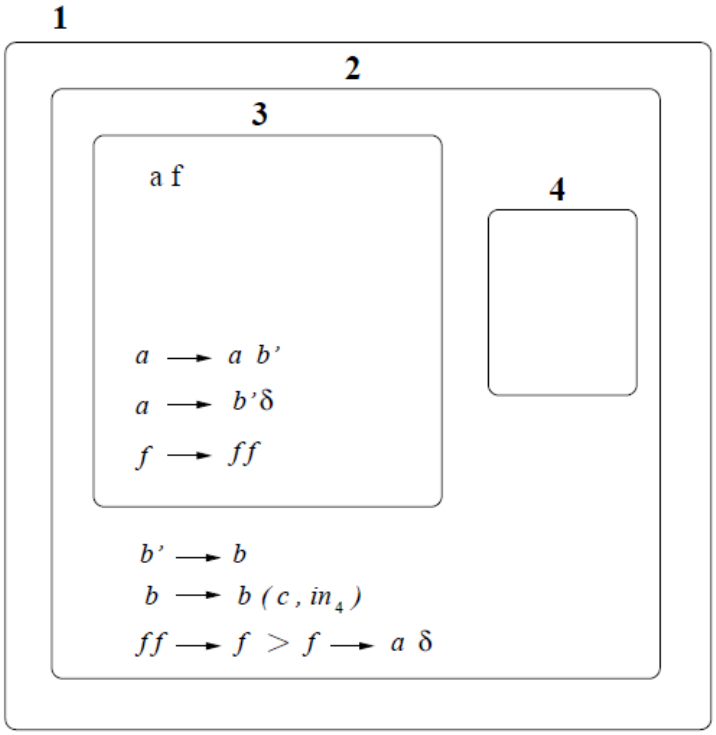
General-Purpose GPU accelerated simulation of membrane systems

Manuel García-Quismondo Fernández

Index

- A brief introduction to Membrane Computing
- A brief Introduction to GP-GPU
- Case study: The SAT Problem
- Technology and design
- Performance results
- Conclusions and future work

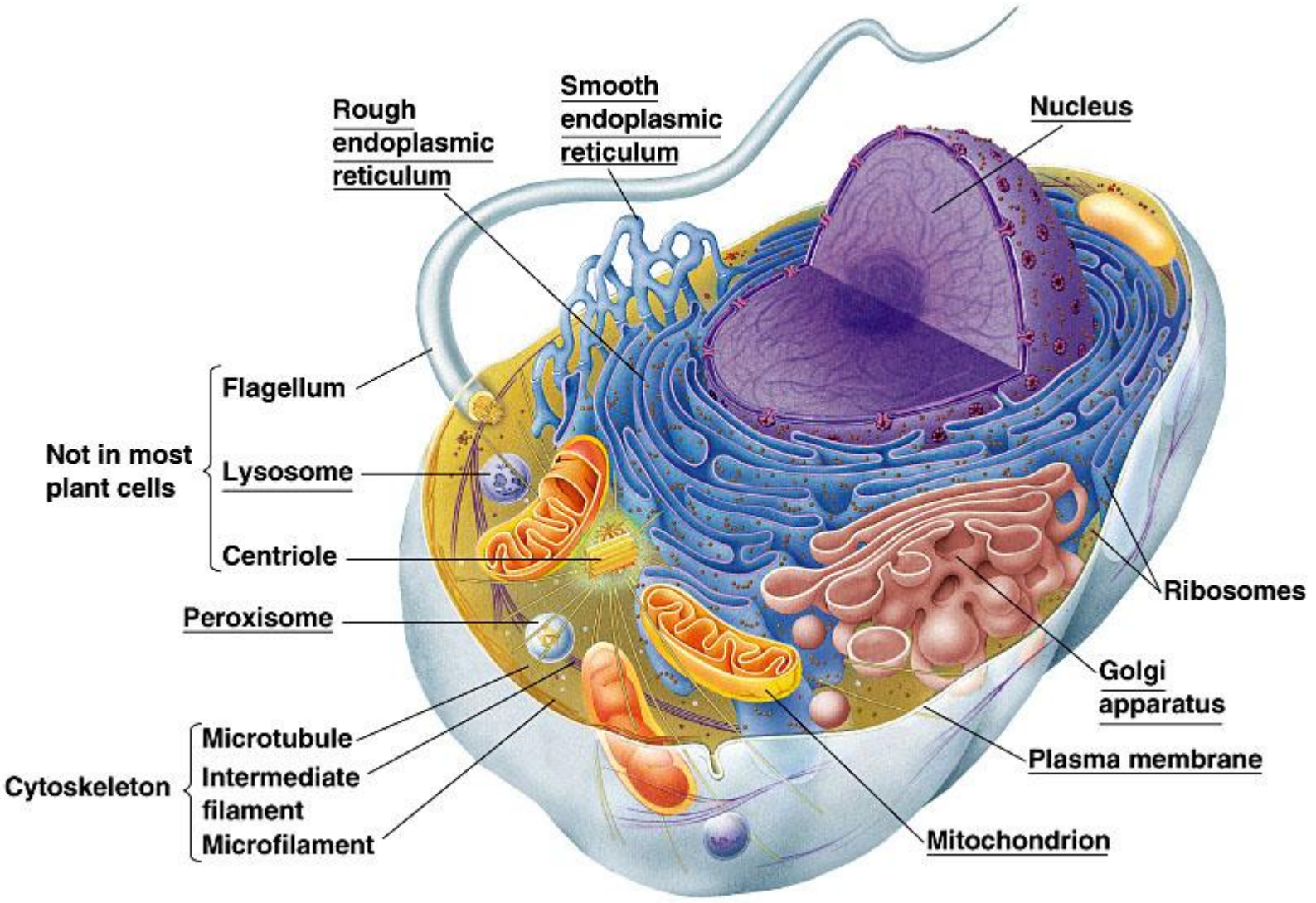
A brief introduction to Membrane Computing



What is Membrane Computing?

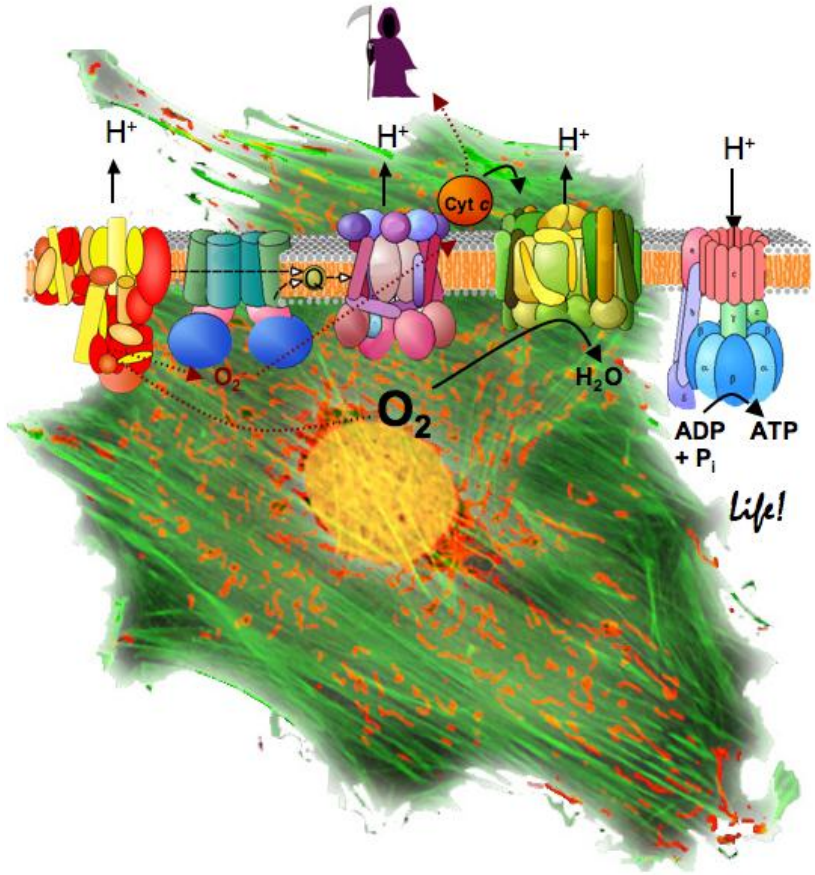
A branch of Natural Computing inspired by the structure and functioning of the living cell [1]

[1] Gh. Paun. *Membrane Computing. An introduction*, Springer-Verlag, 2002, XI+419p.



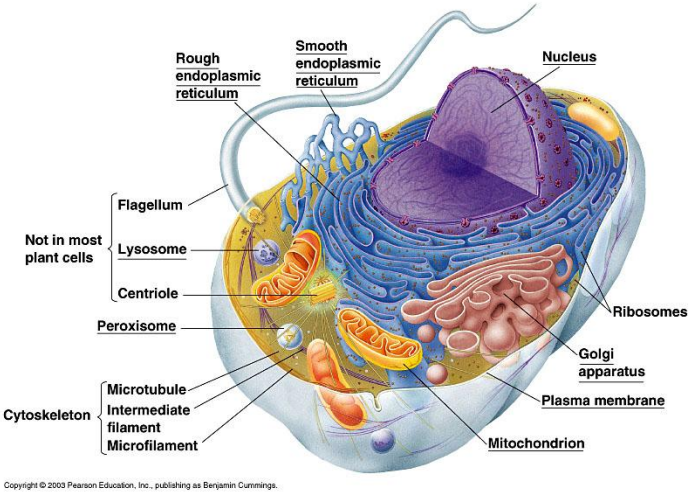
Bio-chemical processes

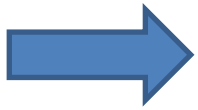
ocurr within living cells



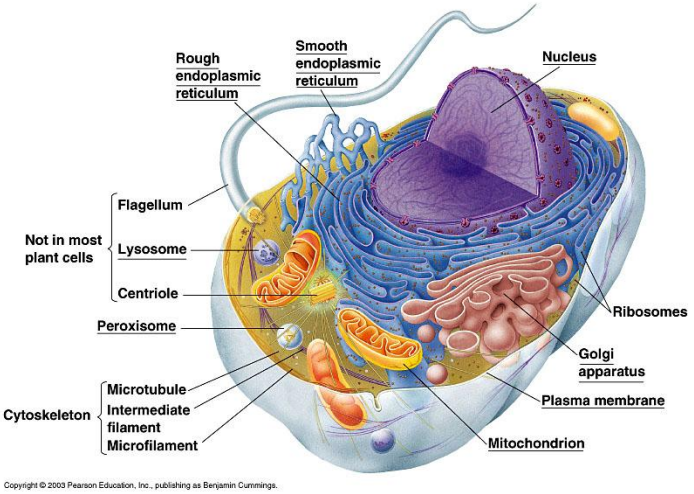
Main Idea: Bio-chemical processes can be interpreted as ***calculus procedures***

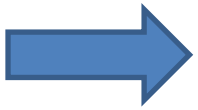
Look at cells as ***computing devices***

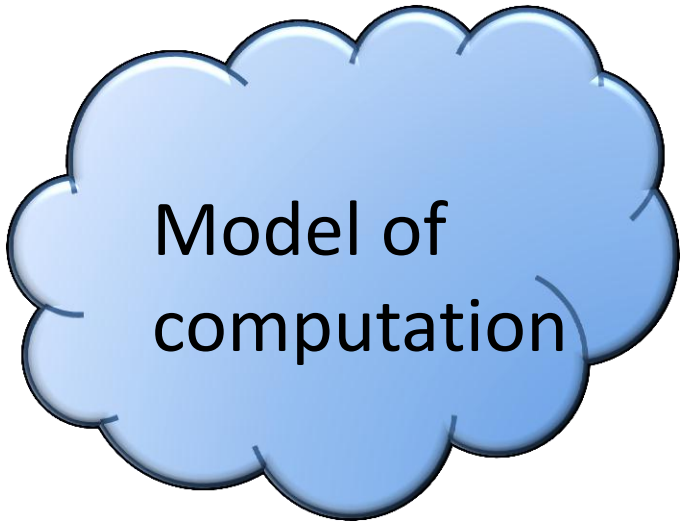




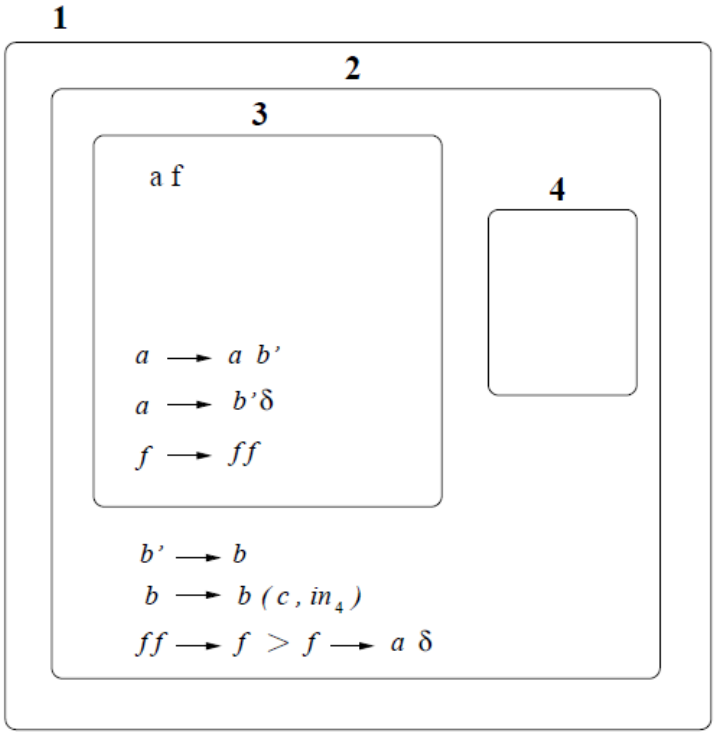
Model of *cells* as *computing devices*







Theoretical devices: P Systems or ***Membrane Systems***



Membrane Computing model: ***syntax*** and ***semantics***

These elements determine the ***functioning*** of the Membrane Systems defined within the model

Semantical ***key features***

- Non-Determinism
- Maximal **Parallelism**

Key definitions

- Configuration
- Transition step
- Halting configuration
- Computation

Now, we introduce the concept of
recognizer P system

- $\forall$ Computation $\Rightarrow$ ***halts***
- ***{Yes, No}*** $\subseteq \Gamma$
- ***Accepting*** or ***rejecting*** computation

A special kind of ***recognizer P systems*** are ***confluent recognizer P systems***

- One of two possibilities
  - Every computation is an accepting one
  - Every computation is a rejecting one

A brief introduction to *GP-GPU*

P Systems are ***not*** yet
implemented

… which does **not** mean
that they will not be
implemented in the future

So far, all we can do is ***simulate*** them by using ***different devices***

We can use **PCs**

We can also use **parallel devices**

… it seems appropriate, as Membrane Computing models are maximally **parallel**


Computer *clusters*

FPGAs

Microcontrollers

Graphic cards

Graphic cards (GPUS) are designed for **graphics processing**

Graphics processing are usually
parallel problems

Hence, GPUs are

parallel devices

However, solving general-purpose problems with GPUs is ***not*** straightforward

We need ***techniques*** to adapt general-purpose problems to be solved by GPUs

These techniques are studied by a discipline named ***GP-GPU***

Case study: the ***SAT problem***

Key concepts

- Literal
- Clause
- Propositional formula in CNF

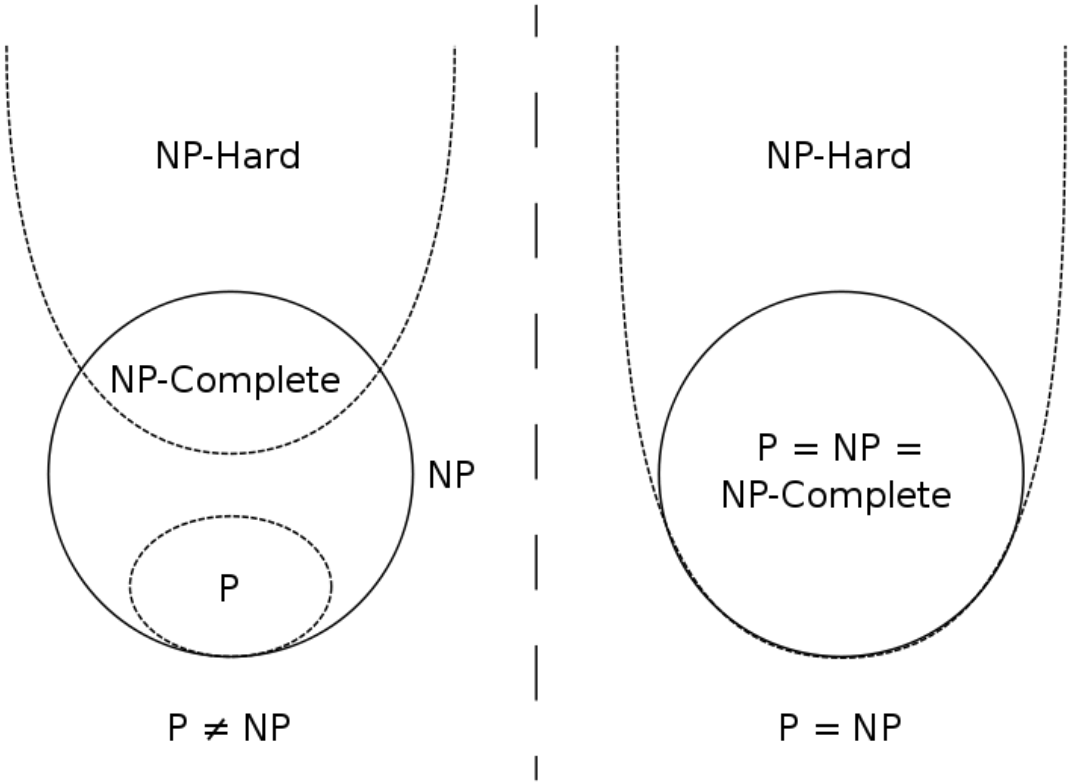
Key concepts

- Satisfiability of a propositional formula

Is this formula ***satisfiable***?

Does there exist any combination of values which makes the formula ***true***?

NP-Complete problem



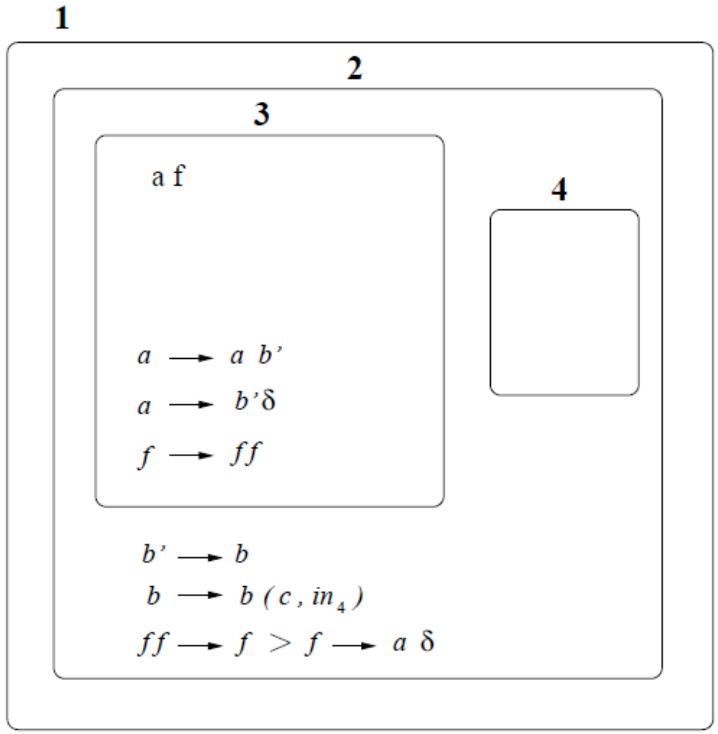
Efficient solution with P systems

$(X1 \text{ or } X2 \text{ or } \overline{X3})$

$(\overline{X1} \text{ or } \overline{X2} \text{ or } X3)$

$(\overline{X1} \text{ or } \overline{X2} \text{ or } \overline{X3})$

$(\overline{X1} \text{ or } X2 \text{ or } X3)$



P system with active membranes

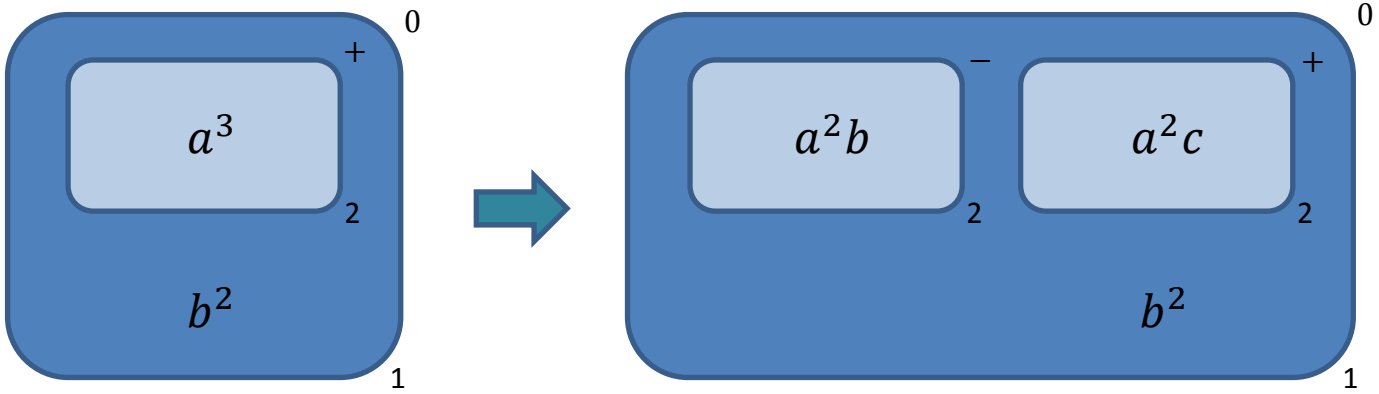
- Electrical charges
- Division rules
- No cooperation

Division rules

- Membrane ***duplication***
- Object ***replication***

Division rules

$$[a]_2^+ \rightarrow [b]_2^-[c]_2^+$$



Definition of PMC_R

Problems which can be solved by P systems in ***Polynomially bounded time***

A ***family*** of P systems with active membranes will solve the SAT problem

- All computations of a P system return ***Yes*** iff the formula is ***satisfiable***
- All computations of a P system return ***No*** iff the formula is ***not satisfiable***

Codification of a formula

Given ***m*** clauses and ***n*** variables

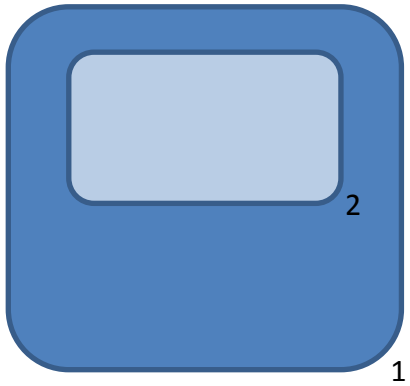
Codification of a formula

x_{ij}: Variable ***j*** appears on clause ***i*** in ***affirmative*** form

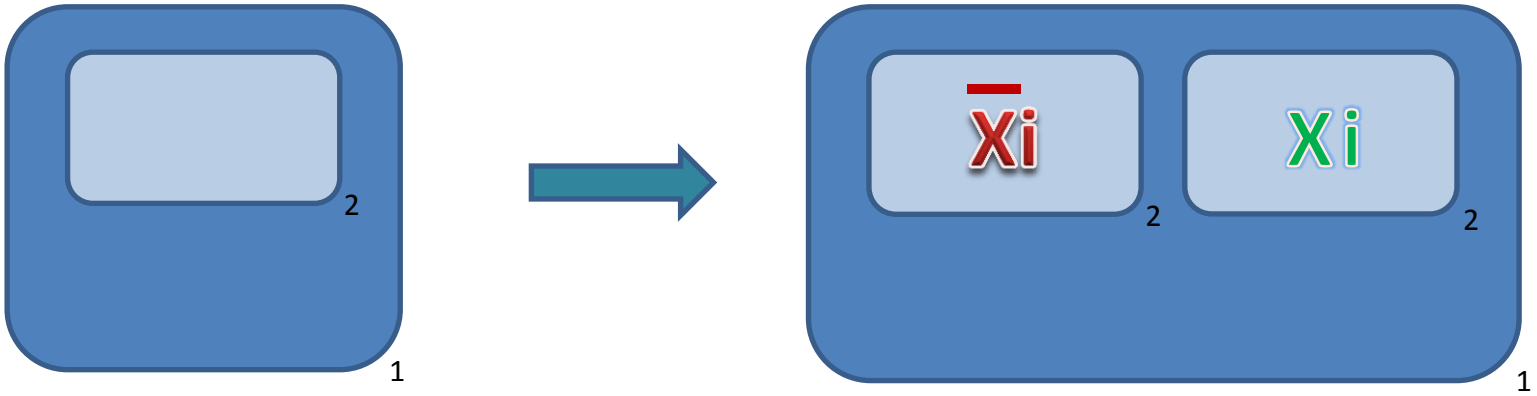
$\bar{x}_{ij}$: Variable ***j*** appears on clause ***i*** in ***negative*** form

Then, we use the following ***codification***

All P systems have a two-level ***initial membrane structure*** (μ_0)

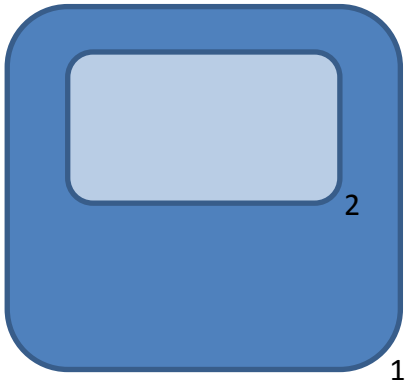


The idea is to use membrane division to create possible ***scenarios*** (combinations of values)



Membranes have associated **multisets**

Hence, we have a ***three-leveled parallel*** P system family



+ M_2

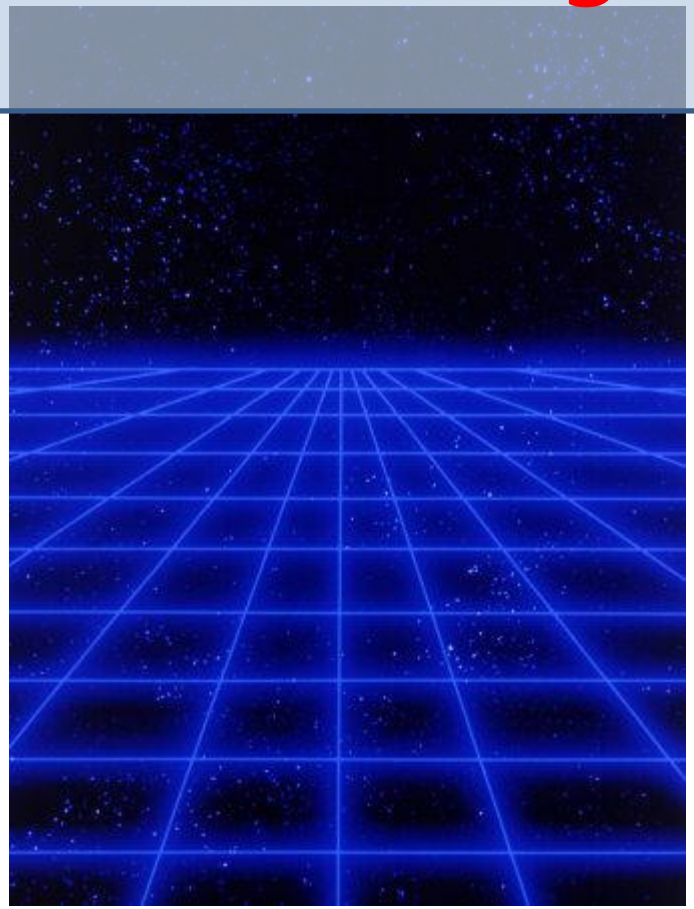
Technology and design

For our simulator, we use a language named ***CUDA***

This language is specifically designed for GP-GPU

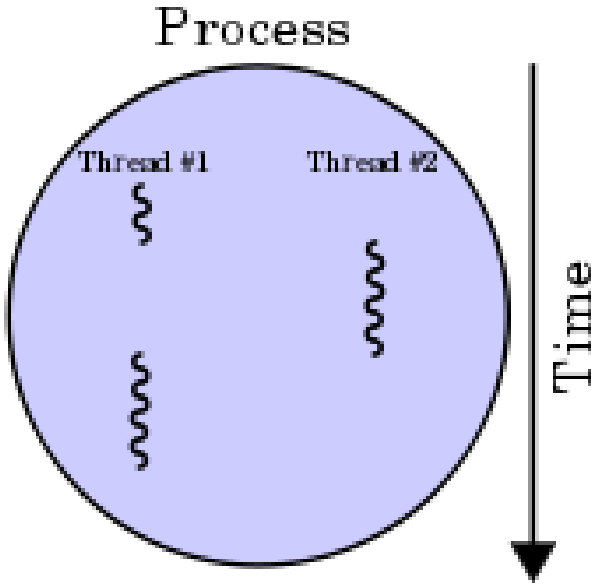
CUDA describes a programming ***model***

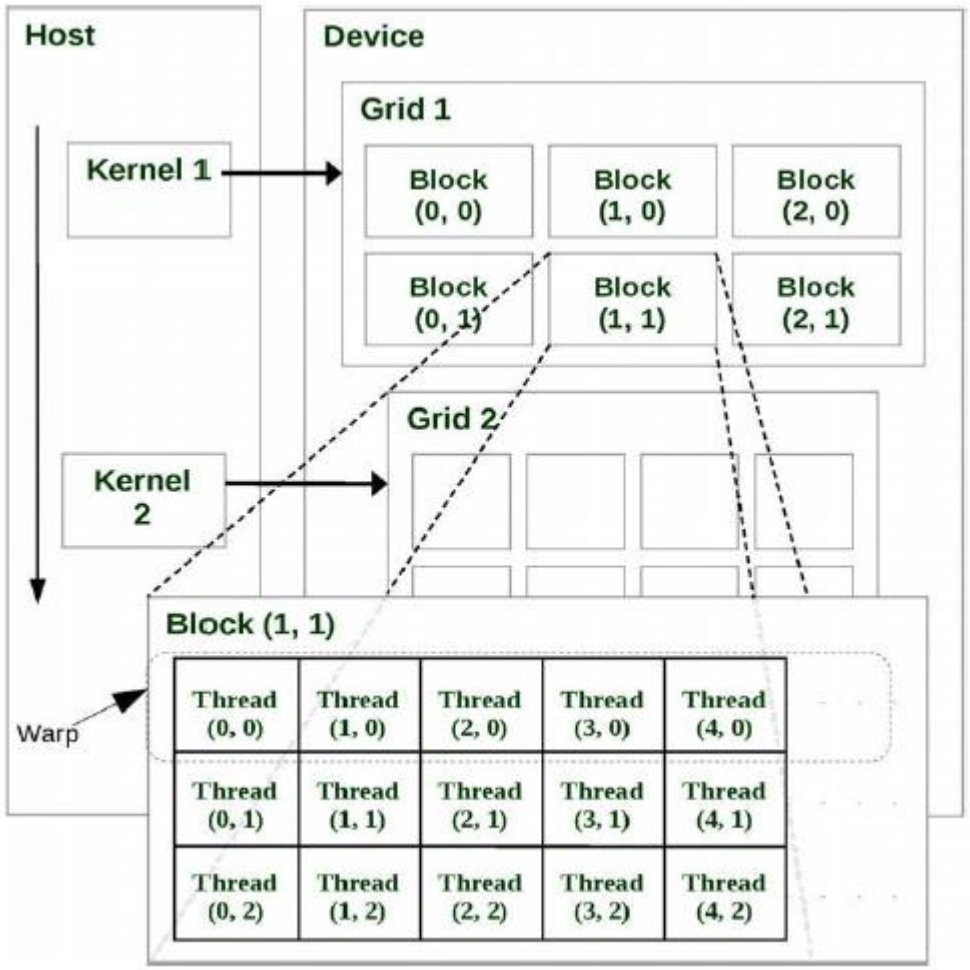
Basically, the model describes a ***grid***



This grid is composed of ***blocks***

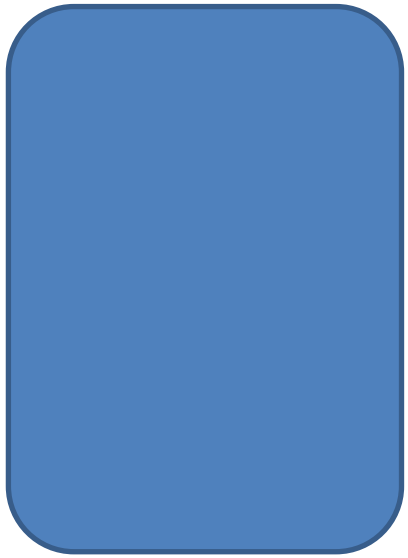
Each block is composed of ***threads***



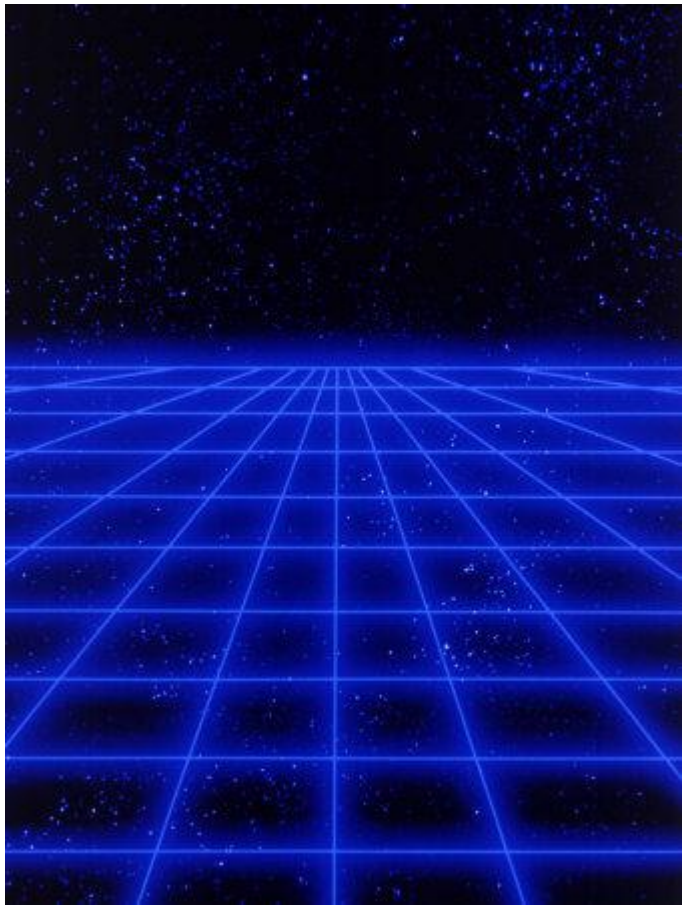

Host
Device
Grid 1
Kernel 1
Block (0, 0)
Block (1, 0)
Block (2, 0)
Block (0, 1)
Block (1, 1)
Block (2, 1)
Grid 2
Kernel 2
Block (1, 1)
Warp
Thread (0, 0)
Thread (1, 0)
Thread (2, 0)
Thread (3, 0)
Thread (4, 0)
Thread (0, 1)
Thread (1, 1)
Thread (2, 1)
Thread (3, 1)
Thread (4, 1)
Thread (0, 2)
Thread (1, 2)
Thread (2, 2)
Thread (3, 2)
Thread (4, 2)

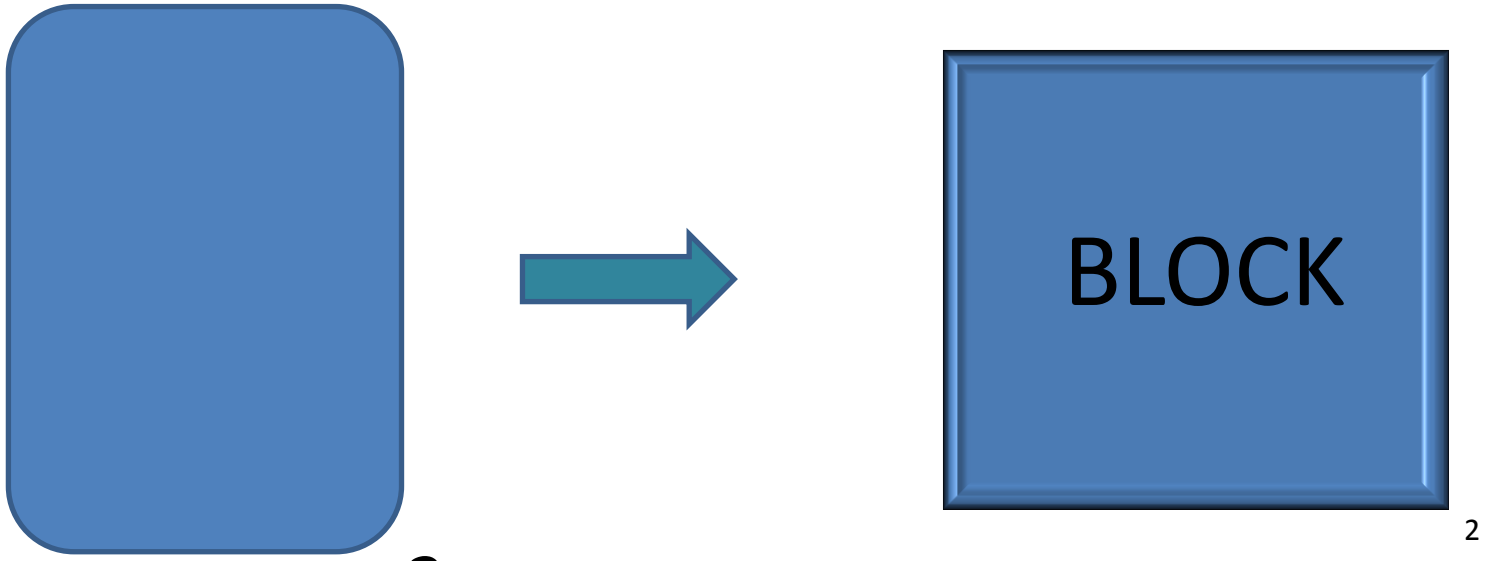
Thus, we have a ***three-level parallel model***, just like our P system family

From this point of view, we can ***assign responsibilities***

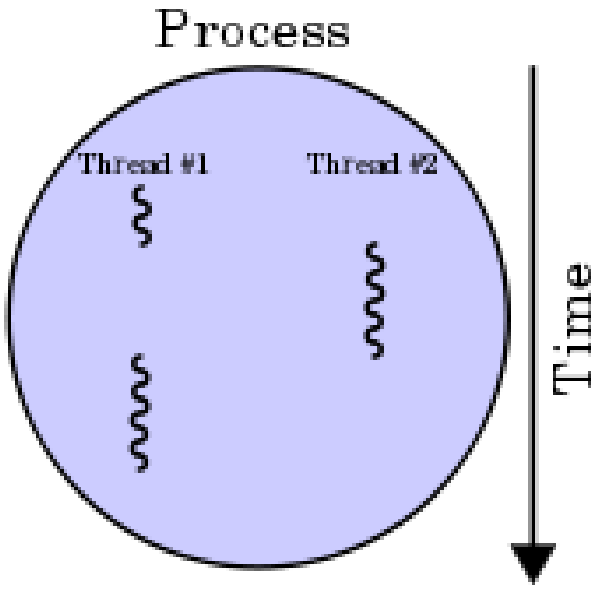


1



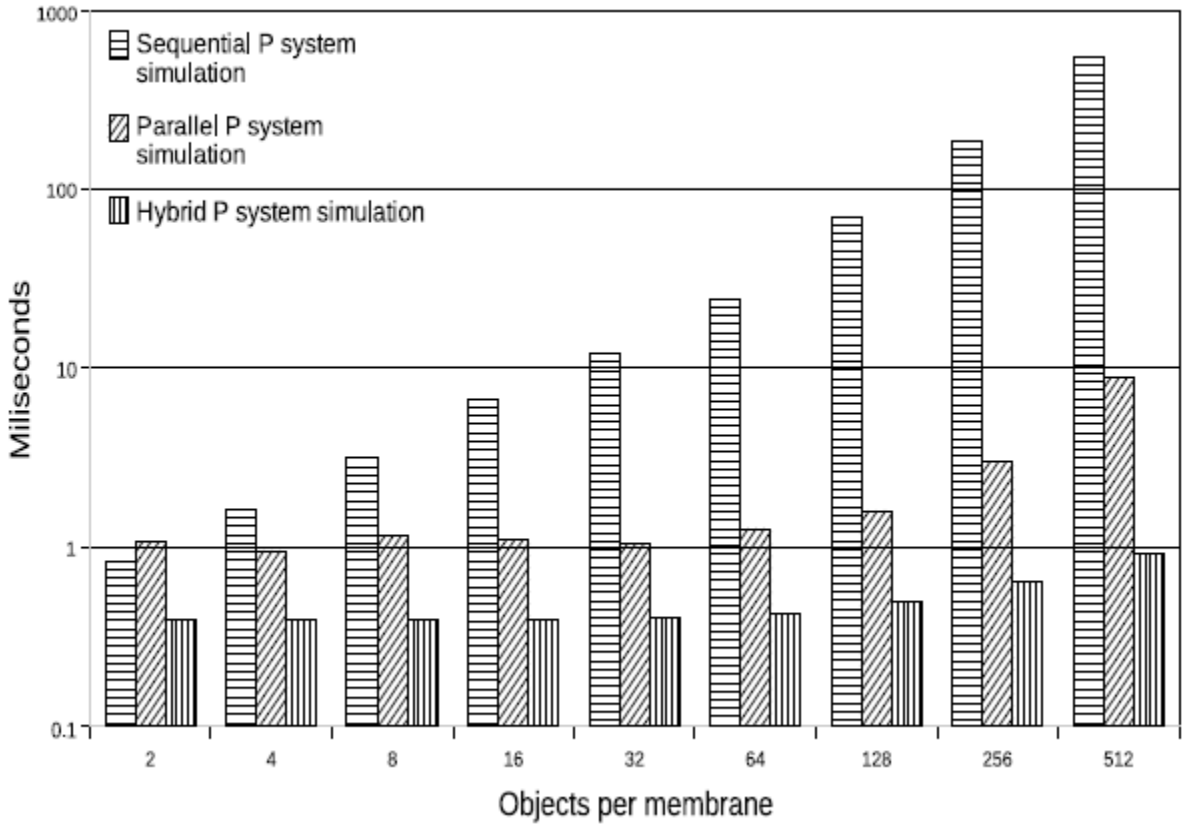

BLOCK
2
2

M_2

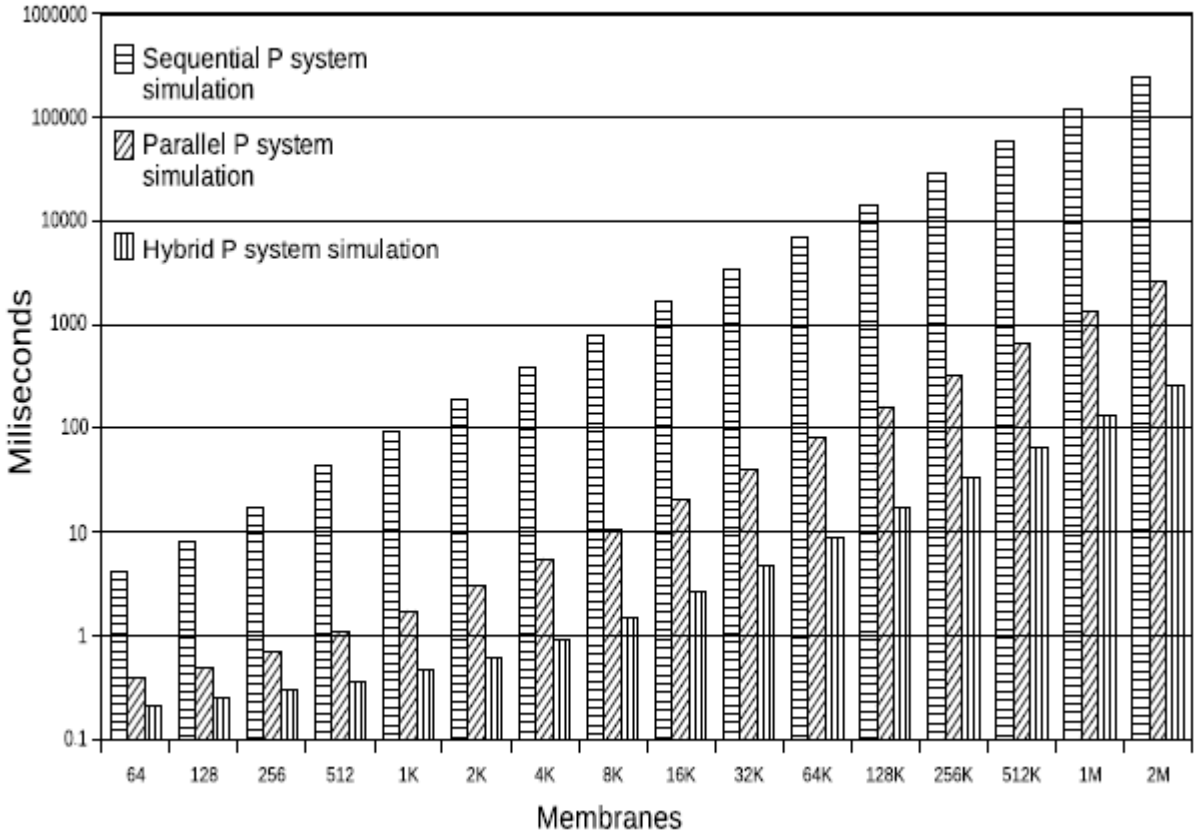


Performance results

Objects per membrane vs *miliseconds*



Membranes vs *miliseconds*



Conclusions and future work

GP-GPU is a ***promising option*** when simulating P systems

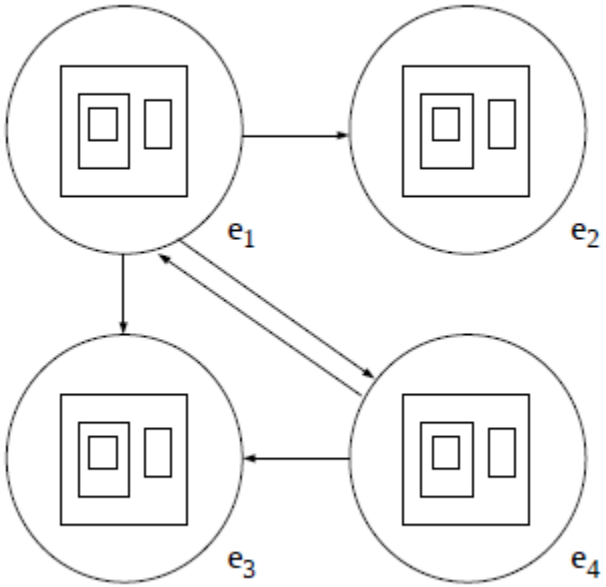
Execution times are

dramatically trimmed

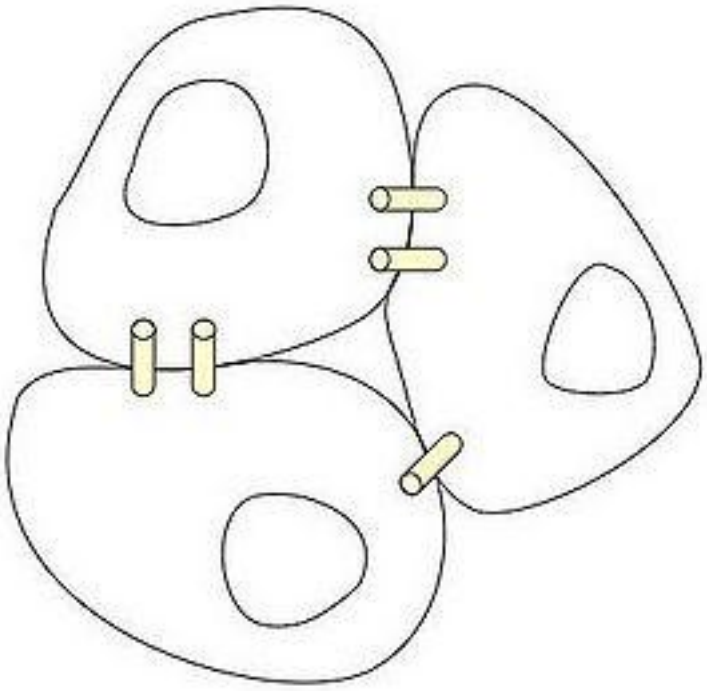
We propose several

future works

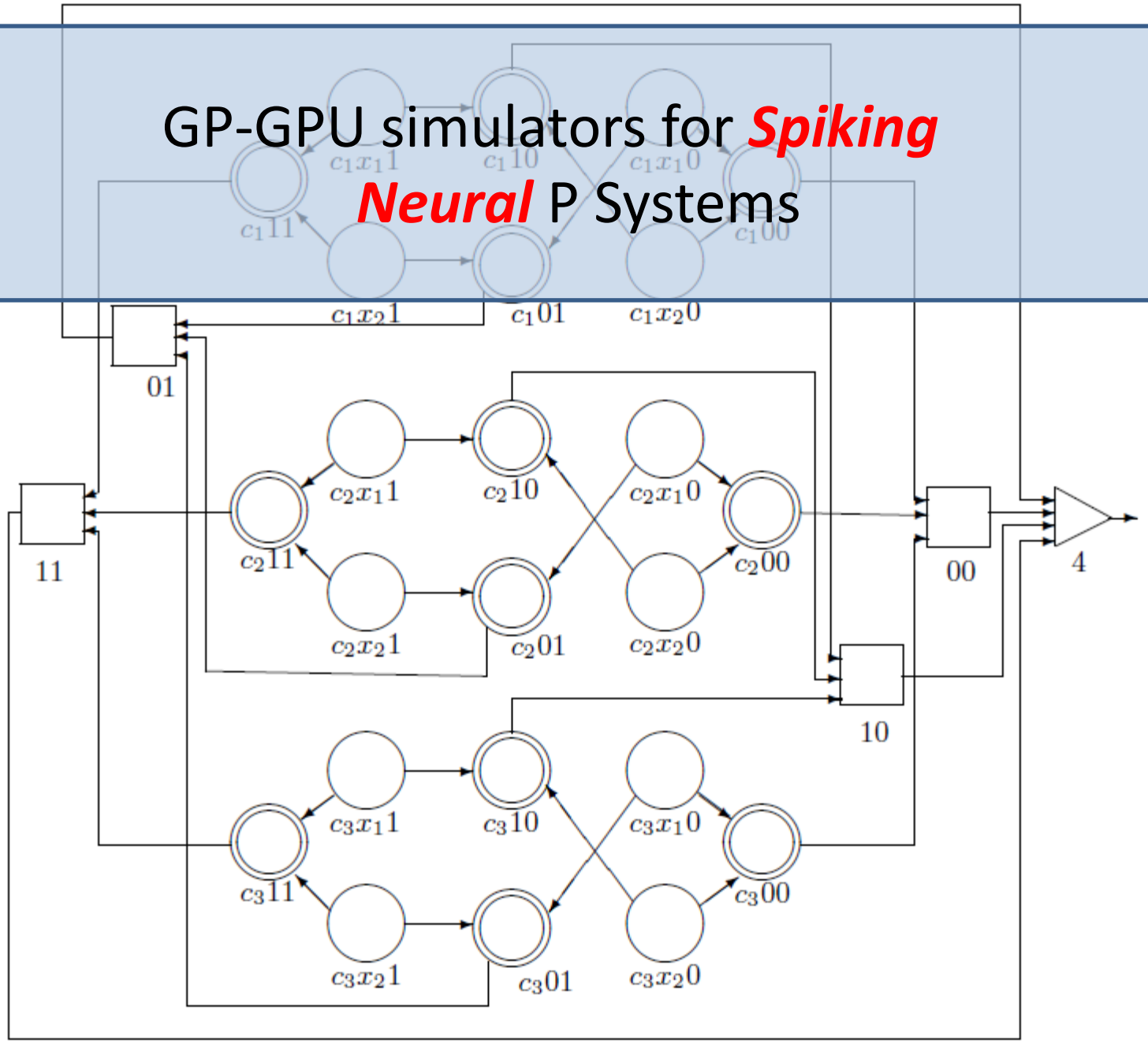
GP-GPU simulators for **Multienvironmental Probabilistic** P Systems



GP-GPU simulators for **Tissue** P Systems



GP-GPU simulators for ***Spiking Neural*** P Systems




Thank You!