Institute of Computer Science
Department of Mathematics and Computer Science

Algorithms and Programming IV
Deadlocks

Summer Term 2023 | 15.05.2023

Barry Linnert

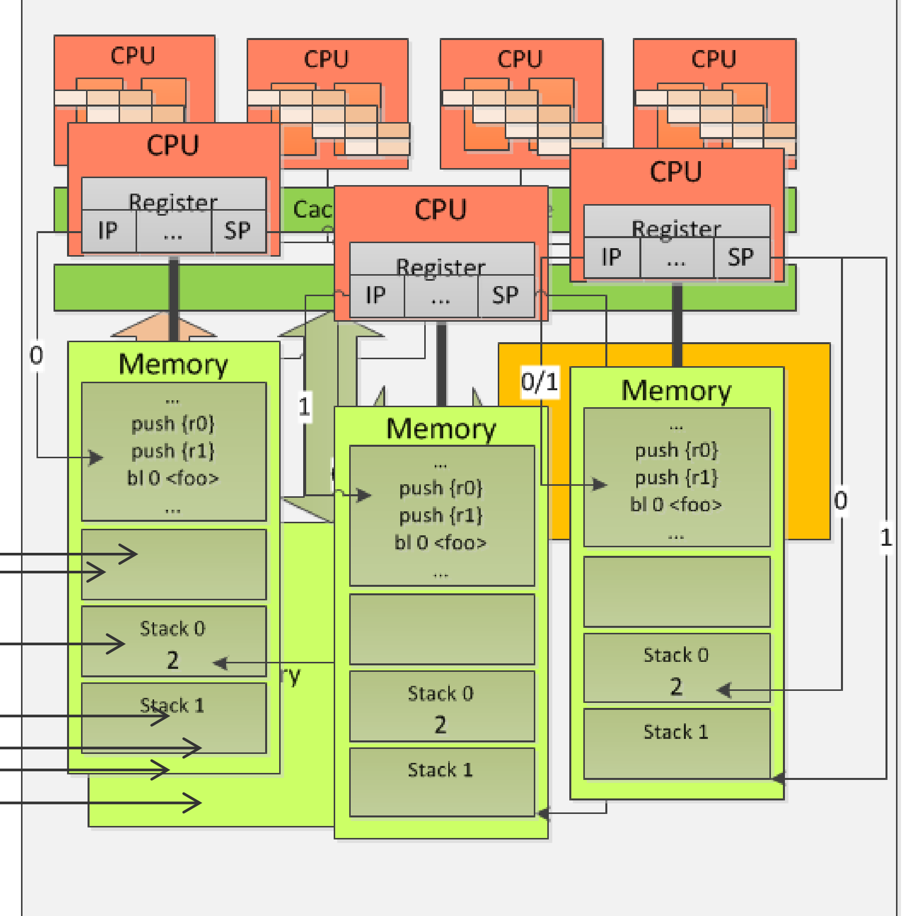
Objectives of Today‘s Lecture

- Defining Deadlocks
- Requirements for a Deadlock
- Handling Deadlocks
- Modelling Resource Allocations
- Banker's Algorithm

concepts of non-sequential and distributed programming

RECAP

Machine and Execution Model



Instruction Pointer
Instruction Pointer
Instruction Pointer
Instruction Pointer
Instruction Pointer
Instruction Pointer
Instruction Pointer

Correctness

- Correct implementation of commands and functions
  - Compiler/Interpreter, HW

- Correct execution of the set of commands
  - Sequential processing of the operations of the critical section by lock variables (lock/mutex) using the operating system and hardware.
  - Programming model and machine model (execution model) correspond to each other

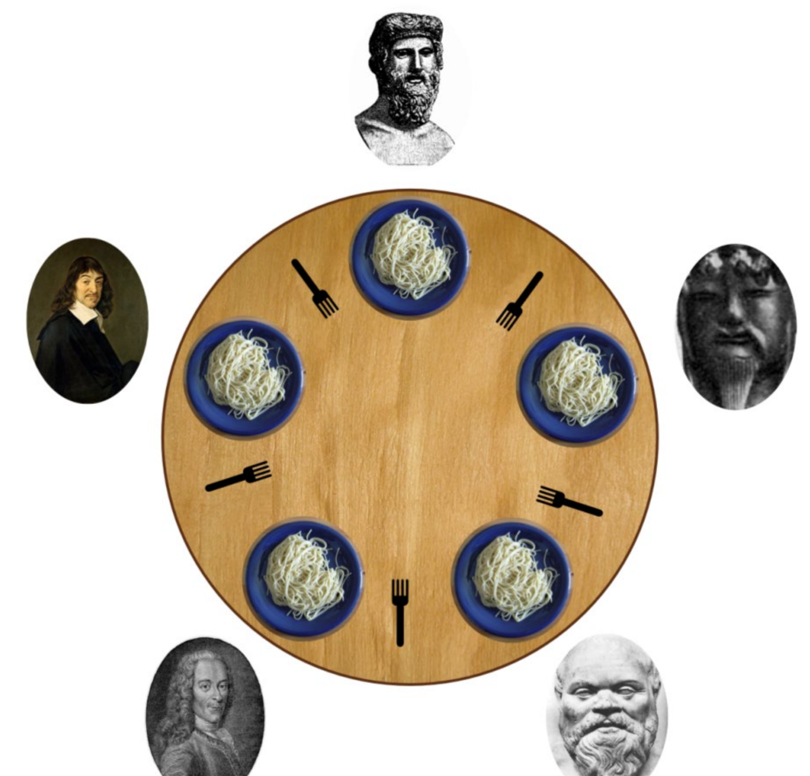
- Check with
  - Hoare calculation
  - Testing

Requirements for Programs

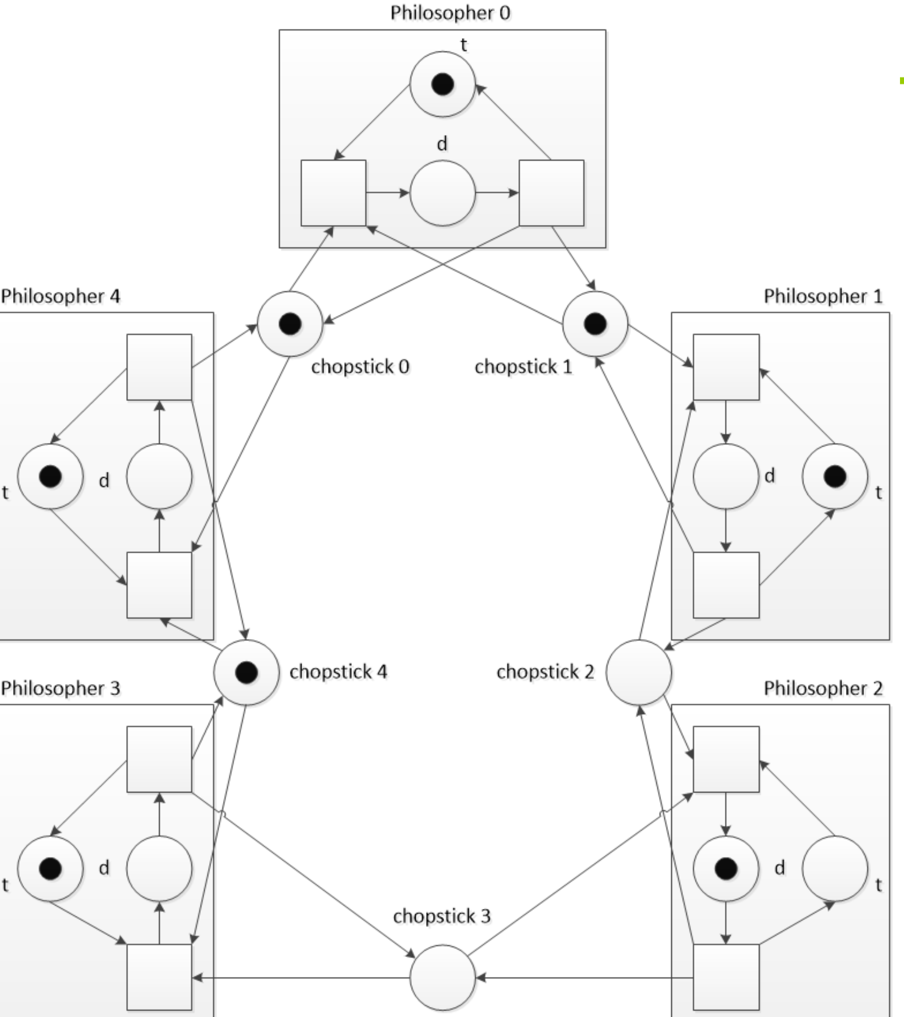
- The program should do what it has been programmed to do!
  - Functional properties
    - Scope of functions
    - Correctness

- The program should follow certain rules for this purpose!
  - Non-functional properties
    - Performance
    - Security
    - …

Example: Dining Philosophers



wikipedia.org



Philosopher 0
t
d
Philosopher 4
Philosopher 1
chopstick 0
chopstick 1
t
d
d
t
Philosopher 3
chopstick 4
chopstick 2
Philosopher 2
t
d
d
t
chopstick 3

Concepts of Non-sequential and Distributed Programming

DEADLOCKS

Examples

Locking

T_1	T_2
LOCK(LO1)	LOCK(LO2)
LOCK(LO2)	LOCK(LO1)

Signaling

P_1	P_2
WAIT(SO_1)	WAIT(SO_2)
SIGNAL(SO_2)	SIGNAL(SO_1)

Communication

P_1	P_2
RECEIVE_S(CO1)	RECEIVE_S(CO2)
SEND_A(CO2)	SEND_A(CO1)

Deadlock in Communication

- Communication of more than one processes can lead to a deadlock.

R R R R R

S S S S S

Deadlock in everyday Life

waits for

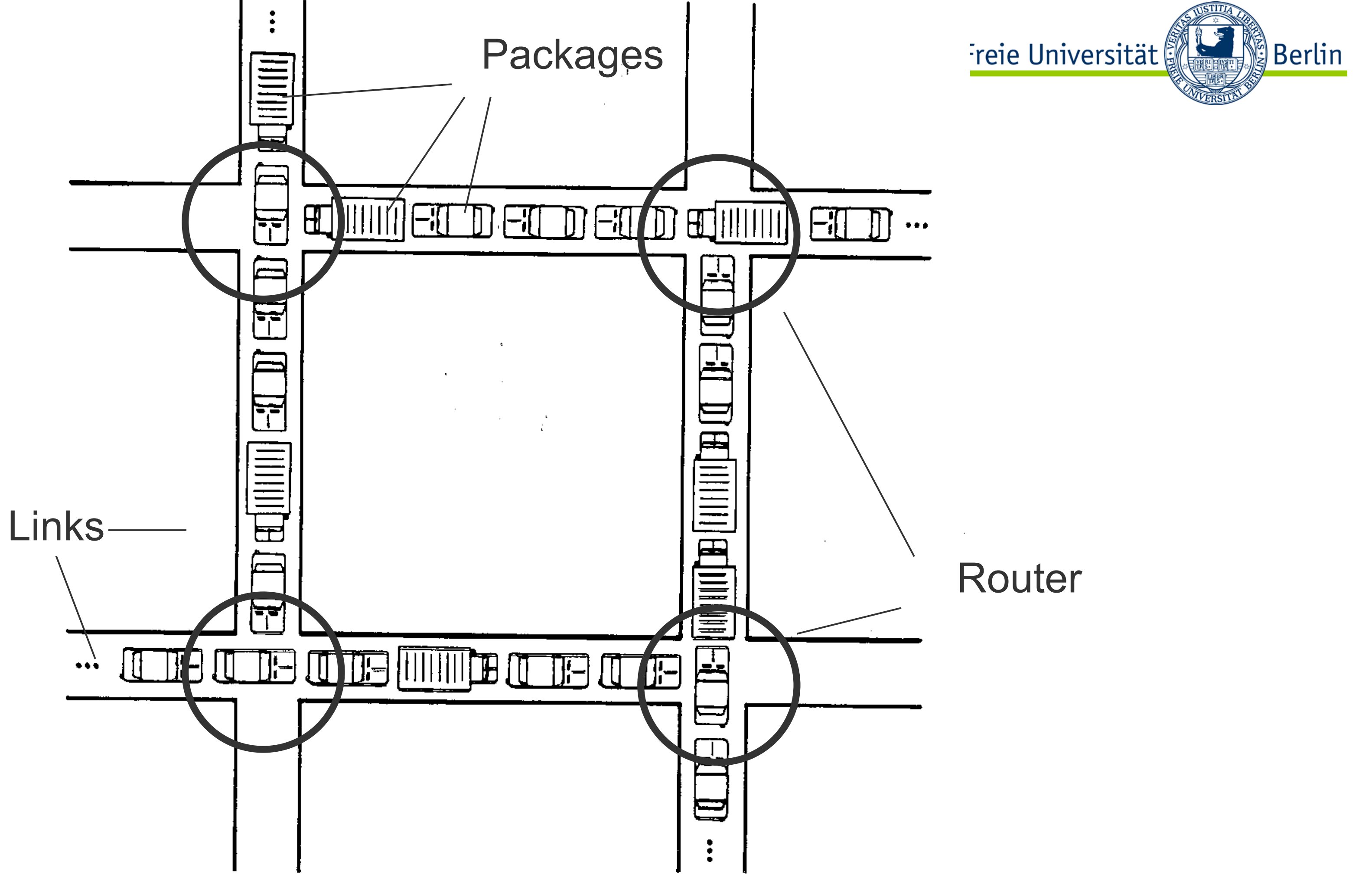
waits for

waits for

waits for

Betriebs-
systeme

Deadlock Resolution?


Packages
Links
Router

Resources

- The term ***resource*** covers every item that a thread or process needs for it‘s execution or processing.

- Resources come in many different forms. The specific exemplar of a resource usually can be identified by a name.

- Resource may be limited or unlimited. Usually resources do have some limit and are needed by more than one thread or process.

- Thus, resources can lead to problems if they are limited, or if they can used exclusively only.

Examples of Resources

Example 1:

- A thread needs the program code to be executed accessible in main memory.
- The program code is a resource of the thread.
- Other threads may execute the same program code. These threads can access the code as well. There is no need to manage the access to the resource.

Example 2:

- Threads need main memory to store some data.
- Memory is limited and should be assigned exclusively. Therefore the memory must be managed.

Example 3:

- Threads of a process must access a common variable (shared memory).
- The access to the variable is a critical section and therefore protected.
- Access to the critical section is itself a resource.

Wait-for Graph

- A directed graph with the threads or processes as nodes and wait relations as directed edges is called *wait-for graph*.
- A cycle in graph indicates a deadlock.

with two threads/processes

waits for

T1

T2

waits for

with six threads/processes

T1

T2

T6

T3

T5

T4

Deadlock

- Deadlocks can occur in different situations.
- In context of resource management these three **requirements are necessary** for a deadlock:
  1. Resources are used exclusive.
  2. Threads or processes hold allocation of a resource and try to allocate another.
  3. There is no preemption.

  With these requirements fulfilled the following constraint may occur and together with the first requirements this is **sufficient** for a deadlock:

  4. There is a cycle in the wait-for graph.

Handling Deadlock

- To deal with a deadlock different counteractions have to be provided:
  - **Prevention**
  - **Avoidance**
  - **Detection**
  - **Resolution, Recovery**

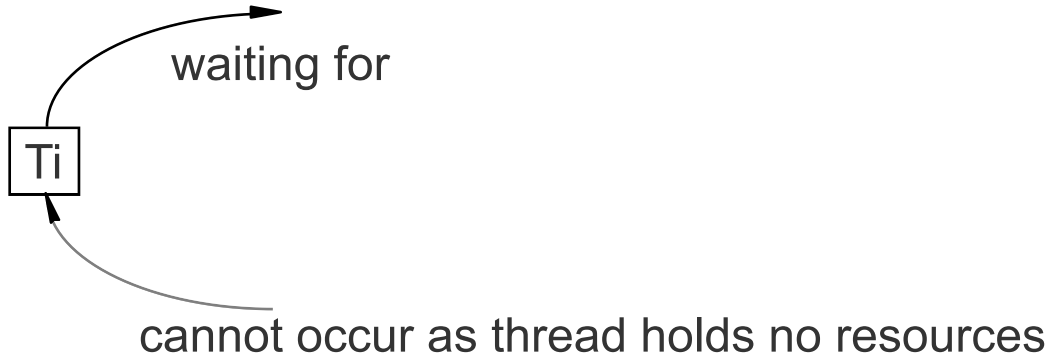
Deadlock Prevention

- Prevention stands for a methodic procedure handle resource assignment restrict in a way no deadlock can occur.

- Approaches for deadlock prevention are:
  - Pre-claiming
  - Overall release at request
  - Allocation by (given) order

Deadlock Prevention: Pre-claiming

- All resources needed by a thread are requested (and allocated) at start time.



- Deadlock requirement 2 cannot occur.
- In dynamic systems the overall demand is difficult to predict.
- It's an uneconomical approach as resources are occupied longer than needed.

Deadlock Prevention: Overall Release at Request

Allocate (Res_1, Res_2)

Release (Res_1, Res_2)

Allocate (Res_3)

eventual forced release

Release (Res_3)

Allocate (Res_1, Res_2)

Deadlock Prevention: Overall Release at Request II

- Deadlock requirement 2 cannot occur.
- As the thread does not possess any resource at allocation the cycle in wait-for graph is prevented, too.

Deadlock Prevention: Allocation by (given) Order

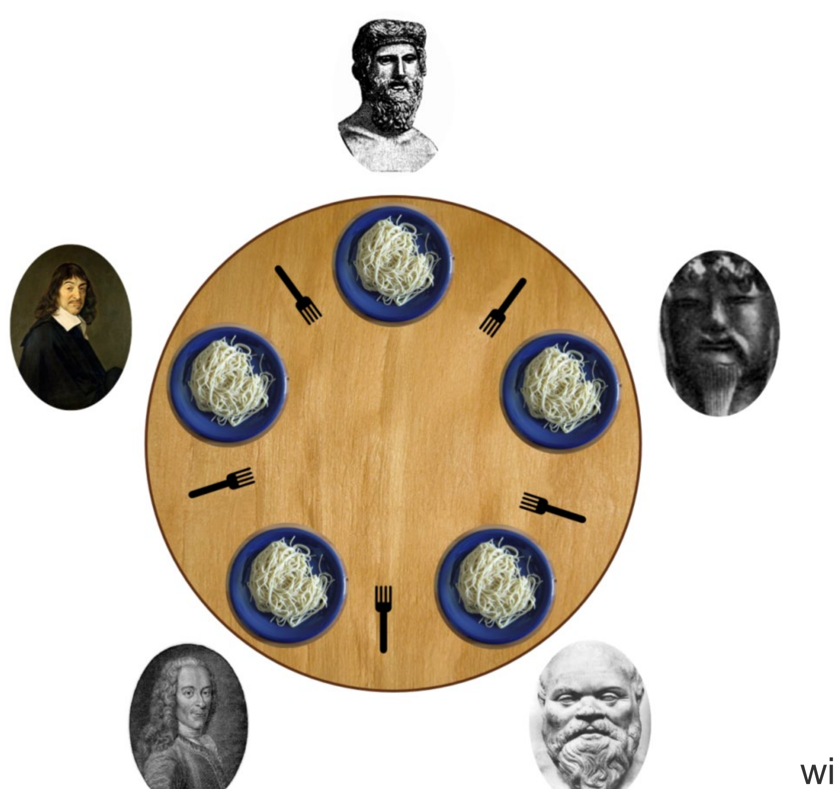
- Resource are sorted (Res_1, Res_2, Res_3, …).
- Resource allocation is performed in order only.

- Deadlock requirement 4 cannot occur.
- With this approach cycles in the wait-for graph are prevented successfully.

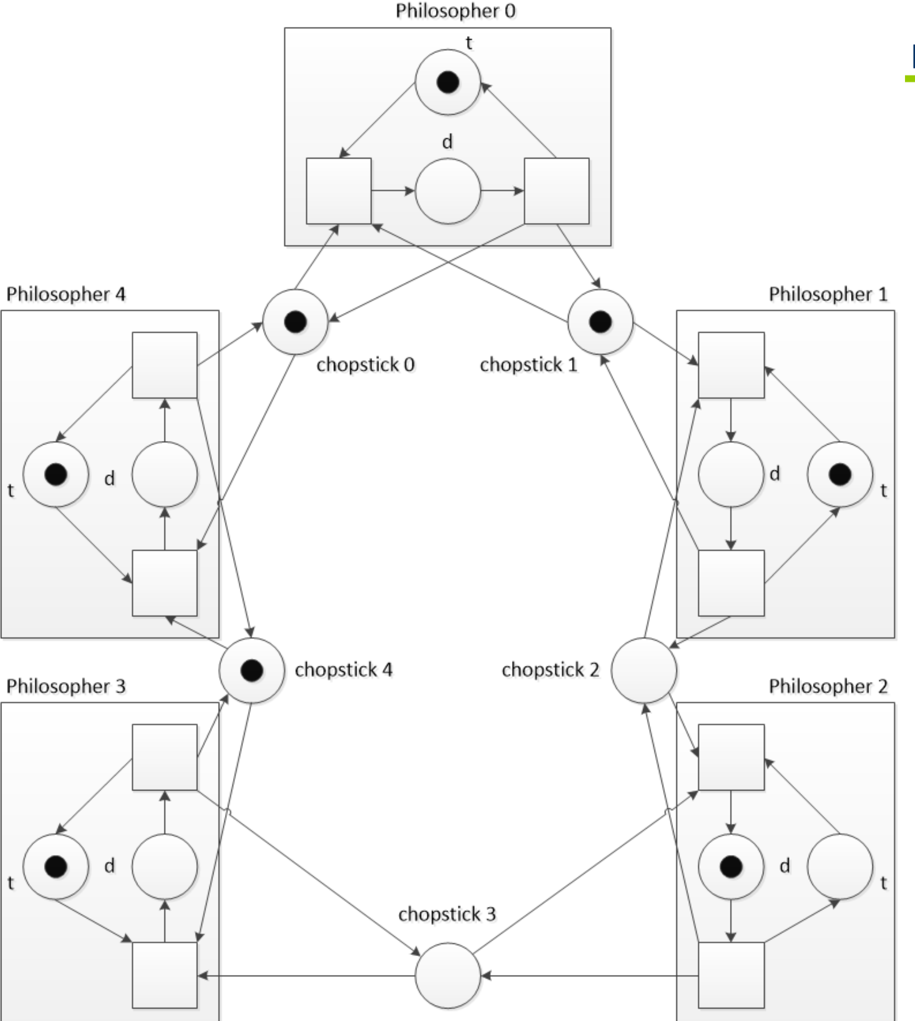
T1	T2
Allocate (Res_1)	
	Allocate (Res_2)
	Allocate (Res_3)
Allocate (Res_3)	
	Allocate (Res_5)
Allocate (Res_4)	

Concepts of Non-sequential and Distributed Programming

MODELLING OF RESOURCE ALLOCATION



wikipedia.org



Philosopher 0
t
d
Philosopher 4
chopstick 0
chopstick 1
Philosopher 1
t
d
d
t
chopstick 4
chopstick 2
Philosopher 3
Philosopher 2
t
d
d
t
chopstick 3

Depiction of Resource Allocation

- T — Set of threads (or processes), |T| = m
- Rs — Set of resource typs, |Rs| = n
- $\vec{v} := (v_1, v_2, \ldots, v_n)$ — available resources

- Allocated (busy) resources:

$$B := \begin{pmatrix} b_{11} & \ldots & b_{1n} \\ \vdots & & \vdots \\ b_{m1} & \ldots & b_{mn} \end{pmatrix}$$

Requests for resources (asked for):

$$A := \begin{pmatrix} a_{11} & \ldots & a_{1n} \\ \vdots & & \vdots \\ a_{m1} & \ldots & a_{mn} \end{pmatrix}$$

Depiction of Resource Allocation

- Overall requests – maximum of all requests (global requests):

$$G := \begin{pmatrix} g_{11} & \cdots & g_{1n} \\ \vdots & & \vdots \\ g_{m1} & \cdots & g_{mn} \end{pmatrix}$$

- The Quintuple *(T, Rs,* $\vec{v}$*, B, A)* represents current resource allocation.
- The current resource allocation is depicted completely.

Constrains

1. $\forall j \in \{1,\ldots,n\} : \sum_{i=1}^{m} b_{ij} \le v_j$ no more than available resources can be allocated.

2. $\forall i \in \{1,\ldots,m\} \ \forall j \in \{1,\ldots,n\} :$ $$a_{ij} + b_{ij} \le v_j$$ $$g_{ij} \le v$$

   Requests can only contain amount of resource available.

3. Requesting threads or processes are blocked until allocation.

4. Only non-blocked threads or processes can request another resource (see 3.).

Additional Identifier

- Free resources

$$\vec{f} := (f_1, f_2, \ldots, f_n)$$

with

$$f_j := v_j - \sum_{i=1}^{m} b_{ij}$$

free res. = (existing) res. – allocated res.

- Remaining requests

$$R := \begin{pmatrix} r_{11} & \cdots & r_{1n} \\ \vdots & & \vdots \\ r_{m1} & \cdots & r_{mn} \end{pmatrix}$$

with

$$r_{ij} := g_{ij} - b_{ij}$$

remaining req. = overall req. – allocated res.

Notation

- Row vector instead of matrix
  - $\overrightarrow{a_i} := (a_{i1}, a_{i2}, \ldots, a_{in})$ request of thread *i*
  - $\overrightarrow{b_i} := (b_{i1}, b_{i2}, \ldots, b_{in})$ allocated resources of thread *i*
  - $\overrightarrow{g_i} := (g_{i1}, g_{i2}, \ldots, g_{in})$ overall request of thread *i*
  - $\overrightarrow{f_i} := (f_{i1}, f_{i2}, \ldots, f_{in})$ remaining request of thread *i*

Notation II

- Relational operators
  - $\vec{x} \leq \vec{y} \Leftrightarrow \forall k : x_k \leq y_k$
  - $\vec{x} \nleq \vec{y} \Leftrightarrow \exists k : x_k > y_k$

Definitions

- A thread T_i is blocked if $\vec{a}_i \not\leq \vec{v} - \sum_{k=1}^{m} \vec{b}_k = \vec{f}$, the current request cannot be satisfied.
- Set of threads $T = \{T_1, T_2, \ldots, T_m\}$ is in a deadlock if in any point in time from now on $\exists I \subseteq \{1,2,\ldots,m\} : \forall k \in I : \vec{a}_k \not\leq \vec{v} - \sum_{k \in I} \vec{b}_k$, there is a subset of threads whose requests could not be satisfied by the amount of resource not allocated by the threads of T.
- The subset of all threads defined by I $\{T_i \mid i \in I\}$ is also called to be in a deadlock.

Example

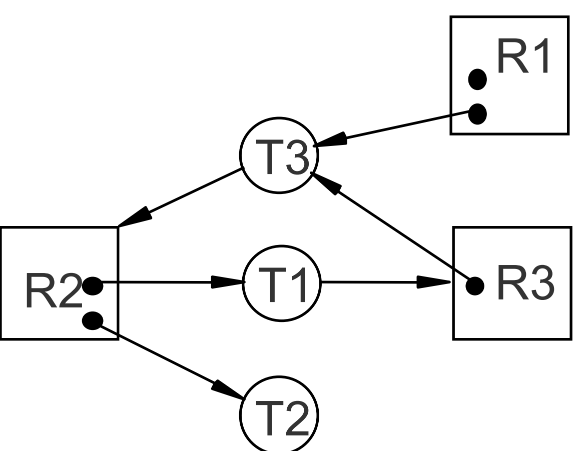
- Set of threads: $I = \{1,2\},\ n = 2$
- Available resources: $\vec{v} = (4,4)$
- Resources assigned to thread 1: $\vec{b}_1 = (2,3)$
- Resources requested by thread 1: $\vec{a}_1 = (0,1)$
- Resources assigned to thread 2: $\vec{b}_2 = (1,1)$
- Resources requested by thread 2: $\vec{a}_2 = (2,0)$
- Free resources: $\vec{v} - \sum_{k \in I} \vec{b}_k = (4,4) - (2,3) - (1,1) = (1,0)$
- Check if requests can fulfilled for thread 1: $\vec{a}_1 = (0,1) \not\leq (1,0)$
- Check if requests can fulfilled for thread 2: $\vec{a}_2 = (2,0) \not\leq (1,0)$
- According to the previous definition, the set of threads $\{T_i \mid i \in I\}$ is in a deadlock.

Modeling of Resource Allocation with Resource Graph

- With resource graph the request and allocation status can be formally described.
- T is set of threads, Rs is set of resource types.
- A resource graph is a directed graph *(V, E)* with $V = T \cup Rs$ and
  - $(t, r) \in E \Leftrightarrow$ Thread t make a request for a unit of resource type r
  - $(r, t) \in E \Leftrightarrow$ Thread t holds allocation of a unit of resource type r

Resource Graph

- Resource graph is bipartite regarding *T* (cycles) and resources *Rs* (rectangles), so there are edges between *T* and *Rs* only.
- The number of units provided by a resource type is depicted as node weight (points within the rectangle) and determines the max. number of units that can be allocated by threads.



Resource Graph: Properties and Operations

- A resource graph depict a specific status of system, shows the current resource status.
- It is comparable to the wait-for graph a cycle points to a deadlock condition.
- But in difference to the wait-for graph with a cycle in the resource graph there is no necessity to have a deadlock, but it is necessary to have such a cycle to get one.
- Every operation of resource management (request, allocation, release) implies a transformation of the graph (add or remove edge).
- A thread only can perform such an operation if it’s not blocked.
- In case all of the remaining requests can be fulfilled the thread may finish successfully (termination).
- With termination of a thread all allocated resources are being released.

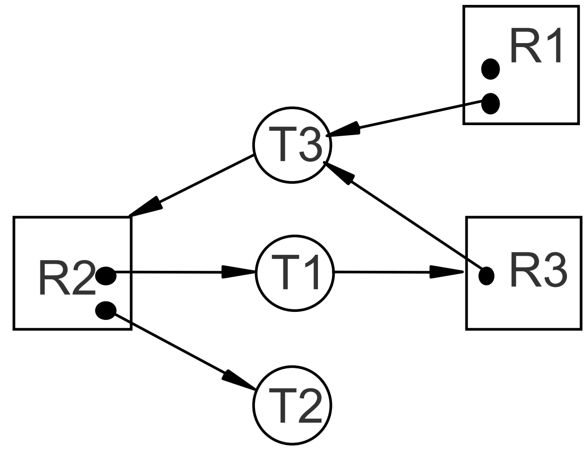
Resource Graph: Reduction

- A thread *t* can reduce the resource graph by removing all of the edges for allocation if it’s not isolated or blocked.
- A resource graph can be reduced completely if there is an order of reductions (threads reducing the graph by releasing the resources) so that all edges will be removed at the end.

- Theorem of deadlocks for resource graphs:
  - There is a deadlock if the resource graph cannot be reduced completely.

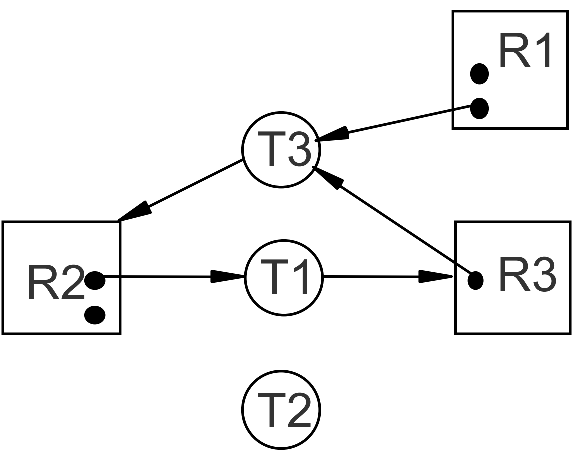
Resource Graph: Reduction II

Initial situation a)

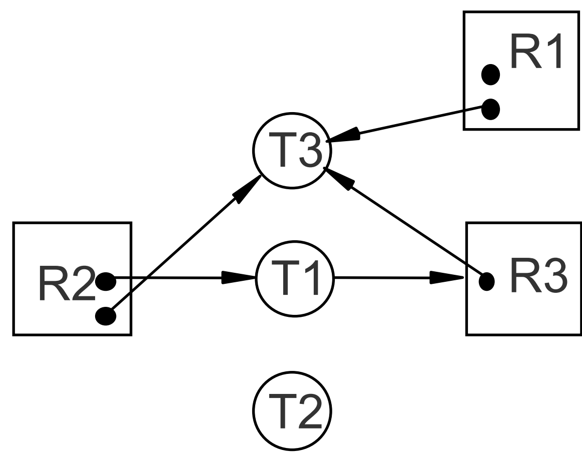


Resource Graph: Reduction III

b)



c)



Resource Graph: Reduction IV

d)

R1

T3

R2

T1

R3

T2

e)

R1

T3

R2

T1

R3

T2

Deadlock Avoidance

- The definition of a deadlock describes a current system situation.
- To *avoid* such a deadlock the remaining requests of the threads have to be known.
- In worst case all of the remaining requests occur at the same time.
- If all of these remaining requests can be satisfied the situation is *safe*.
- Otherwise the situation is *unsafe*.
- The system status is unsafe if there is a subset of threads that have remaining requests that cannot be satisfied with the resource currently available.
- There might be some requests that can be satisfied, but in worst case of orders of allocation and release a deadlock may occur.

Allocation Trajectory (with Deadlock)

T1

R
Rsy
A
R
Rsx
A

safe
safe
safe
unsafe

safe safe safe

A Rsx R T2

A Rsy R

Allocation Trajectory (without Deadlock)

T1

R
Rsy
R
A
Rsx
A

safe
safe
safe
unsafe
safe
safe
safe
safe
safe

A R
Rsx
T2
A
Rsy
R

Deadlock Avoidance

- Deadlock avoidance implement the resource management in such a way no unsafe situation can occur. So requests are satisfied only if it leads to a safe situation.
- A set of threads $T = \{T_1, T_2, \ldots, T_m\}$ is called safe if

  $\exists$ Permution $T_{k_1}, T_{k_2}, \ldots, T_{k_m}$ with

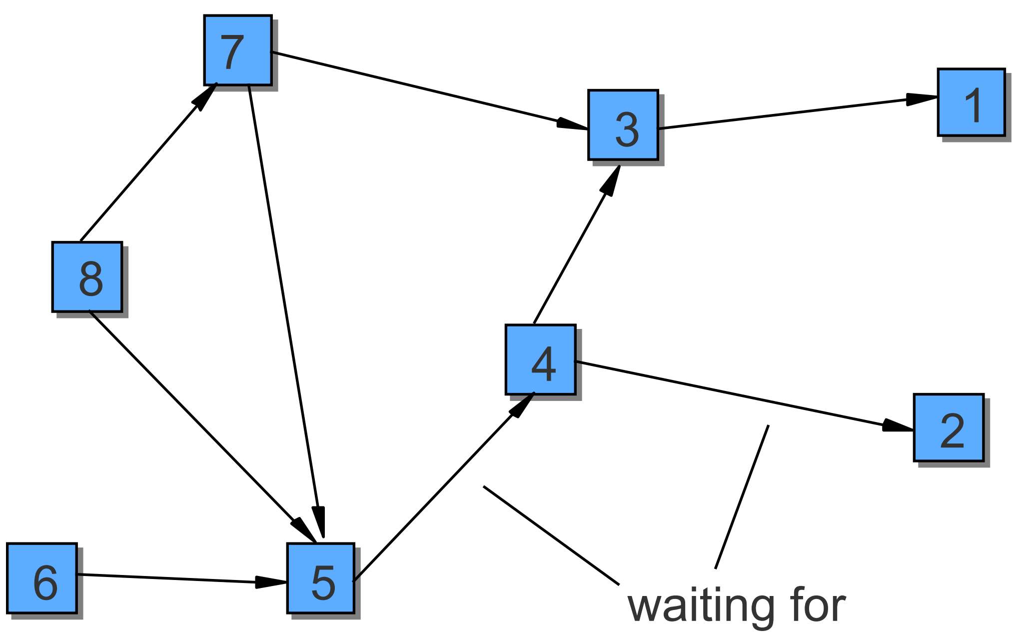
$$\forall r \in \{1,2,\ldots,m\} : \vec{r}_{k_r} \leq \vec{f} + \sum_{s=1}^{r-1} \vec{b}_{k_s}$$

or

$$\forall r \in \{1,2,\ldots,m\} : \vec{r}_{k_r} \leq \vec{v} - \sum_{s=r}^{m} \vec{b}_{k_s}$$

i.e. there is an order of thread terminations so the remaining requests of the other threads can be satisfied.

Example of a Termination Sequence



Deadlock Avoidance

- Check for every request if the allocation would lead to an unsafe situation.
- In case the request would lead to an unsafe situation postpone request.
- In order to perform the check for the current situation the **Banker's Algorithm** can be used.

Banker's Algorithm

- Calculate the vector of the (currently) free resources f.
- As long as threads cannot be considered as terminated:
  - Checking for each thread whether the remaining requirements can be met with the free resources f
    - If so, consider the thread as terminated and add all resources occupied by it to the vector f of all free resources
    - If no, select the next thread
  - Termination of the algorithm if the set of threads is empty or no thread can be terminated
- The situation is safe if all threads have been terminated and unsafe if not.

Complexity

- Complexity of the Banker's Algorithm
  - Selection of a thread (pass loop): $O(m\,n)$
  - with max. m threads: $O(m^2\,n)$

Example

- Given system with m=4 threads (i=1,..,4) and n=2 type of resources (j=1,2). Current status:

Allocated (busy) resources :

$$B = \begin{pmatrix} 0 & 4 \\ 1 & 0 \\ 3 & 0 \\ 5 & 4 \end{pmatrix}$$

Overall requests:

$$G = \begin{pmatrix} 4 & 6 \\ 2 & 4 \\ 3 & 1 \\ 10 & 6 \end{pmatrix}$$

Free resources:

$$f = \begin{pmatrix} 3 & 2 \end{pmatrix}$$

Example II

- Thus, the remaining requests can be calculated as *R* and the overall available resources *v*:

$$R = \begin{pmatrix} 4 & 2 \\ 1 & 4 \\ 0 & 1 \\ 5 & 2 \end{pmatrix} \qquad v = \begin{pmatrix} 12 & 10 \end{pmatrix}$$

Is this a *safe* situation?

- Following the algorithm:

  1. T3 can be terminated: $r3$ = (0 1) f = (3 2)

     $f := f + b3$ = (3 2) + (3 0) = (6 2)

  2. T1 can be terminated: $r1$ = (4 2) f = (6 2)

     $f := f + b1$ = (6 2) + (0 4) = (6 6)

  3. T2 can be terminated: $r2$ = (1 4) f = (6 6)

     $f := f + b2$ = (6 6) + (1 0) = (7 6)

  4. T4 can be terminated: $r4$ = (5 2) f = (7 6)

     $f := f + b4$ = (7 6) + (5 4) = (12 10) = v

- As there is a order to terminate the threads successfully (T3, T1, T2, T4) the situation is *safe*! (T3, T1, T4, T2 is also possible.)

Deadlock Detection

- In case there is no knowledge available about the remaining requests deadlock avoidance cannot be performed.
- So deadlocks may occur.
- At least the occurrence of a deadlock has to be *detected*.
- This can be done by searching for all of the threads that are not involved in a deadlock situation.

Deadlock Detection

- The threads $T_1, T_2, \ldots, T_m$ are not involved in a deadlock situation if

  $\exists$ Permution $T_{k_1}, T_{k_2}, \ldots, T_{k_m}$ with

  $$\forall r \in \{1,2,\ldots,m\} : \vec{a}_{k_r} \leq \vec{f} + \sum_{s=1}^{r-1} \vec{b}_{k_s}$$

  or

  $$\forall r \in \{1,2,\ldots,m\} : \vec{a}_{k_r} \leq \vec{v} - \sum_{s=r}^{m} \vec{b}_{k_s}$$

  there is a order of termination of the threads in such kind all request can be satisfied by free resources or by resources released earlier.

Deadlock Detection

- The approaches, the first one to *avoid* a deadlock by enforcing safe situations and the second one to *detect* deadlocks, differ only to a limited extend.
- In case of deadlock avoidance a permutation is searched for that can be executed if all of the *remaining requests* occur at the same time.
- For deadlock detection the same operations are performed for the current request.
- So we can use the Banker's Algorithm for deadlock detection too.
- We do have to replace the *remaining requests* with the *current requests*.
- The algorithm has to consider all of the threads.

Deadlock Resolution

- Every deadlock resolution focuses on cutting the cycle in the wait-for graph.
- In case it is impossible to (controlled) withdraw the resource the abort of the thread is necessary.
- Then there is the question which thread is to be aborted.

concepts of non-sequential and distributed programming

NEXT EVENT

Institute of Computer Science
Department of Mathematics and Computer Science

Semaphore and Monitor

APL IV: Concepts of Non-sequential and Distributed Programming (Summer Term 2023)