Motion learning in robotics

Karel Zimmermann

http://cmp.felk.cvut.cz/~zimmerk/

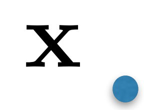
Vision for Robotics and Autonomous Systems
https://cyber.felk.cvut.cz/vras/

Center for Machine Perception
https://cmp.felk.cvut.cz

Department for Cybernetics
Faculty of Electrical Engineering
Czech Technical University in Prague

Tasks often formalised as MDP

States: $\mathbf{x} \in \mathcal{R}^n$



Czech Technical University in Prague
Faculty of Electrical Engineering, Department of Cybernetics

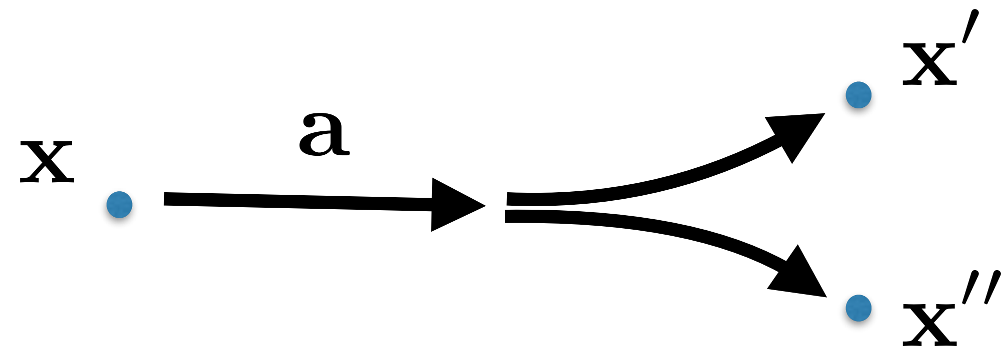
Both tasks formalised as reinforcement learning problems

States: $\mathbf{x} \in \mathcal{R}^n$

Actions: $\mathbf{a} \in \mathcal{R}^m$

Both tasks formalised as reinforcement learning problems

States: $\mathbf{x} \in \mathcal{R}^n$

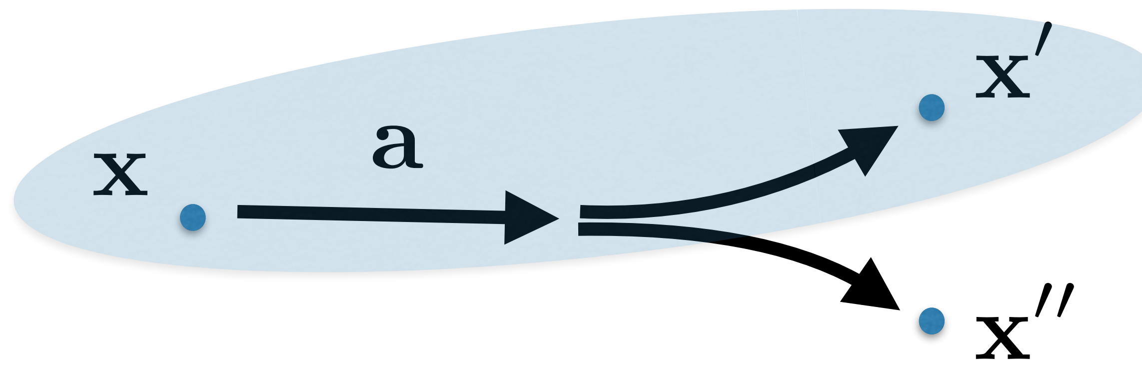


Actions: $\mathbf{a} \in \mathcal{R}^m$

Model: $p(\mathbf{x}'|\mathbf{x}, \mathbf{a})$

Both tasks formalised as reinforcement learning problems

States: $\mathbf{x} \in \mathcal{R}^n$



Actions: $\mathbf{a} \in \mathcal{R}^m$

Model: $p(\mathbf{x}'|\mathbf{x}, \mathbf{a})$

Rewards: $r(\mathbf{x}, \mathbf{a}, \mathbf{x}') \in \mathcal{R}$

Both tasks formalised as reinforcement learning problems

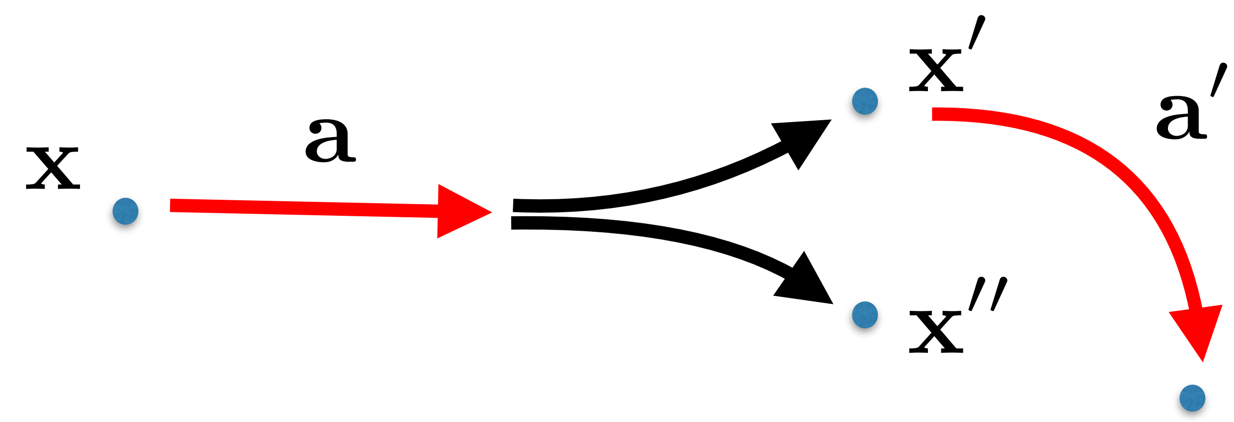
States: $\mathbf{x} \in \mathcal{R}^n$

Actions: $\mathbf{a} \in \mathcal{R}^m$

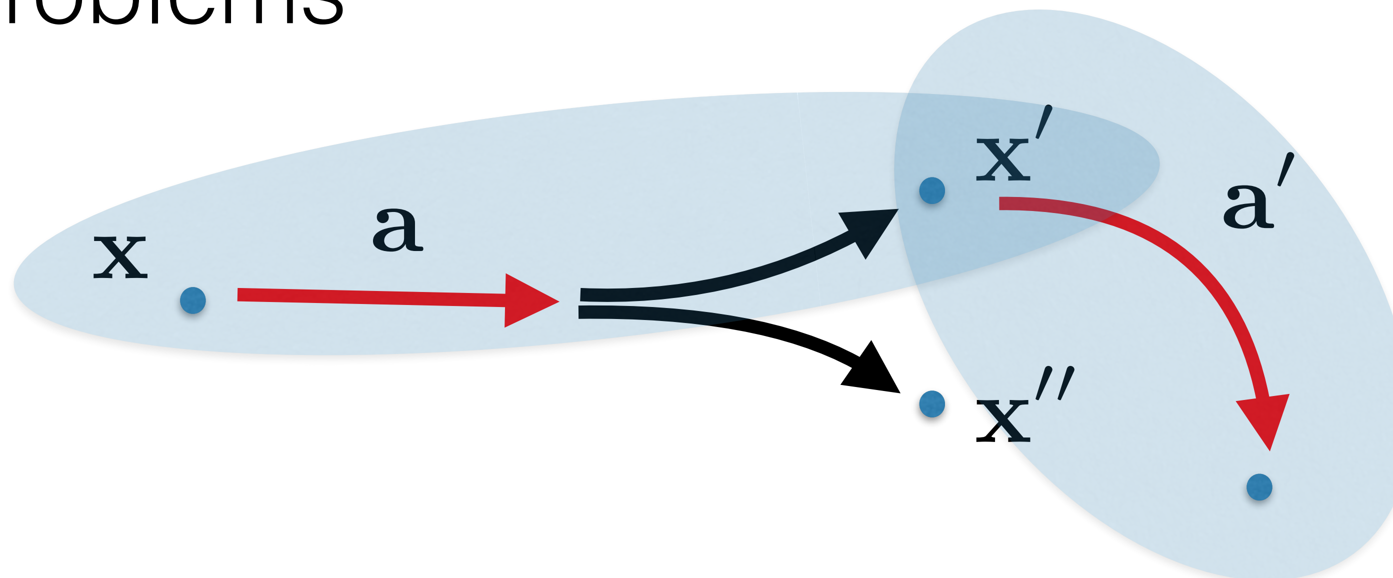
Model: $p(\mathbf{x}'|\mathbf{x}, \mathbf{a})$

Rewards: $r(\mathbf{x}, \mathbf{a}, \mathbf{x}') \in \mathcal{R}$

Policy: $\pi(\mathbf{a}|\mathbf{x})$



Both tasks formalised as reinforcement learning problems



States: $\mathbf{x} \in \mathcal{R}^n$

Actions: $\mathbf{a} \in \mathcal{R}^m$

Model: $p(\mathbf{x}'|\mathbf{x}, \mathbf{a})$

Rewards: $r(\mathbf{x}, \mathbf{a}, \mathbf{x}') \in \mathcal{R}$

Policy: $\pi(\mathbf{a}|\mathbf{x})$

Goal: $\pi^* = \arg\max_{\pi} J_\pi$ (e.g. $J_\pi = \mathtt{E}\left[\sum_{t=0}^{T} r_t\right]$)

Challenges in real tasks

States: $\mathbf{x} \in \mathcal{R}^n$ — incomplete, noisy

Actions: $\mathbf{a} \in \mathcal{R}^m$ — continuous high-dimensional

Model: $p(\mathbf{x}'|\mathbf{x}, \mathbf{a})$ — inaccurate model

Rewards: $r(\mathbf{x}, \mathbf{a}, \mathbf{x}') \in \mathcal{R}$ — hard to engineer

Policy: $\pi(\mathbf{a}|\mathbf{x})$ — execution endanger the robot

Goal: $\pi^* = \arg\max_{\pi} J_\pi$ (e.g. $J_\pi = \mathtt{E}\left[\sum_{t=0}^{T} r_t\right]$)

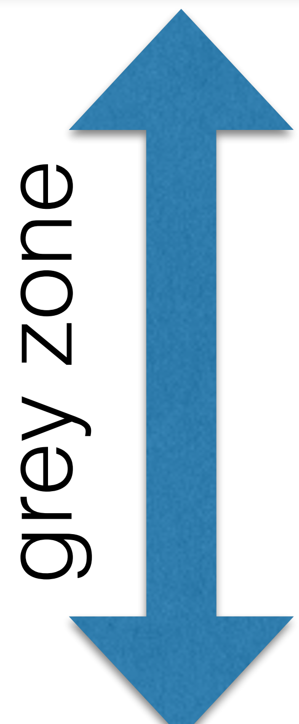
Challenges in real tasks

- Can I learn something without the model $p(\mathbf{x}'|\mathbf{x}, \mathbf{a})$ just from interactions?

Taxonomy of policy search methods

- Direct policy search (primal task)

  e.g. gradient ascent for $\pi^* = \arg\max_{\pi} J_\pi$



Episodic REPS [Peters, 2010]

PILCO [Deisenroth, ICML 2011]

Actor-critic (e.g. DPG [Silver,JMLR 2014])

Deep Q-learning (e.g. [Mnih,Nature 2015])

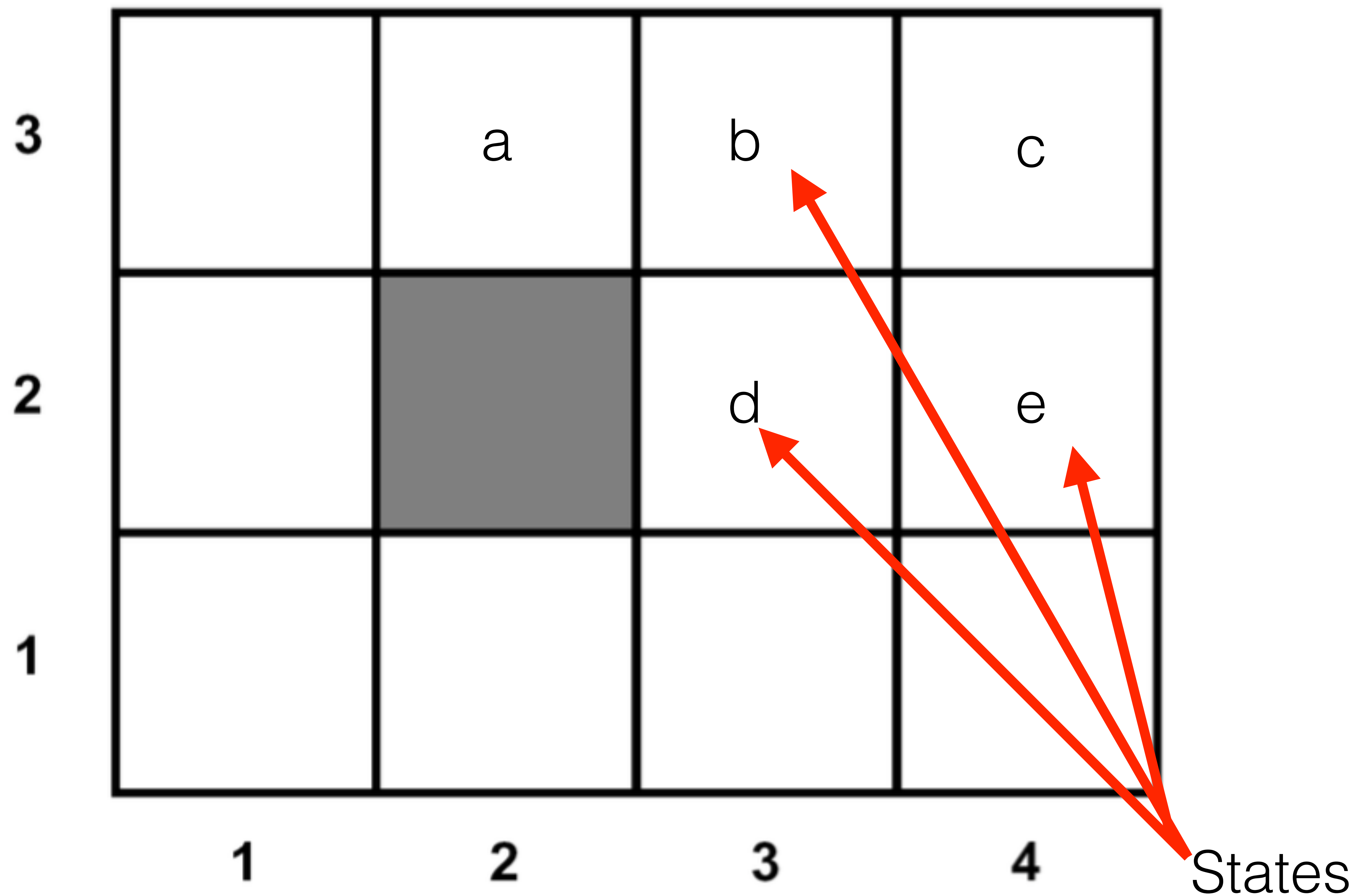
- Value-based methods (dual function [Kober, 2013])

  e.g. search for $Q(\mathbf{x},\mathbf{a}) = r(\mathbf{x},\mathbf{a},\mathbf{x}') + \gamma \max_{\mathbf{a}'} Q(\mathbf{x}',\mathbf{a}')$

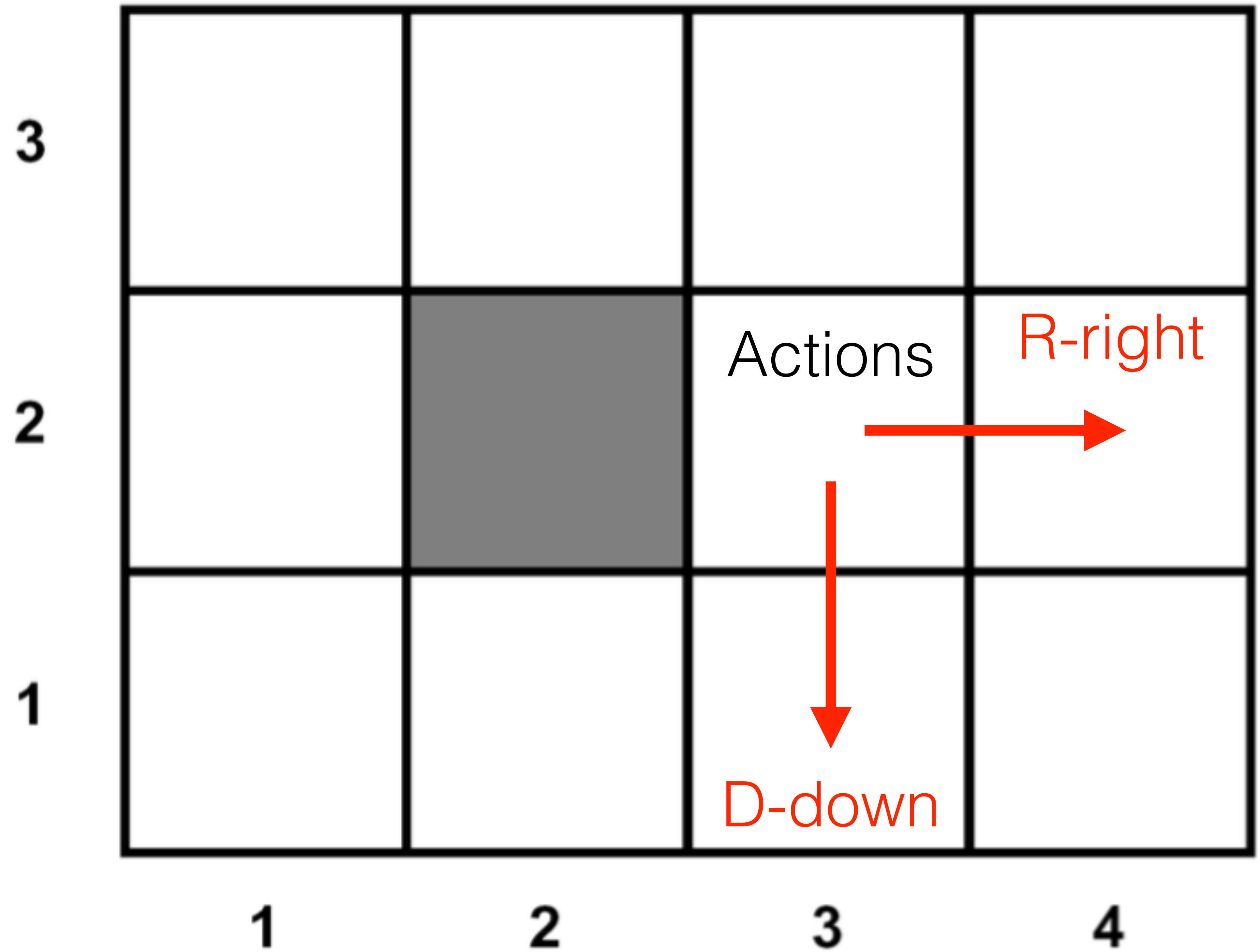
$$\pi^* = \arg\max_{a} Q(\mathbf{x},\mathbf{a})$$

Czech Technical University in Prague
Faculty of Electrical Engineering, Department of Cybernetics

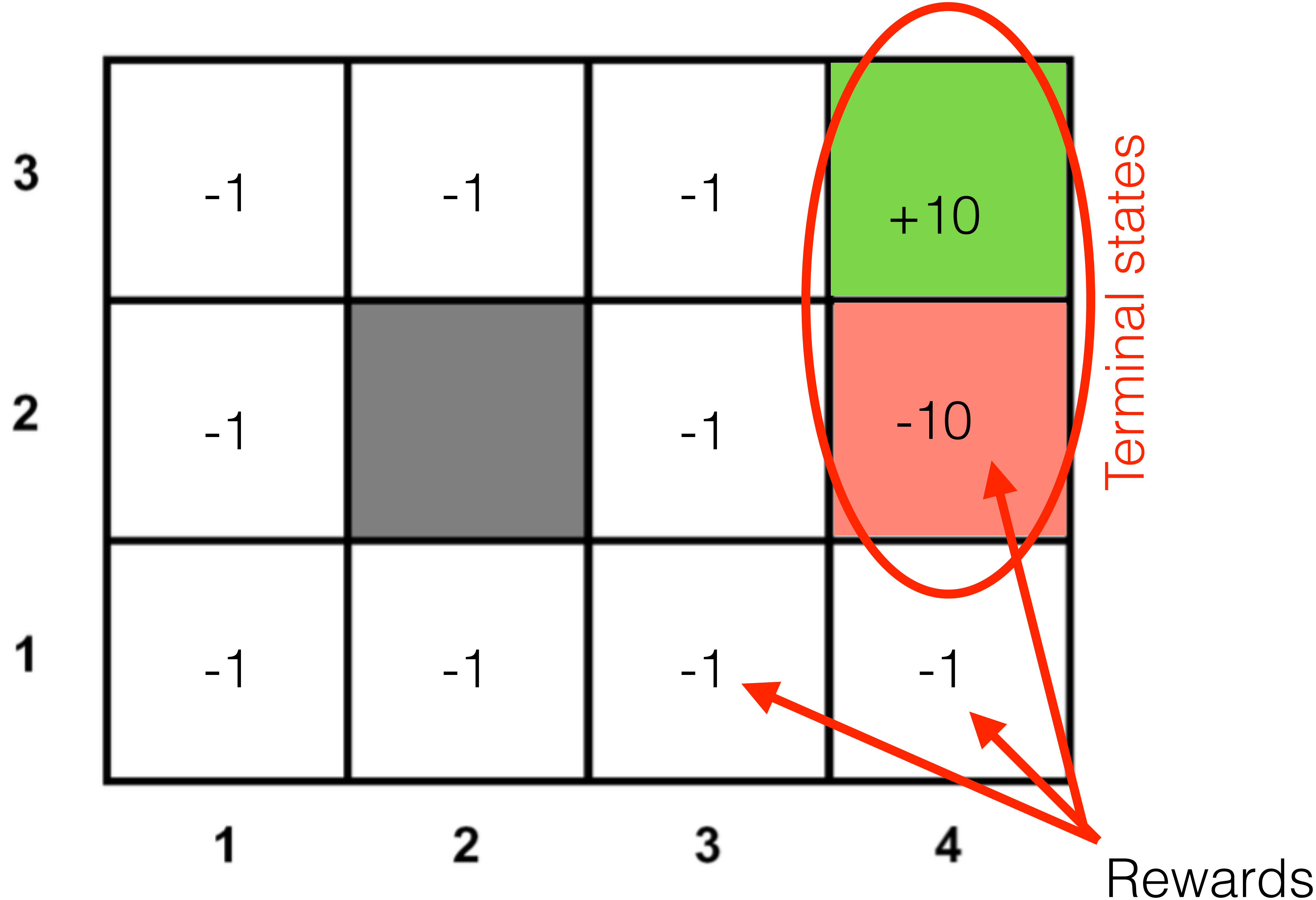
Value-based methods: Q-learning

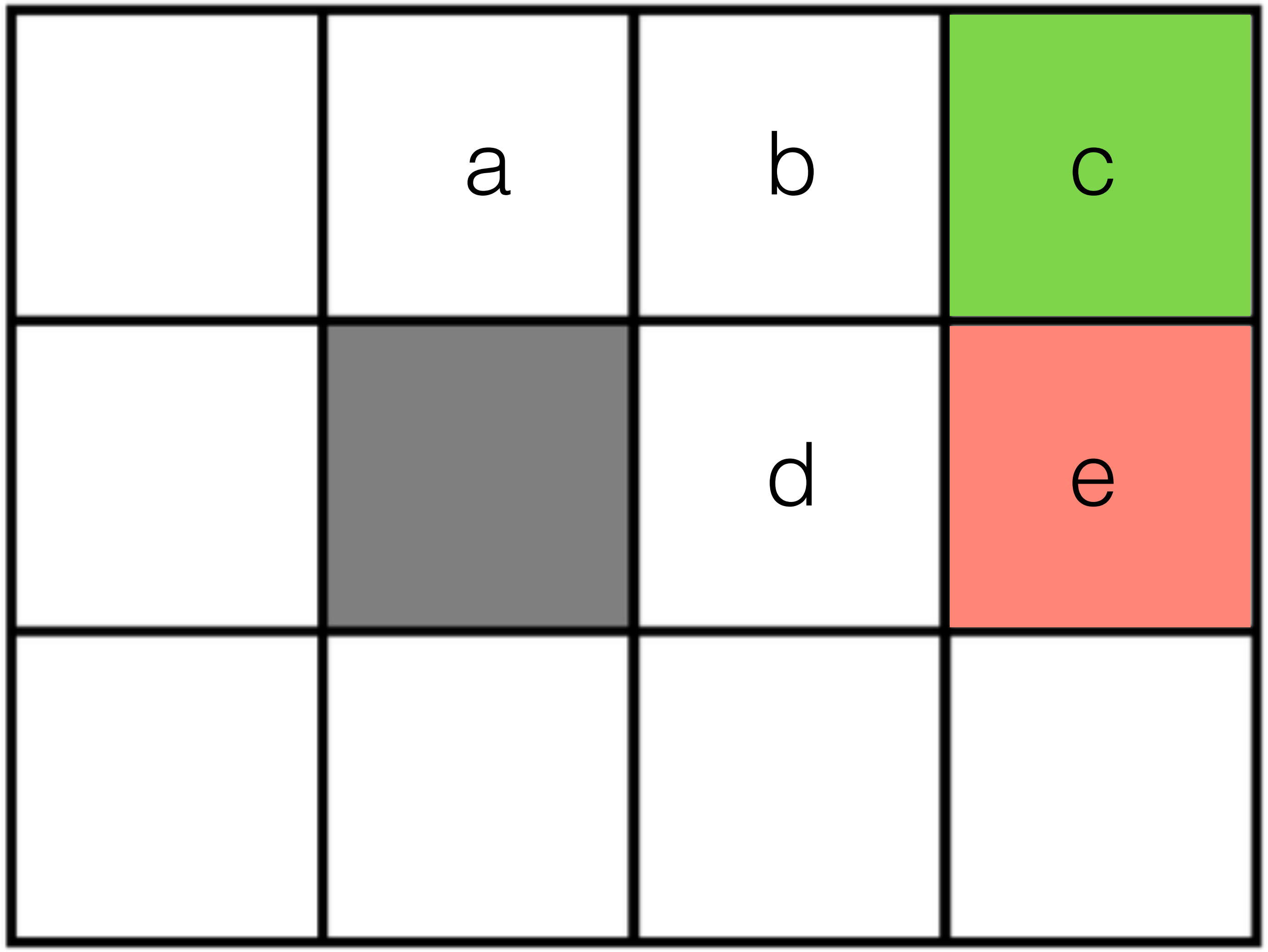


Value-based methods: Q-learning



Value-based methods: Q-learning



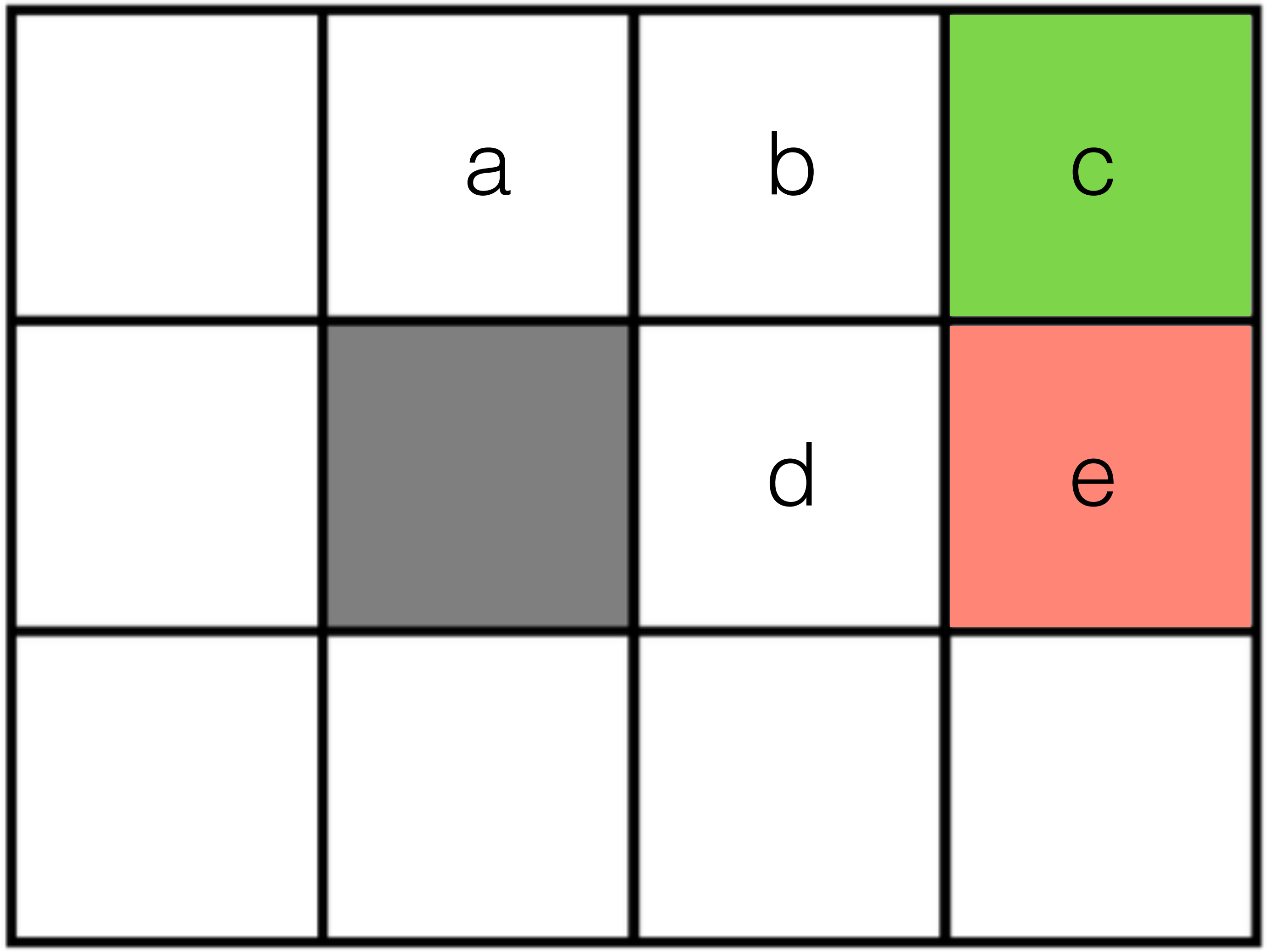


State-action value function

$$Q(\mathbf{x}, \mathbf{u}) : X \times U \rightarrow \mathbb{R}$$

The best sum of rewards I can get, when following action u in state x and then controlling optimally

- Search for the Q, which satisfies Bellman equation
$$Q(\mathbf{x}, \mathbf{u}) = r(\mathbf{x}, \mathbf{u}, \mathbf{x}') + \max_{\mathbf{u}'} Q(\mathbf{x}', \mathbf{u}')$$



State-action value function

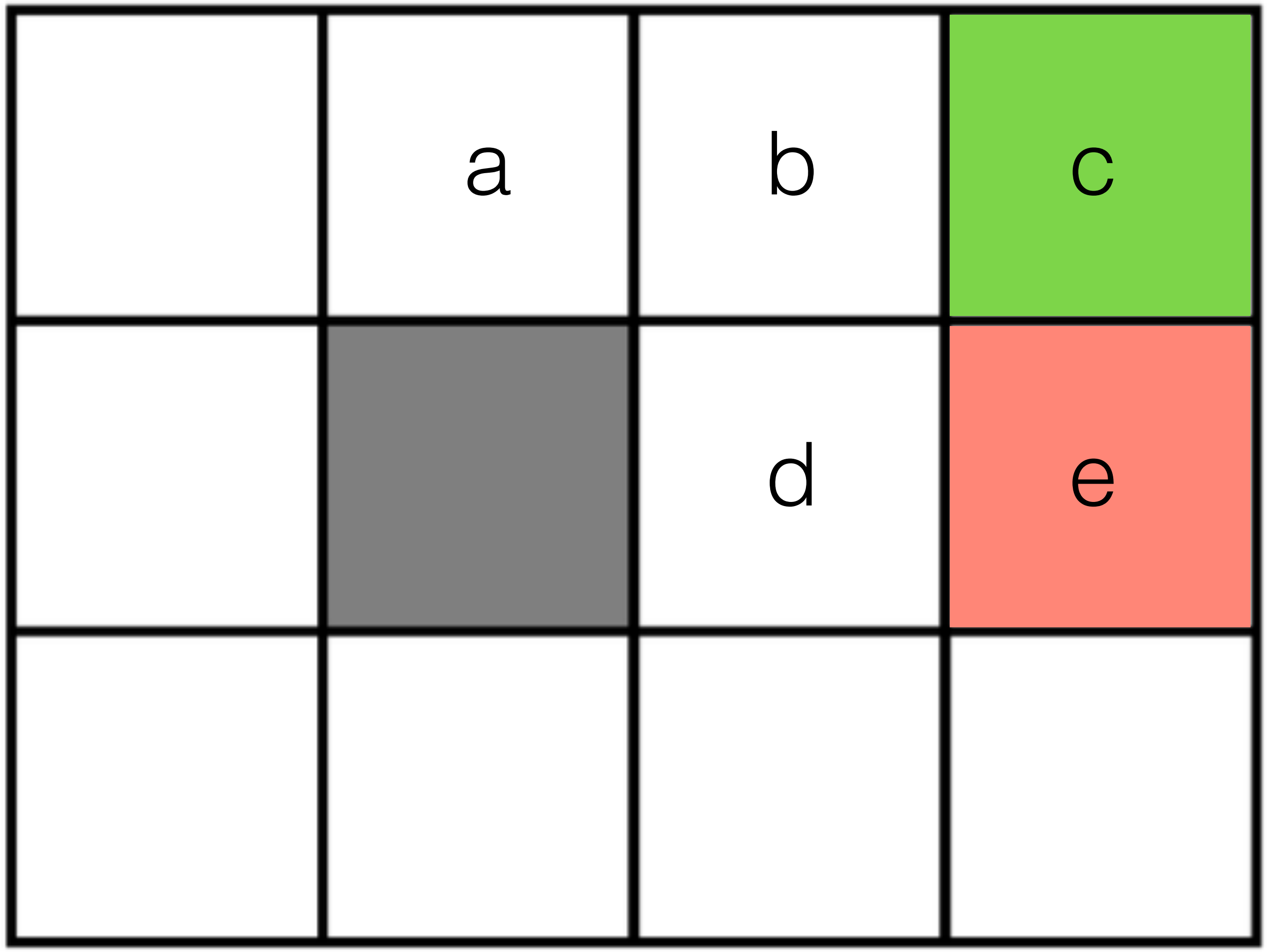
$$Q(\mathbf{x}, \mathbf{u}) : X \times U \rightarrow \mathbb{R}$$

The best sum of rewards I can get, when following action u in state x and then controlling optimally

- Search for the Q, which satisfies Bellman equation
$$Q(\mathbf{x}, \mathbf{u}) = r(\mathbf{x}, \mathbf{u}, \mathbf{x}') + \max_{\mathbf{u}'} Q(\mathbf{x}', \mathbf{u}')$$

- Once we find it, we can control optimally as follows:

$$\pi^*(\mathbf{x}) = \arg\max_{\mathbf{u}} Q(\mathbf{x}, \mathbf{u}) = \arg\max_{\pi} J_\pi$$



State-action value function

$$Q(\mathbf{x}, \mathbf{u}) : X \times U \rightarrow \mathbb{R}$$

The best sum of rewards I can get, when following action u in state x and then controlling optimally

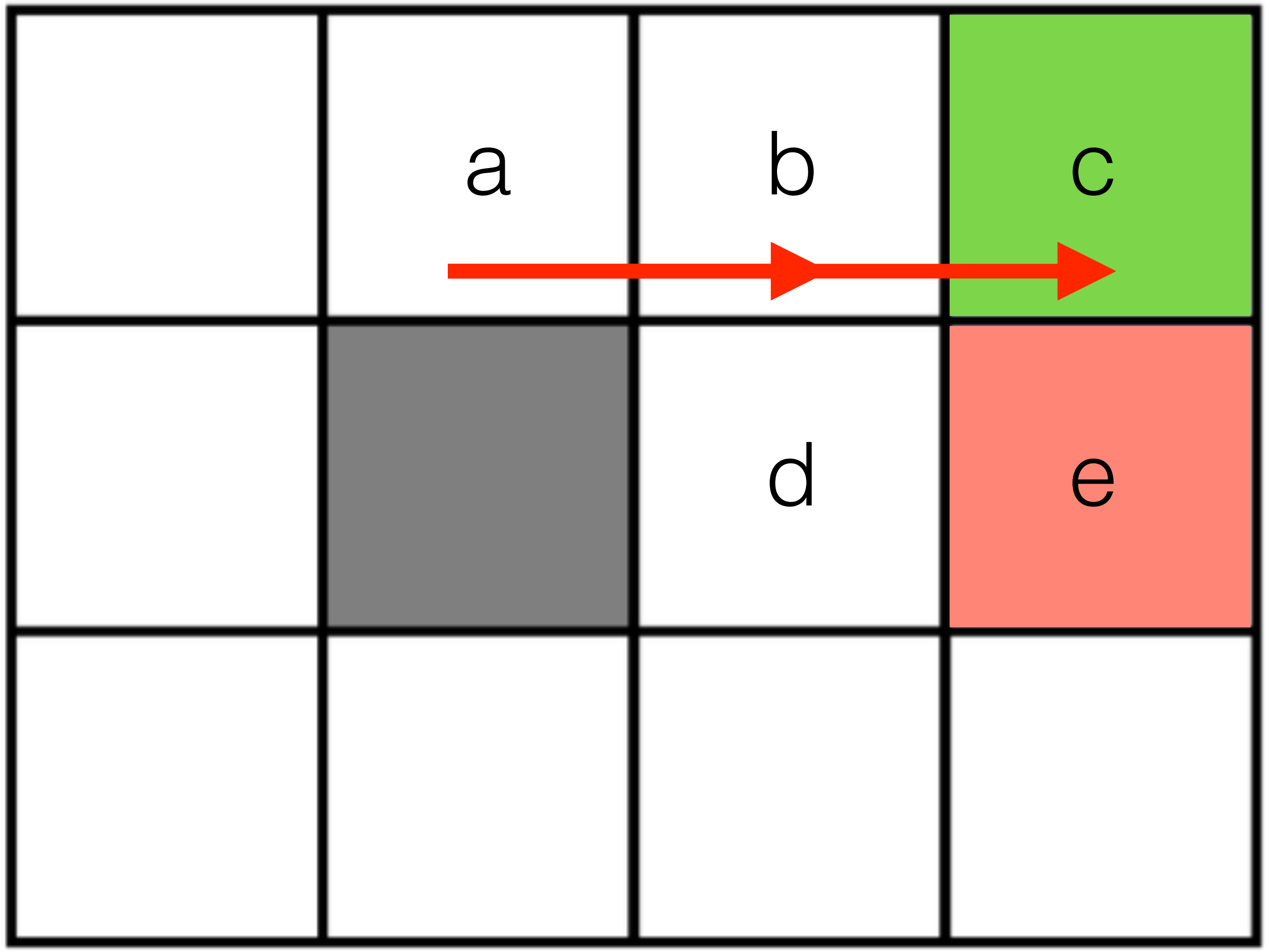
- Search for the Q, which satisfies Bellman equation

$$Q(\mathbf{x}, \mathbf{u}) = r(\mathbf{x}, \mathbf{u}, \mathbf{x}') + \max_{\mathbf{u}'} Q(\mathbf{x}', \mathbf{u}')$$

- Once we find it, we can control optimally as follows:

$$\pi^*(\mathbf{x}) = \arg\max_{\mathbf{u}} Q(\mathbf{x}, \mathbf{u}) = \arg\max_{\pi} J_\pi$$

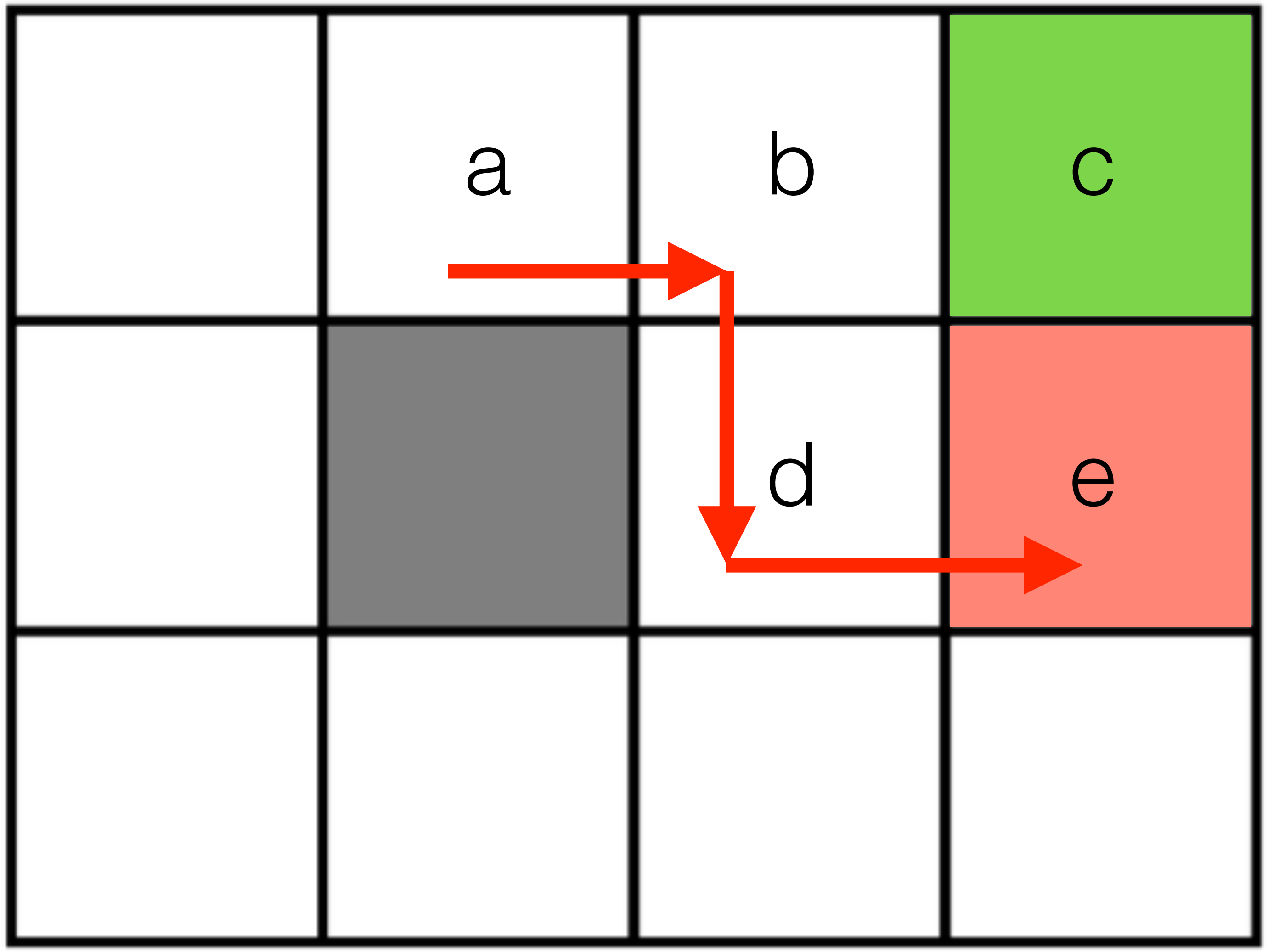
- Search without model is based on collecting trajectories



$$\tau_1 :$$

$$(a, R, \text{-}1),\ (b, R, \text{-}1),\ (c, R, 10)$$

Q	R - right	D - down
a	?	?
b	?	?
c	?	?
d	?	?
e	?	?

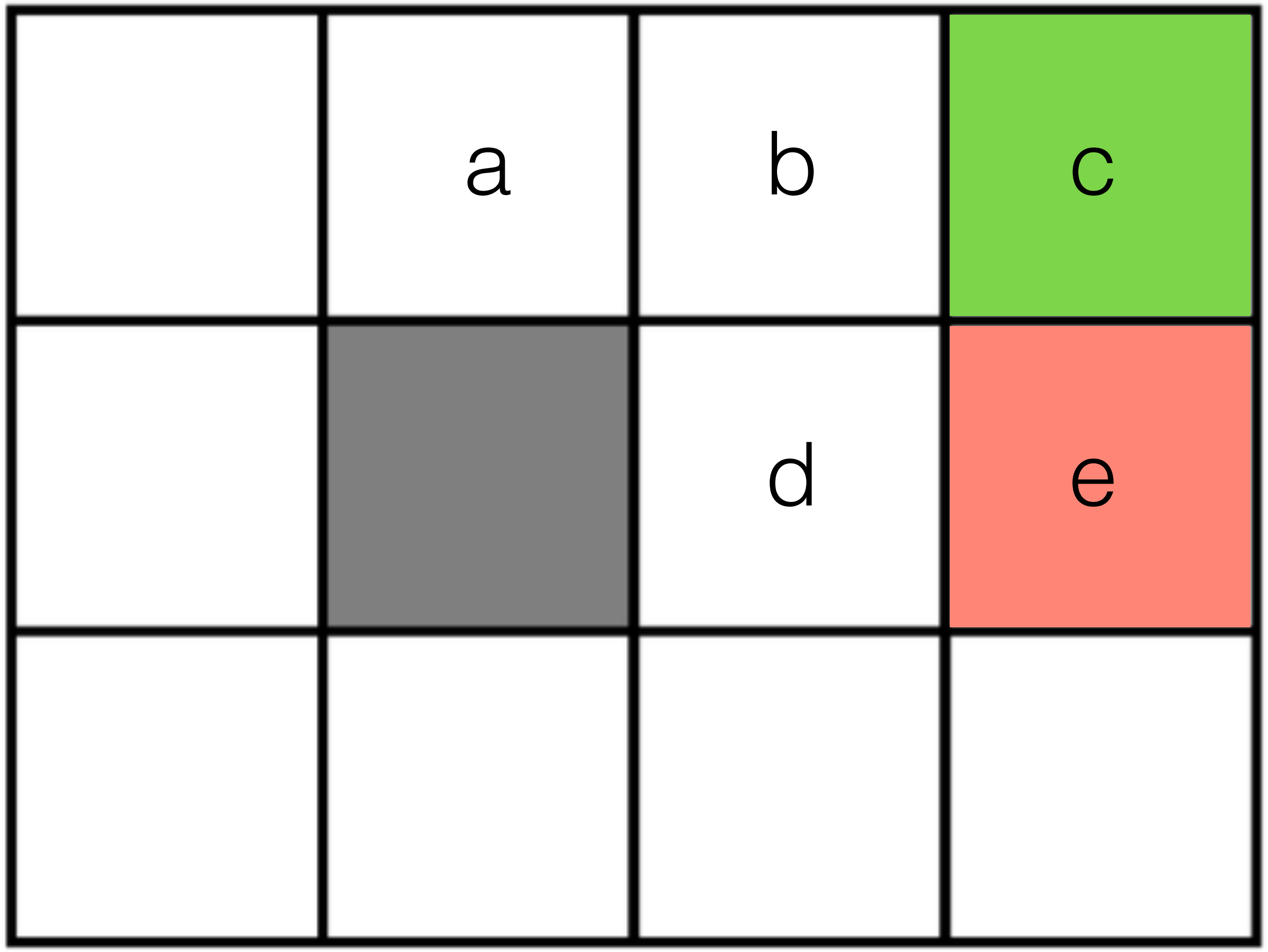


$$\tau_2 :$$

$$(a, R, \text{-}1),\ (b, D, \text{-}1),$$
$$(d, R, \text{-}1),\ (e, R, \text{-}10)$$

Q	R - right	D - down
a	?	?
b	?	?
c	?	?
d	?	?
e	?	?

Having a trajectory, each transition gives one equation



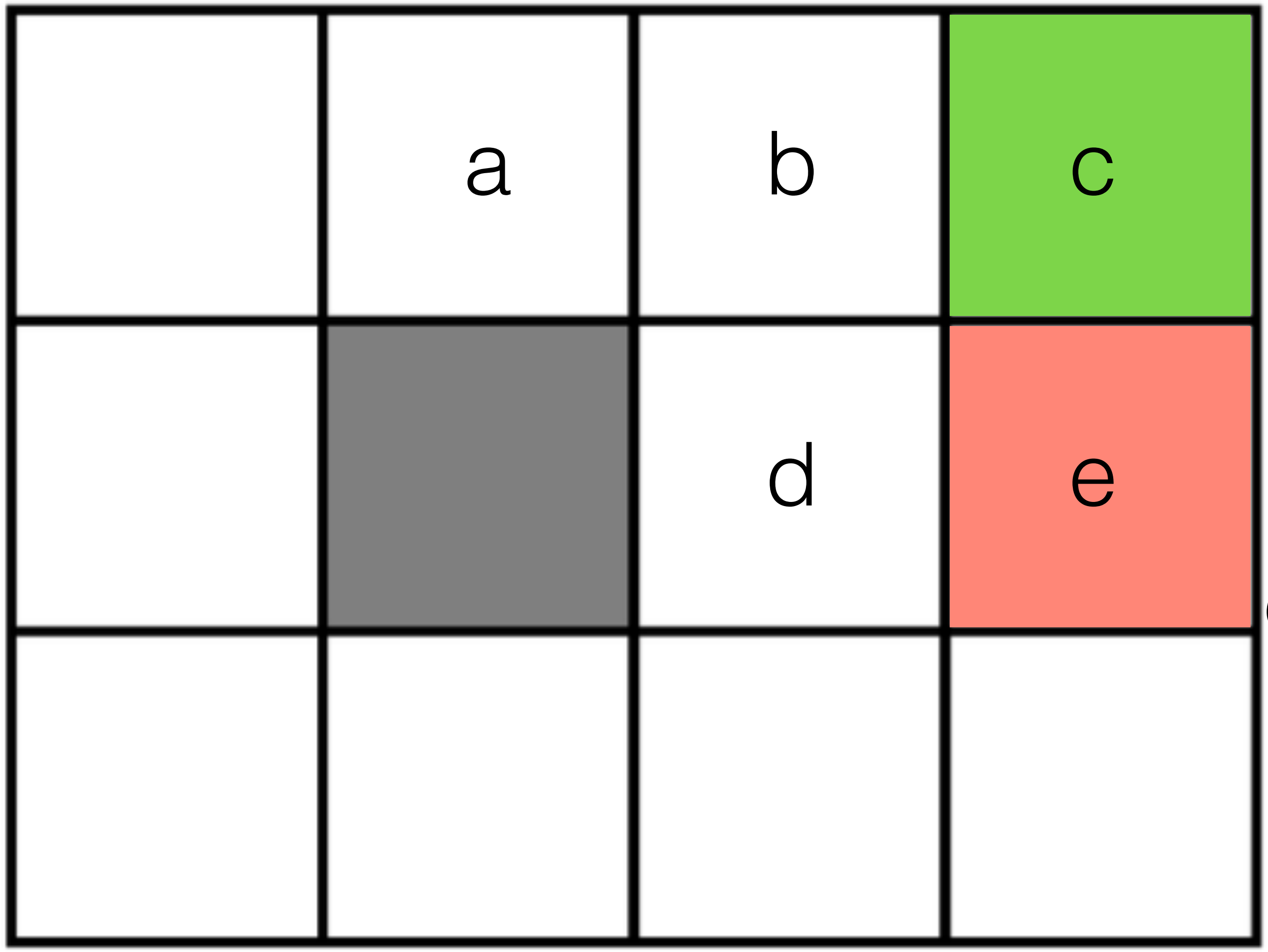
τ_2 :

$(a, R, \text{-}1),\ (b, D, \text{-}1),$
$(d, R, \text{-}1),\ (e, R, \text{-}10)$

$Q(\mathrm{e}, \mathrm{R}) = r(\mathrm{e})$

Q	R - right	D - down
a	?	?
b	?	?
c	?	?
d	?	?
e	?	?

Having a trajectory, each transition gives one equation



τ_2 :

$(a, R, \text{-}1),\ (b, D, \text{-}1),$

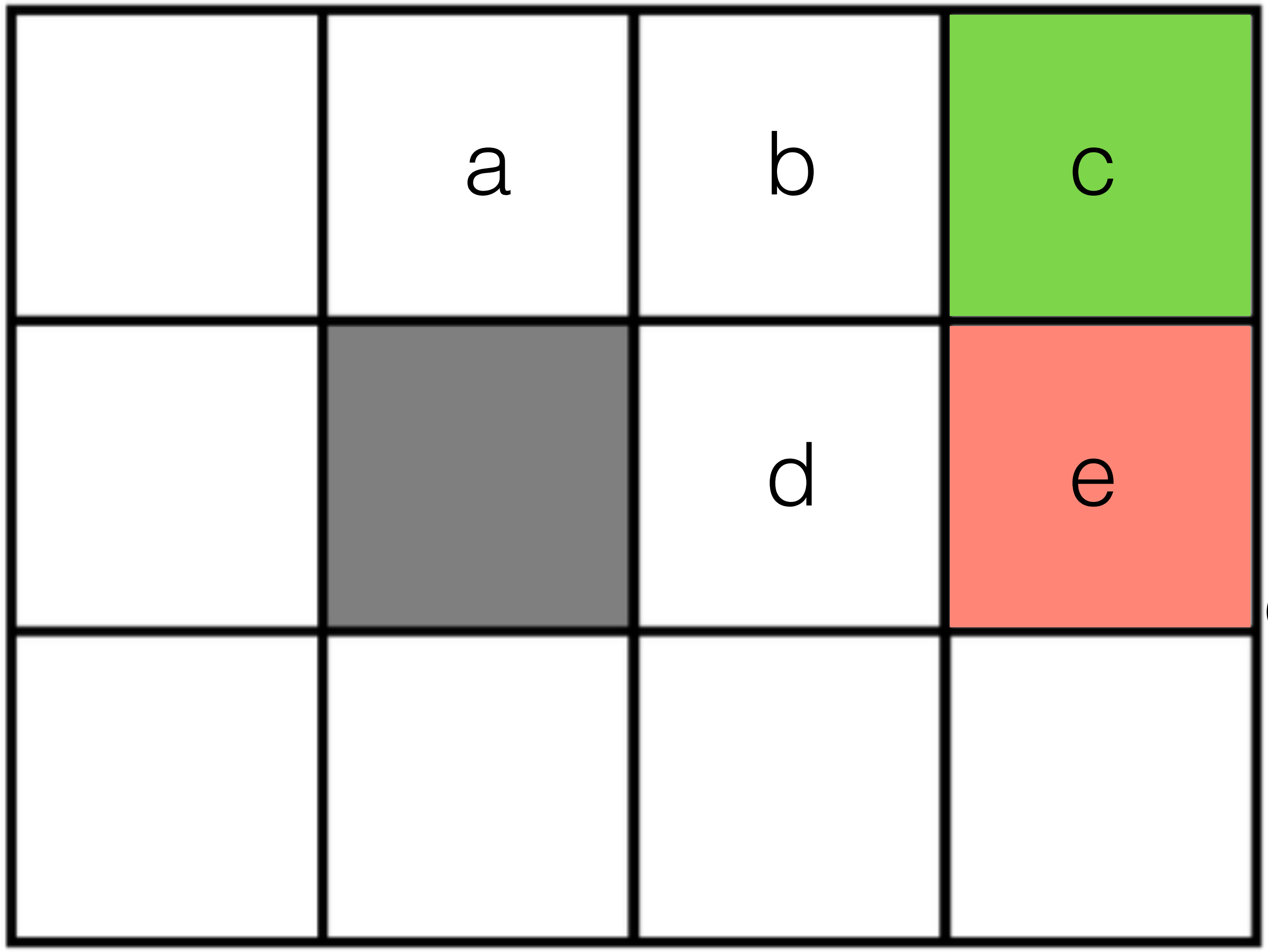
$(d, R, \text{-}1),\ (e, R, \text{-}10)$

$$Q(\mathrm{e}, \mathrm{R}) = r(\mathrm{e})$$

$$Q(\mathrm{b}, \mathrm{R}) = r(\mathrm{b}) + \max_{\mathbf{u}} Q(\mathrm{d}, \mathbf{u})$$

Q	R - right	D - down
a	?	?
b	?	?
c	?	?
d	?	?
e	?	?

Having a trajectory, each transition gives one equation

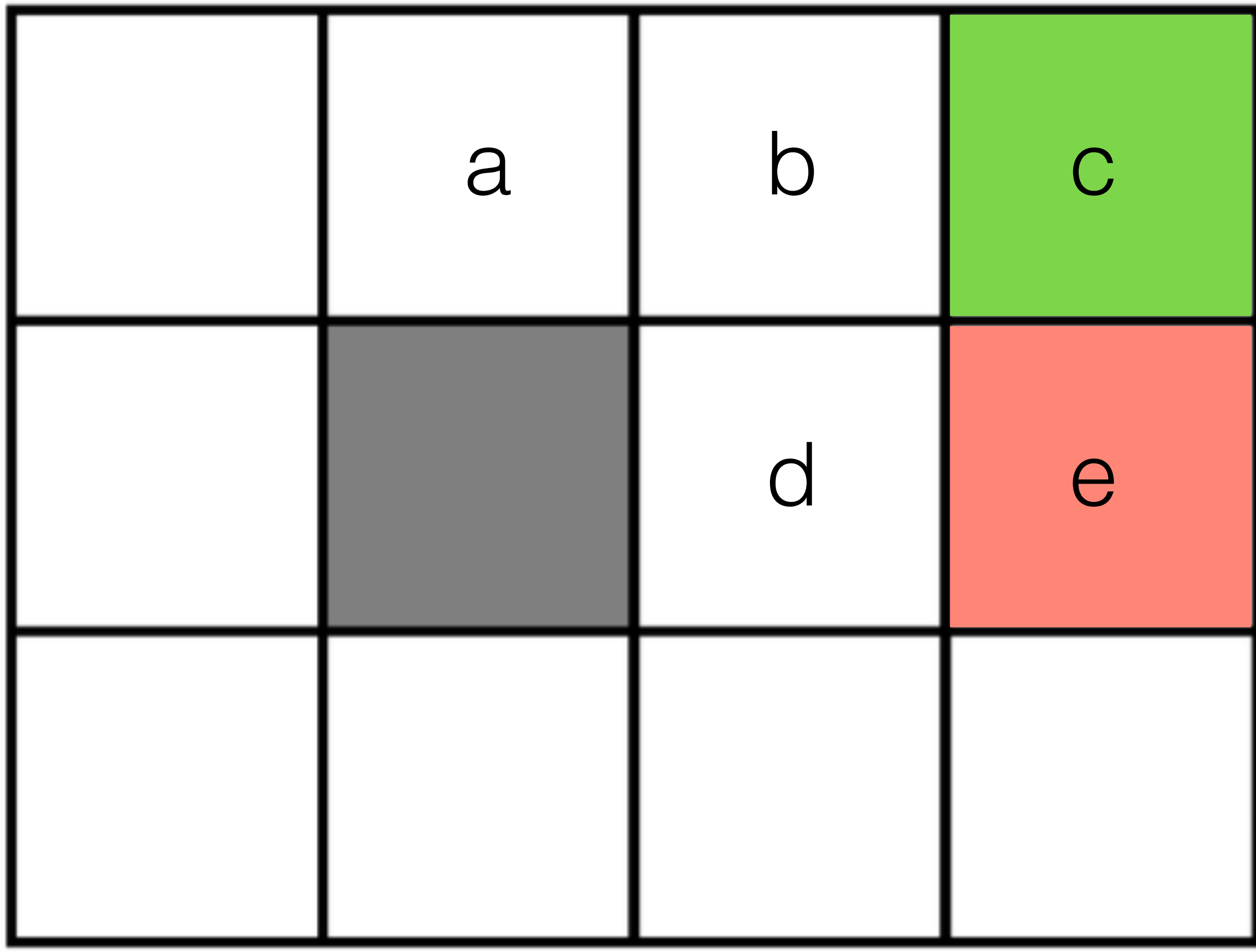


$$\tau_2:$$
$$(a, R, \text{-}1),\ (b, D, \text{-}1),$$
$$(d, R, \text{-}1),\ (e, R, \text{-}10)$$

$$Q(\mathrm{e}, \mathrm{R}) = r(\mathrm{e})$$
$$Q(\mathrm{b}, \mathrm{R}) = r(\mathrm{b}) + \max_{\mathbf{u}} Q(\mathrm{d}, \mathbf{u})$$
$$Q(\mathrm{d}, \mathrm{R}) = r(\mathrm{d}) + \max_{\mathbf{u}} Q(\mathrm{e}, \mathbf{u})$$
$$Q(\mathrm{a}, \mathrm{R}) = r(\mathrm{a}) + \max_{\mathbf{u}} Q(\mathrm{b}, \mathbf{u})$$

Q	R - right	D - down
a	?	?
b	?	?
c	?	?
d	?	?
e	?	?

Having a trajectory, each transition gives one equation



$$\tau_2:$$

$$(a, R, \text{-}1),\ (b, D, \text{-}1),$$

$$(d, R, \text{-}1),\ (e, R, \text{-}10)$$

$$Q(\mathrm{e}, \mathrm{R}) = r(\mathrm{e})$$

$$Q(\mathrm{b}, \mathrm{R}) = r(\mathrm{b}) + \max_{\mathbf{u}} Q(\mathrm{d}, \mathbf{u})$$

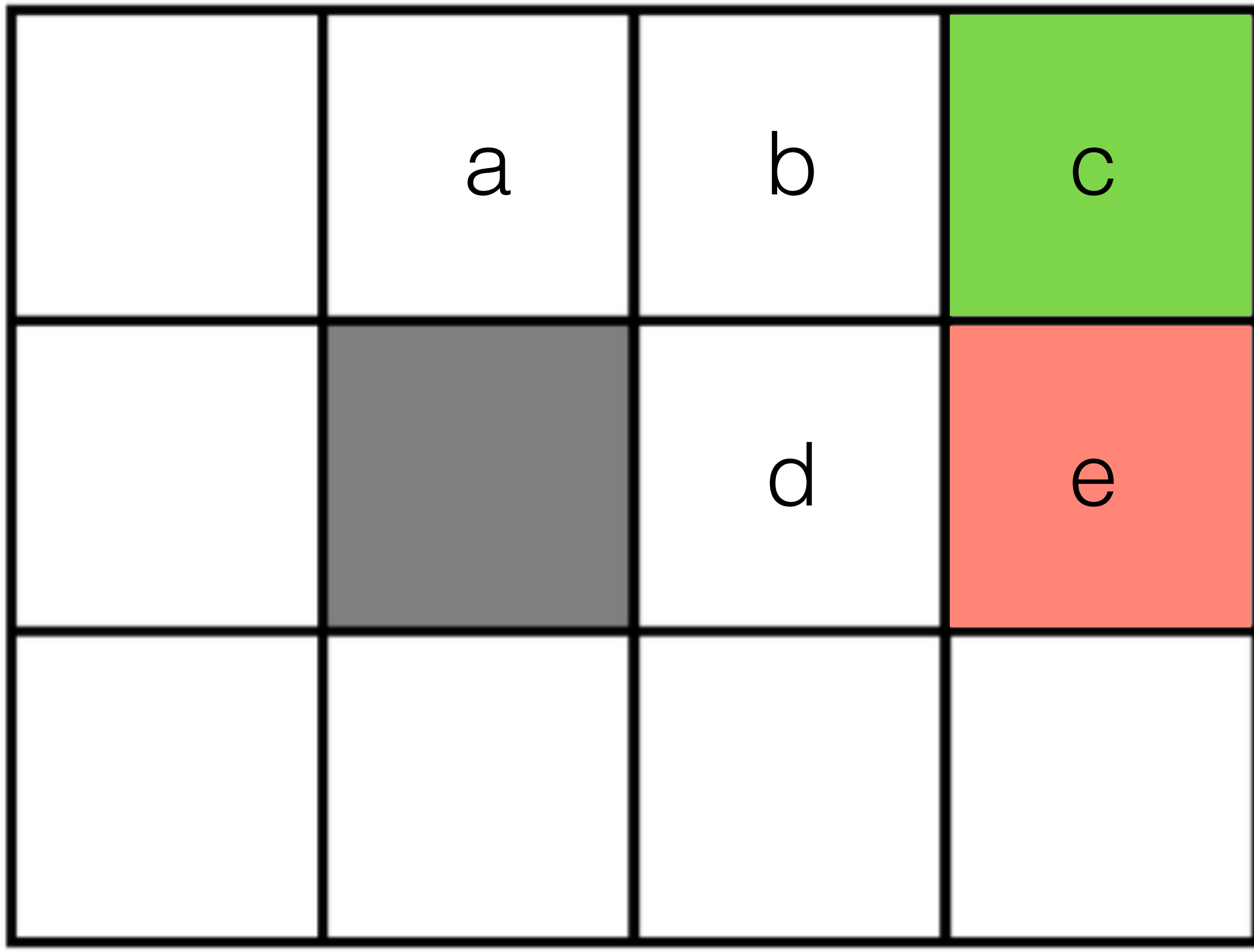
$$Q(\mathrm{d}, \mathrm{R}) = r(\mathrm{d}) + \max_{\mathbf{u}} Q(\mathrm{e}, \mathbf{u})$$

$$Q(\mathrm{a}, \mathrm{R}) = r(\mathrm{a}) + \max_{\mathbf{u}} Q(\mathrm{b}, \mathbf{u})$$

Q	R - right	D - down
a	?	?
b	?	?
c	?	?
d	?	?
e	?	?

unknowns

Having a trajectory, each transition gives one equation



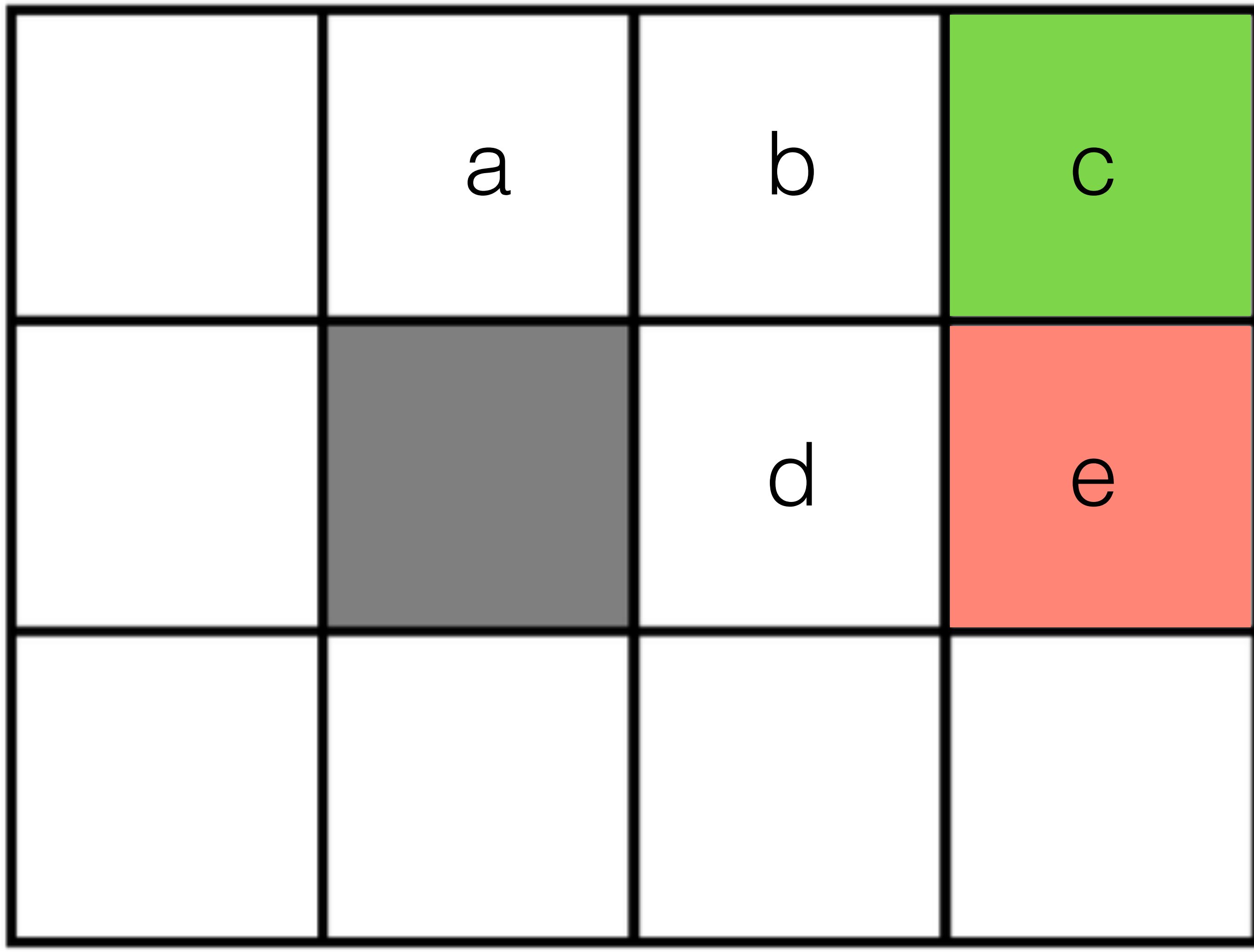
$$\tau_2:$$

$$(a, R, \text{-}1),\ (b, D, \text{-}1),$$
$$(d, R, \text{-}1),\ (e, R, \text{-}10)$$

$$Q(\mathrm{e}, \mathrm{R}) = r(\mathrm{e})$$
$$Q(\mathrm{b}, \mathrm{R}) = r(\mathrm{b}) + \max_{\mathbf{u}} Q(\mathrm{d}, \mathbf{u})$$
$$Q(\mathrm{d}, \mathrm{R}) = r(\mathrm{d}) + \max_{\mathbf{u}} Q(\mathrm{e}, \mathbf{u})$$
$$Q(\mathrm{a}, \mathrm{R}) = r(\mathrm{a}) + \max_{\mathbf{u}} Q(\mathrm{b}, \mathbf{u})$$

(1) Substitute transitions and current Q-values to the right side and solve for left side.

Q	R - right	D - down
a	0	0
b	0	0
c	0	0
d	0	0
e	0	0



τ_2 :

$(a, R, \text{-}1),\ (b, D, \text{-}1),$

$(d, R, \text{-}1),\ (e, R, \text{-}10)$

$$Q(\mathrm{e}, \mathrm{R}) = \text{-}10$$

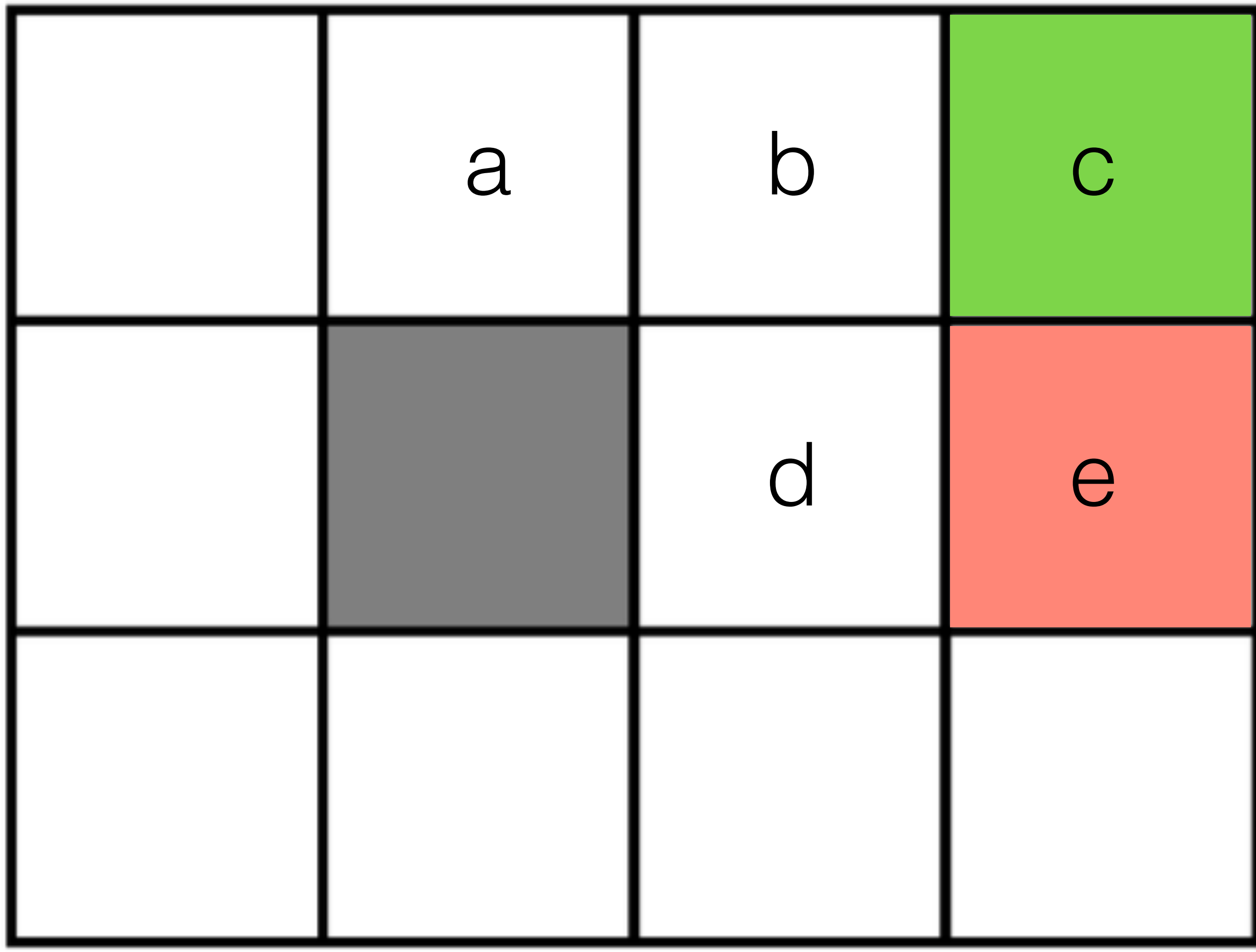
$$Q(\mathrm{b}, \mathrm{R}) = r(\mathrm{b}) + \max_{\mathbf{u}} Q(\mathrm{d}, \mathbf{u})$$

$$Q(\mathrm{d}, \mathrm{R}) = r(\mathrm{d}) + \max_{\mathbf{u}} Q(\mathrm{e}, \mathbf{u})$$

$$Q(\mathrm{a}, \mathrm{R}) = r(\mathrm{a}) + \max_{\mathbf{u}} Q(\mathrm{b}, \mathbf{u})$$

(1) Substitute transitions and current Q-values to the right side and solve for left side.

Q	R - right	D - down
a	0	0
b	0	0
c	0	0
d	0	0
e	-10	0



τ_2 :

$(a, R, \text{-}1),\ (b, D, \text{-}1),$

$(d, R, \text{-}1),\ (e, R, \text{-}10)$

$$Q(\mathrm{e}, \mathrm{R}) = \text{-}10$$

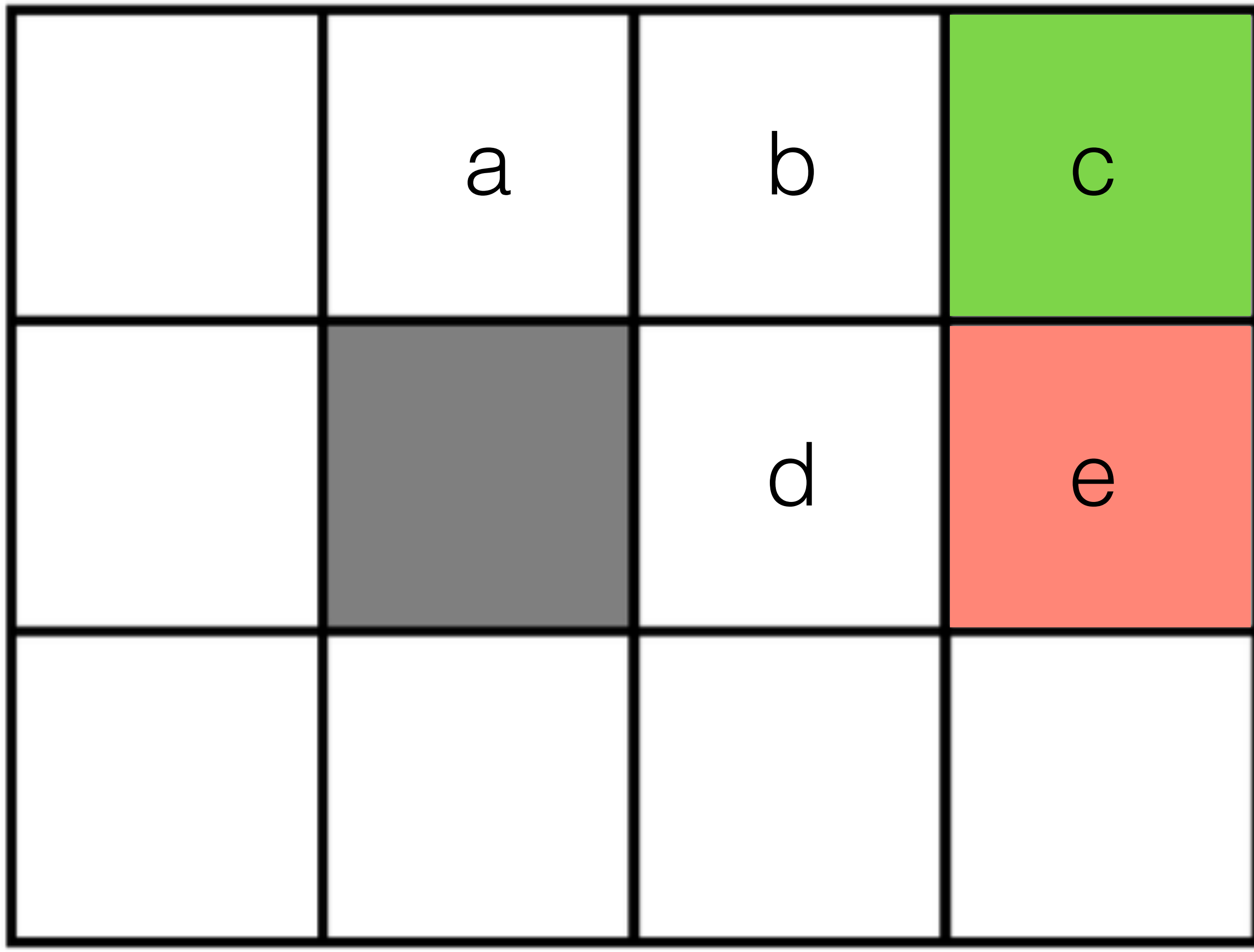
$$Q(\mathrm{b}, \mathrm{R}) = \text{-}1$$

$$Q(\mathrm{d}, \mathrm{R}) = r(\mathrm{d}) + \max_{\mathbf{u}} Q(\mathrm{e}, \mathbf{u})$$

$$Q(\mathrm{a}, \mathrm{R}) = r(\mathrm{a}) + \max_{\mathbf{u}} Q(\mathrm{b}, \mathbf{u})$$

Q	R - right	D - down
a	0	0
b	0	-1
c	0	0
d	0	0
e	-10	0

(1) Substitute transitions and current Q-values to the right side and solve for left side.



τ_2 :

$(a, R, \text{-}1),\ (b, D, \text{-}1),\ (d, R, \text{-}1),\ (e, R, \text{-}10)$

$$Q(\mathrm{e}, \mathrm{R}) = \text{-}10$$

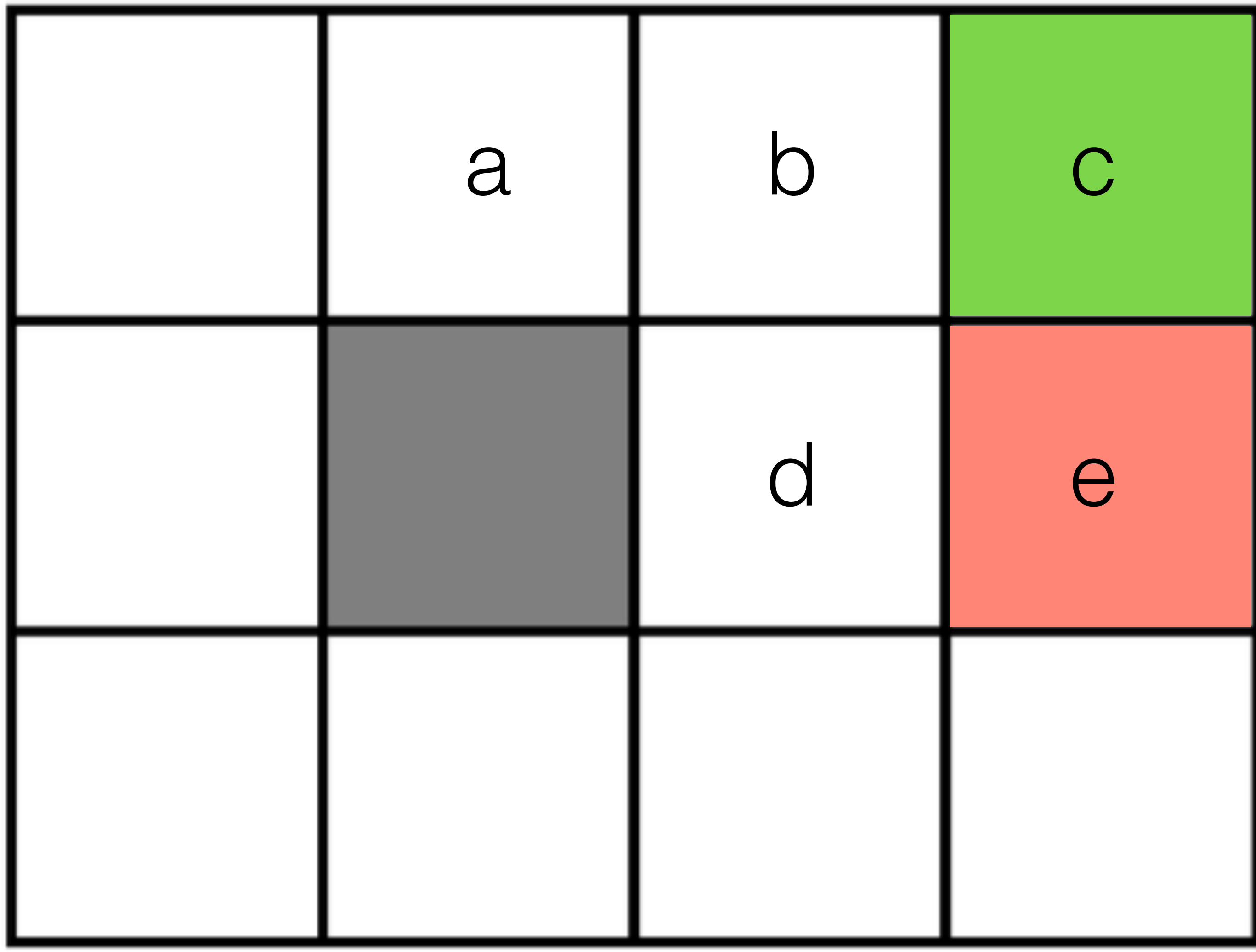
$$Q(\mathrm{b}, \mathrm{R}) = \text{-}1$$

$$Q(\mathrm{d}, \mathrm{R}) = \text{-}1$$

$$Q(\mathrm{a}, \mathrm{R}) = r(\mathrm{a}) + \max_{\mathbf{u}} Q(\mathrm{b}, \mathbf{u})$$

Q	R - right	D - down
a	0	0
b	0	-1
c	0	0
d	0	-1
e	-10	0

(1) Substitute transitions and current Q-values to the right side and solve for left side.



τ_2 :

$(a, R, \text{-}1)$, $(b, D, \text{-}1)$, $(d, R, \text{-}1)$, $(e, R, \text{-}10)$

$Q(\mathrm{e}, \mathrm{R}) =$ -10

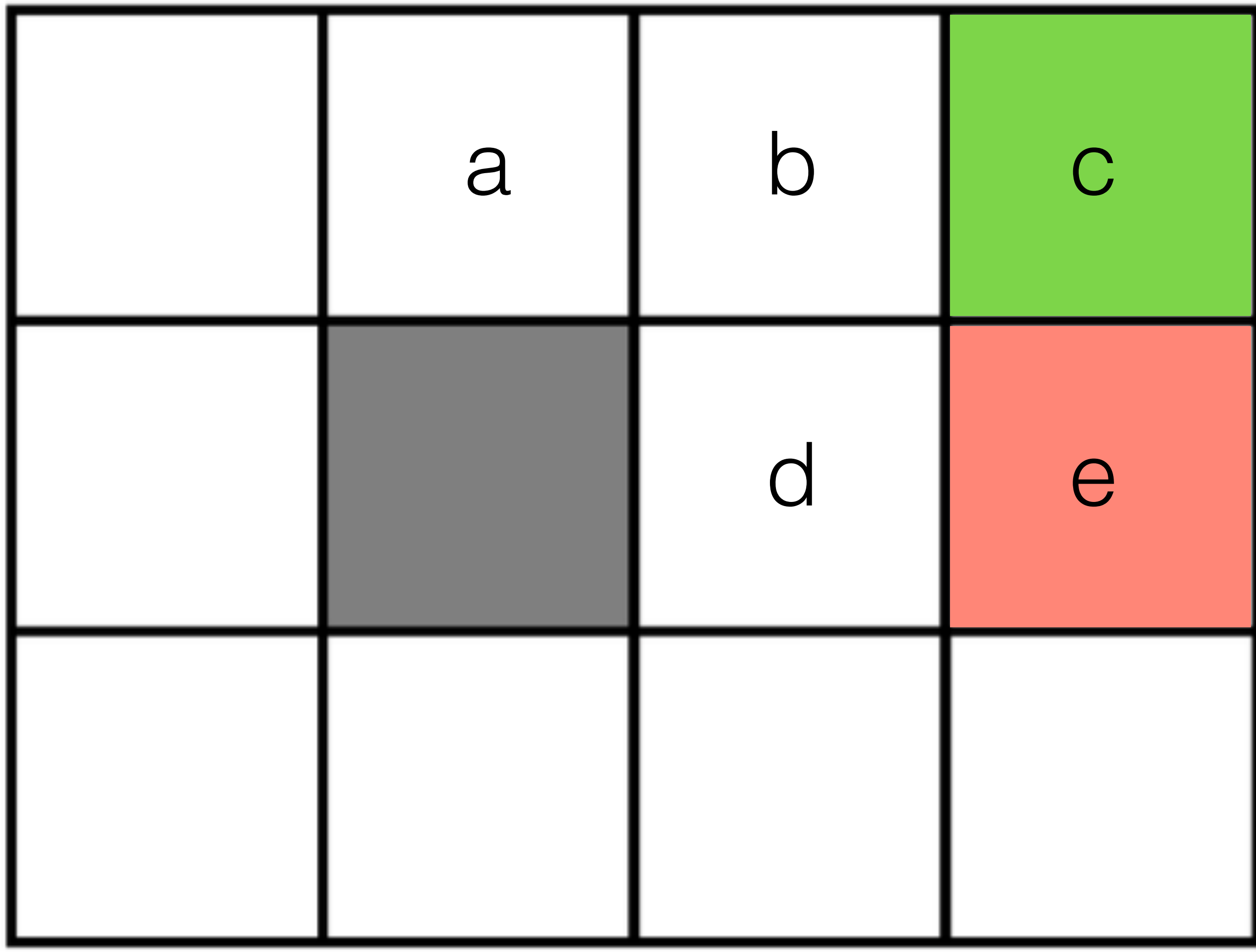
$Q(\mathrm{b}, \mathrm{R}) =$ -1

$Q(\mathrm{d}, \mathrm{R}) =$ -1

$Q(\mathrm{a}, \mathrm{R}) =$ -1

Q	R - right	D - down
a	0	-1
b	0	-1
c	0	0
d	0	-1
e	-10	0

(1) Substitute transitions and current Q-values to the right side and solve for left side.



τ_2 :

$(a, R, \text{-}1),\ (b, D, \text{-}1),$

$(d, R, \text{-}1),\ (e, R, \text{-}10)$

$$Q(\mathrm{e}, \mathrm{R}) = r(\mathrm{e})$$

$$Q(\mathrm{b}, \mathrm{R}) = r(\mathrm{b}) + \max_{\mathbf{u}} Q(\mathrm{d}, \mathbf{u})$$

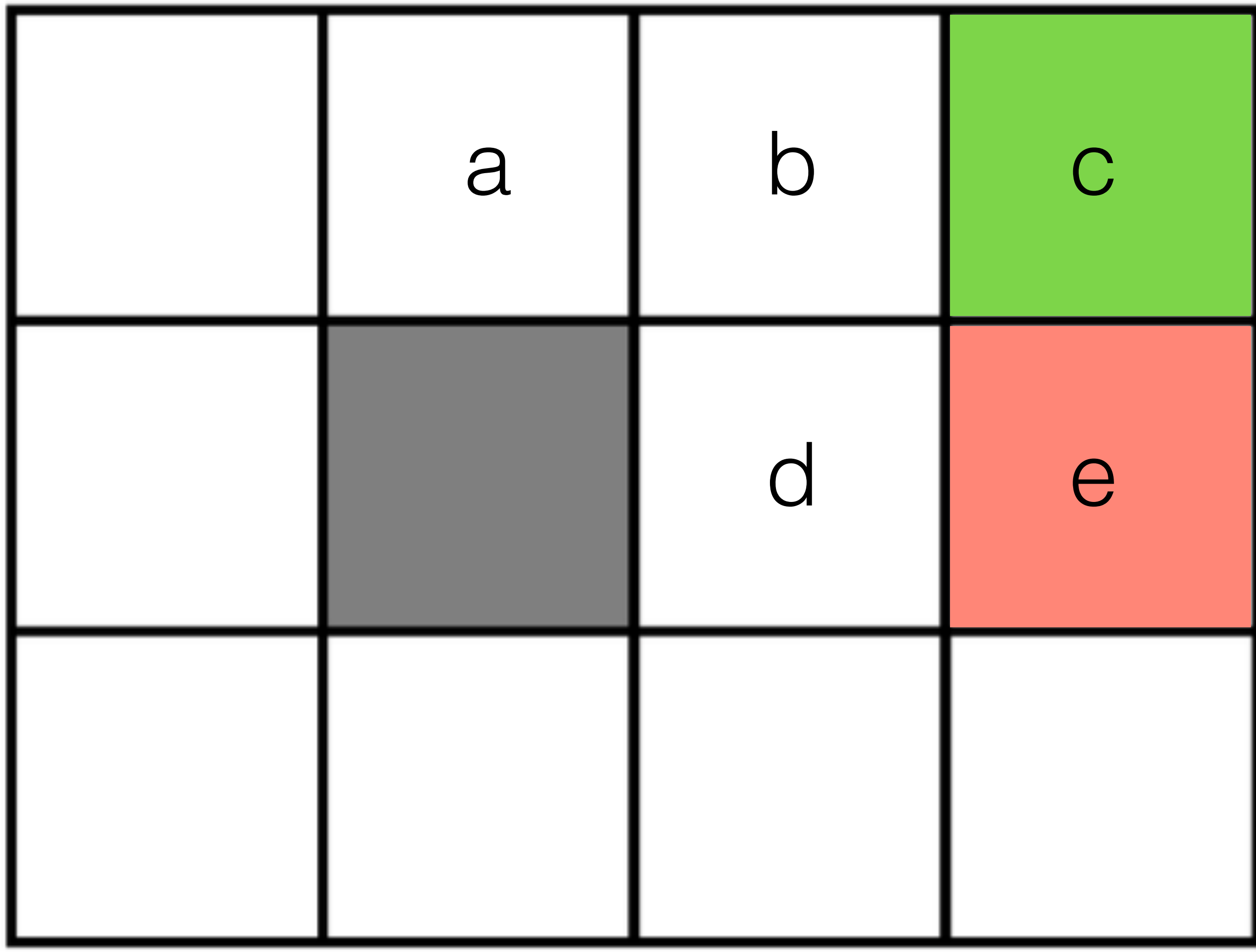
$$Q(\mathrm{d}, \mathrm{R}) = r(\mathrm{d}) + \max_{\mathbf{u}} Q(\mathrm{e}, \mathbf{u})$$

$$Q(\mathrm{a}, \mathrm{R}) = r(\mathrm{a}) + \max_{\mathbf{u}} Q(\mathrm{b}, \mathbf{u})$$

Q	R - right	D - down
a	0	-1
b	0	-1
c	0	0
d	0	-1
e	-10	0

(1) Substitute transitions and current Q-values to the right side and solve for left side.

(2) Repeat several times



τ_2 :

$(a, R, \text{-}1),\ (b, D, \text{-}1),$

$(d, R, \text{-}1),\ (e, R, \text{-}10)$

$$Q(\mathrm{e}, \mathrm{R}) = r(\mathrm{e})$$

$$Q(\mathrm{b}, \mathrm{R}) = r(\mathrm{b}) + \max_{\mathbf{u}} Q(\mathrm{d}, \mathbf{u})$$

$$Q(\mathrm{d}, \mathrm{R}) = r(\mathrm{d}) + \max_{\mathbf{u}} Q(\mathrm{e}, \mathbf{u})$$

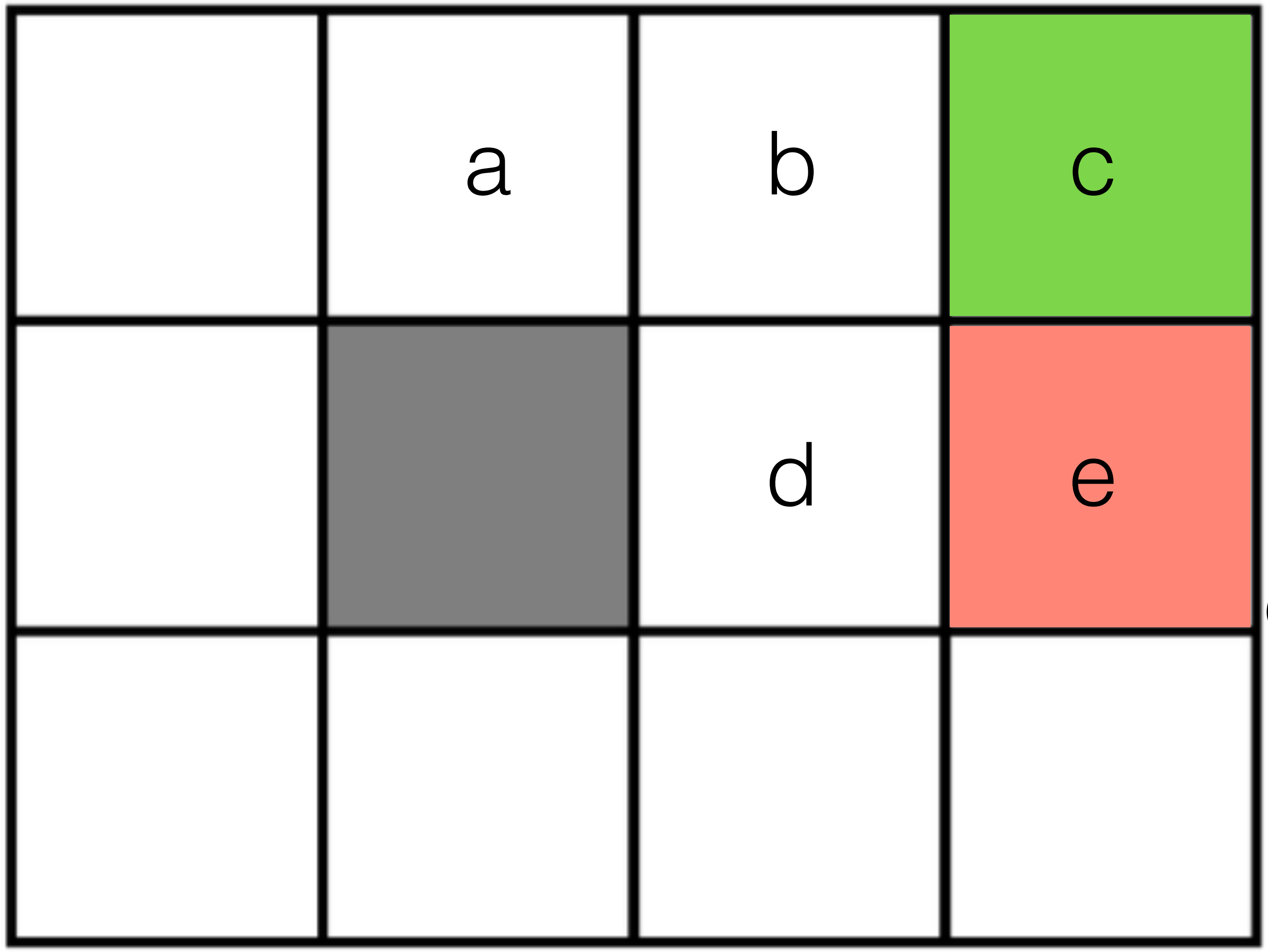
$$Q(\mathrm{a}, \mathrm{R}) = r(\mathrm{a}) + \max_{\mathbf{u}} Q(\mathrm{b}, \mathbf{u})$$

Q	R - right	D - down
a	0	-1
b	0	-1
c	0	0
d	0	-1
e	-10	0

(1) Substitute transitions and current Q-values to the right side and solve for left side.

(2) Repeat several times (search for the fixed point of the Bellman operator)

$$Q = \mathcal{B}(Q)$$



τ_2 :

$(a, R, \text{-}1),\ (b, D, \text{-}1),$

$(d, R, \text{-}1),\ (e, R, \text{-}10)$

$$Q(\mathrm{e}, \mathrm{R}) = r(\mathrm{e})$$

$$Q(\mathrm{b}, \mathrm{R}) = r(\mathrm{b}) + \max_{\mathbf{u}} Q(\mathrm{d}, \mathbf{u})$$

$$Q(\mathrm{d}, \mathrm{R}) = r(\mathrm{d}) + \max_{\mathbf{u}} Q(\mathrm{e}, \mathbf{u})$$

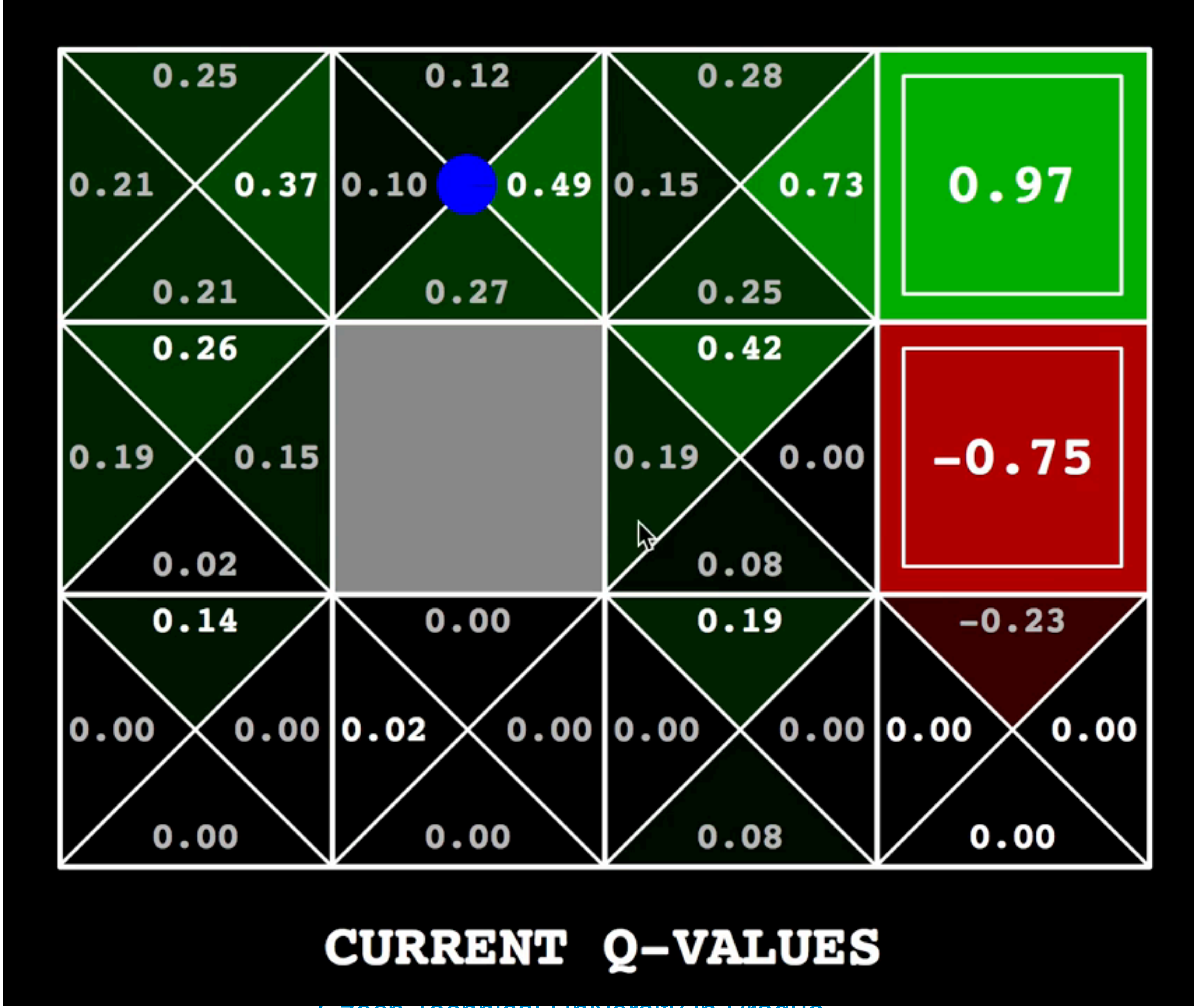
$$Q(\mathrm{a}, \mathrm{R}) = r(\mathrm{a}) + \max_{\mathbf{u}} Q(\mathrm{b}, \mathbf{u})$$

(1) Substitute transitions and current Q-values to the right side and solve for left side.

(2) Repeat several times (search for the fixed point of the Bellman operator)

$$Q = \mathcal{B}(Q)$$

Iterations of the Bellman operator always converge to a fixed point !!!


0.25
0.21
0.37
0.21
0.12
0.10
0.49
0.27
0.28
0.15
0.73
0.25
0.97
0.26
0.19
0.15
0.02
0.42
0.19
0.00
0.08
-0.75
0.14
0.00
0.00
0.00
0.00
0.02
0.00
0.00
0.19
0.00
0.00
0.08
-0.23
0.00
0.00
0.00
CURRENT Q-VALUES

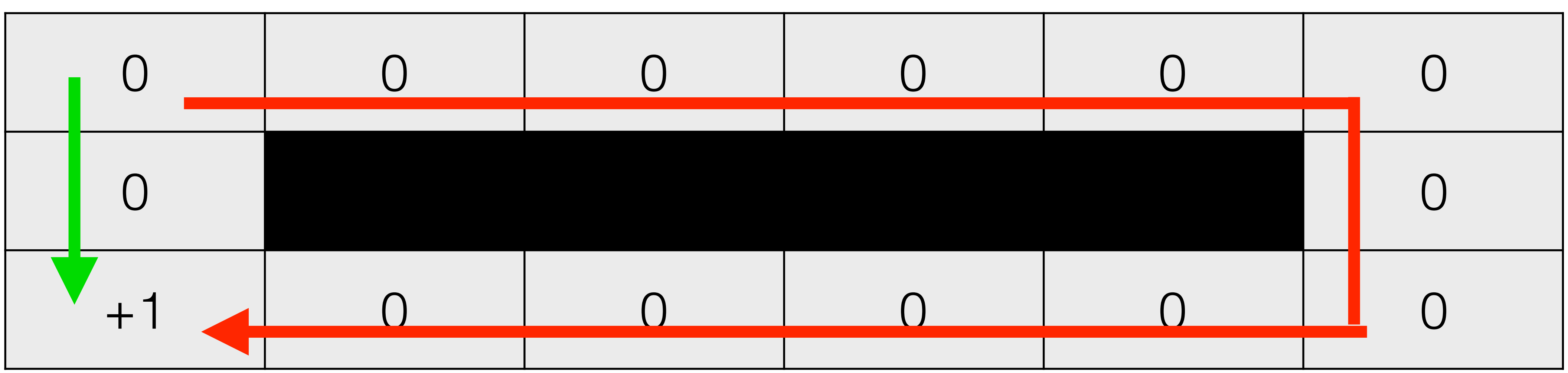
Bellman equation

$$Q(\mathbf{x},\mathbf{u}) = r(\mathbf{x},\mathbf{u},\mathbf{x}') + \max_{\mathbf{u}'} Q(\mathbf{x}',\mathbf{u}')$$

reward for transition

the best you can do from the following state

Which path is better?



Czech Technical University in Prague
Faculty of Electrical Engineering, Department of Cybernetics

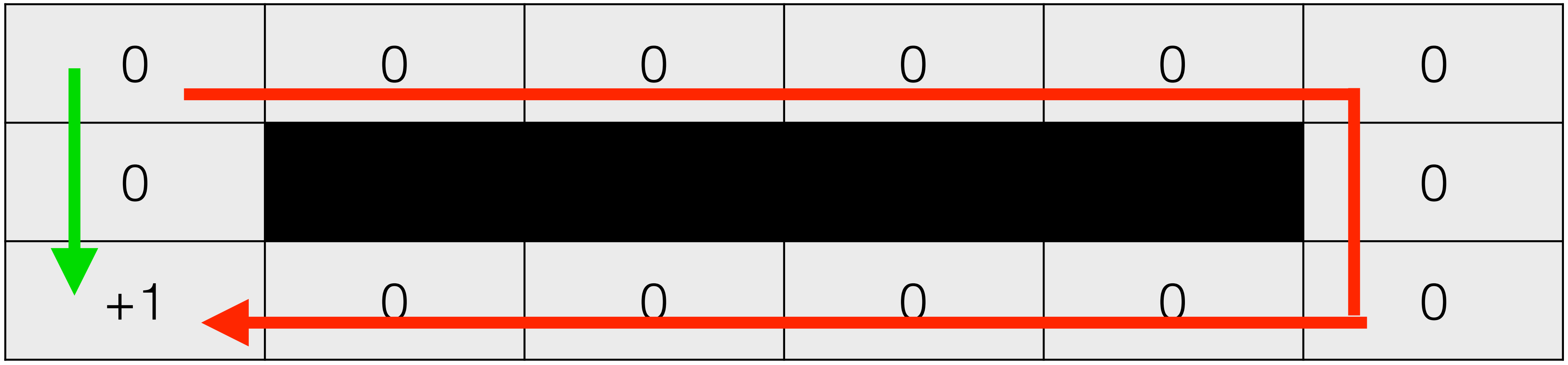
Bellman equation

$$Q(\mathbf{x},\mathbf{u}) = r(\mathbf{x},\mathbf{u},\mathbf{x}') + \gamma \max_{\mathbf{u}'} Q(\mathbf{x}',\mathbf{u}')$$

reward for transition

the best you can do from the following state

discount factor $\gamma \in [0;1]$



Q-learning

1. Collect trajectories $\tau_1, \tau_2, \tau_3, \dots$
2. Solve $Q(\mathbf{x}, \mathbf{u}) = r(\mathbf{x}, \mathbf{u}, \mathbf{x}') + \gamma \max_{\mathbf{u}'} Q(\mathbf{x}', \mathbf{u}')$
3. Repeat from 1

Q-learning

1. Collect trajectories $\tau_1, \tau_2, \tau_3, \ldots$
2. Solve $Q(\mathbf{x}, \mathbf{u}) = r(\mathbf{x}, \mathbf{u}, \mathbf{x}') + \gamma \max_{\mathbf{u}'} Q(\mathbf{x}', \mathbf{u}')$
3. Repeat from 1

- Curse of dimensionality

Q-learning

1. Collect trajectories $\tau_1, \tau_2, \tau_3, \ldots$
2. Solve $Q(\mathbf{x}, \mathbf{u}) = r(\mathbf{x}, \mathbf{u}, \mathbf{x}') + \gamma \max_{\mathbf{u}'} Q(\mathbf{x}', \mathbf{u}')$
3. Repeat from 1

- Curse of dimensionality
- Replace table $Q(\mathbf{x}, \mathbf{u})$ by function $Q_\theta(\mathbf{x}, \mathbf{u})$

Q-learning

1. Collect trajectories $\tau_1, \tau_2, \tau_3, \ldots$
2. Solve $Q(\mathbf{x}, \mathbf{u}) = r(\mathbf{x}, \mathbf{u}, \mathbf{x}') + \gamma \max_{\mathbf{u}'} Q(\mathbf{x}', \mathbf{u}')$
3. Repeat from 1

- Curse of dimensionality
- Replace table $Q(\mathbf{x}, \mathbf{u})$ by function $Q_\theta(\mathbf{x}, \mathbf{u})$

Approximate Q-learning

1. Collect trajectories $\tau_1, \tau_2, \tau_3, \ldots$, initialize $\theta = \text{rand}$
2. Estimate $\mathbf{y} = r(\mathbf{x}, \mathbf{u}, \mathbf{x}') + \gamma \max_{\mathbf{u}'} Q_\theta(\mathbf{x}', \mathbf{u}')$
3. Update parameters by learning

$$\arg\min_\theta \sum_{\mathbf{x},\mathbf{u},\mathbf{y}} \|Q_\theta(\mathbf{x}, \mathbf{u}) - \mathbf{y}\|$$

4. Repeat from 2
5. Repeat from 1

Q-learning

1. Collect trajectories $\tau_1, \tau_2, \tau_3, \dots$
2. Solve $Q(\mathbf{x}, \mathbf{u}) = r(\mathbf{x}, \mathbf{u}, \mathbf{x}') + \gamma \max_{\mathbf{u}'} Q(\mathbf{x}', \mathbf{u}')$
3. Repeat from 1

- Curse of dimensionality
- Replace table $Q(\mathbf{x}, \mathbf{u})$ by function $Q_\theta(\mathbf{x}, \mathbf{u})$

Approximate Q-learning

1. Collect trajectories $\tau_1, \tau_2, \tau_3, \dots$, initialize $\theta = \text{rand}$
2. Estimate $\mathbf{y} = r(\mathbf{x}, \mathbf{u}, \mathbf{x}') + \gamma \max_{\mathbf{u}'} Q_\theta(\mathbf{x}', \mathbf{u}')$
3. Update parameters by learning

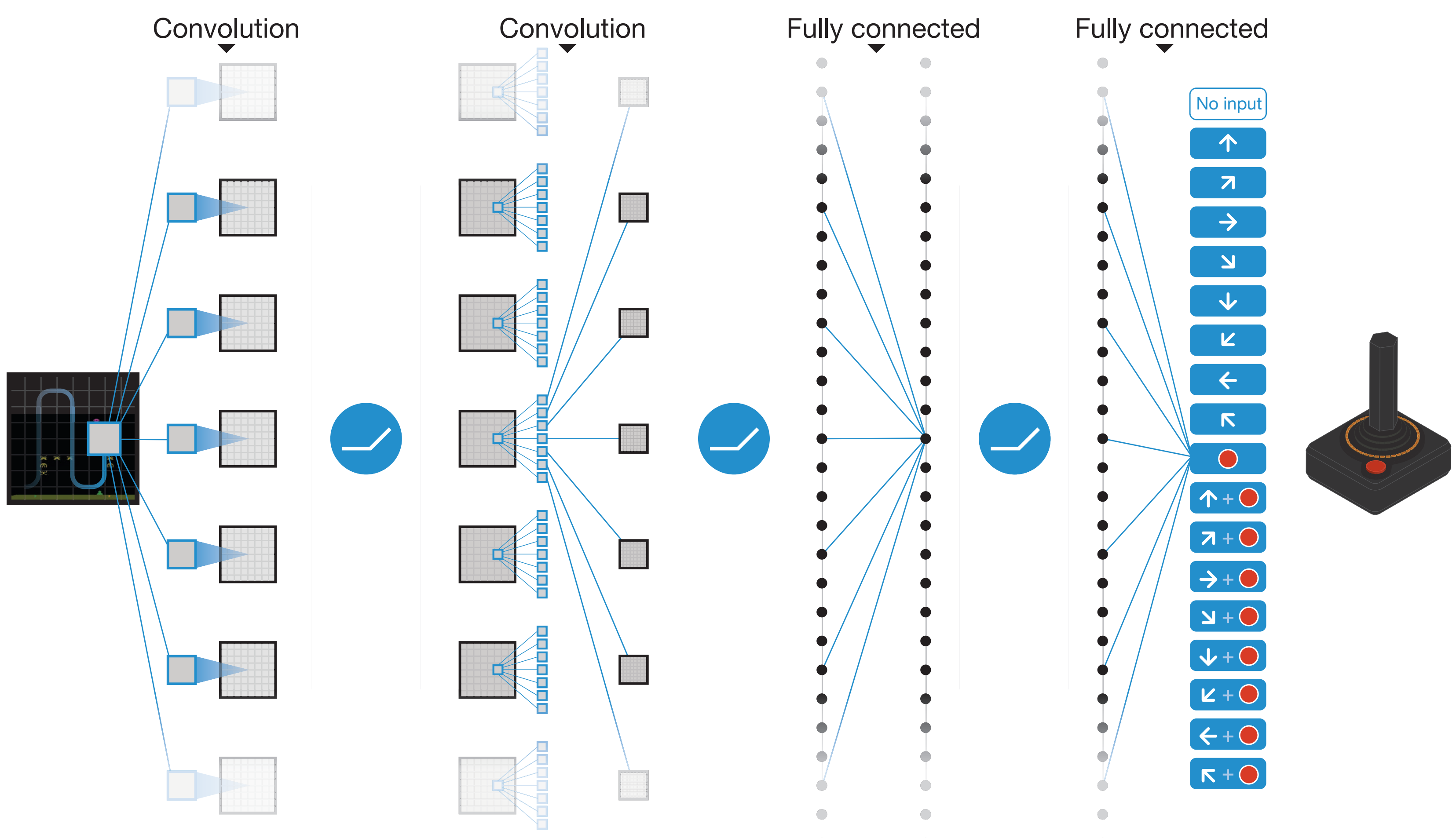
$$\arg\min_\theta \sum_{\mathbf{x},\mathbf{u},\mathbf{y}} \|Q_\theta(\mathbf{x}, \mathbf{u}) - \mathbf{y}\|$$

4. Repeat from 2
5. Repeat from 1

Approximated Q-learning does not have to converges to a fixed-point !!!

Mnih et al. Nature 2015

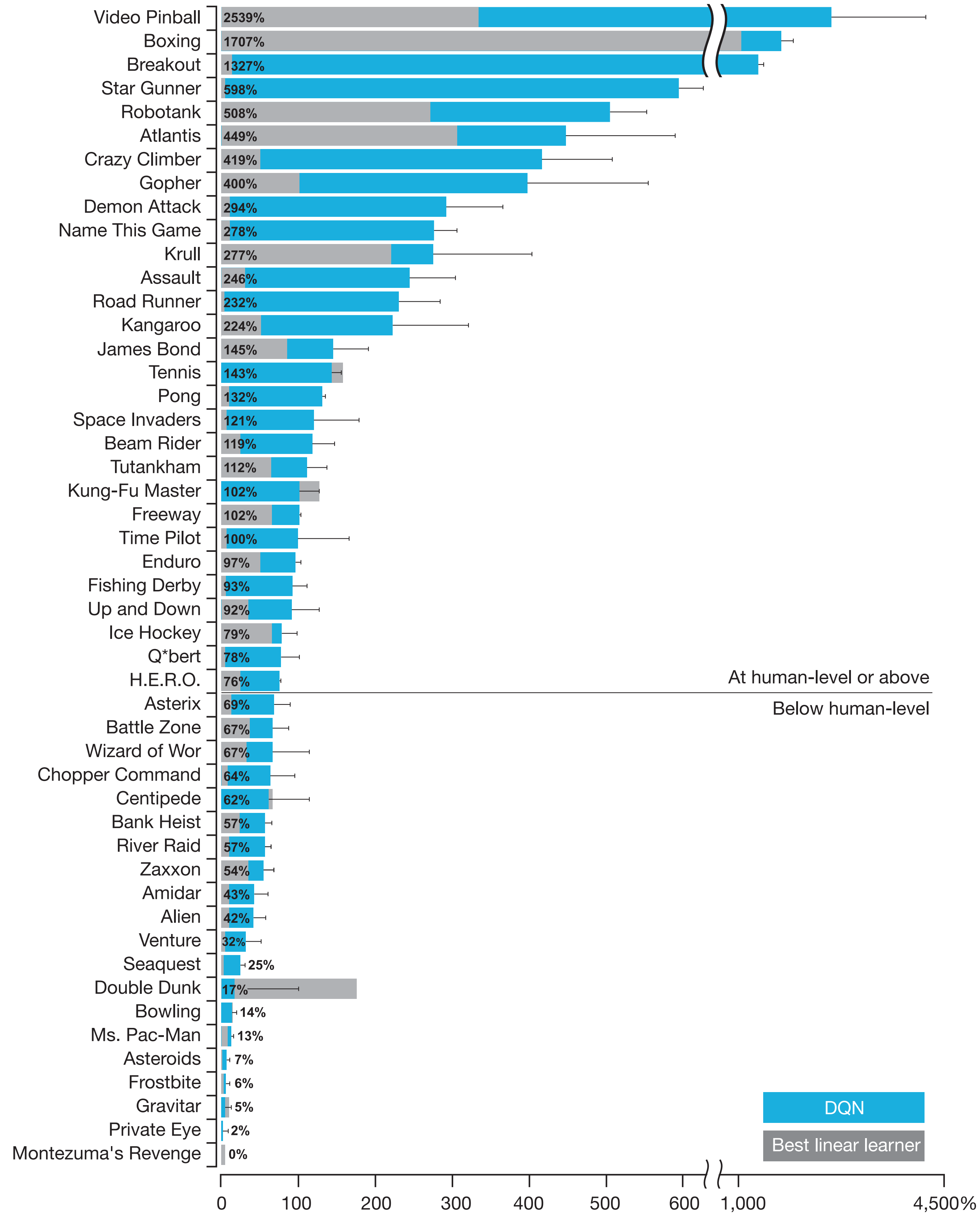
- 2600 atari games
- **state space:** pixels (e.g. VGA resolution)
- **action space:** discrete joystic actions (8 direction + 8 direction with button + neutral action)
- replay buffer (decorrelates samples to be “more i.i.d”)
- two Q-networks (suppress oscilations)



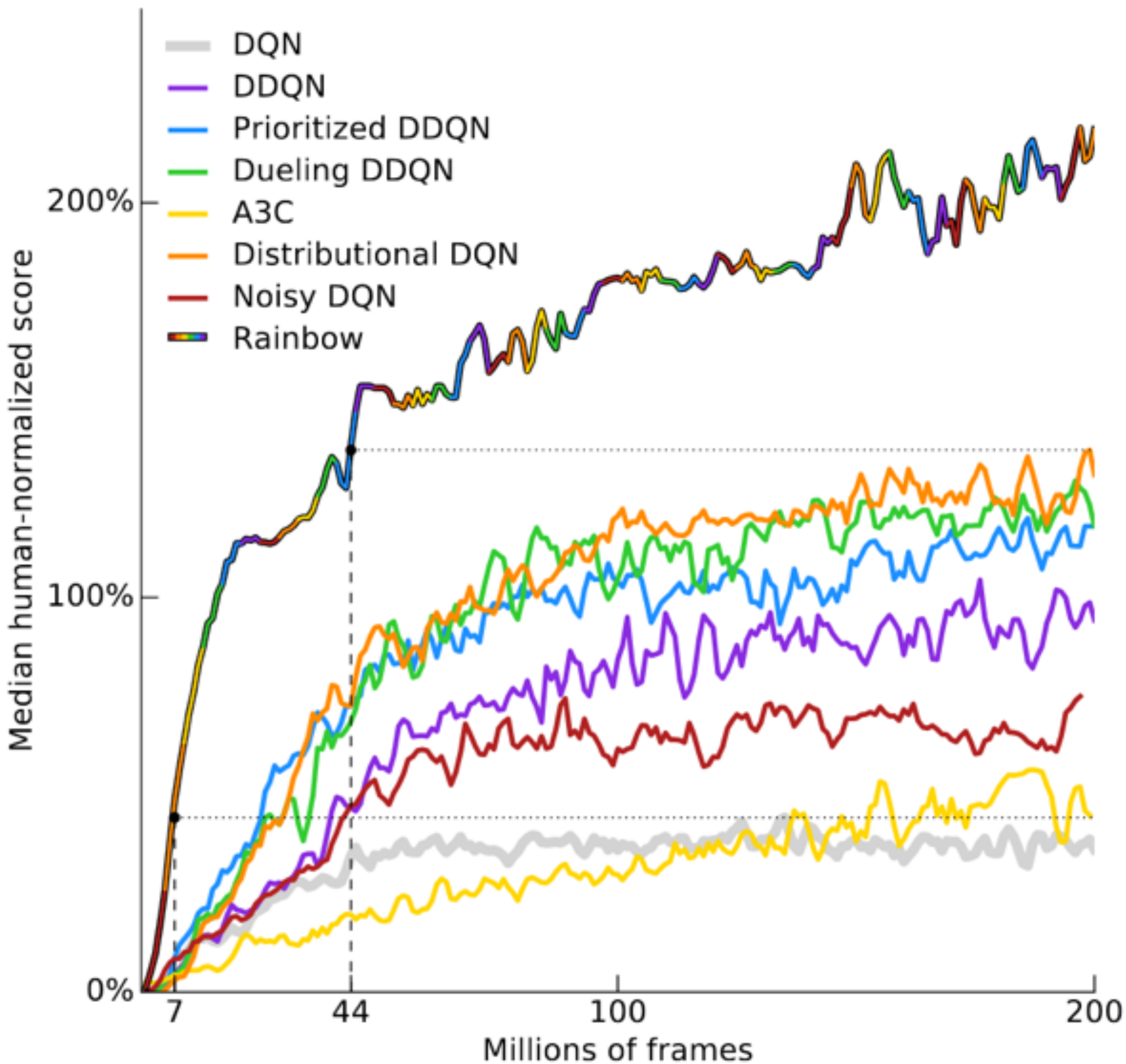
Mnih et al. Nature 2015

- 2600 atari games
- **state space:** pixels (e.g. VGA resolution)
- **action space:** discrete joystic actions (8 directions + 8 directions with button)
- collection of control tasks: https://gym.openai.com

Mnih et al. Nature 2015

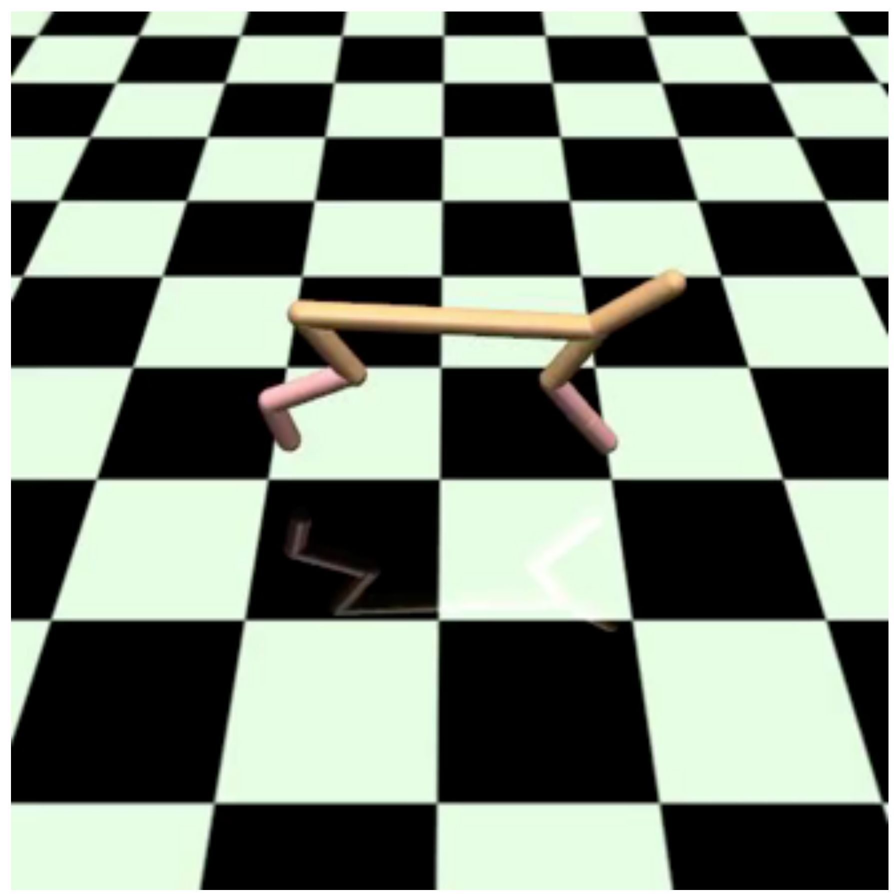


Hessel et. al Rainbow DQN, 2017



Reward shaping

- Sparse rewards are easier to design correctly
- Dense rewards are easier to learn
- Half cheetah:
  - sparse rewards (for reaching the goal position fast)
  - dense rewards (for velocity)



Reward shaping

- Sparse rewards are easier to design correctly
- Dense rewards are easier to learn

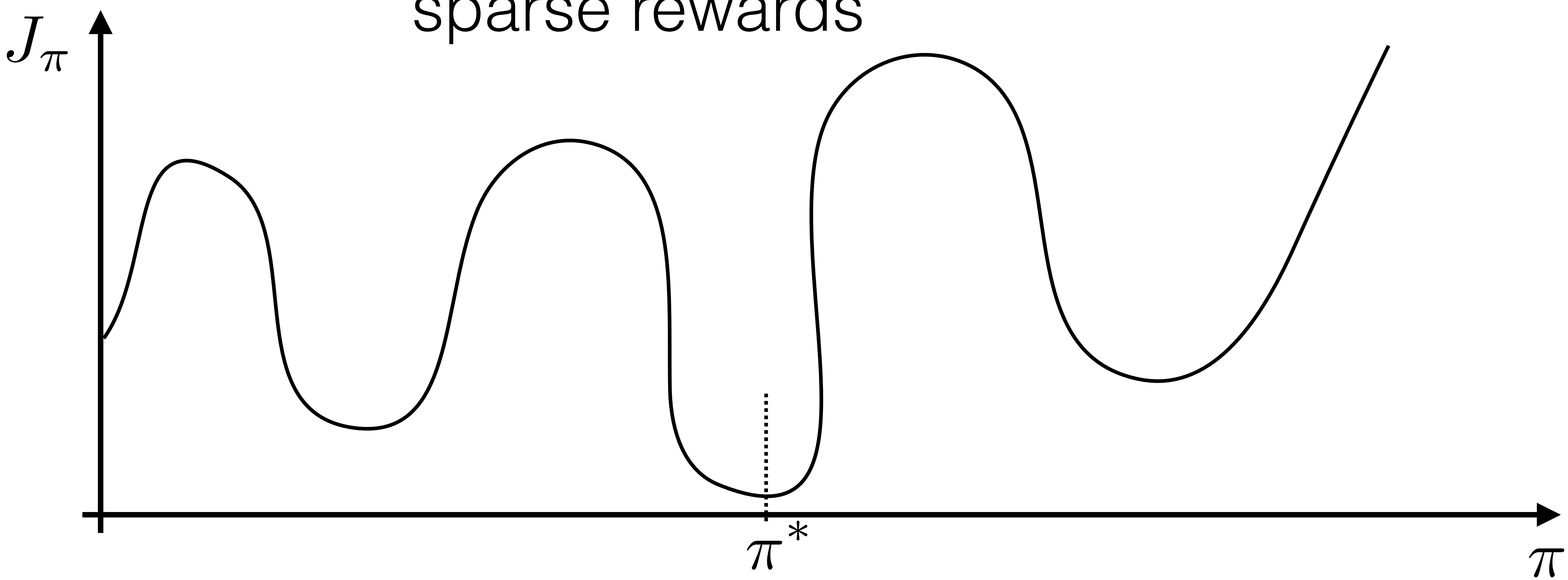
Reward shaping

- Sparse rewards are easier to design correctly
- Dense rewards are easier to learn

Reward shaping

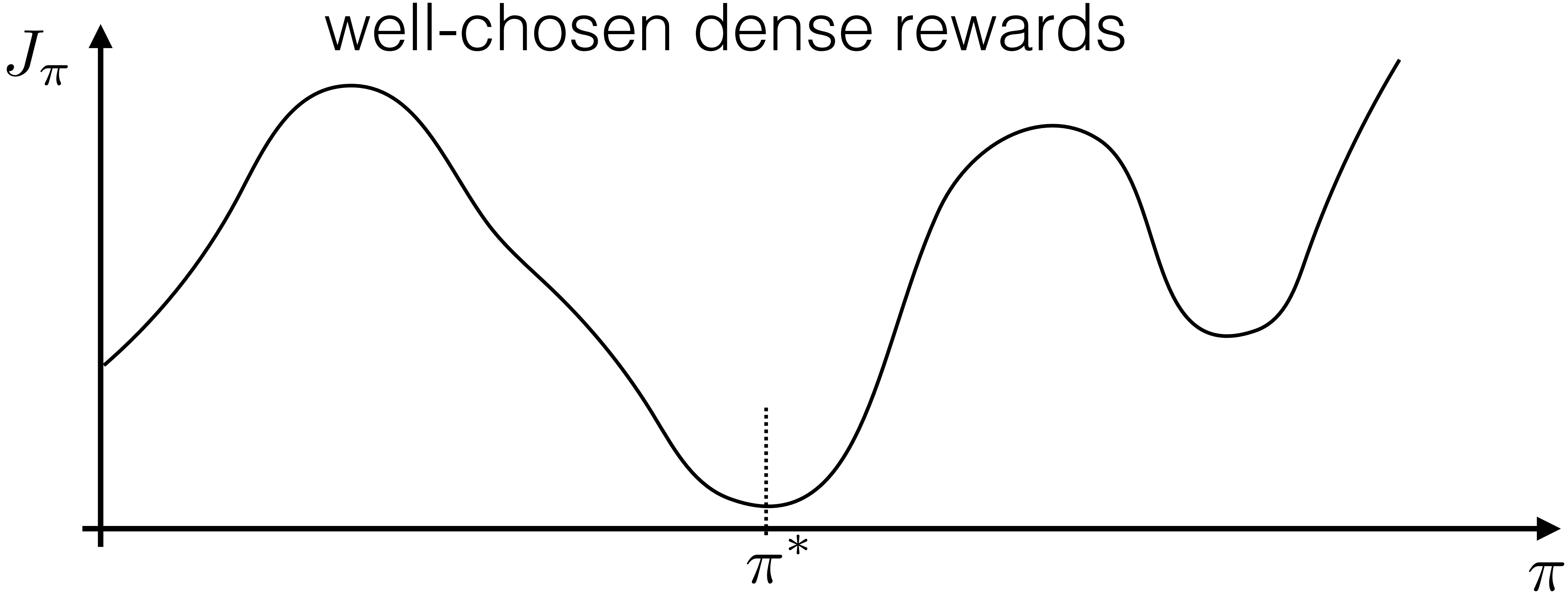
- Sparse rewards are easier to design correctly
- Dense rewards are easier to learn
- Half cheetah:
  - sparse rewards (for reaching the goal position fast)
  - dense rewards (for velocity)

sparse rewards



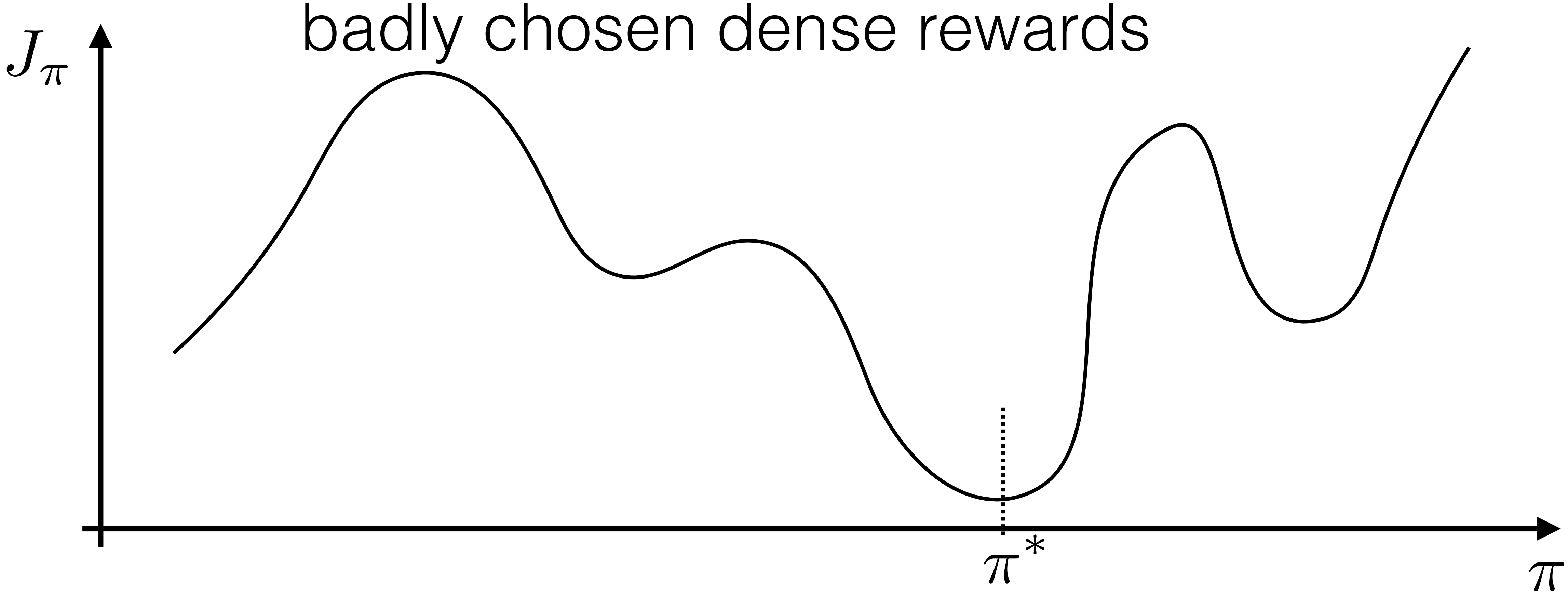
Reward shaping

- Sparse rewards are easier to design correctly
- Dense rewards are easier to learn
- Half cheetah:
  - sparse rewards (for reaching the goal position fast)
  - dense rewards (for velocity)



Reward shaping

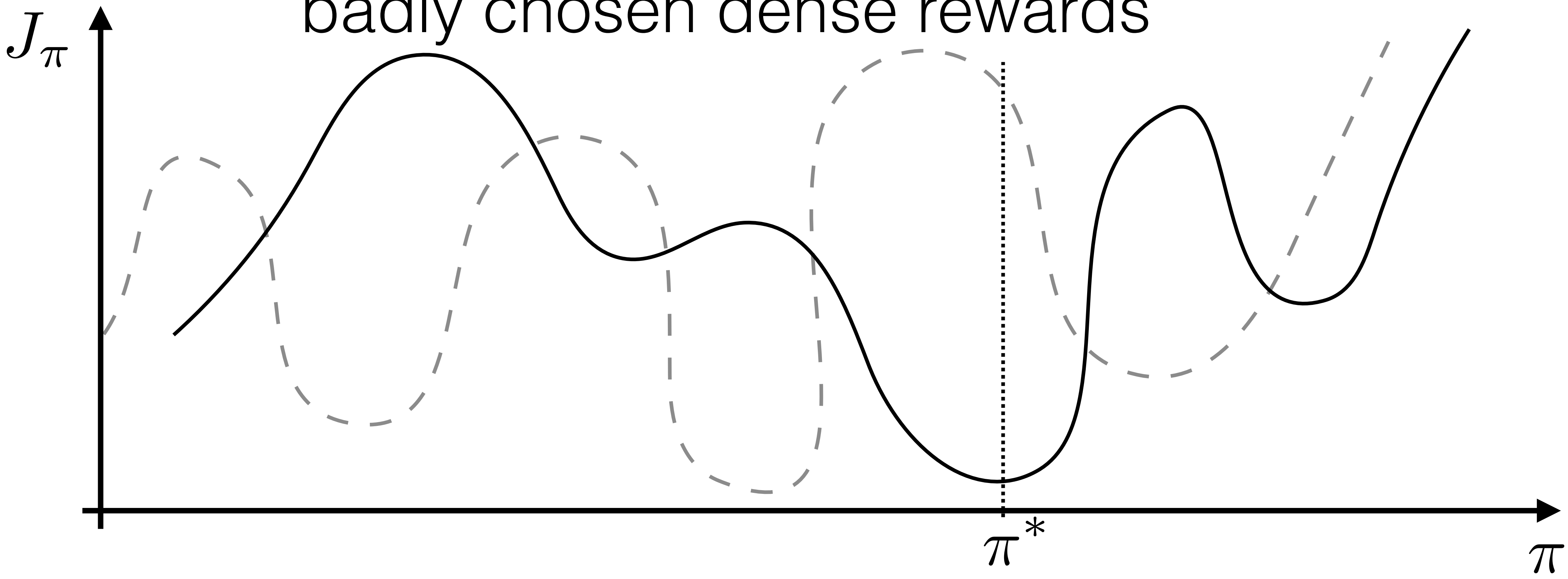
- Sparse rewards are easier to design correctly
- Dense rewards are easier to learn
- Half cheetah:
  - sparse rewards (for reaching the goal position fast)
  - dense rewards (for velocity)



Reward shaping

- Sparse rewards are easier to design correctly
- Dense rewards are easier to learn
- Half cheetah:
  - sparse rewards (for reaching the goal position fast)
  - dense rewards (for velocity)

badly chosen dense rewards



Reward shaping

- Sparse rewards are easier to design correctly
- Dense rewards are easier to learn
- Boat racing (bad dense rewards):
  - sparse rewards (winning the race)
  - dense rewards (collecting powerups, checkpoints …)

Levine

Learning from expert demonstrations

- Sometimes easier to provide good trajectories than good rewards.
- Imitation learning setup

Learning from expert demonstrations

- Sometimes easier to provide good trajectories than good rewards.
- Imitation learning setup
  1. Collect expert trajectories $\tau_1^*, \tau_2^*, \tau_3^*, \ldots$
  2. Find policy $\arg\min_\theta \sum_{(\mathbf{x}_i, \mathbf{a}_i) \in \tau^*} \|\pi_\theta(\mathbf{x}_i) - \mathbf{a}_i\|_2^2$

Learning from expert demonstrations

- Sometimes easier to provide good trajectories than good rewards.

- Imitation learning setup (statistically inconsistent+ blackbox)
  1. Collect expert trajectories $\tau_1^*, \tau_2^*, \tau_3^*, \ldots$
  2. Find policy $\arg\min_\theta \sum_{(\mathbf{x}_i, \mathbf{a}_i) \in \tau^*} \|\pi_\theta(\mathbf{x}_i) - \mathbf{a}_i\|_2^2$

- Inverse reinforcement learning setup

Learning from expert demonstrations

- Sometimes easier to provide good trajectories than good rewards.

- Imitation learning setup (statistically inconsistent+ blackbox)
  1. Collect expert trajectories $\tau_1^*, \tau_2^*, \tau_3^*, \ldots$
  2. Find policy $\arg\min_\theta \sum_{(\mathbf{x}_i, \mathbf{a}_i) \in \tau^*} \|\pi_\theta(\mathbf{x}_i) - \mathbf{a}_i\|_2^2$

- Inverse reinforcement learning setup
  1. Collect expert trajectories $\tau_1^*, \tau_2^*, \tau_3^*, \ldots$
  2. Find reward function $r_\mathbf{w}$

$$\arg\min_\mathbf{w} \|\mathbf{w}\|_2^2$$

$$\text{subject to:} \quad \sum_{(\mathbf{x},\mathbf{u},\mathbf{x}') \in \tau^*} r_\mathbf{w}(\mathbf{x}, \mathbf{u}, \mathbf{x}') \leq \sum_{(\mathbf{x},\mathbf{u},\mathbf{x}') \in \{\mathcal{T} \setminus \tau^*\}} r_\mathbf{w}(\mathbf{x}, \mathbf{u}, \mathbf{x}')$$

Learning from expert demonstrations

- Sometimes easier to provide good trajectories than good rewards.

- Imitation learning setup (statistically inconsistent+ blackbox)
  1. Collect expert trajectories $\tau_1^*, \tau_2^*, \tau_3^*, \dots$
  2. Find policy $\arg\min_\theta \sum_{(\mathbf{x}_i, \mathbf{a}_i) \in \tau^*} \|\pi_\theta(\mathbf{x}_i) - \mathbf{a}_i\|_2^2$

- Inverse reinforcement learning setup
  1. Collect expert trajectories $\tau_1^*, \tau_2^*, \tau_3^*, \dots$
  2. Find reward function $r_\mathbf{w}$

$$\arg\min_\mathbf{w} \|\mathbf{w}\|_2^2$$

$$\text{subject to:} \quad \sum_{(\mathbf{x},\mathbf{u},\mathbf{x}') \in \tau^*} r_\mathbf{w}(\mathbf{x}, \mathbf{u}, \mathbf{x}') \leq \sum_{(\mathbf{x},\mathbf{u},\mathbf{x}') \in \{\mathcal{T} \setminus \tau^*\}} r_\mathbf{w}(\mathbf{x}, \mathbf{u}, \mathbf{x}')$$

  3. Solve underlying RL task

Abbeel et al. IJRR 2010

- inverse reinforcement learning
- **state space:** angular and euclidean position, velocity, acceleration
- **action space:** motor torques
- learning reward function from expert pilot

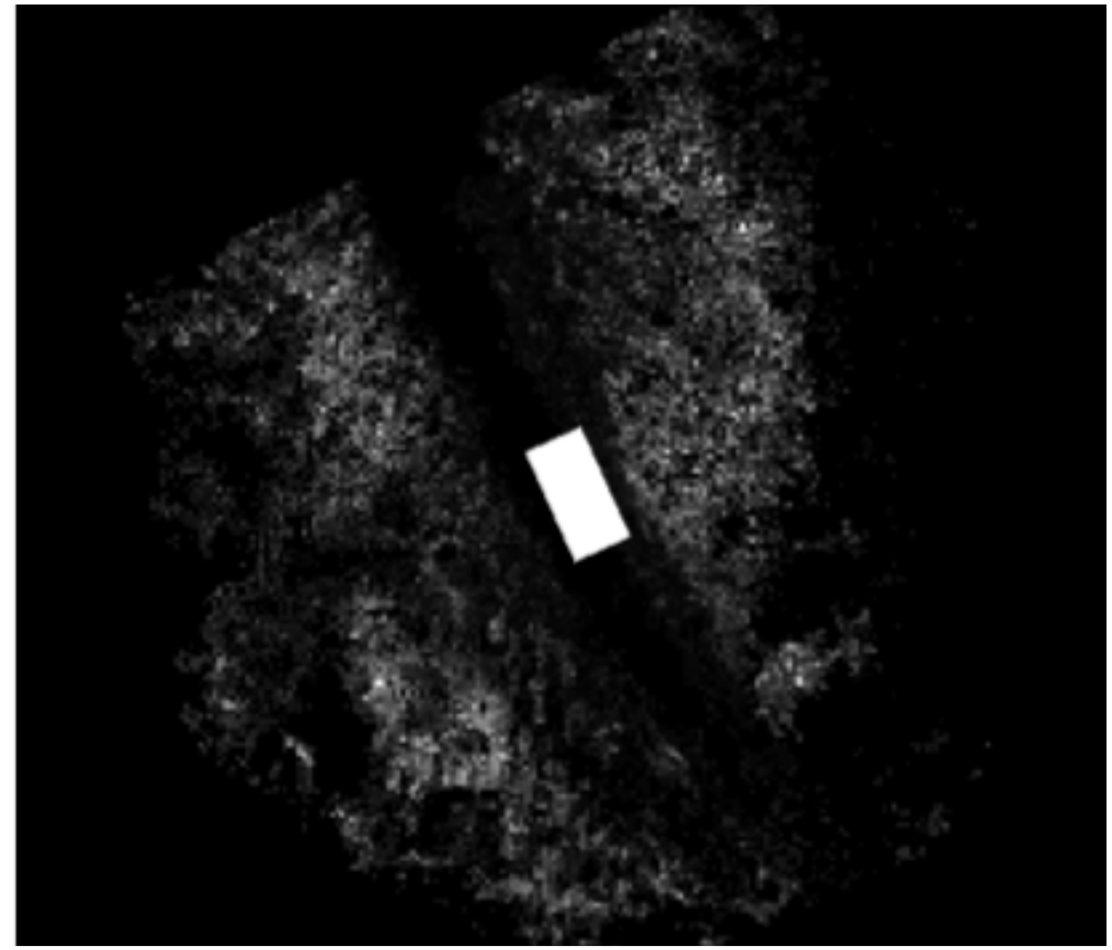
Abbeel et al. IJRR 2010

Silver et al. IJRR 2010

http://www.dtic.mil/dtic/tr/fulltext/u2/a525288.pdf

Silver et al. IJRR 2010

input image (state)



learned reward function (traversability map)

http://www.dtic.mil/dtic/tr/fulltext/u2/a525288.pdf

Taxonomy of policy search methods

- Direct policy search (primal task)

  e.g. gradient ascent for $\pi^* = \arg\max_{\pi} J_{\pi}$

grey zone

Episodic REPS [Peters, 2010]

PILCO [Deisenroth, ICML 2011]

Actor-critic (e.g. DPG [Silver,JMLR 2014])

Deep Q-learning (e.g. [Mnih,Nature 2015])

- Value-based methods (dual function [Kober, 2013])

  e.g. search for $Q(\mathbf{x},\mathbf{a}) = r(\mathbf{x},\mathbf{a},\mathbf{x}') + \gamma \max_{\mathbf{a}'} Q(\mathbf{x}',\mathbf{a}')$

$$\pi^* = \arg\max_{a} Q(\mathbf{x},\mathbf{a})$$

Primal task

1. Randomly initialize policy π_θ

Primal task

1. Randomly initialize policy π_θ
2. Collect trajectories τ with policy π_θ

Primal task

1. Randomly initialize policy π_θ
2. Collect trajectories τ with policy π_θ
3. Denote $p(\tau|\pi_\theta)$ probability of τ occurs when following π_θ

Primal task

1. Randomly initialize policy π_θ
2. Collect trajectories τ with policy π_θ
3. Denote $p(\tau|\pi_\theta)$ probability of τ occurs when following π_θ
4. Define criterion

$$J(\theta) = \mathbb{E}_{\tau \sim p(\tau|\pi_\theta)}\{r(\tau)\} = \int\limits_{\tau \in \mathcal{T}} p(\tau|\pi_\theta) r(\tau)\, \mathrm{d}\tau \approx \frac{1}{N} \sum_{i=1}^{N} r(\tau_i)$$

Primal task

1. Randomly initialize policy π_θ
2. Collect trajectories τ with policy π_θ
3. Denote $p(\tau|\pi_\theta)$ probability of τ occurs when following π_θ
4. Define criterion

$$J(\theta) = \mathbb{E}_{\tau \sim p(\tau|\pi_\theta)}\{r(\tau)\} = \int\limits_{\tau \in \mathcal{T}} p(\tau|\pi_\theta) r(\tau)\, \mathrm{d}\tau \approx \frac{1}{N}\sum_{i=1}^{N} r(\tau_i)$$

5. Optimize criterion (e.g. gradient descent)

$$\theta^* = \arg\min_\theta J(\theta)$$

6. Repeat from 2

Primal task

$$J(\theta) = \mathbb{E}_{\tau \sim p(\tau|\pi_\theta)}\{r(\tau)\} = \int\limits_{\tau \in \mathcal{T}} p(\tau|\pi_\theta) r(\tau)\, \mathrm{d}\tau \approx \frac{1}{N}\sum_{i=1}^{N} r(\tau_i)$$

$$\theta^* = \arg\min_\theta J(\theta)$$

- What do I need for gradient descent optimization? $\frac{\partial J(\theta)}{\partial \theta}^\top$
- Perturb parameters by $\Delta\theta_i$ and estimate $J(\theta + \Delta\theta_i)$

$$J(\theta + \Delta\theta_i) = J(\theta) + \frac{\partial J(\theta)}{\partial \theta}^\top \Delta\theta_i$$

$$\Delta\theta_i^\top \frac{\partial J(\theta)}{\partial \theta} = J(\theta) - J(\theta + \Delta\theta_i)$$

Primal task

$$J(\theta) = \mathbb{E}_{\tau\sim p(\tau|\pi_\theta)}\{r(\tau)\} = \int_{\tau\in\mathcal{T}} p(\tau|\pi_\theta)r(\tau)\,\mathrm{d}\tau \approx \frac{1}{N}\sum_{i=1}^{N} r(\tau_i)$$

$$\theta^* = \arg\min_\theta J(\theta)$$

- What do I need for gradient descent optimization? $\dfrac{\partial J(\theta)}{\partial\theta}^\top$
- Perturb parameters by $\Delta\theta_i$ and estimate $J(\theta+\Delta\theta_i)$

$$J(\theta+\Delta\theta_i) = J(\theta) + \frac{\partial J(\theta)}{\partial\theta}^\top \Delta\theta_i$$

$$\Delta\theta_i^\top \frac{\partial J(\theta)}{\partial\theta} = J(\theta) - J(\theta+\Delta\theta_i)$$

$$\underbrace{\begin{bmatrix}\Delta\theta_1^\top \\ \vdots \\ \Delta\theta_n^\top\end{bmatrix}}_{\text{matrix }\mathtt{A}} \frac{\partial J(\theta)}{\partial\theta} = \underbrace{\begin{bmatrix} J(\theta) - J(\theta+\Delta\theta_1)) \\ \vdots \\ J(\theta) - J(\theta+\Delta\theta_n))\end{bmatrix}}_{\text{vector }\mathbf{b}}$$

Primal task

$$\underbrace{\begin{bmatrix} \Delta\theta_1^\top \\ \vdots \\ \Delta\theta_n^\top \end{bmatrix}}_{\text{matrix } \mathtt{A}} \frac{\partial J(\theta)}{\partial \theta} = \underbrace{\begin{bmatrix} J(\theta) - J(\theta + \Delta\theta_1)) \\ \vdots \\ J(\theta) - J(\theta + \Delta\theta_n)) \end{bmatrix}}_{\text{vector } \mathbf{b}}$$

$$\frac{\partial J(\theta)}{\partial \theta} = \begin{bmatrix} \Delta\theta_1^\top \\ \vdots \\ \Delta\theta_n^\top \end{bmatrix}^+ \cdot \begin{bmatrix} J(\theta) - J(\theta + \Delta\theta_1)) \\ \vdots \\ J(\theta) - J(\theta + \Delta\theta_n)) \end{bmatrix}$$

Primal task

1. Randomly initialize θ

2. Collect trajectories randomly perturbed policy $\pi_{\theta+\Delta\theta_i}$

3. Compute gradient $\frac{\partial J(\theta)}{\partial \theta}^\top$ using pseudo-inverse

$$\frac{\partial J(\theta)}{\partial \theta} = \begin{bmatrix} \Delta\theta_1^\top \\ \vdots \\ \Delta\theta_n^\top \end{bmatrix}^+ \cdot \begin{bmatrix} J(\theta) - J(\theta + \Delta\theta_1)) \\ \vdots \\ J(\theta) - J(\theta + \Delta\theta_n)) \end{bmatrix}$$

4. Update parameters

$$\theta \leftarrow \theta + \alpha \frac{\partial J(\theta)}{\partial \theta}$$

Primal task

REINFORCE: better gradient approximation

- stochastic policy

$$\pi_\theta(\mathbf{u}|\mathbf{x}) : X \times U \rightarrow [0;1]$$

- gradient of the criterion

$$\nabla_\theta J(\theta) = \int_T \nabla_\theta p(\tau|\theta) r(\tau) d\tau$$

- likelihood ratio trick express gradient of the prob distr.

$$\nabla_\theta p(\tau|\theta) = p(\tau|\theta) \nabla_\theta \log p(\tau|\theta)$$

Primal task

- after substitution

$$\nabla_\theta J(\theta) = \int_T p(\tau|\theta)\nabla_\theta \log p(\tau|\theta) r(\tau) d\tau =$$

$$= E[\nabla_\theta \log p(\tau|\theta) r(\tau)] \approx \frac{1}{N}\sum_{i=1}^{N} \nabla_\theta \log p(\tau_i|\theta) r(\tau_i)$$

- where prob distribution simplified using MDP assumption

$$p(\tau|\theta) = p(\mathbf{x_0}) \prod_k p(\mathbf{x}_{k+1}|\mathbf{x}_k, \mathbf{u}_k)\pi_\theta(\mathbf{u}_k|\mathbf{x}_k)$$

$$\nabla_\theta \log p(\tau|\theta) = \nabla_\theta[\, \log p(\mathbf{x_0}) + \sum_k \log p(\mathbf{x}_{k+1}|\mathbf{x}_k, \mathbf{u}_k) + \sum_k \log \pi_\theta(\mathbf{u}_k|\mathbf{x}_k) \,]$$

$$= \sum_k \nabla_\theta \log \pi_\theta(\mathbf{u}_k|\mathbf{x}_k)$$

Primal task

REINFORCE algorithm:

- collect N trajectories

$$\tau_1 = [(\mathbf{u}_{1,1}, \mathbf{x}_{1,1}) \dots \mathbf{u}_{M,1}, \mathbf{x}_{M,1})]$$

$$\vdots$$

$$\tau_N = [(\mathbf{u}_{1,N}, \mathbf{x}_{1,N}) \dots \mathbf{u}_{M,N}, \mathbf{x}_{M,N})]$$

- compute gradient

$$\nabla_\theta J(\theta) = \frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{M} \nabla_\theta \log \pi_\theta(\mathbf{u}_{k,i} | \mathbf{x}_{k,i})$$

- update parameters

$$\theta \leftarrow \theta + \alpha \frac{\partial J(\theta)}{\partial \theta}$$

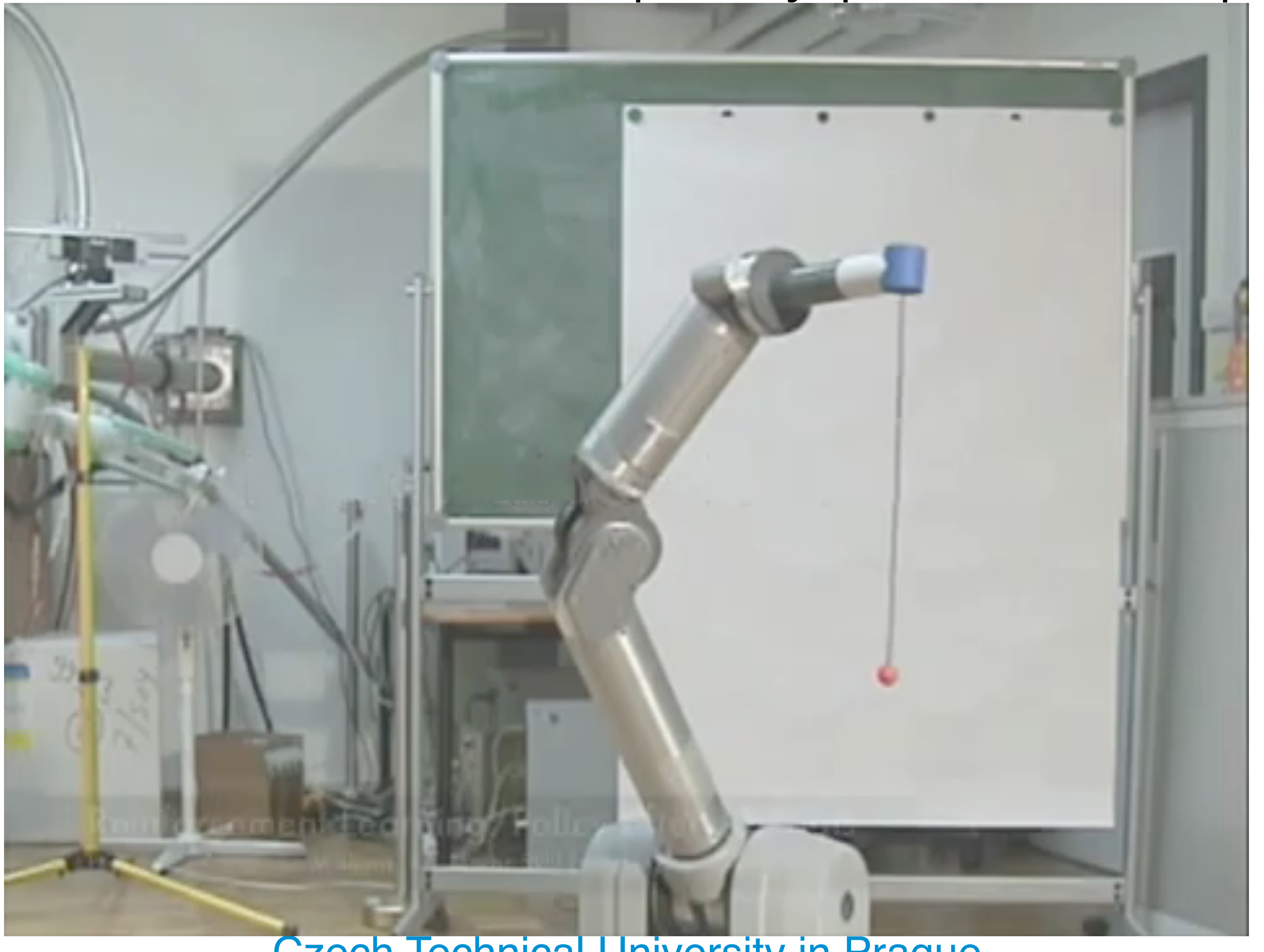
Primal task

- No motion model required
- Converges to local optima (good initialization needed)
- High-dimensional parameters are requires many samples
- Imitation learning from expert trajectories

- There are better gradient approximations [Deisenroth 2013] (e.g. REINFORCE, GPREPS, …)
  [Deisenroth 2013] M. Deisenroth, G. Neumann and J. Peters, A Survey on Policy Search for Robotics, NOW, 2013

Peters et al. NOW 2013

- imitation learning from human demonstration
- **state space:** joint positions, velocities, acceler.
- **action space:** motor torques
- gradient minimization in policy parameter space



Primal task

- No motion model required
- Converges to local optima (good initialization needed)
- High-dimensional parameters are requires many samples
- Imitation learning from expert trajectories

- There are better gradient approximations [Deisenroth 2013] (e.g. REINFORCE, GPREPS, …)
  [Deisenroth 2013] M. Deisenroth, G. Neumann and J. Peters, A Survey on Policy Search for Robotics, NOW, 2013

- If motion model is available then trajectory optimization
  [Tassa 2013] Tassa, Synthesis and Stabilization of Complex Behaviors through Online Trajectory Optimization, IROS2013

Taxonomy of policy search methods

- Direct policy search (primal task)

  e.g. gradient ascent for $\pi^* = \arg\max_{\pi} J_{\pi}$

grey zone

Episodic REPS [Peters, 2010]

PILCO [Deisenroth, ICML 2011]

Actor-critic (e.g. DPG [Silver,JMLR 2014])

Deep Q-learning (e.g. [Mnih,Nature 2015])

- Value-based methods (dual function [Kober, 2013])

  e.g. search for $Q(\mathbf{x}, \mathbf{a}) = r(\mathbf{x}, \mathbf{a}, \mathbf{x}') + \gamma \max_{\mathbf{a}'} Q(\mathbf{x}', \mathbf{a}')$

$$\pi^* = \arg\max_{a} Q(\mathbf{x}, \mathbf{a})$$

Actor-critic methods

1. Collect trajectories $\tau_1, \tau_2, \tau_3, \ldots$, initialize $\theta = \text{rand}$
2. Estimate $\mathbf{y} = r(\mathbf{x}, \mathbf{u}, \mathbf{x}') + \gamma \max_{\mathbf{u}'} Q_\theta(\mathbf{x}', \mathbf{u}')$
3. Update parameters by learning

$$\arg\min_\theta \sum_{\mathbf{x},\mathbf{u},\mathbf{y}} \|Q_\theta(\mathbf{x}, \mathbf{u}) - \mathbf{y}\|$$

Approximated Q-learning

Actor-critic methods

1. Collect trajectories $\tau_1, \tau_2, \tau_3, \ldots$, initialize $\theta = \text{rand}$
2. Estimate $\mathbf{y} = r(\mathbf{x}, \mathbf{u}, \mathbf{x}') + \gamma \max_{\mathbf{u}'} Q_\theta(\mathbf{x}', \mathbf{u}')$
3. Update parameters by learning

$$\arg\min_{\theta} \sum_{\mathbf{x},\mathbf{u},\mathbf{y}} \|Q_\theta(\mathbf{x}, \mathbf{u}) - \mathbf{y}\|$$

4. Learn policy π_ω which do actions maximizing the state-action value function on the collected trajectories

$$\arg\max_{\omega} \sum_{\mathbf{x}\in\tau} Q_\theta(\mathbf{x}, \pi_\omega(\mathbf{x}))$$

Direct policy optimization on Q

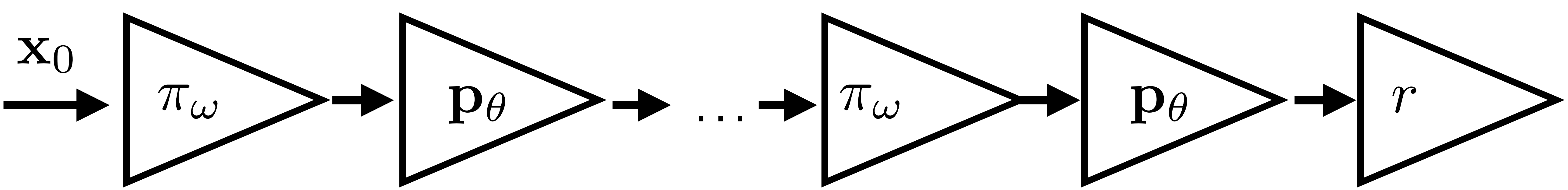
Unrolling in time

1. Collect trajectories $\tau_1, \tau_2, \tau_3, \dots,$ ini: $\theta = \text{rand}, \omega = \text{rand}$
2. Estimate motion model

$$\arg\min_{\theta} \sum_{(\mathbf{x},\mathbf{x}')\in\tau^*} \|\mathbf{p}_\theta(\mathbf{x}) - \mathbf{x}'\|_2^2$$

3. Learn policy maximizing the rewards on model-based trajectories

$$\arg\max_{\omega} \sum_{\mathbf{x}_0} r(\mathbf{p}_\theta(\ \dots\ \pi_\omega(\mathbf{p}_\theta(\mathbf{x}_0, \pi_\omega(\mathbf{x}_0)))))$$



- penalizing distance from training trajectories

[Tassa IROS 2013] https://homes.cs.washington.edu/~todorov/papers/TassaIROS12.pdf

3D humanoid

Degrees-of-freedom: **22**

- 6 spatial
- 2 abdomen
- 2·2 shoulders
- 2·1 elbows
- 2·2 hips
- 2·1 knees
- 2·1 ankles

Control dimensions: **16** all joints

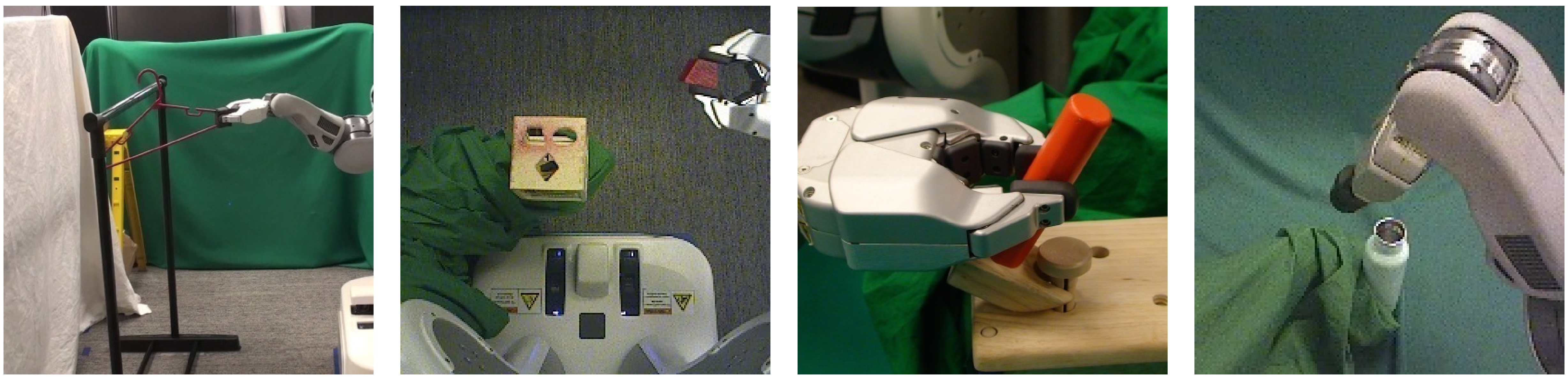
Cost: [CoM over mean of feet, (in xy)] + [torso over CoM (in xy)] + [torso 1.3m over mean of feet (in z)]

\+ [minimize horizontal torso velocity] + [minimize actuation]

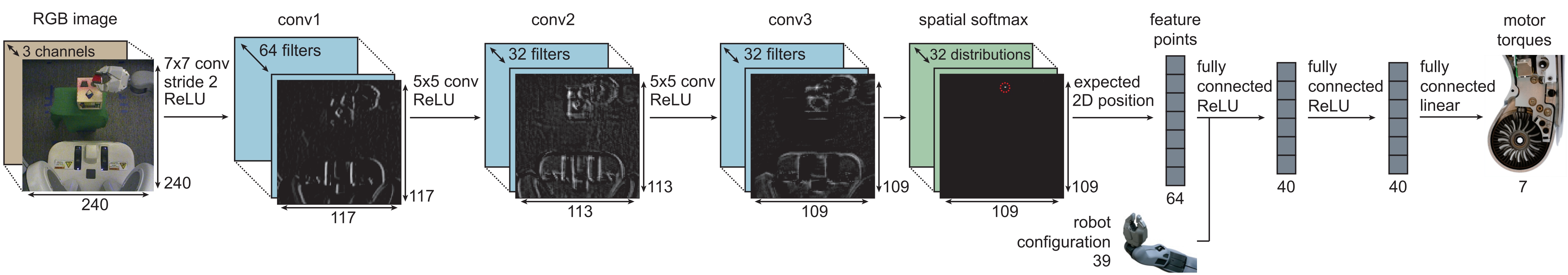
[Heess 2017] https://arxiv.org/abs/1707.02286

Levine et al JMLR 2016

- guides policy gradient method by optimal trajectories
- **state space:** RGB camera images
- **action space:** motor torques

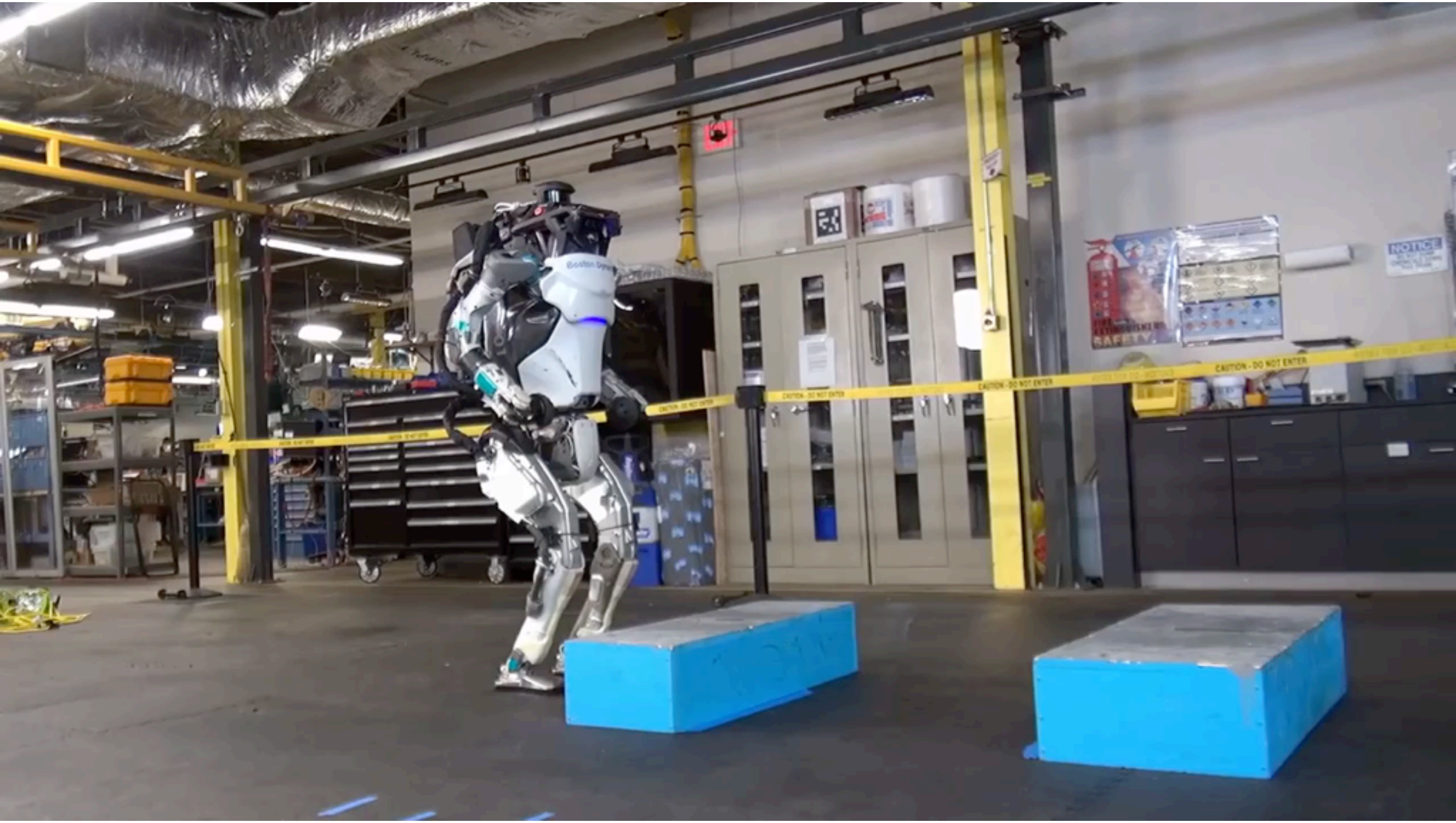


(a) hanger (b) cube (c) hammer (d) bottle



Levine et al JMLR 2016

Boston dynamics - Atlas - NO RL AT ALL



Boston dynamics - Big dog - NO RL AT ALL

Known RL successes

- AlphaGo/Alpha Zero https://en.wikipedia.org/wiki/AlphaZero
- SearchTrees has no chance in huge state-action spaces
  - AlphaGo:
    - beat professional Go player
    - 9 dan professional ranking
  - Alpha Zero: Top Chess Engine Championship 2017
    - 9h of self-play, no openingbooks nor endgames tables
    - 1 minute per move, 1GB RAM
    - 28 wins, 72 withdraws
- DOTA 2 openAI+ bot https://blog.openai.com/dota-2/
- AutoML https://cloud.google.com/automl/
  - [Zoph 2016] REINFORCE learns RCNN policy which generates deep CNN architectures.

Summary

- If accurate differentiable motion model and reward functions are known, than optimal control in MDP is straightforward optimization problem (efficiently tackled by DP or DDP)
- State-action value function is dual variable wrt policy. It serves as auxiliary function in the policy optimization:
  - actor-critic methods
  - heuristic in planning methods (LQR trees)
- **Holy grail** is to efficiently combine motion model, state-action value function and the policy optimization with efficient exploration
- RL will be much more useful for motion control, when accurate domain transfer methods (from simulators to reality) become available.