OPERATING SYSTEMS

#4

After A.S.Tanenbaum, Modern Operating Systems, 3rd edition

Uses content with permission from Assoc. Prof. Florin Fortis, PhD

DEADLOCKS

General Information

DEADLOCKS
RESOURCE TYPE

- A situation of deadlock usually occurs when the access to **resources** is made in an exclusive manner.
- Two types of resources could be identified:
  - *Preemptive resources*: resources that can be “obtained” from the process that is holding them;
  - *Non-preemptive resources*,: resources that cannot be “obtained”, not even forcibly, from the process that is holding them.
- Only non-preemptive resources can lead to a deadlock situation.
  - Question: WHY?

DEADLOCKS
RESOURCE TYPE

Non-preemptive resources

- We are using the term of ***resource*** to identify *non-preemptive resources*
  - These are the only ones that can be involved in a deadlock situation.
- Typical steps for using resources. Three simple steps are required:
  1. A request for getting the resource;
  2. The effective usage of the resource;
  3. A request for freeing the resource.
- Typically, these activities can be modeled by using simple devices, like semaphores.
- When a request for getting a resource fails, it can be further "solved" by simply using a busy-waiting cycle or a semaphore.
  - In any case, we will make the presumption that the process is waiting.

DEADLOCKS
RESOURCE TYPE

Resource acquisition

- Resources that are acquisitioned by processes can be managed at the level of the OS or at the level of each process.
- When the management is made at process level, it should be enough to simply use a binary semaphore or a mutex variable in order to gain the exclusive access to the resource
  - Needed because of the non-preemption.
- The management of a situation with two or more resources can be solved in the same manner, observing that processes should follow the same scenario in order to get or free resources.

DEADLOCKS DEFINITION

A set of processes is in a deadlock situation if every process in this set is waiting for an event that can be triggered only by another process in the same set.

- Each process is waiting, not being able to generate any event, eventually needed by another process:
  - all processes are condemned to an infinite wait!
- We are going to use the following assumptions:
  - Every process has only one thread, and any process can be waked up only by the occurrence of the waited event.

DEADLOCKS DEFINITION

- Four necessary conditions exist for a deadlock situation:

1. *Mutual exclusion*: every resource is either assigned to exactly one process, or available;
2. *Hold and wait*: if a process holds resources, it is able to ask for more resources;
3. *Inexistence of preemption*: no resource can be forcibly obtained from any process;
4. *Circular wait*: one can emphasize a chain of process, where each process is waiting for a resource that belongs to the previous process in the chain.

DEADLOCKS DEFINITION

- For a deadlock to occur, all the four conditions must be satisfied.
  - For example, when ***the third condition*** is missing, one can obtain resources from processes, and a deadlock situation cannot occur.
  - If the ***second condition*** is missing, then a process can hold at most one resource, so no process should ever wait forever for another resource.
- Notice that first three conditions are highly related with the local policy of the system;
- The last condition reflect more a "*real situation*" in the activity of a system.

DEADLOCKS
DEADLOCK MODELING

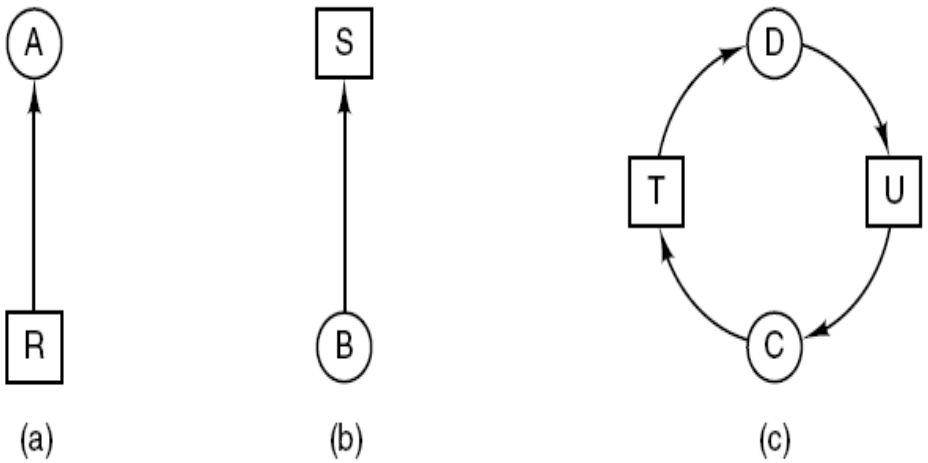
- In order to model a deadlock situation, one can use an oriented graph.
  - Represent processes by circles
  - Represent resources by squares.
- An arc that is going from a process to a resource indicates that the process is waiting for the resource to be available (is asking the resources).
- An arc from a resource to a process specify that the process is holding the resource.

RESOURCE ALLOCATION GRAPHS

(a) Holding a resource.

(b) Requesting a resource.

(c) A deadlock.



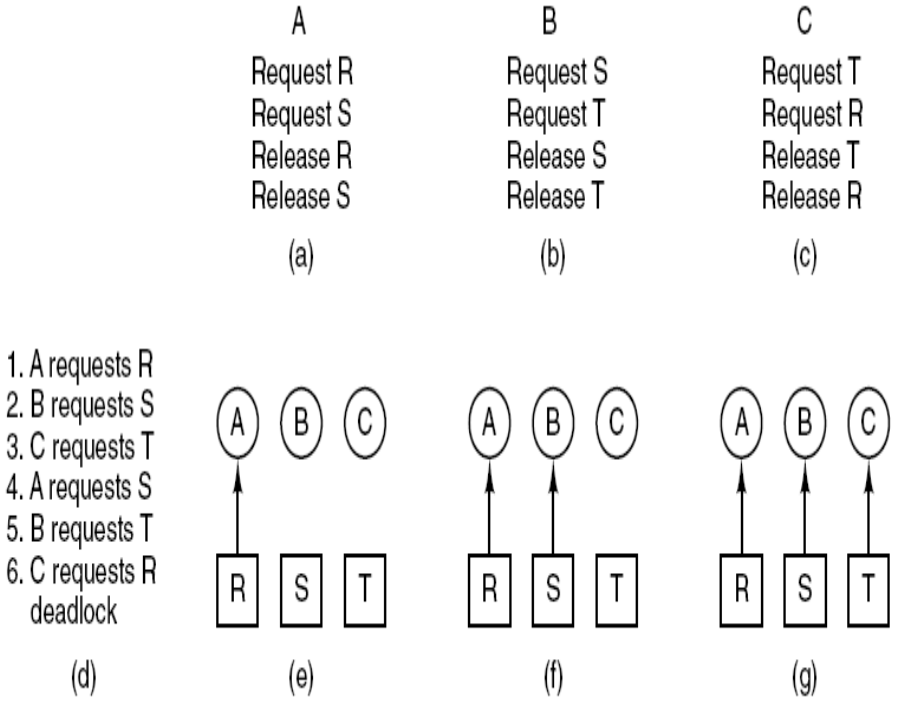
DEADLOCKS
DEADLOCK MODELING

- A deadlock situation cannot occur:
  - when the execution of processes is sequential,
  - when there is no competition for resources.
- Even if one cannot impose an execution order of the processes, the OS could postpone the execution of several processes in order to avoid a deadlock situation (thus waiting for a “sure” state).
- Allocation graphs can offer important information to the OS in order to detect a deadlock situation.

DEADLOCK OCCURRENCE

An example of how deadlock occurs

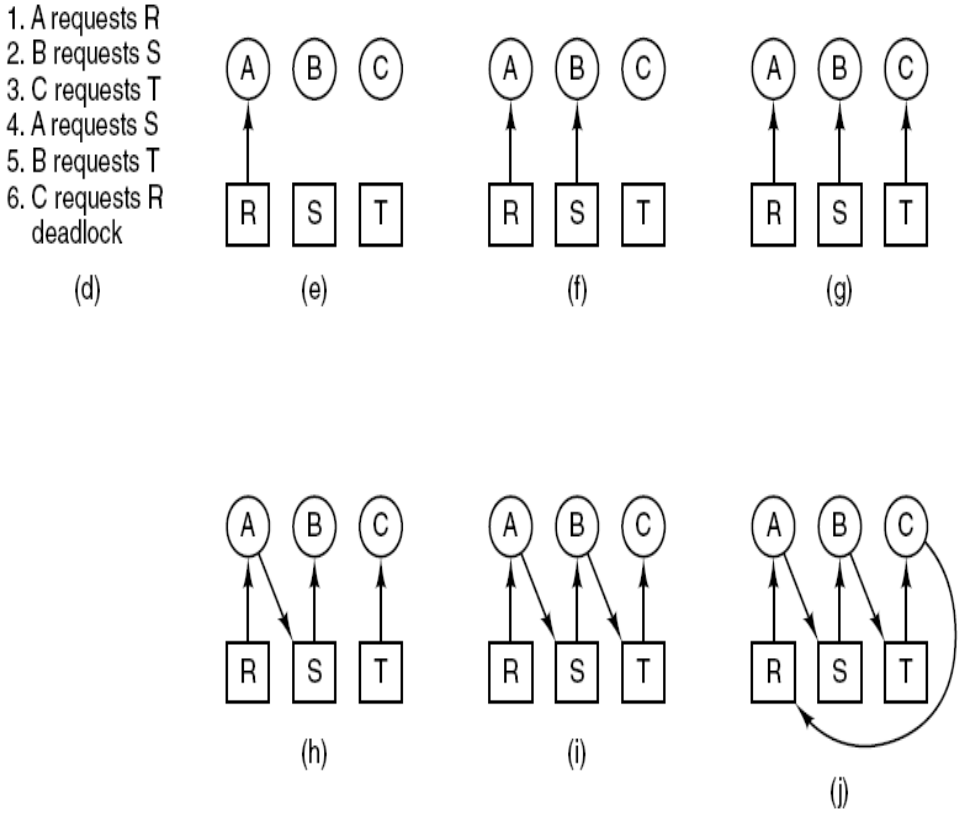
and how it can be avoided



DEADLOCK OCCURRENCE

An example of how deadlock occurs

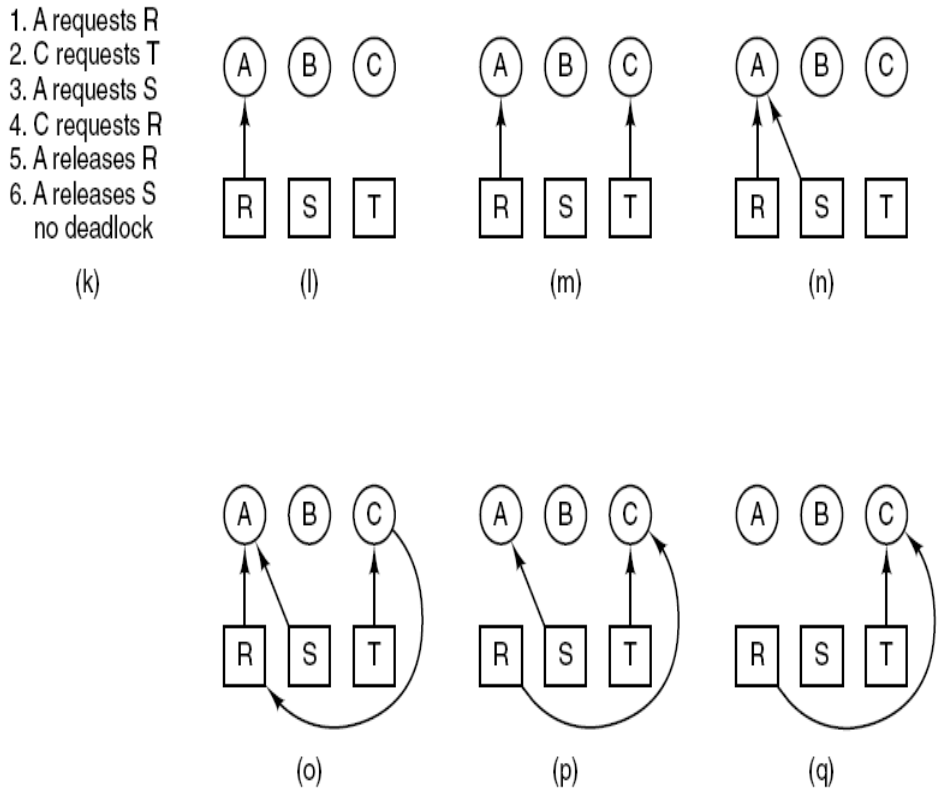
and how it can be avoided



DEADLOCK OCCURRENCE

An example of how deadlock occurs

and how it can be avoided



DEADLOCKS

- Strategies for dealing with deadlocks:

1. Just ***ignore*** the problem.
2. ***Detection and recovery***. Let deadlocks occur, detect them, take action.
3. ***Dynamic avoidance*** by careful resource allocation.
4. ***Prevention***, by structurally negating one of the four required conditions.

DEADLOCKS

The Ostrich algorithm

THE OSTRICH ALGORITHM IGNORE THE PROBLEM

- The ostrich algorithm is based on the following approach of the deadlock problem:
  - Completely ignore the situation, hoping that things are not going to be worst (or even hoping that the situation is able to solve by itself ☺)
- This “technique” is used when the frequency of a deadlock situation is incomparable smaller than the frequency of other major problems.
- Since the frequency of deadlock situations is quite small, PC operating systems are using this simple solution, because.
  - This solution is preferred in order to avoid other drastic limitations imposed by several approaches of the deadlock problem.
  - PC systems prefer convenience against fairness.

DEADLOCKS

Deadlock prevention

DEADLOCKS. PREVENTION MUTUAL EXCLUSION CONDITION

Deadlock situations can be prevented by "***attacking***" any of the four conditions.

- Attacking the mutual exclusion,
  - We can presume that the system is offering only preemptive resources.
  - However, any system is based on the existence of several non-preemptive resources (for example, a line printer).
- We can attack this condition by using the old "spooling" technique.
  - In this situation, we are able to avoid a deadlock situation, but in rare situations it is possible that another deadlock situation to occur at the level of disc space.
  - Question: Can you tell why?

DEADLOCKS. PREVENTION HOLD AND WAIT CONDITION

Attacking the hold and wait condition

1. Impose to the processes that are holding resources to wait for the resources to be available when they are requesting new resources.
   - If processes are able to declare from the beginning all the resources they need, then they can hold all the necessary resources from the beginning.
   - However, this approach is quite difficult to satisfy.
     - There are several systems (mainframes, for example), that are using this approach at some level.
2. A process that is requesting new resources should leave temporarily all its current resources, and obtain them back together with the other resources.
   - Now processes could be forced to have a continuous communication with the OS about resources!

DEADLOCKS. PREVENTION NON-PREEMPTION CONDITION

Attacking the non-preemption condition

- Non-preemptive resources could be of major importance in an operating system.
  - Question: why?
- This condition cannot be attacked for resources that can save/restore some state information (for example, the processor).
  - Question: can you name more resources?

DEADLOCKS. PREVENTION CIRCULAR WAIT CONDITION

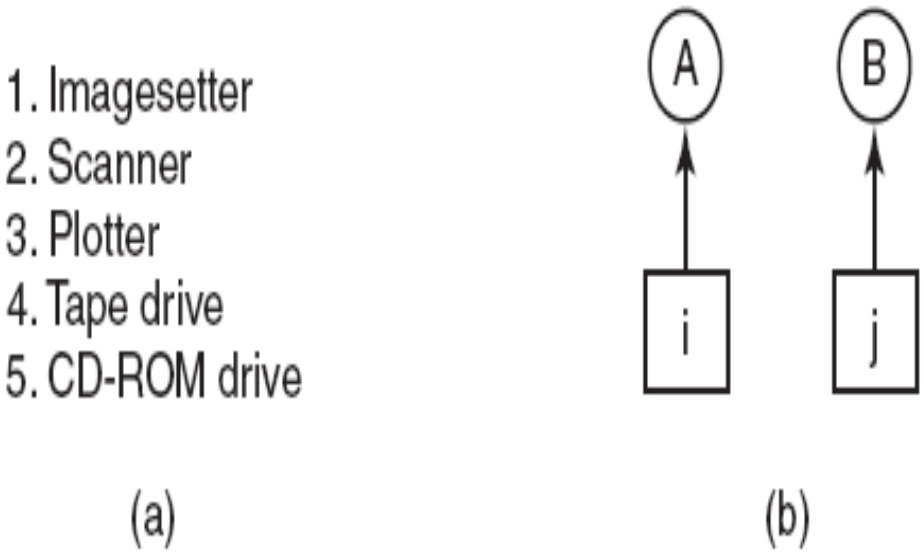
Attacking the circular wait

1. Limit the resources that are at the disposition of a process (for example, only one resource at a time).
   - In order to obtain another resource, the process could first release its current resource.
2. Resources are numbered in a circular manner.
   1. New resources can be obtained in the strict order of their numbers.
   2. The deadlock situation can be avoided since we can deny the access to resources for some processes.
   - However, this is not a feasible solution since it is not possible to impose a numbering scheme for all of the resources of the OS.
   - Moreover, it is even possible that several processes cannot access now resources that do not belong to any other process!

ATTACKING CIRCULAR WAIT CONDITION

(a) Numerically ordered resources.

(b) A resource graph.



DEADLOCKS

Detection and recovery

DEADLOCKS. DETECTION AND RECOVERY

- This technique can be used when, instead of trying to prevent a deadlock situation, the system instead prefers to detect the occurrence of such a situation and to react only in this moment.

- The actions to be carried out form the *recovering actions* that follow the detection of a deadlock situation.

DEADLOCKS. DETECTION AND RECOVERY ONE RESOURCE OF EACH TYPE

- The simplest situation that one can imagine is using only one resource of each type.
  - For this situation, the existence of an deadlock situation can be signaled by the existence of a cycle in the resource graph.
  - Each process in this cycle is in a deadlock situation.
- A system where one cannot emphasize a cycle in the resource graph is called ***free of deadlocks***.

DEADLOCKS. DETECTION AND RECOVERY ONE RESOURCE OF EACH TYPE

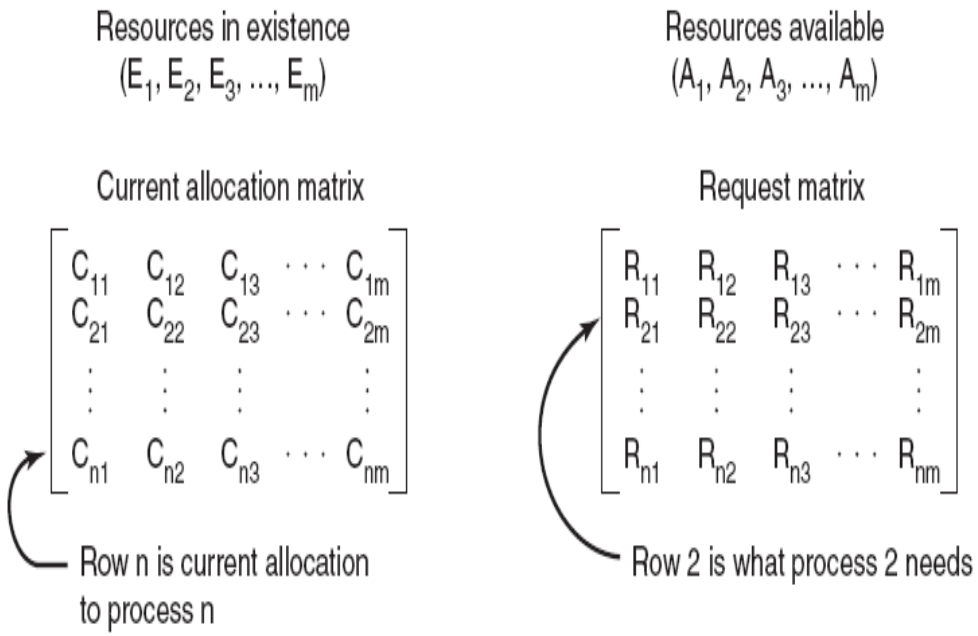
- **The algorithm:**
  1. For every node N of the graph, the following steps are executed, with N the start node.
  2. Initialize a list of nodes, L, with the empty list. All arcs are not marked.
  3. Add current node in the list L. If this node occurs twice in the list, a cycle has been identified.
  4. If there are unmarked arcs LEAVING current node, go to step 5, otherwise go to step 6.
  5. Mark one of the unmarked arcs. The node in which this arc is entering becomes current node. Go to step 3, using the updated current node.
  6. Remove current node from the list and go to the previous node (this becomes the new current node). If this is the initial node, STOP. Otherwise, go to step 3.

DEADLOCKS. DETECTION AND RECOVERY SEVERAL RESOURCES OF EACH TYPE

- Previous algorithm is not useful when there are several resources of every type.
  - Now resources are regrouped in classes of resources (resources of the same type are, obviously, in the same class).
- Processes are identified by P_i, while the number of resources in a class is E_j (where j is the class number)
  - Existent resources are grouped together in a vector E.
- All available resources are stored in the vector A (resources that are not allocated to other processes),
- The matrix C holds current allocation information.
- The matrix R holds the requests of the processes.

DETECTION AND RECOVERY SEVERAL RESOURCES OF EACH TYPE

The four data structures needed by the deadlock detection algorithm



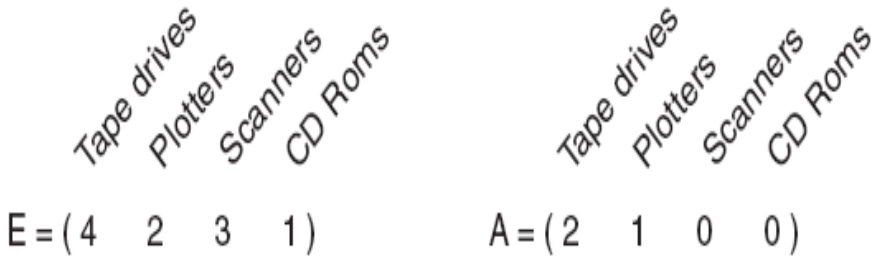
DEADLOCKS. DETECTION AND RECOVERY SEVERAL RESOURCES OF EACH TYPE

The algorithm for deadlock detection:

1. Look for an unmarked process P_i, such that its line in matrix R is less or equal with the values of the vector A (meaning that one can satisfy the allocation requests).
2. If there is such a process, add line i from matrix C to the vector A, mark the process, and go back to step 1.
3. If there is no such process, the algorithm STOPs.
4. After the execution of this algorithm, all the processes that are not marked should be in a deadlock situation. Marked processes are processes that are deadlock free.

DETECTION AND RECOVERY SEVERAL RESOURCES OF EACH TYPE

Example for deadlock detection algorithm



Current allocation matrix

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{bmatrix}$$

Request matrix

$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 \end{bmatrix}$$

DEADLOCKS. DETECTION AND RECOVERY DEADLOCK RECOVERY

- Detection alone cannot solve the deadlock problem.
  - In order to be efficient, one should establish the moments when the method is to be executed.
- If the execution is quite frequent, the performance of the entire system is dropping down.
- Appropriate moments could be moments of low activity of the system, or moments that come from a very rare verification scheme.

DEADLOCKS. DETECTION AND RECOVERY
DEADLOCK RECOVERY

- Mechanisms for unlocking the situation are based on the four conditions:

1. ***Preemption***: the system should obtain (forcibly, when required) some resources from a few processes in order to unlock other processes.
2. ***Come back***: system is able to keep information that define several periodic "check points".
   - When a deadlock occurs, the resource graph is determined again based on these information.
   - One of the processes in the deadlock situation is forced to go back to a previous checkpoint, offering necessary resources to other processes.
3. ***Process removal***: this is the final solution.
   - Several processes are destroyed, hoping that the others are able to go over the deadlock situation.
   - This mechanism should be used such that the consequences are minimal.

DEADLOCKS

Deadlock avoidance

DEADLOCKS. AVOIDANCE STATE TYPE

- The matrix and vectors E, A, C, R define the current state of the system.
- A state is ***sure*** if there is no deadlock and if processes can be planned such that they are able to finish their activity.
- If a state is not sure, it is ***unsure***.
- It is possible that, by a certain order of process planning, an ***unsure*** situation could be reached from a ***sure*** state.
- The deadlock could be avoided if the next state is always sure.

	Has	Max
A	3	9
B	2	4
C	2	7

Free: 3
(a)

	Has	Max
A	3	9
B	4	4
C	2	7

Free: 1
(b)

	Has	Max
A	3	9
B	0	–
C	2	7

Free: 5
(c)

	Has	Max
A	3	9
B	0	–
C	7	7

Free: 0
(d)

	Has	Max
A	3	9
B	0	–
C	0	–

Free: 7
(e)

- The state in (a) (first row) is sure; the state in (b) (second row) is unsure

	Has	Max
A	3	9
B	2	4
C	2	7

Free: 3
(a)

	Has	Max
A	4	9
B	2	4
C	2	7

Free: 2
(b)

	Has	Max
A	4	9
B	4	4
C	2	7

Free: 0
(c)

	Has	Max
A	4	9
B	—	—
C	2	7

Free: 4
(d)

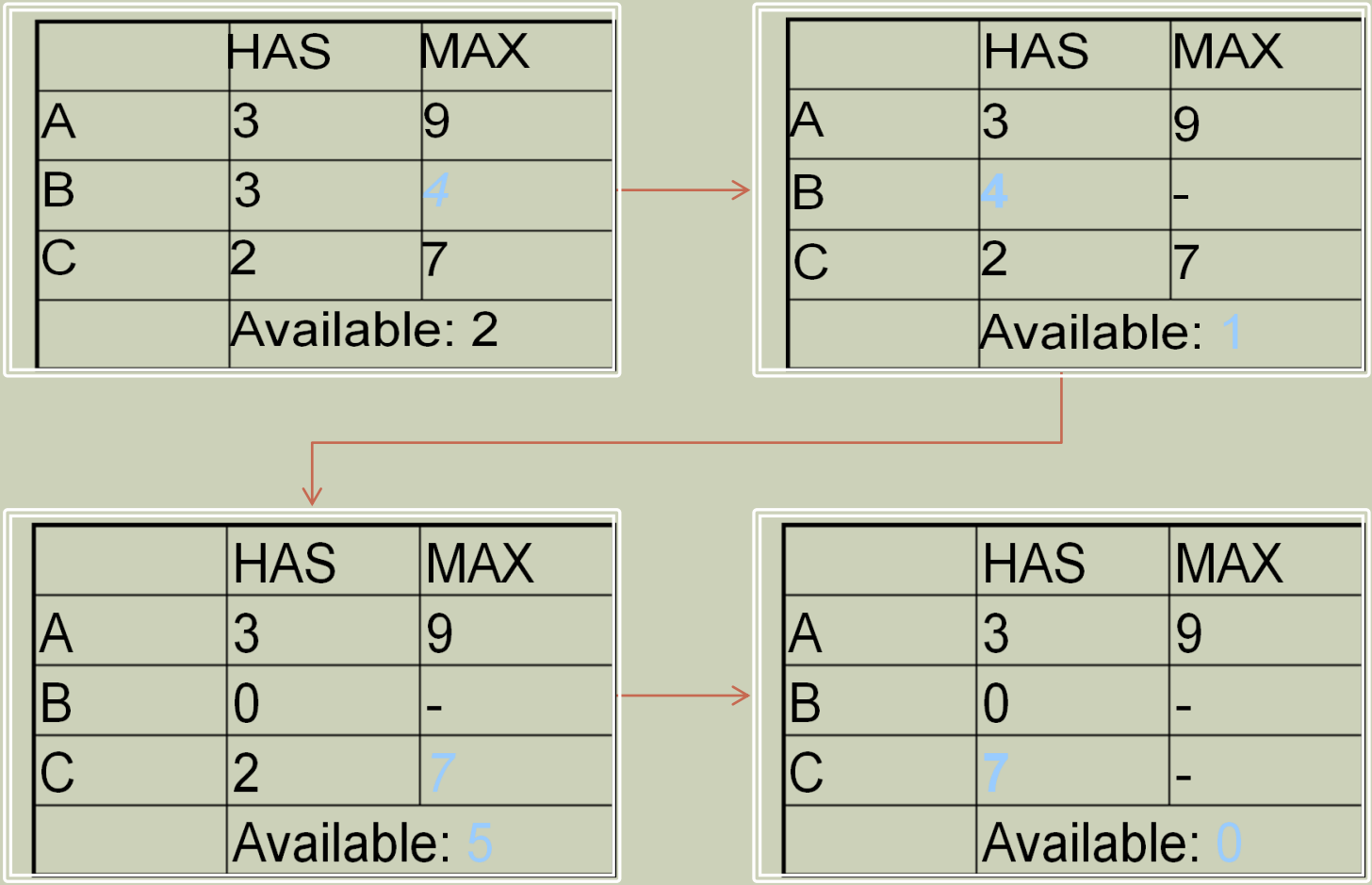
DEADLOCKS. AVOIDANCE
THE BANKER'S ALGORITHM

- The Banker's algorithm (due to Dijkstra) is used to verify if by satisfying the requests of a process (by scheduling the process), it is possible to reach an unsure state. The request is denied in this situation.
- In order to verify if a state is sure, the banker should check first if the available resources are enough to fulfill the requirements of some client.
- Notice that every request is treated just after its occurrence. Requests that can lead to a deadlock situation should be denied and postponed, if necessary.

DEADLOCKS. AVOIDANCE
THE BANKER'S ALGORITHM FOR SINGLE RESOURCE

- This is a very simple situation.
- In this algorithm, a request is treated as soon as it occurs:
  - If a request can lead to an unsure state, the request is denied or postponed until it can lead to a sure state.
  - From a sure state, the banker must check only if the remaining existing resources are enough for the requirements of a client.

DEADLOCKS. AVOIDANCE
THE BANKER'S ALGORITHM FOR SINGLE RESOURCE



DEADLOCKS. AVOIDANCE
THE BANKER'S ALGORITHM FOR SINGLE RESOURCE

- If **Avail=1**, this state becomes unsure!
- If process A requests one resource, then offering the resource to process A we can reach an unsure state.
- Following the simple Banker's algorithm, in this state each request coming from process A must be denied.
- Process C requests could be denied too, even if allocating one resource to process C cannot change this state!

DEADLOCKS. AVOIDANCE
THE BANKER'S ALGORITHM FOR SINGLE RESOURCE

Initial situation

	Has	Max
A	0	6
B	0	5
C	0	4
D	0	7
	Available: 10	

Sample requests

- For the requests (1, 1, 2, 4) the new state is sure.
  - Only one process is able to continue!
- If after this the requests are (0, 1, 0, 0), the new state is unsafe.
  - Can you say why?

DEADLOCKS. AVOIDANCE
THE BANKER'S ALGORITHM FOR SINGLE RESOURCE

Initial situation

	Has	Max
A	70	45
B	60	40
C	60	15
	Total: 150	

Sample requests

- Is this a sure state?
  - Show a possible path to justify your answer.
- If there is a new process, D, with Max=60 and Has (initial)=25, the new state is sure?
  - What if for the new process Has=35?
- Detect the maximum value for Has for process D such that the new state to be sure

DEADLOCKS. AVOIDANCE
THE BANKER'S ALGORITHM: MULTIPLE RESOURCES

- The generalized version of the Banker's algorithm.
- For this algorithm we are using again
  - the current allocation matrix (C),
  - the requested resources matrix (R),
  - the array of available resources (A),
  - the array of existent resources (E),
  - and the array of resources at processes disposition (P).
- The algorithm consists of the following simple steps:

DEADLOCKS. AVOIDANCE
THE BANKER'S ALGORITHM: MULTIPLE RESOURCES

1. Look for row, R, whose unmet resource needs all ≤ A. If no such row exists, system will eventually deadlock since no process can run to completion
2. Assume process of row chosen requests all resources it needs and finishes. Mark process as terminated, add all its resources to the A vector.
3. Repeat steps 1 and 2 until either all processes marked terminated (initial state was safe) or no process left whose resource needs can be met (there is a deadlock).

DEADLOCKS. AVOIDANCE
THE BANKER'S ALGORITHM: MULTIPLE RESOURCES

State Matrix description

The R Matrix			
2	0	0	1
1	0	1	0
2	1	0	0

The C Matrix			
0	0	1	0
2	0	0	1
0	1	0	0

- Consider
  - A=(2,1,0,0)
- Detect the values of E.
  - Is this a deadlock situation?

DEADLOCKS. AVOIDANCE
THE BANKER'S ALGORITHM: MULTIPLE RESOURCES

State Matrix description

The MAX Matrix			
0	0	1	2
2	7	5	0
6	6	5	6
4	3	5	6
0	6	5	2

The C Matrix			
0	0	1	2
2	0	0	0
0	0	3	4
2	3	5	4
0	3	3	2

- Consider
  - A=(2,1,0,0)
- Detect the values of E.
  - Is this a sure state?
- Is the system in a deadlock situation?
  - Which of the five processes could reach a deadlock?
  - What happens if (0,1,0,0) was granted for Process 3?

EXERCISES

- Consider a situation where A=(2, 1, 0, 0),
- Max=

0	0	1	2
2	7	5	0
6	6	5	6
4	3	5	6
0	6	5	2

, C=

0	0	1	2
2	0	0	0
0	0	3	4
2	3	5	4
0	3	3	2

- Detect the value of E. Is this a sure state?
- Is the system in a deadlock situation?
- Which of the 5 processes could reach a deadlock?
- Analyze the situation when a request of (0, 1, 0, 0) is granted to process P3.