Business Process Simulation

Lecture 4

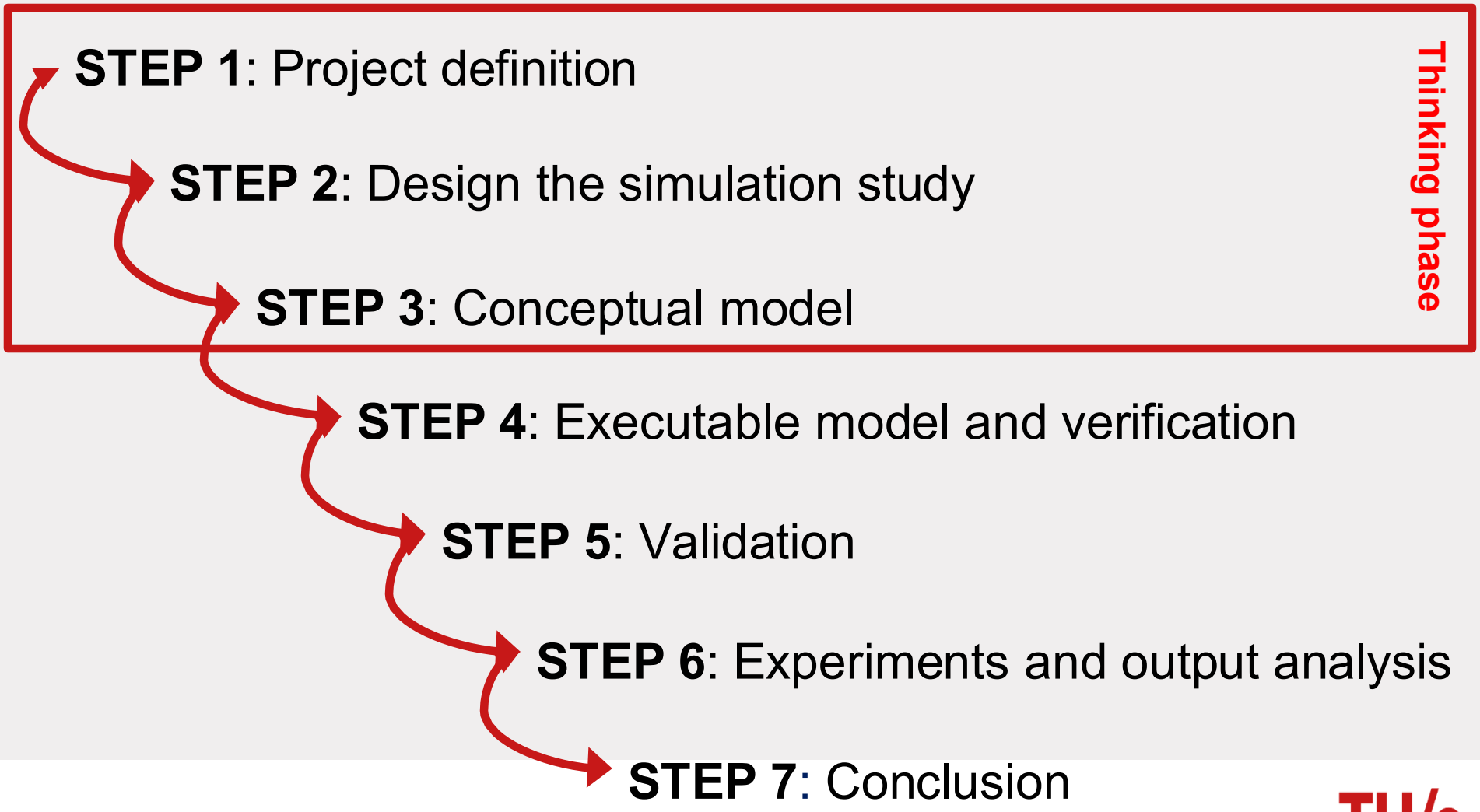
Laura Genga

Department of Industrial Engineering and Innovation Sciences, Information Systems Group

Overview of today's lecture

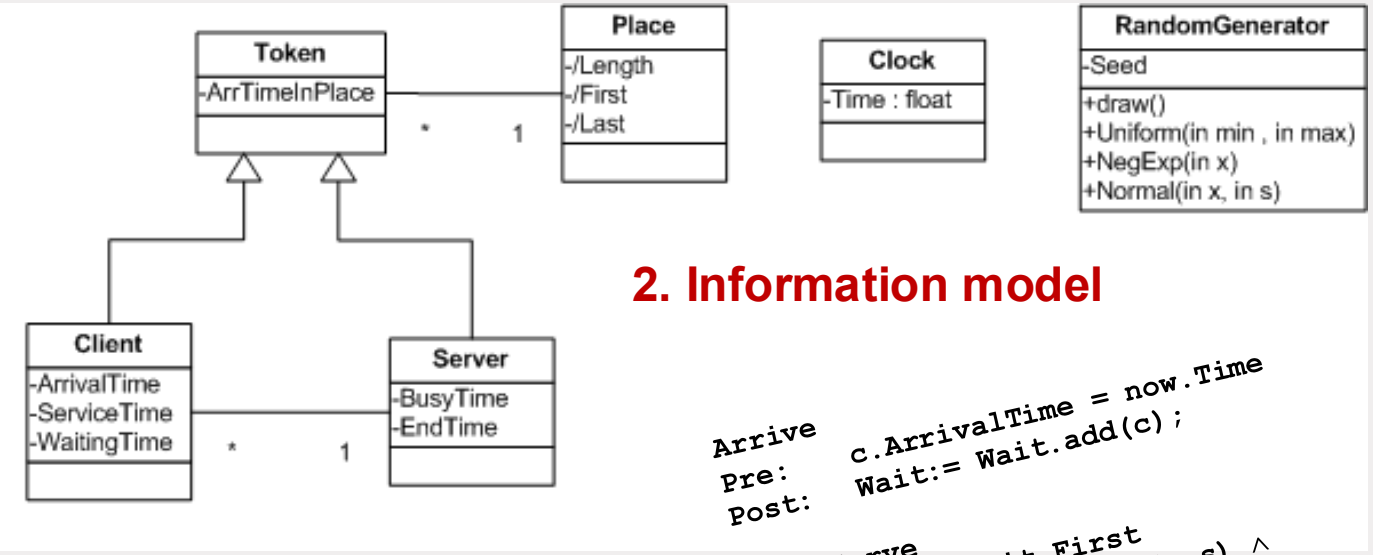
a) Introduction to SimPN
b) Advanced constructs of SimPN and Mapping from conceptual model
c) Solution patterns in SimPN
d) Simulation output and verification

Simulation Methodology (7 steps)



TU/e

Recap STEP 3: Conceptual model

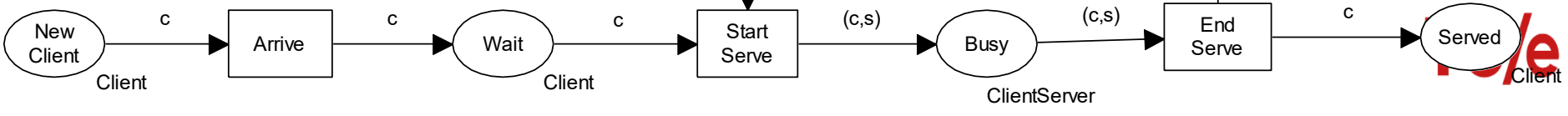


2. Information model

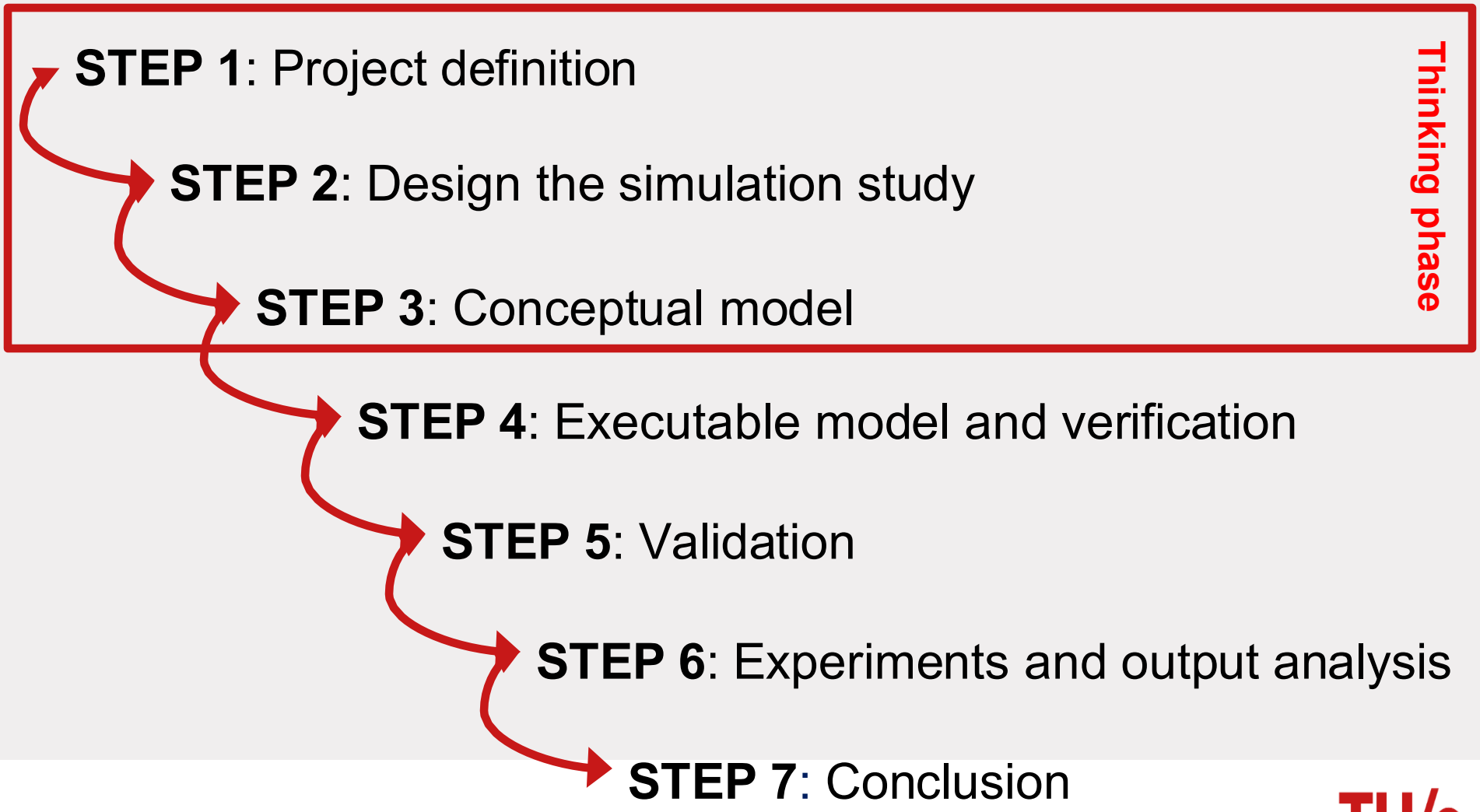
```
Arrive
Pre:    c.ArrivalTime = now.Time
Post:   Wait:= Wait.add(c);

StartServe
Pre     c in Wait.First
Post    Busy:= Busy.add(c,s) ^
        c.WaitingTime := now.Time - c.ArrivalTime ^
        s.EndTime := now.Time + c.ServiceTime ^
        s.BusyTime := s.BusyTime + c.ServiceTime;

EndServe
Pre     now.Time = s.EndTime
Post    Served := Served.add(c)
        Free := Free.add(s);
```

3. Specifications

1. Process model

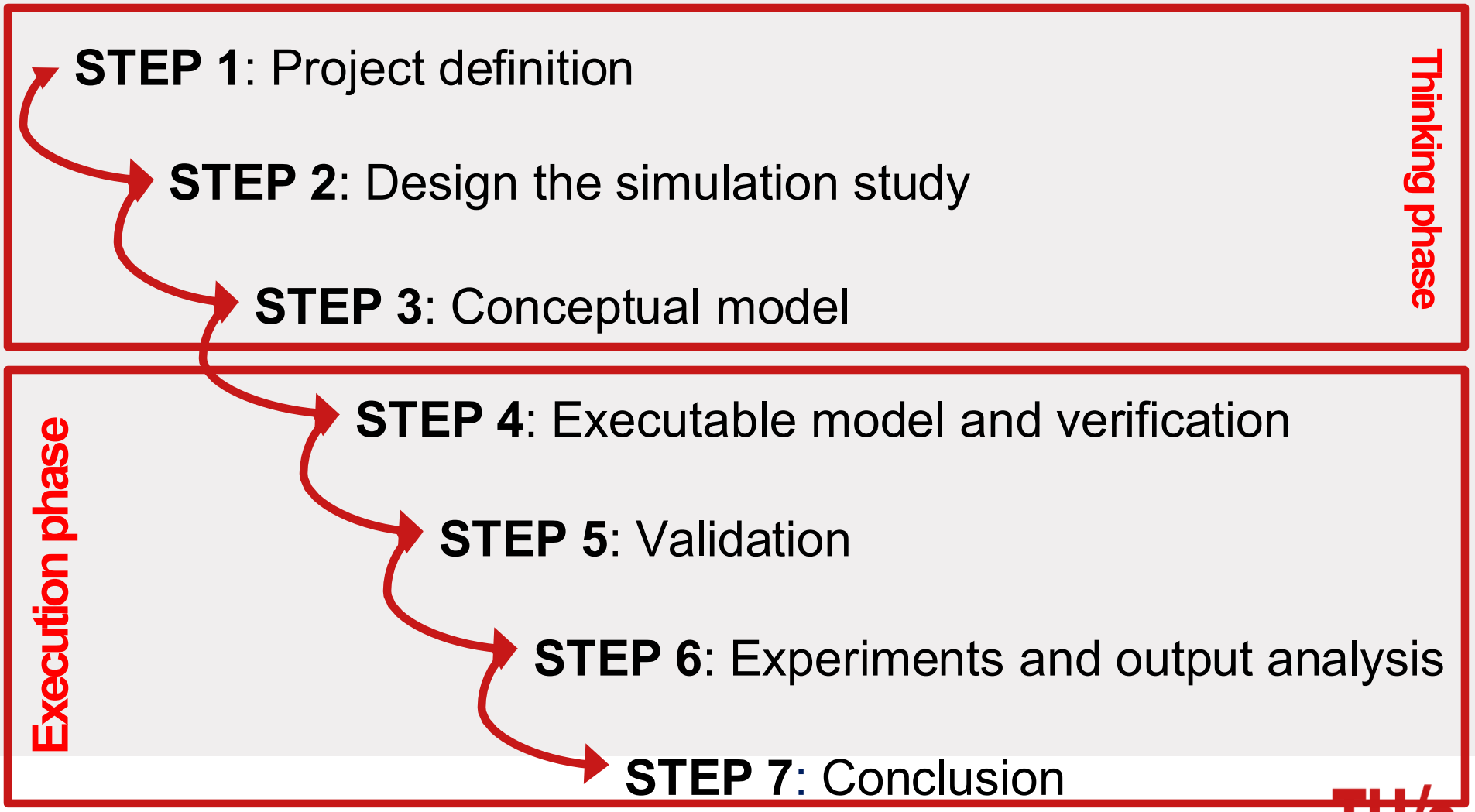


Simulation Methodology (7 steps)



TU/e

Simulation Methodology (7 steps)



Simulation Methodology (7 steps)

STEP 1: Project definition

STEP 2: Design the simulation study

STEP 3: Conceptual model

STEP 4: Executable model and verification

- **Dependent on simulation tool**
- **Does the system behave as intended/expected?**

Business Process Simulation

Lecture 4a – introduction to SimPN

Laura Genga

Department of Industrial Engineering and Innovation Sciences, Information Systems Group

THE CONCEPT

The concept

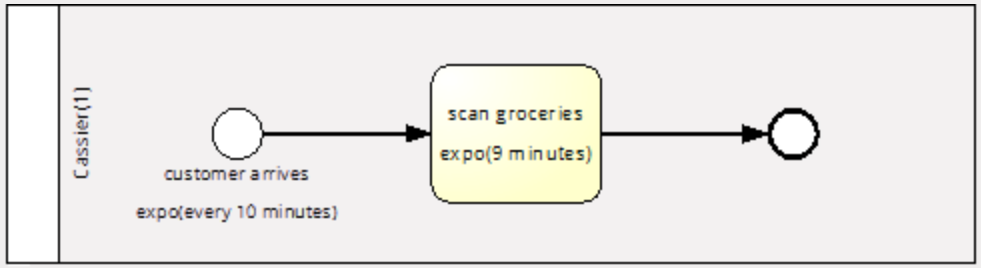


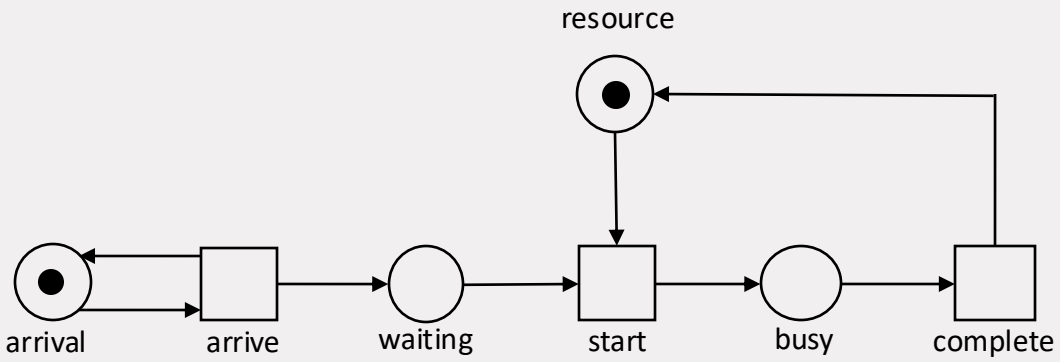
Photo by Eduardo Soares on Unsplash

Installation instructions:
https://github.com/bpogroup/simpn/tree/master

Model components:

- Variables
- Initial state
- events

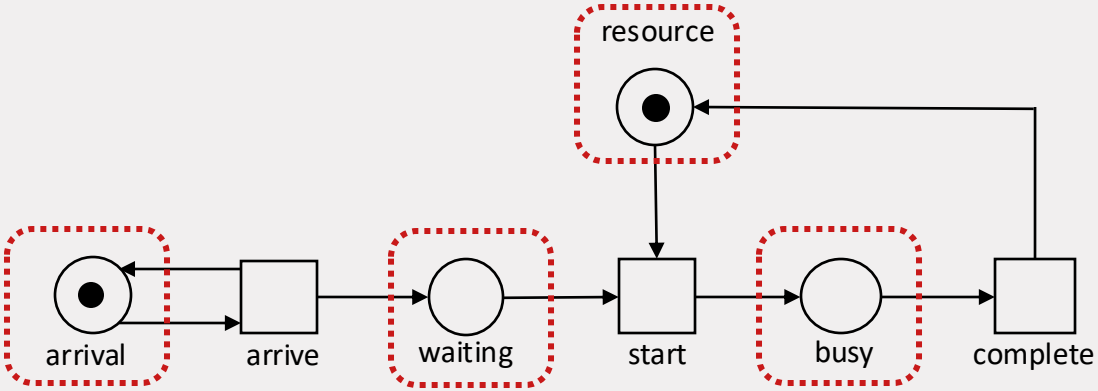
The Concept



An event can happen if it has a value on each of its incoming variables.

TU/e

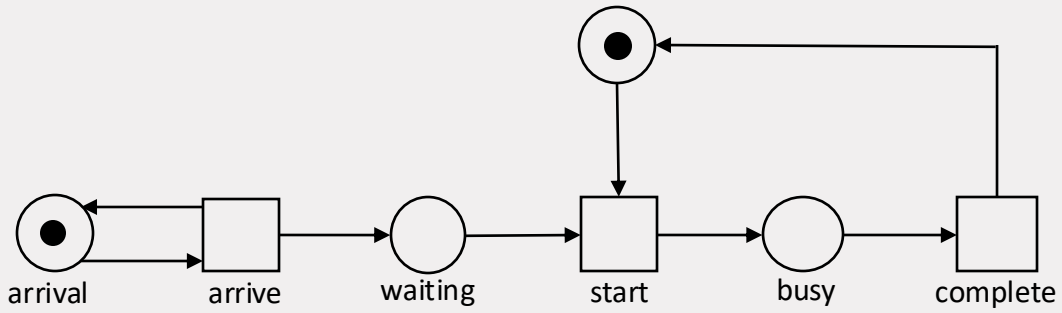
The Concept



An event can happen if it has a value on each of its incoming variables.

TU/e

The Concept

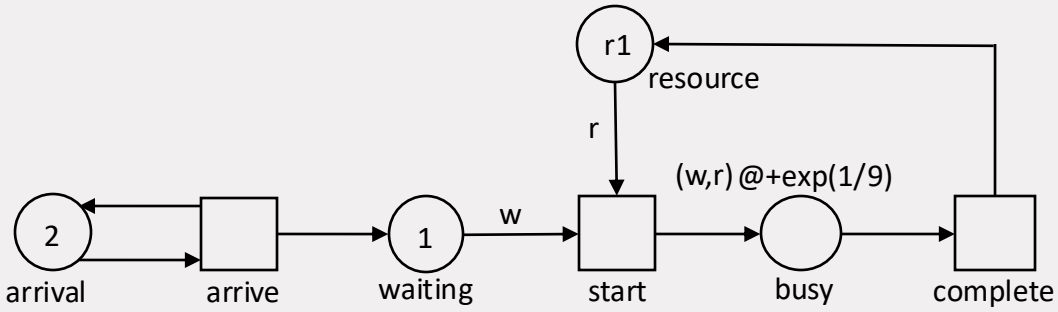


An event can happen if it has a value on each of its incoming variables.
When it happens, it:

- takes a value from each of its incoming variables
- produces a value in its outcoming variables

TU/e

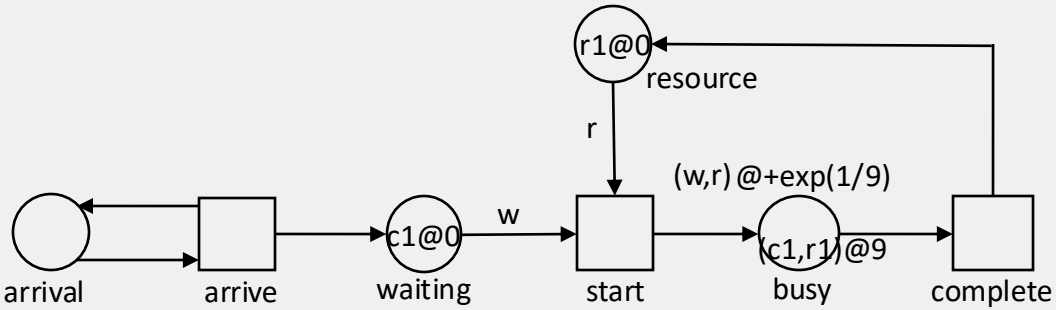
The Code



Values have time and must be produced in a list with one element for each outgoing variable.

```
def start(w, r):
  return [SimToken( (w,r), exp(1/9) )]
```

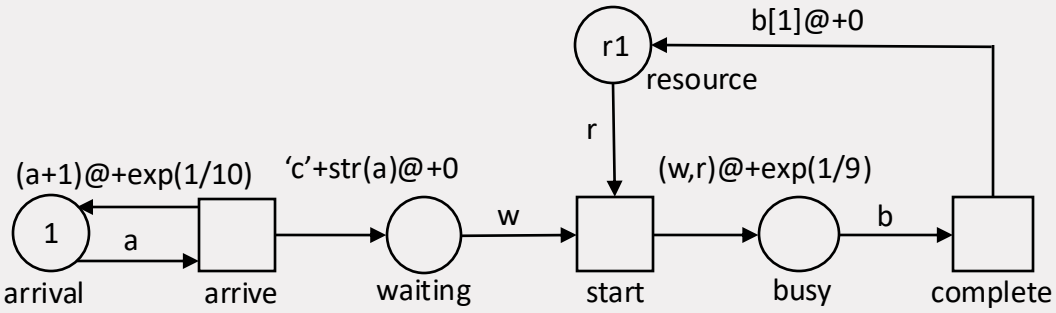
The Code



Values have time and must be produced in a list with one element for each outgoing variable.

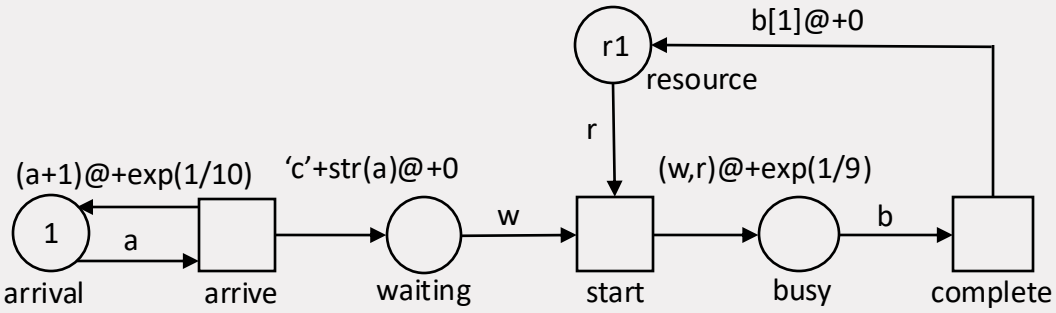
```
def start(w, r):
  return [SimToken( (w,r), delay=exp(1/9) )]
```

The Code




```
def complete(b):
  return [SimToken( b[1] )]
```

The Code



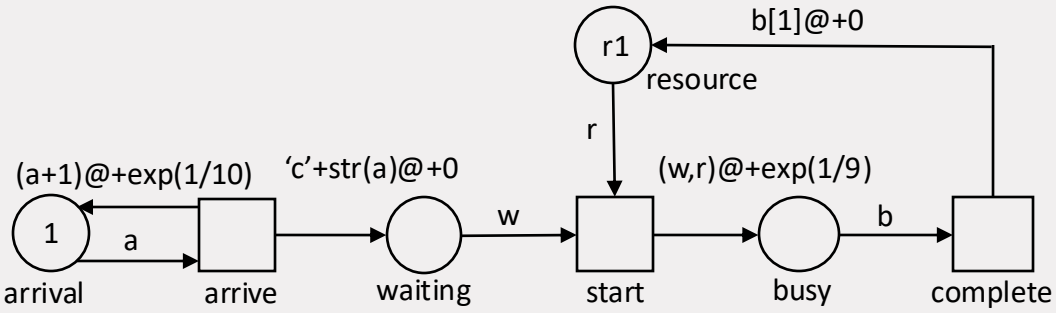

```
def arrive(a):
  return [SimToken( 'c' + str(a) ),
          SimToken( a+1, delay=exp(1/10) )]
```

How to change the ID value of the new tokens?

www.menti.com
Code: **85146877**

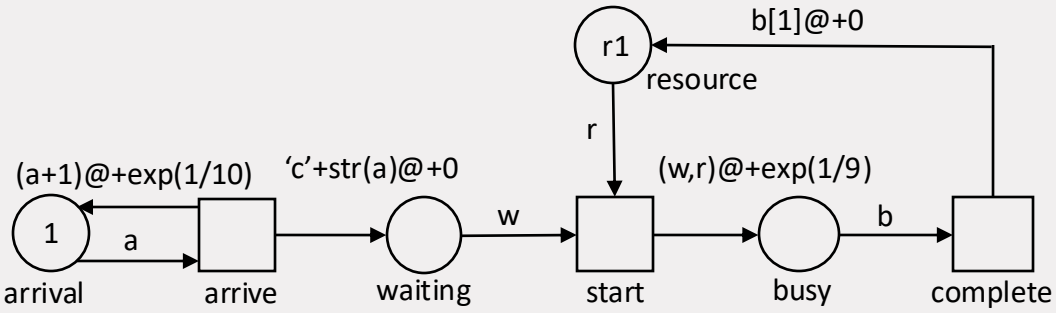
TU/e

The Code




```
shop = SimProblem()

resources = shop.add_var("resources")

resources.put("r1")
```

The Code




```
def start(w, r):
  return [SimToken( (w,r), delay=exp(1/9) )]

shop.add_event([waiting, resource], [busy], start)
```

Warning on how to instantiate SimTokens

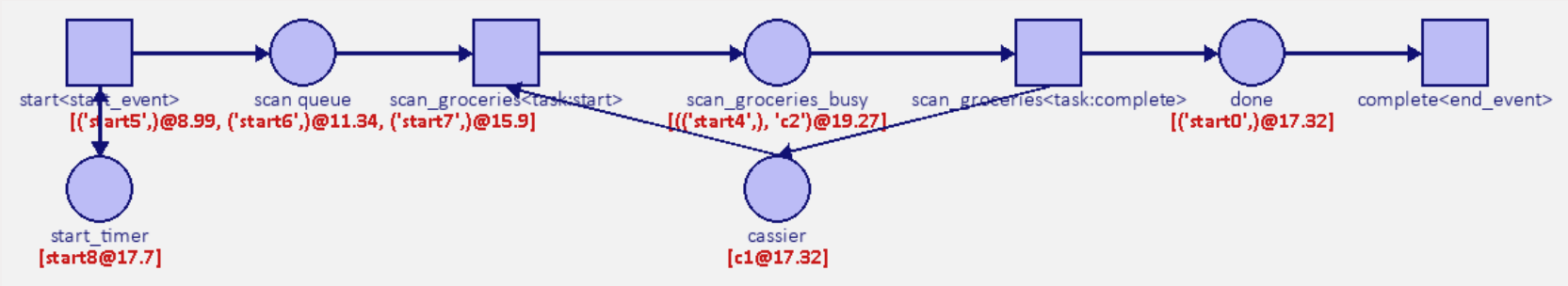
SimTokens can also be instantiated using *dictionaries*

```
# Define events.
def new_call():
    return [SimToken({'is_new': 1})] #tokens with value 1 are new customers, tokens with value 0 are
```

Easy management of multiple values carried by a token

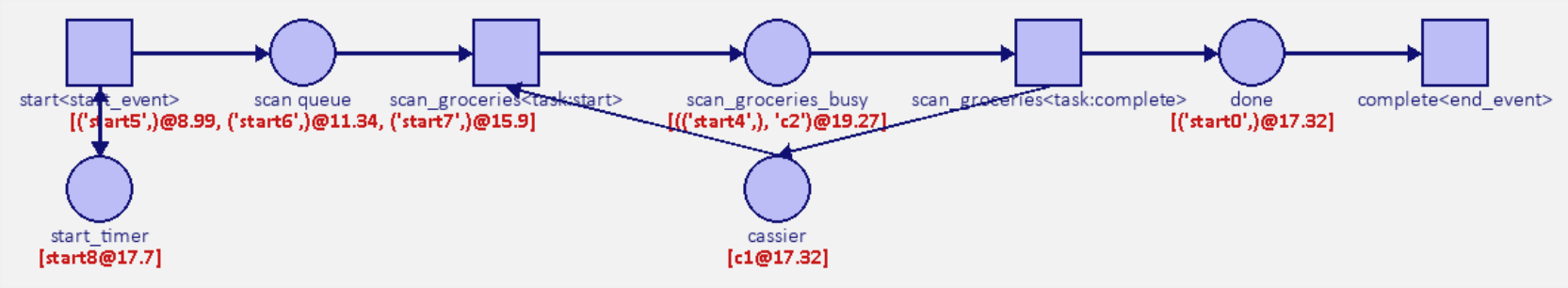
More on this at the tutorial session

Visualization



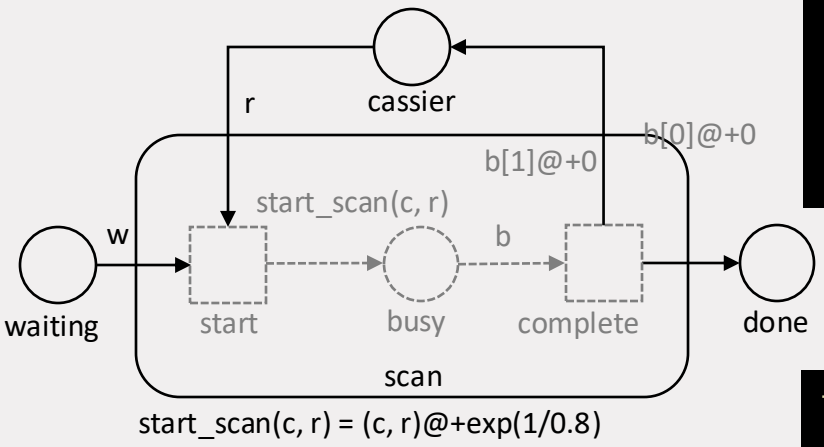

```
m = Visualisation(shop)
m.show()
```

Visualization




```
m = Visualisation(shop, "layout.txt")
m.show()
m.save_layout("layout.txt")
```

Shorthands



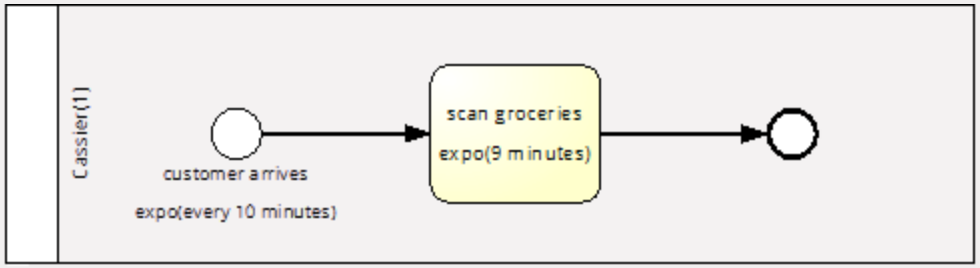

```
busy = shop.add_var("busy")

def start(c, r):
  return [SimToken((c, r), delay=exp(1/0.8))]

shop.add_event([waiting, resource], [busy], start)

def complete(b):
  return [SimToken(b[1], delay=0)]

shop.add_event([busy], [resource], complete)
```

```
task(shop,
     [waiting, cassier],
     [done, cassier],
     "scan_groceries",
     lambda c, r: [SimToken((c, r), delay=exp(1/9))])
```


TU/e

Shorthands

- BPMN Task
- BPMN Start Event
- BPMN Intermediate Event
- BPMN End Event



TU/e

Business Process Simulation

Lecture 4b – Advanced constructs of SimPN and Mapping from conceptual model

Laura Genga

Department of Industrial Engineering and Innovation Sciences, Information Systems Group

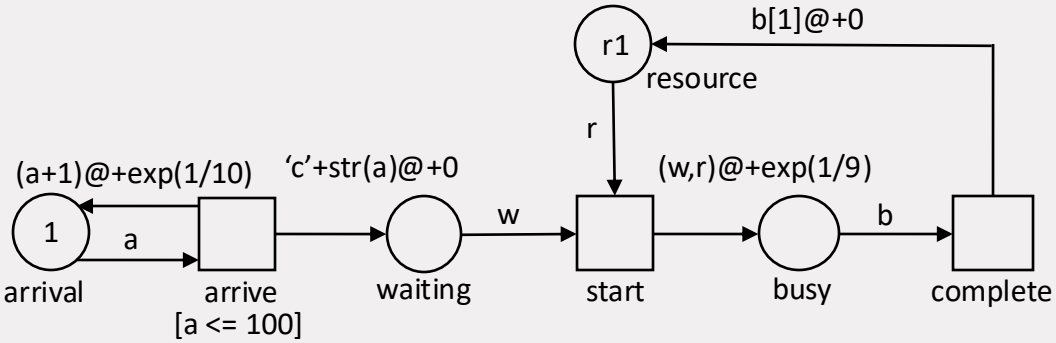
Overview on lecture modules

a) Introduction to SimPN
b) Advanced constructs of SimPN and Mapping from conceptual model
c) Solutions patterns in SimPN
d) Simulation output and verification

The Extensions

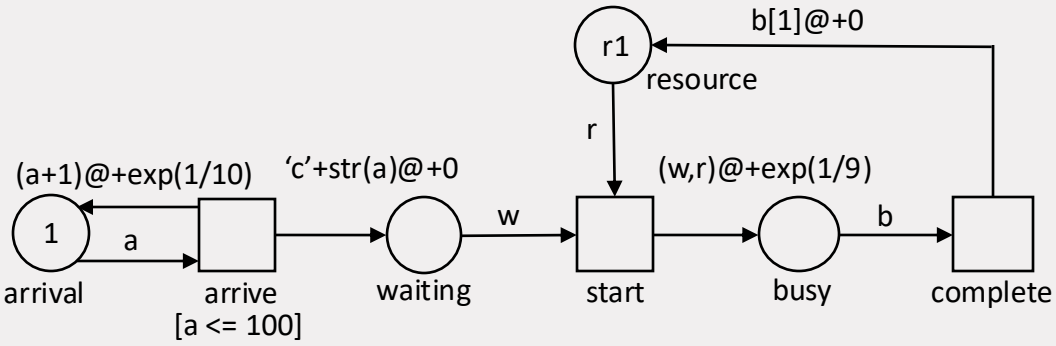
- Guards
- Choice
- Parallel Processing
- Manipulating queues

Guards



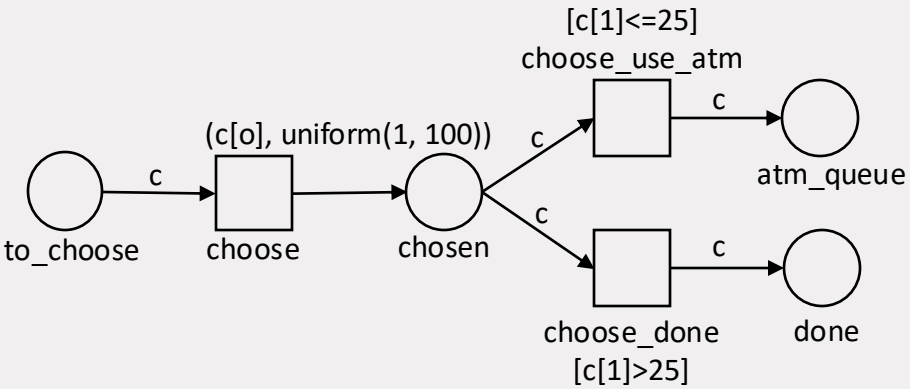

```
shop.add_event([arrival], [arrival, waiting], arrive
                guard = max_100
)
def max_100(a):
    return a <= 100
```

Guards



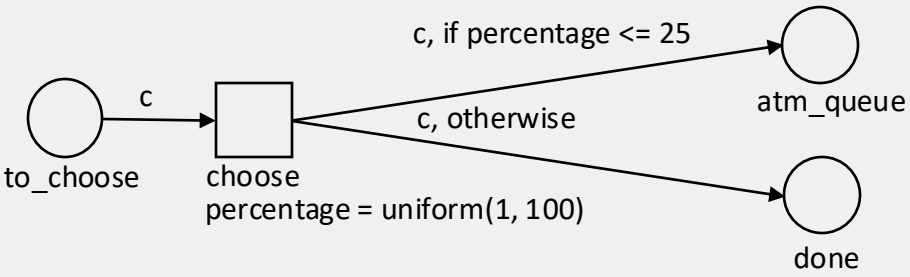

```
shop.add_event([arrival], [arrival, waiting], arrive
                guard = lambda a: a <= 100
)
```

Choice


[c[1]<=25]
choose_use_atm
c
(c[o], uniform(1, 100))
c
atm_queue
c
c
to_choose
choose
chosen
c
choose_done
[c[1]>25]
done

Choice

```
def choose(c):
    percentage = uniform(1,100)
    if percentage <= 25:
        return [SimToken(c), None]
    else:
        return [None, SimToken(c)]

shop.add_event([to_choose], [atm_queue, done], choose)
```



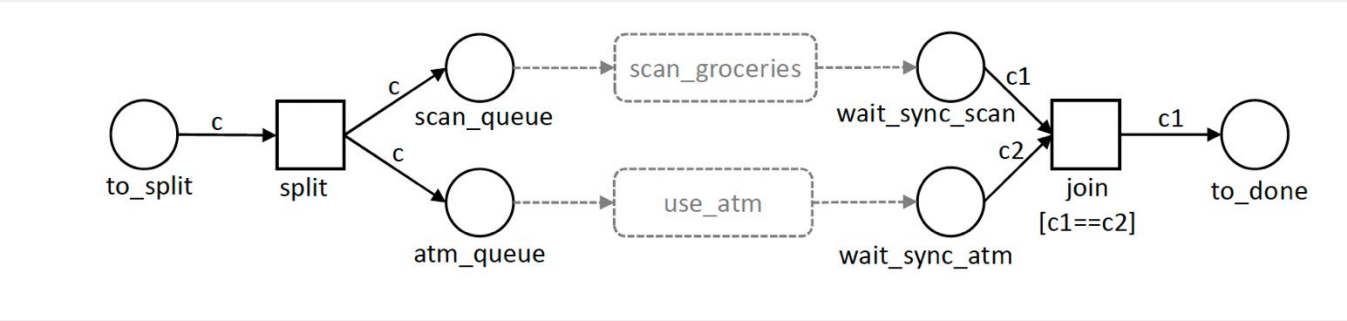
www.menti.com
Code: **85146877**

What is the output size of the transition?

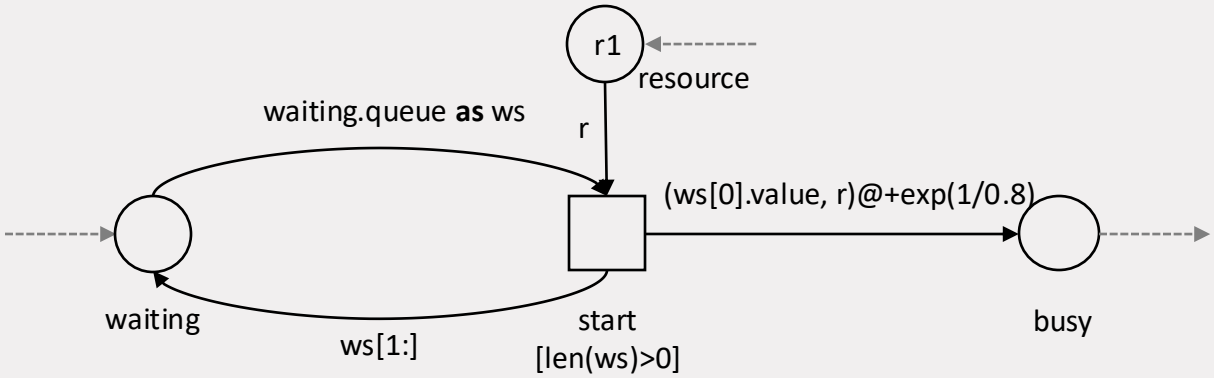
TU/e

Parallel processing

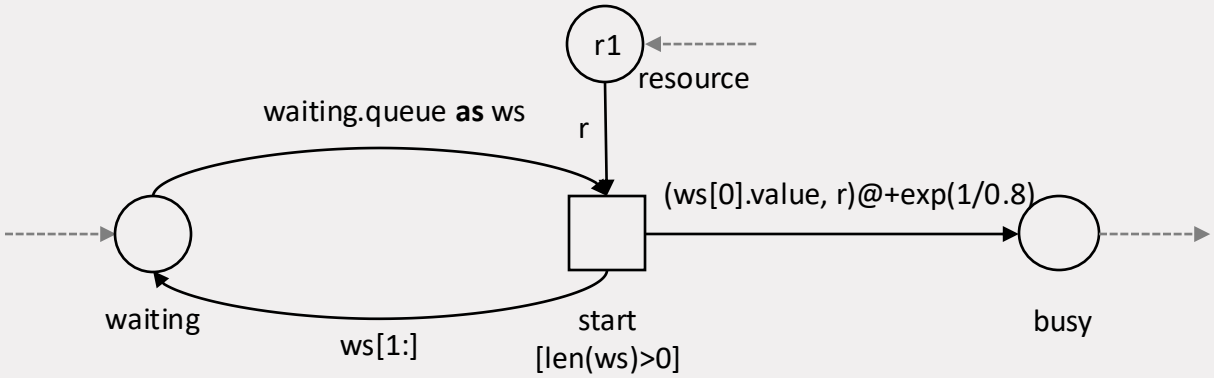
```
shop.add_event([to_split], [scan_queue, atm_queue],
               lambda c: [SimToken(c), SimToken(c)], name="split")

...

shop.add_event([wait_sync_atm, wait_sync_scan], [to_done],
               lambda c1, c2: [SimToken(c1)], name="join",
               guard=lambda c1, c2: c1 == c2)
```



Queues as Places



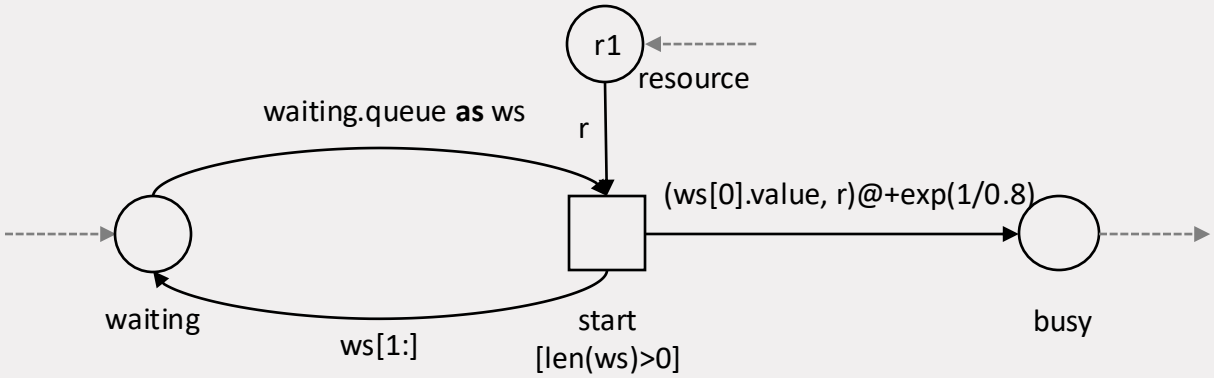
Queues as Places



The queue is obtained from the variable as a *token*

- The queue has a time, which is is the time of the latest token in the queue

Queues as Places

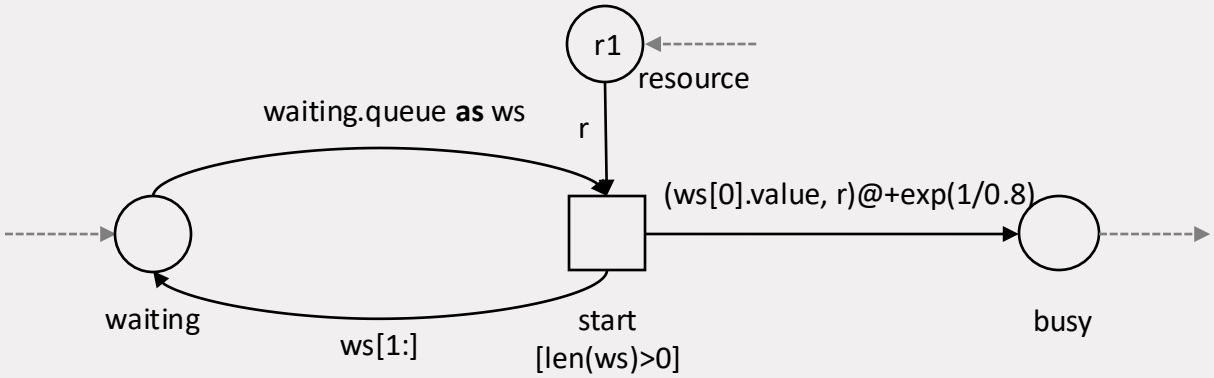


The queue is obtained from the variable as a *token*

- The queue has a time, which is is the time of the latest token in the queue
- The queue will be removed from the variable and it has to be put back

Queues as Places

```
def start(ws, r):
  c = ws[0].value
  return [ws[1:], SimToken((c, r), delay=exp(1/0.8))]

def start_condition(ws, r):
  return len(ws) > 0

shop.add_event([waiting.queue, resource], [waiting.queue, busy],
               start, guard=start_condition)
```



The queue is obtained from the variable as a *token*

- The queue has a time, which is is the time of the latest token in the queue
- The queue will be removed from the variable and it has to be put back

Time

```
time = shop.var("time")

def check_break(c, time):
    if time > c[1] + 2
        return [SimToken(c, delay=2)]
    else:
        return [SimToken(c)]

shop.add_event([cassier_break_check, time], [cassier], check_break)
```

SIMPN tutorial

Build simple SimPN models yourselves

- From M/M/1 to
- M/M/c to
- Different types of customers
- Customers leaving the queue
- etc.

Use conceptual model to derive SimPN Model

Process model

For each place:

- Identify corresponding variables in SimPN

for each transition:

- identify corresponding events in SimPN
- check post conditions

for each multi input transition

- check proper resources

for each pre condition

- check corresponding priority rules or guards

Information model

for all object classes or instances:

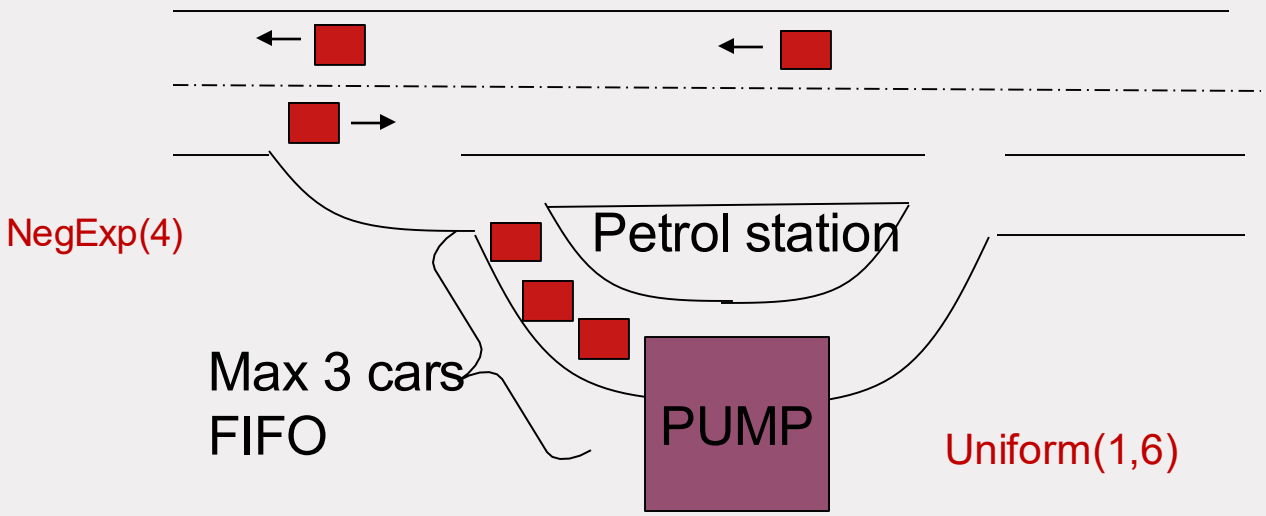
- identify representation in SimPN
- check representation of attributes

Assumptions

Describe how the assumptions are taken into account in the model

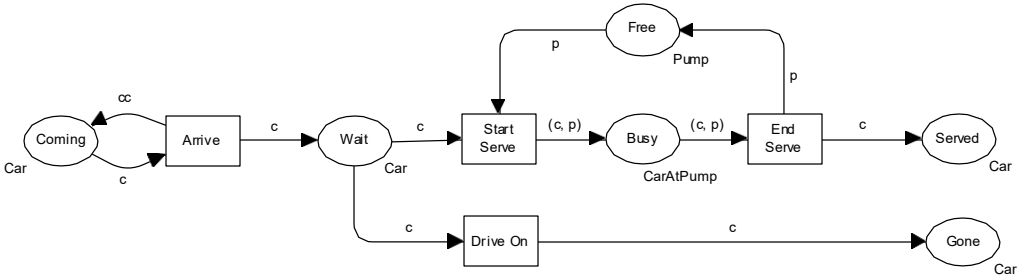
TU/e

EXAMPLE: The Petrol Station



The owner of the petrol station has the feeling that some potential clients are leaving the station because there is no place to wait for service

Petrol Station



Arrive

Pre: c.ArrivalTime = now.Time

Post: Wait :=Wait.add(c) ∧
Coming:= Coming.add(n) ∧
n.ArrivalTime := c.ArrivalTime
+r.NegExpo(4) ∧
n.ServiceTime := r.Uniform(1,6) ∧
n.WaitingTime := 0

StartServe

Pre: c in Wait.First ∧ Wait.Length ≤ Wait.Size;

Post: Busy:= Busy.add(c,p) ∧ p.endTime=now.Time+c.serviceTime ∧
c.WaitTime=now.Time-c.ArrivalTime

EndServe

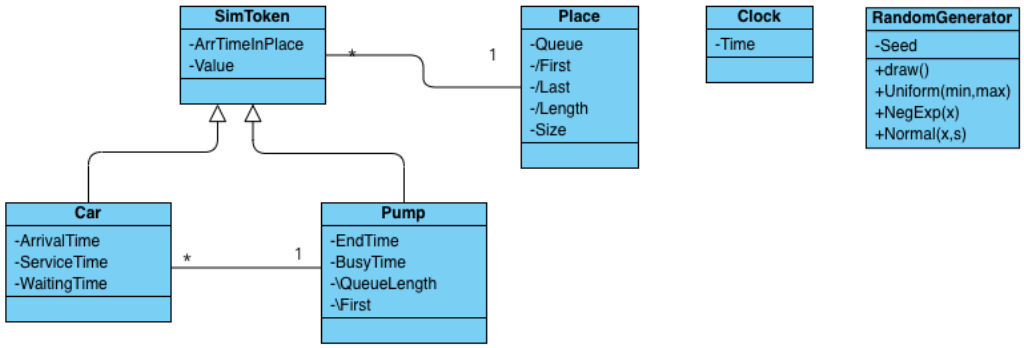
Pre now.Time = p.EndTime

Post Free.add(p) ∧
Served.add(c);

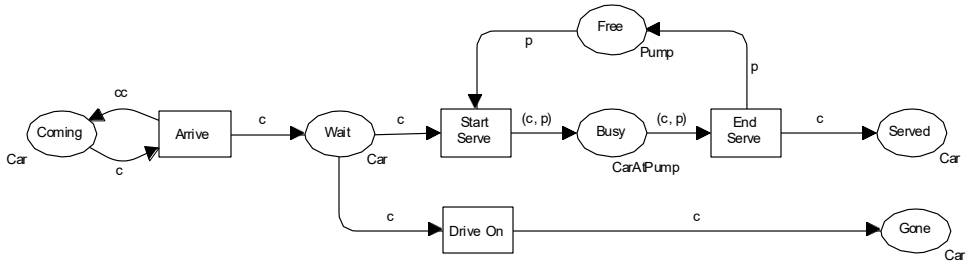
DriveOn

Pre Wait.Length > Wait.Size

Post Gone.add(c);

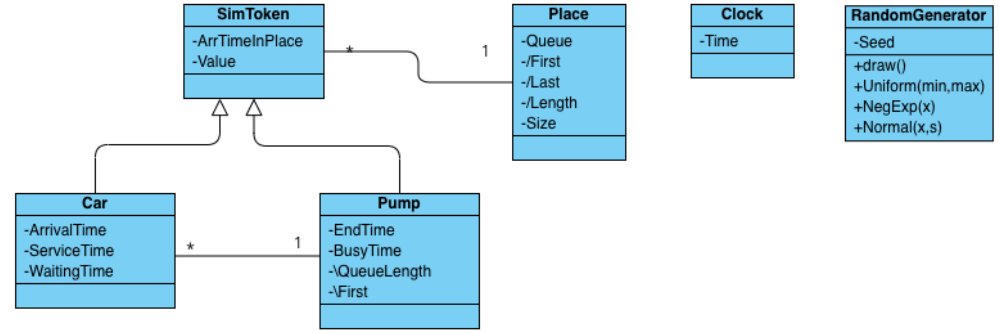


Mapping process model to SimPN



Process model	SimPn	Function
(Coming) + Arrive	Start_event start	Create car instances
Wait	to_check_queue, waiting	Variables storing the tokens generated by the arrival event and the tokens admitted to the pupm queue
Wait+(Start Serve OR Drive On)	Check_queue_length	Checks whether the car is admitted to the station
Served, Gone	Done, to_leave, End_event leave, End_event complete	Variables storing served and not served tokens, together with the corresponding end events
Free	pump	Variable storing the resources
Start serve + Busy + End Serve +Free	Task refueling	Task serving the cars at the station

Mapping information model



Information model	SimPn
p1, Class Pump	Resource “p1”
c1...c10, Class Car	Entity Type Car (in start_event start)
now	Var time
r	Random package

Mapping of assumptions, conditions, and functions

Show, if possible, how these are translated to your model, e.g. :

A1. The business operates 24 / 7

=> Run setup

PercUnServed = (Gone.Length / (Gone.Length + Served.Length)) * 100;
AvgWaitTime = Sum(c.WaitTime: c inServed) /Served.Length;

=> Simulation report

Explain how priority/guards in transitions preconditions are accounted for in the model

TU/e

Business Process Simulation

Lecture 4c – Solutions patterns in SimPN

Laura Genga

Department of Industrial Engineering and Innovation Sciences, Information Systems Group

Overview on lecture modules

a) Introduction to SimPN
b) Advanced constructs of SimPN and Mapping from conceptual model
c) Solutions patterns in SimPN
d) Simulation output and verification

Solution patterns in SimPN:

Pattern 1a - Balking: deciding to not join the queue
Petrol Station example

Pattern 1b - Jockeying: switching queues

Pattern 1c – Reneging: leaving queue after a certain time

Pattern 2 - Sharing resources with other activities

Pattern 3 - Shifting task priorities

Petrol Station: Pattern 1a - Balking

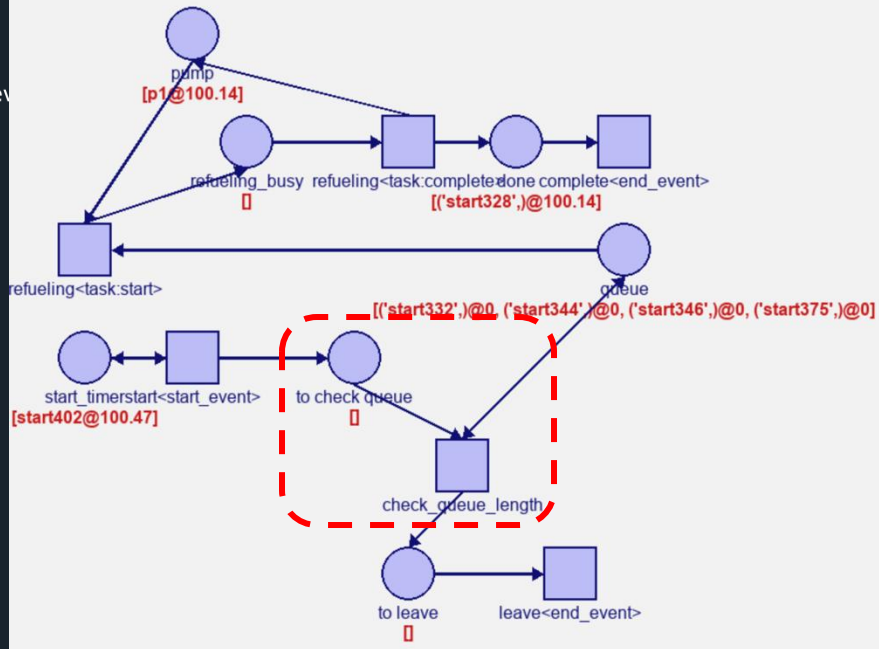
Balking is a situation in which an arriving customer does not join the queue but goes away or goes someplace else.

How to model this?

- Assign tolerance to an entity attribute or a variable
- When the tolerance is less than or equal to the current number in the queue we “balk” the arrival from the system

Petrol Station: Pattern 1a - Balking

```
from simpn.simulator import SimProblem
from simpn.simulator import SimToken
from random import expovariate as exp, uniform
from simpn.reporters import SimpleReporter
from simpn.prototypes import start_event, task, end_event, intermediate_ev
from simpn.visualisation import Visualisation


# Instantiate a simulation problem.
station=SimProblem()

# Define queues and other 'places' in the process.
to_check_queue = station.add_var("to check queue")
waiting = station.add_var("queue")
done = station.add_var("done")
to_leave = station.add_var("to leave")

# Define resources.
pump = station.add_var("pump")
pump.put("p1")

# Define events.
def interarrival_time():
  return exp(1/4)

start_event(station, [], [to_check_queue], "start", interarrival_time)

# Check the queue length
def check_queue_length(customer, complete_queue):
    if len(complete_queue) < 4: # If it is less than 4 customers in the queue, the customer goes into the queue
        # we add the customer to the queue by actually manipulating the queue variable
        complete_queue.append(SimToken(customer))
        return [complete_queue, None]
    else: # If it is more than 4 customers in the queue, the customer leaves
        # note that this means that the queue is not changed, but we need to put it back.
        return [complete_queue, SimToken(customer)]

station.add_event([to_check_queue, waiting.queue], [waiting.queue, to_leave], check_queue_length)
task(station, [waiting, pump], [done, pump], "refueling", lambda c, r: [SimToken((c, r), delay=uniform(1,6))])

end_event(station, [done], [], "complete")
end_event(station, [to_leave], [], "leave")
```



www.menti.com
Code: **85146877**

Which components are used to model the pattern?

TU/e

Pattern 1b: Jockeying

Jockeying is a situation in which customers switch queues when a certain condition is satisfied

How to model this?

- Use guards to implement switch conditions
- Move entity to other queues

Example Pattern 1b: Jockeying

Customers arrive at a checkout according to an exponential distribution of 5 min.

There are two checkouts,

- checkout 1: a skilled employee serves at EXPO(5) minutes,
- checkout 2: an unskilled employee serves at EXPO(12) minutes.

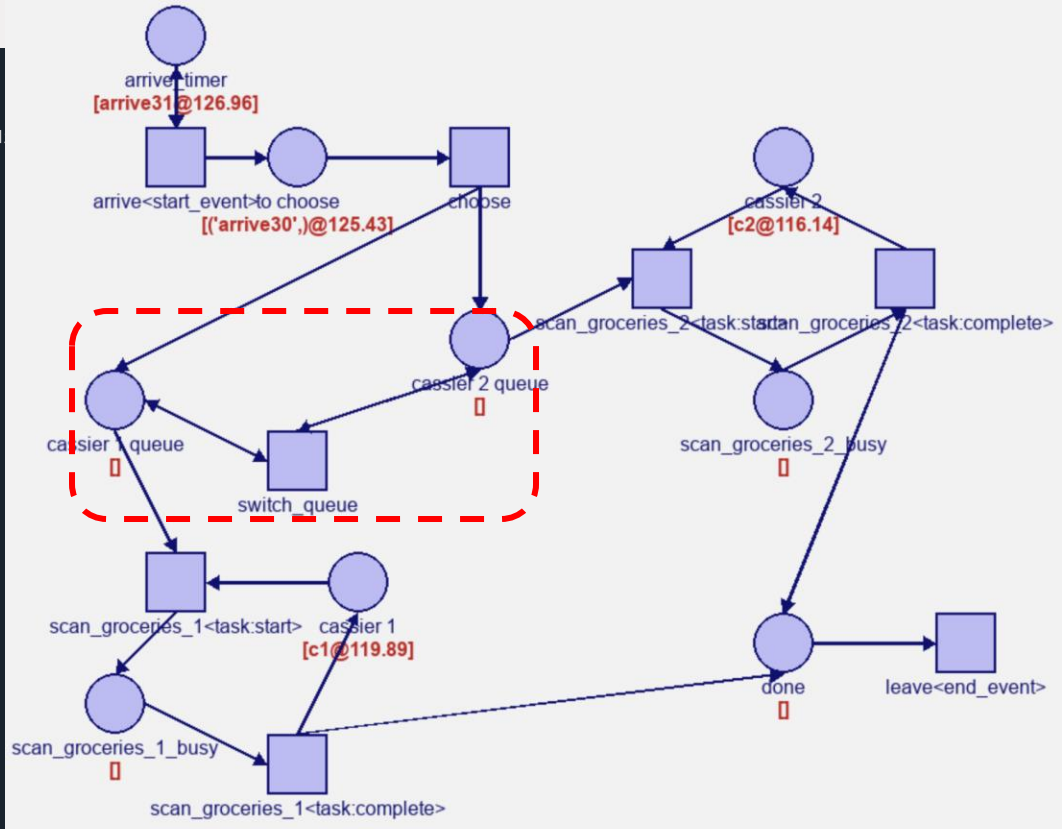
Customers are split equally over the two checkouts

When customers are in the queue of checkout 2 and this queue is more than 2 persons longer than the queue at checkout 1, customers will switch to queue 1.

Customers at checkout 1 cannot switch queues

Example Pattern 1b: Jockeying

```
from simpn.simulator import SimProblem
from simpn.simulator import SimToken
from random import expovariate as exp, uniform
from simpn.reporters import SimpleReporter
from simpn.prototypes import start_event, task, end_event, intermed
from simpn.visualisation import Visualisation


shop = SimProblem()

to_choose = shop.add_var("to choose")
waiting_1 = shop.add_var("cassier 1 queue")
waiting_2 = shop.add_var("cassier 2 queue")
cassier_1 = shop.add_var("cassier 1")
cassier_2 = shop.add_var("cassier 2")

cassier_1.put("c1")
cassier_2.put("c2")

done = shop.add_var("done")

start_event(shop, [], [to_choose], "arrive", lambda: exp(1/5))

def choose(c):
    percentage = uniform(1,100)
    if percentage <= 51:
        print("first queue")
        return [SimToken(c), None]
    else:
        print("second queue")
        return [None, SimToken(c)]
    return [None, SimToken(c)]

def switch_condition(c,c1_queue, c2_queue):
    return len(c1_queue)>len(c2_queue)+1

def switch_queue(c,c1_queue, c2_queue):
    last_token=c1_queue.pop()
    c2_queue.append(last_token)
    return [c1_queue, c2_queue]

shop.add_event([to_choose],[waiting_1, waiting_2], choose)
shop.add_event([waiting_1, waiting_1.queue, waiting_2.queue],[waiting_1.queue, waiting_2.queue], switch_queue,guard=switch_co

task(shop, [waiting_1, cassier_1], [done, cassier_1],"scan_groceries_1", lambda c, r: [SimToken((c, r), delay=exp(1/5))])
task(shop, [waiting_2, cassier_2], [done, cassier_2], "scan_groceries_2", lambda c, r: [SimToken((c, r), delay=exp(1/12))])
end_event(shop, [done], [], "leave")
```



www.menti.com
Code: **85146877**

Which components are used to model the pattern?

TU/e

Pattern 1c: Reneging

Reneging is a situation in which arriving customers are willing to wait for only a limited period of time before they renege from the queue.

How to model this?

- Allow the arrival to enter the queue
- When the renege time is reached we need to find the entity and remove it from the queue.

Example Pattern 1c: Reneging

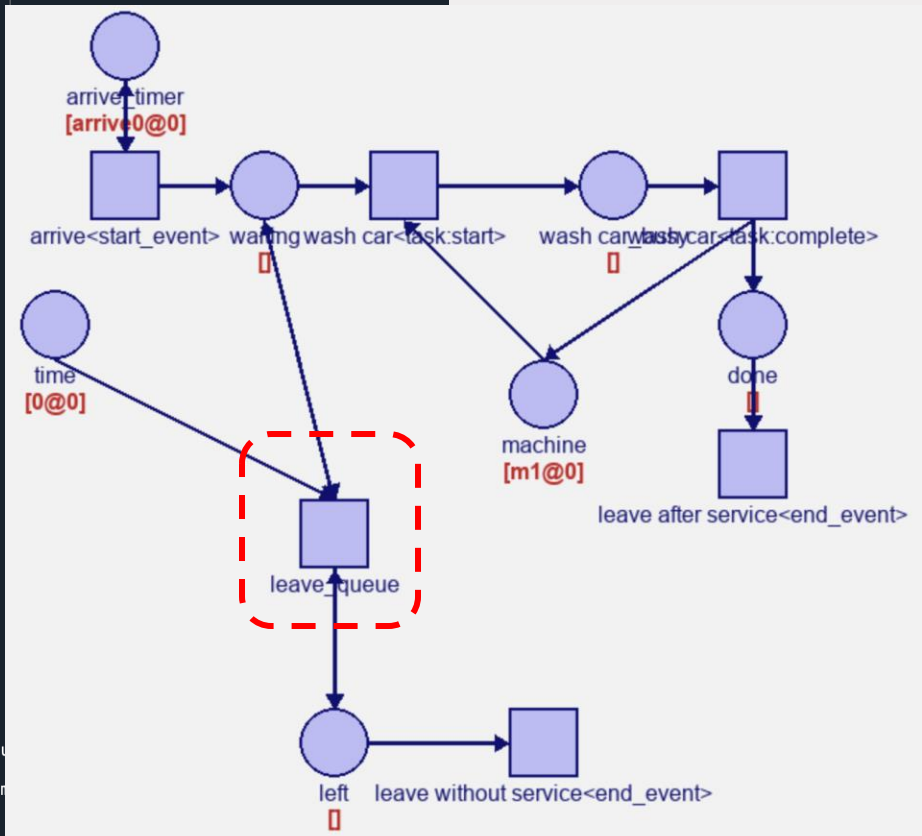
Cars arrive at a carwash according to an exponential distribution of 10 min.

Washing a car takes EXPO(7) minutes (1 server)

Cars that are waiting for more than 10 minutes leave the system with a probability of 80%

Example Pattern 1c: Reneging

```
from random import expovariate as exp, uniform
from simpn.reporters import SimpleReporter
from simpn.prototypes import start_event, task, end_event, intermediate_event
from simpn.visualisation import Visualisation

carwash = SimProblem()

left=carwash.add_var("left")
waiting = carwash.add_var("waiting")
machine = carwash.add_var("machine")

machine.put("m1")

done = carwash.add_var("done")

time_var = carwash.var("time")


start_event(carwash, [], [waiting], "arrive", lambda: exp(1/10))

def leave_condition(t, c1_queue, left_queue):
    for c in c1_queue:
        if t - c.time > 10:
            return True
    return False



def leave_queue(t, c1_queue, left_queue):
    for c in c1_queue:
        if t - c.time > 10:
            percentage = uniform(1,100)
            if percentage >= 80:
                c1_queue.remove(c)
                left_queue.append(c)
    return [c1_queue, left_queue]

carwash.add_event([time_var, waiting.queue, left.queue],[waiting.queue, left.qu

task(carwash, [waiting, machine], [done, machine],"wash car", lambda c, r: [Sim
end_event(carwash, [done], [], "leave after service")
end_event(carwash, [left], [], "leave without service")
```



www.menti.com
Code: **85146877**

Which components are used to model the pattern?

TU/e

Pattern 2: Sharing resources

Resources have different activities and may therefore be unavailable to the process for some time

How to model this?

- Seize the resource for a different activity

Pattern 2: Sharing resources

Cars arrive at a garage according to an exponential distribution of 20 min.

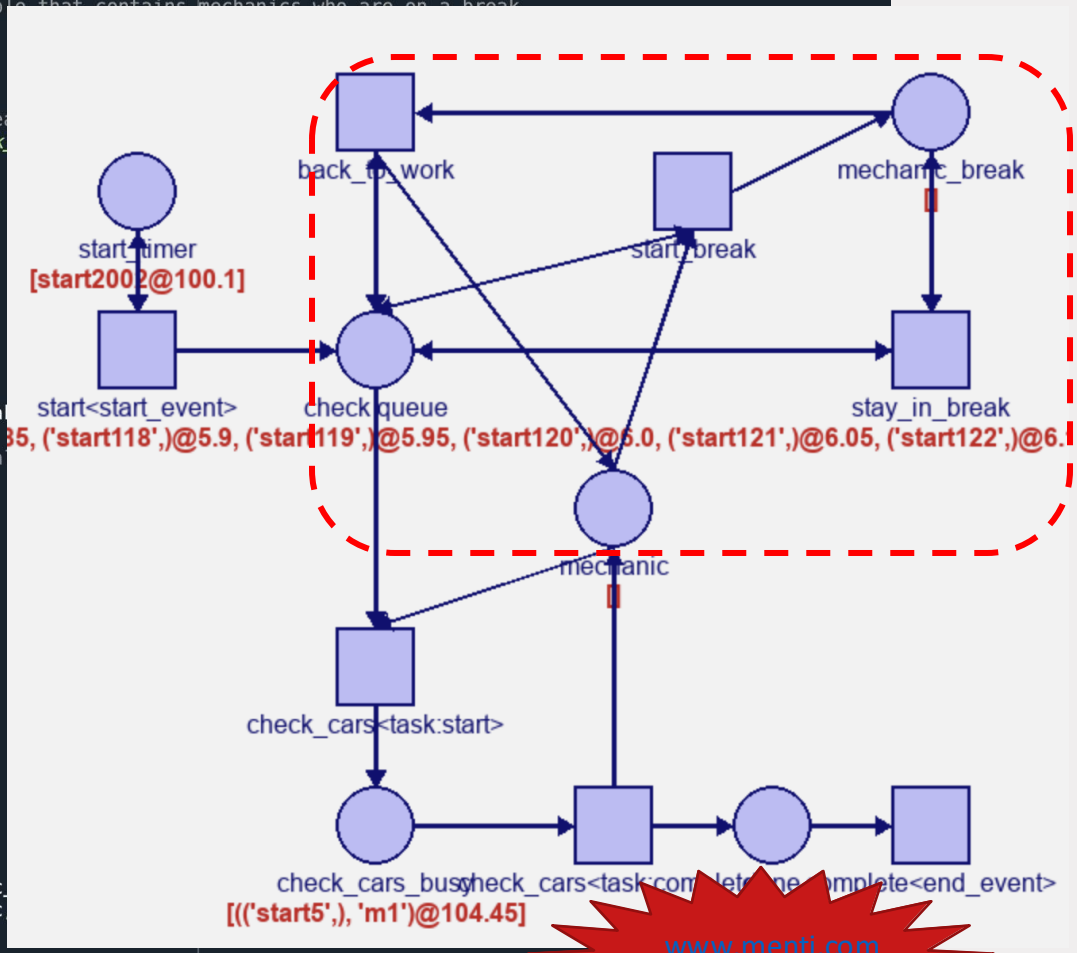
Checking a car takes Uniform(15,20) minutes (1 mechanic)

If the queue is empty the mechanic will take a coffee break of 10 minutes

He will stay in his break unless more than 2 cars are waiting in the queue

Pattern 2: Sharing resources

```
# Define queues and other 'places' in the process.
check_queue = garage.add_var("check queue")
done = garage.add_var("done")

# Define resources and resource state variables.
mechanic = garage.add_var("mechanic") # the variable that contains available mechanics
mechanic.put("m1") # mechanic m1
mechanic_break = garage.add_var("mechanic_break") # the variable that contains mechanics who are on a break

# Define events.
start_event(garage, [], [check_queue], "start", lambda: 1/20)

# Note: after the task is done, we send mechanics for the 'brea
task(garage, [check_queue, mechanic], [done, mechanic], "check

def start_break_condition(c,car_queue):
    if len(car_queue)==0:
        return True
    else:
        return False
# The break check is basically a choice
def start_break(c, car_queue):
        return [SimToken(c, delay=10), car_queue]


garage.add_event([mechanic, check_queue.queue], [mechanic_breal

#once the mechanic is on a break, it will be availble again on
def stay_in_break_condition(c, car_queue):
    if len(car_queue)<2:
        return True
    else:
        return False

def back_to_work_condition(c, car_queue):
    if len(car_queue)>=2:
        return True
    else:
        return False

def stay_in_break(c, car_queue):
    return [SimToken(c, delay=10),car_queue]

def back_to_work(c, car_queue):
    return [SimToken(c),car_queue]

garage.add_event([mechanic_break, check_queue.queue],[mechanic_
garage.add_event([mechanic_break, check_queue.queue],[mechanic_

end_event(garage, [done], [], "complete")
```



www.menti.com
Code: **85146877**

Which components are used to model the pattern?

TU/e

Pattern 3: Changing task priorities

Certain tasks have priority over others under certain conditions

How to model this?

- Determine the conditions that give priority
- Check for conditions and set priority values

TU/e

Pattern 3: Changing task priorities

Defect products arrive at a repair shop according to an exponential distribution of 10 min.

One part of the product needs cleaning EXPO(5) while the other half needs repairing EXPO(10) minutes. Both activities can be done in parallel.

After repairing & cleaning the product parts are assembled and packed (packing takes 3 minutes) for distribution.

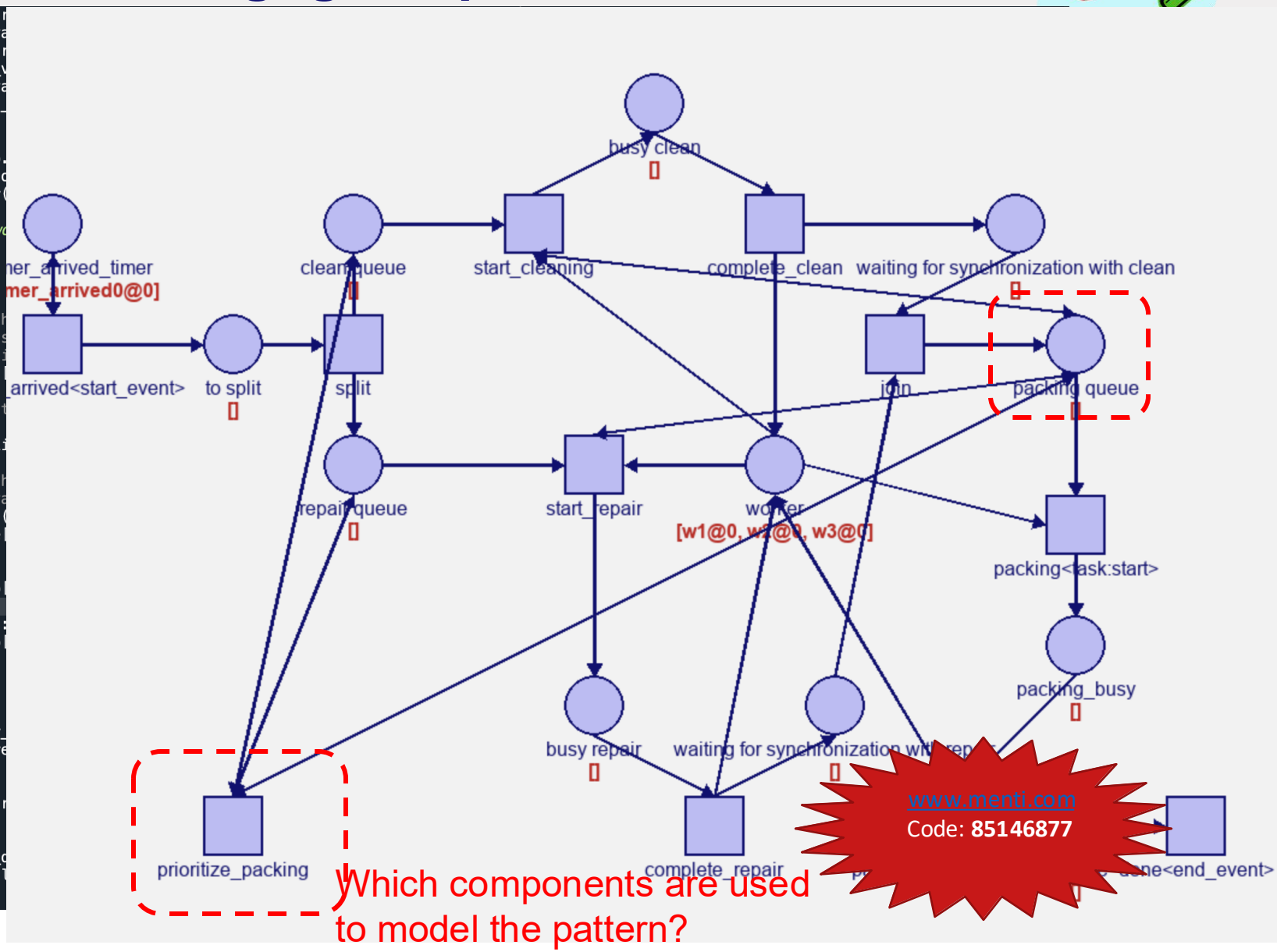
There are two workers in the repair shop which can be used to clean, repair or pack the products.

If more than 3 products are waiting for packing and less than 5 parts are waiting at the repair or clean queue, packing gets the highest priority. When packing is empty, its priority is set back to low.

TU/e

Pattern 3: Changing task priorities

```
to_split = shop.add_var
clean_queue=shop.add_va
busy_clean=shop.add_var
repair_queue=shop.add_v
busy_repair=shop.add_va
packing_queue=shop.add_


wait_sync_w_rep = shop.
wait_sync_w_clean = sho
to_done = shop.add_var(

worker=shop.add_var("wo
worker.put("w1")
worker.put("w2")
worker.put("w3")

#Customer arrives at th
#Every new token is ass
#Every new token is ini
start_event(shop, [],

# Here I am using the t
# concurrently
shop.add_event([to_spli

#this guard verifies th
#have a higher/equal va
def priority_condition(
    return all(t.value

def complete_clean(b):
    return [SimToken(b

def complete_repair(b):
    return [SimToken(b

def start_repair(c, r,
  return [SimToken((c,

shop.add_event([repair_
shop.add_event([busy_re


def start_cleaning(c, r
  return [SimToken((c,

shop.add_event([clean_c
shop.add_event([busy_cl
```



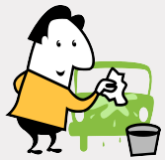
Solution patterns in SimPN:

Pattern 1a: Balking: deciding to not join the queue

Petrol Station example

Pattern 1b: Jockeying: switching queues

Checkout example, Harbour example



Pattern 1c: Reneging pattern: leaving queue after certain time

Car Wash example

Pattern 2: Sharing resources with other activities

Coffee Break example

Pattern 3: Shifting task priorities

Order Processing example

TU/e

Business Process Simulation

Lecture 4d – Simulation output and verification

Laura Genga

Department of Industrial Engineering and Innovation Sciences, Information Systems Group

Overview on lecture modules

a) Introduction to SimPN
b) Advanced constructs of SimPN and Mapping from conceptual model
c) Solutions patterns in SimPN
d) Simulation output and verification

Reporting Measurements

SimPN enables monitoring each event occurring during a simulation run

Different reporting functions are available within the library; however, you can also write your own

Simple reporting

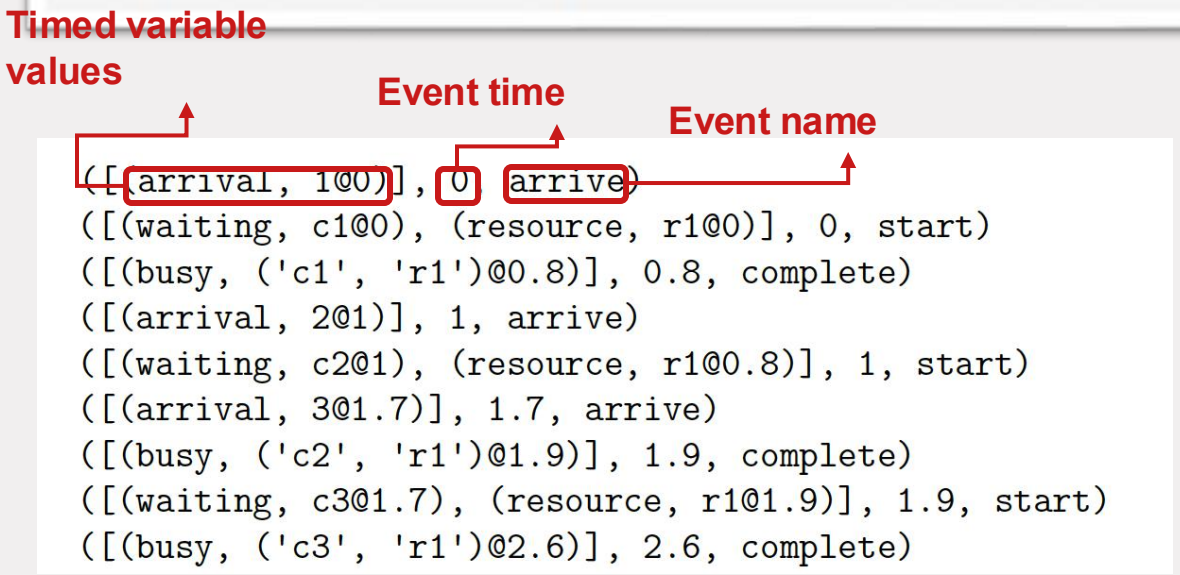
```
from simpn.reporters import Reporter

class MyReporter(Reporter):
    def callback(self, timed_binding):
        print(timed_binding)

shop.simulate(3.0, MyReporter())
```

```
([(arrival, 1@0)], 0, arrive)
([(waiting, c1@0), (resource, r1@0)], 0, start)
([(busy, ('c1', 'r1')@0.8)], 0.8, complete)
([(arrival, 2@1)], 1, arrive)
([(waiting, c2@1), (resource, r1@0.8)], 1, start)
([(arrival, 3@1.7)], 1.7, arrive)
([(busy, ('c2', 'r1')@1.9)], 1.9, complete)
([(waiting, c3@1.7), (resource, r1@1.9)], 1.9, start)
([(busy, ('c3', 'r1')@2.6)], 2.6, complete)
```


TU/e

Simple reporting

```
from simpn.reporters import Reporter

class MyReporter(Reporter):
    def callback(self, timed_binding):
        print(timed_binding)

shop.simulate(3.0, MyReporter())
```




```
([(arrival, 1@0)], 0, arrive)
([(waiting, c1@0), (resource, r1@0)], 0, start)
([(busy, ('c1', 'r1')@0.8)], 0.8, complete)
([(arrival, 2@1)], 1, arrive)
([(waiting, c2@1), (resource, r1@0.8)], 1, start)
([(arrival, 3@1.7)], 1.7, arrive)
([(busy, ('c2', 'r1')@1.9)], 1.9, complete)
([(waiting, c3@1.7), (resource, r1@1.9)], 1.9, start)
([(busy, ('c3', 'r1')@2.6)], 2.6, complete)
```


TU/e

Process reporter

```
start_event(shop, [], [waiting], "arrive", lambda: exp(1/10))

task(shop, [waiting, cassier], [done, cassier], "scan_groceries",
     lambda c, r: [SimToken((c, r), delay=exp(1/9))])

end_event(shop, [done], [], "complete")

reporter = ProcessReporter()
shop.simulate(24*60, reporter)
reporter.print_result()
```

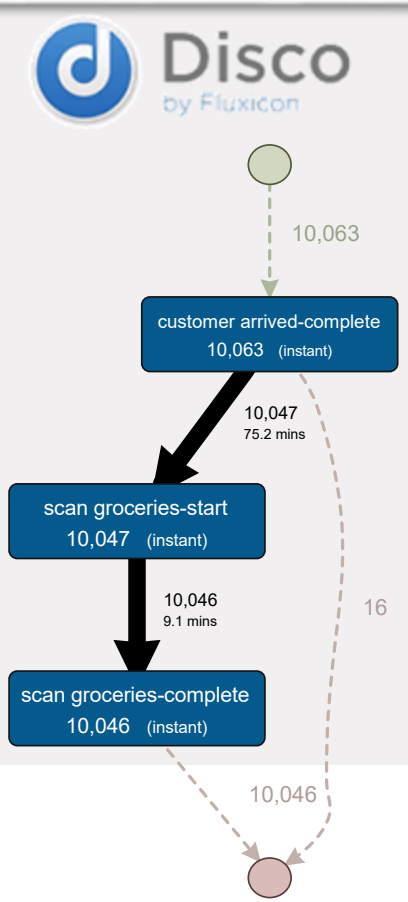
Output : average waiting time per case, average processing time per case, average cycle time per case, utilization rate for each resource

It relies on the use of the *event* and *task* prototypes

TU/e

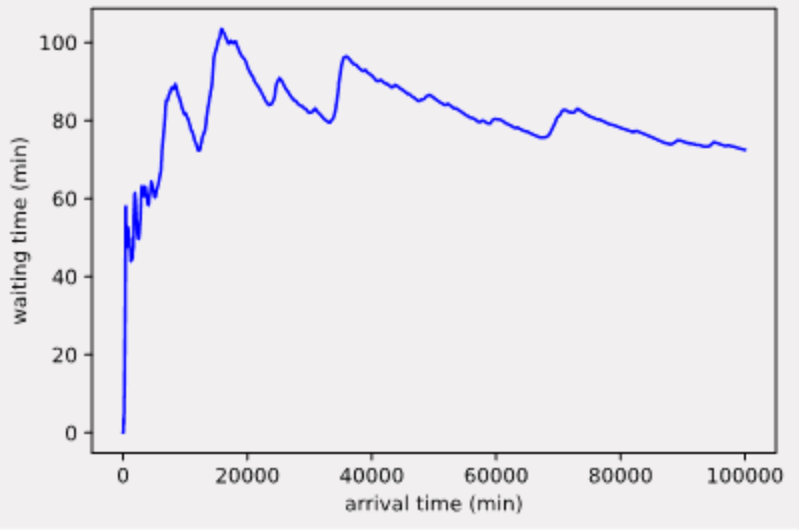
Event log reporter

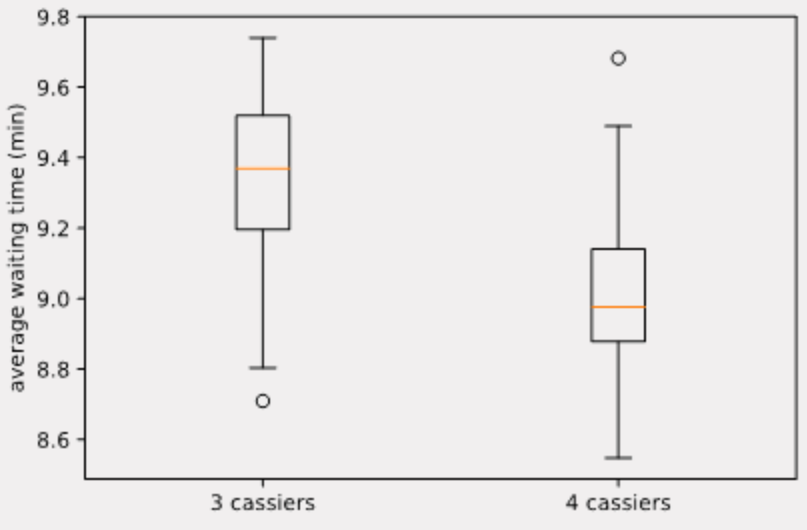
```
reporter = EventLogReporter("reporting.csv")
shop.simulate(24*60, reporter)
reporter.close()
```



Note: only works for BPMN shorthands

Warmup Time and Replications





Working with replications

A simulation study consist of several runs called replications

If the system is initialized between replications,
time starts each replication from 0, with 0 entities in the system

- Warm-up period is neglected

If “system” is not initialized between replications, time continues and entities stay in the system

- Warm-up period once, at the beginning of rep. 1

```
shop.store_checkpoint("initial state")

average_cycle_times = []

NR_REPLICATIONS = 50
SIMULATION_DURATION = 40000
WARMUP_TIME = 20000
for _ in range(NR_REPLICATIONS):
  reporter = ProcessReporter(WARMUP_TIME)
  shop.restore_checkpoint("initial state")
  shop.simulate(SIMULATION_DURATION, reporter)

  average_cycle_times.append(reporter.total_cycle_time /
                             reporter.nr_completed)
```

TU/e

Definition of Verification (Banks et al.)

Verification of a simulation model is the process of confirming that it is correctly implemented with respect to the conceptual model (it matches specifications and assumptions deemed acceptable for the given purpose of application).

During verification the model is tested to find and fix errors in the implementation of the model

Definition of Verification

Determining whether the conceptual simulation model (and model assumptions) has been correctly translated into a “computer” or executable program, i.e. including debugging the simulation computer program.

In other words, for your project:

- Does the SimPN model behave exactly as the Process model + Information model + Transitions specifications?

Verification activities

Observe animation of model

- Look if all ‘special’ behaviour is in the model
- Look if something goes wrong
- Assess the simulation output

Show model to expert

Ask reviewer

Summary of today's lecture

STEP 4: Executable model

- Introduction to SimPN
- Mapping of conceptual model to a SimPN model
- Solution patterns in SimPN
- Verification

Homework

For SimPN tutorial;

- Install SimPN