Deadlocks

In everyday life:

- You can't get a job if you have no experience, but you can't get experience if you have had no job.
- "It takes money to make money"
- The "**chicken and the egg**": You can't have a chicken if there was no egg, but you can't have an egg if there was no chicken.

In computing (simple example):

- Process A is waiting for some resource, that is held by process B, to continue its execution. Meanwhile, process B is waiting for some (different) resource, that is held by process A, to continue its execution.

Typical examples of resources are CPU cycles, RAM (memory), I/O devices (printers etc.)

Conditions for Deadlock

ALL of the following must hold SIMULTANEOUSLY to have a deadlock:

Mutual Exclusion At least one process is holding/using a resource exclusively (i.e. that resource is not / cannot be used by any other process at the same time).

Hold and Wait A process is holding a resource while waiting for another (different) resource.

No preemption A process cannot be "forced" to release a resource it is holding.

Circular Wait A circular wait with n processes is where

- Process P_1 is waiting for a resource held by process P_2,
- Process P_2 is waiting for a resource held by process P_3,
- Process P_3 is waiting for a resource held by process P_4,

  $\vdots$

- Process P_n is waiting for a resource held by process P_1,

The Dining Philosophers (from *E. W. Dijkstra*)

There are five philosophers seated at a round table. In front of each philosopher is a plate of spaghetti. Between each plate of spaghetti, there is one fork (so 5 forks on the table). Each philosopher can only eat using two forks. Philosophers spend all their time either eating spaghetti or thinking (but not both) ☺.

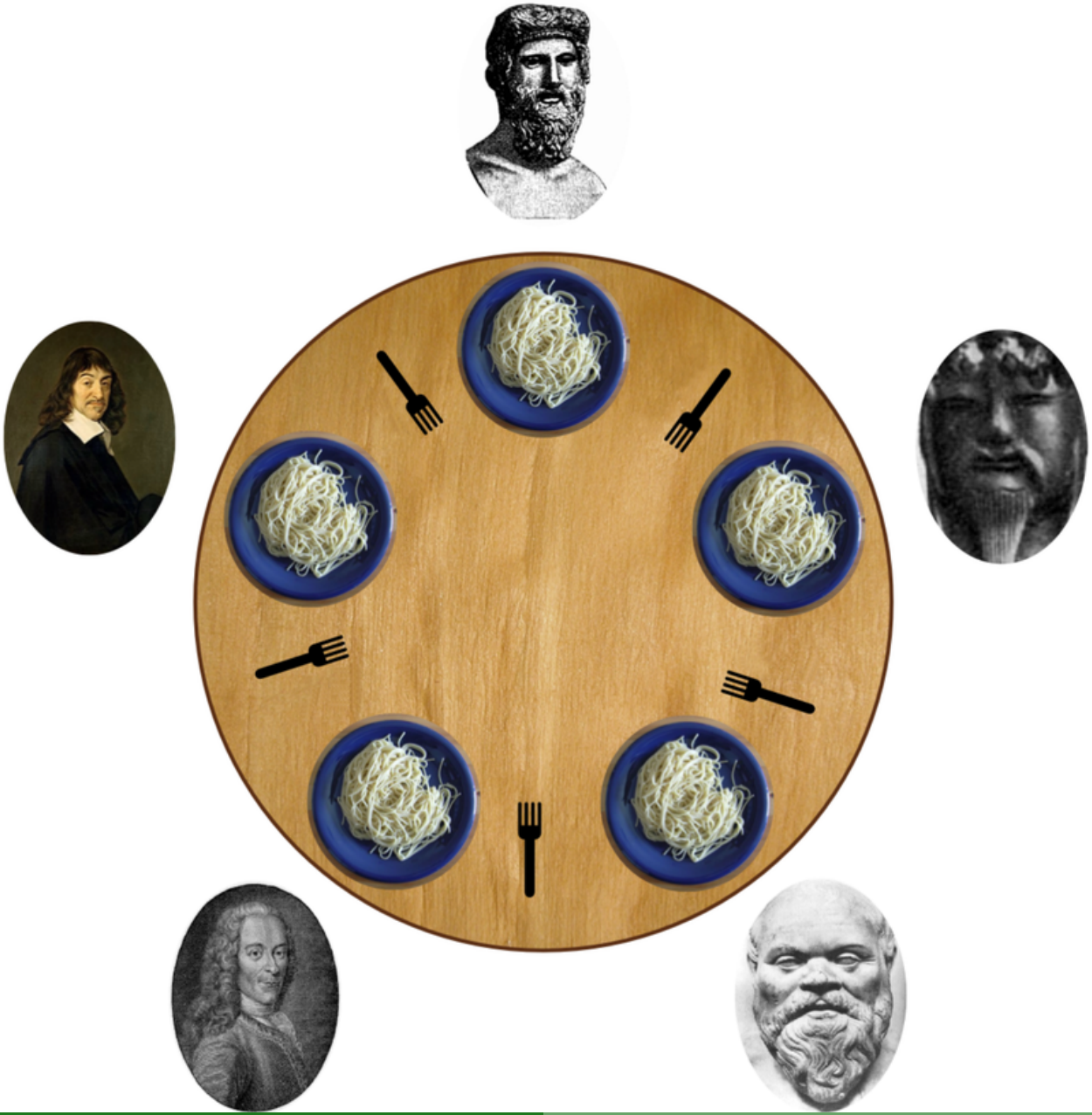
Problem: Design an algorithm (sequence of instructions) such that no Philosopher will starve.

Algorithm 1 Each Philosopher picks up the fork to his/her left, holds on to it, and waits for the fork to the right to become available in order to eat. **Result:** All philosophers starve.

Algorithm 2 Philosophers 1 and 3 pick up forks on each side and eat, without stopping. **Result:** Philosophers 2, 4, and 5 starve.

Algorithm 3 Philosophers 1 and 3 pick up forks on each side and eat. After a fixed amount of time, they put down both forks and think, while philosophers 2 and 4 pick up both forks to eat. After a fixed amount of time, they put down both forks and think, while philosophers 3 and 5 pick up their forks. Following this pattern, the philosophers who eat are the pairs $(1, 3), (2, 4), (3, 5), (4, 1), (5, 2), (1, 3), \ldots$ etc.

The Dining Philosophers (from *E. W. Dijkstra*)



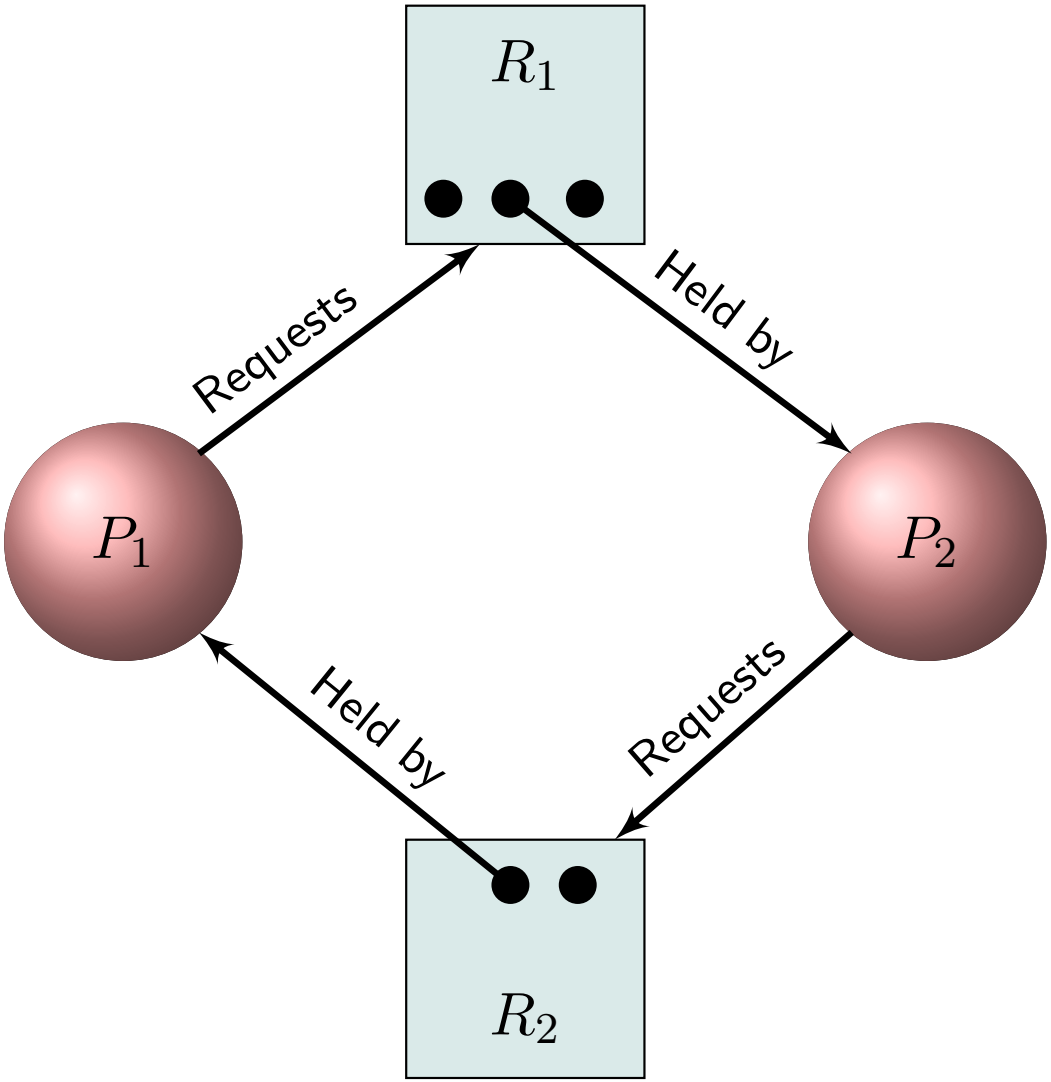
Resource Allocation Graphs

This describes which resources are being (a) used or/and (b) requested by which processes. The graph has

- **nodes** representing the processes (traditionally circles) $P_1, P_2, \ldots P_n$, and the resources (traditionally squares) $R_1, R_2, \ldots R_m$.
- **(directed) edges**

  from P_i to R_j meaning that process P_i is requesting resource R_j.
  from R_i to P_j meaning that resource R_i has been assigned to (and is being used by) process P_j.

Each resource may have multiple identical/equivalent copies, represented by separate dots inside the square representing that resource.



In this Resource Allocation Graph, process P_1 is using ("holds") resource R_2, while it is requesting resource R_1. Meanwhile, process P_2 is using resource R_1 while it is requesting resource R_2. There are three copies of resource R_1 and two copies of resource R_2. Because of the multiple resource copies, there is no deadlock.

A cycle in the Resource Allocation Graph is necessary but not sufficient for deadlock.

System State

The system state at any given time can be described by the following data:

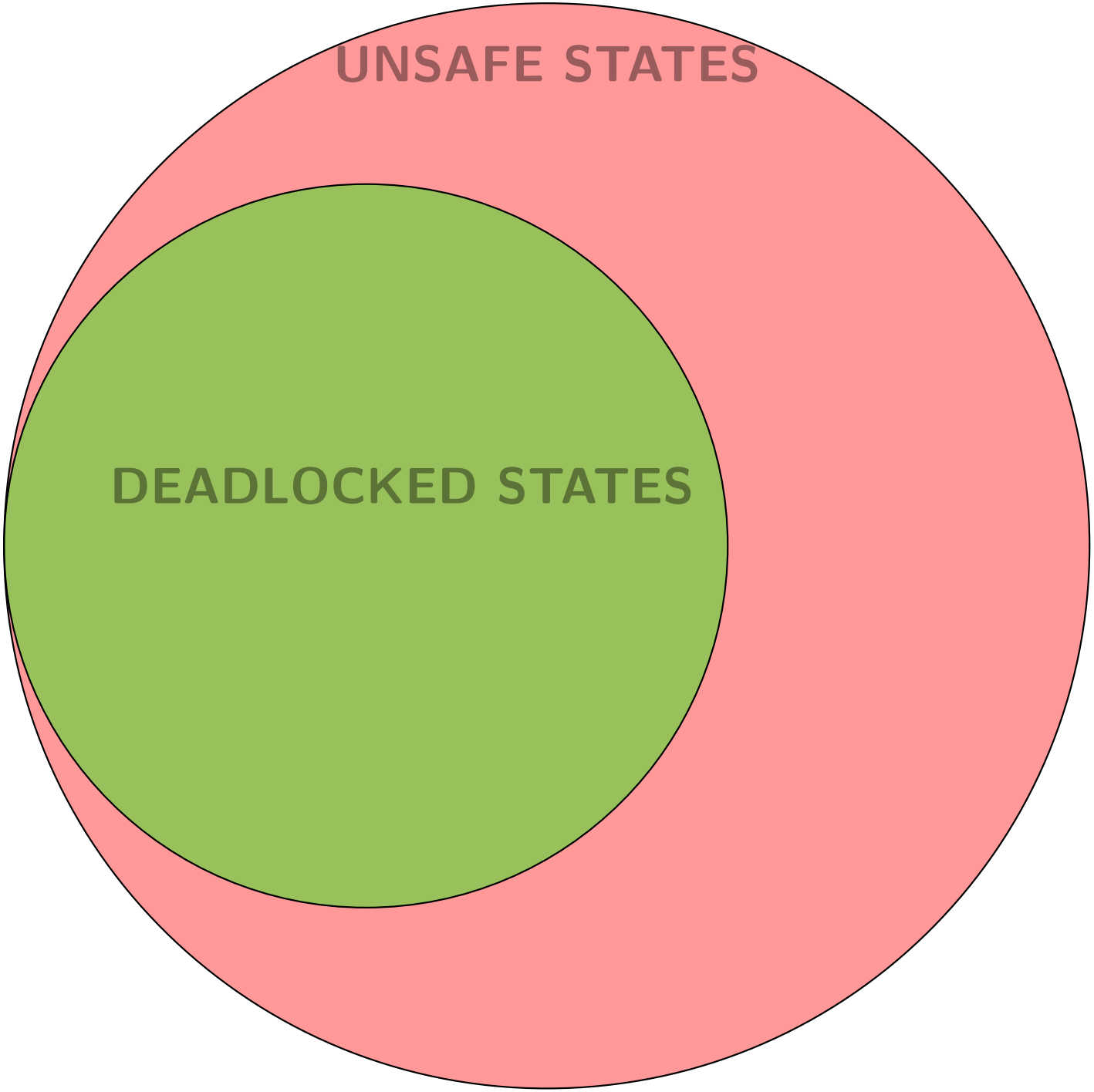
1. A vector giving total amounts of each resource in the system, e.g. $(R_1, R_2, R_3) = (1, 5, 2)$, means there are two instances of resource 3, five instances of resource 2 and one instance of resource 1. This is time invariant.
2. A vector giving total amounts of each resource that are currently free, $(F_1, F_2, \ldots F_m)$. This varies with time, and note that $F_i \leq R_i$ for all i.
3. Resource requirements for each process: C_{ij}, the amount of resource j that is needed by process i (time invariant). Note that C is in general not a square matrix.
4. Current resource allocations: A_{ij}, the amount of resource j that is currently "held" by process i. This varies with time, and $A_{ij} \leq C_{ij}$ for all i, j.

Safe State

A **Safe State** is one which permits a **safe sequence** (of processes).

Safe Sequence

A **Safe Sequence** $P_{i_1}, P_{i_2}, P_{i_3}, \ldots$ is one where we have enough resources to execute P_{i_1}, and at each successive step, the resources released by termination of previous processes in the sequence are sufficient to allow execution of the current process in the sequence.


UNSAFE STATES
DEADLOCKED STATES

Example: Safe State

In this example, we consider $n = 3$ processes and $m = 1$ resource, of which there are 12 instances. The process requirements are given by $C_{11} = 10, C_{21} = 4, C_{31} = 9$. The current allocations are $A_{11} = 5, A_{21} = 2, A_{31} = 2$. This state is a safe state because it admits the safe sequence P_2, P_1, P_3:

Before execution of P_2: $F = (3), A^t = (5, 2, 2)$

After execution of P_2: $F = (5), A^t = (5, 0, 2)$

After execution of P_1: $F = (10), A^t = (0, 0, 2)$

After execution of P_3: $F = (12), A^t = (0, 0, 0)$

Example: Unsafe State

Consider the previous example, where we give process P_3 one more instance of the resource, to produce $A^t = (5, 2, 3), F = (2)$. Now we can only execute process P_2 to get $A^t = (5, 0, 3), F = (4)$. We are now in a deadlocked state: P_1 needs five instances of the resource, while P_3 needs six, and there areonly four instances available.

Example 1.

State S1

$$C = \begin{pmatrix} 3 & 2 & 2 \\ 6 & 1 & 3 \\ 3 & 1 & 4 \\ 4 & 2 & 2 \end{pmatrix} \quad A = \begin{pmatrix} 1 & 0 & 0 \\ 6 & 1 & 2 \\ 2 & 1 & 1 \\ 0 & 0 & 2 \end{pmatrix} \quad R = \begin{pmatrix} 9 & 3 & 6 \end{pmatrix} \quad F = \begin{pmatrix} 0 & 1 & 1 \end{pmatrix}$$

We prove this is a safe state by showing P_2, P_1, P_3, P_4 is a safe sequence:

- Execute P_2:

State S2

$$C = \begin{pmatrix} 3 & 2 & 2 \\ 0 & 0 & 0 \\ 3 & 1 & 4 \\ 4 & 2 & 2 \end{pmatrix} \quad A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 2 & 1 & 1 \\ 0 & 0 & 2 \end{pmatrix} \quad R = \begin{pmatrix} 9 & 3 & 6 \end{pmatrix} \quad F = \begin{pmatrix} 6 & 2 & 3 \end{pmatrix}$$

- Execute P_1:

State S3

$$C = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 3 & 1 & 4 \\ 4 & 2 & 2 \end{pmatrix} \quad A = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 2 & 1 & 1 \\ 0 & 0 & 2 \end{pmatrix} \quad R = \begin{pmatrix} 9 & 3 & 6 \end{pmatrix} \quad F = \begin{pmatrix} 7 & 2 & 3 \end{pmatrix}$$

- Execute P_3:

State S4

$$C = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 4 & 2 & 2 \end{pmatrix} \quad A = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 2 \end{pmatrix} \quad R = \begin{pmatrix} 9 & 3 & 6 \end{pmatrix} \quad F = \begin{pmatrix} 9 & 3 & 4 \end{pmatrix}$$

- Execute P_4:

State S5

$$C = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \quad A = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \quad R = \begin{pmatrix} 9 & 3 & 6 \end{pmatrix} \quad F = \begin{pmatrix} 9 & 3 & 6 \end{pmatrix}$$

The Banker's Algorithm

This is an algorithm for calculating a safe sequence (if it exists) and hence avoiding deadlock. When a process makes a request for resources,

1. If the resources are not available, tell the process to wait.
2. If the resources are available, determine if granting the resources and executing the process will result in a safe state: if it does, grant the resources to the process.
3. On the other hand, if granting the resources will result in an unsafe state, deny the request and tell the process to wait.

The Banker's Algorithm (step by step)

Using our state parameters C, A, R, F:

1. Calculate the matrix $N = C - A$ (N stands for "needs", it stores the resources currently needed by the different processes).
2. Find a row in N that is less than or equal to F: i.e. find some k for which $N_{kj} \leq F_j$ for all values of j. If there is no such row the state is not a safe state.
3. Suppose the row you have found is the kth. row: Add the values on the kth. row of A to F, then set all the values in the kth. rows of C, A and N to zero (this corresponds to running process P_k).
4. Repeat from step 2. above: If at any stage all the elements of the matrix C are zero (meaning all processes have terminated) then the original state was a safe state.

Example 2.

QUESTION: Use the Banker's algorithm to determine whether or not the following state is a safe state.

State S1

$$C = \begin{pmatrix} 7 & 5 & 3 \\ 3 & 2 & 2 \\ 9 & 0 & 2 \\ 2 & 2 & 2 \\ 4 & 3 & 3 \end{pmatrix} \quad A = \begin{pmatrix} 0 & 1 & 0 \\ 2 & 0 & 0 \\ 3 & 0 & 2 \\ 2 & 1 & 1 \\ 0 & 0 & 2 \end{pmatrix} \quad R = \begin{pmatrix} 10 & 5 & 7 \end{pmatrix} \quad F = \begin{pmatrix} 3 & 3 & 2 \end{pmatrix}$$

We calculate

$$N = C - A = \begin{pmatrix} 7 & 4 & 3 \\ 1 & 2 & 2 \\ 6 & 0 & 0 \\ 0 & 1 & 1 \\ 4 & 3 & 1 \end{pmatrix}$$

Example 2. (continued)

Since the second row of N is less-than-or-equal-to F, we add the second row of A to F and zero the second rows of $C, A,$ and N (corresponding to allocating its requested resources to P_2 and executing P_2).

State S2

$$C = \begin{pmatrix} 7 & 5 & 3 \\ 0 & 0 & 0 \\ 9 & 0 & 2 \\ 2 & 2 & 2 \\ 4 & 3 & 3 \end{pmatrix} \quad A = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 3 & 0 & 2 \\ 2 & 1 & 1 \\ 0 & 0 & 2 \end{pmatrix} \quad R = \begin{pmatrix} 10 & 5 & 7 \end{pmatrix} \quad F = \begin{pmatrix} 5 & 3 & 2 \end{pmatrix}$$

Example 2. (continued)

Proceeding with the successive steps of the Banker's Algorithm:

1. Since $N_{4j} \leq F_j$ for all j, execute process P_4, giving $F = (7, 4, 3)$.
2. Since $N_{5j} \leq F_j$ for all j, execute process P_5, giving $F = (7, 4, 5)$.
3. Since $N_{1j} \leq F_j$ for all j, execute process P_1, giving $F = (7, 5, 5)$.
4. Since $N_{3j} \leq F_j$ for all j, execute process P_3, giving $F = (10, 5, 7)$.

Since we have found a safe sequence of process execution, the original state is a safe state.