CS 175: Project in Artificial Intelligence

Winter 2026

Reinforcement Learning in a Nutshell

Roy Fox
Department of Computer Science
School of Information and Computer Sciences
University of California, Irvine

Logistics

assignments

- Exercise 1 is due next Wednesday (individual)
- Project proposals are due next Friday (team)

meetings

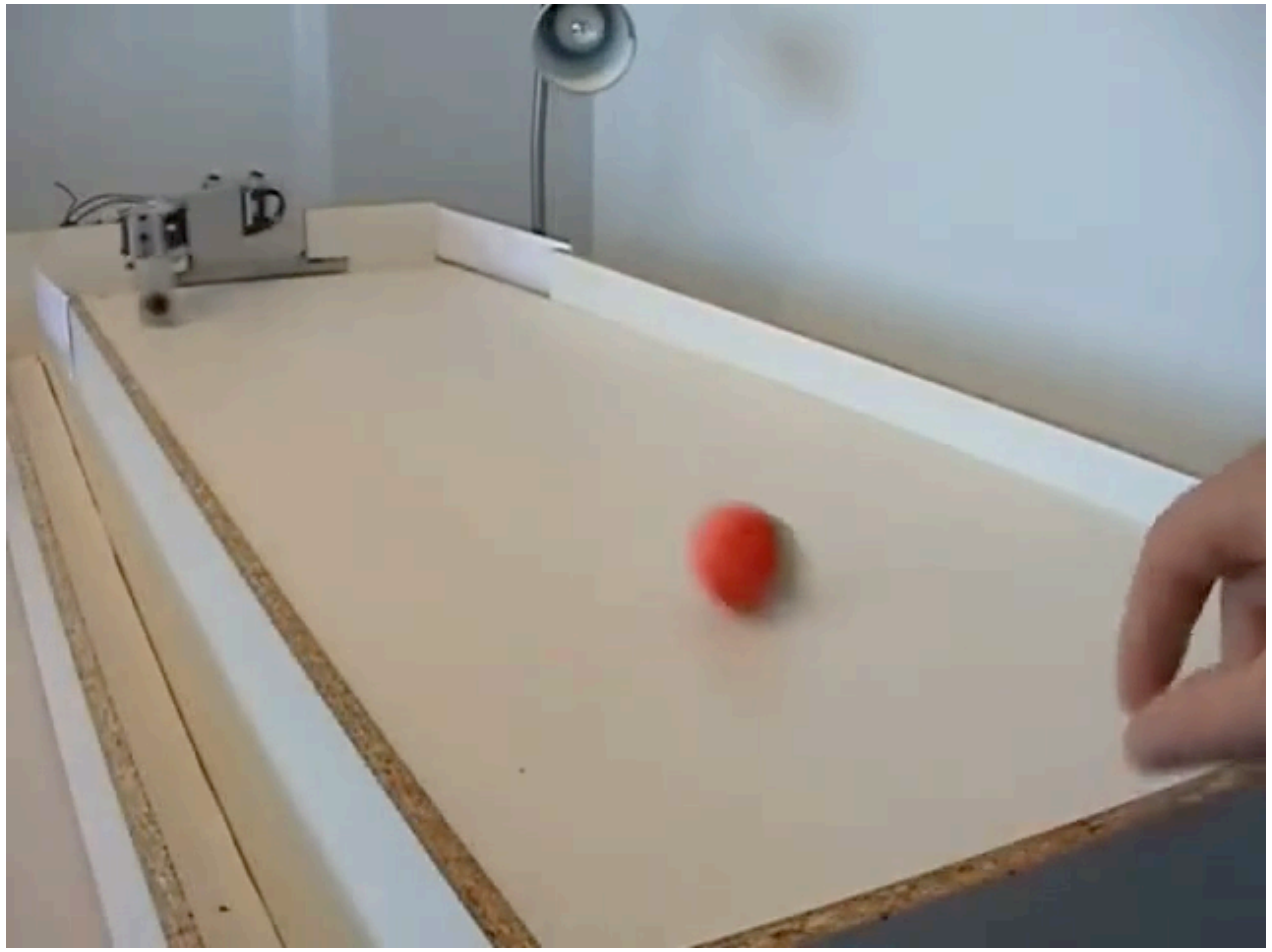
- Meet the instructor at least once by next Friday
- Welcome to schedule as much as you need

Basic RL concepts

- State: $s \in \mathcal{S}$; action: $a \in \mathcal{A}$; reward: $r(s, a) \in \mathbb{R}$
- Dynamics: $p(s_{t+1} | s_t, a_t)$ for stochastic; $s_{t+1} = f(s_t, a_t)$ for deterministic
- Policy: $\pi(a_t | s_t)$ for stochastic; $a_t = \pi(s_t)$ for deterministic
- Trajectory: $p_\pi(\xi = s_0, a_0, s_1, a_1, \ldots) = p(s_0) \prod_t \pi(a_t | s_t) p(s_{t+1} | s_t, a_t)$
- Return: $R(\xi) = \sum_t \gamma^t r(s_t, a_t) \qquad 0 \leq \gamma < 1$
- Value: $V_\pi(s) = \mathbb{E}_{\xi \sim p_\pi}[R | s_0 = s]$

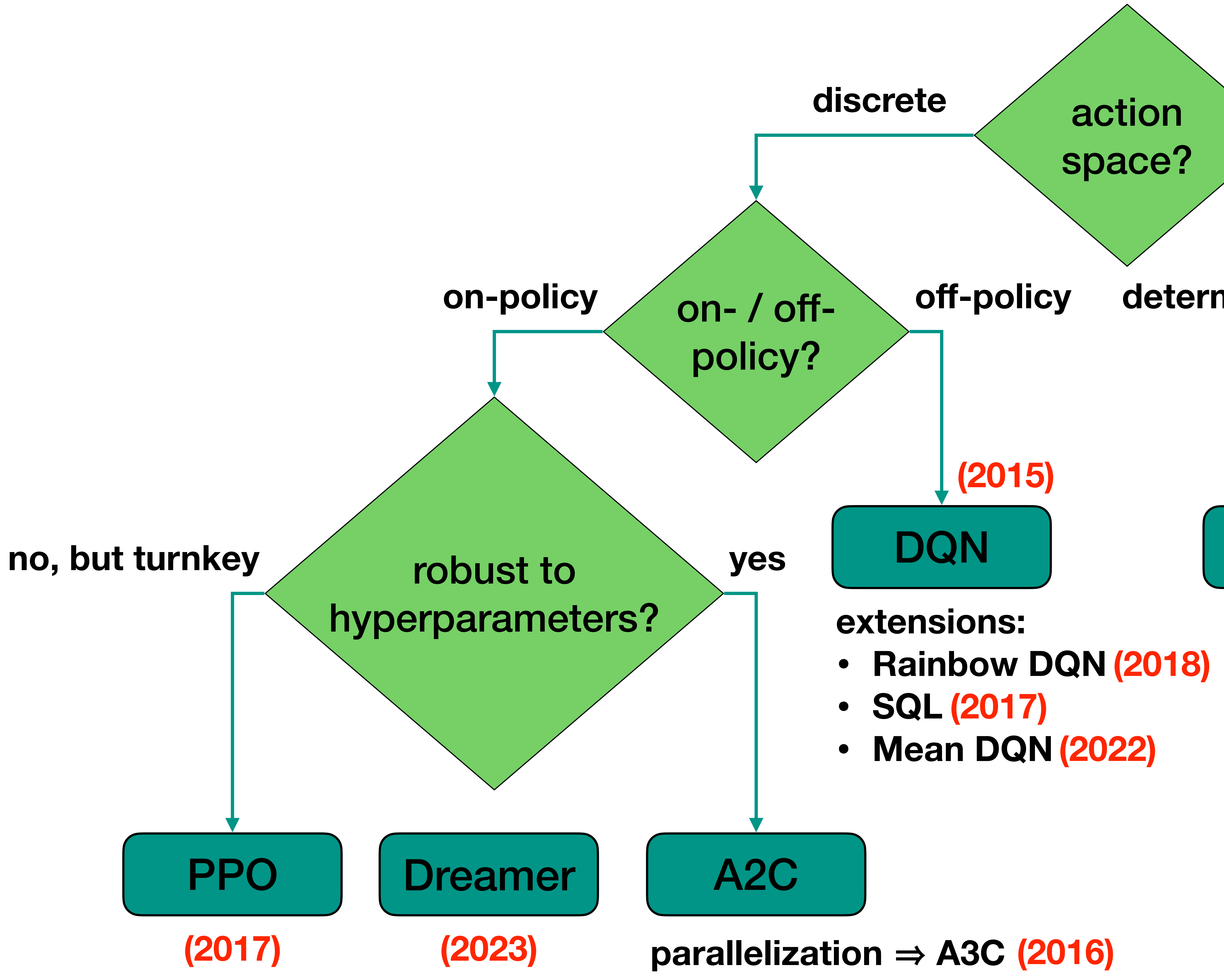
  $Q_\pi(s, a) = \mathbb{E}_{\xi \sim p_\pi}[R | s_0 = s, a_0 = a]$

Example: Table Soccer



https://www.youtube.com/watch?v=CIF2SBVY-J0

Flowchart: which algorithm to choose?



Today's lecture

Behavior Cloning

Temporal Difference

Policy Gradient

and more…

Imitation Learning (IL)

- How can we teach an agent to perform a task?
- Often there is an expert that already knows how to perform the task
  - A human operator who controls a robot
  - A black-box artificial agent that we can observe but not copy
  - An agent with different representation or embodiment

- The expert can demonstrate the task to create a training dataset $\mathscr{D} = \{\xi^{(i)}\}_i$
  - Each demonstration is a trajectory $\xi = s_0, a_0, s_1, a_1, \ldots$
  - Then the learner imitates these demonstrations

IL = Learning from Demonstrations (LfD)

- Teacher provides demonstration trajectories $\mathscr{D} = \{\xi^{(1)}, \ldots, \xi^{(m)}\}$
- Learner trains a policy π_θ to minimize a loss $\mathscr{L}(\theta)$
- For example, negative log-likelihood (NLL):

$$\arg\min_\theta \mathscr{L}_\theta(\xi) = \arg\min_\theta(-\log p_\theta(\xi))$$

$$= \arg\max_\theta \left(\log p(s_0) + \sum_{t=0}^{T-1} \log \pi_\theta(a_t \mid s_t) + \log p(s_{t+1} \mid s_t, a_t) \right)$$

$$= \arg\max_\theta \sum_{t=0}^{T-1} \log \pi_\theta(a_t \mid s_t)$$

model-free
= no need to know the environment dynamics p

Behavior Cloning (BC)

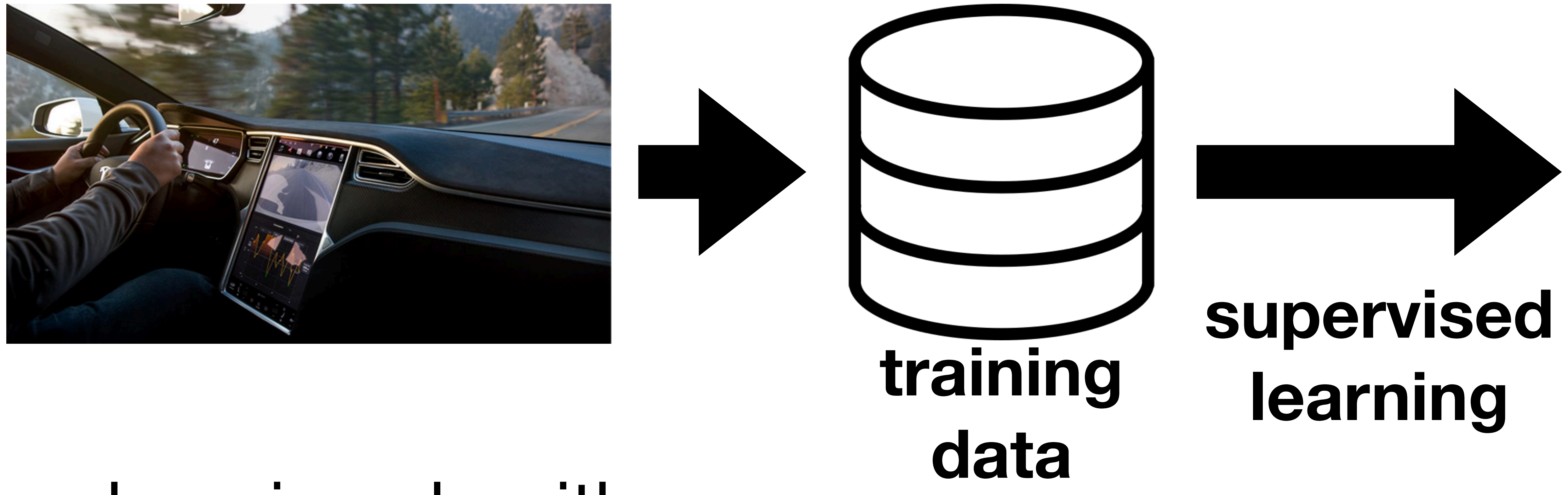
- Behavior Cloning:
  - Break down trajectories $\{\xi^{(1)}, \ldots, \xi^{(m)}\}$ into steps $\{(s_0^{(1)}, a_0^{(1)}), \ldots, (s_{T_m-1}^{(m)}, a_{T_m-1}^{(m)})\}$
  - Train $\pi_\theta : s \mapsto a$ using supervised learning

observations + actions → **training data** → $\pi_\theta(a \mid s)$

$$\mathcal{D} = \{(s_t^{(i)}, a_t^{(i)})\}_{i,t}$$

$$\max_\theta \frac{1}{|\mathcal{D}|} \sum_{(s,a)\in\mathcal{D}} \log \pi_\theta(a \mid s)$$

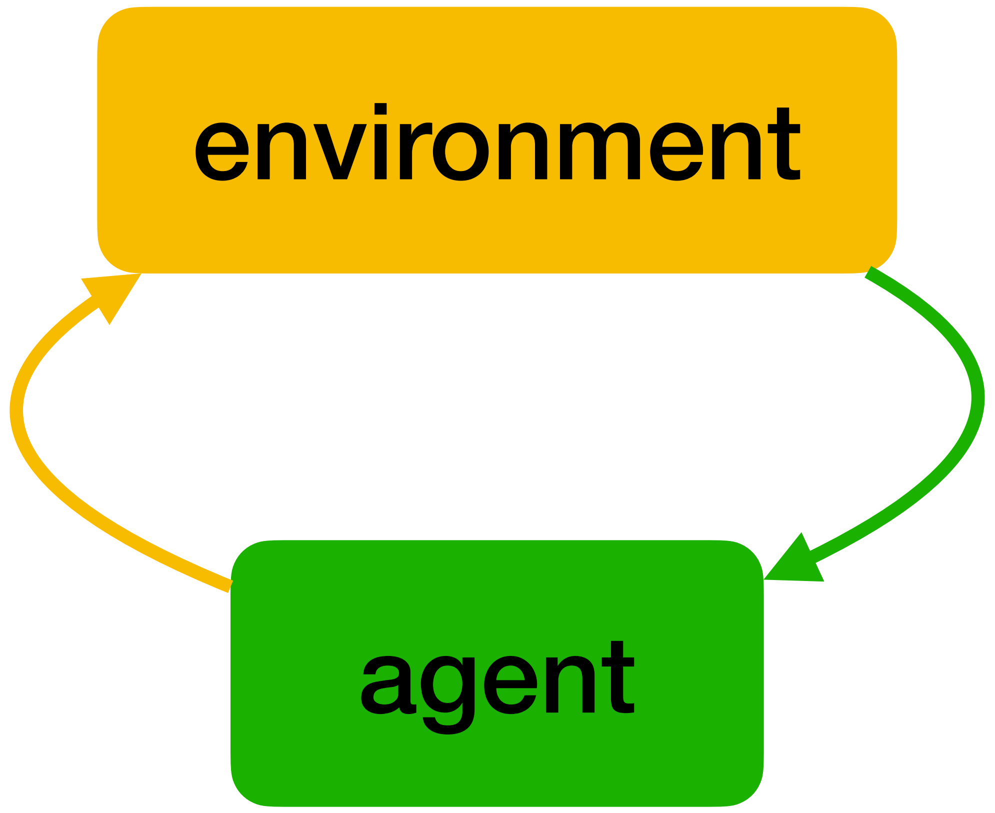
Behavior Cloning (BC)

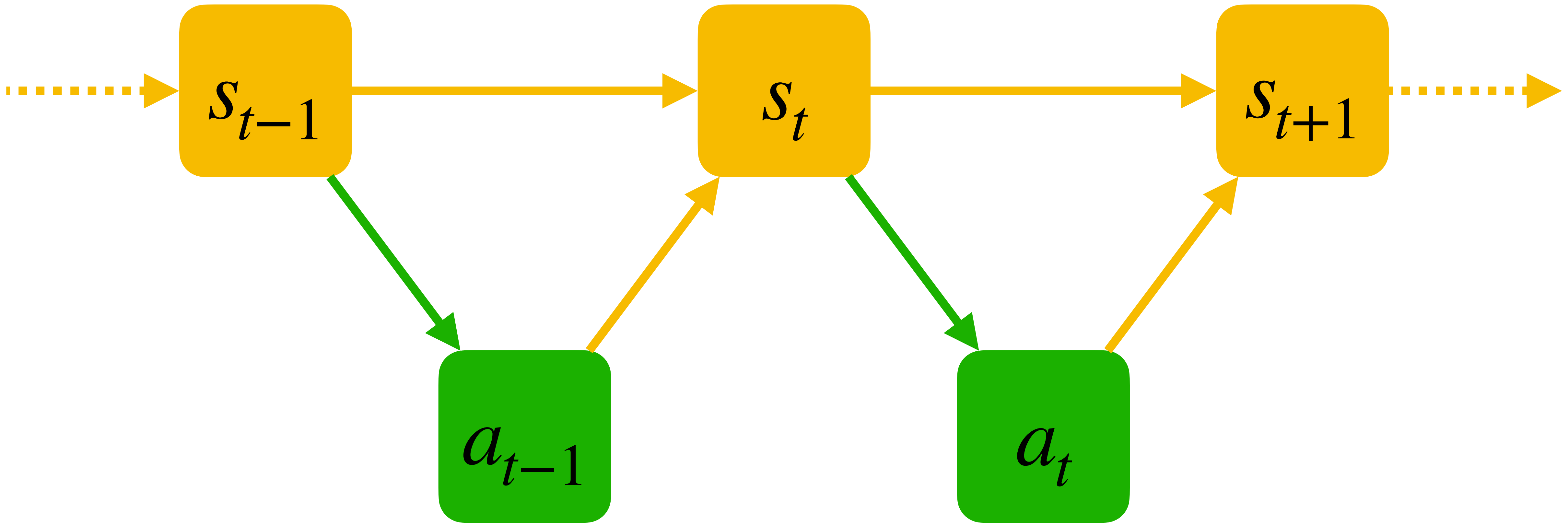


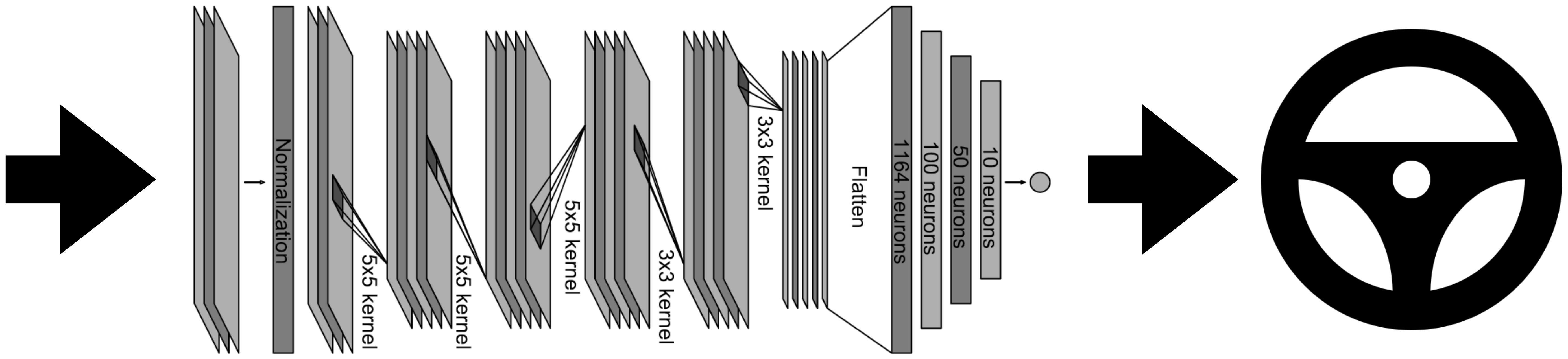
$\pi_\theta(a \mid s)$

- Benefits:
  - Simple, flexible — can use any learning algorithm
  - Model-free — never need to know the environment
- Drawbacks:
  - Only as good as the demonstrator
  - Only good in demonstrated states — how do we evaluate?
    - Validation loss (on held out data)? Task success rate in rollouts?

A policy is a (stochastic) function





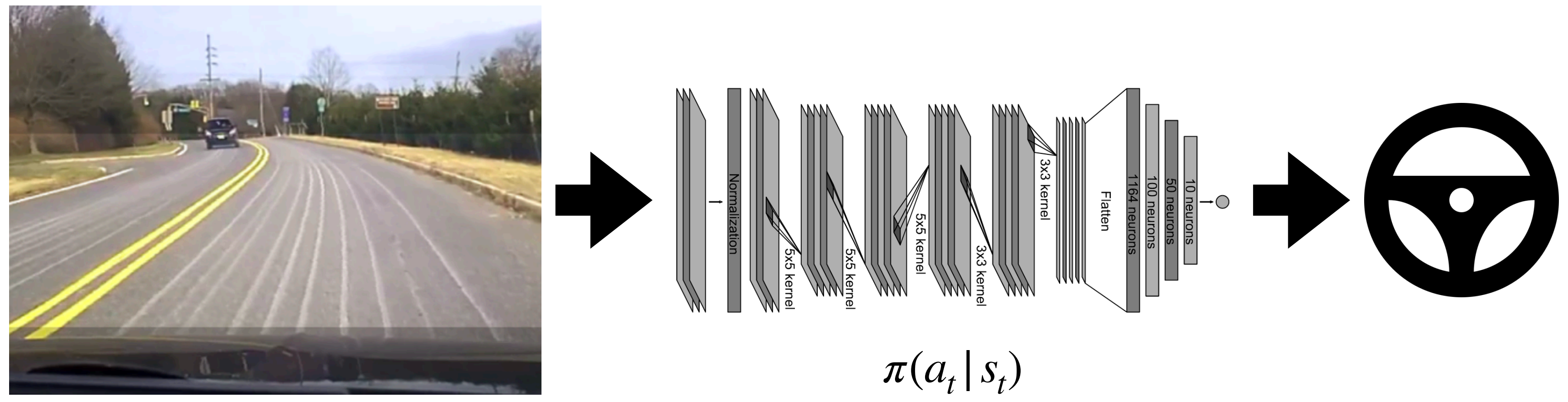


$$\pi(a_t \mid s_t)$$

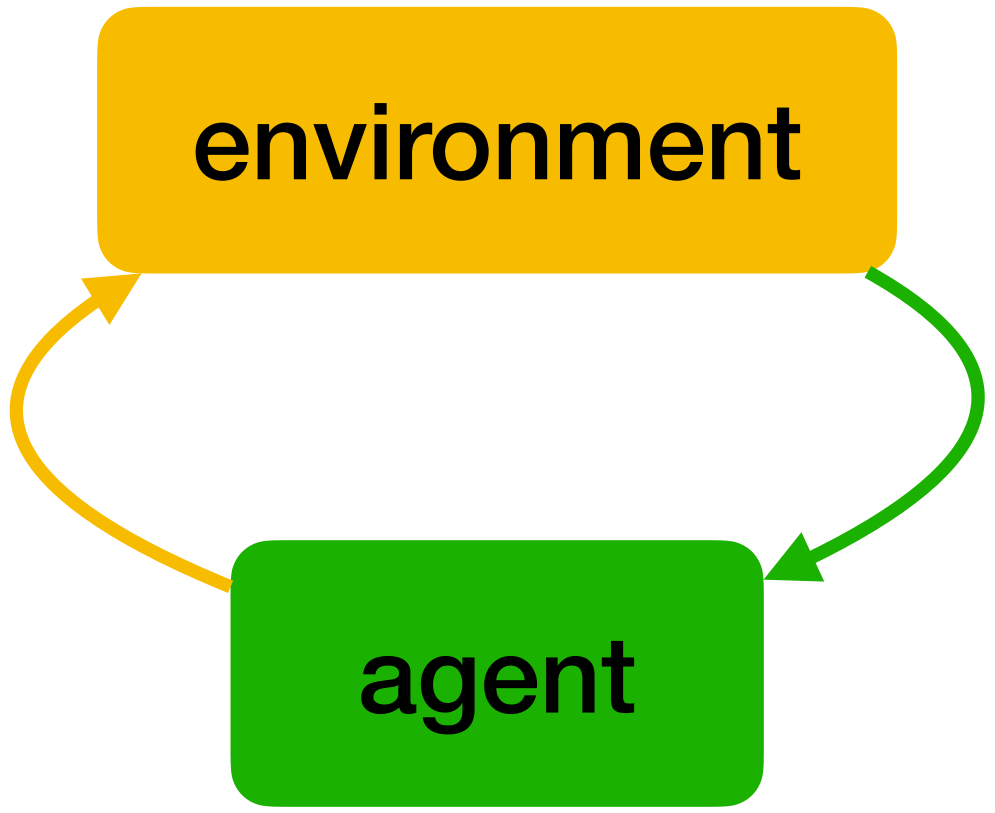
[Bojarski et al., 2016]

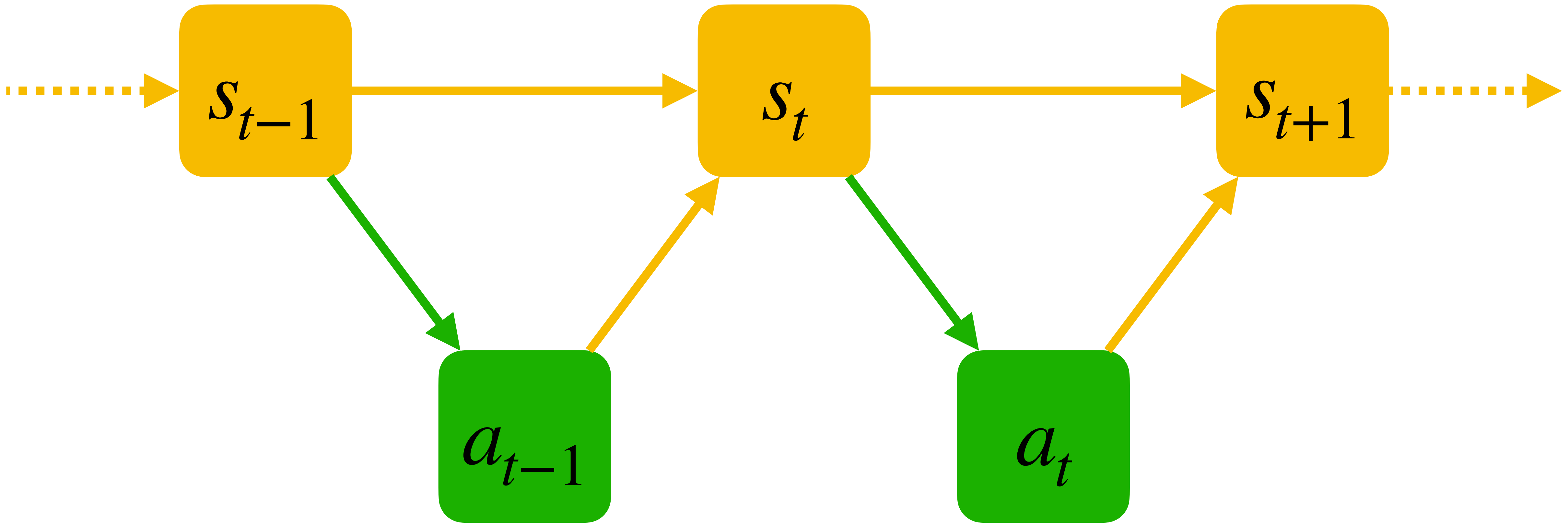
Stochastic policies

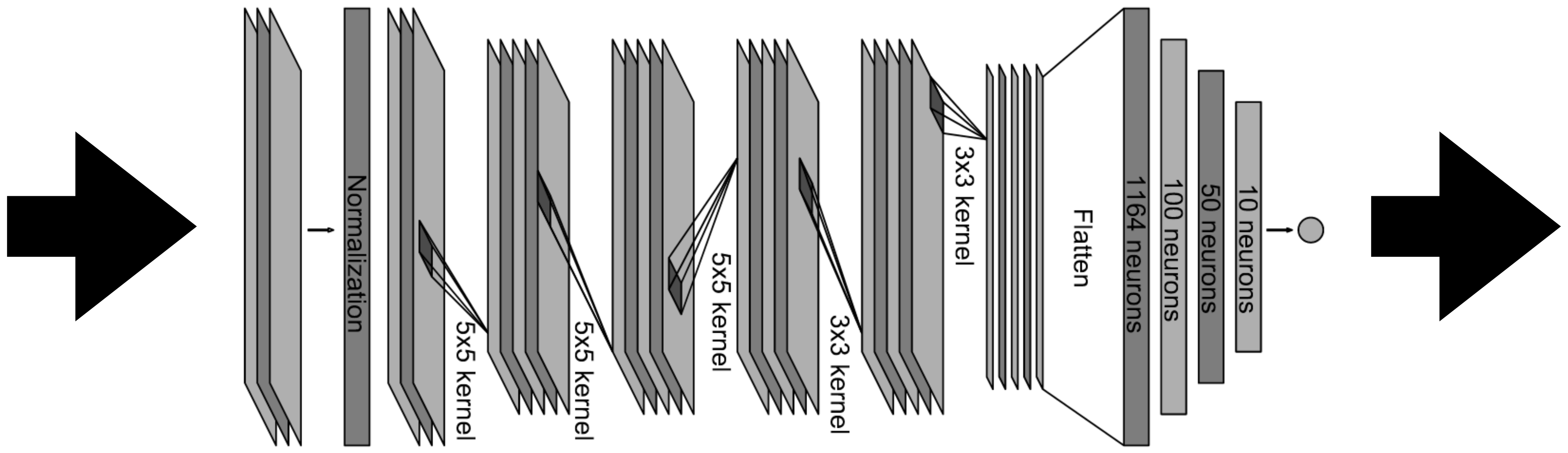
- Learned models are often deterministic functions $f_\theta : x \mapsto y$
- To implement a stochastic policy: output distribution parameters
- Examples:
  - Discrete action space: categorical distribution
    - $\pi_\theta : s \mapsto \{\lambda_a\}_a; \pi_\theta(a \mid s) = \text{softmax}_a \lambda_a \propto \exp \lambda_a$
  - Continuous action space: Gaussian distribution
    - $\pi_\theta : s \mapsto (\mu, \Sigma); \pi_\theta(a \mid s) = \mathcal{N}(\mu, \Sigma)$



A policy is a (stochastic) function

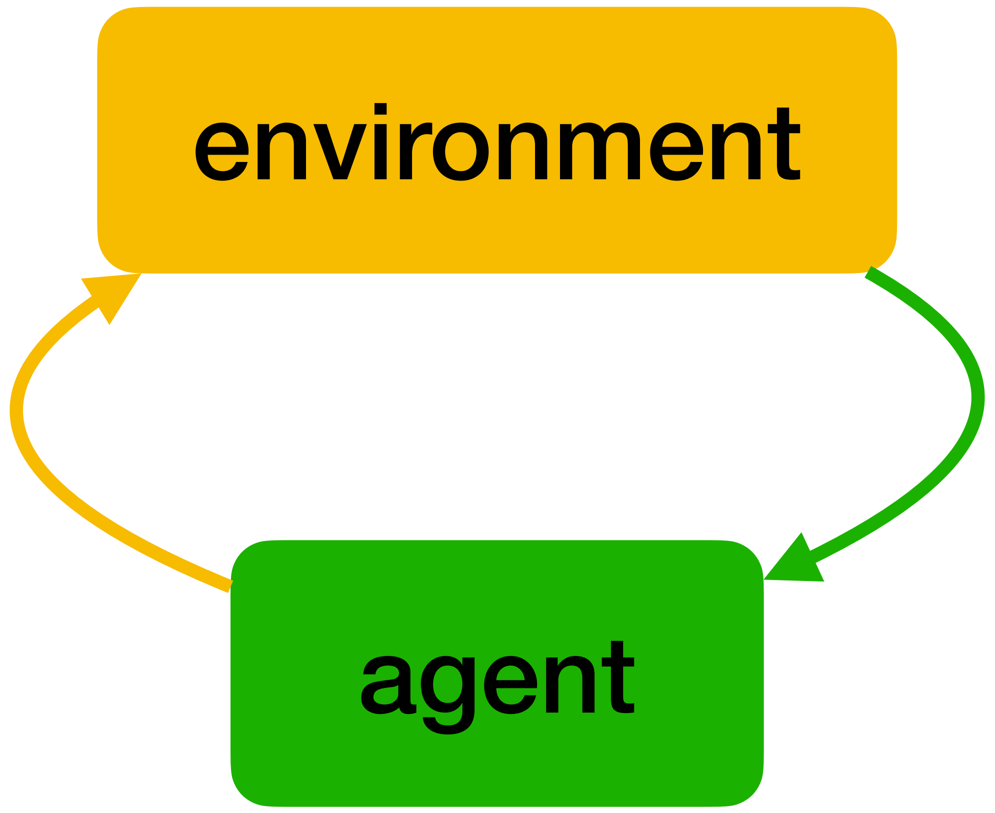


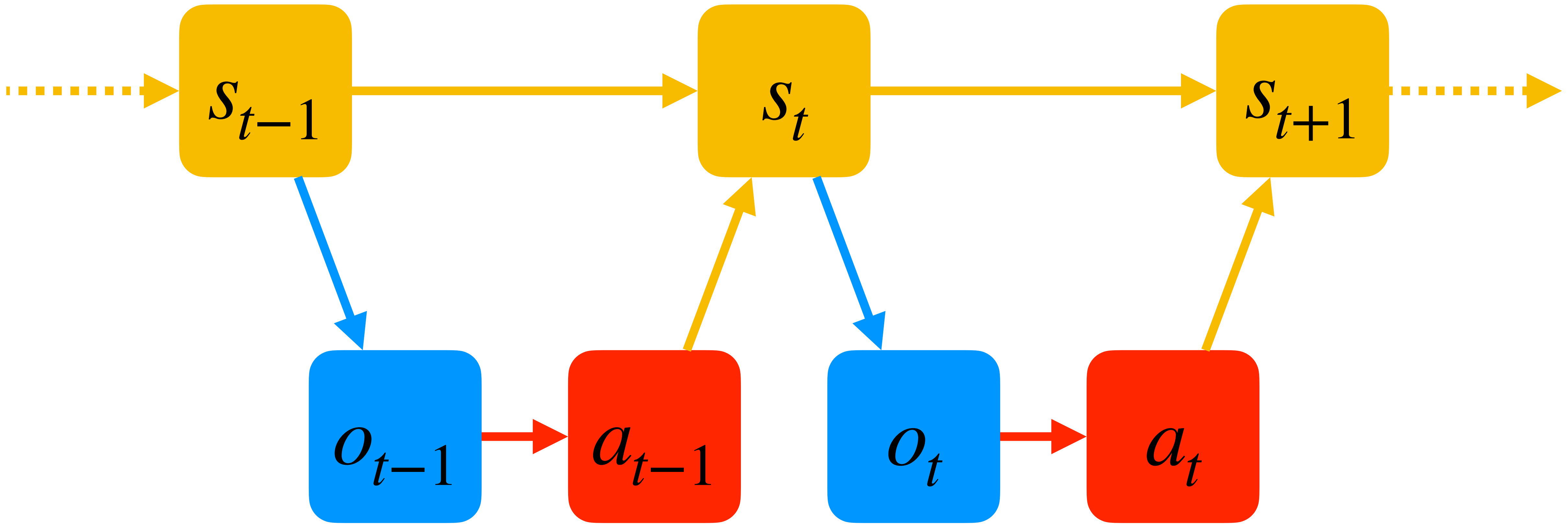


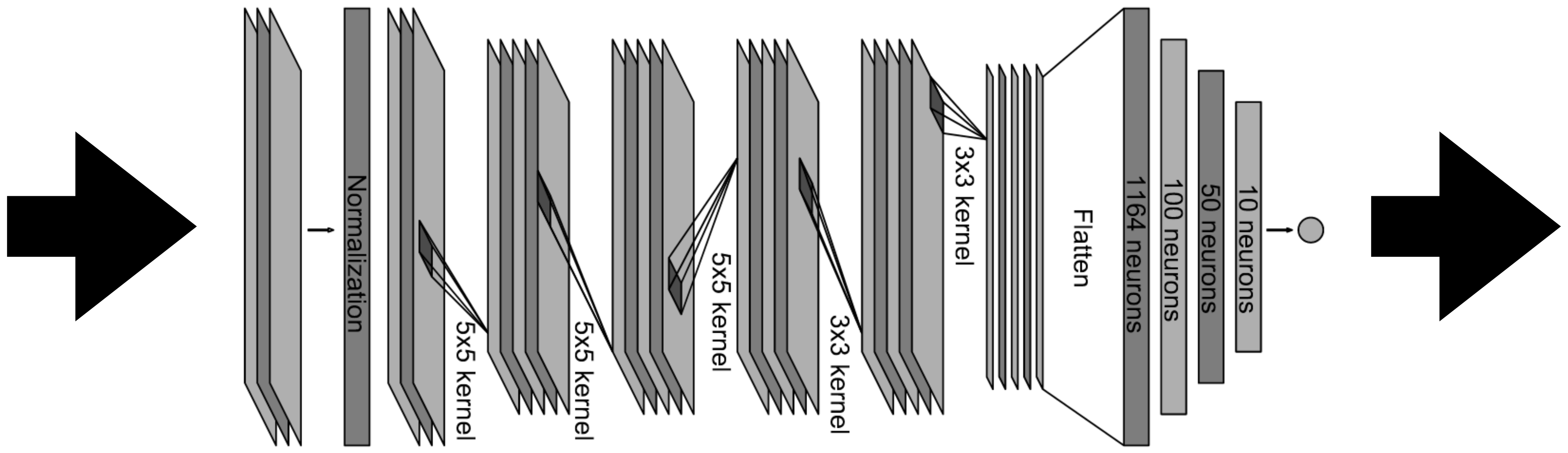


$$\pi(a_t \mid s_t)$$

A policy is a (stochastic) function







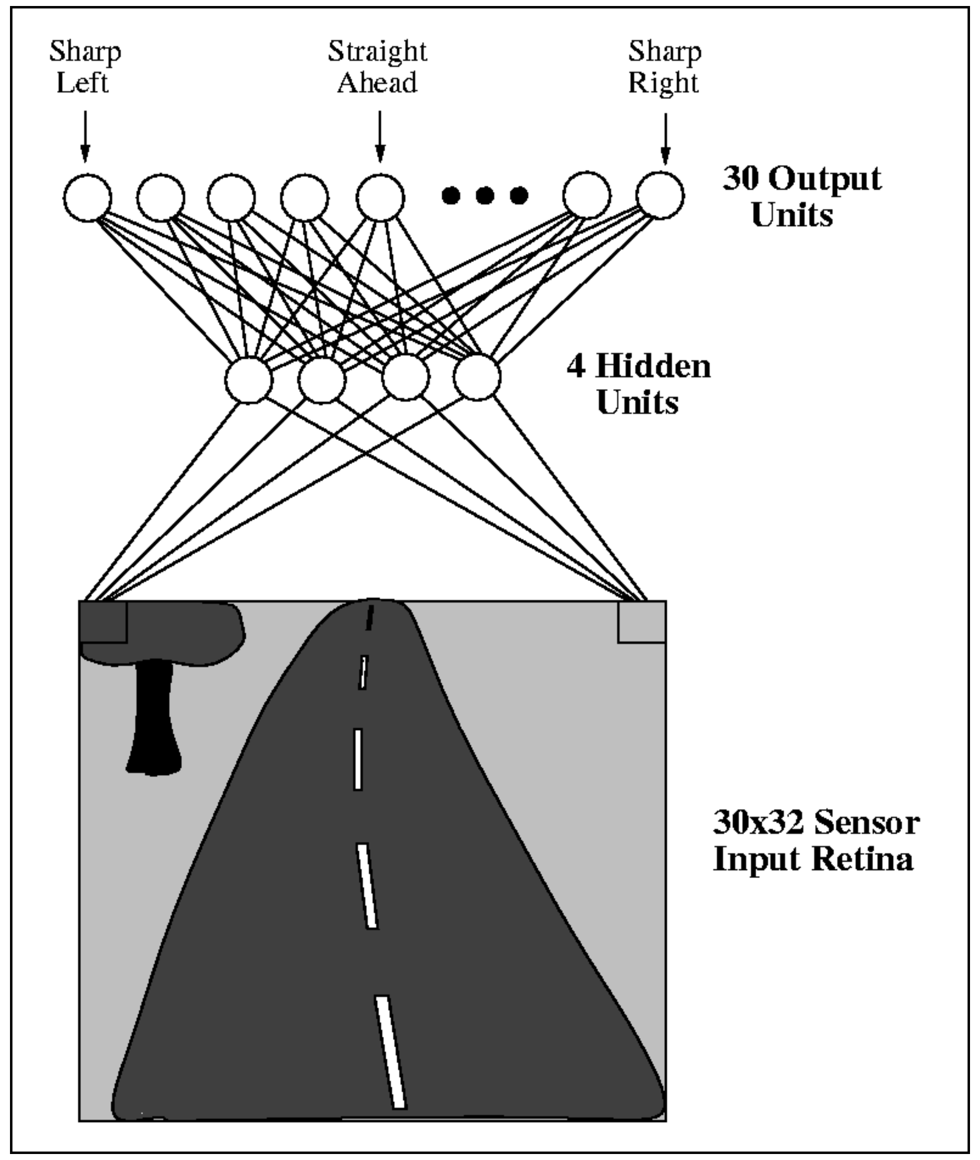
$\pi(a_t \mid o_t)$

observation

action

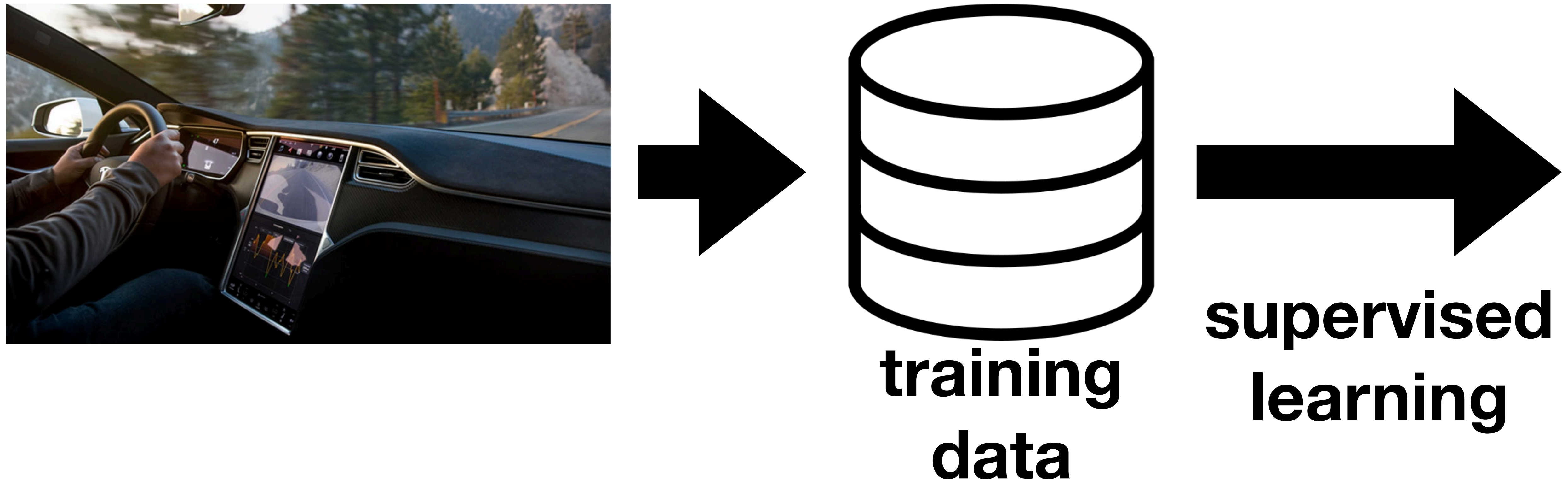
ALVINN

- Autonomous Land Vehicle in a Neural Network (ALVINN, 1989)



[Pomerleau, 1989]

Inaccuracy in BC



$\pi_\theta(a|s)$

- We could evaluate on held out teacher data, but really interested in using π_θ
- If the policy approximates the teacher $\pi_\theta(a_t|s_t) \approx \pi^*(a_t|s_t)$
  - The trajectory distribution will also approximate teacher behavior $p_\theta(\xi) \approx p^*(\xi)$
- But errors accumulate over time
  - May reach states not seen in the training dataset

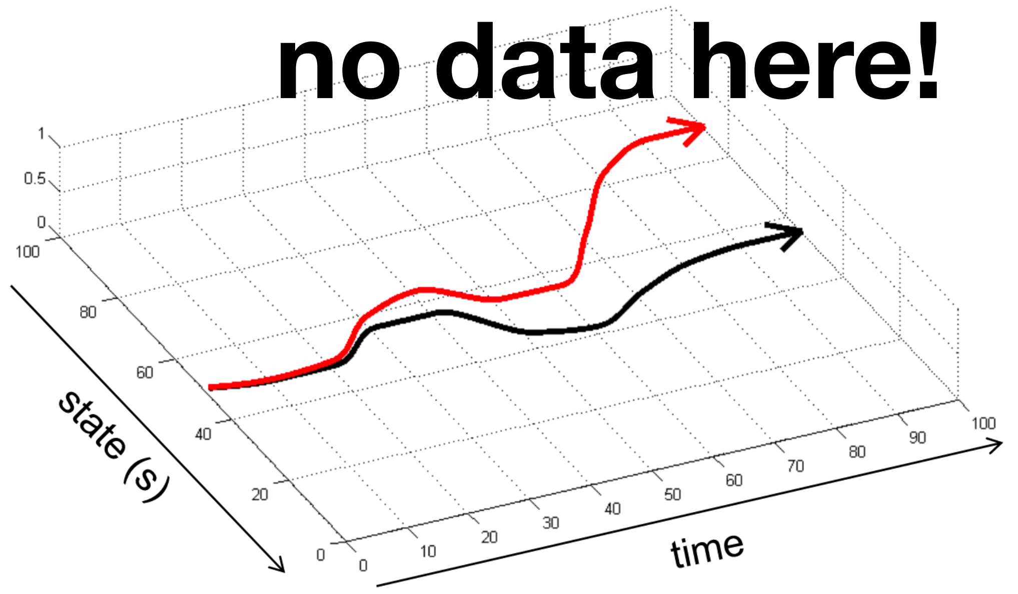
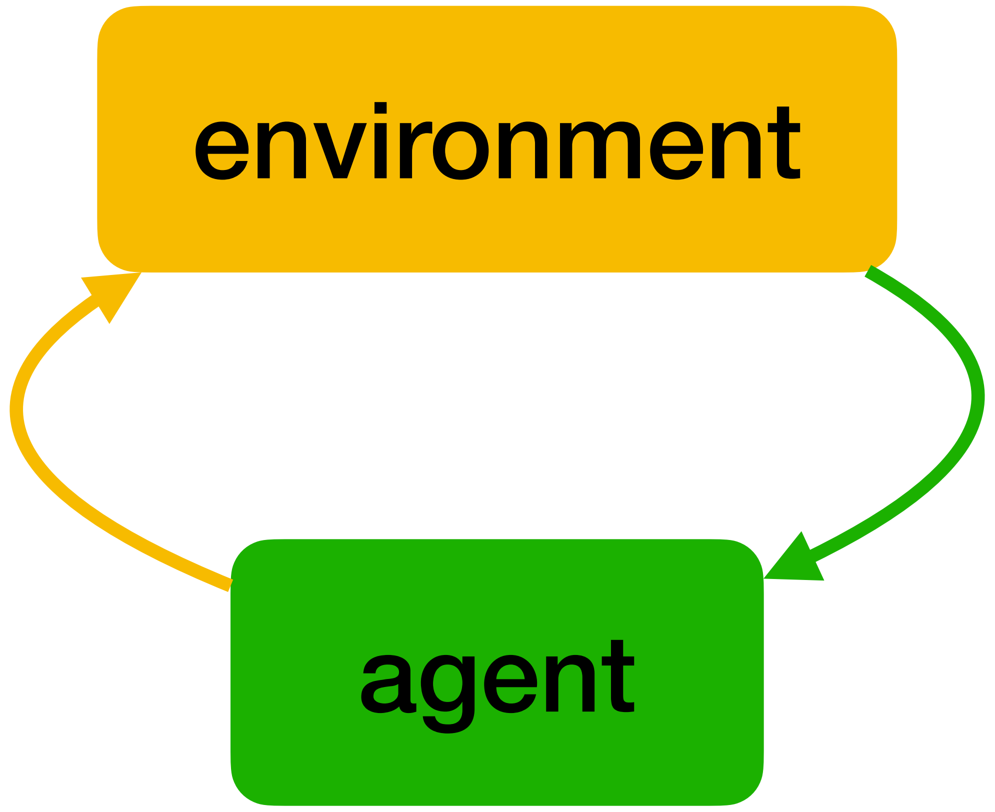


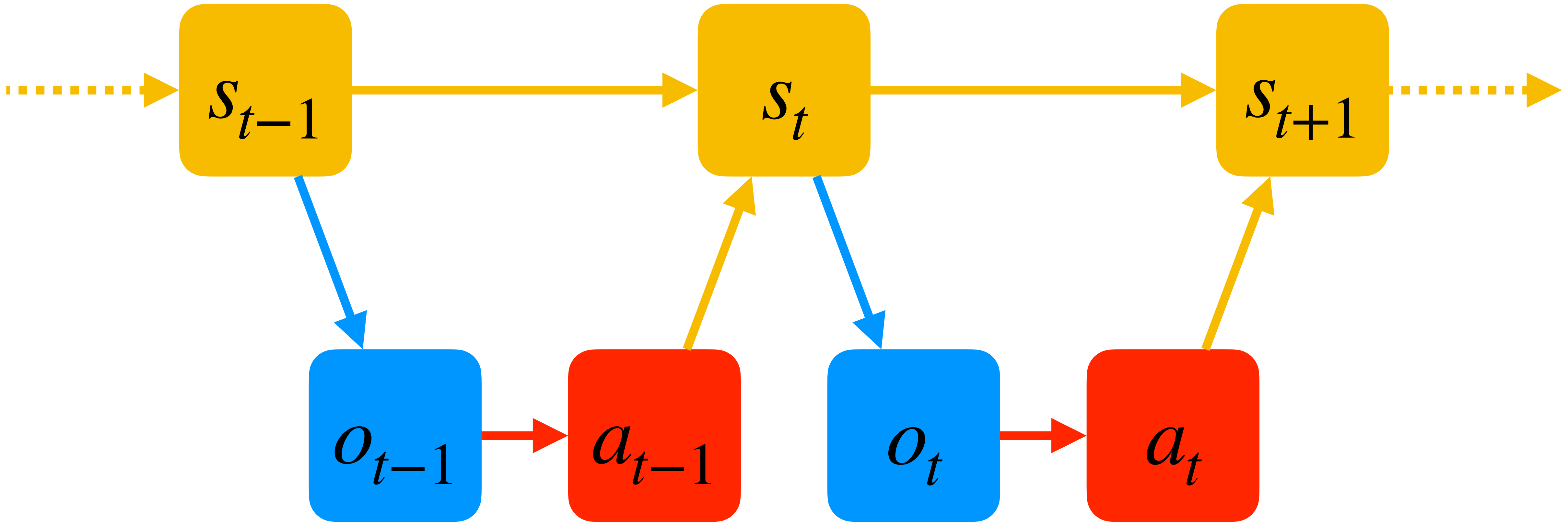
Image: Sergey Levine

Modeling partially observable behavior

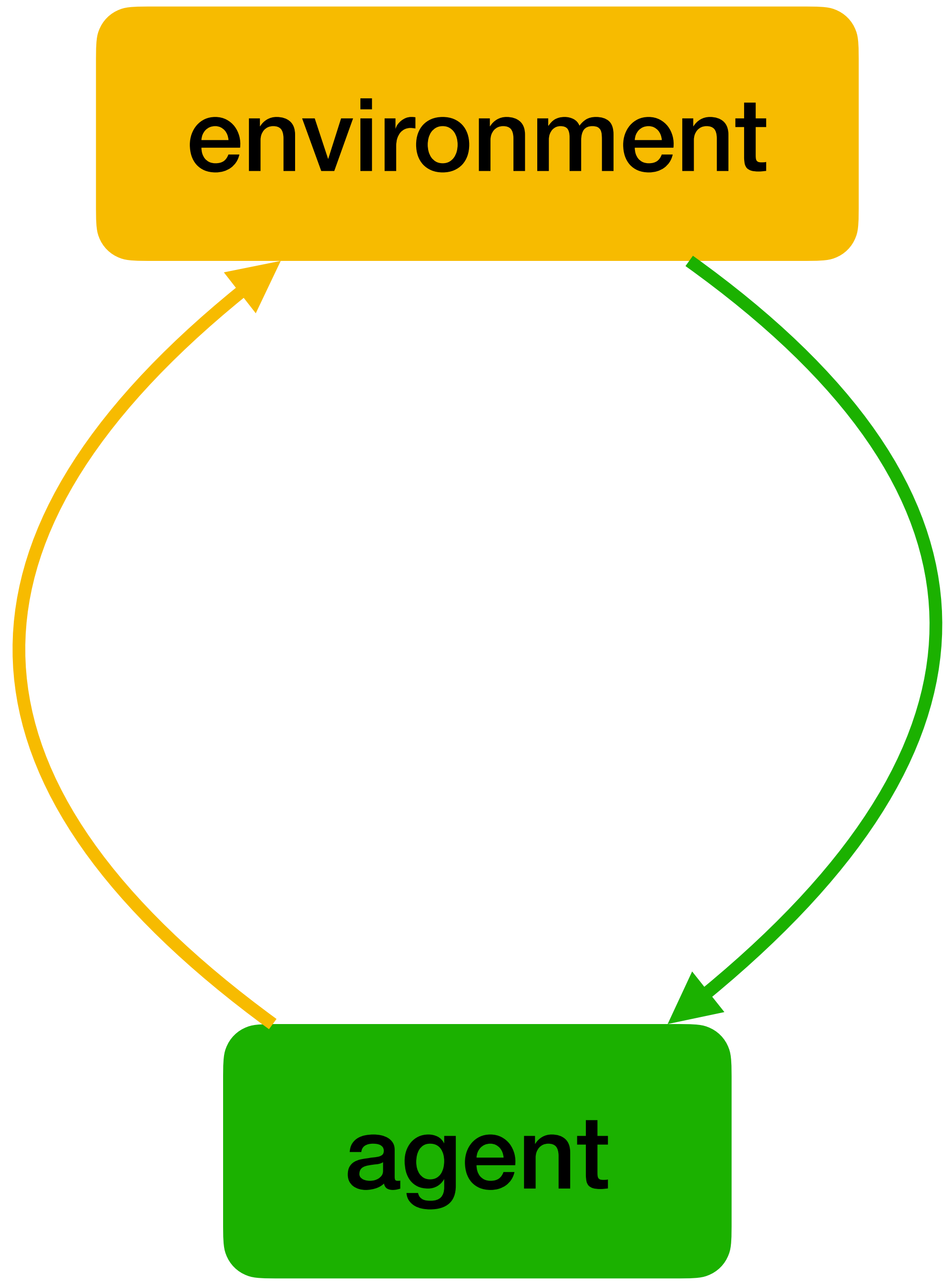
- Partial observations are not Markov
  - Generally, this means $p(o_{t+1} | o_t, a_t) \neq p(o_{t+1} | o_{\leq t}, a_{\leq t})$
  - Reactive policy $\pi_\theta(a_t | o_t)$ may not be optimal
    - May need $\pi_\theta(a_t | o_{\leq t})$, or even $\pi_\theta(a_t | o_{\leq t}, a_{<t})$; but how?
- Can use RNNs $f_\theta : (h_{t-1}, a_{t-1}, o_t) \mapsto h_t$, or other memory models
- But memory state is latent in demonstrations
  - Modeling memory is hard

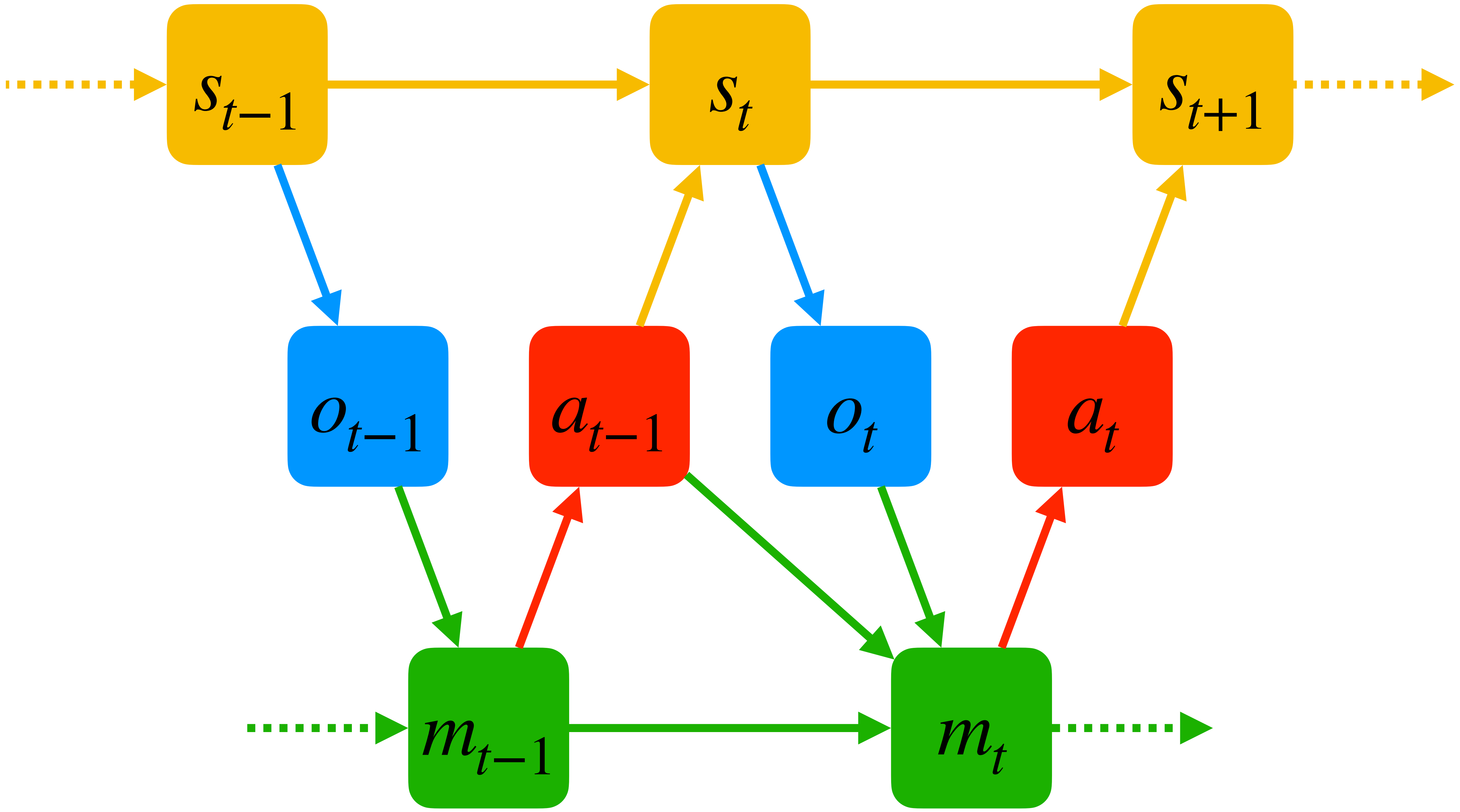
Modeling memory





Modeling memory





$\pi_\theta(m_t, a_t \mid m_{t-1}, a_{t-1}, o_t)$

- A common architecture:
    - A recurrent model $m_t = f_\theta(m_{t-1}, a_{t-1}, o_t)$; and an action model $\pi_\theta(a_t \mid m_t)$

Today's lecture

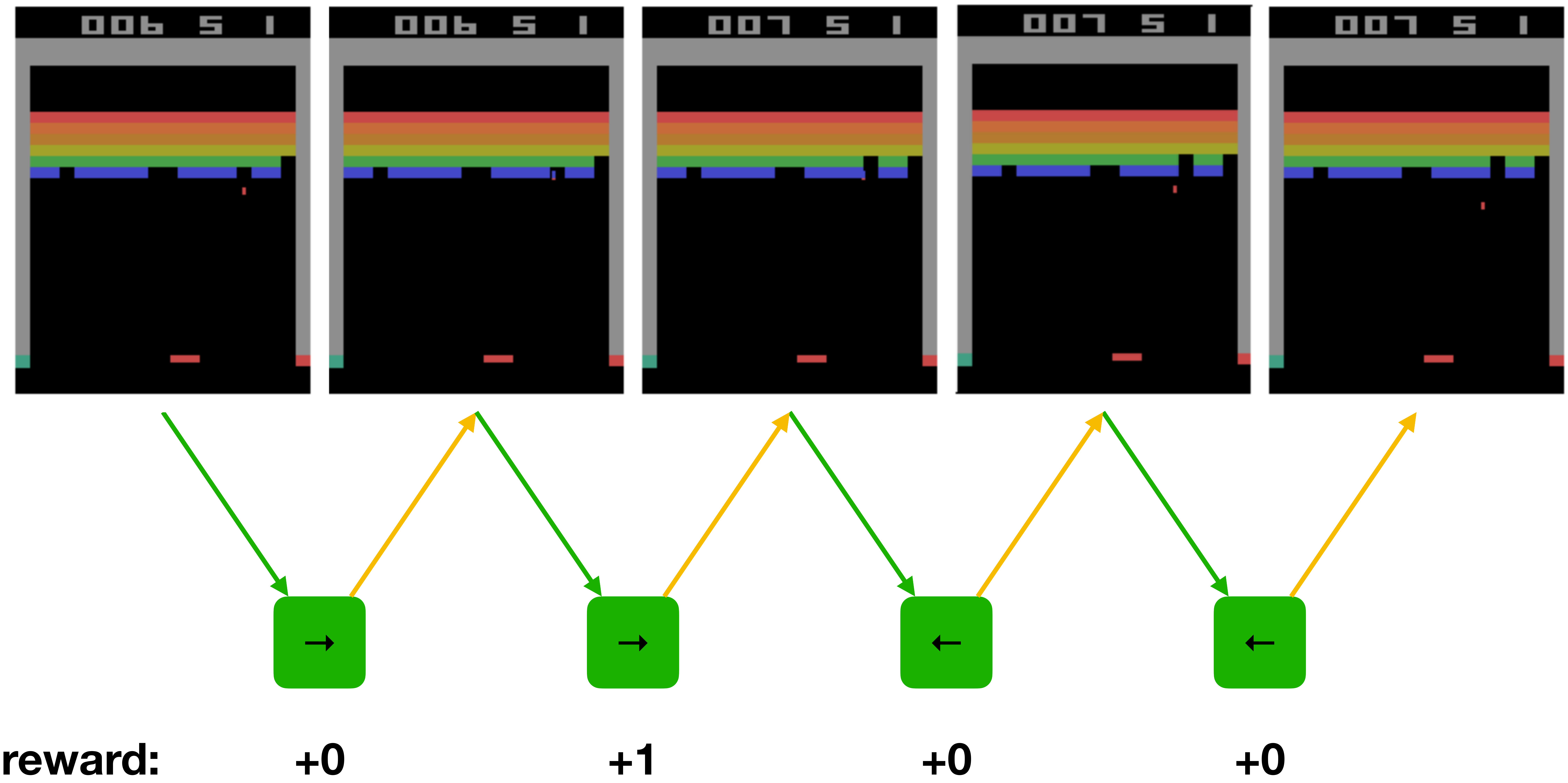
Behavior Cloning

Temporal Difference

Policy Gradient

and more...

Example: Breakout



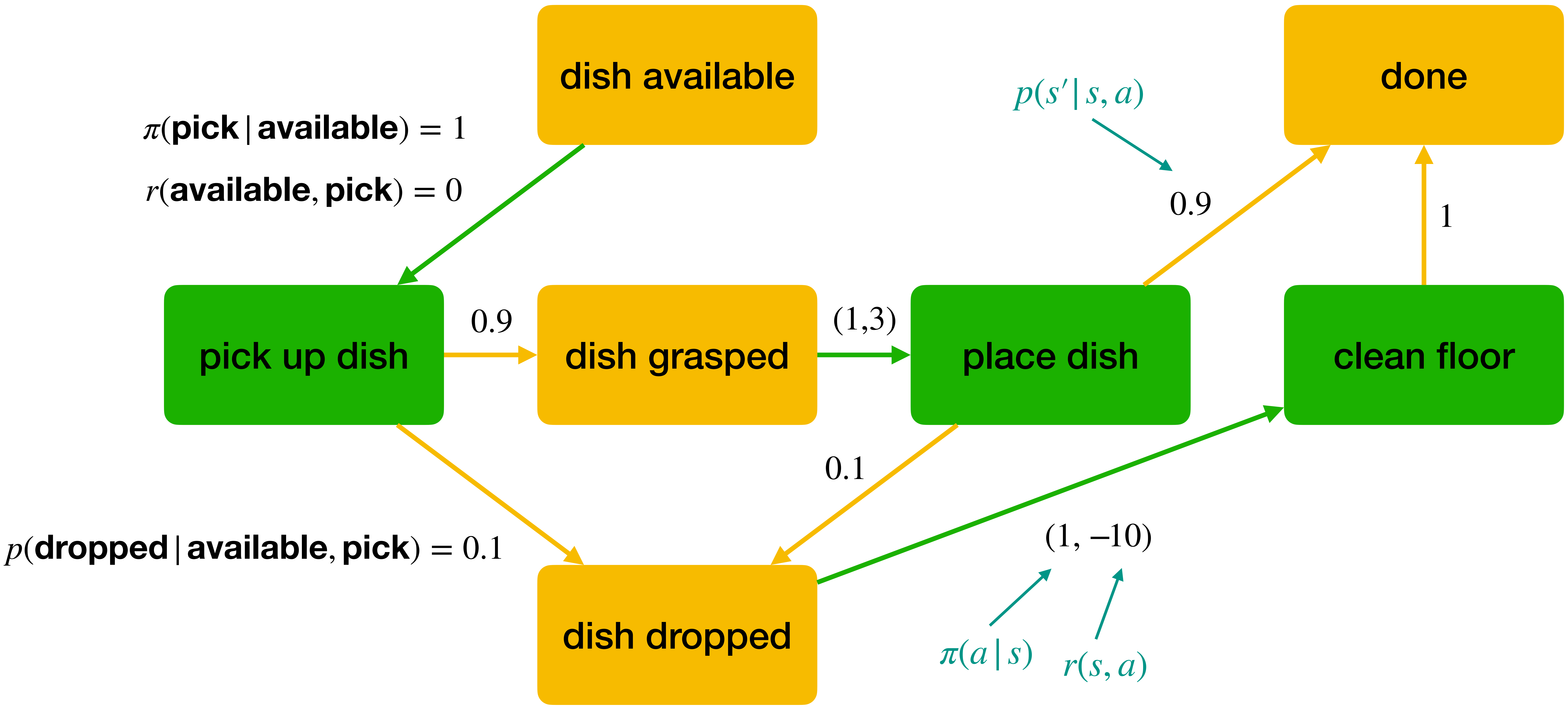
Formulating reward: considerations

- We define $r(s, a)$, is that general enough?
- What if the reward depends on the next state s'?
  - If we only care about expected reward, define $r(s, a) = \mathbb{E}_{(s'|s,a)\sim p}[r(s, a, s')]$
- What if the reward is a random variable $\tilde{r}$?
  - Define $r(s, a) = \mathbb{E}[\tilde{r} \,|\, s, a]$
  - In practice we see $\tilde{r}$ ⇒ don't just assume you know $r(s, a) = \tilde{r}$

RL objective: expected return

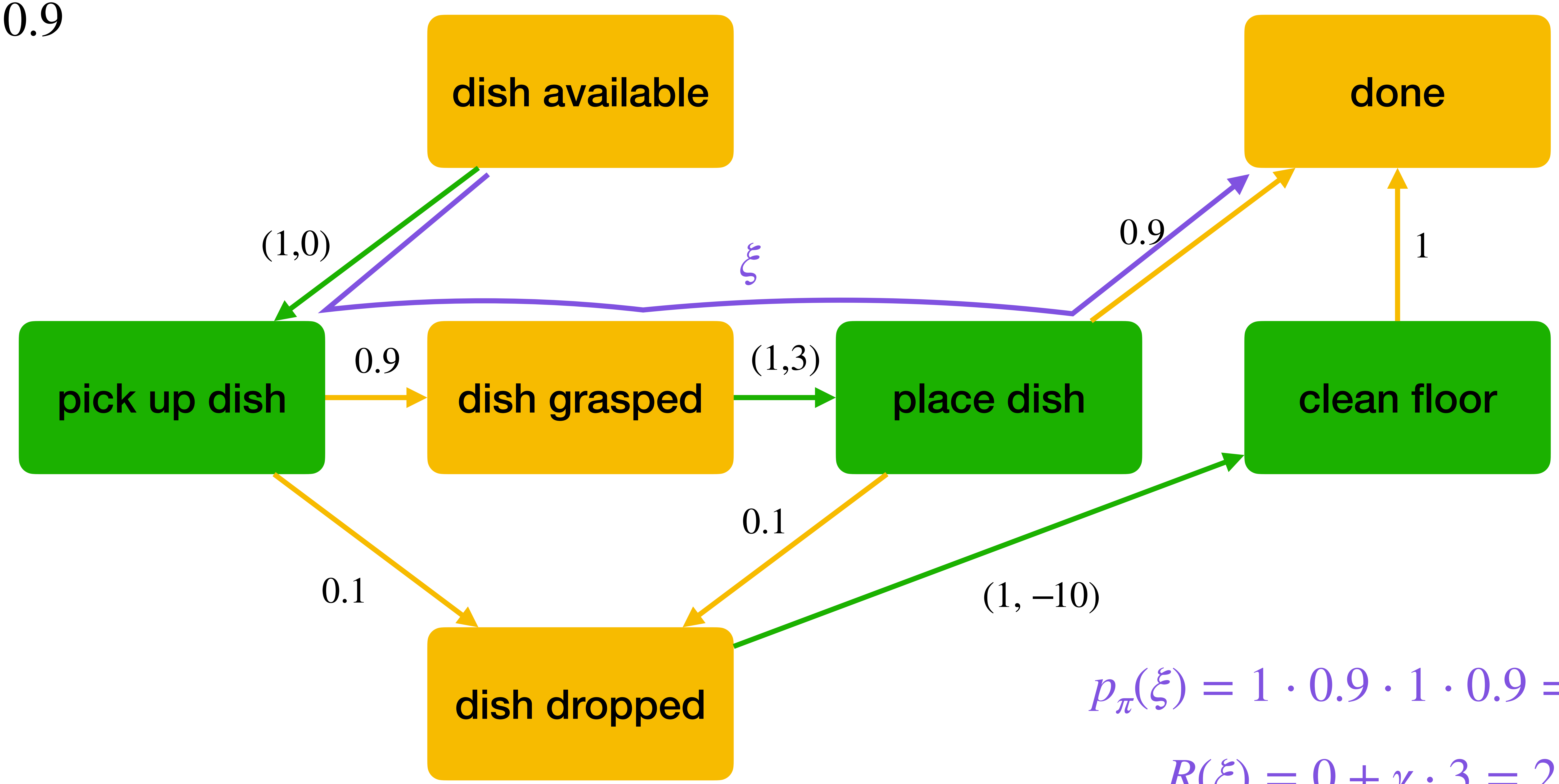
- We need a scalar to optimize
- Step 1: we have a whole sequence of rewards $\{r_t = r(s_t, a_t)\}_{t \geq 0}$
  - Summarize as return $R(\xi) = \sum_{t \geq 0} \gamma^t r(s_t, a_t)$
- Step 2: $R(\xi)$ is a random variable, induced by $p_\pi(\xi)$
  - Take expectation $J_\pi = \mathbb{E}_{\xi \sim p_\pi}[R(\xi)]$
- J_π can be calculated and optimized

Policy evaluation: example



Policy evaluation: example

$\gamma = 0.9$

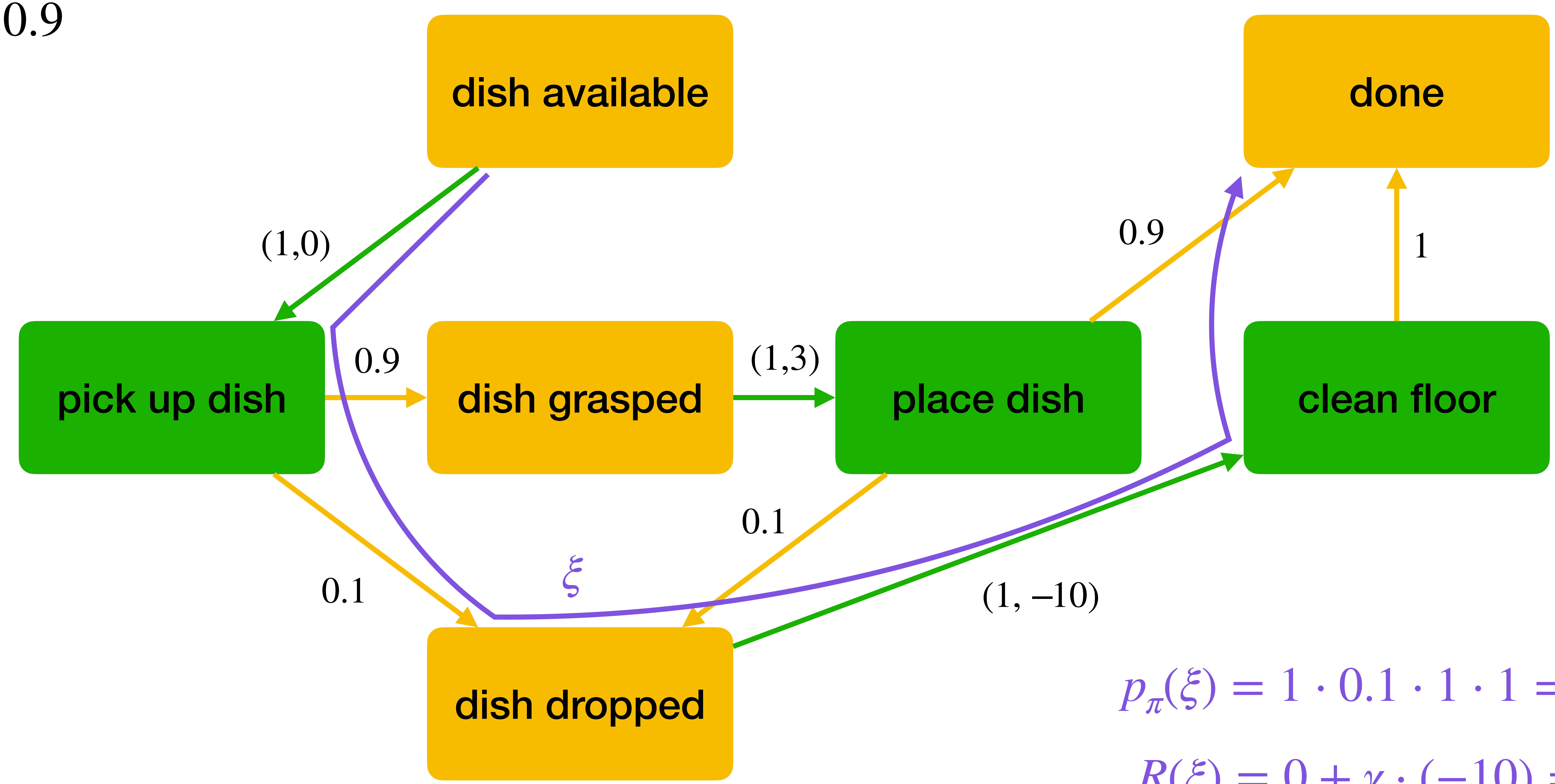


$p_\pi(\xi) = 1 \cdot 0.9 \cdot 1 \cdot 0.9 = 0.81$

$R(\xi) = 0 + \gamma \cdot 3 = 2.7$

Policy evaluation: example

$\gamma = 0.9$



$p_\pi(\xi) = 1 \cdot 0.1 \cdot 1 \cdot 1 = 0.1$

$R(\xi) = 0 + \gamma \cdot (-10) = -9$

Monte Carlo (MC) policy evaluation

- Computing $J_\pi = \mathbb{E}_{\xi \sim p_\pi}[R(\xi)] = \sum_\xi p_\pi(\xi) R(\xi)$ can be hard
  - Exponentially many trajectories
  - Model-based = requires $p(s' | s, a)$, which may not be known
- Monte Carlo: estimate expectation using empirical mean

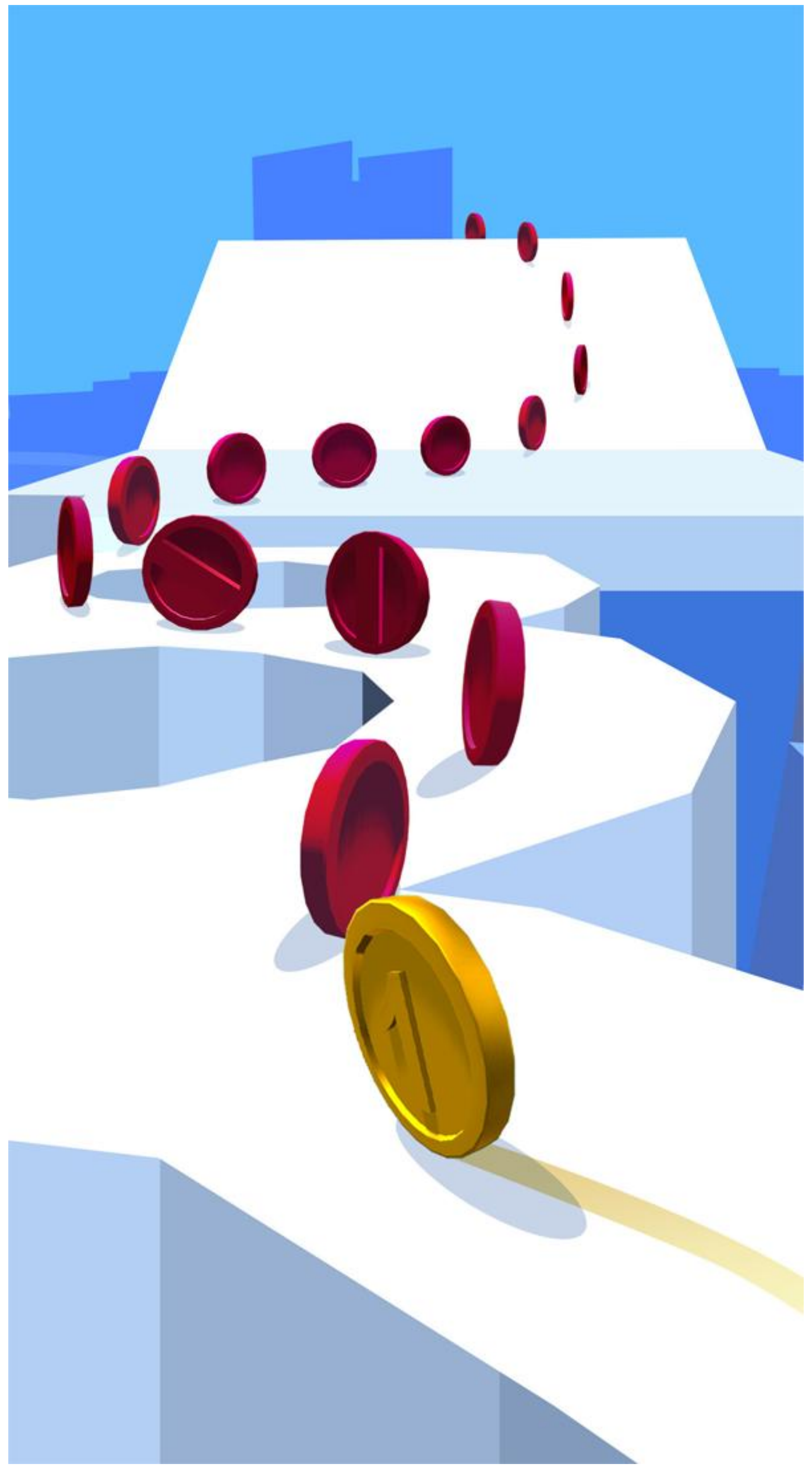
$$J_\pi \approx \frac{1}{m} \sum_i R(\xi^{(i)}) \qquad\qquad \xi^{(i)} \sim p_\pi$$

  - Model-free = can sample with rollouts, without knowing p

Value function

- RL objective: maximize expected return $J_\pi = \mathbb{E}_{\xi\sim p_\pi}[R]$
- We don't control s_0, can break down: $J_\pi = \mathbb{E}_{s_0\sim p}[V_\pi(s_0) \,|\, s_0]$
  - with the value function $V_\pi(s) = \mathbb{E}_{\xi\sim p_\pi}[R \,|\, s_0 = s]$
- $V_\pi(s)$ is the expected reward-to-go (= future return):
  - For any t, define $R_{\geq t} = \sum_{\Delta t \geq 0} \gamma^{\Delta t} r(s_{t+\Delta t}, a_{t+\Delta t})$
  - Then $V_\pi(s) = \mathbb{E}_{\xi\sim p_\pi}[R_{\geq t} \,|\, s_t = s]$

future reward after being in state s in time t



MC for value-function estimation

Algorithm MC for value-function estimation

Initialize $V(s) \leftarrow 0$ for all $s \in S$
repeat
  Sample $\xi \sim p_\pi$
  Update $V(s_0) \to R(\xi)$

"update LHS towards RHS"
i.e. $V(s_0) \mathrel{+}= \alpha(R(\xi) - V(s_0))$
with learning rate α

- Why not use the same samples for non-initial states?

Algorithm MC for value-function estimation (version 2)

Initialize $V(s) \leftarrow 0$ for all $s \in S$
repeat
  Sample $\xi \sim p_\pi$
  Update $V(s_t) \to R_{\geq t}(\xi)$ for all $t \geq 0$

MC with function approximation

- What if the state space is large?
  - Can't represent $V(s)$ as a big table
  - Won't have enough data to estimate each $V(s)$
- Function approximation: represent $V_\theta : S \to \mathbb{R}$
  - $\theta \in \Theta$, a parametric family of functions; for example, a neural network
- Generalization over state space $\Rightarrow$ data efficiency

Algorithm MC with function approximation

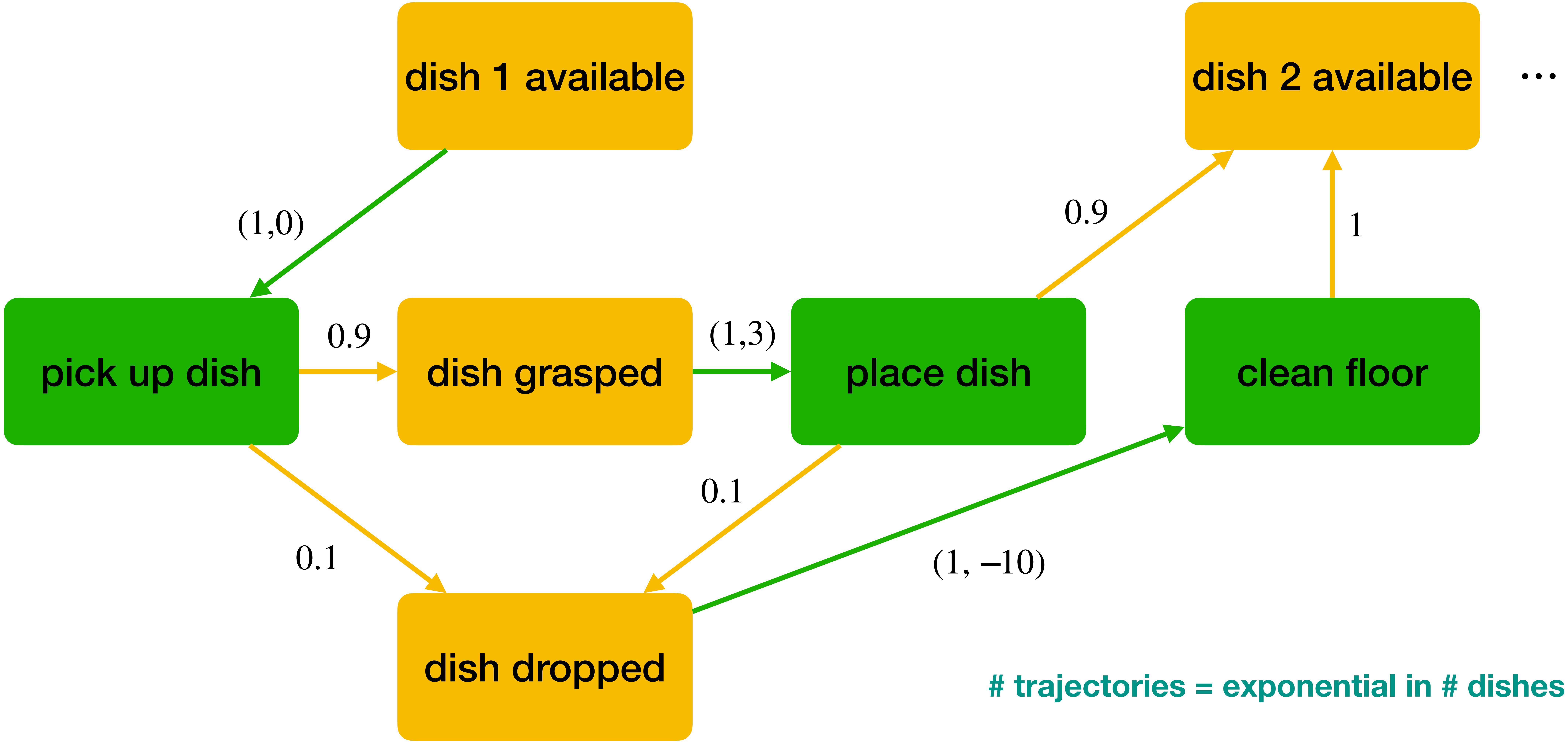
Initialize V_θ
repeat
  Sample $\xi \sim p_\pi$
  Descend on $\mathcal{L}_\theta = \sum_{t\geq 0}(R_{\geq t}(\xi) - V_\theta(s_t))^2$

with tabular representation:

$V(s_t) \mathrel{+}= -\alpha \nabla_{V(s_t)} \mathcal{L} = 2\alpha(R_{\geq t}(\xi) - V(s_t))$

same as in previous slide

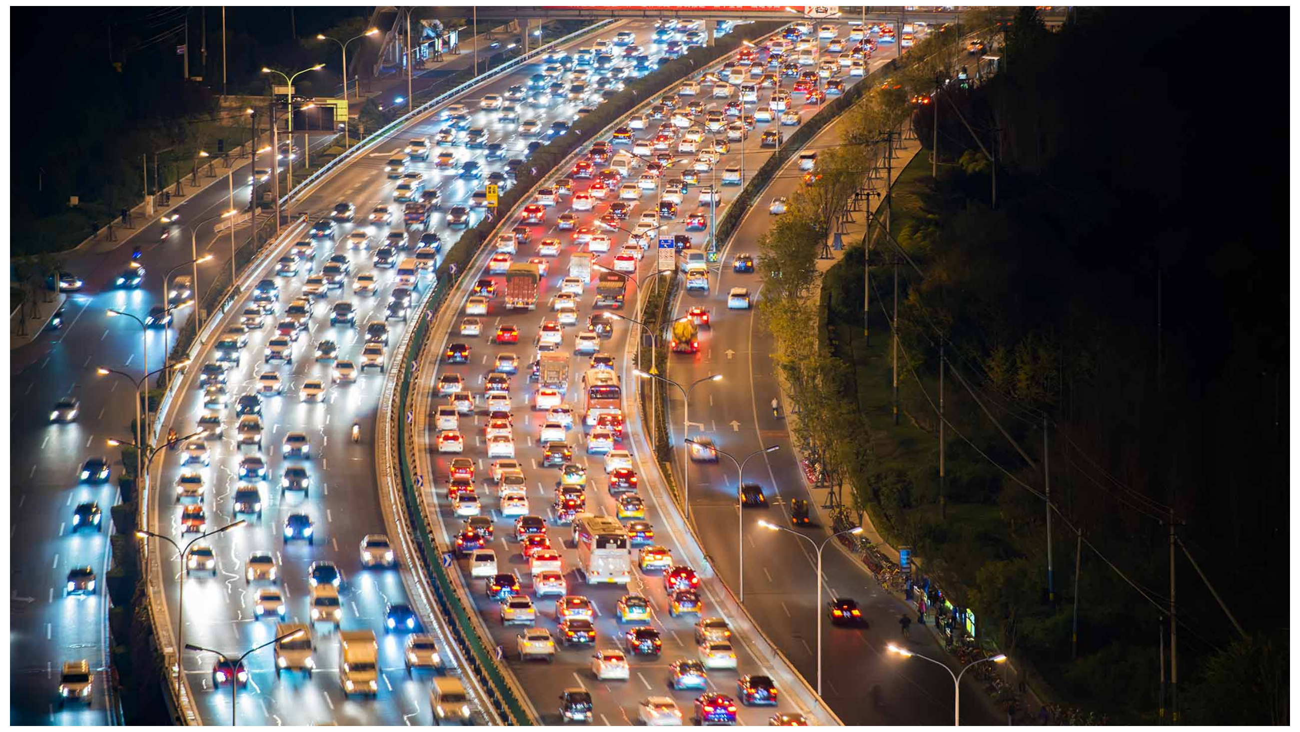
Policy evaluation: example



trajectories = exponential in # dishes

MC inefficiency

- The MC estimator is unbiased (correct expectation), but high variance
  - Requires many samples to give good estimate



- But MC misses out on the sequential structure
- Credit assignment problem:
  - Day 1: I take route 1 to work — 40 minutes; I take route 2 home — 10 minutes
  - Day 2: I take route 3 to work — 30 minutes; I take route 4 home — 30 minutes
- Which route should I take to work?
  - Route 1 → 50-minute daily commute, route 3 → 60-minute; is route 1 better?

Dynamic Programming (DP)

Richard Bellman

- Dynamic Programming = remember reusable partial results
- Value recursion:

$$V_\pi(s) = \mathbb{E}_{\xi \sim p_\pi}[R \,|\, s_0 = s]$$

break down sum of rewards $$= \mathbb{E}_{\xi \sim p_\pi}[r(s_0, a_0) + \gamma R_{\geq 1} \,|\, s_0 = s]$$

first reward only depends on a $$= \mathbb{E}_{(a|s) \sim \pi}[r(s, a) + \gamma \mathbb{E}_{\xi \sim p_\pi}[R_{\geq 1} \,|\, s_0 = s, a_0 = a]]$$

s' is a state, all that matters for $R_{\geq 1}$ $$= \mathbb{E}_{(a|s) \sim \pi}[r(s, a) + \gamma \mathbb{E}_{(s'|s,a) \sim p}[\mathbb{E}_{\xi \sim p_\pi}[R_{\geq 1} \,|\, s_1 = s']]]$$

definition of $V_\pi(s')$ $$= \mathbb{E}_{(a|s) \sim \pi}[r(s, a) + \gamma \mathbb{E}_{(s'|s,a) \sim p}[V_\pi(s')]]$$

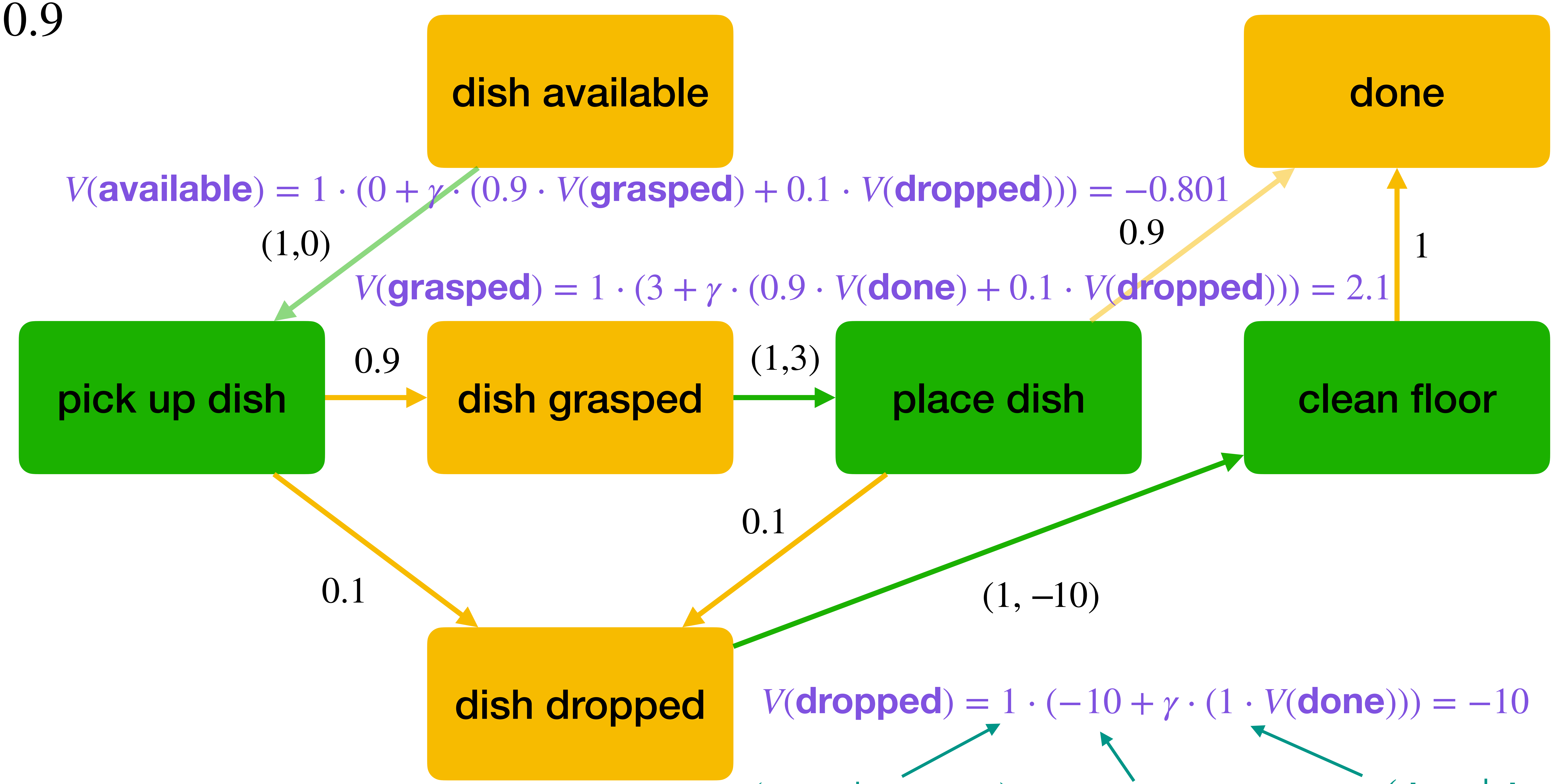
[Bellman, 1956]

Policy evaluation: example

$$V_\pi(s) = \mathbb{E}_{(a|s)\sim\pi}[r(s,a) + \gamma\mathbb{E}_{(s'|s,a)\sim p}[V_\pi(s')]]$$

$\gamma = 0.9$

$V(\textbf{done}) = 0$



$V(\textbf{available}) = 1 \cdot (0 + \gamma \cdot (0.9 \cdot V(\textbf{grasped}) + 0.1 \cdot V(\textbf{dropped}))) = -0.801$

$V(\textbf{grasped}) = 1 \cdot (3 + \gamma \cdot (0.9 \cdot V(\textbf{done}) + 0.1 \cdot V(\textbf{dropped}))) = 2.1$

$V(\textbf{dropped}) = 1 \cdot (-10 + \gamma \cdot (1 \cdot V(\textbf{done}))) = -10$

$\pi(\textbf{clean} \mid \textbf{dropped})$ $\quad r(\textbf{dropped}, \textbf{clean})$ $\quad p(\textbf{done} \mid \textbf{dropped}, \textbf{clean})$

DP + MC: Temporal Difference (TD)

- Policy evaluation with DP: $V_\pi(s) = \mathbb{E}_{(a|s)\sim\pi}[r(s,a) + \gamma\mathbb{E}_{(s'|s,a)\sim p}[V_\pi(s')]]$
  - Drawback: model-based = need to know p

recursion from s' to s = backward in time!

- MC: $V(s) \to R_{\geq t}(\xi)$, where $\xi \sim p_\pi$ and $s_t = s$
  - Drawback: high variance
- Put together: $V(s) \to r + \gamma V(s')$
  - where $s = s_t$, $r = r(s_t, a_t)$, and $s' = s_{t+1}$ in some trajectory
  - In other words: $V(s) \leftarrow V(s) + \alpha(r + \gamma V(s') - V(s))$

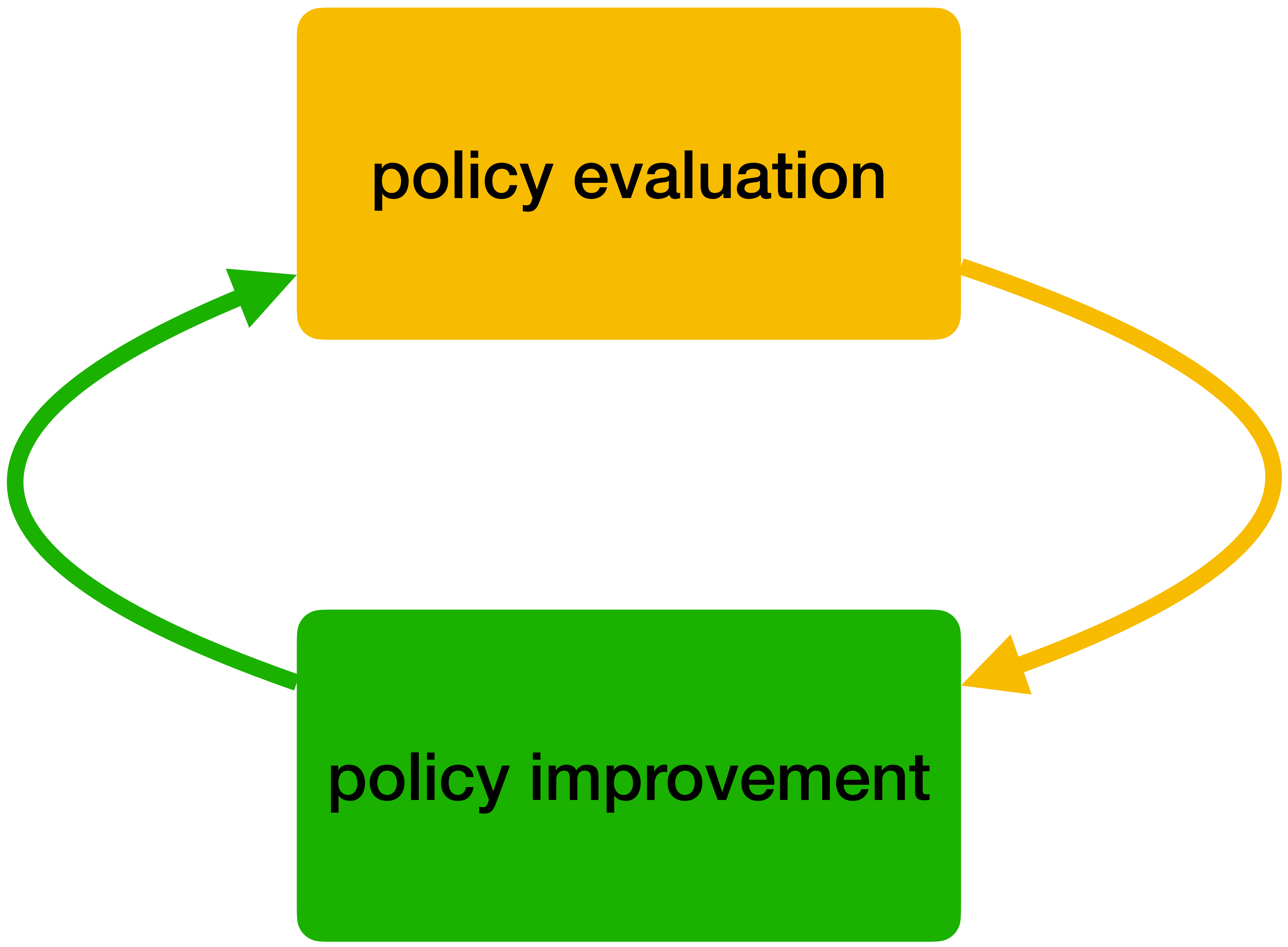
temporal difference between $V(s')$ and $V(s)$

Q function

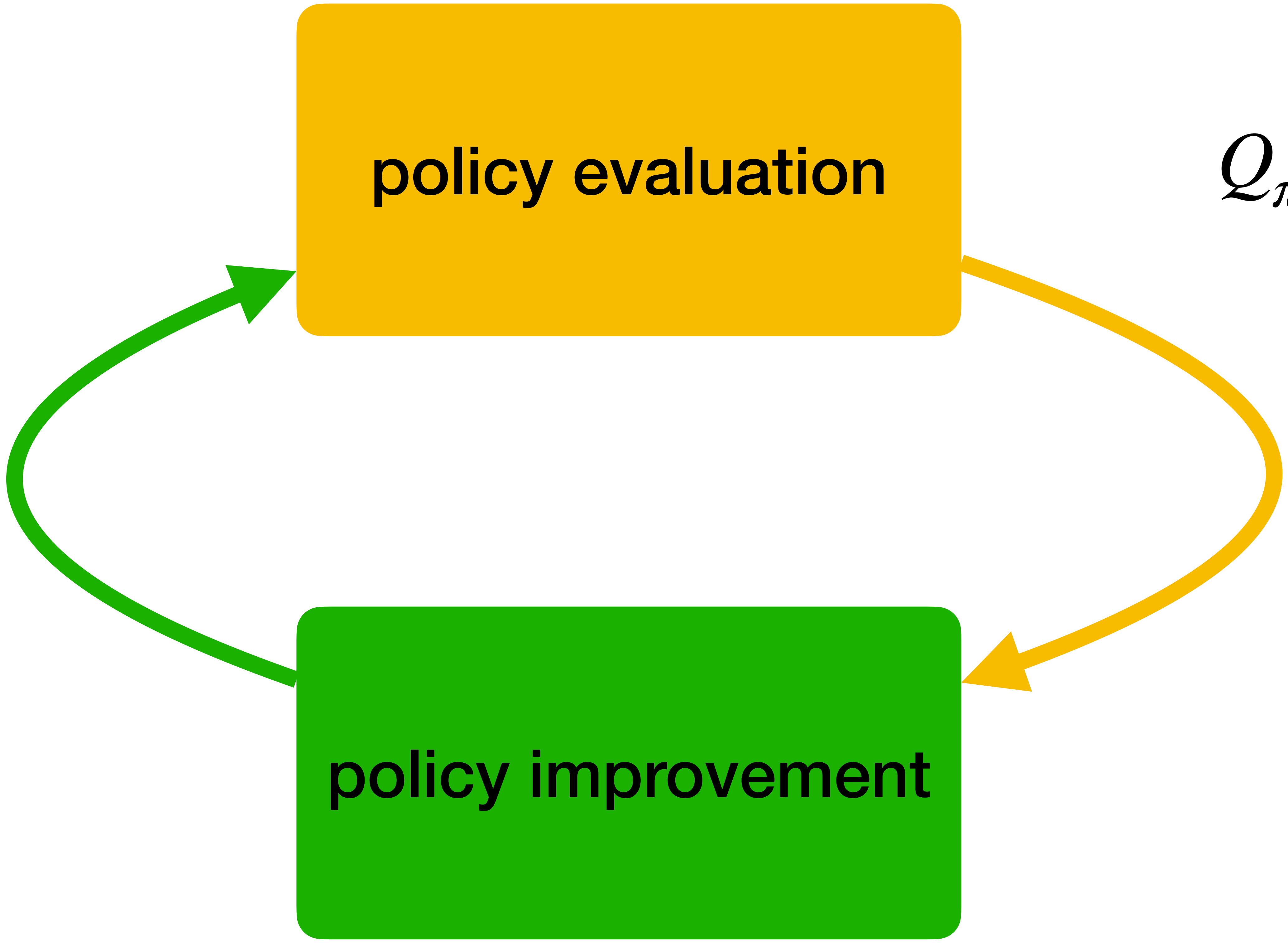
- To approach V_π when we update $V(s) \to r + \gamma V(s')$, we need on-policy data
  - Roll out π to see transition $(s, a) \to s'$ with reward r
- On-policy data is expensive: need more every time π changes
- Action-value function: $Q_\pi(s, a) = \mathbb{E}_{\xi \sim p_\pi}[R \,|\, s_0 = s, a_0 = a]$
  - Compare: $V_\pi(s) = \mathbb{E}_{\xi \sim p_\pi}[R \,|\, s_0 = s] = \mathbb{E}_{(a|s) \sim \pi}[Q_\pi(s, a)]$

- Action-value backward recursion: $Q_\pi(s, a) = r(s, a) + \gamma \mathbb{E}_{(s'|s,a) \sim p}[V_\pi(s')]$
- Advantage: $A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s) =$ benefit of counterfactual a

The RL scheme



Q-learning



$$V_\pi(s') = \max_{a'} Q_\pi(s', a')$$

$$Q_\pi(s, a) \to r(s, a) + \gamma V_\pi(s')$$

$$\pi(s) = \arg\max_a Q(s, a)$$

Q-Learning

Algorithm Q-Learning

Initialize Q

$s \leftarrow$ reset state

repeat

Take some action a

Receive reward r

Observe next state s'

Update $Q(s, a) \rightarrow \begin{cases} r & s' \text{ terminal} \\ r + \gamma \max_{a'} Q(s', a') & \text{otherwise} \end{cases}$

$s \leftarrow$ reset state if s' terminal, else $s \leftarrow s'$

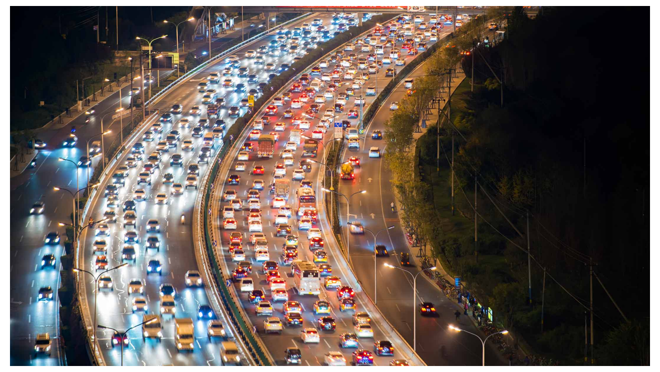
[Watkins and Dayan, 1992]

After the break:
Deep RL

Experience policy

- Which distribution should the training data have?
  - The policy may not be good on other distributions / unsupported states
  - ⇒ ideally, the test distribution p_π for the final π
- On-policy methods (e.g. MC): must use on-policy data (from the current π)
- Off-policy methods (e.g. Q) can use different policy (or even non-trajectories)
  - But both should eventually use p_π or suffer train–test distribution mismatch

Exploration policies



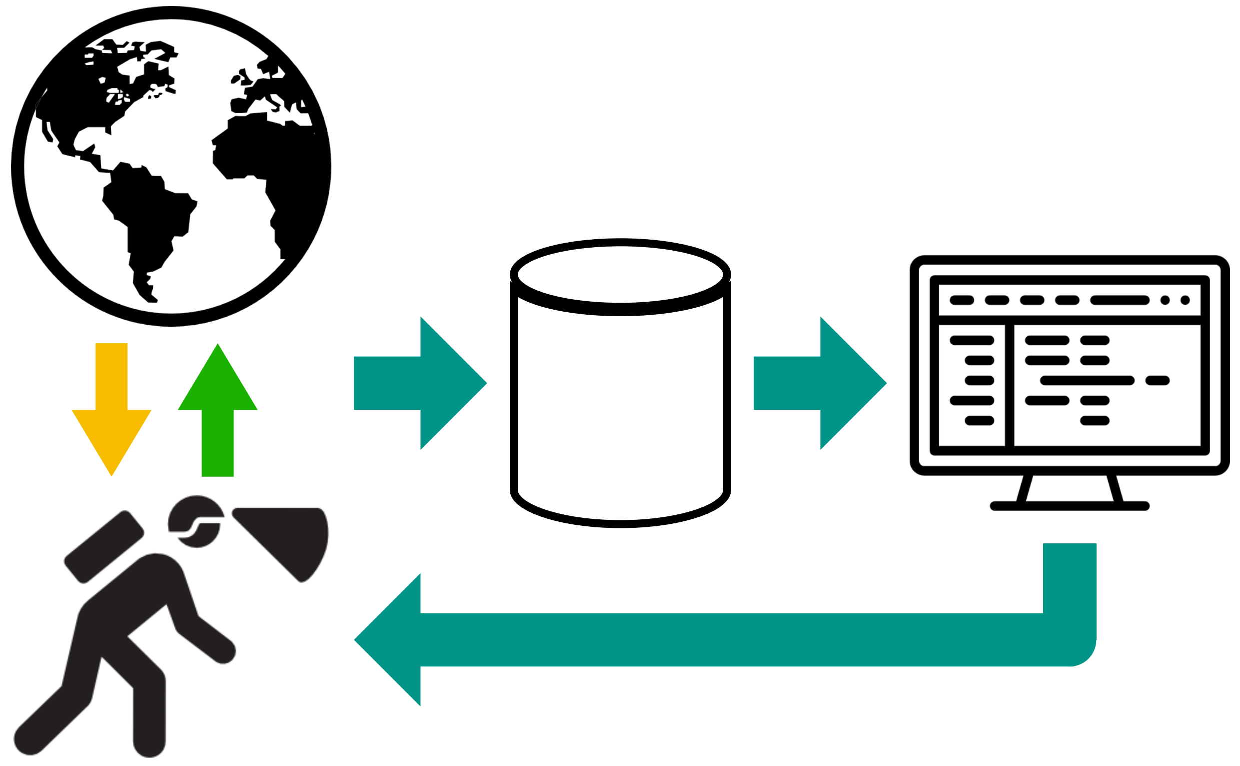
- Example: I tried route 1: {40, 20, 30}; route 2: {30, 25, 40}
  - Suppose route 1 really has expected time 30min, should you commit to it forever?
- To avoid overfitting, we must try all actions infinitely often
- ϵ-greedy exploration: select uniform action with prob. ϵ, otherwise greedy
- Boltzmann exploration:

$$\pi(a|s) = \operatorname*{soft\,max}_a(Q(s,a);\beta) = \frac{\exp(\beta Q(s,a))}{\sum_{\bar{a}} \exp(\beta Q(s,\bar{a}))}$$

  - Becomes uniform as the inverse temperature $\beta \to 0$, greedy as $\beta \to \infty$

Experience replay

- On-policy methods are inefficient: throw out all data with each policy update
- Off-policy methods can keep the data = experience replay
  - Replay buffer: dataset of past experience
  - Diversifies the experience (beyond current trajectory)
- Variants differ on
  - How often to add data vs. sample data
  - How to sample from the buffer
  - When to evict data from the buffer, and which

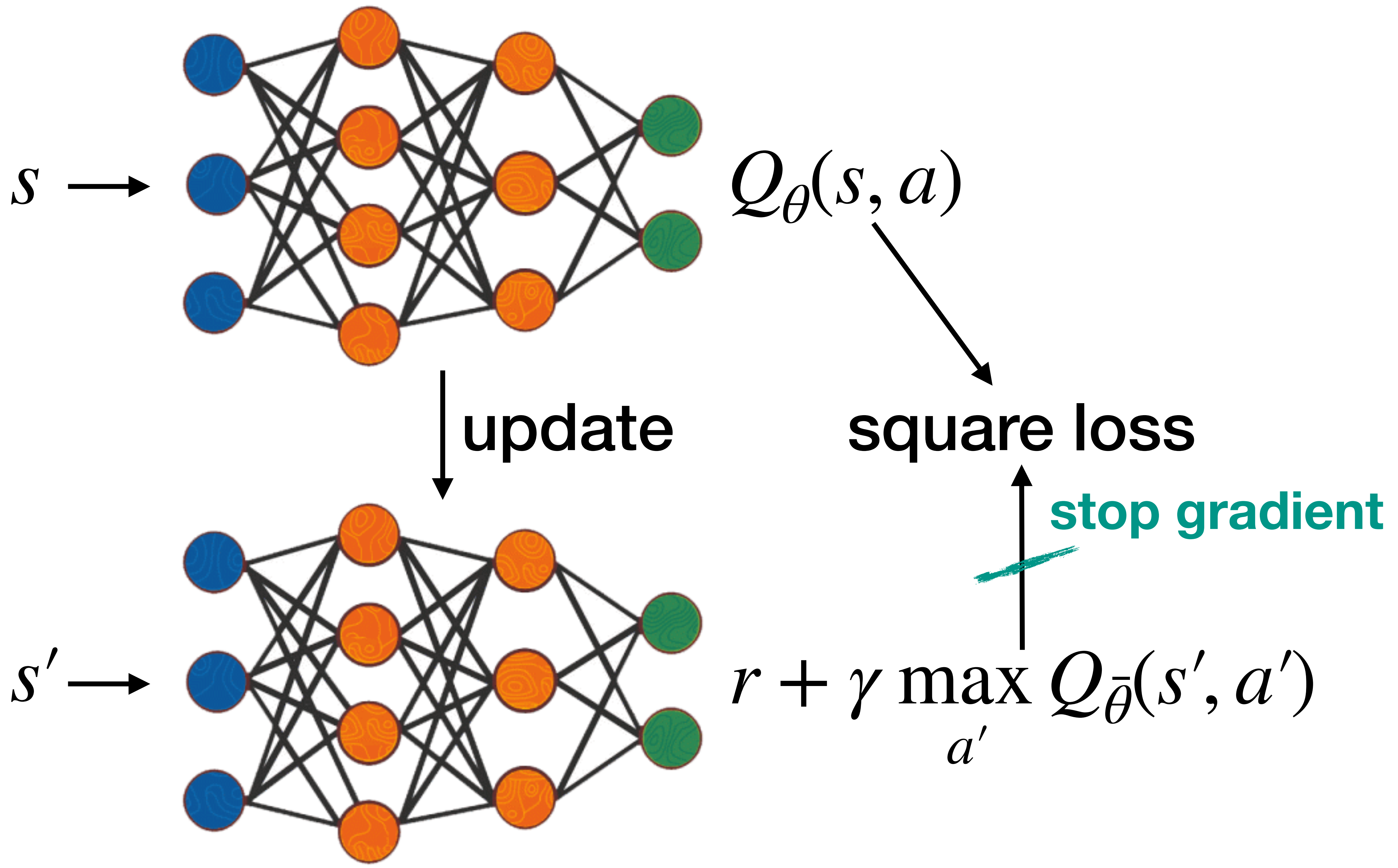


Why use target network?

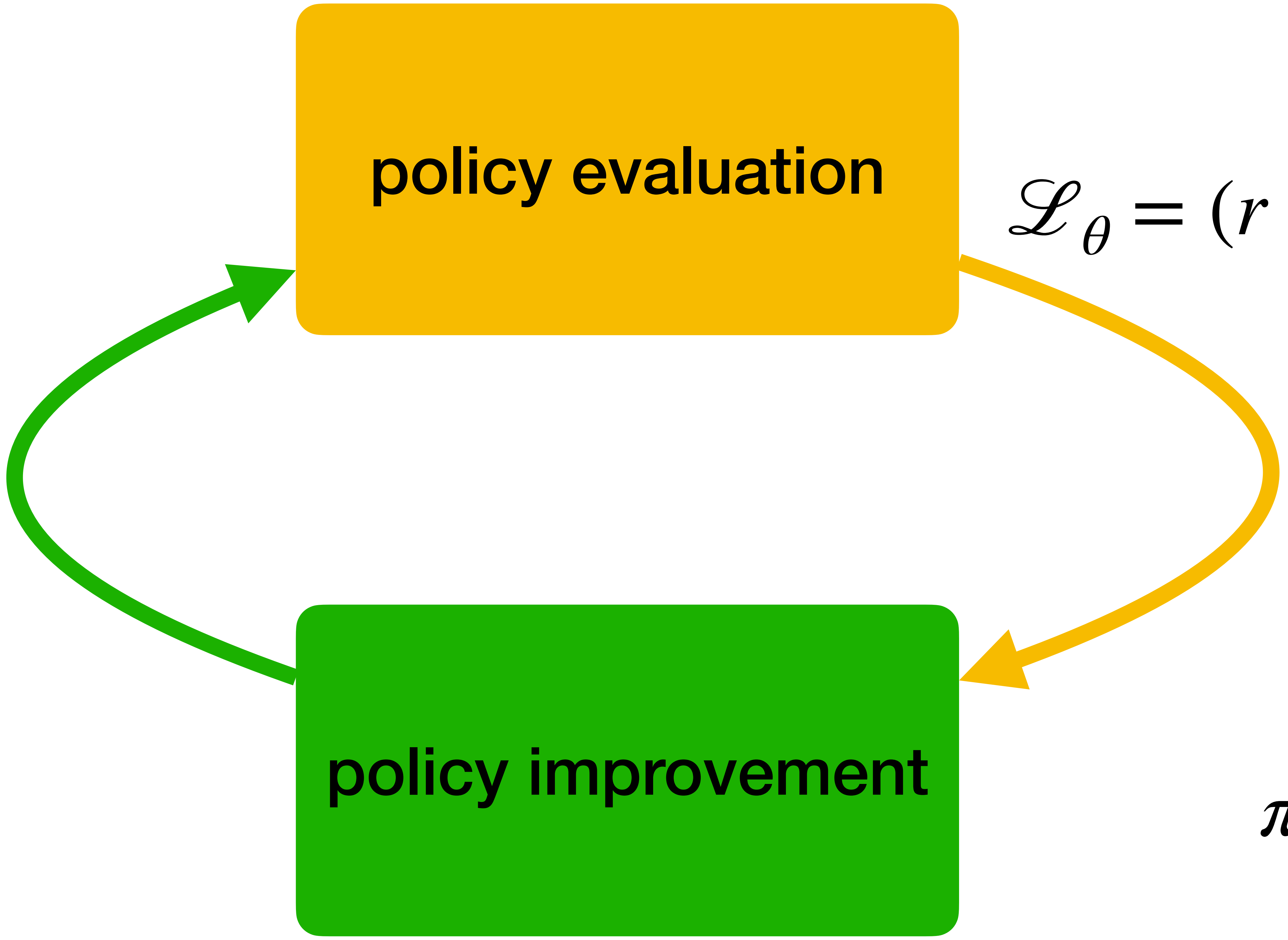
- Fitted-Q loss: $\mathscr{L}_\theta = (r + \gamma \max_{a'} Q_{\bar\theta}(s', a') - Q_\theta(s, a))^2$

  no gradient from the target term

- Target network = lagging copy of $Q_\theta(s, a)$
  - Periodic update: $\bar\theta \leftarrow \theta$ every T_{target} steps
  - Exponential update: $\bar\theta \leftarrow (1 - \eta)\bar\theta + \eta\theta$
- $Q_{\bar\theta}$ is more stable
  - Less of a moving target
  - Less sensitive to data ⇒ less variance
- But $\bar\theta \neq \theta$ introduces bias



Putting it all together: DQN



fitted-Q loss

$$\mathscr{L}_\theta = (r + \gamma \max_{a'} Q_{\bar{\theta}}(s', a') - Q_\theta(s, a))^2$$

greedy policy

$$\pi(s) = \arg\max_a Q(s, a)$$

exploration

e.g. ϵ-greedy

Deep Q-Learning (DQN)

Algorithm DQN

Initialize θ, set $\bar{\theta} \leftarrow \theta$

$s \leftarrow$ reset state

for each interaction step

Sample $a \sim \epsilon$-greedy for $Q_\theta(s, \cdot)$

Get reward r and observe next state s'

Add (s, a, r, s') to replay buffer $\mathcal{D}$

Sample batch $(\vec{s}, \vec{a}, \vec{r}, \vec{s}') \sim \mathcal{D}$

$$y_i \leftarrow \begin{cases} r_i & s'_i \text{ terminal} \\ r_i + \gamma \max_{a'} Q_{\bar{\theta}}(s'_i, a') & \text{otherwise} \end{cases}$$

Descend $\mathcal{L}_\theta = (\vec{y} - Q_\theta(\vec{s}, \vec{a}))^2$

every T_{target} steps, set $\bar{\theta} \leftarrow \theta$

$s \leftarrow$ reset state if s' terminal, else $s \leftarrow s'$

[Mnih et al., 2015]

Today's lecture

Behavior Cloning

Temporal Difference

Policy Gradient

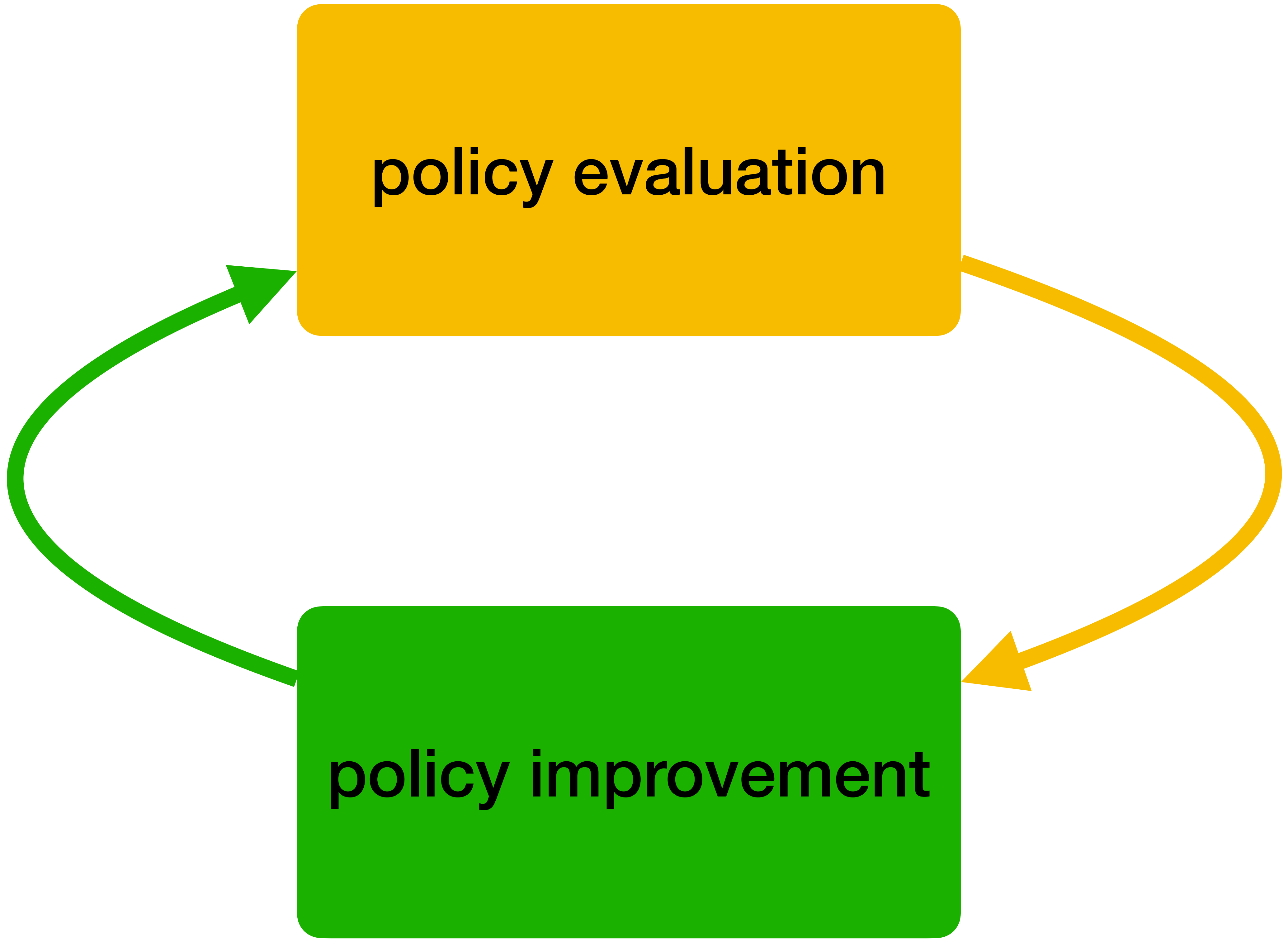
and more...

Value-based vs. policy-based methods

value-based

$Q_\theta(s,a)$

$\arg\max_a Q_\theta(s,a)$



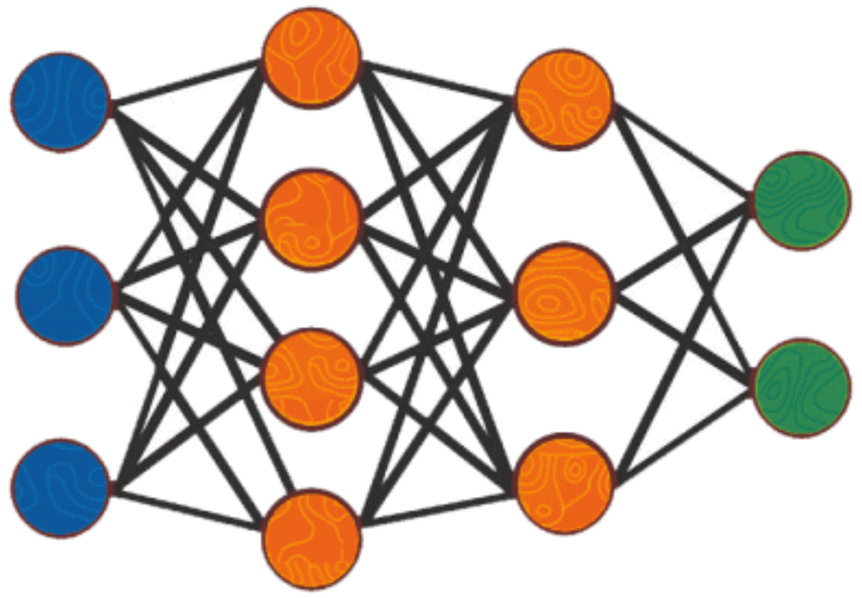
policy-based

$\mathbb{E}_{\xi\sim p_\theta}[R(\xi)]$

$\pi_\theta(a|s)$

Policy Gradient (PG)

- Gradient-based learning: $\theta \to \theta - \nabla_\theta \mathbb{E}_{x\sim D}[\mathscr{L}_\theta(x)]$
  - Expectation gradient = expected gradient, estimate with samples



- Policy-Gradient RL: $\theta \to \theta + \nabla_\theta J_\theta$, with $J_\theta = \mathbb{E}_{\xi\sim p_\theta}[R]$
  - Can we also use samples $\xi \sim p_\theta$ to estimate $\nabla_\theta J_\theta$?
- The sampling distribution itself depends on θ

  - Problem 1: data must be on-policy
  - Problem 2: cannot backprop gradient through samples

Score-function gradient estimation

- Log-derivative + chain rule: $\nabla_\theta \log p_\theta(\xi) = \frac{1}{p_\theta(\xi)} \nabla_\theta p_\theta(\xi)$

- Log-derivative / score-function / REINFORCE trick:

$$\nabla_\theta J_\theta = \sum_\xi R(\xi) \nabla_\theta p_\theta(\xi)$$

$$= \sum_\xi R(\xi) p_\theta(\xi) \nabla_\theta \log p_\theta(\xi)$$

$$= \mathbb{E}_{\xi \sim p_\theta}[R(\xi) \nabla_\theta \log p_\theta(\xi)]$$

  - Allows estimating $\nabla_\theta J_\theta$ using samples $\xi \sim p_\theta$

REINFORCE

- To find $\nabla_\theta J_\theta = \mathbb{E}_{\xi\sim p_\theta}[R(\xi)\,\nabla_\theta \log p_\theta(\xi)]$, sample $\xi \sim p_\theta$, then:

$$\nabla_\theta \log p_\theta(\xi) = \nabla_\theta \left(\log p(s_0) + \sum_t \log \pi_\theta(a_t \mid s_t) + \log p(s_{t+1} \mid s_t, a_t) \right)$$

$$= \nabla_\theta \sum_t \log \pi_\theta(a_t \mid s_t)$$

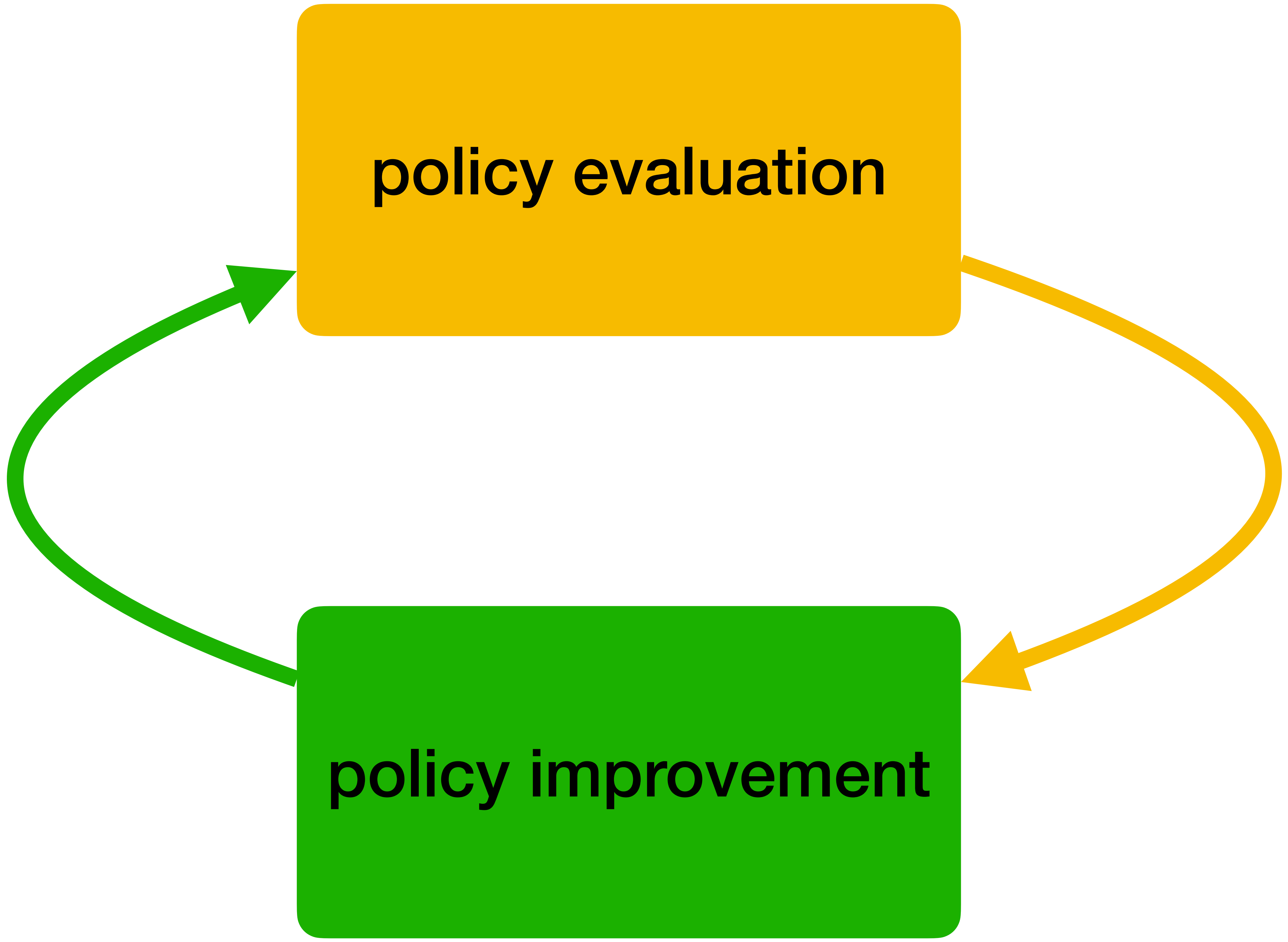
  - Model-free, but on-policy and high variance (like MC)

Algorithm REINFORCE

Initialize π_θ
repeat
  Roll out $\xi \sim p_\theta$
  Update with gradient $g \leftarrow R(\xi) \sum_t \nabla_\theta \log \pi_\theta(a_t | s_t)$

[Williams, 1992]

Actor–Critic (AC) methods



critic $Q_\phi(s, a)$
evaluates actor

actor $\pi_\theta(a \mid s)$
uses critic to improve

Advantage Actor–Critic (A2C)

Algorithm Advantage Actor–Critic

Initialize π_θ and V_ϕ

repeat

Roll out $\xi \sim p_\theta$

Update $\Delta\theta \leftarrow \sum_t (R_{\geq t}(\xi) - V_\phi(s_t)) \nabla_\theta \log \pi_\theta(a_t | s_t)$

Descend $L_\phi = \sum_t (R_{\geq t}(\xi) - V_\phi(s_t))^2$

[Mnih et al., 2016]

Importance Sampling

- Suppose you want to estimate $\mathbb{E}_{x\sim p}[f(x)]$
  - but only have samples $x \sim p'$

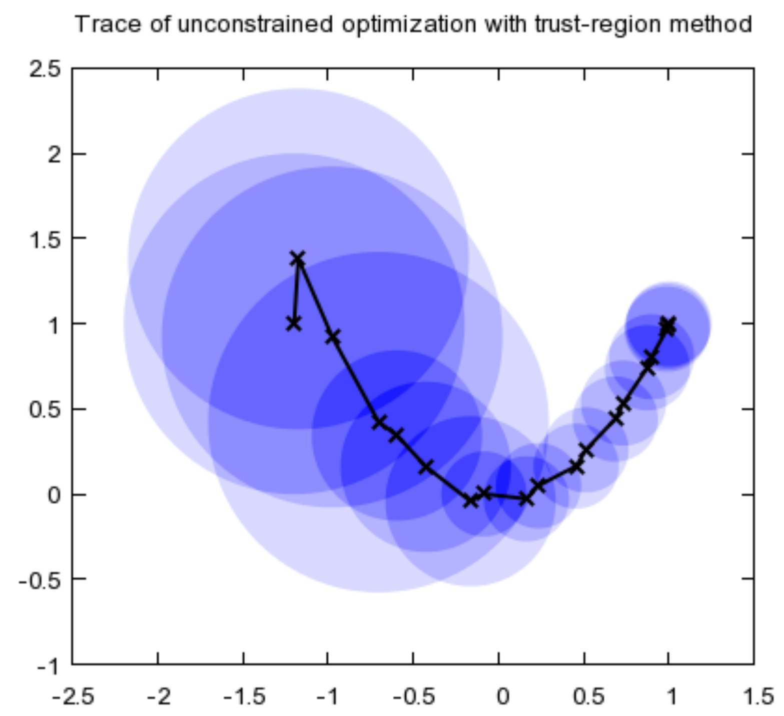
- Importance sampling:

$$\mathbb{E}_{x\sim p}[f(x)] = \mathbb{E}_{x\sim p'}\left[\frac{p(x)}{p'(x)} f(x)\right]$$

  - Importance (IS) weights: $\rho(x) = \frac{p(x)}{p'(x)}$
  - Estimate: $\rho(x) f(x)$ with $x \sim p'$

Finding best next policy

- Performance Difference Lemma: $J_\pi - J_{\bar\pi} = \sum_t \gamma^t \mathbb{E}_{(s_t,a_t)\sim p_\pi}[A_{\bar\pi}(s_t, a_t)]$
  - Idea: with current policy $\bar\pi$, find $\max_\pi J_\pi - J_{\bar\pi}$ by maximizing the RHS



- Step 1: use $\bar\pi$ to evaluate $A_{\bar\pi}$; step 2: estimate $\mathbb{E}_{(s_t,a_t)\sim p_\pi}[A_{\bar\pi}(s_t, a_t)]$
  - But we don't have data $(s_t, a_t) \sim p_\pi$; idea: sample from $\bar\pi$

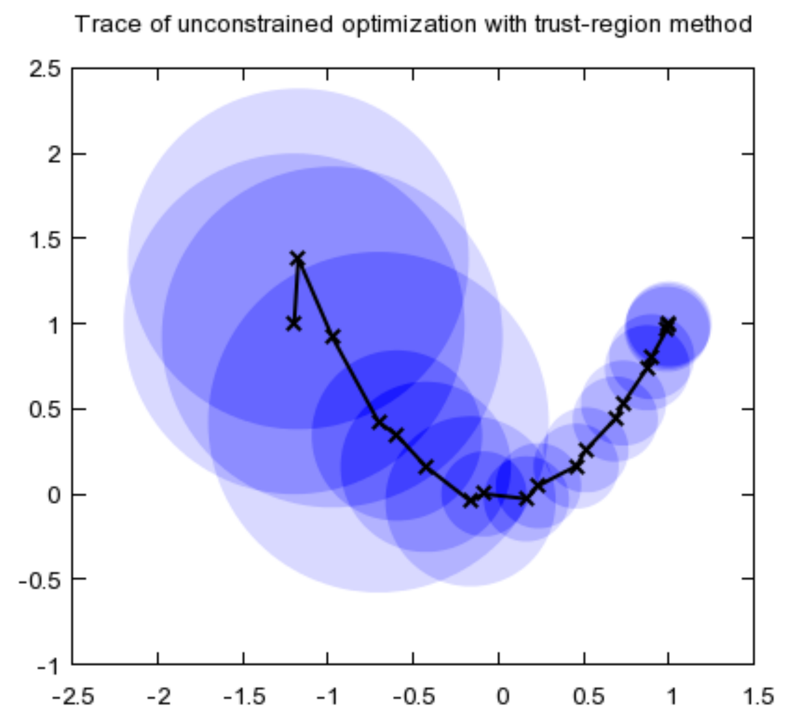
$$\max_\pi \sum_t \gamma^t \mathbb{E}_{\xi_{\leq t}\sim p_{\bar\pi}}[\rho^\pi_{\bar\pi}(\xi_{\leq t}) A_{\bar\pi}(s_t, a_t)]$$

high variance!

- When is it reasonable to use $\rho^\pi_{\bar\pi}(a_t \mid s_t) = \frac{\pi(a_t \mid s_t)}{\bar\pi(a_t \mid s_t)}$ instead? i.e. drop $\rho^\pi_{\bar\pi}(\xi_{<t}) = \prod_{t'<t} \frac{\pi(a_{t'} \mid s_{t'})}{\bar\pi(a_{t'} \mid s_{t'})}$
  - Intuitively, when $\mathbb{D}[\bar\pi_\theta(a \mid s) \| \pi(a \mid s)]$ is small

Proximal Policy Optimization (PPO)

- Idea: ascend $\mathbb{E}_{(s,a)\sim p_{\bar\theta}}[\rho^\theta_{\bar\theta}(a|s)A_{\bar\theta}(s,a)]$ with π_θ staying near $\pi_{\bar\theta}$
  - PPO-Penalty: add a penalty term for $\mathbb{E}_{s\sim p_{\bar\theta}}[\mathbb{D}[\pi_{\bar\theta}(a|s)\|\pi_\theta(a|s)]]$
  - PPO-Clip: ascend $\mathbb{E}_{(s,a)\sim p_{\bar\theta}}[L^\theta_{\bar\theta}(s,a)]$ with



$$L^\theta_{\bar\theta}(s,a) = \min(\rho^\theta_{\bar\theta}(a|s)A_{\bar\theta}(s,a), A_{\bar\theta}(s,a) + |\epsilon A_{\bar\theta}(s,a)|)$$

- Positive / negative advantage ⇒ increase / decrease $\rho^\theta_{\bar\theta}(a|s) = \dfrac{\pi_\theta(a|s)}{\pi_{\bar\theta}(a|s)}$
  - But no incentive beyond $\rho^\theta_{\bar\theta}(a|s) = 1 \pm \epsilon$

- **no incentive ≠ doesn't happen**
- **PPO has lots more tricks to limit divergence**

[Schulman et al., 2017]

Today's lecture

Behavior Cloning

Temporal Difference

Policy Gradient

and more...

Bounded optimality

- Bounded optimizer = trades off value and divergence from prior $\pi_0(a \mid s)$

$$\max_\pi \mathbb{E}_{(s,a)\sim p_\pi}[r(s,a)] - \tau \mathbb{D}[\pi \| \pi_0] = \max_\pi \mathbb{E}_{(s,a)\sim p_\pi}\left[r(s,a) - \tau \log \frac{\pi(a \mid s)}{\pi_0(a \mid s)}\right]$$

- $\tau = \frac{1}{\beta}$ is the tradeoff coefficient between value and relative entropy
  - As $\tau \to \infty$, the agent will fall back to the prior $\pi \to \pi_0$
  - As $\tau \to 0$, the agent will be a perfect value optimizer $\pi \to \pi^*$
- Early in training, τ should be finite to avoid overfitting
- Bellman recursion: $V(s) = \max_\pi \mathbb{E}_{(a|s)\sim\pi}\left[r(s,a) - \tau \log \frac{\pi(a \mid s)}{\pi_0(a \mid s)} + \gamma \mathbb{E}_{(s'|s,a)\sim p}[V(s')]\right]$

optimal $\pi \propto \pi_0 \exp(\beta Q(s,a))$

Soft Actor–Critic (SAC)

- Optimally: $\pi(a \mid s) = \dfrac{\pi_0(a \mid s) \exp \beta Q(s, a)}{\exp \beta V(s)}$ $\qquad V(s) = Q(s, a) - \dfrac{1}{\beta} \log \dfrac{\pi(a \mid s)}{\pi_0(a \mid s)}$
- In continuous action spaces, we can't explicitly softmax $Q(s, a)$ over a
- We can train a critic off-policy

$$L_\phi(s, a, r, s', a') = \left(r + \gamma \left(Q_{\bar{\phi}}(s', a') - \frac{1}{\beta} \log \frac{\pi_\theta(a' \mid s')}{\pi_0(a' \mid s')} \right) - Q_\phi(s, a) \right)^2$$

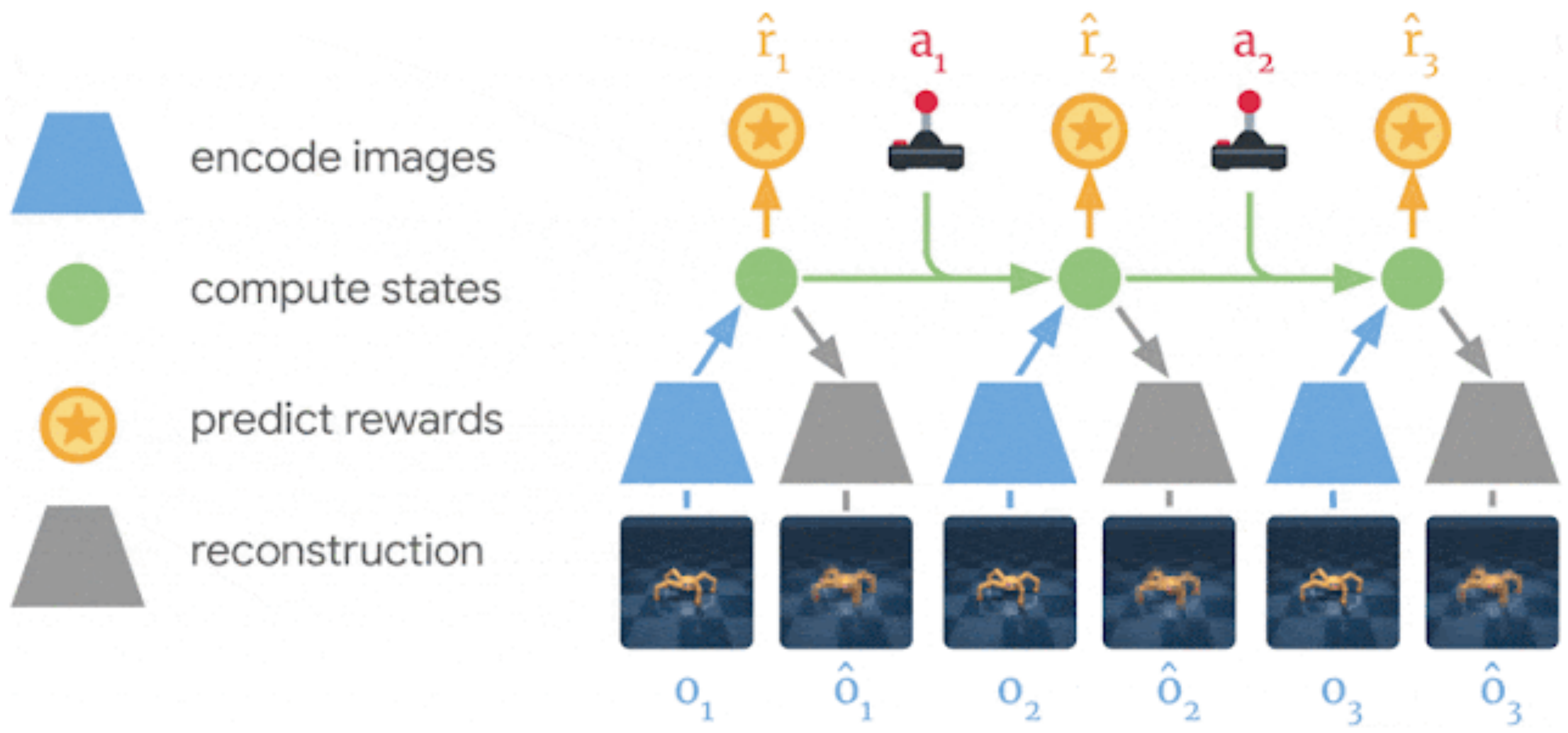
- And a soft-greedy actor = imitate the critic

$$L_\theta(s) = \mathbb{E}_{(a|s) \sim \pi_\theta}[\log \pi_\theta(a \mid s) - \log \pi_0(a \mid s) - \beta Q_\phi(s, a)]$$

- Can optimize τ to match a target entropy $L_\tau(s, a) = -\tau \log \pi_\theta(a \mid s) - \tau H$

Dreamer

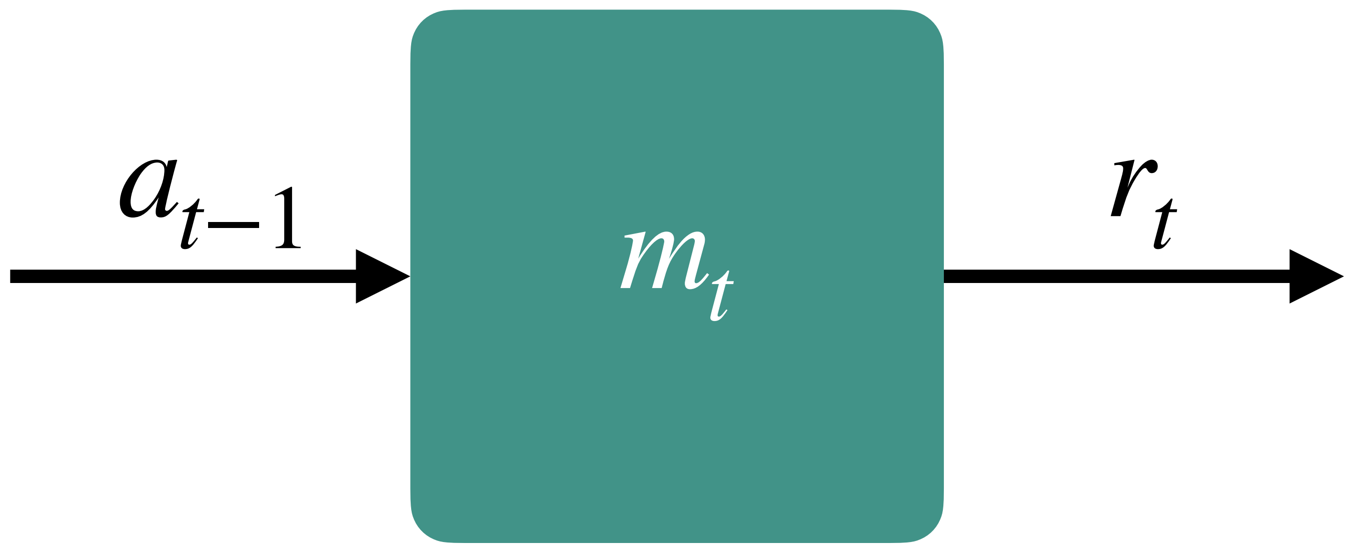
- Dreamer learns a latent state process to
  - Reconstruct observation
  - Predict reward
  - Predict next latent state distribution
- Then performs RL in this model
  - We really only need the rewards and transitions
  - Reconstruction is an auxiliary task



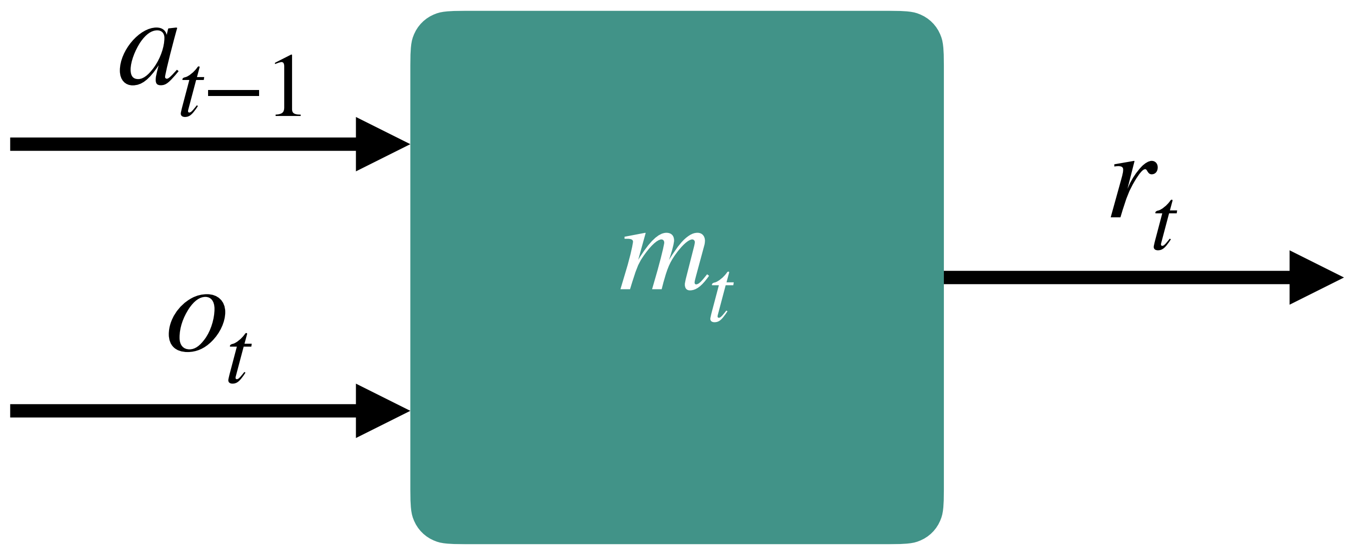
[Hafner et al., Mastering Diverse Domains through World Models, 2023]

The interface of a world model

- We would like a model where we can run RL algorithms
  - That gives the same $\mathbb{E}_{\xi \sim p_\pi}[R(\xi)]$ as the world for all π

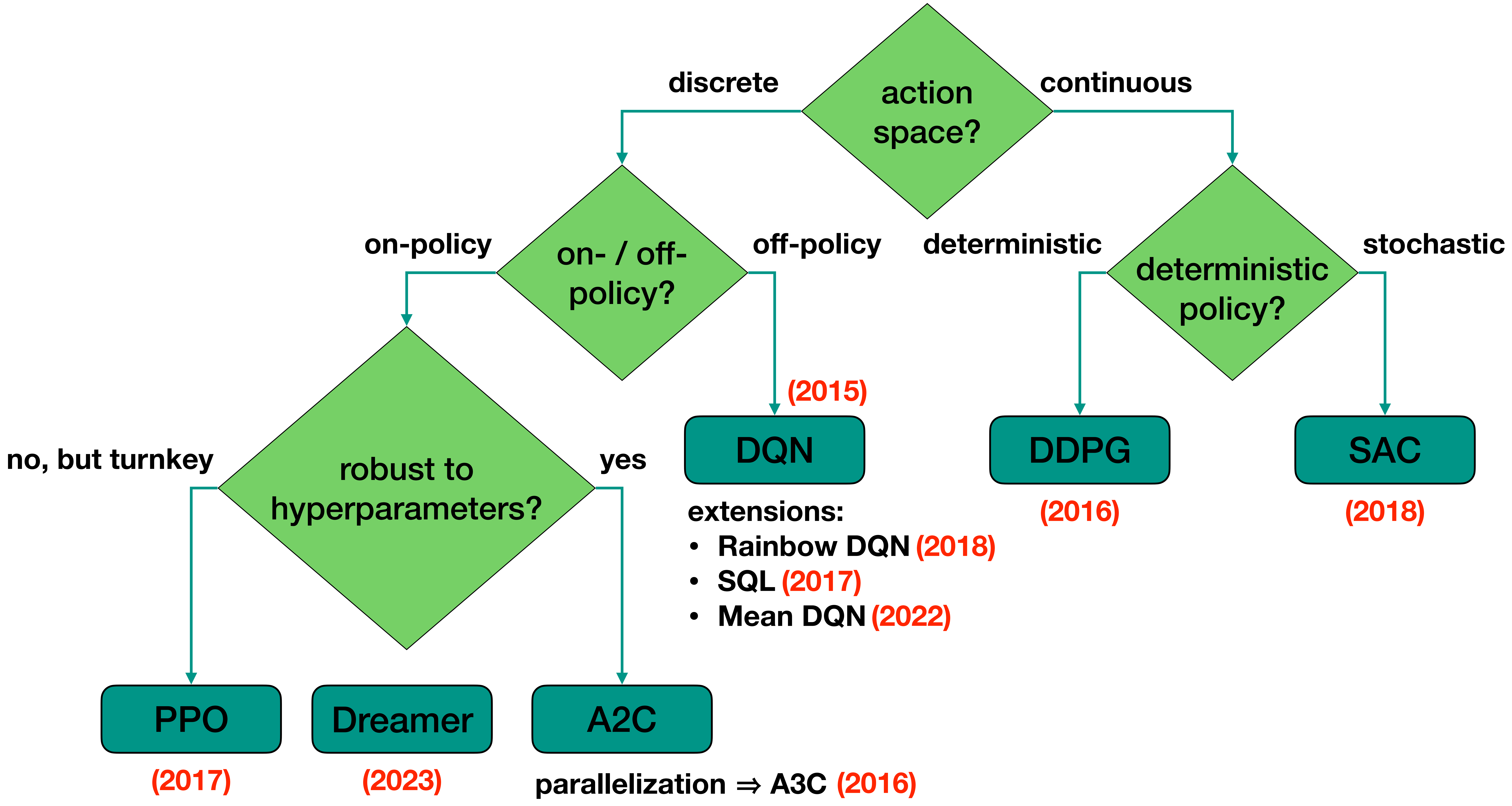


- But that's not possible without seeing the observations
  - But here we can't run RL in imagination



- How to keep the imagination and interaction modes matched?
  - Method 1: in imagination, predict o_{t+1} (or its embedding) and feed it back
  - Method 2: keep $\hat{p}(m_t | a_{<t})$ and $\mathbb{E}_{o_{\leq t}|a_{<t} \sim p}[\hat{p}(m_t | o_{\leq t}, a_{<t})]$ close

Flowchart: which algorithm to choose?



Why so many algorithms?

- We may have different modeling assumptions
  - Is the environment stochastic or deterministic?
  - Is the state / action space continuous or discrete?
  - Is the horizon episodic or infinite?
- We may care about different tradeoffs
  - Sample efficiency? Computational efficiency while learning / executing? Succinct representation?
  - Algorithmic stability, reproducibility, ease of use (existing code), ease of adaptation
- Different difficulty to represent or learn in different domains
  - Represent / learn a policy or a model?
  - Discover structure? Memory? Transfer / share with other tasks? Non-stationarity / multi-agent?

On- or off-policy data?

- The faster our simulator ⇒ the faster we can refresh our data
  - And still keep sufficient diversity for training
- Fast enough ⇒ can use on-policy data
  - No need for replay buffer
  - No train⟶test distributional mismatch (= covariate shift)
  - Can still use off-policy algorithms with on-policy data
- Extremely slow simulator ⇒ not even off-policy, just offline RL

Reward shaping

- Ideal reward: $r(s, a) = -\infty$ for any suboptimal action $\implies$ as hard to provide as π^*
  - We need supervision signal that's sufficiently easy to program $\implies$ generate more data
- Sparse reward functions may be easier than dense ones
  - E.g., may be easy to identify good goal states, safety violations, etc.
- Reward shaping: art of adjusting the reward function for easier RL; some tips:
  - Reward "bottleneck states": subgoals that are likely to lead to bigger goals
  - Break down long sequences of coordinated actions $\implies$ better exploration
    - E.g. reward beacons on long narrow paths, for exploration to stumble upon

Logistics

assignments

- Exercise 1 is due next Wednesday (individual)
- Project proposals are due next Friday (team)

meetings

- Meet the instructor at least once by next Friday
- Welcome to schedule as much as you need