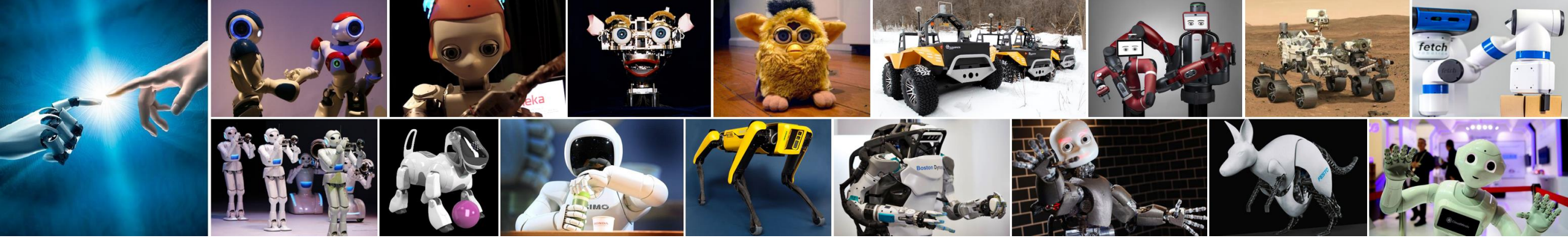
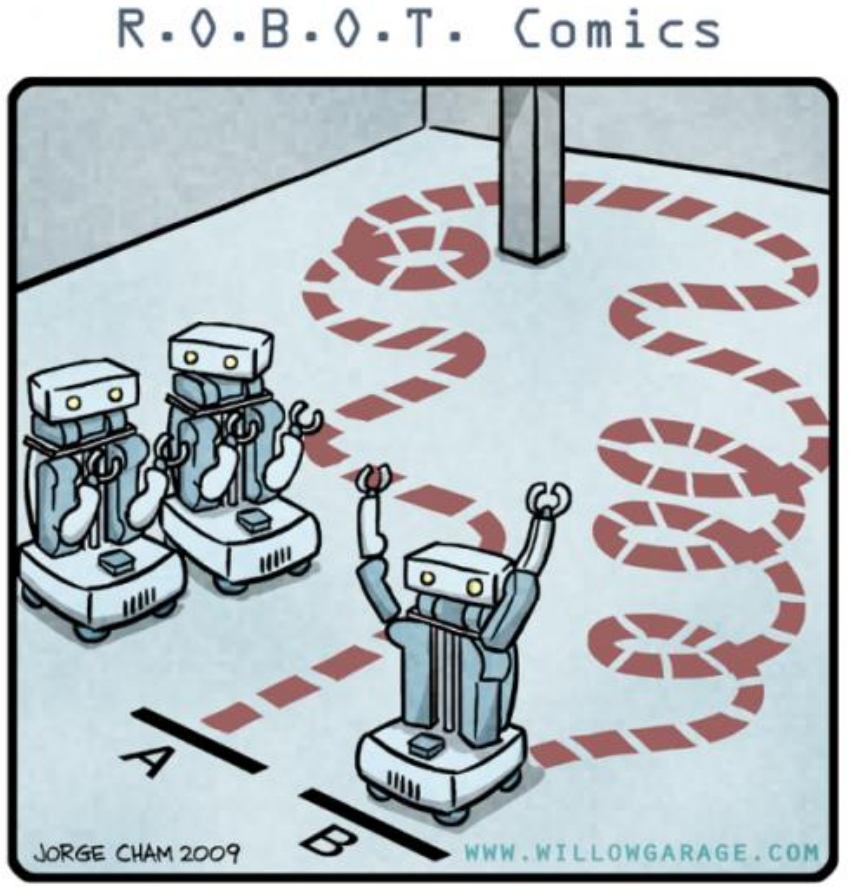


COMPSCI-603: Robotics

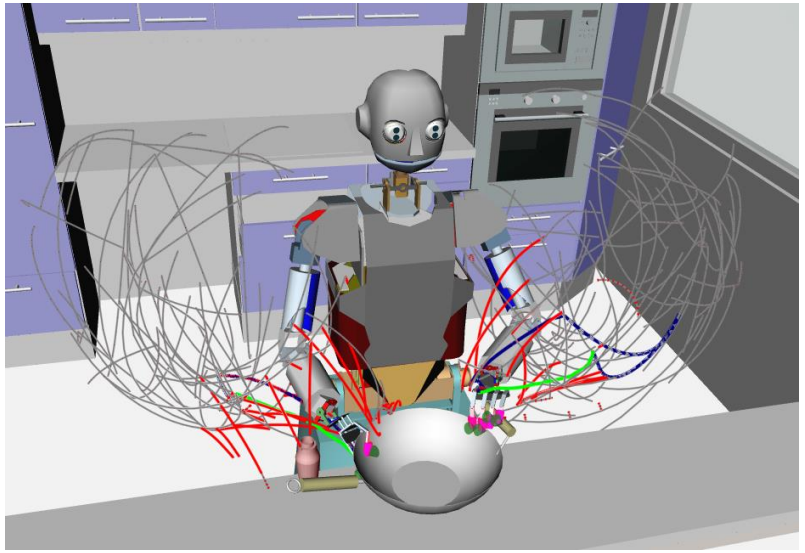
Robot Learning

Robot Decision Making and Planning



Robot Decision Making and Planning

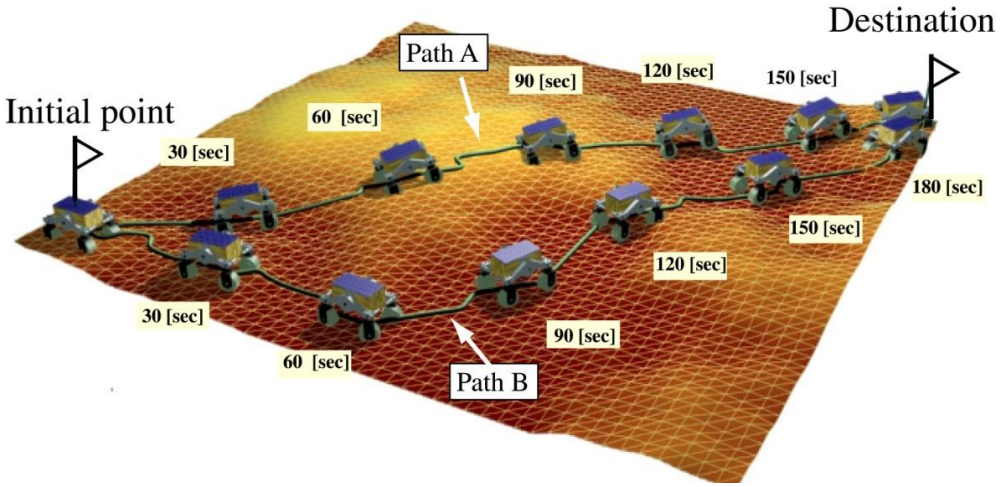
- Robots need to make various decisions and construct different plans, for example:
  - Decision making.
  - Planning: task planning, motion planning (e.g., for robotic arms), and path planning (e.g., for mobile robotics).



- Decision making and planning characteristics:
  - Reactive (one-time) decision making versus sequential planning.
  - Certain versus uncertain scenarios.
  - Observable versus partially observable space.

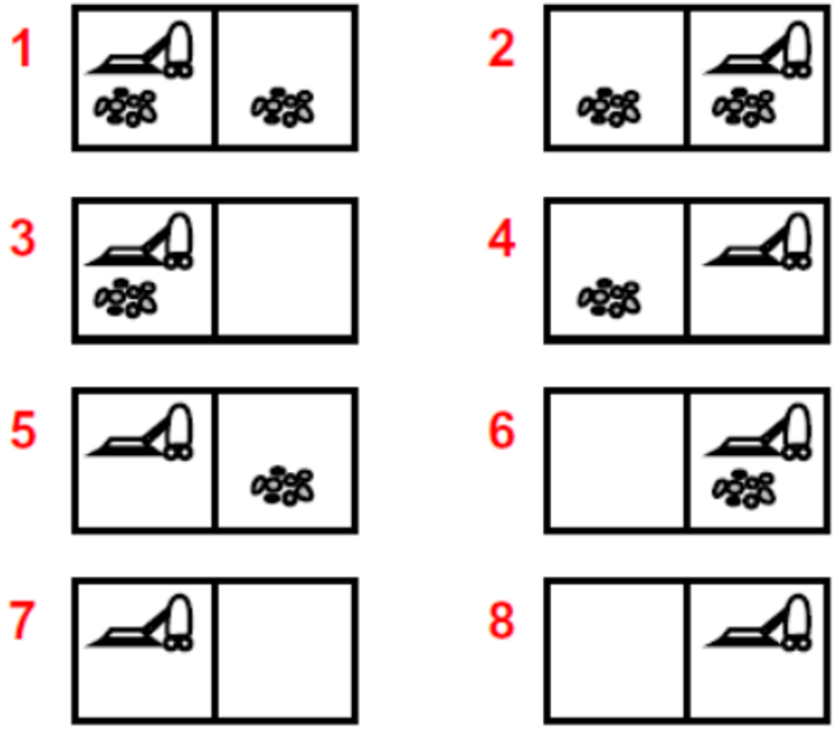
Common Scenarios of Planning

- Deterministic, fully observable:
  - Agent knows exactly which state it is in.
  - Agent action is executed as expected.
- Stochastic, partially observable:
  - Observations provide new information about current state with uncertainty.
  - Robot actions may not be successfully executed.
- Non-observable:
  - Agent may have no idea where it is.



Example: Vacuum World

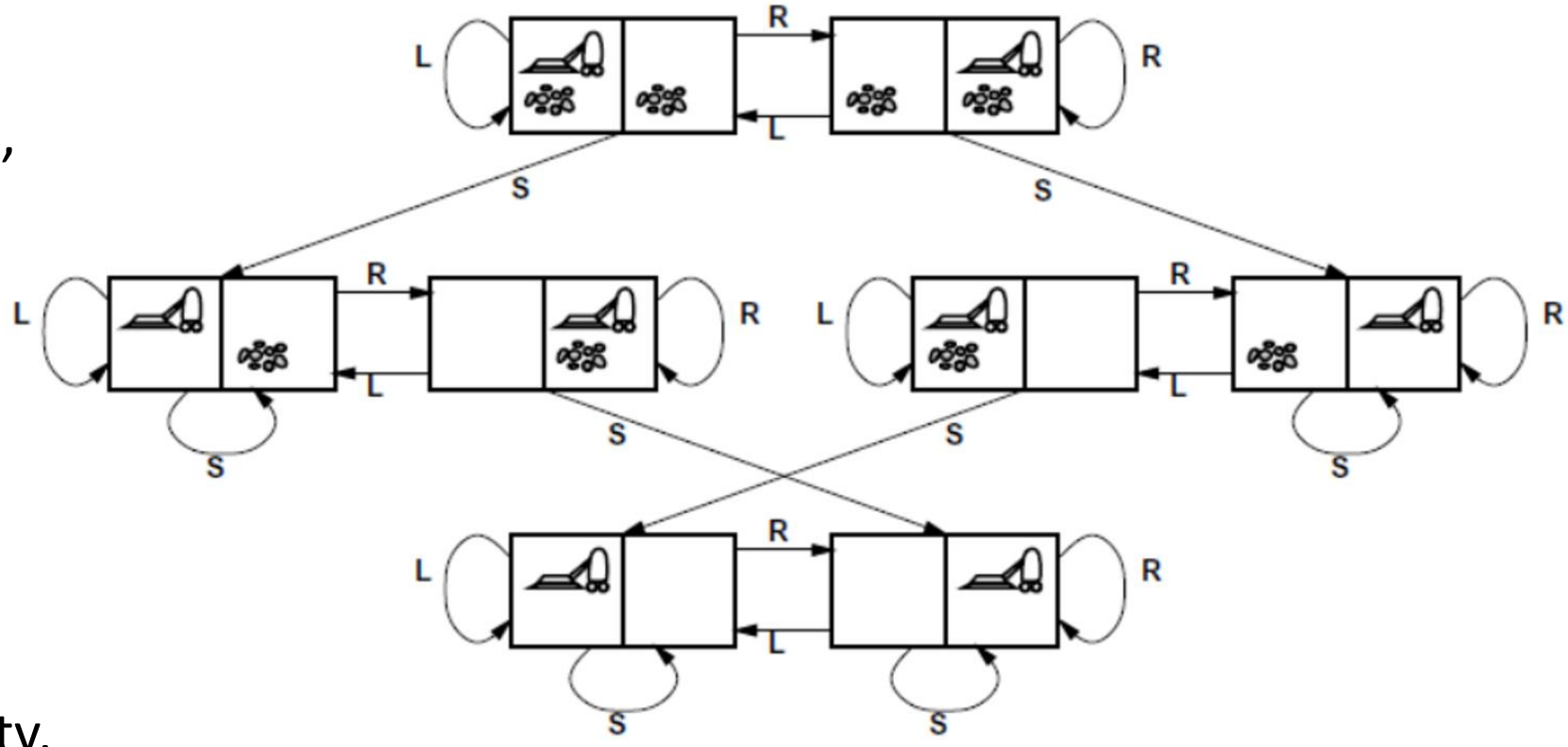
- Observable:
  - Start in #5
  - Actions: [Right; Suck]
- Non-observable:
  - Start in {1;2;3;4;5;6;7;8}
  - E.g., action Right goes to {2;4;6;8}
  - Actions: [Right; Suck; Left; Suck]
- Partially observable:
  - Start in #5, local sensing only
  - Stochastic actions, suck can make a clean carpet dirty
  - Actions: [Right; if dirt then Suck]



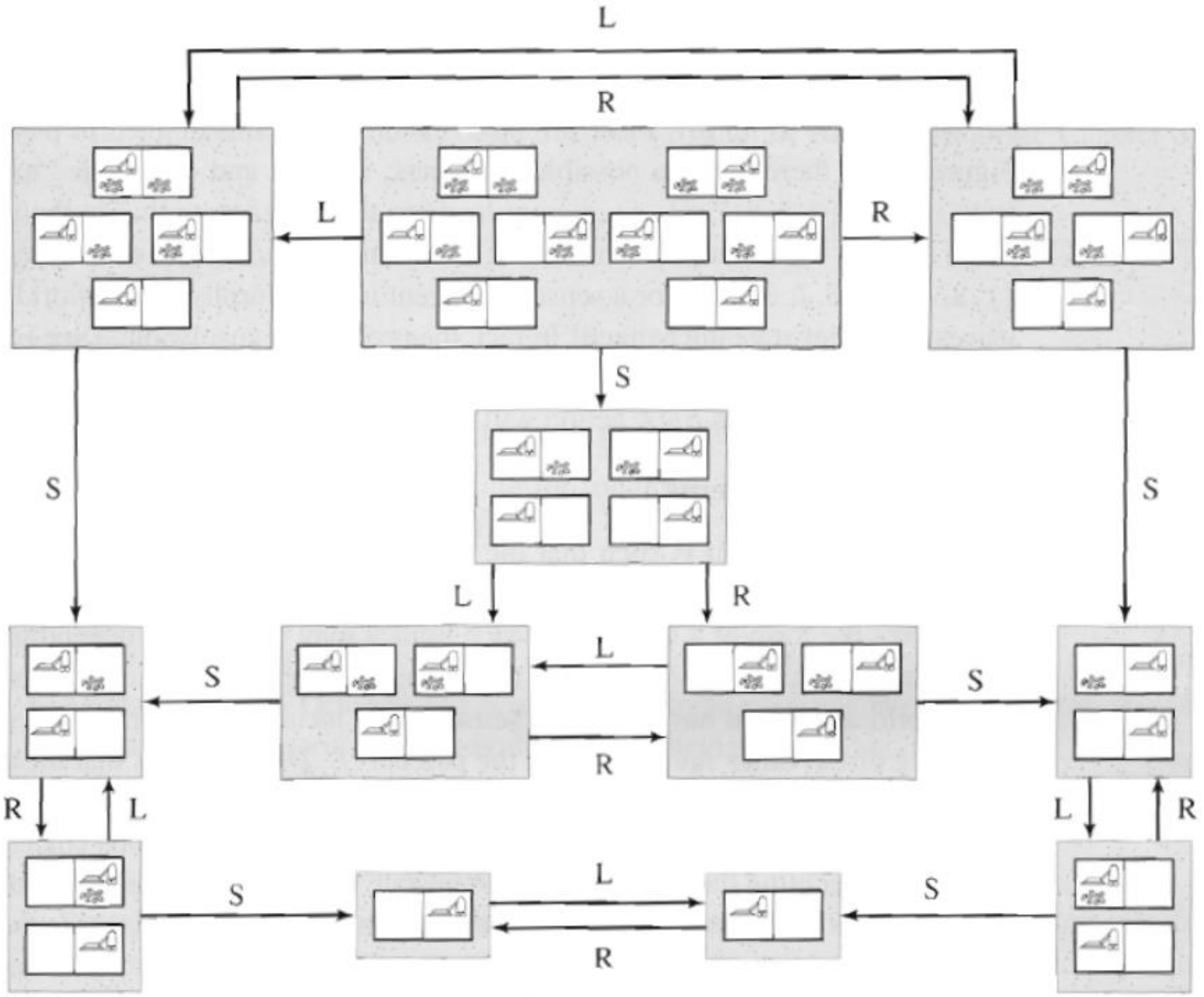
Possible actions: left, right, suck

Example: Vacuum World (Observable)

- States: cross product of robot locations and dirtiness
- Actions: Left, Right, Suck, NoOp
- Successor function: Left/Right changes location, Suck changes dirtiness
- Goal: no dirt
- Cost: 1 per action (0 for NoOp), also called penalty, utility, or reward



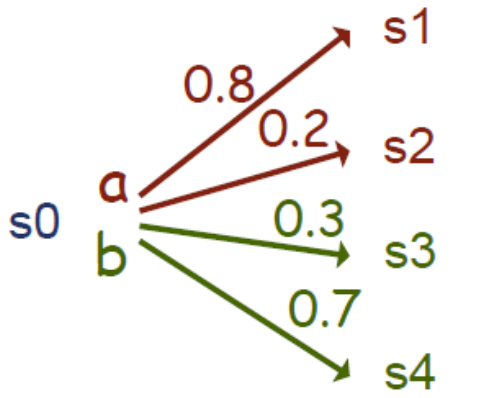
Example: Vacuum World (Non-Observable)



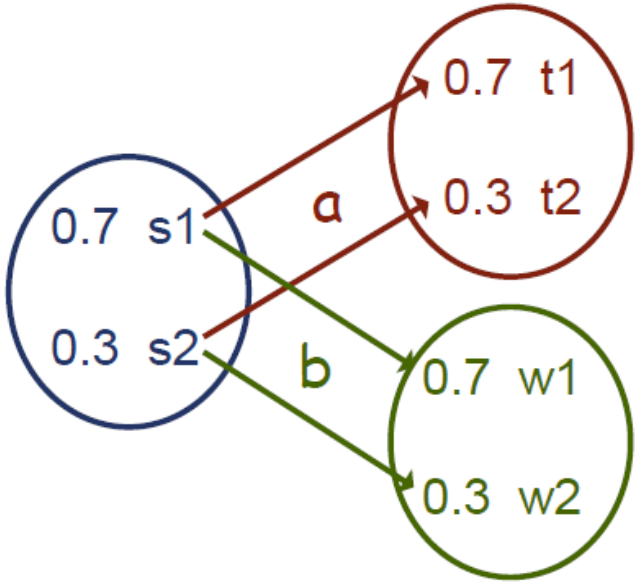
- Definition of states is different in the case of non-observable vacuum world.
- If actions are stochastic, action successor function is also defined differently.

Planning under Uncertainty

- In unstructured environments, robot decision making and planning must be performed under uncertainty.
  - Uncertainty in action outcomes
  - Uncertainty in state of knowledge
  - Any combination of the two



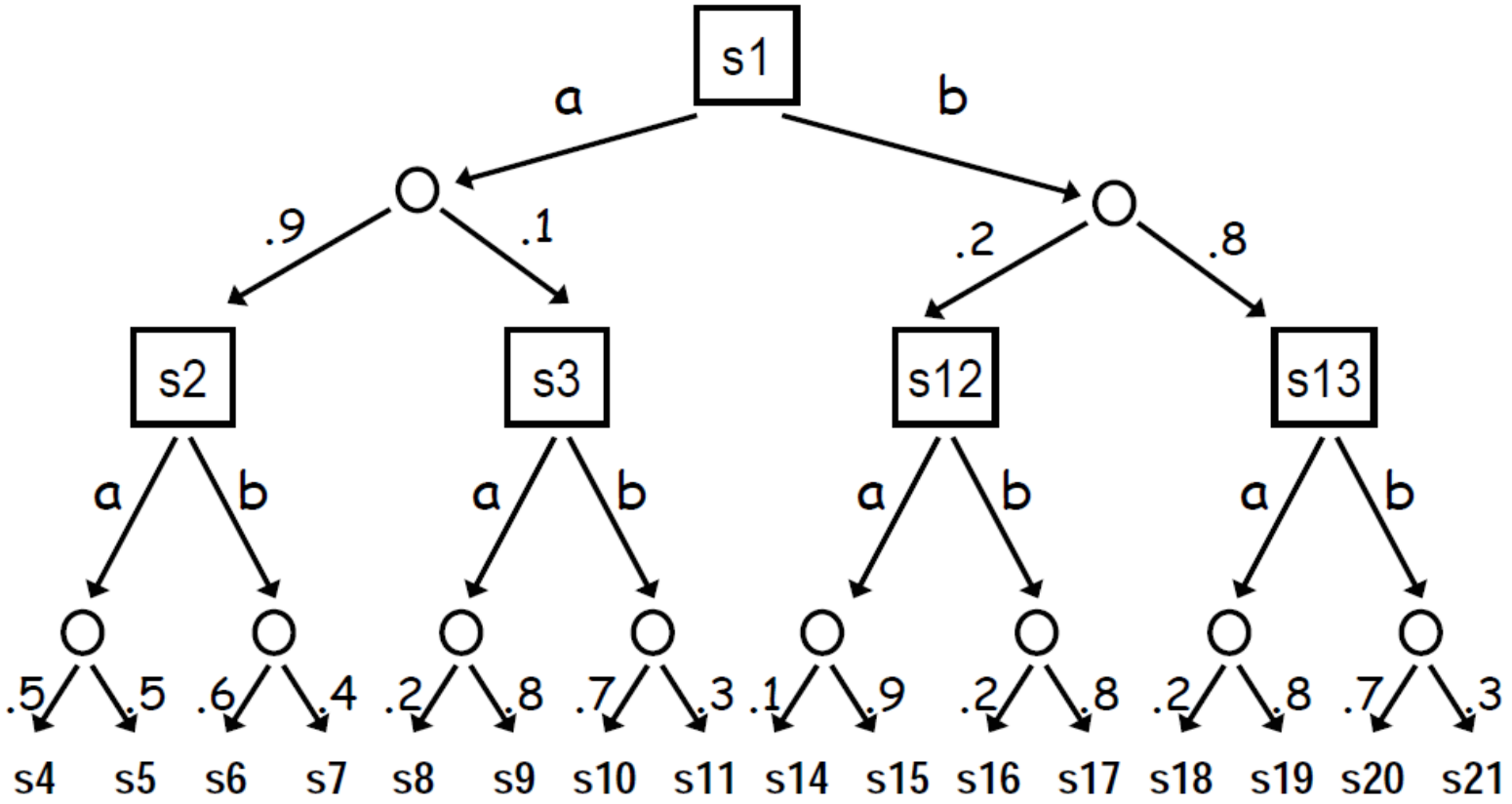
Stochastic actions



Uncertain knowledge

Planning under Uncertainty

- Decision tree provides a classic solution to decision making under uncertainty:



Planning under Uncertainty

- Utility (i.e., reward or cost) function associates a real-valued utility (reward or cost) with each outcome (state or state-action pair).
- With utilities, we can compute and optimize expected utilities for planning under uncertainty.
- The **expected utility** of decision d in the state s can be defined as:

$$EU(d) = \sum_{s \in S} \Pr_d(s) U(s)$$

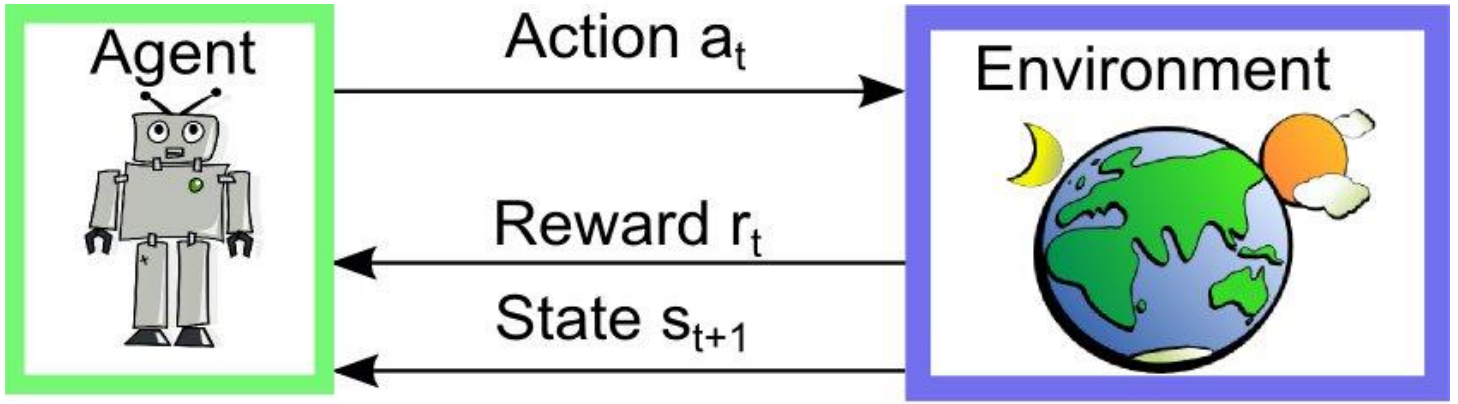
- The principle of **maximum expected utility** states that the optimal decision under uncertainty is the one that has greatest expected utility.

Planning via Reinforcement Learning

- Two fundamental problems in sequential decision making:
  - Planning:
    - A model of the environment is known.
    - Robots perform planning and decision making using this environment model.
    - Robots do not need interactions with the environment for planning.
  - Reinforcement Learning:
    - The environment is initially unknown.
    - The robot interacts with the environment.
    - The robot improves its behaviors.

Reinforcement Learning

- Definition: an area of machine learning inspired by behaviorist psychology, concerned with how robots seek to take actions in an environment so as to maximize a cumulative reward.



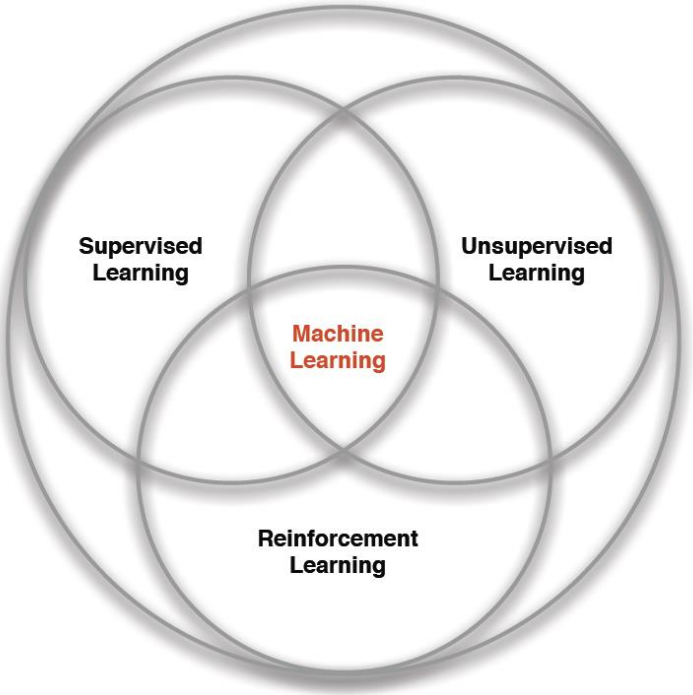
Reinforcement Learning Setup

Reinforcement Learning

- Reinforcement learning is based on the reward hypothesis.
- Reward Hypothesis: All goals can be described by the *maximization of expected cumulative reward.*
  - A reward R_t is a scalar feedback signal.
  - Indicates how well agent is doing at step t.
  - The agent's job is to maximize cumulative reward.
- Actions may have long term consequences; thus reward may be delayed.
  - It may be better to sacrifice immediate reward to gain more long-term reward.

Reinforcement Learning

- Differences from other machine learning paradigms?
  - There is no supervisor, only a reward signal.
  - Feedback is delayed, not instantaneous.
  - Time really matters (sequential, non i.i.d data).
  - Agent's actions affect the subsequent data it receives.

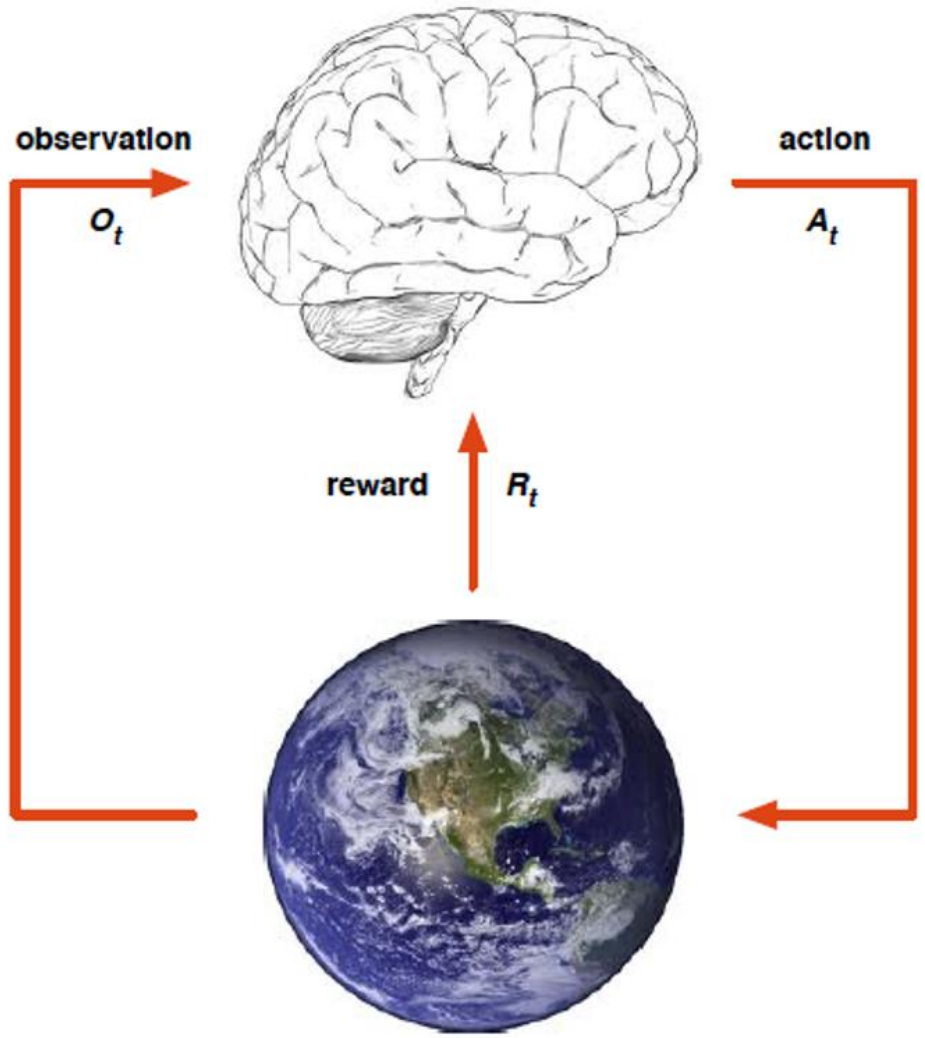


- In robotics, learning from demonstration and reinforcement learning are expected to work together:
  - Learning from demonstration provides an initial solution.
  - Reinforcement learning further adapt and improve the initial solution.

https://www.youtube.com/watch?v=M-QUkgk3HyE

Agent (Robot) and Environment

- At each step t, the agent:
  - Receives observation O_t
  - Receives scalar reward R_t
  - Executes action A_t
- The environment:
  - Receives action A_t
  - Generates observation O_{t+1}
  - Generates scalar reward R_{t+1}
- t increments at environment step



Slides modified from Dr. David Silver

History and State

- The history is a sequence of observations, rewards and actions:

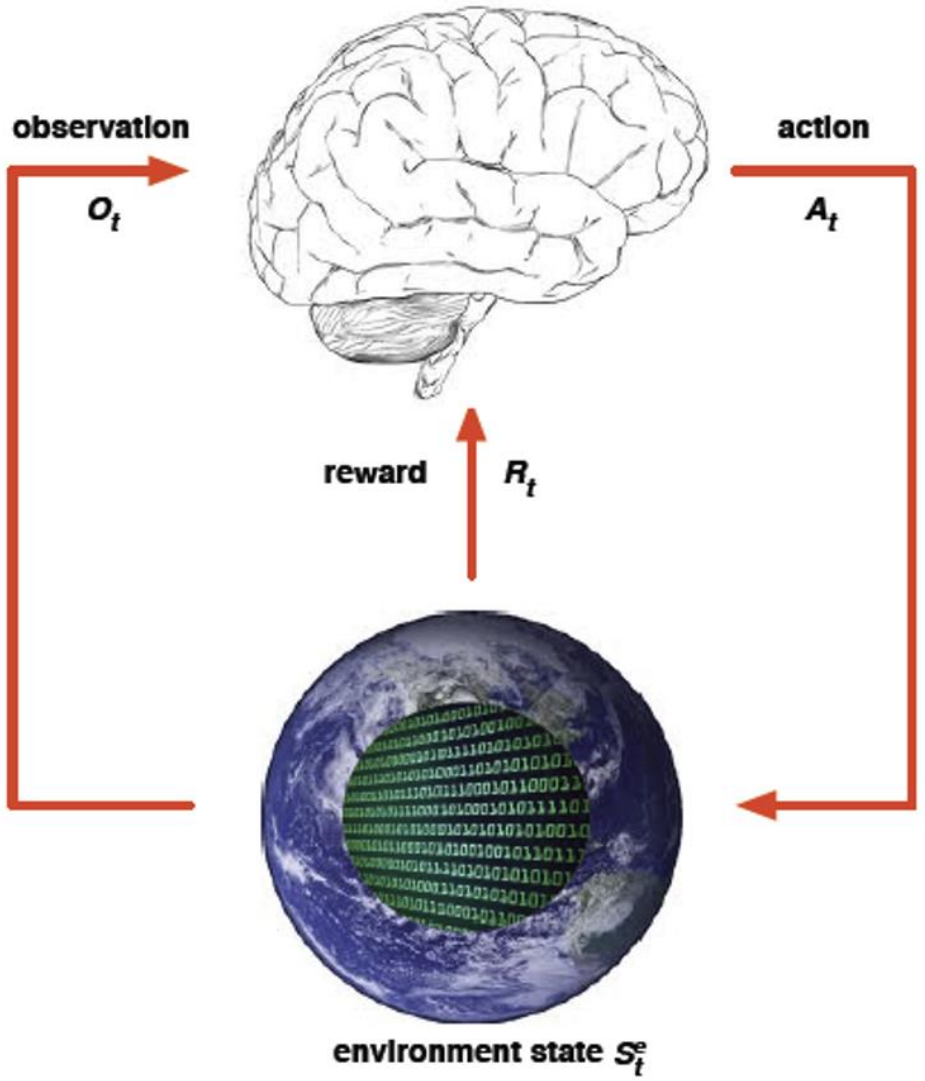
$$H_t = O_1, R_1, A_1, \ldots, O_{t-1}, R_{t-1}, A_{t-1}, O_t, R_t$$

  - It is also called the sensorimotor stream of an agent.
  - All observable variables (observations and rewards) are up to time t.
  - What happens next depends on the history:
    - The agent selects actions.
    - The environment selects observations and rewards.
- State is the information used to determine the next action, which is formally defined as a function of the history:

$$S_t = f(H_t)$$

Environment State

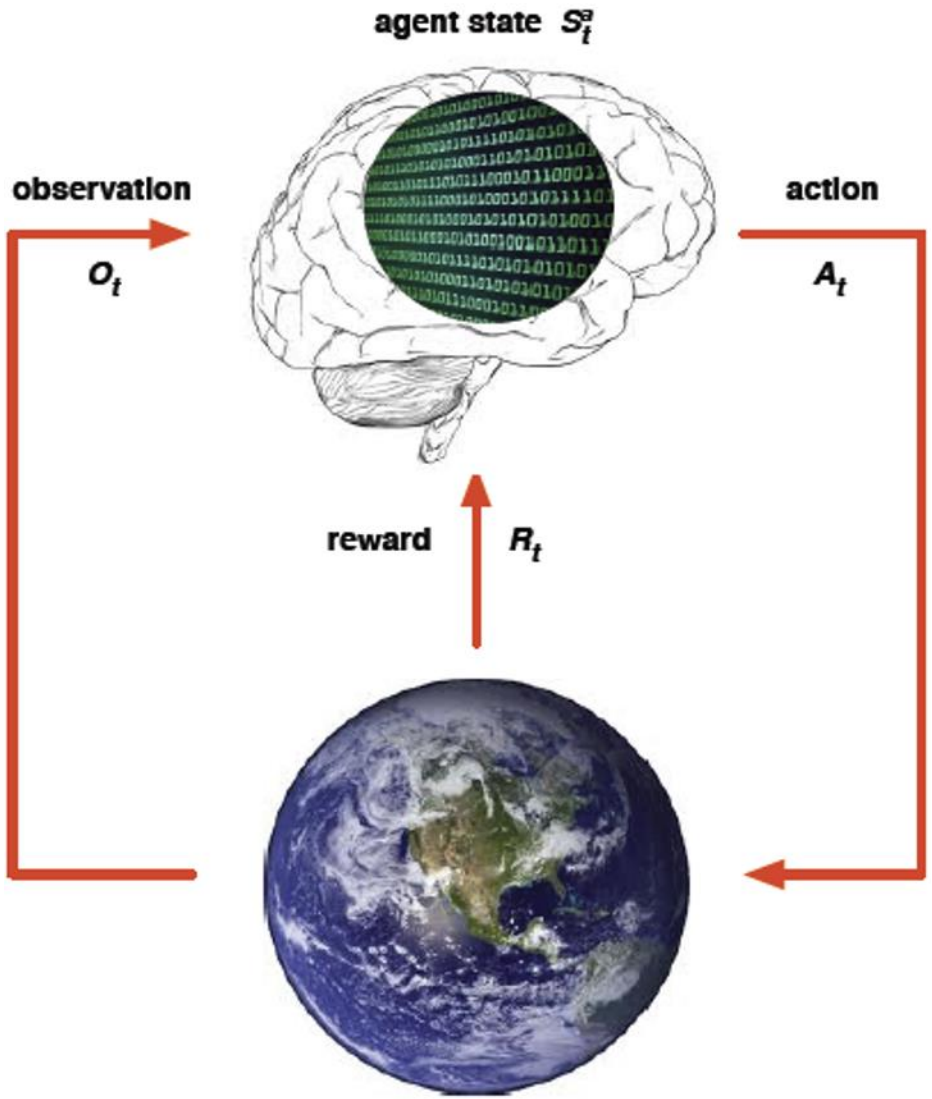
- The environment state S_t^e is the environment's private representation.
- It is used by the environment to pick the next observation and reward.
- The environment state S_t^e is not usually visible to the agent.
- Even if S_t^e is visible, it may contain irrelevant information.



Agent State

- The agent state S_t^a is the agent's internal representation.
- It can be used by the agent to pick the next action.
- It can be computed based on the history:

$$S_t^a = f(H_t)$$



Information State

- An information state (a.k.a., Markov state) contains all useful information from the history.

Definition

A state S_t is Markov if and only if

$$\mathbb{P}[S_{t+1} \mid S_t] = \mathbb{P}[S_{t+1} \mid S_1, \ldots, S_t]$$

  - "The future is independent of the past given the present."

$$H_{1:t} \rightarrow S_t \rightarrow H_{t+1:\infty}$$

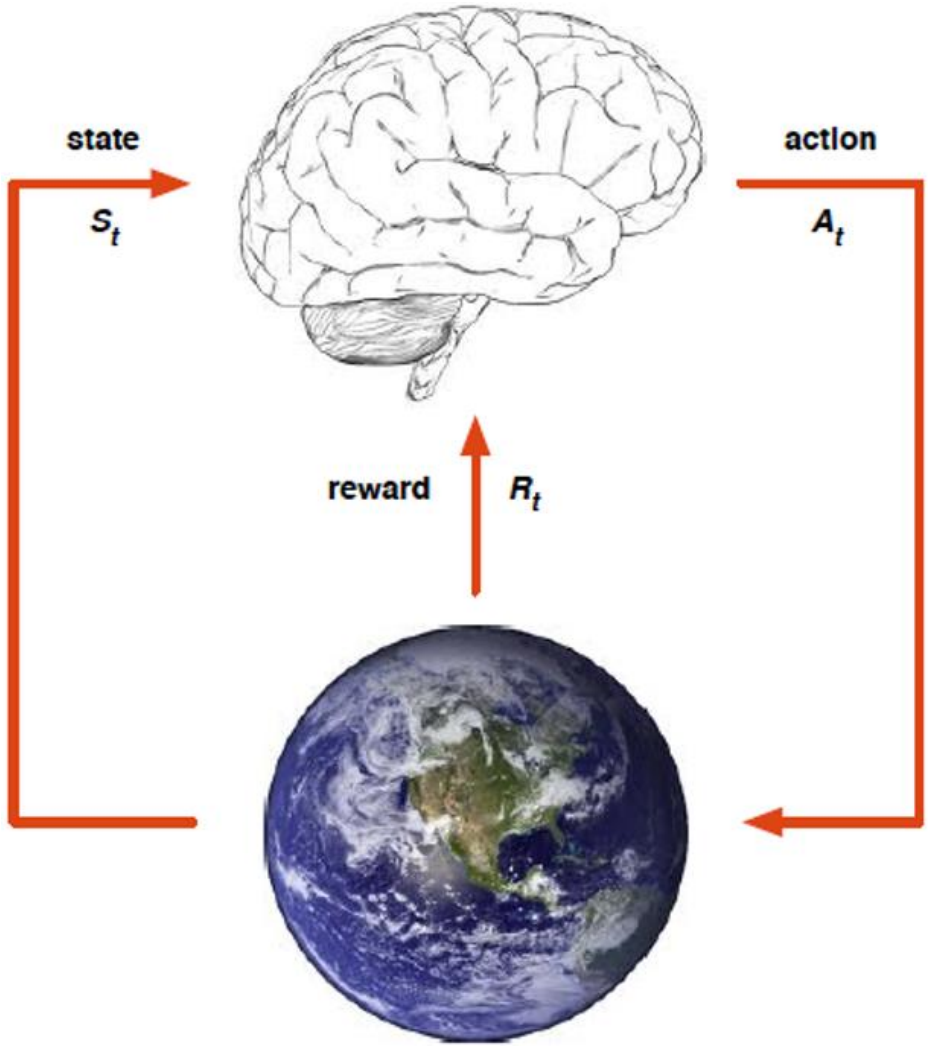
  - The state is a sufficient statistic of the future.

Fully Observable Environment

- Full observability: agent directly observe state:

$$O_t = S_t$$

- Information state = observation.
- Each state must be unique.
- In this case, agent-environment interaction can be formally modeled with a Markov Decision Process (MDP).



Markov Property

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

Definition

A state S_t is *Markov* if and only if

$$\mathbb{P}[S_{t+1} \mid S_t] = \mathbb{P}[S_{t+1} \mid S_1, \ldots, S_t]$$

  - The current state captures all relevant information from the history.
  - Once the current state is known, the history can be thrown away.

State Transition

- For a Markov state s and successor state s', the state transition probability is defined by:

$$\mathcal{P}_{ss'} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s\right]$$

- State transition matrix $\mathcal{P}$ defines transition probabilities from all state to all successor state, where each row sums to 1.

$$\mathcal{P} = \begin{matrix} \\ from \end{matrix} \begin{matrix} to \\ \end{matrix} \begin{bmatrix} \mathcal{P}_{11} & \dots & \mathcal{P}_{1n} \\ \vdots & & \\ \mathcal{P}_{n1} & \dots & \mathcal{P}_{nn} \end{bmatrix}$$

Markov Process

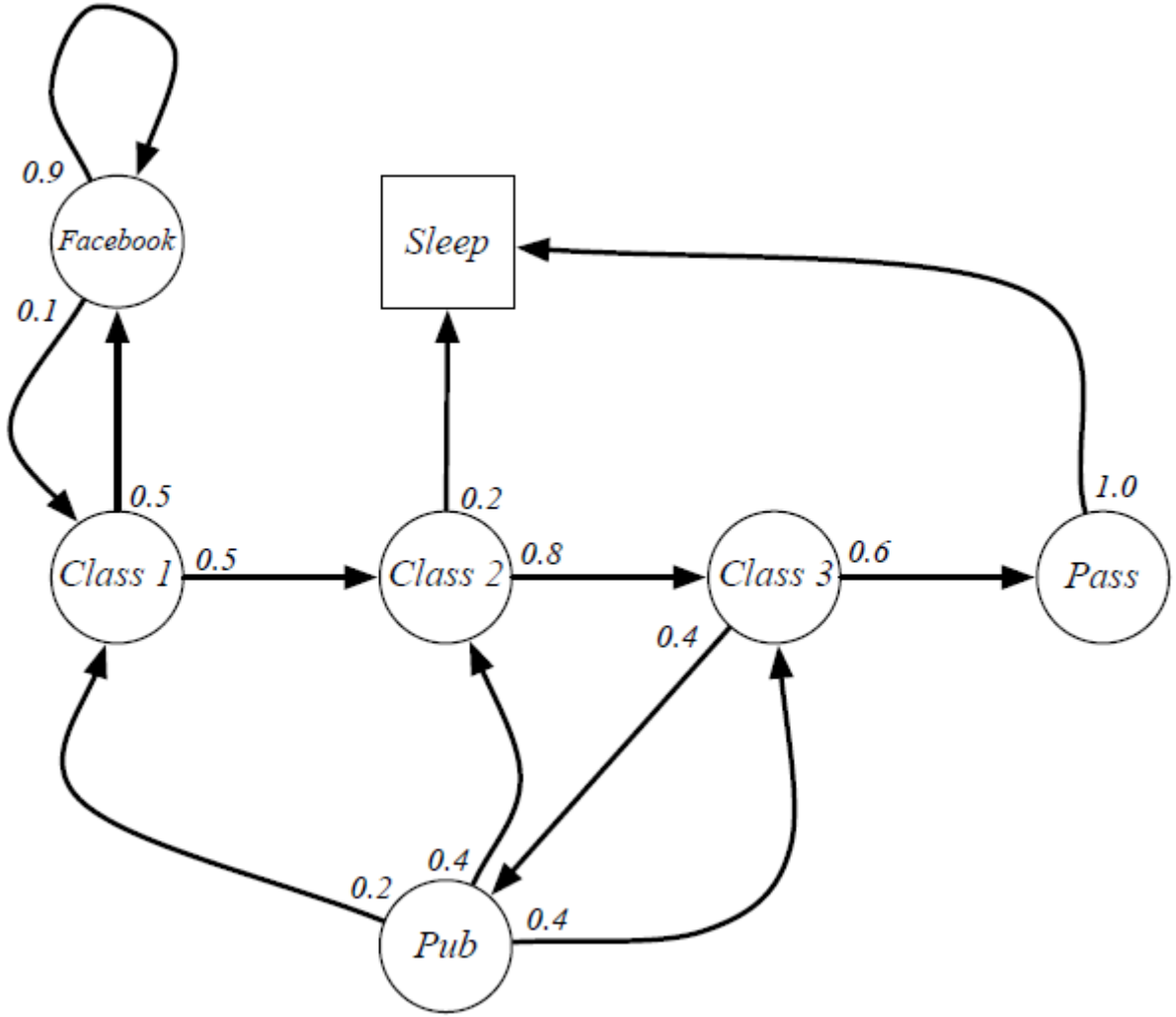
- A Markov process is a memoryless random process, i.e., a sequence of states $S_1, S_2, \cdots S_t$ with the Markov property.

Definition

A *Markov Process* (or *Markov Chain*) is a tuple $\langle \mathcal{S}, \mathcal{P} \rangle$

- $\mathcal{S}$ is a (finite) set of states
- $\mathcal{P}$ is a state transition probability matrix, $\mathcal{P}_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s]$

Markov Process: Example



$$\mathcal{P} = \begin{array}{c} \\ C1 \\ C2 \\ C3 \\ Pass \\ Pub \\ FB \\ Sleep \end{array} \begin{array}{c} \begin{array}{ccccccc} C1 & C2 & C3 & Pass & Pub & FB & Sleep \end{array} \\ \begin{bmatrix} & 0.5 & & & & 0.5 & \\ & & 0.8 & & & & 0.2 \\ & & & 0.6 & 0.4 & & \\ & & & & & & 1.0 \\ 0.2 & 0.4 & 0.4 & & & & \\ 0.1 & & & & & 0.9 & \\ & & & & & & 1 \end{bmatrix} \end{array}$$

- Episodes sampled from the Markov Process starting from C1 to Sleep:
  - C1 C2 C3 Pass Sleep
  - C1 FB FB C1 C2 Sleep
  - C1 C2 C3 Pub C2 C3 Pass Sleep
  - C1 FB FB C1 C2 C3 Pub C1 FB FB FB C1 C2 C3 Pub C2 Sleep

Markov Reward Process

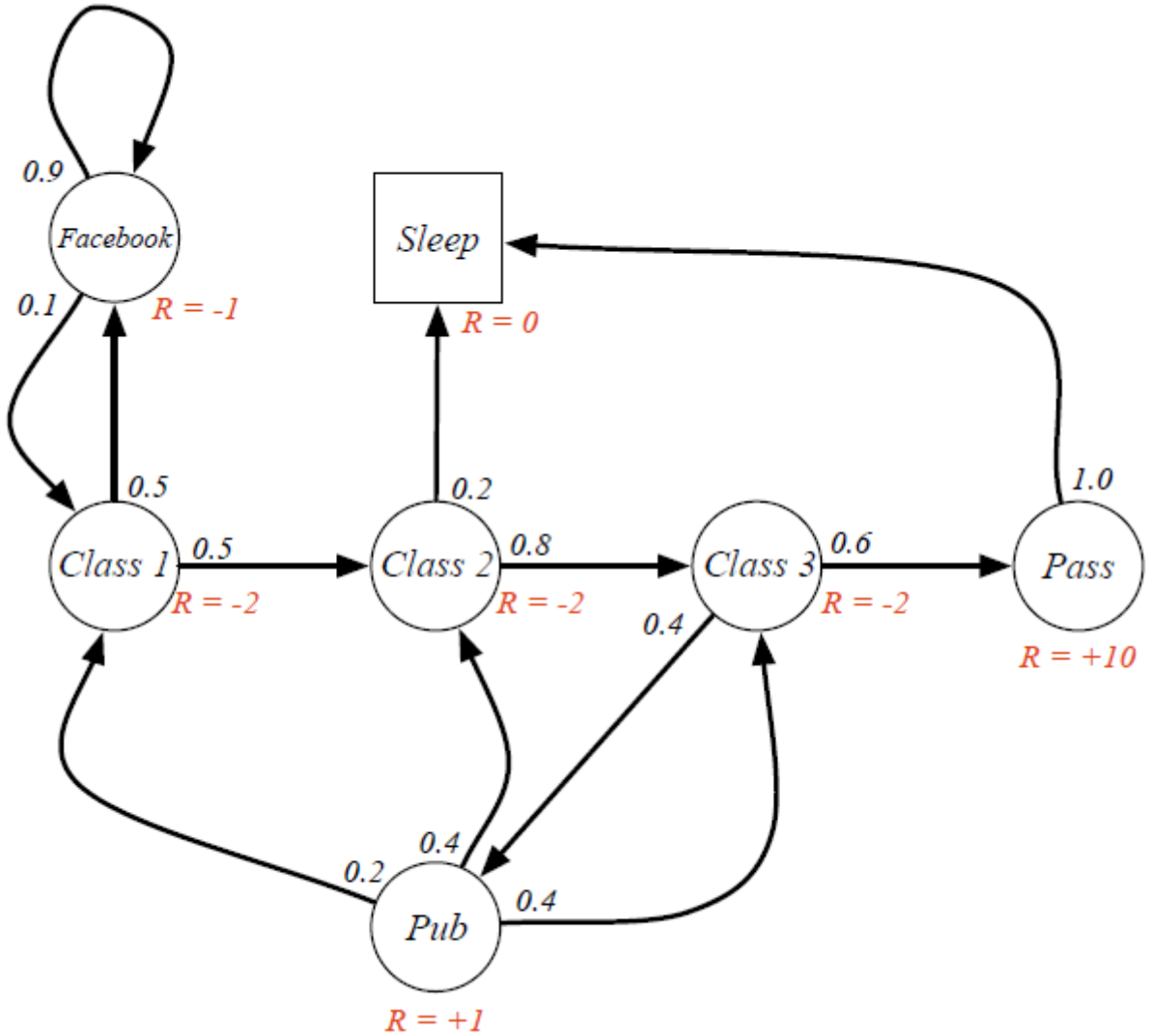
- A Markov reward process is a Markov chain of states with a reward value associated with each state.

Definition

A *Markov Reward Process* is a tuple $\langle \mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma \rangle$

- $\mathcal{S}$ is a finite set of states
- $\mathcal{P}$ is a state transition probability matrix, $\mathcal{P}_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s]$
- $\mathcal{R}$ is a reward function, $\mathcal{R}_s = \mathbb{E}[R_{t+1} \mid S_t = s]$
- γ is a discount factor, $\gamma \in [0, 1]$

Markov Reward Process: Example



Markov Decision Process

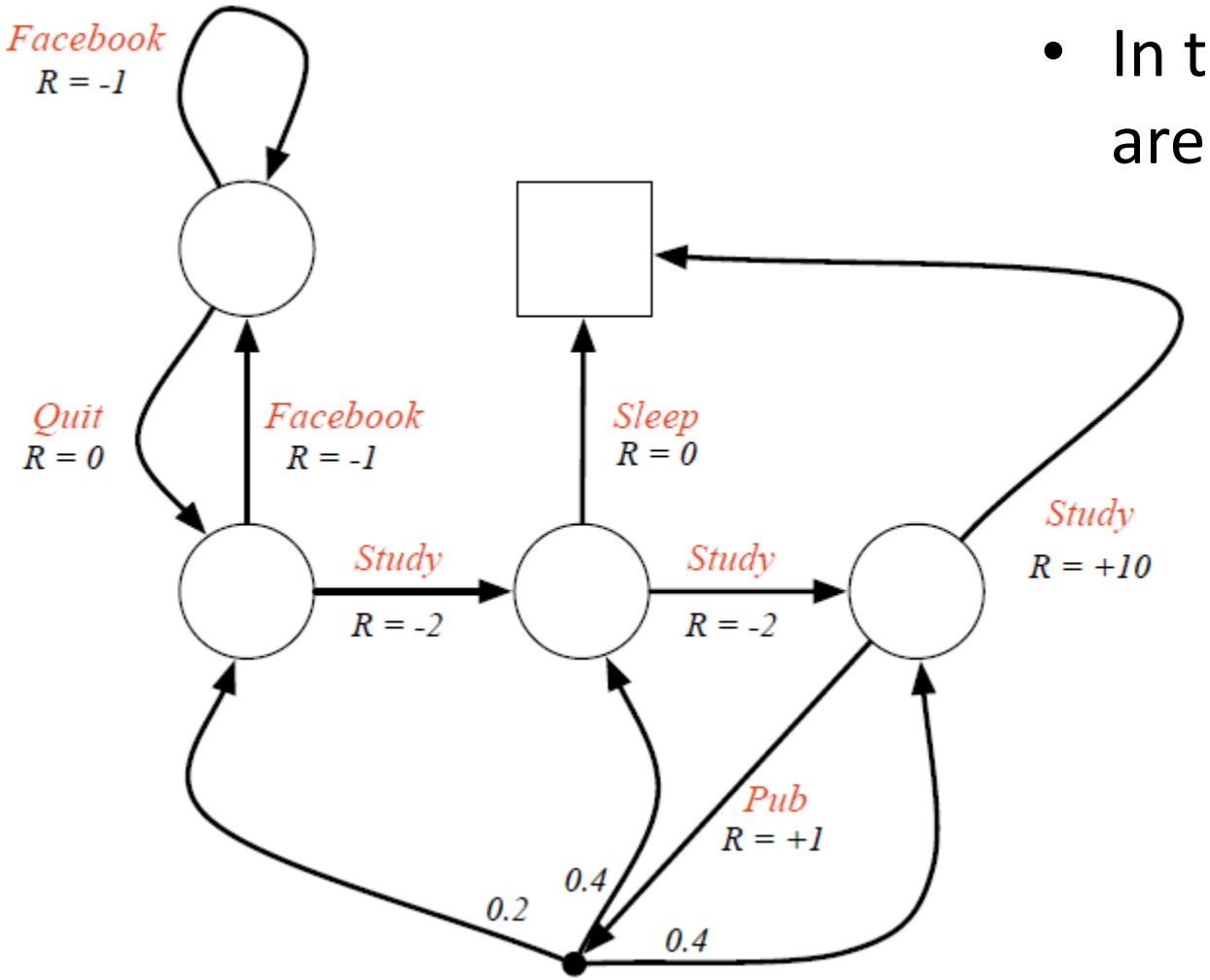
- A Markov decision process (MDP) is a Markov reward process with actions that transit the agent among states.

Definition

A *Markov Decision Process* is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$

- $\mathcal{S}$ is a finite set of states
- $\mathcal{A}$ is a finite set of actions
- $\mathcal{P}$ is a state transition probability matrix, $\mathcal{P}^{a}_{ss'} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s, A_t = a\right]$
- $\mathcal{R}$ is a reward function, $\mathcal{R}^{a}_{s} = \mathbb{E}\left[R_{t+1} \mid S_t = s, A_t = a\right]$
- γ is a discount factor $\gamma \in [0, 1]$.

Markov Decision Process: Example



- In this example, actions are deterministic.

Markov Decision Process: Drag Racing Example

- MDP can also model stochastic actions:

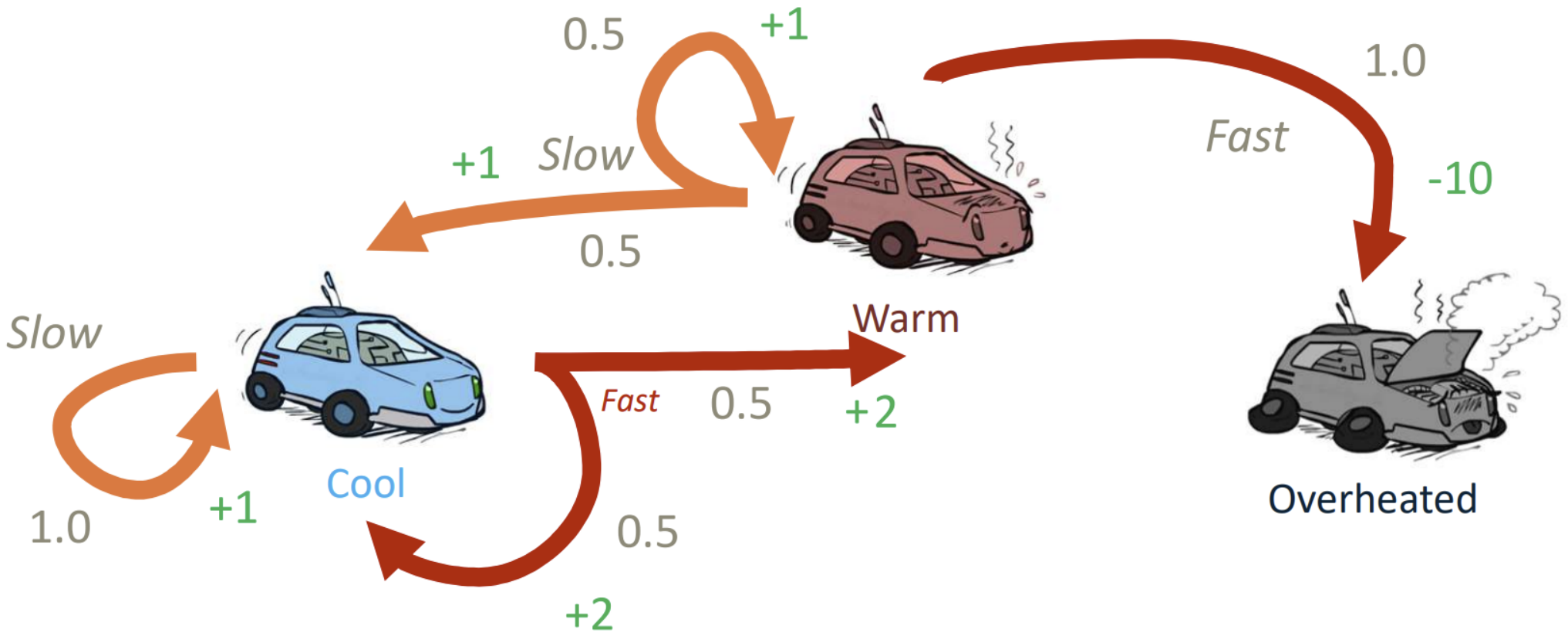


Image modified from http://ai.berkeley.edu

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

MDP and Reinforcement Learning

- A Markov decision process (MDP) formally describes an agent-environment interaction for reinforcement learning (RL):
  - MDPs assume that the environment is fully observable.
    - The current state completely characterizes the process.
  - Almost all RL problems can be formulated under MDPs, for example:
    - Adaptive control primarily deals with continuous MDPs.
    - Partially observable problems can be converted into MDPs.

- An RL approach may include several components:
  - Policy: a function that determines agent actions.
  - Value function: how good each state or state/action pair is.
  - Model: agent's representation of the environment.

RL Components

- Policy

Definition

A *policy* π is a distribution over actions given states,

$$\pi(a|s) = \mathbb{P}[A_t = a \mid S_t = s]$$

- A policy fully defines the action of an agent in each state.
- MDP policies depend on the current state only (not on the history).
- Policies are stationary (time-independent): $A_t \sim \pi(\cdot|S_t), \forall t > 0$
- Policies are can be deterministic (and greedy): $a = \pi(s)$
  or stochastic: $\pi(a|s) = \mathbb{P}[A_t = a|S_t = s]$

RL Components

- Value Function
  - Value function is a prediction of the overall future reward.
  - It is used to evaluate the goodness or badness of each state.
  - It is then used to select the action given each state.

$$v_\pi(s) = \mathbb{E}_\pi \left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \mid S_t = s \right]$$

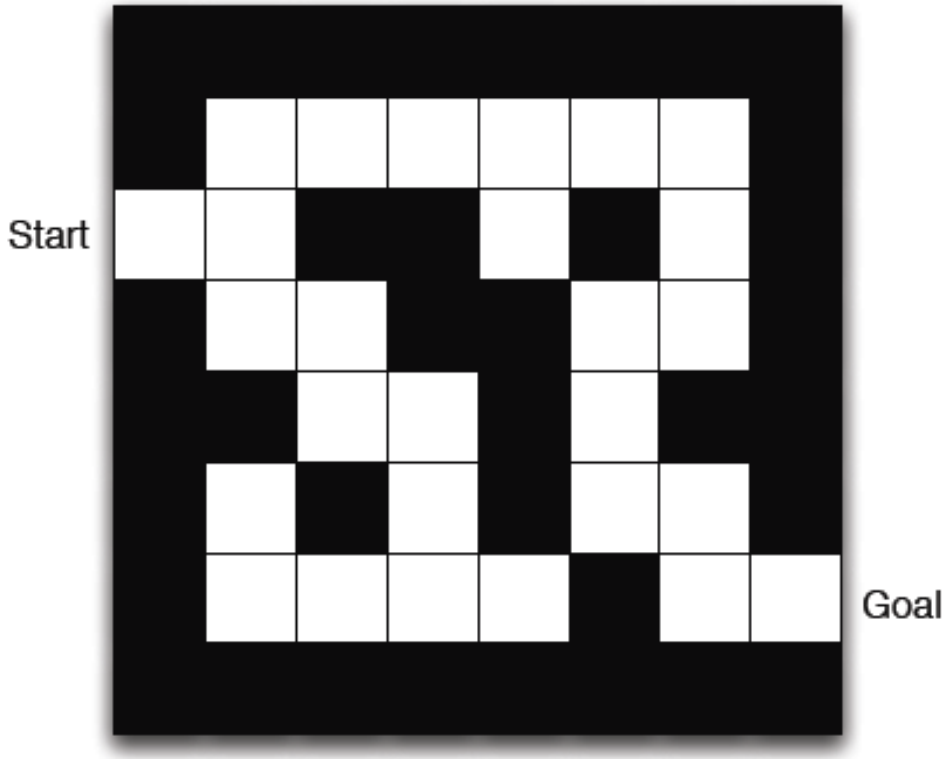
- Model
  - A model represents the environment and predicts what it will do next.
    - The state transition matrix $\mathcal{P}$ predicts the next state:

$$\mathcal{P}^a_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$$

    - The reward function $\mathcal{R}$ predicts the next immediate reward:

$$\mathcal{R}^a_s = \mathbb{E}\left[R_{t+1} \mid S_t = s, A_t = a \right]$$

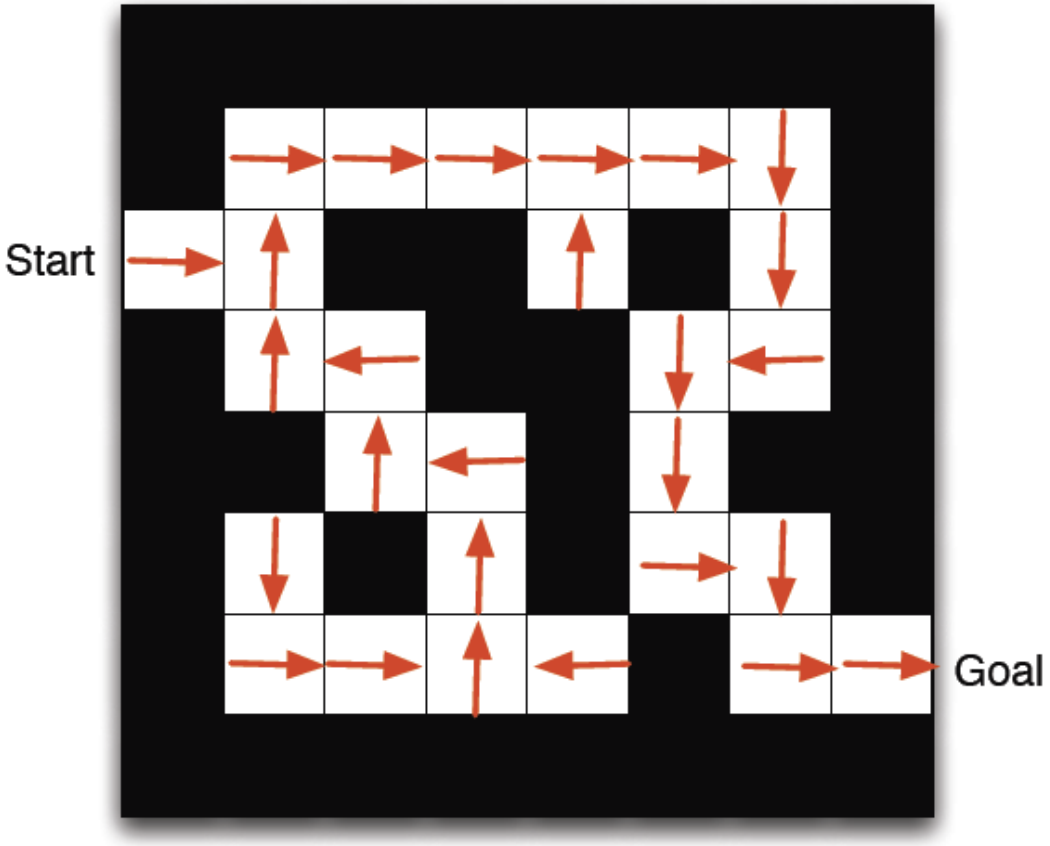
RL Components: Example



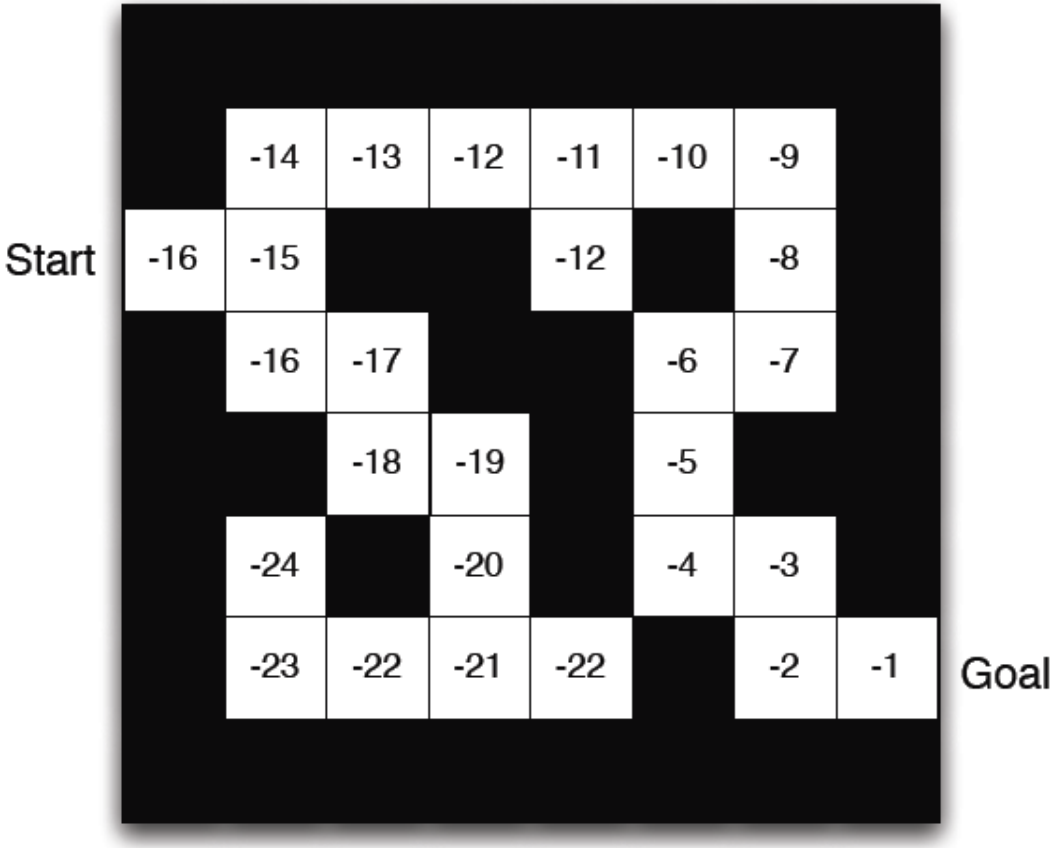
- Rewards: -1 per time-step
- Actions: N, E, S, W
- States: Agent's location

RL Components: Example

Policy



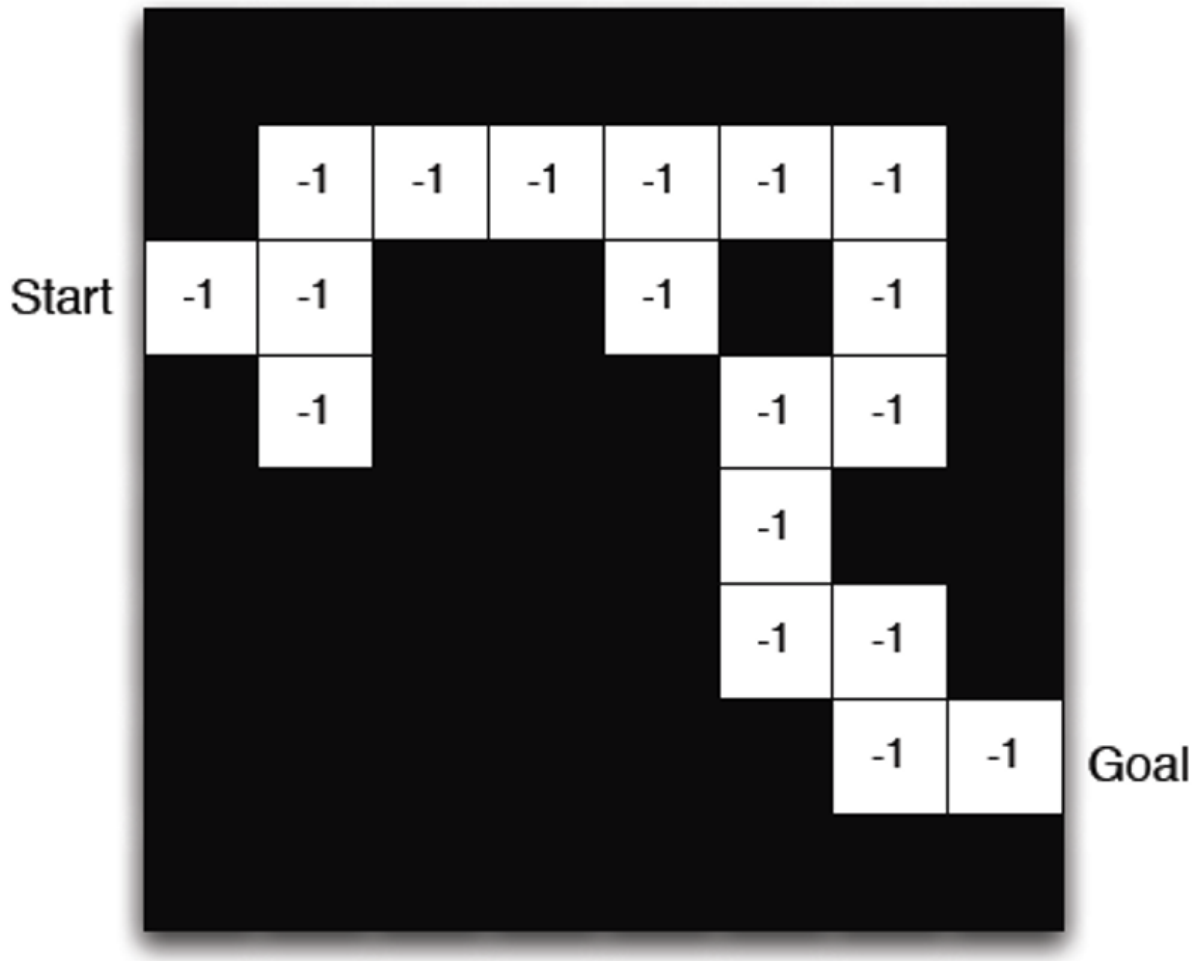
Value Function



- Arrows represent policy $\pi(s)$ for each state s
- Numbers represent value $v_\pi(s)$ of each state s

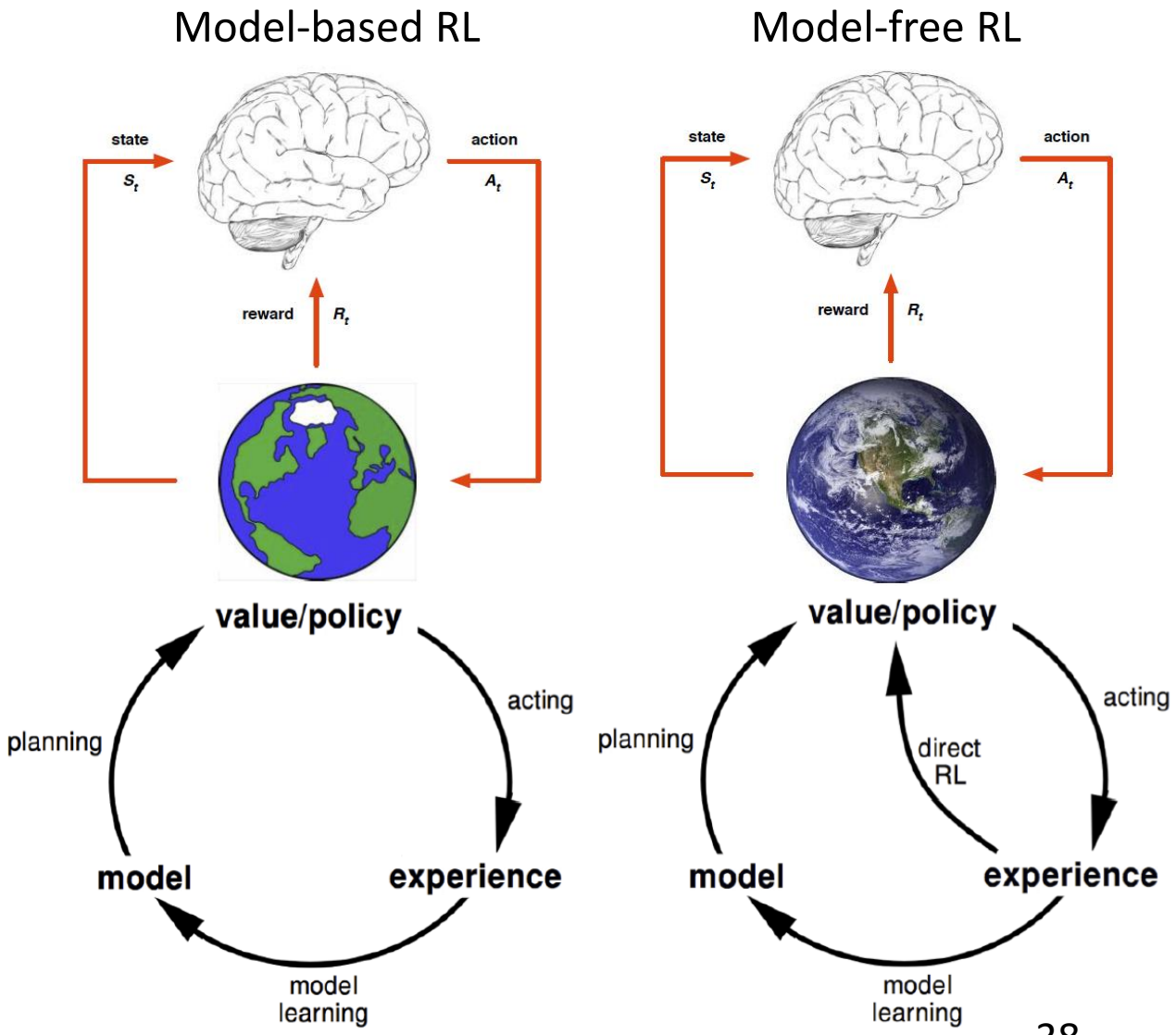
RL Components: Example

- The mode uses the grid map to represent the state transition $\mathcal{P}^a_{ss'}$.
- Numbers encode immediate reward $\mathcal{R}^a_s$ from each state (same for all actions)
- The model may be imperfect.



Model-Based and Model-Free RL

- Model-based RL
  - Learn a model from experience.
  - Plan a value function (and/or policy) from model.
- Model-free RL
  - No model.
  - Learn a value function (and/or policy) from experience.



Q-Learning

- We're going to learn a model-free RL (although knowing a model also works).
- We will focus on finding a way to directly estimate the value function.
  - The value function is not necessary to directly associate with the world and represent the world.

- The value function is the **Q-function.**
  - A recursive way to approximate a value function.
- The process of estimating the Q-function is called **Q-learning.**
  - Q-learning integrates learning and planning.

Q-Learning

- Given a sequence of states, actions, and rewards defined by an MDP:

$$s_0 a_0 r_0\, s_1 a_1 r_1\, s_2 a_2 r_2\, s_3 a_3 r_3 \ldots\, s_k a_k r_k \ldots$$

  we define a unit experience as $< s_k\, a_k\, r_k\, s_{k+1} >$.

- At each step *s*, choose the action *a* that maximizes the Q-function Q(s, a).
  - Q is the estimated value function.
  - It tells us how good an action is given a state.
  - Q(s, a) = immediate reward for making an action + best value (Q) from the resulting (future) states.

Q-Learning: Mathematical Formulation

- Q-function has a recursive formulation:

$$Q(s,a) = r(s,a) + \gamma \, max_{a'}(Q(s',a'))$$

$r(s,a)$ = Immediate reward

γ = relative value of delayed vs. immediate rewards (0 to 1)

s' = the new state after action a

a, a' : actions in states s and s', respectively

Selected action:

$$\pi(s) = argmax_a Q(s,a)$$

- Q-learning estimates the table of Q-values, called **Q-table**, which updates Q-values related to the state-action pairs that are visited.

Q-Learning: Algorithm

- The Q-Learning algorithm is recursive, using the unit experience:

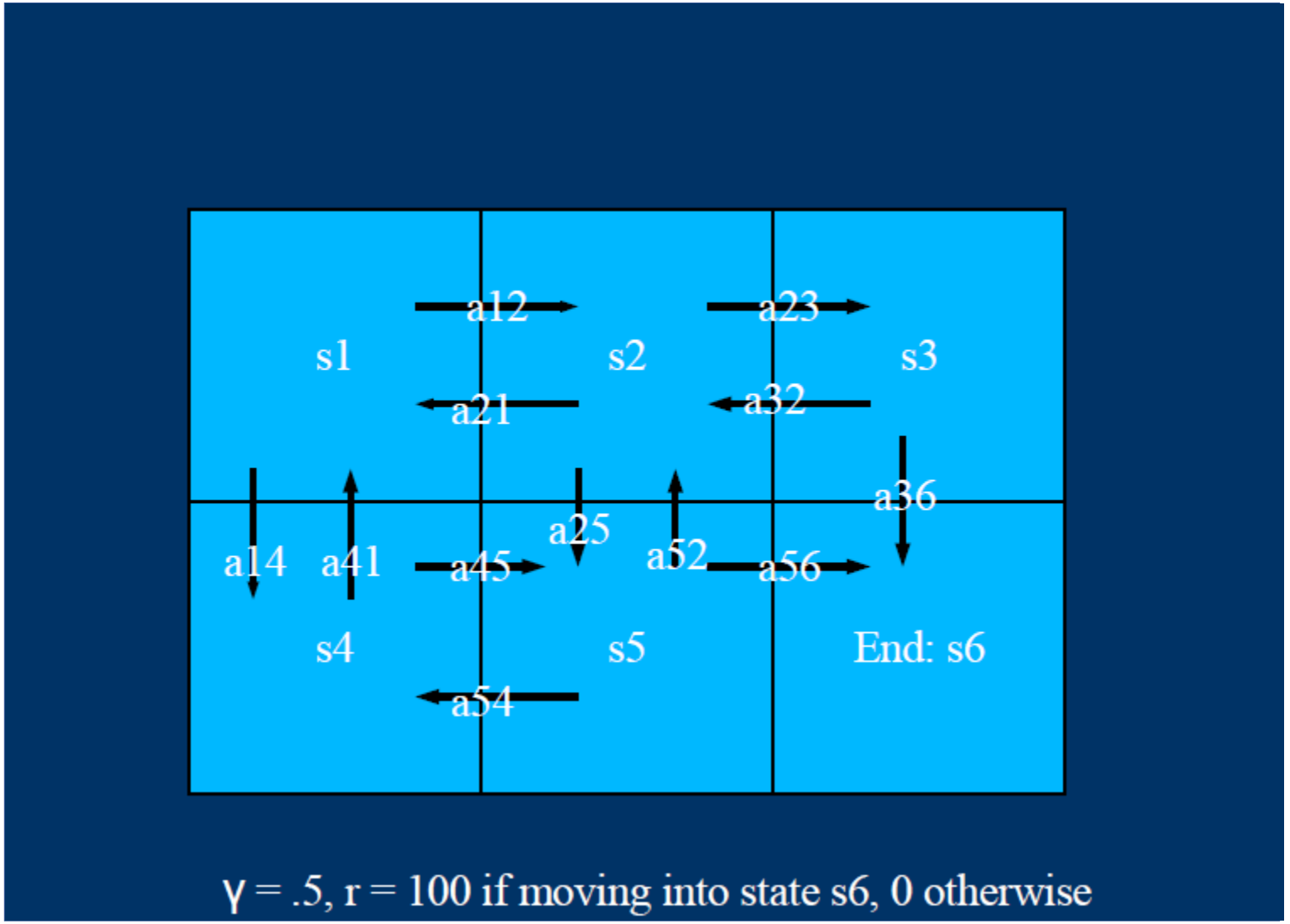
$$< s_k\ a_k\ r_k\ s_{k+1} >$$

For each state-action pair (s, a), initialize the table entry $\hat{Q}(s, a)$ to zero
Observe the current state s
Do forever:
---Select an action a and execute it
---Receive immediate reward r
---Observe the new state s'
---Update the table entry for $\hat{Q}(s, a)$ as follows:

$$\hat{Q}(s, a) = r + \gamma \max_{a'} \hat{Q}(s', a')$$

--- $s = s'$

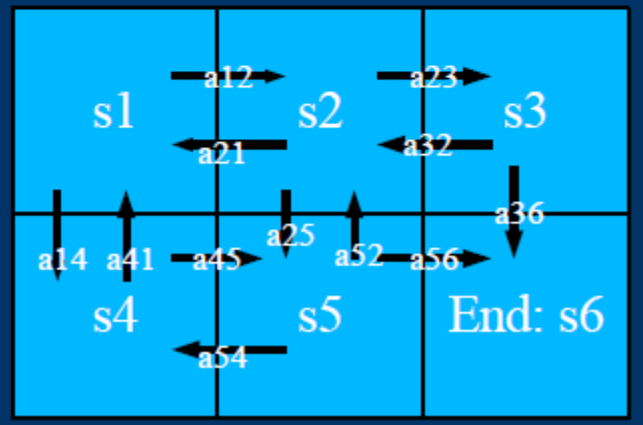
Q-Learning: Example



Q-Learning: Example

Initial State

s1, a12	0
s1, a14	0
s2, a21	0
s2, a23	0
s2, a25	0
s3, a32	0
s3, a36	0
s4, a41	0
s4, a45	0
s5, a54	0
s5, a52	0
s5, a56	0



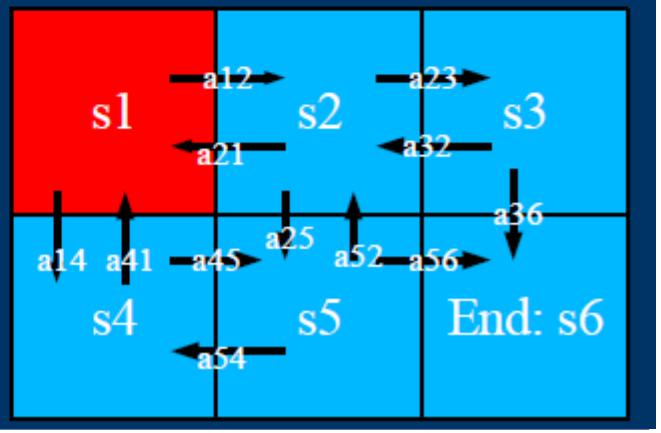
Q-Learning: Example

The Algorithm

s1, a12	0
s1, a14	0
s2, a21	0
s2, a23	0
s2, a25	0
s3, a32	0
s3, a36	0
s4, a41	0
s4, a45	0
s5, a54	0
s5, a52	0

Current Position: Red

Available actions: a12, a14
Chose a12



Q-Learning: Example

Update Q(s1, a12)

s1, a12	0
s1, a14	0
s2, a21	0
s2, a23	0
s2, a25	0
s3, a32	0
s3, a36	0
s4, a41	0
s4, a45	0
s5, a54	0
s5, a52	0

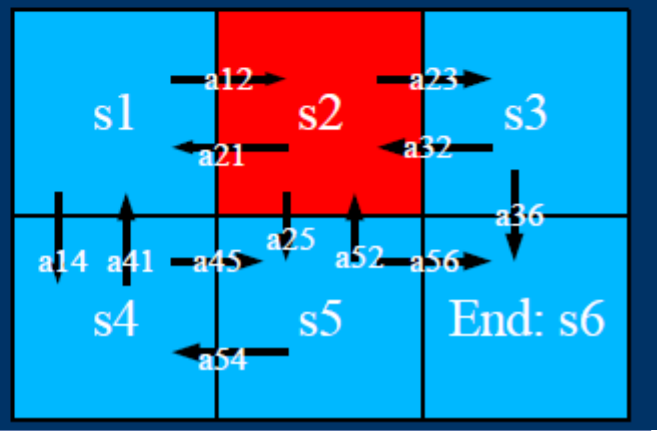
Current Position: Red

Available actions: a21, a25, a23

$$\hat{Q}(s,a) = r + \gamma \max_{a'} \hat{Q}(s',a')$$

Update Q(s1, a12):
Q(s1, a12) = r + .5 * max(Q(s2,a21), Q(s2,a25), Q(s2,a23))

= 0



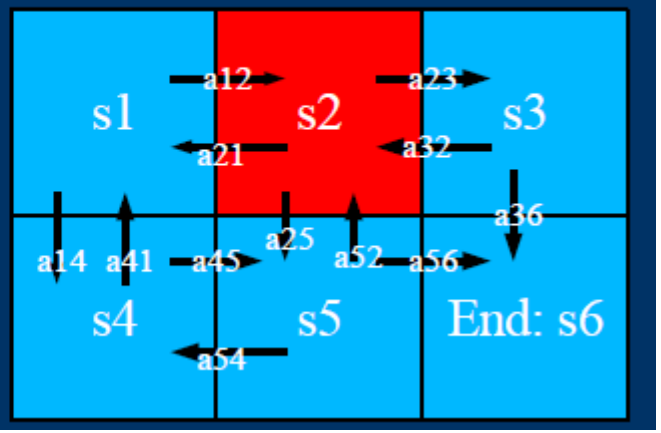
Q-Learning: Example

Next Move

s1, a12	0
s1, a14	0
s2, a21	0
s2, a23	0
s2, a25	0
s3, a32	0
s3, a36	0
s4, a41	0
s4, a45	0
s5, a54	0
s5, a52	0

Current Position: Red

Available actions: a21, a25, a23
Chose a23



Q-Learning: Example

Update Q(s2, a23)

s1, a12	0
s1, a14	0
s2, a21	0
s2, a23	0
s2, a25	0
s3, a32	0
s3, a36	0
s4, a41	0
s4, a45	0
s5, a54	0
s5, a52	0

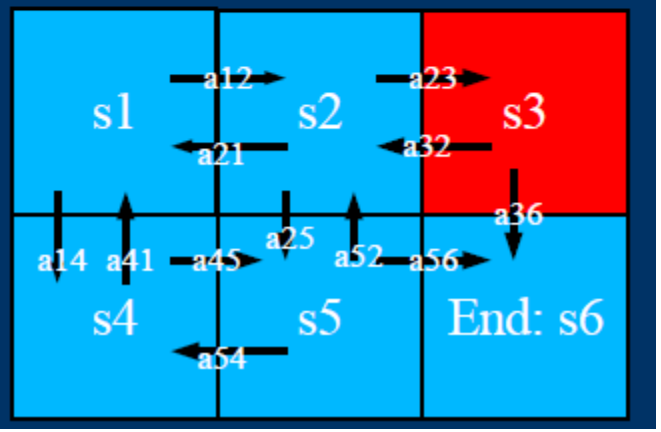
Current Position: Red

Available actions: a32, a36

Update Q(s2, a23):
Q(s2, a23) = r + .5 * max(Q(s3,a32), Q(s3,a36))

= 0

$$\hat{Q}(s,a) = r + \gamma \max_{a'} \hat{Q}(s',a')$$



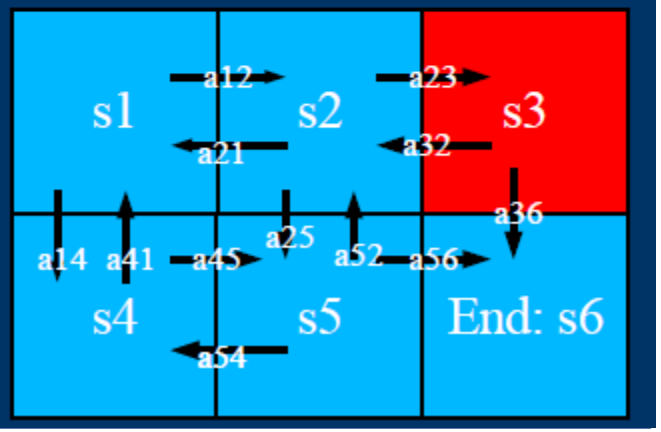
Q-Learning: Example

Next Move

s1, a12	0
s1, a14	0
s2, a21	0
s2, a23	0
s2, a25	0
s3, a32	0
s3, a36	0
s4, a41	0
s4, a45	0
s5, a54	0
s5, a52	0

Current Position: Red

Available actions: a32, a36
Chose a36



Q-Learning: Example

Update Q(s3,a36)

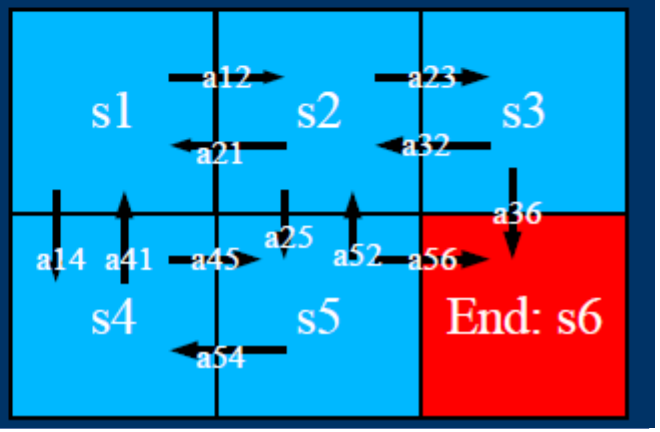
s1, a12	0
s1, a14	0
s2, a21	0
s2, a23	0
s2, a25	0
s3, a32	0
s3, a36	100
s4, a41	0
s4, a45	0
s5, a54	0
s5, a52	0

Current Position: Red

FINAL STATE!

Update Q(s3, a36):
Q(s3, a36) = r = 100

$$\hat{Q}(s,a) = r + \gamma \max_{a'} \hat{Q}(s',a')$$



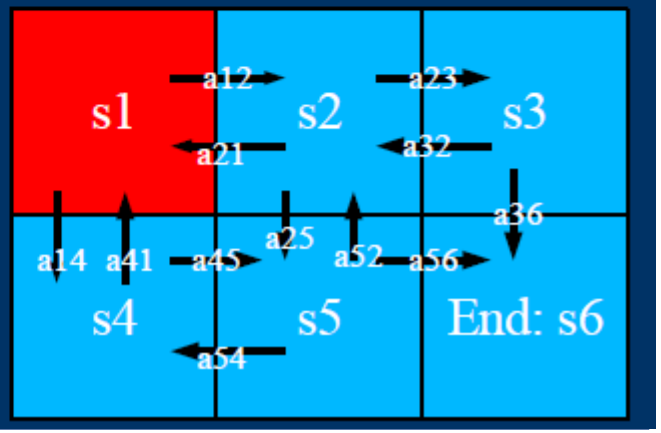
Q-Learning: Example

The Algorithm

s1, a12	0
s1, a14	0
s2, a21	0
s2, a23	0
s2, a25	0
s3, a32	0
s3, a36	0
s4, a41	0
s4, a45	0
s5, a54	0
s5, a52	0

Current Position: Red

Available actions: a12, a14
Chose a12



Q-Learning: Example

New Episode **Update Q(s1, a12)**

s1, a12	0
s1, a14	0
s2, a21	0
s2, a23	0
s2, a25	0
s3, a32	0
s3, a36	100
s4, a41	0
s4, a45	0
s5, a54	0
s5, a52	0

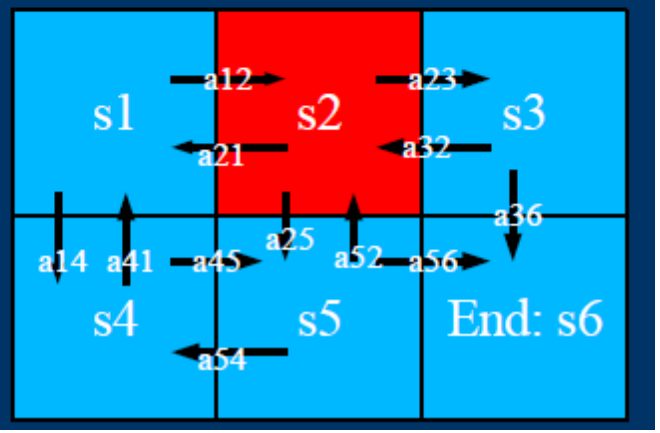
Current Position: Red

Available actions: a21, a25, a23

$$\hat{Q}(s,a) = r + \gamma \max_{a'} \hat{Q}(s',a')$$

Update Q(s1, a12):
Q(s1, a12) = r + .5 * max(Q(s2,a21), Q(s2,a25), Q(s2,a23))

= 0



Q-Learning: Example

Update Q(s2, a23)

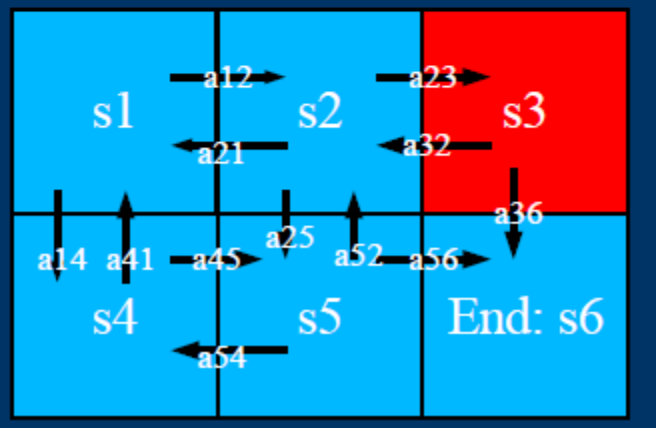
s1, a12	0
s1, a14	0
s2, a21	0
s2, a23	50
s2, a25	0
s3, a32	0
s3, a36	100
s4, a41	0
s4, a45	0
s5, a54	0
s5, a52	0

Current Position: Red

Available actions: a32, a36

Update Q(s2, a23):
Q(s2, a23) = r + .5 * max(Q(s3,a32), Q(s3,a36))
= 0 + .5 * 100 = 50

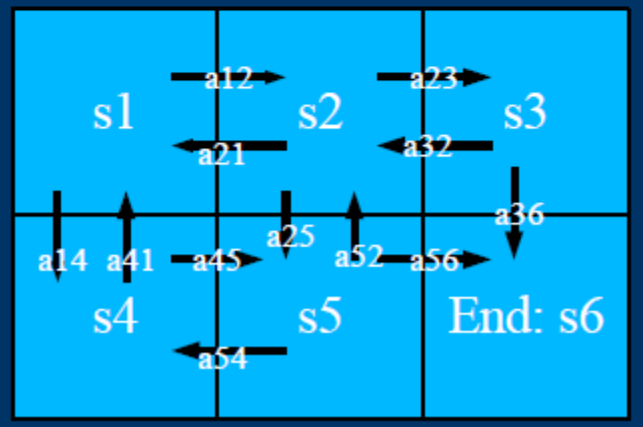
$$\hat{Q}(s,a) = r + \gamma \max_{a'} \hat{Q}(s',a')$$

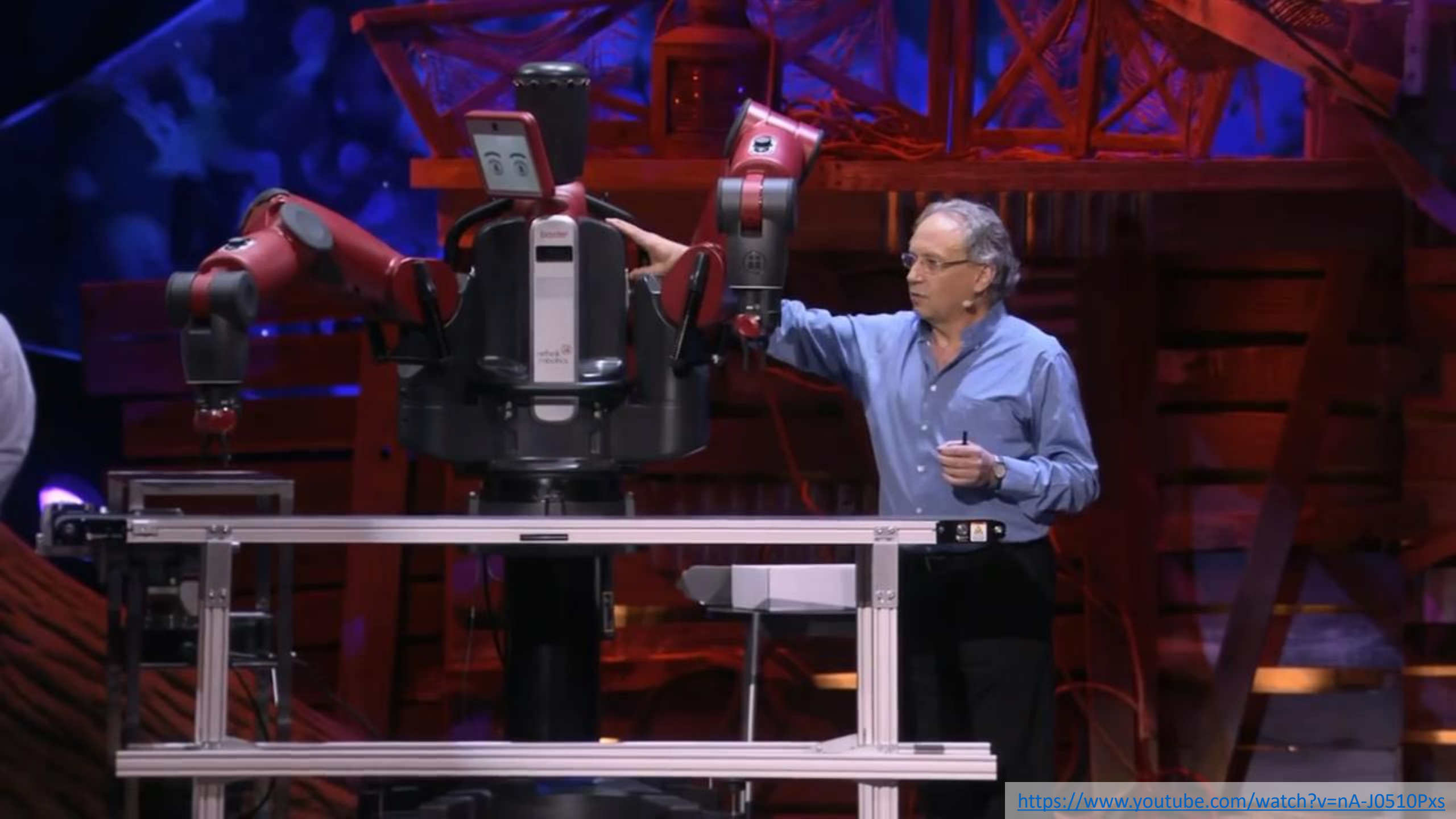


Q-Learning: Example

Final State (after many iterations)

s1, a12	25
s1, a14	25
s2, a21	12.5
s2, a23	50
s2, a25	25
s3, a32	25
s3, a36	100
s4, a41	12.5
s4, a45	50
s5, a54	25
s5, a52	25
s5, a56	100





https://www.youtube.com/watch?v=nA-J0510Pxs

Q-Learning: Algorithm

- Two problems:

For each state-action pair (s, a), initialize the table entry $\hat{Q}(s, a)$ to zero
Observe the current state s
Do forever:
---Select an action a and execute it (Greedy action selection)
---Receive immediate reward r
---Observe the new state s'
---Update the table entry for $\hat{Q}(s, a)$ as follows:

$$\hat{Q}(s, a) = r + \gamma \max_{a'} \hat{Q}(s', a')$$

--- $s = s'$ (Complete overwriting old Q values)

Action Selection by ε-greedy Policies

- The ε-greedy policy is most widely used (for both on- and off-policy) to choose an action given a state:

> **ε-greedy policy:**
>
> 1. Generate a random number $r \in [0,1]$
> 2. If $r > \varepsilon$, choose an action derived from the Q values (which yields the maximum reward)
> 3. Else, choose a random action

- The value of ε *determines the exploration-exploitation of the agent.*
  - A larger ε results in more exploration and less exploitation.
  - As a rule of thumb, ε is usually chosen to be close to 1 and decreased over time.

Temporal Difference Update

- Temporal Difference (TD) algorithms enable the agent to *incrementally update* its Q-table through every single action it takes.

$$\text{NewEstimate} \leftarrow \text{OldEstimate} + \text{StepSize}\big[\text{Target} - \text{OldEstimate}\big]$$

  - The value *Target-OldEstimate* is called the target error.
  - *StepSize* is usually denoted by α is also called the **learning rate**, between 0 and 1, and 1 means completely overwrites the old Q value.
- With the temporal difference update, Q-learning becomes:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\Big].$$

Temporal Difference Update

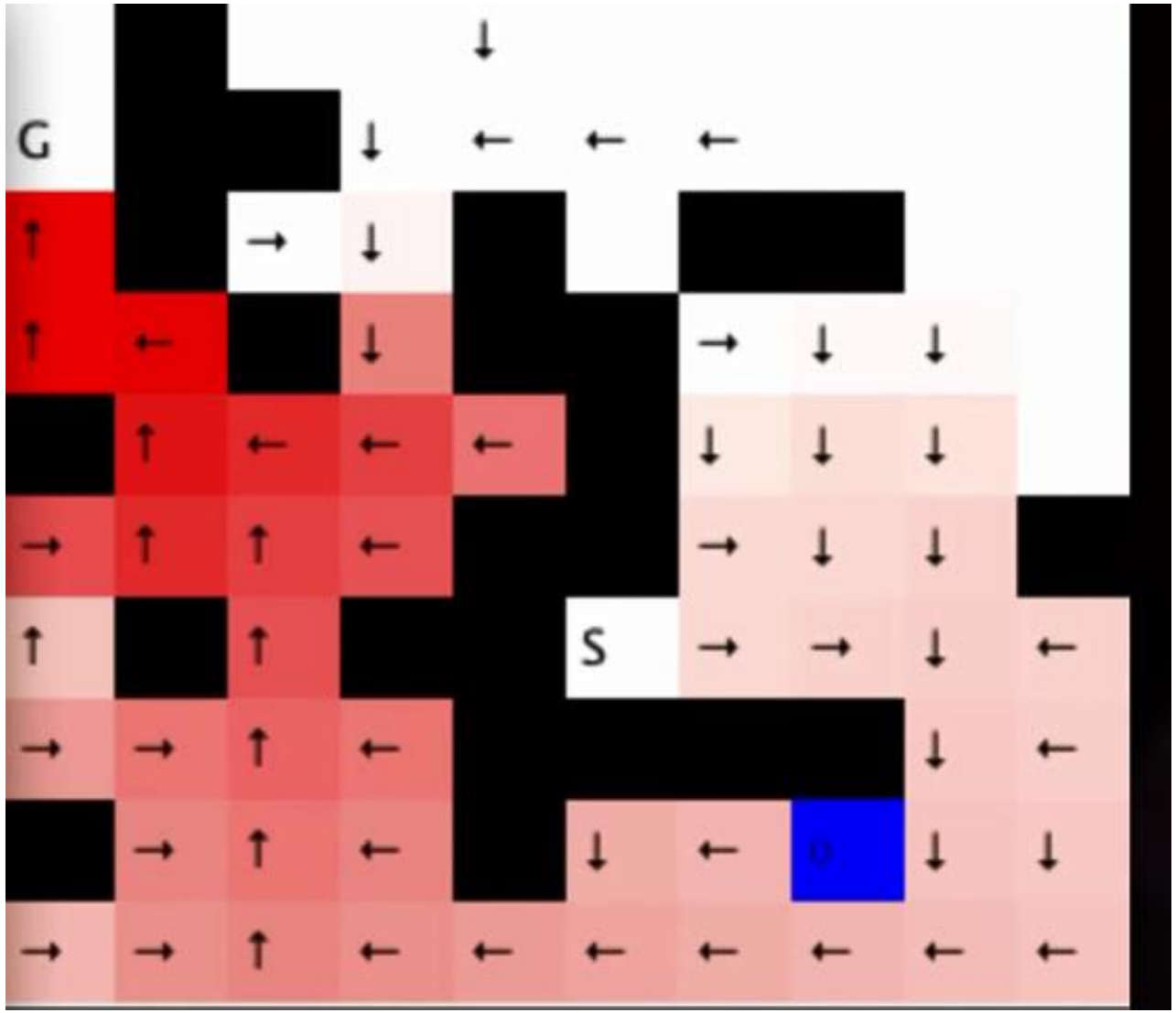
- Q-Learning is an **off-policy** learning algorithm, because:
  - It directly finds the optimal Q-value without any dependency on the policy being followed (due to the max operation).

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Initialize $Q(s,a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
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, S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$
    $S \leftarrow S'$
  until S is terminal

Ref: Introduction to Reinforcement learning by Sutton and Barto - Chapter 6.8

Q-Learning: A Visual Demonstration



SARSA

- SARSA is acronym for State-Action-Reward-State-Action.
- SARSA is an **on-policy** TD learning algorithm, because:
  - It evaluates and improves the same policy that is being used to select actions.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\Big].$$

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
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 R, S'
    Choose A' from S' using policy derived from Q (e.g., ϵ-greedy)
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

Ref: Introduction to Reinforcement learning by Sutton and Barto - Chapter 6.7

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Initialize $Q(s,a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
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, S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
    $S \leftarrow S'$
  until S is terminal

Ref: Introduction to Reinforcement learning by Sutton and Barto - Chapter 6.8

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Initialize $Q(s,a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
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 R, S'
    Choose A' from S' using policy derived from Q (e.g., ϵ-greedy)
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma Q(S',A') - Q(S,A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

Ref: Introduction to Reinforcement learning by Sutton and Barto - Chapter 6.7

Difficulties of RL on Real Robots

- When the number of states and actions becomes larger, the Q-table becomes intractable, and Q-learning easily suffers from the curse of dimensionality:
  - The amount of memory required to save and update the Q-table would increase as the number of states and actions increases.
  - The amount of time required to explore each state to create the required Q-table would be unrealistic.
- Design of states, actions, and rewards is not trivial in robotics applications:
  - States/actions are typically continuous variables in robotics applications.
  - Reward definition often requires significant expert or domain knowledge.

Difficulties of RL on Real Robots

- RL algorithms are notoriously difficult to train for real robots.
  - Sample efficiency and operation safety.
  - Convergence and reliability due to huge exploration space.
  - Sim-to-real gaps.
  - Generalizability to changes in the environment and robot configurations.

https://www.youtube.com/watch?v=iaF43Ze1oeI