ECE437/CS481

M05B: DEADLOCKS

CHAPTER 7

Xiang Sun

The University of New Mexico

Deadlock Handling

❑ Strategies to solve the deadlock problem

- **Prevention**: design a strategy that negates one of four deadlock necessary conditions.
- **Avoidance**: dynamic strategy of processing resource requests to ensure that the system is always in a safe state.
  - ✓ safe state --- if there exists at least one sequence of execution for all processes such that all of them can run to completion
  - ✓ unsafe state ≠ deadlock (sometimes it's unsafe, but not deadlock)
- **Detection and recovery**: strategy for detecting the presence of deadlock and eliminating deadlocks.

Deadlock Handling

- Deadlock Prevention
- Deadlock Avoidance
- Deadlock Detection and recovery

Deadlock Prevention

- Prevent the mutual exclusion condition from coming true
  - Recall mutual exclusion: at least one shared resource is held in a non-sharable mode. Other requesting processes must wait until the resource is released
  - This is opposite of our original goal, which was to provide mutual exclusion.
  - Also, many resources are non-sharable and must be accessed in a mutually exclusive way
    - ✓ Example: a printer should print a file X to completion before printing a file Y. A printer should not print half of file X, and then print the first half of file Y on the same paper.
  - Thus, it is unrealistic to prevent mutual exclusion

Deadlock Prevention

- Prevent the hold and wait condition from coming true
  - Prevent a process from holding resources and requesting others.
  - Solution I (holding while not waiting): reserve all needed resources at process creation.
  - Solution II (waiting while not holding): release all the holding resources before requesting new resources.
  - Example: a process 1) reads a file from the DVD drive, 2) writes the file to disk, 3) reads another file from the disk, 4) sends the file to the printer.
    - ✓ Solution I: reserve the DVD drive, disk, and printer once the process is created.
    - ✓ Solution II: break the task down into wholly contained nondependent pieces:
      1. obtain the DVD and disk together for the file transfer, then release both together
      2. next obtain the disk and printer together for the printing operation, then release both together

Deadlock Prevention

- Disadvantages of preventing hold-and-wait solutions
  - Don't know in advance all resources needed.
  - Poor resource utilization.
  - Possible starvation.
    - ✓ a process that needs several popular resources simultaneously may have to wait for a very long time.

Deadlock Prevention

- Prevent the "No Preemption" condition from coming true
  - Allow resources to be preempted
  - Various policies are made for preemption, e.g.,
    - ✓ If process X requests a resource held by process Y, then preempts the resource from process Y, but only if Y is waiting on another resource. Otherwise, X must wait.
  - Disadvantages
    - ✓ Preemption cannot be applied to all resources, e.g., printers should not be preempted while in the middle of printing, disks should not be preempted while in the middle of writing a block of data
    - ✓ May cause unexpected process behavior, as application developers might not know in advance which policy is in use.

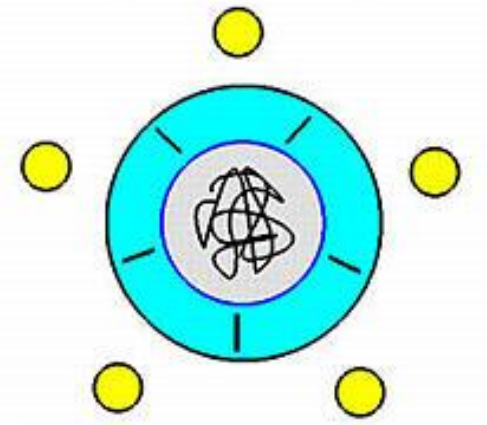
Deadlock Prevention

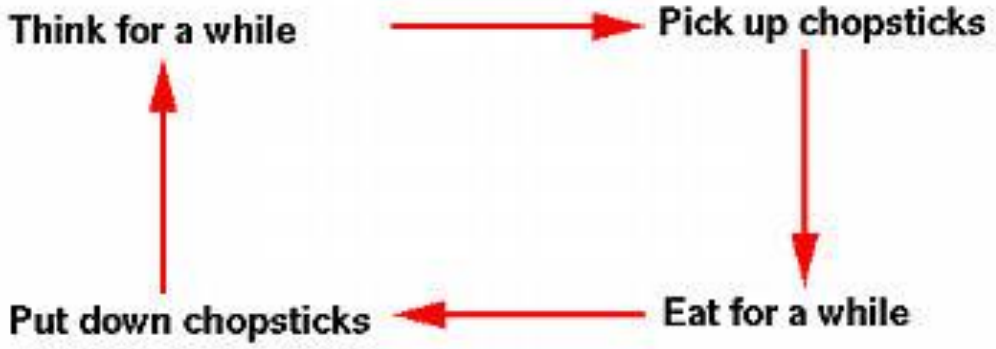
- Prevent the circular wait condition from coming true
  - Basic idea: impose a total ordering of all resource instances and require each process to request resources in increasing order
  - That is:
    - ✓ Order all resource instances into a list: $R_1, R_2, \ldots, R_m$
    - ✓ Impose the rule that a process holding R_i can only request R_j, where $j > i$.
    - ✓ If a process holds R_i and requests R_k (where $k < i$), then the process must release R_i, acquire R_k, and then request R_i.

Deadlock Prevention

❑ Ordering of resource types to solve deadlock in the Dining Philosophers problem

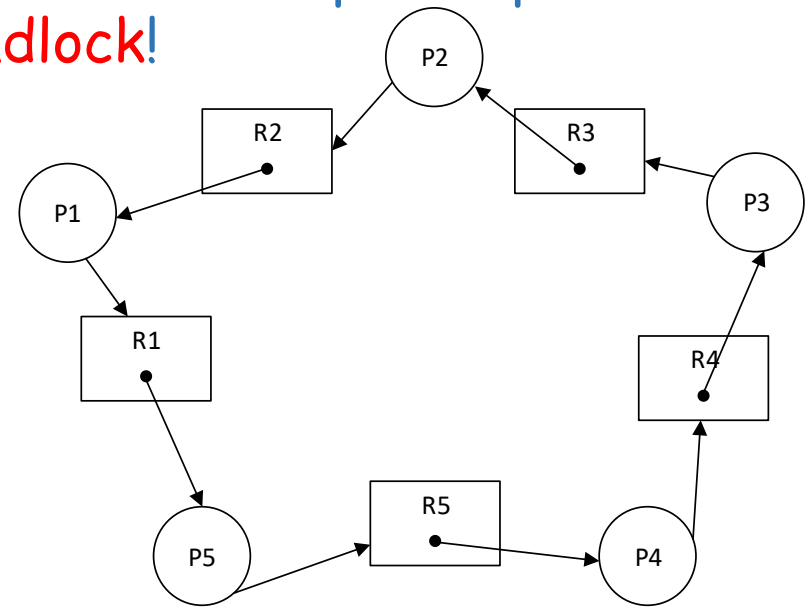
- Five philosophers sitting around a table. They spend their lives just thinking and eating.
- There are five chopsticks available. When a philosopher get hungry, he tries to pick up two chopsticks that are closest to him.
- A philosopher can eat only if he picks up two chopsticks.
- Each chopstick can only be used by at most one philosopher at a time (mutual exclusion).
- If a philosopher holds one chopstick but the other chopstick is currently used by another one, the philosopher will hold the chopstick and wait the other chopstick (hold & wait).
- If a chopstick has been picked up by a philosopher, other philosophers cannot rob this chopstick (non-preemption).

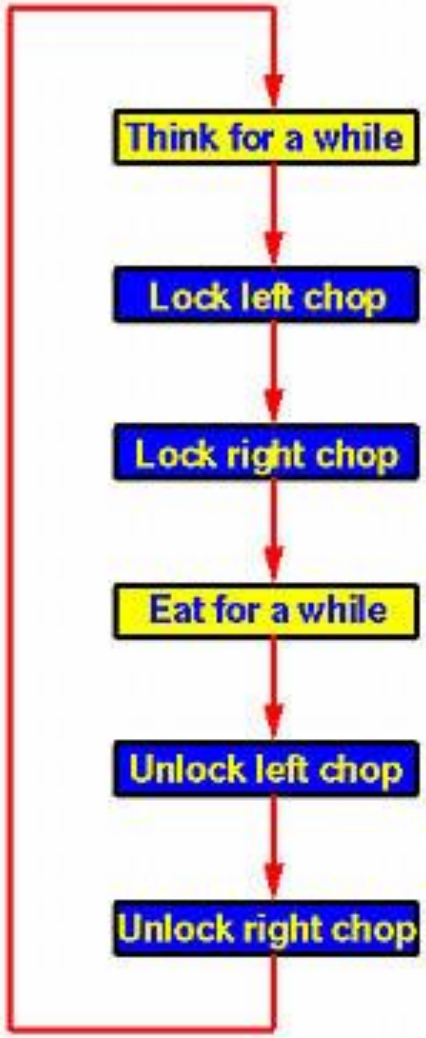




Deadlock Prevention

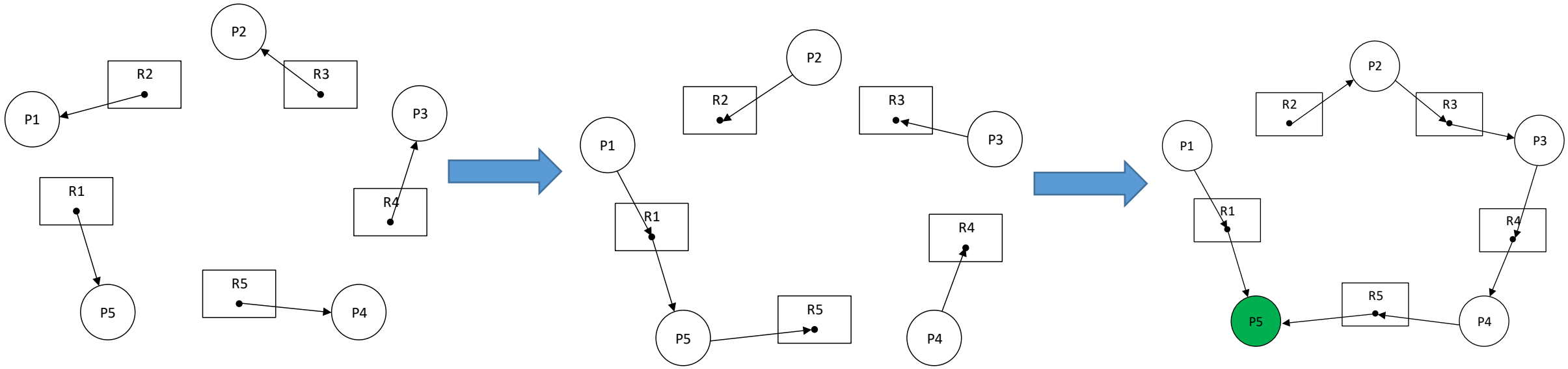
- Ordering of resource types to solve deadlock in the Dining Philosophers problem
  - A philosopher first locks his left chopstick, and then his right chopstick.
  - If the acquisitions of both locks are successful, this philosopher now owns two locks (hence two chopsticks), and can eat. After finishes eating, this philosopher releases both locks/chopsticks and thinks.
  - What if five philosophers sit down and pick up their left chopsticks simultaneously?---deadlock!





Deadlock Prevention

- Ordering of resource types to solve deadlock in the Dining Philosophers problem
  - R1 , R2 , R3 , R4 , R5



- Deadlock Prevention
- Deadlock Avoidance
- Deadlock Detection and recovery

Deadlock Avoidance

- Each process provides the OS with the maximum number of instances for each resource type that it may need.

- The deadlock-avoidance algorithm dynamically examines the resource allocation state to ensure that the system is in safe state (there can never be a circular-wait condition).

  - A resource allocation state is defined by
    - ✓ # of available instances of a resource
    - ✓ # of allocated instances of a resource to each process
    - ✓ maximum # of required instances of a resource for each process

- Weakness of deadlock avoidance: need a priori info-- maximum amount of resource of each type

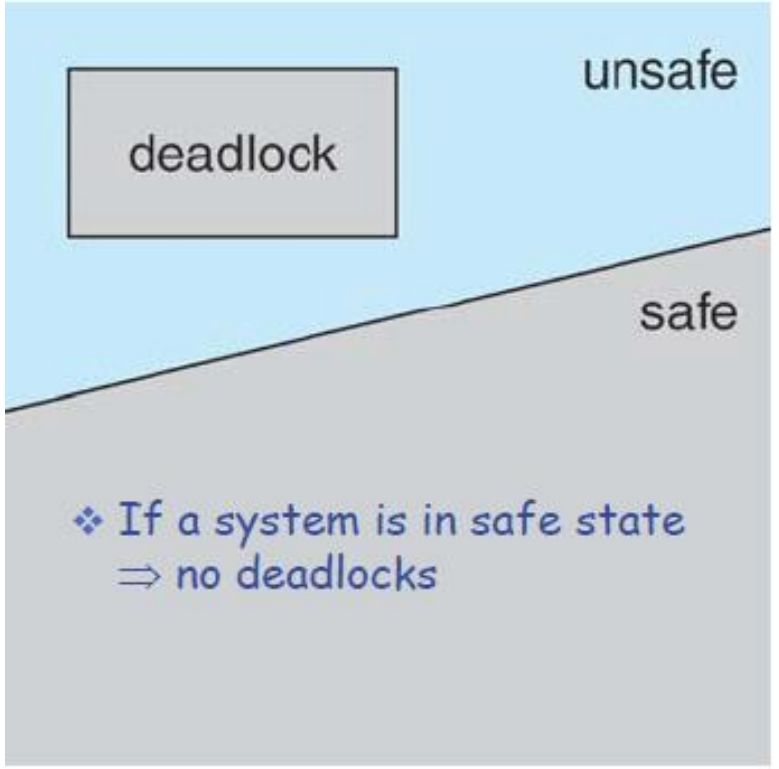
Deadlock Avoidance

- Definition of safe state: system is safe if there exists a safe sequence of processes <P_1, ..., P_n> for the current resource allocation state
  - A sequence of processes is safe---for each process P_i in the sequence, the resource request of P_i can be satisfied by using the resources currently allocated to P_i and the resources of all the processes, who will finish prior to P_i and free up their resources.

- That is
  - If P_i resource needs are not immediately available, then P_i can wait until all the previous processes have finished.
  - When all the previous processes are finished, P_i can obtain all the released resources, being executed and terminated.
  - When P_i terminates, P_{i+1} can use its resources.

Deadlock Avoidance

- A safe state provides a safe "escape" sequence.
- A deadlocked state is unsafe.
- An unsafe state is not necessarily deadlocked.



Deadlock Avoidance

- Example 1:
  - 12 instances of a resource
  - At time t_0, P0 holds 5, P1 holds 2, P2 holds 2
  - Available = 3 free instances

processes	max needs	allocated
P0	10	5
P1	4	2
P2	9	2

Deadlock Avoidance

- Example 1:

processes	max needs	allocated
P0	10	5
P1	4	2
P2	9	2

  - Is the system in a safe state? Does a safe sequence exist?

  - Yes. <P1, P0, P2> is a safe sequence.
    - ✓ P1 requests its maximum (currently has 2, so needs 2 more) and holds 4, then there is only 1 free resource
    - ✓ Then, P1 releases all of its held resources, so that there are 5 free resource instances
    - ✓ Next, suppose P0 requests its maximum (currently has 5, so needs 5 more) and holds 10, so that there are 0 free resource instance
    - ✓ Then, P0 releases all of its held resources, so that there are 10 free resource instances
    - ✓ Next, P2 requests its max of 9, leaving 3 free, and then releases them all.

  - Thus, the sequence <P1, P0, P2> is safe, i.e., the system is able to allocate maximum amount of requested resource to each process.

Deadlock Avoidance

- Example 2:
  - At time t_0, P2 is given 1 more instance of the resource (P2 holds 3 instances of the resource), then
  - Available = 2 free instances

processes	max needs	allocated
P0	10	5
P1	4	2
P2	9	3

  - Is the system in a safe state?

Deadlock Avoidance

processes	max needs	allocated
P0	10	5
P1	4	2
P2	9	3

- Example 2:
  - P1 can request its maximum (currently holds 2, and needs 2 more) of 4, so that there are 0 free resource instance.
  - P1 releases all its held resources, so that available = 4 free resource instance.
  - Neither P0 nor P2 can request their maximum resources (P0 needs 5, P2 needs 6, and there are only 4 free resource instance).

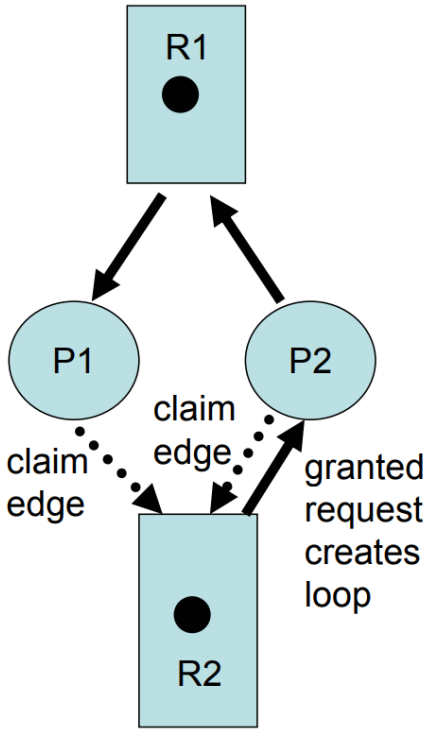
    ---Both would have to wait, so there could be deadlock

  - The system is unsafe.

Deadlock Avoidance

❑ Using a Resource Allocation Graph for deadlock avoidance

- ➢ Each process identifies possible future claims, drawn as claim edges (dotted lines).
- ➢ If converting a claim edge to a request edge creates a loop, then don't grant the request.
- ➢ Example: Granting P2's request for R2 may lead to a deadlock, so don't grant it.
- ➢ High complexity when there are more than 1 instances for each resource type.



Deadlock Avoidance

Banker's Algorithm:

- Denote
  - ✓ i=1…n as index of processes.
  - ✓ j=1,…,m as the index of resource types.
  - ✓ Available[j] as the current available resource instances for resource type j. For example, Available[j]=5 indicates there are 5 instances of resource type j (i.e., Rj) available.
  - ✓ Max[i,j] as the maximum demands of process i for resource type j.
  - ✓ Alloc[i,j] as the number of resource instances that are currently allocated to process i for resource type j. For example, Alloc[i,j]=5 means 5 instances of resource type j are currently allocated to process i.
  - ✓ Need[i,j] as the number of resource instances of resource type j that are currently needed by process. Here, Need[i,j] = Max[i,j] -Alloc[i,j].

Deadlock Avoidance

- Banker's Algorithm: find a safe sequence, i.e., is the system in a safe state?

1. Let ***Finish*** be the vector to indicate the processes are finished or not. Initially, Finish[i]=false for i=0,...,n-1

2. Find a process i such that
   - ✓ Finish[i]==false, and
   - ✓ Need[ij] ≤ Available[j], for all j

   If no such process i exists, go to step 4.

3. Available[j]:= Available[j]+Alloc[ij]
   Finish[i] = True
   Go to Step 2.

4. If Finish[i]==true for all i, then the system is in a safe state; otherwise, the system is not in a safe state.

Deadlock Avoidance

Example of Banker's Algorithm

- 3 resources (A,B,C) with total instances (10,5,7)
- 5 processes
- Snapshot at time t_0:

Process	Allocation	Max	Available	Need
	A B C	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2	7 4 3
P_1	2 0 0	3 2 2		1 2 2
P_2	3 0 2	9 0 2		6 0 0
P_3	2 1 1	2 2 2		0 1 1
P_4	0 0 2	4 3 3		4 3 1

Step 2:
P1's or P3's needs can be satisfied by what is available.

Deadlock Avoidance

❑ Example of Banker's Algorithm

- ➢ Let's allocate resources to P1 such that it can finish first.
- ➢ After P1 is finished, all the resources of P1 will be released.

Finish

Process	Allocation	Max	Available	Need
	A B C	A B C	A B C	A B C
P_0	0 1 0	7 5 3	~~3 3 2~~ 5 3 2	7 4 3
True P_1	~~2 0 0~~	~~3 2 2~~		~~1 2 2~~
P_2	3 0 2	9 0 2		6 0 0
P_3	2 1 1	2 2 2		0 1 1
P_4	0 0 2	4 3 3		4 3 1

Step 2:
P3's or P4's needs can be satisfied by what is available.

Deadlock Avoidance

❑ Example of Banker's Algorithm

- ➢ Let's allocate resources to P3 such that it can finish next.
- ➢ After P3 is finished, all the resources of P3 will be released.

Finish	Process	Allocation	Max	Available	Need
		A B C	A B C	A B C	A B C
	P_0	0 1 0	7 5 3	~~3 3 2~~ 7 4 3	7 4 3
True	~~P_1~~	~~2 0 0~~	~~3 2 2~~		~~1 2 2~~
	P_2	3 0 2	9 0 2		6 0 0
True	~~P_3~~	~~2 1 1~~	~~2 2 2~~		~~0 1 1~~
	P_4	0 0 2	4 3 3		4 3 1

Step 2:
All P0, P2 and P4 can be satisfied by what is available.

Thus, the sequence <P1, P3, …> is safe.

Deadlock Avoidance

Summary of deadlock avoidance

- a conservative strategy: process may have to wait even the resource requested is not in use.

- Drawbacks:

1. Difficult to predict a process's max resource requirement.
   ✓ giving a high estimate may result in long waiting time

2. High runtime overhead

3. All deadlock avoidance algorithms assume that processes are independent, that is, free from any synchronization constraint

- Deadlock Prevention
- Deadlock Avoidance
- Deadlock Detection and recovery

Detection and recovery

- Deadlock detection
  - Wait For Graph
  - Resource Allocation Graph
  - Banker's Algorithm (check if ***Finish(i)=="true"*** for all process i)
- Deadlock recovery
  - Recovery through preemption
    - ✓ take a resource from some other process
    - ✓ depends on nature of the resource
  - Recovery through rollback
    - ✓ checkpoint a process state periodically
    - ✓ rollback a process to its checkpoint state if it is found deadlocked
  - Recovery from process termination -- abort one process at a time until the deadlock cycle is eliminated. The other processes get its resources.

Detection and recovery

Deadlock recovery

- Process termination -- abort one process at a time until the deadlock cycle is eliminated.

- Which process is selected to abort?
  - ✓ Priority of the processes
  - ✓ How long the processes have computed, and how long the processes will be completed
  - ✓ Resources the processes has used
  - ✓ Resources processes needs to complete
  - ✓ Is process interactive or batch?