# ECON441B : Intro Machine Learning Lab

## Week 10, Lecture 10 | Reinforcement Learning

Sam Borghese

Wednesday, March 13th, 2024

1. Unsupervised Learning
2. Wen Reinforcement Learning?
3. K-means Clustering
4. Coding
5. In-Class Assignment

# Unsupervised Learning

# Quadrants of Machine Learning tasks

Been taught that explanatory models and predictive models can fit into one of these categories

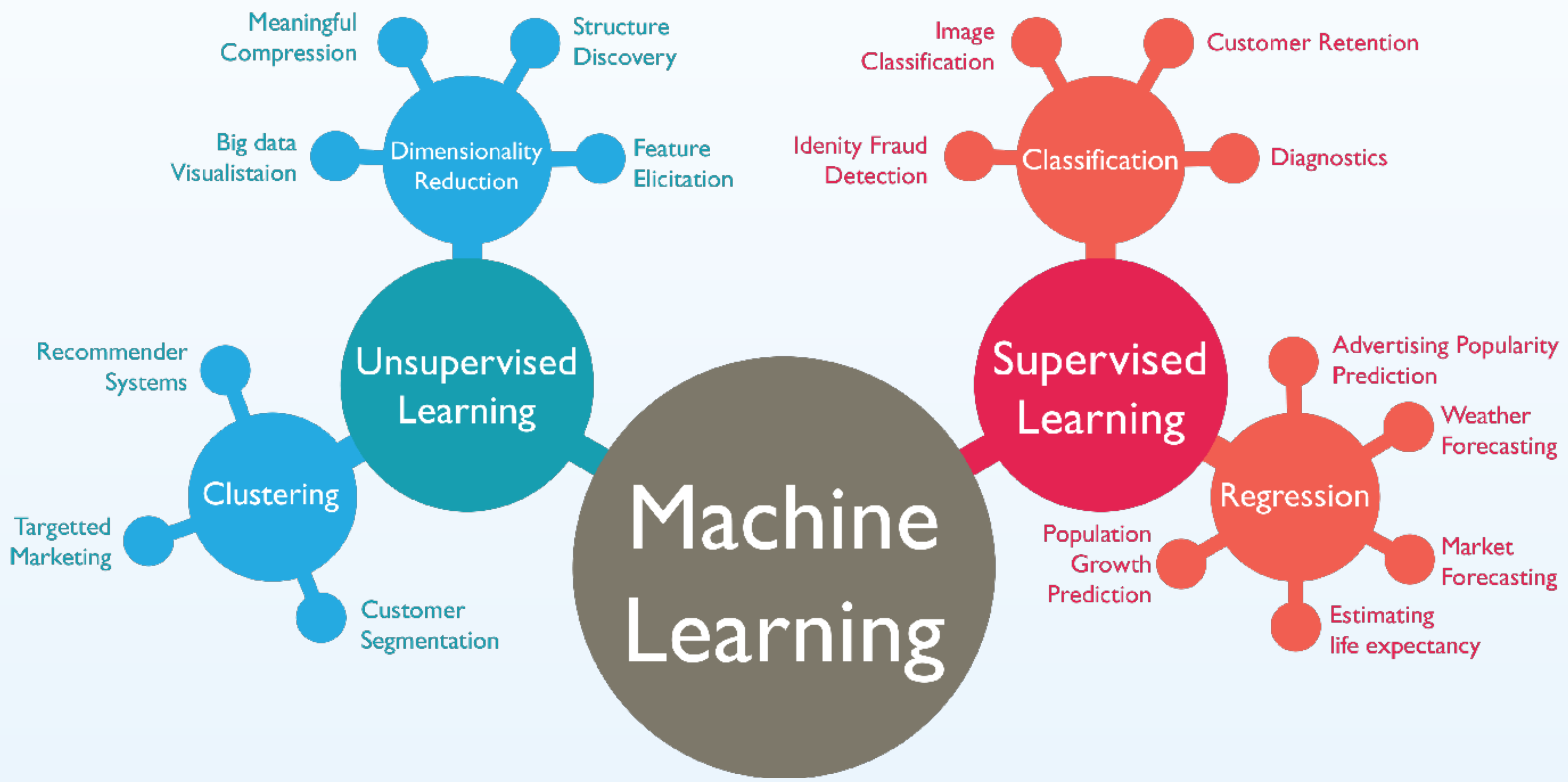
| | Supervised Learning | Unsupervised Learning |
|---|---|---|
| Discrete | classification or categorization | clustering |
| Continuous | regression | dimensionality reduction |

# Branches of Machine Learning

Creates a Series of Non-linear indicators

- <u>Two Questions to ask yourself :</u>

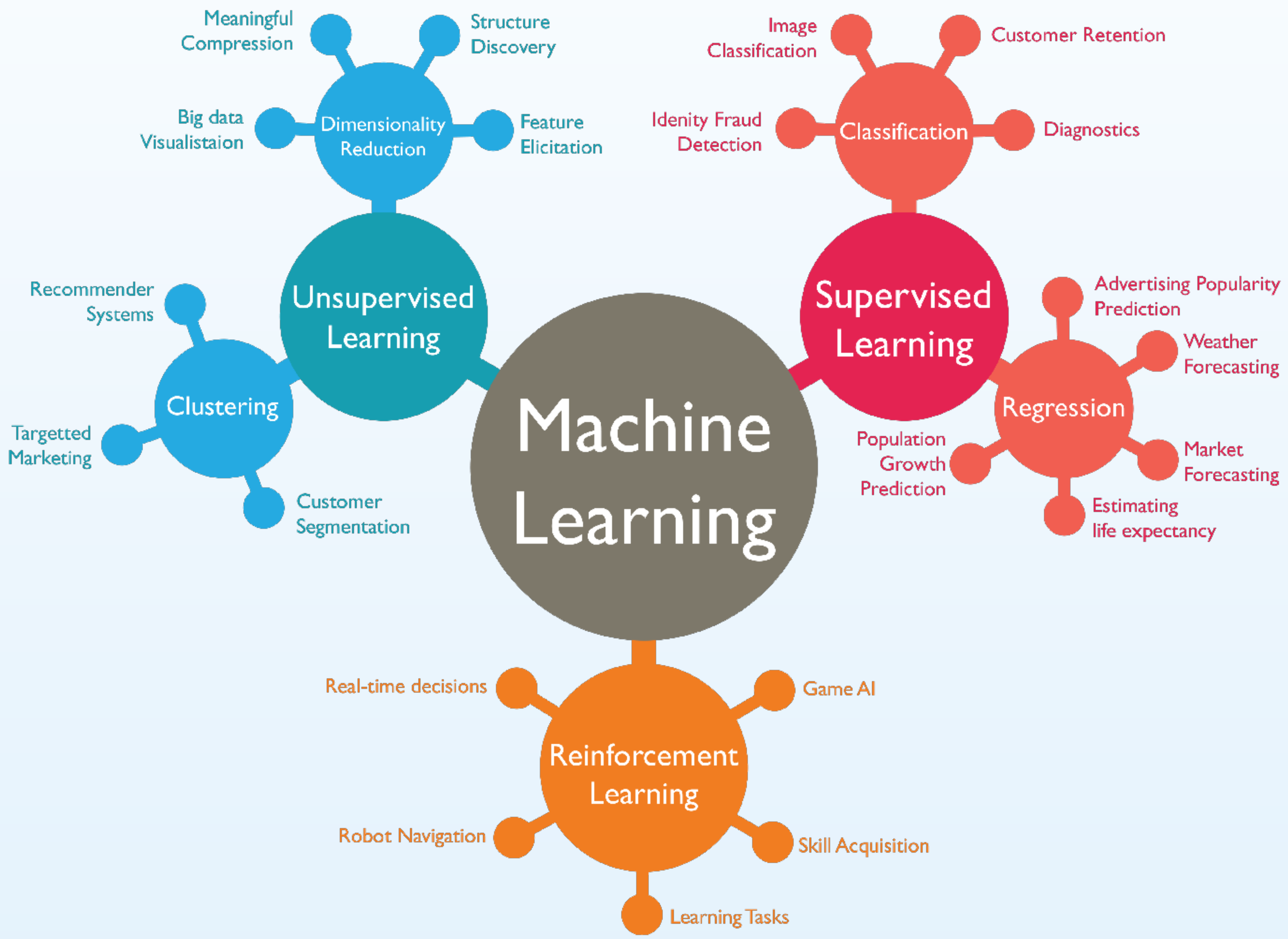
- Is there a measurable dependent variable?
- Is the desired output continuous or discrete?
- These encompass all of the ML models learned in the program:
- RNN, Multivariate regression, Decision trees, PCA, etc…



# Third Branch of Machine Learning

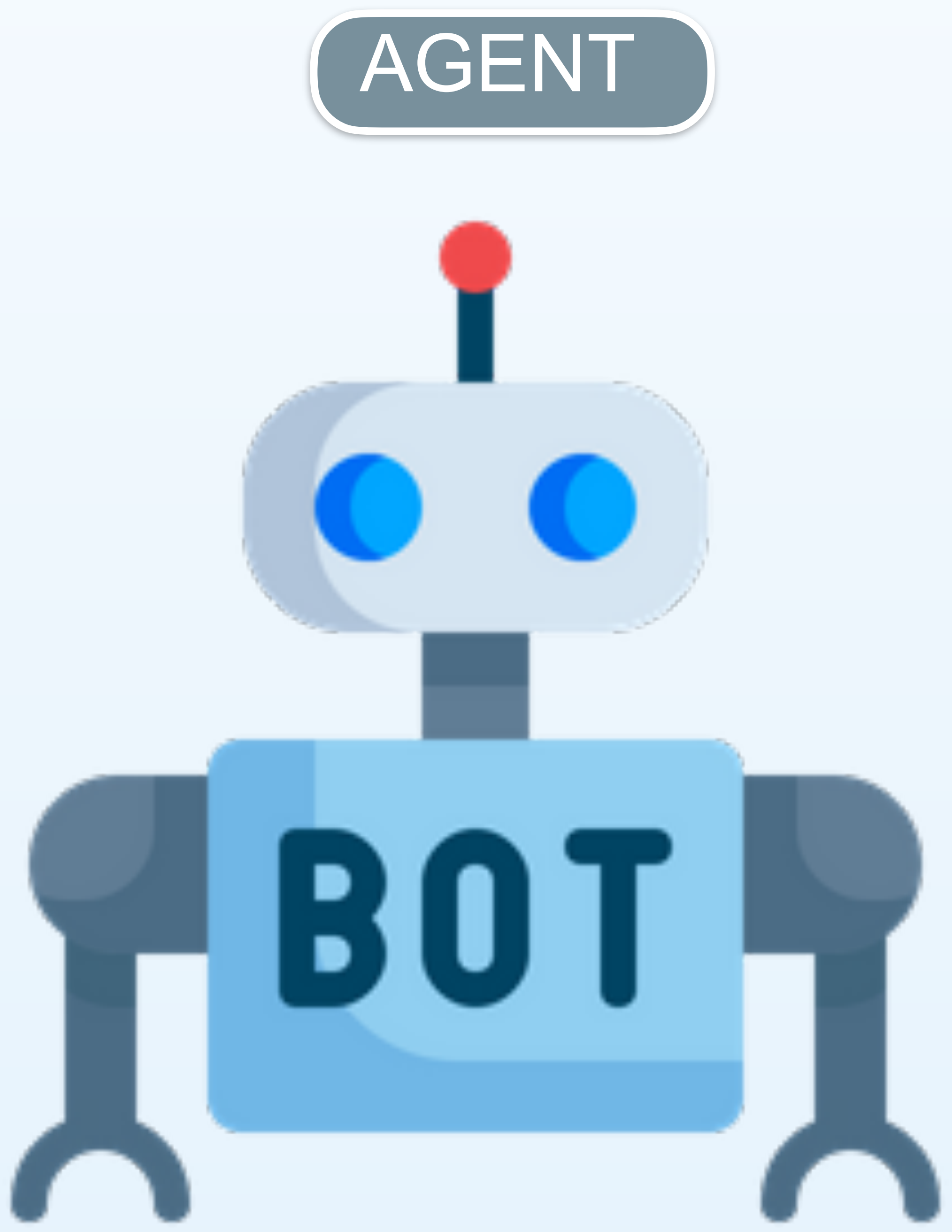
## Creates a Series of Non-linear indicators

- Train an "Agent" to make decisions as data comes in
- Decision framework is updated in real-time
- Modeled after the training of Humans and Dogs
- Psychological concept was introduced in the 60s, while in the 1980s it was thought of with AI



# Agent

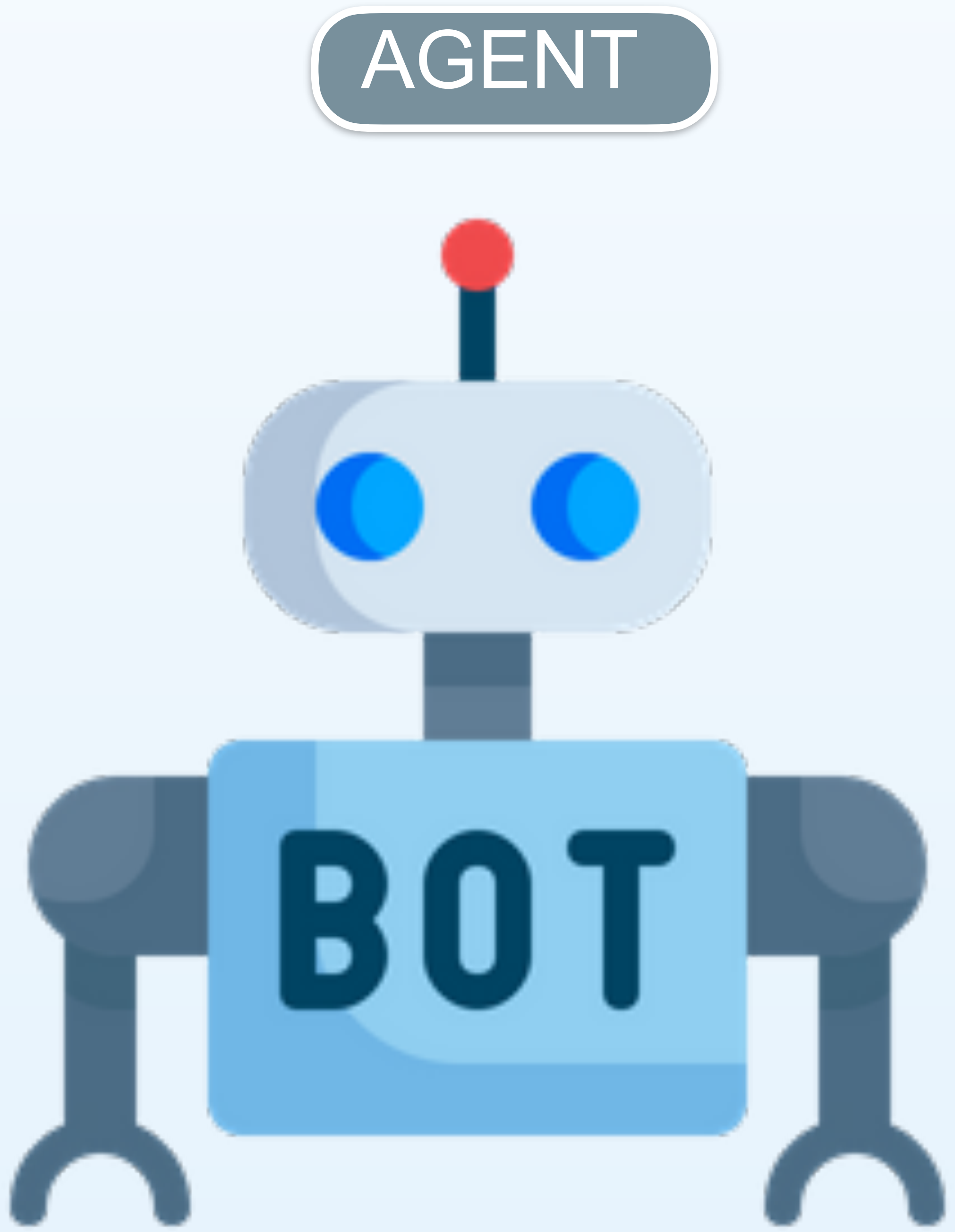
One who takes decisions based on the rewards and punishments



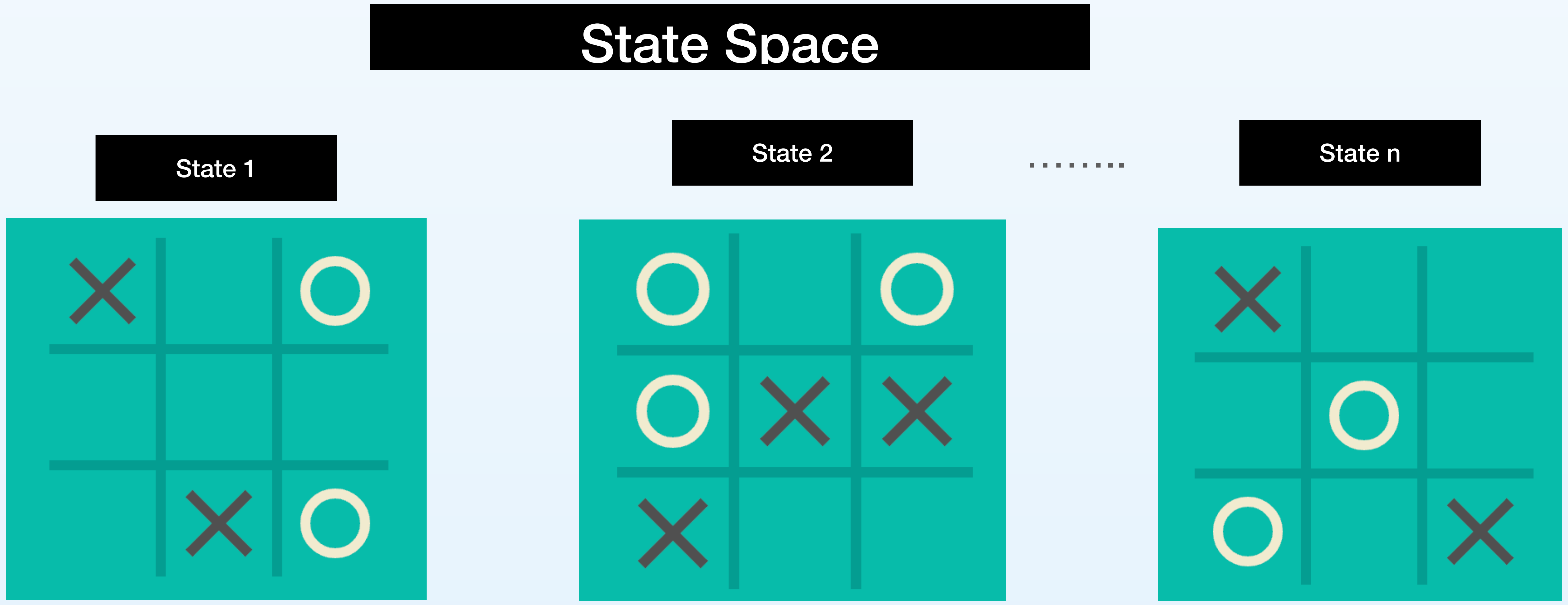

- Has a policy (Rules) on how they make decisions (take there actions)
- This is the algorithm you develop
- Many ways to build an agent but we will start with the simplest

# Environment/State

Environment is the "world" in which the agent interacts with

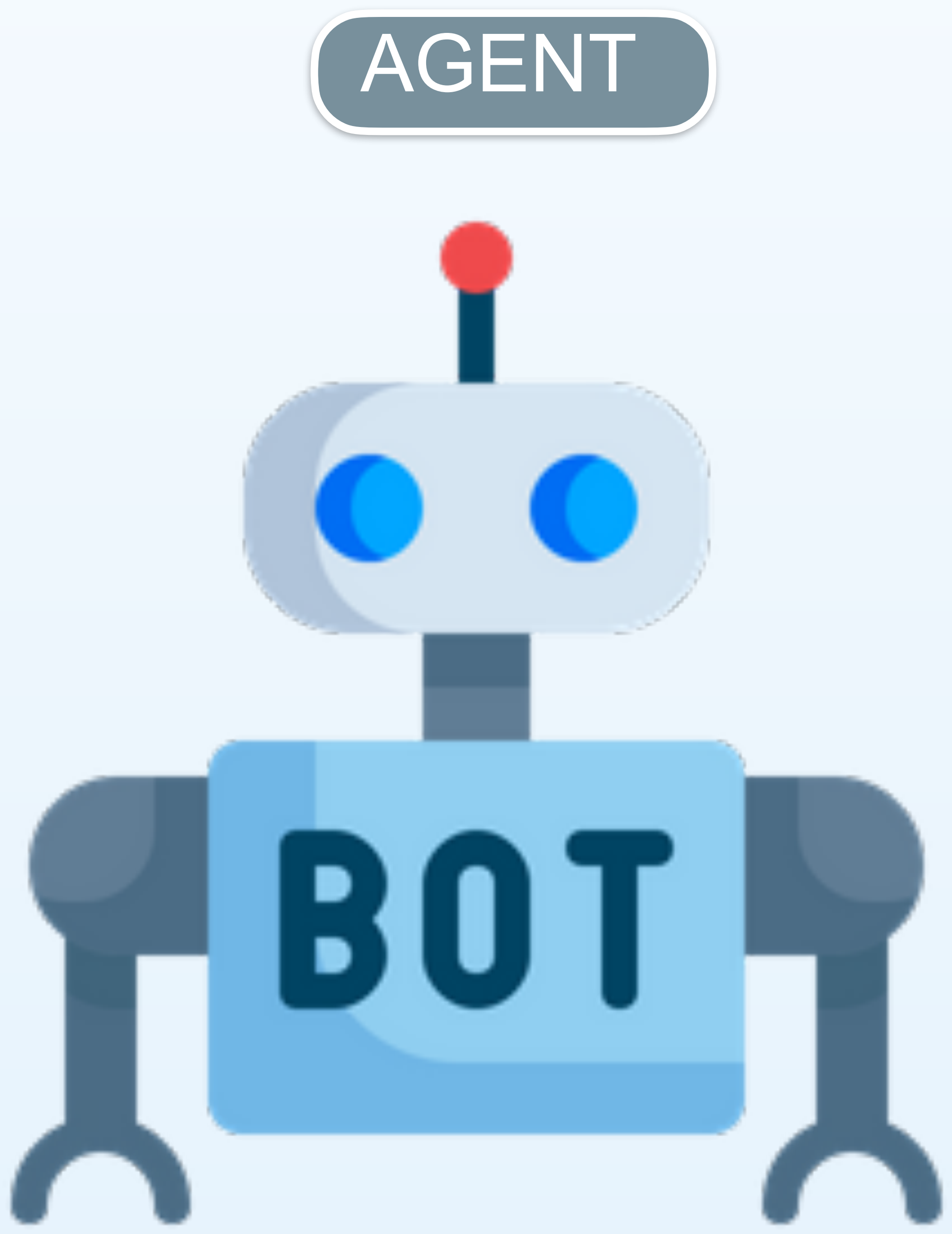



- A state is the current attributes of the environment
- All states together are the state space
- The environment is always changing into different states



# Actions

Predefined set of responses that an Agent can take

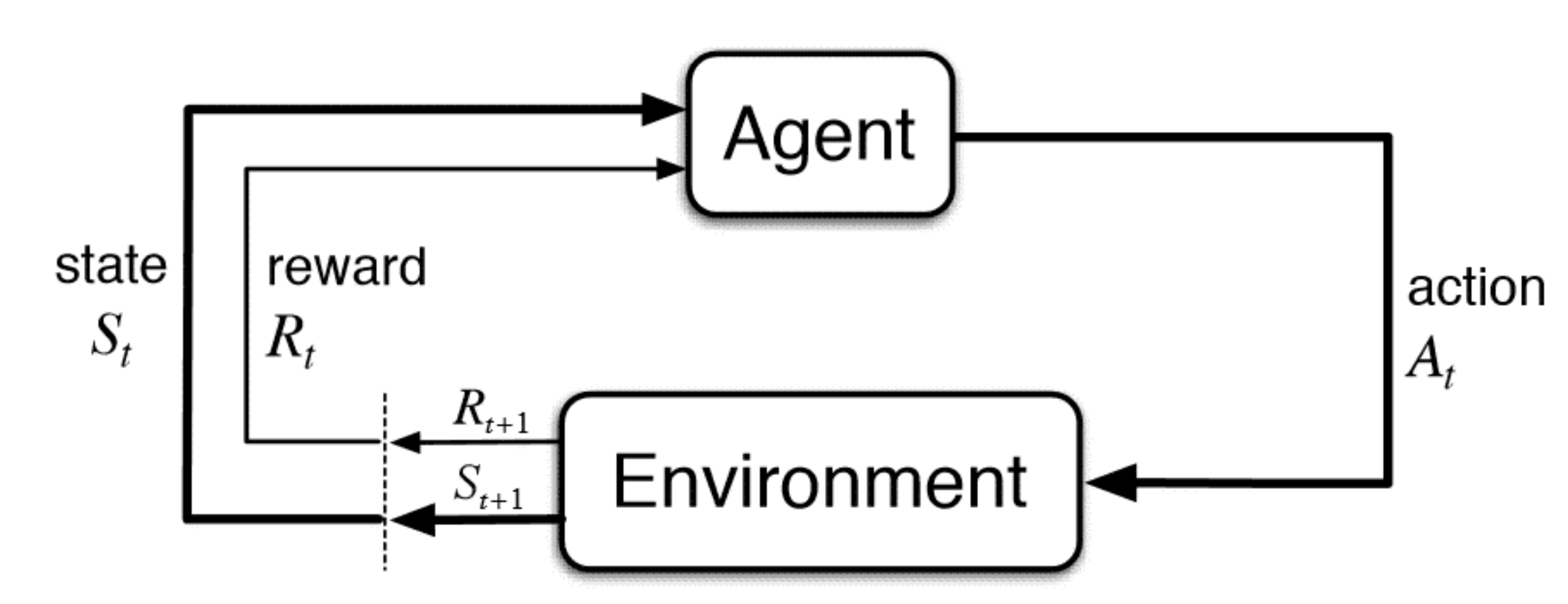



- The actions interact with the environment
- The actions can be limited by the given state
- The action taken gives some reward back to the Agent
- The action can change the environment

# RL Framework

Given a State - An Agent will choose an action based on a policy

The action will return the Agent a new State and some reward

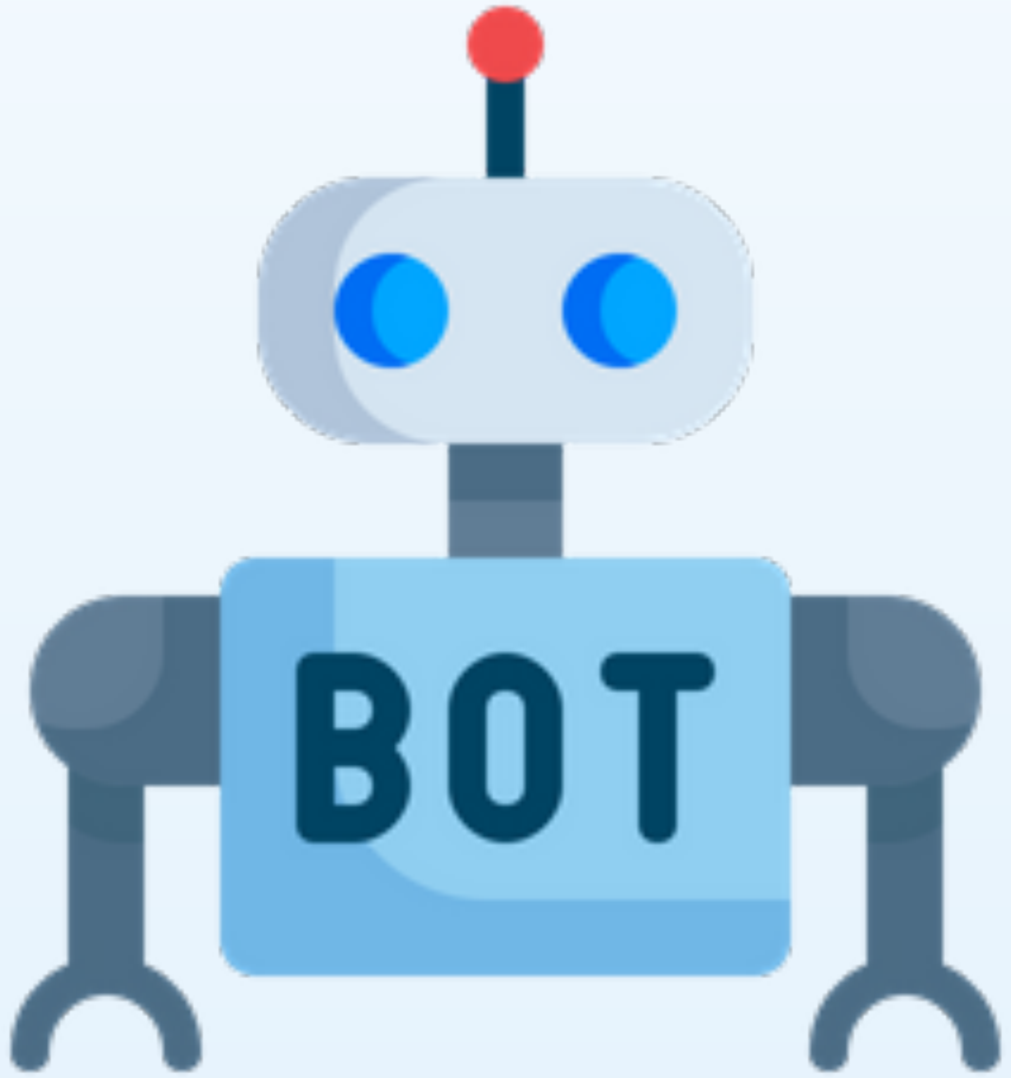


# Actions

Predefined set of responses that an Agent can take

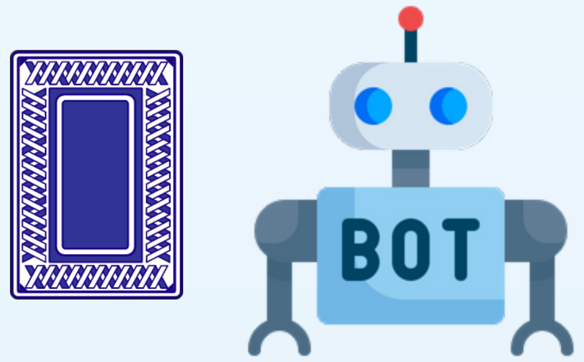
- Agent has to make a decision
- Let’s say policy says the best action is to bet 10
- => Opponent raises 20

AGENT




## Card Game

Opponent



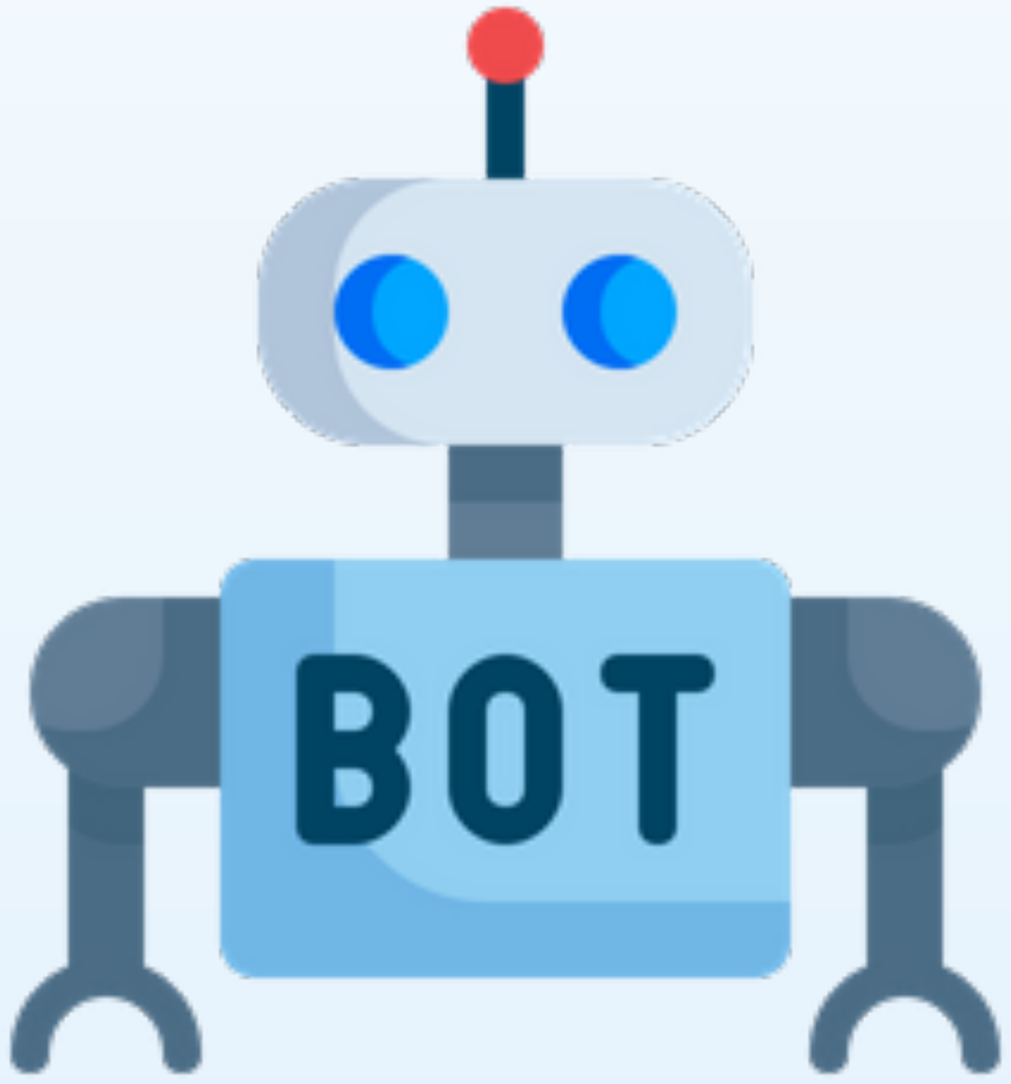

$$actions \in \{fold,\ \text{bet } 0, \text{ bet } 10, \text{ bet } 20, ..., \text{ bet } 1000\}$$

# Actions

Predefined set of responses that an Agent can take

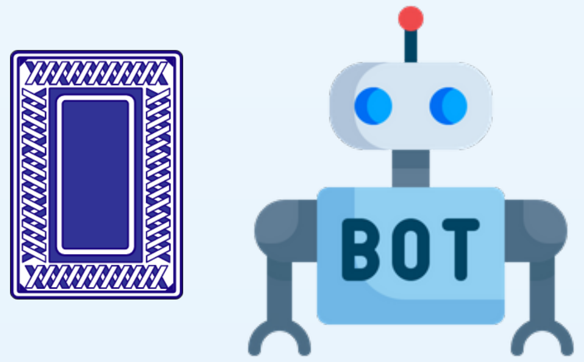
- We do NOT know what the outcome would have been from taking the other actions
- Our only feedback is the reward

AGENT




## Card Game

Opponent




Changes state to opponent bets 20

$$actions \in \{fold, \text{bet } 0, \text{bet } 10, \text{bet } 20, \ldots, \text{bet } 1000\}$$

# Fifth Quadrant of ML

Supervised Learning : Make a prediction you know if you are wrong and by how much

Unsupervised learning : there is no metric to gauge how correct you are

| | *Supervised Learning* | *Unsupervised Learning* |
|---|---|---|
| *Discrete* | classification or categorization | clustering |
| *Continuous* | regression | dimensionality reduction |

Card Game

X

# When Reinforcement Learning?

# Why not just use metric-targeted supervised learning

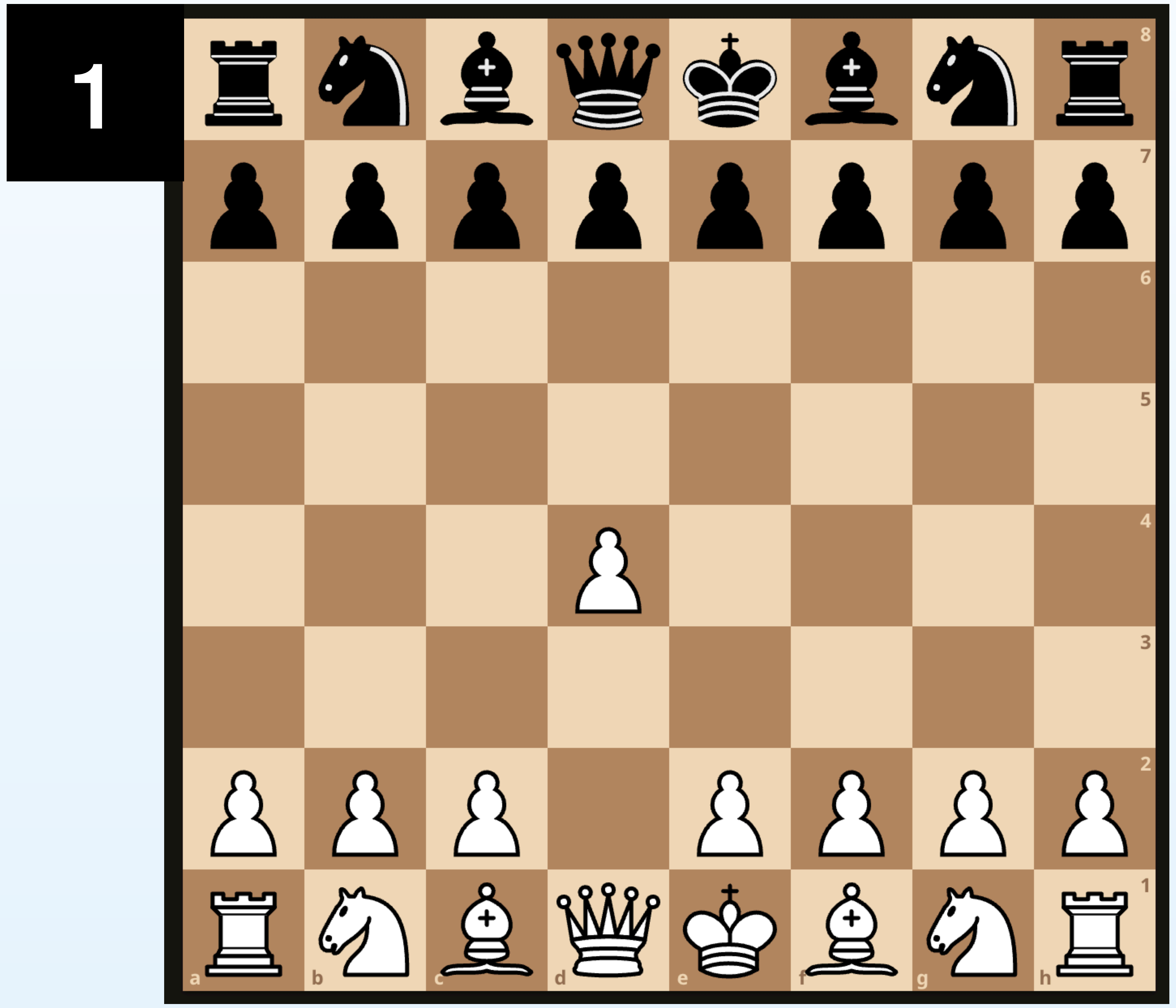
We use reinforcement learning when the action taken by the model determines the next state

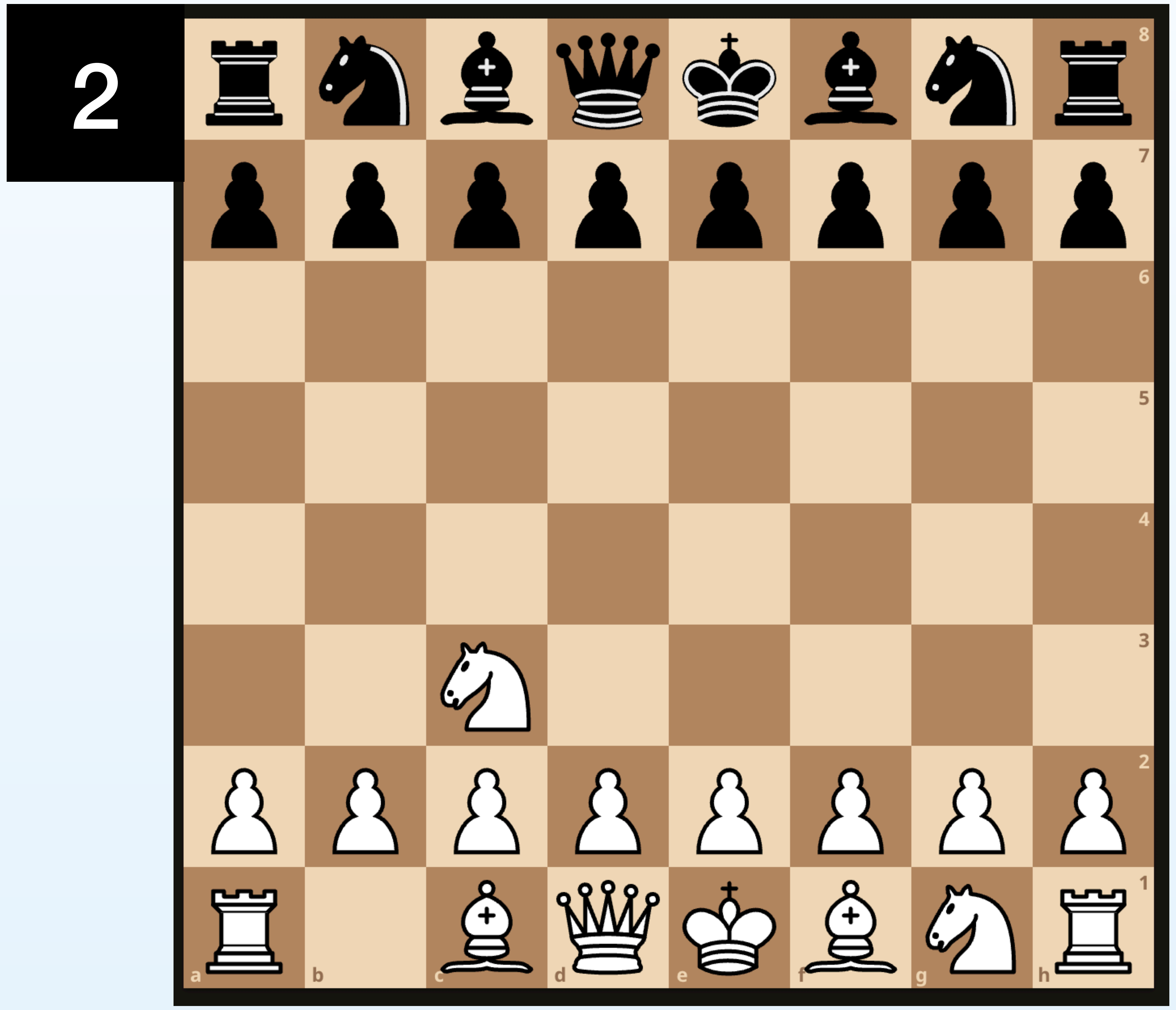
AND

We do not know what the other states would be if the other actions had been taken

# Chess as an Example

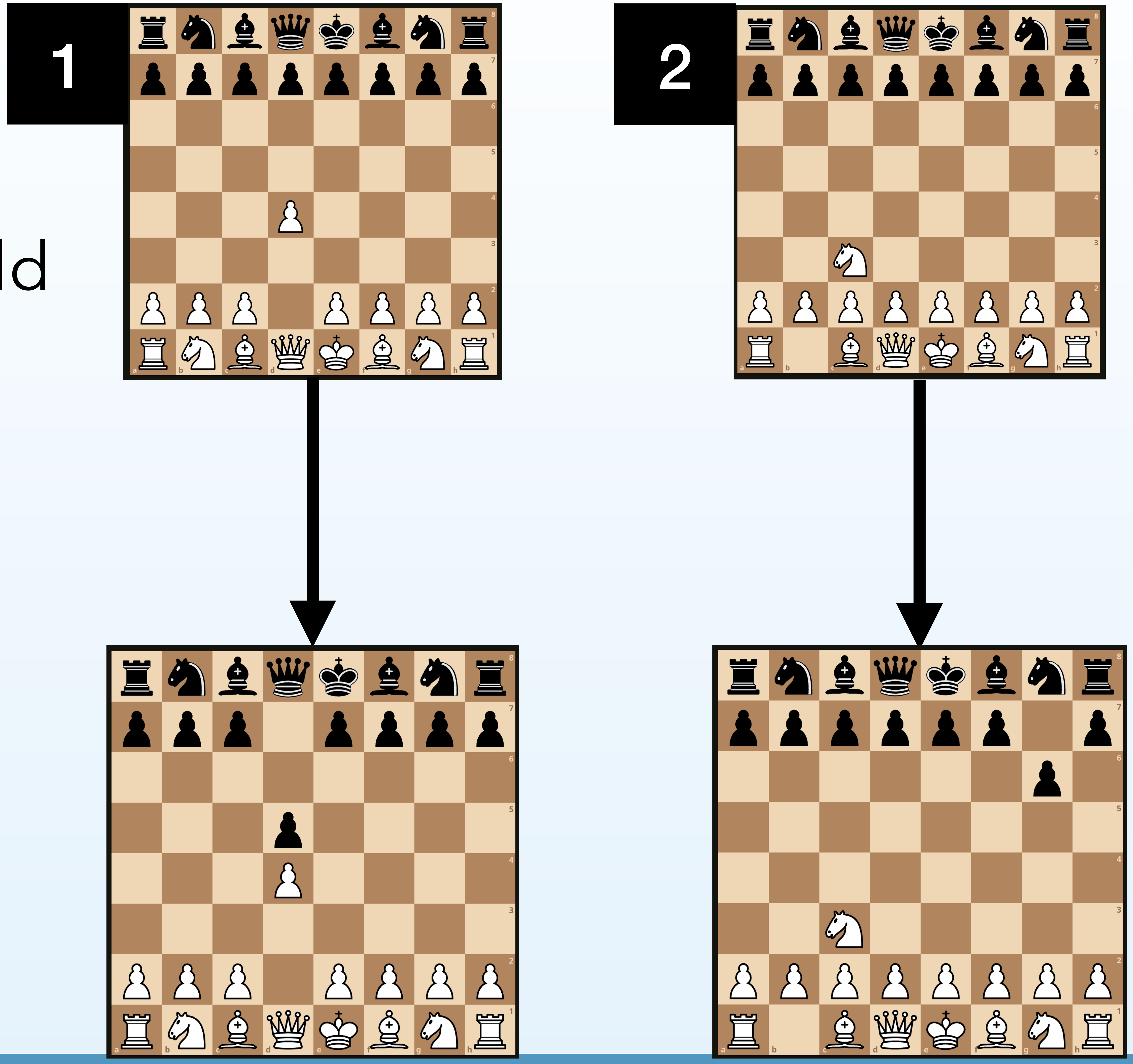
Lets say you have two options for starting moves



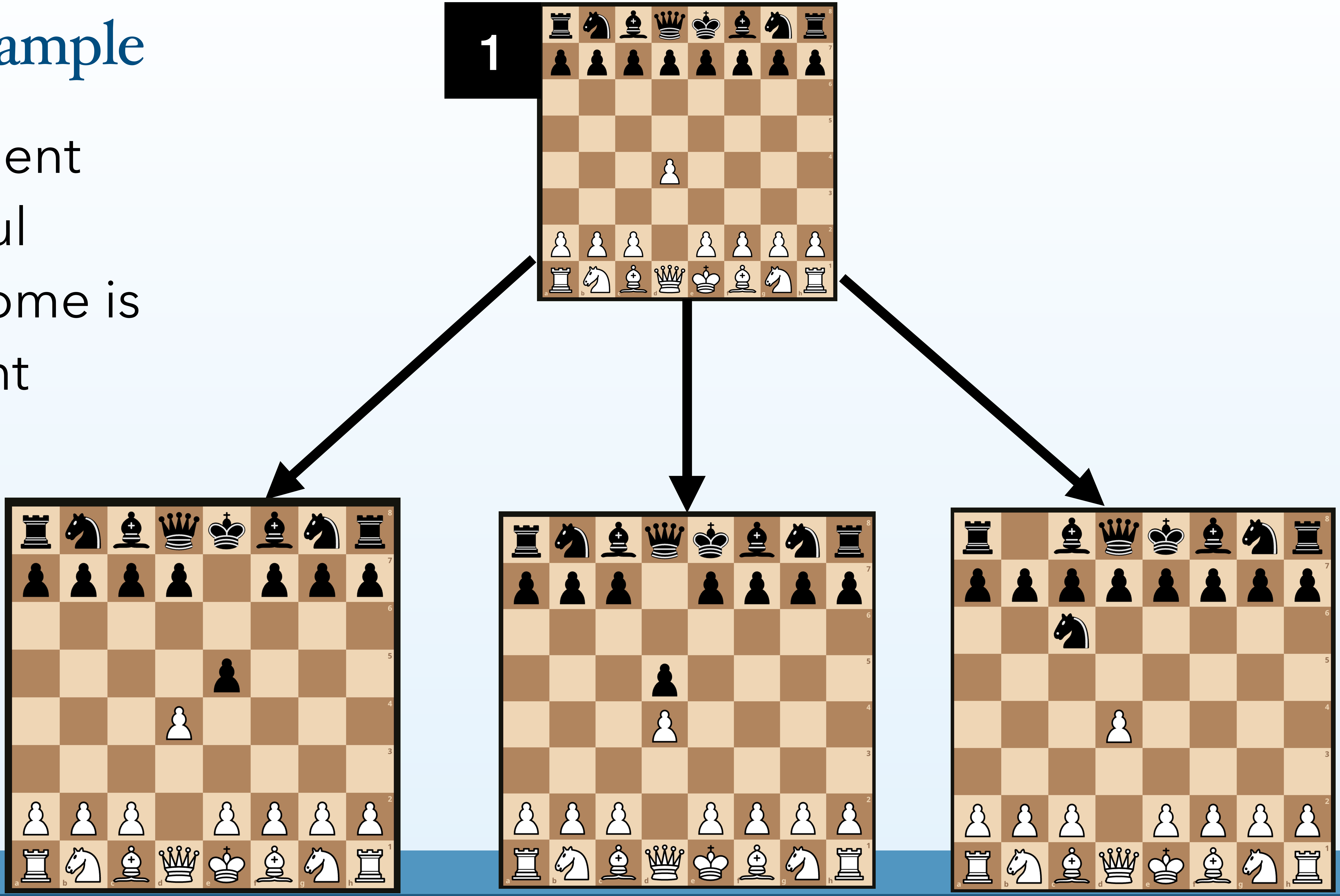
# Chess as an Example

If you take option 1 you do not know what your opponent would have done if you took option 2



# Chess as an Example

Also reinforcement learning is useful where the outcome is non-determinant

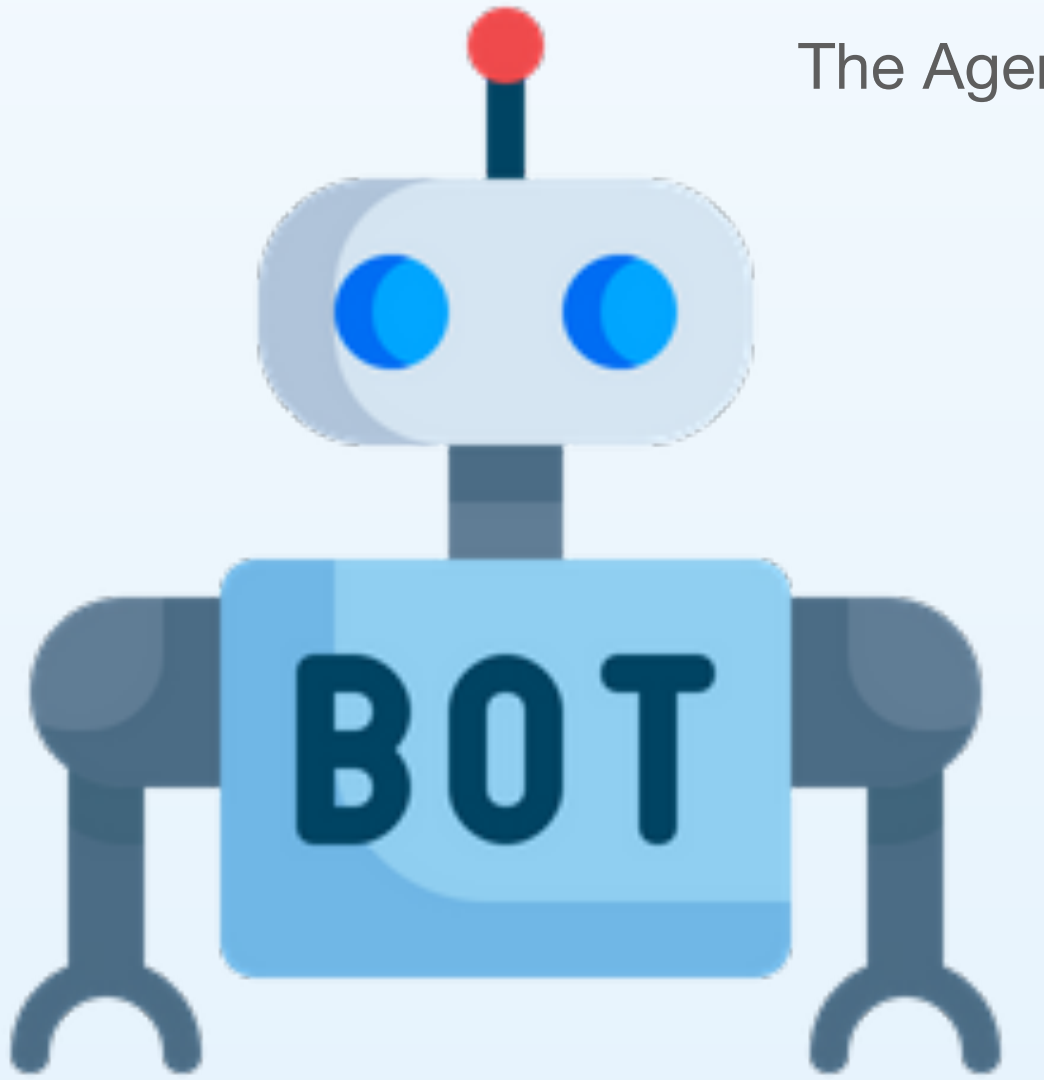


# K-armed Bandit Problem

# K-armed Bandit Problem

A decision must be made at each time to try and maximize expected gain

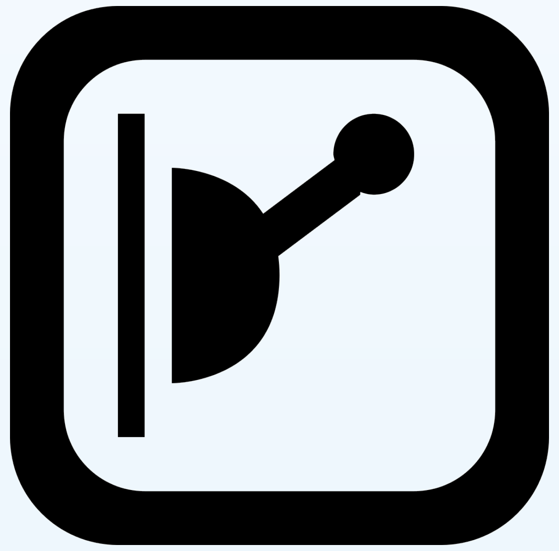
AGENT



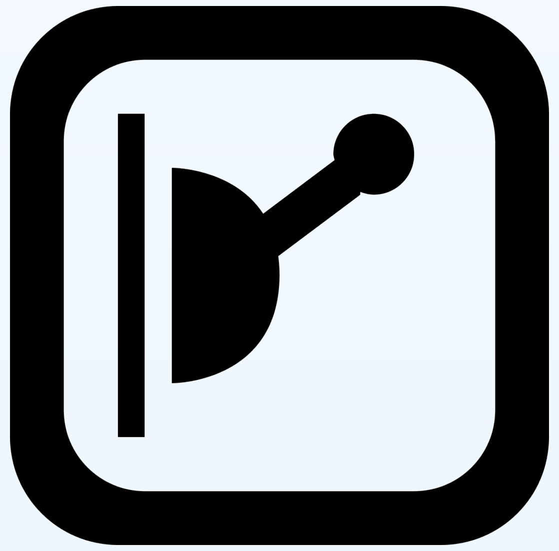

$$a \in \{1,2\}$$

The Agent can choose to pull lever 1 or 2

Lever 1



Lever 2

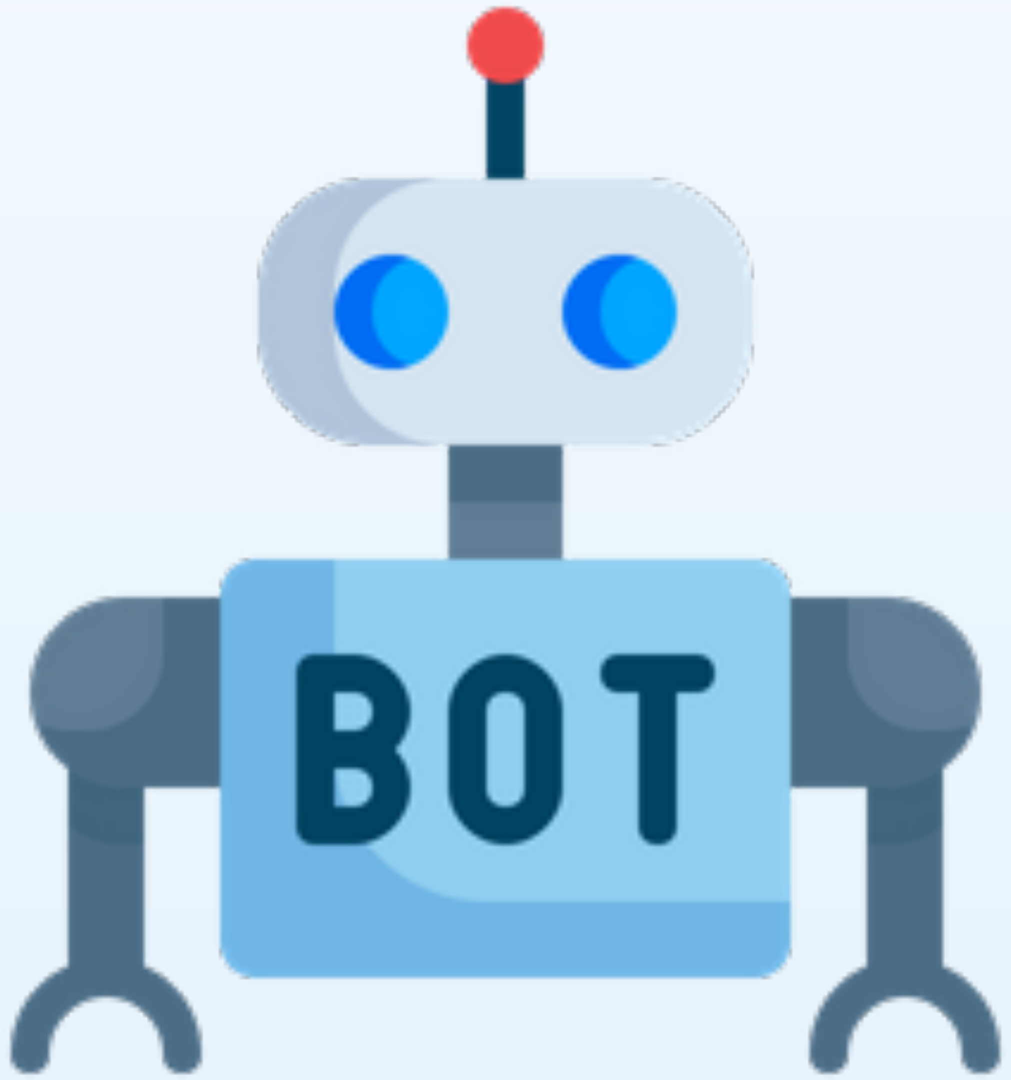


- State (only one) : {both levers unpulled}
- Environment : {Two levers that each output rewards}
- Actions : {pull lever 1, pull lever 2}

# K-armed Bandit Problem

A decision must be made at each time to try and maximize expected gain

AGENT



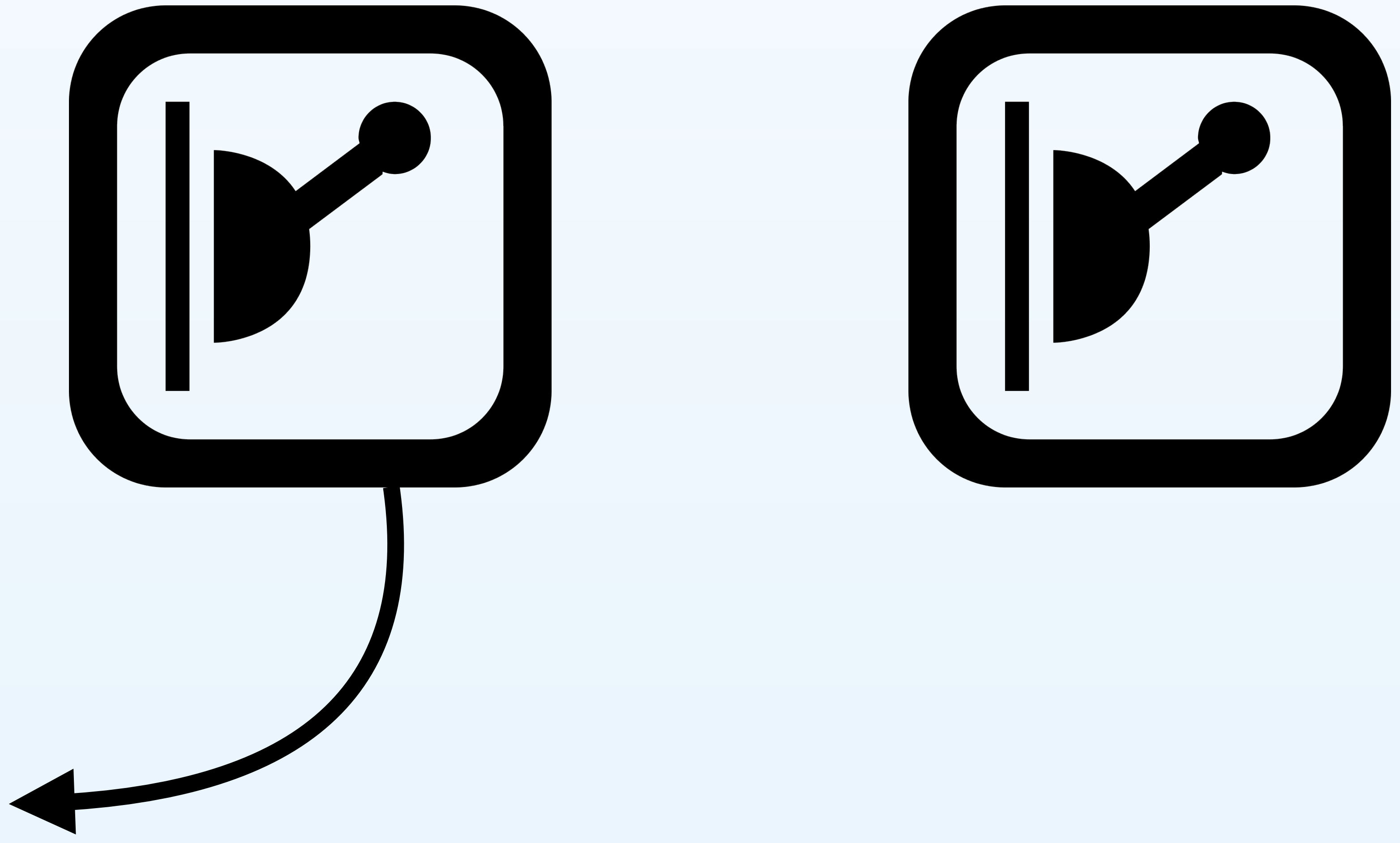

$$a \in \{1,2\}$$

The Agent can choose to pull lever 1 or 2

$$a_t = 1$$

Lever 1

Lever 2



$$R_t = 20$$

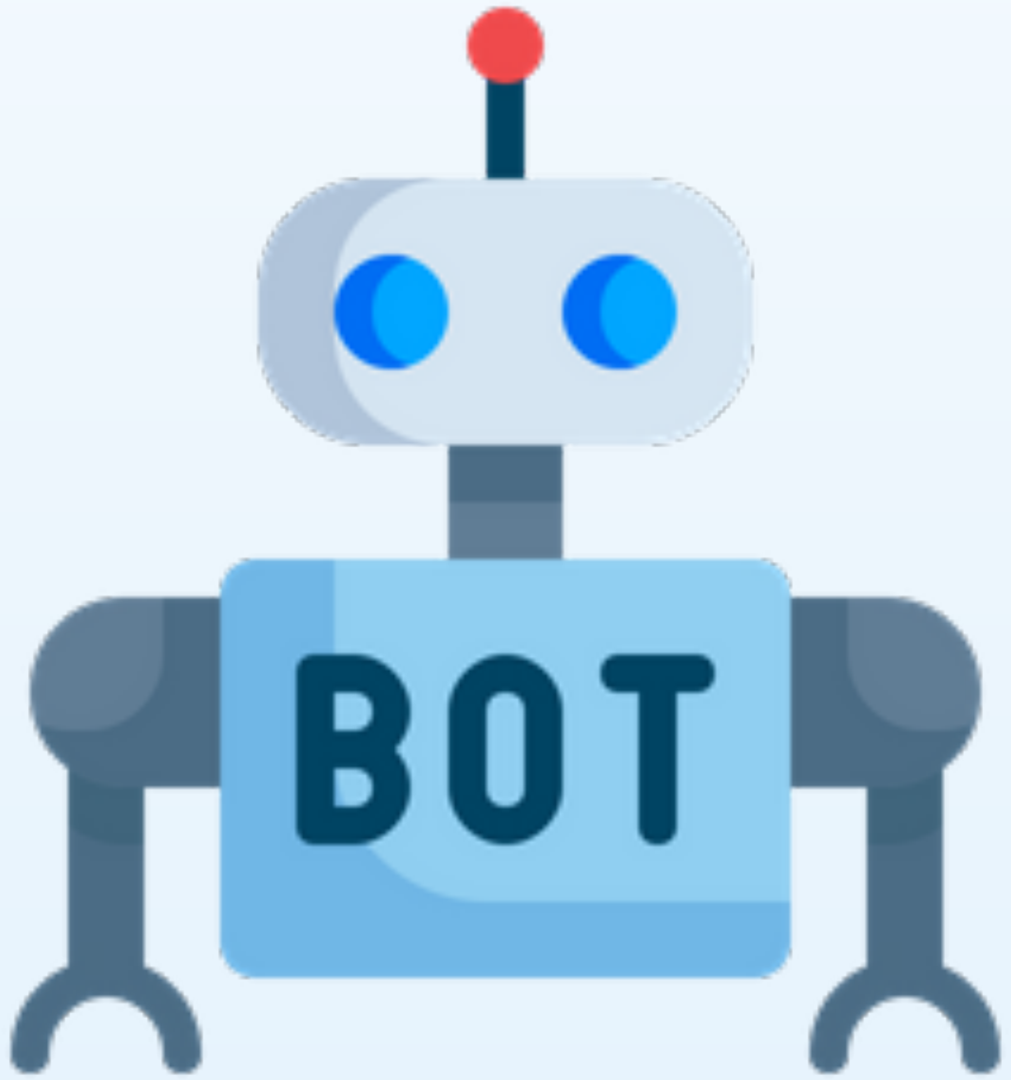
# K-armed Bandit Problem

## Action Value function

Lever 1 Lever 2

- Based on Historical Actions/ reward there is an expected return for each action

AGENT




$$a \in \{1,2\}$$

$$R_t = 20$$
$$R_{t+1} = 10$$
$$R_{t+2} = 30$$

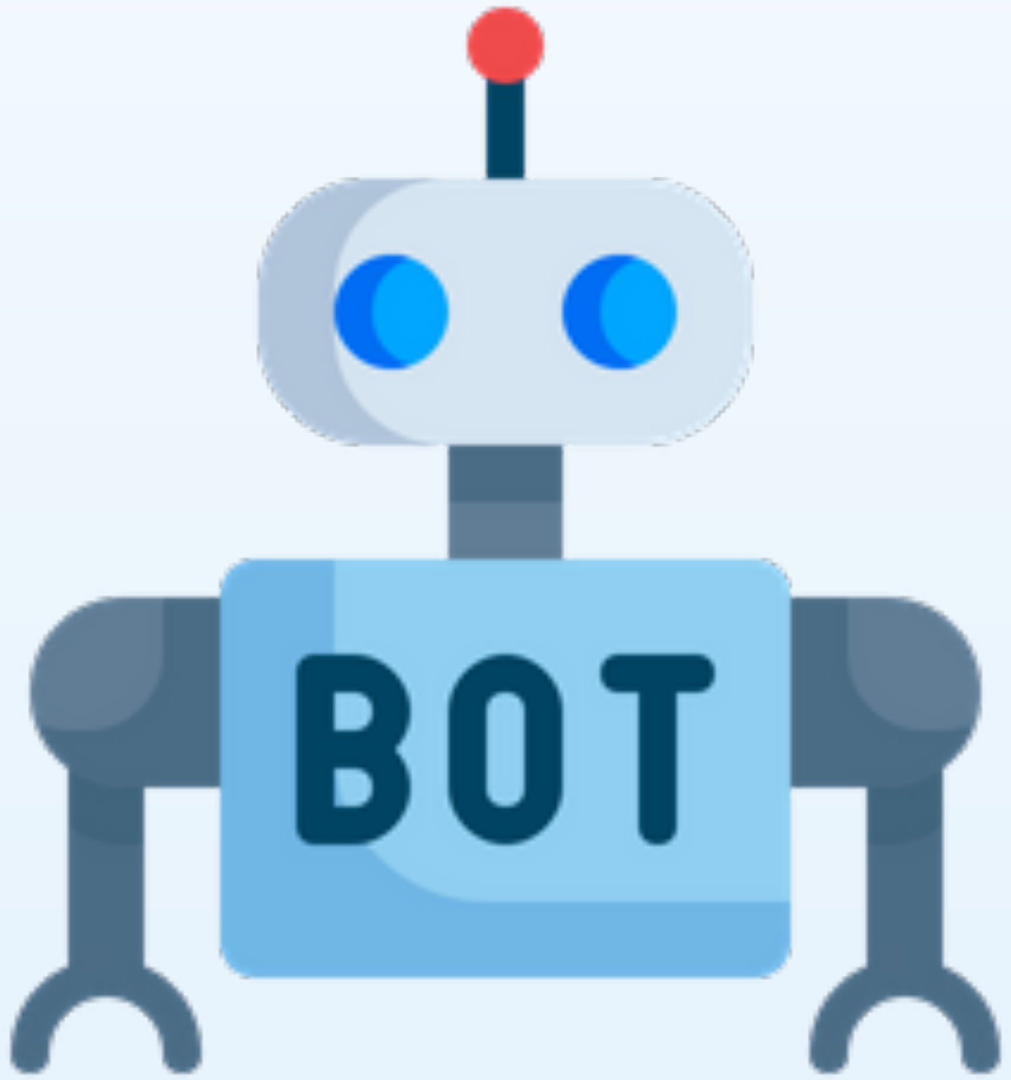
$$q_*(a) \doteq \mathbb{E}[R_t \mid A_t = a] \quad \forall a \in \{1, \dots, k\}$$

# K-armed Bandit Problem

## Action Value function

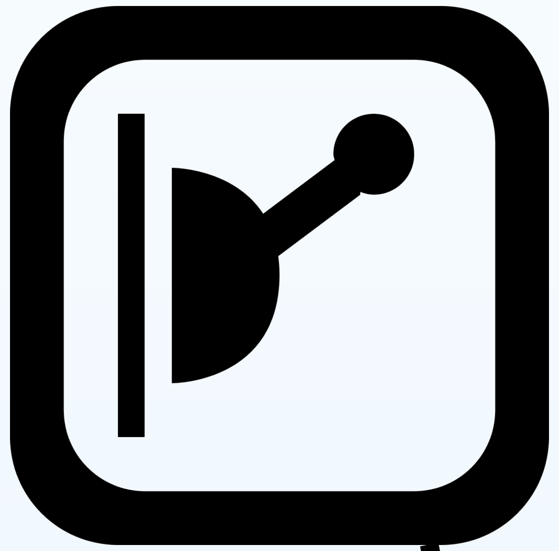
- Based on Historical Actions/ reward there is an expected return for each action

AGENT

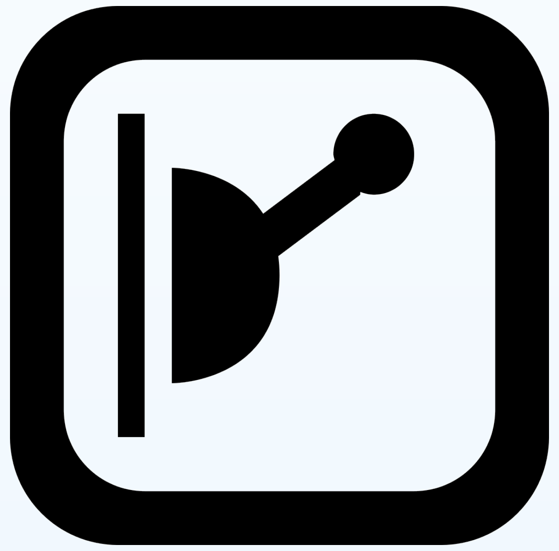



$$a \in \{1,2\}$$

Lever 1



Lever 2



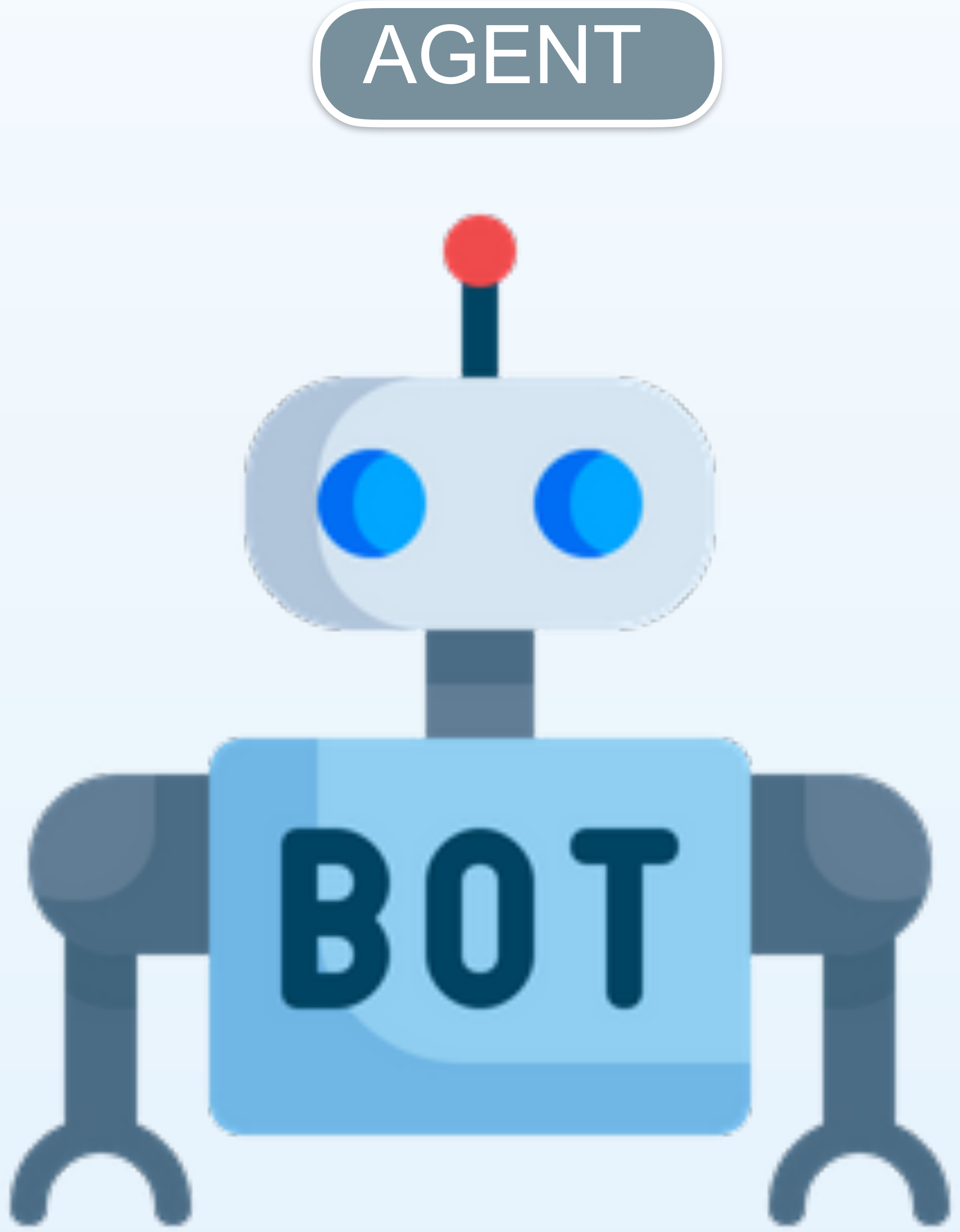
$$R_t = 20$$
$$R_{t+1} = 10$$
$$R_{t+2} = 30$$

$$Q_t(a) \doteq \frac{\text{sum of rewards when } a \text{ taken prior to } t}{\text{number of times } a \text{ taken prior to } t}$$

$$= \frac{\sum_{i=1}^{t-1} R_i}{t-1}$$

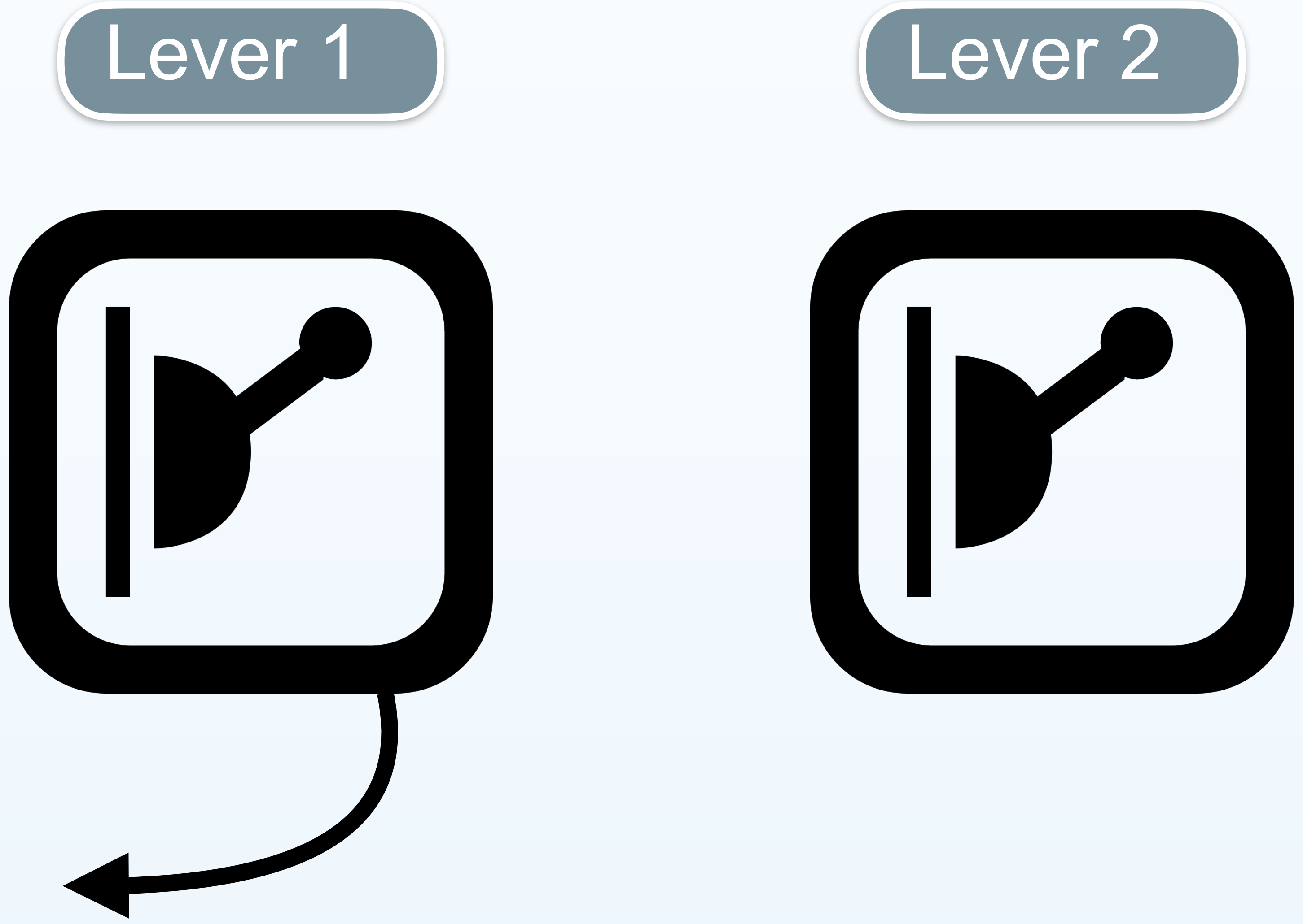
# K-armed Bandit Problem

## Action Value function




- Can only approximate q*
- So we use Q as an estimate using the "Sample Average Update Rule"

$$a \in \{1,2\}$$




$$Q_t(a) \doteq \frac{\text{sum of rewards when } a \text{ taken prior to } t}{\text{number of times } a \text{ taken prior to } t}$$

$$= \frac{\sum_{i=1}^{t-1} R_i}{t-1}$$

# K-armed Bandit Problem

## Incremental Update Rule

- Only have to store 2 values

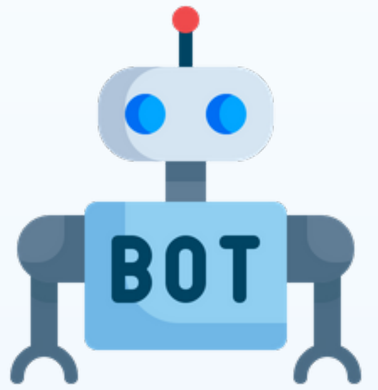
- More efficient moving forward

$$Q_{t+1}(a) = Q_t(a) + \alpha_t\Big(R_t - Q_t(a)\Big)$$

$$NewEstimate \leftarrow OldEstimate + StepSize[Target - OldEstimate]$$

# K-armed Bandit Problem

## Exploitation vs. Exploration

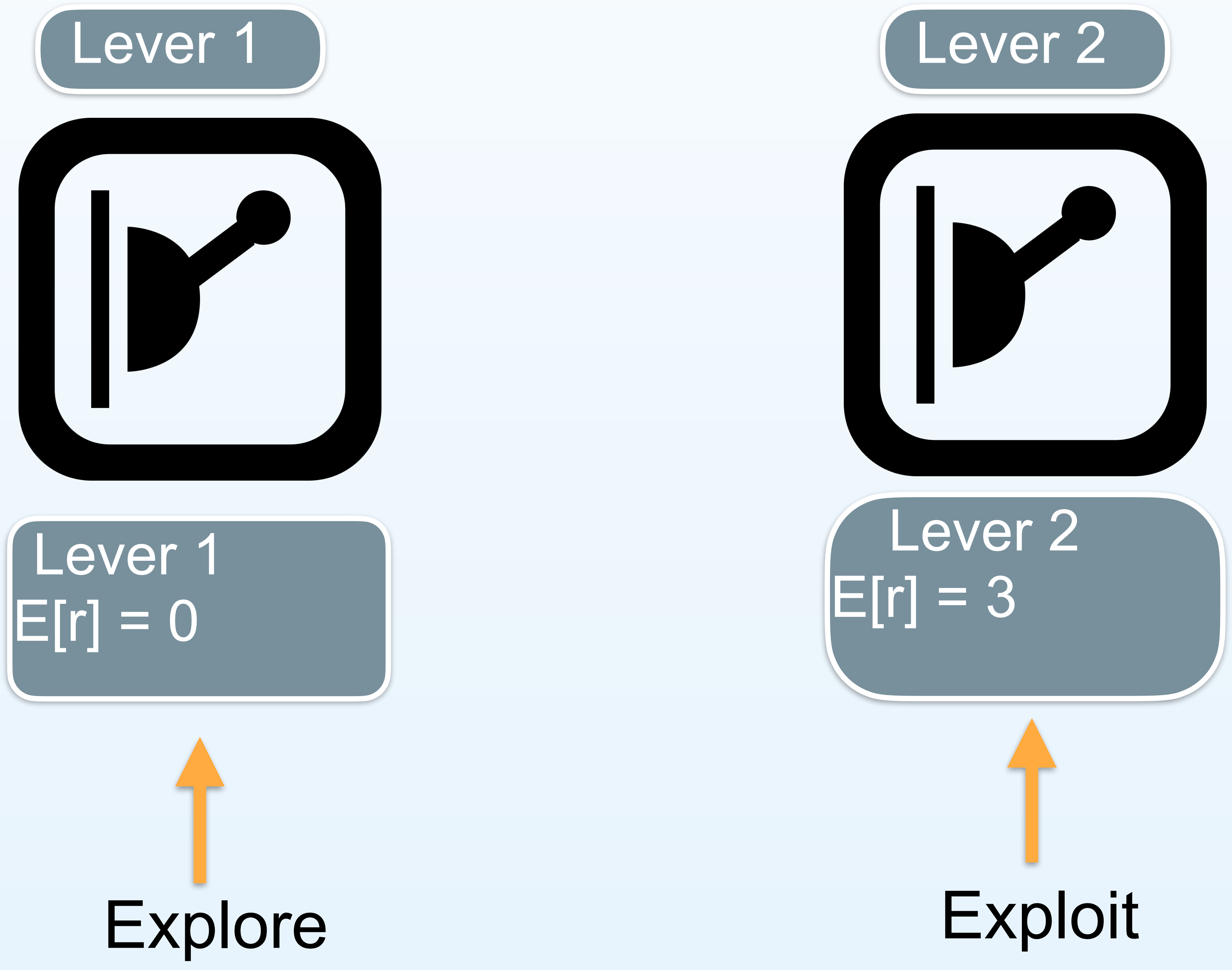



AGENT

$$a \in \{1,2\}$$

- If your policy is always choose, *a*, that gives the highest *Q(a)*  This is called exploitation
- This is the "Greedy" Policy

$$\underset{a}{\operatorname{argmax}}\ Q_t(a)$$

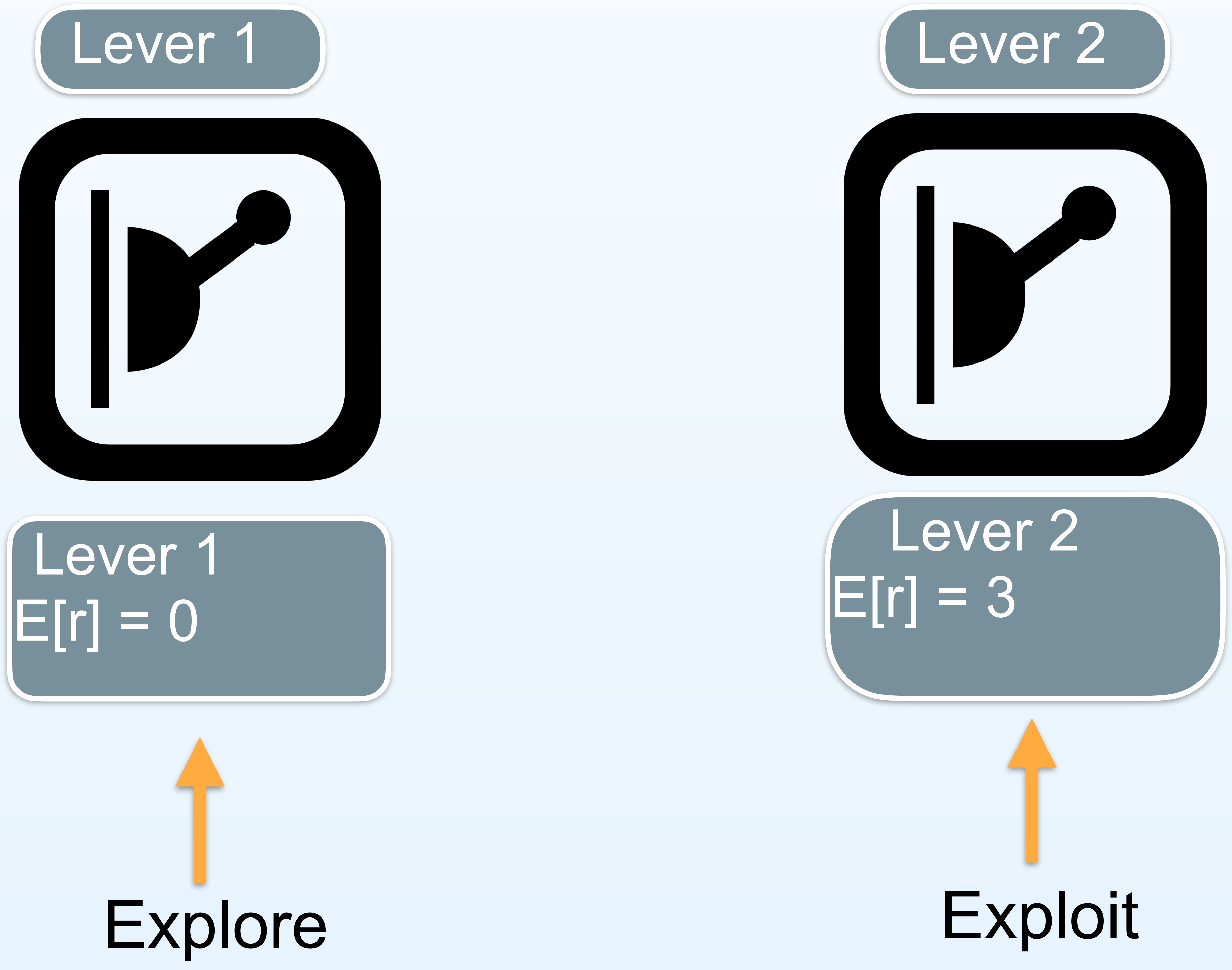


# K-armed Bandit Problem

Exploitation vs. Exploration

- Exploration - improve knowledge for long-term benefit

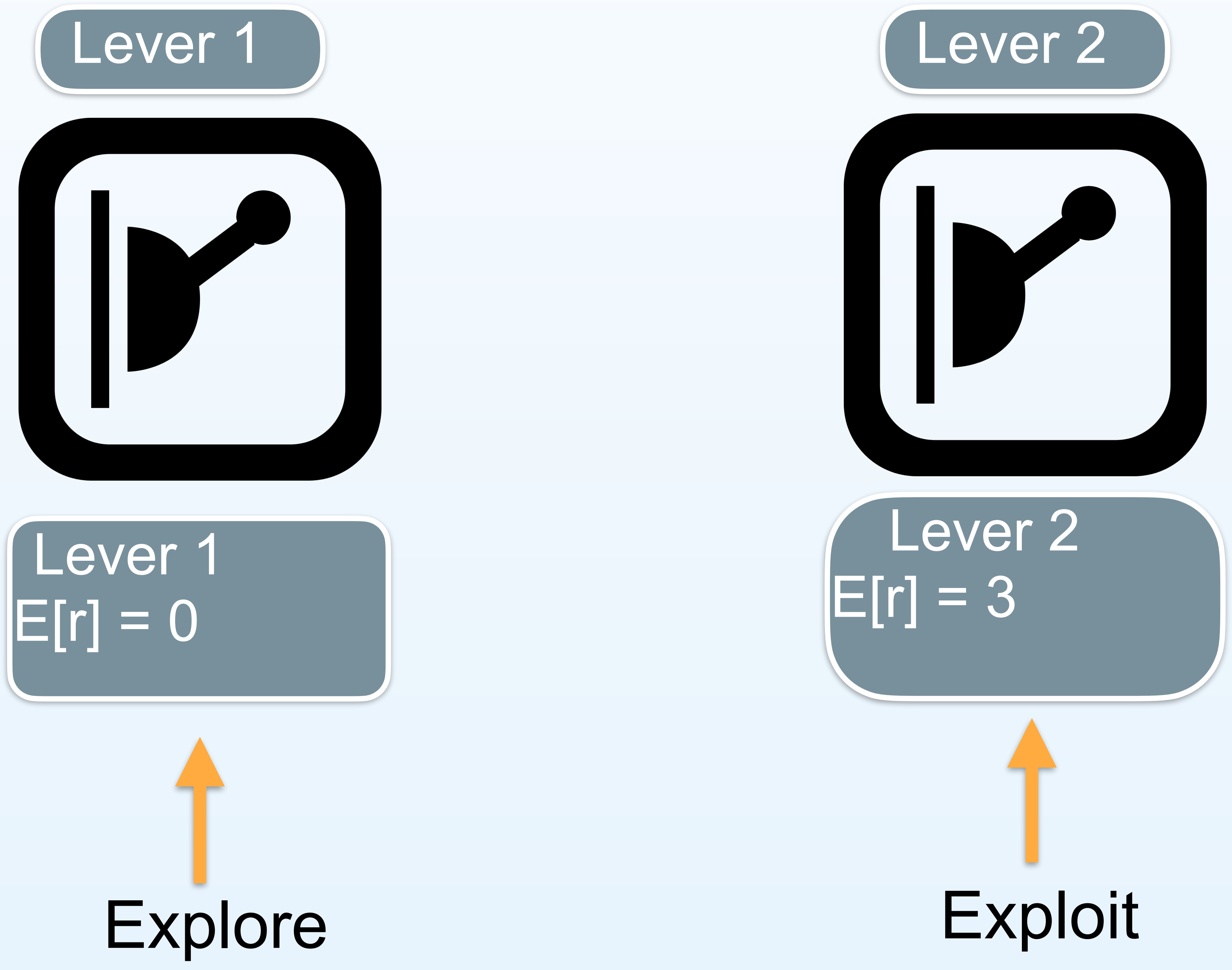
- Exploitation - exploit knowledge for short-term benefit



# K-armed Bandit Problem

## Exploitation vs. Exploration

- Exploration - improve knowledge for long-term benefit
- Exploitation - exploit knowledge for short-term benefit



# K-Armed Bandit Problem

Average reward per action

## Epsilon-greedy Policy

- What is the optimal epsilon for a 10 arm bandit problem

# Optimistic Initial Values

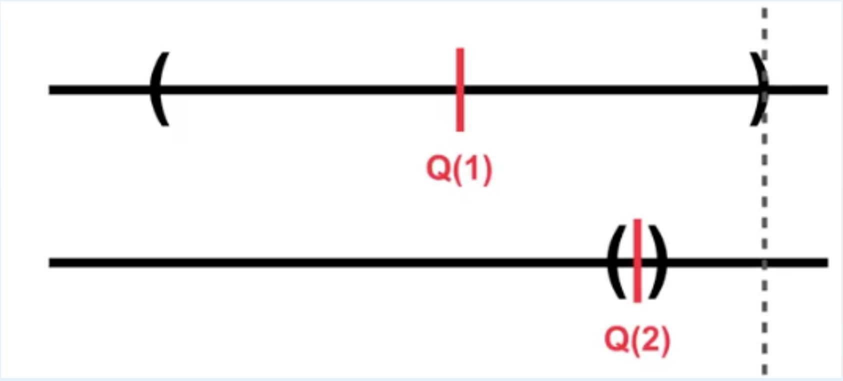
Set every initial Expected value to something very high

- Naturally encourage exploration that will fade over time

# Upper Confidence Bound (UCB) Action Selection

Explore actions with high variance to increase confidence of future choice

$$A_t \doteq argmax\left[\underbrace{Q_t(a)}_{\textbf{Exploit}} + \underbrace{c\sqrt{\frac{ln\ t}{N_t(a)}}}_{\textbf{Explore}}\right]$$

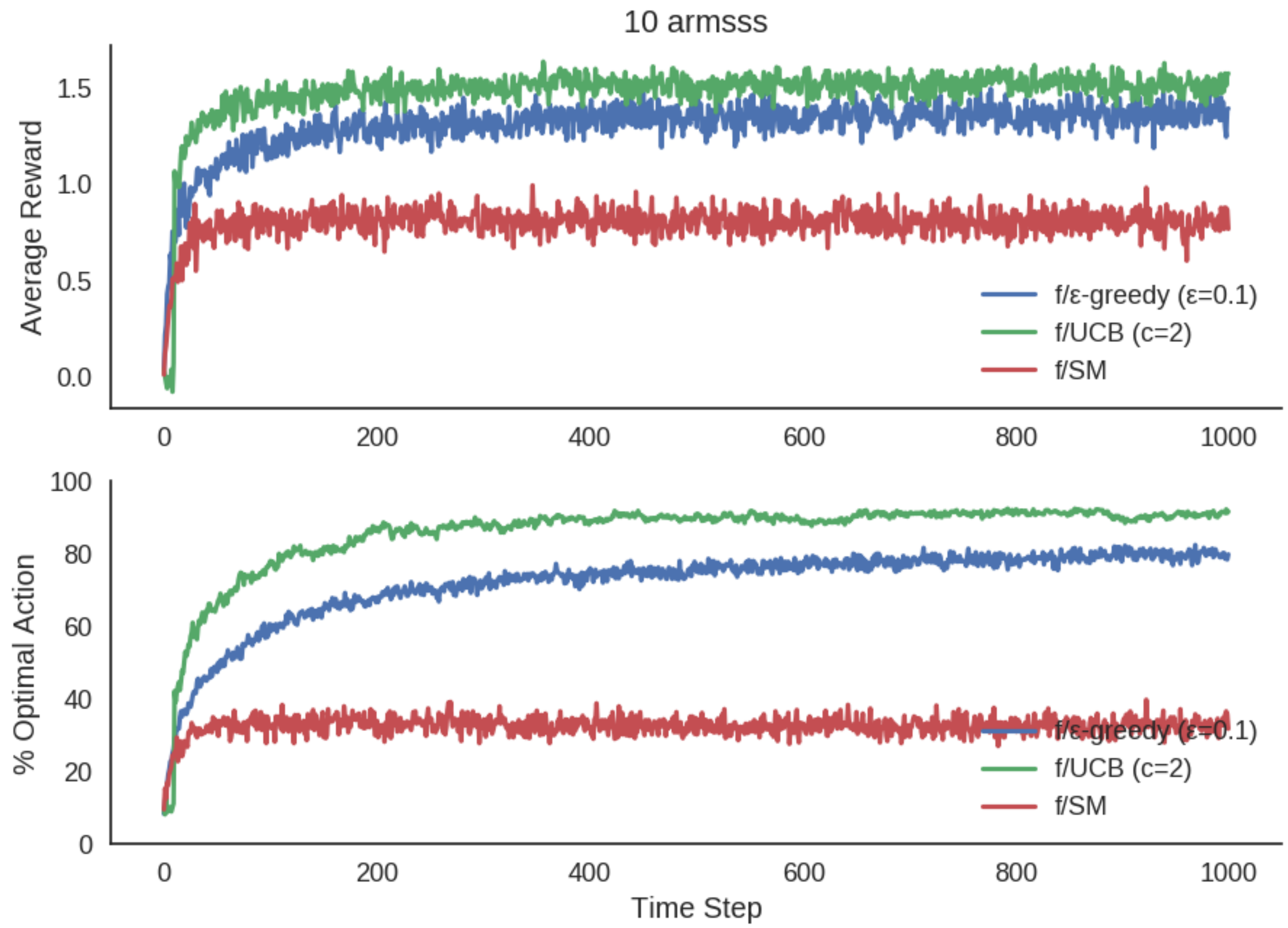



- Will explore actions that have high variance
- More action selections smaller bounds
- Bounds increase over time
- C is a hyperparameter

# Results

## UCB out performs Epsilon greedy algorithm

- UCB outperforms in the short and long run compared to epsilon greedy because it is not weighed down with a fixed need to explore



# Q-Learning

# Introducing multiple states Q-Table

Q-Table is matrix of expected values

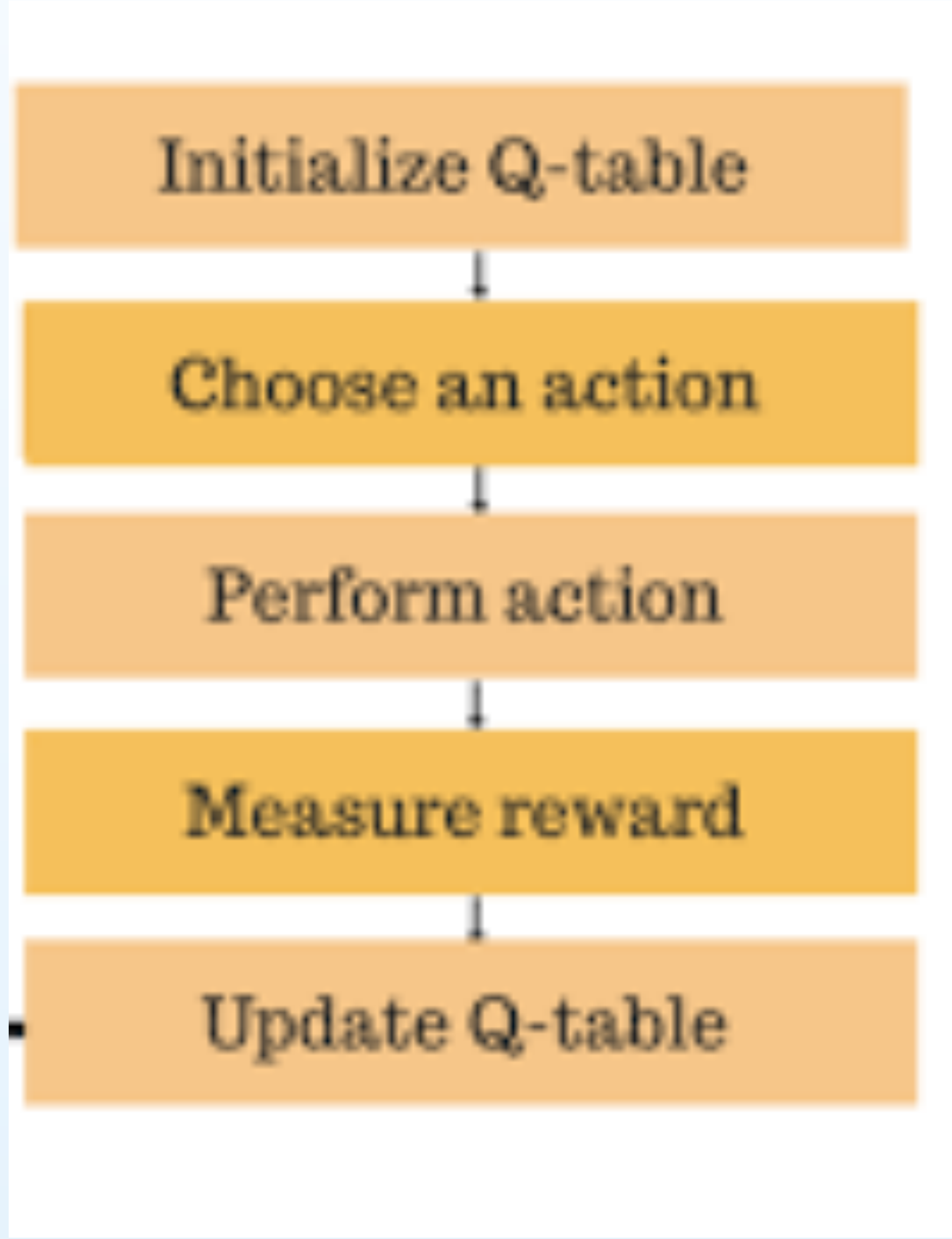
- Let's build a simple financial forecasting state space

| T-1 price change | E[R] a=long | E[R] a=short | E[R] a=nuetral |
|---|---|---|---|
| >1% | 0 | 0 | 0 |
| 1-0% | 0 | 0 | 0 |
| 0- -1% | 0 | 0 | 0 |
| <-1% | 0 | 0 | 0 |

# Introducing multiple states Q-Table

Q-Table is matrix of expected values

- Let’s build a simple financial forecasting state space



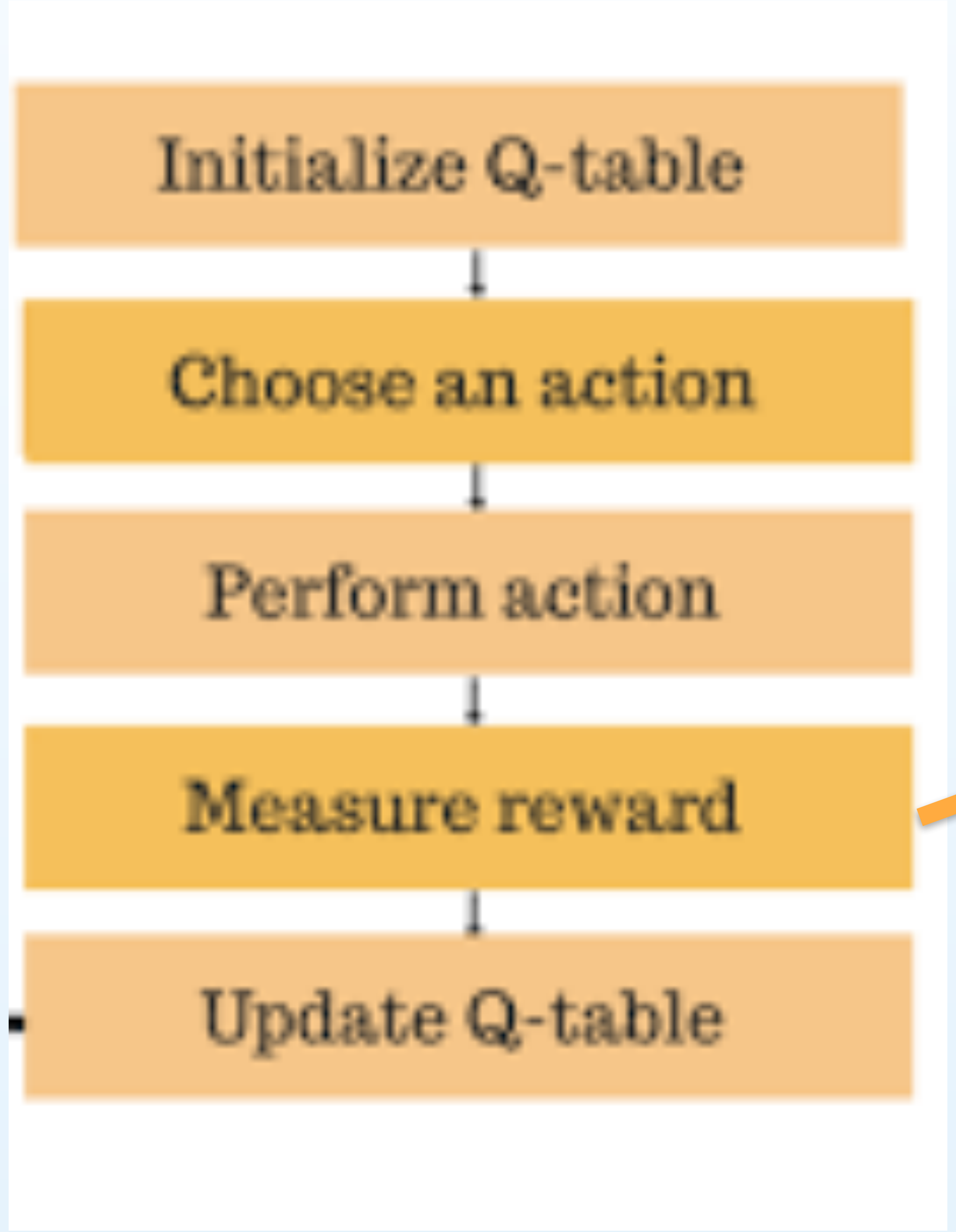

For Financial Forecasting

| T-1 price change | E[R] a=long | E[R] a=short | E[R] a=nuetral |
|---|---|---|---|
| >1% | 0 | 0 | 0 |
| 1-0% | 0 | 0 | 0 |
| 0- -1% | 0 | 0 | 0 |
| <-1% | 0 | 0 | 0 |

# Introducing multiple states Q-Table

## Q-Table is matrix of expected values

- Let's build a simple financial forecasting state space




Based on e-greedy or other algos. Will pick random position

For Financial Forecasting

| T-1 price change | E[R] a=long | E[R] a=short | E[R] a=nuetral |
|---|---|---|---|
| >1% | 0 | 0 | 0 |
| 1-0% | 0 | -0.3% | 0 |
| 0- -1% | 0 | 0 | 0 |
| <-1% | 0 | 0 | 0 |

# Bellman Equation

One equation to update the entire Q-Table

- Has the additional hyper parameter lambda that allows for recent expectations more

$$Q(S_t, A_t) = (1 - \alpha)\, Q(S_t, A_t) + \alpha * (R_t + \lambda * max_a\, Q(S_{t+1}, a))$$

**S** = the State or Observation

**A** = the Action the agent takes

**R** = the Reward from taking an Action

**t** = the time step

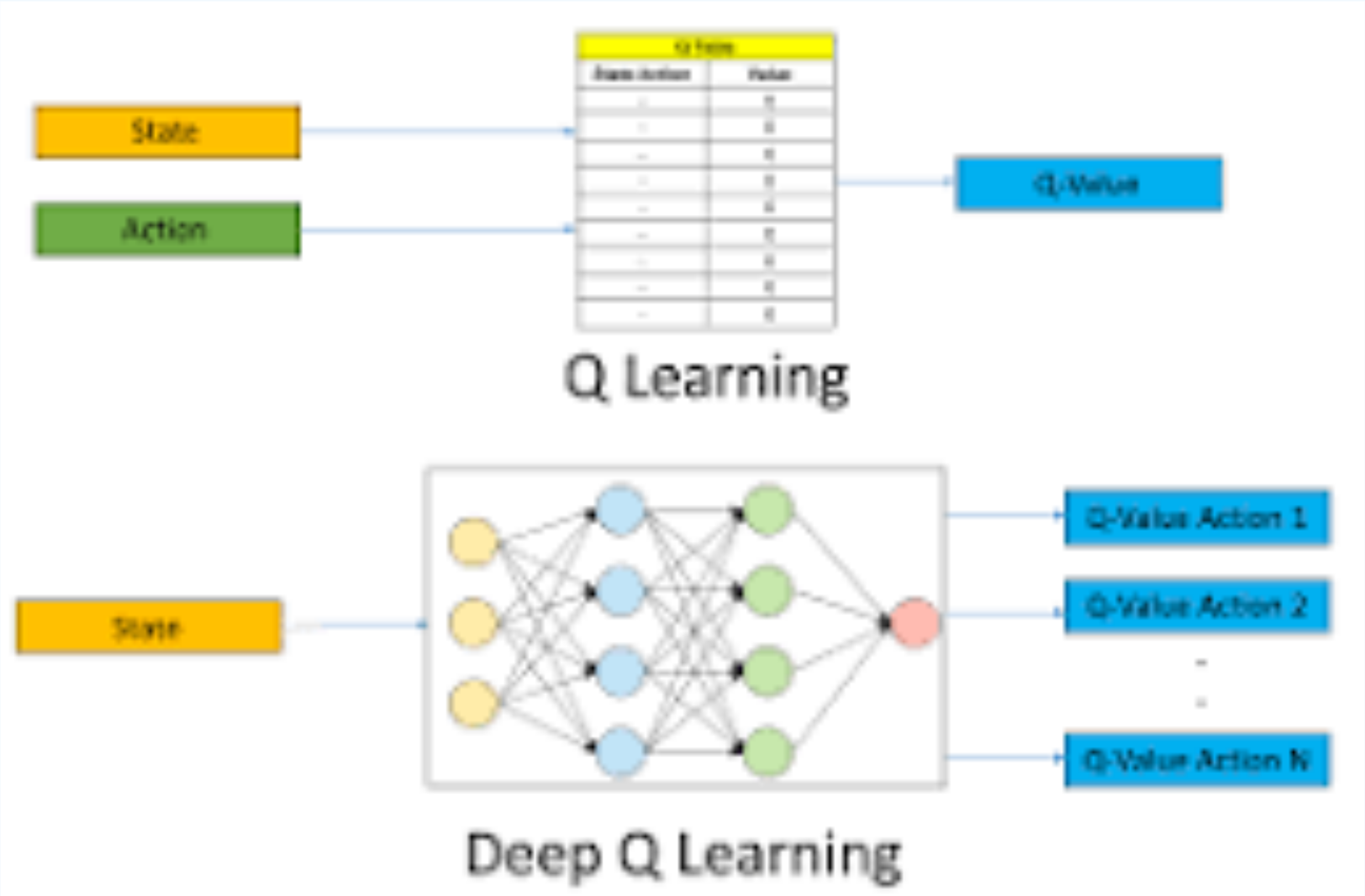
**α** = the Learning Rate

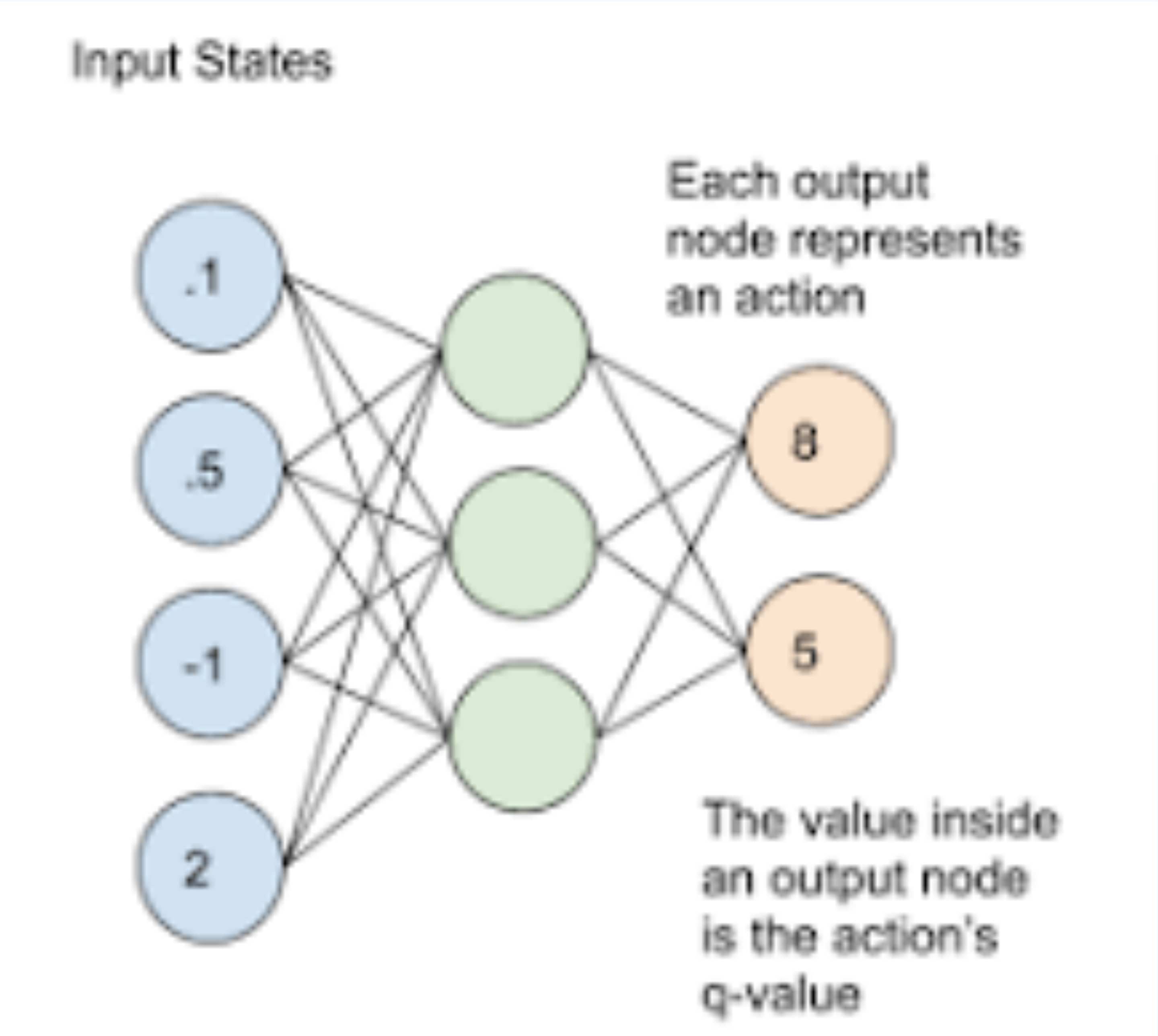
**λ** = the discount factor which causes rewards to lose their value over time so more immediate rewards are valued more highly

# Deep Q-Learning

## Replaces a Q-Table with a Neural Network

- Instead of {state, action} pair mapped to q-value
- Deep Q-learning maps {States} to {action, Q-values}

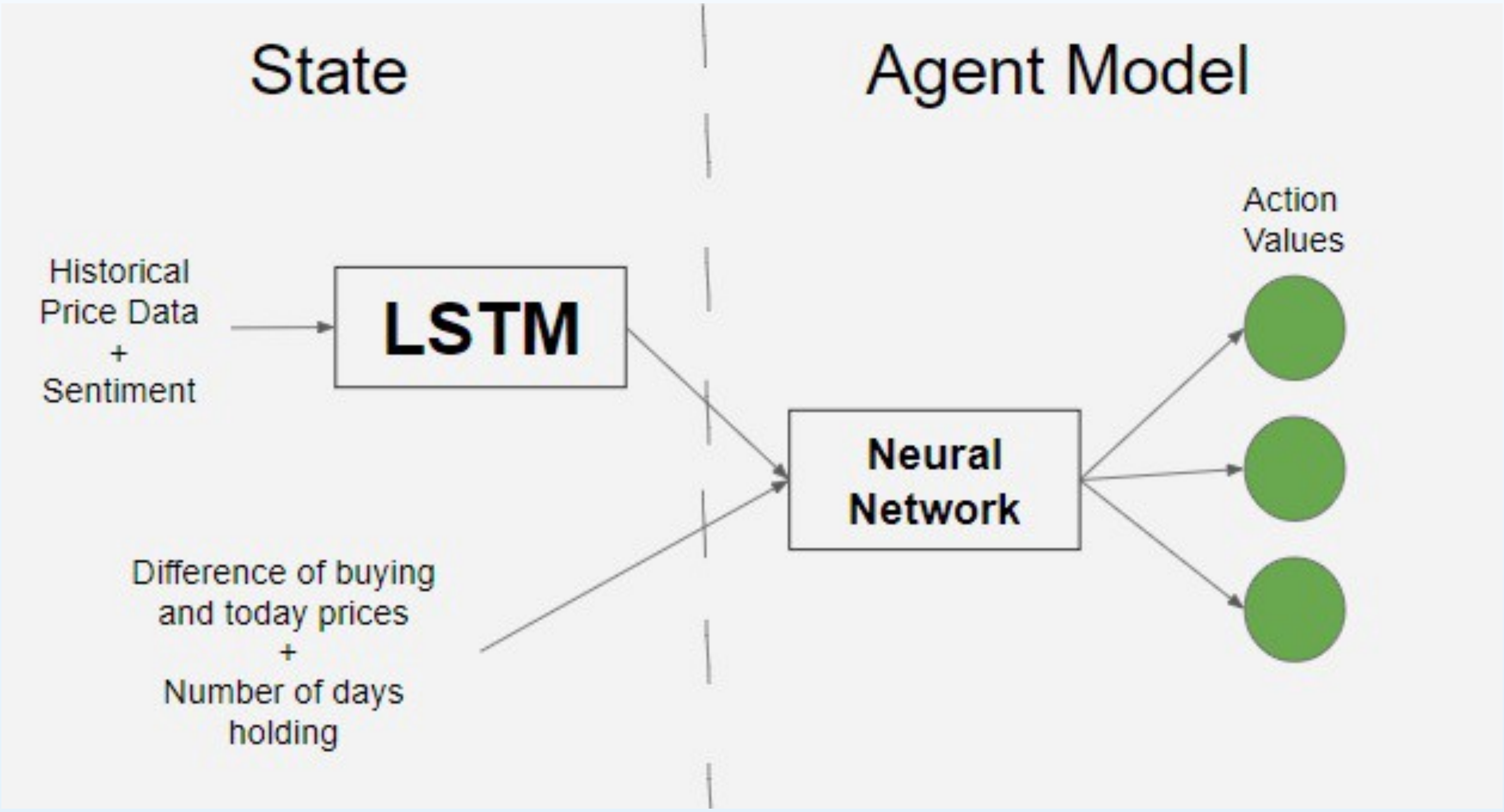


# Deep Q-Learning

## For Sequential Data

- Utilizing the LSTM architecture allows for the agent to record historical data Fred through



# KerasRL

Has 5 models to choose from

- We wil focus on Deep Q-Learning

- **Deep Q-Learning (DQN)** and its improvements (**Double** and **Dueling**)
- **Deep Deterministic Policy Gradient (DDPG)**
- **Continuous DQN (CDQN** or **NAF)**
- **Cross-Entropy Method (CEM)**
- **Deep SARSA**

# Epochs

## Training your network

- Run through your training data multiple times to fit better (100 Epochs is standard)

```
epoch: 10, total rewards: 444.790016.3,
epoch: 20, total rewards: 419.160018.3,
epoch: 30, total rewards: 414.375014.3,
epoch: 40, total rewards: 413.445012.3,
epoch: 50, total rewards: 435.830012.3,
epoch: 60, total rewards: 422.640012.3,
epoch: 70, total rewards: 463.735011.3,
epoch: 80, total rewards: 470.200010.3,
```