Institute of Computer Science
Department of Mathematics and Computer Science

Freie Universität Berlin

Algorithms and Programming IV

Petri Nets

Summer Term 2023 | 10.05.2023

Barry Linnert

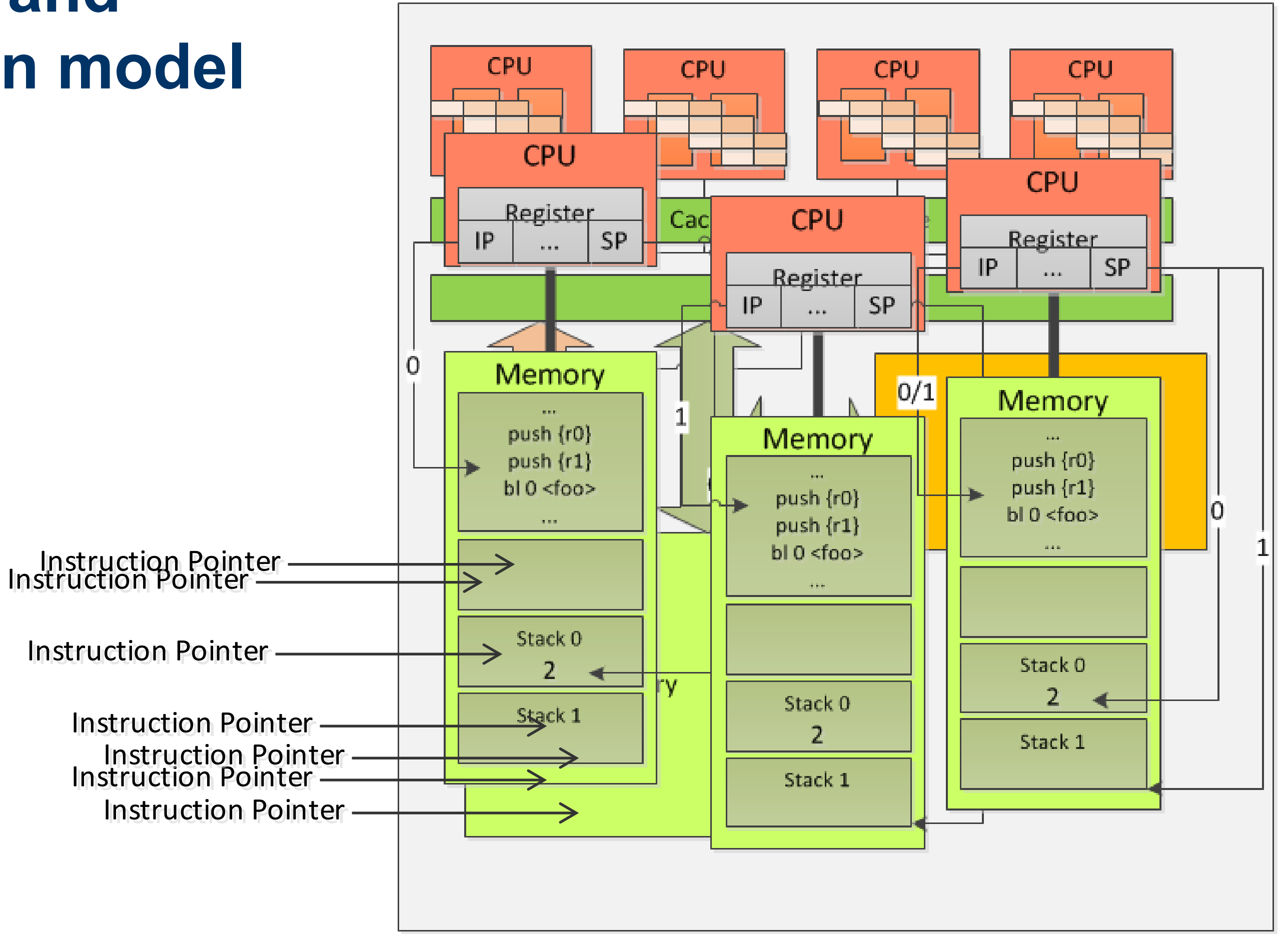
Objectives of Today‘s Lecture

- Introduction to Petri nets

Concepts of Non-sequential and Distributed Programming

PETRI NETS

Machine and Execution model



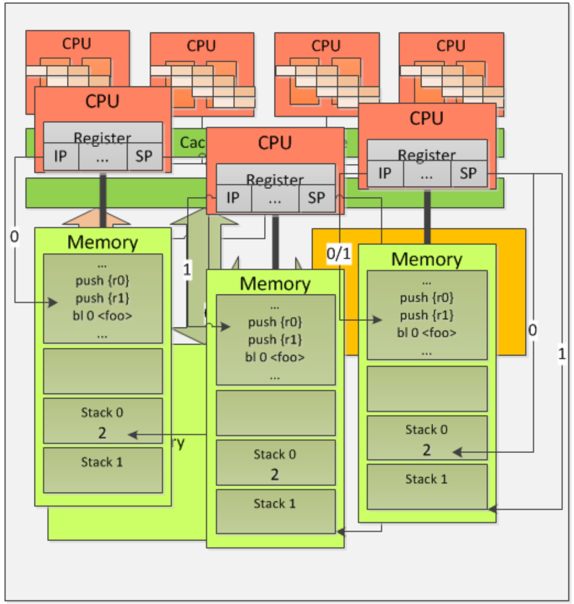
Correctness

- Correctness is ensured based on
  - Correct implementation of commands and functions
    - compiler/interpreter, HW
  - Correct execution of the set of commands and instructions
    - Sequential processing of the instructions 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 logic (calculus)
    - Simulation
    - Testing

Requirements for Programs

- A program should do what it is expected to do!
  - Functional requirements, such as ✓
    - Scope of functions
    - Correctness

- A program should comply with certain requirements about its behavior.
  - Non-functional requirements, such as ✓
    - Performance
    - Usability
    - Security
    - …



Example: Accounting

- A bank transfers money from one account to another.
- The amount of money to be debit from one account equals the amount of money that is transferred to the other account.
  - Money does not disappear or is created out of nothing.
- The transfers usually are performed on a multitude of accounts and more than once between different accounts.
- So the tasks may be performed concurrently.

- In our example two different threads transfer money between two accounts.

```c
// simple accounting with pthreads
#include ...
#define NUM_THREADS     2
// shared data
int account[NUM_THREADS];
pthread_mutex_t lock;

// accounting
void *bank_action (void *threadid)
{
  long tid;
  int i, amount, acc_nr, error = 0;
  tid = (long) threadid;
  for (i = 0; i < 300000; i++) {
    // calc acc_nr and amount
    // try to enter the critical section
    _error = pthread_mutex_lock(&lock);
    account[tid]                -= amount;
    account[acc_nr]             += amount;
    _error = pthread_mutex_unlock(&lock);
  }
  pthread_exit (NULL);
}
```

```c
int main (int argc, char *argv[])
{
  pthread_t threads[NUM_THREADS];
  int rc;
  long t;
  int i, j;

  // init lock
  pthread_mutex_init(&lock, NULL);

  // init data

  // creating threads

  // joining threads

  // output results

  /* Last thing that main() should do */
  pthread_exit(NULL);
}
```

```
// simple accounting with pthreads
#define NUM_THREADS     2
...
int main (int argc, char *argv[])
{
  ...
  // init lock
  // init data
  // creating threads
  // joining threads
  // output results
}
```

```
// accounting
void *bank_action (void *threadid)
{
  ...
    _error = pthread_mutex_lock(&lock);
    account[tid]                -= amount;
    account[acc_nr]             += amount;
    _error = pthread_mutex_unlock(&lock);
  ...
}
```

```
// accounting
void *bank_action (void *threadid)
{
  ...
    _error = pthread_mutex_lock(&lock);
    account[tid]                -= amount;
    account[acc_nr]             += amount;
    _error = pthread_mutex_unlock(&lock);
  ...
}
```

Modeling the Program Behavior

- The source code provides some structure that can be used to model the program behavior.
- Thus, the functions may be used as units of execution. But some of the functions may be used more often while the program is executed.
- The amount of execution and the sequence of execution have to be considered for modeling the runtime behavior of the program.
- For the example this is:

$$\begin{tikzcd}
\text{init} \arrow[d] & \texttt{main ()} \arrow[d] & 1 \arrow[d] \\
\text{working} \arrow[d] & \texttt{bank\_action ()} \arrow[d] & N \arrow[d] \\
\text{finish} & \texttt{main ()} & 1
\end{tikzcd}$$

Petri Nets

- Petri nets, also known as a **place/transition (PT) nets,** are a modeling approach to model dynamic behavior.
- Petri nets are introduced and named after Carl Adam Petri (12.07.1926 - 02.07.2010)
  - First draft in 1939.
  - Formulation as part of his PhD thesis (Dissertation) in 1962.

- There are many different version and variations in use today.
- Other modeling languages or approaches use parts of the Petri net modeling language, too:
  - UML (Unified Modeling Language – Activity Diagrams), Business Process Model and Notation, and EPC (Event-driven process chain)

Petri Nets II

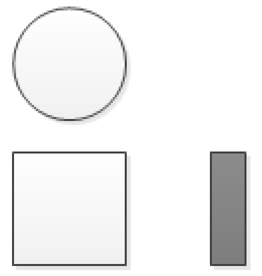
- Petri nets are especially used in areas of Modeling of business management processes and embedded systems.

- Some mathematical properties of a specific Petri net can be analyzed:
  - Reachability
  - Liveness
  - Boundedness

Basics

- A Petri net consists of **nodes** and/or **edges**.

- Nodes
  - Places
  - Transitions



- Edges
  - Edges are directed, connection two nodes.
  - Connect places with transitions and vice versa.
  - The direction of the edges depict the flow of the actions of the Petri net.

Basics II

- Petri net is a triple (P, T, F) with

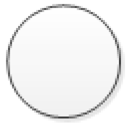
– $P \cap T = \varnothing$	the sets of places and transitions are disjoint.
– $P \cup T \neq \varnothing$	the set of nodes contains at least one position or transition.
– $F \subseteq (P \times T) \cup (T \times P)$	the set of edges only contains elements that connect nodes of different sets (places or transitions).

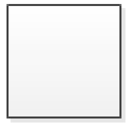
- For a node $x \in P \cup T$ sets are named

– $\bullet x := \{y \mid y\ F\ x\}$	a domain (subnet before the node) and
– $x \bullet := \{y \mid x\ F\ y\}$	a codomain (subnet after the node).

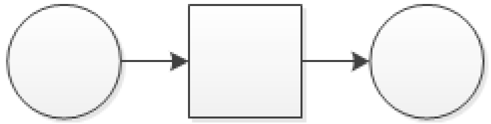
Simple Petri Nets

- These most simple Petri nets show isolated places or transitions $\bullet x \cup x\bullet = \varnothing$.

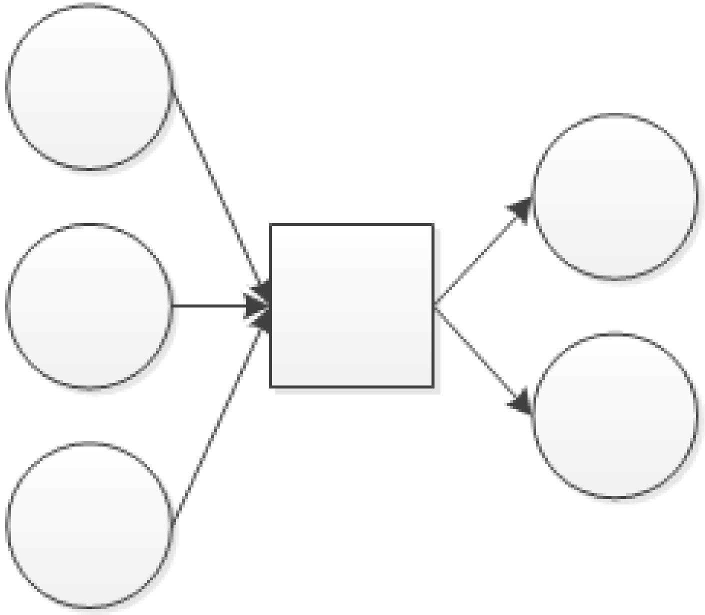




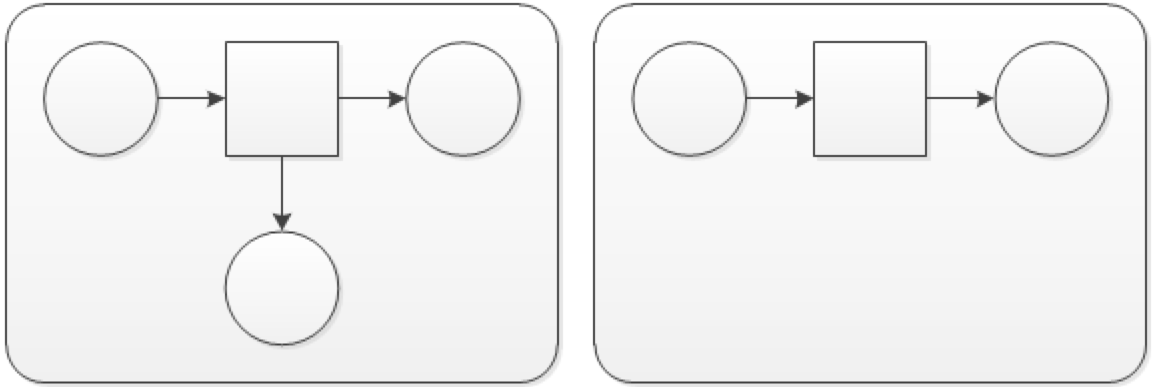
- This simple Petri net shows all fundamental elements: places, transition, edges.



Simple Petri Nets II

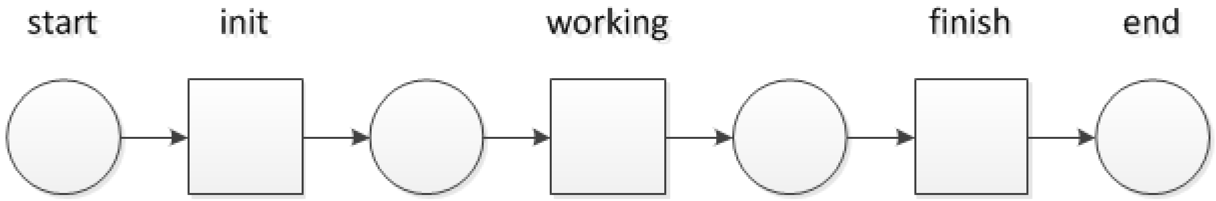


Isolated Subnets



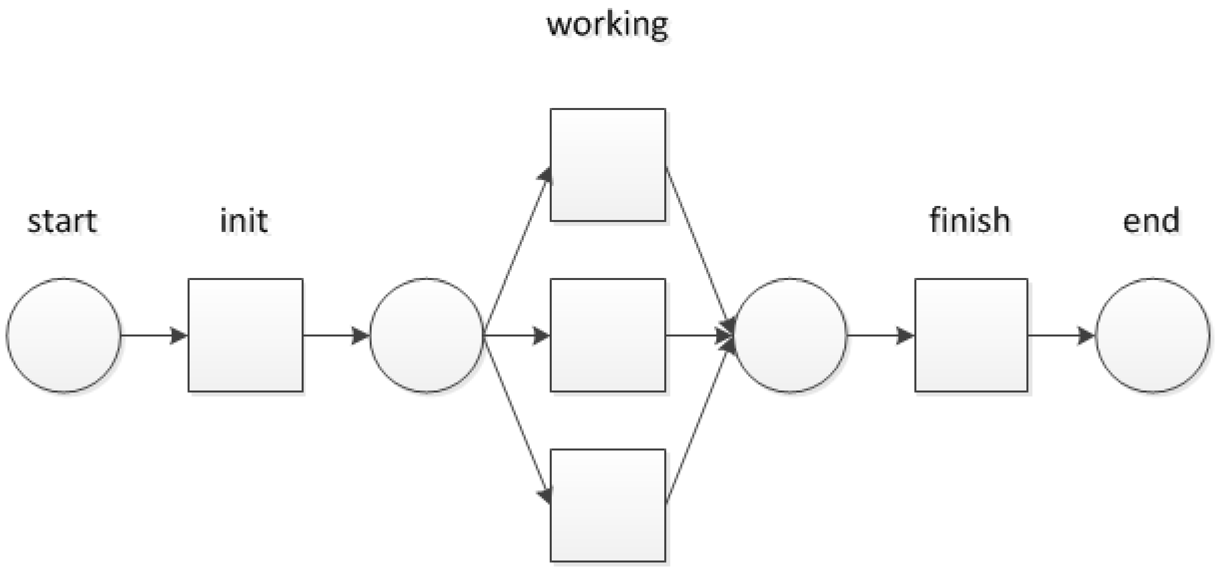
- Subnet
  - $S' \subseteq S$, $T' \subseteq T$, $F' \subseteq F$, $S'' = S \setminus S'$, $T'' = T \setminus T'$
- is isolated, if for all $x \in S' \cup T'$ holds:
  - $\bullet x \cap (S'' \cup T'') = \emptyset$
  - $x\bullet \cap (S'' \cup T'') = \emptyset$

Example: Accounting



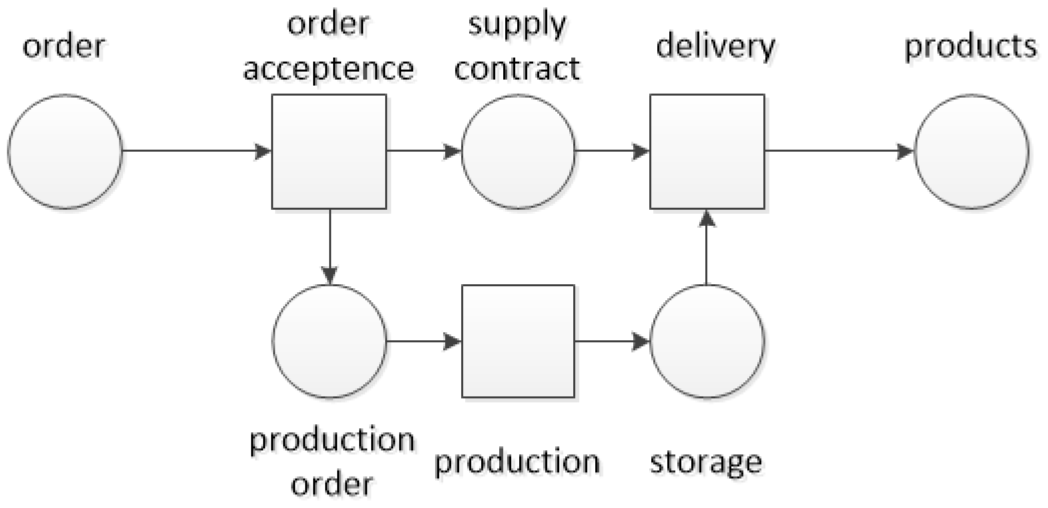
- Places display states, artifacts or objects.
- Transitions stand for activities.

Example: Accounting II

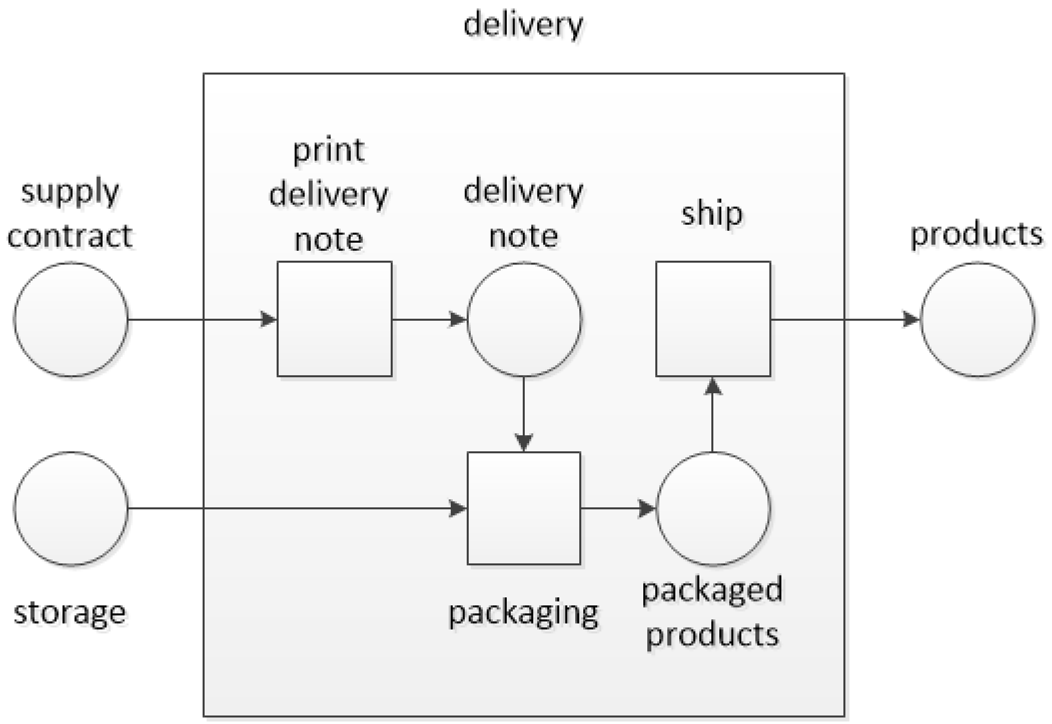


- Places display states, artifacts or objects.
- Transitions stand for activities.

Example: Production Management

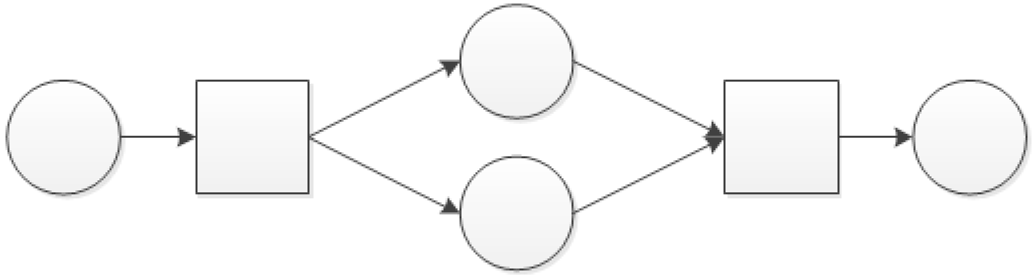


Example: Production Management II



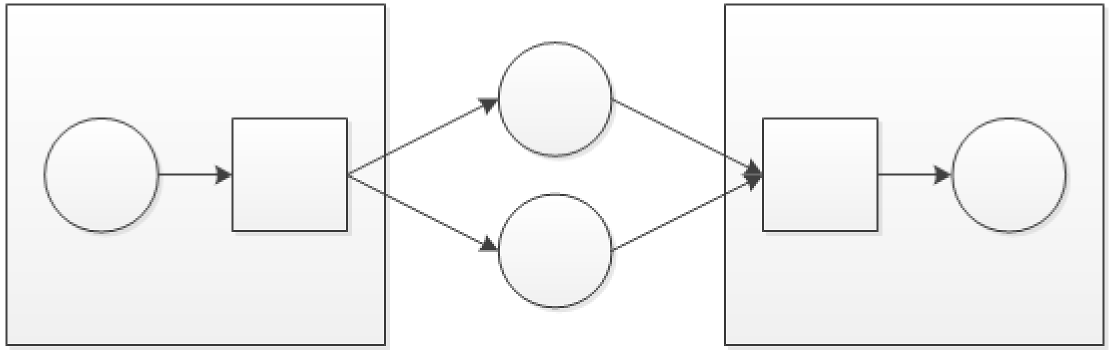
Net Morphisms

- In many cases it is useful to simplify a complex Petri net (abstraction) or to go into detail of a more abstract one (refinement).



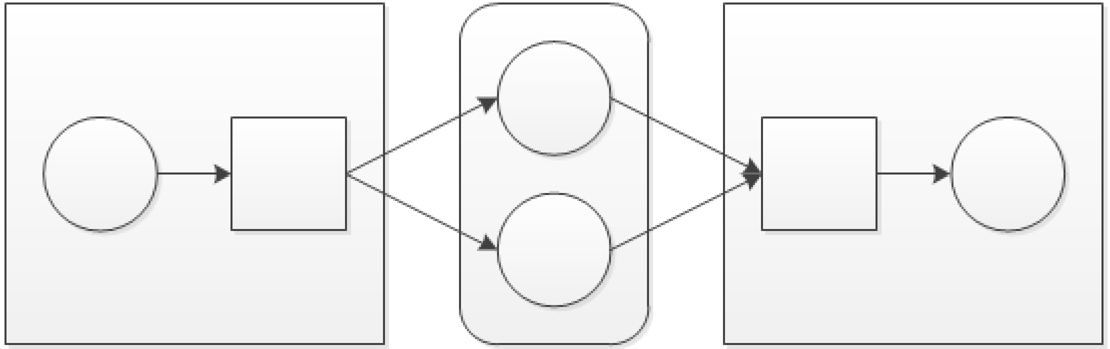
Net Morphisms II

- Parts of the Petri net can be combined to build a basic element of the Petri net.



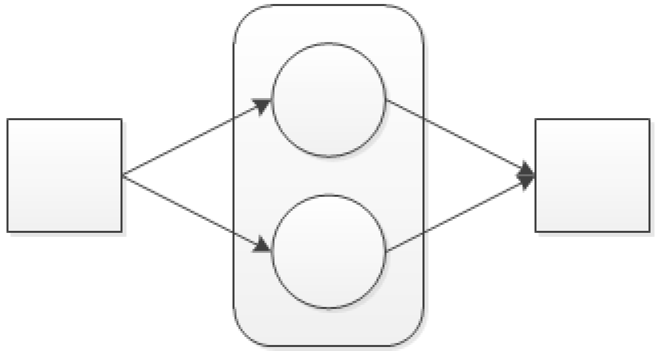
Net Morphisms III

- Combinations can be found for places and transitions and places only.



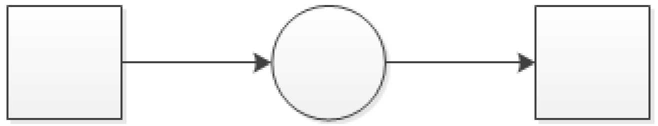
Net Morphisms IV

- Thus, the combinations can be modelled as transitions….

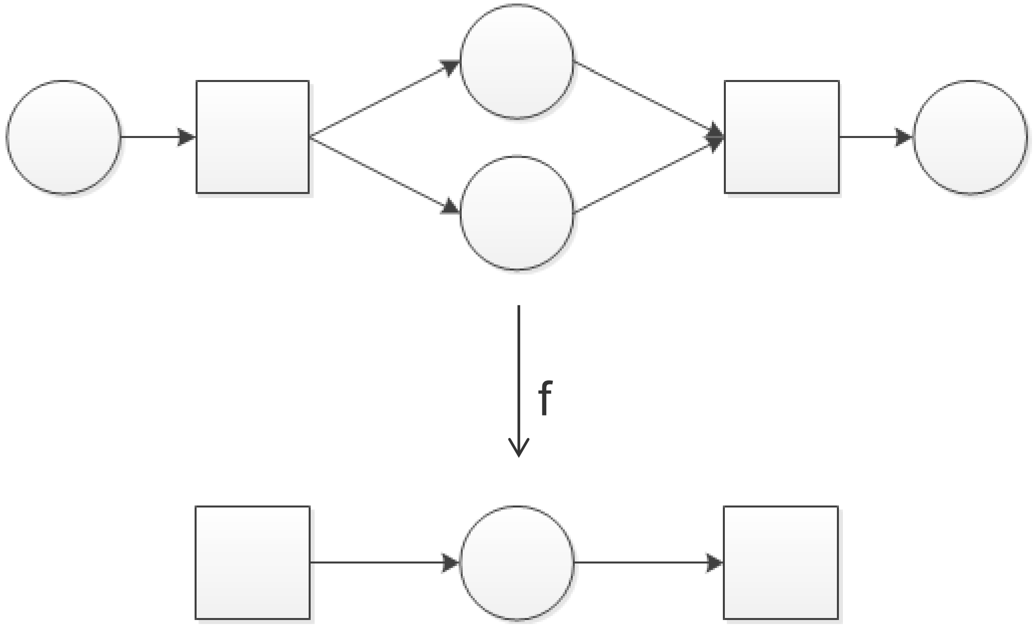


Net Morphisms V

- … and as places.



Net Morphisms VI



Net Morphisms

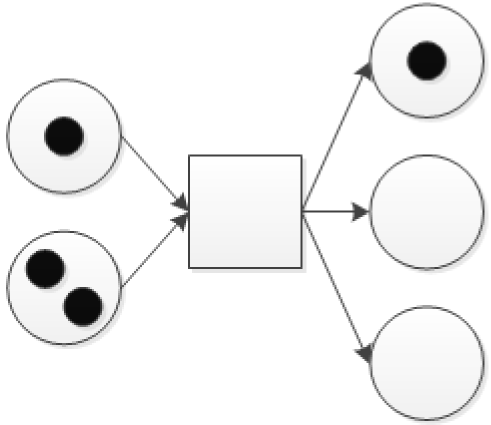
- While **f** is a net morphism it has to fulfill some requirements.

- Thus, the structure of the original Petri net must be preserved:
  - Equal number of incoming and outgoing edges,
  - With equal nodes from or to the incoming or outgoing edges

- Additionally it is useful to keep the names of the nodes or find a new label related to the original name.

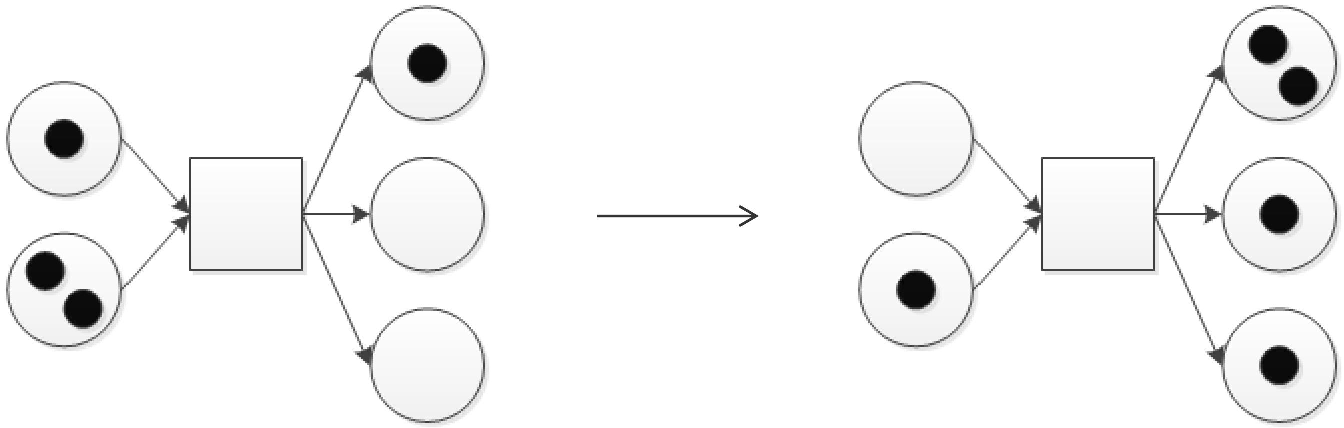
Petri Nets with Tokens

- In order to model a dynamic behavior **tokens** can be used to depict the state of the Petri net and to change from one state to another.
- These Petri nets are also called state transition nets.



Petri Nets with Tokens II

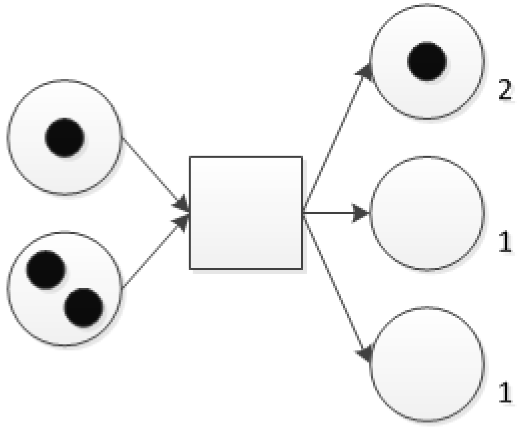
- The change from one state to another is represented moving tokens through transitions. The transition is ***firing***.



- A token is taken from each incoming place of the transition.
- A token is added to each outgoing place of the transition.

Petri Nets with Tokens III

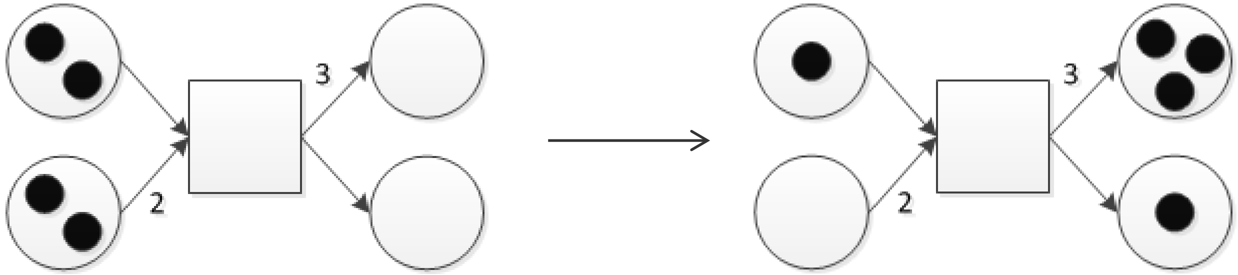
- The number of tokens that a place can hold can be limited by the capacity of the place.



- The capacity is assigned as a number to the place.
- If no number is assigned to the place, the capacity is unlimited.

Petri Nets with Tokens IV

- The number of tokens that are stored at a place or removed from a place by firing of a transition is called weight. The weight is depicted by a number assigned to the edge to and from the transition.

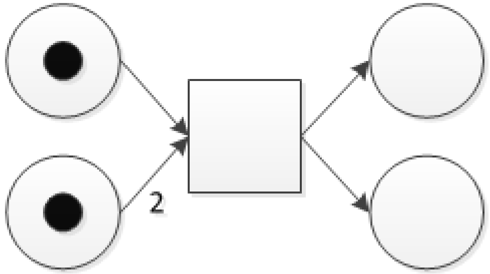


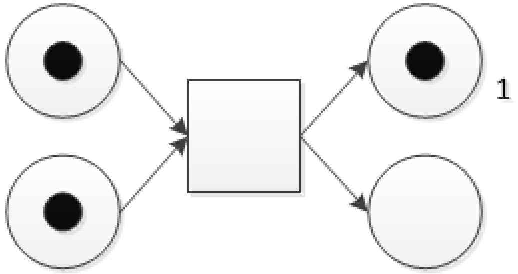
- If no weight is specified, a weight of 1 is defined.

Firing of Transitions

- Transitions are firing (immediately and without delay), when all requirements on the input and output side are fulfilled.
- If not all of the requirements are fulfilled the transition will not fire.

- Examples of transitions that cannot fire:





Properties of Petri Nets

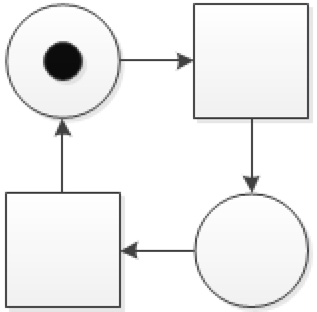
- Reachability
  - A part of the Petri net is reachable if there is a sequence of transitions able to fire at some point in time and leading to mark the places or one of the places of this part with tokens. This is interesting in order to decide if some part of the program code is executed or a sub-system is during operation of the whole system.

- Liveness
  - Different types of liveness can be investigated, describing if the net or parts of the Petri net are able to show some actions by firing of transitions unless the final state is reached.
  - Weak liveness is given if there is a transition that can fire and the firing of that transition enables the firing of (at least) one other transition.
  - Strong liveliness is given when the weak liveliness applies to all transitions.

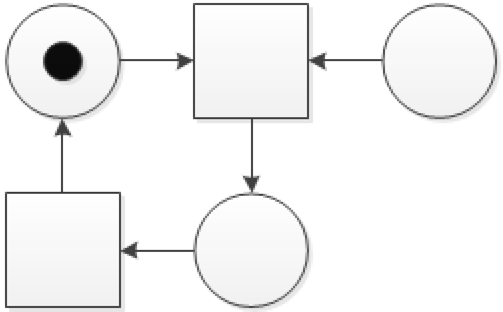
Properties of Petri Nets II

- Termination
  - After a sequence of transitions the Petri net is no longer alive.
- Boundedness
  - Indicates if parts of the Petri net show or may show an overload situation.
- Security
  - A Petri net is secure, if an increase in capacities of a place does not lead to an increased possibility in firing of transitions.

Liveness

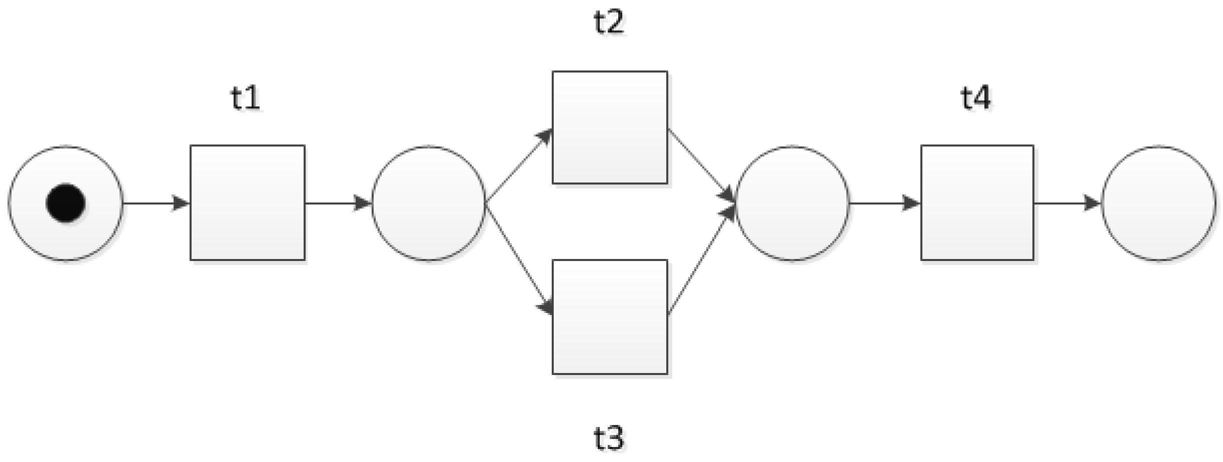


This Petri net is alive.



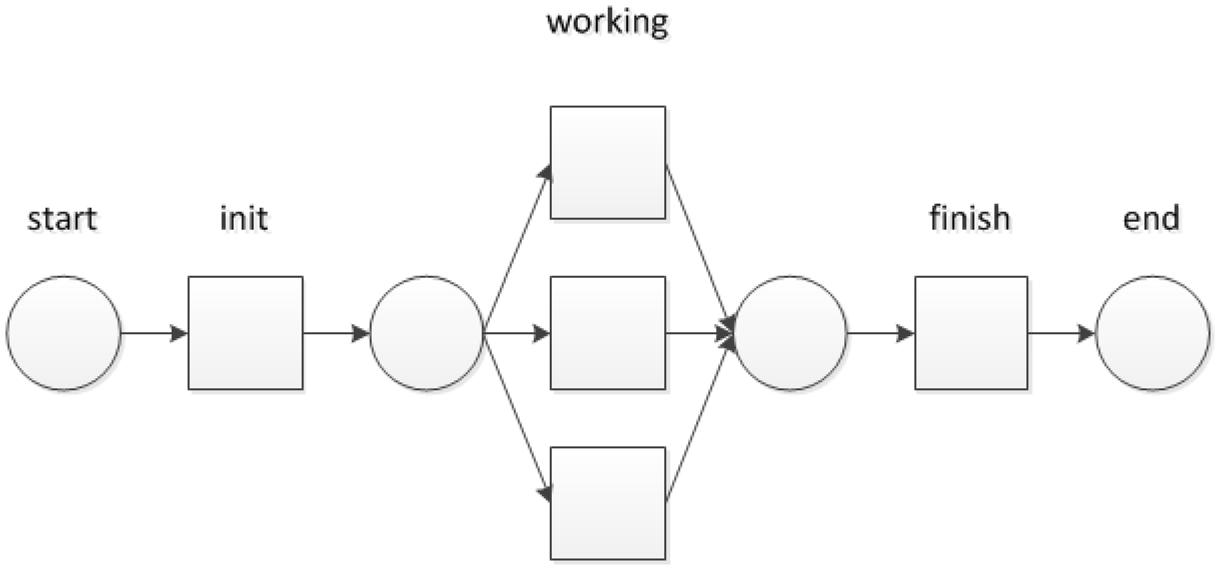
This Petri net is not alive anymore.

Non-determinism



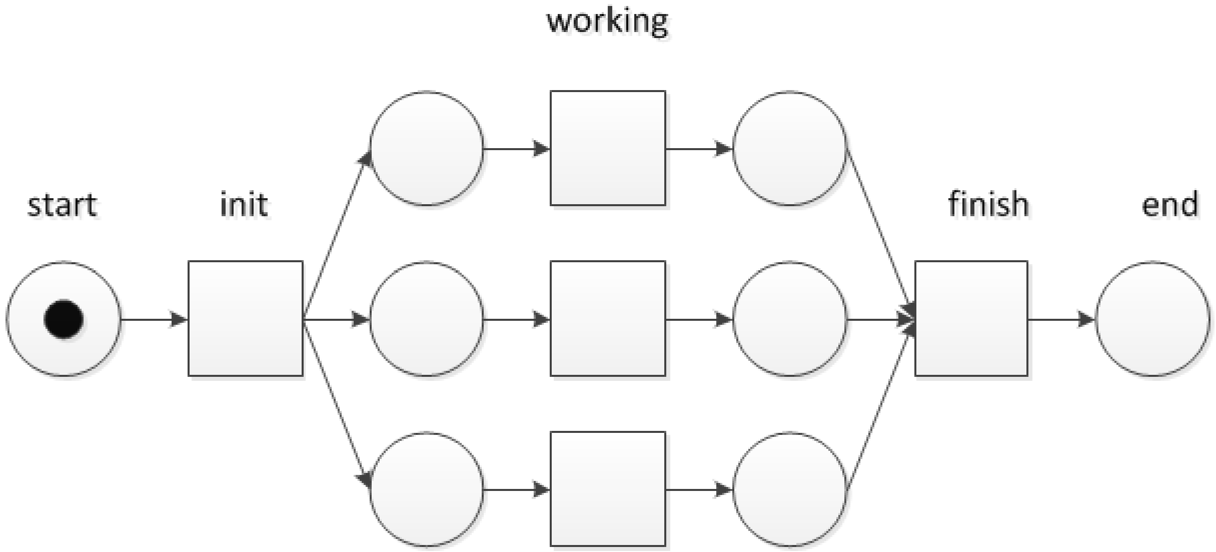
Concurrency/Parallel processing

- Example: Accounting

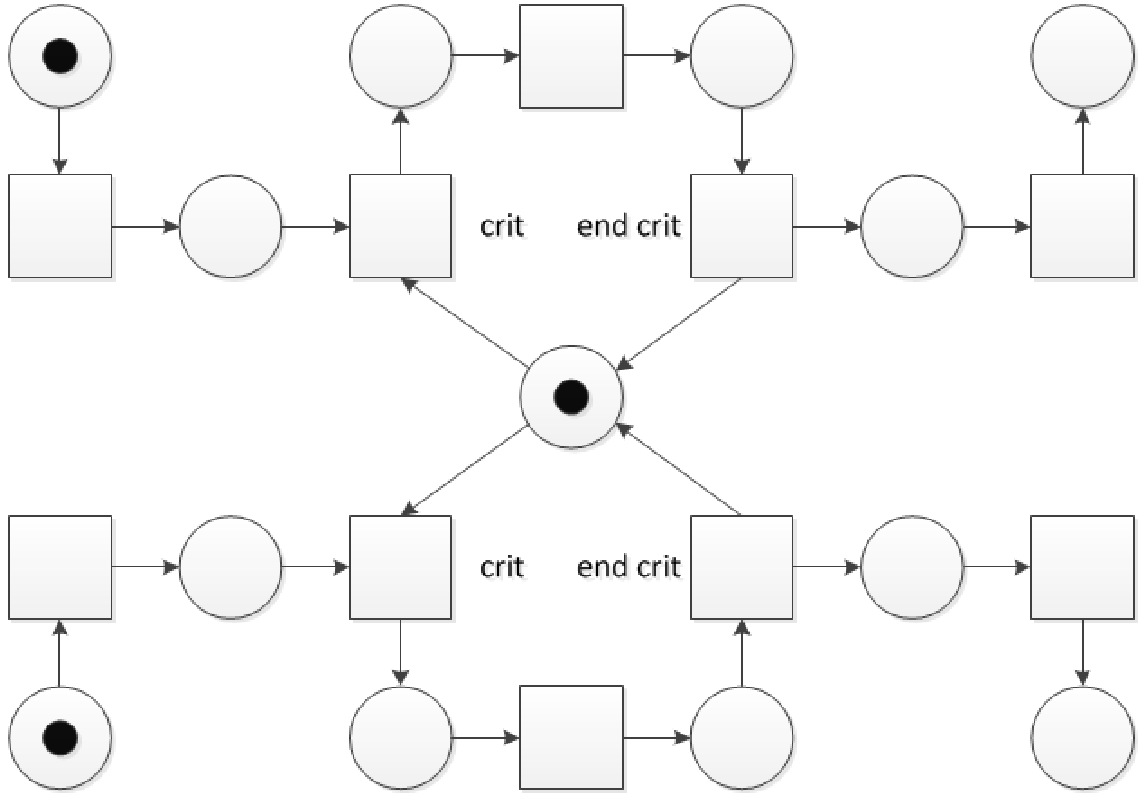


Concurrency/Parallel processing II

- Example: Accounting

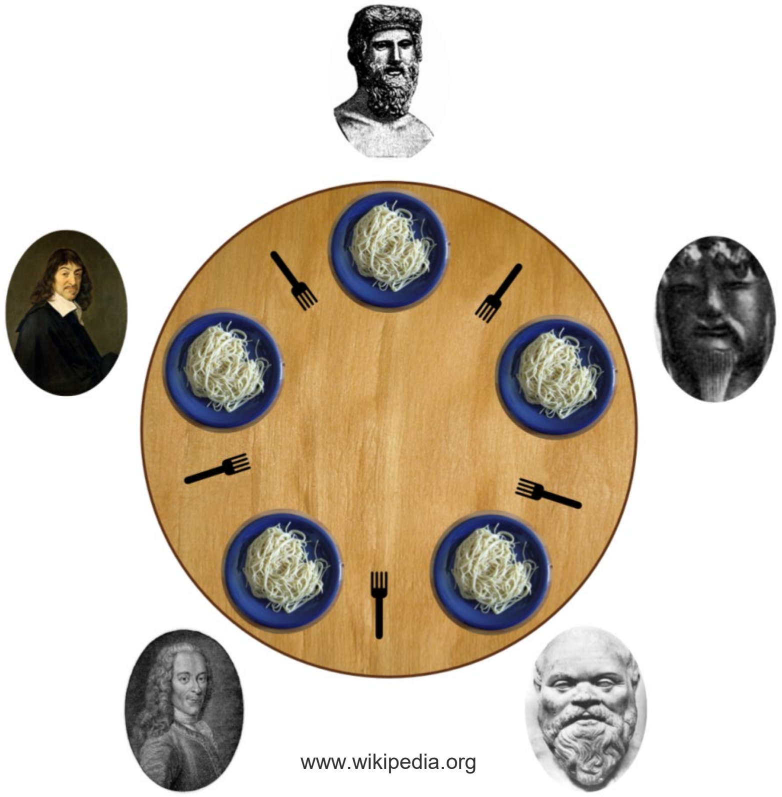


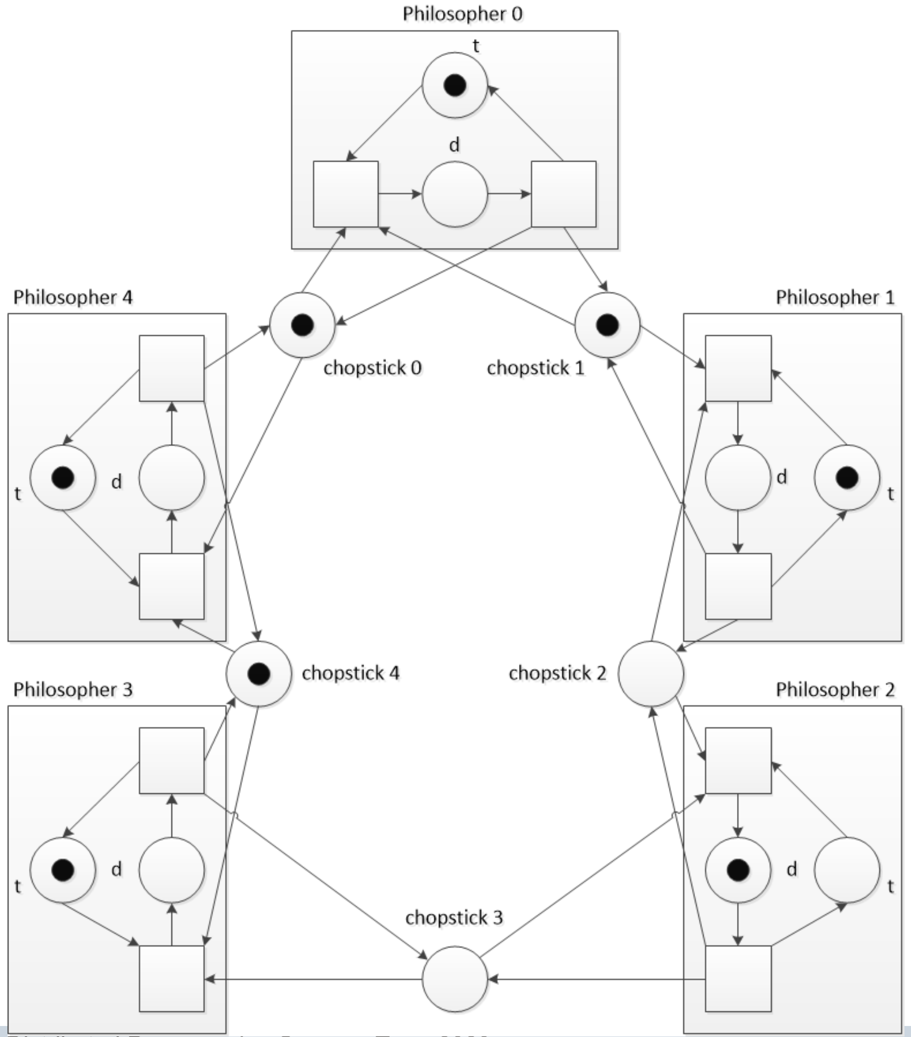
Protection of the Critical Section



Example: Dining Philosophers

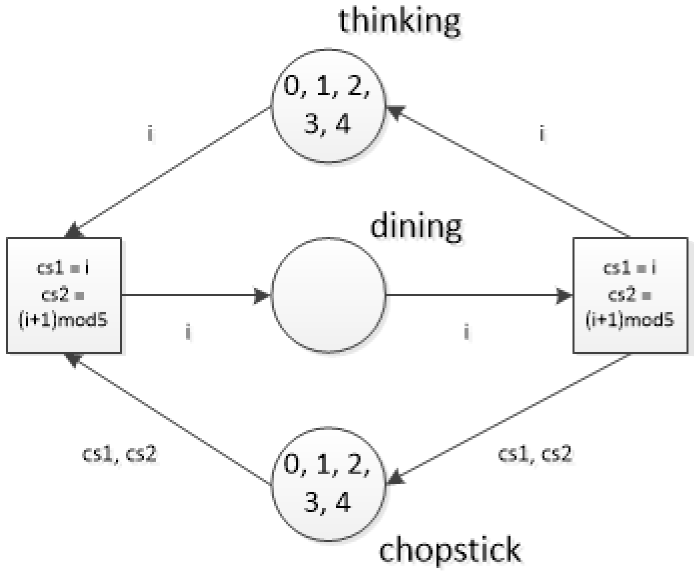
- Introduced by Edsger Dijkstra in 1965.
- Philosophers sitting around a table thinking and eating (not so much talking…).
- In order to eat a philosopher needs two forks (chopsticks). Thus, a philosopher has to check if both of the forks – right and left of hers/his plate – are available.
- Apparently there are not enough forks to have all philosophers eating and if everyone is taking one fork there is no other left, so nobody is able to have their pasta.
- Today, it's a classical problem in the field of concurrent algorithms.




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

Colored Petri Nets

- For simplification of complex Petri nets the tokens can be distinct by using colors or numbers.
- Example of the Dining Philosophers modeled by a colored Petri net:



Generalized Modular Petri Nets (GMPN)

- Especially in embedded systems and production processes the time one action needs to be performed is crucial and has to be considered in the model, too.

- Diploma thesis Felix Kühling (TUB, today AMD Toronto, Canada)
- Timed Petri nets
  - The individual transitions have a time it takes to fire.

Stochastic Petri Nets

- The firing of the individual transition depends on a random/stochastic function.

Concepts of Non-sequential and Distributed Programming

NEXT LECTURE

Institute of Computer Science
Department of Mathematics and Computer Science

Freie Universität Berlin

Deadlocks

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