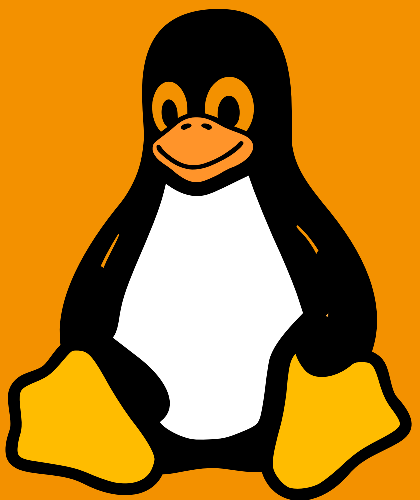
Technische Hochschule Rosenheim
Technical University of Applied Sciences


# Prof. Florian Künzner

Technical University of Applied Sciences Rosenheim, Computer Science


# OS 10 – Deadlocks

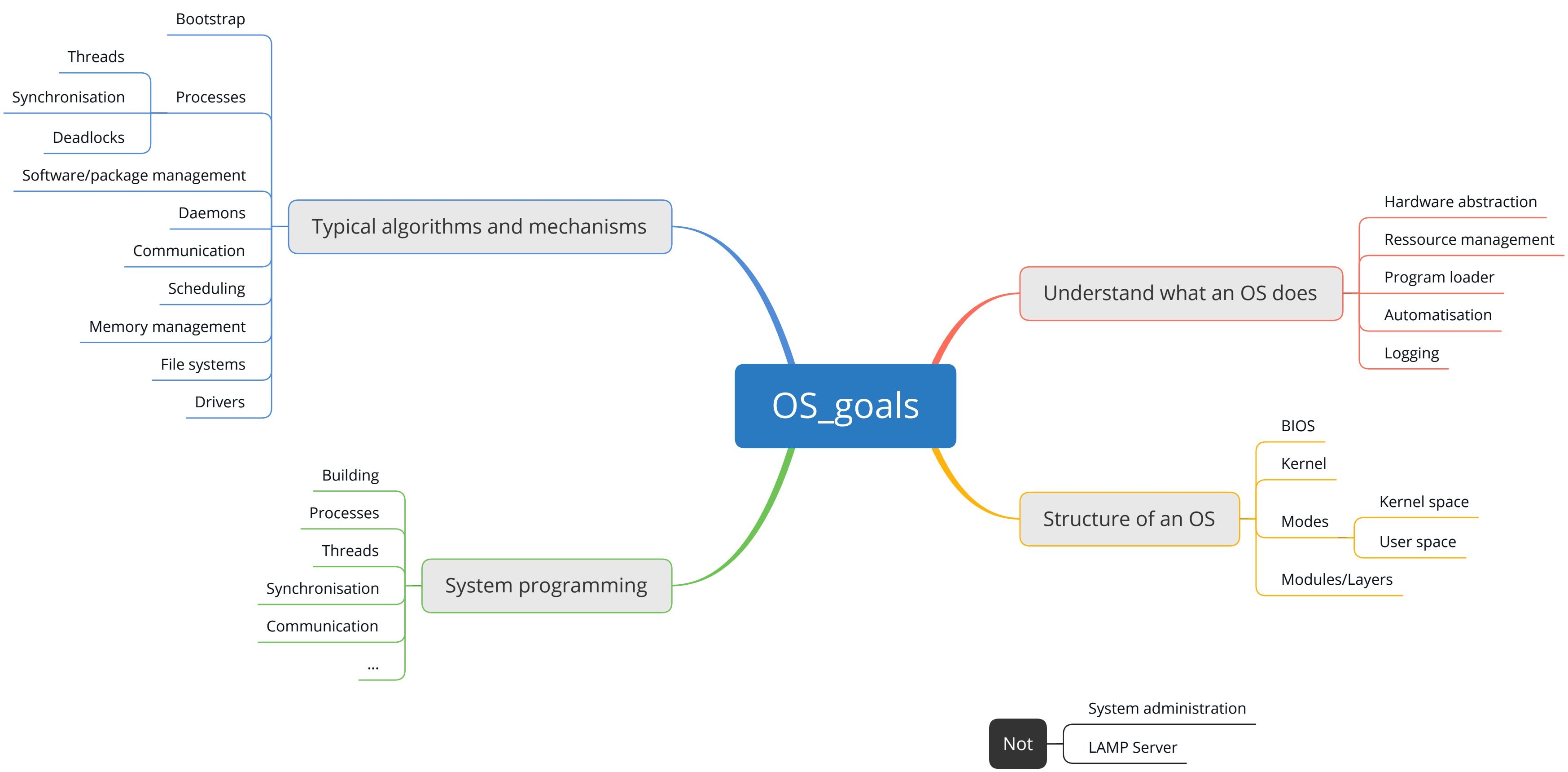

source: iconspng.com

**The lecture is based on the work and the documents of Prof. Dr. Ludwig Frank**

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

# Goal



**CAMPUS Rosenheim**
**Computer Science**

# Goal

## OS::Deadlocks

- Intro
- Analysis
- Safe states
- Prevention
- Deadlock recovery

CAMPUS Rosenheim
Computer Science

# Intro

**Parallelisation with processes and threads and their synchronisation is nice, but...**

**CAMPUS Rosenheim**
**Computer Science**

Technische Hochschule Rosenheim
Technical University of Applied Sciences

# Intro

## Example 1

time

```
process1() {
  request(resource_a);
  //process change
  request(resource_b);
  //...
}
```

```
process2() {
  request(resource_b);
  //process change
  request(resource_a);
  //...
}
```

## Problem

- **Deadlock**
- Reason: Resource is **already assigned** to other process (**both wait**)

# Intro

## Example 2

```
seminit(s, 0);
process1() {
     P(s);
     //critical area..,
     V(s);
}
process2() {
     P(s);
     //critical area..,
     V(s);
}
```

## Problem

- **Deadlock**
- Reason: Critical area can't be accessed, because the **semaphore is initialised with 0**.

CAMPUS Rosenheim
Computer Science

# Definition

A **deadlock** is a situation where one ore more **processes wait for resource(s)** and **no one is able to get its required resource(s)**, because they are **waiting**.

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

# Analysis

**How can a deadlock occur? We try a systematic analysis.**

**CAMPUS Rosenheim**
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

# Deadlock characterisation

## A deadlock can occur under these conditions

| Condition | Description |
|---|---|
| **Mutual exclusion** | Tasks **claim exclusive control** of the resources they require ("**mutual exclusion**" condition). |
| **Hold and wait** | Tasks **hold resources already allocated** to them while waiting for additional resources ("**wait for**" condition). |
| **No preemption** | Resources **cannot be forcibly removed** from the tasks holding them until the resources are used to completion ("**no preemption**" condition). |
| **Circular wait** | A circular chain of tasks exists, such that **each task holds one or more resources that are being requested by the next task** in the chain ("**circular wait**" condition). |

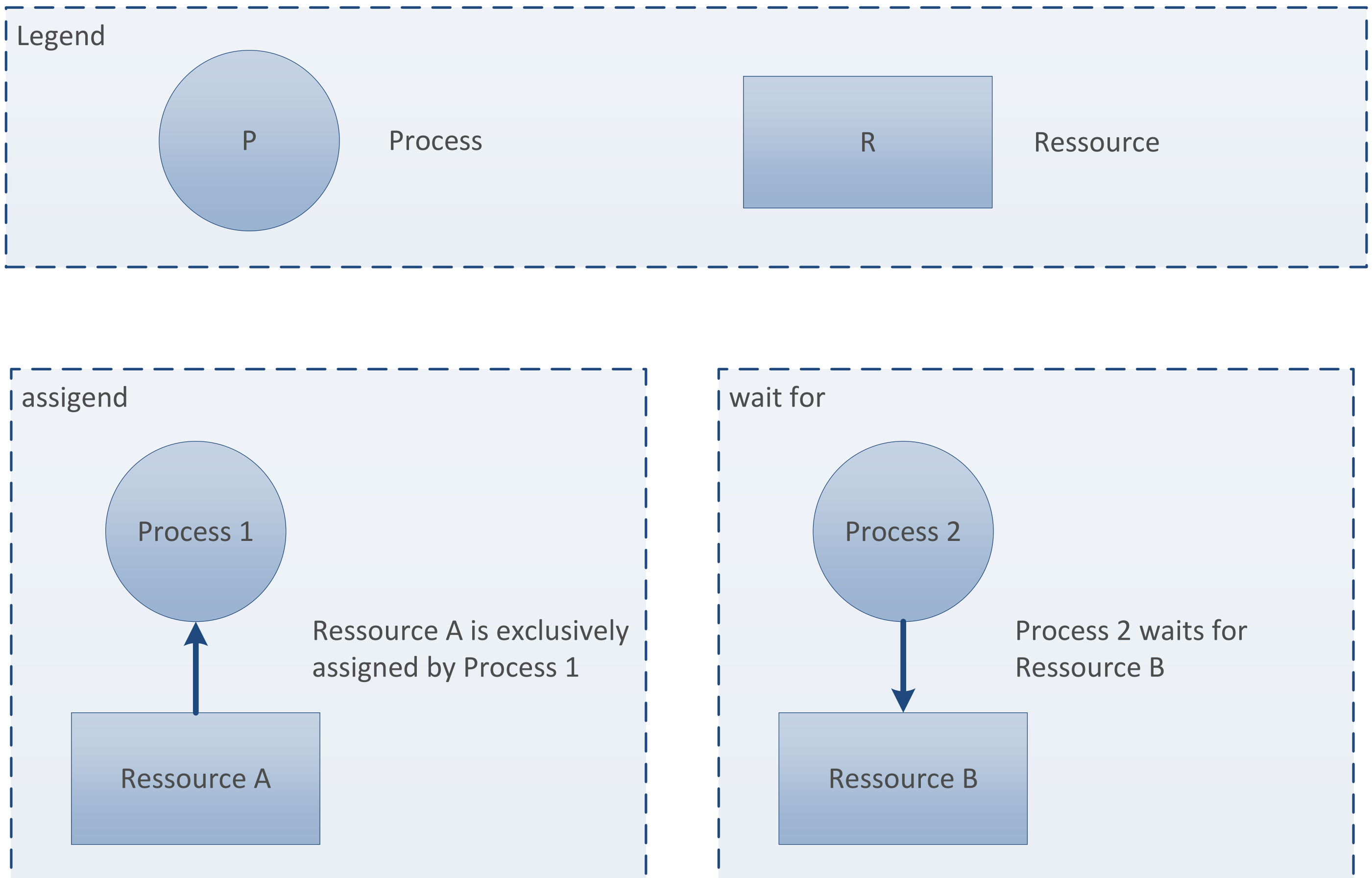
Original paper: (E. G. COFFMAN et al., 1971) coffman_deadlocks.pdf

More details: `https://en.wikipedia.org/wiki/Deadlock`

**CAMPUS Rosenheim**
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

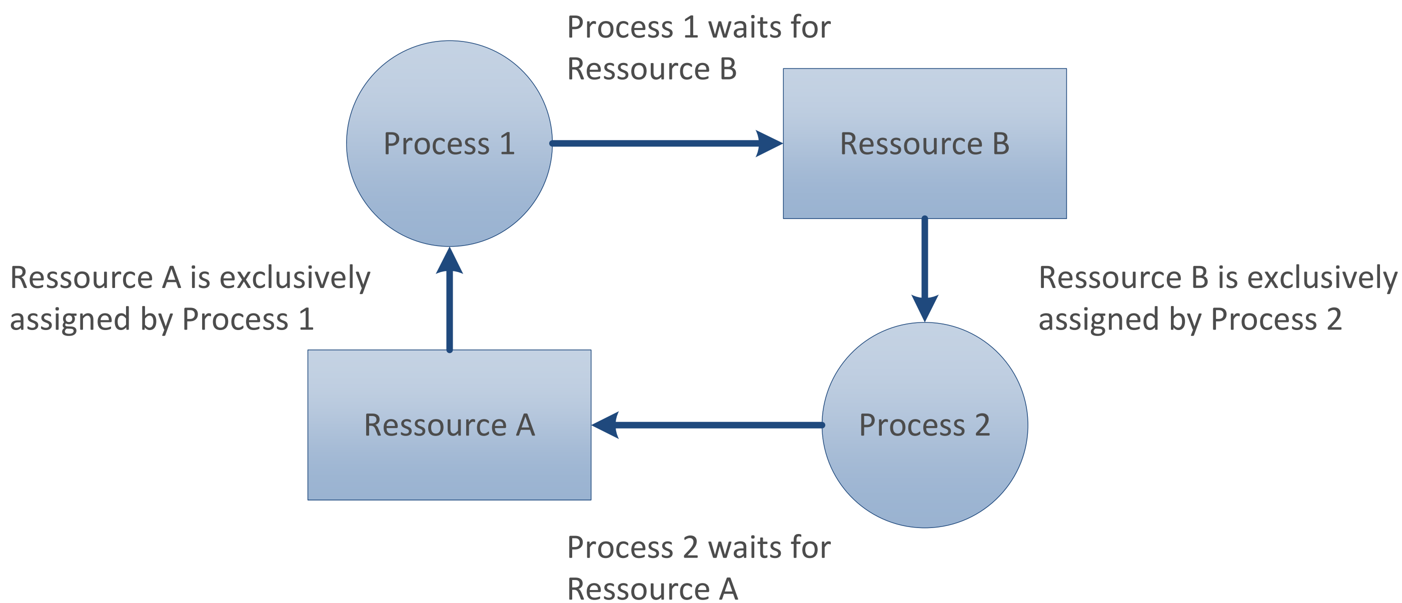
# Analysis: notation



CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

# Analysis: example 1 (graphically)




- P1 <- A, P1 -> B, P2 <- B, P2 -> A
- **Circular wait: deadlock.**

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

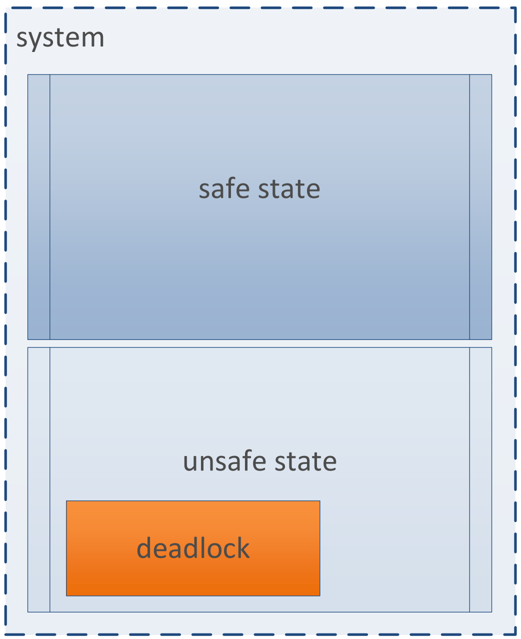
# Safe states

## Is the system in a safe state?

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

# Safe states




## Safe state

- A state is **safe**, if there is **no deadlock**, and there **exists at least one sequence** of processes that **ends not** in a **deadlock**.
- It is **guaranteed** that all processes can finish.

## Unsafe state

- A state is **unsafe** if there is **a deadlock**, or there **exists only sequences** that **ends** in a **deadlock**.
- It is **not guaranteed** that all processes can finish.
- A deadlock is an unsafe state.

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

# Safe states: terms

## Safe sequence of processes

A sequence of processes $(P_1, \dots, P_n)$ is safe for a certain state of acquired resources, if for every process $P_i$ $(1 \leq i \leq n)$ it is guaranteed that all required resources $(R_1, \dots, R_m)$ can be granted including the resources that are acquired by $P_j$ $(j < i)$.

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

# Safe states: example 1

There are 12 resources in total available.

| Process | Max. need | Acquisition $T_0$ | $T_1$ | $T_2$ | $T_3$ | $T_4$ | $T_5$ | $T_6$ |
|---|---|---|---|---|---|---|---|---|
| $P_1$ | 10 | 5 | 5 | 5 | 10 | - | - | - |
| $P_2$ | 4 | 2 | 4 | - | - | - | - | - |
| $P_3$ | 9 | 2 | 2 | 2 | 2 | 2 | 9 | - |
| **free** | | 3 | 1 | 5 | 0 | 10 | 3 | 12 |

A safe sequence of processes exists $\Rightarrow$ **the system is in a safe state** ☺

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

# Safe states: example 2

There are 12 resources in total available.

| **Process** | **Max. need** | **Acquisition** $T_0'$ | $T_1$ | $T_2$ |
|---|---|---|---|---|
| $P_1$ | 10 | 5 | 5 | 5 |
| $P_2$ | 4 | 2 | 4 | - |
| $P_3$ | 9 | 3 | 3 | 3 |
| **free** | | 2 | 0 | 4 |

No safe sequence of processes exists ⇒ **the system is in an unsafe state** ☹

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

# Prevention

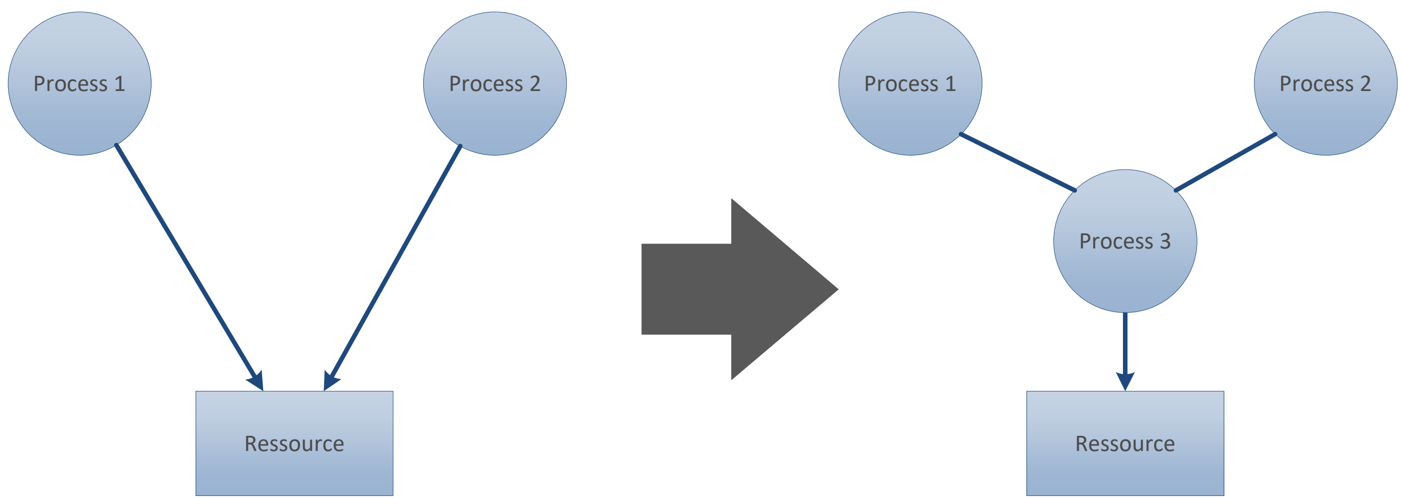
We try to **further investigate the conditions** under these a deadlock can occur in order to **avoid it as much as possible**.

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

# Prevent mutual exclusion

## Idea: try to avoid the mutual exclusion.




### Problems

- Not always possible
- Requires redesign of the application

**CAMPUS Rosenheim**
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

# Prevent non-preemption

**Idea: try to avoid the non-preemption.**

Release all resources if a required resource isn't immediately available.

## Problems

- With the release of a resource, often the whole work is lost and has to be repeated.
- It can happen that a process has to release the resources very often until it can complete its work.

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

# Prevent hold-on-wait

**Idea: try to avoid the hold-on-wait.**

A process requests all resources at startup. If even one resource is not available, the process has to wait.

## Problems

- Often a process doesn't know at startup which resources it needs.
- This wastes resources, because they are acquired, but by no means used.
- It can happen that a process has to try it very often until it can acquire all resources.

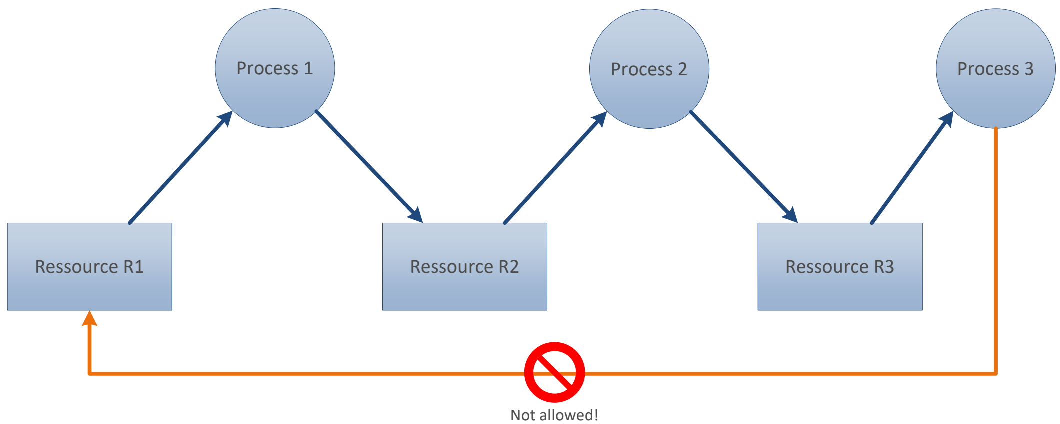
**CAMPUS Rosenheim**
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

# Prevent circular wait

## Idea: try to avoid circles.

All resources are numbered. If a process has already acquired resource $i$, it can only acquire resource $k$ (if $k > i$).




## Problems

- Finding an order in which everyone is satisfied is not easy and not always possible.
- Some resources are created and removed at runtime. Numbering not possible.
- But: With this it is possible to proof the **deadlock free** acquisition: **All processes have to acquire the resources in the same order.**

**CAMPUS Rosenheim**
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

# Prevention

## The banker's algorithm

- For every resource that is acquired by a process it checks if the system stays in the safe state.
  - **True**: The **resource is given** to the process.
  - **False**: The **resource is not given** to the process.

Cost: $C = num\_resources * num\_processes^2$

**CAMPUS Rosenheim**
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

# Deadlock recovery

## What can we do if we are in an **unsafe** state with a deadlock?

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

# Deadlock recovery

## Deadlock recovery strategy

- Terminate one or more processes involved in the deadlock.
- Inform the system operator and give him instructions how to proceed (e.g. manual restart).
- Watchdog: automatically restart processes/device.

source (compare): `https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/7_Deadlocks.html`

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

# Summary and outlook

## Summary

- Deadlock intro
- Deadlock analysis
- Deadlock safe states
- Deadlock prevention
- Deadlock recovery

## Outlook

- Scheduling theory
- Scheduling algorithms