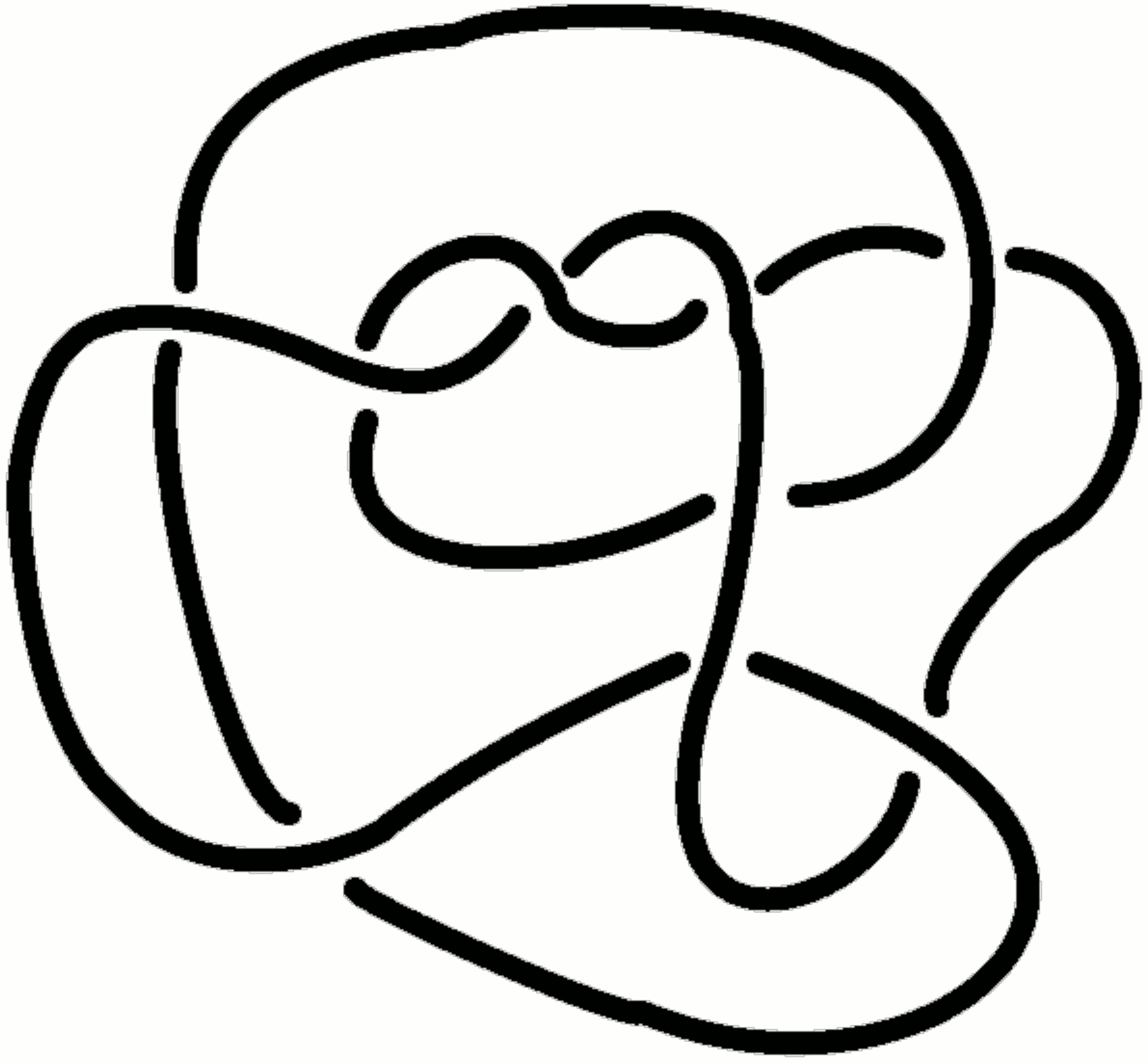
Lecture 2
Intro to RL

Fabian Ruehle (Northeastern University & IAIFI)

ML in Maths and Physics 2023
University of Oxford

17-21 July 2023

Motivation



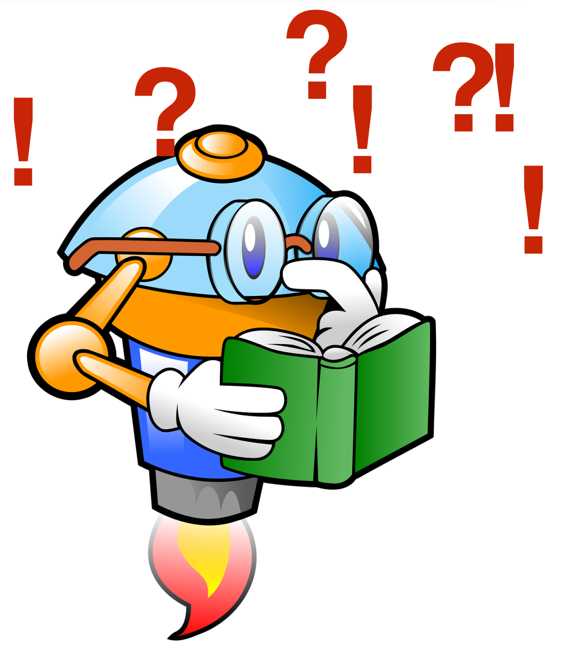
Question:

Can this knot be untangled?

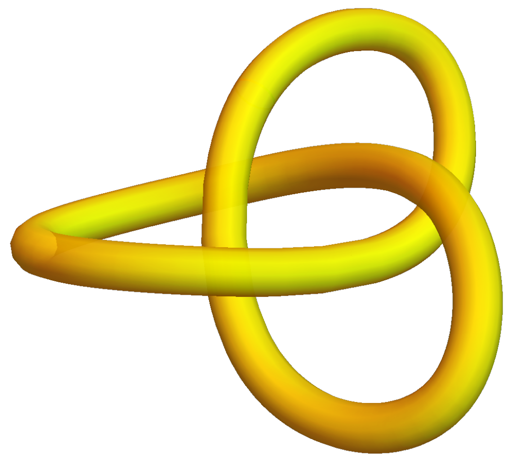
Outline

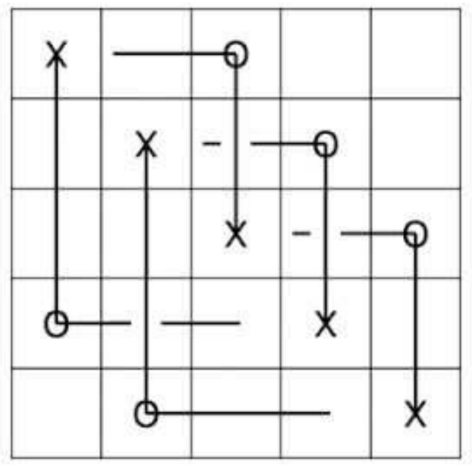
1 Why are some problems hard?

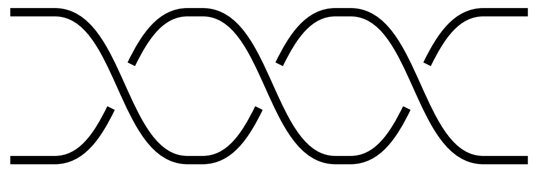
$$P \stackrel{?}{=} NP$$



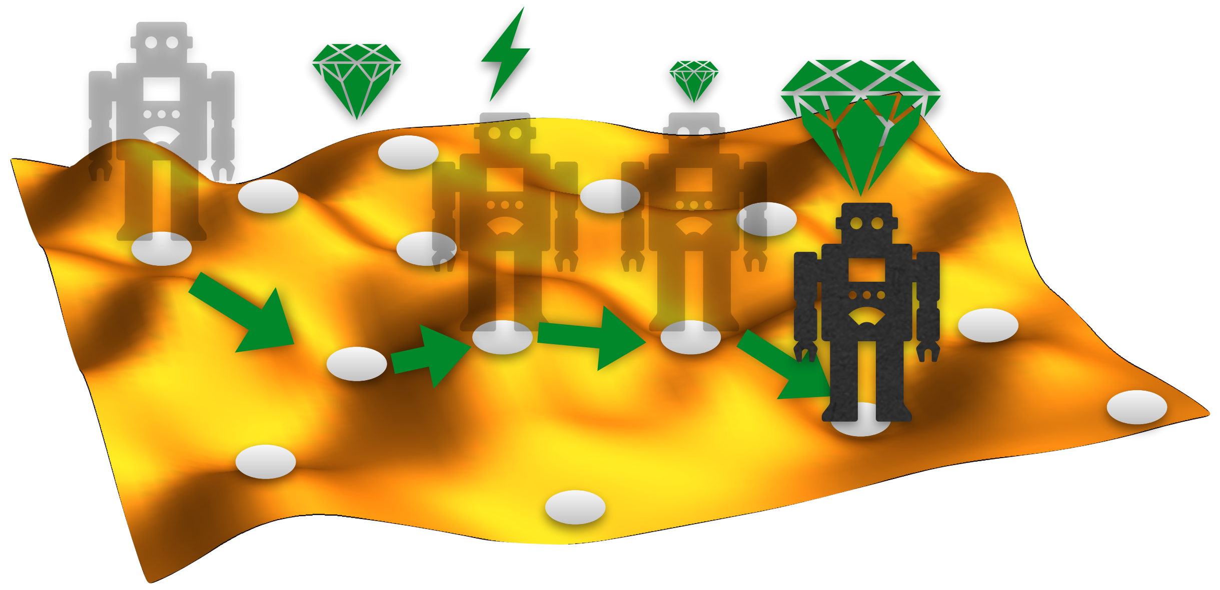
3 Intro to Knot Theory



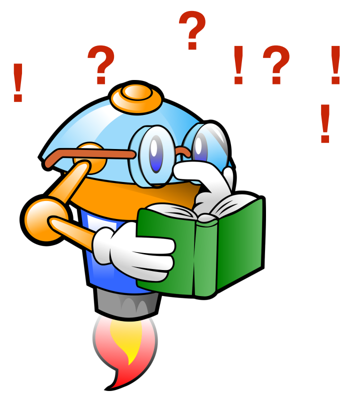


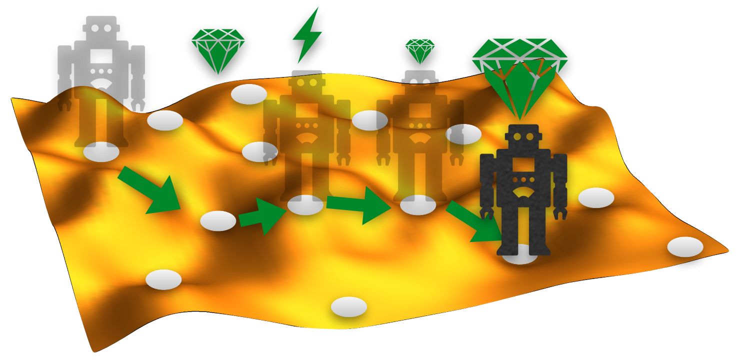


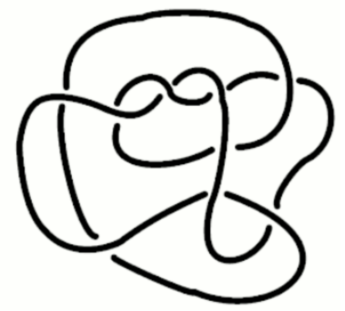
2 Intro to Reinforcement Learning

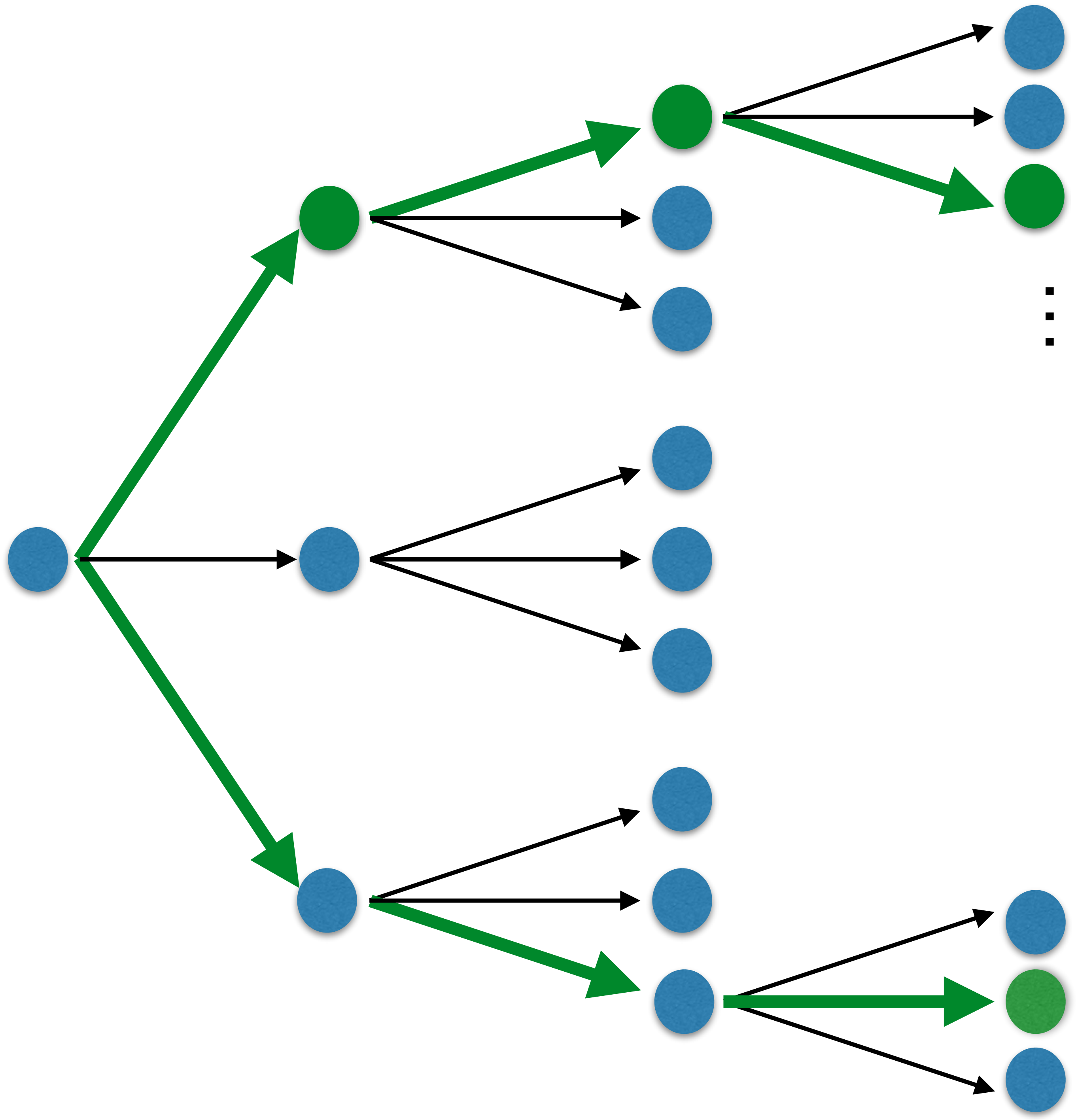


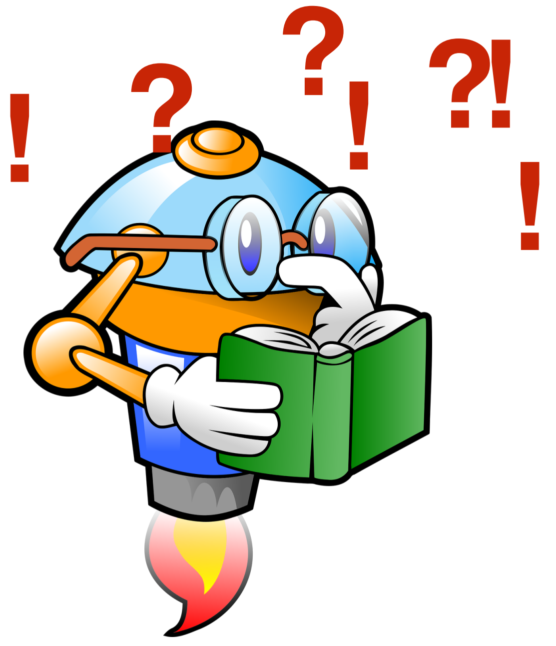
4 Recap











$$\boxed{P \stackrel{?}{=} NP}$$

Why are some problems hard?

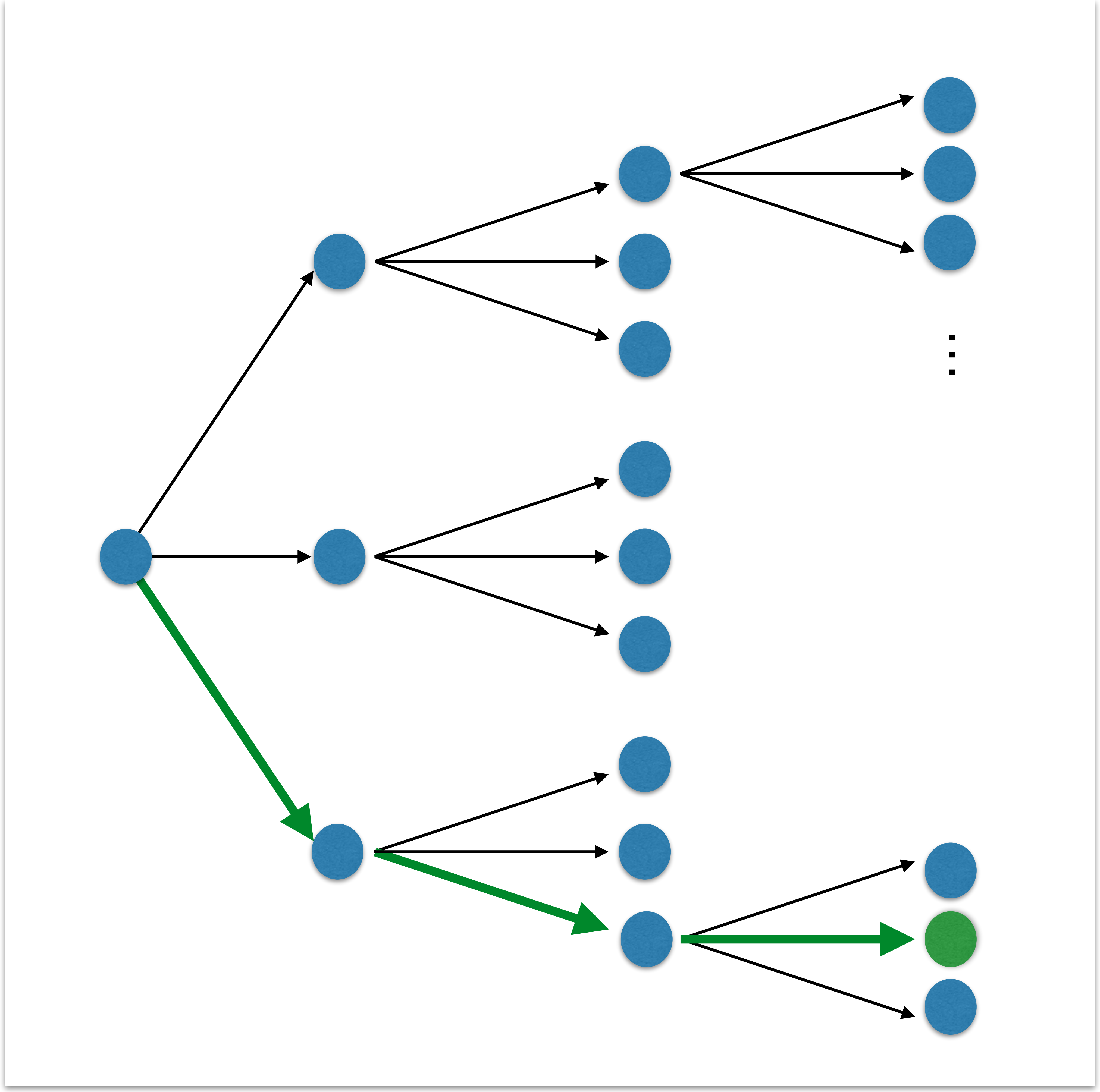
Computational Complexity

- Problems from discrete mathematics are ubiquitous:
    - In mathematics they appear in group theory, number theory, graph theory, topology,…
    - In physics they appear in any quantum theory
- These problems involve the following:
    - You are given some (mathematical) object
    - There is a number (need not be big) of manipulations you can do to the object
    - The correct sequence of manipulations will answer whether the problem has a solution in the affirmative
    - But there are exponentially many possibilities to combine the manipulations, so brute forcing the problem will take exponentially long

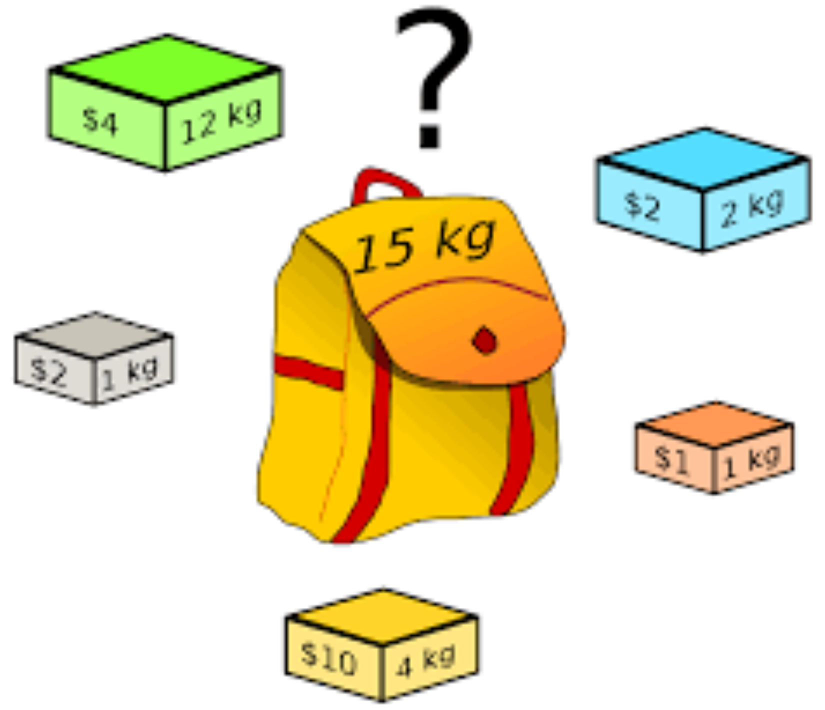
Computational Complexity - Examples

$$\boxed{35 = 5 \times 7}$$

Integer Factorization



Traveling Salesman



Knapsack Problem

Are there more efficient solution techniques?

- Problems that can be solved in polynomial time (=fast) are in the complexity class P
- Problems that can be verified in polynomial time are in the complexity class NP
- Question: Do there exist problems in NP that are not in P, i.e., $P \neq NP$?
  - Open (millennium prize) problem
  - Multi-trillion dollar bet against P=NP
  - Fact: There exist problems (like Knapsack, TSP) that are "as hard as it gets" within NP: If you can solve this problem in polynomial time, then you can solve **any** problem in NP in polynomial time by reducing it to this problem.

Other complexity classes

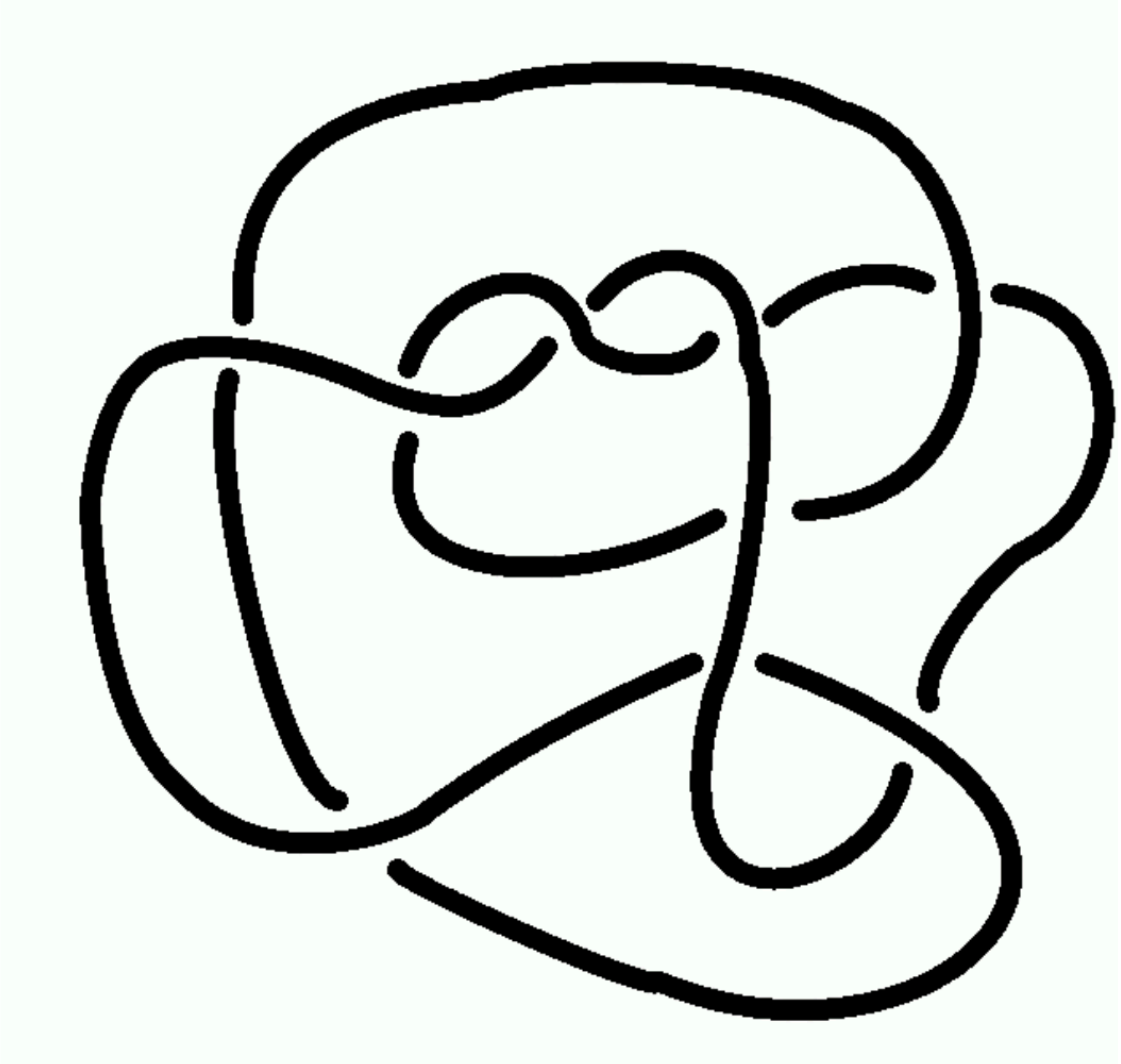
- There are problems that, given a potential solution, cannot even be verified in polynomial time
- This leads to a hierarchy of successively harder complexity classes:

$$\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE} \subseteq \mathrm{EXPSPACE} \subseteq \mathrm{DECIDABLE}$$

- But that is as bad as it gets …right?
- There exist undecidable problems, for example the halting problem: Write a computer program that, given some code as input, decides whether the input code halts eventually or runs forever…
- Also Diophantine equations are undecidable

$$a_4 x^4 + a_3 x^3 + a_2 x^2 + a_1 x + a_0 \stackrel{?}{=} 0$$

Binary Classification?



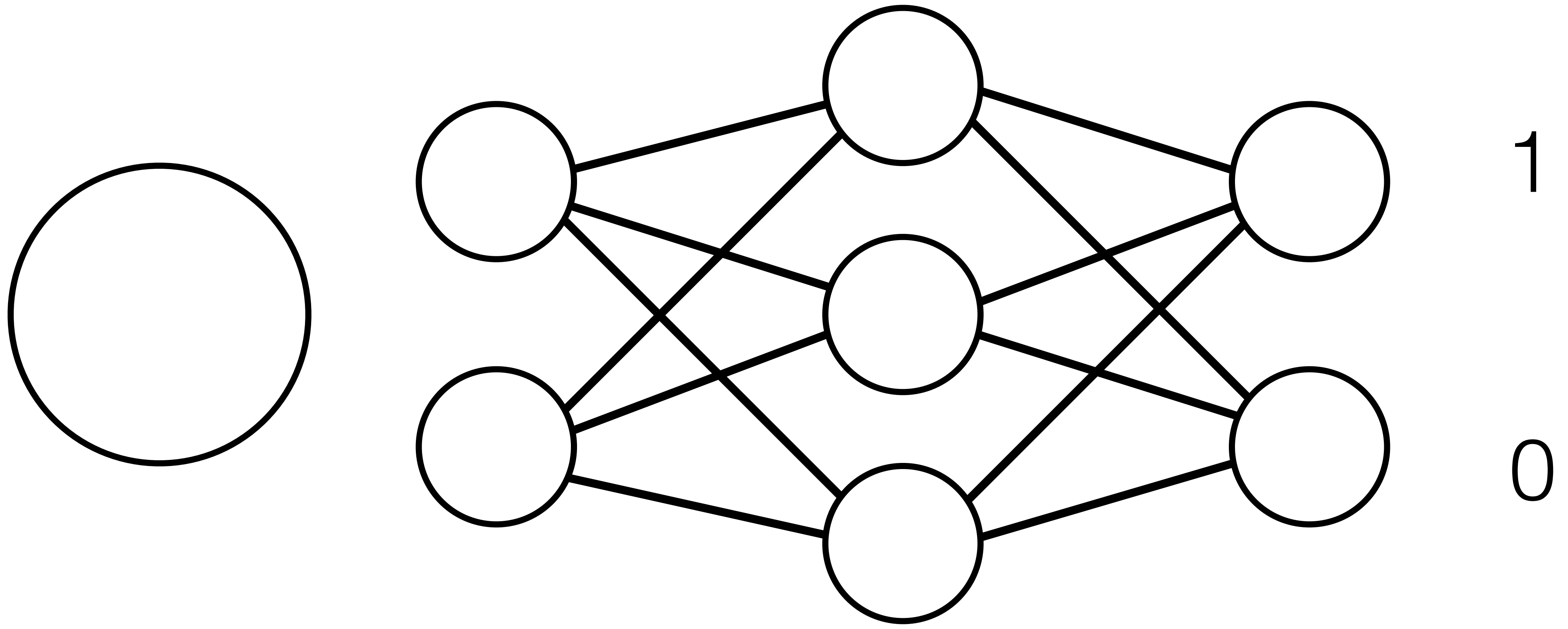
Question:

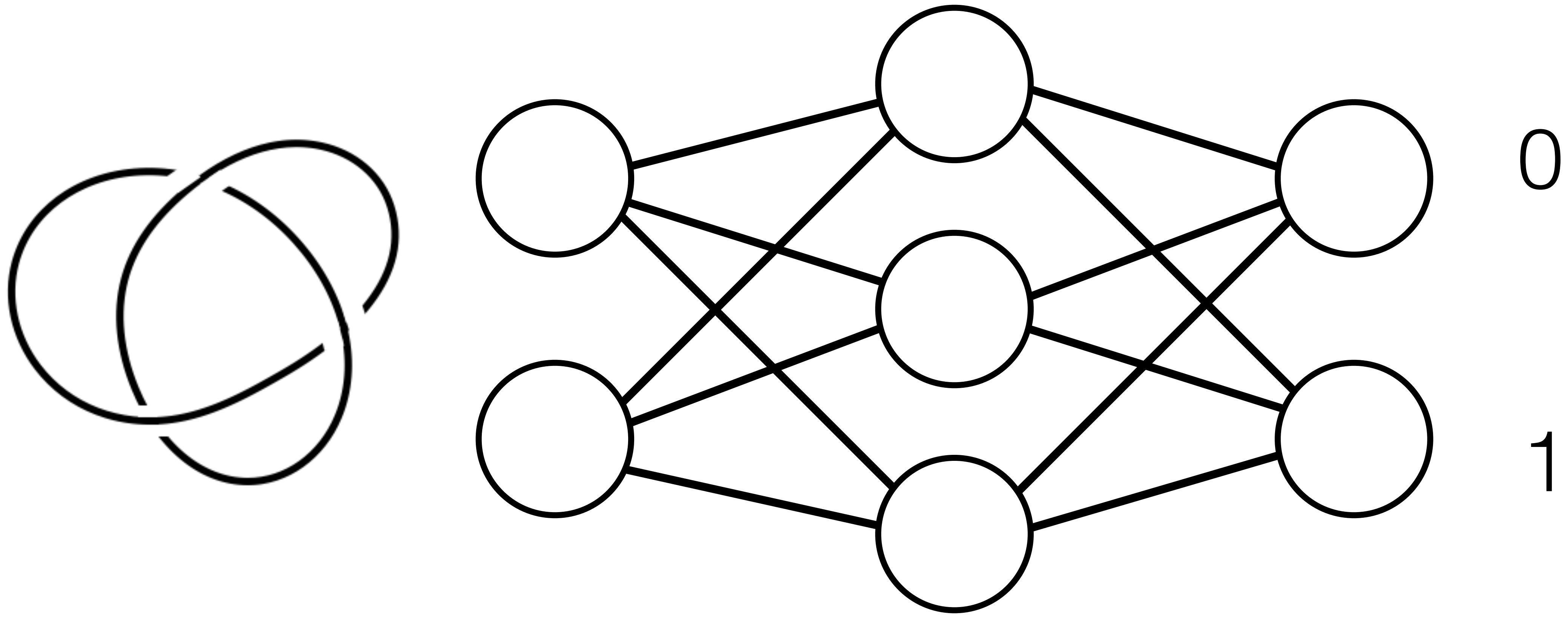
Can this knot be untangled?

Binary Classification

Can you define binary classification in 3 sentences

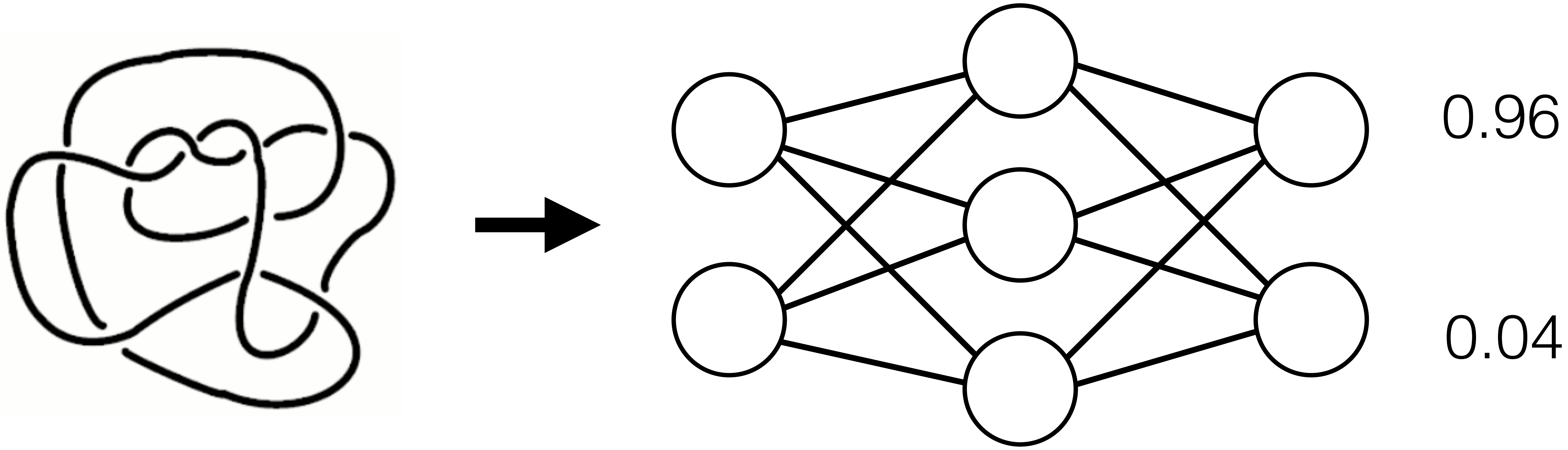
Binary classification is a type of supervised machine learning task where the goal is to classify input data into one of two mutually exclusive classes or categories. It involves training a model on labeled data to learn patterns and features that differentiate the two classes. The model then predicts the class label of new, unseen instances based on its learned knowledge.





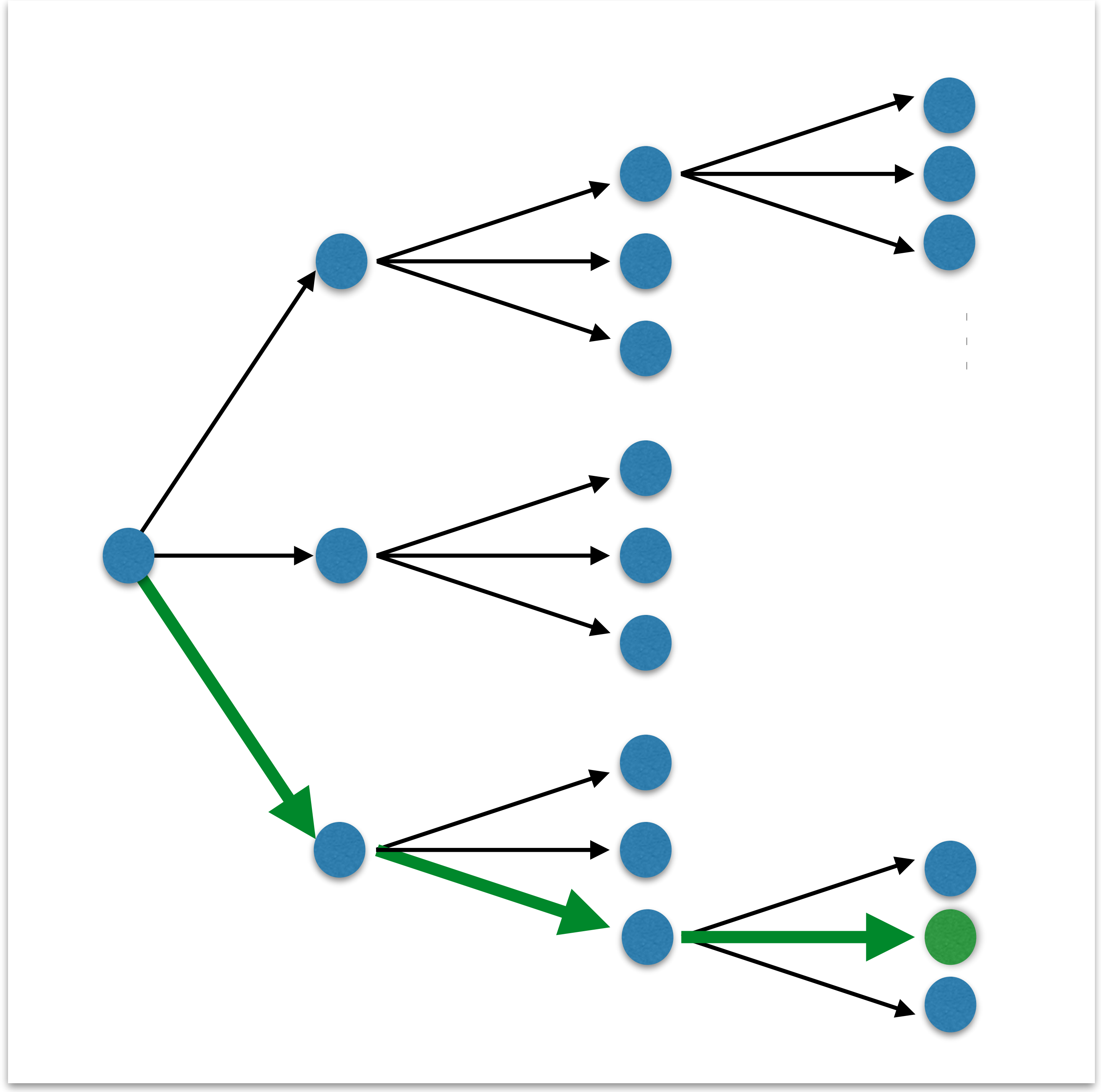
Binary Classification

… but in practice this happens:

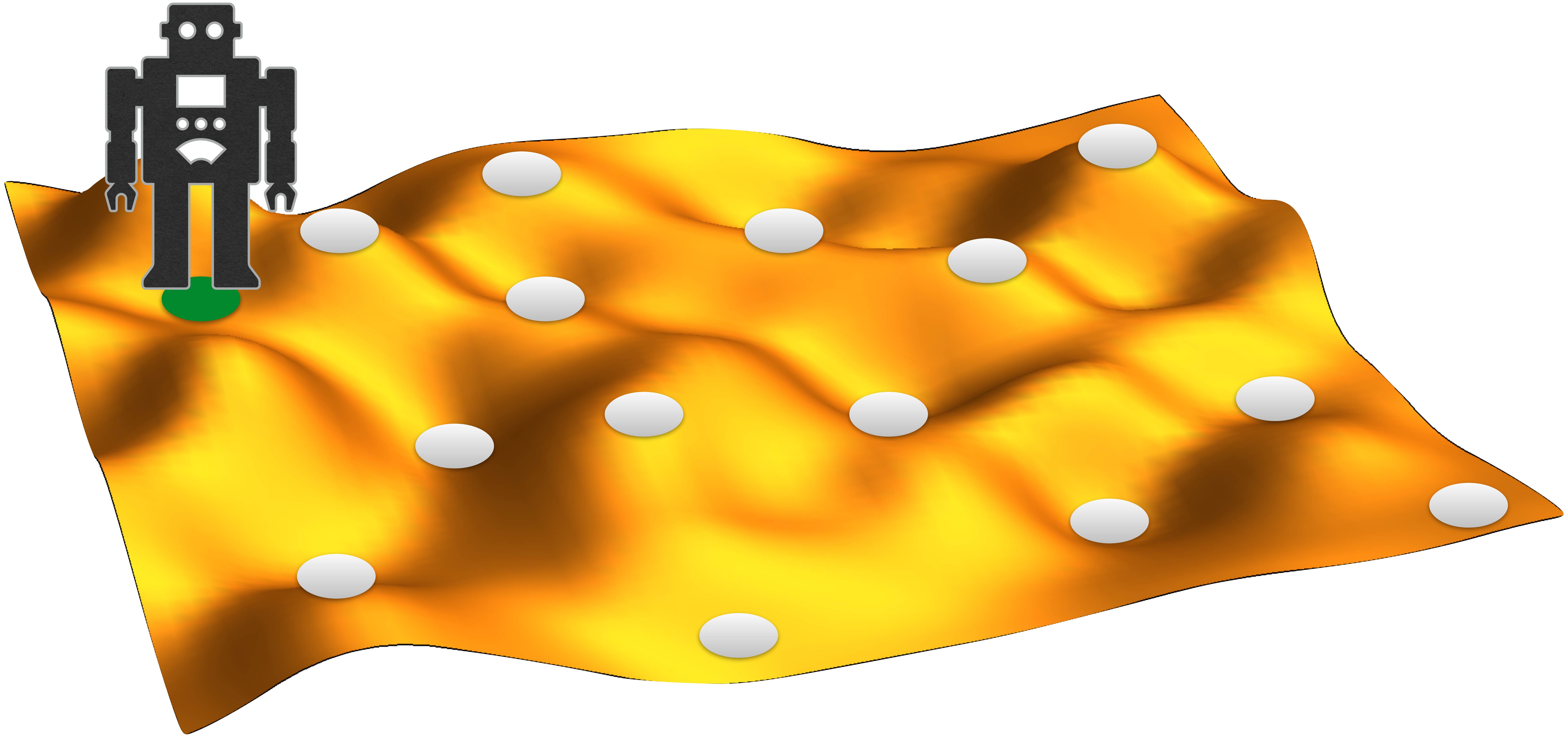


- So it is 94% unknot?
- Even if the answer was 100% unknot, and we wouldn't know how/why the NN says that)

…coming back to computational complexity



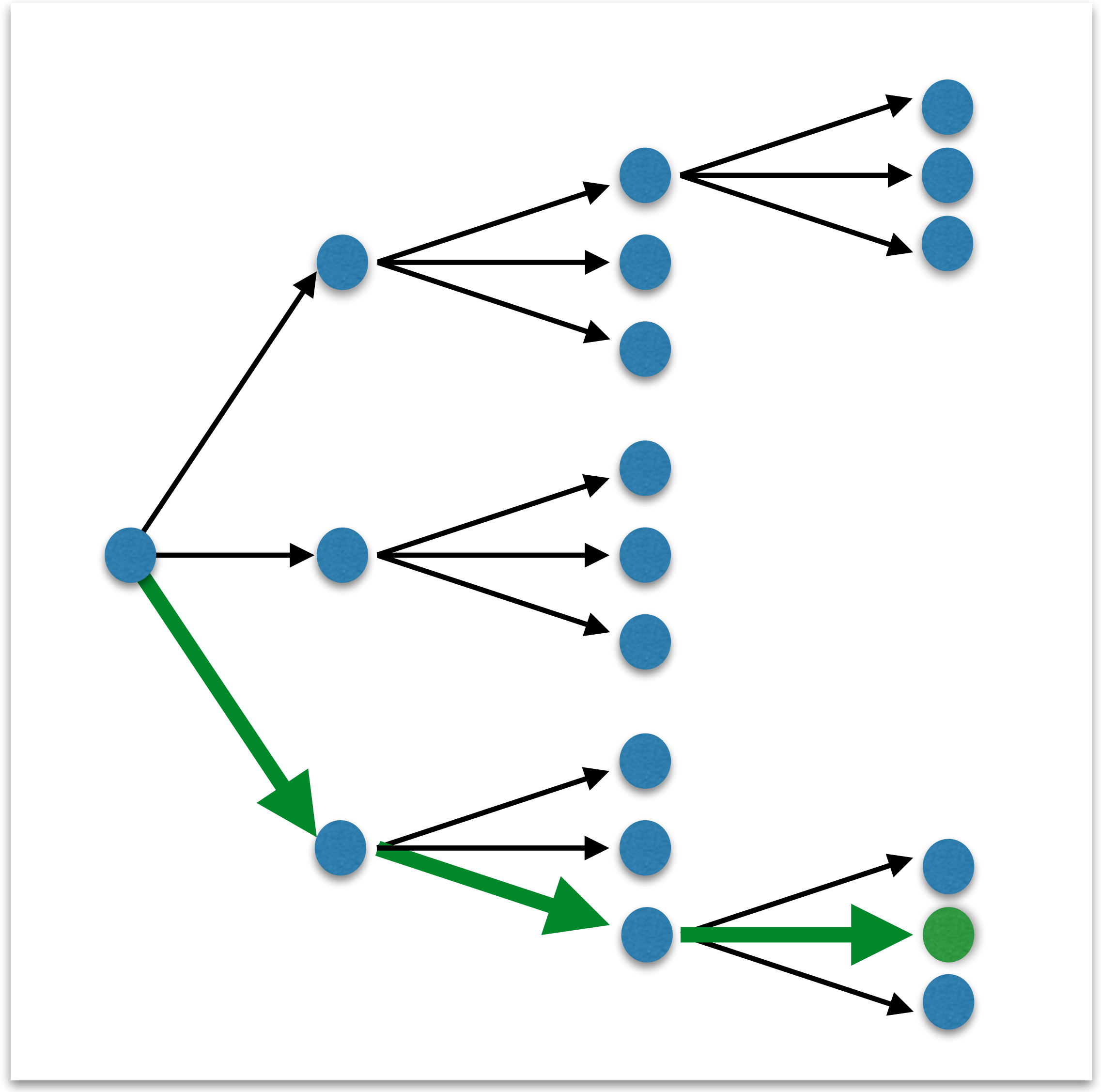
Go has $19 \times 19 = 361$ actions but $3^{361} = 10^{170}$ different states. Impossible to brute-force.
Also, asking "can you win from this position" would be useless



Intro to Reinforcement Learning

Reinforcement Learning - Intro

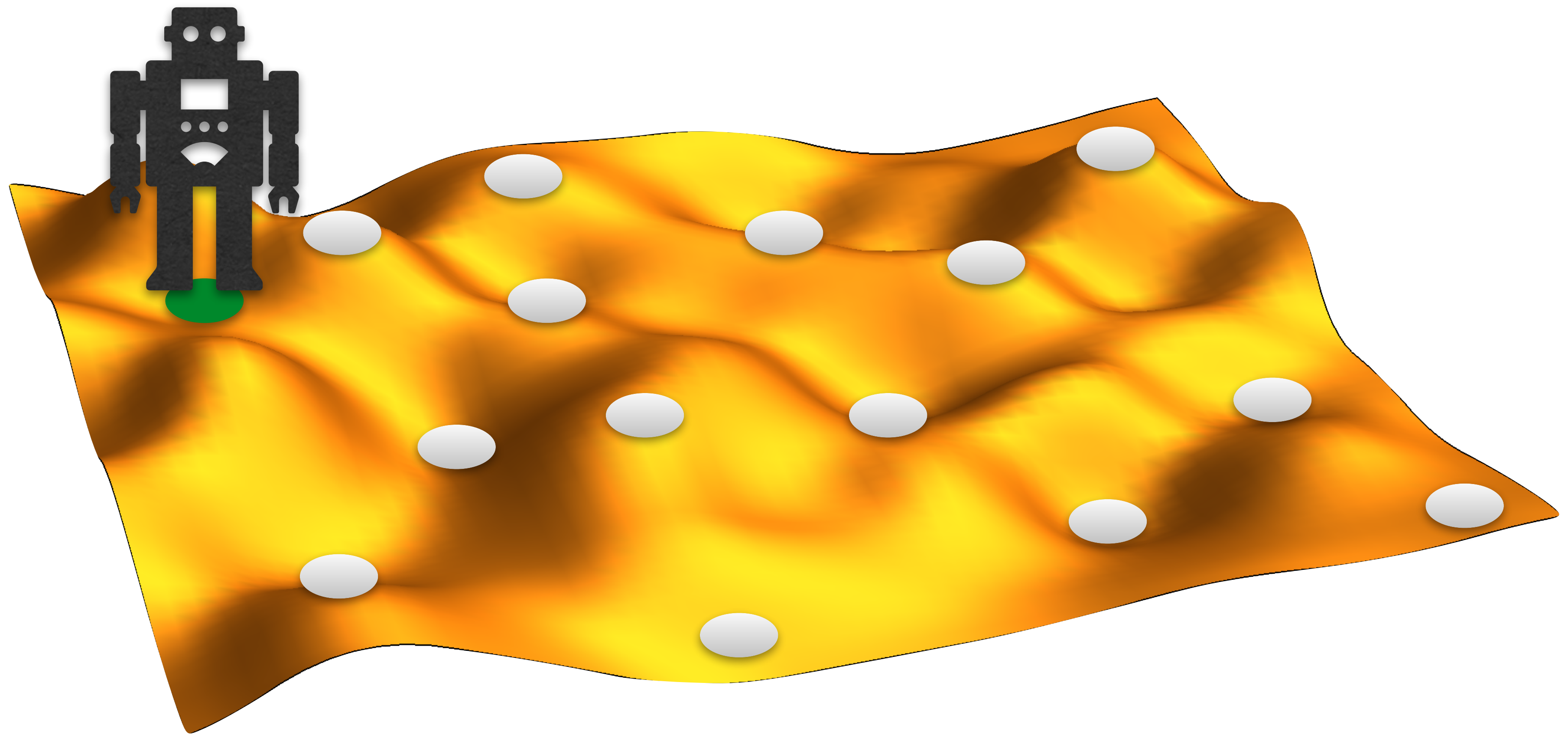
- RL is a way to deal with combinatorially large search spaces
- Instead of trying all combinations by brute force (which would require visiting all states), you train a *policy NN* that learns some policy according to which it decides for you which state to visit next
- To do so, RL solves a Markov Decision Problem, meaning once a certain state is reached, the next action is independent of how it was reached



Reinforcement Learning

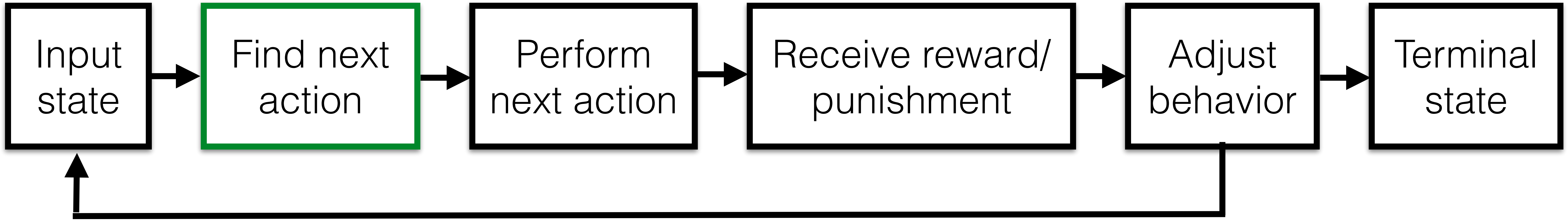
- Task: Find sequence of actions that optimize a process

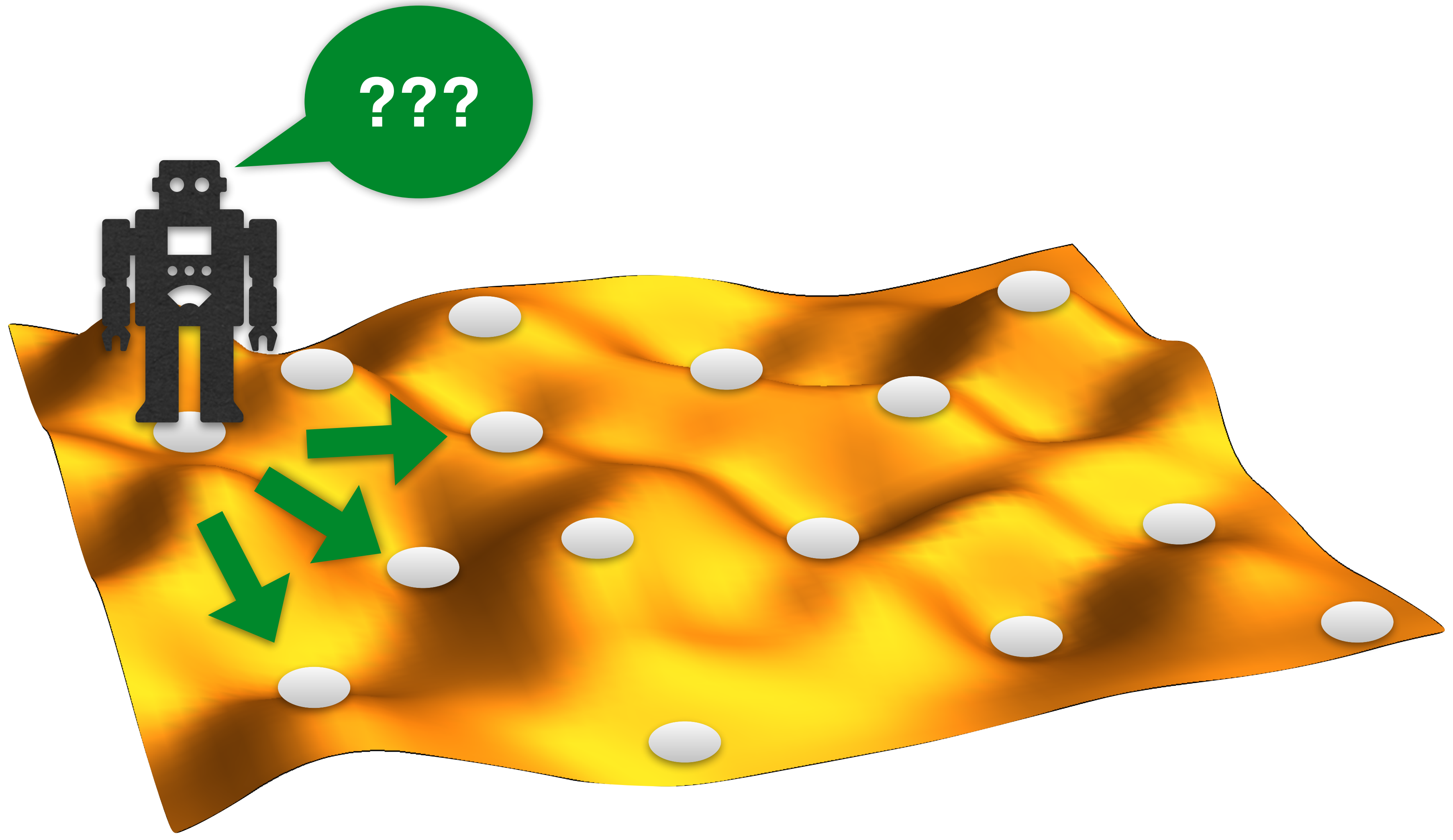
Input state → Find next action → Perform next action → Receive reward/ punishment → Adjust behavior → Terminal state



Reinforcement Learning

- Task: Find sequence of actions that optimize a process

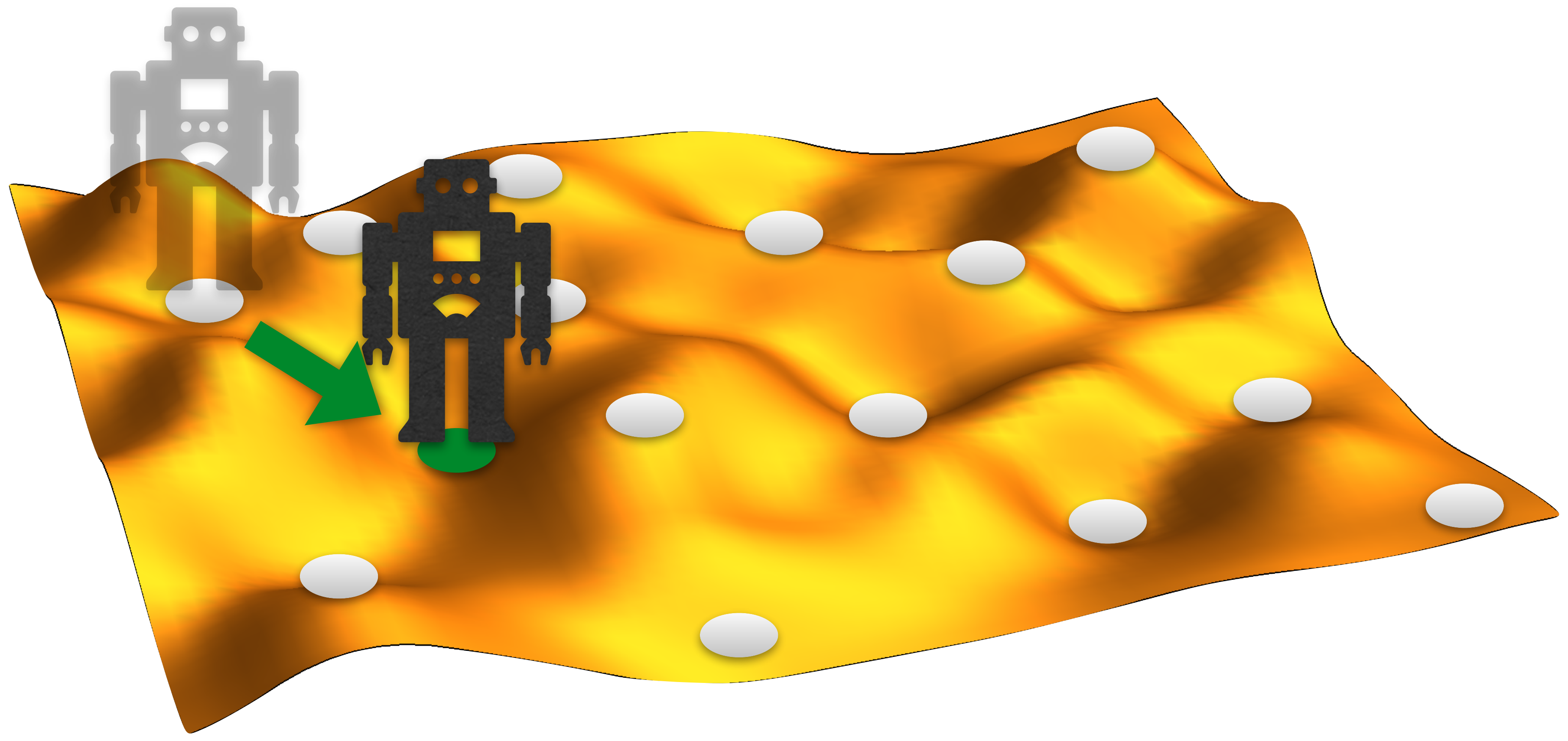




Reinforcement Learning

- Task: Find sequence of actions that optimize a process

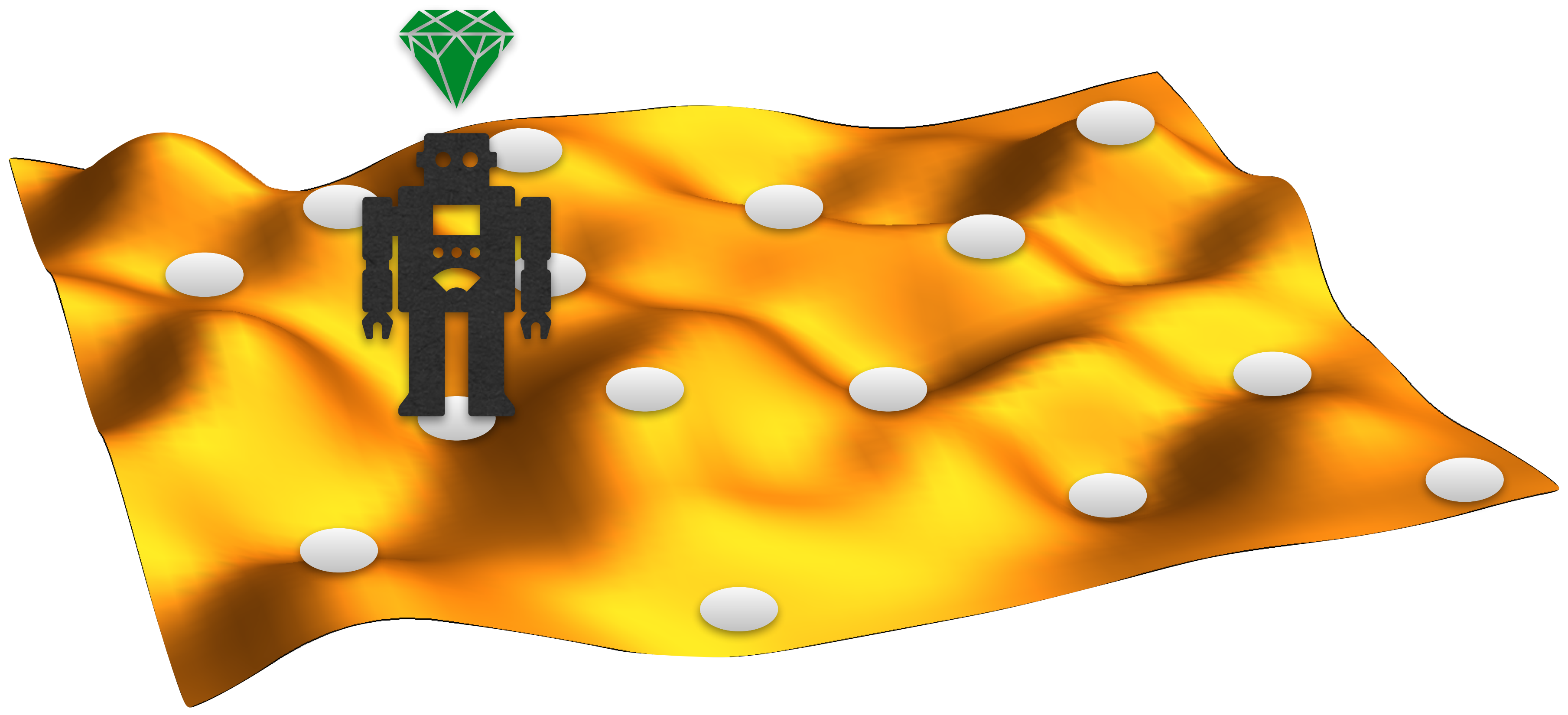
Input state → Find next action → Perform next action → Receive reward/ punishment → Adjust behavior → Terminal state



Reinforcement Learning

- Task: Find sequence of actions that optimize a process

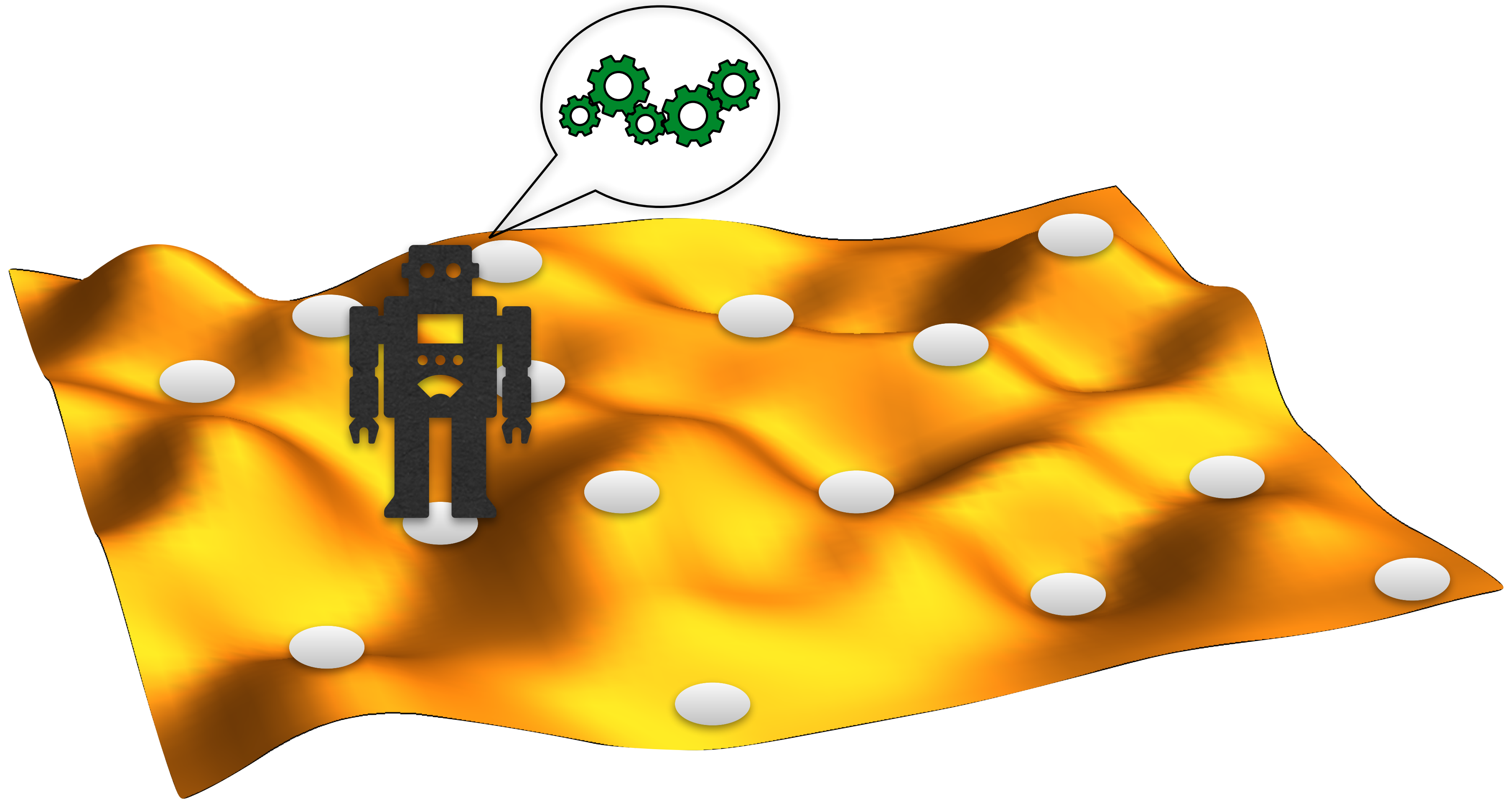
Input state → Find next action → Perform next action → Receive reward/ punishment → Adjust behavior → Terminal state



Reinforcement Learning

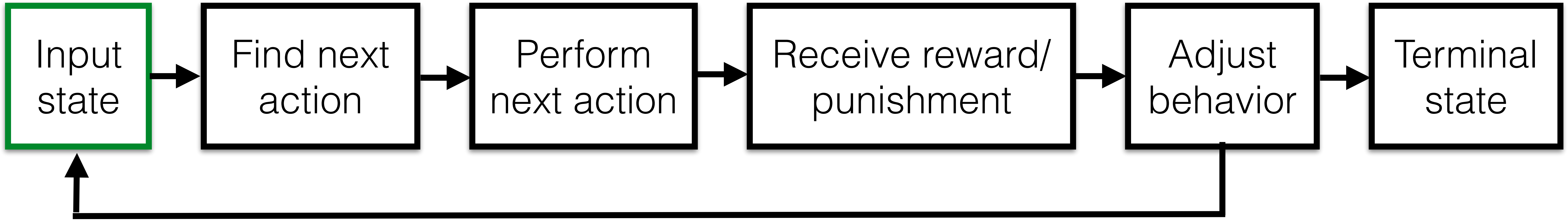
- Task: Find sequence of actions that optimize a process

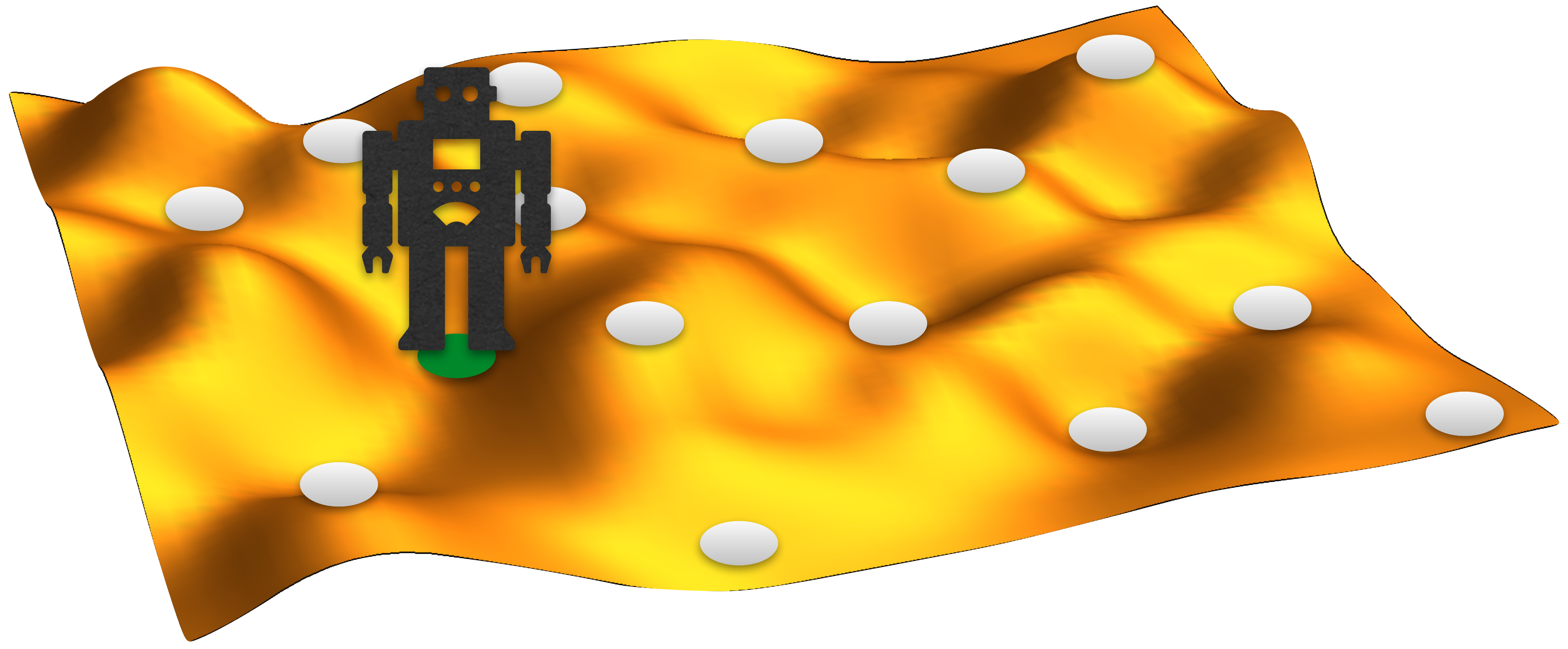
Input state → Find next action → Perform next action → Receive reward/ punishment → Adjust behavior → Terminal state



Reinforcement Learning

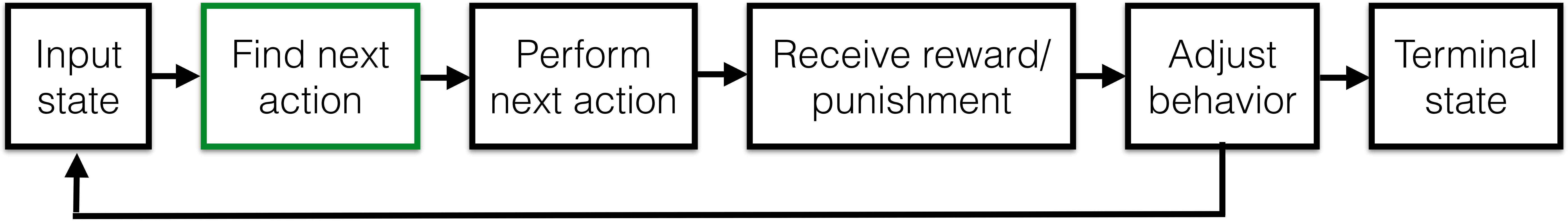
- Task: Find sequence of actions that optimize a process

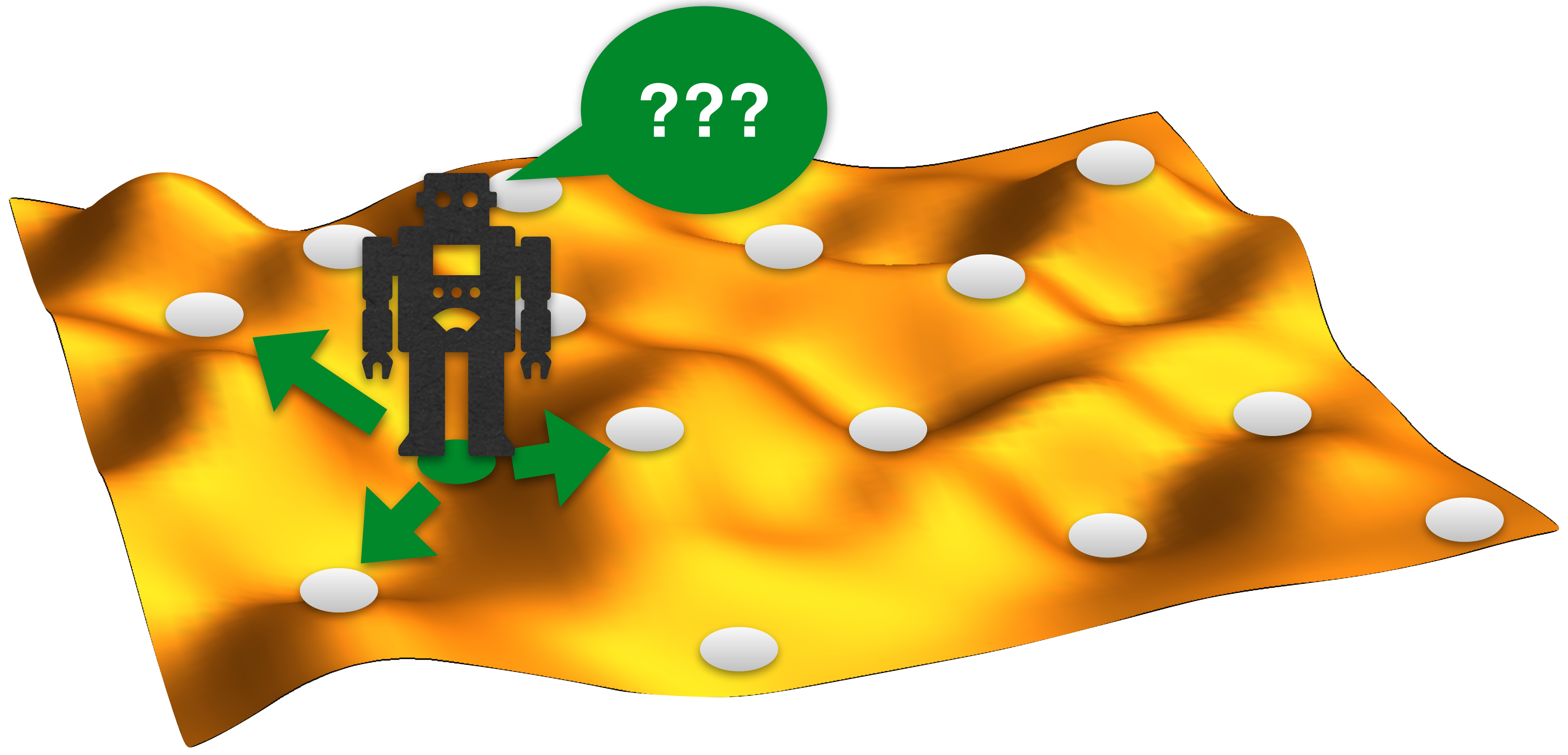




Reinforcement Learning

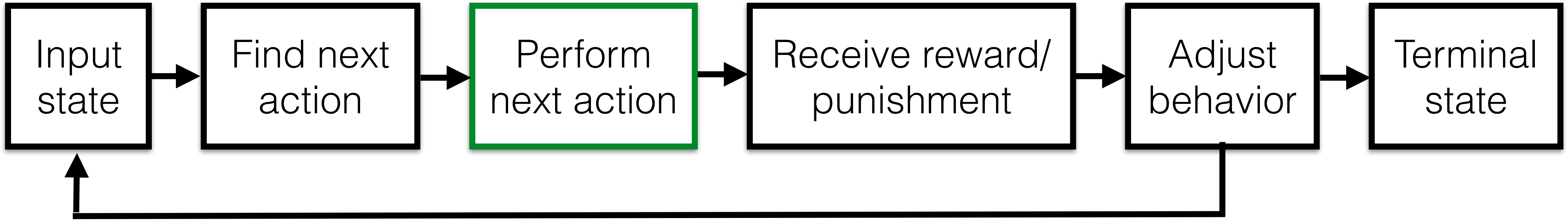
- Task: Find sequence of actions that optimize a process

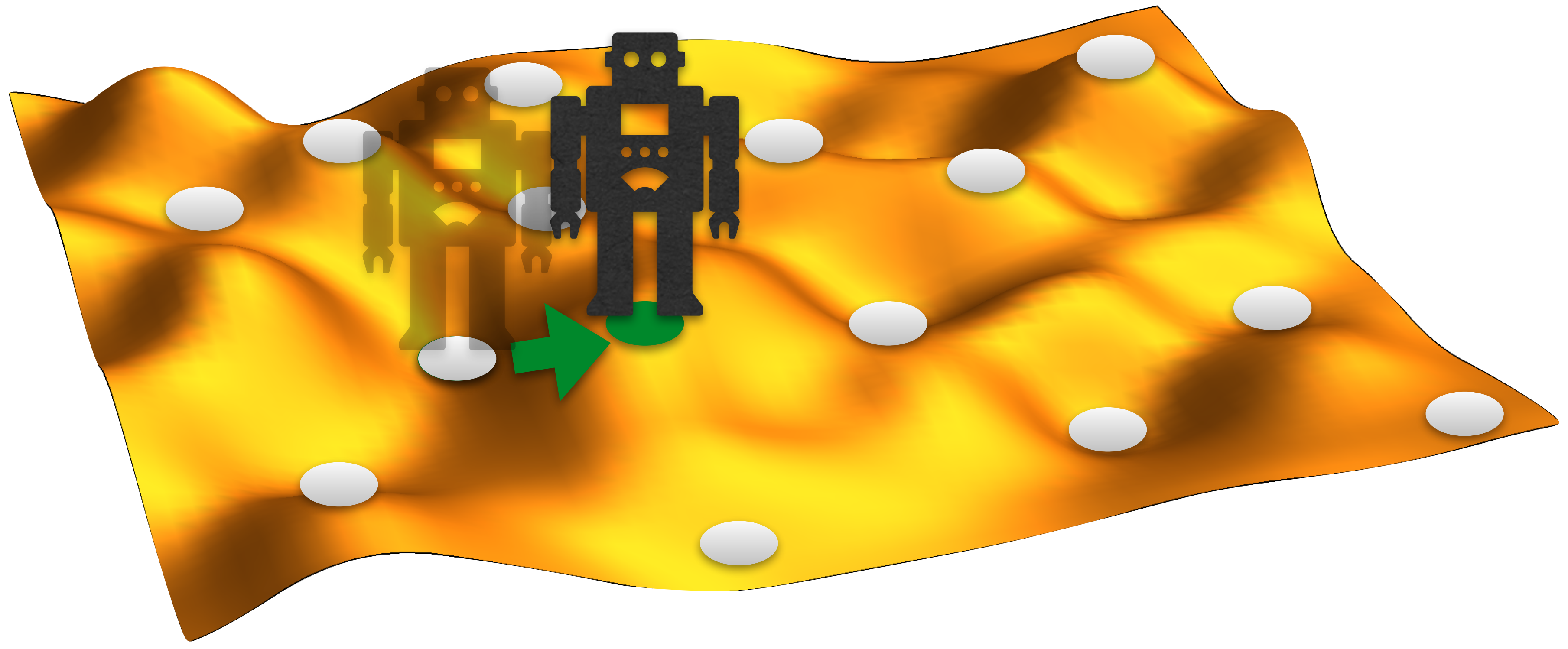




Reinforcement Learning

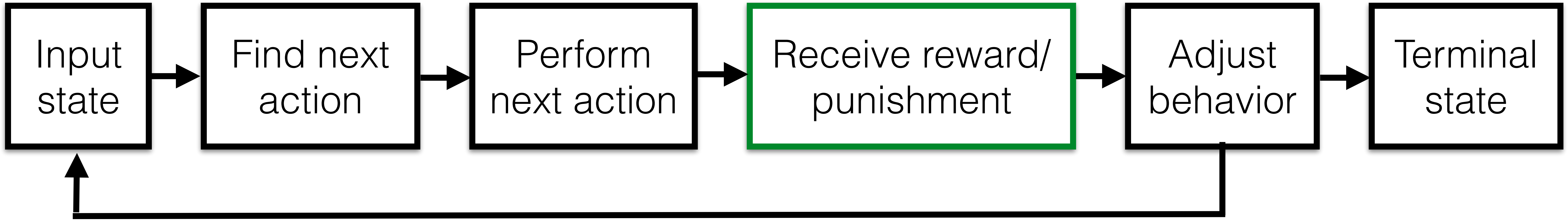
- Task: Find sequence of actions that optimize a process

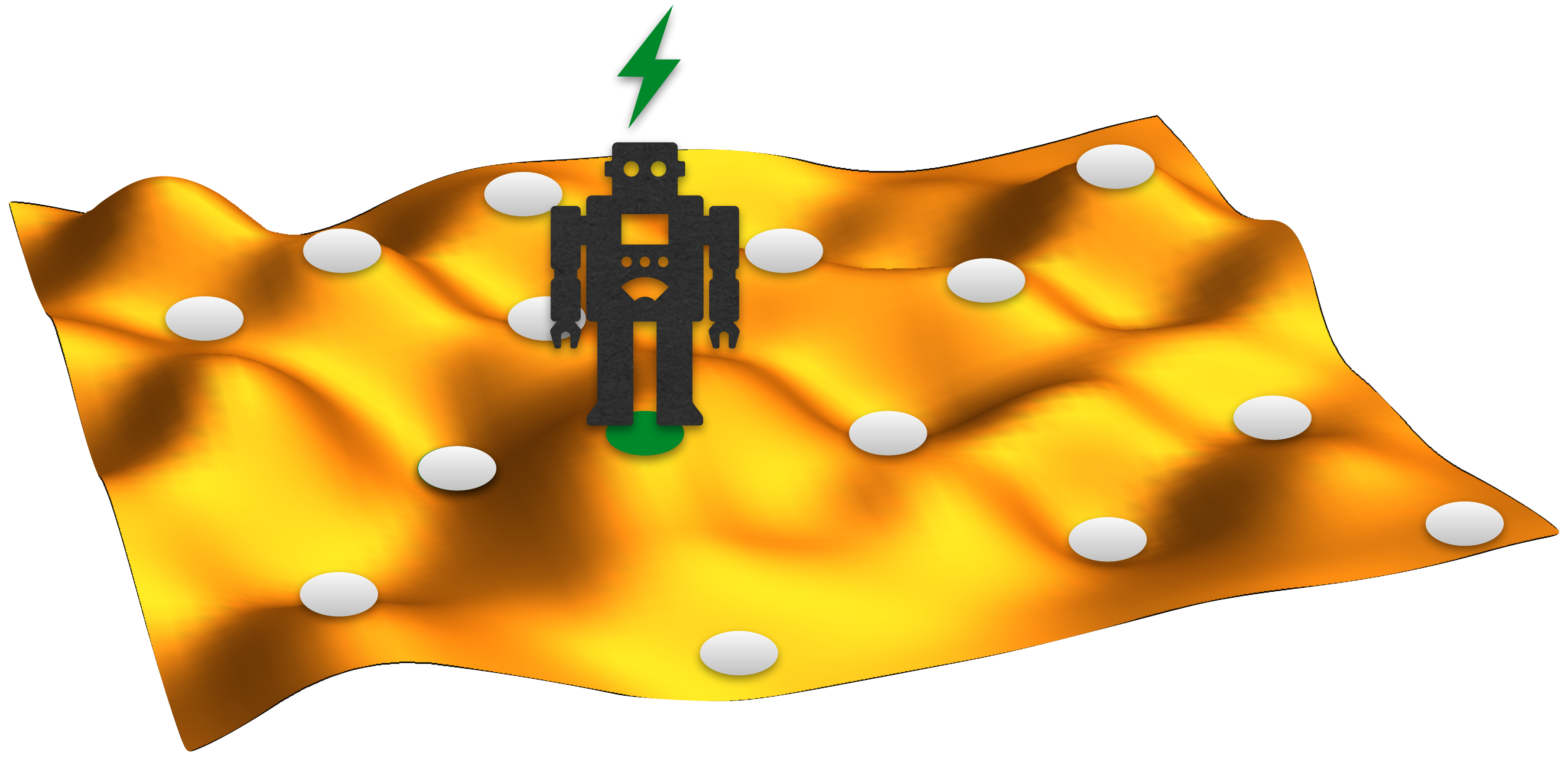




Reinforcement Learning

- Task: Find sequence of actions that optimize a process

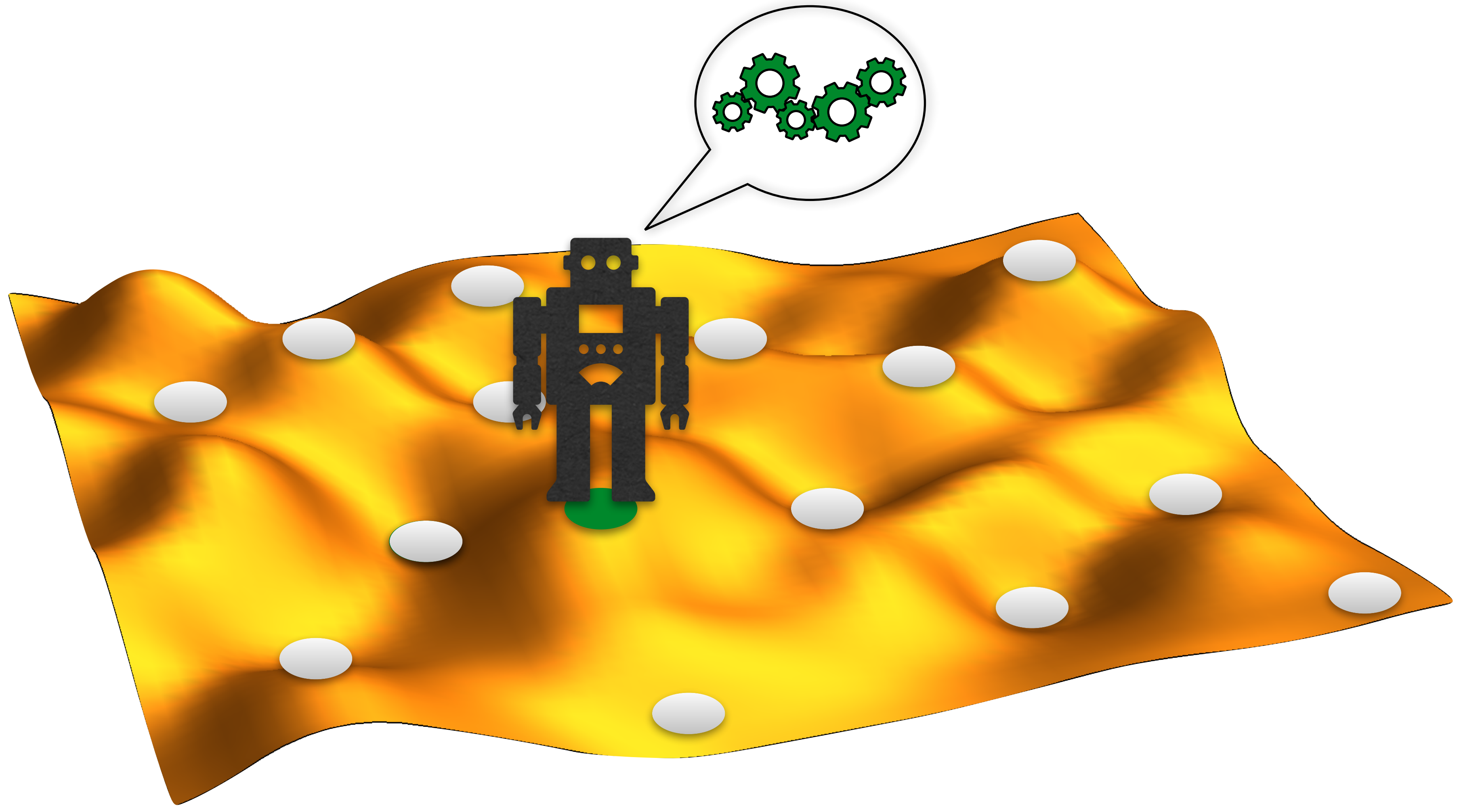




Reinforcement Learning

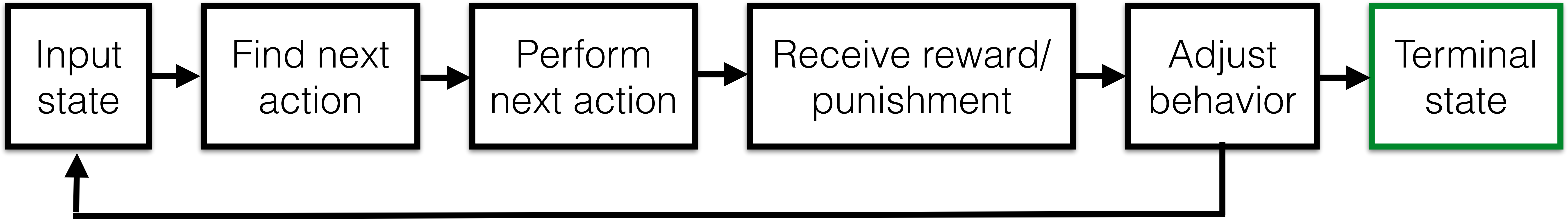
- Task: Find sequence of actions that optimize a process

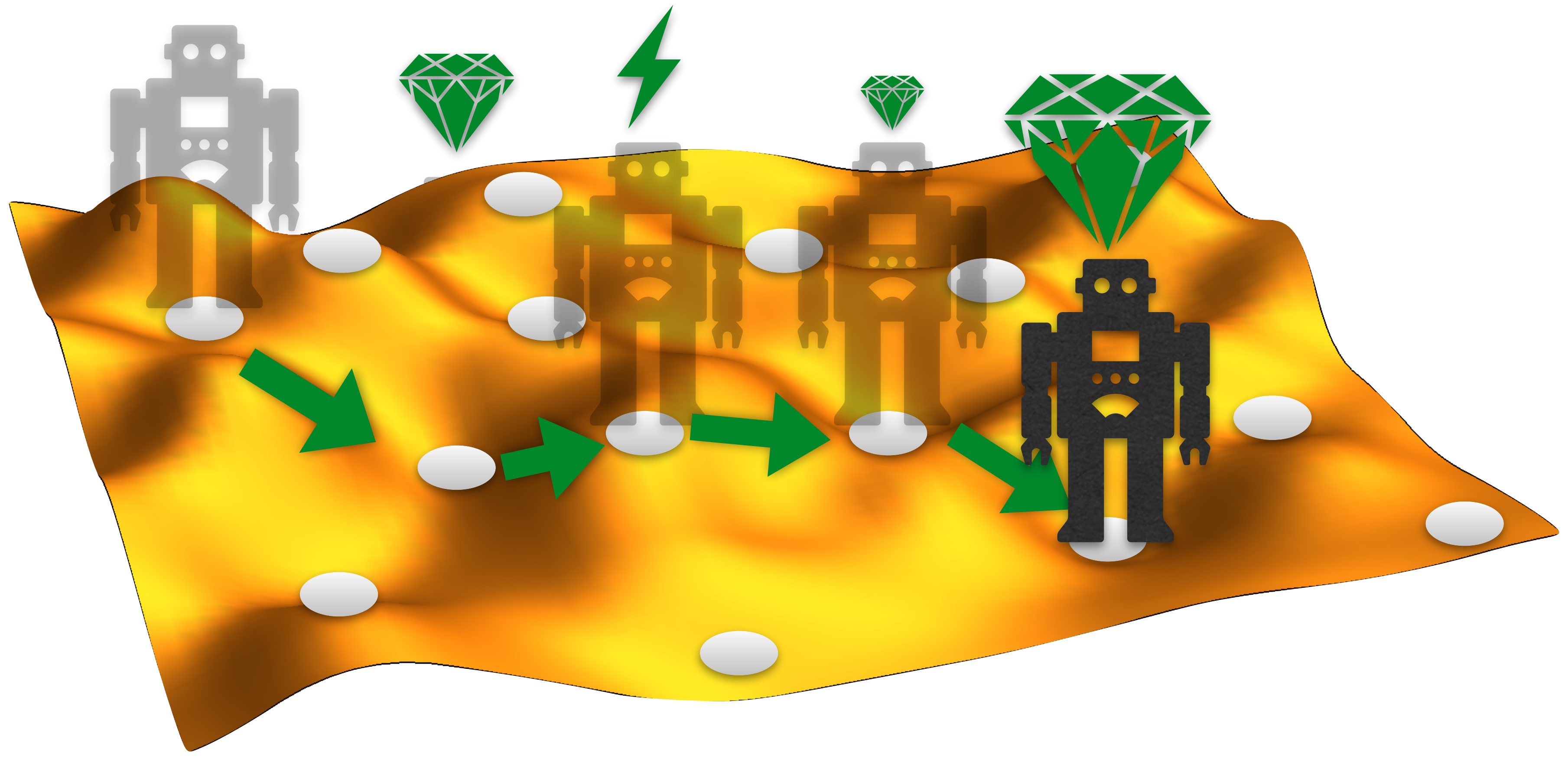
Input state → Find next action → Perform next action → Receive reward/ punishment → Adjust behavior → Terminal state



Reinforcement Learning

- Task: Find sequence of actions that optimize a process





Reinforcement Learning - Vocab

- Environment: Set of states and actions on these states $\mathcal{E} = \{\mathcal{S}, \mathcal{A}\}$
- States: Set of possible "configurations", $\mathcal{S} = \{s\}$
  Can be discrete (finite or infinite), or continuous
- Actions: Set of actions that transition between states. Can be discrete (finite or infinite), or continuous. $\mathcal{A} = \{a \mid a : \mathcal{S} \to \mathcal{S}\}$
  In my experience, RL shines for huge state spaces but modestly large action spaces
- Terminal states: Subset of states for which no action is possible (the search has ended) $\mathcal{S} \supset \mathcal{T} = \{t \in \mathcal{S} \mid a(t) = \emptyset\}$
- Episode: A sequence of states and actions that ends in a terminal state

$$E = [(s_1, a_1), (s_2, a_2), \ldots, (s_n, \emptyset)], \quad a_i \in \mathcal{A}, \; s_i \in \mathcal{S} \backslash \mathcal{T}, \; s_n \in \mathcal{T}$$

Reinforcement Learning - Vocab

- Policy: Describes how the agent selects an action given its current state

  $$\pi : \ \mathcal{S} \rightarrow \mathcal{A}$$

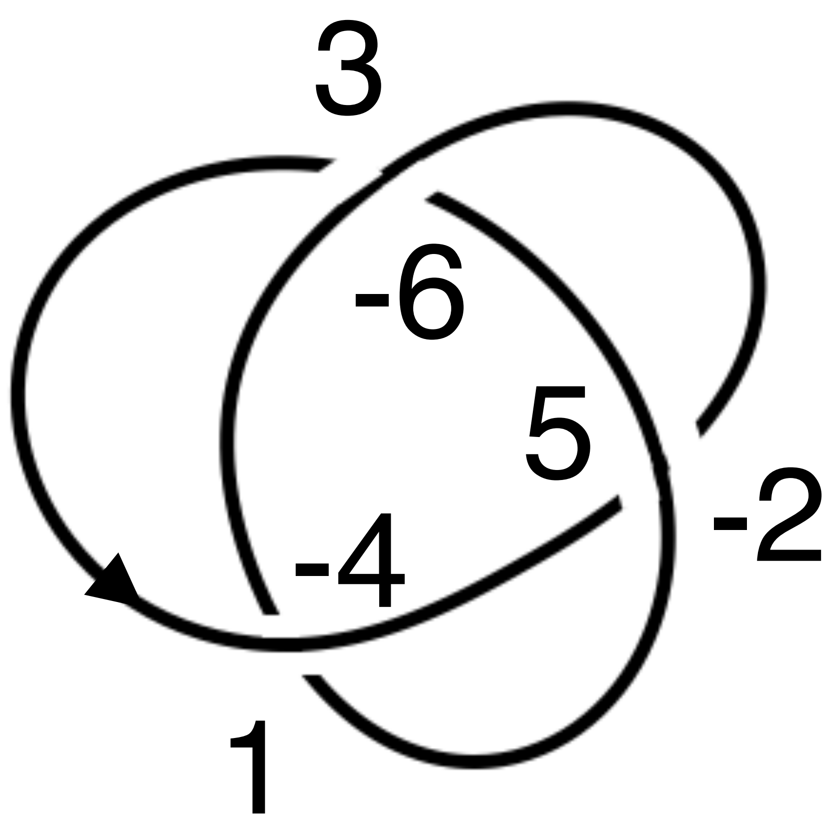
  - Deterministic policy: a is determined uniquely from s
  - Non-deterministic policy: multiple s are possible for the same a

- Reward: The feedback given to the agent for following policy π .
  Usually the reward $r \in \mathbb{R}\,, \ \ r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$

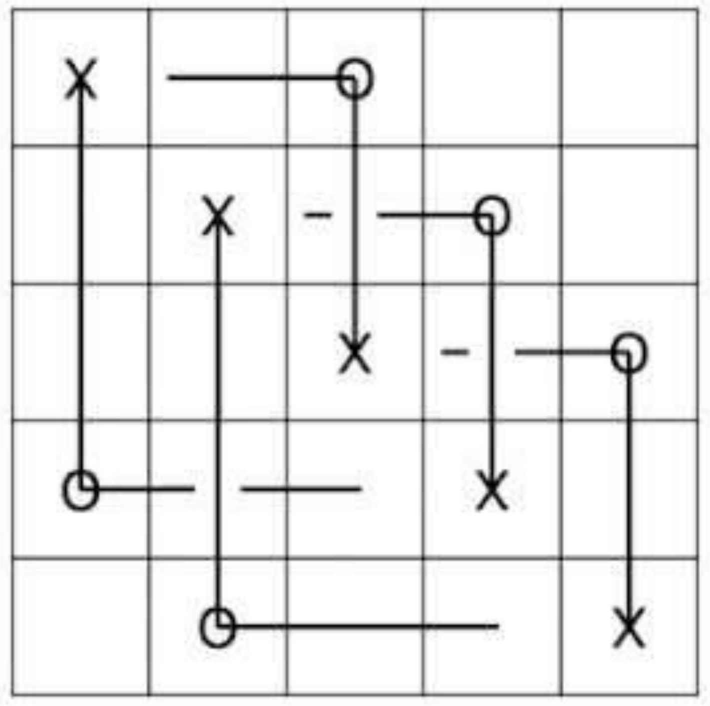
- Return: accumulated reward from current position onward: $G = \sum_{t=0}^{\infty} \gamma^t r_t$
  $0 < \gamma \leq 1$ is the so-called discount factor.
  Note that G depends on π

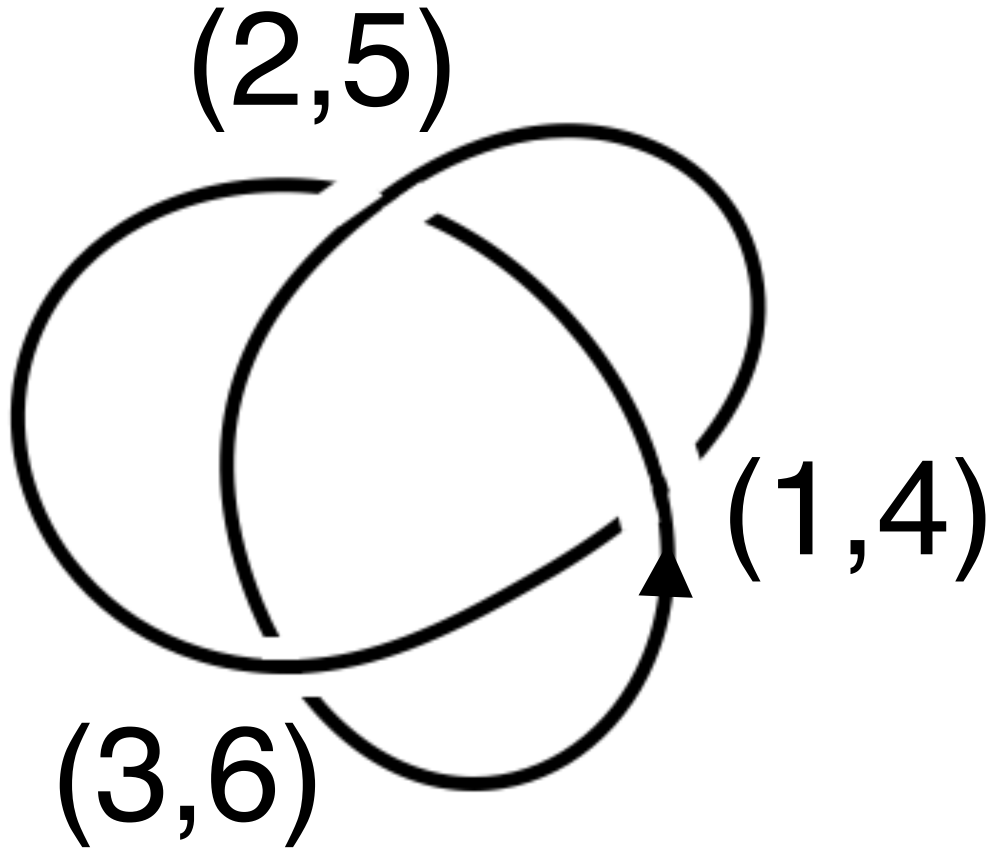
Reinforcement Learning - Vocab

- State value function: Expected return from current state onward when following policy π : $v(s) = \mathbb{E}(G_t \mid s = s_t)$
- Action value function: Expected return for choosing action a_t in state s_t
  $q(s, a) = \mathbb{E}(G_t \mid s = s_t,\ a = a_t)$
- We need to find an estimator for the best state value function, action value function, and for the policy itself. These are inter-dependent and in RL (at least some) are approximated by a NN:
  - policy: Show a state to a NN and have it predict which action to take next
  - state value function: Show the state to a NN and have it predict how good the state is, i.e., how much return to expect from being in this state
  - action value function: Show the state and action to a NN and have it predict how good the action is in this state

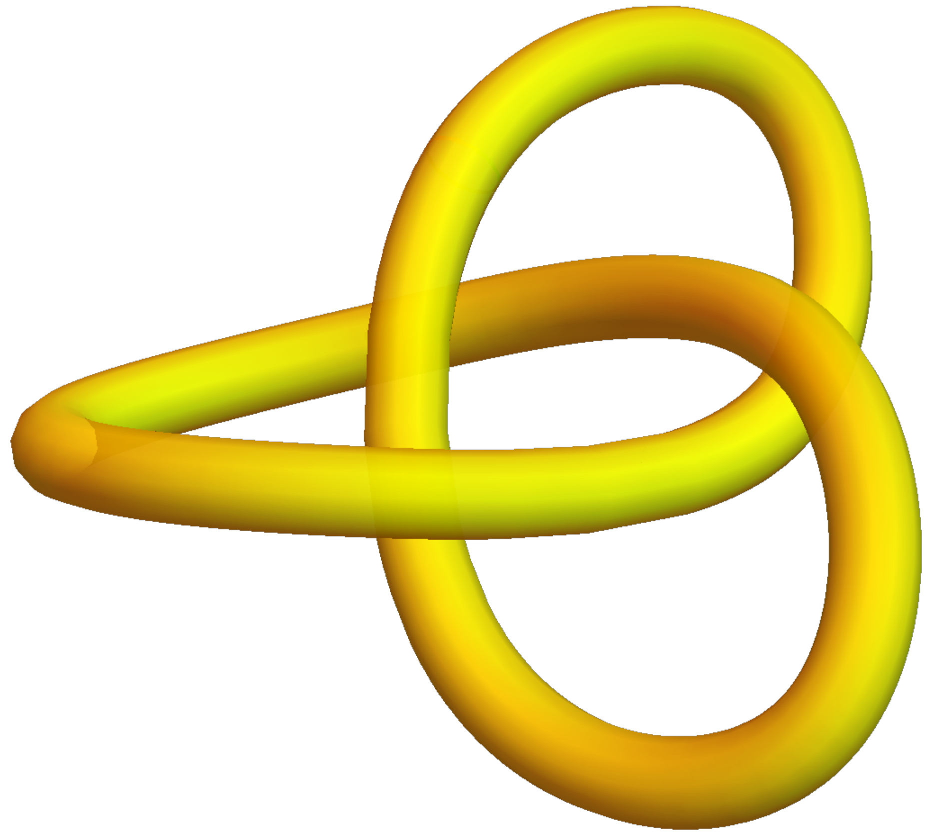


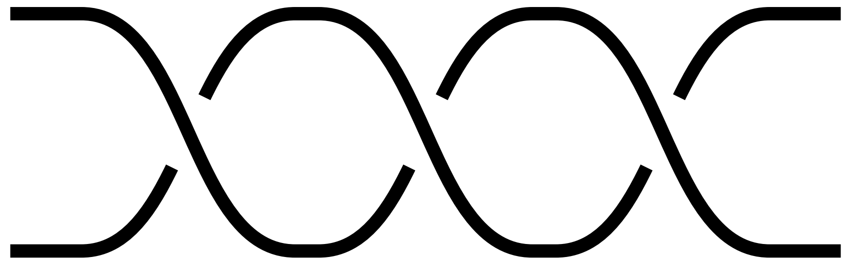
[1,-2, 3,-4, 5,-6]





[4,6,2]

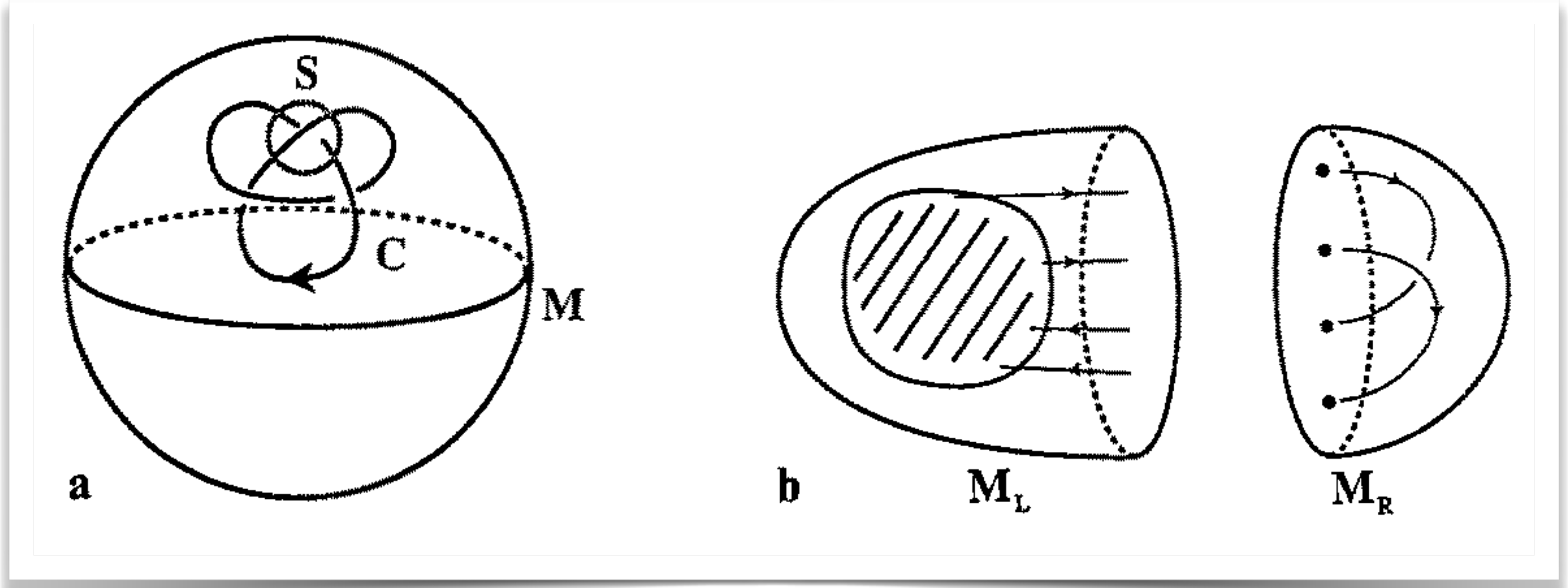




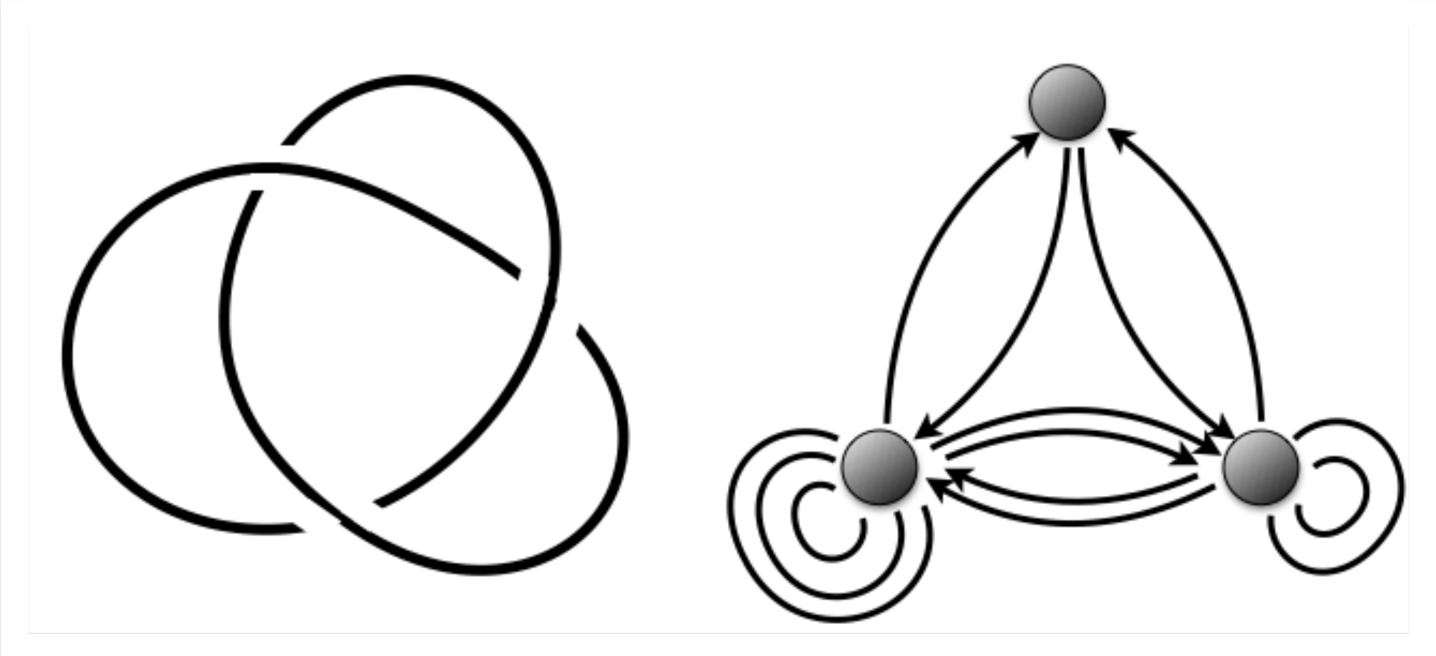
[1,1,1]

Intro to Knot Theory

Motivation

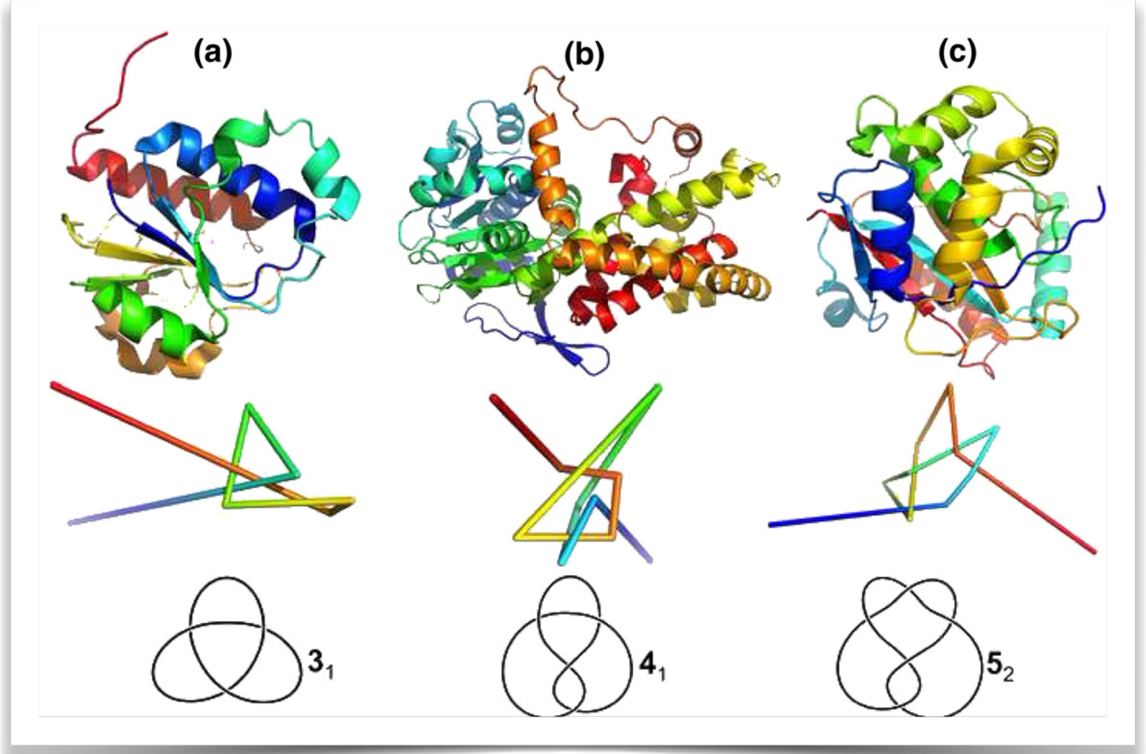


[Witten `89]



Knots	Quivers
Homological degrees, framing	Number of loops
Colored HOMFLY-PT	Motivic generating series
LMOV invariants	Motivic DT-invariants
Classical LMOV invariants	Numerical DT-invariants
Algebra of BPS states	Cohom. Hall Algebra

[Kucharski, Reineke, Stosic, Sulkowski `17]



[PFN Faisca `15]

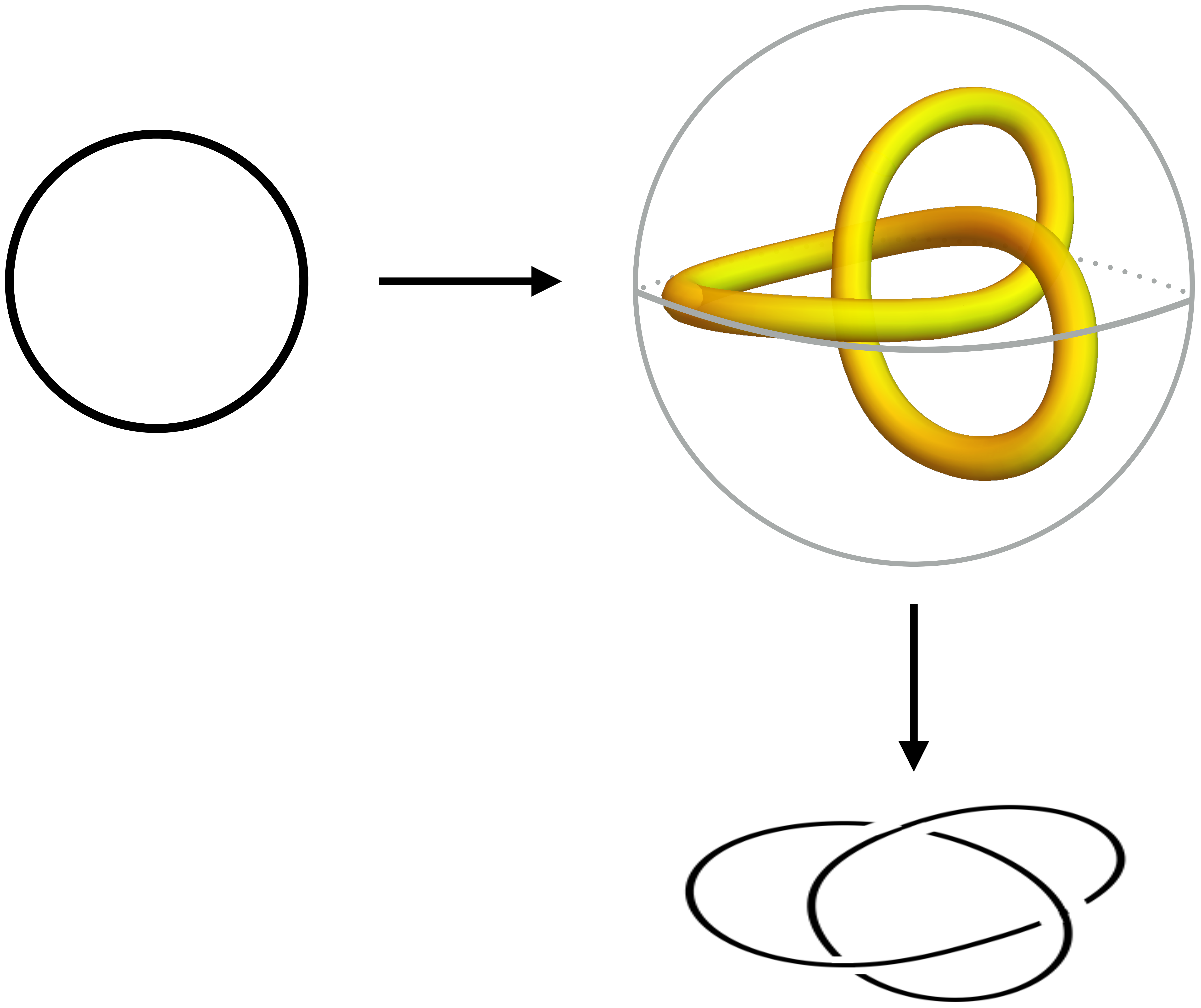
Is every 4D manifold that is homotopy equivalent to a 4-sphere diffeomorphic to the standard 4-sphere?

[Poincare 1904]

Knots - Definition

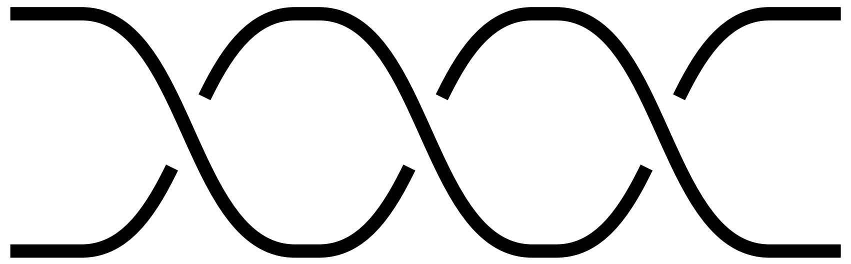
Definition (Knot):

A knot is an embedding of a circle into the three-sphere.

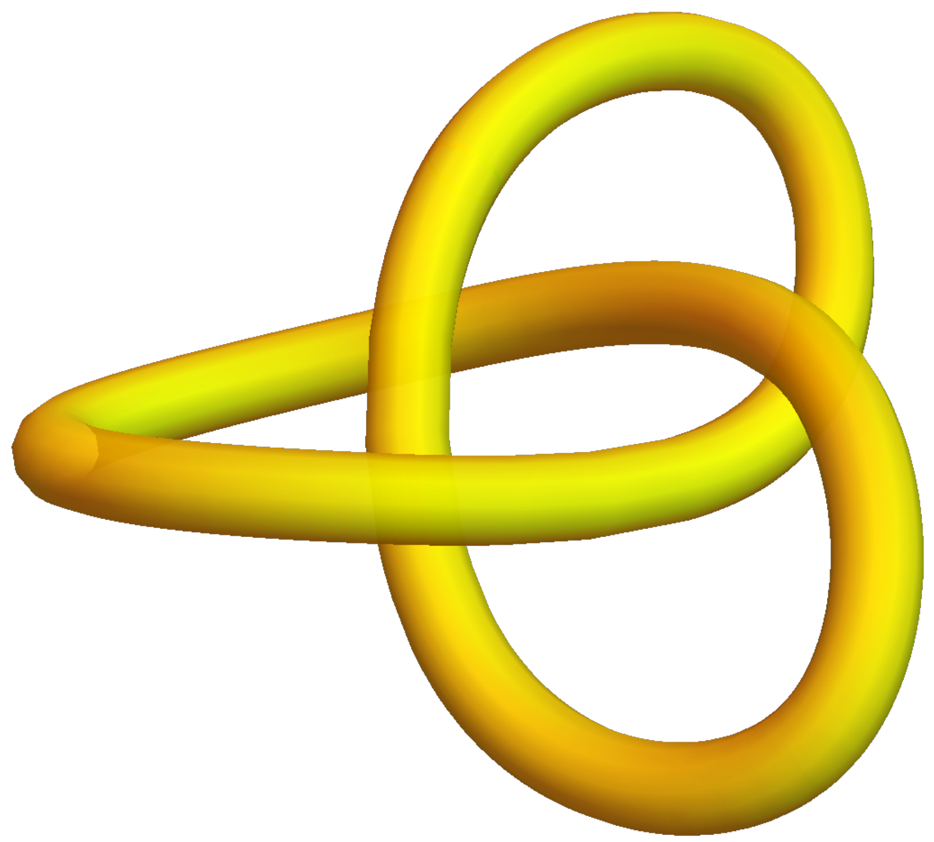


Knots - Representations

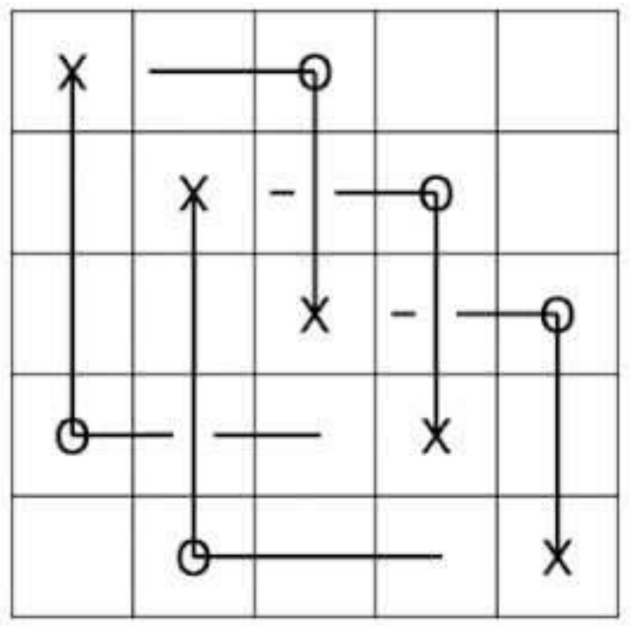
Braid



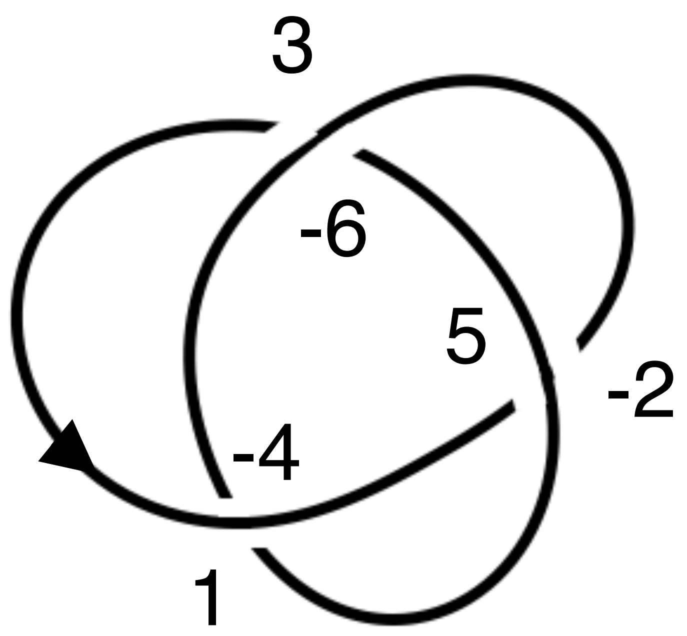
[1,1,1]



Grid Diagram

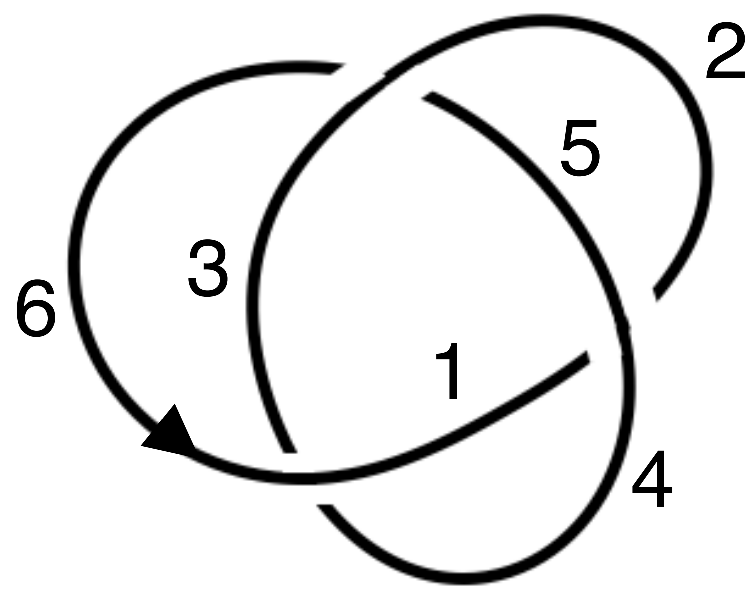


Gauss code



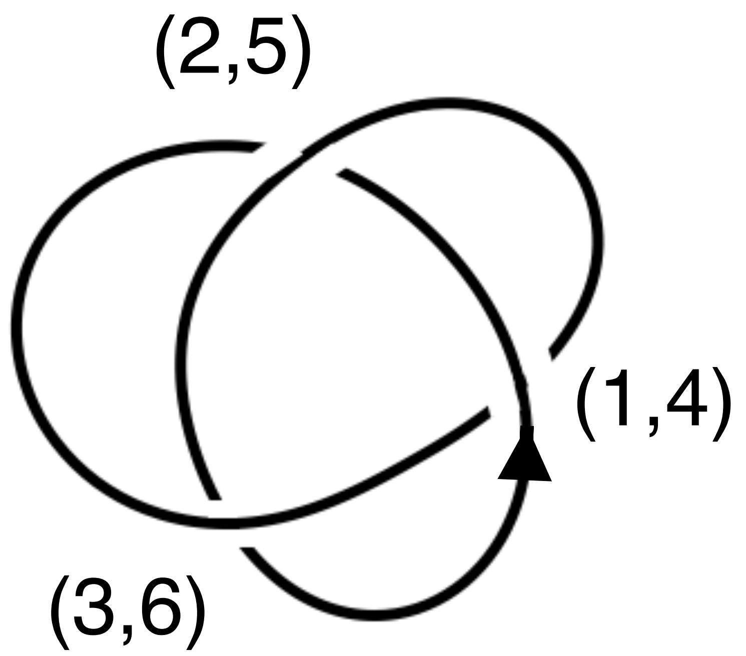
[1,-2, 3,-4, 5,-6]

Planar Diagram



[(1,4,2,5), (5,2,6,3), (3,6,4,1)]

DT Code



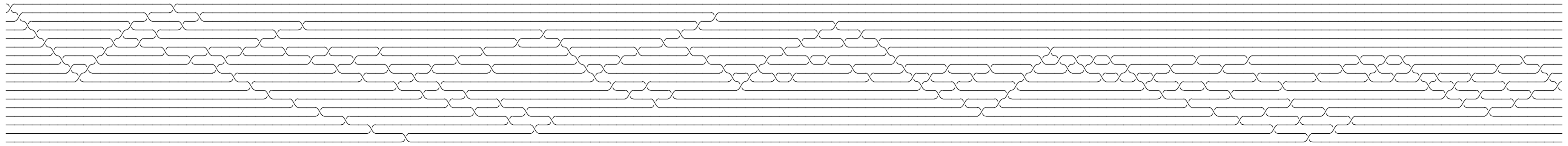
[4,6,2]

Which representation to choose?

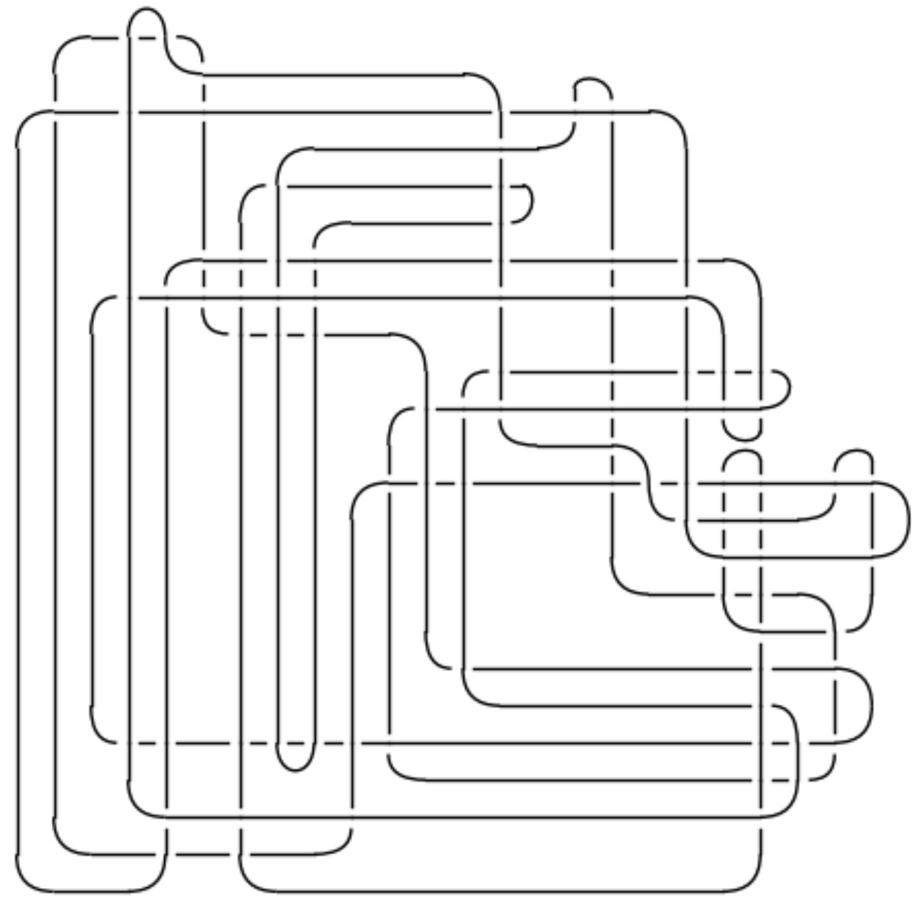
- In principle it should not matter: There exist algorithms to transform one representation to another; in practice, there are different aspects to this:
- CS
  - Which representations are most efficient, and does the ML algorithm make use of efficiency in encoding?
  - Can the problem be recast in a form that resembles problems that have been attacked by the CS community
- Math
  - Does a representation have a beneficial property that others do not?

Example 1: Efficiency of encoding

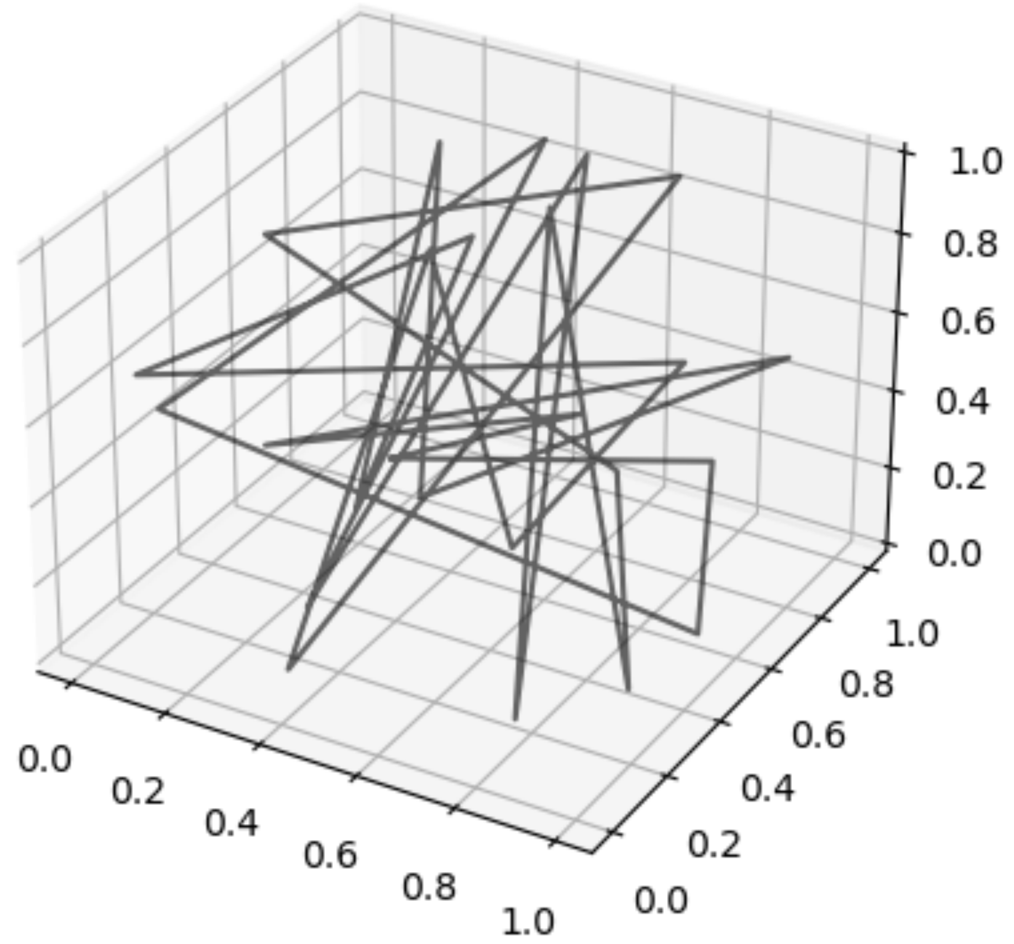
- The number of crossings in a knot projection is larger than the number of points needed to specify the knot in 3D **[Bar-Natan, Bar-Natan, Halacheva, Scherich `21]**
- This means that some quantities can be computed faster in 3D than in 2D…



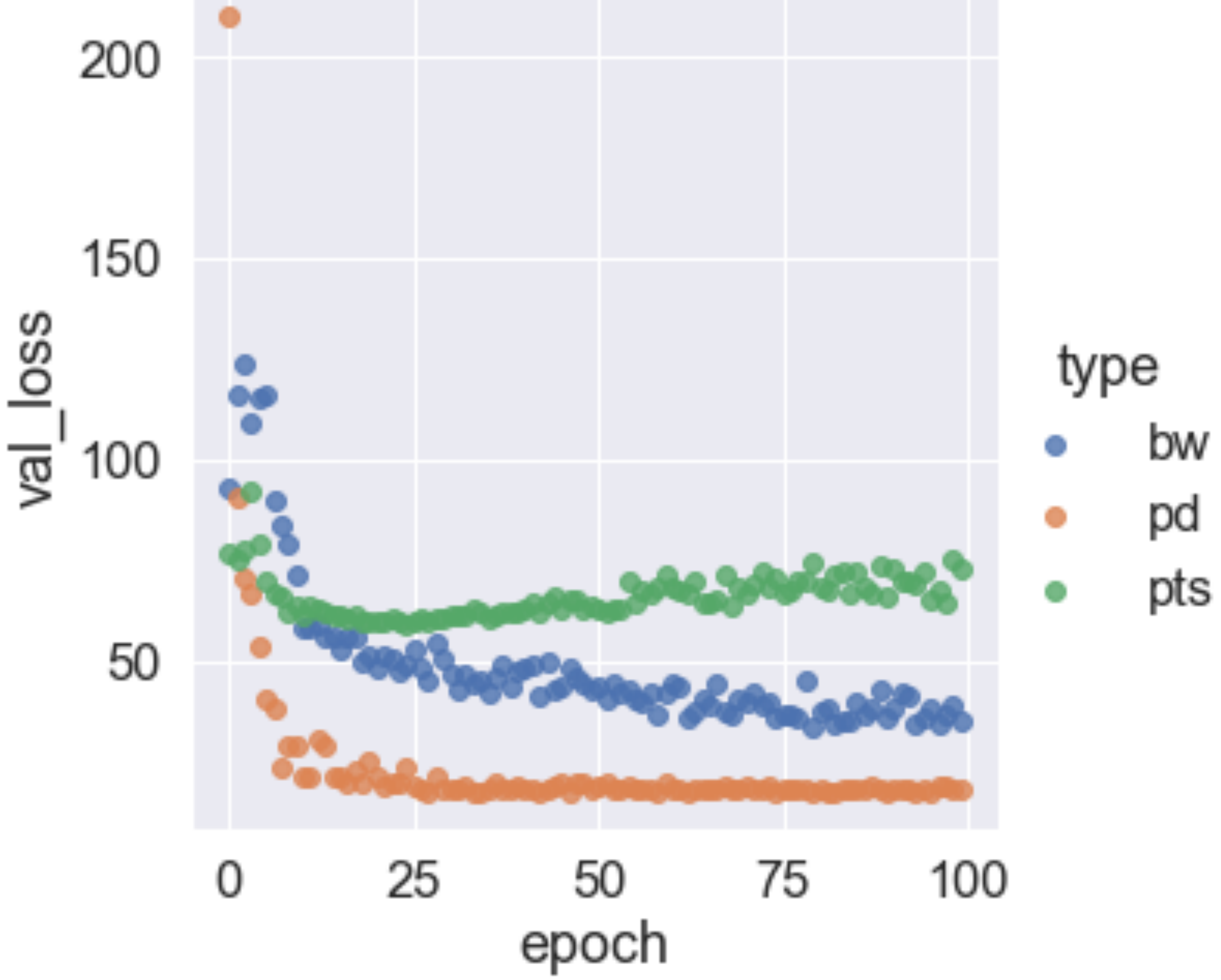
[-1, -2, 3, 4, -5, 6, 7, -8, -9, -8, 7, -6, 5, -4, -3, -5, 2, 4, 6, 1, 3, -7, 2, 6, 8, 7, 9, -6, 10, -5, 11, 4, 6, 12, 3, -7, 13, -6, 8, 14, -7, 9, 15, -6, 8, 10, 16, 9, 11, -8, 10, 12, 7, 11, 13, 6, 8, 12, 14, -5, 13, 15, -4, 14, -5, -5, 5, 6, 7, 8, 9, 8, -10, -7, -11, 6, -10, 12, 5, -11, 4, 6, -3, 7, -2, 8, 9, -10, -9, 8, -7, 9, 6, -9, 5, 7, -4, -7, -3, 5, 8, -4, 9, 5, 6, -7, 8, -9, 10, -9, 11, -8, 10, -12, 9, -13, 8, -12, -11, -10, 9, -8, 7, -6, -7, 8, 7, -7, 7, 8, -7, 9, 7, -9, -8, 9, 10, -9, 11, 8, -10, 12, -7, 10, 13, -8, 11, -14, -7, 9, -13, -15, 10, -12, -14, -16, 9, -13, -15, 8, -14, 7, 9, 8, -7, -9, 7, -8, -9, 10, 9, -11, -10, -12, -9, -11, -13, -8, -10, -12, 7, -11, -8, -9, -10]



[[0, 70, 1, 69], [1, 116, 2, 115], [164, 2, 165, 3], [3, 92, 4, 91], [4, 22, 5, 21], [173, 6, 78, 5], [29, 7, 30, 6], [7, 97, 8, 98], [8, 138, 9, 139], [9, 59, 10, 58], [10, 59, 11, 60], [11, 104, 12, 103], [12, 104, 13, 105], [13, 106, 14, 105], [14, 62, 15, 61], [15, 54, 16, 55], [16, 38, 17, 37], [17, 142, 18, 141], [18, 153, 19, 152], [19, 149, 20, 150], [78, 20, 79, 21], [22, 92, 23, 93], [23, 133, 24, 134], [168, 25, 169, 24], [169, 25, 170, 26], [26, 95, 27, 94], [27, 136, 28, 135], [172, 29, 173, 28], [30, 151, 31, 150], [31, 151, 32, 152], [32, 140, 33, 141], [33, 57, 34, 56], [34, 100, 35, 101], [35, 102, 36, 101], [55, 37, 56, 36], [142, 38, 143, 39], [153, 39, 154, 40], [148, 41, 149, 40], [111, 42, 112, 41], [160, 43, 161, 42], [66, 44, 67, 43], [86, 45, 87, 44], [45, 84, 46, 83], [46, 84, 47, 85], [85, 47, 86, 48], [48, 66, 49, 65], [159, 49, 160, 50], [50, 111, 51, 110], [147, 51, 148, 52], [52, 154, 53, 155], [143, 54, 144, 53], [57, 99, 58, 100], [60, 103, 61, 102], [62, 106, 63, 107], [157, 63, 158, 64], [158, 65, 159, 64], [87, 68, 88, 67], [68, 83, 69, 82], [116, 70, 117, 71], [125, 71, 126, 72], [130, 73, 131, 72], [121, 74, 122, 73], [120, 74, 121, 75], [119, 76, 120, 75], [128, 76, 129, 77], [127, 0, 128, 77], [90, 80, 91, 79], [113, 81, 114, 80], [162, 82, 163, 81], [161, 88, 162, 89], [89, 113, 90, 112], [93, 134, 94, 135], [170, 96, 171, 95], [137, 96, 138, 97], [139, 99, 140, 98], [107, 157, 108, 156], [145, 108, 146, 109], [146, 110, 147, 109], [163, 115, 164, 114], [126, 117, 127, 118], [129, 119, 130, 118], [122, 168, 123, 167], [132, 123, 133, 124], [165, 125, 166, 124], [166, 131, 167, 132], [171, 137, 172, 136], [144, 156, 145, 155]]



[[0.63, 0.0, 0.73], [0.46, 1.0, 0.17], [0.0, 0.59, 0.17], [1.0, 0.78, 0.63], [0.23, 0.78, 0.0], [0.33, 0.69, 1.0], [0.56, 0.0, 0.59], [0.22, 1.0, 0.56], [0.0, 0.17, 0.61], [1.0, 0.37, 0.86], [0.47, 0.71, 0.0], [0.52, 0.33, 1.0], [0.42, 0.0, 0.13], [0.66, 1.0, 0.85], [0.0, 0.61, 0.71], [1.0, 0.13, 0.75], [0.89, 0.39, 0.0], [0.62, 0.57, 1.0], [0.88, 0.0, 0.18], [0.46, 1.0, 0.84], [0.25, 0.34, 0.0], [0.49, 0.79, 1.0], [0.0, 0.24, 0.48], [1.0, 0.45, 0.14], [0.74, 1.0, 0.14]]



[work in progress]

Example 2: Transformation to other problems

- Different representations encode topological info in different ways (e.g. braid words can be read left to right, DT codes are harder to "picture")
  - Braid *words* might lend themselves to ML techniques developed for Natural Language Processing
  - Grid *Diagrams* to ML techniques developed for Computer Vision or Graph Networks
- Example: Unknot recognition problem

Empirical observation: Braid words work better than Gauss or DT

  - Perhaps the sequential representation is "less confusing" for the neural network even though the description is less efficient **[Gukov, Halverson, FR, Sulkowski `20]**

Example 3: Mathematical properties of representation

- There are *hard unknots* that require adding braid generators (crossings) before being able to simplify

[Kauffman, Lambropoulou `06; Tuzun, Sikora `16; Burton, Chang, Löffler, Mesmay, Maria, Schleimer, Sedgwick, Spreer `21]

> - Is this property preserved under maps such as Vogel's algorithm?
> - On specific knots, is one representation simpler than another? Does the algorithm benefit from seeing both representations?

- This property is absent in Grid Diagrams with Dynnikov moves **[Dynnikov `06]**
- Kauffman et.al. used this to design an ML algorithm that monotonically decreases crossings **[Kauffman, Russkikh, Taimanov `20]**

> Can my algorithm deal with the fact that it needs to make knots more complicated to ultimately simplify them further?

Recap

1 Hard problems

- Hierarchy of hardness
- Combinatorially hard problems in NP good for RL

2 Reinforcement Learning

- Solve MDP
- Approximate policy, SVF, AVF by NN

3 Intro to Knot Theory

- Knot is $S^1 \hookrightarrow S^3$
- Representation of data informs algorithm and NN architecture:
  - RL vs monotonic optimizer
  - CNN vs Transformer vs …