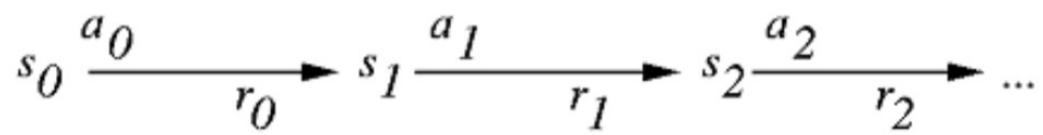
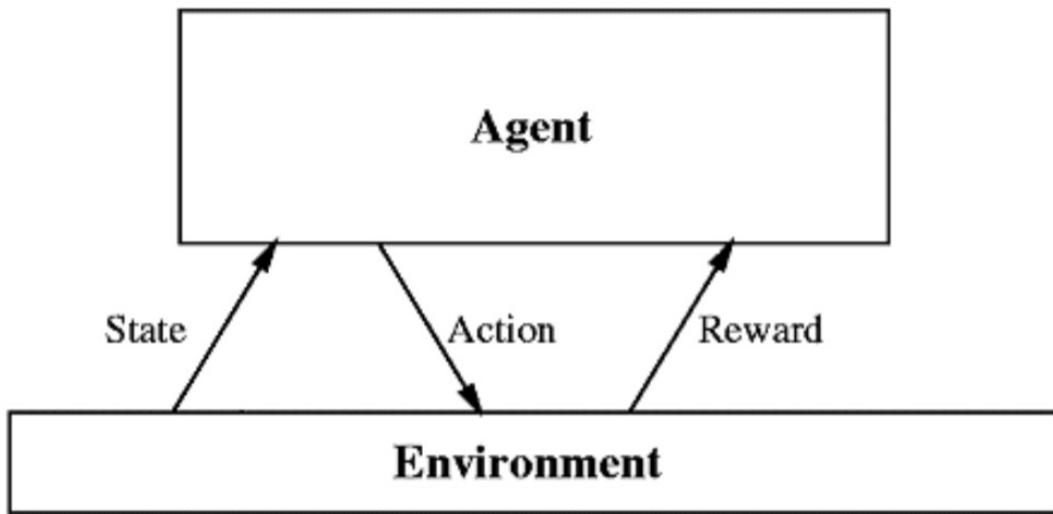
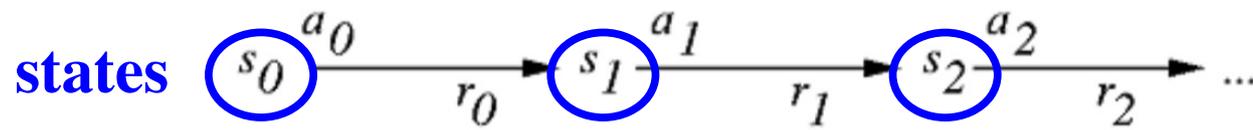
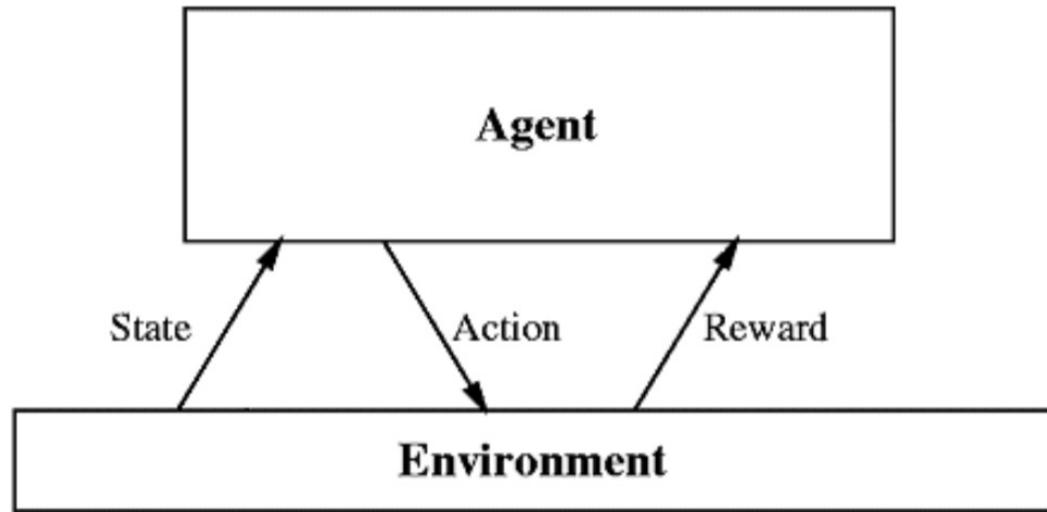
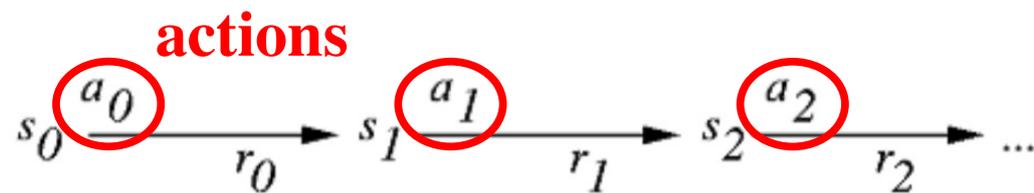
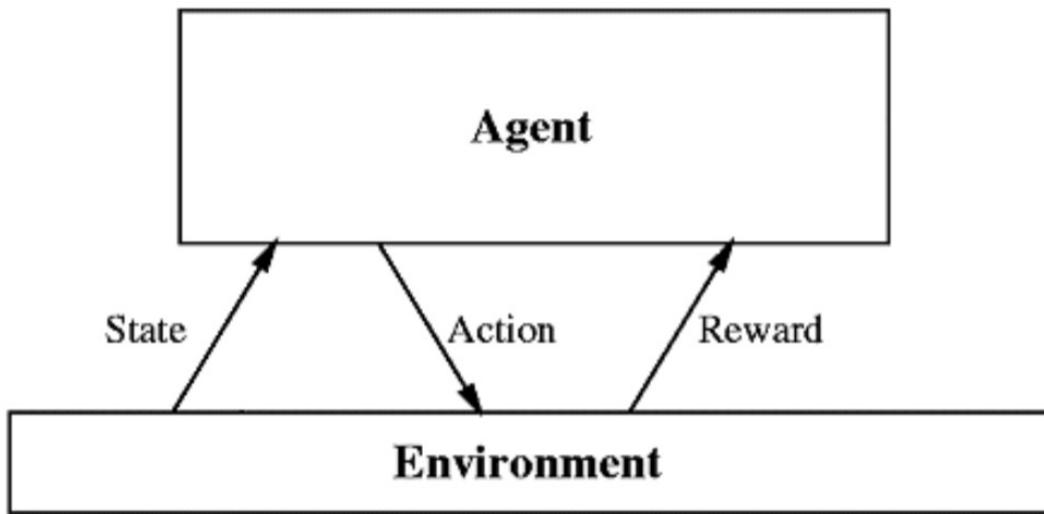
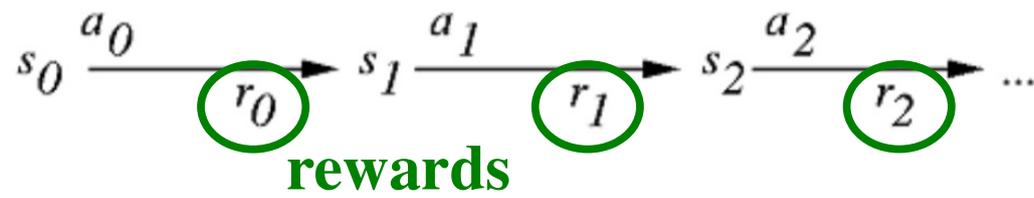
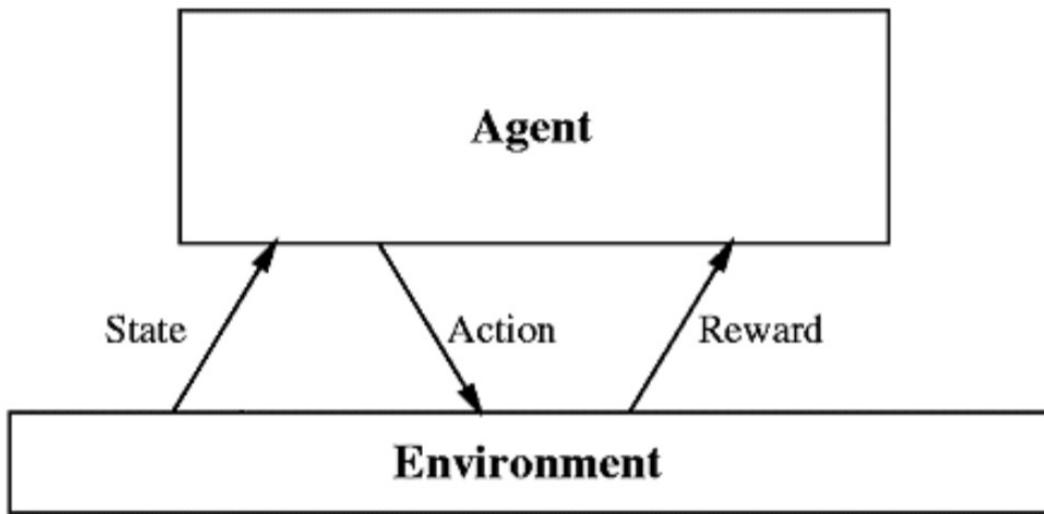


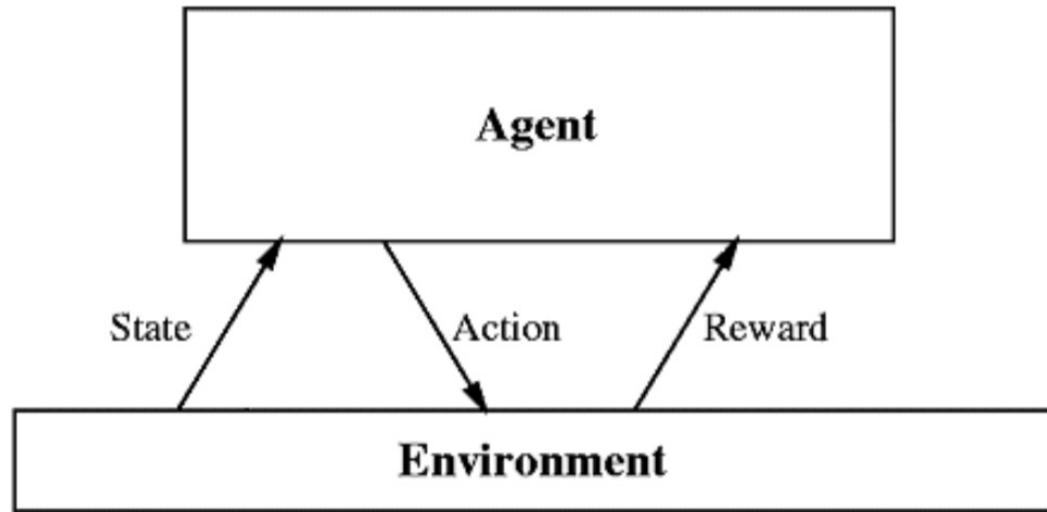
Q Learning



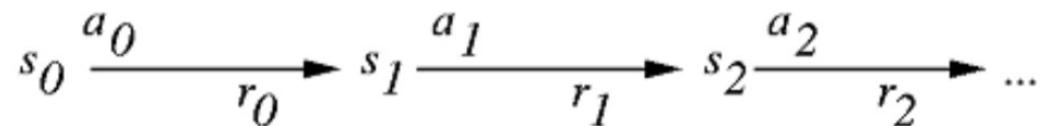








A **policy** tells the agent what actions to choose in each state



cumulative reward over time (starting from a particular state):

$$r_0 + r_1 + r_2 + r_3 + r_4 + \dots$$

The **optimal policy** is the one that maximizes the cumulative reward for all states

The **value** $V^*(s)$ of a state s is how much cumulative reward the agent can achieve by starting in that state and following the optimal policy

Generally speaking, agents value **immediate** rewards over **delayed** rewards

The more delayed a reward is in the future, the less value it has to the agent

The value function $V^*(s)$ reflects this idea by multiplying each future reward by a **discount factor** between 0 and 1 for every time step it is delayed

Example: discount factor = 0.8

discounted cumulative reward starting in state s_0 is

$$r_0 + \mathbf{0.8} \times r_1 + \mathbf{0.8} \times \mathbf{0.8} \times r_2 + \mathbf{0.8} \times \mathbf{0.8} \times \mathbf{0.8} \times r_3 + \dots$$

In general, with a discount factor of γ , the value of state s is

$$V^*(s) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \dots$$

Value of **state 0** (Rosie standing in location 0 but ball is gone)

R _ _ _ _ _ _ _ _ _ _

Maximum cumulative reward possible is 0

$$V^*(0) = 0$$

Value of **state 1** (Rosie standing in location 0 next to the ball)

oR _ _ _ _ _ _ _ _ _ _

Action chosen by the optimal policy: *kick*

Reward: 10

New state: 0 (Rosie still in location 0 but ball is now gone)

$$\begin{aligned} V^*(1) &= 10 + 0.8 \times V^*(0) \\ &= 10 \end{aligned}$$

Value of **state 2** (Rosie standing 1 step away from the ball)

○ R _ _ _ _ _ _ _ _ _ _

Action chosen by the optimal policy: *forward*

Reward: 0

New state: 1 (Rosie standing in location 0 next to the ball)

$$\begin{aligned} V^*(2) &= 0 + 0.8 \times V^*(1) \\ &= 0.8 \times 10 \\ &= 8 \end{aligned}$$

Value of **state 3** (Rosie standing 2 steps away from the ball)

○ _ R _ _ _ _ _ _ _ _

Action chosen by the optimal policy: *forward*

Reward: 0

New state: 2 (Rosie standing 1 step away from the ball)

$$\begin{aligned} V^*(3) &= 0 + 0.8 \times V^*(2) \\ &= 0 + 0.8 \times 8 \\ &= 6.4 < V^*(2) \text{ or } V^*(1) \text{ because payoff is further in the future} \end{aligned}$$

The environment is described by two functions:

reward function r

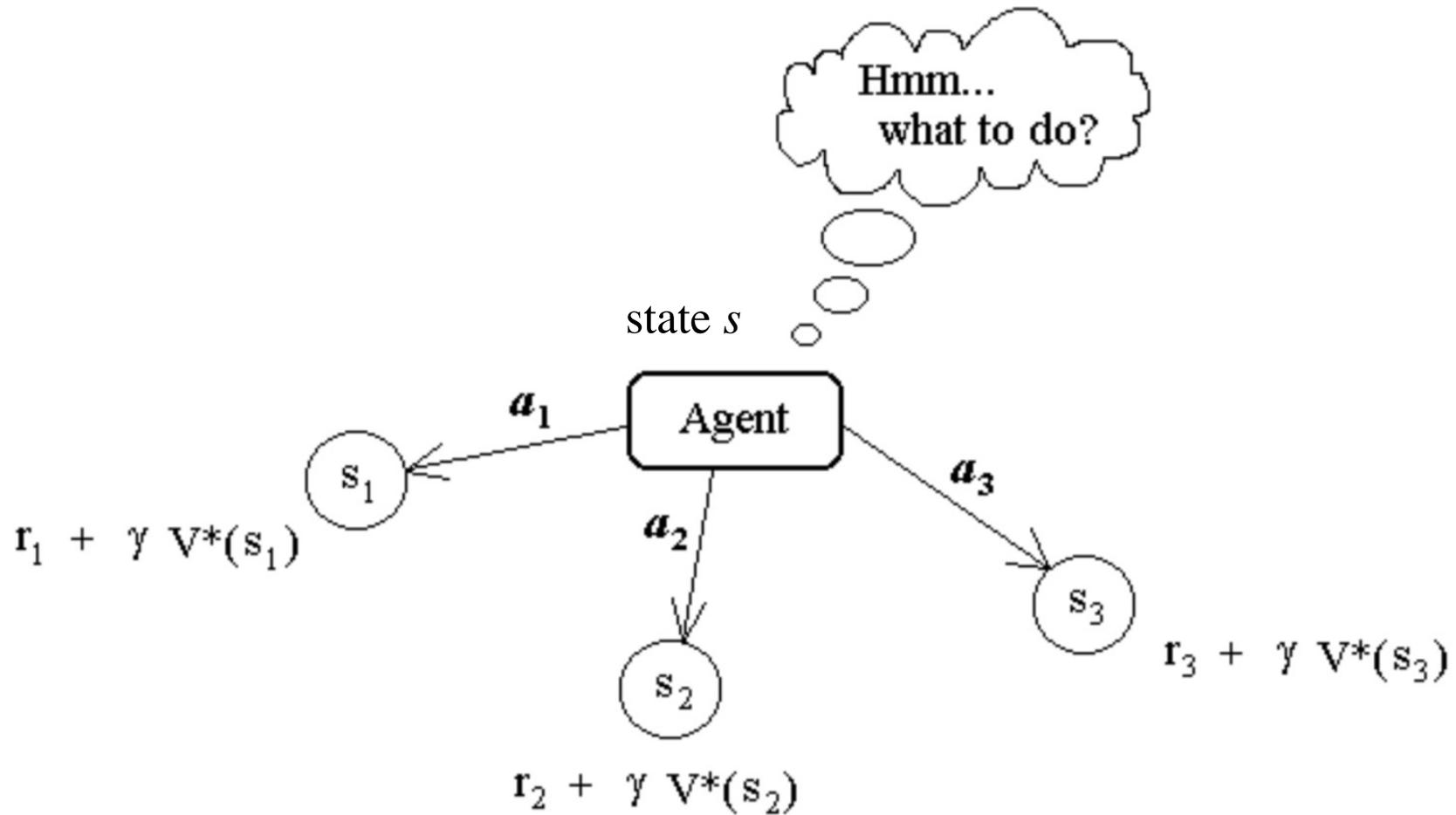
$r_t = r(s_t, a_t)$ is the reward received for performing action a_t in state s_t

state transition function δ

$s_{t+1} = \delta(s_t, a_t)$ is the new state that results from performing action a_t in state s_t

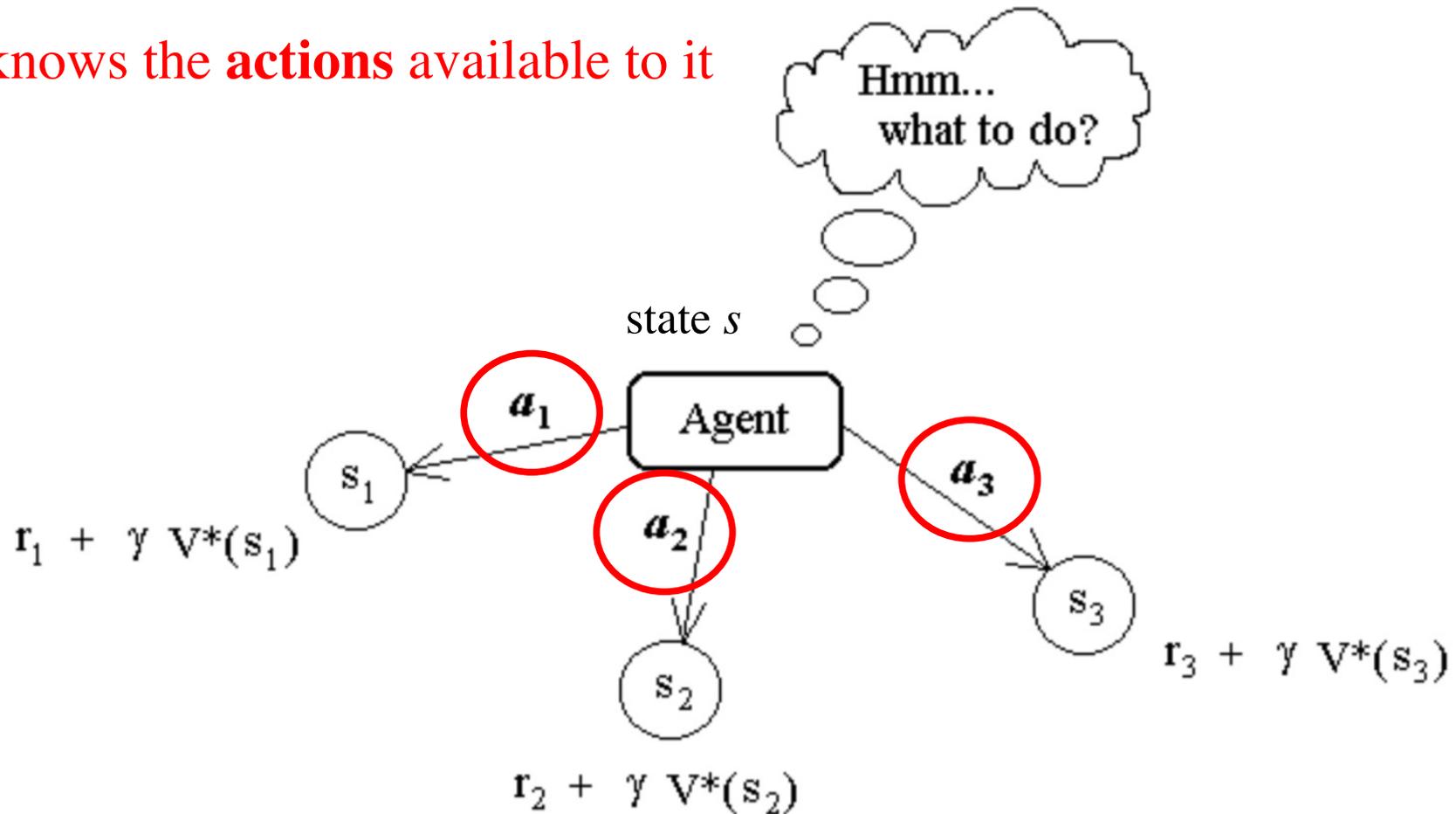
r and δ may be **nondeterministic** or **unknown** to the agent, meaning that the agent may be unable to predict the results of its actions

If an agent had **perfect knowledge** of the value function V^* , the reward function r , and the state transition function δ , it could always choose the best action from any state s :



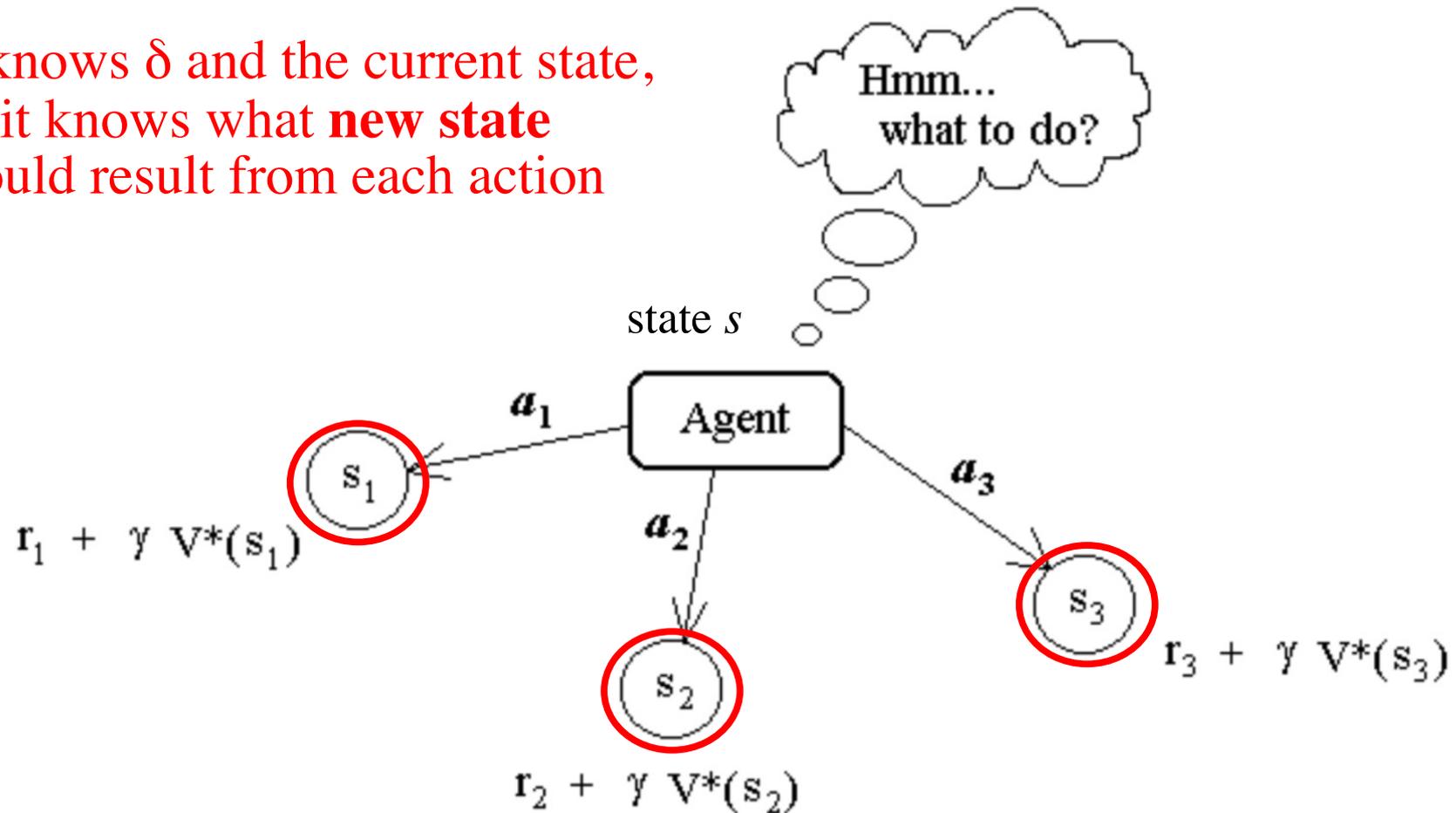
If an agent had **perfect knowledge** of the value function V^* , the reward function r , and the state transition function δ , it could always choose the best action from any state s :

It knows the **actions** available to it



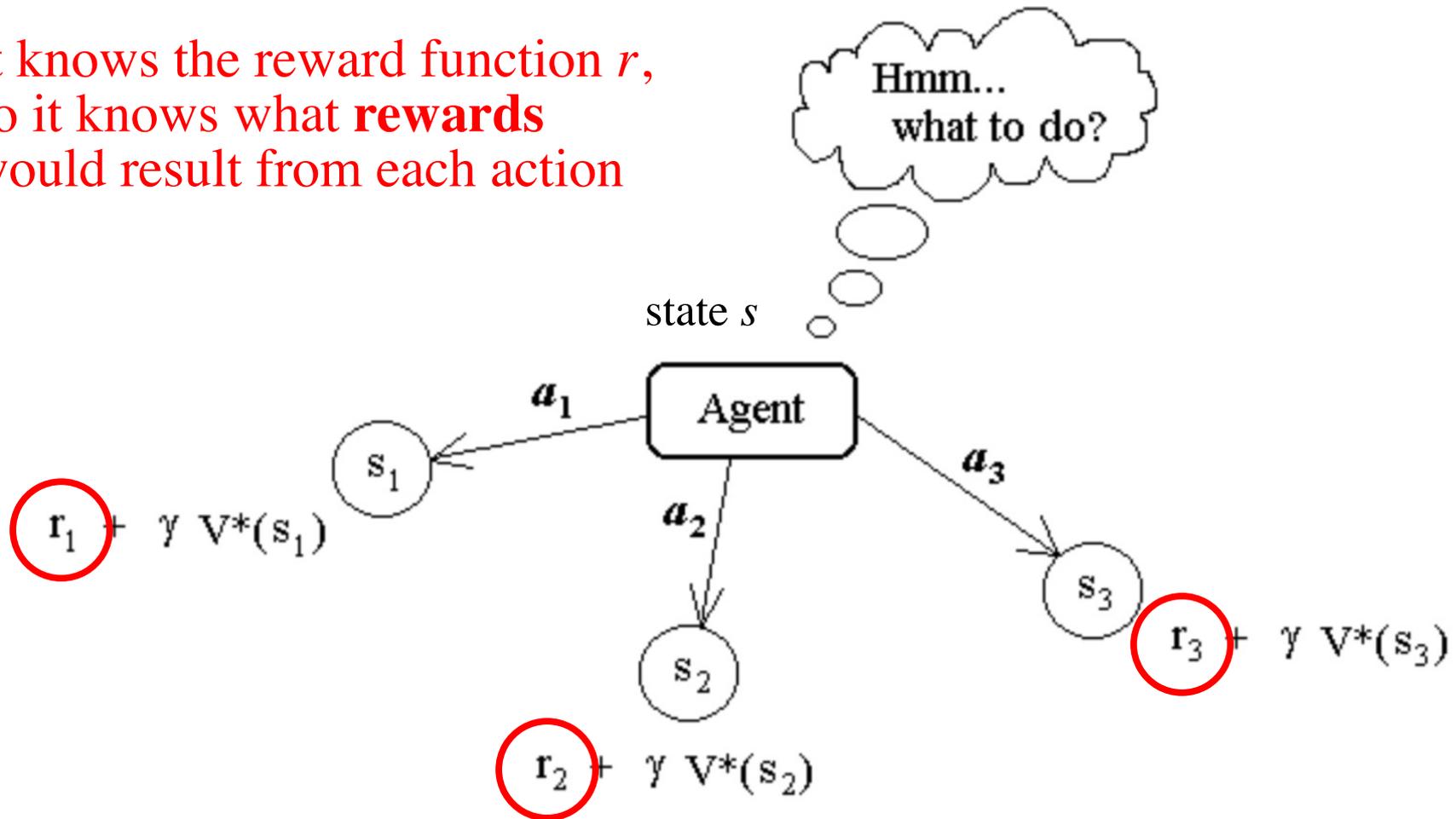
If an agent had **perfect knowledge** of the value function V^* , the reward function r , and the state transition function δ , it could always choose the best action from any state s :

It knows δ and the current state, so it knows what **new state** would result from each action



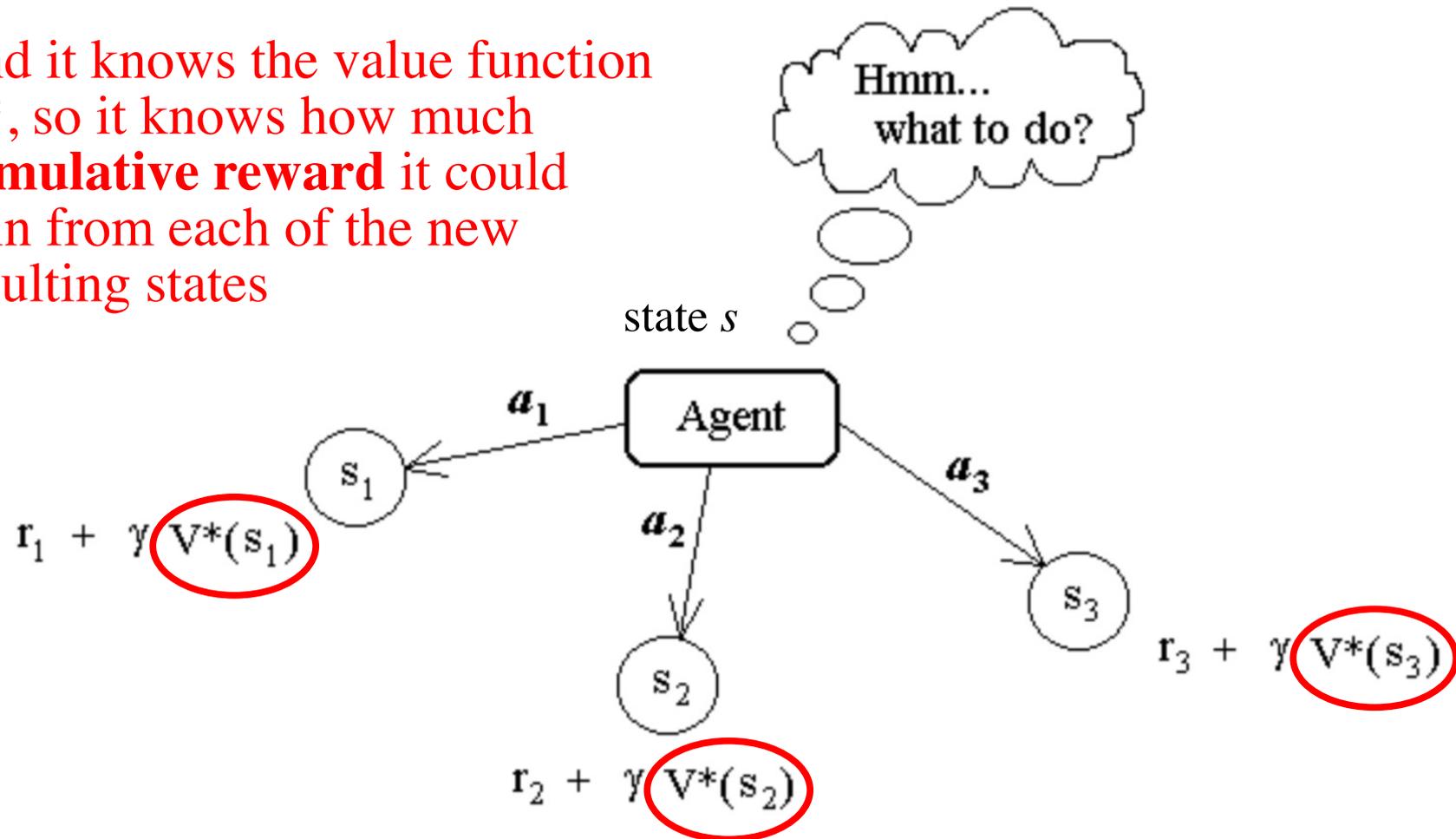
If an agent had **perfect knowledge** of the value function V^* , the reward function r , and the state transition function δ , it could always choose the best action from any state s :

It knows the reward function r , so it knows what **rewards** would result from each action



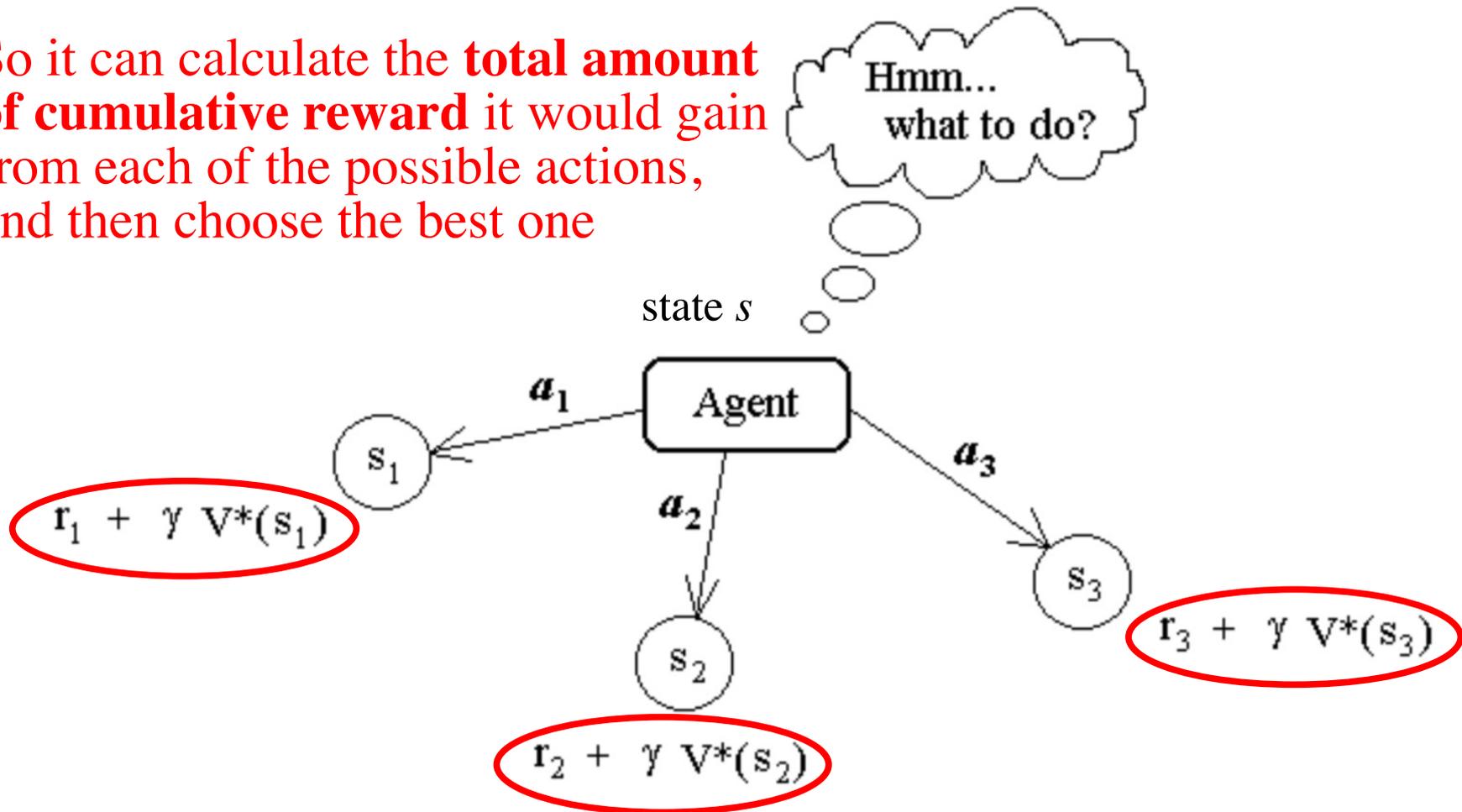
If an agent had **perfect knowledge** of the value function V^* , the reward function r , and the state transition function δ , it could always choose the best action from any state s :

And it knows the value function V^* , so it knows how much **cumulative reward** it could gain from each of the new resulting states



If an agent had **perfect knowledge** of the value function V^* , the reward function r , and the state transition function δ , it could always choose the best action from any state s :

So it can calculate the **total amount of cumulative reward** it would gain from each of the possible actions, and then choose the best one

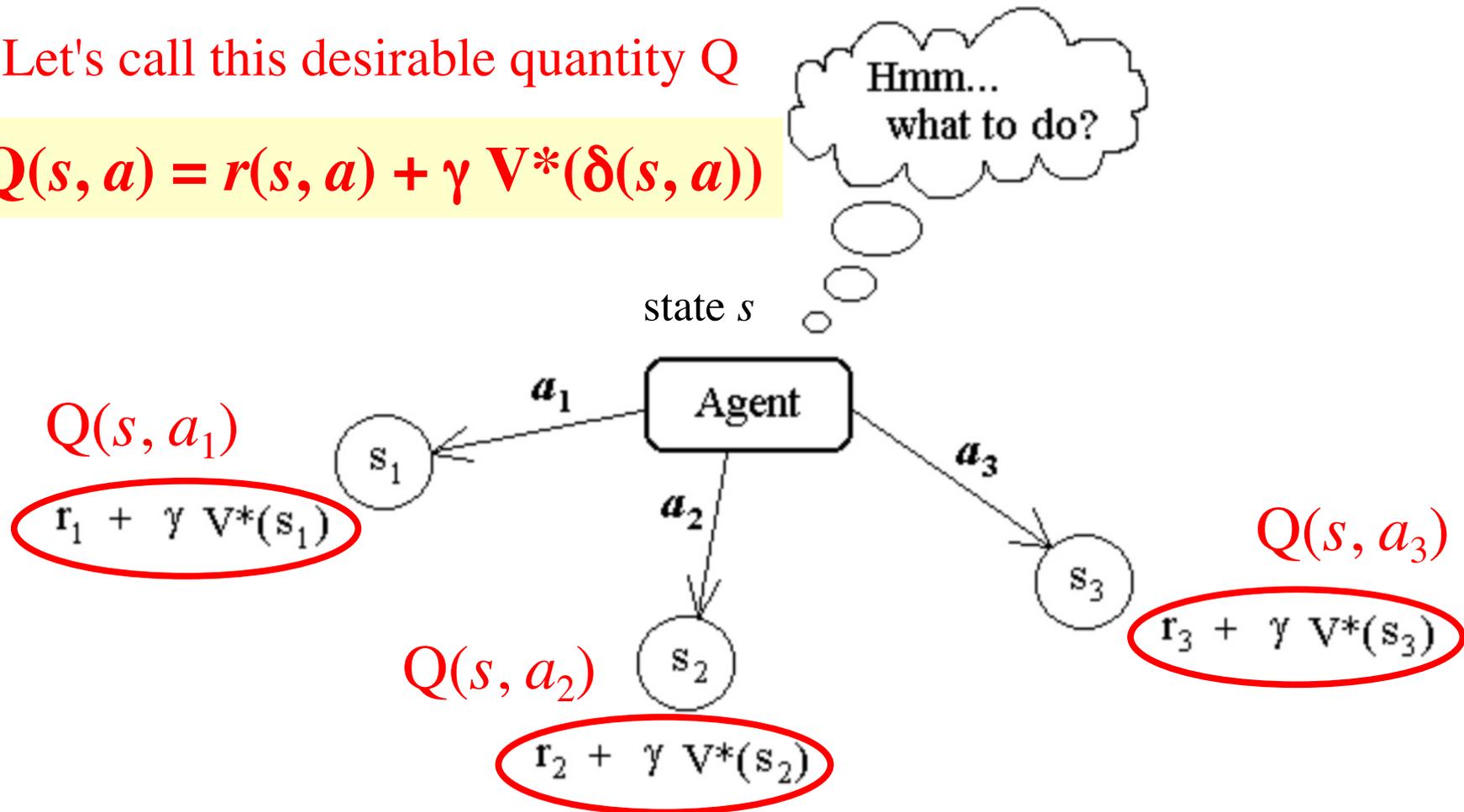


But the agent doesn't actually know any of these functions! 😞
It only knows the **current state** s and the available **actions** a_1, a_2, a_3

If an agent had **perfect knowledge** of the value function V^* , the reward function r , and the state transition function δ , it could always choose the best action from any state s :

Let's call this desirable quantity Q

$$Q(s, a) = r(s, a) + \gamma V^*(\delta(s, a))$$

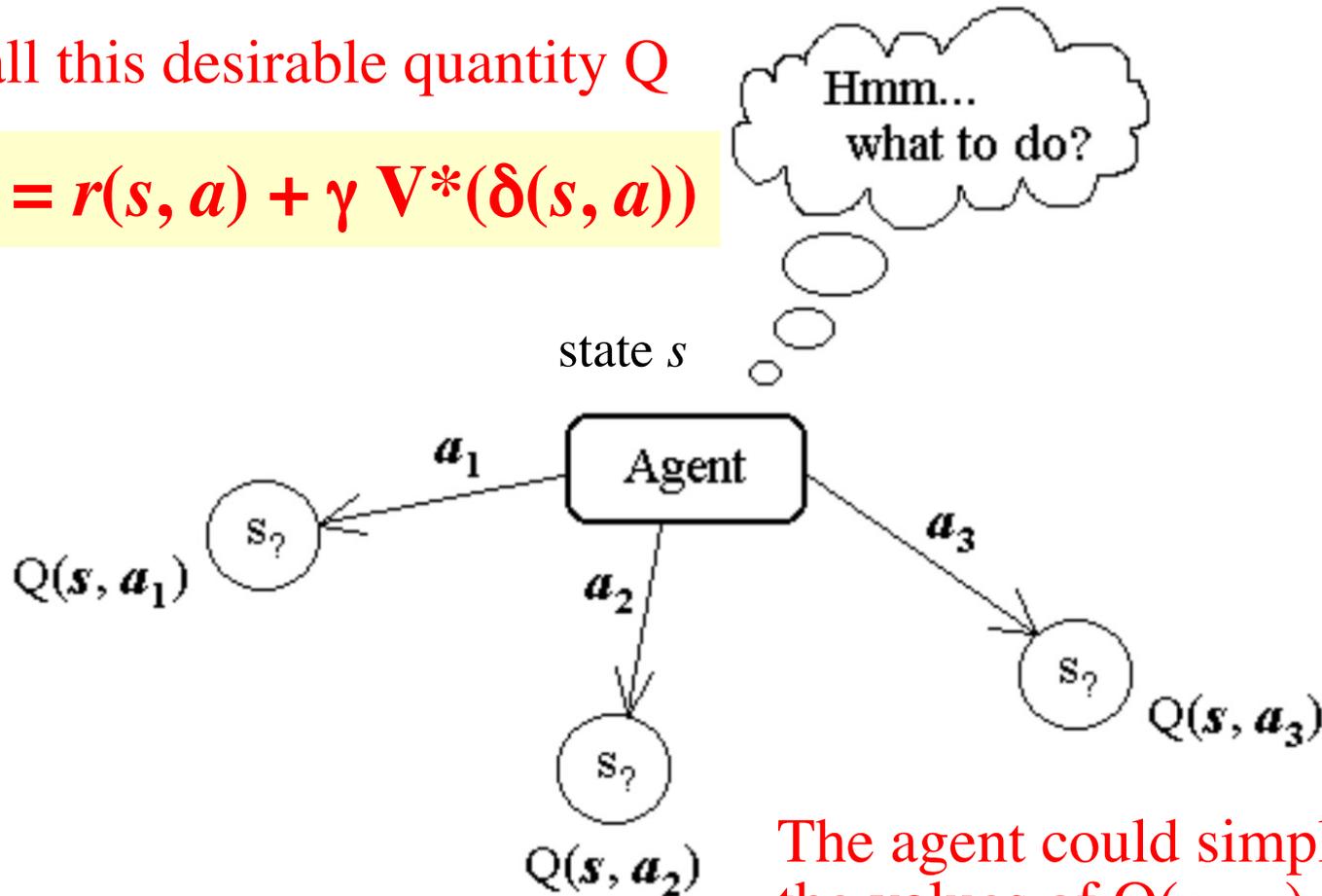


But the agent doesn't actually know any of these functions! 😞
It only knows the **current state** s and the available **actions** a_1, a_2, a_3

What if the agent **knew the Q function** without knowing anything about the functions V^* , r , and δ that are “hidden inside” of it?

Let's call this desirable quantity Q

$$Q(s, a) = r(s, a) + \gamma V^*(\delta(s, a))$$

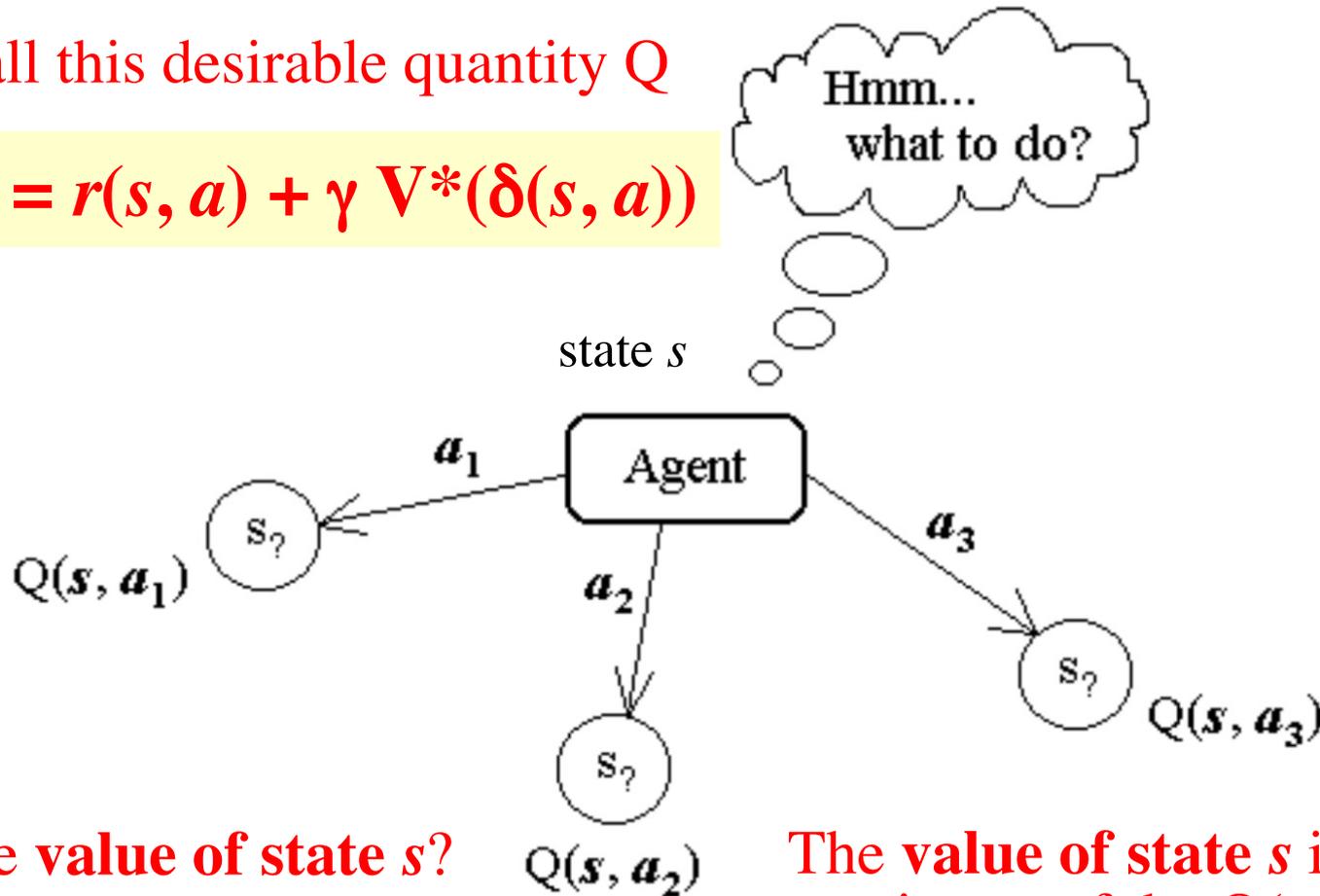


The agent could simply compare the values of $Q(s, a_1)$, $Q(s, a_2)$, and $Q(s, a_3)$ **directly**, and choose the best action

What if the agent **knew the Q function** without knowing anything about the functions V^* , r , and δ that are “hidden inside” of it?

Let's call this desirable quantity Q

$$Q(s, a) = r(s, a) + \gamma V^*(\delta(s, a))$$



What is the **value of state s** ?

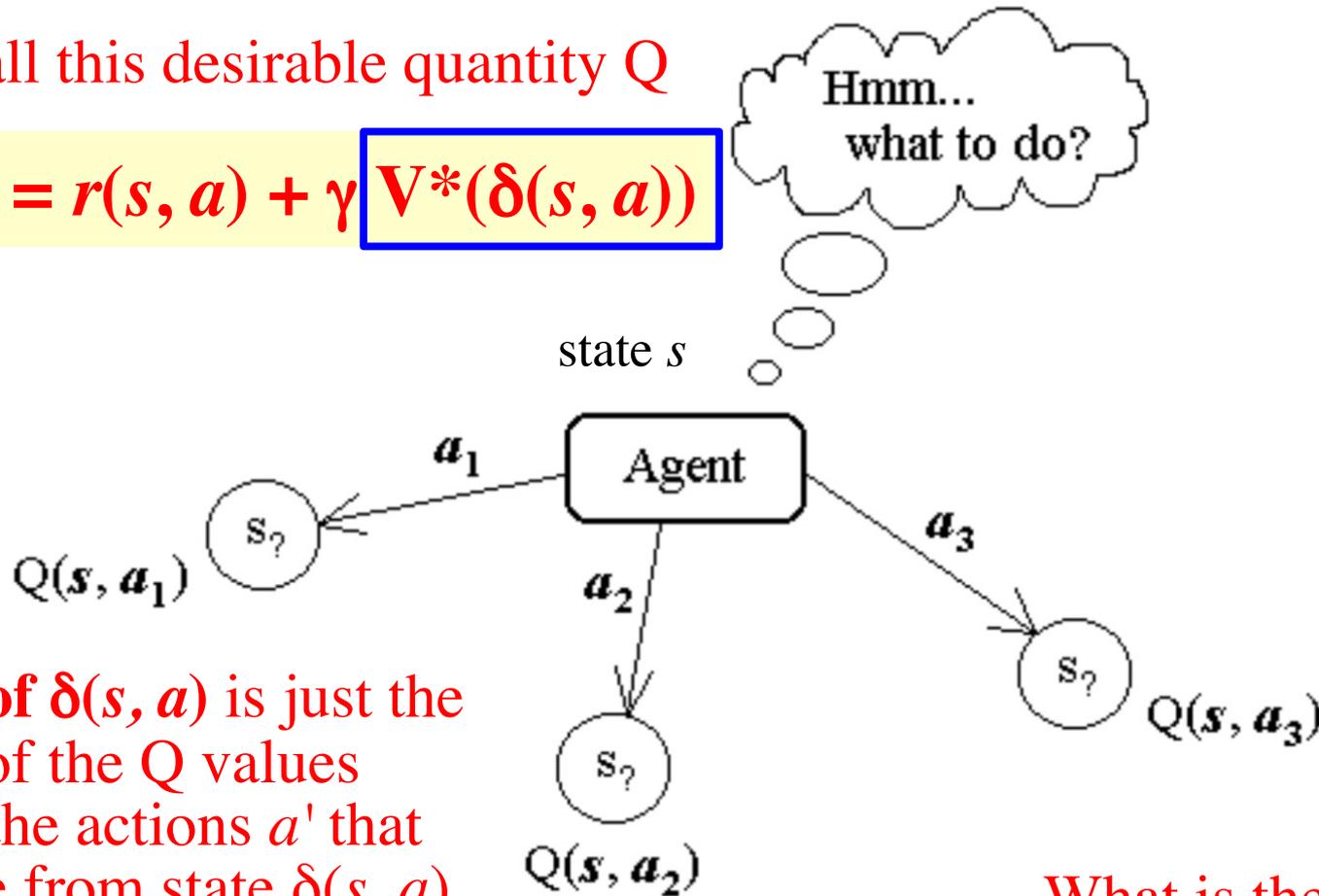
$$V^*(s) = \max_{a'} \{Q(s, a')\}$$

The **value of state s** is just the maximum of the $Q(s, a')$ values over all of the actions a' that are possible from state s

What if the agent **knew the Q function** without knowing anything about the functions V^* , r , and δ that are “hidden inside” of it?

Let's call this desirable quantity Q

$$Q(s, a) = r(s, a) + \gamma V^*(\delta(s, a))$$



The value of $\delta(s, a)$ is just the maximum of the Q values over all of the actions a' that are possible from state $\delta(s, a)$

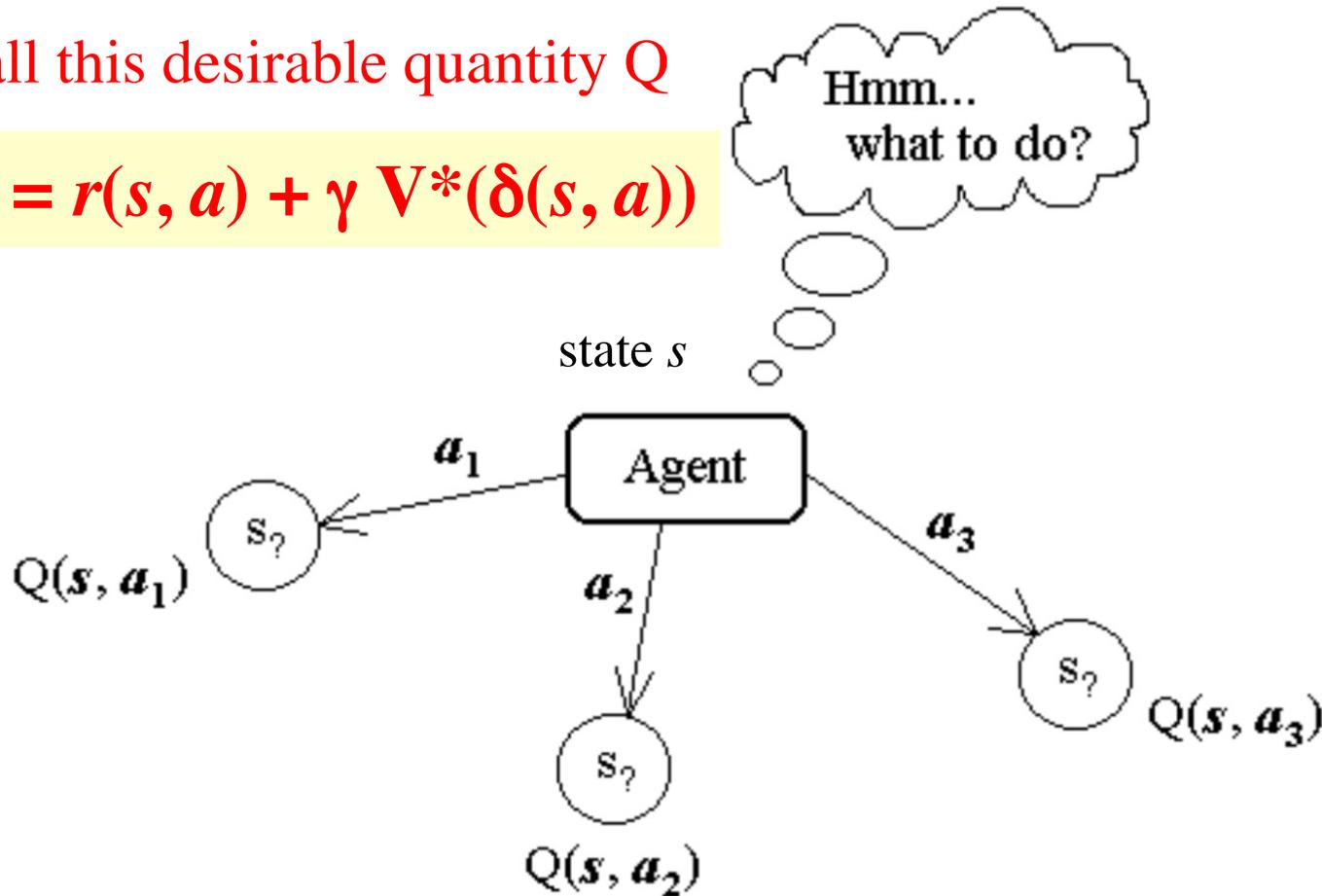
$$V^*(\delta(s, a)) = \max_{a'} \{Q(\delta(s, a), a')\}$$

What is the value of the new state $\delta(s, a)$?

What if the agent **knew the Q function** without knowing anything about the functions V^* , r , and δ that are “hidden inside” of it?

Let's call this desirable quantity Q

$$Q(s, a) = r(s, a) + \gamma V^*(\delta(s, a))$$



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

Q Learning Algorithm

- Let Q denote the true Q function (*i.e.*, the target function).
- Let Q' denote an approximation to Q
- We can represent Q' by a table:

		Possible actions								
		a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	
Possible states	s_1									
	s_2									
	s_3									
	s_4									
	s_5									
	s_6									
	s_7									

$Q'(s_3, a_6)$

Example:

Perform action a_6 in state s_3

Result: reward = 2, new state = s_5

Update: $Q'(s_3, a_6) \leftarrow 2 + \gamma \max_{a'} \{Q'(s_5, a')\}$

Update rule for action a in state s

$$Q'(s, a) \leftarrow \underset{\substack{\uparrow \\ \text{observed} \\ \text{reward}}}{r} + \gamma \max_{a'} \{ \underset{\substack{\uparrow \\ \text{observed} \\ \text{new state}}}{Q'(s_{\text{new}}, a')} \}$$

Algorithm:

1. Initialize all Q' table entries to 0 (or random values)
2. Observe the current state s
3. Repeat:
 - a. Choose an action a and execute it
 - b. Receive immediate reward r
 - c. Observe the resulting state s_{new}
 - d. Update Q' entry for s and a : $Q'(s, a) \leftarrow r + \gamma \max_{a'} \{Q'(s_{new}, a')\}$
 - e. Update current state: $s \leftarrow s_{new}$

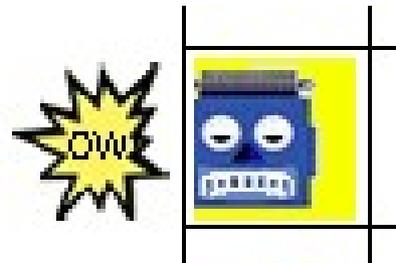
Example: Robby the Robot

Actions

- Move North
- Move South
- Move East
- Move West
- Stay Put
- Pick Up Can
- Move at Random

Rewards / Punishments

- +10** Successfully picked up a can
- 1** Tried to pick up a can that wasn't there
- 5** Crashed into a wall



The Q Table

Possible Actions

Situation

North South East West StayPut PickUpCan RandomMove

#1 **EEEEEE**

#2 **EEEEEC**

#3 **EEEEEW**

#4 **EEECE**

#5 **EEEC**

#6 **EEECW**

#7 **EEWE**

#8 **EEWC**

...

#243 **WWWW**

	North	South	East	West	StayPut	PickUpCan	RandomMove
#1 EEEEEE							
#2 EEEEEC							
#3 EEEEEW							
#4 EEECE							
#5 EEEC							
#6 EEECW							
#7 EEWE							
#8 EEWC							
...							
#243 WWWW							

Step 1: determine the current situation

Possible Actions

Situation

North South East West StayPut PickUpCan RandomMove

#1 **EEEEEE**

#2 **EEEEEC**

#3 **EEEEEW**

#4 **EEECE**

#5 **EEECC**

#6 **EEECW**

#7 **EEWE**

#8 **EEWC**

...

#243 **WWWW**

	North	South	East	West	StayPut	PickUpCan	RandomMove
#1							
#2							
#3							
#4							
#5							
#6							
#7							
#8							
...							
#243							

Step 2: choose an action to perform

Possible Actions

Situation

North South East West StayPut PickUpCan RandomMove

#1 **EEEEEE**

#2 **EEEEEC**

#3 **EEEEEW**

#4 **EEECE**

#5 **EEECC**

#6 **EEECW**

#7 **EEEWE**

#8 **EEEWC**

...

#243 **WWWW**

	North	South	East	West	StayPut	PickUpCan	RandomMove
#1							
#2							
#3							
#4							
#5	2.9	-0.4	3.5	8.9	-6.7	10.2	4.4
#6							
#7							
#8							
...							
#243							

Step 2: choose an action to perform

Possible Actions

Situation

North South East West StayPut PickUpCan RandomMove

#1 **EEEEEE**

#2 **EEEEEC**

#3 **EEEEEW**

#4 **EEECE**

#5 **EEEC**

#6 **EEECW**

#7 **EEEWE**

#8 **EEEWC**

...

#243 **WWWWW**

	North	South	East	West	StayPut	PickUpCan	RandomMove
#1							
#2							
#3							
#4							
#5	2.9	-0.4	3.5	8.9	-6.7	10.2	4.4
#6							
#7							
#8							
...							
#243							

Step 3: observe reward and new situation

Reward: +10

Situation

North South East West StayPut PickUpCan RandomMove

#1 **EEEEEE**

#2 **EEEEEC**

#3 **EEEEEW**

#4 **EEECE**

#5 **EEECC**

#6 **EEECW**

#7 **EEEWE**

#8 **EEEWC**

...

#243 **WWWWW**

	North	South	East	West	StayPut	PickUpCan	RandomMove
#1							
#2							
#3							
#4							
#5	2.9	-0.4	3.5	8.9	-6.7	10.2	4.4
#6							
#7							
#8							
...							
#243							

Step 4: find max Q-value for new situation

Reward: +10

Situation

North South East West StayPut PickUpCan RandomMove

#1 **EEEEEE**

#2 **EEEEEC**

#3 **EEEEEW**

#4 **EEECE**

#5 **EEECC**

#6 **EEECW**

#7 **EEEWE**

#8 **EEEWC**

...

#243 **WWWWW**

	-7.0	25.5	-0.9	-2.3	11.3	3.6	-1.2
	2.9	-0.4	3.5	8.9	-6.7	10.2	4.4

... and repeat many, many times!

Possible Actions

Situation

North South East West StayPut PickUpCan RandomMove

#1 **EEEEEE**

#2 **EEEEEC**

#3 **EEEEEW**

#4 **EEECE**

#5 **EEECC**

#6 **EEECW**

#7 **EEWE**

#8 **EEWC**

...

#243 **WWWW**

	North	South	East	West	StayPut	PickUpCan	RandomMove
#1 EEEEEE							
#2 EEEEEC							
#3 EEEEEW							
#4 EEECE							
#5 EEECC							
#6 EEECW							
#7 EEWE							
#8 EEWC							
...							
#243 WWWW							

This Q update rule assumes that the reward function r and the state transition function δ are **deterministic**

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

What if the environment is **nondeterministic**?

This Q update rule assumes that the reward function r and the state transition function δ are **deterministic**

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

Instead of just replacing the current value of $Q(s, a)$ by the new value $r(s, a) + \gamma \max_{a'} \{Q(\delta(s, a), a')\}$, we will calculate a **weighted average** of the current Q value and the new value

$$\text{Weighted average of } X \text{ and } Y = (1 - \alpha) X + \alpha Y$$

where $0 \leq \alpha \leq 1$

$$\text{Examples: } \alpha = 0.3 \quad 0.7 X + 0.3 Y$$

$$\alpha = 0.5 \quad 0.5 X + 0.5 Y = (X + Y)/2$$

$$\alpha = 0 \quad X$$

$$\alpha = 1 \quad Y$$

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

$$Q(s, a) \leftarrow (1 - \alpha) \underbrace{Q(s, a)}_{\text{“X”}} + \alpha \underbrace{[r(s, a) + \gamma \max_{a'} \{Q(\delta(s, a), a')\}]}_{\text{“Y”}}$$

↑
“learning rate”

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

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

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

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

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

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

How to choose actions?

Naïve strategy: at each time step, choose the action a that **maximizes** the value of $Q(s, a)$ for the current state s (“greedy”)

This **exploits** the current Q-table knowledge, but doesn't **explore** the state-action space any further

This is dangerous, because the Q-table values could be way off (and they almost certainly will be at the start of training)

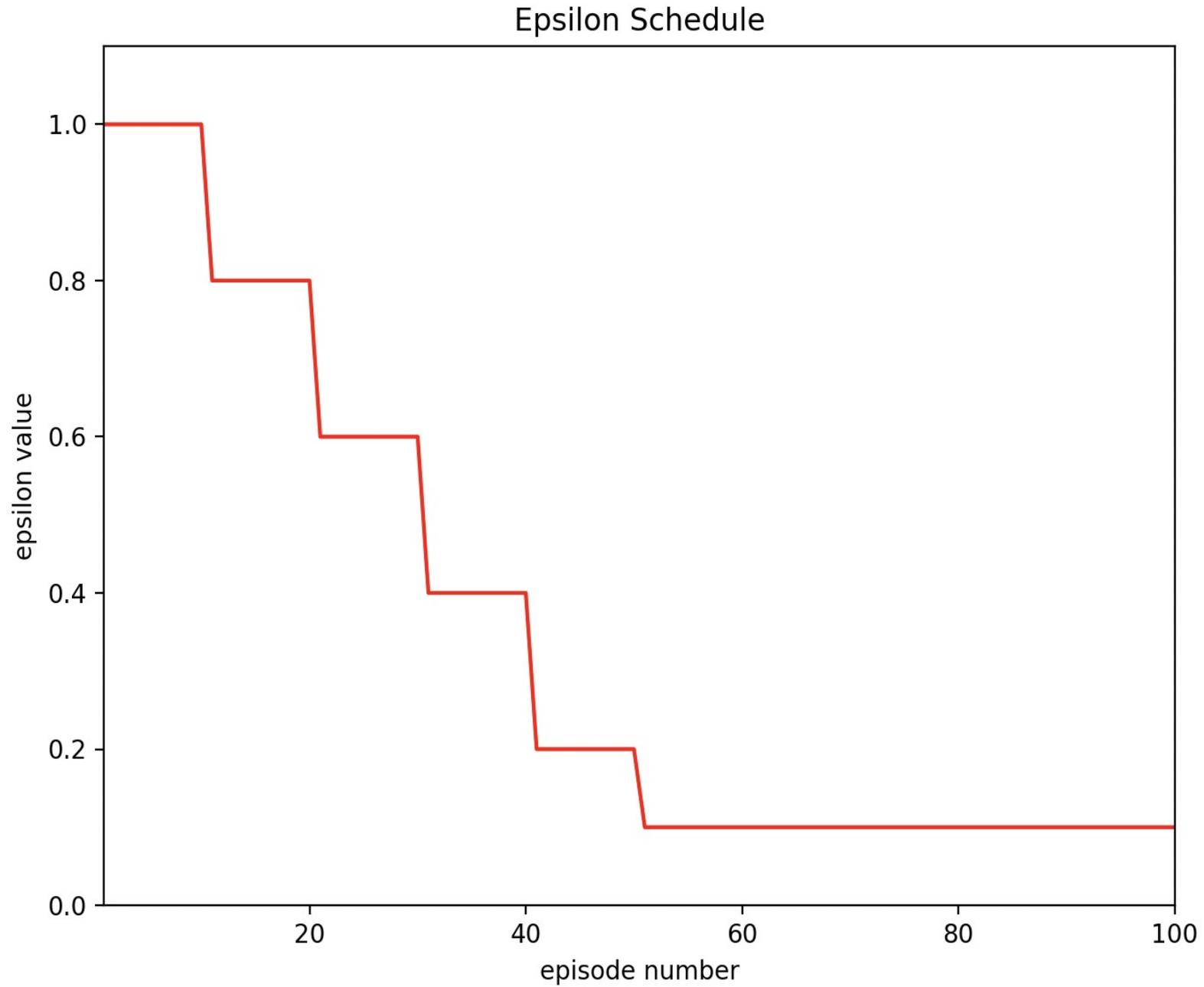
“Epsilon greedy” strategy:

With **probability ϵ** , choose a **random action**

With **probability $(1 - \epsilon)$** , choose the action that **maximizes $Q(s, a)$**

Even better: start with high ϵ and gradually decrease it over the run

Example “ ϵ schedules”



Example “ ϵ schedules”

