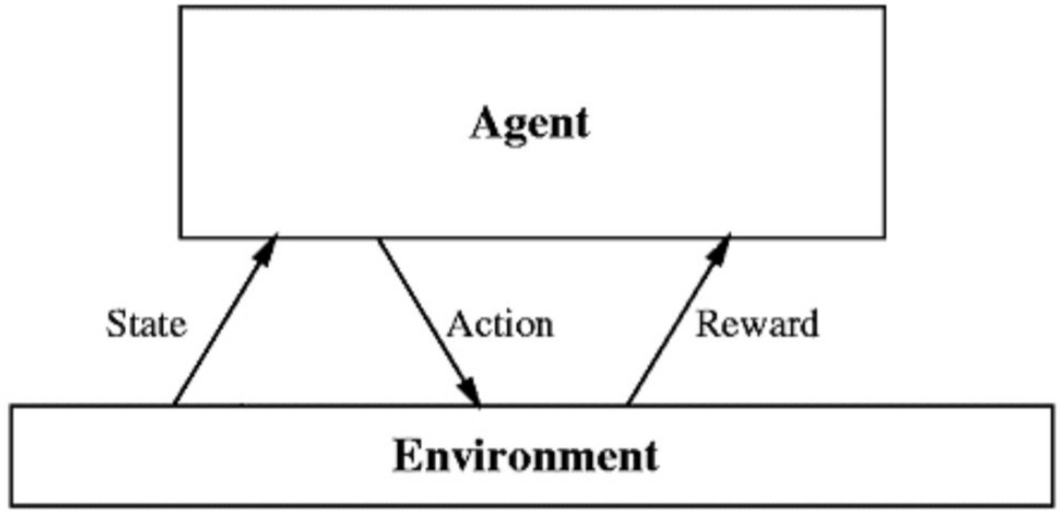
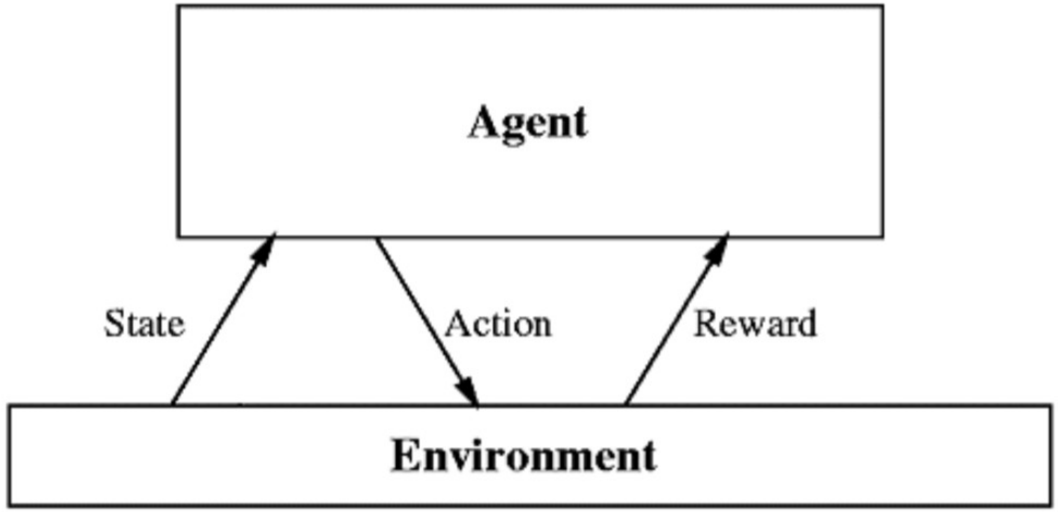
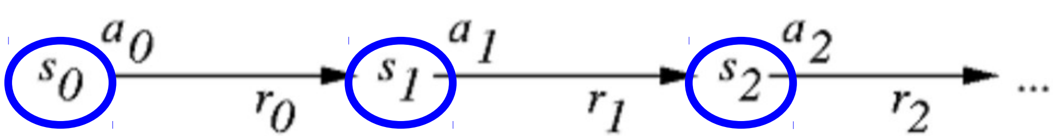
Q Learning

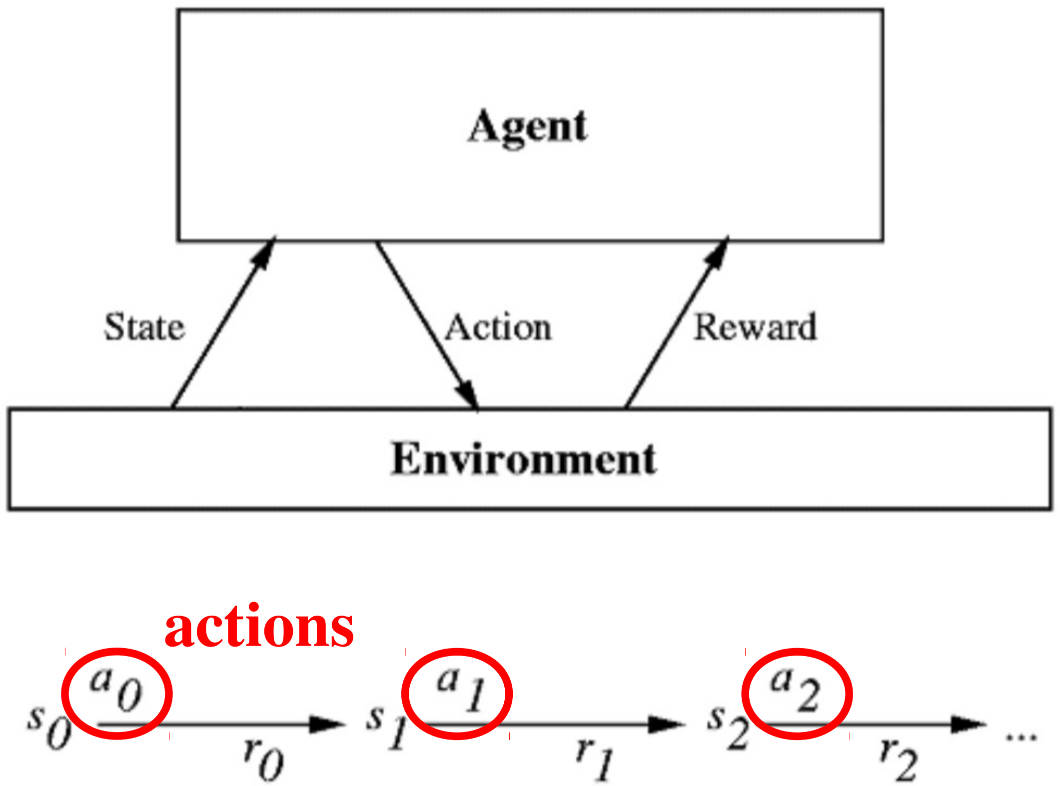


$$s_0 \xrightarrow[r_0]{a_0} s_1 \xrightarrow[r_1]{a_1} s_2 \xrightarrow[r_2]{a_2} \dots$$

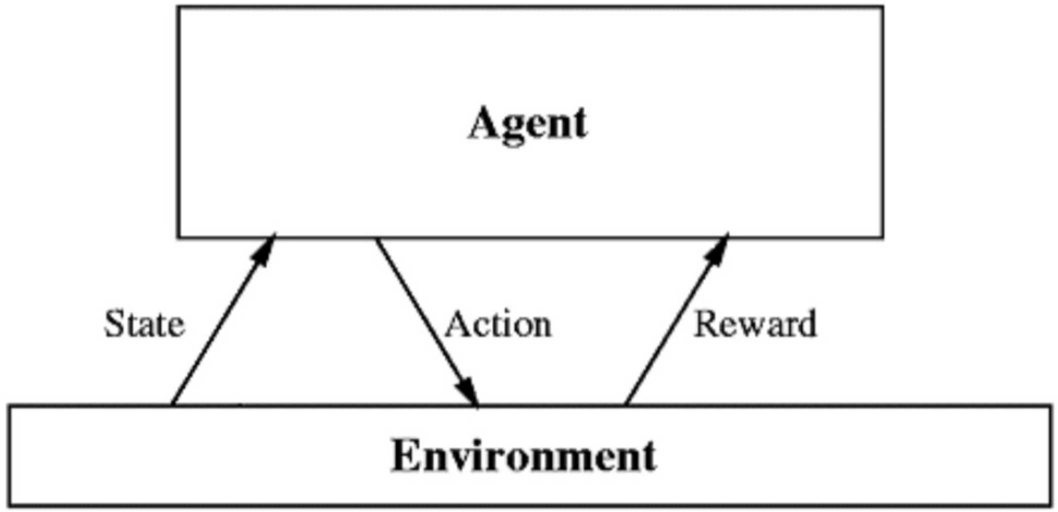

Agent
State
Action
Reward
Environment

states


s0
a0
r0
s1
a1
r1
s2
a2
r2
...

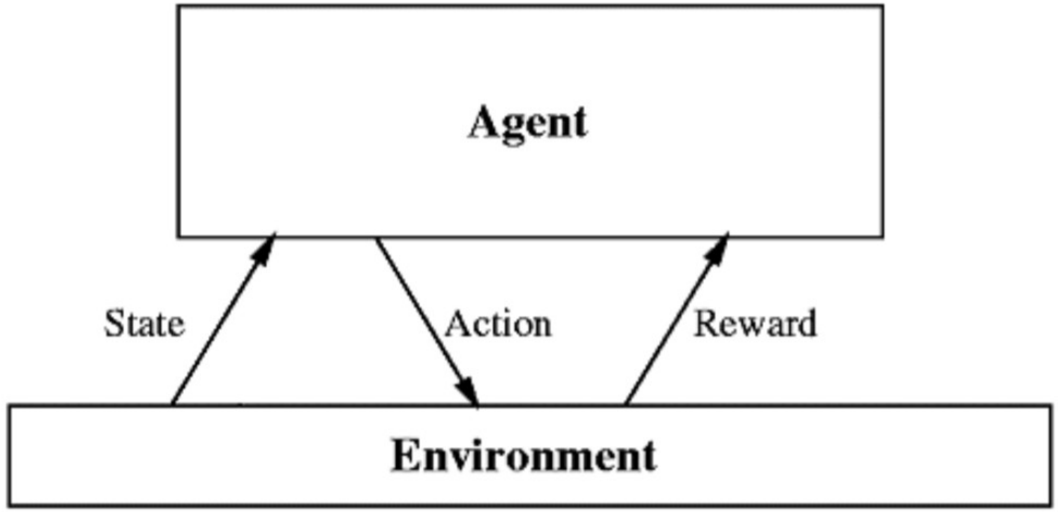


Agent
State
Action
Reward
Environment
actions
s_0
a_0
r_0
s_1
a_1
r_1
s_2
a_2
r_2
...



$$s_0 \xrightarrow[r_0]{a_0} s_1 \xrightarrow[r_1]{a_1} s_2 \xrightarrow[r_2]{a_2} \cdots$$

rewards



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

$$s_0 \xrightarrow[r_0]{a_0} s_1 \xrightarrow[r_1]{a_1} s_2 \xrightarrow[r_2]{a_2} \ldots$$

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

$$r0 + r1 + r2 + r3 + r4 + \ldots$$

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 *s0* is

$$r0 + \mathbf{0.8} \times r1 + \mathbf{0.8} \times \mathbf{0.8} \times r2 + \mathbf{0.8} \times \mathbf{0.8} \times \mathbf{0.8} \times r3 + \ldots$$

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

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

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)

```
o 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)

```
o _ R _ _ _ _ _ _ _ _ _ _
```

Action chosen by the optimal policy: *forward*
Reward: 0
New state: 2 (Rosie standing 1 step away from the ball)

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

The environment is described by two functions:

reward function $\boldsymbol{r}$

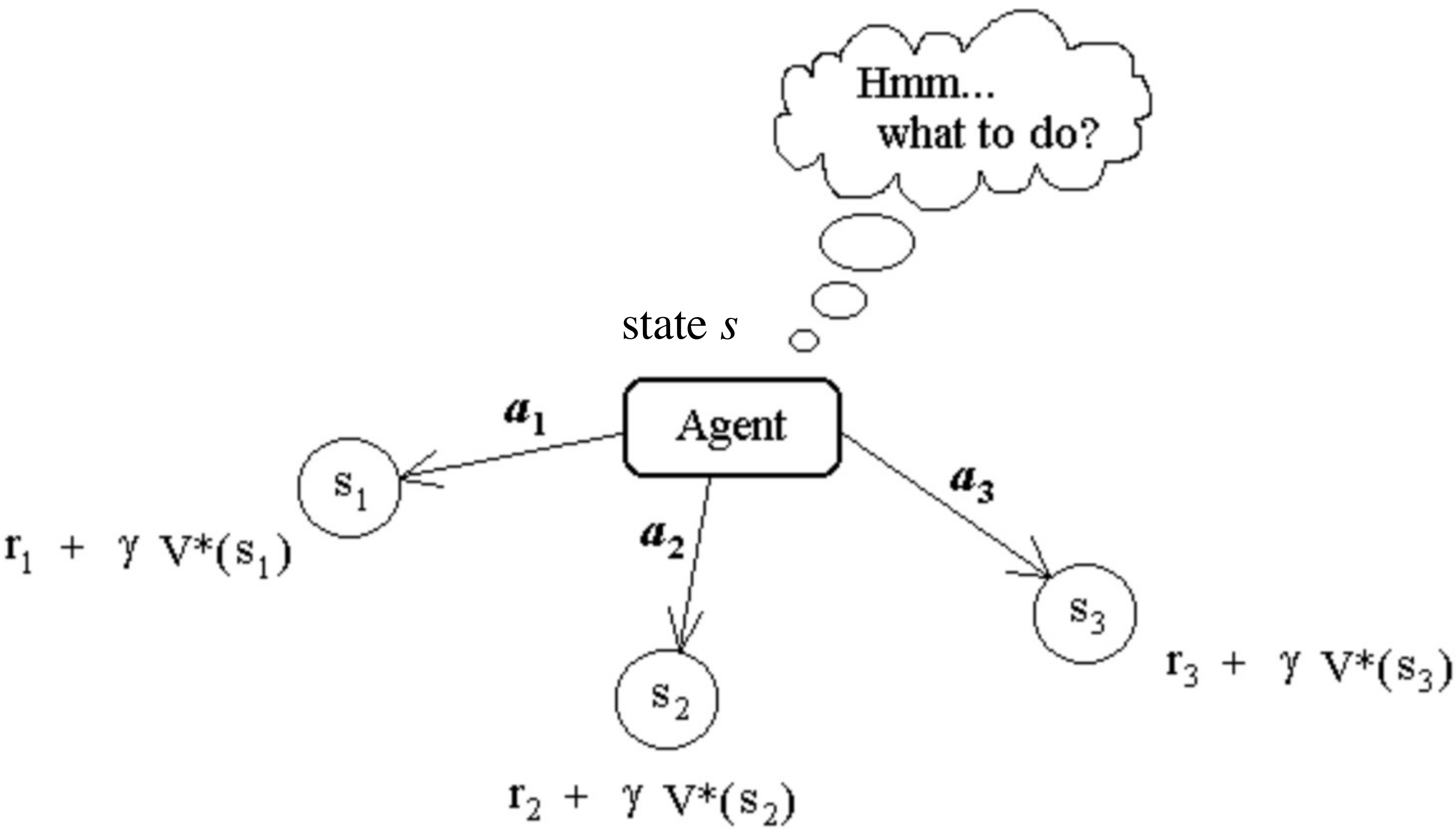
$r_t = r(s_t, a_t)$ is the reward received for performing action a_t in state s_t

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

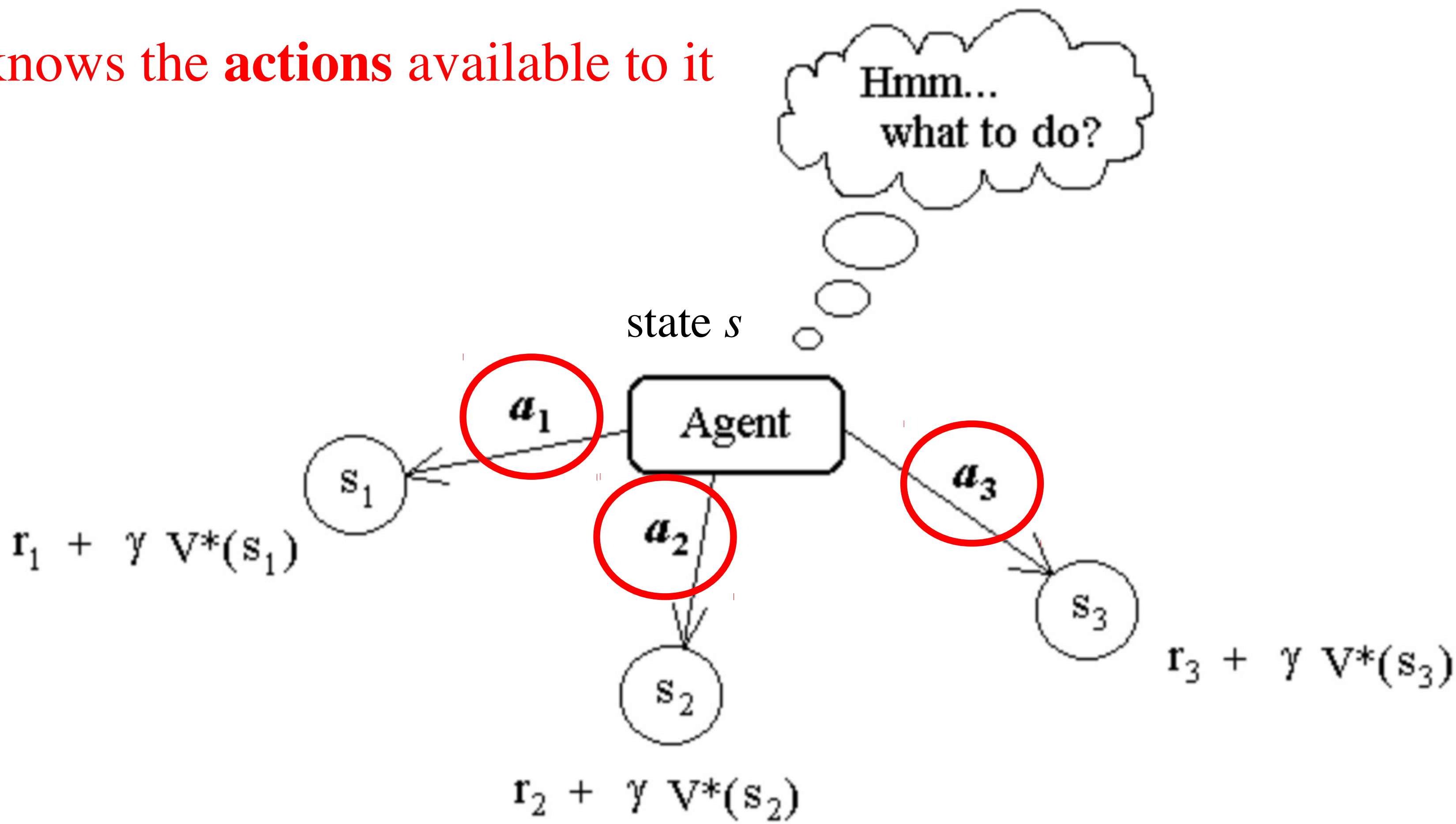
r and δ may be **nondeterministic** or **unknown** to the agent, meaning that the agent may be unable to predict the results of its actions

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*:



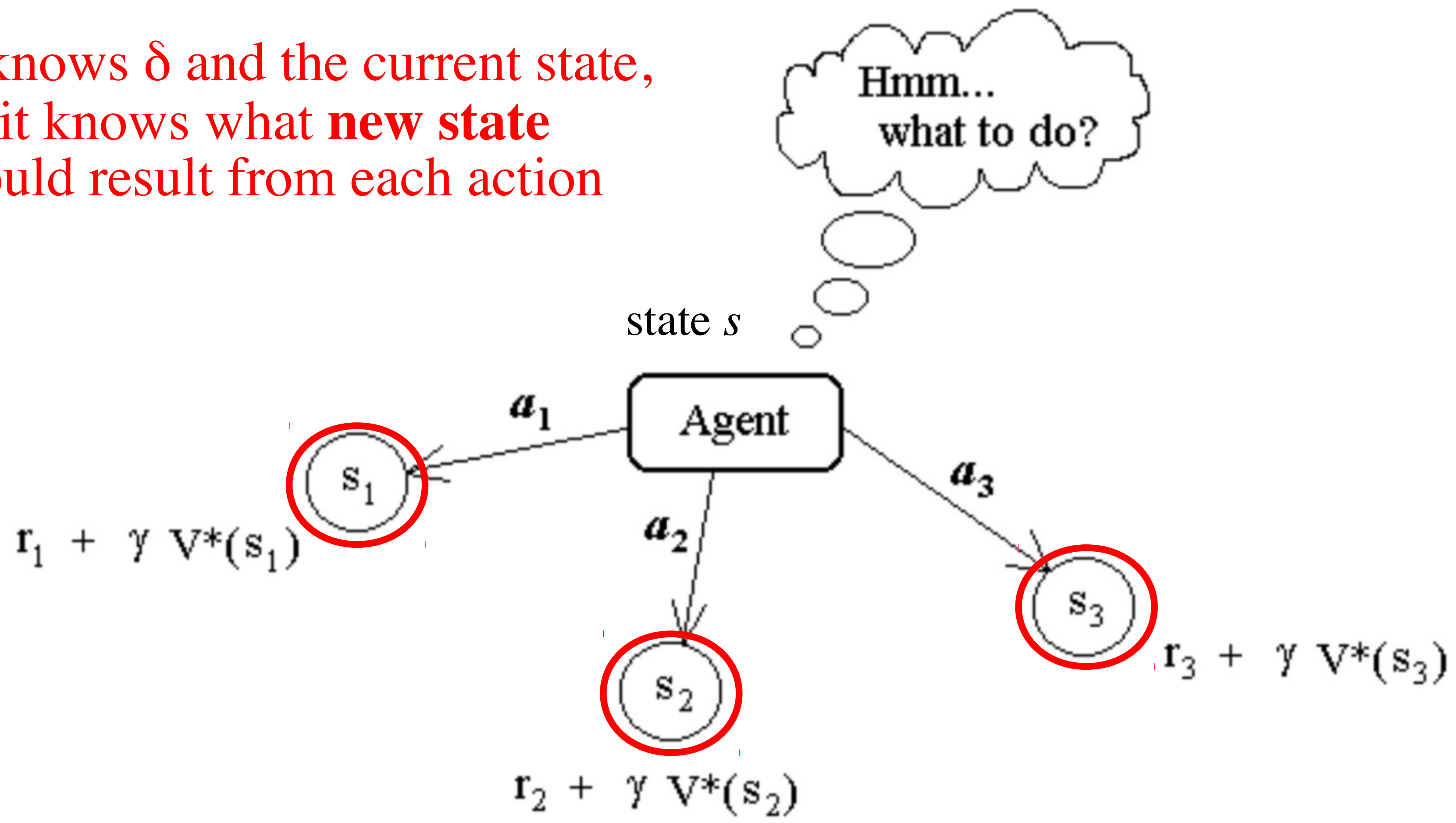
If an agent had **perfect knowledge** of the value function V*, the reward function *r*, and the state transition function δ, it could always choose the best action from any state *s*:

It knows the **actions** available to it



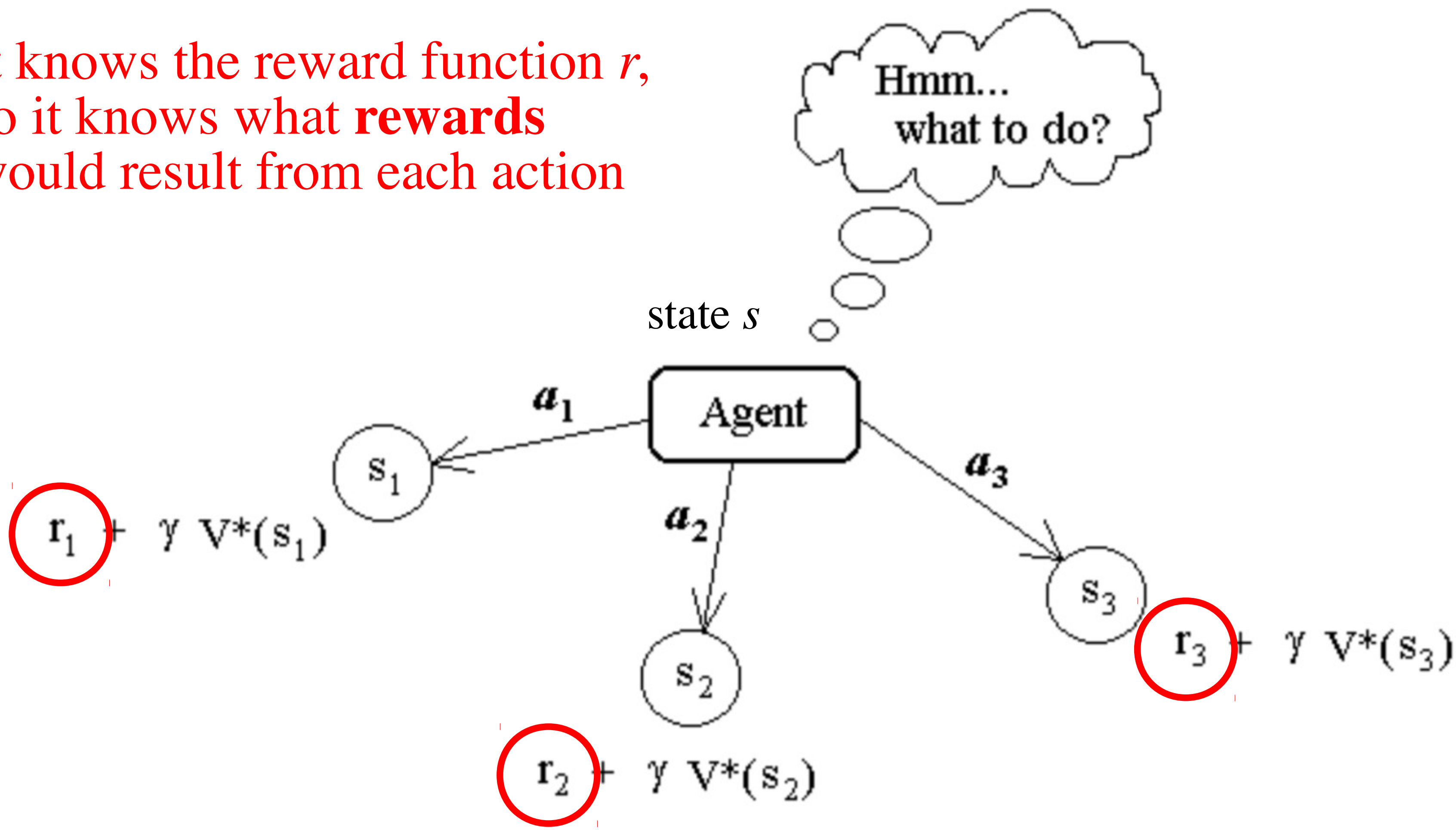
If an agent had **perfect knowledge** of the value function V*, the reward function *r*, and the state transition function δ, it could always choose the best action from any state *s*:

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



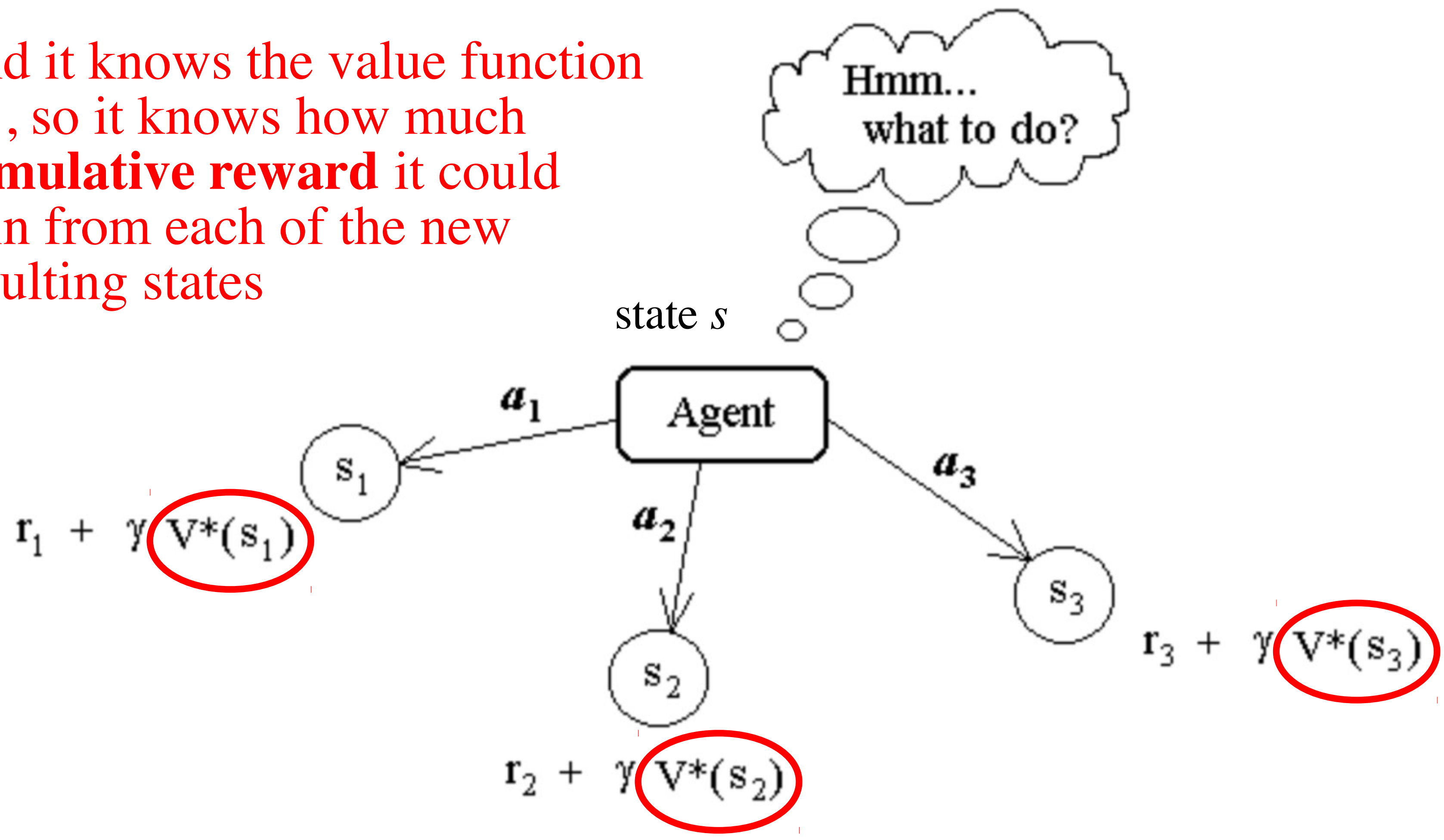
If an agent had **perfect knowledge** of the value function V*, the reward function *r*, and the state transition function δ, it could always choose the best action from any state *s*:

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



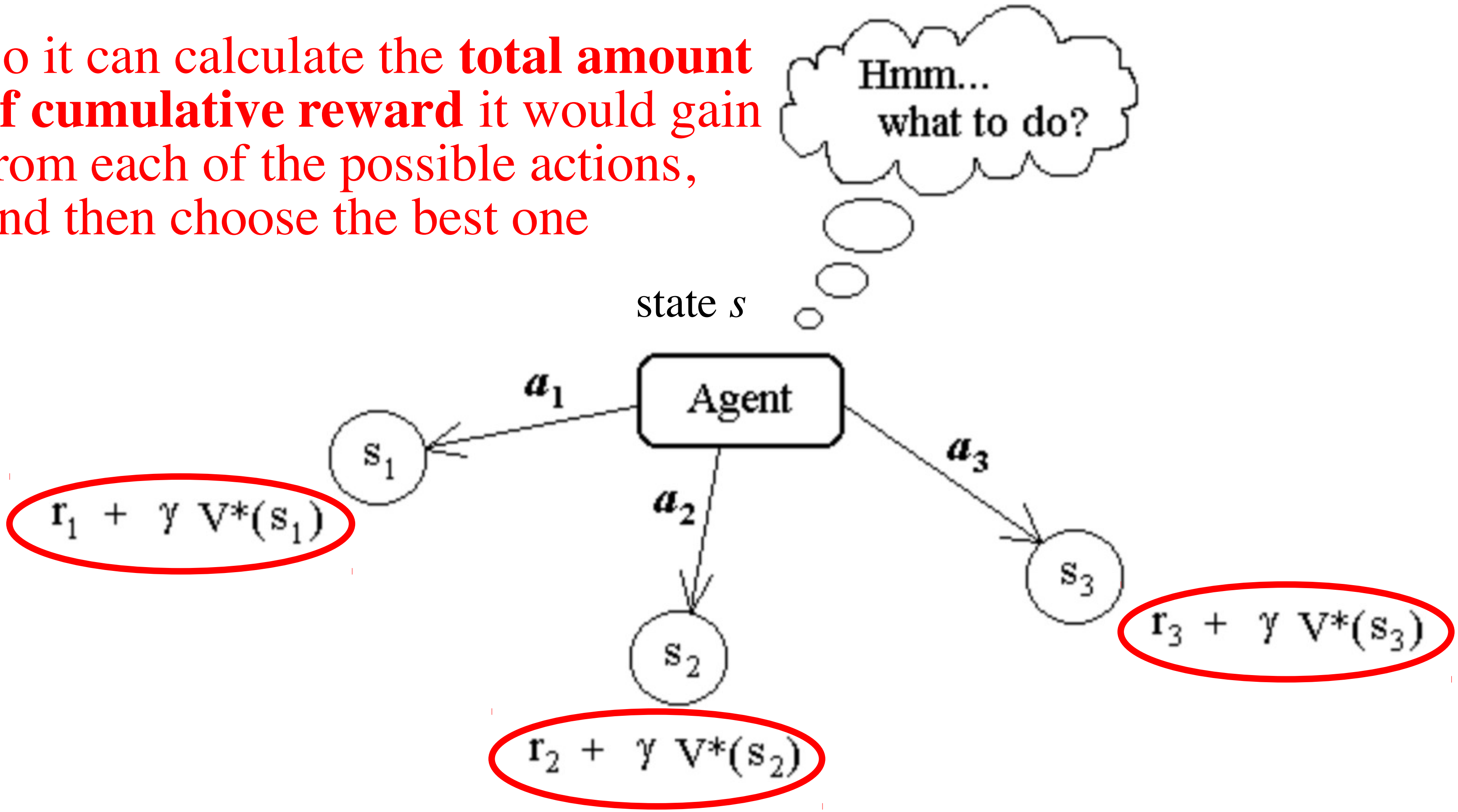
If an agent had **perfect knowledge** of the value function V*, the reward function *r*, and the state transition function δ, it could always choose the best action from any state *s*:

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

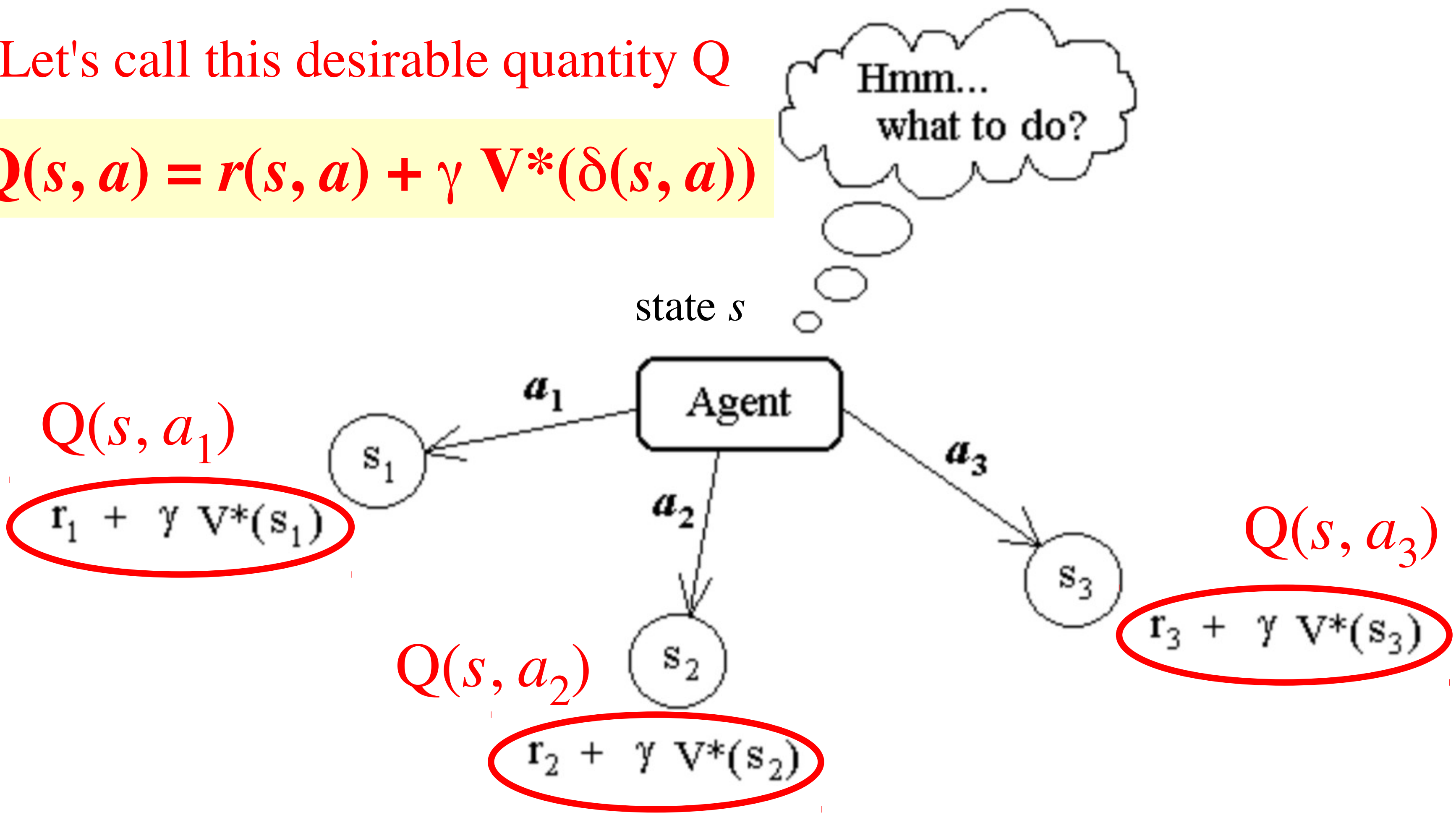


But the agent doesn't actually know any of these functions!
It only knows the **current state *s*** and the available **actions a_1, a_2, a_3**

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

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

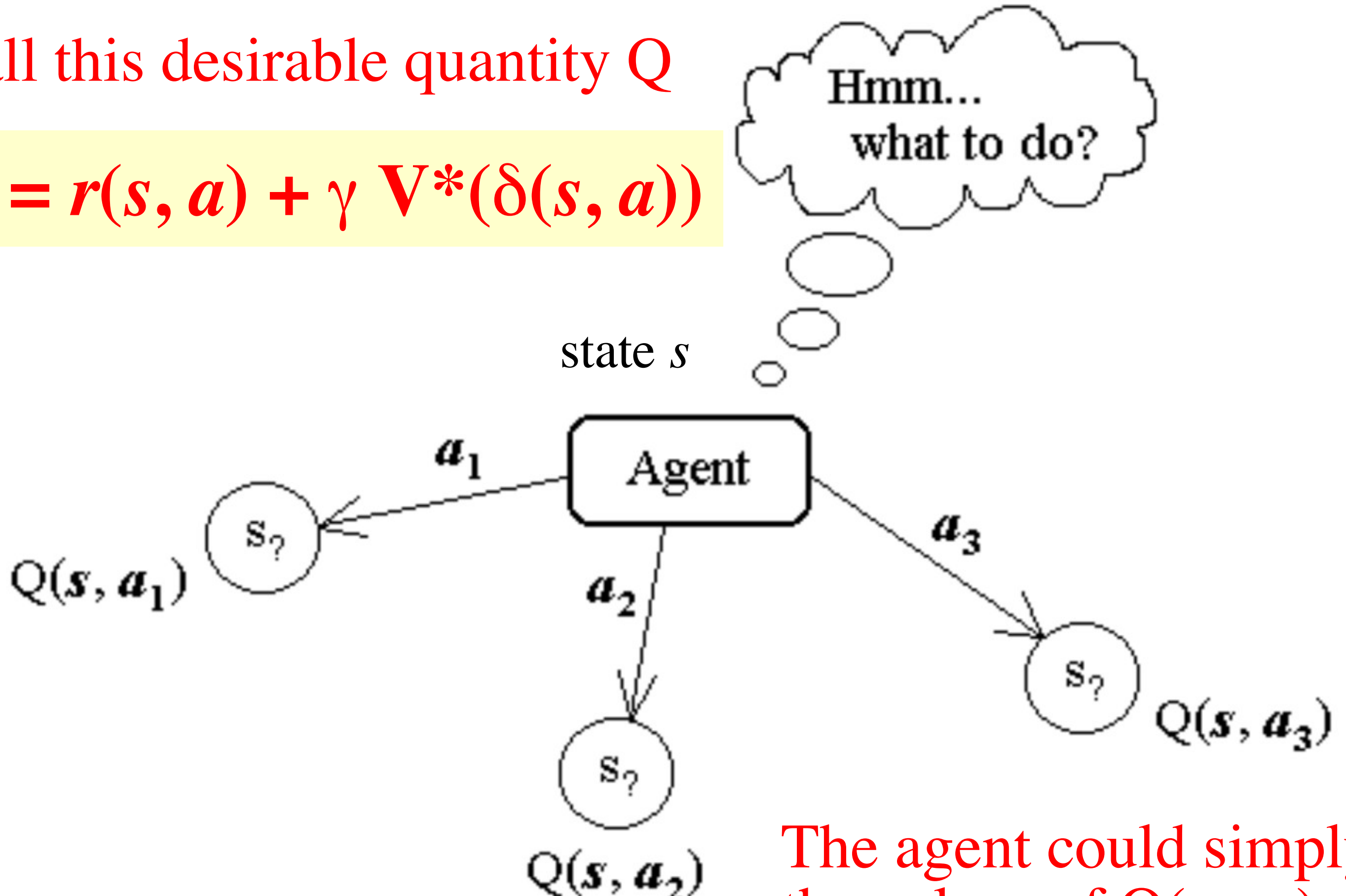


But the agent doesn't actually know any of these functions!
It only knows the **current state *s*** and the available **actions a_1, a_2, a_3**

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

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

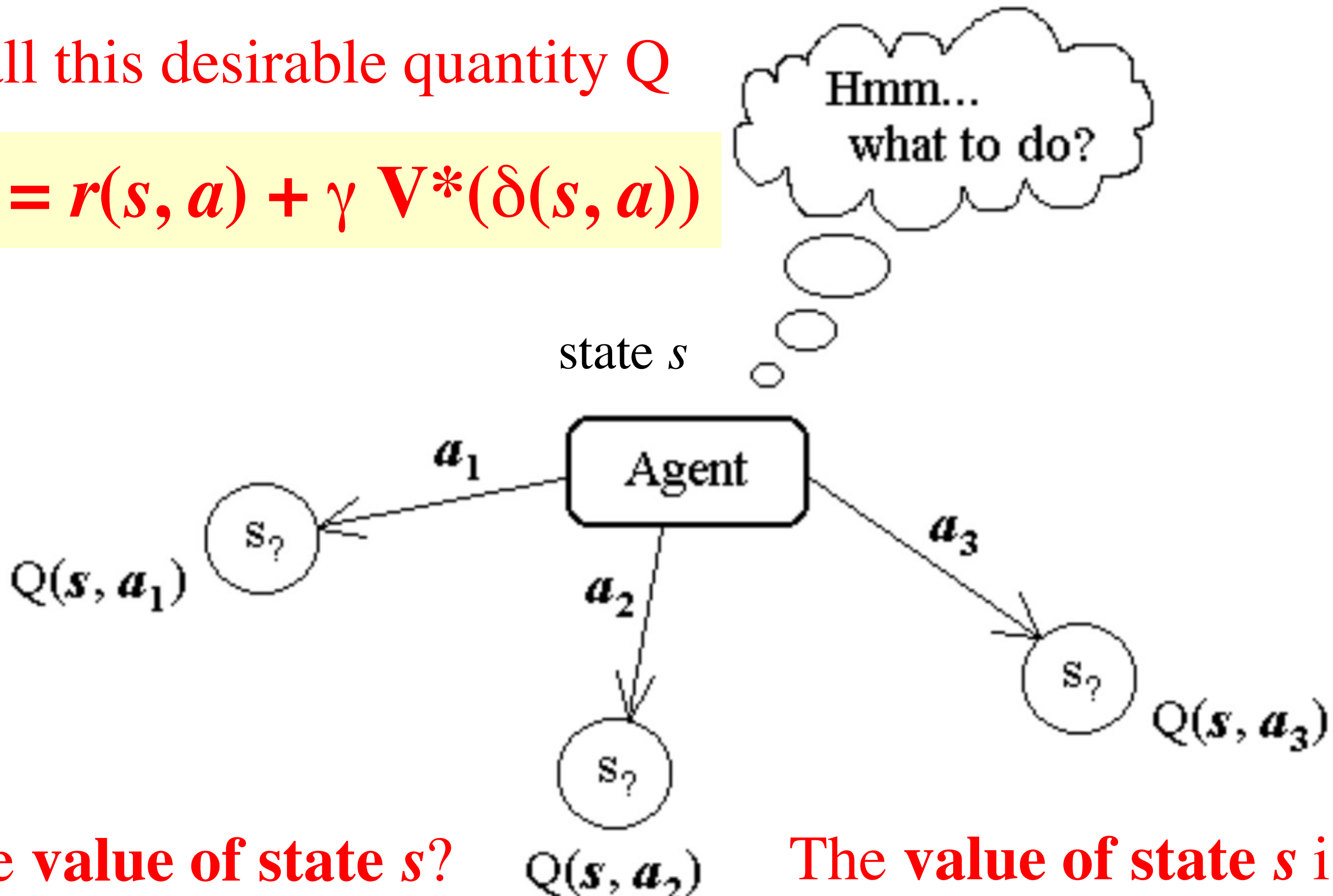


The agent could simply compare the values of $Q(s, a_1)$, $Q(s, a_2)$, and $Q(s, a_3)$ **directly**, and choose the best action

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

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



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

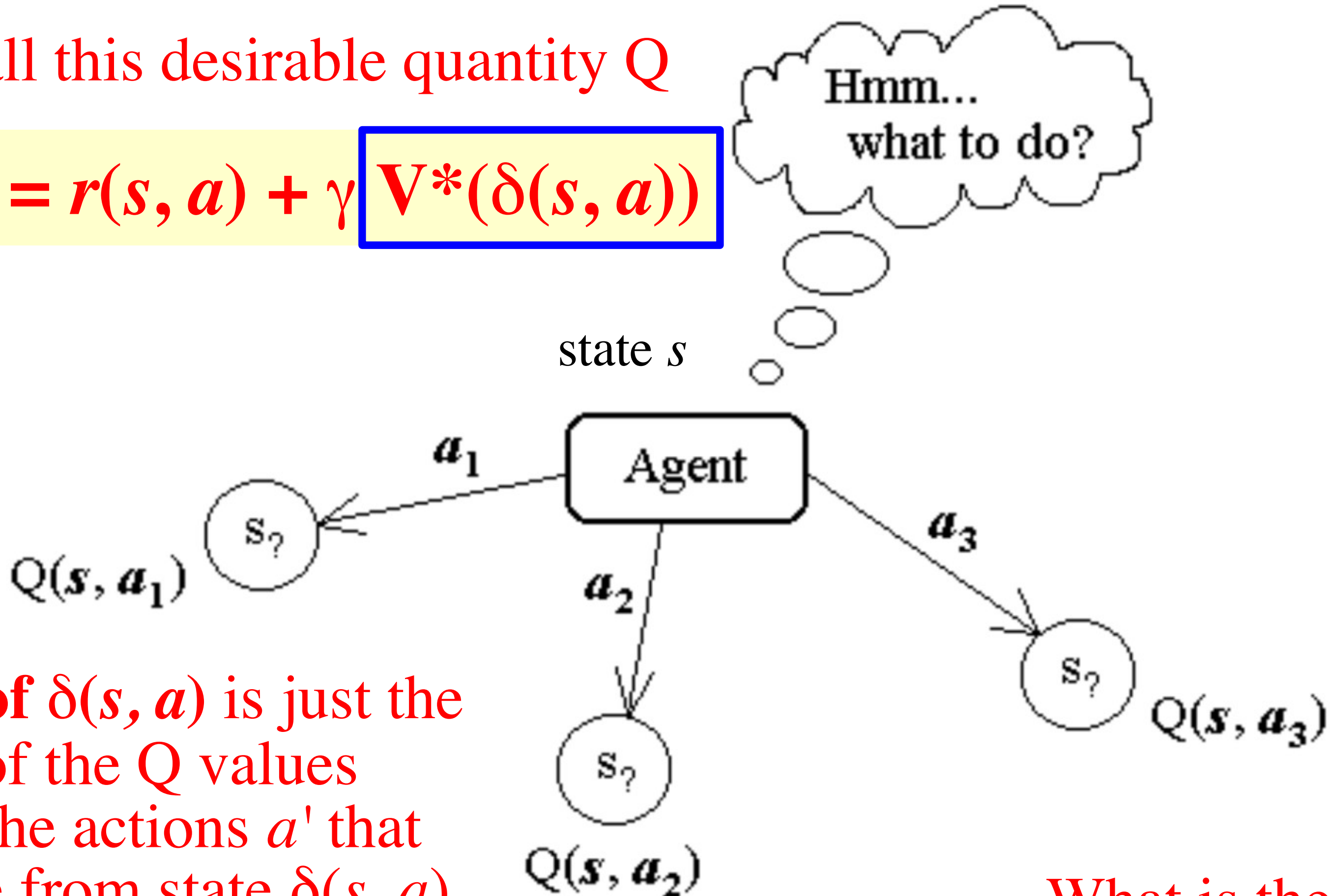
$$V^*(s) = \max_{a'}\{Q(s, a')\}$$

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

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



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 $\delta(s, a)$

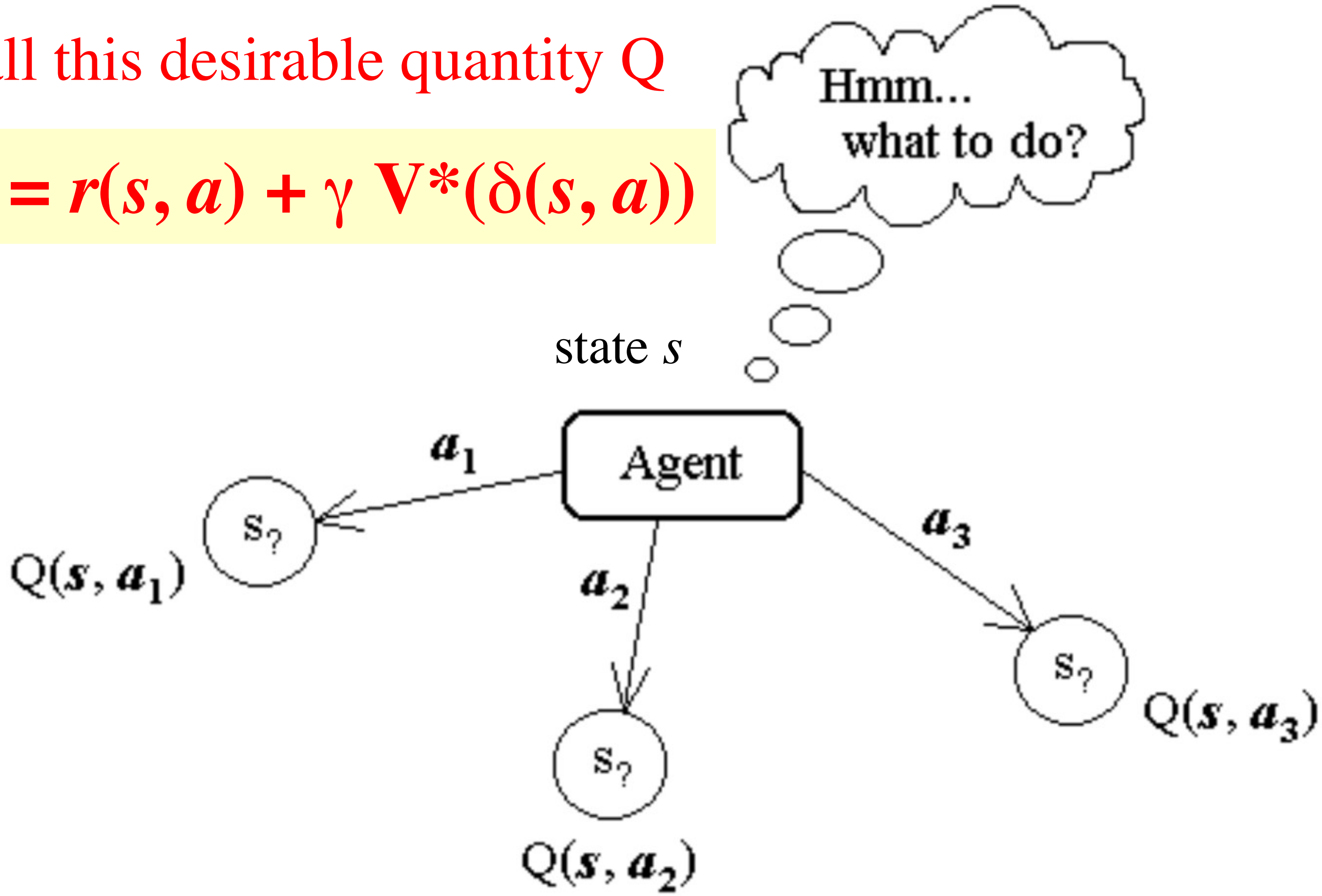
$$V^*(\delta(s, a)) = \max_{a'}\{Q(\delta(s, a), a')\}$$

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

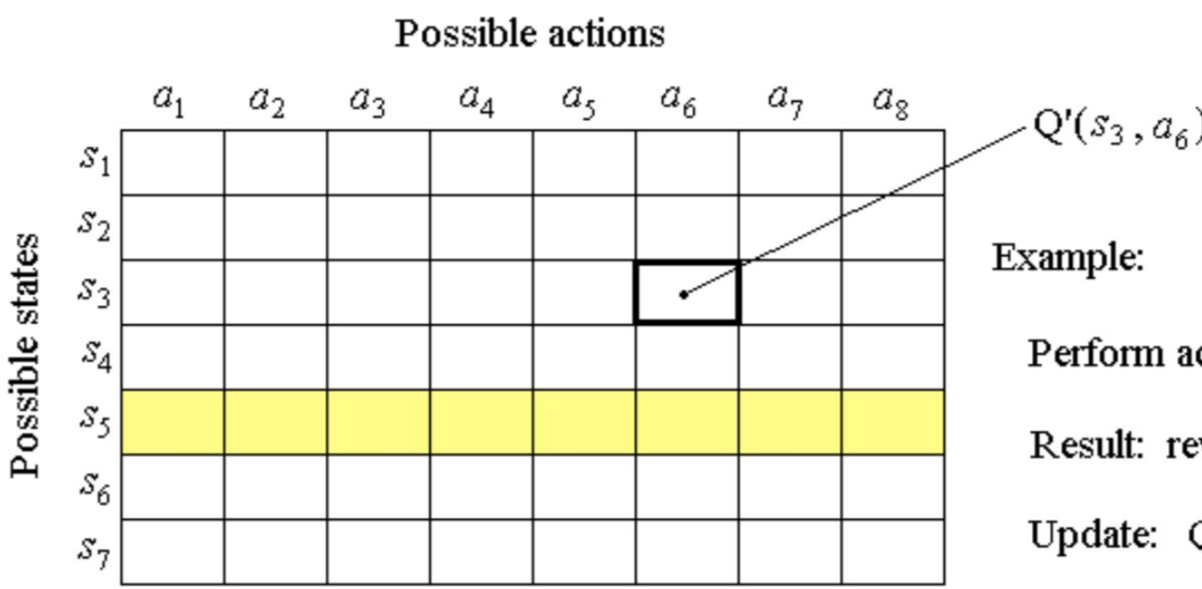
$$Q(s, a) = r(s, a) + \gamma\, V^*(\delta(s, a))$$



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

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:



Example:

Perform action a_6 in state s_3

Result: reward = 2 , new state = s_5

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

Update rule for action a in state s

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

- **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

Possible Actions

Situation		North	South	East	West	StayPut	PickUpCan	RandomMove
#1	**EEEEE**							
#2	**EEEEC**							
#3	**EEEEW**							
#4	**EEECE**							
#5	**EEECC**							
#6	**EEECW**							
#7	**EEEWE**							
#8	**EEEWC**							
	. . .							
#243	**WWWWW**							

Step 1: determine the current situation

Situation		North	South	East	West	StayPut	PickUpCan	RandomMove
#1	**EEEEE**							
#2	**EEEEC**							
#3	**EEEEW**							
#4	**EEECE**							
#5	**EEECC**							
#6	**EEECW**							
#7	**EEEWE**							
#8	**EEEWC**							
	. . .							
#243	**WWWWW**							

Possible Actions (column headings North through RandomMove)

Step 2: choose an action to perform

Situation	North	South	East	West	StayPut	PickUpCan	RandomMove
#1 **EEEEE**							
#2 **EEEEC**							
#3 **EEEEW**							
#4 **EEECE**							
#5 **EEECC**	*2.9*	*-0.4*	*3.5*	*8.9*	*-6.7*	*10.2*	*4.4*
#6 **EEECW**							
#7 **EEEWE**							
#8 **EEEWC**							
. . .							
#243 **WWWWW**							

Possible Actions

Step 3: observe reward and new situation

Reward: +10

Situation		North	South	East	West	StayPut	PickUpCan	RandomMove
#1	**EEEEE**							
#2	**EEEEC**							
#3	**EEEEW**							
#4	**EEECE**							
#5	**EEECC**	*2.9*	*-0.4*	*3.5*	*8.9*	*-6.7*	*10.2*	*4.4*
#6	**EEECW**							
#7	**EEEWE**							
#8	**EEEWC**							
	. . .							
#243	**WWWWW**							

Step 4: find max Q-value for new situation

Reward: +10

Situation	North	South	East	West	StayPut	PickUpCan	RandomMove
#1 **EEEEE**							
#2 **EEEEC**							
#3 **EEEEW**							
#4 **EEECE**	*-7.0*	*25.5*	*-0.9*	*-2.3*	*11.3*	*3.6*	*-1.2*
#5 **EEECC**	*2.9*	*-0.4*	*3.5*	*8.9*	*-6.7*	*10.2*	*4.4*
#6 **EEECW**							
#7 **EEEWE**							
#8 **EEEWC**							
. . .							
#243 **WWWWW**							

Step 5: update Q-value for previous action

Reward: +10

Situation	North	South	East	West	StayPut	PickUpCan	RandomMove
#1 **EEEEE**							
#2 **EEEEC**							
#3 **EEEEW**							
#4 **EEECE**	*-7.0*	*25.5*	*-0.9*	*-2.3*	*11.3*	*3.6*	*-1.2*
#5 **EEECC**	*2.9*	*-0.4*	*3.5*	*8.9*	*-6.7*	***30.4***	*4.4*
#6 **EEECW**							
#7 **EEEWE**							
#8 **EEEWC**							
. . .							
#243 **WWWWW**							

observed reward + *discount factor* × MAX(*Q-values for new situation*)

$$10 + 0.8 \times 25.5 = 30.4$$

. . . and repeat many, many times!

Situation	North	South	East	West	StayPut	PickUpCan	RandomMove
#1 EEEEE							
#2 EEEEC							
#3 EEEEW							
#4 EEECE							
#5 EEECC							
#6 EEECW							
#7 EEEWE							
#8 EEEWC							
. . .							
#243 WWWWW							

Possible Actions

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

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

where $0 \leq \alpha \leq 1$

Examples: $\alpha = 0.3$ $\quad 0.7\,X + 0.3\,Y$

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

$\alpha = 0$ $\quad X$

$\alpha = 1$ $\quad Y$

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

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

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

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

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

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

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

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

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

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” approach:

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

With **probability (1 – ε)**, choose the action that maximizes Q(s, a)

Good strategy: start with high ε and gradually decrease it over the run