AMRITA VISHWA VIDYAPEETHAM
DEEMED TO BE UNIVERSITY

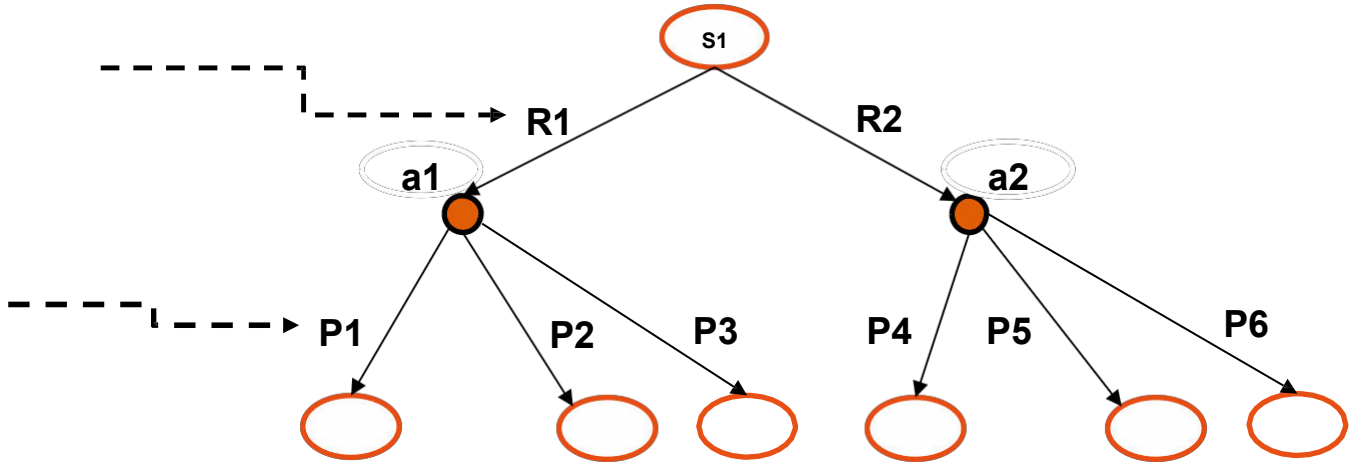
Model Free learning

Model Based Learning

- Environment is known
- Learn a model from experience

rewards for state-action are known

state transition probabilities are known

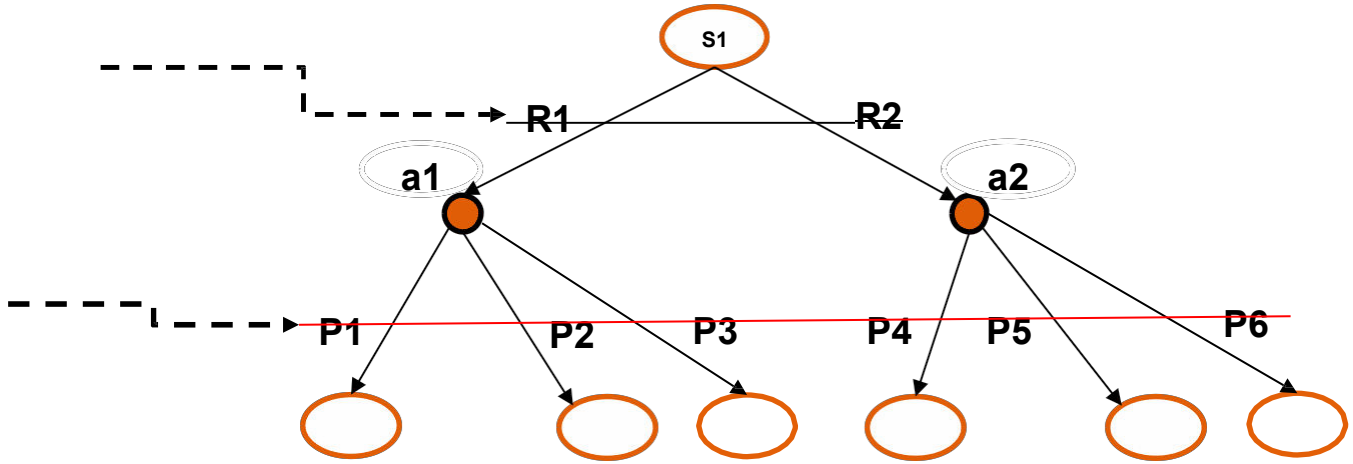


Model – Free Learning

- No model
- Learn the value function q(s, a) and/or the policy π(a|s) from experience

rewards for state-action are NOT known

state transition probabilities are NOT known



- **Model-free RL**:
  - Learner does **not** know transition probabilities or rewards beforehand.
  - Learns directly from experience (simulated or real).
  - Uses **trial and error + sample returns**.
  - Example: Monte Carlo learning, Q-learning.

👉 Key difference: *Model-based = you plan with a model*, *Model-free = you learn by interacting*.

Prediction and Control Problems

Prediction Problem

Policy

Evaluate the given Policy

Value function

Control Problem

The environment dynamics (rewards, transitions).

Evaluate the given Policy

Policy Improvement

Optimal Policy

In Reinforcement Learning, the **two fundamental problems** we want to solve are:

1.Prediction: *How good is this policy?*

2.Control: *What is the best policy?*

Every RL algorithm is basically trying to do one or both of these.

Classes of RL Algorithms

Prediction	
Model –based Planning	• Dynamic Programming
Model –free Learning	• Monte Carlo prediction • Temporal Difference TD(0) • TD(λ) Backward

Control	
Model –based Planning	• Dynamic Programming Value Iteration • Dynamic Programming Policy Iteration
Model –free Reinforcement Learning	• Monte Carlo Control • Sarsa • Q Learning • Deep Q Networks • Policy Gradient • Actor Critic

Model-based RL

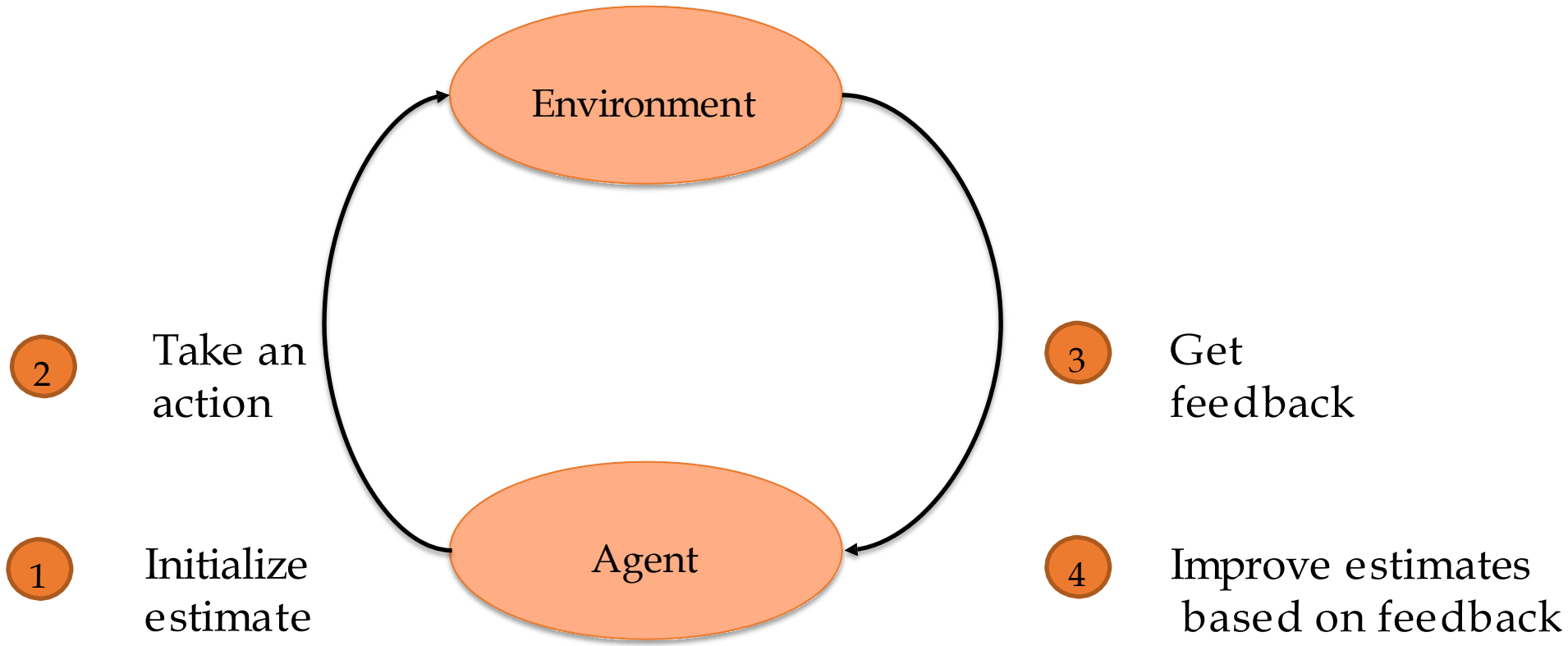
- MDP formulation, Dynamic Programming, Bellman Equations, Policy Iteration, Value Iteration.

Model-free RL

- **Gradient-free methods:**
  - Off-policy Q-learning.
  - On-policy SARSA, TD(0), TD(λ), Monte Carlo.
- **Gradient-based methods:**
  - Policy Gradient Optimization.
  - Deep Policy Networks.

- How does a model free algorithm works?

Model-free algorithms



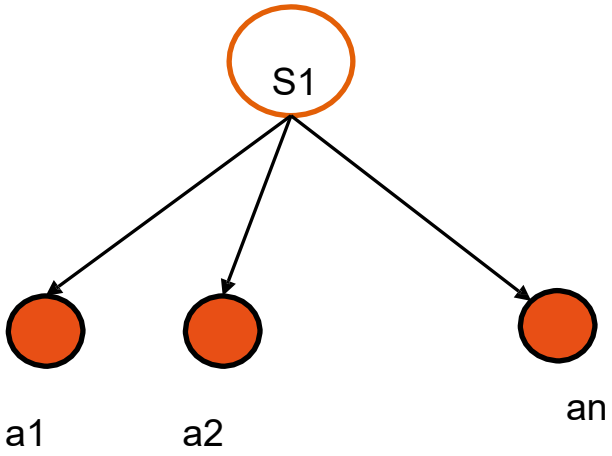
1. Initialize State-Action Value table

Initialize estimates with zero values and pick the initial state.

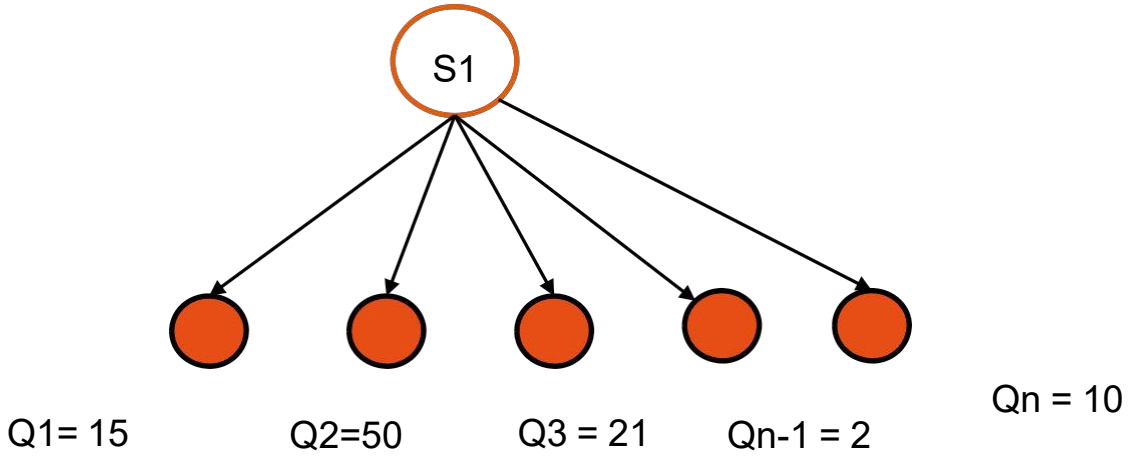
	a1	…	an
S1	0		0
S2	0		0
…			
Sn	0		0

2. Take an action

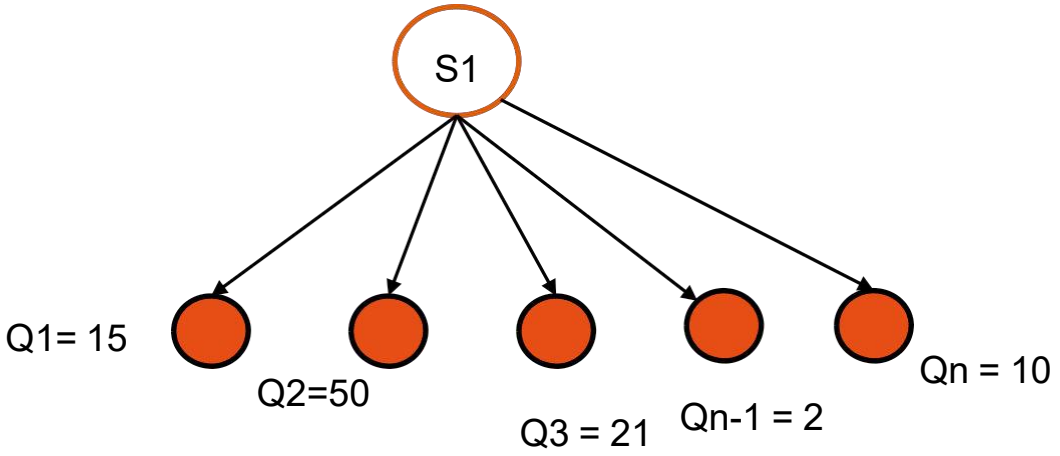
Select an action



Action Selection : Random



Action Selection : Greedy



Action Selection : ε-greedy

- makes use of the exploration-exploitation tradeoff

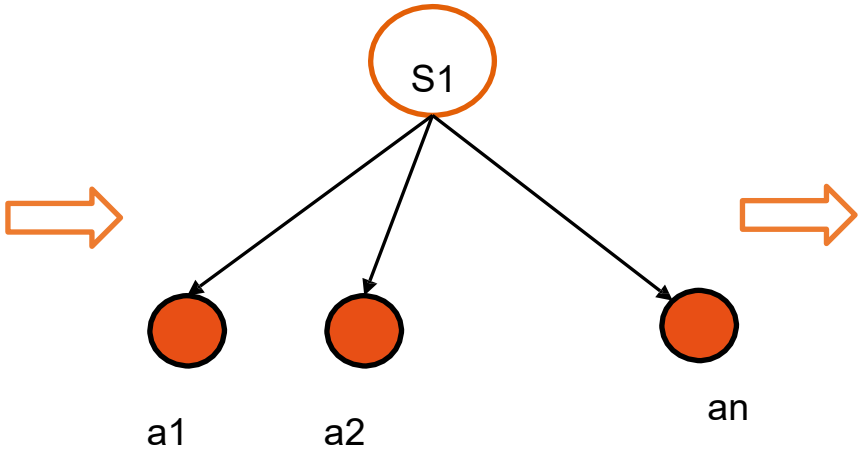
- Explore - choose a random option with probability epsilon
- Exploit - choose the option which so far seems to be the best

3. Get feedback from the environment

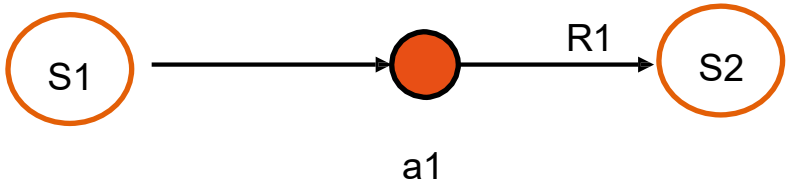
Initialization

	a1	…	an
S1	0		0
S2	0		0
…			
Sn	0		0

Agent action selection



Feedback (S1,a1,R1,S2)

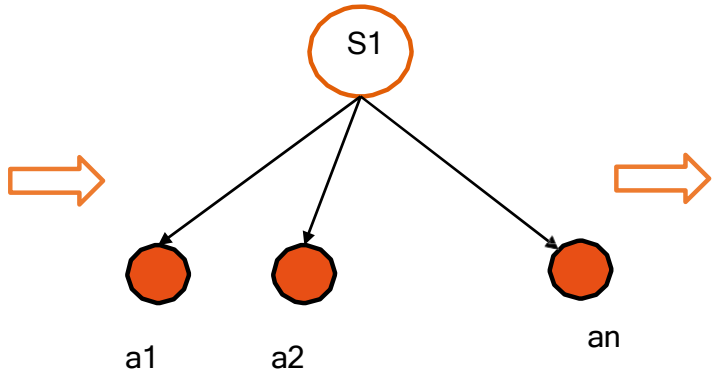


4. Improve Estimates

Initialization

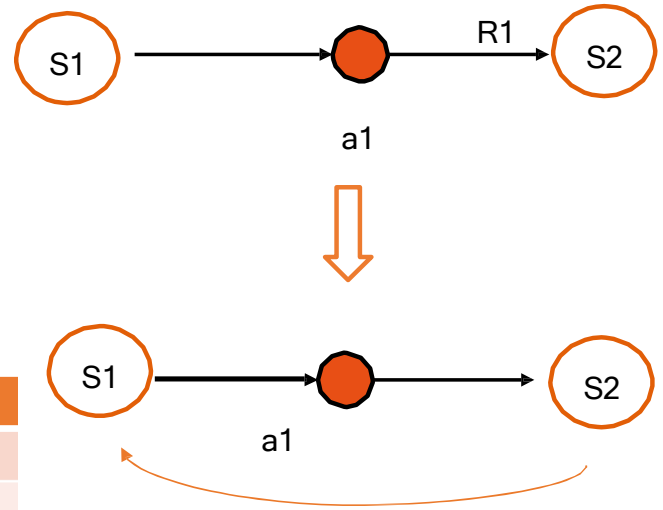
	a1	…	an
S1	0		0
S2	0		0
…			
Sn	0		0

Agent action selection



	a1	…	an
S1	q1		0
S2	0		0
…			
Sn	0		0

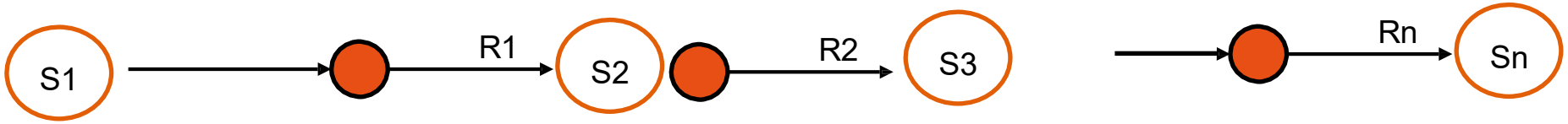
Feedback (S1,a1,R1,S2)



Update estimates using the observed reward

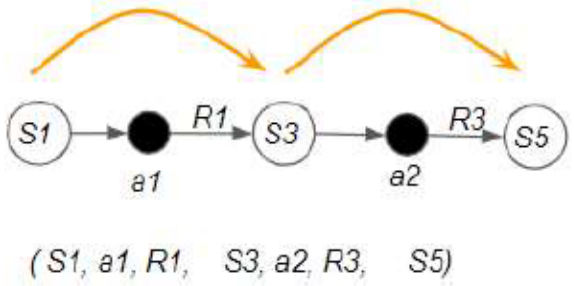
How to Improve Estimates?

- Frequency —the number of forward steps taken before an update.
  - Episode —At the end of the episode, add all rewards acquired and uses it to update the estimates.
  - One Step —Immediately after one step, observe rewards and update
  - NSteps –update after N steps.
- Depth —the number of backward steps to propagate an update.
- Formula to compute the updated estimate.



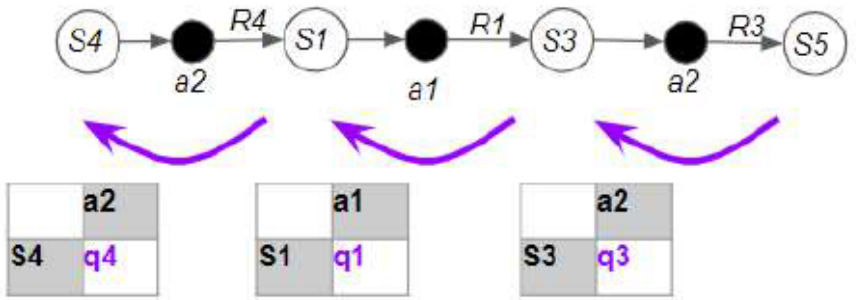
How to Improve Estimates?

- Frequency —the number of forward steps taken before an update.
  - Episode —At the end of the episode, add all rewards acquired and uses it to update the estimates.
  - One Step —Immediately after one step, observe rewards and update
  - NSteps –update after N steps.



How to Improve Estimates?

- Depth —how far back should the algorithm propagate its update estimates?
  - Episode —At the end of the episode, add all rewards acquired and uses it to update the estimates.
  - One Step —Immediately after one step, observe rewards and update
  - NSteps –update after N steps.



Update Formula

The formula used to update the estimates has many variations eg:

- Value-based updates use some flavor of the Bellman Equation to update the Q-value with an 'error' value. For example, this formula incrementally updates the Q-value estimate with an error value known as the TD Error.

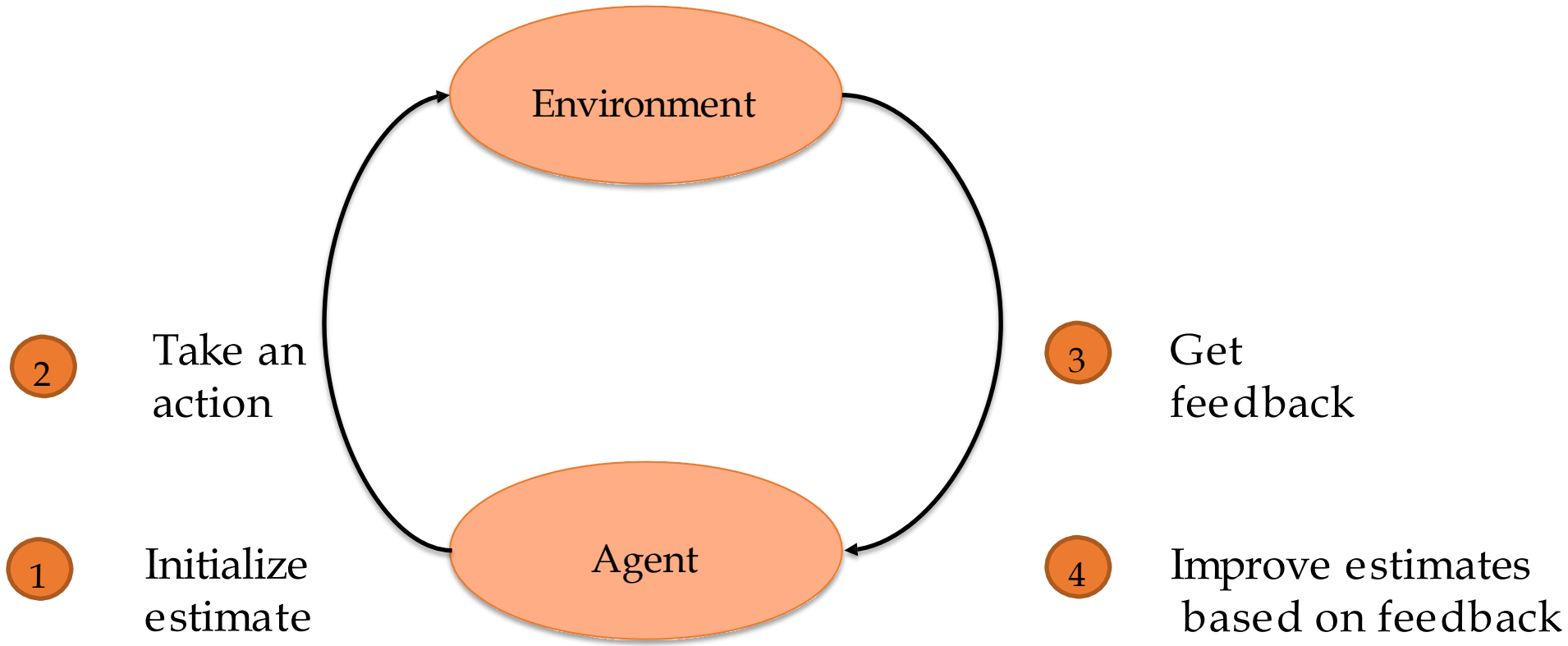
Error value is based on the difference between
i) Q-value of the next state + observed reward, and
ii) Q-value of the current state.

$$Q(S,A) = Q(S,A) + \alpha(R + \gamma\, Q(S',A') - Q(S,A))$$

(Image by Author)

- Policy-based updates increase or decrease the probability of the action that the agent took, based on whether we received a good reward or not.

Model-free algorithms



Monte Carlo Methods

The term **Monte Carlo** refers to a **method of estimation using random sampling**.

Monte Carlo Methods

- Monte-Carlo (MC) methods are statistical techniques for estimating properties of complex systems via **random sampling**
- MC for RL: learning method for estimating value functions and optimal policies
- **Complete knowledge of the environment is not required**
- Learn only from experience
- Can be applied only in episodic problems

Monte Carlo Learning

- Model free learning
- Agent learns from sampled experience
- Estimate value functions by simply *averaging sample returns*.

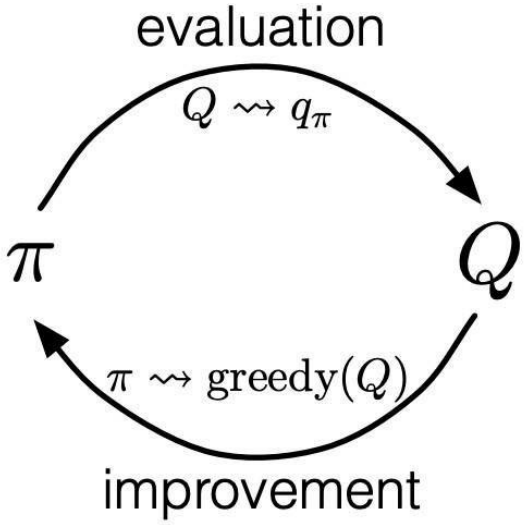
$$v_\pi(s) = E[G_t | S_t = s] = E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \ldots . | S_t = s]$$

$$q_\pi(s, a) = E[G_t | S_t = s, A_t = \text{a}] = E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \ldots . | S_t = s, A_t = a]$$

- Value updation only after a complete episode

Monte Carlo Methods

- Monte Carlo methods solve the reinforcement learning problem based on **averaging sample returns**.
- Monte Carlo methods sample and average returns for each state-action pair and average rewards for each action.
- The policy is updated only at the end of an episode.



Monte Carlo Prediction

- Objective: Estimates value function for a given policy $V^{\pi}(s)$
  - Input: episodes under π which contain s
  - Average returns observed after visits to s
  - Each occurrence of state s in an episode is called a **visit to s**
- **First-visit MC**: average returns only for first time s is visited in an episode
- **Every- Visit MC**.' average returns for every time s is visited in an episode
- Both converge asymptotically

Monte Carlo Algorithm (for Prediction)

Goal: Estimate $v_\pi(s)$for all states s, given a fixed policy π.

Algorithm:

1.Initialize:

1. $V(s)$arbitrarily for all states s(e.g., 0).
2. Returns(s) = empty list for all states.

2.Loop over many episodes:

1. Generate an episode following policy π: $S_0, A_0, R_1, S_1, A_1, R_2, \dots, S_T$.
2. For each state s appearing in the episode:
    1. Compute the **return G** following that occurrence.
    2. If using **First-Visit MC**: only consider the first time s is visited in this episode.
    3. If using **Every-Visit MC**: consider all visits.
    4. Append $G$to Returns(s).
    5. Update value estimate:

$$V(s) = average\big(Returns(s)\big)$$

1.Repeat until $V(s)$converges.

Monte Carlo prediction

Initialise V arbitrarily
$Returns(s) \leftarrow$ empty list $\forall s \in \mathcal{S}$
repeat
    Generate an episode using π
    for s in the episode **do**
    $Returns(s) \leftarrow$ append return following s
    end for
    $V(s) = average(Returns(s))$
until convergence

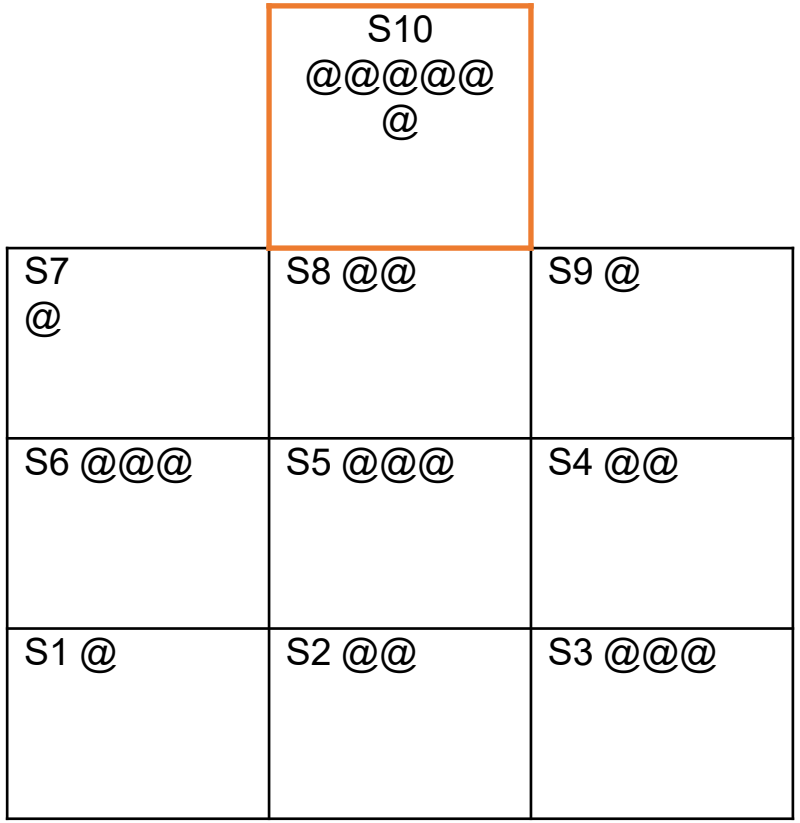
Example Problem

The Problem Setup (Chocolates as Rewards)

•Imagine a grid world where each state has some **chocolates** (that's the reward).
•The agent (child) moves around the grid, following some policy π.
•At the end of an **episode** (like a day of play), the total chocolates collected form the **return**.

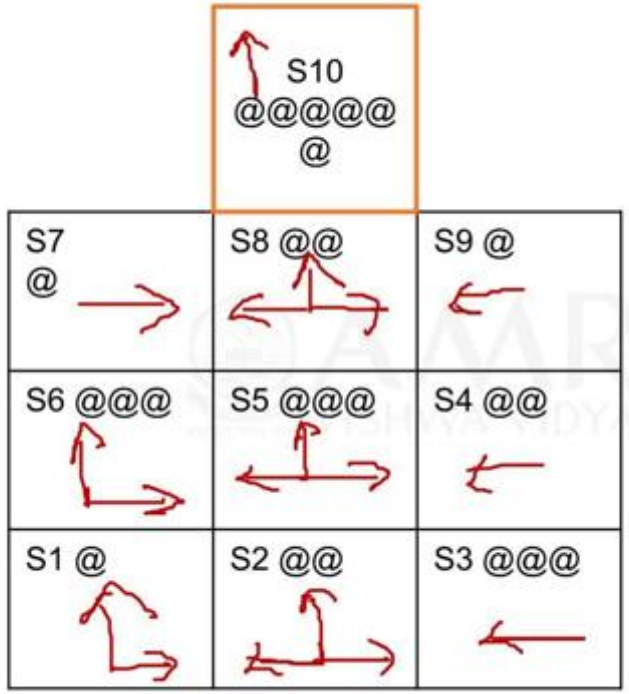
•We want to estimate the **value of a state Vπ(s)**:

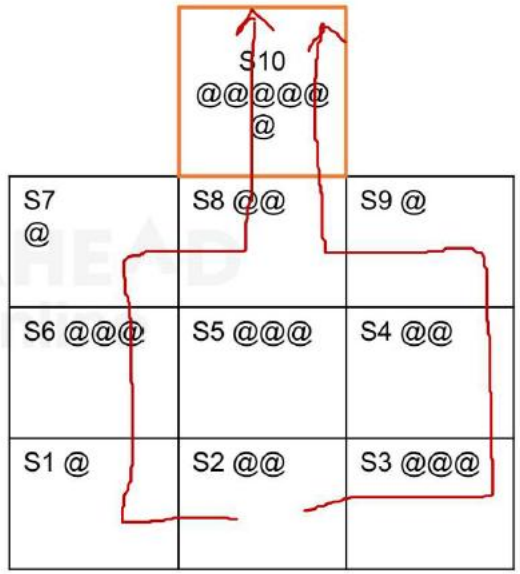
•*"If the agent starts at state s, how many chocolates can it expect on average in the future by following policy π?"*



The grid world has states S1 … S10.
S10 at the top is the terminal state (the goal).
Each cell (S1–S9) has "@" symbols — those are chocolates (rewards) you collect when you pass through the state.

Policy

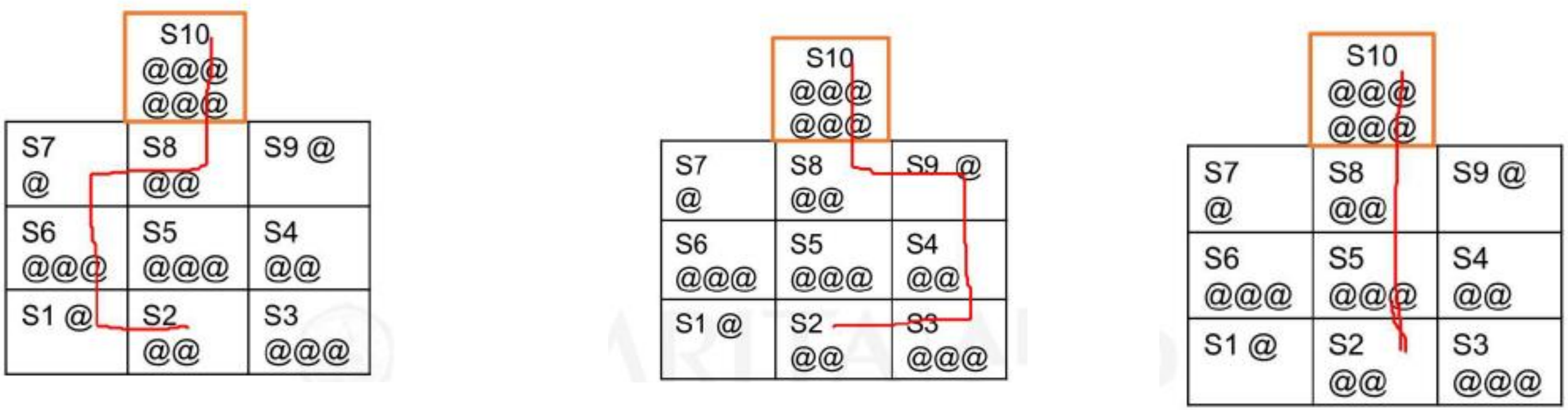




one possible pathway (e.g., go through S1 S6 → S8 → S10).

-another pathway (e.g., go S3 → S4 → S9->S10).

First Visit Monte Carlo Method



3 samples starting from state S_{05}

Return(Sample 01) = 2+ 1+ 3 + 1 + 2+ 6 = 15
Return (Sample 02) = 16
Return (Sample 03) = 13
Observed mean return =(15 + 16 + 13)/3 =14.6
Thus state value as per Monte Carlo Method, V_π (S 2) is 14.6 based on 3 samples following policy π.

First Visit Monte Carlo Method

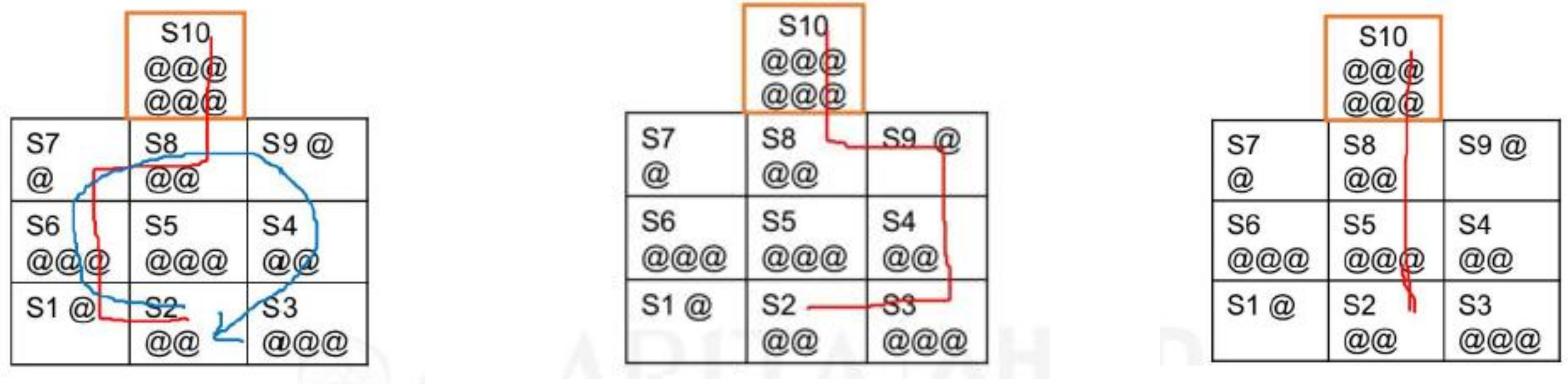
- To evaluate state s
- The first time-step t that state s is visited in an episode,
  - Increment counter N(s) ← N(s) + 1
  - Increment total return S(s) ← S(s) + G_t
  - Value is estimated by mean return V (s) =S(s)/N(s)

- Return(Sample 01) = 15
- Return (Sample 02) = 14
- Return (Sample 03) = 13
- Observed mean return =(15 + 14 + 13)/3 =14

Every Visit Monte Carlo Method

- To evaluate state s
- Every time-step t that state s is visited in an episode,
  - Increment counter N(s) ←N(s) +1
  - Increment total return S(s) ← S(s) + Gt
  - Value is estimated by mean return V (s) =S(s)/N(s)

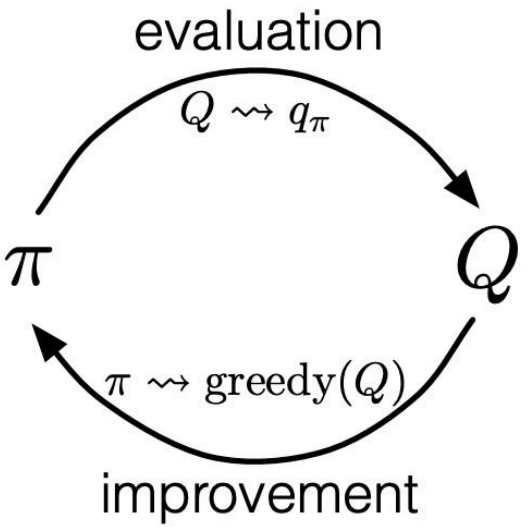
Every Visit Monte Carlo Method



- Return(Sample 01) = 2+ 1+ 3 + 1 + 2+ 6 + (2+ 1 +3+1+2+1+2 +3)= 30
- Return (Sample 02) = 16
- Return (Sample 03) = 13
- Observed mean return =(30 + 16 + 13)/4 = 14.75
- State value, V_π (S 2) is 14.25 based on 3 samples following policy π.

	Advantages	Disadvantages
First-Visit MC	✔ Simpler and avoids correlated samples. ✔ More stable for long episodes. ✔ Easier theoretical analysis.	✘ Fewer data per episode → slower convergence. ✘ Wastes some information when states repeat.
Every-Visit MC	✔ Uses all available information. ✔ Converges faster in practice (more samples).	✘ Returns from same episode are correlated. ✘ Slightly more computation.

Monte Carlo Control

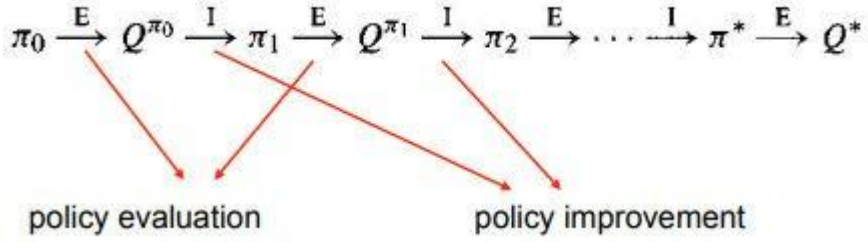


- To approximate optimal policies
  - **MC policy iteration**: Policy evaluation using MC methods followed by policy improvement
  - **Policy improvement step**: greedify with respect to value (or action value) function

Follow generalized policy iteration

Policy improvement part is done by taking greedy actions in each state.

$$\pi(s) \doteq \arg\max_a q(s,a).$$



Goal:

To *approximate optimal policies* π^* —that is, to learn what actions to take in each state to maximize the total expected return.

Exploring Starts

- Many state-action pairs may never be visited
- For policy evaluation, the agent should visit all actions in every state
  - Such That every episode will start in some state–action pair
- **Every pair has a probability > 0 - to be selected as the start pair**

$$Q \xrightarrow[\text{evaluation}]{\pi} q_\pi \quad \Rightarrow \quad \pi \xrightarrow[\text{improvement}]{\text{greedy}(Q)} \pi'$$

$$\pi_0 \xrightarrow{E} Q^{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} Q^{\pi_1} \xrightarrow{I} \cdots \rightarrow \pi^*, Q^*$$

E: Evaluation step.

I: Improvement step.

Monte Carlo Exploring Starts

Monte Carlo ES (Exploring Starts), for estimating $\pi \approx \pi_*$

Initialize:

$\pi(s) \in \mathcal{A}(s)$ (arbitrarily), for all $s \in \mathcal{S}$
$Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$
$Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Loop forever (for each episode):

Choose $S_0 \in \mathcal{S}$, $A_0 \in \mathcal{A}(S_0)$ randomly such that all pairs have probability > 0
Generate an episode from S_0, A_0, following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
$G \leftarrow 0$
Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
$G \leftarrow \gamma G + R_{t+1}$
Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:
Append G to $Returns(S_t, A_t)$
$Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$
$\pi(S_t) \leftarrow \text{argmax}_a\, Q(S_t, a)$

Monte Carlo Control without Exploring Starts

- On-policy methods: evaluate or improve the current policy.

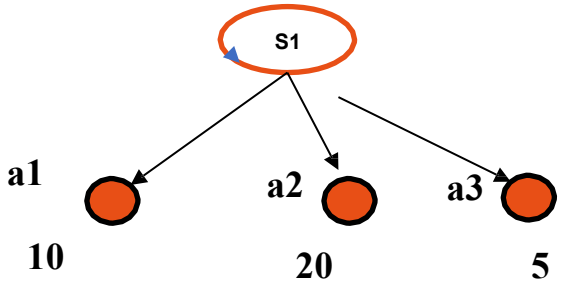
  The agent **learns about the same policy it is following** to generate data.

- Off-policy methods: evaluate or improve one policy, and use another for actions.

  The agent learns about **one policy (the target policy)** while behaving according to **another (the behavior policy).**

On-policy Monte Carlo Control

- On-policy: learn about policy currently executing
- How to solve exploring starts?
  - soft policies: $\pi(s,a) > 0$ for all s and a
  - ε-greedy policy



$$\frac{\varepsilon}{|A(s)|} \quad non-max$$

$$1-\varepsilon+\frac{\varepsilon}{|A(s)|} \quad greedy$$

- Converges to best ε-soft policy

We want to:
•**Keep exploring** (try all actions sometimes)
•**But mostly exploit** (choose the best-known action most of the time)
This balance is achieved through an **ε-greedy policy**.

On-policy Monte Carlo Control- ε-greedy policy

We decide that **a small fraction ε** of the total probability will be reserved for *exploration*.
This means:

- With probability ε, choose a random action.
- So the remaining $1-\varepsilon$ is for *exploitation* (taking the best action).

- **Greedy action** $a^* = \arg\max_{a'} Q(s,a')$:
  It gets all the exploitation probability $1-\varepsilon$
  plus its equal share of exploration $\frac{\varepsilon}{|A(s)|}$.

$$\pi(a^*|s) = (1-\varepsilon) + \frac{\varepsilon}{|A(s)|}$$

- **Non-greedy actions:**
  They only get their exploration share $\frac{\varepsilon}{|A(s)|}$.

$$\pi(a|s) = \frac{\varepsilon}{|A(s)|}$$

$$P_{\text{explore}}(a|s) = \frac{1}{|A(s)|}.$$

Across all steps contribution

$$\varepsilon \times \frac{1}{|A(s)|} = \frac{\varepsilon}{|A(s)|}.$$

Sum up to 1

$$(1-\varepsilon) + \frac{\varepsilon}{|A(s)|} + (|A(s)|-1)\frac{\varepsilon}{|A(s)|} = (1-\varepsilon) + \varepsilon = 1.$$

On-policy first-visit MC control (for ε-soft policies), estimates $\pi \approx \pi_*$

Algorithm parameter: small $\varepsilon > 0$

Initialize:

$\pi \leftarrow$ an arbitrary ε-soft policy

$Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

$Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Repeat forever (for each episode):

Generate an episode following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

$G \leftarrow \gamma G + R_{t+1}$

Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:

Append G to $Returns(S_t, A_t)$

$Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$

$A^* \leftarrow \arg\max_a Q(S_t, a)$ (with ties broken arbitrarily)

For all $a \in \mathcal{A}(S_t)$:

$$\pi(a|S_t) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$$

Monte Carlo off-policy methods

- In off-policy methods : target policy the policy being learned
  - Target policy is the greedy policy with respect to Q
  - Behaviour policy the policy that generates behaviour
  - Behaviour policy for exploration
  - Behaviour policy is soft (select all actions in all states with non-zero probability)

In on-policy methods (like ε-greedy MC):

- The same policy must explore and learn.
- That means it can't be fully greedy — it must keep a nonzero ε forever to explore.

In off-policy methods:

- We can have a **fully greedy target policy** (learning the best actions)
- While still exploring using a **different behavior policy** (like random or ε-greedy).

Hence, we can learn **optimal deterministic policies** even while exploring!

In **on-policy learning, the agent learns from the outcomes of its own behavior**.
In **off-policy learning, it can learn from data** collected under a different behavior — maybe a random one — while still improving its own greedy target policy.
This **makes off-policy methods more flexible,** though a bit more complex mathematically.