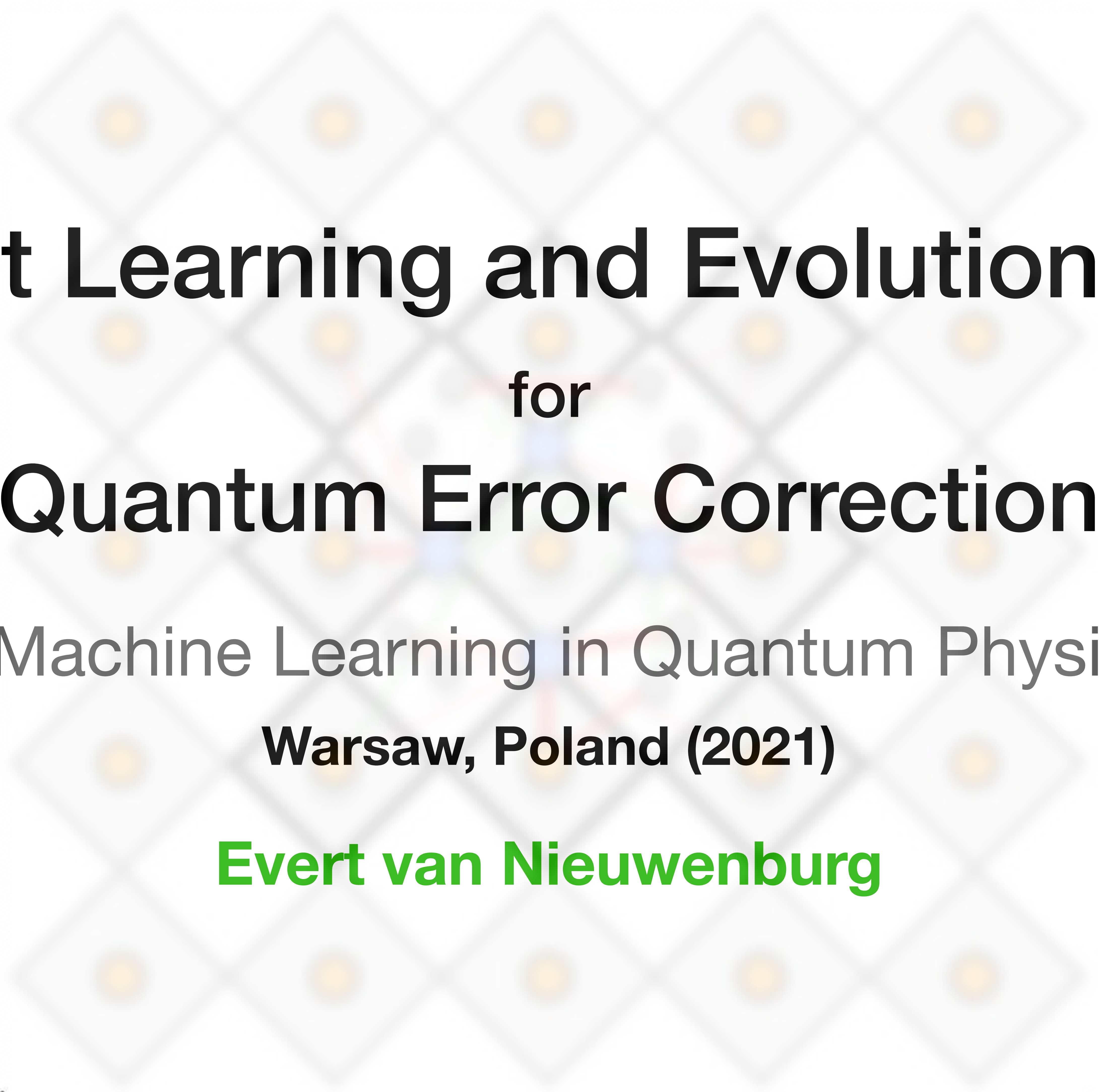


Reinforcement Learning and Evolutionary Strategies

for

Quantum Error Correction

Summer School: Machine Learning in Quantum Physics and Chemistry

Warsaw, Poland (2021)

Evert van Nieuwenburg

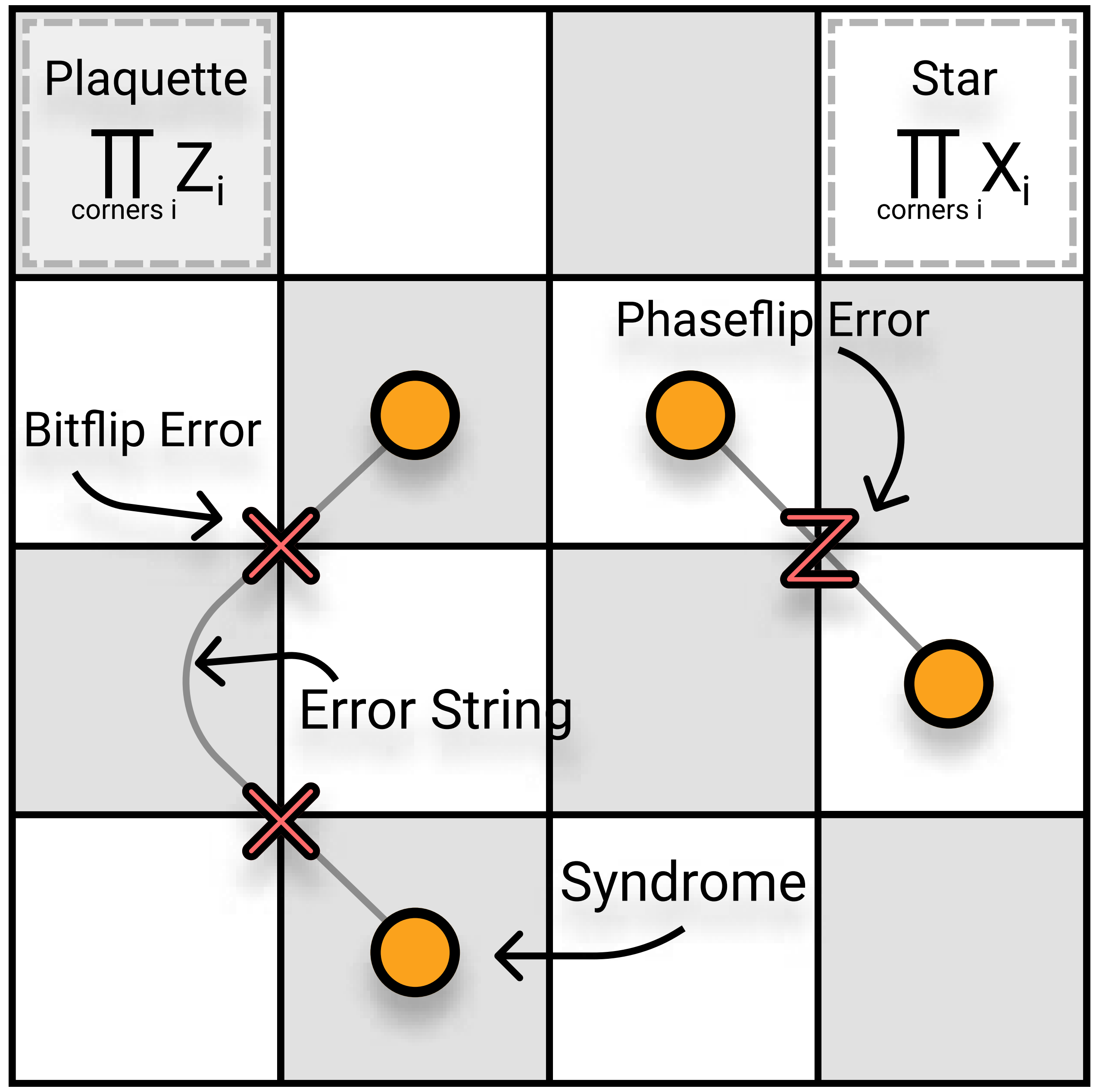
The Niels Bohr International Academy

European Commission | Horizon 2020 European Union funding for Research & Innovation

There are three main concepts for this talk

Don't hesitate to ask!

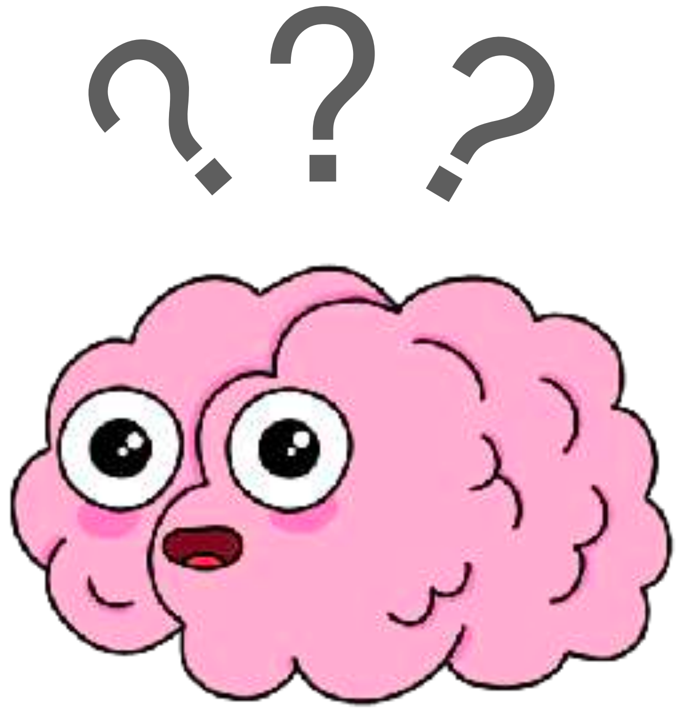
Stabilizer codes



Plaquette
$\prod_{\text{corners } i} Z_i$
Star
$\prod_{\text{corners } i} X_i$
Bitflip Error
Phaseflip Error
Error String
Syndrome

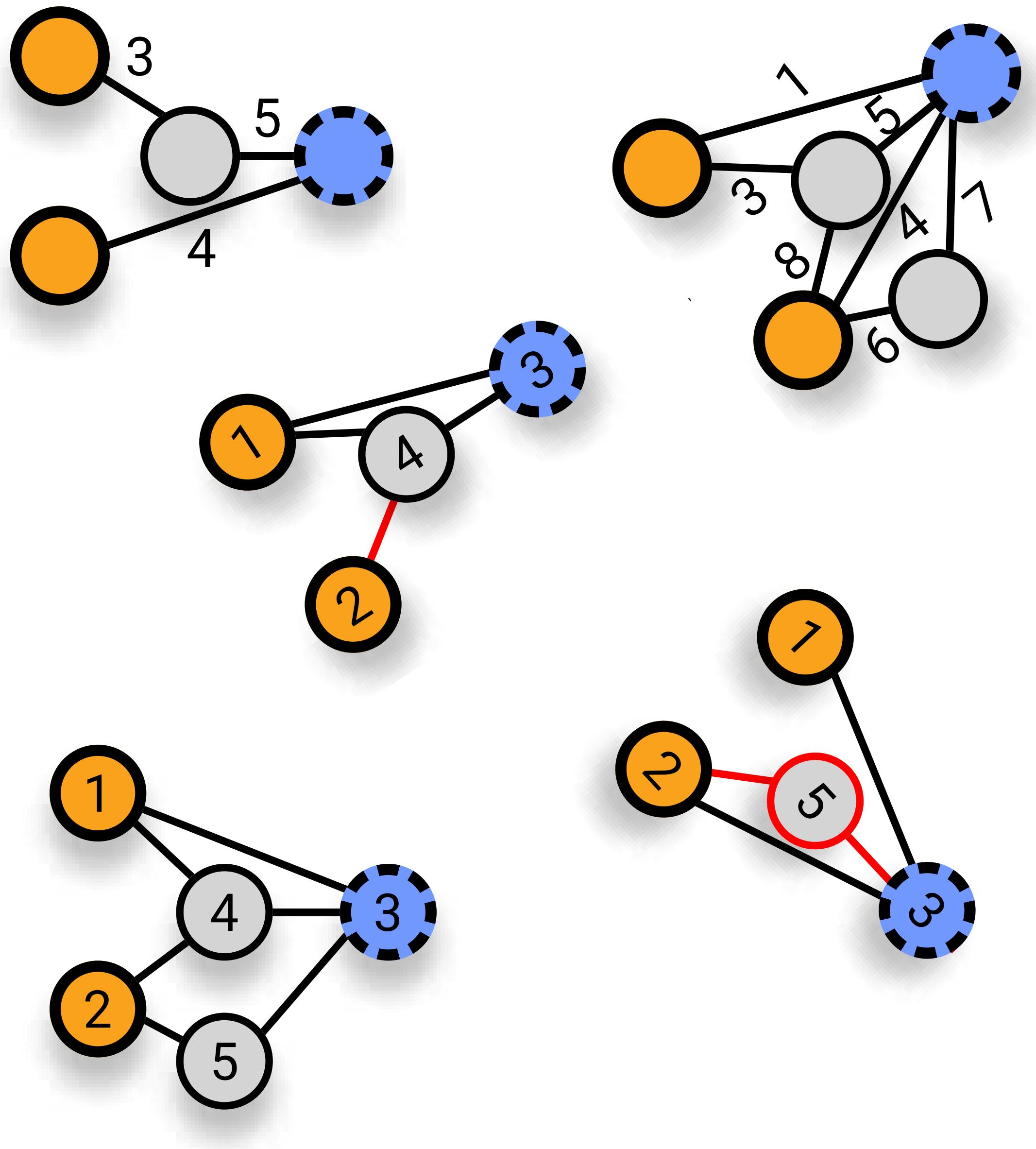
Quantum Computation

Reinforcement Learning



Deep Q-Network

Evolutionary Strategy

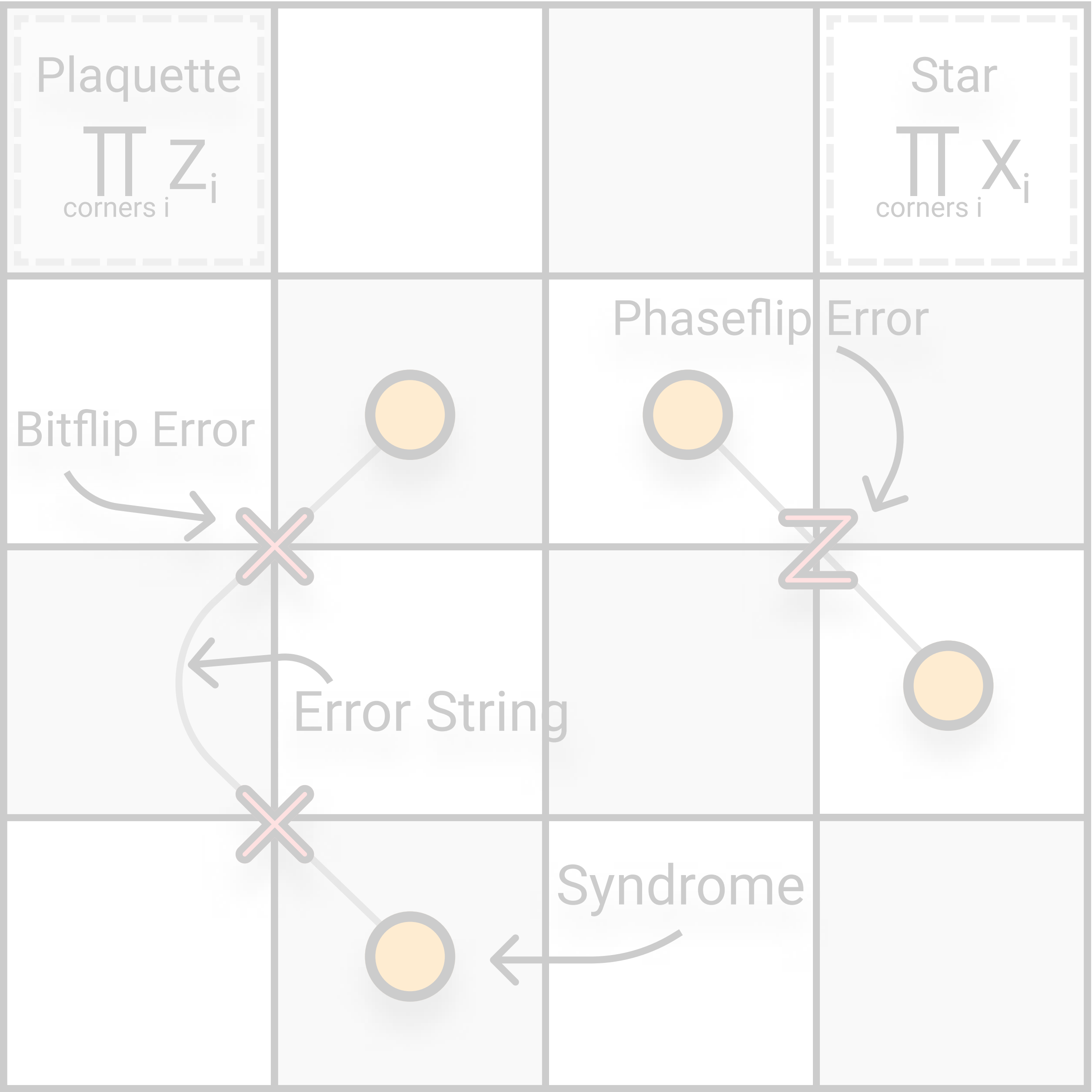


Policy Networks

There are three main concepts for this talk

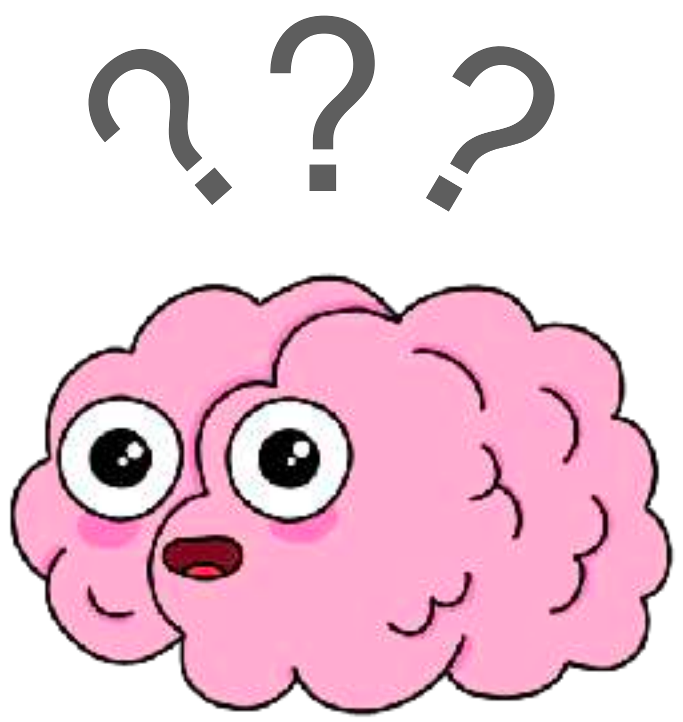
Don’t hesitate to ask!

Stabilizer codes


Plaquette
$\prod_{\text{corners } i} Z_i$
Star
$\prod_{\text{corners } i} X_i$
Phaseflip Error
Bitflip Error
Error String
Syndrome

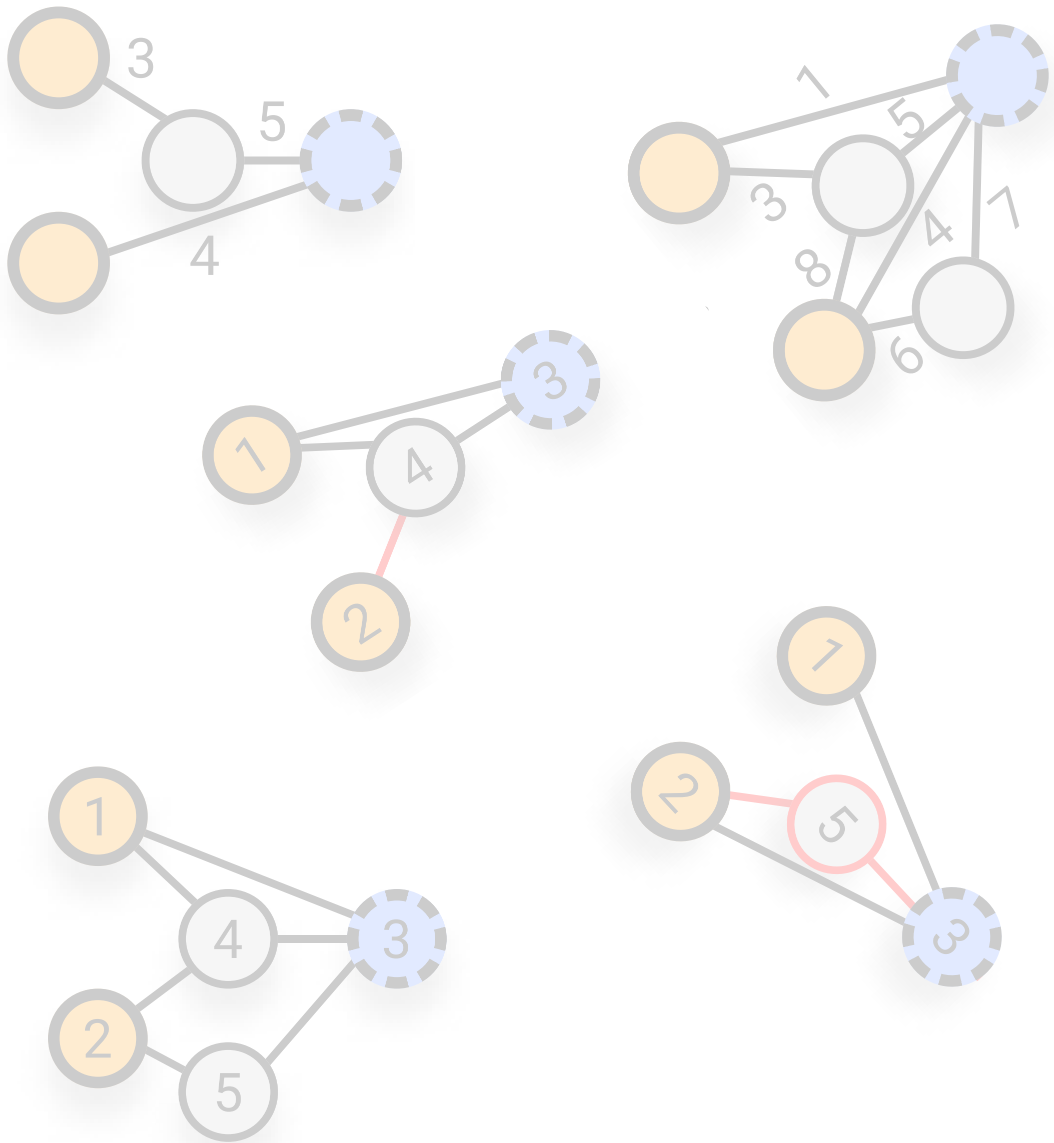
Quantum Computation

Reinforcement Learning



Deep Q-Network

Evolutionary Strategy



Policy Networks

Reinforcement learning is fun

Multi-Agent Hide & Seek

https://openai.com/blog/emergent-tool-use/

Locomotion

deepmind.com/blog/article/producing-flexible-behaviours-simulated-environments

Reinforcement learning is fun

Multi-Agent Hide & Seek

Locomotion

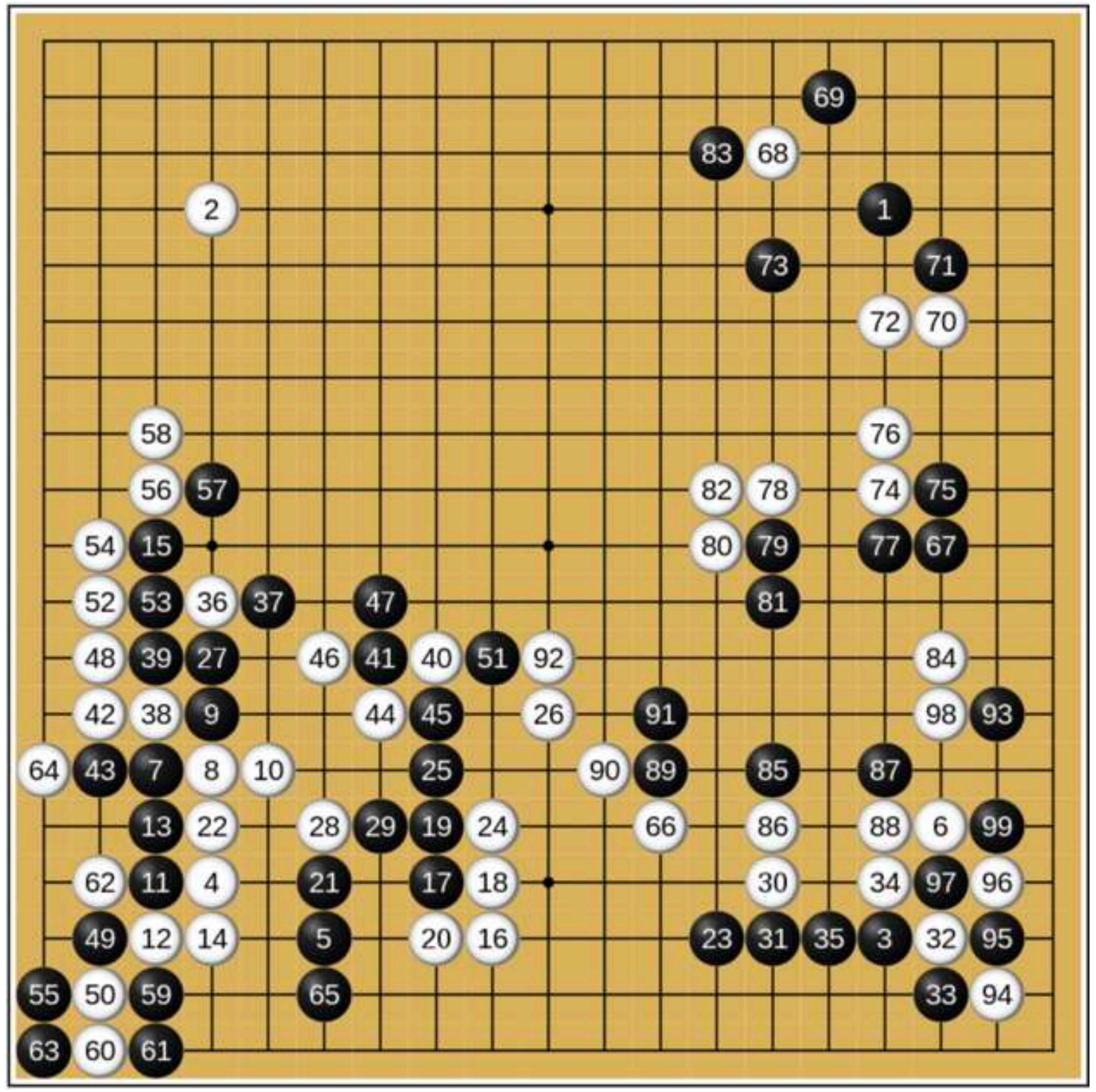
https://openai.com/blog/emerg

-flexible-behaviours-simulated-environments

https://deepmind.com/blog/article/generally-capable-agents-emerge-from-open-ended-play

Computers can learn to play games

Reinforcement learning is pretty good at most games!

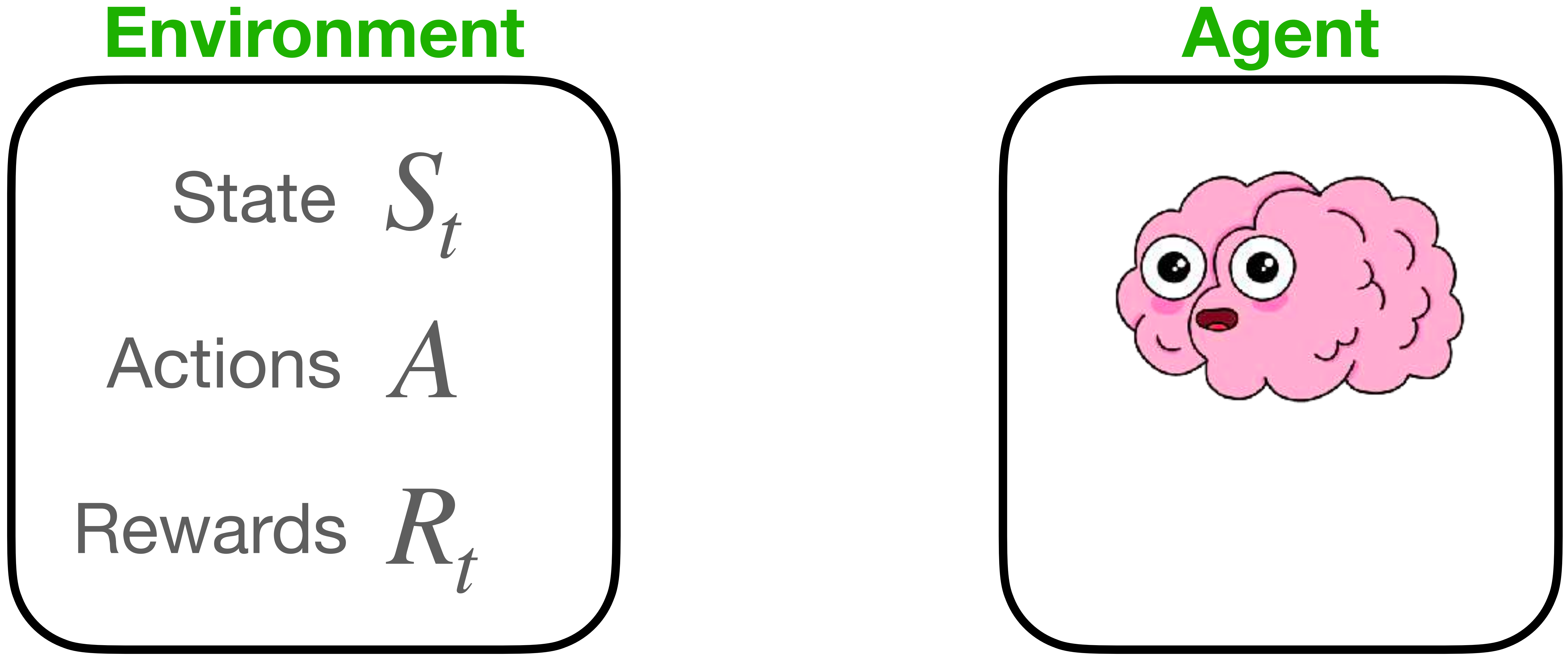


https://en.wikipedia.org/wiki/AlphaGo

https://deepmind.com/blog/article/Agent57-Outperforming-the-human-Atari-benchmark

Reinforcement learning in a nutshell

Intuitive: learning from trial and error



Environment

State S_t

Actions A

Rewards R_t

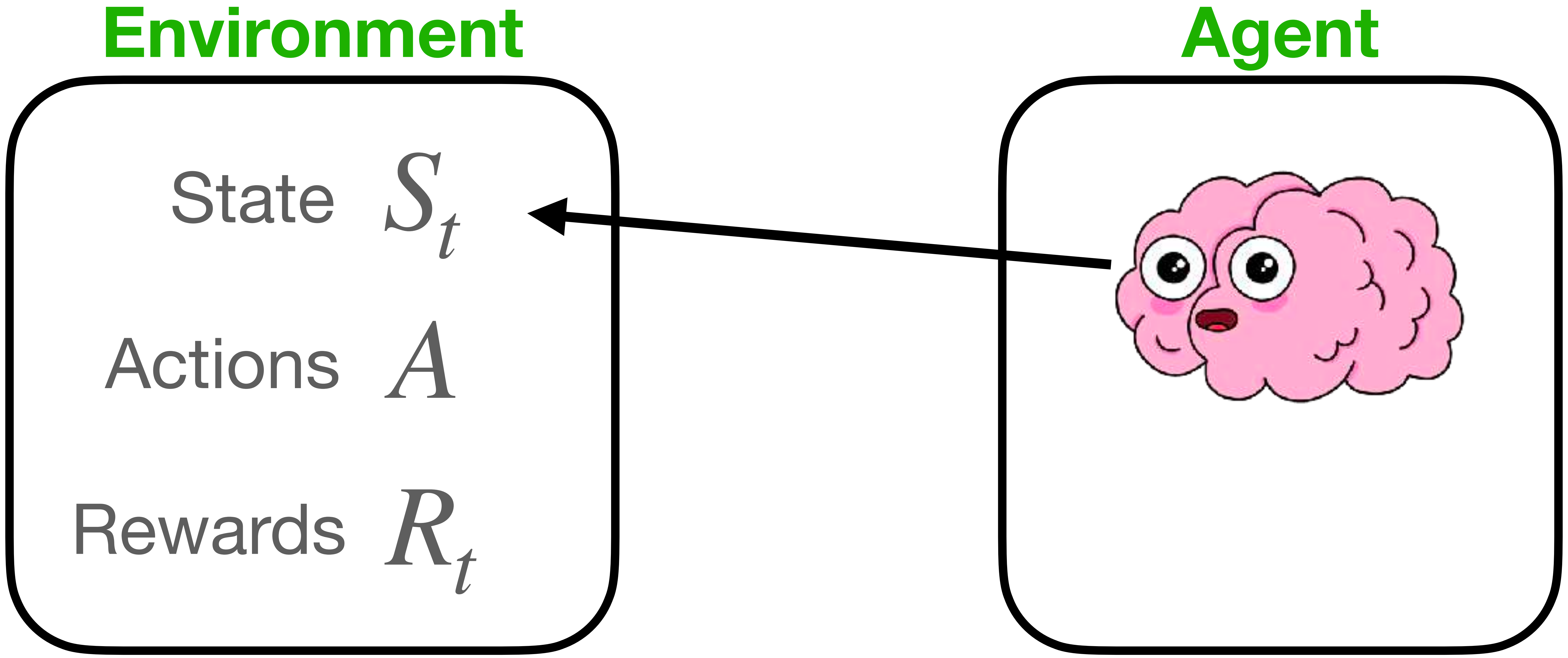
Agent

Sutton & Barto

http://www.incompleteideas.net/book/the-book.html

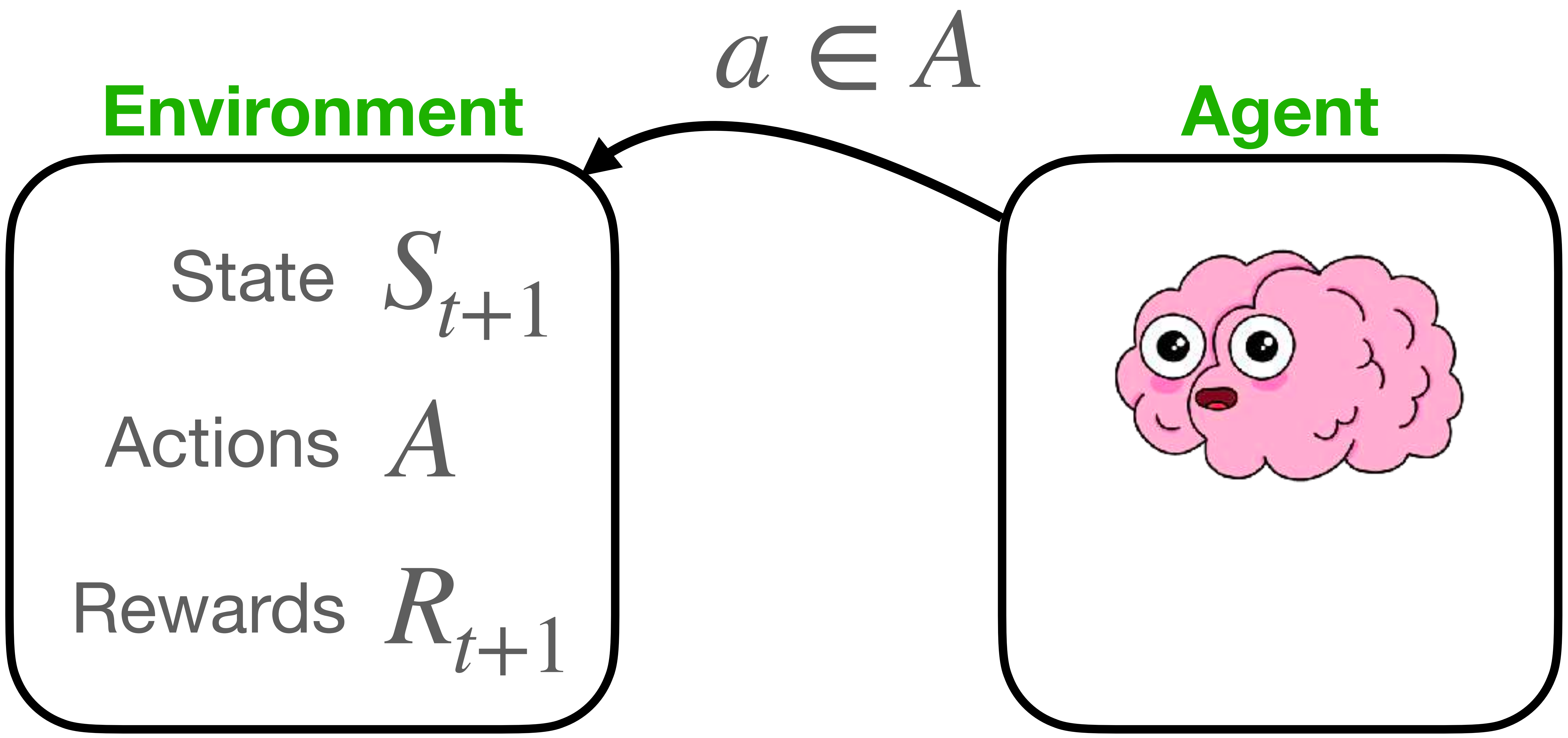
The agent can observe the (state of the) environment

If not, the environment is *partially* observable



The agent can act on the environment

The environment defines which actions are possible



$a \in A$

Environment

State S_{t+1}

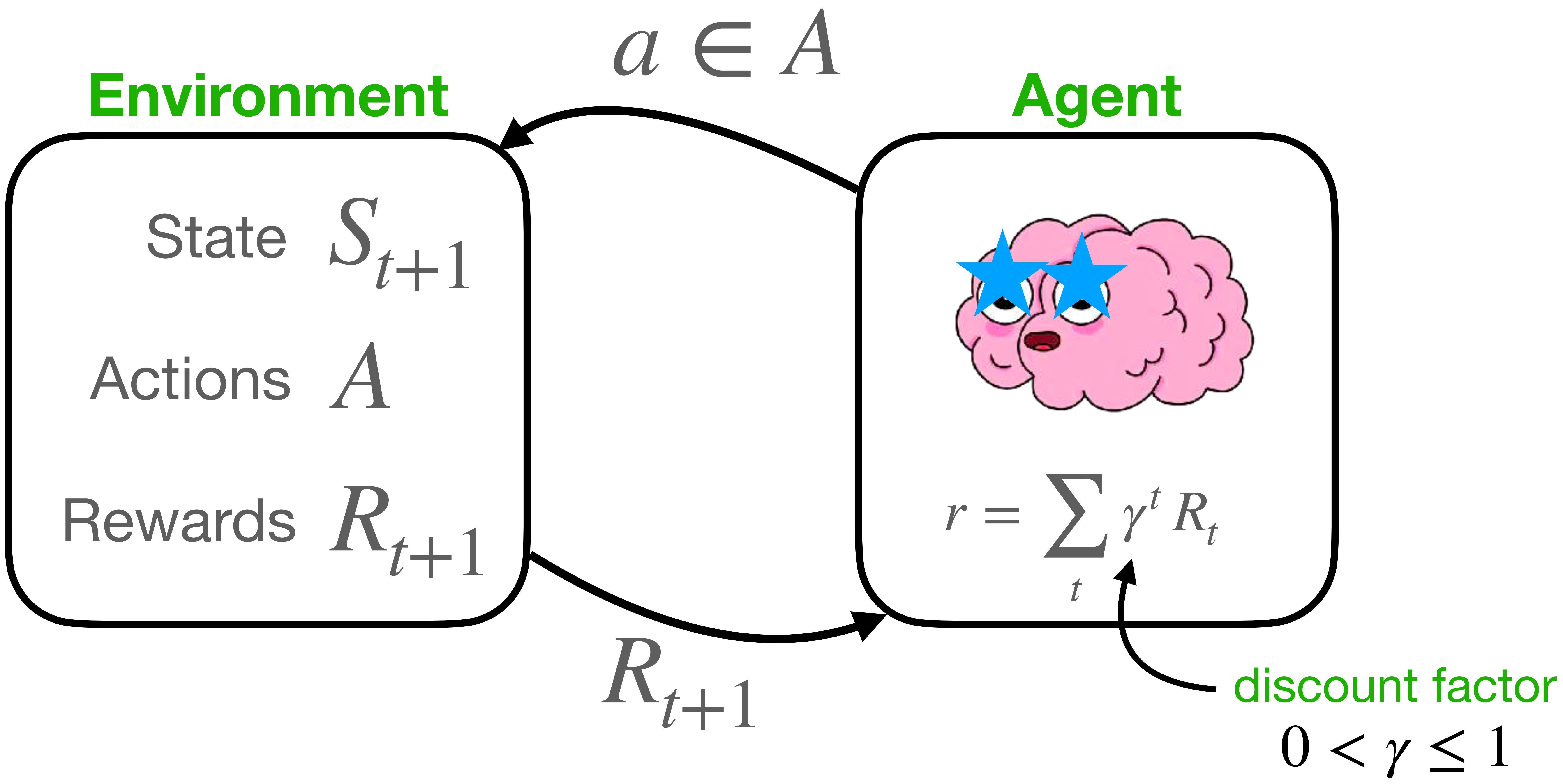
Actions A

Rewards R_{t+1}

Agent

The agent receives a reward

A discount factor regulates near- vs long-term rewards



$a \in A$

Environment

State S_{t+1}

Actions A

Rewards R_{t+1}

Agent

$r = \sum_t \gamma^t R_t$

R_{t+1}

discount factor

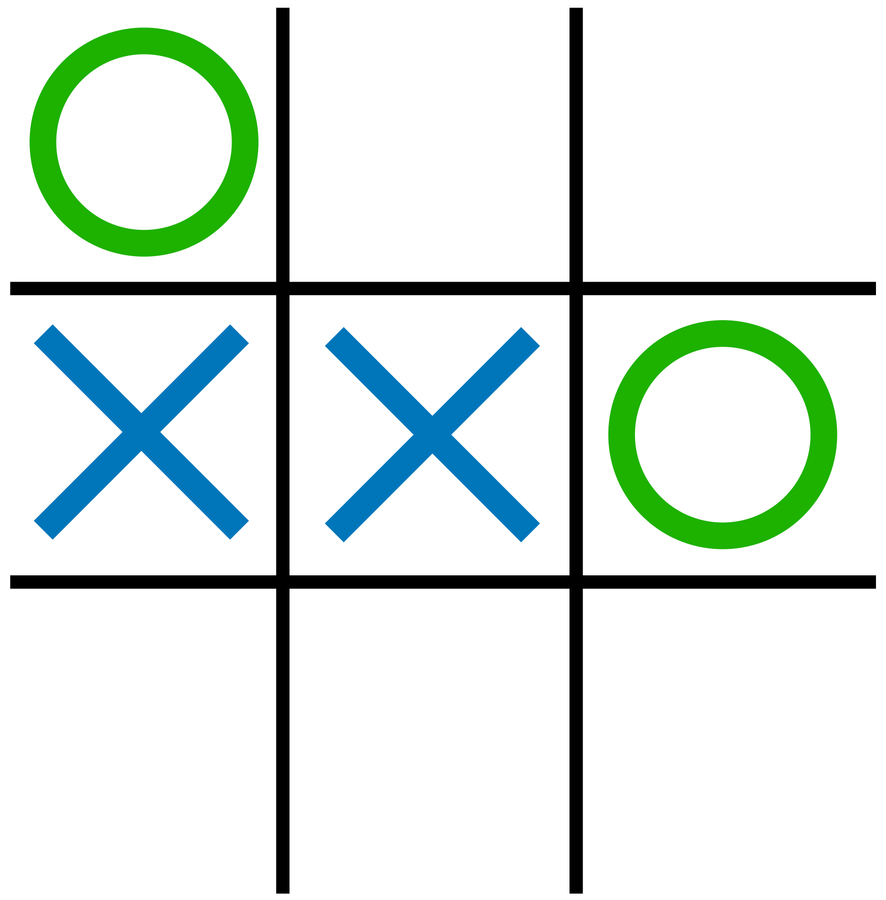
$0 < \gamma \leq 1$

Goal: Maximize the score (cumulative reward *r*)

Example environment for TicTacToe

Assume we are playing as X

State



O		
X	X	O

Actions

1 = place X on square 1

2 = place X on square 2

3 = place X on square 3

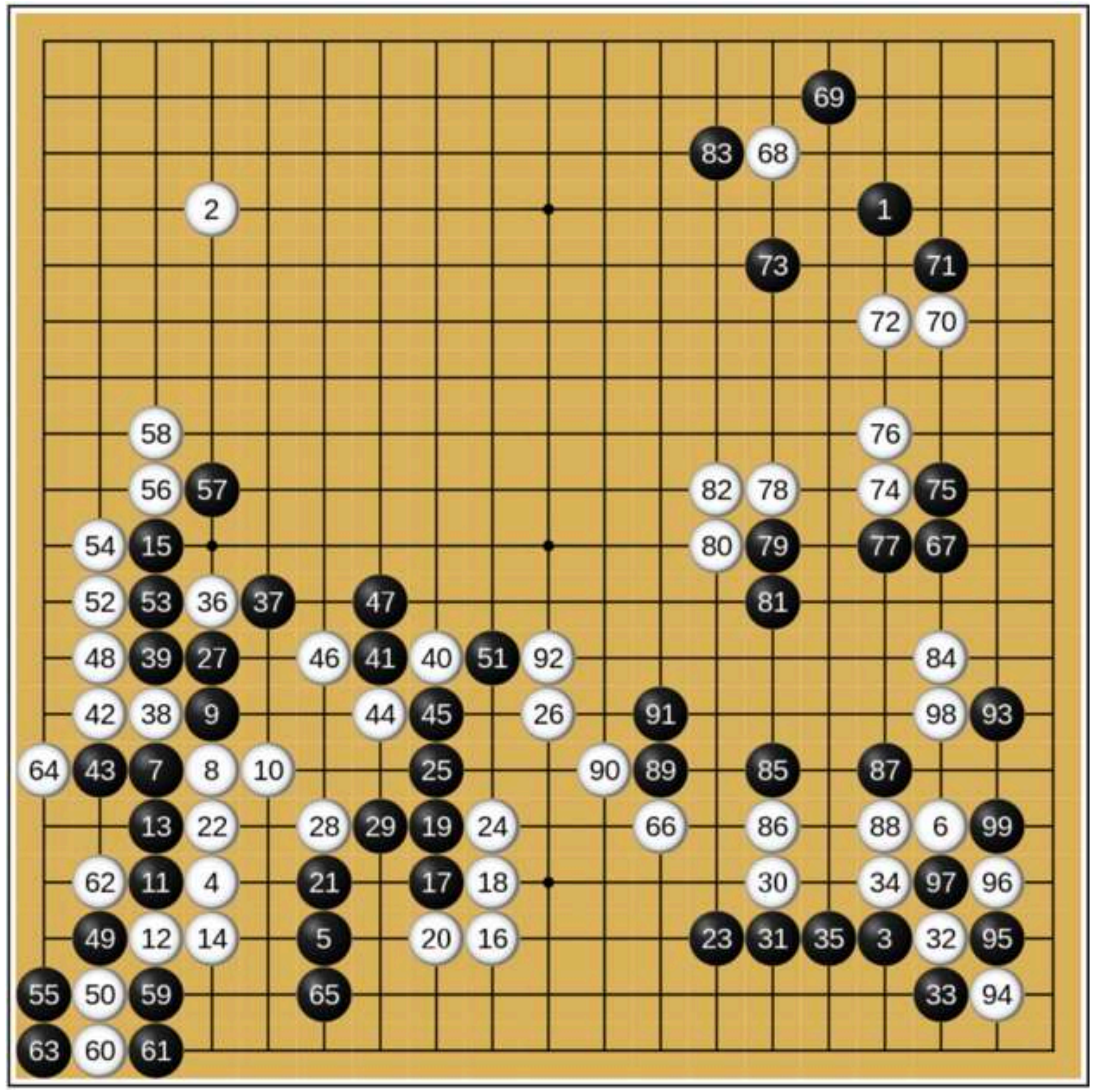
•
•
•

9 = place X on square 9

Rewards

$$R_t = \begin{cases} 0 \text{ if not finished} \\ -1 \text{ if O wins or draw} \\ +1 \text{ if X wins} \end{cases}$$

Each of these games can be formulated similarly

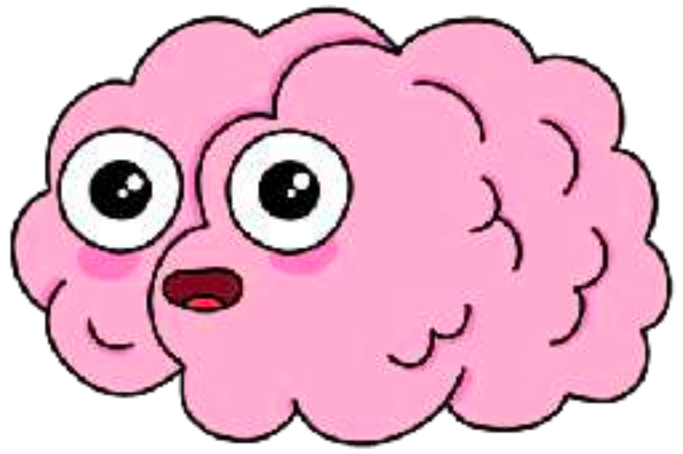


https://en.wikipedia.org/wiki/AlphaGo

https://deepmind.com/blog/article/Agent57-Outperforming-the-human-Atari-benchmark

The goal for the agent is to learn an optimal policy

Following this policy, the agent will maximize its total expected reward



$$= \pi(a \mid s)$$

Example policy for TicTacToe

$$\pi\left(6 \,\middle|\, \begin{array}{c|c|c} O & & \\ \hline X & X & \\ \hline O & & \end{array}\right) = 1 \qquad \pi\left(9 \,\middle|\, \begin{array}{c|c|c} O & X & \\ \hline & O & \\ \hline & & \end{array}\right) = 1$$

A RL problem is modelled as a Markov Decision Process

For a more complete intro, see next week's lectures by Florian Marquardt!

"The future (discounted) reward"

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+1+k}$$

$$V^{\pi}(s) = \mathbb{E}^{\pi}\left[G_t \mid S_t = s \right]$$

"How good is it to be in state s?"

(think: Chess Grandmaster looking at chess board 🤔)

$$Q^{\pi}(s, a) = \mathbb{E}^{\pi}\left[G_t \mid S_t = s, A_t = a \right]$$

"How good is it to be in state s and take action a?"

Q-learning is a way to find the optimal policy

$$\boxed{\pi(s) \rightarrow \max_{a} \; Q^*(s,a)}$$

"How good is it to be in state s and take action a?"

Finding the optimal Q-function: The Bellmann Equation

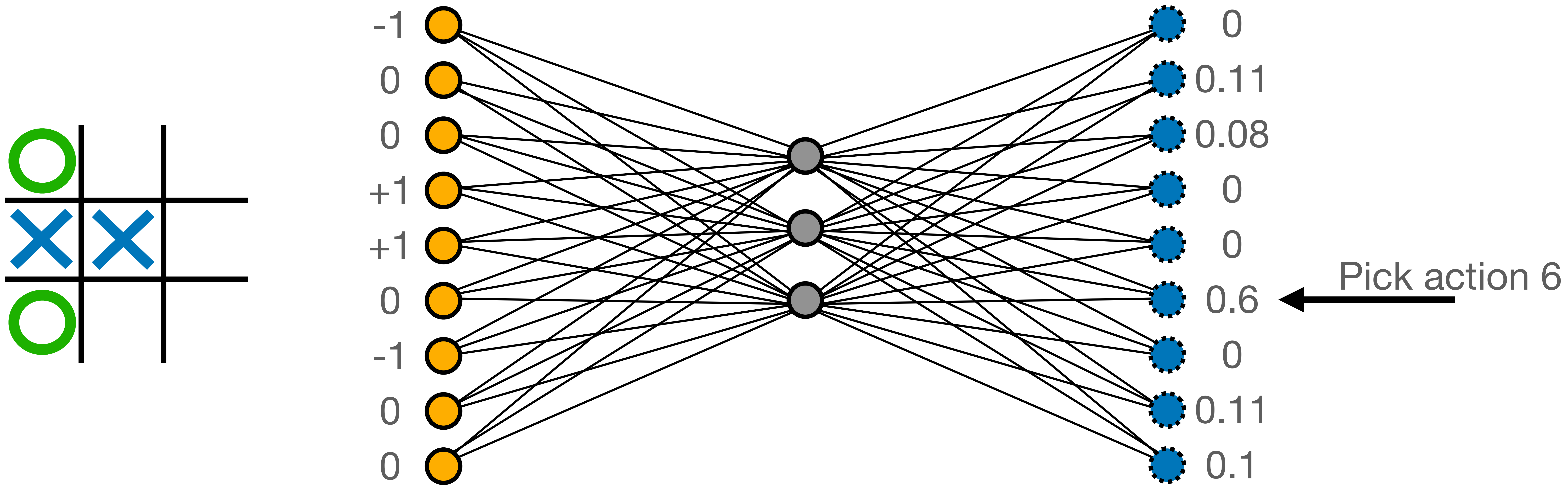
$$Q^*(s,a) = \mathbb{E}^{\pi}\left[R_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') \mid S_t = s, A_t = a\right]$$

Value iteration **Monte Carlo estimation** **Temporal Difference Learning**

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left[R_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a)\right]$$

A one-slider on Deep-Q learning

Instead of storing Q(s,a) as an array, use a network to parameterise it



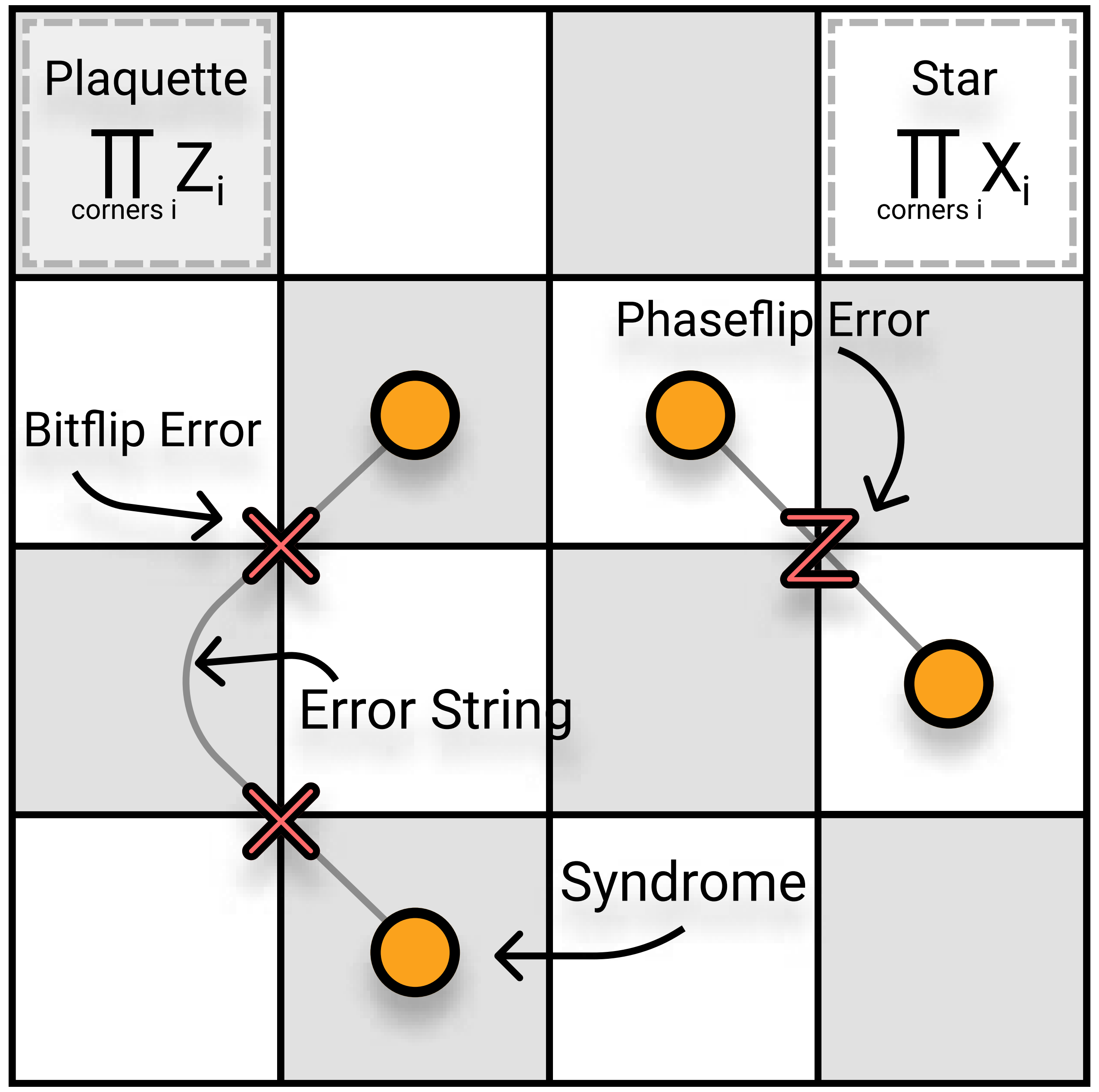
$$\boxed{\pi(s) \rightarrow \max_{a} \; Q^*(s,a)}$$

● input neuron ● hidden neuron ● output neuron

There are three main concepts for this talk

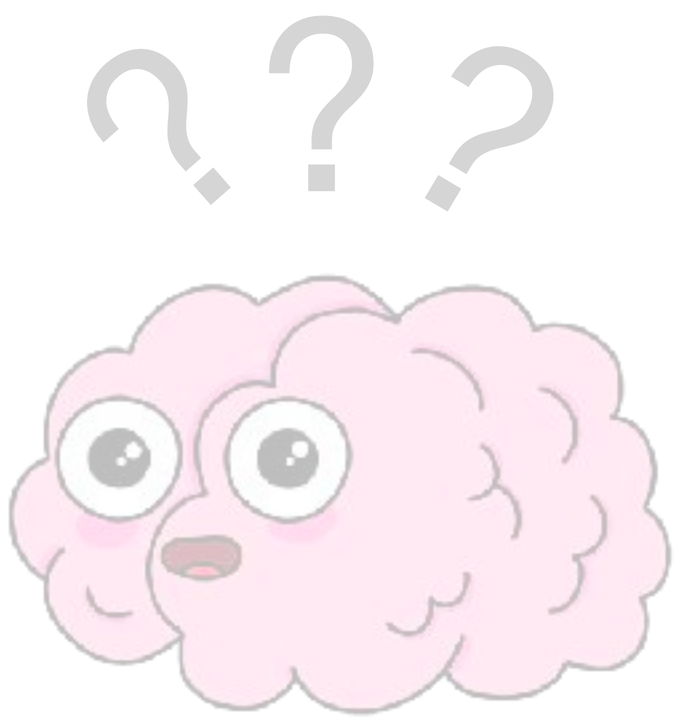
Don’t hesitate to ask!

Stabilizer codes


Plaquette
$\prod_{\text{corners } i} Z_i$
Star
$\prod_{\text{corners } i} X_i$
Phaseflip Error
Bitflip Error
Z
Error String
Syndrome

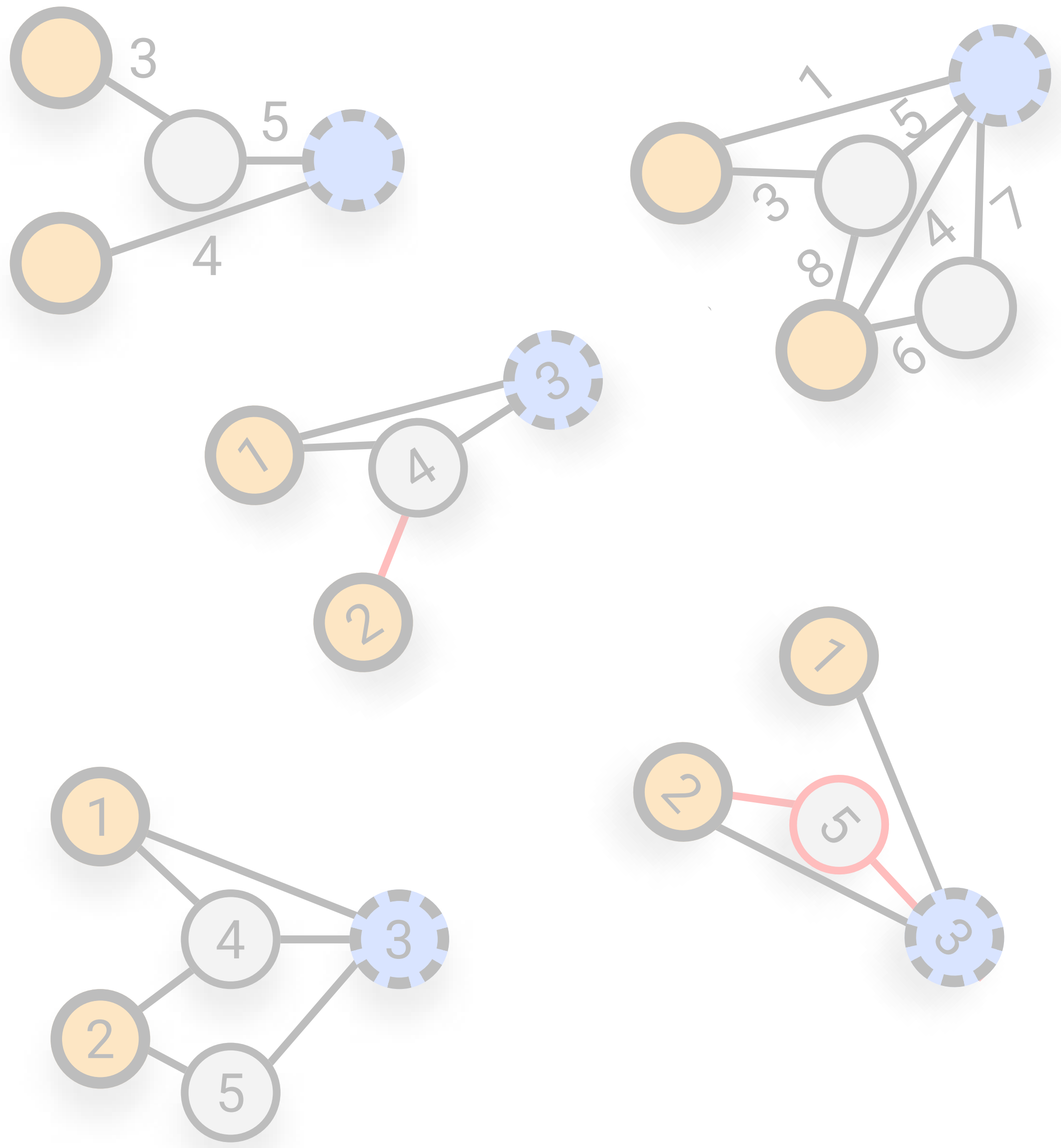
Quantum Computation

Reinforcement Learning



Deep Q-Network

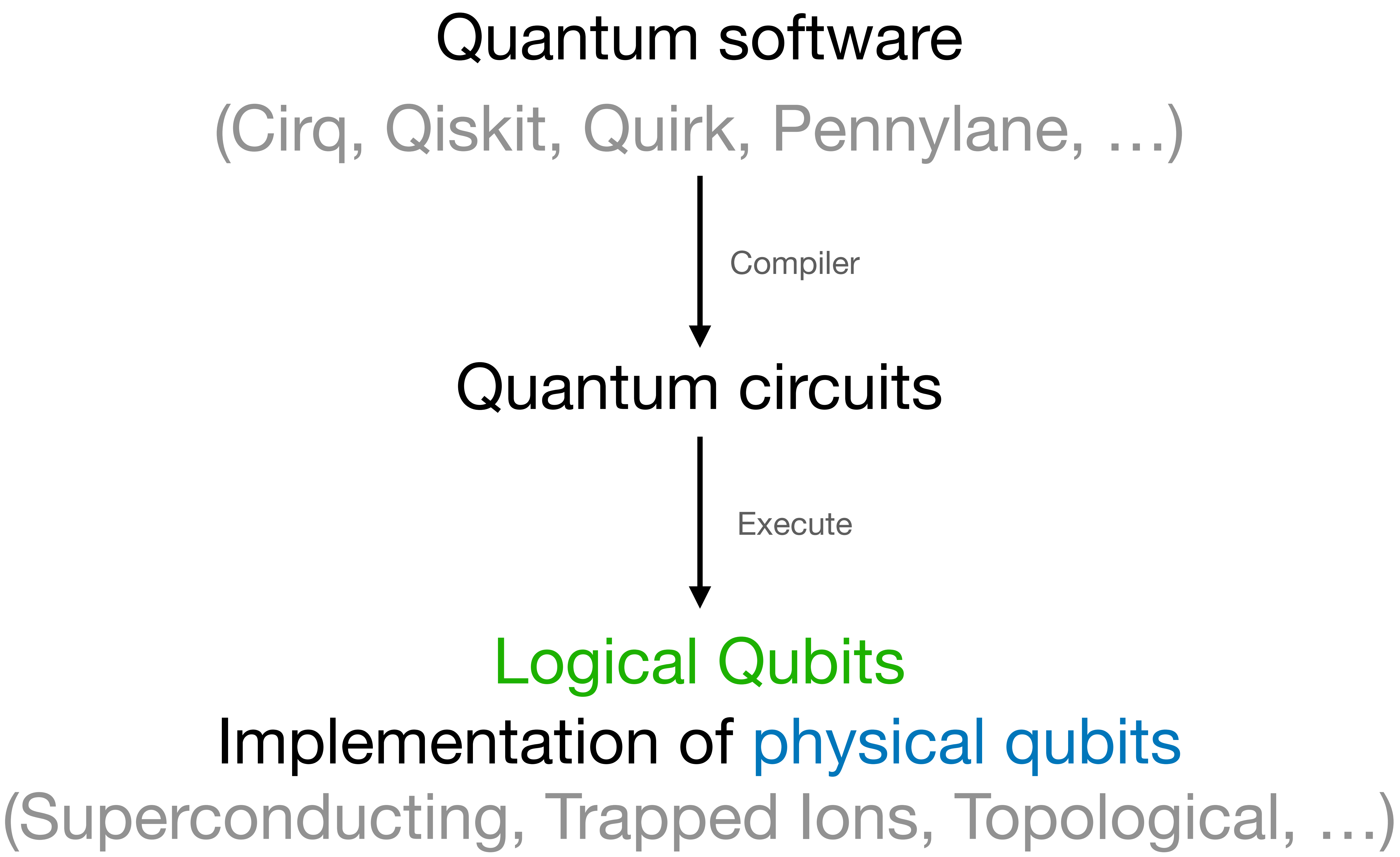
Evolutionary Strategy



Policy Networks

The layers of abstraction of a quantum computer

Quantum error correction is likely necessary for large Qcomputers



Quantum software

(Cirq, Qiskit, Quirk, Pennylane, …)

Compiler

Quantum circuits

Execute

Logical Qubits

Implementation of physical qubits

(Superconducting, Trapped Ions, Topological, …)

A very brief recap of the decoding problem

The name of the game is redundancy

$$0 \to 000$$
$$1 \to 111$$

$$0110 \to 000111111000$$

$$\Big\downarrow \text{Errors}$$

$$\underbrace{010}_{0}\ \underbrace{111}_{1}\ \underbrace{011}_{1}\ \underbrace{001}_{0} \xrightarrow{\text{Correcting}} 000111111000 \to 0110$$

Decoding

$$|0\rangle \to |000\rangle \qquad |000\rangle + e^{i\phi}|010\rangle + \dots \xrightarrow{\text{Measurement}} |000\rangle$$

So we need a way to **encode** (redundancy!) a qubit in such a way, that we can know an error happened (**decoding**), and fix it (**correction**), without destroying a superposition!

A very simple quantum code

For bitflip errors only (for now)

$$|0\rangle \rightarrow \frac{1}{\sqrt{2}}\Big(|000\rangle + |111\rangle\Big)$$

$$Z_1Z_2|0\rangle = |0\rangle \qquad Z_2Z_3|0\rangle = |0\rangle$$

$$X_1|0\rangle = \frac{1}{\sqrt{2}}\Big(|100\rangle + |011\rangle\Big)$$

$$Z_1Z_2X_1|0\rangle = -|0\rangle$$

$$Z_2Z_3X_1|0\rangle = |0\rangle$$

	Z_1Z_2	Z_2Z_3
I	1	1
X_1	-1	1
X_2	-1	-1
X_3	1	-1

A very simple quantum code

For bitflip errors only (for now)

$$|0\rangle \rightarrow \frac{1}{\sqrt{2}}\Big(|000\rangle + |111\rangle\Big)$$

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \xrightarrow{\text{Error}} |\tilde{\psi}\rangle$$

$$\langle\tilde{\psi}|S_1|\tilde{\psi}\rangle = 1$$
$$\langle\tilde{\psi}|S_2|\tilde{\psi}\rangle = -1$$

$S_1 = Z_1Z_2$ $\quad S_2 = Z_2Z_3$

	S_1	S_2
I	1	1
X_1	-1	1
X_2	-1	-1
X_3	1	-1

Stabilizer codes

Stabiliser measurements result in a syndrome that identifies errors

$$\begin{pmatrix} S_1 \\ S_2 \\ S_3 \\ S_4 \end{pmatrix} = \begin{pmatrix} X & Z & Z & X & I \\ I & X & Z & Z & X \\ X & I & X & Z & Z \\ Z & X & I & X & Z \end{pmatrix}$$

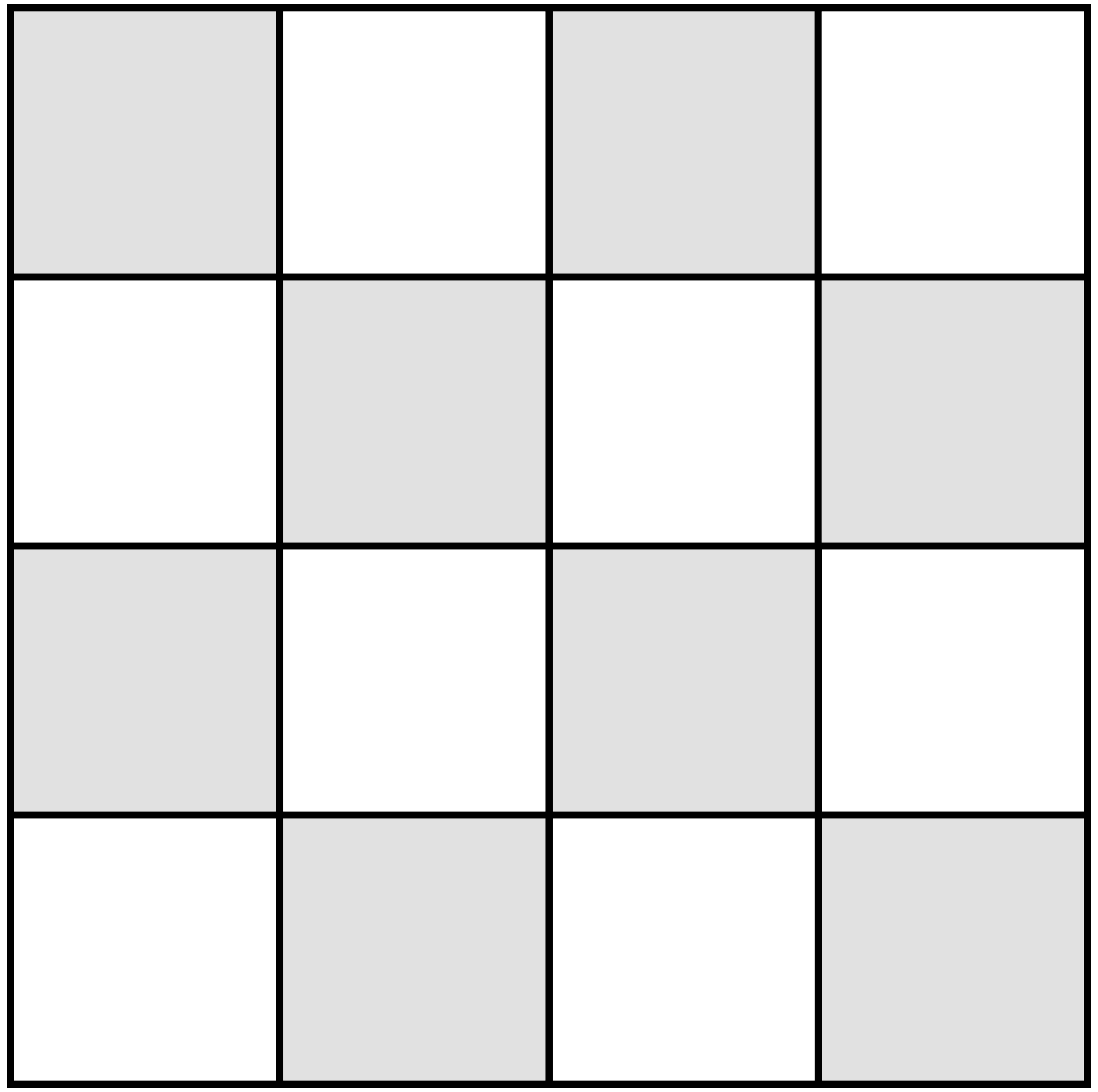
$$S_i|0\rangle = |0\rangle \;\; \forall i \qquad S_i|1\rangle = |1\rangle \;\; \forall i \qquad \langle 0|1\rangle = 0$$

Each single qubit error causes a -1 pattern of ‘violated stabilisers’

Syndrome

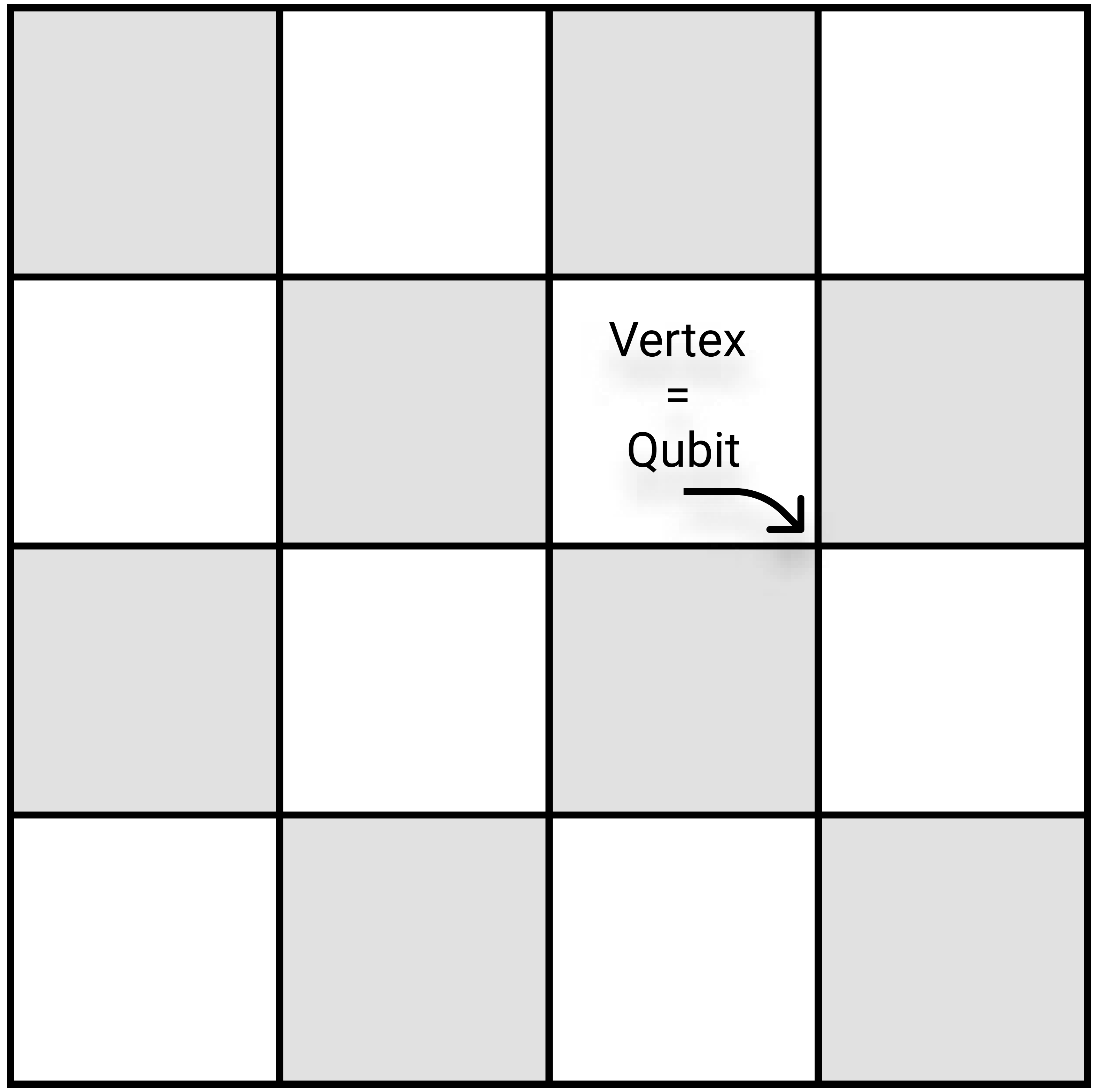
The toric code is a famous stabilizer code

Use many (physical) qubits to encode a pair of ‘logical’ qubits



The toric code is one way of getting qubits

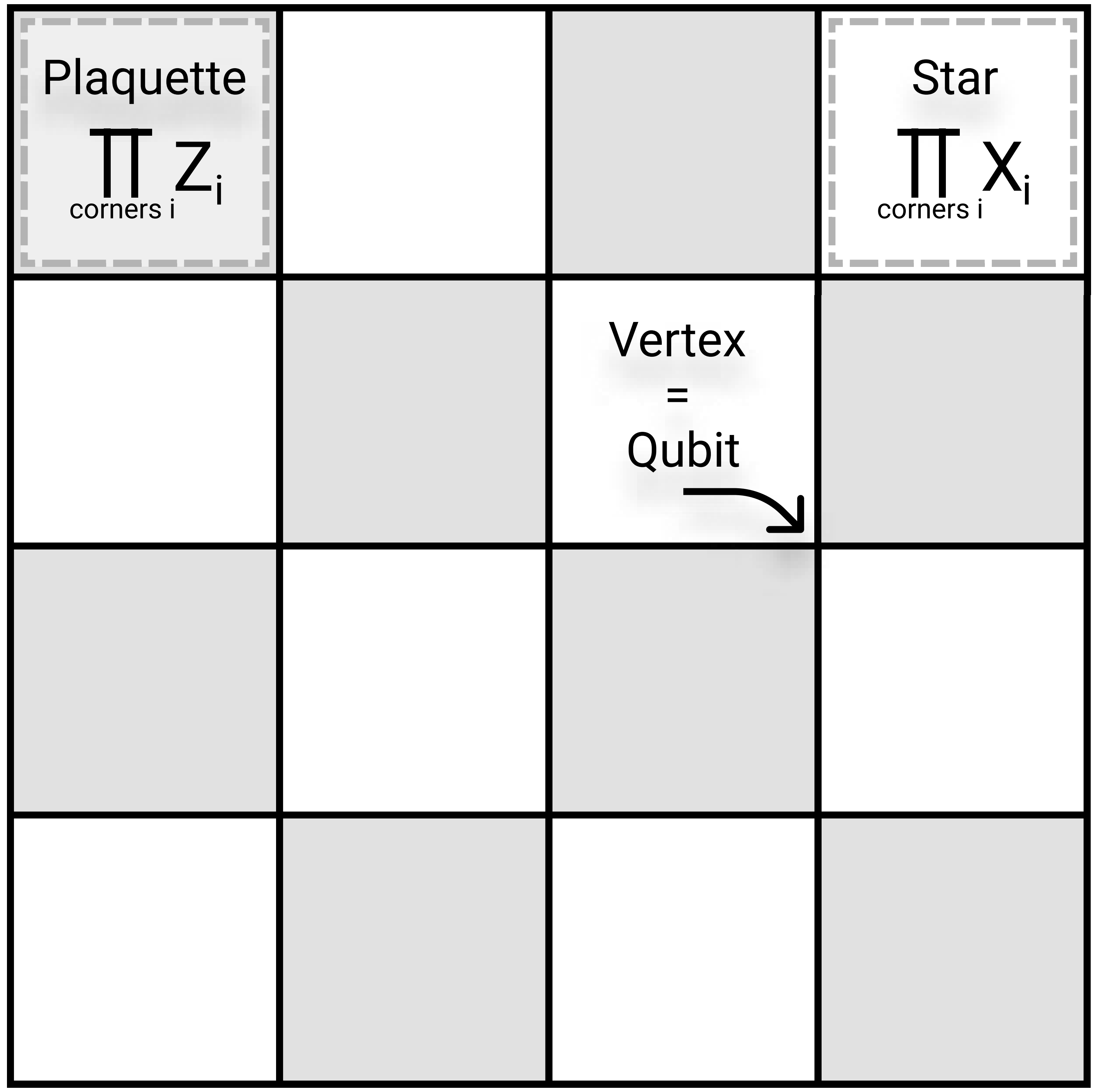
Use many (physical) qubits to encode a pair of 'logical' qubits



This system has **periodic boundaries** (cf the surface code)

The toric code encodes 2 logical qubits

They are built out of four degenerate ground states

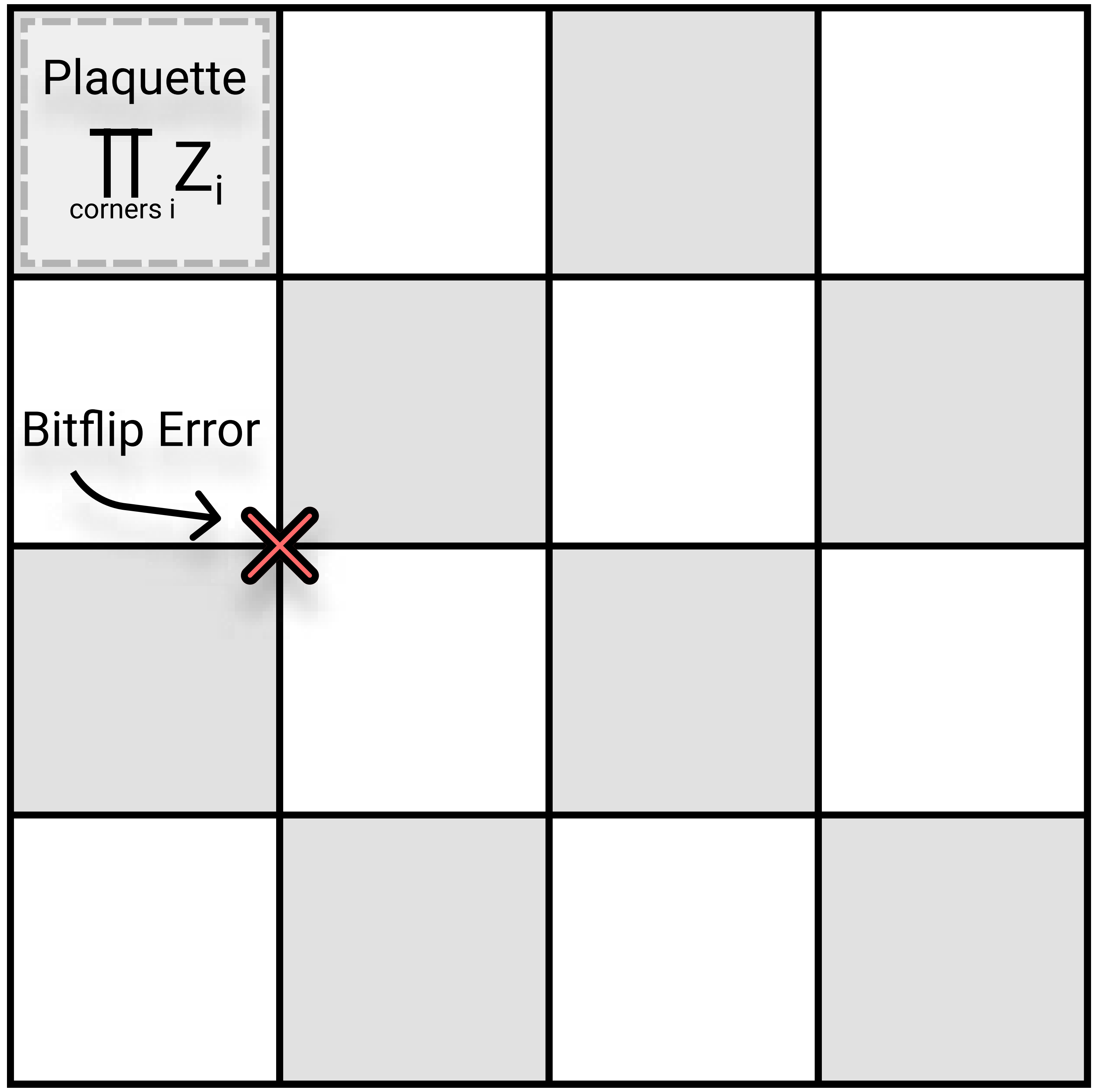

Plaquette
$\prod_{\text{corners } i} Z_i$
Star
$\prod_{\text{corners } i} X_i$
Vertex
=
Qubit

$$H = -\sum_{\text{plaquette}} P - \sum_{\text{star}} S$$

All ground states have Plaquette = +1 and Star = +1

Physical qubits have errors

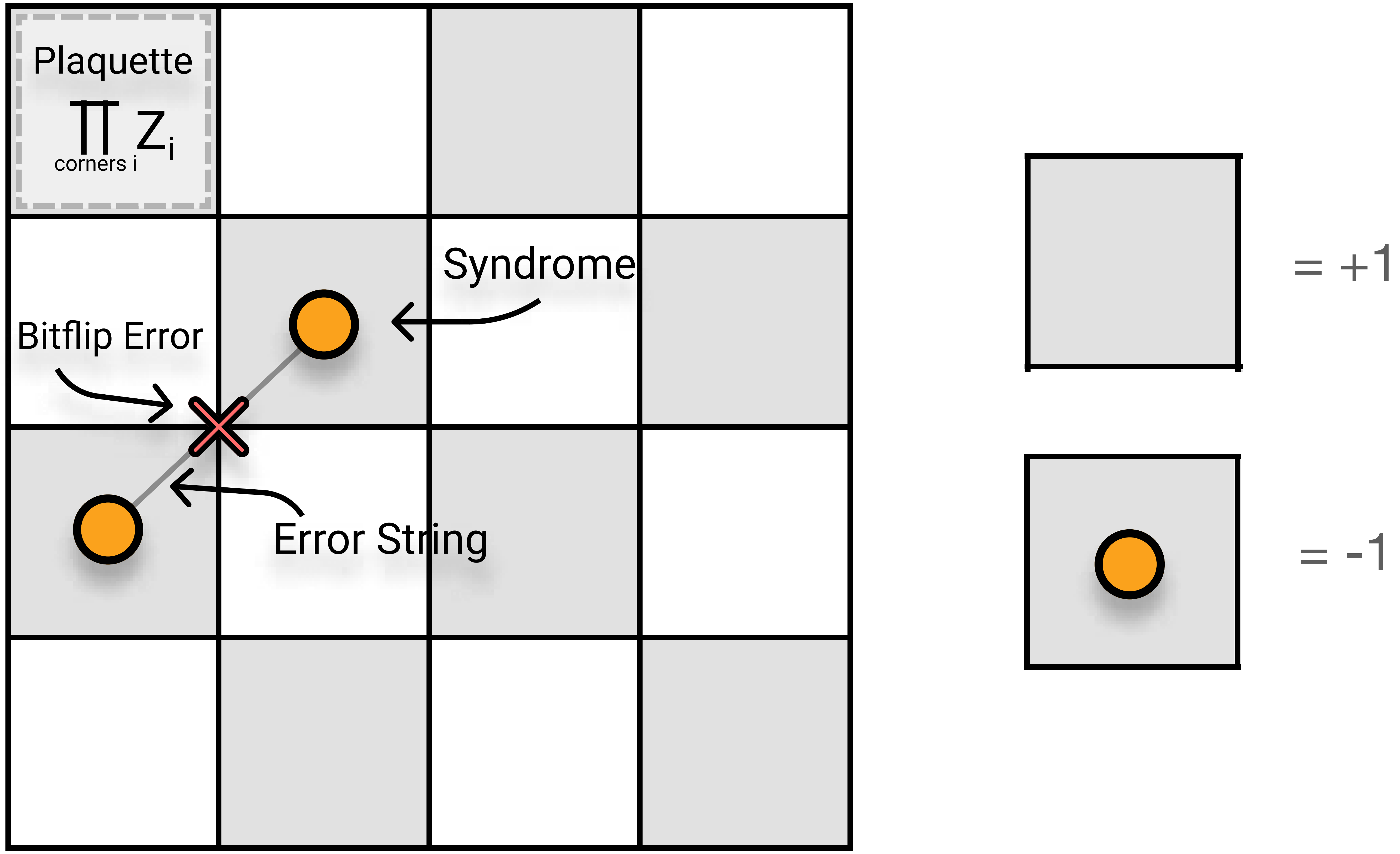
They can have Pauli X errors, or Pauli Y errors, or Pauli Z errors



Error takes us out of the 4-fold degenerate ground state space

We can not observe the errors (would collapse)

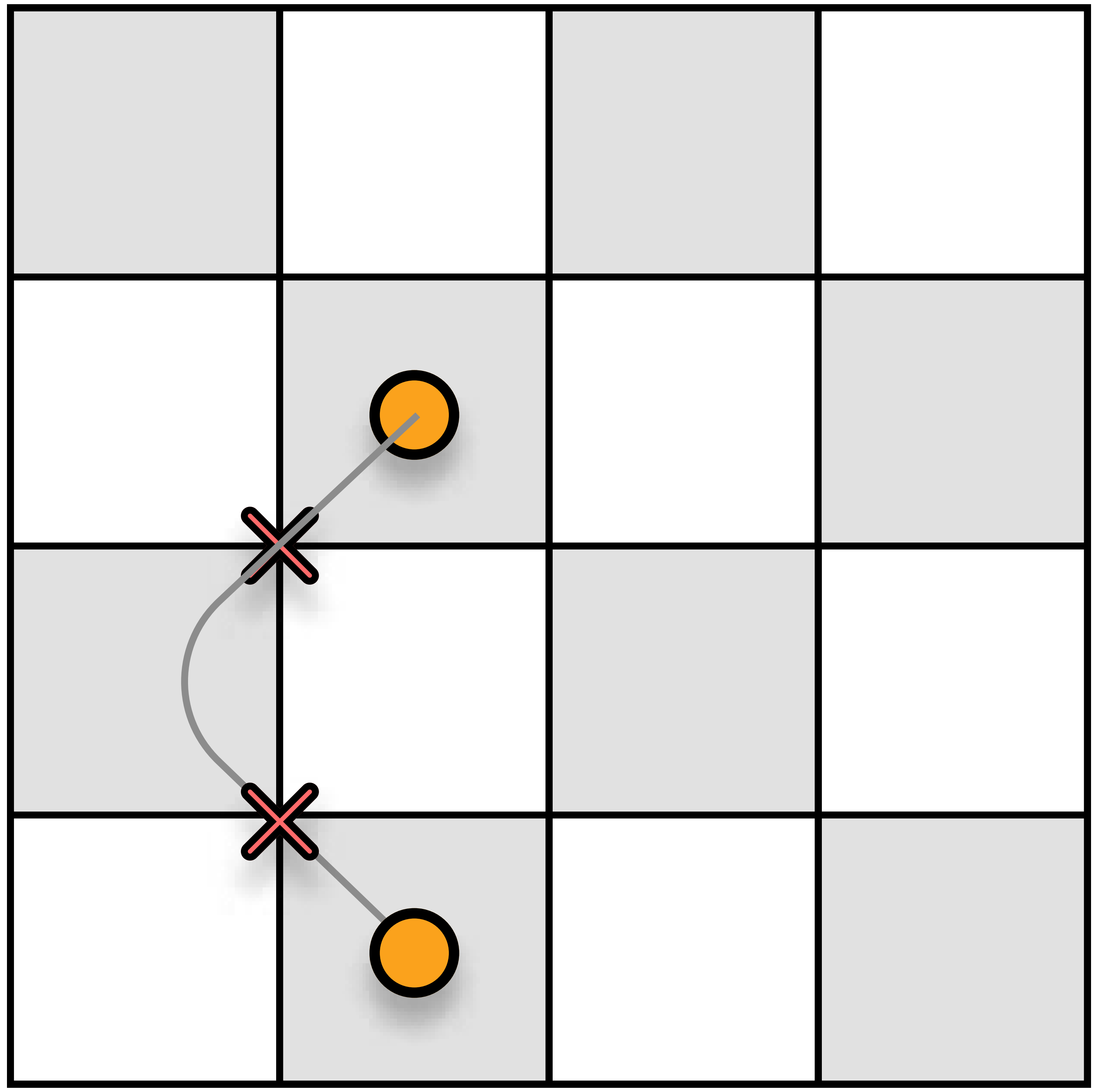
But the plaquette operator changes sign!



Errors leave behind a **syndrome**

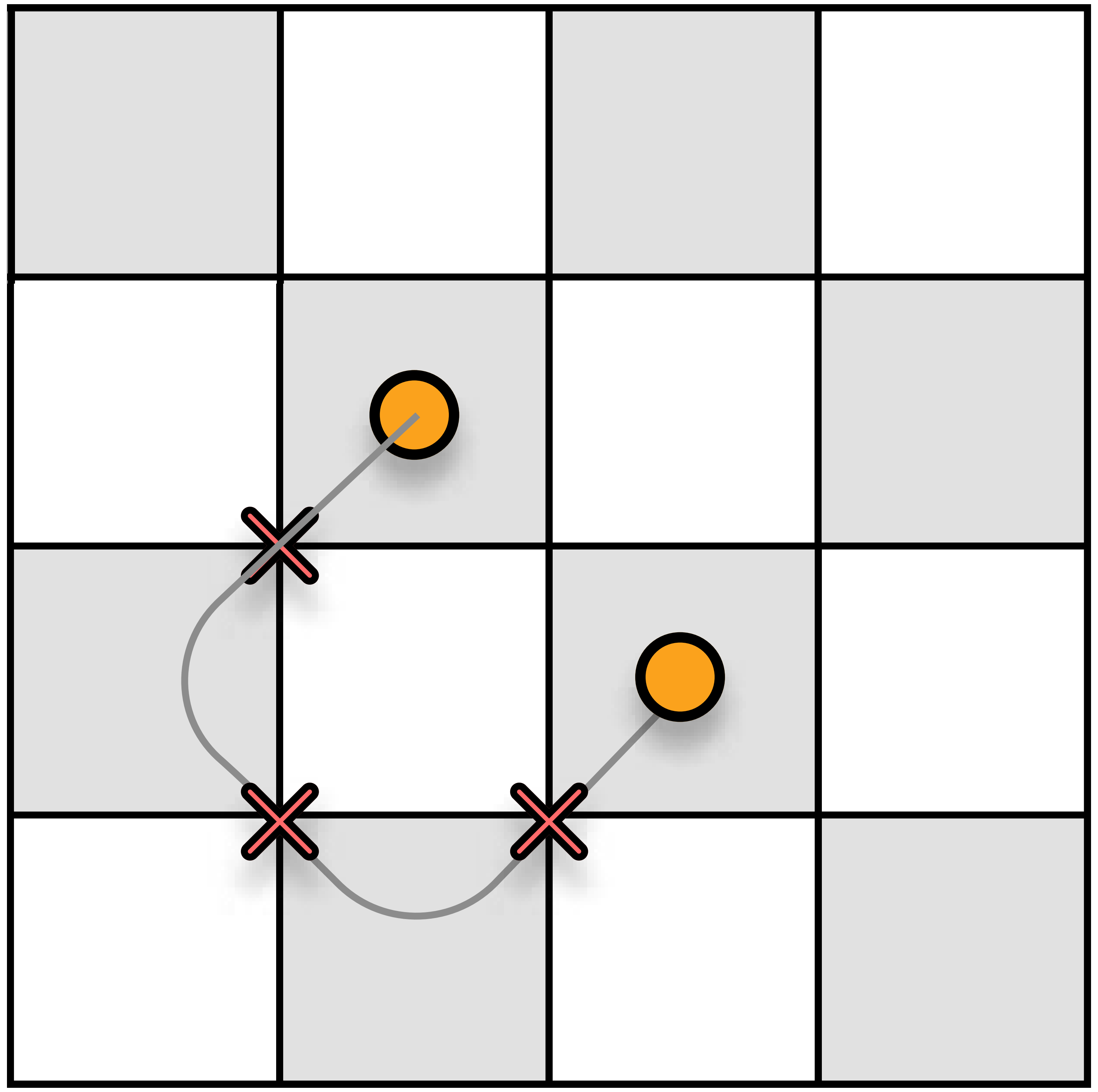
More errors move syndrome endpoints around

Pairs of syndrome points connected by an error string



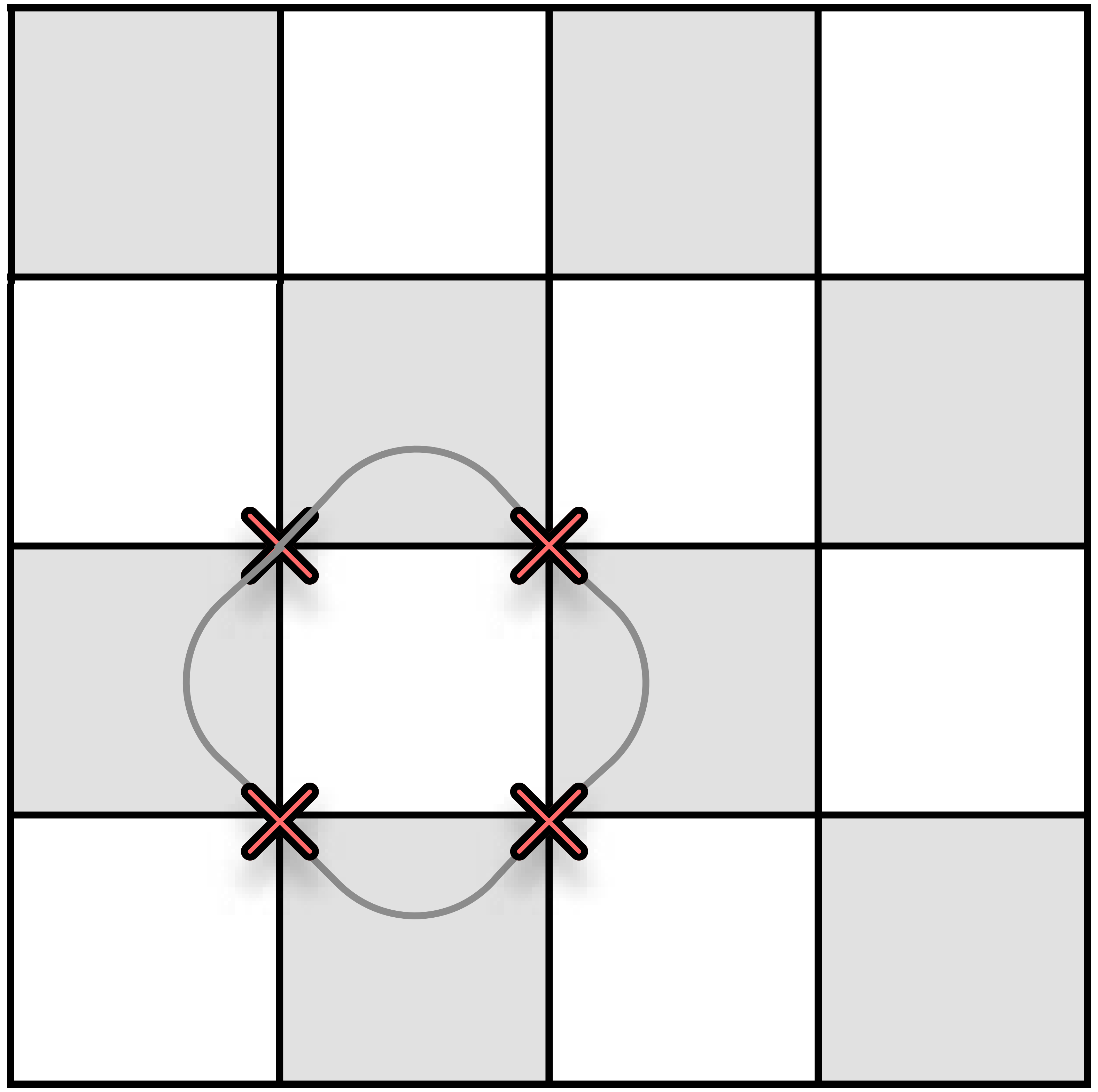
If errors happen to do this…

…wait for it



No more syndrome!

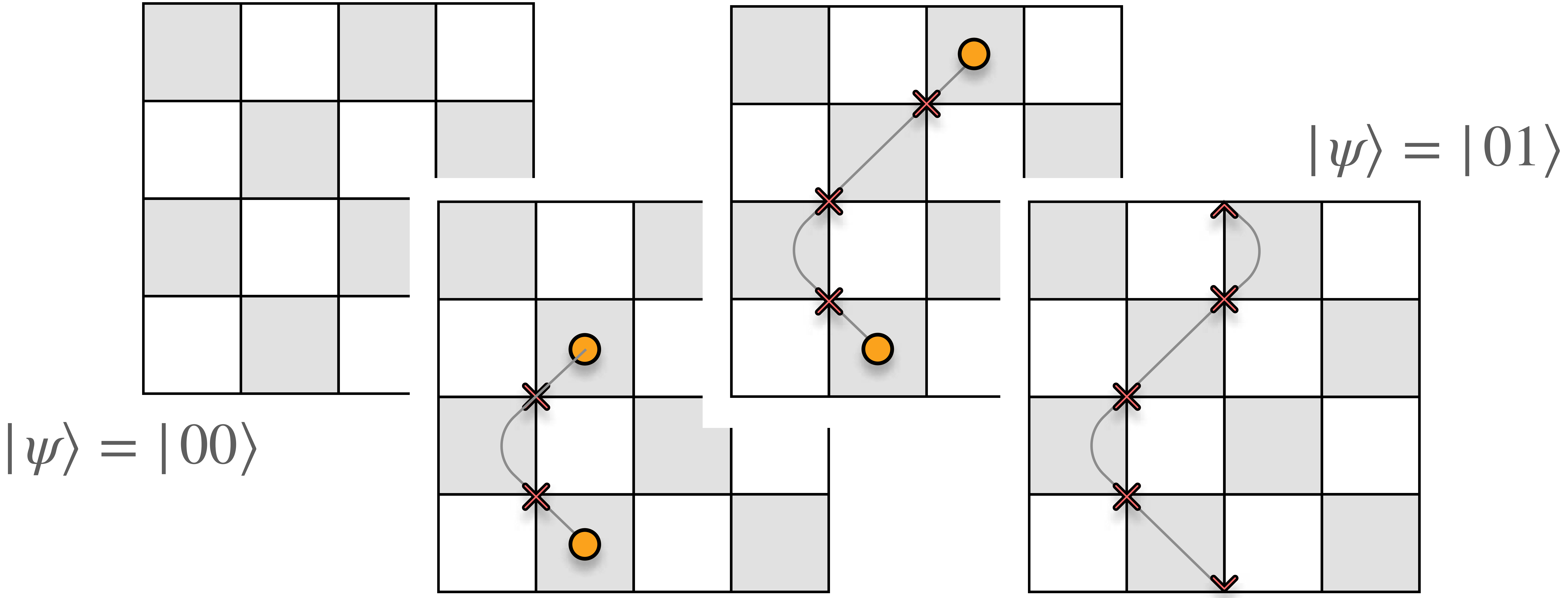
Errors occurred, but we are now back in the groundstate space!



This error string forms a contractable loop

Alternative history

Error strings connecting boundaries are logical operations!

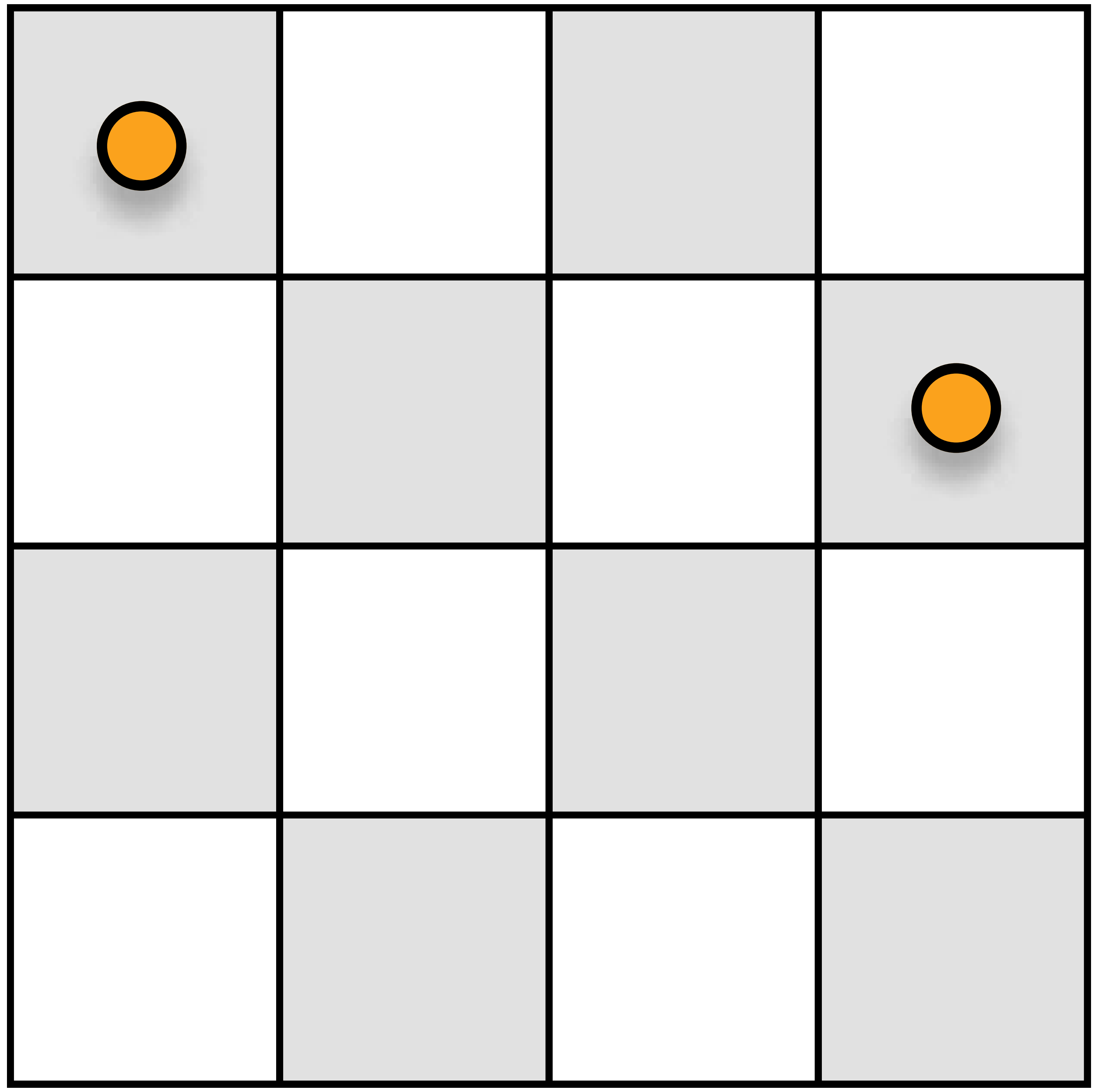


This error string forms a non-contractable loop

This is a “distance” d=4 code

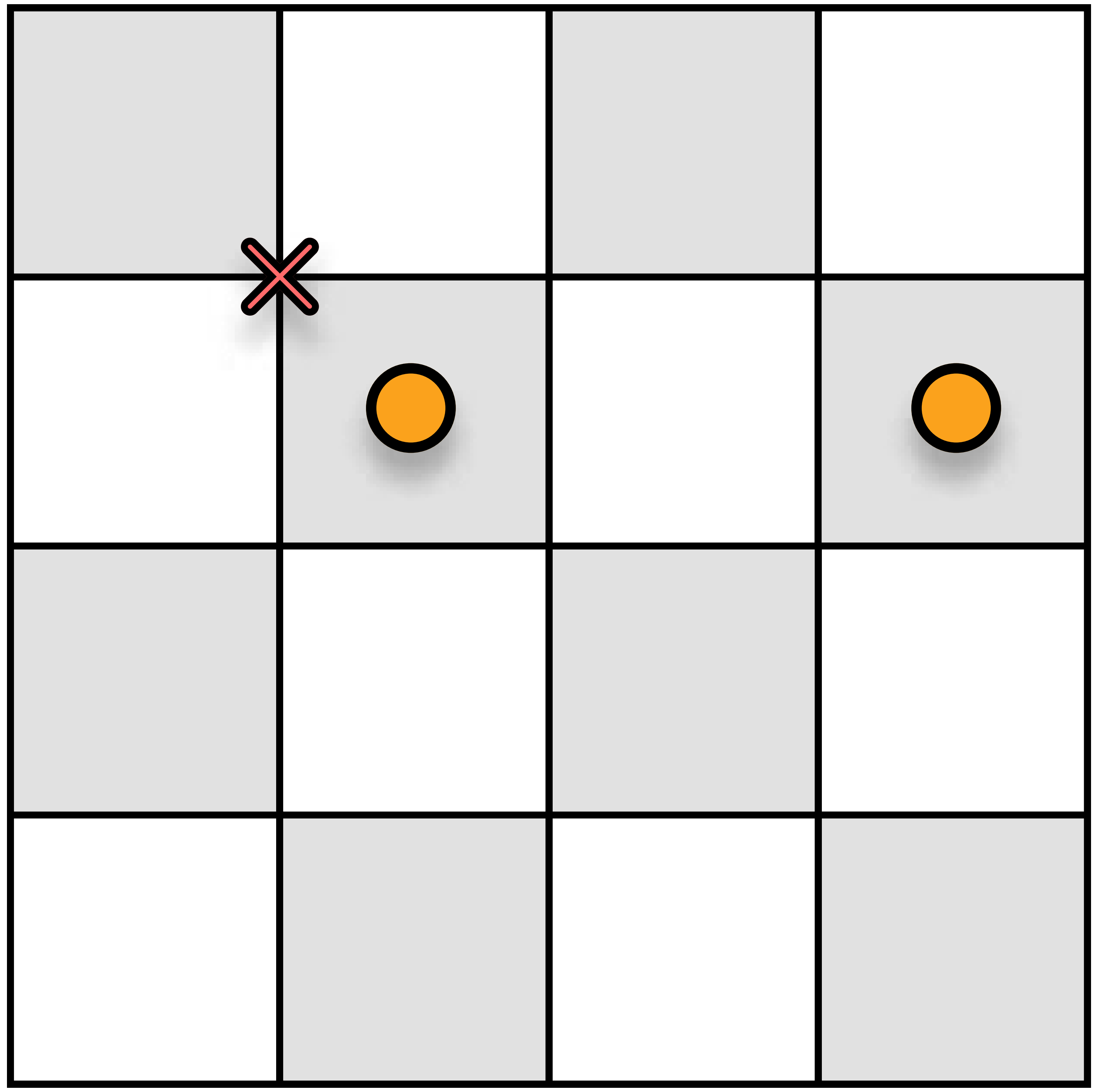
QEC = finding the right error string

This is a single player game: Merge syndrome points, one move at a time



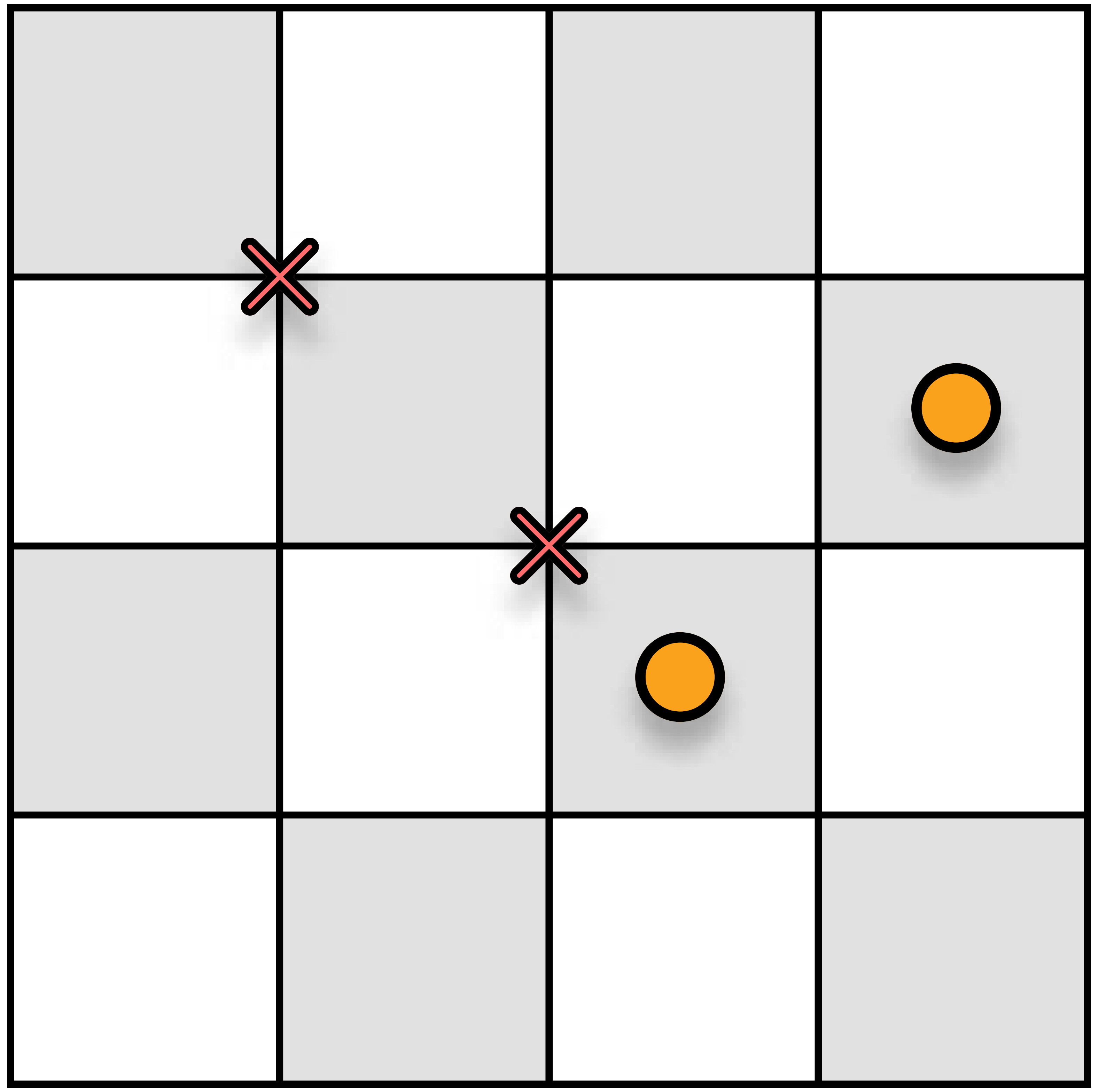
QEC = finding the right error string

This is a single player game: Merge syndrome points, one move at a time



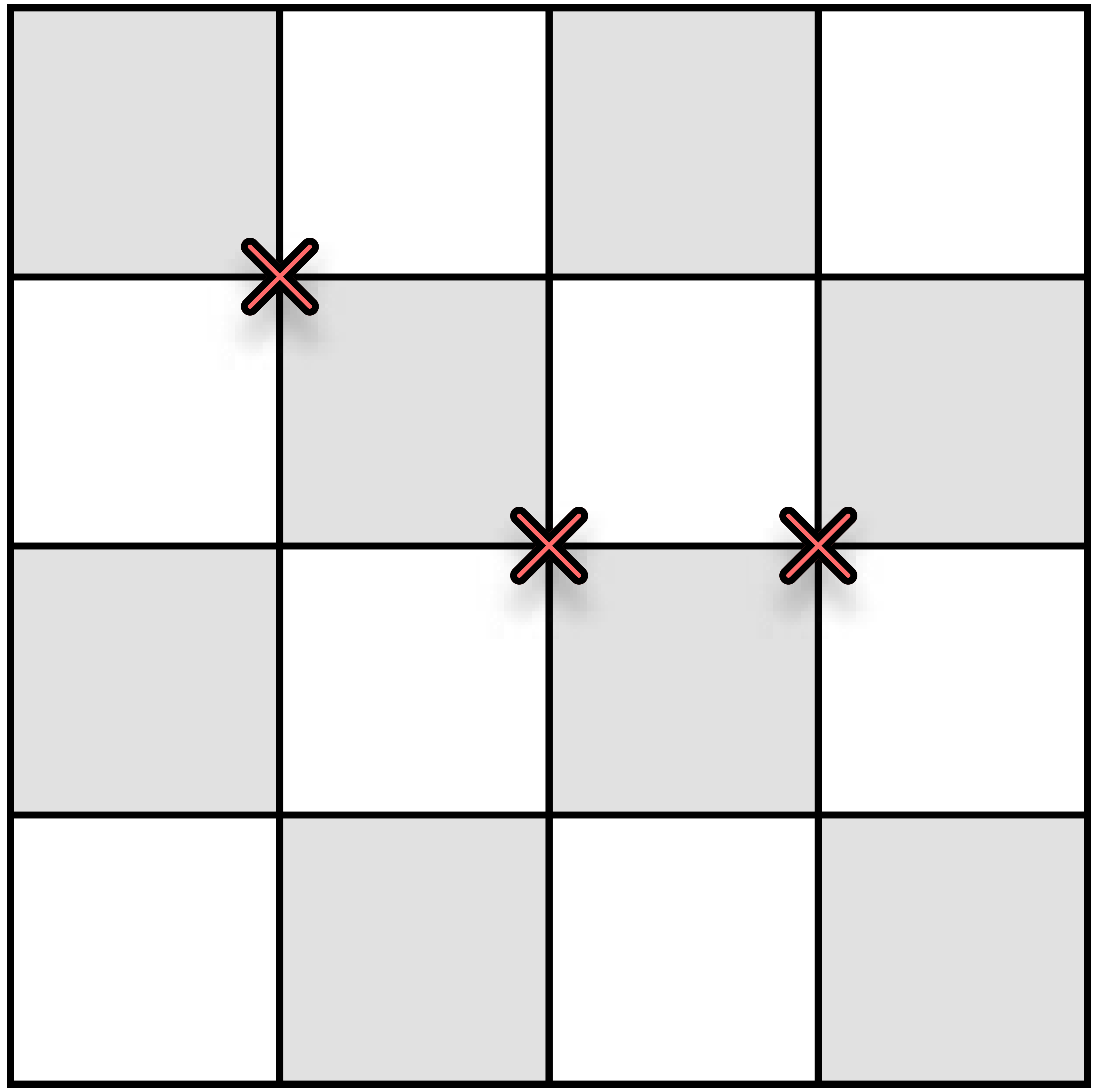
QEC = finding the right error string

This is a single player game: Merge syndrome points, one move at a time



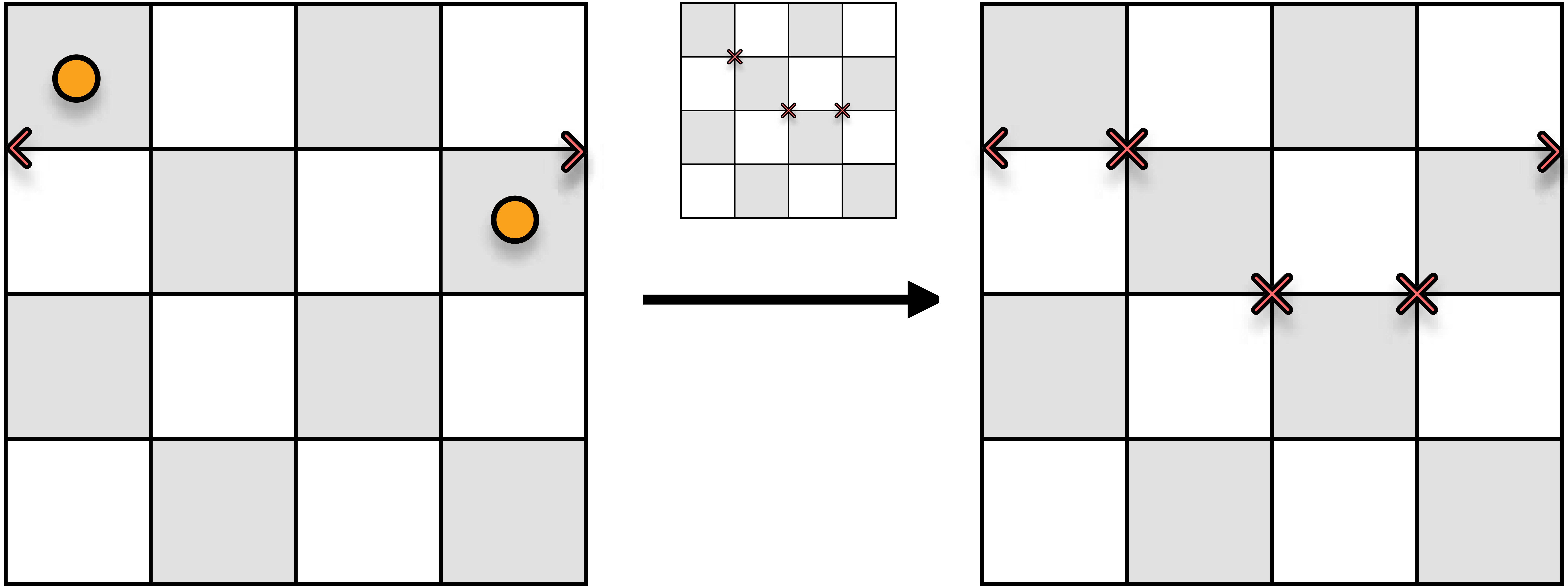
QEC = finding the right error string

This is a single player game: Merge syndrome points, one move at a time



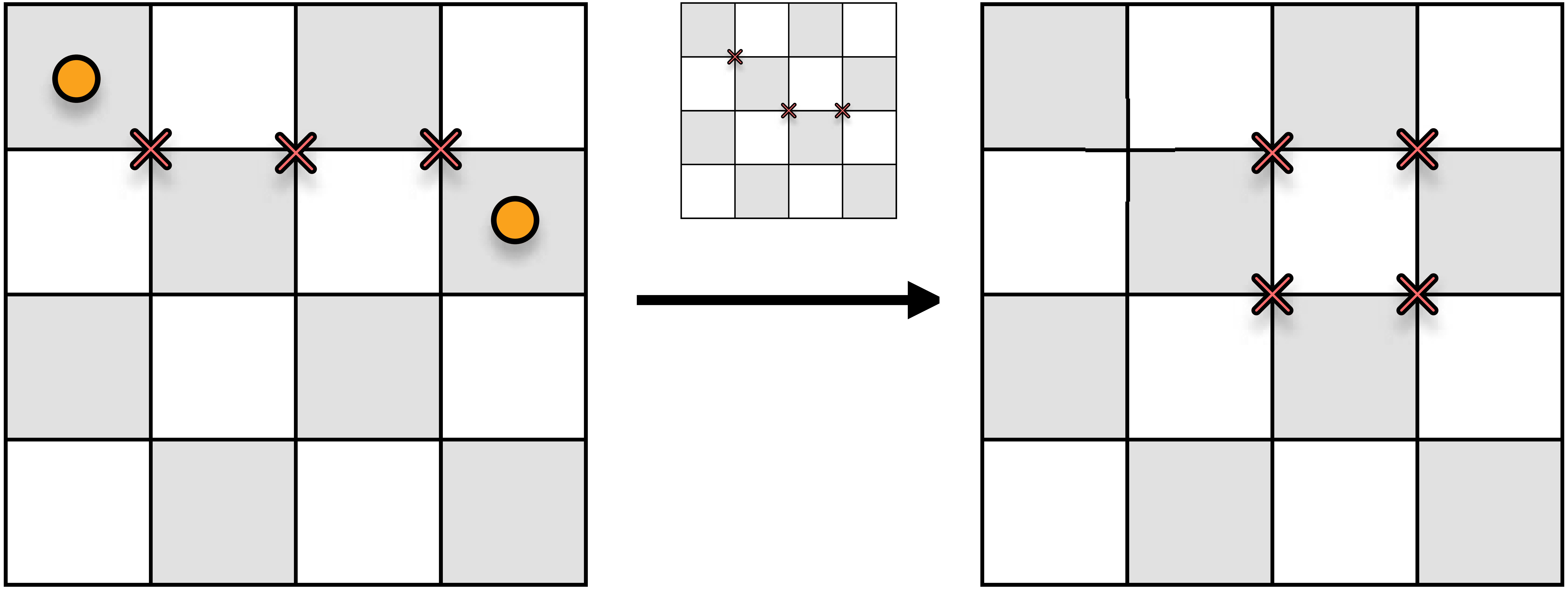
You thought you won, but alas…

Error string wraps around boundaries (non-contractable)



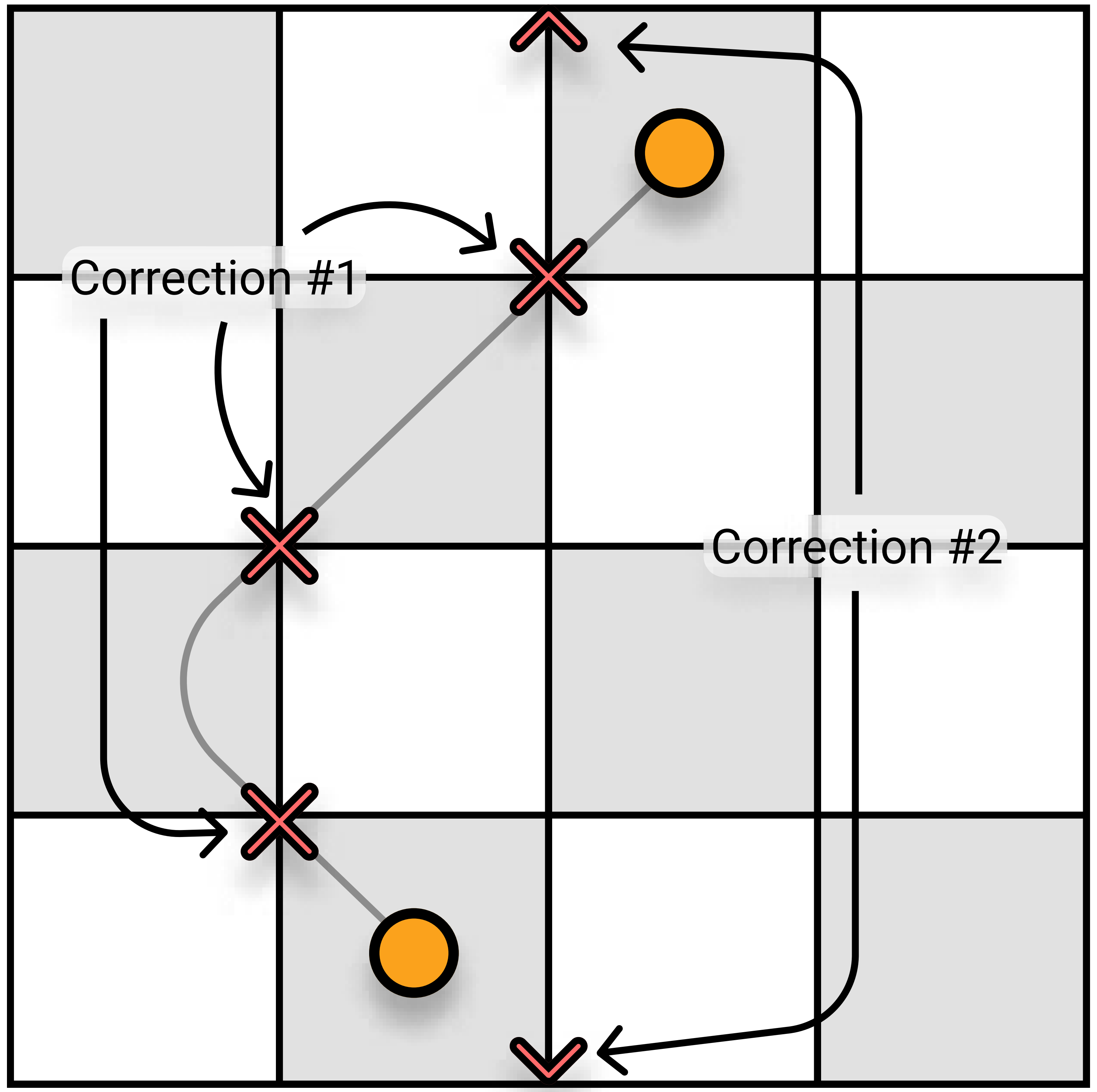
But in this scenario, you *did* win!

No error string connecting boundaries (contractable)



There is a (near-)optimal solution

The “Minimum Weight Perfect Matching” (MWPM) algorithm



The near-optimal solution

The “Minimum Weight Perfect Matching” (MWPM) algorithm

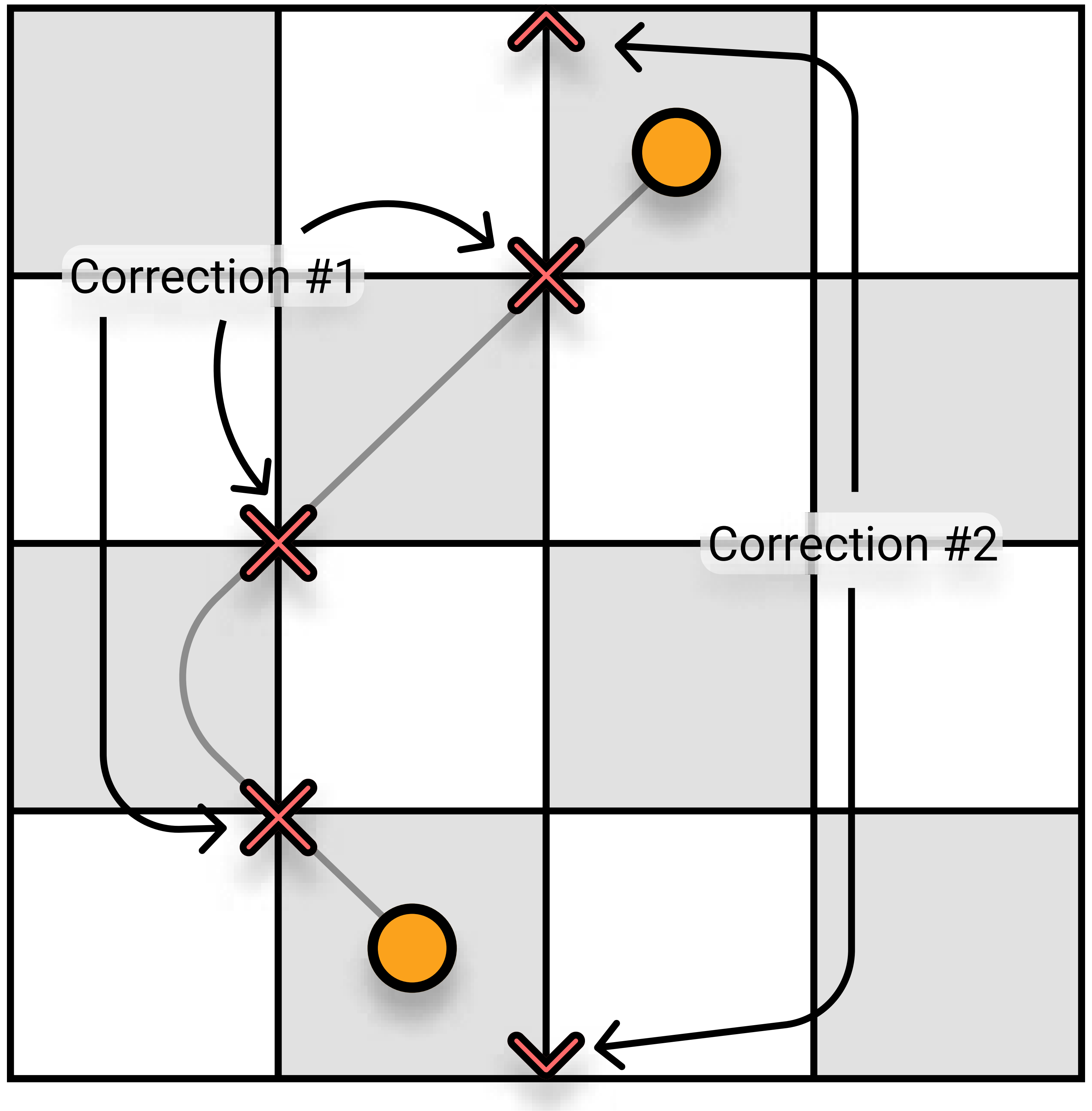
Physical qubit error probability: p

Error string of length **L** probability: p^L

Shorter string is *likelier*

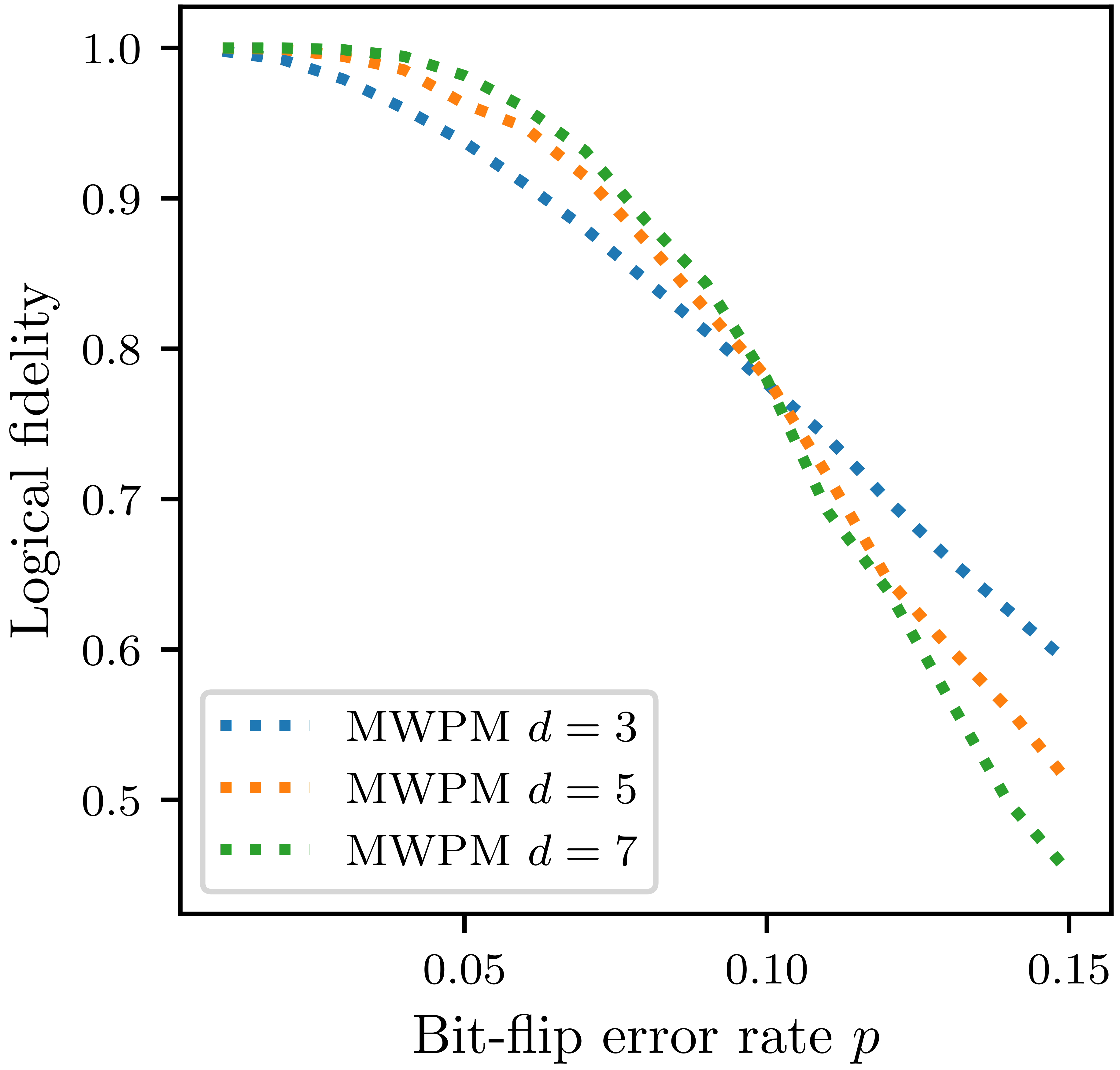
↓

Find shortest string! (= **MWPM** algorithm)



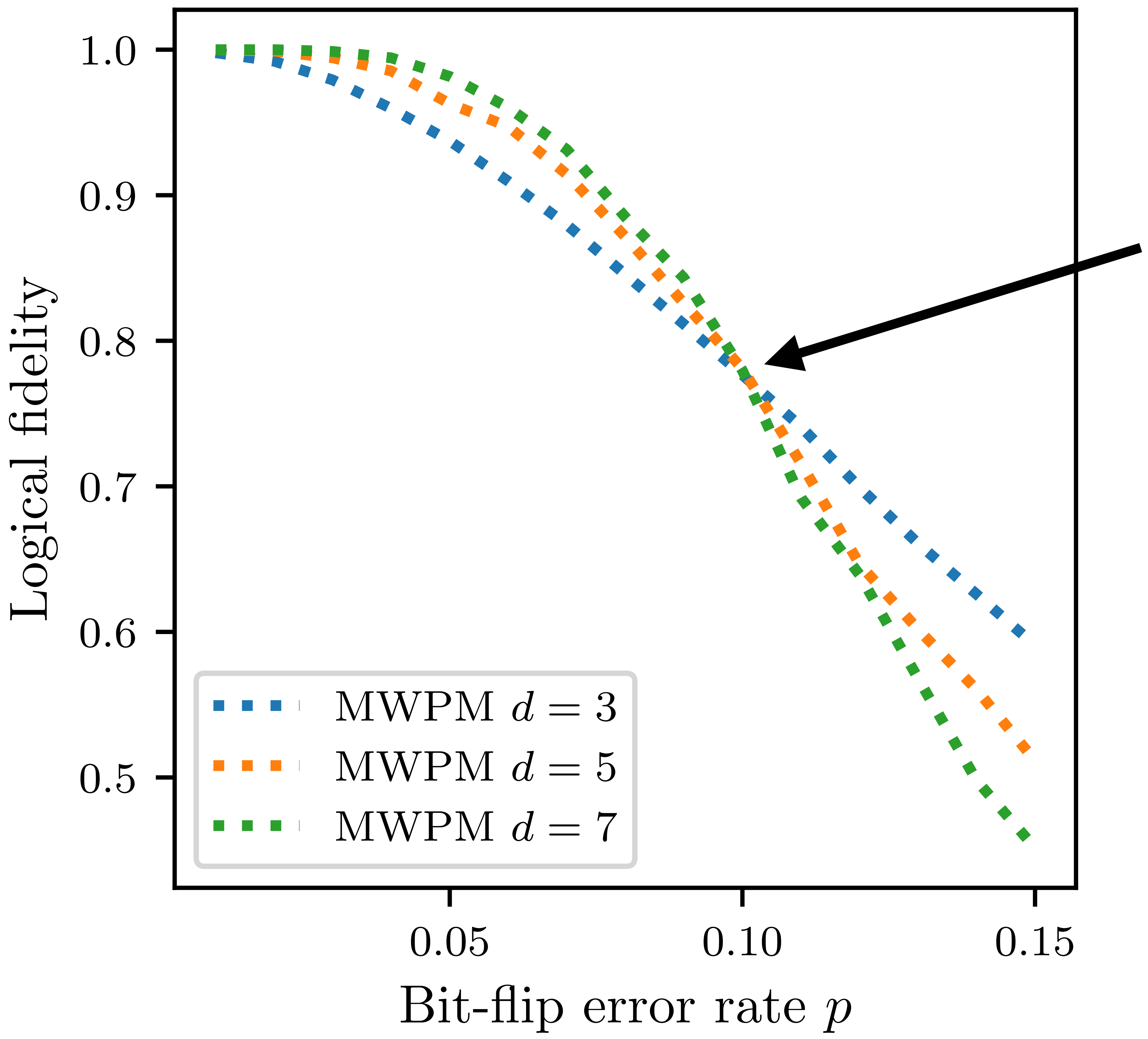
Tracking logical fidelity versus error rate

This is a property of the correction algorithm



MWPM has a “threshold”

Not all correction algorithms have a threshold; having one is good!

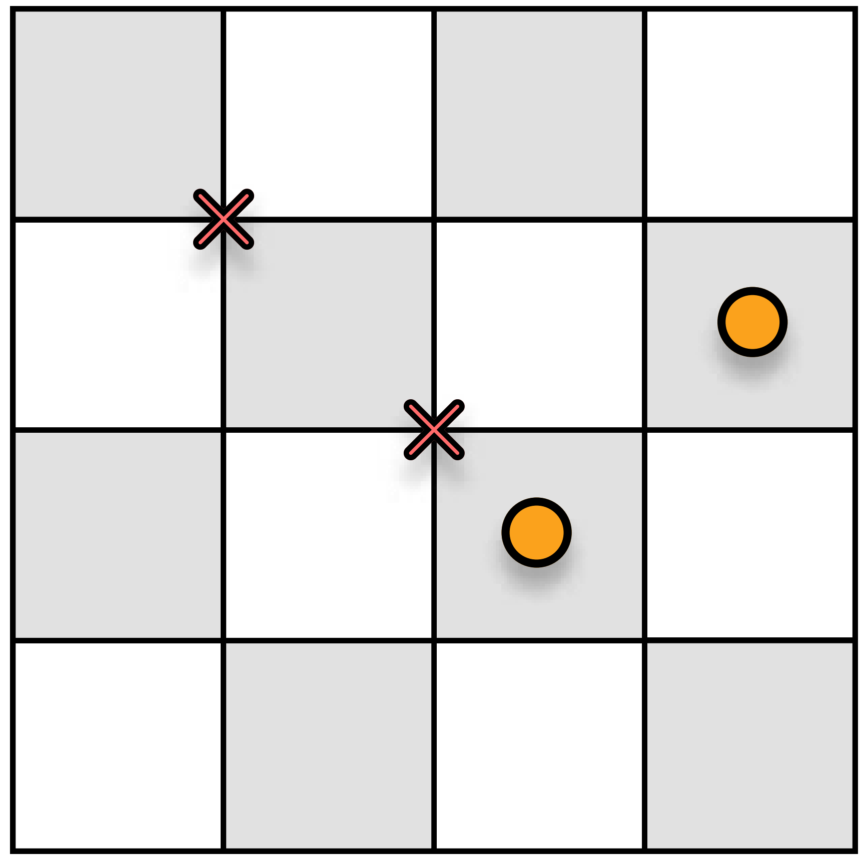


Threshold

The Toric Game

A reinforcement learning environment for QEC

State



Actions

Act with Pauli X, Y or Z on qubit

($3d^2$ actions)

-OR-

Use translational invariance

Move syndrome to reference loc

X,Y,Z on 4 qubits around syndrome

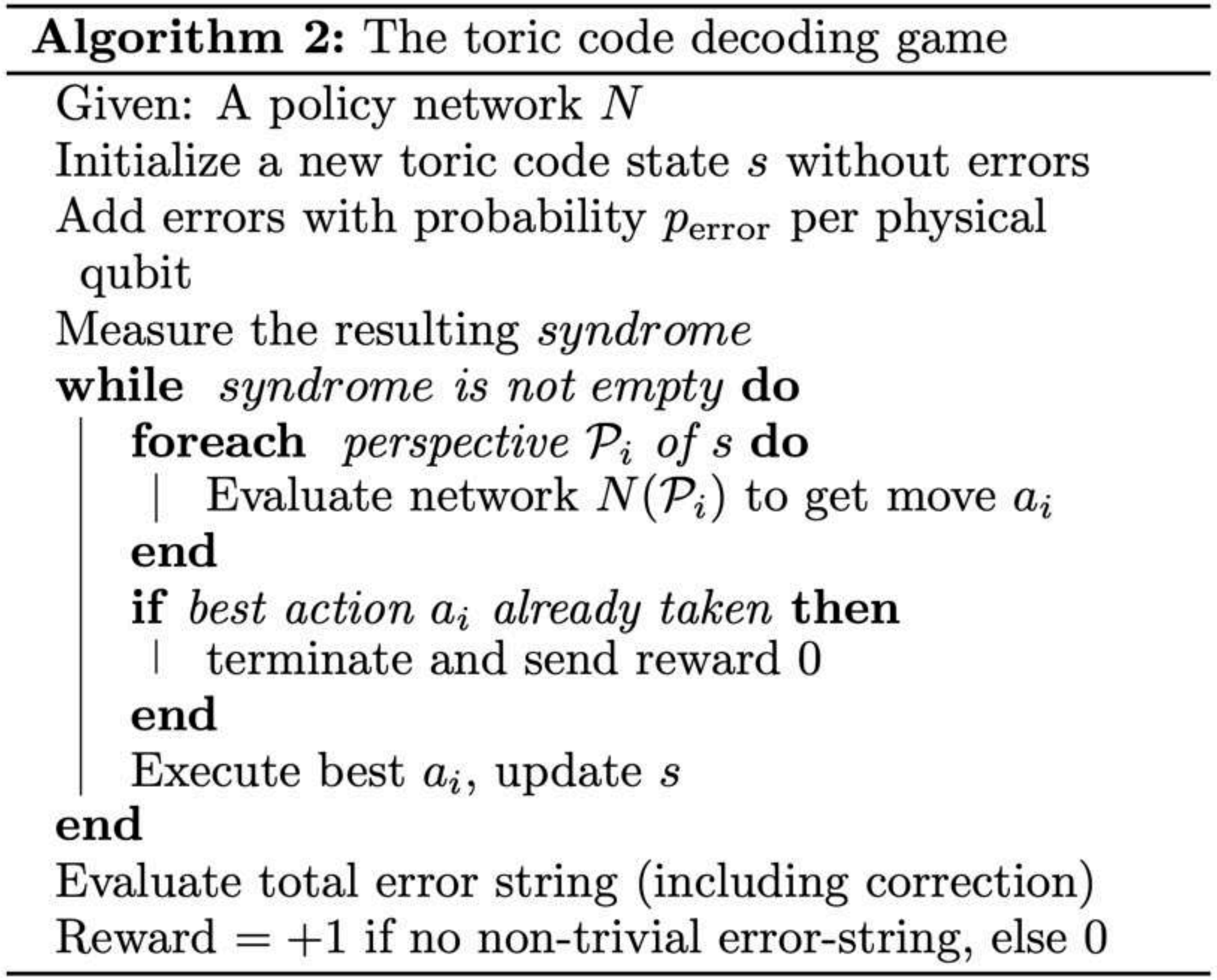
(12 actions)

Rewards

$$R_t = \begin{cases} 0 \text{ if syndrome} \\ -1 \text{ if logical error} \\ +1 \text{ if no logical error} \end{cases}$$

The Toric Game

As pseudocode



Algorithm 2: The toric code decoding game

```
Given: A policy network $N$
Initialize a new toric code state $s$ without errors
Add errors with probability $p_{\text{error}}$ per physical
  qubit
Measure the resulting syndrome
while syndrome is not empty do
    foreach perspective $\mathcal{P}_i$ of $s$ do
        Evaluate network $N(\mathcal{P}_i)$ to get move $a_i$
    end
    if best action $a_i$ already taken then
        terminate and send reward 0
    end
    Execute best $a_i$, update $s$
end
Evaluate total error string (including correction)
Reward $= +1$ if no non-trivial error-string, else 0
```

Gym

https://gym.openai.com/

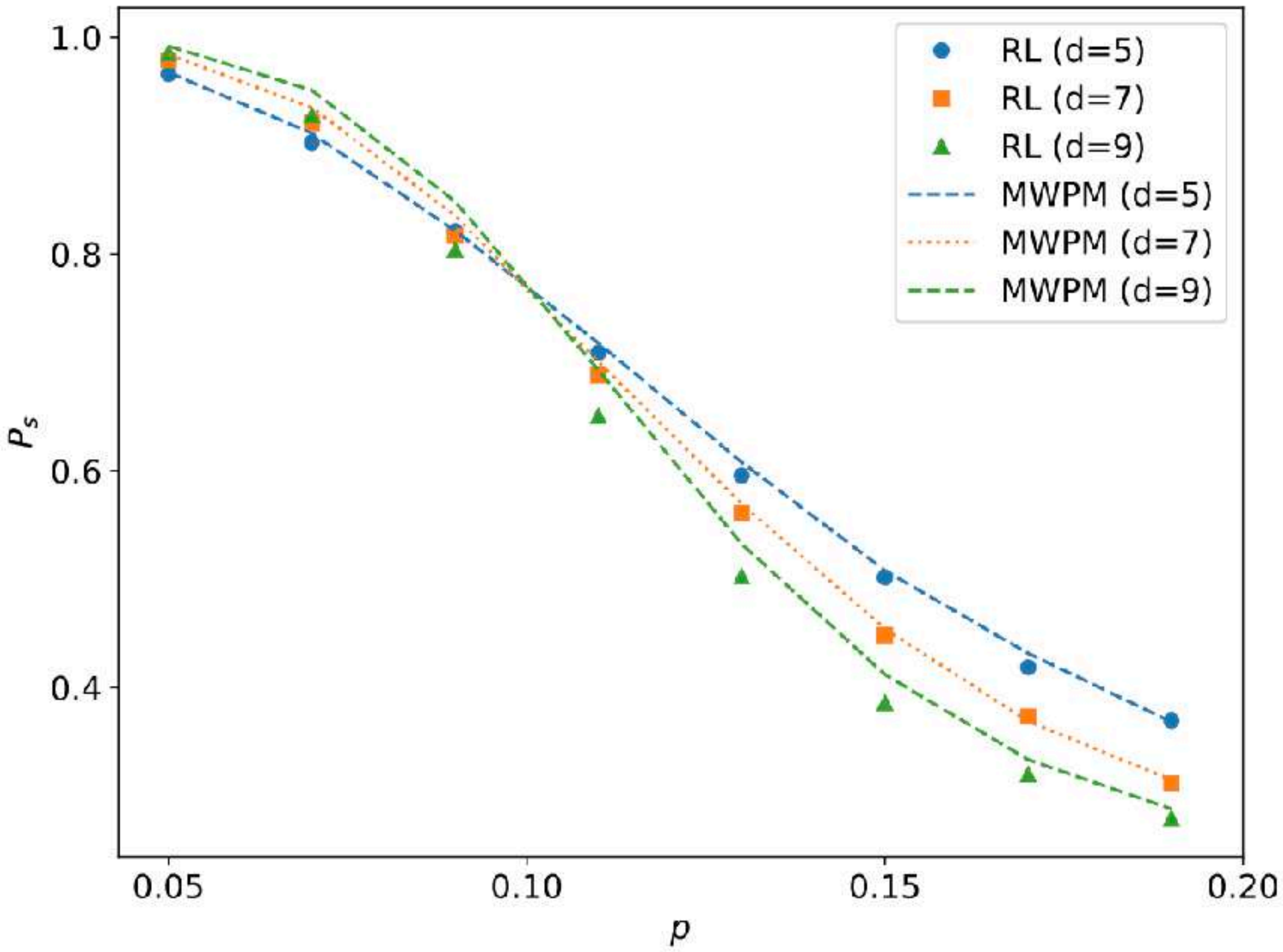
SciGym

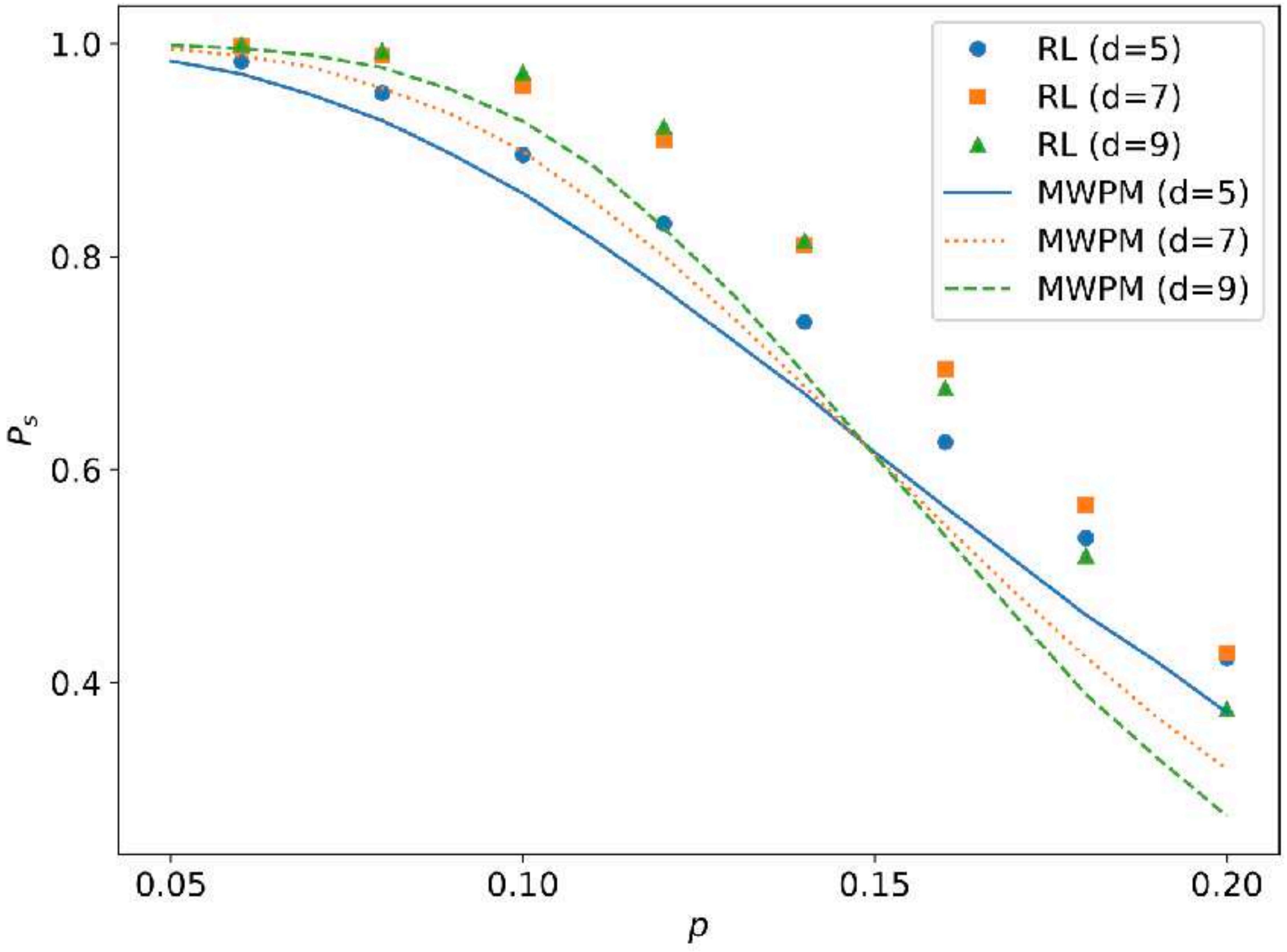
http://www.scigym.net/

Can an AI learn to play this game?

Yes it can!

It does better than MWPM for depolarising noise!

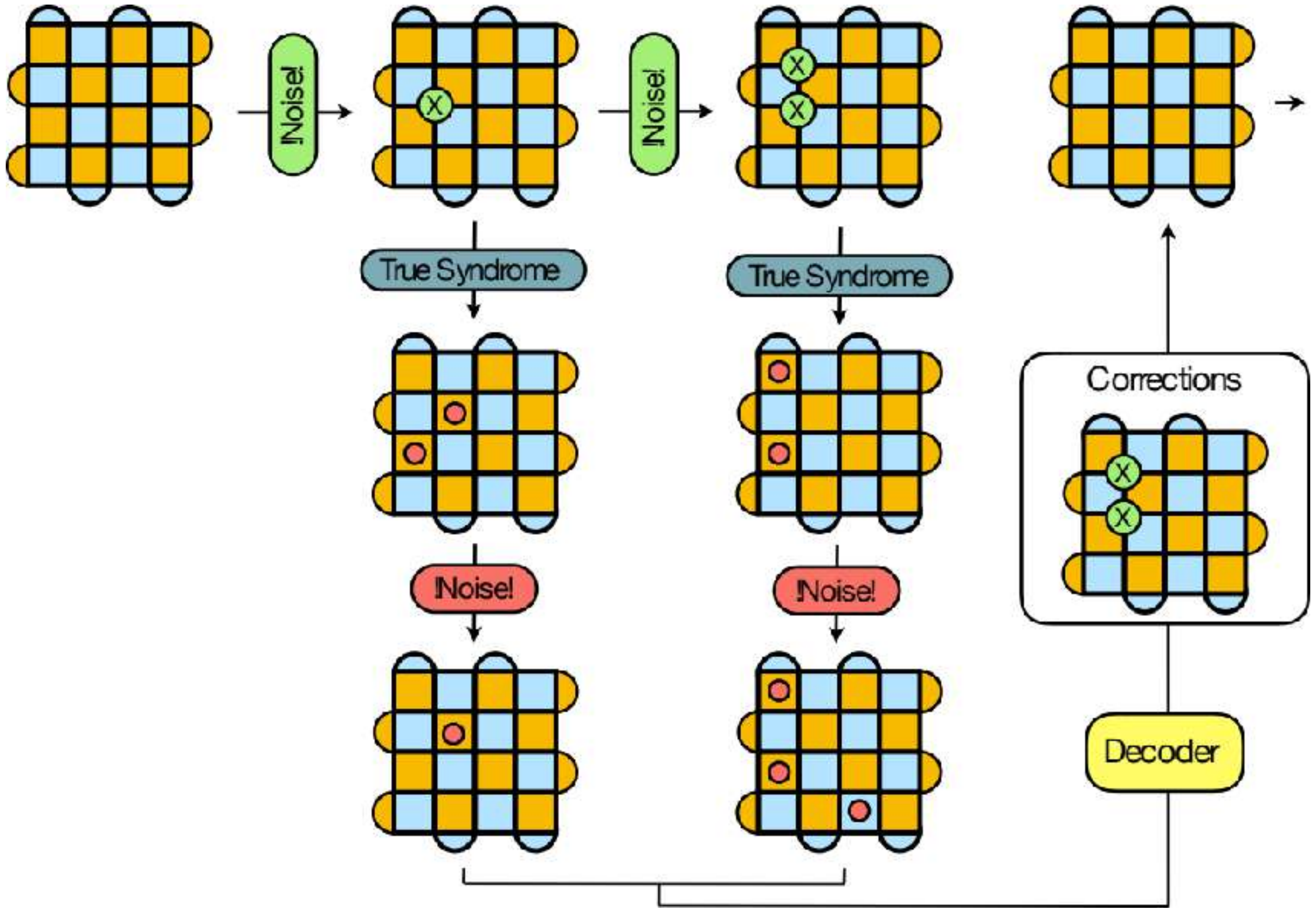


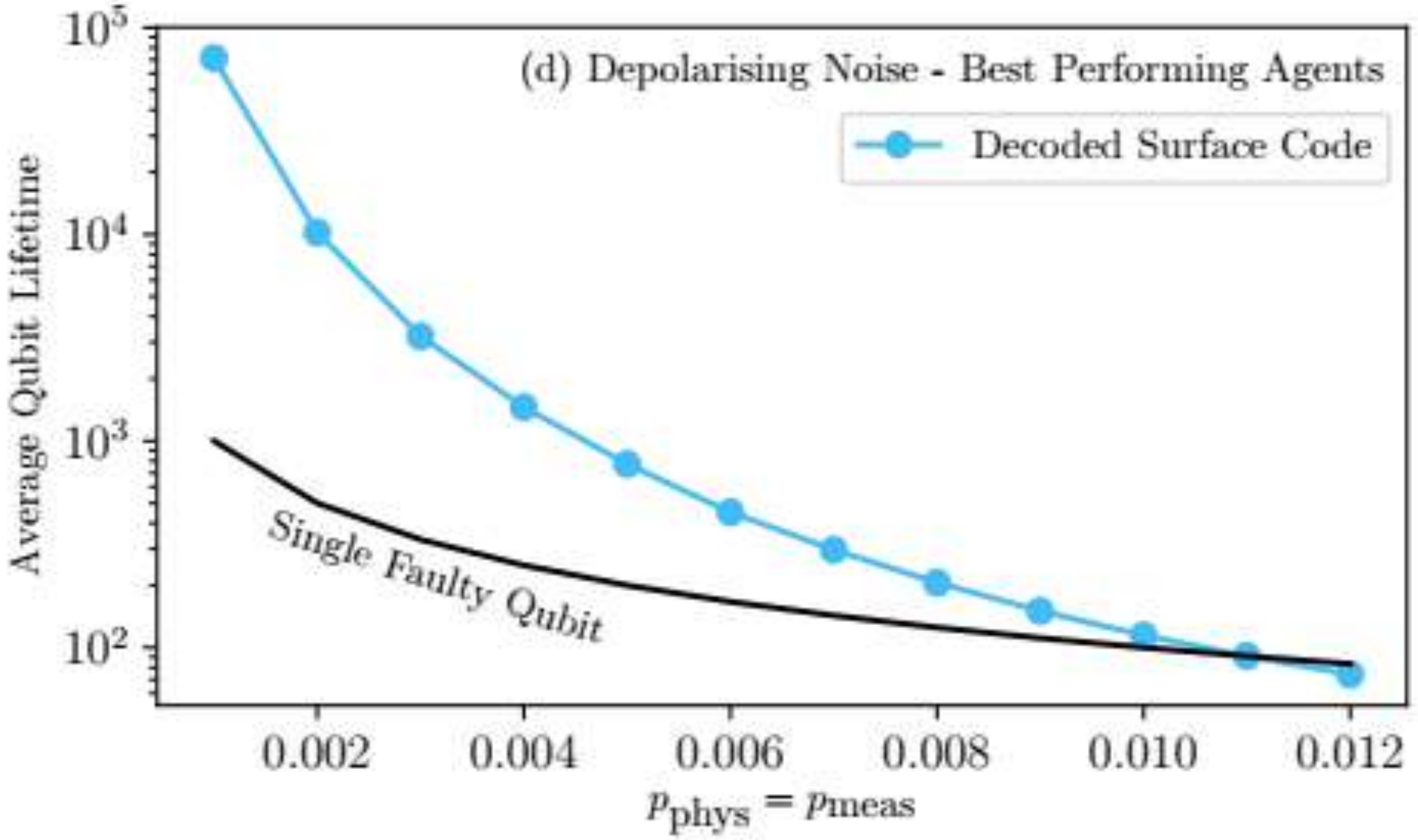


Distance 5 and 7 required 900.000 and 9.000.000 parameters in the network!

Physical Review Research 2, 023230 (2020)

Even works for faulty measurements





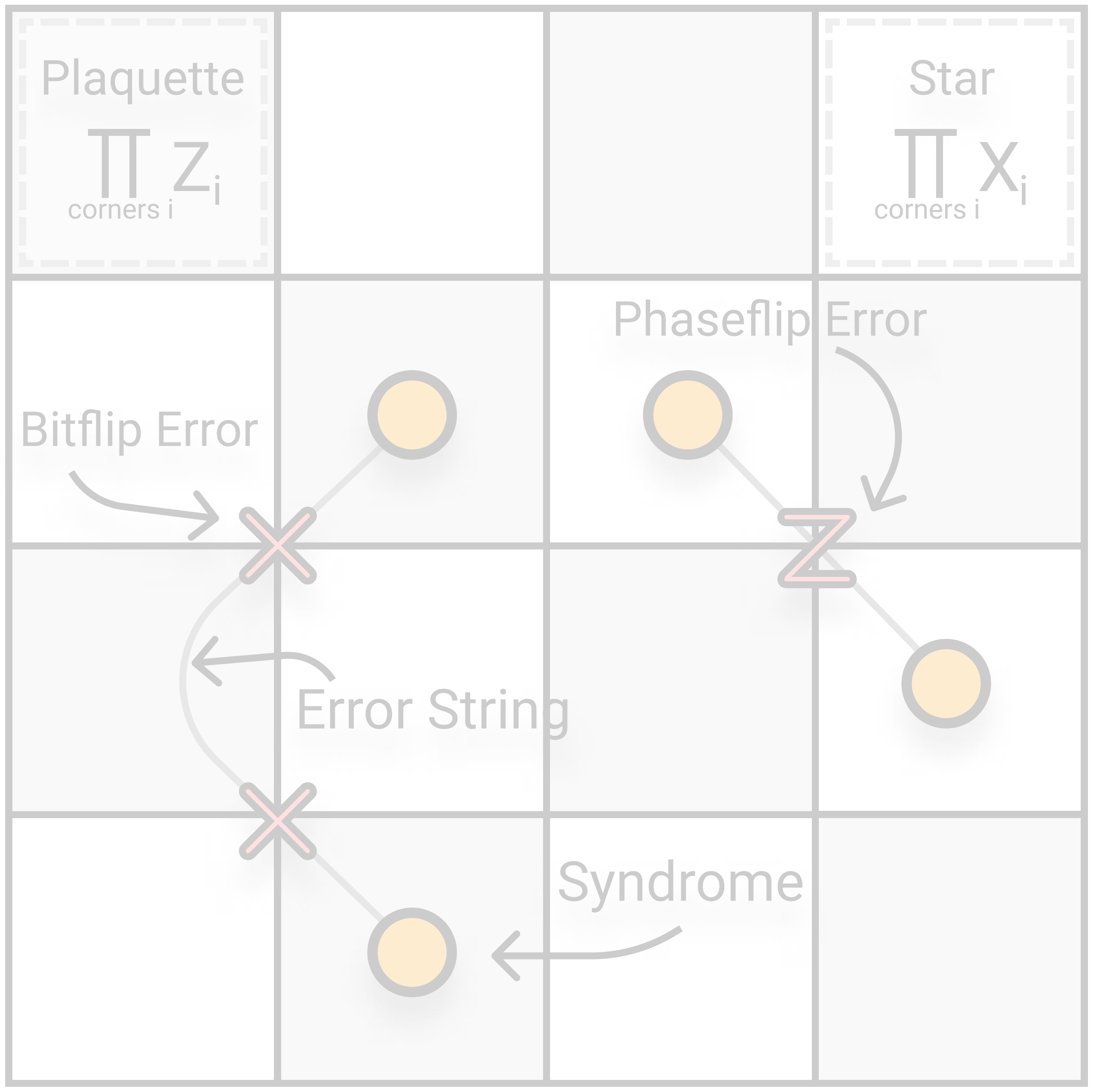
Distance 5 required 2.000.000 parameters in the network!

Mach. Learn. Sci. Technol. 2, 025005 (2021)

There are three main concepts for this talk

Don’t hesitate to ask!

Stabilizer codes

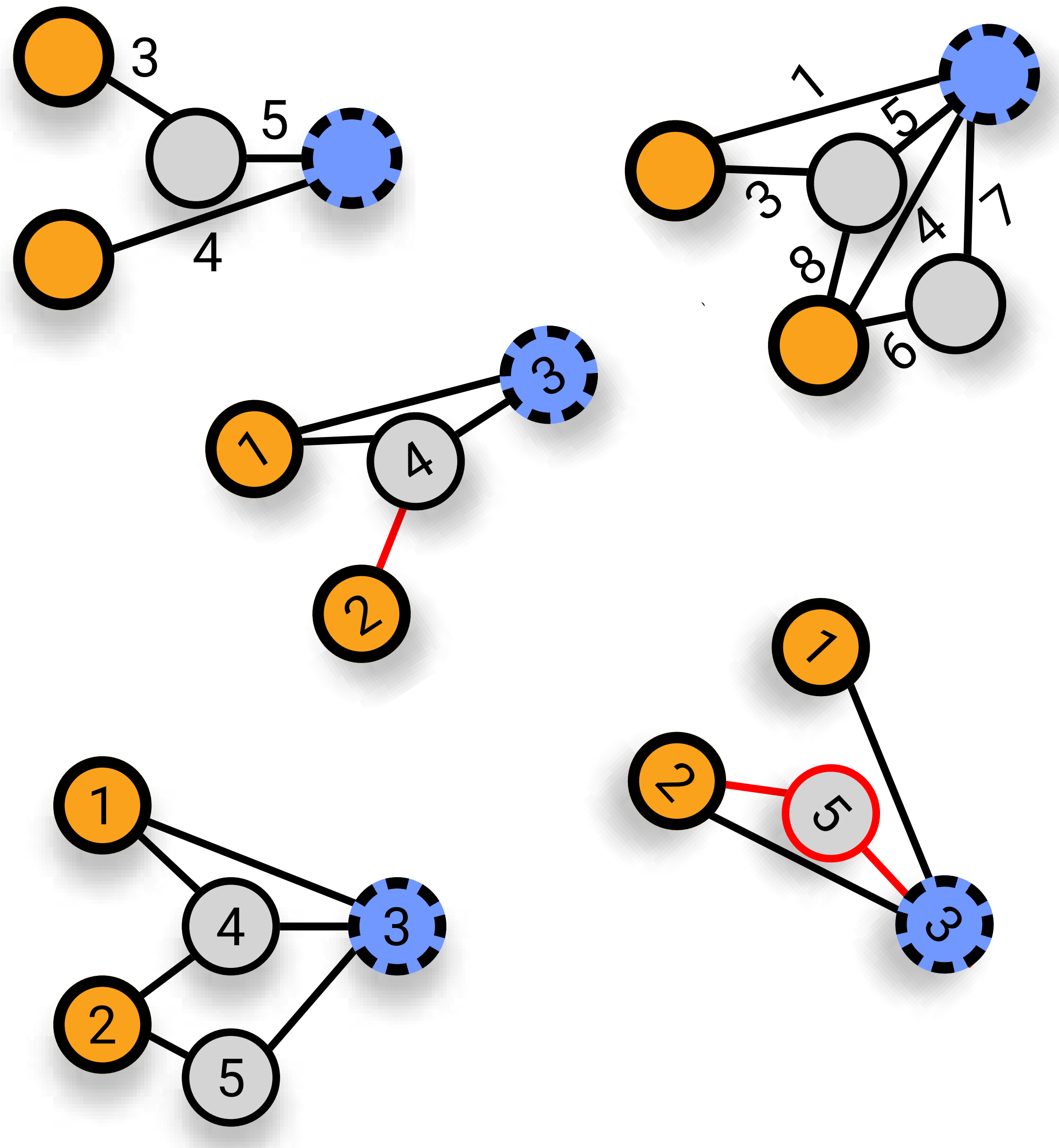

Plaquette
$\prod_{\text{corners } i} Z_i$
Star
$\prod_{\text{corners } i} X_i$
Phaseflip Error
Bitflip Error
Z
Error String
Syndrome

Quantum Computation

Reinforcement Learning

Deep Q-Network

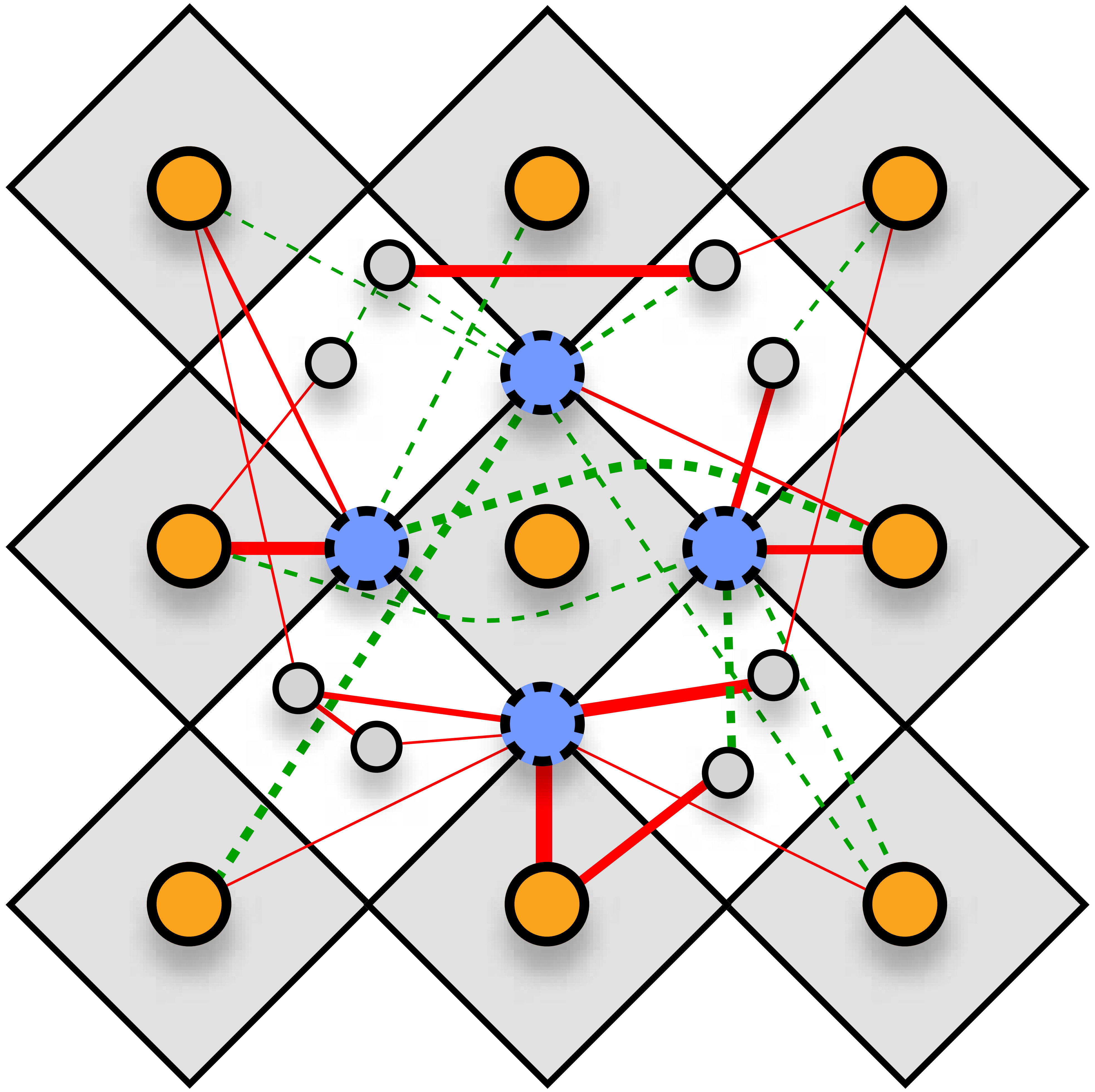
Evolutionary Strategy



Policy Networks

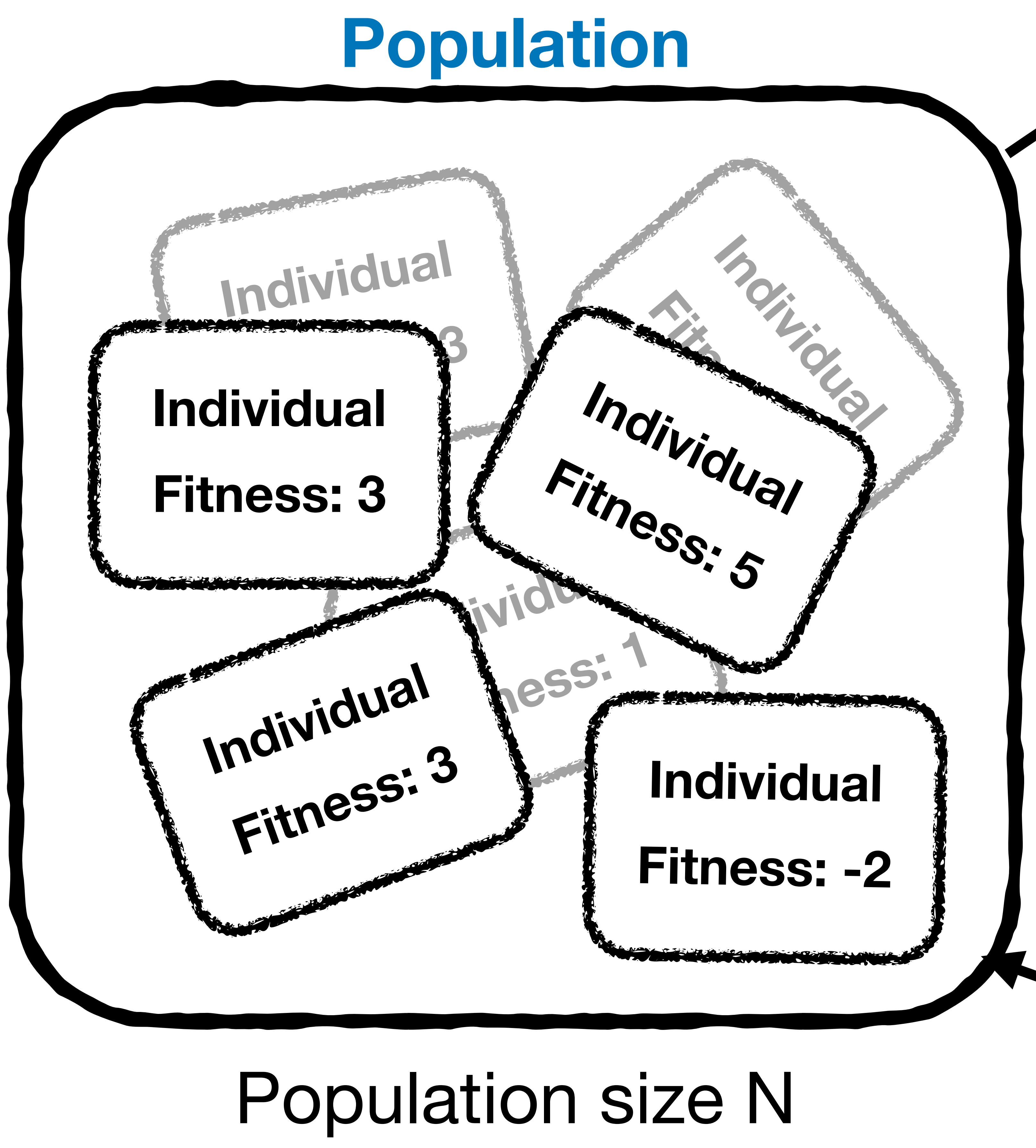
The NEAT algorithm

“Neural Evolution of Augmented Topologies”



What is “Neural evolution”?

First, look at ‘evolutionary strategies’ or ‘genetic algorithms’



```
Pick the top x% (by fitness)

While populationsize < N:
    ‘Combine’ two individuals (parents)
    and generate new individuals (offspring)

For each individual:
    With probability p,
    apply one of multiple ‘mutations'

Re-calculate fitness for each individual
```

A quick example

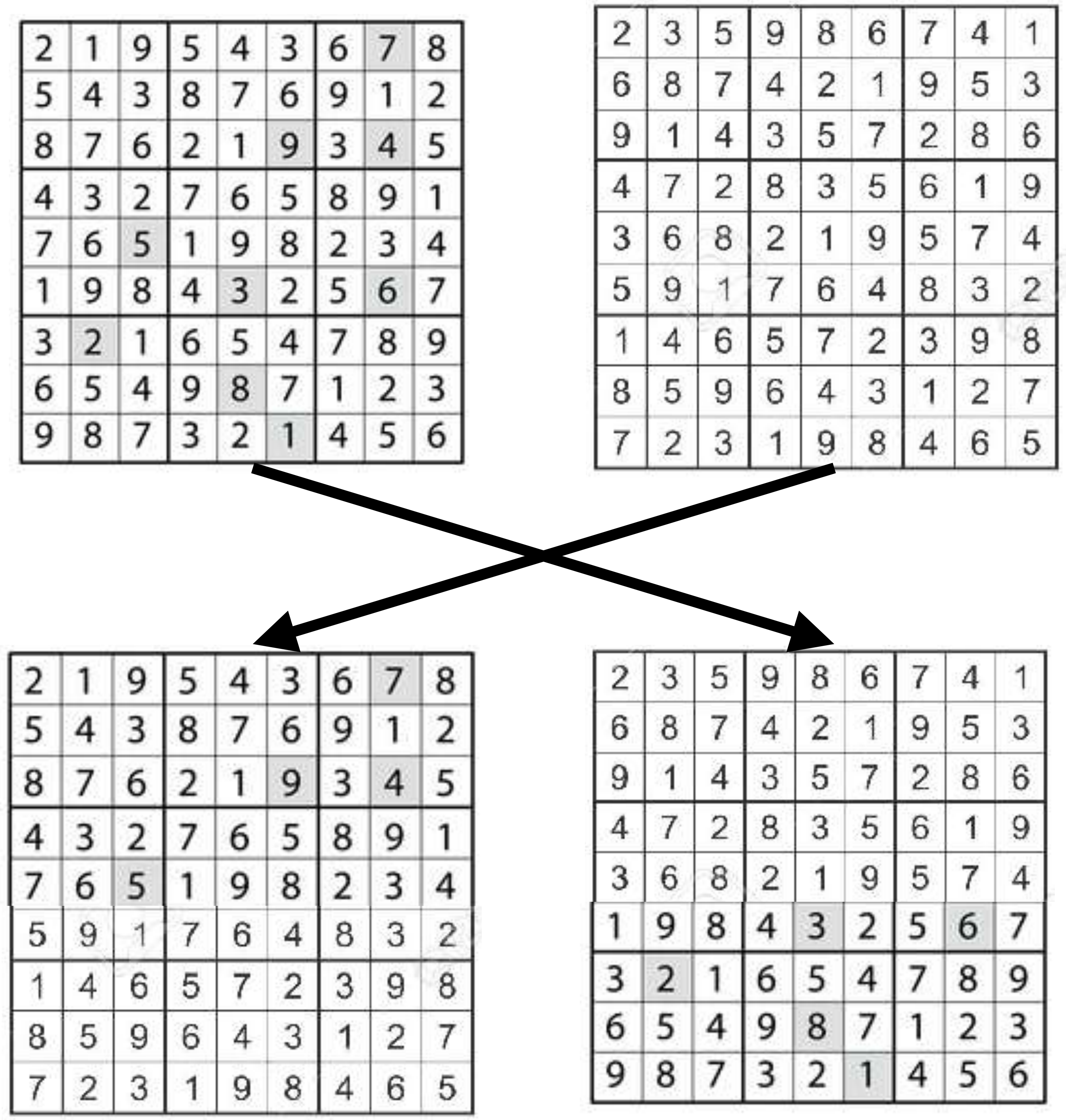
A genetic sudoku generator

Individual: Sudoku puzzle

Fitness: +1 for every row, column, block that is correct
(Or penalty for every violation?)

Cross-over example:
Take top 5 rows of parent 1, bottom 4 of parent 2
(and reverse)

Mutations: randomly change number



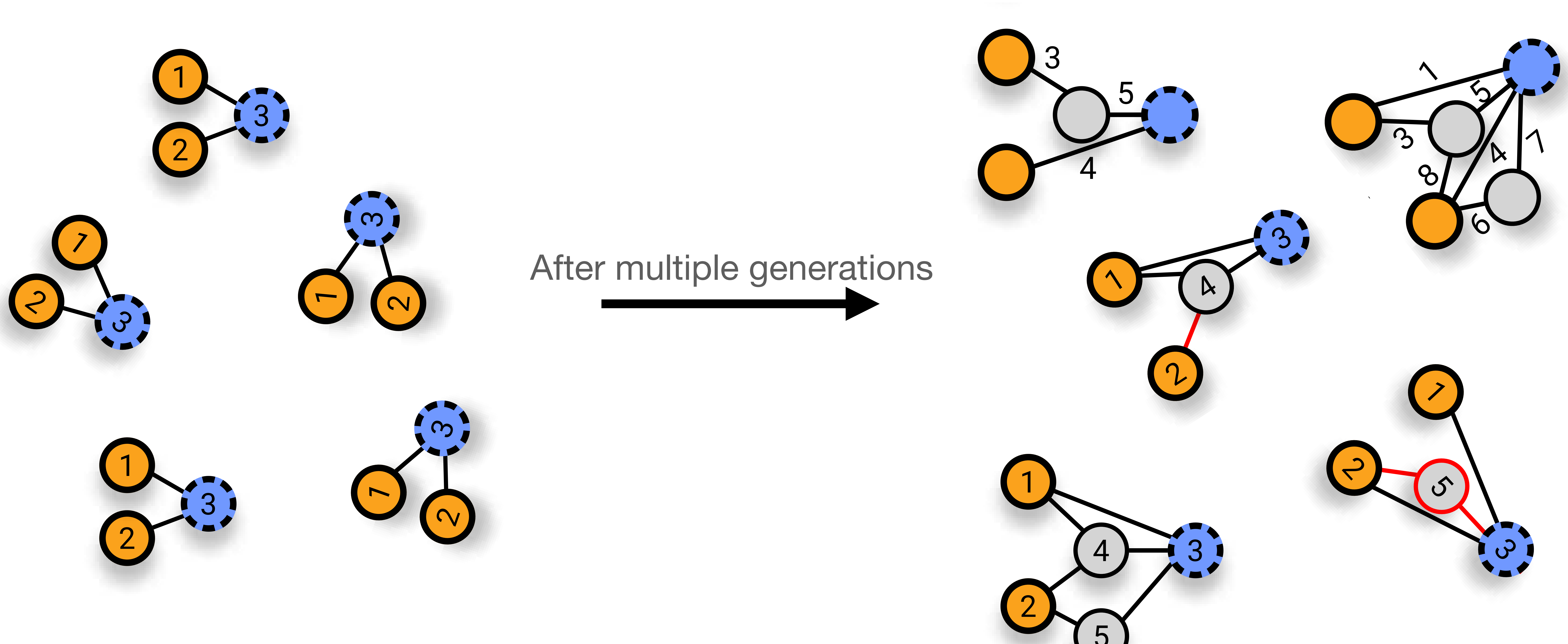
2	1	9	5	4	3	6	7	8
5	4	3	8	7	6	9	1	2
8	7	6	2	1	9	3	4	5
4	3	2	7	6	5	8	9	1
7	6	5	1	9	8	2	3	4
1	9	8	4	3	2	5	6	7
3	2	1	6	5	4	7	8	9
6	5	4	9	8	7	1	2	3
9	8	7	3	2	1	4	5	6

2	3	5	9	8	6	7	4	1
6	8	7	4	2	1	9	5	3
9	1	4	3	5	7	2	8	6
4	7	2	8	3	5	6	1	9
3	6	8	2	1	9	5	7	4
5	9	1	7	6	4	8	3	2
1	4	6	5	7	2	3	9	8
8	5	9	6	4	3	1	2	7
7	2	3	1	9	8	4	6	5

2	1	9	5	4	3	6	7	8
5	4	3	8	7	6	9	1	2
8	7	6	2	1	9	3	4	5
4	3	2	7	6	5	8	9	1
7	6	5	1	9	8	2	3	4
5	9	1	7	6	4	8	3	2
1	4	6	5	7	2	3	9	8
8	5	9	6	4	3	1	2	7
7	2	3	1	9	8	4	6	5

2	3	5	9	8	6	7	4	1
6	8	7	4	2	1	9	5	3
9	1	4	3	5	7	2	8	6
4	7	2	8	3	5	6	1	9
3	6	8	2	1	9	5	7	4
1	9	8	4	3	2	5	6	7
3	2	1	6	5	4	7	8	9
6	5	4	9	8	7	1	2	3
9	8	7	3	2	1	4	5	6

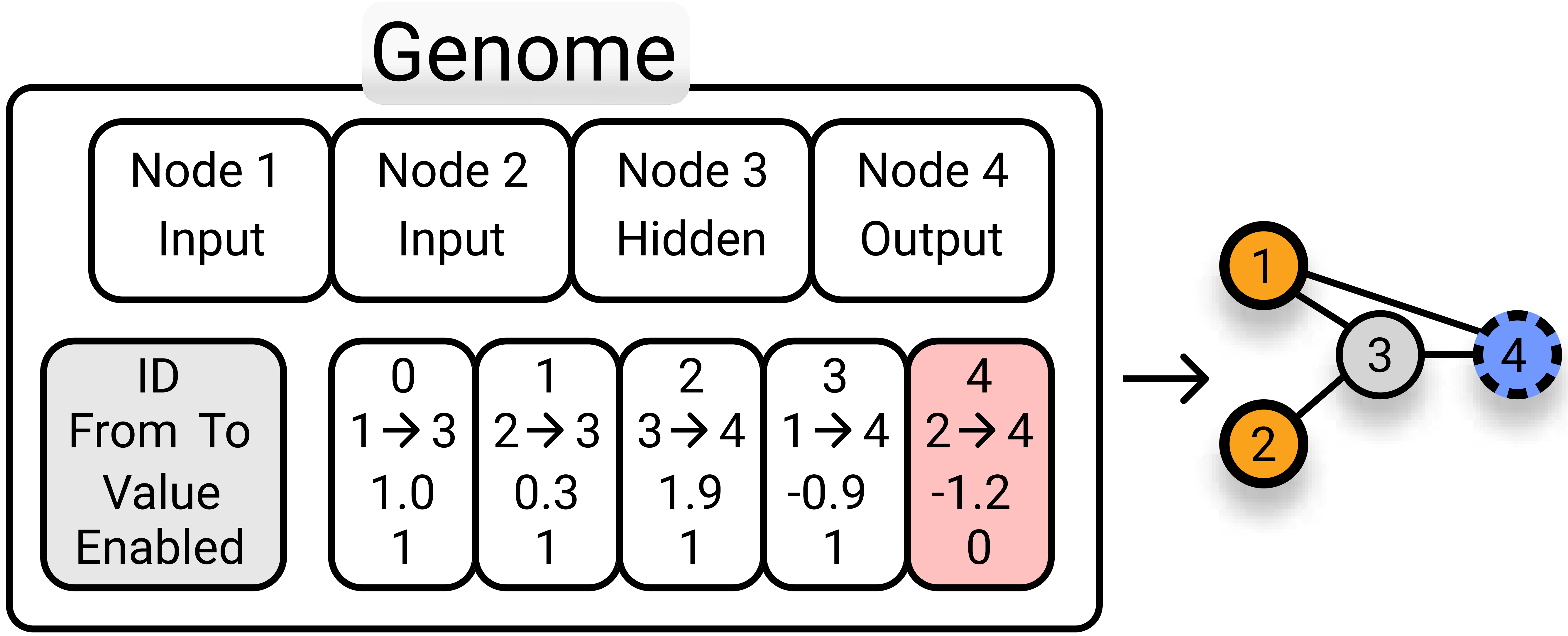
In NEAT, individuals are networks



After multiple generations

Typically ~100 members

The NEAT algorithm uses a genome



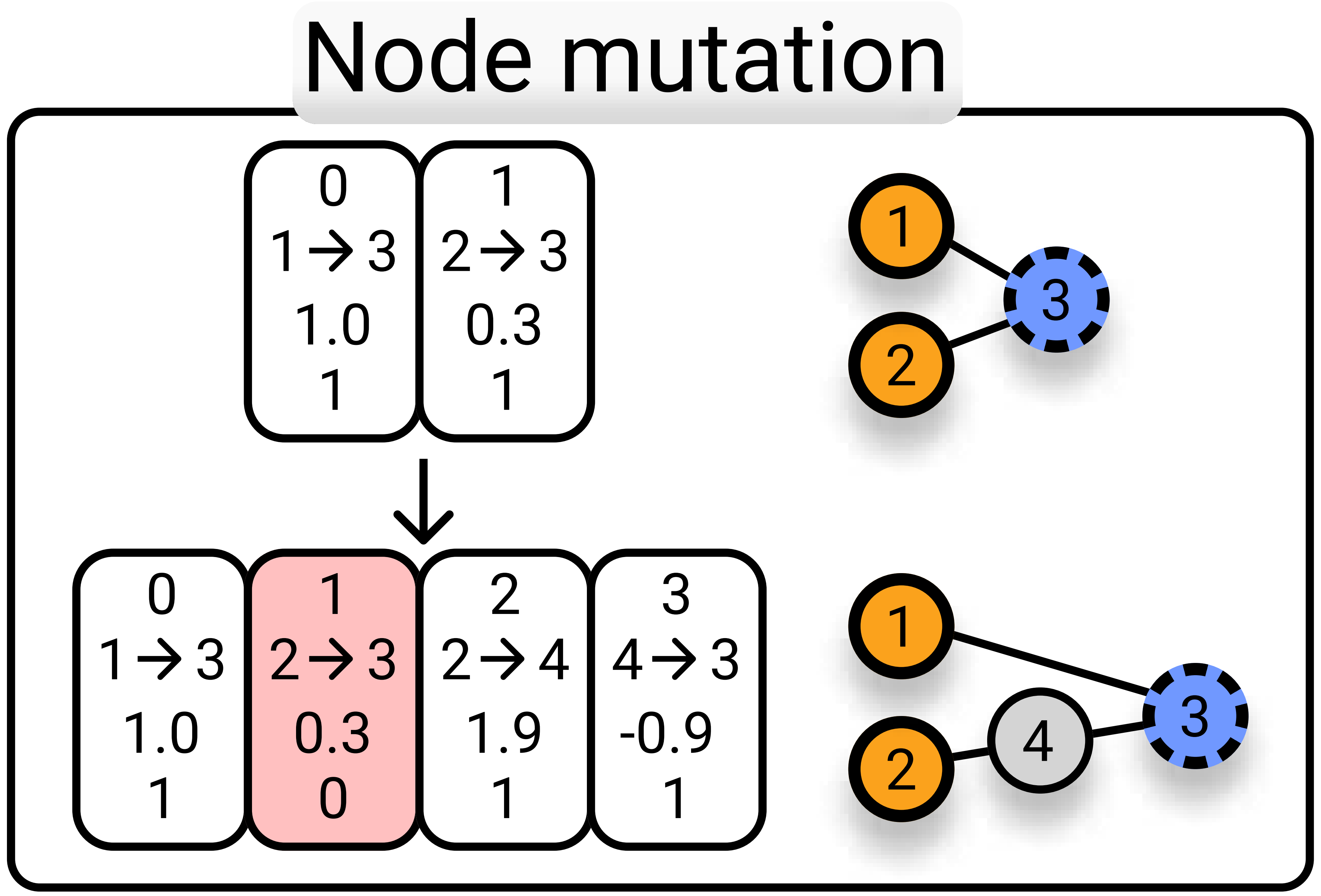
Genome

Node 1	Node 2	Node 3	Node 4
Input	Input	Hidden	Output

ID	0	1	2	3	4
From To	1 → 3	2 → 3	3 → 4	1 → 4	2 → 4
Value	1.0	0.3	1.9	-0.9	-1.2
Enabled	1	1	1	1	0

Genomes will get mutations

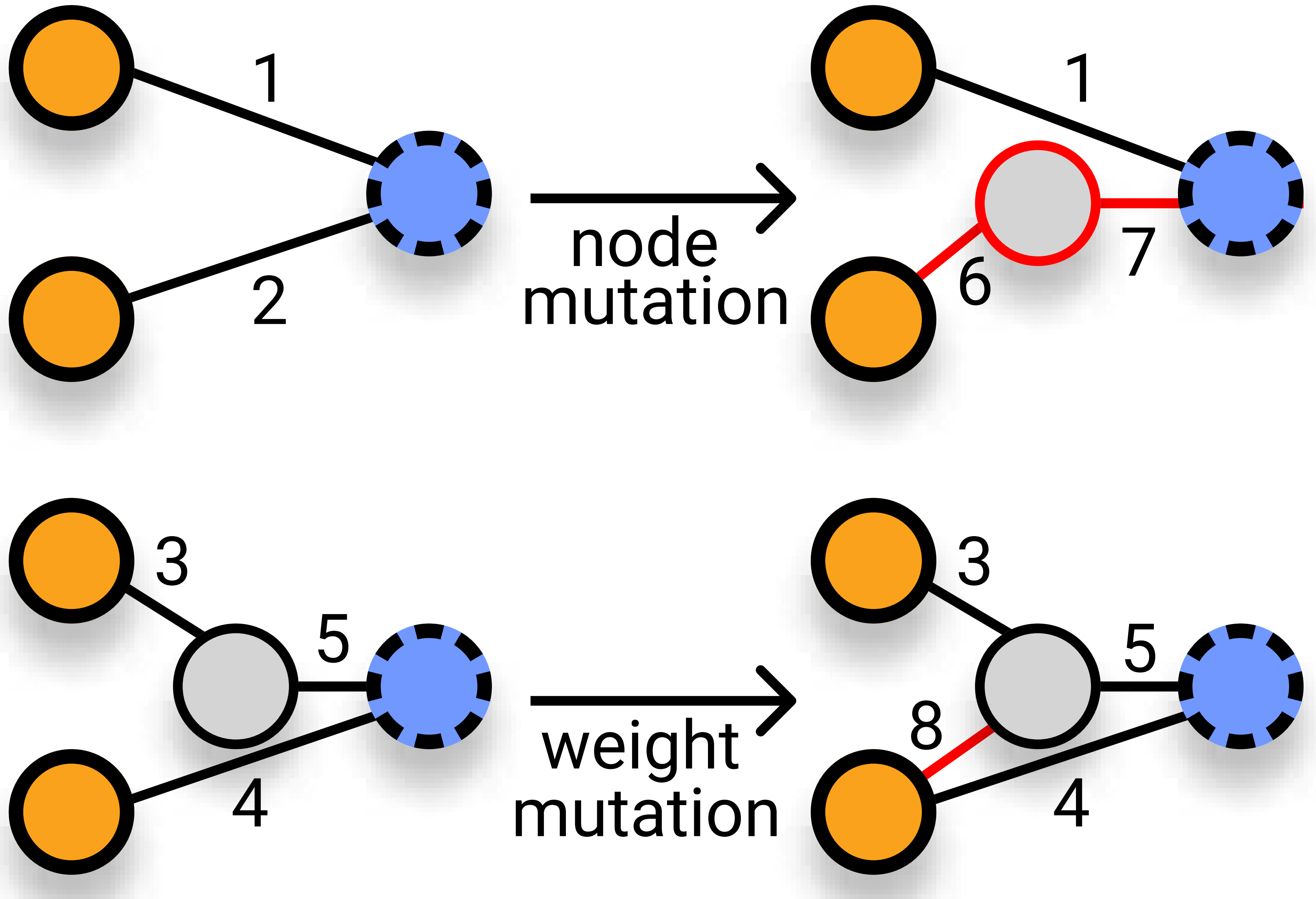
Randomly selected



Node mutation

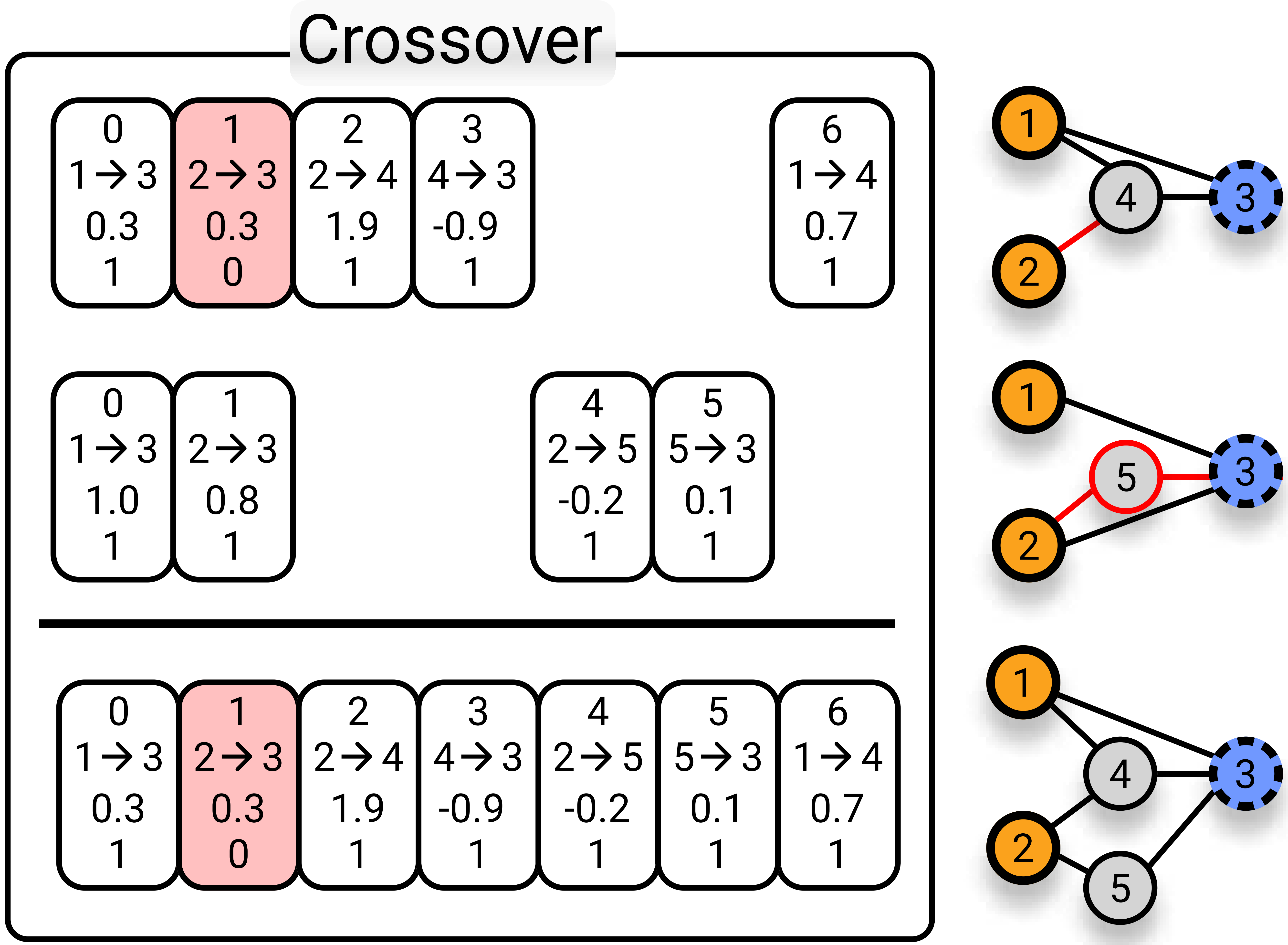
0	1
1→3	2→3
1.0	0.3
1	1

0	1	2	3
1→3	2→3	2→4	4→3
1.0	0.3	1.9	-0.9
1	0	1	1



Without crossover, this would be random

The key feature of NEAT is having a ‘meaningful’ crossover

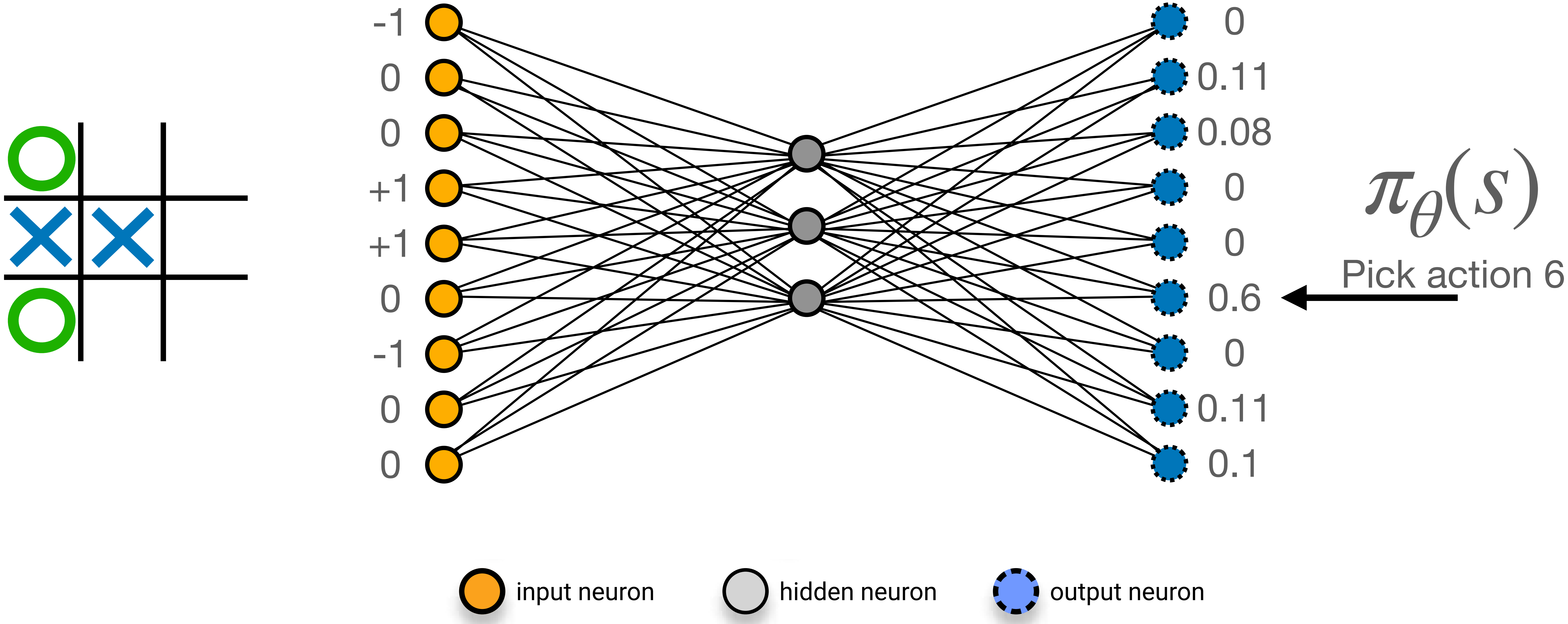


Crossover

0 1→3 0.3 1	1 2→3 0.3 0	2 2→4 1.9 1	3 4→3 -0.9 1			6 1→4 0.7 1
0 1→3 1.0 1	1 2→3 0.8 1			4 2→5 -0.2 1	5 5→3 0.1 1	
0 1→3 0.3 1	1 2→3 0.3 0	2 2→4 1.9 1	3 4→3 -0.9 1	4 2→5 -0.2 1	5 5→3 0.1 1	6 1→4 0.7 1

The networks are policy networks

As opposed to Q-learning



$\pi_\theta(s)$

Pick action 6

Features and advantages of NEAT

Compared to gradient methods and other networks

Gradient-free (good?)

Optimizes network architecture

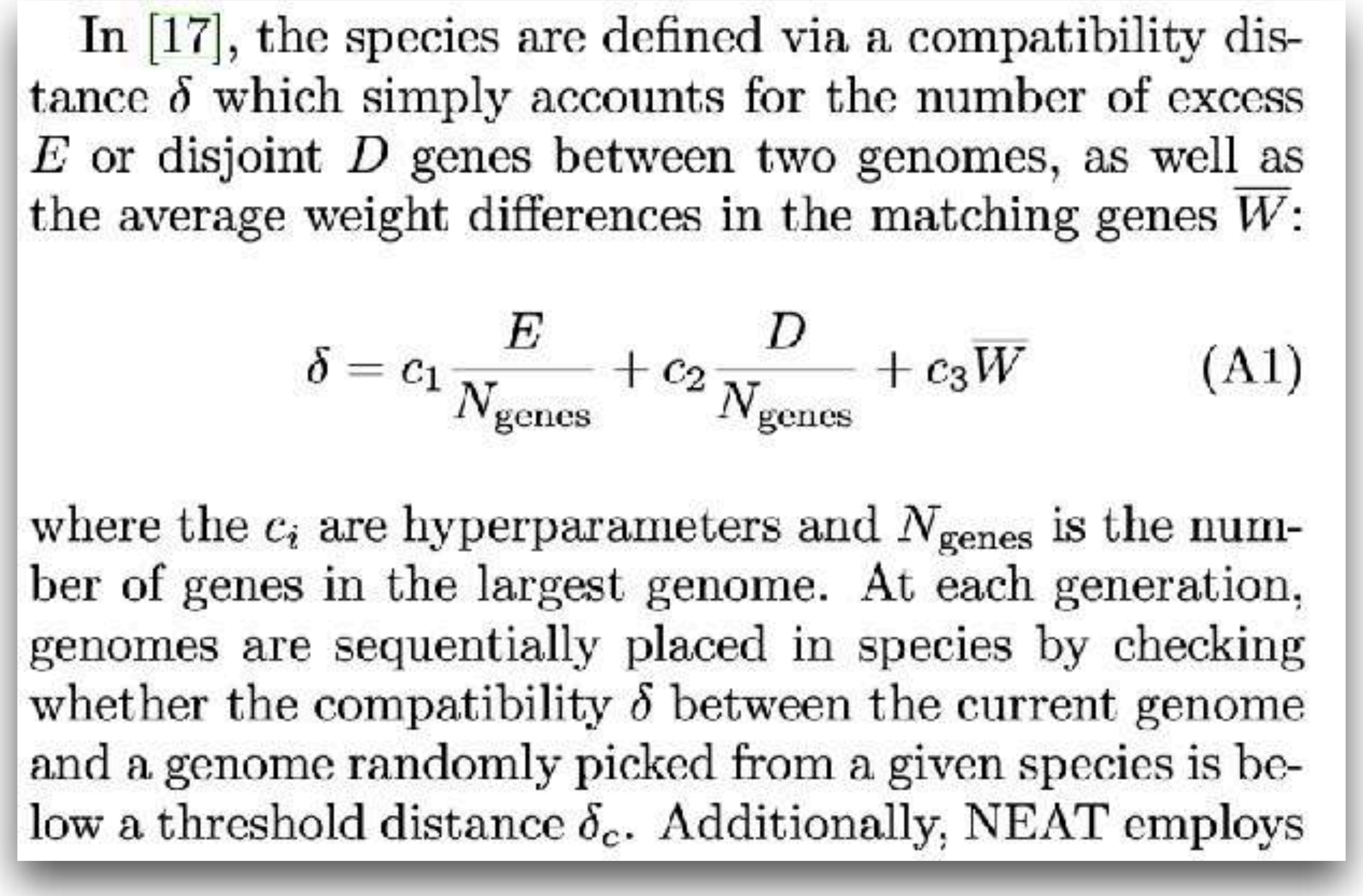
Start from simplest -> Add complexity when necessary!

Three to four orders of magnitude fewer parameters!

Uses speciation

A mutation often leads to a drop in fidelity at first

Straightforward ‘transfer learning’



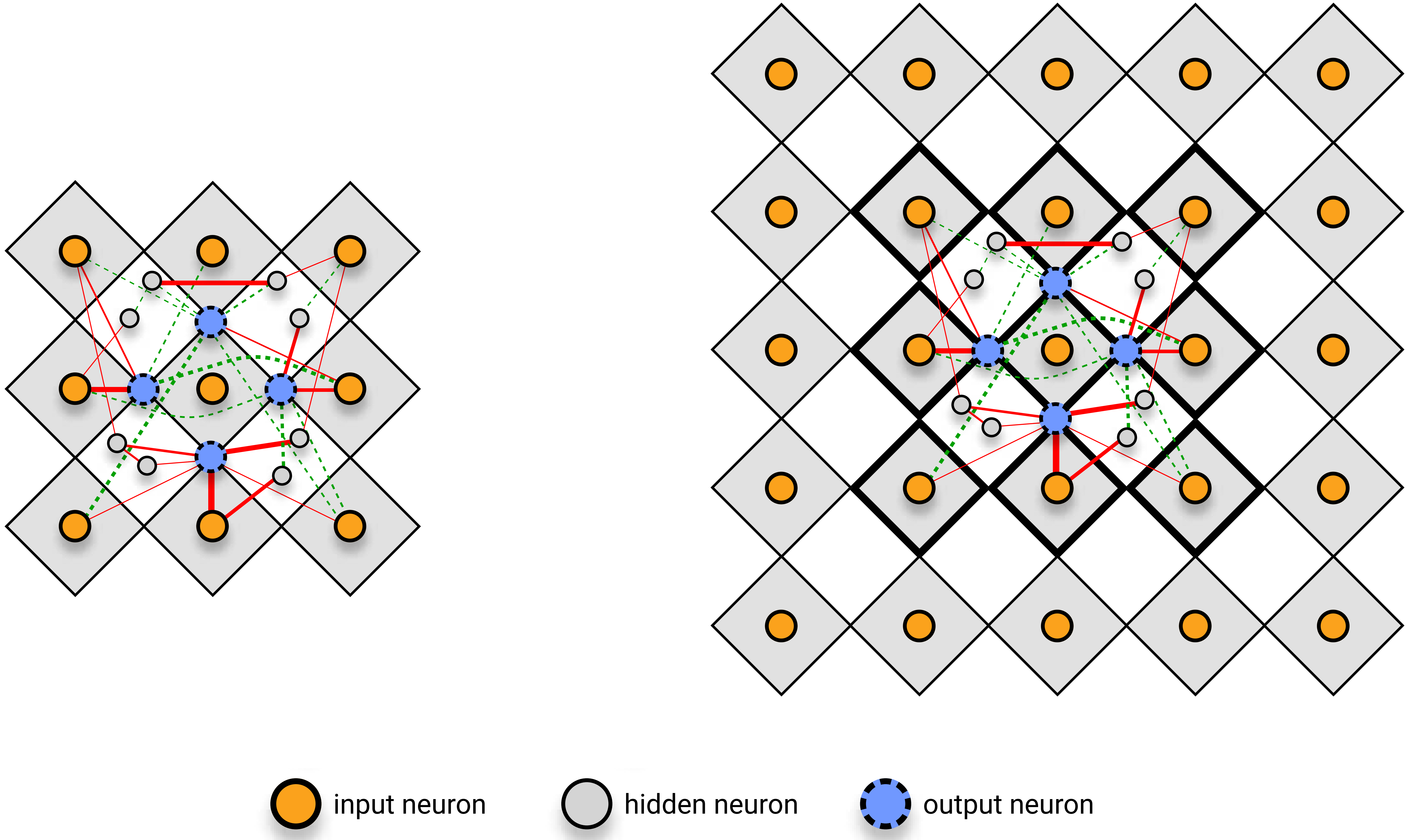
In [17], the species are defined via a compatibility distance δ which simply accounts for the number of excess E or disjoint D genes between two genomes, as well as the average weight differences in the matching genes $\overline{W}$:

$$\delta = c_1 \frac{E}{N_{\text{genes}}} + c_2 \frac{D}{N_{\text{genes}}} + c_3 \overline{W} \qquad \text{(A1)}$$

where the c_i are hyperparameters and N_{genes} is the number of genes in the largest genome. At each generation, genomes are sequentially placed in species by checking whether the compatibility δ between the current genome and a genome randomly picked from a given species is below a threshold distance δ_c. Additionally, NEAT employs

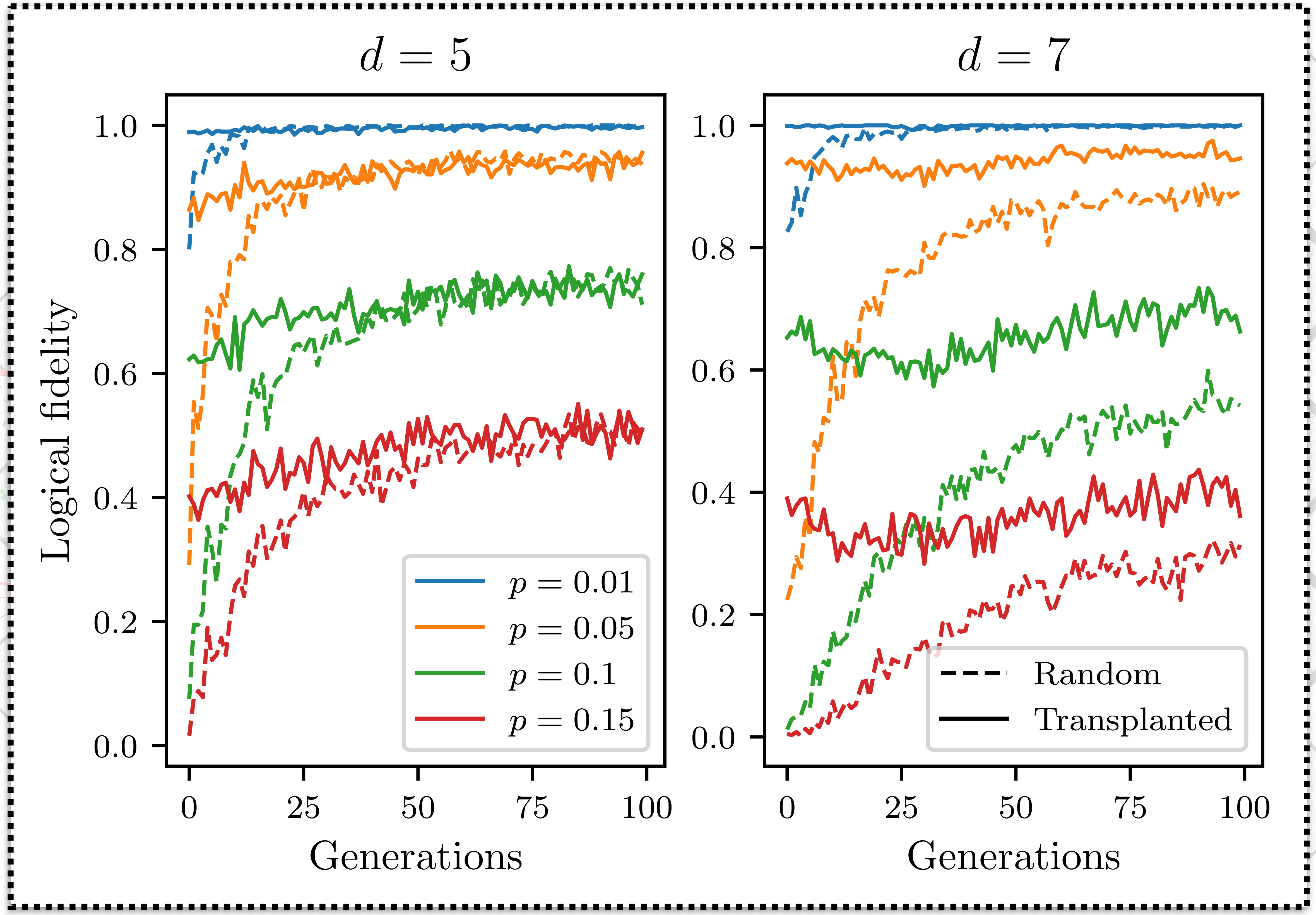
Transfer learning

Transplanting solutions due to genome representation!



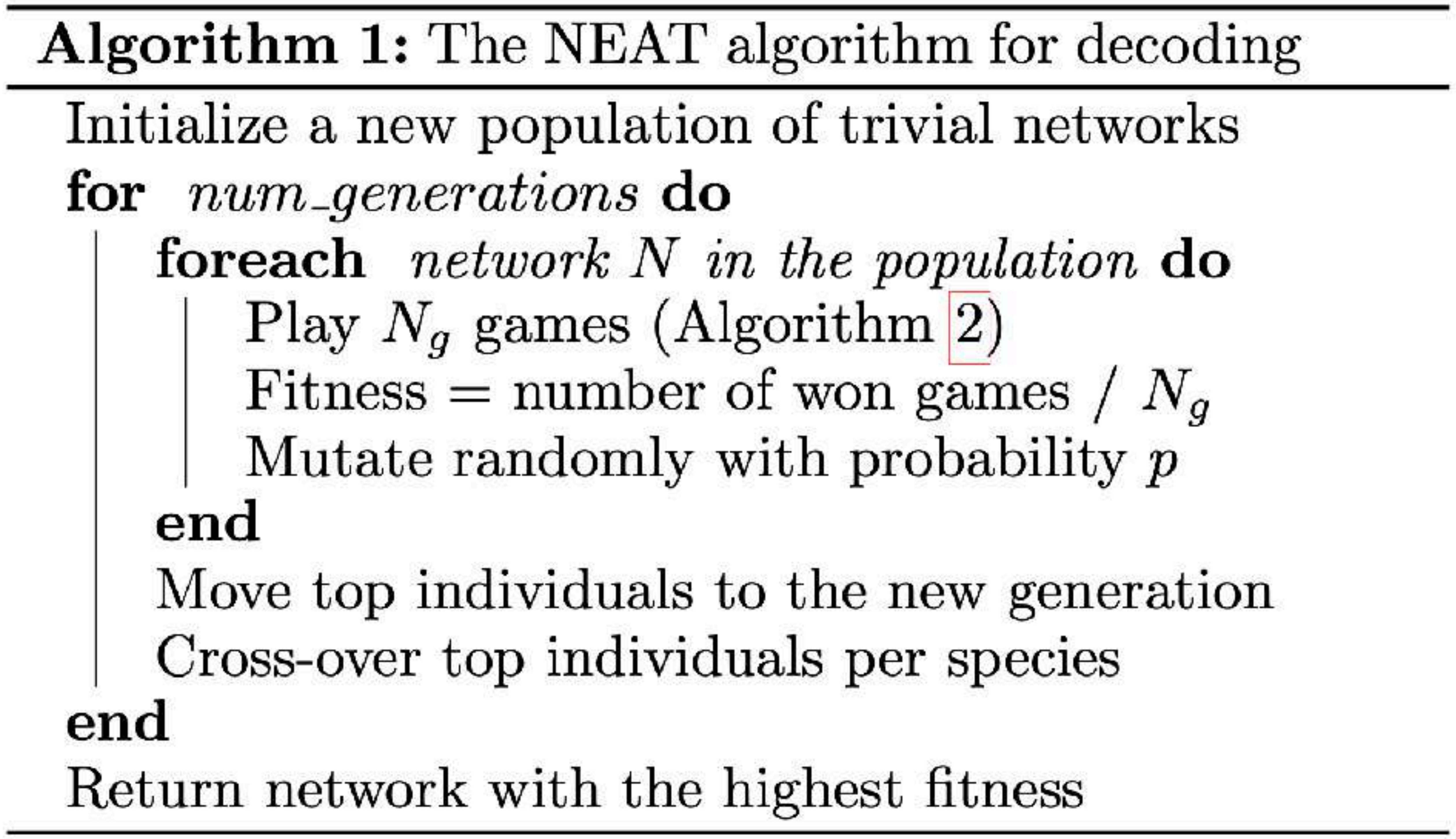
Transfer learning

Transplanting solutions due to genome representation!



The NEAT Algorithm

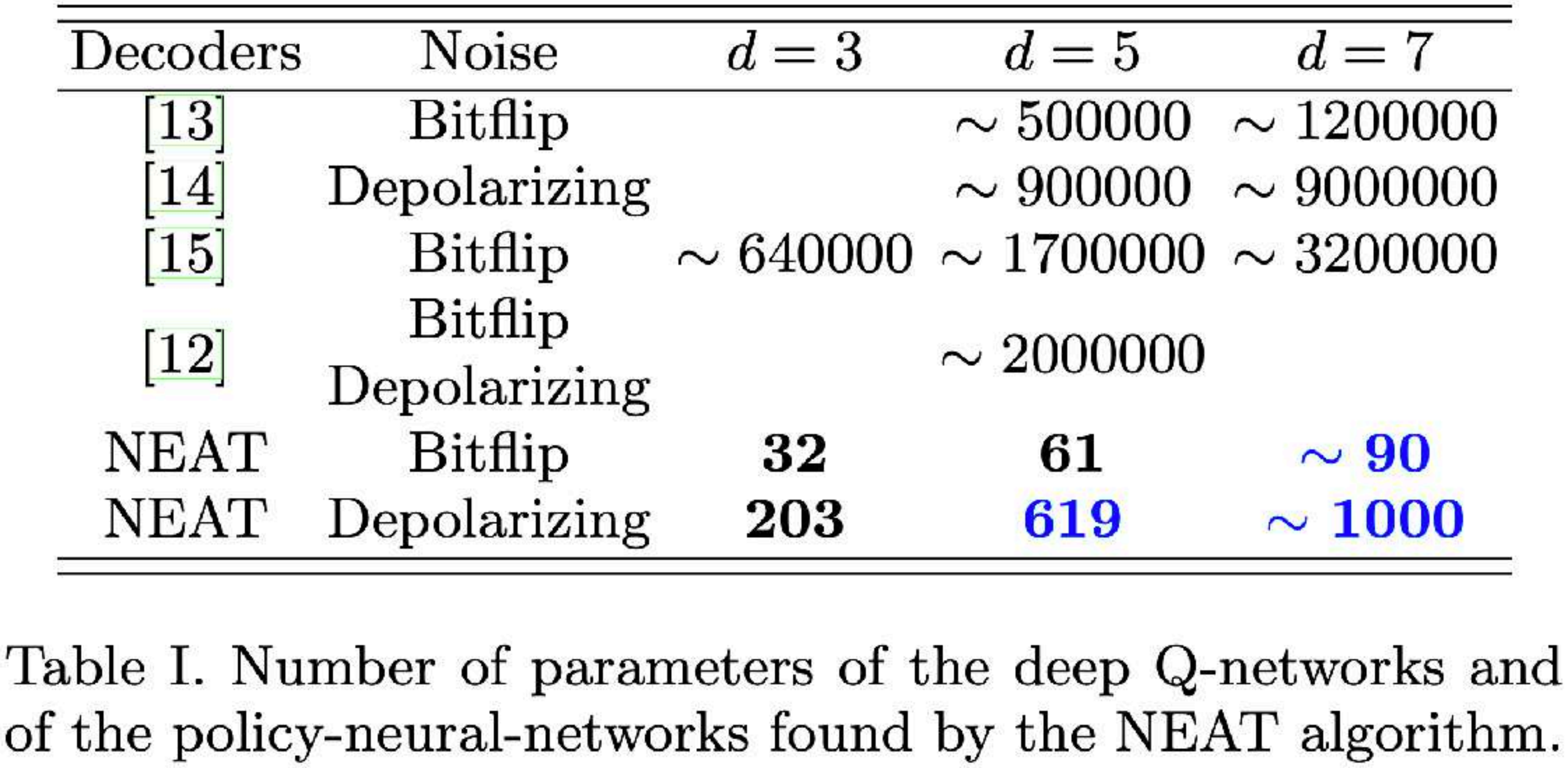
As pseudocode



```
Algorithm 1: The NEAT algorithm for decoding
Initialize a new population of trivial networks
for num_generations do
    foreach network N in the population do
        Play $N_g$ games (Algorithm 2)
        Fitness = number of won games / $N_g$
        Mutate randomly with probability $p$
    end
    Move top individuals to the new generation
    Cross-over top individuals per species
end
Return network with the highest fitness
```

Learning = Optimising the weights

Typically done using gradient descent



Decoders	Noise	$d = 3$	$d = 5$	$d = 7$
[13]	Bitflip		$\sim$ 500000	$\sim$ 1200000
[14]	Depolarizing		$\sim$ 900000	$\sim$ 9000000
[15]	Bitflip	$\sim$ 640000	$\sim$ 1700000	$\sim$ 3200000
[12]	Bitflip Depolarizing		$\sim$ 2000000	
NEAT	Bitflip	**32**	**61**	**$\sim$ 90**
NEAT	Depolarizing	**203**	**619**	**$\sim$ 1000**

Table I. Number of parameters of the deep Q-networks and of the policy-neural-networks found by the NEAT algorithm.

[12] R. Sweke, M. S. Kesselring, E. P. L. van Nieuwenburg, and J. Eisert, Reinforcement learning decoders for faulttolerant quantum computation, Machine Learning: Science and Technology 2, 025005 (2021).

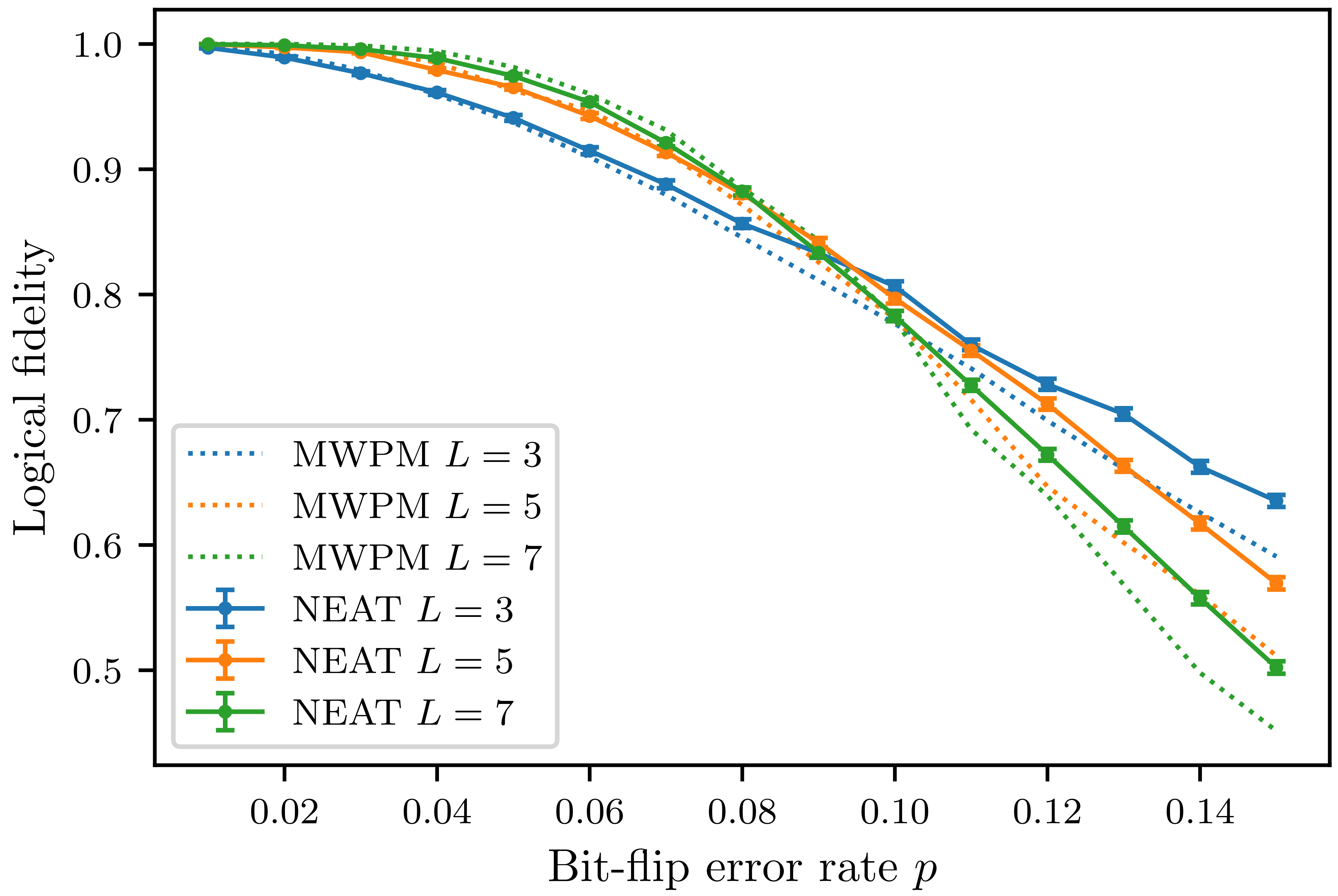
[13] L. Domingo Colomer, M. Skotiniotis, and R. Mu˜nozTapia, Reinforcement learning for optimal error correction of toric codes, Physics Letters A 384, 126353 (2020)

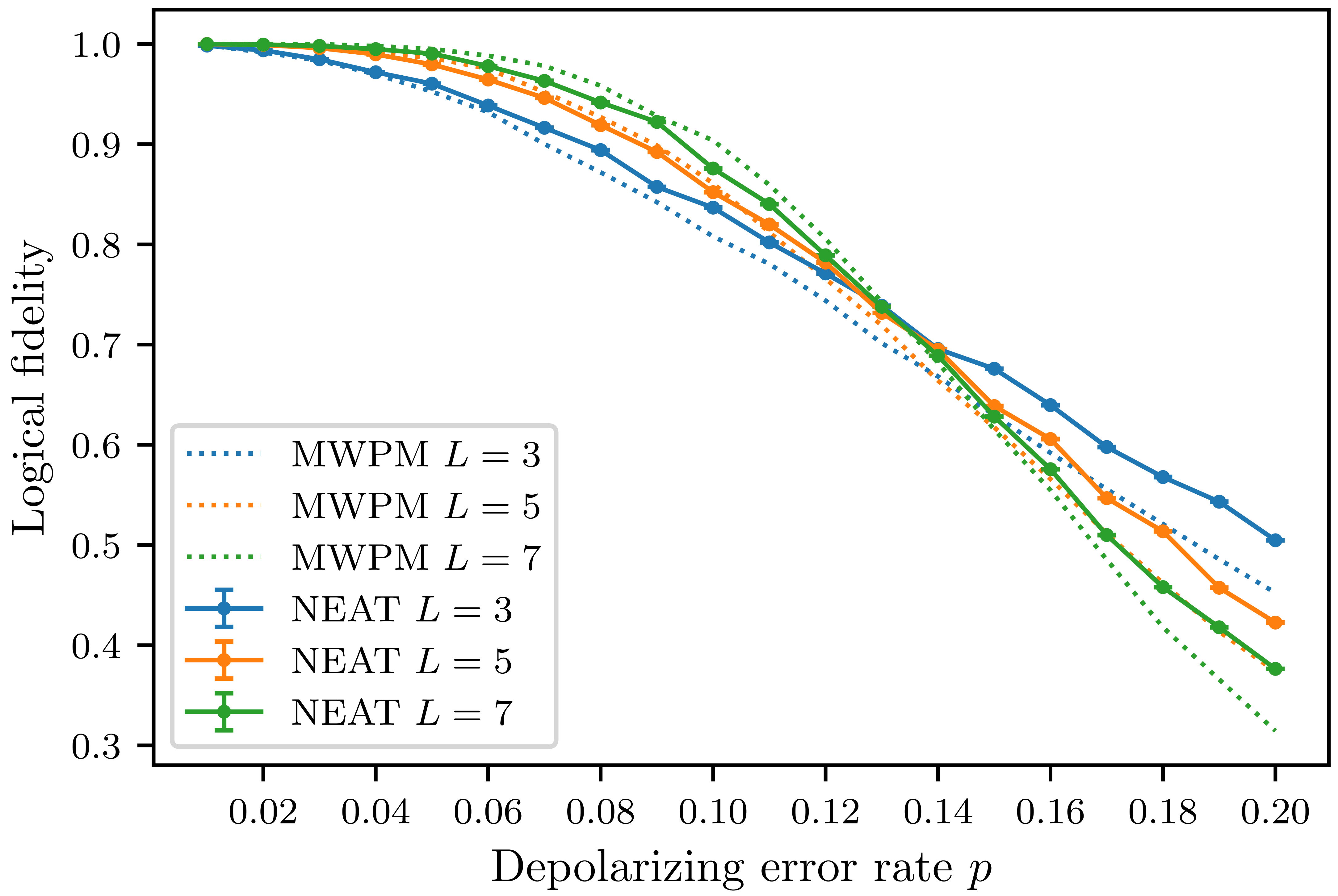
[14] D. Fitzek, M. Eliasson, A. F. Kockum, and M. Granath, Deep Q-learning decoder for depolarizing noise on the toric code, Phys. Rev. Research 2, 023230 (2020).

[15] P. Andreasson, J. Johansson, S. Liljestrand, and M. Granath, Quantum error correction for the toric code using deep reinforcement learning, Quantum 3, 183 (2019).

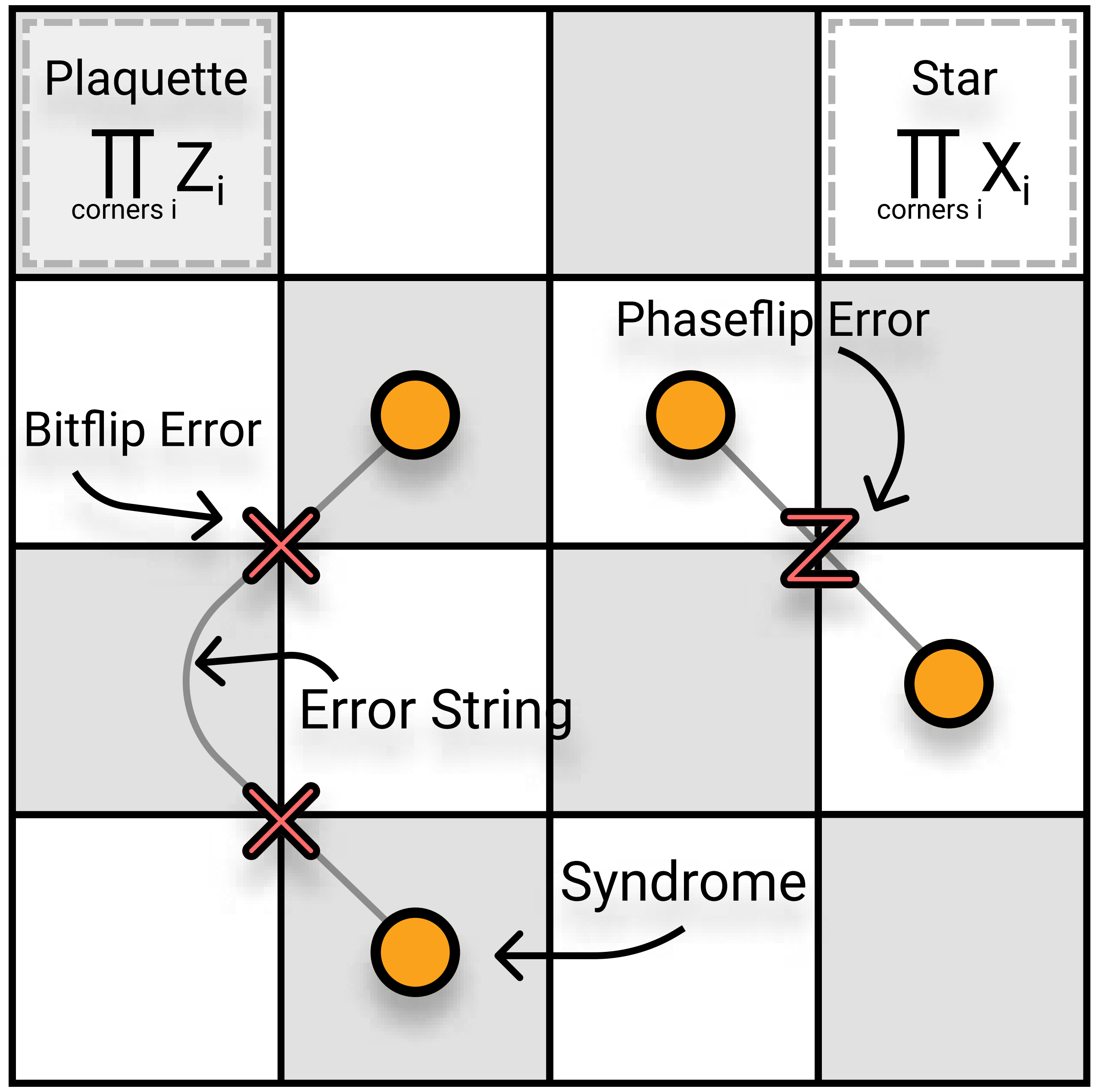
Our agent learns ~MWPM

The real challenge will be other codes + scaling up!



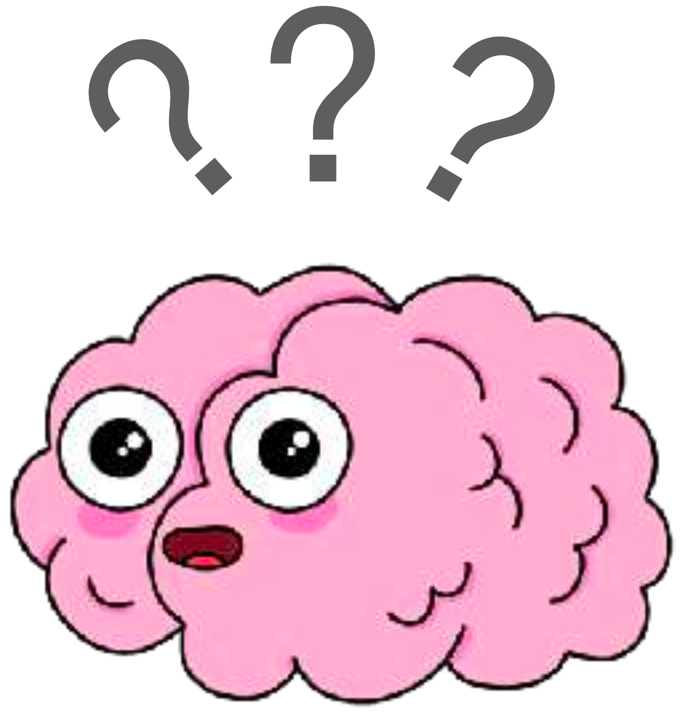


Stabilizer codes


Plaquette
$\prod_{\text{corners } i} Z_i$
Star
$\prod_{\text{corners } i} X_i$
Phaseflip Error
Bitflip Error
Z
Error String
Syndrome

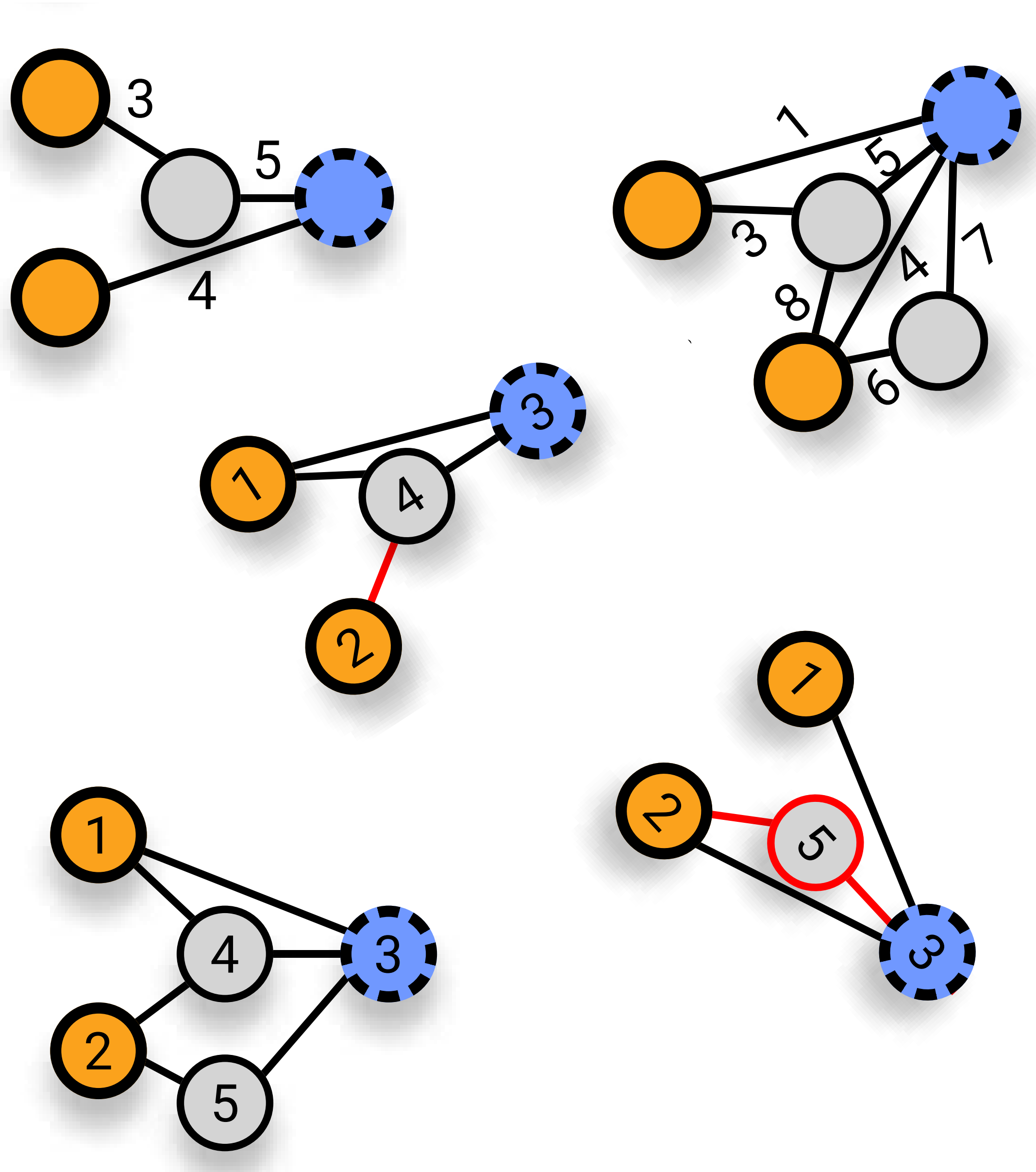
Quantum Computation

Reinforcement Learning



Deep Q-Network

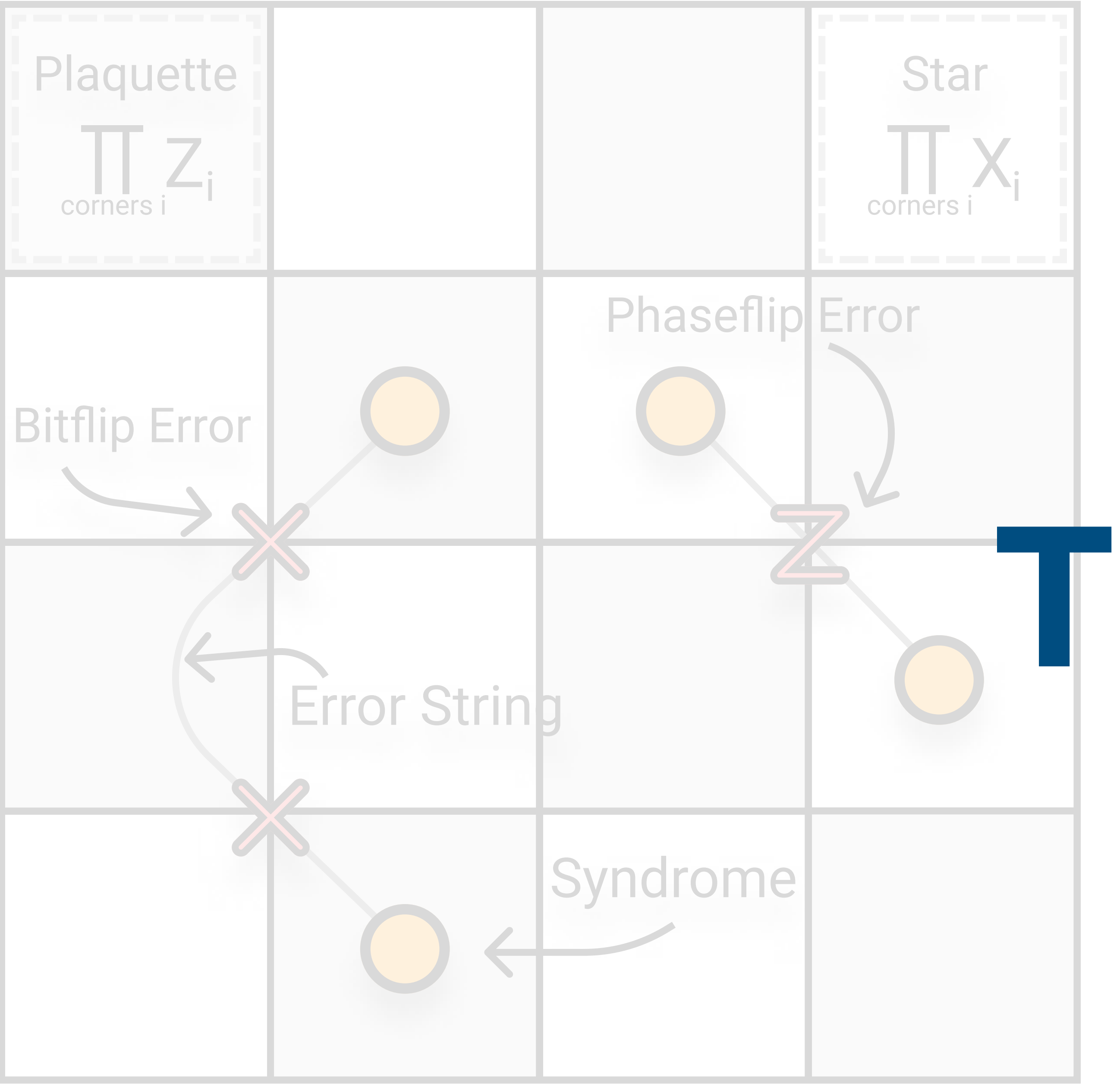
Evolutionary Strategy



Policy Networks

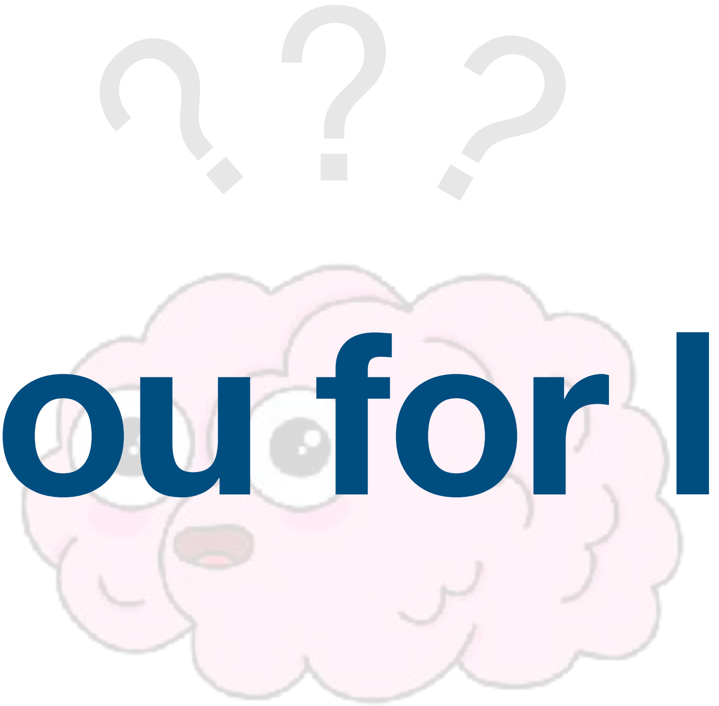
Thank you for listening!

Stabilizer codes



Quantum Computation

Reinforcement Learning



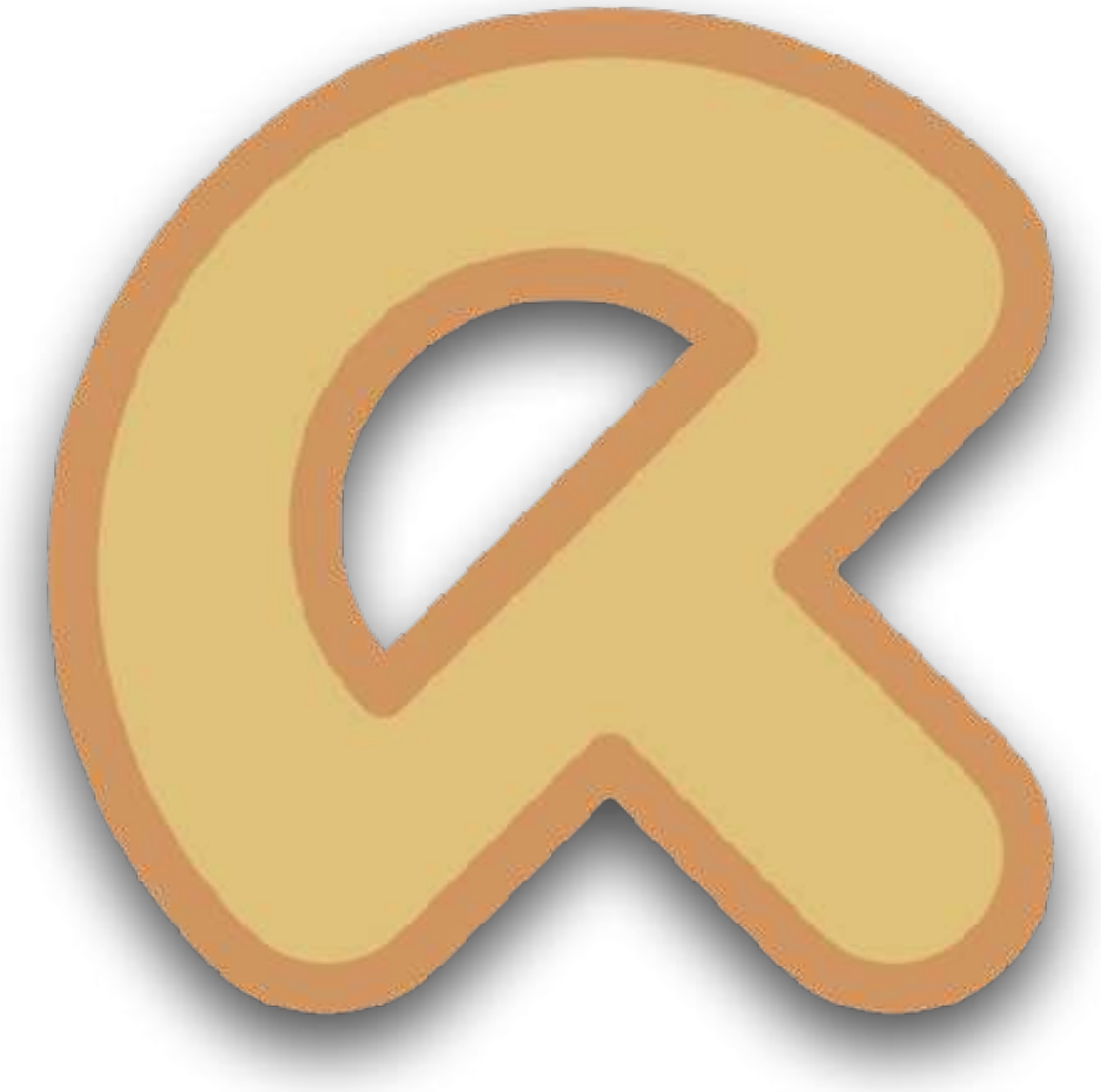
Deep Q-Network

Evolutionary Strategy

Policy Networks

Bonus content

Quantum Games!



Quantum TiqTaqToe
www.quantumtictactoe.com

Quantum Chess
www.quantumchess.net