Relational Databases Management Issues (Part 2)

Overview

- Snapshot isolation
- Timestamping concurrency control
- Security
  - This week

- Recovery in DBMSs
  - Next week

- The lecture slides are based on the ones provided with the textbooks:
  - Connolly. and Begg, C. "*Database Systems - A Practical Approach to Design, Implementation, and Management*.", 6th Ed., Pearson Education Ltd
  - Korth, H., Silberschatz, A. and Sudarshan S., *"Database Systems Concepts*.". 7th Ed., International Ed, McGraw Hill, 2019 (ISBN 9781260084504)

Snapshot Isolation (SI*) – An alternative isolation level (to Serializable)

- Each transaction works with a "*snapshot*" of the database, valid at the time the transaction began.
  - Same reads (SELECTs) of a transaction see *consistent* data *[recall data anomalies we discussed last week]*.
  - A transaction cannot read updates made by concurrent transactions
- Ensures no *write-write* conflict occurs: two concurrent transactions CANNOT write to the same data-item – the necessary rule for commit
  - The likelihood of a *write-write* conflict is dependent on: the number and profile of concurrent transactions - workload; the state of the database; the size of the database

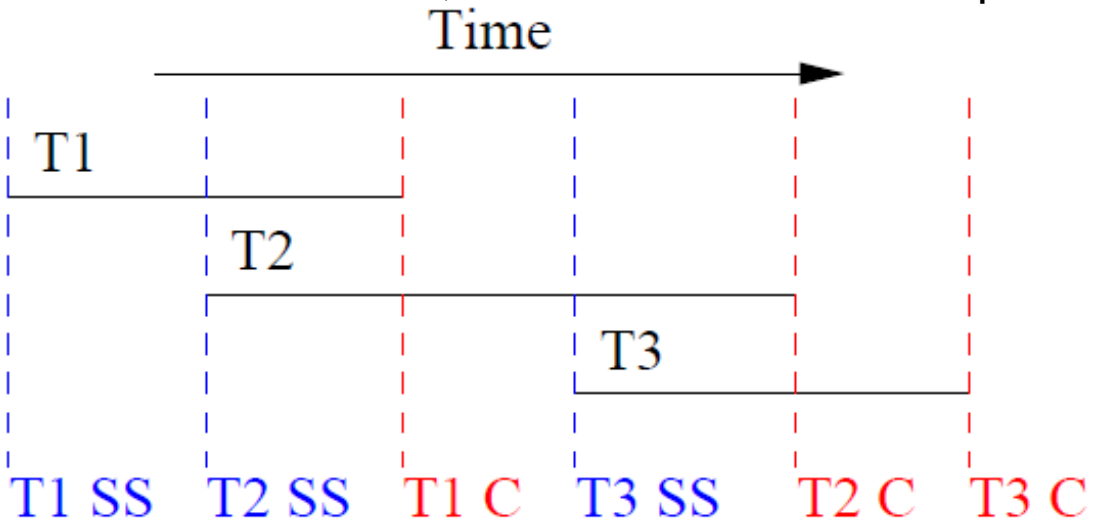
- *See e.g. Silbershatz et al. 18.8, and especially 18.8.3*

*Berenson, Hal; Bernstein, Phil; Gray, Jim; Melton, Jim; O'Neil, Elizabeth; O'Neil, Patrick (1995), "A Critique of ANSI SQL Isolation Levels", Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, pp. 1–10, doi:10.1145/223784.223785

Snapshot Isolation (SI) – An alternative isolation level (to Serializable)

- "Readers never conflict with writers" in SI
- "Update transaction" / "Write transaction" – at least one *write* operation (in SQL: DELETE, INSERT, UPDATE)

- Read-only transactions *always* allowed to progress
- NOTE: two transactions, T_i and T_j, **concurrent** iff: $b_i < c_j < c_i$
  - "concurrent". i.e. "simultaneous", is *intransitive* relationship



The image is from B Vandiver's PhD thesis, MIT 2008, p.79

Validation approaches for Update/Write Transactions (FCW)

- "**First committer wins**" (FCW) and "**First updater wins**" (FUW)

- "**First committer wins**"- when a transaction T_i *enters the partially committed state*, the following actions are taken in an *atomic* action:
  - A test is made to see if any transaction that was *concurrent* with T_i has already updated some data item that T_i intends to write and committed.
  - If some such transaction is found, then T_i aborts.
  - If no such transaction is found, then T_i commits and its updates are written to the database.
  - NB: In Oracle this check is enforced not by validation at commit-time, but by checks done at the time of updating (see next slide(s))

Validation approaches for Update Transactions (FUW)

"**First updater wins**" – for the system that uses a locking CC mechanism (applies only to writes!). When a transaction *T_i attempts to update a data item*, it requests a write lock on the data item; then:

1. If the *lock is not held* by a concurrent transaction:
   a. If the item has been updated by any concurrent transaction which had committed, then T_i aborts.
   b. Otherwise T_i may proceed with its execution, including *possibly* committing later on.
2. If another concurrent transaction T_j *already holds a write/exclusive lock,* T_i waits until T_j aborts or commits:
   a. If T_j aborts, then the lock is released and T_i *can* obtain the lock. Once the lock is acquired, the check for an update by any other concurrent transaction is performed as described earlier (see 1.)
   b. If T_j commits, then T_i must abort. Locks are released when the transaction commits or aborts.

An alternative: use "NOWAIT"

The ultimate effect of FCW and FUW is the same:
One of the concurrent transactions attempting to update the same data item aborts

First Committer Wins (FCW)

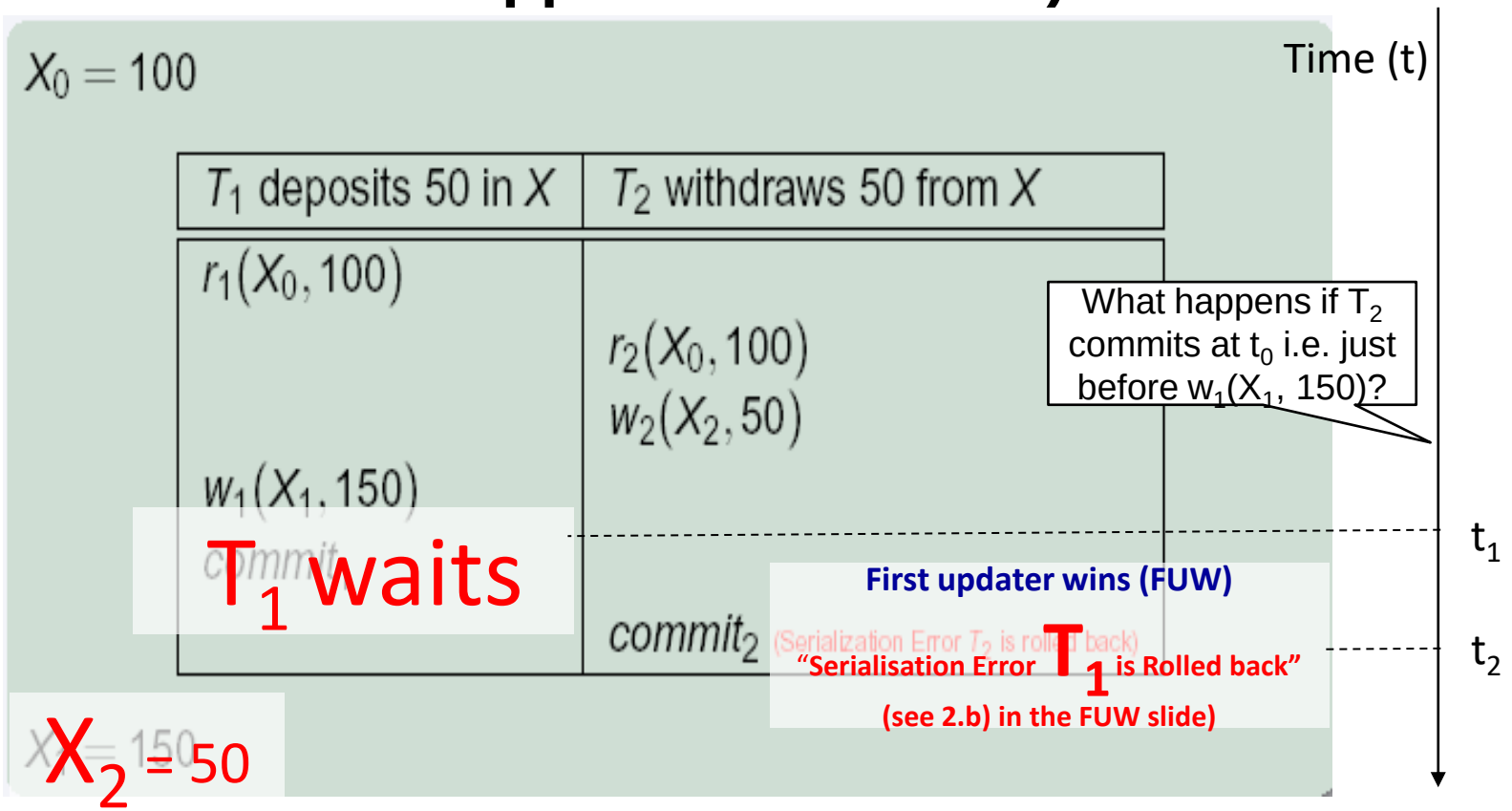
$X_0 = 100$

T_1 deposits 50 in X	T_2 withdraws 50 from X
$r_1(X_0, 100)$	
	$r_2(X_0, 100)$
	$w_2(X_2, 50)$
$w_1(X_1, 150)$	
$commit_1$	
	$commit_2$ (Serialization Error T_2 is rolled back)

$X_1 = 150$

Time

First Updater Wins (FUW) (Lock-based CC approach assumed)



- Check for concurrent updates **when write occurs** by **locking** the item
  - Differs only in when abort occurs, otherwise equivalent to FCW
  - Lock(s) held till all concurrent transactions have finished - Oracle uses this plus some extra features
  - Lock(s) held until the transaction has finished – PostgreSQL (it offers Serializable Snapshot Isolation)

Snapshot Isolation (cont.)

- The main reason for its adoption:
  - Allows better performance than Serializable isolation level
  - Avoids most of the concurrency issues that serializable level avoids (but not always all)
    - There exist schedules of transaction operations that are valid under SI but invalid under Serializable isolation level, and vice-versa!

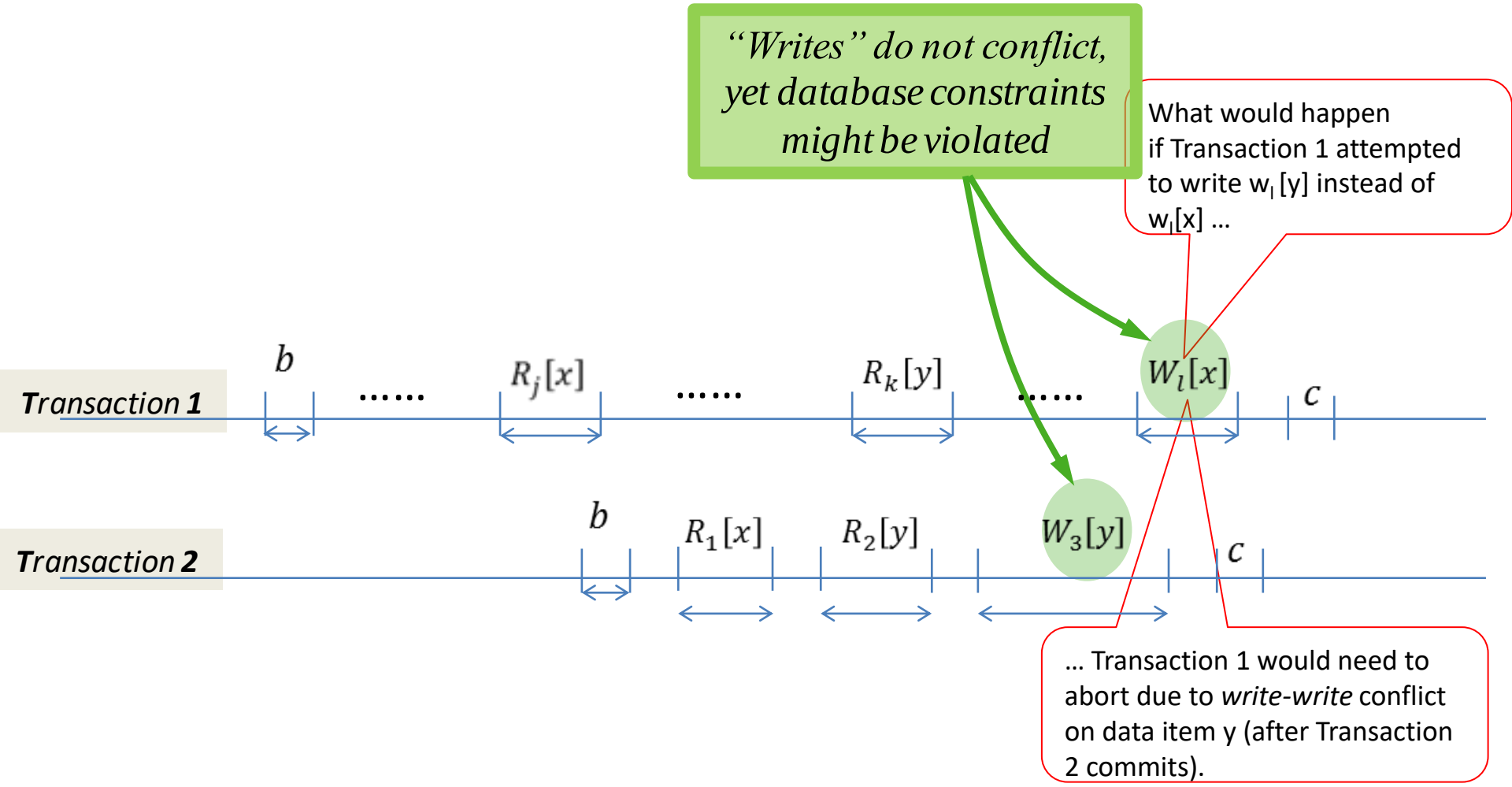
- (Usually) implemented with multiversion concurrency control (MVCC),
  - Multiple values of each data item (versions) maintained
    - A new version generated each time data item written to, thus allowing concurrent transactions to read different (corresponding) versions
  - Common way to increase concurrency / performance
  - MVCC/SI available in major Relational DBMSs: Oracle, MSSQL (since v.2005), PostgreSQL, Firebird (claims to be (one of) the first DBMS products to have used MVCC!) etc. Also, in some NoSQL, e.g. MongoDB, etc.

Snapshot Isolation (cont.)

- SI used to critique the ANSI SQL-92 standard's definition of isolation levels:
  - exhibits none of the data anomalies that the SQL standard prohibited (see last Week slides), YET NOT conflict serializable (according to the serializability theory)
    - E.g. “phantom reads” cannot happen since a transaction always reads from the same data snapshot.
  - Whether a particular schedule of transactions executed under SI is serializable or not depends on the application, i.e. profile/workload of concurrent operations submitted to the DBMS
  - E.g., the TPC-C benchmark runs correctly under SI

- SI susceptible to *database constraint* violations, so-called *write-skew* anomalies

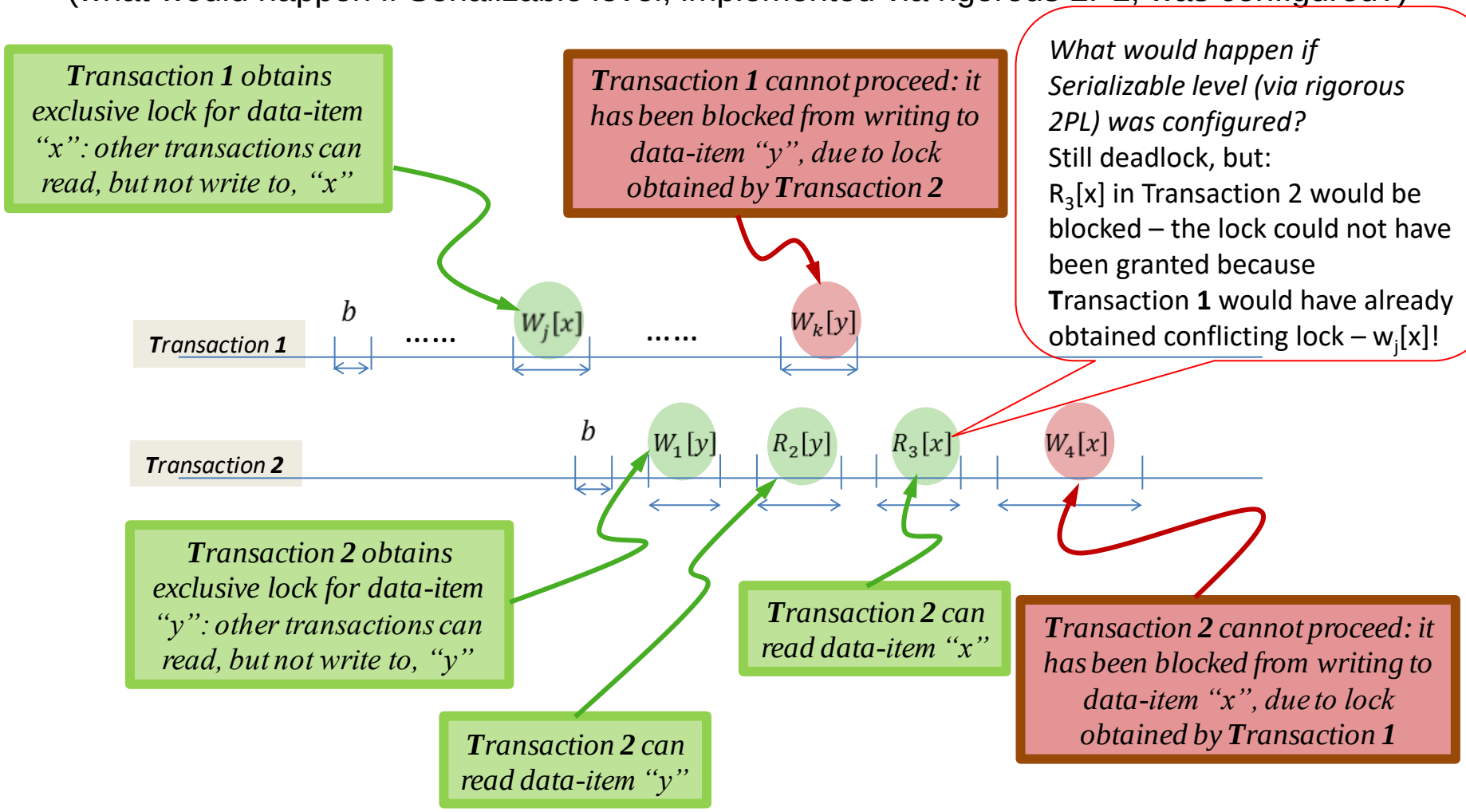
Snapshot Isolation-Level: *Write-skew* anomaly



Note: The subscripts are ordinal numbers of the SQL statements in the respective transaction.

An example of a Deadlock due to use of "write locking on row level" mechanism to implement SI

(what would happen if Serializable level, implemented via rigorous 2PL, was configured?)



Timestamping (TS) – another CC mechanism

- Transactions ordered globally so that *older* transactions - transactions with *smaller* timestamps - *get priority* in the case of conflict.
- Conflict is resolved by rolling back and restarting transaction.
  - New timestamps must be assigned to restarted transactions
- No locks so no deadlock.
  - But *liveness* might be an issue; e.g. a long transaction might be made to repeatedly restart by the conflicting short-lived transactions

Timestamp

A unique identifier created by DBMS that indicates relative starting time of a transaction.

- E.g. generated by using system clock at the time transaction started, or by incrementing a logical counter every time a new transaction starts.

- NB: transaction timestamps: a similar concept used for a **different** purpose – *Deadlock prevention* (see previous week's slides)

Timestamping

- TS protocol ensures that any conflicting read or write operations are executed in timestamp order.
  - *Write* proceeds if the *last update and read on that data item* was carried out by an older transaction.
  - *Read* proceeds if the *last update on that data item* was by an older txn
  - Otherwise, transaction requesting read/write is restarted and given a new timestamp.
- Also, there exist timestamps for data items:
  - read-timestamp(x) - timestamp of the most recent transaction to successfully read the item x;
  - write-timestamp(x) - timestamp of the most recent transaction to successfully write the item x.

- Recall: a clash between two operations exists *iff* at least one of them is a write.

Timestamping – Txn T issues Write(x)

- Consider a transaction T with timestamp ts(T):

ts(T) < read_timestamp(x)

  - x already read by a younger (later) transaction.
  - Roll back transaction T, and restart it with a new, more recent timestamp.

ts(T) < write_timestamp(x)

  - x already updated by a younger (later) transaction; T is trying to write an **obsolete** value.
  - Roll back transaction T, and restart it with a new, more recent timestamp.
    - Or Thomas's Write rule: the Write can be **ignored** – "*ignore obsolete write*" rule
    - Thus, we allow a greater level of concurrency and avoid unnecessary rollbacks.

- Otherwise, the write(x) by T is executed
  - We set write_timestamp(x) := ts(T)

See e.g. http://www.cs.ubc.ca/~welu/thomas.pdf (just a 2-page doc!) for an explanation, and a correctness proof, of Thomas's Write Rule, as well as Timestamping CC in general.

Timestamping - Txn T issues Read(x)

Consider a transaction T with timestamp ts(T):

ts(T) < write_timestamp(x)

- x already updated by a younger transaction.
- Roll back transaction T and restart T with a new, more recent timestamp.

- Otherwise, the operation read(x) by T is executed.
  - We set read-timestamp(x) := *max*{read-timestamp(x), ts(T)}.

Example – Basic Timestamp Ordering

And an example of *ignore obsolete write* rule – Thomas's write rule – obsolete write can be discarded

Time	Op	T_{19}	T_{20}	T_{21}
t_1		begin_transaction		
t_2	read(bal_x)	read(bal_x)		
t_3	$bal_x = bal_x + 10$	$bal_x = bal_x + 10$		
t_4	write(bal_x)	write(bal_x)	begin_transaction	
t_5	read(bal_y)		read(bal_y)	
t_6	$bal_y = bal_y + 20$		$bal_y = bal_y + 20$	begin_transaction
t_7	read(bal_y)			read(bal_y)
t_8	write(bal_y)		write(bal_y)+	
t_9	$bal_y = bal_y + 30$			$bal_y = bal_y + 30$
t_{10}	write(bal_y)			write(bal_y)
t_{11}	$bal_z = 100$			$bal_z = 100$
t_{12}	write(bal_z)			write(bal_z)
t_{13}	$bal_z = 50$	$bal_z = 50$		commit
t_{14}	write(bal_z)	write(bal_z)‡	begin_transaction	
t_{15}	read(bal_y)	commit	read(bal_y)	
t_{16}	$bal_y = bal_y + 20$		$bal_y = bal_y + 20$	
t_{17}	write(bal_y)		write(bal_y)	
t_{18}			commit	

+ At time t_8, the write by transaction T_{20} violates the first timestamping write rule described above and therefore is aborted and restarted at time t_{14}.

‡ At time t_{14}, the write by transaction T_{19} can safely be ignored using the ignore obsolete write rule, as it would have been overwritten by the write of transaction T_{21} at time t_{12}.

+ T20, which is an older txn than T21, attempts to write the data item ("y") at t_8, but this data item has been already read by the younger txn – T21 – at t_7 timestamp.

Security

- Security is a serious concern for most organizations.
  - Data is a valuable resource that must be strictly controlled and managed, as with any corporate resource.
  - Part or all of the corporate data may have strategic importance and therefore needs to be kept secure and confidential.
- The protection of a database against unauthorised disclosure, alteration, or destruction
  - Unauthorised: intentional or accidental
- Unit of data to be protected can range from an entire relation to a specific data value in a specific row-column position
- Generally, the security of DBMS as good as the security of the underlying OS

Security – 3 main properties

- *Confidentiality* – protection of data from unauthorised disclosure (secrecy over data)
- *Integrity* – maintaining and assuring the accuracy and consistency of data – preventing data becoming corrupt/invalid, as a result of e.g. unauthorised data access
- *Availability* – the data ought to be available when needed; need to recover (quickly) from hardware and software errors, or malicious activities

Countermeasures – Computer-based Controls

Range from physical control to administrative procedures

- Authorisation/ Authentication
- **Access control**
  - [we look in detail at this one only.]
- Views
- Backup and recovery
- Encryption
- Integrity (constraints)
- Associated procedures

Countermeasures – Access Control

- Access control
  - A privilege allows a user to access or create (that is read, or write) some database object (such as a relation, view, and index etc.) or to run certain DBMS utilities.
  - Privileges are granted to users to accomplish the tasks required for their jobs.
  - A user who creates a DB object (e.g. relation) automatically gets all privileges on that object.

- Setting improper access levels might allow unauthorised user to deduce the data from the legitimate results; e.g. HR person issuing command to check the salaries of less than £50k can infer who has more than £50k.

Countermeasures – Access Control (DAC)

- Most DBMS provide a type of Discretionary Access Control (DAC).
  - Other types: Role-based Access Control, Mandatory Access Control
- SQL standard supports DAC through the GRANT and REVOKE commands.
  - The GRANT command gives privileges to users, and the REVOKE command takes away privileges.
  - Users with certain rights can grant or revoke rights to/from other users

GRANT/REVOKE Examples

- **GRANT** SELECT **ON** TABLE S **TO** ANN **WITH GRANT OPTION**;
- **GRANT** SELECT, UPDATE (STATUS, CITY) **ON** TABLE S **TO** TOM, PETER;
- **GRANT** ALL **ON** TABLE P **TO** PUBLIC;
- **REVOKE** SELECT **ON** TABLE S **FROM** ANN;
- **REVOKE** ALL **ON** TABLE S, SP, P **FROM** SAM;

- Different treatment for allowing *specific column(s) privileges* in different DBMS products, and the SQL standard!
  - E.g. PostgreSQL and MSSQL allow SELECT privileges for specific column(s), while Oracle does not.

Required Reading

- Connolly,T. and Begg, C. "*Database Systems - A Practical Approach to Design, Implementation, and Management*.", 6th Ed., Pearson Education Ltd
  - See previous week required/recommended reading
  - Security – Sections 7.6, 20.1, 20.2.2 (NOT Multilevel Relations and Polyinstantiation).
- Korth, H., Silberschatz, A. and Sudarshan S., "Database Systems Concepts." 7th Ed., International Ed, McGraw Hill, 2019 (ISBN 9781260084504)
  - Especially relevant material about Snapshot Isolation: Sections 18.8

- This list is only indicative, and the text in these sections should be of course looked at as "additional", but required, reading to the lecture slides.
- This is certainly not an exhaustive list.
- The content of the chapters referred to might NOT be fully relevant to the module. You need to make the final decision about what is relevant and what is not yourselves, given the material that we covered in the lecture.

Coursework

- Material covered
- Structure
- Date, time, location
- Etc.