THE UNIVERSITY OF MELBOURNE

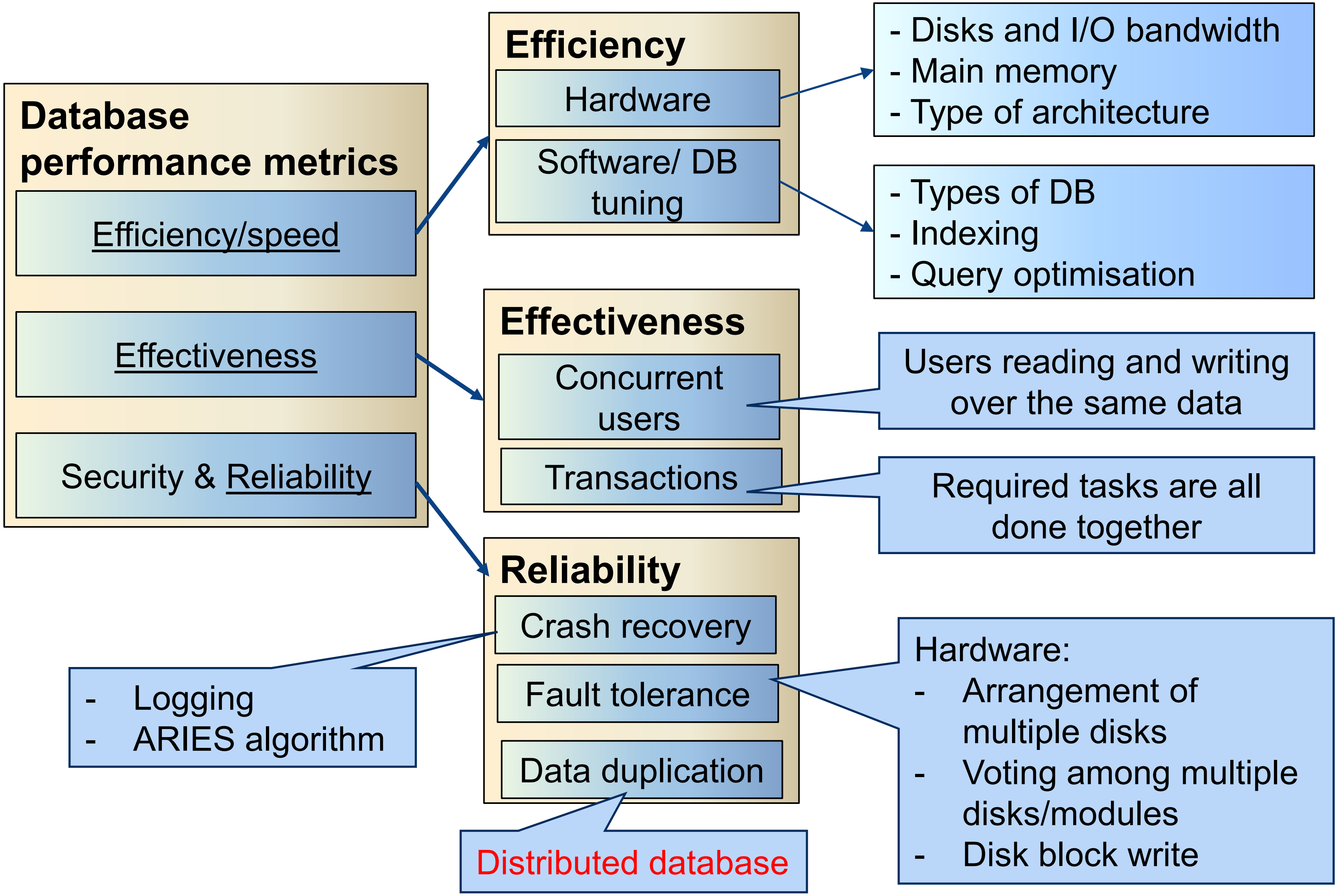
COMP90050 Advanced Database Systems

Winter Semester, 2023

Lecturer: Farhana Choudhury (PhD)

Week 4 part 4

Core Concepts of Database management system



Database performance metrics

- Efficiency/speed
- Effectiveness
- Security & Reliability

Efficiency

- Hardware
  - Disks and I/O bandwidth
  - Main memory
  - Type of architecture
- Software/ DB tuning
  - Types of DB
  - Indexing
  - Query optimisation

Effectiveness

- Concurrent users: Users reading and writing over the same data
- Transactions: Required tasks are all done together

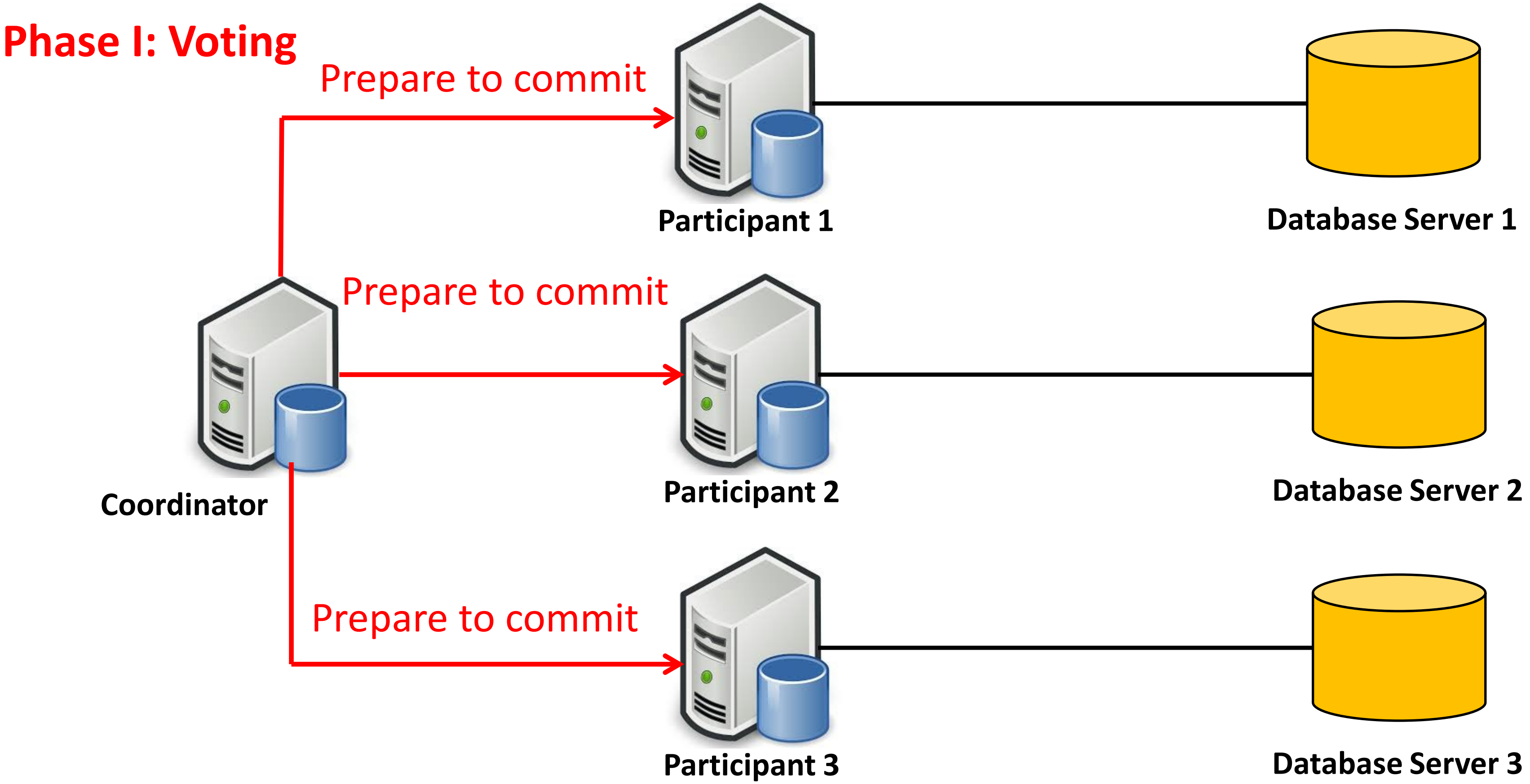
Reliability

- Crash recovery
  - Logging
  - ARIES algorithm
- Fault tolerance
  - Hardware:
    - Arrangement of multiple disks
    - Voting among multiple disks/modules
    - Disk block write
- Data duplication: Distributed database

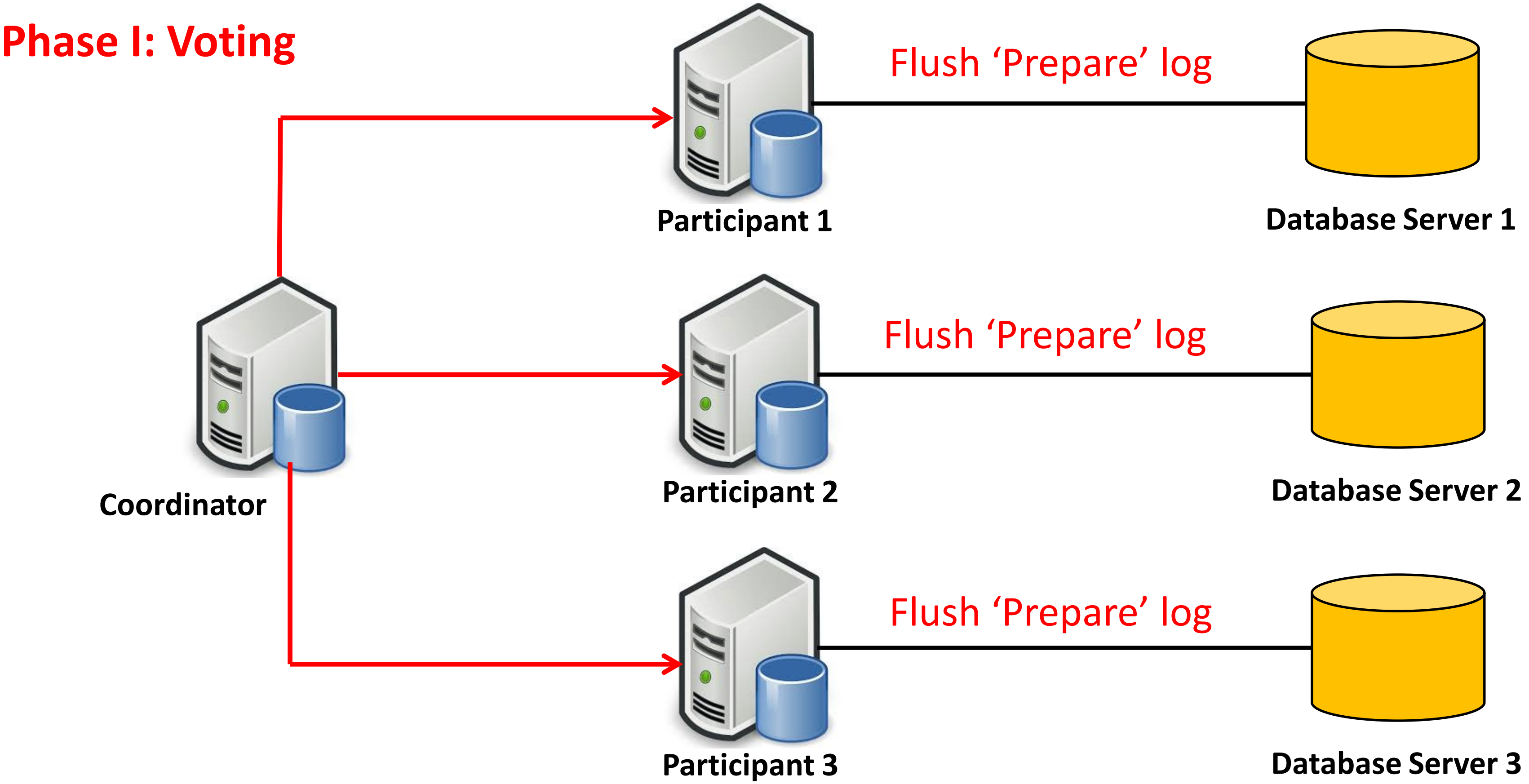
A related concept on distributed databases first - atomicity

Atomicity in distributed transaction processing

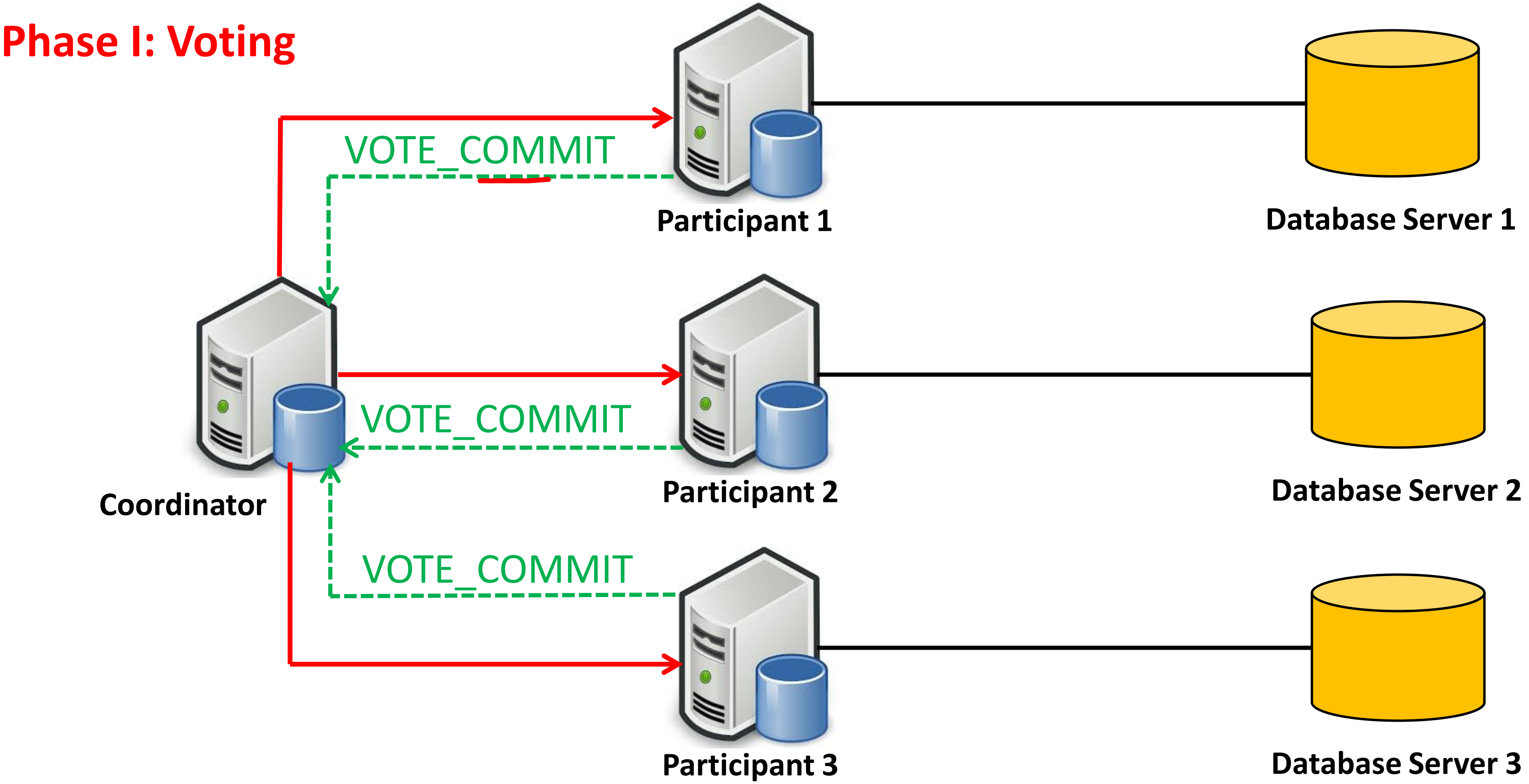
The two-phase commit protocol (2PC) can help achieve atomicity in distributed transaction processing



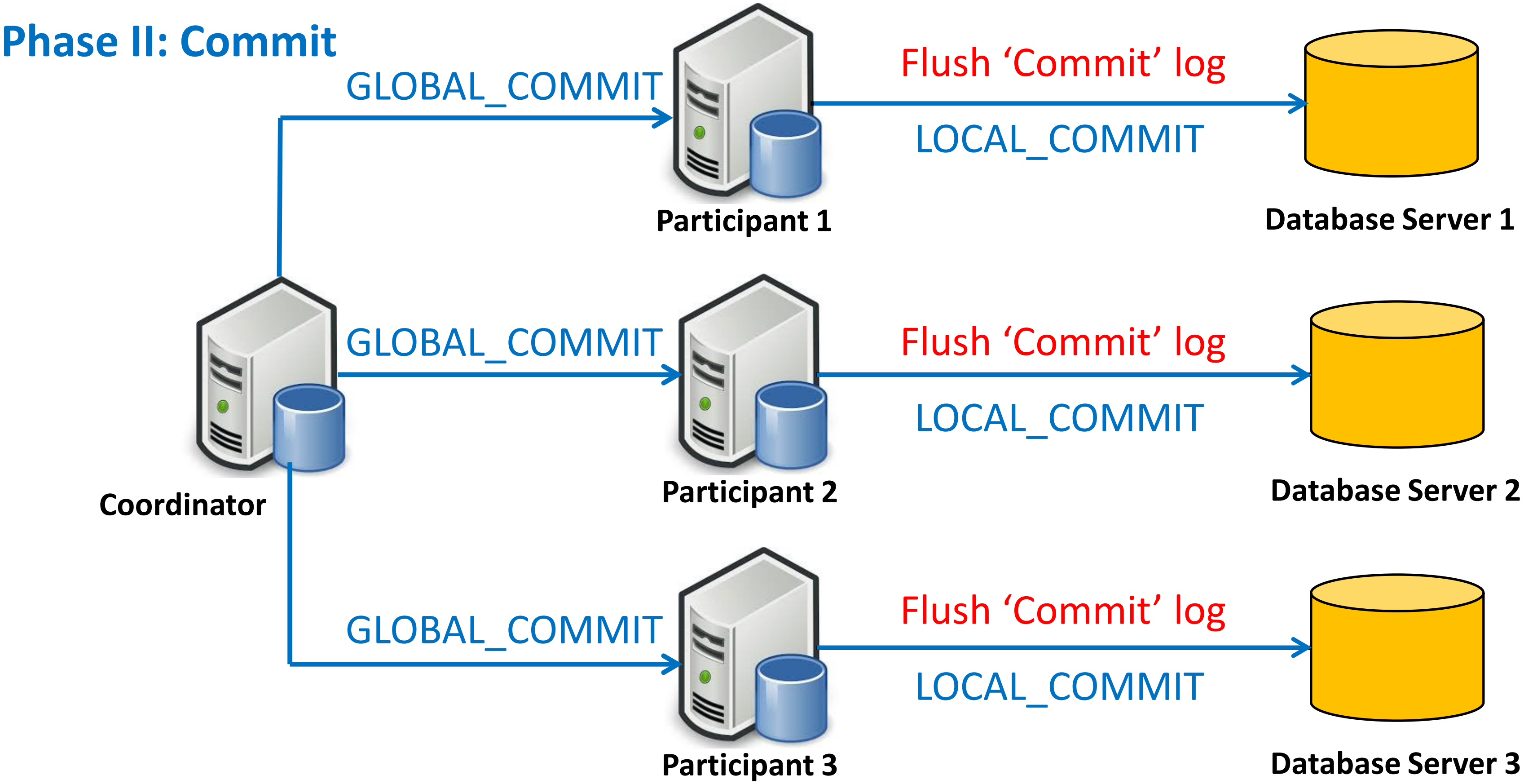
Two phase commit protocol



Two phase commit protocol



Two phase commit protocol



Two phase commit protocol

- Coordinator or participant can abort transaction
  - If a participant abort, it must inform coordinator
  - If a participant does not respond within a timeout period, coordinator will abort
- If abort, coordinator asks all participates to rollback
- If abort, abort logs are forced to disk at coordinator and all participants

Another related concept on distributed databases - concurrency control

Concurrency control in distributed DBs

- **Each server is responsible for applying concurrency control to its own objects**

- The members of a collection of servers of distributed transactions are jointly responsible for ensuring that they are performed in a serially equivalent manner

- ***BUT*** **servers independently acting would not work**

- If transaction ***T*** **is before transaction** ***U*** in their conflicting access to objects at one of the servers then:
  - They **must be in that order at all of the servers** whose objects are accessed in a conflicting manner by both *T* and *U*

- The central **Coordinator should assure this**

Other Considerations for Locking-based systems

- A local lock manager cannot release any locks until it knows that the transaction has been committed or aborted at all the servers involved in the transaction.

- The objects remain locked and are unavailable for other transactions during the commit protocol.
  - An aborted transaction releases its locks after phase 1 of the protocol.

Timestamp ordering concurrency control revisited

- The coordinator accessed by a transaction issues a **globally unique timestamp**

- The **timestamp is passed with each object access**

- The servers are jointly responsible for ensuring serial equivalence:
  - that is **if T access an object before U, then T is before U at all objects**

Optimistic concurrency control revisited

For distributed transactions to work:

1) **Validation takes place in phase 1 of 2PC protocol at each server**

2) **Transactions use a globally unique order for validation**

What if objects in different servers are replicas for increased availability

Client 1:	Client 2:
setBalance$_B$(x,1)	
setBalance$_A$(y,2)	
	getBalance$_A$(*y*) → 2
	getBalance$_A$**(*x*) → 0**

Server

Initial balance of x and y is $0

The **behaviour above cannot occur if A and B did not exist** (that is, if we had only one server)

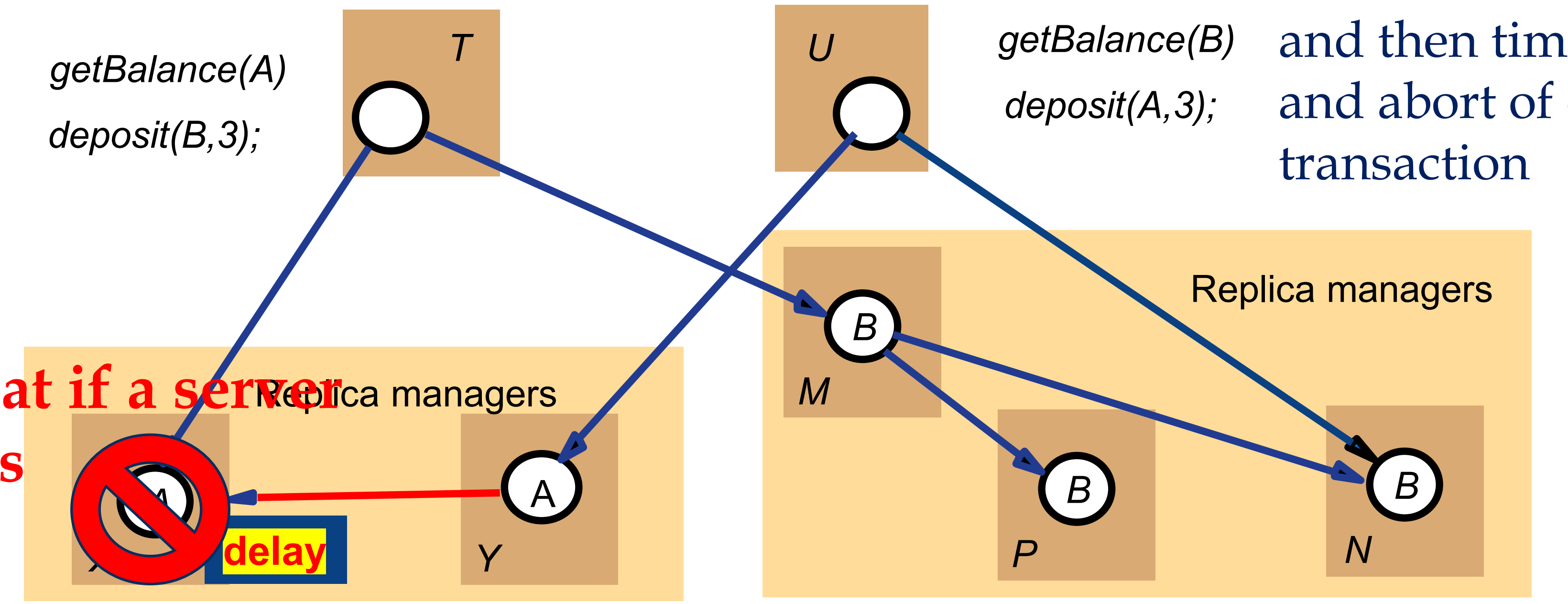
Transactions with replicated data

- the effect of transactions on replicated objects should be the **same as if they had been performed one at a time on a single set of objects**
- this property is called ***one-copy serializability***
- **If all servers are available then no issue – but what if some servers are not available?**

Lets build the solution step by step

The **available copies replication** scheme is designed to allow some servers to be temporarily unavailable

May lead to deadlock and then timeout and abort of one transaction



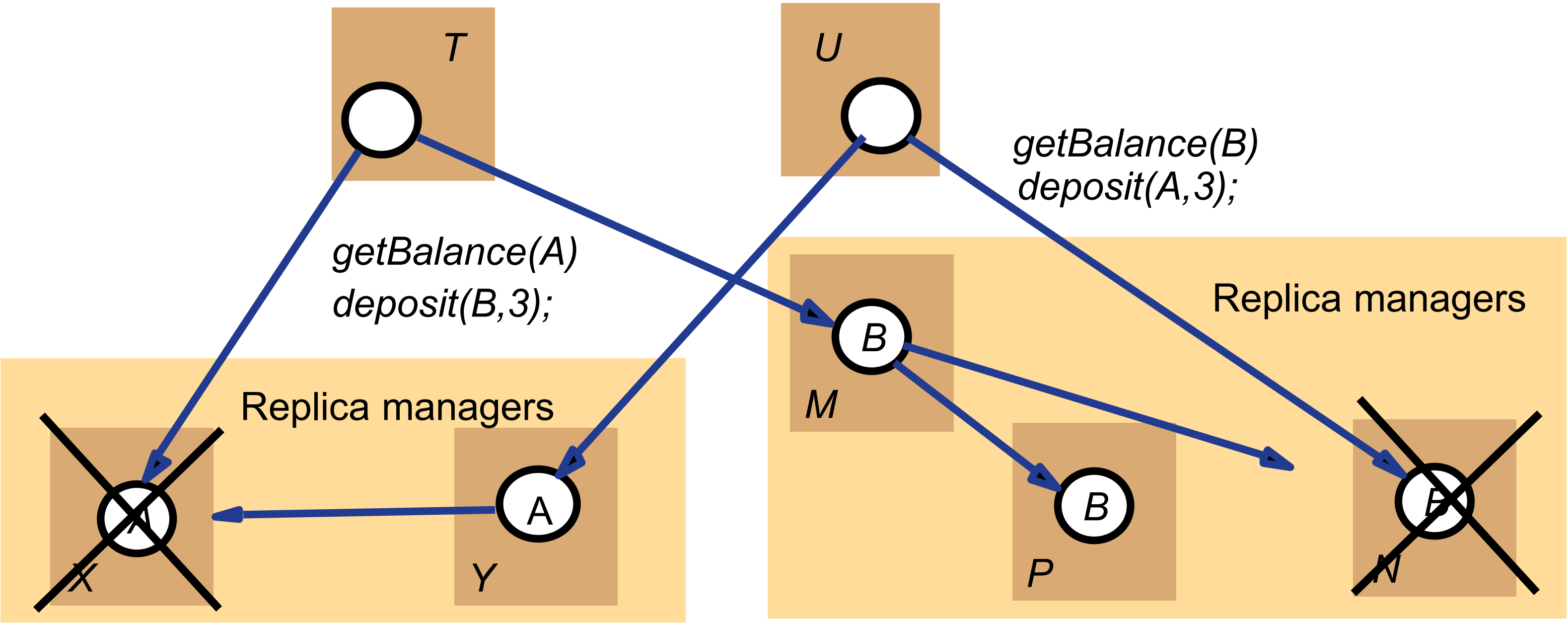
What if a server fails

At *X, T* has read *A* and has locked it. Therefore *U*'s *deposit* is delayed until *T* finishes. Normally, this leads to good concurrency control only if the servers do not fail….

Cannot the other servers simply be used?

assume that *X* fails just after *T* has performed *getBalance*

and *N* failing just after *U* has performed *getBalance*



therefore *T*'s deposit will be performed at *M* and *P* (all available)

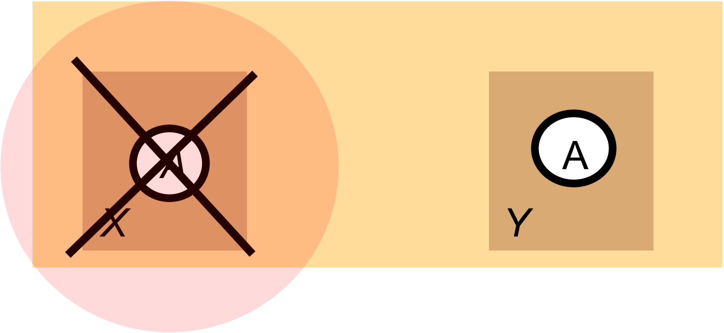
and *U*'s deposit will be performed at *Y* (all available)

NOT GOOD!!

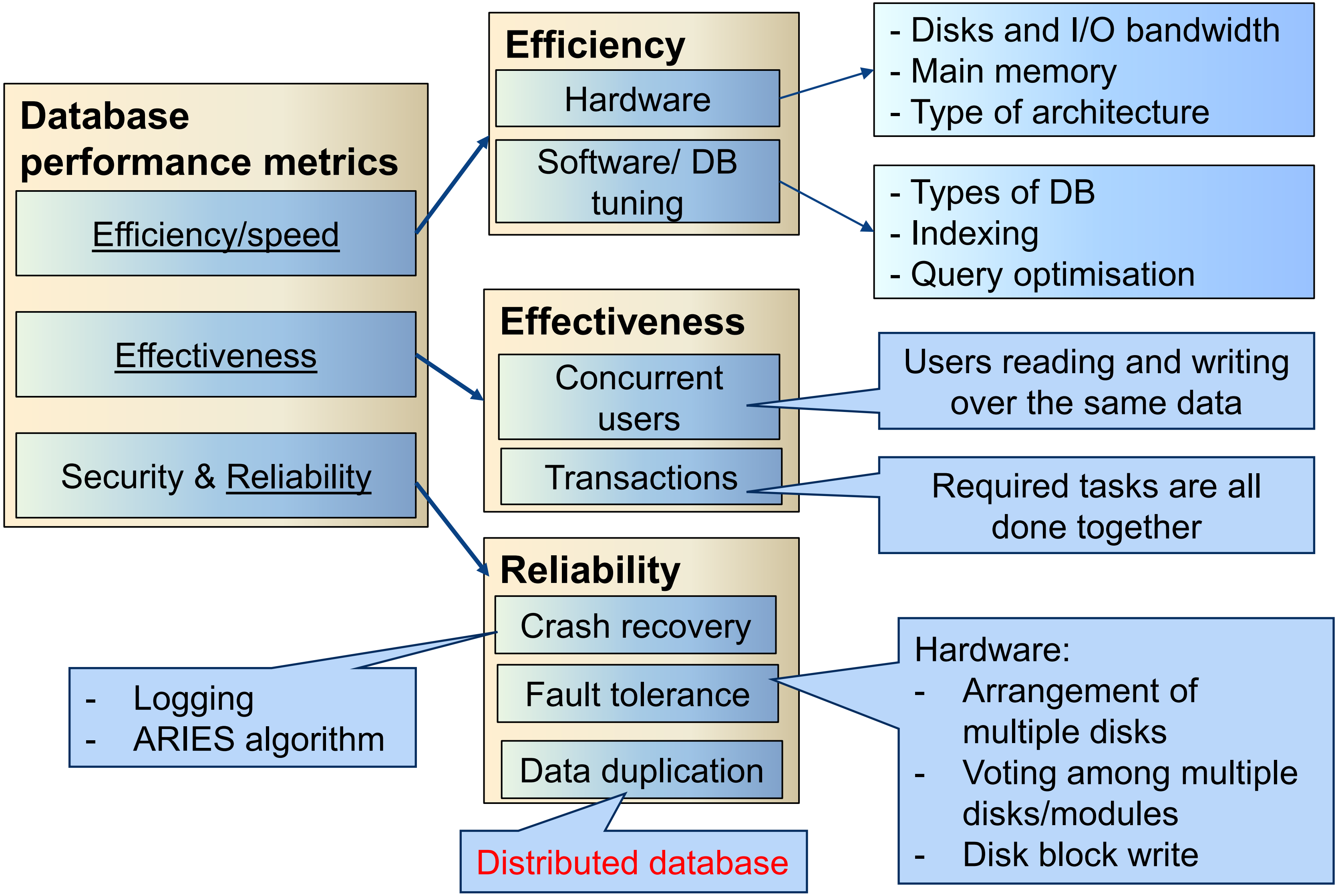
Available copies replication rule

Before a transaction commits, it checks for failed and available servers it has contacted, the set should not change during execution:

- E.g., *T* would check if X is still available among others.
- We said X fails before *T*'s *deposit*, in which case, T would have to abort.
- Thus no harm can come from this execution now.



Core Concepts of Database management system



Database performance metrics

- Efficiency/speed
- Effectiveness
- Security & Reliability

Efficiency

- Hardware
  - Disks and I/O bandwidth
  - Main memory
  - Type of architecture
- Software/ DB tuning
  - Types of DB
  - Indexing
  - Query optimisation

Effectiveness

- Concurrent users: Users reading and writing over the same data
- Transactions: Required tasks are all done together

Reliability

- Crash recovery
  - Logging
  - ARIES algorithm
- Fault tolerance: Hardware:
  - Arrangement of multiple disks
  - Voting among multiple disks/modules
  - Disk block write
- Data duplication: Distributed database