 for f=1, N=5 for f=2, ...

Summary

- Flexible and efficient solution for data replication
  - Example: Amazon Aurora DB
- Wasteful when there are multiple concurrent writers:
  - At most one of multiple write operation can be accepted
  - But it can happen that none is accepted if each operation is applied in less than N_W servers

Leader election (Bully algorithm)

- A process that wants to be the leader (suspects that there is no leader), broadcasts its address to all others
- If address > local is received, acknowledge the new leader
- If address < local is received, broadcast local address (i.e. <u>bully</u> other candidates into submission)

- Transiently, there can be multiple or no leaders...

Consensus

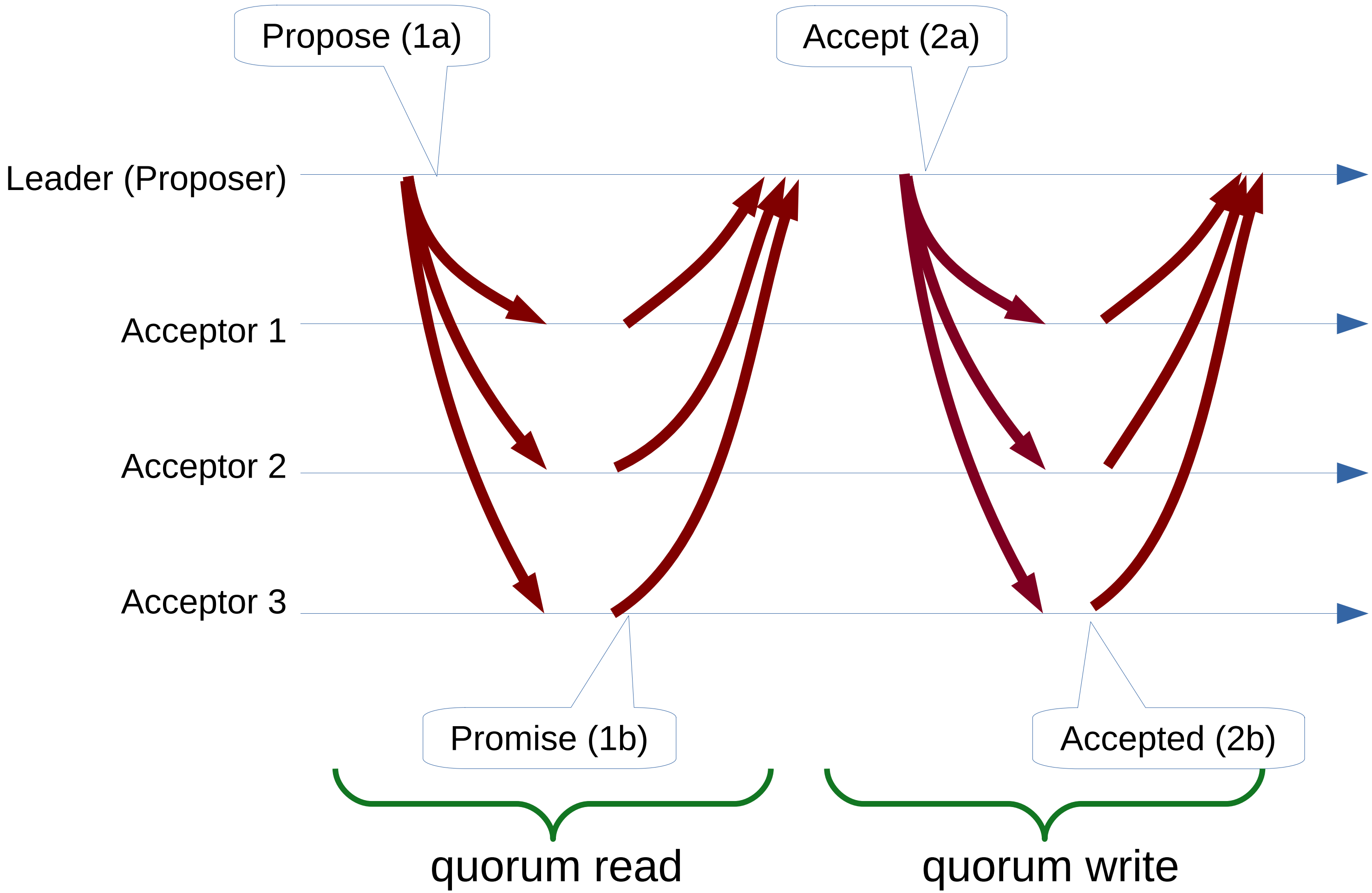
- Distributed agreement problem, in which processes propose alternatives and decide as follows:
  - All correct participants get a decision
  - All decisions are the same
  - The decision is one of the proposed alternatives
- How to solve this problem?

- 2PC + Quorum + Bully ~= Paxos consensus!

Paxos

- Processes in 3 main roles:
  - Proposers (with a Leader) → coordinate the voting
  - Acceptors → vote for the decision
  - Learners → are informed of the decision
- Usually, each participant plays all the 3 roles

Paxos



Propose (1a)
Accept (2a)
Leader (Proposer)
Acceptor 1
Acceptor 2
Acceptor 3
Promise (1b)
Accepted (2b)
quorum read
quorum write

Paxos

- Phase 1 is leader election + quorum read
  - Propose (1a):
    - A proposer tries to become the current leader
    - Uses (time,id) to bully others
  - Promise (1b):
    - The acceptor promises to forget previous leaders
- A Leader is established with a majority of promises and uses the value read or, if none, its own opinion as the candidate value

Paxos

- Phase 2 is quorum write:
  - Accept (2a):
    - The leader imposes its decision
  - Accepted (2b):
    - Accepted if the Leader was not deposed
    - The acceptor confirms the decision to Leader and Learners
- Upon receiving a majority of accepted messages, the decision is done!

Paxos

- Can always recover from a minority of failed processes if there is a trusted leader
- What's the worst case scenario?
  - No trusted Leader (unstable system)
  - More than a minority of faults (catastrophic fault)
- In both cases, the algorithm stops but never produces inconsistent decisions

Applications of consensus

- Replicated State Machine: Replicas execute the same sequence of commands
  - Clients issue commands to Proposers
  - Learners execute each request decided as the next and send replies to Clients
- Group communication: Agreement on operational processes that receive each message
  - The decision is on group membership and messages seen by each participant

Summary

- Consensus protocols at the heart of fault-tolerant distributed systems:
  - Container orchestration in Kubernetes
  - Transactions in Google Spanner database
- Consensus algorithms are available for even more demanding scenarios (Byzantine faults), where participants can misbehave, lie, …
  - Applicable to blockchains