Membrane Computing

BIC 2007
2nd ISCV Thematic Workshop: Biologically-Inspired Computing
December 3-7, 2007

Fernando Sancho Caparrini
Universidad de Sevilla

Outline

- Bio-inspired…
  - Living cells
  - Membranes: structure and function
  - Transport
- …Computing
  - Basics
  - Examples
  - Some results
  - Some variants
    - Spiking Neurons P systems
- Conclusions
- References

Bio-inspired…

Living Cells

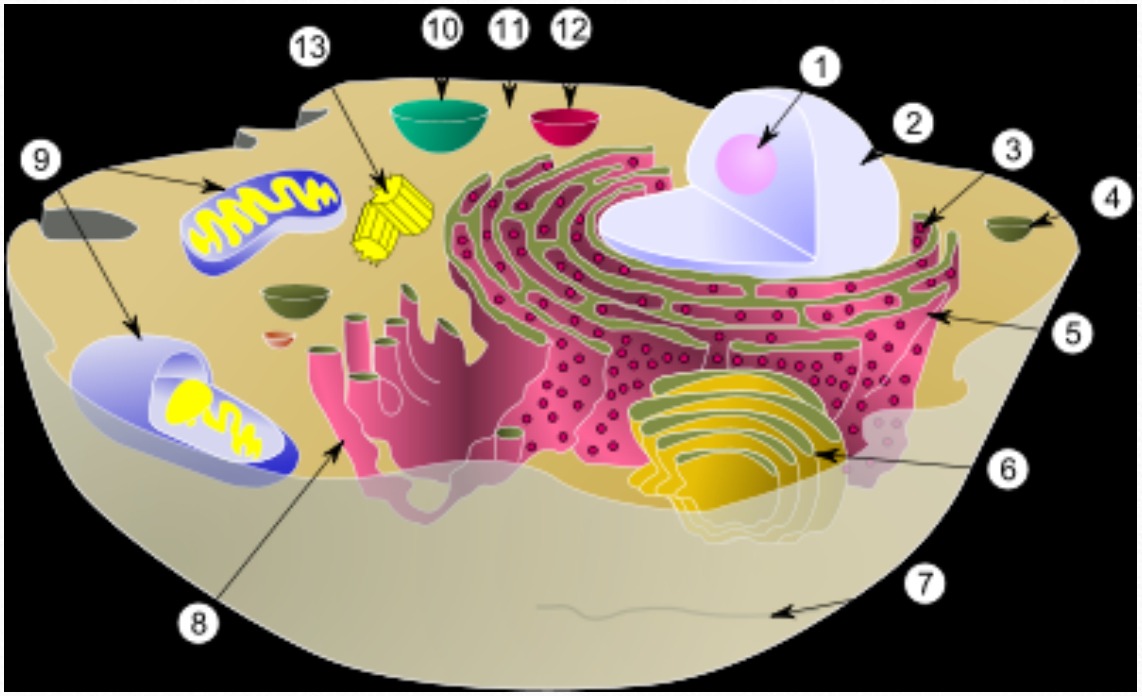
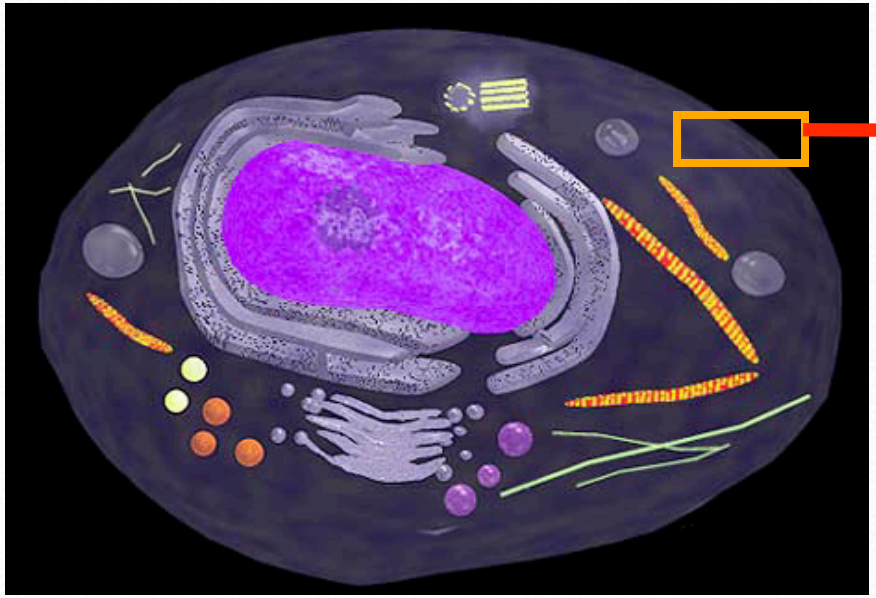
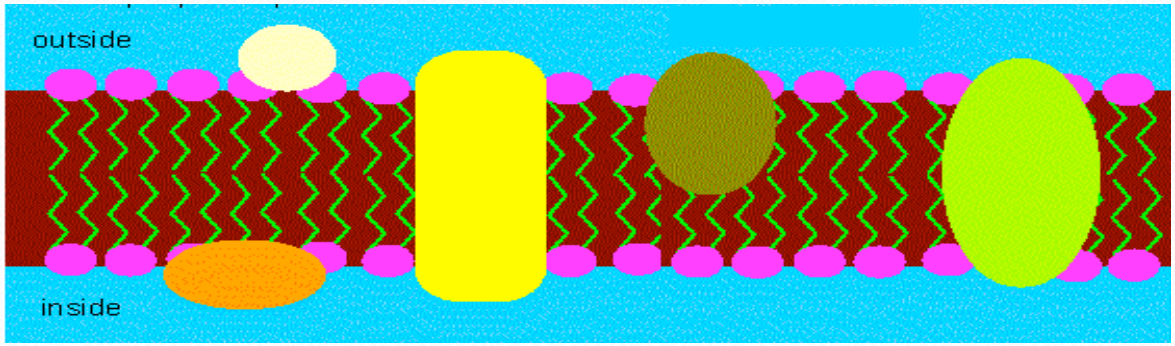


Diagram of a typical eukaryotic cell, showing subcellular components. Organelles:
(1) nucleolus (2) nucleus (3) ribosome (4) vesicle (5) rough endoplasmic reticulum (ER)
(6) Golgi apparatus (7) Cytoskeleton (8) smooth ER (9) mitochondria (10) vacuole
(11) cytoplasm (12) lysosome (13) centrioles within centrosome

http://en.wikipedia.org/wiki/Cell_(biology)

Cell membranes





Cell membranes

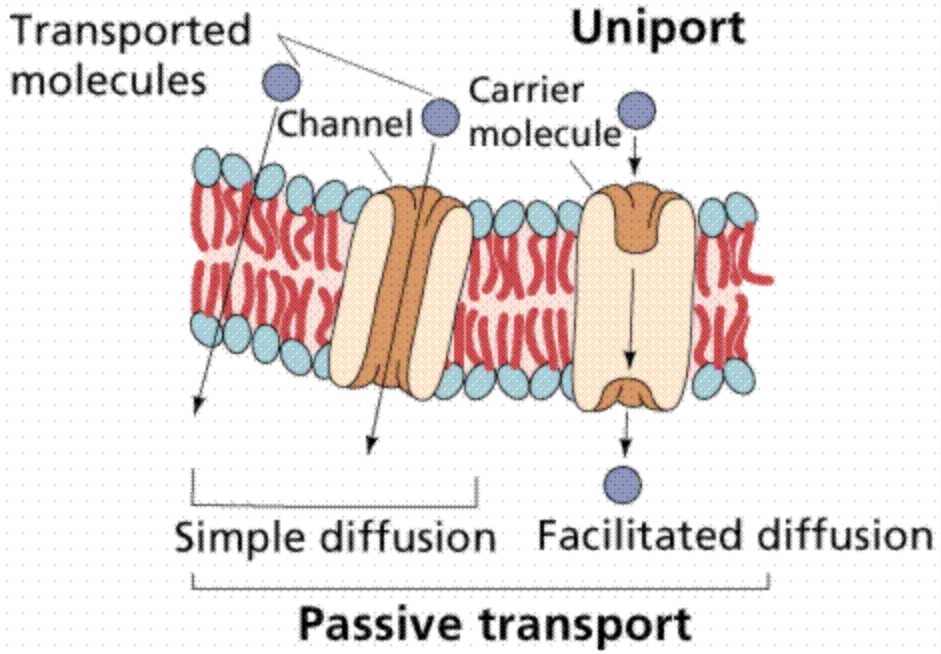
- Cell membranes are selective barriers that separate individual cells from environment and cellular compartments.
- Some functionalities:
  - regulate the transport of molecules,
  - control information flow between cells,
  - generate signals to alter cell behavior,
  - contain molecules helping the formation of tissues,
  - …
- Cell membranes are dynamic, constantly being formed and degraded.

Membrane Transport

Three main types of transport:

- **Passive Transport**: requires no energy from the cell.
- **Active Transport**: requires the cell to spend energy.
- **Vesicle Transport**: vesicles that fuse with the cell membrane may be utilized to release or transport chemicals out of the cell or to allow them to enter a cell.

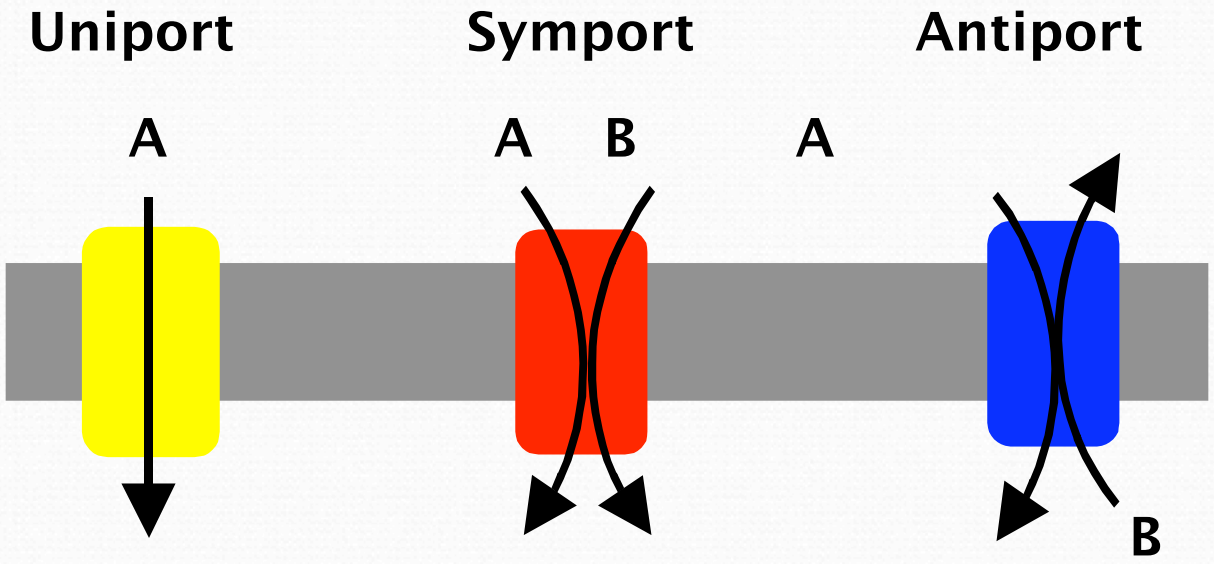
Passive Transport



Examples include the diffusion of oxygen and carbon dioxide, osmosis of water, and facilitated diffusion.

Image from Purves et al., Life: The Science of Biology, 4th Edition, by Sinauer Associates (www.sinauer.com) and WH Freeman (www.whfreeman.com)

Active Transport

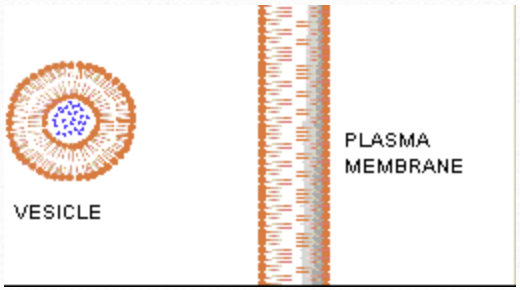


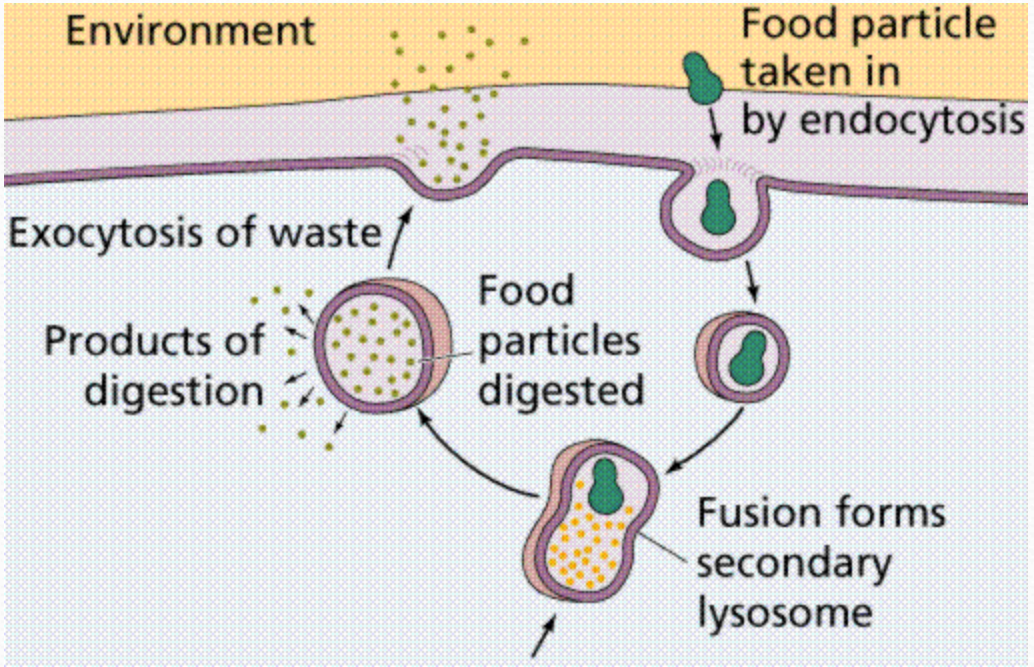
Uniport (facilitated diffusion): carriers mediate transport of a single component.

Symport (cotransport): carriers bind 2 dissimilar components (substrates) & transport them together across a membrane.

Antiport (exchange diffusion): carriers exchange one component for another across a membrane.

Vesicle Transport





Exocytosis is the term applied when transport is out of the cell.

Endocytosis is the case when a molecule causes the cell membrane to bulge inward, forming a vesicle.

… Computing

A look at history

Since the origins, Computer Scientist have looked to relationships among machines and living organisms:

- McCulloch and Pitts: ***Neural Networks*** (1943).
- Von Neumann: ***Cellular Automata*** (1966).
- Lindenmayer: ***L systems*** (1968).
- Holland: ***Genetic Programming*** (1975).
- Adleman: ***DNA Computing*** (1994).

Membrane computing

- Gh. Păun, ***Computing with Membranes***, 1998-2000.
- Membrane Computing looks at the whole cell structure and functioning as a computing device.
- Membranes play a fundamental role in the cell as filters and separators.
- **Modeling the living cell is beyond the purpose of Membrane Computing!!**

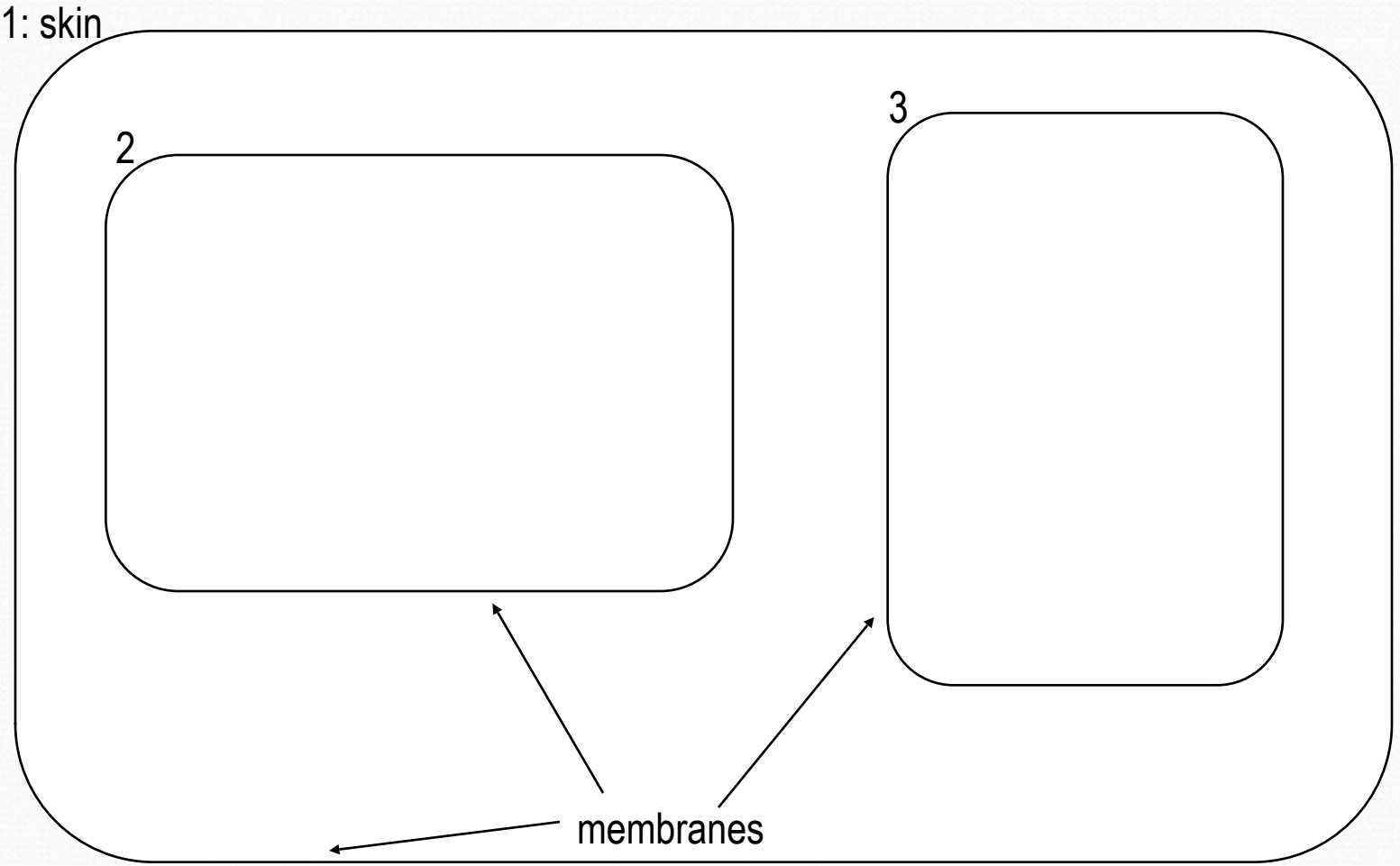
A membrane system (or P-system)

- A membrane structure formed by several membranes embedded in a unique main membrane.
- The objects are represented as symbols of a given alphabet (each symbol denotes a different object).
- Multi-sets of objects are placed inside the regions delimited by the membranes (one per each region).
- Sets of evolution rules associated with the regions (one set per region), which allow the system:
  - to produce new objects starting form the existing ones
  - to move objects from one region to another

A membrane system (or P-system)

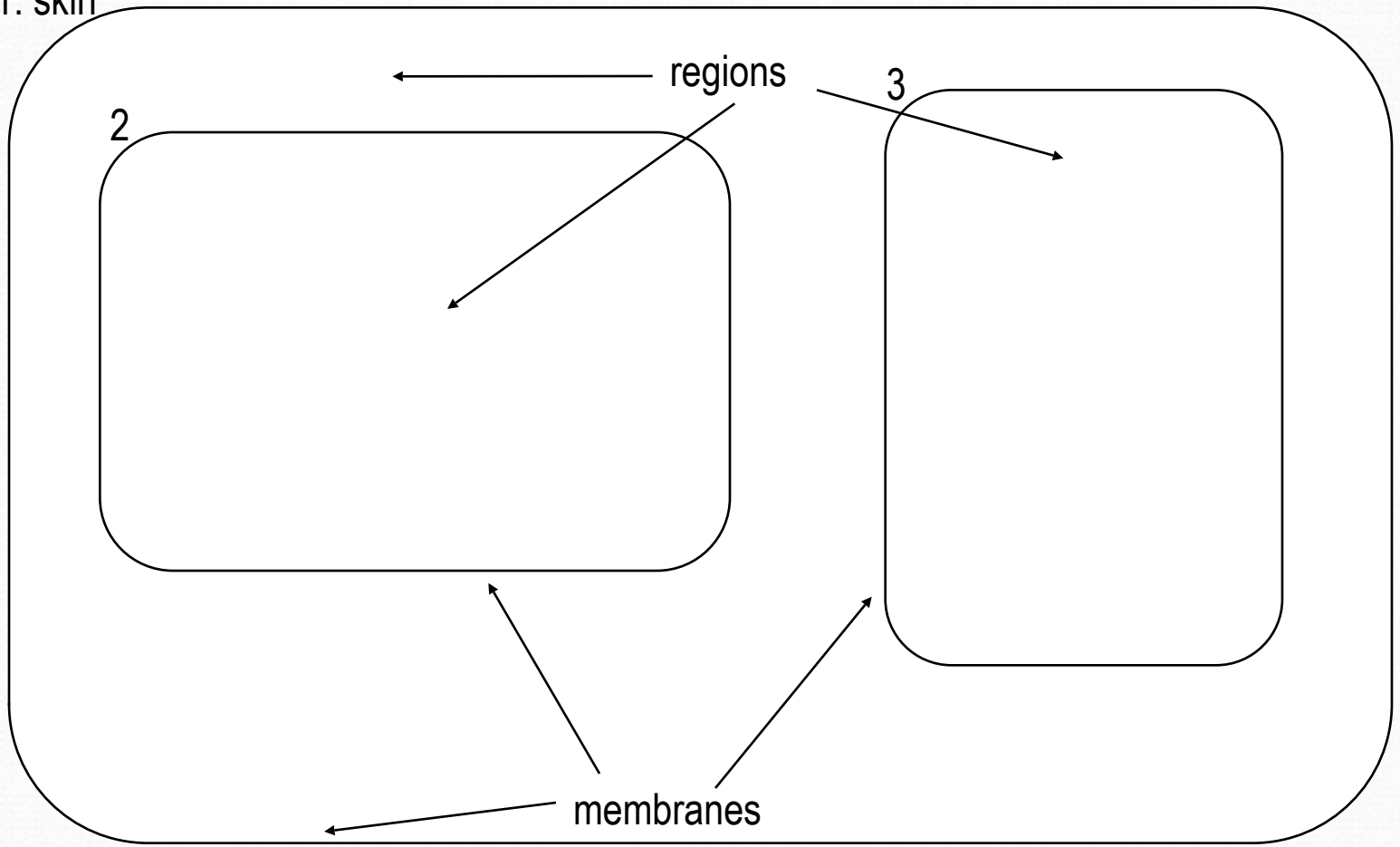
A membrane system (or P-system)

Membrane structure



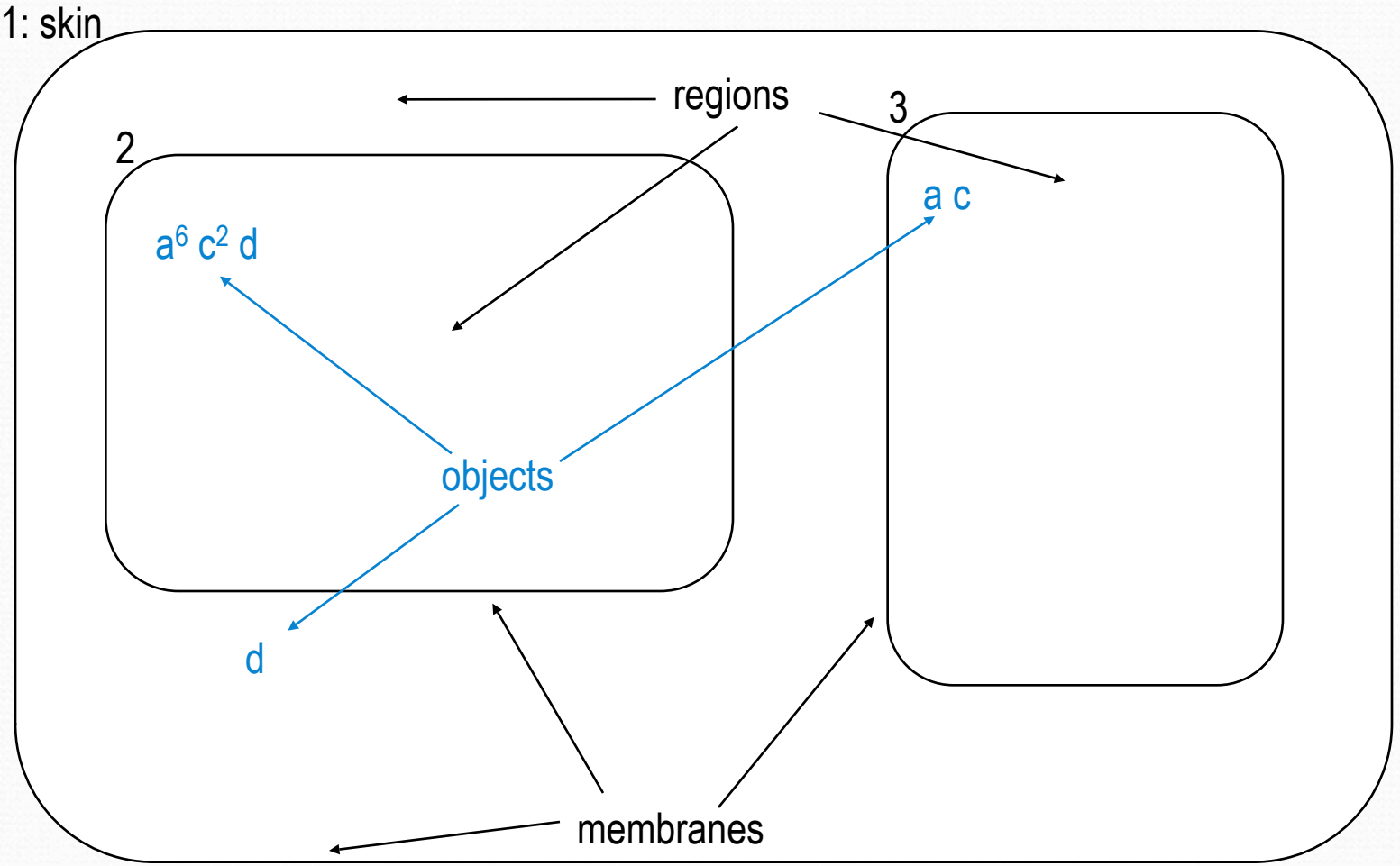
A membrane system (or P-system)

Membrane structure



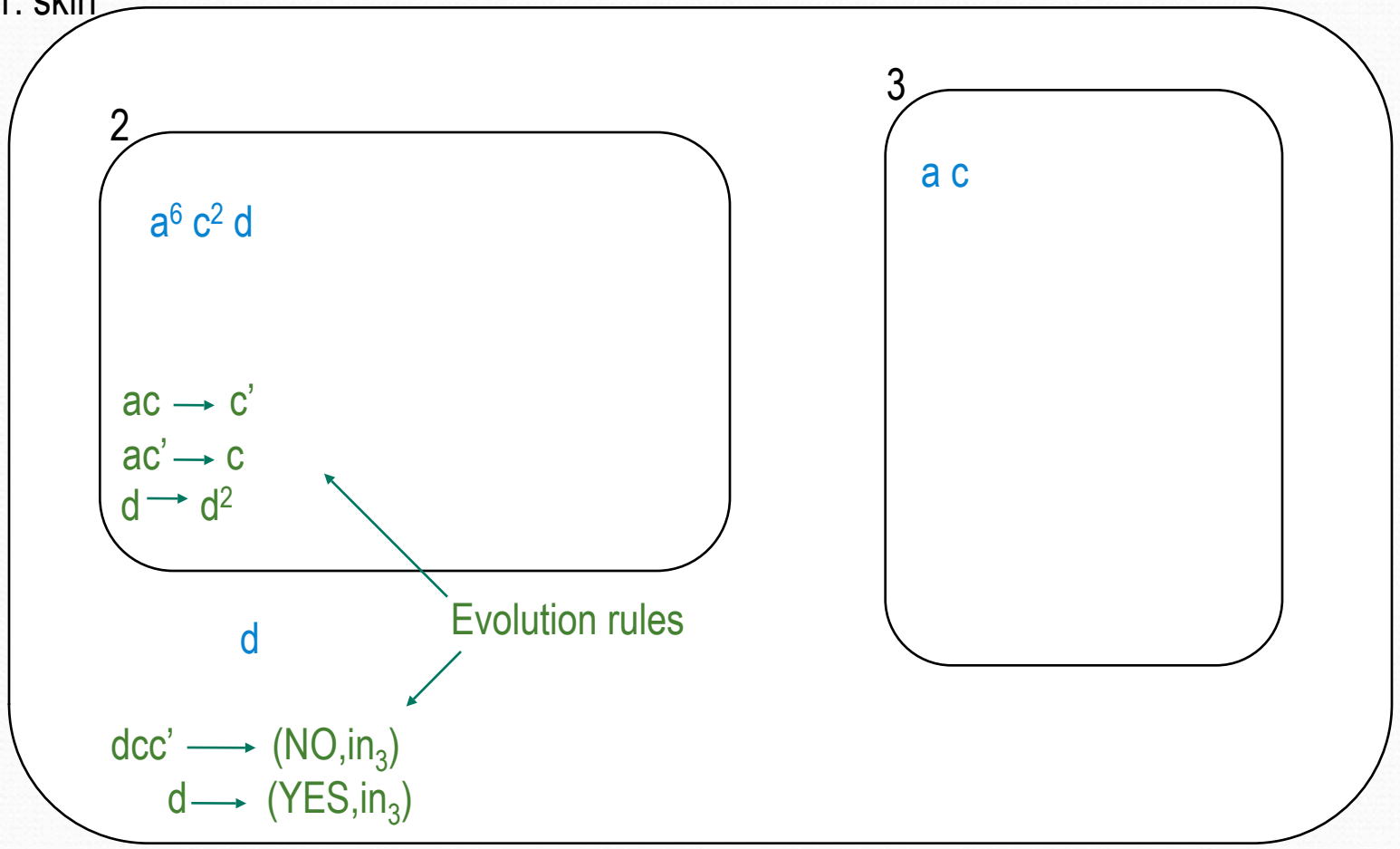
A membrane system (or P-system)

Membrane structure

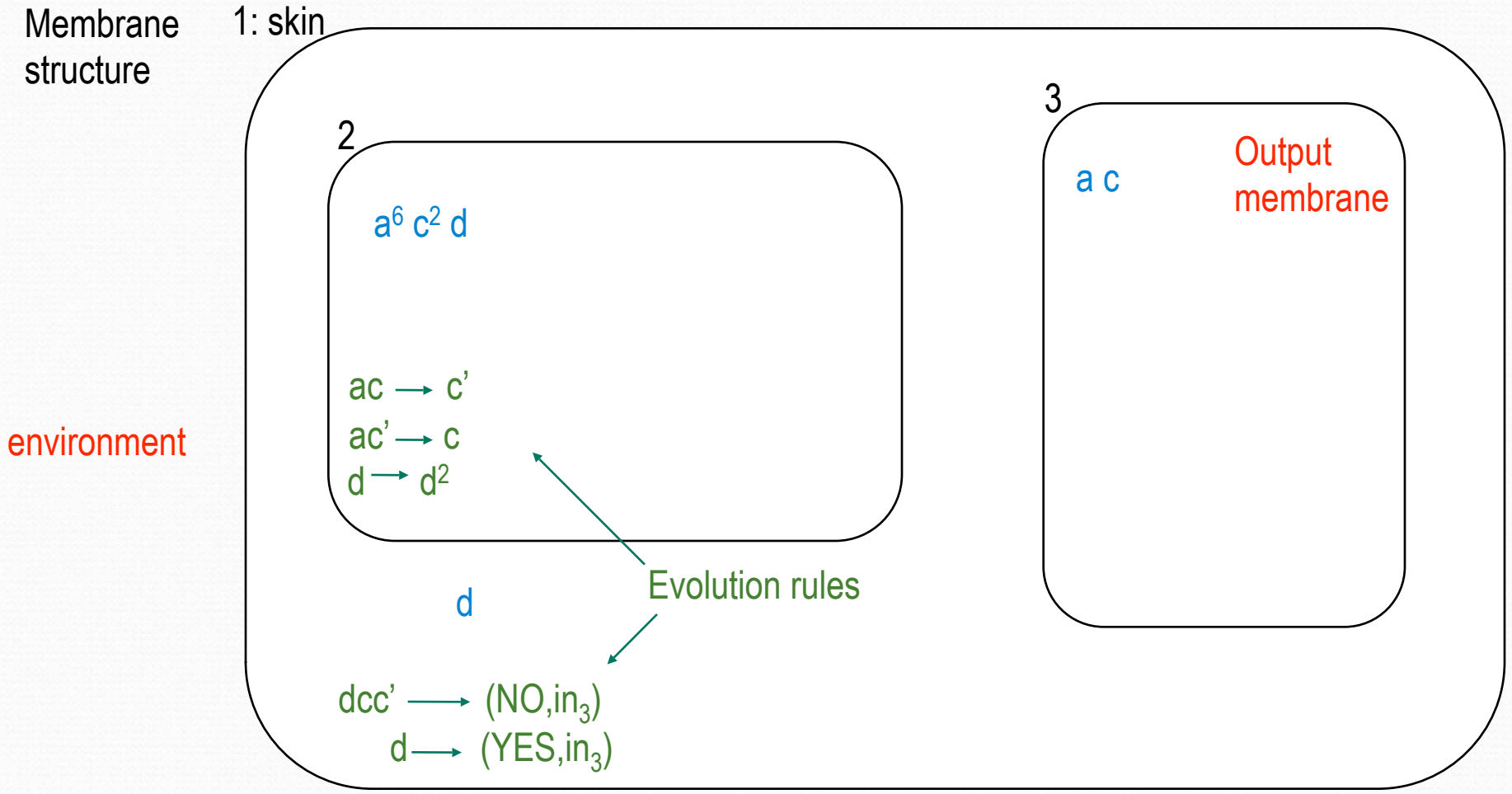


A membrane system (or P-system)

Membrane structure



A membrane system (or P-system)



A membrane system (or P-system)

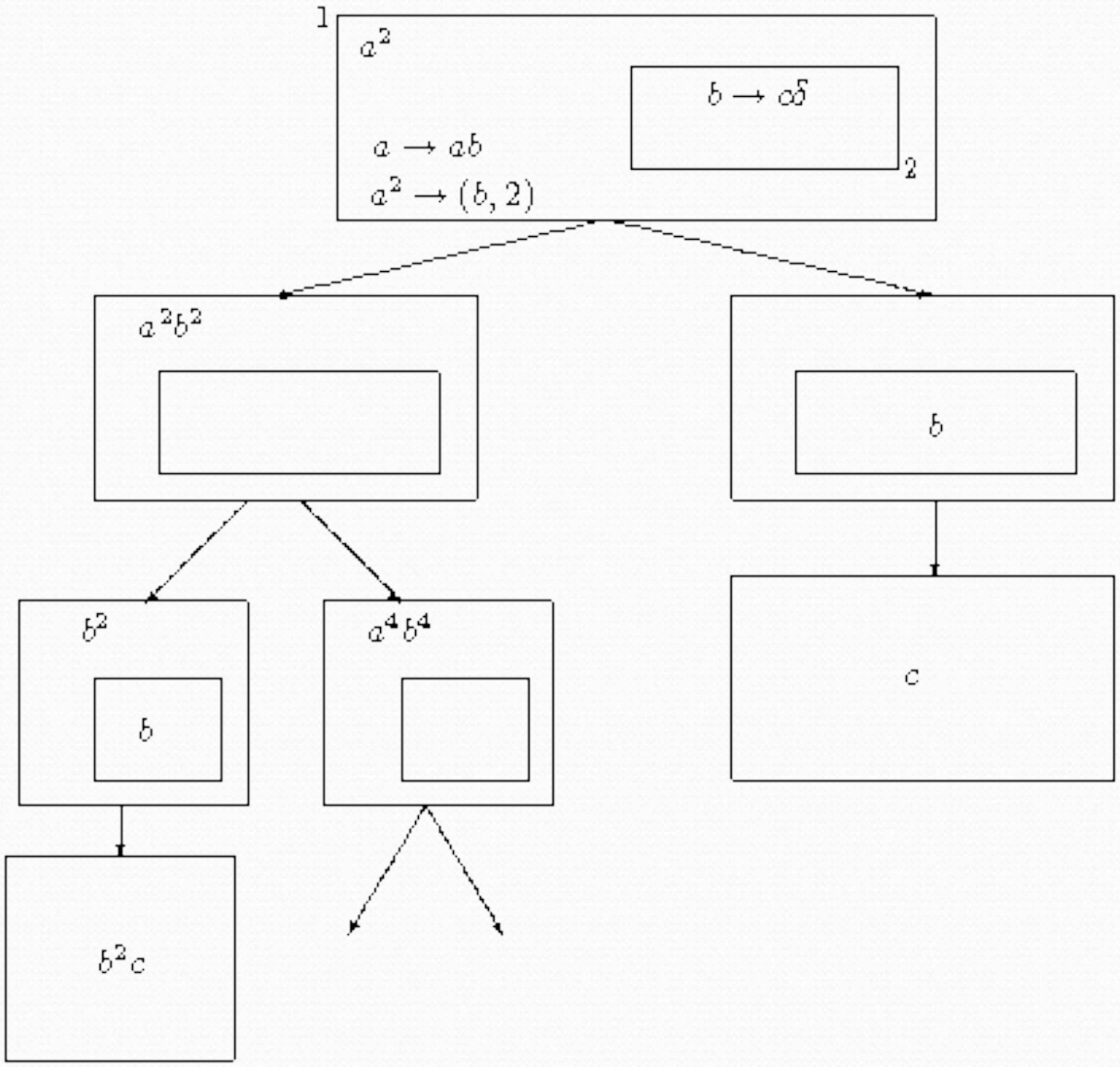
- A multiset is like a set, but every element can have more than one copy (we can formally define it as an application that associates a natural number to every element).
- Typically, an evolution rule is of the form $u \rightarrow v$ and it says that a copy of every object in u is replaced by a copy of the objects appearing in v (with some extra information).
- Example: $d^2c \rightarrow a\ (b,in_3)$

  The application of this rule in a region R, consumes 2 elements of type d and 1 of type c, and produces 1 element a in the same region, and 1 element b in region 3 (supposed it is inner to R).

A computation in a P-system

- We start with an **initial configuration**: an initial membrane structure and some initial multisets of objects placed inside the regions of the system.
- We apply the rules in a **non-deterministic maximal parallel manner**: in each step, in each region, each object that can be evolved according to some rule must do it.
- A computation is said **successful** if it halts, that is, it reaches a configuration where no rules can be applied.
- The **result** of a successful computation may be the multisets formed either by the objects contained in a specific output membrane or by the objects sent out of the system during the computation, or…
- A non-halting computation yields no result.

One more example: Non-determinism



More than evolution and communication

- **Membrane dissolution**: a special operator which can be used for dissolving a membrane.

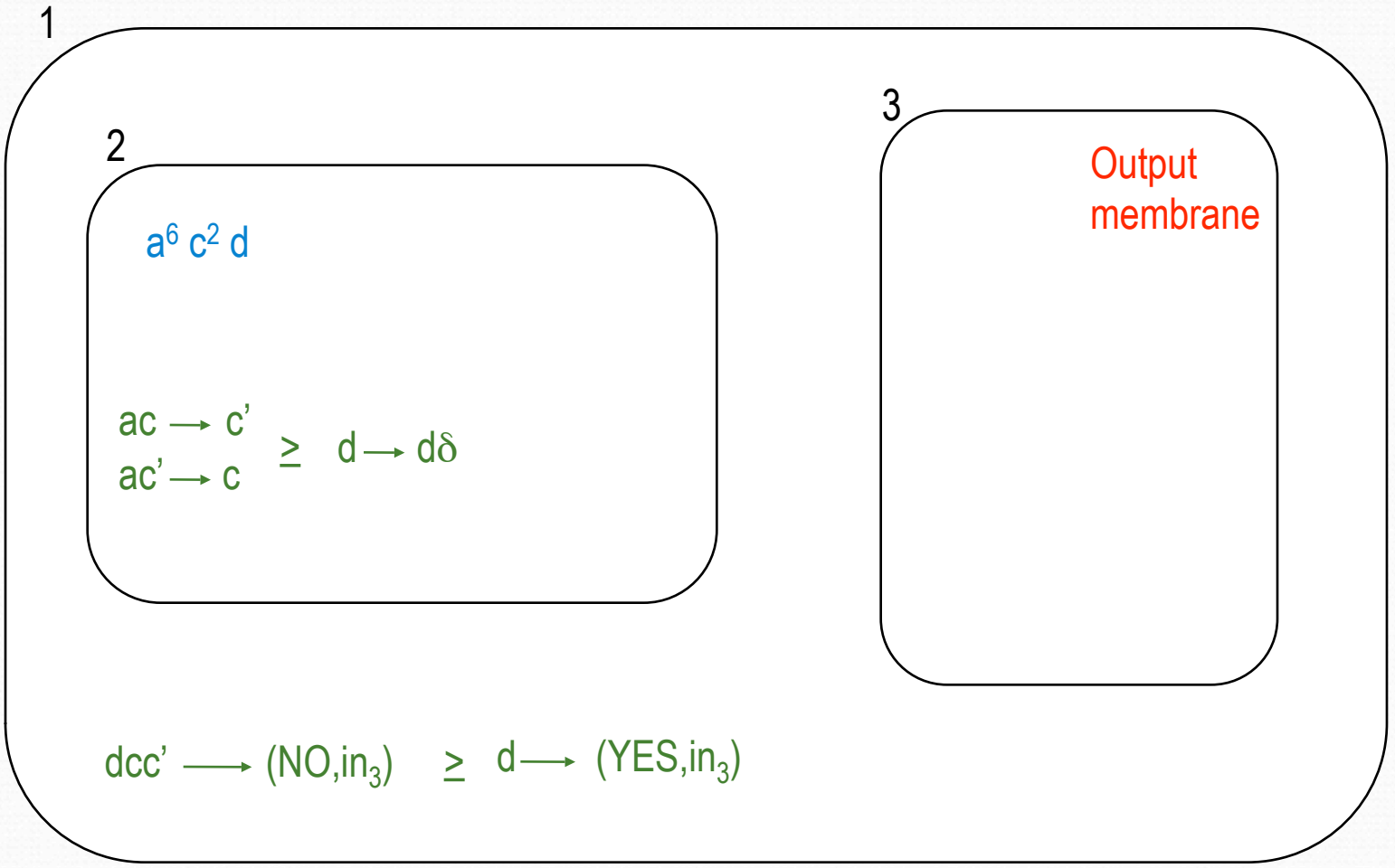
$$r: u \rightarrow v\delta$$

  if the rule r is used inside a membrane, such a membrane is dissolved after the application of the rule.

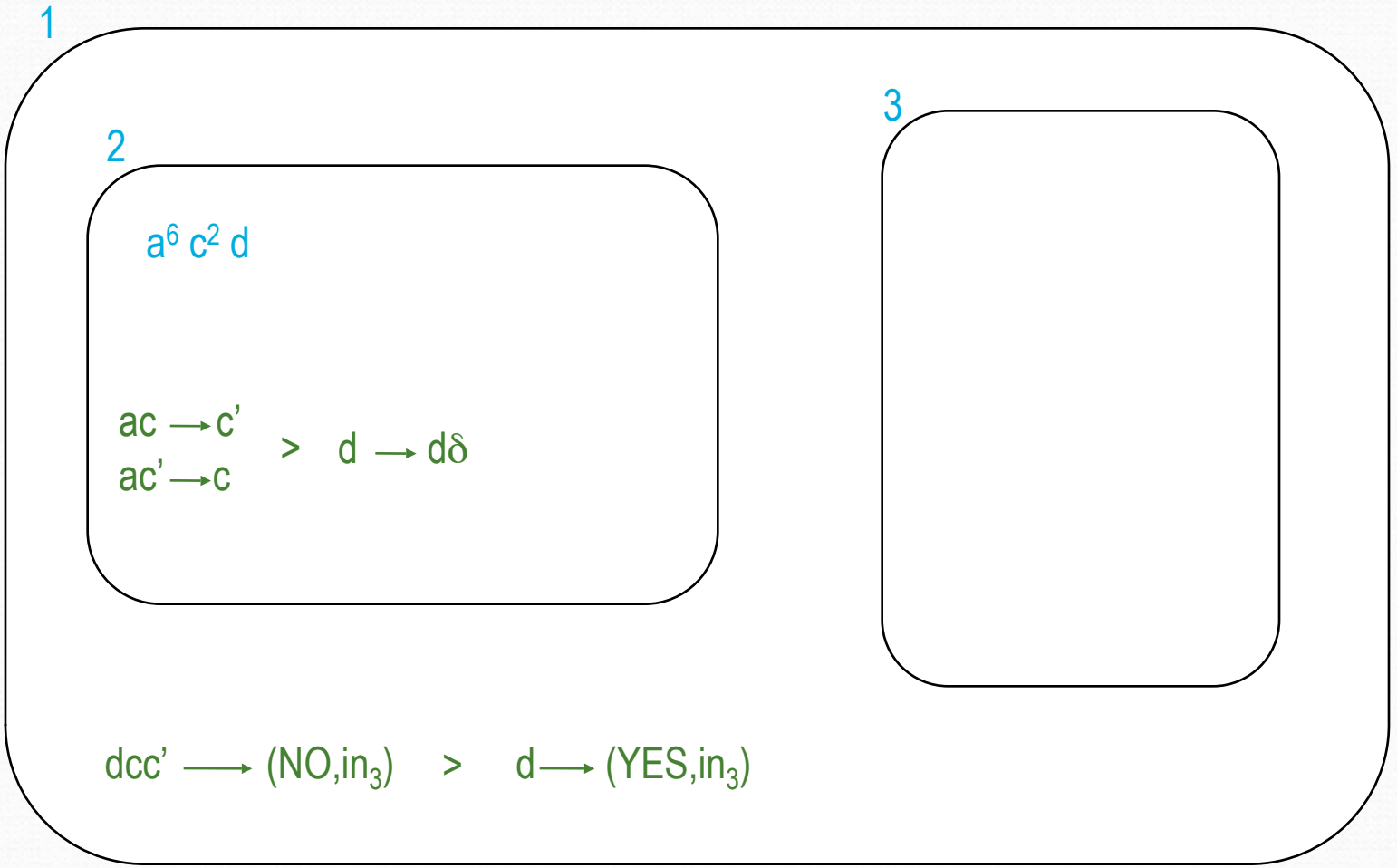
- **Membrane thickness**: two operators δ, τ for varying the permeability of the membranes.
- **Priority**: a partial order among the rules, which define a priority relationship:

  In each step, if a rule with high priority is applied then no rule with a lower priority can be applied in the same step.

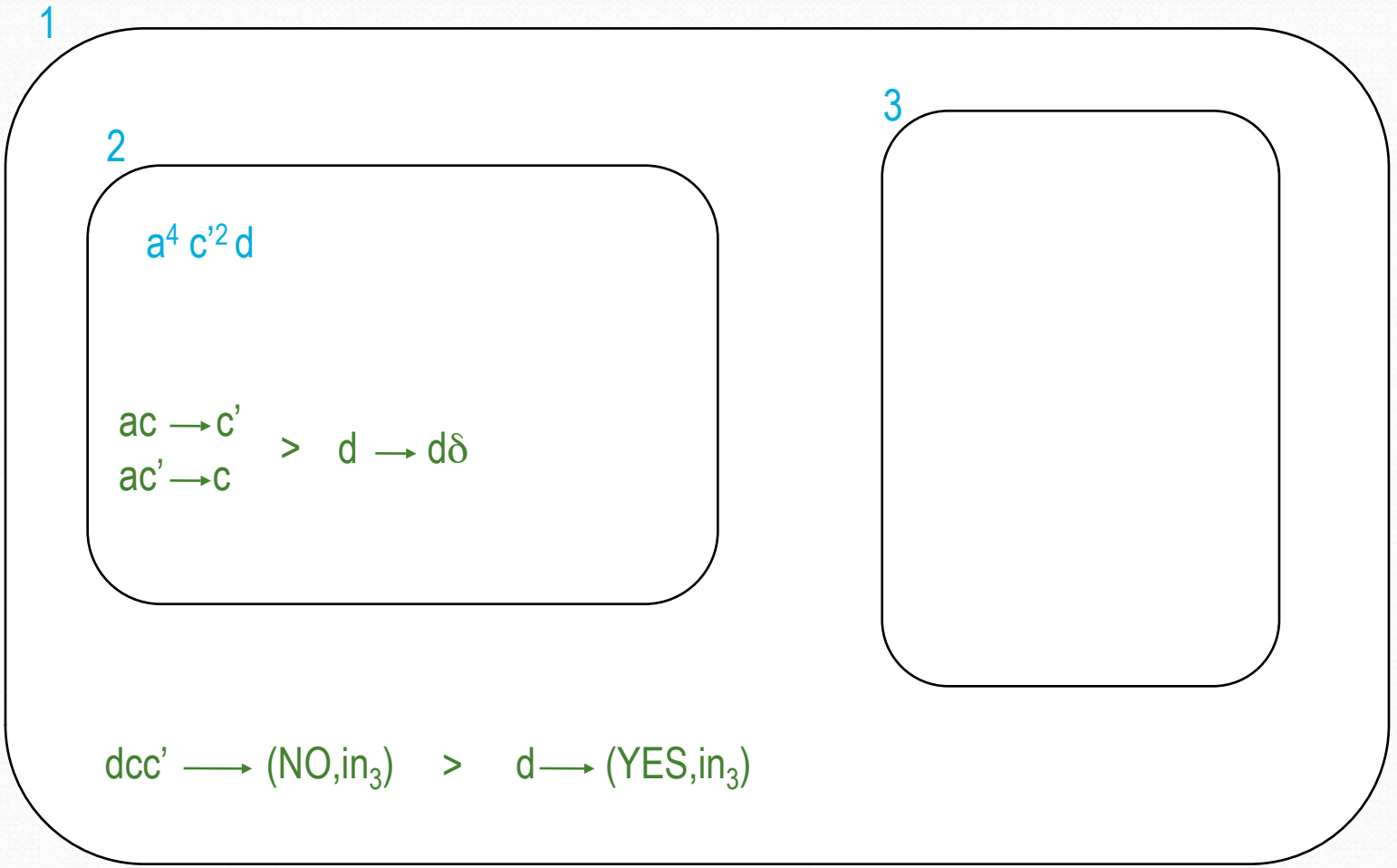
An example: divisibility predicate



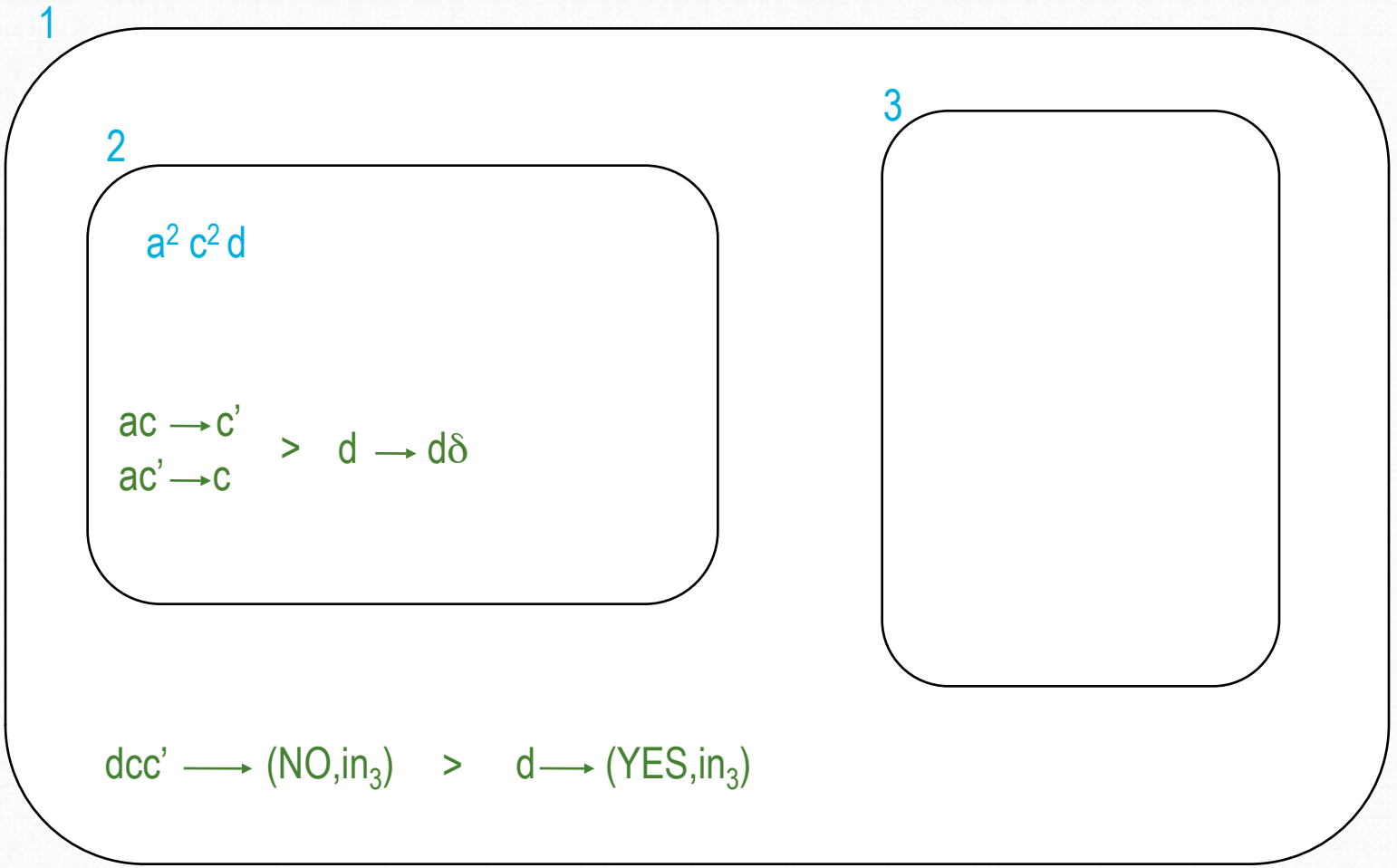
An example: divisibility predicate



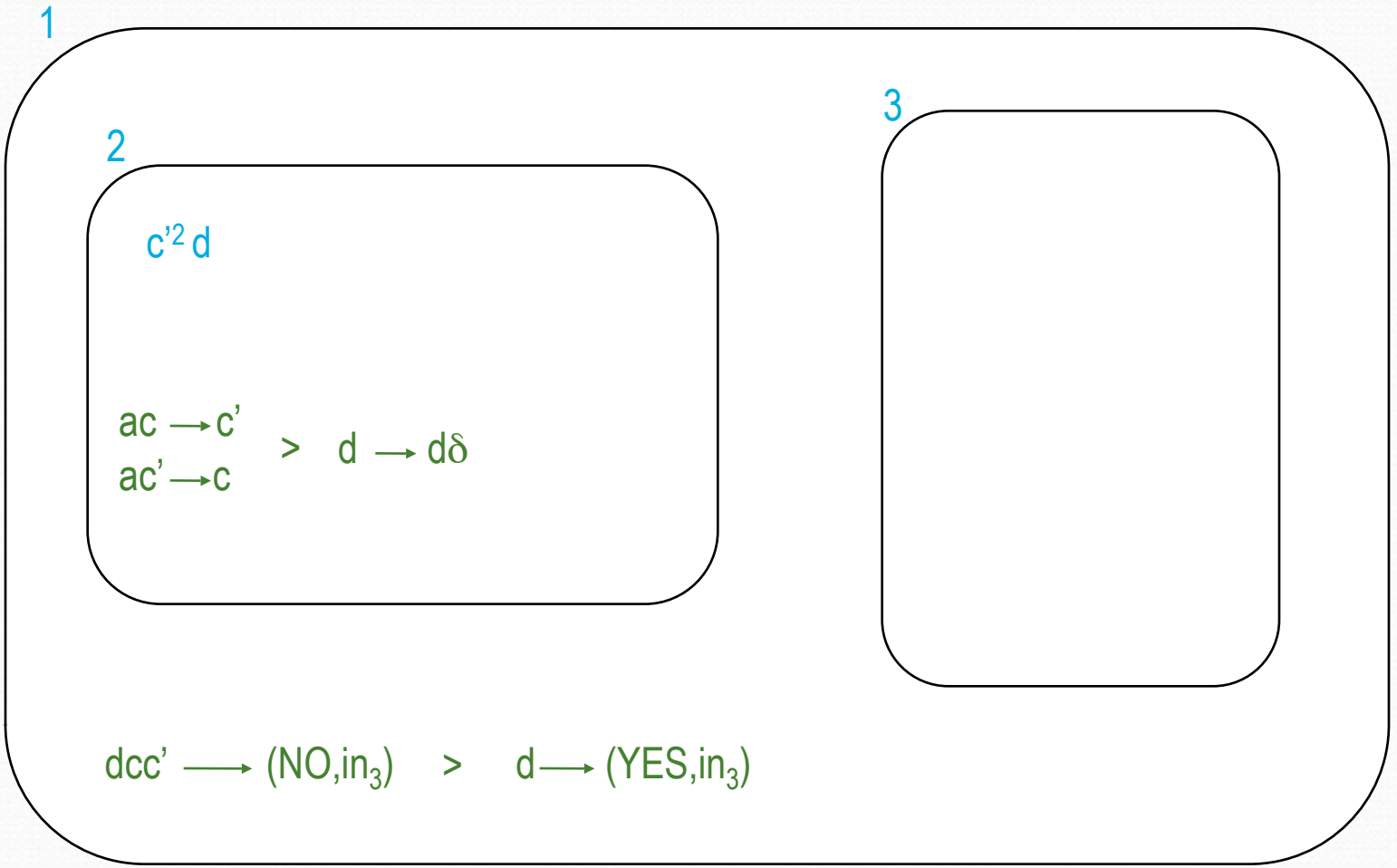
An example: divisibility predicate



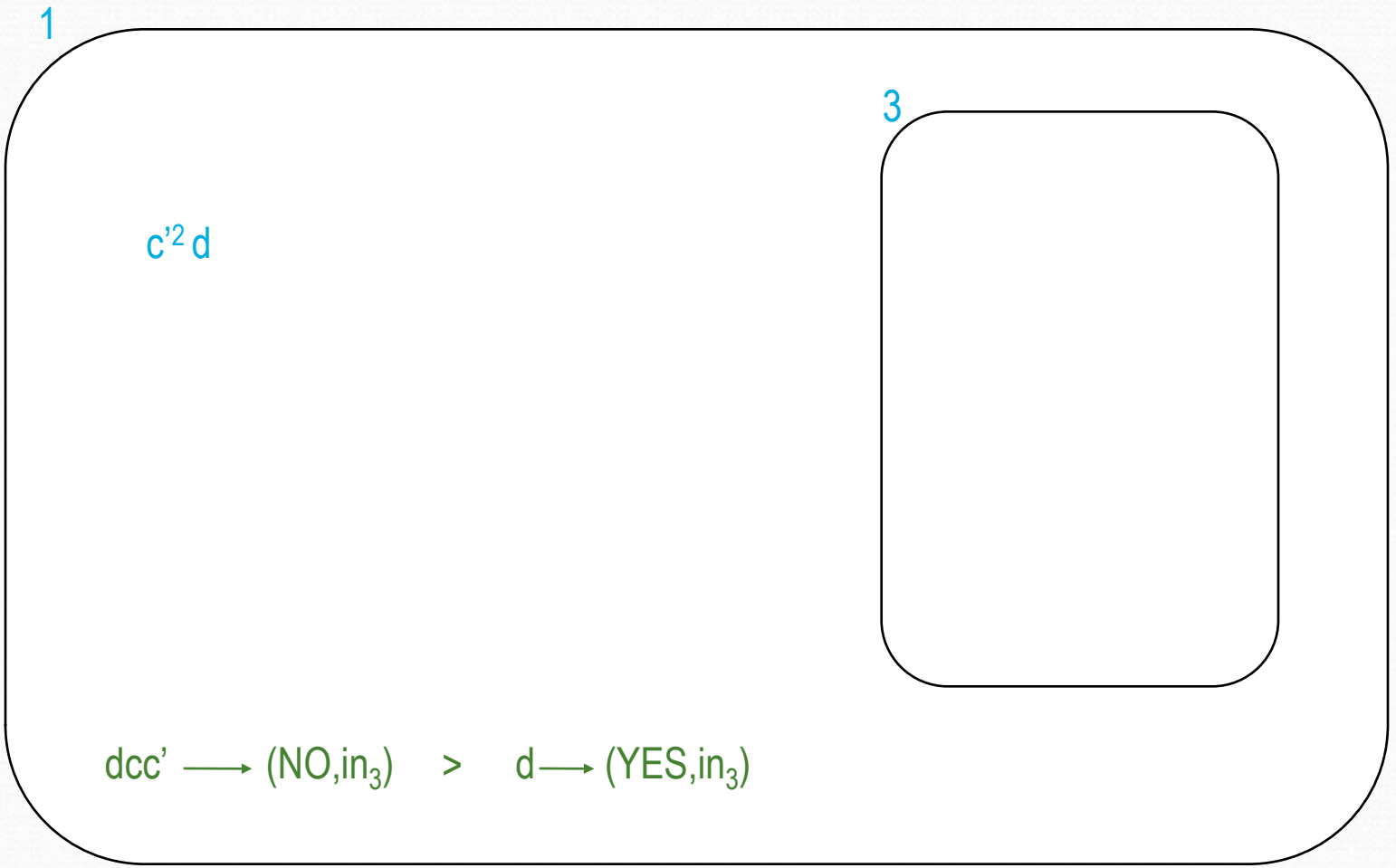
An example: divisibility predicate



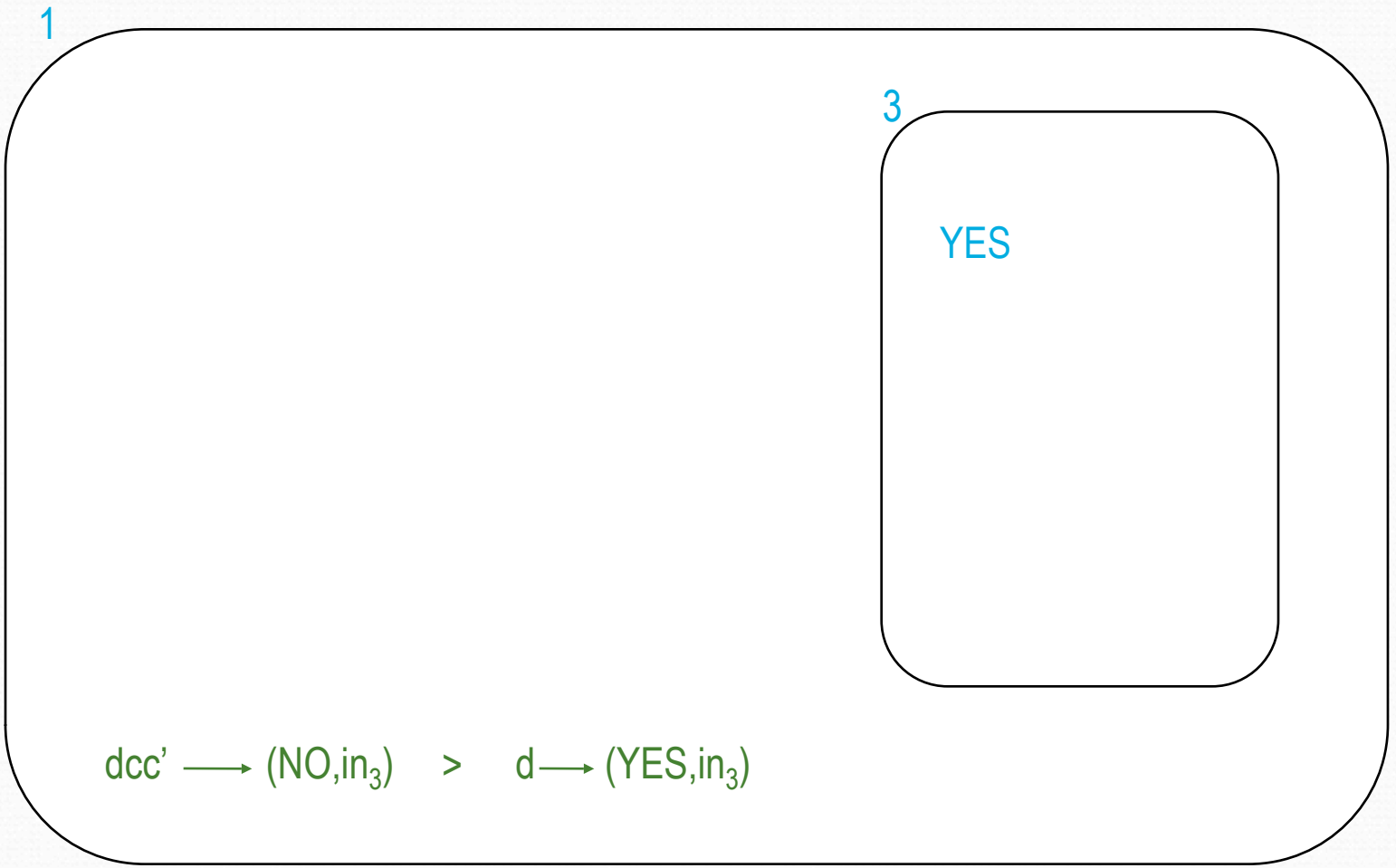
An example: divisibility predicate



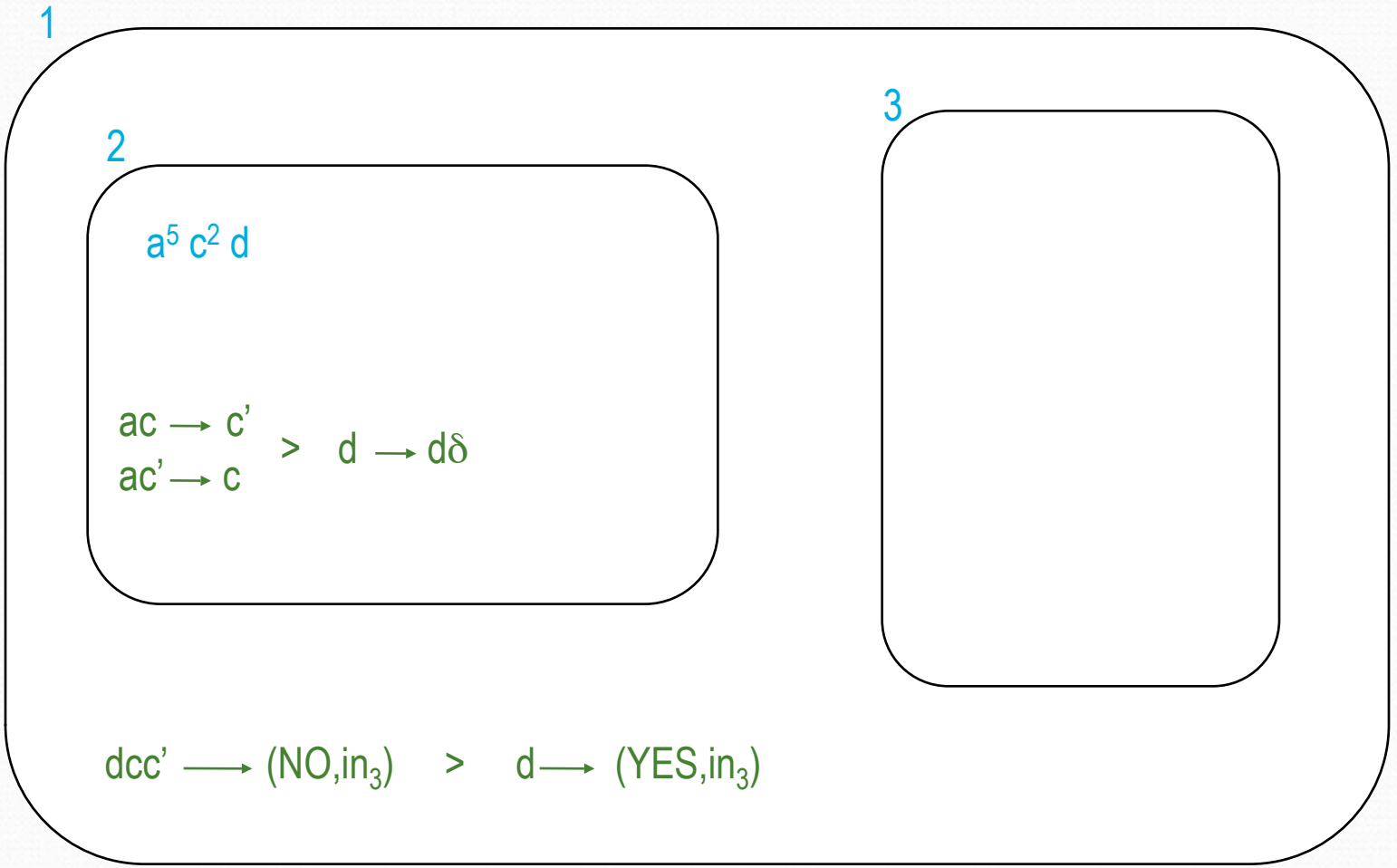
An example: divisibility predicate



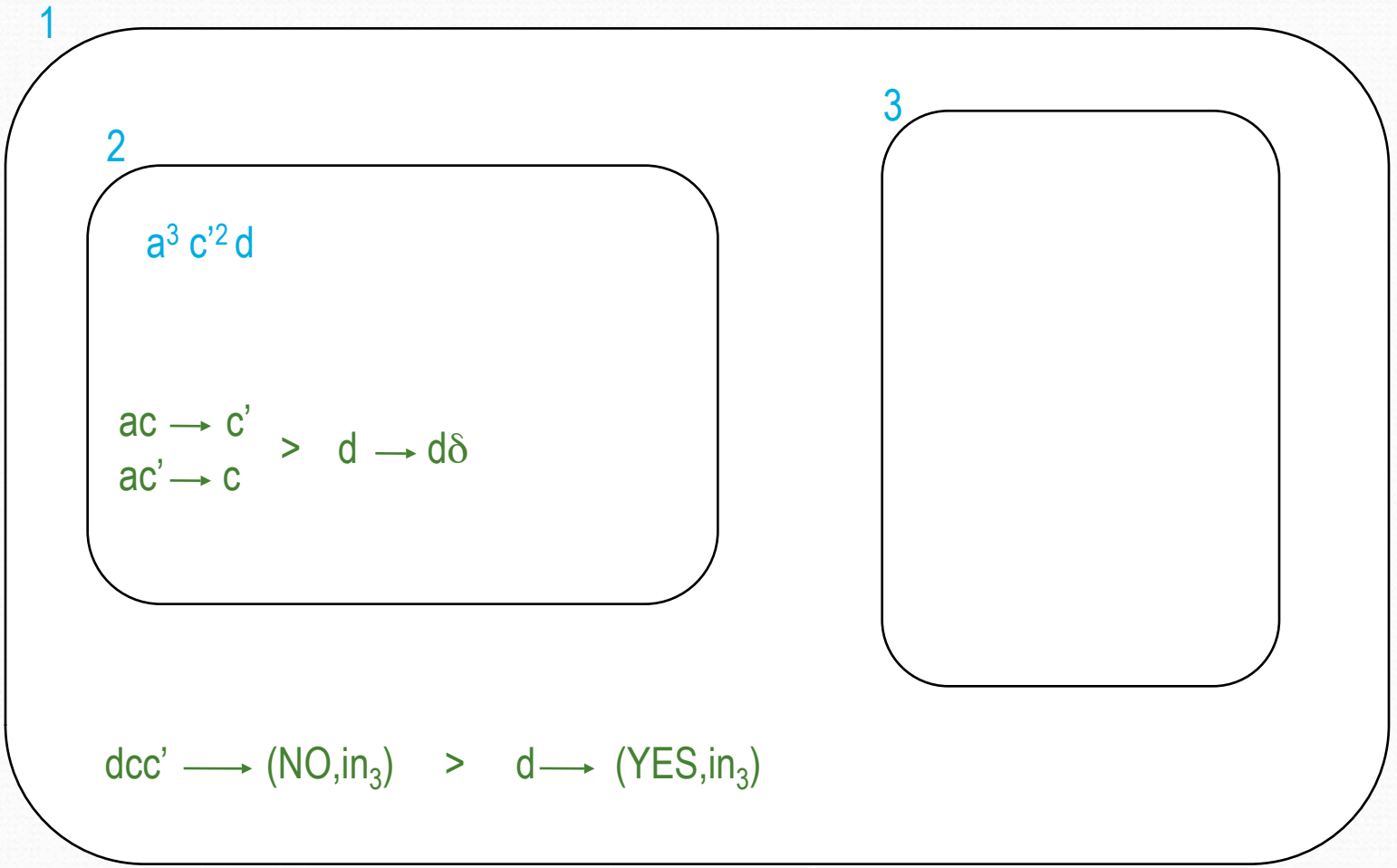
An example: divisibility predicate



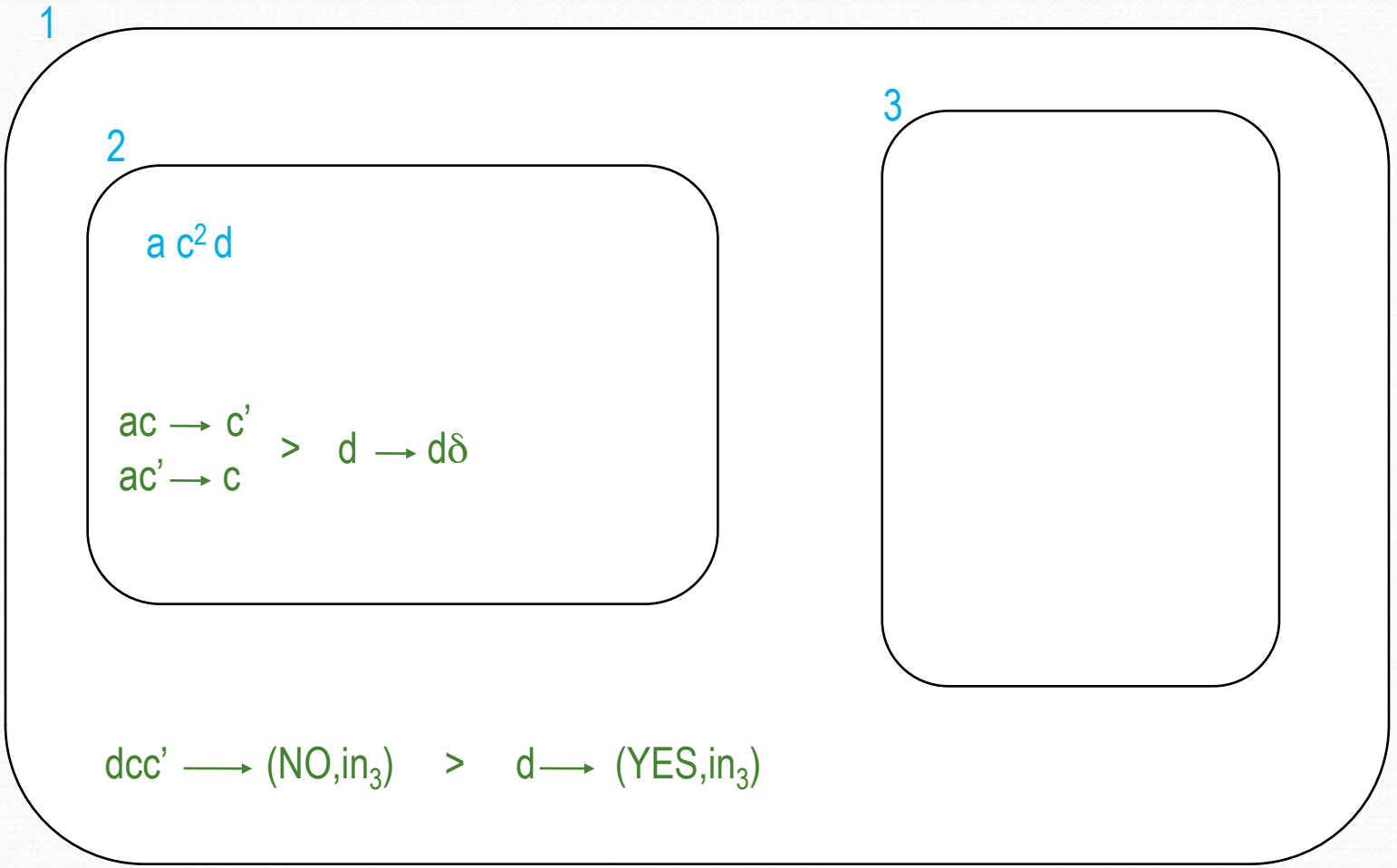
An example: divisibility predicate



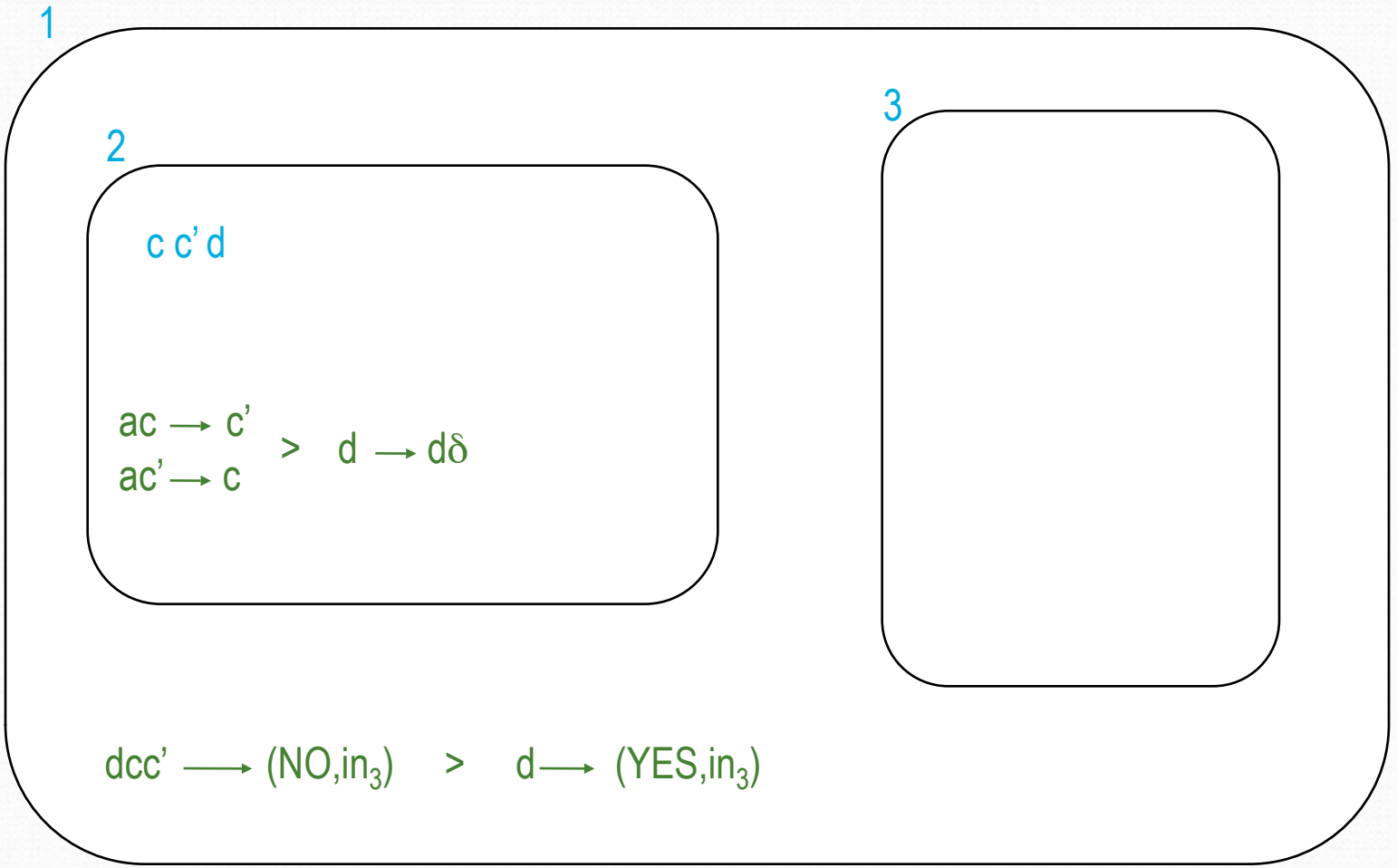
An example: divisibility predicate



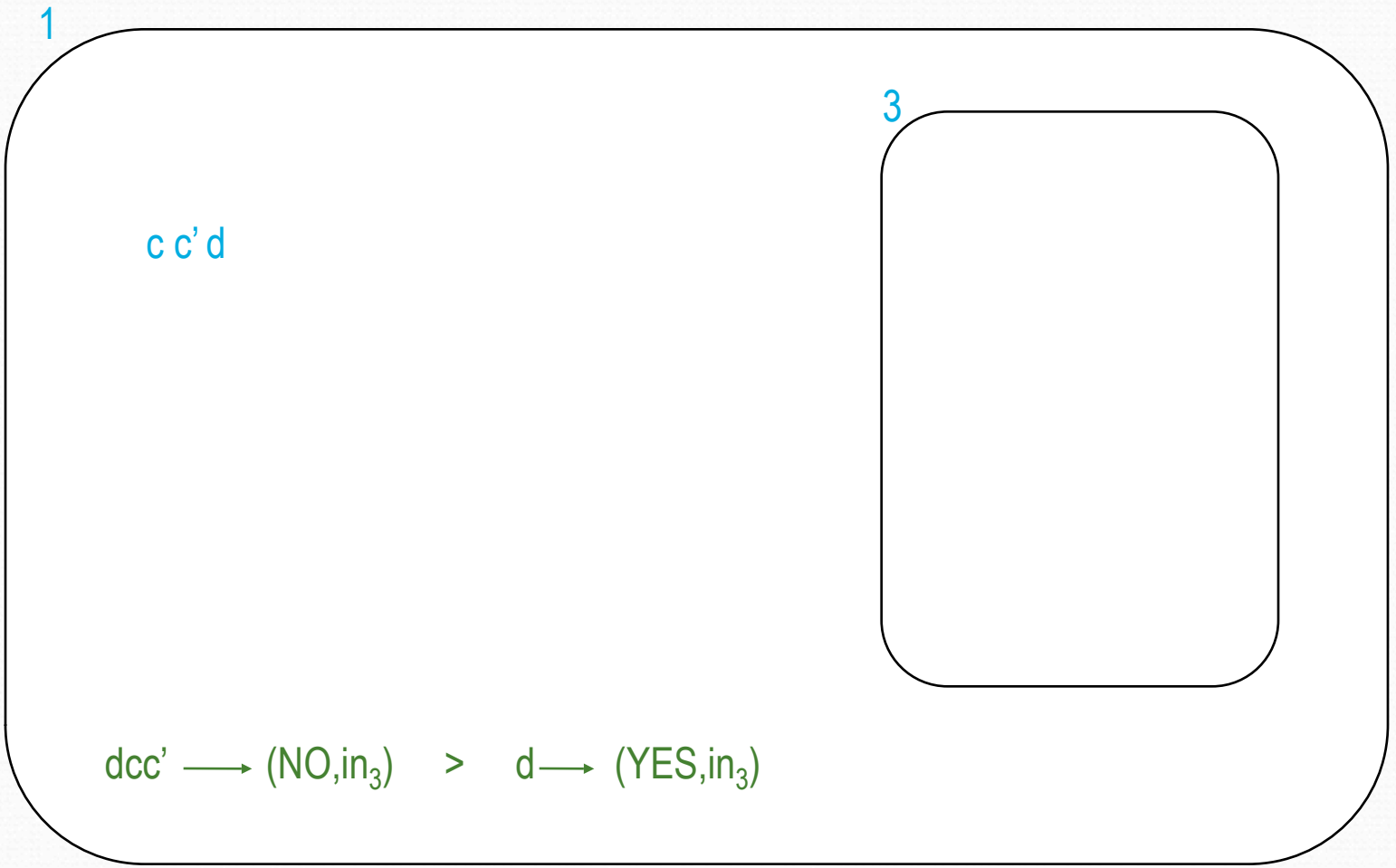
An example: divisibility predicate



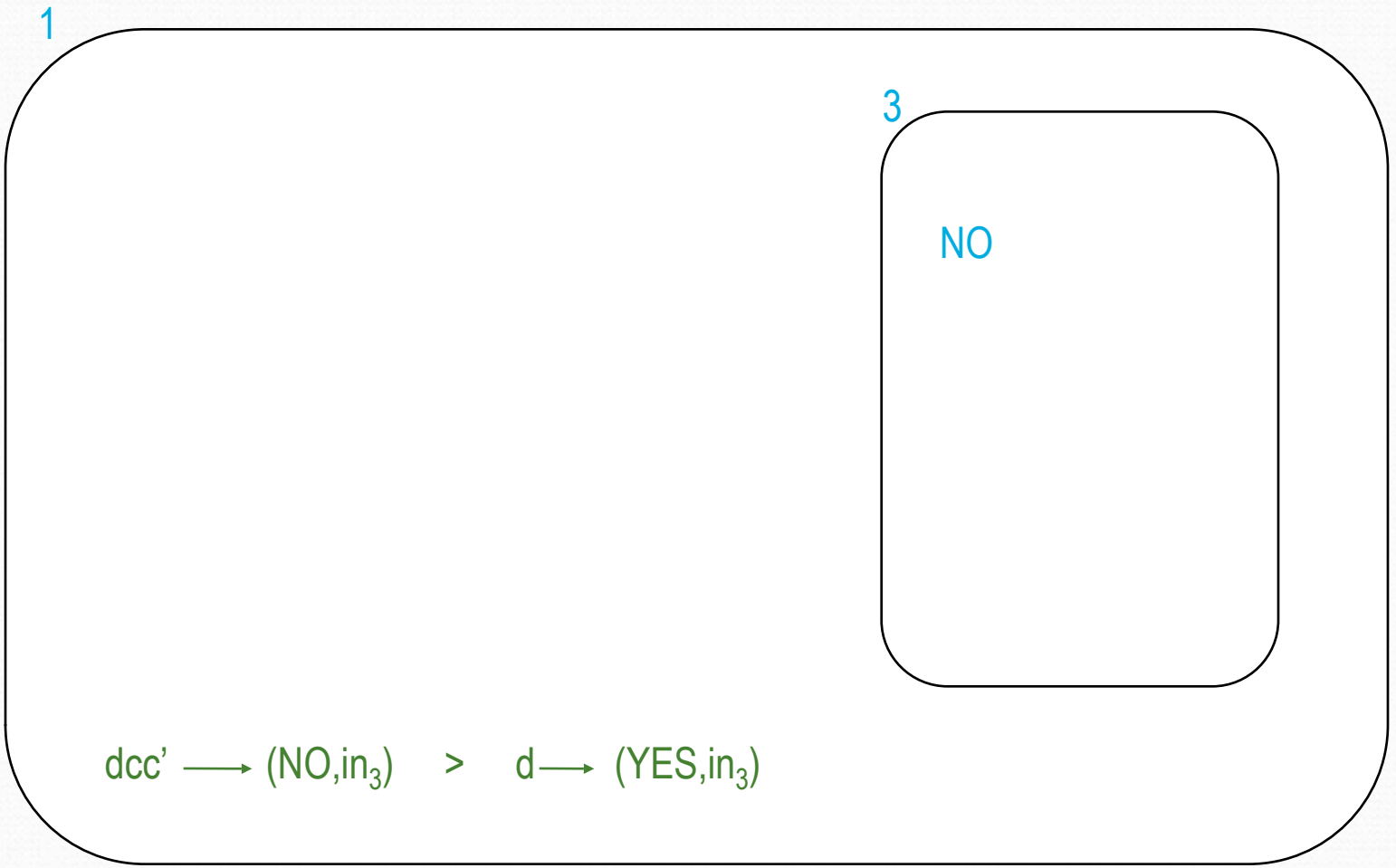
An example: divisibility predicate



An example: divisibility predicate



An example: divisibility predicate



Using catalysts

- Cooperation restricted to some special objects called **catalysts**

$$ca \rightarrow cv$$

Catalysts cannot be modified by any rule and cannot be moved from one region to another, but they "trigger" the application of the rule.

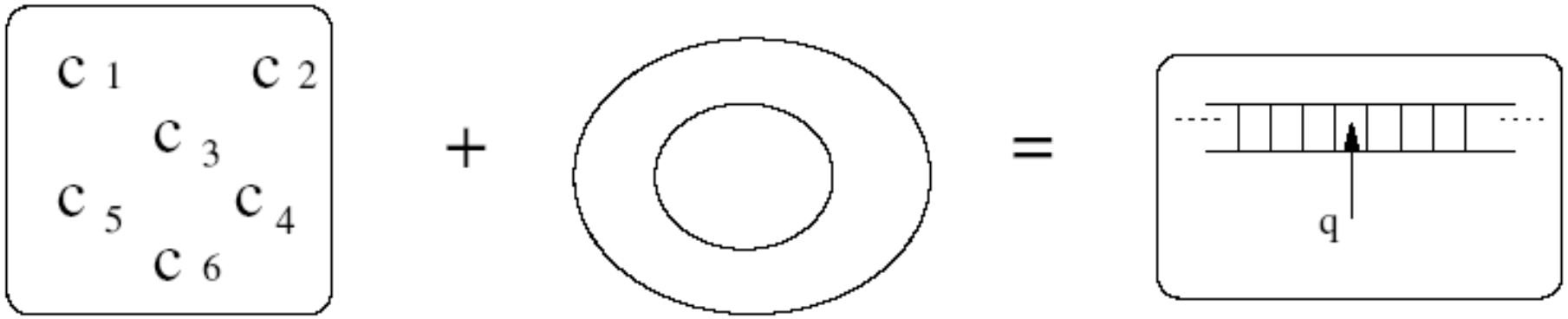
- **Bi-stable catalysts**: catalysts with two states

$$c\ a \rightarrow \underline{c}\ v$$

$$\underline{c}\ a' \rightarrow c\ v'$$

Some universality results

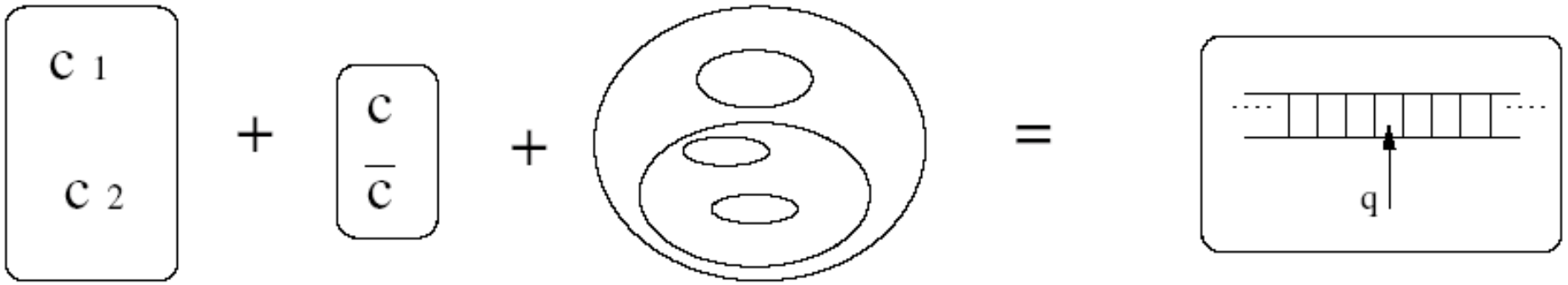
P-systems with catalysts are computationally universal



Cat_6 + 2Mem = T.M.

Some universality results

Using catalysts and bi-stable catalysts



$$Cat_2 + BiCat_1 + 5Mem = T.M.$$

Summary 1

- P-systems are bio-inspired distributed and parallel computing devices.
- The objects are located inside specific regions delimited by membranes.
- The system operates on multisets of objects.
- The objects evolve according to local rules associated with the regions.
- The rules can modify the objects or move them through the membranes.

… Some variants

V1: P-systems with string objects

- Now, the objects are **strings** over a given alphabet.
- The regions have associated languages instead of multisets of objects. The rules encode **string-operations**:
  - rewriting: $X \rightarrow (y, tar)$, with $tar \in \{here, in, out\}$
  - replicated rewriting:

    $X \rightarrow (y_1, tar_1) \| \ldots \| (y_n, tar_n)$, with $tar_1, \ldots, tar_n \in \{here, in, out\}$
  - splicing
  - ...
- More ingredients: membrane dissolution, membrane thickness, priority, etc…

V2: Communicative P-systems

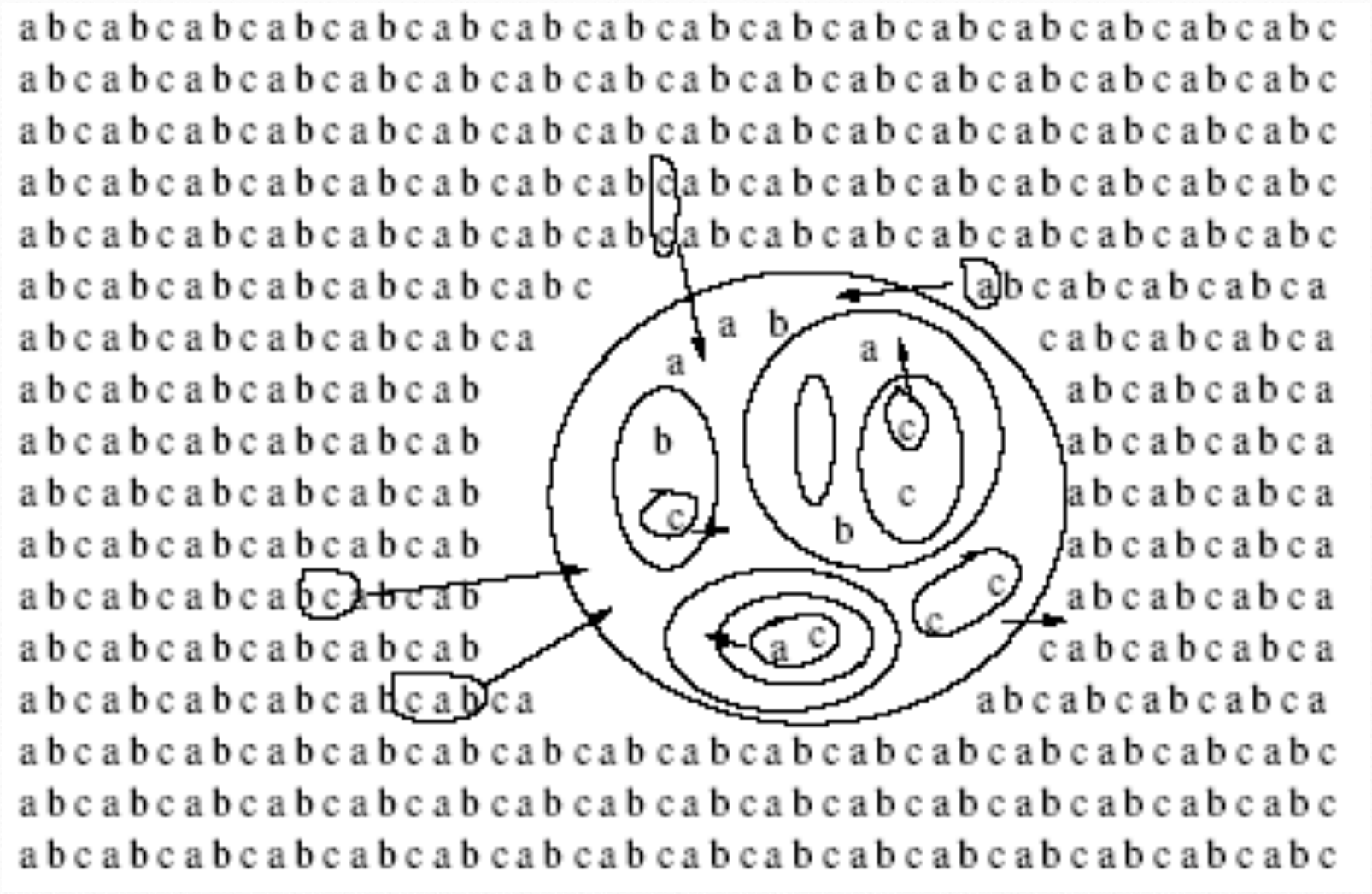
- **Communication** of objects through membranes is one of the most important ingredients of every P-system.

- What can we do if we allow only communication?

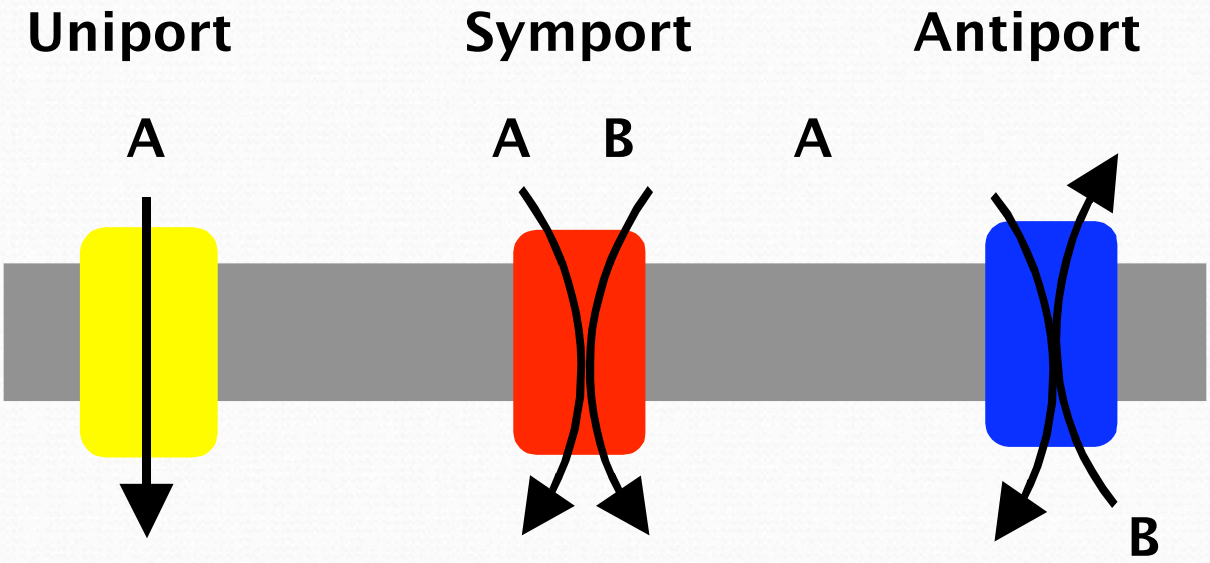
  Purely communicative systems: the objects are not changed during a computation, but they just change their place inside the system.

- In order to have enough elements for computation, we suppose that the system is embedded in an **infinite environment**, which contains an arbitrary number of copies of each object. The environment provides the objects the system needs to perform its internal computations.

V2: Communicative P-systems

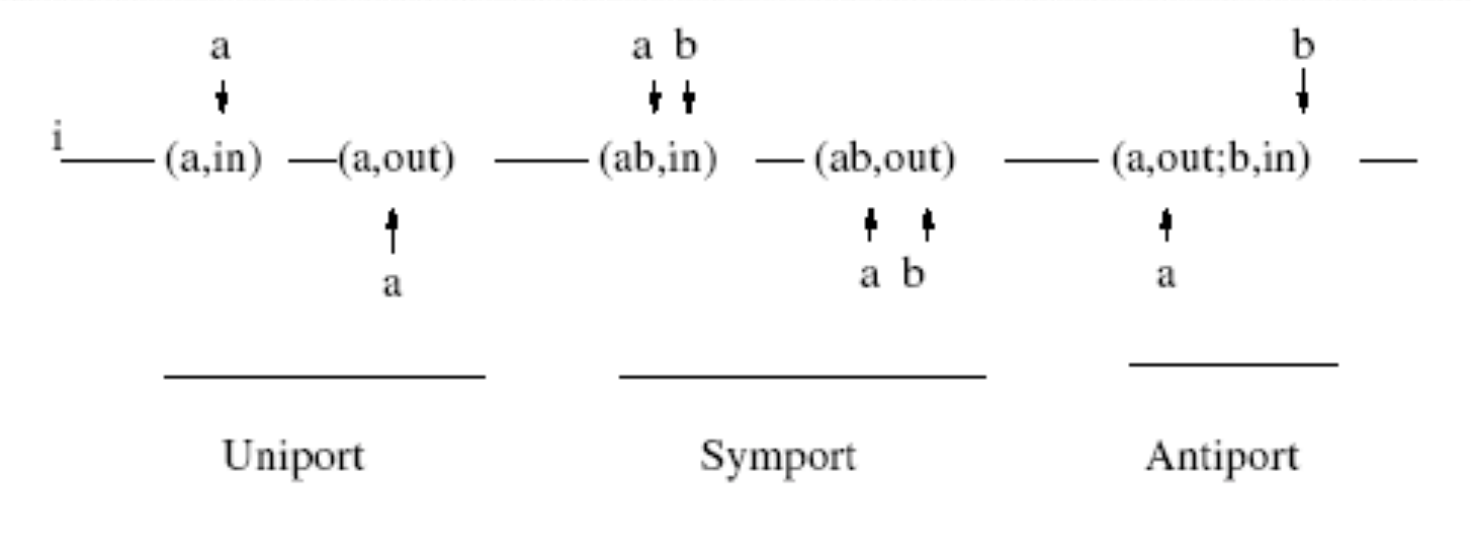


Bio-justification :Membrane transport of small molecules



P-systems with symport/antiport

- Rules encode symport/antiport mechanism:



- Initially they use only symbol objects.

- Generalization: *(x,in), (x,out), (x,in; y,out)*, for *x, y* multisets of arbitrary size

The power of communication

P-systems with symport/antiport are computationally universal (Păun, A., Frisco, P., Păun, Gh., 2003)

(1,2) + 1 Mem = T.M.

(2,0) + 4 Mem = T.M.

(3,0) + 2 Mem = T.M.

(3,0) + 1 Mem = T.M.

(n,m) denotes the size of the rules: for each *(x,in)*, *(x,out)*, $|x| \leq n$, and for each *(x,in; y,out)*, $max\{|x|,|y|\} \leq m$

Evolution–Communication P–systems

P-systems with boundary rules (rules associated to membranes, more than regions):

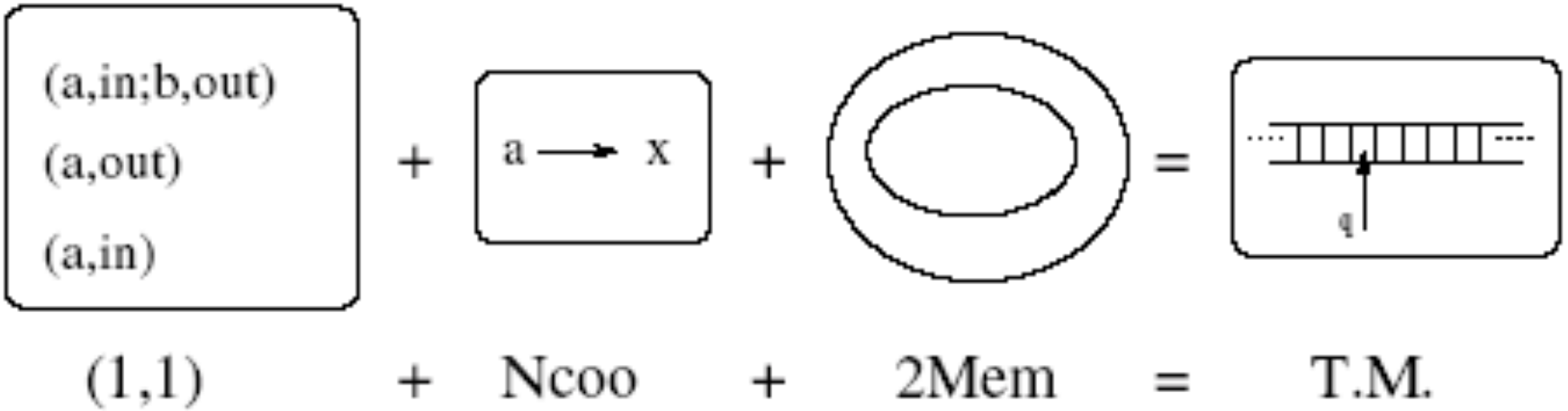
- Communication rules: $xx' [_i \, y'y \rightarrow xy' [_i \, x'y$

  $(x,in), (y,out), (x,in; y,out)$

- Evolution rules: $[_i \, y \rightarrow [_i \, y'$

The Power of EC P-systems

EC P systems are computationally universal



V3: P-systems with active membranes

- Rules are able to perform operation for modifying the membrane structure:
  - membrane creation: $[_i\ a\ ]_i \rightarrow [_j\ b\ ]_j$
  - membrane division: $[_i\ a\ ]_i \rightarrow [_k\ b\ ]_k [_j\ c\ ]_j$
  - membrane duplication: $[_i\ a\ ]_i \rightarrow [_k\ b\ [_j\ c\ ]_j\ ]_k$
  - membrane dissolution: $[_i\ a\ ]_i \rightarrow a$ where *a*, *b* are objects and *i*, *j*, *k* are labels of possible membranes.

- Communication and Evolution rules assume the form:

$$[_i\ a \rightarrow v\ ]_i, \quad [_i\ a\ ]_i \rightarrow [_i\ ]_i\ b, \quad a[_i\ ]_i \rightarrow [_i\ b\ ]_i$$

where *a*, *b* are objects and *i*, *j*, *k* are labels of possible membranes.

Trading time for space

- The Hamiltonian Path Problem (HPP) can be solved in quadratic time and the SAT problem can be solved in linear time by P-systems with active membranes, by using membrane division and dissolution.

- HPP can be solved in linear time by using membrane creation.

  Idea: generate in an *efficient manner* all paths from a specified initial node, then checking whether or not at least one of these paths is Hamiltonian.

What else

Whatever you want…

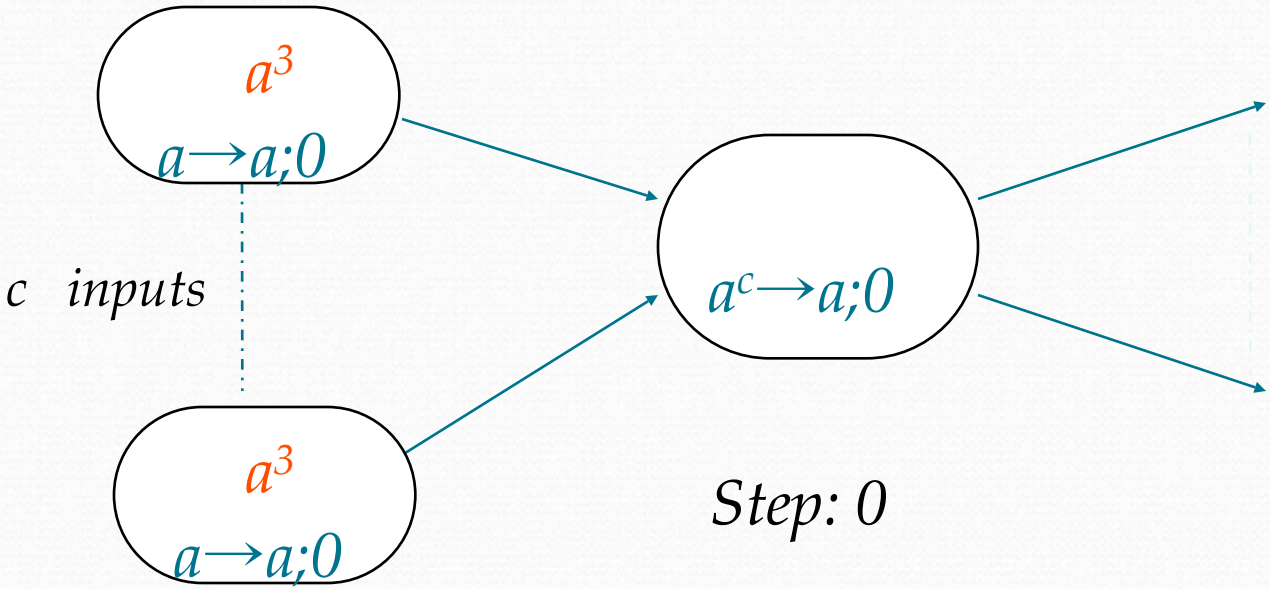
- Energy-Controlled P-systems.
- P-systems with promoters/inhibitors.
- P-systems with carriers.
- P-systems with gemmation of mobile membranes.
- Tissue P-systems.
- Probabilistic P-systems.
- P-systems with elementary graph productions.
- Parallel Rewriting P-systems.
- ...
- ...

V5: Spiking Neural P systems

- Based on the specific ideas of *spiking neurons.*
- A set of neurons (only one membrane) placed on the nodes of a (directed) graph and sending signals using the edges of the graph.
- Rules of two types:
  - *Spiking rules* (*firing*): $a^c \rightarrow a;d$ ($c \geq 1$, $d \geq 0$).
  - *Forgetting rules:* $a^s \rightarrow \lambda$.

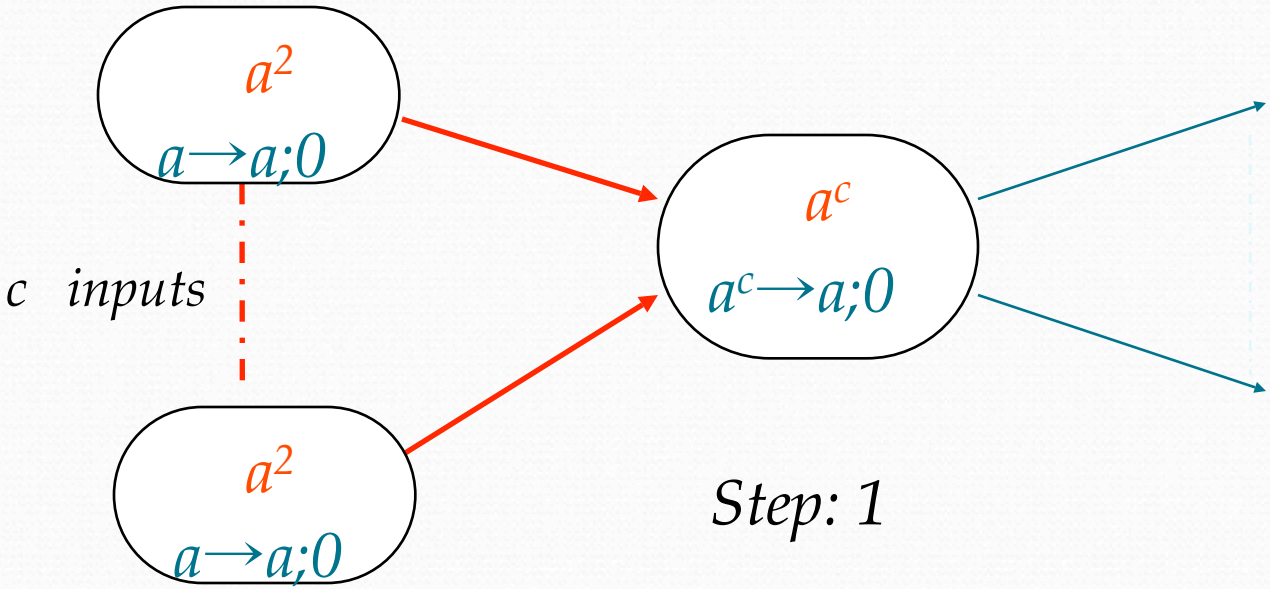
How the rules work

Spiking (*firing*) rules: $a^{c} \rightarrow a;d$



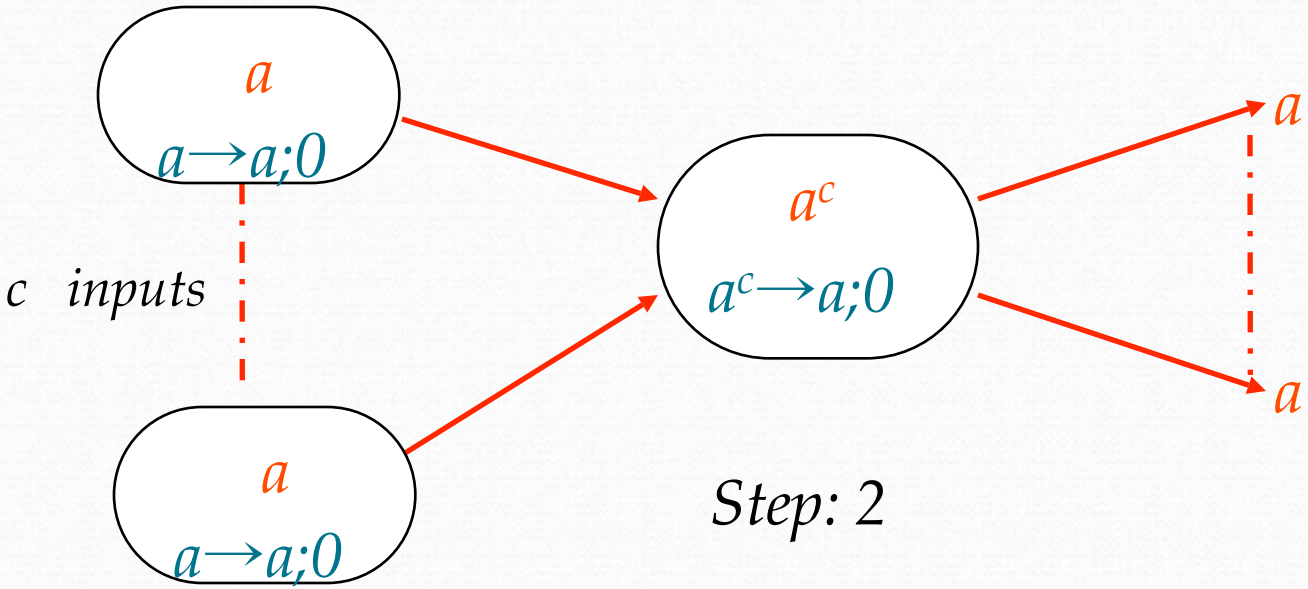
How the rules work

Spiking (*firing*) rules: $a^{c} \rightarrow a;d$



How the rules work

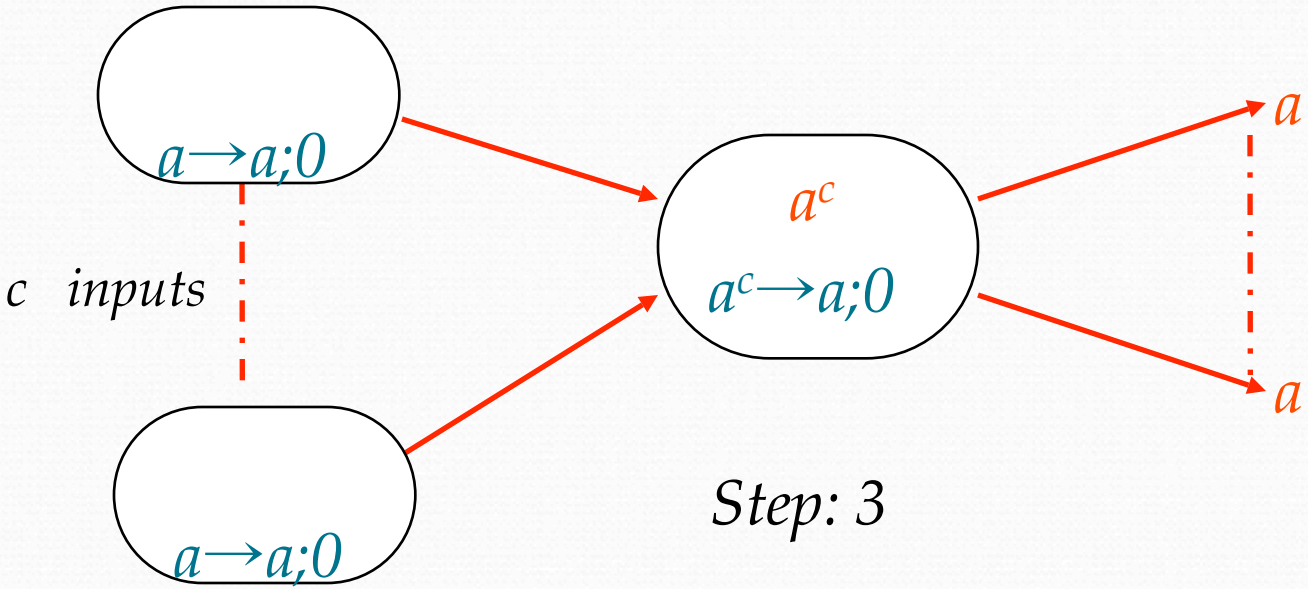
Spiking (*firing*) rules: $a \xrightarrow{c} a;d$



Step: 2

How the rules work

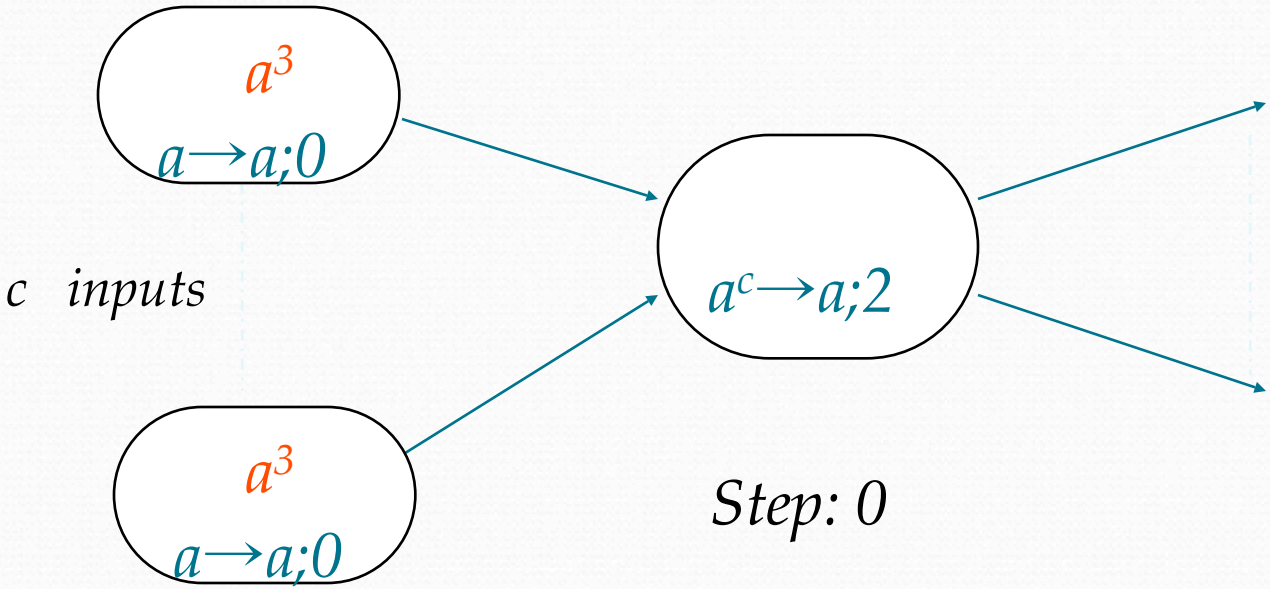
Spiking (*firing*) rules: $a^{c} \rightarrow a;d$



Step: 3

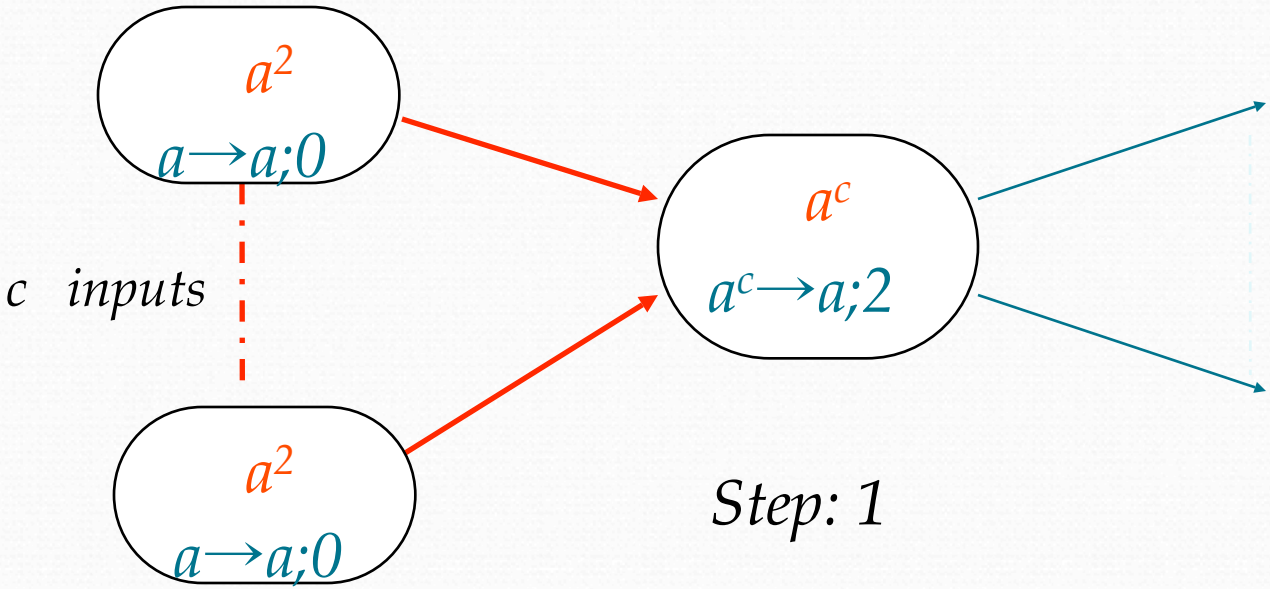
How the rules work

Spiking (*firing*) rules: $a^{c} \rightarrow a;d$



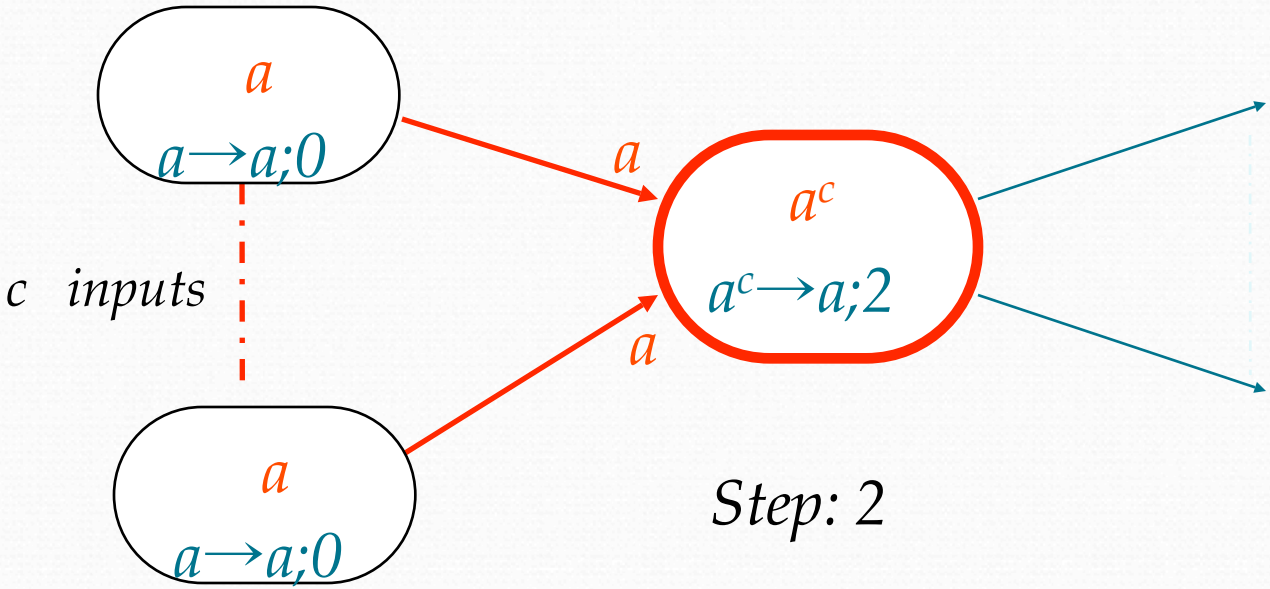
How the rules work

Spiking (*firing*) rules: $a^{c} \rightarrow a;d$



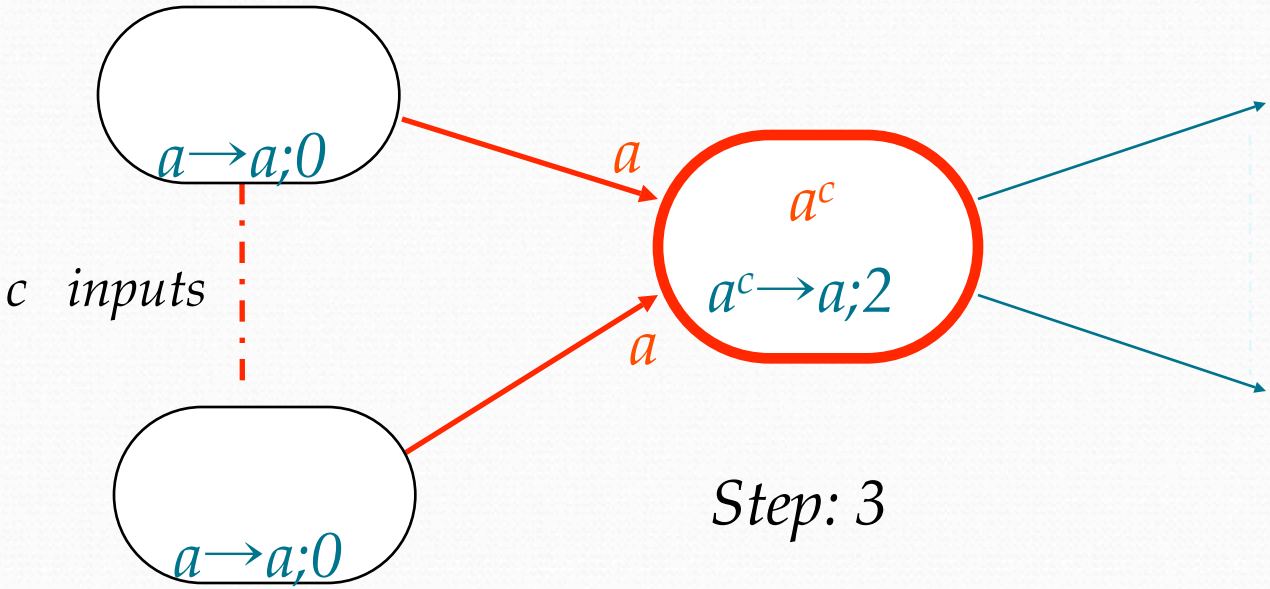
How the rules work

Spiking (*firing*) rules: $a^{c} \rightarrow a;d$



How the rules work

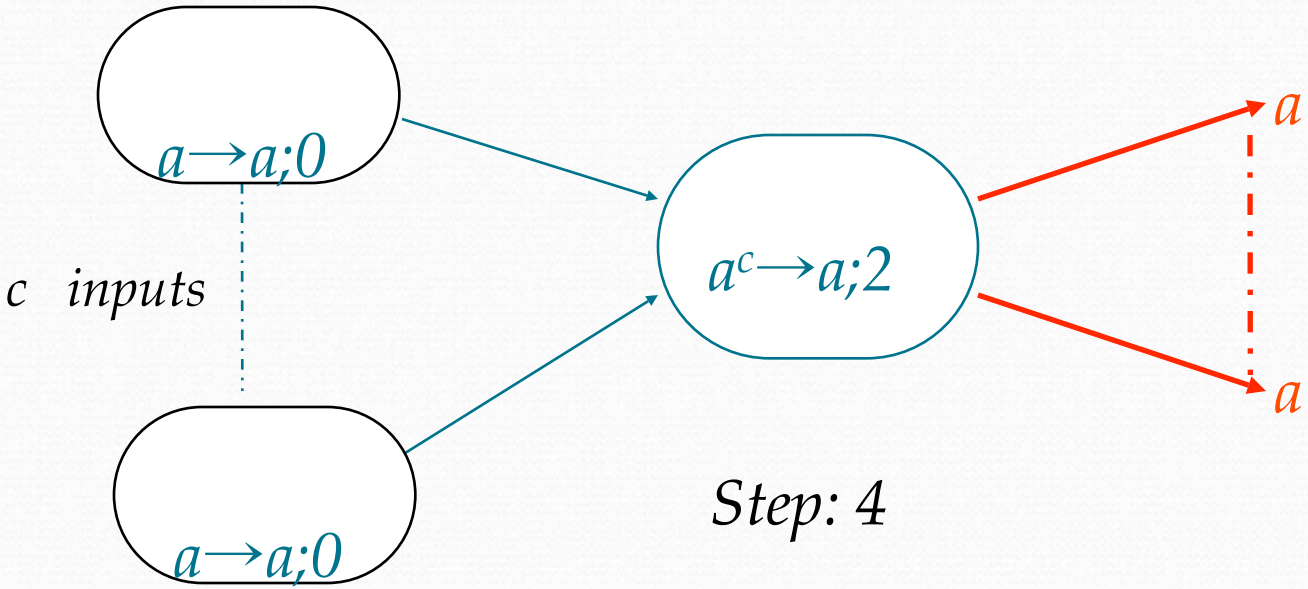
Spiking (*firing*) rules: $a^{c} \rightarrow a;d$



Step: 3

How the rules work

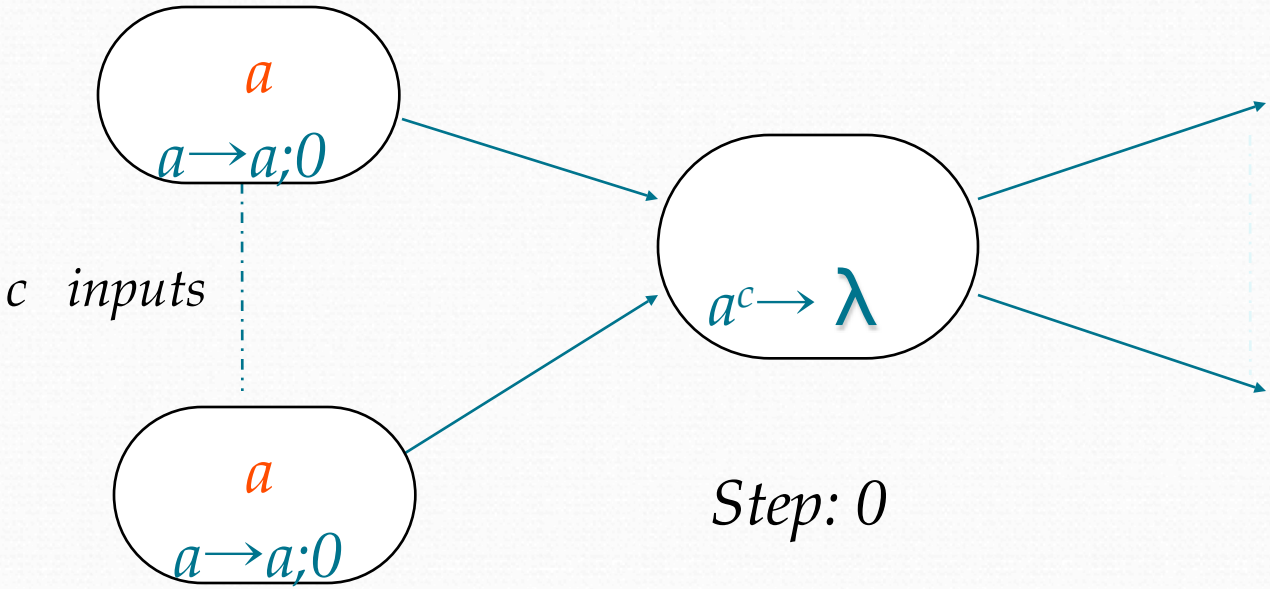
Spiking (*firing*) rules: $a^{c} \rightarrow a;d$



Step: 4

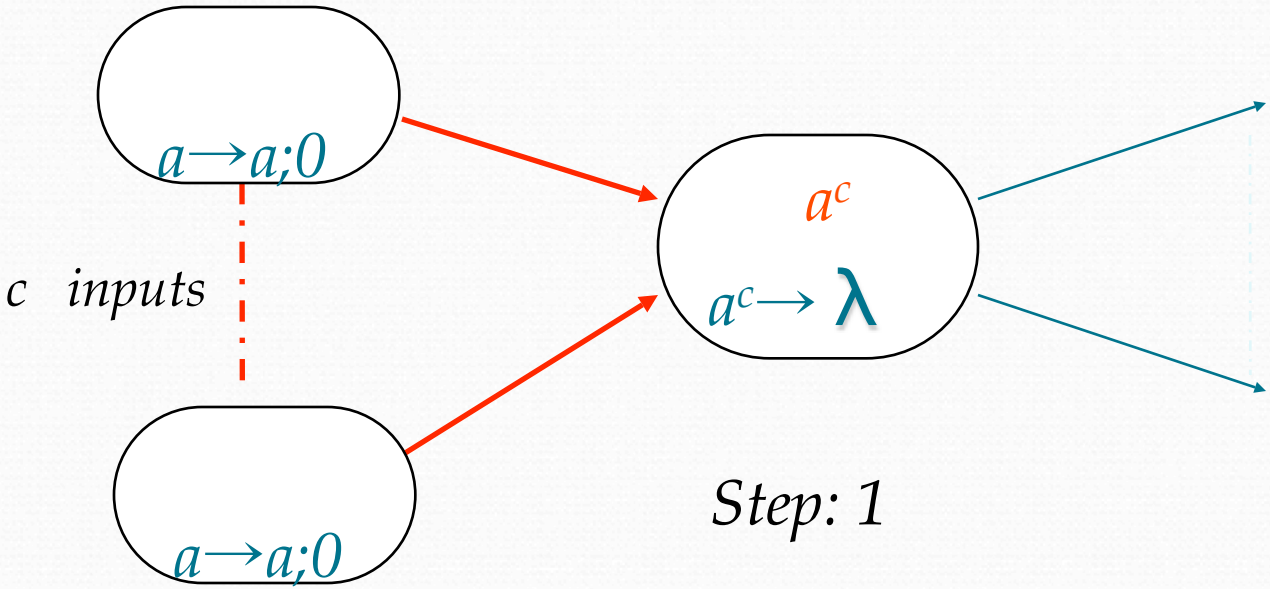
How the rules work

Forgetting rules: $a^c \rightarrow \lambda$



How the rules work

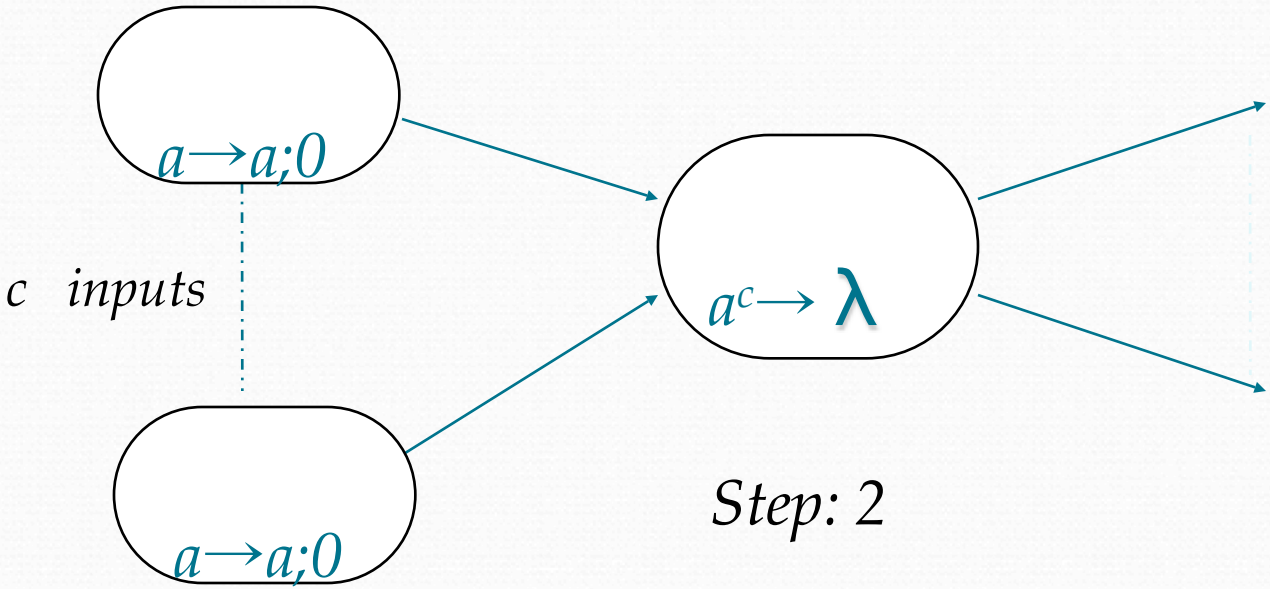
Forgetting rules: $a^c \rightarrow \lambda$



Step: 1

How the rules work

Forgetting rules: $a^c \rightarrow \lambda$



Step: 2

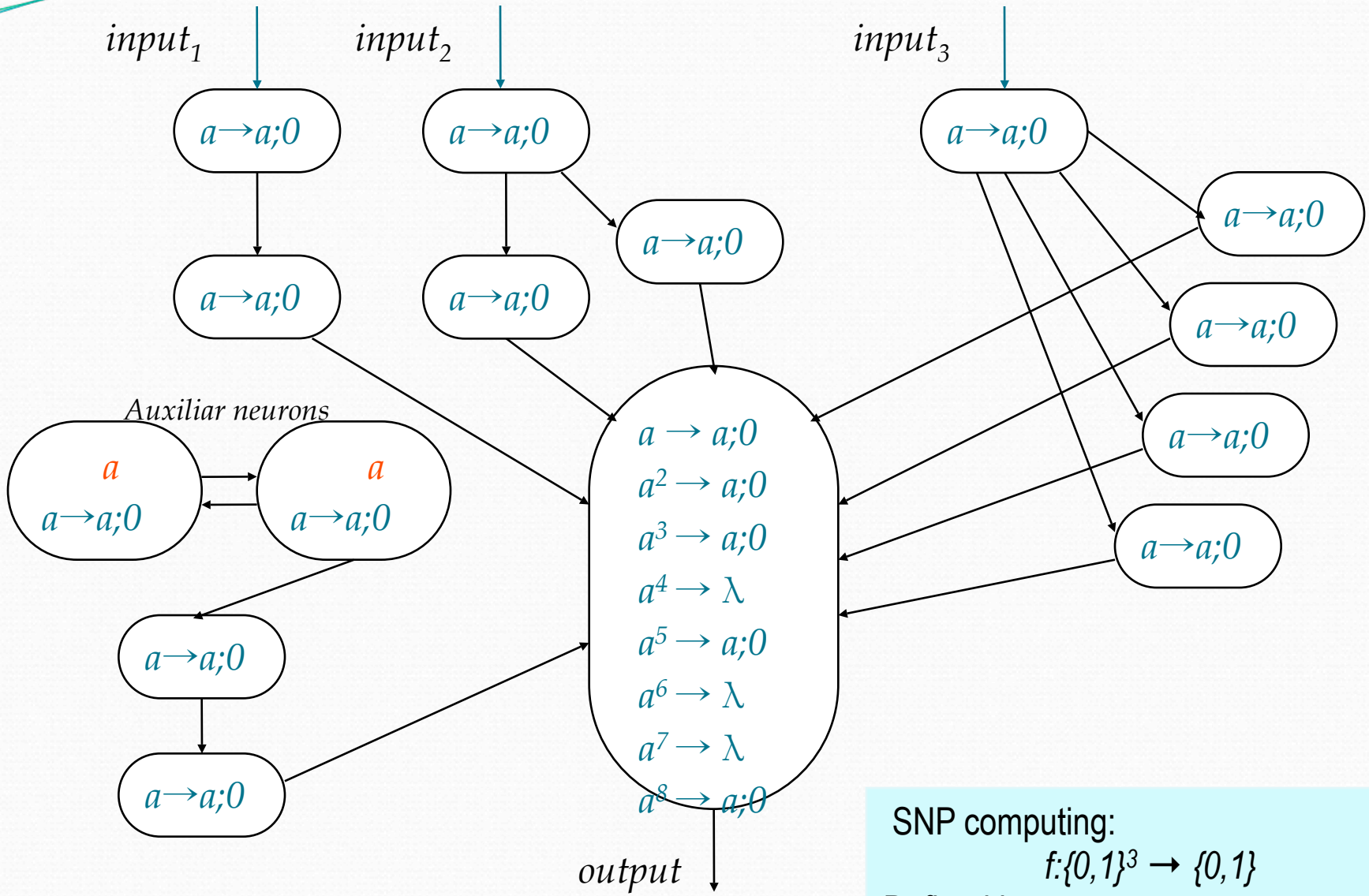
Some results

Theorem. Any Boolean function, *$f:\{0,1\}^k \rightarrow \{0,1\}$, can be computed by an SNP with k input neurons (also using further 2k+4 neurons, one being the output one).*

Theorem. The universality of *SNP can be obtained for systems that:*

1. *Do not use delays in the rules,*
2. *Do not use forgetting rules,*
3. *(1 ó 2) + the restriction of having a graph with output degree less or equal to 2.*
4. *(1 ó 2) + the restrcition of having a graph with input degree less or equal to 2.*

One example



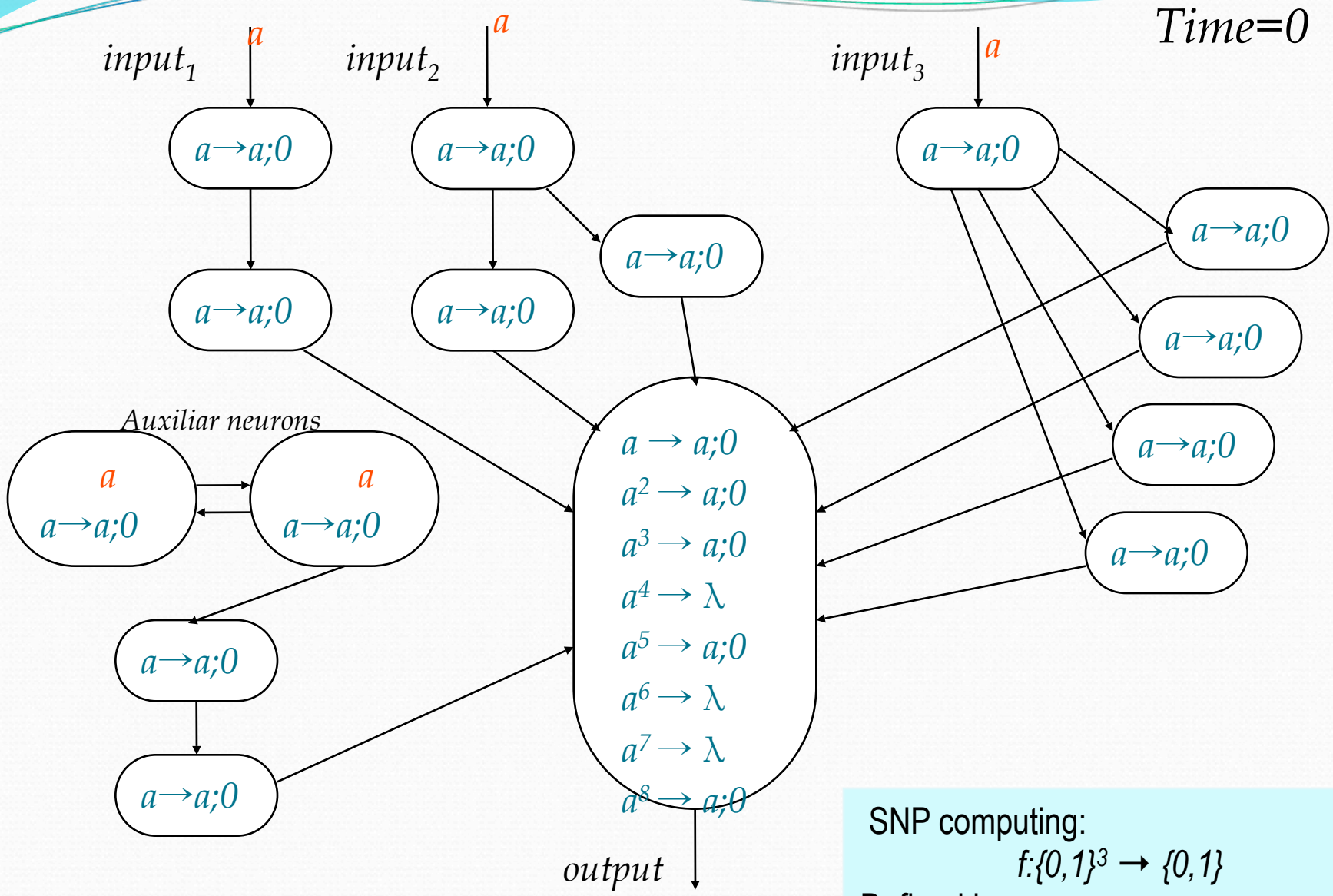
SNP computing:

$f:\{0,1\}^3 \rightarrow \{0,1\}$

Defined by:

$f(b_1,b_2,b_3)=1$ iff $b_1+b_2+b_3 \neq 2$

One example: Computing f(1,1,1)



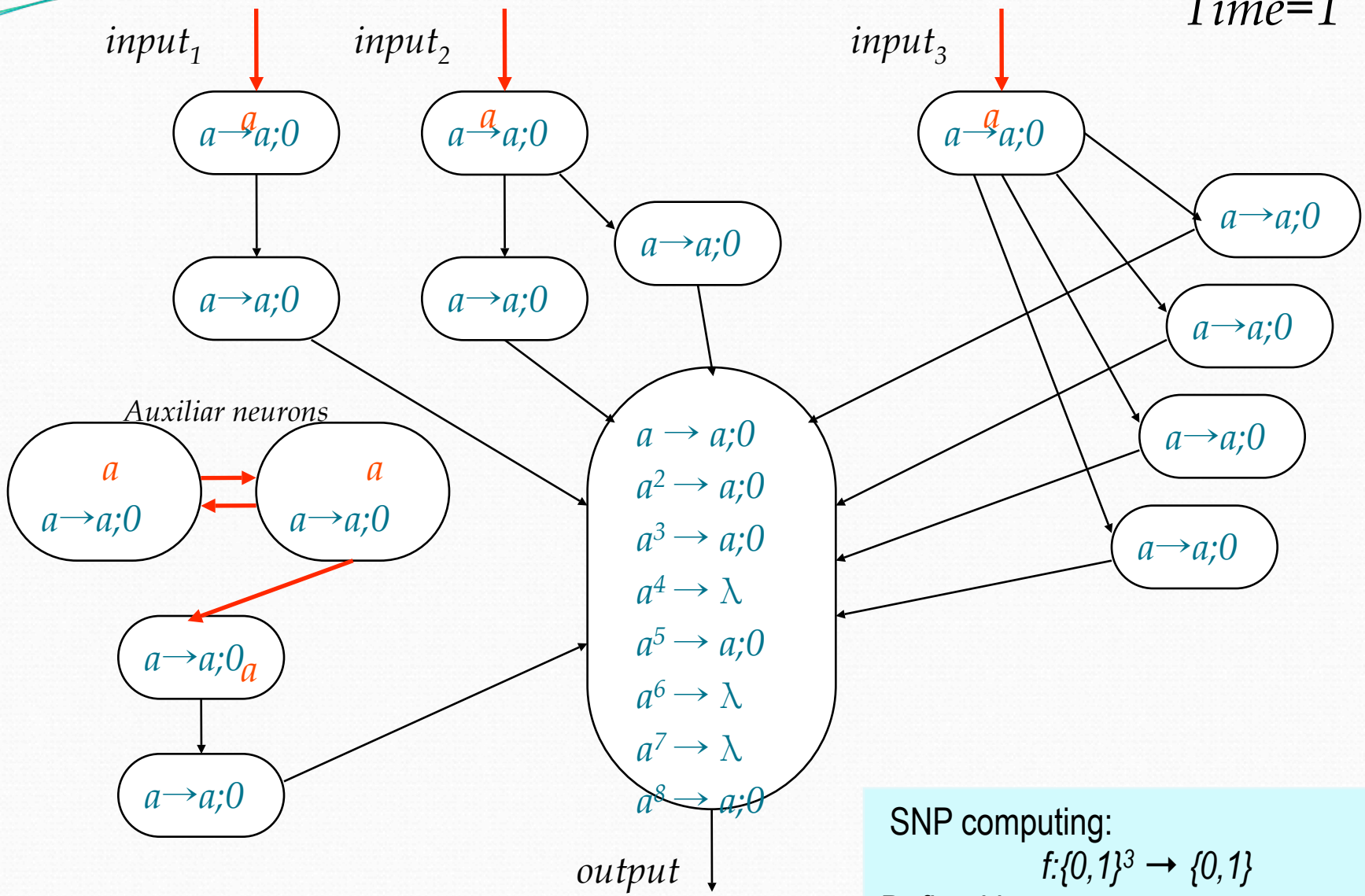
SNP computing:

$f:\{0,1\}^3 \rightarrow \{0,1\}$

Defined by:

$f(b_1,b_2,b_3)=1$ iff $b_1+b_2+b_3 \neq 2$

One example: Computing f(1,1,1)



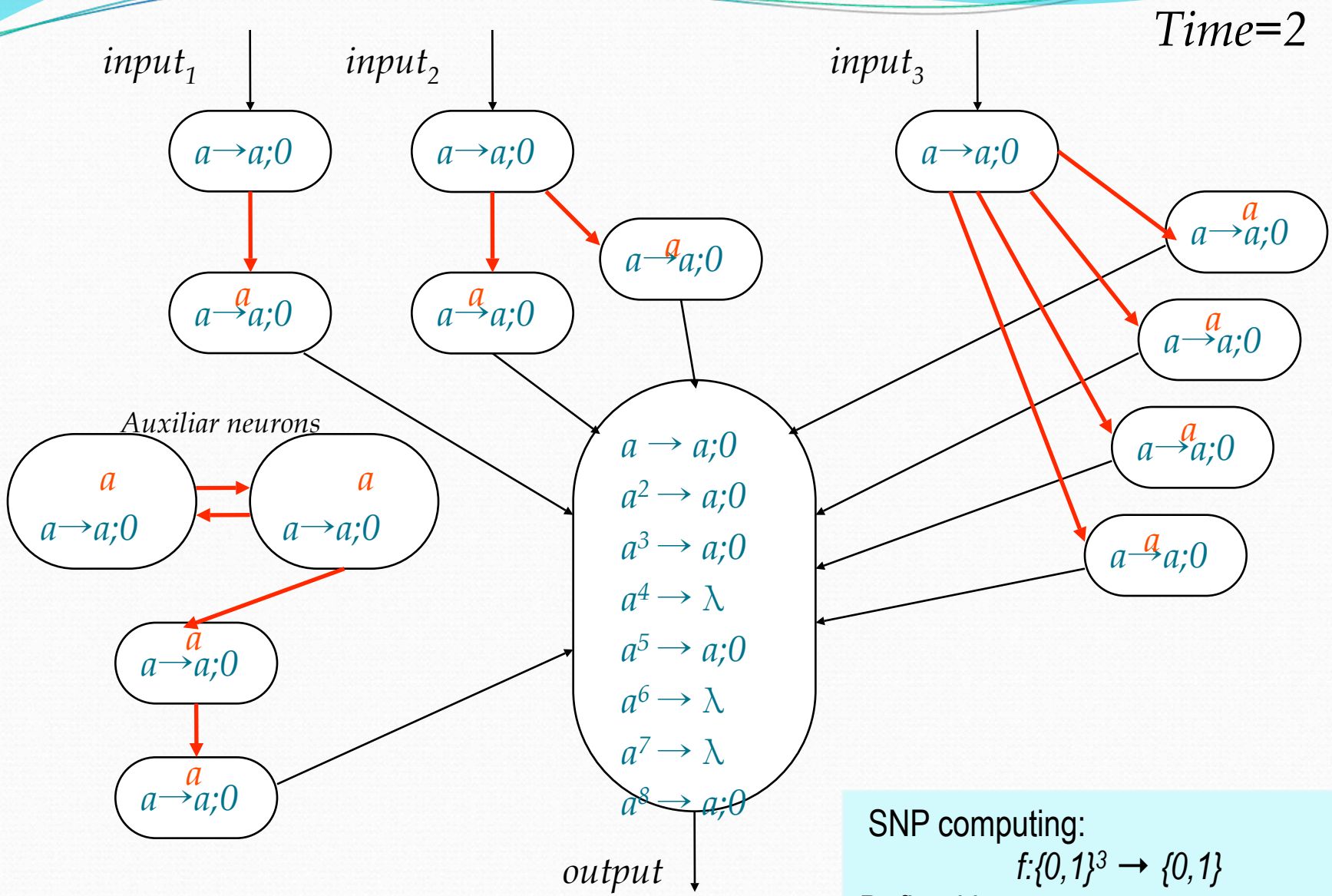
SNP computing:

$f:\{0,1\}^3 \rightarrow \{0,1\}$

Defined by:

$f(b_1,b_2,b_3)=1$ iff $b_1+b_2+b_3 \neq 2$

One example: Computing f(1,1,1)



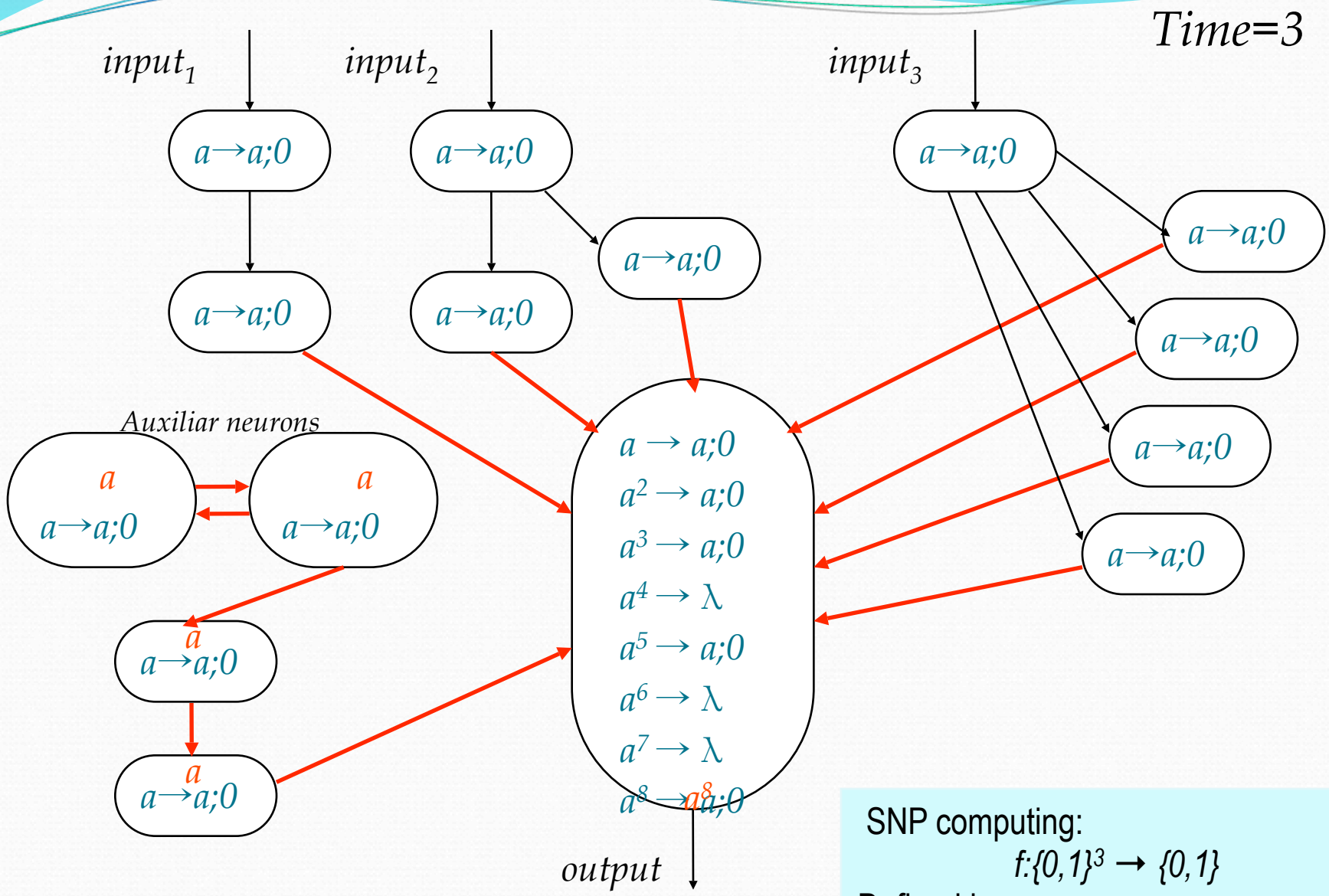
SNP computing:

$f:\{0,1\}^3 \rightarrow \{0,1\}$

Defined by:

$f(b_1,b_2,b_3)=1$ iff $b_1+b_2+b_3 \neq 2$

One example: Computing f(1,1,1)



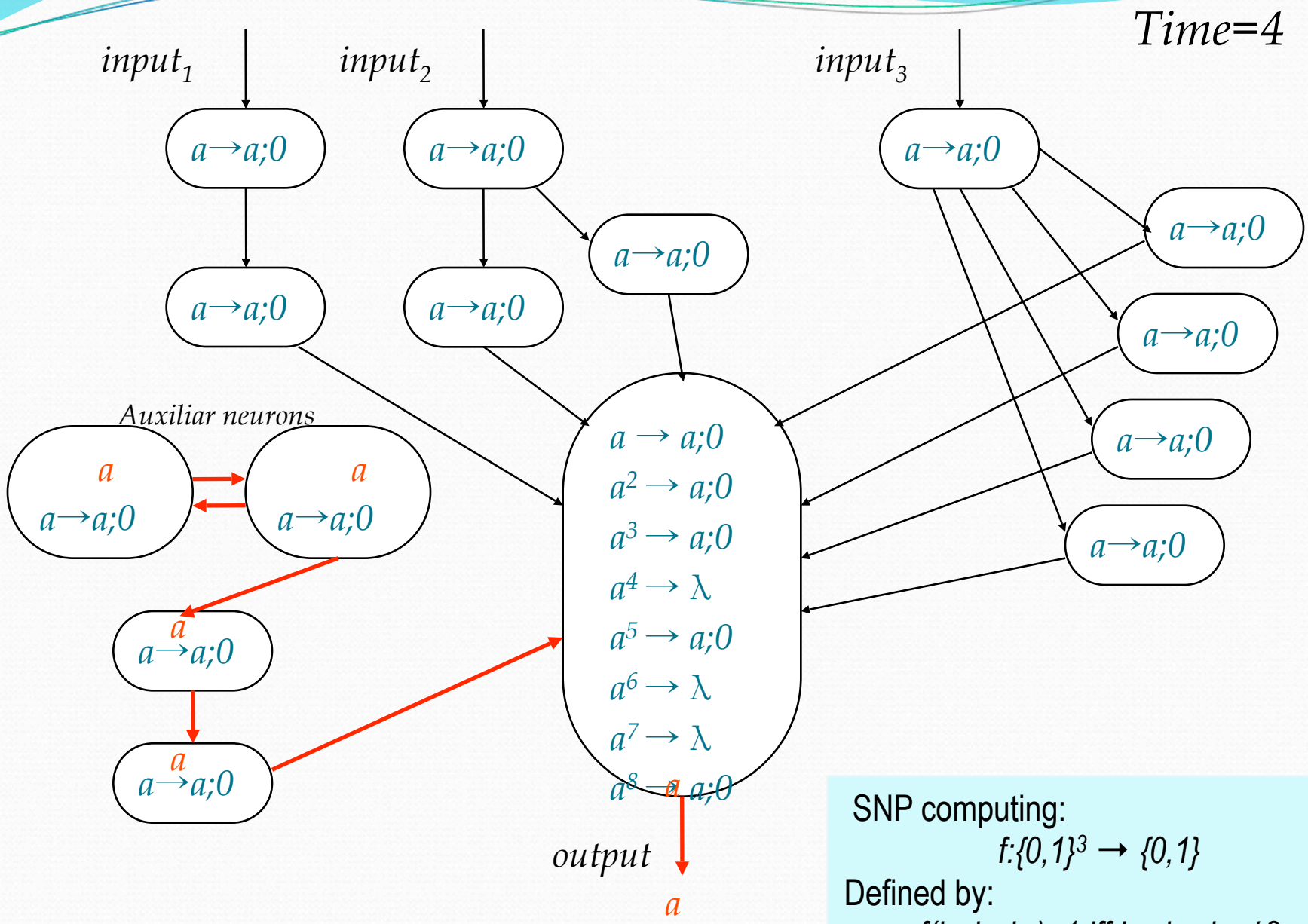
SNP computing:

$f:\{0,1\}^3 \rightarrow \{0,1\}$

Defined by:

$f(b_1,b_2,b_3)=1$ *iff* $b_1+b_2+b_3 \neq 2$

One example: Computing f(1,1,1)



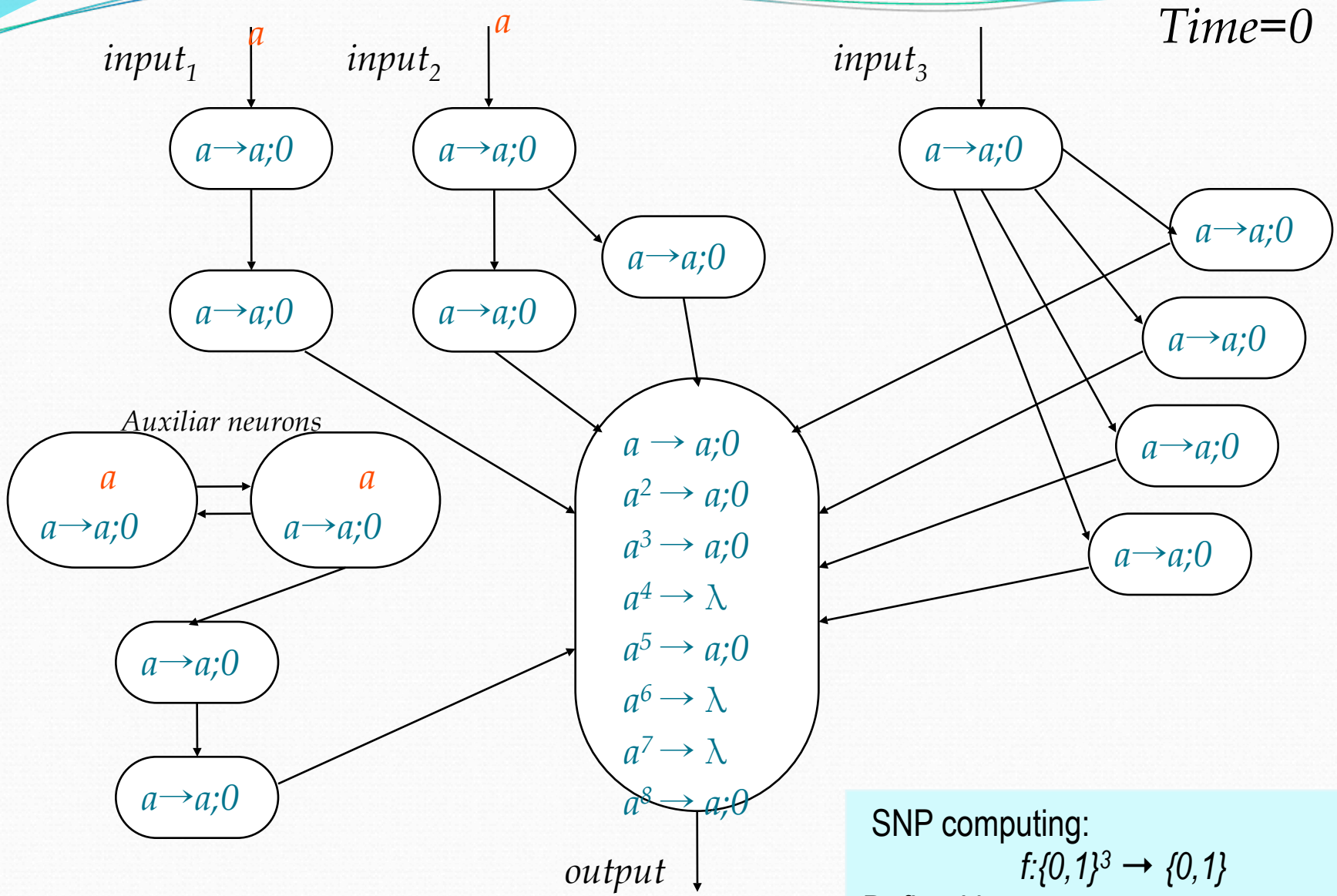
SNP computing:

$f:\{0,1\}^3 \rightarrow \{0,1\}$

Defined by:

$f(b_1,b_2,b_3)=1$ iff $b_1+b_2+b_3 \neq 2$

One example: Computing f(1,1,0)



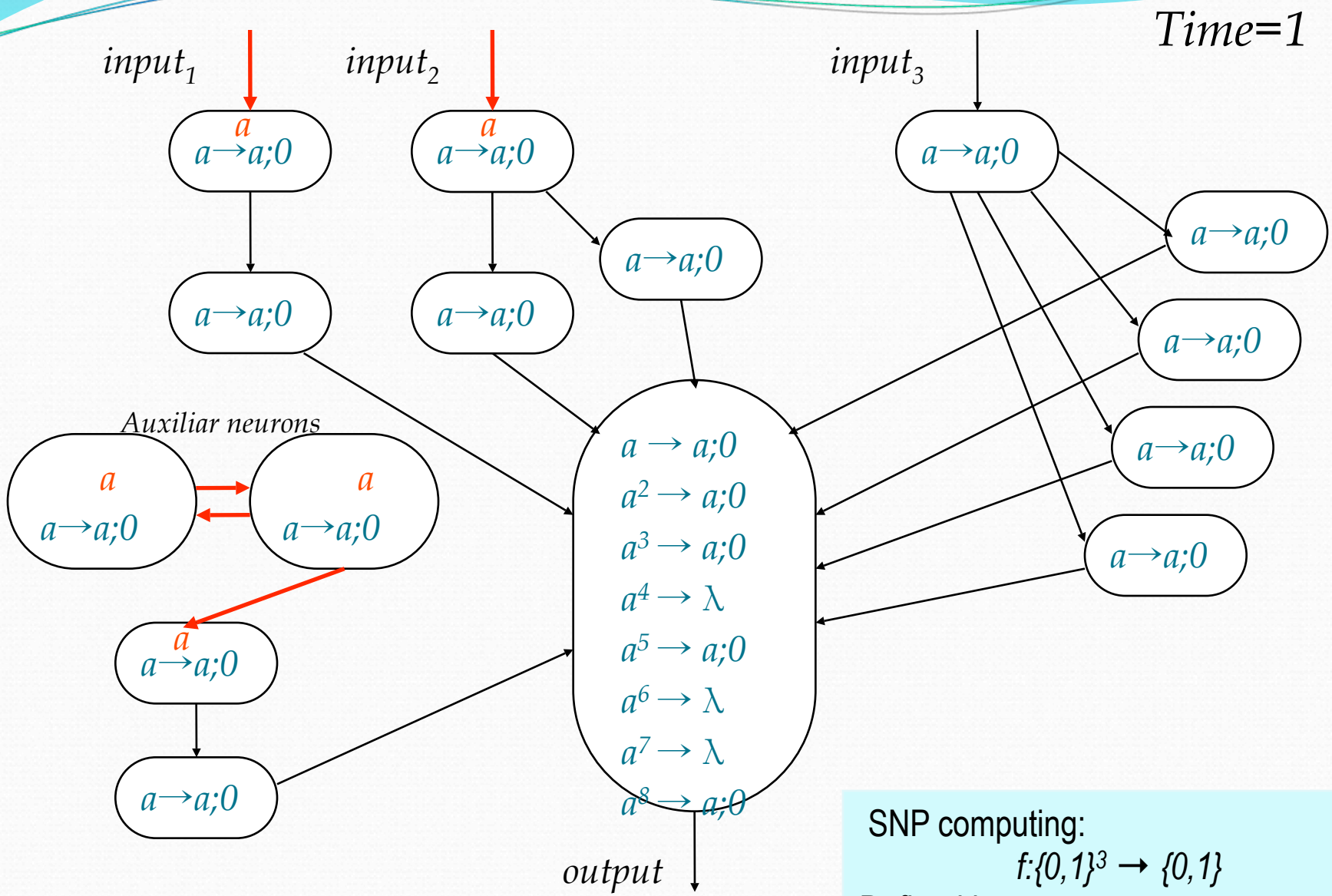
SNP computing:

$f:\{0,1\}^3 \rightarrow \{0,1\}$

Defined by:

$f(b_1,b_2,b_3)=1$ iff $b_1+b_2+b_3 \neq 2$

One example: Computing f(1,1,0)



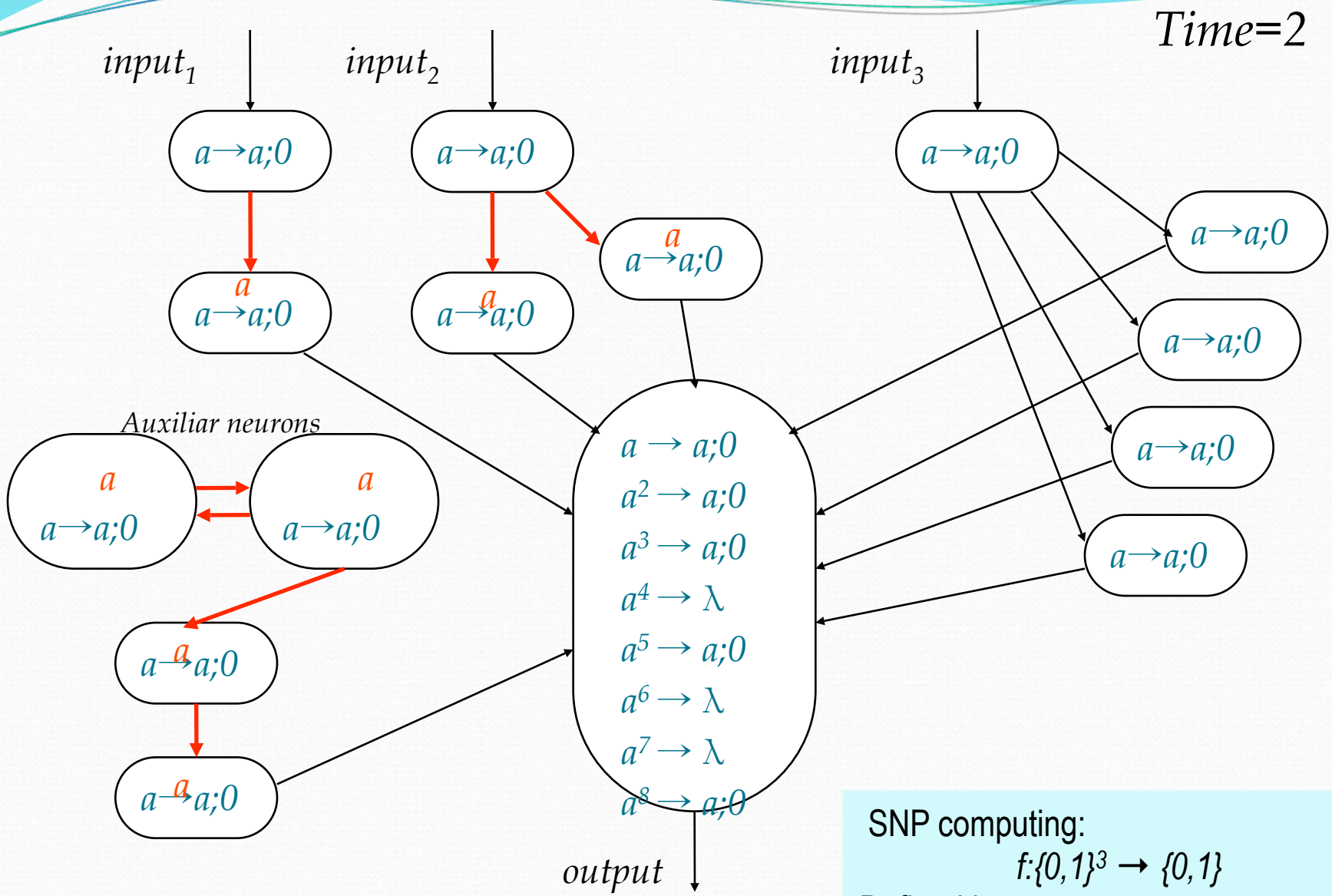
SNP computing:

$f:\{0,1\}^3 \rightarrow \{0,1\}$

Defined by:

$f(b_1,b_2,b_3)=1$ iff $b_1+b_2+b_3 \neq 2$

One example: Computing f(1,1,0)



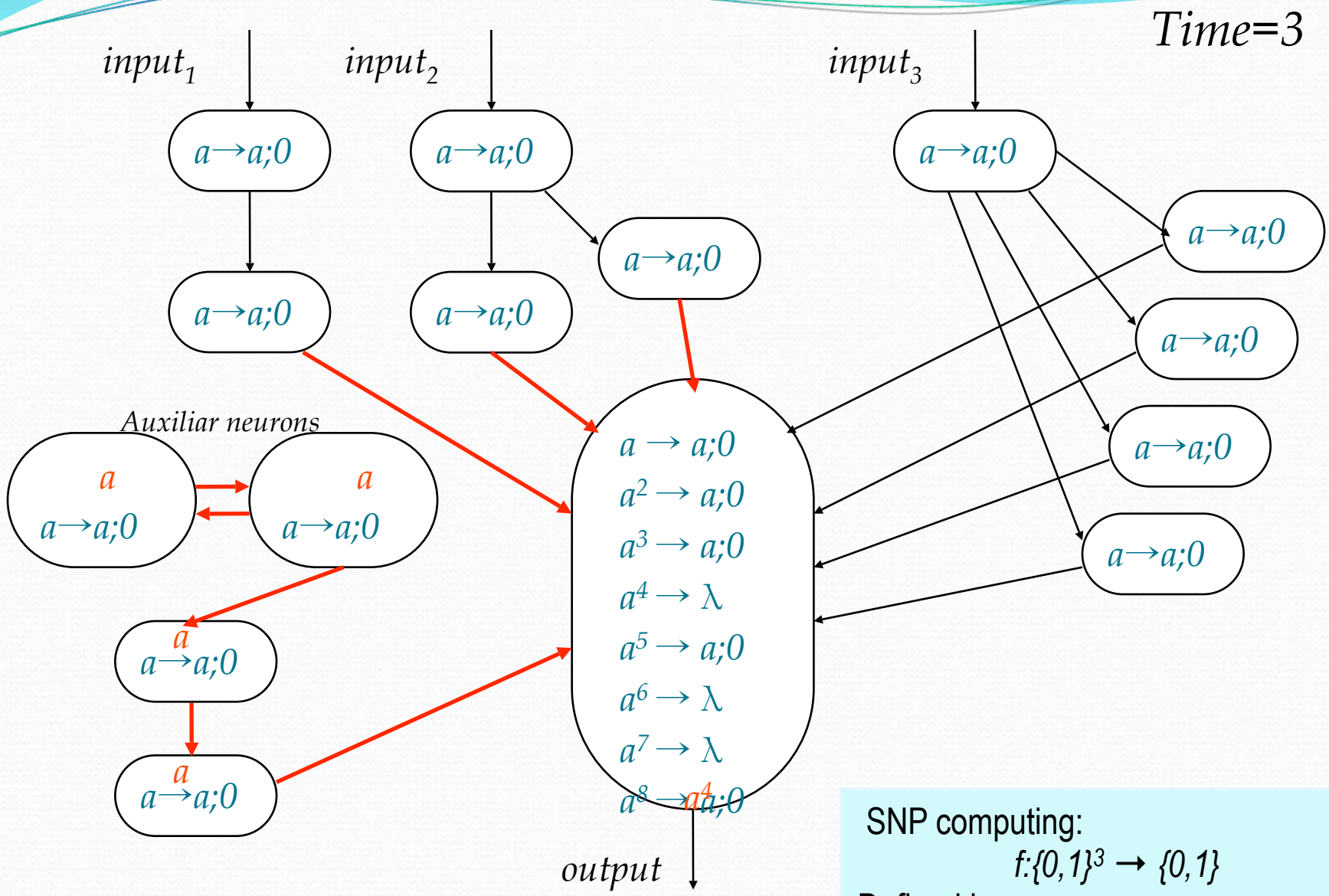
SNP computing:

$f:\{0,1\}^3 \rightarrow \{0,1\}$

Defined by:

$f(b_1,b_2,b_3)=1$ iff $b_1+b_2+b_3 \neq 2$

One example: Computing f(1,1,0)



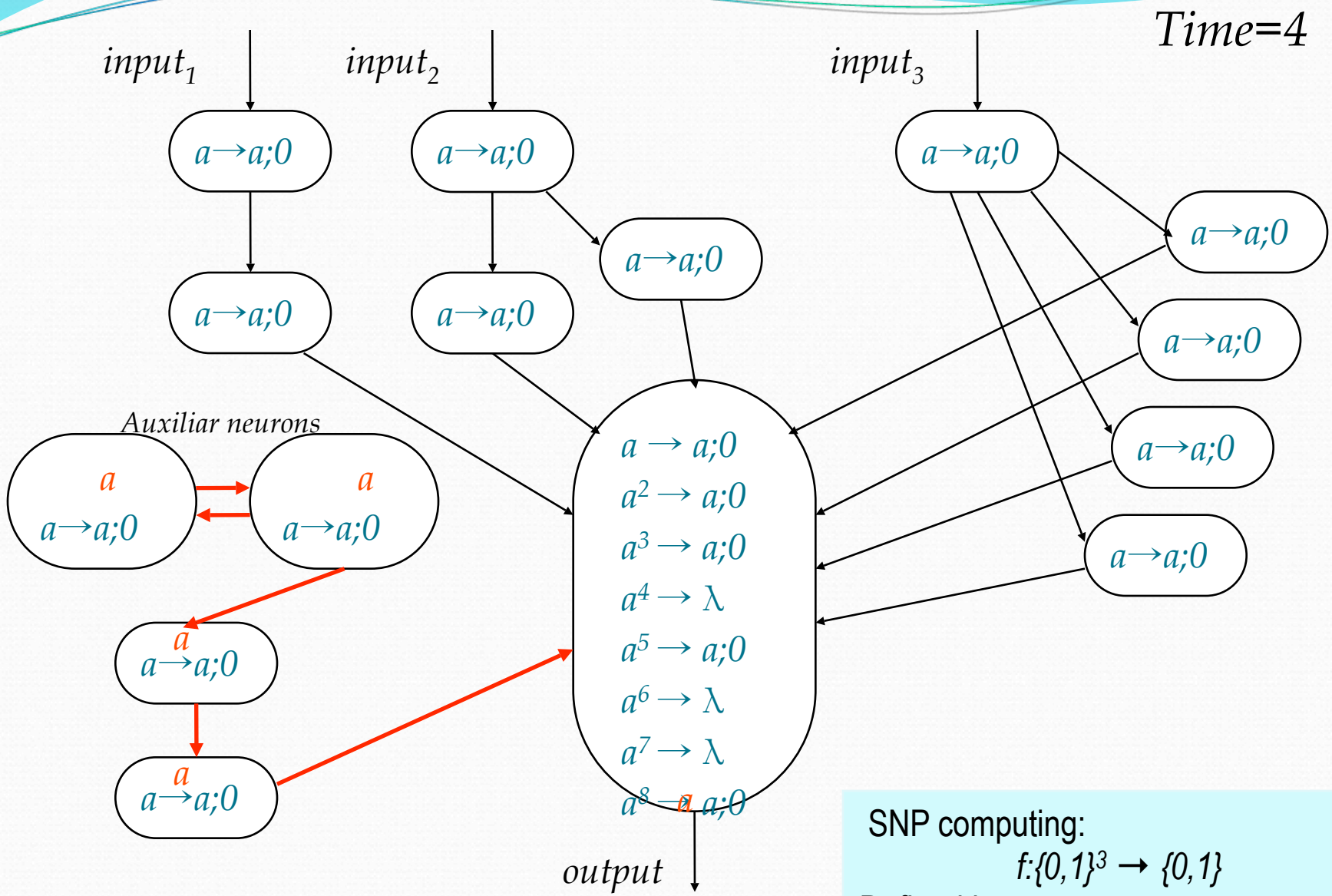
SNP computing:

$f:\{0,1\}^3 \rightarrow \{0,1\}$

Defined by:

$f(b_1,b_2,b_3)=1$ iff $b_1+b_2+b_3 \neq 2$

One example: Computing f(1,1,0)



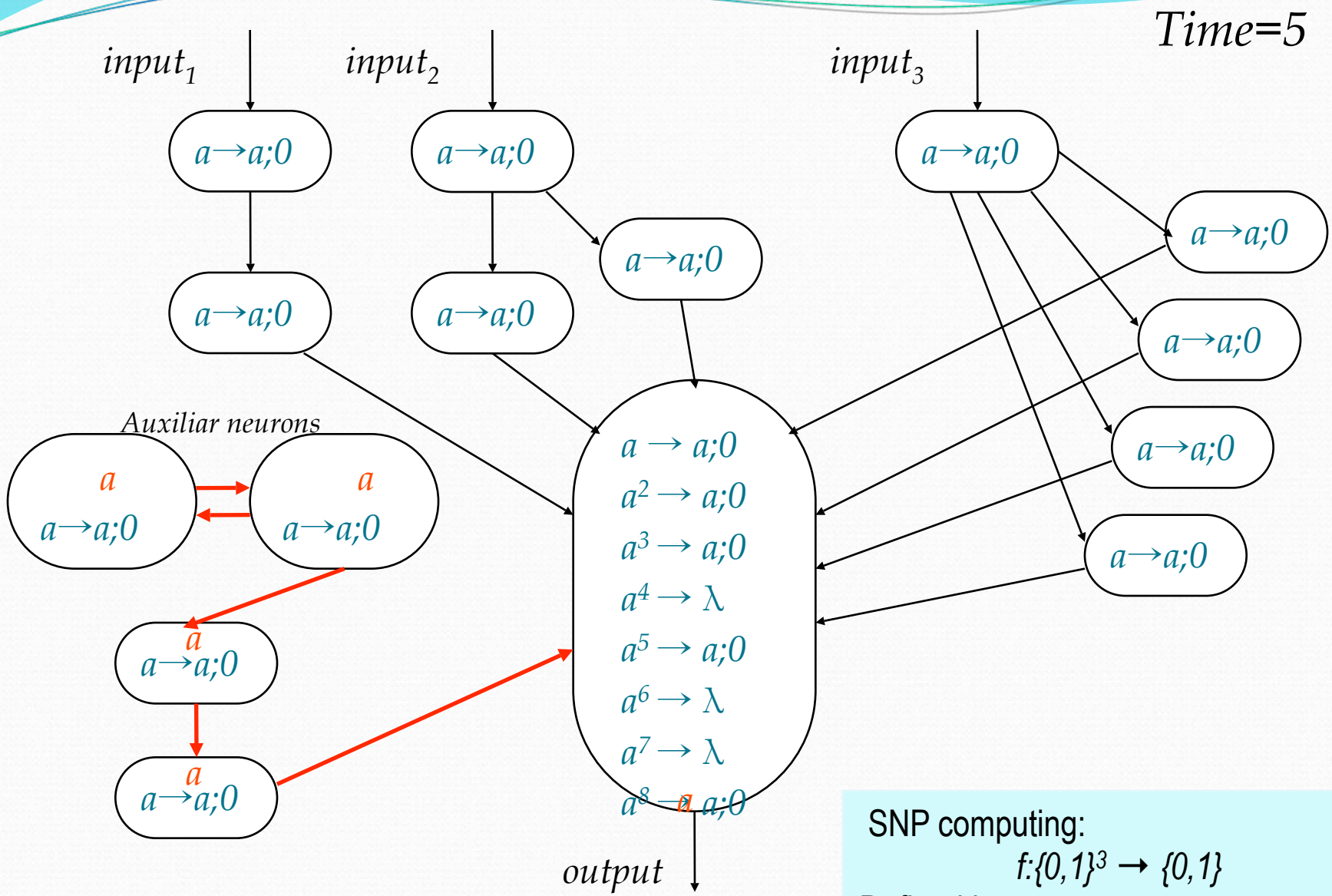
SNP computing:

$f:\{0,1\}^3 \rightarrow \{0,1\}$

Defined by:

$f(b_1,b_2,b_3)=1$ iff $b_1+b_2+b_3 \neq 2$

One example: Computing f(1,1,0)



SNP computing:

$f:\{0,1\}^3 \rightarrow \{0,1\}$

Defined by:

$f(b_1,b_2,b_3)=1$ iff $b_1+b_2+b_3 \neq 2$

Conclusions

Conclusions

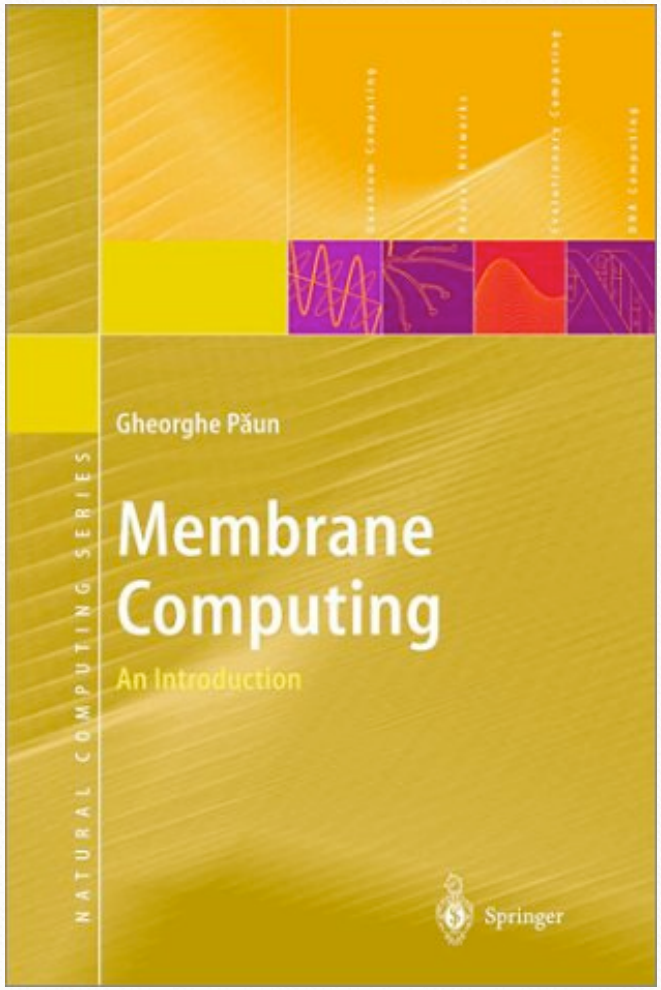
- Membrane Computing provides computational models that abstract from the living cells structure and functioning.
- Such models have been proved to be computationally powerful (equiv. to T.M.) and efficient (solving NP-Complete problems in polynomial time).
- Membrane Computing defines an abstract framework for reasoning about:
  - distribute architectures
  - communication
  - parallel information processing
- Such features are relevant both for Computer Science (Distributed Computing Models, Multi-Agent Systems) and Biology (Modeling and Simulation of Biological Systems).

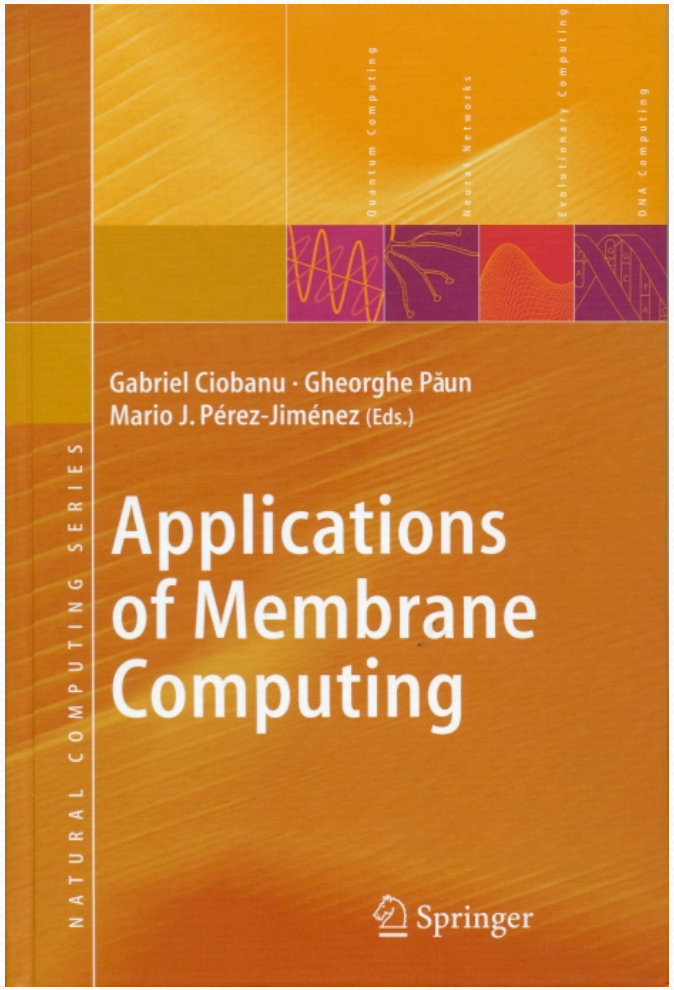
Conclusions

- Hard to make real implementations: some attempts were made, and some are in development now.
- Non-Determinism and Maximal Parallelism are not always desirable features and some variants try to “control” them.

References

Main References: books





Main References

Web Page:

- The P systems Web Page: http://psystems.disco.unimib.it
  Where you can find a lot of papers, reports, software, etc.

Conferences/Workshops (every year):

- (BWMC) Brainstorming Week on Membrane Computing.
- (WMC) Workshop on Membrane Computing.
- (BIC-TA) International Conference on Bio-Inspired Computing: Theory and Applications.

Thank you…