



# DATA AND WEB DEVELOPMENT

[CC6012NI]

WEEK - 05

Database Concurrency II

# Earlier Lecture

Concurrency can  
be managed by  
adding LOCKS to  
the table



# Deadlock - Definition

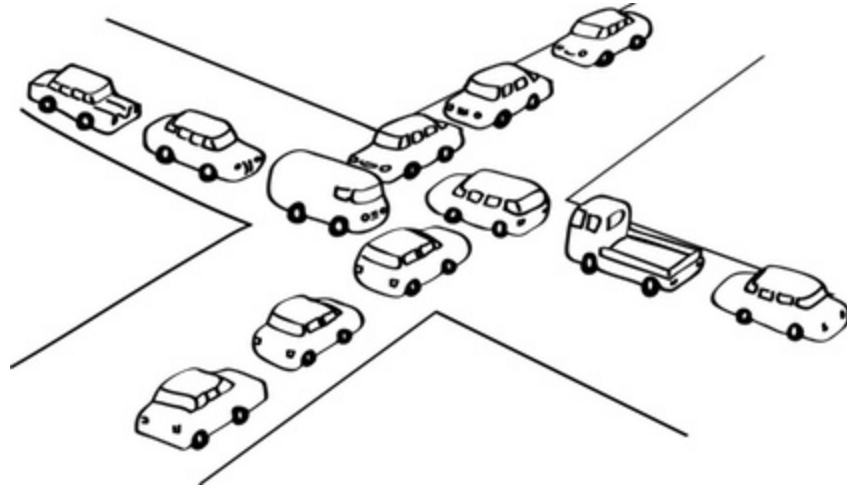
A system is in a state of deadlock if there exists a set of transactions such that every transaction in the set is waiting for another transaction in the set.



# Deadlock - Definition

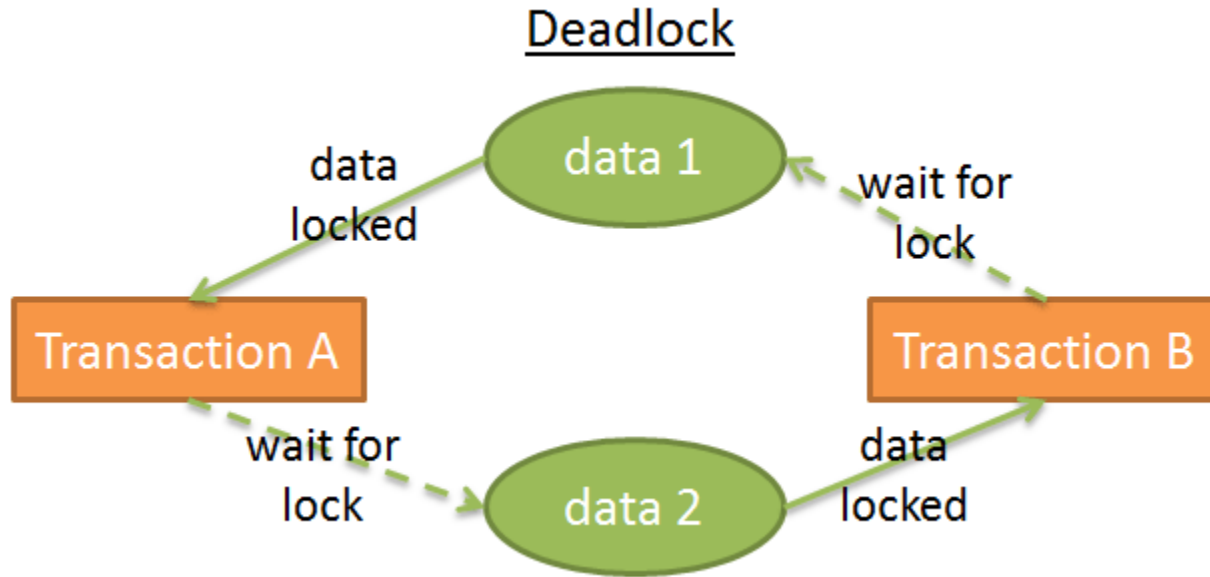
Deadlock is a situation in which two or more transactions are in a simultaneous **wait** state, each of them waiting for one of the others to release a lock before it can proceed



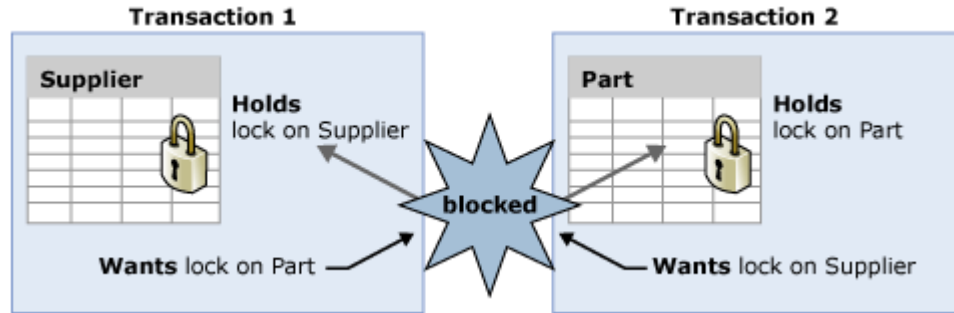


Database  
Concurrency – A  
Situation Likely to  
Occur

# Deadlock



# Deadlock



- They hold locks that may be required by other transactions
- DBMS must either PREVENT or DETECT and RESOLVE deadlock

# Deadlock Problem - Example

<i>Transaction A</i>	<i>Time</i>	<i>Transaction B</i>
.....		
acquire X lock on p1	t1	
.....	t2	acquire X lock on p2
request X lock on p2	t3	.....
wait	t4	request X lock on p1
wait	↓	wait
wait		wait

Here, neither of transactions can proceed !



# Handling Methods

---

## Deadlock Prevention

- Prevents the deadlock state

## Detection and Recovery

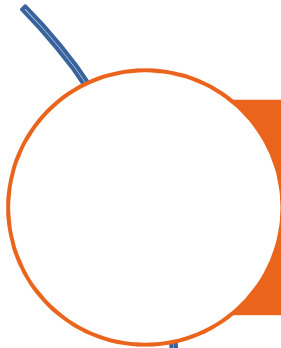
- Implements detection and recovery scheme

# Handling Methods

---

May result in  
Transaction Rollback  
Processing Overhead

# Deadlock Prevention Scheme



Each transaction locks **ALL** its required data items before it begins execution.



Either **ALL** data items needed are locked in one step (so the transaction can then proceed) or **NONE** are locked (to avoid locking only some of the data items needed)

# Best Use Situation

---

This scheme could be used if the probability of the system entering a deadlock state is relatively high (e.g. for long transactions needing many locks).

# Disadvantages

---

## Low Data Utilization

Some data items could be locked for a long time before they are used.

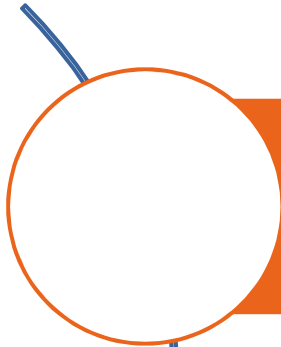
---

## Possible Starvation

A transaction which requires a number of data items may find itself in a 'indefinite' wait state while at least one of the data items is always locked by some other transactions.

---

# Deadlock Detection & Recovery



The state of the system is examined periodically to **detect** whether a deadlock has occurred in the system.




If it has, the system attempts to **recover** from the deadlock (often involving *rollback*).

# Deadlock Detection & Recovery Process


Maintain information about the **current allocation** of data items to different transactions – namely, locks that have been granted)



Maintain information about any **outstanding requests** for data items – namely, locks that have been requested but not granted yet.



Activate an algorithm (periodically or when required) which uses the above information to **determine** whether the system has entered a deadlock state.



If a deadlock state is entered (which may involve more than one deadlock), the system attempts to **recover** from the deadlock – namely, **breaking** the deadlock.

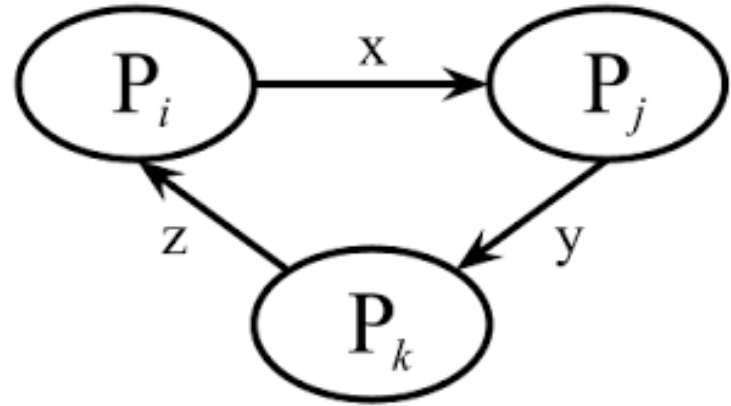
# Wait for Graph

- ❑ A **wait-for graph** in computer science is a directed graph used for deadlock detection in operating systems and relational database systems.
- ❑ In computer science, a system that allows concurrent operation of multiple processes and locking of resources and which does not provide mechanisms to avoid or prevent deadlock must support a mechanism to detect deadlocks and an algorithm for recovering from them.



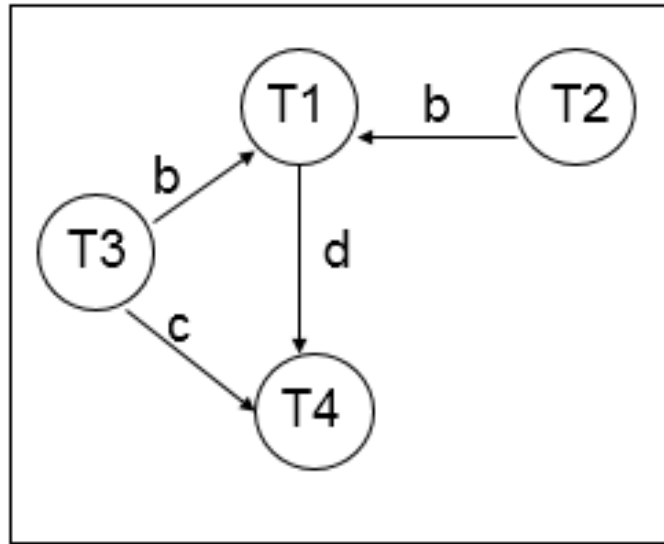
# Wait for Graph

**WFG** is a directed graph  $G=(N,E)$  which consists of a set of nodes  $N$  and a set of directed edges  $E$ .



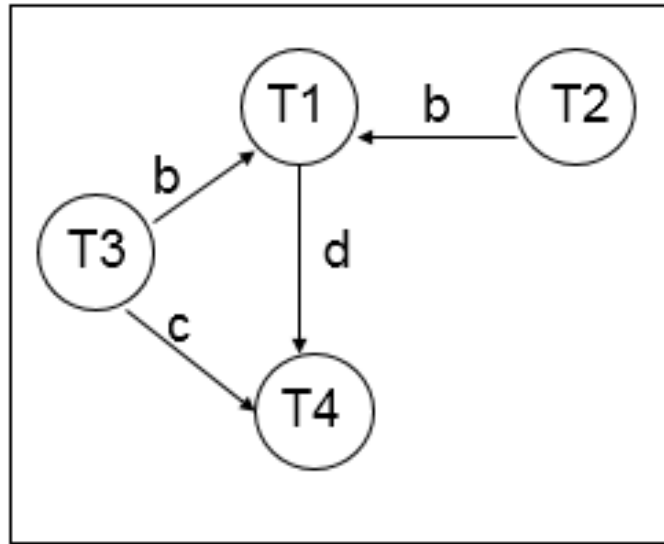
Deadlock exists if and only if WFG contains a CYCLE

# Wait for Graph - Example



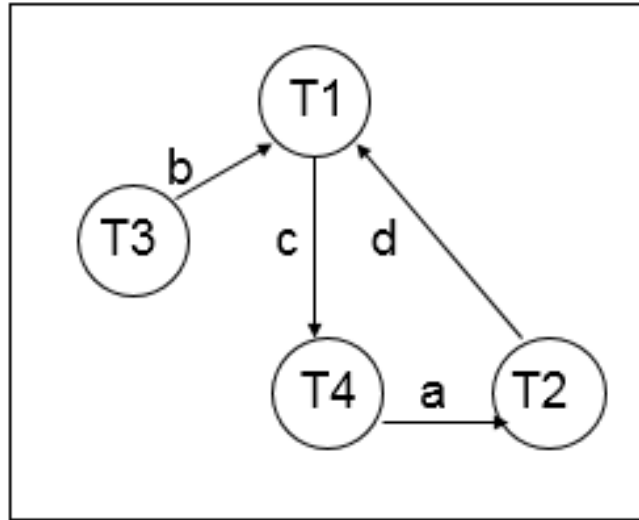
Does the above graph form a CYCLE ?

# Wait for Graph - Example



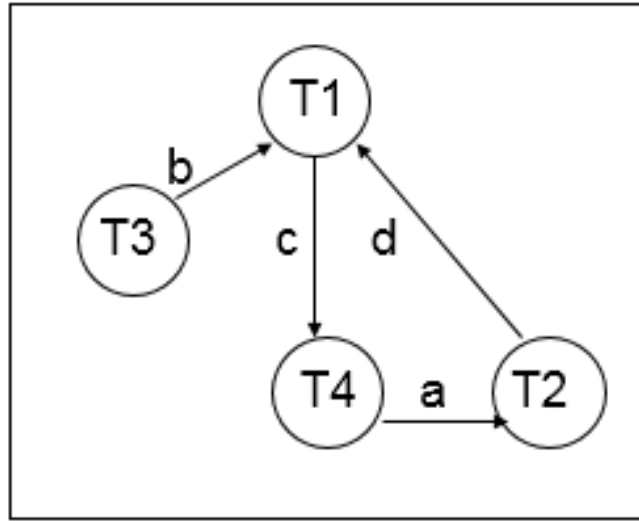
**NO ! Hence, no DEADLOCK.**

# Wait for Graph



Does the above graph form a CYCLE ?

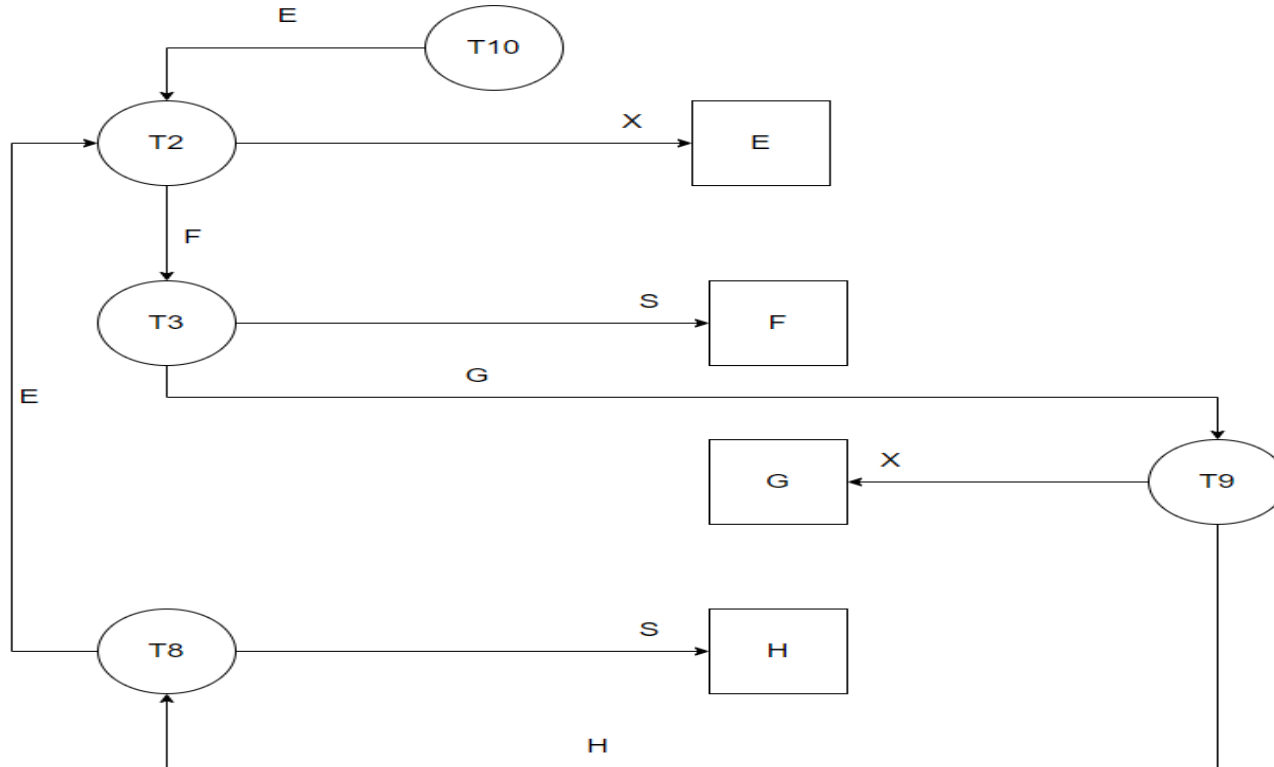
# Wait for Graph



YES, T1, T4 and T2 form a CYCLE!

Hence, DEADLOCK situation.

# Wait for Graph – How to Draw ?

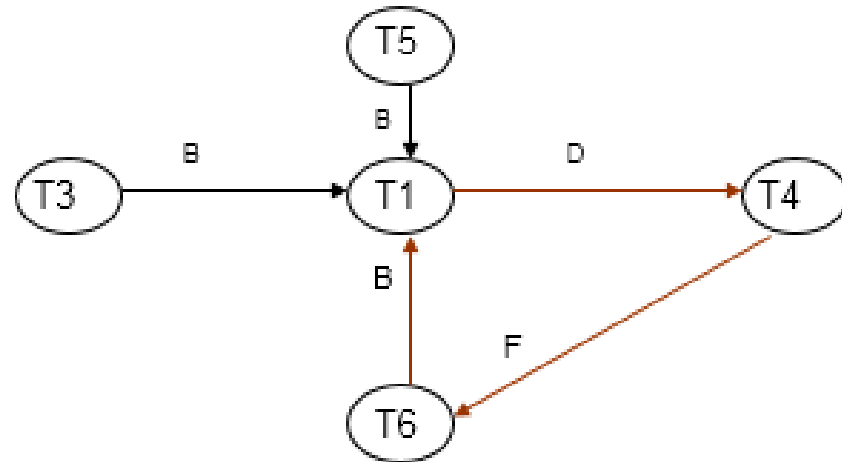


# Example Worked Out

time	transaction	event
t1	T1	FETCH B
t2	T2	FETCH C
t3	T1	UPDATE B
t4	T2	UPDATE C
t5	T2	COMMIT
t6	T3	FETCH B
t7	T4	FETCH D
t8	T5	FETCH B
t9	T4	UPDATE D
t10	T1	FETCH D
t11	T4	FETCH F
t12	T6	FETCH F
t13	T4	UPDATE F
t14	T6	FETCH B
t15	T7	FETCH A
t16	T7	ROLLBACK
t17		

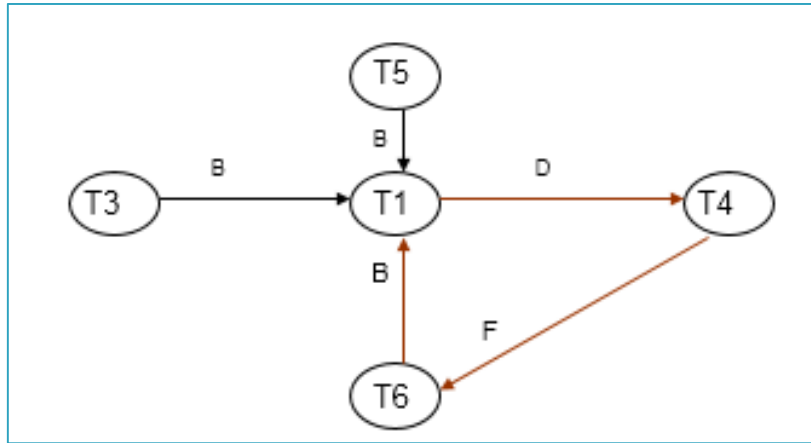
Provide a Wait-for-Graph to determine whether there is a deadlock at time t17, and if so, how the system could recover from the deadlock.

# Wait for Graph





# Wait for Graph



Cycle formed by T1, T6 and T4, resulting in a Deadlock

T1 waiting for T4 on D,  
T4 waiting for T6 on F,  
T6 waiting for T1 on B.

# Deadlock Recovery - Issues

## Issue of choosing a victim

Determine which transaction(s), among a set of deadlocked transactions, to be rolled back in order to break the deadlock. Criteria for choosing such a 'victim' transaction often depends on cost factors.

## Issue of rollback operation

Determine how far the chosen victim transaction should be rolled back (total rollback or partial rollback).

## Issue of starvation

Avoid situations where some transaction may always be chosen as the victim due to cost factors based selection, hence never completing its job.

# Summary

---

- Concurrency Situations
- THREE types of concurrency problems
- Locking: Might result in Deadlock
- Deadlock Handling
- Wait for Graph for Deadlock Detection and Recovery

# Thank You

