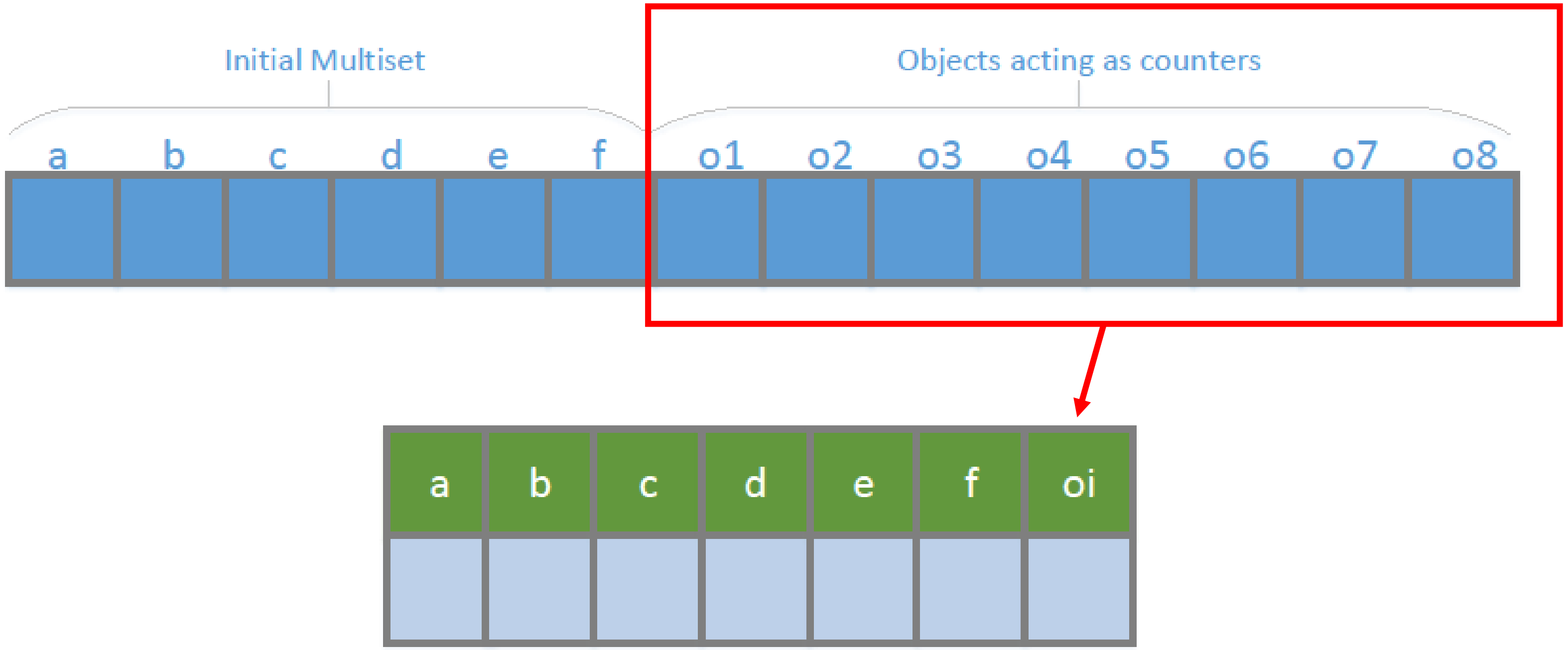
Adaptive GPU Simulators

Miguel Á. Martínez-del-Amor, Ignacio Pérez-Hurtado,
David Orellana-Martín, Agustín Riscos-Núñez,
Mario J. Pérez-Jiménez

Research Group on Natural Computing, Universidad de Sevilla

Motivation

- Parallel simulators for P systems require wasting resources for:
  - Handling **worst-case** scenarios
  - **Maximal** parallelism
  - Global **synchronization**
  - ….
- For example: **sparse** vs **dense** representation of multisets (discussed in CMC19)



Idea

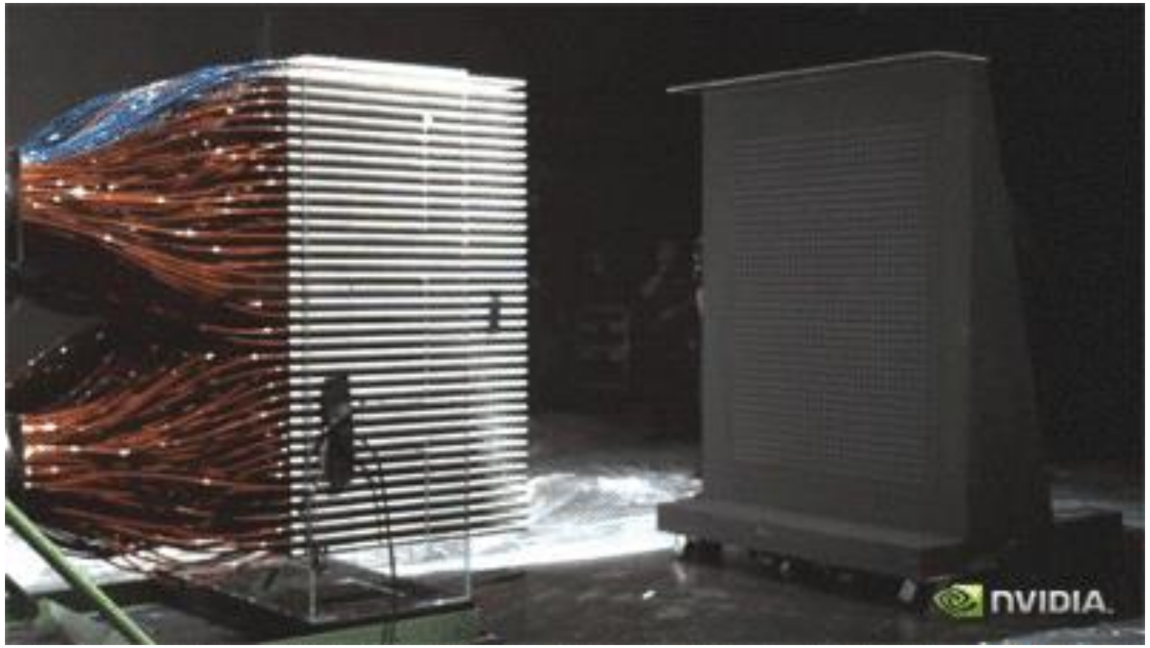
- Most P system based solutions are constructed using an **algorithmic scheme:**
  - **Stages** in solutions to NP-problems: e.g a solution to SAT with AM uses *generation, synchronization, check-out and output stages*
  - **Modules** in models of biological systems: e.g. in tritrophic interactions there are *reproduction, feeding, migration, mortality*…
- The **designer** of the P system has this interesting information
- **Current simulators** are agnostic to this ….

Outline

- Context
    - GPU computing basics
    - GPU-parallel simulators P systems
    - Population Dynamics P systems
    - GPU simulator for PDP systems
- Modular GPU simulator for PDP systems
- P-Lingua 5’s *features*
- Provocative ideas

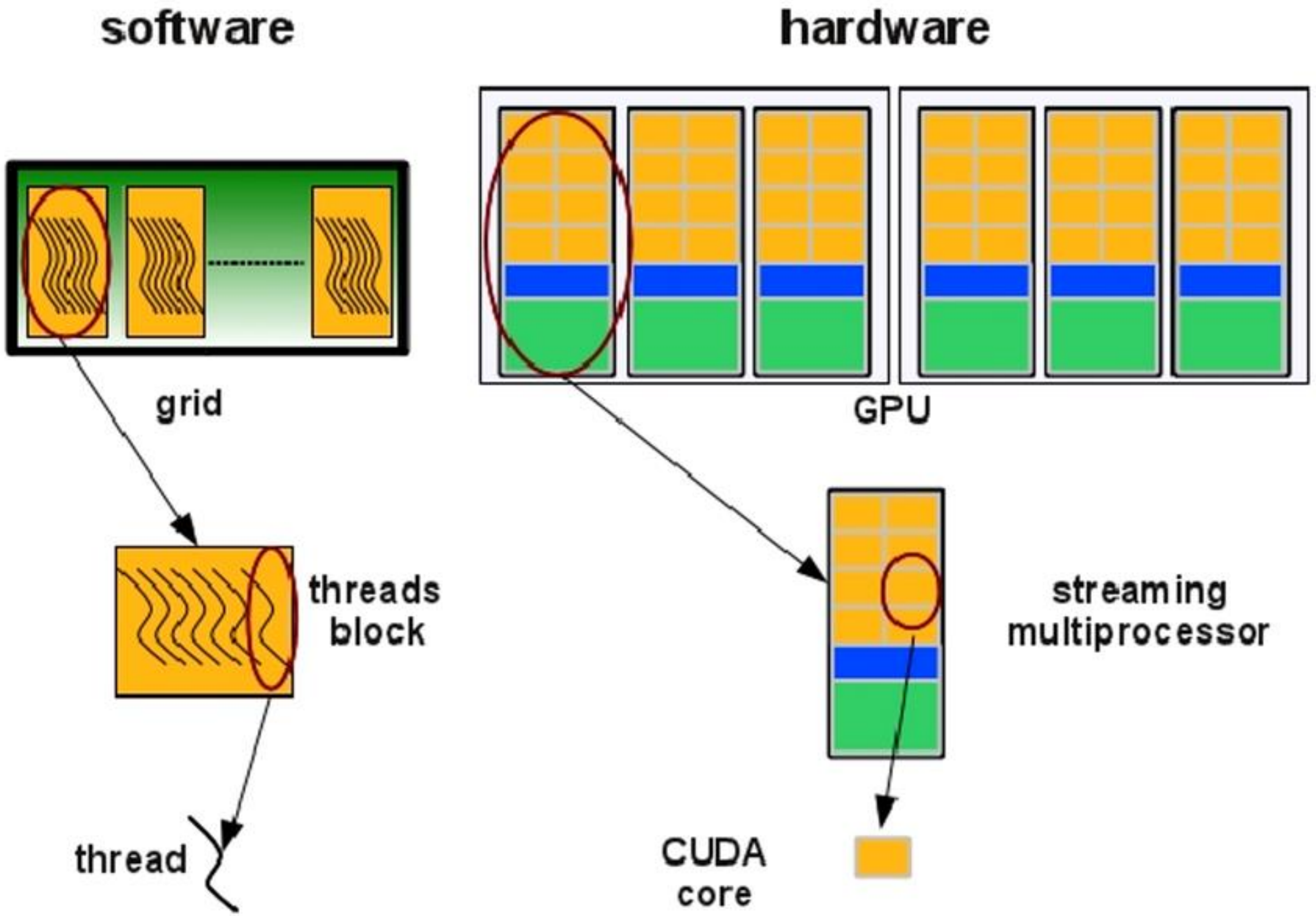
Context – GPU computing basics

- **GPU =** Graphics Processing Unit
- Features:
  - 700-3000 of **cores**
  - 4-16GB of **memory**
- Now a powerful parallel device for scientific computing, e.g.
  - Bio-inspired computing
  - Machine learning
  - Bitcoins



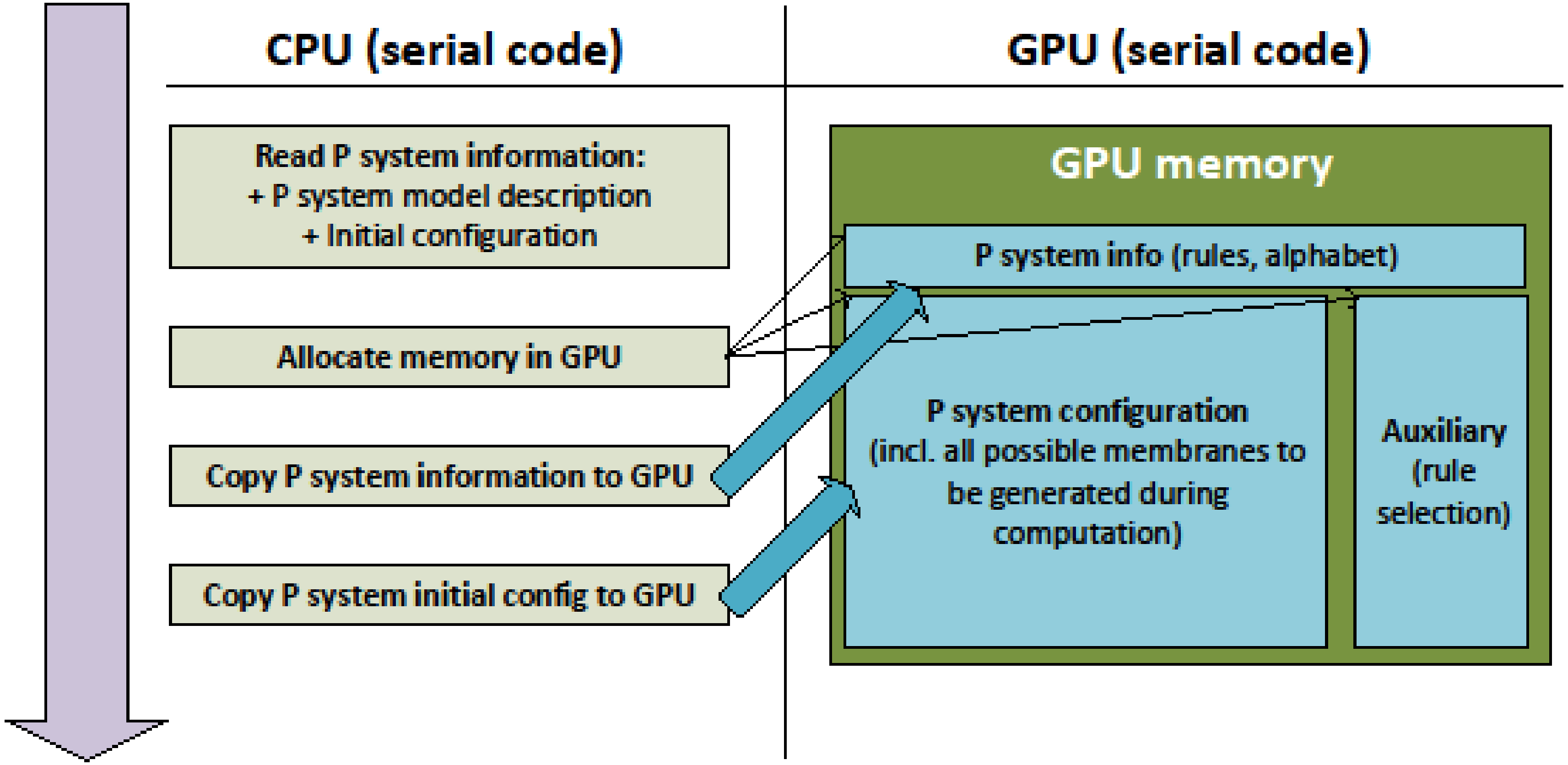
Context – GPU computing basics

- **CUDA** since 2017
- **Kernels** are C/C++ functions
- Thousands of lightweight threads execute a Kernel.
- Flexible **Data Parallelism**
- **Memory** hierarchy



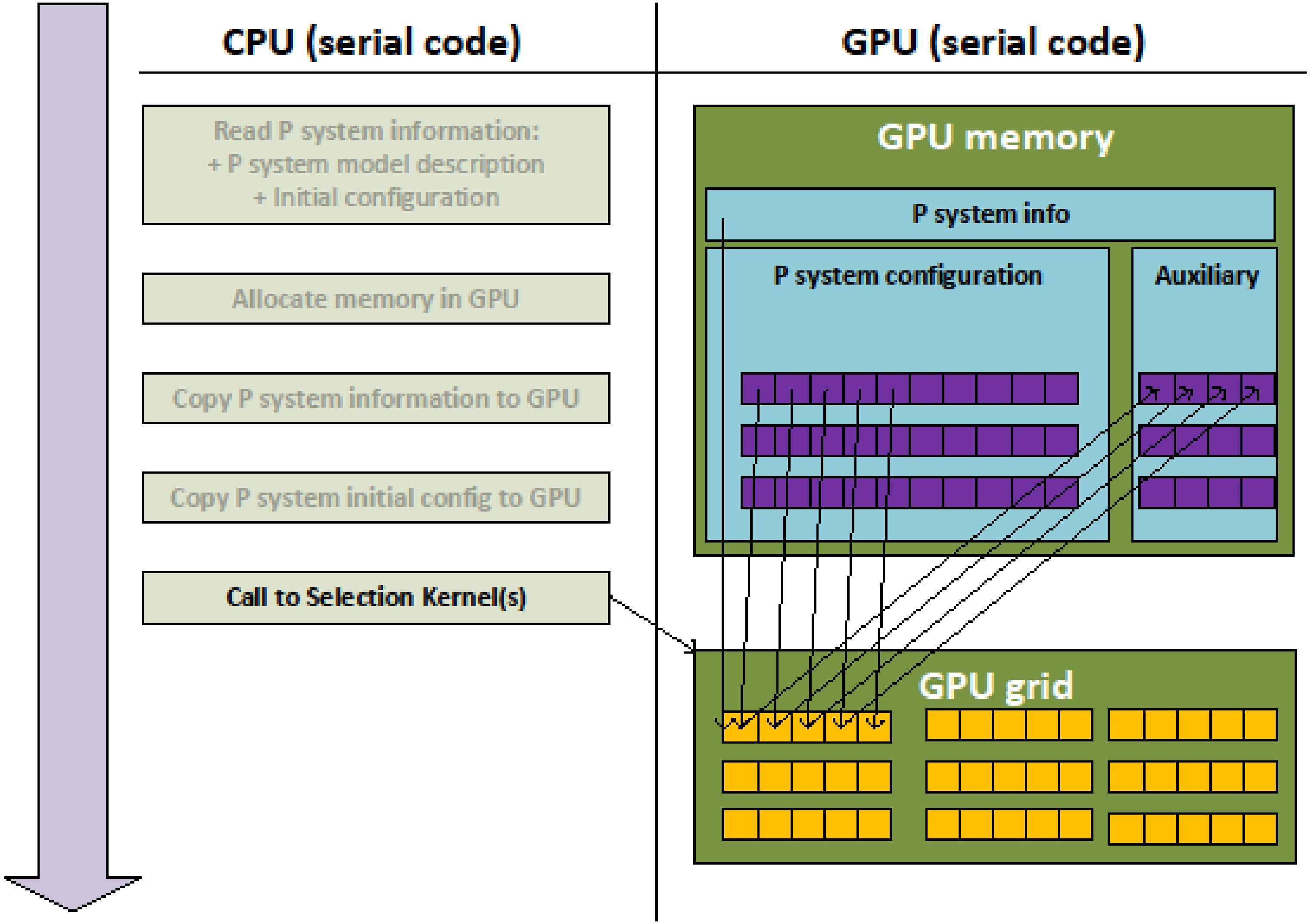
Context – GPU simulators of P systems

- General design
- **Step 1**



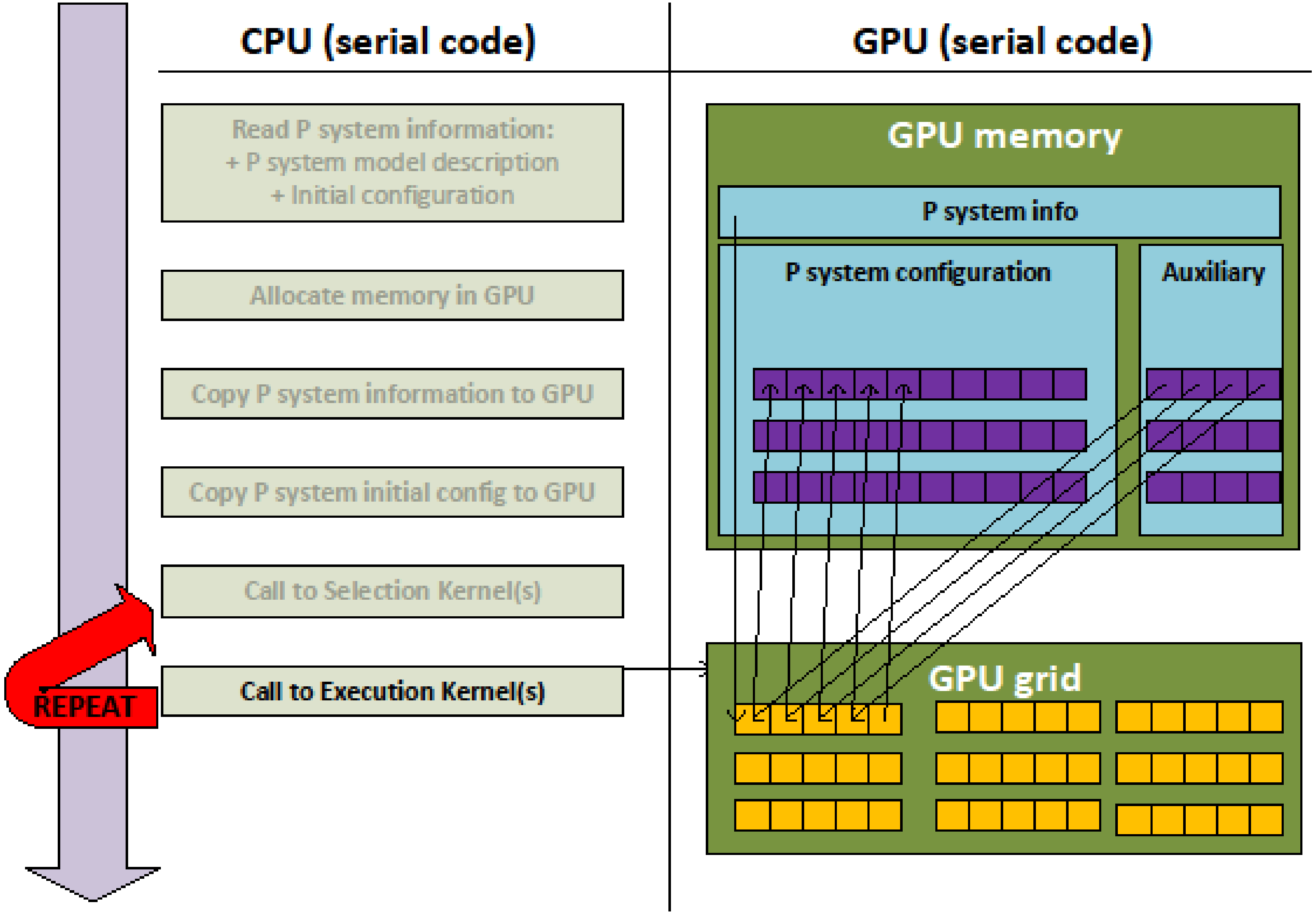
Context – GPU simulators of P systems

- General design
- **Step 2**



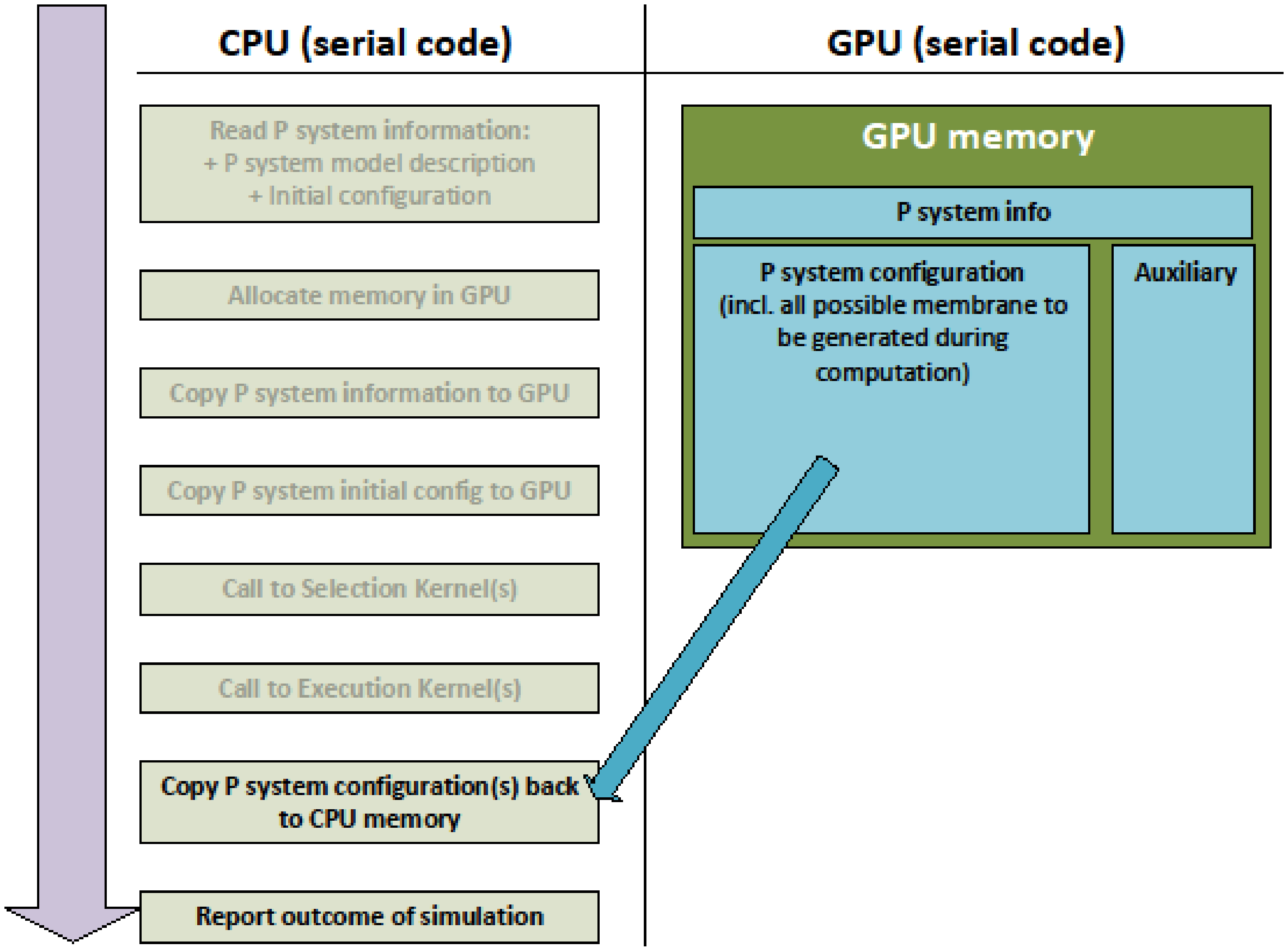
Context – GPU simulators of P systems

- General design
- **Step 3**



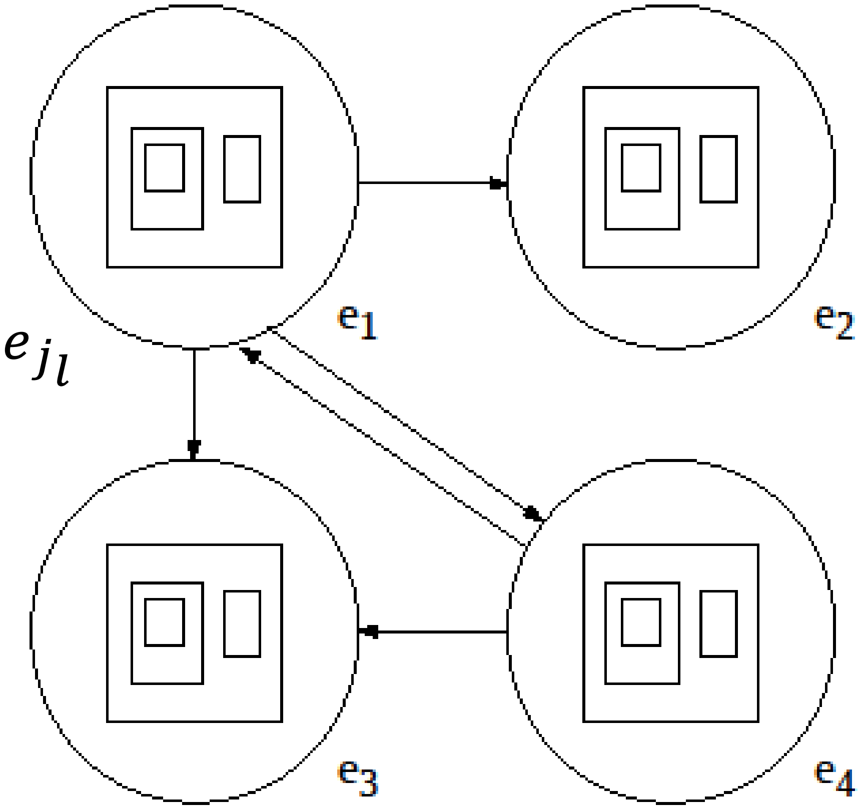
Context – GPU simulators of P systems

- General design
- **Step 4**



Context – Population Dynamics P Systems

- **Skeleton** rules: $u[v]_h^{\alpha} \rightarrow u'[v']_h^{\beta}$
- Rules in environments: $u[v]_h^{\alpha} \rightarrow^{f_{r,j}} u'[v']_h^{\beta}$
- **Communication** rules: $(x)_{e_j} \rightarrow^{p_r} (y_1)_{e_{j_1}} \ldots (y_l)_{e_{j_l}}$



- **Maximally** parallel application of rules
- **Blocks** of rules: same LHS
- **Probabilities** within a block.
- Requires several **simulations**.

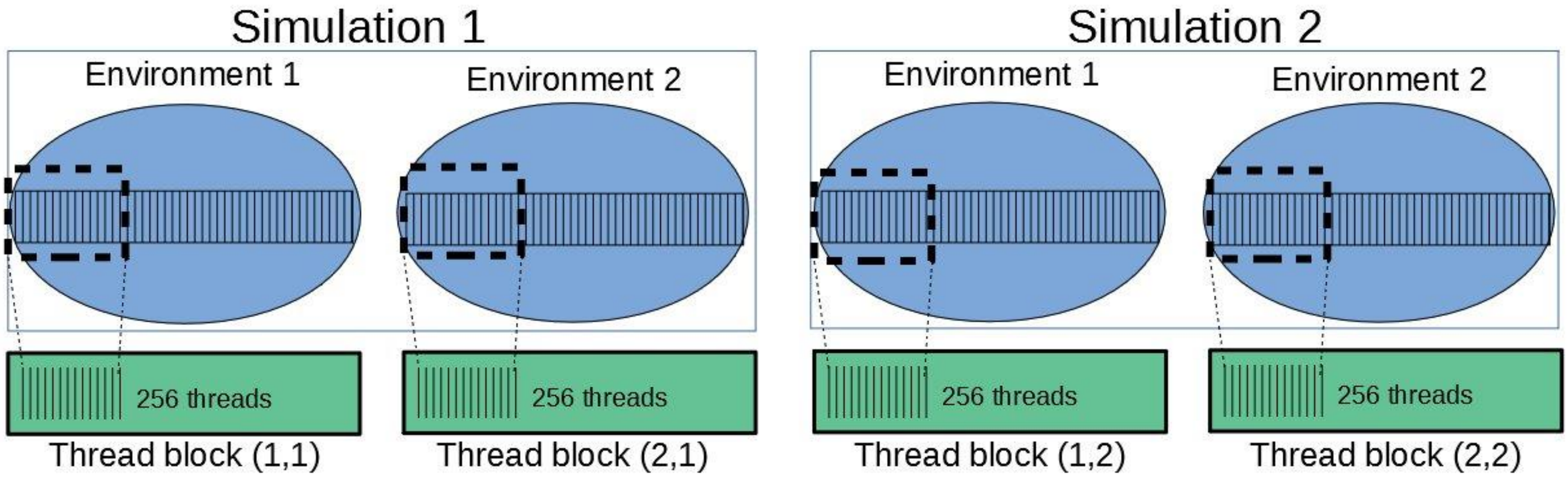
Context – Population Dynamics P Systems

- **Applications** to ecological modelling:

- Simulation algorithms: BBB, DNDP, **DCBA**.

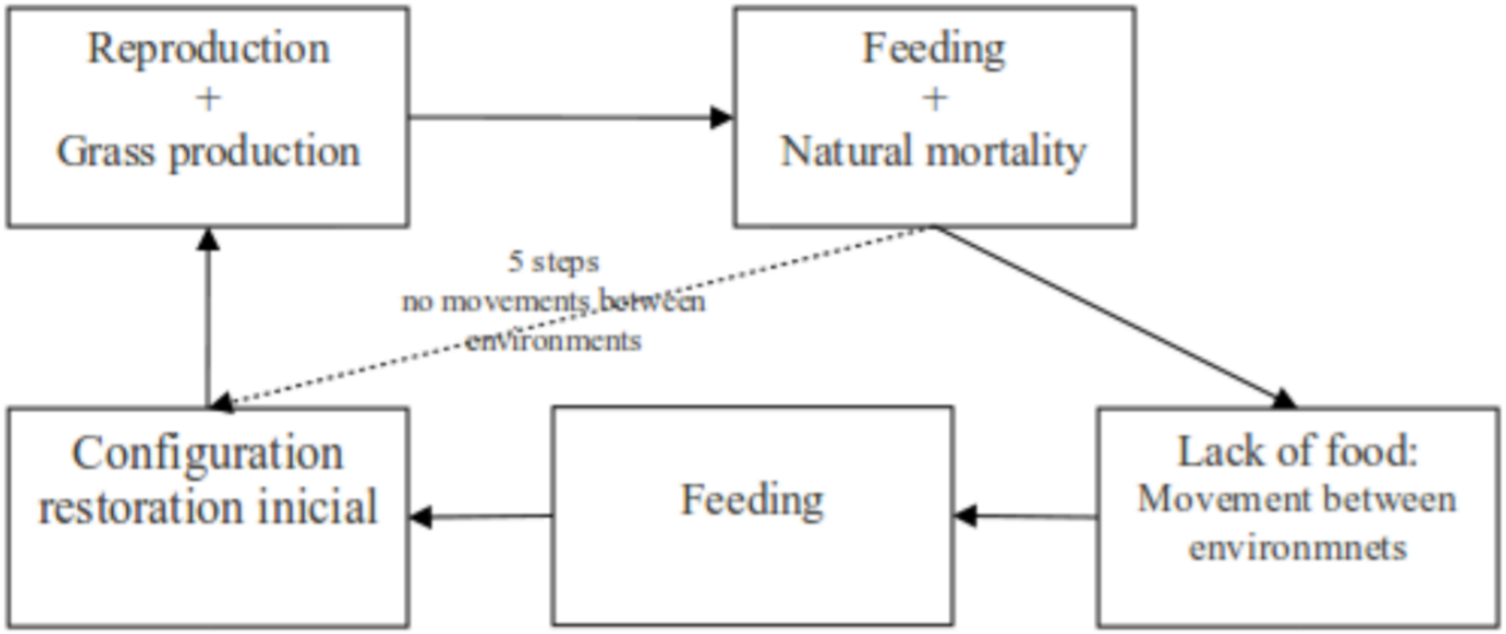
Context – GPU simulator of PDP systems

- ABCD-GPU implements **DCBA**



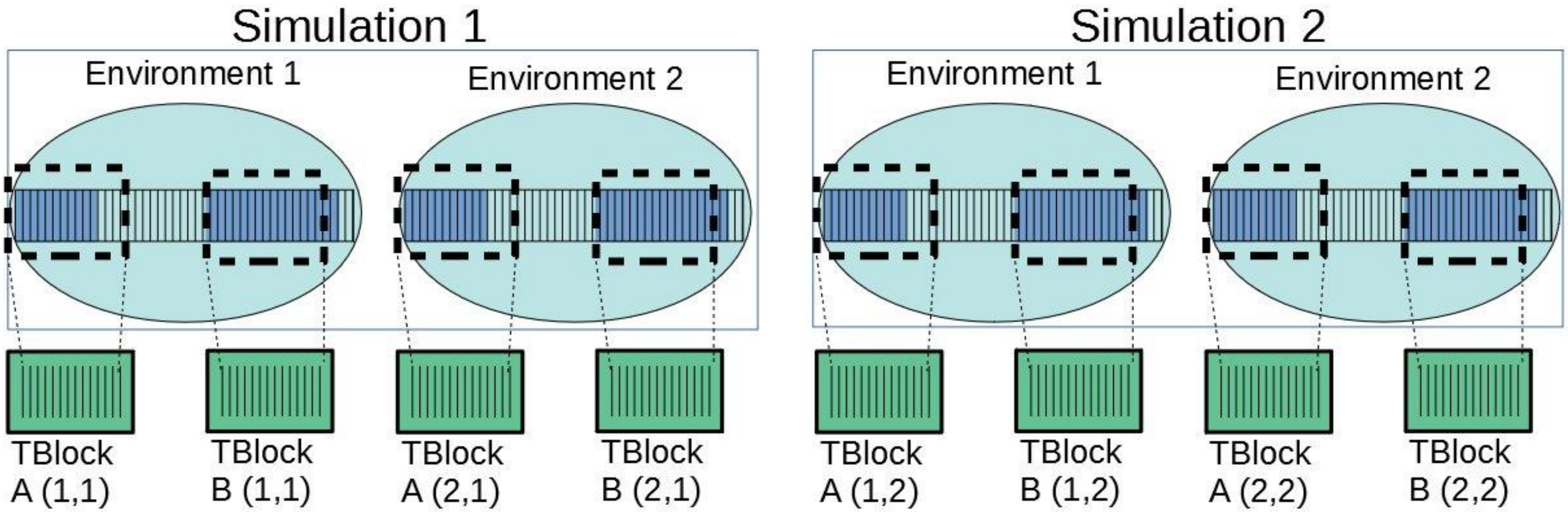
Modular GPU simulator

- Imagine the simulator knows which **module** each rule belongs to
- E.g. Modules of the tritrophic interactions model within a **cycle**



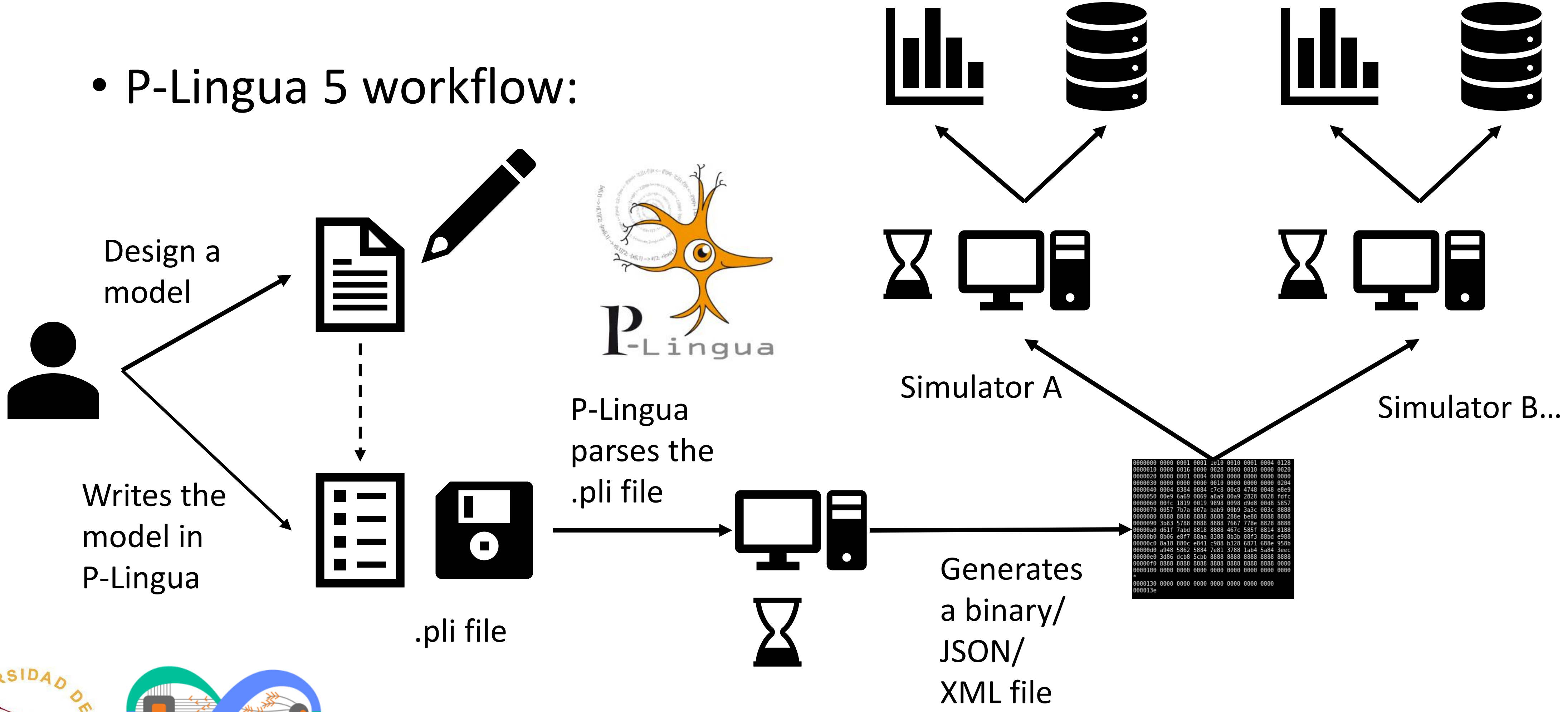
Modular GPU simulator

- Explicit parallelism



P-Lingua 5's *features*

- P-Lingua 5 workflow:



P-Lingua 5's *features*

- So far, a .pli file looks like:

```
@model<probabilistic>

def main()
{
    @mu = [ [ [ []'2 ]'1 ]'101,101 ]'global;

    @ms(2,101) = a*90, b*72, c*66, d*30;
    @ms(1,101) = a*60;
    @ms(101,101) = b;

    /* Checking accuracy */
    /*B0*/  [ a*4, b*4, c*2 ]'2 --> e*2 []'2 :: 0.7;
        [ a*4, b*4, c*2 ]'2 --> [ e*2 ]'2 :: 0.2;
        [ a*4, b*4, c*2 ]'2 --> [ e, f ]'2 :: 0.1;

    /*B1*/  [ a*4, d*1 ]'2 --> f*2 []'2 :: 1;

    /*B2*/  [ b*5, d*2 ]'2 --> g*2 []'2 :: 1;

    /* Checking filters */
    /*B3*/  b -[ a*7 ]'1 --> -[ h*100 ]'1 :: 1;

    /*B4*/  a*3 [ ]'2 --> [ e*3 ]'2 :: 1;

    /*B5*/  a, b [ ]'2 --> -[ g*3 ]'2 :: 1;
}
```

P-Lingua 5’s *features*

- In P-Lingua 5, a .pli file looks like:

```
@model<probabilistic>
@include "pdp_model.pli"

def main()
{
    @mu = [ [ [ []'2 ]'1 ]'101,101 ]'global;

    @ms(2,101) = a*90, b*72, c*66, d*30;
    @ms(1,101) = a*60;
    @ms(101,101) = b;

    /* Checking accuracy */
    /*B0*/  [ a*4, b*4, c*2 ]'2 --> e*2 []'2 :: 0.7;
            [ a*4, b*4, c*2 ]'2 --> [ e*2 ]'2 :: 0.2;
            [ a*4, b*4, c*2 ]'2 --> [ e, f ]'2 :: 0.1;

    /*B1*/  [ a*4, d*1 ]'2 --> f*2 []'2 :: 1;

    /*B2*/  [ b*5, d*2 ]'2 --> g*2 []'2 :: 1;

    /* Checking filters */
    /*B3*/  b -[ a*7 ]'1 --> -[ h*100 ]'1 :: 1;

    /*B4*/  a*3 [ ]'2 --> [ e*3 ]'2 :: 1;

    /*B5*/  a, b [ ]'2 --> -[ g*3 ]'2 :: 1;
}
```

- Includes a .pli for rule syntax:

```
!skeleton_rule
{
    u1 ?[v1]'h -> u2 ?[v2]'h :: probability(double_t);
}

!communication_rule
{
    [[a]'e1 [ ]'e2 ]'p -> [[ ]'e1 [b]'e2 ]'p :: probability(double_t);
    [[a]'e ]'p -> [[b]'e ]'p :: probability(double_t);
    [[a]'e ]'p -> [[ ]'e ]'p :: probability(double_t);
    [[a]'e1 ]'p -> [[b]'e2 ]'p :: probability(double_t);
}

@model(probabilistic) = skeleton_rule,communication_rule;
```

P-Lingua 5's *features*

- In most programming languages we can find the concept of directive

Sequential

```
void main() {
  double a[1000],b[1000],c[1000];
  for (int i = 0; i< 1000; i++){
      a[i] = b[iI] + c[i];
  }
}
```

Parallel with OpenMP

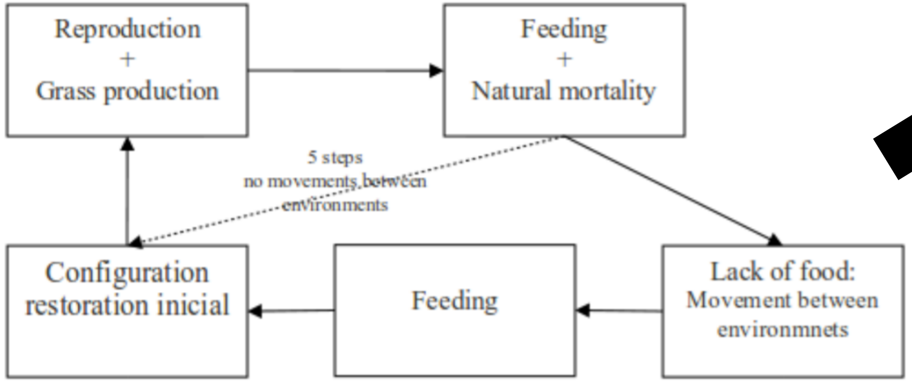
```
void main() {
  double a[1000],b[1000],c[1000];
  #pragma omp parallel for
  for (int i = 0; i< 1000; i++){
      a[i] = b[iI] + c[i];
  }
}
```

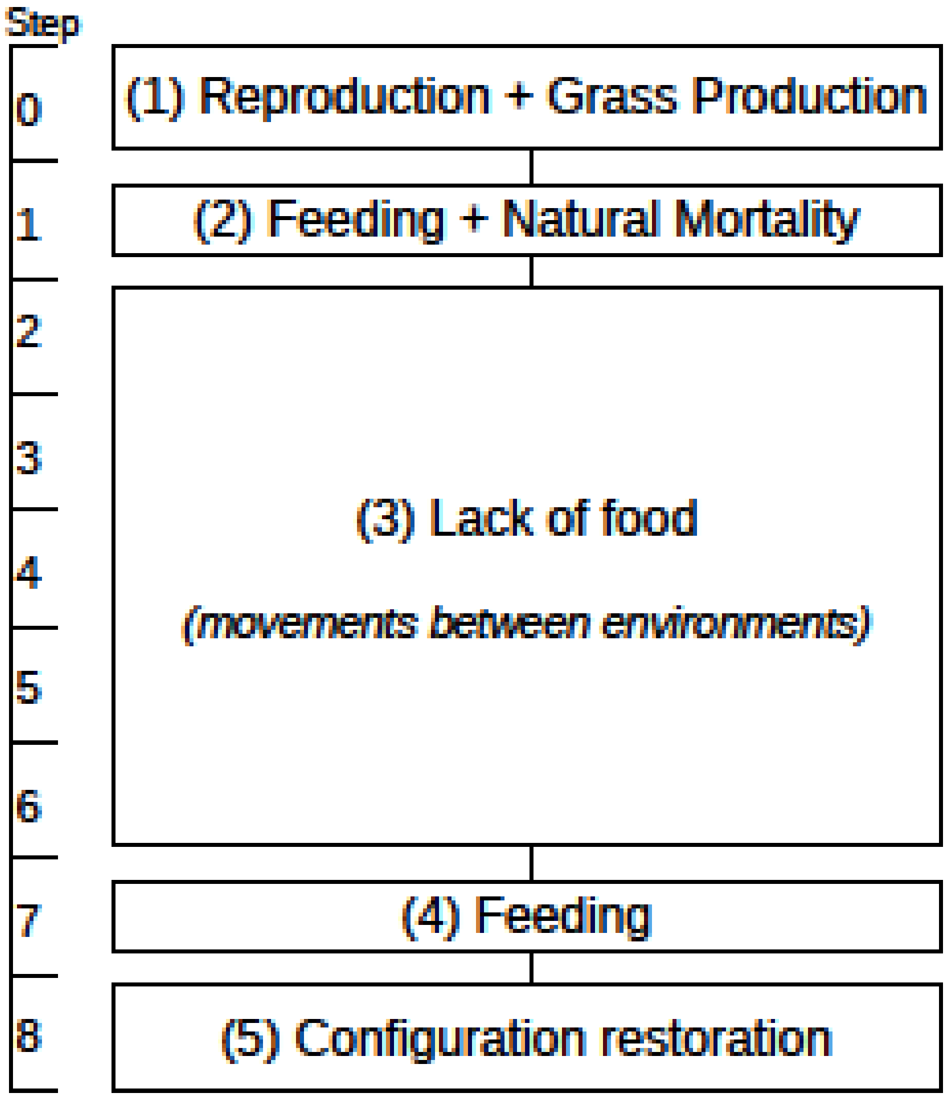
P-Lingua 5's *features*

- A **feature** in P-Lingua 5 is a piece of high-level information provided by the P system designer
- Syntax: **`@name=value;`** (value can be Integer or Character String)
- Two types:
  - **Global**: at the P system level
  - **Local**: at the rule level

P-Lingua 5's *features*

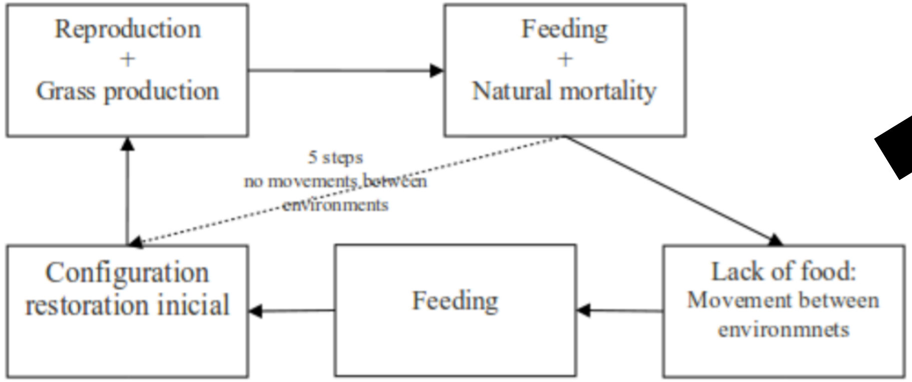
- Example of **tritrophic interactions**:

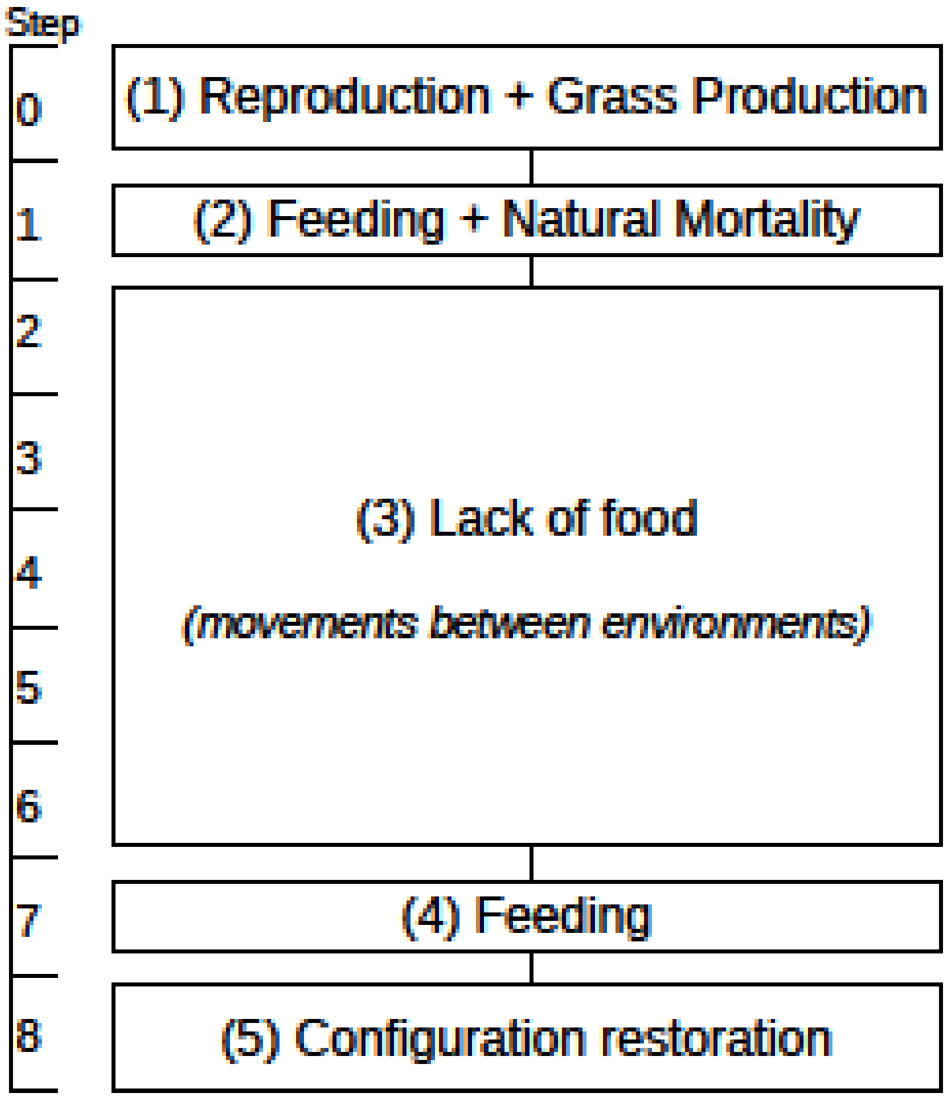




P-Lingua 5’s *features*

- Example of **tritrophic interactions**:

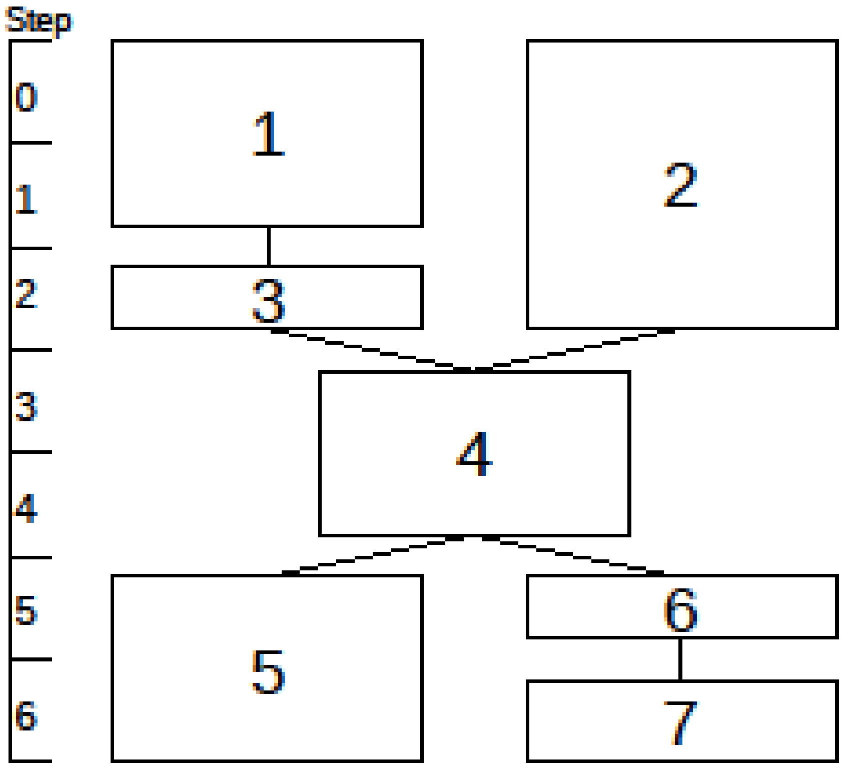




Global: `@modules="(1,1,{2});(2,1,{3});(3,5,{4});(4,1,{5});(5,1,{})";`

Local: `-[R{i}]’1 --> -[R{i+1}]'1 :: 1 @module="3": 0<=i<=4;`

P-Lingua 5's *features*



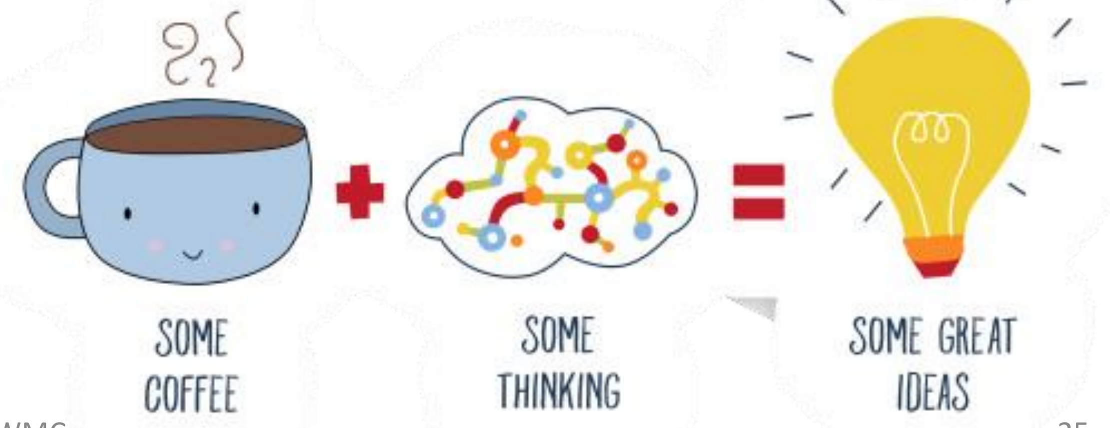
- Another example:
  - `@modules= "(1,2,{3});(2,3,{4});(3,1,{4});(4,2,{5,6});(5,2,{}); (6,1,{7});(7,1,{})";`
- Features are **transparent** to simulators:
  - They are fetched explicitly, so only simulators supporting them will take advantage

P-Lingua 5's *features*

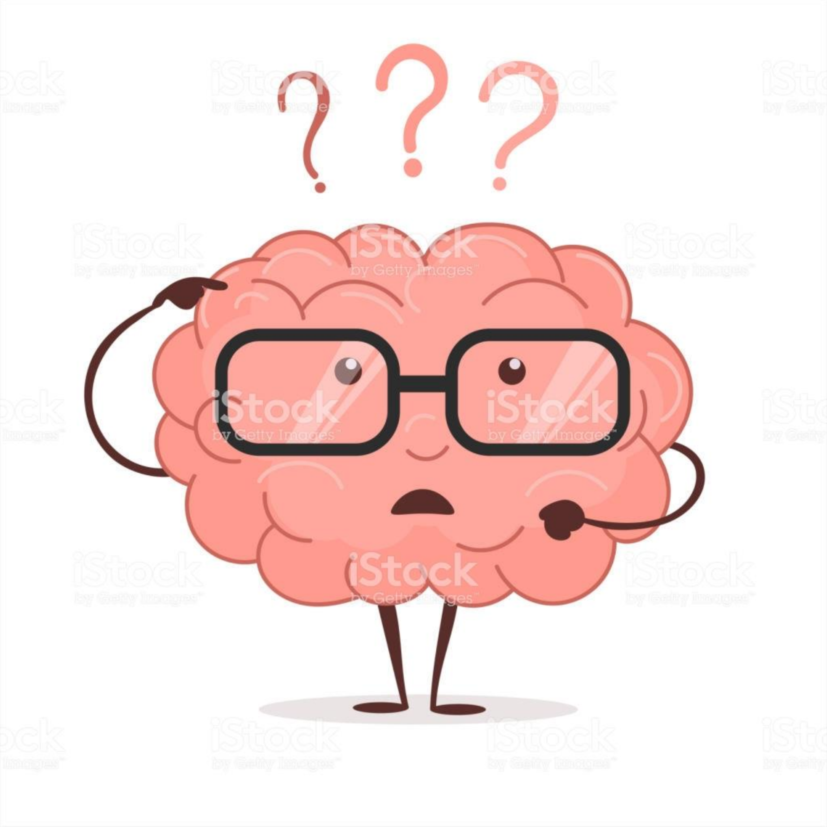
- **Results** with modular GPU simulator of PDP systems on tritrophic interactions:
  - **Tested** on:
    - GPU Tesla K40: 2880 cores, old generation
    - GPU GTX1050Ti: 768 cores, newer generation
    - CPU Intel Xeon: 8 cores
  - Peak **speedup**:
    - **K40**: up to **2.5x**
    - **GTX1050**: up to **1.7x**
    - **CPU**: up to **2.7x**
  - For small models (e.g. just 7 species) there is almost no improvement

Provocative ideas

- What else?
  - Apply to **other solutions** (SAT, FACTORIZATION, SUBSET SUM, …)
    - Stages = modules
  - How to use modules with other classes of P systems?
    - Spiking Neural P systems?
  - Use features to **mark objects**, indicating they are counters
  - Disabling membranes for some steps
  - …..



Thank you very much



- This work was supported by the research project TIN2017-89842-P, co-financed by Ministerio de Economía, Industria y Competitividad (MINECO) of Spain, through the Agencia Estatal de Investigacion (AEI), and by Fondo Europeo de Desarrollo Regional (FEDER) of the European Union. The authors also acknowledge the donation of a Tesla K40 by the NVIDIA CUDA Research Center program in 2014.