Dead Lock

def: Infinite wait condition is refered as "dead-lock"

→ if deadlock occurs in an application, then the only solution is "terminate the application"

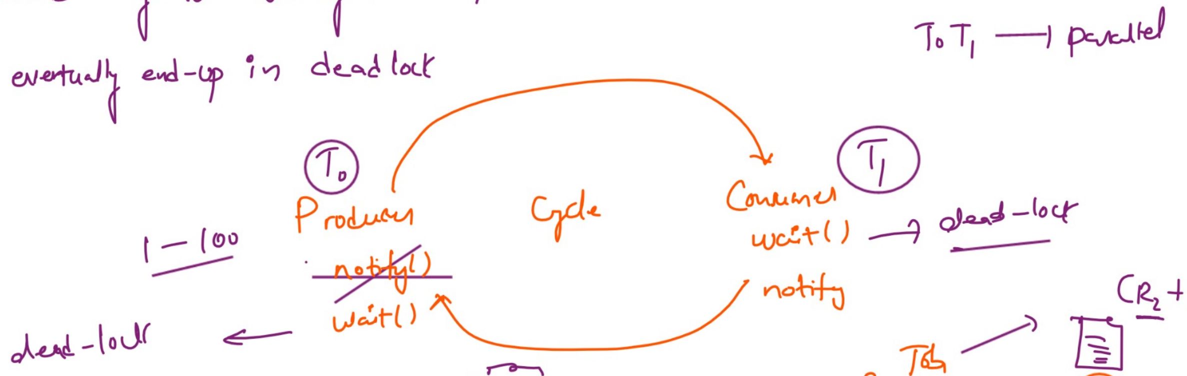
→ during the design of application, we should see that dead-lock condition will never occur.

→ the main reason for deadlock is if cyclic dependency exist b/w the threads (i.e one thread depends on the another thread)

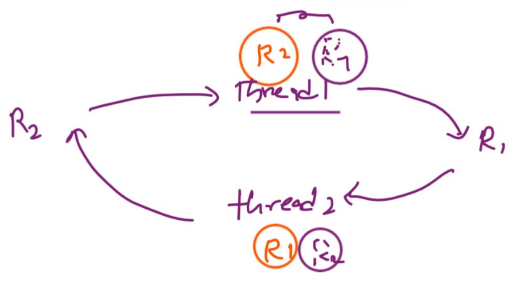
→ during the design of application, we should see that the threads are not dependent on cyclic manner [avoid dead-lock]

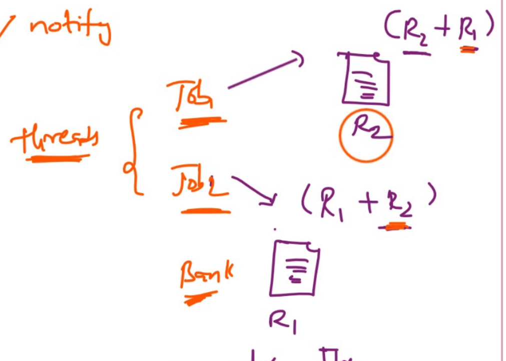
example case: producer-consumer

→ In case of PC problem, if we remove the "notify()" call, the threads would go to waiting state & no one will there to notify back. which will eventually end-up in dead lock



Case 2





→ thread 1 which is holding resource R2 & waiting for the resource R1

→ Thread2 is holding resource R1 & waiting for the Resource R2

→ Both the threads would release the resources, only when there task is completed, since all the necessary resources are not available for the task to be completed

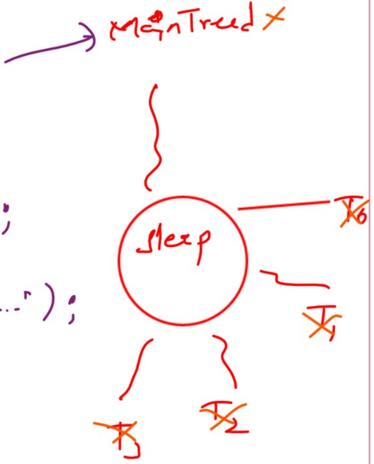
→ Both the threads would wait for infinite time which results in dead-lock

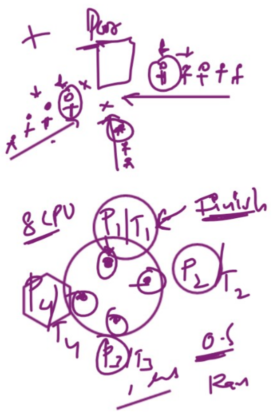
Case 3 (simple code to illustrate dead-lock)

```
public class Launch {

    p.s.v main (String args[ ]) throws InterruptedException {

        Thread mainThread = Thread.currentThread();
        S.O.P("Going to waiting state ...");
        mainThread.join();
        S.O.P("came out of waiting state ...");
    }
}
```





Race Condition

$\rightarrow T_0\ T_1\ T_0\ T_1\ T_0\ T_1$

$\rightarrow T_0\ T_1\ T_1\ T_0\ T_0 T_1\ T_0\ T_0 T_1\ T_0\ T_1$

→ In multithreading environment the threads created will always compete for the CPU time

→ In the process of acquiring CPU time, there cases where the order in which the threads execution change

→ If an application is designed to generate a specific output, depending upon the sequence in which the threads execute, then in such cases due to racing b/w the threads in application the sequence in which they have to change, this scenario in which the sequence of execution of thread would change in-turn alters the output which we can refer it as race-condition

→ we can overcome the race-condition through

1. sleep() → timer
2. wait() → notify
3. join()
4. synchronized

← { $T_0\ T_1\ T_2\ T_3\ T_4\ T_5$
$T_6\ T_7\ T_8\ T_9\ T_{10}$ }