SNS COLLEGE OF ENGINEERING

Kurumbapalayam (Po), Coimbatore – 641 107

An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

COURSE NAME : 23CST207 - DATABASE MANAGEMENT SYSTEMS

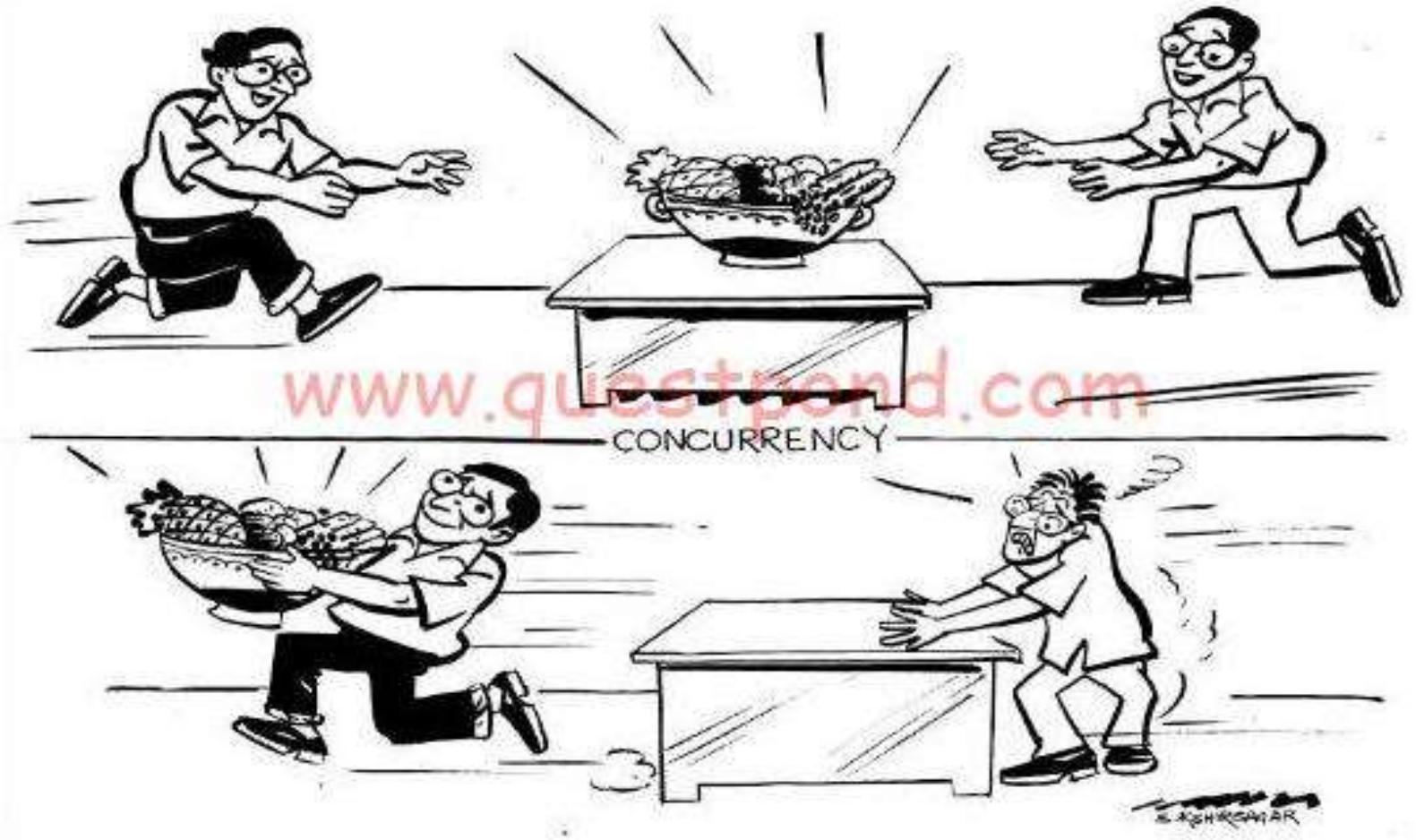
II YEAR / IV SEMESTER

Unit 2- Transactions

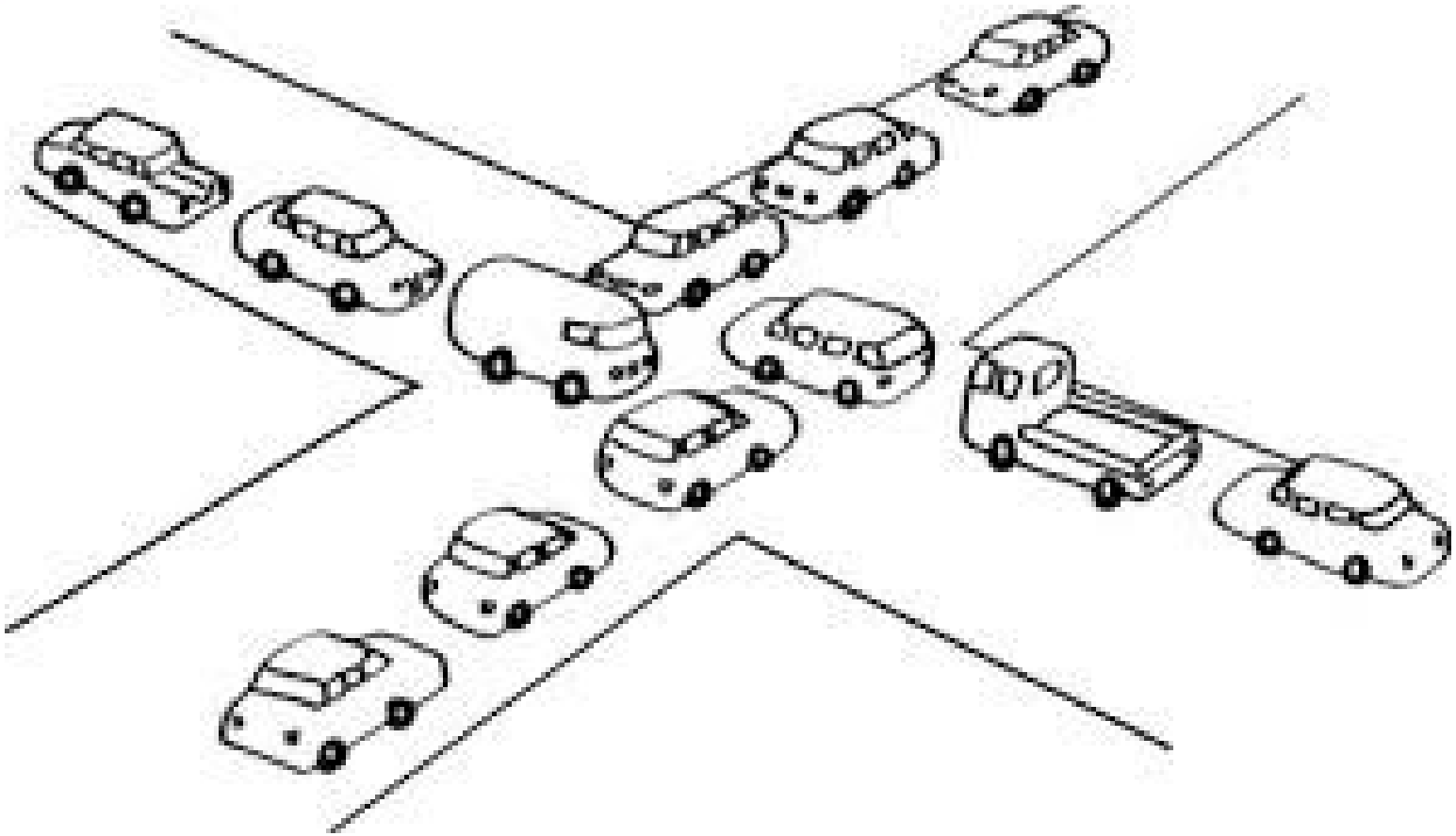
Topic 6 : **Concurrency Control**

Concurrency control

- **Concurrency control** is a database management systems (**DBMS**) concept that is used to address conflicts with the simultaneous accessing or altering of data that can occur with a multi-user system



EXAMPLE - ROAD TRAFFIC



Why do we need concurrency control?

- If DBMS allows both of their actions concurrently, it would be impossible to recognize which data is updated. Or, if DBMS choose one of the changes, the other one becomes meaningless. ... Both of solutions make the database inconsistent and useless.

(OR)

- If transactions are executed *serially*, i.e., sequentially with no overlap in time, no transaction concurrency exists. However, if concurrent transactions with interleaving operations are allowed in an uncontrolled manner, some unexpected, undesirable result.

- The lost update problem: A second transaction writes a second value of a data-item (datum) on top of a first value written by a first concurrent transaction, and the first value is lost to other transactions running concurrently which need, by their precedence, to read the first value. The transactions that have read the wrong value end with incorrect results.

- The dirty read problem: Transactions read a value written by a transaction that has been later aborted. This value disappears from the database upon abort, and should not have been read by any transaction ("dirty read"). The reading transactions end with incorrect results.

- The incorrect summary problem: While one transaction takes a summary over the values of all the instances of a repeated data-item, a second transaction updates some instances of that data-item. The resulting summary does not reflect a correct result for any (usually needed for correctness) precedence order between the two transactions (if one is executed before the other), but rather some random result, depending on the timing of the updates, and whether certain update results have been included in the summary or not.

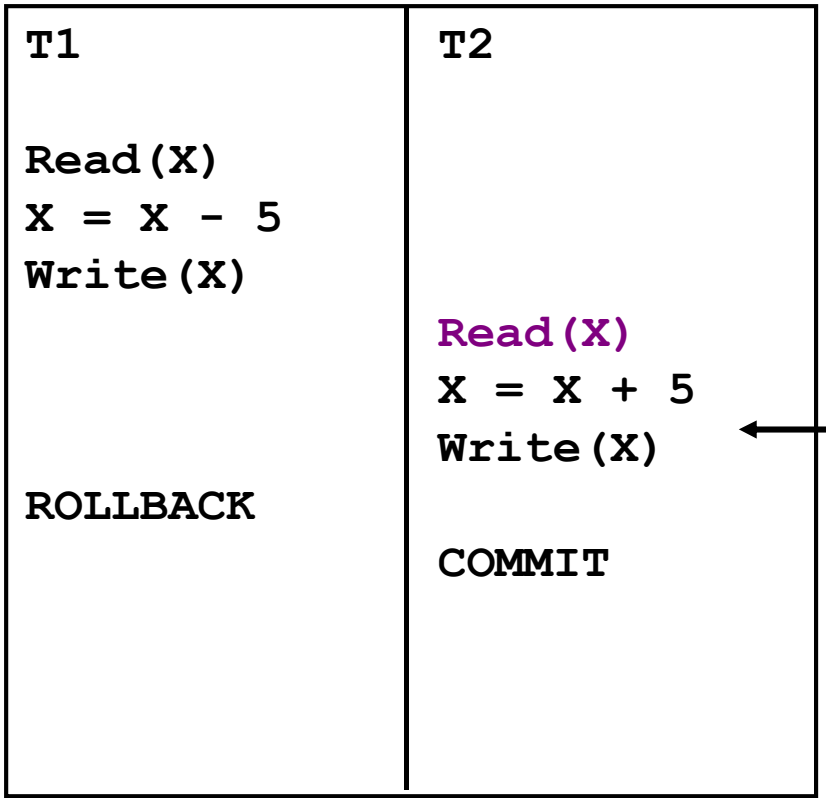
Lost Update

T1	T2
Read(X)	
X = X - 5	
	Read(X)
	X = X + 5
Write(X)	
	Write(X)
COMMIT	
	COMMIT

This update Is lost

Only this update succeeds

Uncommitted Update ("dirty read")



Inconsistent analysis

T1	T2
Read(X)	
X = X - 5	
Write(X)	
	Read(X)
	Read(Y)
	Sum = X+Y
Read(Y)	
Y = Y + 5	
Write(Y)	

Summing up data while it is being updated

Concurrency Control

- **Concurrency Control Mechanisms**
- The main categories of concurrency control mechanisms are:
- **Optimistic**
- **Pessimistic**
- **Semi-optimistic**

Optimistic

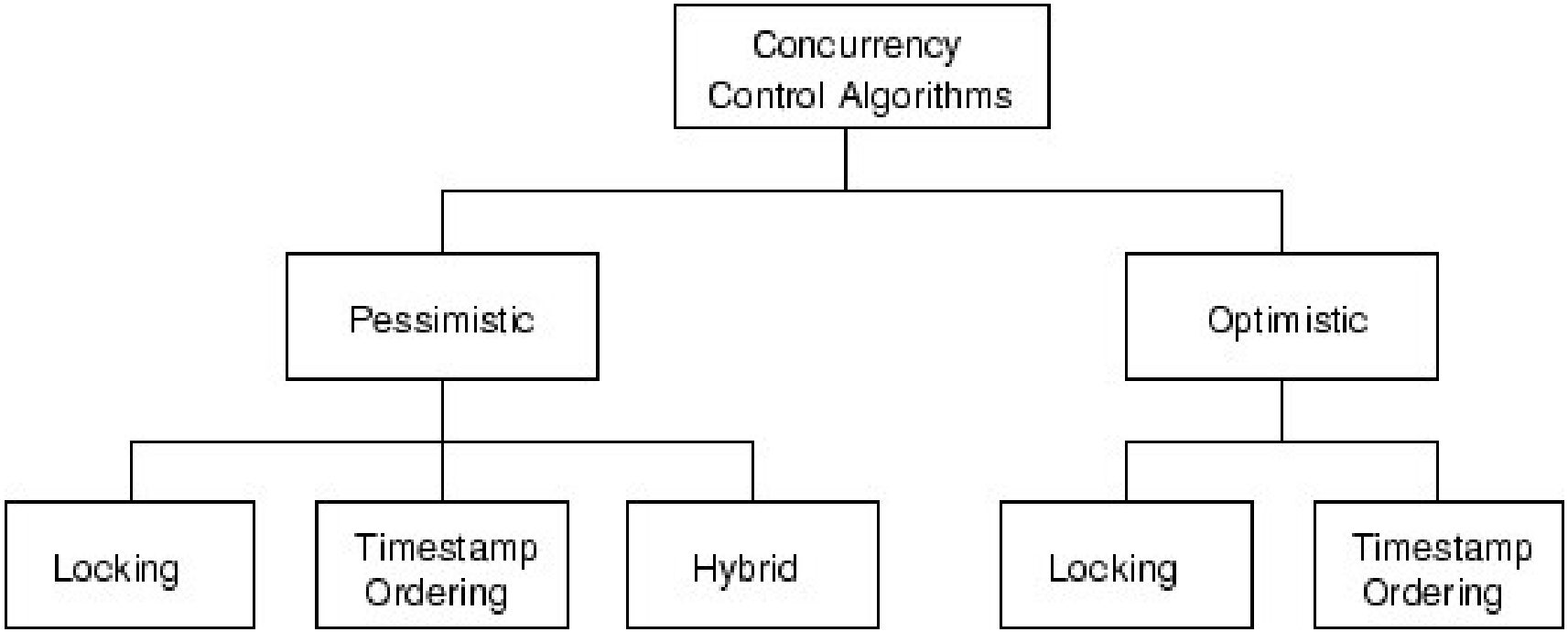
- Delay the checking of whether a transaction meets the isolation and other integrity rules (e.g., serializability and recoverability) until its end, without blocking any of its (read, write) operations ("...and be optimistic about the rules being met..."), and then abort a transaction to prevent the violation, if the desired rules are to be violated upon its commit.
- An aborted transaction is immediately restarted and re-executed, which incurs an obvious overhead (versus executing it to the end only once). If not too many transactions are aborted, then being optimistic is usually a good strategy.

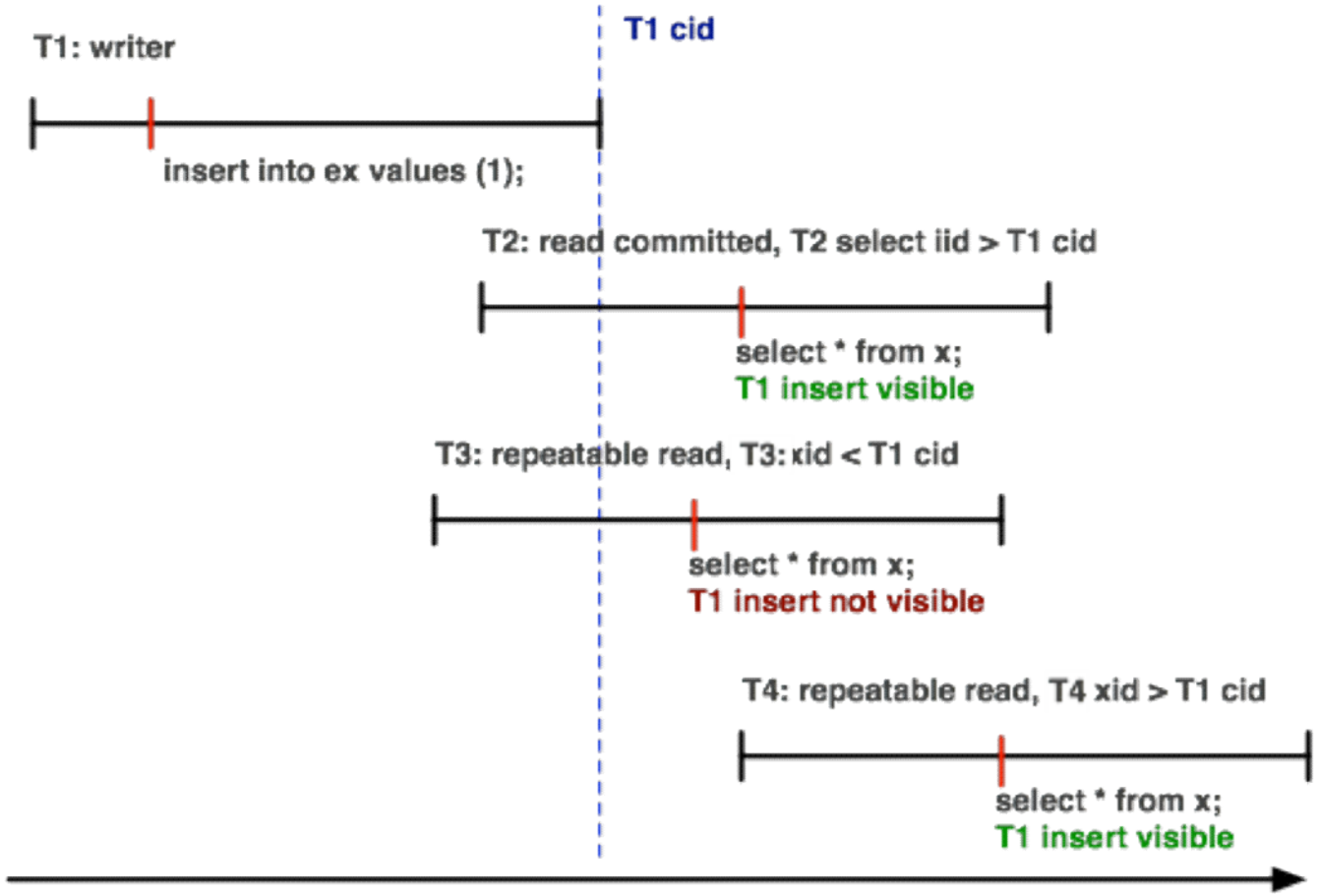
Pessimistic & Semi-optimistic

- **Pessimistic** - Block an operation of a transaction, if it may cause violation of the rules, until the possibility of violation disappears. Blocking operations is typically involved with performance reduction.
- **Semi-optimistic** - Block operations in some situations, if they may cause violation of some rules, and do not block in other situations while delaying rules checking (if needed) to transaction's end, as done with optimistic.

Methods

- Locking (e.g., **Two-phase locking** - 2PL) - Controlling access to data by locks assigned to the data. Access of a transaction to a data item (database object) locked by another transaction may be blocked (depending on lock type and access operation type) until lock release.
- **Serialization graph checking** (also called Serializability, or Conflict, or Precedence graph checking) - Checking for cycles in the schedule's graph and breaking them by aborts.
- **Timestamp ordering** (TO) - Assigning timestamps to transactions, and controlling or checking access to data by timestamp order.
- **Commitment ordering** (or Commit ordering; CO) - Controlling or checking transactions' chronological order of commit events to be compatible with their respective precedence order.


Concurrency Control Algorithms
Pessimistic
Optimistic
Locking
Timestamp Ordering
Hybrid
Locking
Timestamp Ordering


T1 cid
T1: writer
insert into ex values (1);
T2: read committed, T2 select iid > T1 cid
select * from x;
T1 insert visible
T3: repeatable read, T3: xid < T1 cid
select * from x;
T1 insert not visible
T4: repeatable read, T4 xid > T1 cid
select * from x;
T1 insert visible

EVALUATION

- List out the **Concurrency Control Mechanisms**

A)_____

B)_____

C)_____

ANSWER

A)Optimistic

B) Pessimistic

C) Semi-optimistic

THANK YOU