Distributed Systems

ID2201

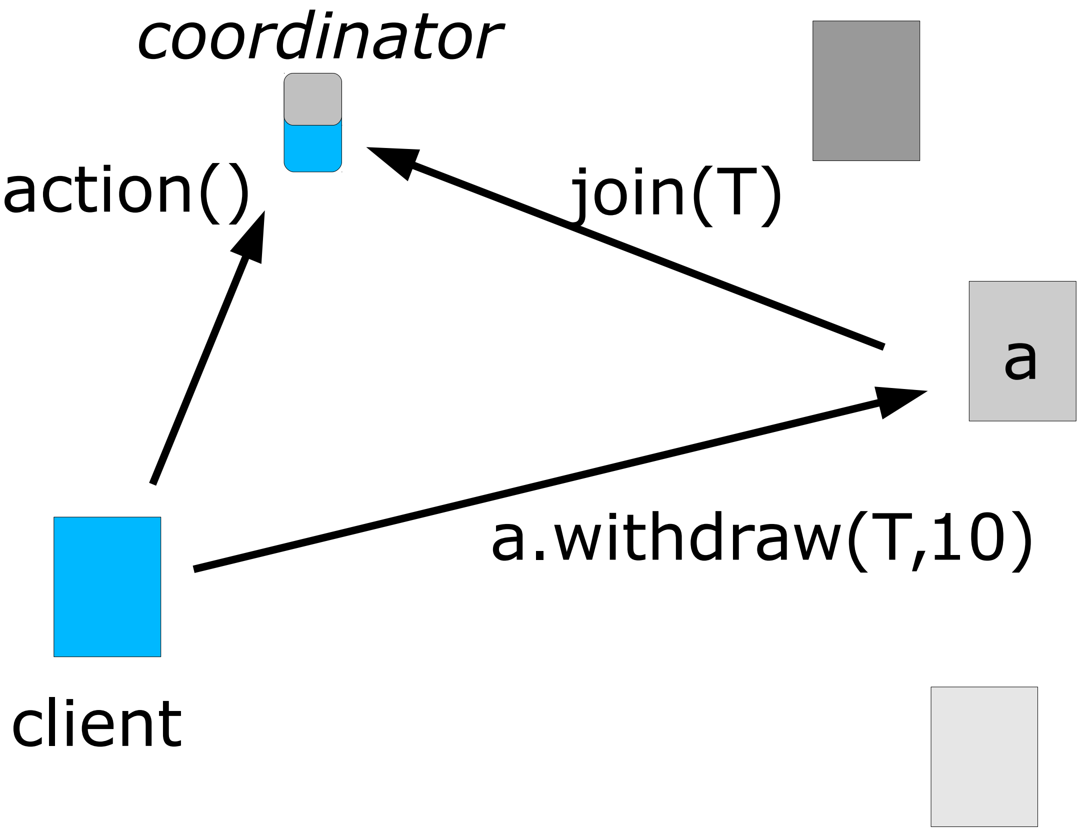
distributed transactions

Johan Montelius

Distributed transactions

- Problem:
  - Several independent transaction servers should be coordinated in one transaction.
  - How do we coordinate operations to guarantee serial equivalence?

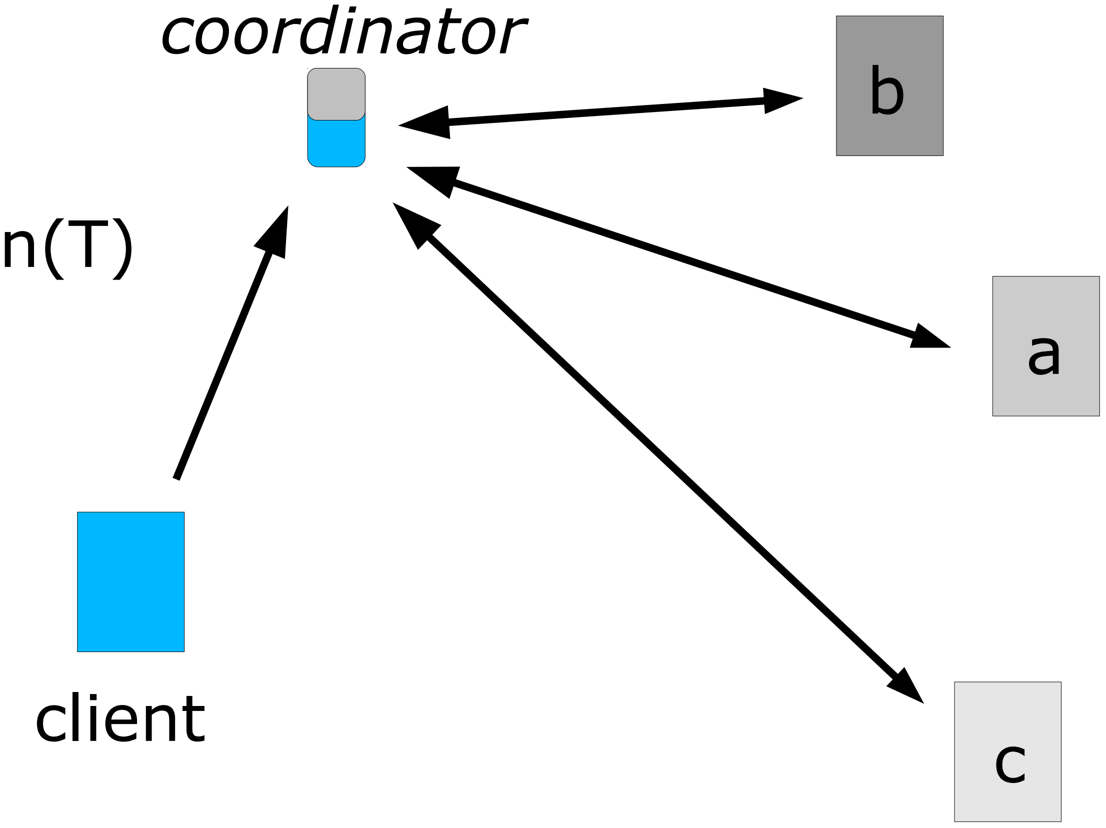
Coordination



```
T = openTransaction()
  a.withdraw(10)
  b.withdraw(20)
  c.deposit(30)
  closeTransaction(T)
```

Coordination

closeTransaction(T)



```
T = openTransaction()
  a.withdraw(10)
  b.withdraw(20)
  c.deposit(30)
  closeTransaction(T)
```

one-phase commit

- Client sends closeTransaction to coordinator.
- Coordinator tells participants to commit the transaction.
- Problems:
  - what if a participant can not commit and has to abort
  - a client could have crashed and have forgotten about the transaction

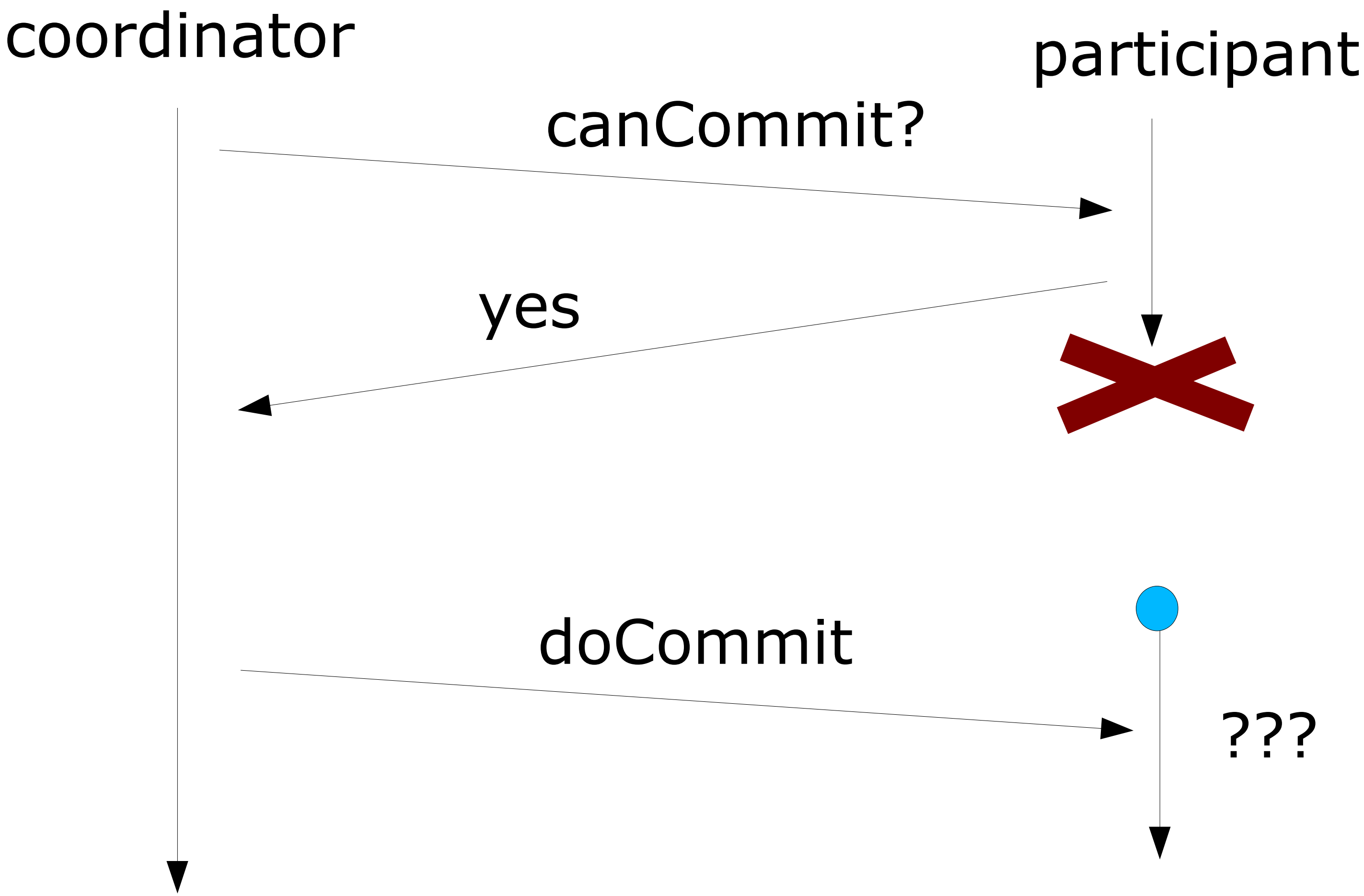
two-phase commit

- phase one: ask participants to vote for commit or abort
  - if voting for commit one has to be able to commit even after a node crash
- collect replies:
  - if anyone aborts all must abort
- phase two: inform all participants of the result
  - optionally participants acknowledge decision

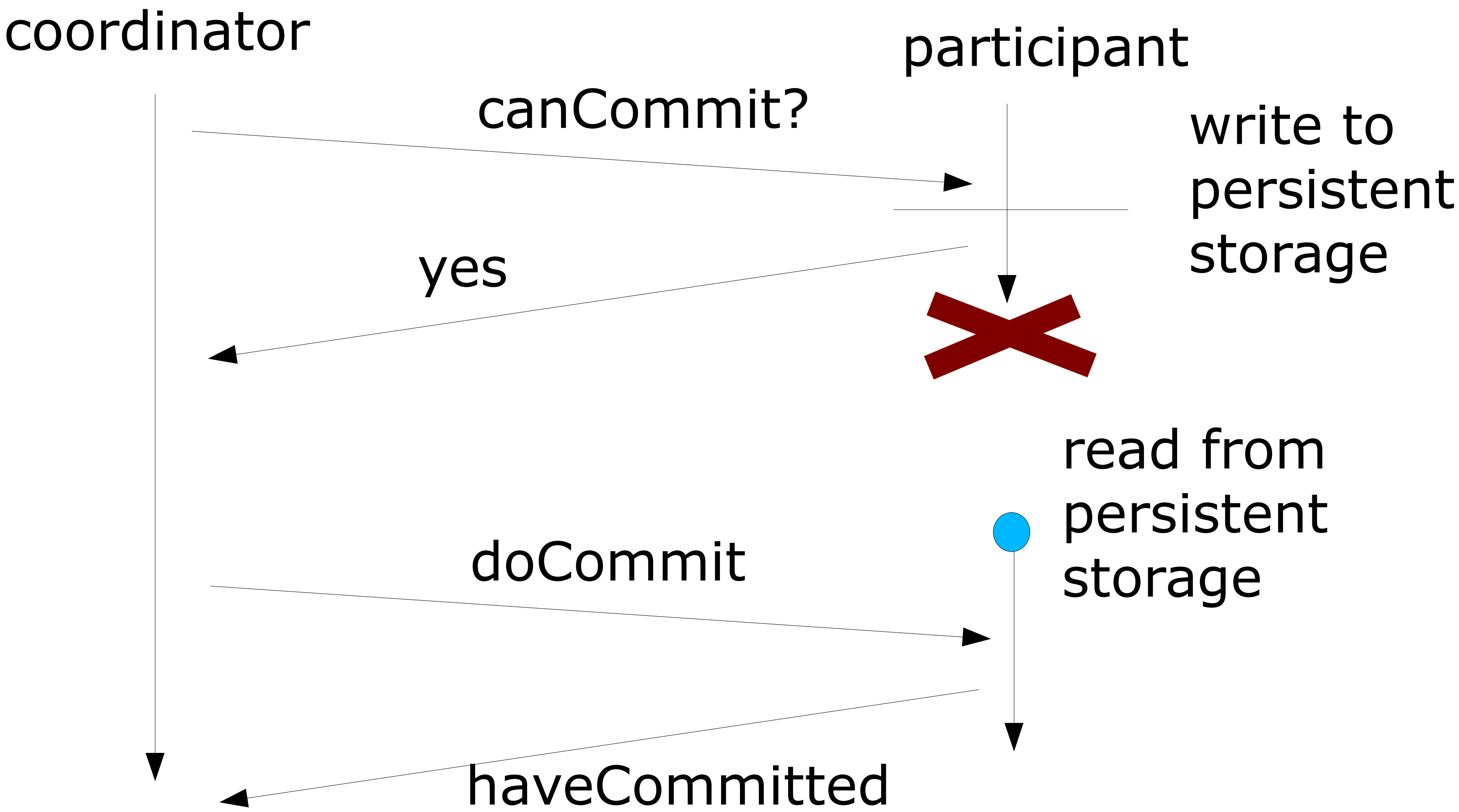
Consensus

- Two-phase commit is a consensus protocol but:
  - all clients must vote
  - if any client votes for abort we must abort

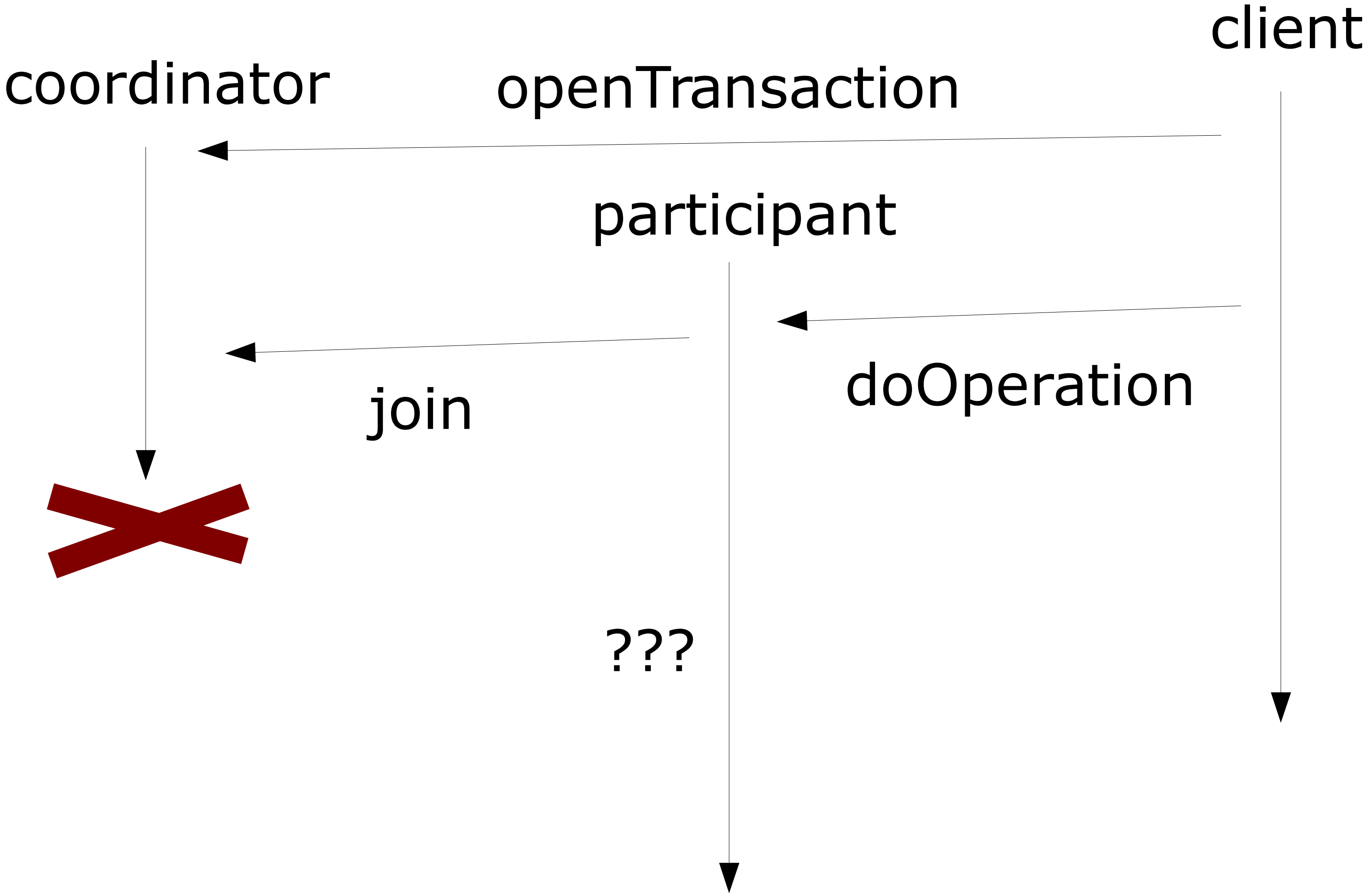
What if we crash?



What if we crash?



What if coordinator crashes



two-phase commit

- The protocol survives if nodes crash and later restart.
  - ...if they have written their state to persistent memory
- The protocol can be delayed waiting for any participant or the coordinator to reply.
- If successful:
  - all participants will commit or abort

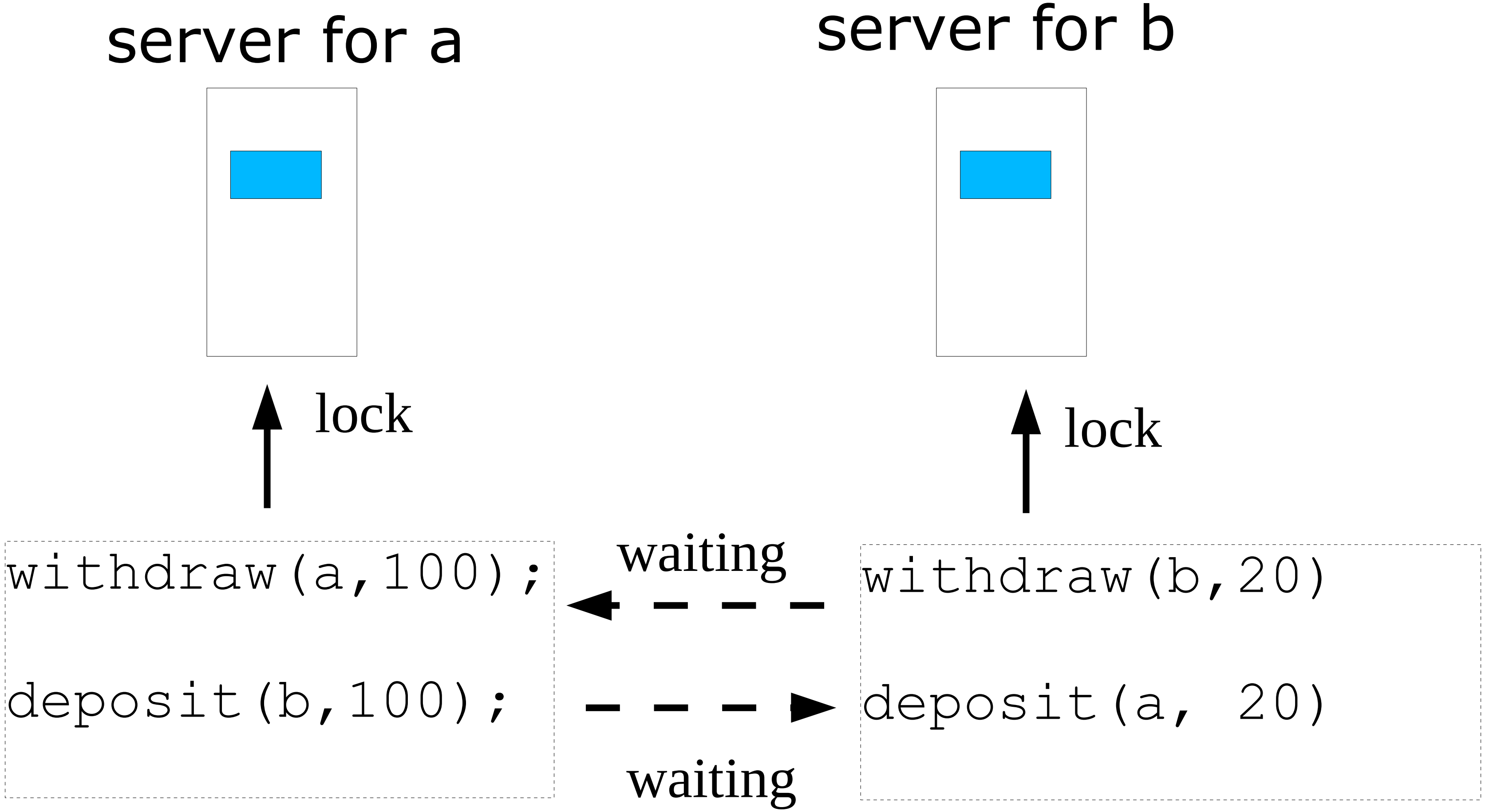
Distributed concurrency control

- Each server is responsible for concurrency control of its own objects.
- All participants must agree on order to guarantee *serial equivalence*.
  - If the operations of transaction T is before U in one server then all servers should have T before U.
- We can use: locks, optimistic control or time stamps.

Distributed locks

- Strict two-phase locking: locks are held until commit or abort.
- Can we prevent deadlock
  - harder to order all locks in the system
  - how do we synchronize taking of locks
- If each server maintain its own locks we will have distributed dead-locks.
  - detect and resolve rather than avoid

Deadlock



server for a

server for b

lock

lock

```
withdraw(a,100);

deposit(b,100);
```

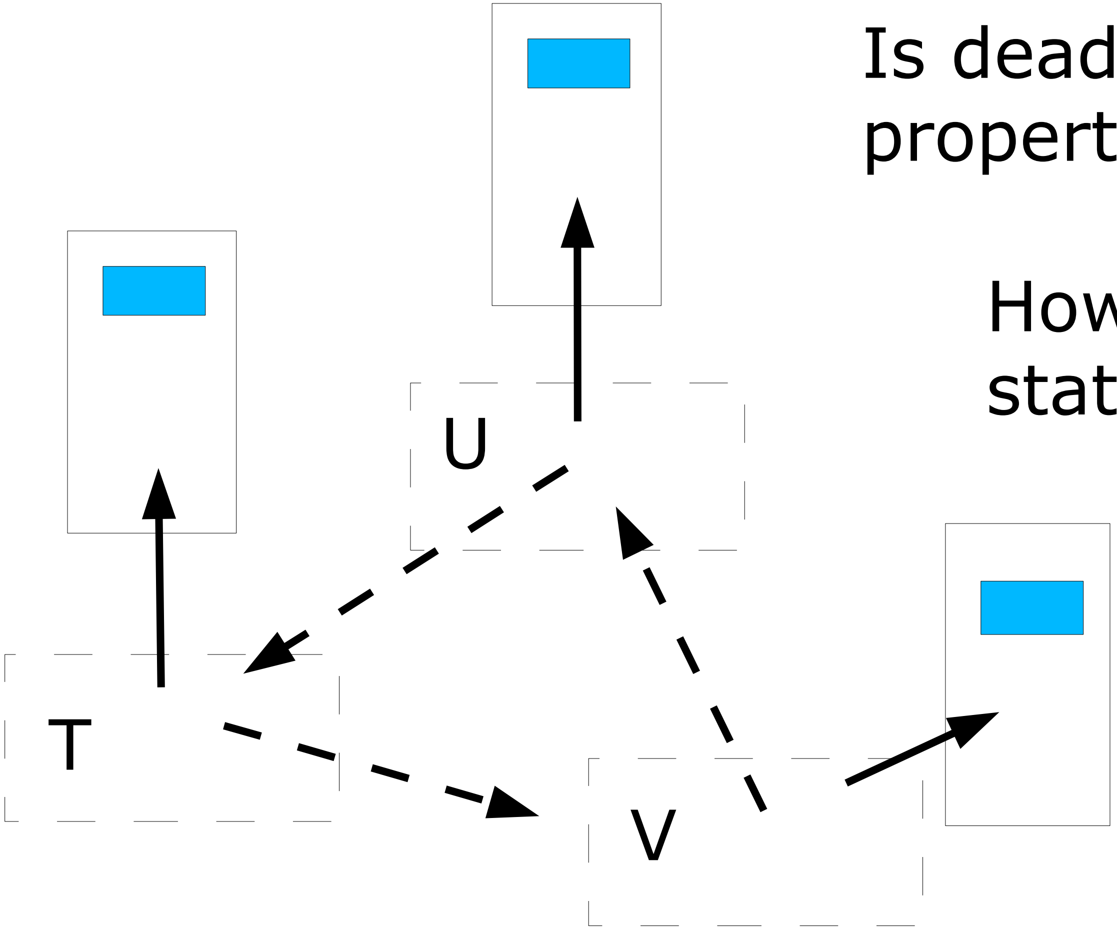
waiting

waiting

```
withdraw(b,20)

deposit(a, 20)
```

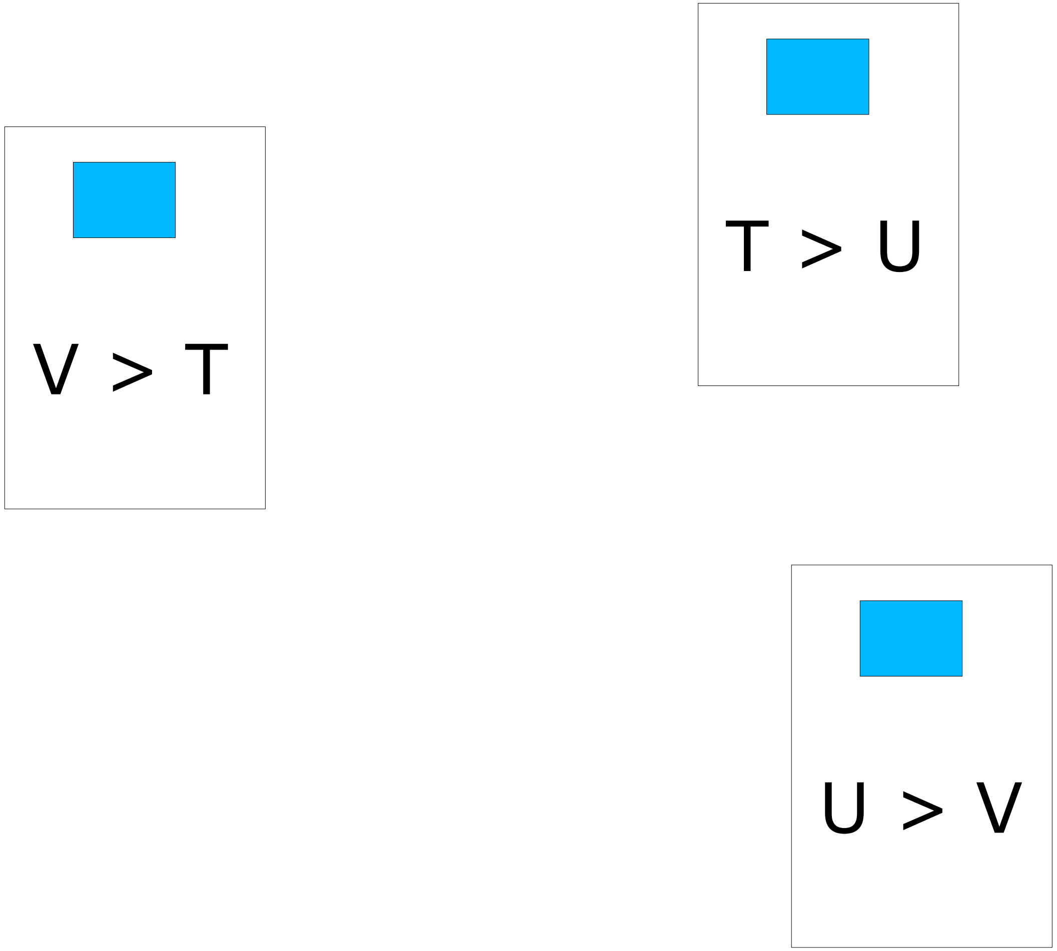
Distributed dead-lock



Is dead-lock a stable property?

How do we know the state of the system?

Wait for graphs



V > T

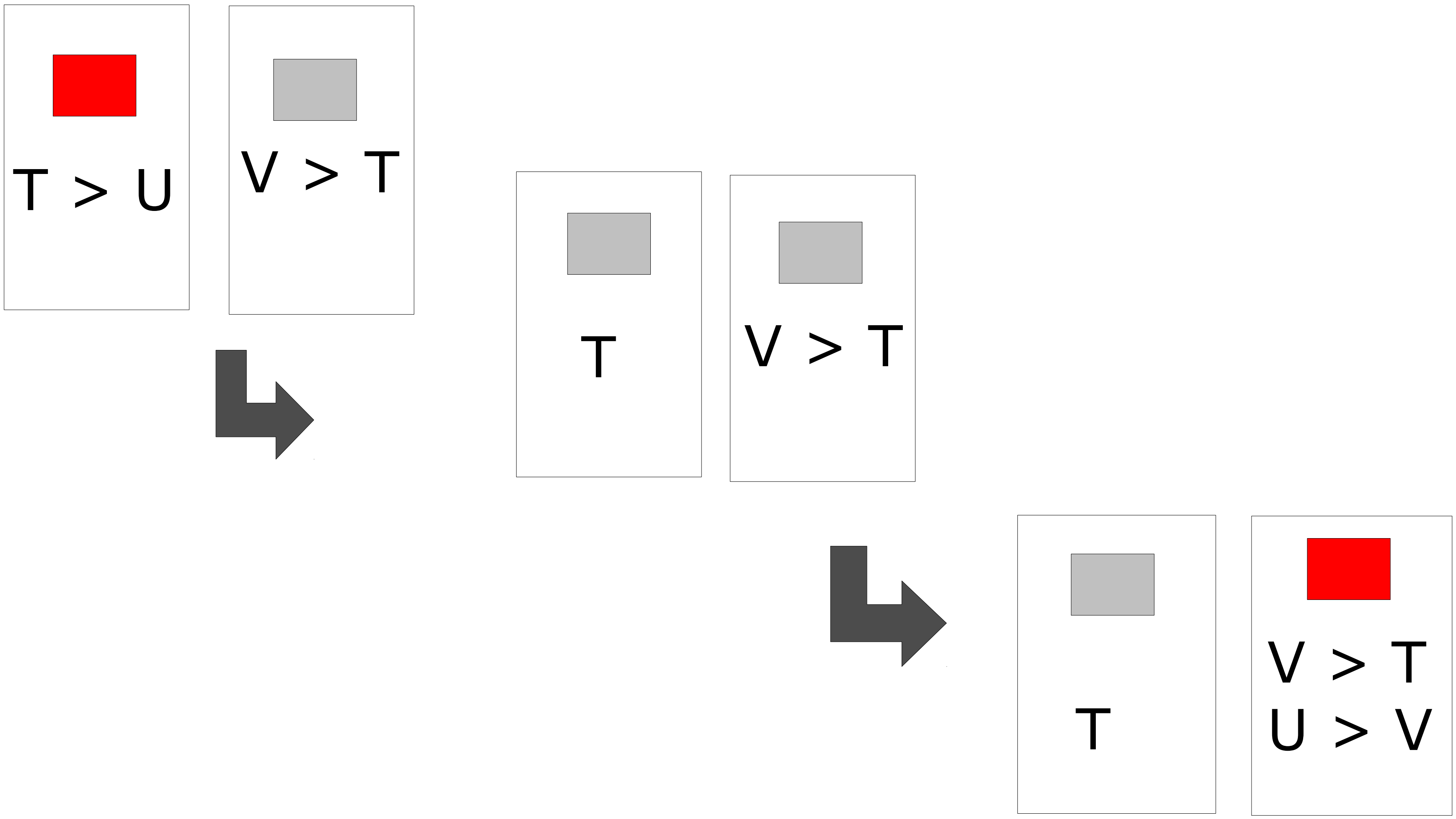
T > U

U > V

Centralized detection

- One server acts as a deadlock detector. Collects wait graphs from servers and tries to detect cycles.
  - what about messages in transit
  - how often should we collect sets
- What to do when cycle detected?
  - abort one transaction
  - which one?
- Can we falsely detect deadlocks?

Phantom deadlock



T > U

V > T

T

V > T

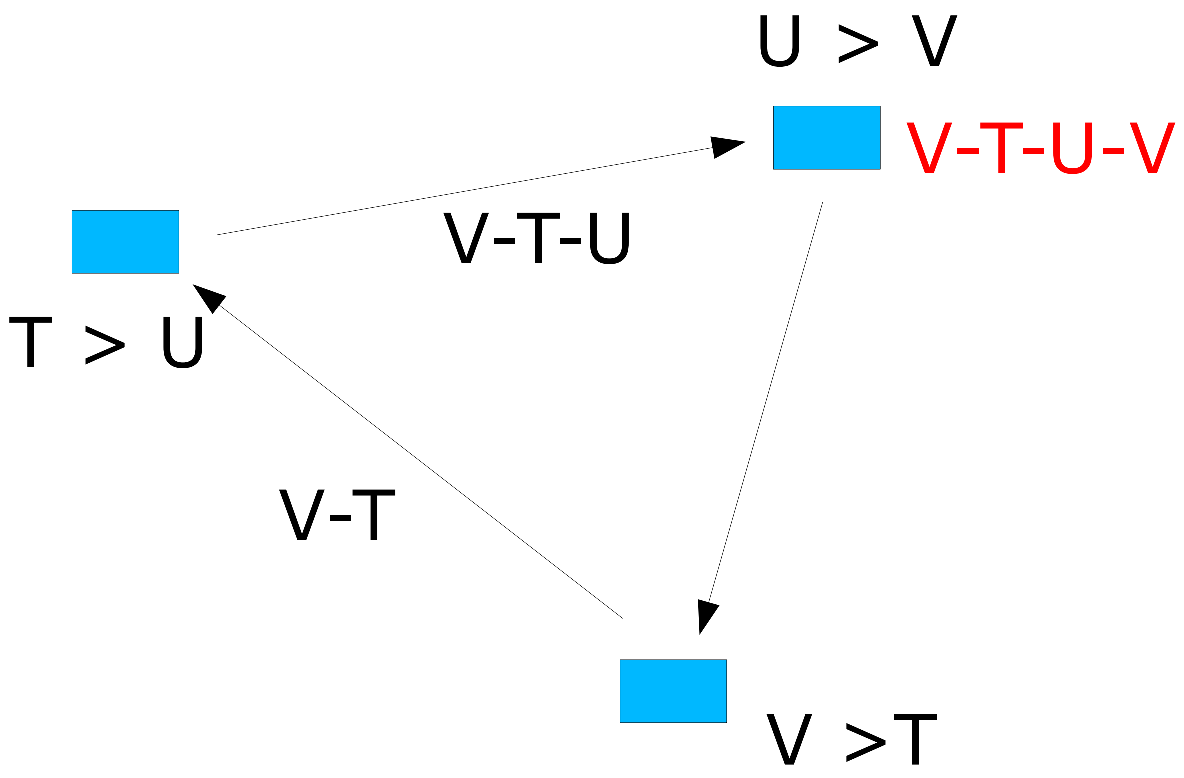
T

V > T
U > V

Probe the graph

- A different approach is to send a *probe* along the path of a wait graph.
- Probes must only be sent if the transaction is waiting for a lock held by a transaction that is also waiting for a lock.
- The probe consist of the wait graph detected so far

Probe the graph



Probe the graph

- In general deadlock cycles are small and do not generate long paths.
- We could have a situation where two probes are sent and the cycle is detected at two different points in the graph.
  - Could be resolved if transactions are ordered and both decide to abort the same transaction.

Optimistic concurrency control

- Commit only allowed after validation.
- Validation is a easier to implement as a sequential process and quite efficient if only one server is involved.
- Approaches:
  - Perform local validation and then check if we have global serial equivalence.
  - Assign a global transaction sequence number that all servers must use.

Time stamp control

- Assign a global time stamp at the start of the transaction.
  - Can clients be synchronized?
- Locally, the time stamp protocol acts as normal.

Summary

- Two-phase commit is used to provide distributed atomicity.
- Distributed deadlock is a problem.
  - How do we detect it?
  - How do we resolve it?