Basics of Reinforcement Learning

Overview

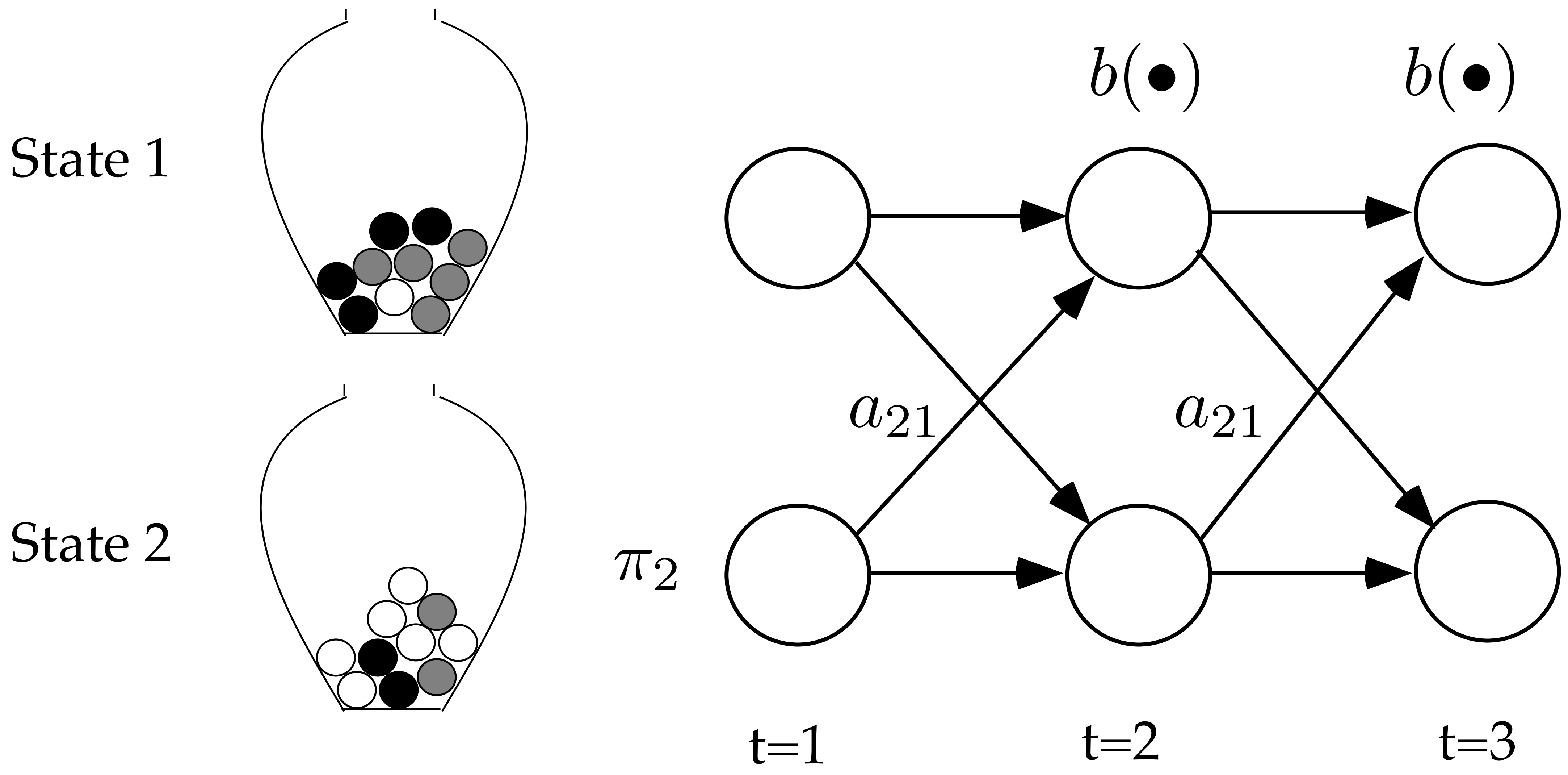
HMMs The probability transformation matrix a_{ij} allows forward and backward propagation

Dynamic programing An objective function and dynamics uses backward propagation

Reinforcement Learning A reward function and probabilistic dynamics uses forward propagation with iteration

Model-free Reinforcement Learning A reward function and probabilistic dynamics uses forward propagation with search for model estimation

The Urn model of a HMM



Dynamic Programing

Final value problem

The dynamics is expressed by a difference equation,

$$\mathbf{x}(k+1) = f[\mathbf{x}(k), \mathbf{u}(k)]$$

The initial condition is:

$$\mathbf{x}(0) = \mathbf{x}_0$$

The allowable control is also discrete:

$$\mathbf{u}(k) \in U, k = 0, \ldots, \ T$$

The integral in the objective function is expressed as a sum:

$$J = \psi[\mathbf{x}(T)] + \sum_{0}^{T-1} \ell[\mathbf{u}(k), \mathbf{x}(k)]$$

Solution: discretize the state space and work backwards

Let V(k) keep track of estimates of the objective function J

At the end. when k=T $\quad V(\boldsymbol{x}, N) = \psi[\boldsymbol{x}(T)]$

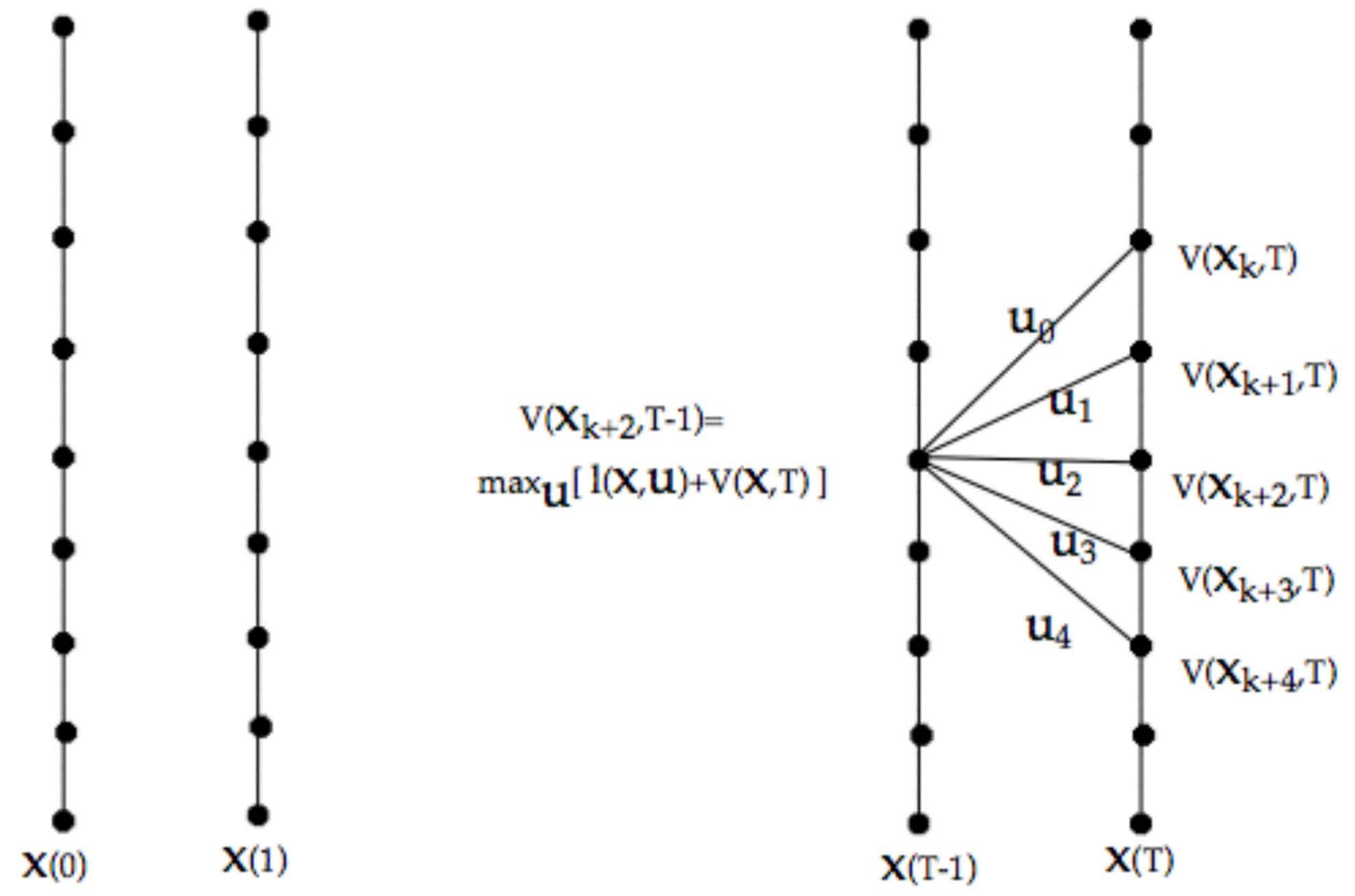
One step back,

$$V(\boldsymbol{x}, T-1) = \max_{\boldsymbol{u} \in U}\{\ell[\boldsymbol{u}(T-1), \boldsymbol{x}(T-1)] + \psi[\boldsymbol{x}(T)]\}$$

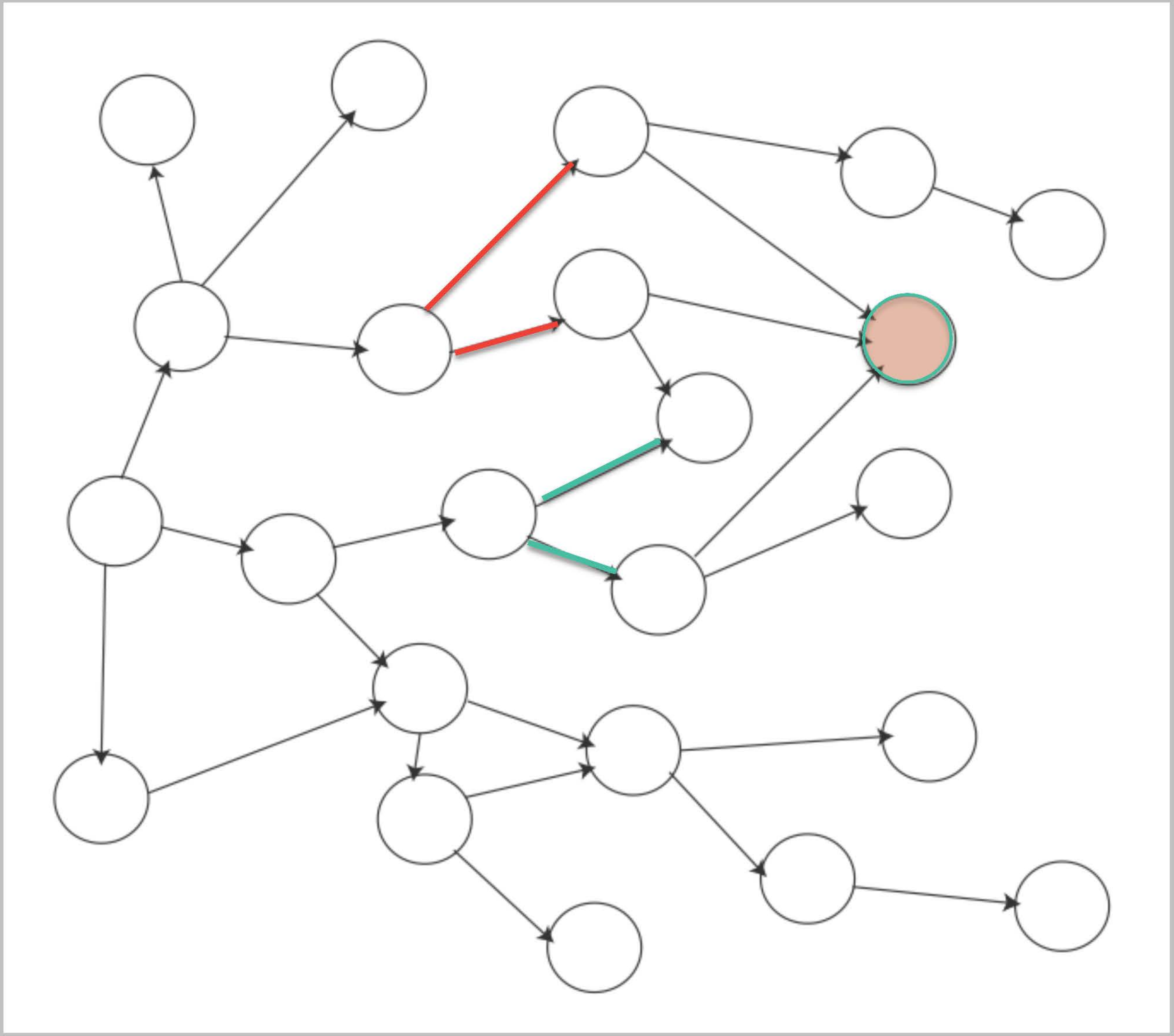
And in general, for $k < T-1$,

$$V(\boldsymbol{x}, k-1) = \max_{\boldsymbol{u} \in U}\{\ell[\boldsymbol{u}(k-1), \boldsymbol{x}(k-1)] + V[\boldsymbol{x}(k), k]\}, k = 0, \ldots, T$$

Discretizing the state to a fine scale is very expensive. *Bellman*, the originator of Dynamic Programming called it **"The curse of dimensionality"**



Basic RL Problems



Location of reward uncertain

Transitions between states uncertain

Policy constantly changing

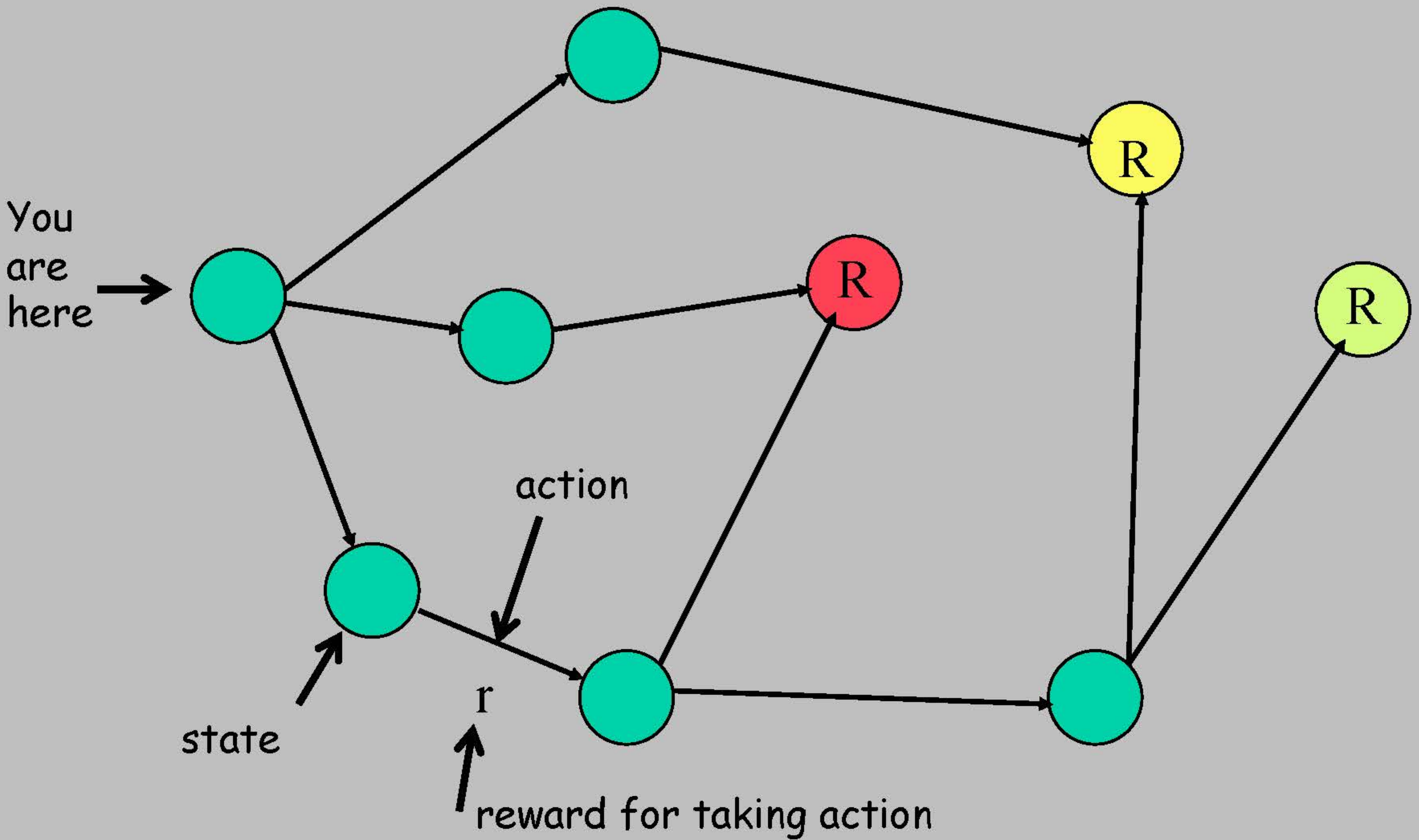
Markov Decision Processes

Problems with delayed reinforcement are well modeled as *Markov decision processes* (MDPs). An MDP consists of

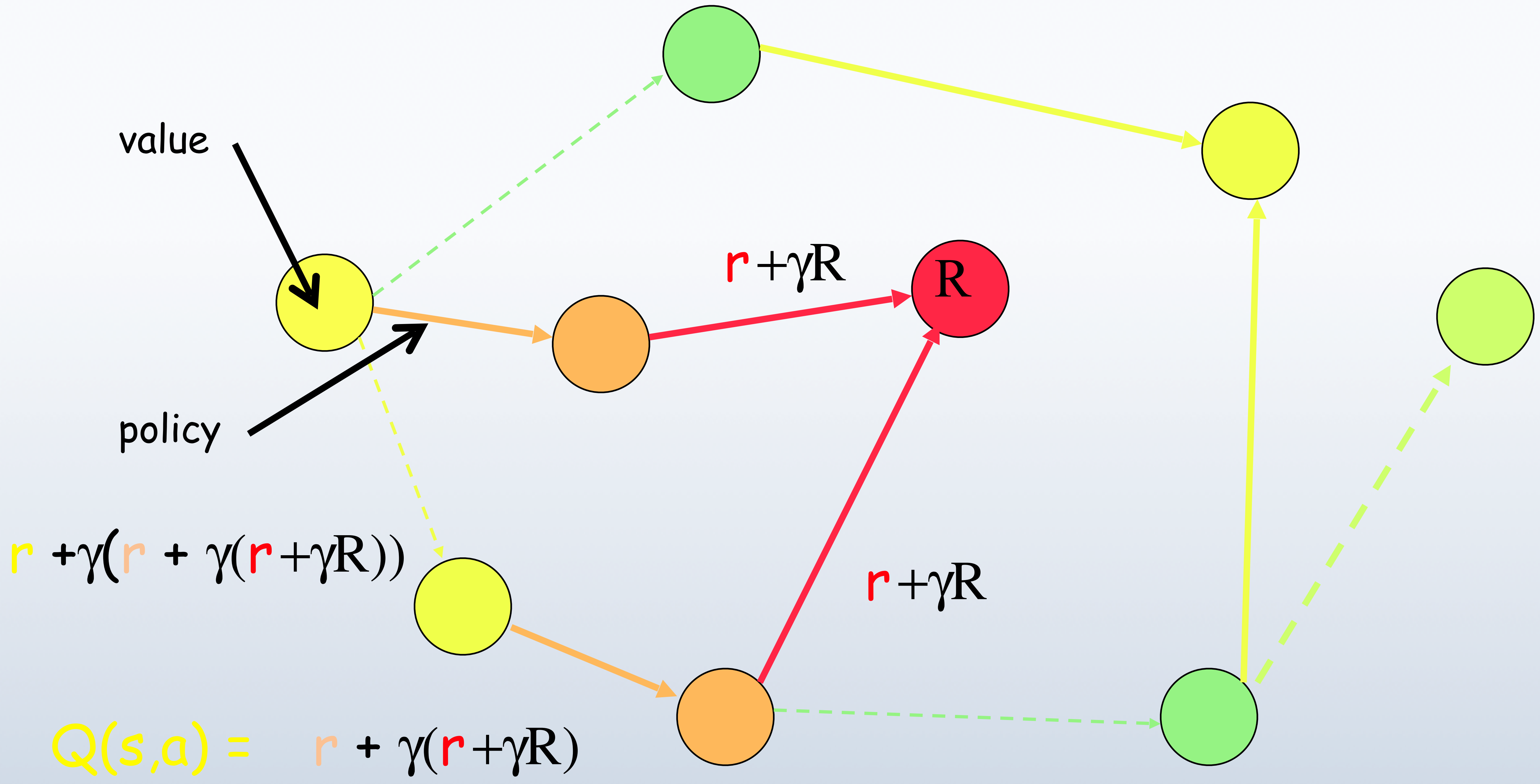
- a set of states $\mathcal{S}$,
- a set of actions $\mathcal{A}$,
- a reward function $R : \mathcal{S} \times \mathcal{A} \rightarrow \Re$, and
- a state transition function $T : \mathcal{S} \times \mathcal{A} \rightarrow \Pi(\mathcal{S})$, where a member of $\Pi(\mathcal{S})$ is a probability distribution over the set $\mathcal{S}$ (i.e. it maps states to probabilities). We write $T(s,a,s')$ for the probability of making a transition from state s to state s' using action a.

The state transition function probabilistically specifies the next state of the environment as a functio of its current state and the agent's action. The reward function specifies expected instantaneous rewa as a function of the current state and action. The model is *Markov* if the state transitions are independent of any previous environment states or agent actions. There are many good references to MDP models

Reinforcement Learning Primer : Before Learning

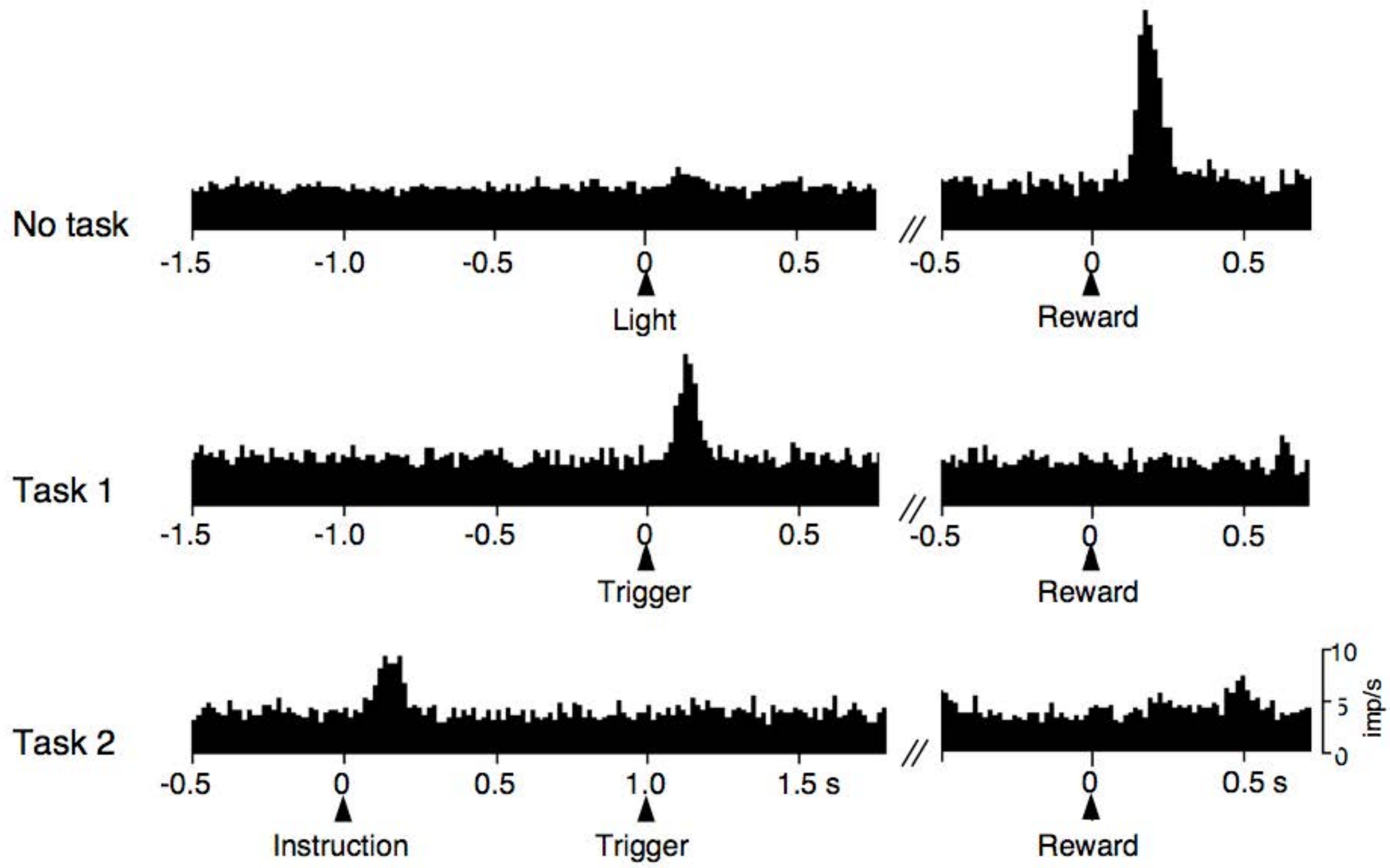


Reinforcement Learning Primer

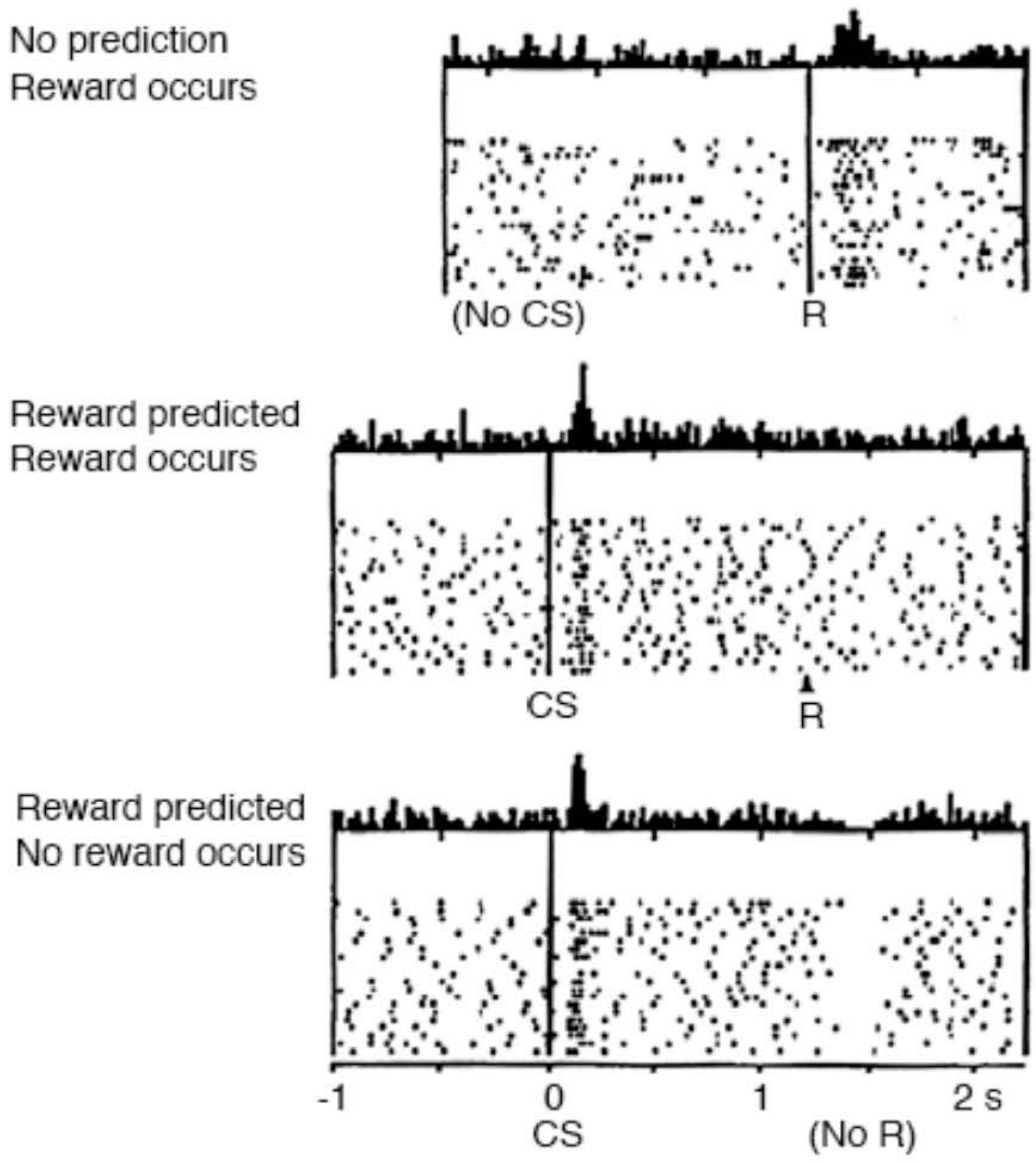


By trying different actions from different starting points, gradually learn the expected reward value from any starting point

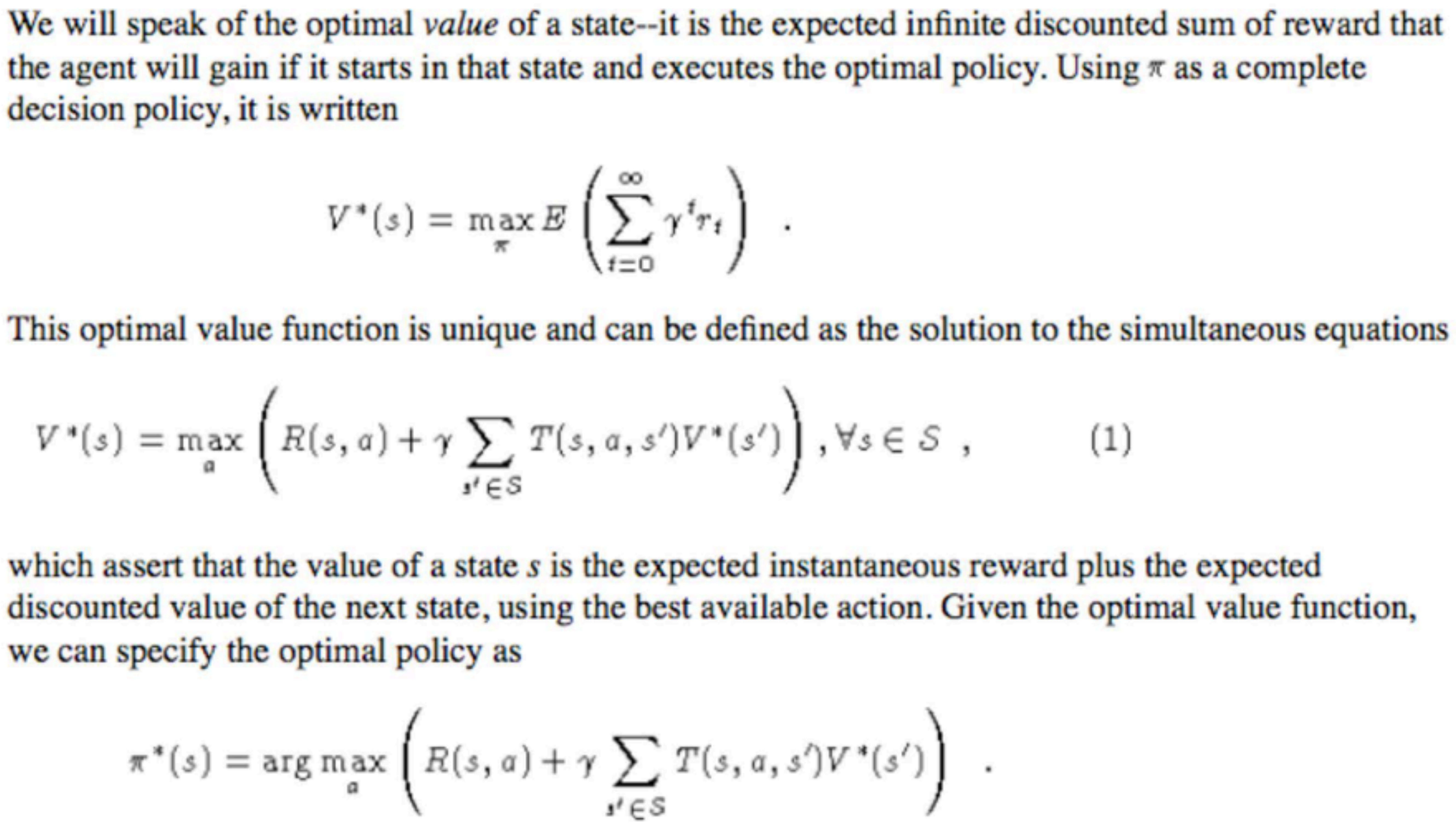
A Monkey uses Secondary Reward



Do dopamine neurons report an erro□ in the prediction of reward?



The basic Reinforcement Learning model



We will speak of the optimal *value* of a state--it is the expected infinite discounted sum of reward that the agent will gain if it starts in that state and executes the optimal policy. Using π as a complete decision policy, it is written

$$V^*(s) = \max_{\pi} E\left(\sum_{t=0}^{\infty} \gamma^t r_t\right) .$$

This optimal value function is unique and can be defined as the solution to the simultaneous equations

$$V^*(s) = \max_{a}\left(R(s,a) + \gamma \sum_{s' \in S} T(s,a,s')V^*(s')\right), \forall s \in S \;, \qquad (1)$$

which assert that the value of a state *s* is the expected instantaneous reward plus the expected discounted value of the next state, using the best available action. Given the optimal value function, we can specify the optimal policy as

$$\pi^*(s) = \arg\max_{a}\left(R(s,a) + \gamma \sum_{s' \in S} T(s,a,s')V^*(s')\right) .$$

Value Iteration

```
initialize V(s) arbitrarily
loop until policy good enough
        loop for $s \in \mathcal{S}$
                loop for $a \in \mathcal{A}$
                        $Q(s,a) := R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V(s')$
                $V(s) := \max_a Q(s,a)$
        end loop
end loop
```

Policy Iteration

```
choose an arbitrary policy $\pi'$
loop
        $\pi := \pi'$
        compute the value function of policy $\pi$ :
                solve the linear equations
                        $V_\pi(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V_\pi(s')$
        improve the policy at each state:
                $\pi'(s) := \arg\max_a \left( R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V_\pi(s') \right)$
until $\pi = \pi'$
```

Temporal Difference Learning

$$\bar{V}_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$

where $0 \leq \gamma < 1$. This formula can be expanded

$$\bar{V}_t = r_t + \sum_{i=1}^{\infty} \gamma^i r_{t+i}$$

by changing the index of i to start from 0.

$$\bar{V}_t = r_t + \sum_{i=0}^{\infty} \gamma^{i+1} r_{t+i+1}$$
$$\bar{V}_t = r_t + \gamma \sum_{i=0}^{\infty} \gamma^i r_{t+i+1}$$
$$\bar{V}_t = r_t + \gamma \bar{V}_{t+1}$$

Thus, the reinforcement is the difference between the ideal prediction and the current prediction.

$$r_t = \bar{V}_t - \gamma \bar{V}_{t+1}$$

Q - Learning

Temporal difference learning [Sutton and Barto, 1998], uses the error between the current estimated values of states and the observed reward to drive learning. In a related Q-learning form, the estimate of the quality value of a state-action pair is adjusted by this error δ_Q using a learning rate α:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \delta_Q \tag{3}$$

Two fundamental learning rules for δ_Q are 1) the original Q-learning rule [Watkins, 1989] and 2) SARSA [Rummery and Niranjan, 1994]. While Q-learning rule is an off-policy rule, i.e. it uses errors between current observations and estimates of the values for following an optimal policy, while actually following a potentially suboptimal policy during learning, SARSA[1] is an on-policy learning rule, i.e. the updates of the state and action values reflect the current policy derived from these value estimates. While in the general case of Q-learning, the temporal difference is:

$$\delta_Q = r_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \tag{4}$$

for the more specific case of SARSA it is:

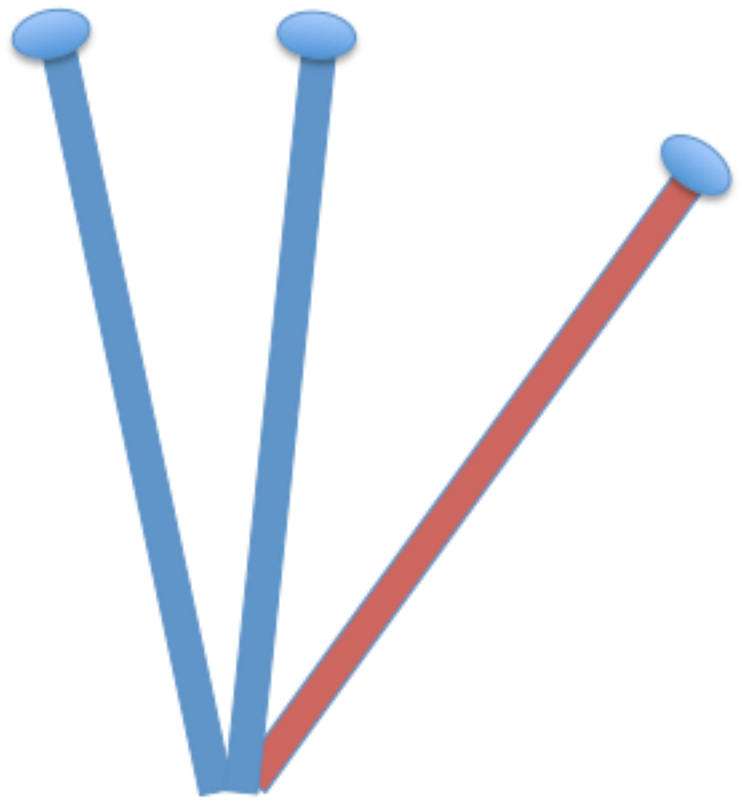
$$\delta_Q = r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t). \tag{5}$$

Which action to try?

This is known as the *multi-arm bandit problem* after Las Vegas slot machines

pulling an arm results in a reward for that arm that is random.

Which arm has the best average reward?

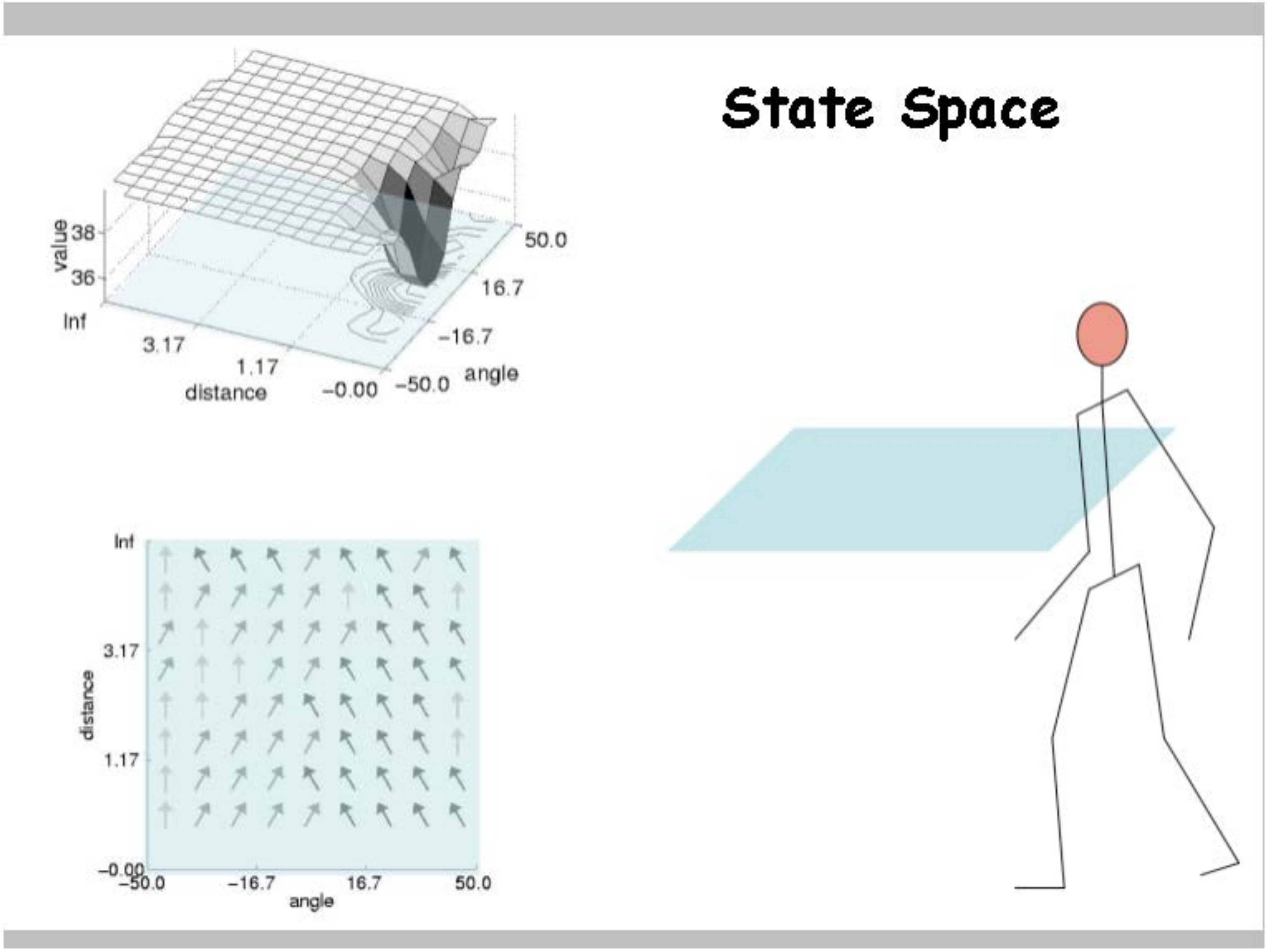


Epsilon-greedy algorithm

Pull the best arm with probability 1- ϵ
Pull the other arms, including the best, with probability $\frac{\epsilon}{3}$

A value for ϵ might be 10%

Avoiding obstacles while walking


State Space
value
Inf
distance
angle
Inf
distance
angle