Collaboration of Tasks

Tamás Kovácsházy, Phd

14th topic
Deadlocks

Méréstechnika és
Információs Rendszerek
Tanszék

Budapest University of Technology and Economics
Department of Measurement and Information Systems

Deadlock

- Deadlock is one of the most serious error can happen (caused by programmers actually) in parallel programs
- Exact definition:
  - An H subset of tasks of a system is in deadlock, if all tasks in the H subset wait for such event, that can be only created by other tasks in the H subset
  - The H subset is only a subset of all tasks in the system, not all tasks are influenced by the deadlock

Typical example

- 2 resources, A and B.
- 2 tasks want to use them the following way

```
P1 task:
Acquire(A);
Acquire(B);
        use A and B;
Release(A);
Release(B);
```

```
P2 task:
Acquire(B);
Acquire(A);
        use A and B;
Release(B);
Release(A);
```

Typical example

- 2 resources
- 2 tasks war

Race condition between P1 and P2

```
P1 task:
Acquire(A);
Acquire(B);
      use A and B;
Release(A);
Release(B);
```

```
P2 task:
Acquire(B);
Acquire(A);
      use A and B;
```

Which one runs first?

Typical example

Acquiring resources in the same order the deadlock cannot occur!

Most cases the names of the resources are not so easy to order, so the solution is not so straightforward…

```
P1 task:
Acquire(A);
Acquire(B);
        use A and B;
Release(A);
Release(B);
```

```
P2 task:
Acquire(A);
Acquire(B);
        use A and B;
Release(B);
Release(A);
```

Deadlock in real life

- There are very complex cases compared to the previous simple demonstration
- Hard to identify...
  - It manifests itself in the form of a race condition
    - It works sometimes, and sometimes it ends up in a deadlock…
    - Hard to reproduce the problem and correct it
  - Certain hard to reproduce conditions must be met to recreate the deadlock
- The deadlock may influence other tasks also:
  - E.g. tasks in the H subset may reserve and never free (freeze) other resources in the system limiting the system

Necessary conditions

1. Mutual Exclusion

   There are shared resources in the system to be used by more tasks than the resource can handle concurrently without errors

2. Hold and Wait

   There are tasks in the system that acquires and may wait for additional shared resources while it holds (uses) some other shared resources

3. No resource preemption

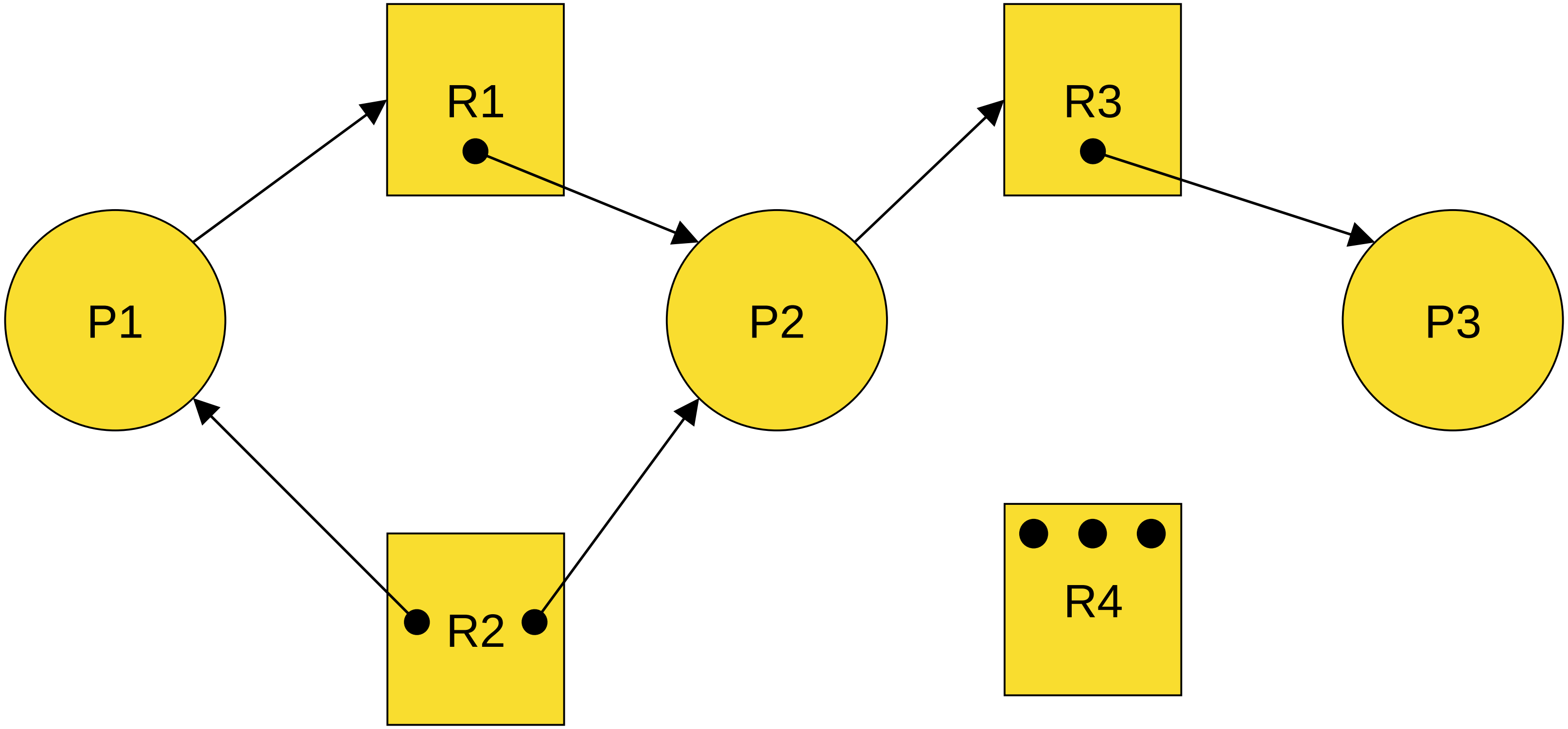
   No task gives up shared resources on request involuntarily, it only releases a shared resource when that is not needed anymore by the task (the exception is the CPU in preemptive schedulers, but there preemption is handled properly)

4. Circular Wait

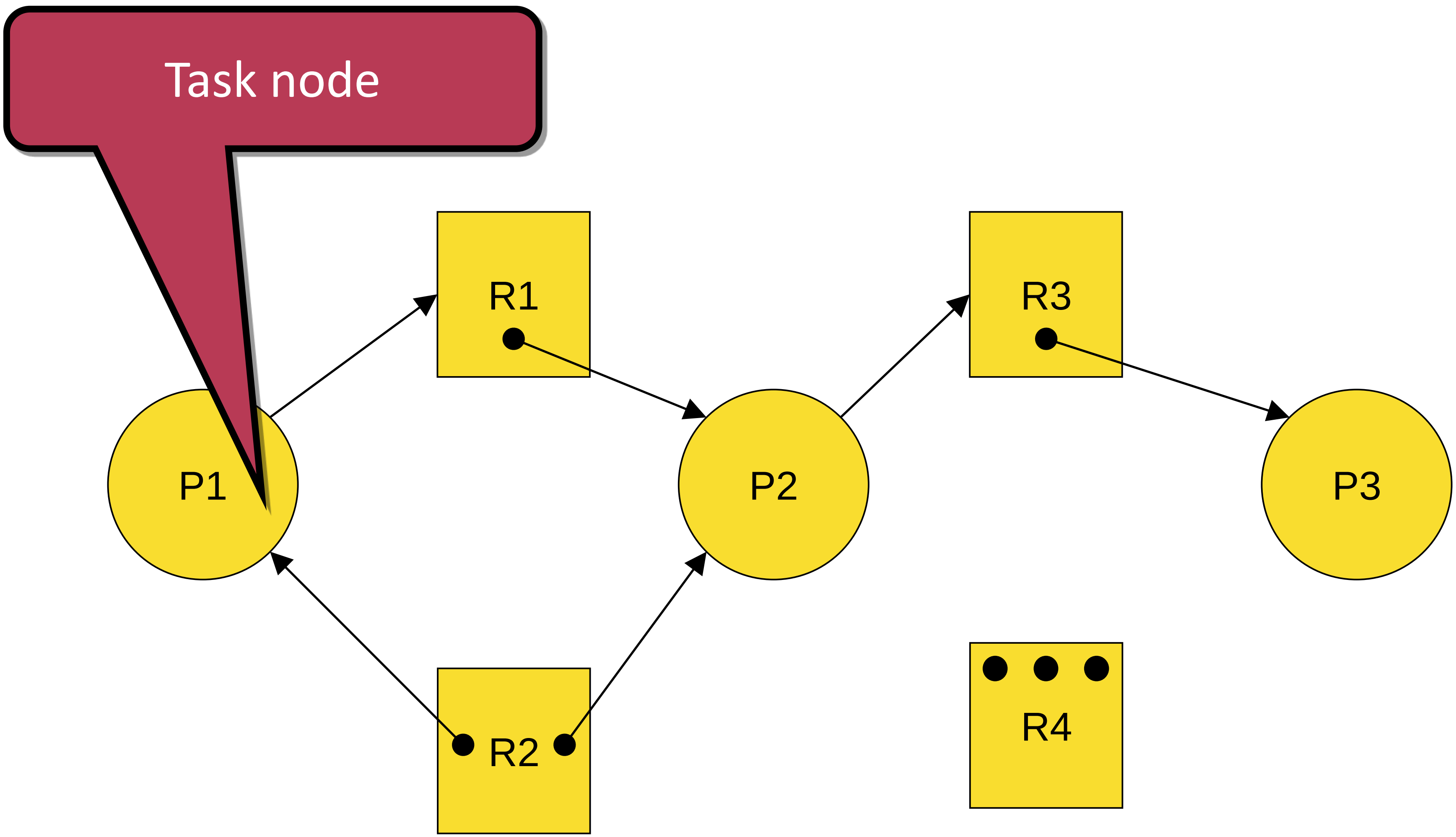
   There is a {P0, P1, ... Pn} list of tasks in the system, for witch Pi waits for Pi+1 ($0 \leq i < n$), and Pn waits for P0

Resource reservation graphs 1.

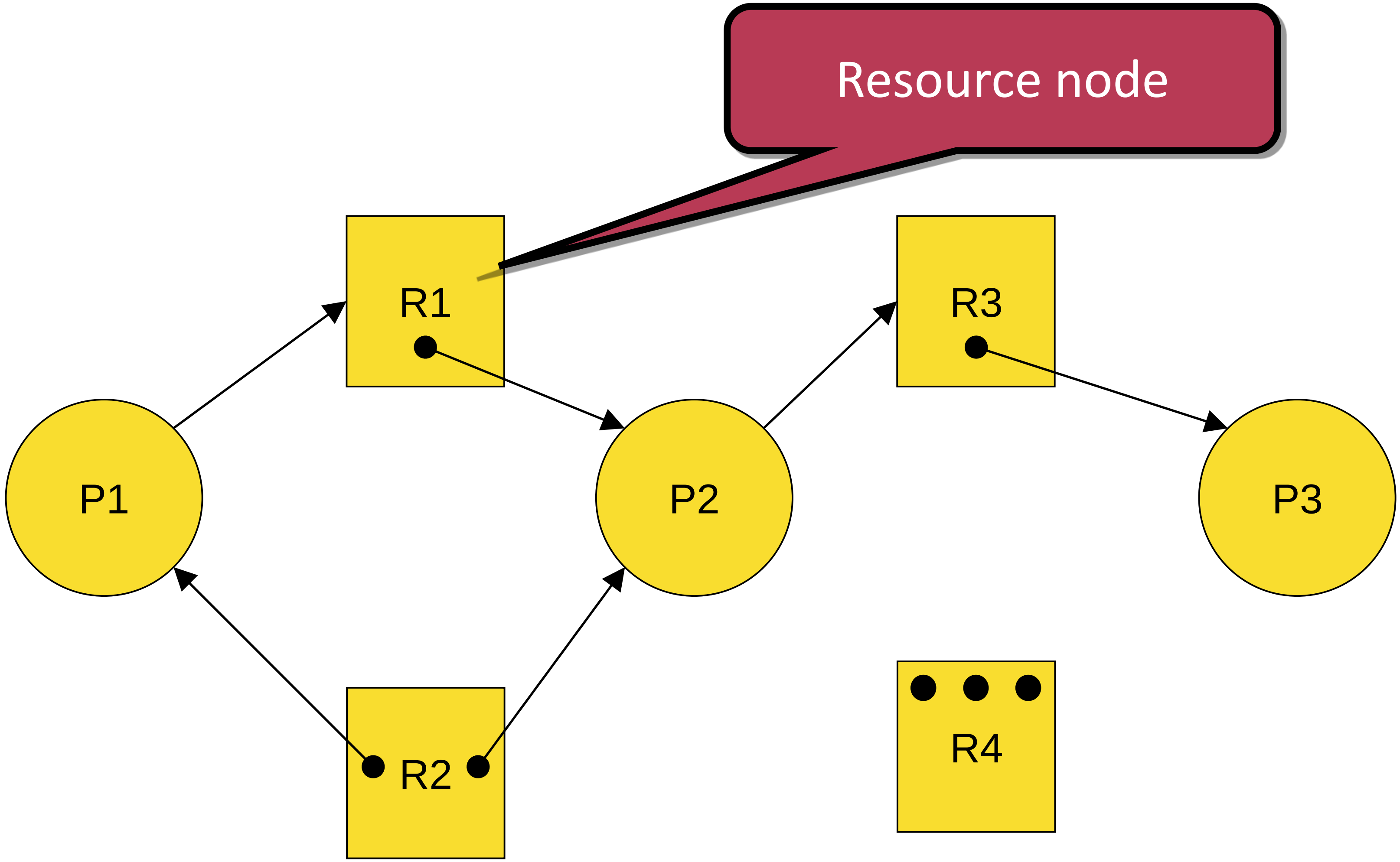
- Resource-allocation graph to visualize the situation…



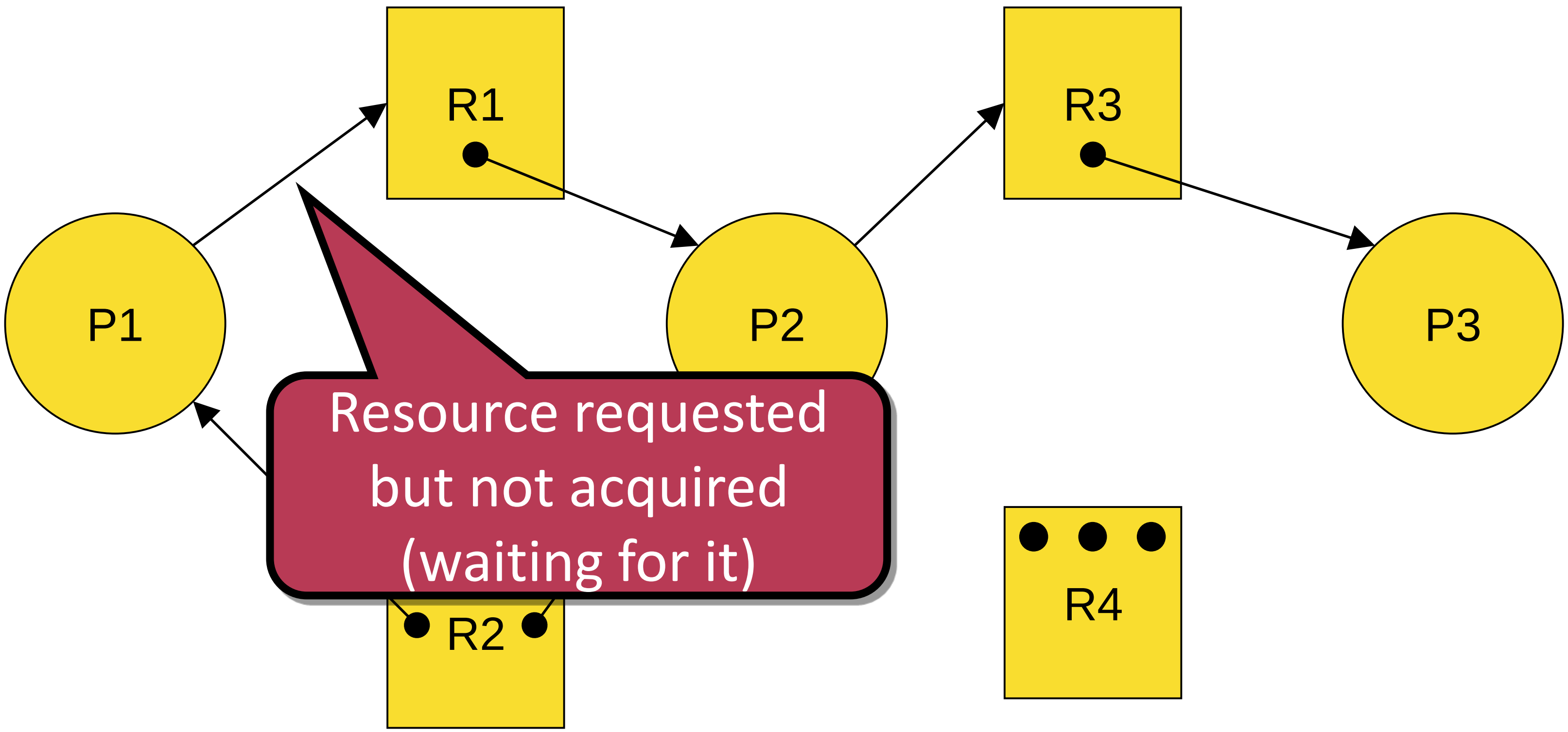
Resource reservation graphs 2.



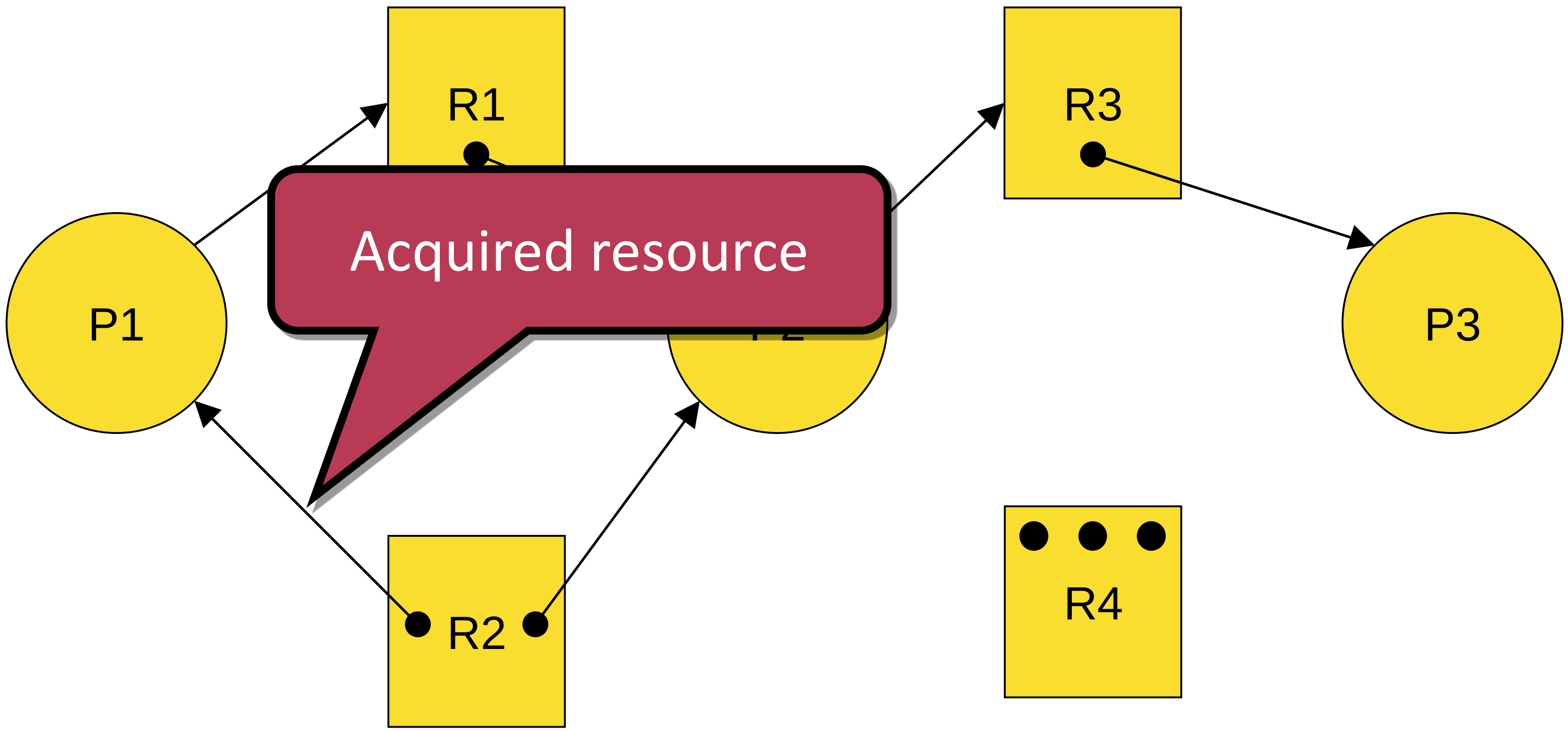
Resource reservation graphs 3.



Resource reservation graphs 4.



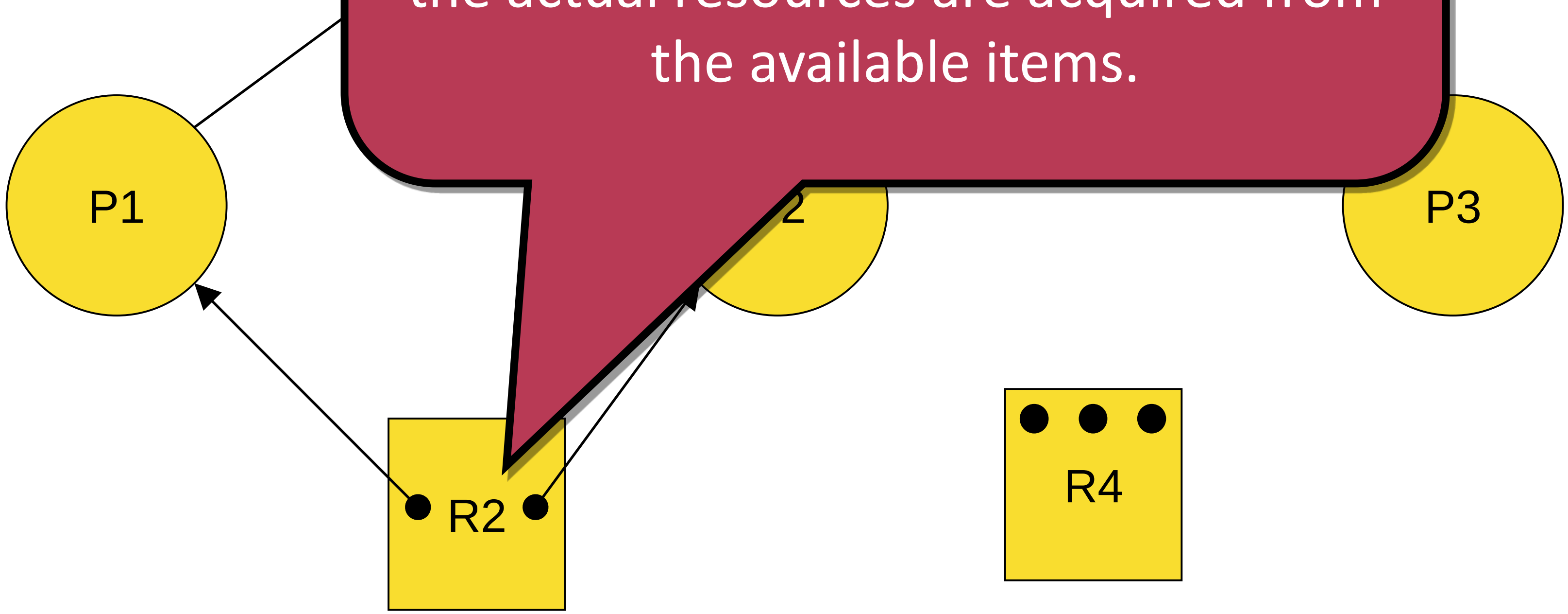
Resource reservation graphs 5.



Méréstechnika és Információs Rendszerek Tanszék

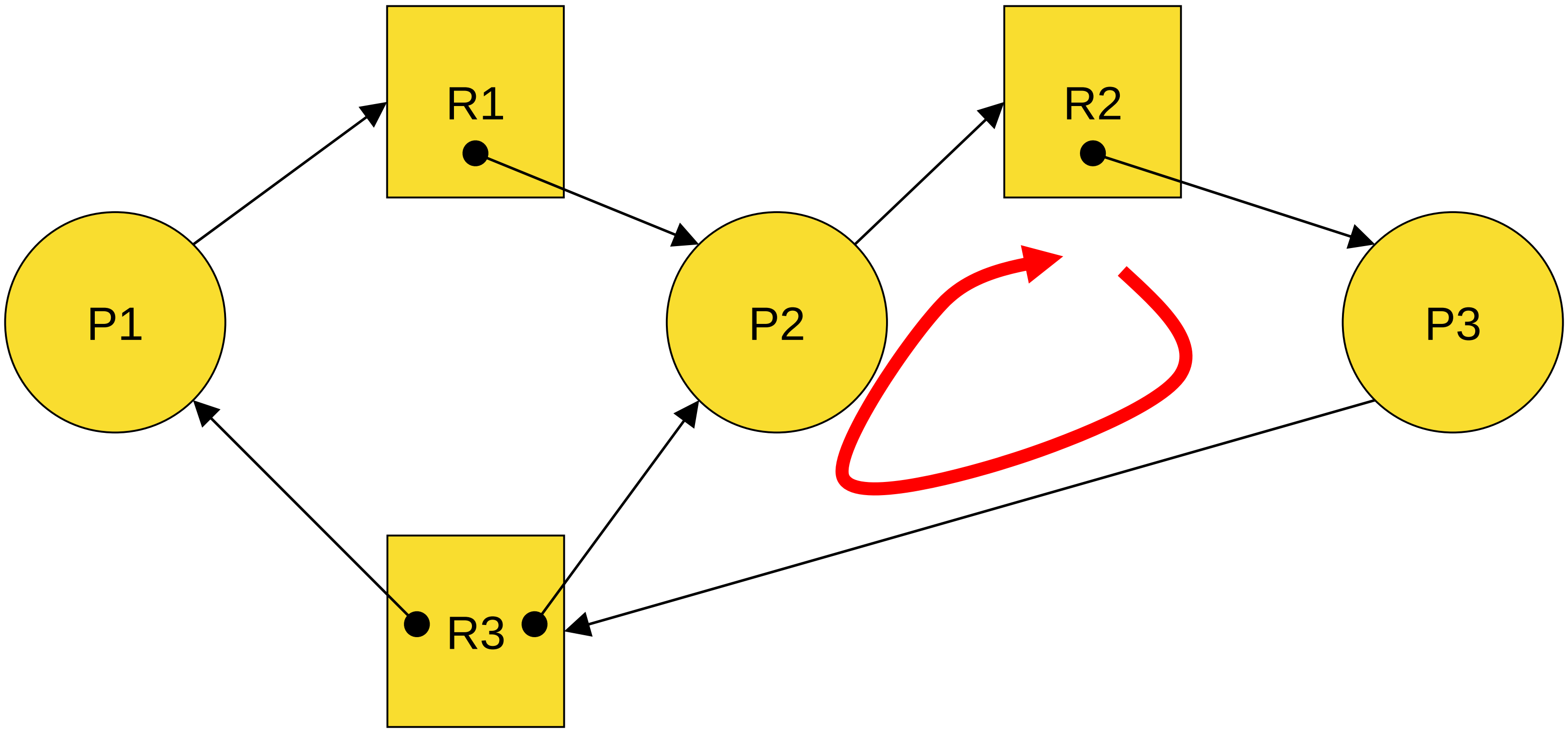
Resource reservation graphs 5.

The number of black dots constitutes the number of resources available.
For the task it is arbitrary which one of the actual resources are acquired from the available items.



How deadlock are identified?

- Directed loop on the graph (4th necessary condition).
- It can be a deadlock, but it may or may not happen (only the possibility is shown)



Handling of deadlock

- Ostrich algorithm (we do not care about the possibility)
  - Classic joke: How car would handle malfunctions if cars were designed by Microsoft?
  - Unfortunately, due to the huge software content of modern cars the joke is reality!
    - New Windows operating systems are much better from the point of view of reliability, especially if you take configurability and complexity into account
  - In non mission critical systems it is a solution (let's restart this…)
- Deadlock detection and recovery
- Deadlock prevention
  - Structural prevention of deadlock
  - Analyses of the solution
- Deadlock avoidance

Deadlock detection and recovery 1.

- Detection:
  - The cases of single item resource and multiple item resource must be handled differently
  - There are large number of published algorithms
  - Single item resource: Wait-for graphs
    - You can capture which one waits...
  - General multiple item resource : Coffman's deadlock detection algorithm (We do not have time to introduce it).
  - When the algorithm should run?
    - One extremity : For any resource request cannot be granted immediately (may cause a deadlock)
    - The other extremity : Periodically in case of low CPU usage when lot of tasks wait for resources (this situation may be caused by a deadlock)
  - Algorithms usable in real life are CPU intensive
    - The should be executed rarely (the second extremity is realistic).

Méréstechnika és Információs Rendszerek Tanszék

Deadlock detection and recovery 2.

- Recovery:
  - Multiple solutions
    - The radical one : All tasks in the deadlock are terminated
    - The humane one: Only selected tasks are terminated, a decision must be made, and the success is not guaranteed
  - Tasks to do:
    - Selecting a victim
    - Rollback, if there exist any intermediate safe state to roll back the resource to
  - In case of consecutive deadlocks we should pick different victims
    - Avoiding starvation , not to end up in a higher level livelock

Deadlock prevention 1.

- Design time solution
- A deadlock free system is developed
  - At least one of the necessary conditions of deadlock cannot happen
    - Mutual exclusion is a precondition in this environment
- 1th: No run-time resource reservation:
  - Drastic, but it is possible in some simple embedded systems
- 2nd: No Hold and Wait:
  - A task holding a resource request an another resource
  - All resources are reserved by one system call
  - It utility highly depends on the application, i.e., how resources must be handled
  - Resource utilization is decreased (sometimes drastically)

Méréstechnika és Információs Rendszerek Tanszék

Deadlock prevention 2.

- 3rd : Resource preemption:
  - The resource must have a state to which it can be rolled back
  - No rollback is required on the task level
- 4th : No circular waiting:
  - Restricting the resource usage during development
  - E.g. Total ordering algorithm
    - Programmers must adhere the algorithm
    - Automatic checking of the source code may enhance the situation by identifying errors of programmers
  - Formal methods
    - A formal model of the program is constructed and checked

Deadlock avoidance

- Run-time method
- Resources are not automatically granted, but the effect of granting the resource is reconsidered:
  - Is the system is going to be in a safe state if the resource granted?
  - Banker's algorithm (Dijsktra, 1965)
    - Old school bankers used this algorithm to allocate resources to clients with long term projects
    - To allocate resources in way that allows clients to be financed in the long term from reimbursements
    - The modern bankers used a quite different algorithm and the last financial crisis let us to see it in detail with its devastating effects… ☹

Banker's algorithm 1.

- N tasks, M resource types
- The resources are available in multiple numbers
- Tasks announce the maximum number of resource items they use from a given resource:
  - MAX matrix, size: NxM,
- The resources actually reserved by tasks
  - RESERVED matrix, size: NxM
  - Can be computed by the incoming requests
- Number of free resources:
  - FREE vector, size: M

Banker's algorithm 2.

- MAXr stores the number of available resources
- RESERVEDr stores the number of reserved resources
- The number of possible resource reservation that can be submitted:
  - LEFT = MAX-RESERVED a matrix with the siye NxM
- Reservation request waiting are stored in
  - REQUEST with the size of NxM

Banker's algorithm in operation 1.

- The algorithm is detailed in the book
  - I am going to show you only an example
- In one iteration (for one resource reservation request) there are four steps in the algorithm

  1st step: Check if there is enough resource

  2nd step: Set the state

  3rd step: Check if the state in step 2 is safe

  4th step: If the system is not safe it rolled back to a safe state

Banker's algorithm in operation 2.

- Checking if the state is safe

  1st step: Setting initial state

  2nd step: Searching for task with the possibility of continuing of the resource reservation

  - The maximum request can be reserved for the task with the available resources
  - If yes, the task can get the resources, it can run, free the resources after usage, and those resources can be reclaimed into the FREE pool
  - If there is a task to run, we can continue with step 2

  3rd step: Evaluation

  Pi is the list of tasks that can end up in a deadlock

Banker’s algorithm excersize

- A system has 4 resource classes (A, B, C and D), and there are 10, 11, 7, and 10 resources maximum available from these resource classes. There are 5 tasks competing for these resources in the system with the following actual reservation and maximum required resource numbers.

	Maximum required				Actual reservation			
	A	B	C	D	A	B	C	D
P1	2	2	5	4	0	2	3	3
P2	7	7	3	4	3	1	2	2
P3	5	6	6	4	2	2	0	2
P4	4	1	2	3	2	1	2	2
P5	6	3	1	1	1	3	0	0

- The system uses the Banker’s algorithm to avoiud deadlocks. Is the system in a safe state? If your answer is yes, show how the tasks can finish their work. If your answer is no, show how deadlock may form in the system.

Solition 1.

- LEFT = MAX-RESERVED
- FREE = MAXr-RESERVEDr

$$LEFT = \begin{bmatrix} 2 & 2 & 5 & 4 \\ 7 & 7 & 3 & 4 \\ 5 & 6 & 6 & 4 \\ 4 & 1 & 2 & 3 \\ 6 & 3 & 1 & 1 \end{bmatrix} - \begin{bmatrix} 0 & 2 & 3 & 3 \\ 3 & 1 & 2 & 2 \\ 2 & 2 & 0 & 2 \\ 2 & 1 & 2 & 2 \\ 1 & 3 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 2 & 1 \\ 4 & 6 & 1 & 2 \\ 3 & 4 & 6 & 2 \\ 2 & 0 & 0 & 1 \\ 5 & 0 & 1 & 1 \end{bmatrix}$$

$$LEFT = \begin{bmatrix} 10 & 11 & 7 & 10 \end{bmatrix} - \begin{bmatrix} 8 & 9 & 7 & 9 \end{bmatrix} = \begin{bmatrix} 2 & 2 & 0 & 1 \end{bmatrix}$$

Solution 2.

- In which of the rows of LEFT has smaller or equal numbers than in FREE?
  - That can run with the currently available resources if it runs alone
- In or example P4 is executable in this sense
  - There is [2 2 0 1] free, and P4 needs only [2 0 0 1]
  - After running P4 resources used by P4 goes back to the FREE pool of resources
  - FREE = [4 3 2 3] after running P4

Solution 3.

- In which of the rows of LEFT has smaller or equal numbers than in FREE?
  - That can run with the currently available resources if it runs alone
- P1 is executable
  - There is [4 3 2 3] free, and P1 needs only [2 0 2 1]
  - After running P1 resources used by P1 goes back to the FREE pool of resources
  - FREE = [4 5 5 6] after running P1

Solution 4.

- In which of the rows of LEFT has smaller or equal numbers than in FREE?
  - That can run with the currently available resources if it runs alone
- There is no safely executable task
  - There is [4 5 5 6] free, but P2 needs [4 6 1 2]
    - Cannot run safely because 1 unit is missing from resource B
  - There is [4 5 5 6] free, but P3 needs [3 4 6 2]
    - Cannot run safely because 1 unit is missing from resource C
  - There is [4 5 5 6] free, but P5 needs [5 0 0 1]
    - Cannot run safely because 1 unit is missing from A
- The system is not in a safe state!

Evaluation of the Banker’s algorithm

- The algorithm shows the possibility of deadlock, but the system will not necessarily end up in a deadlock!
  - The actual sequence of resources reservations and freeing resource define if the deadlock develops or not!
  - The algorithm returns the worst case results
  - How we can now matrix MAX?
    - Most cases it is very hard or impossible to know it

How we handle deadlock in practice

- Ostrich algorithm (we do not care about the possibility) is very common unfortunately
  - Not a solution in safety critical systems, but who cares if a media player needs to be restarted 2-3 times a week under Windows…
- Most cases we handle (try to handle) deadlocks in design time, so we attempt to do deadlock prevention
- Deadlock detection and recovery are done by human operators
  - All tasks are killed and resources are rolled back into a safe state (if possible)
- Why?: Run-time algorithms are complex, they need a lot of resources, not all the required data available to run them (MAX in case of the Banker's algorithm), etc.