Deadlock prevention

It is a set of methods for ensuring that at least one of the necessary conditions can not hold. These methods prevent deadlocks by constraining how requests for sources can be made.

- The **mutual-exclusion** condition must hold for non –sharable resource, sharable resources on the other hand do not require mutually exclusive access.

- To ensure that **hold-and-wait** condition never occurs in the system must guarantee that whenever a process request a resource it does not hold any other resources. This can be implemented by :

1- Allocate all resources to process before its execution .

2- A process request a resource only allowed when it has none. The problems with this the low utilization of resource usage and starvation.

- **No –preemption**

  If a process that is holding some resources request another resources that can not allocated to it then all resources currently being held are preempted.
- **Circular wait**

Let R ={ R1 , R2 , R3 , ….. , R_n } be the set of resource types we can assign to each type a number which allow us to compare two resources . If we define a one – to – one function $F : R \rightarrow N$ where N is the set of numbers.
Example :
F(T/Drive)=1
F(Disk/Drive)=5
F(Printer)=12
Each process can request resources only in an increasing order of enumeration. That is process initially request any number of instances of R_i after that the process can request instances of resource type R_j if and only if

$F(R_j) \succ F(R_i)$

Deadlock Avoidance

For avoiding deadlock is to require additional information about how resources are to be requested.
For example in C/S with one tape and one printer we might be told that process P will request first M/T and later L/P before releasing both resources. Process Q on the other hand will request first the printer and then the M/T. With this knowledge of the complete sequence of request and releases for each process we can decide for each request whether or not the process should wait.
Each request requires that then consider the following :

1- The resources currently available .
2- The resources currently allocated to each process.
3- The future requests and releases of each process.

The above information used to decide whether the current request can be satisfied or must wait to avoid a possible future deadlock.
The various algorithms differ in the amount and type of information required. The simplest and most useful model requires that each process declare the maximum number of each type that it may need. A deadlock avoidance algorithm dynamically examines the resource-allocation state to ensure that these can never be a circular – wait condition.
The resource allocation state is defined by number of available and allocated resources and maximum demands of the processes.

- **Safe state**

A state is safe if the system can allocate resources to each process (up to maximum) in some order and still avoid a deadlock.
A safe state is not a deadlock state , and a deadlock state is an unsafe state , but not all unsafe states are deadlock. An unsafe state may lead to a deadlock.

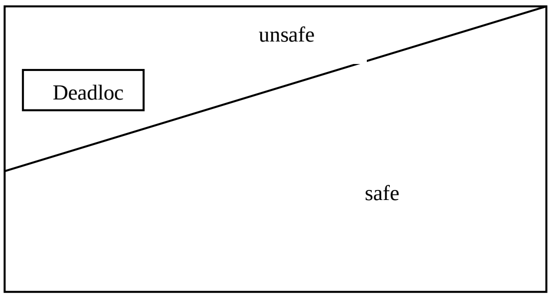

Fig : unsafe &deadlock state space

Example :- to illustrate consider a system with 12 M/T and 3 process :
P0 , p1 and p2 . The maximum needs and current needs for each process as indicated below

(Allocated)

	Maximum needs	Current needs	available
P0	10	5	3
P1	4	2	
P2	9	2	

At time the system is in a safe state. The sequence [p0 , p1 , p2]
Suppose that at time t1 process p2 requests and is allocated 1 move tape drive . the system is no longer in safe state .At this point only process p1 can be allocated all

its tape drives . When it returns the system have only if available and this number not satisfy the request for p0 or p2 therefore

$$\begin{tikzcd} P0 \arrow[r, "Wait"] & p2 \arrow[r, "Wait"] & P0 \end{tikzcd}$$

If we had made p2 wait until either of the order process had finished and released its resources then we could have avoided the deadlock situation.
There are many deadlock avoidance algorithms , some of these are :

a- RAG Algorithm

If we have a RAG system with only one instance of each resource type . In addition to the request and assignment edges we introduce a new type of edge called a claim edge $P_i \rightarrow R_j$ indicates that process P_i may request resource R_j at some time in the future. It is as a request edge in direction but is represented by a dashed- line when process p_i request R_j the claim edge $P_i \rightarrow R_j$ is converted to a request edge . When a resource R_j is released by p_i the assignment edge $R_j \rightarrow P_i$ is reconverted to a claim edge $P_i \rightarrow R_j$.
To illustrate this algorithm consider the RAG in fig below suppose that p_2 request r_2 . Although r_2 is currently free we can not allocate it to p_2 since this action will create a cycle in the graph figure . A cycle indicates that the system is in an unsafe state . If P_1 then requests r_2 a deadlock will occur.

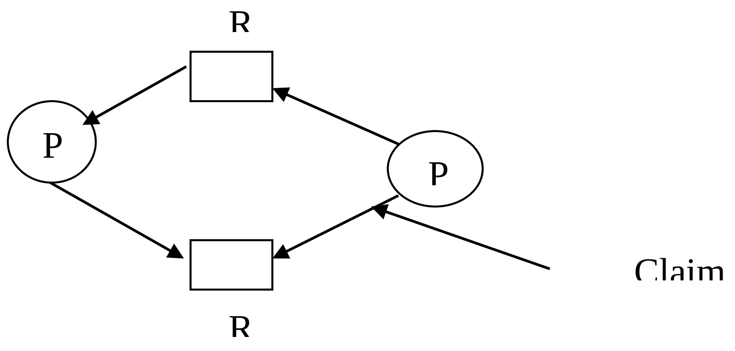

Fig : RAG for deadlock avoidance

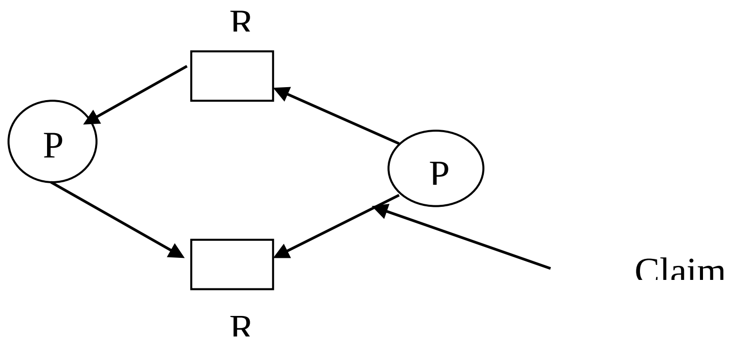

Fig : An unsafe state in a RAG