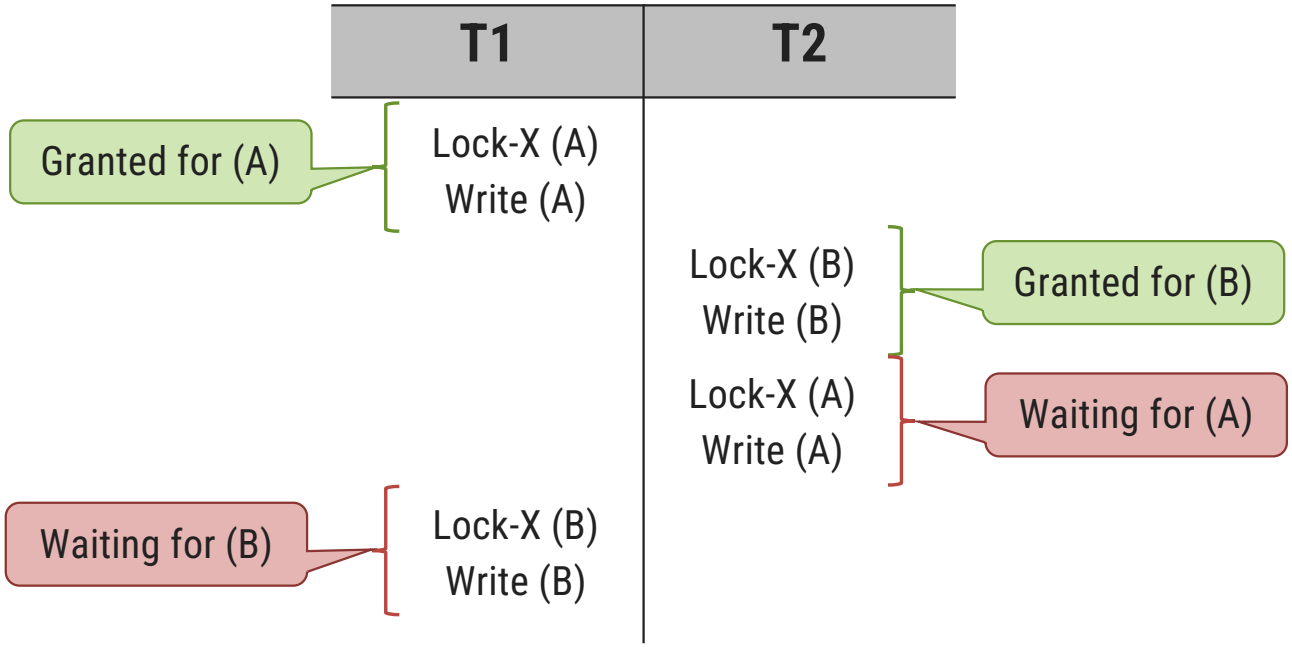
Deadlock

What is deadlock?

- Consider the following two transactions:



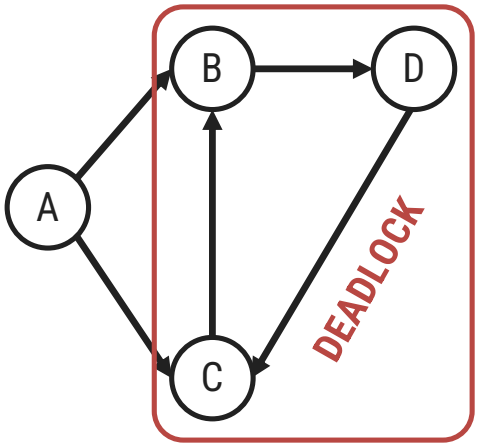
- A deadlock is a **situation in which two or more transactions are waiting for one another to give up locks**.

Deadlock detection

- A simple way to detect deadlock is with the help of **wait-for graph**.
- One **node is created** in the wait-for graph for **each transaction that is currently executing**.
- Whenever a **transaction Ti is waiting to lock an item X that is currently locked by a transaction Tj, a directed edge from Ti to Tj (Ti→Tj) is created in the wait-for graph**.
- When **Tj releases the lock(s) on the items that Ti was waiting for**, the **directed edge is dropped** from the wait-for graph.
- We have a state of **deadlock if and only if the wait-for graph has a cycle**.
- Then **each transaction involved in the cycle is said to be deadlocked**.

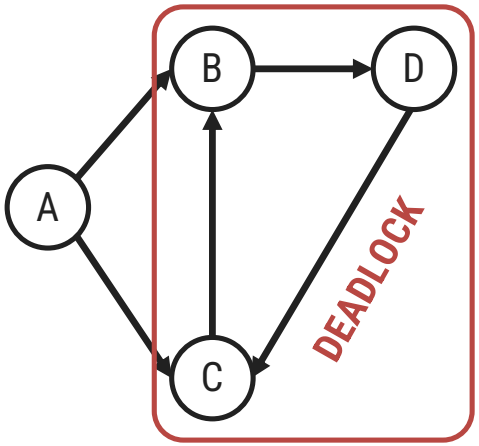
Deadlock detection

- Transaction **A is waiting for** transactions **B and C**.
- Transactions **C is waiting** for transaction **B**.
- Transaction **B is waiting** for transaction **D**.
- This wait-for graph has **no cycle**, so there is **no deadlock state**.
- Suppose now that transaction **D is requesting an item held by C**. Then the **edge D → C is added to the wait-for graph**.
- Now this **graph contains the cycle**.
- B → D → C → B
- It means that **transactions B, D and C are all deadlocked**.



Deadlock recovery

- When a deadlock is detected, the system must recover from the deadlock.
- The most common **solution is to roll back one or more transactions to break the deadlock**.
- Choosing which transaction to abort is known as **victim selection**.
- In this wait-for graph transactions B, D and C are deadlocked.
- In order to remove deadlock one of the transaction out of these three (B, D, C) transactions must be roll backed.
- We should **rollback those transactions that will incur the minimum cost**.
- When a deadlock is detected, the choice of which transaction to abort can be made using following criteria:
  - The transaction which have the fewest locks
  - The transaction that has done the least work
  - The transaction that is farthest from completion



Deadlock prevention

- A protocols **ensure that the system will never enter into a deadlock state**.
- Some prevention strategies :
  - Require that **each transaction locks all its data items before it begins execution** (pre-declaration).
  - Impose **partial ordering on all data items and a transaction can lock data items only in the specified order**.

- Drawbacks:
  - It is difficult to predict what data items are required.
  - Data item utilization will be low.
  - Ordering of the data item may be difficult and time consuming.

Deadlock prevention

- Following schemes use transaction timestamps for the sake of deadlock prevention alone.

1. Wait-die scheme — non-preemptive
   - If an **older transaction is requesting a resource** which is held by younger transaction, then **older transaction is allowed to wait** for it till it is available.
   - If an **younger transaction is requesting a resource** which is held by older transaction, then **younger transaction is killed**.
2. Wound-wait scheme — preemptive
   - If an **older transaction is requesting a resource** which is held by younger transaction, then **older transaction forces younger transaction to kill** the transaction and release the resource.
   - If an **younger transaction is requesting a resource** which is held by older transaction, then **younger transaction is allowed to wait** till older transaction will releases it.

	Wait-die	Wound-wait
O needs a resource held by Y	O waits	Y dies
Y needs a resource held by O	Y dies	Y waits

Deadlock prevention

- Following schemes use transaction timestamps for the sake of deadlock prevention alone.

3. Timeout-Based Schemes
   - A **transaction waits for a lock only for a specified amount of time**. **After that, the wait times out and the transaction is rolled back**. So deadlocks never occur.
   - **Simple to implement; but difficult to determine good value of the timeout interval**.