Universität Freiburg
Institut für Informatik
Dr.-Ing. Thomas Hornung

Georges-Köhler Allee, Geb. 51
D-79110 Freiburg
hornungt@informatik.uni-freiburg.de

# Exercises
# Distributed Systemes: Part 2
# Summer Term 2013

5.7.2013

## 2. Exercise sheet: Distributed Concurrency Control and ~~Reliability~~

**Exercise 1**

✓ Give an example of a serializable schedule that has been generated by a timestamp-based scheduler that could not have been generated by a 2PL scheduler.

$S_{TS} \neq S_{2PL}$

Hossein

**Exercise 2**

✓ Consider the distributed waiting graph for transactions $T_1$ to $T_6$ that are executed at sites 1, 2, and 3 (cf. Figure **??**). Assume that $T_1$ is requesting a lock at site 1 which is already occupied by $T_2$. Simulate the path pushing algorithm to detect a deadlock and give the resulting messages that are exchanged.

Julian

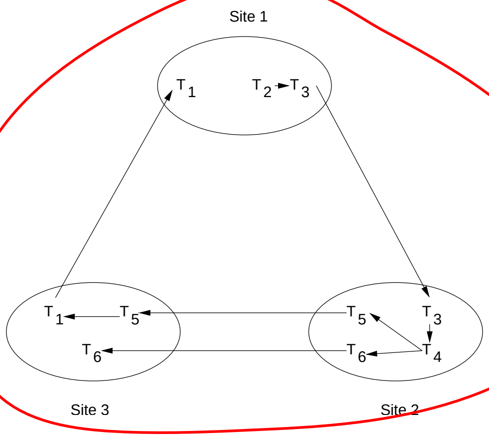


Figure 1: Distributed waiting graph

**Exercise 3**

✓ a) Find a global schedule using the procedure with explicit tickets for heterogeneous federations. Consider the local transaction $T_1$ and the global transactions $T_2$ und $T_3$ (for 2 sites with $D_1 = \{C, D\}$ and $D_2 = \{A, B\}$).

test

$$\begin{aligned} T_1 &= RC\,WC \quad RD\,WD \\ T_2 &= RC\,WC \quad RA\,WA \\ T_3 &= RD\,WD \quad RB\,WB \end{aligned}$$

Extend schedule $S$ with the corresponding *take-a-ticket*-operations. Does an equivalent serial global scheduler exist for schedule $S$ ? (Hint: Analyse the local/global conflict graphs).

$$S = \begin{array}{lllll} \text{Site 1:} & R_1C\ W_1C & R_1D\ W_1D & R_2C\ W_2C & R_3D\ W_3D \\ \text{Site 2:} & R_2A\ W_2A & R_3B\ W_3B & & \end{array}$$

✓ b) Prove the following statement: All schedules that the procedure with explicit tickets accepts are serializeable.

**Exercise 4**

Consider this notation for ordered messages that occur for a 2P-commit: $(i, j, M)$ means that site $i$ is sending a message $M$ to site $j$, where the value of $M$ is either P (prepare), R (ready), D (don't commit), C (commit), or A (abort). Assume that site 0 is the coordinator and sites 1 and 2 the participants. The following example shows a possible ordering of messages that result in a successful transaction:

$$(0, 1, P), (0, 2, P), (2, 0, R), (1, 0, R), (0, 2, C), (0, 1, C)$$

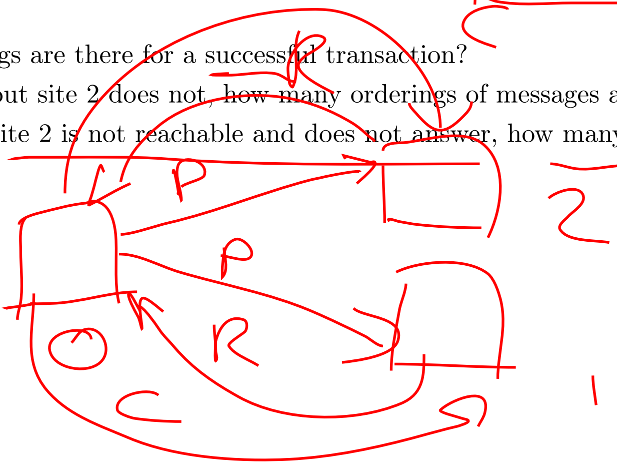
a) Give an ordering of messages that describes the following situation: site 1 requests a *commit*, site 2 requests an *abort*.

b) How many possible orderings are there for a successful transaction?

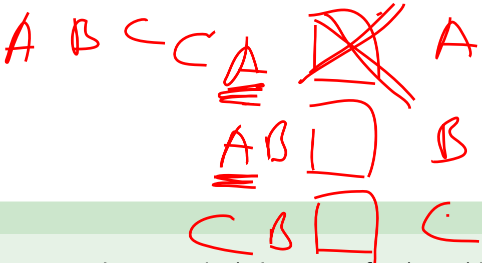
c) If site 1 wants to *commit*, but site 2 does not, how many orderings of messages are there for this situation?

✓ d) If site 1 wants to *commit*, site 2 is not reachable and does not answer, how many orderings of messages are there for this situation?

Sven



# 6. Replication

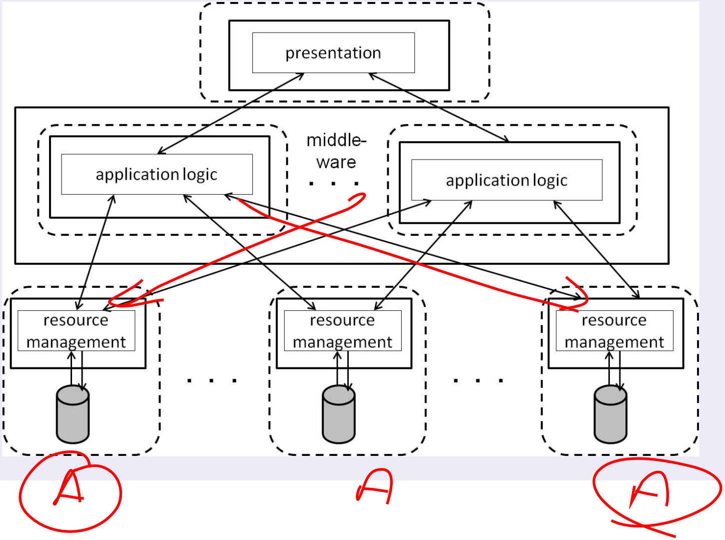


## Motivation

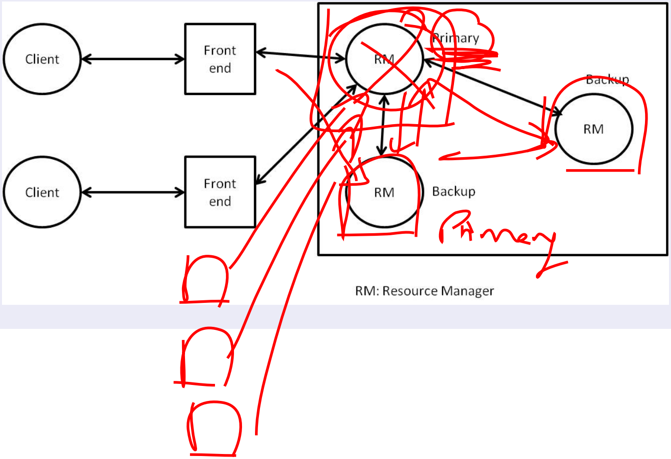
- Reliable and high-performance computation on a single instance of a data object is prone to failure.
- Replicate data to overcome single points of failure and performance bottlenecks.

Problem: Accessing replicas uncoordinatedly can lead to different values for each replica, jeopardizing consistency.

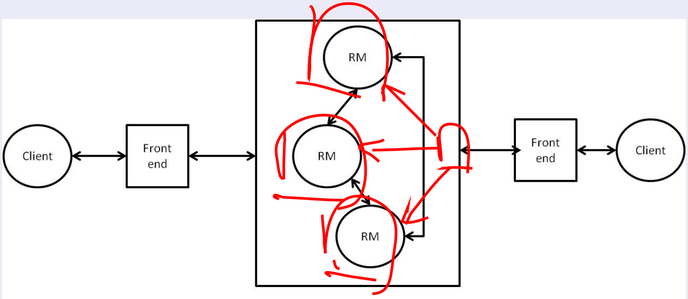
## Basic architectural model



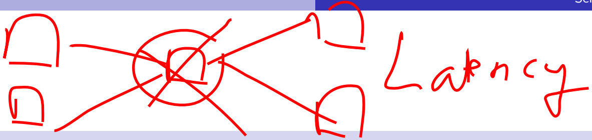
## Passive (primary-backup) replication model



## Active replication model



RM: Resource Manager



## CAP-Theorem

From the three desirable properties of a distributed shared-data system:

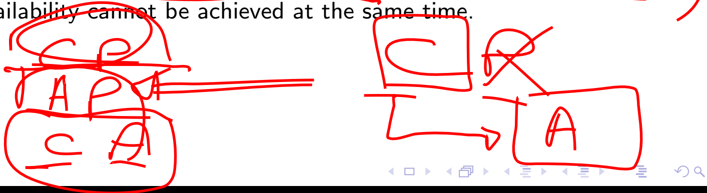
1 ■ atomic data consistency (i.e. operations on a data item look as if they were completed at a single instant),

2 ■ system availability (i.e. every request received by a non-failing node must result in a response), and

■ tolerance to network partition (i.e. the system is allowed to lose messages),

only two can be achieved at the same time at any given time.

$\Longrightarrow$ Given that in distributed large-scale systems network partitions cannot be avoided, consistency and availability cannot be achieved at the same time.



## the two options:

- Distributed ACID-transactions:

  Consistency has priority, i.e. updating replicas is part of the transaction - thus availability is not guaranteed.

- Large-scale distributed systems:

  Availability has priority - thus a weaker form of consistency is accepted: *eventually consistent*.

  $\Longrightarrow$ Inconsistent updates may happen and have to be resolved on the application level, in general.

## Eventually Consistent - Revisited. Werner Vogels (CTO at Amazon):[1]

- *Strong consistency*

  After the update completes, any subsequent access will return the updated value.

- *Weak consistency*

  The system does not guarantee that subsequent accesses will return the updated value. A number of conditions need to be met before the value will be returned. The period between the update and the moment when it is guaranteed that any observer will always see the updated value is dubbed the inconsistency window.

- *Eventual consistency*

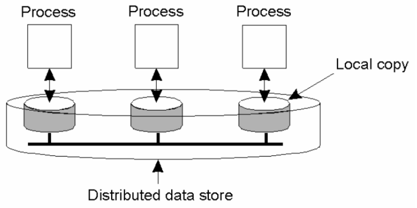
  This is a specific form of weak consistency; the storage system guarantees that if no new updates are made to the object, eventually all accesses will return the last updated value. If no failures occur, the maximum size of the inconsistency window can be determined based on factors such as communication delays, the load on the system, and the number of replicas involved in the replication scheme. The most popular system that implements eventual consistency is DNS (Domain Name System). Updates to a name are distributed according to a configured pattern and in combination with time-controlled caches; eventually, all clients will see the update.

[1] http://www.allthingsdistributed.com/2008/12/eventually_consistent.html

## 6.1: Systemwide Consistency

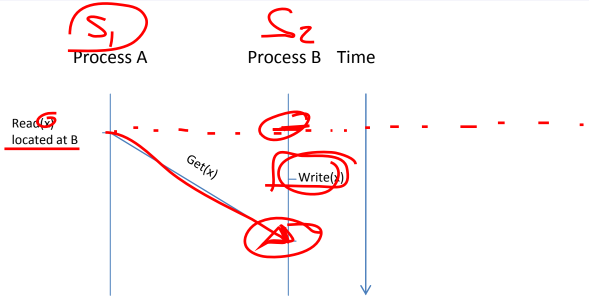
**Systemwide consistent view on a data store.**

- Processes read and write data in a data store.
  - Each process has a local (or near-by) copy of each object,
  - Write operations are propagated to all replicas.
- Even if processes are not considered to be transactions, we would expect, that read operations will always return the value of the last write – however what does "last" mean in the absence of a global clock?



## The difficulty of strict consistency

- Any read on a data item returns the value of the most recent write on it.
- This is the expected model of a uniprocessor system.
- In a distributed system there does not exist a global clock!



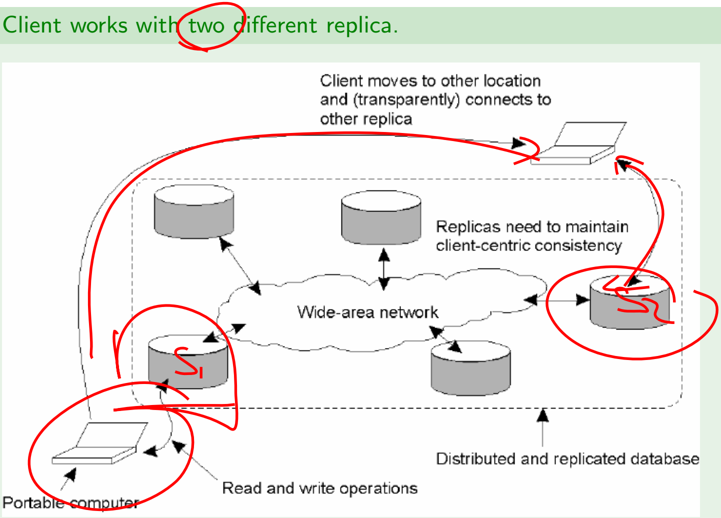
Which value shall be returned?
Old or new one?

## 6.2: Client-side consistency

Consistent view on a data store shall be guaranteed for clients, not necessarily for the whole system.

- Goal: *eventual consistency*.
  - In the absence of updates, all replicas *converge* towards identical copies of each other.
  - However, it should be guaranteed, that if a client has access to different replica, it sees consistent data.

## Example: Client works with two different replica.


Client moves to other location and (transparently) connects to other replica
Replicas need to maintain client-centric consistency
Wide-area network
Distributed and replicated database
Read and write operations
Portable computer
S1
S2

## 6.3 Server-side Consistency

### Problem

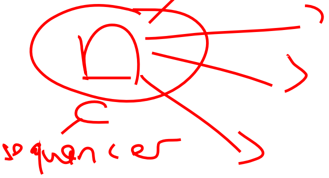
- We would like to achieve consistency between the different replicas of one object.
- This is an issue for active replication.
- It is further complicated by the possibility of network partitioning.

$$U_1 \ldots U_n$$

$$U_j < U_{j+1} < U_{j+2}$$

## Active Replication

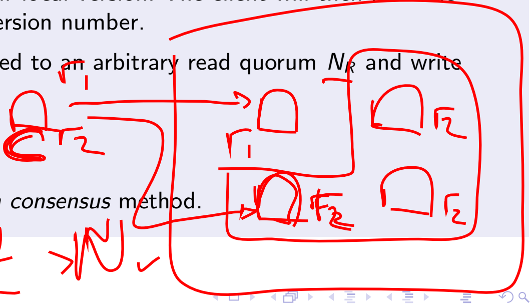
- Update operations are propagated to each replica.
- It has to be guaranteed, that different updates have to be processed in the same order for each replica.
- This can be achieved by totally-ordered multicast or by establishing a central coordinator called *sequencer*, which assigns unique sequence numbers which define the order in which updates have to be carried out.
- These approaches do not scale well in large distributed systems.



## Quorum-Based Protocols

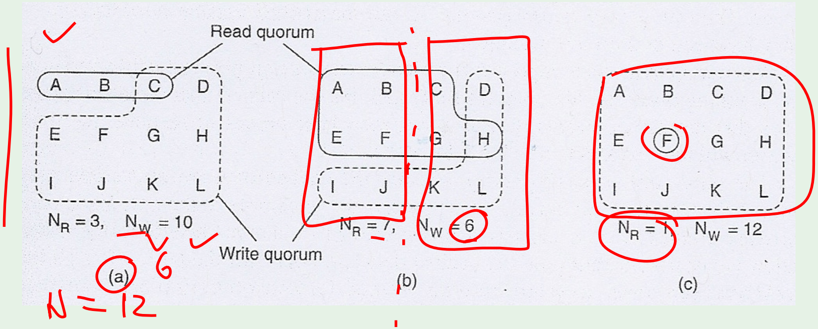
- Idea: Clients have to request and acquire the permission of multiple servers before either reading or writing a replicated data item.
- Assume an object has $N$ replicas.
  - For update, a client must first contact at least $\boxed{\frac{N}{2}+1}$ servers and get them to agree to do the update. Once they have agreed, all contacted servers process the update assigning a new version number to the updated object.
  - For read, a client must first contact at least $\frac{N}{2}$ servers and ask them to send the version number of their local version. The client will then read the replica with the highest version number.
- This approach can be generalized to an arbitrary read quorum $N_R$ and write quorum $N_W$ such that holds:
  - $\boxed{N_R + N_W} > N$, $\frac{N}{2}$
  - $N_W > \frac{N}{2}$.

This approach is called *quorum consensus* method.



$$\left(\underbrace{\frac{N}{2}+1}_{W}\right) + \frac{N}{2} > N \checkmark$$

## Example



(a) Correct choice of read and write quorum.

(b) Choice running into possible inconsistencies.

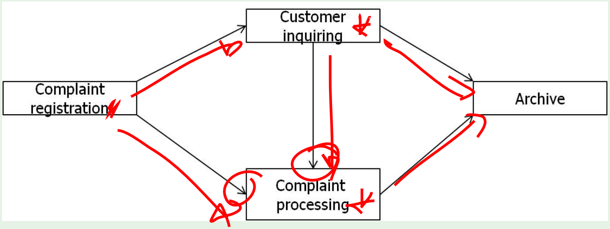
(c) Correct choice known as ROWA (read one, write all).

# Chapter 7: Modeling and Analysis of Distributed Applications

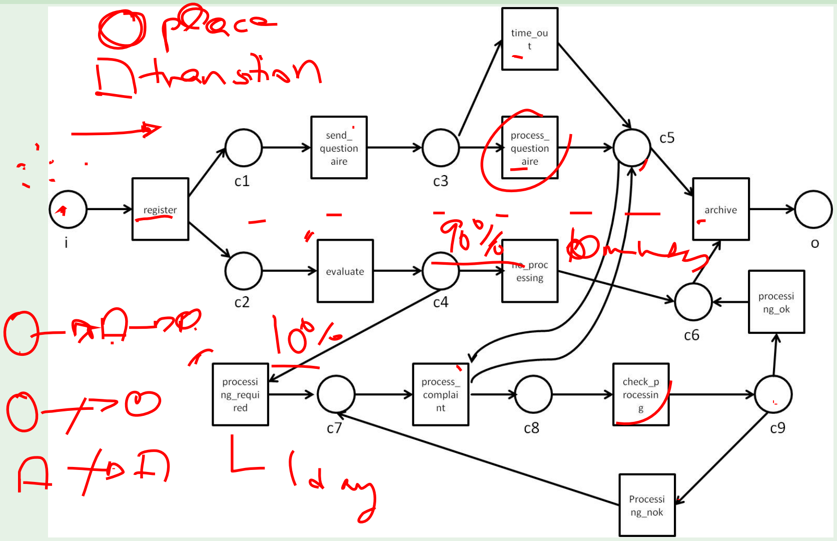
## Petri-Nets

- Petri-nets are abstract formal models capturing the flow of information and objects in a way which makes it possible to describe distributed systems and processes at different levels of abstraction in a unified language.
- Petri-nets have the name from their inventor Carl Adam Petri, who introduced this formalism in his PhD-thesis 1962.

## Processing of complaints: informal description.

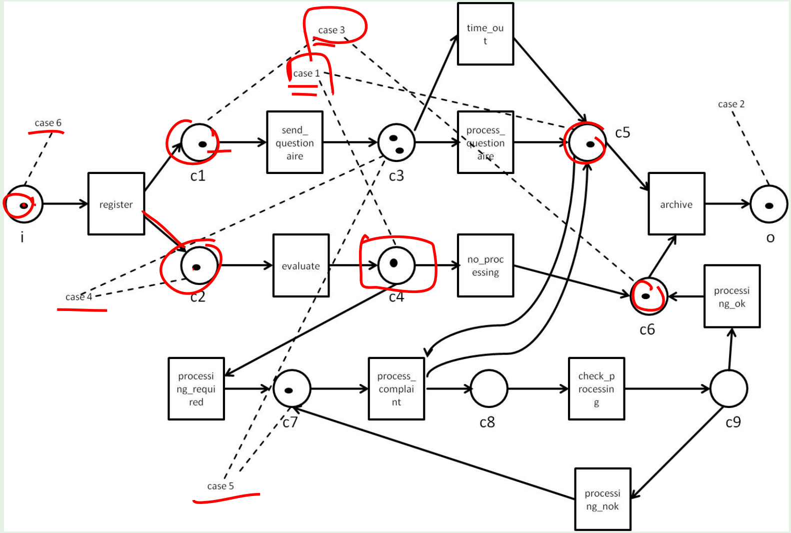


## Complaints processing: formal Petri-net orchestration.[1]

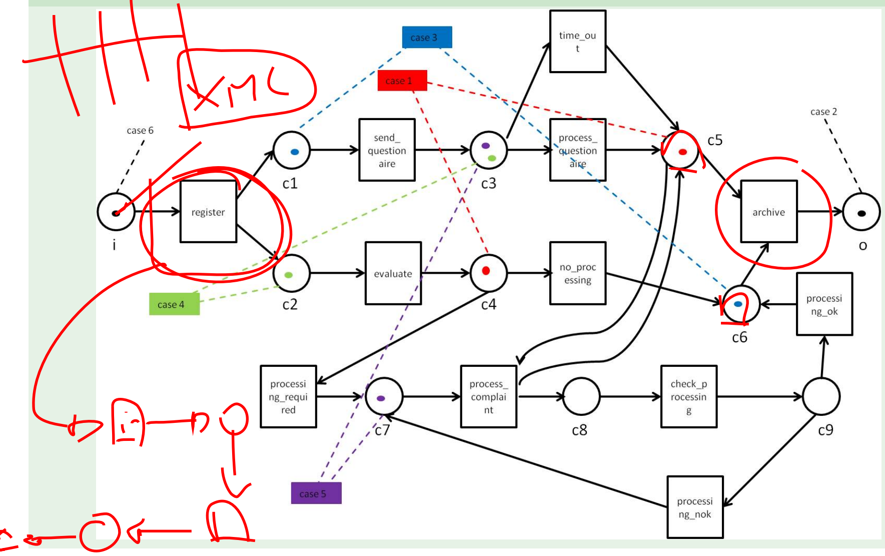


[1] van der Aalst: The Application of Petri nets to Workflow Management. Journal of Circuits, Systems, and Computers 8(1): 21-66 (1998)

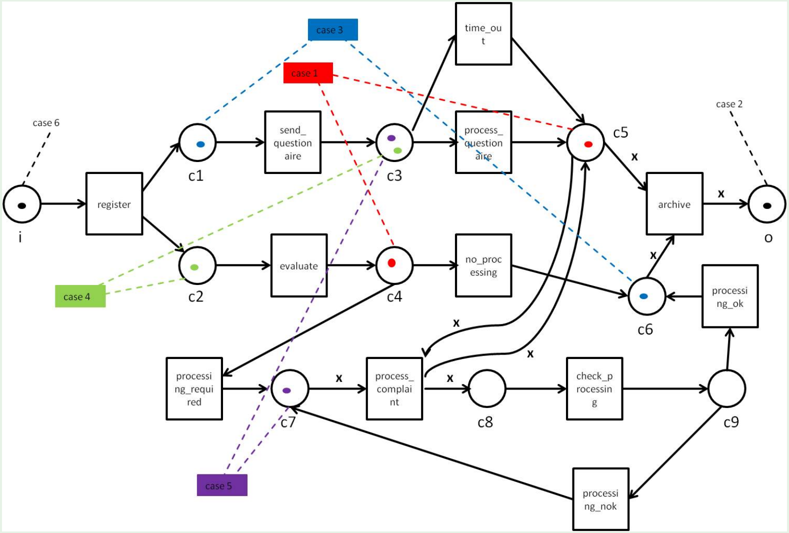
## Complaints processing: more than one complaint



## Complaints processing: how to distinguish complaints



## Complaints processing: keeping things together



## Petri-nets

Petri-nets model system dynamics.

- Activities trigger state transitions,
- activities impose control structures,
- applicable for modelling discrete systems.

## Benefits

- Uniform language,
- can be used to model sequential, causual independent (concurrent, parallel, nondeterministic) and monitored exclusive activities.
- open for formal analysis, verification and simulation,
- graphical intuitive representation.

The name *Petri-net* denotes a variety of different versions of nets - we will discuss the special case of *System Nets* following the naming introduced by W. Reisig.

## Petri-nets

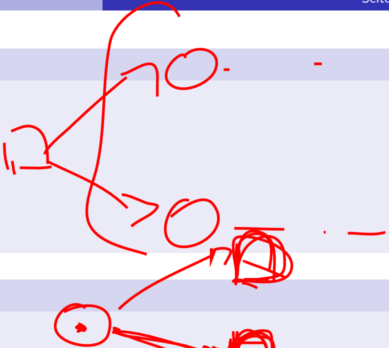
Petri-nets model system dynamics.

- Activities trigger state transitions,
- activities impose control structures,
- applicable for modelling discrete systems.

## Benefits

- Uniform language,
- can be used to model sequential, causual independent (concurrent, parallel, nondeterministic) and monitored exclusive activities.
- open for formal analysis, verification and simulation,
- graphical intuitive representation.

The name *Petri-net* denotes a variety of different versions of nets - we will discuss the special case of *System Nets* following the naming introduced by W. Reisig.



## Petri-nets

Petri-nets model system dynamics.

- Activities trigger state transitions,
- activities impose control structures,
- applicable for modelling discrete systems.

## Benefits

- Uniform language,
- can be used to model sequential, causual independent (concurrent, parallel, nondeterministic) and monitored exclusive activities.
- open for formal analysis, verification and simulation,
- graphical intuitive representation.

The name *Petri-net* denotes a variety of different versions of nets – we will discuss the special case of *System Nets* following the naming introduced by W. Reisig.

## Section 7.1 Elementary System Nets

### Basic elements of an elementary System Net (eS-Net)

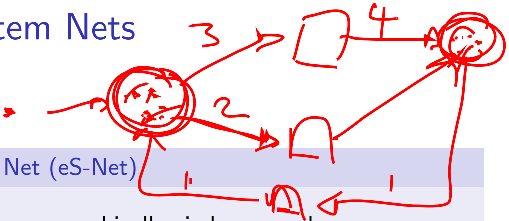
- System states are represented by *places*, graphically circles or ovals.
- A place may be marked by an arbitrary number of *tokens* graphically represented by black dots.
- System dynamics is represented by *transitions*, graphically rectangles.
- *Transitions* represent activities (events) and the causalities between such activities (events) are represented by edges.
- *Multiplicities* represent the consumption, respectively creation of resources which are caused by the *occurence* of activities.

## Section 7.1 Elementary System Nets

### Basic elements of an elementary System Net (eS-Net)

- System states are represented by *places*, graphically circles or ovals.
- A place may be marked by an arbitrary number of *tokens* graphically represented by black dots.
- System dynamics is represented by *transitions*, graphically rectangles.
- *Transitions* represent activities (events) and the causalities between such activities (events) are represented by edges.
- *Multiplicities* represent the consumption, respectively creation of resources which are caused by the *occurence* of activities.

## Section 7.1 Elementary System Nets

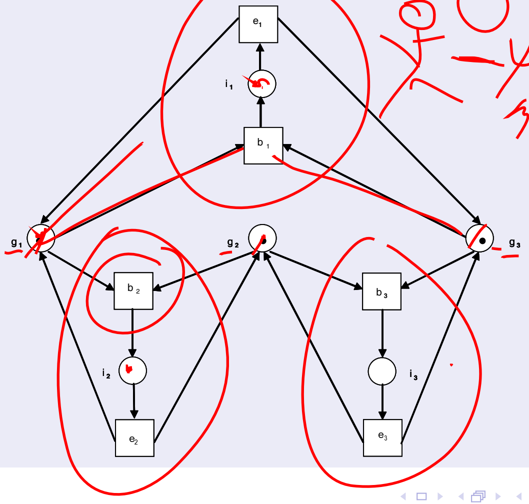


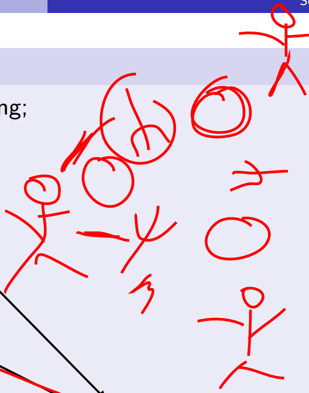
**Basic elements of an elementary System Net (eS-Net)**

- System states are represented by *places*, graphically circles or ovals.
- A place may be marked by an arbitrary number of *tokens* graphically represented by black dots.
- System dynamics is represented by *transitions*, graphically rectangles.
- *Transitions* represent activities (events) and the causalities between such activities (events) are represented by edges.
- *Multiplicities* represent the consumption, respectively creation of resources which are caused by the *occurence* of activities.

## 3-Philosopher-Problem

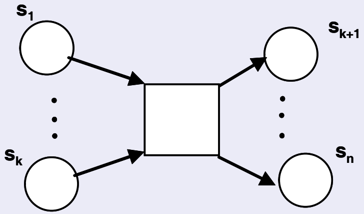
$b_j$: philosopher starts eating; $e_j$: philosopher stops eating;
$i_j$: philosopher is eating; $g_j$: fork on the desk;
$1 \leq j \leq 3$.





A transition *may* occur when certain conditions with respect to the markings of its directly connected places are fulfilled; the *occurence* of a transition - also called its *firing* - effects the markings of its directly connected edges, i.e. has local effects.

The *surrounding* of a transition $t$ is given by $t$ and all its directly connected places:

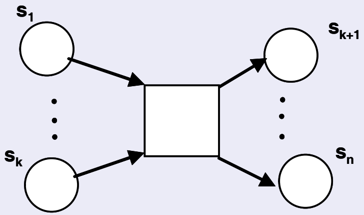


$s_1, \ldots, s_k$ are called *preconditions (pre-places)*, $s_{k+1}, \ldots, s_n$ *postconditions (post-places)*.

A place which is pre- and post-place at the same time is called a *loop*.

A transition *may* occur when certain conditions with respect to the markings of its directly connected places are fulfilled; the *occurence* of a transition - also called its *firing* - effects the markings of its directly connected edges, i.e. has local effects.

The *surrounding* of a transition $t$ is given by $t$ and all its directly connected places:

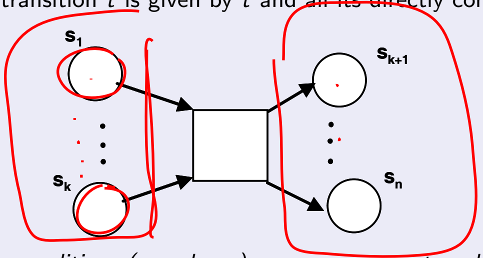


$s_1, \dots, s_k$ are called *preconditions (pre-places)*, $s_{k+1}, \dots, s_n$ *postconditions (post-places)*.

A place which is pre- and post-place at the same time is called a *loop*.

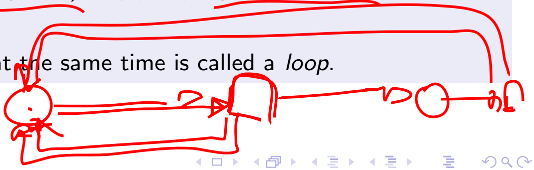
A transition *may* occur when certain conditions with respect to the markings of its directly connected places are fulfilled; the *occurence* of a transition - also called its *firing* - effects the markings of its directly connected edges, i.e. has local effects.

The *surrounding* of a transition $t$ is given by $t$ and all its directly connected places:



$s_1, \ldots, s_k$ are called *preconditions (pre-places)*, $s_{k+1}, \ldots, s_n$ *postconditions (post-places)*.

A place which is pre- and post-place at the same time is called a *loop*.



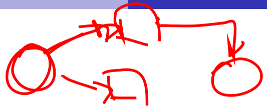
A *net* is given as a tripel $N = (P, T, F)$, where

- $P$, the set of *places*, and $T$, the set of *transitions*, are non-empty disjoint sets,
- $F \subseteq (P \times T) \cup (T \times P)$, is the set of directed edges, called *flow relation*, which is a binary relation such that $dom(F) \cup cod(F) = P \cup T$.

Let $N = (P, T, F)$ be a net and $x \in P \cup T$.

$$xF := \{y \mid (x, y) \in F\}$$
$$Fx := \{y \mid (y, x) \in F\}$$

For $p \in P$, $pF$ is the set of *post-transitions* of $p$; $Fp$ is the set of *pre-transitions* of $p$.
For $t \in T$, $tF$ is the set of *post-places* of $t$; $Ft$ is the set of *pre-places* of $t$.



A *net* is given as a tripel $N = (P, T, F)$, where

- $P$, the set of *places*, and $T$, the set of *transitions*, are non-empty disjoint sets,
- $F \subseteq (P \times T) \cup (T \times P)$, is the set of directed edges, called *flow relation*, which is a binary relation such that $dom(F) \cup cod(F) = P \cup T$.

dom = (p, t1
(t1, p2)
(p1, t1)
p1
pt

Let $N = (P, T, F)$ be a net and $x \in P \cup T$.

$$xF := \{y \mid (x, y) \in F\}$$
$$Fx := \{y \mid (y, x) \in F\}$$

For $p \in P$, $pF$ is the set of *post-transitions* of $p$; $Fp$ is the set of *pre-transitions* of $p$.
For $t \in T$, $tF$ is the set of *post-places* of $t$; $Ft$ is the set of *pre-places* of $t$.