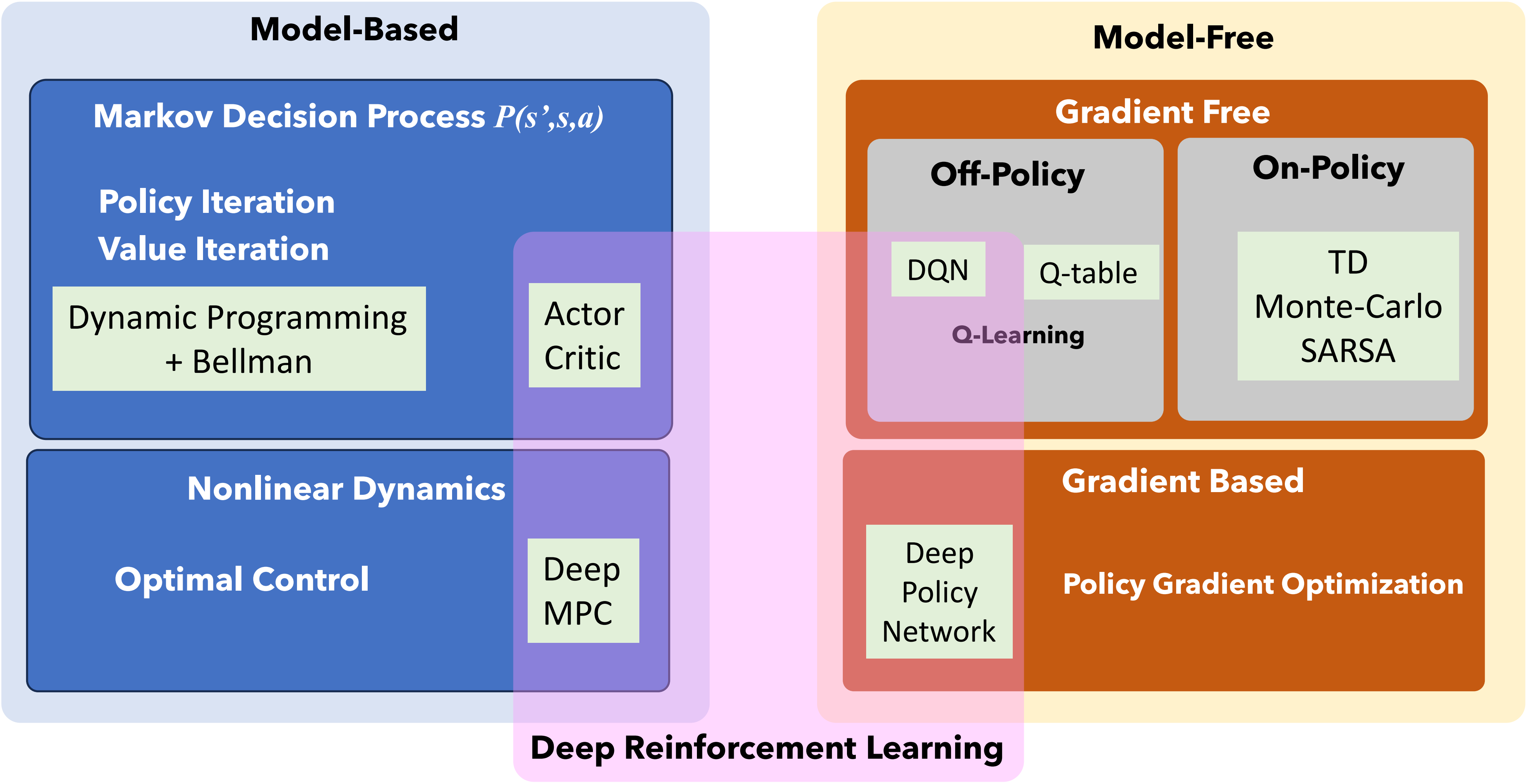
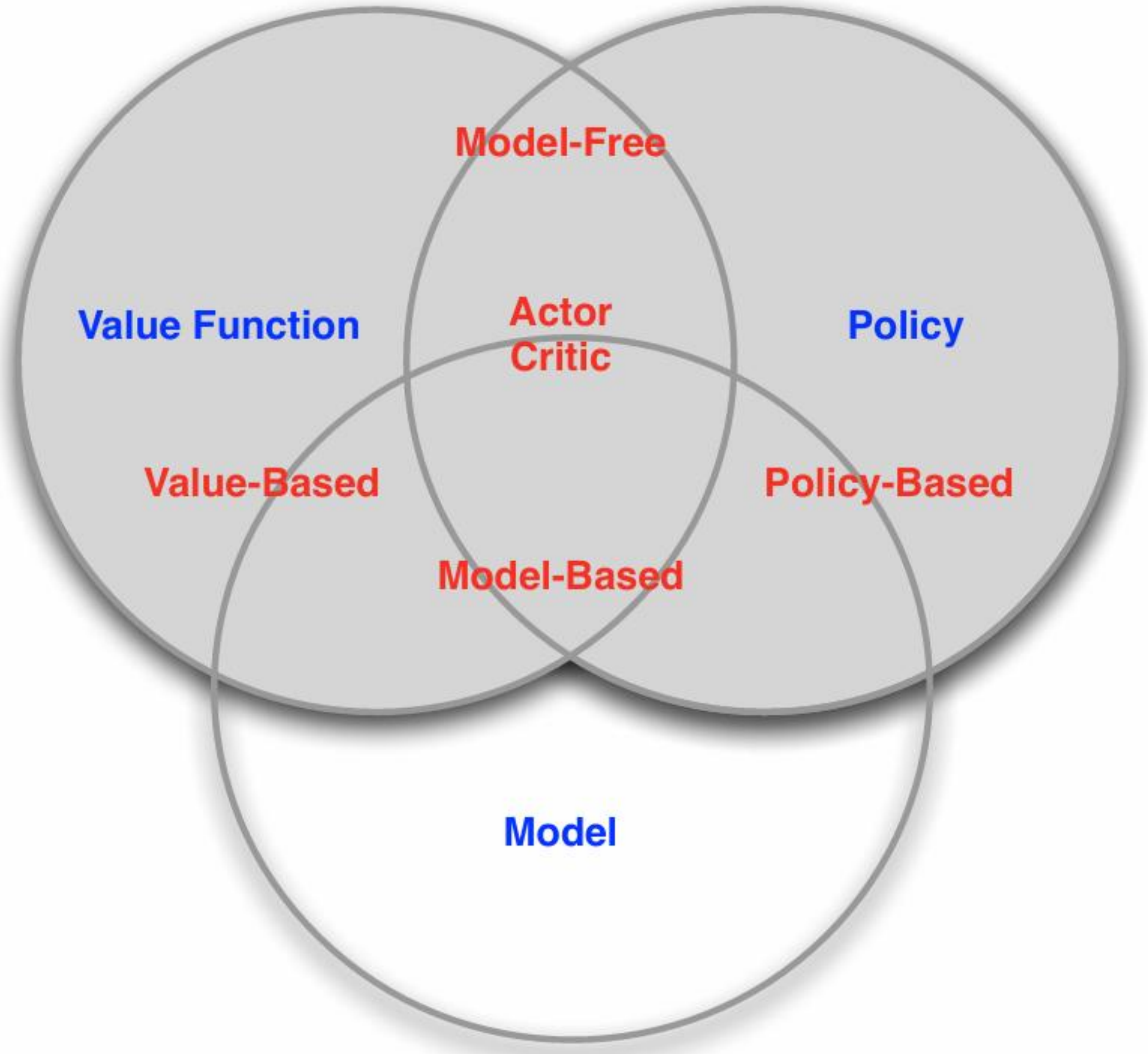
Lecture 7
SARSA / Q-Learning

jmanero@faculty.ie.edu

Classification of RL Methods



Taxonomy of RL Agents



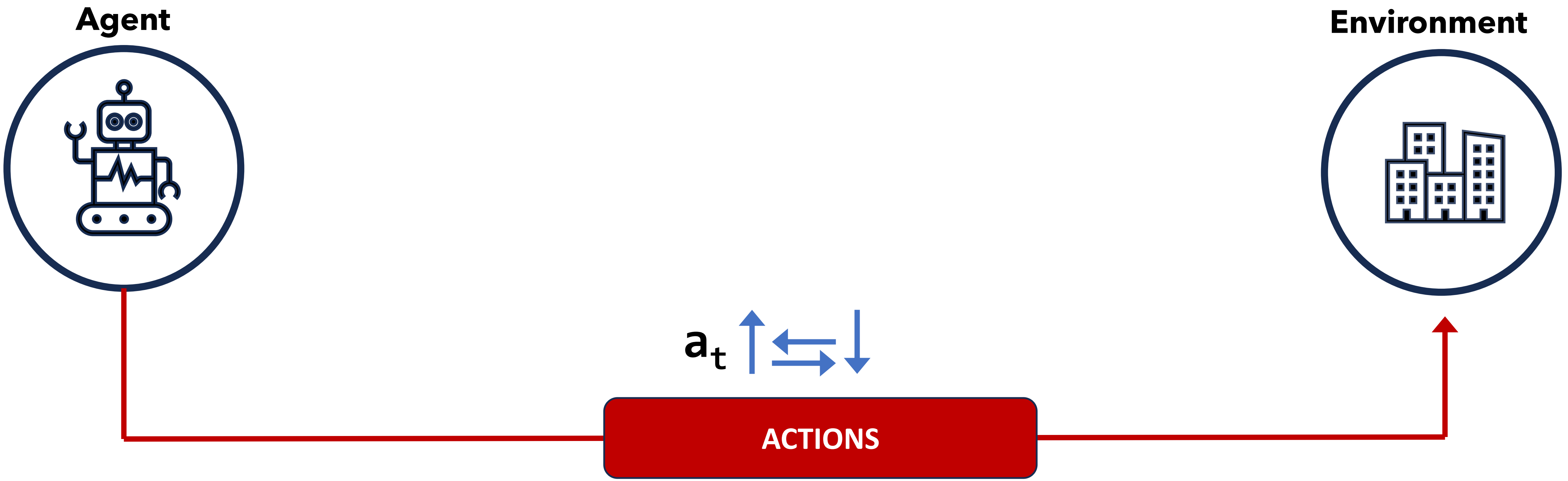
Taxonomy of RL Agents

(*) From David Silver RL Course UCL

Contents

- Review Concepts
- SARSA
- Q-learning

Key Concepts: Actions



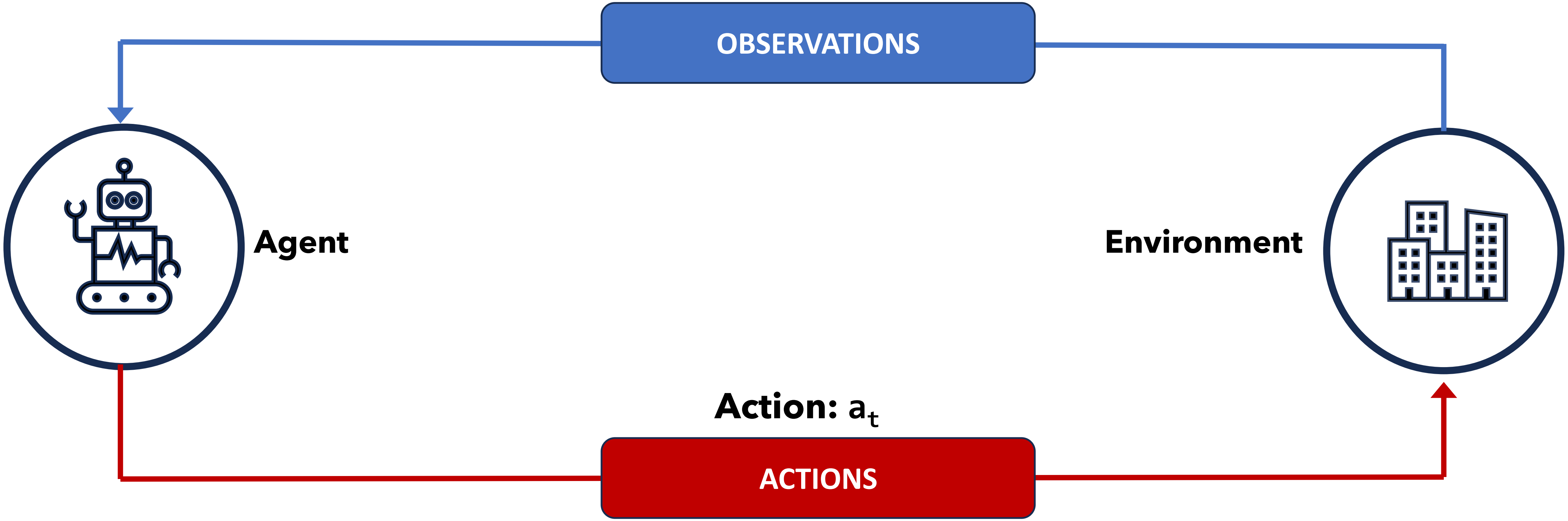
ACTION DEFINITION

A move the agent can make in the Environment

Action Space $\boldsymbol{A}$: The set of possible actions an agent can perform in the environment

$$A = \{a_1, a_2, \; .. \; , a_n\}$$

Key Concepts: Observations

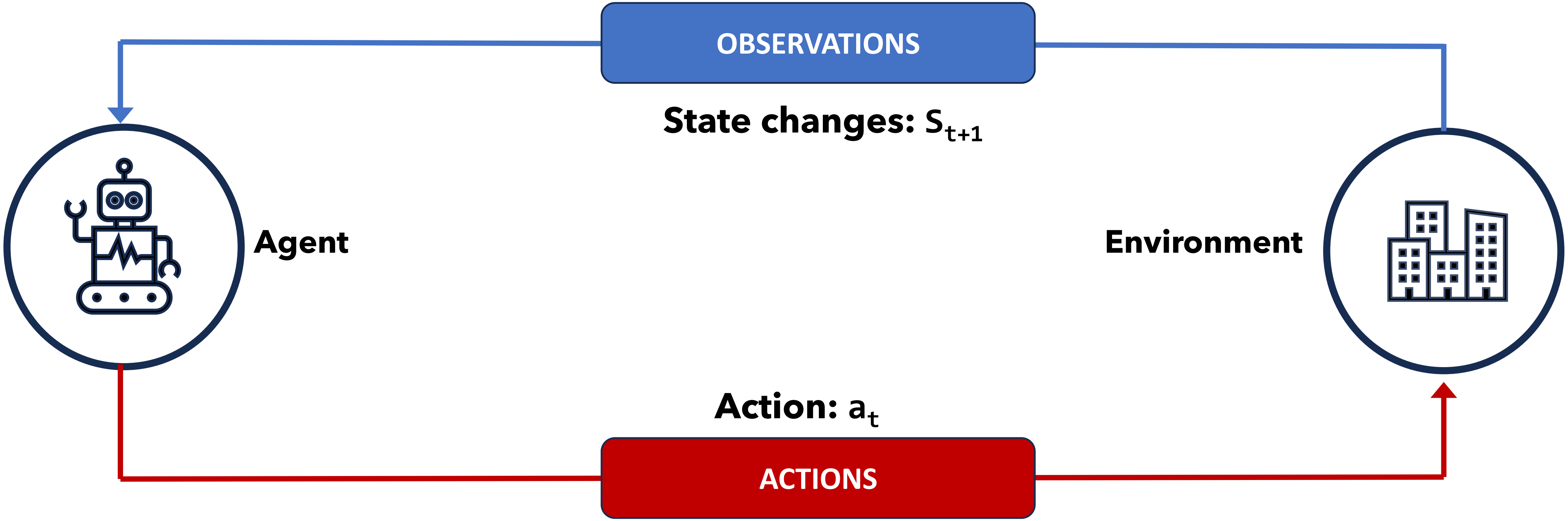


OBSERVATION DEFINITION

Understand the environment after taking actions

(What has changed from last observation, what is new, …)

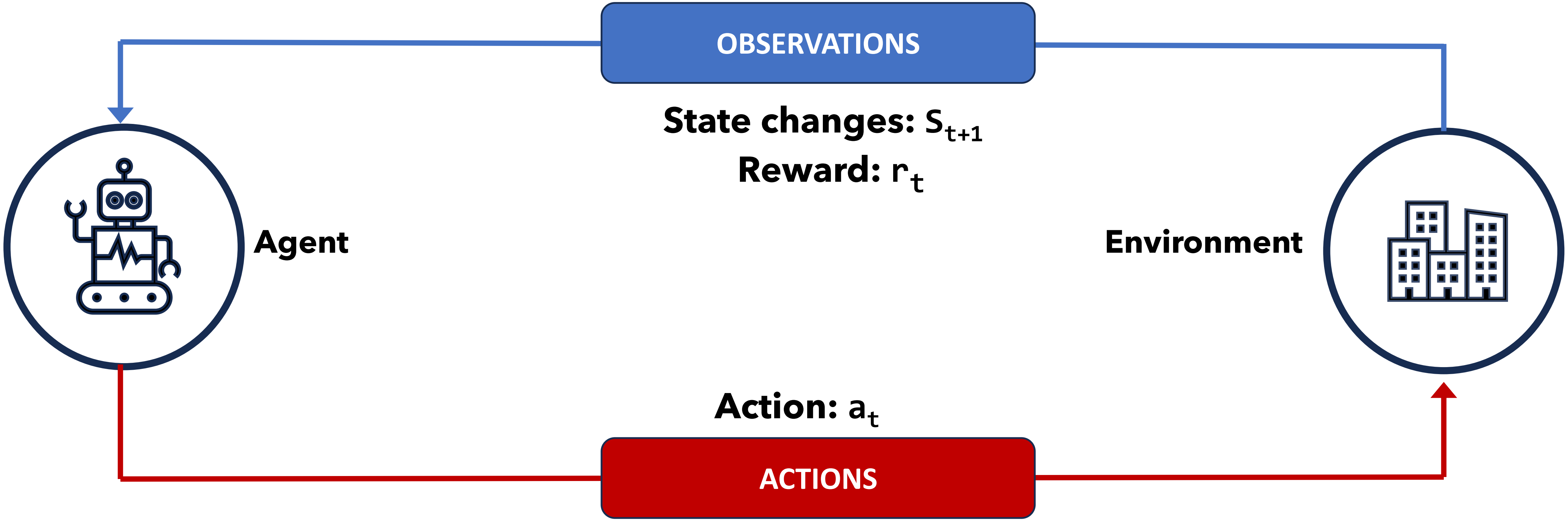
Key Concepts: State and State change



STATE DEFINITION

A situation of the environment that the Agent can perceive as different

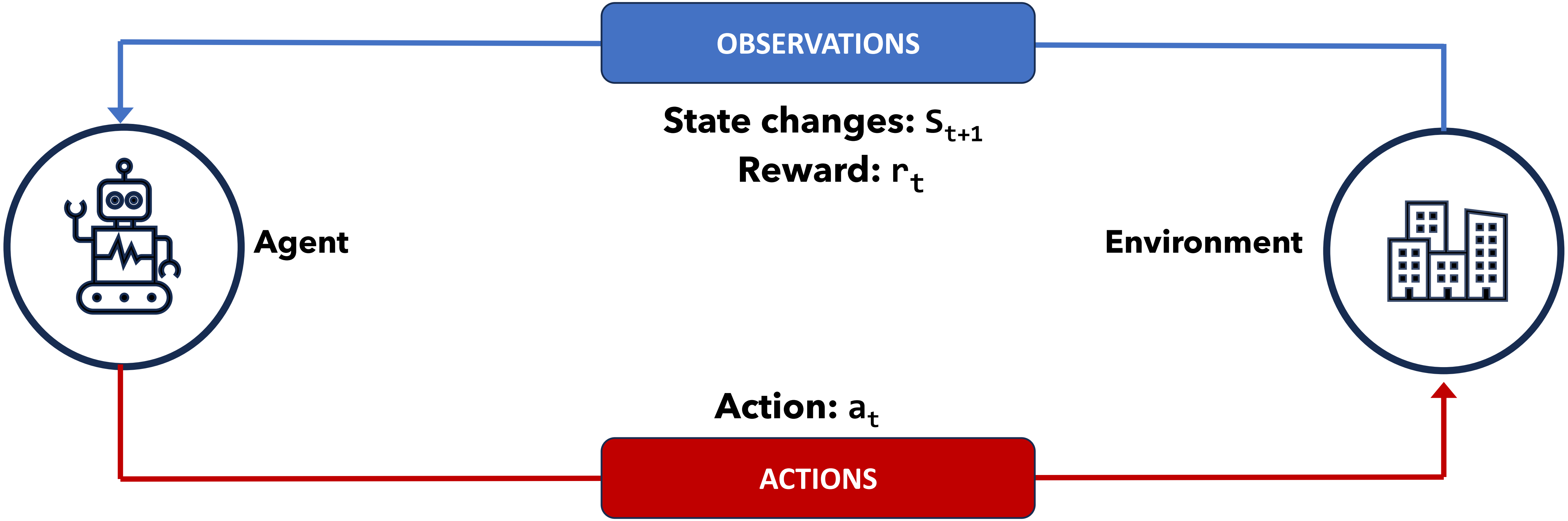
Key Concepts: Reward



REWARD DEFINITION

Feedback that measures the success or failure of the agent's action

Key Concepts: Total Reward

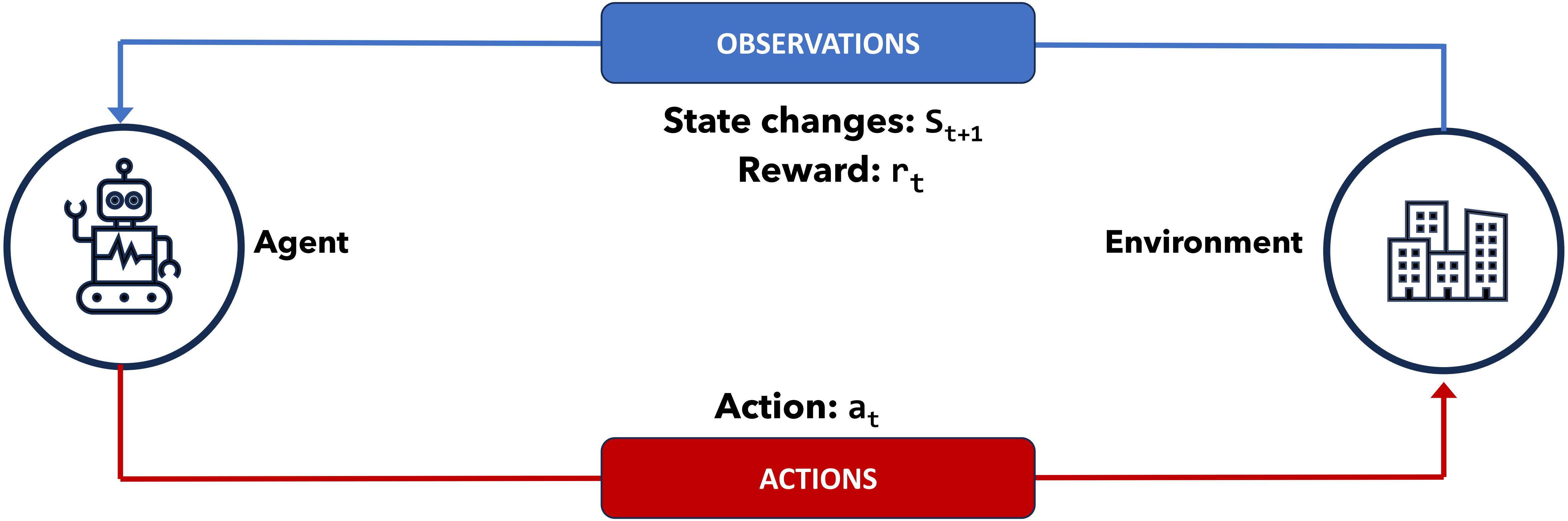


TOTAL REWARD (Return) DEFINITION

Is the Summarization of the total rewards pending until the end of the movement in the universe

$$R_t = \sum_{i=t}^{\infty} r_i$$

Key Concepts: Total Reward decomposition



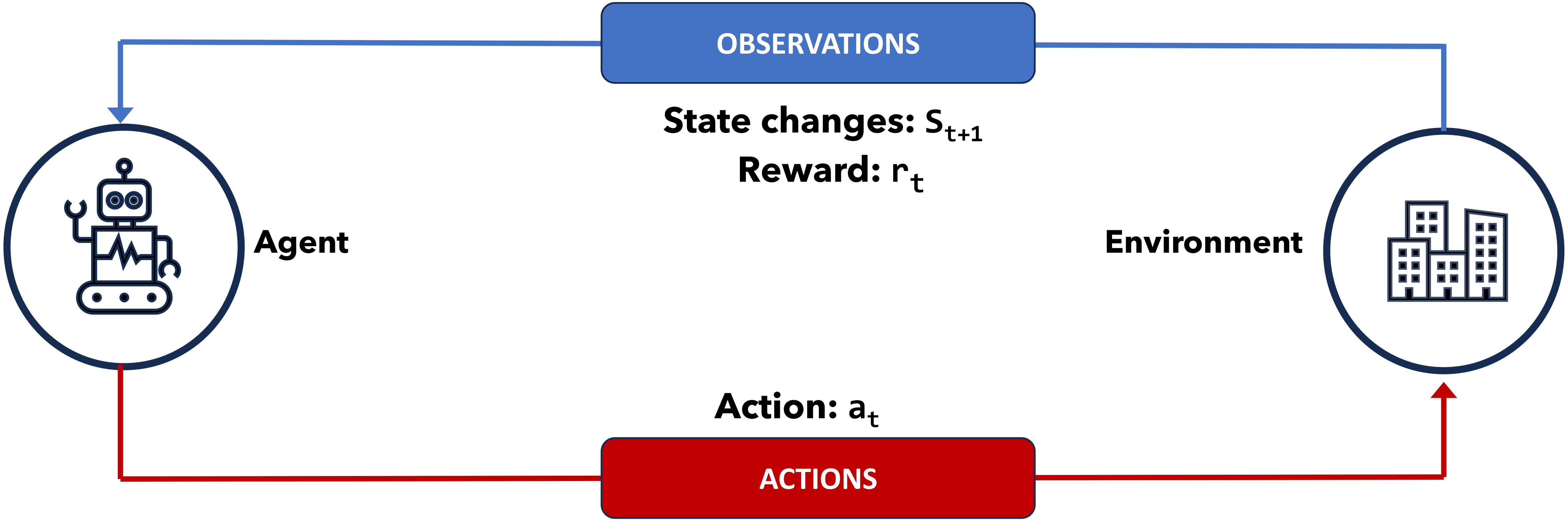
DEFINITION

TOTAL REWARD (Return)

Is the Summarization of the total rewards pending until the end of the movement in the universe

$$R_t = \sum_{i=t}^{\infty} r_i = r_t + r_{t+1} + \cdots + r_{t+n} + \cdots$$

Key Concepts: Discounted Total Reward



DEFINITION

Discount is a value between 0 and 1

$$R_t = \sum_{i=t}^{\infty} \gamma^{i} r_i$$

Exploration vs exploitation

TD(0) Prediction - Explotation-Exploration

Input, the policy π to be evaluated
    $\epsilon \leftarrow$ Exploration Rate
Initialize $V(s)$ arbitrarily (e.g. $V(s) = 0, \forall s \in \mathrm{S}^+$)
Repeat (for each episode):
    Initialize S
    Repeat (for each step of episode):

    if $random < \epsilon$ **then**
        $A \leftarrow$ Random possible action
    else
        $A \leftarrow$ action given by π for S

    $A \leftarrow$ action given by π for S
    Take action A; observe reward, R and next state S'
    $V(S) \leftarrow V(S) + \alpha[\mathrm{R} + \gamma V(S') - V(S)]$
    $S \leftarrow S'$

    $\epsilon \leftarrow$ decay

    until S is terminal

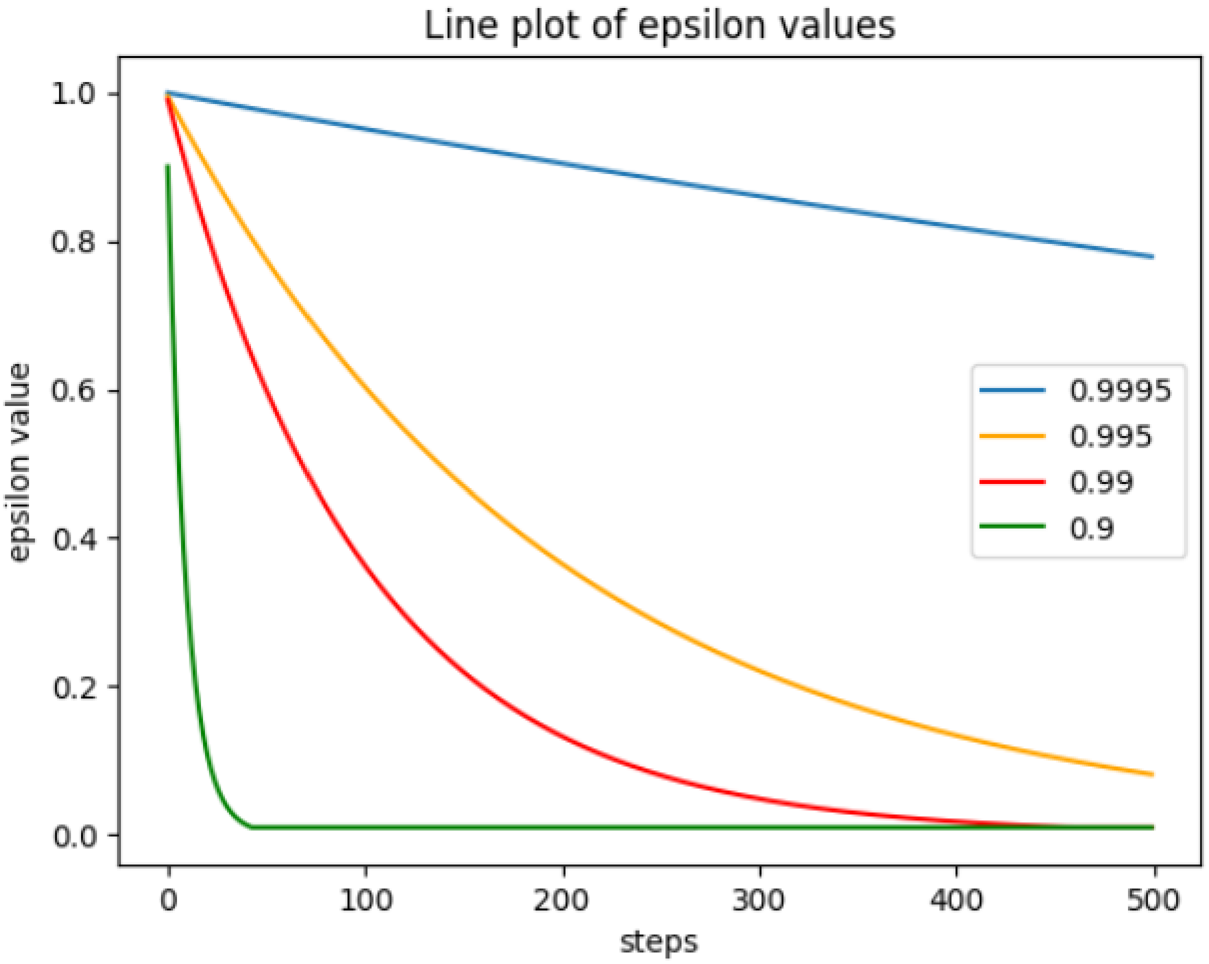
Exploration vs exploitation

- Simplest idea for ensuring continual exploration
- All m actions are tried with non-zero probability
- With probability $1 - \epsilon$ choose the greedy action
- With probability ϵ choose an action at random

$$\pi(a|s) = \begin{cases} \epsilon/m + 1 - \epsilon, & \text{if } a = \arg\max_{a' \in \mathcal{A}} Q(s, a') \\ \epsilon/m, & \text{otherwise} \end{cases}$$

where $m = |\mathcal{A}(s)|$

Epsilon is a sensible parameter



00_Epsilon_decay_visualization.ipynb

Policy vs Value learning

Policy Learning

- The policy is modeled and updated directly without consulting a value function.
- Policy gradient methods are commonly used to optimize the policy by estimating which direction improves returns

Value Learning

- In value-based RL, the focus is on learning the optimal value function, denoted Q* or V*. The value function estimates expected cumulative future rewards for being in a given state and following the current policy thereafter.
- Key notes on value-based methods:
- Finding the optimal value function allows deriving the optimal policy.
- Temporal-difference learning is commonly used to update value estimates.
- Algorithms Monte-Carlo, Q-Learning, SARSA, Actor-critic

Policy

"Right action for each state"

```
Policy Format (4x4):
---------------------
S → ↓ ←
↓ X ↓ X
→ ↓ ↓ X
X → → G
```

The Policy function tells us the best action to take on each state

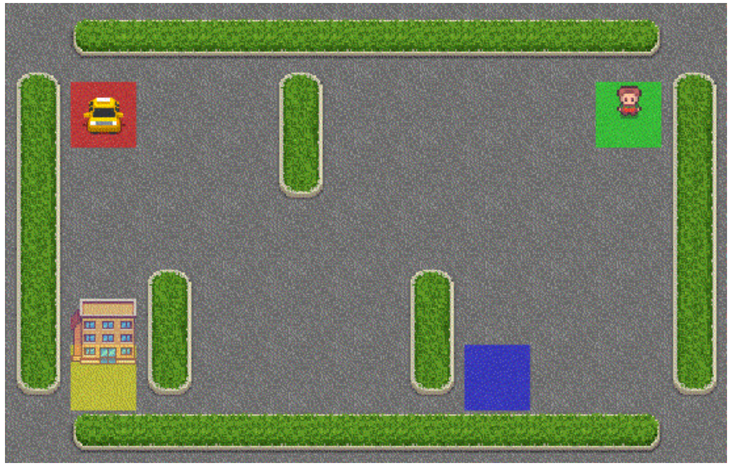
Value Function

```
Value Function Format (4x4):
------------------------------
  S   0.656 0.729 0.656
0.656   X   0.810   X
0.729 0.810 0.900   X
  X   0.900 1.000   G
```

The Value function gives us an estimate of value for each possible movement from our actual state

Environment examples

Action space : 4
Observation space: 16 – 4x4
Ice patches: (3,0), (1,1), (1, 3), (2,3)
Goal: (3,3)
Termination: Reaches goal



Action space : 6
Observation space: 500 (5x5) There are 500 discrete states since there are 25 taxi positions, 5 possible locations of the passenger (including the case when the passenger is in the taxi), and 4 destination locations.
Starting state: 300 starting states
Termination: drops passenger

```
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
```

Stochastic Environment

Definition

- A stochastic environment in Reinforcement Learning (RL) is one in which the outcome of an agent's action is not deterministic – that is, taking the same action in the same state may lead to different results with some probability.

Stochastic Effect: If you try to move **right**, there's a chance you'll slip **down** or **stay in place**.

If **is_slippery=True** the player will move in intended direction with probability of 1/3 else will move in either perpendicular direction with equal probability of 1/3 in both directions.

For example, if action is left and is_slippery is True, then:

- P(move left)=1/3
- P(move up)=1/3
- P(move down)=1/3

Solving Stochastic environments

- Agents must **learn expectations**, not absolute rules.
- Algorithms like SARSA or Q-learning estimate **expected value functions**

$$Q(s, a) = \mathrm{E}[\mathrm{R}]$$

- **Exploration** is crucial to properly model the stochasticity.
- In stochastic environments, we don't aim for perfect prediction — we aim for optimal expectation.

Real life examples of stochastic environments

🚙 Real-world Examples:

Scenario	Why It's Stochastic
Robot on slippery floor	Wheels may slip; direction changes unpredictably
Stock trading agent	Market responds probabilistically to actions
Game with dice (e.g., Monopoly)	Player's move depends on dice outcome
Medical treatment agent	Same treatment may affect patients differently

SARSA

State-action-reward-state-action

SARSA

- Updates the action-value function based on experiences in an environment
- **Action-value function is the Q(s,a) function**
- Requires a decaying exploration strategy (like gradually reducing ε in ε-greedy policies)

Definition

- SARSA is the acronym of State-action-reward-state-action
- Is a model-free on-policy method
- Idea: Maximize the cumulative reward obtained by the agent over time
- It is basically a TD algorithm on-policy
- The goal is to update the action-value function Q(s,a)
- Operates by estimating the Q-values iteratively based on the experiences
- The main update rule in the process is :

$$Q(s,a) \leftarrow Q(s,a) + \alpha[R + \gamma Q(s',a') - Q(s,a)]$$

Definition

SARSA: On-Policy TD Control

Initialize $Q(s,a), \forall s \in \mathrm{S}, a \in \mathrm{A}(s)$ arbitrarily and $Q(terminal\_state, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Choose A from S using policy derived from Q (e.g., ϵ-greedy)
  Repeat (for each step of episode):
    Take action A; observe reward, R and next state S'
    Choose A' from S' using policy derived from Q (e.g., ϵ-greedy)
    $Q(S,A) \leftarrow Q(S,A) + \alpha[\mathrm{R} + \gamma Q(S', A') - Q(S,A)]$
    $S \leftarrow S'; A \leftarrow A'$
  until S is terminal

- Uses the Q-value of the actual next action a′ chosen by the policy.
- This makes SARSA an on-policy algorithm because it updates the Q-values using the same policy that the agent follows during learning.

ε-greedy by definition

- The learning process in SARSA involves the agent interacting with the environment in a sequence of episodes.
- In each episode:
  - the agent starts in an initial state and selects actions based on its current policy (e.g., ε-greedy policy).
  - The ε-greedy policy balances exploration and exploitation by selecting the action with the highest estimated Q-value with probability 1-ε and selecting a random action with probability ε.
- This allows the agent to explore the environment and potentially discover better actions while still exploiting its current knowledge.

Why is SARSA on-policy

- Updates the values based the actions taken by the actual policy
- Continuously refines its policy
- The Q-values are intimately related to the previous experience of the agent and improve as the experience increases. The update to the policy is based on the experience
- Very important SARSA is on-policy while Q-learning is off-policy

Why is SARSA good?

- Fairly straightforward to implement
- Can handle discrete and continuous state and action spaces
- It converges most of the time

- But
  - Selection learning rate (α) and discount factor (γ) impacts algorithm efficiency and convergence
  - If learning rate is too high – maybe does not converge
  - If learning is too low – slow learning
  - Not very efficient for large action-state spaces

SARSA vs Q-learning

- SARSA shares similarities with Q-learning
- Both algorithms aim to learn the optimal action-value function and use similar update rules. However, the key difference lies in their learning strategies.
- While Q-learning is an off-policy algorithm that updates the Q-values based on the maximum expected future reward, regardless of the action taken, SARSA updates the Q-values based on the actual action taken by the current policy. This difference can lead to different learning dynamics and performance characteristics in certain problems.

SARSA vs Q-learning

https://www.youtube.com/watch?v=ma8MgPB0iXc

https://www.youtube.com/watch?v=vhuQLxvfIxg

26:28 – 27:..

Q-learning

Q-learning is off-policy

- While Q-learning is an off-policy algorithm that updates the Q-values based on the maximum expected future reward, regardless of the action taken, SARSA updates the Q-values based on the actual action taken by the current policy. This difference can lead to different learning dynamics and performance characteristics in certain scenarios.
- It is off-policy!!

$$Q(S,A) \leftarrow Q(S,A) + \alpha \left[R + \gamma \, max_a \, Q(S',a) - Q(S,A)\right]$$

The algorithm

Q-learning: Off-policy TD control

Initialize $Q(s,a), \forall s \in \mathrm{S}, a \in \mathrm{A}(s)$ arbitrarily and $Q(terminal\_state, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q (e.g., ϵ-greedy)
    Take action A observe R and next state S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha[\mathrm{R} + \gamma \max_{a'} Q(S', a') - Q(S,A)]$
    $S \leftarrow S'$;
  until S is terminal

- Uses the maximum possible future *Q-value* for the next state s ' , regardless of the agent's actual next action

- This makes Q-Learning an off-policy algorithm because it learns from the best possible actions rather than the actions actually taken by the current policy.

See the Differences

Uses Policy to choose next action

Update slightly different

SARSA: On-Policy TD Control

Initialize $Q(s,a), \forall s \in S, a \in A(s)$ arbitrarily and $Q(terminal\_state, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Choose A from S using policy derived from Q (e.g., ϵ-greedy)
  Repeat (for each step of episode):
    Take action A; observe reward, R and next state S'
    Choose A' from S' using policy derived from Q (e.g., ϵ-greedy)
    $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma Q(S',A') - Q(S,A)]$
    $S \leftarrow S'; A \leftarrow A'$
  until S is terminal

Q-learning: Off-policy TD control

Initialize $Q(s,a), \forall s \in S, a \in A(s)$ arbitrarily and $Q(terminal\_state, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q (e.g., ϵ-greedy)
    Take action A observe R and next state S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_{a'} Q(S',a') - Q(S,A)]$
    $S \leftarrow S'$;
  until S is terminal

1. The most important difference between the two is how Q is updated after each action. SARSA uses the ***Q'*** following a ε-greedy policy exactly, as A' is drawn from it. In contrast, Q-learning uses the maximum ***Q'*** over all possible actions ***a'*** for the next step. This makes it look like following a greedy policy with ε=0, i.e. NO exploration in this part.
2. However, when actually taking an action, Q-learning still uses the action taken from a ε-greedy policy. This is why "Choose A ..." is inside the repeat loop.
3. Following the loop logic in Q-learning, A' is still from the ε-greedy policy.

See the Differences

	SARSA	Q-learning
Choosing A'	π	π
Updating Q	π	μ

Where π is a ε-greedy policy (e.g. ε > 0 with exploration), and μ is a greedy policy (e.g. ε == 0, NO exploration).

1. Given that Q-learning is using different policies for choosing next action A' and updating Q. In other words, it is trying to evaluate π while following another policy μ, so it's an off-policy algorithm.

2. In contrast, SARSA uses π all the time, hence it is an on-policy algorithm.

https://stackoverflow.com/questions/6848828/what-is-the-difference-between-q-learning-and-sarsa

Example

See the Differences

```
# Q-Learning algorithm - Commens changes from SARSA example
for episode in range(episodes):
    state, _ = env.reset()
    done = False
    while not done:
        action = choose_action(state)  # Action selection moved here for Q-learning

        next_state, reward, done, _, _ = env.step(action)

        # Q-learning update
        # The key difference is using max Q-value of next state instead of Q-value of next action
        Q[state, action] += alpha * (reward + gamma * np.max(Q[next_state, :]) - Q[state, action])

        state = next_state
        # Removed: action = next_action (Q-learning is off-policy, so we don't need to track the next action)
```

```
# SARSA algorithm
for episode in range(episodes):
    state, _ = env.reset()
    action = choose_action(state)
    done = False
    while not done:

        next_state, reward, done, _ , _ = env.step(action)

        next_action = choose_action(next_state)

        # SARSA update
        Q[state, action] += alpha * (reward + gamma * Q[next_state, next_action] - Q[state, action])

        state = next_state
        action = next_action
```

Exploration vs Exlotaition

```
def choose_action(state):
    if np.random.uniform(0, 1) < epsilon:
        return env.action_space.sample()  # Explore
    else:
        return np.argmax(Q[state, :])  # Exploit
```

SARSA & Q-Learning

Policy Type

- SARSA is an on-policy algorithm: learns Q-values based on actions actually taken using current policy
- Q-Learning is off-policy: learns about optimal policy regardless of actions taken

Action Selection for Updates

- SARSA considers the next action that will actually be taken (including exploration)
- Q-Learning assumes optimal future actions regardless of exploration policy, using max Q-value

Convergence Properties

- SARSA converges to the optimal policy when using an epsilon-greedy policy with epsilon decreasing to zero
- Q-Learning converges to the optimal Q-function regardless of the exploration policy (as long as all state-action pairs are visited infinitely often)

Safety and Conservative Behavior

- SARSA tends to learn safer paths as it considers exploration in its value estimates
- Q-Learning can learn optimal paths that might be risky during training (because it assumes optimal future actions)

Exploration-Exploitation Trade-off

- SARSA directly incorporates exploration into its learning, making it more conservative
- Q-Learning separates exploration policy from learning, potentially leading to more optimal but riskier policies Example scenario:

SARSA vs Q-learning

https://www.youtube.com/watch?v=ma8MgPB0iXc

https://www.youtube.com/watch?v=vhuQLxvfIxg

26:28 – 27:..

Wrap-up

Summary differences SARSA and Q-Learning

Aspect	Q-Learning	SARSA
Policy Type	Off-policy	On-policy
Next Q-value Estimation	Uses max Q(s',a') (greedy action)	Uses Q(s', a') of the actual action taken
Exploration vs. Exploitation	Encourages more exploitation (greedy updates)	Encourages more exploration (policy-following updates)
Risk Sensitivity	More aggressive, assumes optimal future actions	More conservative, considers the actual exploration strategy
Convergence Speed	Typically, faster but less stable in stochastic environments	More stable but slower

END

Session 7

UNIVERSITY

SCHOOL OF
SCIENCE &
TECHNOLOGY