

# Programação Dinâmica

# O que é?

*Definição*

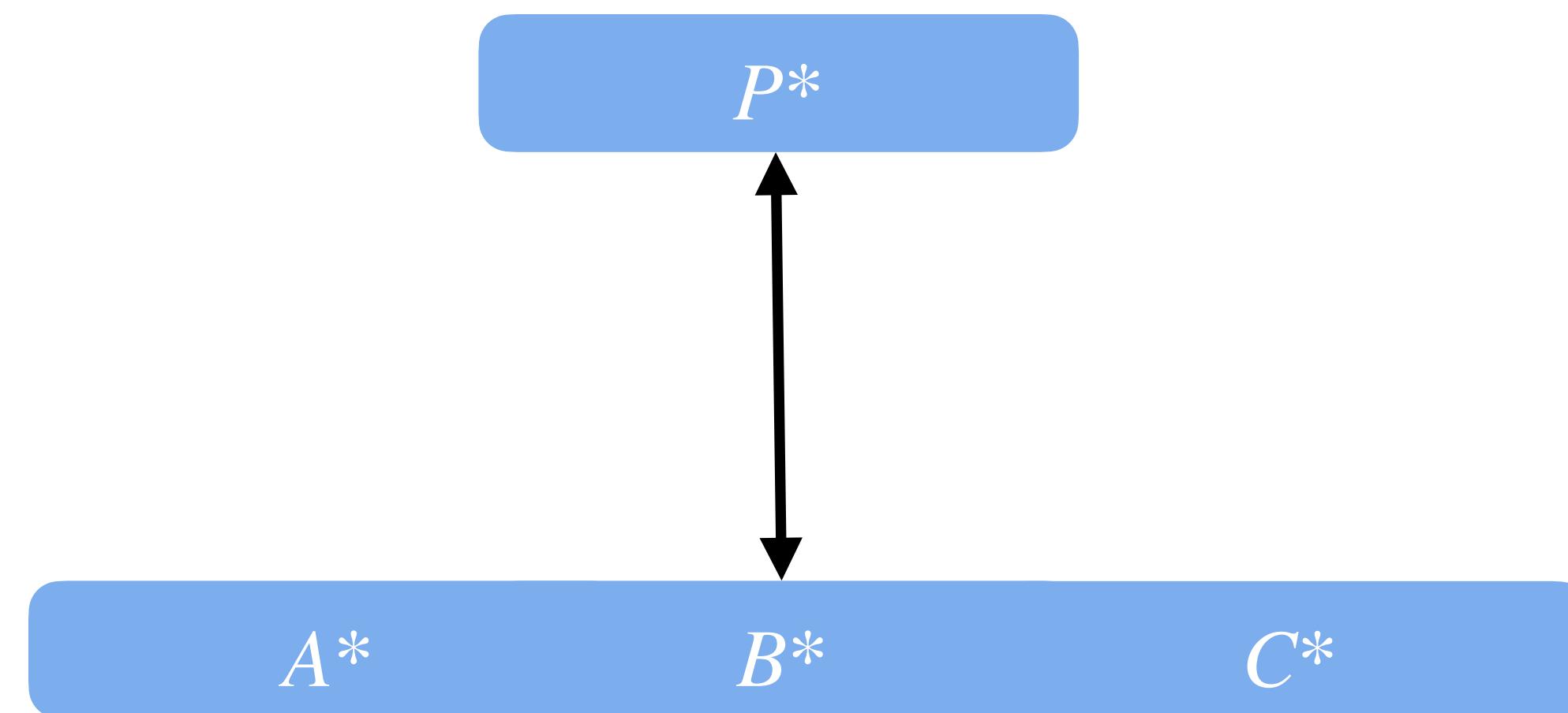
- Programação Dinâmica não se resume a apenas um algoritmo de RL
- É uma maneira de encontrar uma solução de um problema diminuindo-o em problemas menores, fáceis de resolver
- Utiliza-se de um modelo completo
- Qual o nosso problema? Encontrar  $\pi^*$

# O que é?

*Propriedades 1*

- PD só pode ser usado se e somente se:

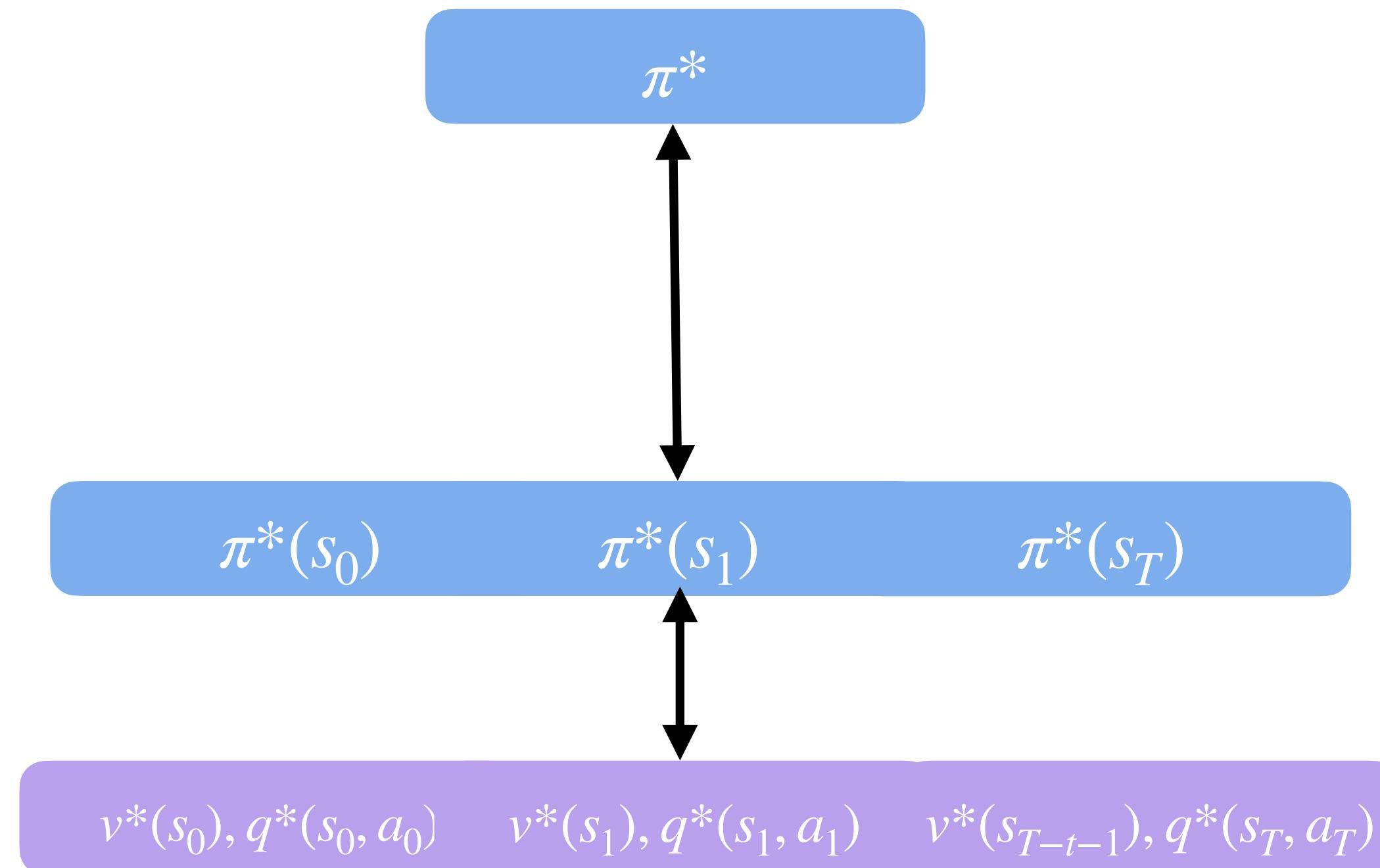
**O problema é separável**



# O que é?

*Propriedades 1*

- Neste caso, PD procura a melhor politica para cada estado:

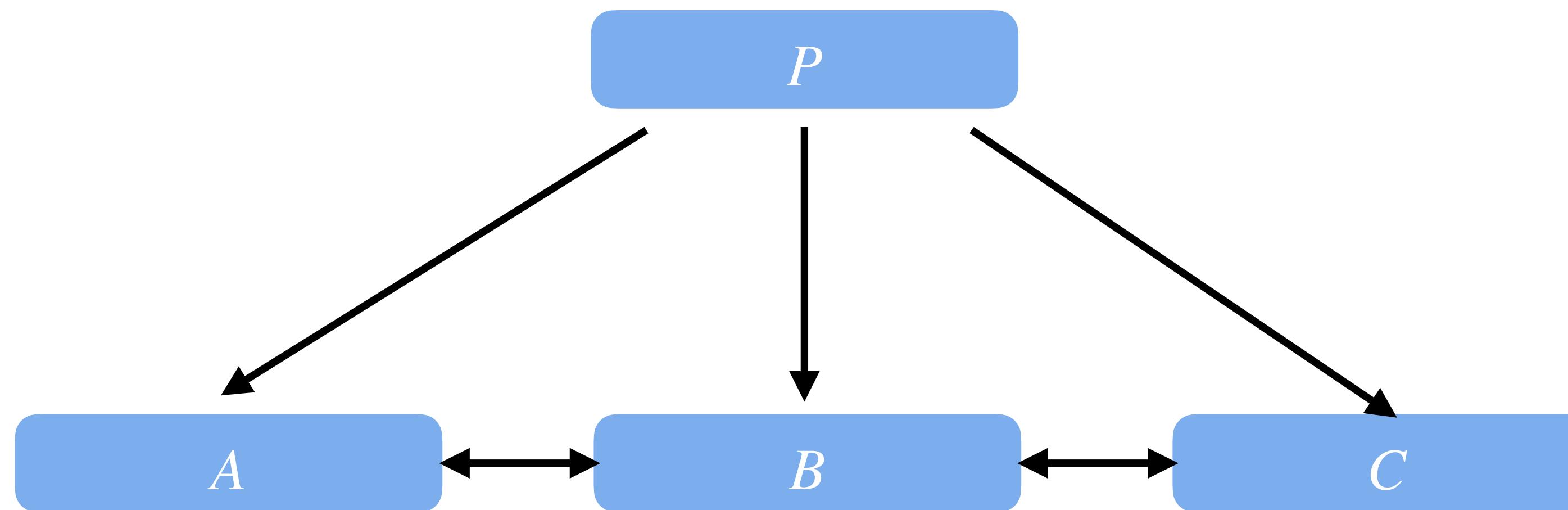


# O que é?

*Propriedades 2*

- PD só pode ser usado se e somente se:

**As soluções de cada subproblema  
São mutualmente dependentes**



# O que é?

*Propriedades 2*

- Voltemos mais uma vez as equações de Bellman

$$v^*(s) = \max_a \sum_{s',r} p(s', r | s, a) [r + \gamma v^*(s')]$$

$$q^*(s, a) = \sum_{s',r} p(s', r | s, a) [r + \gamma \max_{a'} q^*(s', a')]$$

# Como funciona a PD

- PD transforma essas equações em formulas de atualização de estado:

$$v^*(s) = \max_a \sum_{s',r} p(s', r | s, a)[r + \gamma v^*(s')]$$

$$V(s) \leftarrow \max_a \sum_{s',r} p(s', r | s, a)[r + \gamma v^*(s')]$$

# Como funciona o PD

- PD usa as formulas para estimar o valor de cada estado e guardar em uma tabela de transição

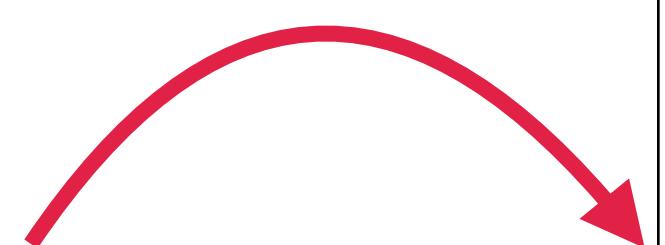
$v(s_0)$	3
$v(s_1)$	2
$v(s_2)$	2
$v(s_T)$	1

# Como funciona o PD

- PD atualiza iterativamente a tabela com os valores das equações:

$$v^*(s) = \max_a \sum_{s',r} p(s', r | s, a)[r + \gamma v^*(s')]$$

$$V(s) \leftarrow \max_a \sum_{s',r} p(s', r | s, a)[r + \gamma v^*(s')]$$

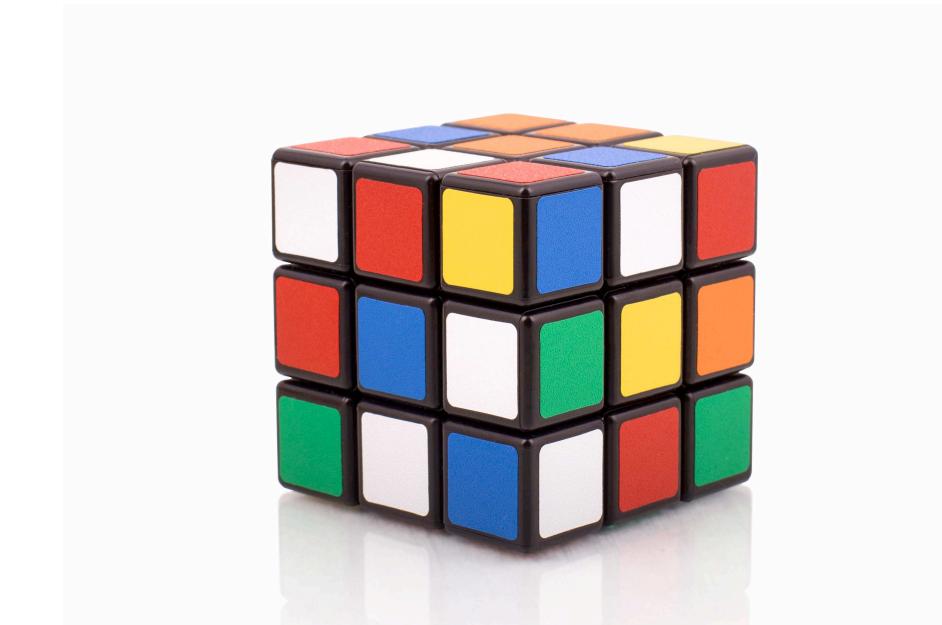


$v(s_0)$	3
$v(s_1)$	2
$v(s_2)$	2
$v(s_T)$	1

# Como funciona o PD

- É preciso saber **antecipadamente** a probabilidade de cada estado e quais pagamentos se recebem ao realizar a ação em cada estado, ou seja **NECESSITA DE UM MODELO**

$$V(s) \leftarrow \max_a \sum_{s',r} p(s', r | s, a) [r + \gamma v^*(s')]$$



# Primeiro algoritmo: Iteração por Valores de Estado

*State-value iteration*

- Para se achar  $\pi^*$  deve-se achar  $v^*$

$$\pi_*(s) = \operatorname{argmax}_a \sum_{s',r} p(s', r | s, a) [r + \gamma v^*(s)]$$

# Primeiro algoritmo: Iteração por Valores de Estado

*State-value iteration*

- Criar uma tabela com a estimativa de cada estado (arbitrário)
- Atualizar **iterativamente** cada estado usando
$$V(s) \leftarrow \max_a \sum_{s',r} p(s', r | s, a)[r + \gamma v^*(s')]$$
- Repetir quantas vezes for necessária até que se chegue no valor da política ótima  $\pi^*$

# Primeiro algoritmo: Iteração por Valores de Estado

*State-value iteration*

---

**Algorithm 2** Value Iteration

---

```
1: Input:  $\theta > 0$  tolerance parameter,  $\gamma$  discount factor
2: Initialize  $V(s)$  arbitrarily, with  $V(\text{terminal}) = 0$ 
3: repeat
4:    $\Delta \leftarrow 0$ 
5:   for  $s \in S$  do
6:      $v \leftarrow V(s)$ 
7:      $V(s) \leftarrow \max_{a \in A(s)} \sum_{s',r} p(s',r|s,a)[r + \gamma V(s')]$ 
8:      $\Delta \leftarrow \max(\Delta, |v - V(s)|)$ 
9:   end for
10:  until  $\Delta > \theta$ 
11: Output:  $\pi$ : greedy policy w.r.t.  $V(s)$ 
```

---

# Primeiro algoritmo: Iteração por Valores de Estado

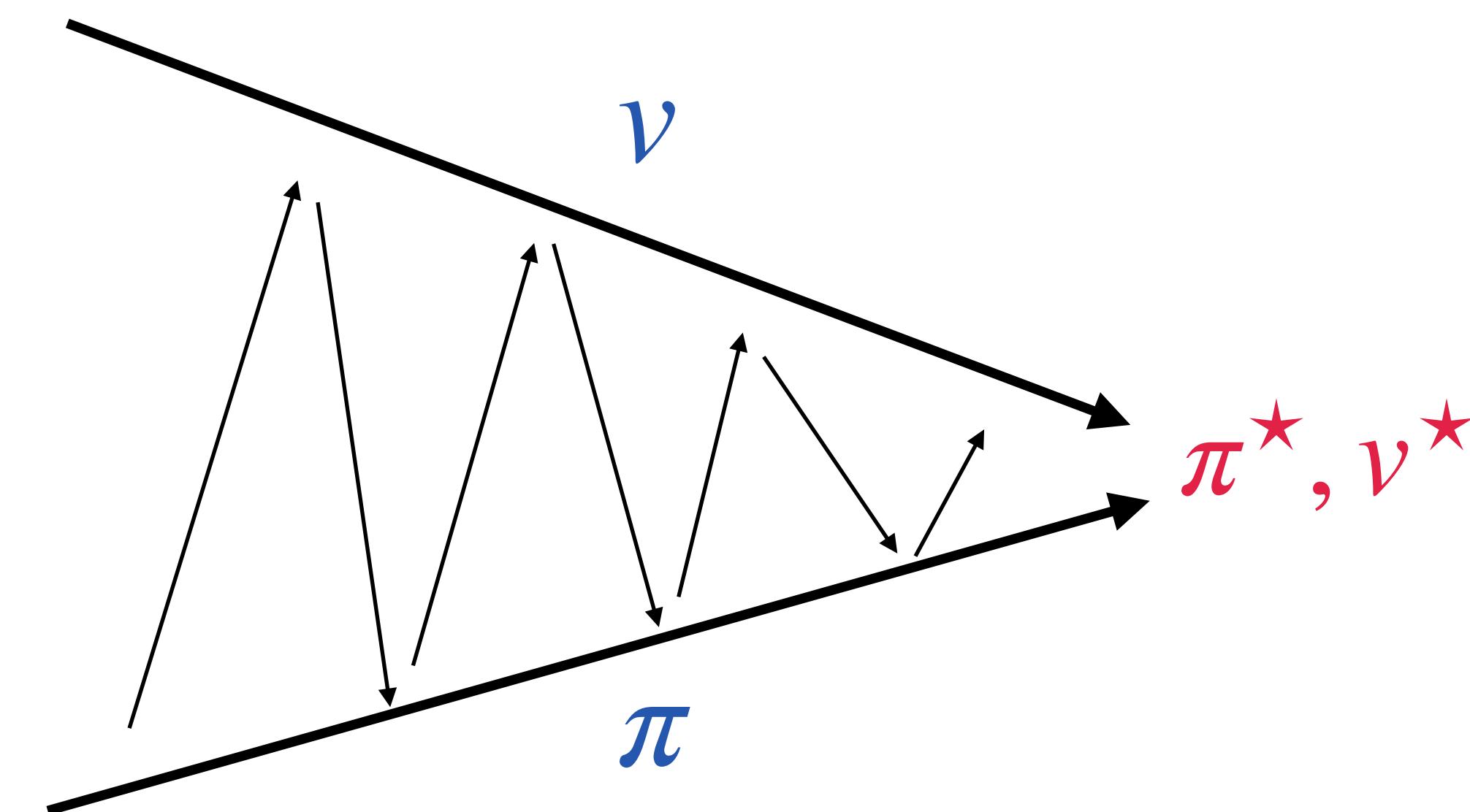
*State-value iteration*

**Abra o Notebook Value\_iteration.ipynb**

# Segundo algoritmo: Iteração por Políticas

*Policy Iteration*

- Alicerce da maioria dos algoritmos de RL
- Alterna entre estimar os valores de estado e a política



# Segundo algoritmo: Iteração por Políticas

## *Policy Iteration*

### **Algorithm 2** Policy Iteration

```
1: Input:  $\theta > 0$  tolerance parameter,  $\gamma$  discount factor
2: Initialize  $V(s)$  and  $\pi(a|s)$  arbitrarily
3: while policy-stable = false do
4:
5:   Policy Evaluation:
6:   while  $\Delta > \theta$  do
7:      $\Delta \leftarrow 0$ 
8:     for  $s \in S$  do
9:        $v \leftarrow V(s)$ 
10:       $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma V(s')]$ 
11:       $\Delta \leftarrow \max(\Delta, |v - V(s)|)$ 
12:    end for
13:   end while
14:
15:   Policy Improvement:
16:   policy-stable = true
17:   for  $s \in S$  do
18:     old-action  $\leftarrow \pi(s)$ 
19:      $\pi(s) \leftarrow \arg \max_{a \in A(s)} \sum_{s',r} p(s',r|s,a)[r + \gamma V(s')]$ 
20:     if old-action  $\neq \pi(s)$  then
21:       policy-stable  $\leftarrow \text{false}$ 
22:     end if
23:   end for
24:
25: end while
26: Output: Optimal policy  $\pi(a|s)$  and state values  $V(s)$ 
```

# Segundo algoritmo: Iteração por Políticas

*Policy iteration*

**Abra o Notebook Policy\_iteration.ipynb**

# Segundo algoritmo: Iteração por Políticas

*Policy Iteration*

$$v_{\pi}(s) \leftarrow \sum_a \pi(s | a) \sum_{s',r} p(s', r | s, a) [r + \gamma v_{\pi}(s')]$$

$$V(s) \leftarrow \sum_a \pi(s | a) \sum_{s',r} p(s', r | s, a) [r + \gamma V(s')]$$

```
5: Policy Evaluation:  
6: while  $\Delta > \theta$  do  
7:    $\Delta \leftarrow 0$   
8:   for  $s \in S$  do  
9:      $v \leftarrow V(s)$   
10:     $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r | s, a) [r + \gamma V(s')]$   
11:     $\Delta \leftarrow \max(\Delta, |v - V(s)|)$   
12:   end for  
13: end while
```

# Como melhorar uma política?

*Policy Iteration*

- A política melhoraria se mudássemos a ação escolhida no atual estado?
- E melhor seguir a política como é ou mudar a ação?

$$q_{\pi}(s, a) = \sum_{s', r} p(s', r | s, a)[r + \gamma v_{\pi}(s')]$$

# Teorema da melhora de Política

*Policy Iteration*

- Alicerce da maioria dos algoritmos de RL
- Alterna entre estimar os valores de estado e a política

Se:

$$\underline{q}_{\underline{\pi}}(s, \underline{\pi}'(s)) \geq v_{\pi}(s)$$

Então:

$$v_{\pi'}(s) \geq v_{\pi}(s)$$

# Teorema da melhora de Política

## *Policy Iteration*

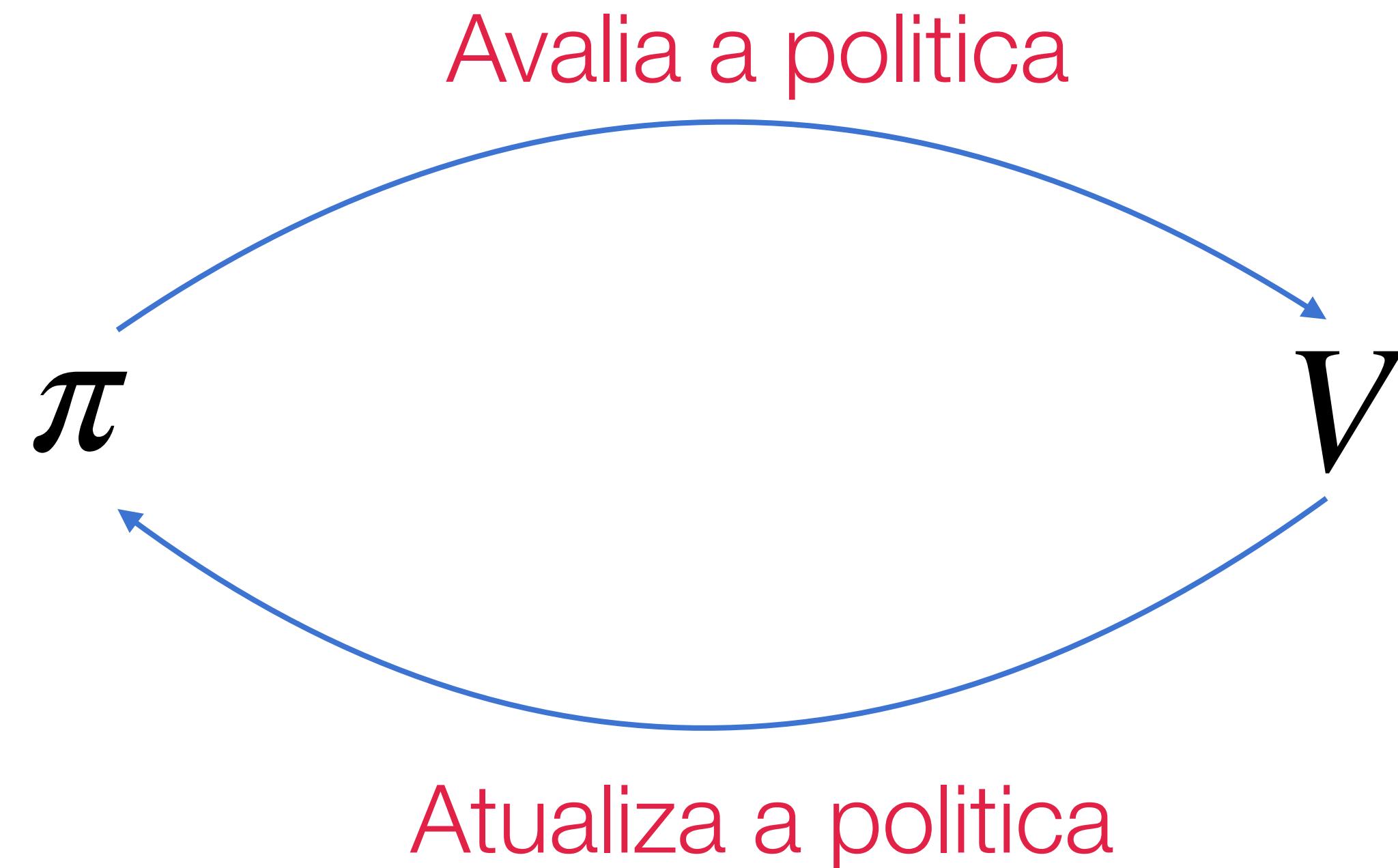
- Alicerce da maioria dos algoritmos de RL
- Alterna entre estimar os valores de estado e a política

```
15: Policy Improvement:  
16: policy-stable = true  
17: for  $s \in S$  do  
18:   old-action  $\leftarrow \pi(s)$   
19:    $\pi(s) \leftarrow \arg \max_{a \in A(s)} \sum_{s',r} p(s',r|s,a)[r + \gamma V(s')]$   
20:   if old-action  $\neq \pi(s)$  then  
21:     policy-stable  $\leftarrow false$   
22:   end if  
23: end for
```

# Padrão dos algoritmos de RL

*Policy Iteration*

- Daqui em diante, todos os algoritmos de RL irão seguir mais ou menos o mesmo processo de atualização de política da PD



# Padrão dos algoritmos de RL

*Policy Iteration*

- Tentam replicar o que o PD faz, sem necessitar de um modelo
- DP é terrivelmente ineficiente
- Utiliza-se de **tentativa e erro** para construir um modelo
- Muito mais eficiente que PD



# Próximo destino: Monte Carlo

*Policy iteration*

**Vamos ver um algoritmo de RL que  
utiliza tentativa e erro para construir  
um modelo e atualizar  $\pi$**

