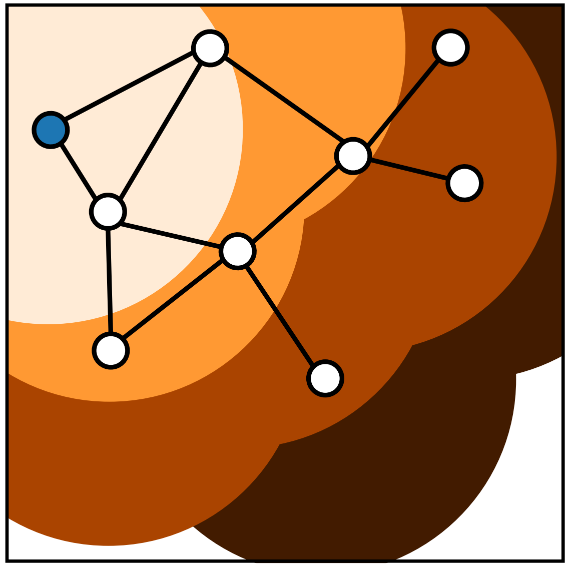
Discrete Event Systems
Petri Nets



Lana Josipović
Digital Systems and Design Automation Group
dynamo.ethz.ch

ETH Zurich (D-ITET)

December 19, 2024

Most materials from Lothar Thiele and Romain Jacob

Last week in Discrete Event Systems

Token Game of Petri Nets

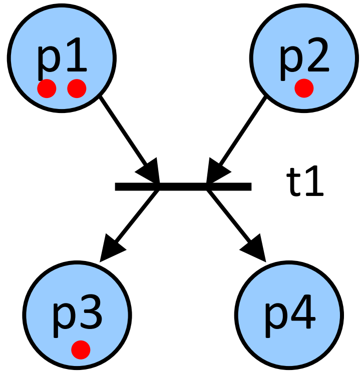
A marking M activates a transition t ∈ T if each place p
connected through an edge f towards t contains at least w(p,t) tokens.

If a transition t is activated by M,
a state transition to M' fires (happens) eventually.

Only one transition is fired at any time.

When a transition fires

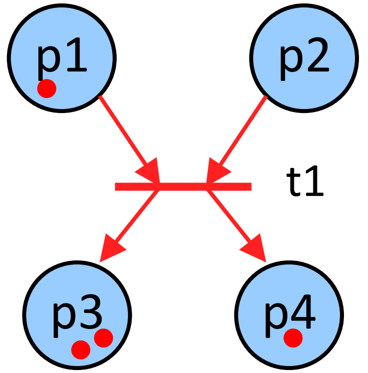
- it consumes a token from each of its input places,
- it adds a token to each of its output places.



Token Game of Petri Nets

A marking M activates a transition t ∈ T if each place p
connected through an edge f towards t contains at least w(p,t) tokens.

If a transition t is activated by M,
a state transition to M' fires (happens) eventually.



Only one transition is fired at any time.

When a transition fires

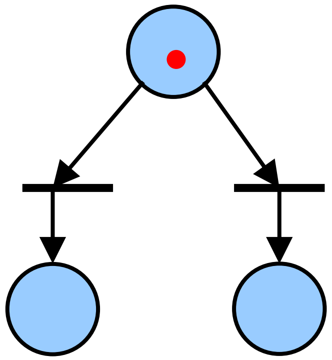
- it consumes a token from each of its input places,
- it adds a token to each of its output places.

Concurrent Activities

Finite Automata allow the representation of decisions, but no concurrency.

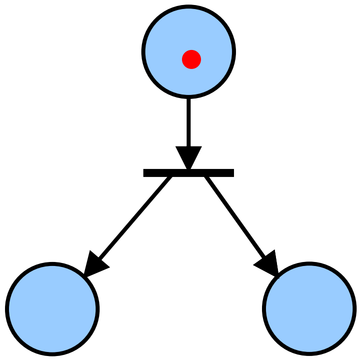
Petri nets support concurrency with intuitive notations:

Decision

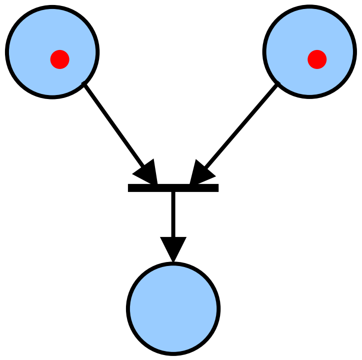


decision / conflict

Concurrency



fork



join / synchronization

Definition

- Semantics
- Token game

Properties

- Safety
- Liveness

Analysis

- Coverability tree
- Incidence matrix

This week in
Discrete Event Systems

Discrete Event Models with Time

In many discrete event systems, time is an important factor.

- queuing systems
- computer systems
- digital circuits
- workflow management
- business processes

Discrete Event Models with Time

In many discrete event systems, time is an important factor.

- queuing systems
- computer systems
- digital circuits
- workflow management
- business processes

Based on a **timed discrete event model,** we would like to determine properties:

- delay
- throughput
- execution rate
- resource load
- buffer sizes

Discrete Event Models with Time

In many discrete event systems,
time is an important factor.

- queuing systems
- computer systems
- digital circuits
- workflow management
- business processes

Based on a **timed discrete event model,**
we would like to determine properties:

- delay
- throughput
- execution rate
- resource load
- buffer sizes

▶ There are many ways of adding the concept of time to Petri nets and finite automata.
In the following, we present one specific model.

Discrete Event Models with Time

What can you do with a timed model?

Verify timed properties

- How long does it take until a certain event happens?
- What is the minimum time between two events?

Discrete Event Models with Time

What can you do with a timed model?

Verify timed properties

- How long does it take until a certain event happens?
- What is the minimum time between two events?

Simulate the model

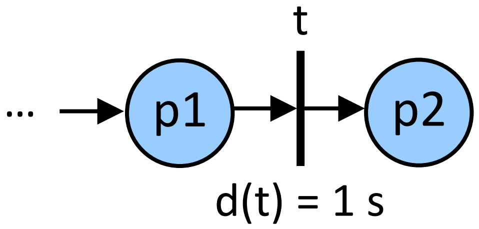
- Given a specific input, how does the system state evolve over time?
- Is the resulting trace of execution what we had in mind?

Definition

Simulation

Time Petri Net

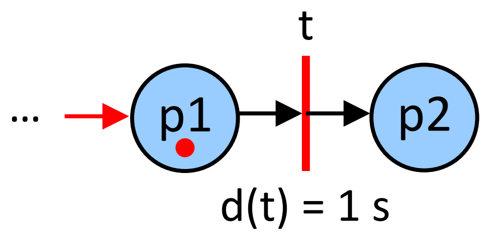
We define a delay function $d: T \rightarrow R$
that determines the delay between the activation of a transition t and its firing.



- Repeated calls may lead to the same value — constant delay
  or to different ones every time. — values of some random variable
- The function is called for every new activation of t and determines the time until t fires.

Time Petri Net

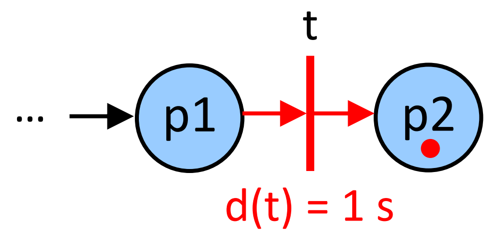
We define a delay function $d: T \rightarrow R$
that determines the delay between the activation of a transition t and its firing.



- Repeated calls may lead to the same value — constant delay
  or to different ones every time. — values of some random variable
- The function is called for every new activation of t and determines the time until t fires.

Time Petri Net

We define a delay function $d: T \rightarrow R$
that determines the delay between the activation of a transition t and its firing.

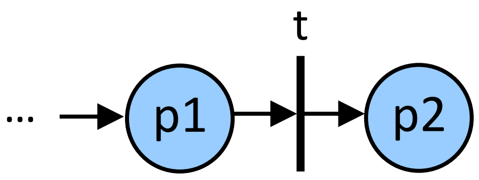


- Repeated calls may lead to the same value or to different ones every time.
  constant delay
  values of some random variable
- The function is called for every new activation of t and determines the time until t fires.

Transition t is activated if all its input places p have at least w(p,t) tokens
- Adding a new token while t is activated does not reactivate t

Time Petri Net

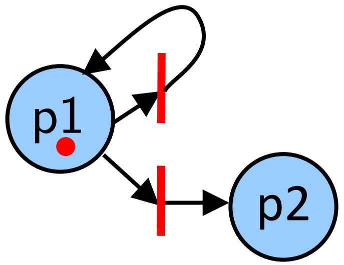
We define a delay function $d: T \rightarrow R$
that determines the delay between the activation of a transition t and its firing.



- Repeated calls may lead to the same value — constant delay
  or to different ones every time. — values of some random variable
- The function is called for every new activation of t and determines the time until t fires.

Transition t is activated if all its input places p have at least w(p,t) tokens

- Adding a new token while t is activated does not reactivate t

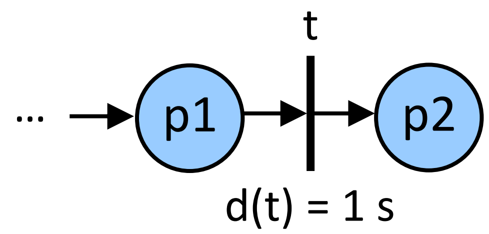


An activation is canceled whenever a token is removed from some input place of t.

- d(t) is called again at the next activation
- New activation can start immediately (all input places still have sufficient tokens)

Time Petri Net

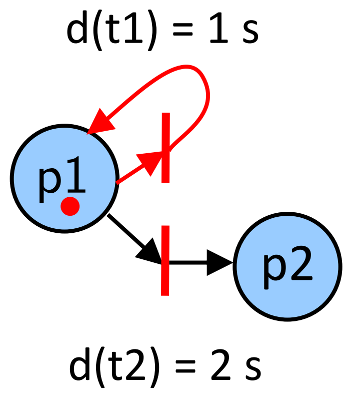
We define a delay function $d: T \rightarrow R$
that determines the delay between the activation of a transition t and its firing.



- Repeated calls may lead to the same value — constant delay
  or to different ones every time. — values of some random variable
- The function is called for every new activation of t and determines the time until t fires.

Transition t is activated if all its input places p have at least w(p,t) tokens
- Adding a new token while t is activated does not reactivate t

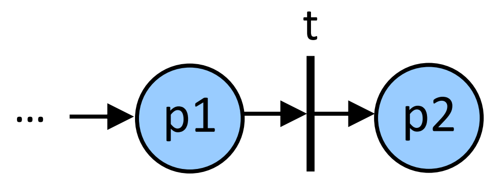
An activation is canceled whenever a token is removed from some input place of t.
- d(t) is called again at the next activation
- New activation can start immediately (all input places still have sufficient tokens)



t2 is reactivated:
it will never fire!
(same if 2 tokens in p1)

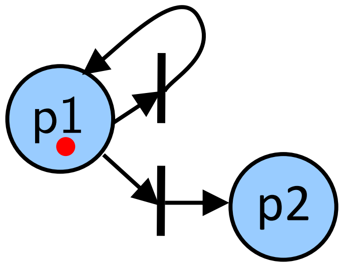
Time Petri Net

We define a delay function $d: T \rightarrow R$
that determines the delay between the activation of a transition t and its firing.



- Repeated calls may lead to the same value — constant delay
  or to different ones every time. — values of some random variable
- The function is called for every new activation of t and determines the time until t fires.

Transition t is activated if all its input places p have at least w(p,t) tokens
- Adding a new token while t is activated does not reactivate t

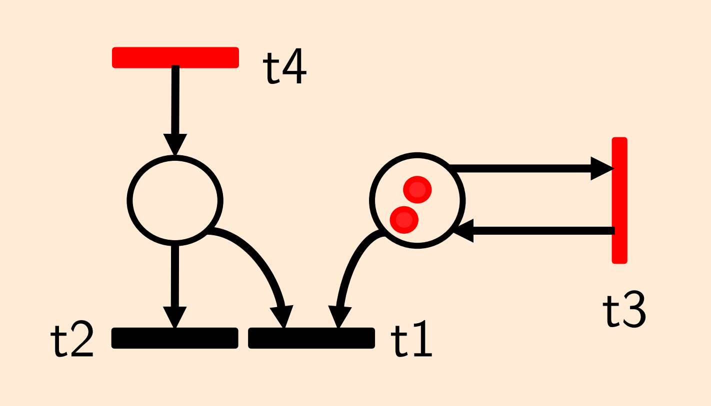


An activation is canceled whenever a token is removed from some input place of t.
- d(t) is called again at the next activation
- New activation can start immediately (all input places still have sufficient tokens)

Only one transition fires at a time (same as with regular Petri nets).
- If two transitions have the same firing time,
  one of them is chosen non-deterministically to fire first.

Time Petri Net

All input places of t contain at least w(p,t) tokens:
t is activated; adding a token does not reactivate t.
Token removed from some input place of t:
cancel activation; call d(t) at next activation.



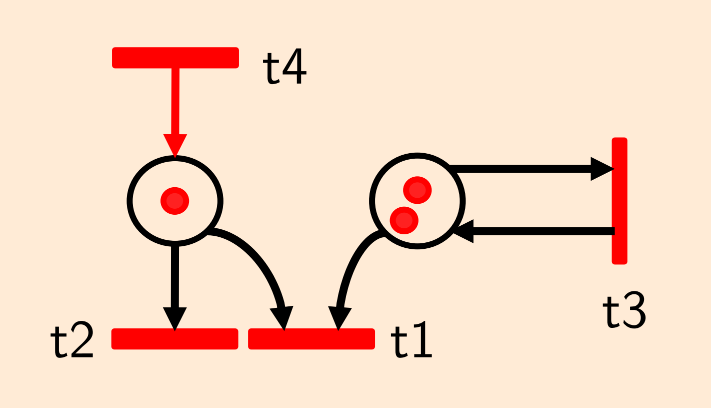
d(t1)

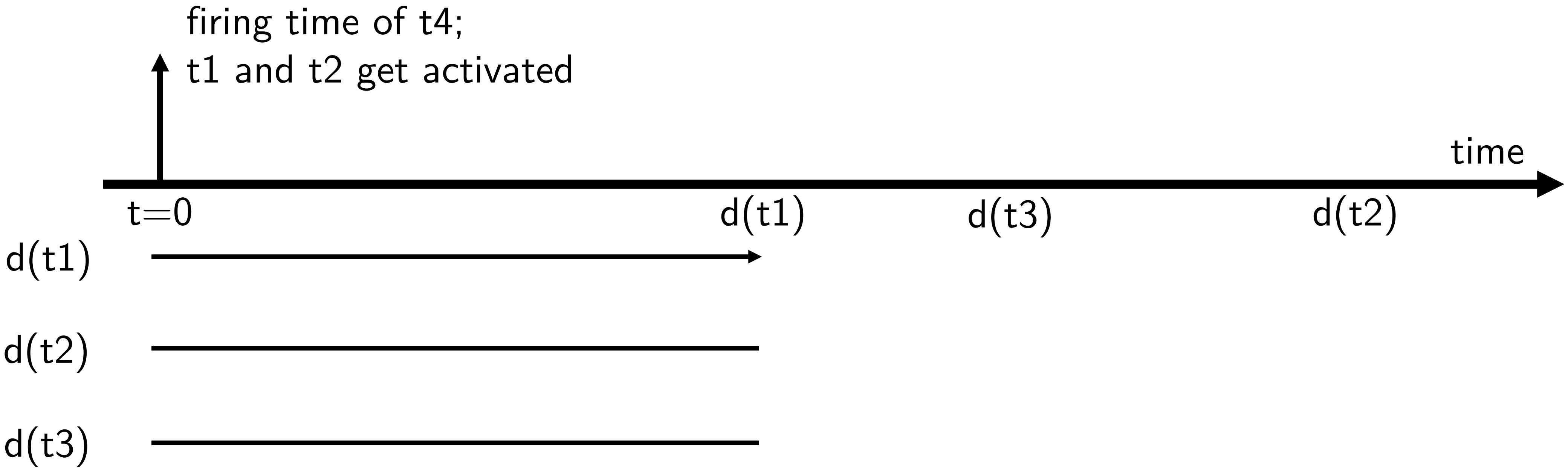
d(t2)

d(t3)

Time Petri Net

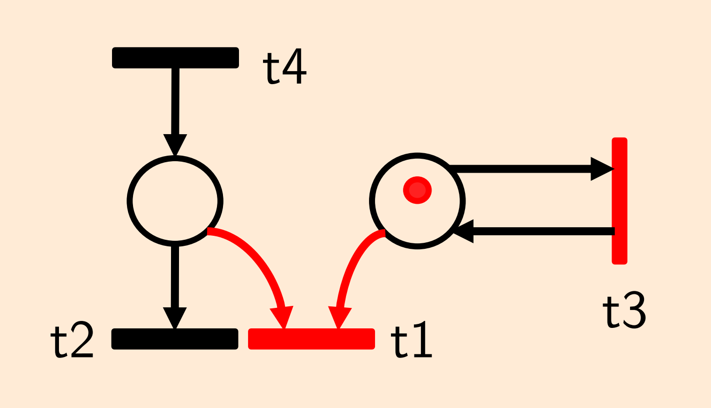
All input places of t contain at least w(p,t) tokens:
t is activated; adding a token does not reactivate t.
Token removed from some input place of t:
cancel activation; call d(t) at next activation.

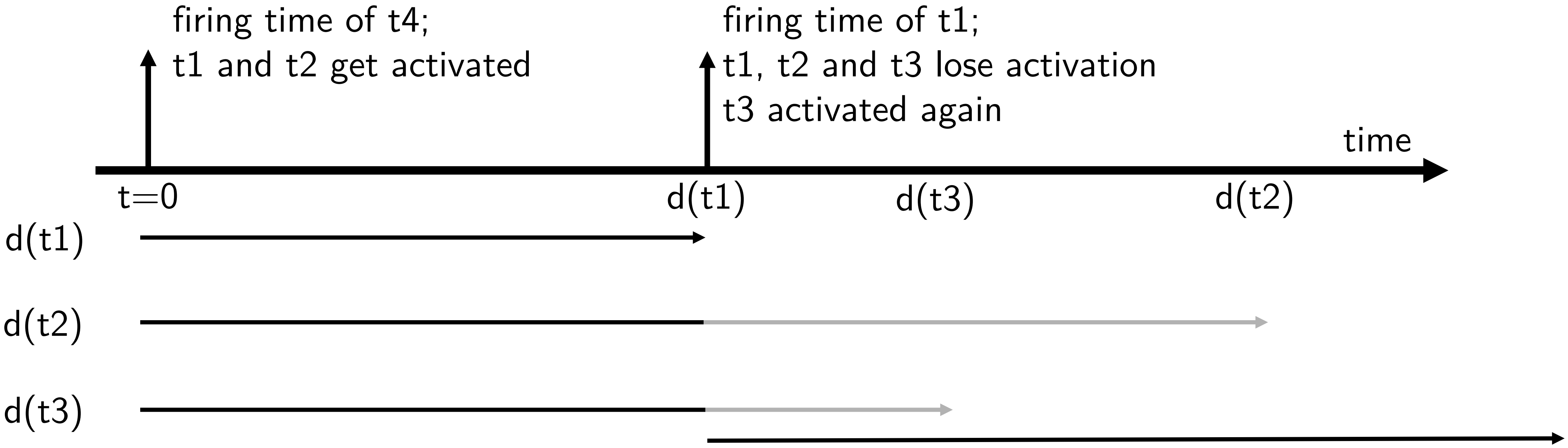




Time Petri Net

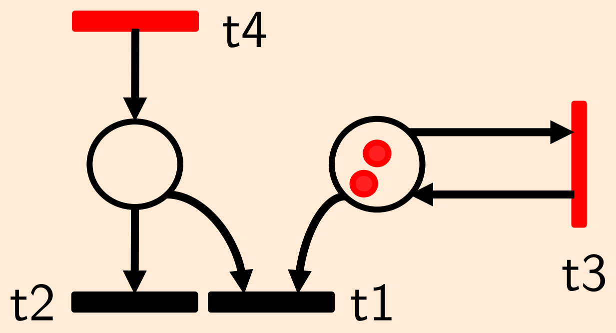
All input places of t contain at least w(p,t) tokens:
t is activated; adding a token does not reactivate t.
Token removed from some input place of t:
cancel activation; call d(t) at next activation.





Time Petri Net

All input places of t contain at least w(p,t) tokens:
t is activated; adding a token does not reactivate t.
Token removed from some input place of t:
cancel activation; call d(t) at next activation.



time

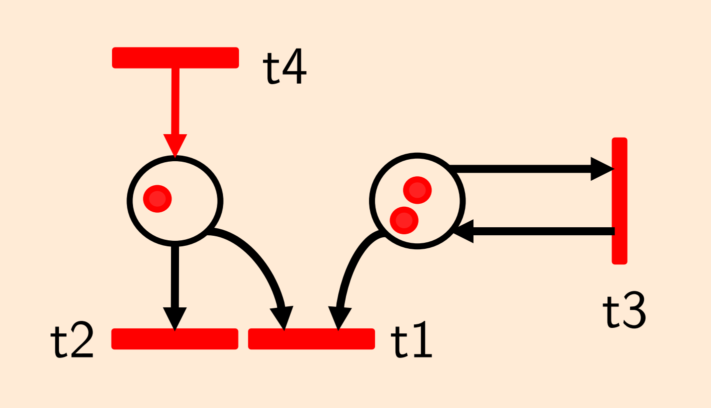
d(t1)

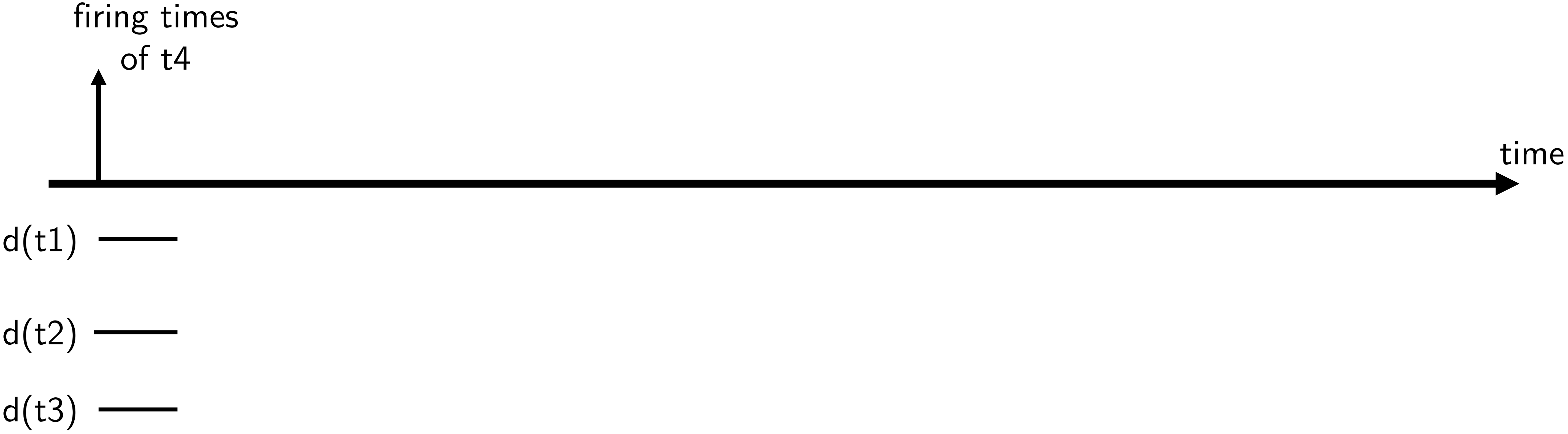
d(t2)

d(t3)

Time Petri Net

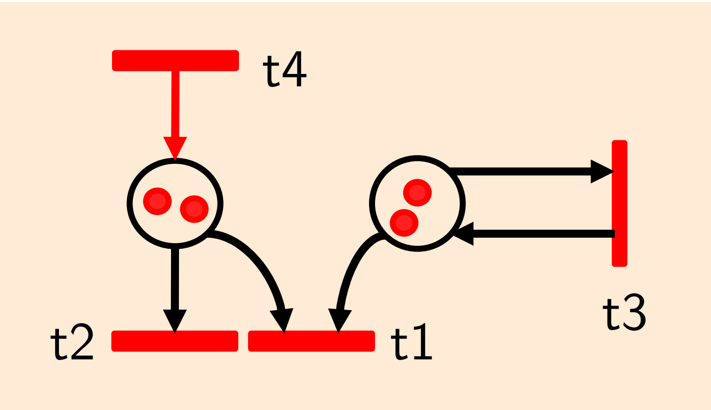
All input places of t contain at least w(p,t) tokens:
t is activated; adding a token does not reactivate t.
Token removed from some input place of t:
cancel activation; call d(t) at next activation.

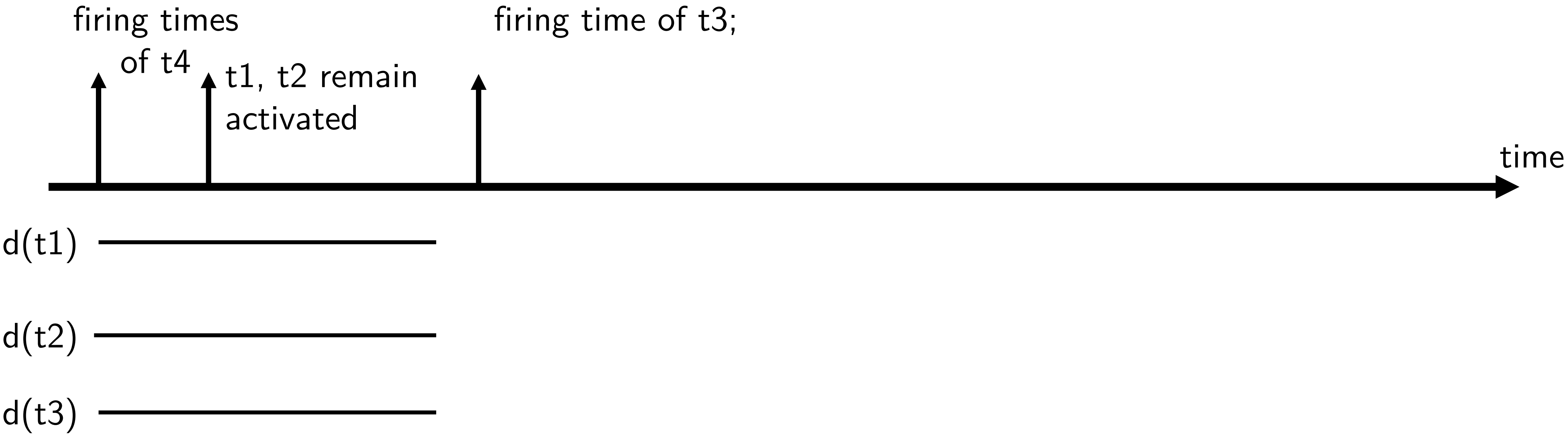




Time Petri Net

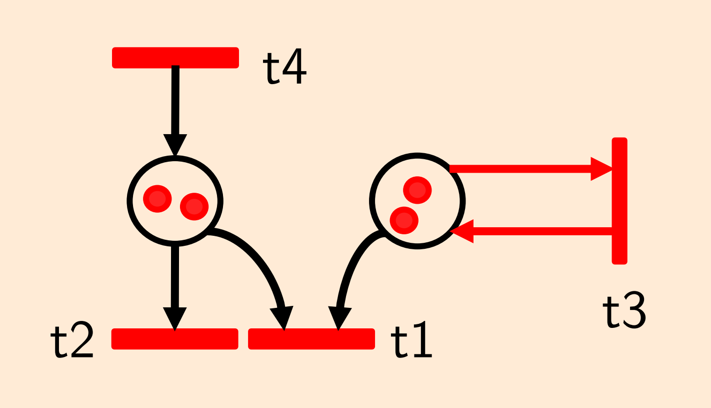
All input places of t contain at least w(p,t) tokens:
t is activated; adding a token does not reactivate t.
Token removed from some input place of t:
cancel activation; call d(t) at next activation.

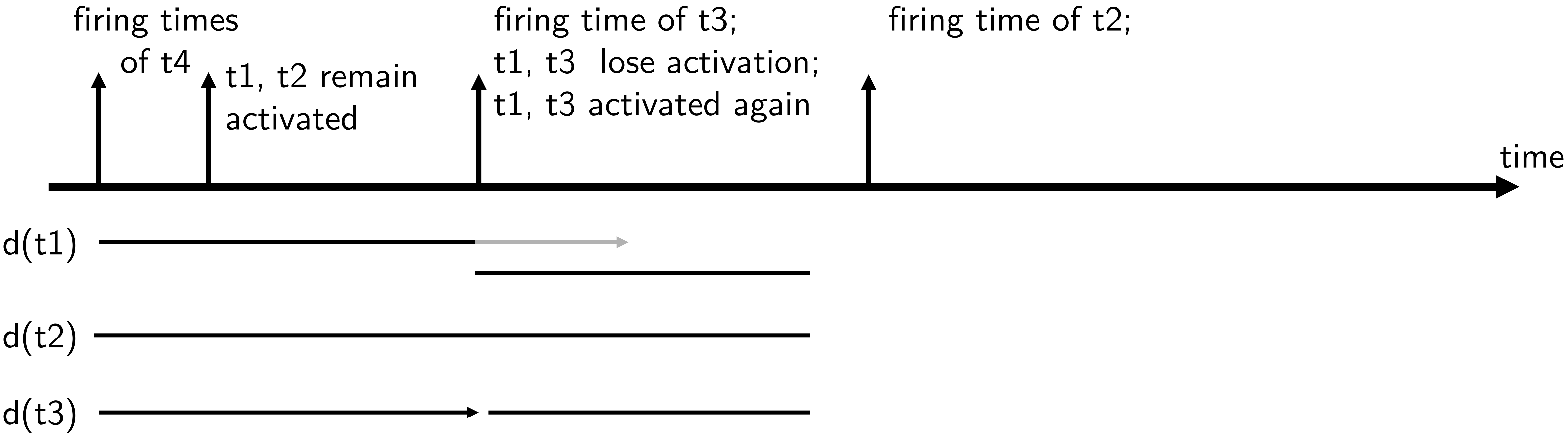




Time Petri Net

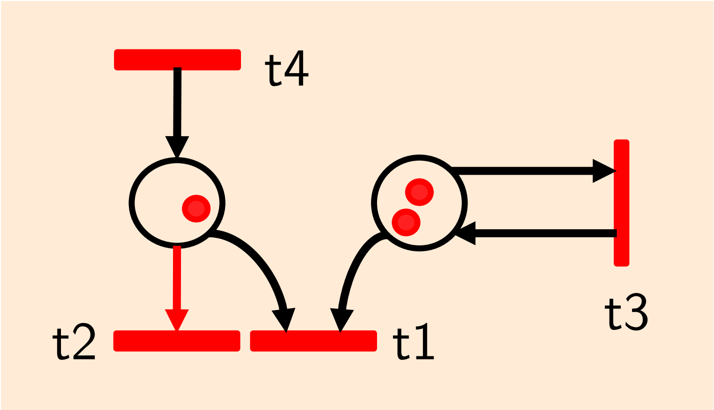
All input places of t contain at least w(p,t) tokens:
t is activated; adding a token does not reactivate t.
Token removed from some input place of t:
cancel activation; call d(t) at next activation.

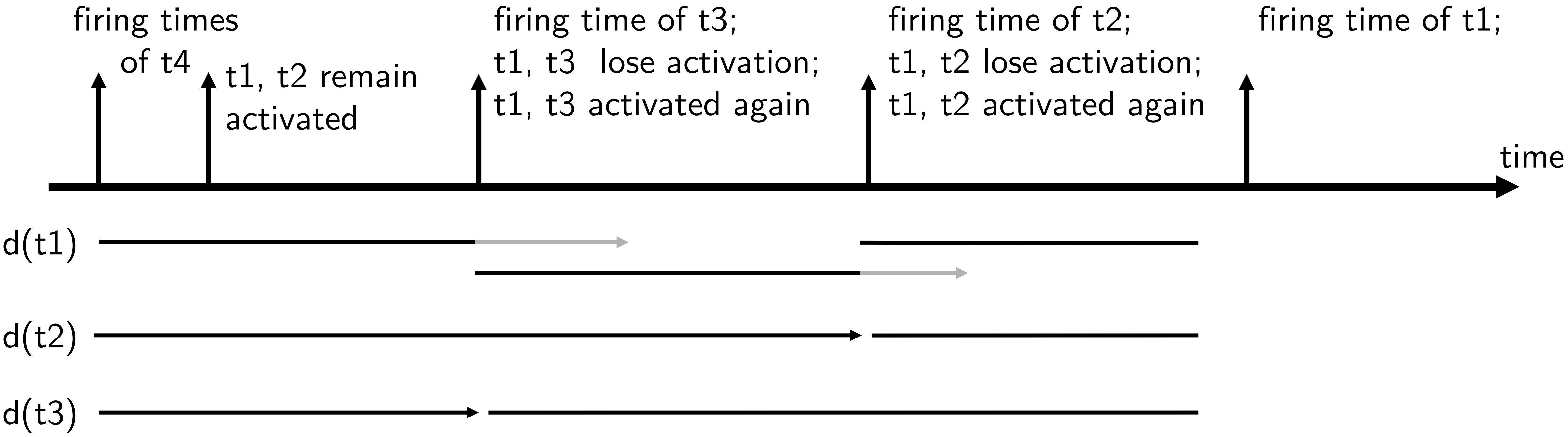




Time Petri Net

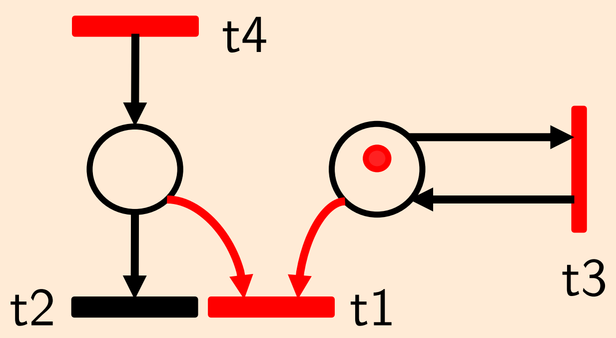
All input places of t contain at least w(p,t) tokens:
t is activated; adding a token does not reactivate t.
Token removed from some input place of t:
cancel activation; call d(t) at next activation.

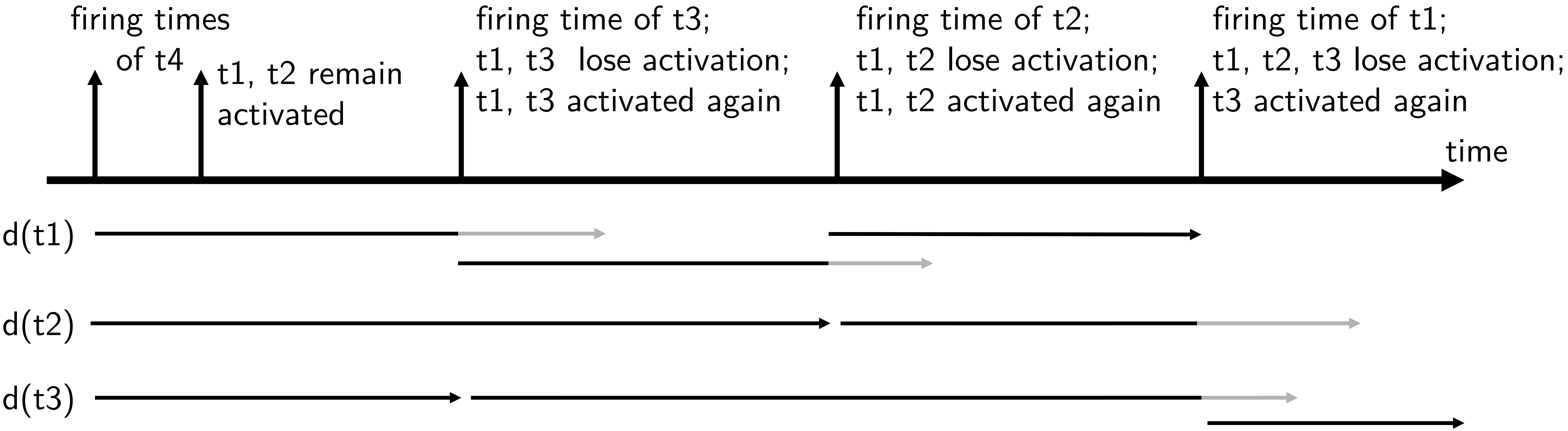




Time Petri Net

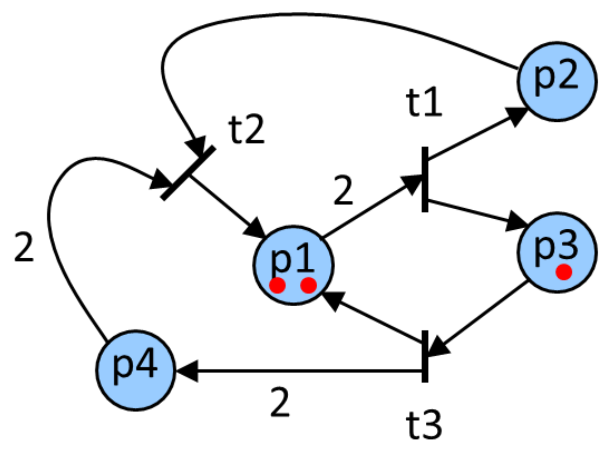
All input places of t contain at least w(p,t) tokens:
t is activated; adding a token does not reactivate t.
Token removed from some input place of t:
cancel activation; call d(t) at next activation.





Time Petri Net

- The time when a transition t fires is called the firing time.
- A time Petri net can be regarded as a generator for firing times of its transitions.



$d(t1) = 1$

$d(t2) = 2$

$d(t3) = 3$

t1 → $\{1, 6, 9, 12, \ldots\}$

t2 → $\{5, 8, 11, 13, \ldots\}$

t3 → $\{3, 6, 9, 12, \ldots\}$

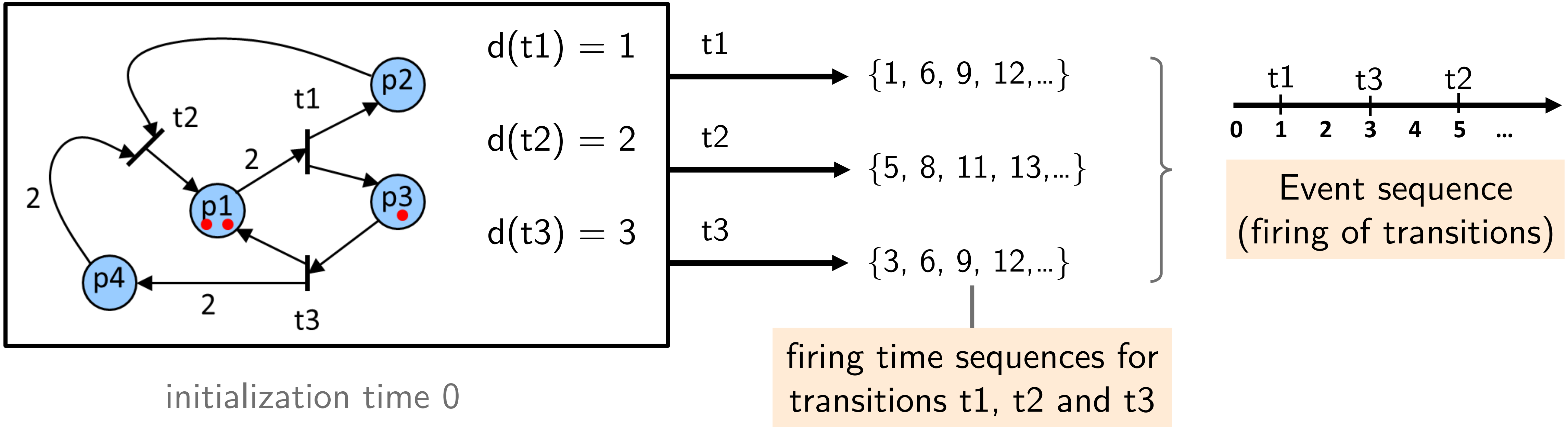
initialization time 0

firing time sequences for transitions t1, t2 and t3

- How do we get the firing times? By simulation!

Time Petri Net

- The time when a transition t fires is called the firing time.
- A time Petri net can be regarded as a generator for firing times of its transitions.



initialization time 0

- How do we get the firing times? By simulation!

Definition

Simulation

Simulation Principle

This simulation principle holds in one form or another for any simulator of timed discrete event models.

The simulation is based on the following basic principles.

1. The simulator maintains a set L of currently activated transitions and their firing times. We call L the event list from now on.
2. A transition with the earliest firing time is selected and fired. The state of the Petri net as well as the current simulation time is updated accordingly.
3. All transitions that lost their activation during the state transition are removed from the event list L.
4. Afterwards, all transitions that are newly activated are added to the event list L together with their firing times.
5. Then we continue with 2. unless the event list L is empty.

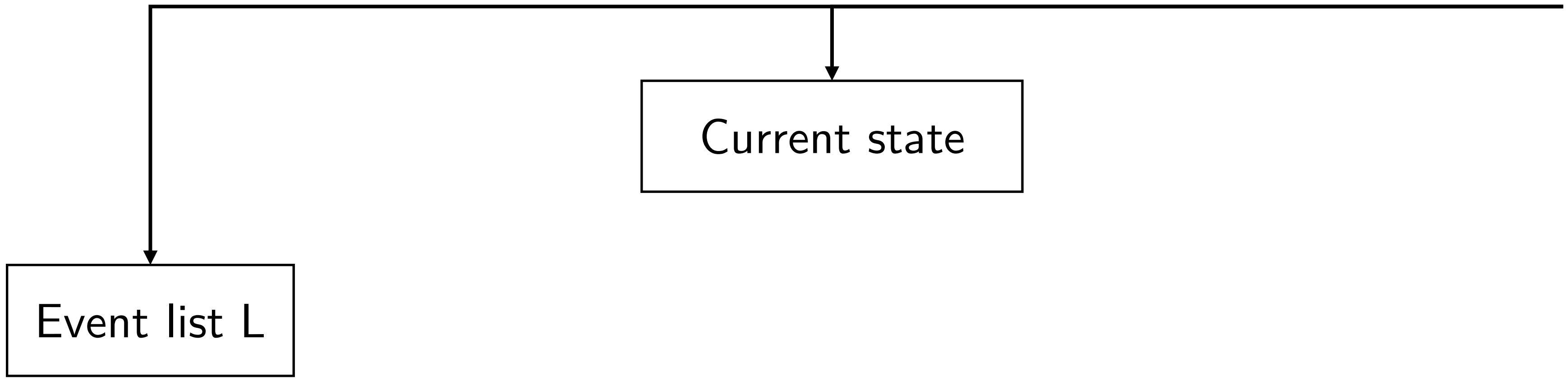
Add tuple to L when t_i is activated:

$$L = \{ (t_i, \tau_i) \}$$

$$\tau_i = \tau + d(t_i)$$

τ: current simulation time (activation time of t_i)

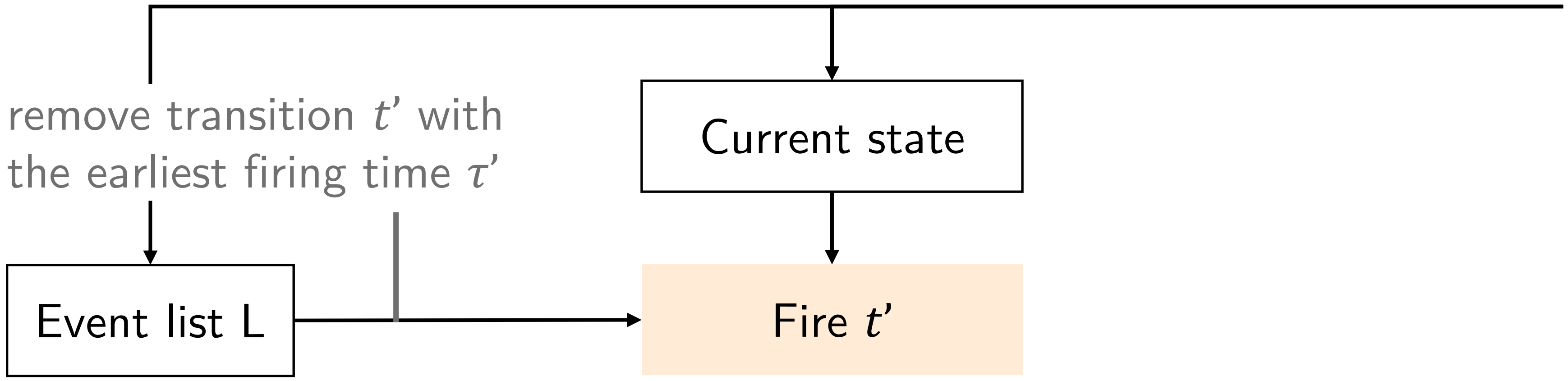
Simulation Principle



Initialization

- Event list L
- State M
- Simulation time τ

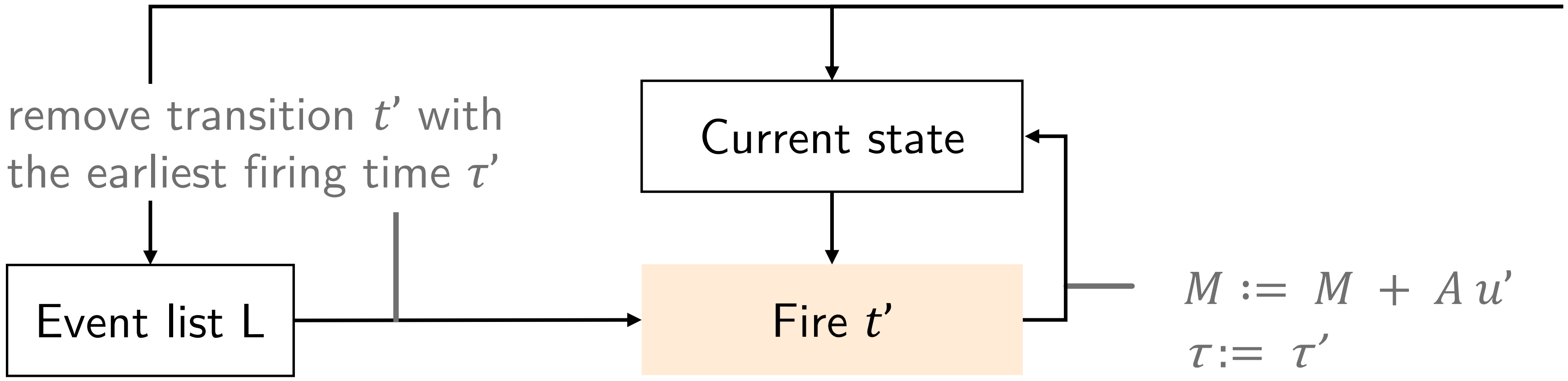
Simulation Principle



Initialization

- Event list L
- State M
- Simulation time τ

Simulation Principle



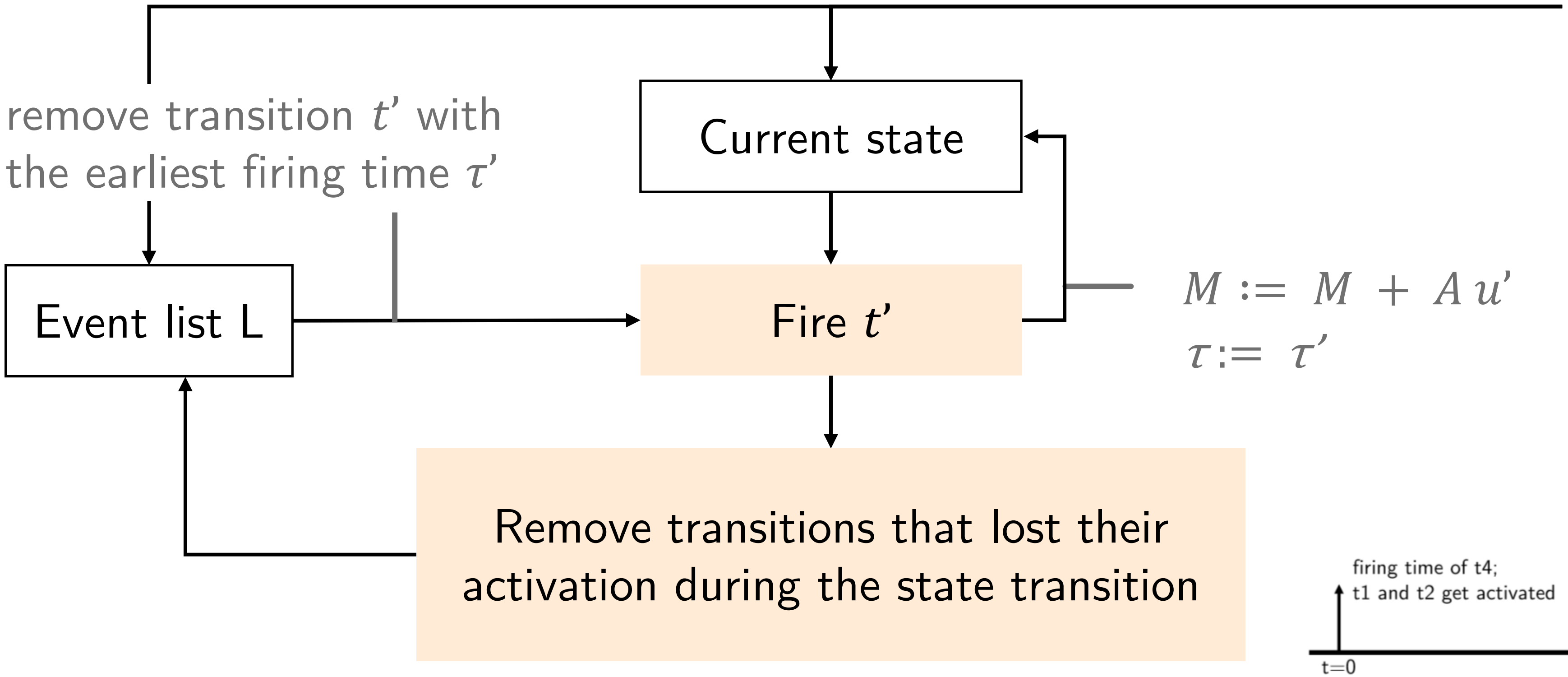
Initialization

- Event list L
- State M
- Simulation time τ

Update

- state
- simulation time

Simulation Principle

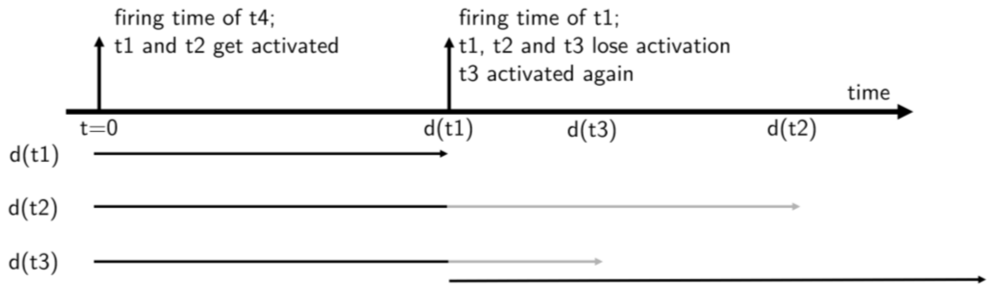


Initialization

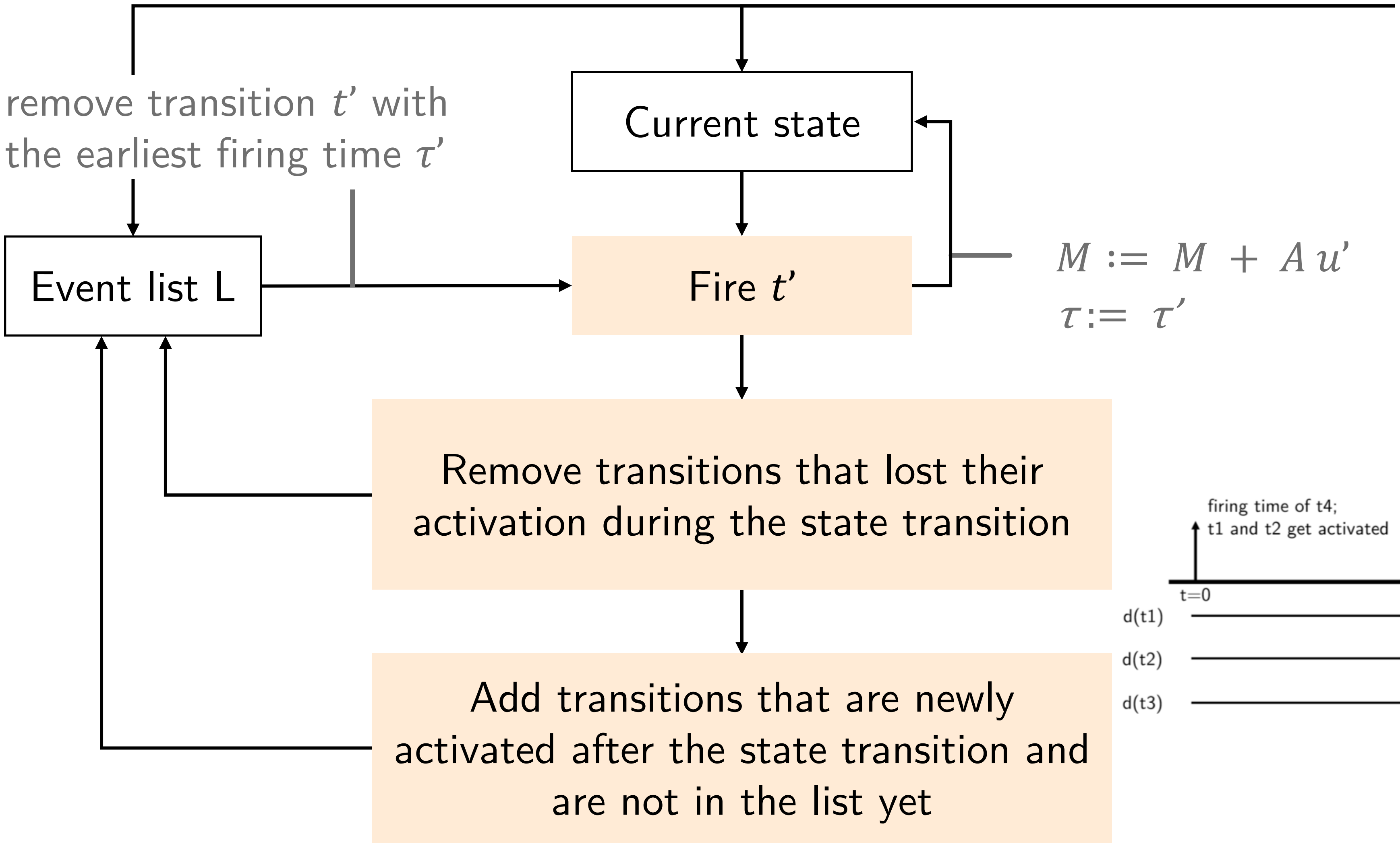
- Event list L
- State M
- Simulation time τ

Update

- state
- simulation time



Simulation Principle

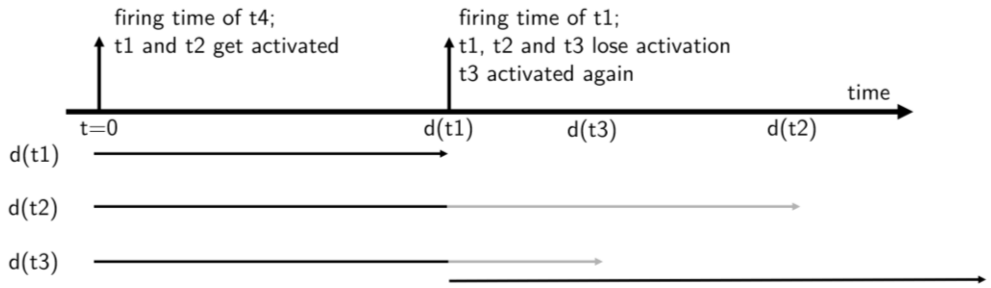


Initialization

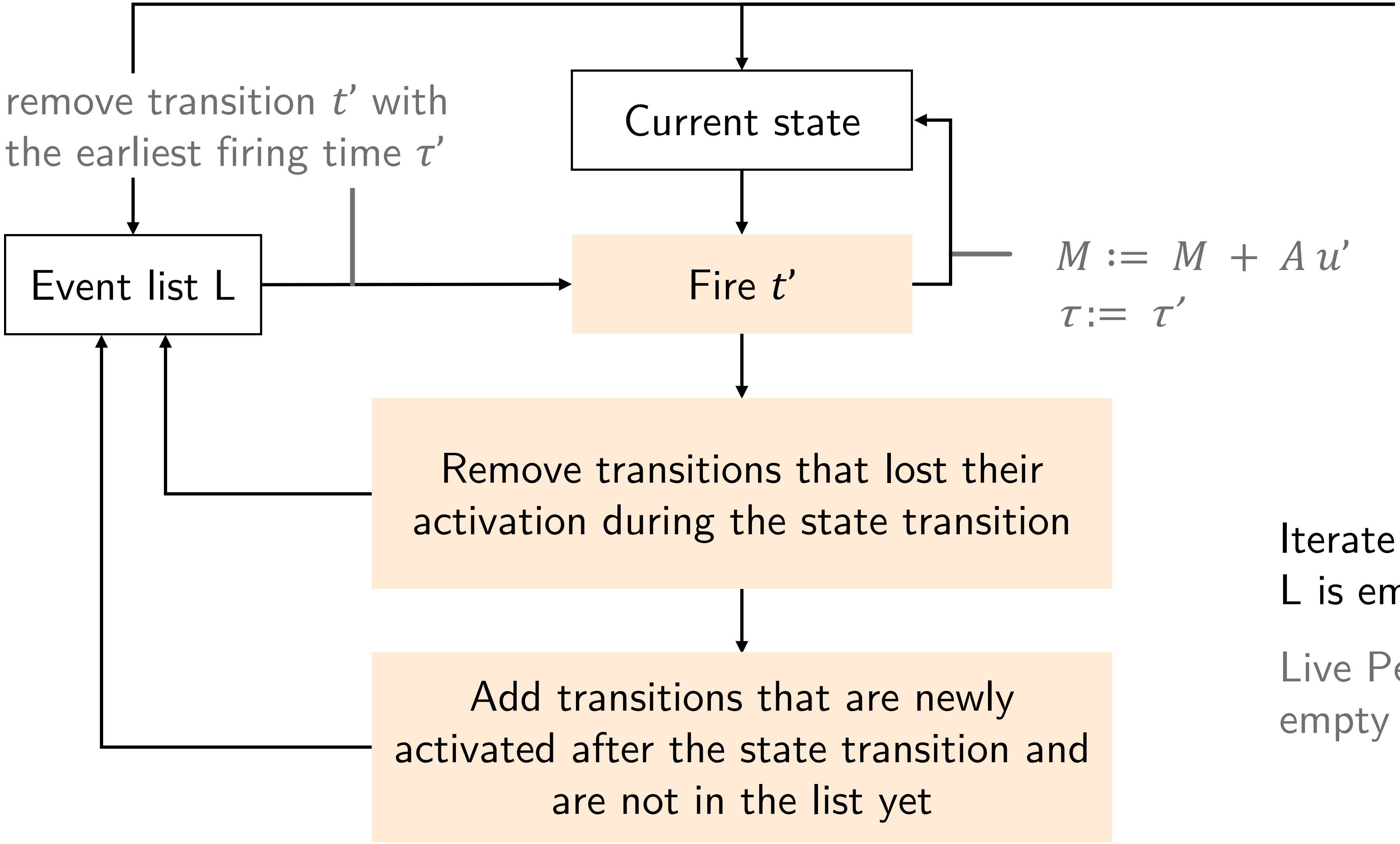
- Event list L
- State M
- Simulation time τ

Update

- state
- simulation time



Simulation Principle



Initialization

- Event list L
- State M
- Simulation time τ

Update

- state
- simulation time

Iterate until L is empty

Live Petri net: list never empty (infinite simulation)

Simulation Algorithm (1)

Initialization:

- Set the initial simulation time $\tau := 0$
- Set the current state to $M := M_0$
- For each activated transition t, add the event $(t, \tau + d(t))$ to the event list L

Determine and remove current event:

- Determine a firing event (t', τ') with the earliest firing time:

$$\forall 1 \leq i \leq N \ : \ \tau' \leq \tau_i \quad \text{where} \quad L = \{(t_1, \tau_1), (t_2, \tau_2), \cdots, (t_N, \tau_N)\}$$

- Remove event (t', τ') from the event list L: $L := L \setminus \{(t', \tau')\}$

Update current simulation time: Set current simulation time $\tau := \tau'$

Update token distribution M:

- Suppose that the firing transition has index j, i.e. $t_j = t'$. Then, the firing vector is:

$$u' = [\ 0 \ \cdots \ 0 \ \underset{j}{1} \ 0 \ \cdots \ 0\ ]^t$$

- Update current state $M := M + A\,u'$

Simulation Algorithm (2)

Remove transitions from L that lost activation:

- Determine the set of places S' from which at least one token was removed during the state transition caused by t':

$$S' = \{p \,|\, (p, t') \in F\}$$

- Remove from event list L all transitions in T' that lost their activation due to this token removal:

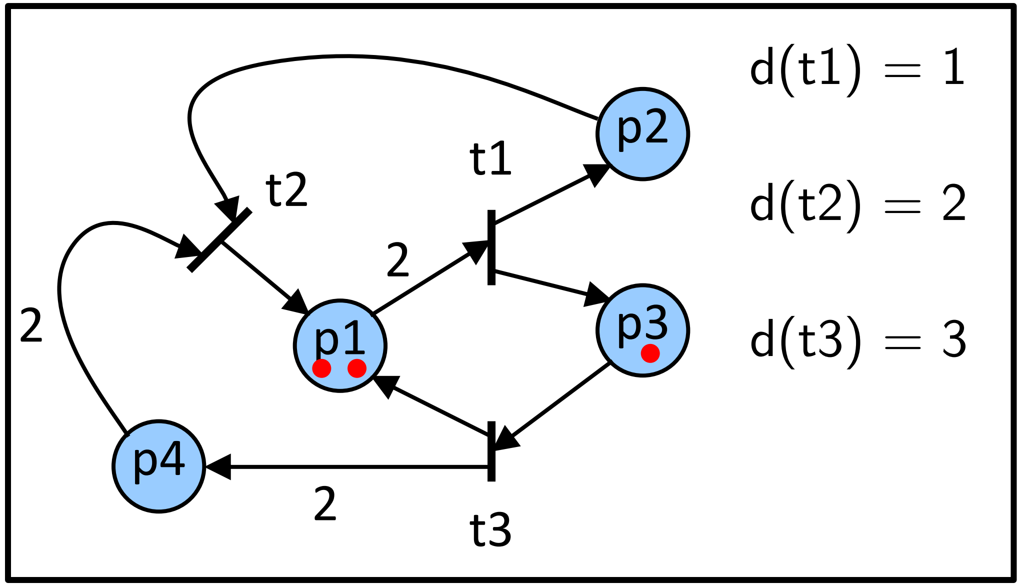
$$T' = \{t \,|\, (p, t) \in F \wedge p \in S'\}$$

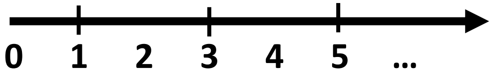
Add all transitions to event list L that are activated but not in L yet:

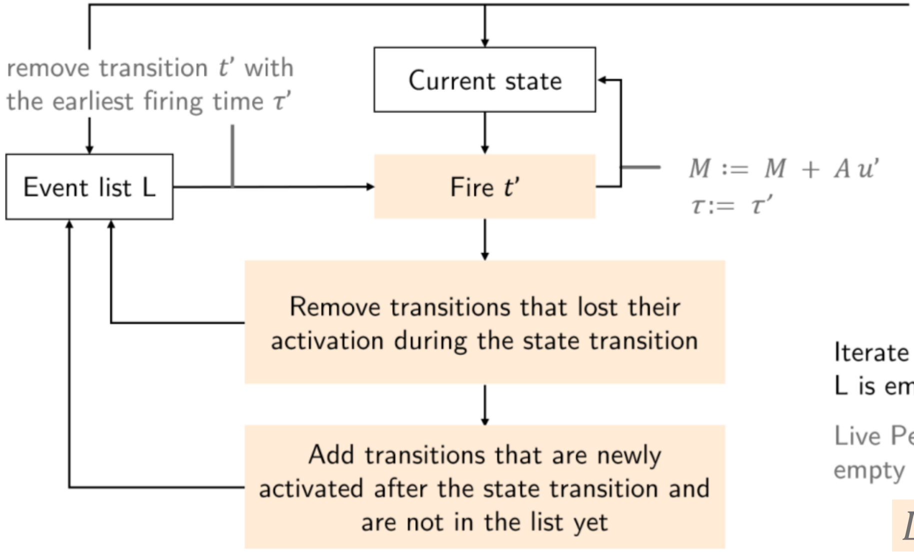
- If some transition t with $M(p) \geq W(p, t)$ for all $(p, t) \in F$ is not in L, then add $(t, \tau + d(t))$ to the event list:

$$L := L \cup \{(t, \tau + d(t))\}$$

Simulation Example







Initialization
- Event list L
- State M
- Simulation time τ

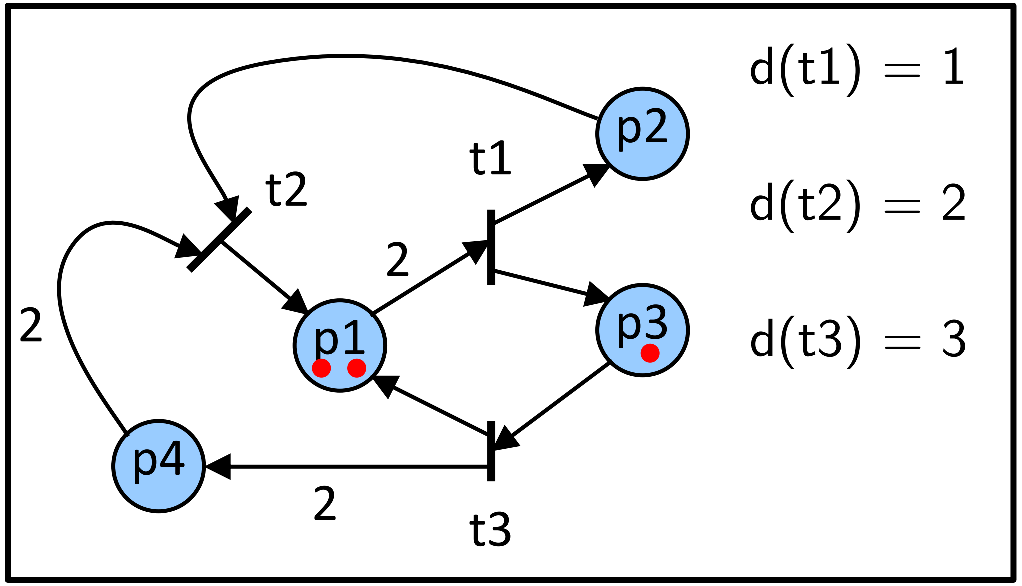
Update
- state
- simulation time

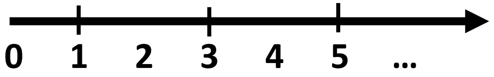
Iterate until L is empty

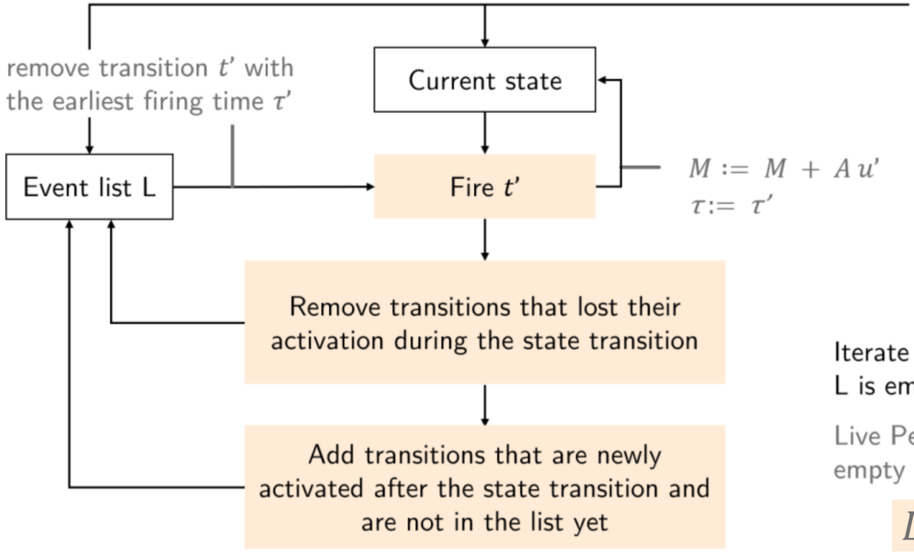
Live Petri net: list never empty (infinite simulation)

$L = \{ (t_i, \tau + d(t_i)) \}$

Simulation Example







Initialization
- Event list L
- State M
- Simulation time τ

Update
- state
- simulation time

Iterate until L is empty

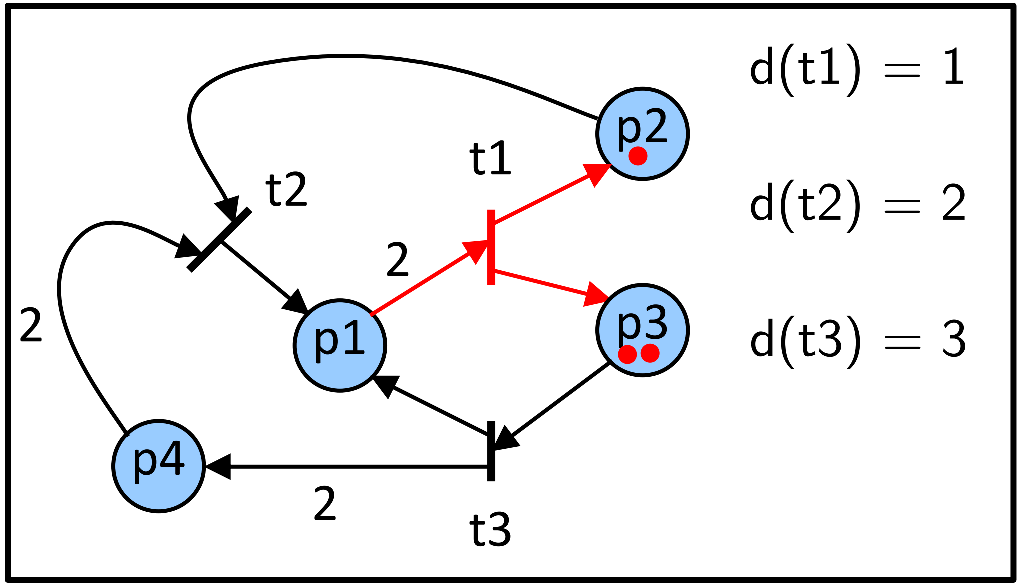
Live Petri net: list never empty (infinite simulation)

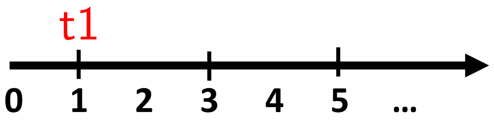
$L = \{ (t_i, \tau + d(t_i)) \}$

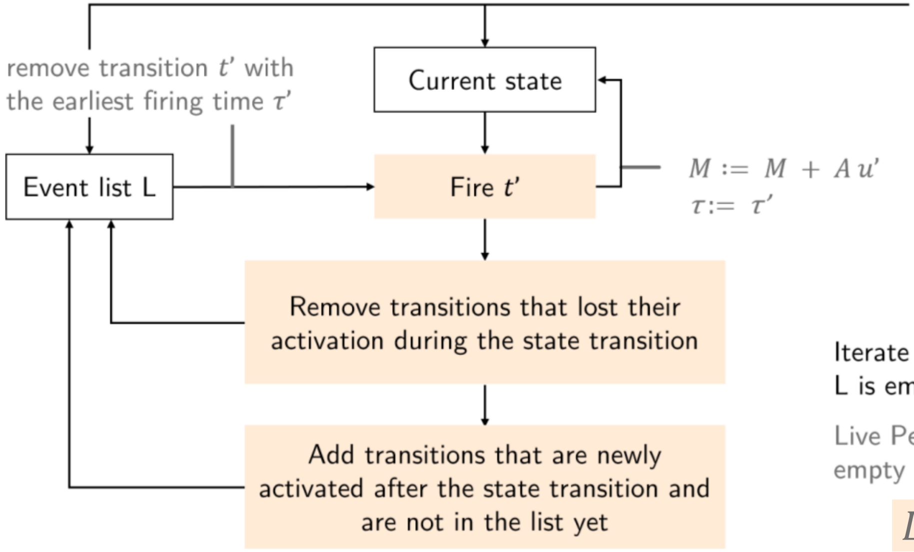
$\tau = 0$:

$M = [2\ 0\ 1\ 0]$ $\quad L = \{ (t_1, 1), (t_3, 3) \}$

Simulation Example







Initialization
- Event list L
- State M
- Simulation time τ

Update
- state
- simulation time

Iterate until L is empty

Live Petri net: list never empty (infinite simulation)

$L = \{ (t_i, \tau + d(t_i)) \}$

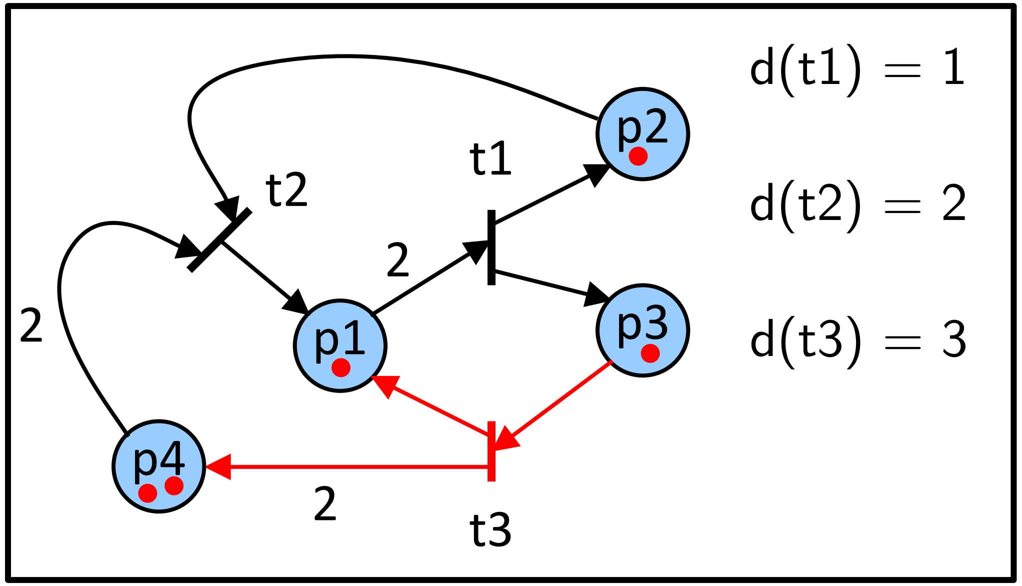
$\tau = 0$:

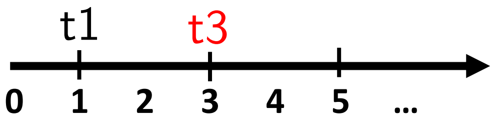
$M = [2\ 0\ 1\ 0]$ $\quad L = \{ (t_1, 1), (t_3, 3) \}$

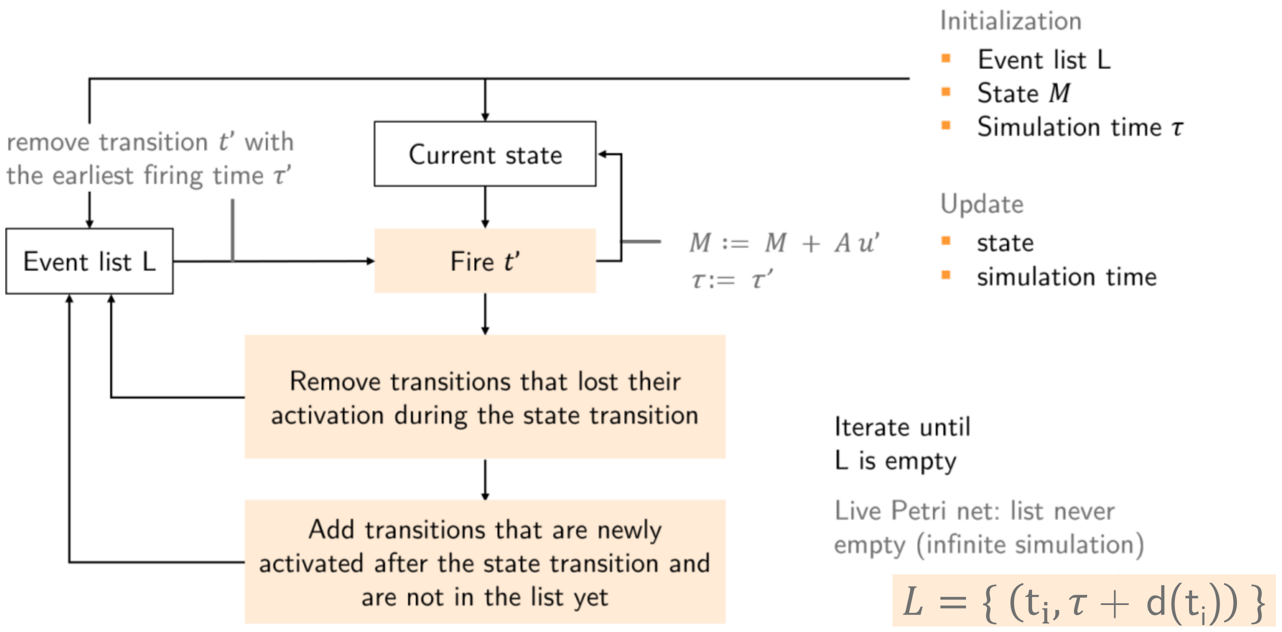
$\tau = 1$:

$M = [0\ 1\ 2\ 0]$ $\quad L = \{ (t_3, 3) \}$

Simulation Example







$\tau = 0$:

$M = [2\ 0\ 1\ 0]$ $\quad L = \{ (t_1, 1), (t_3, 3)\}$

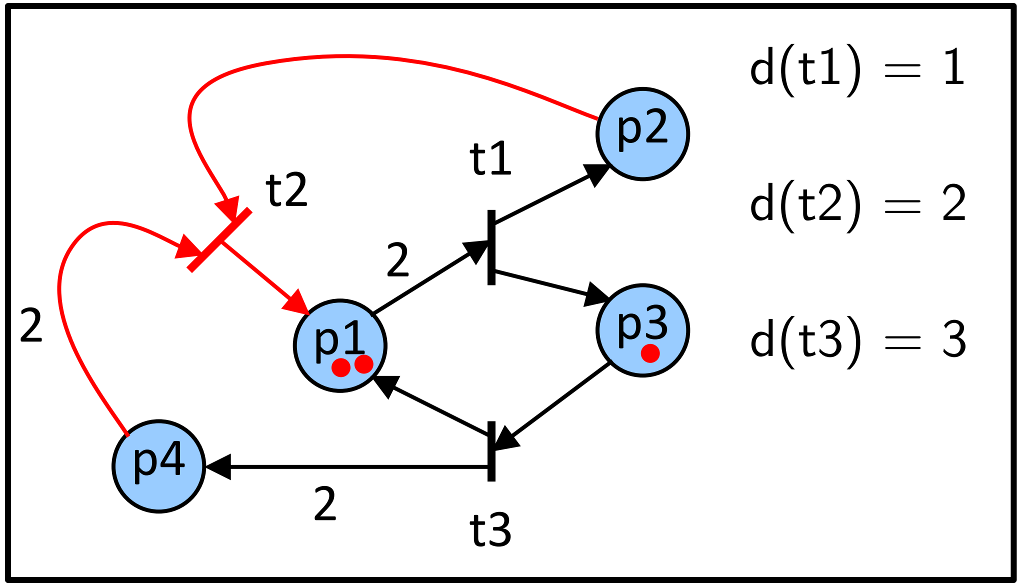
$\tau = 1$:

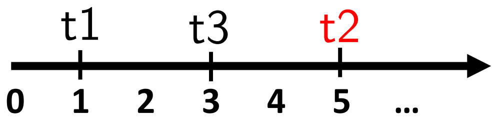
$M = [0\ 1\ 2\ 0]$ $\quad L = \{(t_3, 3)\}$

$\tau = 3$:

$M = [1\ 1\ 1\ 2]$ $\quad L = \{(t_3, 6), (t_2, 5)\}$

Simulation Example

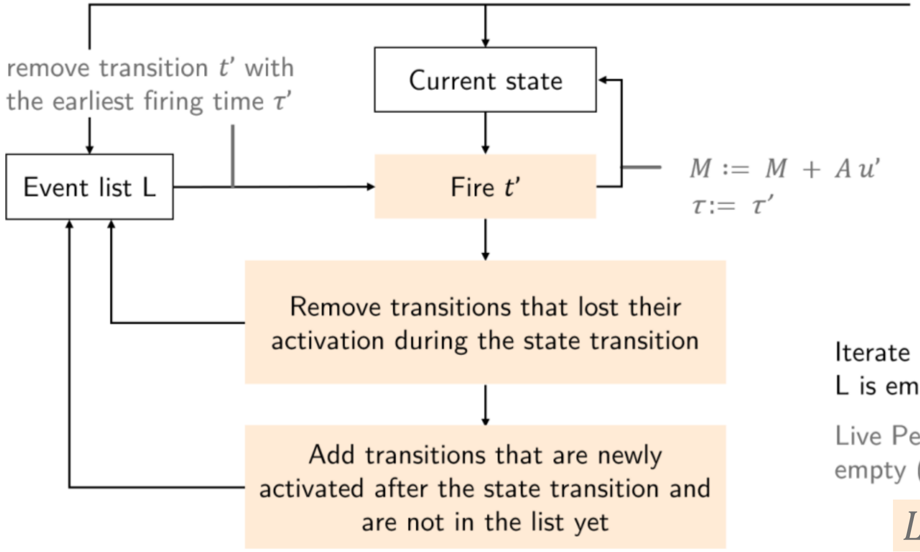




If two transitions have the same firing time,
one of them is chosen non-deterministically to fire first.

Initialization
- Event list L
- State M
- Simulation time τ

Update
- state
- simulation time



Iterate until
L is empty

Live Petri net: list never
empty (infinite simulation)

$L = \{ (t_i, \tau + d(t_i)) \}$

$\tau = 0$:

$M = [2\ 0\ 1\ 0]$ $\quad L = \{ (t_1, 1), (t_3, 3) \}$

$\tau = 1$:

$M = [0\ 1\ 2\ 0]$ $\quad L = \{ (t_3, 3) \}$

$\tau = 3$:

$M = [1\ 1\ 1\ 2]$ $\quad L = \{ (t_3, 6), (t_2, 5) \}$

$\tau = 5$:

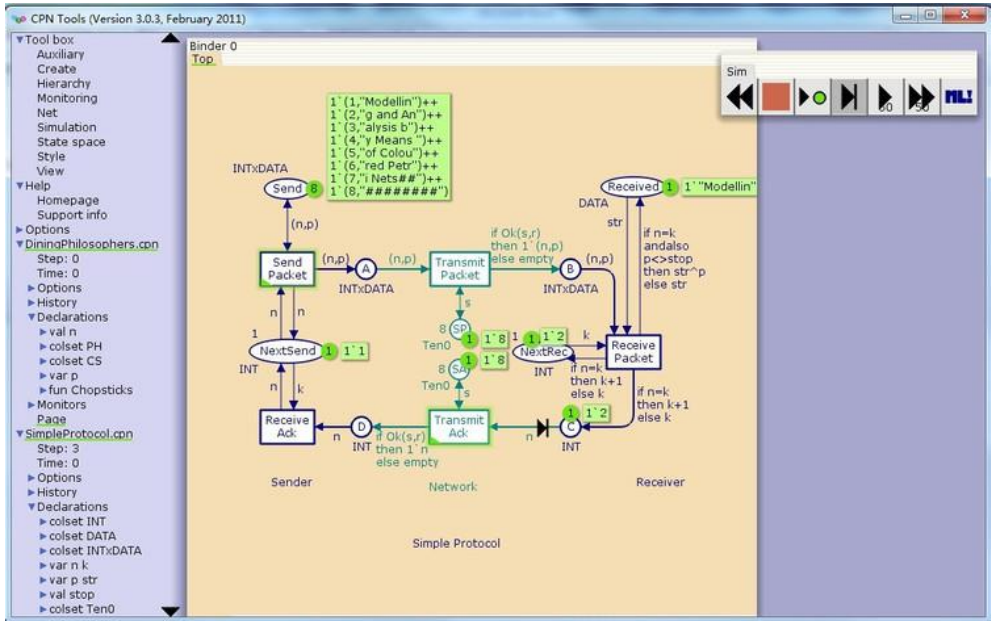
$M = [2\ 0\ 1\ 0]$ $\quad L = \{ (t_3, 6), (t_1, 6) \}$

Petri Net Simulators

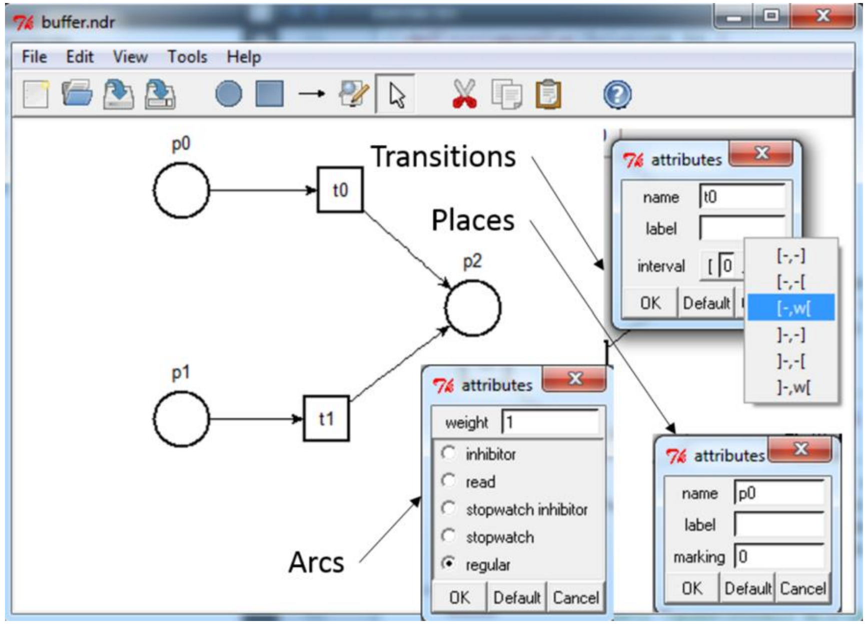
There are many simulators available

An overview
www.informatik.uni-hamburg.de/TGI/PetriNets/tools/quick.html

Examples



CPN Tools



TINA

Discrete Event Models with Time

In many discrete event systems, time is an important factor.

- queuing systems
- computer systems
- digital circuits
- workflow management
- business processes

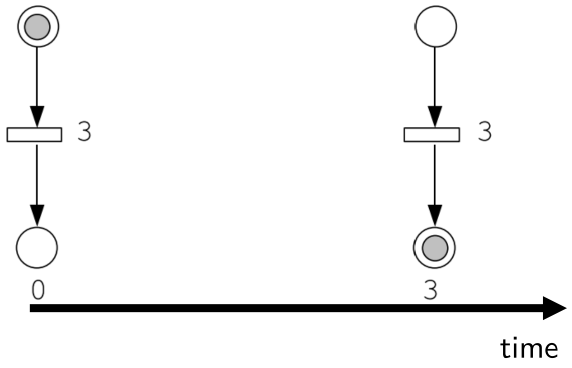
Based on a **timed discrete event model,** we would like to determine properties:

- delay
- throughput
- execution rate
- resource load
- buffer sizes

▶ There are many ways of adding the concept of time to Petri nets and finite automata.
In the following, we present one specific model. — What are the others?

There are mainly three ways to count time

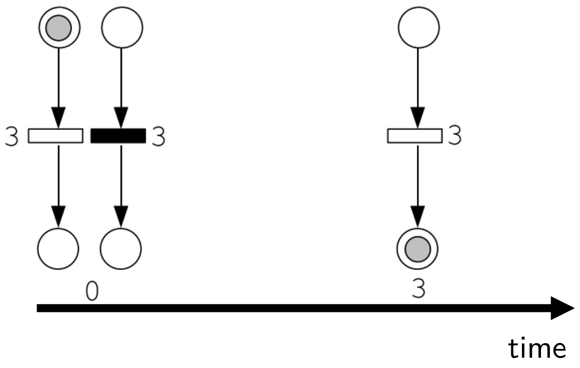
Delay on the transition firing



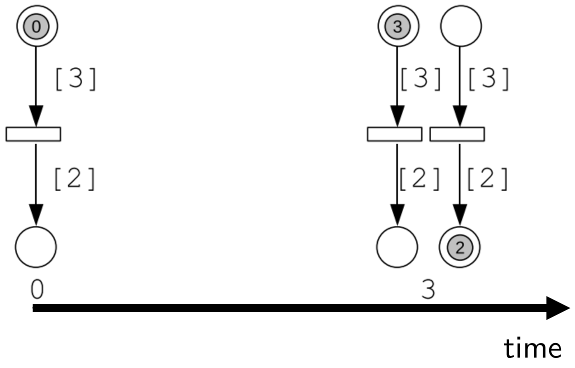
Time Petri nets

Covered here

Duration of the transition



Age of the tokens



Time**d** Petri nets

www.lsv.fr/~haddad/disc11-part1.pdf

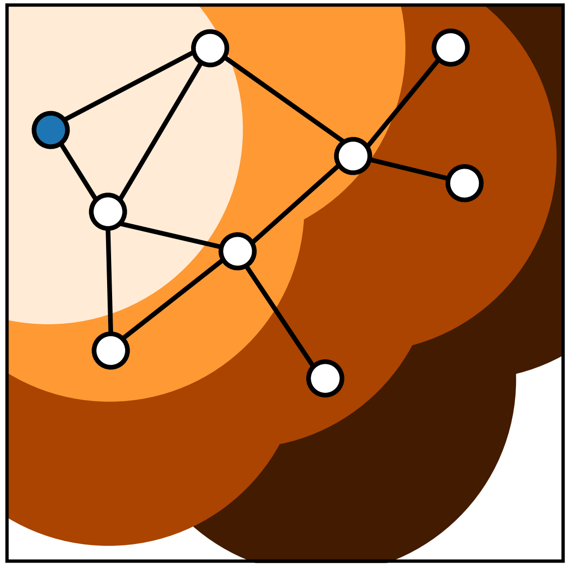
Expressivity and analysis feasibility may vary between the models.

Your turn to practice!
after the break

1. Model arithmetic operations with Petri nets
2. Use a simulator to explore the timed behavior of a simple Petri net model
3. Use a model-checker to adapt a system design

Quick recap

Discrete Event Systems (Part 3)



- How to efficiently explore the state space of DES models? — Set of states & BDDs
- How to formulate temporal properies of interest? — CTL fomulas
- How to formally verify such properties? — Reachability & model-checking
- How to efficiently model concurrency in DES? — Petri nets w/ and w/o time