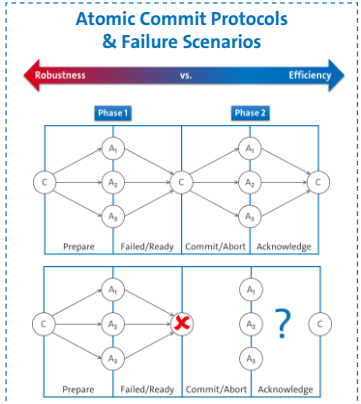
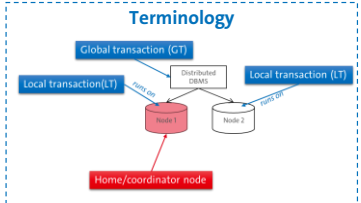
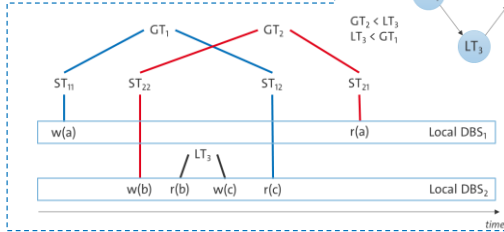
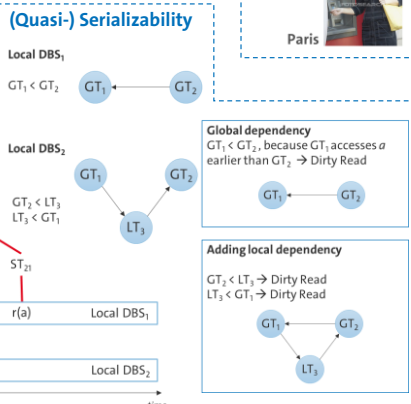
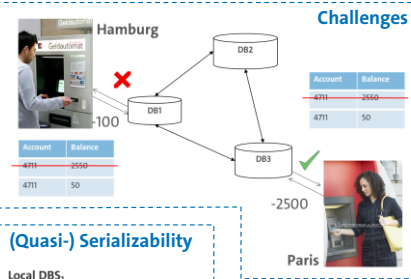
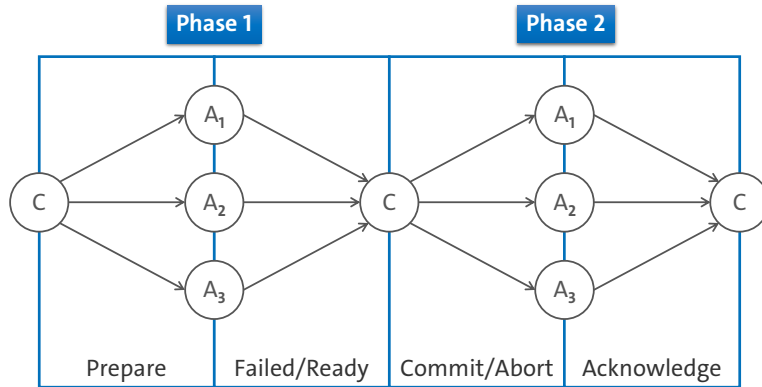


Summary

- Architecture of Database Systems
- Transaction Management
- Modern Database Technology
- Data Warehouses and OLAP
- Data Mining
- Big Data Analytics



Recap Two-Phase-Commit Protocol

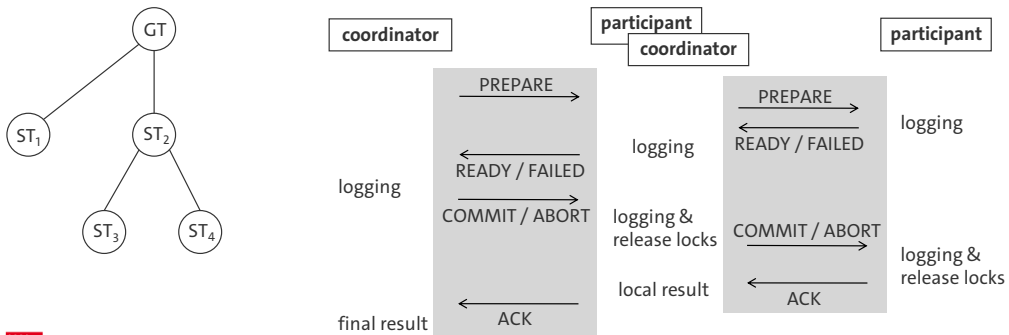


Subtransactions can be arbitrarily nested

→ Tree 2PC

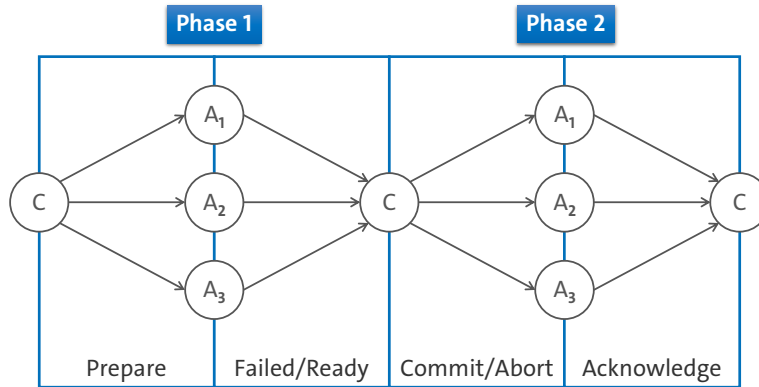
Tree 2PC

General model with arbitrary nesting → participants play the role of a coordinator for hierarchy of sub-transactions



- Answering time increases with nesting depth (lower parallelization)
- Flatten tree to apply “normal” 2PC

Recap Two-Phase-Commit Protocol

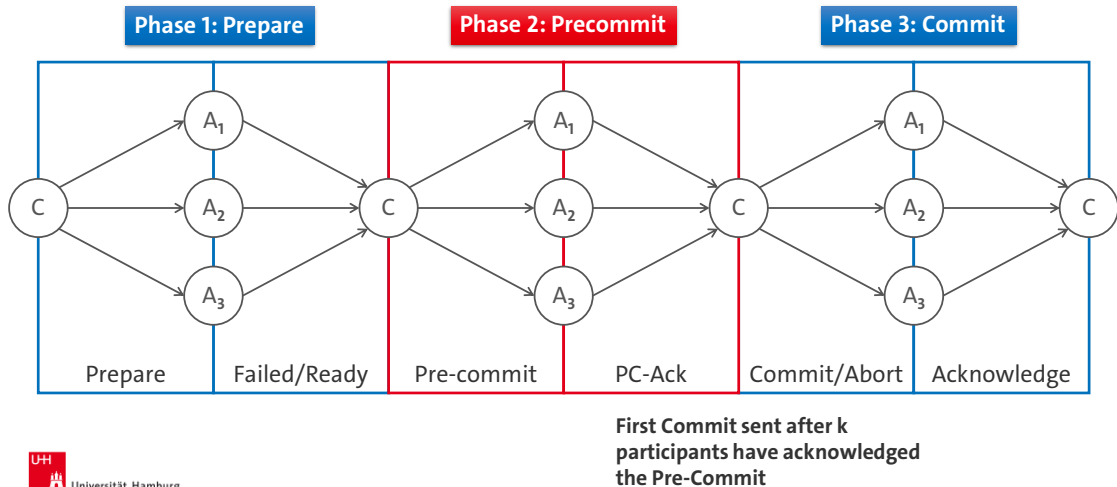


Failures are tolerated BUT

- During restart of coordinator/participant, state of unfinished transactions must be reconstructed from logs
- Long blocking → All participants are blocked until recovery is done

→ **Solution: 3PC**

Three-Phase-Commit Protocol



In case of a coordinator crash

- Coordinator crash is detected by timeout
- Selection of new coordinator (usually one of the participants)
- New coordinator asks remaining participants about the decisions logged for transactions affected by crash
- If one participant has COMMIT/ABORT in its log, that is the global decision
- If one participant has PRECOMMIT in its log, coordinator continues with sending out PRECOMMITs
- In any other case, the coordinator aborts the transaction

Advantage

- Non-blocking protocol, allowing $k < n$ coincidental crashes of participants
- Participant, who received a pre-commit, knows that global decision will be Commit
- If the pre-commit is not received (because the coordinator failed or the coordinator is still waiting for an answer of a failed participant), the participant rolls back and releases its blocked resources → no requirement for repeatedly asking the coordinator for a decision

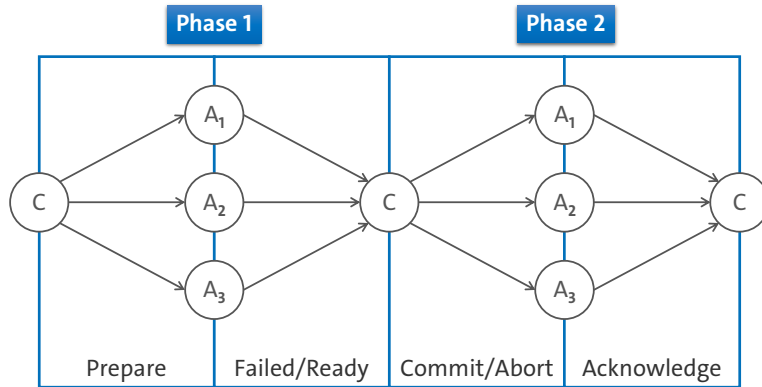
Problem 1: Complexity

- High number of messages
- n participants (including coordinator) require $6(n-1)$ messages in normal run
- $3n \log$ messages
- Solution: 1PC (among others)

Problem 2: Fails in case of network partitioning

- One partition selects new coordinator while the other still has the old coordinator
- New coordinator could come to other global decision than old coordinator
- Solution: Enhanced Three Phase Commit (E3PC) → Quorum based approach for recovery

Recap Two-Phase-Commit Protocol

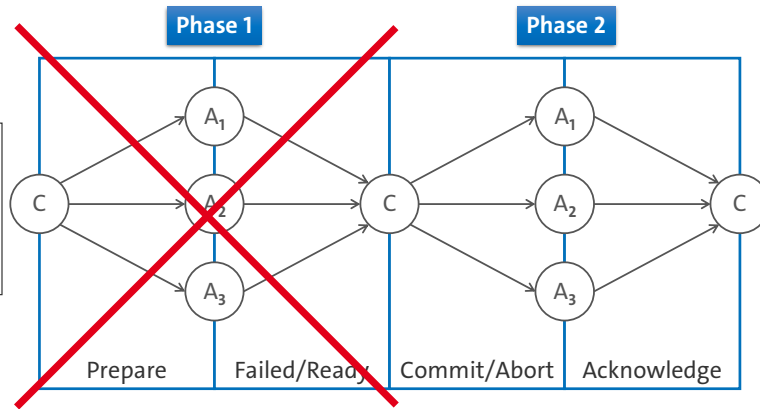


For short running transactions
commit protocol is more costly
than transaction itself

→ Solution: 1PC, Linear 2PC

One-Phase-Commit Protocol

- Actual work starts here, before phase 1 (after calling a function $f()$)
- READY message is sent when $f()$ has returned, and the participant is aware of the prepare request



The Ready message is not sent after the prepare request, but directly after returning from function $f()$

→ After last operation, participant switches to prepared state

Advantages

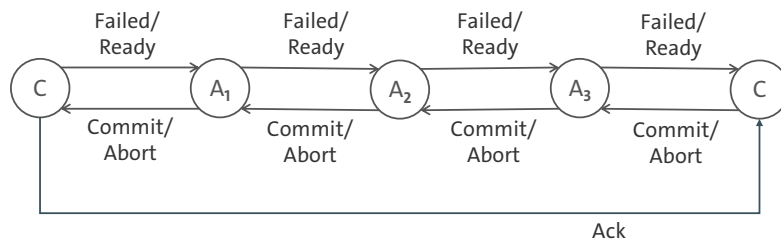
- Only $2(n-1)$ messages

Disadvantages

- Increased probability of blocking because of early prepared state
- High dependency from coordinator (no early unilateral abort) → Rarely used for distributed systems

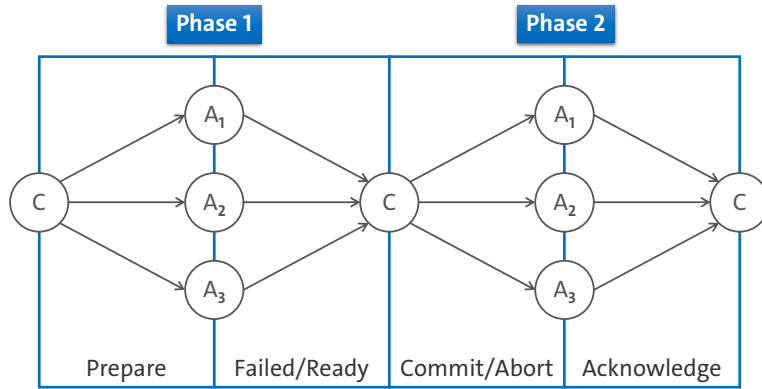
Linear 2PC

- Daisy chaining the communication between the participants
- Phase 1: Forward communication (READY / FAILED)
- Phase 2: Backward communication (COMMIT / ABORT)



- Coordinator acts only as initiator
 - Goes to PREPARED state and sends local decision (READY/FAILED) to participant next in chain
- Each participant in the chain
 - Sends READY to next if it received READY and it locally decides for READY, too
 - Sends FAILED to next if it received FAILED or it locally decides for FAILED
- Last participant takes coordinator role and log global decision
- Global decision is communicated in reverse order back to the initiator
- Initiator sends ACKNOWLEDGE to last participant
- **Advantage:** Only $2n-1$ messages
- **Disadvantage:** Significantly longer response time for large number of participants

Recap Two-Phase-Commit Protocol

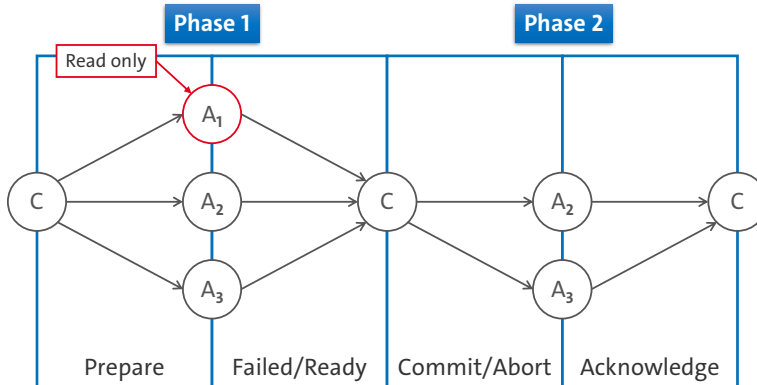


Read-only transactions do not need to propagate any changes
 → 2PC/3PC is overkill

→ Solution: 2PC/3PC with read-only STs

Read-Only (Sub-)Transactions with 2PC

No 2nd phase if transaction is only reading



- No logging, no recovery no changes that would have to be undone/redone)
- Commit only releases locks (at the latest, locks might be released earlier, i.e. in phase 1, because they must be released irrespective of the success of the global transaction)

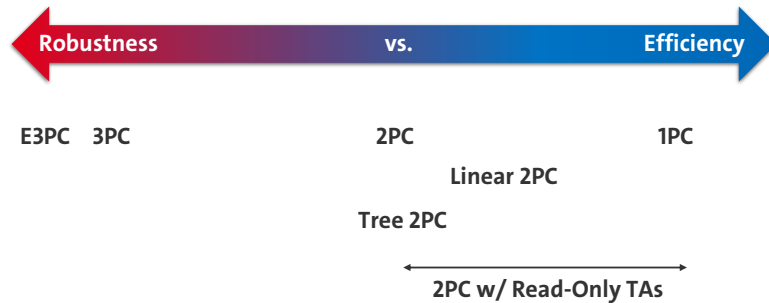
→ Protocol can be optimized

- Assuming m readers among $n-1$ sub-transactions ($m < n$)
- Reduced message complexity: $4(n - 1) - 2m$ messages
- Reduced log complexity: $2n - m$ log writes
- Special case: read-only global transaction ($m = n$)
→ only $2(n-1)$ message, no log writes (like 1PC)

Atomic Commit Protocols

- Non-blocking
- Each node should be allowed to abort as long possible

Generate as few messages and log entries as possible

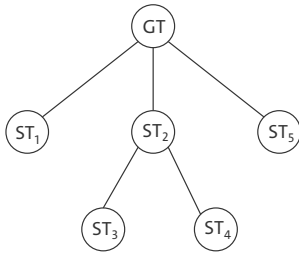


!Here, efficiency is not necessarily equal to the performance of transactions, but only refers to the number of generated messages and log entries!

→ Performance also depends on multiple other factors, e.g. nesting, number of STs, ratio of reading STs to updating STs.

Exercise

Distributed transaction running
on 6 different nodes

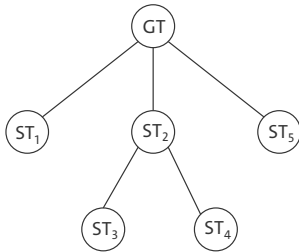


- How many messages are required to perform the 2-Phase-Commit Protocol (with a flattened process tree) for this transaction in case of a successful processing?
- How would that change, if T_5 fails?
- How does it change if 1PC, Tree 2PC, or 3PC is used?
- How many messages are necessary with 2PC when ST_1 and ST_5 are read-only?

Show the communication topology for each variant!

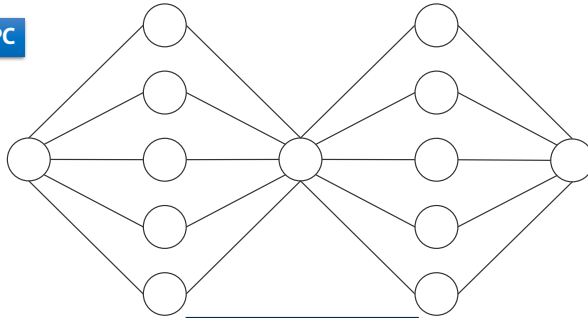
Exercise

Distributed transaction running
on 6 different nodes



- How many messages are required to perform the 2-Phase-Commit Protocol (with a flattened process tree) for this transaction in case of a successful processing?
- How would that change, if T_5 fails?
- How does it change if 1PC, Tree 2PC, or 3PC is used?
- How many messages are necessary with 2PC when ST_1 and ST_5 are read-only?

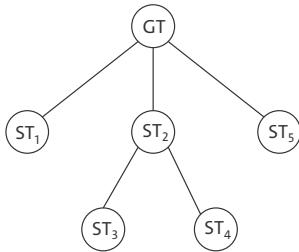
2PC



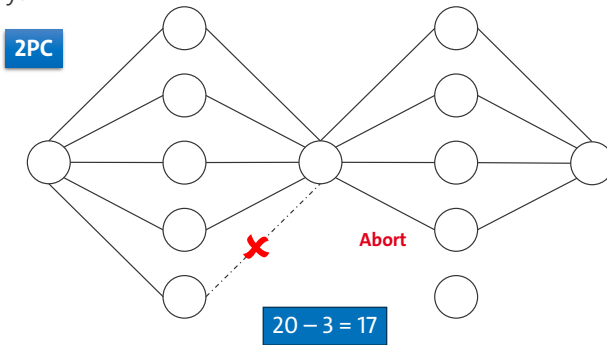
$$4 * (n-1) = 4 * (6-1) = 20$$

Exercise

Distributed transaction running
on 6 different nodes

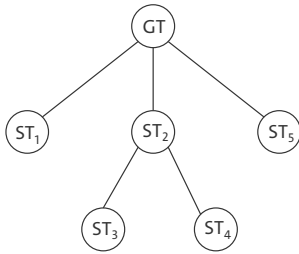


- How many messages are required to perform the 2-Phase-Commit Protocol (with a flattened process tree) for this transaction in case of a successful processing?
- **How would that change, if T_5 fails before sending the Ready message?**
- How does it change if 1PC, Tree 2PC, or 3PC is used?
- How many messages are necessary with 2PC when ST_1 and ST_5 are read-only?



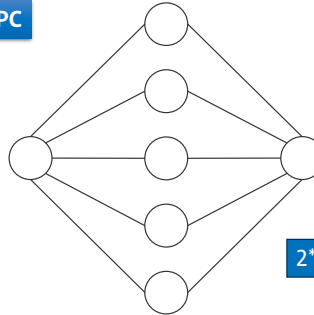
Exercise

Distributed transaction running
on 6 different nodes



- How many messages are required to perform the 2-Phase-Commit Protocol (with a flattened process tree) for this transaction in case of a successful processing?
- How would that change, if T_5 fails before sending the Ready message?
- **How does it change if 1PC, Tree 2PC, or 3PC is used?**
- How many messages are necessary with 2PC when ST_1 and ST_5 are read-only?

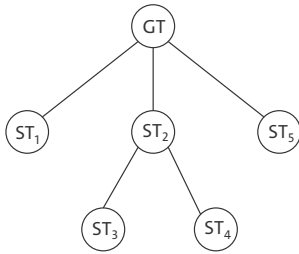
1PC



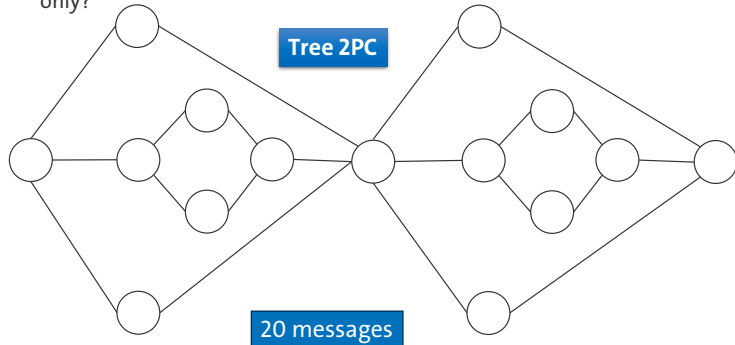
$$2 * (n-1) = 2 * (6-1) = 10$$

Exercise

Distributed transaction running
on 6 different nodes

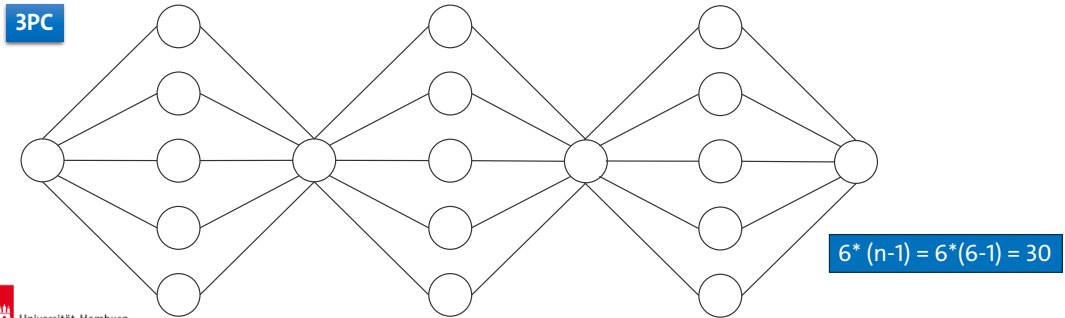


- How many messages are required to perform the 2-Phase-Commit Protocol (with a flattened process tree) for this transaction in case of a successful processing?
- How would that change, if T_5 fails before sending the Ready message?
- **How does it change if 1PC, Tree 2PC, or 3PC is used?**
- How many messages are necessary with 2PC when ST_1 and ST_5 are read-only?



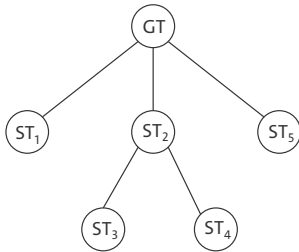
Exercise

- How many messages are required to perform the 2-Phase-Commit Protocol (with a flattened process tree) for this transaction in case of a successful processing?
- How would that change, if T_5 fails before sending the Ready message?
- **How does it change if 1PC, Tree 2PC, or 3PC is used?**
- How many messages are necessary with 2PC when ST_1 and ST_5 are read-only?

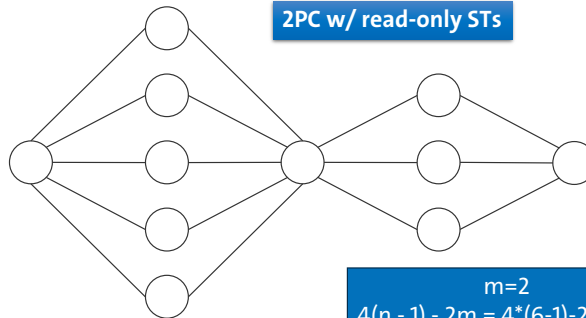


Exercise

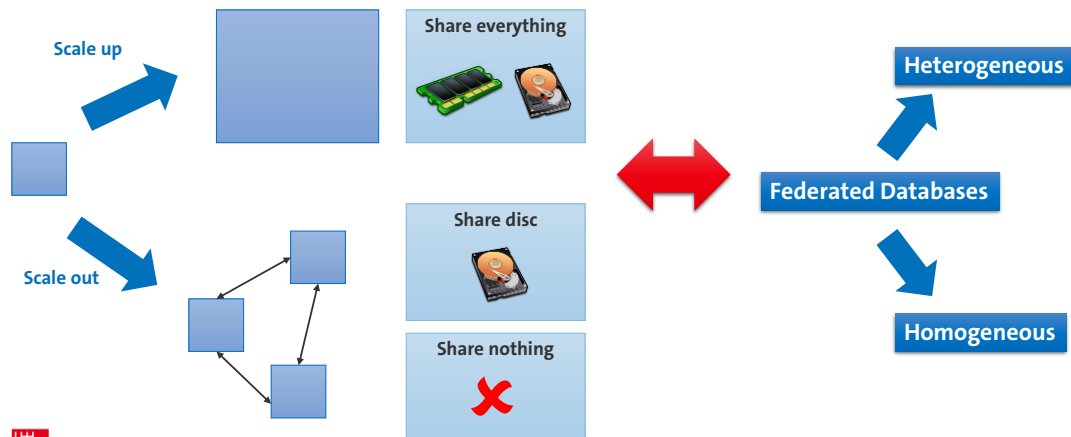
Distributed transaction running
on 6 different nodes



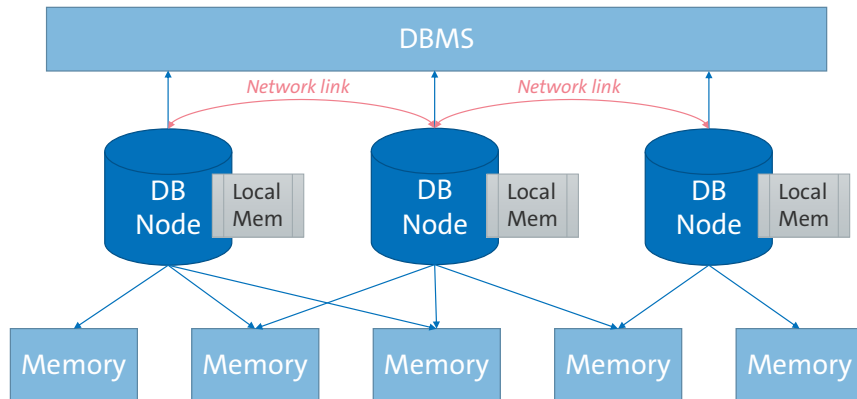
- How many messages are required to perform the 2-Phase-Commit Protocol (with a flattened process tree) for this transaction in case of a successful processing?
- How would that change, if T_5 fails?
- How does it change if 1PC, Tree 2PC, or 3PC is used?
- **How many messages are necessary with 2PC when ST_1 and ST_5 are read-only?**



To share or not to share: Types of distributed DBs



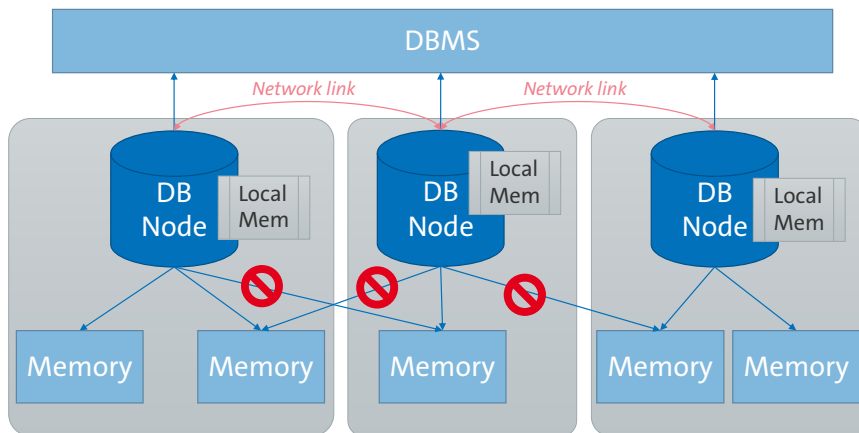
Shared Disc



Shared (Disk) Cluster

- Distributed Database
- One or more servers have equal access to memory (e.g. to discs)
- Servers don't share their own memory

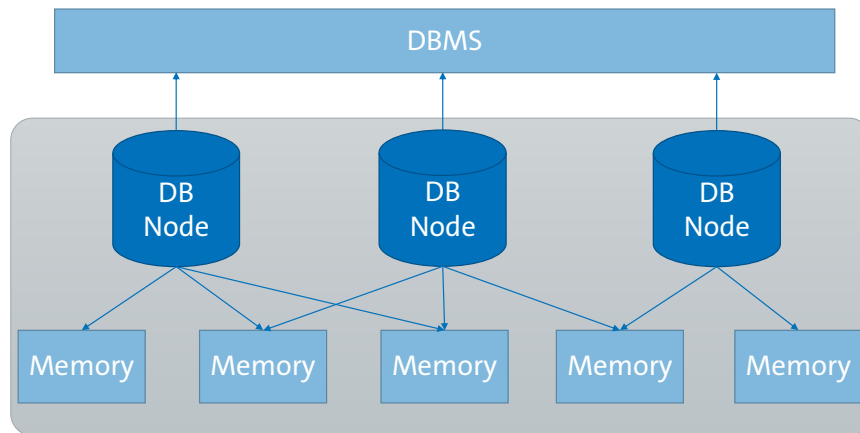
Shared Nothing



Shared Nothing

- Distributed Database
- Each node has exclusive access to its memory
- Servers don't share their own memory

Share Everything




Shared Everything

- One big system instead of multiple (small) systems → scale-up instead of scale-out
- Disk and main memory is shared (NUMA system)

To Share Or Not To Share

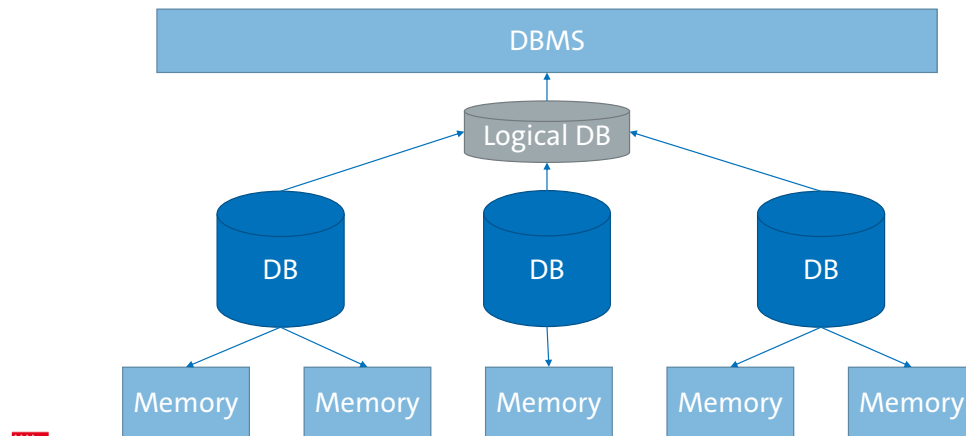
	Scale-out		Scale-up
	Shared Disk	Shared Nothing	Share Everything
Advantages	<ul style="list-style-type: none"> - Robust in case of node failure (discs can be accessed by another node) - Usually easy to set up (if there is already a shared file system) 	<ul style="list-style-type: none"> - Robust in case of disk failure (frequently used data can be replicated across nodes) - No distributed locking necessary - High performance if query is executed on node where most of the data is 	<ul style="list-style-type: none"> - Comes for free with many systems → no additional setup - Faster than accessing remote memory
Disadvantages	<ul style="list-style-type: none"> - Simultaneous disk access is a potential bottleneck - Overhead to maintain cache consistency - Requires complex locking mechanisms for updates 	<ul style="list-style-type: none"> - Partitioning (sharding) of data needs additional care to get optimal performance (store data where it is processed) 	<ul style="list-style-type: none"> - Limited scaling possibilities - Hardware for large systems becomes expensive
Example System(s)	<ul style="list-style-type: none"> - Add-on/Feature of data management systems, e.g. Microsoft Azure shared disk, Oracle RAC - Hybrid Systems (SD & SN), e.g. Snowflake 	Couchbase*, MariaDB SkySQL (not actively developed anymore),...	Each system working with NUMA architectures, e.g. MonetDB, PostgreSQL, SQL Server,...
Comments	Requires shared file system		Usually not noticed by the user

 Distributed Systems

*Couchbase white paper:

http://info.couchbase.com/rs/northscale/images/Couchbase_Architectural_Document_Whitepaper_2015.pdf

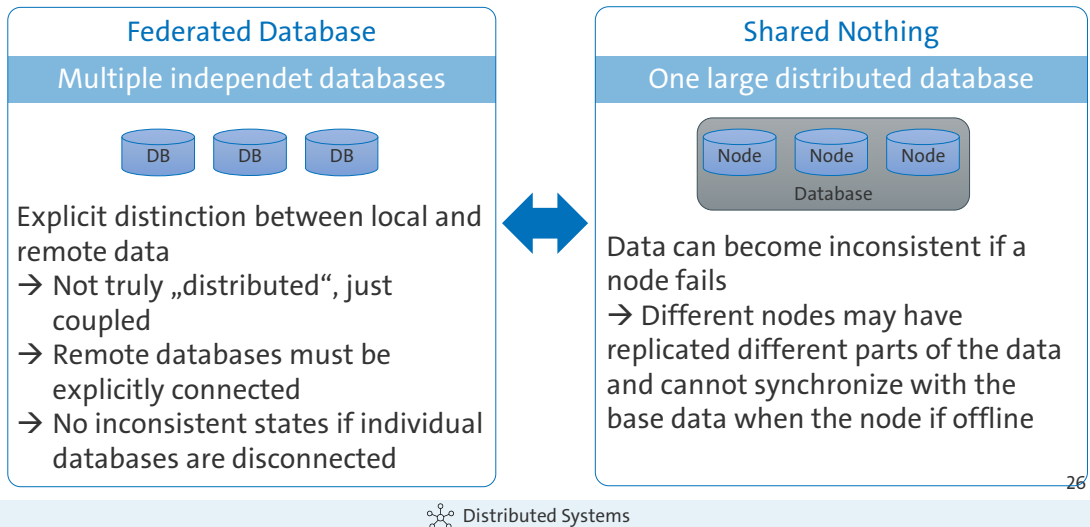
Federated Databases



Federated Database

- Multiple Databases
- Databases are connected to one logical view
- Databases do not directly share data
- Data is not replicated

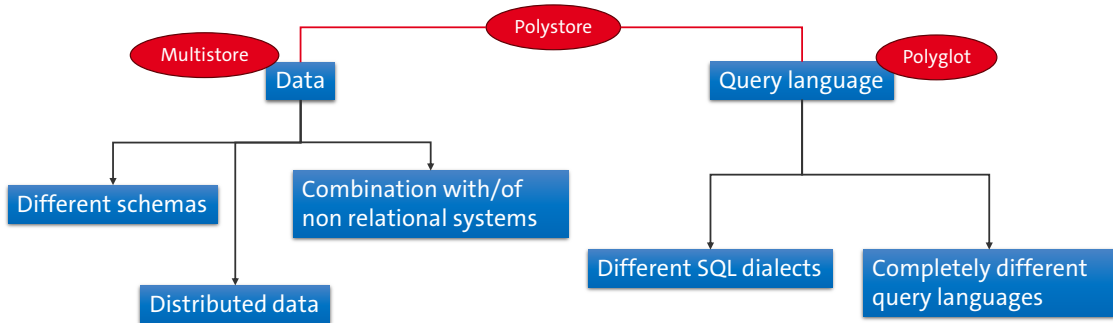
Federated Databases



- In a shared nothing DB, a node is just a node in the DB system
- In a federated DB, each “node”/component is an autonomous DB system → Data is not copied between these DBs
- A DB as a component can itself be a distributed DB

Heterogeneous Federated DBs

Components can vary in different aspects → This is one of various possible taxonomies



Heterogeneous DBS spawn a variety of research topics, e.g. schema integration and novel interfaces

Further reading

An approach to classify Federated DBMS:

Azevedo, Leonardo Guerreiro, et al. "Modern Federated Database Systems: An Overview." *ICEIS (1)* (2020): 276-283.

Different approach for classification based on Data coupling

Loosely coupled → Local stores accessed by their local language or a common language

Tightly coupled → Local stores accessed by a multistore system using the same language for structured and unstructured data (e.g. Hadoop)

Hybrid → Some Stores are loosely coupled and some are tightly coupled (e.g. Spark SQL, BigDawg)

Important information for those who learn with the material of the previous year

→ In the slides Prof. Ritter, yet another taxonomy is used: Homogeneous

Federations can run only global transactions whereas heterogeneous federations can also run local transactions

→ This describes heterogeneity in terms of autonomy