

CSCI235 Database Systems

Introduction to Transaction Processing (3)

Dr Janusz R. Getta

School of Computing and Information Technology -
University of Wollongong

Introduction to Transaction Processing

Outline

[Serialization graph](#)

[Serialization graph testing protocol](#)

[Two phase locking protocol \(2PL\)](#)

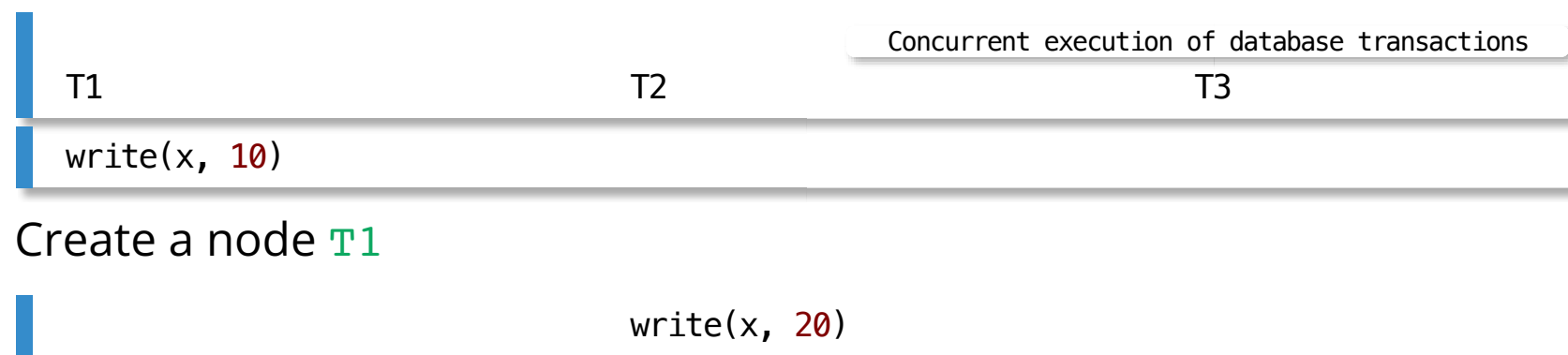
[Timestamp ordering protocol](#)

Serialization graph

Serialization graph is constructed in the following way

- If a transaction T participates in a concurrent execution then we add a node labeled with T to a **serialization graph**
- If the transactions T_i and T_j process conflicting operations such that T_i processes its operation first then we add an edge directed from T_i to T_j

Sample construction of a **serialization graph**



Serialization graph

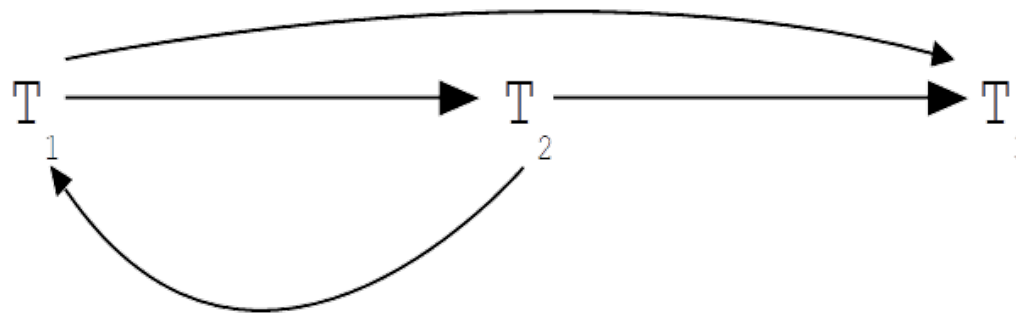
write(x,30)

Create a node **T3** and add the edges from **T1** to **T3** and from **T2** to **T3**

write(y,10)

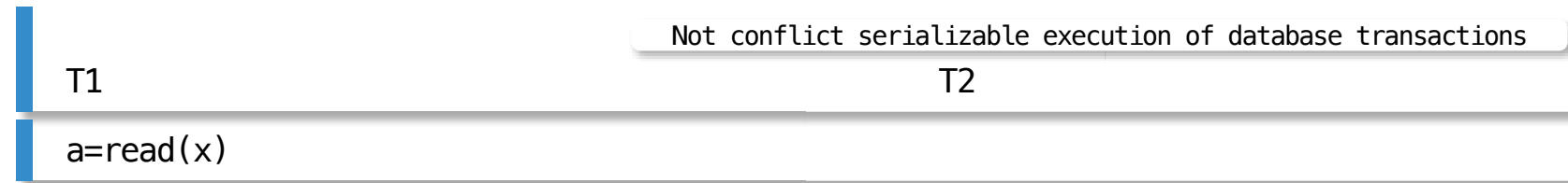
a=read(y)

Add an edge from **T2** to **T1**



Serialization graph

A serialization graph of **not conflict serializable** execution of database transactions



Create a node **T1**



Create a node **T2**



Add an edge from **T2** to **T1**



Add an edge from **T1** to **T2**

Serialization graph

A serialization graph for **not conflict serializable** execution of database transactions

T1
 a=read(x)

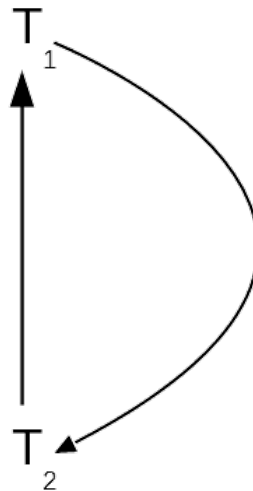
 write(x, a-10)

Not conflict serializable execution of database transactions

T2

 b=read(x)

 write(x, b+20)



Introduction to Transaction Processing

Outline

[Serialization graph](#)

[Serialization graph testing protocol](#)

[Two phase locking protocol \(2PL\)](#)

[Timestamp ordering protocol](#)

Serialization graph testing protocol (SGT)

Principles

- Scheduler maintains and tests **serialization graph**
- If an operation issued by a transaction violates conflict serializability, i.e. if it creates a cycle in **serialization graph** then such transaction is aborted

Problems

- **Cascading aborts**: if a transaction **T** that created a data item **x** is aborted then all transactions that read a new value of **x** must be aborted
- **Performance**: testing acyclicity of serialization graph has $O(n^2)$ complexity

Introduction to Transaction Processing

Outline

[Serialization graph](#)

[Serialization graph testing protocol](#)

[Two phase locking protocol \(2PL\)](#)

[Timestamp ordering protocol](#)

Two-phase locking (2PL) protocol

Principles

- Access to data items is controlled by **shared (read)** and **exclusive (write)** locks
- A transaction can **read** a data item only if a data item is **read** or **write locked** by the same transaction or a data item is **read locked** by another transaction
- A transaction can **write** a data item only if a data item is **write locked** by the same same transaction
- **Each transaction must acquire all locks before releasing any lock**

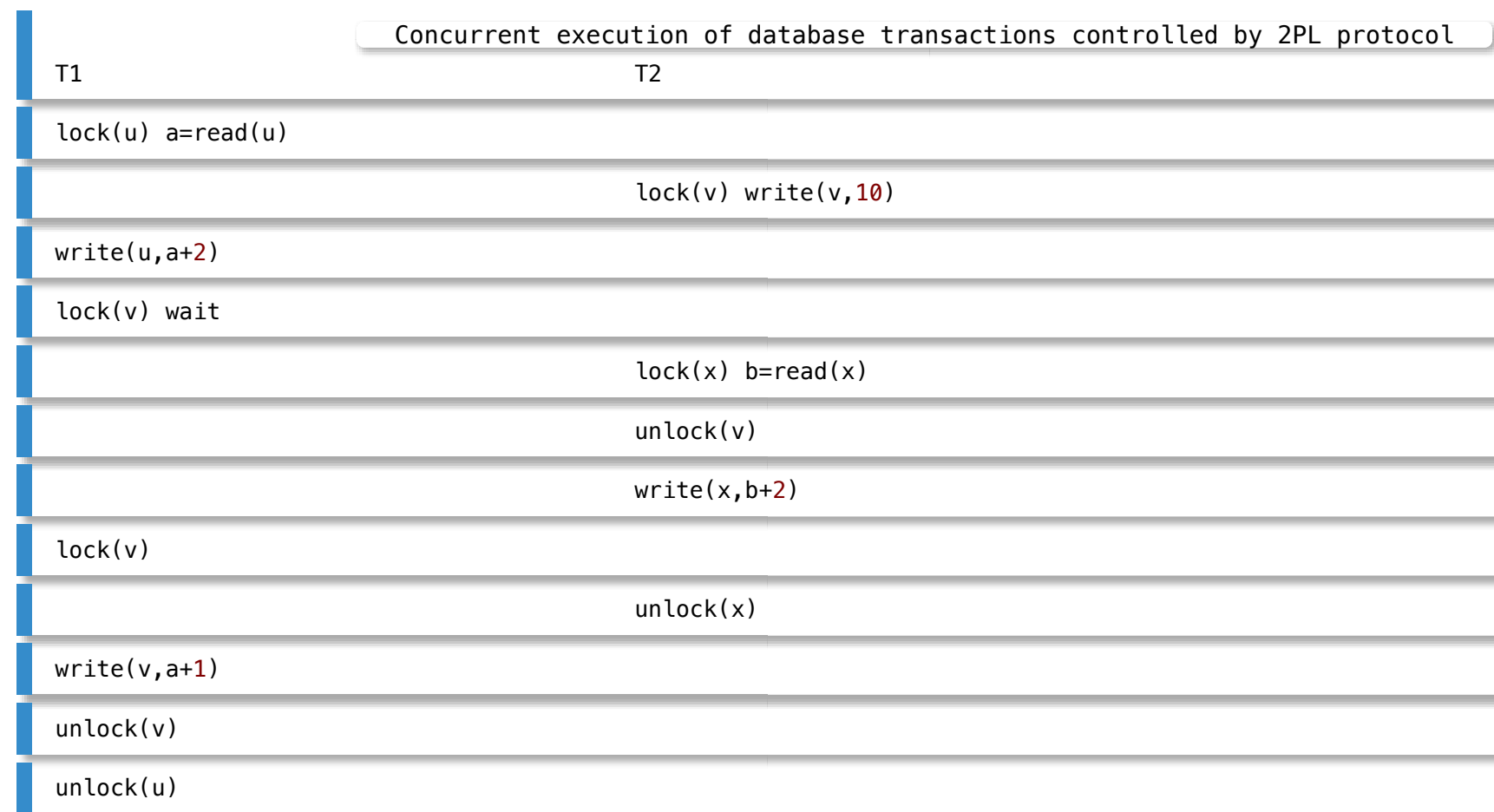
Two-phase locking protocol belongs to a class of **pessimistic protocols**

Problems

- **Deadlocks**
- **Unnecessary locks** and **delays** when an execution is conflict serializable

Two-phase locking (2PL) protocol

A sample execution controlled by **two-phase locking** protocol



In HTML view press 'p' to see the lecture notes

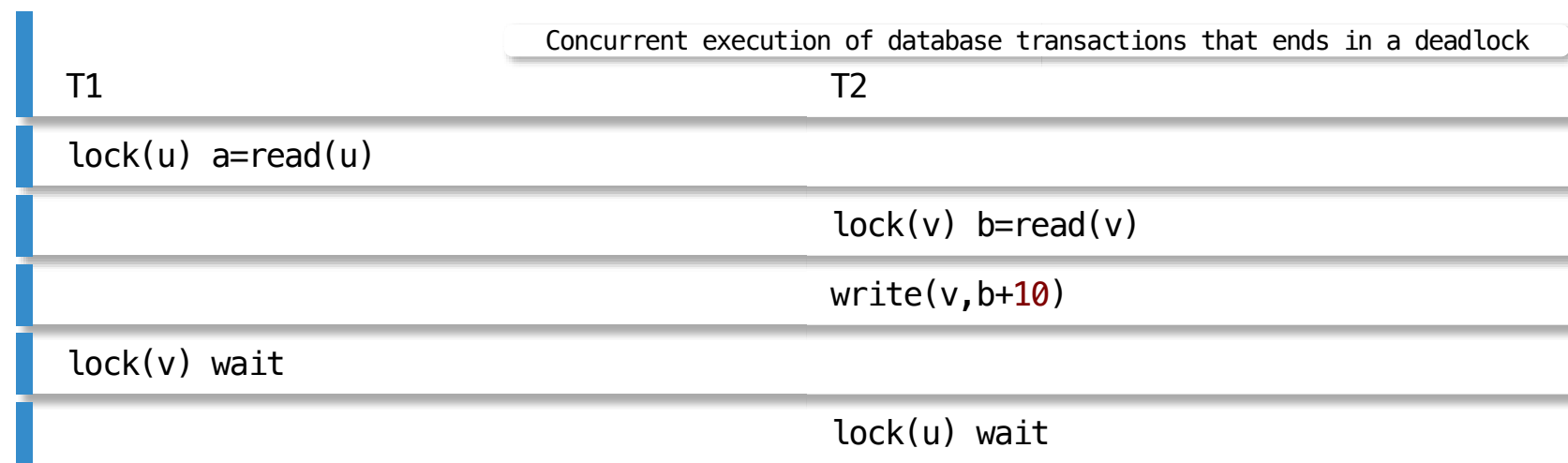
[TOP](#)

Created by Janusz R. Getta, CSCI235 Database Systems, Spring 2023

11/19

Two-phase locking (2PL) protocol

A sample execution that ends in a **deadlock**

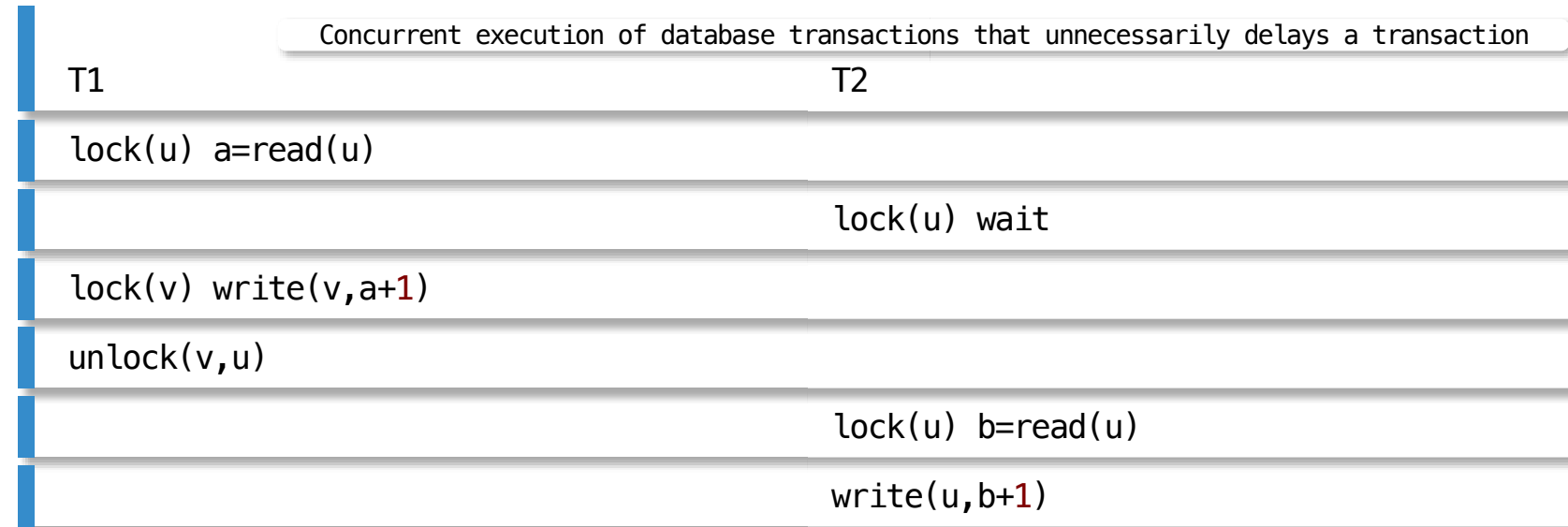


Both transactions are in a **wait** state

In a database system **deadlock** is eliminated through either **wait for graph** or through **timeout**

Two-phase locking (2PL) protocol

A sample execution that **unnecessarily delays** a transaction



The transactions **T1** and **T2** never get into **not conflict serializable** execution

Therefore, there is no need to delay a transaction **T2**

Introduction to Transaction Processing

Outline

[Serialization graph](#)

[Serialization graph testing protocol](#)

[Two phase locking protocol \(2PL\)](#)

[Timestamp ordering protocol](#)

Timestamp ordering (TO) protocol

Principles

- Each transaction obtains a **timestamp** at its start point
- Data items are **stamped** each time any transaction accesses data items in a read or write mode
- Access to data items is permitted in an **increasing order** of **timestamps**

Timestamp ordering protocol belongs to a class of **optimistic protocols**

Problems

- Cascading aborts
- Unnecessary aborts when execution is conflict serializable

Timestamp ordering (TO) protocol

A sample execution controlled by the **timestamp ordering** protocol

Concurrent execution of database transactions controlled by TO protocol		
T1	T2	x,y
timestamp(t1)		
a=read(x)		x:t1
write(x,a-10)		
	timestamp(t2)	
	write(x,10)	x:t1:t2
	b=read(y)	y:t2
write(y,a+1)		y:t2:t1
abort		

Timestamp ordering (TO) protocol

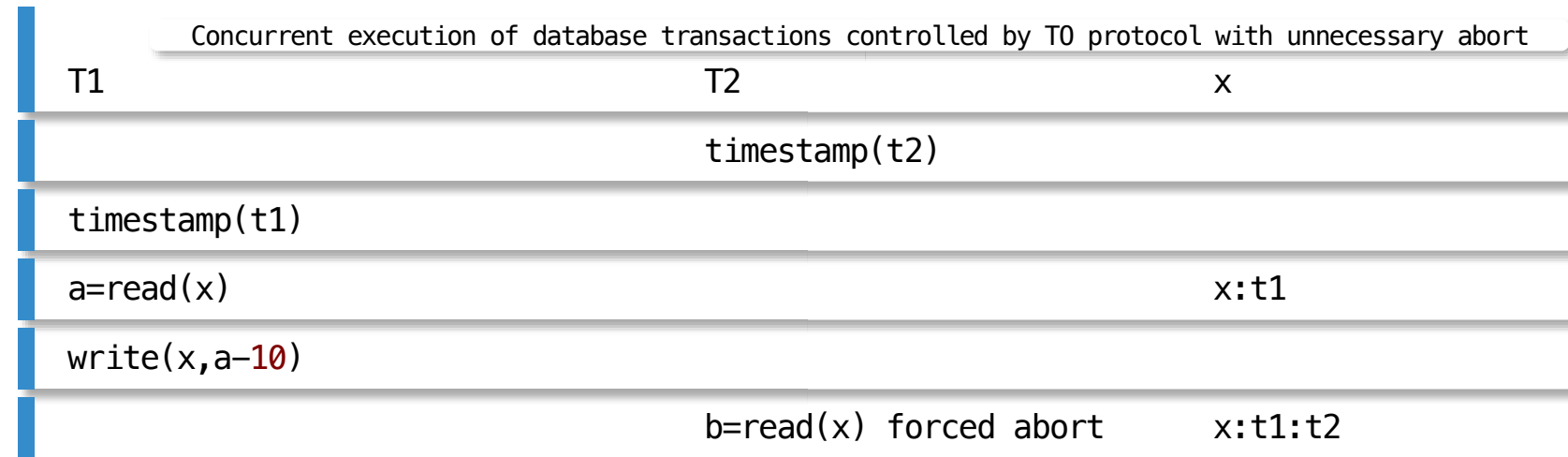
A sample execution controlled by the **timestamp ordering** protocol with **cascading abort**

Concurrent execution of database transactions controlled by TO protocol with cascading abort		
T1	T2	x
timestamp(t1)		
a=read(x)		x:t1
write(x,x-10)		
	timestamp(t2)	
	b=read(x)	x:t1:t2
fail		
	forced abort	

A transaction **T2** is forced to **abort** because it reads a **dirty data item** written by a failed transaction **T1**

Timestamp ordering (TO) protocol

A sample execution controlled by the **timestamp ordering** protocol where a transaction is **unnecessarily aborted**



A transaction **T2** is **aborted** even the execution is **conflict serializable**

References

T. Connolly, C. Begg, Database Systems, A Practical Approach to Design, Implementation, and Management, Chapter 22.2 Concurrency Control, Pearson Education Ltd, 2015