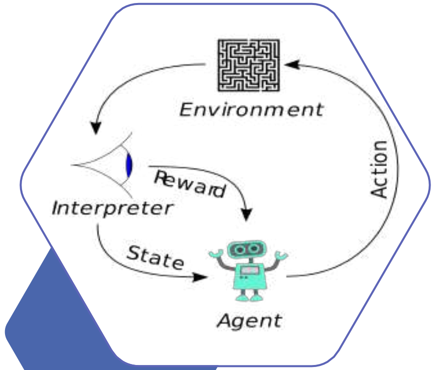
Reinforcement Learning and Markov Decision Processes

Georgia Koutsandria

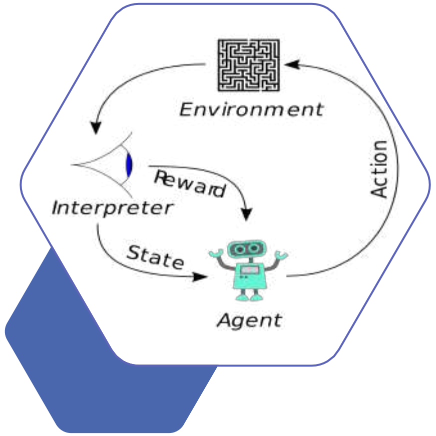
Internet of Things A.Y. 18-19

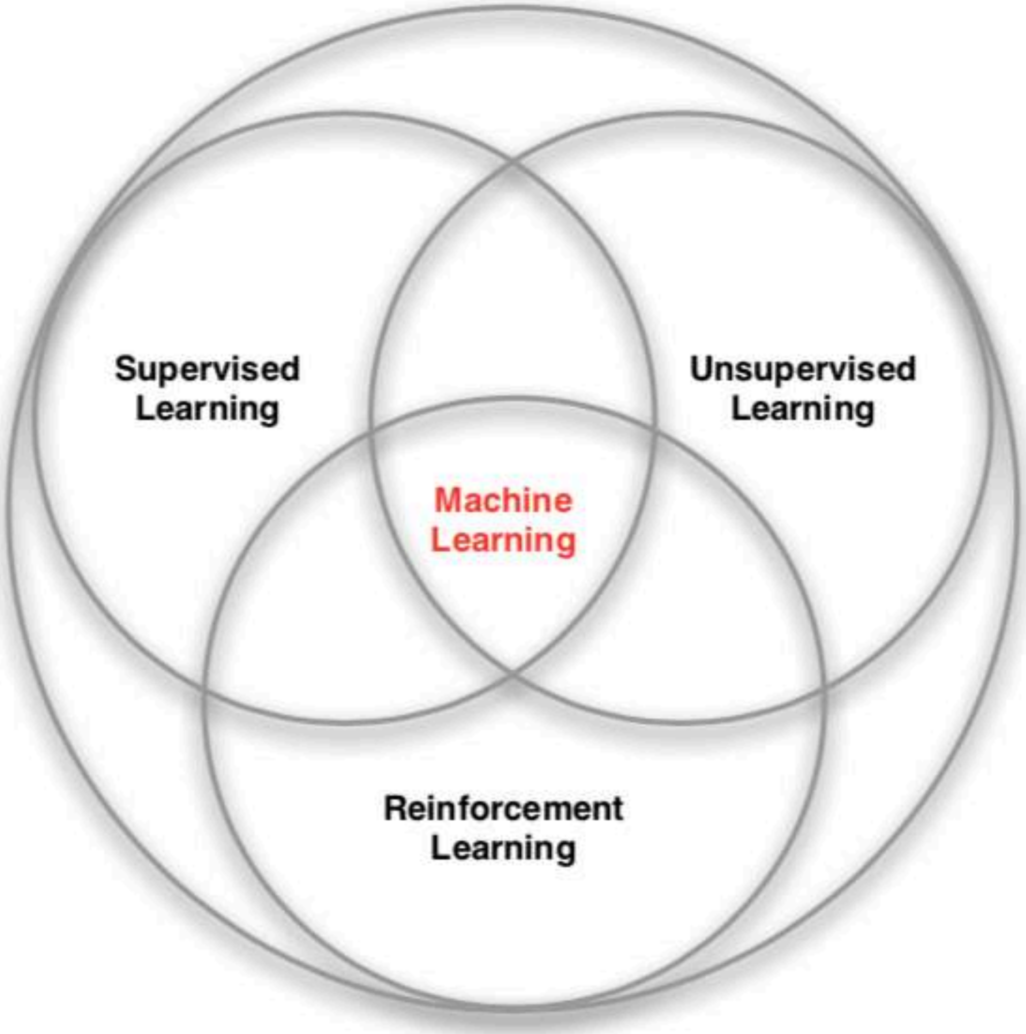
Prof. Chiara Petrioli
Dept. of Computer Science
Sapienza University of Rome



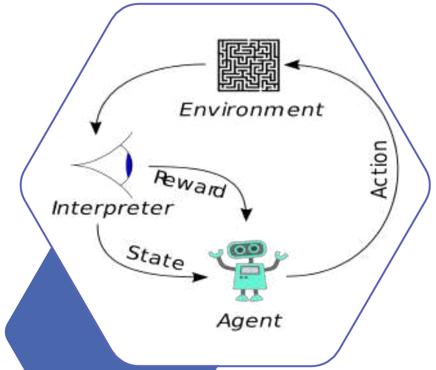
Reinforcement Learning (RL)

Branches of Machine Learning





Branches of Machine Learning

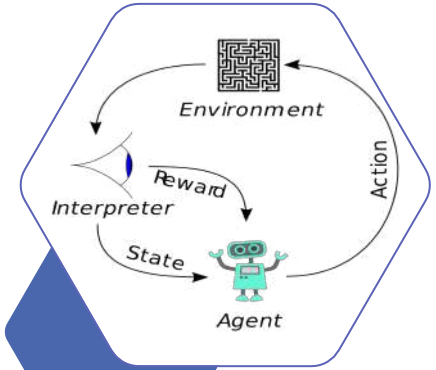


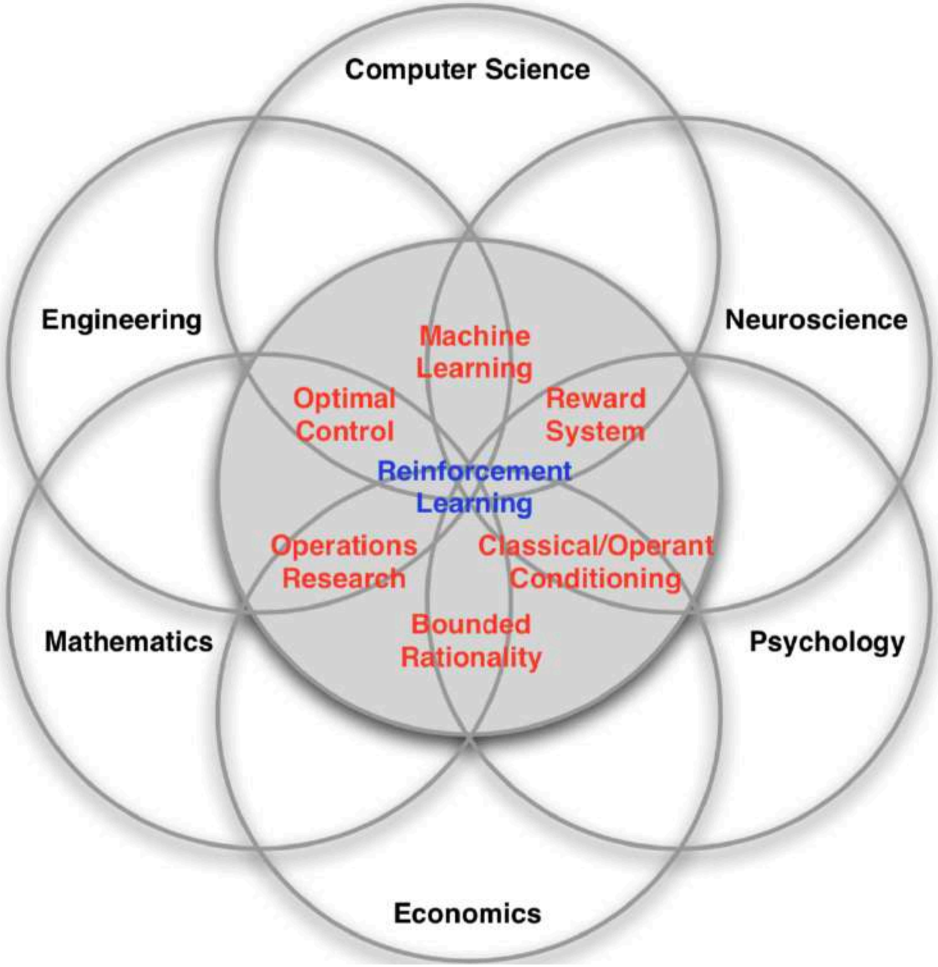
- **Supervised learning:** Learning from a training set of labeled data provided by a knowledgable external supervisor.

- **Unsupervised learning**: Learning the inherent structure of data without the use of explicitly-provided labels.

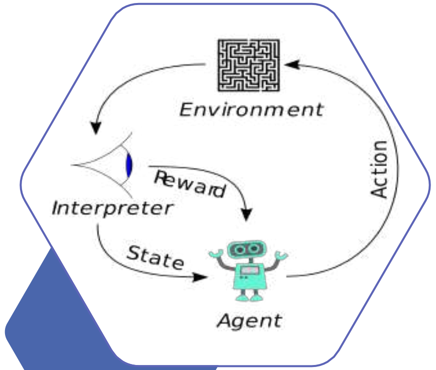
- **Reinforcement learning**: Learning what to do—how to map situations to actions—so as to maximize a numerical reward signal.

Many faces of RL



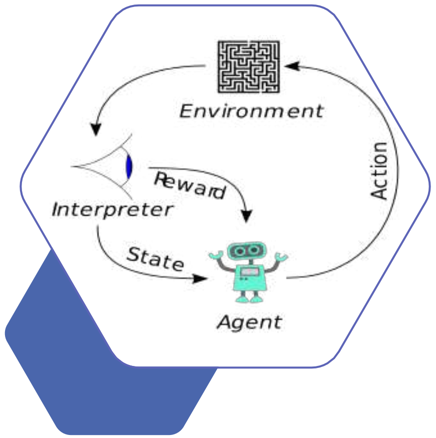


Machine vs. Reinforcement Learning



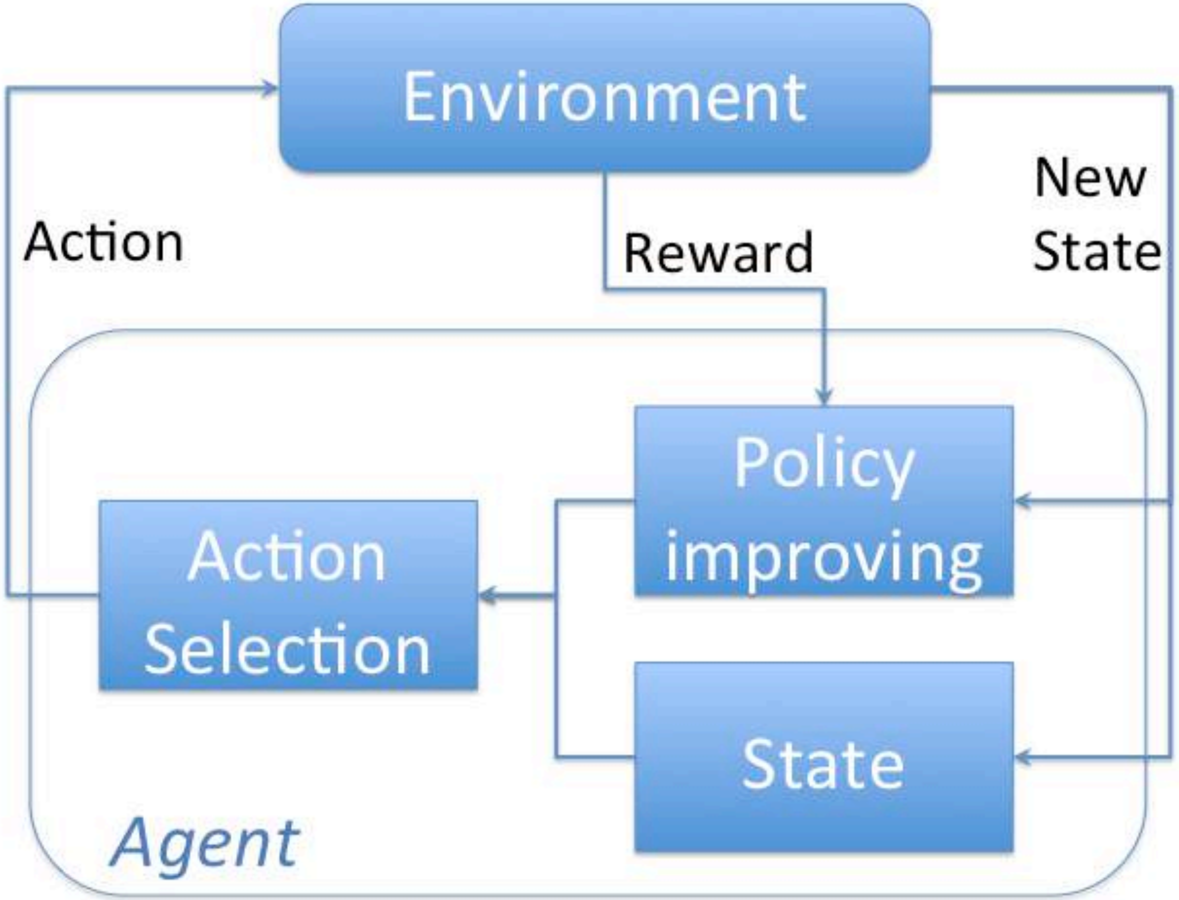
- There is no supervisor, only a *reward* signal
- No instantaneous feedback (delayed)
- Time really matters (sequential, non i.i.d. data)
- Agent’s actions affect the subsequent data it receives.

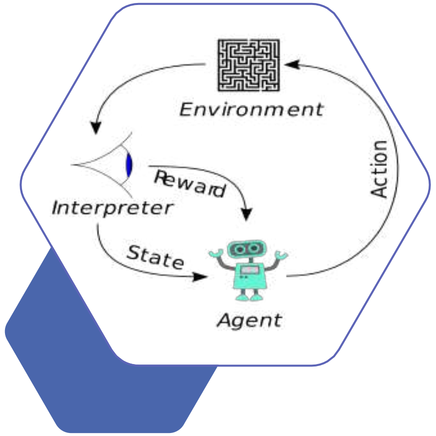
Historical Background



- Original motivation: animal learning
- Early emphasis: neural net implementations and heuristic properties
- Operations research, optimal control theory, dynamic programming, AI state-space research
- Best formalized as a set of techniques to handle: MDPs or P(artially)O(bservable)MDPs

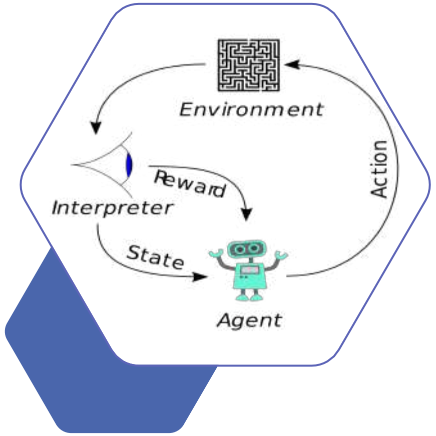
RL Task


Environment
Action
Reward
New State
Policy improving
Action Selection
State
Agent


Environment
Reward
Interpreter
Action
State
Agent

RL Task

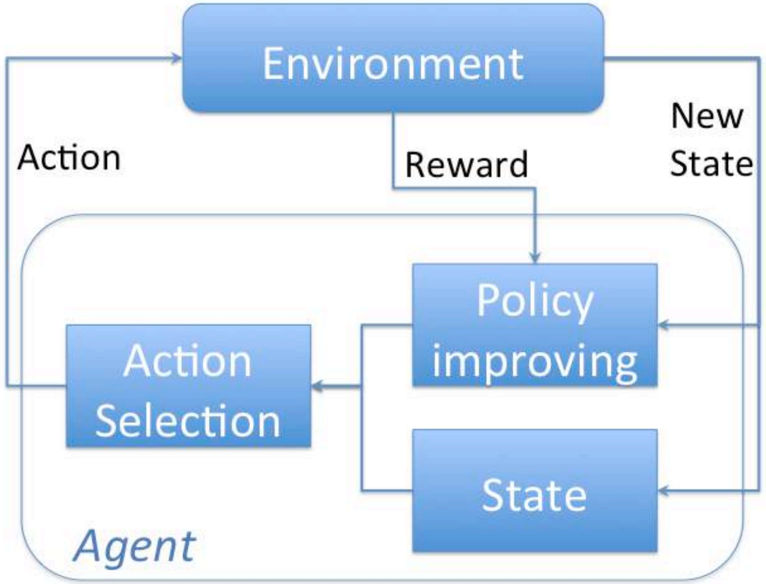
$$s(0) \xrightarrow[r(1)]{a(1)} s(1) \xrightarrow[r(2)]{a(2)} s(2) \xrightarrow[r(3)]{a(3)} s(3)\ldots$$



Goal: learn to choose actions that maximize the cumulative reward

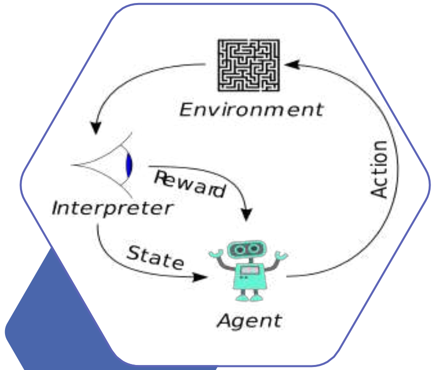
$$r(0) + \gamma r(1) + \gamma r(2) + \gamma r(3) + \ldots$$

where $0 \leq \gamma < 1$



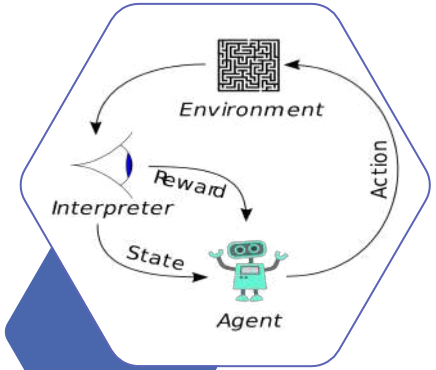
Reinforcement Learning

Elements of an RL agent



- Policy: Defines the agent’s way of behaving
- Reward signal: Defines the goal of the RL problem
- Value function: Specifies what is good in the long run
- Model: Mimics the behavior of the environment.

Foresigthed Optimization

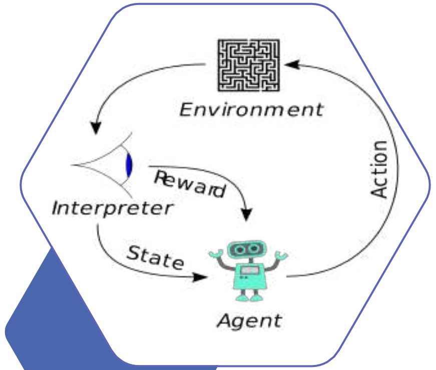


- Key feature: Actions affect **immediate** and **future** system performance
  - System optimazation “on the long run”.

- When actions/decisions affect immediate and future performance, myopic heuristic solutions are suboptimal because they ignore the expected future utility.

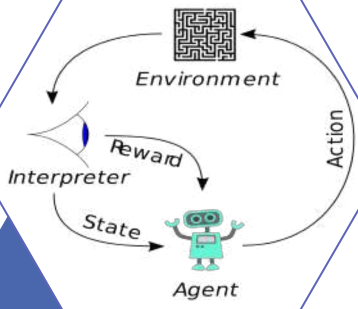
  - Dramatic improvements can be achieved using long term optimization.

Reinforcement Learning



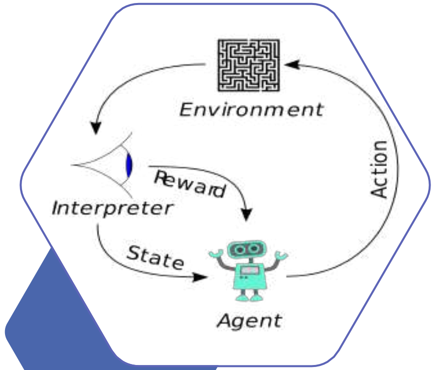
Markov Decision Processes (MDP)

Introduction to MDPs



- MDPs formally describe an environment for RL.
- The environment is fully observable, i.e., the current state completely characterizes the process.
- Almost all RL problems can be formalized as MDPs.

Markov Decision Processes



- Finite set of states S
- Finite set of actions $A(s), s \in S$
- Immediate reward function

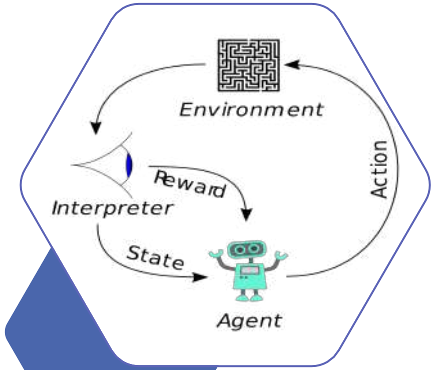
$$R: S \times A \rightarrow R$$

- Transition (next-state) function

$$T: S \times A \rightarrow S$$

- More generally, R and T are treated as stochastic
- Our focus: discrete time MDPs.

Markov Decision Processes

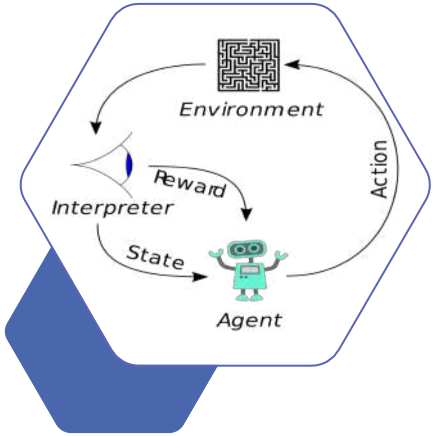


- Markov Property for MDPs

$$P(s' \mid s, a)$$
$$s', s \in S, \qquad a \in A(s)$$

- Next state is a function of current <u>*state*</u> and the <u>*action taken*</u>!

Markov Decision Processes



- If no rewards and only one action, this is just a Markov chain(or Controlled Markov Chain)

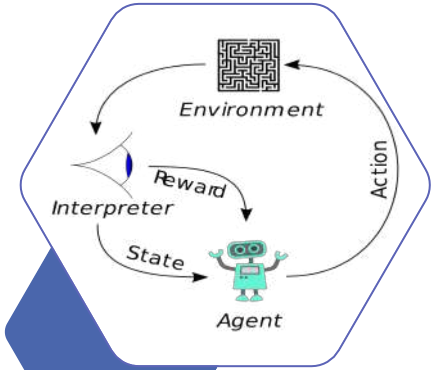
- Overall objective is to determine a policy

$$\pi : S \rightarrow A$$

such that some measure of cumulative reward is optimized

E.g, $r(0) + \gamma r(1) + \gamma r(2) + \gamma r(3) + \ldots$

What is a policy?

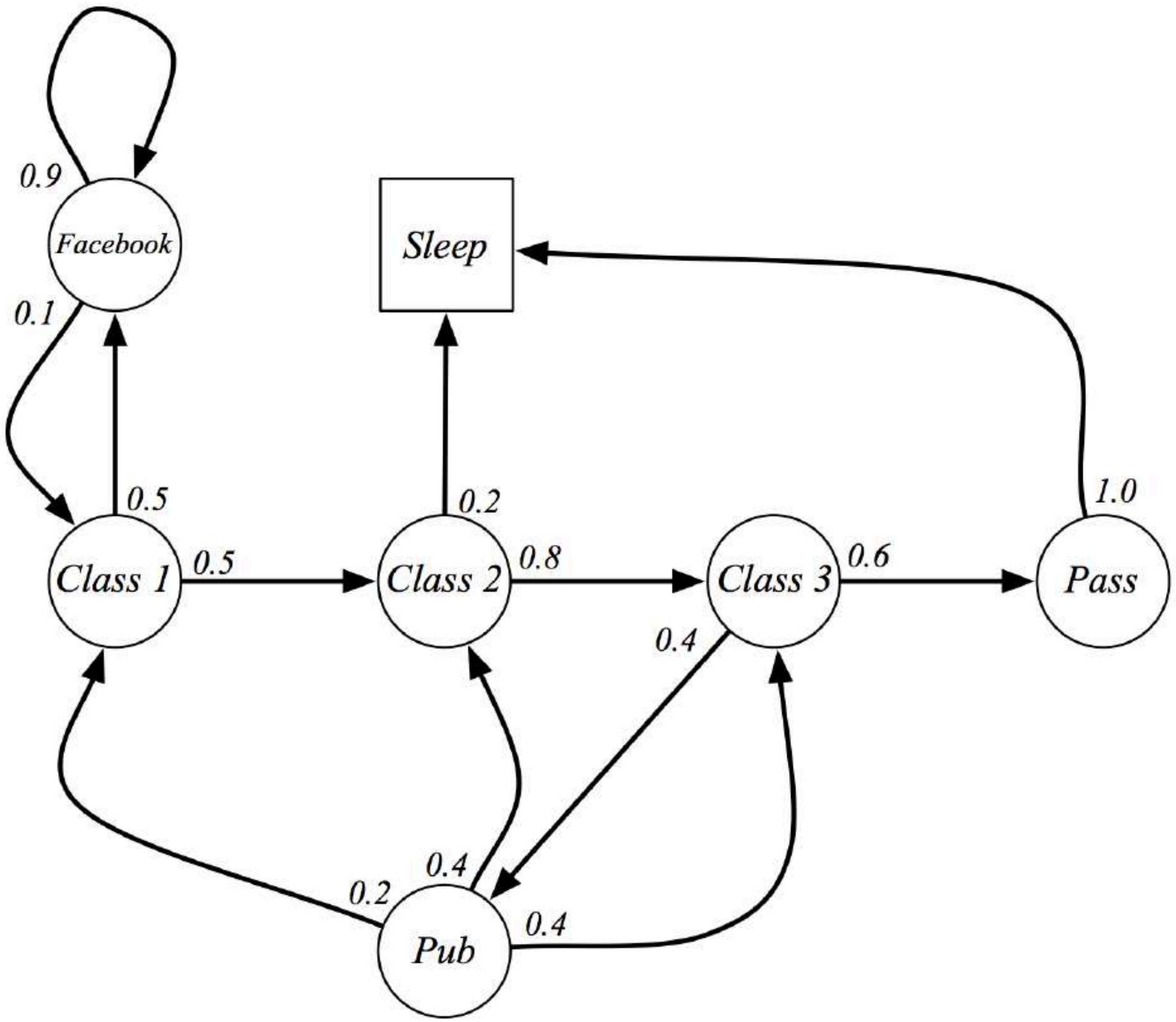


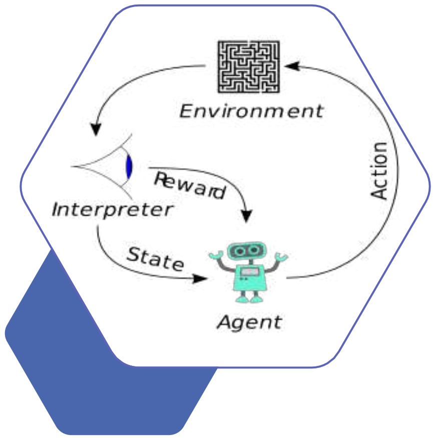
- A **policy** is the agent's behavior

- It is a map from state to action.

If agent is in state	Then a good action is
s_1	a_3
s_2	a_7
s_3	a_1
s_4	a_3
…	…

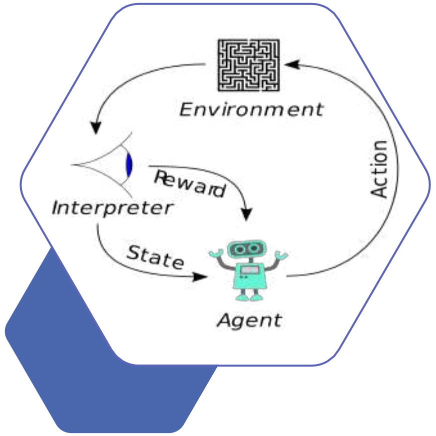
Student Markov Chain

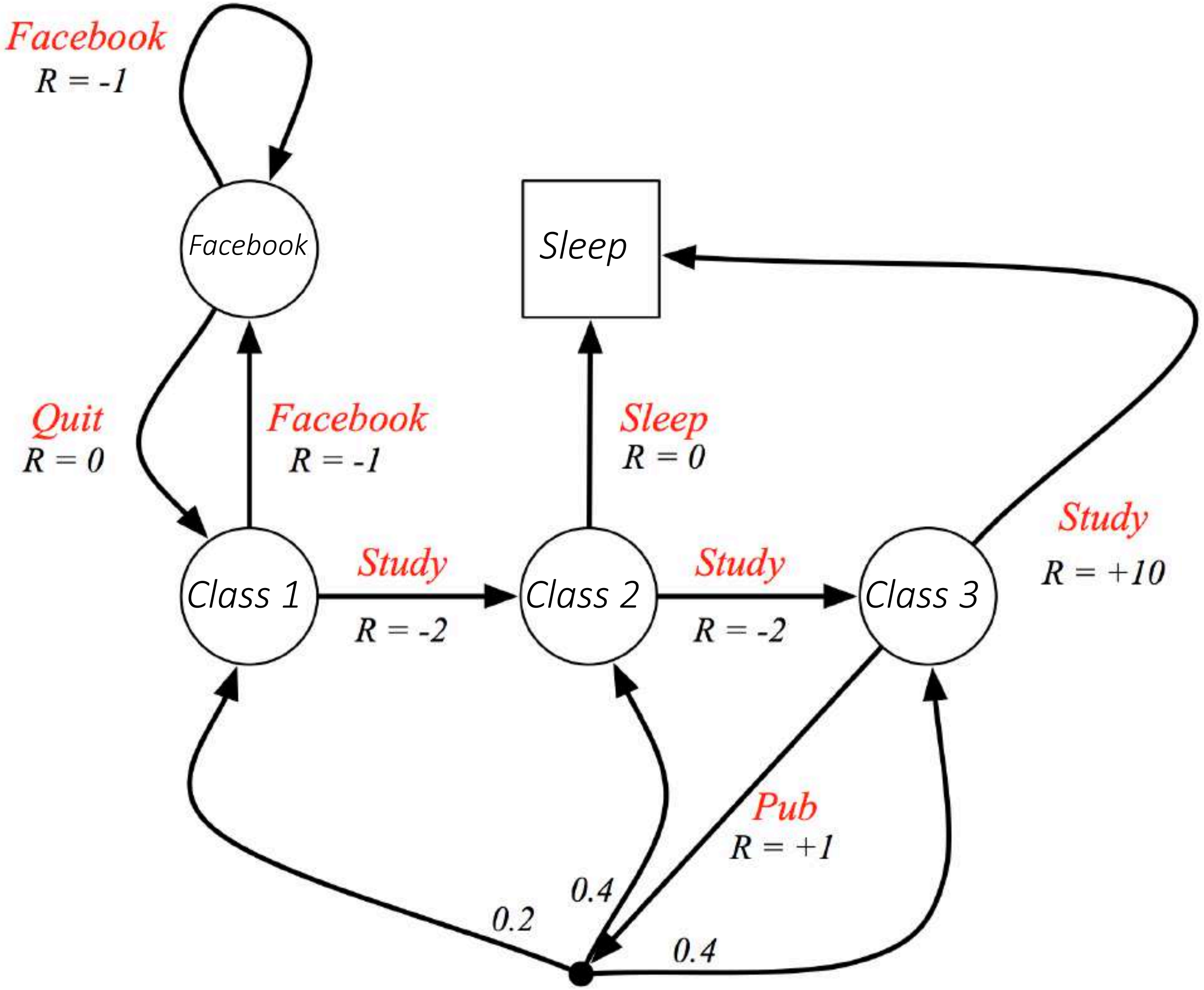


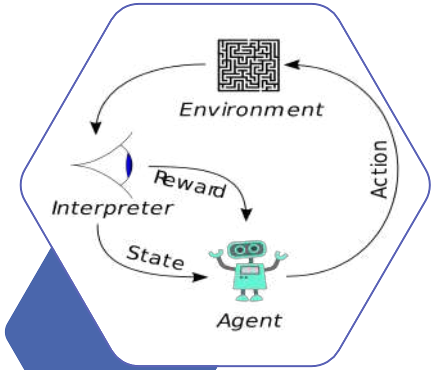


Markov Decision Process

Student Markov Decision Process



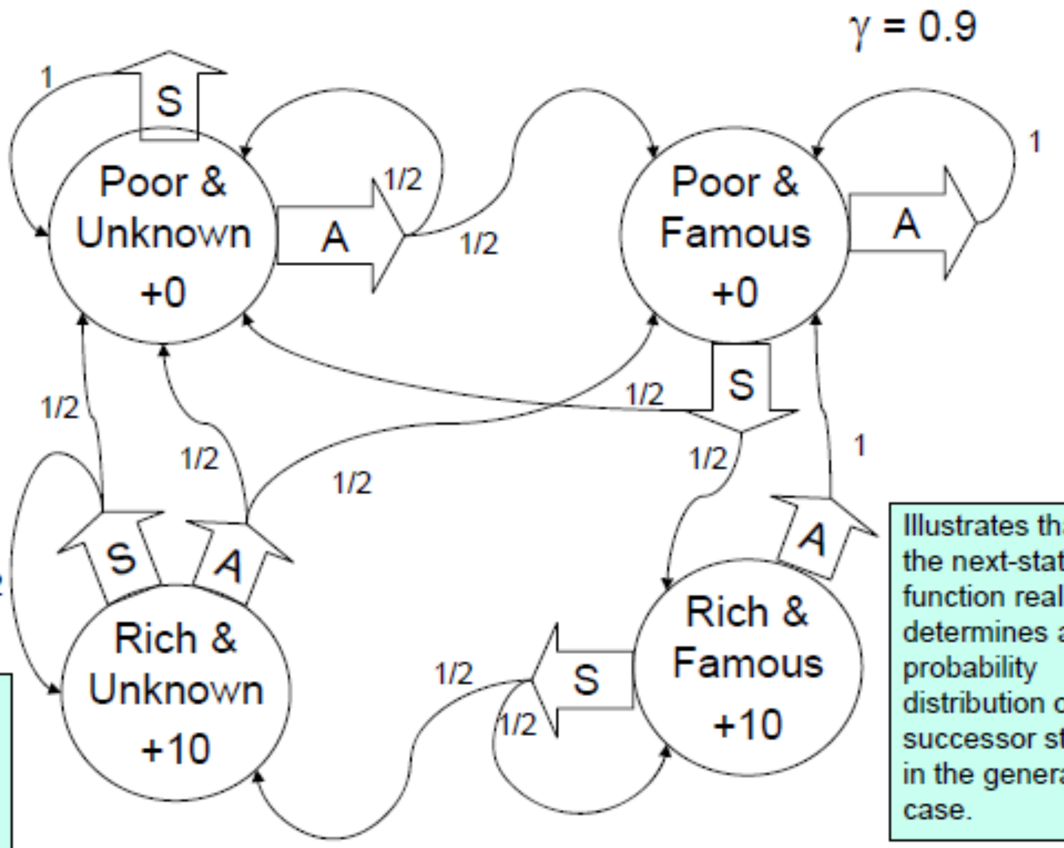




A Markov Decision Process

You run a startup company.

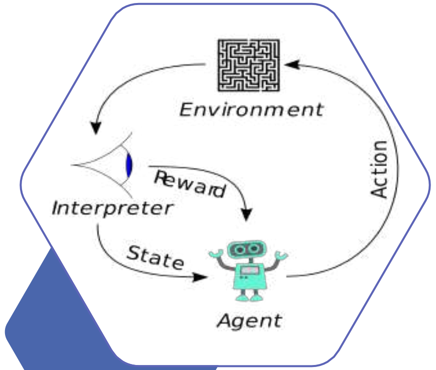
In every state you must choose between Saving money or Advertising.

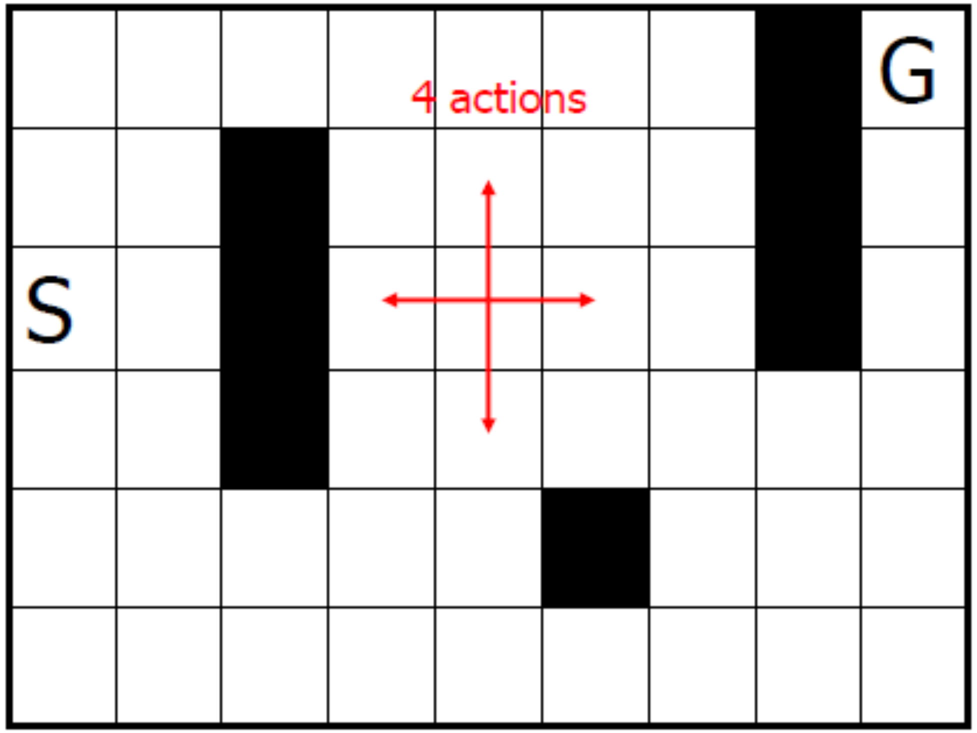


Here the reward shown inside any state represents the reward received upon entering that state.

Illustrates that the next-state function really determines a probability distribution over successor states in the general case.

Another MDP





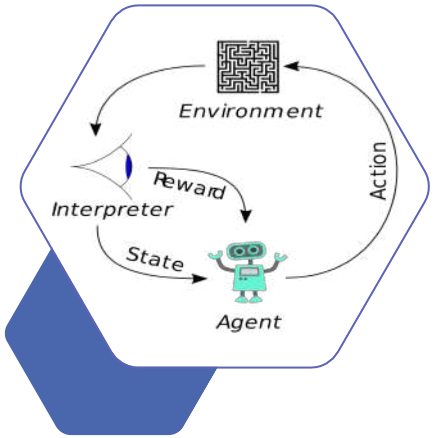
47 states

Reward = -1 at every step $\gamma = 1$

G is an absorbing state, terminating any single trial, with a reward of 100

Effect of actions is deterministic

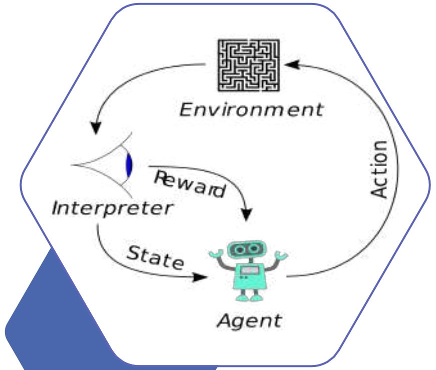
Applications of MDPs



- Many important problems are MDPs
  - Robot path planning
  - Travel route planning
  - Elevator scheduling
  - Autonomous aircraft navigation
  - Manufacturing processes
  - Network switching & routing
  - …

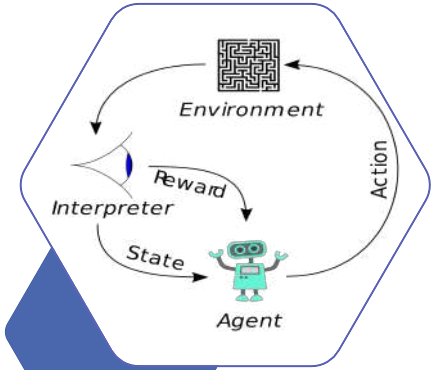
- Many of these have been successfully handled using RL methods.

Brief Summary of Concepts



- The *agent* and its *environment* interact over a sequence of discrete time steps
- The specification of their interface defines a particular task:
  - the *actions* are the choices made by the agent
  - the *states* are the basis for making the choices
  - the *rewards* are the basis for evaluating the choices
- A *policy* is a stochastic rule by which the agent selects actions as a function of states
  - The agent's objective is to maximize the amount of reward it receives over time.

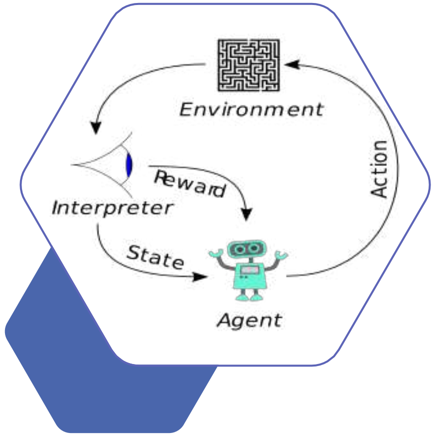
Value Function



- For any policy π, define the Value function as the function $V^{\pi}: S \rightarrow \mathbb{R}$ assigning to each state the quantity
$$V^{\pi}(s) = \sum_{t=0}^{\infty} \gamma^{t} r(t),$$
where $s(0) = s$
- Each action $a(t)$ is chosen according to policy π
- Each subsequent $s(t+1)$ arises from the transition function T
- Each immediate reward $r(t)$ is determined by the immediate reward function R
- γ is a given discount factor in $[0, 1]$

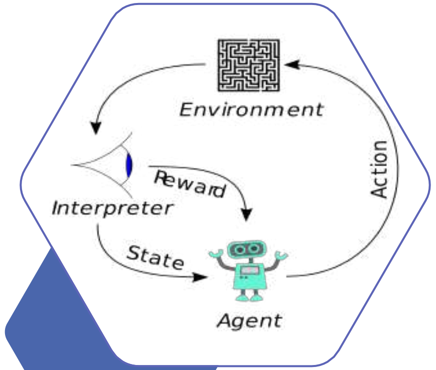
Reminder: Use expected values in the stochastic case.

Discount factor γ



- Models idea: Future rewards are not worth as much as immediate rewards
  - Used in economic models
  - Uncertainty about the future
- Models situations where there is a nonzero fixed probability $1 - \gamma$ of termination at any time
- Tradeoff between myopic ($\gamma = 0$) vs. foresighted optimization (γ close to 1)
- …and makes the math work out nicely with bounded rewards, sum guaranteed to be finite even in infinite-horizon case.

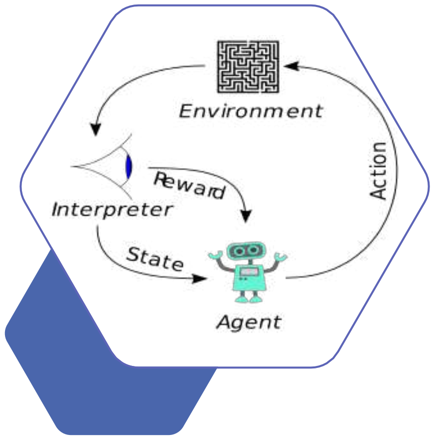
Technical Remarks



- If the next state and/or immediate reward functions are stochastic, then the $r(t)$ values are random variables and the return is defined as the expectation of this sum.

- If the MDP has absorbing states, the sum may actually be finite
  - In that case $\gamma = 1$ is allowed, i.e., no discount
  - We stick with this infinite sum notation for the sake of generality
  - The formulation we use is called infinite-horizon.

Optimal Policies



- **Objective**: Find a policy π^* such that

$$V^{\pi^*}(s) \geq V^{\pi}(s)$$

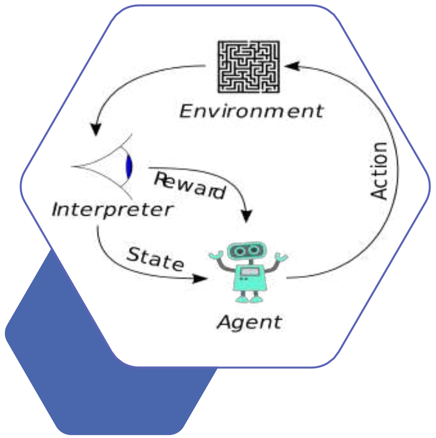
for any policy π and any state s

- Such a policy is called an optimal policy

We define:

$$V^* = V^{\pi^*}$$

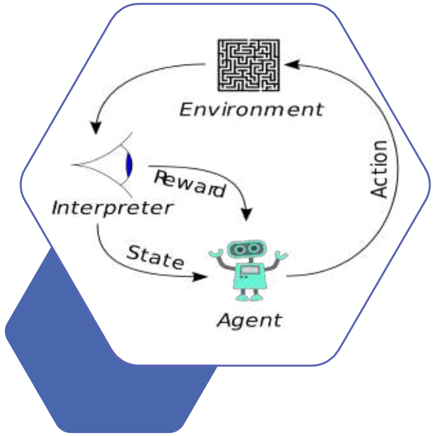
Interesting Facts



- For every MDP such that S is discrete and $A(s)$ is finite there exists an optimal policy
  - This theorem can be easily extended to the case in which $A(s)$ is a *compact* set.

- It is a policy such that for every possible start state there is no better option than to follow the policy.

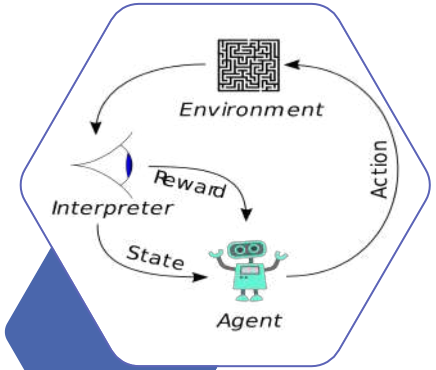
Finding an Optimal Policy



- Idea:
    1. Run through all possible policies.
    2. Select the best.

What's the problem ??

Finding an Optimal Policy

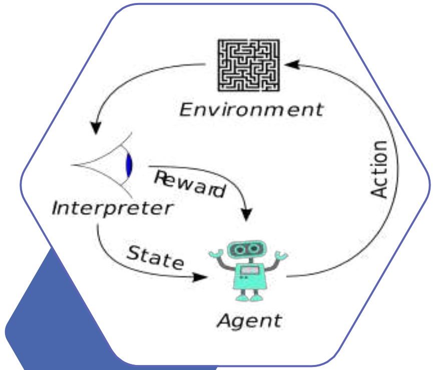


- Dynamic Programming approach:
  Determine the optimal value function for each state
  Select actions according to this optimal value function V^*

- How do we compute V^*?
  - Magic words: Bellman equation(s)

Reinforcement Learning

Derivation of the Bellman Equation



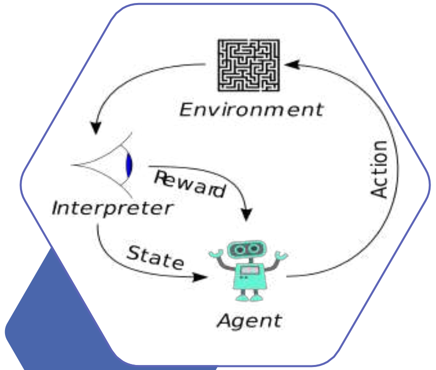
- Given the state transition $s \rightarrow s'$

$$V^{\pi}(s) = \sum_{t=0}^{\infty} \gamma^t r(t)$$

$$= r(0) + \gamma \sum_{t=0}^{\infty} \gamma^t r(t+1)$$

$$= r(0) + \gamma V^{\pi}(s')$$

Bellman Equations



- For any state s and policy π

$$V^{\pi}(s) = R(s, \pi(s)) + \gamma V^{\pi}(T(s, \pi(s)))$$

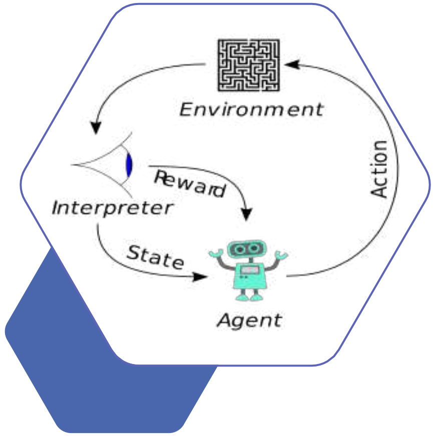
- For any state s, the optimal value function is

$$V^{*}(s) = \max_{a}\{R(s,a) + \gamma V^{*}(T(s,a))\}$$

- Recurrence relations
  - Can be used to compute the return from a given policy or to compute the optimal return via value iteration.

Reinforcement Learning

Bellman Equations: General Form

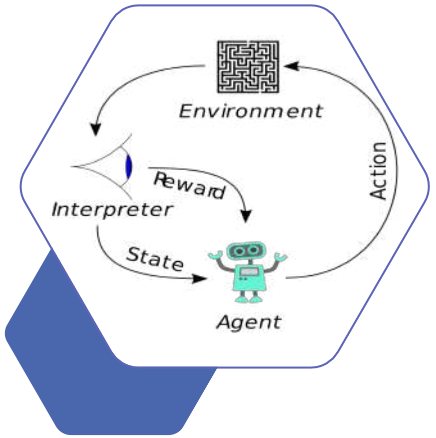


- For completeness, here are the Bellman equations for stochastic and discrete time MDPs:

$$V^{\pi}(s) = R(s, \pi(s)) + \gamma \sum_{s'} P_{ss'}(\pi(s)) V^{\pi}(s')$$
$$V^{*}(s) = \max_{a} \{R(s,a) + \gamma \sum_{s'} P_{ss'}(a) V^{*}(s')\}$$

where $R(s,a)$ now represents $E(R \mid s,a)$ and $P_{ss'}(a)$ is the probability that the next state is s' given that action a is taken in state s

From Values to Policies



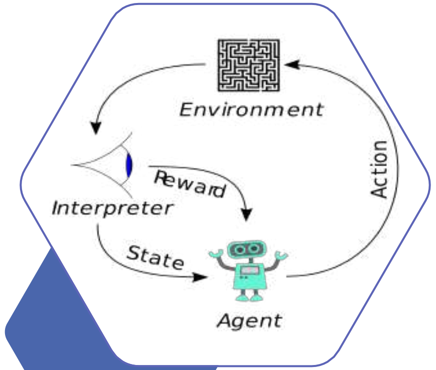
- Given the optimal value function V^* it follows from Bellman equation that the optimal policy can be computed as:

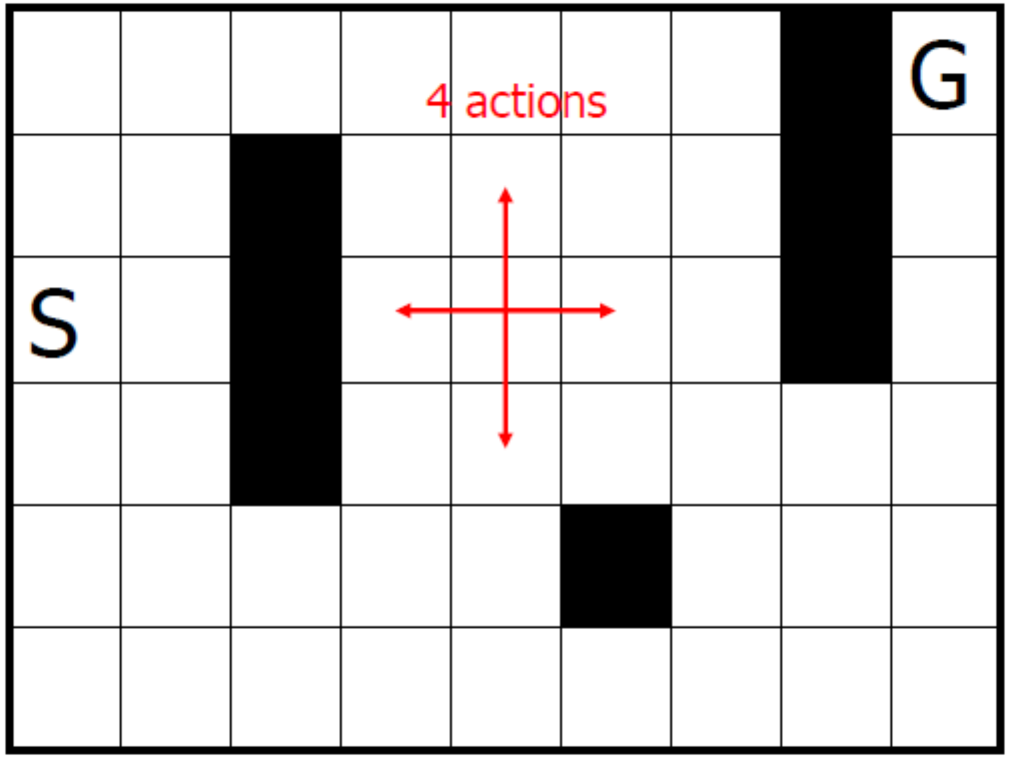
$$\pi(s) = \underset{a}{\operatorname{argmax}}\{R(s,a) + \gamma V^*(s')\}$$

- An optimal policy is said to be greedy for V^*

- If π is not optimal then a greedy policy for V^π will yield a larger return than π
  - Not hard to prove
  - Basis for another DP approach to finding optimal policies: policy iteration.

An example: Maze task



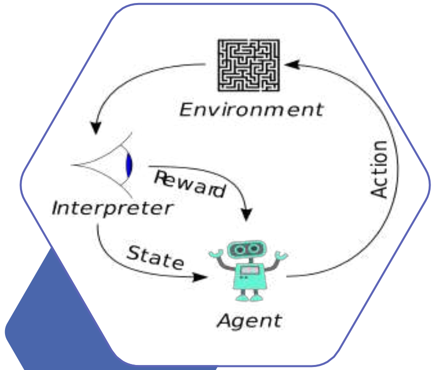


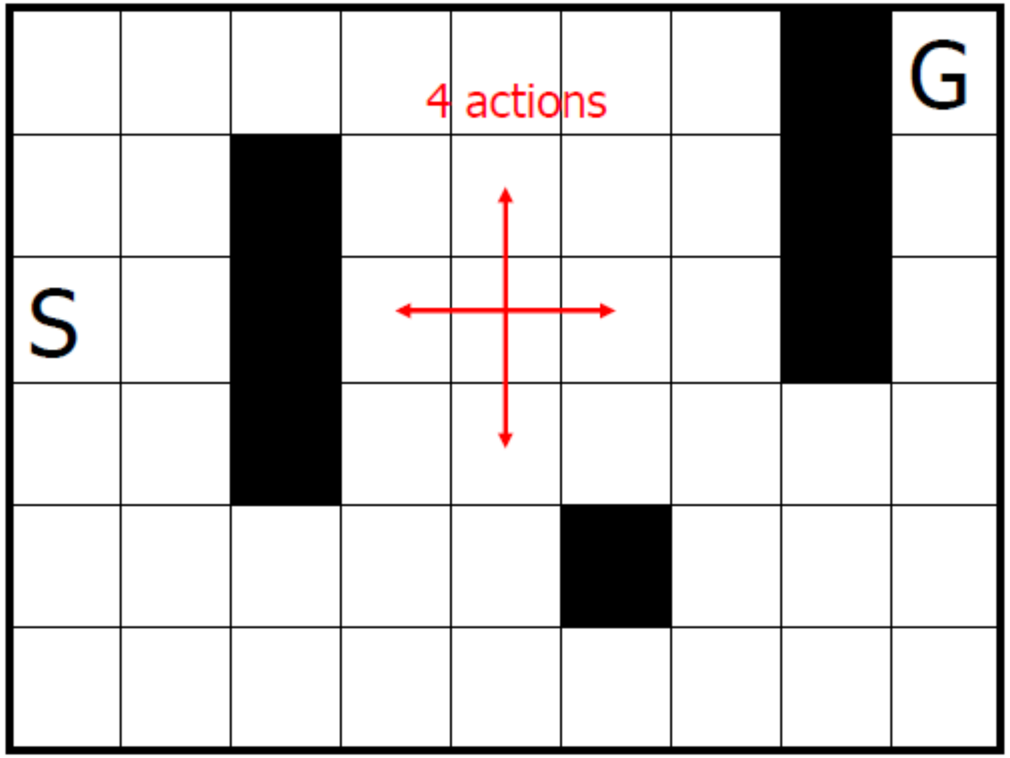
Reward = -1 at every step $\gamma = 1$

G is an absorbing state, terminating any single trial, with a reward of 100

Effect of actions is deterministic

An example: Maze task





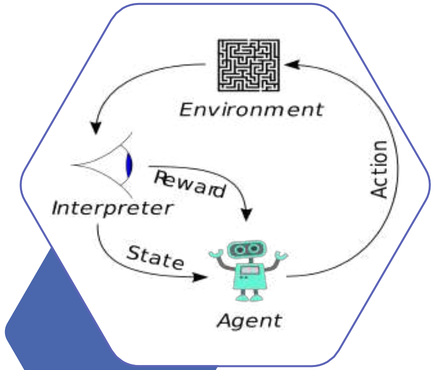
Reward = -1 at every step $\gamma = 1$

G is an absorbing state, terminating any single trial, with a reward of 100

Effect of actions is deterministic

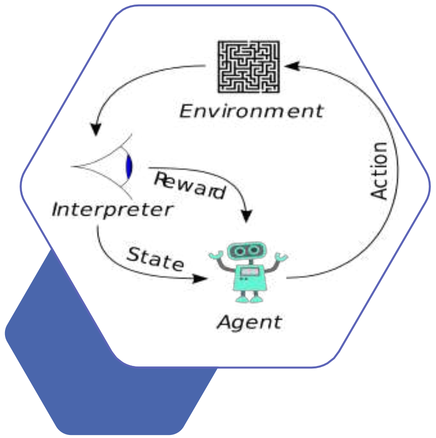
How would you model the MDP?

Maze task: MDP Model



- State is a pair: $s = (x, y), \;\; x, y \in \{1, \ldots, 9\}$ defining the robot's position.
- Actions: $A(s) = \{\mathrm{up}, \mathrm{down}, \mathrm{left}, \mathrm{rig}ht\}$
  (except for those states near the black squares)
- Reward Function: $R(s) = \begin{cases} -1, & \forall s \neq G \\ 100, & \text{if } s = G \end{cases}$
- Transition function: $s' = \begin{cases} (x+1, y) & \text{if } a = \mathrm{right} \\ (x-1, y) & \text{if } a = \mathrm{left} \\ (x, y+1) & \text{if } a = \mathrm{up} \\ (x, y-1) & \text{if } a = \mathrm{down} \end{cases}$

Maze task: Value Function



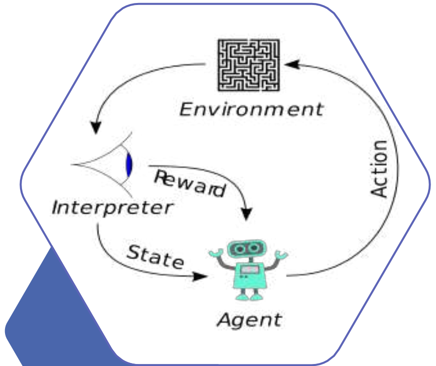
	86	87	88	89	90	91	92		100	G
	85	86		90	91	92	93		99	
S	86	87		91	92	93	94		98	
	87	88		92	93	94	95	96	97	
	88	89	90	91	92		94	95	96	
	87	88	89	90	91	92	93	94	95	

V^*

What's an optimal path from S to G?

Reinforcement Learning

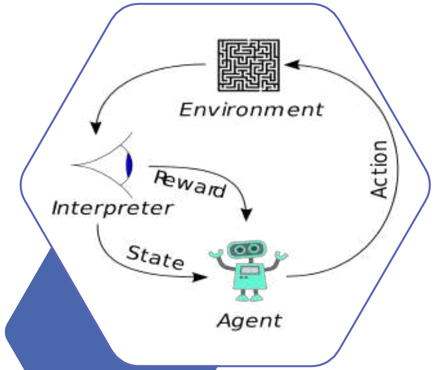
Maze task: Optimal Path



	86	87	88	89	90	91	92		100	G
	85	86		90	91	92	93		99	
S	86	87		91	92	93	94		98	
	87	88		92	93	94	95	96	97	
	88	89	90	91	92		94	95	96	
	87	88	89	90	91	92	93	94	95	

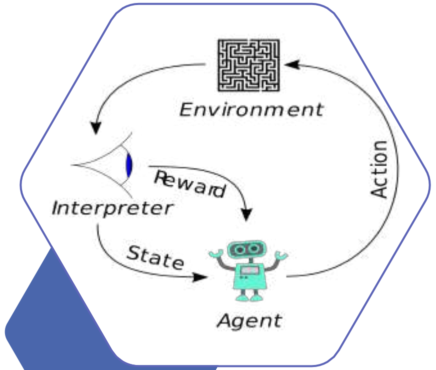
V^*

Why On-line learning is important?



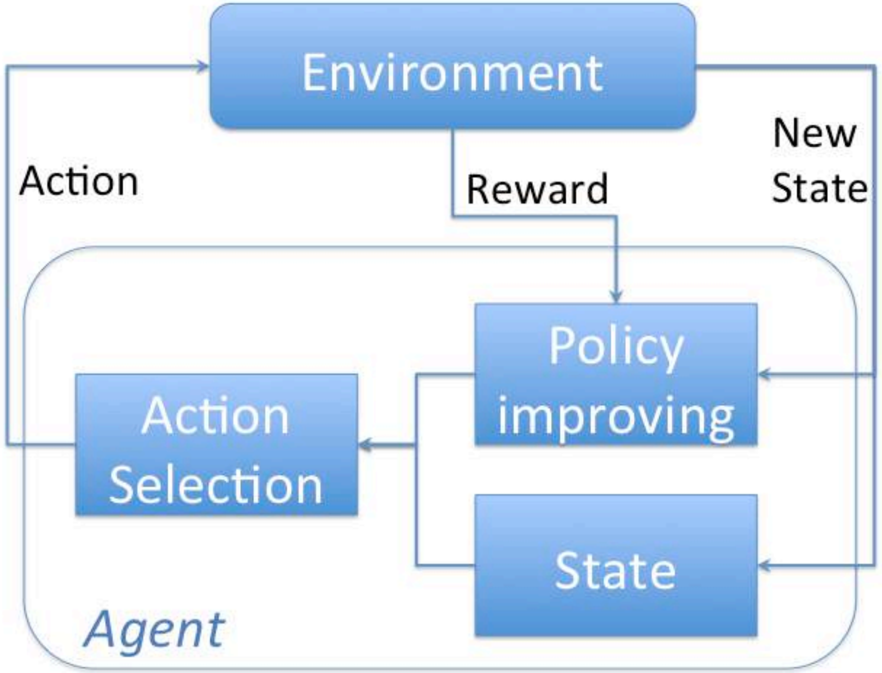
- Assumption: All system parameters to be known.
- Finding the optimal policy becomes a straightforward computational problem
  - E.g., value iteration, but even policy iteration, linear programming, etc…
- What if rewards/transitions probabilities are unknown? Can we compute the optimal policy?
- We have to deal with a *Reinforcement Learning problem*!

Agent-Environment Interaction

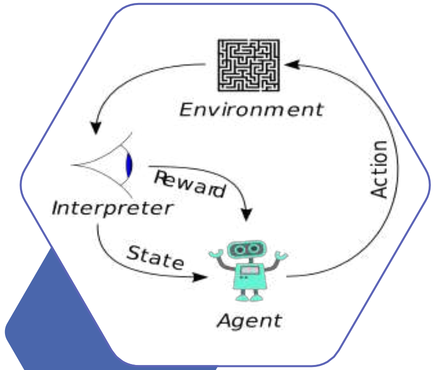


- Everything *inside* the agent is completely known and controllable by the agent.

- Everything *outside* is incompletely controllable but may or may not be completely known.

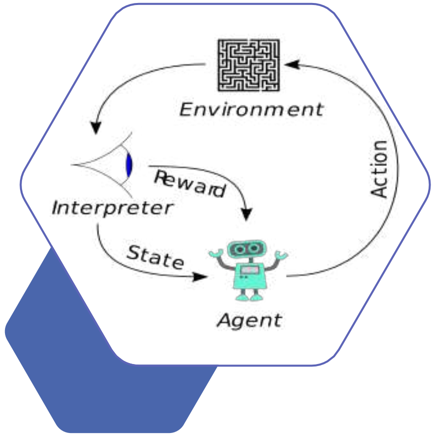


Agent Knowledge



- A reinforcement learning problem can be posed in a variety of different ways depending on assumptions about the level of knowledge initially available to the agent.
- In problems of *complete knowledge*, the agent has a complete and accurate model of the environment's dynamics .
- If the environment is an MDP, then such a model consists of the one-step *transition probabilities* and *expected rewards* for all states and their allowable actions.
    - In problems of *incomplete knowledge*, a complete and perfect model of the environment is not available.

Q-Learning



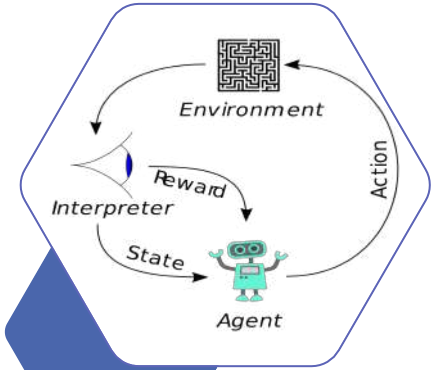
- For any policy π, define $Q^{\pi}: S \times A \rightarrow R$ by

$$Q^{\pi}(s,a) = \sum_{t=0}^{\infty} \gamma^{t} r(t)$$

  - $s(0) = s$ is the initial state,
  - $a(0) = a$ is the action taken,
  - all subsequent states, actions, and rewards arise following policy π
  - Just like V^{π} except that action a is taken at the very first step and only after this, policy π is followed Bellman equations can be rewritten in terms of Q-values.

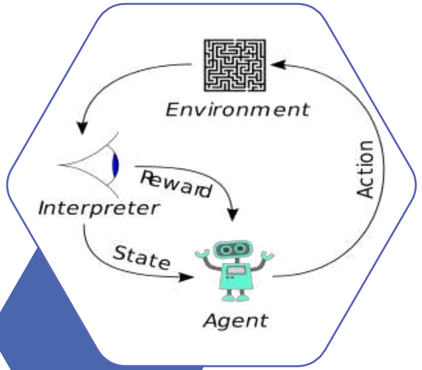
Reinforcement Learning

Q-Values



If agent is in state	An starts with this action and then follows the policy	Return should be
s_1	a_1	-5
s_1	a_2	3
s_2	a_1	17.1
s_2	a_2	10
…	…	…

Q-Values



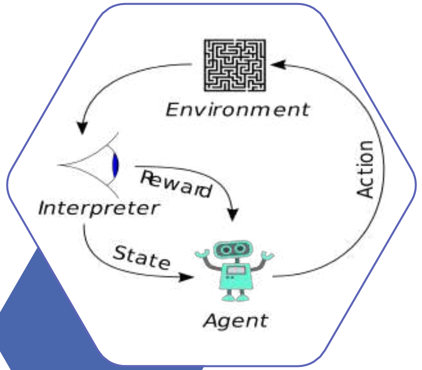
- Relationship between Value function and Q-function (given the state transition $s \rightarrow s'$)

$$V^{\pi}(s) = R(s, \pi(s)) + \gamma V^{\pi}(s')$$

vs.

$$Q^{\pi}(s, a) = R(s, a) + \gamma V^{\pi}(s')$$

Q-Values



- Relationship between Value function and Q-function (given the state transition $s \rightarrow s'$)

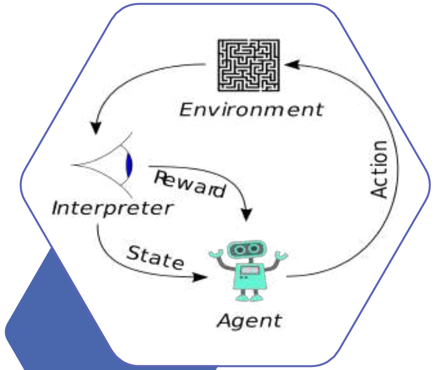
$$V^{\pi}(s) = R(s, \pi(s)) + \gamma V^{\pi}(s')$$

vs.

$$Q^{\pi}(s, a) = R(s, a) + \gamma V^{\pi}(s')$$

Reinforcement Learning

Q-Values



- Define $Q^* = Q^{\pi^*}$, where π* is an optimal policy

$$Q^*(s,a) = R(s,a) + \gamma V^*(s')$$

- Since:

$$V^*(s) = \max_a \{R(s,a) + \gamma V^*(s')\}$$

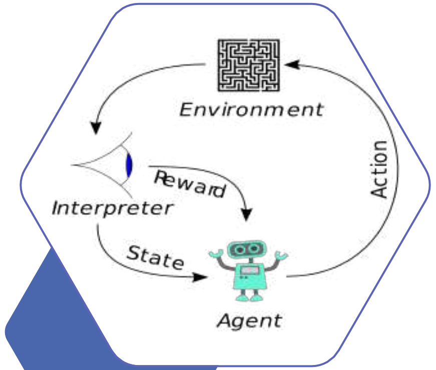
- Then:

$$V^*(s) = \max_a Q^*(s,a)$$

- And:

$$Q^*(s,a) = R(s,a) + \gamma \max_{a'} Q^*(s',a')$$

Q-Values



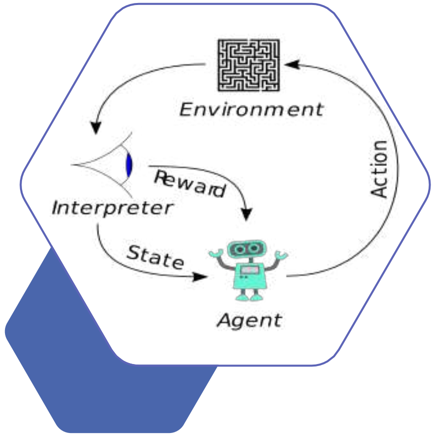
- The optimal policy π^* is greedy for Q^*, that is

$$\pi^*(s) = \underset{a}{\operatorname{argmax}}\, Q^*(s,a)$$

[it follows from $V^*(s) = \max_a Q^*(s,a)$]

Reinforcement Learning

Q-learning Algorithm

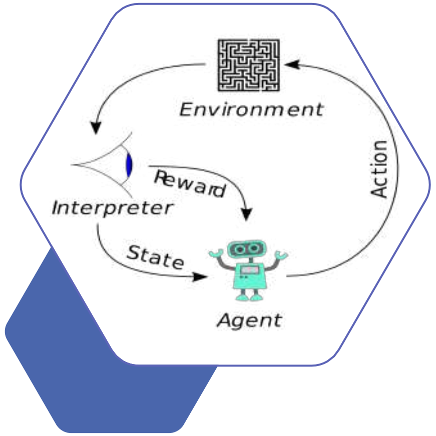


- Q is the estimated utility function
  - It tells us how good an action is, given a certain state
  - It includes immediate reward for making an action + best utility (Q) for the resulting state (future utility)
  - It allows to compute the optimal policy.

- Q-learning is based on an online estimation of the Q function

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha\,[\,r(s,a) + \gamma \max_{a'} Q(s',a')]$$

Q-learning Algorithm



Initialize $Q(s,a)$ arbitrarily

Repeat (for each decision epoch)

    Initialize s

    Repeat (for each step of episode)

        Choose a from s using a policy derived from Q

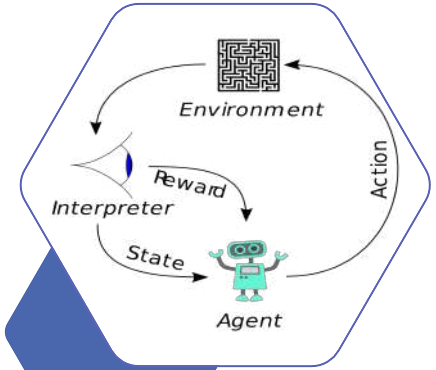
        <u>Take</u> action a, <u>observe</u> $r(s,a)$,

        $Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha\,[\,r(s,a) + \gamma \max_{a'} Q(s',a')]$

        $s \leftarrow s'$

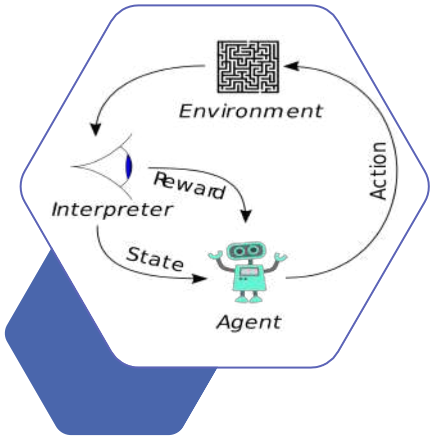
    until s is terminal

Exploitation and Exploration



- Q-learning algorithm does not specify what the agent should actually do. The agent learns a *Q*-function that can be used to determine an optimal action.
- There are two things that are useful for the agent to do:
  - exploit the knowledge that it has found at the current state *s* by taking one of the actions *a* that maximizes *Q[s,a]*.
  - explore in order to build a better estimate of the optimal *Q*-function. That is, it should select a different action from the one that it currently thinks is best.

Exploitation and Exploration



Choose *a* from *s* using a policy derived from *Q*

– Simple Approach: ε-greedy policy

  – ε small number, e.g., 0.1

Generate a random number p

if $p \leq \varepsilon$

  Choose an action at random → explore

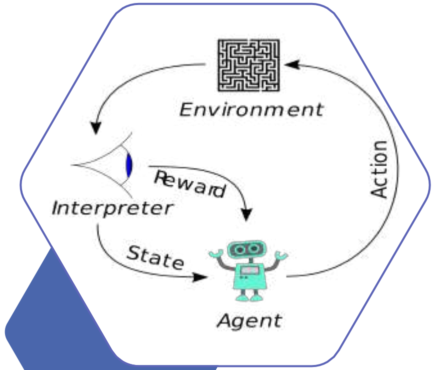
else

  Choose the greedy action

$$a^* = \arg\max_a Q(s,a) \rightarrow \text{exploit}$$

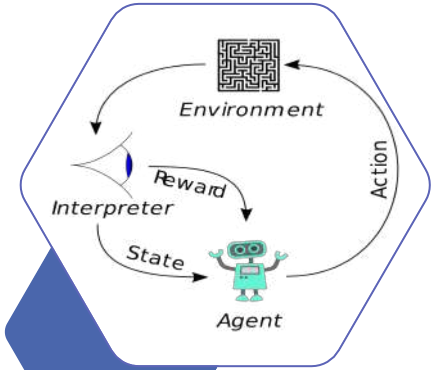
end

Q-learning Discussion



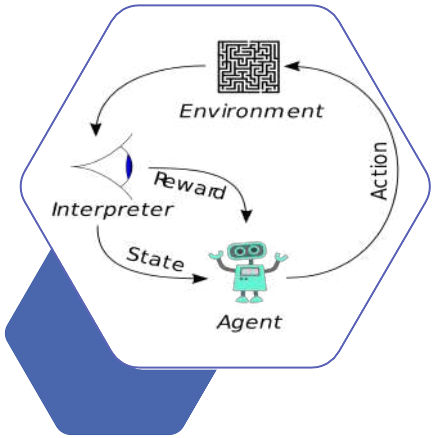
- Q-learning is guaranteed to converge to the optimal Q-values if all Q(s,a) values are updated infinitely often (Watkins and Dayan 1992).
- It follows that exploration is necessary
  - A common approach is the ε-greedy strategy
- Q-learning can be very slow to converge to the optimal policy, especially if the state space is large.
- One of the biggest challenges in the RL field is to *speed up* the learning process.

Learning or Planning?



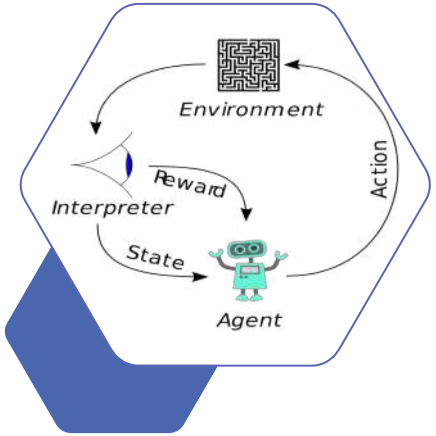
- Classical DP emphasis for optimal control
  - Dynamics and reward structure known
  - Off-line computation
- Traditional RL emphasis
  - Dynamics and/or reward structure initially unknown
  - On-line learning

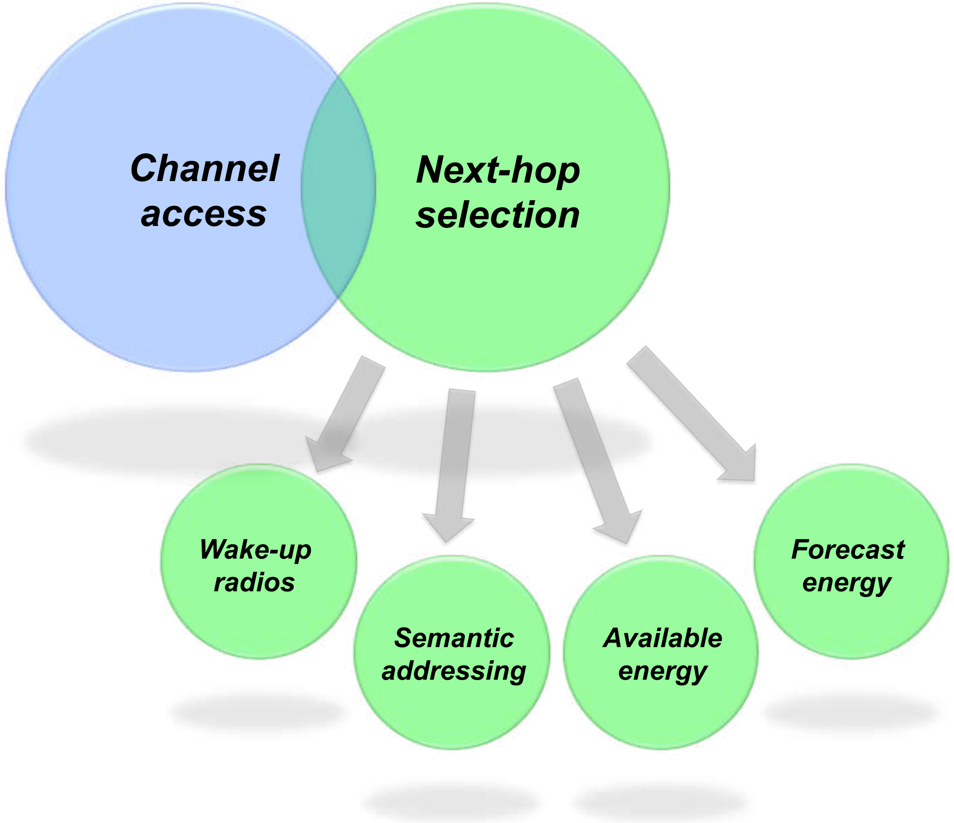
- Computation of an optimal policy off-line with known dynamics and reward structure can be regarded as planning.



Reinforcement Learning in Practice

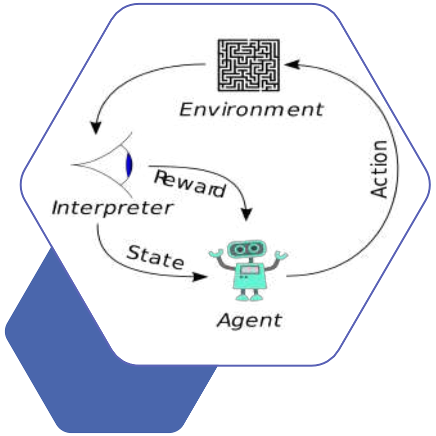
The WHARP forwarding strategy



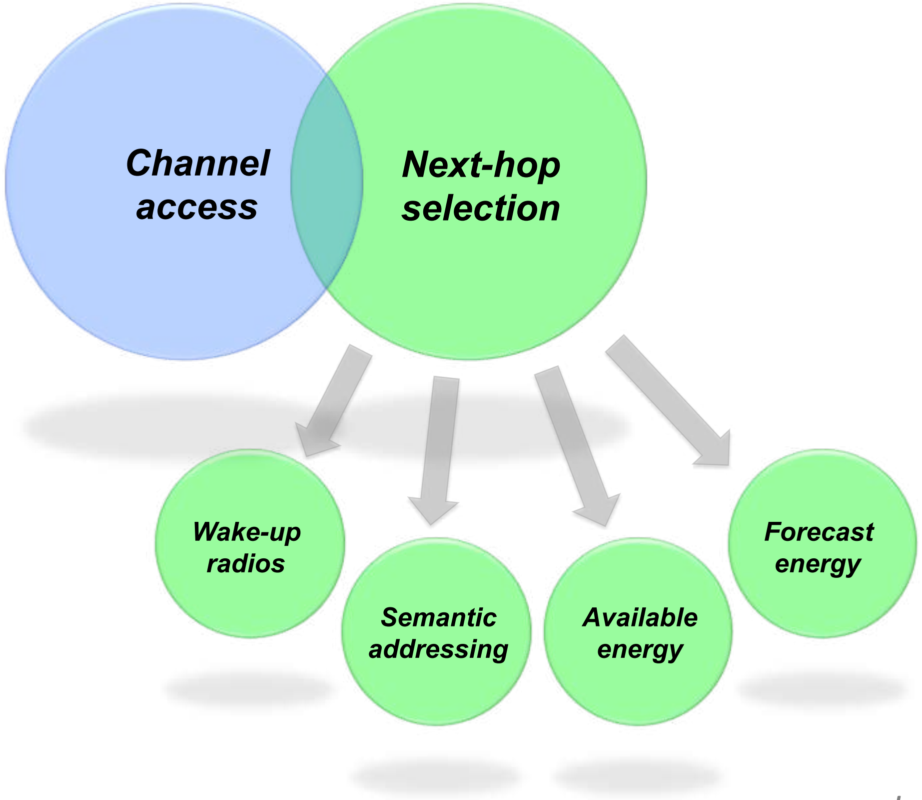


S. Basagni, V. Di Valerio, G. Koutsandria, C. Petrioli, and D. Spenza. “WHARP: A wake-up radio and harvesting-based forwarding strategy for green wireless networks, " in *Proceedings of IEEE MASS 2017*, Orlando, FL, USA, October 22–25 2017.

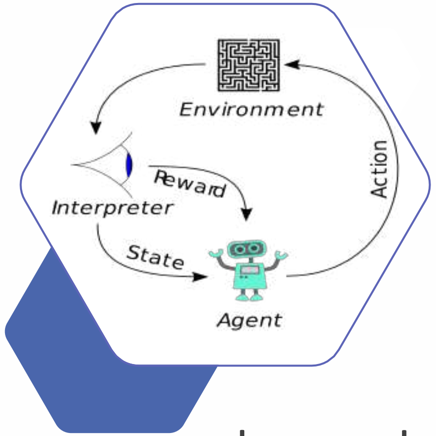
The WHARP forwarding strategy



- **Objective**: Optimize energy consumption through a «smart» selection of next-hop relays.



The WHARP forwarding strategy



- Semantic awakenings: Distance from the sink

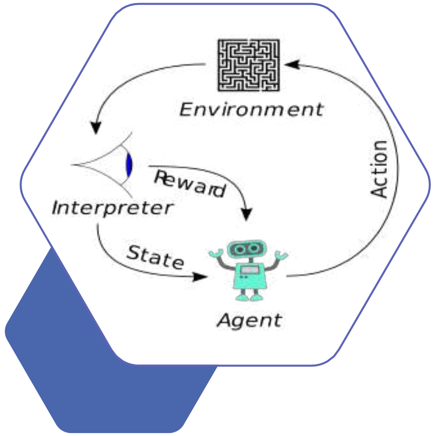
- Nodes decide whether to participate to the relay selection process based on a proactive **Markov Decision Process** mechanism
    1. Available energy
    2. Forecast energy

- MDP solution method: Backward Value Iteration (BVI)

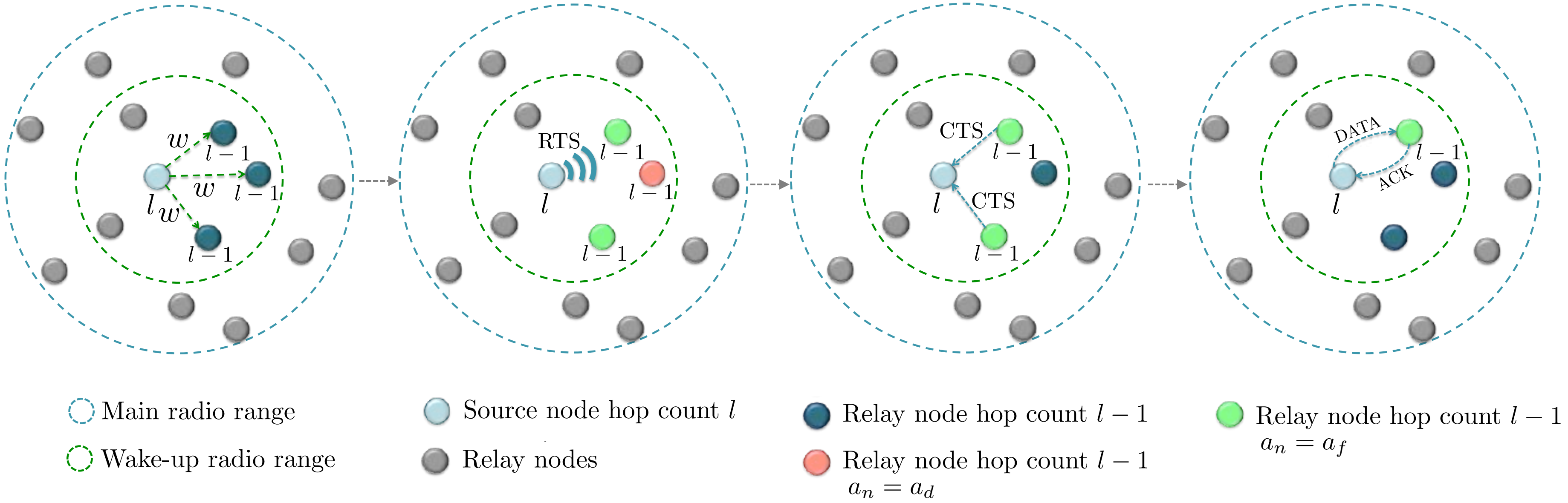
    - WHARP decisions optimize system performance over time

S. Basagni, V. Di Valerio, G. Koutsandria, C. Petrioli, and D. Spenza. “WHARP: A wake-up radio and harvesting-based forwarding strategy for green wireless networks, " in *Proceedings of IEEE MASS 2017*, Orlando, FL, USA, October 22–25 2017.

The WHARP forwarding strategy

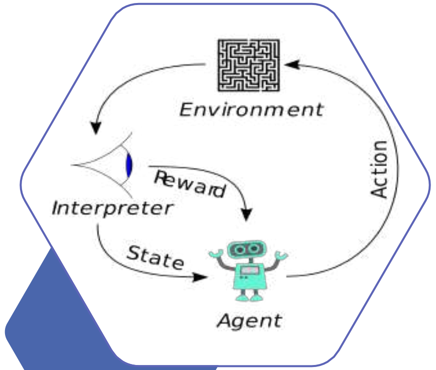


- The MDP policy outputs either green or red.
- Green output: Nodes turn on their main radio
- Red output: Nodes remain asleep



S. Basagni, V. Di Valerio, G. Koutsandria, C. Petrioli, and D. Spenza. "WHARP: A wake-up radio and harvesting-based forwarding strategy for green wireless networks, " in *Proceedings of IEEE MASS 2017*, Orlando, FL, USA, October 22–25 2017.

MDP model: States



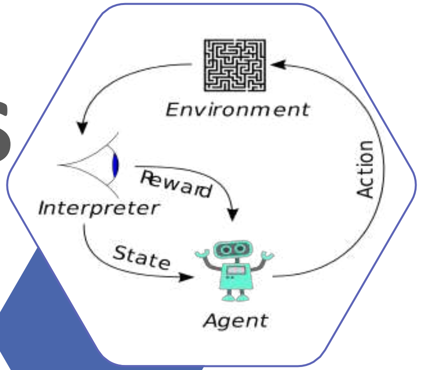
Node

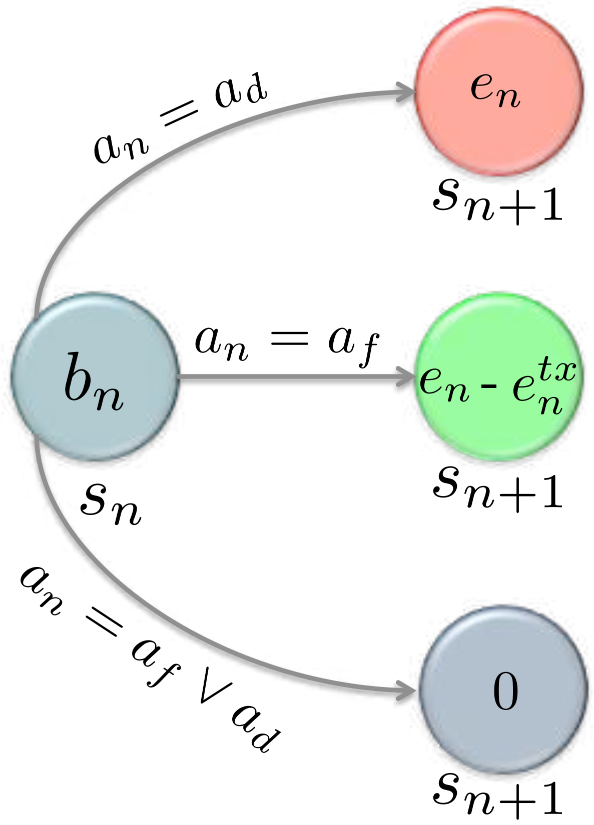
- e_n : overall energy available for packet forwarding in decision epoch n
- b_n : energy level in decision epoch n
- h_n : energy harvested in decision epoch n
- e_n^x : energy spent for sensing and transmitting data

$$e_n = b_n + h_n - e_n^x$$

S. Basagni, V. Di Valerio, G. Koutsandria, C. Petrioli, and D. Spenza. "WHARP: A wake-up radio and harvesting-based forwarding strategy for green wireless networks, " in *Proceedings of IEEE MASS 2017*, Orlando, FL, USA, October 22–25 2017.

MDP model: Actions and Transitions





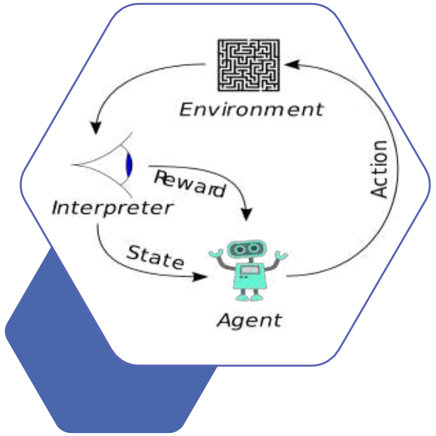
a_n : Action in decision epoch n
a_d : Do not forward data
a_f : Forward data

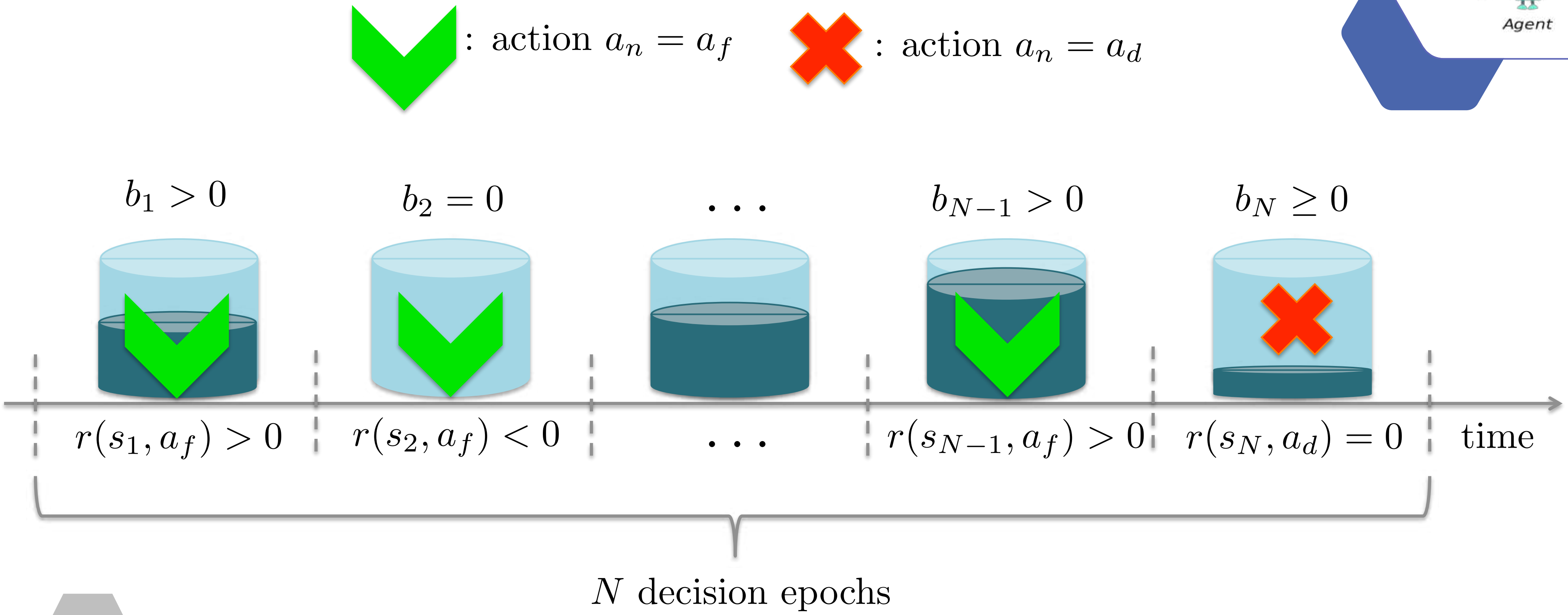
e_n^{tx} : Energy spent to relay data

$s_n = 0 \Rightarrow b_n = 0$: All-off node

S. Basagni, V. Di Valerio, G. Koutsandria, C. Petrioli, and D. Spenza. "WHARP: A wake-up radio and harvesting-based forwarding strategy for green wireless networks, " in *Proceedings of IEEE MASS 2017*, Orlando, FL, USA, October 22–25 2017.

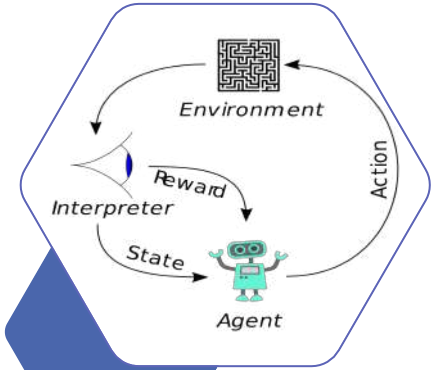
MDP model: Revenue





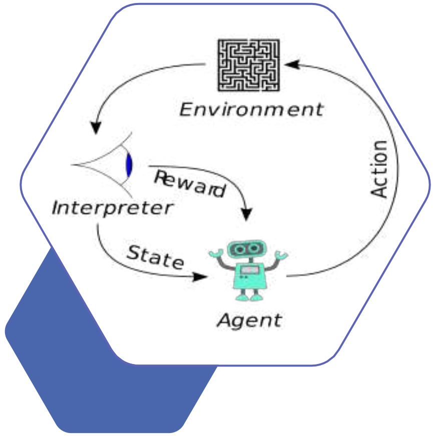
S. Basagni, V. Di Valerio, G. Koutsandria, C. Petrioli, and D. Spenza. “WHARP: A wake-up radio and harvesting-based forwarding strategy for green wireless networks, " in *Proceedings of IEEE MASS 2017*, Orlando, FL, USA, October 22–25 2017.

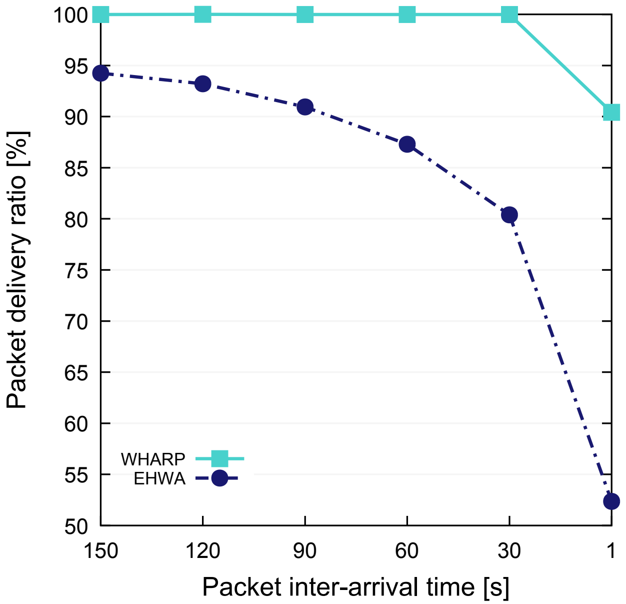
WHARP: Performance Evaluation



- Scenario: WHARP vs. EHWA
  - GreenCastalia Simulator
  - M=120; Grid: 200 x 200 m^2
  - R = 60m; R_w = 45m
  - Energy model: Magonode++ mote

WHARP: Performance Evaluation

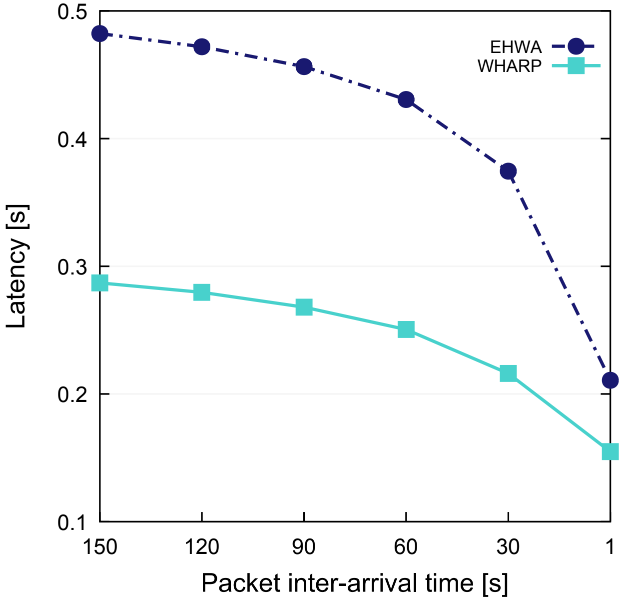




PDR = 91%

PDR = 52%

70%



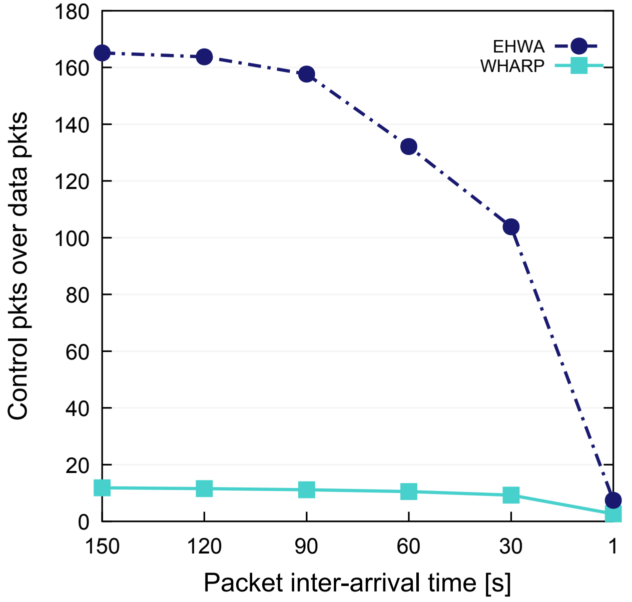
0.21 sec

0.15 sec

x 1.36

Reinforcement Learning in Practice

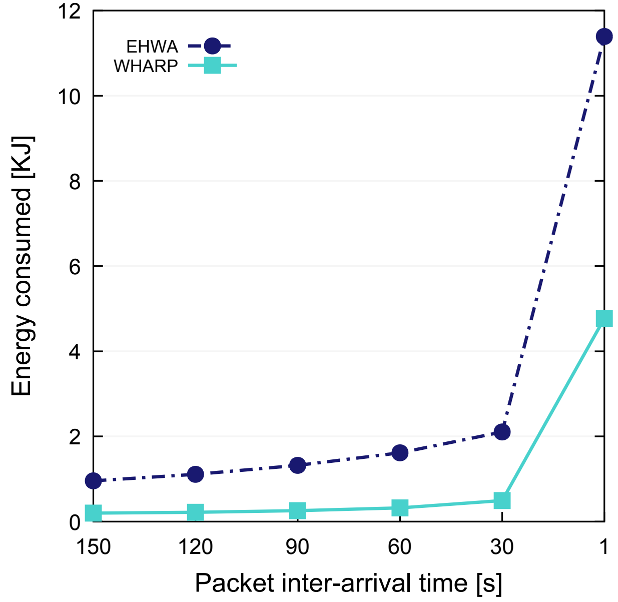
WHARP: Performance Evaluation



165

12

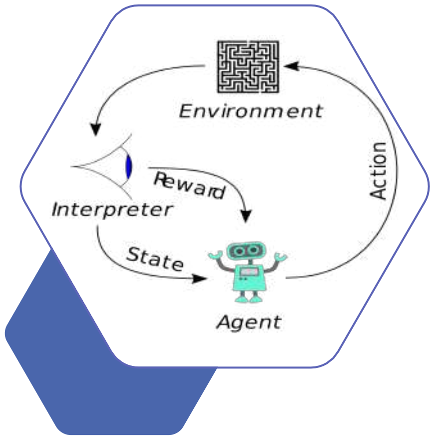
x 14



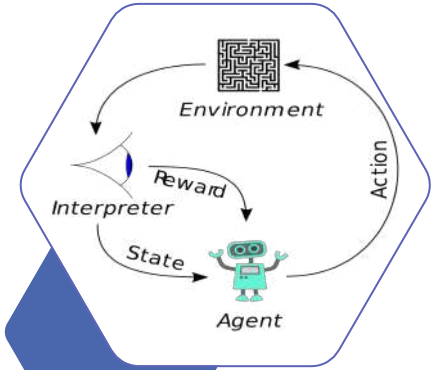
11.39 KJ

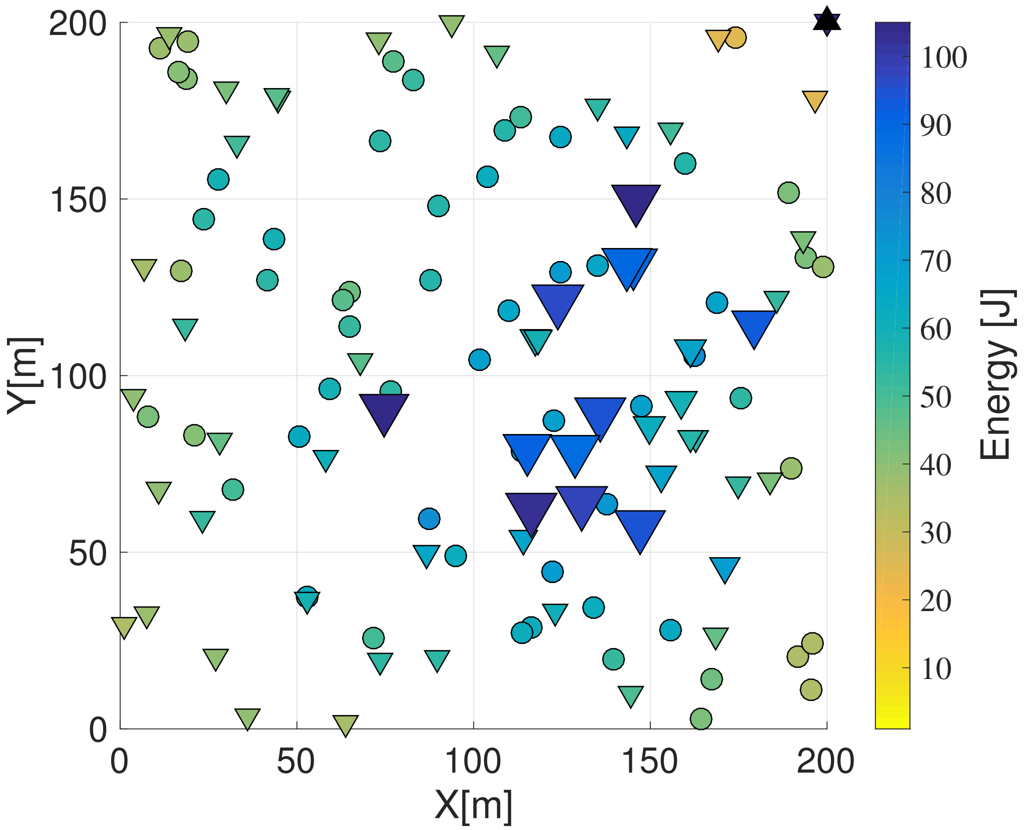
4.77 KJ

x 2.4



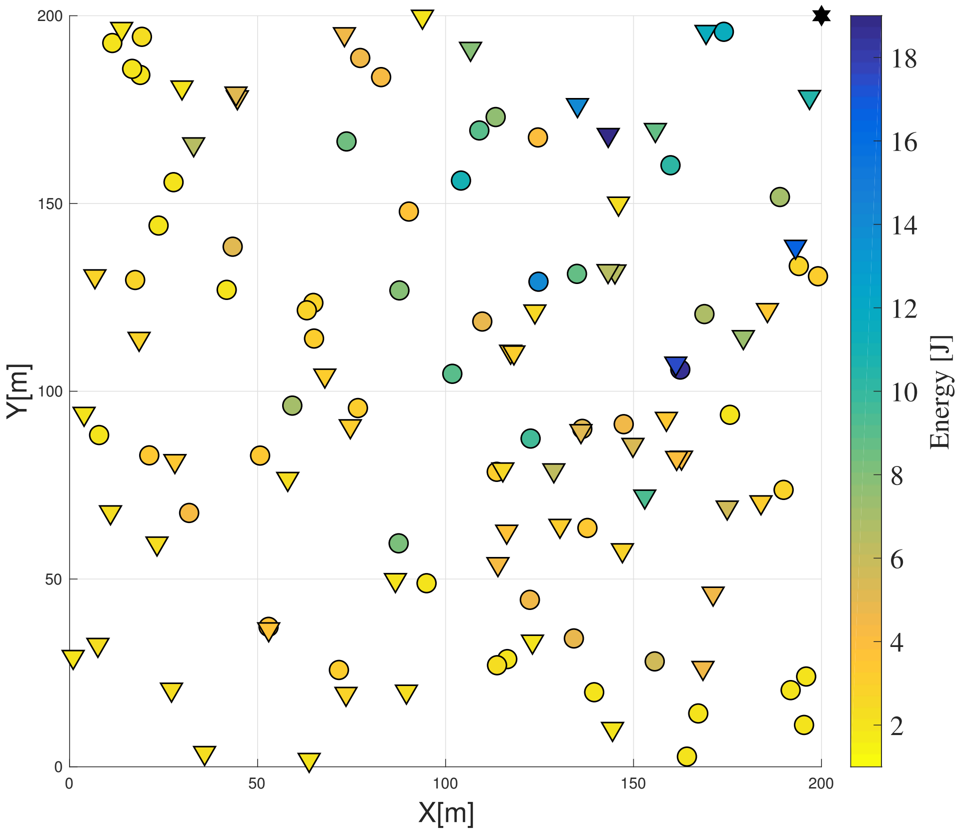
WHARP: Performance Evaluation





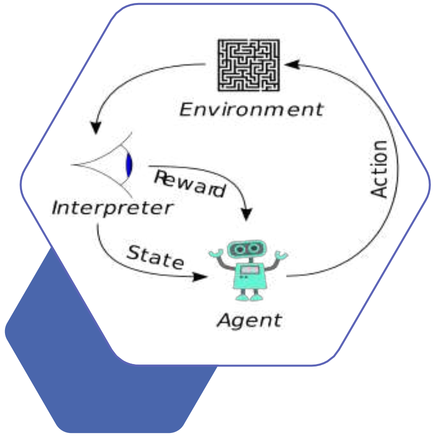
EHWA

WHARP



Reinforcement Learning in Practice

Additional Resources



- An Introduction to Reinforcement Learning, Sutton and Barto, MIT Press 1998
  - Book available free online: https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf

- Algorithms for Reinforcement Learning, Szepesvari, Morgan and Claypool 2010
  - Book available free online: https://sites.ualberta.ca/~szepesva/papers/RLAlgsInMDPs-lecture.pdf

Questions?