# Counting complexity and oracles 🧞 in natural computing

Antonio E. Porreca
Laboratoire d’Informatique et Systèmes
Équipe CANA
https://aeporreca.org

# Outline

- Complexity classes for parallel computing models
- Cellular automata in various geometric spaces
- Membrane systems, counting and oracles
- Expanding cellular automata
- Conclusions and future work

# The first and second machine classes

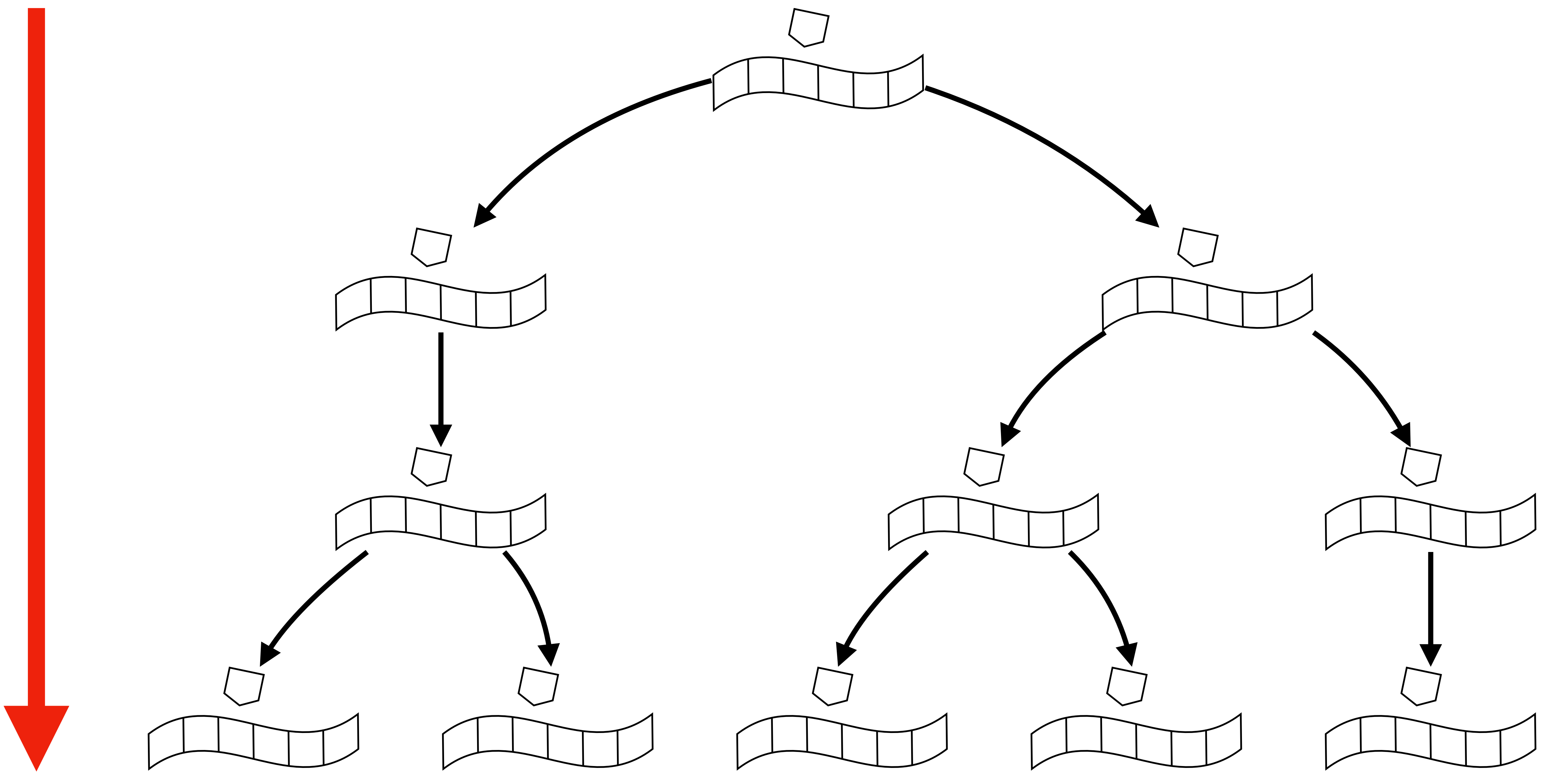
# The first machine class and P

- Includes the deterministic Turing machine and all models that simulate and are simulated by it efficiently:
  - Random access machines (RAM) with constant-time addition and subtraction
  - Cellular automata with a finite initial configuration

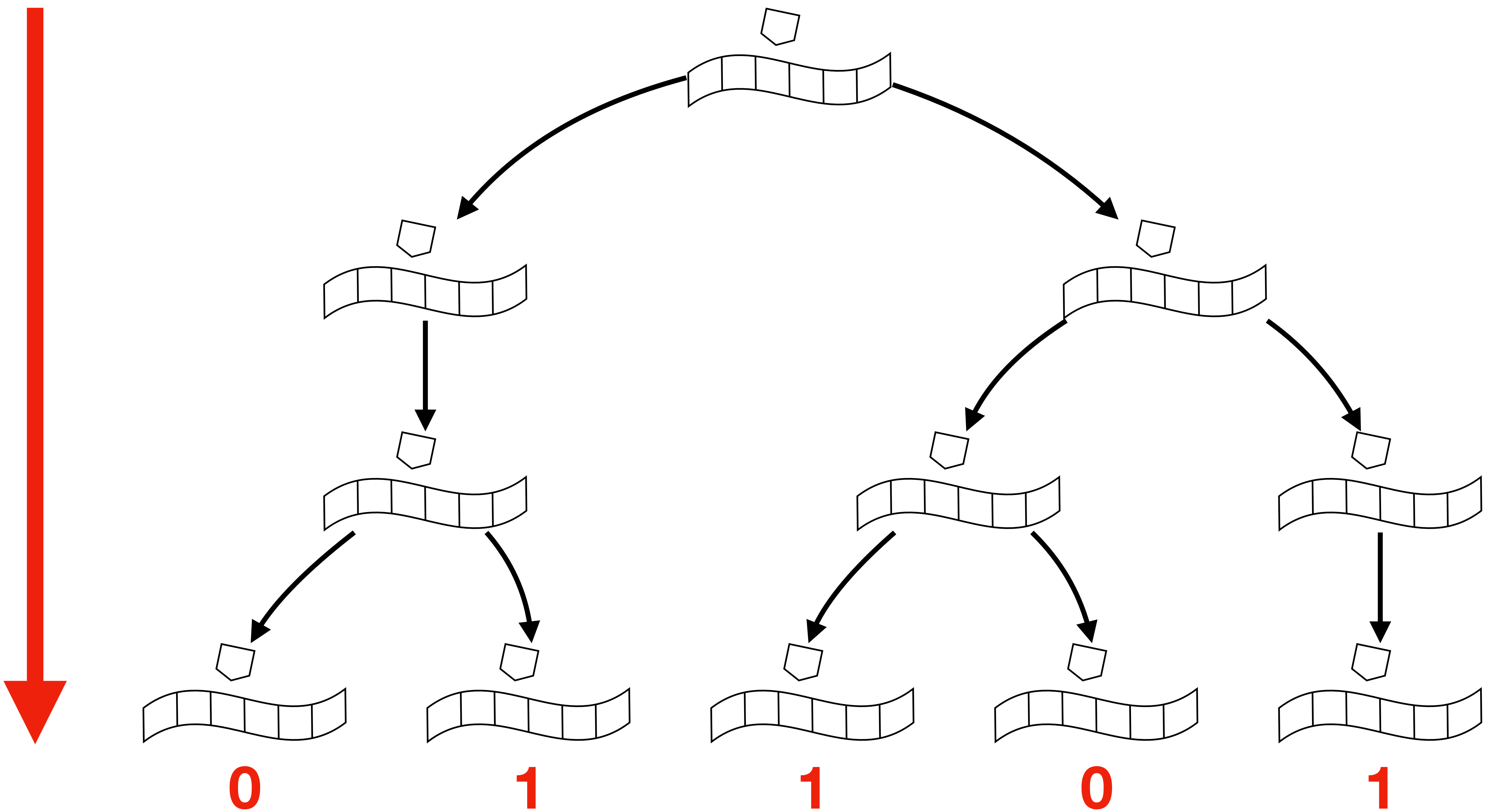
# The second machine class and **PSPACE**

- Includes models of computation that solve in polynomial time what a Turing machine solves in polynomial space:
  - Alternating Turing machines
  - Random access machines including constant time multiplication and division
  - Parallel processes generated by `fork(2)` running on an unbounded number of processors
  - Cellular automata over hyperbolic grids

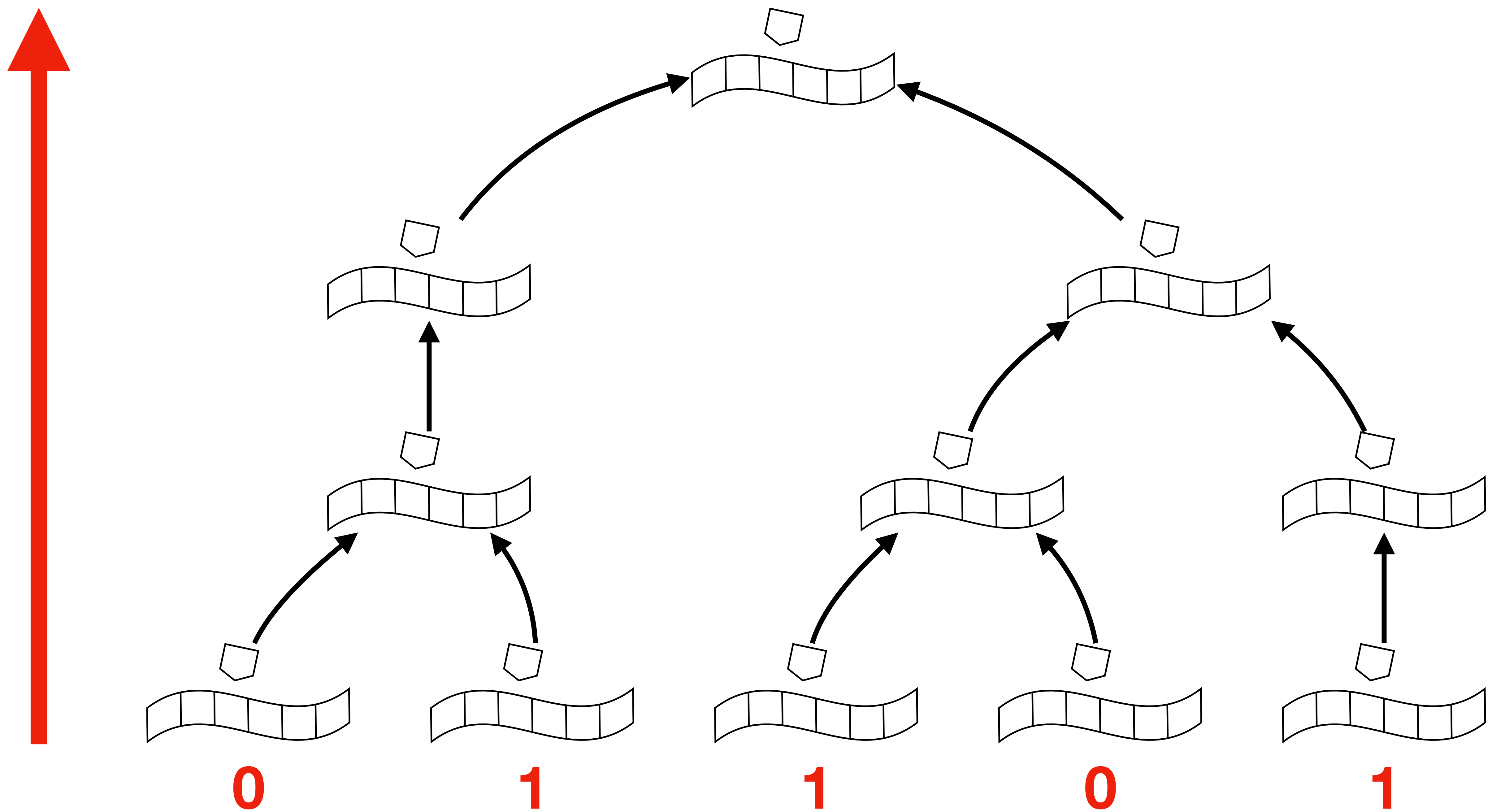
# Nondeterministic Turing machines: **NP**



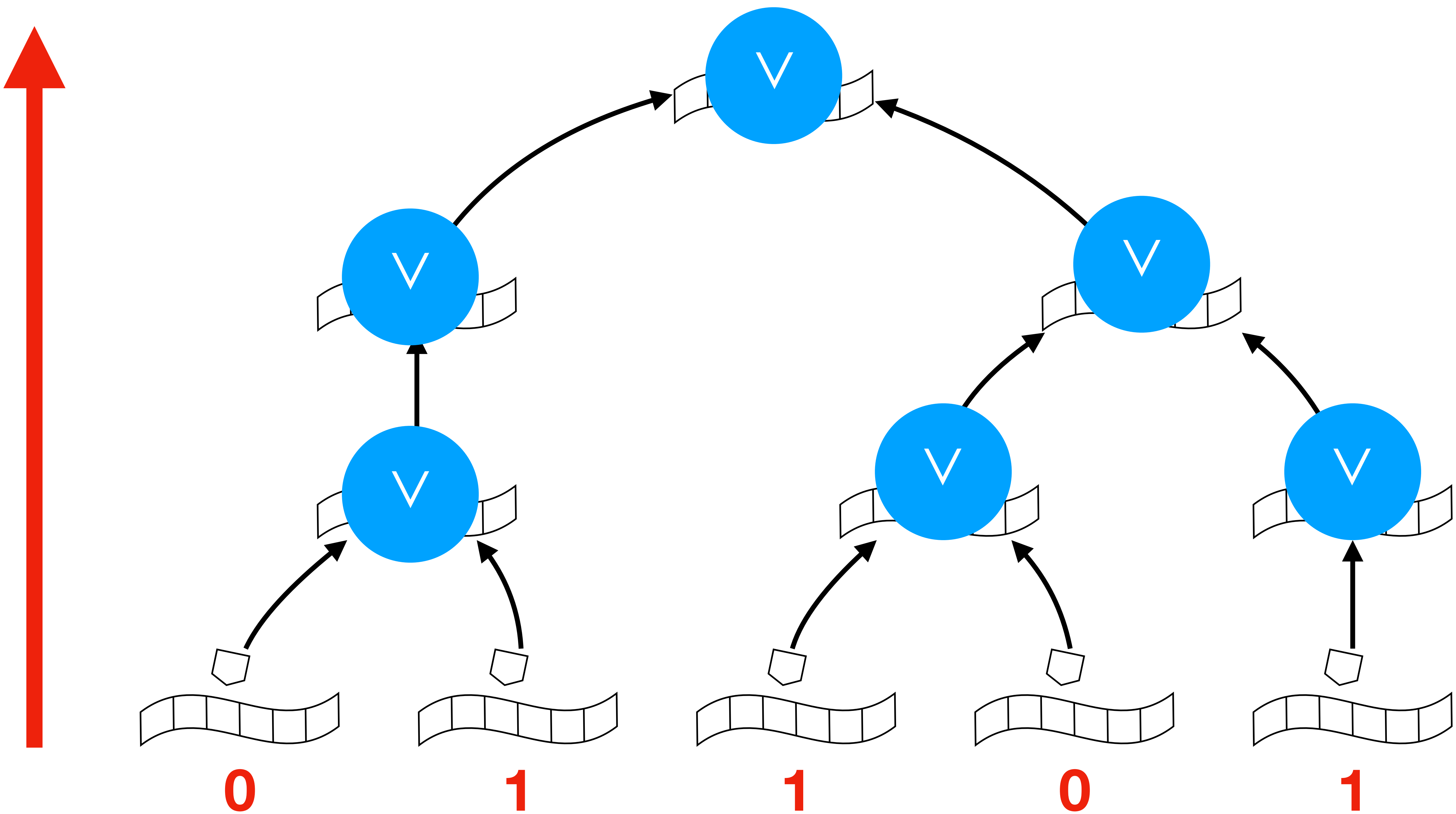
# Nondeterministic Turing machines: **NP**



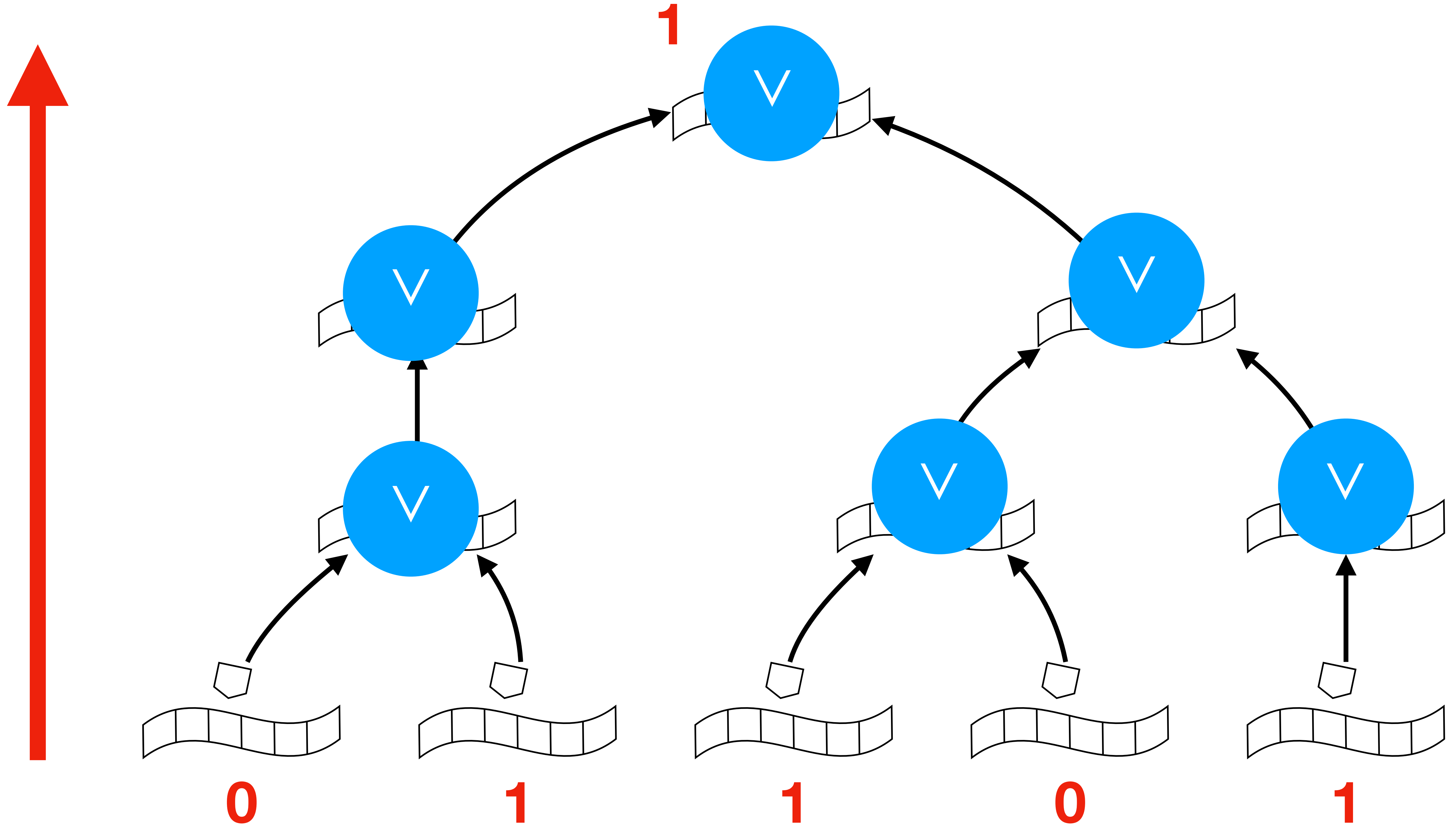
# Nondeterministic Turing machines: **NP**



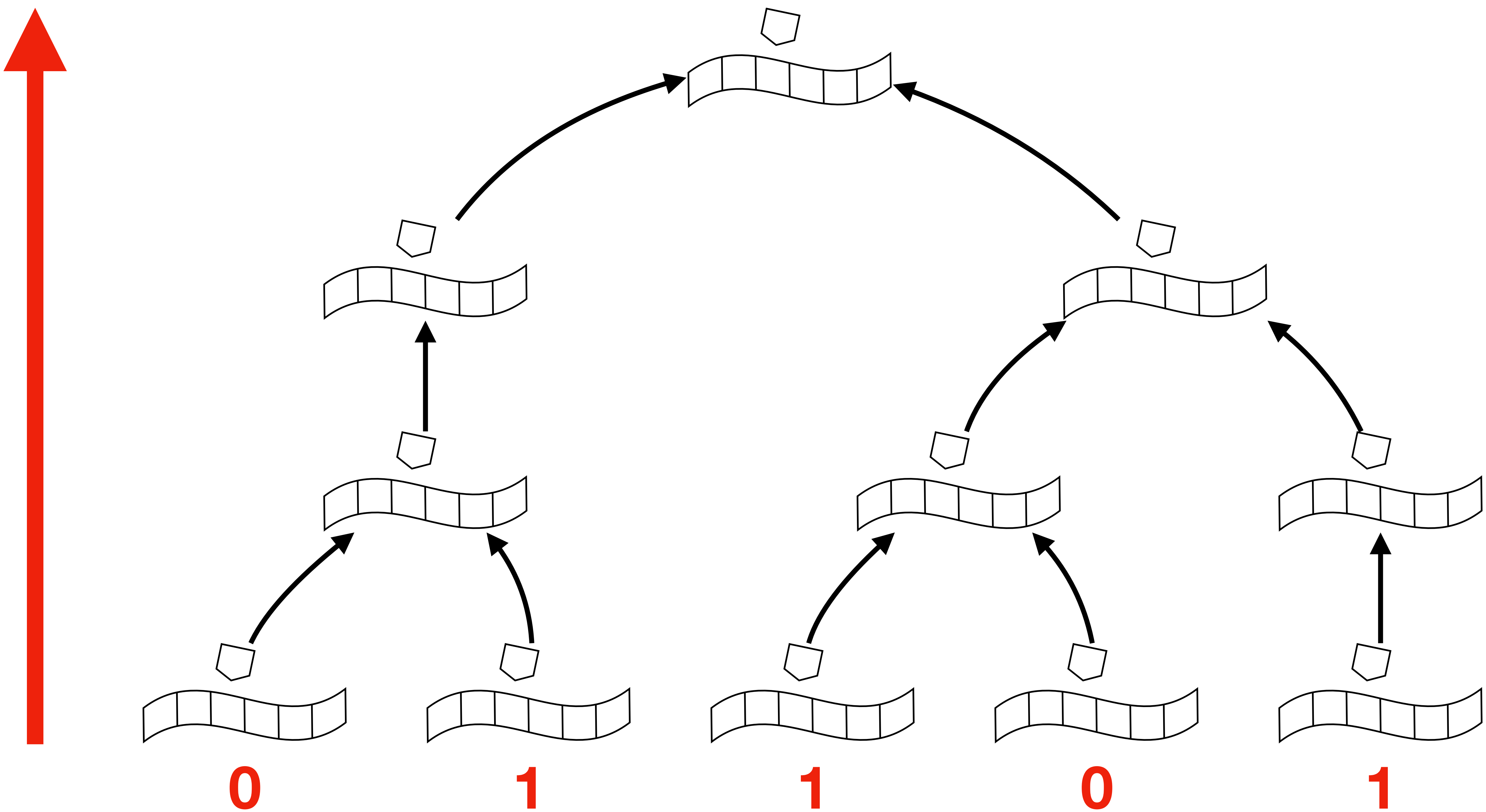
# Nondeterministic Turing machines: **NP**



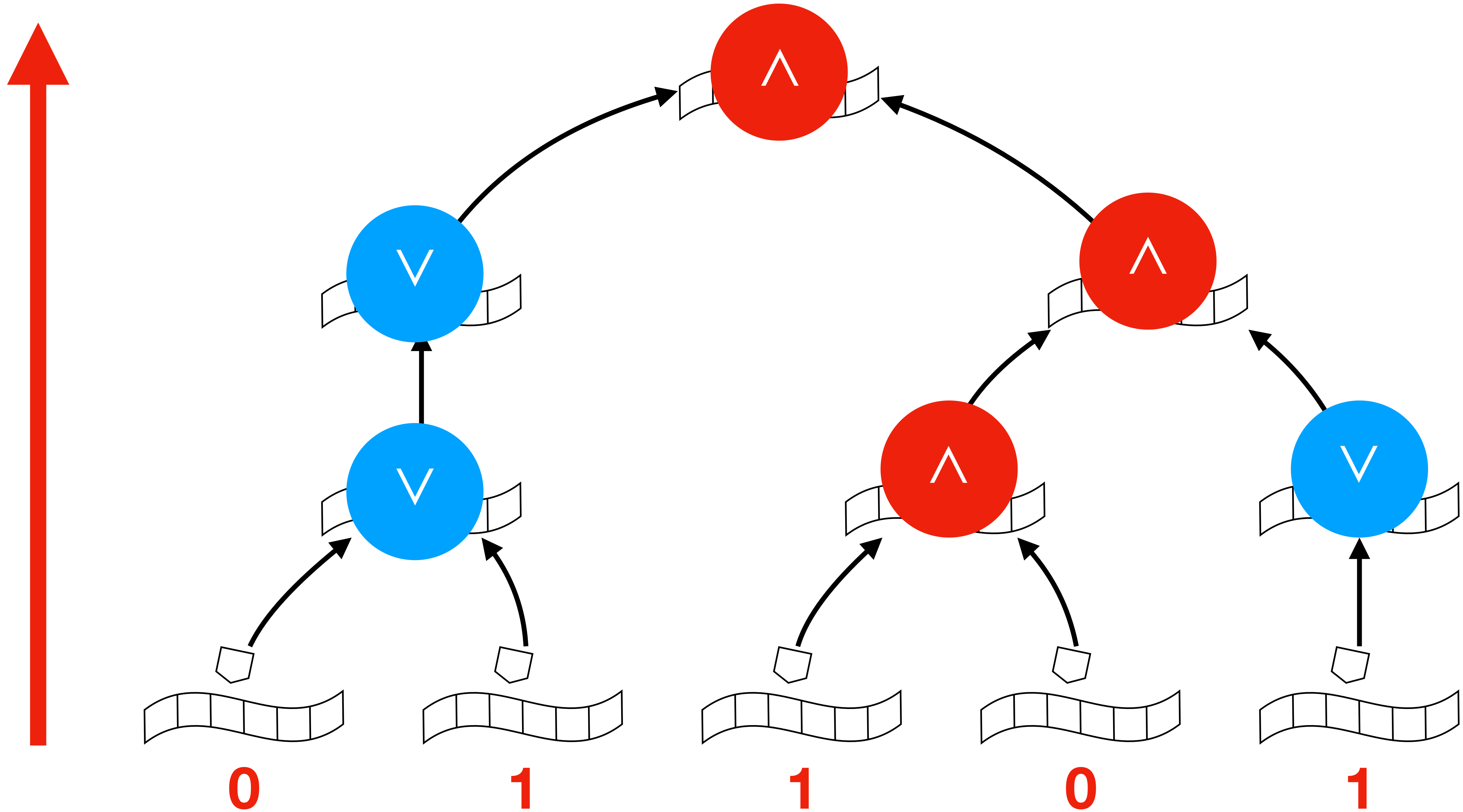
# Nondeterministic Turing machines: **NP**



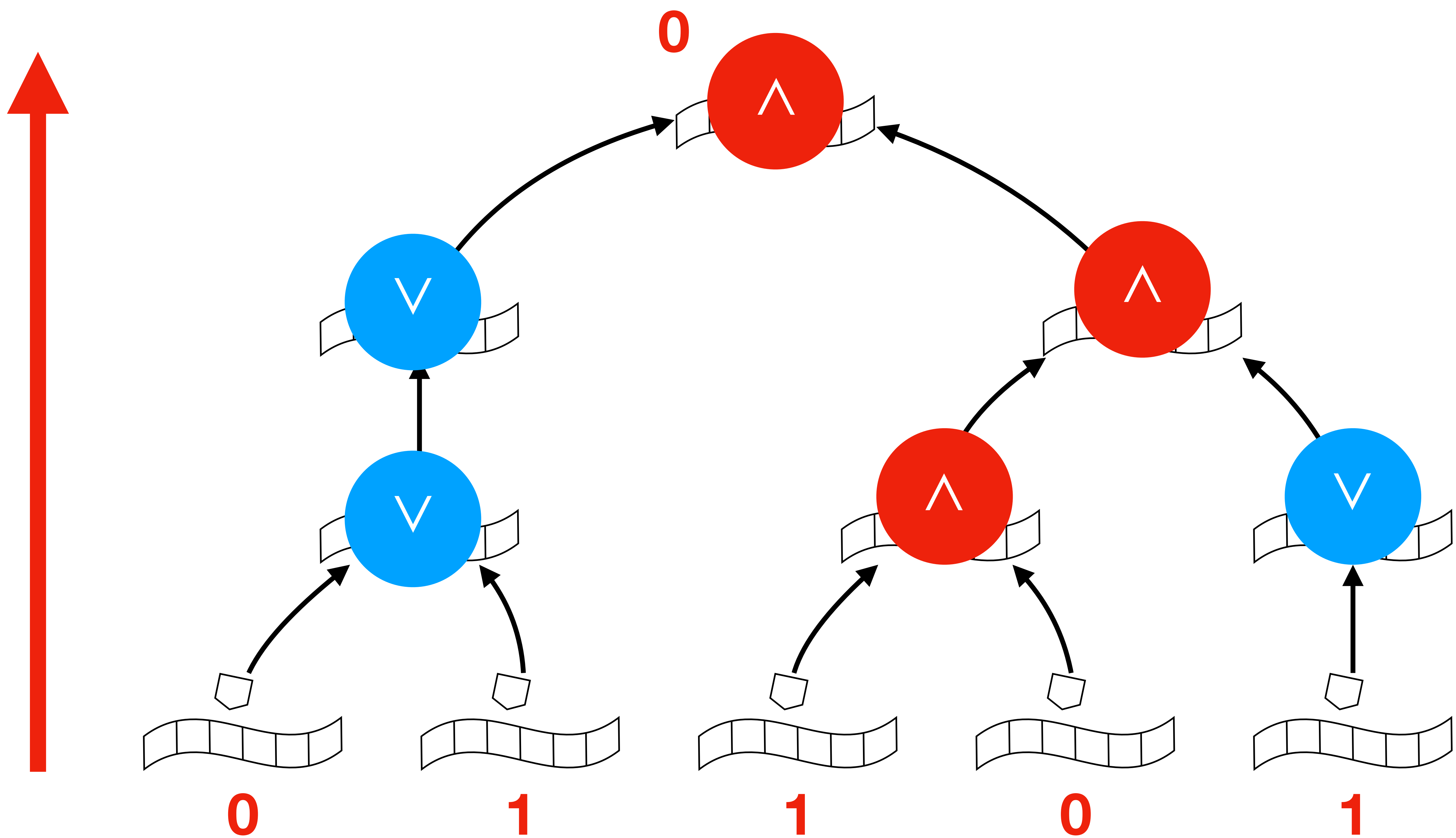
# Alternating Turing machines: **PSPACE**



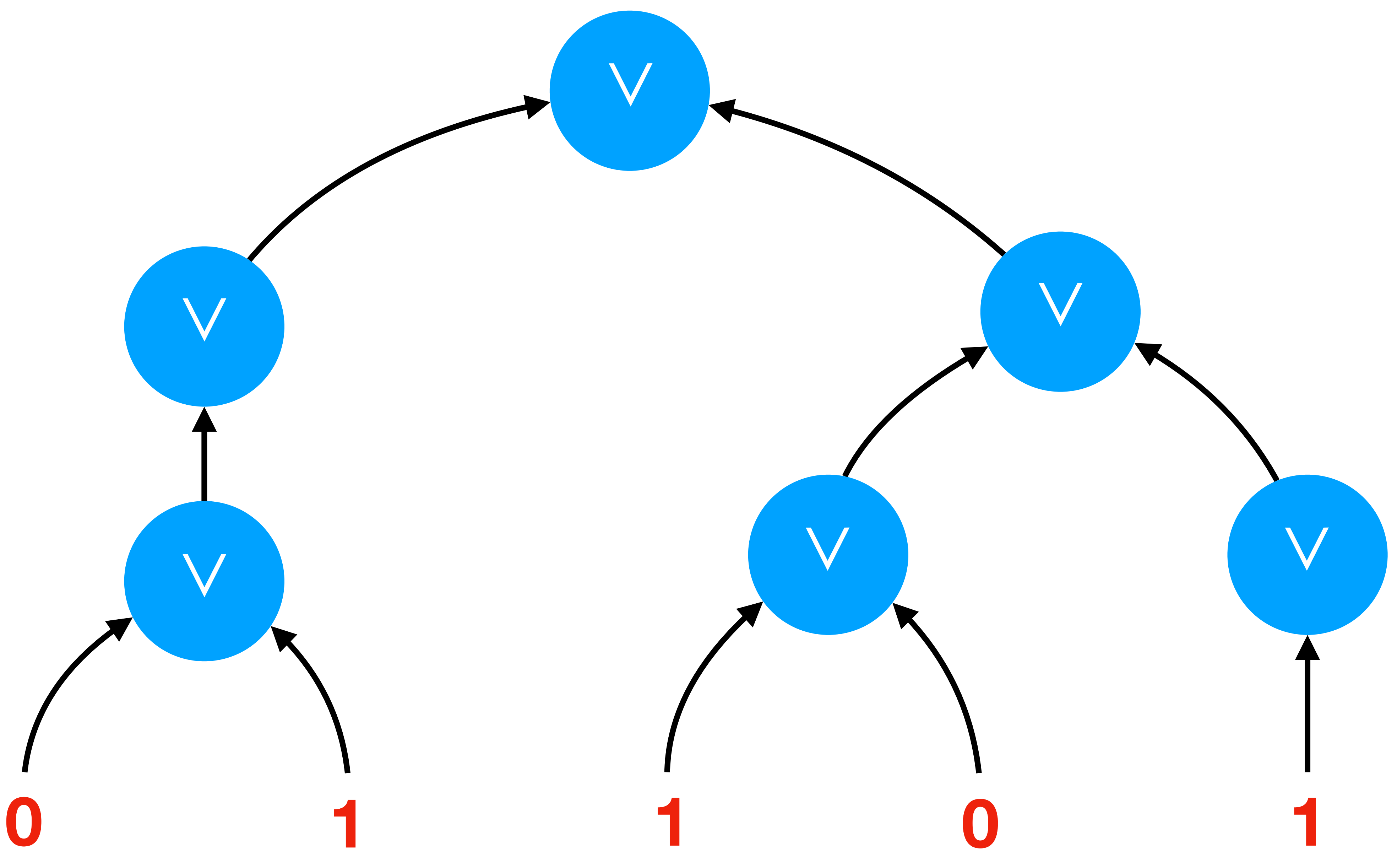
# Alternating Turing machines: **PSPACE**



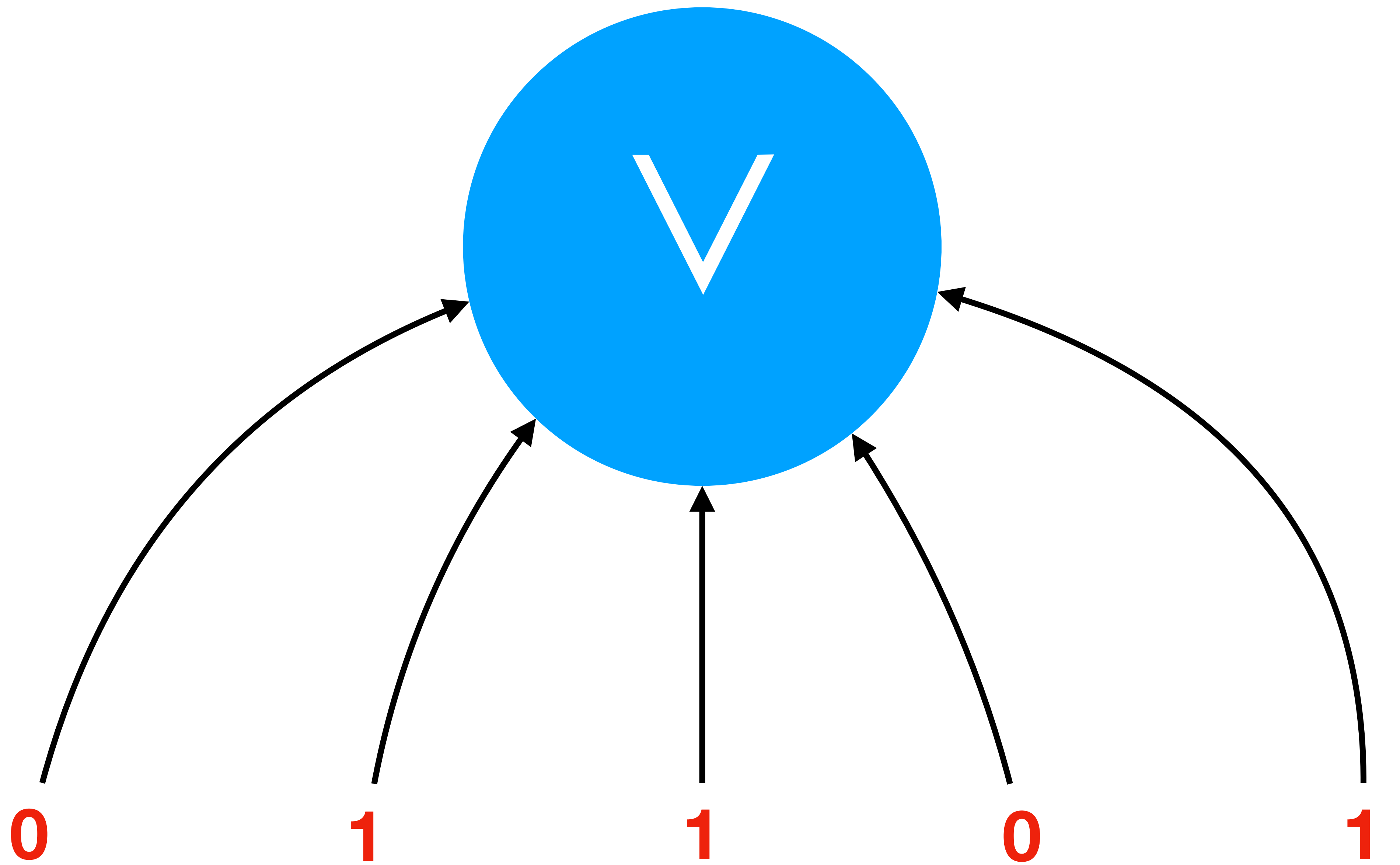
# Alternating Turing machines: **PSPACE**



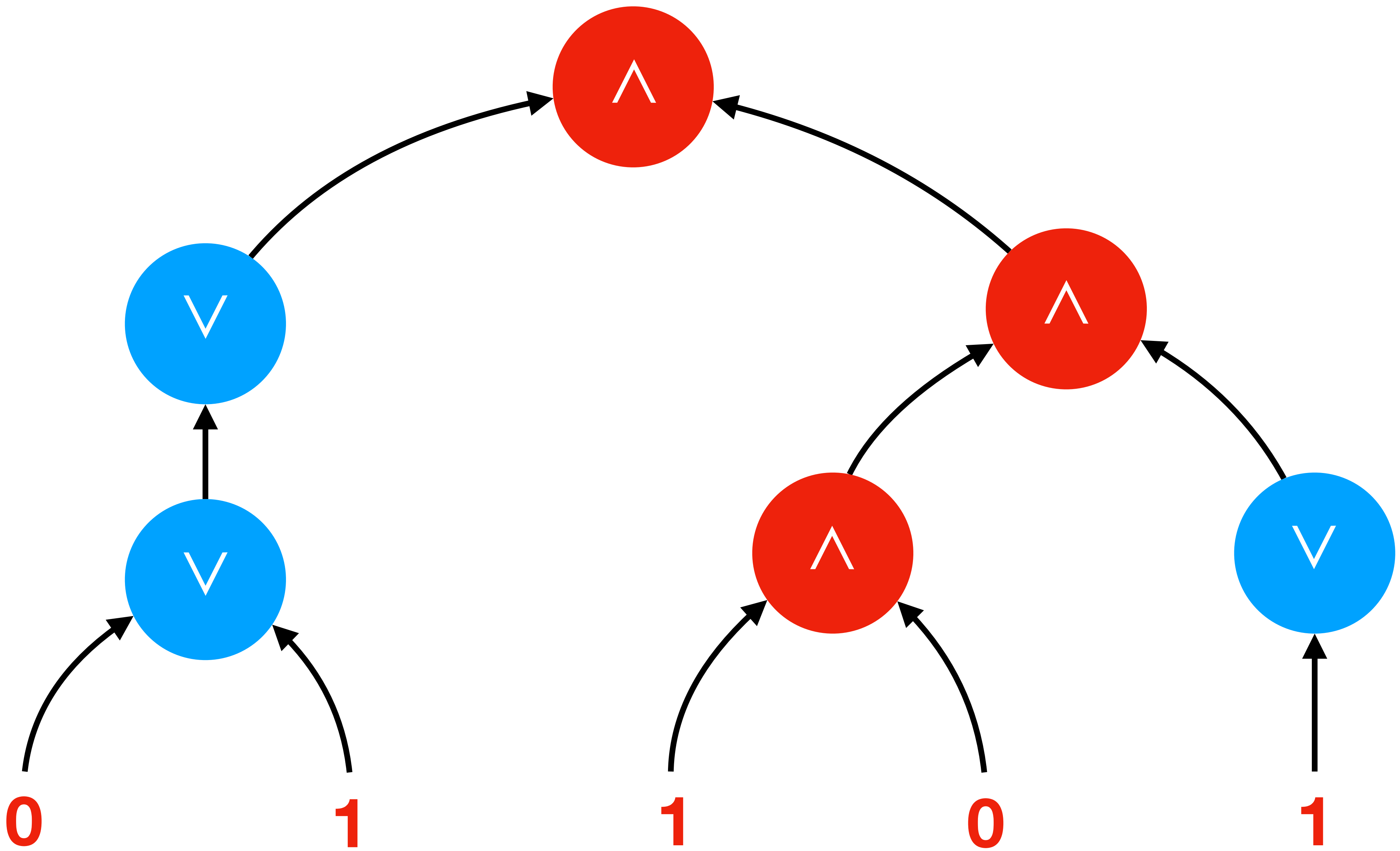
# Flattening ∨-circuits



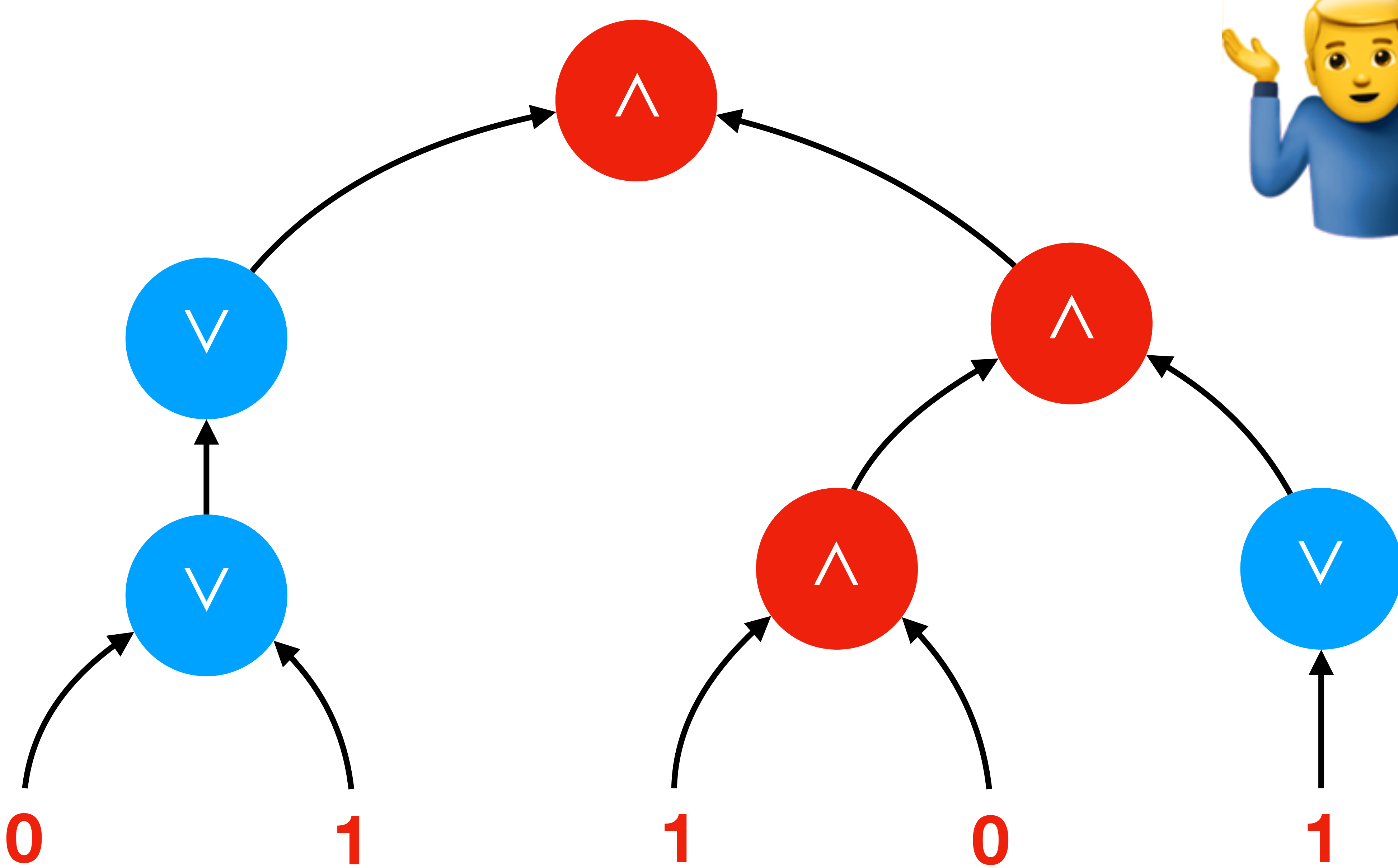
# Flattening ∨-circuits



# Flattening $\wedge\vee$-circuits?

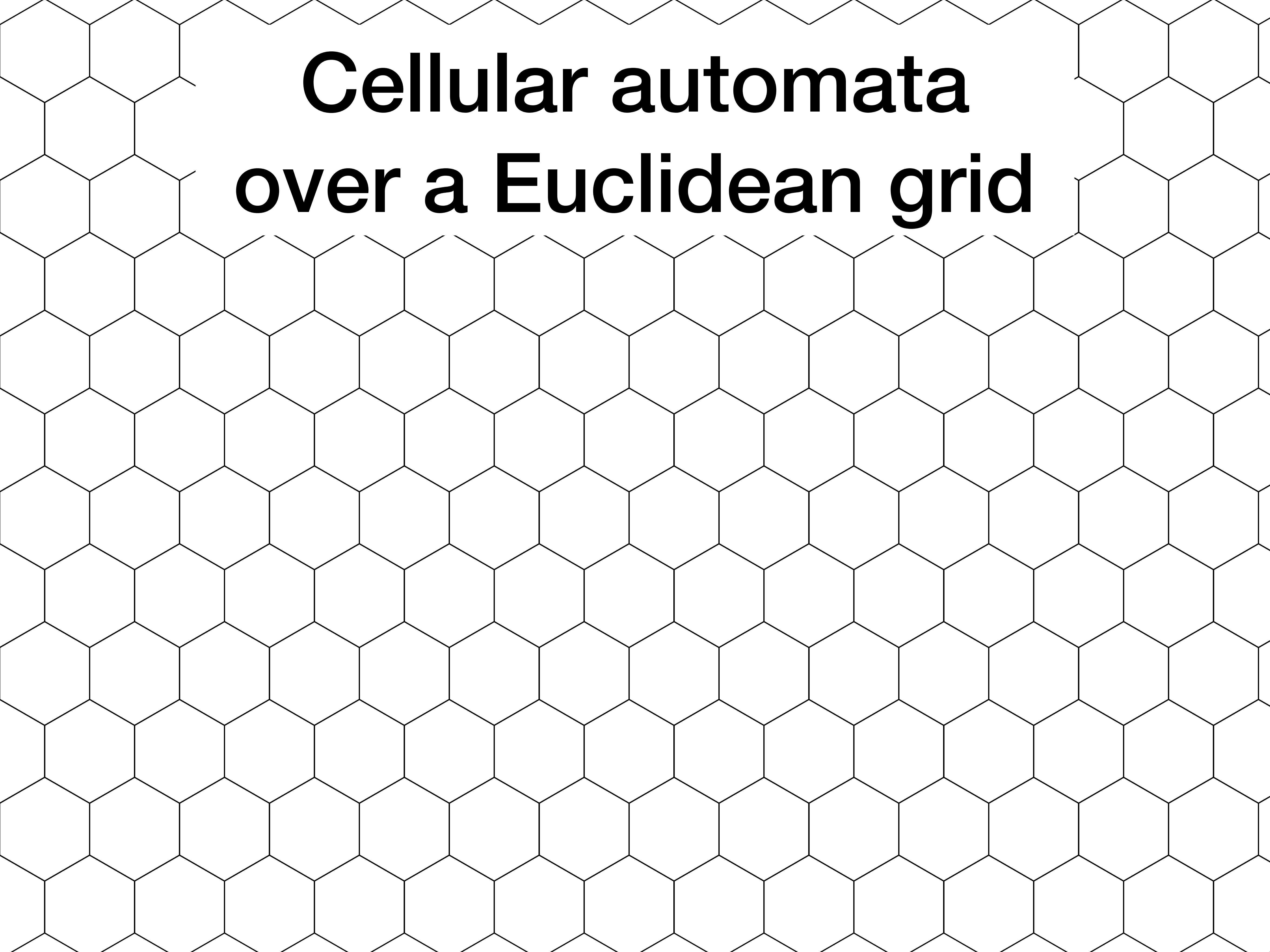


# Flattening ∧∨-circuits?

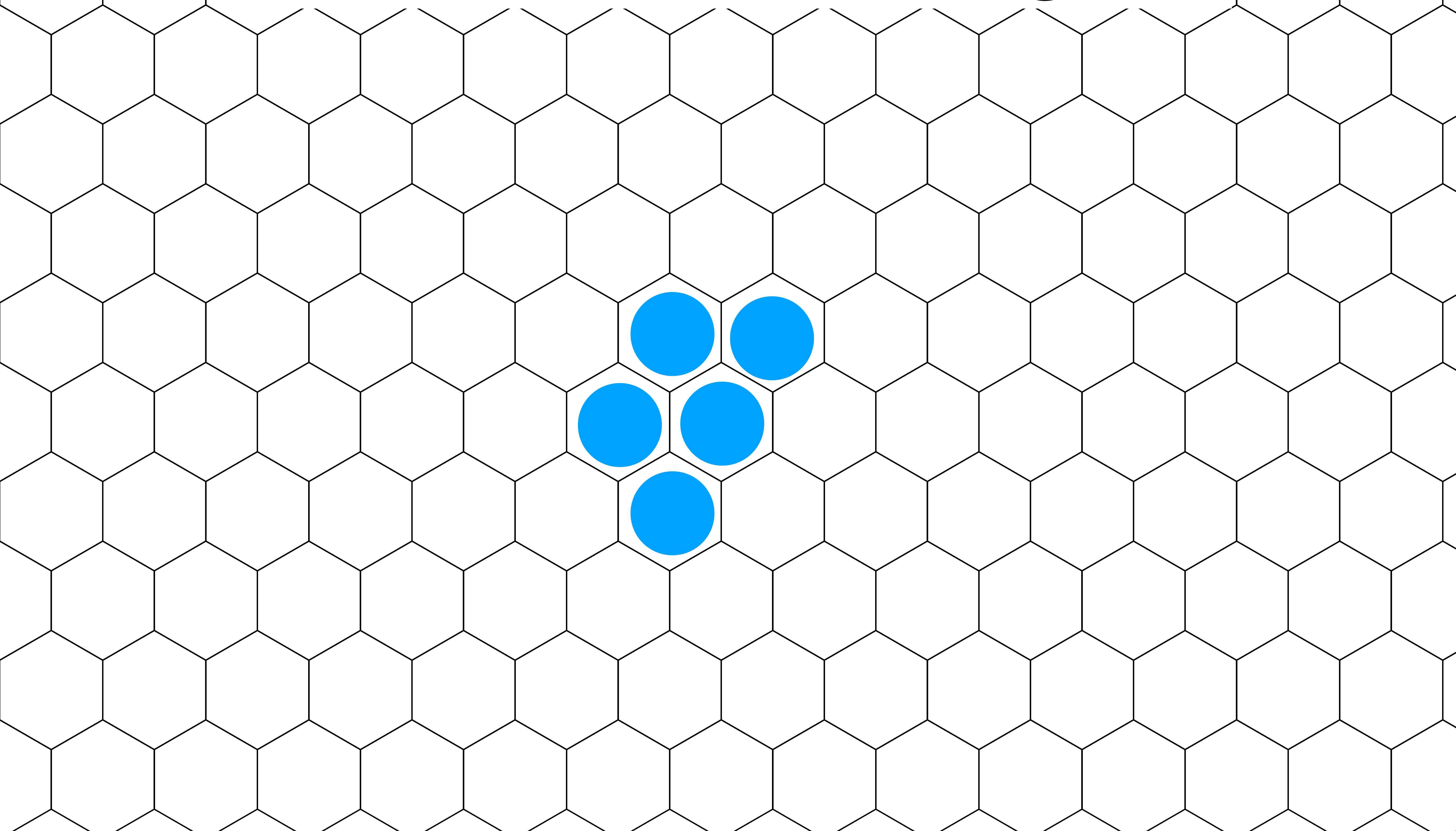


# Computation space
# vs computation efficiency

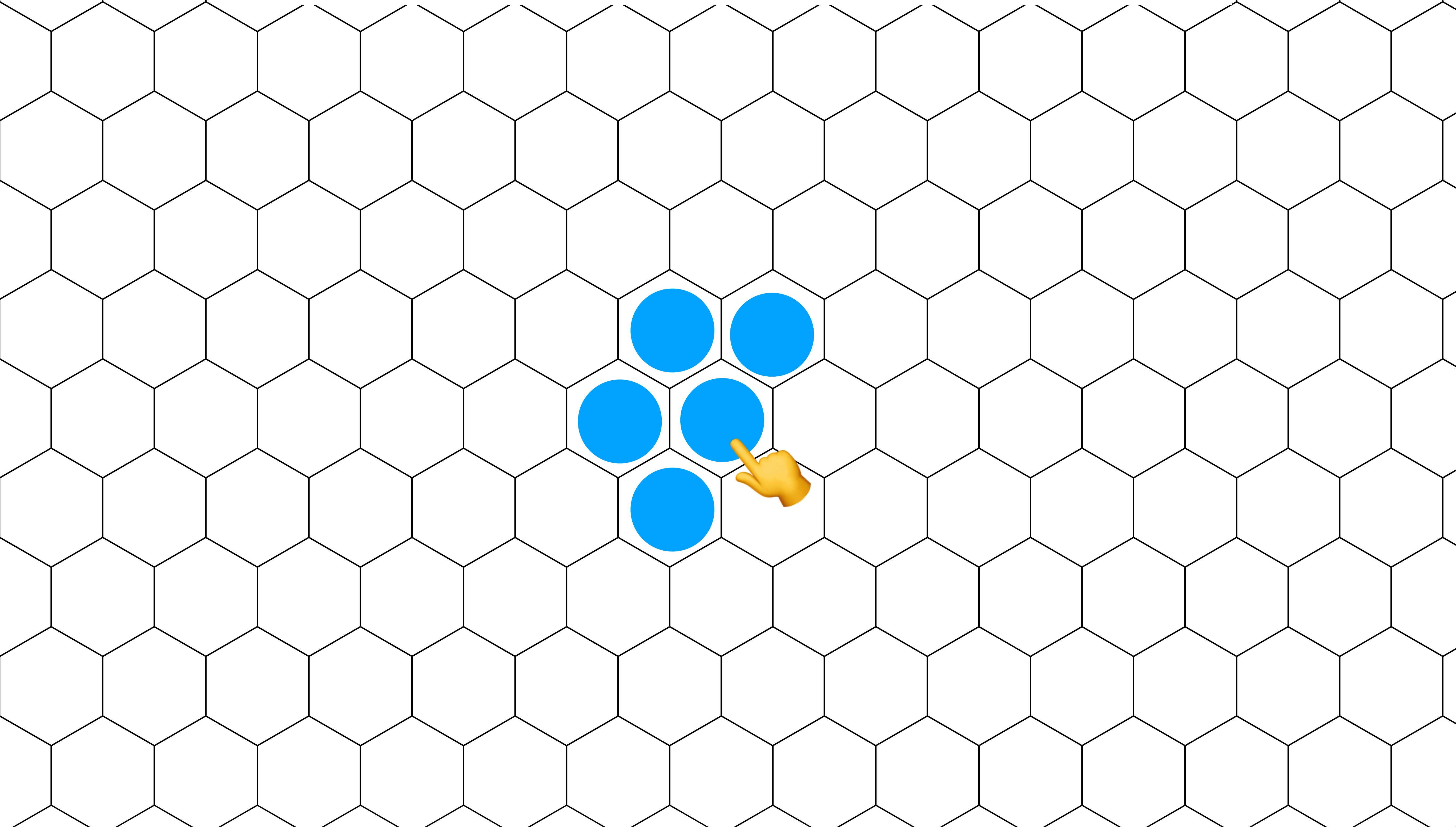
# Cellular automata over a Euclidean grid



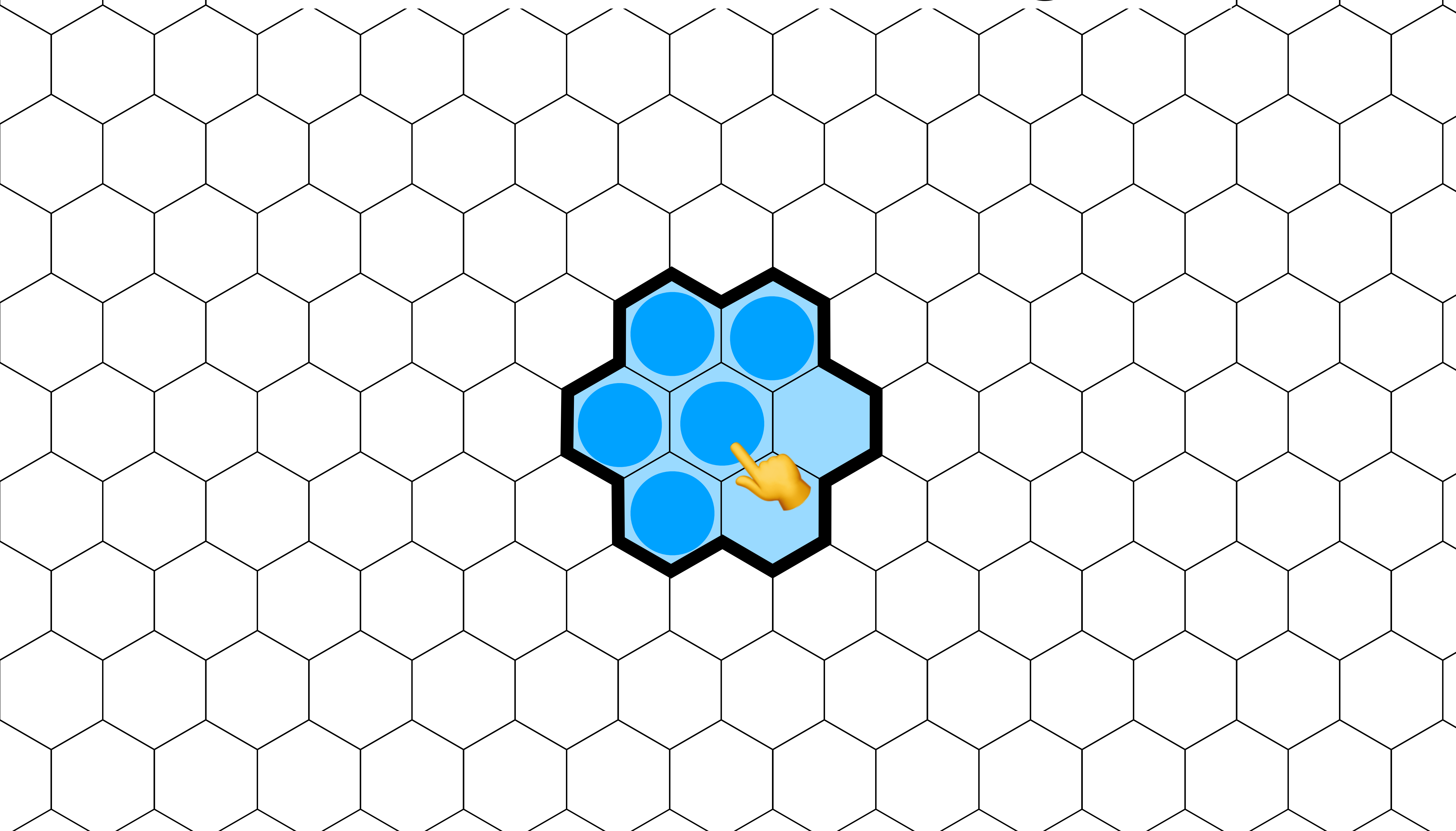
# Cellular automata over a Euclidean grid



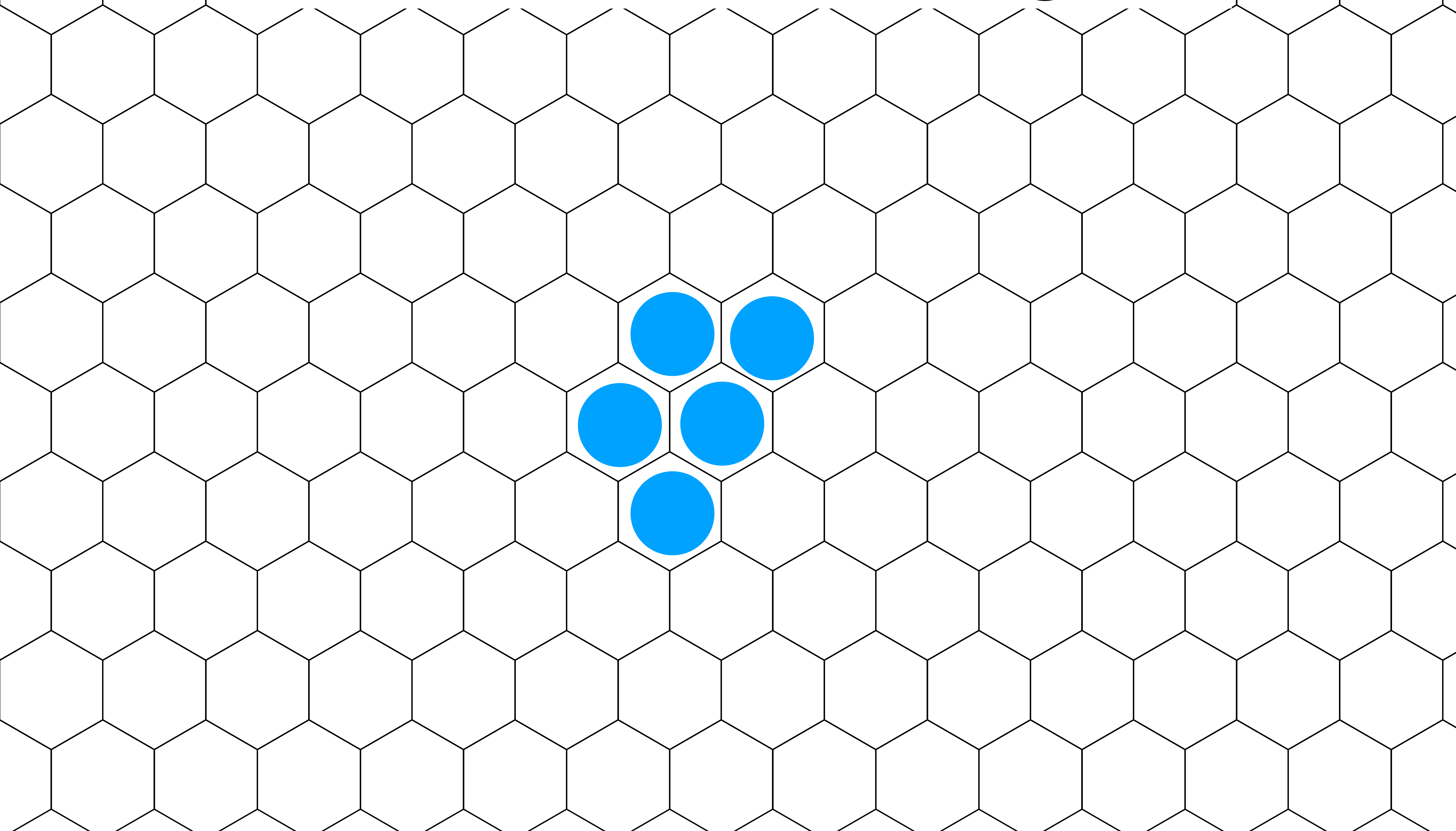
# Cellular automata over a Euclidean grid



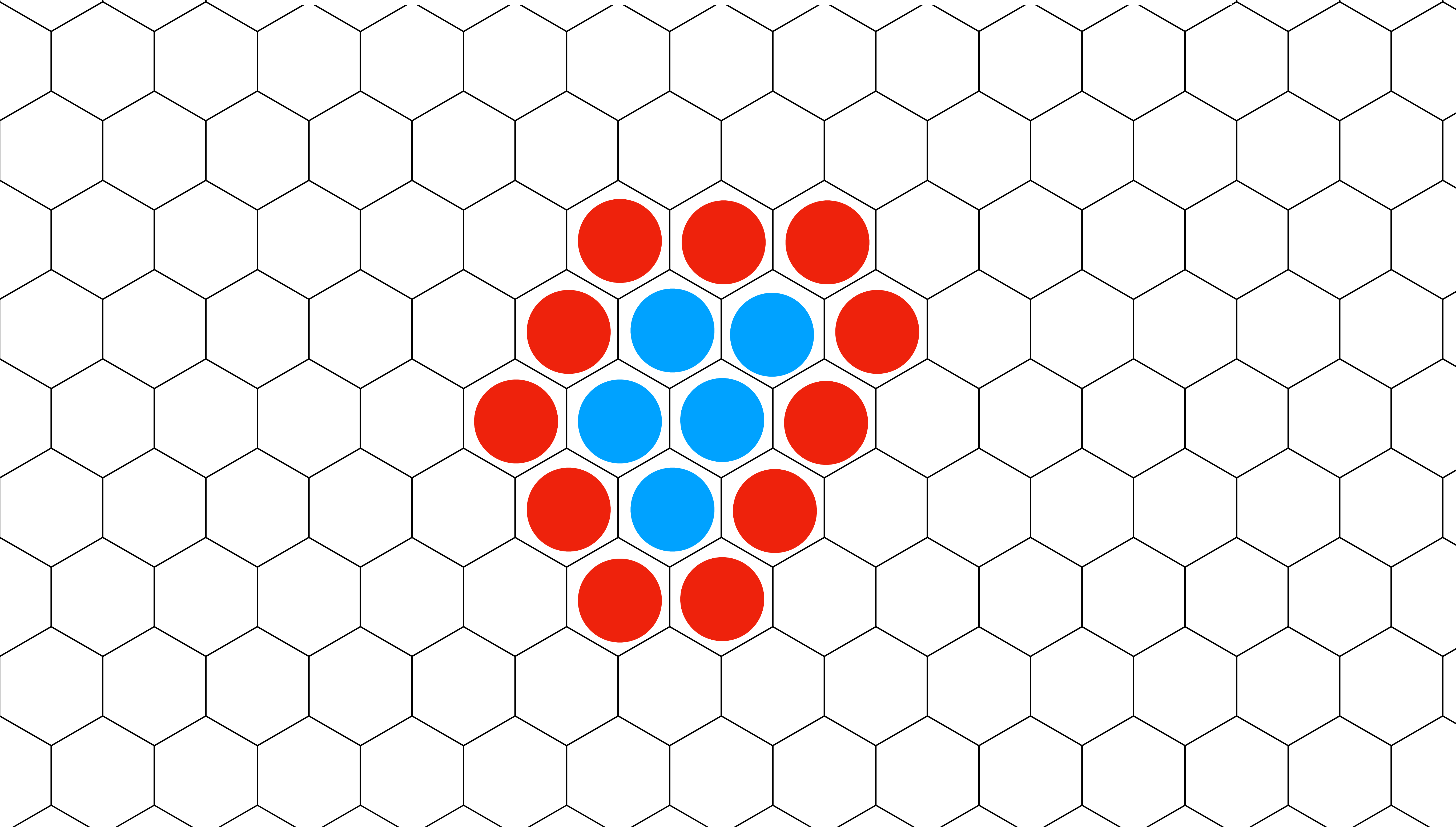
# Cellular automata over a Euclidean grid



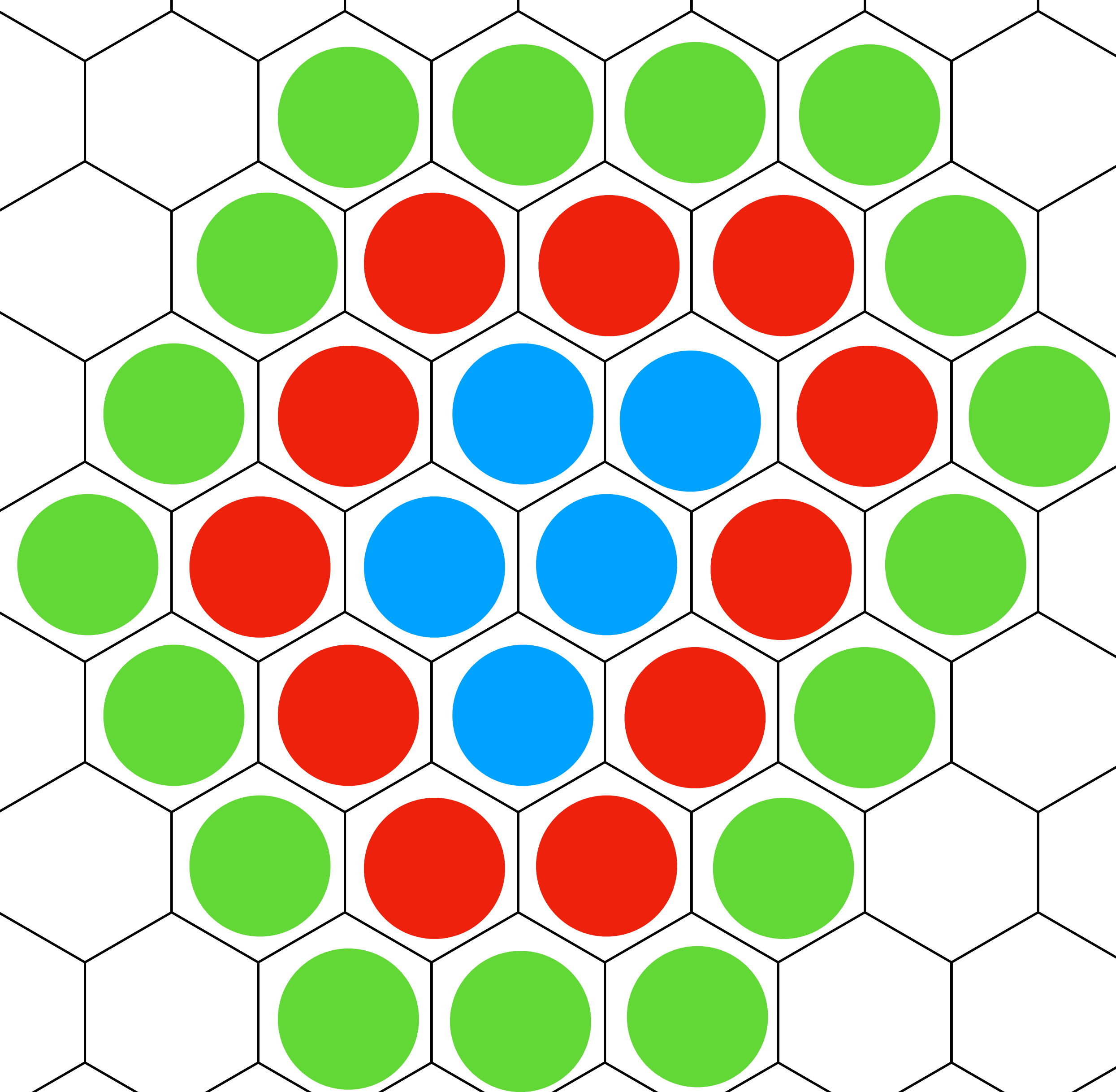
# Cellular automata over a Euclidean grid



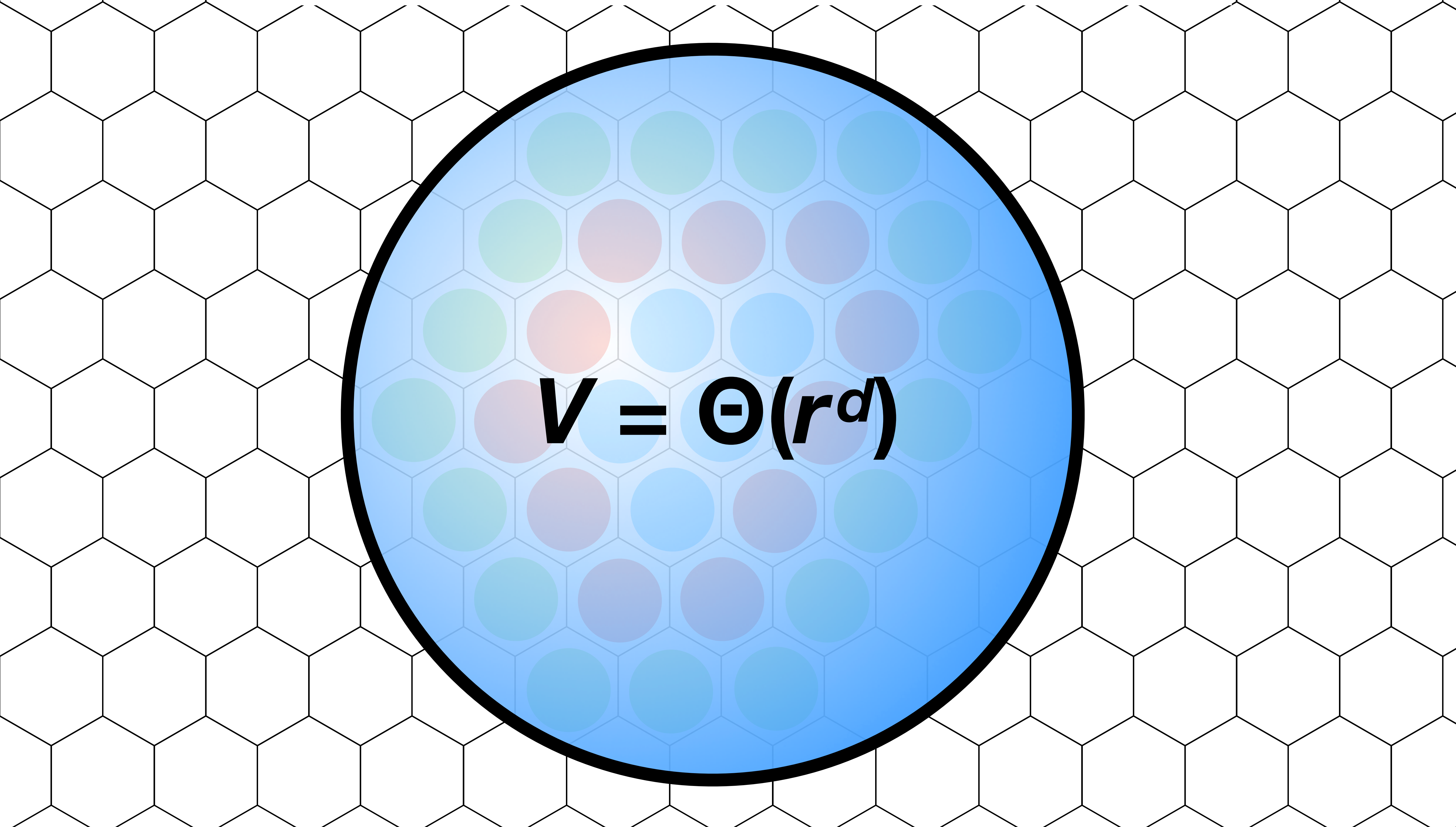
# Cellular automata over a Euclidean grid



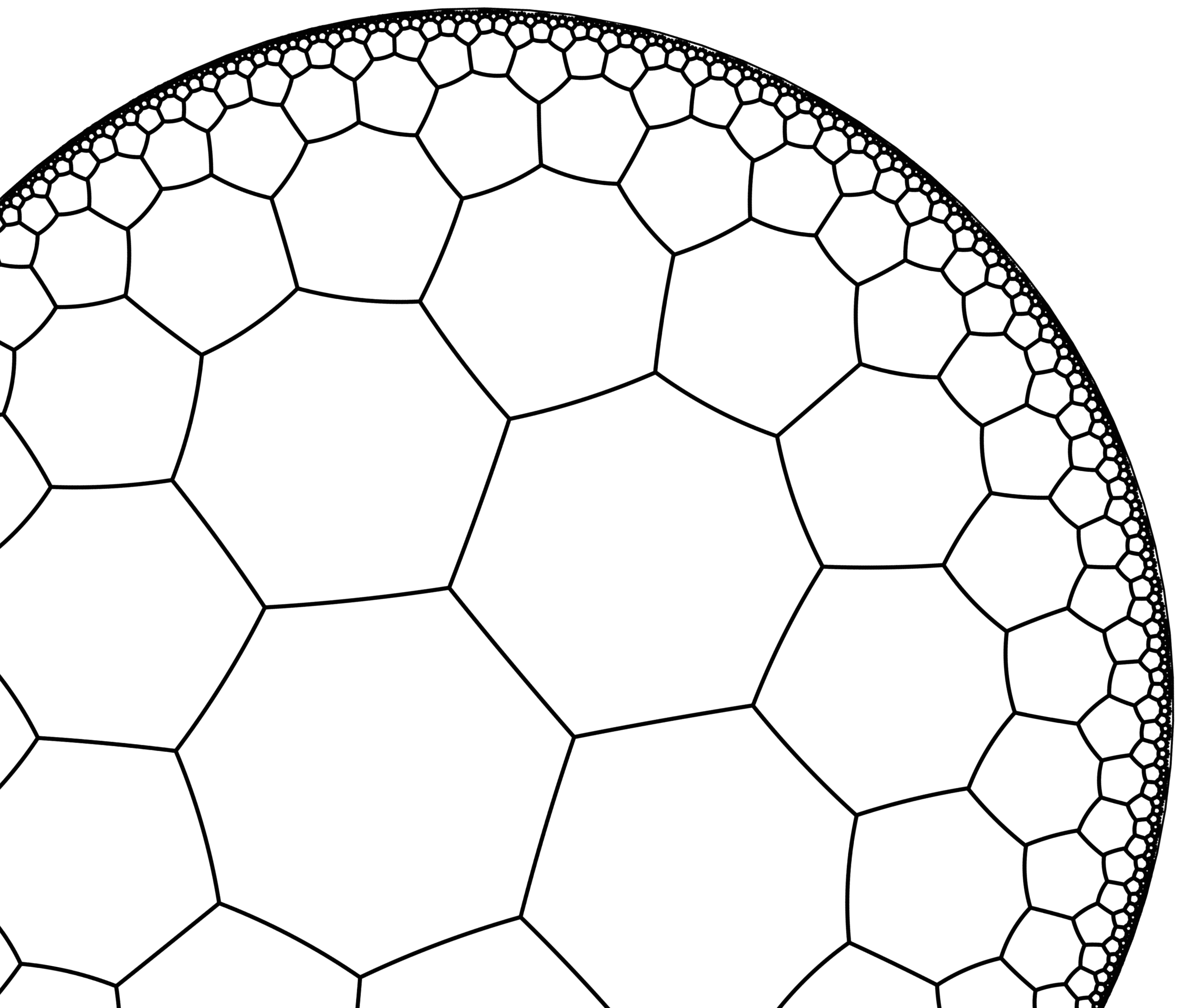
# Cellular automata over a Euclidean grid



# Cellular automata over a Euclidean grid

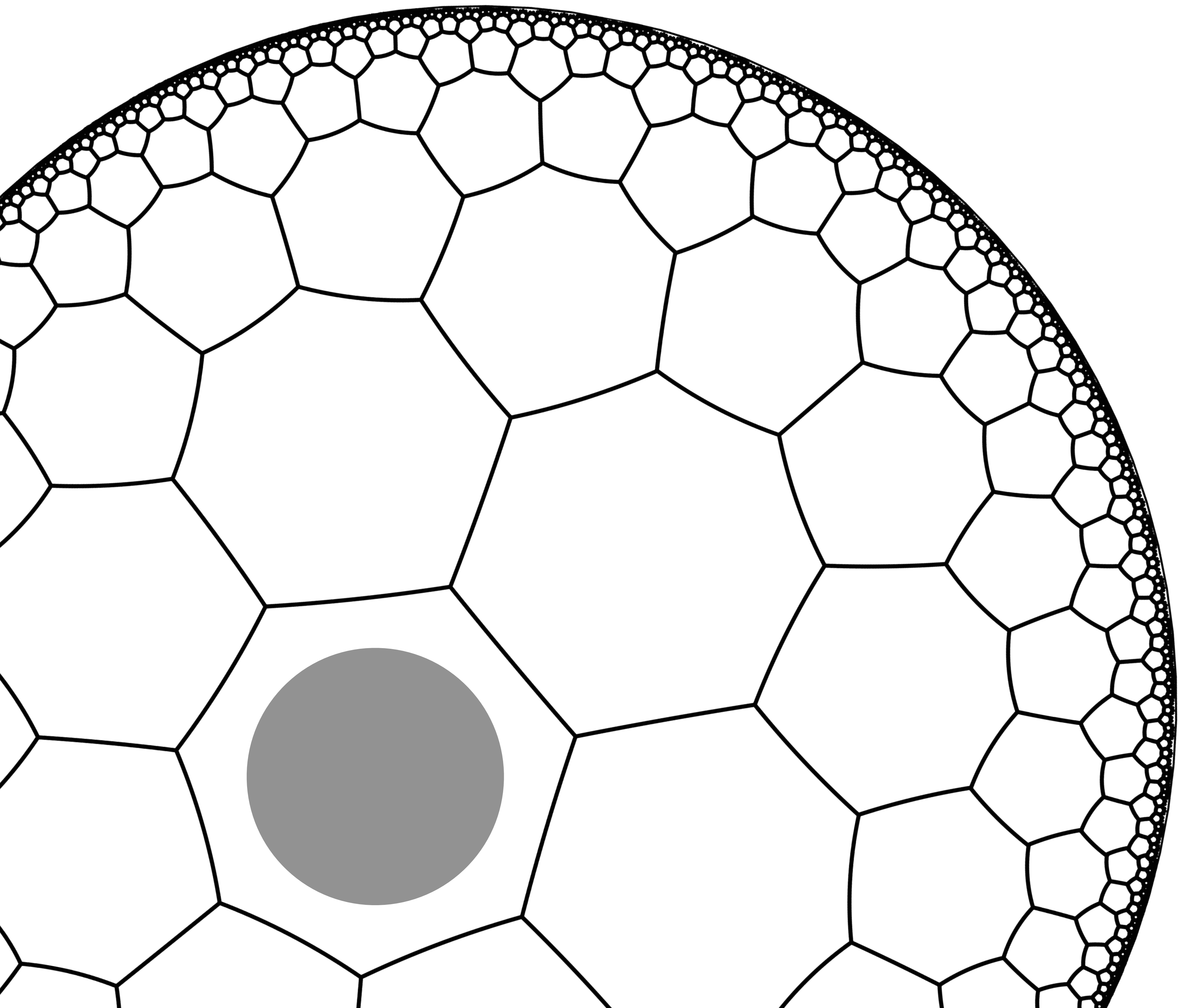


# Cellular automata over a hyperbolic grid



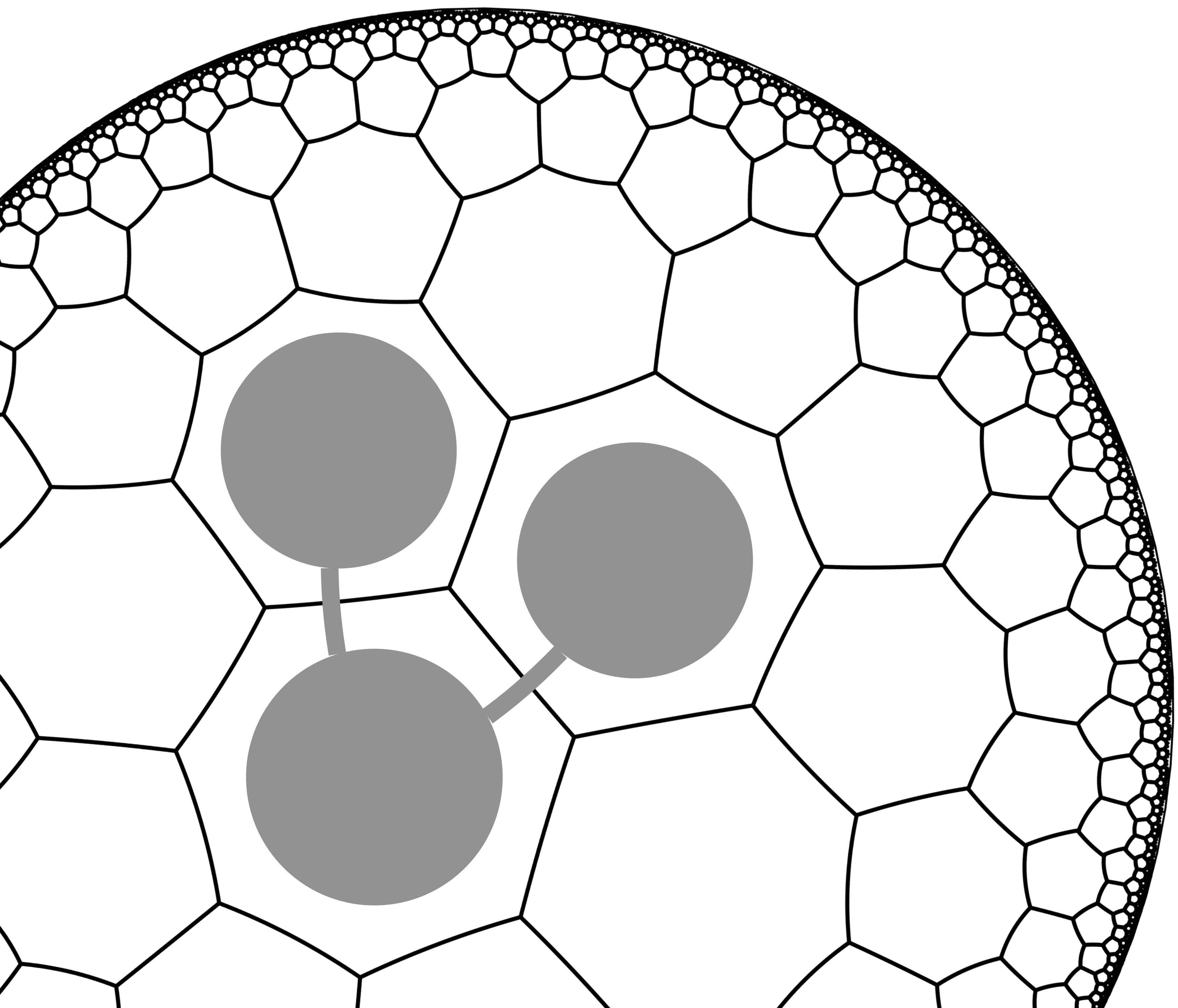
Pavage du plan hyperbolique par des heptagones, dans le modèle du disque de Poincaré. By Theon, used under CC BY-SA 3.0 https:/en.wikipedia.org/wiki/File:PavageHypPoincare2.svg

# Cellular automata over a hyperbolic grid



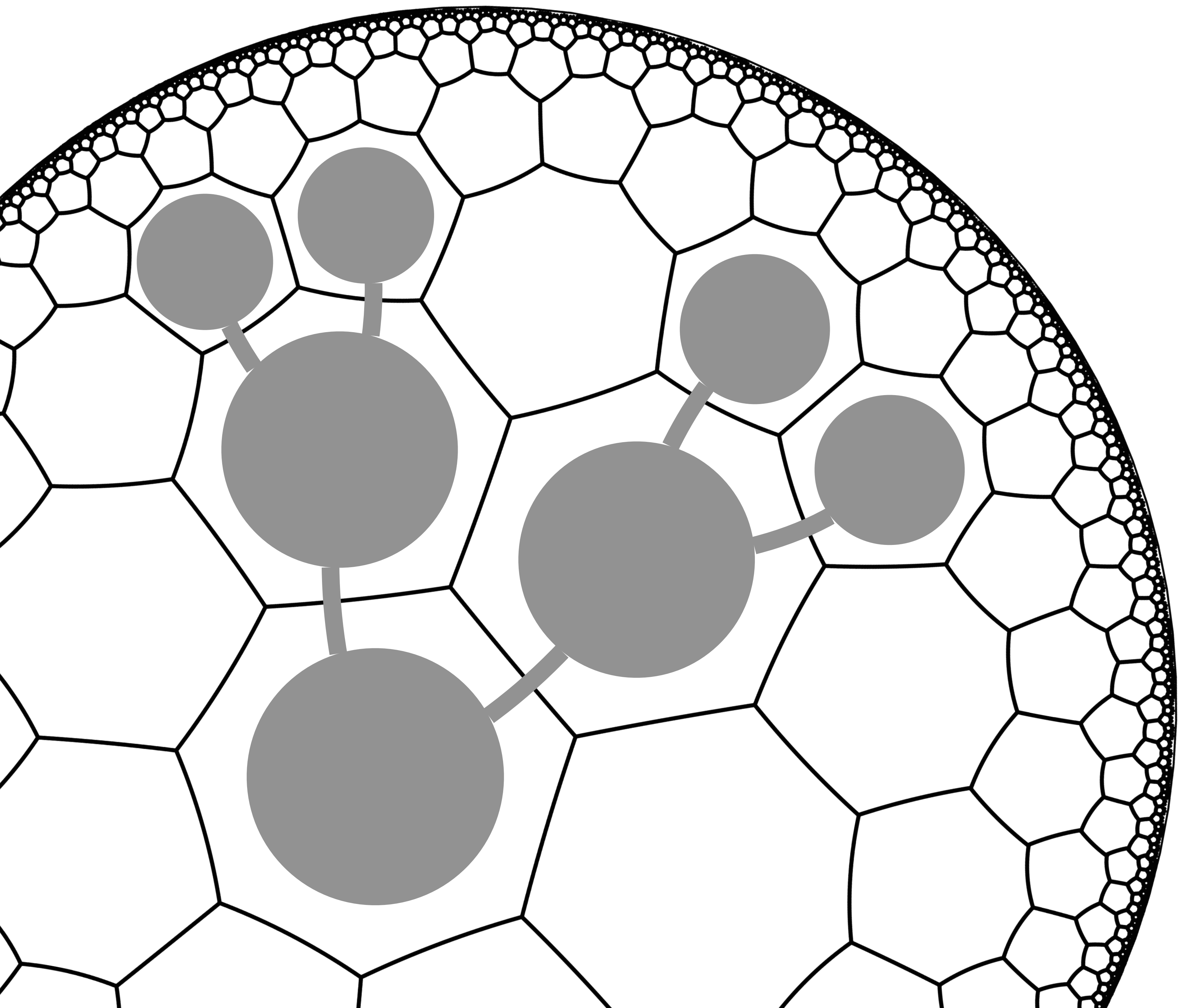
Pavage du plan hyperbolique par des heptagones, dans le modèle du disque de Poincaré. By Theon, used under CC BY-SA 3.0 https:/en.wikipedia.org/wiki/File:PavageHypPoincare2.svg

# Cellular automata over a hyperbolic grid



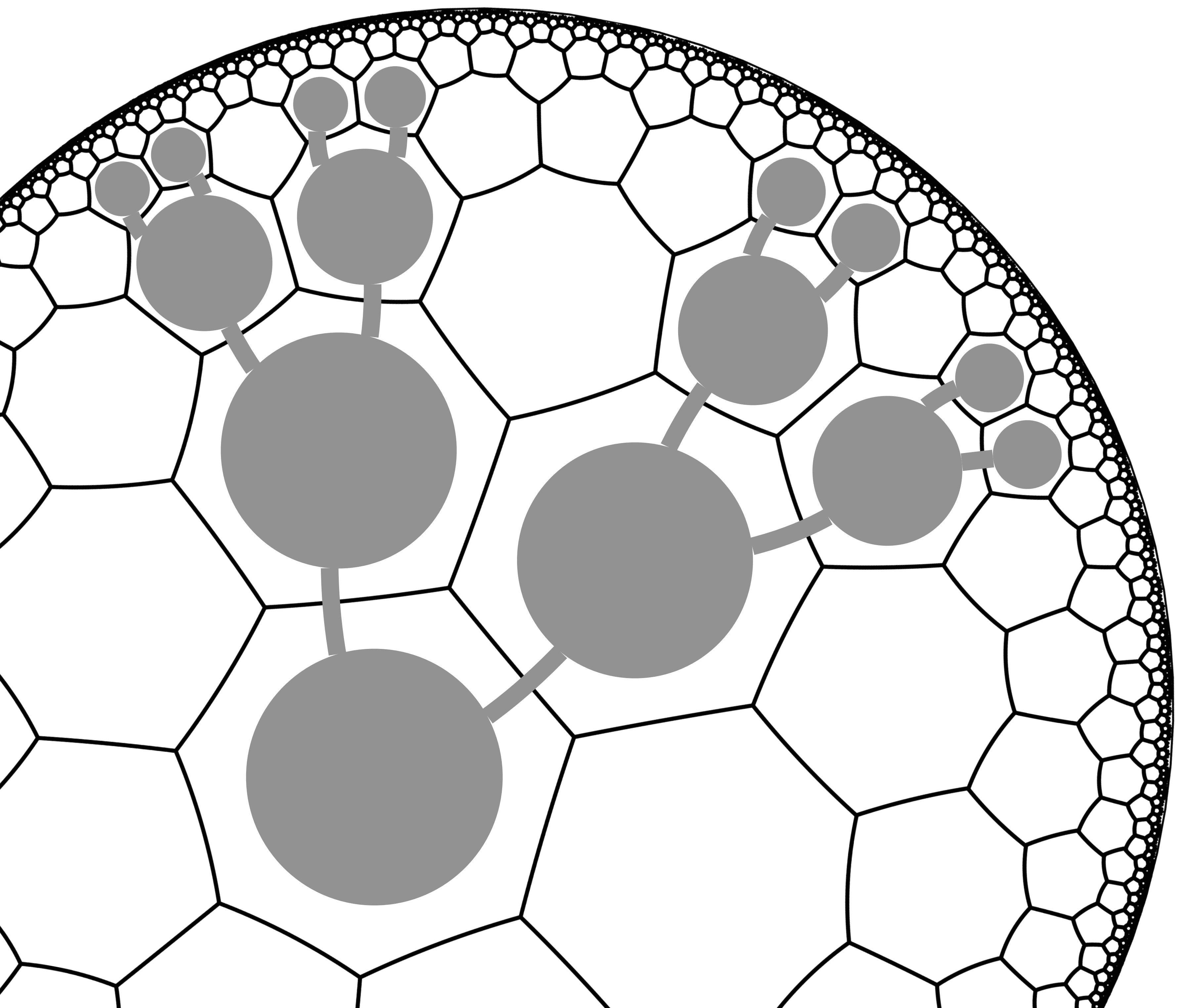
Pavage du plan hyperbolique par des heptagones, dans le modèle du disque de Poincaré. By Theon, used under CC BY-SA 3.0 https:/en.wikipedia.org/wiki/File:PavageHypPoincare2.svg

# Cellular automata over a hyperbolic grid



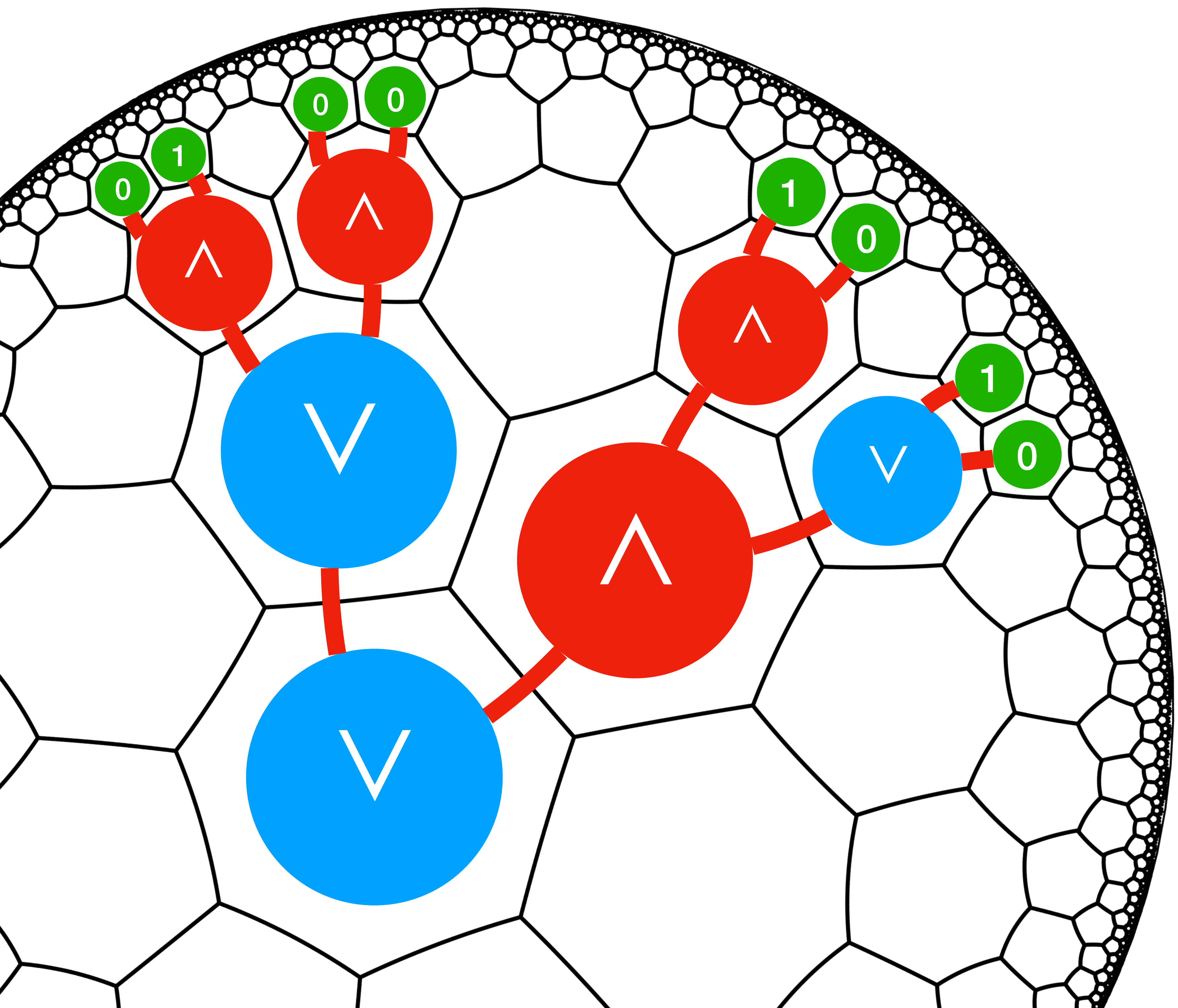
Pavage du plan hyperbolique par des heptagones, dans le modèle du disque de Poincaré. By Theon, used under CC BY-SA 3.0 https:/en.wikipedia.org/wiki/File:PavageHypPoincare2.svg

# Cellular automata over a hyperbolic grid



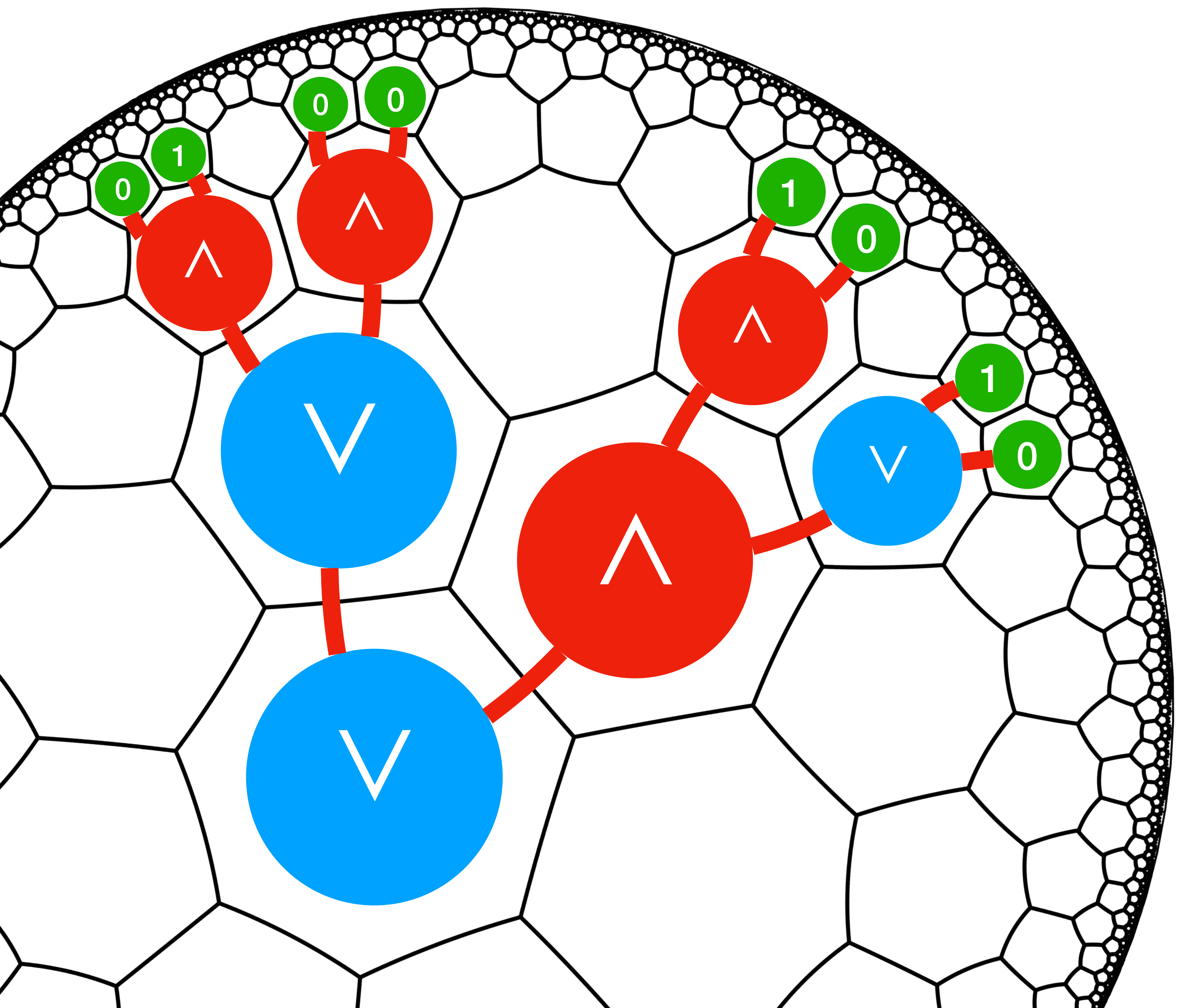
Pavage du plan hyperbolique par des heptagones, dans le modèle du disque de Poincaré. By Theon, used under CC BY-SA 3.0 https:/en.wikipedia.org/wiki/File:PavageHypPoincare2.svg

# Cellular automata over a hyperbolic grid




Pavage du plan hyperbolique par des heptagones, dans le modèle du disque de Poincaré. By Theon, used under CC BY-SA 3.0 https:/en.wikipedia.org/wiki/File:PavageHypPoincare2.svg

# Cellular automata over a hyperbolic grid



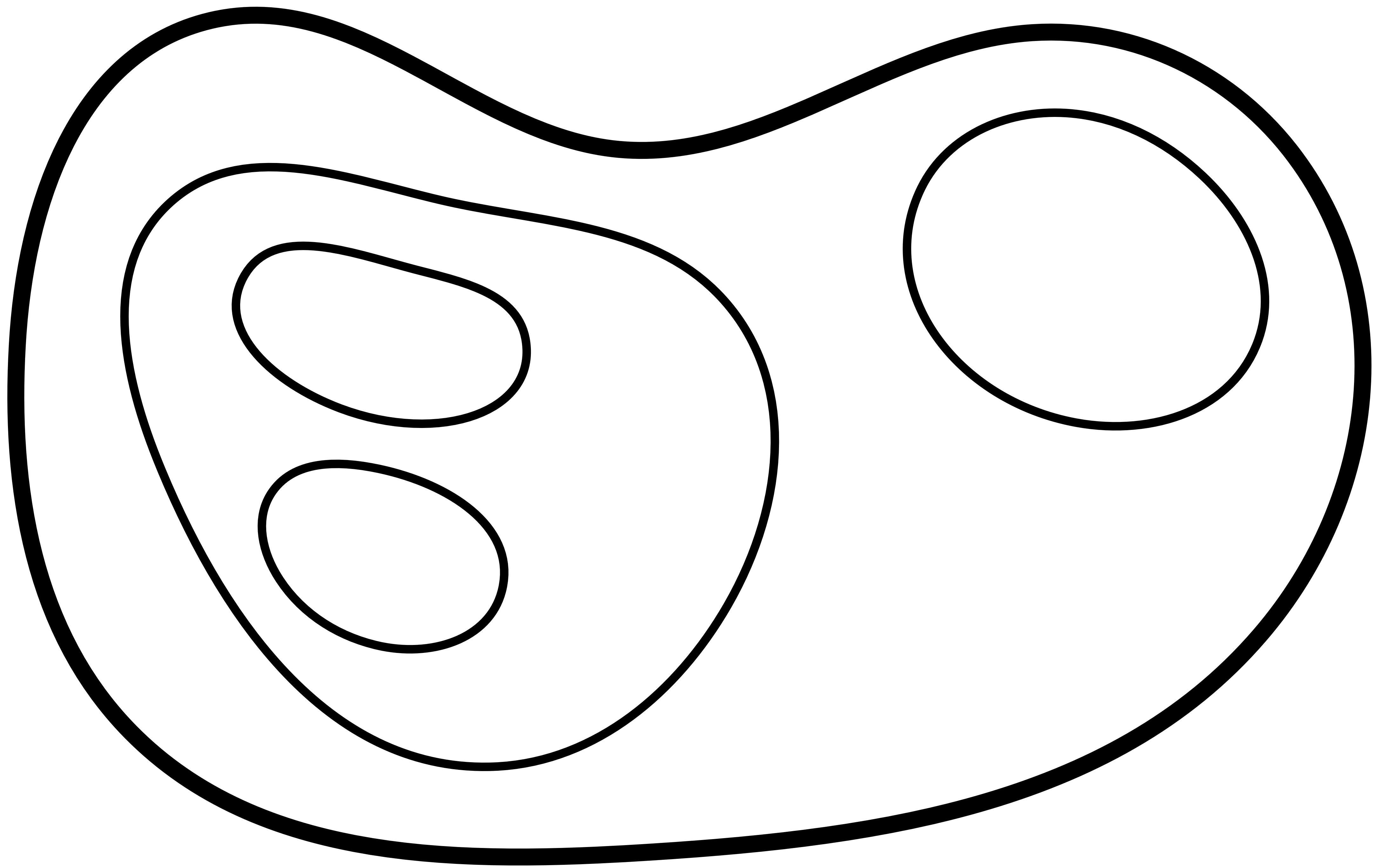

$$V = \Omega(2^r)$$

# Rule of thumb 👍

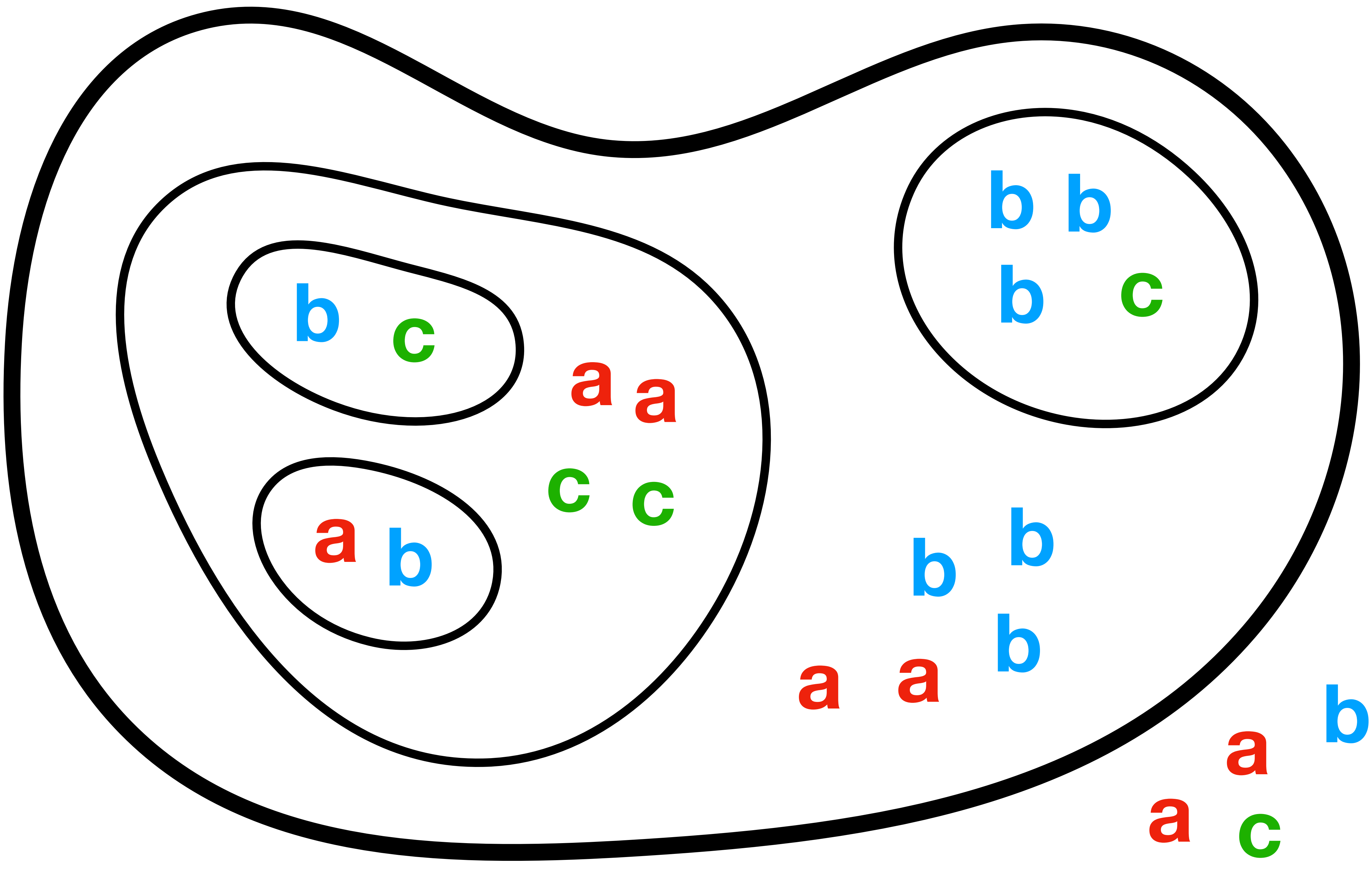
- Sequential machines are first class
- (Constant or polynomial) bounded parallel machines are also first class
- Unbounded (or exponential bounded) parallel machines are second class
- Apparently, this holds even for unconventional computing models ☹️

# A “more unconventional” model of computation: membrane systems

# Membrane systems



# Membrane systems



# Evolution rules

$[ab \rightarrow cd]$

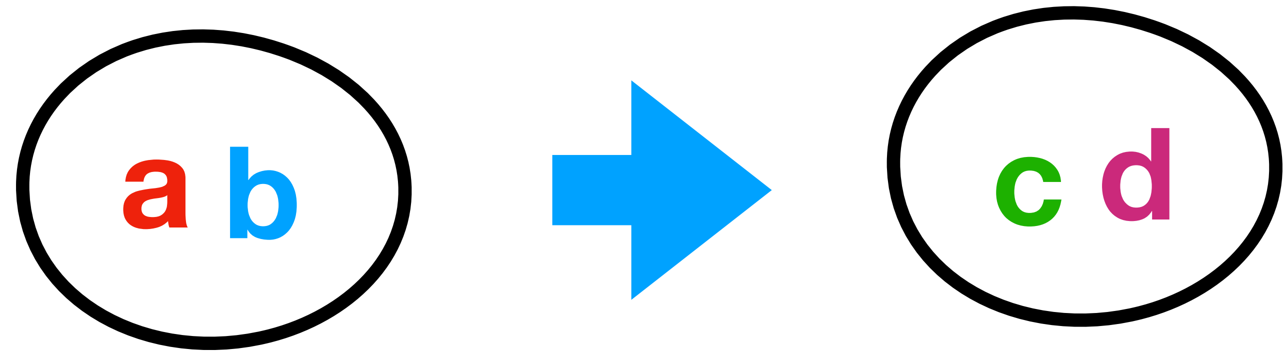
$[a] \rightarrow [\ ]\ b$

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

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

# Evolution rules

[ab → cd]




[a] → [ ] b

a [ ] → [b]

[a] → [b] [c]

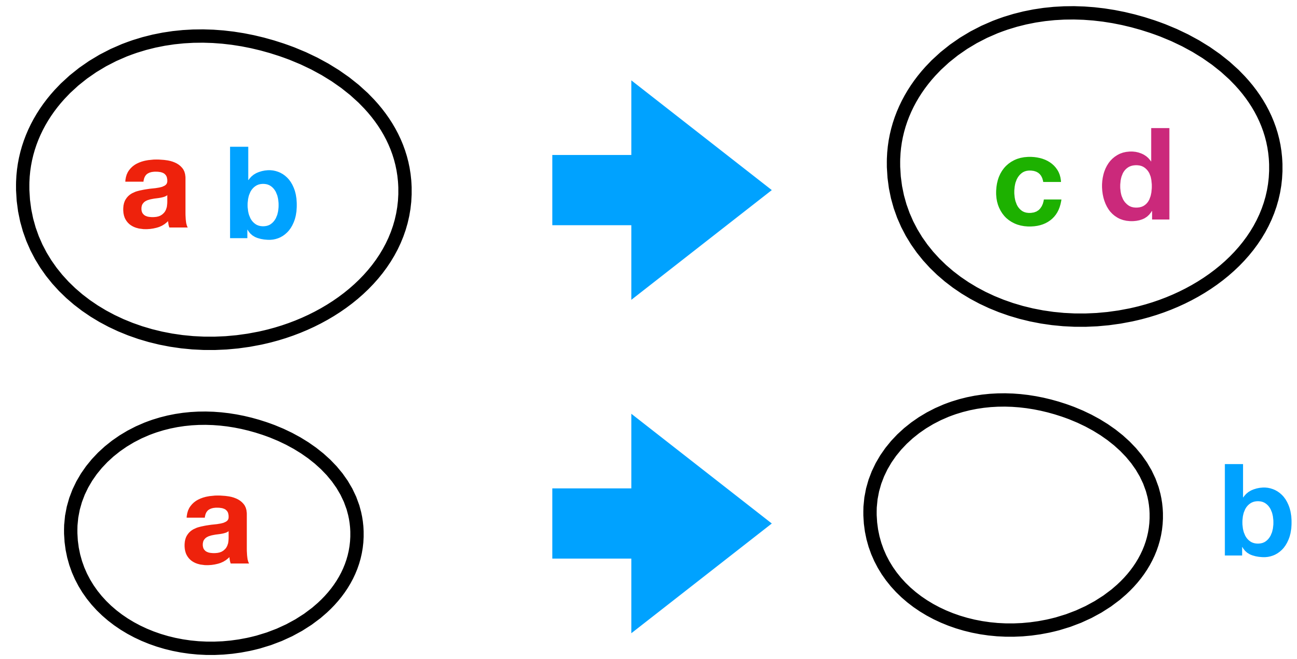
# Evolution rules

[ab → cd]

[a] → [ ] b

a [ ] → [b]

[a] → [b] [c]



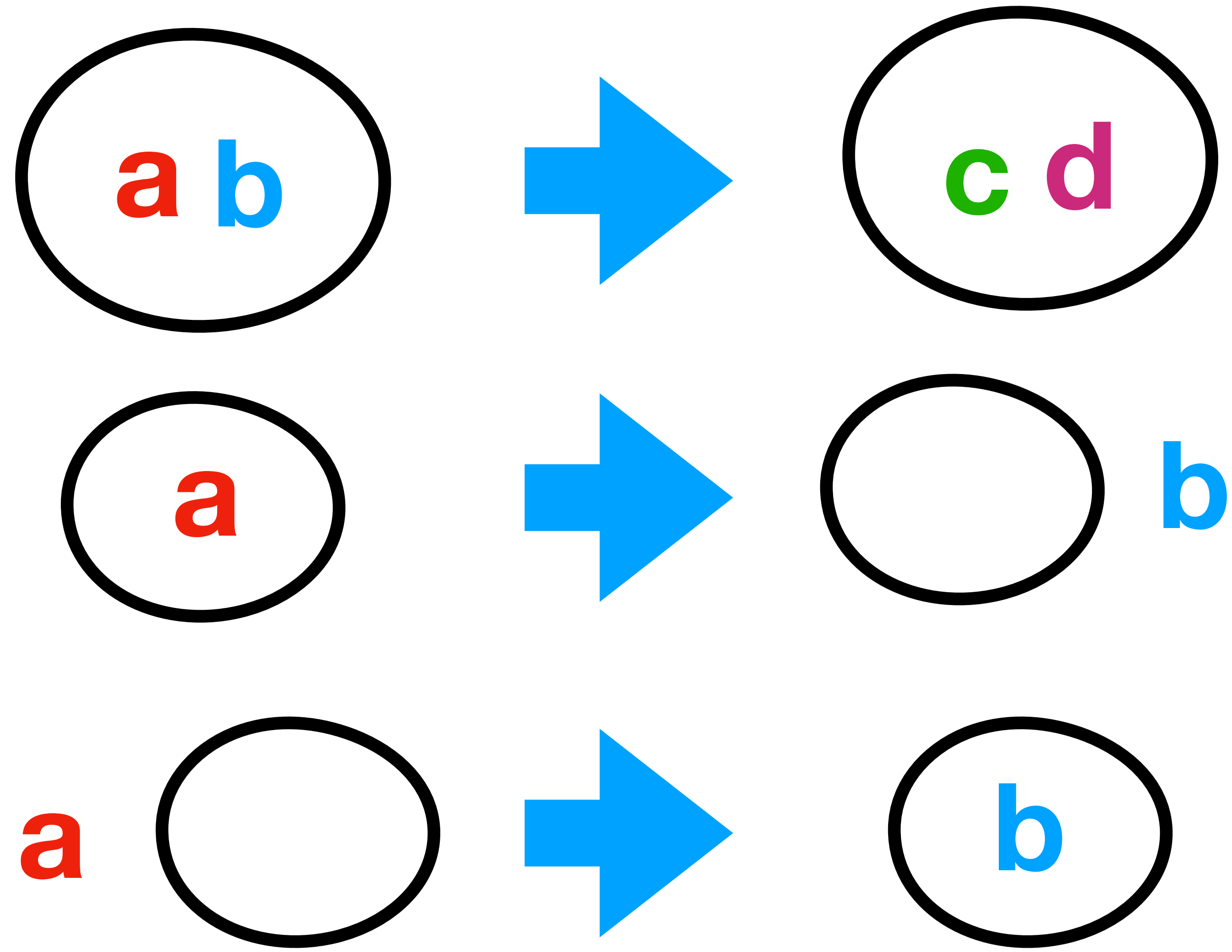
# Evolution rules

[ab → cd]

[a] → [ ] b

a [ ] → [b]

[a] → [b] [c]



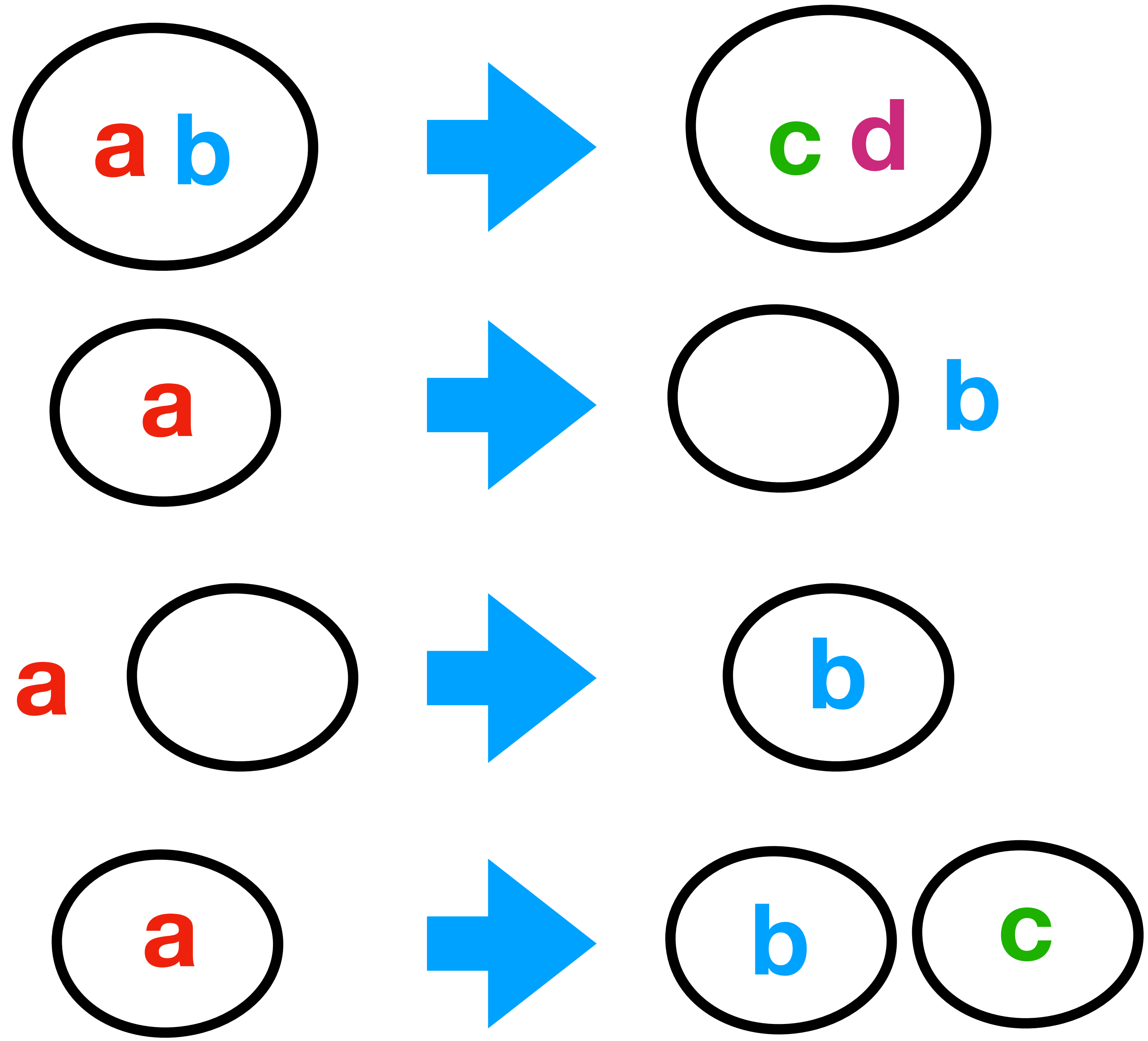
# Evolution rules

[ab → cd]

[a] → [ ] b

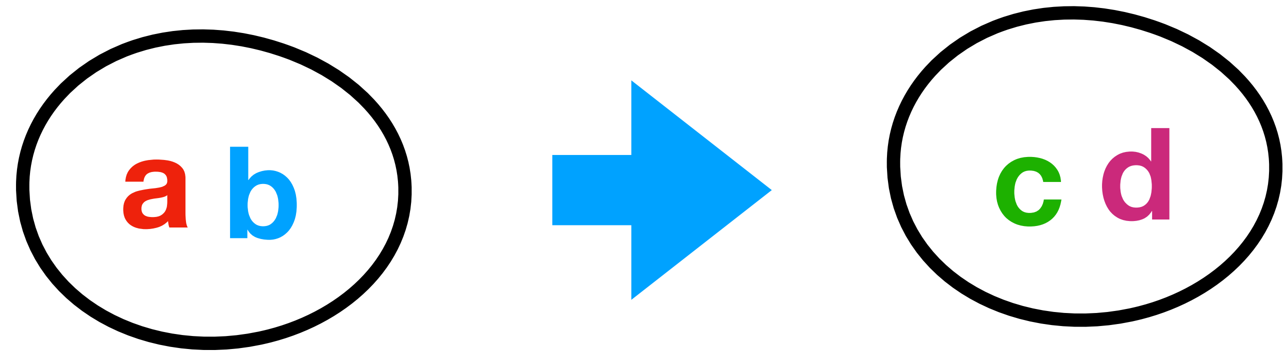
a [ ] → [b]

[a] → [b] [c]

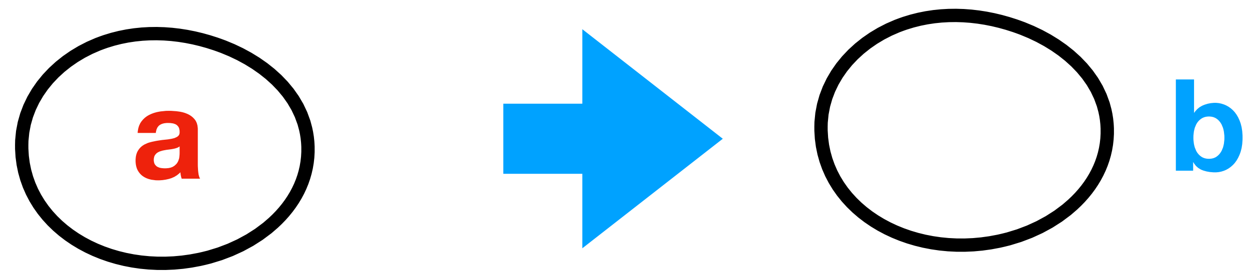


# Evolution rules

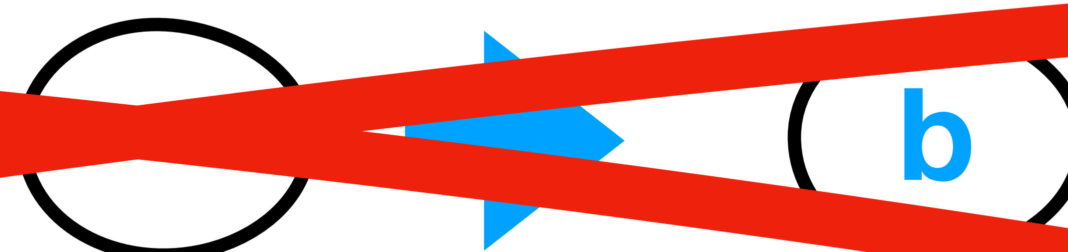
[ab → cd]




[a] → [ ] b

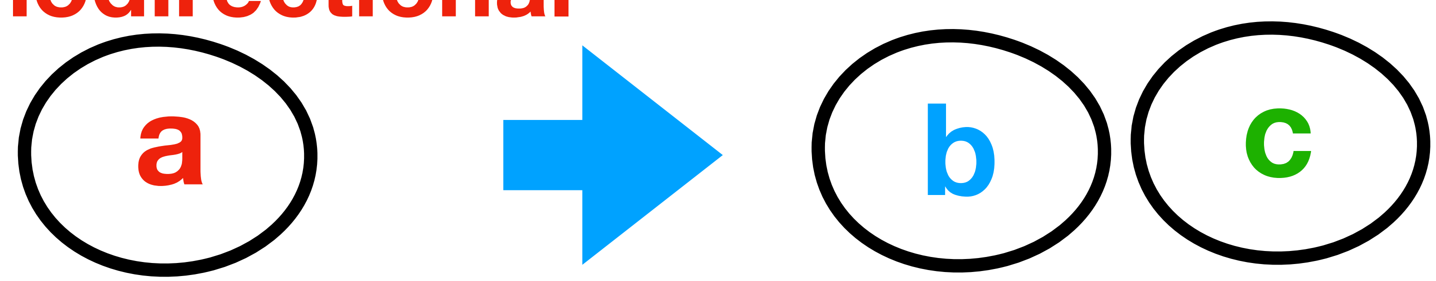



~~a [ ] → [b]~~



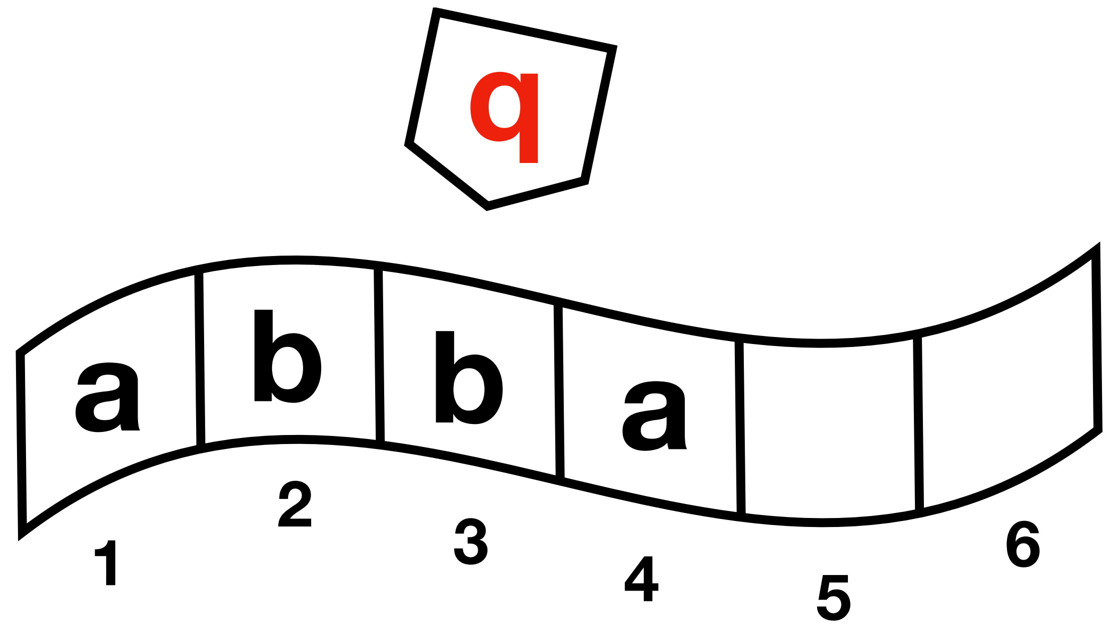

“monodirectional”

[a] → [b] [c]

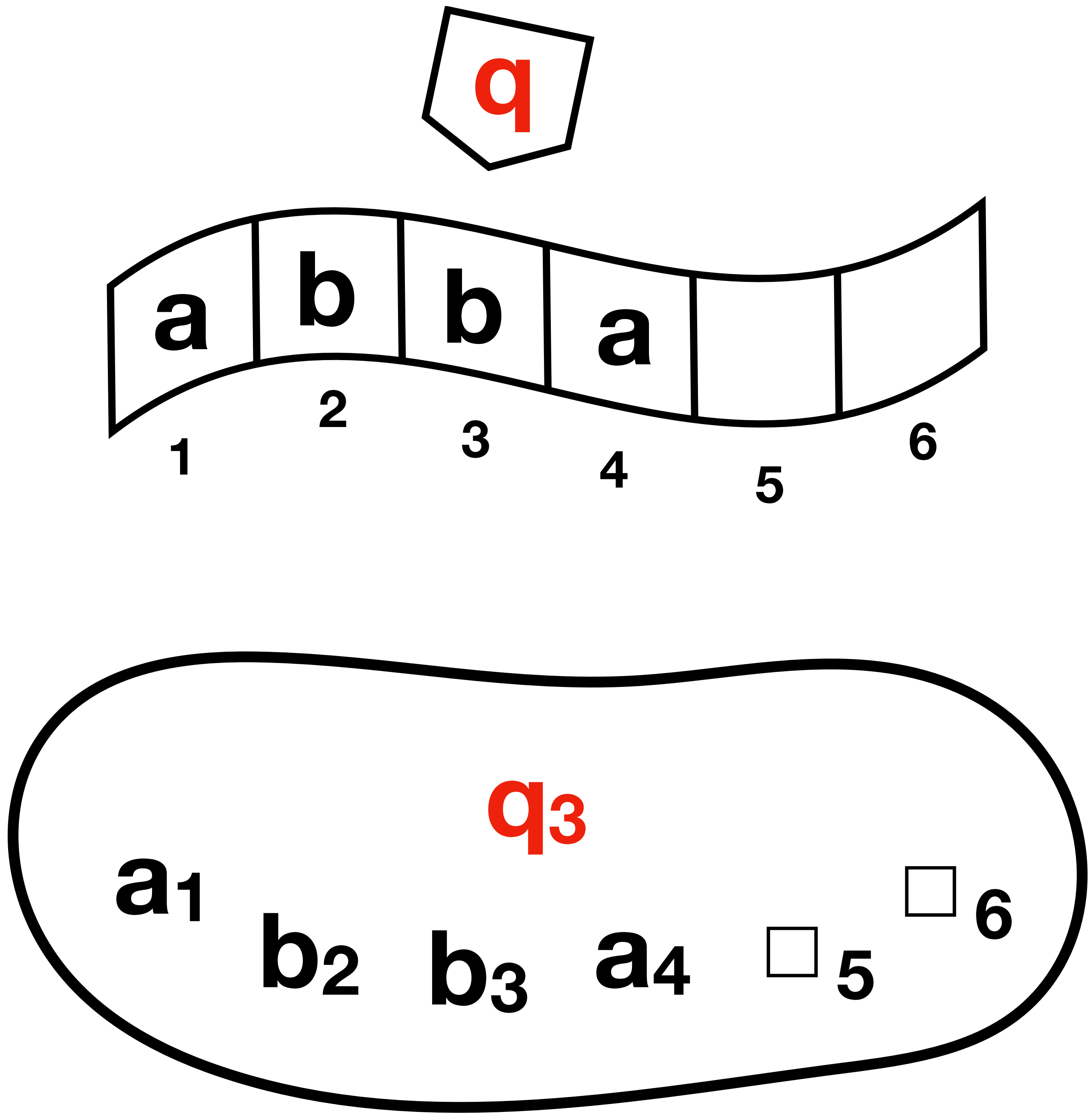


# Simulating Turing machines with membrane systems

# Encoding the configuration

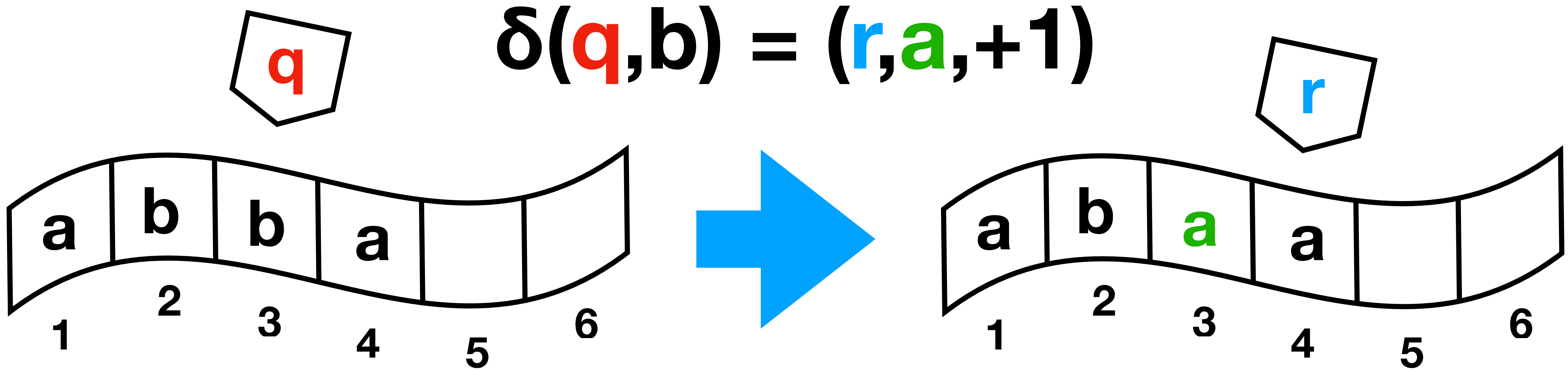


# Encoding the configuration



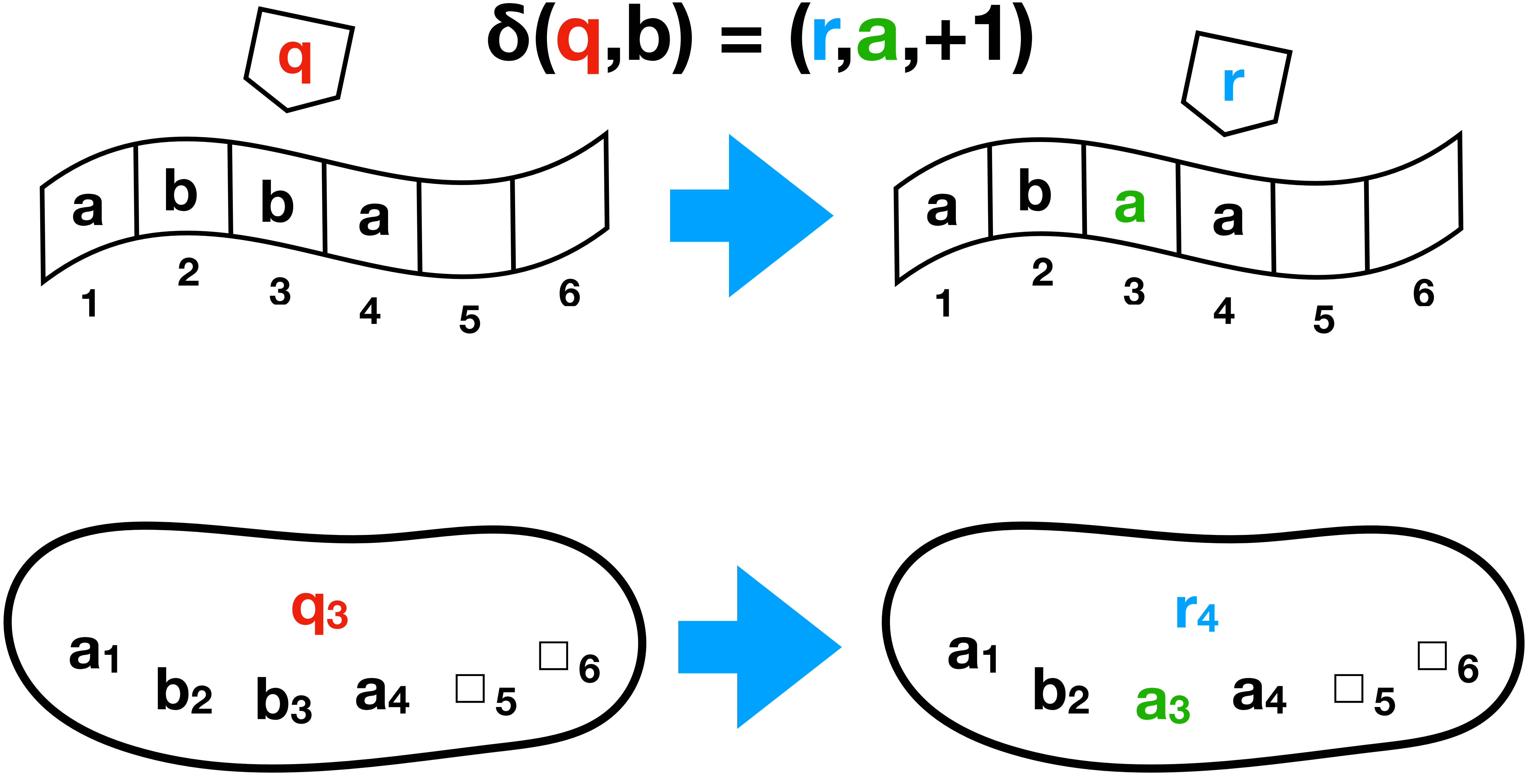
# Simulating transitions

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



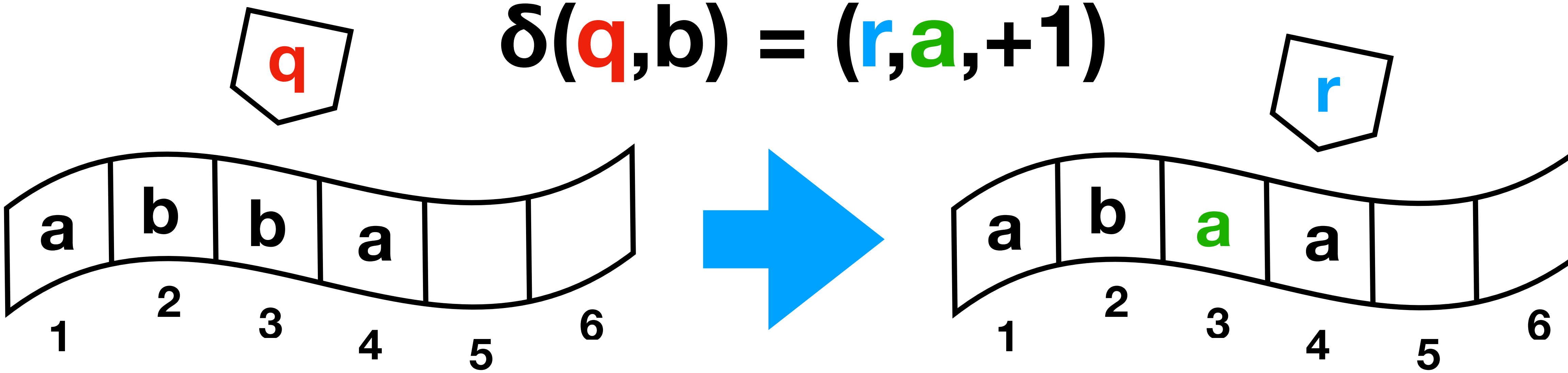
# Simulating transitions

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

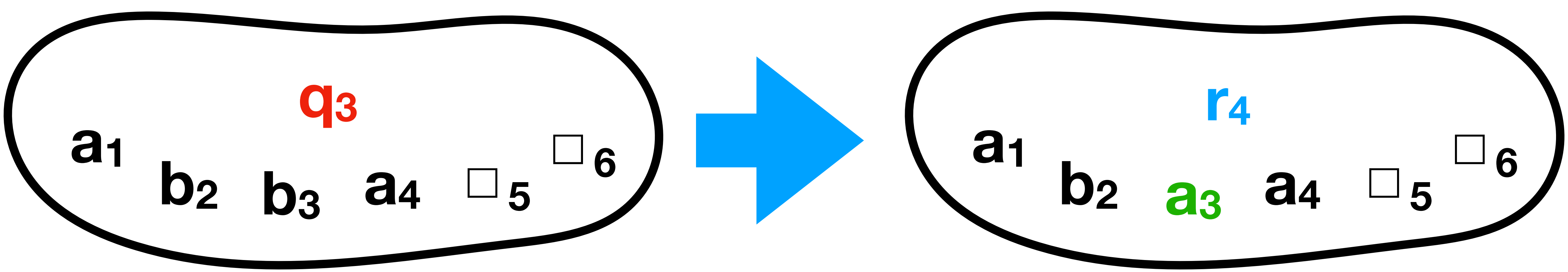


# Simulating transitions

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

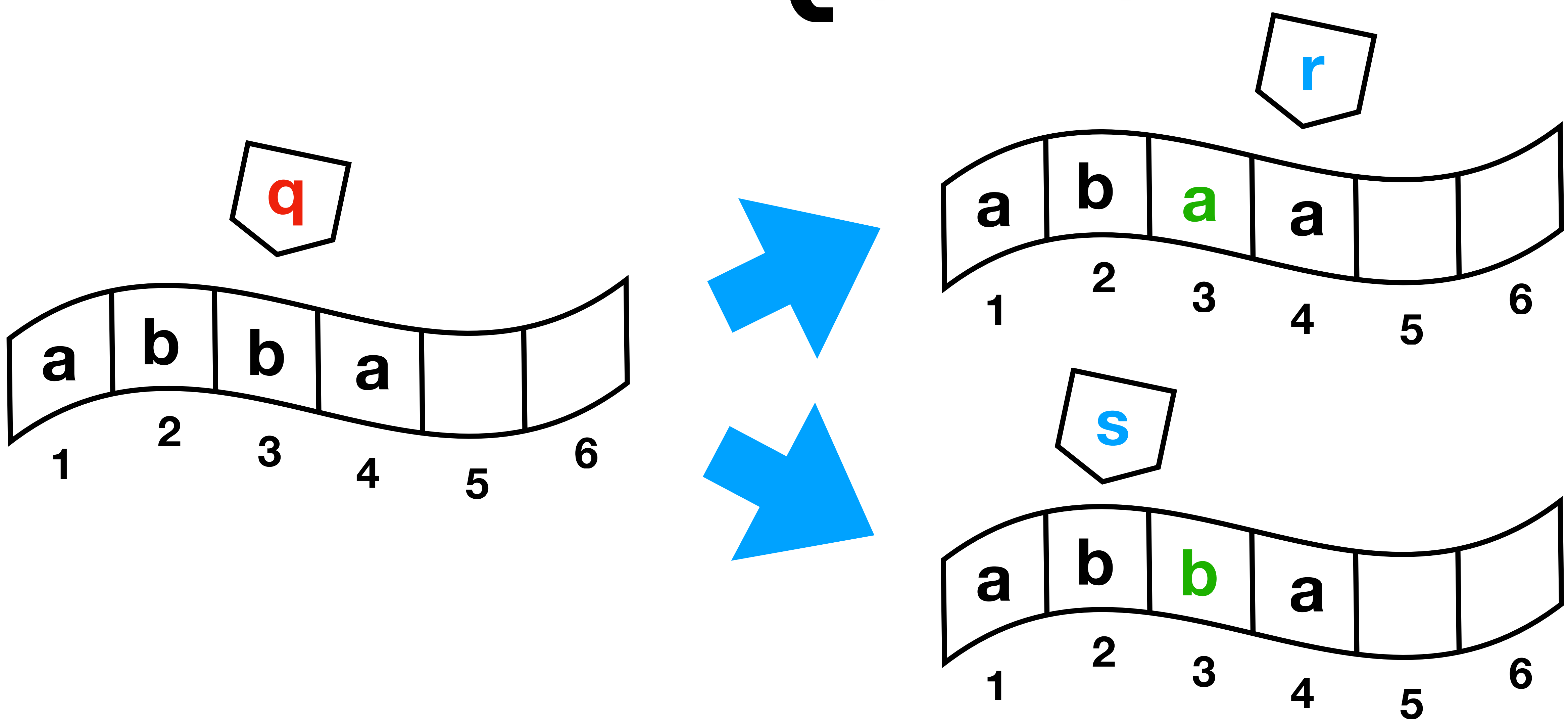



$$[q_3\ b_3 \rightarrow r_4\ a_3]$$



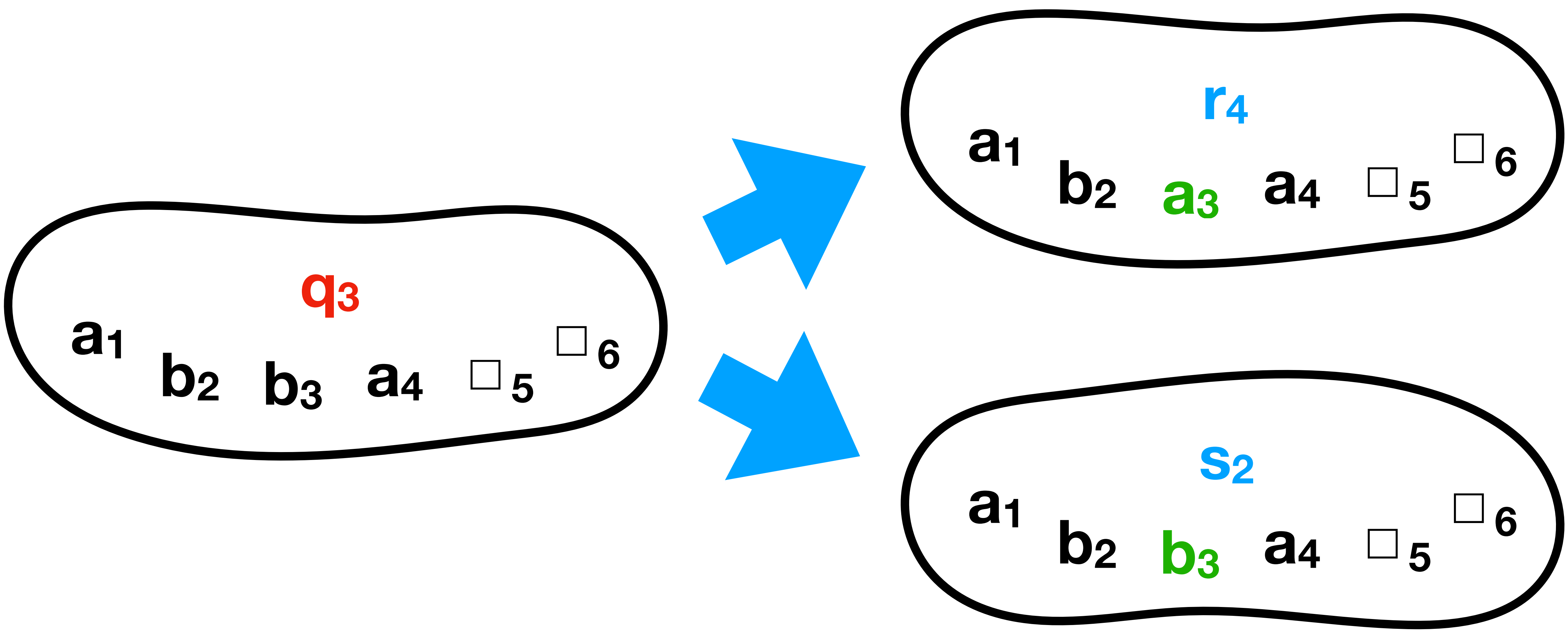
# Simulating nondeterminism

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



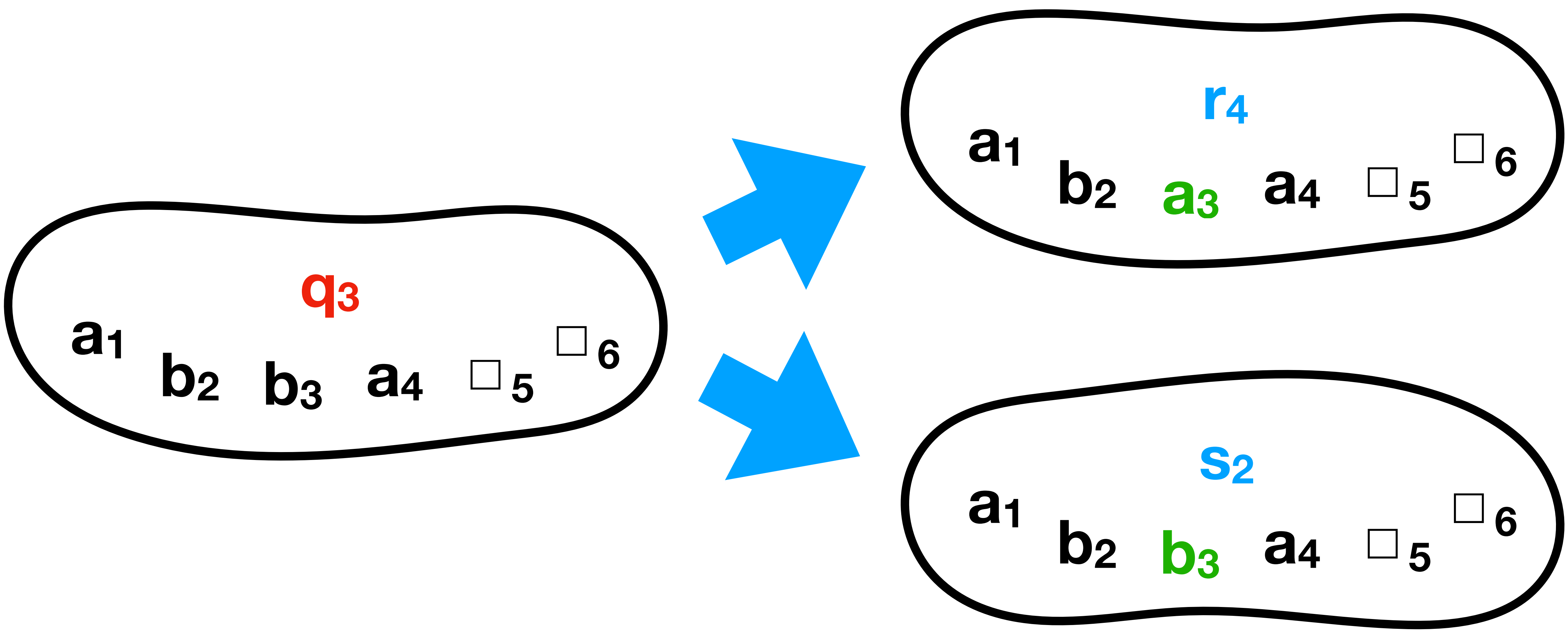
# Simulating nondeterminism

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

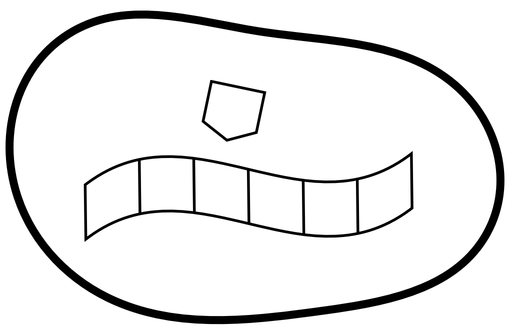


# Simulating nondeterminism

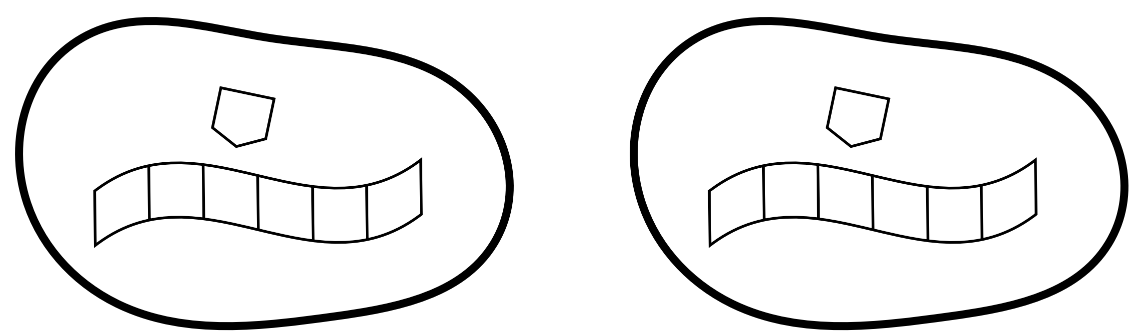
$$\delta(q,b) = \begin{cases} (r,a,+1) \\ (s,b,-1) \end{cases} \qquad [q_3 b_3] \rightarrow [r_4 a_3]\ [s_2 b_3]$$



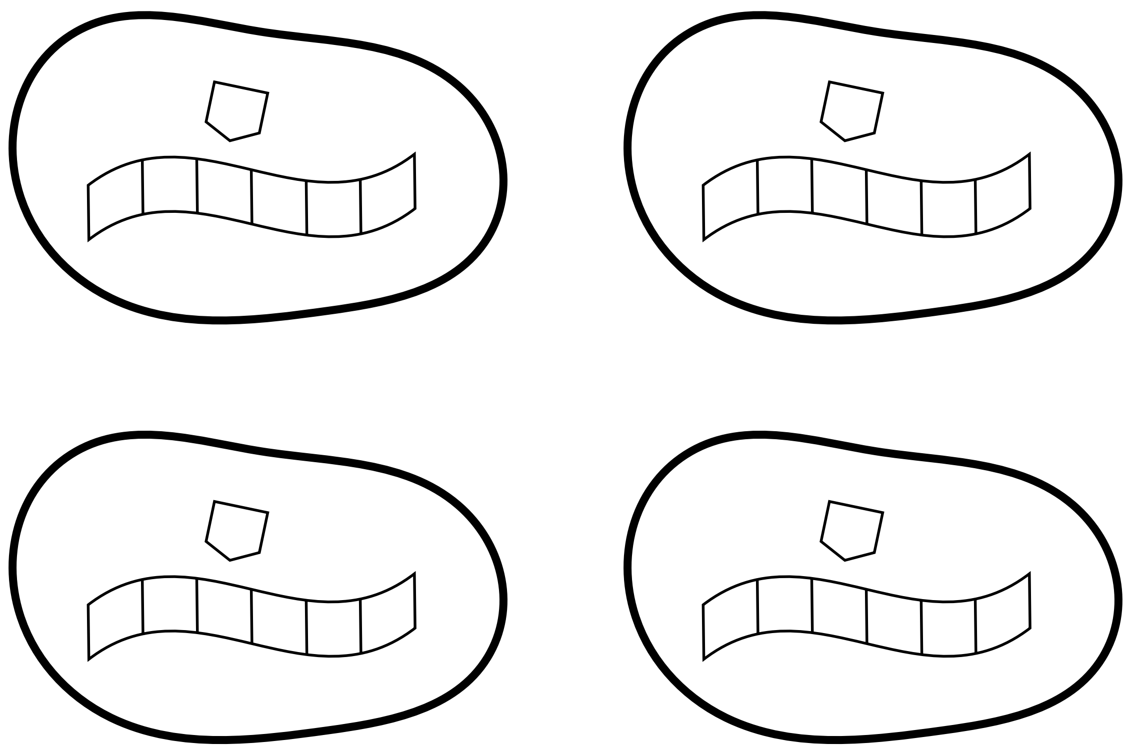
# Simulating NDTM ⇒ **NP**



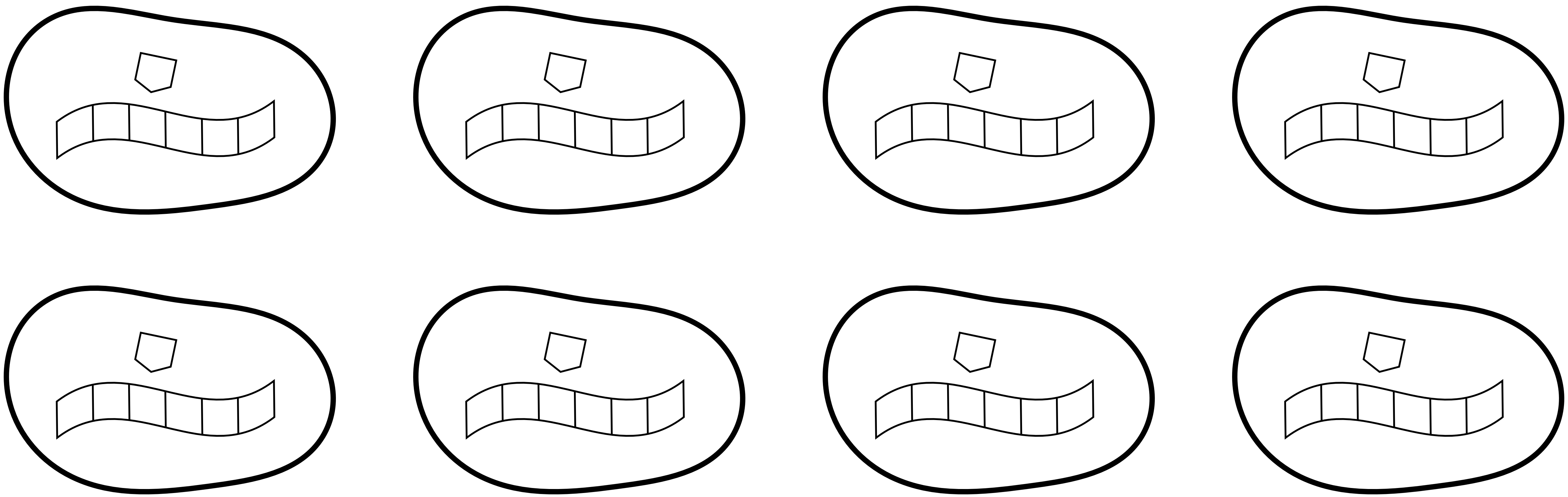
# Simulating NDTM ⇒ **NP**



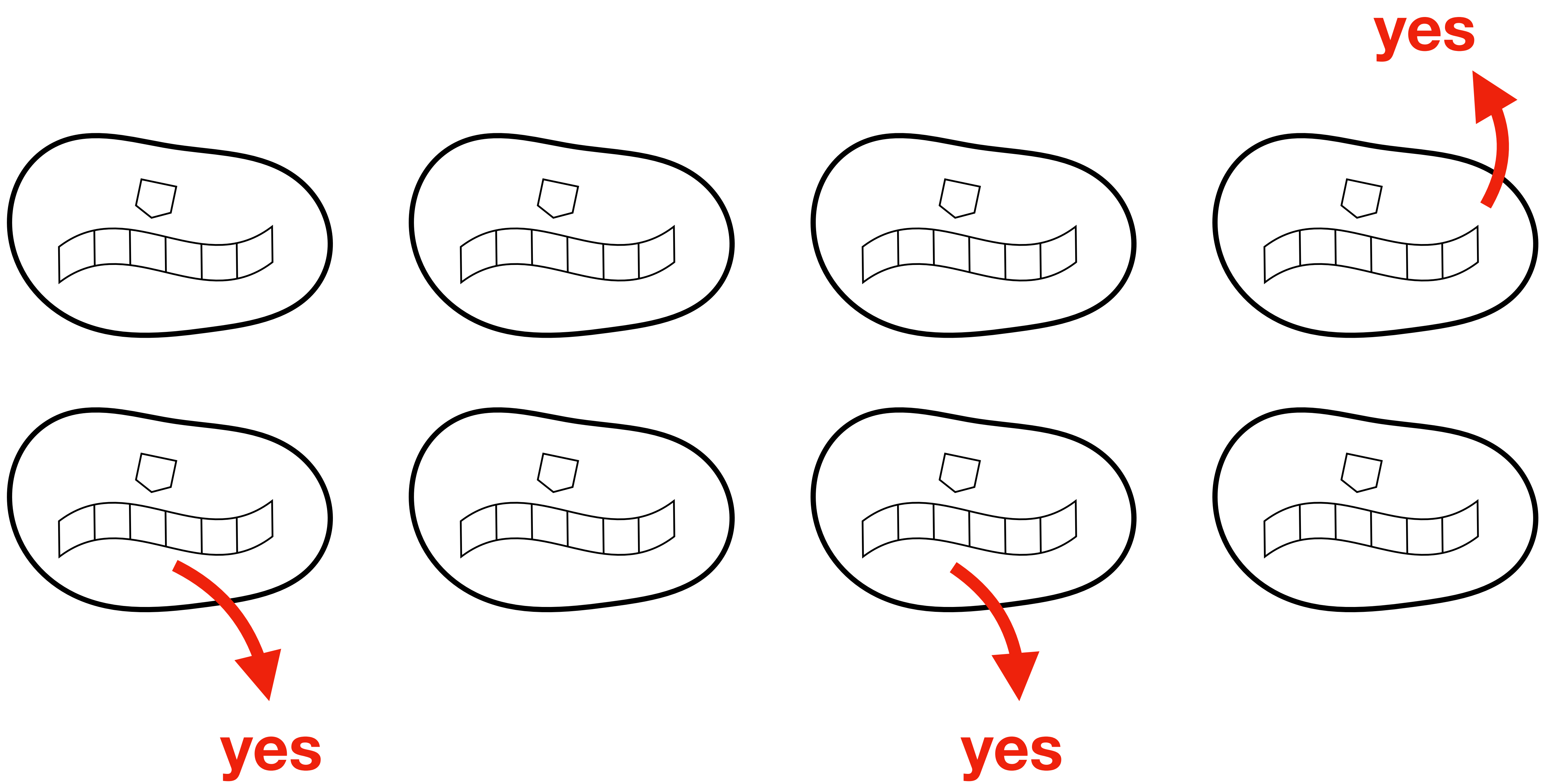
# Simulating NDTM ⇒ **NP**



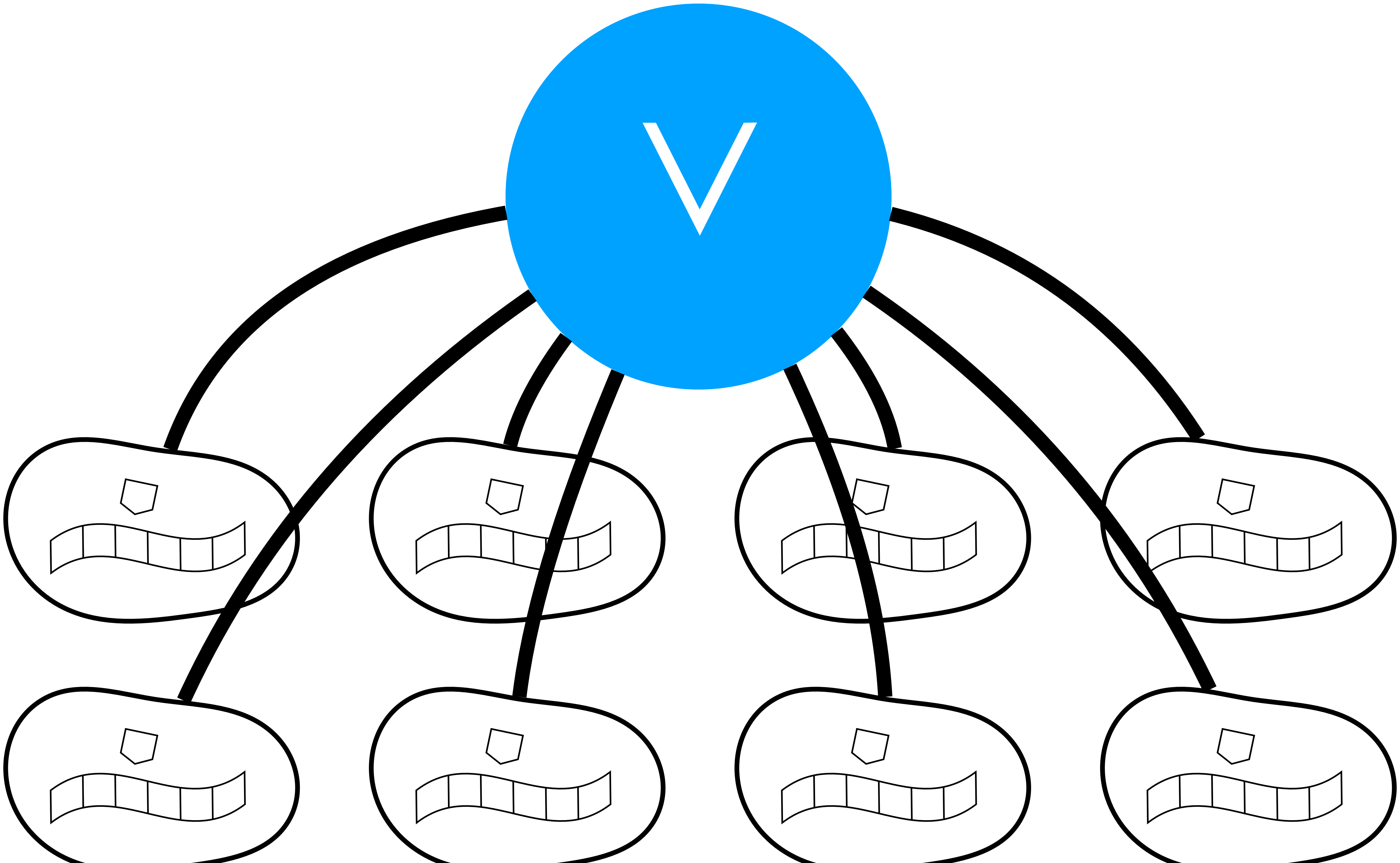
# Simulating NDTM ⇒ **NP**



# Simulating NDTM ⇒ NP

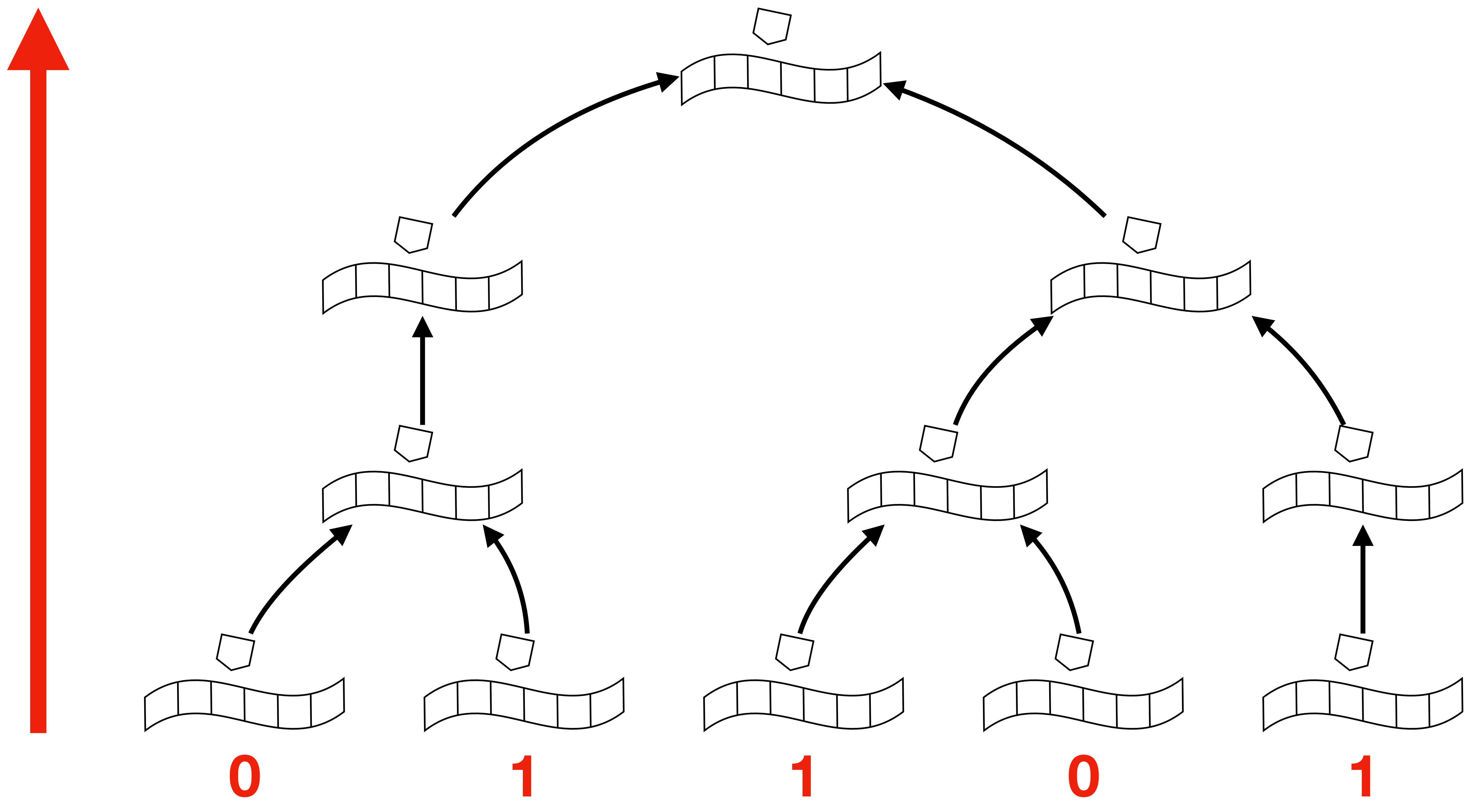


# Simulating NDTM ⇒ **NP**

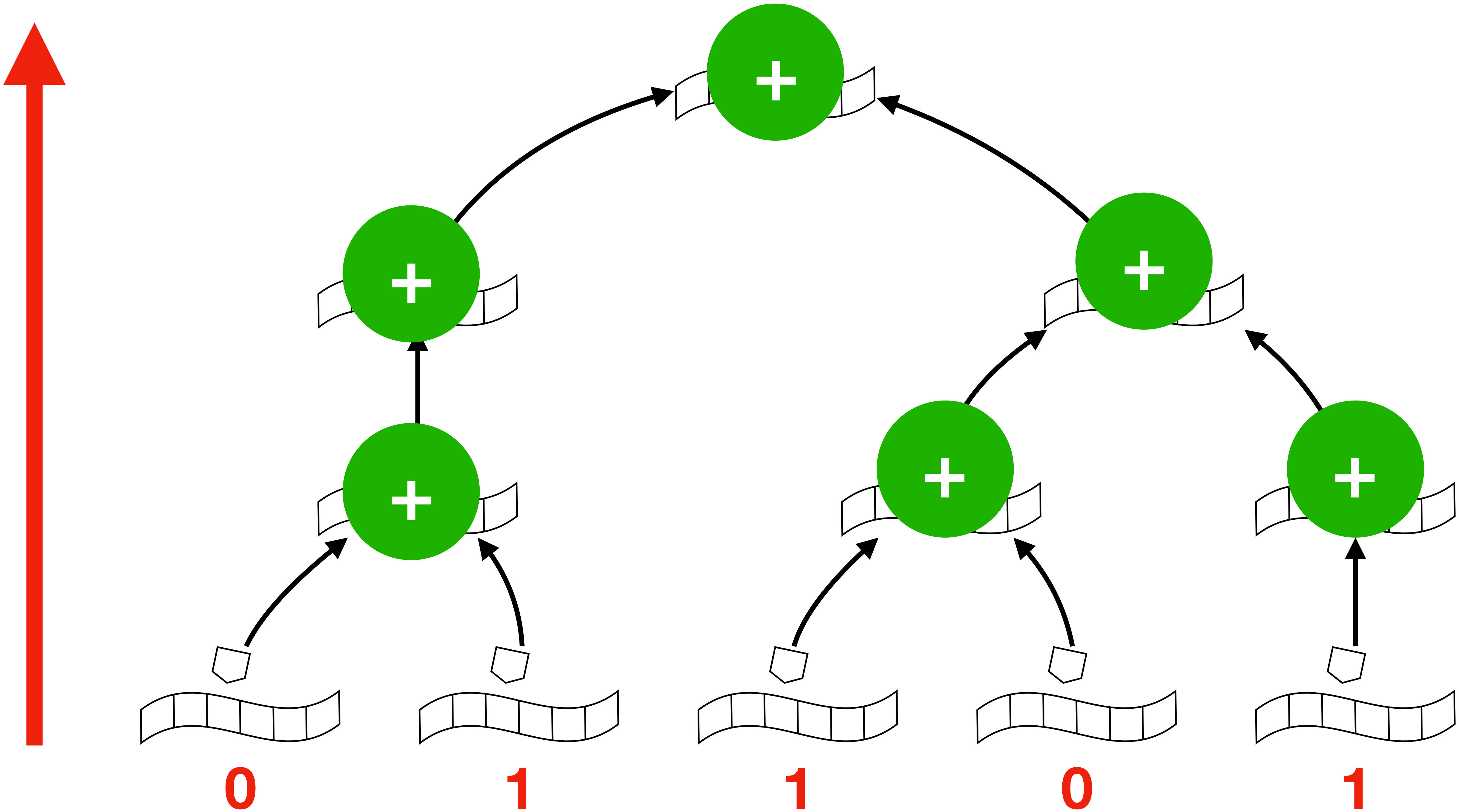


# Counting complexity

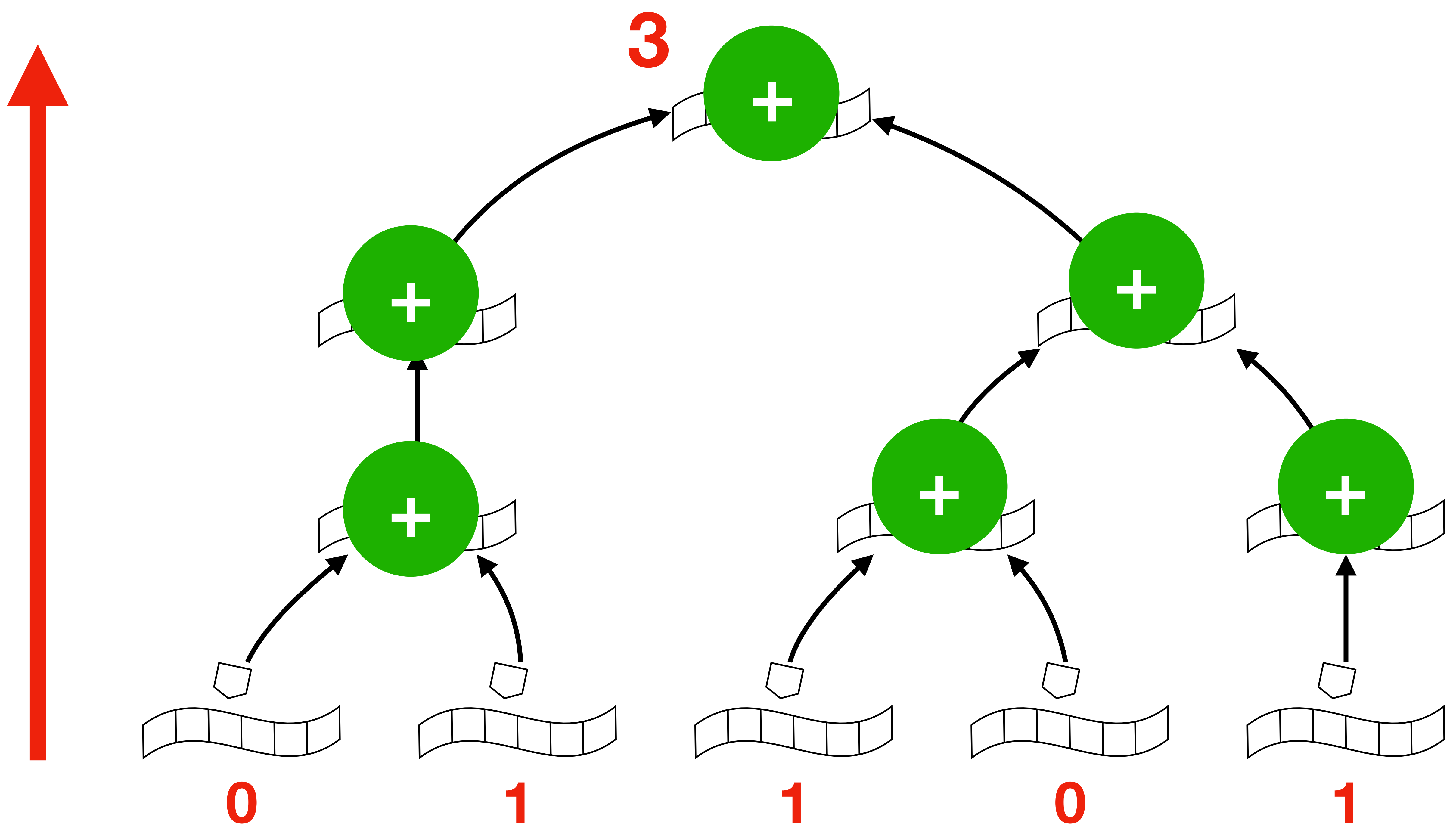
# Counting
# Turing machines: #P



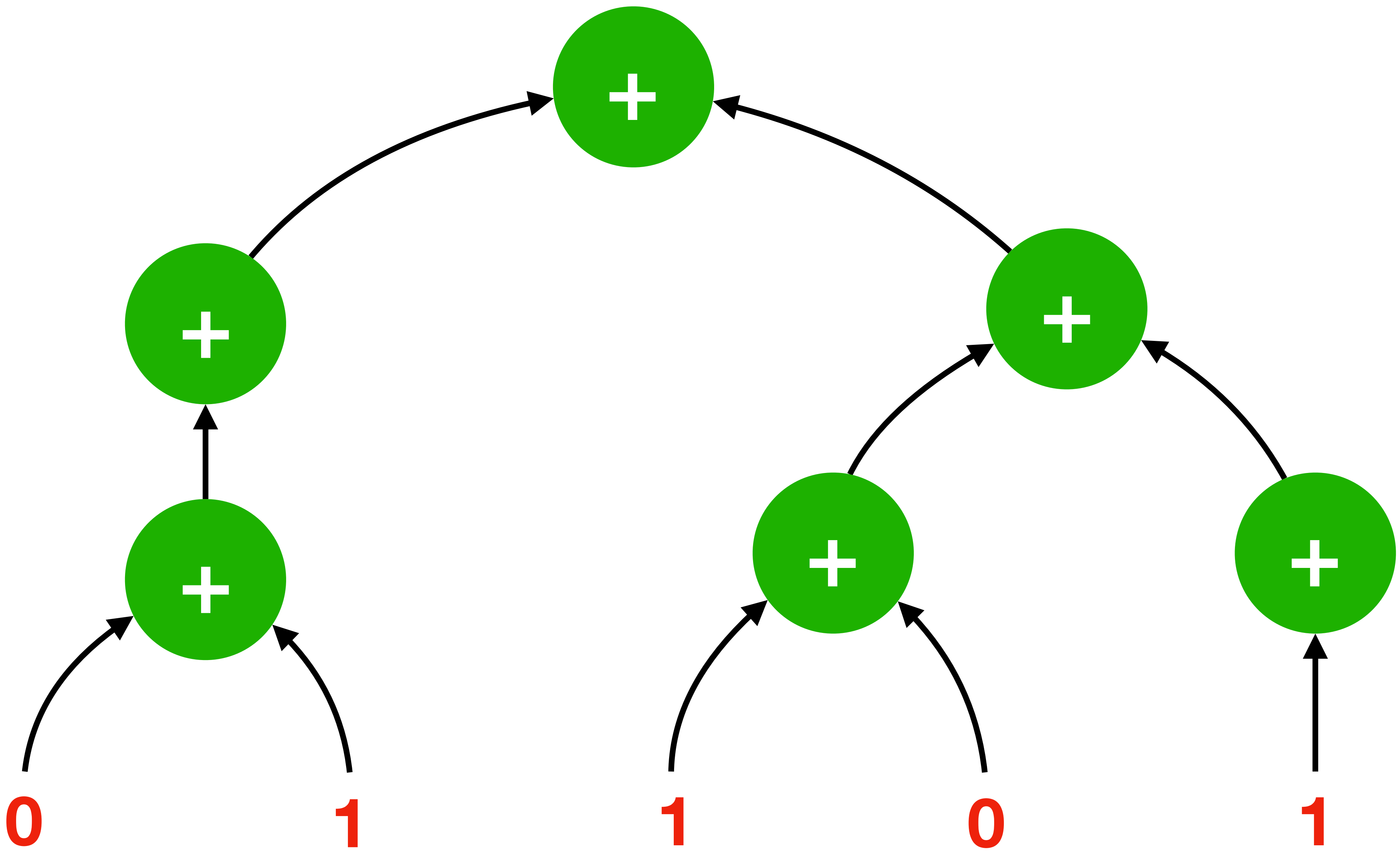
# Counting
# Turing machines: #P



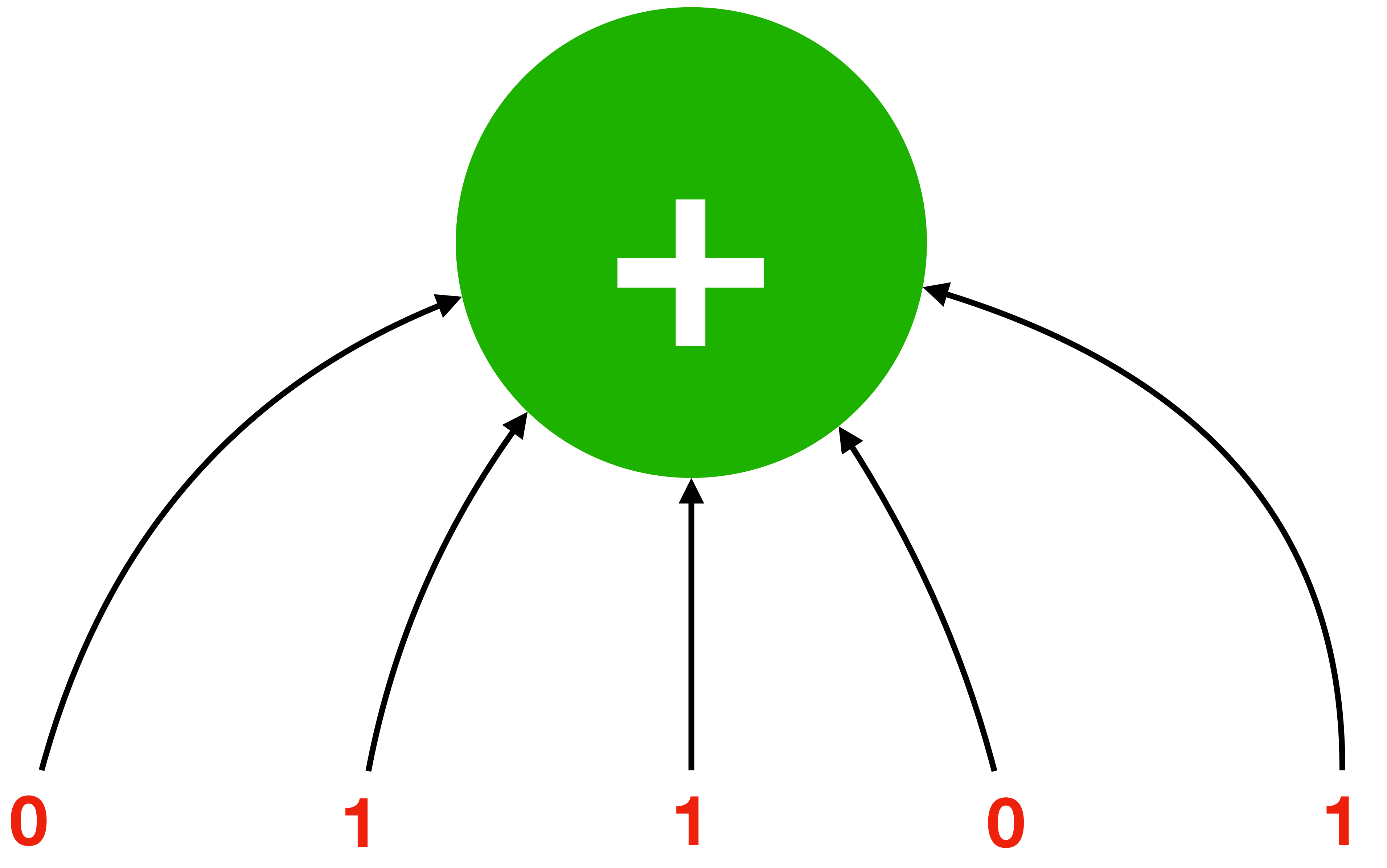
# Counting
# Turing machines: #P



# Flattening ⊕-circuits

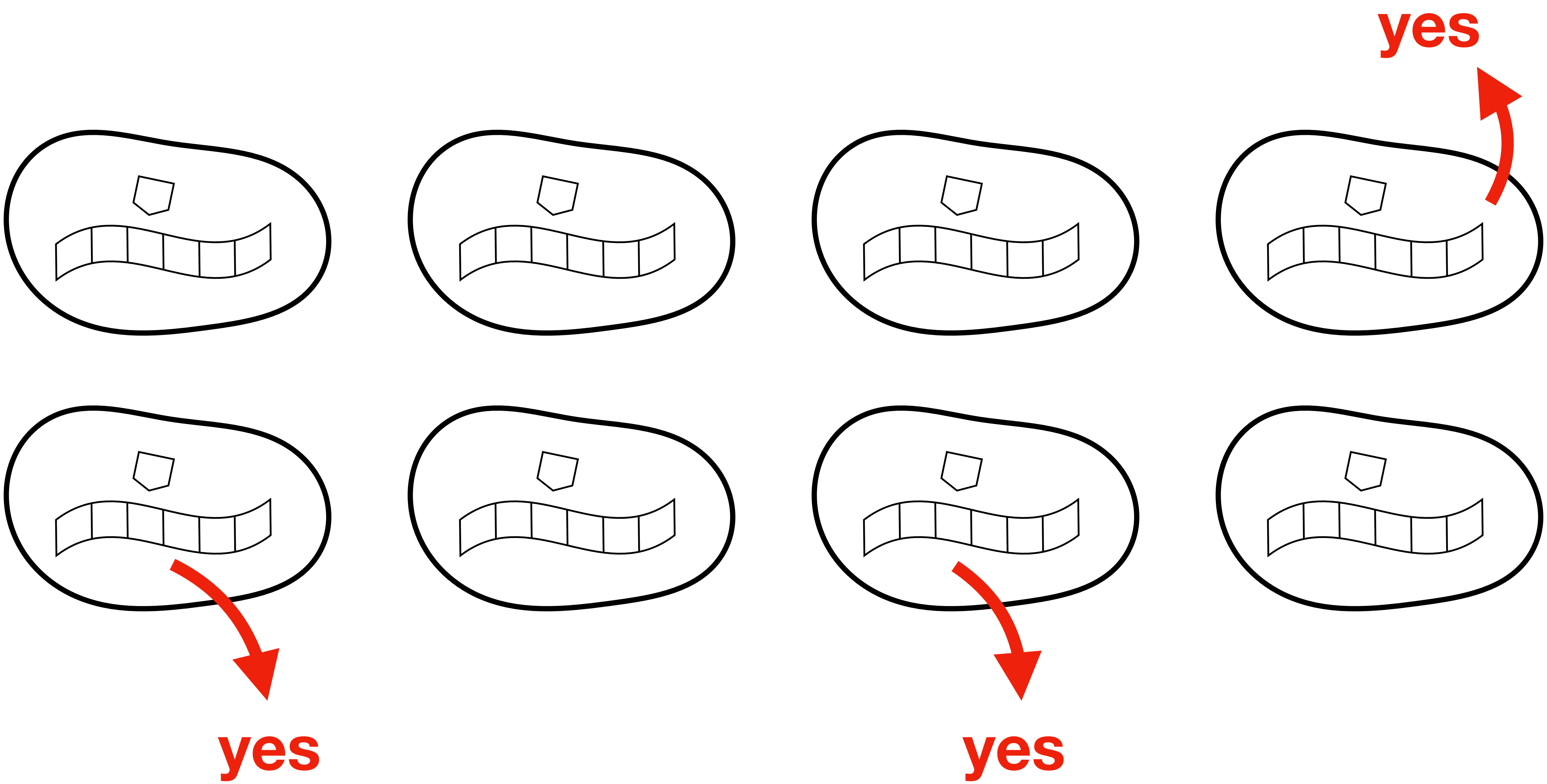


# Flattening ⊕-circuits

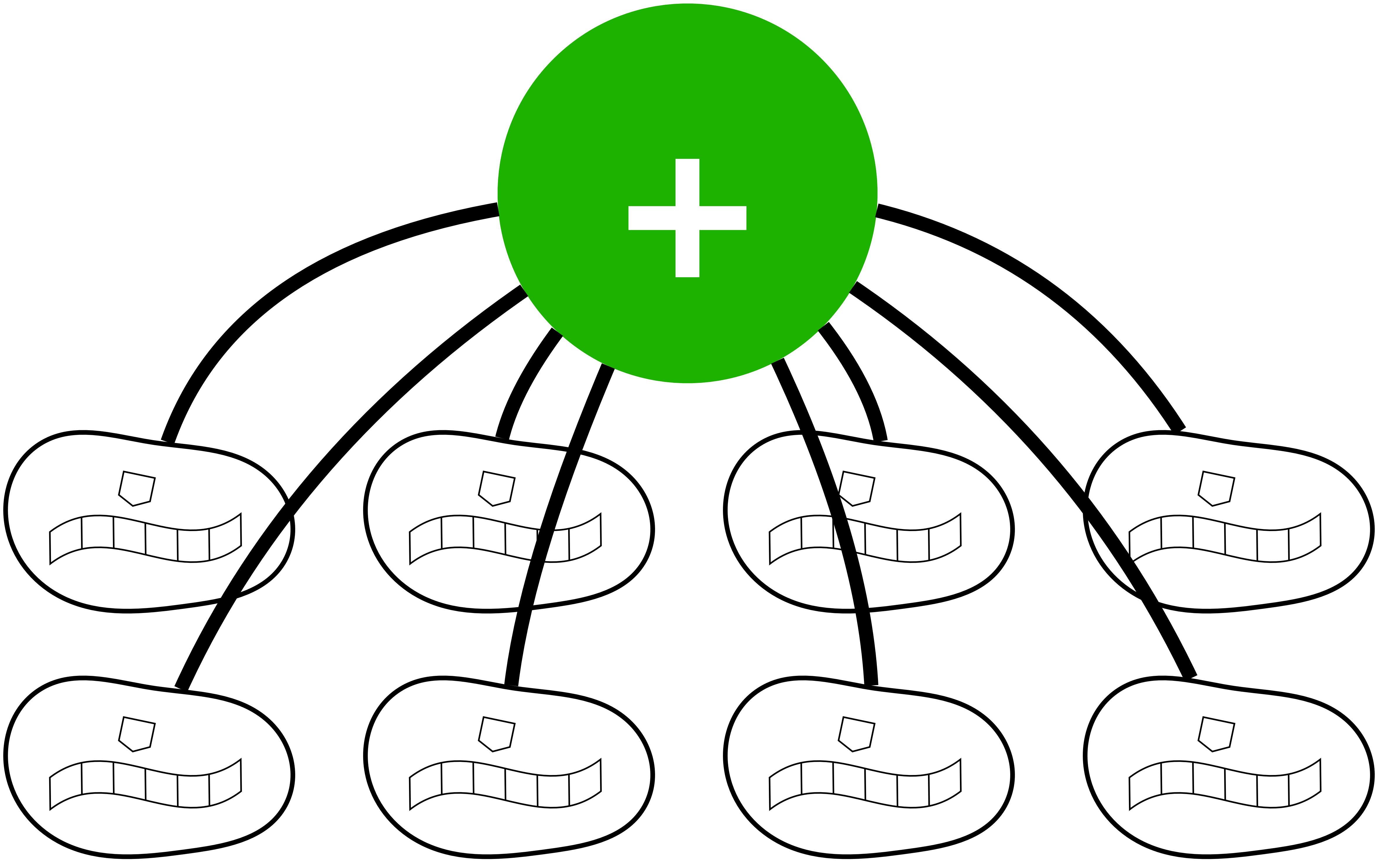


# Counting with membrane systems

# Simulating CTM ⇒ #P

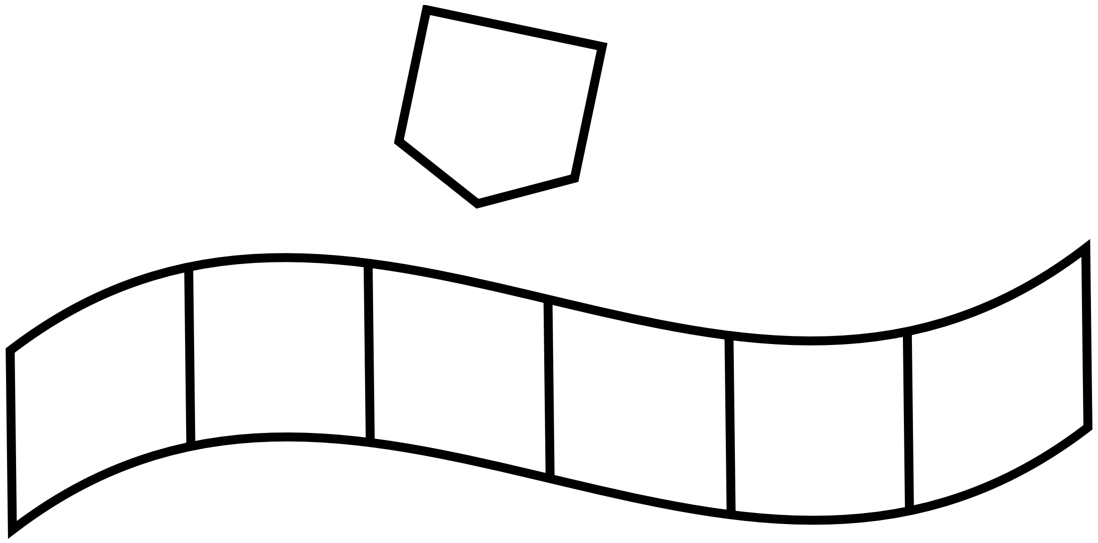


# Simulating CTM ⇒ #P

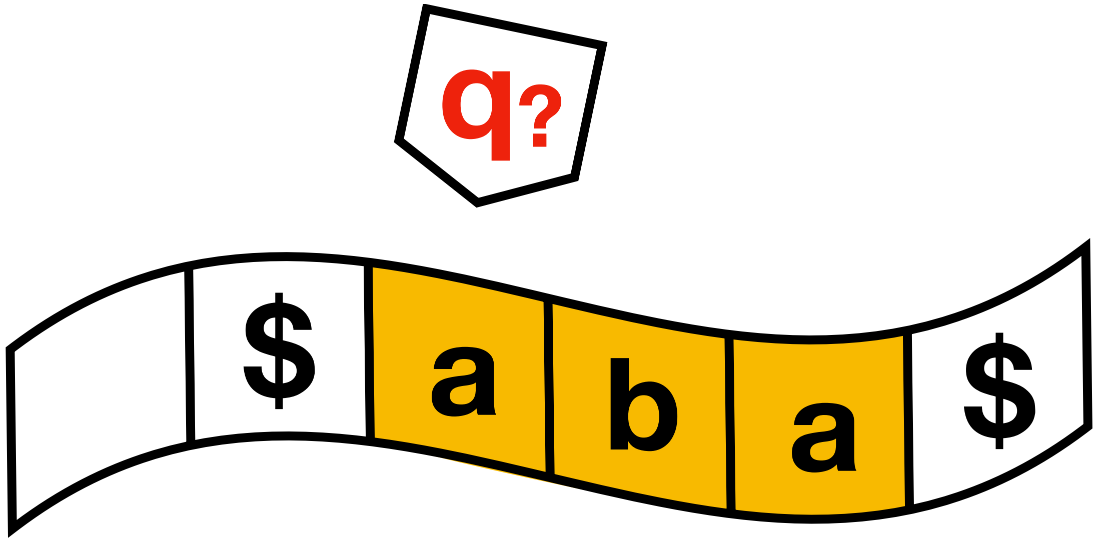


# Oracles 🧞‍♀️

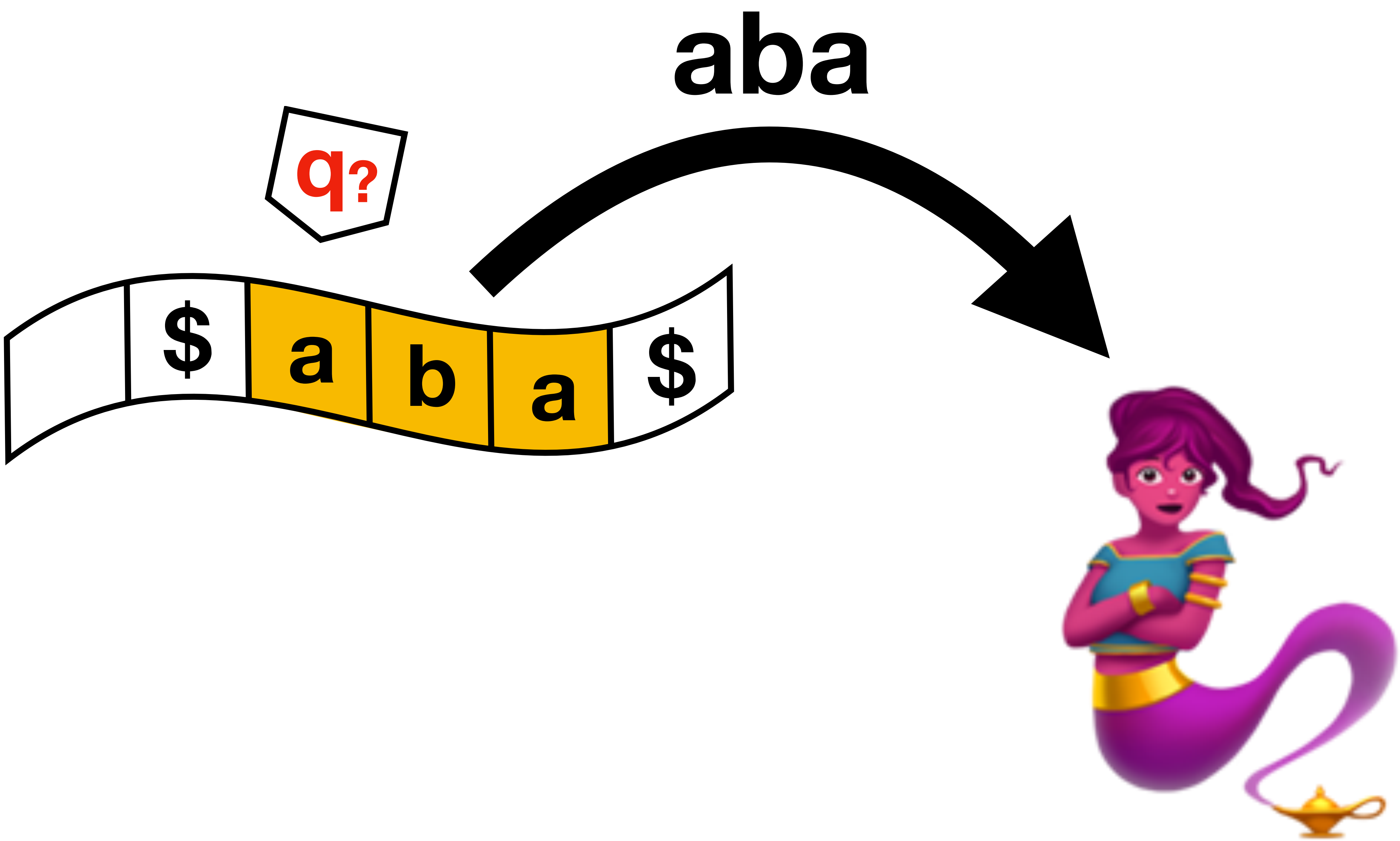
# TM with oracle



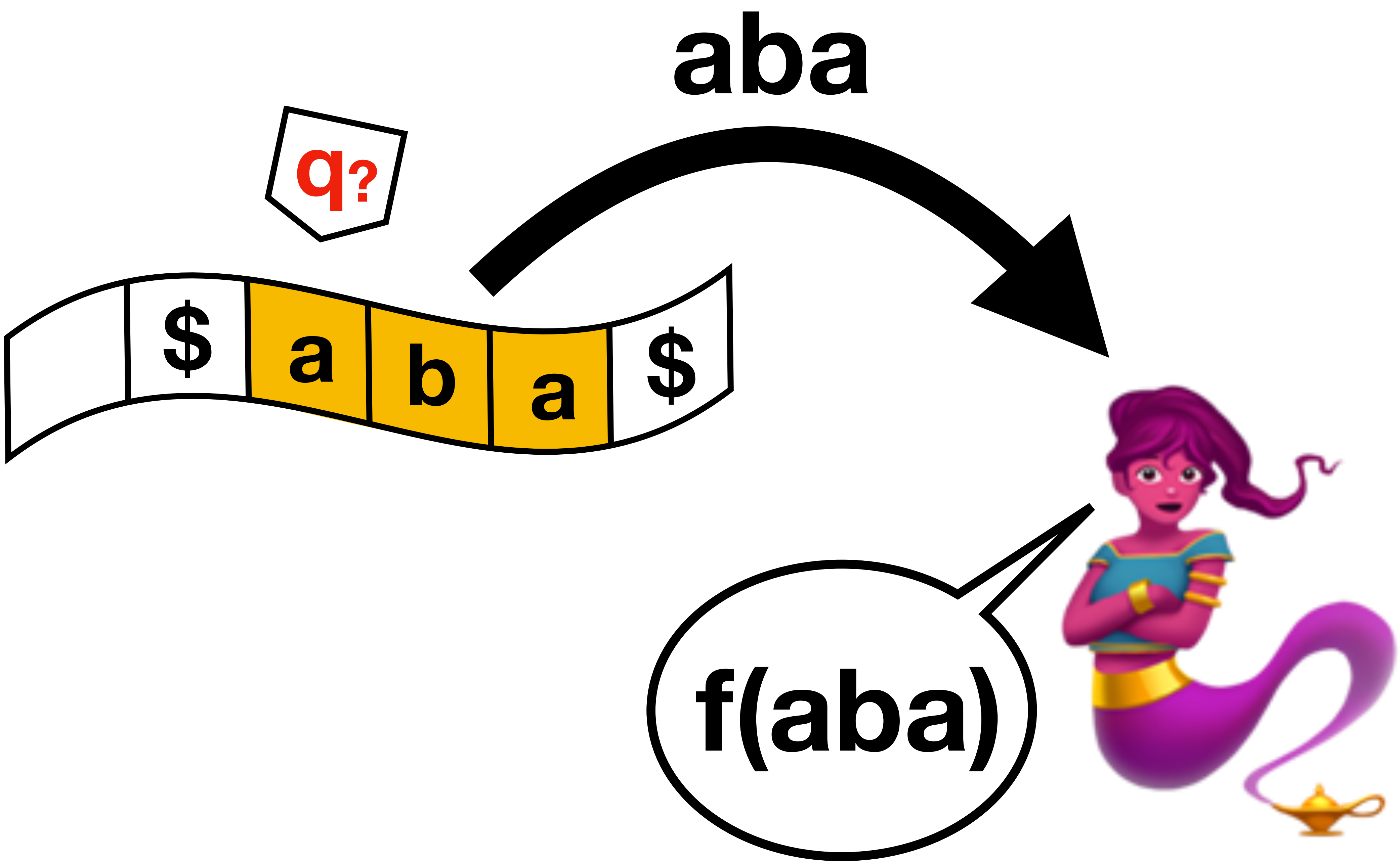
# TM with oracle



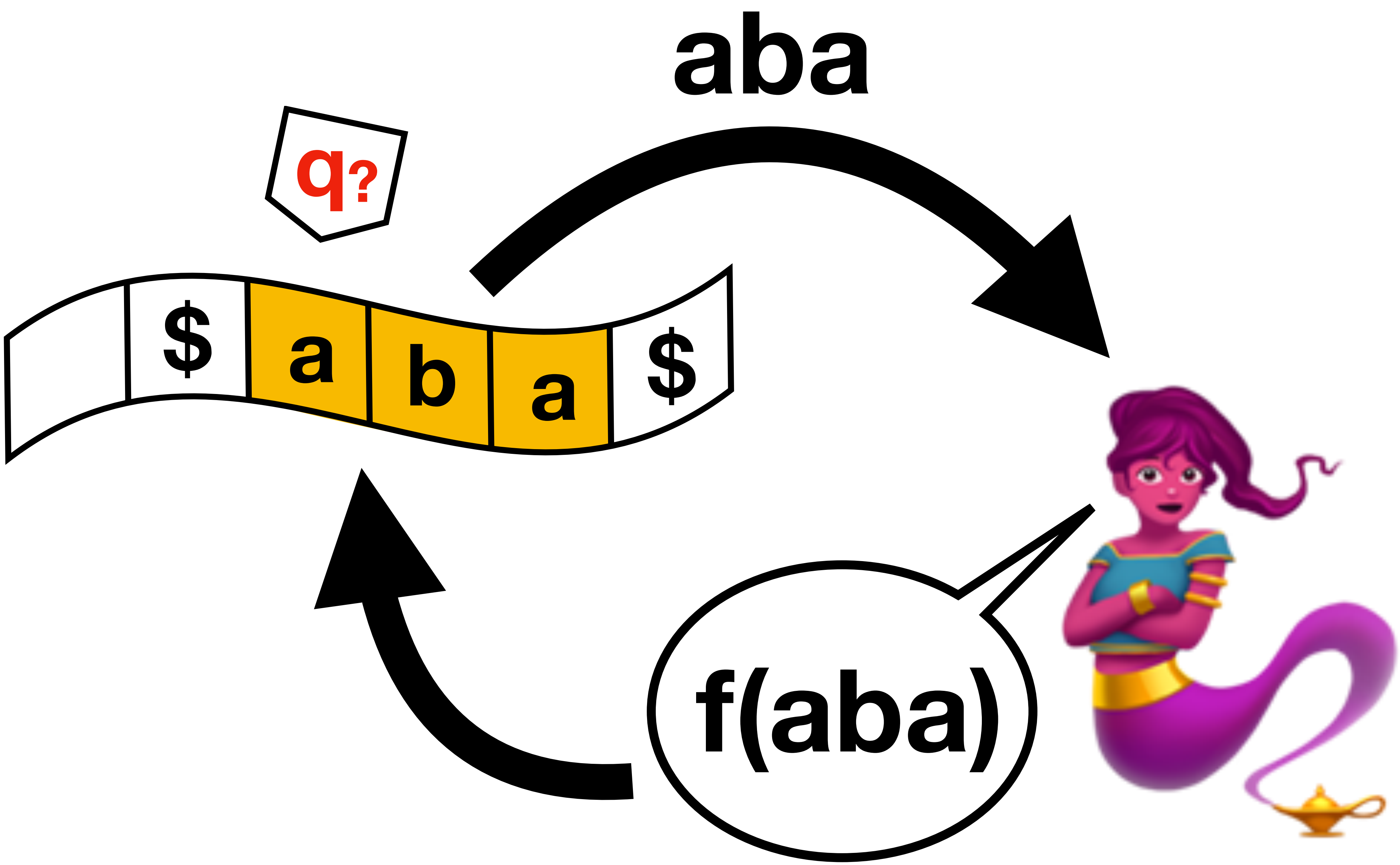
# TM with oracle



# TM with oracle

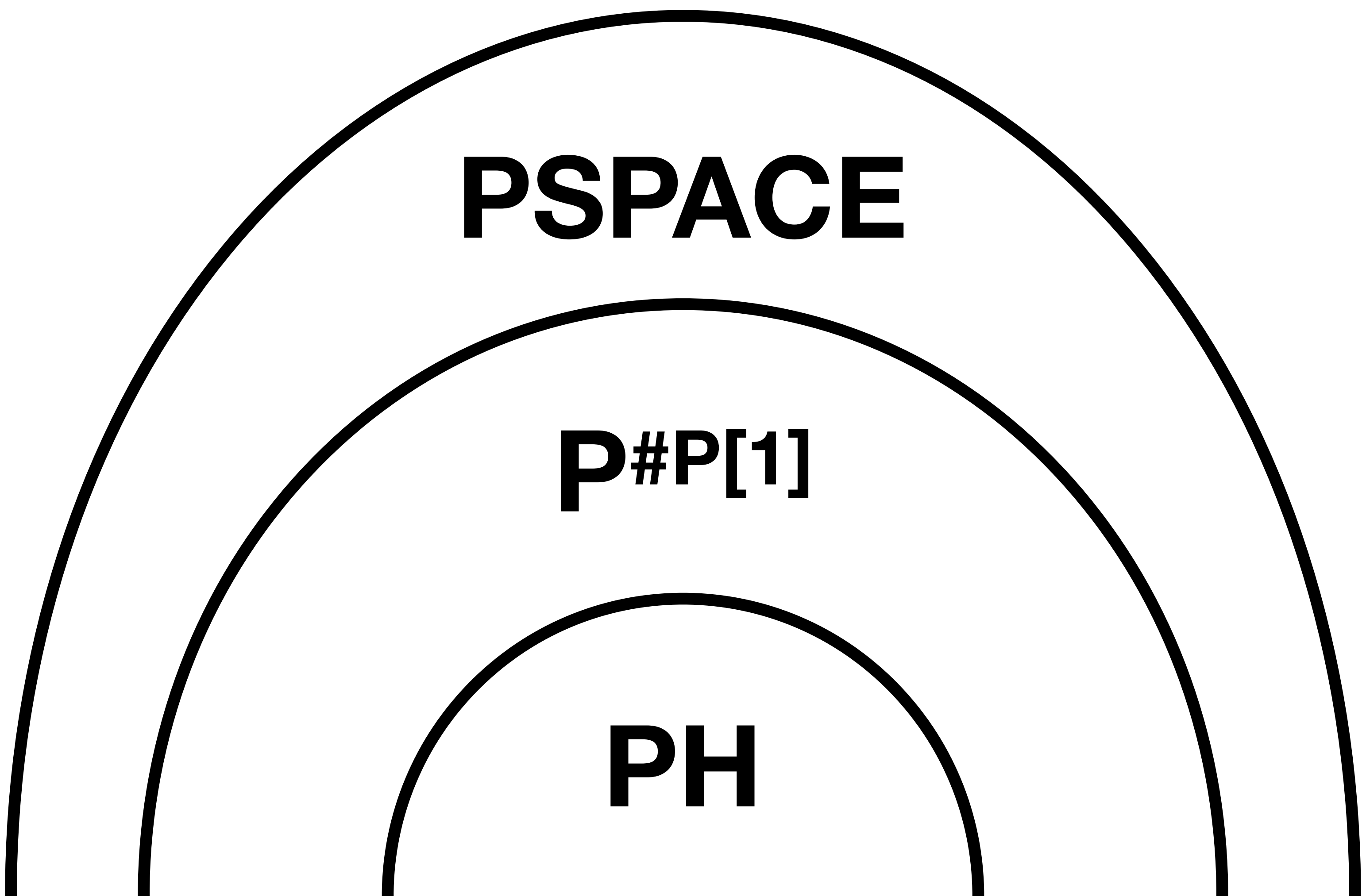


# TM with oracle



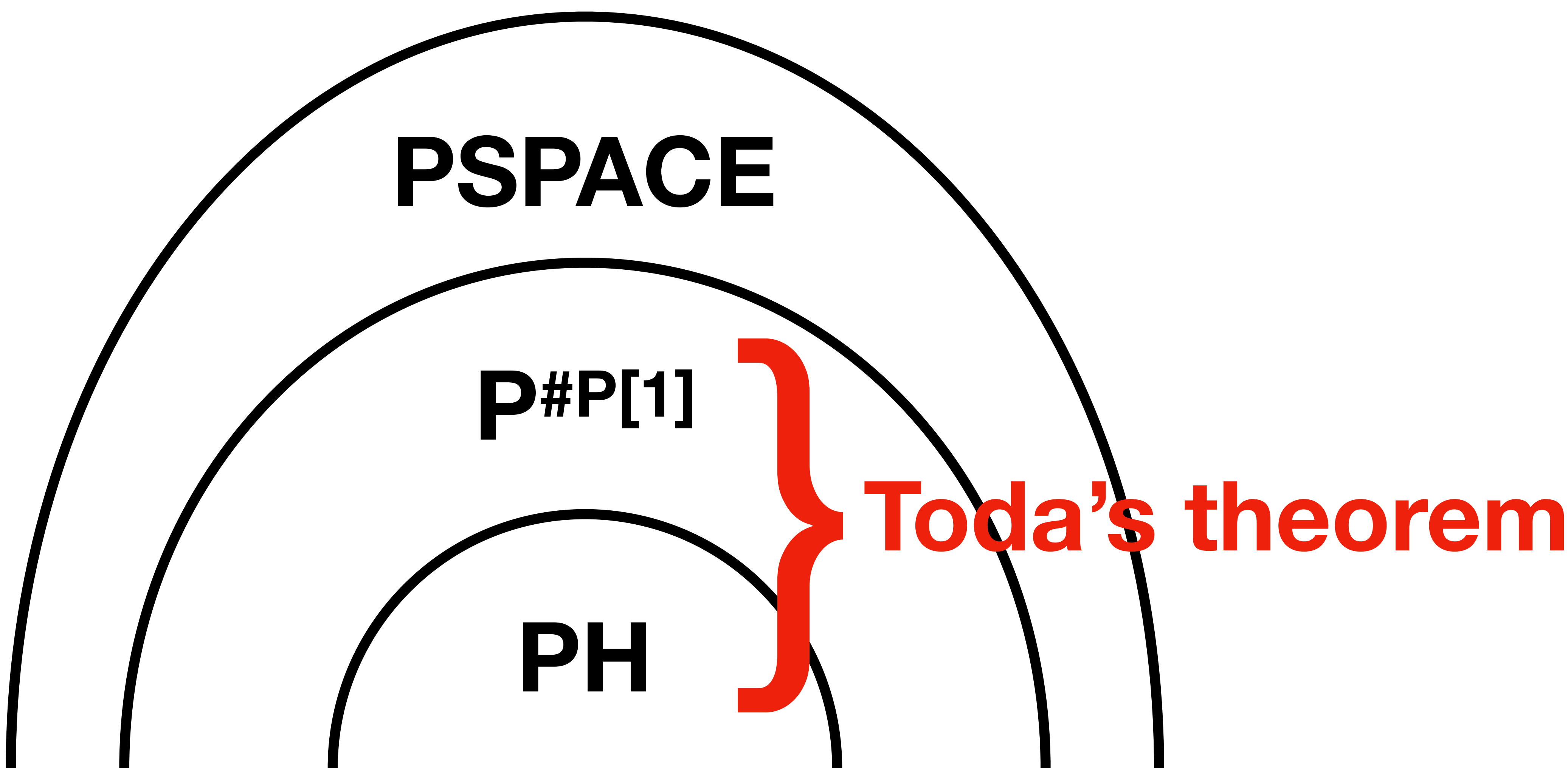
# The complexity class $\mathbf{P}^{\#P[1]}$

The class of problems solved by
polynomial-time Turing machines with a single query
to an oracle for a **#P**-complete problem



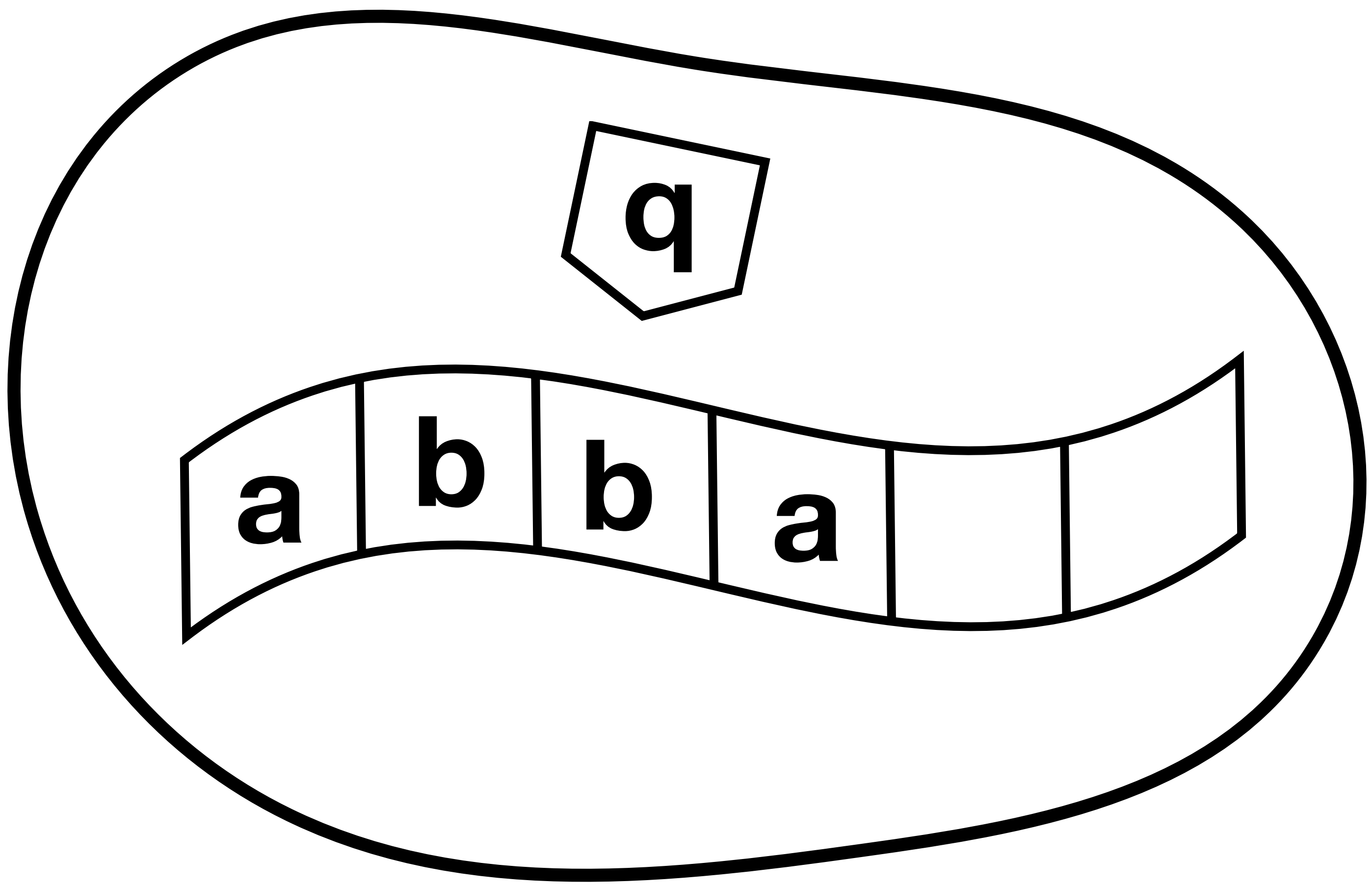
# The complexity class $\mathbf{P}^{\#\mathbf{P}[1]}$

The class of problems solved by polynomial-time Turing machines with a single query to an oracle for a **#P**-complete problem

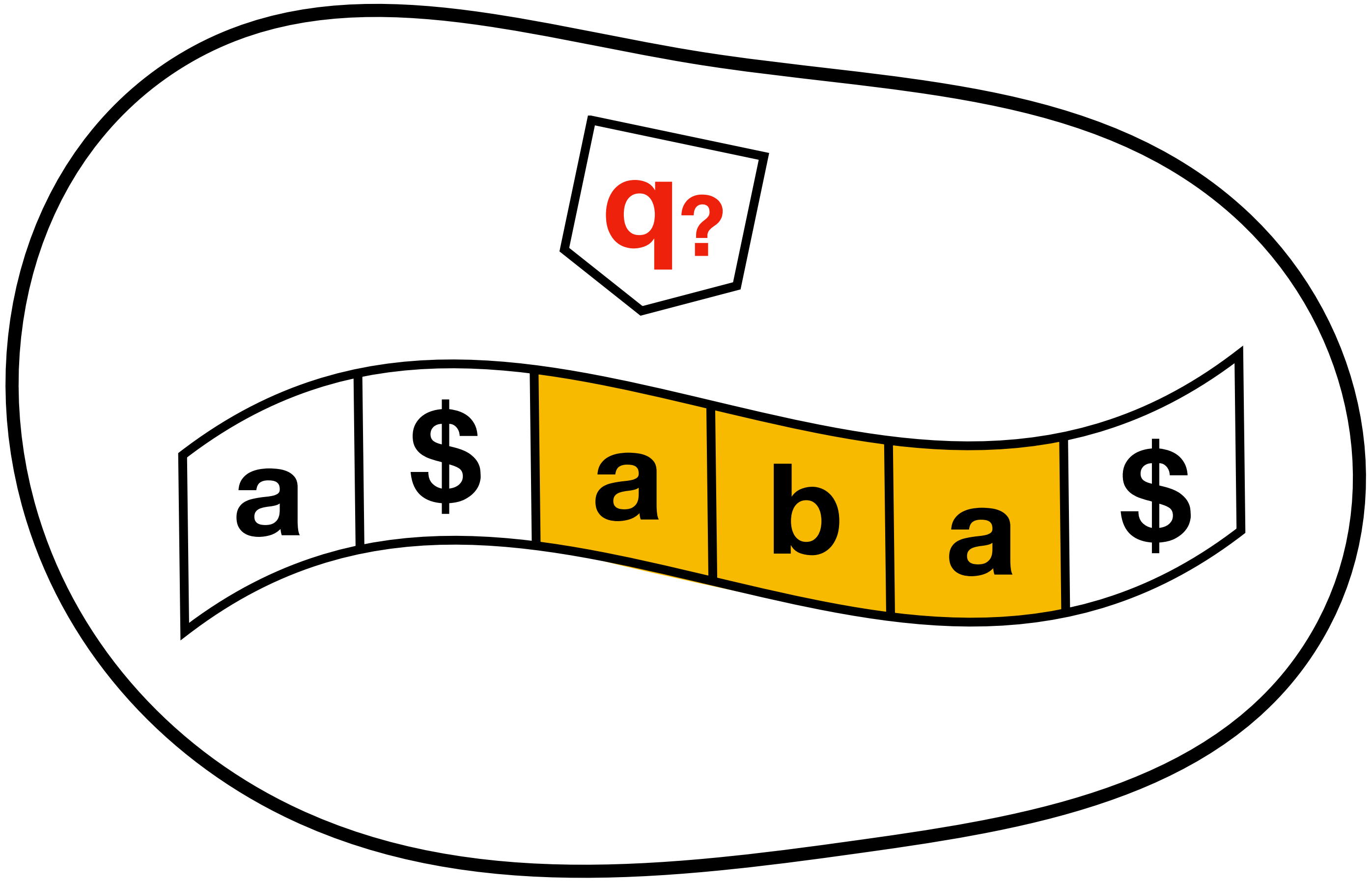


# Solving $\mathbf{P}^{\#\mathrm{P}[1]}$ with (monodirectional, shallow) membrane systems

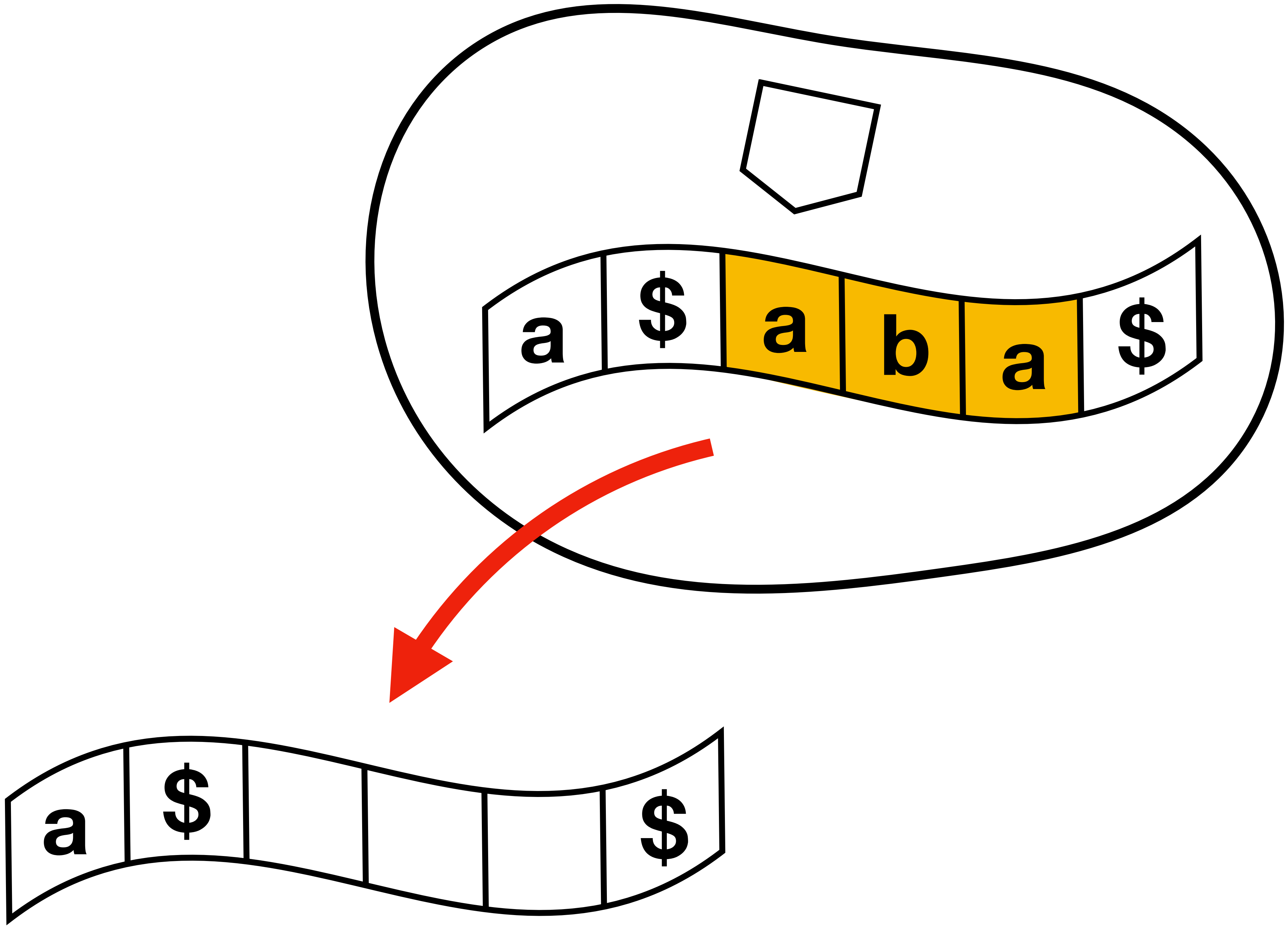
# Pre-query computation



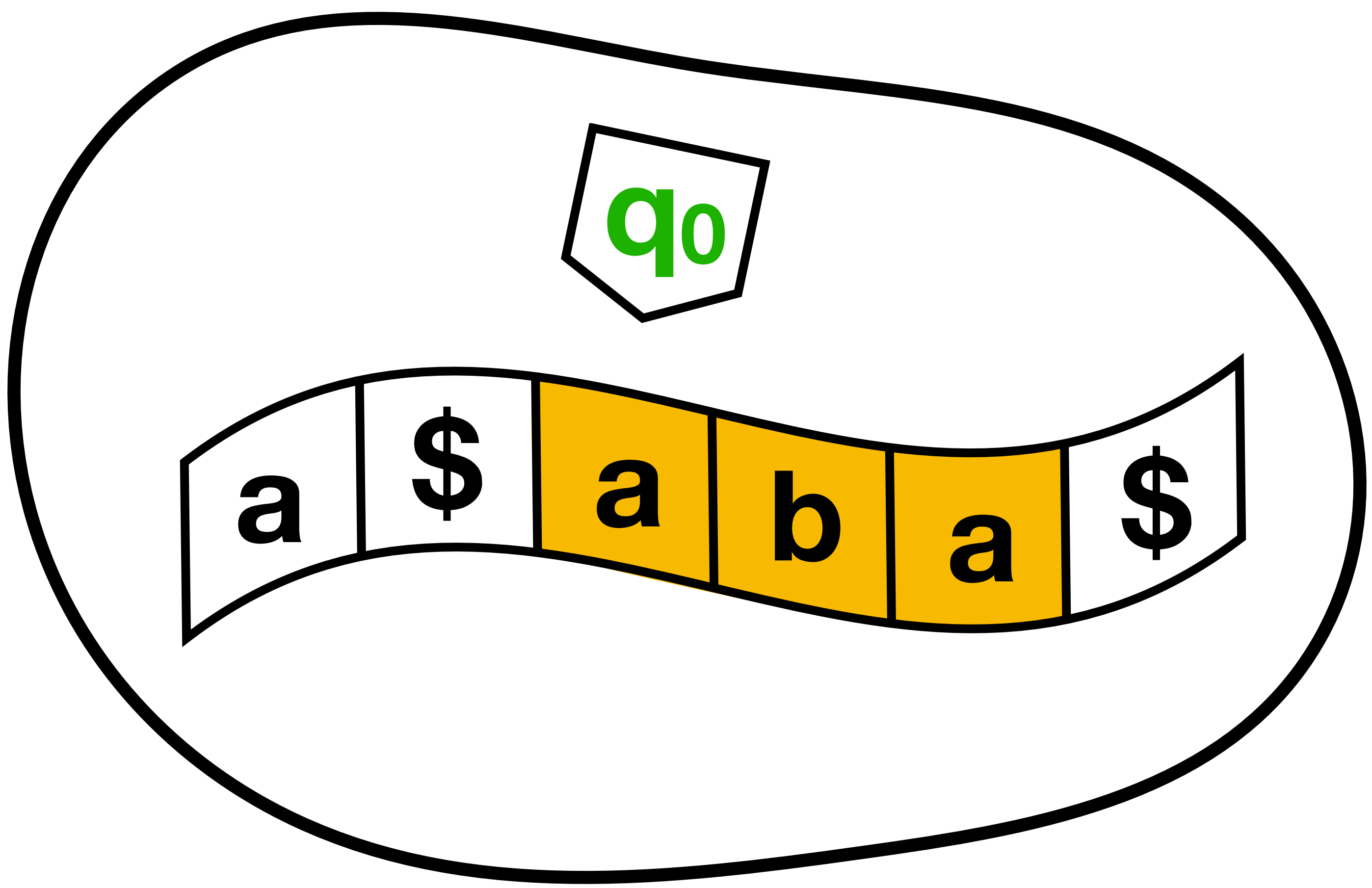
# Entering the query state



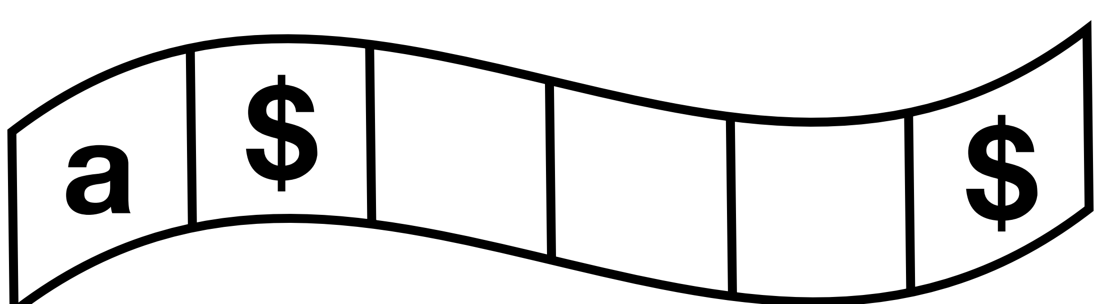
# Entering the query state



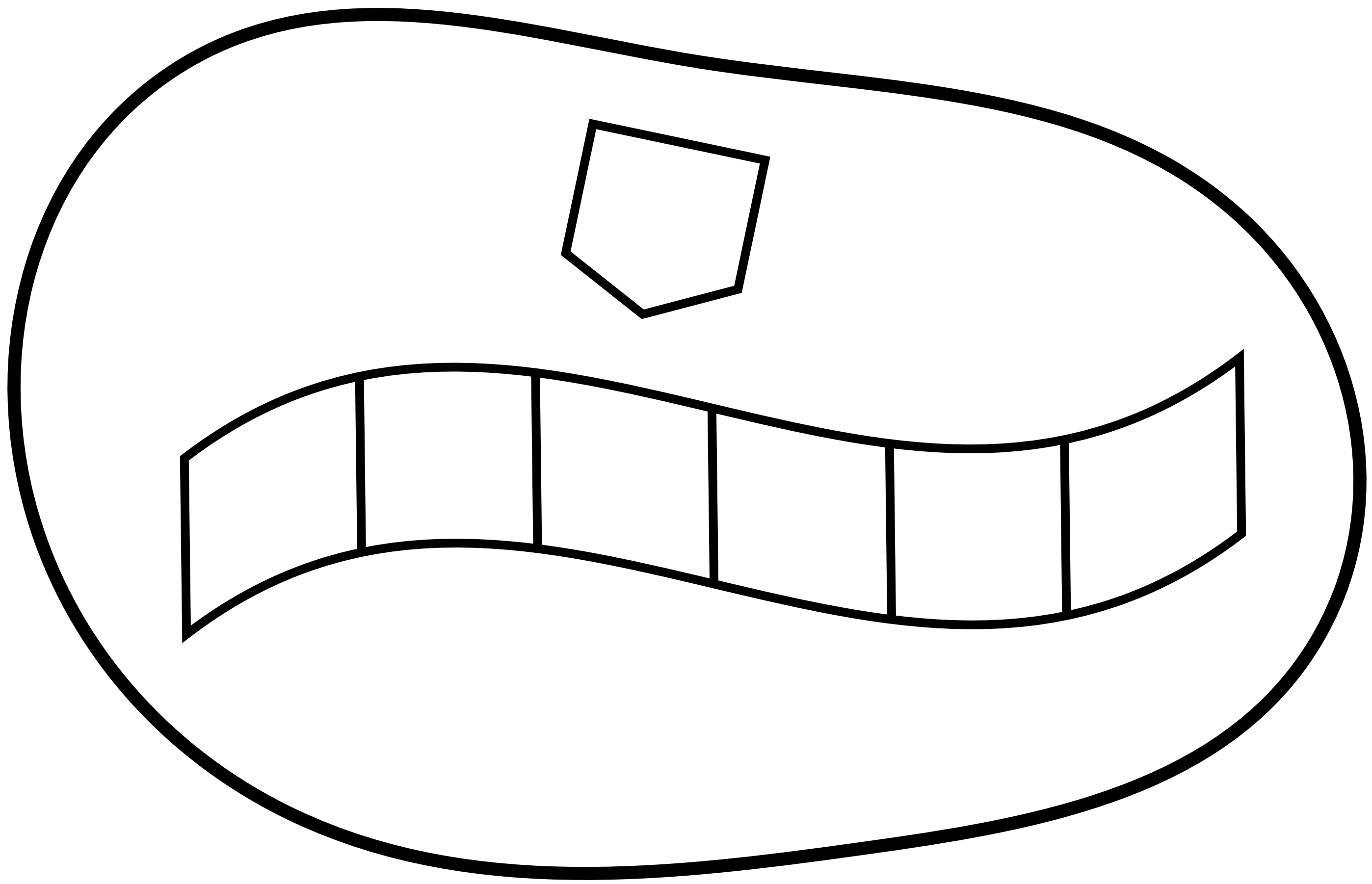
# Entering the query state

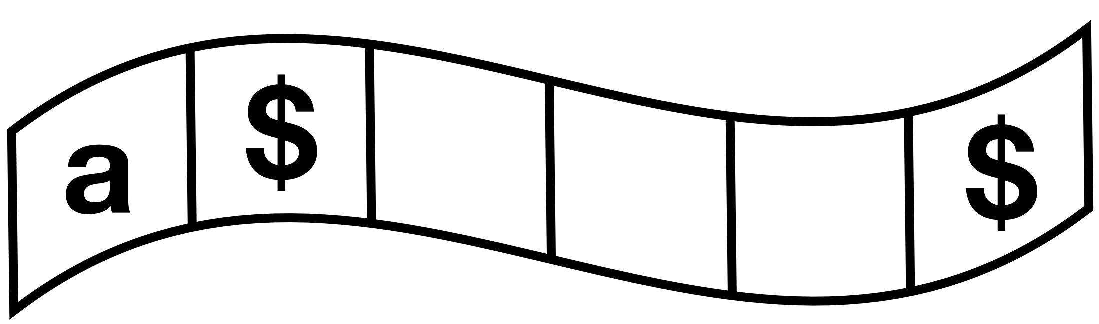



**Initial state of a TM solving the #P-complete oracle problem**

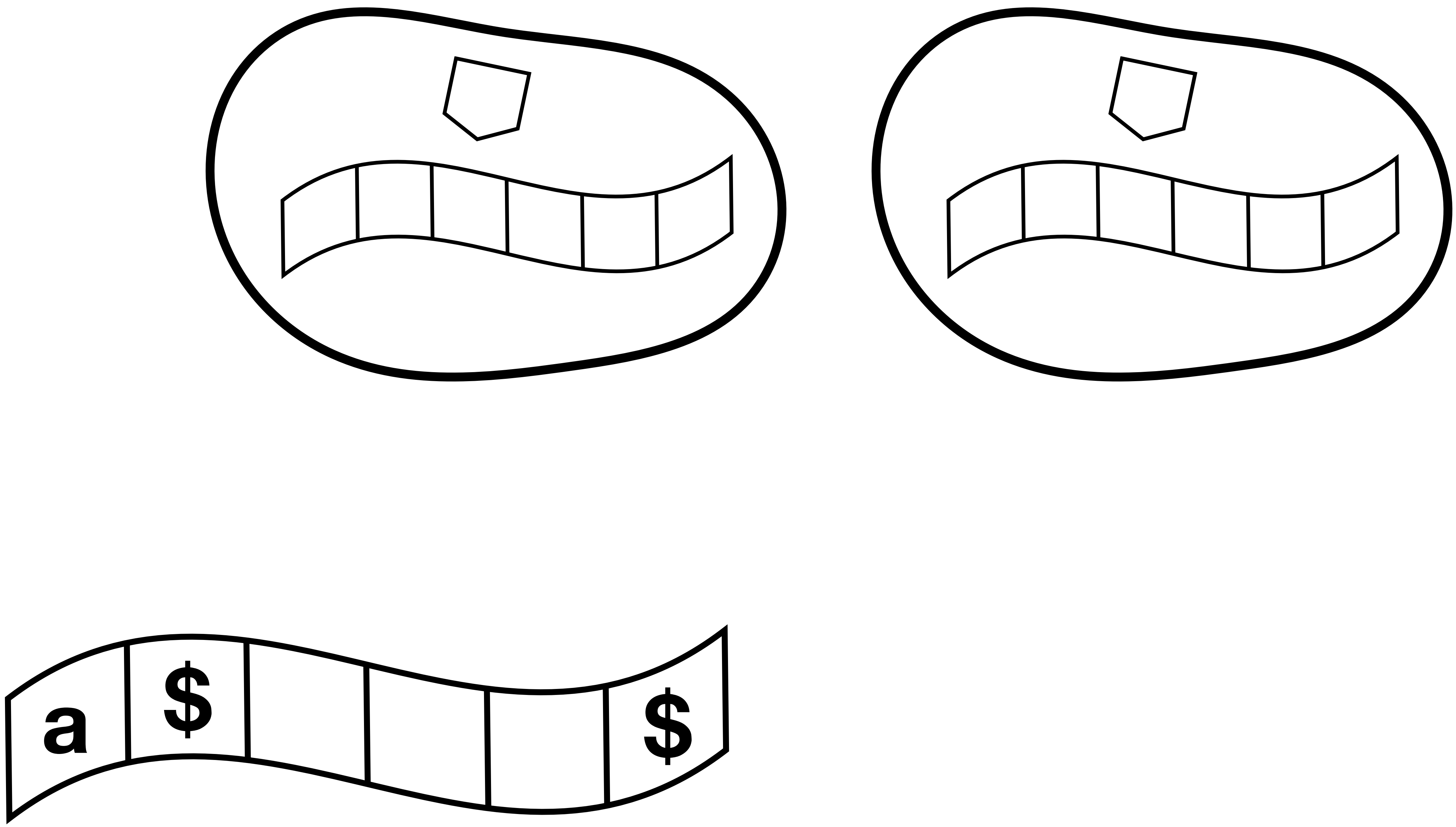


# Simulating the auxiliary TM

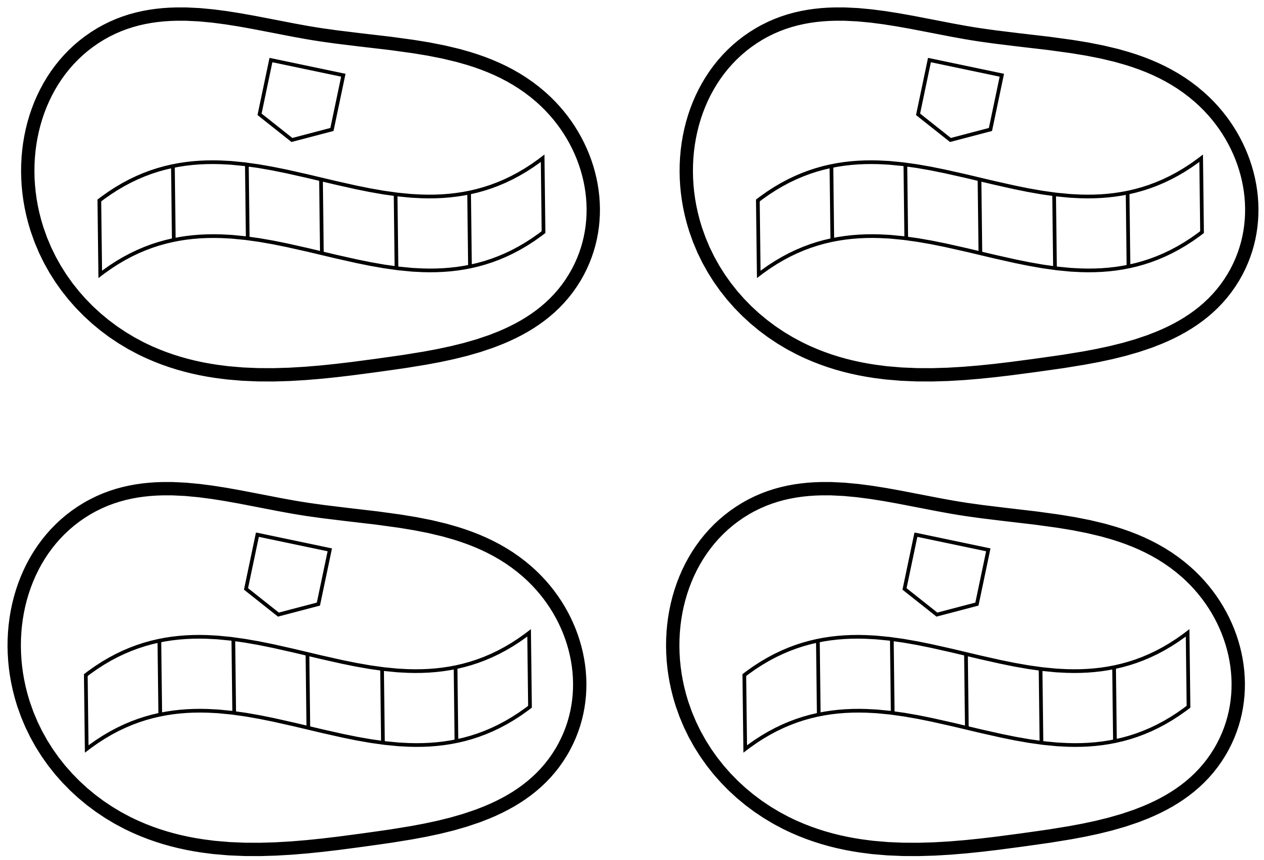


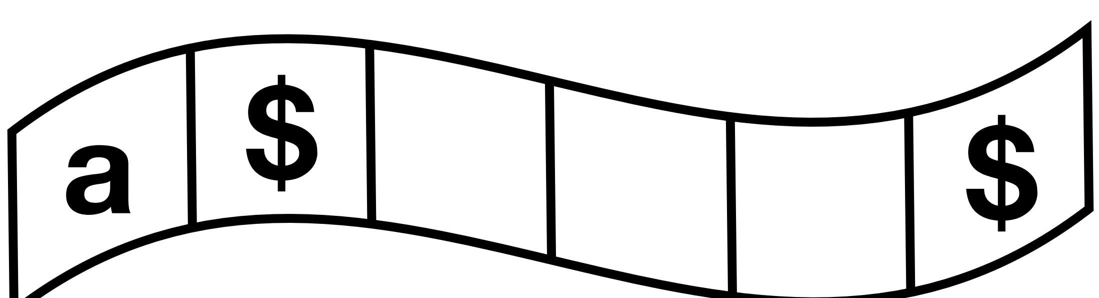


# Simulating the auxiliary TM

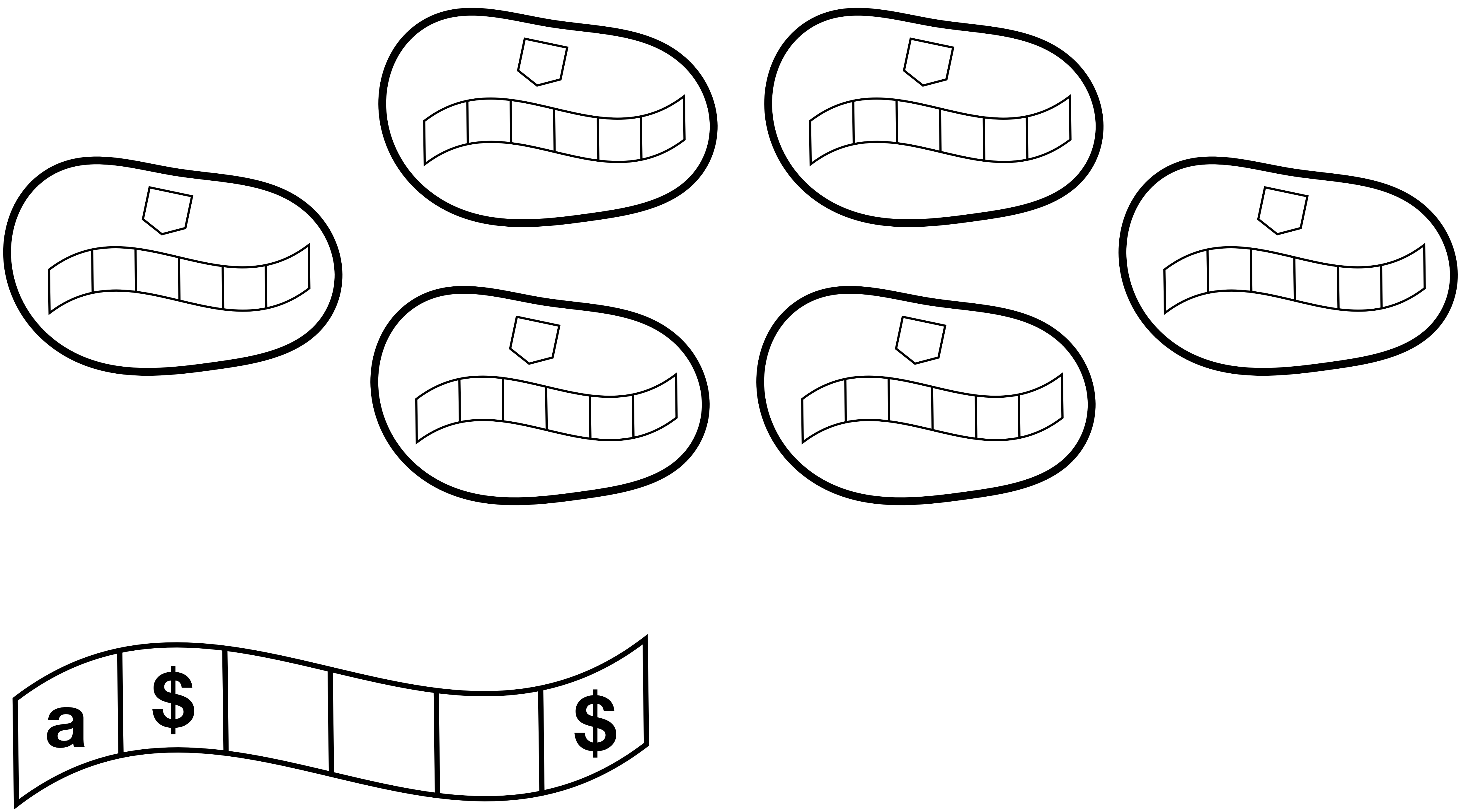


# Simulating the auxiliary TM

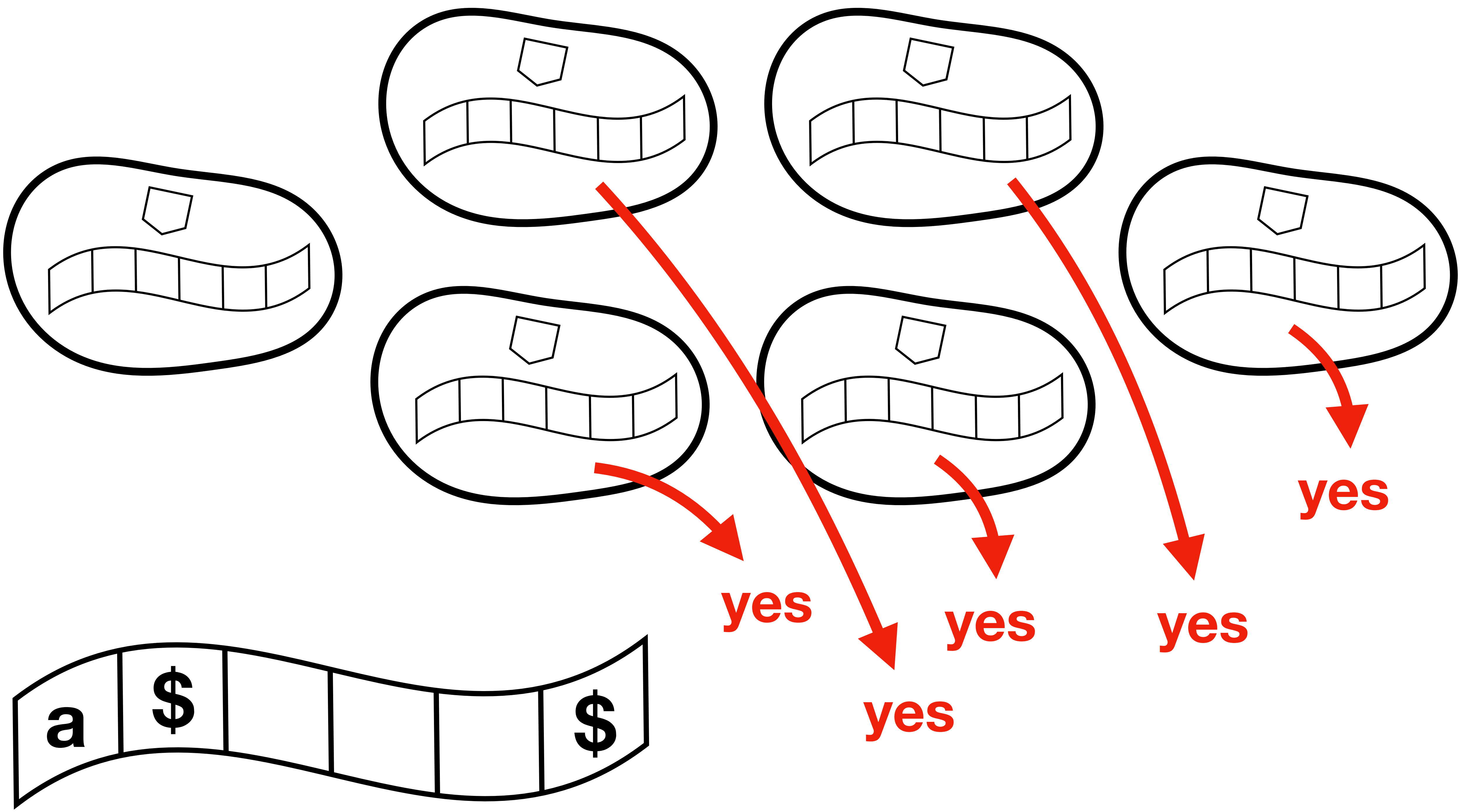




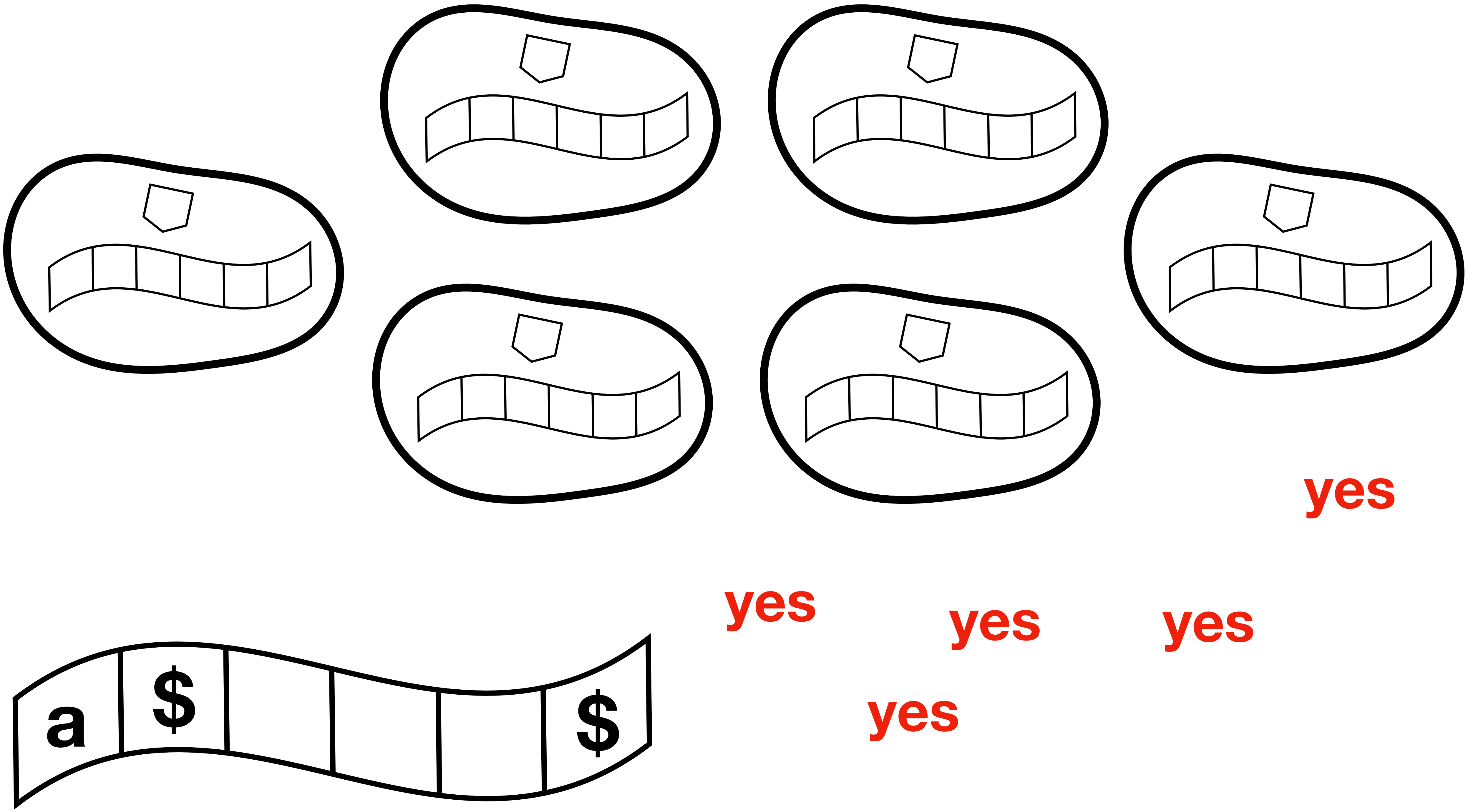
# Simulating the auxiliary TM



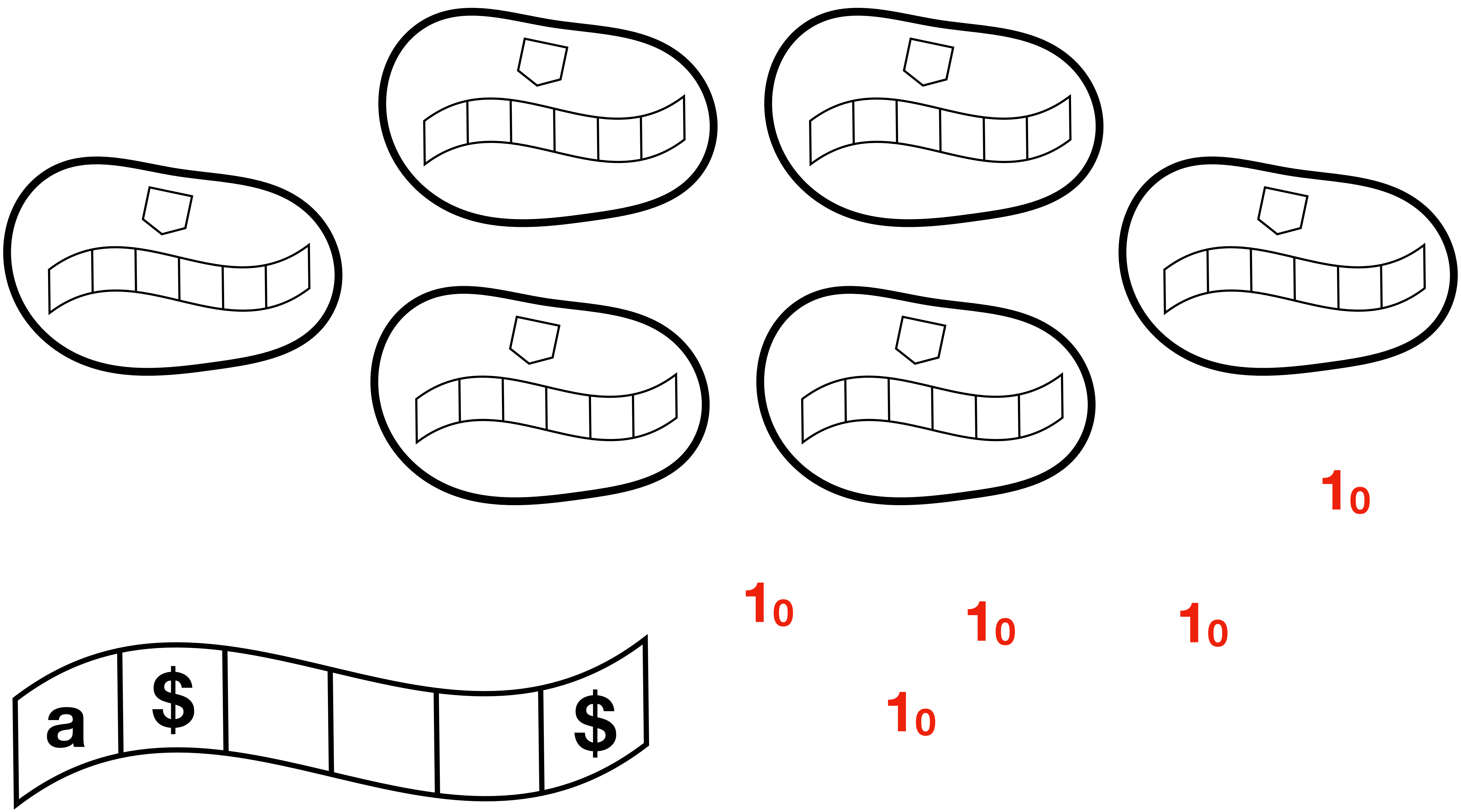
# Collecting the output



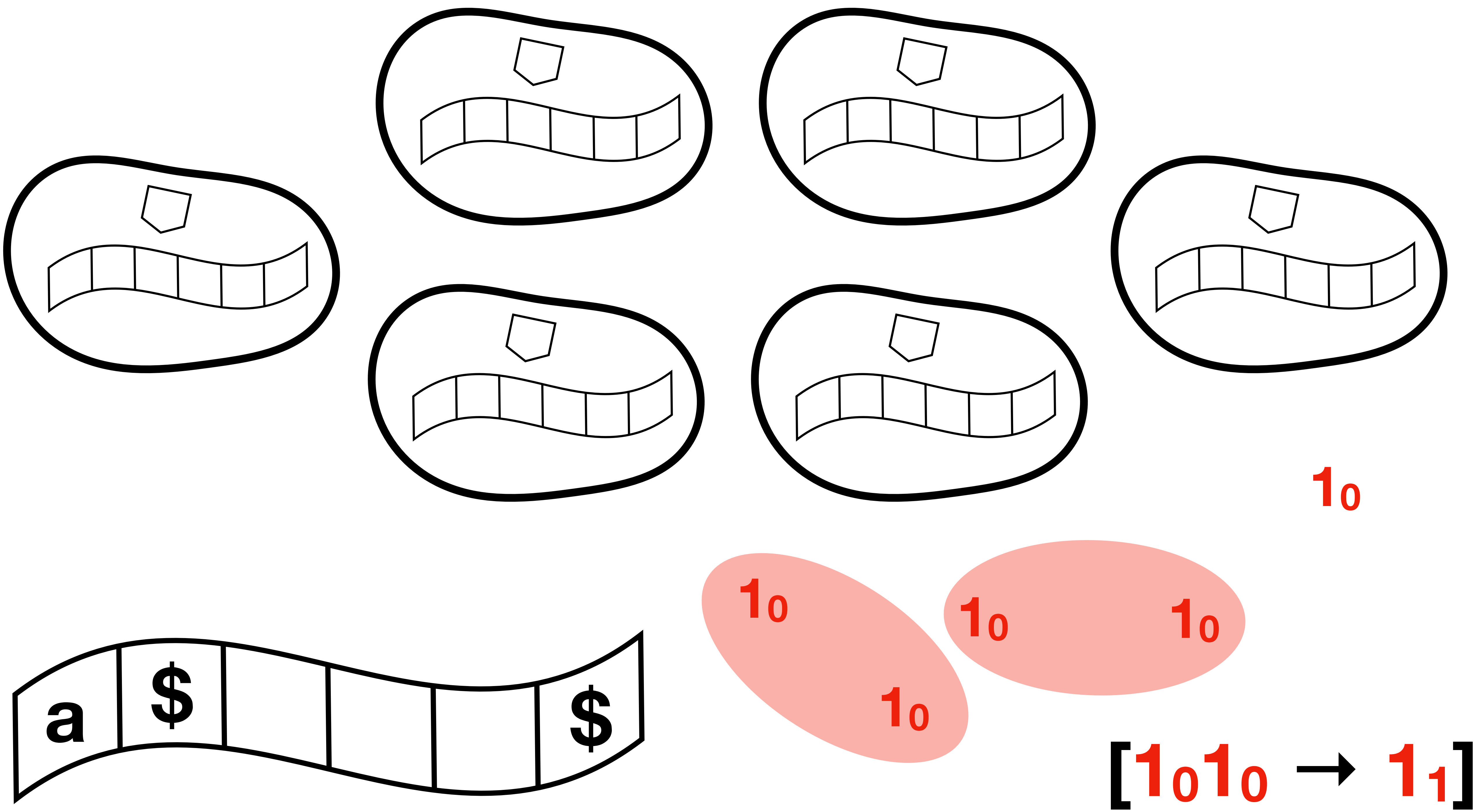
# Converting unary to binary



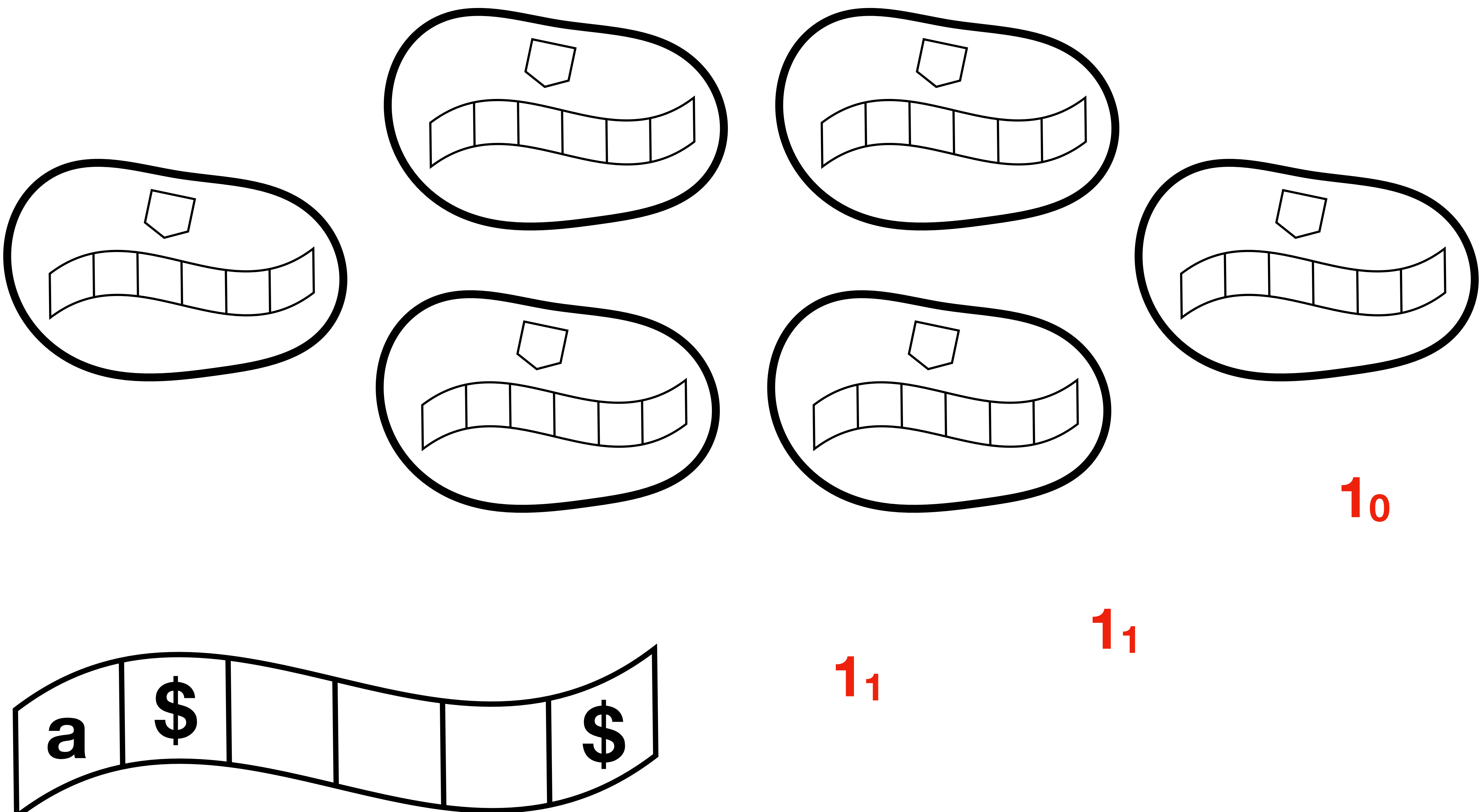
# Converting unary to binary



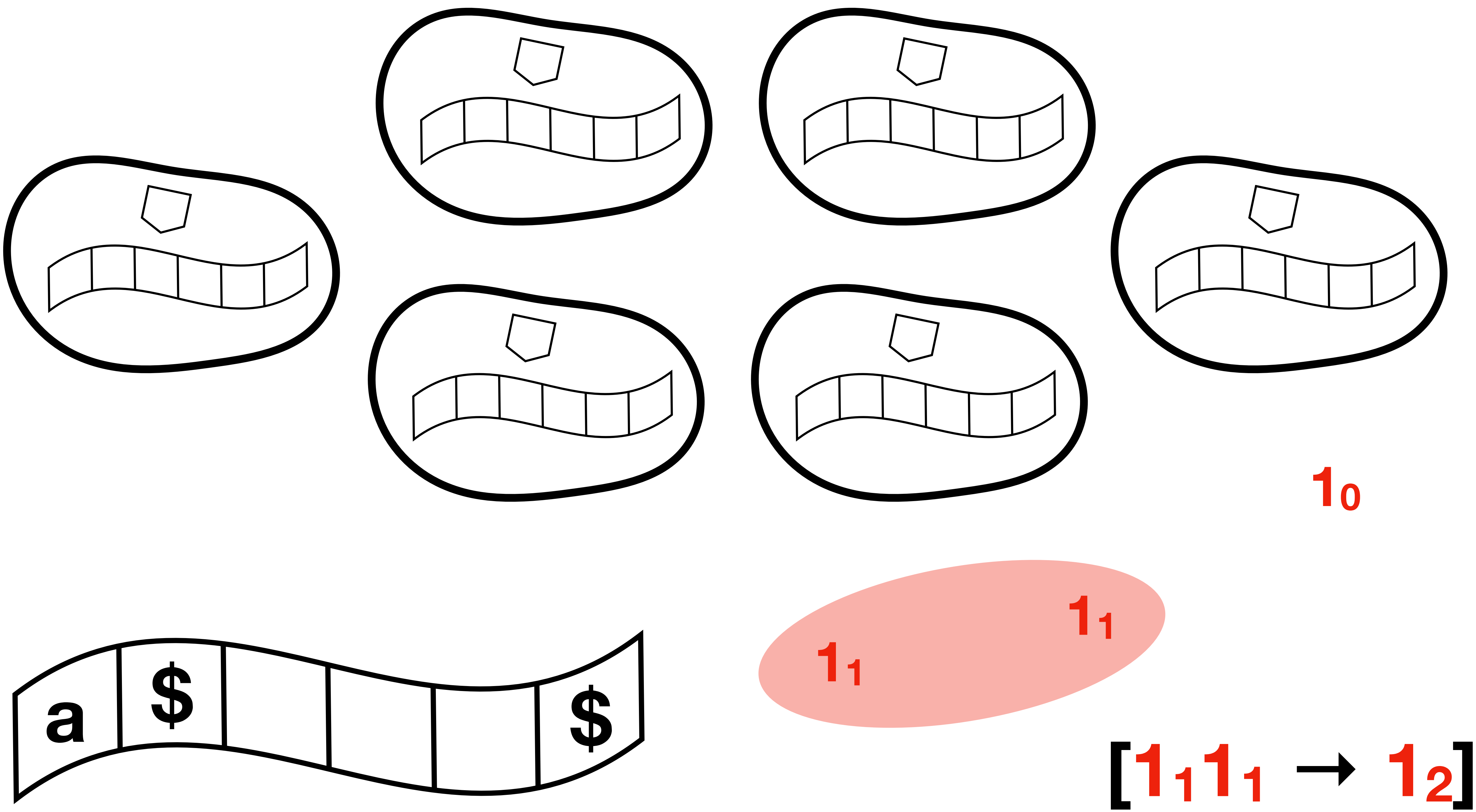
# Converting unary to binary



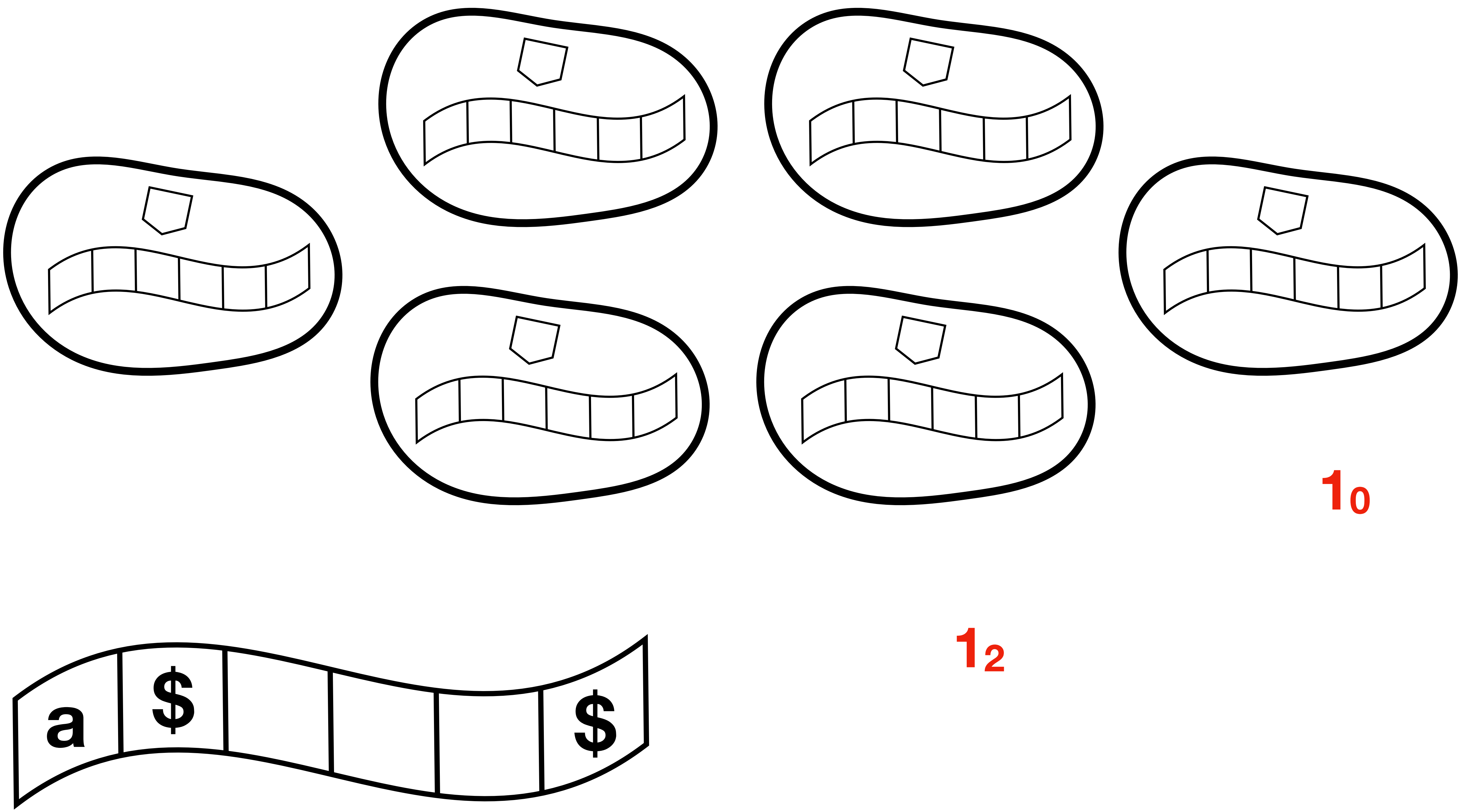
# Converting unary to binary



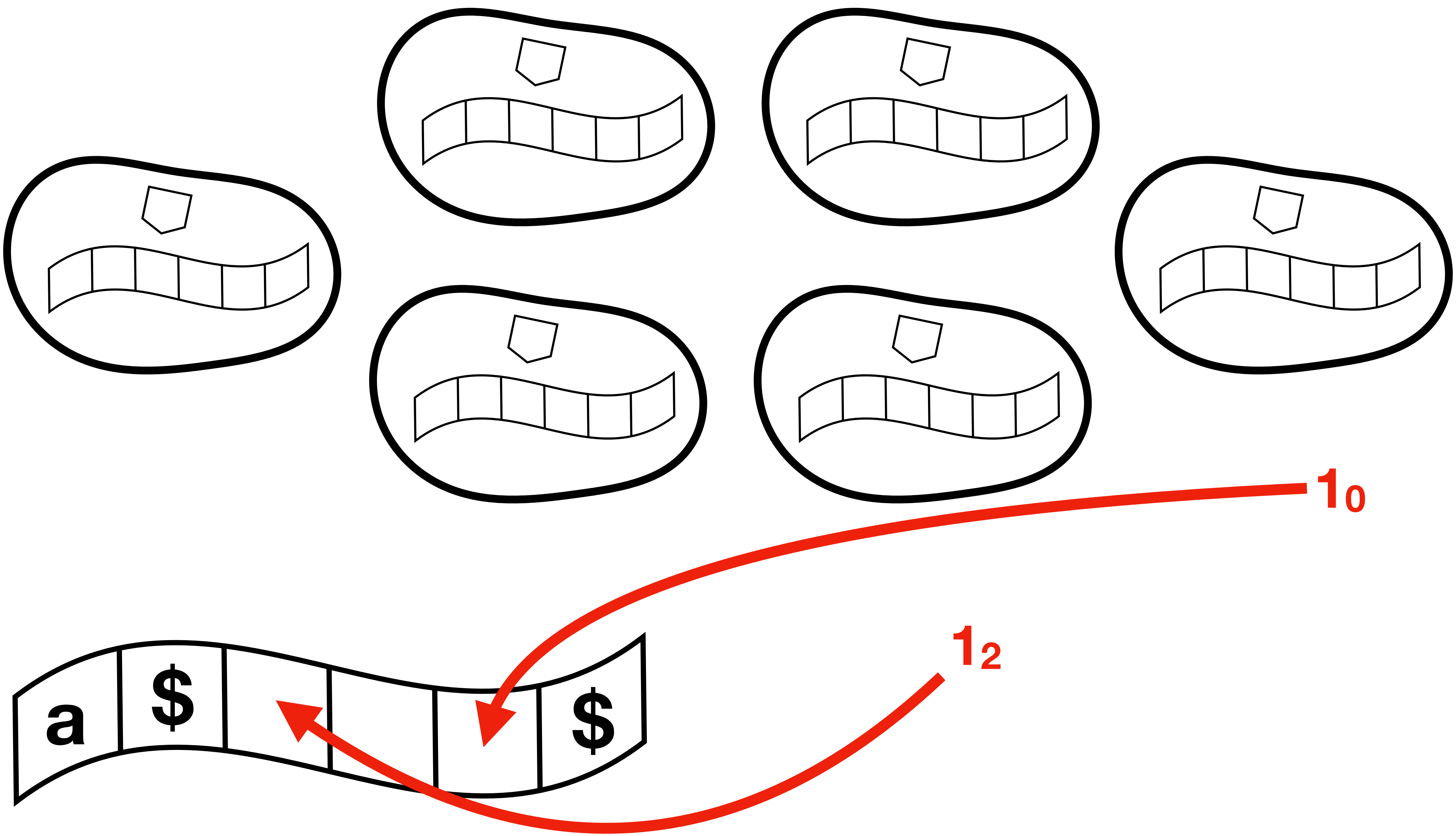
# Converting unary to binary



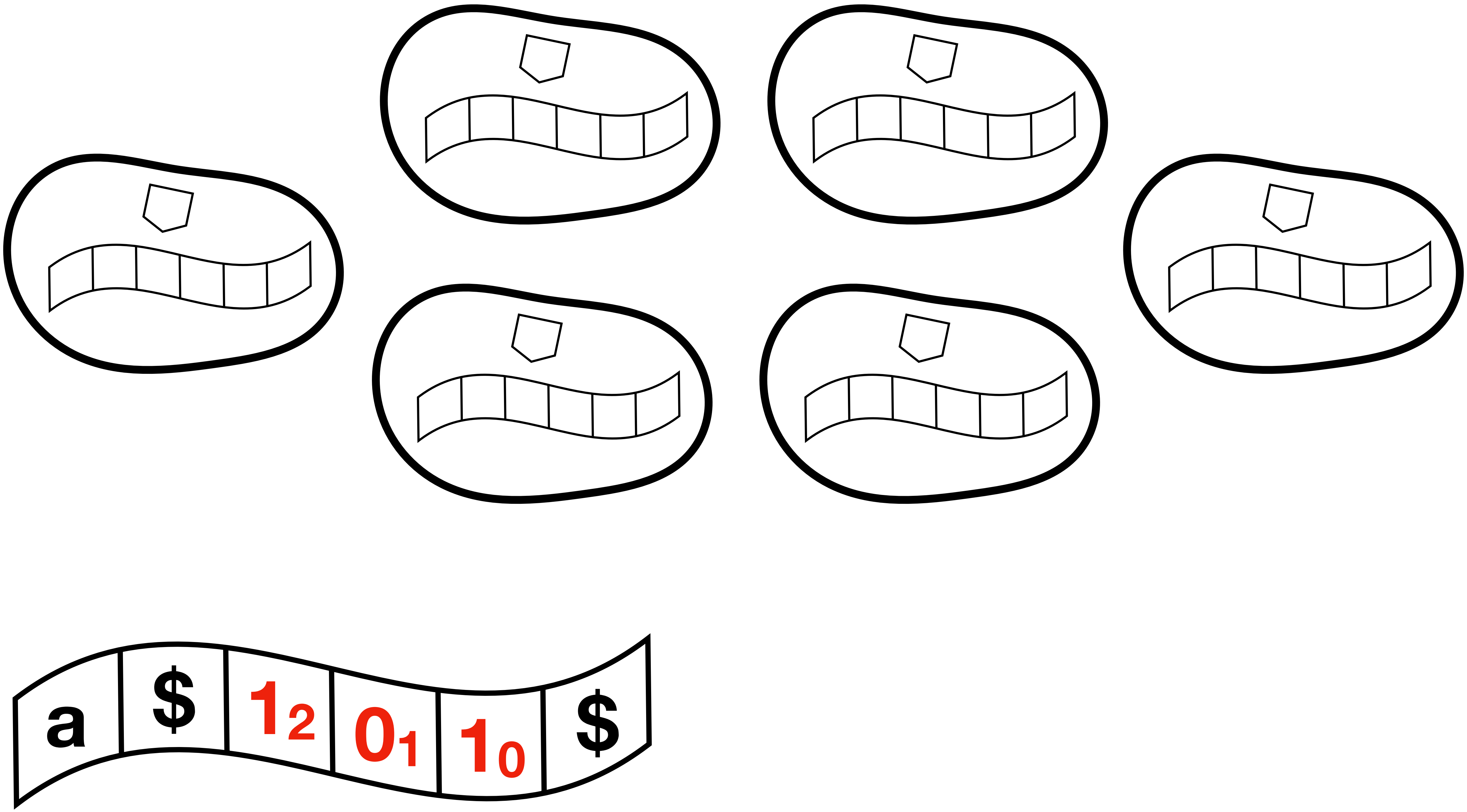
# Converting unary to binary



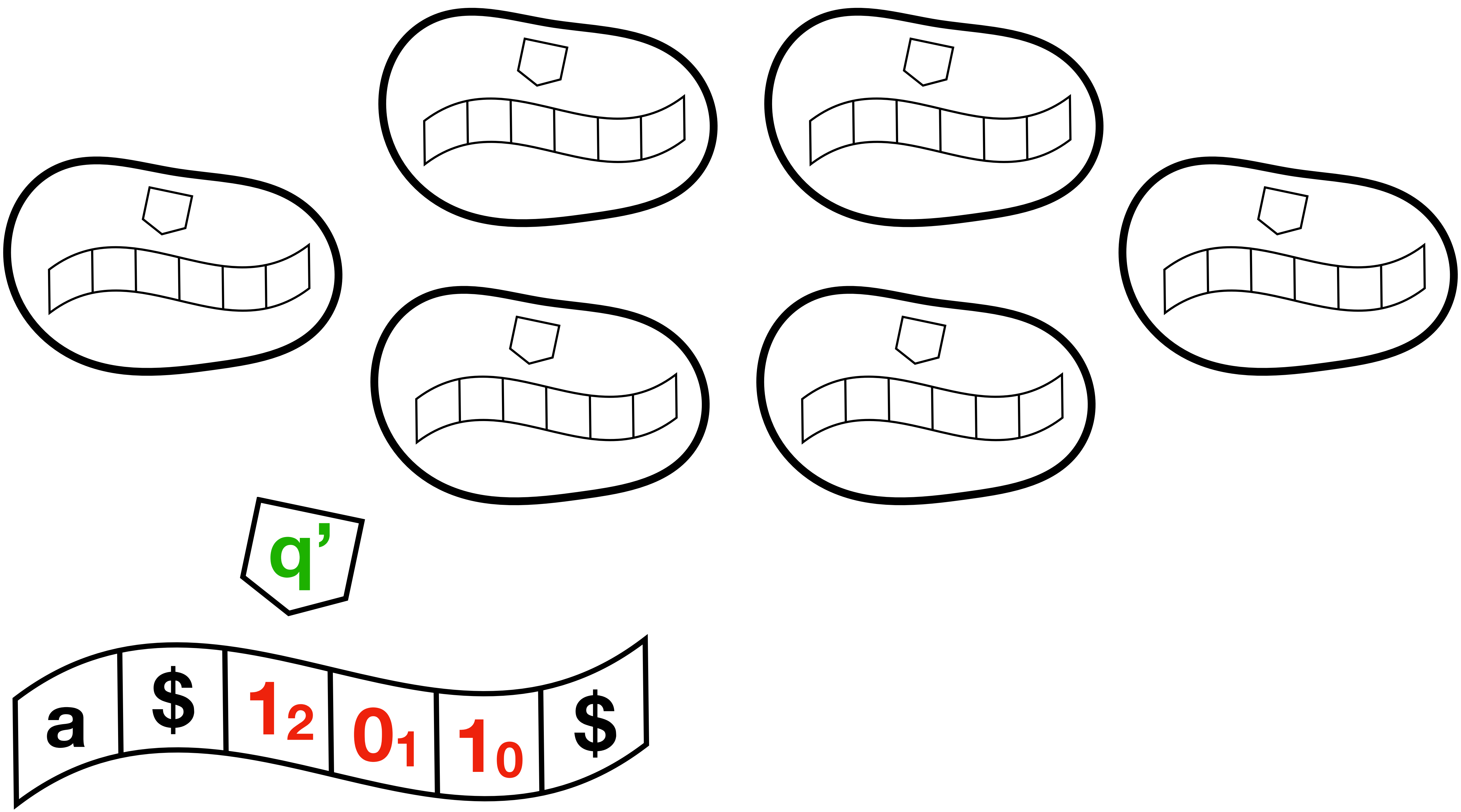
# Answer on the tape



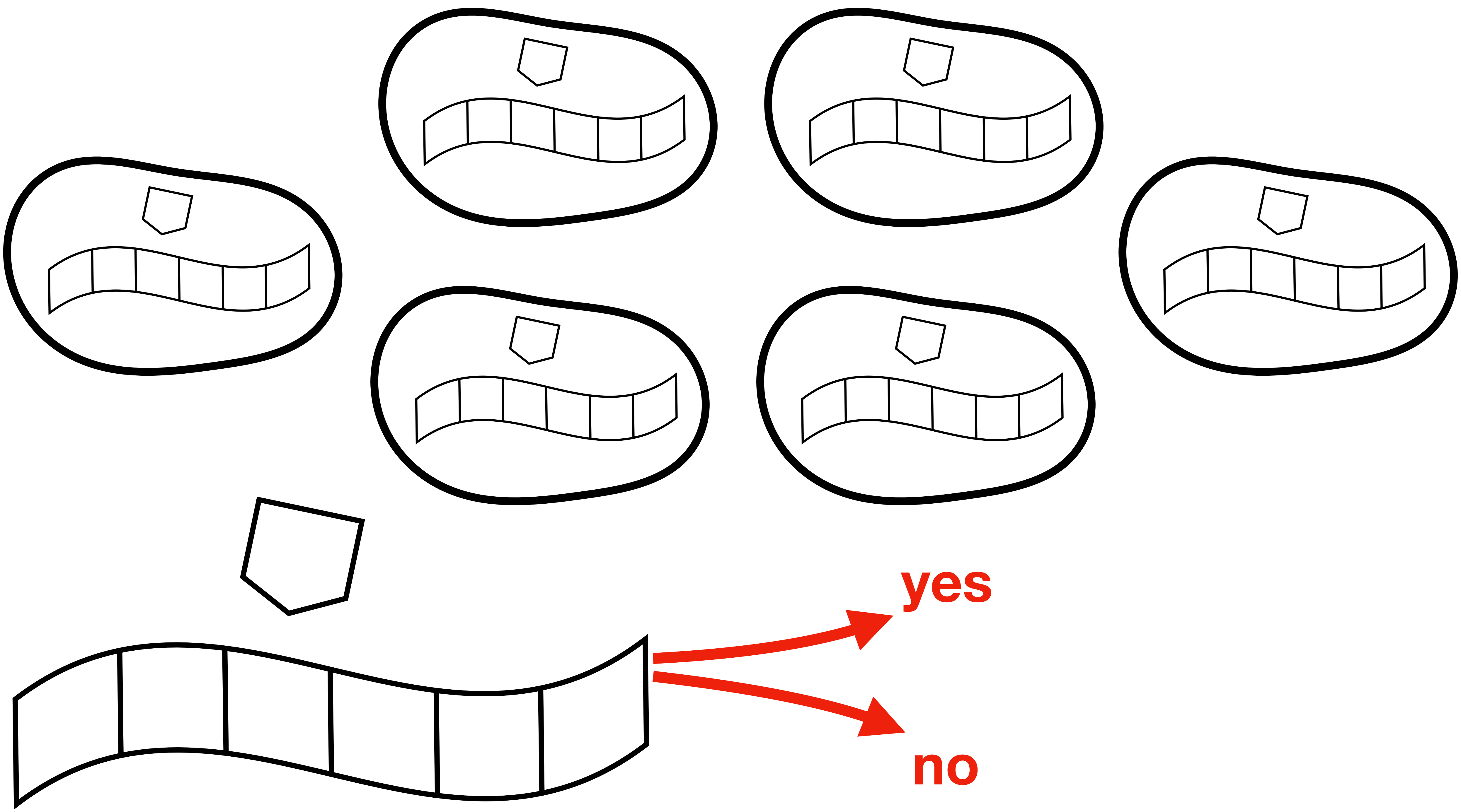
# Answer on the tape



# Resuming the simulation of the main TM



# Final answer



# Simulating (monodirectional, shallow) membrane systems is in $\mathbf{P}^{\#\mathbf{P}}$[1]

# Counting the number of objects a sent out by a membrane at time t is in #P

- **for** i := 0 **to** t **do**
    - deterministically choose a maximal multisite of rules to apply inside the simulated membrane
    - apply all the rules except membrane division in deterministic polynomial time (“Milano Theorem”)
    - if applying membrane division, nondeterministically choose whether to simulate the left or the right resulting membrane
- if an object a was sent out in the last step then **accept**, otherwise **reject**

# Lemma: $\mathbf{P}^{\#\mathbf{P}[1]}$ = parallel $\mathbf{P}^{\#\mathbf{P}}$

- Trivially $\mathbf{P}^{\#\mathbf{P}[1]} \subseteq$ parallel $\mathbf{P}^{\#\mathbf{P}}$
- A polynomial number of queries $f(x_1), \ldots, f(x_m)$ can be replaced by a single query to

$$g(x_1\$x_2\$\cdots\$x_m) = \sum_{i=1}^{n} B^i \times f(x_i) \qquad \Rightarrow \qquad f(x_i) = \left\lfloor \frac{g(x_1\$x_2\$\cdots\$x_m)}{B^{i-1}} \right\rfloor$$

- The function g is also in **#P** because this class is closed under sums and products

# Simulating (shallow, omnidirectional) membrane systems in $P^{\#P[1]}$

- for each membrane in the initial configuration, for each object type a and for each time step t, ask the oracle how many objects of type a are sent out by the membrane at time t
(note: polynomial number of parallel queries!)

- **while** the system has not produced the answer object **do**

    - simulate one step of the external environment deterministically (Milano Theorem)

    - add the objects sent out from the membranes (according to the queries asked) to the environment

- **accept** or **reject** according to the answer of the system simulated

# Computational complexity of membrane systems

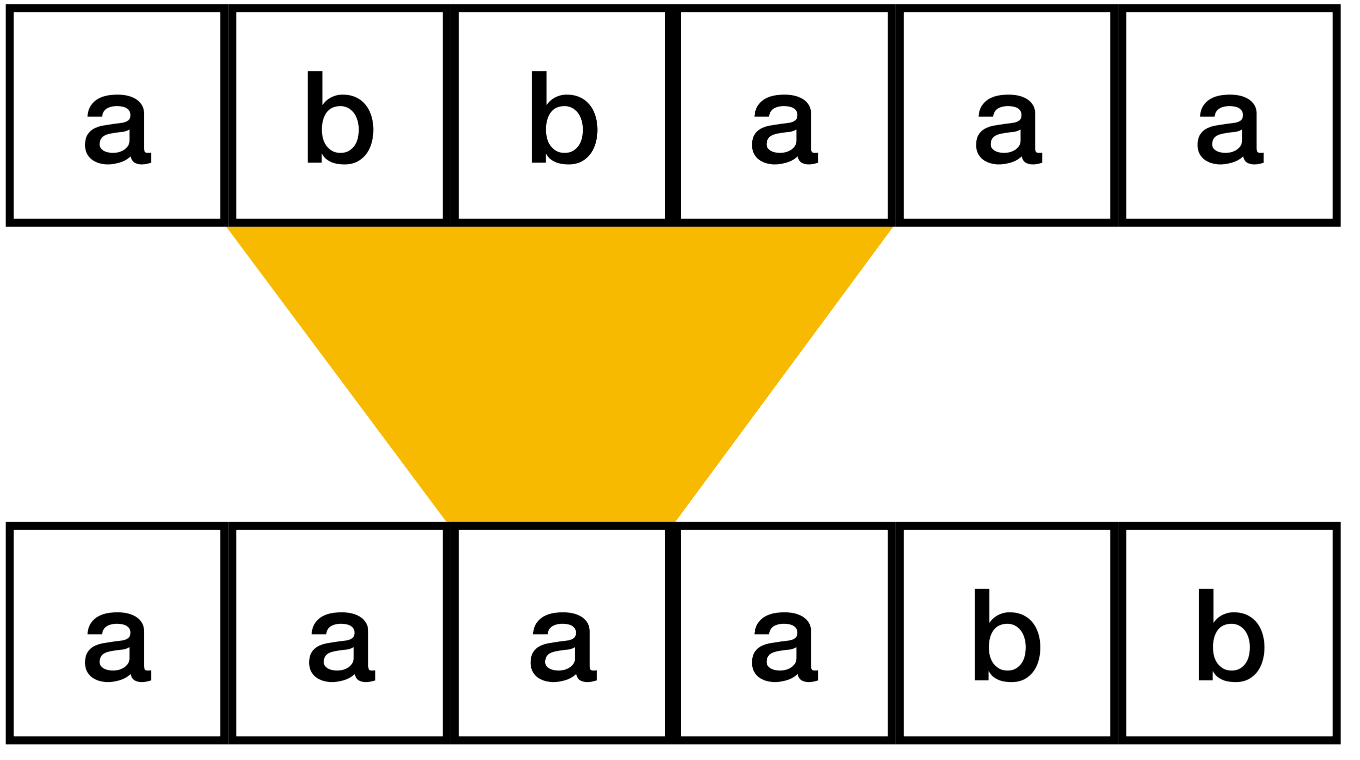
- No membranes (only environment) → **P**
- Shallow, monodirectional → $\mathbf{P}^{\mathbf{\#P}[1]}$ = parallel $\mathbf{P}^{\mathbf{\#P}}$
- Shallow, bidirectional → $\mathbf{P}^{\mathbf{\#P}}$
- Constant depth k, bidirectional → $\mathbf{P}^{\mathbf{C}_k\mathbf{P}}$
  where $\mathbf{C_0P} = \mathbf{P}$, $\mathbf{C_1P} = \mathbf{PP}$, $\mathbf{C_2P} = \mathbf{PP}^{\mathbf{PP}}$, $\mathbf{C_kP} = \mathbf{PP}^{\mathbf{C}_{k-1}\mathbf{P}}$
  is the counting hierarchy [work in progress]
- Unbounded depth, bidirectional → **PSPACE**

# Expanding cellular automata (XCA)

# Expanding CA

| | | | | | |
|---|---|---|---|---|---|
| a | b | b | a | a | a |

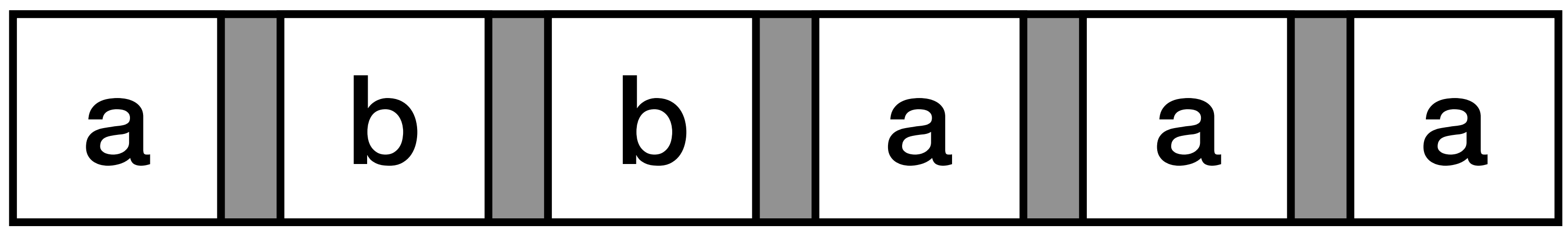
# Expanding CA



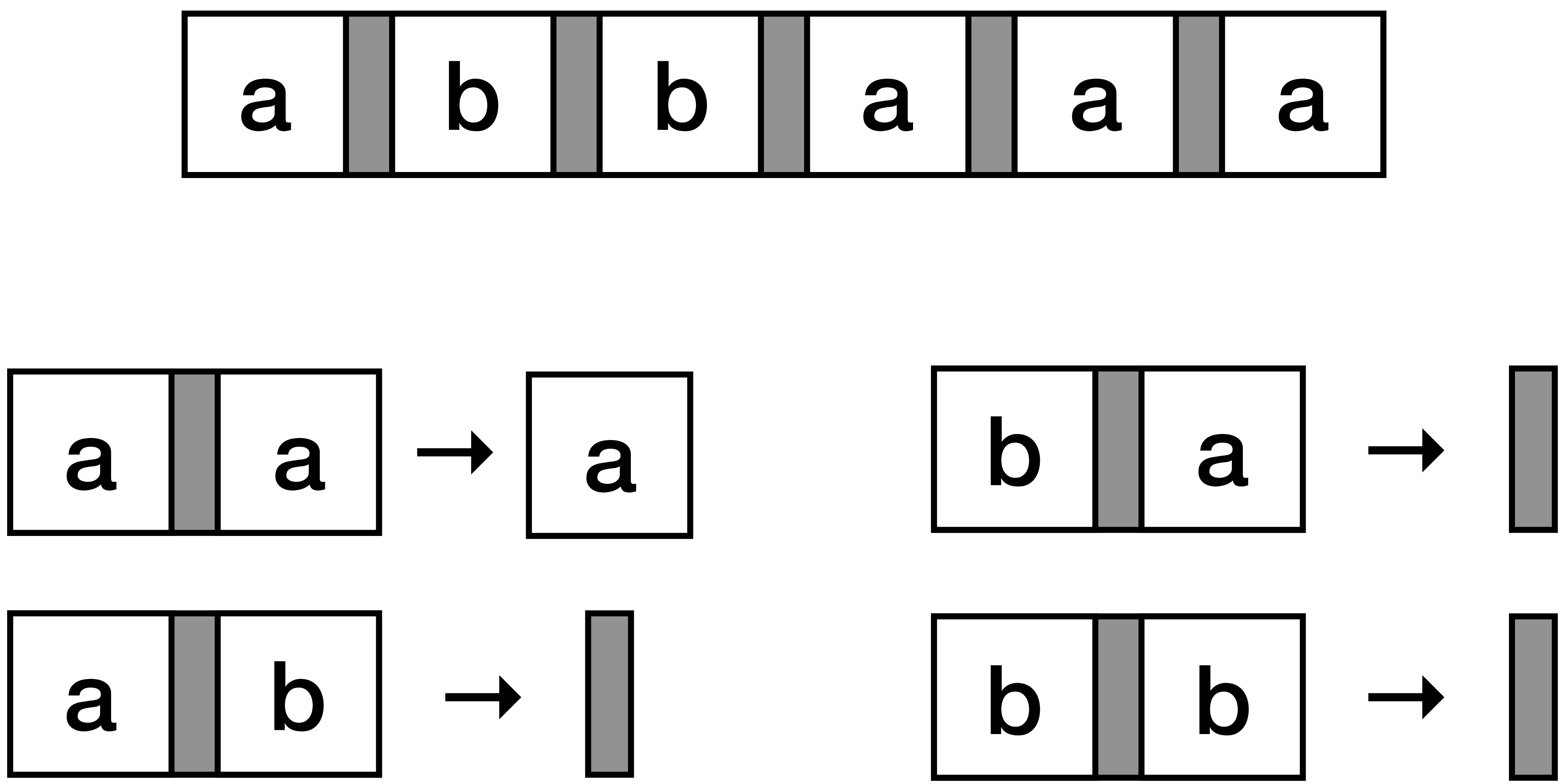
# Expanding CA

| | | | | | |
|---|---|---|---|---|---|
| a | b | b | a | a | a |

# Expanding CA



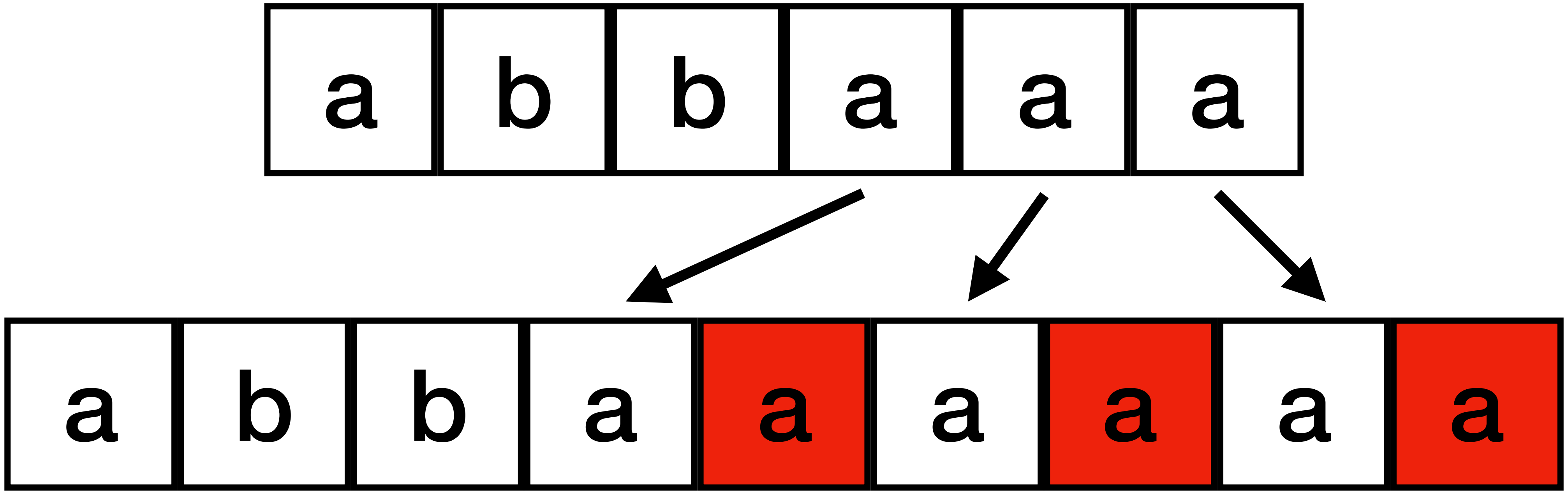
# Expanding CA



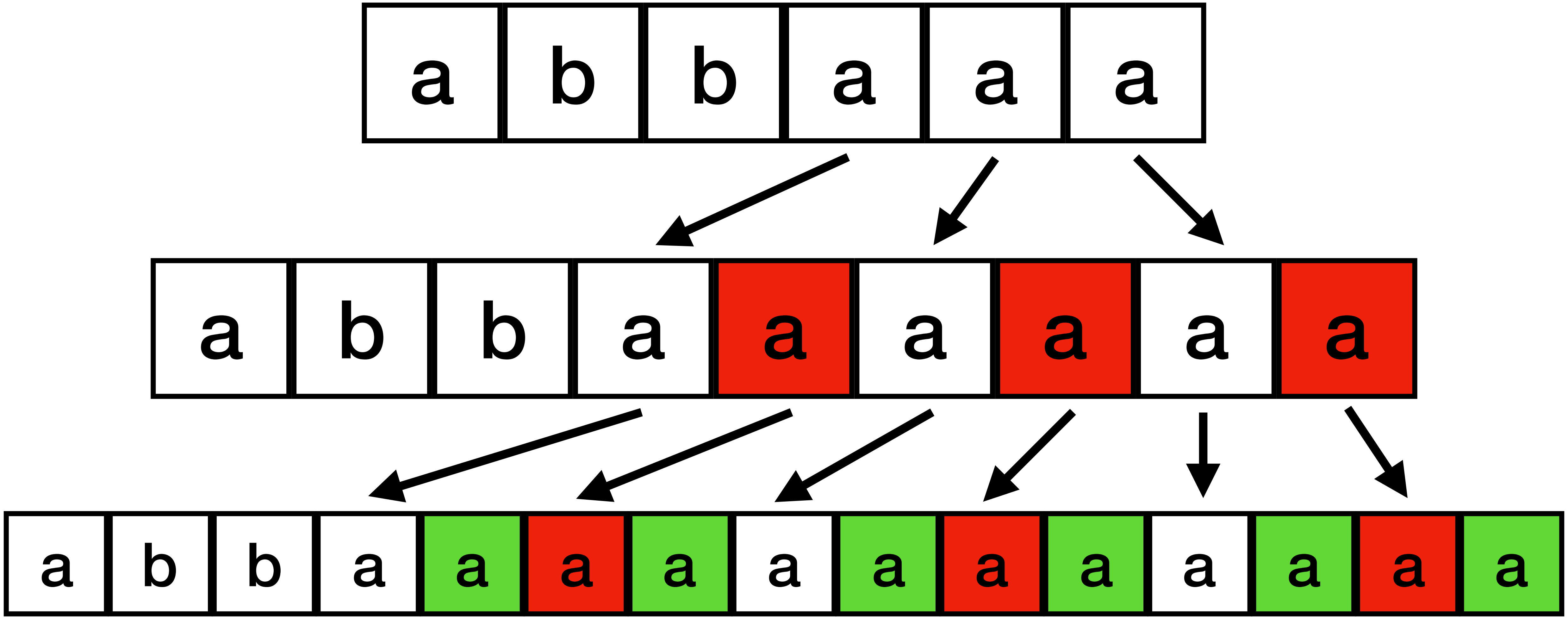
# Expanding CA

| | | | | | |
|---|---|---|---|---|---|
| a | b | b | a | a | a |

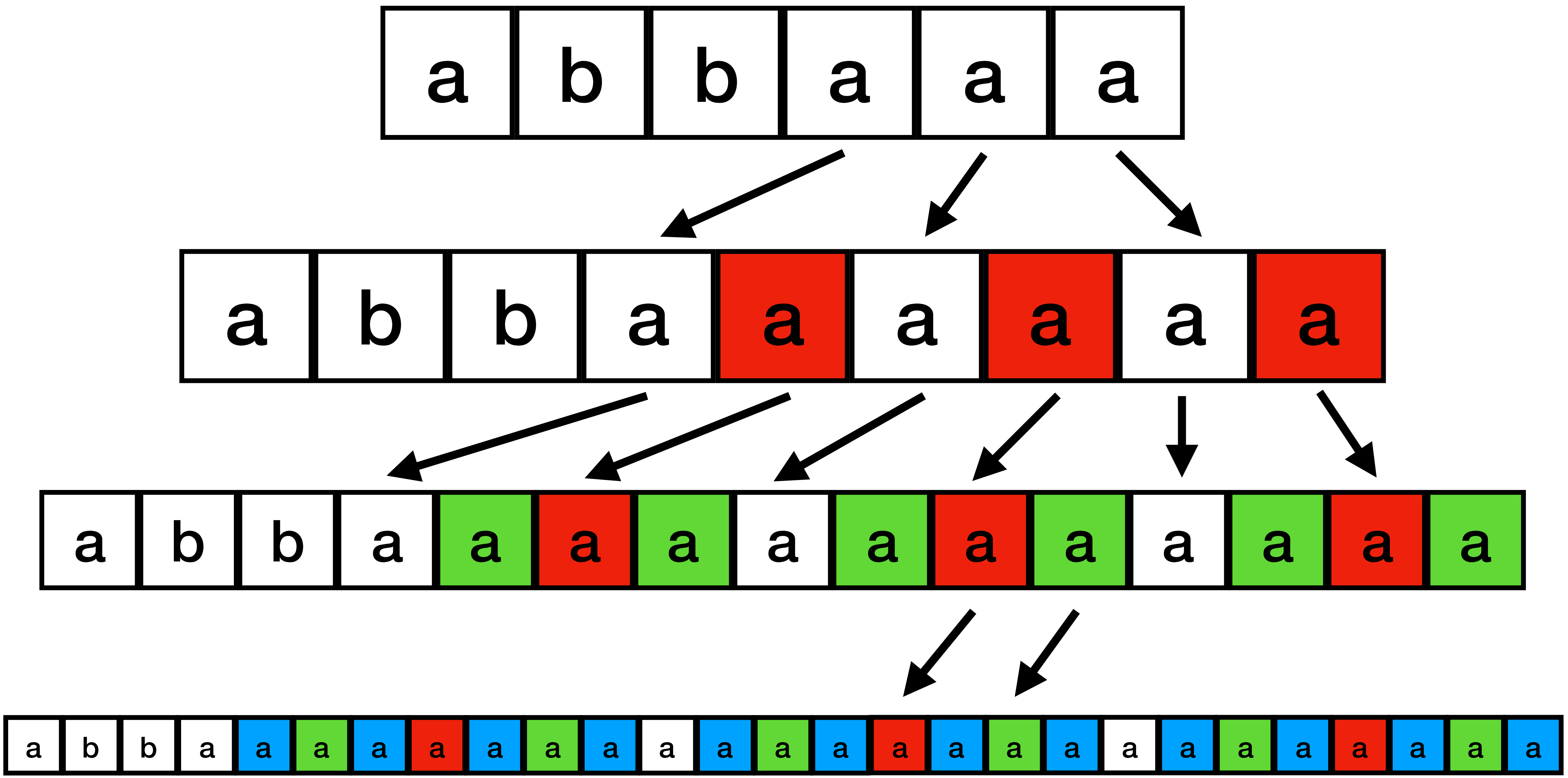
# Expanding CA



# Expanding CA



# Expanding CA



# Complexity results on XCA

- The class of problems solved in polynomial time by XCA is exactly the class of problems truth-table reducible to **NP**

- If shrinking (deleting cells) is also allowed, then the class becomes **PSPACE**

# Conclusions and future work

# Summary of results

- A lot of parallels computing models characterise either **P** or **PSPACE** when working in polynomial time
- Some variants of membrane systems characterise more “exotic” complexity classes with oracles, like $\mathbf{P}^{\mathbf{\#P}[1]}$, $\mathbf{P}^{\mathbf{\#P}}$, $\mathbf{P}^{\mathbf{NP}}$
- Expanding CA characterise the class of problems truth-table reducible to **NP**, which is somehow similar to oracle complexity classes

# Conjectures and future work

- Find out why these models happen to characterise these exotic complexity classes
- Find out how the topology of the parallel computing units influences the efficiency:
  - Trees or stars for membrane systems
  - Linear or Euclidean grid for CA
  - Linear but expanding for XCA

# References


- Sosík, P. and Rodríguez-Patón, A., 2007. Membrane computing and complexity theory: A characterization of PSPACE. Journal of Computer and System Sciences, 73(1), pp. 137–152

- Leporati, A., Manzoni, L., Mauri, G., Porreca, A.E. and Zandron, C., 2017. Characterising the complexity of tissue P systems with fission rules. Journal of Computer and System Sciences, 90, pp. 115–128

- Leporati, A., Manzoni, L., Mauri, G., Porreca, A.E. and Zandron, C., 2016. Monodirectional P systems. *Natural Computing*, *15*(4), pp. 551–564

- Modanese, A., 2019. Complexity-theoretic aspects of expanding cellular automata. arXiv preprint arXiv:1902.05487 (accepted at AUTOMATA 2019)

Thanks for your attention!

Merci de votre attention !