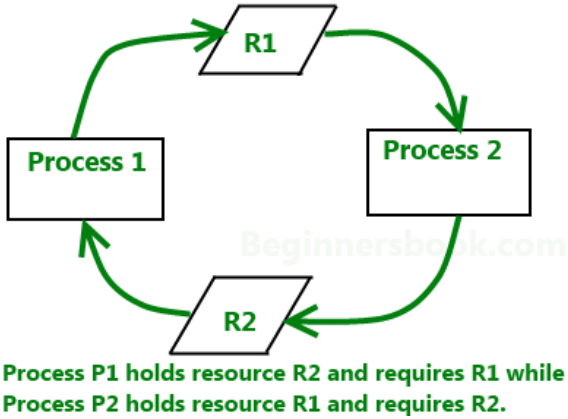
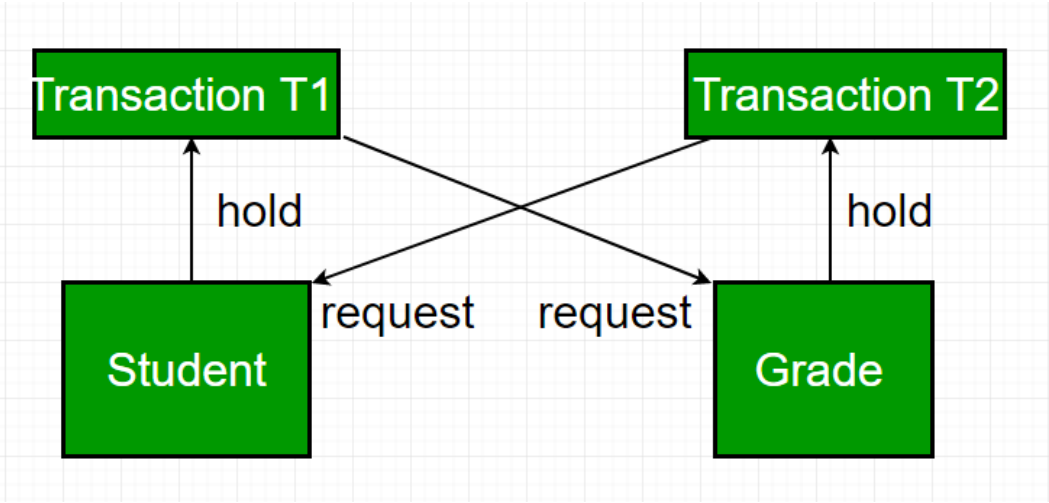
Dead Locks

Dead Locks

A **deadlock** is a condition wherein two or more tasks are waiting for each other in order to be finished but none of the task is willing to give up the resources that other task needs.





In this situation no task ever gets finished and is in waiting state forever.

Dead Locks : Coffman's Conditions of Dead Lock

Coffman stated **four conditions** for a deadlock occurrence.

A deadlock may occur if all the following conditions holds true.

1. **Mutual exclusion**
2. **Hold and wait condition**
3. **No preemption condition**
4. **Circular wait condition**

Dead Locks : Coffman's Conditions of Dead Lock

Mutual exclusion condition:
At least one data item is mutually exclusive(non-sharable)

Hold and wait condition:
A transaction that is holding a data item and waiting for other resource, held by other transaction

No preemption condition: A data item cannot be forcibly taken from a transaction.

Circular wait condition:
A condition where one process is waiting for a resource that is being held by second process and second process is waiting for third process ….so on and the last process is waiting for the first process. Thus making a circular chain of waiting.

Dead Lock : Prevention

To prevent any **deadlock** situation in the system, the **DBMS** inspects all the operations, where transactions are about to execute. ... If it finds that a **deadlock** situation might occur, then that transaction is never allowed to be executed.

Dead Locks Prevention

Preventing one or more of four Coffman conditions could prevent the deadlock

Removing mutual exclusion:

1. All resources must be sharable

Impractical

Dead Locks Prevention

Preventing one or more of four Coffman conditions could prevent the deadlock

Removing hold and wait condition:

1. if the process acquires all the resources that are needed before starting out.

2. Enforce a rule of requesting resource when there are none in held by the process

Dead Locks Prevention

Preventing one or more of four Coffman conditions could prevent the deadlock

Preemption of resources:

1. Preemption of resources from a process can result in rollback and thus this needs to be avoided in order to maintain the consistency and stability of the system.

Impractical

Dead Lock: Prevention

Preventing one or more of four Coffman conditions could prevent the deadlock

Avoid circular wait condition:

1. This can be avoided if process can hold the resources in increasing order of precedence

2. Force one resource per process rule – A process can request for a resource once it releases the resource currently being held by it

Impractical

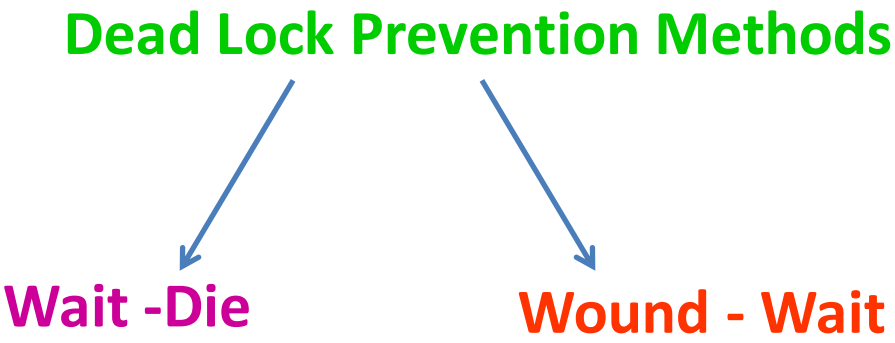
Dead Lock: Prevention

- **Deadlock prevention method is suitable for a large databases**

- **If the resources are allocated in such a way that deadlock never occurs, then the deadlock can be prevented**

- **The Database management system analyzes the operations of the transaction whether they can create a deadlock situation or not. If they do, then the DBMS never allowed that transaction to be executed.**

Dead Lock: Prevention



Dead Lock: Prevention

Wait-Die

If a transaction requests for a resource which is already held with another transaction then the DBMS simply checks the timestamp of both transactions.

It allows the older transaction to wait until the resource is available for execution.

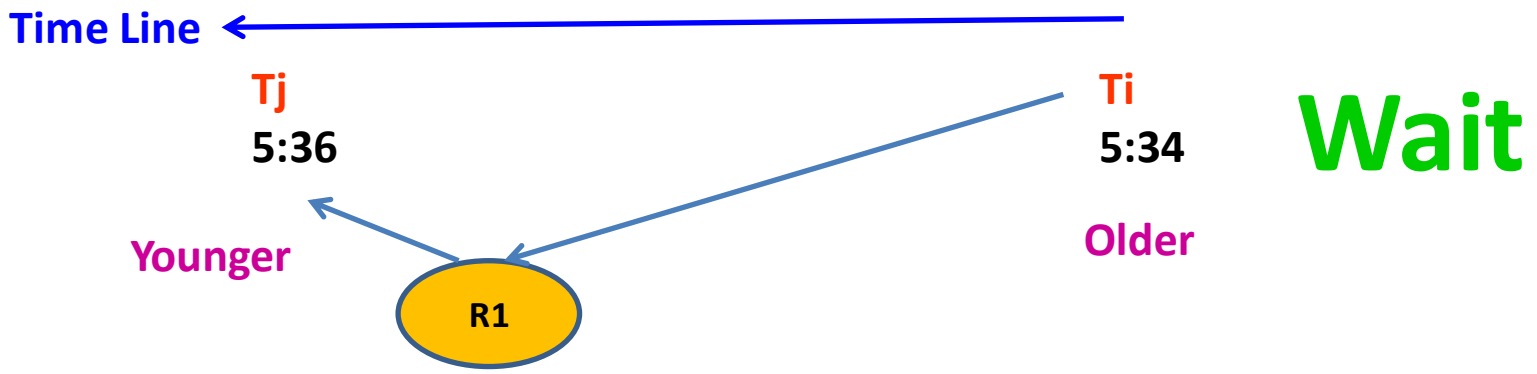
Example :
Assume there are two transactions Ti and Tj
let TS(T) is a timestamp of any transaction T

If T2 holds a lock by some other transaction and T1 is requesting for resources held by T2 then the following actions are performed by DBMS-

Dead Lock: Prevention

Wait-Die

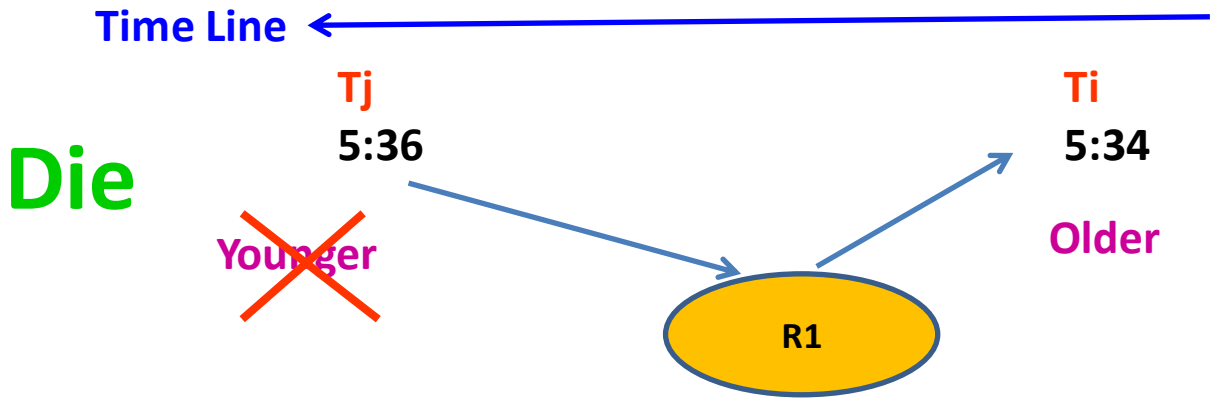
Check if TS(Ti) < TS(Tj) - If Ti is the older transaction and Tj has held some resource, then Ti is allowed to wait until the data-item is available for execution. That means if the older transaction is waiting for a resource which is locked by the younger transaction, then the older transaction is allowed to wait for resource until it is available.



Dead Lock: Prevention

Wait-Die

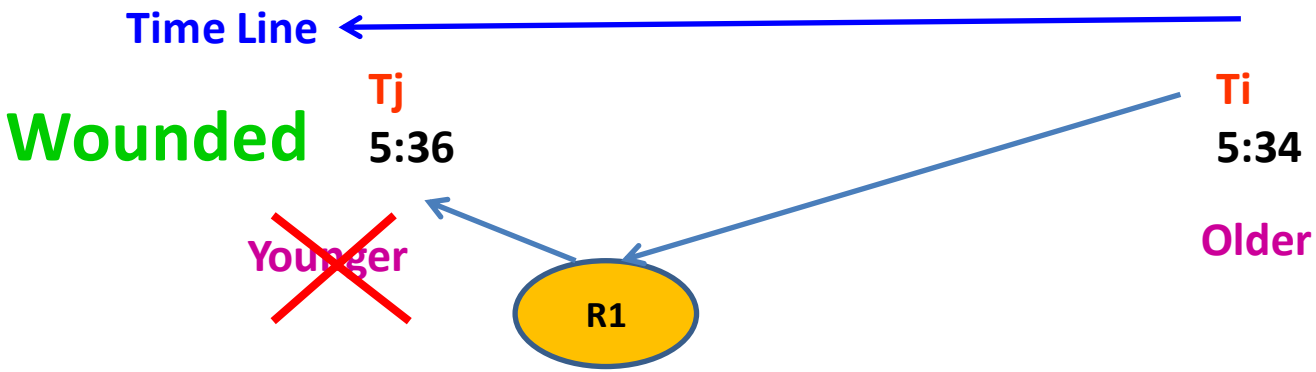
Check if TS(Ti) < TS(Tj) - If Ti is older transaction and has held some resource and if Tj is waiting for it, then Tj is killed and restarted later with the random delay but with the same timestamp.



Dead Lock: Prevention

Wound Wait

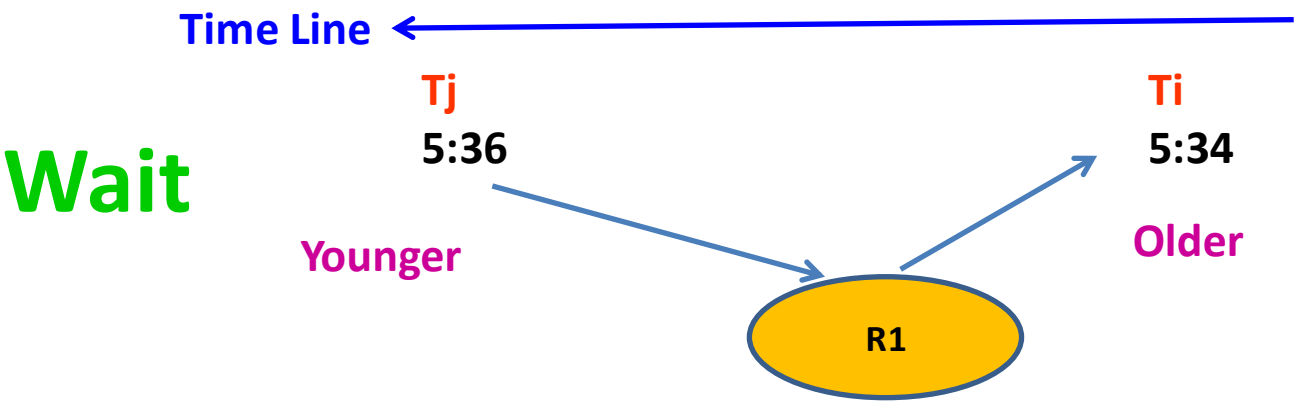
if the older transaction requests for a resource which is held by the younger transaction, then older transaction forces younger one to kill the transaction and release the resource. After the minute delay, the younger transaction is restarted but with the same timestamp



Dead Lock: Prevention

Wound Wait

If the older transaction has held a resource which is requested by the Younger transaction, then the younger transaction is asked to wait until older releases it.



Dead Lock: Prevention

	Wait/Die	Wound/Wait
Older transaction needs a resource held by younger transaction	Older transaction waits	Younger transaction dies
Younger transaction needs a resource held by older transaction	Younger transaction dies	Younger transaction waits

Dead Lock: Detection

In a database, when a transaction waits indefinitely to obtain a lock, then the DBMS should detect whether the transaction is involved in a deadlock or not

The lock manager maintains a Wait for the graph to detect the deadlock cycle in the database

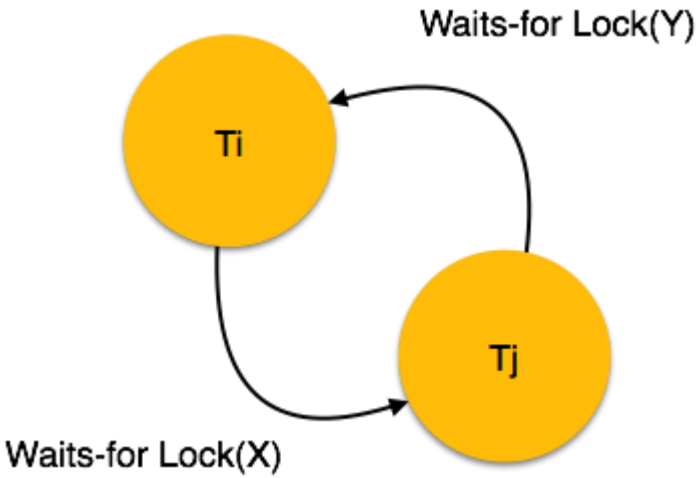
Dead Lock: Detection

Wait for Graph

Used for deadlock detection

A graph is created based on the transaction and their lock

If the created graph has a cycle or closed loop, then there is a deadlock.



Dead Lock: Avoidance

1. Check for deadlock at each step; If dead lock then abort or rollback transaction. disadvantage: waste of time and resource
2. Check for deadlock in advance of a transaction.

This method is suitable only for the smaller database.

For the larger database, deadlock prevention method can be used.