+

Lecture Notes
Week 04

INFS3200 Advanced Database Systems
Semester 1, 2021

The University of Queensland
Australia

Distributed Transaction Management

Professor Xue Li

+ Last Week

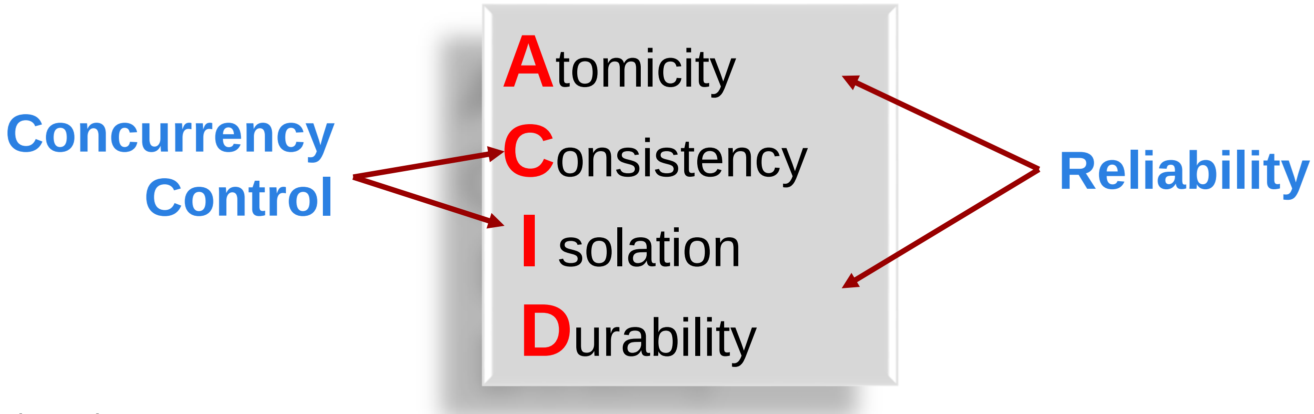
- Distributed Query Processing
  - Query decomposition
  - Data localization: replacement and then optimization
    - HF reduction with selection
    - HF reduction with join
    - VF reduction
  - Global optimization: cost models,
  - Join strategies
  - Semi-join of distributed data fragments

+ Learning Objectives

- An overview of TM (Transaction Management) in centralized DBMS
- Updates in distributed database systems
- Distributed transaction management
  - Concurrency, reliability, replication, network partitioning

+ Transactions: Basic Concepts

- Transaction
  - A logical unit of database processing
  - A sequence of read/write operations treated as one atomic unit
- Transaction Processing Systems
  - Large databases with multiple users executing many concurrent database transactions
  - Guaranteed **ACID** properties even in the presence of conflicting operations



+ Properties of a Transaction

- **Atomicity**
  - A transaction is a sequence of database operations that are either performed entirely or are not performed at all.
- **Consistency**
  - A transaction must transform the database from one consistent state to another consistent state.
- **Isolation**
  - Transactions execute independently of one another. The partial effects of incomplete transactions should not be visible to other transactions.
- **Durability**
  - The effects of a committed transaction are permanently recorded and must not be lost because of a subsequent failure.

+ Transaction: an Example in SQL

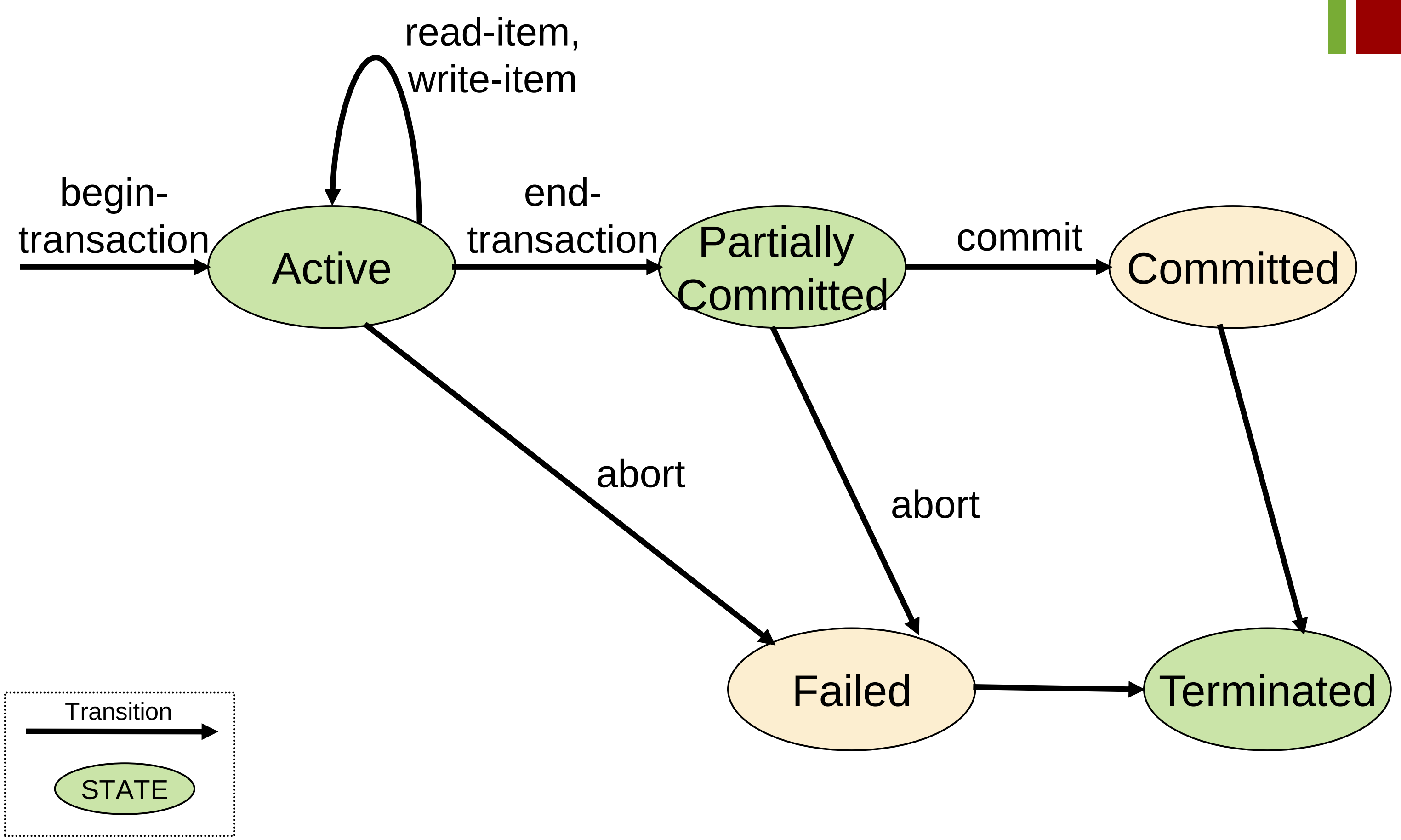
Transaction BUDGET_UPDATE

begin

```
EXEC SQL    UPDATE    PROJ
            SET       BUDGET = BUDGET∗1.1
            WHERE     PNAME = "CAD/CAM"
```

end.

+ Transaction States



...where things may go wrong?

+ Why Concurrency Control ?

Transaction processing systems are large databases with multiple users executing database transactions

↓

Multiple users are concurrently executing transactions

↓

A transaction can have several data access operations, some of which could be accessing the same data item

↓

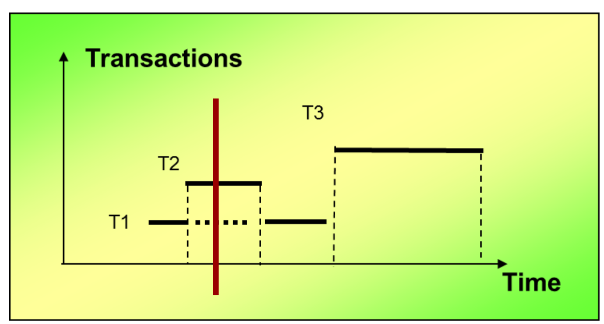
Through multi-programming operating systems, the processes of users are interleaved

+ Schedule

- A **schedule** is the order of execution of operations from various transactions
- A formal definition of a schedule
  - A schedule S of n transactions $T_1, T_2, \ldots, T_n$ is an ordering of the operations of these transactions, given that $\forall\ T_i \in$ S, the order of operations of T_i in S is the same as that in the original T_i
- Schedules consider read-item, write-item, commit and abort operations only and the order of operations in S to be a **total order**.

Interleaved Concurrent Transactions:

At any time point, there may be many active transactions but only one is executing.



T1	T2	T3
R(A)		
	W(A)	
	COMMIT	
W(A)		
COMMIT		
		W(A)
		COMMIT

Serial Transactions

Question:

- Given three transactions T1, T2, T3, how many possible serials for them to be executed correctly (i.e., find their correct serial executions)?

Answer: There are six correct serials:

T1→T2→T3

T1→T3→T2

T2→T1→T3

T2→T3→T1

T3→T1→T2

T3→T2→T1

Operations in serial transactions are not interleaved.

+ Serial Schedules

- Schedules that execute each transaction one by one without any interleaving.
  - Formally, a schedule S is **serial** if for every transaction T participating in S, all operations of T are executed **consecutively**.
  - There are $n!$ serial schedules for n transactions
- A serial schedule is always correct, but unacceptable in practice!
  - Why correct?
  - Why unacceptable?

Operations in serial transactions are not interleaved.

+ Conflicting Operations

- Operations of a schedule are in conflict if they satisfy all the following three conditions:
  - they belong to different transactions
  - they access the same item X
  - at least one is a write-item (X)

Given two transactions with the following concurrent operations.

Case 1:	r1(X); w2(X)
Case 2:	r1(X); w1(X)
Case 3:	r1(X); r2(X)
Case 4:	w1(X); w2(Y)
Case 5:	w1(X); w2(X)
Case 6:	w1(Y); r2(Y)

Conflicting operations

+ Examples of Schedules

- Serial Schedule

T1	T2
read-item (X);	
X:=X-N;	
write-item (X);	
read-item (Y);	
Y:=Y+N;	
write-item (Y);	
	read-item (X);
	X:=X+M;
	write-item (X);

- Non-Serial Schedule

T1	T2
read-item (X);	
X:=X-N;	
write-item (X);	
	read-item (X);
	X:=X+M;
	write-item (X);
read-item (Y);	
Y:=Y+N;	
write-item (Y);	

"Non-Serial Schedule": Schedule with interleaved operations.

+ Serializable Schedules

- A schedule which gives the correct result in spite of interleaving
  - Formally, a schedule *S* of *n* transactions is serializable if it is **equivalent** to <u>any</u> serial schedule of the same *n* transactions

- But, when are two schedules ‘**equivalent**’?

“Conflict serializable schedule”: Schedule with conflict operations may still be equivalent to a serial schedule.

+ Example of Interleaved Transactions

- Two transactions have their operations interleaved in such a way that it makes the value of some data item incorrect

T1	T2
read-item (X);	
X:=X - N;	
	read-item (X);
	X:=X+M;
write-item (X);	
read-item (Y);	
	write-item (X);
Y:=Y+N;	
write-item (Y);	

Consider an example:
Initially: accounts X has $500 and Y has $500.
T1: transfer $100 from X to Y (N=100)
T2: deposit $50 to X (M=50)

+ Non-serializable (Lost Update)

- Two transactions have their operations interleaved in such a way that it makes the value of some data item incorrect

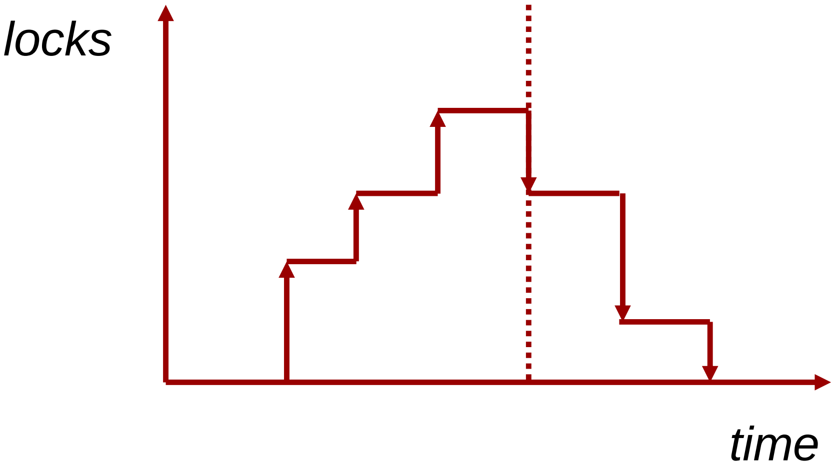
T1	T2
read-item (X);	
X:=X-N;	
	read-item (X);
	X:=X+M;
write-item (X);	
read-item (Y);	
	write-item (X);
Y:=Y+N;	
write-item (Y);	

Item X has an incorrect value because its update by T1 is *lost*

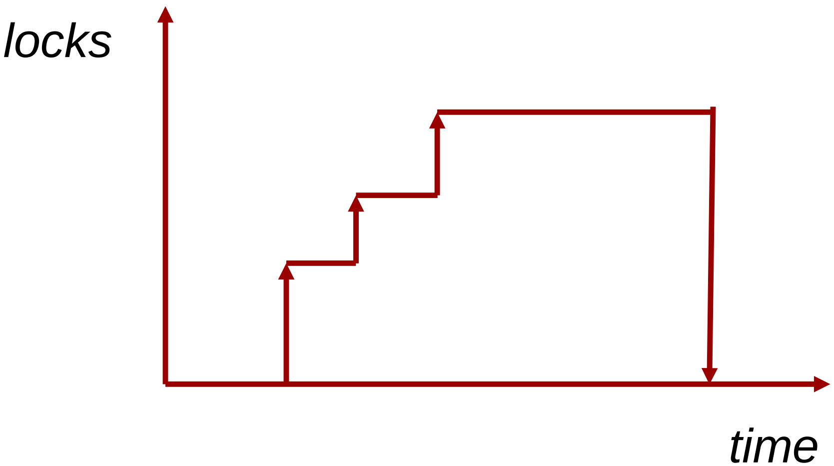
Consider an example:
Initially: accounts X has $500 and Y has $500.
T1: transfer $100 from X to Y (N=100)
T2: deposit $50 to X (M=50)

+ Locking

- Serializability can be achieved by locking
  - Locking is a pessimistic approach
- 2PL



2PL



Strict 2PL

...timestamps can be used for optimistic approaches

Variations of 2PL
(All can guarantee serializability)

	Phase 1 (Expanding)	Phase 2 (Shrinking)	Deadlock?	Advantages
Basic 2PL	Claim Locks one-by-one	Release Locks one-by-one before commit or abort	Yes	Simple
Conservative (Static) 2PL	Claim All locks or nothing	Release Locks one-by-one before Commit or Abort	No	No deadlock (but hard to implement in practice)
Strict 2PL	Claim Locks one-by-one	Release X Locks until after Commit or Abort	Yes	Good recoverability, can expand until it ends.
Rigorous 2PL	Claim Locks one-by-one	Release all Locks until after Commit or Abort	Yes	Easier to implement than strict 2PL

Page 788, Chap 21, 7th Ed Elmasri & Navathe Book

+ Example

Non 2PL

T_1	T_2

read_lock(Y);
read_item(Y);
unlock(Y);
write_lock(X);
read_item(X);
$X := X + Y$;
write_item(X);
unlock(X);
	read_lock(X);
	read_item(X);
	unlock(X);
	write_lock(Y);
	read_item(Y);
	$Y := X + Y$;
	write_item(Y);
	unlock(Y);

Non Serializable

T_1	T_2

read_lock(Y);
read_item(Y);
unlock(Y);	

	read_lock(X);
	read_item(X);
	unlock(X);
	write_lock(Y);
	read_item(Y);
	$Y := X + Y$;
	write_item(Y);
	unlock(Y);

write_lock(X);
read_item(X);
$X := X + Y$;
write_item(X);
unlock(X);	

X and Y uncloked too early!

Non-Serializable means the results of transactions are wrong!

2PL

T_1'

read_lock(Y);
read_item(Y);
write_lock(X);
unlock(Y)
read_item(X);
$X := X + Y$;
write_item(X);
unlock(X);

T_2'

read_lock(X);
read_item(X);
write_lock(Y);
unlock(X)
read_item(Y);
$Y := X + Y$;
write_item(Y);
unlock(Y);

Deadlock may also happen in 2PL.

+ Durability and Recovery

- All-or-nothing
  - When a transaction is submitted to a DBMS, all the operations are completed successfully and their effect is recorded permanently, OR
  - The transaction has no effect on the database or any other transactions
- What happens if a failure occurs during a transaction (or in a schedule of transactions)?
  - Failure types include: computer failure or system crash, transaction or system errors, local errors or exception conditions, concurrency control enforcement, disk failure, physical problems…
  - Unintended vs. malevolent failures, single vs. multiple failures, detectable vs. undetectable failures…

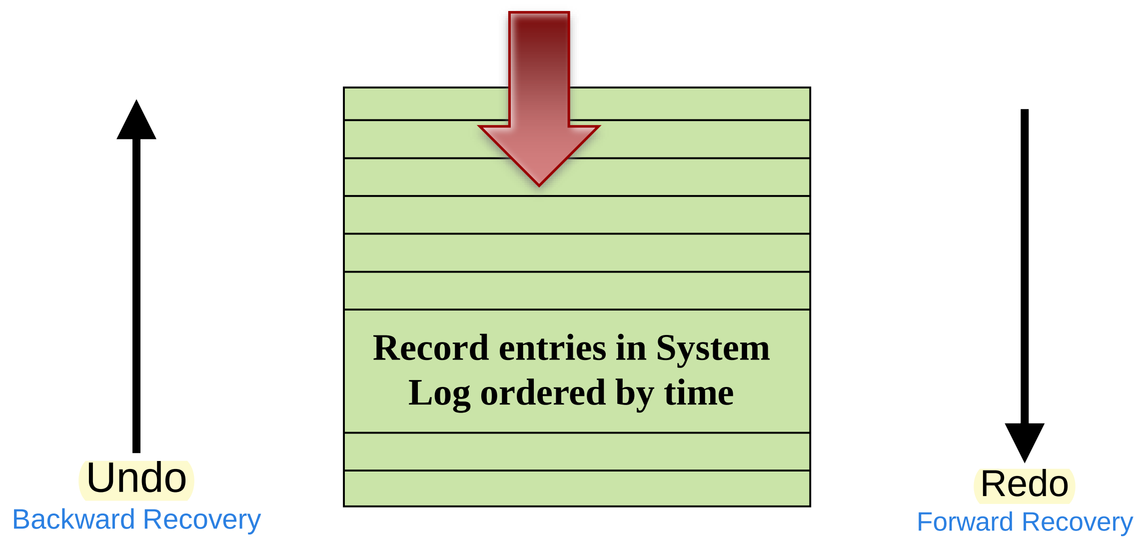
Recoverable Schedule :
A transaction in schedule cannot commit until the transactions it reads from are all committed.

+ Scheduling Transactions in DBMS

- In practice, we do not actually test for serializability. Instead, protocols or rules are developed to guarantee that a schedule will be serializable.
- In other words, **there is no pre-emptive and deterministic scheduling** tasks to be performed by DBMS for the transactions but a non-preemptive and non-deterministic protocol (set of rules) is used to guide interleaves of the transactions.

+ System Log

- A journal that keeps track of all transaction operations on database items
- This is to facilitate recovery from failure



Write Ahead Logging (WAL)

1. Must force-write the log record for an update *before* the corresponding data page is written (becomes *dirty*).

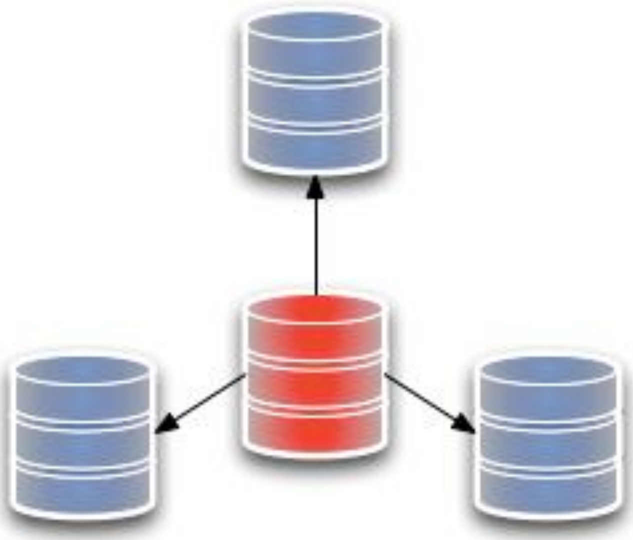
2. Must write all log records for a transaction *before commit*.

- The first guarantees Atomicity.
- The second guarantees Durability.

+ Types of Log Information

- REDO type log entry (recording after image of data Item)
  - Purpose: redo by reinstating the after image of modified data
- UNDO type log entry (recording before image of data Item
  - Purpose: undo by reinstating the before image of modified data
- Checkpoints: forced write of data to disks
  - Periodically: every m minutes, or every t transactions
  - Only REDO after the last checkpoint
- Commits
  - Marks the successful completion of T

+ Updating Distributed Data



+ Updating Distributed Data

- Synchronous Replication (Eager approach)
  - “All copies” of a modified relation (fragment) must be updated <u>before</u> the modifying transaction commits
  - Data distribution is made transparent to users
- Asynchronous Replication (Lazy approach)
  - Copies of a modified relation are only periodically updated; different copies may get out of sync in the meantime
  - Users must be aware of data distribution
  - Most products follow this approach

+ Synchronous Replication

- Voting: A transaction must write a majority of copies to modify an object; must read enough copies to be sure of seeing **at least one most recent copy**
  - E.g., 10 copies; 7 written for update; 4 copies read
  - Each copy has a version number
  - The copy with the **highest** version number is current
  - Not attractive usually because reads are common

Why?

- Read-any Write-all: In this strategy, writes are slower and reads are faster, relative to a voting-based strategy
  - A very common approach to support synchronous replication

Which one is better?

+ Cost of Synchronous Replication

- Before an update transaction can commit, it must obtain locks on all copies to modify
  - Sends lock requests to remote sites, and while waiting for the response, holds on to all other locks!
  - If sites or links fail, transaction cannot commit until they are back up

- So the alternative (asynchronous replication) is more attractive

+ Asynchronous Replication

- Allows the modifying transaction to commit <u>before all copies have been changed</u> (and readers look at just one copy)
  - Users must be aware of which copy they are reading, and that copies may be out-of-sync (for a short period of time)

- Two approaches: Primary Site and Peer-to-Peer replication
  - Both use "master copies". The difference lies in how many copies are "updatable" (i.e., "master copies")

+ Primary Site Replication

- Exactly one copy of a relation is designated the Primary or Master copy. Replicas at other sites cannot be directly updated
  - The primary copy is published
  - Other sites subscribe to (fragments of) this relation; these are secondary copies
- Main issue: How are the changes to the primary copy propagated to the secondary copies?
  - Done in two steps: (1) 'capture' changes made by committed transactions; (2) 'apply' these changes

Page 460, Chap13, Ozsu & Valduriez Book

…reduced concurrency and single-point-of-failure

+ Peer-to-Peer Replication

- More than one of the copies of an object can be a Master in this approach
- Changes to a Master copy must be propagated to other copies somehow
- If two Master copies are changed in a conflicting manner, this must be resolved
- Best used when conflicts do not arise:
  - E.g., Each Master site owns a disjoint fragment
  - E.g., Updating rights owned by one master at a time

+ Distributed Transactions

- Centralized transaction
  - …is a single process running on a single processor

- Distributed transaction
  - …involves multiple sub-transaction processes running on multiple processors (i.e., beyond consistency among multiple versions of the data discussed so far)

+ Commit in Centralized Databases

- All database access operations of the transaction have been <u>executed successfully</u> and their effect on the database has been <u>recorded</u> in the log

- System log entries:
  1. [start-transaction, T];
  2. [read-item, T, X];
  3. [write-item, T, X, old-value, new-value];
  4. [commit, T];

- After the commit point, the transaction T is said to be committed

+ Commit in Distributed Databases?

- Distributed DBMSs may encounter problems not found in a centralized environment.
  - Failure of an individual site
  - Failure of communication links

- If sub-transactions of a transaction Tx execute at different sites, **all or none** must commit.

- Need **a commit protocol** to achieve this.

- Recording the effect of database access operations in the (local) system log is not a sufficient condition for distributed commit.

+ Two-Phase Commit Protocol

- **Two-Phase Commit** Protocol ≠ 2PL
  - 2PL ⟹ serializability
  - 2PC ⟹ atomic transactions
- Site at which Tx originates is **coordinator**
- Other sites at which it executes are **subordinates**.

+ Two-Phase Commit Protocol

- When an Tx wants to commit:
- Coordinator sends **prepare** msg to each subordinate.
- Subordinate writes an **abort** or **prepare** log record and then sends a **no** or **yes** msg to coordinator.

End of *commit-request (voting)* phase,

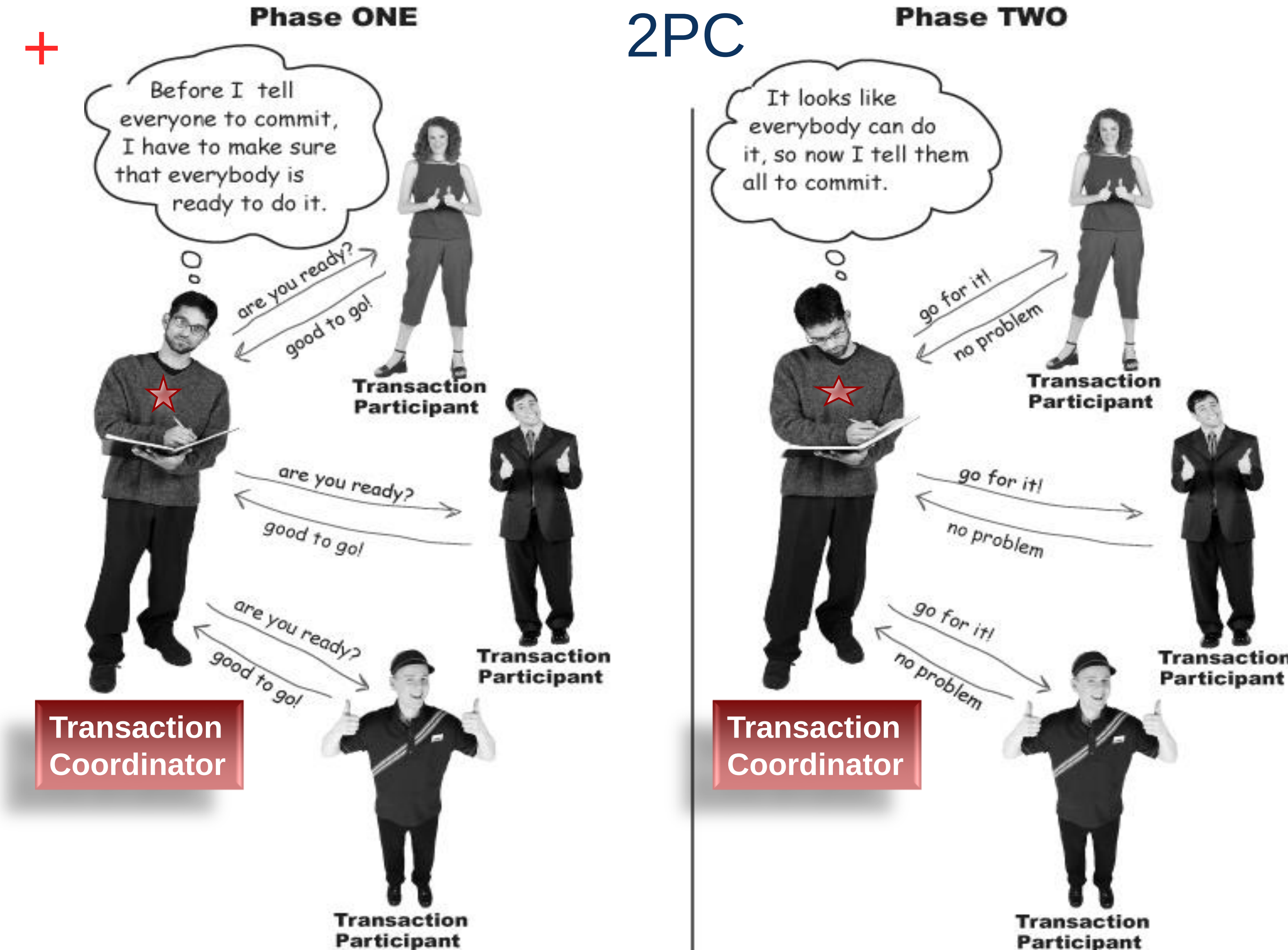
and beginning of *commit phase*

+ Two-Phase Commit Protocol

- If coordinator gets **unanimous** *yes* votes:
  - The coordinator writes a *commit* log record and sends commit message to all subordinates
- Else
  - writes *abort* log rec, and sends abort message
- Subordinates *write* abort/commit log record based on the message they get, then send **ack** message to coordinator
- Coordinator writes end log record after getting all **ack**s

Page 438, Chap12, Ozsu & Valduriez Book

2PC


Phase ONE
Before I tell everyone to commit, I have to make sure that everybody is ready to do it.
are you ready?
good to go!
Transaction Participant
are you ready?
good to go!
Transaction Participant
are you ready?
good to go!
Transaction Participant
Transaction Coordinator
Phase TWO
It looks like everybody can do it, so now I tell them all to commit.
go for it!
no problem
Transaction Participant
go for it!
no problem
Transaction Participant
go for it!
no problem
Transaction Participant
Transaction Coordinator

+ Conclusions of Distributed TM

- ACID property is enforced for DDB to handle concurrent transactions.
- A correct schedule of concurrent transactions is equivalent to a serial of transactions.
- There may be conflict operations in a non-serial schedule, which has interleaved operations of transactions.
- Some conflict schedules are serializable and recoverable.
- 2PL protocols are used to guarantee the serializability of a schedule.
- Deadlock could happen in a serializable schedule.
- A distributed transaction can be committed or aborted, by using 2PC protocol.
- There are many important relevant concepts in Distributed TM, such as Interleaving transactions, WAL, Checkpoint, Rollback, Abort, Commit, Redo, Undo, Synchronous/Asynchronous Replication, etc.

Real DDB & Advanced Systems

- Blockchain (More in COMS4507)
- GFS, HDFS, MapReduce (More in INFS3208 or DATA7201)
- NoSQL
  - Key-Value Store, Document Store, Column-Family Store
  - Graph Database

- Next week: Data Warehouse Design

+ Recommended Readings

- Elmasri & Navathe, 6th edition
  - Chapter 25: Distributed Databases
- Elmasri & Navathe, 7th edition
  - Chapter 23: Distributed Database Concepts
- Ozsu & Valduriez: *Principles of Distributed Database Systems,* 3rd edition, Springer
  - Chapter 10: Introduction to Transaction Management
  - Chapter 11: Distributed Concurrency Control
  - Chapter 12: Distributed DBMS Reliability
  - Chapter 13: Data Replication

Textbook (PDF): **https://link.springer.com/book/10.1007/978-1-4419-8834-8**

https://seu1.org/files/level6/IT344/Fundamentals_of_Database_Systems,_6th_Edition.pdf
https://www.auhd.site/upfiles/elibrary/Azal2020-01-22-12-28-11-76901.pdf