2.	**Multi-arm Bandits** An n-Armed Bandit Problem, Action-Value Methods, Incremental Implementation, Tracking a Non-stationary Problem, Optimistic Initial Values, Upper-Confidence-Bound

- ✓ Multi-Armed Bandit (MAB) problem
- ✓ Exploitation vs Exploration
- ✓ ε-greedy algorithm
- ✓ Upper Confidence Bounds (UCB) algorithm

Aim of this module

- ✓ Understand concepts of Multi-Armed Bandit problem as basic problem for Reinforcement Learning (RL) and discuss multiple approaches to deal with the problem.

Multi-Armed Bandit (MAB)

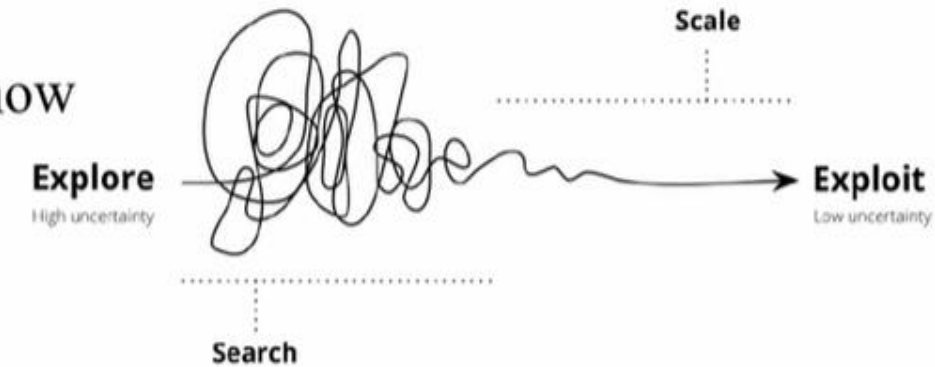
Exploitation vs Exploration:

Exploration: Try new things all the time

Take some risk to collect information about unknown options

Exploitation: Use previous experience

Take advantage of the best option we know



The best long-term strategy may involve short-term sacrifices

Exploration

- Find out more about the Environment
- Give up some immediate Rewards for maximizing Future Rewards

Exploitation

- Exploit known or Profitable Information about the Environment to maximize immediate Rewards.

Exploration and Exploitation

- Exploration and Exploitation is a fundamental challenge that agents face in their quest for optimal decision-making.
- Exploration involves the agent trying out new actions to gather information about their effects, potentially leading to the discovery of more rewarding strategies.
- On the other hand, exploitation entails choosing actions that are deemed to be the best based on current knowledge to maximize immediate rewards.
- Striking the right balance is crucial; too much exploration can impede progress, while excessive exploitation may lead to suboptimal long-term outcomes.
- **To obtained a lot of reward, a RL agent must prefer actions that it has tried in the past and found to be effective in producing reward.**
- **But to discover such actions , it has try actions that it has not selected before that is called exploration.**

Exploration-Exploitation Tradeoff

- RL Agent's objective is to find a Policy which maximizes Future Return.

- It is important to keep a balance between Exploration and Exploitation.

Examples

- **Advertising:**

Exploitation: Showing same profitable ads repeatedly

Exploration: Showing new ads which may be more profitable

- Restaurant:

Exploitation: Going to same favorite restaurant

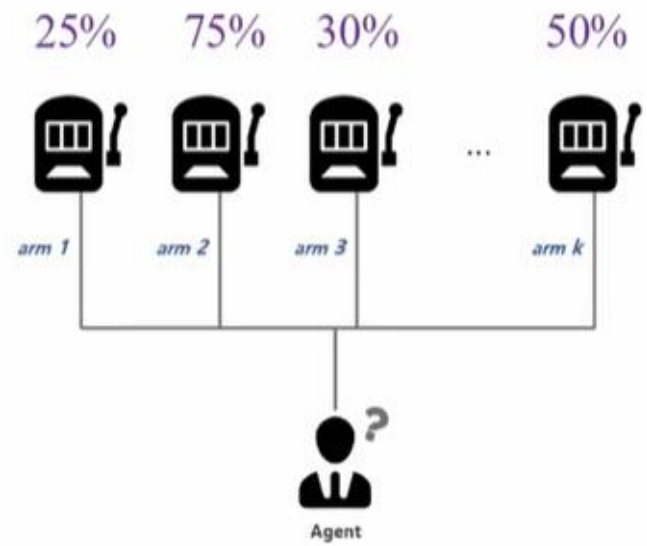
Exploration: Trying out new restaurants

Multi-Armed Bandit (MAB)

- ✓ In a **casino facing multiple slot machines** and each is configured with an **unknown probability** of how likely you can **win or lose** at one play

What is the best strategy to achieve highest long-term rewards

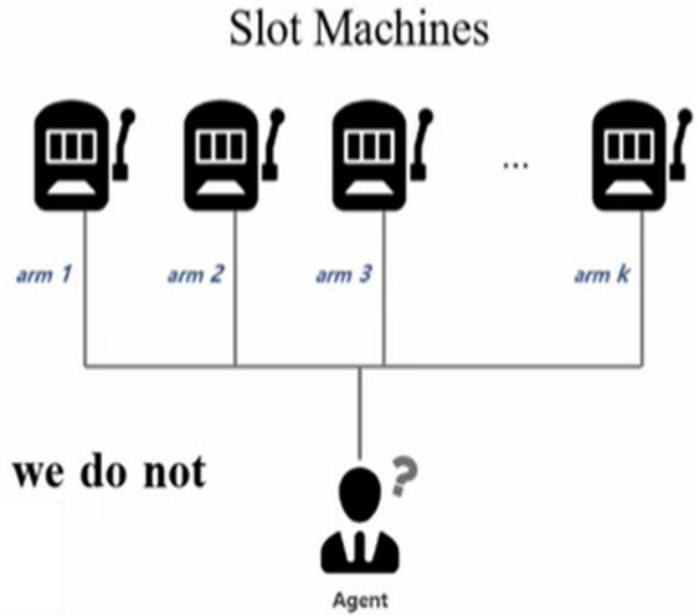
- ✓ Considering **infinite number of trials** (to understand concept)



Multi-Armed Bandit (MAB)

Definition:

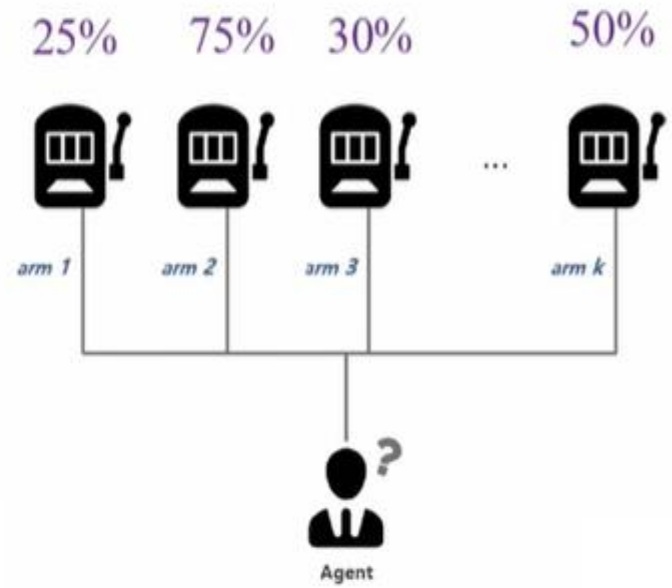
- ✓ A classic problem that demonstrates the exploration vs exploitation and basic concept of RL
- ✓ It is a **one state problem: (why?)**
  - ➢ Because rewards has no effect on the future rewards and **reward distributions are stationary**
- ✓ The goal is to **maximize the cumulative reward** by doing actions (arms do the actins)



Note: It is a simple version of Markov Decision Process (MDP) **since we do not have states** (only one state)

The most basic Approach

- ✓ Playing with **one machine for many rounds** to get reward is simplest approach
- ✓ This **does not guarantees the best long-term reward**
- ✓ Also based on limited set of knowledge and resources we need to have **smart approach!**



Definition:

- ✓ Considered as a tuple of (a, r)
- ✓ Take an action a and receive reward r (on one slot machine).
- ✓ K machines with reward probabilities of $\{P_1, P_2, \dots, P_K\}$
- ✓ The value of action a is the expected reward $V(a) = \mathbb{E}[r|a] = P$
- ✓ At the time step t the reward function $r_t = R(a_t)$ based on the probability

Example:

✓The goal is to maximize the cumulative reward:

$$max \sum_{t=1}^{T} r_t$$

✓ If **we know** the optimal action and best reward the goal can be **minimizing** the **loss function**:

$$\mathcal{L}_T = \mathbb{E} \sum_{t=1}^{T} \left(P^* - Q(a_t)\right)$$

Optimal reward probability

$$P^* = Q(a^*) = \max_{1toK}(P_i)$$

The regret we might have by not selecting the optimal action up to the time step T

Value of taking action a in time t

Multi-Armed Bandit Strategies

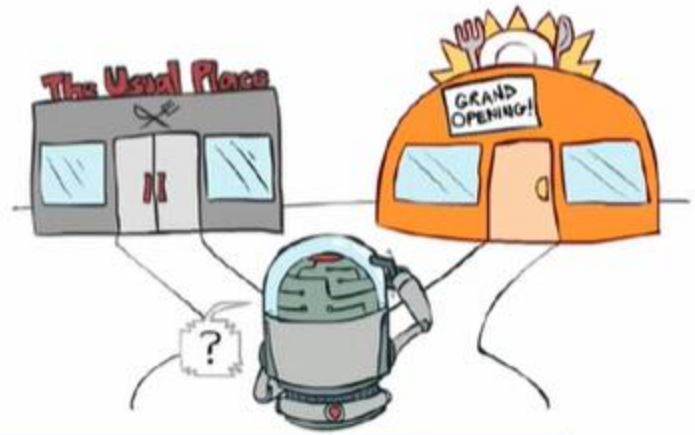
1) Playing with **one machine for many rounds**
   - The most bad and naive approach (no exploration)
2) Exploration randomly
   - Not reliable
3) Exploration preference to uncertainty (**wise exploration**)

ε-greedy policy concept:

- ✓ The ε-greedy algorithm takes the **best action** or choose a **random action** exploration occasionally.

How?

a) Generate a random number (*rand* $\in [0, 1]$)
b) If *rand* $< \varepsilon$ choose a greedy action (**exploitation**)
c) Otherwise choose a random action (**exploration**)



To avoid inefficient exploration, we can decrease the parameter ε in time (we call it epsilon decay)

Multi-Armed Bandit by ε-greedy policy:

✓ **Action value is estimated using the past experience:**

- By averaging the rewards of action that we have observed up to the current time step t:

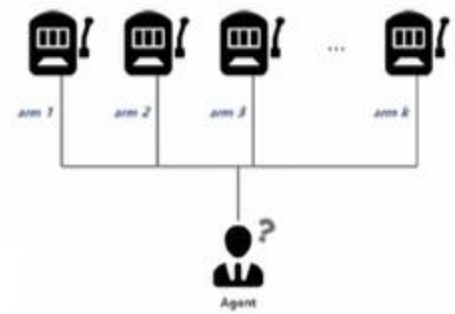
$$\hat{Q}_t(a) = 1/N_t(a) \sum_{t=1}^{T} r_t \, \mathbb{1}[a = a_t]$$

How many times the action a has been selected before time t

Now we pick the best action that we have learnt by choosing an action:

$$\hat{a}_t^* = argmax_{a \in A} \hat{Q}_t(a)$$

Best estimated action in time t



Challenge of ε-greedy policy:

- ✓ If we can tune ε parameter well we will have good result.
- ✓ **Defining proper value** of the **ε** sometimes **is challenging!**

Solution

- ✓ Having an **approach** that **relies on number times that we applied an action**.

Upper Confidence Bounds (UCB)

Upper Confidence Bounds (UCB) algorithm

- ✓ **Favor exploration** of **actions** with a **strong potential** to have a optimal value!
- ✓ We have a **bound per each action** that indicates **how confident** we are about the **Q-value of that action.**
  - ✓ Less confident = Bound will be **big**
  - ✓ More confident = Bound will be **small**

square root

$$U_t(a) = \sqrt{\frac{2 \log t}{N_t(a)}}, a \in A$$

Total number of choosing actions (t)

Total number of times that specific action (a_i) has been selected

✓ Larger $N_t(a_i)$ gives us smaller $U_t(a_i)$

Upper Confidence Bounds (UCB) algorithm

- ✓ In fact, we prefer actions that we haven't had a confident value estimation for it yet.
- ✓ UCB algorithm measures this potential by an **Upper Confidence Bound** of the reward value then add to Q-value
- ✓ So, it **select the greediest action** to maximize the upper confidence bound as follows:

$$a_t^{UCB} = argmax_{a \in A}\, Q_t(a) + U_t(a)$$

Upper Confidence Bounds (UCB) algorithm

$$a_t^{UCB} = argmax_{a \in A}\, Q_t(a) + U_t(a)$$

Example

Assumptions:

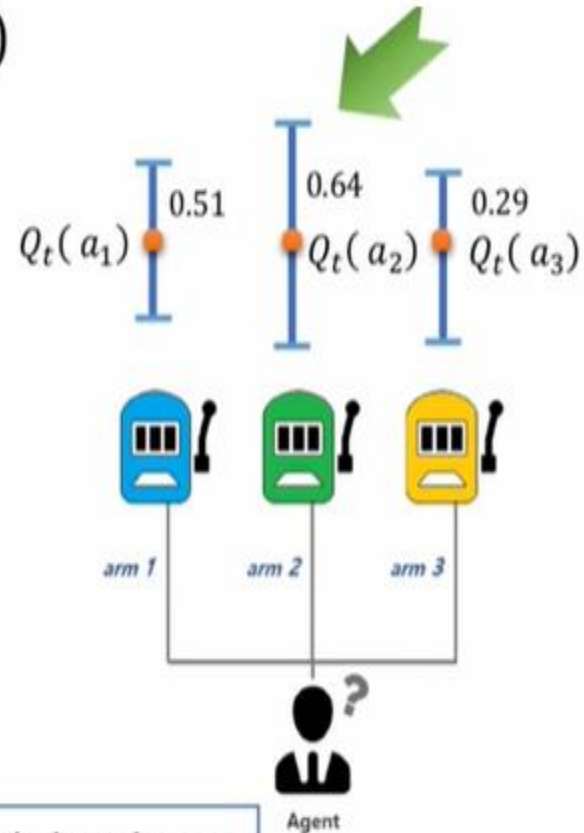
$$U_t(a) = \sqrt{\frac{2 \log t}{N_t(a)}}, a \in A$$

$Q_{1000}(a_1) = 1$	$N_{1000}(a_1) = 200$
$Q_{1000}(a_2) = 1$	$N_{1000}(a_2) = 150$
$Q_{1000}(a_3) = 1$	$N_{1000}(a_3) = 750$

$$U_{1000}(a_1) = \sqrt{\frac{2 \log 1000}{N_t(a_1)}} = \sqrt{\frac{63.25}{200}} = 0.51$$

$$U_{1000}(a_2) = \sqrt{\frac{2 \log 1000}{N_t(a_1)}} = \sqrt{\frac{63.25}{150}} = \mathbf{0.64}$$

$$U_{1000}(a_2) = \sqrt{\frac{2 \log 1000}{N_t(a_1)}} = \sqrt{\frac{63.25}{750}} = 0.29$$



In machine two we have higher hope (not very sure about its Q-value)

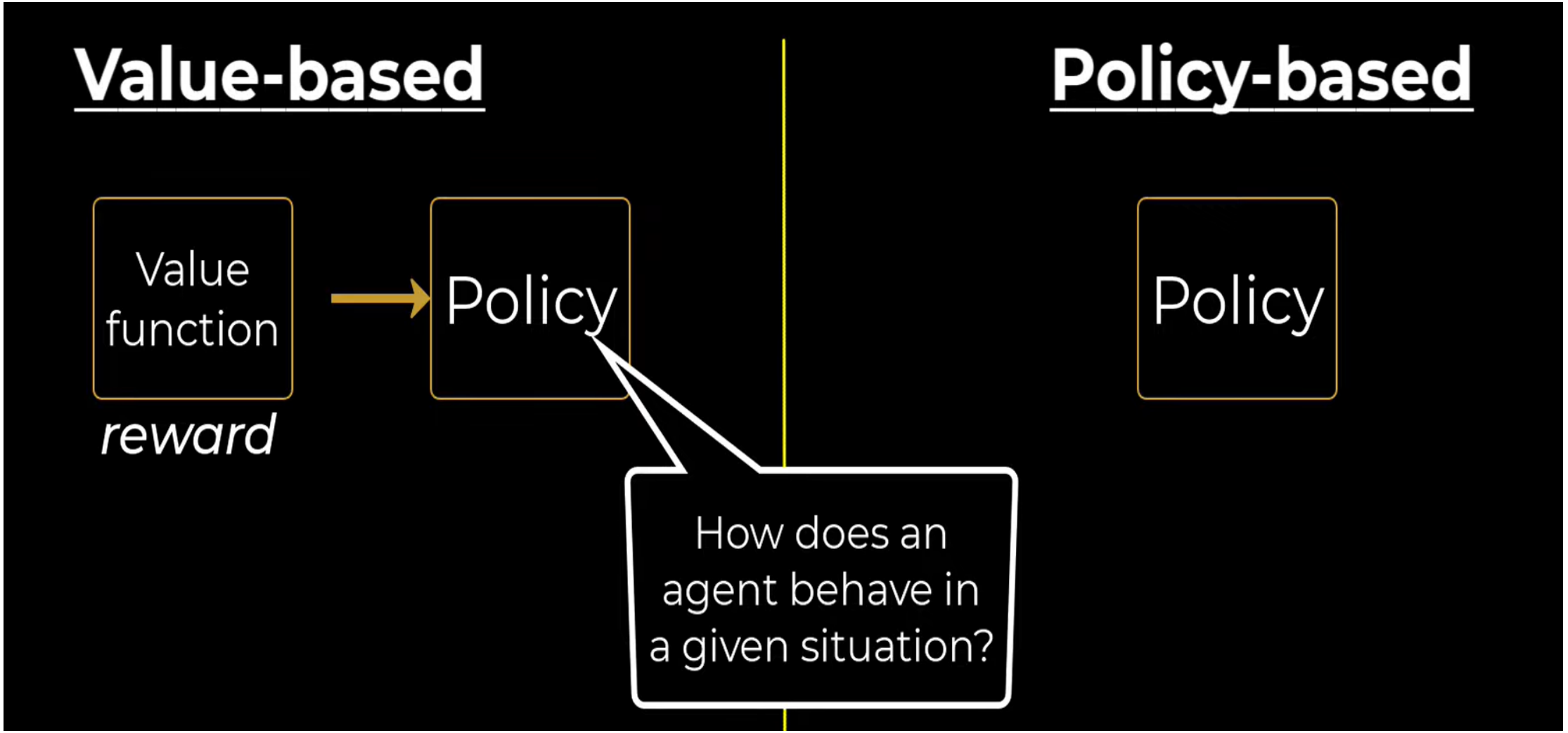
- RL Algorithms can be sub-categorized as **value-based methods** and **policy-based methods**.

- Value based methods determine a **value function** that quantifies total reward. Using this value function, we determine the **optimal policy**.

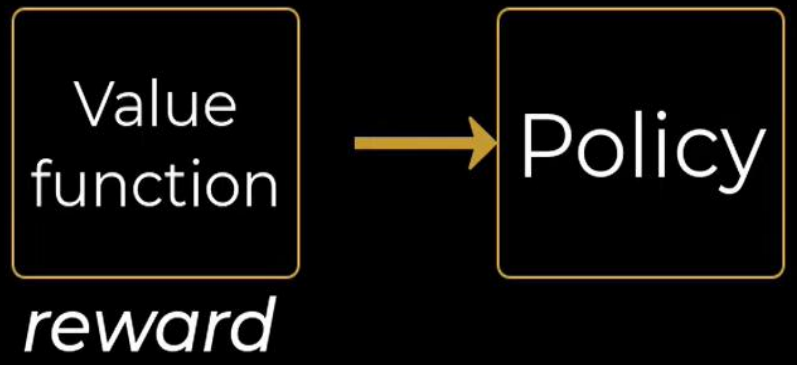
- Value function can be a **state value function** v(s) or **state-action value function** q(s, a).

- **Bellman equation** is a recursive strategy to calculate the value function.

- It is used by some value-based methods like q-learning.


Value-based
Policy-based
Value function
reward
Policy
Policy
How does an agent behave in a given situation?

Value-based



Q-Learning
Deep Q-Learning

SARSA

state-action-reward-state-action

Policy-based

REINFORCE

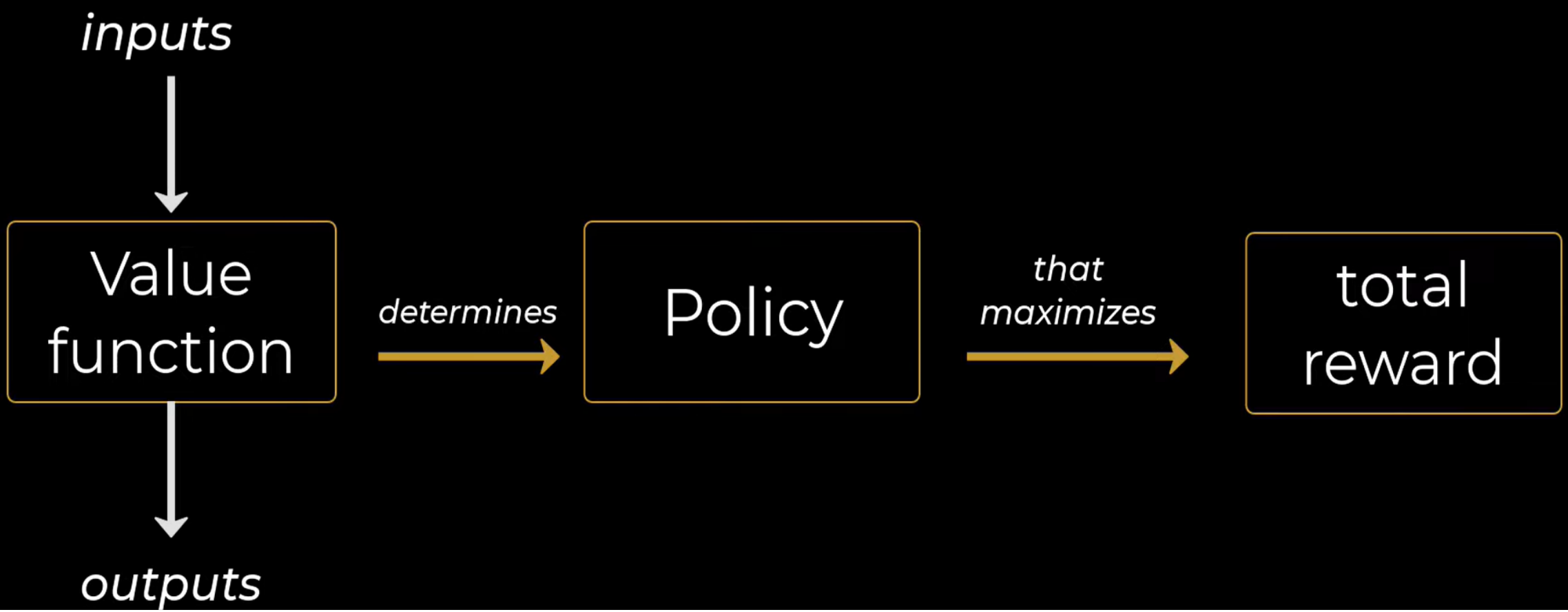
PPO

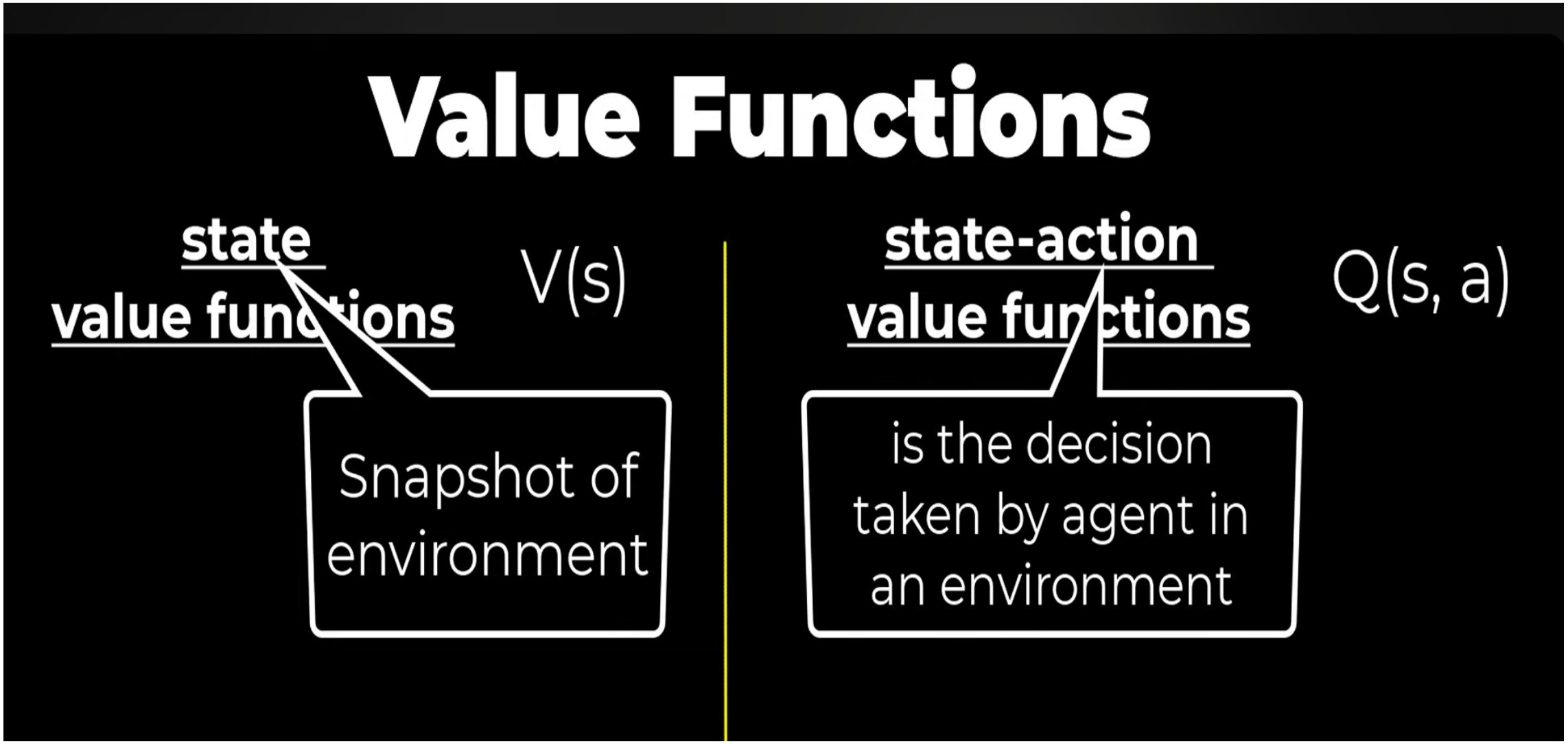
Proximal Policy Optimization

TRPO

Trust Region Policy Optimization

Value-based Methods




Value Functions
state value functions
V(s)
Snapshot of environment
state-action value functions
Q(s, a)
is the decision taken by agent in an environment

Value Functions

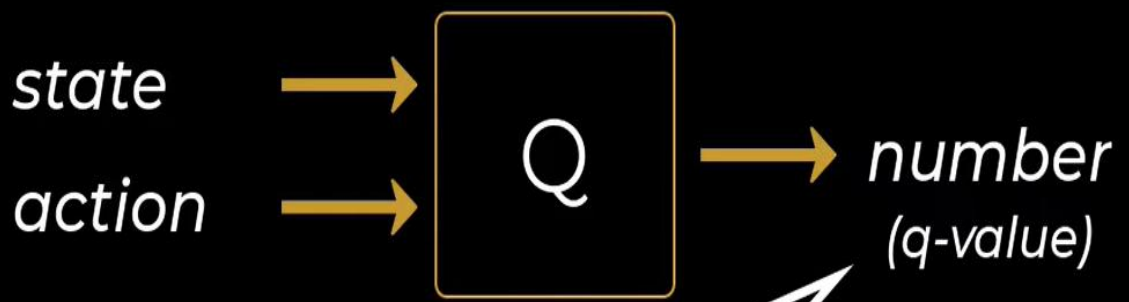
state value functions

V(s)

How good is it to be in the state **s**?

state-action value functions

Q(s, a)



How good is it to be in the state **s** and take an action **a** in this state?

State Value Function

$$v_*(s) = \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a]$$
$$= \max_a \sum_{s',r} p(s', r \mid s, a)\Big[r + \gamma v_*(s')\Big],$$

or

$$q_*(s,a) = \mathbb{E}\Big[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \;\Big|\; S_t = s, A_t = a\Big]$$
$$= \sum_{s',r} p(s', r \mid s, a)\Big[r + \gamma \max_{a'} q_*(s', a')\Big],$$

State- Action Value Function

$$V(S_1) = R(S_1) + \max\left[V(S_2), V(S_4)\right]$$

$$V(S_1) = R(S_1) + \max\Bigg[$$

$$\sum_{s'} P(s'|s = S_1, a = \text{right})V(s'),$$

$$\sum_{s'} P(s'|s = S_1, a = \text{down})V(s')\Bigg]$$

Introducing Stochasticity

Our state value function equation:

$$V(S_1) = R(S_1) + \max_a \left[\sum_{s'} P(s'|s = S_1, a) V(s') \right]$$

These are both versions of the **Bellman Equation**

Textbook:

$$\begin{aligned} v_*(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \max_a \sum_{s', r} p(s', r \mid s, a) \Big[r + \gamma v_*(s') \Big], \end{aligned}$$

State-Action Value Functions

$$Q(S_1, A_1 = \text{right}) = R(S_1) + \left[\max_{a'} Q(s' = S_2, a') \right]$$

Introducing Stochasticity:

$$Q(S_1, A_1) = R(S_1) + \max_{a'} \left[\sum_{s'} P(s'|s = S_1, a = A_1) Q(s', a') \right]$$

State-Action Value Functions

Our state-action value function equation:

$$Q(S_1, A_1) = R(S_1) + \sum_{s'} P(s'|s = S_1, a = A_1) \max_{a'} Q(s', a')$$

Textbook:

These are both versions of the **Bellman Equation**

$$q_*(s,a) = \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \;\middle|\; S_t = s, A_t = a\right]$$

$$= \sum_{s',r} p(s', r|s, a)\left[r + \gamma \max_{a'} q_*(s', a')\right],$$

State-Action Value Functions

$$Q(S_1, A_1 = \text{right}) = R(S_1) + \left[\max_{a'} Q(s' = S_2, a') \right]$$

Introducing Stochasticity:

$$Q(S_1, A_1) = R(S_1) + \max_{a'} \left[\sum_{s'} P(s'|s = S_1, a = A_1) Q(s', a') \right]$$

Removing P from the maximization:

$$Q(S_1, A_1) = R(S_1) + \sum_{s'} P(s'|s = S_1, a = A_1) \max_{a'} Q(s', a')$$

Applications of the Multi-Armed Bandit Problem

1. Online Advertising

In online advertising, MAB algorithms are used to dynamically select ads to display to users, balancing the exploration of new ads with the exploitation of ads that have shown high click-through rates.

2. Clinical Trials

MAB strategies help in clinical trials to allocate patients to different treatment arms, optimizing the trial outcomes by efficiently learning which treatments are most effective.

3. Recommender Systems

Recommender systems use MAB algorithms to suggest products, movies, or content to users, continuously learning and adapting to user preferences.

4. Adaptive Routing in Networks

MAB algorithms assist in adaptive routing by selecting network paths that maximize data transfer rates, balancing the exploration of new routes with the exploitation of known high-performing routes.

Action-Values

- The value is the expected reward

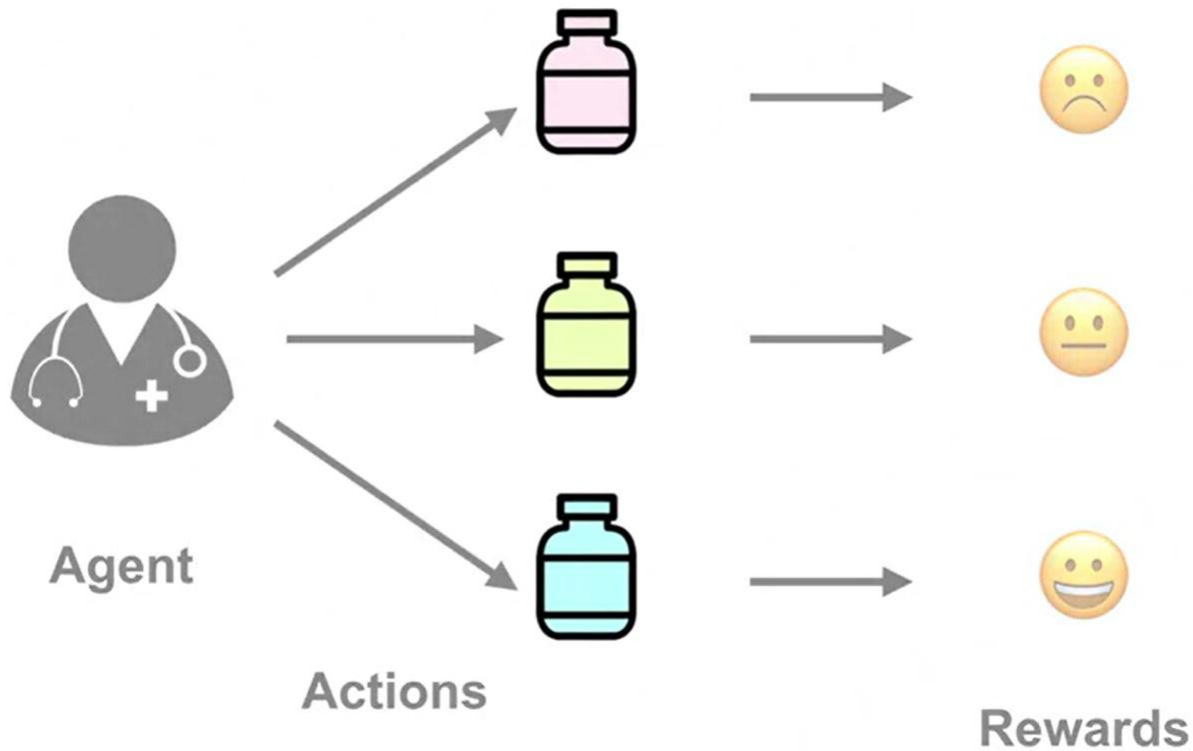
$$q_*(a) \doteq \mathbb{E}[R_t \mid A_t = a] \quad \forall a \in \{1, \ldots, k\}$$

$$= \sum_r p(r \mid a)\, r$$

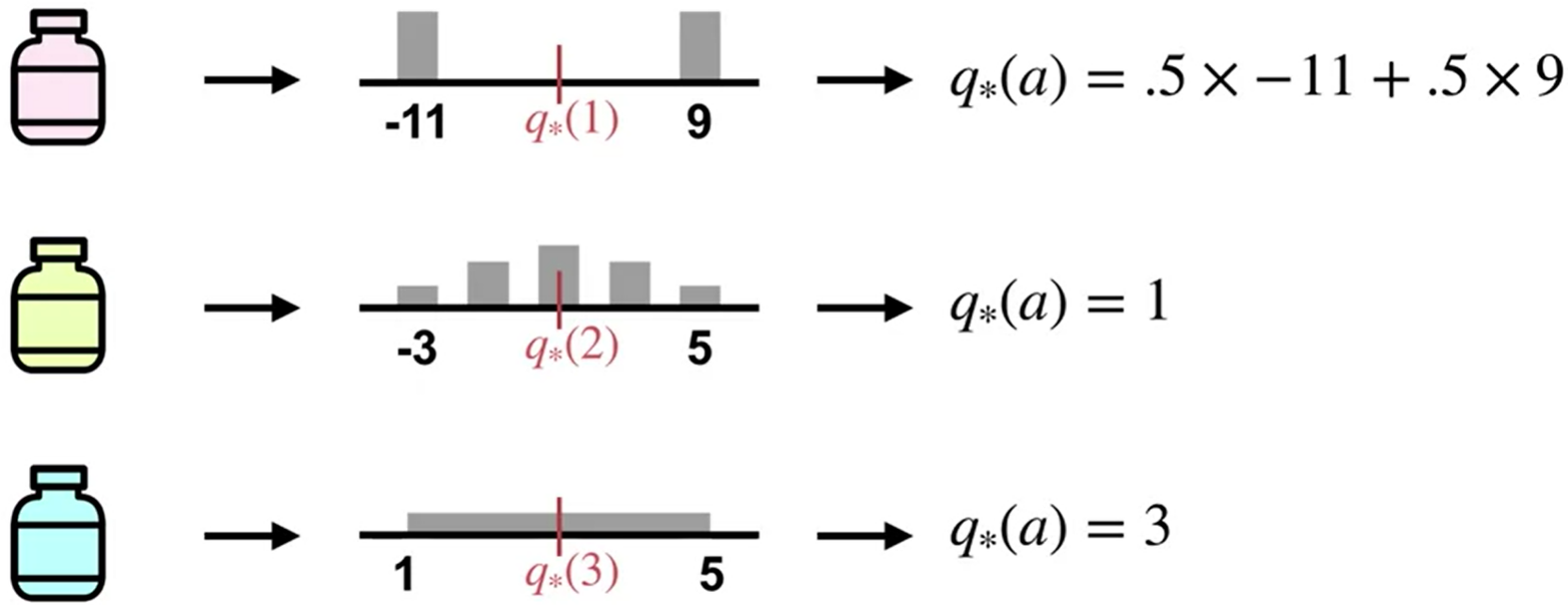
- The goal is to maximize the expected reward

$$\underset{a}{\operatorname{argmax}} \; q_*(a)$$

Calculating $q_*(a)$



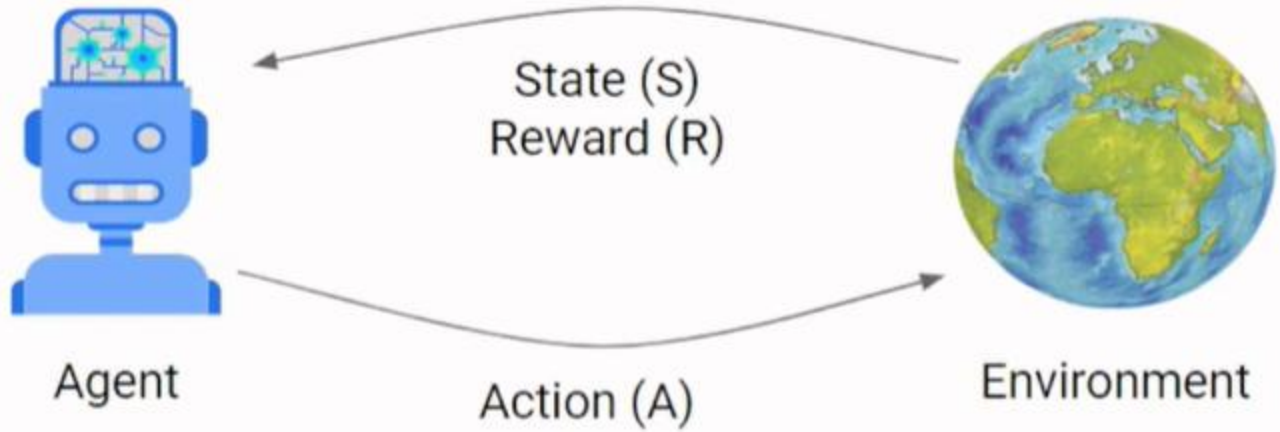
Calculating $q_*(a)$



Q Function vs. Value Function

Q Function	Value Function
Estimates the expected cumulative reward of taking a particular action in a given state.	Estimates the expected cumulative reward of being in a particular state.
State-action function.	State function.
Used to learn an optimal policy.	Used to evaluate different policies.
More complex, but more accurate.	Simpler, but less accurate.

Markov Decision Process

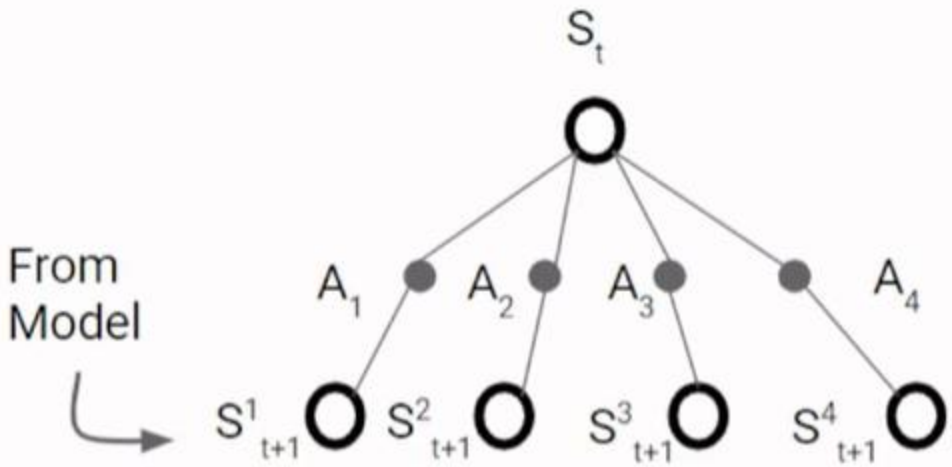


$$p\left(s', r \mid s, a\right) \doteq \Pr\left\{S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\right\}$$

Markov Property: "The future is independent of the past given the present."

$$p\left(R_{t+1} = r, S_{t+1} = s' \mid S_t = s\right) = p\left(R_{t+1} = r, S_{t+1} = s' \mid S_1, ..., S_{t-1}, S_t = s\right)$$

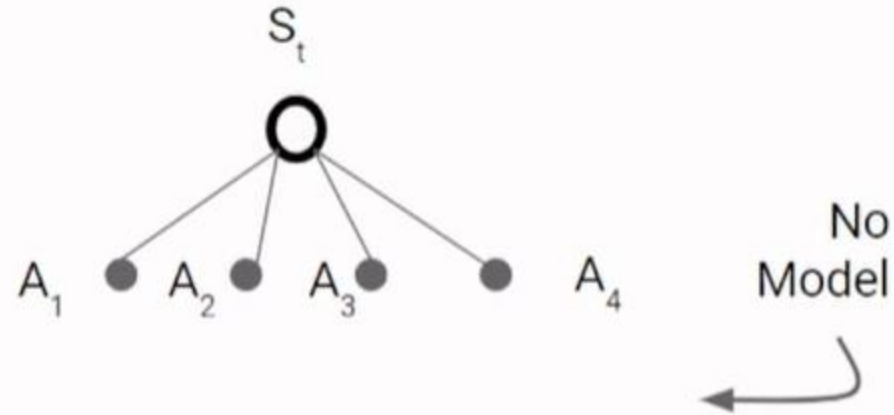
Model-Based



State Value

$$V_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s]$$

Model-Free



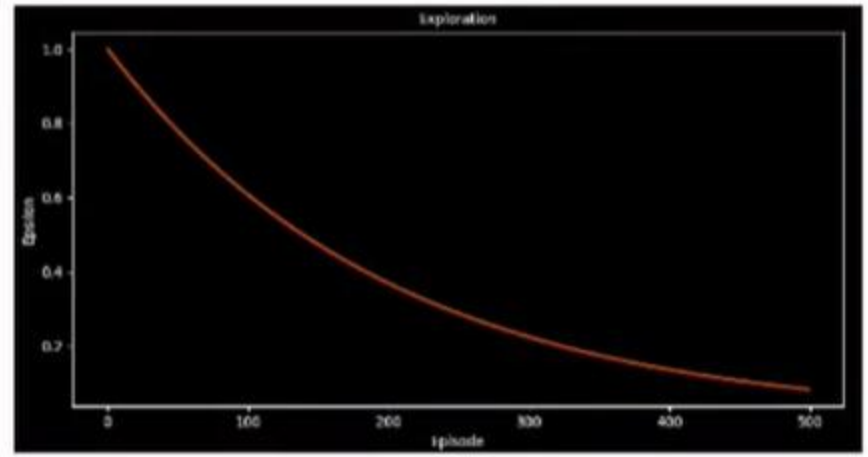
Action Value

$$Q_\pi(s,a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a]$$

Exploration and Exploitation

When do I explore?

When do I apply what I learnt?



Greedy Selection

$$A_t = \underset{a}{argmax}\ Q_t(a)$$

ε-Greedy Selection

$$A_t = \begin{cases} \underset{a}{argmax}\ Q_t(a) & with\ probability\ 1-\epsilon \\ any\ action & with\ probability\ \epsilon \end{cases}$$

What is in Policy and Value functions?

$$V_\pi(s)$$

$$\pi_t(S) \rightarrow \pi_{t+1}(S) \qquad V_{\pi_t}(S) \rightarrow V_{\pi_{t+1}}(S)$$

Policy Change

Reward Accumulation Change

Returns Change

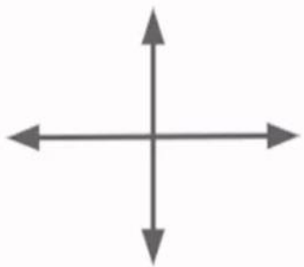
Values Change

How to calculate values in DP?

$$
\begin{aligned}
V_\pi(s) &\doteq \mathbb{E}_\pi\left[G_t \mid S_t = s\right] \\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma G_{t+1} \mid S_t = s\right] \\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma V_\pi(S_{t+1}) \mid S_t = s\right]
\end{aligned}
$$

$$V_\pi(s)$$

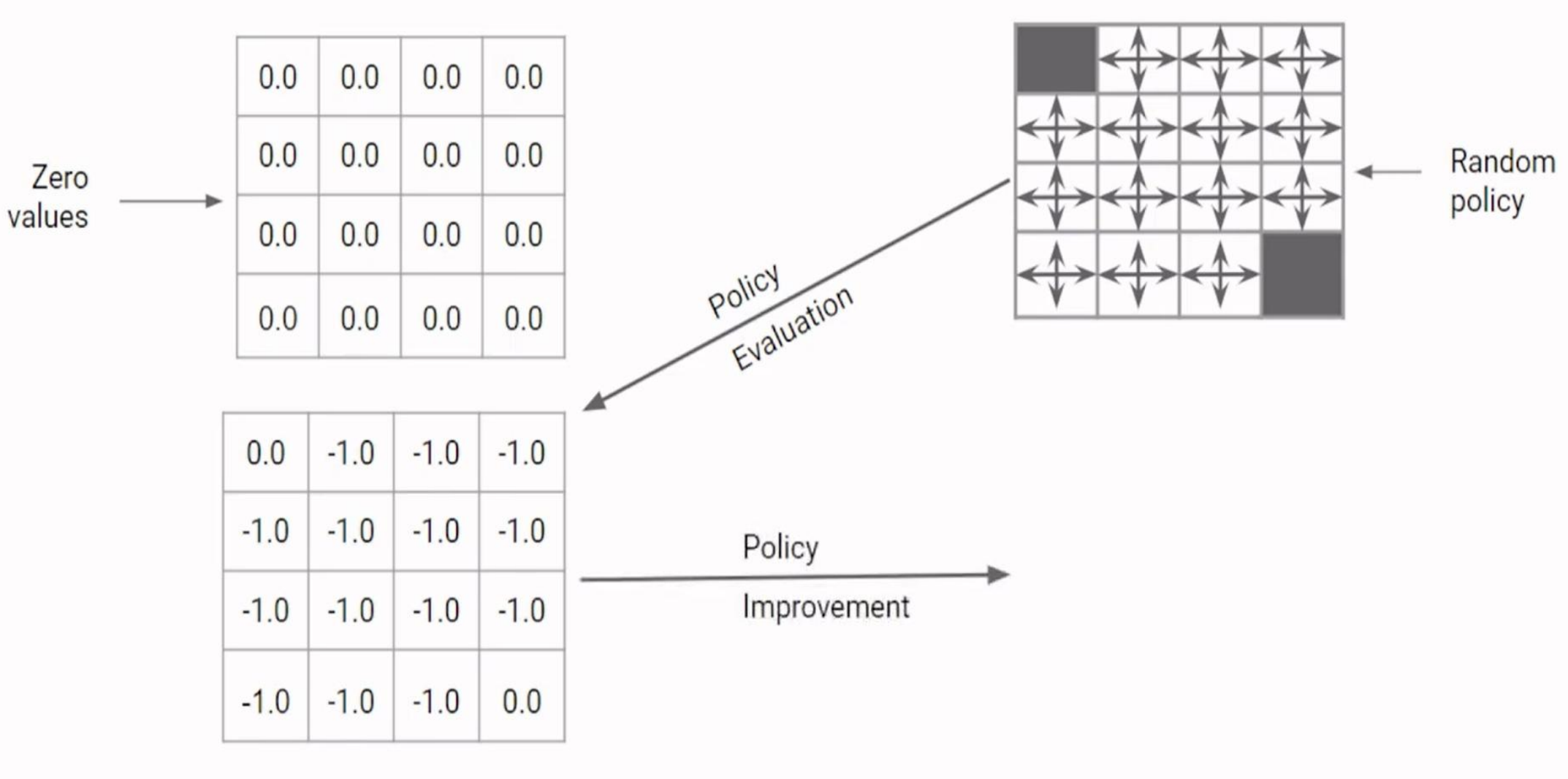
Grid World

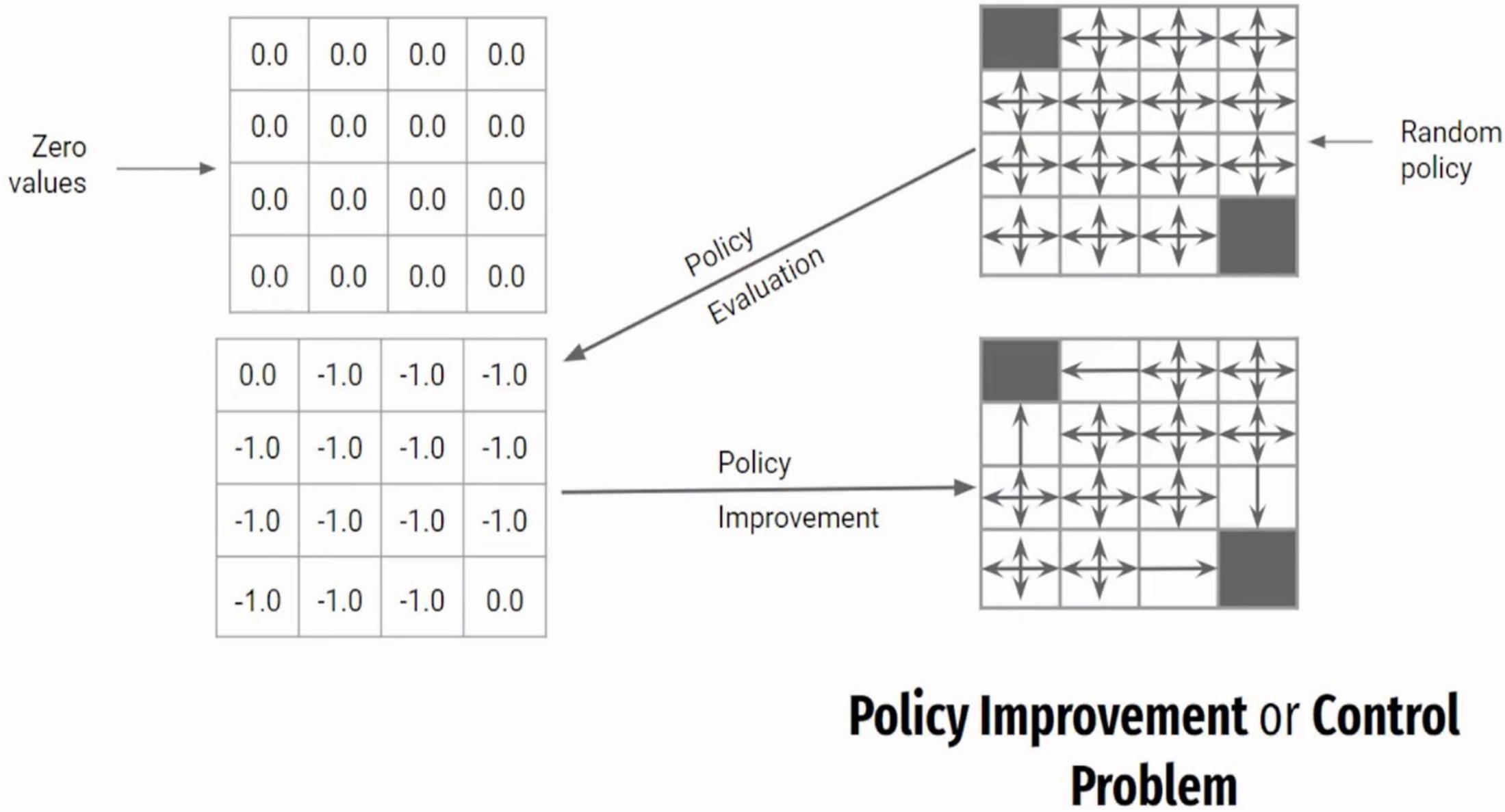


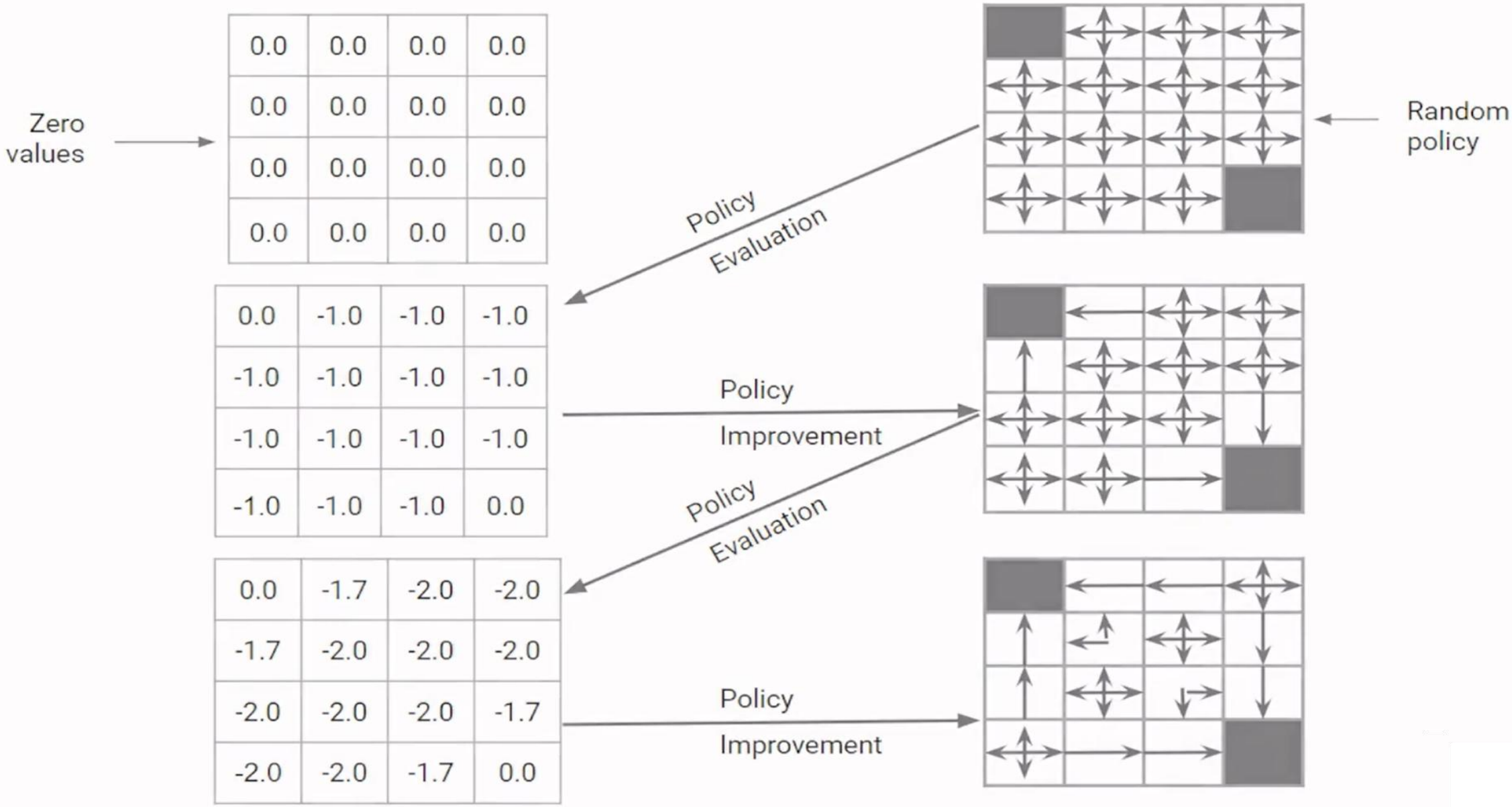
Actions

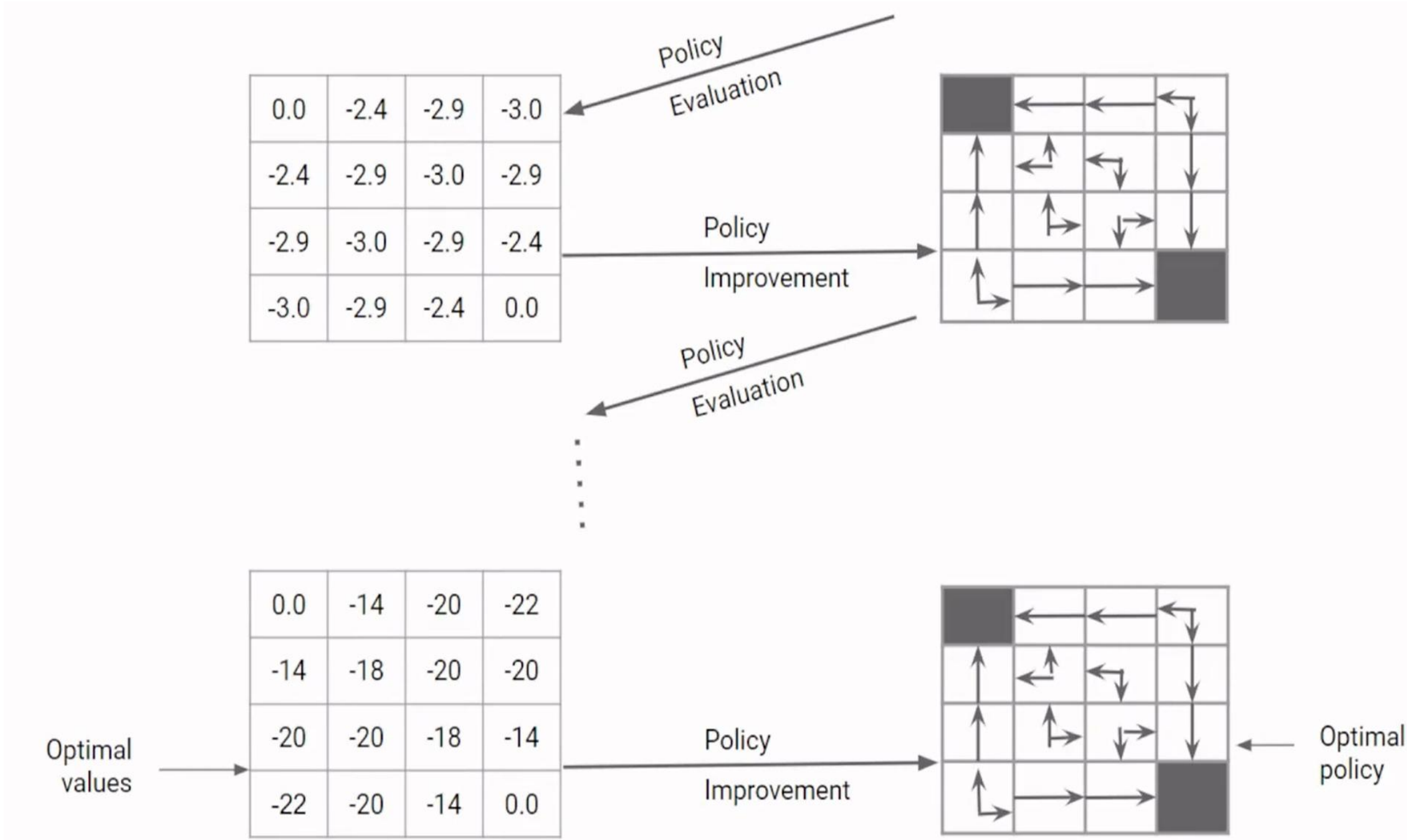
	1	2	3
4	5	6	7
8	9	10	11
12	13	14	

$R_t = -1$
on all transitions


Zero values
0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0
Random policy
Policy Evaluation
0.0 -1.0 -1.0 -1.0
-1.0 -1.0 -1.0 -1.0
-1.0 -1.0 -1.0 -1.0
-1.0 -1.0 -1.0 0.0
Policy Improvement


Zero values
0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0
Policy Evaluation
Random policy
0.0 -1.0 -1.0 -1.0
-1.0 -1.0 -1.0 -1.0
-1.0 -1.0 -1.0 -1.0
-1.0 -1.0 -1.0 0.0
Policy Improvement
Policy Improvement or Control Problem

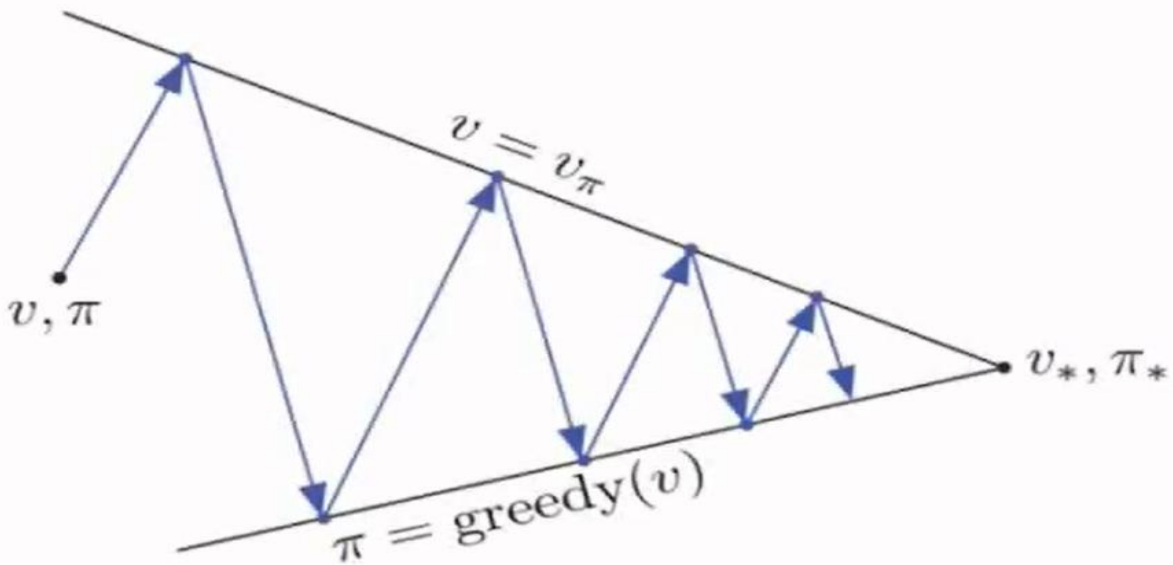

Zero values
0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0
Random policy
Policy Evaluation
0.0 -1.0 -1.0 -1.0
-1.0 -1.0 -1.0 -1.0
-1.0 -1.0 -1.0 -1.0
-1.0 -1.0 -1.0 0.0
Policy Improvement
Policy Evaluation
0.0 -1.7 -2.0 -2.0
-1.7 -2.0 -2.0 -2.0
-2.0 -2.0 -2.0 -1.7
-2.0 -2.0 -1.7 0.0
Policy Improvement

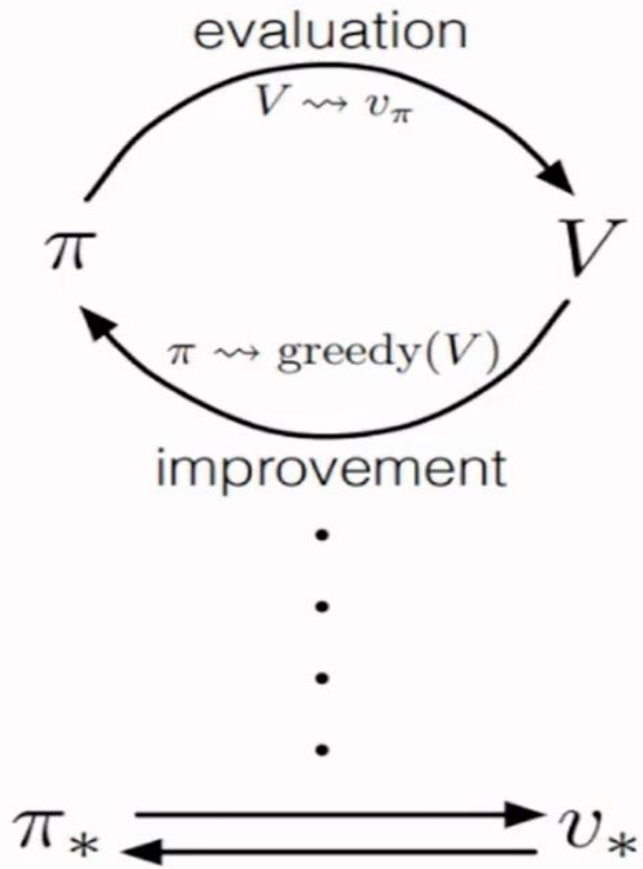

Policy
Evaluation
0.0 -2.4 -2.9 -3.0
-2.4 -2.9 -3.0 -2.9
-2.9 -3.0 -2.9 -2.4
-3.0 -2.9 -2.4 0.0
Policy
Improvement
Policy
Evaluation
Optimal
values
0.0 -14 -20 -22
-14 -18 -20 -20
-20 -20 -18 -14
-22 -20 -14 0.0
Policy
Improvement
Optimal
policy

Policy Iteration

$$\pi_0 \xrightarrow{E} V_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} V_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \cdots \xrightarrow{I} \pi_* \xrightarrow{E} V_*$$

Generalized Policy Iteration





The cycle of policy evaluation and policy improvement is called policy Iteration

Disadvantages of Dynamic Programing

- Calculation values for every state is not possible. In chess game there are huge number of possible state space.
- It require model of the environment which might not be available for all environment.
- Dynamic programming requires complete knowledge of environment
- Due to these computation infeasibility of environment we can go for the next class of method called Monte Carlo Methods