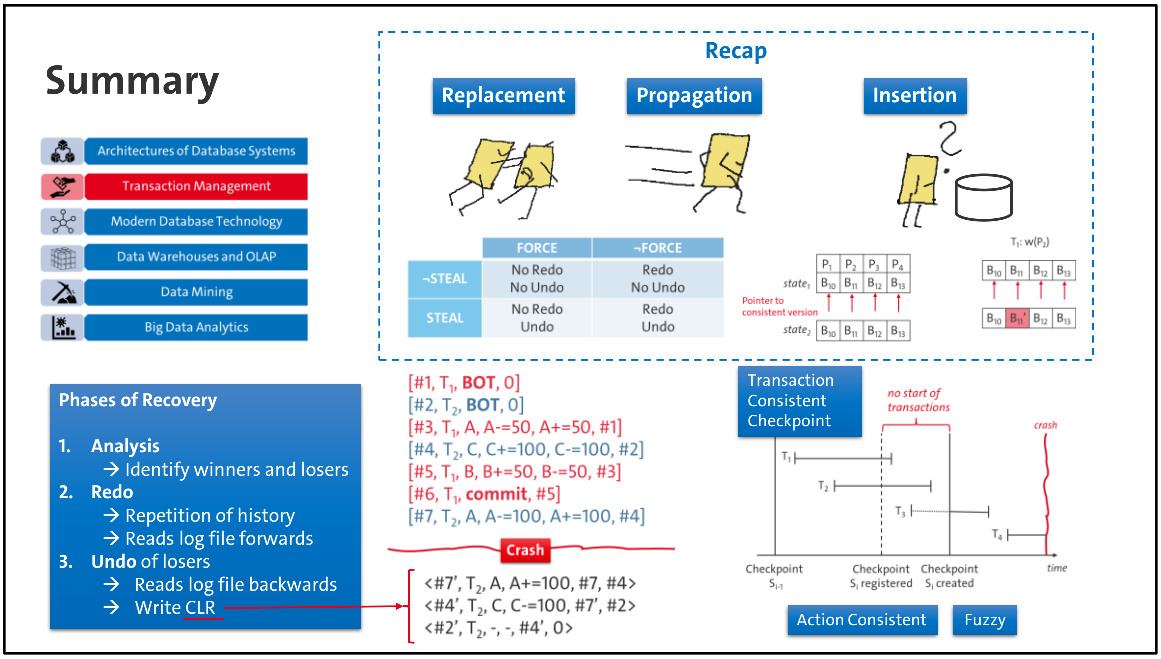
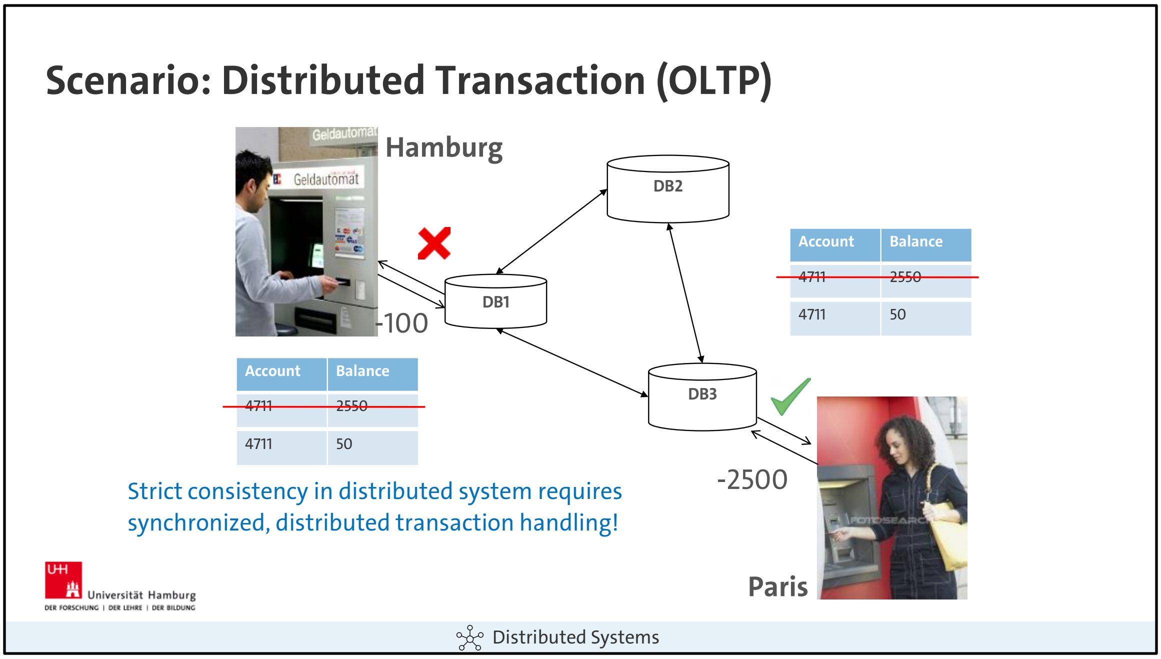

Summary
Architectures of Database Systems
Transaction Management
Modern Database Technology
Data Warehouses and OLAP
Data Mining
Big Data Analytics
Recap
Replacement
Propagation
Insertion
FORCE
¬FORCE
¬STEAL
No Redo No Undo
Redo No Undo
STEAL
No Redo Undo
Redo Undo
T_1: w(P_2)
P_1 P_2 P_3 P_4
$state_1$ B_{10} B_{11} B_{12} B_{13}
Pointer to consistent version
$state_2$ B_{10} B_{11} B_{12} B_{13}
B_{10} B_{11} B_{12} B_{13}
B_{10} B_{11}' B_{12} B_{13}
Phases of Recovery
1. Analysis
→ Identify winners and losers
2. Redo
→ Repetition of history
→ Reads log file forwards
3. Undo of losers
→ Reads log file backwards
→ Write CLR
[#1, T_1, BOT, 0]
[#2, T_2, BOT, 0]
[#3, T_1, A, A-=50, A+=50, #1]
[#4, T_2, C, C+=100, C-=100, #2]
[#5, T_1, B, B+=50, B-=50, #3]
[#6, T_1, commit, #5]
[#7, T_2, A, A-=100, A+=100, #4]
Crash
<#7', T_2, A, A+=100, #7, #4>
<#4', T_2, C, C-=100, #7', #2>
<#2', T_2, -, -, #4', 0>
Transaction Consistent Checkpoint
no start of transactions
crash
T_1
T_2
T_3
T_4
Checkpoint S_{i-1}
Checkpoint S_i registered
Checkpoint S_i created
time
Action Consistent
Fuzzy

Course Outline

- Architecture of Database Systems
- Transaction Management
- Modern Database Technology
  - Distributed Systems
  - Optimizations in RDBMS
  - NoSQL
- Data Warehouses and OLAP
- Data Mining
- Big Data Analytics

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG



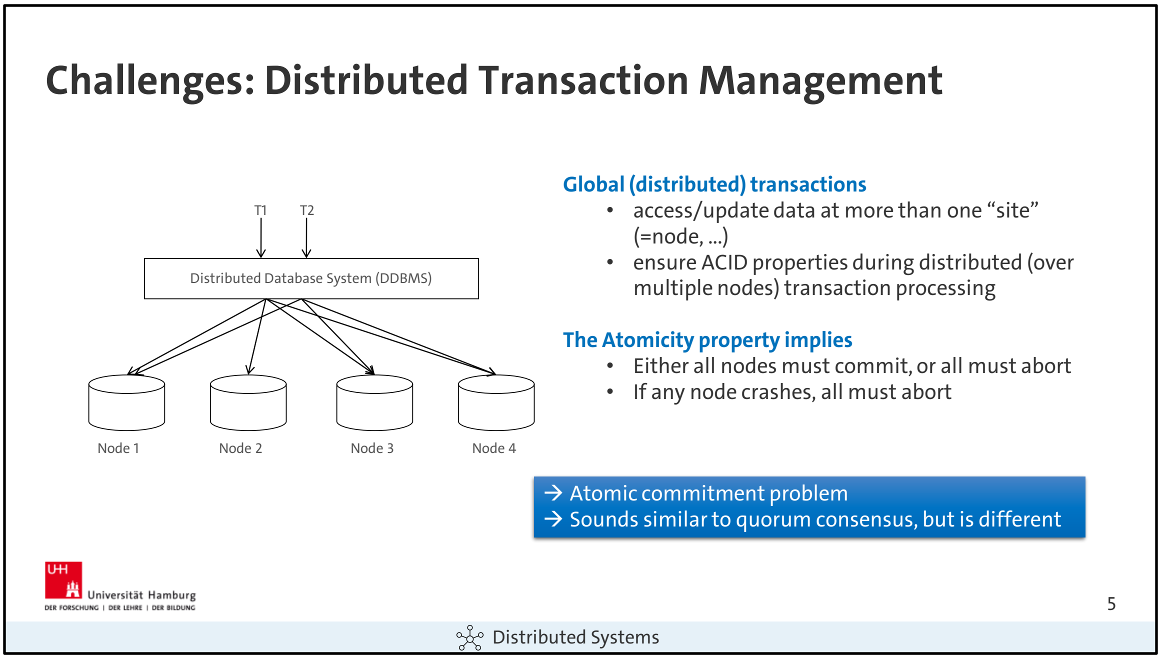
OLTP = **O**n**l**ine **T**ransaction **P**rocessing (not to be confused with OTP)

Scenario

- Both clients want to withdraw money at the same time from the same account but at different locations
- Since the balance is not enough to satisfy both requests, one should be denied
- If there was enough to satisfy both requests, both amounts would have to be subtracted from the account balance

Consistency can have different meanings in different contexts, e.g.

- Are distributed replicas of data in the same state?
- Do read operations on distributed data return the same result?
- Are application specific invariants met?



Recall atomicity in the context of ACID transactions

- A transaction either commits or aborts → Atomicity
- If it commits, its updates are durable → Durability
- If it aborts, it has no visible side-effects → Atomicity

Atomic commit vs. Consensus

<u>Consensus</u>

- One or more nodes propose a value
- Any of the proposed values is decided
- Crashed nodes can be tolerated as long as a quorum is working
- There can be different quorum for read and write
- Quorum = Minimum number of nodes in the system that have committed

<u>Atomic commit</u>

- Every node votes whether to commit or abort
- Must commit if all nodes vote to commit, must abort if at least one node votes abort or a participating node crashes

Challenges: Failures in Distributed DBMS

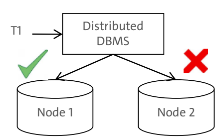
Site Failure

- A node is not available anymore
- Completely or partial

Communication Failure

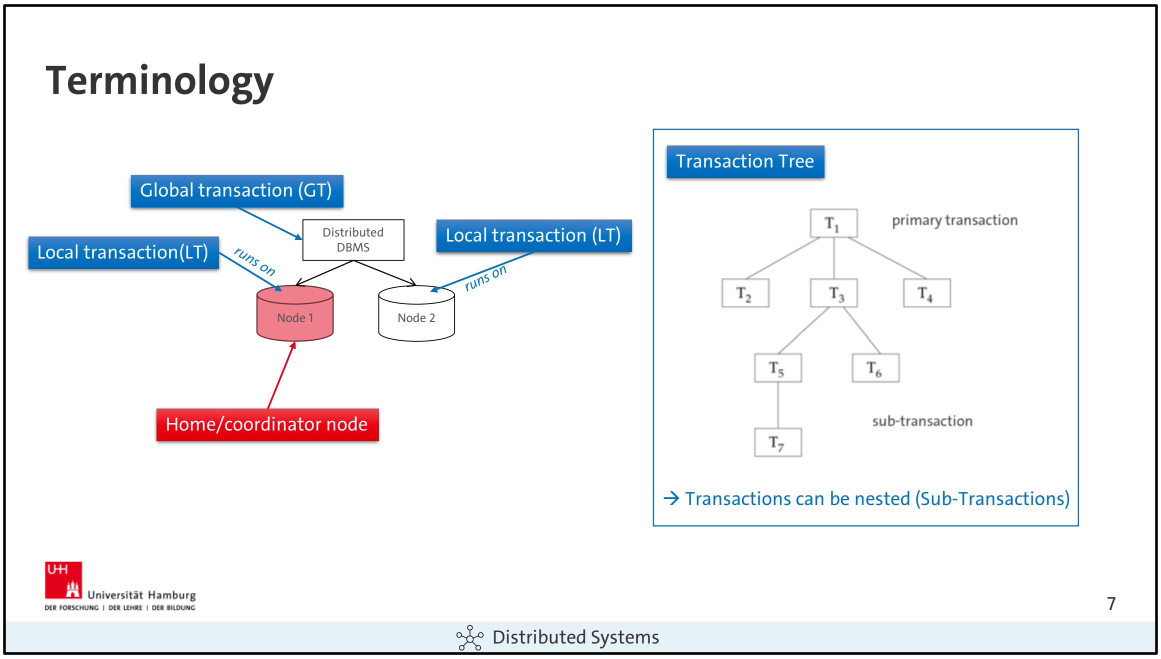
- Failure of a communication link
- Loss of messages/Message corruption/ Message duplication
- Network partitioning

→ Network partitioning and site failures are generally indistinguishable



Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

- A network is defined as being partitioned when it has been split into two or more subsystems that lack any connection between them
- Loss of messages/Message corruption/Message duplication → Handled by network transmission control protocols such as TCP-IP
- Failure of a communication link → Handled by network protocols, by routing messages via alternative links

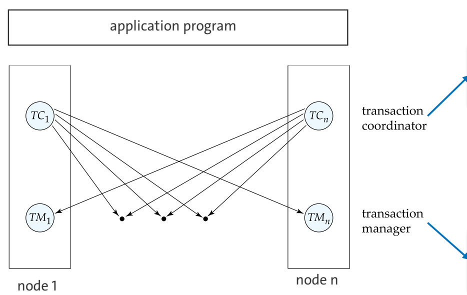


Home/Coordinator node: Node where transaction is started
Local Transaction: Transaction that is completely executed on home node
Global Transaction: Transactions that are executed on multiple nodes
Call Hierarchy: Transactions calling each other form a directed, acyclic graph → Transaction tree

Rules for nested transactions

- A transaction may only end, after all sub-transactions are closed
- Every sub-transaction decides independently to finally abort or temporary commit
- In case of an abort of a transaction, all its sub-transactions are aborted as well
- In case a sub-transaction aborts, the parent transaction decides on the final outcome, i.e., may change an abort of a sub-transaction into a global commit
- Sub-transactions may only finally commit after all parent-transactions on the path to the root transaction have finally committed

Transaction Coordinator and Manager

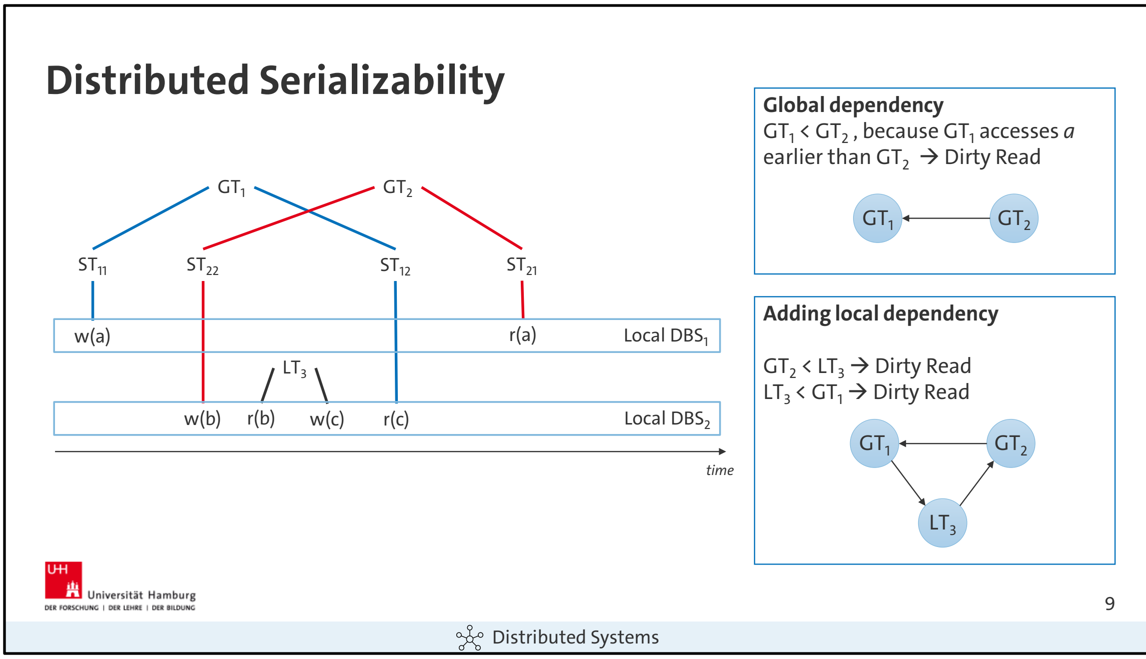


Coordinates overall execution using transaction managers at each site

Executes all local operations, especially local commits and aborts

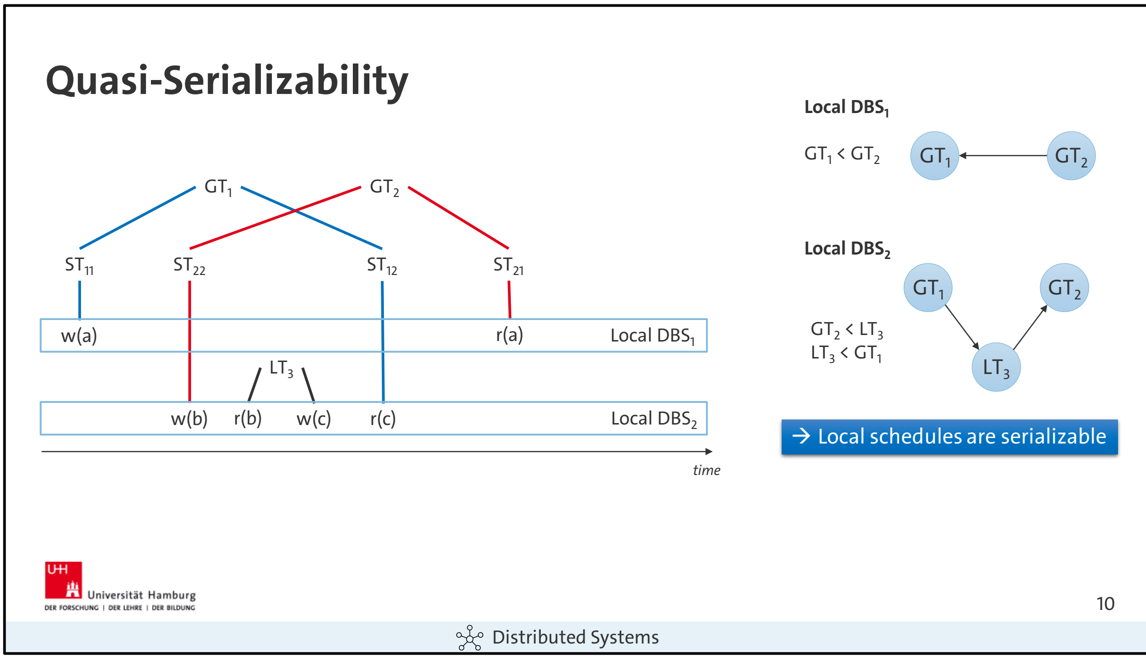
Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

- Each site (node) has a transaction ccordinator and a local transaction manager
- Global transactions can be submitted to any transaction coordinator



Some Solutions

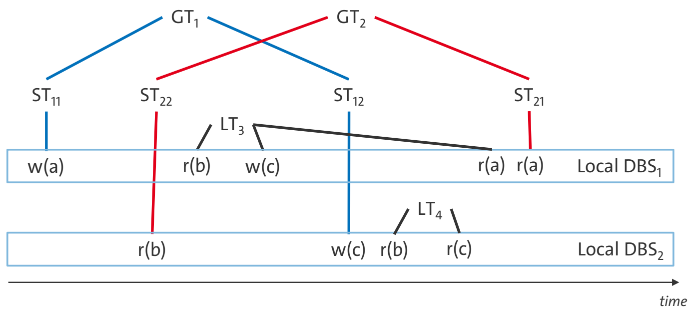
- Implement functional limitations
  - Example: restricted set of update capabilities within global transactions
- Provide non-functional limitations
  - Example: relaxing ACID guarantees, e.g. no guarantee of global serializability
- Reduction of autonomy of local systems
  - Do not allow local transactions
  - Mark transactions as global and local and reflect this knowledge within the scheduler
- Reduction of heterogeneity
  - Agree on using compatible synchronization and commit protocols

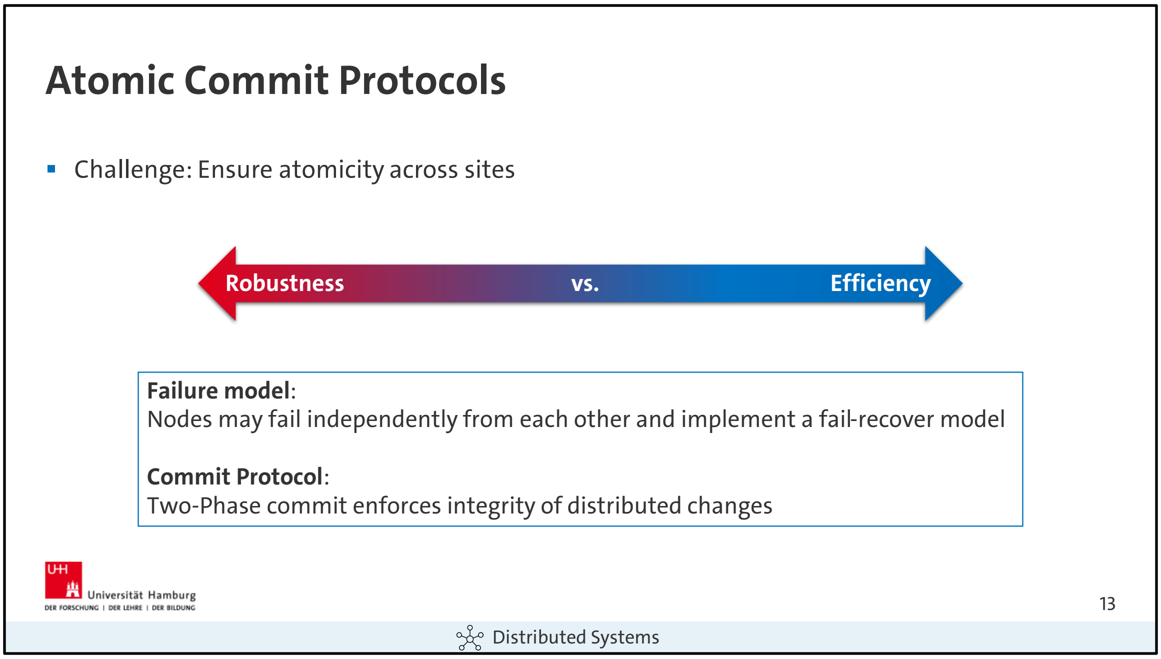


Quasi-Serializability

- Implementation of weaker correctness criteria
- All local schedules are fully serializable
- Global transactions are executed sequentially

Exercise (Quasi)-Serializability





Challenge: Global Atomicity / Single Decision

- All nodes participating in a distributed transaction have to come to a common decision: Either all commit or all abort the transaction
- Atomic commit protocols are used to ensure atomicity across sites → A transaction, which executes at multiple sites, must either be committed at all the sites, or aborted at all the sites
- COMMIT → global transaction is written back on **all** (relevant) nodes
- ABORT → global transaction is rolled back on **all** (relevant) nodes

Conflict: Robustness vs. Efficiency

- Correctness
  → Atomicity (all-or-nothing) and durability have to be guaranteed also in case of failure
- Efficiency
  → Generate as few messages and log entries as possible
- Robustness
  → Non-blocking
  → Each node should be allowed to abort as long possible

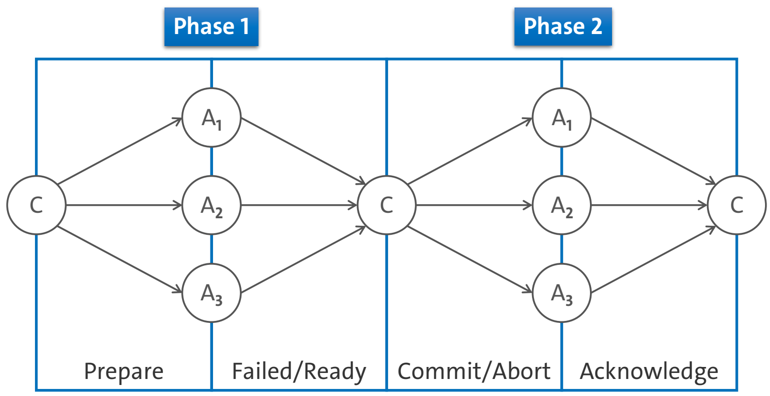
Fail-recover model

- Nodes could crash and at some later date recover from the failure and continue executing normally
- Protocols able to tolerate multiple malicious sites are called byzantine fault-tolerant protocols (which is a research area of its own)

Further reading:

- An example for byzantine fault-tolerant middleware: Luiz, Aldelir Fernando, Lau Cheuk Lung, and Miguel Correia. "Mitra: Byzantine fault-tolerant middleware for transaction processing on replicated databases." *ACM SIGMOD Record* 43.1 (2014): 32-38.
- An early approach to Byzantine Fault-Tolerant Services (not database-specific): Abd-El-Malek, Michael, et al. "Fault-scalable Byzantine fault-tolerant services." *ACM SIGOPS Operating Systems Review* 39.5 (2005): 59-74.
- A recent approach on byzantine fault detection: Yamada, Hiroyuki, and Jun Nemoto. "Scalar DL: Scalable and practical Byzantine fault detection for transactional database systems." *Proceedings of the VLDB Endowment* 15.7 (2022): 1324-1336.

Two-Phase-Commit Protocol (2PC)



A_i ...Participant
C ...Coordinator

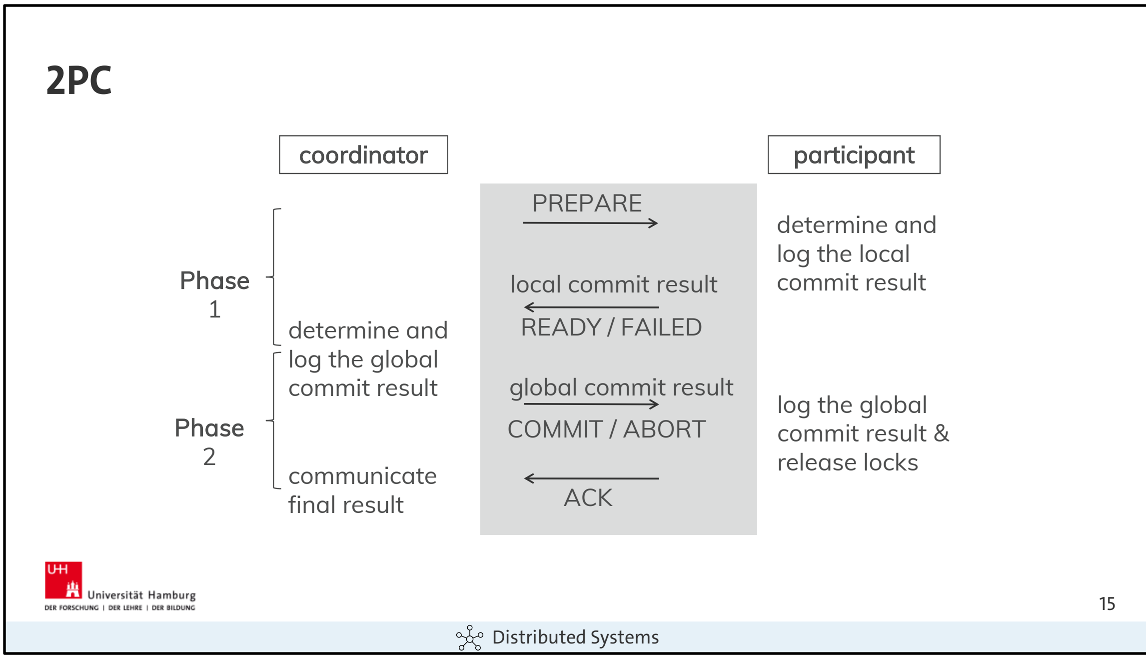
- 2PC can be seen as a special case of 2PL for distributed systems
- Global commit is initiated by the transaction coordinator after the last operation of the transaction has been executed
- The protocol involves all the local sites at which the transaction is executed

Phase 1 (PREPARE)
- Assure ability to execute a transaction
- Log changes and commit (After-Image)

Phase 2 (COMMIT)
- Make changes visible (free locks, delete temp log (UNDO))
- After phase 1, the transaction has a guaranteed outcome

Illustrated Example
- n=4 Participants (includes the coordinator)

→ 4 messages per participant
→ 4(n-1) messages in total



Step 1: PREPARE

- Coordinator sends PREPARE message to all participants to find out if they can commit the transaction

Step 2: READY/FAILED

- Each participant receives the PREPARE message and sends one of the two messages back

    READY → If participant is able to commit the transaction, locally
    FAILED → If participant is not able to commit the transaction, locally (because of failure, consistency, ...)

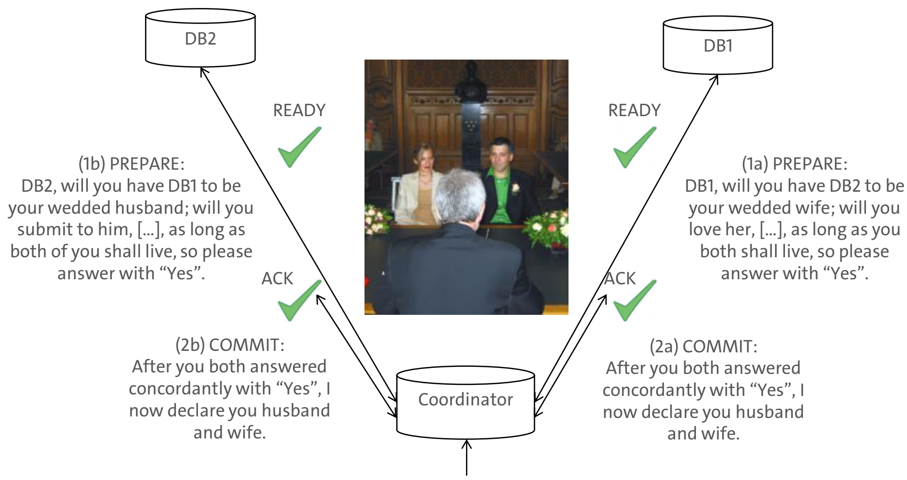
Step 3: COMMIT/ABORT

- If coordinator receives READY back from all participants, the coordinator decides for global COMMIT
- If one of participants answers FAILED or does not answer in time (timeout), coordinator must decide for ABORT
- Coordinator sends decision to all participants for them to COMMT/ABORT locally

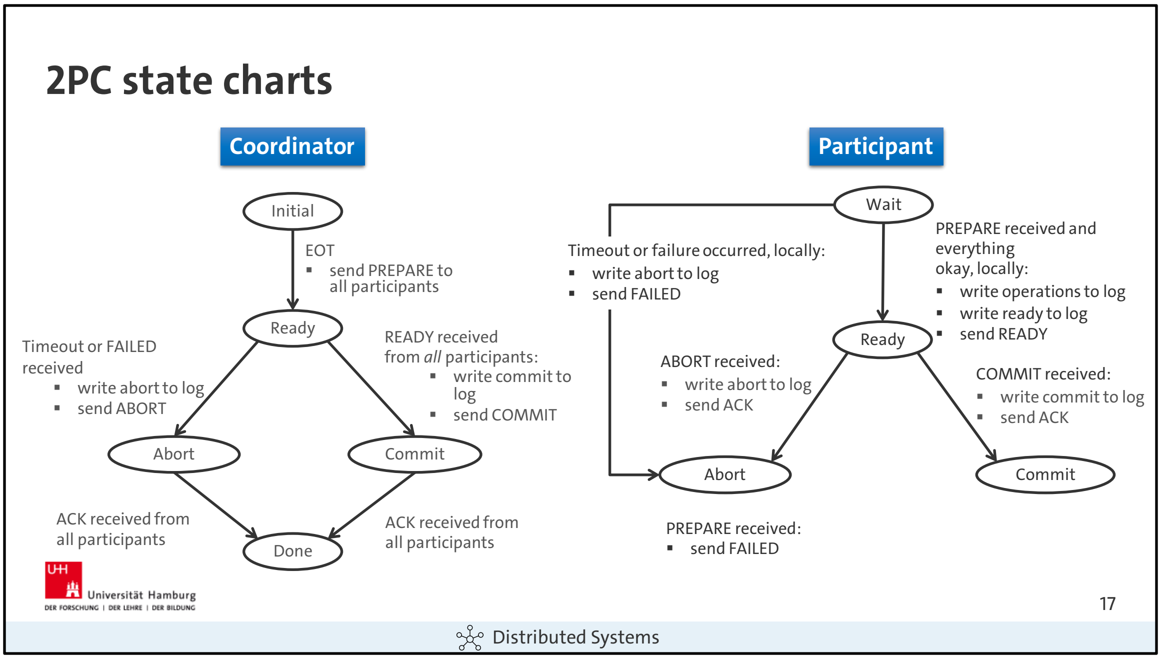
Step 4: ACKNOWLEDGE

- Participants acknowledge reception of coordinator's global decision

Example: “The 2PC Wedding”



Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG



In case of Failures

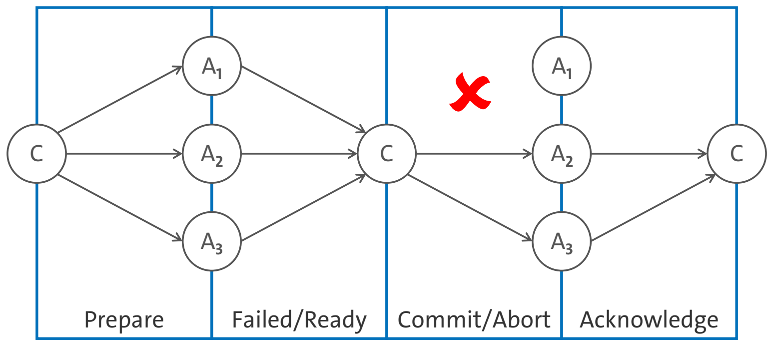
Phase of uncertainty for participants

- From the point a participant replied with READY until the global decision is received
- Participant could actively try to get hold of global decision
  → actively ask coordinator or other participants

Reason of uncertainty situations

- Crash of coordinator
- Crash of other participant (coordinator waits for reply)
- Lost messages

Failure Scenario: Lost Message

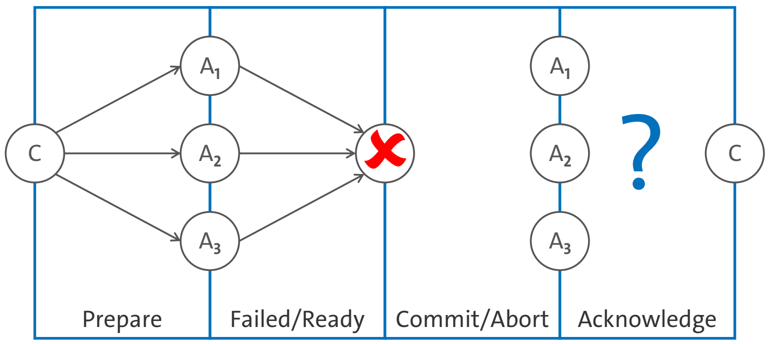


Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Lost in Phase 1
PREPARE message of coordinator gets lost
or
READY/FAILED message of a participant gets lost
→ after timeout coordinator assumes a participant crash and decides for ABORT

Lost in Phase 2
ABORT/COMMIT message of coordinator is lost, while participants are in READY state
→ participants are blocked, cannot decide on their own
→ participants can only ask coordinator repeatedly for global decision

Failure Scenario: Crash of Coordinator



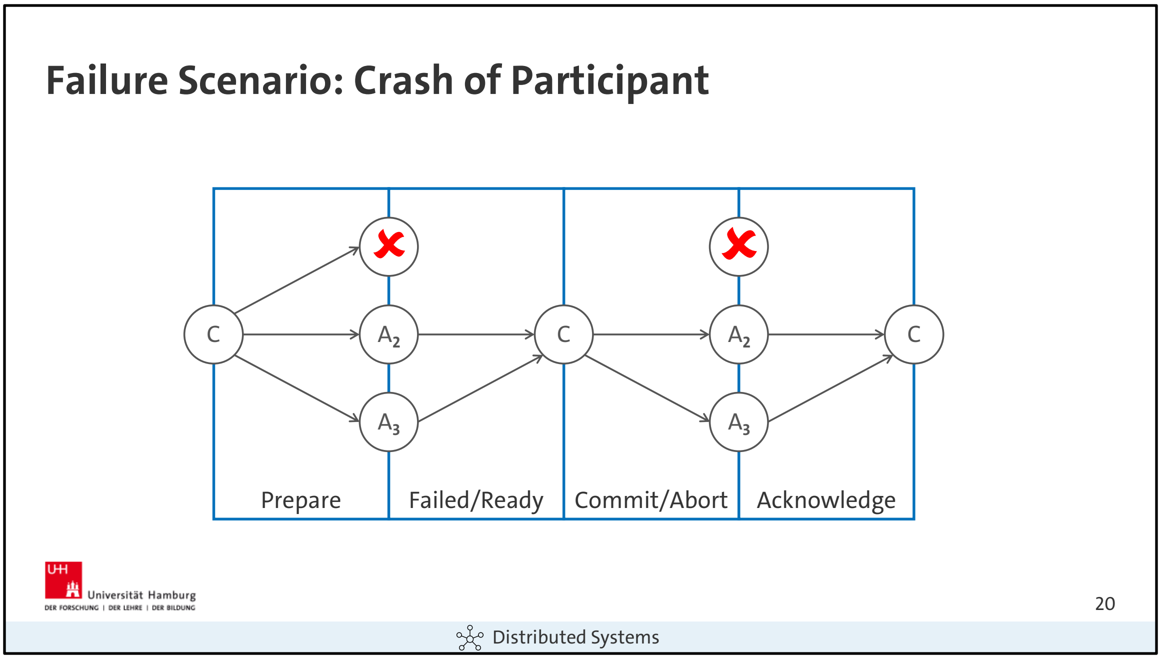
Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Participants need to block until the coordinator recovers or a back-up coordinator takes over

Crash after a participant sent FAILED
- No problem

→ Participants know that coordinator must decide for global ABORT

Crash after participants sent READY
- Participants wait for global decision of crashed coordinator
- Participants are blocked → **Main problem of 2PC-Protocol!**
- Availability of participants for other global and local transaction gets restricted
- Solution: 3PC → avoids blocking (but more messages)



Participant involved in new transactions
- All new transactions that involve a crashed participant can not be executed (for now, we assume data is not replicated)

Participant does not answer to PREPARE message in time
- Coordinator has sent PREPARE and waits for answers of participants
- After timeout participant is considered as crashed → **Long blocking → Main problem of 2PC-Protocol!**
- Coordinator decides for ABORT -> sends an abort to all participants (as well as to the failed node once it is recovered)

Crashed participant restarts
- Check log file for transaction T
- No READY entry → participant performs a local rollback and sends FAILED to coordinator
- READY entry but no COMMIT entry
  → Participant asks coordinator for global decision
  → Depending on coordinator's answer participant performs UNDO (Abort) or REDO (Commit)
- COMMIT entry → participant performs a local REDO of T