COMP 3200
Artificial Intelligence

Lecture 18
Continuing Tasks
Markov Decision Processes
Dynamic Programming

Textbook References

- Markov Decision Process
  - Chapter 3
- Continuing Tasks
  - Chapter 3.4
- Dynamic Programming
  - Chapter 4

Markov Decision Process

The Markov Property

- In the RL framework, agent makes decisions as a function of the state of an environment
- Recall: the ‘state’ of an environment is whatever info is available to the agent
- If a state contains all relevant information for making a decision, it is said to be Markov, or to have the Markov Property
- Intuitively: “no state history required”

Markov Examples


TSM
TL
3
3
37.7k
26:04
TL PigJet
13
0:11
3690
TSM Santorin
11
TSM Dyrus
13
TSM WildTurtle
13
TSM Bjergsen
14
TECH
INSIDER


REPLAY
TL Piglet
13
TSM WildTurtle
13
TSM Bjergsen
14
4
12
9

Markov Property Definition

- In general, environment state and rewards may be dependent on complete action sequence
- $Pr\{s_{t+1}=s', r_{t+1}=r \mid s_t, a_t, r_t, s_{t-1}, a_{t-1}, \ldots, r_1, s_0, a_0\}$

- If Markov Property holds, environment response at t+1 depends only on state and action at t
- $Pr\{s_{t+1}=s', r_{t+1}=r \mid s_t, a_t\}$

Markov Decision Process (MDP)

- An RL task that satisfies the Markov Property is called a Markov Decision Process (MDP)
- If state and action spaces are finite, it is called a finite MDP
- Partially Observable Environment: POMDP

MDP Definition

- MDP = (S, A, P, R, γ)
  - S = finite set of states
  - A = finite set of actions (A_s legal from s)
  - $P_a(s,s') = Pr(s_{t+1}=s' \mid s_t=s, a_t=a)$
  - $R_a(s,s')$ = reward after transition s to s' via a
  - $\gamma \in [0,1]$ = discount factor

MDP Problem / Objective

- Compute a policy that π maximizes the sum of future discounted rewards (Return)

$$\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Discounting: γ

Discounting

- Agent tries to select actions that the sum of discounted rewards is maximized
- Return $G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots$

$$\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

- $0 \leq \gamma \leq 1$ is the Discount Rate

Discounting

$$\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

- The discount rate determines the present value of future rewards
- A reward you get k time steps in the future is worth only γ^{k-1} as if immediate
- If $\gamma = 0$, agent cares about next reward
- If $\gamma = 1$, reward formula same as episodic
  - As γ approaches 1, agent becomes farsighted

Related Returns

- Returns at successive time steps are related in mathematically important way

$$
\begin{aligned}
G_t &\doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \\
&= R_{t+1} + \gamma \left(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \cdots \right) \\
&= R_{t+1} + \gamma G_{t+1}
\end{aligned}
$$

Episodic vs. Continuing Tasks

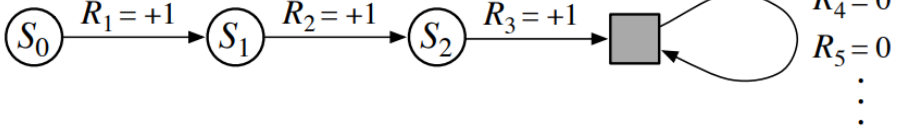
- Recall: Episodic Tasks
- Sequence of actions, terminating state
- Episodes considered independent
- Ex: Play of a game, trip through a maze
- Return $G_t = r_{t+1} + r_{t+2} + \ldots + r_T$
- Object of agent is to determine a policy which maximizes expected return

Continuing Tasks

- Task doesn't break easily into episodes
- Goes on continuously without limit
- Ex: Long lifespan robot, control task
- Episodic return formula doesn't work
  - $G_t = r_{t+1} + r_{t+2} + \ldots + r_T$
  - Final time step T = Infinity, G_T ?= Infinity

Unified Mathematical Notation

- Would be nice to have unified notation for episodic and continuing tasks
- Let's modify episodic tasks to be continuing
- Add a final absorbing state that loops r=0



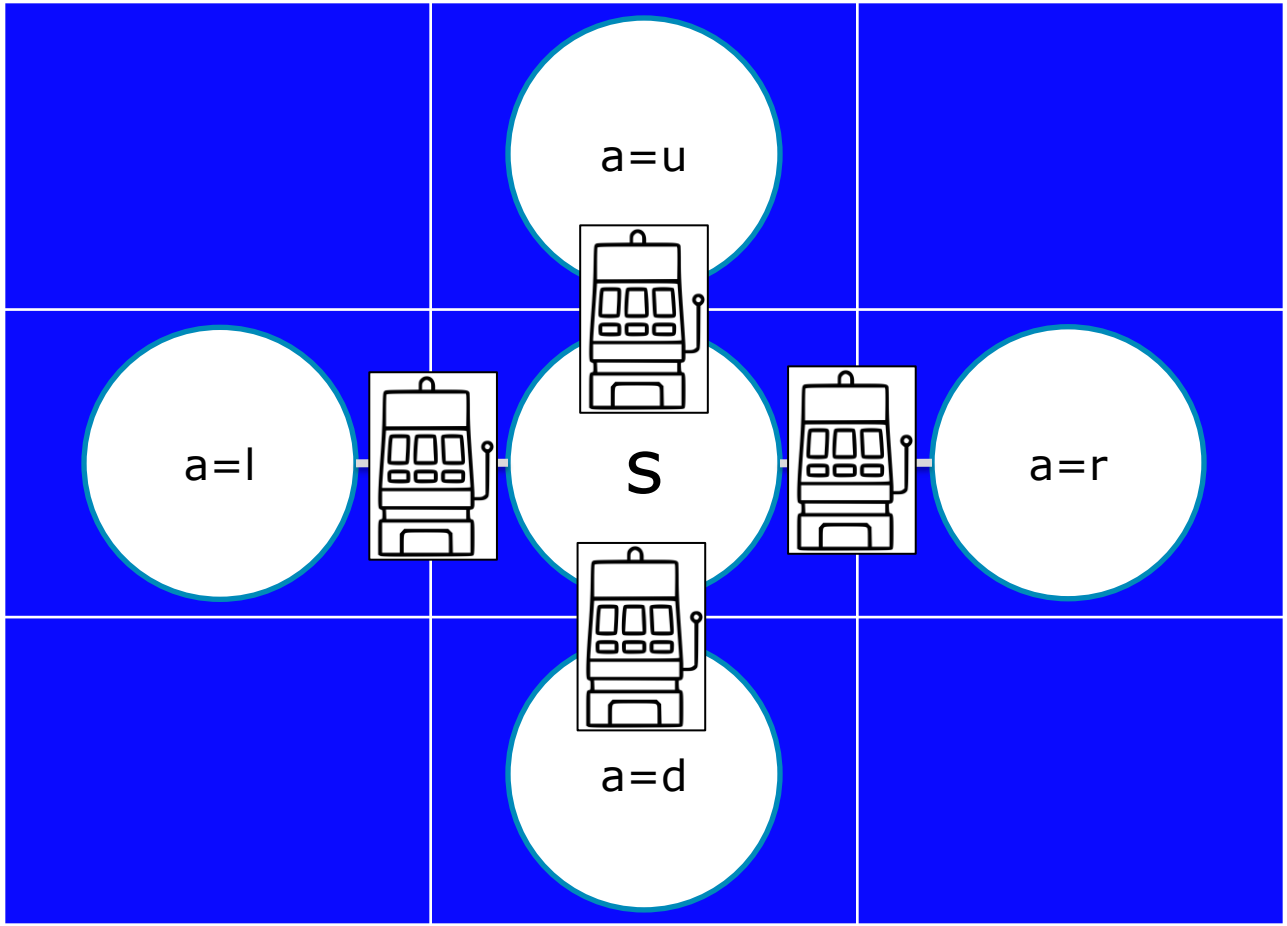
$$\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Action Values

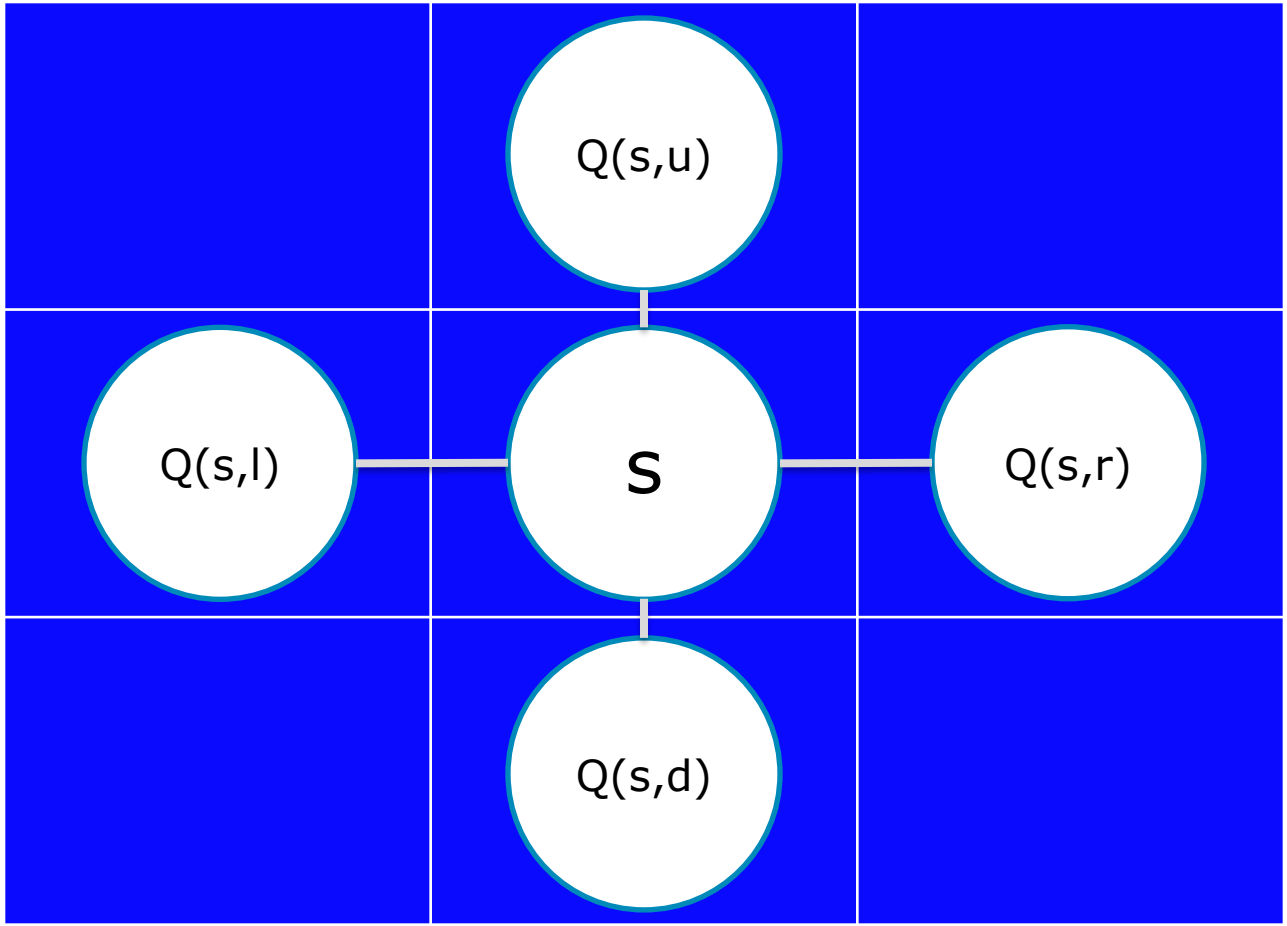
Recall: Action Values Q(a)

- Q(a) = value of doing action **a**
- The value of a specific action will vary depending on the state it was issued
- For example: Moving up is good if the goal is up, but not if the goal is down
- Q(s,a) = value of action **a** at state **s**

Action
N



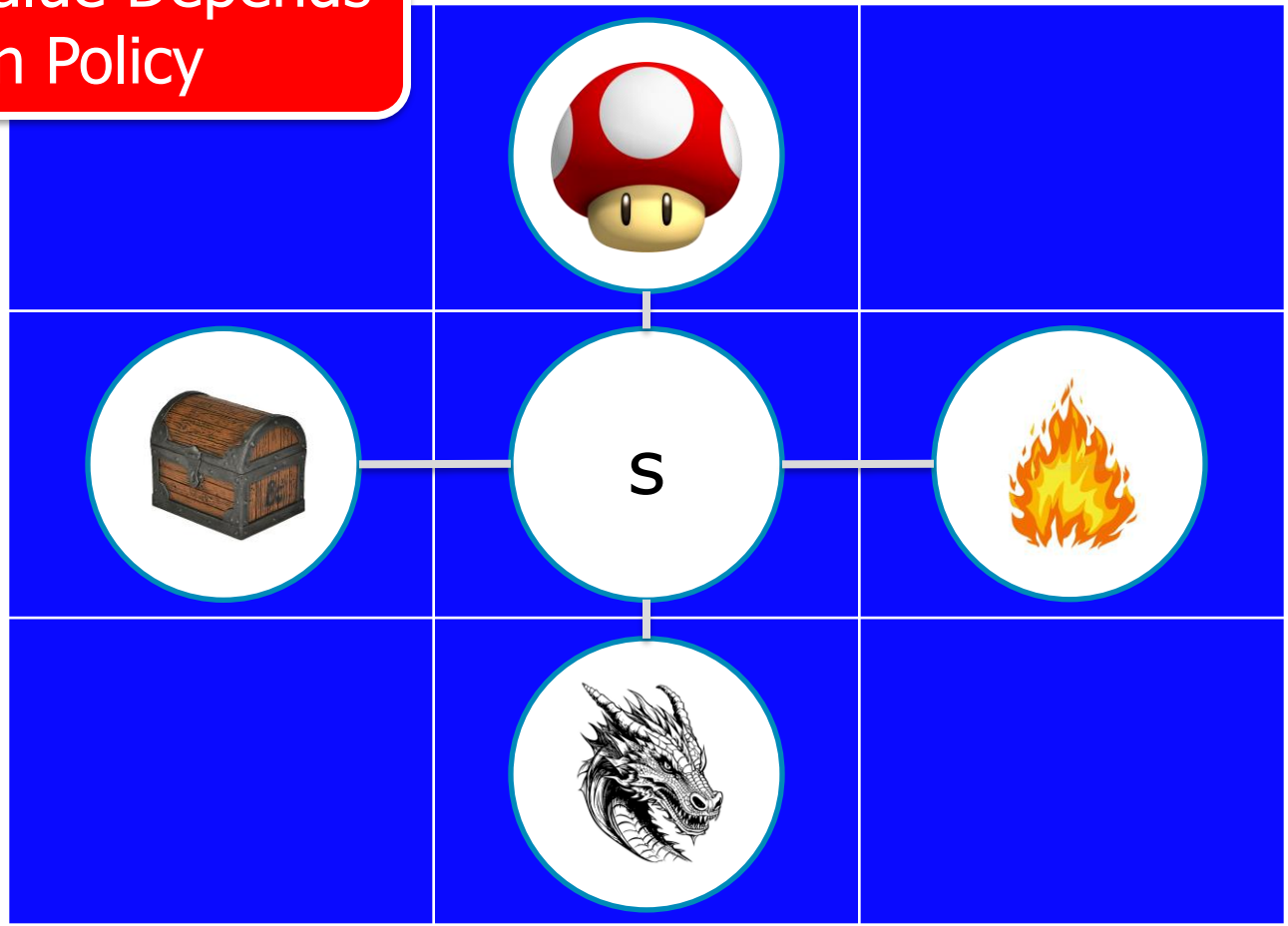
Values
Q(s,a)



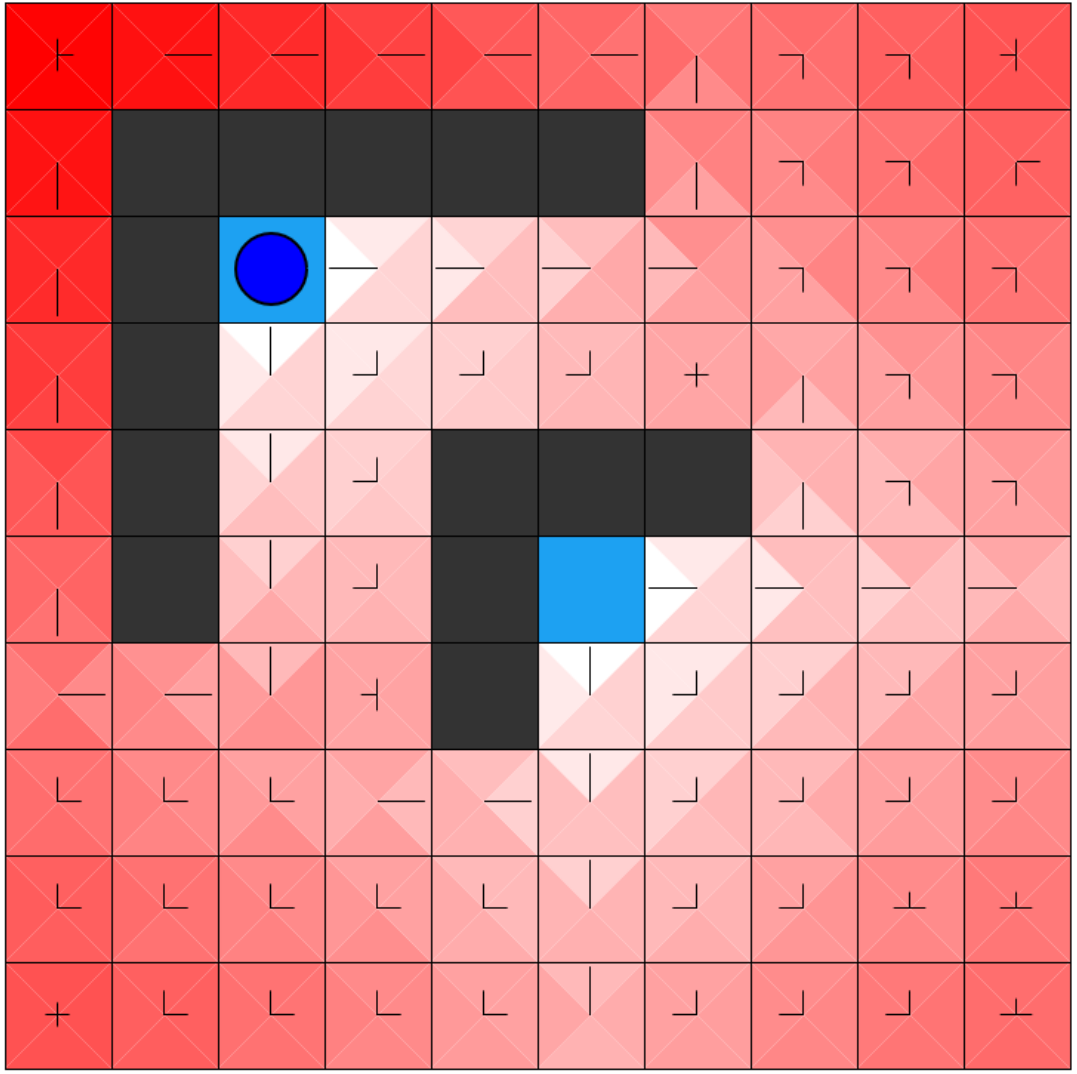
Action Values

- We get rewards by doing actions at states
- A value is an estimate of how much total reward we expect to get in the future
- Rewards depend only on a single action
- Values depend on a policy: we must know what we will do in in the future order to estimate the value of future rewards

State Value Depends on Policy




State Value Depends
on Policy



Q^{π} - Action Value Function

- Overall value only makes sense if we speak in context of a policy: π
- $Q^{\pi}(s,a)$ = Expected Return starting from state s, taking action a, then following π

$$q_\pi(s,a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a]$$

$$= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \;\middle|\; S_t = s, A_t = a\right]$$

V^{π} - State Value Function

- What is the value of being at a given state
- $V^{\pi}(s)$ = Expected Return starting from state s then following policy π

$$v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s]$$

$$= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \;\middle|\; S_t = s\right]$$

MDP Problem / Objective

- Choose a policy that π maximizes Return:

$$\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

- Can be solved by dynamic programming if we have a model of our environment

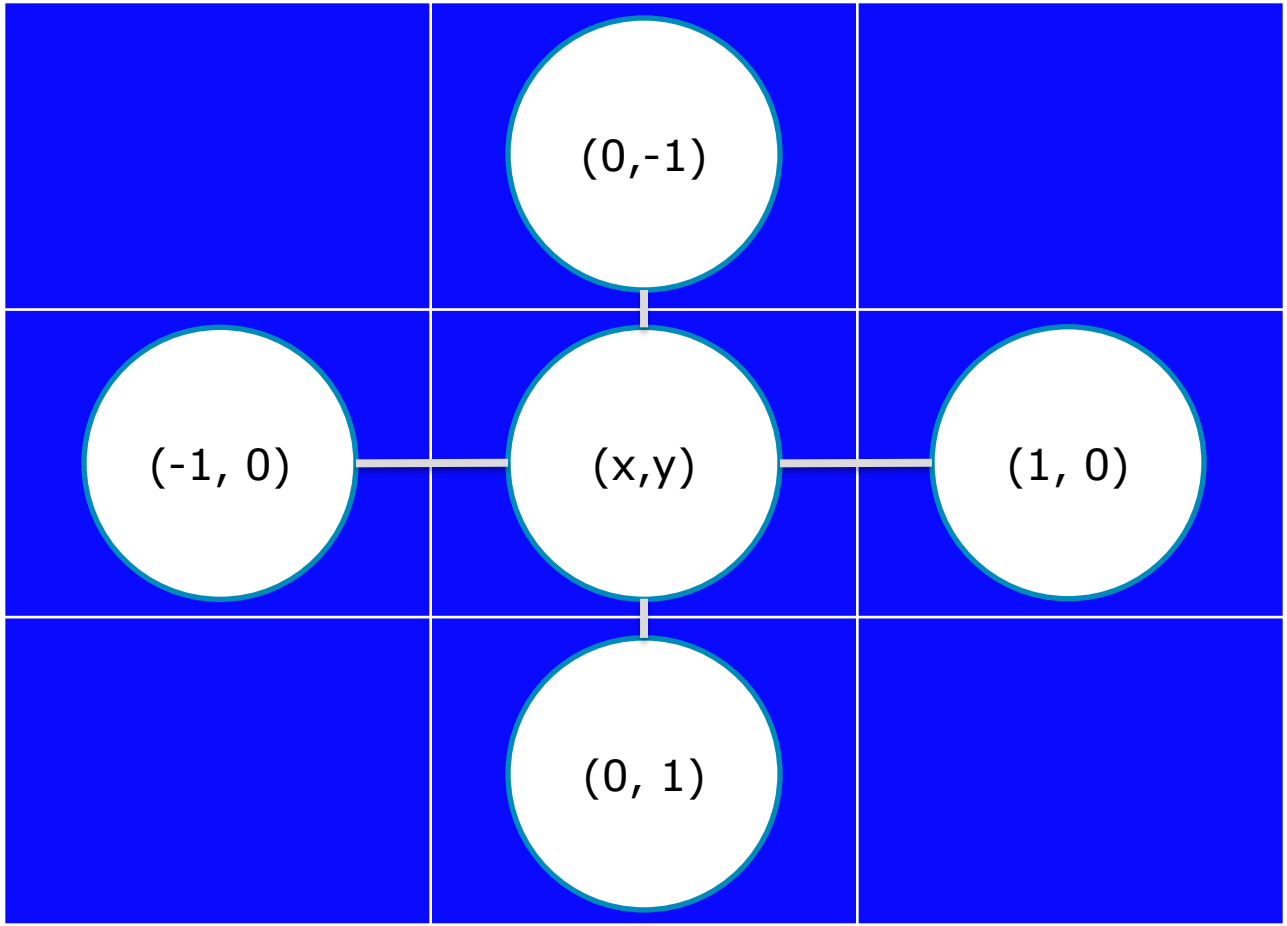
Dynamic Programming

Dynamic Programming

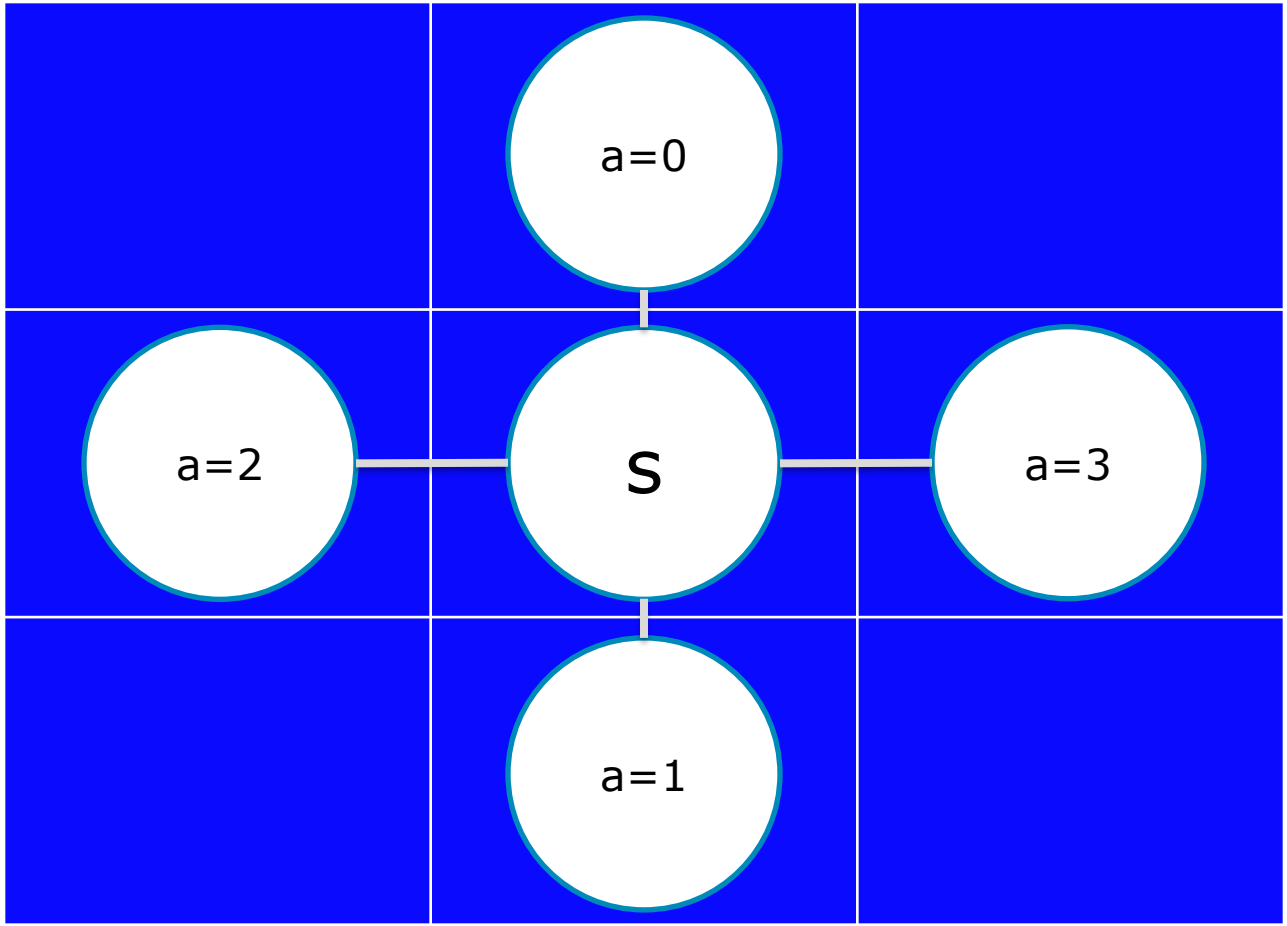
- Collection of algorithms that can compute optimal policies given a perfect model of the environment as an MDP
- DP's main idea: my value is related somehow to the value of my neighbours
- We will use DP to compute the value function, and use the value function to compute the policy

Dynamic Programming Example

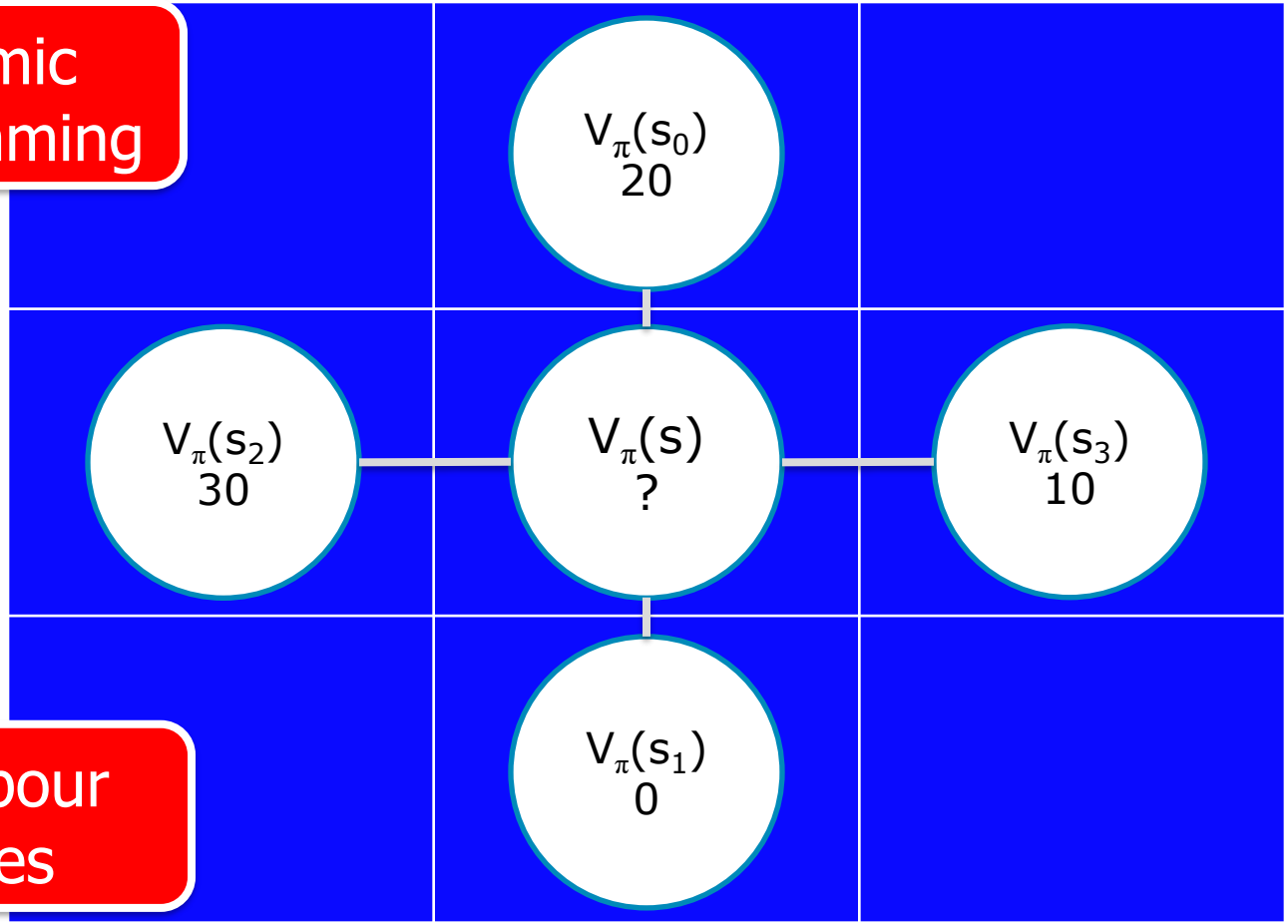
Action [x,y]



Action
N

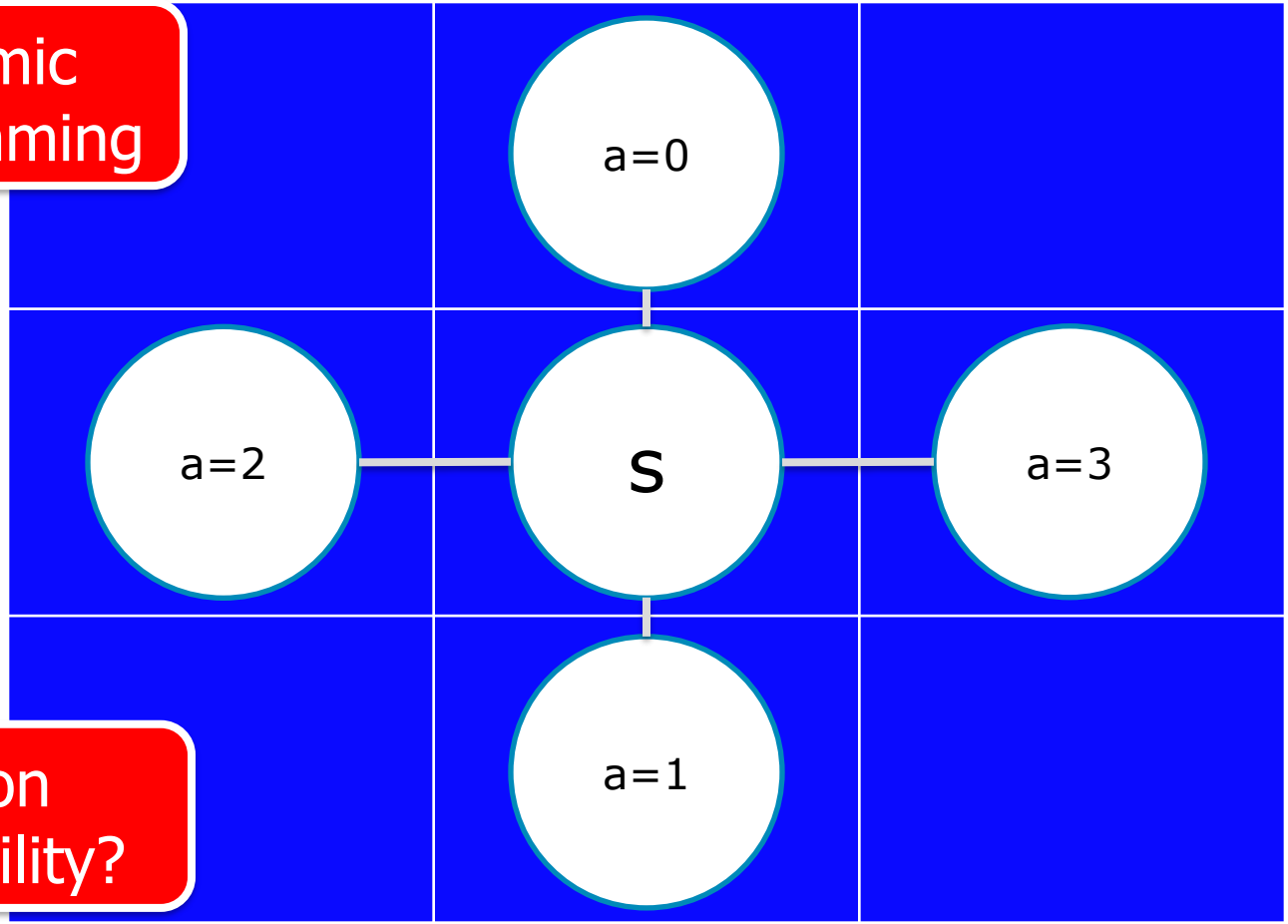


Dynamic Programming



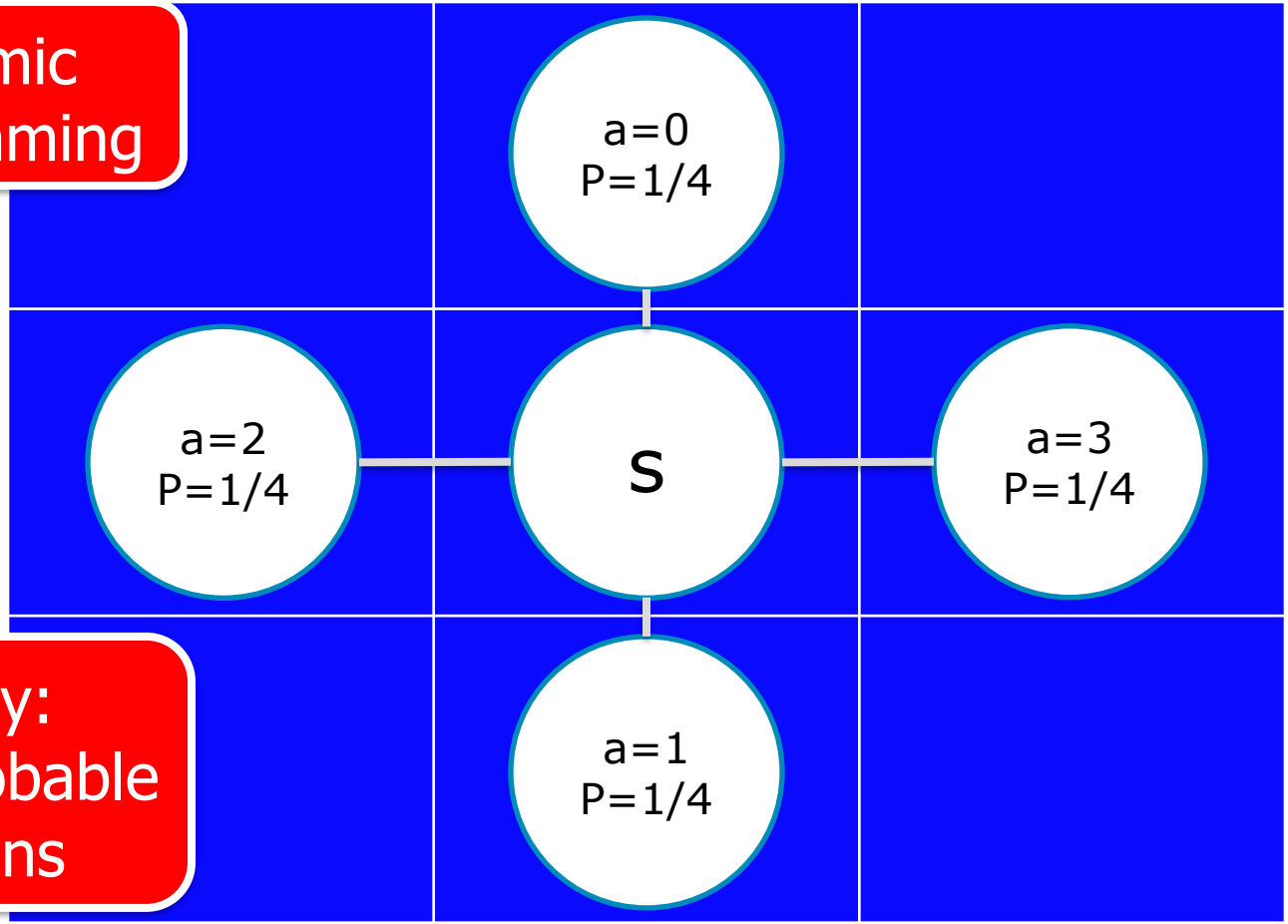
Neighbour Values

Dynamic Programming



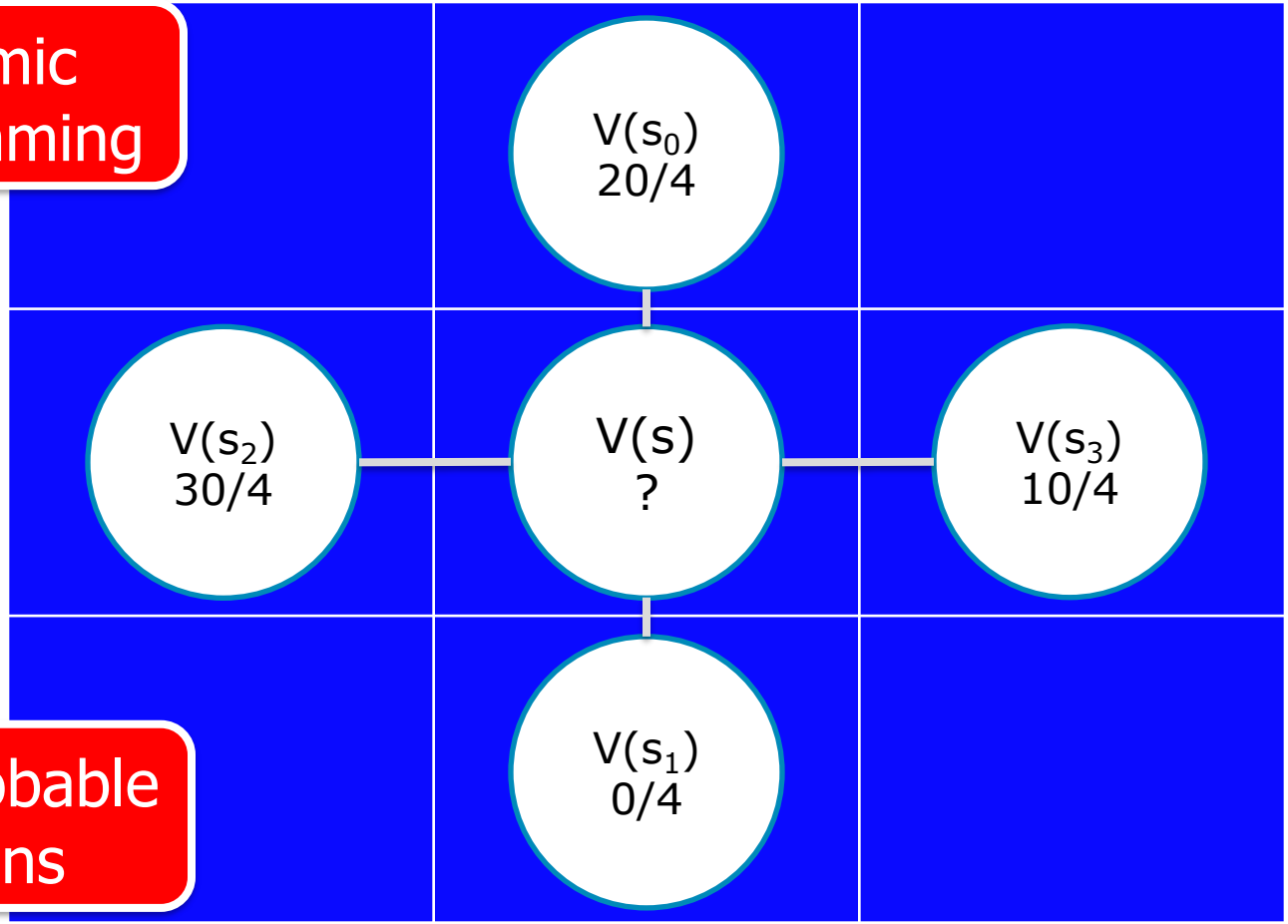
Action Probability?

Dynamic Programming



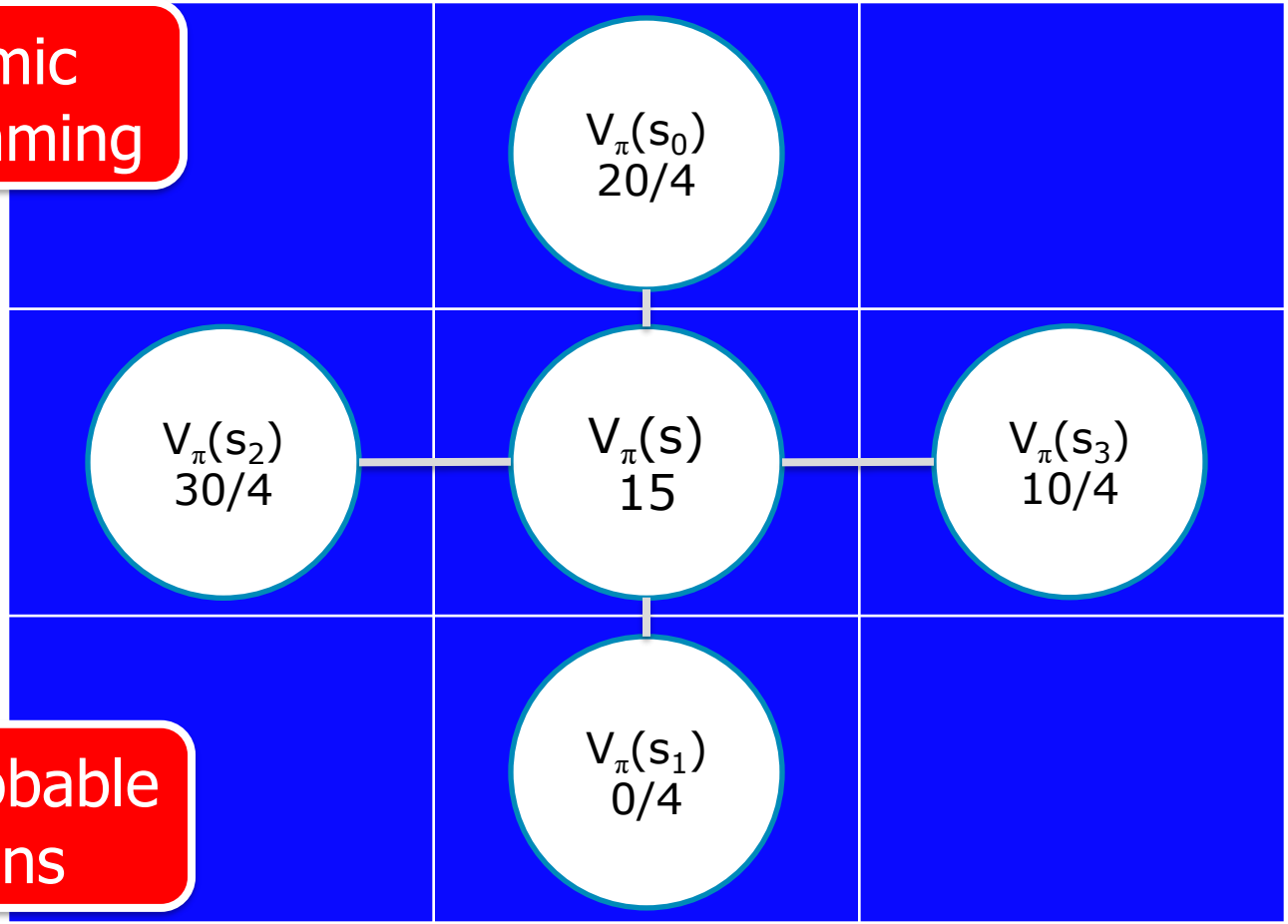
Policy:
Equi-Probable
Actions

Dynamic Programming



Equi-Probable Actions

Dynamic Programming

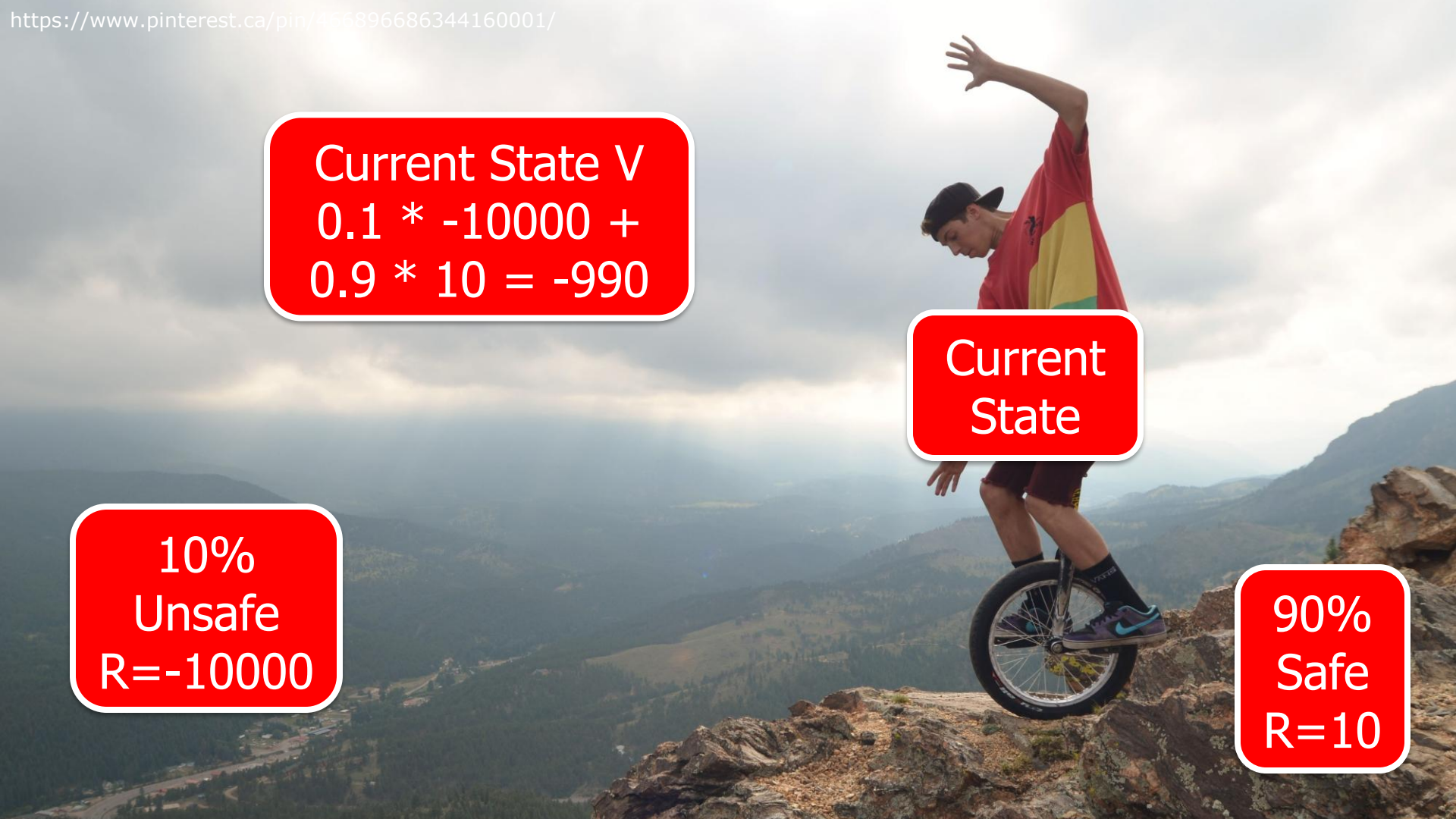


Equi-Probable Actions


Current
State


Current State
90% Safe R=10


Current State
10% Unsafe R=-10000
90% Safe R=10


https://www.pinterest.ca/pin/466896686344160001/
Current State V
0.1 * -10000 +
0.9 * 10 = -990
Current State
10%
Unsafe
R=-10000
90%
Safe
R=10

Bellman Equation

Policy (Value) Evaluation

- Let's compute the state-value function V^{π} for an arbitrary policy π
- We use the Bellman Equation

$$V^{\pi}(s) = \sum_{a} \pi(s,a) \sum_{s'} \mathcal{P}^{a}_{ss'} \left[\mathcal{R}^{a}_{ss'} + \gamma V^{\pi}(s') \right]$$

- $\pi(s,a)$ = Probability of taking a in s under π

Policy Evaluation

$$V^{\pi}(s) = \sum_{a} \pi(s,a) \sum_{s'} \mathcal{P}^{a}_{ss'} \left[\mathcal{R}^{a}_{ss'} + \gamma V^{\pi}(s') \right]$$

Policy Evaluation

$$V^{\pi}(s) = \sum_{a} \pi(s,a) \sum_{s'} \mathcal{P}^{a}_{ss'} \left[\mathcal{R}^{a}_{ss'} + \gamma V^{\pi}(s') \right]$$

- In the case of actions deterministically transition from one state to another:

$$V^{\pi}(s) = \sum_{a} \pi(s,a) \left[\mathcal{R}^{a}_{ss'} + \gamma V^{\pi}(s') \right]$$

Value Update (Code)

$$V^{\pi}(s) = \sum_{a} \pi(s,a) \left[\mathcal{R}^{a}_{ss'} + \gamma V^{\pi}(s') \right]$$

```
1.  v[s] = value of state s
2.  p[s][a] = policy (probability of doing a at s)
3.  for each state s in environment:
4.      v[s] = 0
5.      for each legal action a at s:
6.          pr = probability of choosing a at s = p[s][a]
7.          r = reward from doing action a at s
8.          s' = apply a to s (get next state)
9.          v[s] += pr * (r + gamma * v[s'])
```

Policy Update

- Take the action at s that has highest value

$$Q^{\pi}(s,a) = \sum_{s'} \mathcal{P}^{a}_{ss'} \left[\mathcal{R}^{a}_{ss'} + \gamma V^{\pi}(s') \right]$$

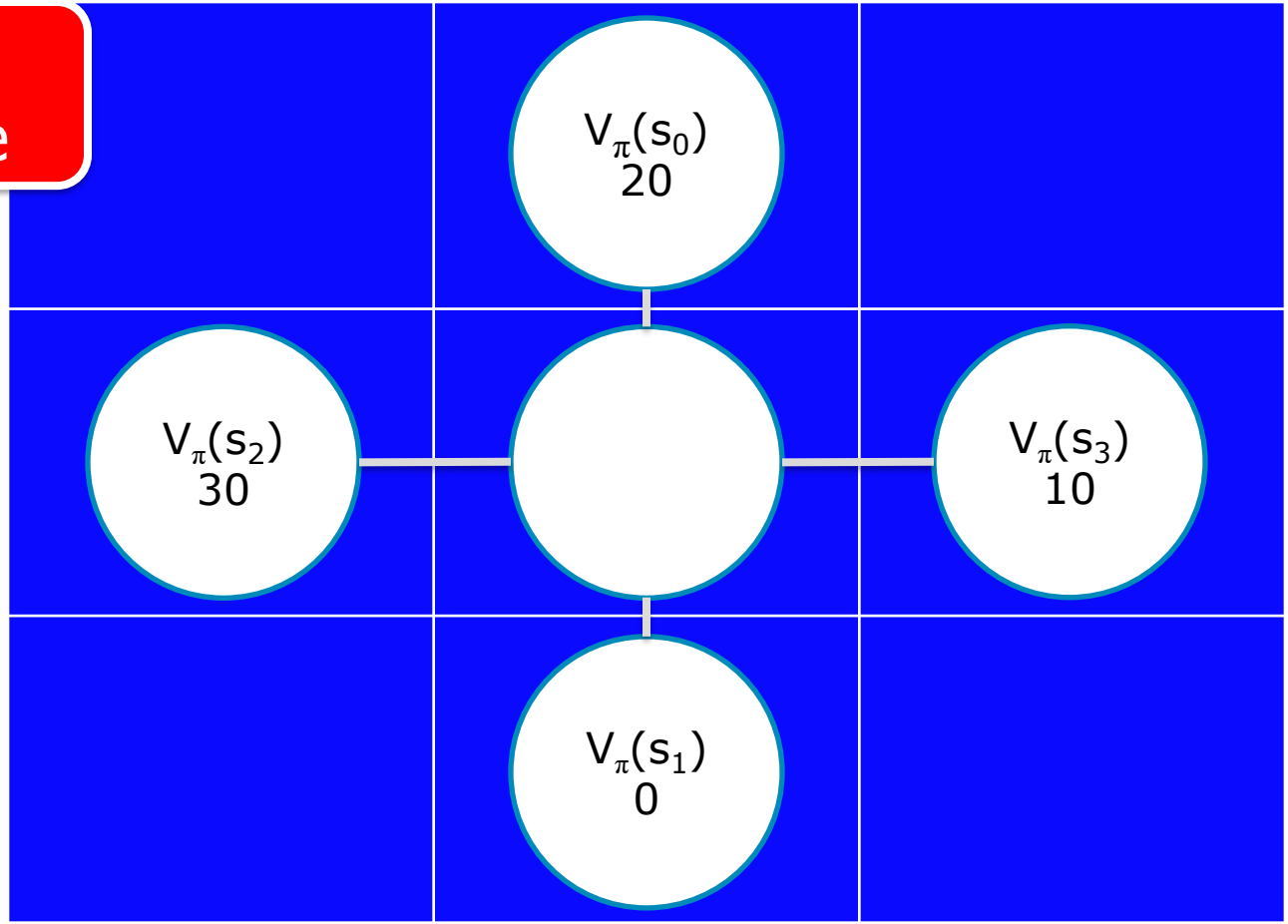
- We just computed this! If we want to know the value of doing action a at state s, look up the value of the resulting v[s’]

Policy Update

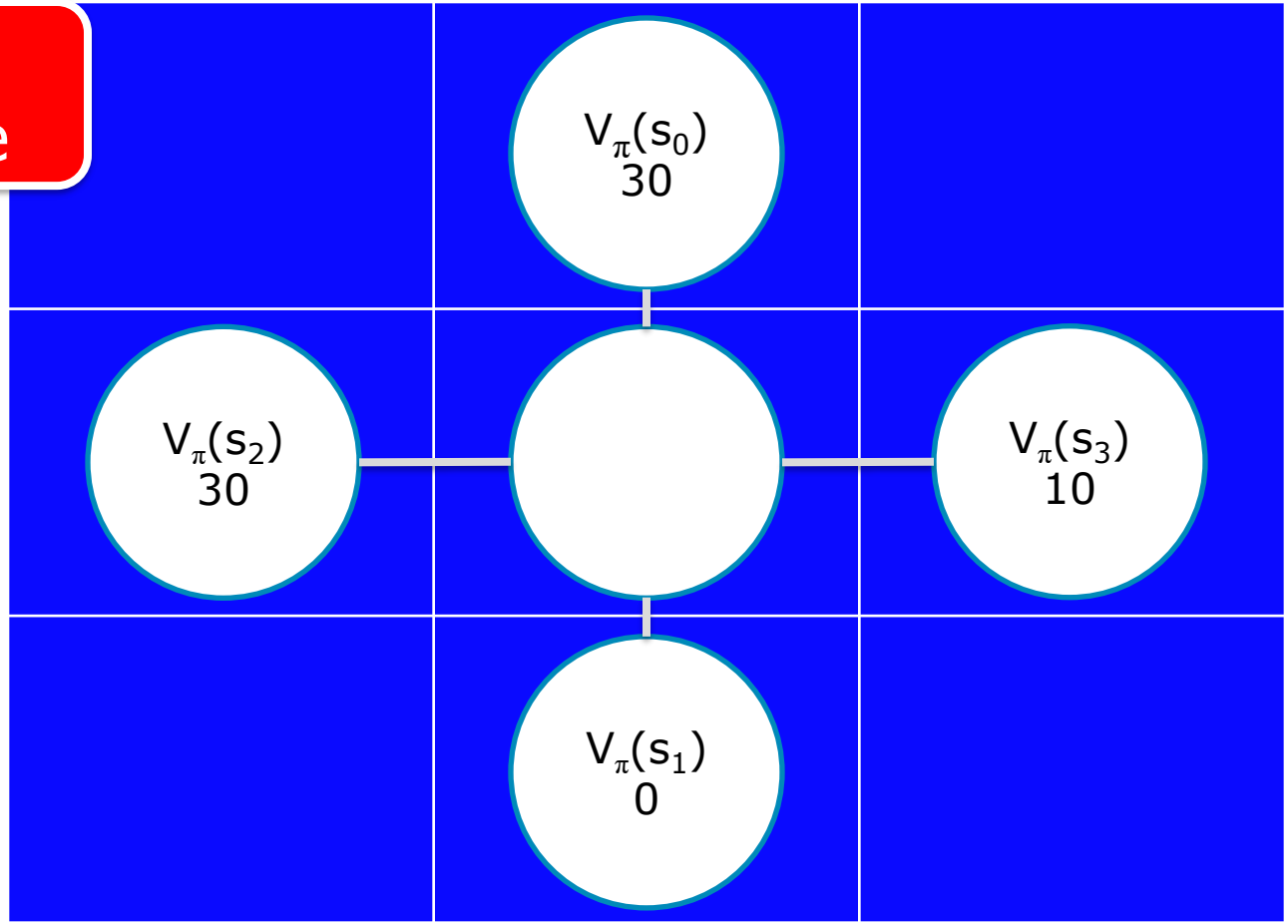
- Update the policy to choose the action that yields the highest value

$$\pi'(s) = \arg\max_{a} Q(s, a)$$

Policy Update



Policy Update



Policy Update In Code

```
1.  p[s][a] = prob of choosing a at s (policy)
2.  maxA = [], maxV = -infinity
3.  for each legal action a at s:
4.      s' = apply a to s
5.      if (v[s'] > maxV)          maxA=[]; maxA.add(a); maxV = v[s']
6.      else if (v[s'] == maxV)    maxA.push(a);
7.      else if (v[s'] < maxV)     // do nothing
8.  for each legal action a at s:
9.      if (a in maxA)             p[s][a] = 1.0 / maxA.length
10.     else                       p[s][a] = 0
```

Example Policy Update

- 5 actions at state s `[a1, a2, a3, a4, a5]`
- Yield new states `[s1, s2, s3, s4, s5]`
- With values `[ 7,  0, 10,  5, 10]`
- Max valued actions `[a3, a5]`
- New p[s][a] `[ 0,  0, .5,  0, .5]`

- New policy = equiprobable all max actions

Dynamic Programming

- Dynamic Programming can be very powerful when it is possible to use
- Need to know state transition probabilities

$$V^{\pi}(s) = \sum_{a} \pi(s,a) \sum_{s'} \mathcal{P}^{a}_{ss'} \left[\mathcal{R}^{a}_{ss'} + \gamma V^{\pi}(s') \right]$$

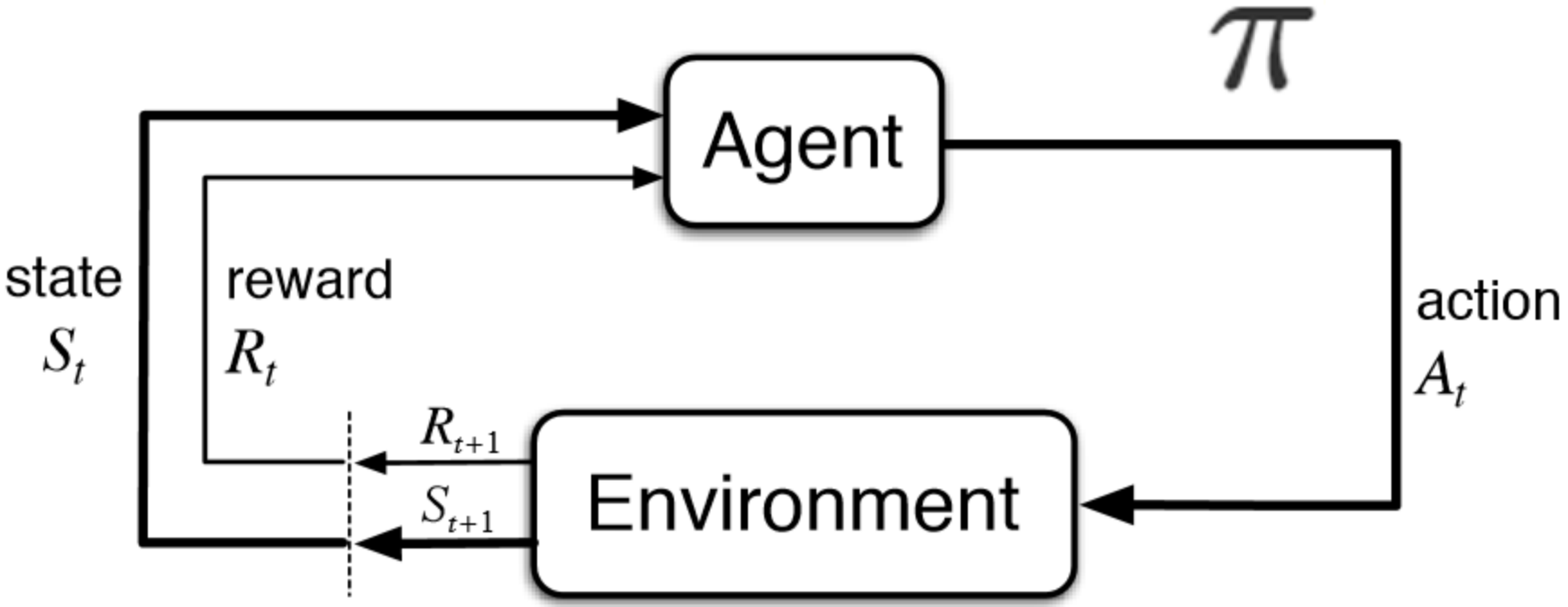
- In practice, can't use for complex problems

Value / Policy Iteration

- Obtaining optimal policy involves these two steps, repeated multiple times:

1. Update value estimates based on policy
2. Update policy to choose actions which lead us to maximum valued states

Reinforcement Learning



Exam Questions

- Action-Value vs State-Value (Q vs V)
- MDP Properties / Definition
- DP Example Calculation
  - Given neighbour values, calculate $V_{\pi}(S)$
- Policy Update
  - Given values, update policy
- Combination of previous 2