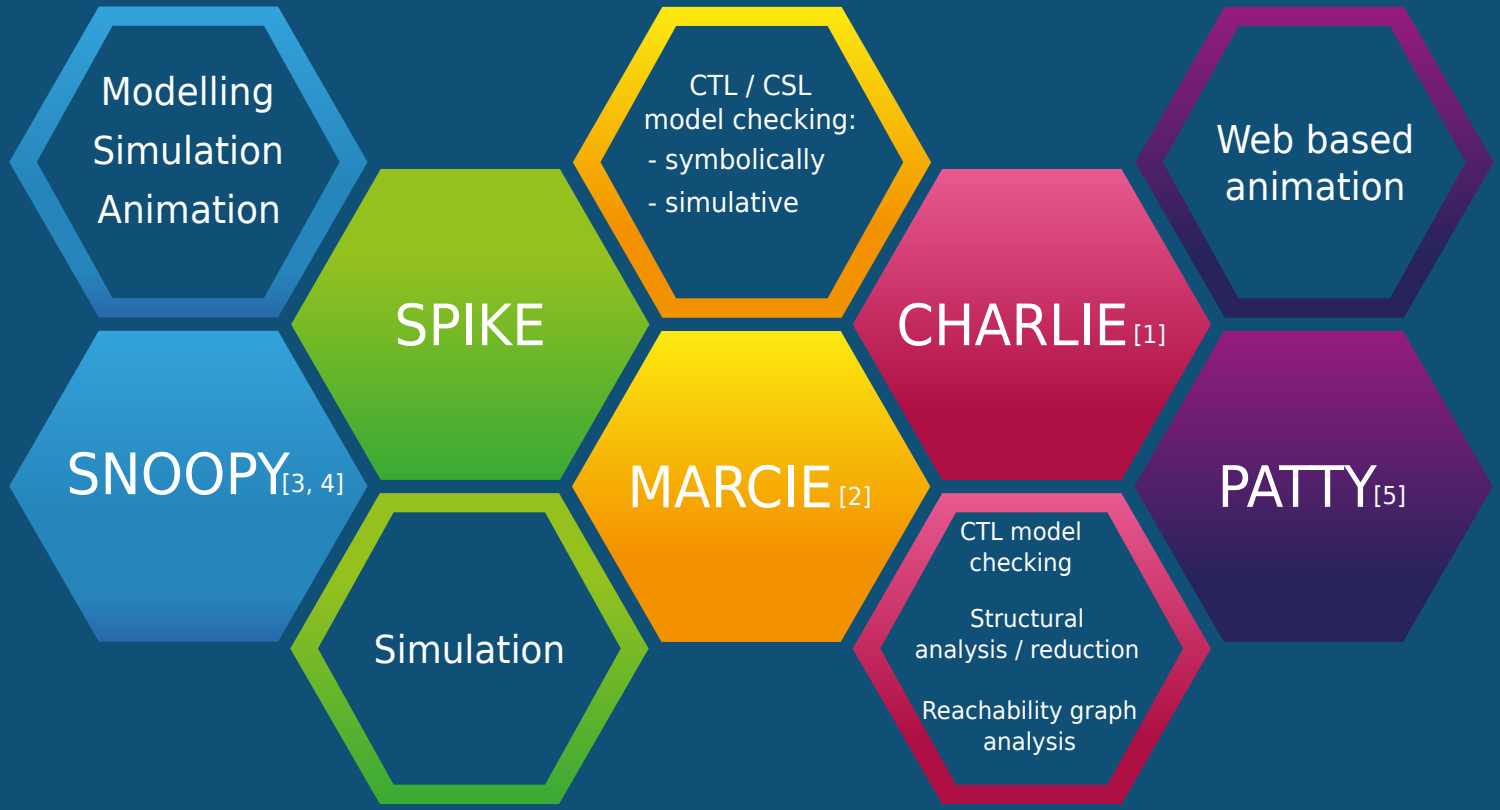
# SPIKE - a Supporting Tool for Parameter Optimization via Branched Simulations

**Jacek Chodak, Monika Heiner (supervisor)**
**Brandenburg University of Technology**
**Cottbus, Germany**
**jacek.chodak@b-tu.de, monika.heiner@b-tu.de**
**http://www-dssz.informatik.tu-cottbus.de/DSSZ/Software/Spike**

# PetriNuts Framework





**References:**

- [1] M Heiner, M Schwarick and J Wegener: PETRI NETS 2015
- [2] M Heiner, C Rohr and M Schwarick: PETRI NETS 2013
- [3] M Herajy, F Liu, C Rohr and M Heiner: BMC Systems Biology 2017
- [4] C Rohr, W Marvan, M Heiner: Bioinformatics 2010
- [5] K Schulz, BA thesis, BTU Cottbus, CS department 2008


**02**

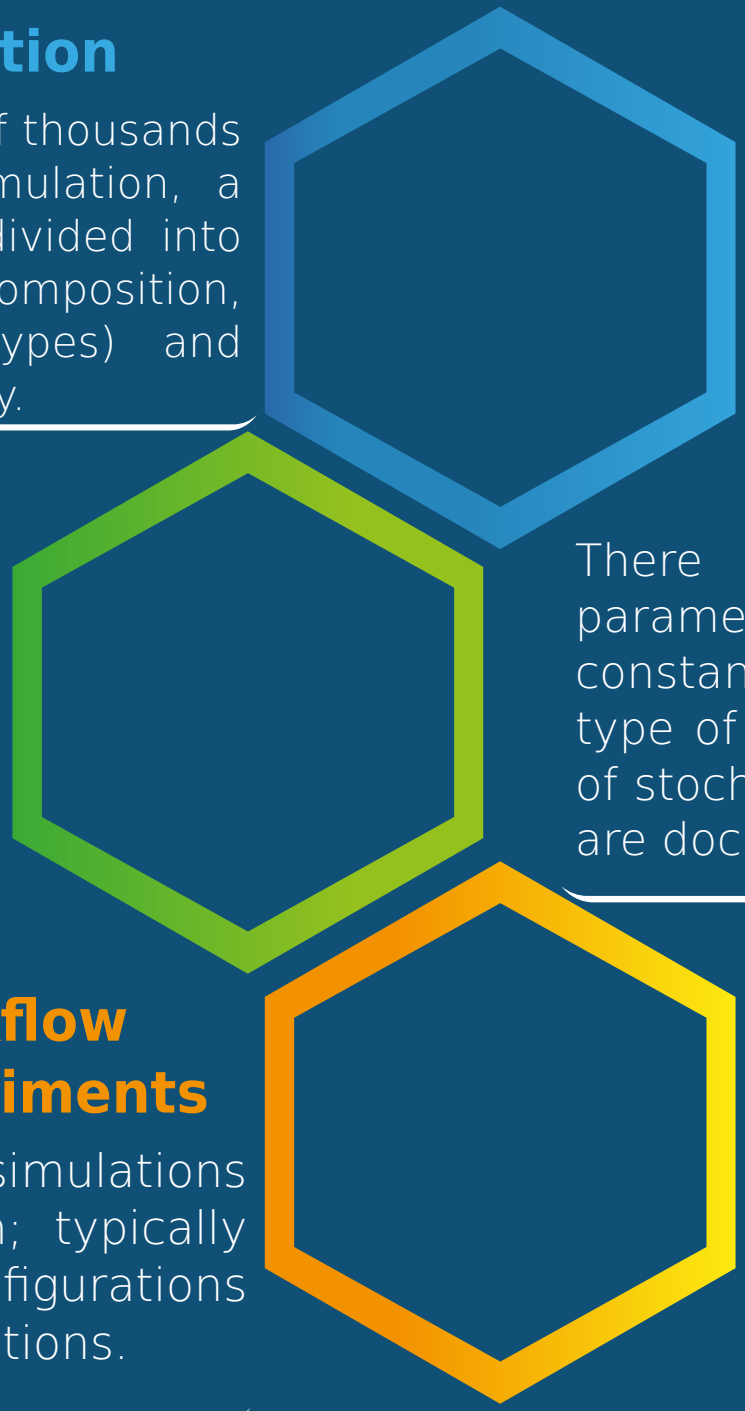
AWPN 2019


jacek.chodak@b-tu.de

# Why Spike

## Speed-up simulation

Models can contain dozens of thousands of nodes. To speed up simulation, a model can be reduced or divided into modules (spatial decomposition, decomposition by node types) and simulated in a distributed way.



## Reproducibility

There are many parameters: model parameters (e.g., initial marking, kinetic constants); simulations parameters (e.g., type of algorithm, length of trace, number of stochastic runs). Experiments with Spike are documented by configuration files.

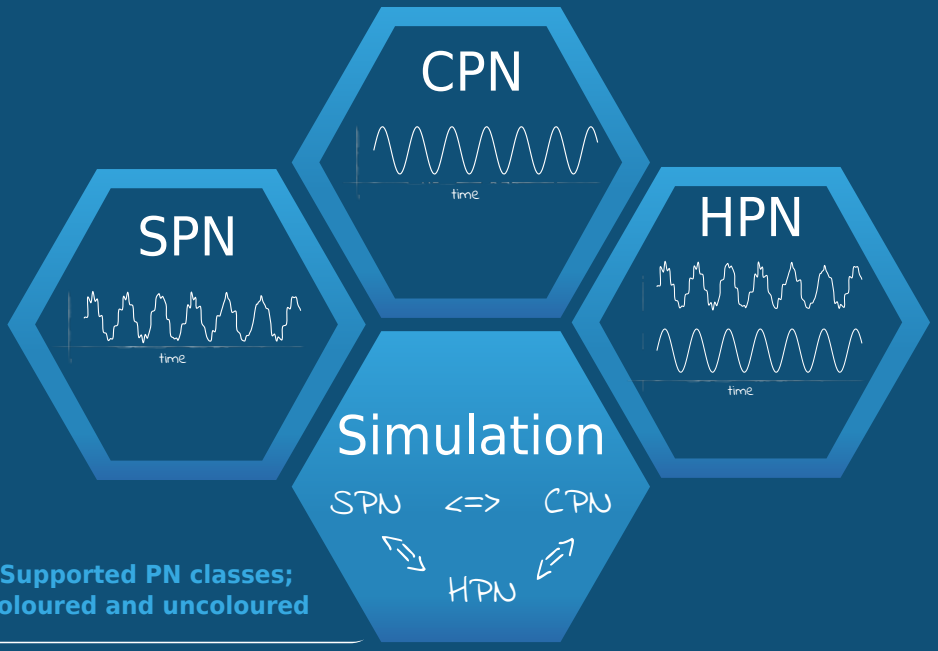
## Simplifying workflow of simulation experiments

CLI controlling multiple simulations without user intervention; typically for different model configurations and/or simulator configurations.

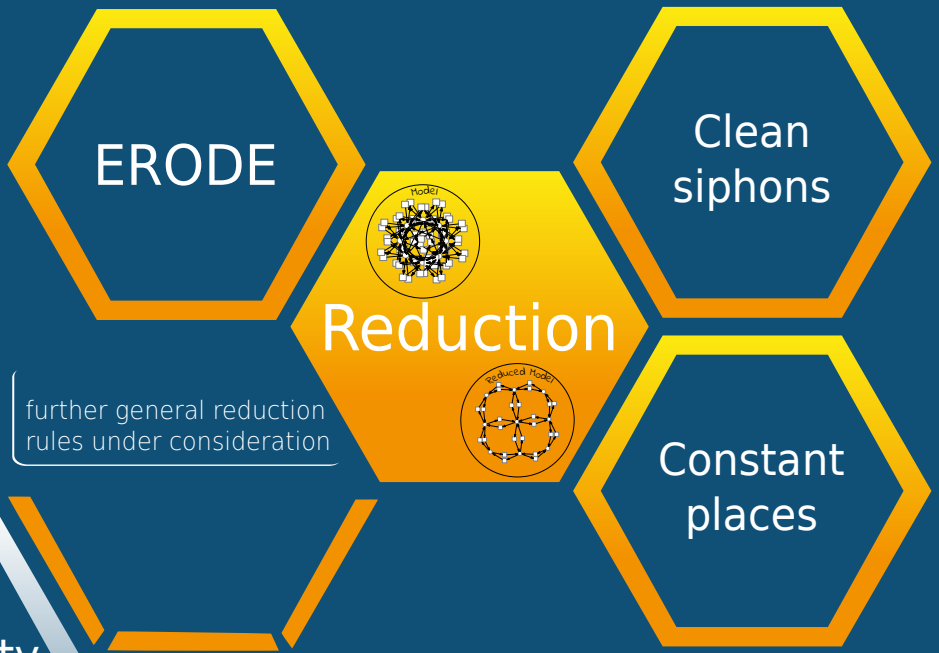


AWPN 2019
jacek.chodak@b-tu.de

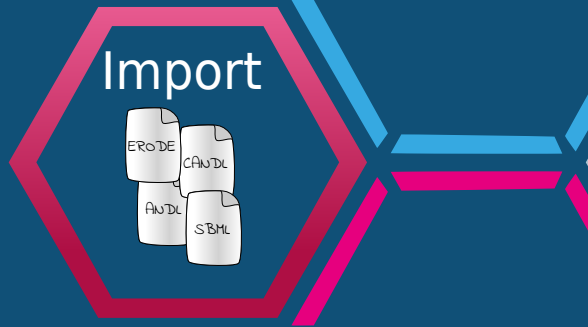
# Functionality

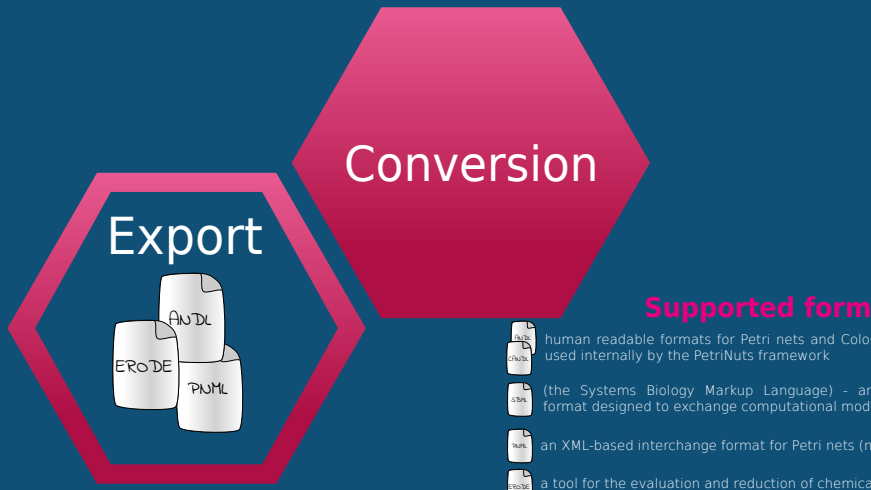



**Supported PN classes; coloured and uncoloured**



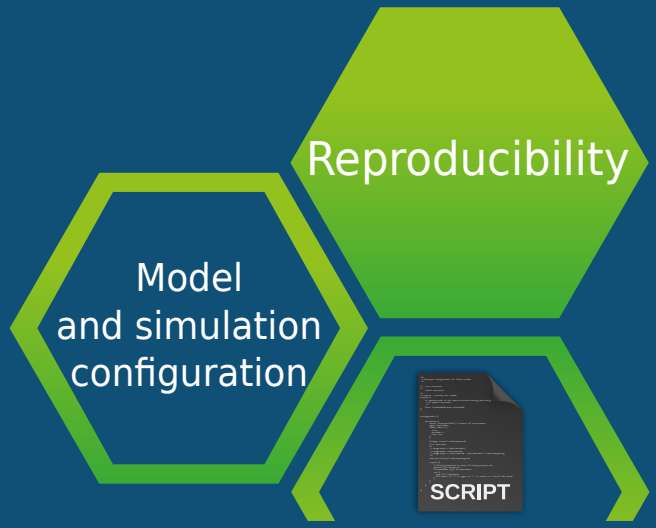

further general reduction rules under consideration




**Supported formats**

- human readable formats for Petri nets and Coloured Petri nets, respectively, used internally by the PetriNuts framework
- (the Systems Biology Markup Language) - an XML-based representation format designed to exchange computational models of biological processes
- an XML-based interchange format for Petri nets (no time)
- a tool for the evaluation and reduction of chemical reaction networks

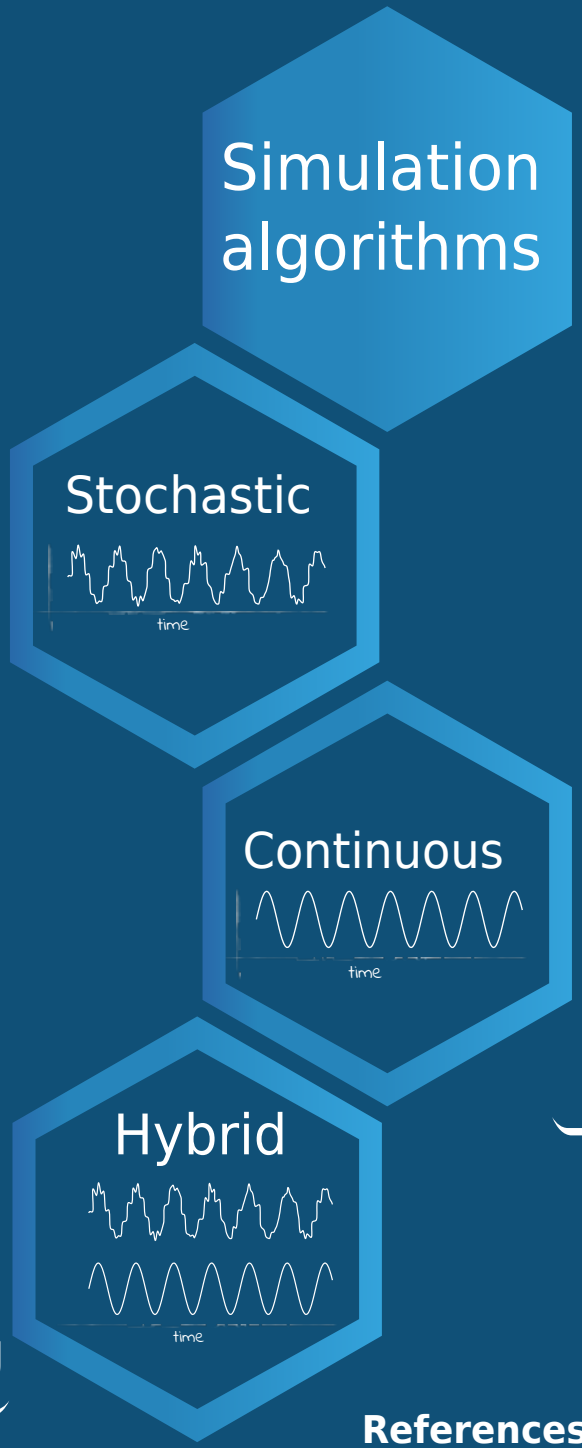



**04**

AWPN 2019

jacek.chodak@b-tu.de

# Supported simulation algorithms




1. Gillespie
2. τ-leaping
3. δ-leaping
4. FAU

1. BDF
2. Adams-Moulton
3. Rosenbrock
4. classic Runge-Kutta
5. Euler
6. two-steps Euler

1. exact
2. accelerated exact
3. HRSSA
4. accelerated HRSSA
5. dynamic partitioning

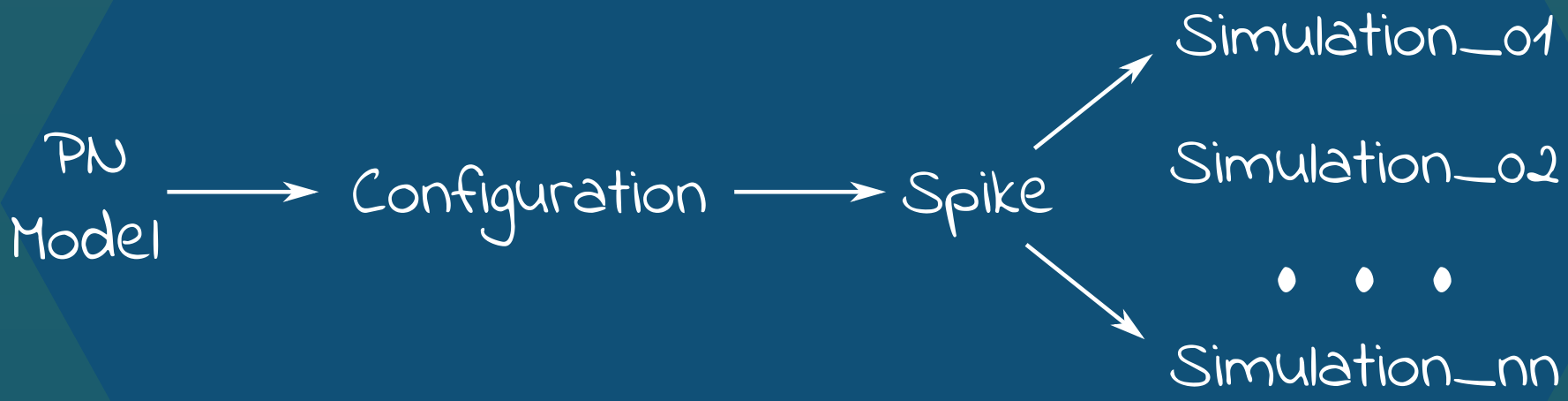
**References:**


- [1] http://www-dssz.informatik.tu-cottbus.de/publications/CMSB_2019/spike-CMSB-2019-appendix.pdf




jacek.chodak@b-tu.de

# Configuration

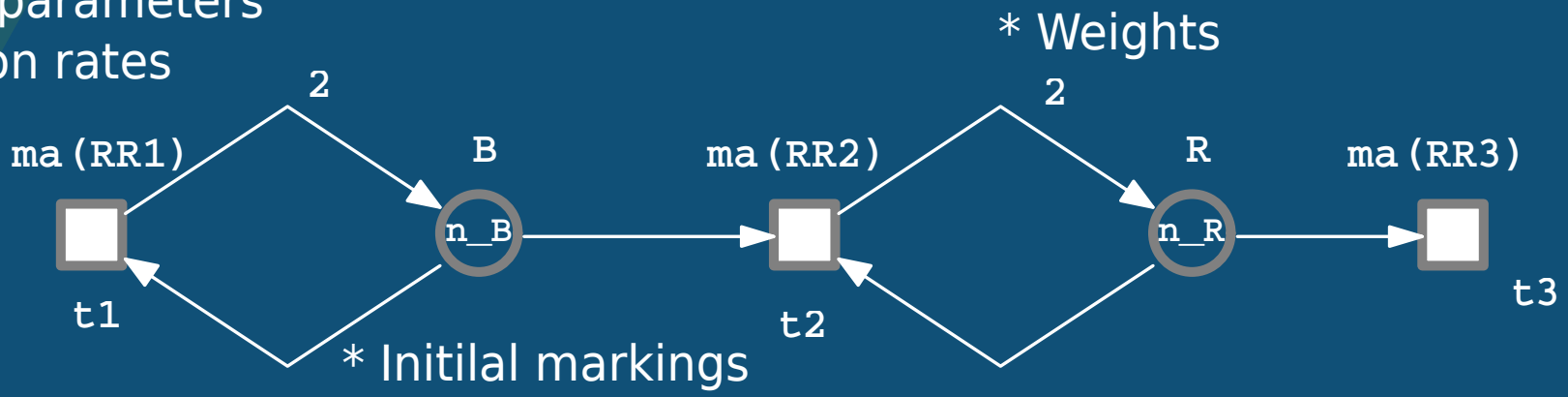




jacek.chodak@b-tu.de

# Configuration

* Kinetics parameters of reaction rates




* Weights

* Initilal markings


jacek.chodak@b-tu.de

# Configuration

definition of (named) constants

definition of observers (auxiliary variables)

```
configuration: {
  model: {
    constants: {
      ...
    }
    places: {
      ...
    }
    observers: {
      ...
    }
  }

  simulation : {
    ...
    export: {
      ...
    }
  }
  ...
}
```

specification of model parameters and simulator options

specification of multiple exports of simulation results


jacek.chodak@b-tu.de

# Configuration

```
constants: {
  // name of a group
  all: {
    /* if constant does not exist
    * then it will be created and
    * can be used in the configuration ,
    * for example in defining a place marking
    */
    M: ”D/2 + 1”;
  }
  kineticParam: {
    Threshold: 1200;
  }
}
```

## definition of (named) constants

which can get a specific value or a set of values,
either specified by a list or an interval;
supported data types: boolean, integer, real, string


jacek.chodak@b-tu.de

# Configuration

```
places: {
  // example of use of the newly created constant M
  P: "1000`(M,M)";
  P_2_2: 500;
}
```

## specification of model parameters

using either constants or (direct) values as arguments; values can be given as a single specific value or a set of values, specified by a list or an interval


jacek.chodak@b-tu.de

# Configuration

```
observers: {
 place: {
  OP01: {
   function: "P_1_1 + P_2_3";
  }
 }
}
```

**definition of observers (auxiliary variables)**

allow for extra measures by defining numerical functions; depending on the type of observer, it can involve, beside constants, places, transitions or simultaneously places and transitions


jacek.chodak@b-tu.de

# Configuration

```
simulation: {
  name: "exampleDiffusion_2D4"; // Name of a simulation
  type: stochastic; // continuous, stochastic, hybrid
  solver:
    direct: {
      // if 0 try to use as many thread as CPU cores
      threads: 0;
      runs: 10;
    }
    // range: 0:2:100;// start:step:end
    // or intervals
    /*
     * range.start = interval.start;
     * range.end = interval.end;
     * range.step = (interval.end - interval.start)
     *              / interval.splitting;
     */
     varSplit: [10:5:20];
     interval: 0:varSplit:100;// start:spliting:end
  ...
}
```

## simulator options

using either constants or (direct) values as arguments; values can be given as a single specific value or a set of values, specified by a list or an interval


jacek.chodak@b-tu.de

# Configuration

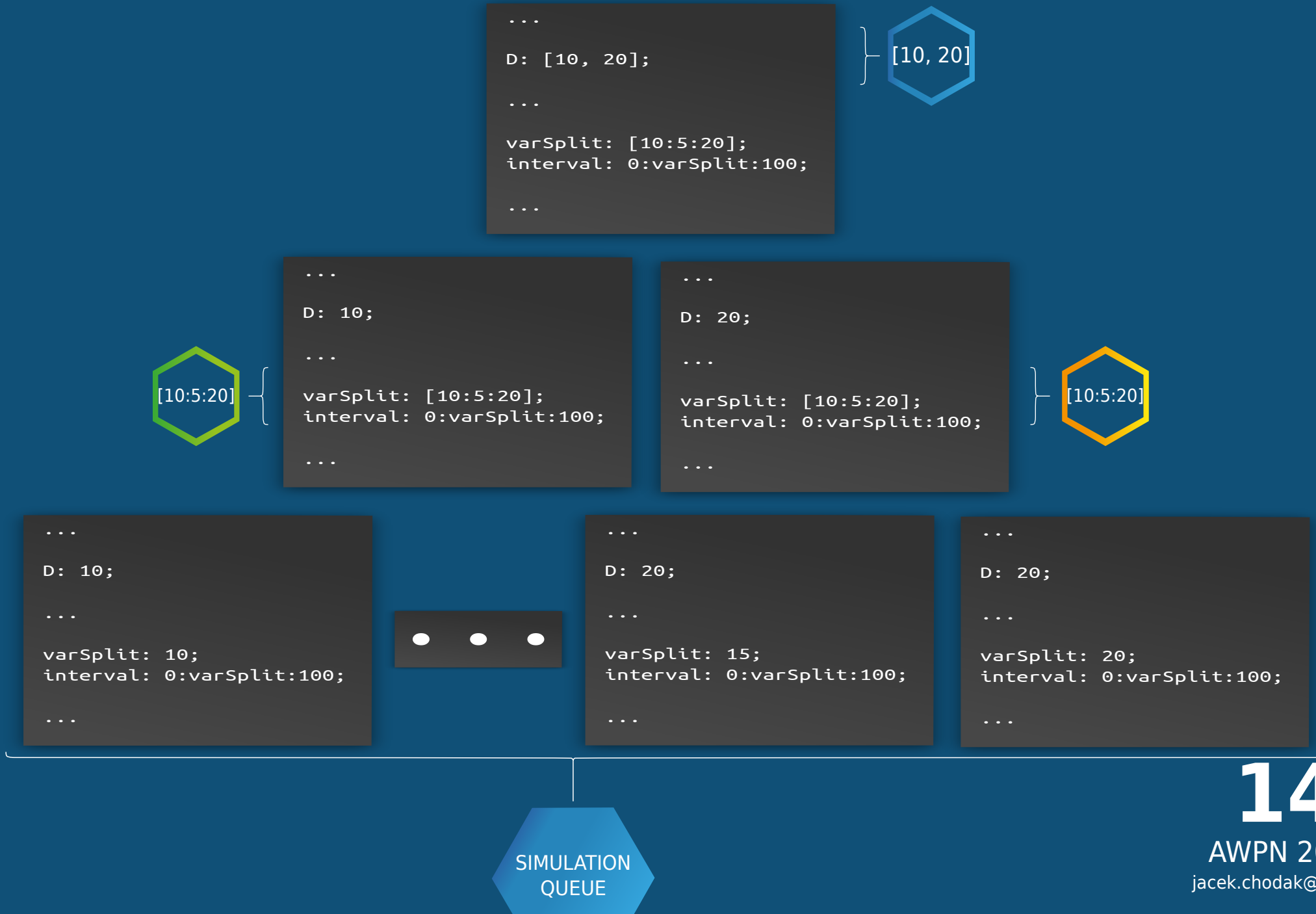
```
export: {
  /*
   * export places with given prefix P_1
   * or which belong to the colour set Grid2D
   */
   places: ["P_1.*", "Grid2D"];
   transitions: c["t3"];// given coloured transitions
   transitions: u["t3_1_1_1_2"];// given uncoloured transitions
   observers: [];// export all observers
   csv: {
     sep: ";";// Separator
     file: import.name
             << "_places"
             << "_" << type
             << "_" << solver
             << "_" << solver.runs
             << "x" << interval.end
             << "x" << model.constants.all.D
             << ".csv";// File name
   }
}

export: { ... }
```

**specification of multiple exports of simulation results**

by use of regular expressions over the nodes of which the simulation traces are to be recorded; it is possible to combine the results of places, transitions and observers, coloured and uncoloured, in one CSV file


jacek.chodak@b-tu.de

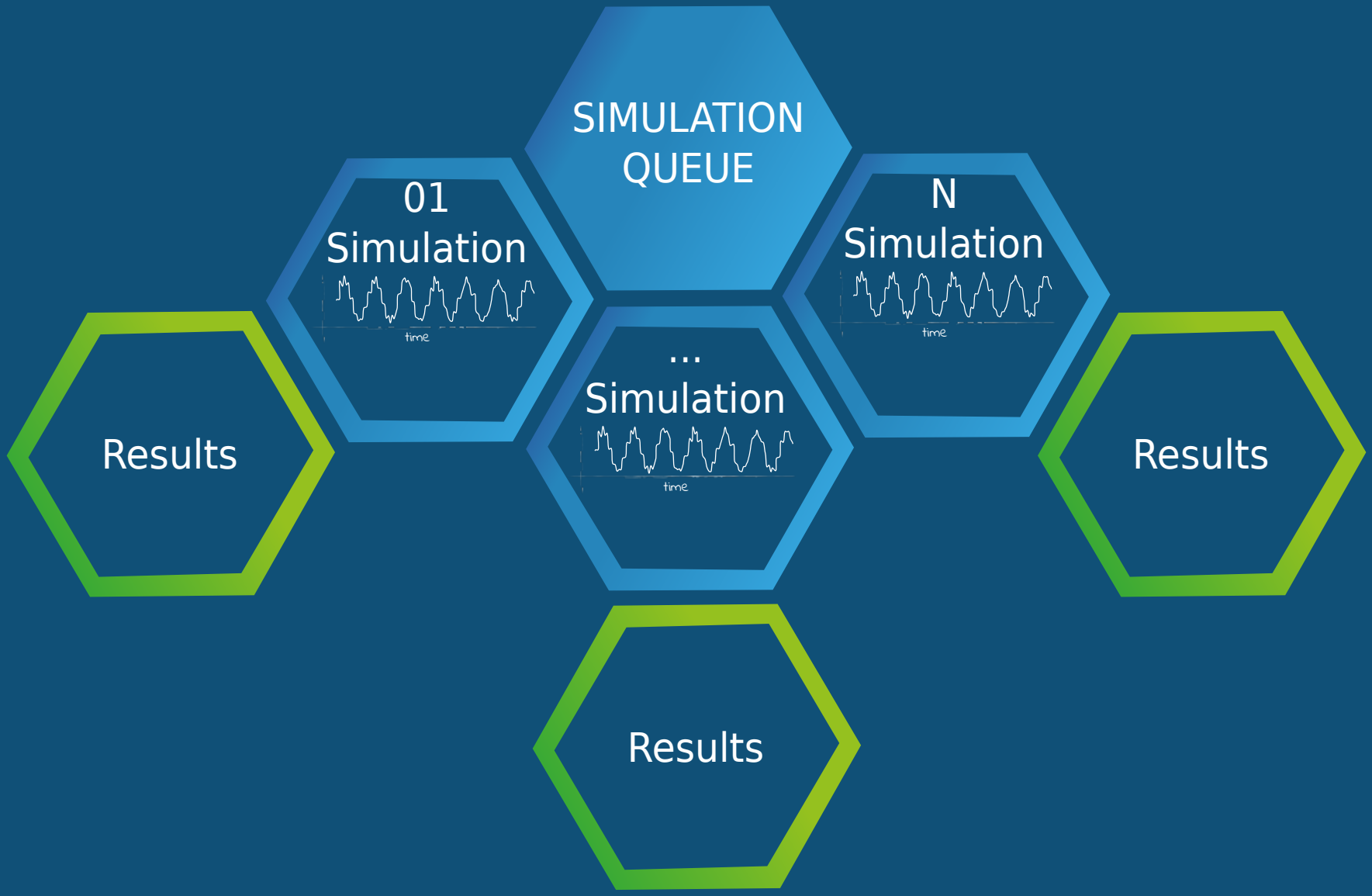
# How branching works





jacek.chodak@b-tu.de

# How branching works

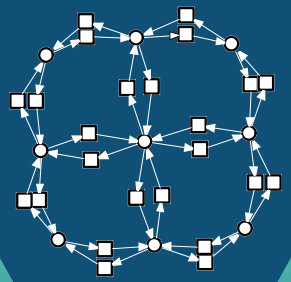




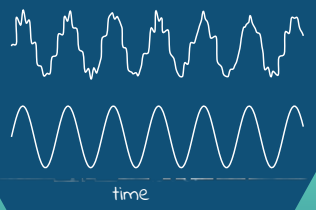
jacek.chodak@b-tu.de

# Parameter optimization through simulation

Model



WetLab Data




Simulation

Model behaviour

Throught parameter optimization like kinetics prameters of reaction rates it is posible to compare model behaviour with data from the wet lab.


jacek.chodak@b-tu.de

# Parameter optimization through simulation

Algorithm 1: Multiple parameter optimisation - brute force

```
1  Load model;
2  Determine simulator configuration;
3  for each unique combination of parameter values do
4  |  Determine model configuration;
5  |  Create new configuration branch;
6  |  Run simulation;
7  |  Save results of the simulation;
8  end
9  for each stored results do
10 |  if results not fit then
11 |  |  Remove results;
12 |  end
13 end
```


jacek.chodak@b-tu.de

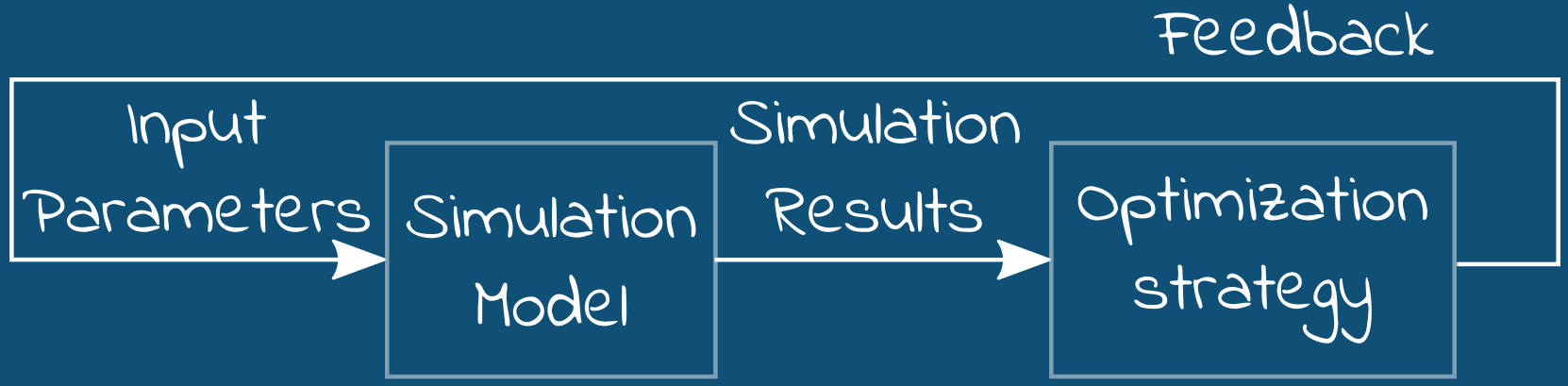
# Parameter optimization through simulation

Algorithm 2: Multiple parameter optimisation - space reduction.

```
1  Load modeL;
2  Determine simulator configuration;
3  repeat
4  |   for each unique combination of parameter values do
5  |   |   Determine model configuration;
6  |   |   Create new configuration branch;
7  |   |   Run simulation;
8  |   end
9  |   Run optimization strategy;
10 until results fit;
```


jacek.chodak@b-tu.de

# Parameter optimization through simulation

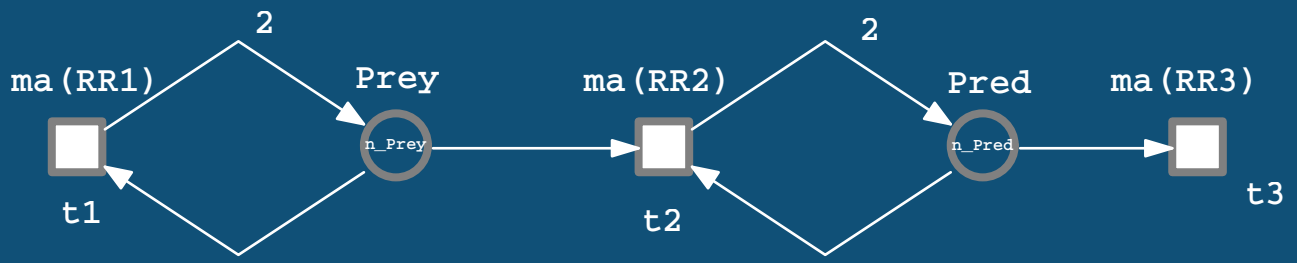




jacek.chodak@b-tu.de

# Example

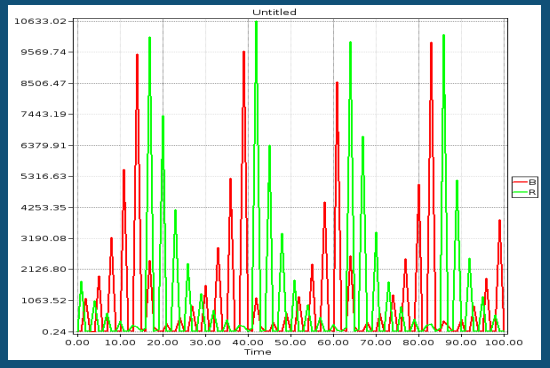
Lotka–Volterra (predator–prey) model




is represented by a pair of first-order nonlinear differential equations, frequently used to describe the dynamics of biological systems in which two species interact, one as a predator and the other as prey.

$$\frac{dPrey}{dt} = 2 * (RR1 * Prey) - (RR1 * Prey) - (RR2 * Prey * Pred)$$

$$\frac{dPred}{dt} = 2 * (RR2 * Prey * Pred) - (RR2 * Prey * Pred) - (RR3 * Pred)$$





jacek.chodak@b-tu.de

# Example: model

```
cpn  [lotka_volterra]
{
  constants:
  all:
    double RR1 = 0;
    double RR2 = 0;
    double RR3 = 0;
  marking:
    double n_Prey = 6;
    double n_Pred = 15;

  places:
  continuous:
    Prey = n_Prey;
    Pred = n_Pred;

  transitions:
  continuous:
    t1
      :
      : [Prey + 2] & [Prey - 1]
      : MassAction(RR1)
      ;
    t2
      :
      : [Pred + 2] & [Prey - 1] & [Pred - 1]
      : MassAction(RR2)
      ;

    t3
      :
      : [Pred - 1]
      : MassAction(RR3)
      ;

}
```

corresponding ODEs are automatically generated


jacek.chodak@b-tu.de

# Example: optimization strategy

GENE_SIZE = 3
POPULATION_SIZE = 101
MAX_EPOCH = 1000
EXPECTED_FITNESS = -0.2

$$fitness = -|PRED - \bar{pred} + S - std(\vec{pred})| - |PREY - \bar{prey} + S - std(\vec{prey})|$$

PRED = 20 - expected avarege number of predators
PREY = 5 - expected avarege number of preys
S = 3 - expected standard deviation
std() - standard deviation

$\vec{pred}$ - number of predators & preys
$\vec{prey}$ over simulation time


jacek.chodak@b-tu.de

# Example: sim configuration

```
...
configuration: {
    model: {
      constants: {
        all: [
          population_inhabitant_0: {
            RR1: 11.319955;
            RR2: 20.726186;
            RR3: 34.832076;
          },
          population_inhabitant_1: {
            RR1: 5.497761;
            RR2: 37.746638;
            RR3: 8.285595;
          },
          ...
        ];
      }
    }
}
...
```


jacek.chodak@b-tu.de

# Example: Run

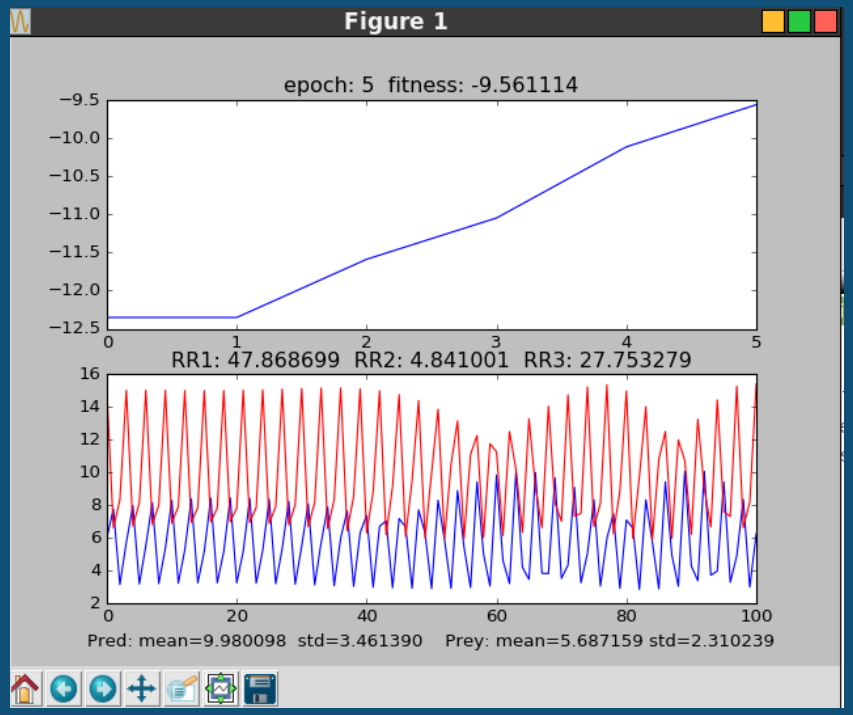
## spike - status

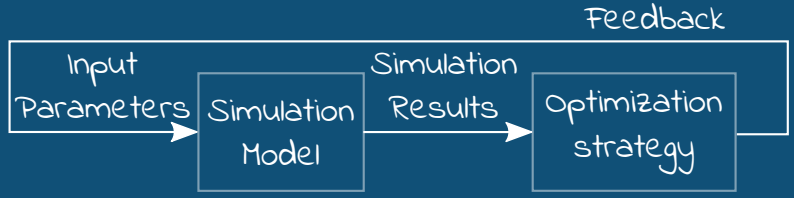
```
python main-spike.py
File  Edit  View  Search  Terminal  Help
> [20:49:51] [EVENT] [0x000048b1]:: loaded::model: lotka_volterra_3_02_1
660: <NOTIFICATION> [20:49:51] [EVENT] [0x000048ae]:: REMOTE: 146: <NOTIFICATION
> [20:49:51] [EVENT] [0x000048b5]:: loading file: ../model/lotka_volterra-3-02-1
.andl
661: <NOTIFICATION> [20:49:51] [EVENT] [0x000048ae]:: REMOTE: 147: <NOTIFICATION
> [20:49:51] [EVENT] [0x000048b5]:: loaded::model: lotka_volterra_3_02_1
662: <NOTIFICATION> [20:49:51] [EVENT] [0x000048ae]:: REMOTE: 148: <NOTIFICATION
> [20:49:51] [EVENT] [0x000048b1]:: simulation::start: PP
663: <NOTIFICATION> [20:49:51] [EVENT] [0x000048ae]:: REMOTE: 149: <NOTIFICATION
> [20:49:51] [EVENT] [0x000048b1]:: simulation::type: continuous
664: <NOTIFICATION> [20:49:51] [EVENT] [0x000048ae]:: REMOTE: 150: <NOTIFICATION
> [20:49:51] [EVENT] [0x000048b1]:: simulation::solver: BDF::bio
665: <NOTIFICATION> [20:49:51] [EVENT] [0x000048ae]:: REMOTE: 13: <> [20:49:51]
[PROGRESS] [0x000048b1]:: simulation in progress:  ...
666: <NOTIFICATION> [20:49:51] [EVENT] [0x000048ae]:: REMOTE: 148: <NOTIFICATION
> [20:49:51] [EVENT] [0x000048b5]:: simulation::start: PP
667: <NOTIFICATION> [20:49:51] [EVENT] [0x000048ae]:: REMOTE: 149: <NOTIFICATION
> [20:49:51] [EVENT] [0x000048b5]:: simulation::type: continuous
668: <NOTIFICATION> [20:49:51] [EVENT] [0x000048ae]:: REMOTE: 150: <NOTIFICATION
> [20:49:51] [EVENT] [0x000048b5]:: simulation::solver: BDF::bio
669: <NOTIFICATION> [20:49:51] [EVENT] [0x000048ae]:: REMOTE: 13: <> [20:49:51]
[PROGRESS] [0x000048b5]:: simulation in progress:  ...
```

```
htop
File  Edit  View  Search  Terminal  Tabs  Help
htop                    hadron@cavalli: ~/dev/dssd-workspace/param-optimization/Simple...

  1  [|||||||||||||||          49.7%]   5  [||||||||||||          33.1%]
  2  [||||||                   17.0%]   6  [||||||||||||||||||    61.9%]
  3  [||||||||||||||||||       56.9%]   7  [|||||||               20.9%]
  4  [|||||||||                26.5%]   8  [|||||||||||||||       48.7%]
  Mem[|||||              1.86G/30.9G]   Tasks: 111, 398 thr; 1 running
  Swp[                      0K/15.4G]   Load average: 1.59 1.33 1.04
                                        Uptime: 02:39:07

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
16894 hadron     20   0 1024M 13884  8432 S 72.8  0.0  0:02.15 /home/hadron/dev/
16891 hadron     20   0 1024M 13880  8432 S 72.1  0.0  0:02.15 /home/hadron/dev/
16892 hadron     20   0 1024M 13944  8496 S 72.1  0.0  0:02.15 /home/hadron/dev/
16890 hadron     20   0 1024M 13880  8432 S 72.1  0.0  0:02.14 /home/hadron/dev/
16961 hadron     20   0 1024M 13944  8496 S 10.4  0.0  0:00.24 /home/hadron/dev/
16958 hadron     20   0 1024M 13880  8432 S  9.7  0.0  0:00.24 /home/hadron/dev/
16955 hadron     20   0 1024M 13880  8432 S  9.1  0.0  0:00.24 /home/hadron/dev/
16959 hadron     20   0 1024M 13884  8432 S  9.1  0.0  0:00.24 /home/hadron/dev/
16887 hadron     20   0  160M 11668 10096 S  3.9  0.0  0:00.20 ../../spike/bin/s
16889 hadron     20   0  160M 11668 10096 S  3.9  0.0  0:00.10 ../../spike/bin/s
16904 hadron     20   0 1024M 13944  8496 S  0.6  0.0  0:00.01 /home/hadron/dev/
16936 hadron     20   0 1024M 13944  8496 S  0.0  0.0  0:00.25 /home/hadron/dev/
F1Help  F2Setup  F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

## system process viewer

## simulation results

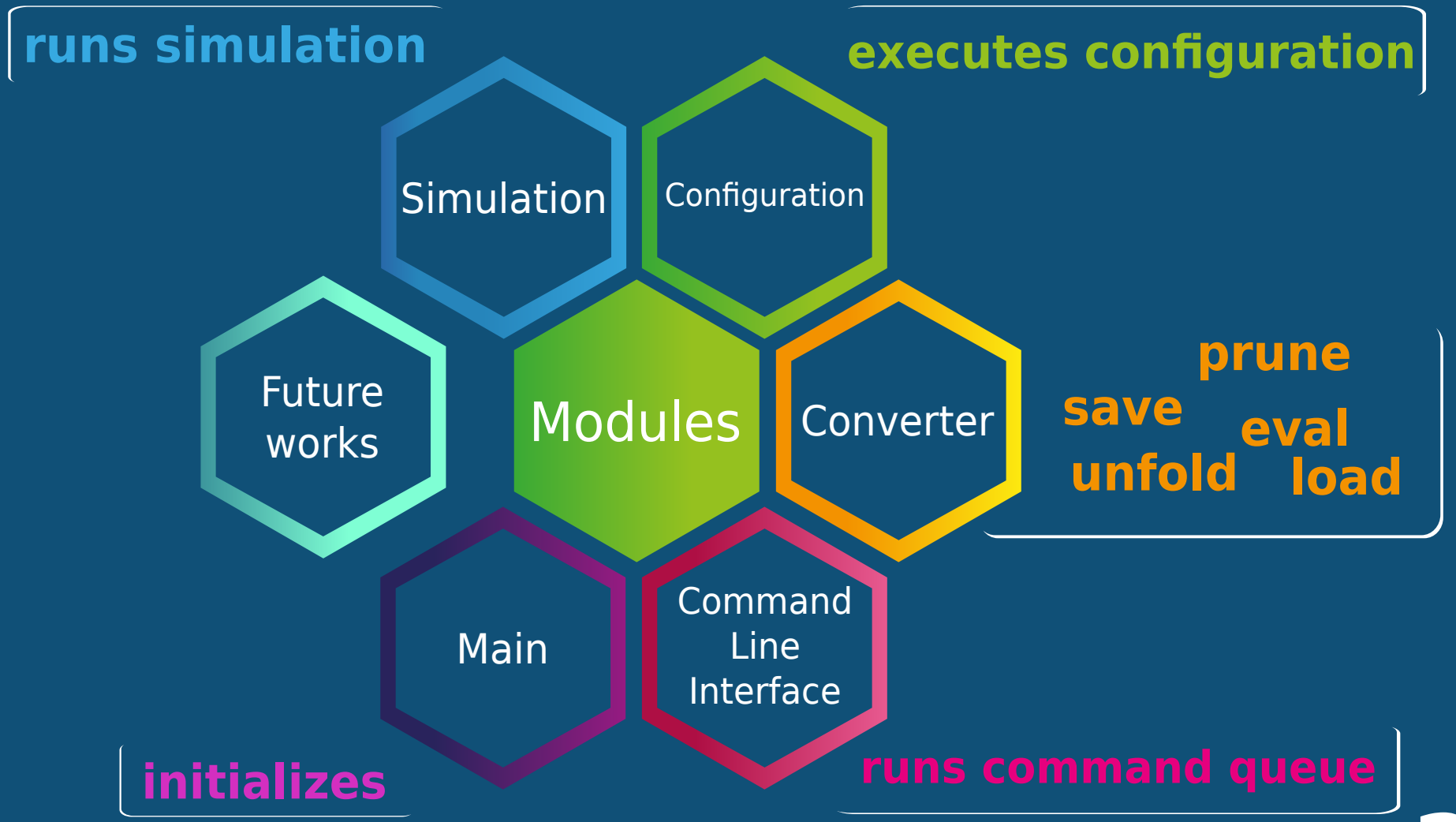



The execution loop is steered by Python script. The optimization strategy and visualisation of results is embedded in the script.


jacek.chodak@b-tu.de

# Architecture

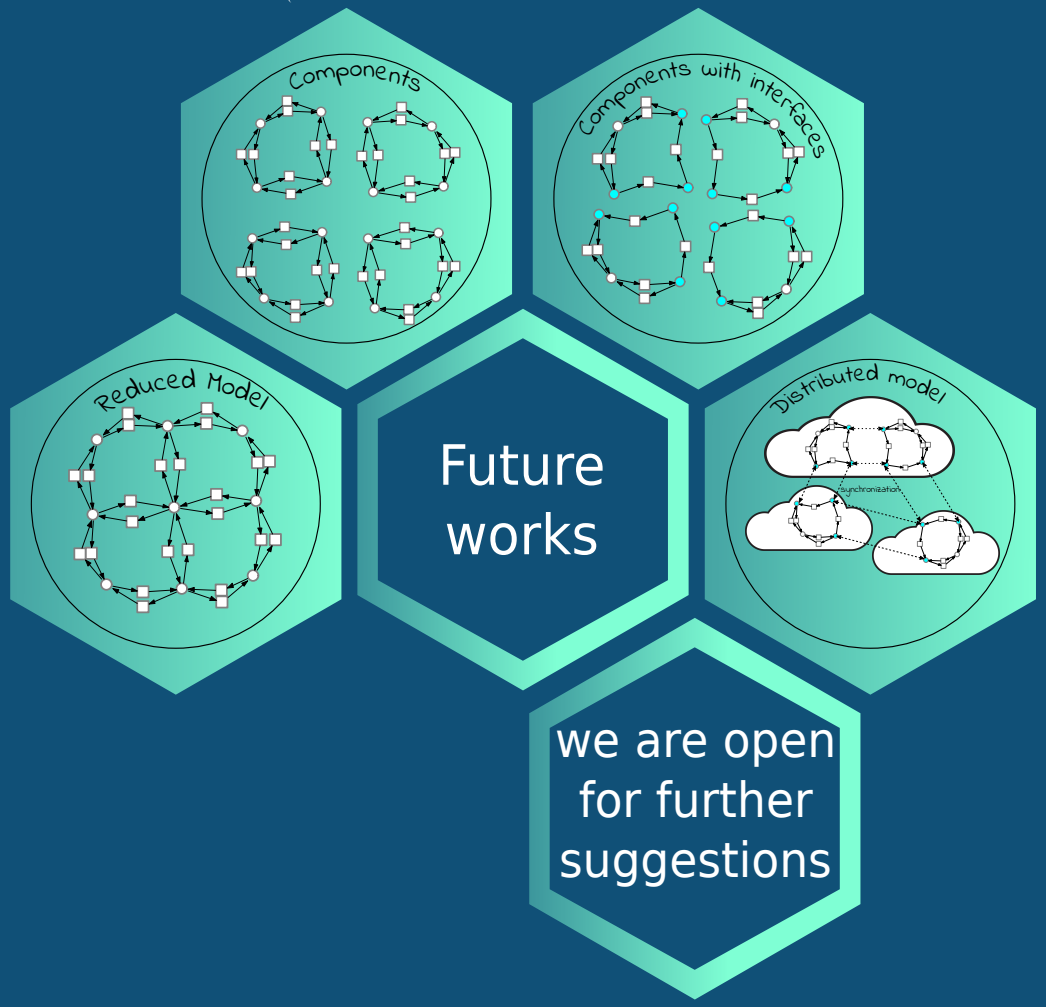




jacek.chodak@b-tu.de

# Future works

**model decomposition**




**sophisticated model reduction**

**distributed simulation**

either for a set of simulations or a decomposed model


jacek.chodak@b-tu.de

# Thank you
# for your attention


jacek.chodak@b-tu.de