Markov Decision Process

Roger Kuo

Markov Chain

範例: 天氣狀態轉移

Markov State Diagram

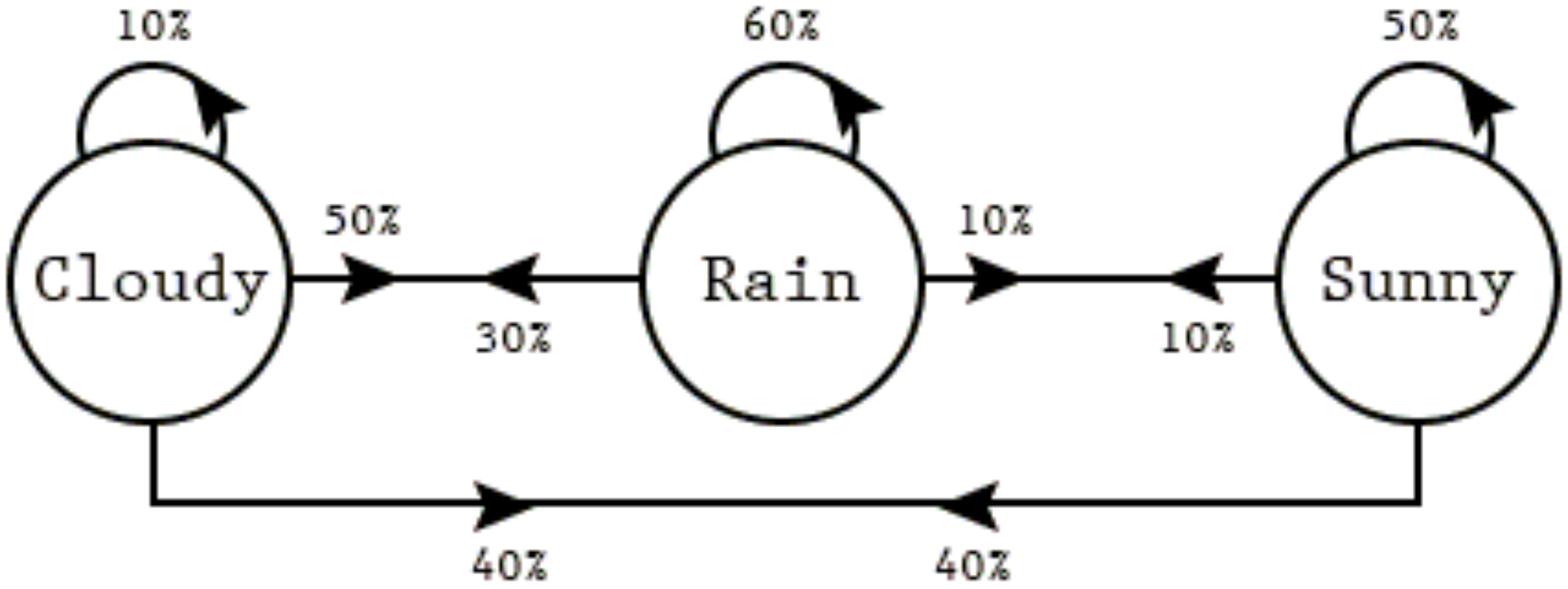
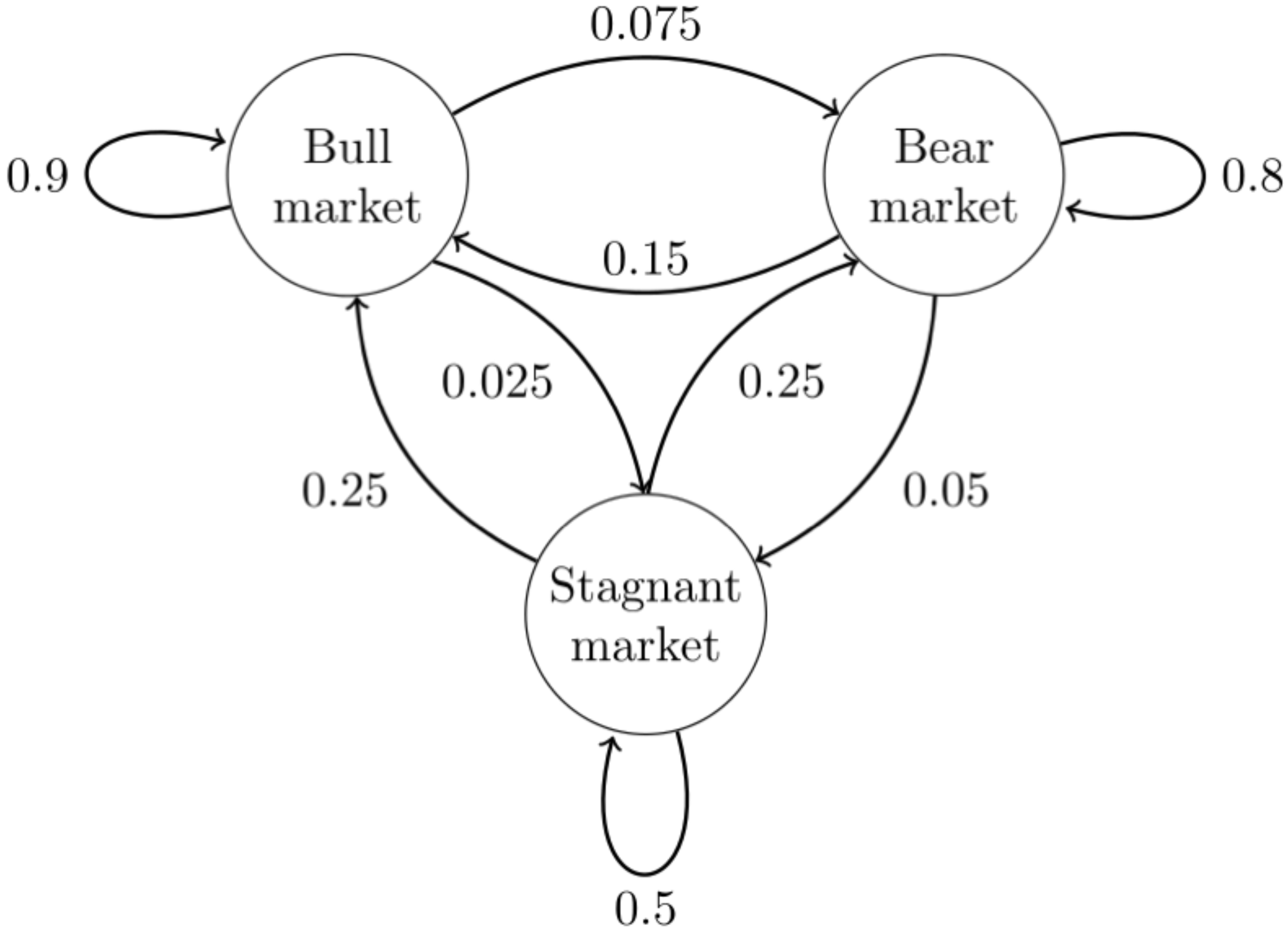


Figure 2

範例: 金融市場變化

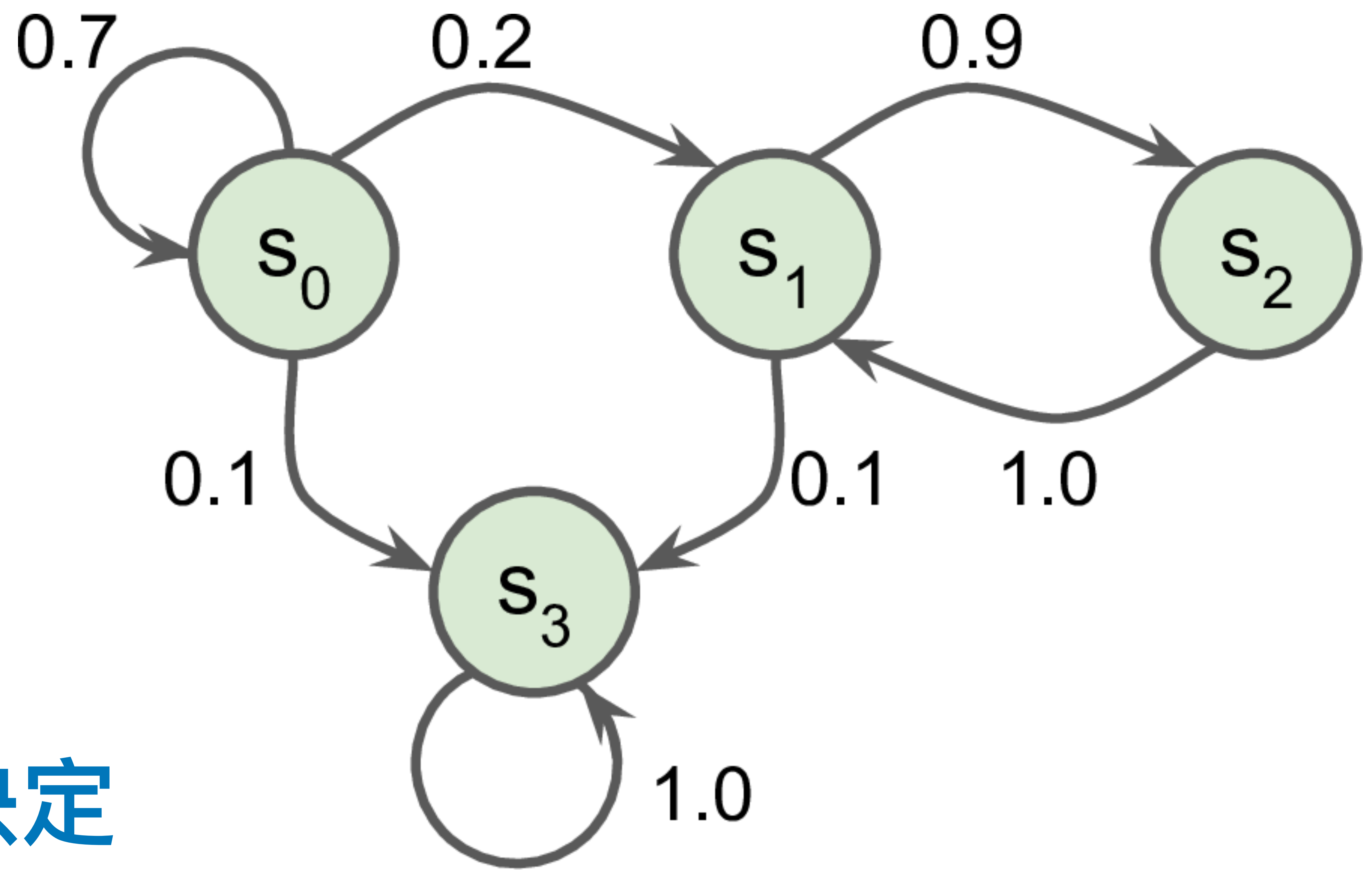

0.075
0.9
Bull
market
Bear
market
0.8
0.15
0.025
0.25
0.25
0.05
Stagnant
market
0.5

特性

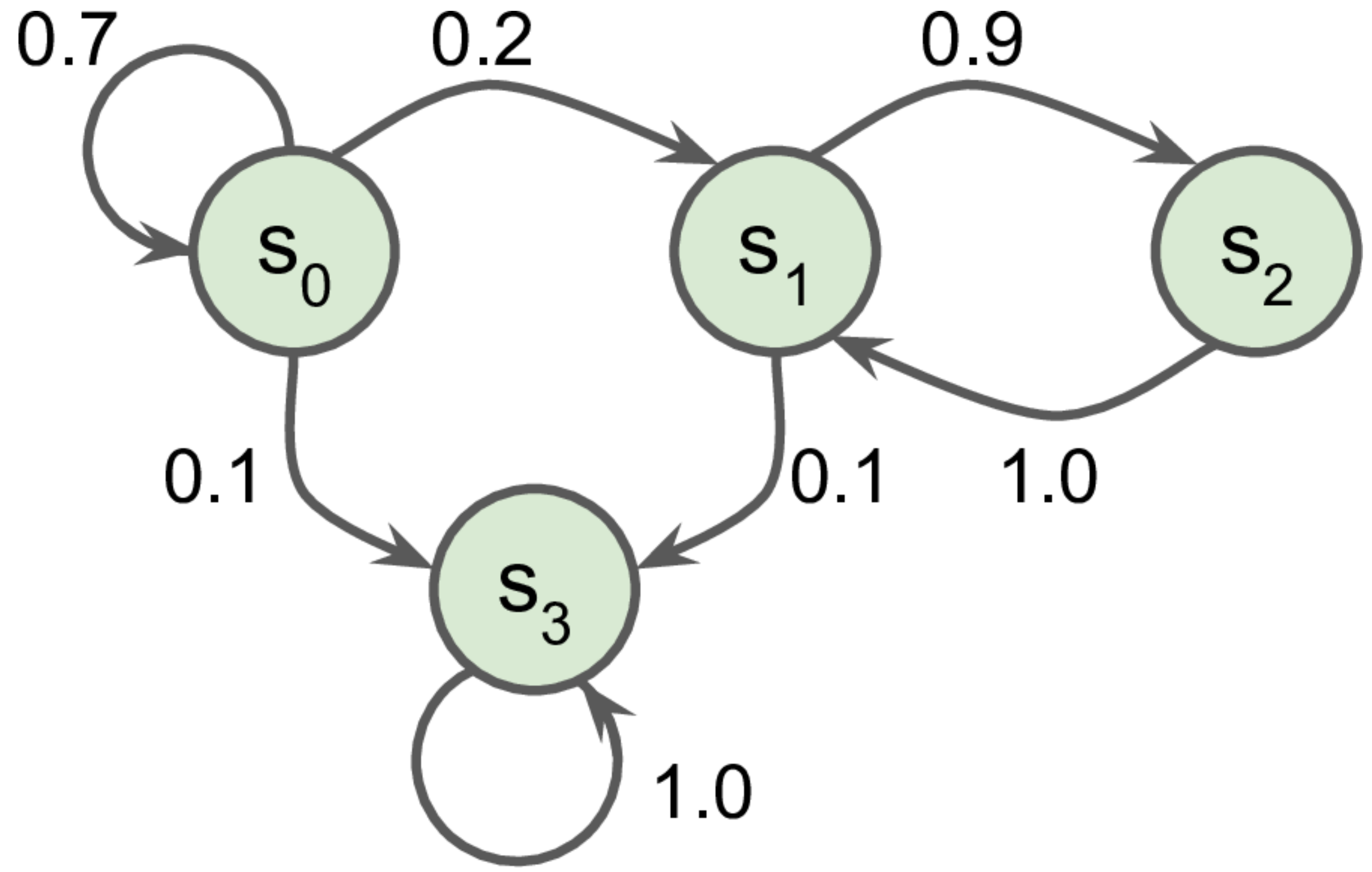
1. 有限狀態 + 轉移機率

```
array([[ 0.7,  0.2,  0. ,  0.1],
       [ 0. ,  0. ,  0.9,  0.1],
       [ 0. ,  1. ,  0. ,  0. ],
       [ 0. ,  0. ,  0. ,  1. ]])
```

2. 無記憶: 下一狀態由當下決定



Markov Model




```
array([ 1.,  0.,  0.,  0.])
```

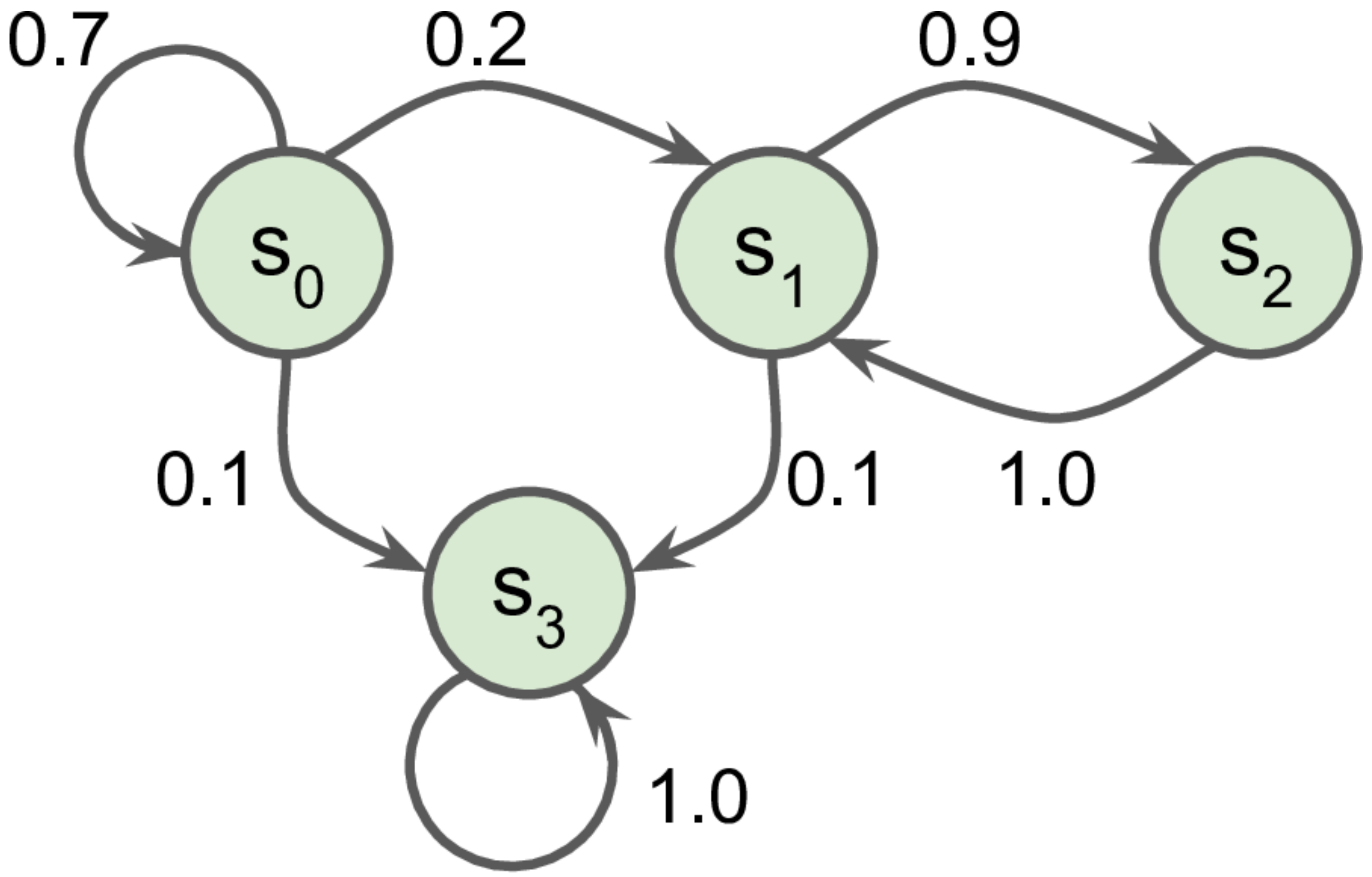
X

```
array([[ 0.7,  0.2,  0. ,  0.1],
       [ 0. ,  0. ,  0.9,  0.1],
       [ 0. ,  1. ,  0. ,  0. ],
       [ 0. ,  0. ,  0. ,  1. ]])
```

=

```
array([ 0.7,  0.2,  0. ,  0.1])
```

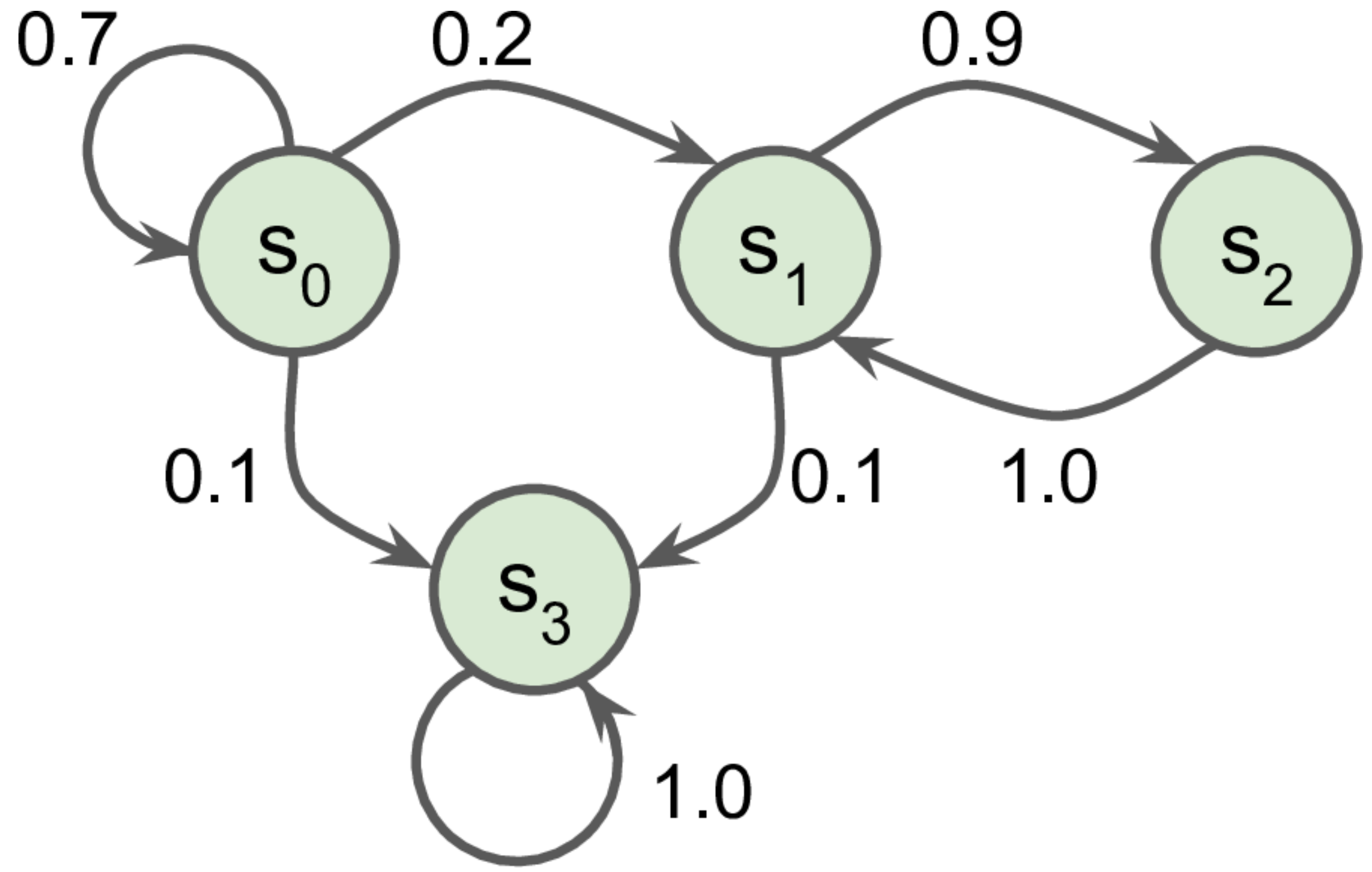
Markov Model



2

```
array([ 1.,   0.,   0.,   0.])   X   array([[ 0.7,  0.2,  0. ,  0.1],
                                          [ 0. ,  0. ,  0.9,  0.1],
                                          [ 0. ,  1. ,  0. ,  0. ],
                                          [ 0. ,  0. ,  0. ,  1. ]])

= array([ 0.49,  0.14,  0.18,  0.19])
```

Markov Model



15

```
array([ 1.,   0.,   0.,   0.])
```

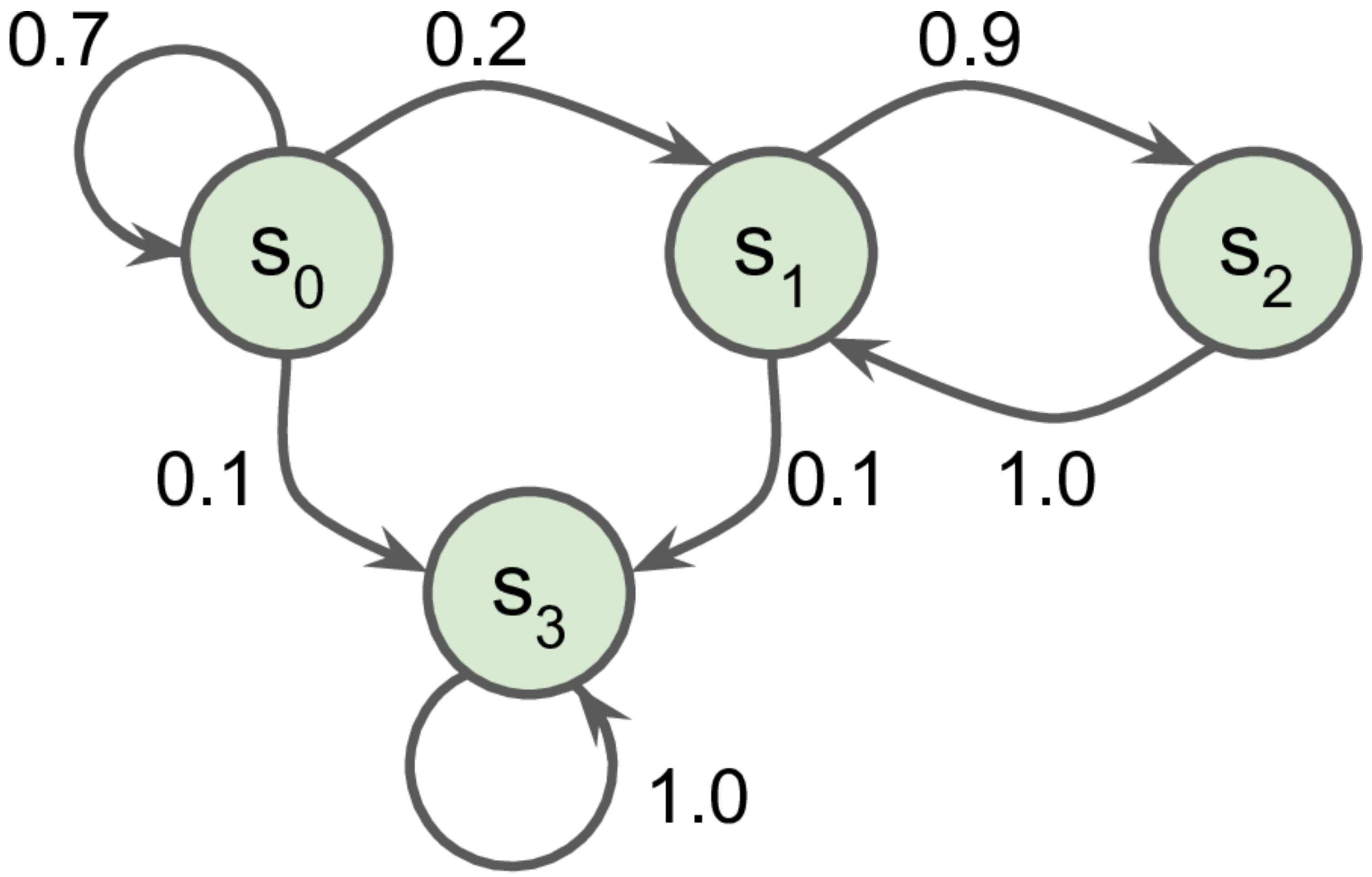
X

```
array([[ 0.7,   0.2,   0. ,   0.1],
       [ 0. ,   0. ,   0.9,   0.1],
       [ 0. ,   1. ,   0. ,   0. ],
       [ 0. ,   0. ,   0. ,   1. ]])
```

=

```
array([ 0.00474756,  0.20836289,  0.14490451,  0.64198504])
```

Markov Model



1000

```
array([ 1.,   0.,   0.,   0.])  X  array([[ 0.7,  0.2,  0. ,  0.1],
                                        [ 0. ,  0. ,  0.9,  0.1],
                                        [ 0. ,  1. ,  0. ,  0. ],
                                        [ 0. ,  0. ,  0. ,  1. ]])
```

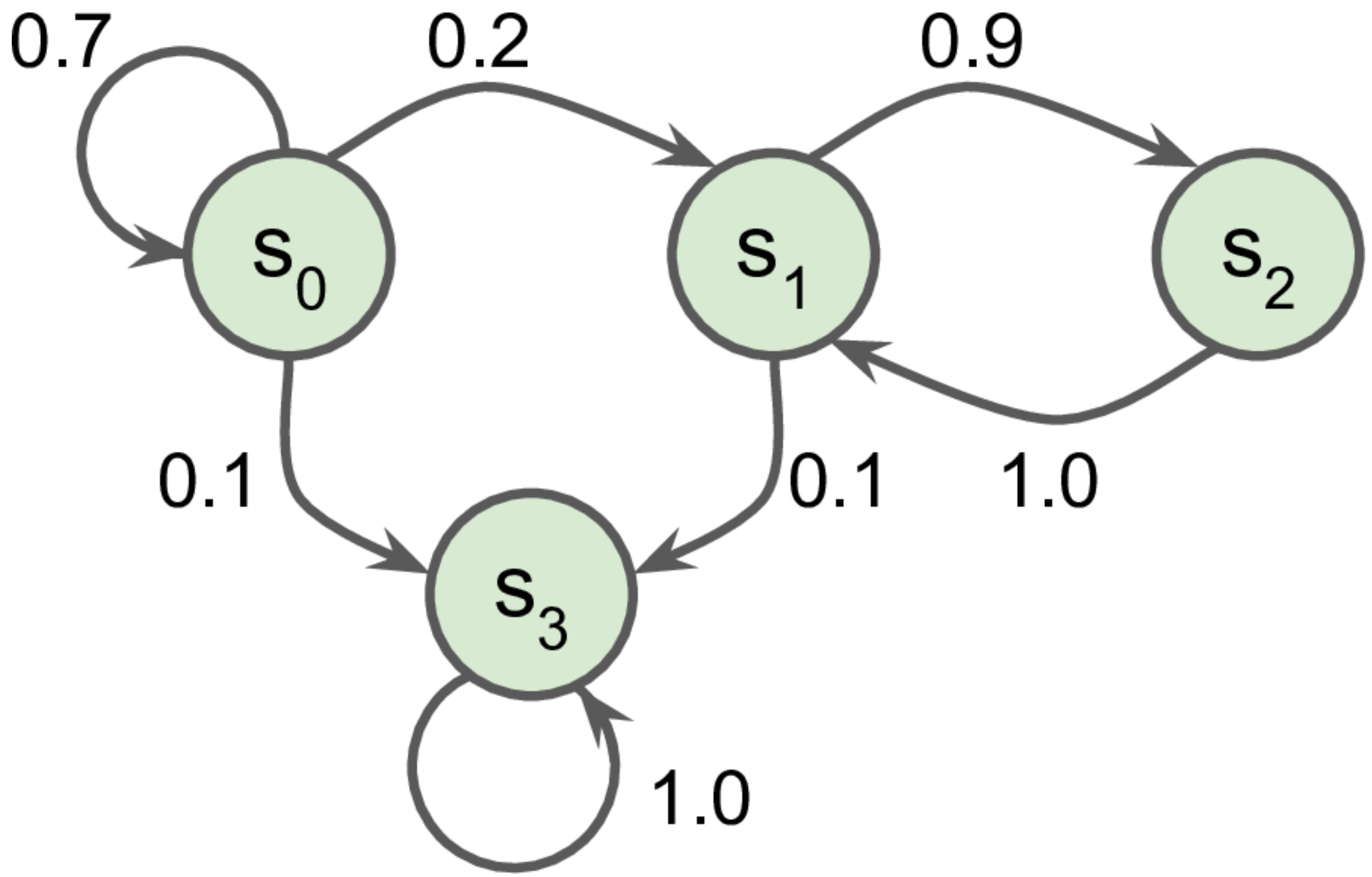
=

```
array([  4.43669541e-154,   7.64515600e-024,   9.82948628e-024,
         1.00000000e+000])
```

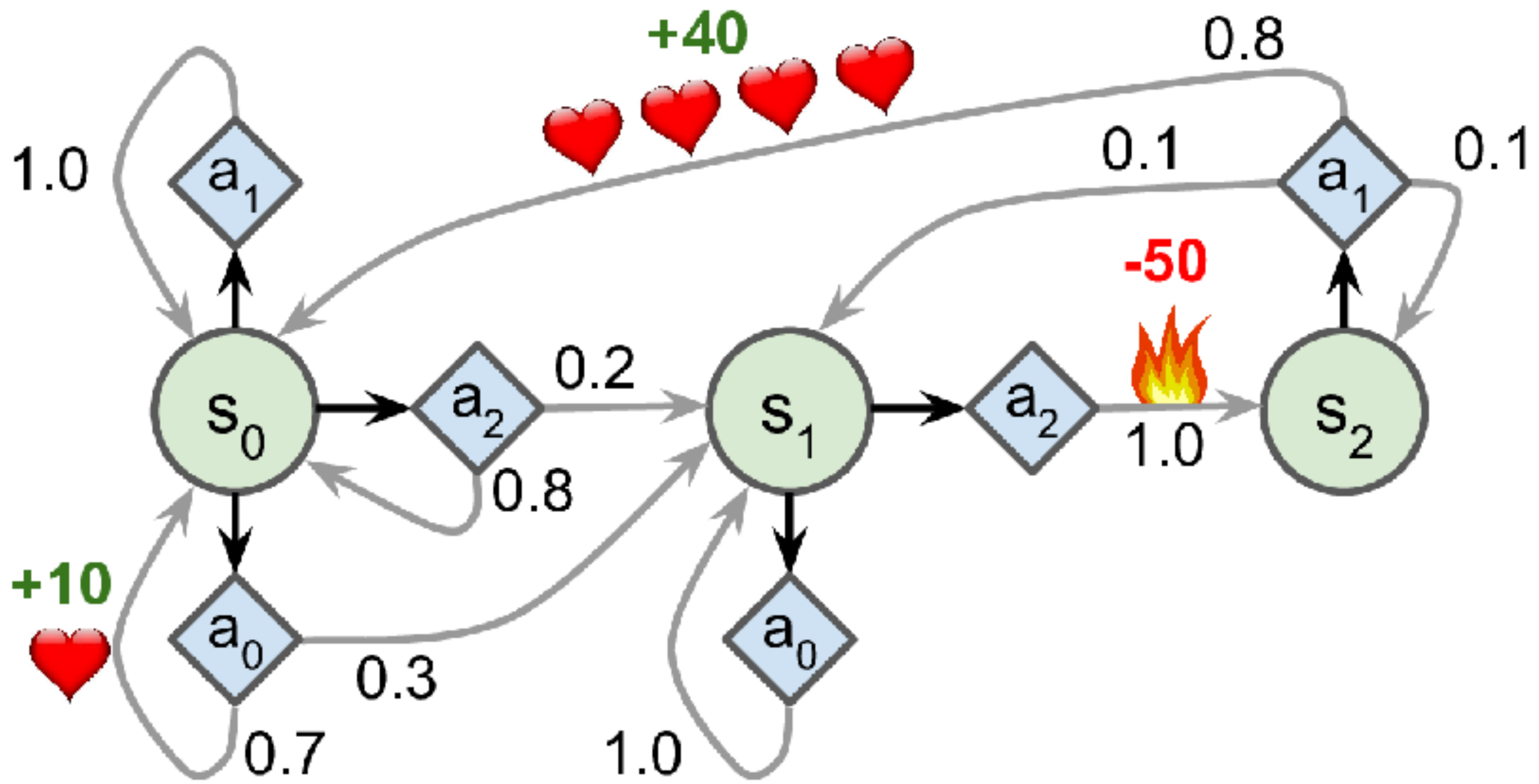
最後會收斂到穩定狀態！

Markov Decision Process ?

不一樣在哪裡？



Markov Chain



MDPs

S1 **States**

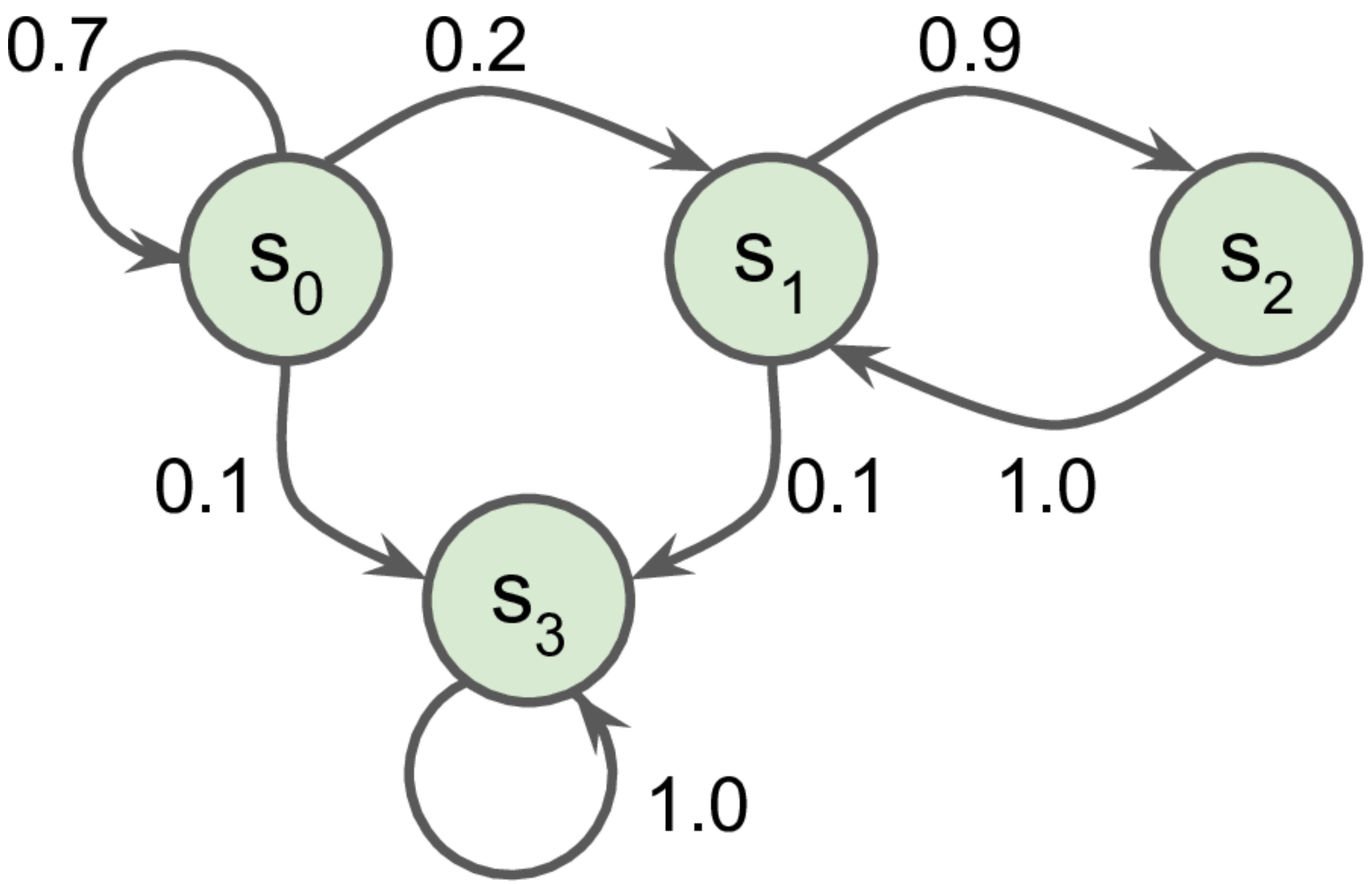
0.7 **Transition Probs**

S1 **States**

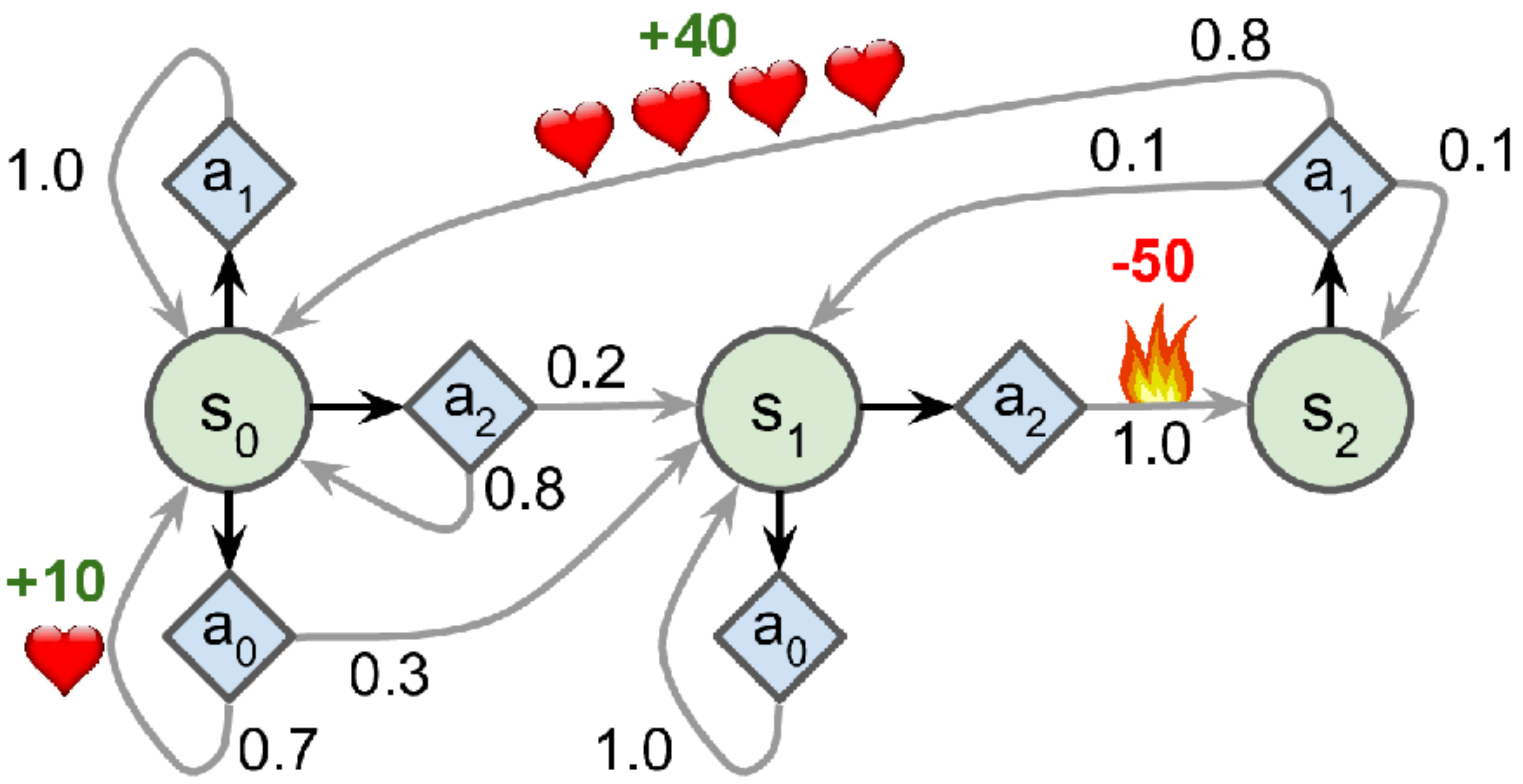
0.7 **Transition Probs**

a1 **Action**

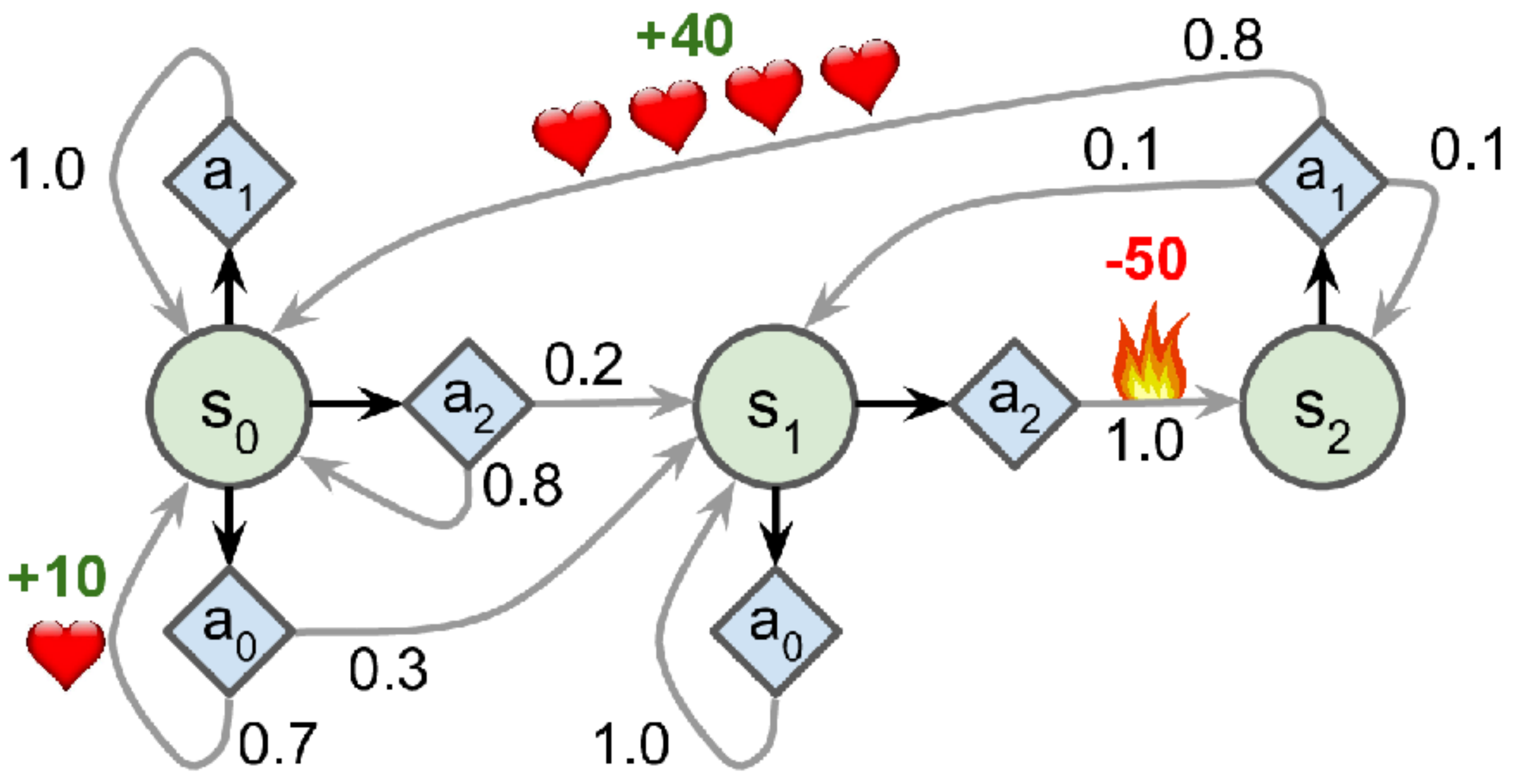
+5 **Reward**



Markov Chain

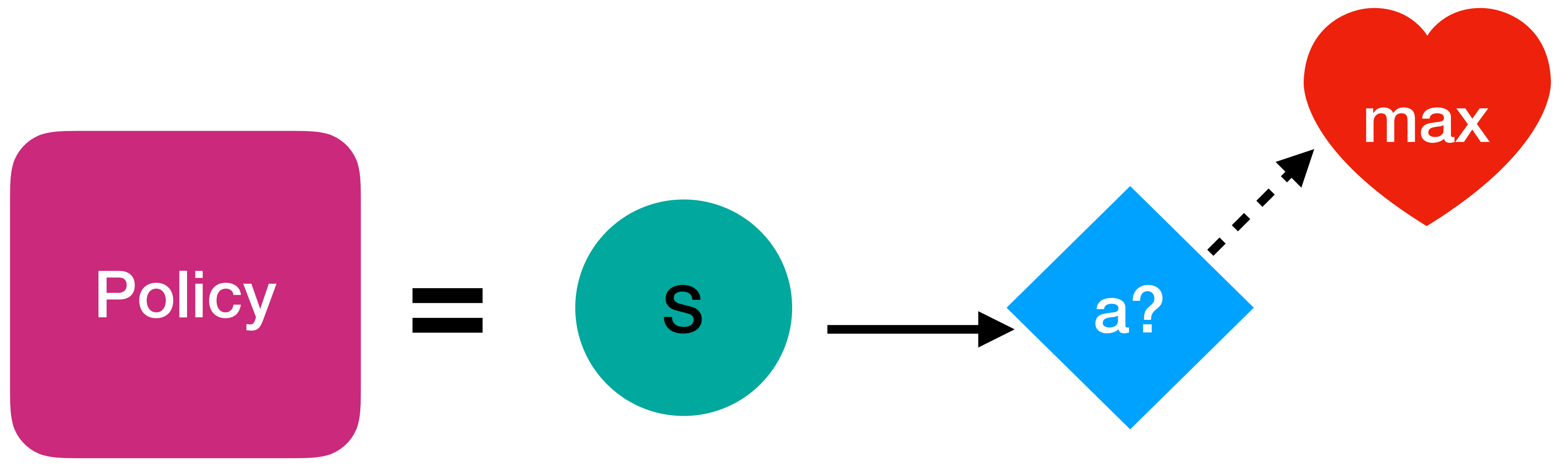


MDPs



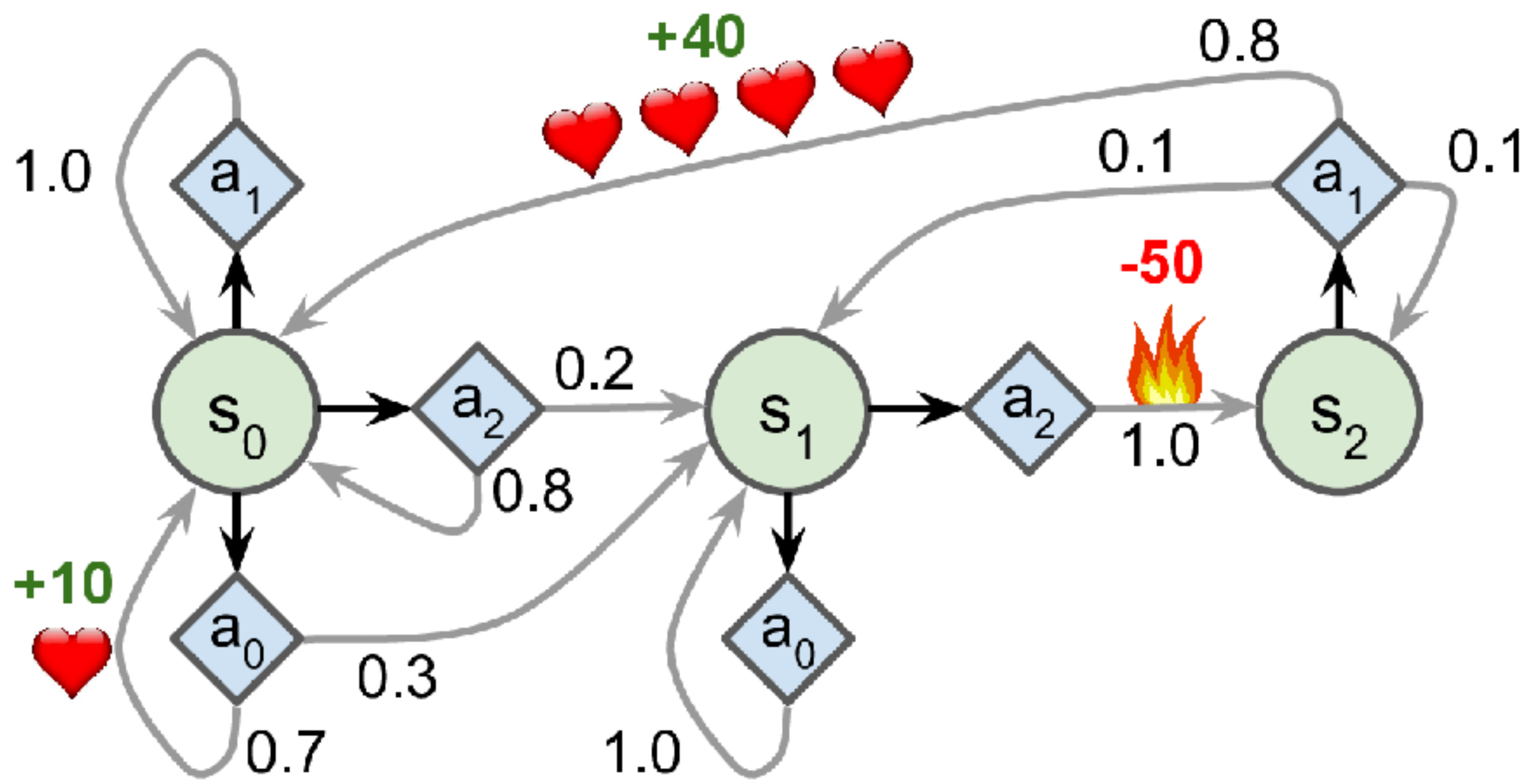
MDPs

目標



在當下的狀態該選什麼*action*
會最大化未來*Rewards?*

Bellman

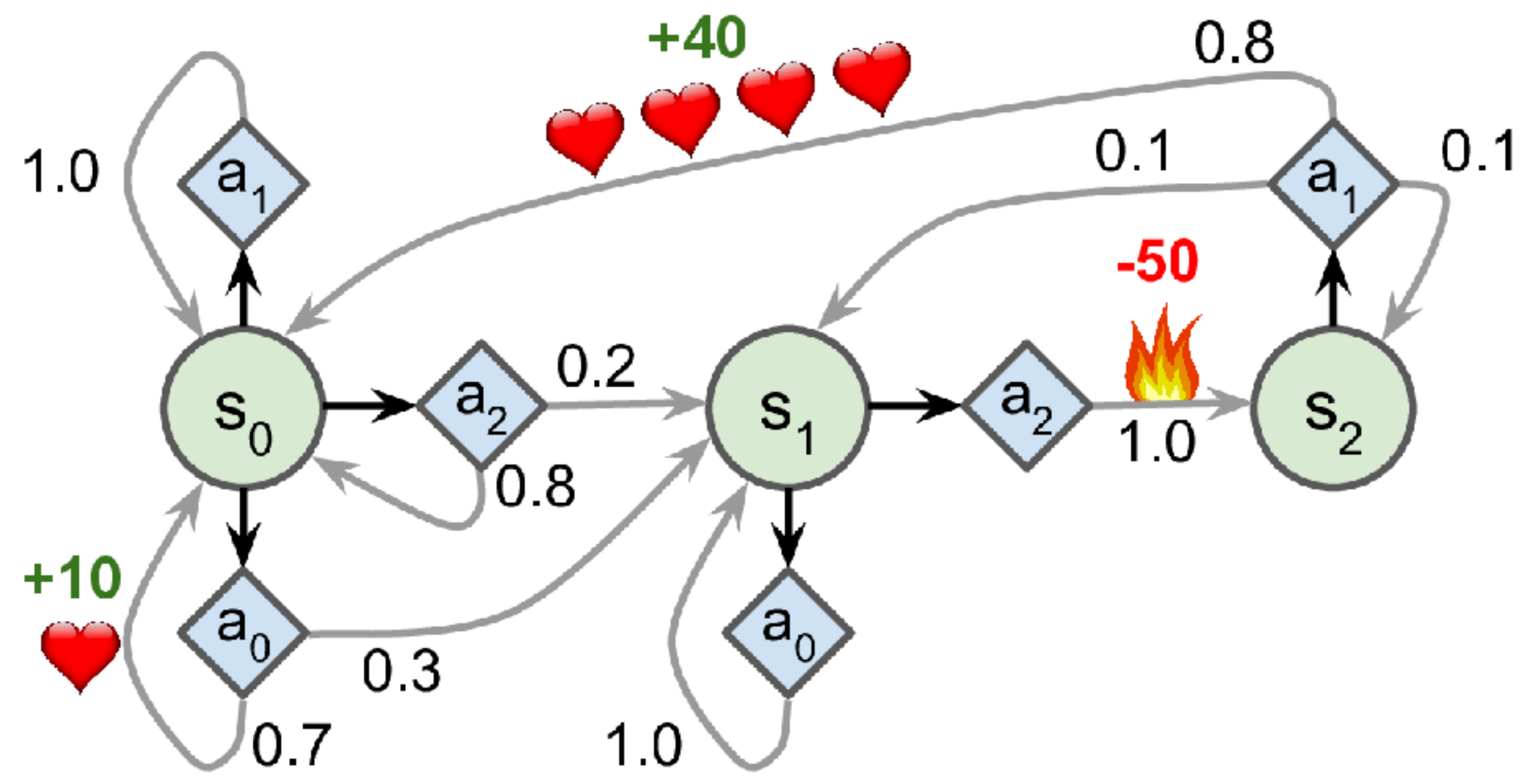


MDPs

Bellman Optimality Equation

$$V^*(s) = \max_a \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma . V^*(s')] \quad \text{for all } s$$

在行使最佳*action*時, 每個狀態的期望價值！



$$V^*(s) = \max_a \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma . V^*(s')] \quad \text{for all } s$$

轉移機率

Reward

discount rate

```
T = np.array([  # shape=[s, a, s']
        [[0.7, 0.3, 0.0], [1.0, 0.0, 0.0], [0.8, 0.2, 0.0]],
        [[0.0, 1.0, 0.0], [nan, nan, nan], [0.0, 0.0, 1.0]],
        [[nan, nan, nan], [0.8, 0.1, 0.1], [nan, nan, nan]],
    ])
```

```
R = np.array([  # shape=[s, a, s']
        [[10., 0.0, 0.0], [0.0, 0.0, 0.0], [0.0, 0.0, 0.0]],
        [[0.0, 0.0, 0.0], [nan, nan, nan], [0.0, 0.0, -50.]],
        [[nan, nan, nan], [40., 0.0, 0.0], [nan, nan, nan]],
    ])
```

怎麼算？

Bellman Optimality Equation

$$V^{*}(s) = \max_a \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma . V^{*}(s')] \quad \text{for all } s$$

Value Iteration algorithm

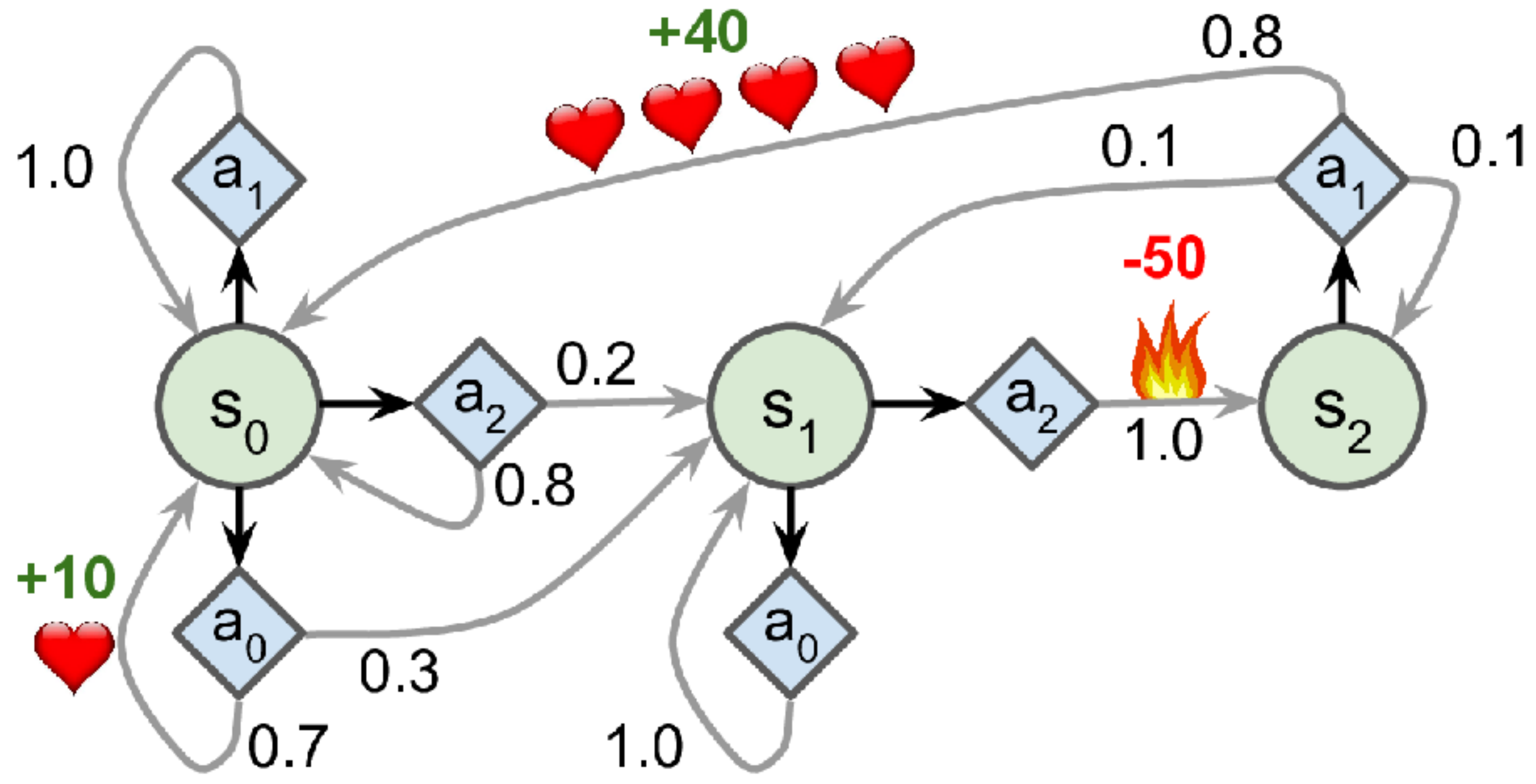
$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma . V_k(s')] \quad \text{for all } s$$

遞迴算出個狀態價值 -> 會收斂

discount rate (r) = 0.9

	S0	S1	S2
V0	0	0	0
V1	7	0	32
V2	11.41	0	39.92
...		...	
V100	18.91	0	50.133

收斂



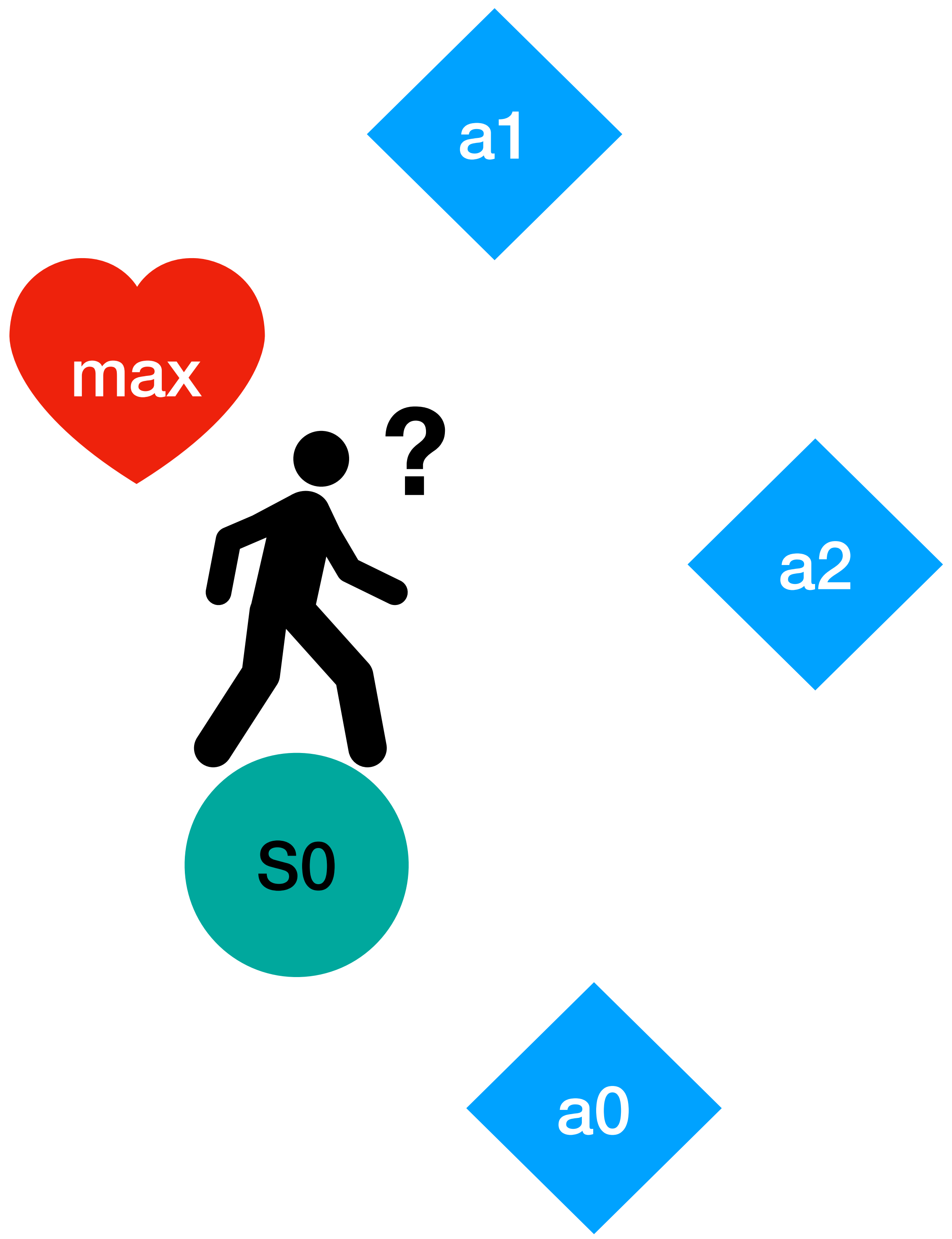
Value Iteration algorithm

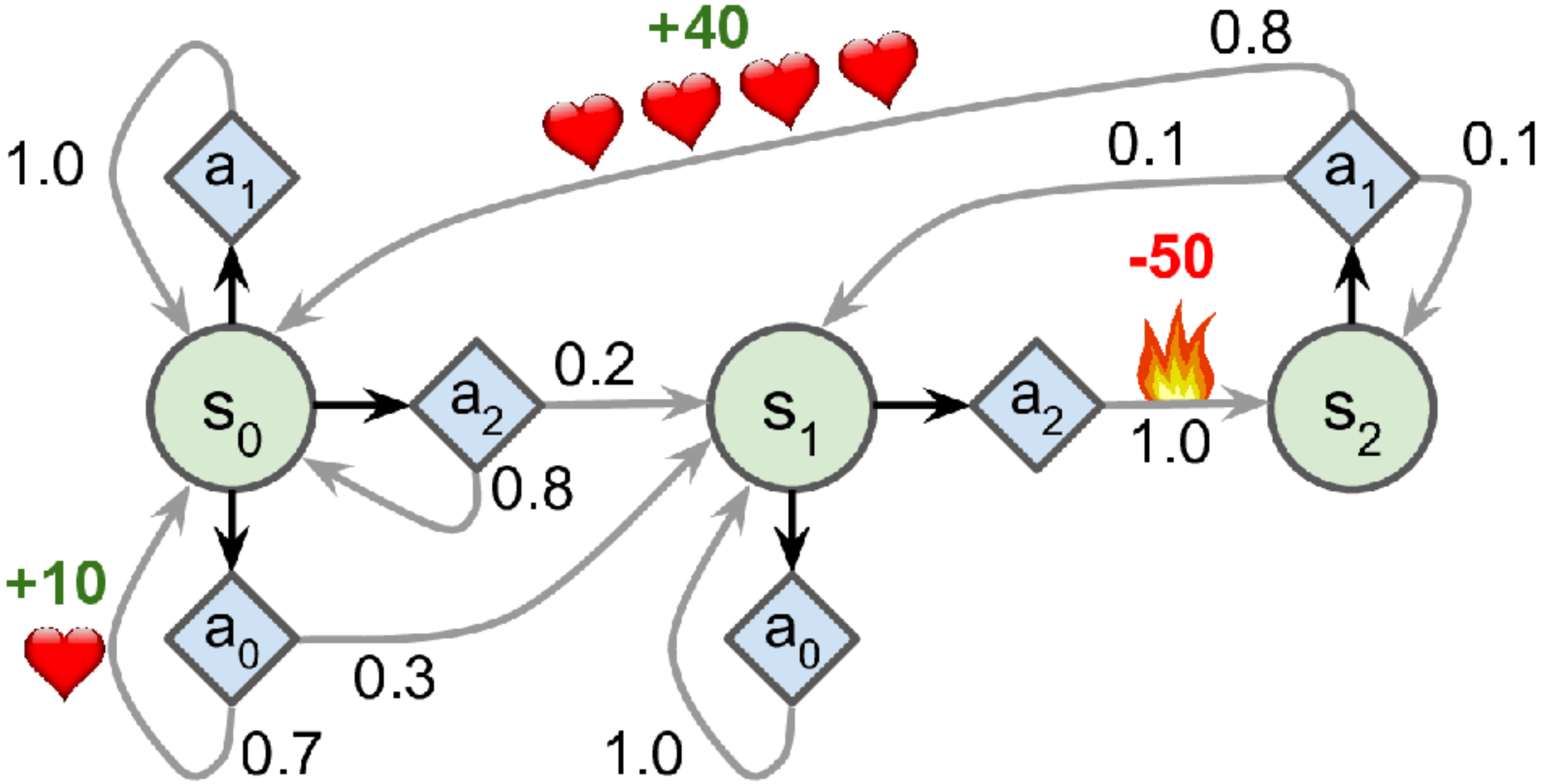
$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma . V_k(s')] \quad \text{for all } s$$

a0: 0.7*[+10 + 0.9*0] + 0.3*[0 + 0.9*0] = 7

a1: 1.0*[0 + 0.9*0] = 0

a2: 0.8*[0 + 0.9*0] + 0.2[0 + 0.9*0] = 0





那要怎麼知道在S0時做哪個action會有最大期望Reward?

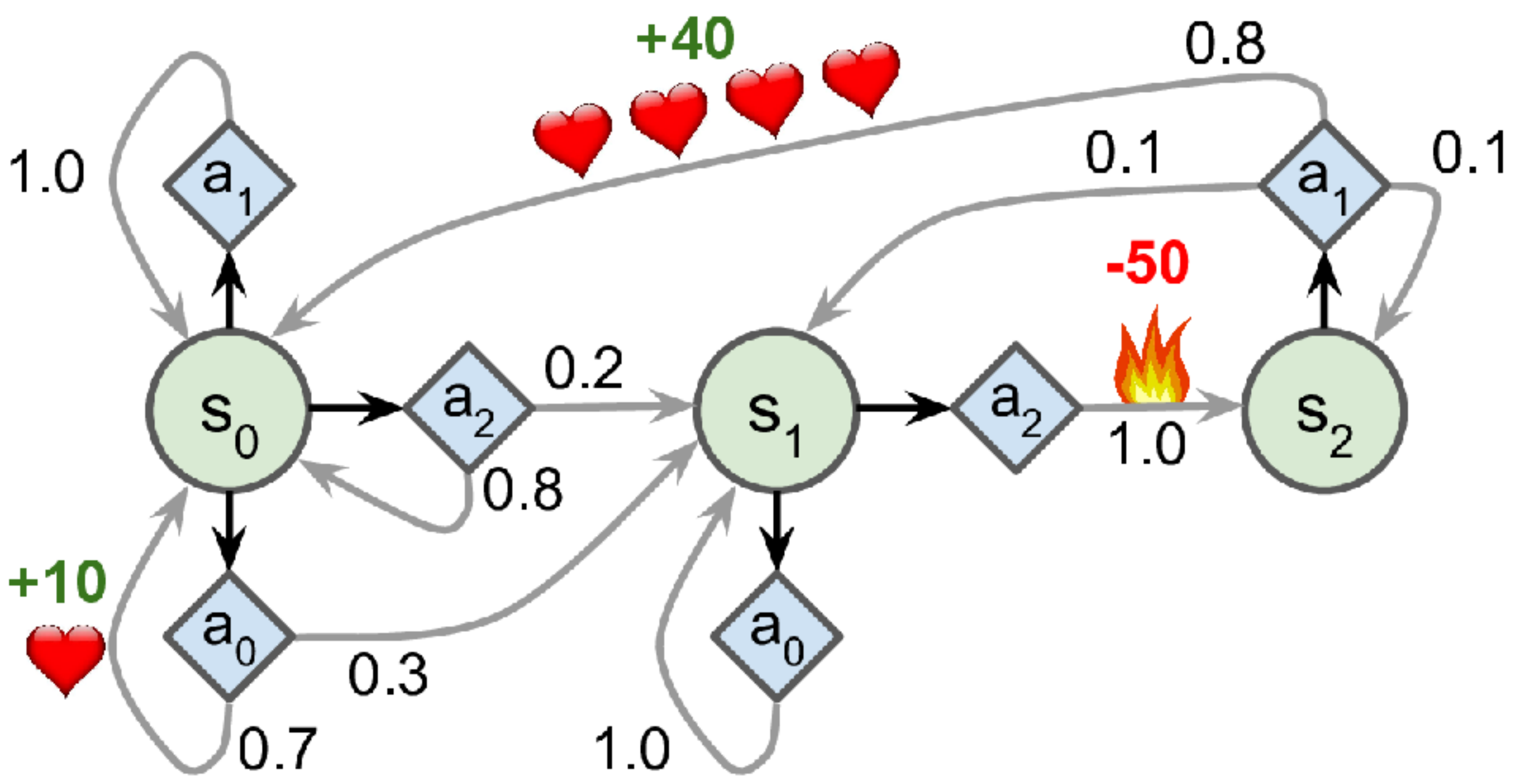
Value Iteration algorithm

$$V_{k+1}(s) \leftarrow \max_{a} \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma . V_k(s')] \quad \text{for all } s$$

Q-Value Iteration algorithm

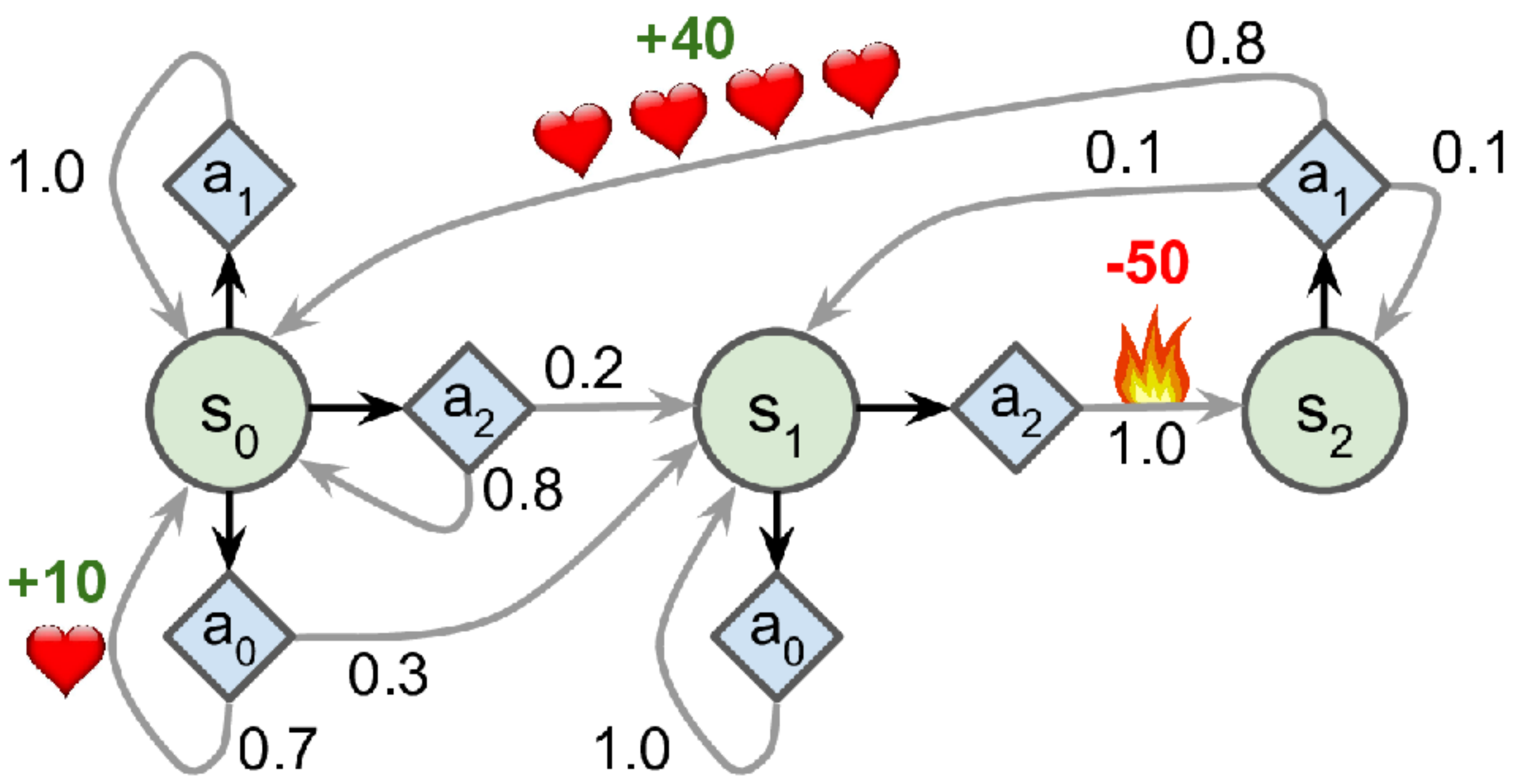
$$Q_{k+1}(s, a) \leftarrow \sum_{s'} T(s, a, s')\left[R(s, a, s') + \gamma . \max_{a'} Q_k(s', a')\right] \quad \text{for all } (s, a)$$

衡量每個狀態執行各*action*的期望價值




```python
STATE_NUM = 3

nan = np.nan

T = np.array([  # shape=[s, a, s']
        [[0.7, 0.3, 0.0], [1.0, 0.0, 0.0], [0.8, 0.2, 0.0]],
        [[0.0, 1.0, 0.0], [nan, nan, nan], [0.0, 0.0, 1.0]],
        [[nan, nan, nan], [0.8, 0.1, 0.1], [nan, nan, nan]],
    ])

R = np.array([  # shape=[s, a, s']
        [[10., 0.0, 0.0], [0.0, 0.0, 0.0], [0.0, 0.0, 0.0]],
        [[0.0, 0.0, 0.0], [nan, nan, nan], [0.0, 0.0, -50.]],
        [[nan, nan, nan], [40., 0.0, 0.0], [nan, nan, nan]],
    ])

possible_actions = [[0, 1, 2], [0, 2], [1]]
```

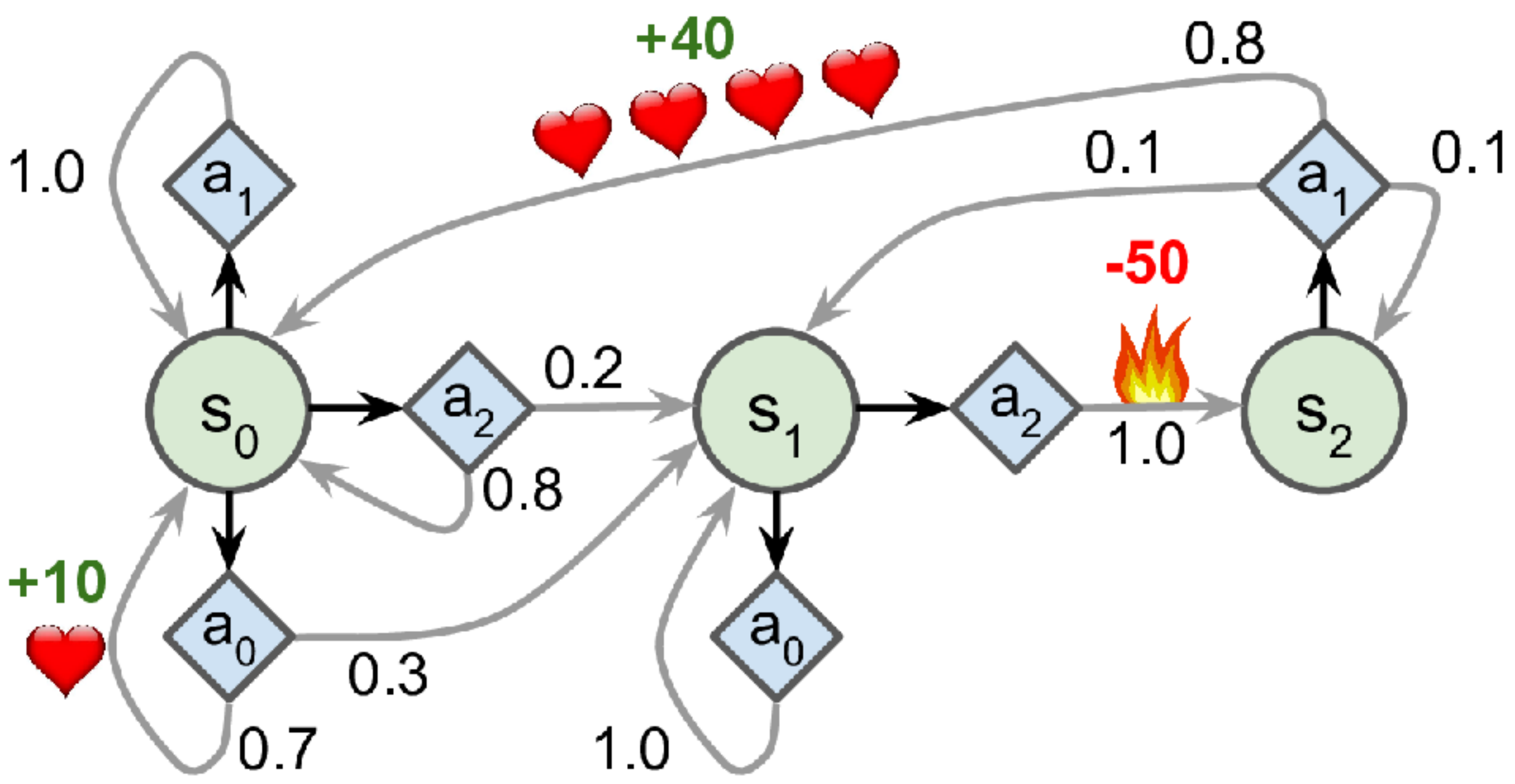
Environment



Init Q Matrix

```
Q = init_Q(STATE_NUM)
Q
```

```
array([[   0.,    0.,    0.],
       [   0., -inf,    0.],
       [-inf,    0., -inf]])
```



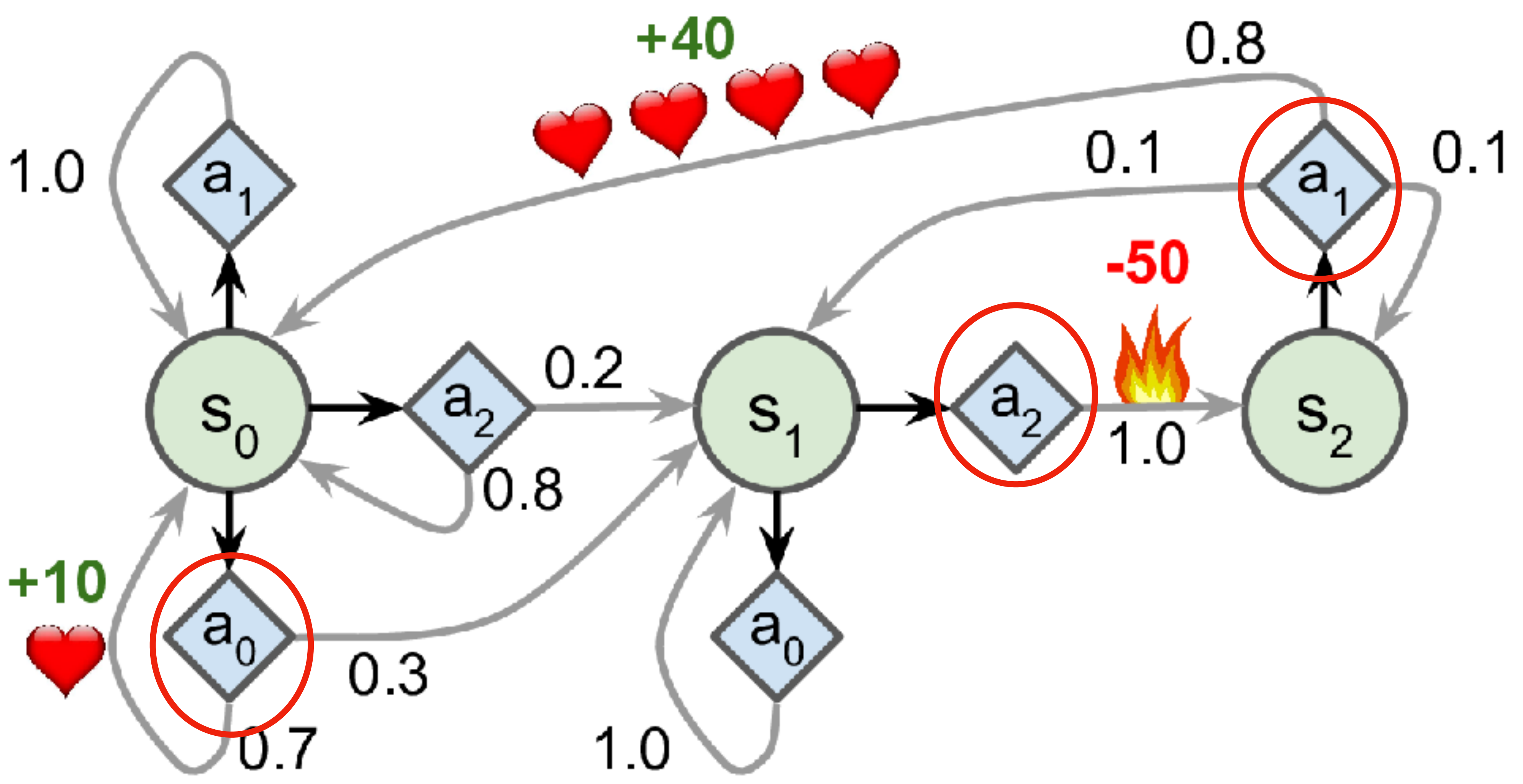
Q Iteration

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma . \max_{a'} Q_k(s',a')\right] \quad \text{for all } (s,a)$$

```
discount_rate = 0.95
n_iterations = 20000
```

```
for i in range(n_iterations):
    Q_prev = Q.copy()
    for s in range(STATE_NUM):
        for a in possible_actions[s]:
            Q[s, a] = np.sum([
                T[s, a, sp] * \
                (R[s, a, sp] + discount_rate * np.max(Q_prev[sp]))
                for sp in range(3)
            ])
print(Q)
```

```
[[ 21.89925005  20.80428755  16.86759588]
 [  1.12082922         -inf   1.17982024]
 [         -inf  53.87349498         -inf]]
```




```python
discount_rate = 0.95
n_iterations = 20000
```

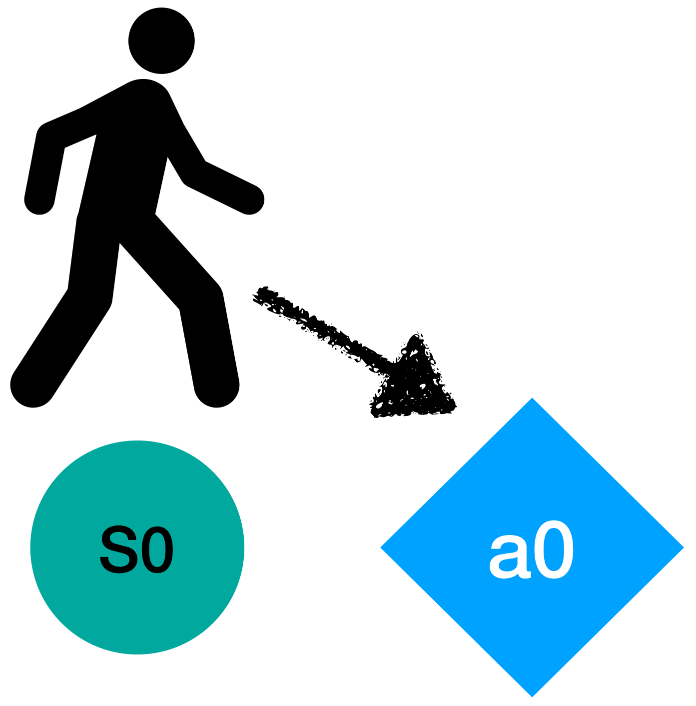
```python
for i in range(n_iterations):
    Q_prev = Q.copy()
    for s in range(STATE_NUM):
        for a in possible_actions[s]:
            Q[s, a] = np.sum([
                T[s, a, sp] * \
                (R[s, a, sp] + discount_rate * np.max(Q_prev[sp]))
                for sp in range(3)
            ])
print(Q)
```

```
[[ 21.89925005  20.80428755  16.86759588]
 [  1.12082922        -inf   1.17982024]
 [        -inf  53.87349498        -inf]]
```

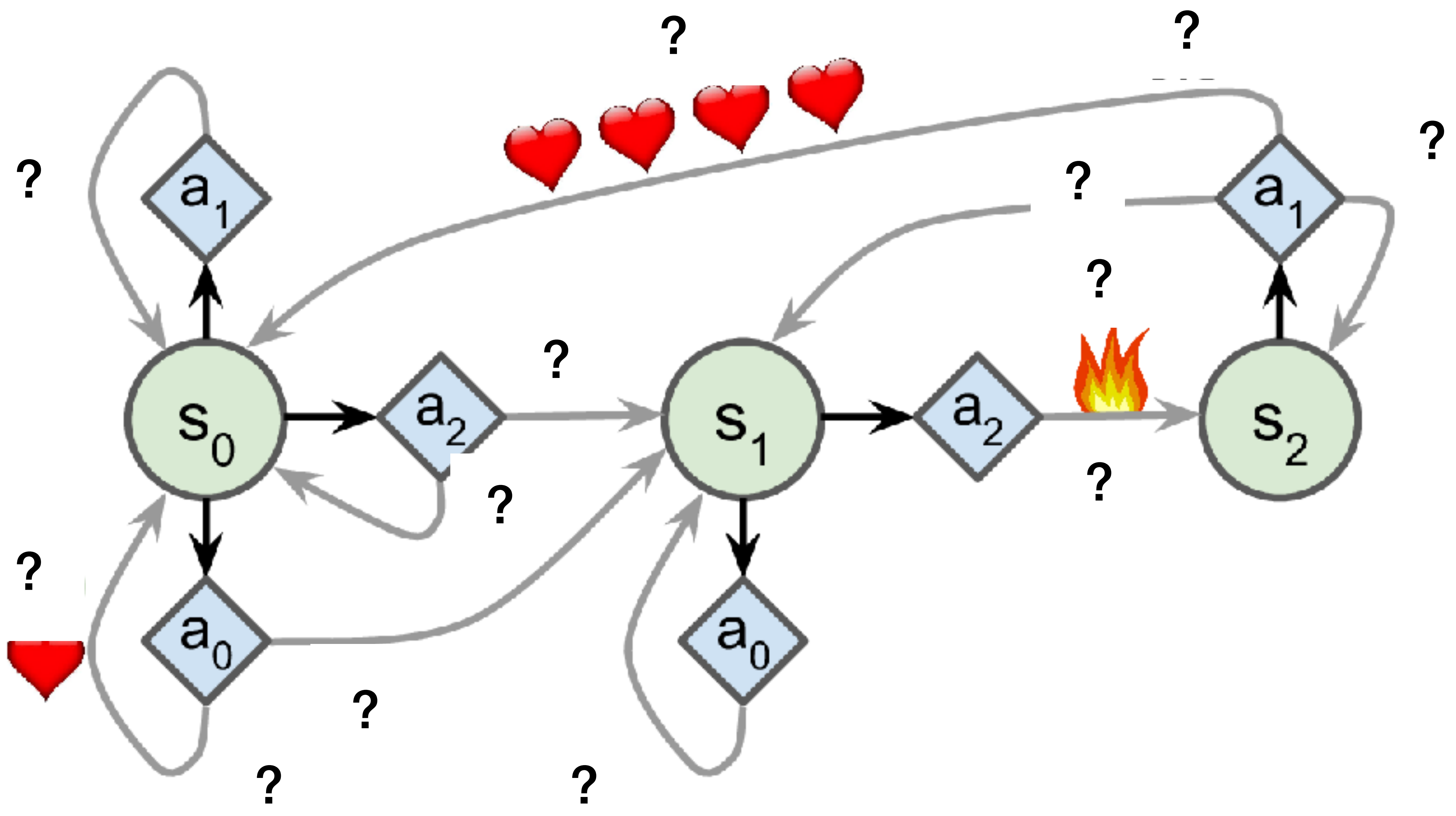
Optimal action for each states

```python
np.argmax(Q, axis=1)
```

```
array([0, 2, 1])
```



現實世界可能更侷限




```
STATE_NUM = 3

nan = np.nan

T = np.array([  # shape=[s, a, s']
        [redacted] initial unknown ???
    ])

R = np.array([  # shape=[s, a, s']
        [redacted] initial unknown ???
    ])

possible_actions = [[0, 1, 2], [0, 2], [1]]
```

Q-Value Iteration algorithm

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma . \max_{a'} Q_k(s',a')\right] \quad \text{for all } (s,a)$$

Q-Learning algorithm

$$Q_{k+1}(s,a) \leftarrow \underbrace{(1-\alpha)Q_k(s,a)}_{\text{歷史狀態}} + \underbrace{\alpha\left(r + \gamma . \max_{a'} Q_k(s',a')\right)}_{\text{觀察更新}}$$

邊觀察邊訓練

Q-Learning algorithm

$$Q_{k+1}(s, a) \leftarrow \underbrace{(1-\alpha)Q_k(s, a)}_{\text{歷史狀態}} + \underbrace{\alpha\left(r + \gamma.\max_{a'} Q_k(s', a')\right)}_{\text{觀察更新}}$$

α = *0.05 / (1 + iteration * 0.1)*

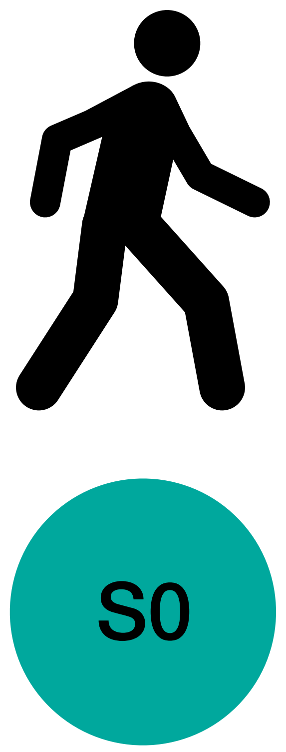
learning rate: 每 1 round會遞減

γ = *0.95*

discount rate

起始 Q Matrix

```
array([[   0.,    0.,    0.],
       [   0., -inf,    0.],
       [-inf,    0., -inf]])
```



Q-Learning algorithm

α = 0.05 / (1 + iteration * 0.1)

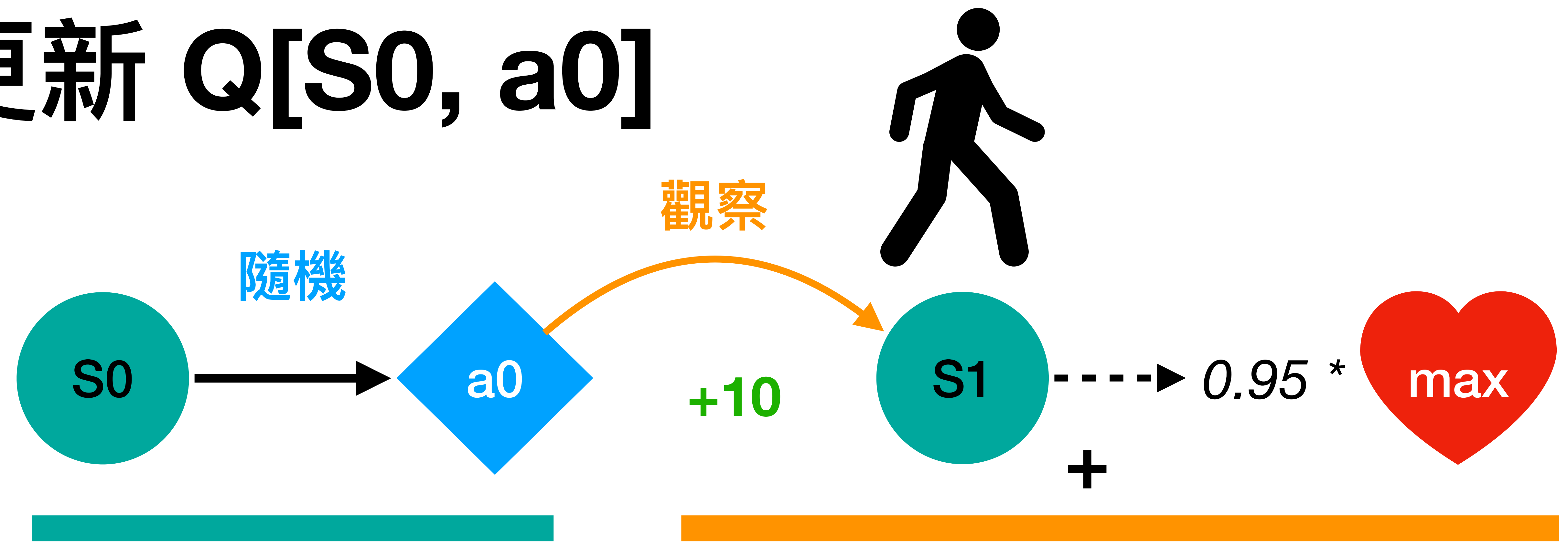
learning rate: 每 1 round會遞減

$$Q_{k+1}(s, a) \leftarrow \underbrace{(1-\alpha)Q_k(s, a)}_{\text{歷史狀態}} + \underbrace{\alpha\left(r + \gamma . \max_{a'} Q_k(s', a')\right)}_{\text{觀察更新}}$$

γ = 0.95

discount rate

更新 Q[S0, a0]



0.95

α = 0.05 / (1 + **0** * 0.1) = 0.05

Q-Learning algorithm

$$Q_{k+1}(s, a) \leftarrow \underbrace{(1-\alpha)Q_k(s, a)}_{\text{歷史狀態}} + \underbrace{\alpha\left(r + \gamma . \max_{a'} Q_k(s', a')\right)}_{\text{觀察更新}}$$

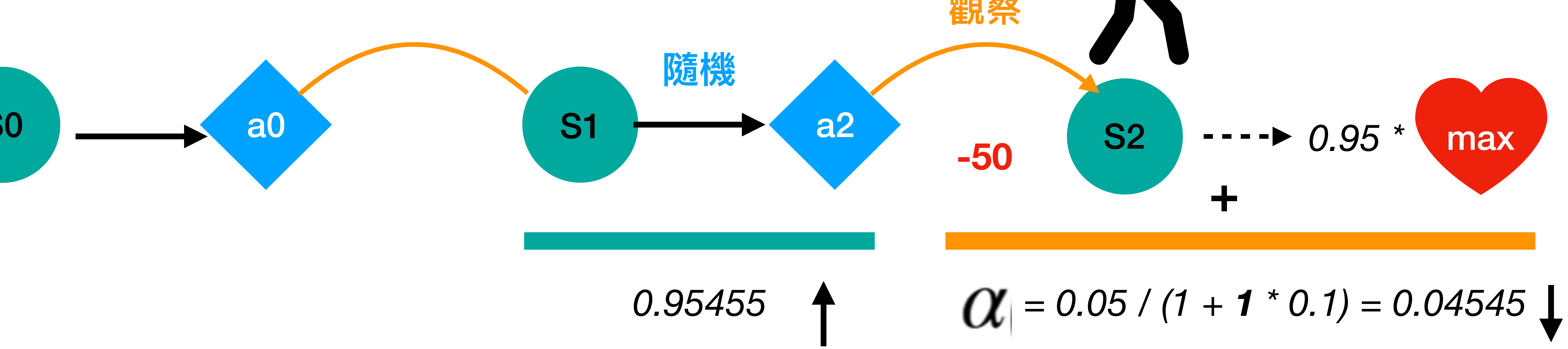
α = 0.05 / (1 + iteration * 0.1)

learning rate: 每 1 round會遞減

γ = 0.95

discount rate

更新 Q[S1, a2]



Q-Learning algorithm

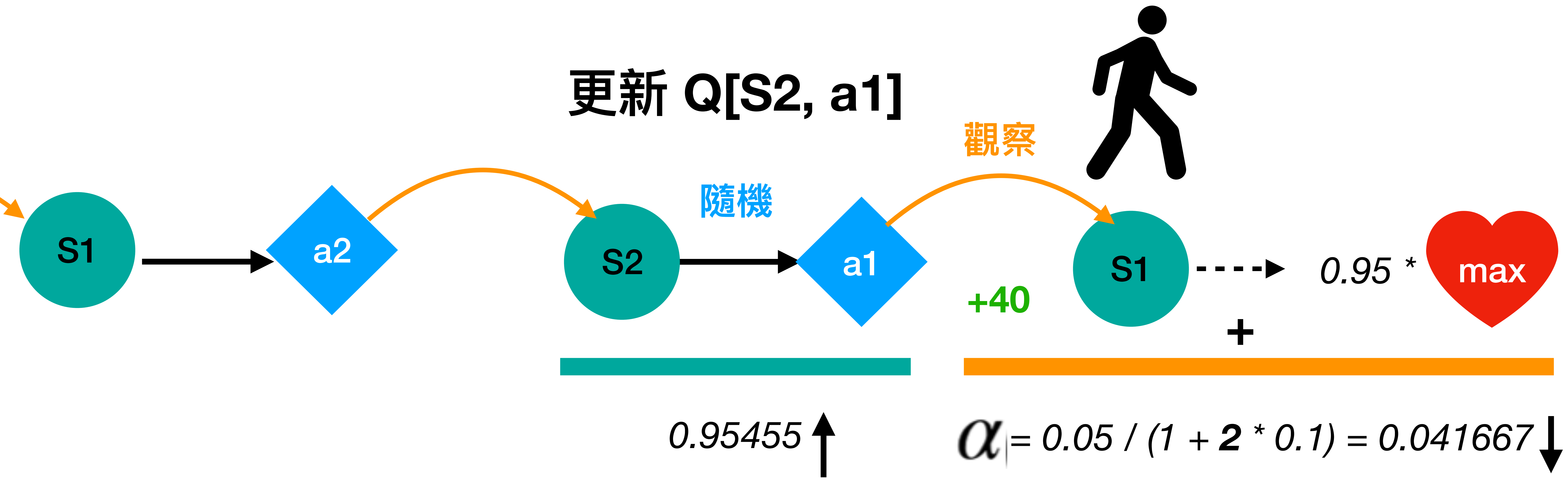
α = 0.05 / (1 + iteration * 0.1)

learning rate: 每 1 round會遞減

$$Q_{k+1}(s, a) \leftarrow \underbrace{(1 - \alpha)Q_k(s, a)}_{\text{歷史狀態}} + \underbrace{\alpha\left(r + \gamma . \max_{a'} Q_k(s', a')\right)}_{\text{觀察更新}}$$

γ = 0.95

discount rate



Q-Learning algorithm

$$Q_{k+1}(s, a) \leftarrow \underbrace{(1 - \alpha)Q_k(s, a)}_{\text{歷史狀態}} + \underbrace{\alpha\left(r + \gamma . \max_{a'} Q_k(s', a')\right)}_{\text{觀察更新}}$$

α = 0.05 / (1 + iteration * 0.1)

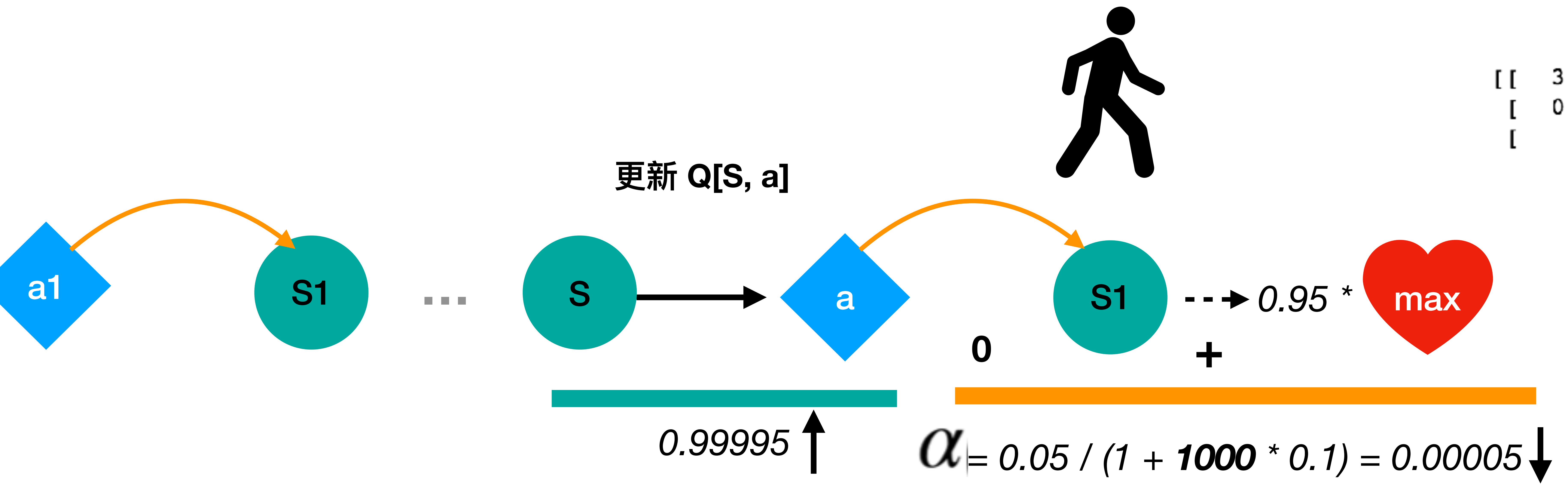
learning rate: 每 1 round會遞減

γ = 0.95

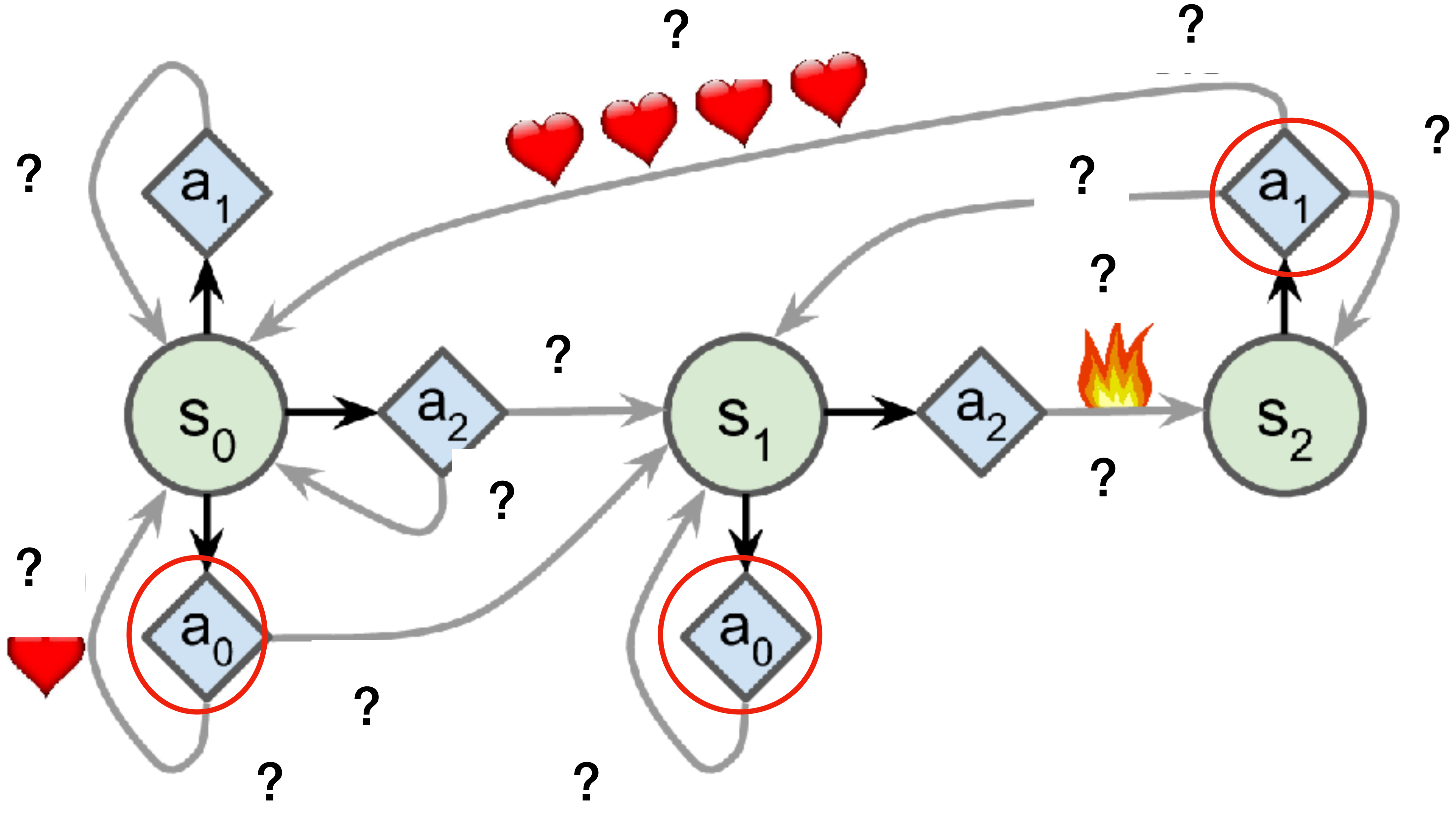
discount rate

收斂 Q Matrix

```
[[  3.79649286   1.26184515   0.70207215]
 [  0.                 -inf -29.846676  ]
 [        -inf  15.63291196        -inf]]
```



$$Q_{k+1}(s, a) \leftarrow (1-\alpha)Q_k(s, a) + \alpha\left(r + \gamma . \max_{a'} Q_k(s', a')\right)$$




```
Q = init_Q(STATE_NUM)
Q
```

```
array([[   0.,    0.,    0.],
       [   0., -inf,    0.],
       [-inf,    0., -inf]])
```

```
s = 0
for i in range(n_iterations):
    a = next_action_by_random(s)
    sp = next_state_by_random(s, a)
    reward = observe_reward(s, a, sp)
    learning_rate = learning_rate0 / (1 + i * learning_rate_decay)
    Q[s, a] = ((1 - learning_rate) * Q[s, a] +
                learning_rate * (reward + discount_rate*np.max(Q[sp])))
    s = sp

print(Q)
```

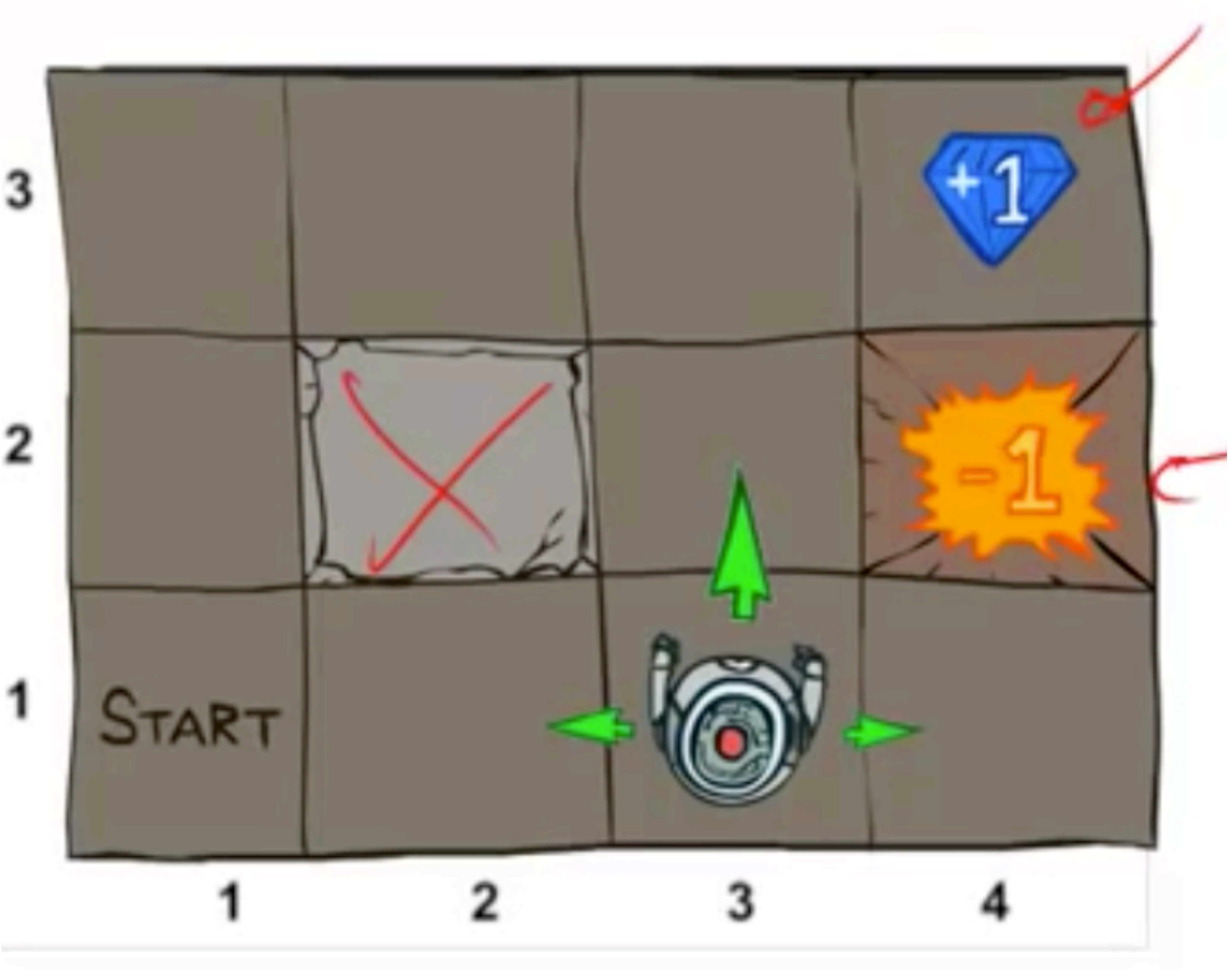
```
[[  2.13388094   0.39599773   0.21791899]
 [  0.                -inf -23.2288997 ]
 [        -inf  10.82558957          -inf]]
```

```
np.argmax(Q, axis=1)
```

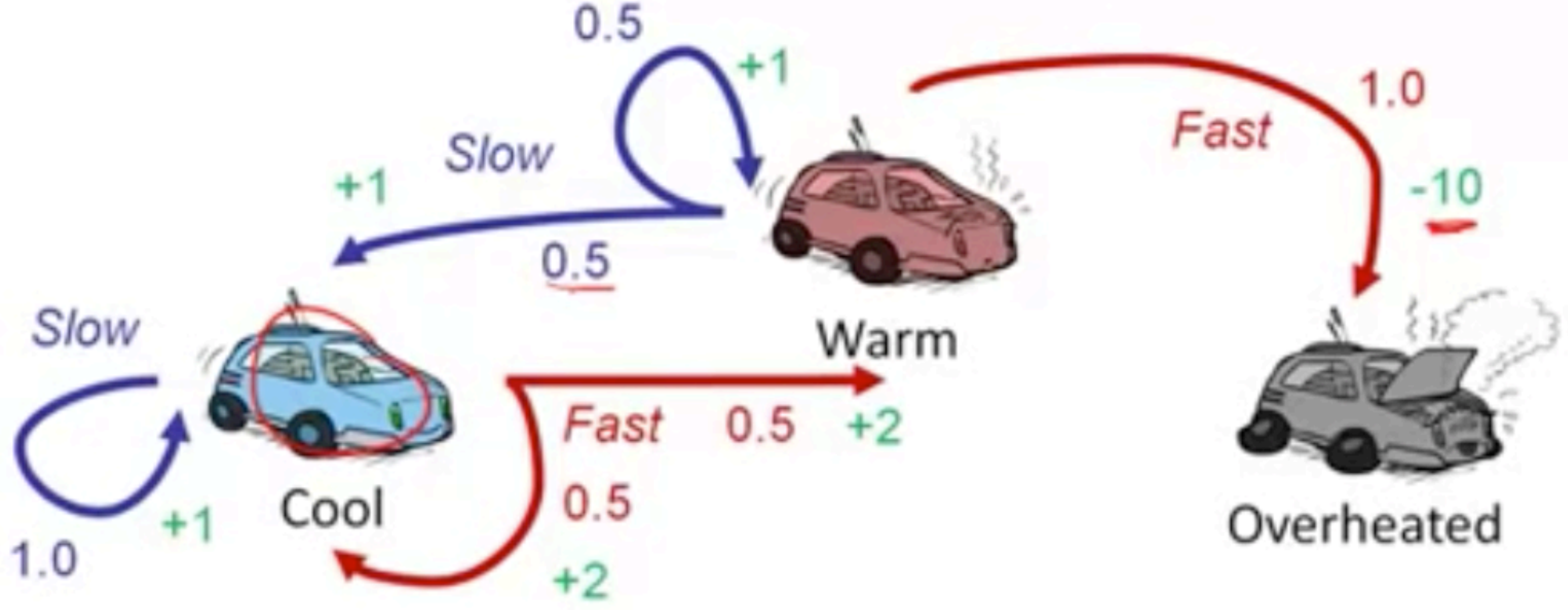
```
array([0, 0, 1])
```

MDP 案例

機器人找寶石

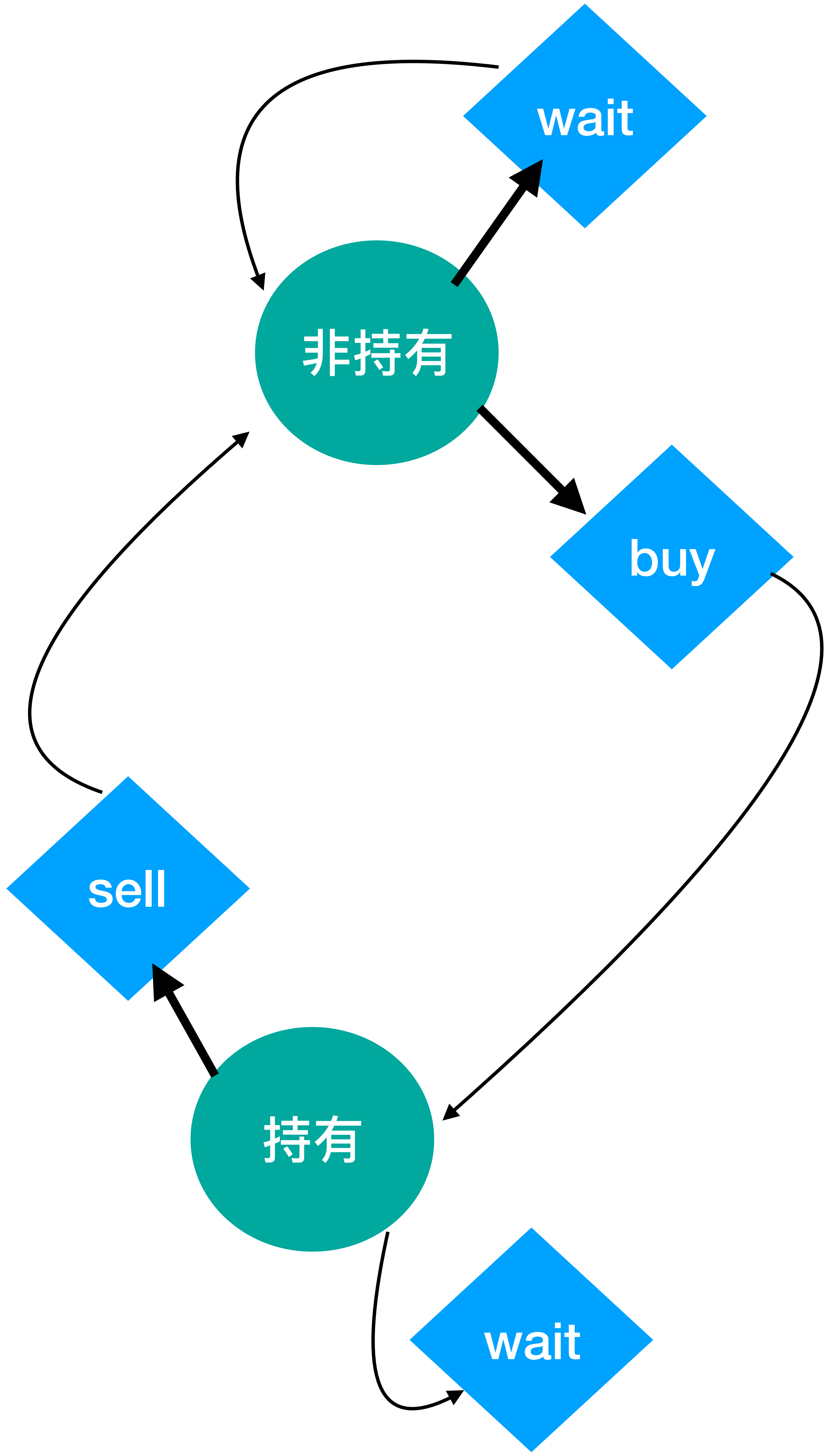


開車



0.5
+1
Slow
+1
0.5
Fast
1.0
-10
Slow
Warm
Cool
Fast
0.5
+2
0.5
+2
1.0
+1
Overheated

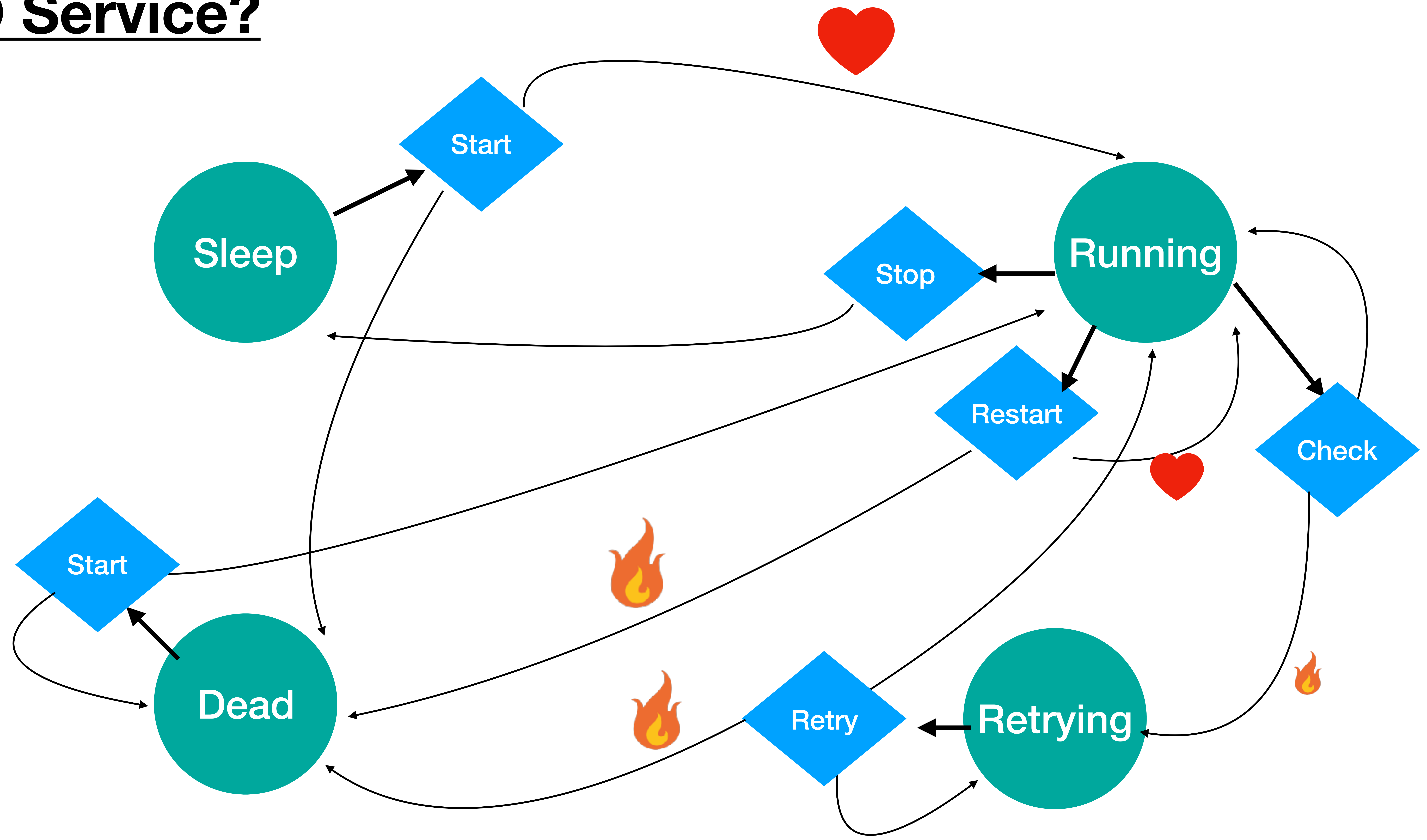
持股策略



wait
非持有
buy
sell
持有
wait

Apple Inc. (AAPL-USA)
369.91 -6.27 -1.67% 08:17:16 AM
12/5/1980 - 8/25/2011
+100
-10

HIPPO Service?



思考: 客戶歷程可以這樣玩？

Reference

- CS188 Artificial Intelligence - MDPs, UC Berkeley
- Hands-on-machine-learning, ch16 Reinforcement Learning