Bio-inspired computation, communication topologies, and computational complexity

Antonio E. Porreca
https://aeporreca.org

Outline

- The first and second machine classes
- Examples of parallel computing models
- Membrane systems
- Complexity theory of membrane systems
- Communication topologies and their role
- A research project

The first machine class

Definition (van Emde Boas). A machine model is first class iff it simulates and is simulated by a Turing machine in polynomial time

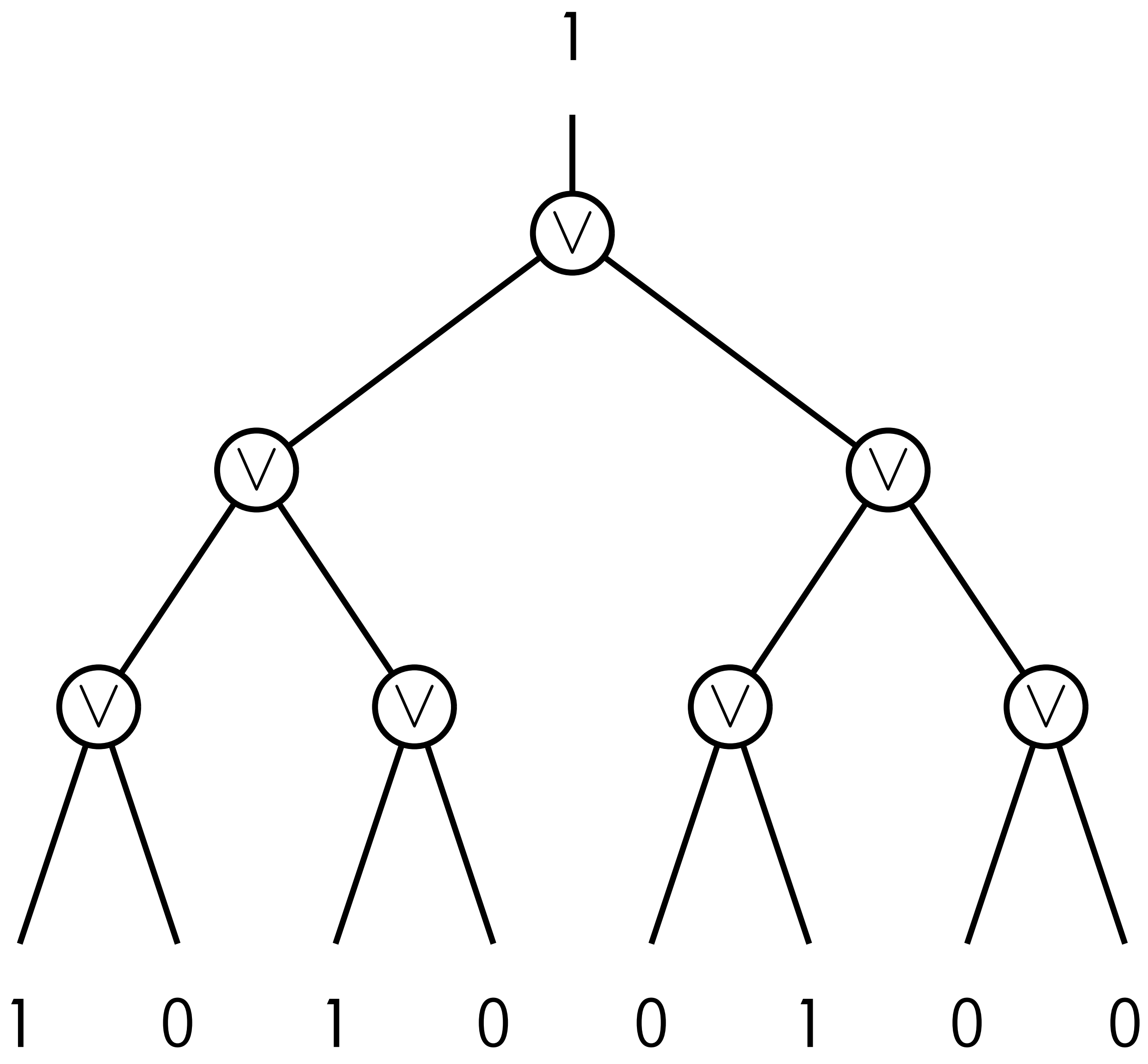
- Turing machines
- Random access machines with $+$ and $-$
- Cellular automata with finite initial configuration

The second machine class

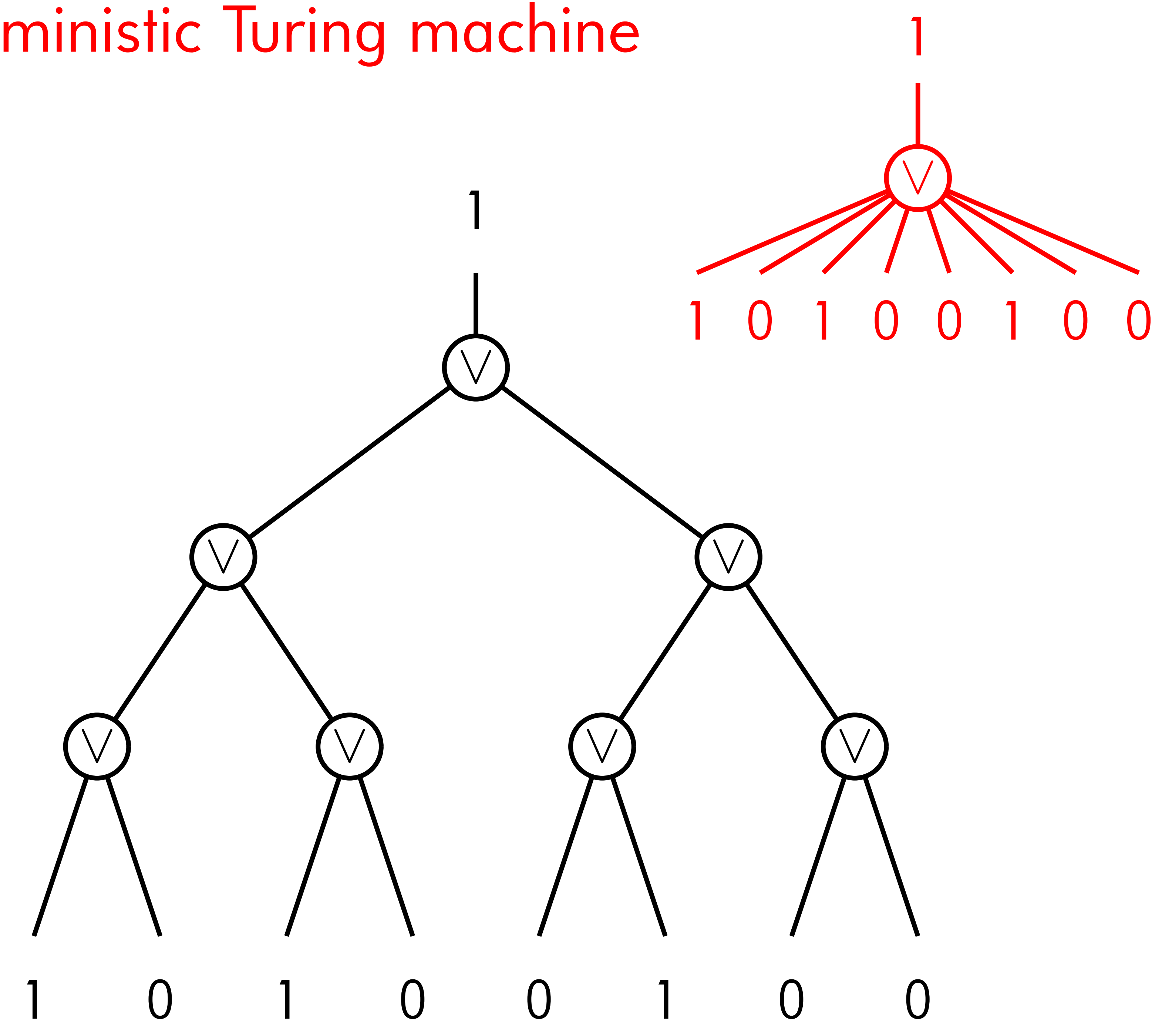
Definition (van Emde Boas). A machine model is second class iff it characterises **PSPACE** in polynomial time (deterministically and nondeterministically)

- Alternating Turing machines
- Vector machines
- Random access machines with $+ - \times \div$
- Parallel processes with `fork()` with unbounded number of processors
- Cellular automata over hyperbolic grids

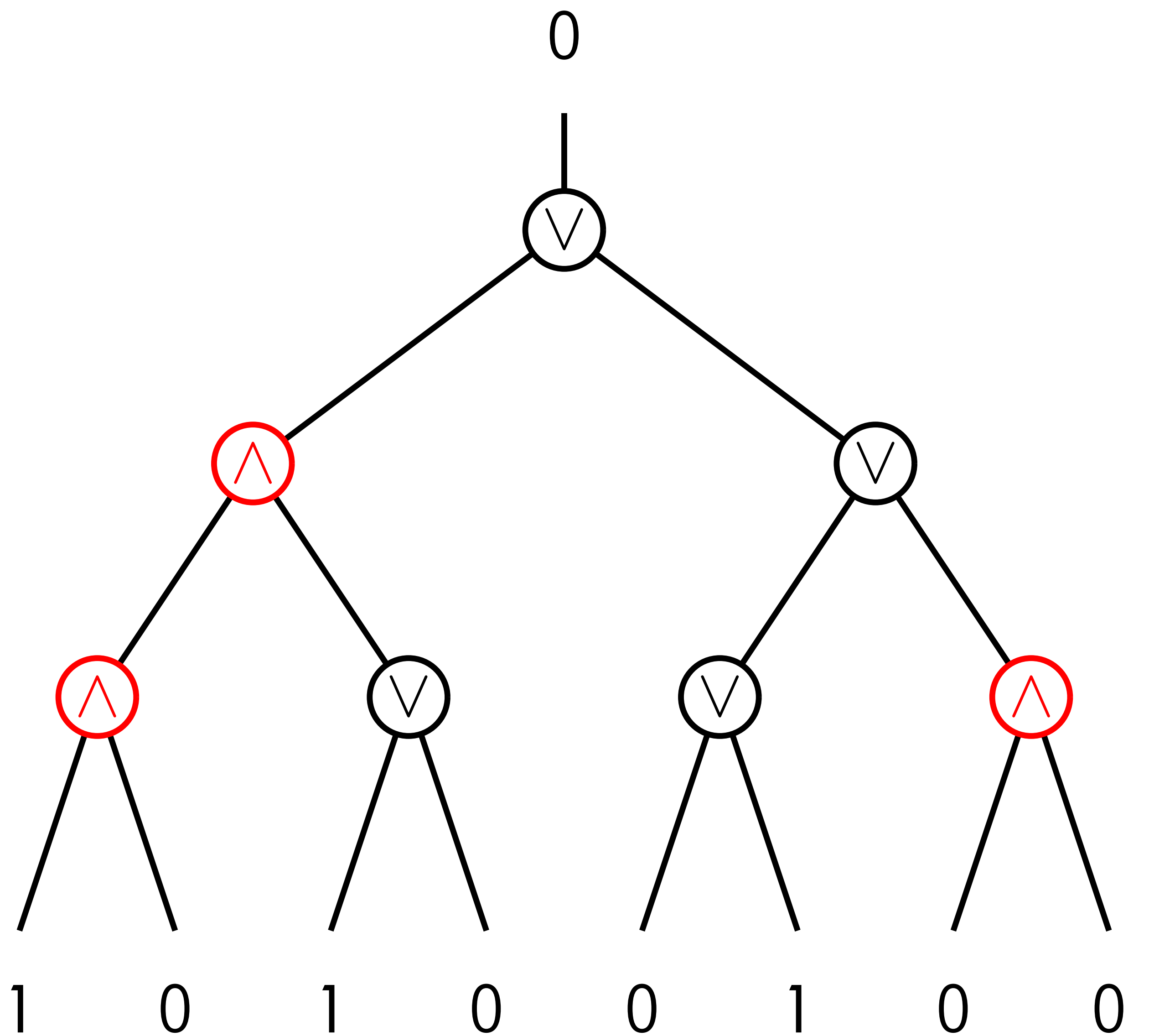
Nondeterministic Turing machine



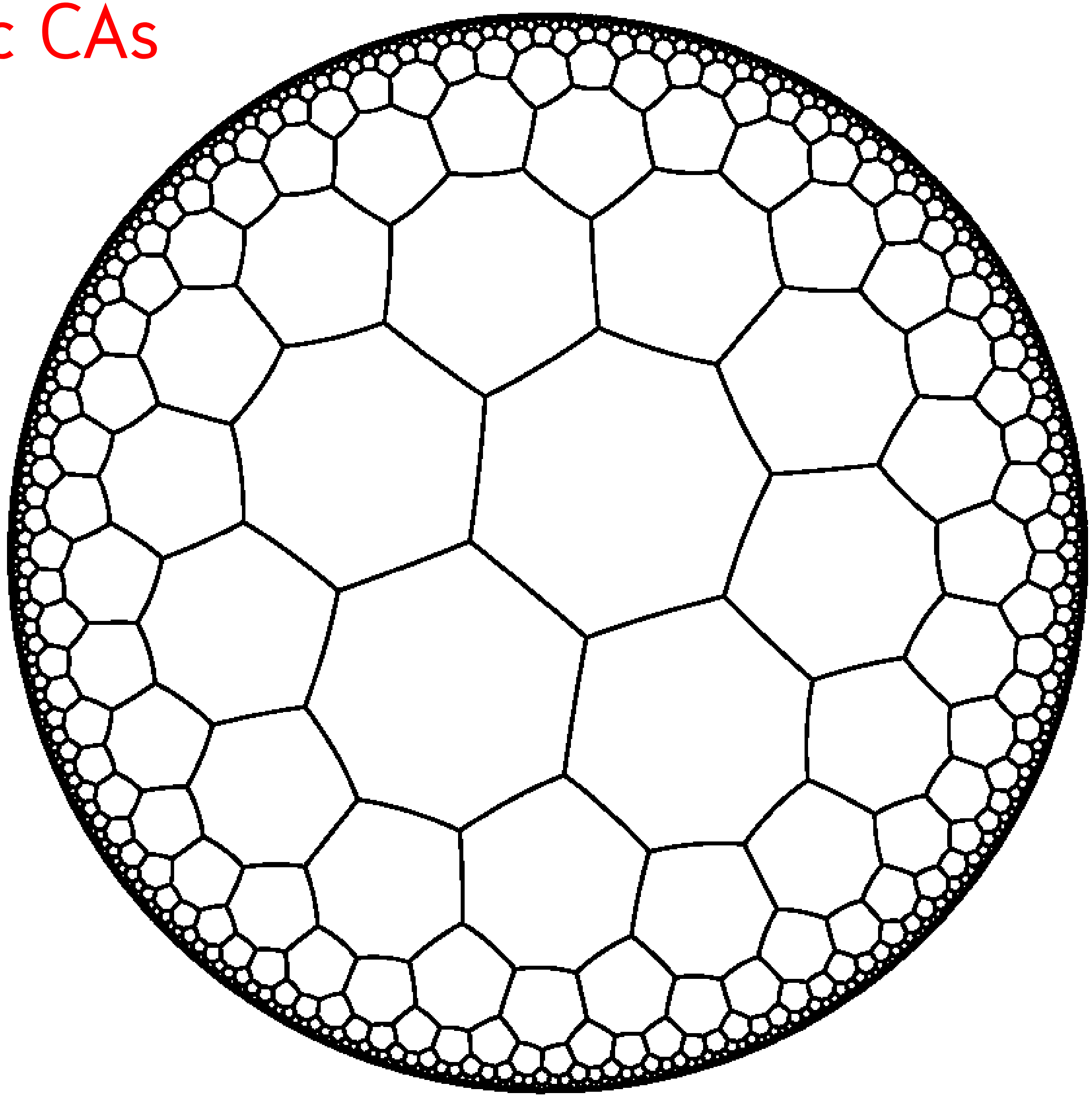
Nondeterministic Turing machine



Alternating Turing machine

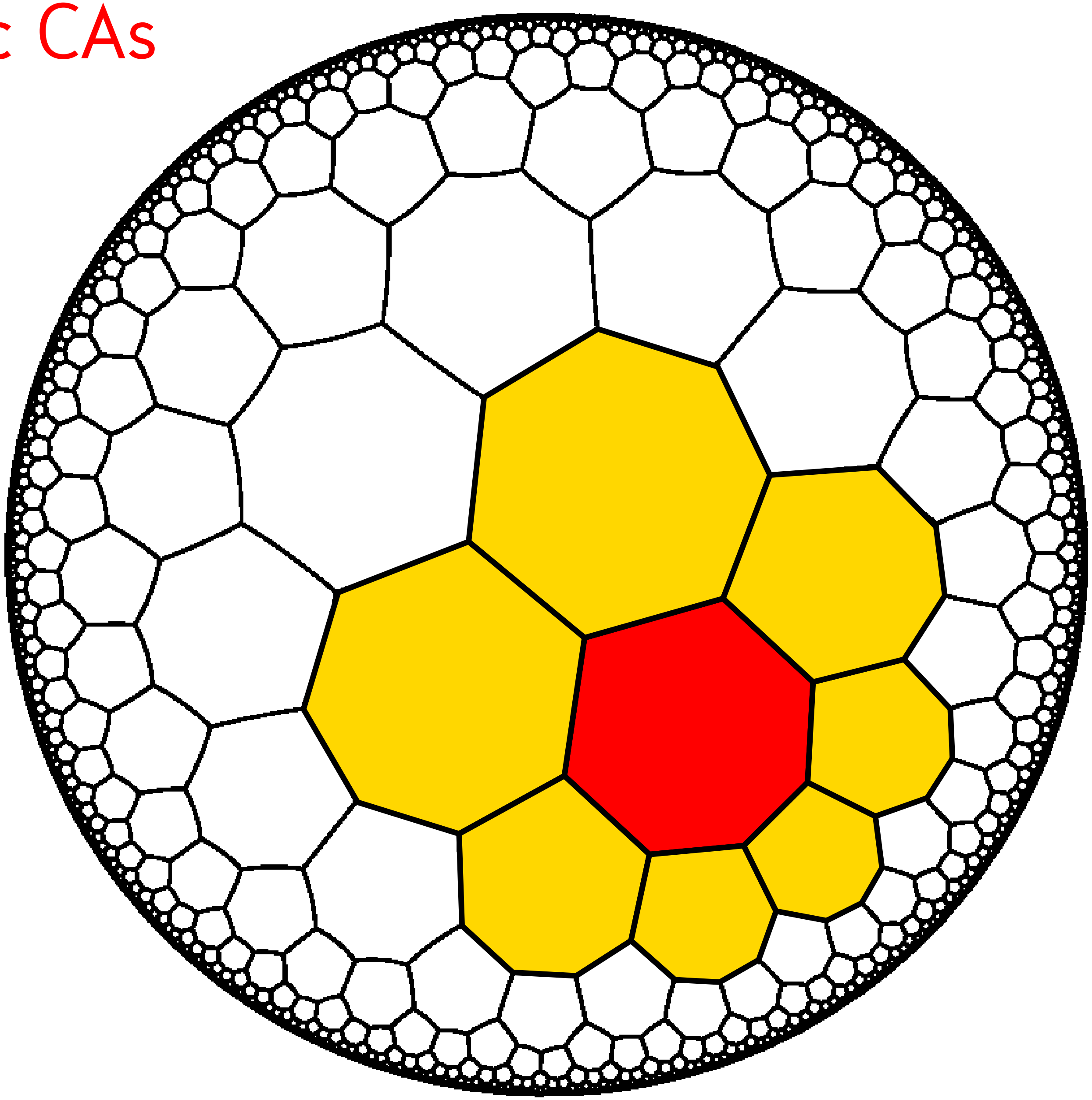


Hyperbolic CAs



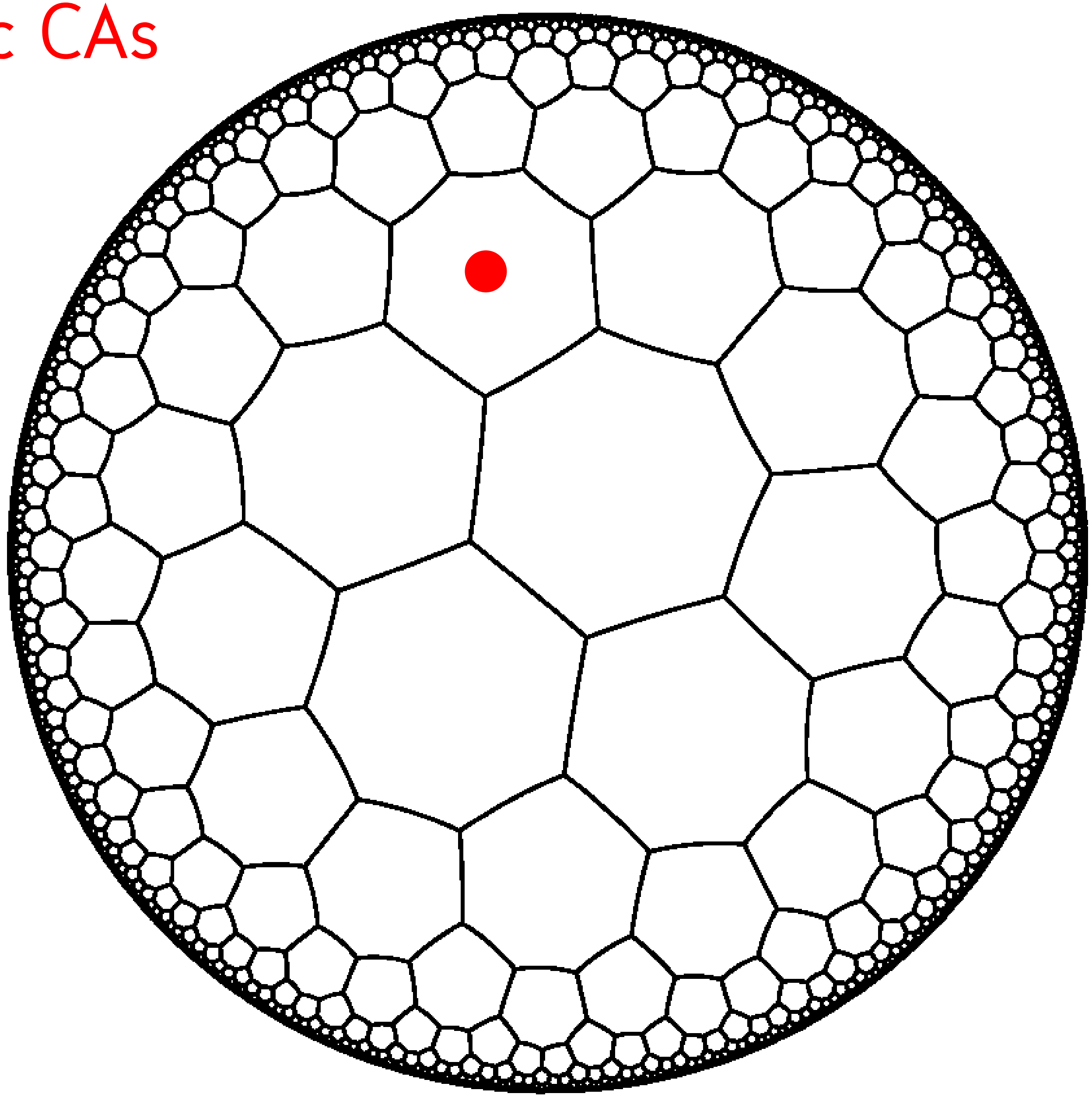
Hyperbolic tiling by Theon, used under CC BY-SA 3.0
`https://en.wikipedia.org/wiki/File:PavageHypPoincare2.svg`

Hyperbolic CAs



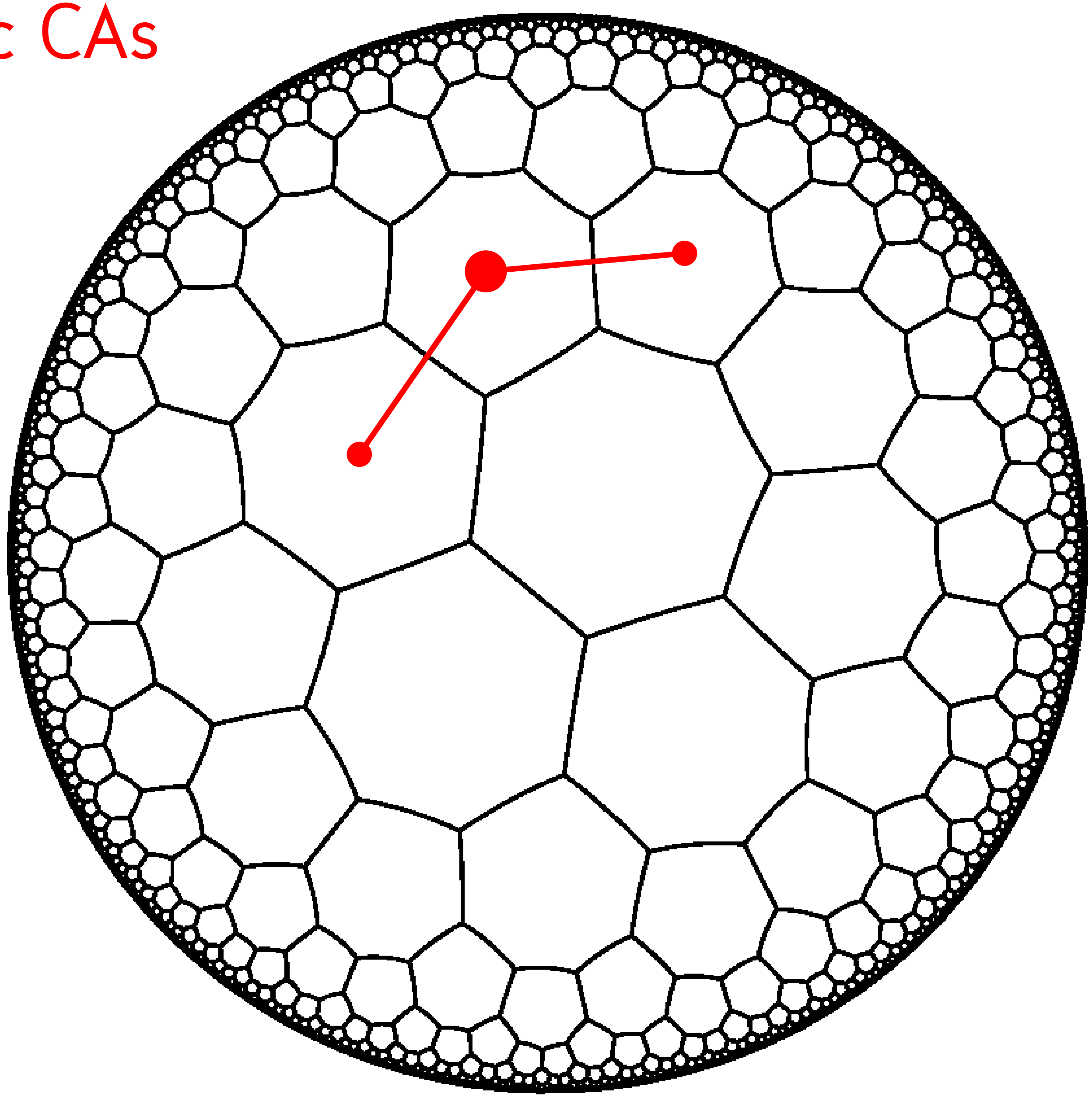
Hyperbolic tiling by Theon, used under CC BY-SA 3.0
`https://en.wikipedia.org/wiki/File:PavageHypPoincare2.svg`

Hyperbolic CAs



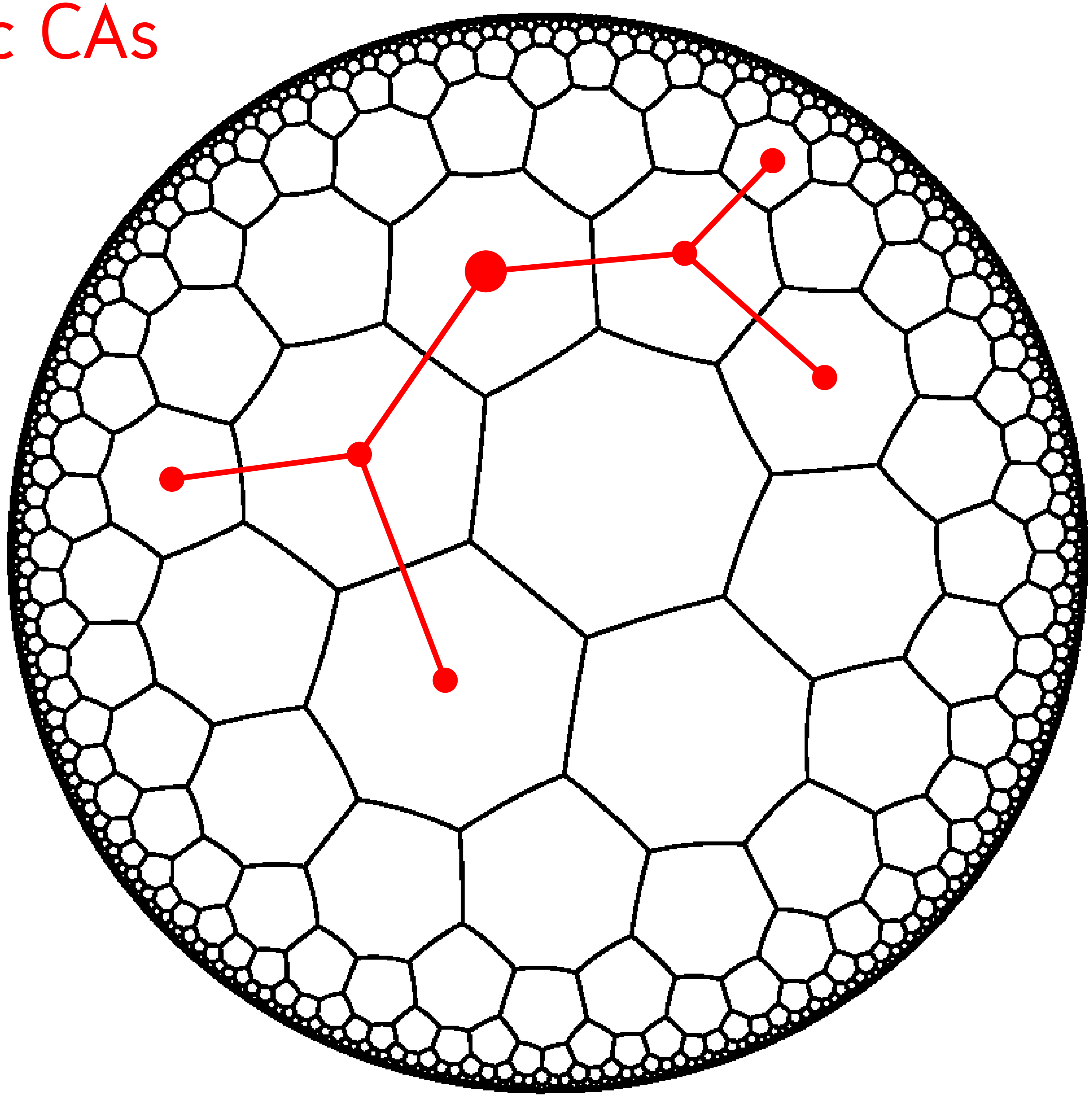
Hyperbolic tiling by Theon, used under CC BY-SA 3.0
`https://en.wikipedia.org/wiki/File:PavageHypPoincare2.svg`

Hyperbolic CAs



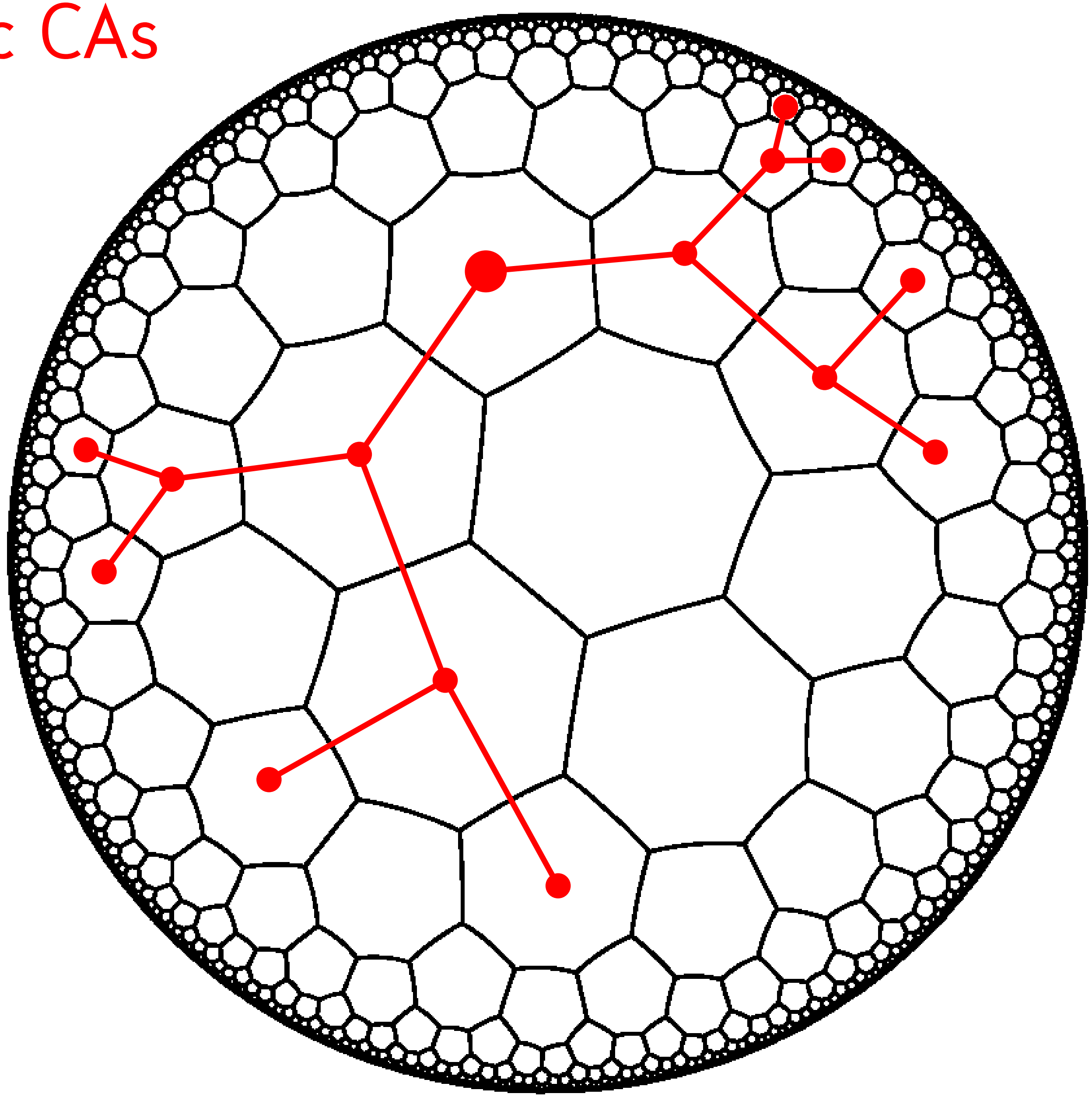
Hyperbolic tiling by Theon, used under CC BY-SA 3.0
`https://en.wikipedia.org/wiki/File:PavageHypPoincare2.svg`

Hyperbolic CAs



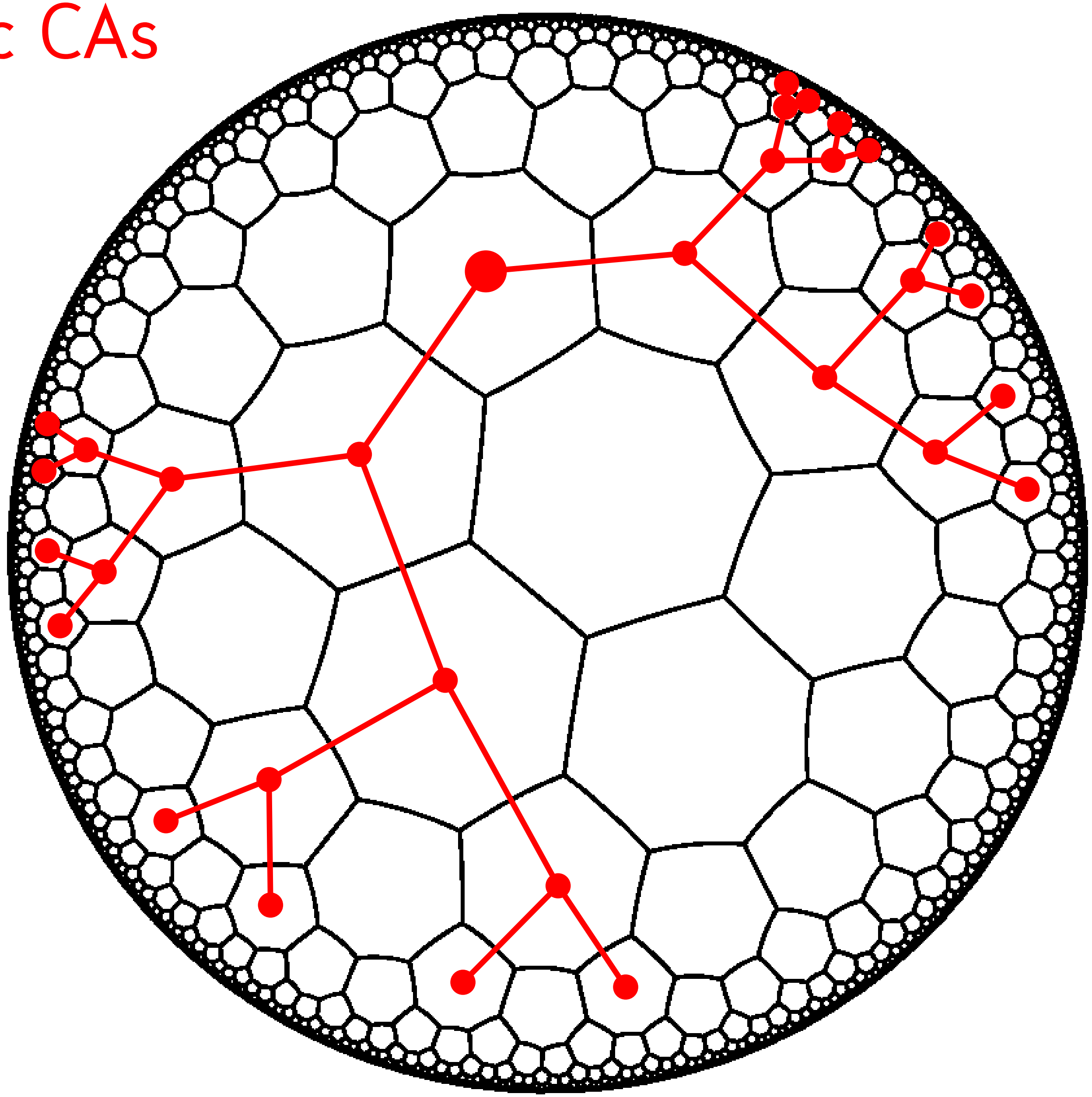
Hyperbolic tiling by Theon, used under CC BY-SA 3.0
`https://en.wikipedia.org/wiki/File:PavageHypPoincare2.svg`

Hyperbolic CAs



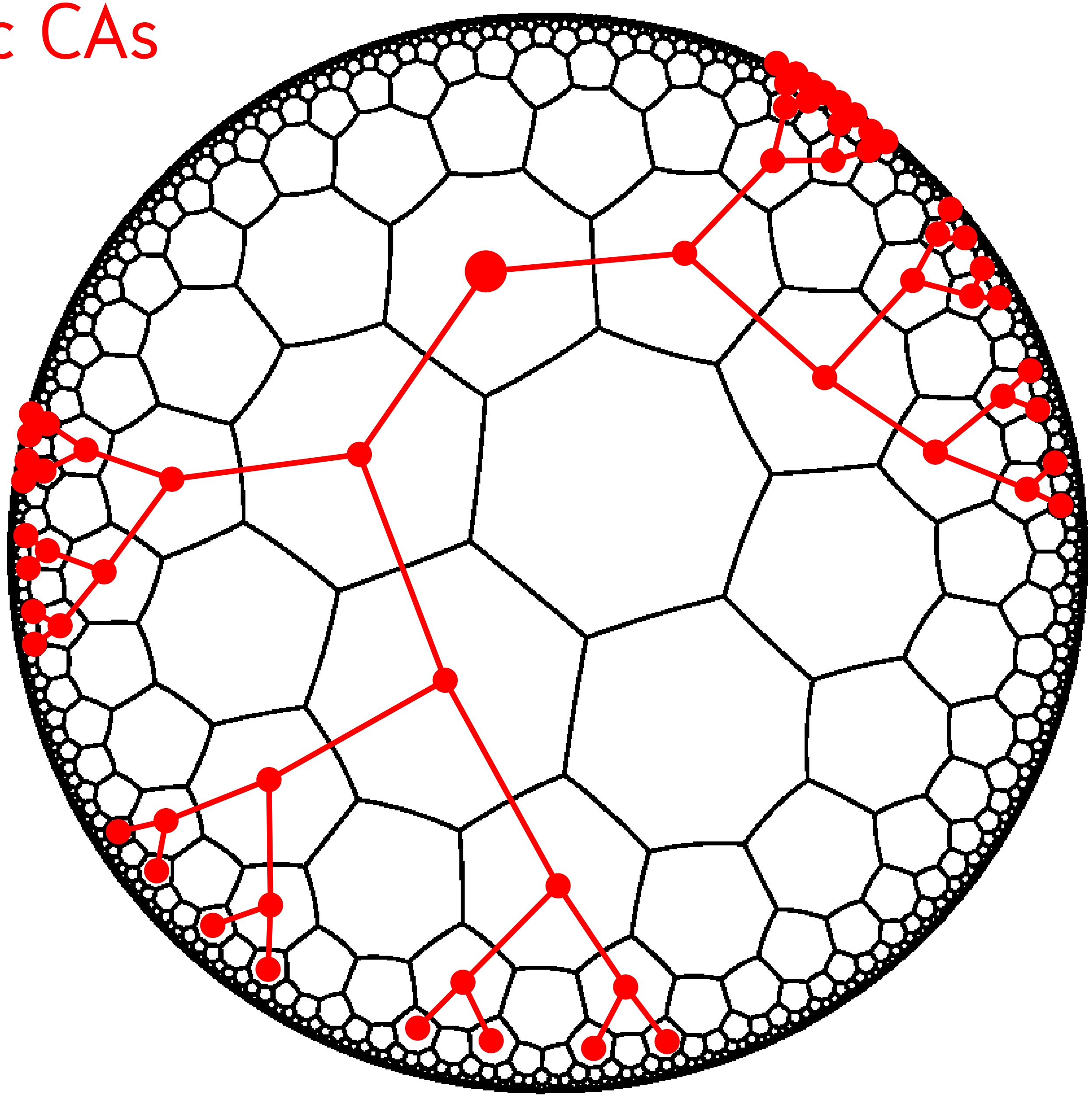
Hyperbolic tiling by Theon, used under CC BY-SA 3.0
`https://en.wikipedia.org/wiki/File:PavageHypPoincare2.svg`

Hyperbolic CAs



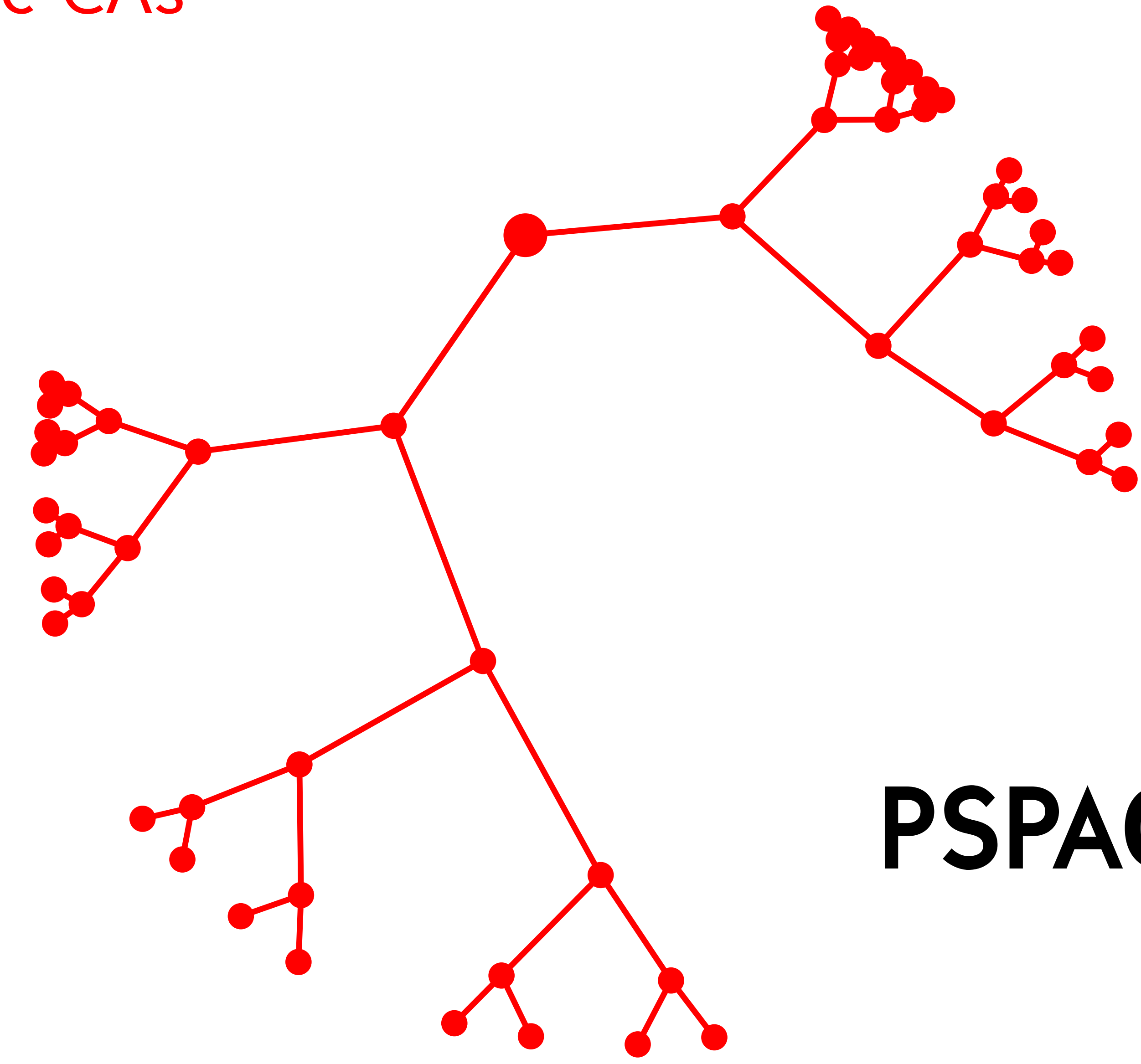
Hyperbolic tiling by Theon, used under CC BY-SA 3.0
`https://en.wikipedia.org/wiki/File:PavageHypPoincare2.svg`

Hyperbolic CAs



Hyperbolic tiling by Theon, used under CC BY-SA 3.0
`https://en.wikipedia.org/wiki/File:PavageHypPoincare2.svg`

Hyperbolic CAs



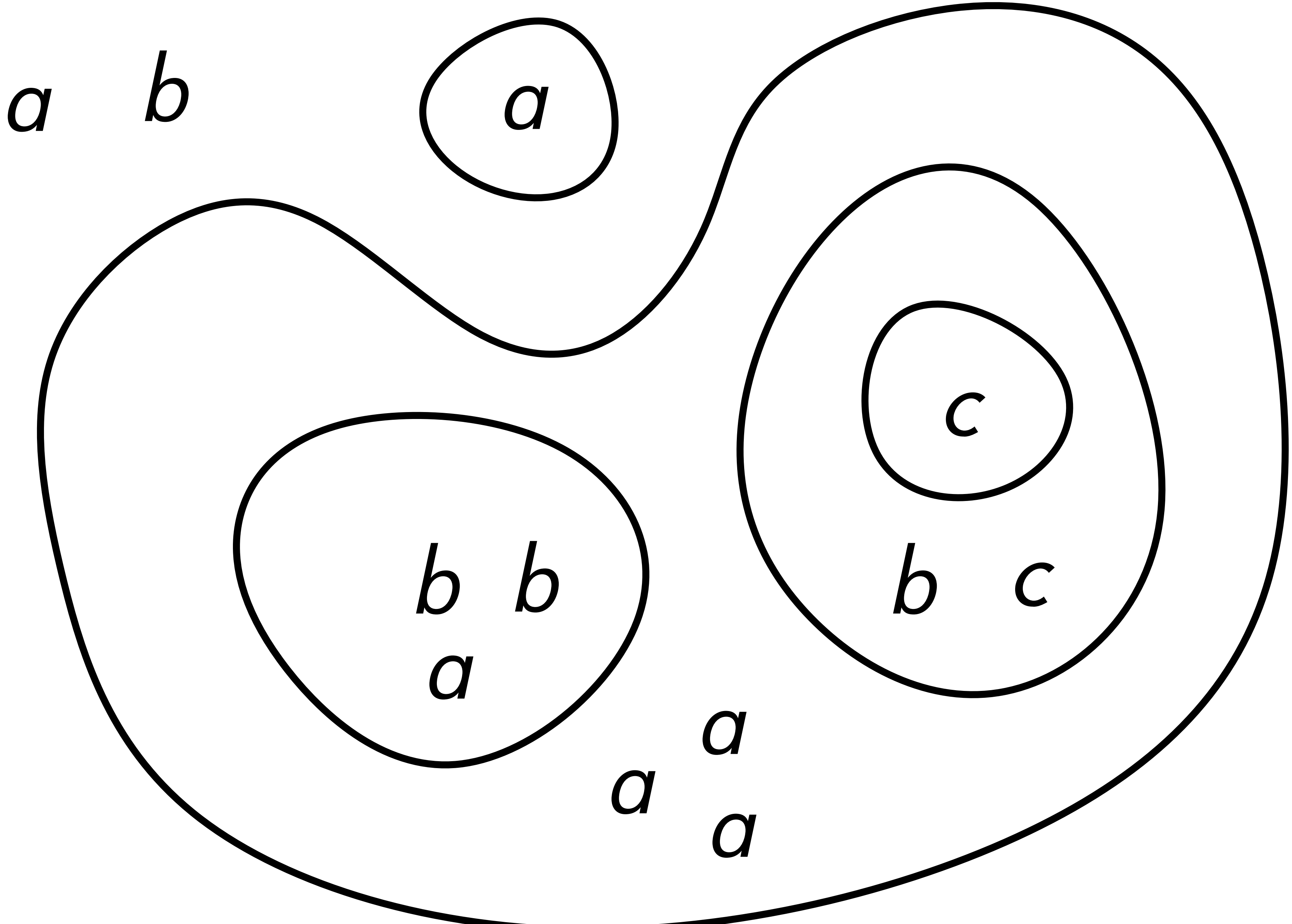
PSPACE

Hyperbolic tiling by Theon, used under CC BY-SA 3.0
`https://en.wikipedia.org/wiki/File:PavageHypPoincare2.svg`

First and second machine class

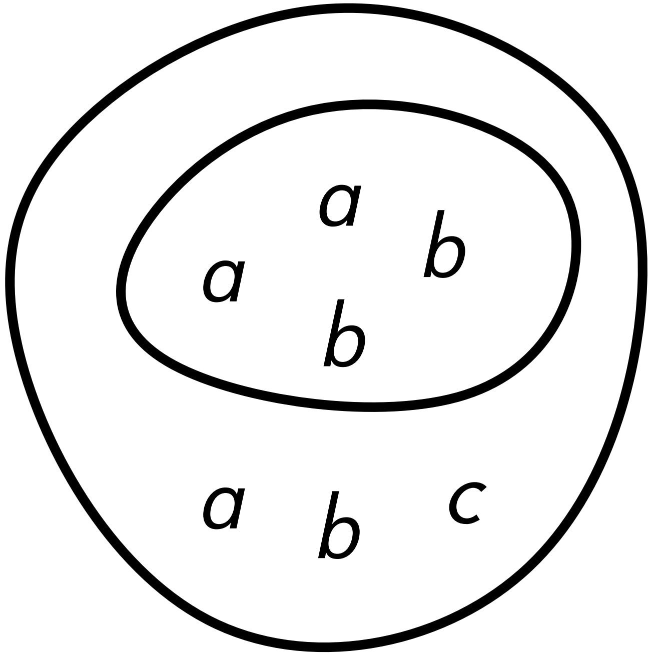
Many “concrete” sequential and parallel computing models are either first or second class machines

Membrane systems



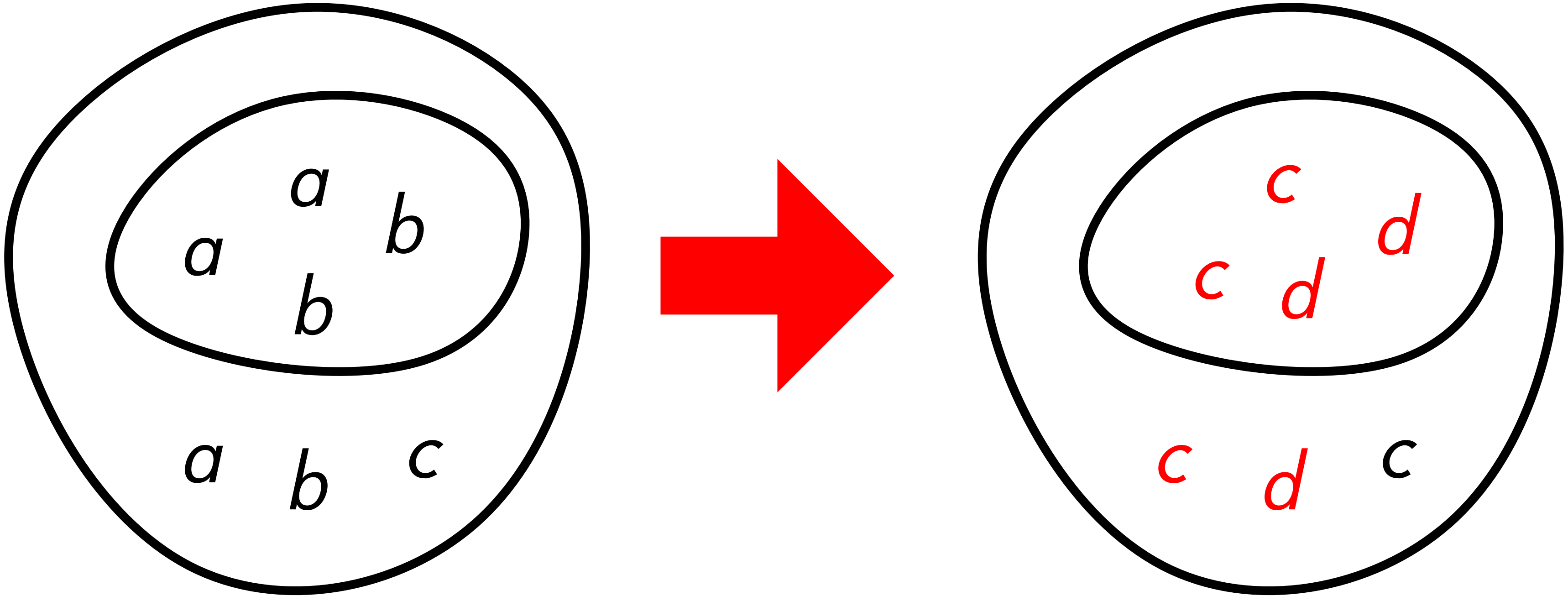
Simple chemical reactions

$$ab \to cd$$



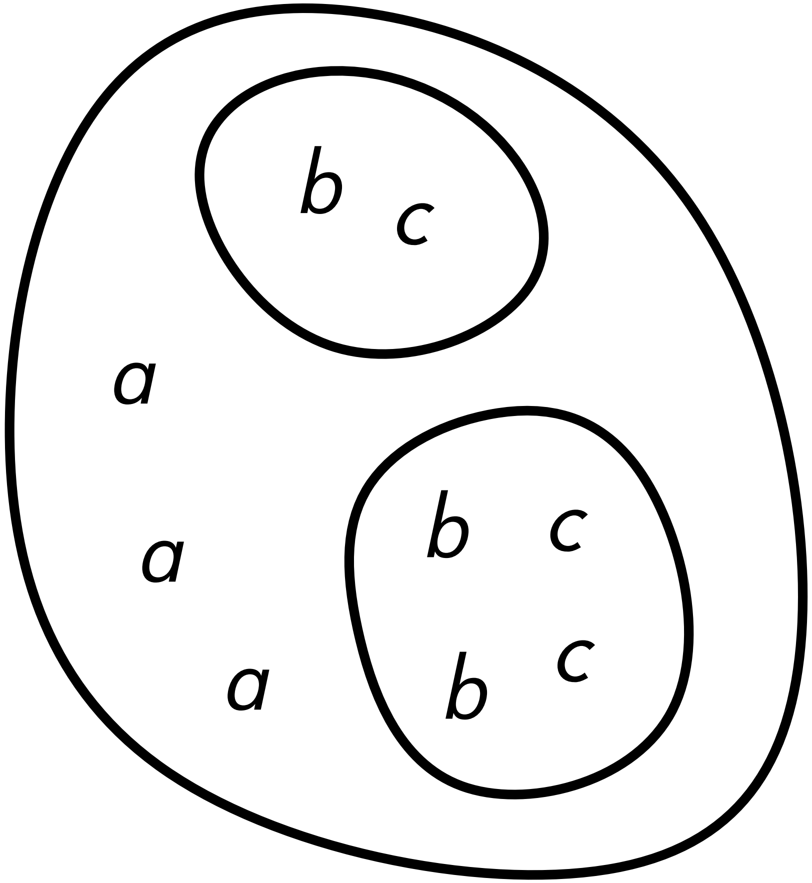
Simple chemical reactions

$$ab \to cd$$



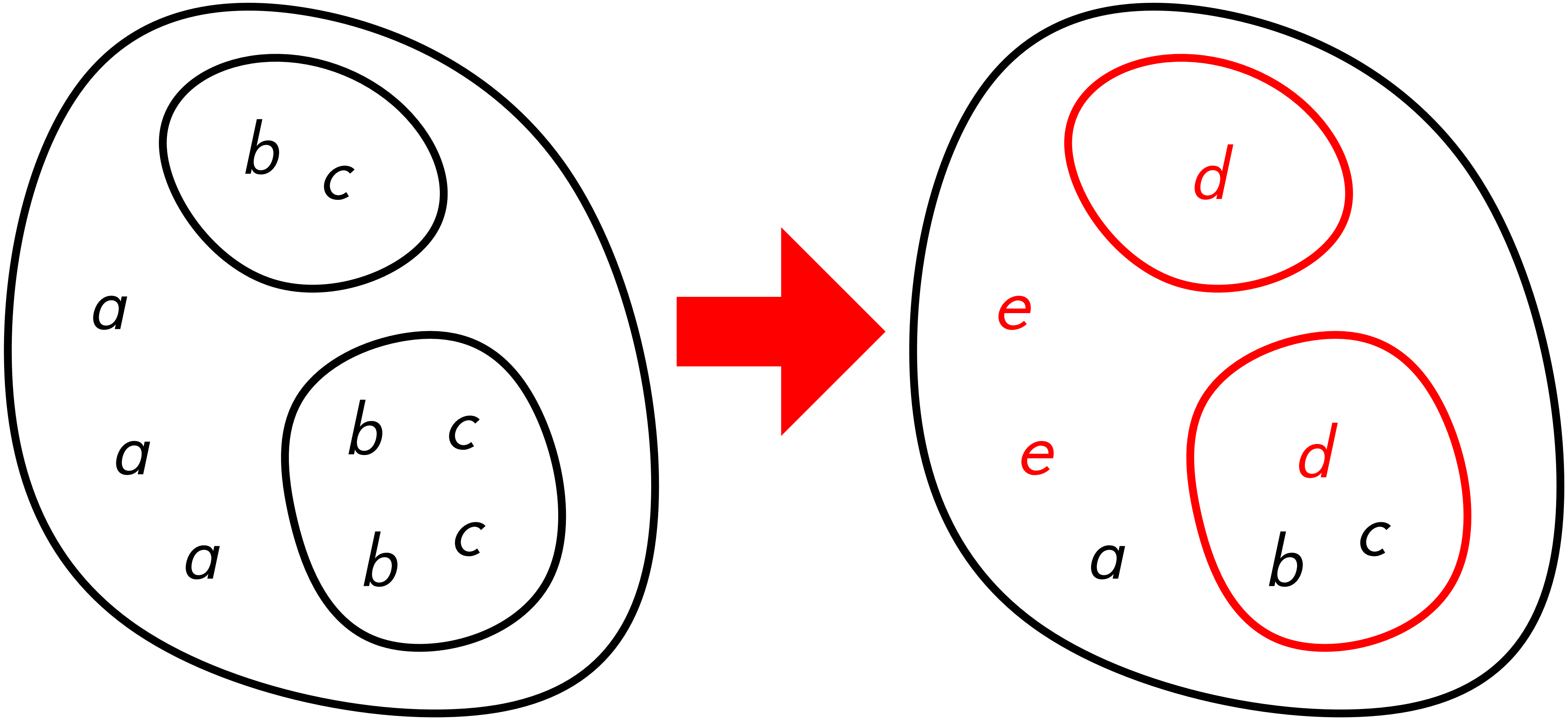
Communication between regions

$$a\,[bc] \rightarrow [d]\,e$$



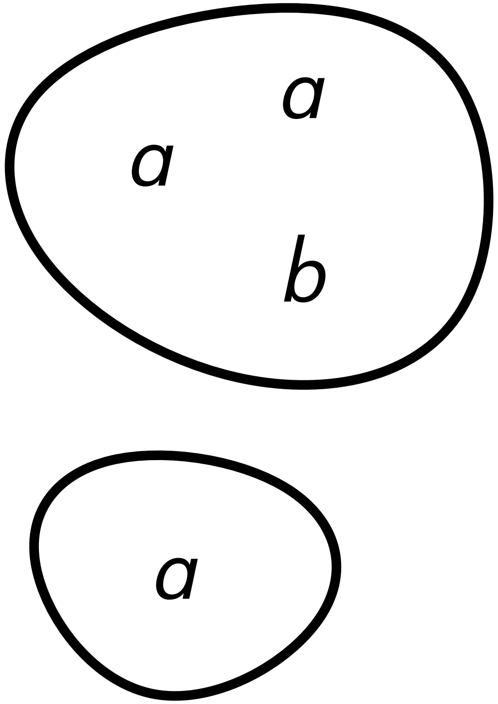
Communication between regions

$$a\,[bc] \rightarrow [d]\,e$$



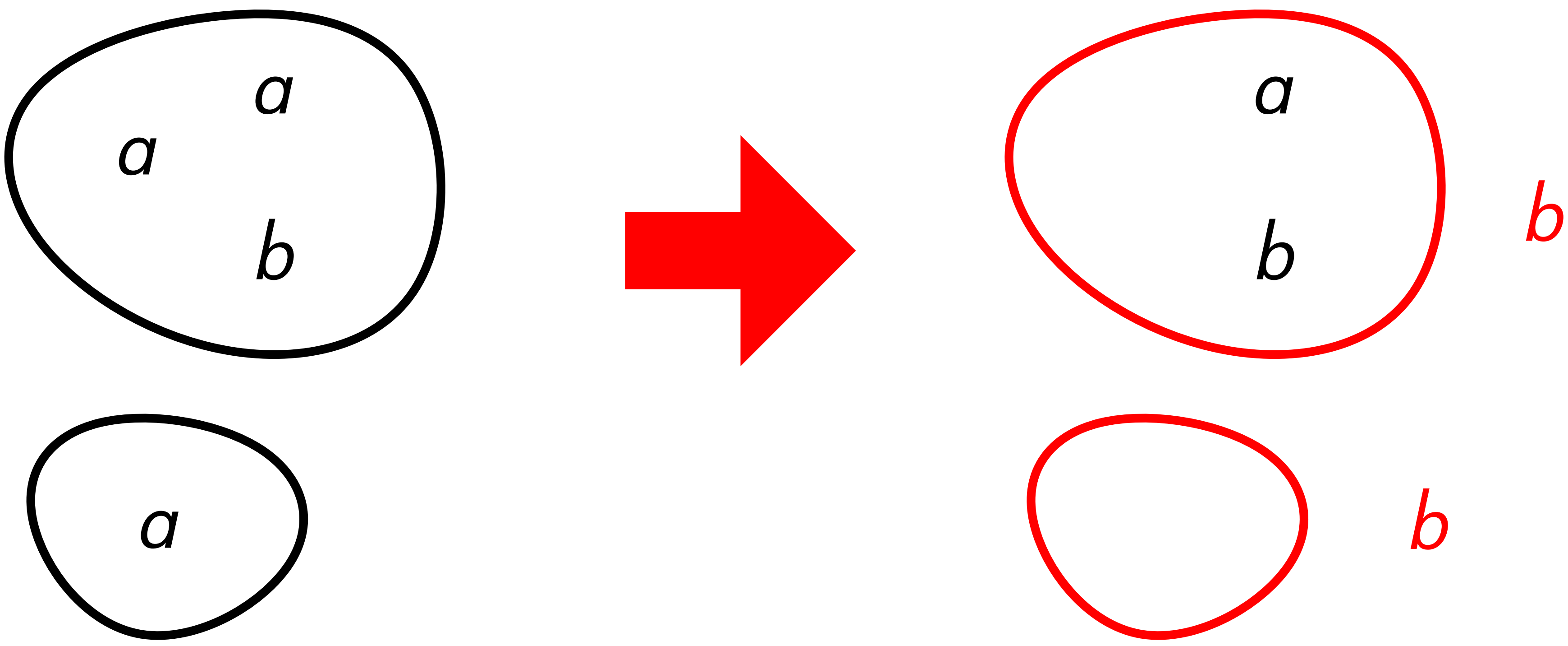
Monodirectional communication between regions

$$[a] \rightarrow [\ ]\, b$$



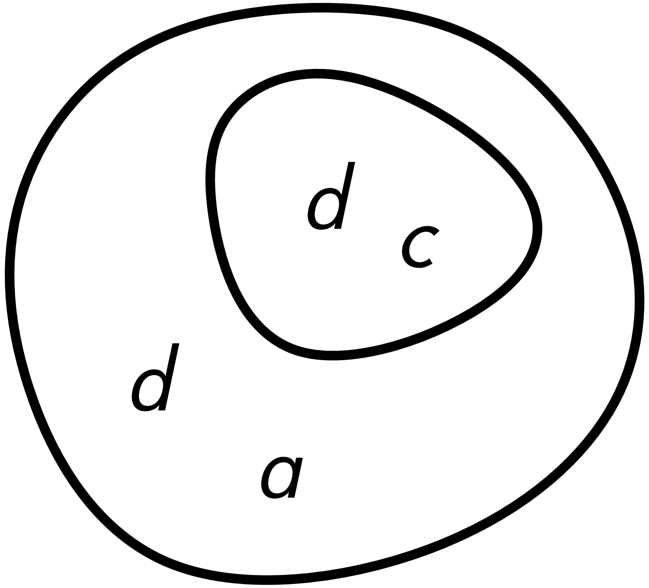
Monodirectional communication between regions

$$[a] \to [\ ]\, b$$



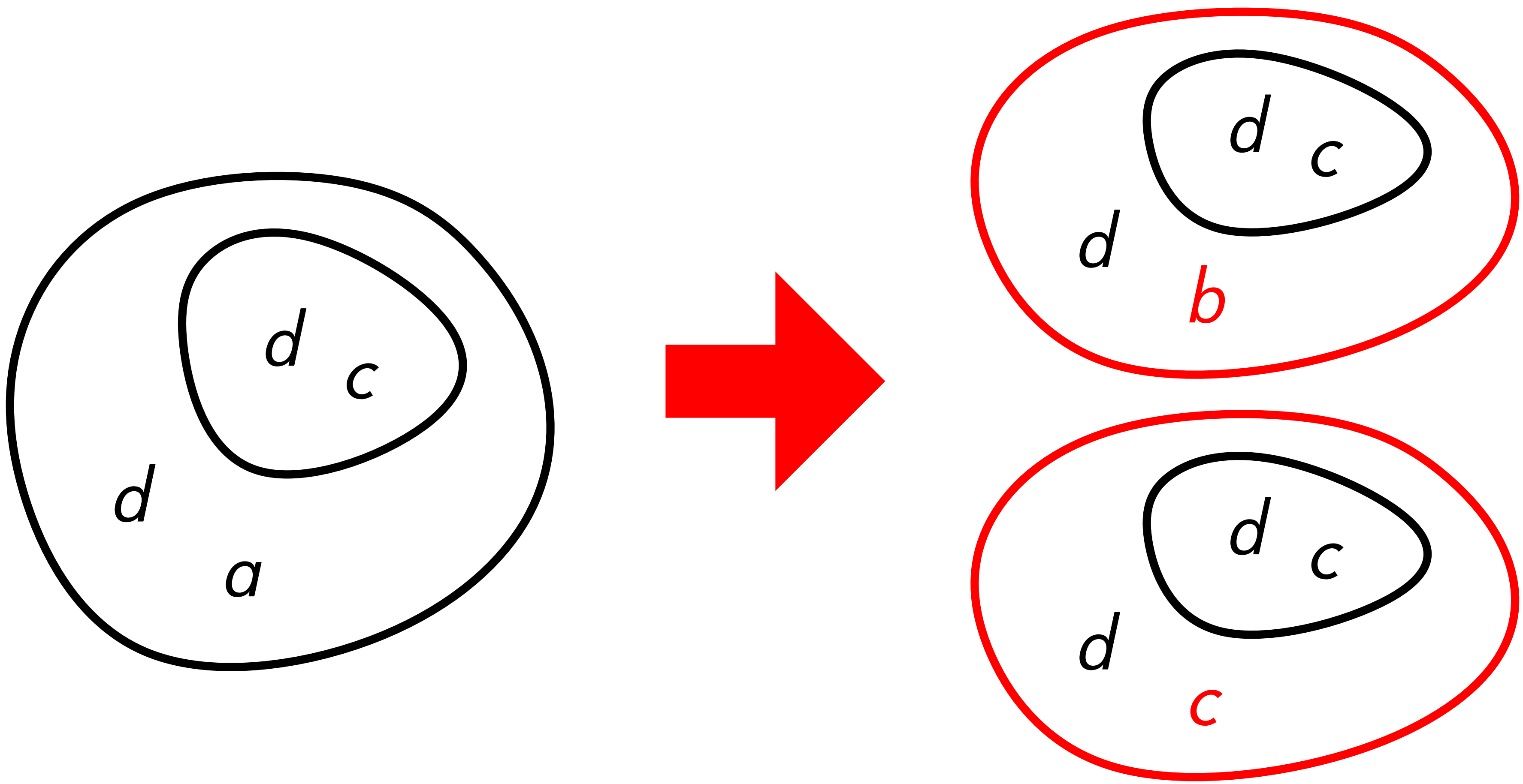
Membrane division

$$[a] \to [b]\ [c]$$



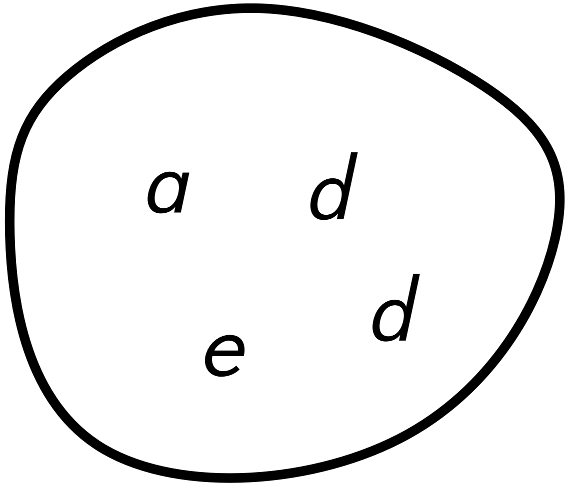
Membrane division

$$[a] \to [b]\,[c]$$



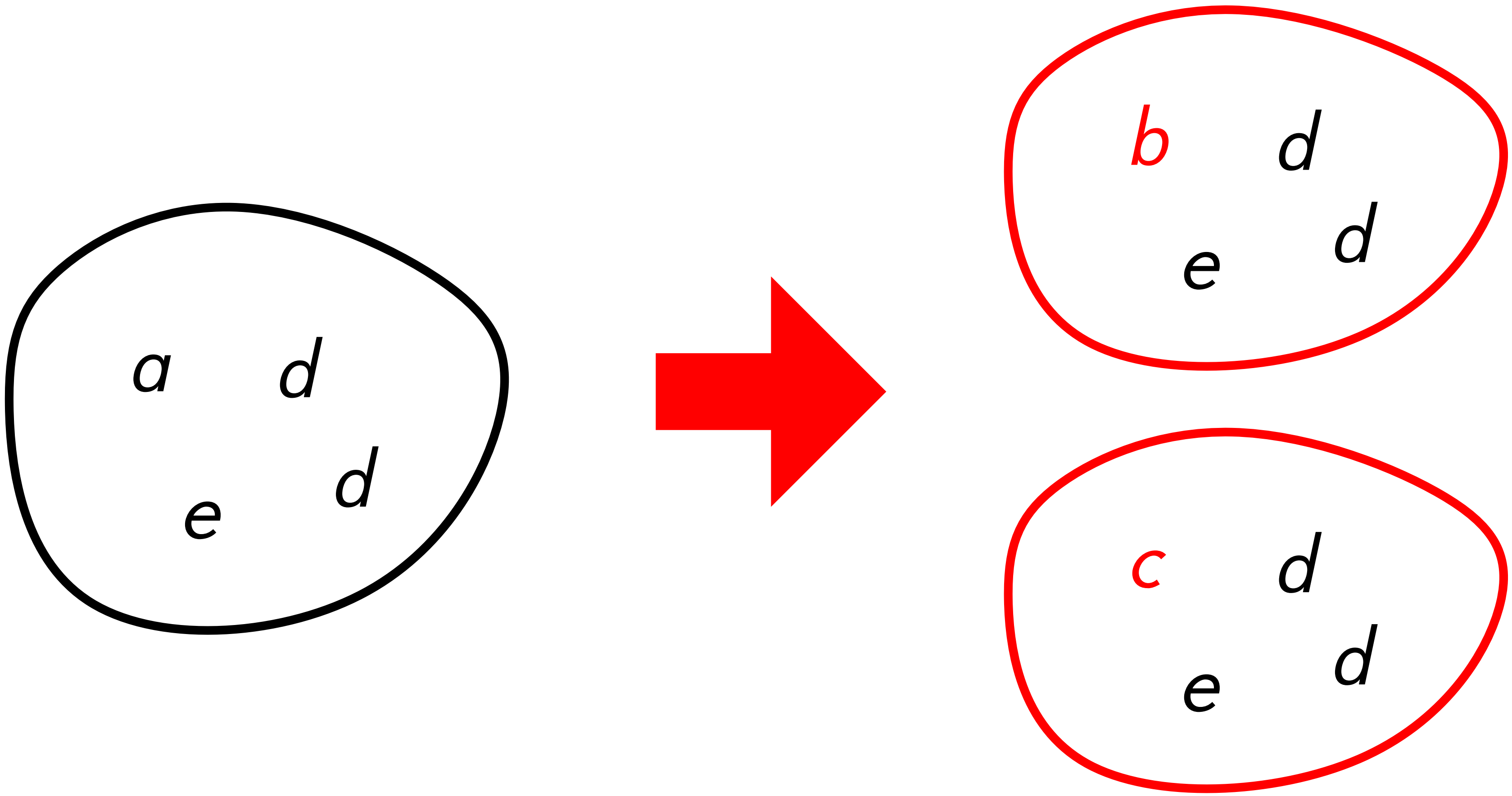
Elementary membrane division

$$[a] \to [b]\ [c]$$



Elementary membrane division

$$[a] \rightarrow [b]\ [c]$$

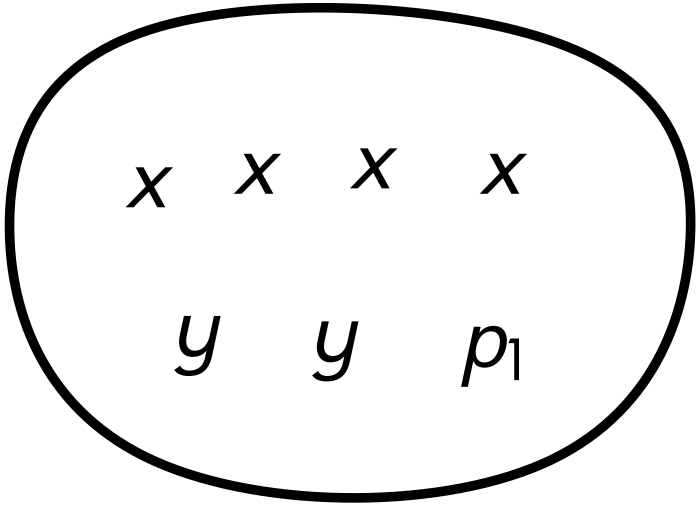


Register/counter machines

1: dec x, 2, 4
2: inc y, 3
3: inc y, 1
4: halt

Register/counter machines

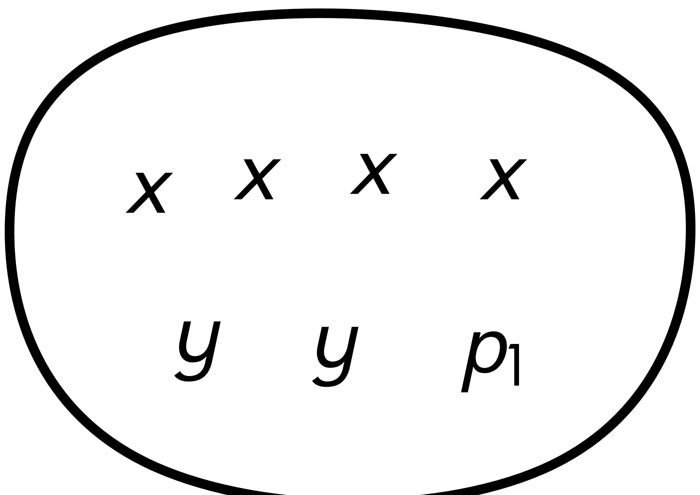
1: dec x, 2, 4
2: inc y, 3
3: inc y, 1
4: halt



Register/counter machines

1: dec x, 2, 4
2: inc y, 3
3: inc y, 1
4: halt

$$p_2 \to y\ p_3$$



Register/counter machines

1: dec x, 2, 4
2: inc y, 3
3: inc y, 1
4: halt

$$p_2 \rightarrow y\ p_3$$

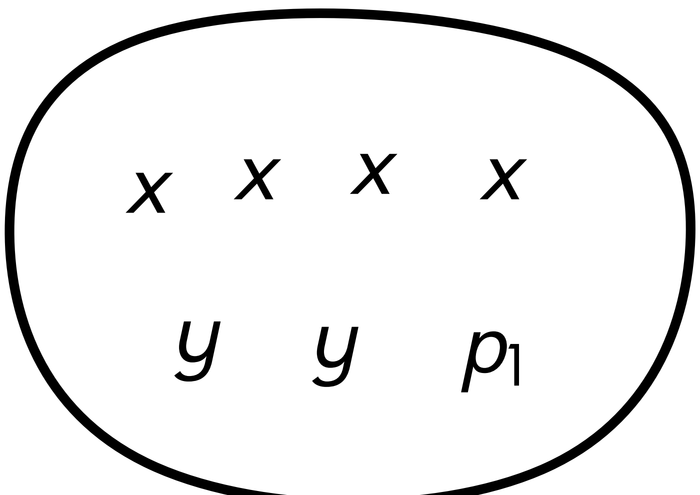
$$p_1 \rightarrow p_1'\ \overline{x}$$

$$p_1' \rightarrow p_1''$$

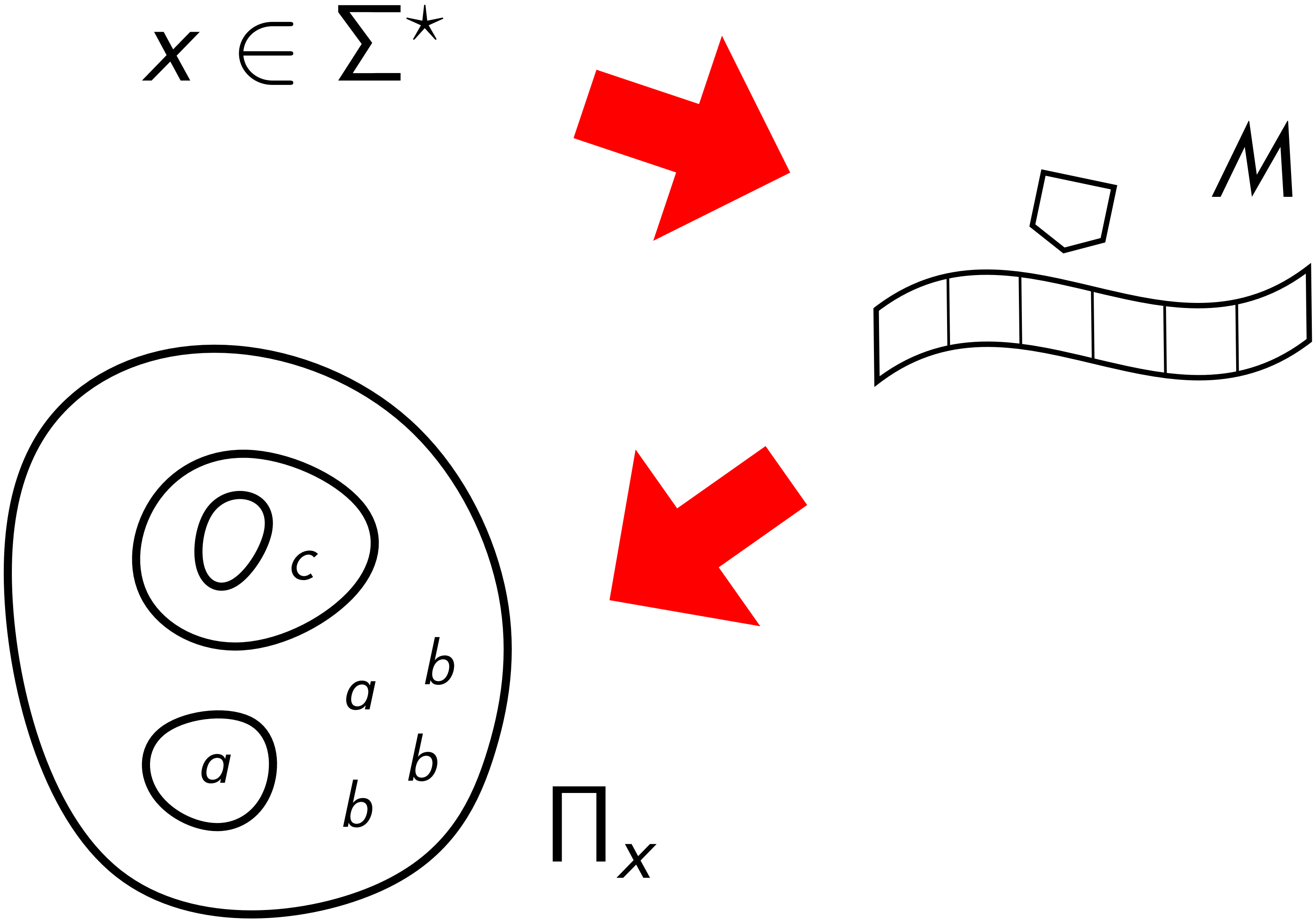
$$x\ \overline{x} \rightarrow x'$$

$$p_1''\ x' \rightarrow p_2$$

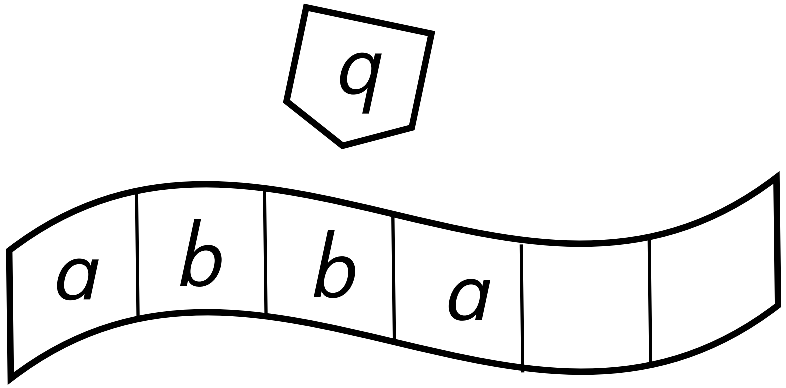
$$p_1''\ \overline{x} \rightarrow p_4$$



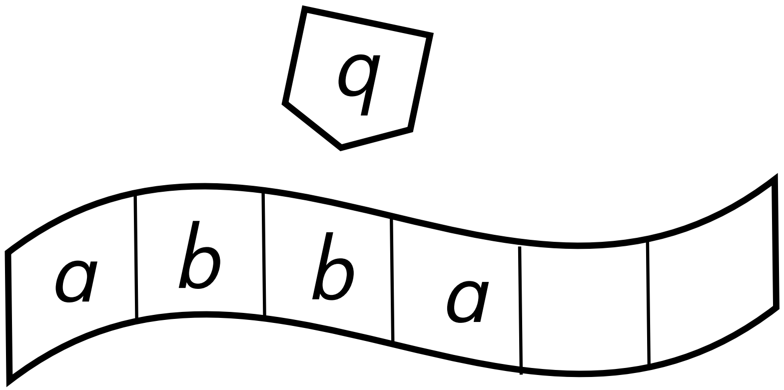
(Semi-)uniform families of membrane systems

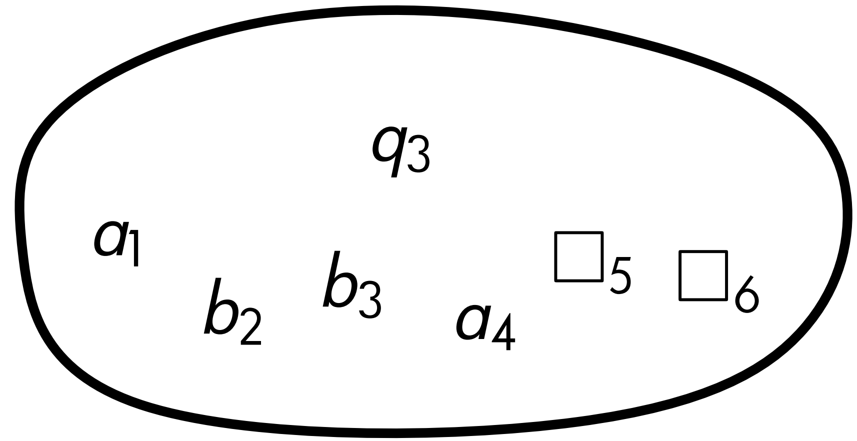


Simulating Turing machines efficiently

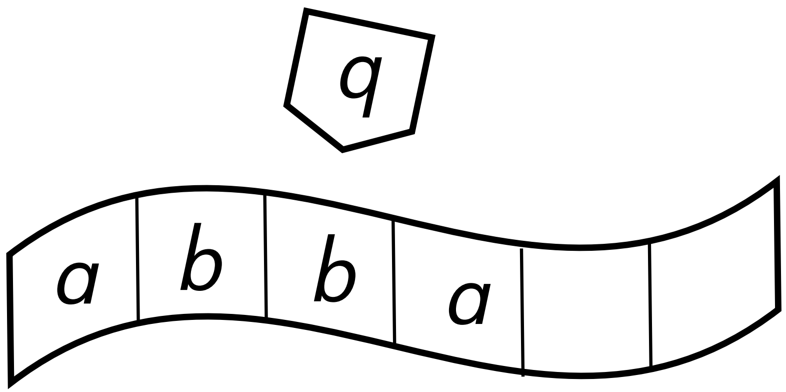


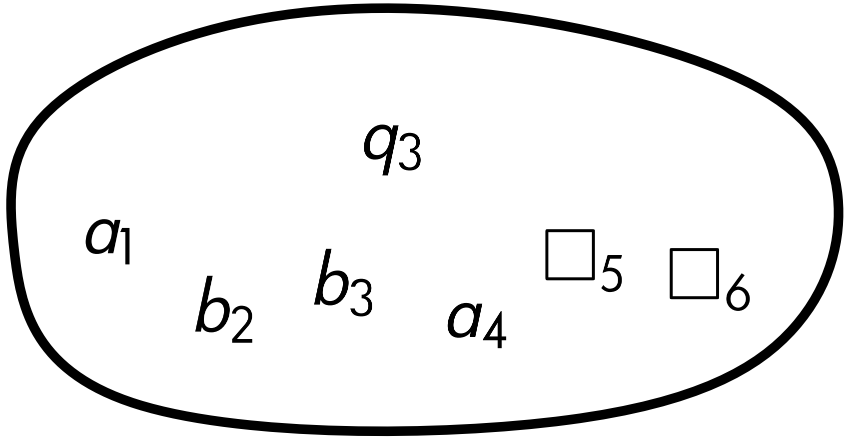
Simulating Turing machines efficiently





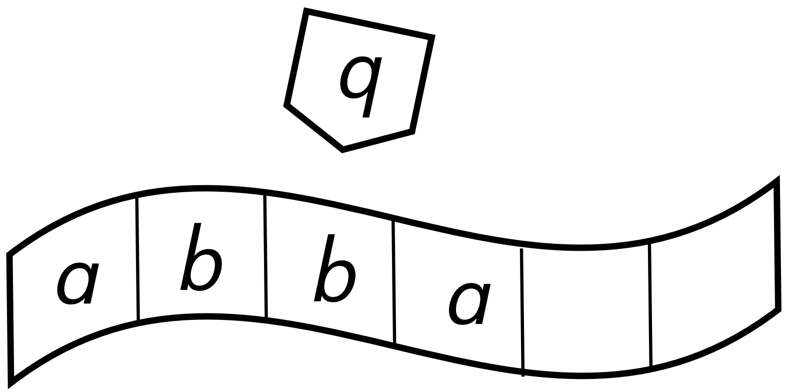
Simulating Turing machines efficiently

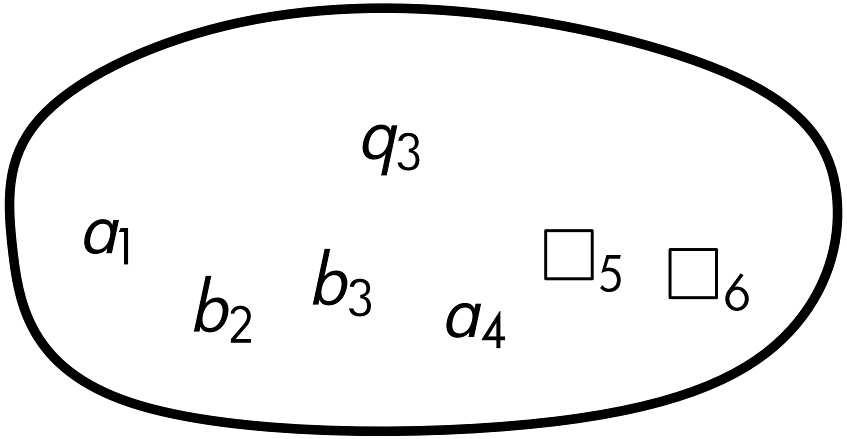




$$\delta(q, b) = (q', a, +1)$$

Simulating Turing machines efficiently

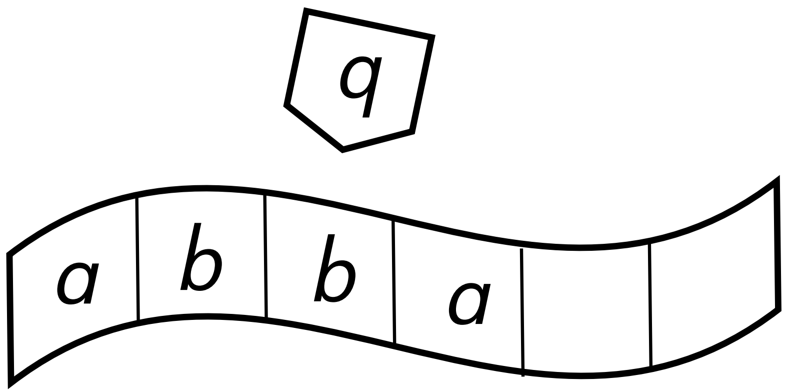


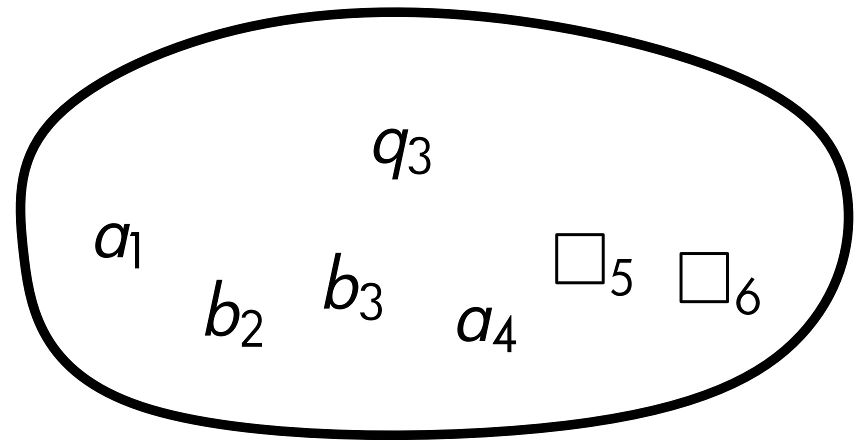


$$\delta(q, b) = (q', a, +1)$$

$$q_i\ b_i \rightarrow q'_{i+1}\ a_i \qquad 1 \leq i \leq n$$

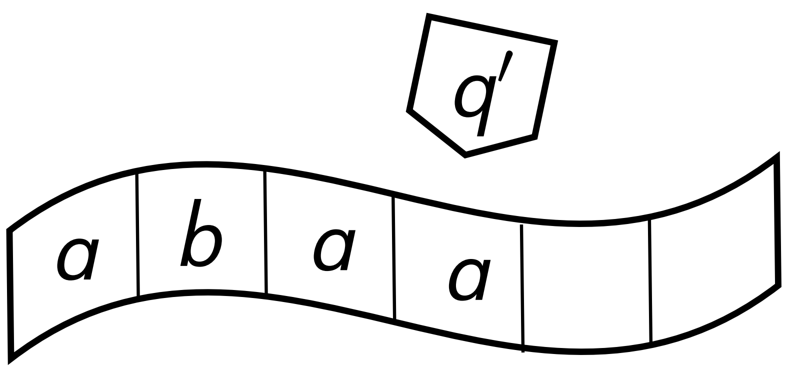
Simulating Turing machines efficiently

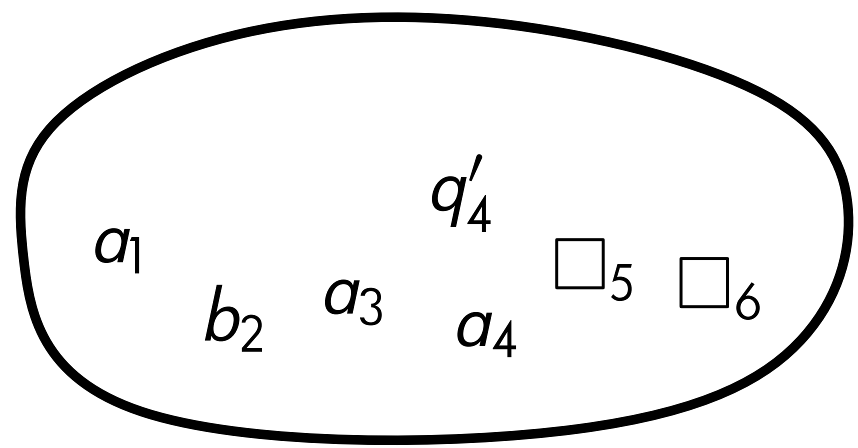




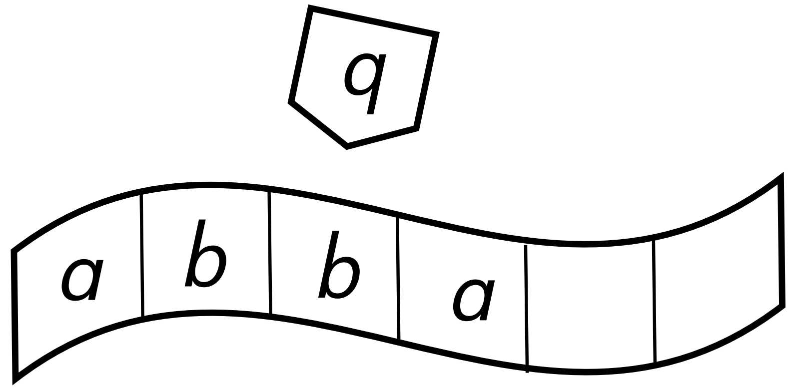
$\delta(q, b) = (q', a, +1)$

$q_i \; b_i \to q'_{i+1} \; a_i \qquad 1 \leq i \leq n$



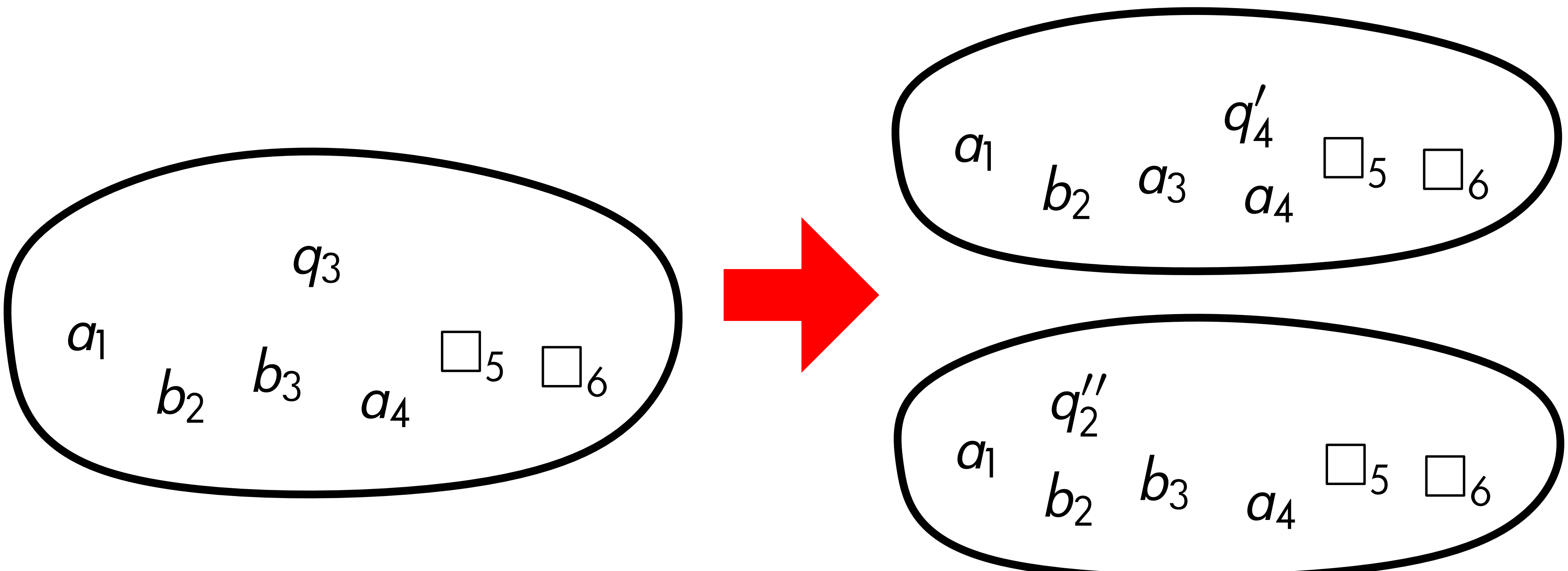


Simulating nondeterministic Turing machines

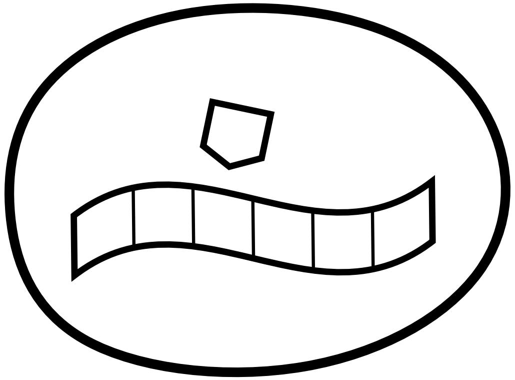


$$\delta(q, b) = \begin{cases} (q', a, +1) \\ (q'', b, -1) \end{cases}$$

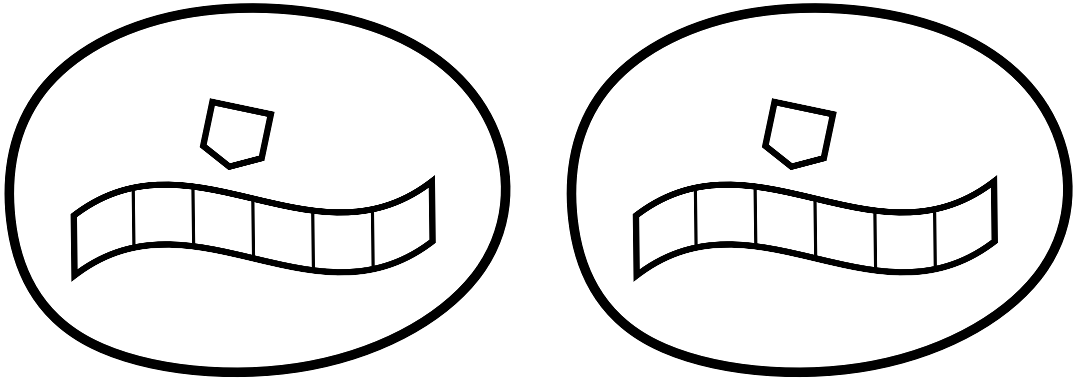
$$[q_i \; b_i] \to [q'_{i+1} \; a_i] \; [q''_{i-1} \; b_i]$$



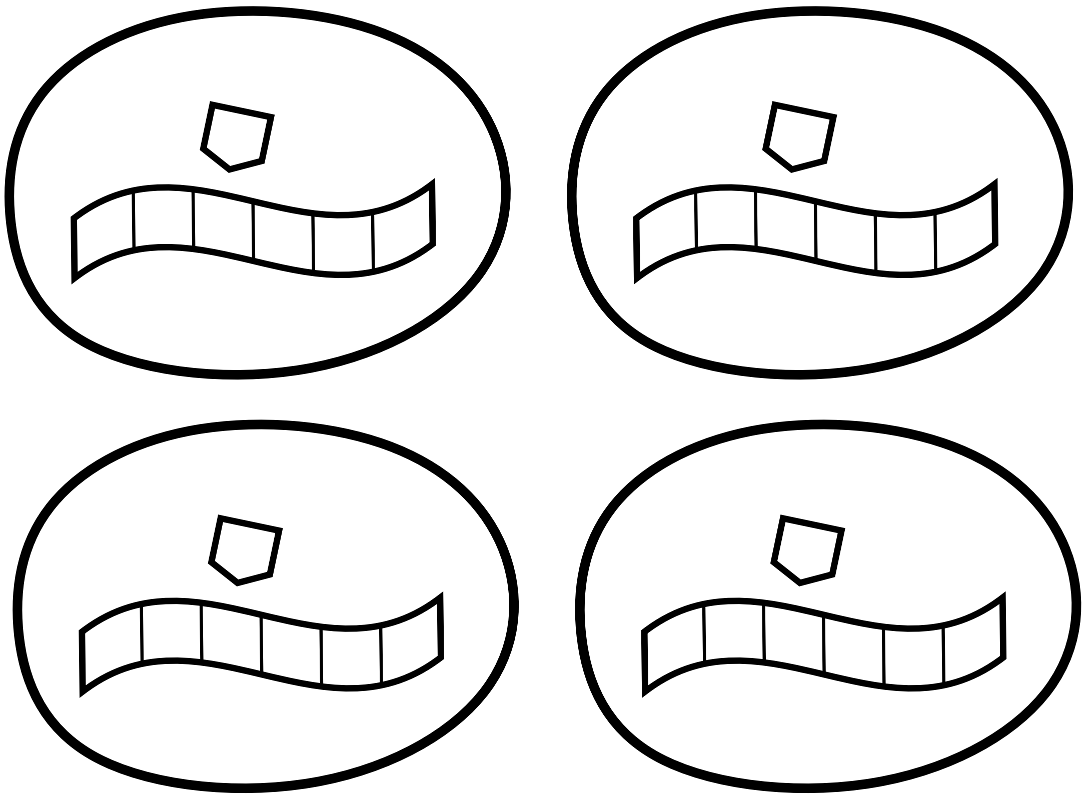
Simulating nondeterministic Turing machines



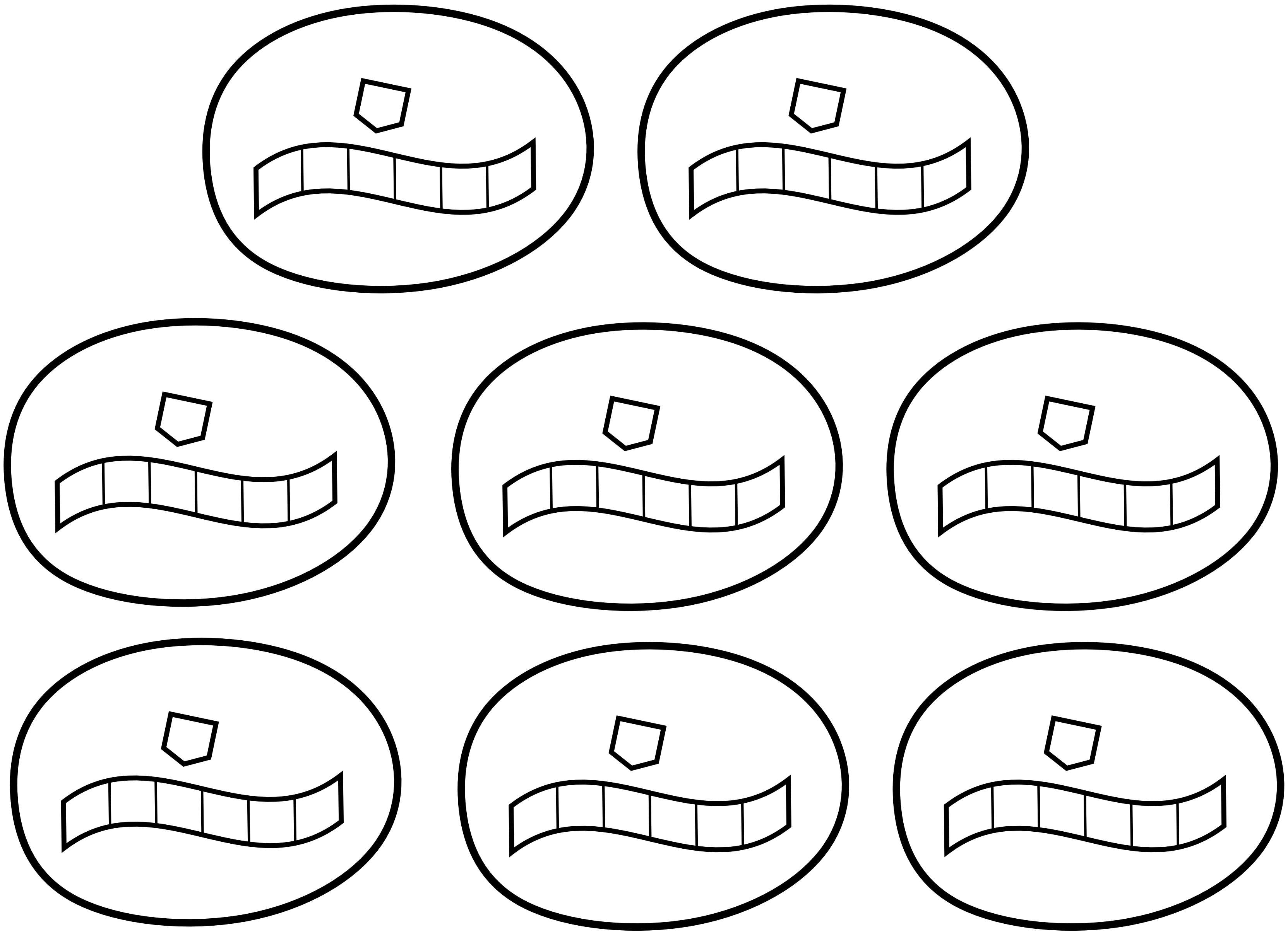
Simulating nondeterministic Turing machines



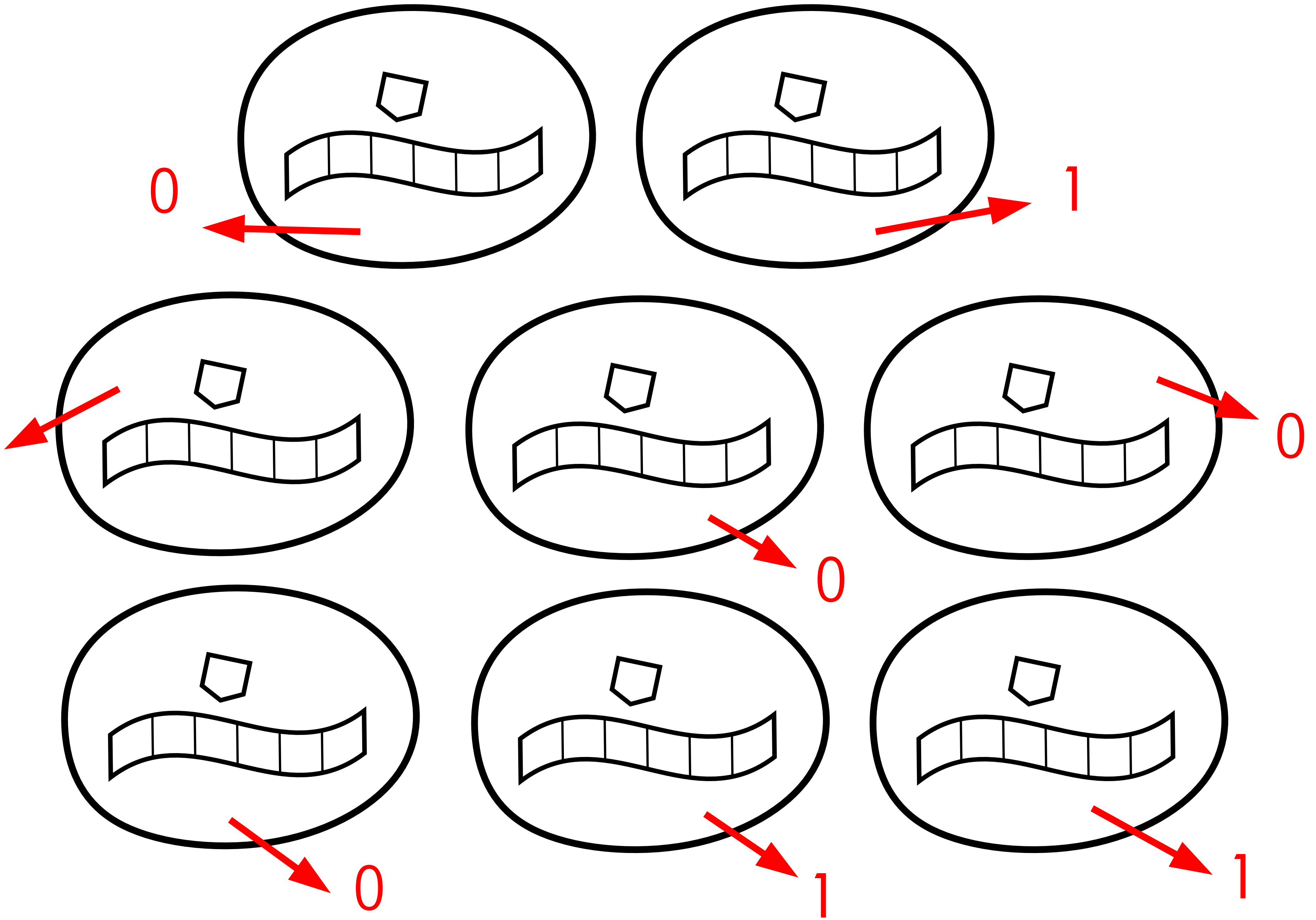
Simulating nondeterministic Turing machines



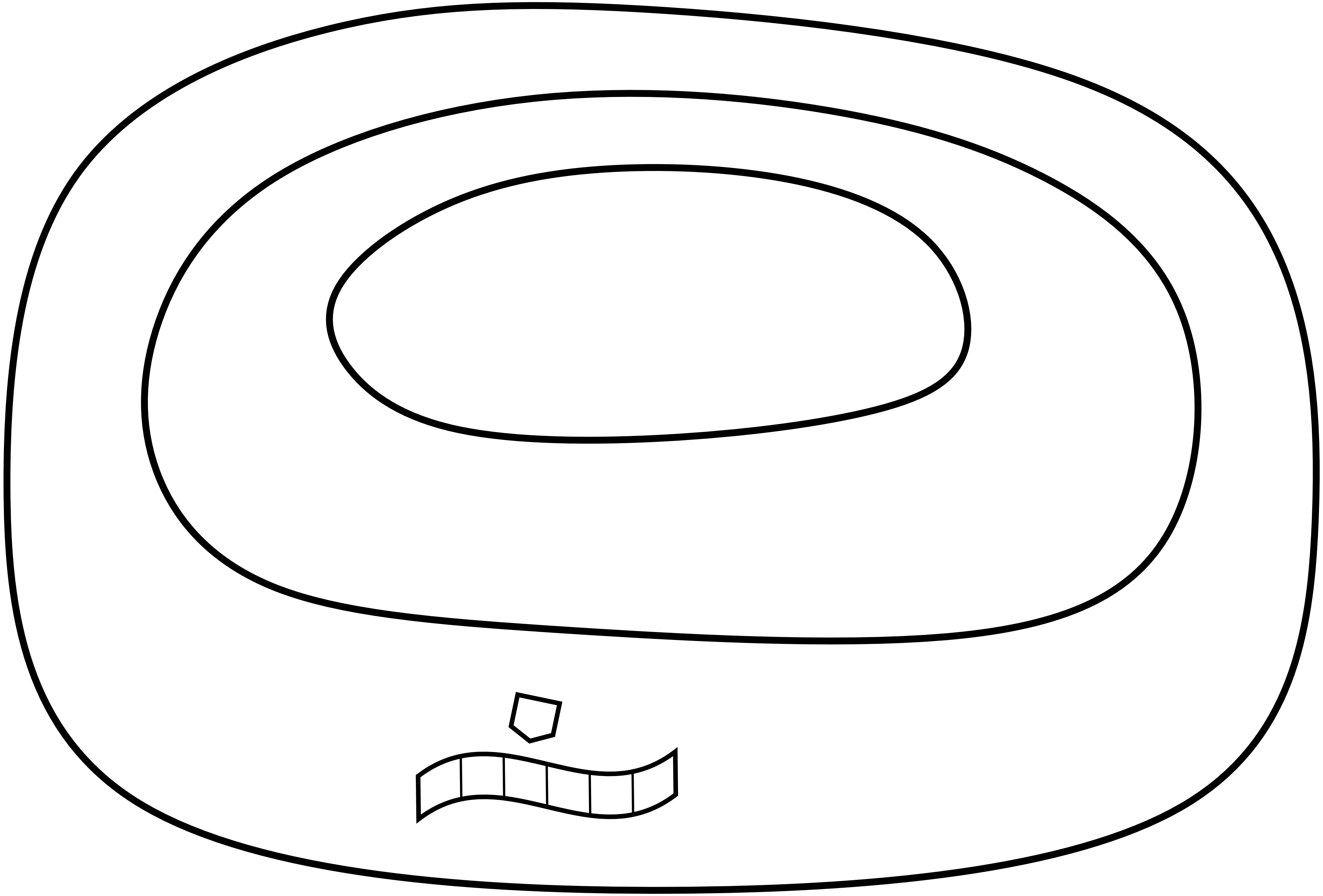
Simulating nondeterministic Turing machines



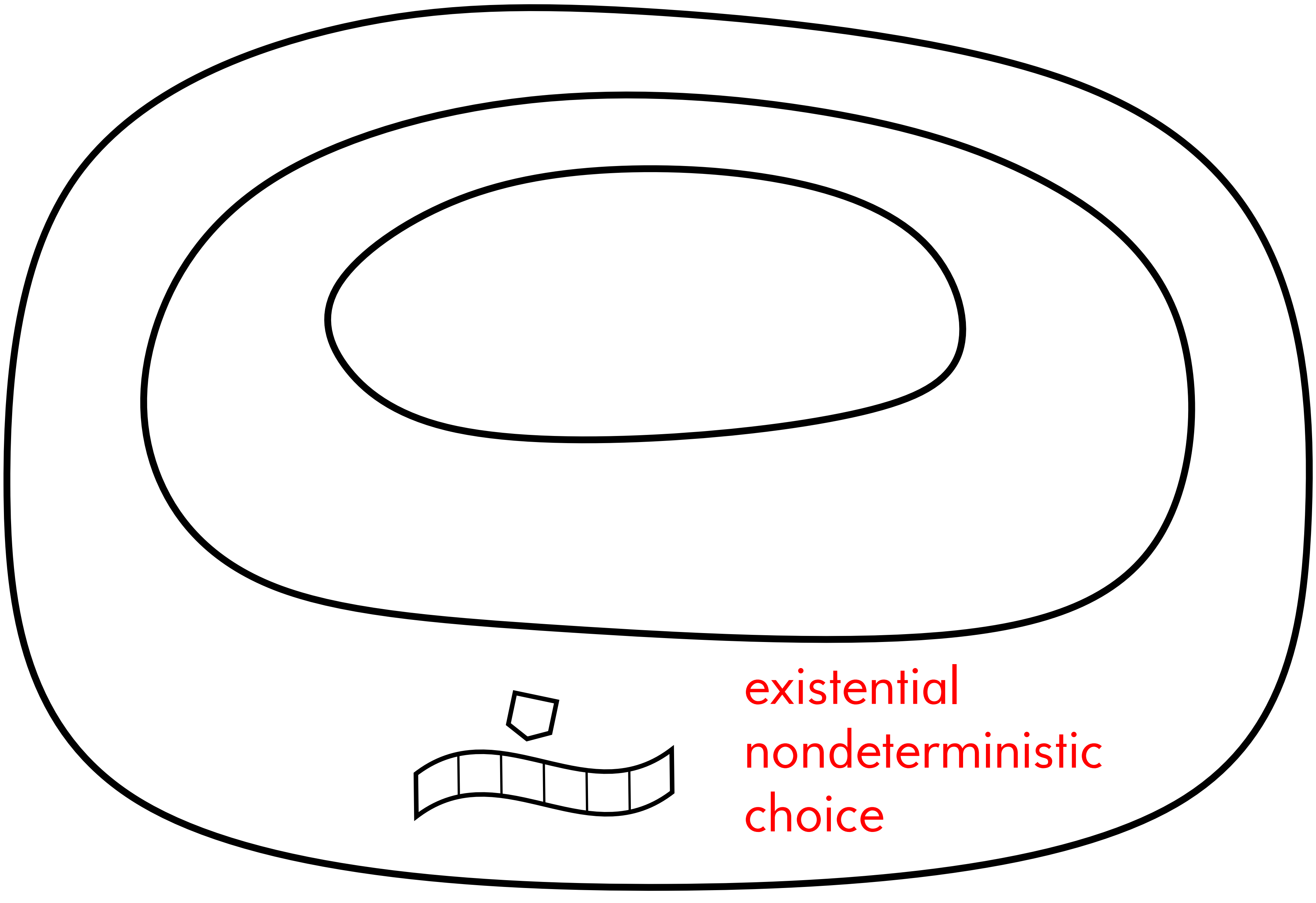
Simulating nondeterministic Turing machines



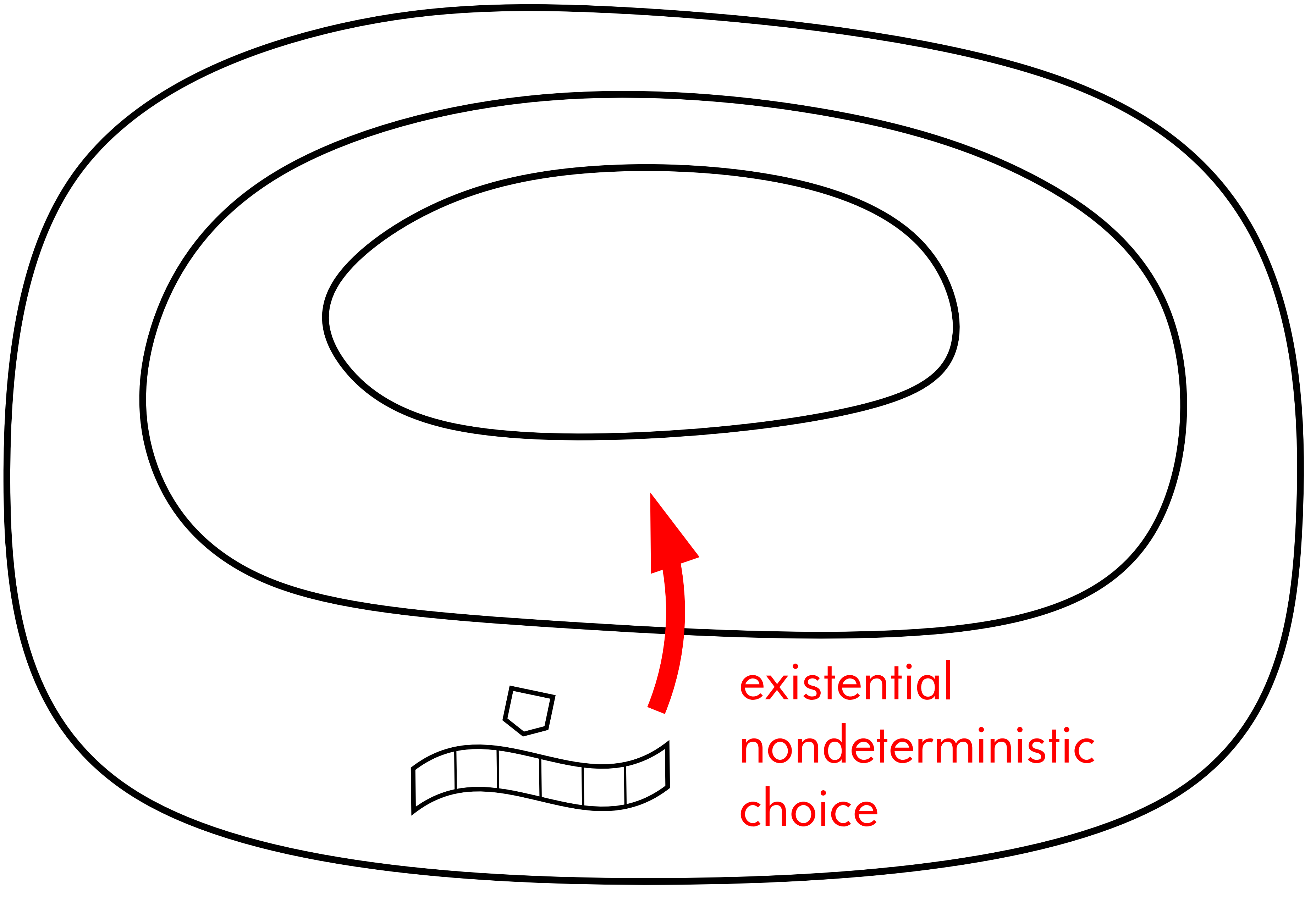
Simulating alternating Turing machines



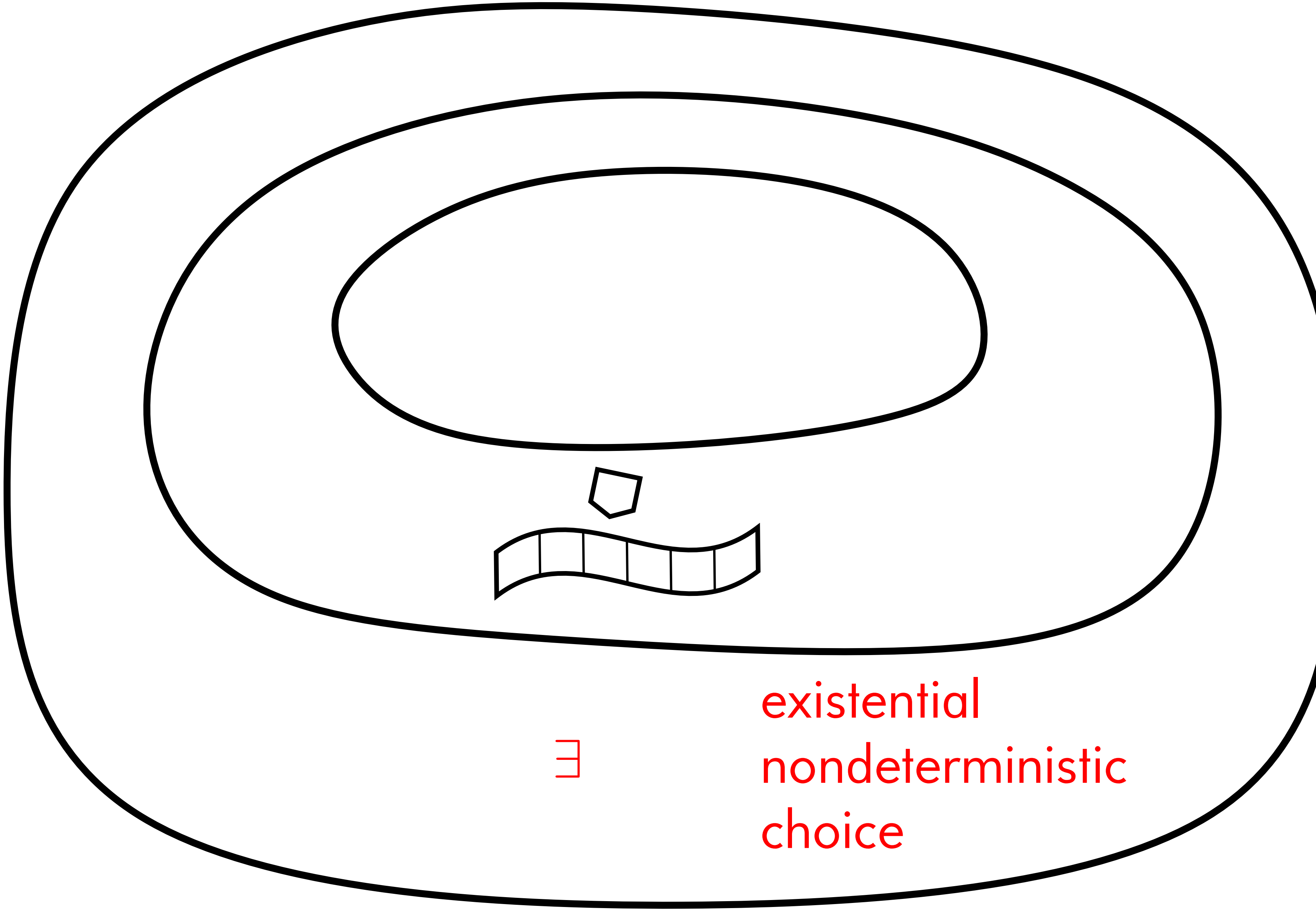
Simulating alternating Turing machines



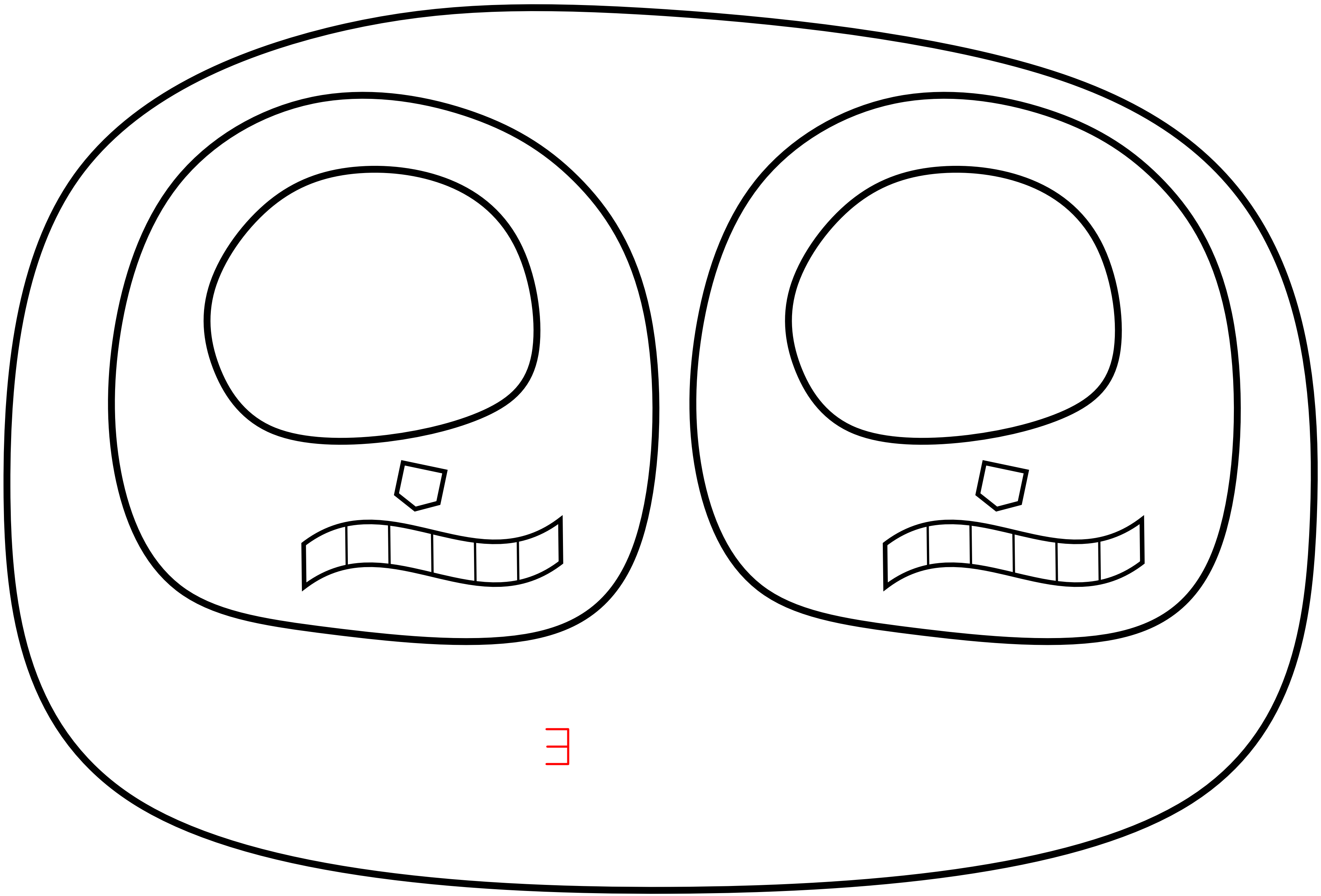
Simulating alternating Turing machines



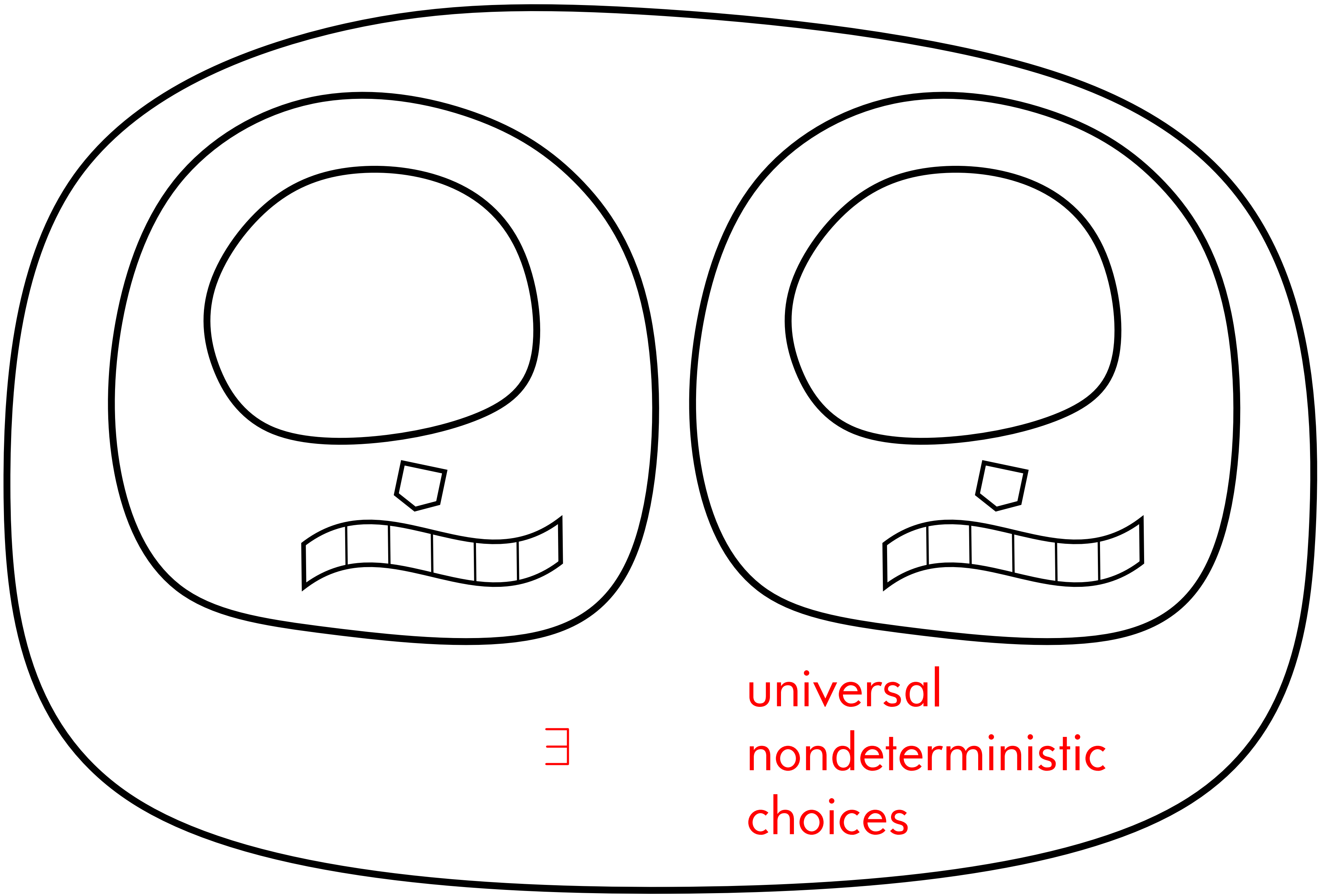
Simulating alternating Turing machines



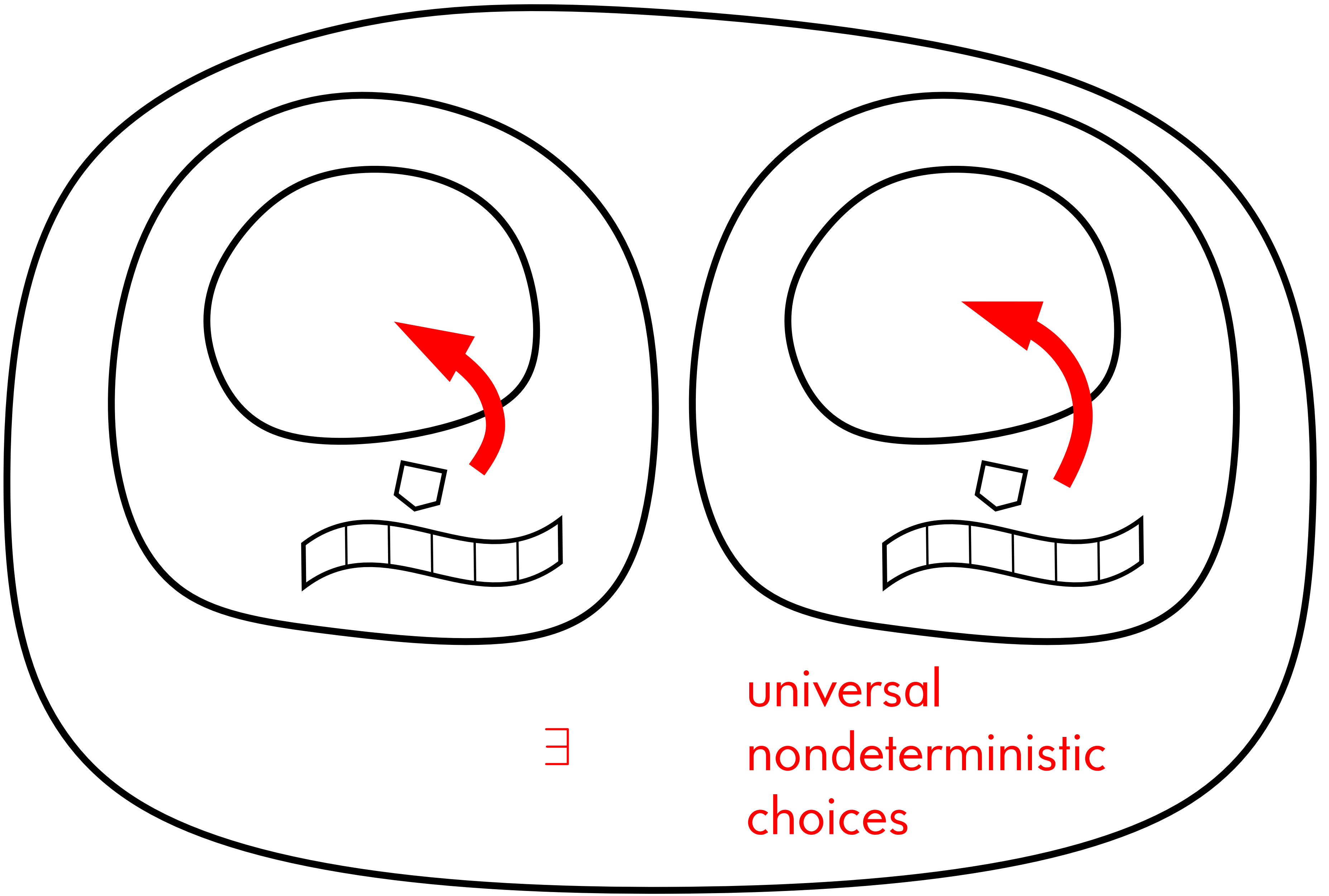
Simulating alternating Turing machines



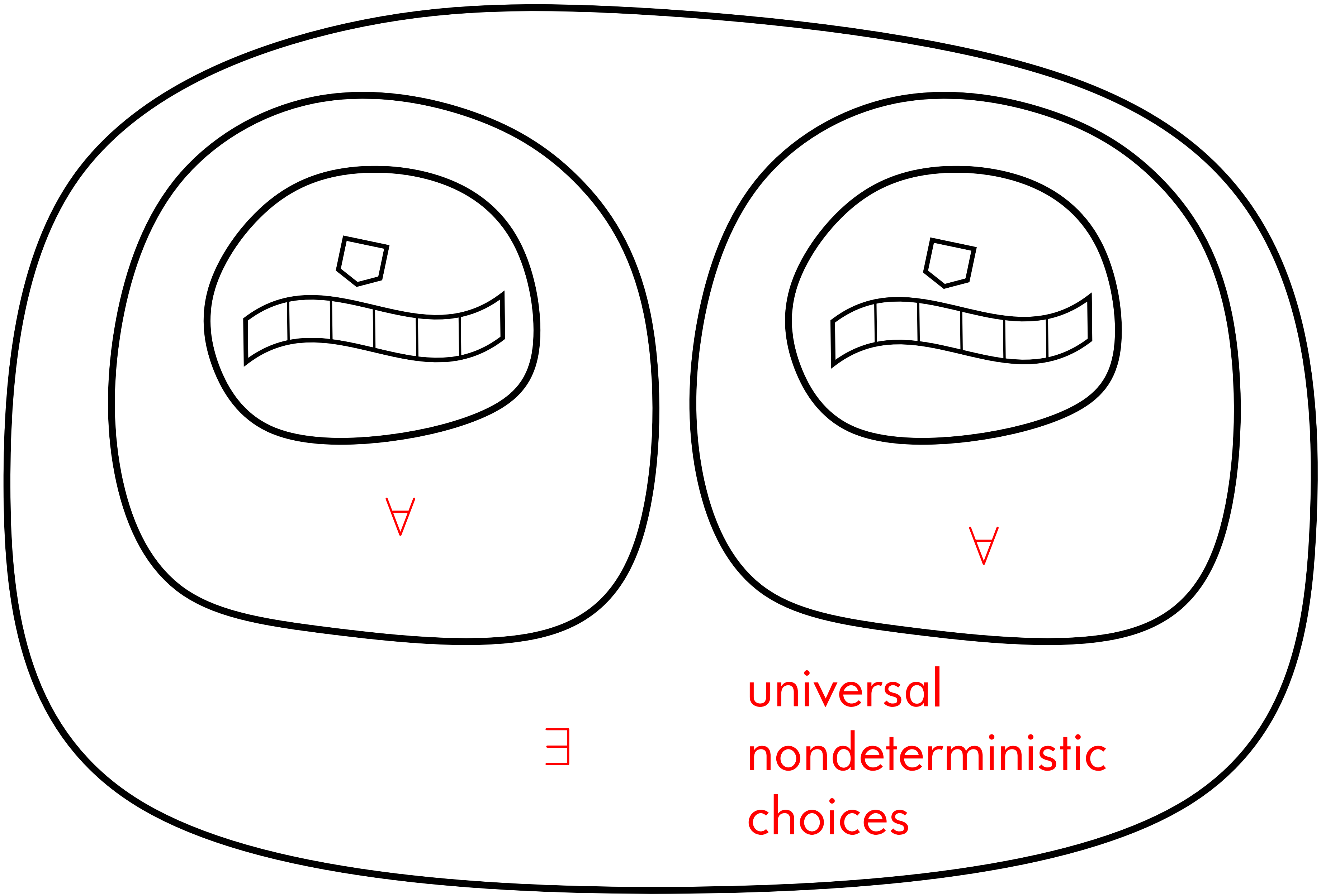
Simulating alternating Turing machines



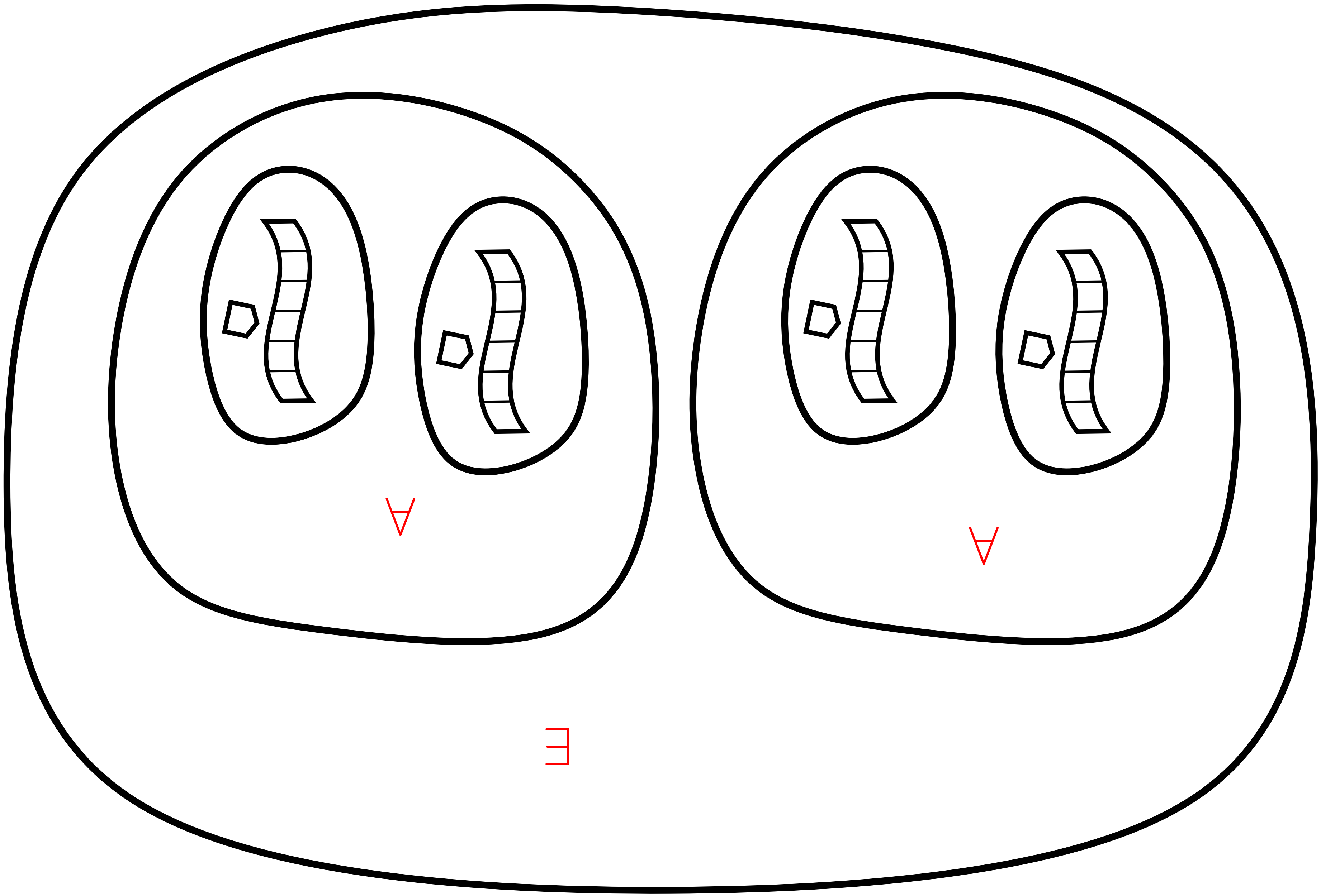
Simulating alternating Turing machines



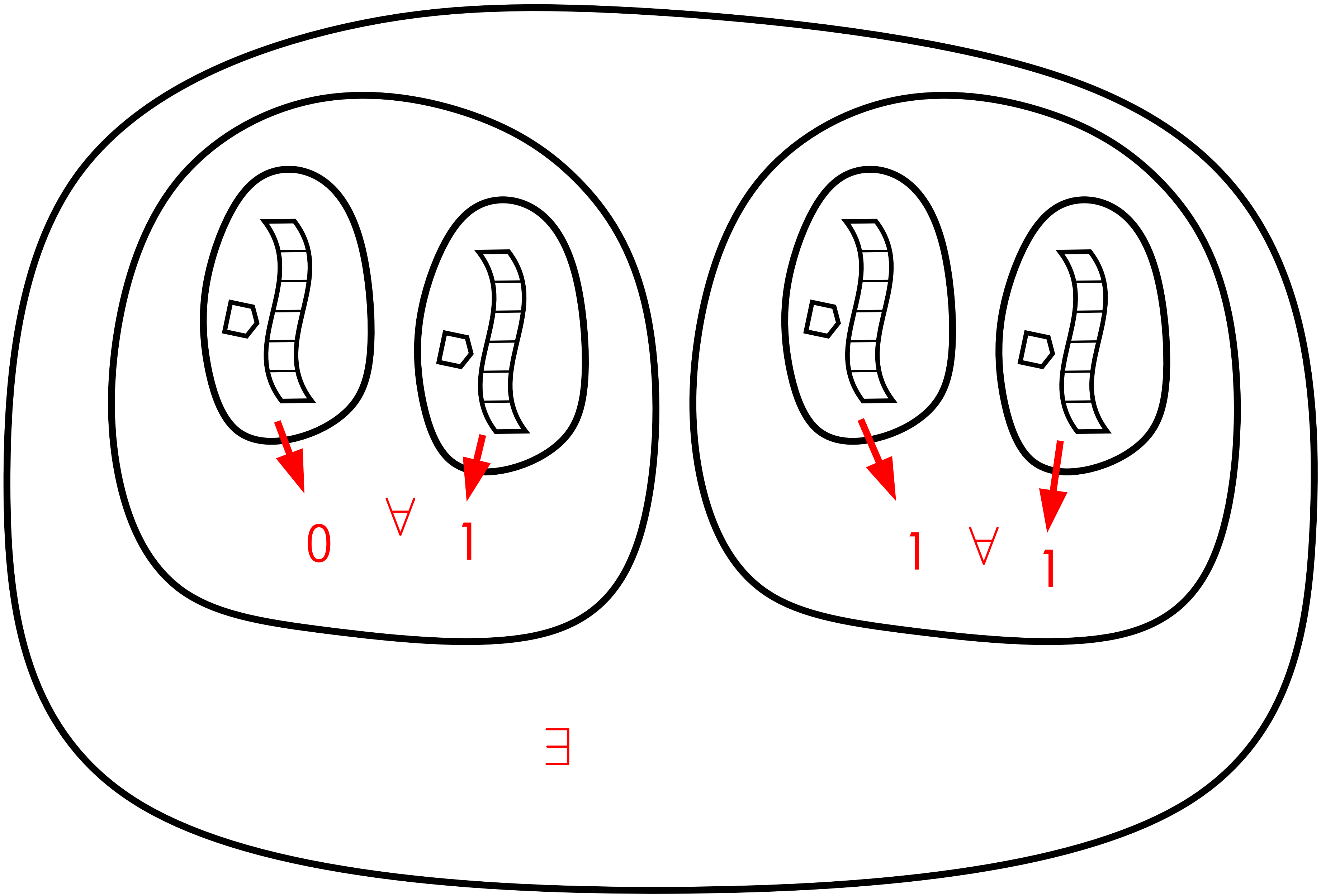
Simulating alternating Turing machines



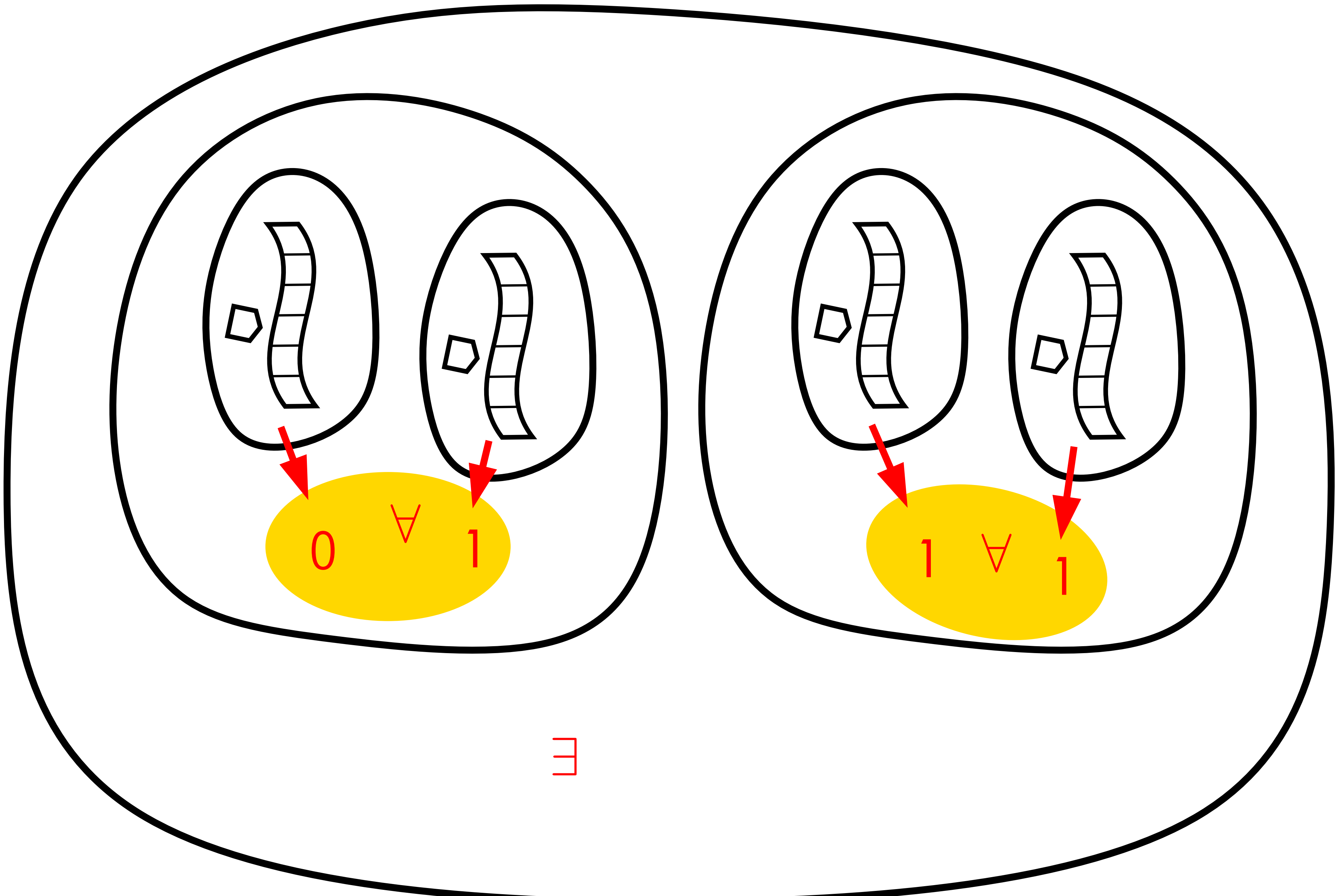
Simulating alternating Turing machines



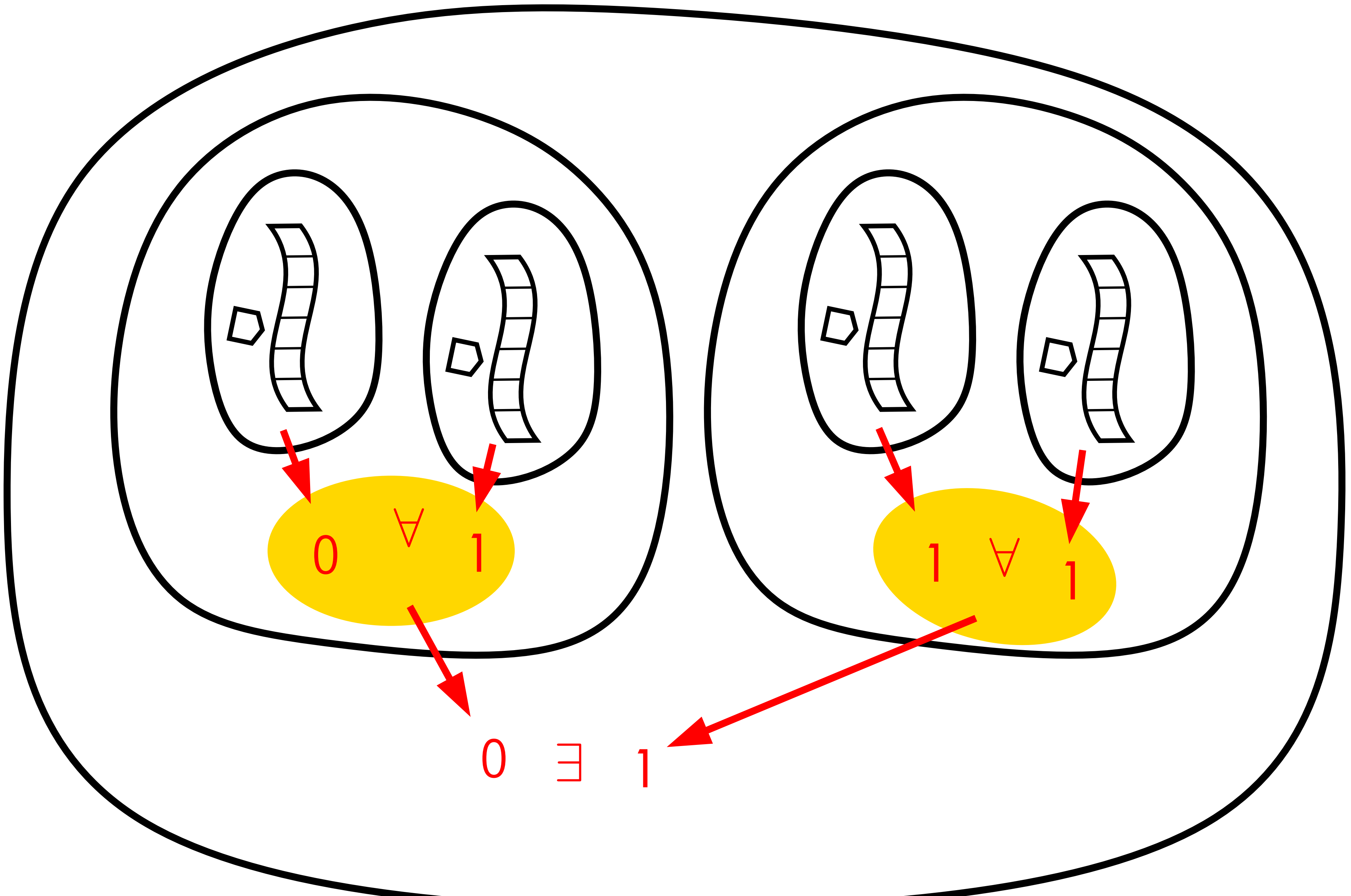
Simulating alternating Turing machines



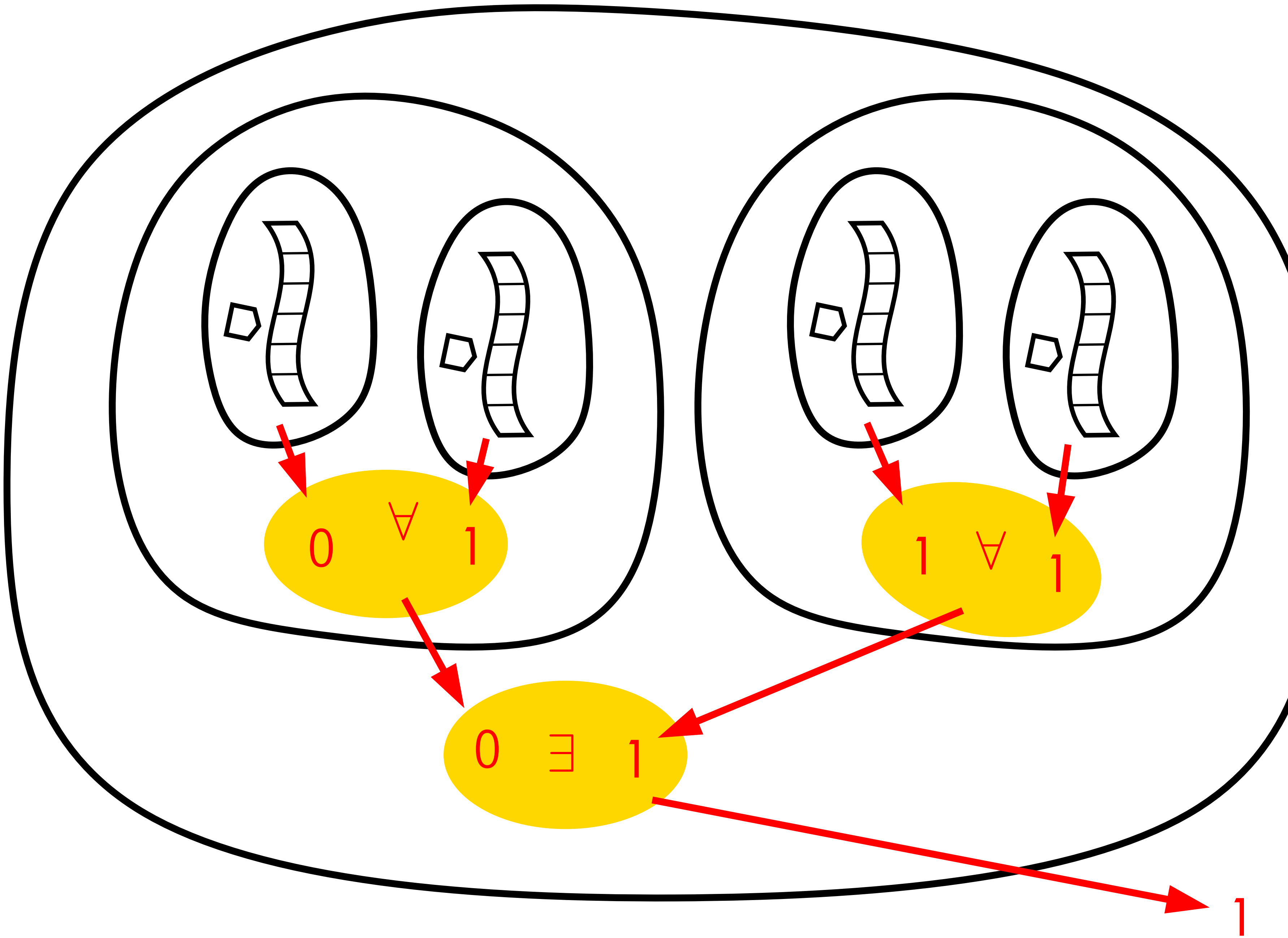
Simulating alternating Turing machines



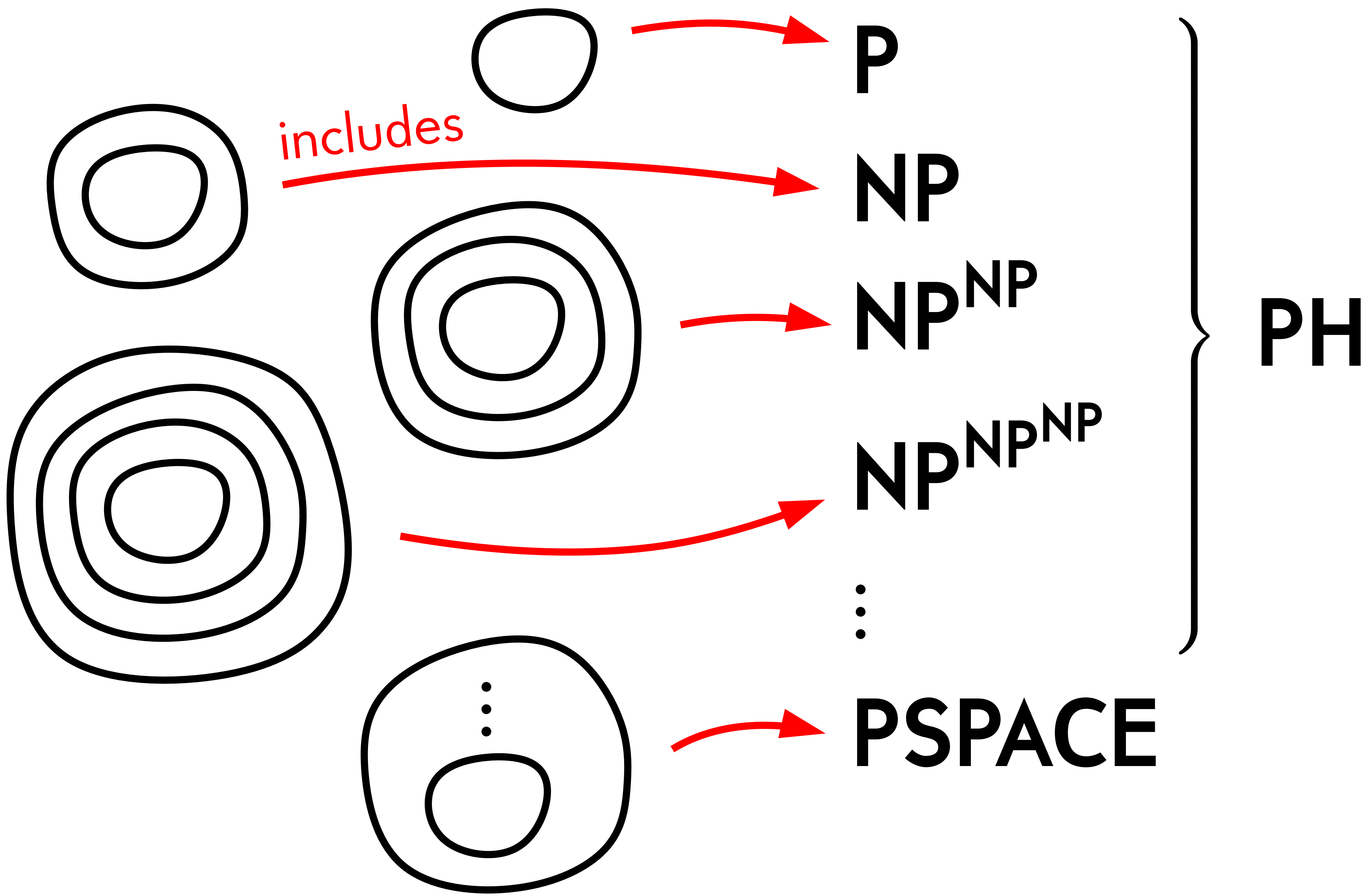
Simulating alternating Turing machines



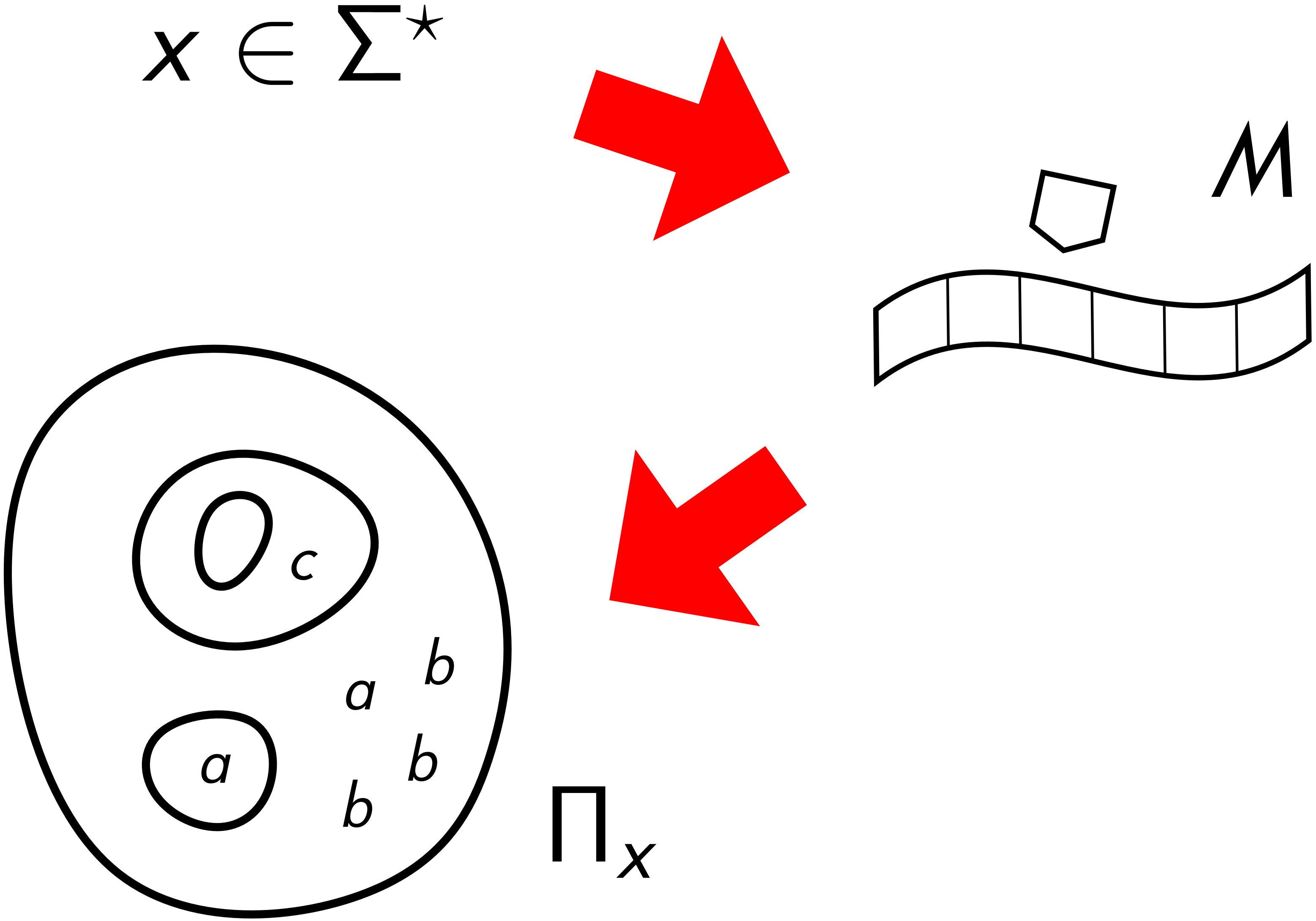
Simulating alternating Turing machines



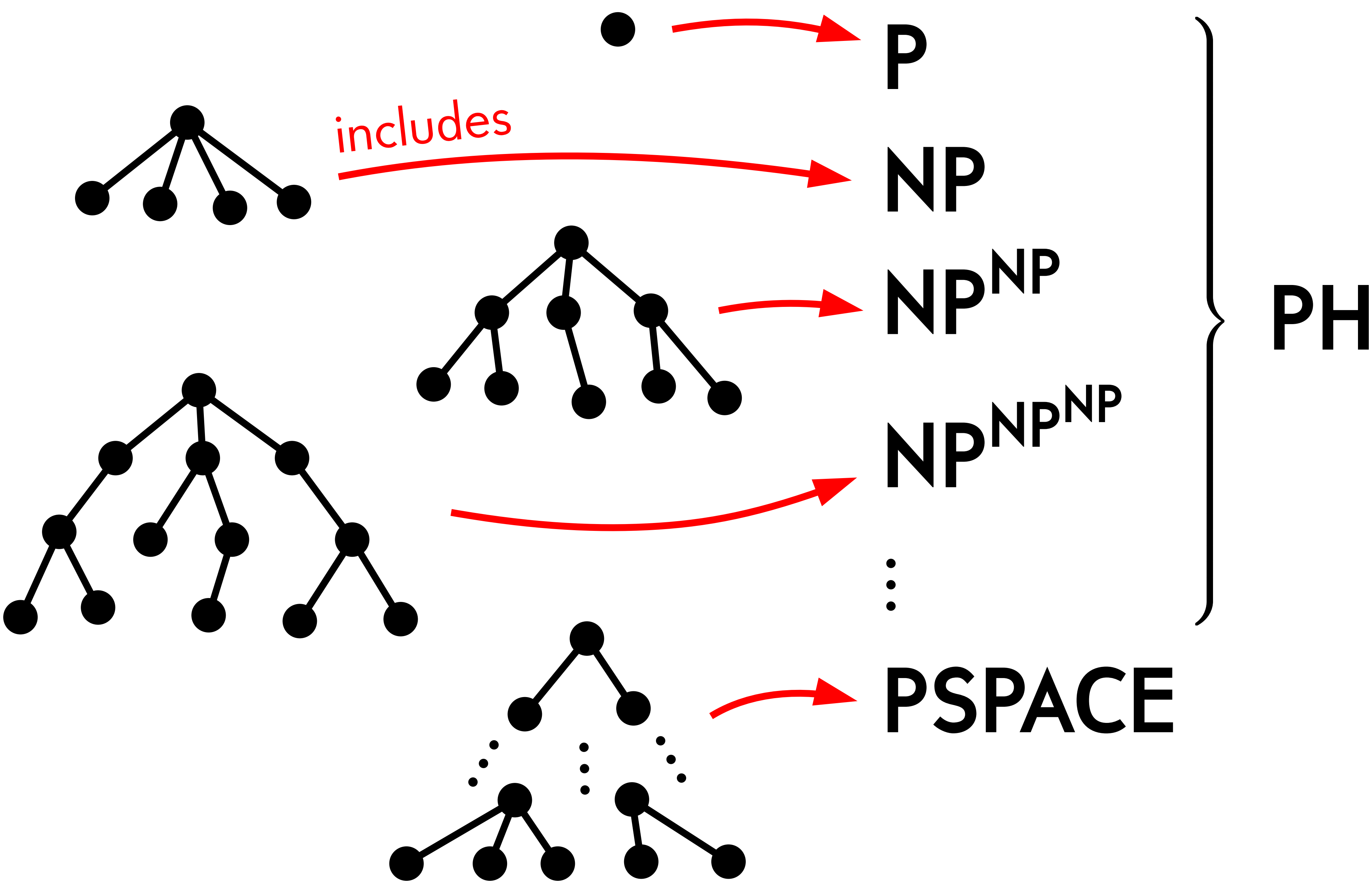
The polynomial hierarchy



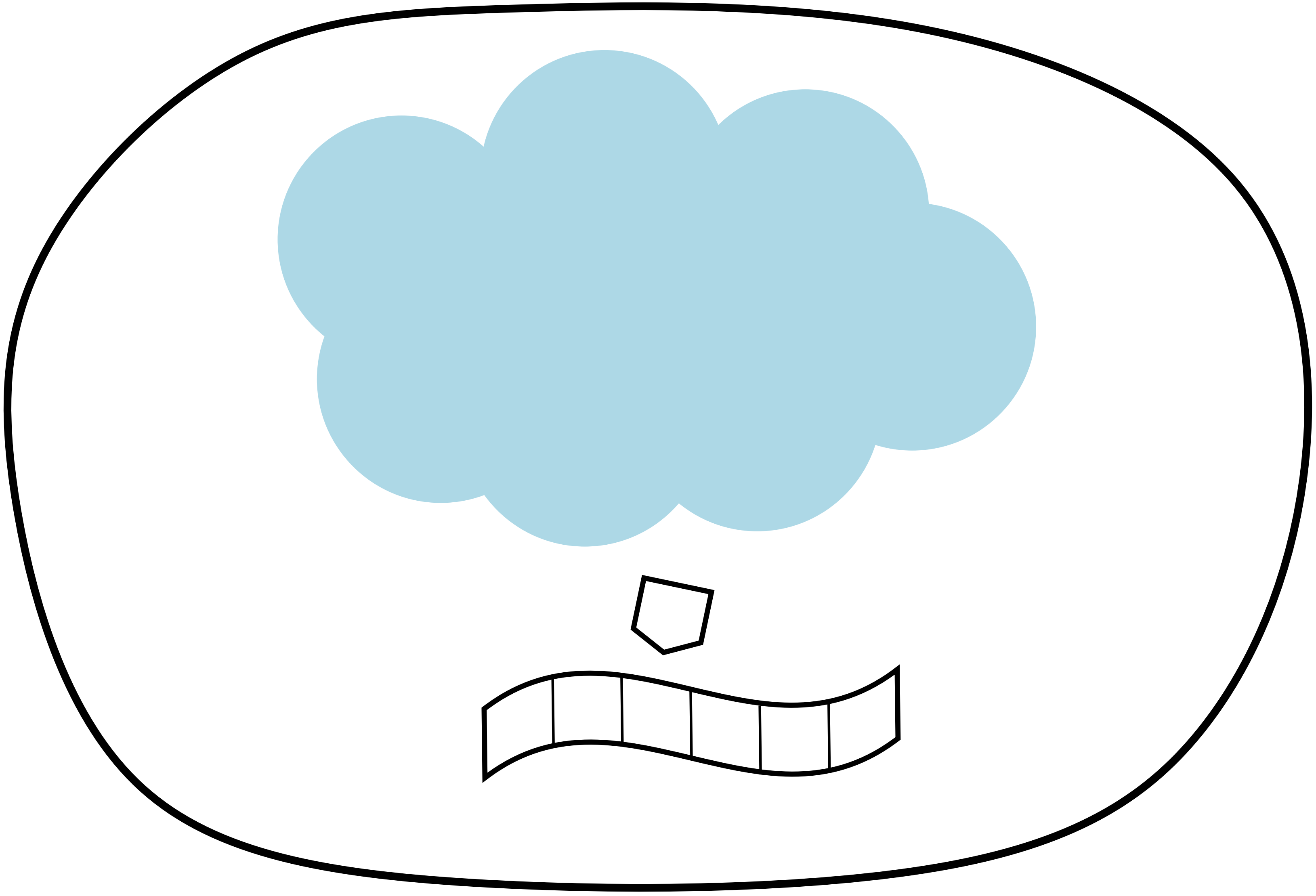
(Semi-)uniform families of membrane systems



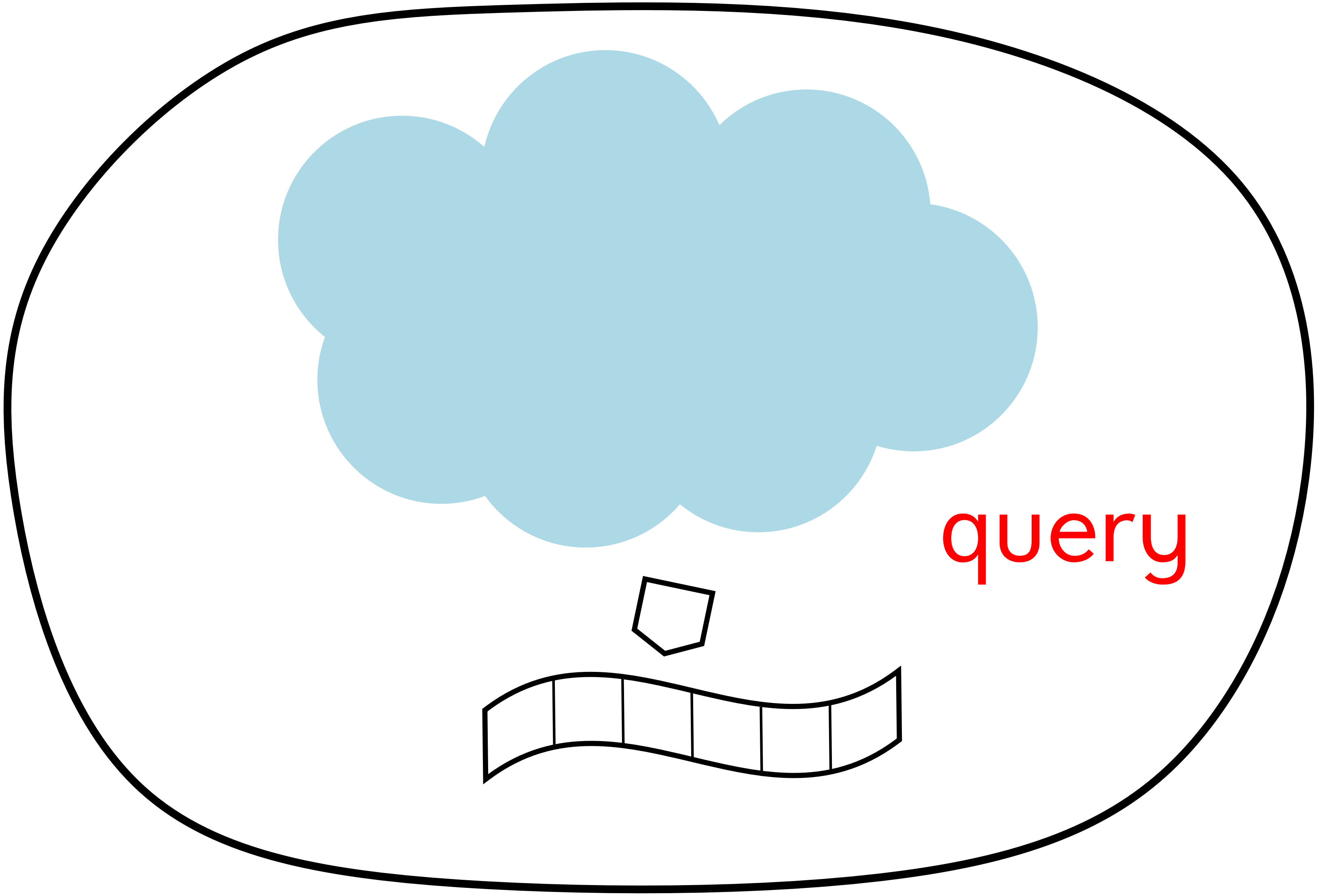
The polynomial hierarchy



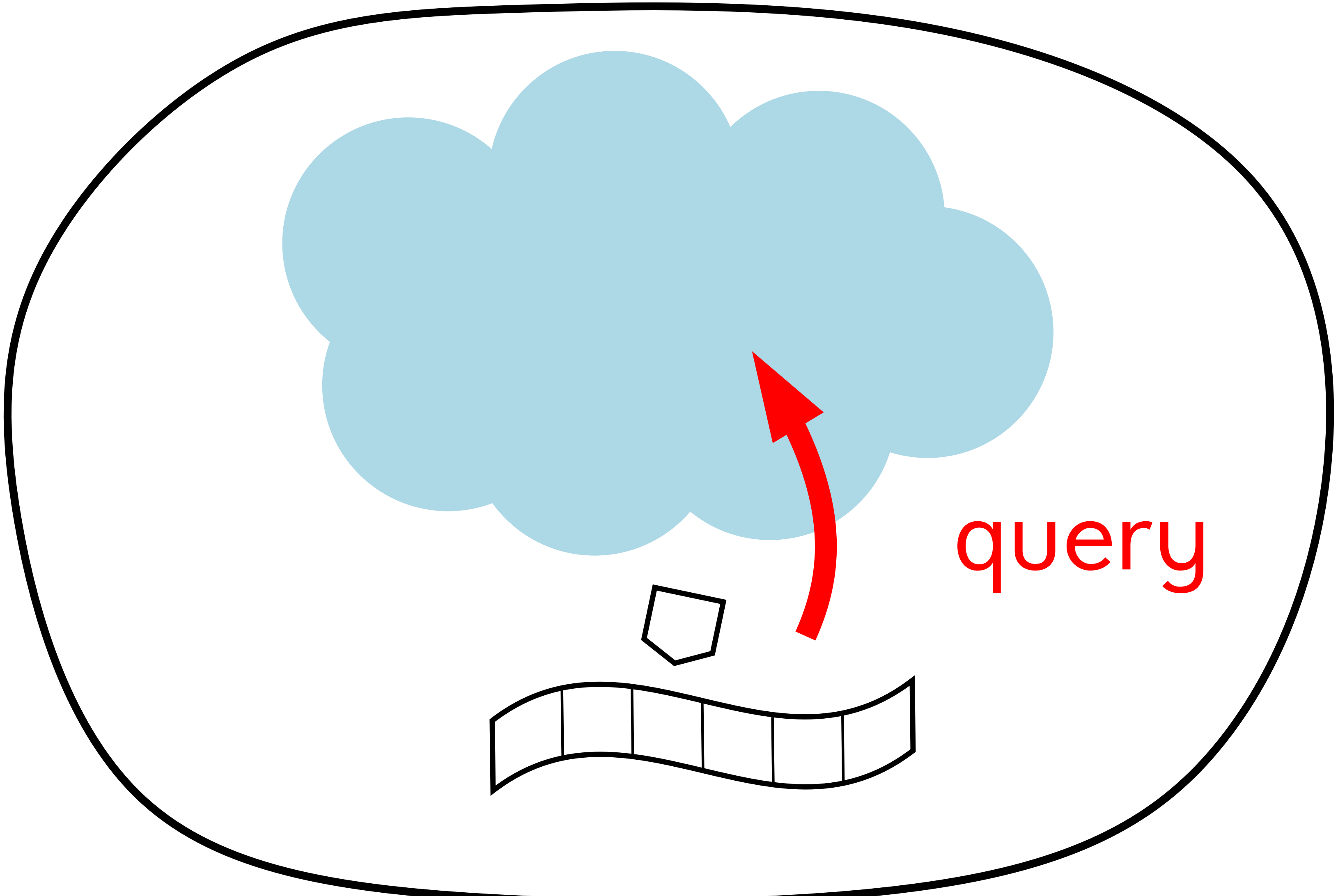
Simulating Turing machines with oracles



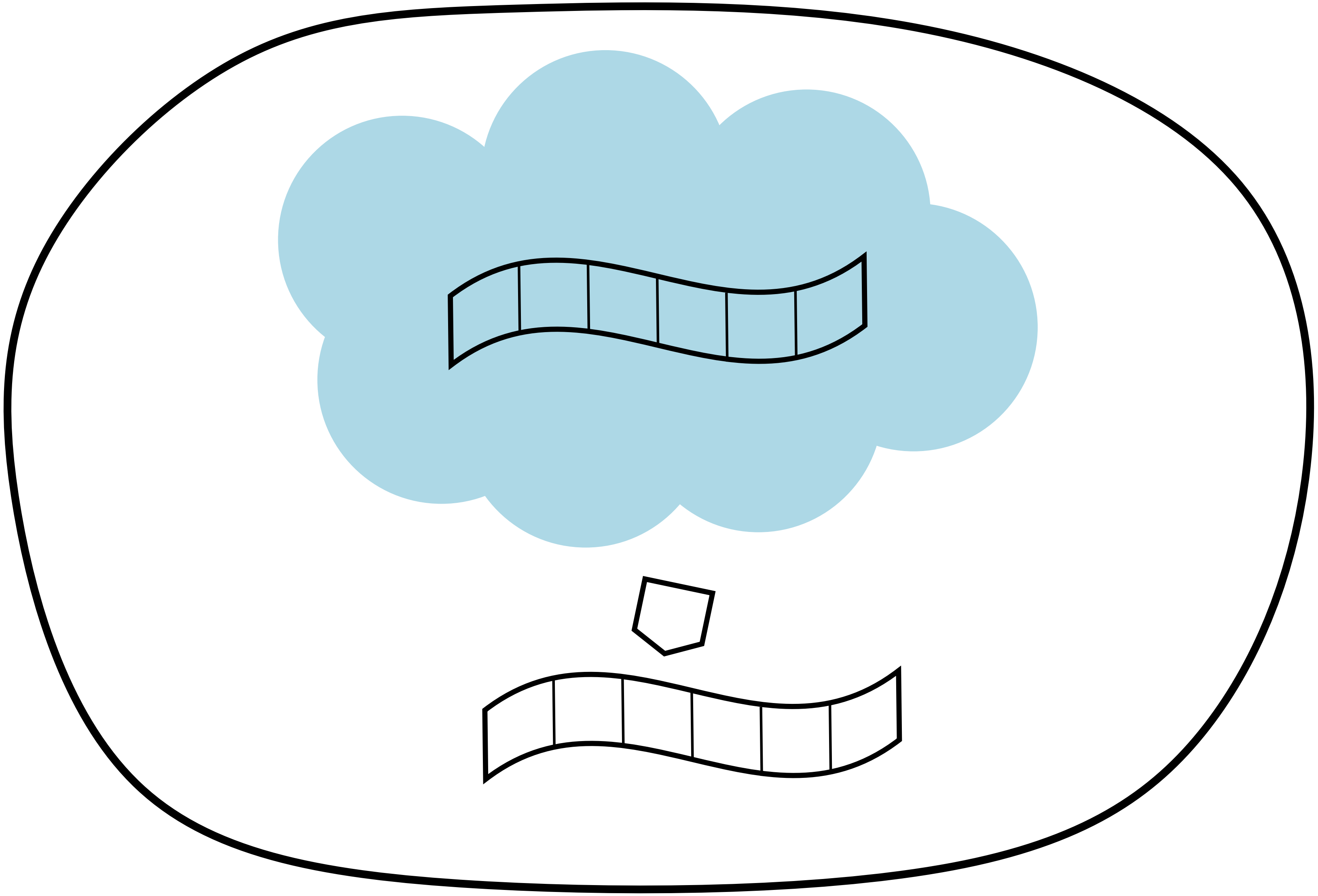
Simulating Turing machines with oracles



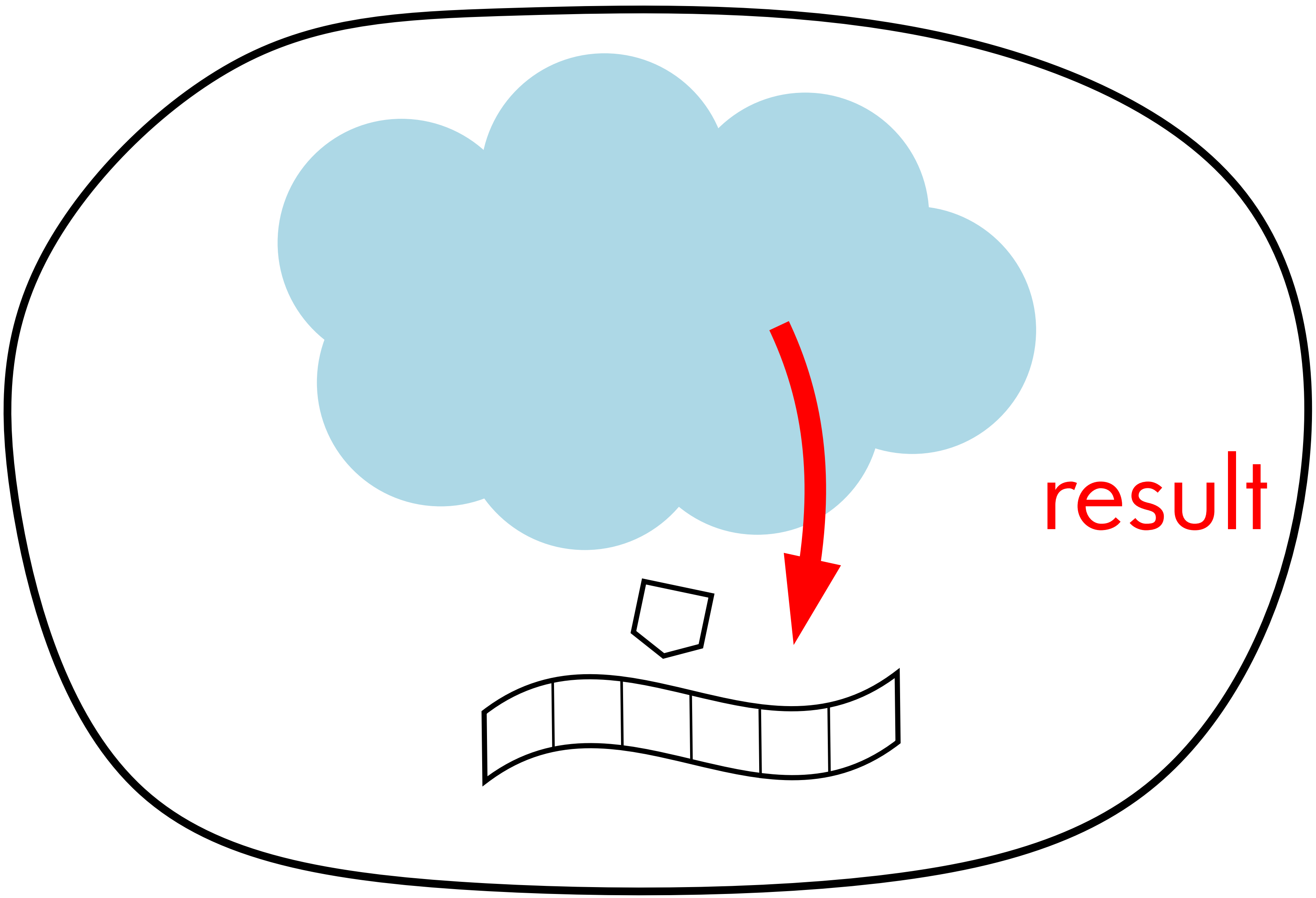
Simulating Turing machines with oracles



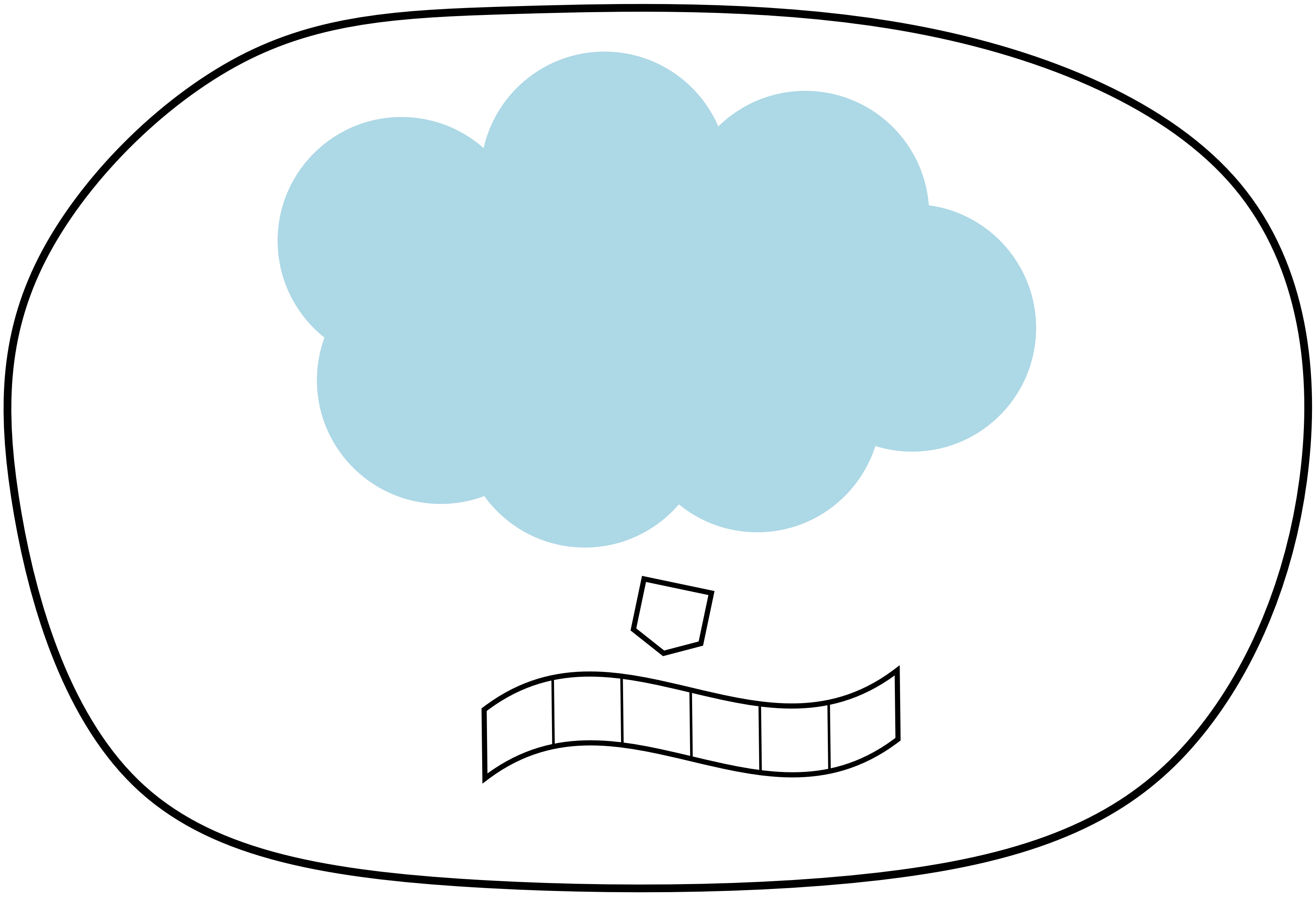
Simulating Turing machines with oracles



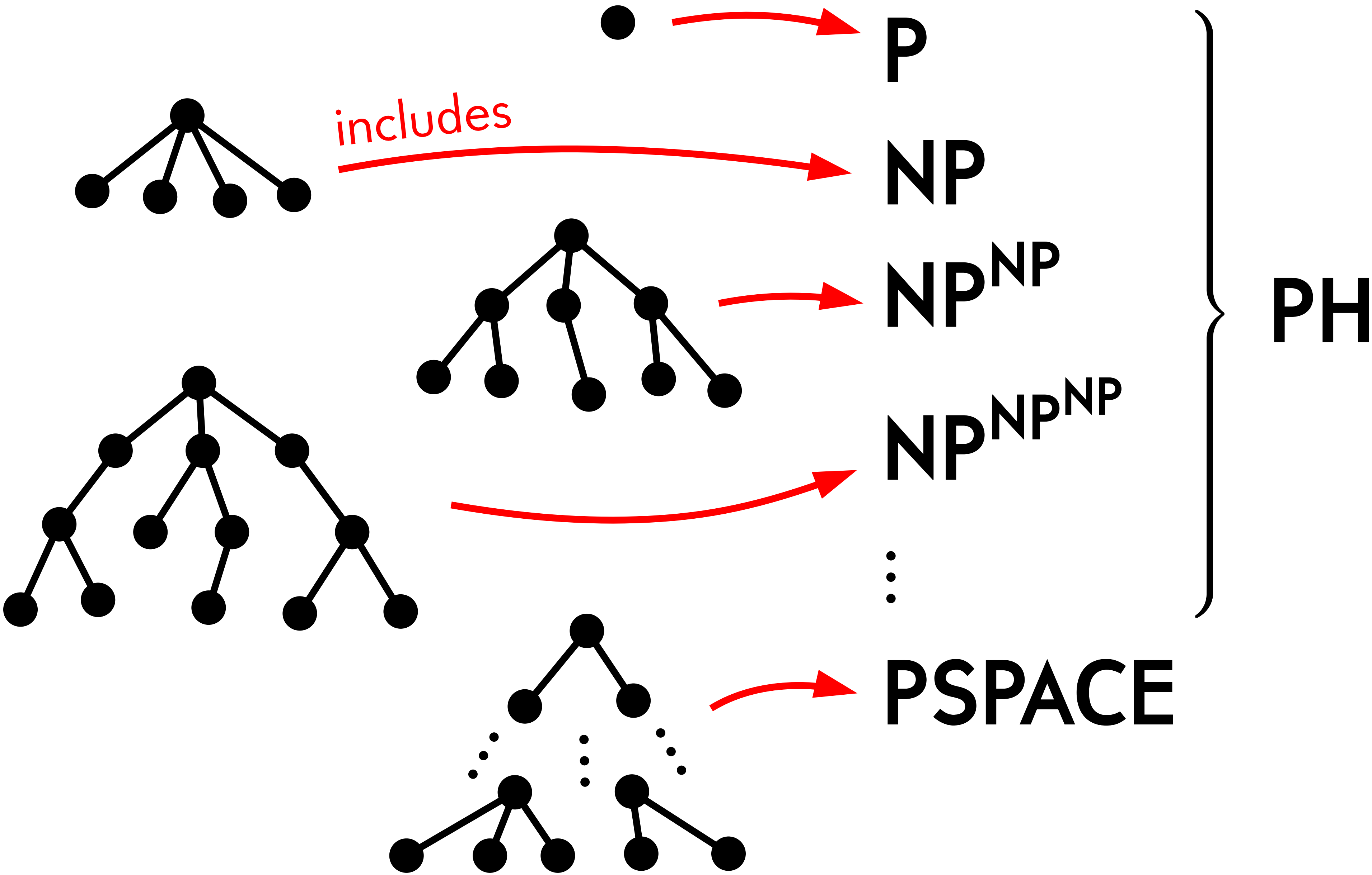
Simulating Turing machines with oracles



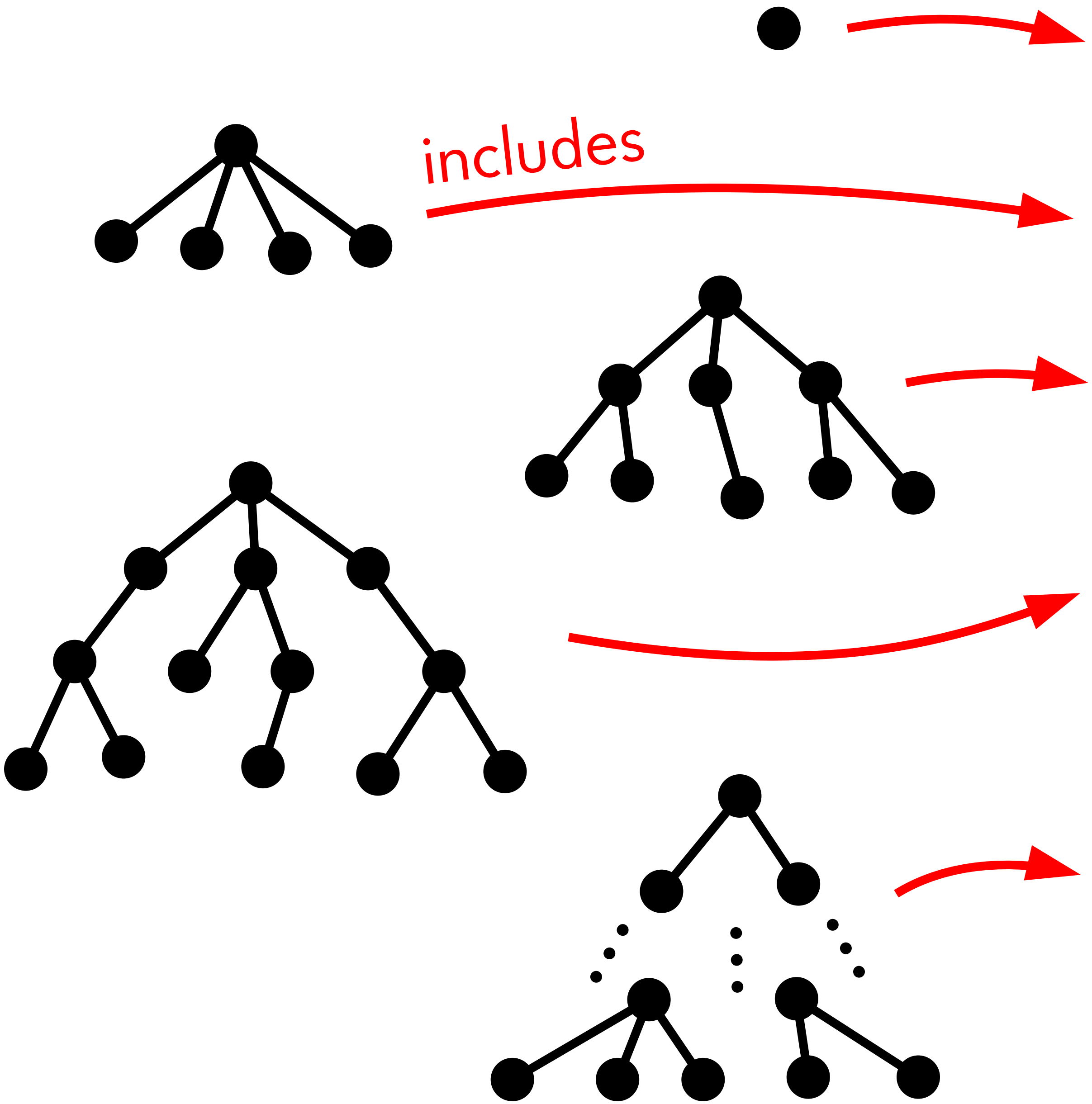
Simulating Turing machines with oracles



The polynomial hierarchy



The polynomial hierarchy



$$P^{P} = P$$

$$P^{NP} \supseteq NP$$

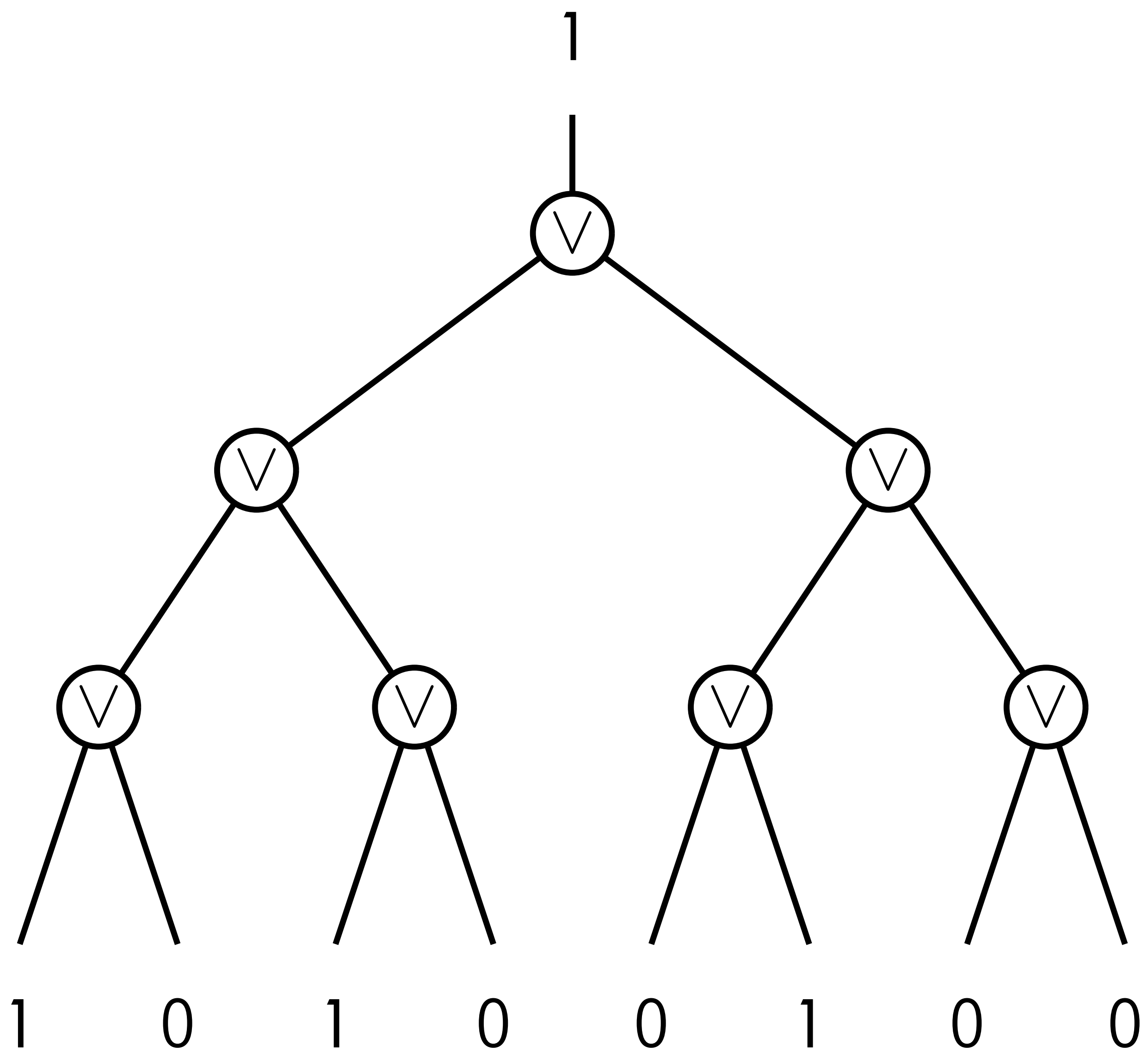
$$P^{NP^{NP}} \supseteq NP^{NP}$$

$$P^{NP^{NP^{NP}}} \supseteq NP^{NP^{NP}}$$

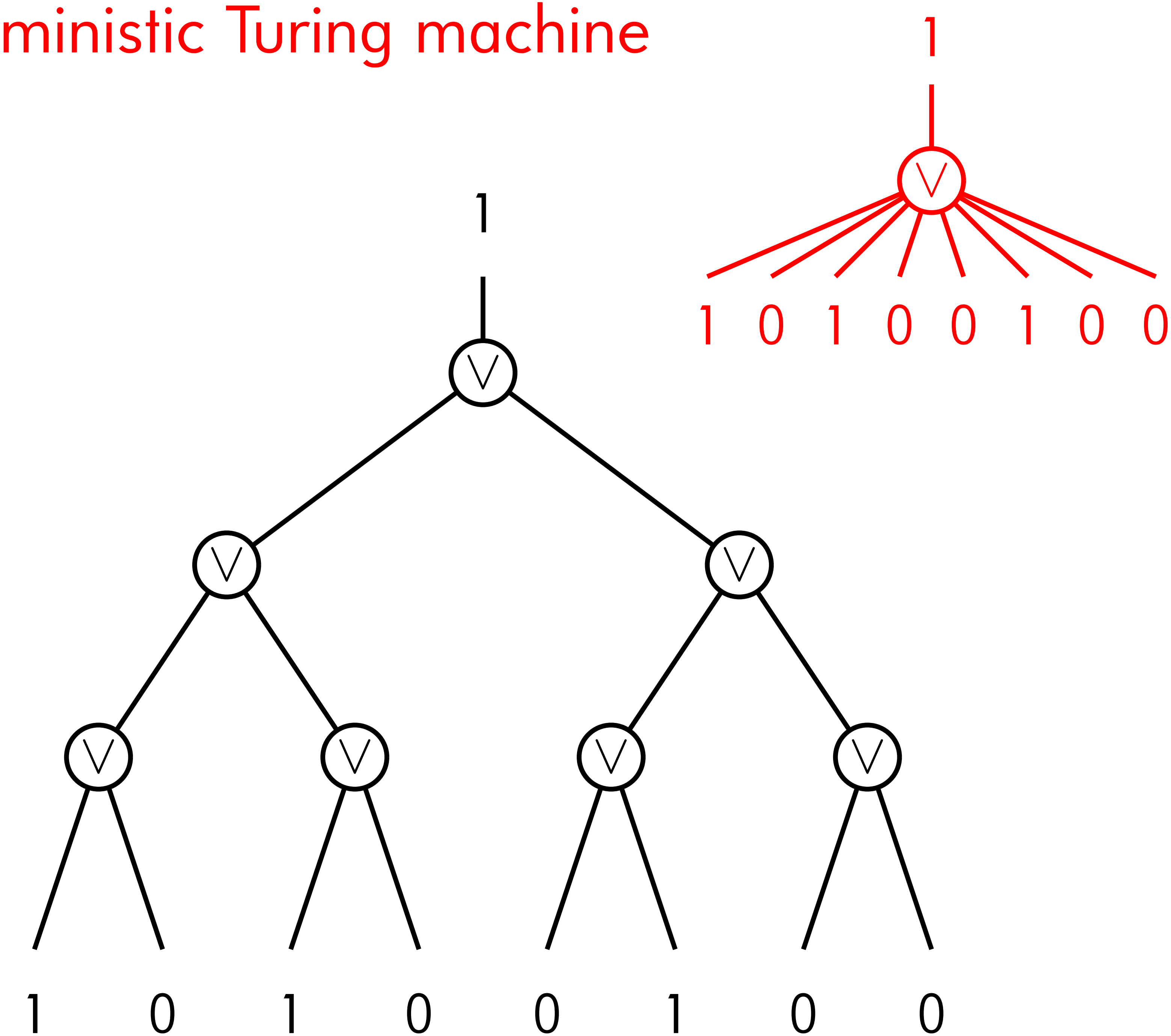
$$\vdots$$

$$P^{PSPACE} = PSPACE$$

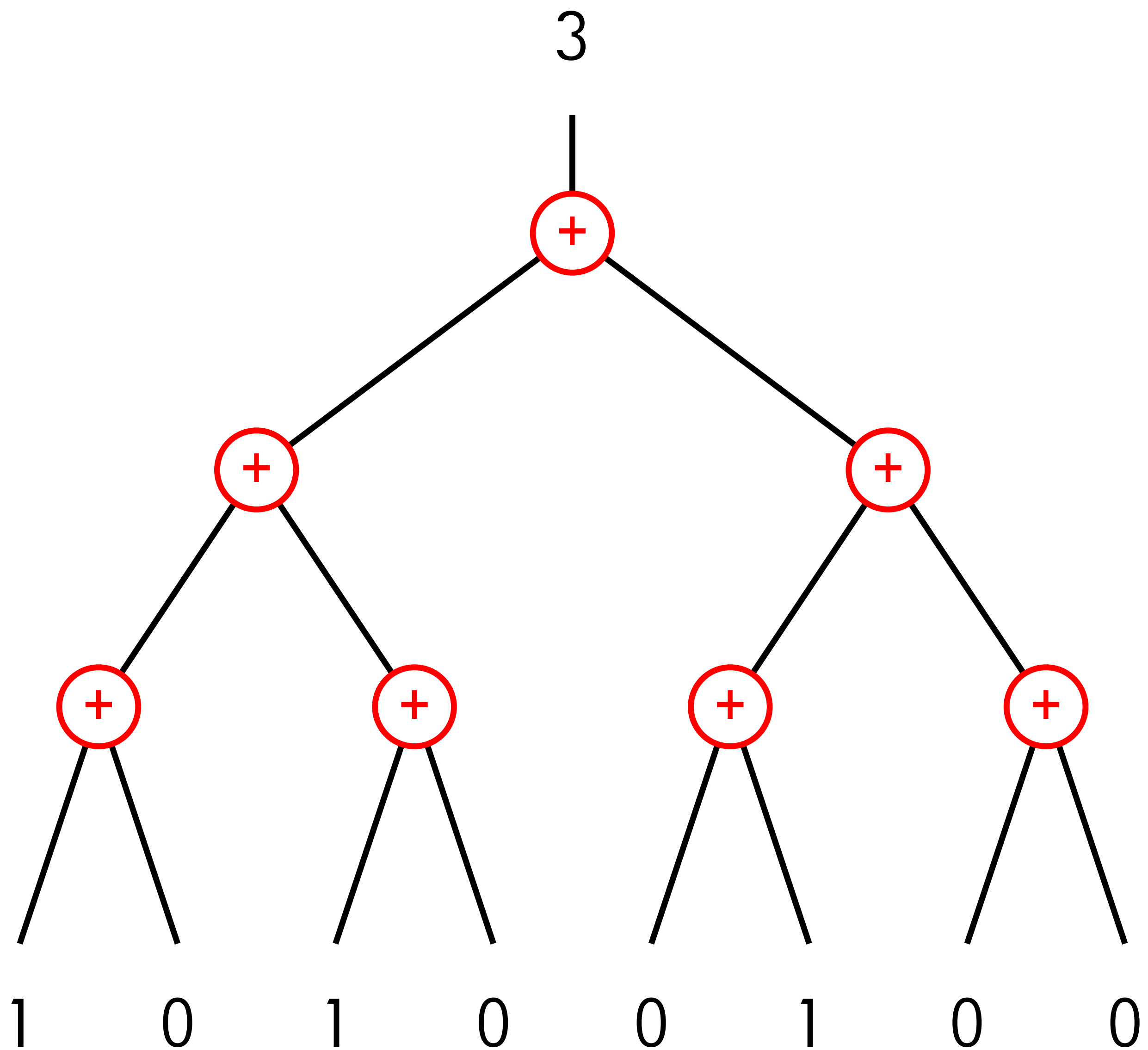
Nondeterministic Turing machine



Nondeterministic Turing machine

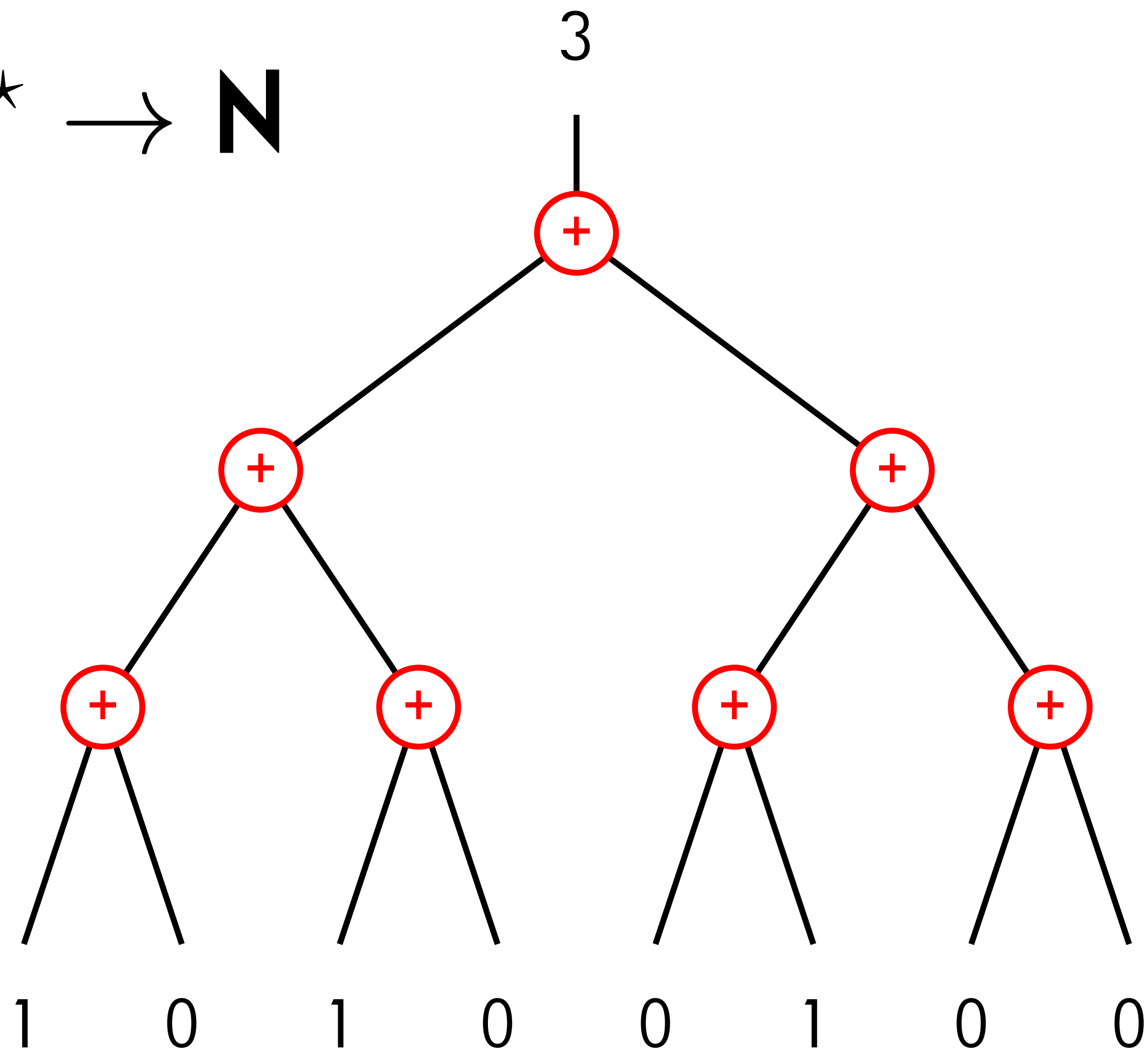


Counting Turing machine



Counting Turing machine

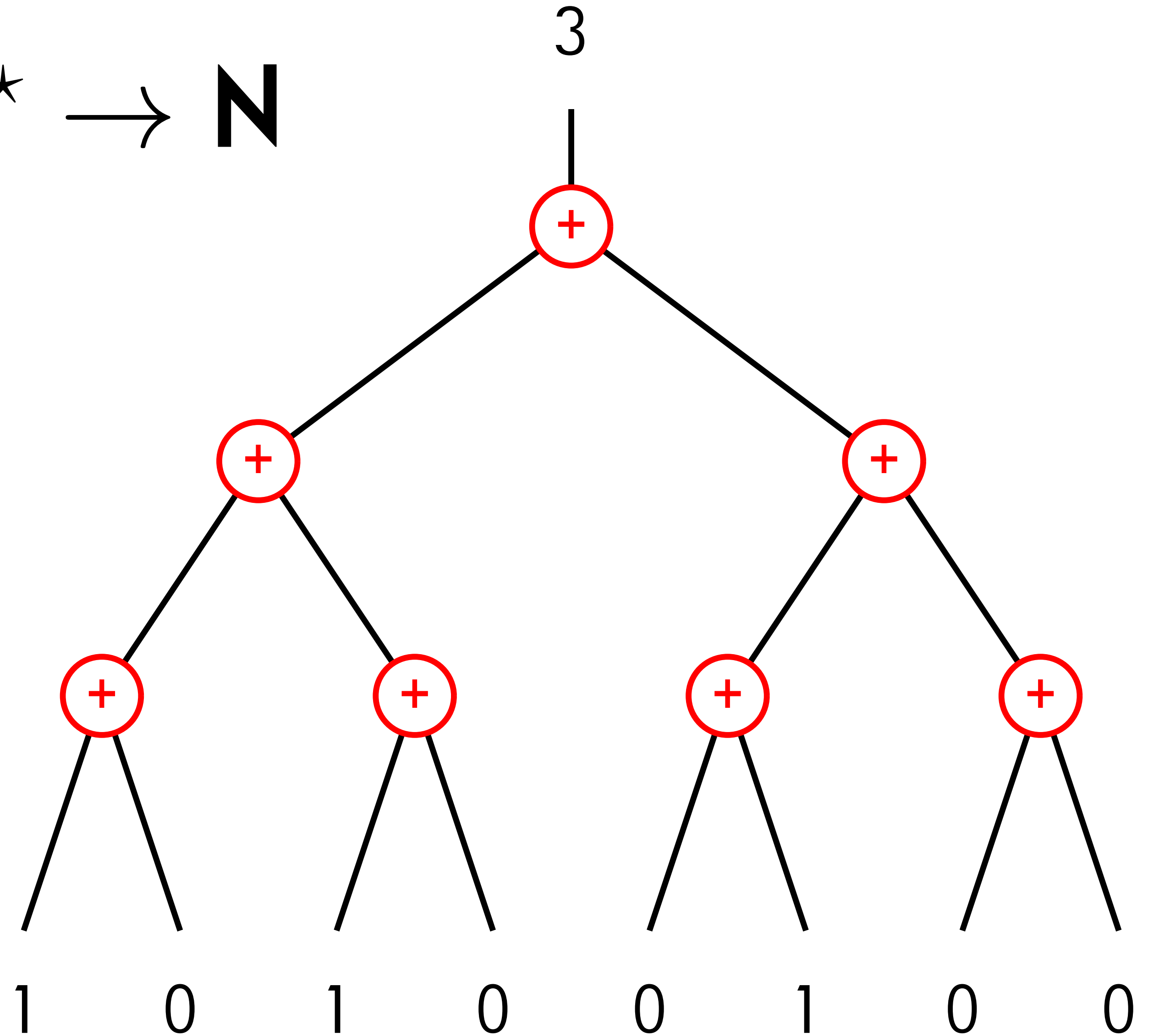
$$f : \Sigma^{\star} \to \mathbf{N}$$



Counting Turing machine

$f : \Sigma^\star \to \mathbf{N}$

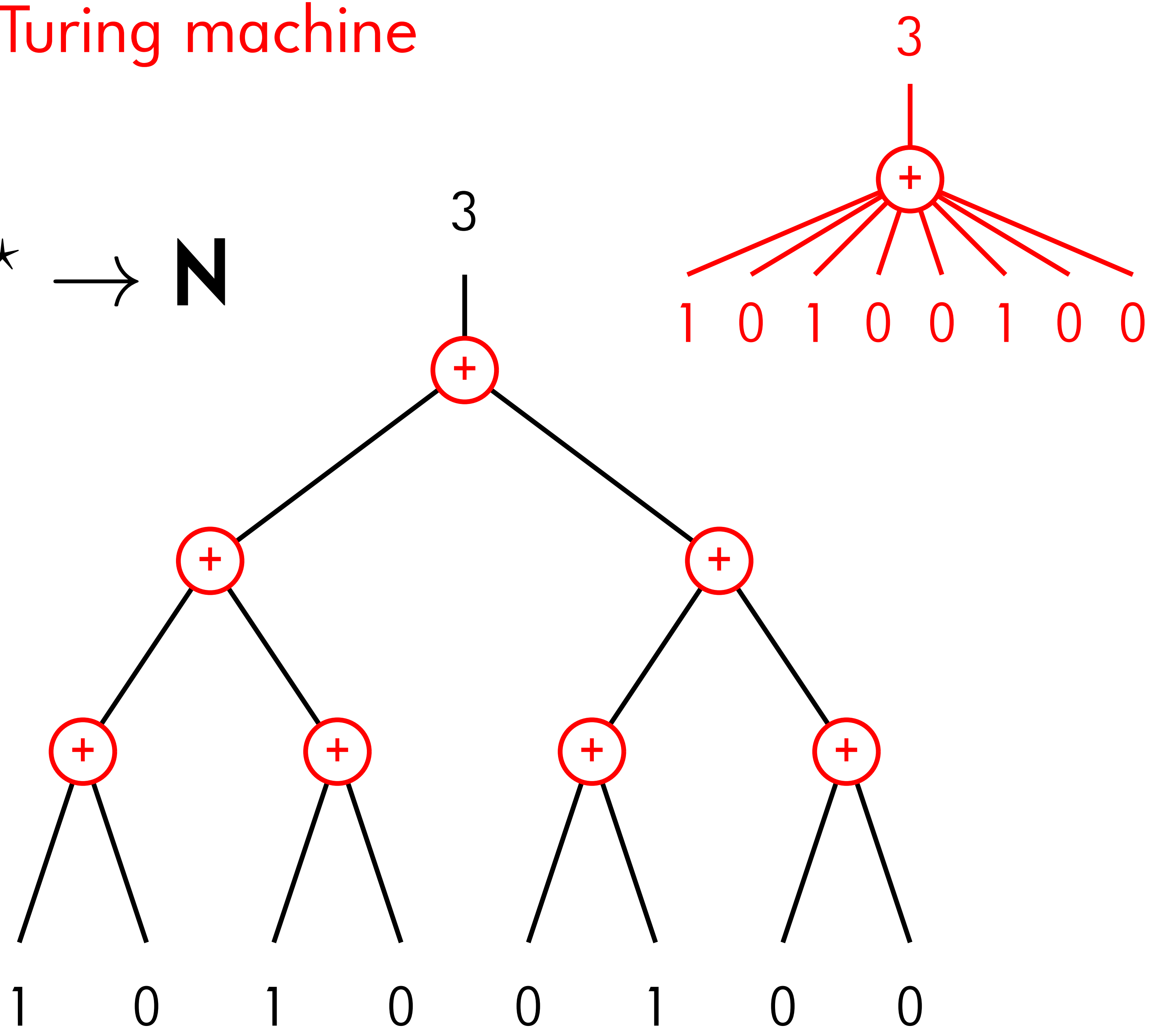
#P



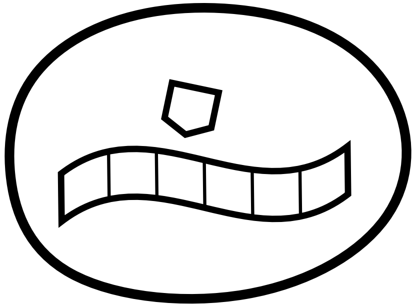
Counting Turing machine

$$f : \Sigma^\star \to \mathbf{N}$$

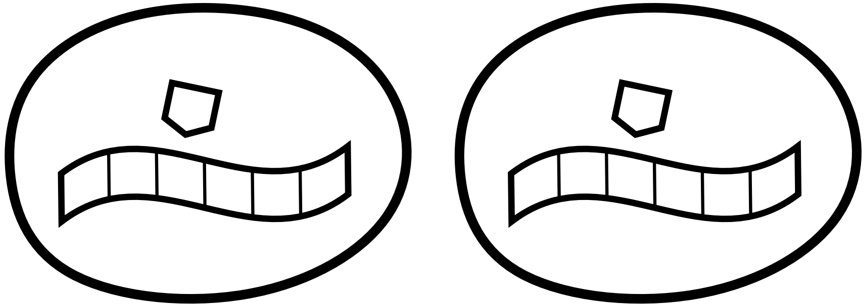
#P



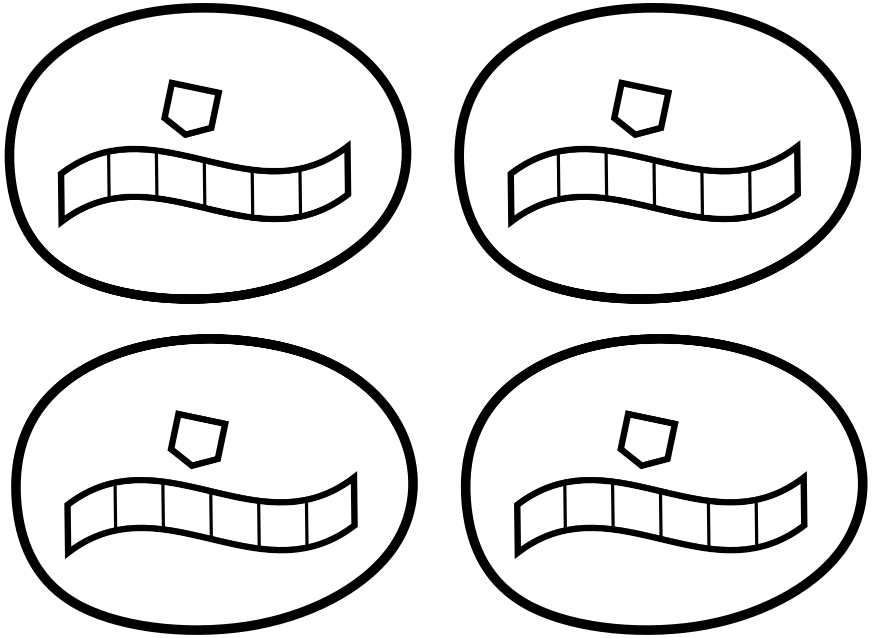
Simulating counting Turing machines



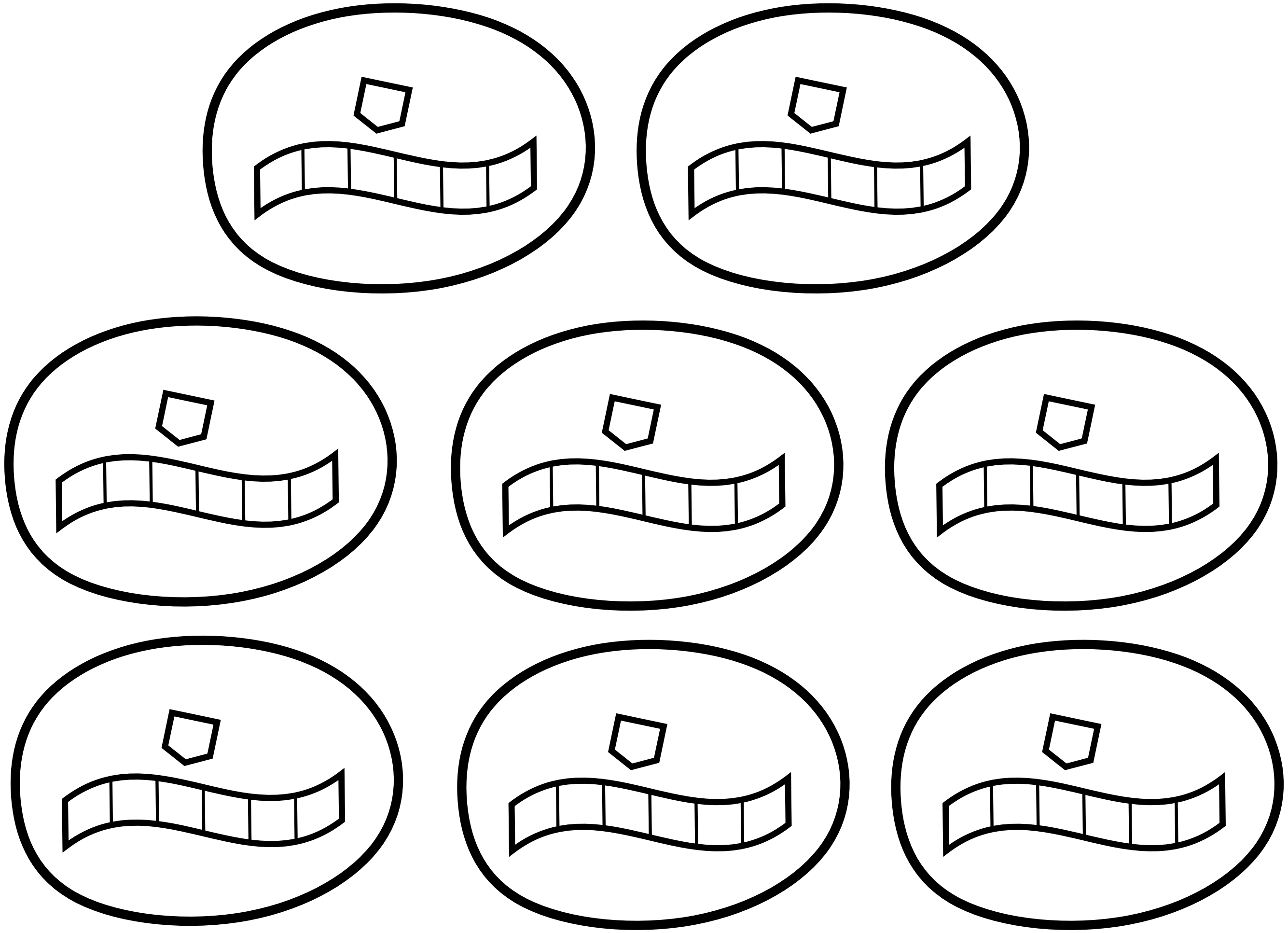
Simulating counting Turing machines



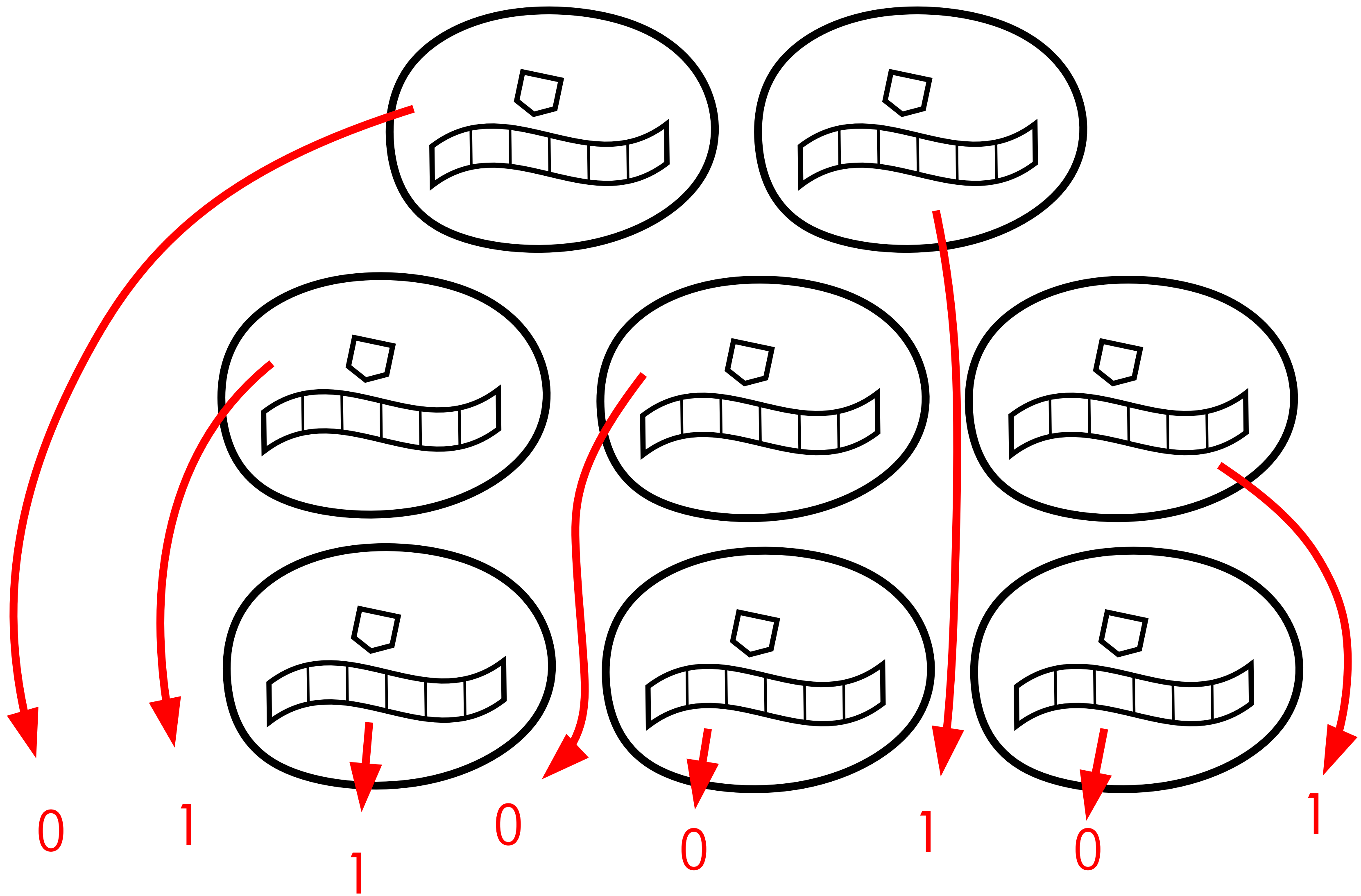
Simulating counting Turing machines



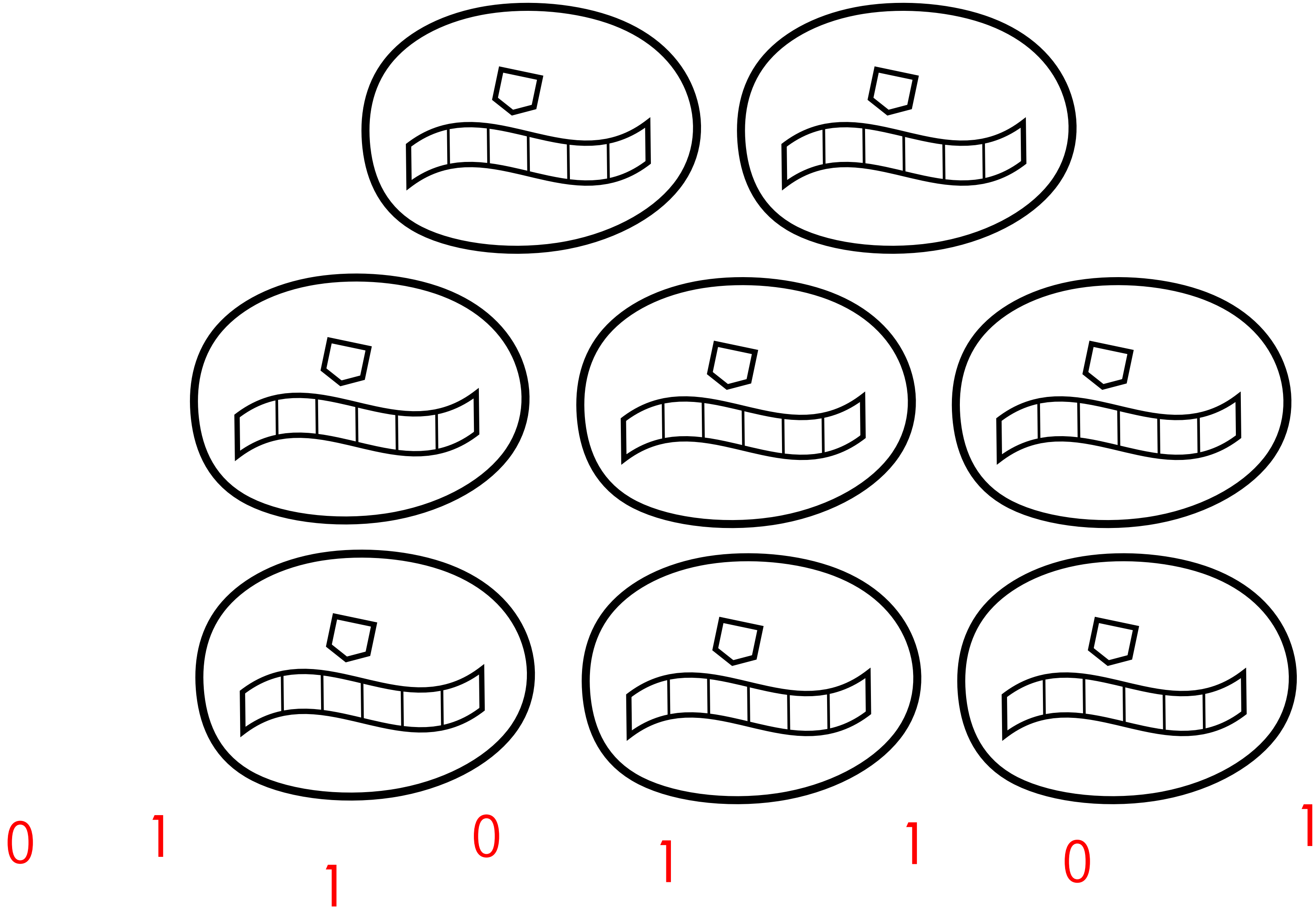
Simulating counting Turing machines



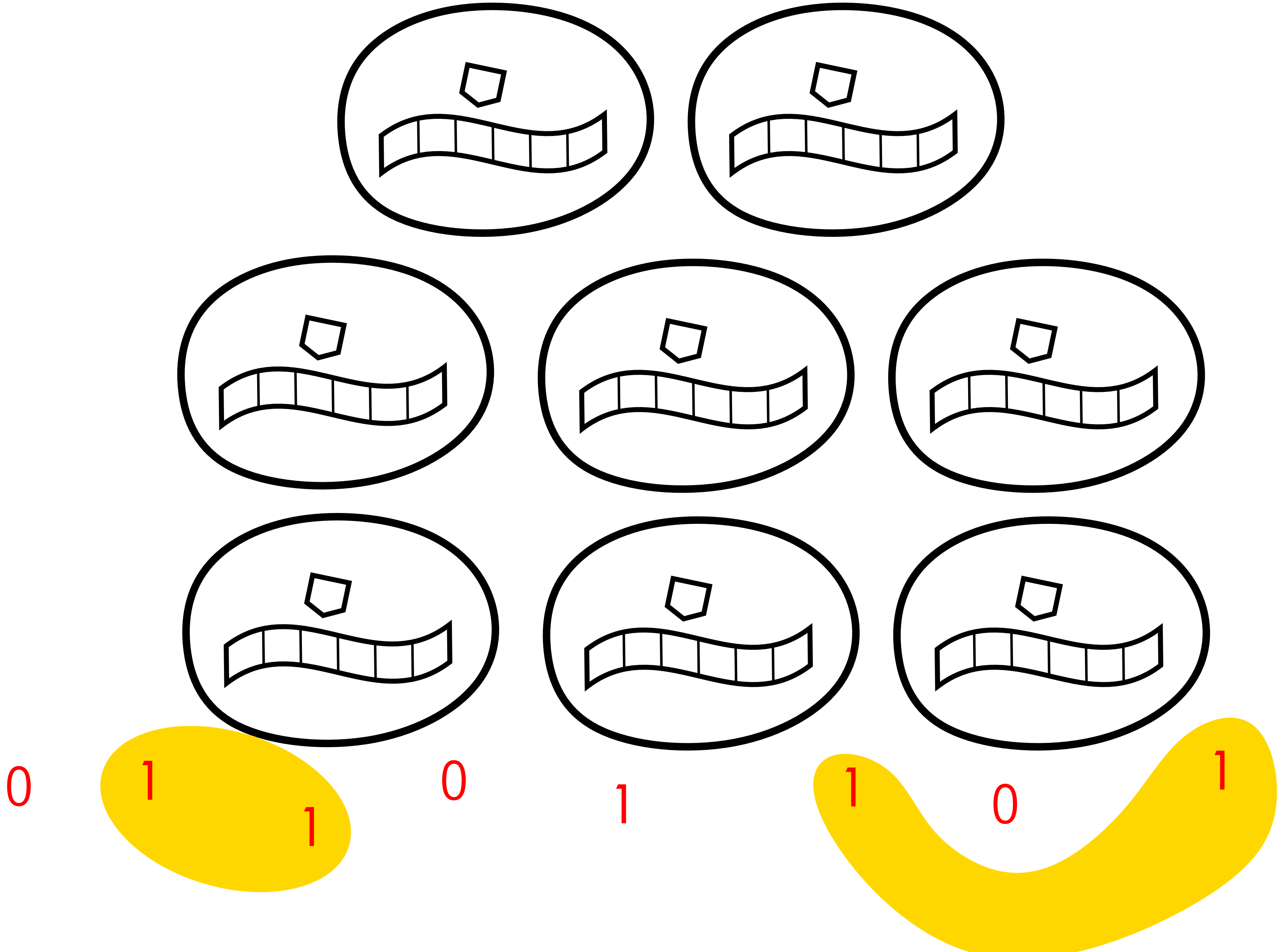
Simulating counting Turing machines



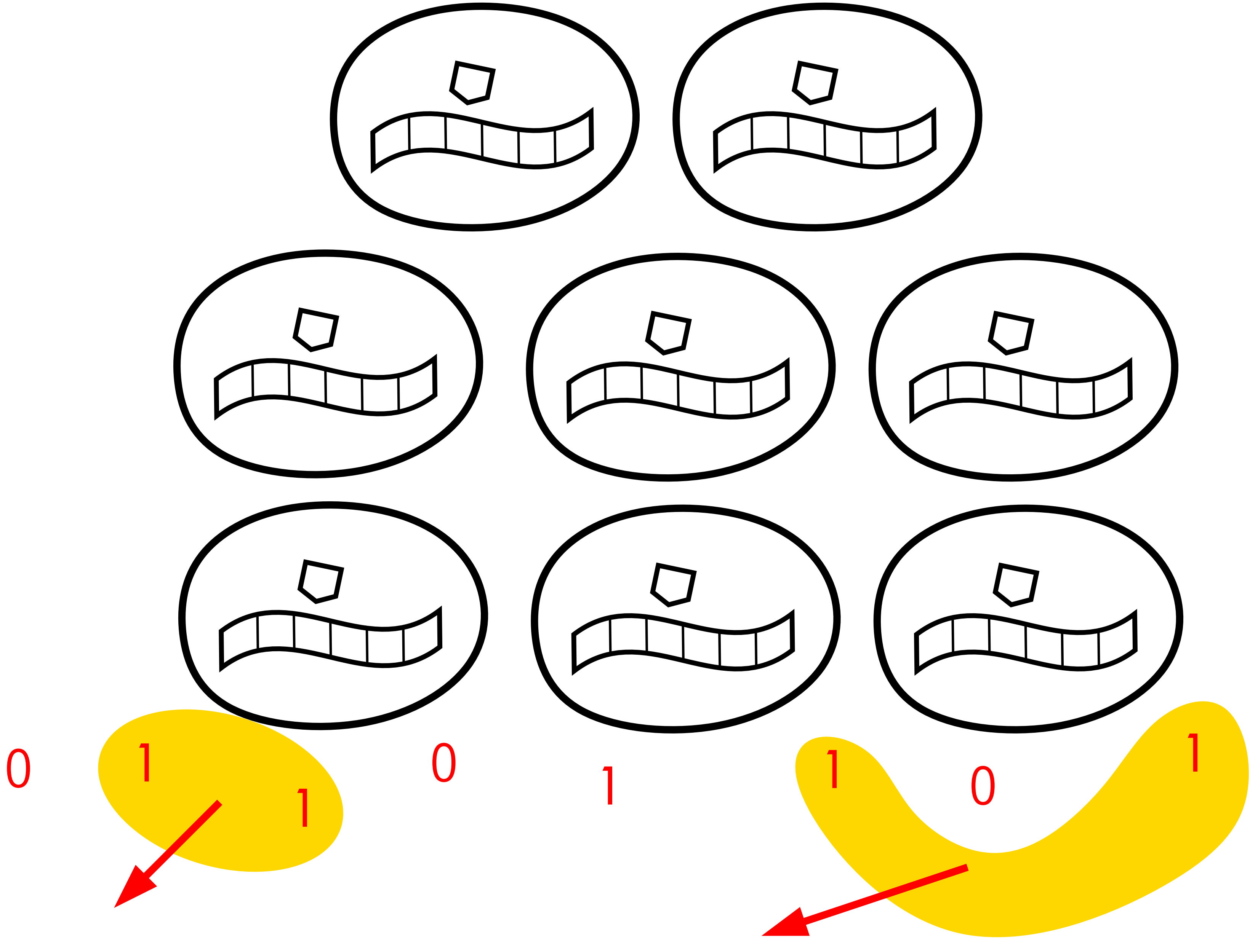
Simulating counting Turing machines



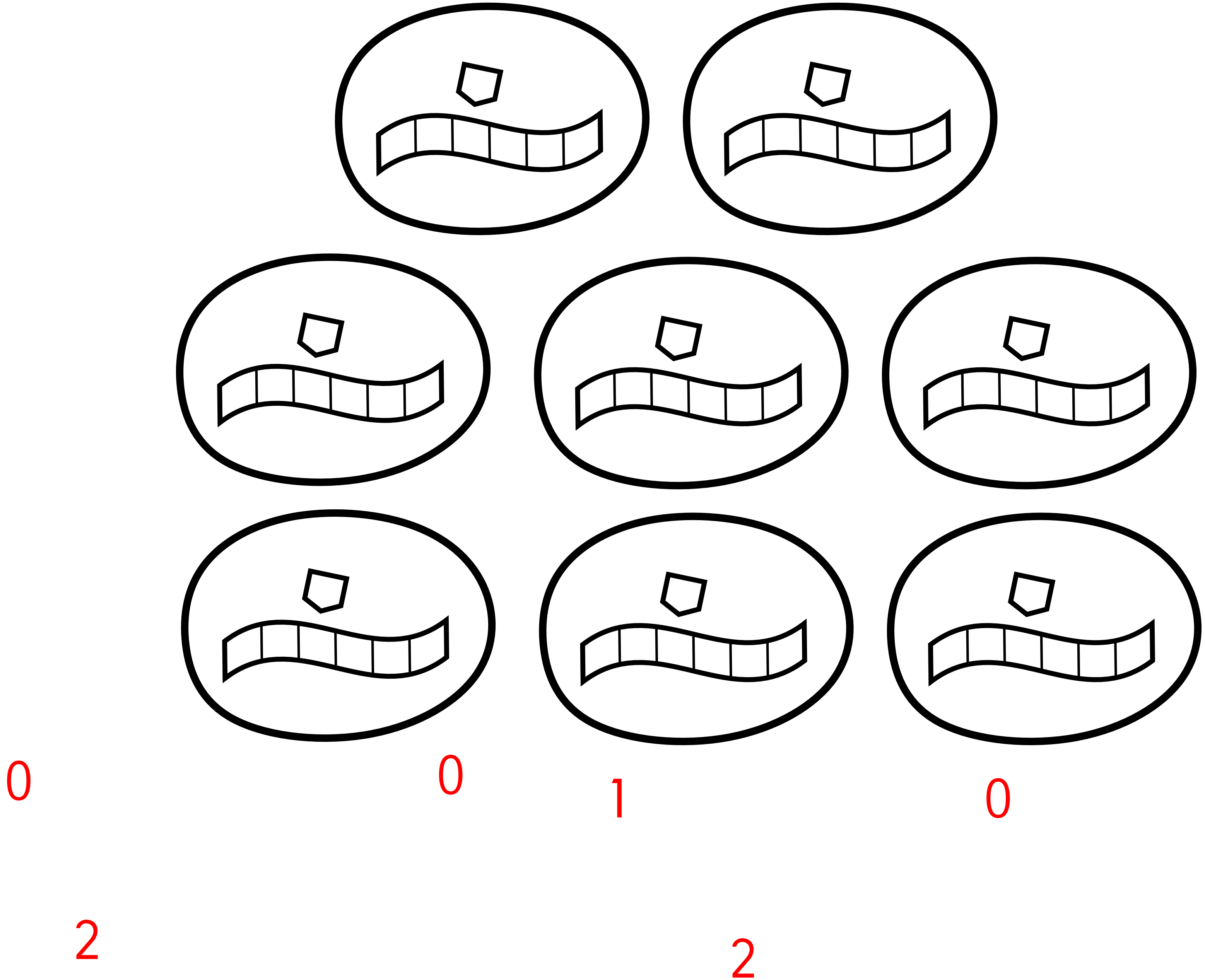
Simulating counting Turing machines



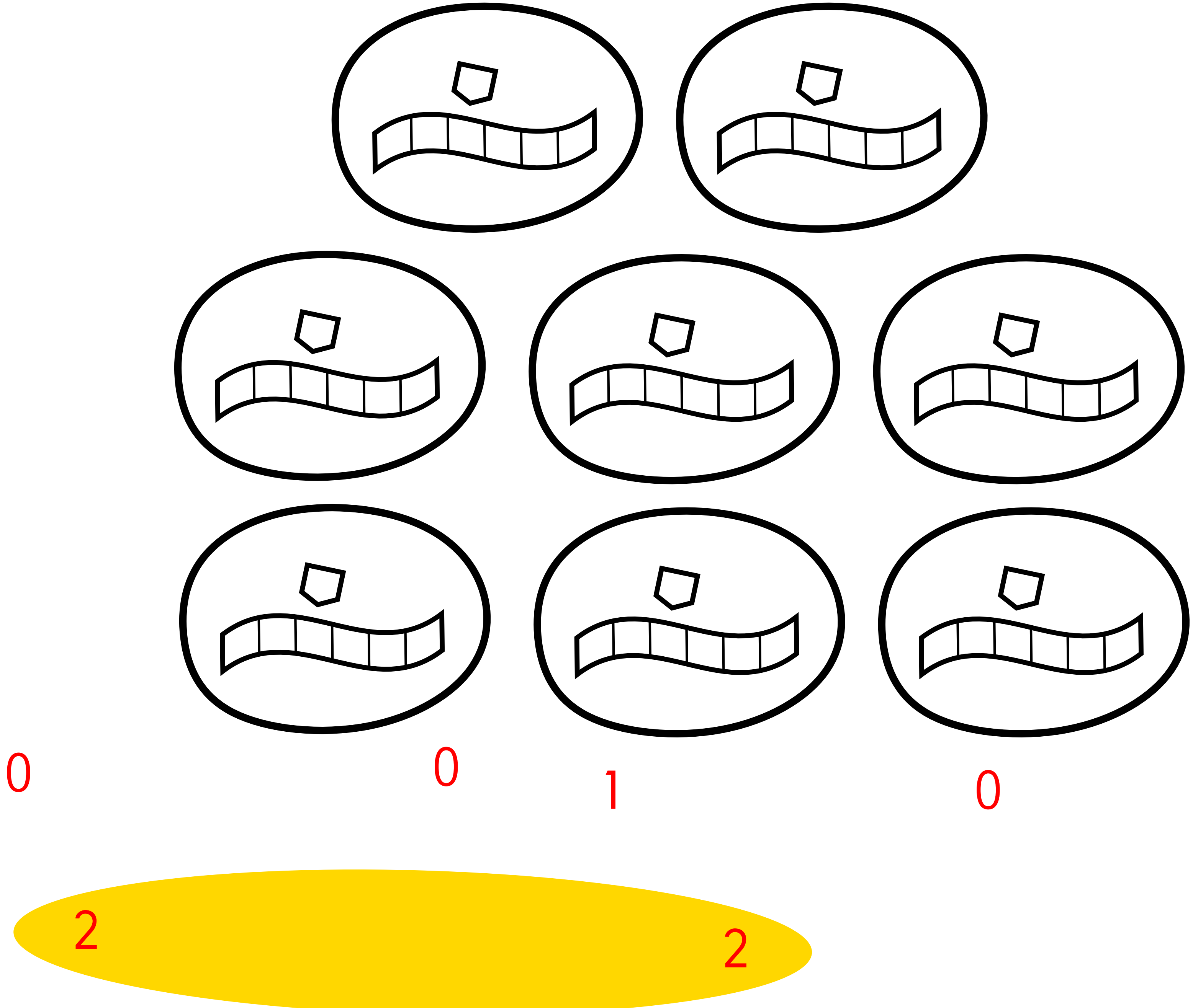
Simulating counting Turing machines



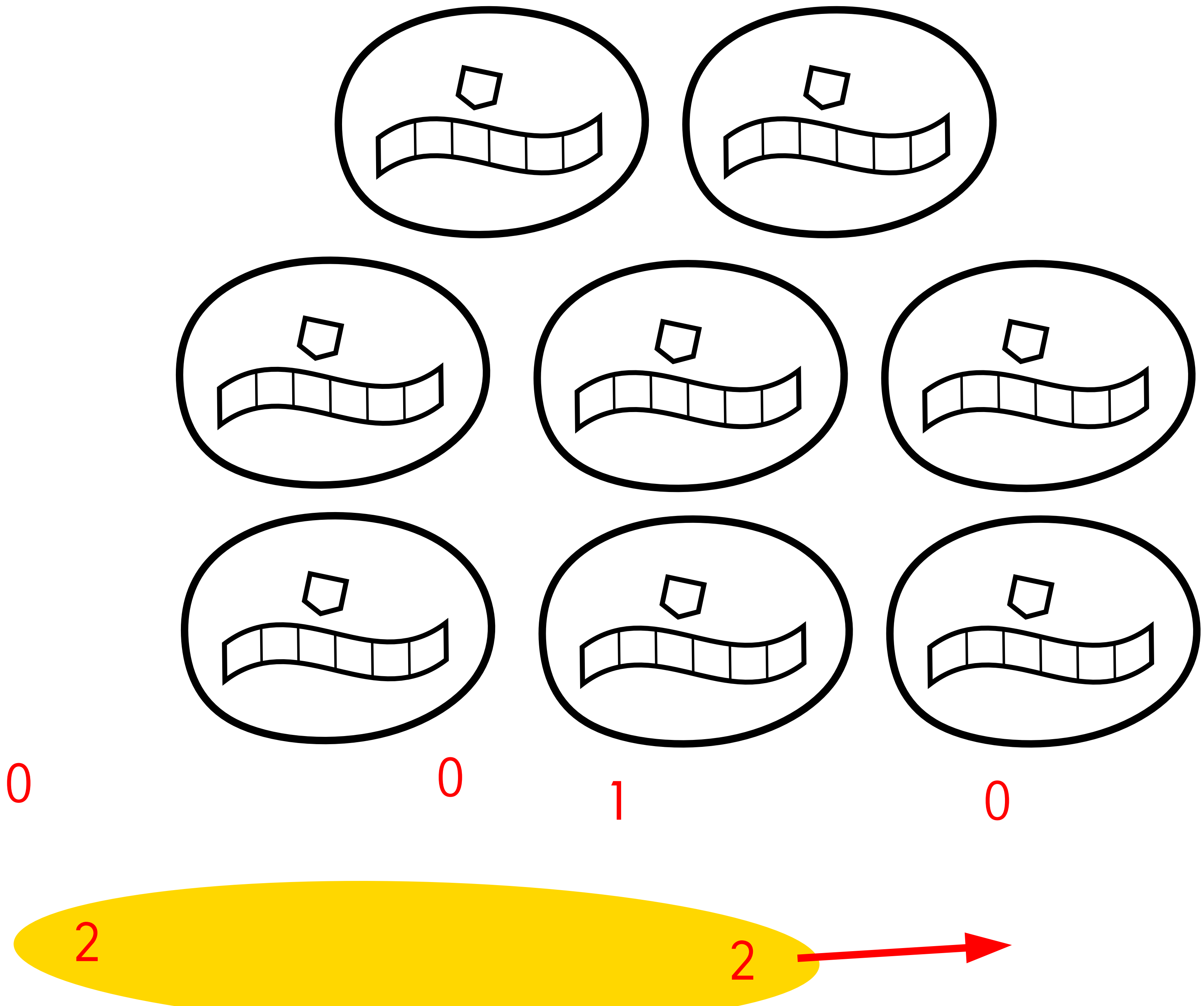
Simulating counting Turing machines



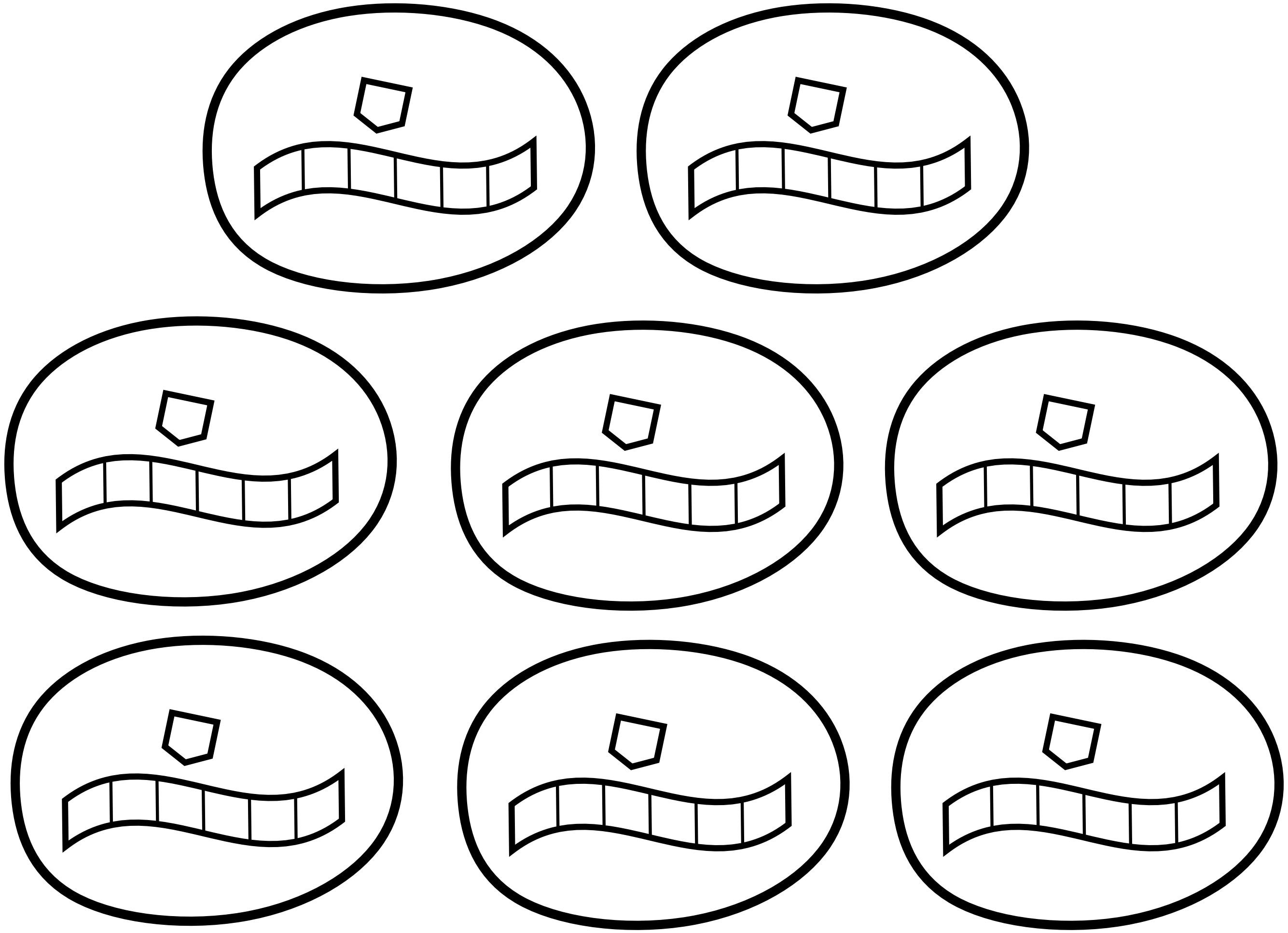
Simulating counting Turing machines



Simulating counting Turing machines



Simulating counting Turing machines



0

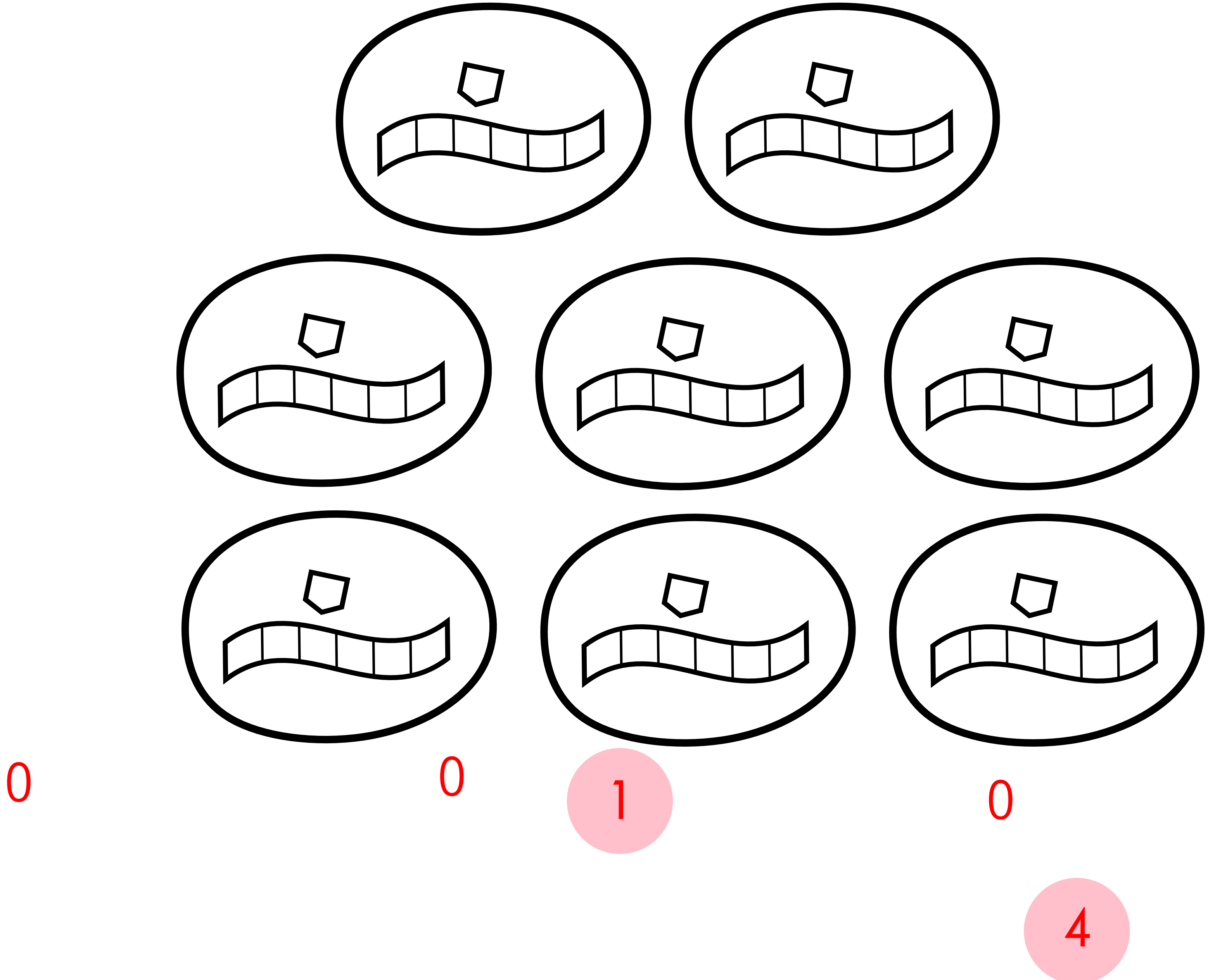
0

1

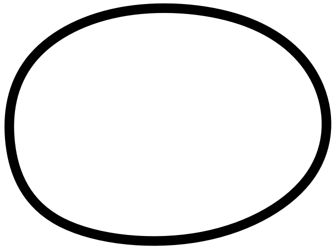
0

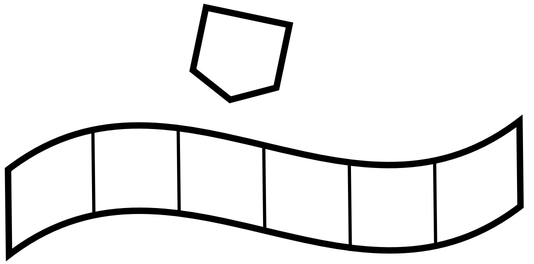
4

Simulating counting Turing machines

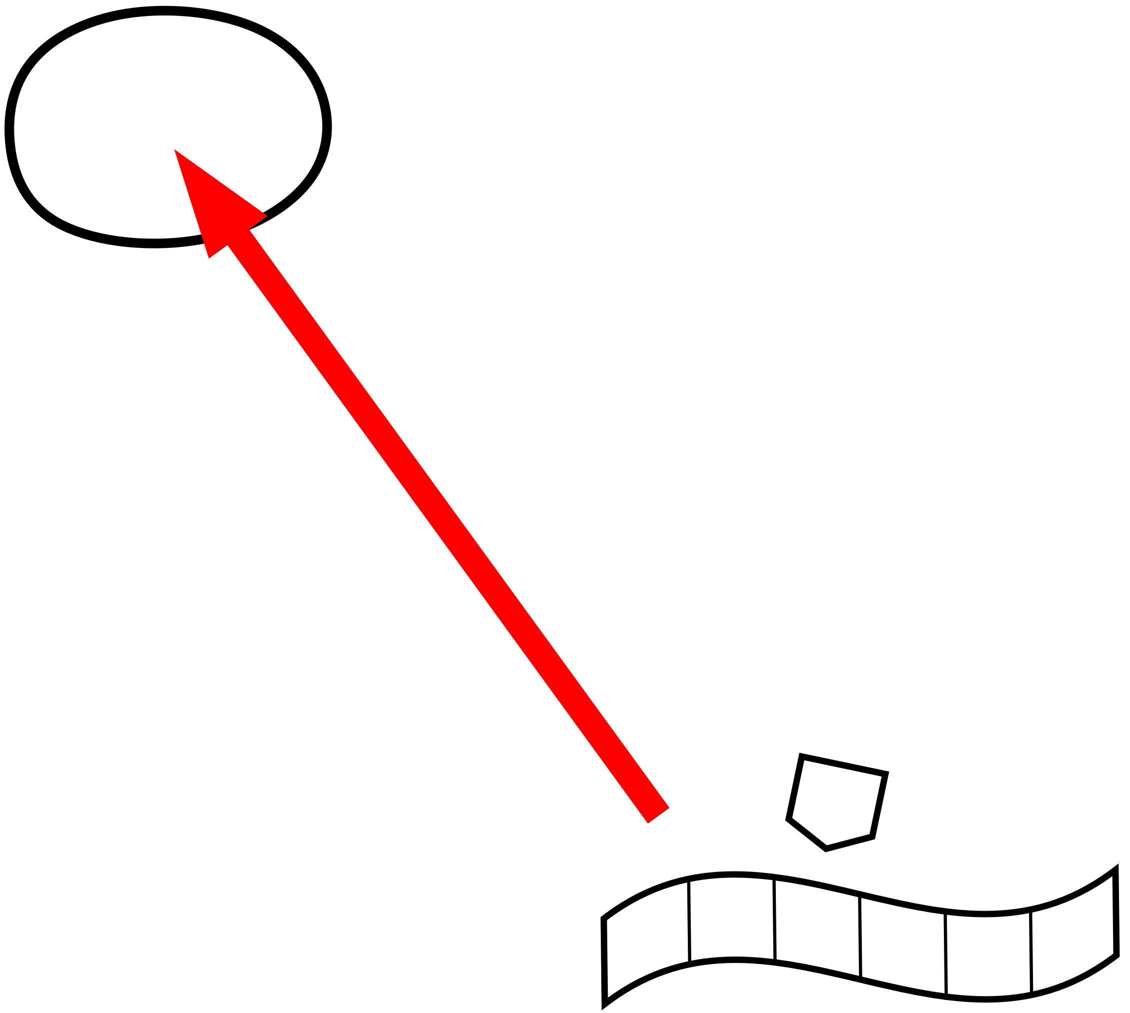


Simulating Turing machines with #P oracles

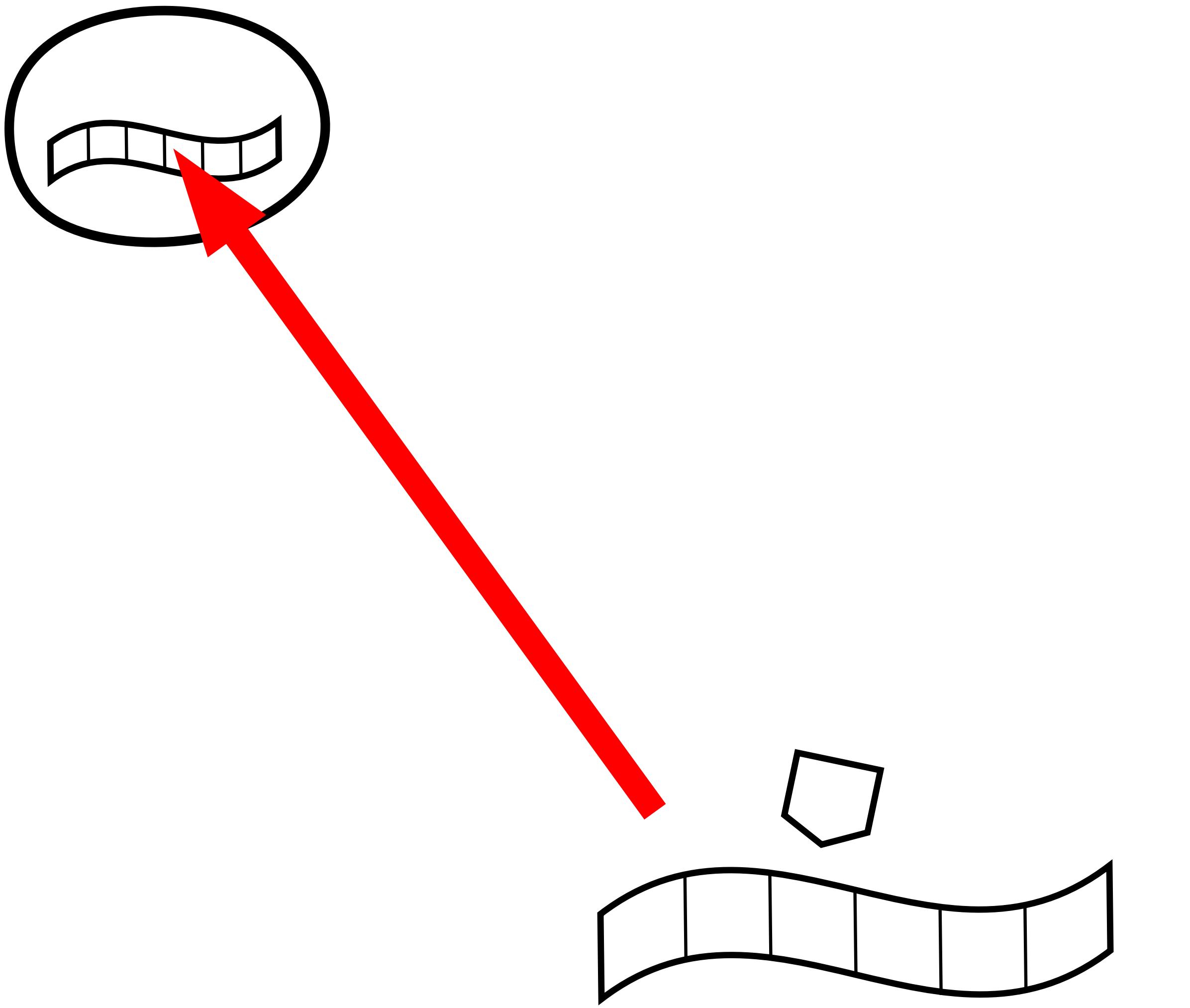




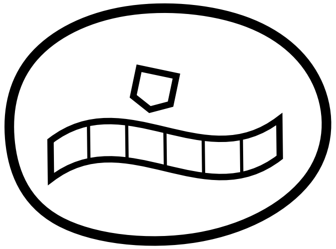
Simulating Turing machines with #P oracles

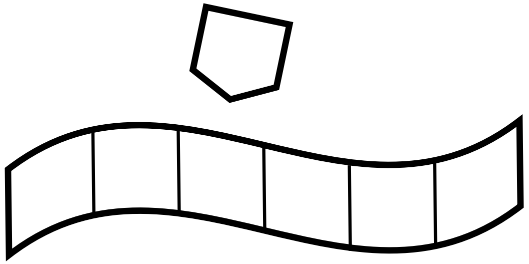


Simulating Turing machines with #P oracles

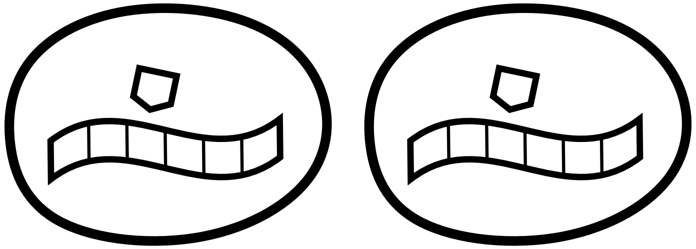


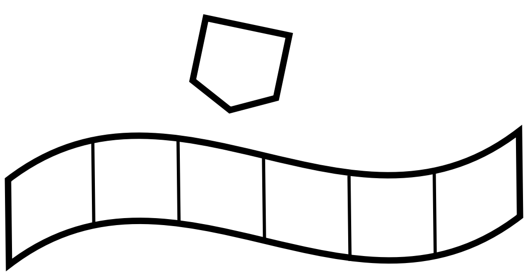
Simulating Turing machines with #P oracles



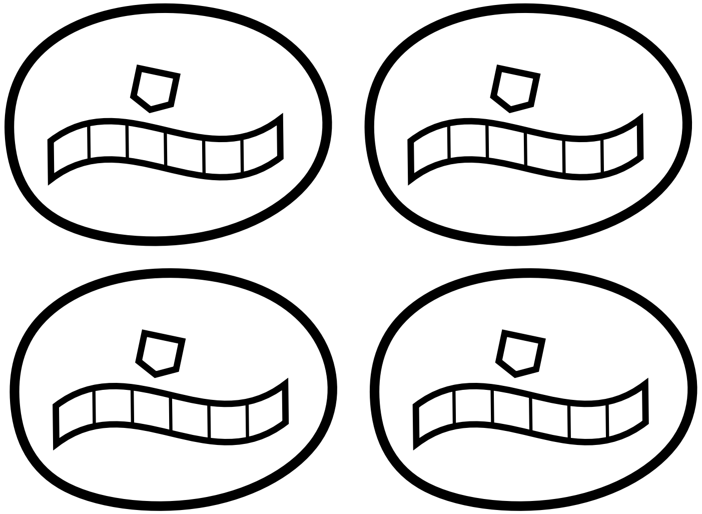


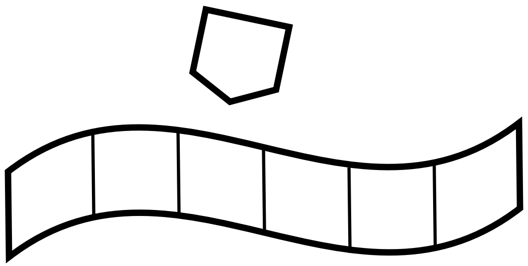
Simulating Turing machines with #P oracles



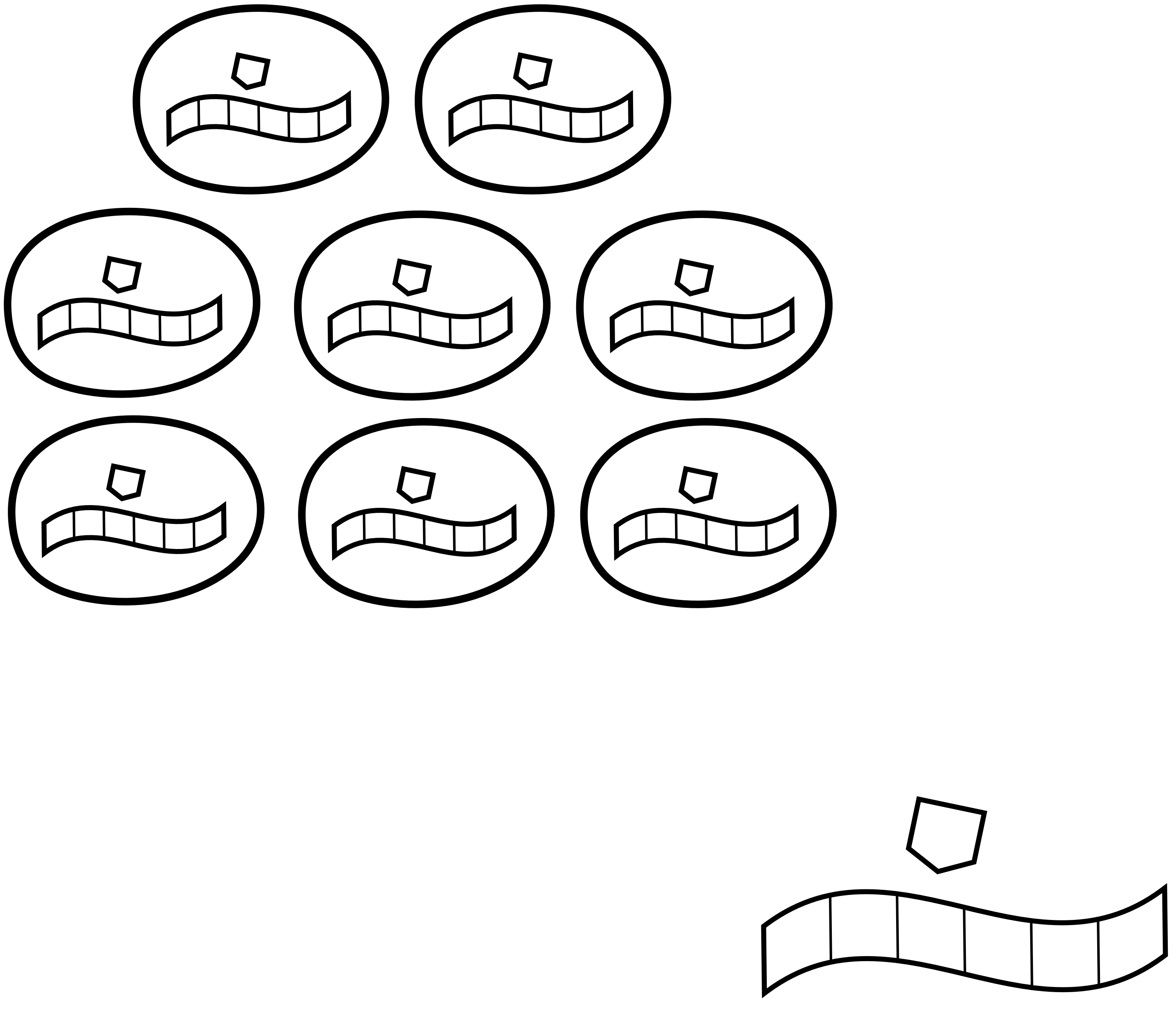


Simulating Turing machines with #P oracles

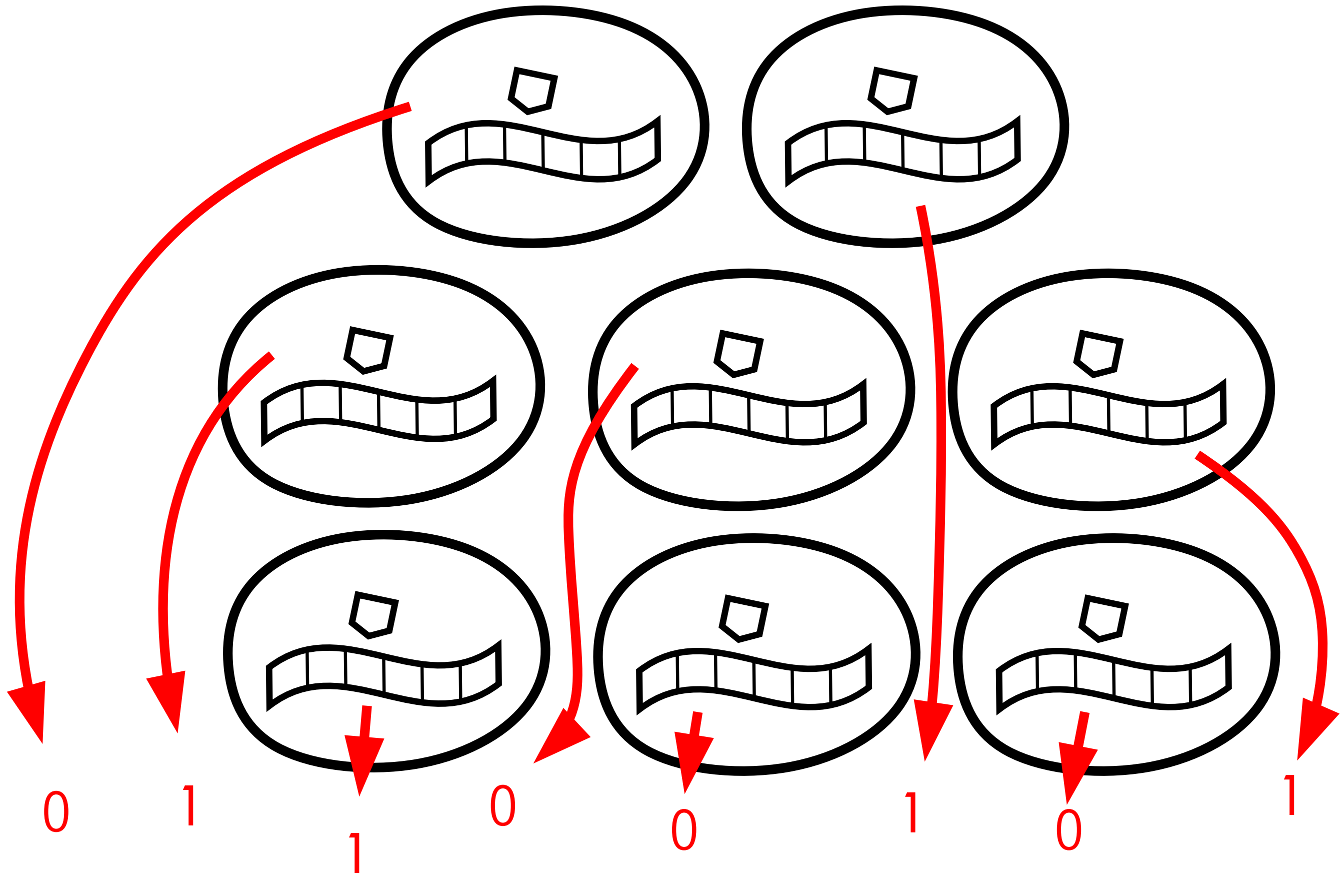


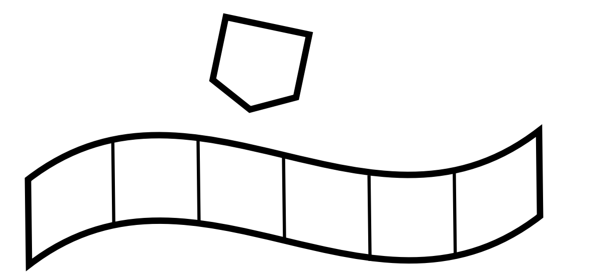


Simulating Turing machines with #P oracles

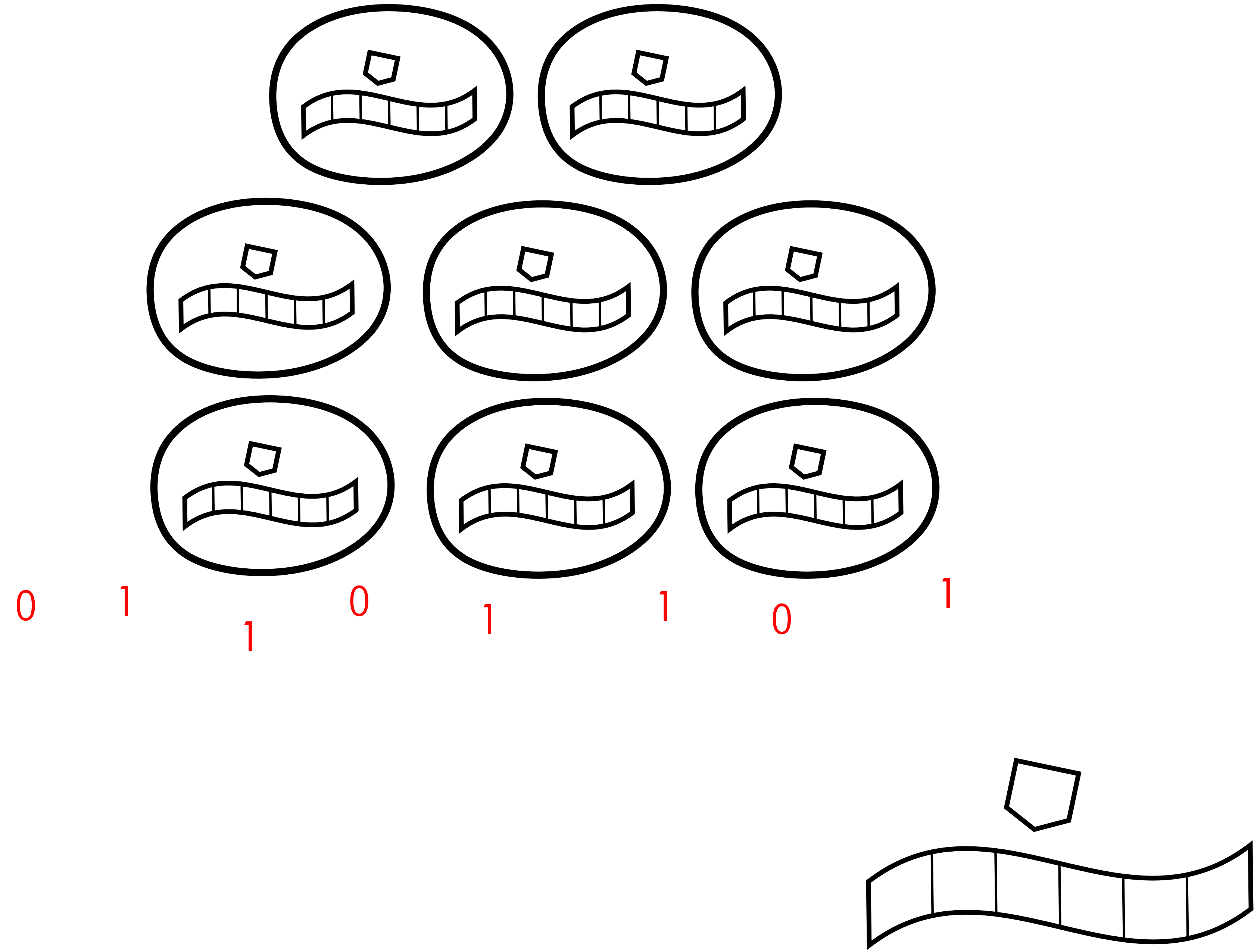


Simulating Turing machines with #P oracles

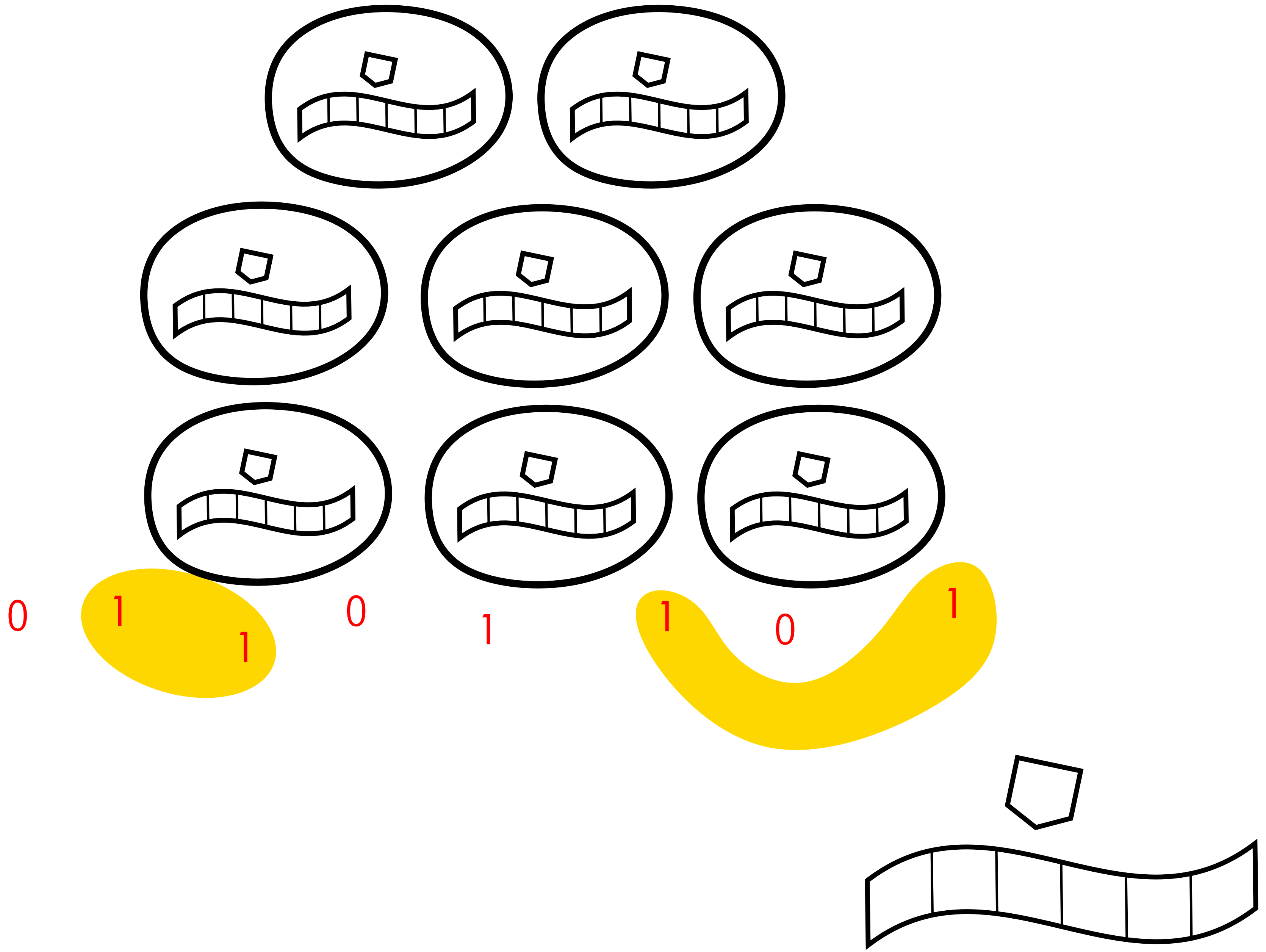




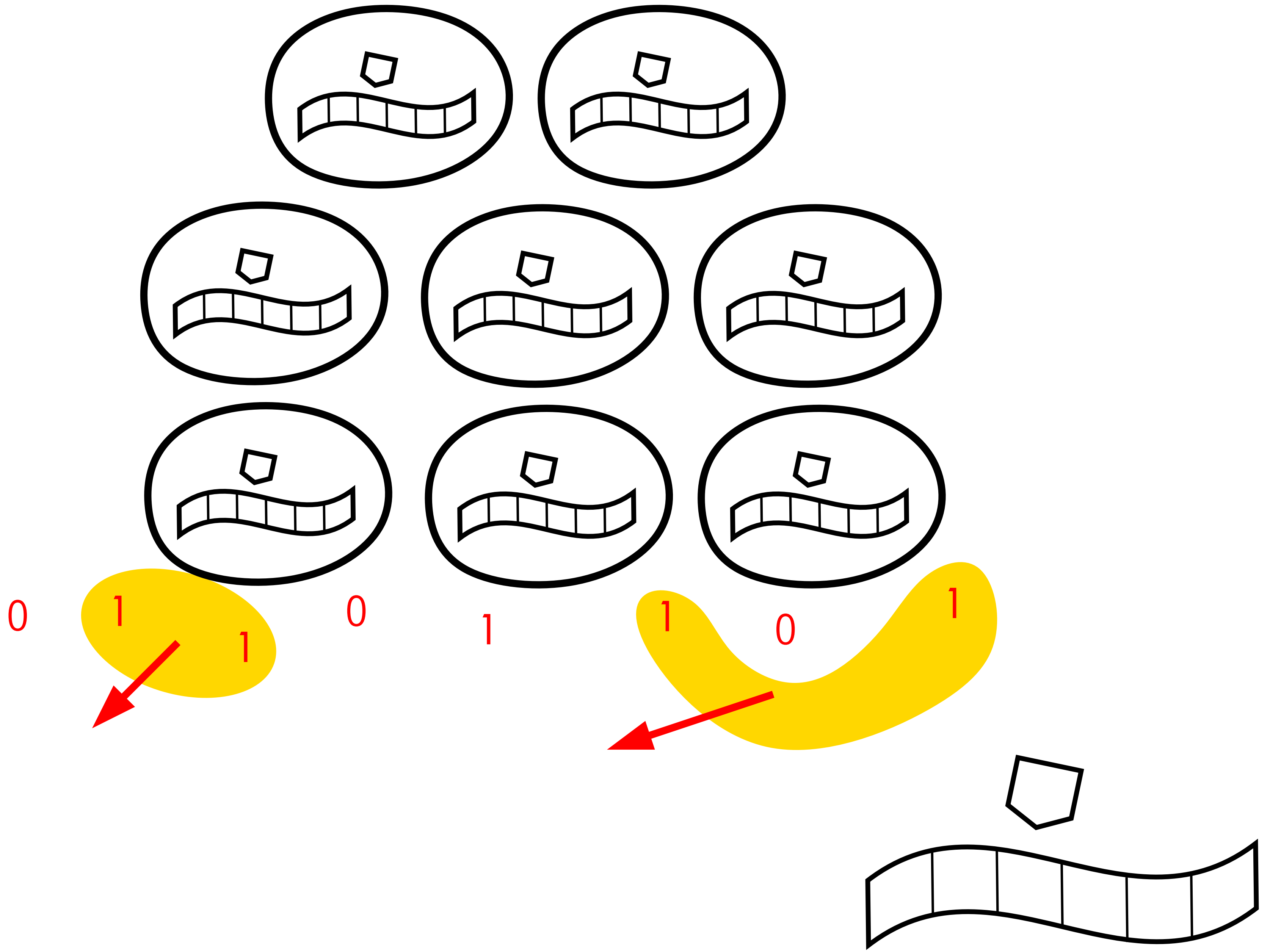
Simulating Turing machines with #P oracles



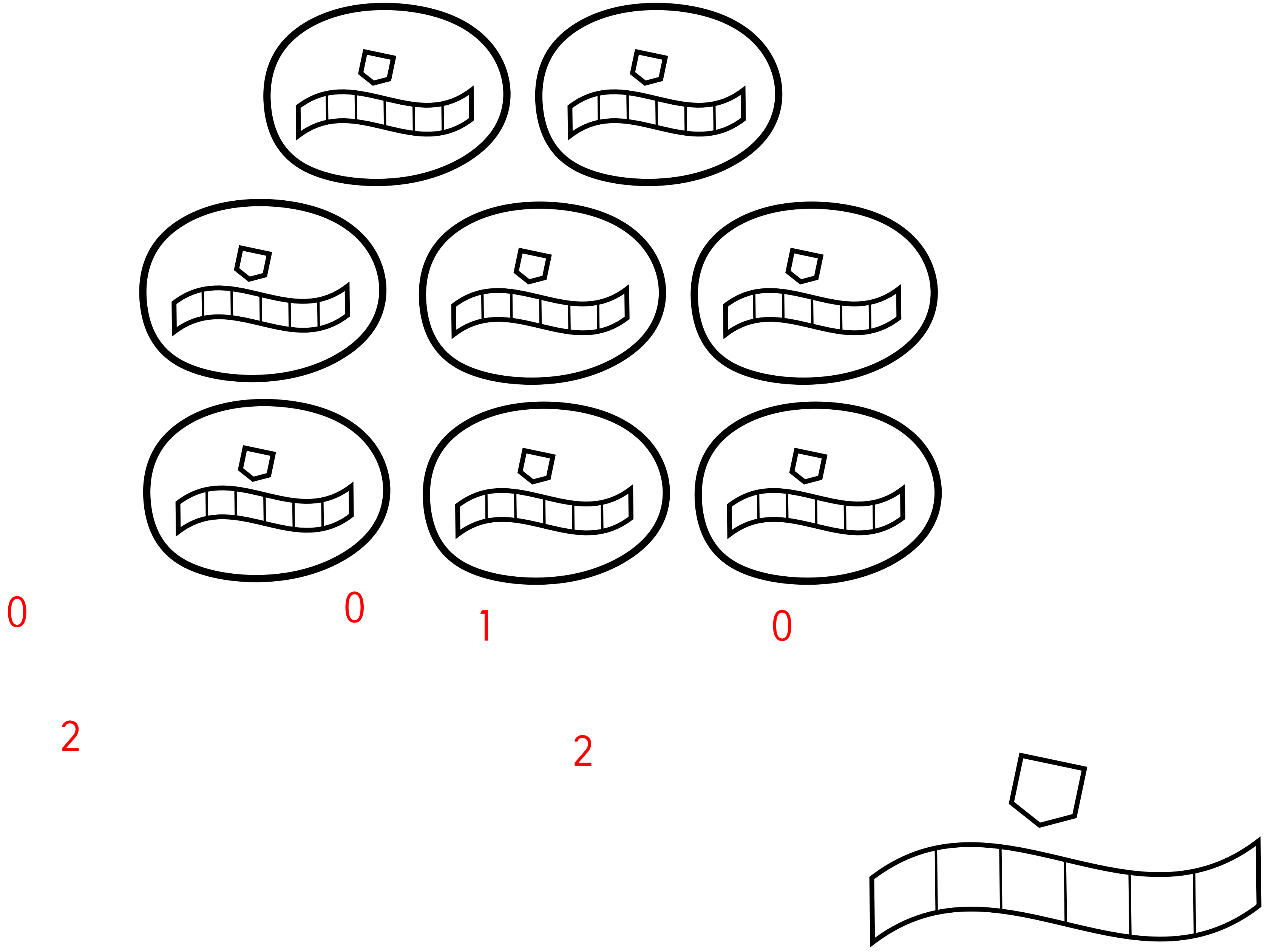
Simulating Turing machines with #P oracles



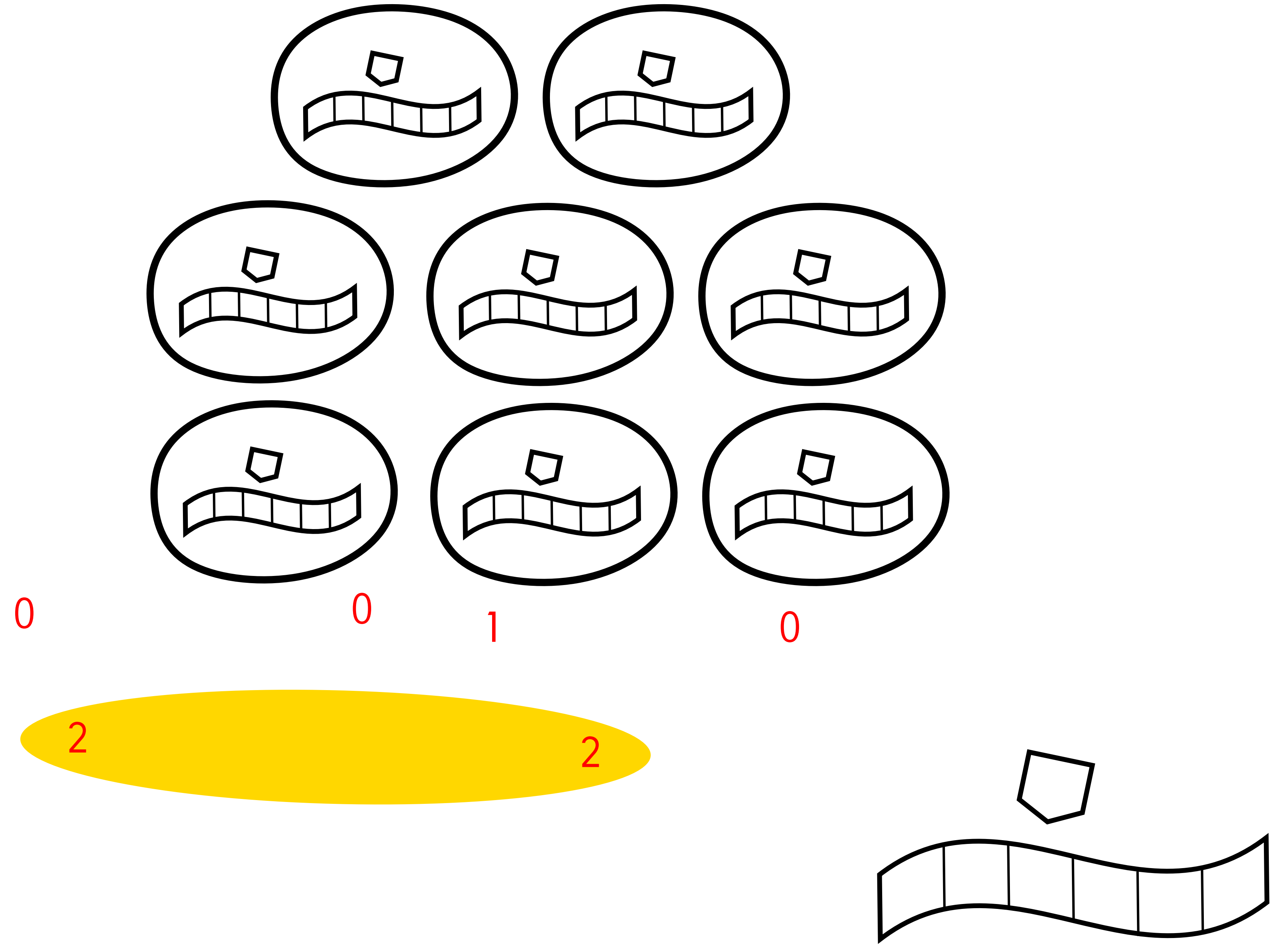
Simulating Turing machines with #P oracles



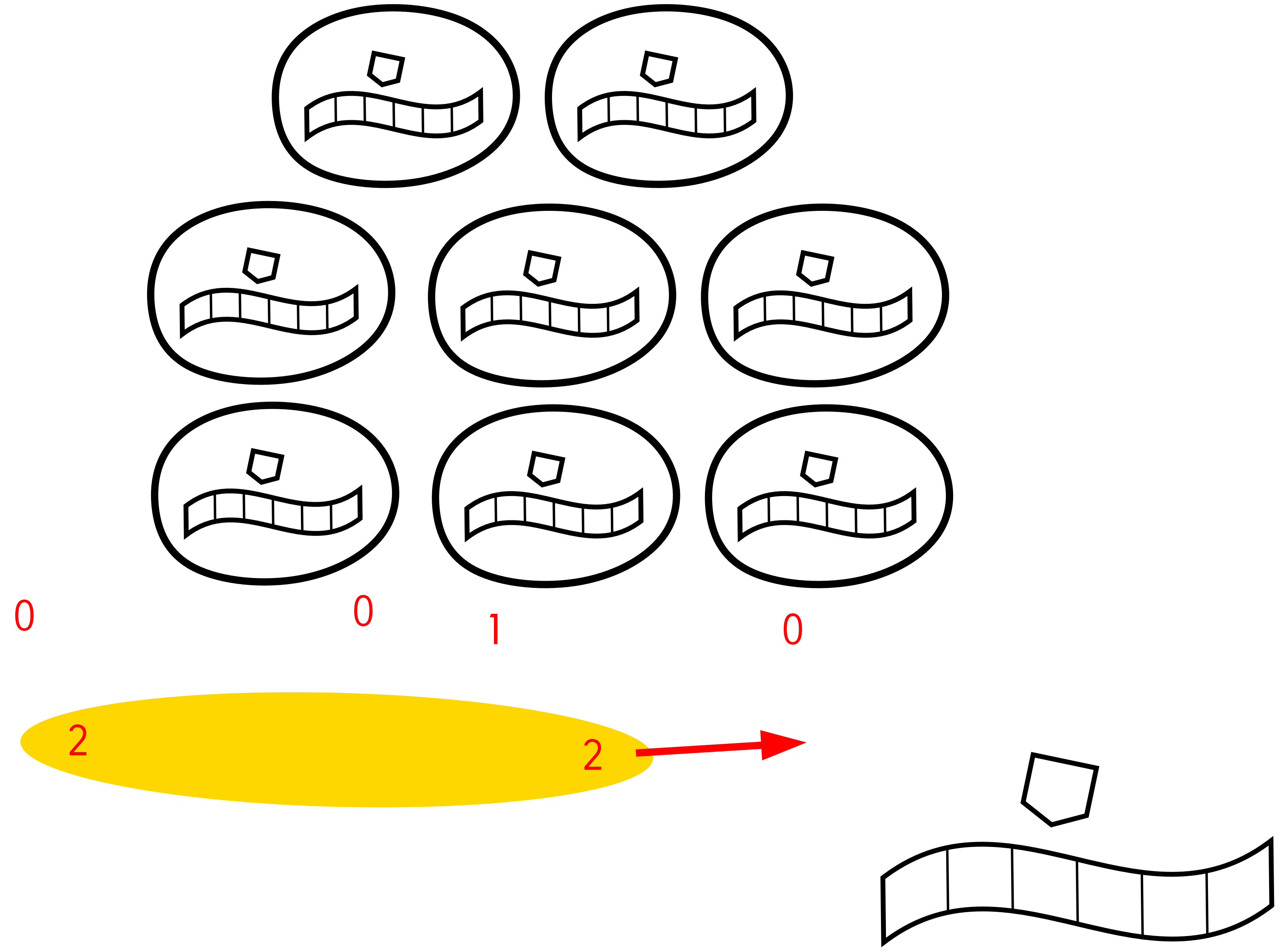
Simulating Turing machines with #P oracles



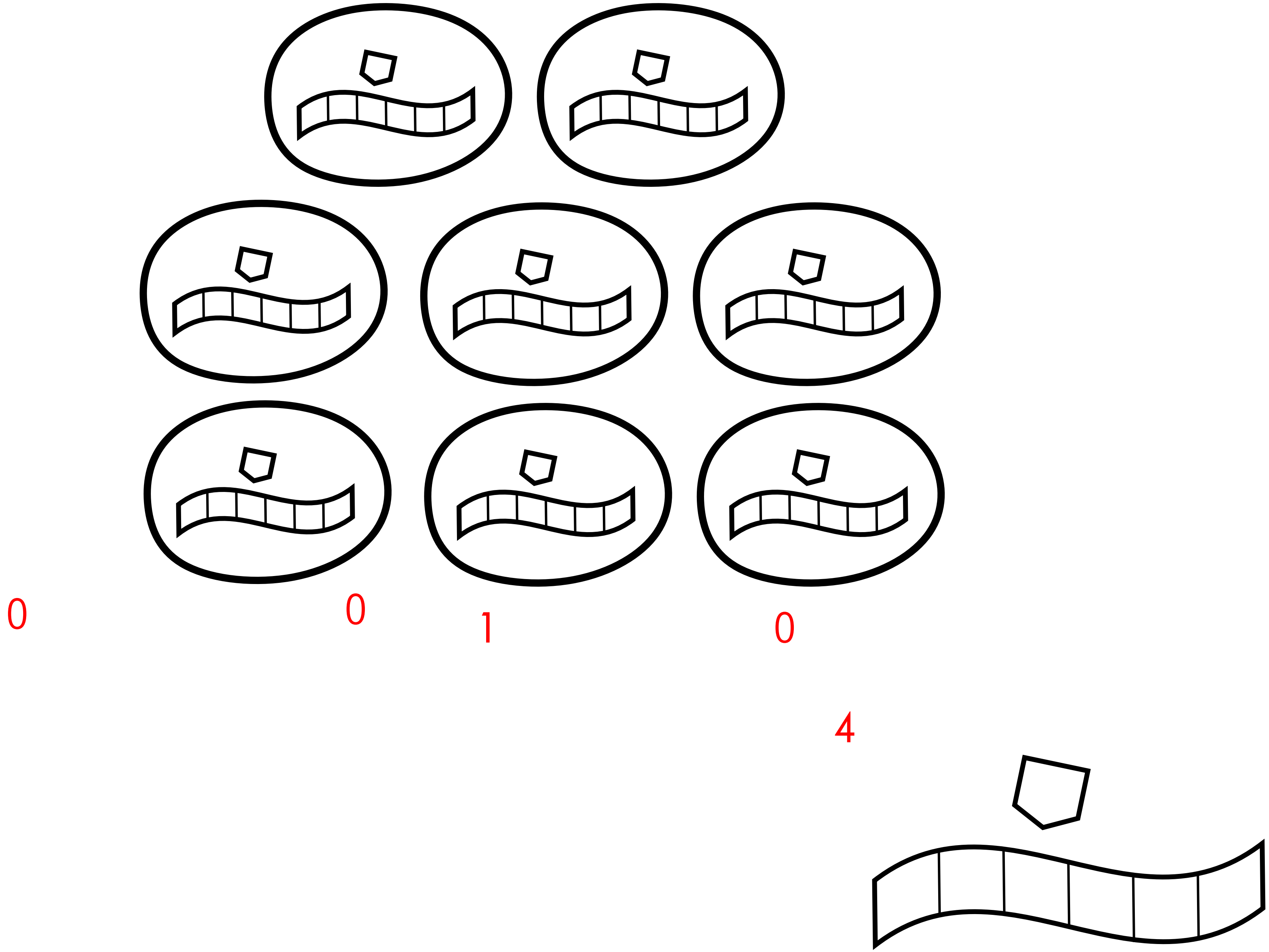
Simulating Turing machines with #P oracles



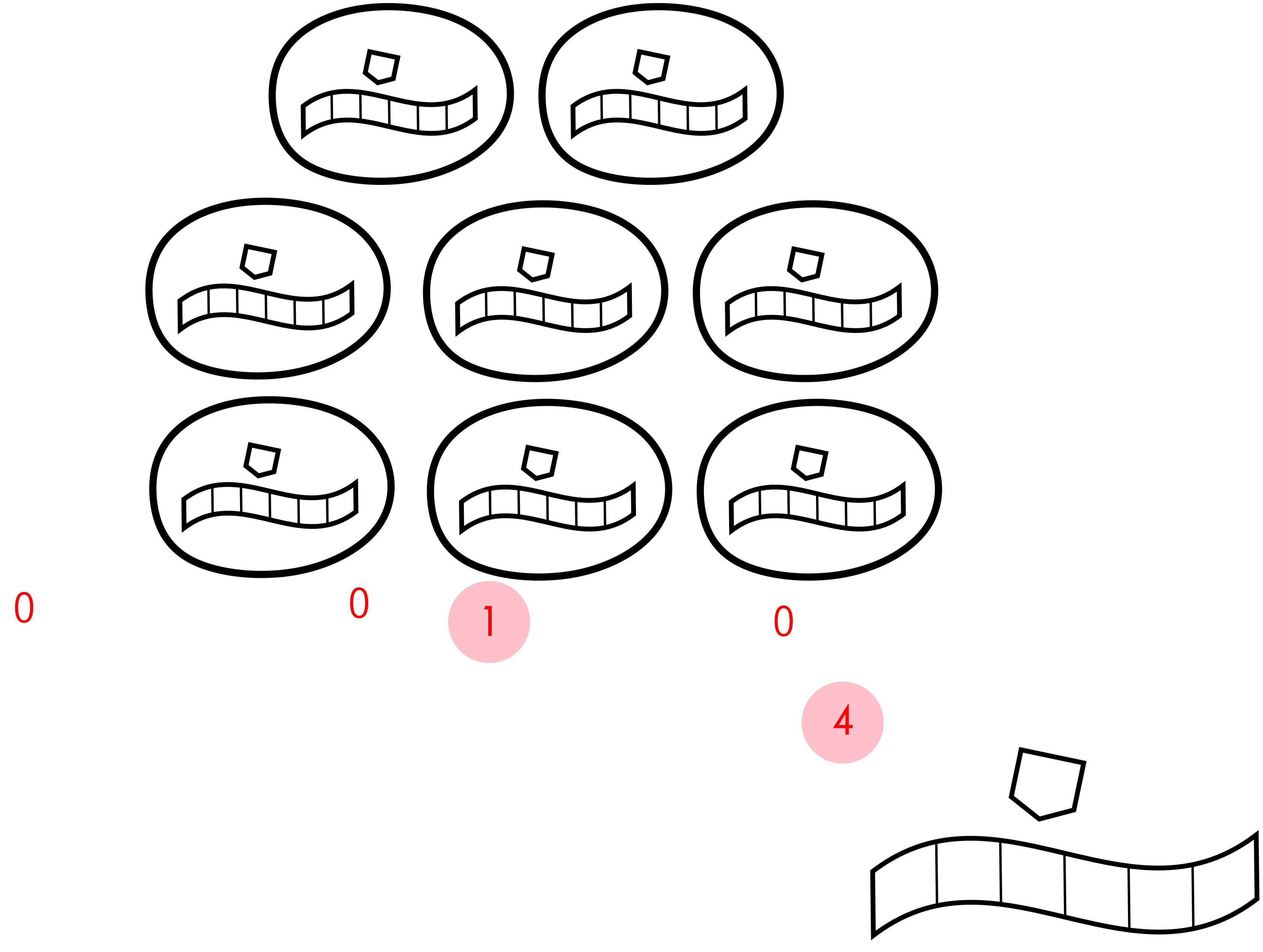
Simulating Turing machines with #P oracles



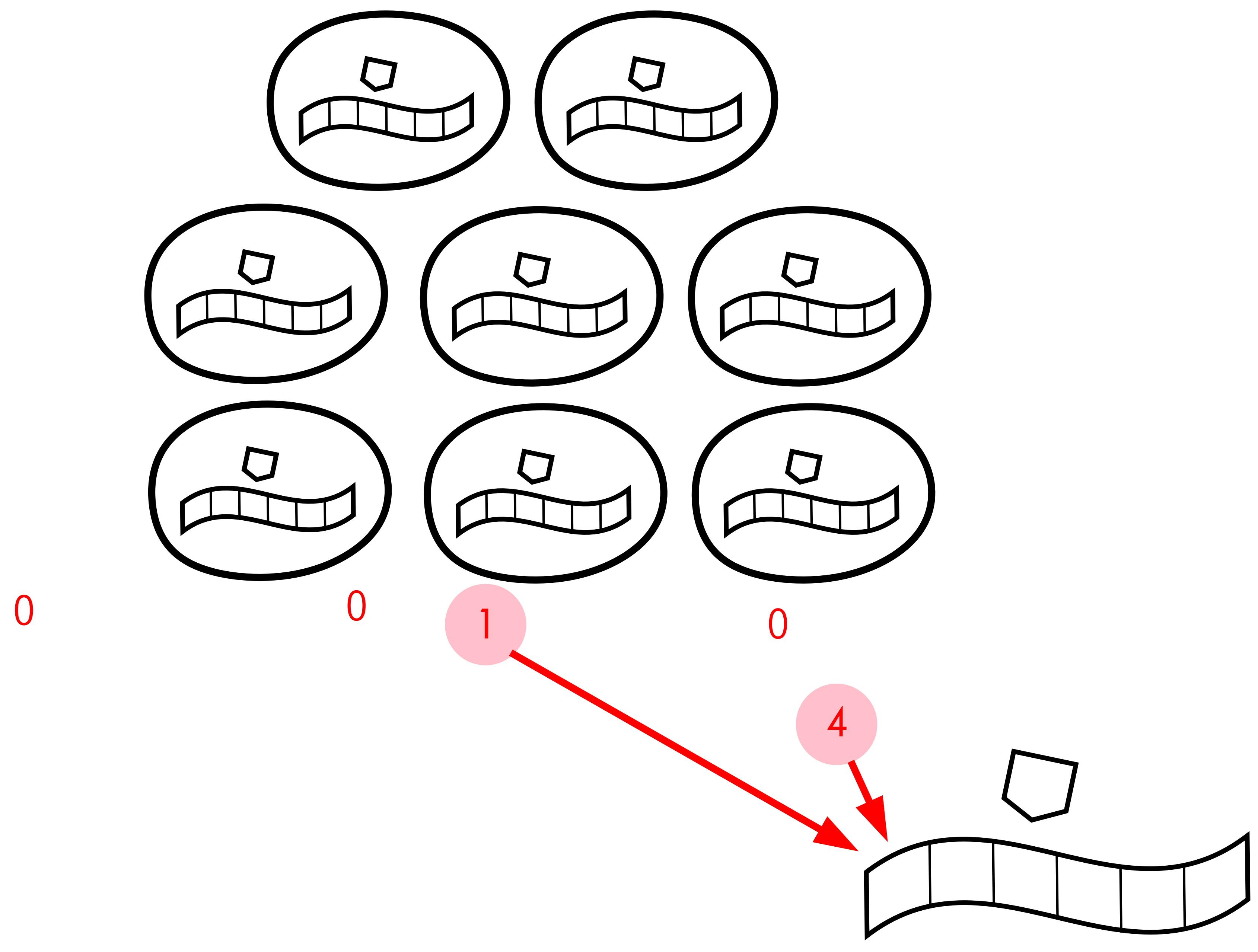
Simulating Turing machines with #P oracles



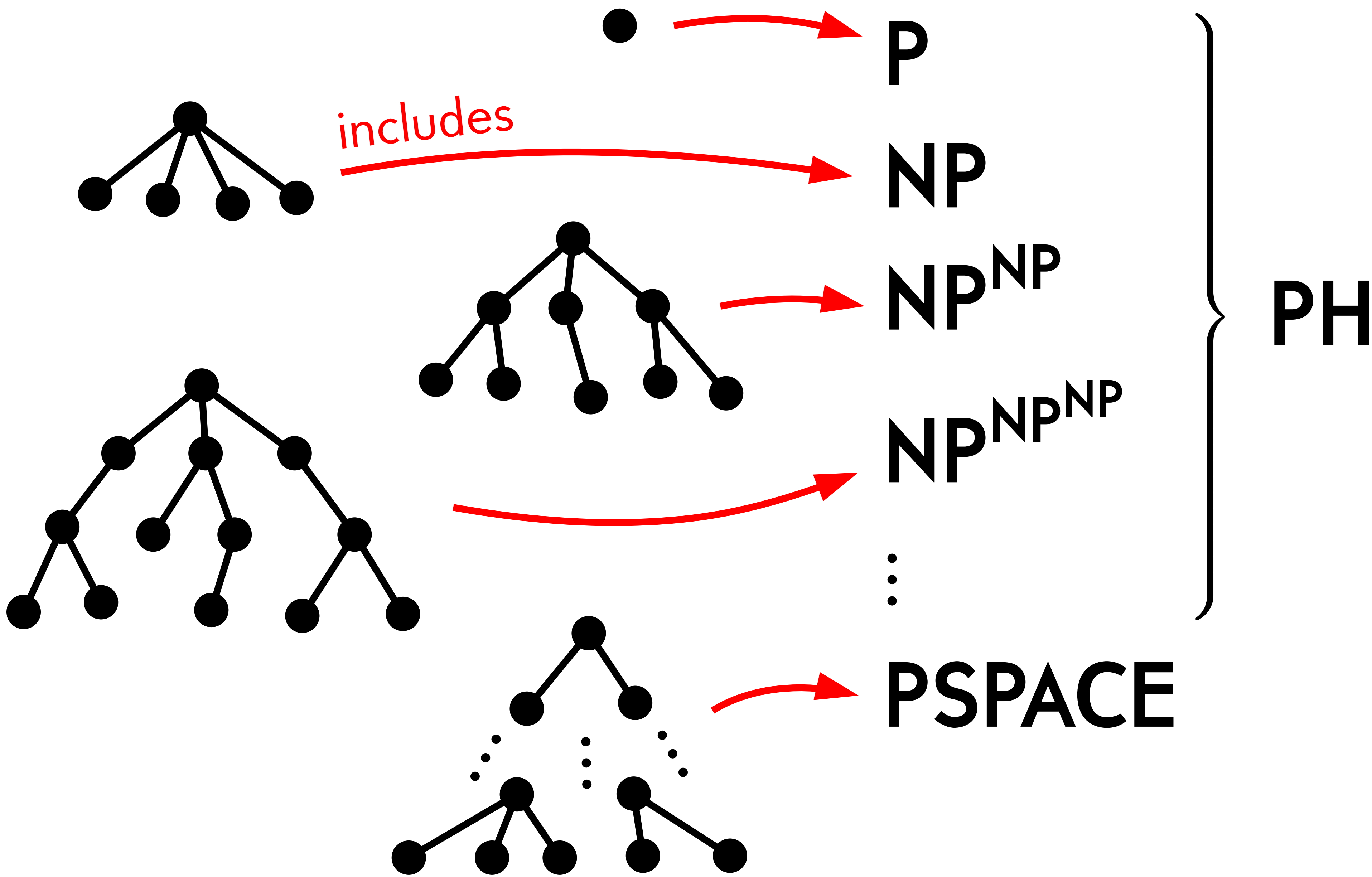
Simulating Turing machines with #P oracles



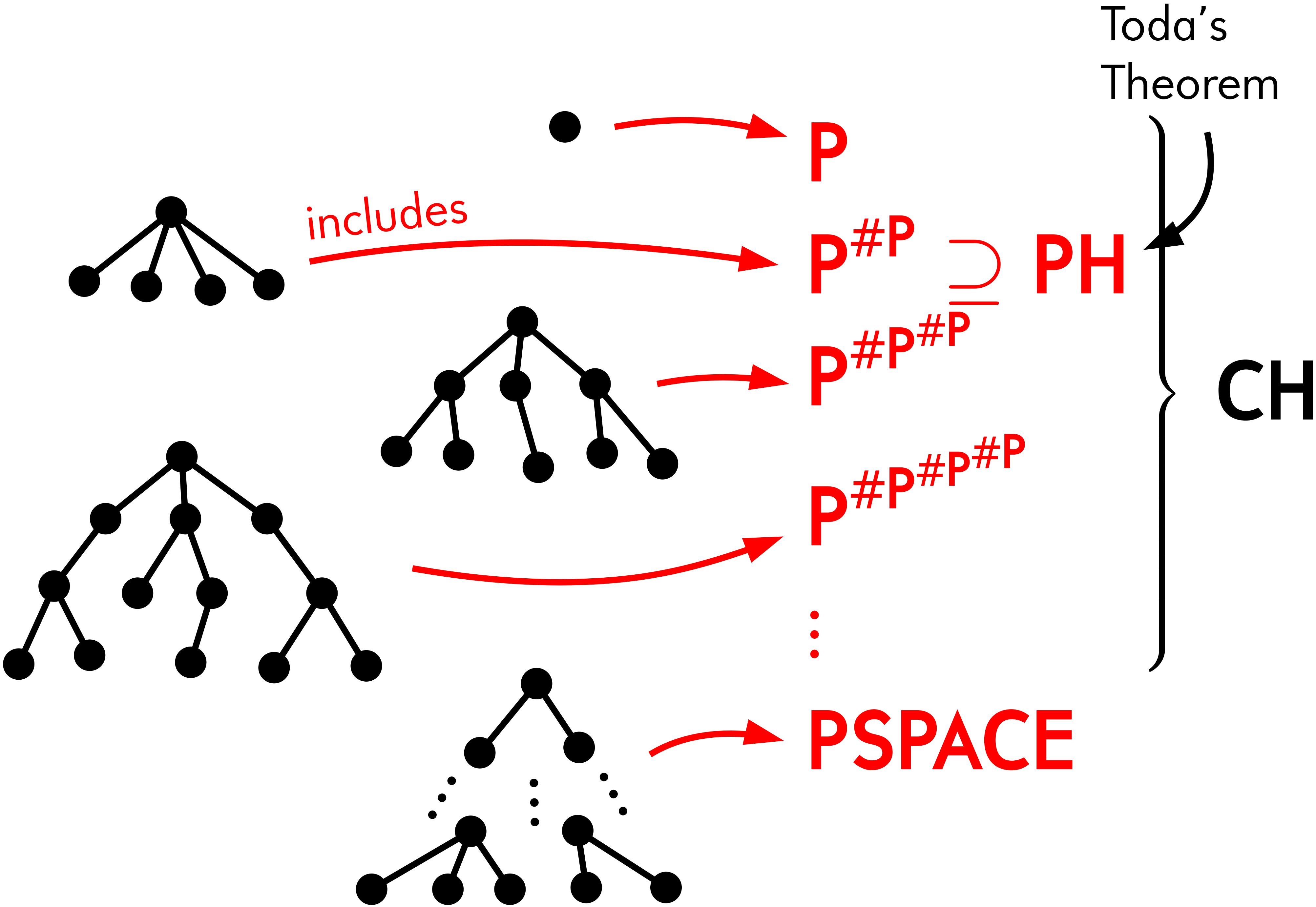
Simulating Turing machines with #P oracles



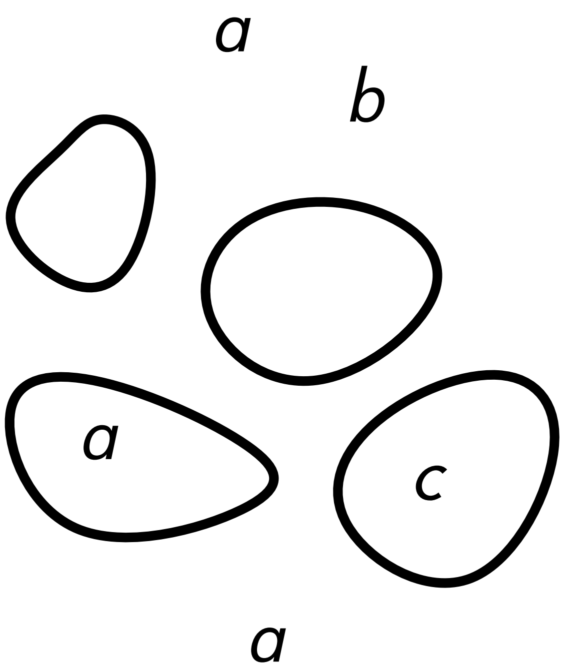
The counting hierarchy



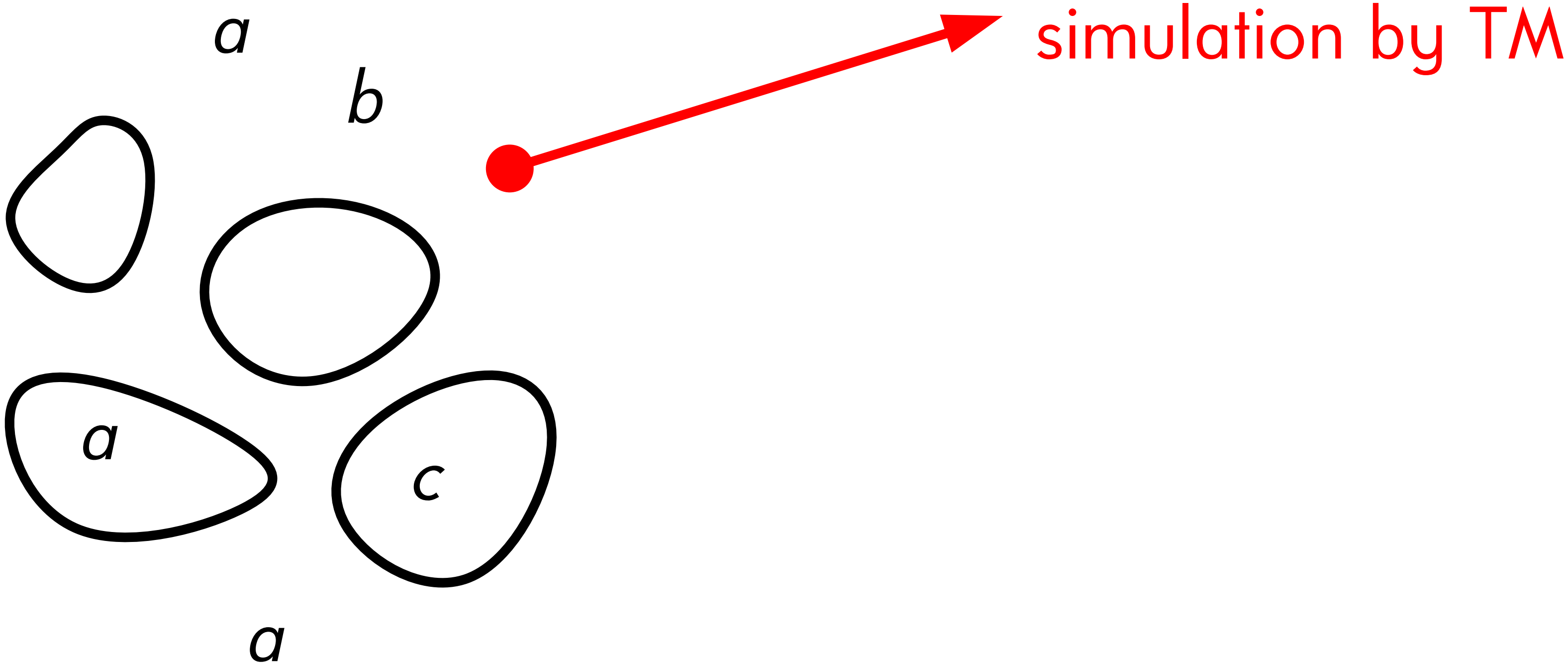
The counting hierarchy



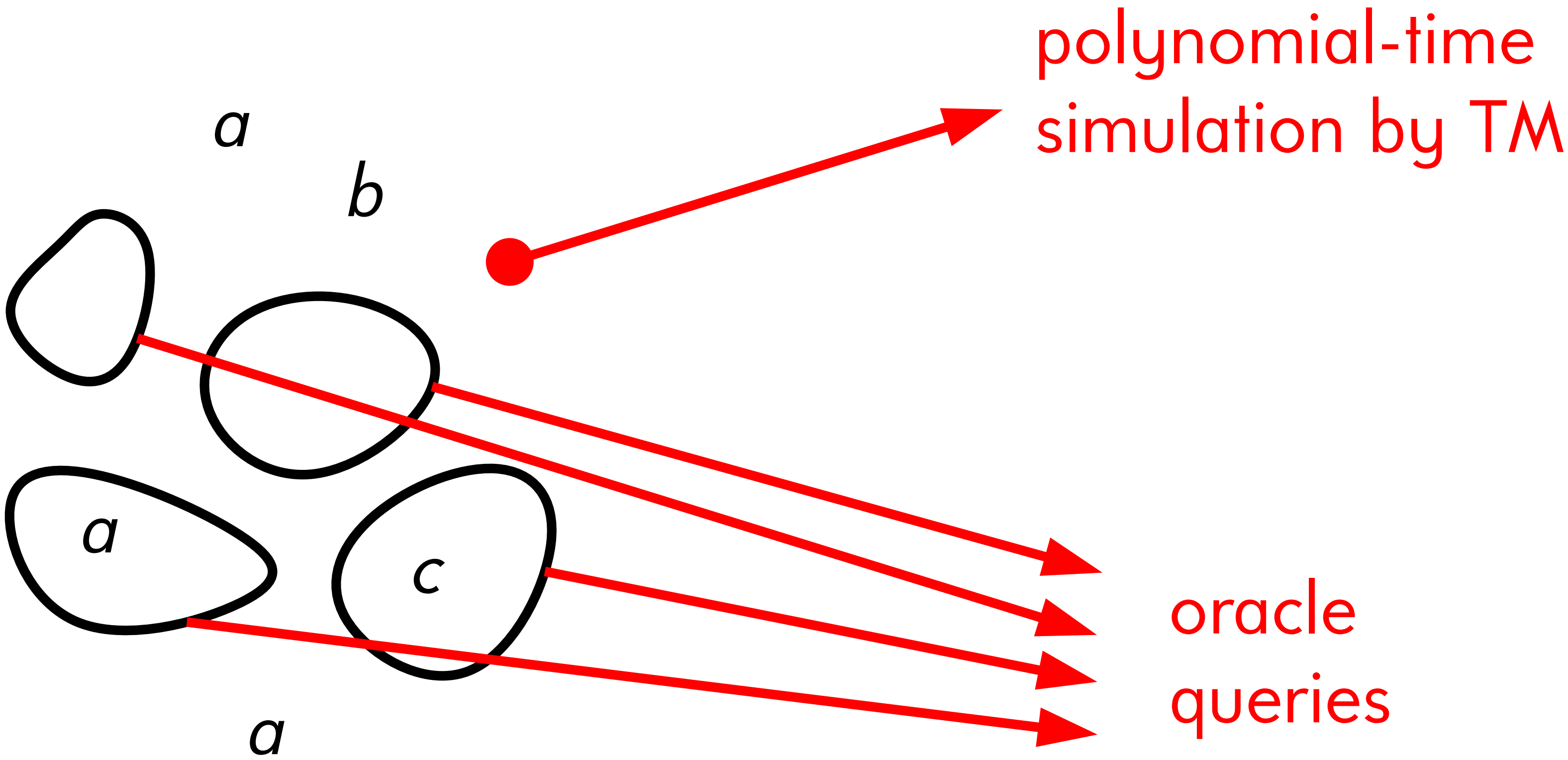
Exact characterisation of $\mathbf{P}^{\#\mathbf{P}}$



Exact characterisation of $\mathbf{P}^{\#\mathbf{P}}$



Exact characterisation of $\mathbf{P}^{\#\mathbf{P}}$

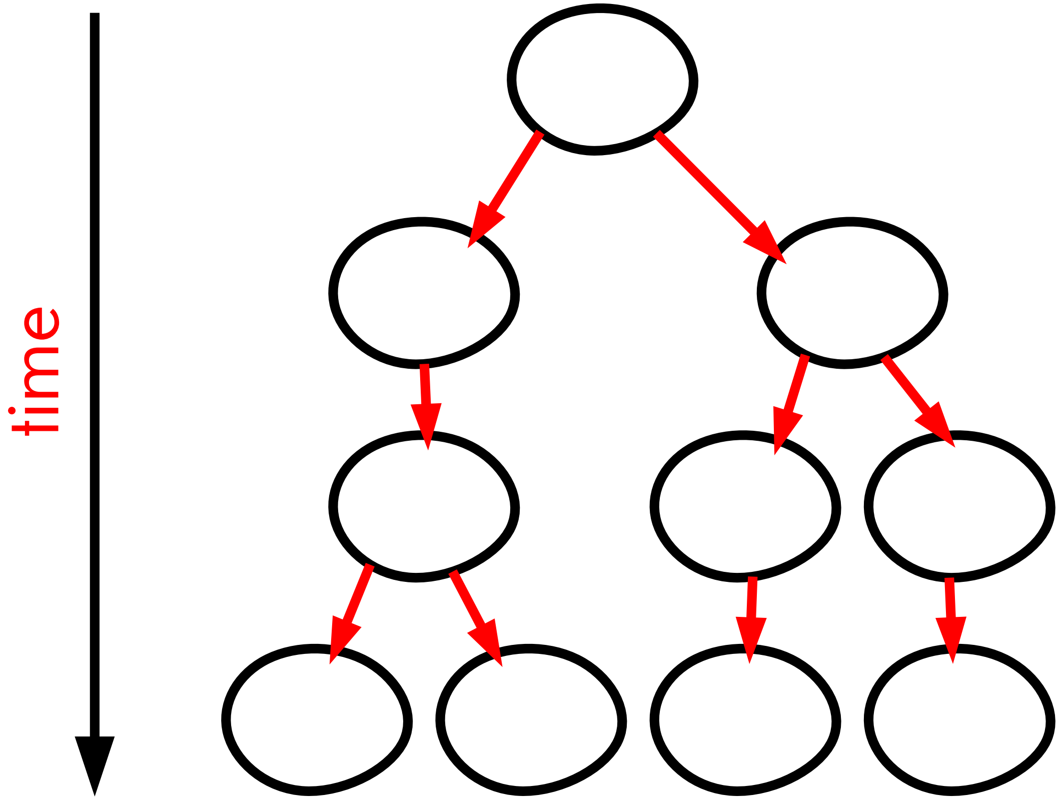


Query simulating dividing membranes

If the dividing membranes receive the sequence of inputs $(m_1, \dots, m_t)$, how many instances of object a are sent out at time $t + 1$?

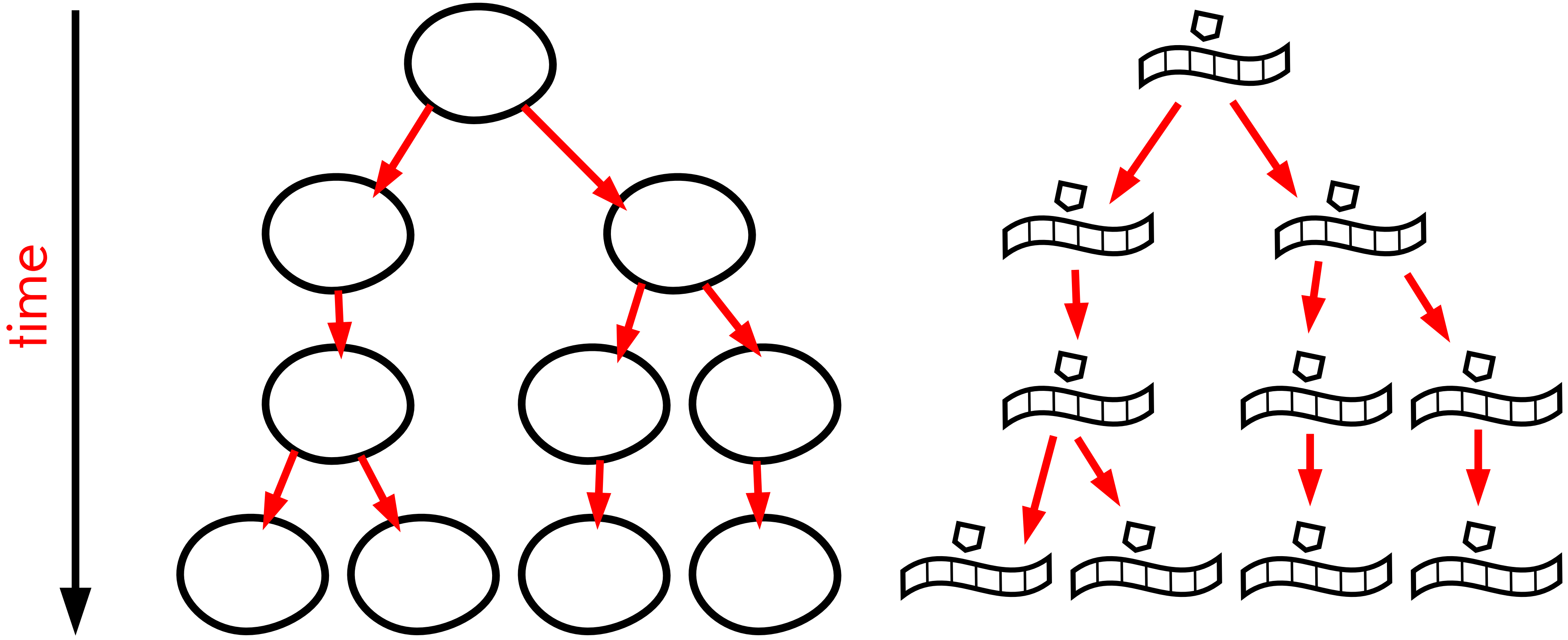
Query simulating dividing membranes

If the dividing membranes receive the sequence of inputs $(m_1, \ldots, m_t)$, how many instances of object a are sent out at time $t + 1$?



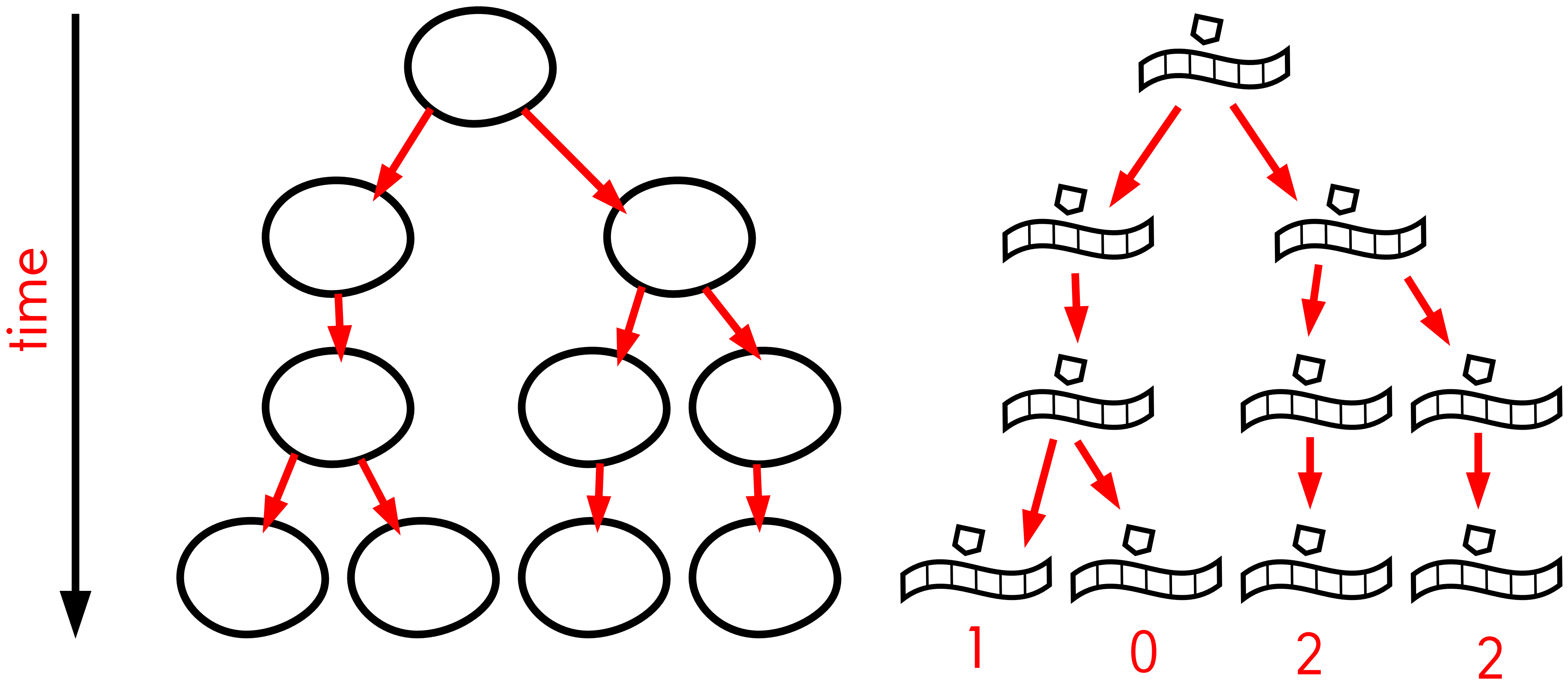
Query simulating dividing membranes

If the dividing membranes receive the sequence of inputs $(m_1, \ldots, m_t)$, how many instances of object a are sent out at time $t + 1$?



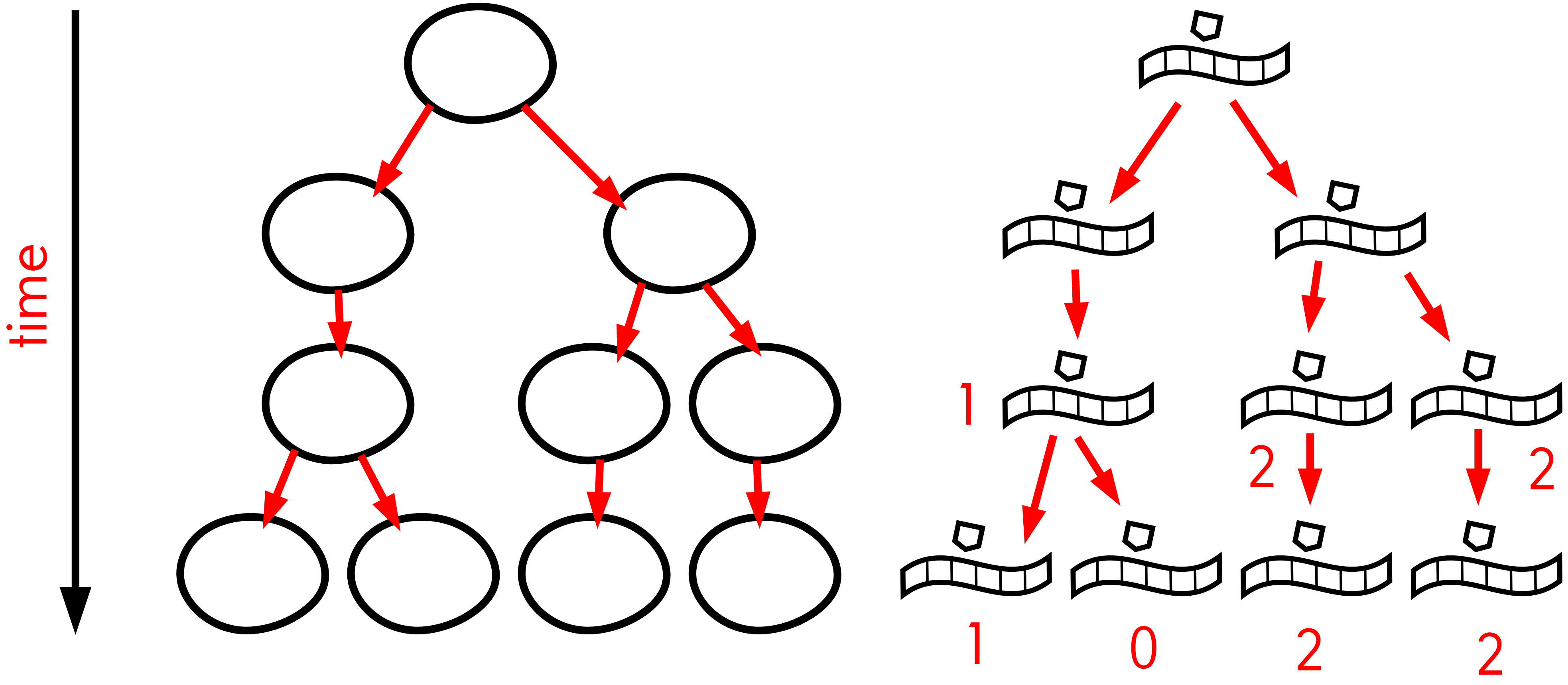
Query simulating dividing membranes

If the dividing membranes receive the sequence of inputs $(m_1, \ldots, m_t)$, how many instances of object a are sent out at time $t + 1$?



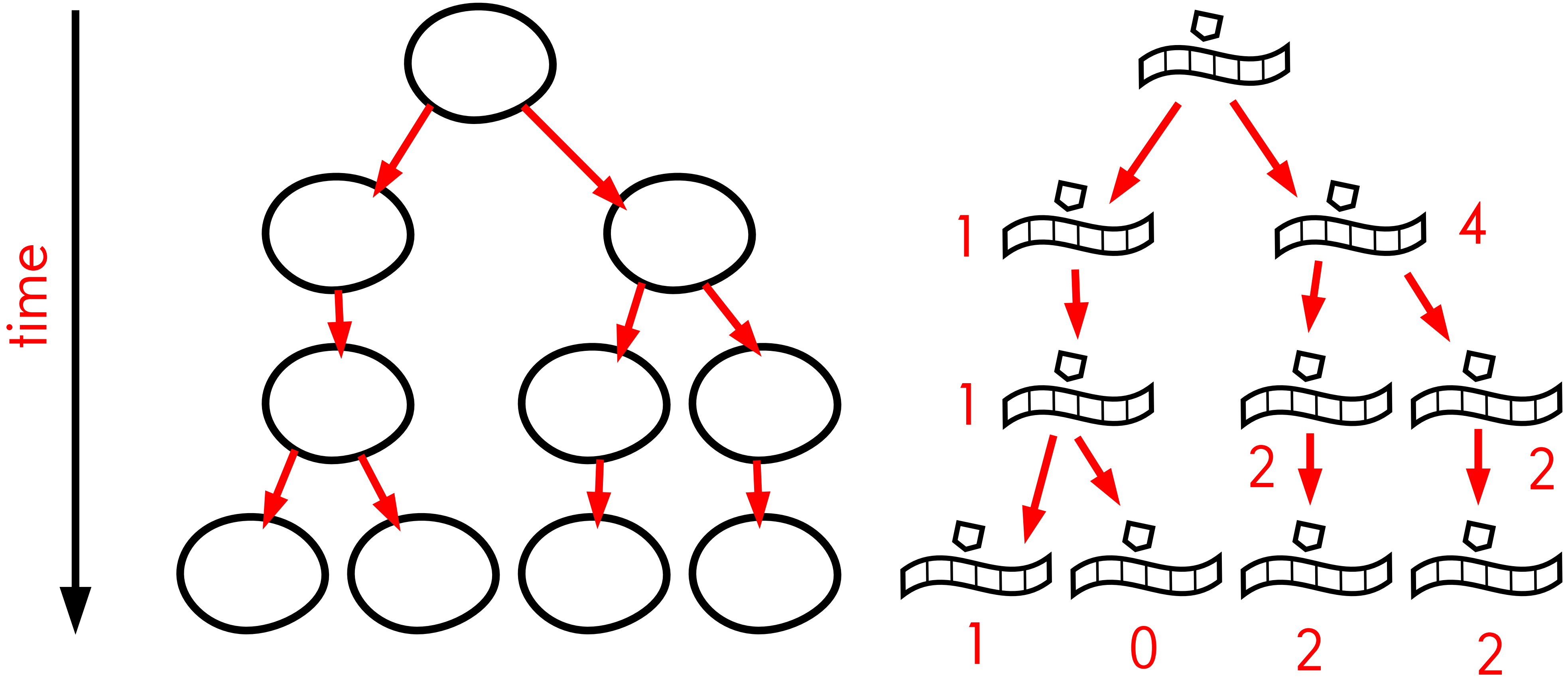
Query simulating dividing membranes

If the dividing membranes receive the sequence of inputs $(m_1, \ldots, m_t)$, how many instances of object a are sent out at time $t + 1$?



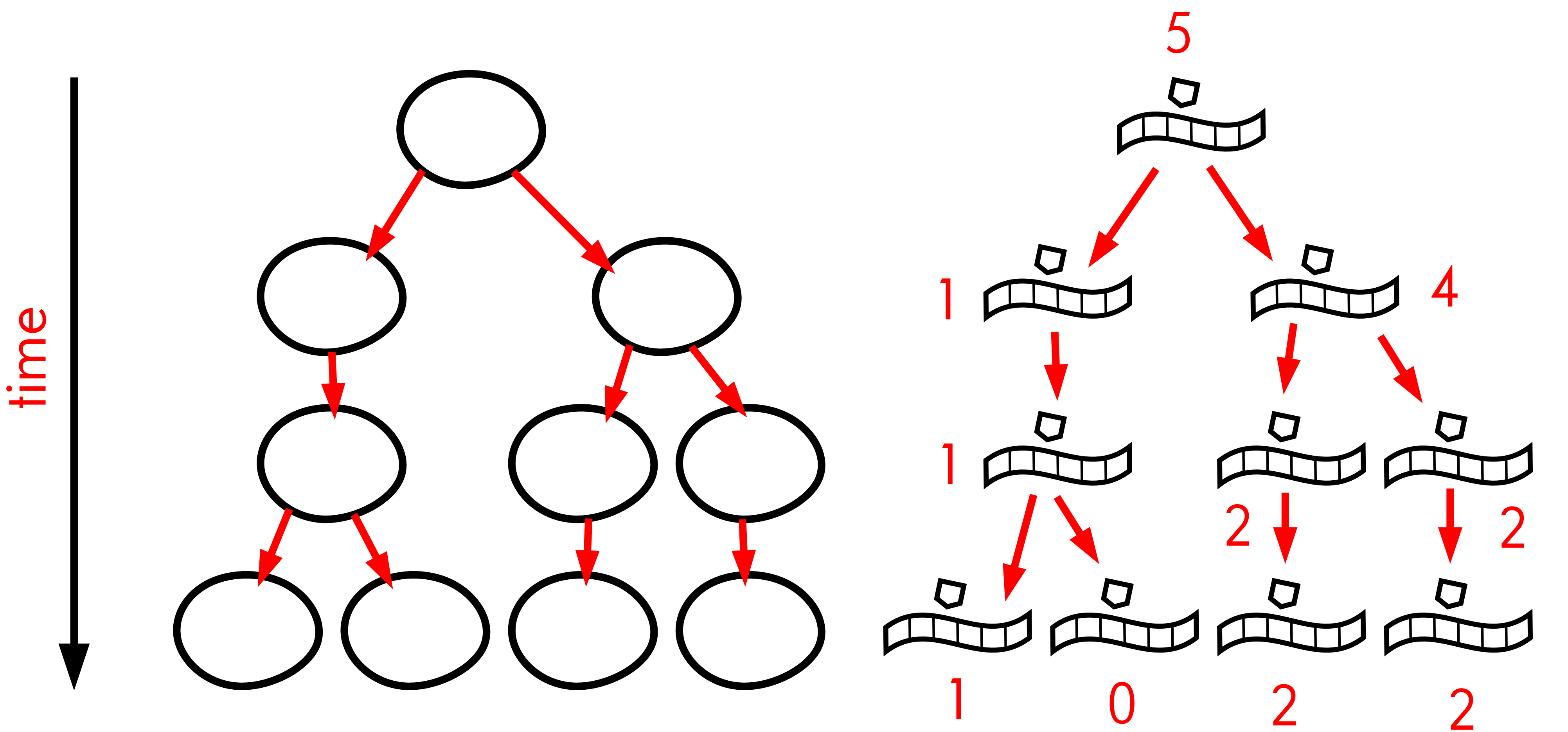
Query simulating dividing membranes

If the dividing membranes receive the sequence of inputs $(m_1, \ldots, m_t)$, how many instances of object a are sent out at time $t+1$?



Query simulating dividing membranes

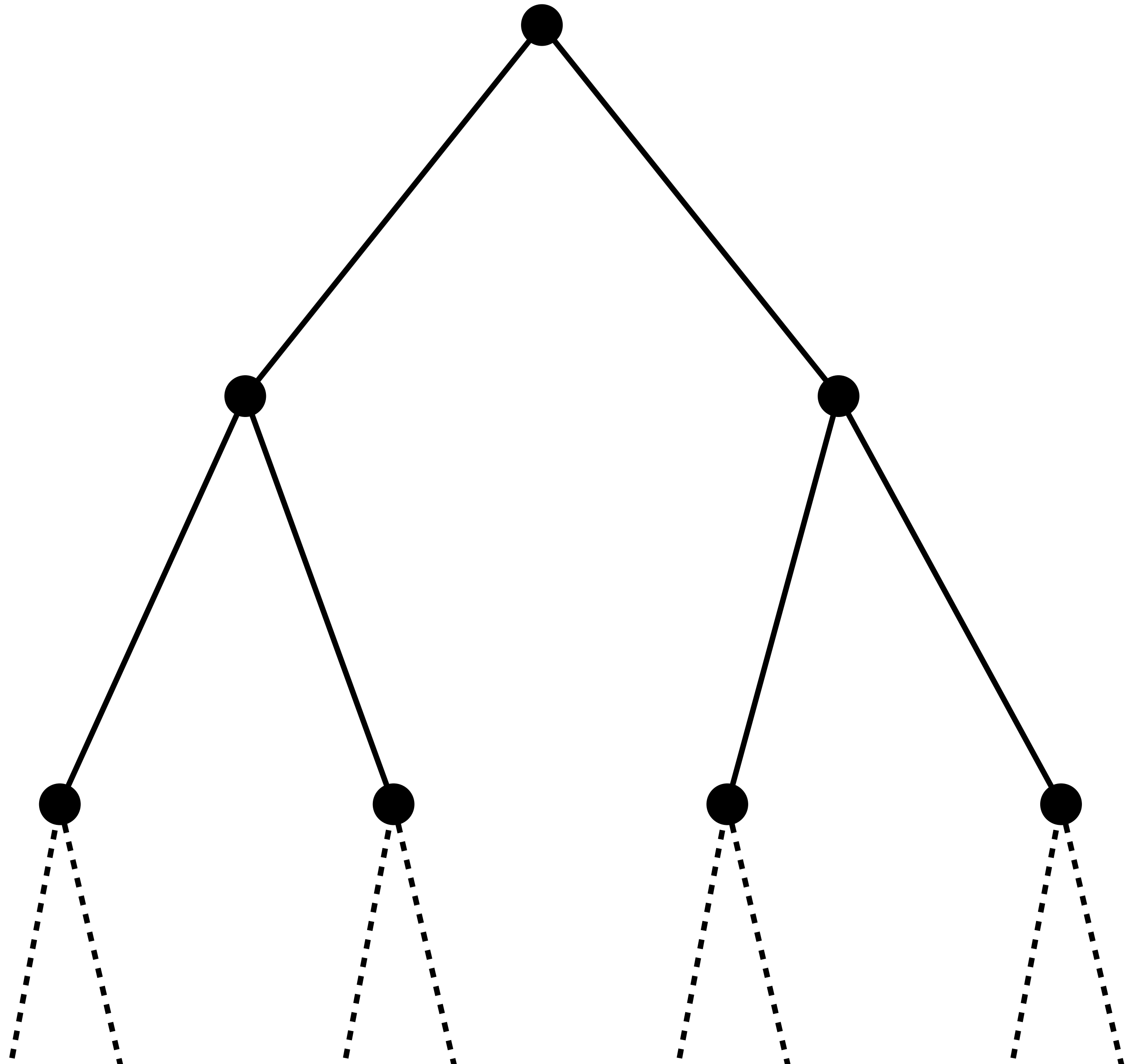
If the dividing membranes receive the sequence of inputs $(m_1, \dots, m_t)$, how many instances of object a are sent out at time $t + 1$?



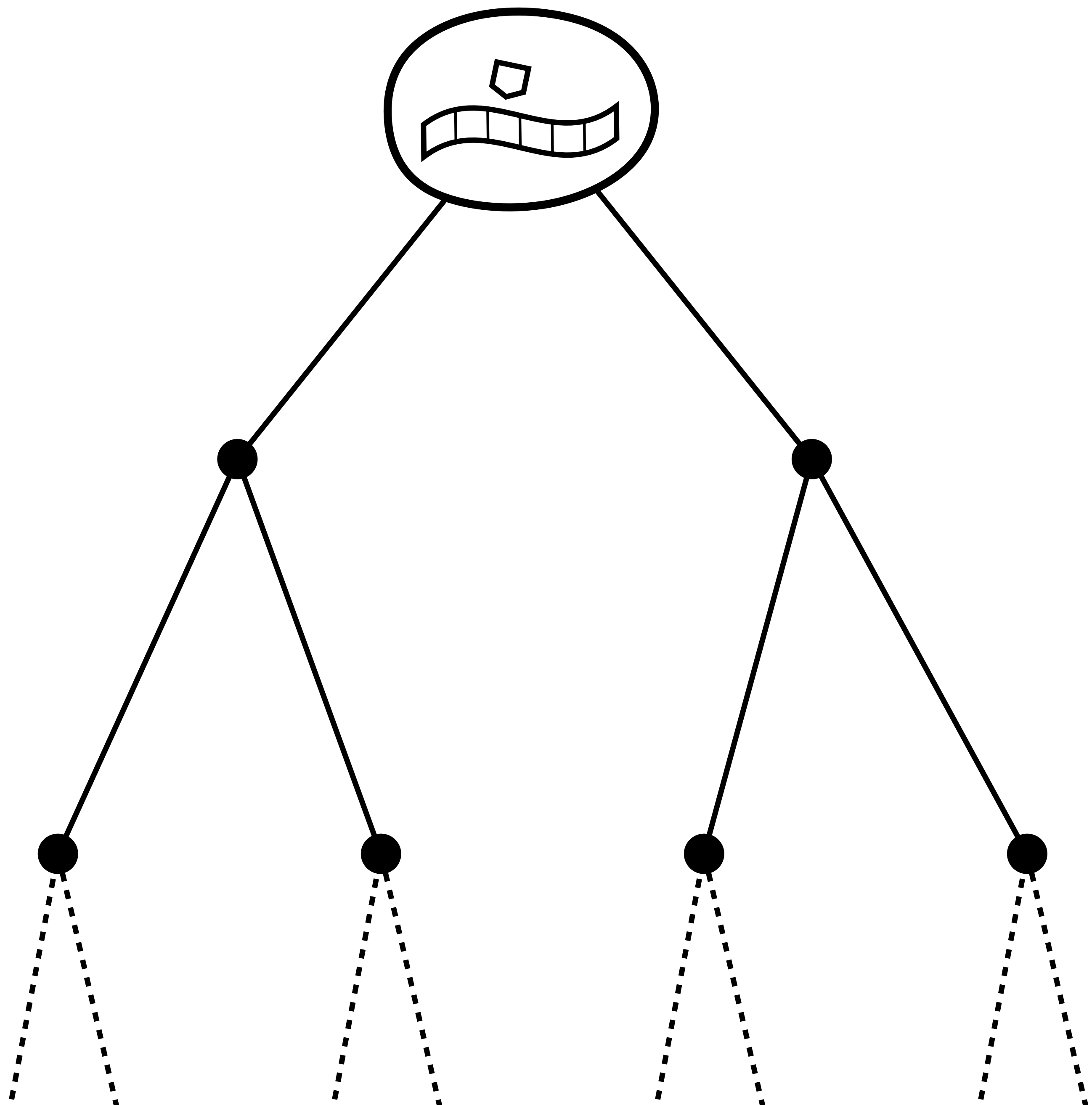
Exact characterisation of $P^{\#P}$

Theorem. Membrane systems where membranes can only divide if they do not contain recursively other membranes characterise $P^{\#P}$ in polynomial time

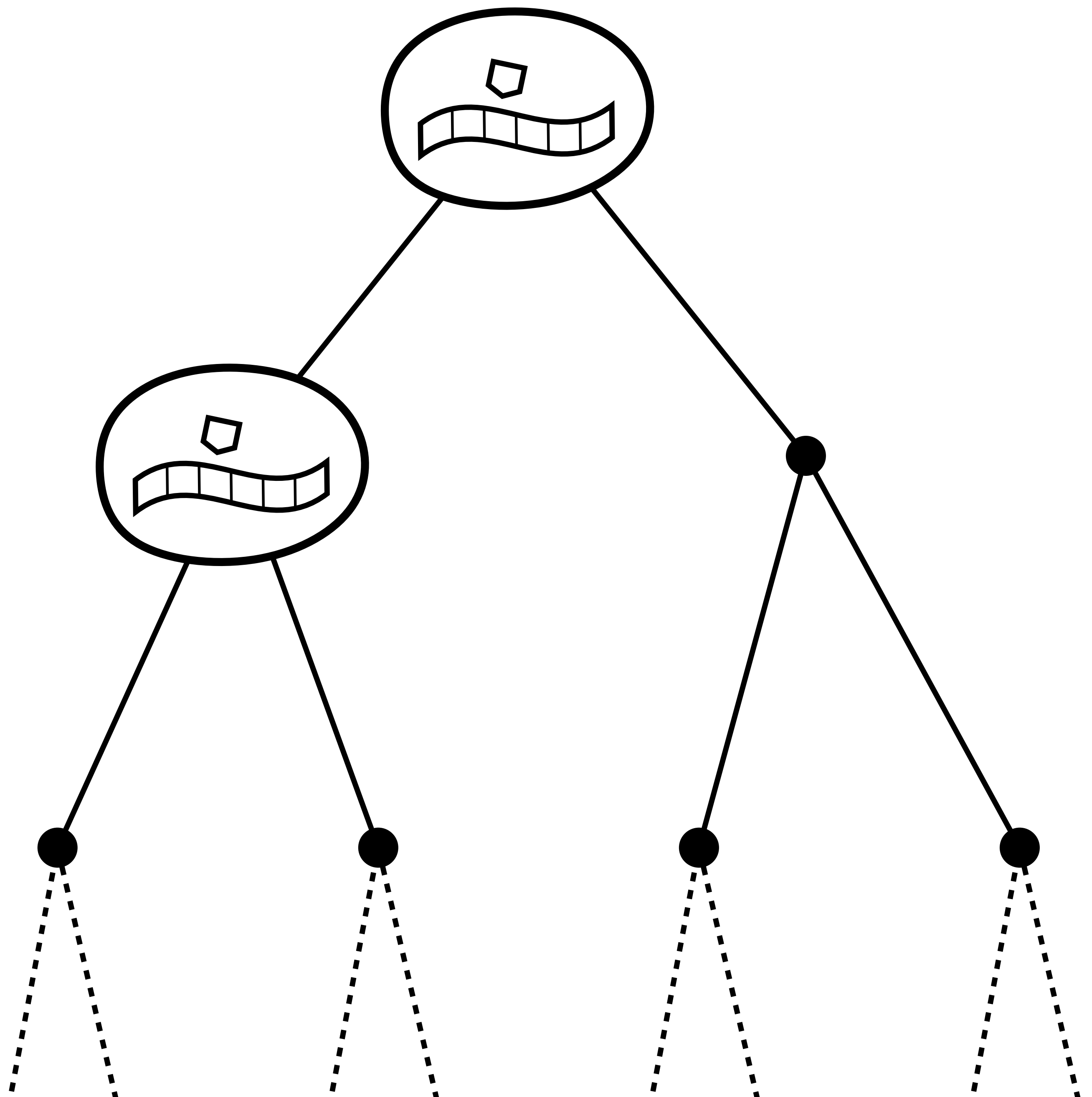
Solving **PSPACE** efficiently in the “binary tree space”



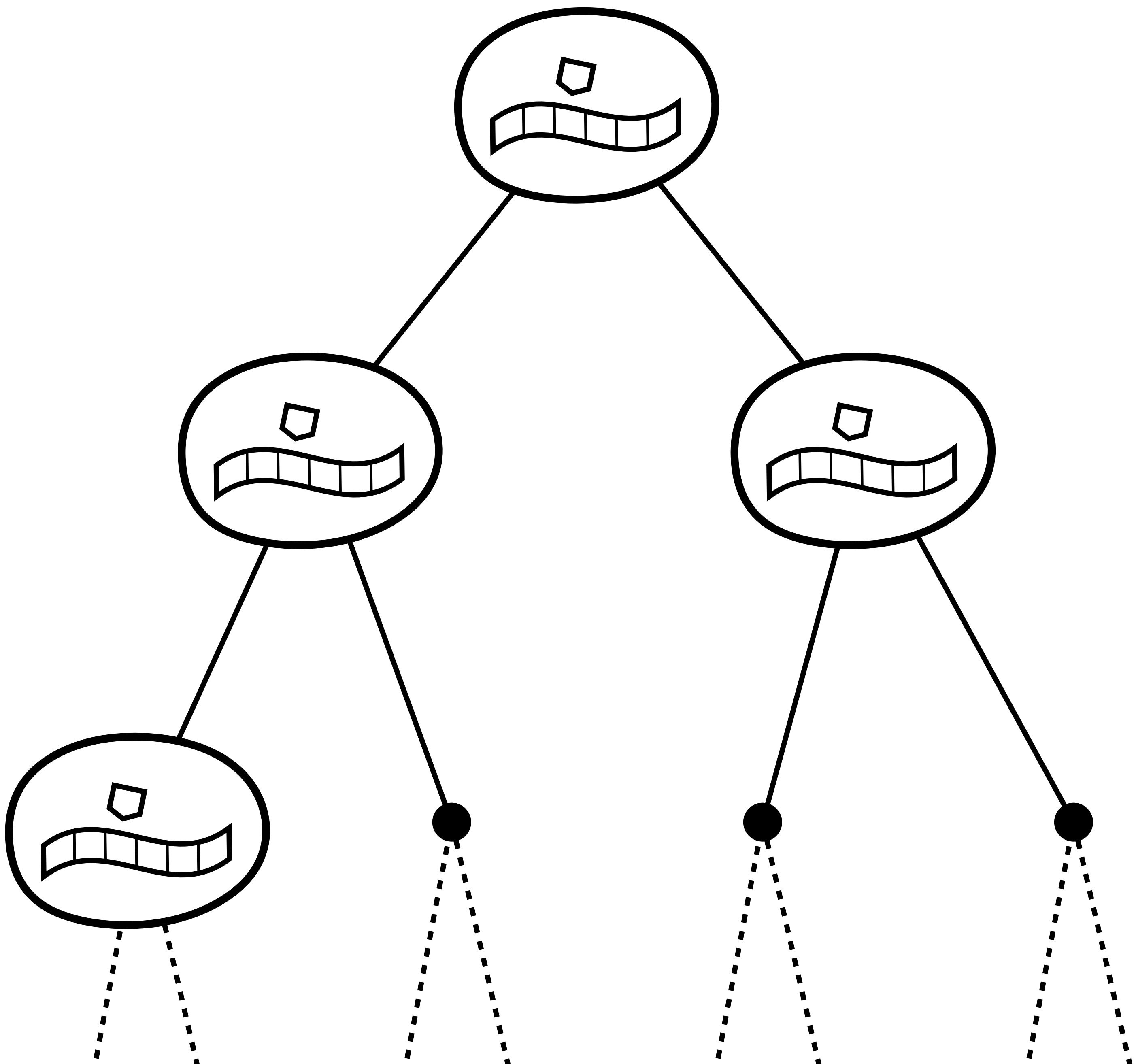
Solving **PSPACE** efficiently in the "binary tree space"



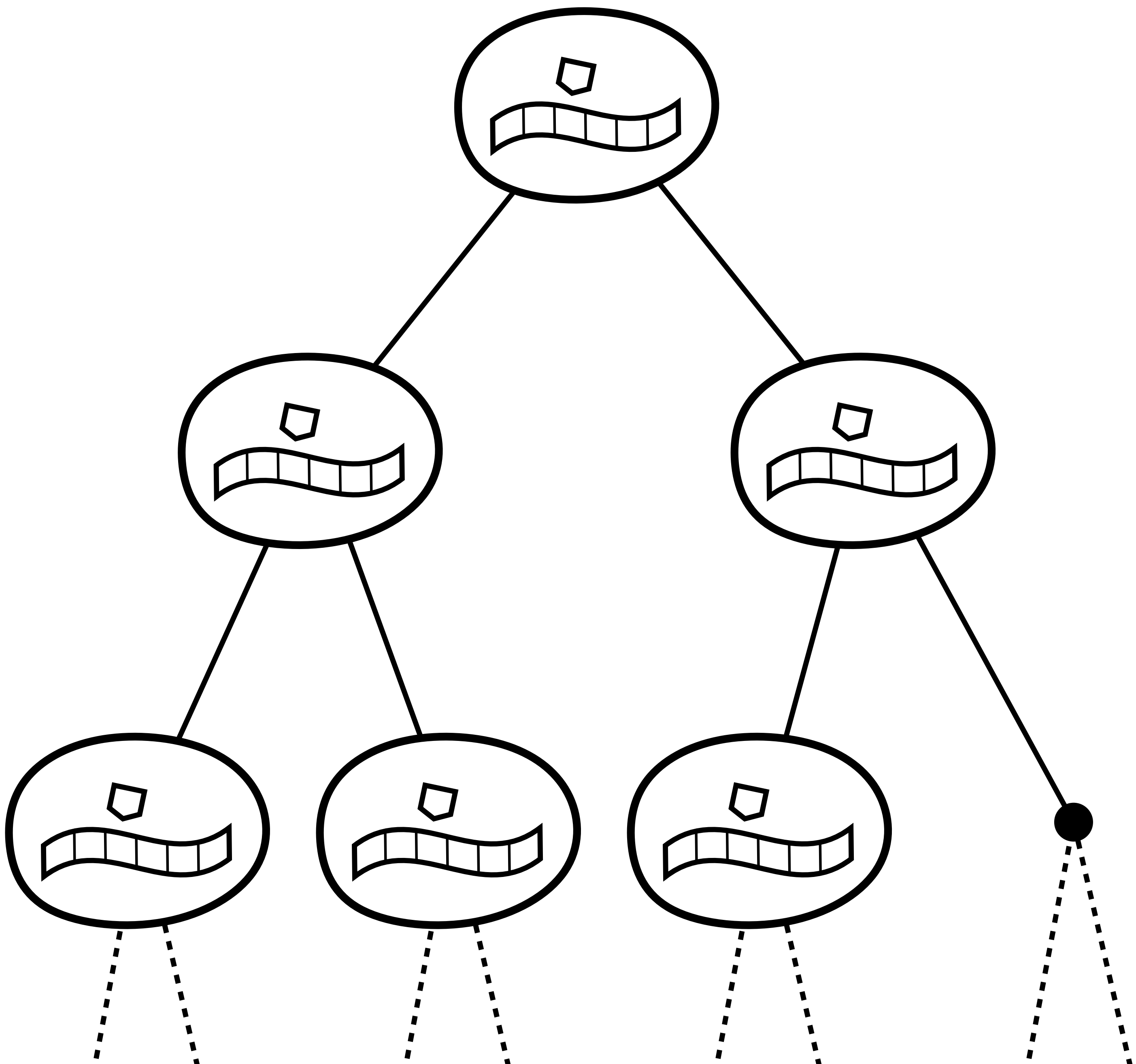
Solving **PSPACE** efficiently in the “binary tree space”



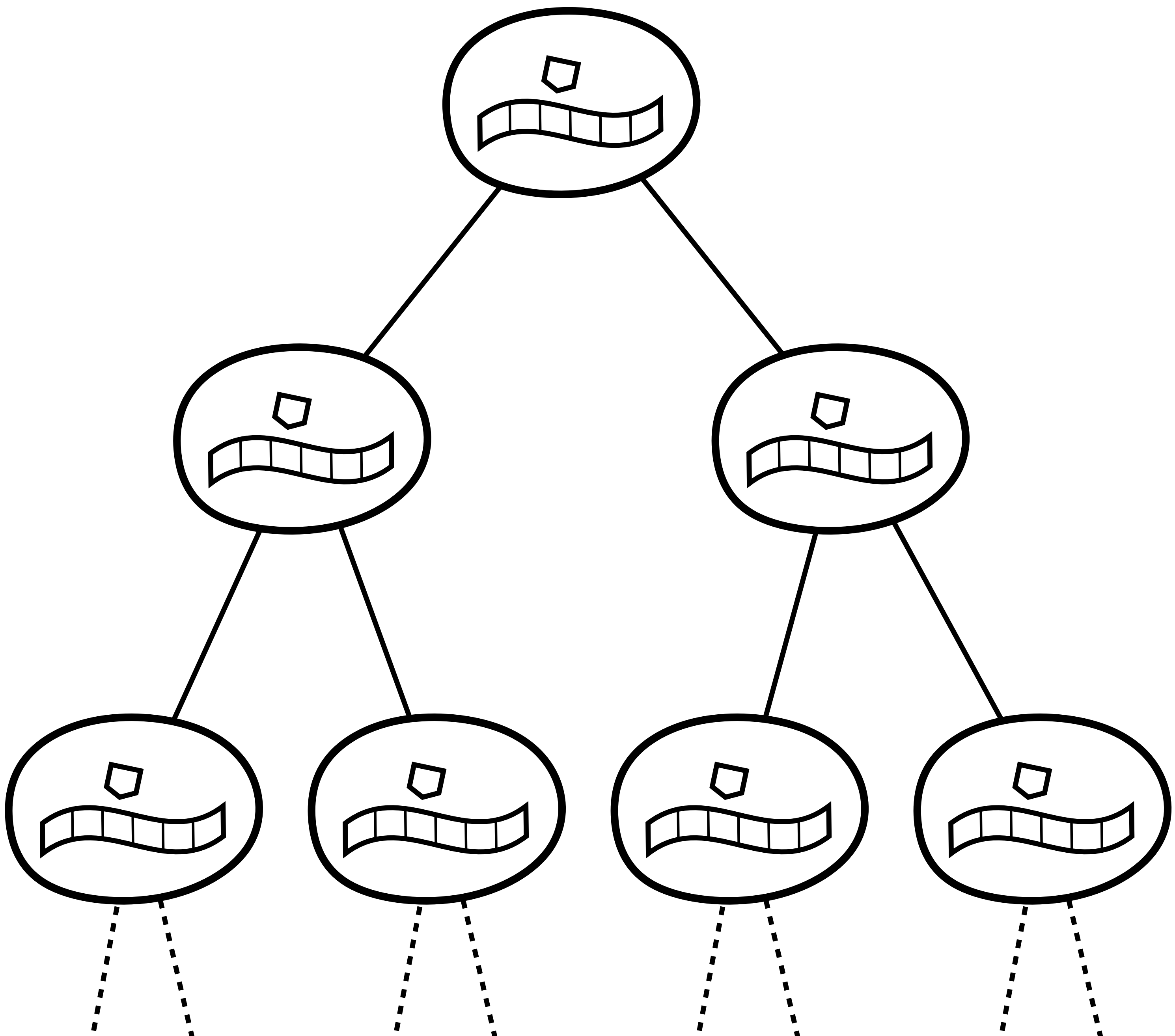
Solving **PSPACE** efficiently in the "binary tree space"



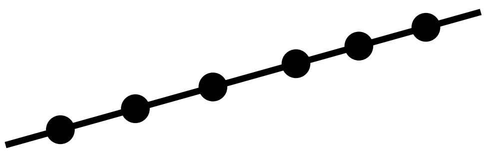
Solving **PSPACE** efficiently in the "binary tree space"

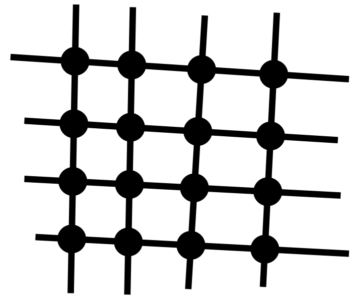


Solving **PSPACE** efficiently in the “binary tree space”

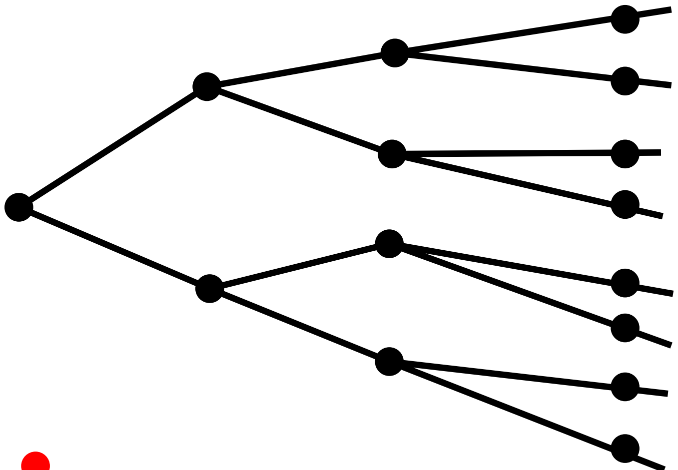


Communication topologies and complexity classes

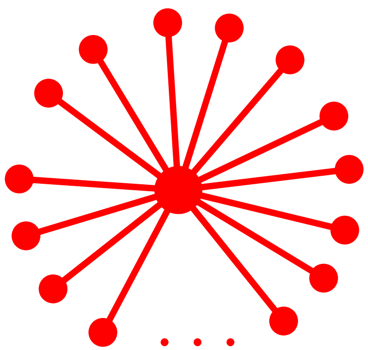




P

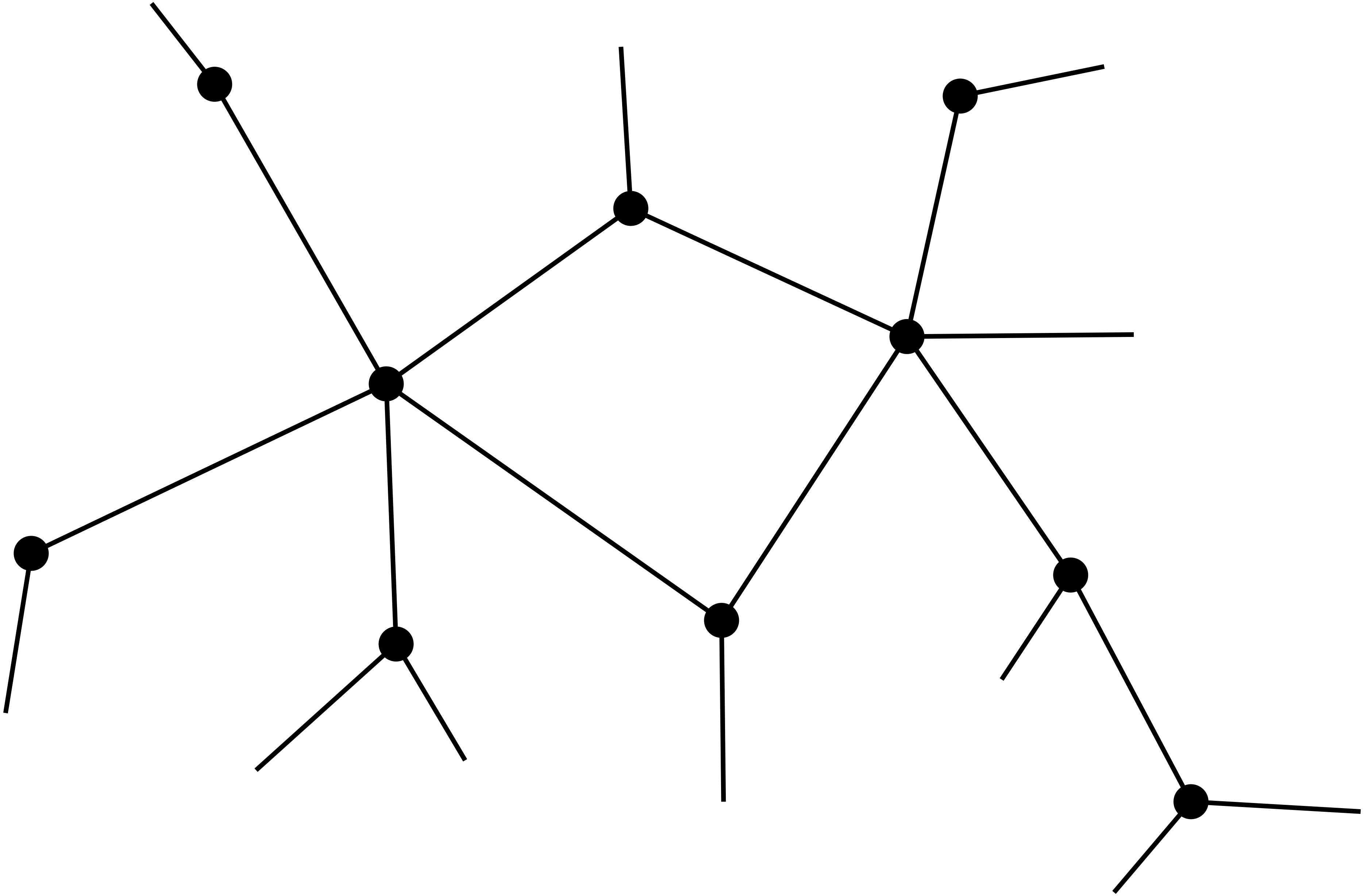


PSPACE

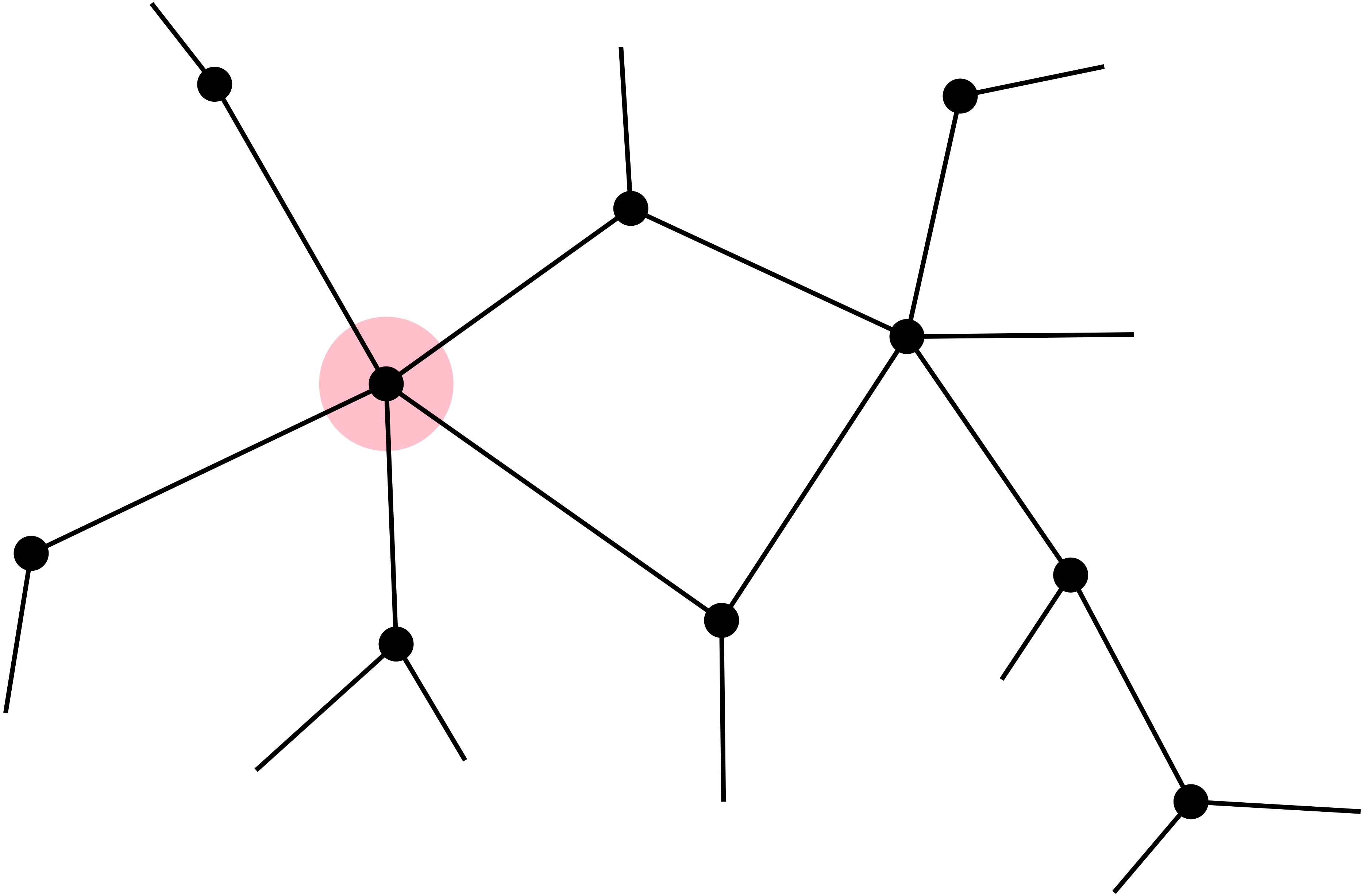


$P^{\#P}$

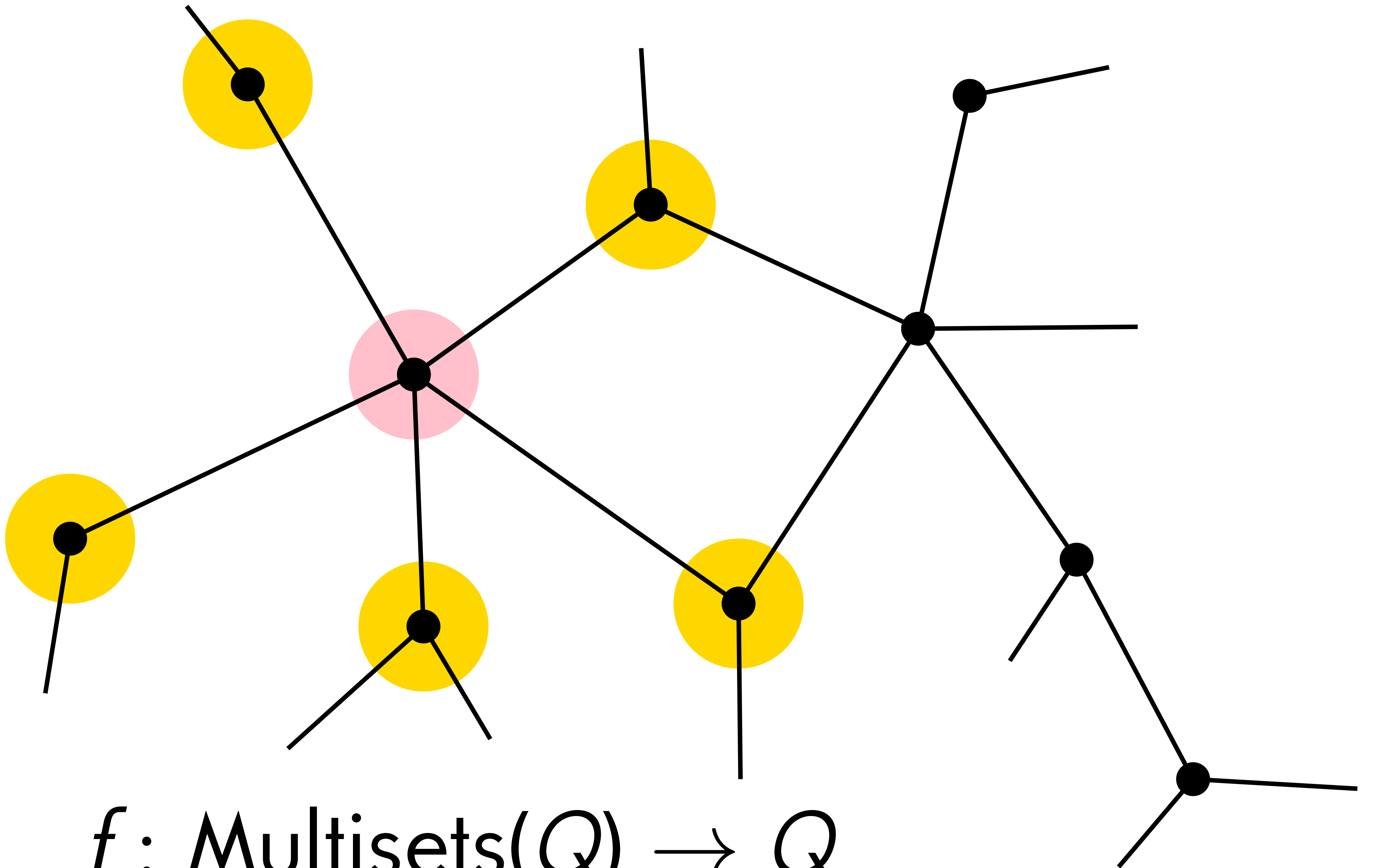
Automata networks over infinite graphs



Automata networks over infinite graphs

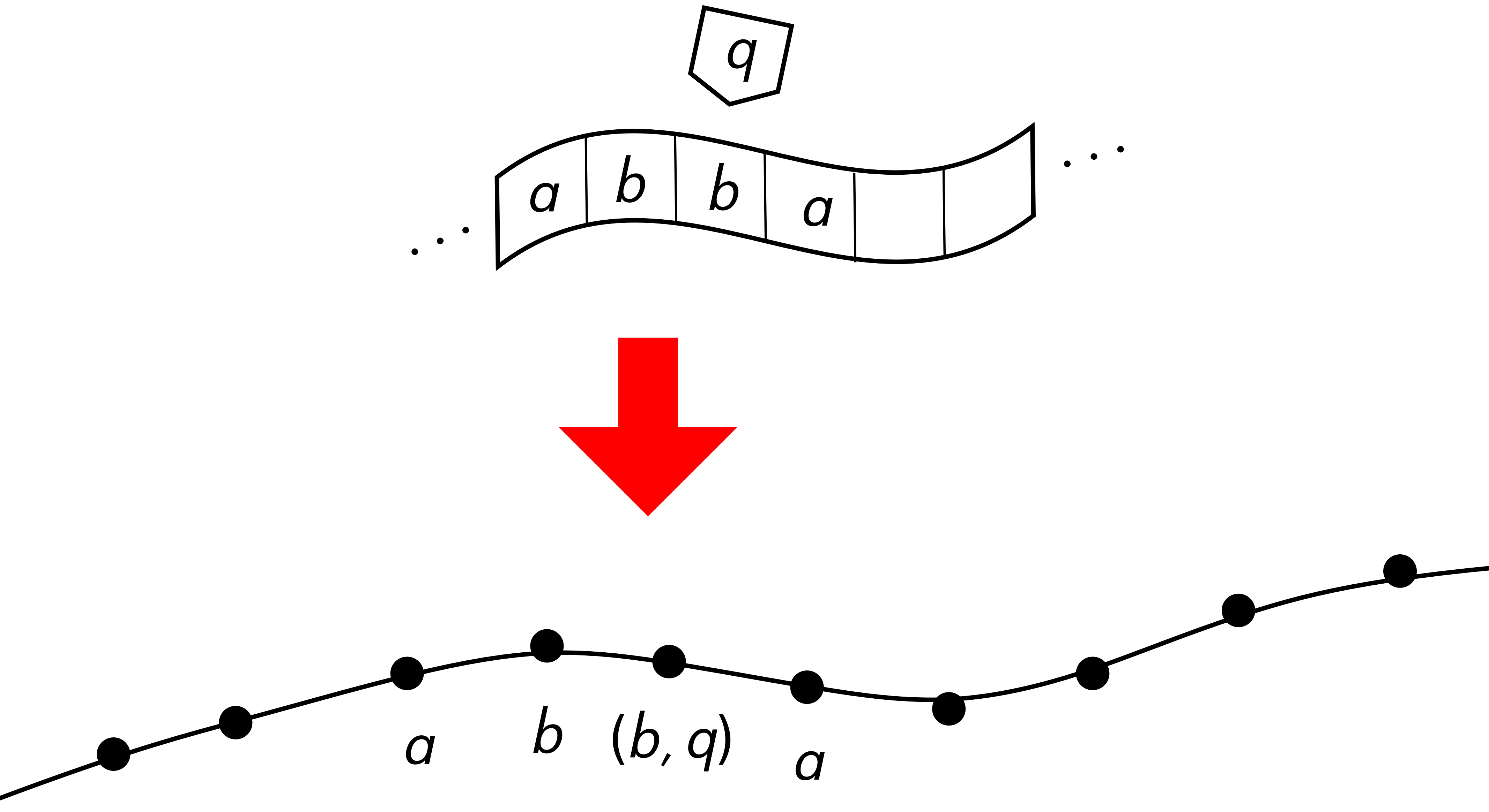


Automata networks over infinite graphs

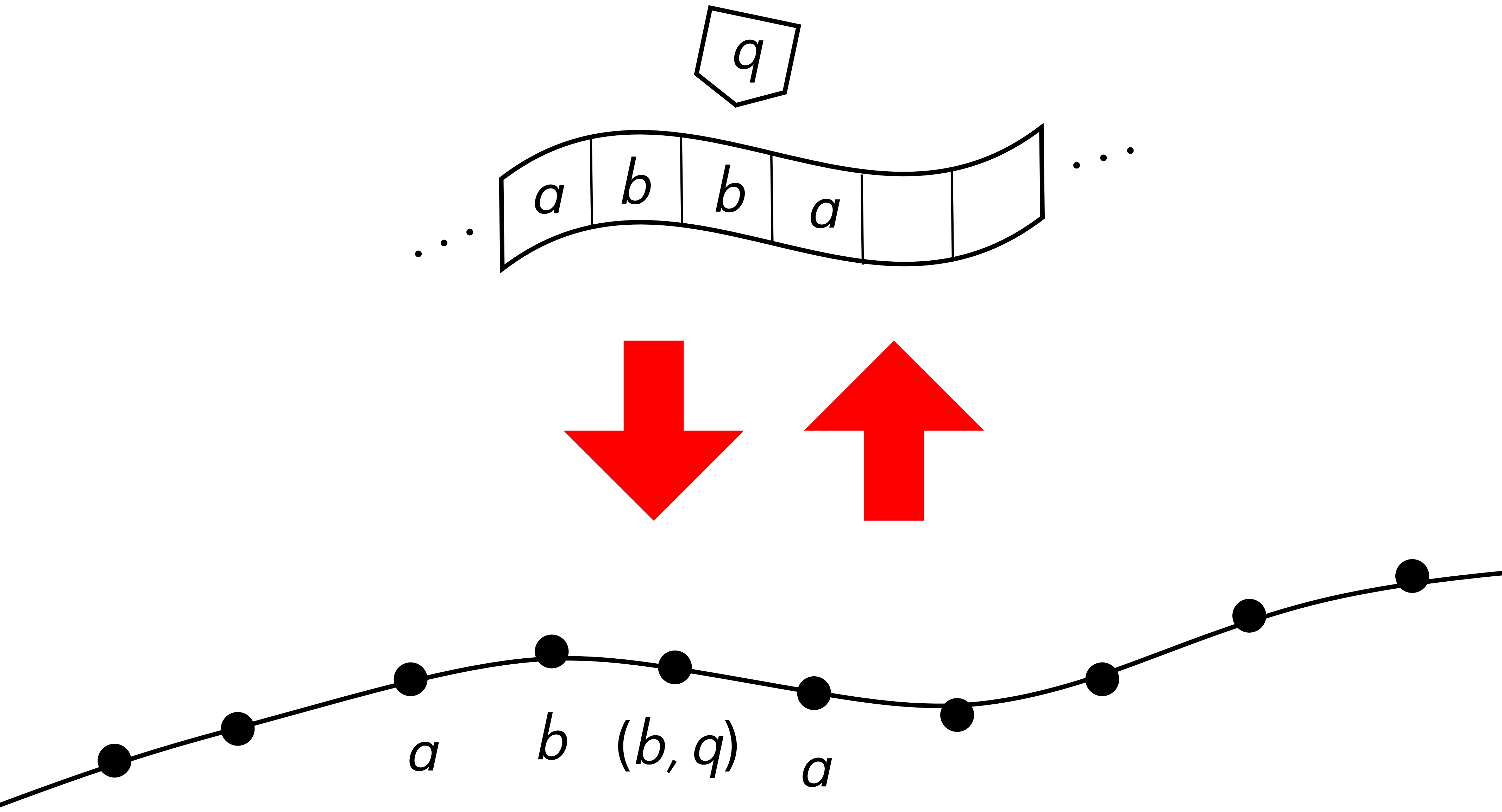


$$f : \mathrm{Multisets}(Q) \to Q$$

1D cellular automata as generalised Turing machines



1D cellular automata as generalised Turing machines



Things to do

- Choose restrictions on the class
  of local transition functions $f\colon \text{Multisets}(Q) \to Q$
  e.g., threshold functions
- Choose a way to encode the input in the initial configuration

Defining new complexity classes

Definition. Given an infinite graph G, let

- **P**(G) be the problems solved by automata networks over G in polynomial time
- **PSPACE**(G) = polynomial space over G
- **EXPTIME**(G) = exponential time over G
- etc.

but also

- **LOGTIME**(G) = logarithmic time over G
- etc.

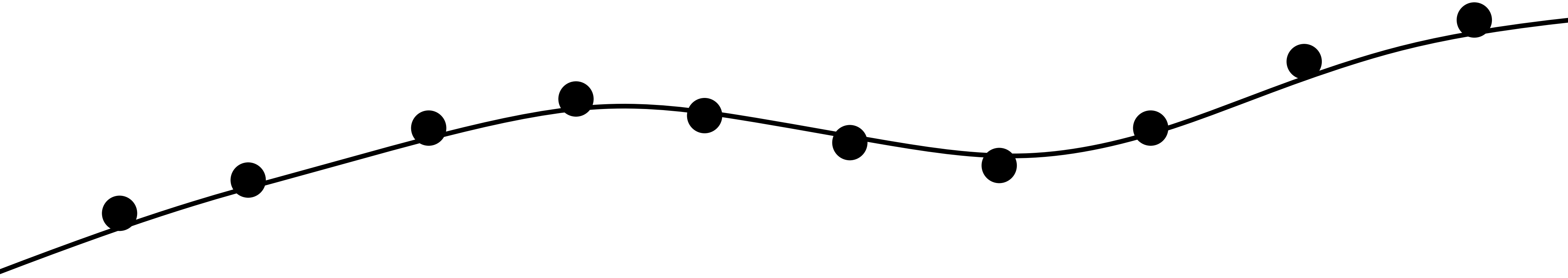
because automata networks are parallel

Preliminary results

- **P**(linear graph) = **P**
- **PSPACE**(linear graph) = **PSPACE**

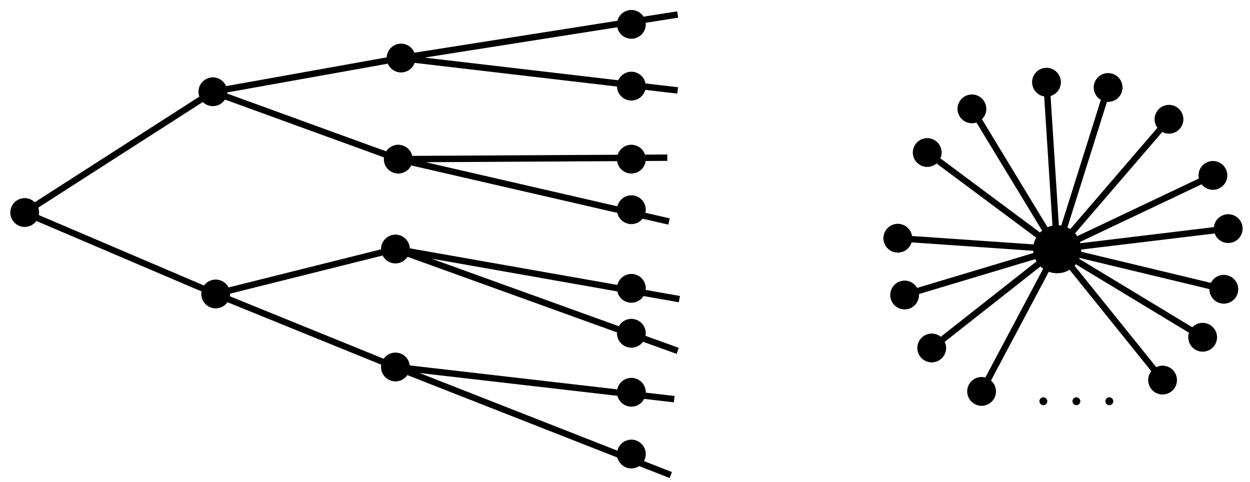
because of the equivalence with Turing machines

- **P**(efficiently computable graph in $\mathbf{R}^d$) = **P**



Expected results

- **P**(infinite binary tree) = **PSPACE**
- **P**(infinite star or variant thereof) = $\mathbf{P}^{\#\mathbf{P}}$



- **P**(non-computable graph) includes undecidable problems

Hopeful developments of the theory

- Find graphs (or classes of graphs) characterising all standard complexity classes
- Find new intermediate complexity classes using "natural" graphs

Hopeful developments of the theory

- Find graphs (or classes of graphs) characterising all standard complexity classes
- Find new intermediate complexity classes using "natural" graphs

- Find algorithms working on all graphs or on certain classes of graphs
- Discover how graph-theoretic or geometric properties of the graphs can speed up or slow down algorithms e.g., algorithms running in time $\Theta(n^{f(d)})$ in $\mathbf{R}^d$

Applications

- Same applications as automata networks (e.g., biology)
- Theory (and practice) of distributed algorithms
- Low-level hardware design
- Machine learning (variants of deep learning? non-Euclidean learning?)

and, of course

- Computability and complexity theory

Thanks for your attention!
Merci de votre attention!

Any questions?