

# Introduction to Reinforcement Learning

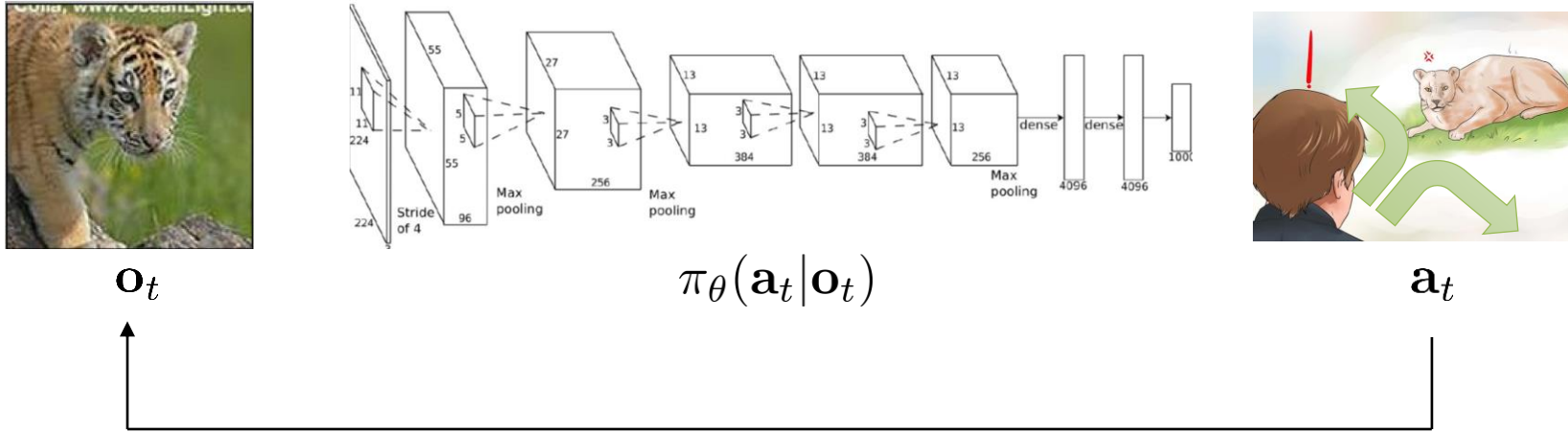
CS 285

Instructor: Sergey Levine  
UC Berkeley



# Definitions

# Terminology & notation



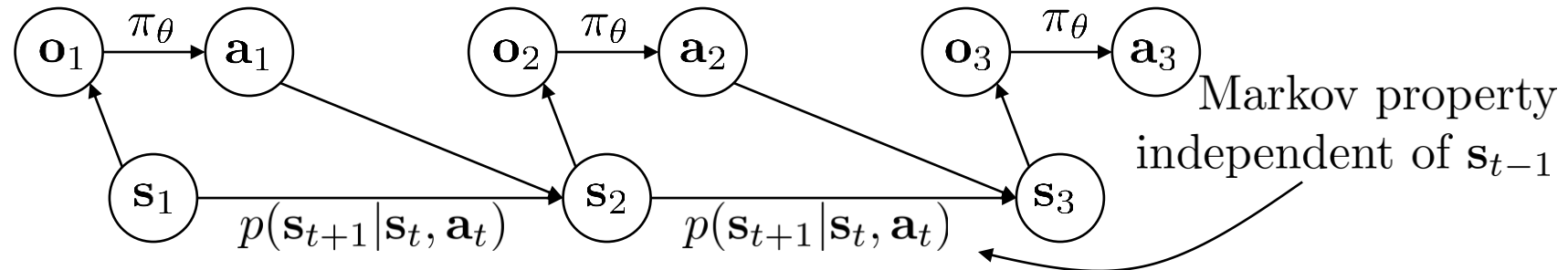
$\mathbf{s}_t$  – state

$\mathbf{o}_t$  – observation

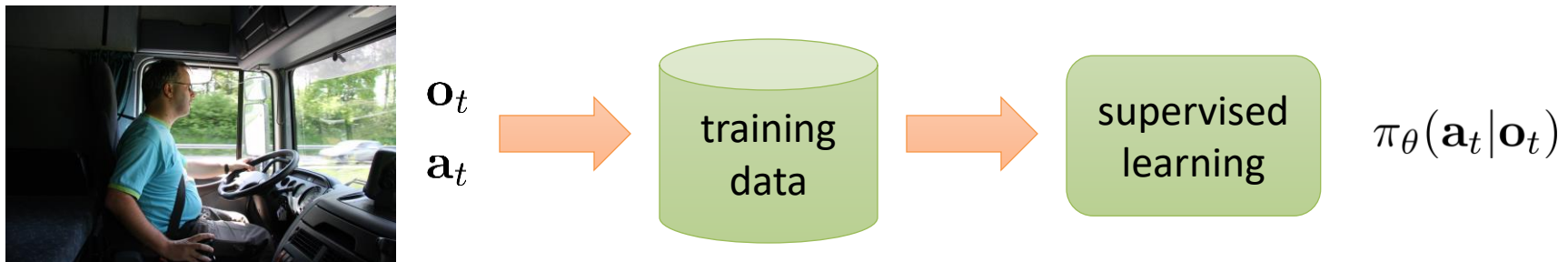
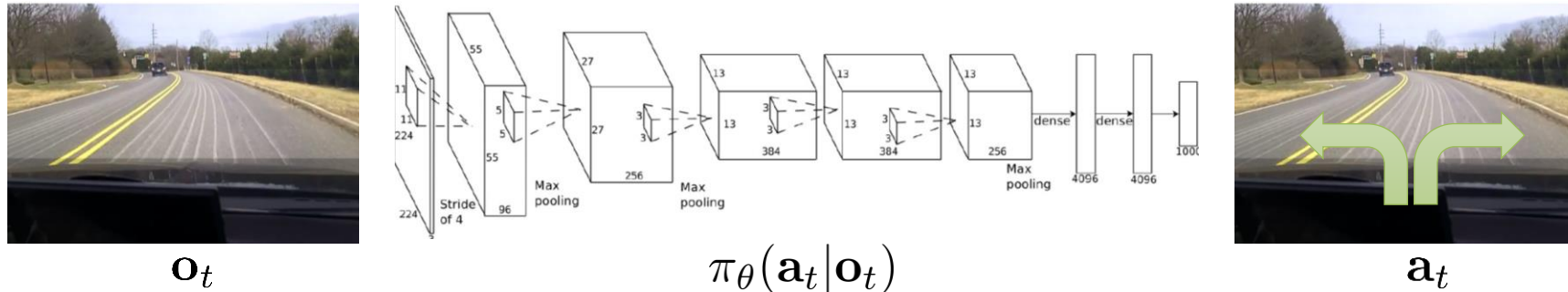
$\mathbf{a}_t$  – action

$\pi_\theta(\mathbf{a}_t | \mathbf{o}_t)$  – policy

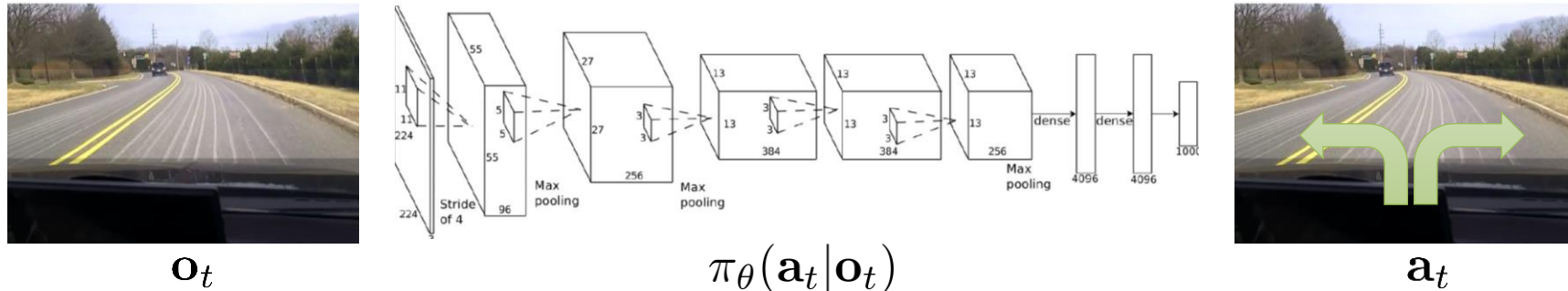
$\pi_\theta(\mathbf{a}_t | \mathbf{s}_t)$  – policy (fully observed)



# Imitation Learning



# Reward functions



which action is better or worse?

$r(\mathbf{s}, \mathbf{a})$ : reward function

tells us which states and actions are better

$\mathbf{s}$ ,  $\mathbf{a}$ ,  $r(\mathbf{s}, \mathbf{a})$ , and  $p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$  define Markov decision process



high reward



low reward

# Definitions

Markov chain

$$\mathcal{M} = \{\mathcal{S}, \mathcal{T}\}$$

$\mathcal{S}$  – state space

states  $s \in \mathcal{S}$  (discrete or continuous)

$\mathcal{T}$  – transition operator

$$p(s_{t+1}|s_t)$$

why “operator”?

$$\text{let } \mu_{t,i} = p(s_t = i)$$

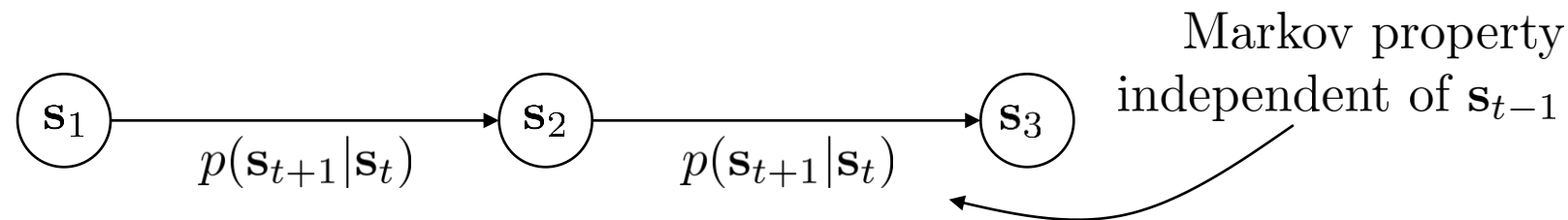
$\vec{\mu}_t$  is a vector of probabilities

$$\text{let } \mathcal{T}_{i,j} = p(s_{t+1} = i | s_t = j)$$

$$\text{then } \vec{\mu}_{t+1} = \mathcal{T} \vec{\mu}_t$$



Andrey Markov



# Definitions

Markov decision process

$$\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{T}, r\}$$

$\mathcal{S}$  – state space

states  $s \in \mathcal{S}$  (discrete or continuous)

$\mathcal{A}$  – action space

actions  $a \in \mathcal{A}$  (discrete or continuous)

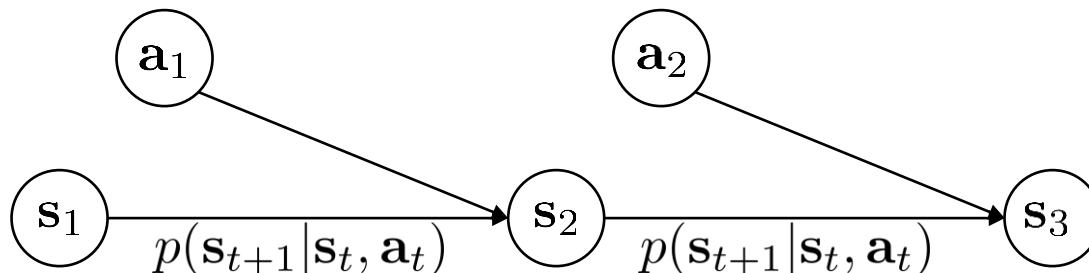
$\mathcal{T}$  – transition operator (now a tensor!)

let  $\mu_{t,j} = p(s_t = j)$

let  $\xi_{t,k} = p(a_t = k)$

let  $\mathcal{T}_{i,j,k} = p(s_{t+1} = i | s_t = j, a_t = k)$

$$\mu_{t+1,i} = \sum_{j,k} \mathcal{T}_{i,j,k} \mu_{t,j} \xi_{t,k}$$



Richard Bellman

# Definitions

Markov decision process

$$\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{T}, r\}$$

$\mathcal{S}$  – state space

states  $s \in \mathcal{S}$  (discrete or continuous)

$\mathcal{A}$  – action space

actions  $a \in \mathcal{A}$  (discrete or continuous)

$\mathcal{T}$  – transition operator (now a tensor!)

$r$  – reward function

$$r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$$

$r(s_t, a_t)$  – reward



Richard Bellman



# Definitions

partially observed Markov decision process

$$\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{T}, \mathcal{E}, r\}$$

$\mathcal{S}$  – state space

states  $s \in \mathcal{S}$  (discrete or continuous)

$\mathcal{A}$  – action space

actions  $a \in \mathcal{A}$  (discrete or continuous)

$\mathcal{O}$  – observation space

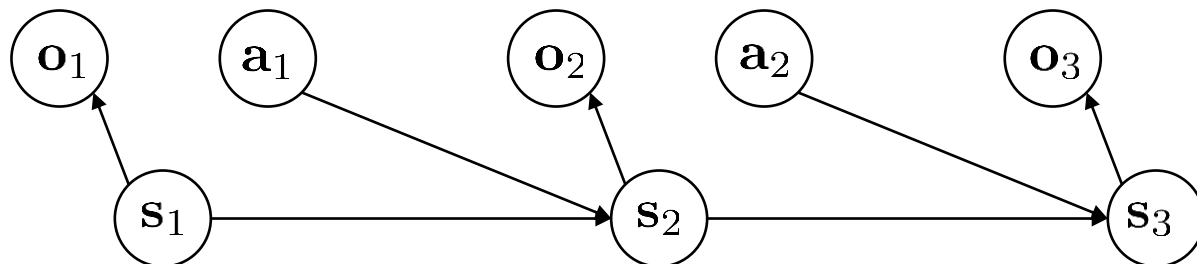
observations  $o \in \mathcal{O}$  (discrete or continuous)

$\mathcal{T}$  – transition operator (like before)

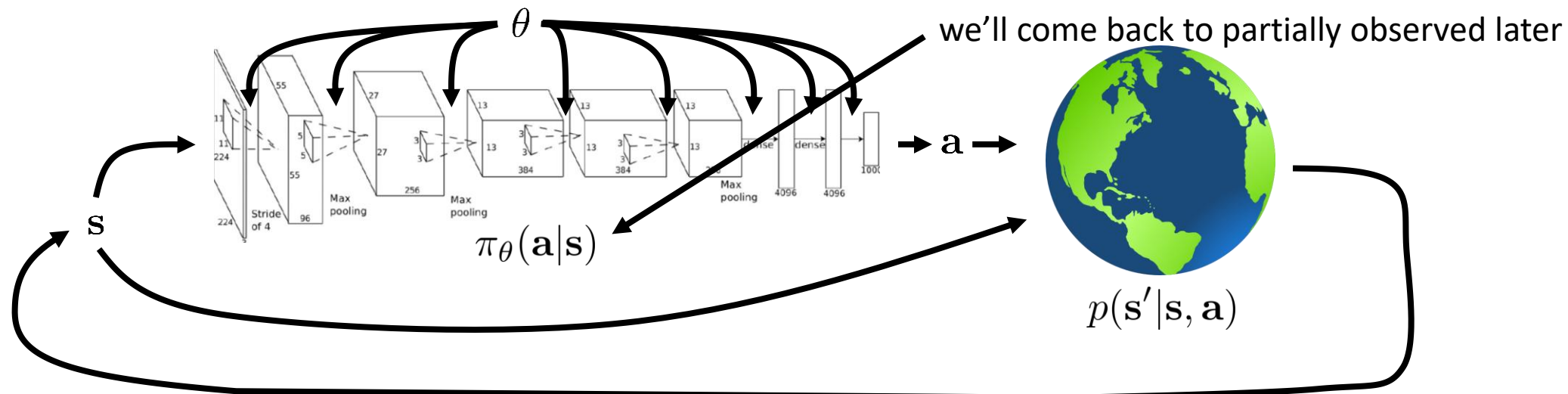
$\mathcal{E}$  – emission probability  $p(o_t | s_t)$

$r$  – reward function

$$r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$$



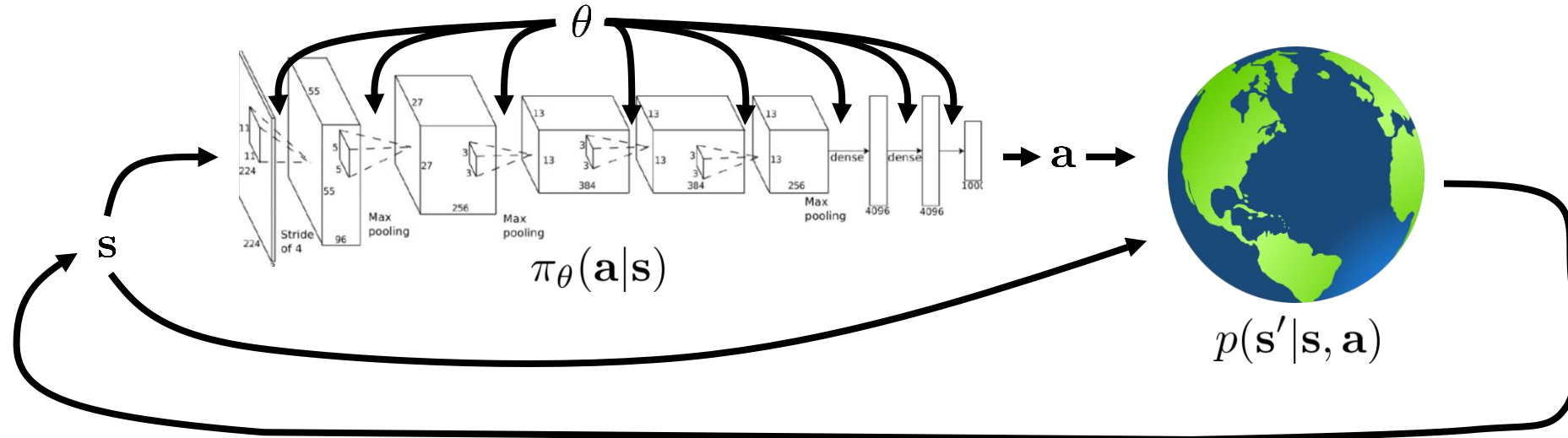
# The goal of reinforcement learning



$$\underbrace{p_{\theta}(\mathbf{s}_1, \mathbf{a}_1, \dots, \mathbf{s}_T, \mathbf{a}_T)}_{p_{\theta}(\tau)} = p(\mathbf{s}_1) \prod_{t=1}^T \pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t) p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t)$$

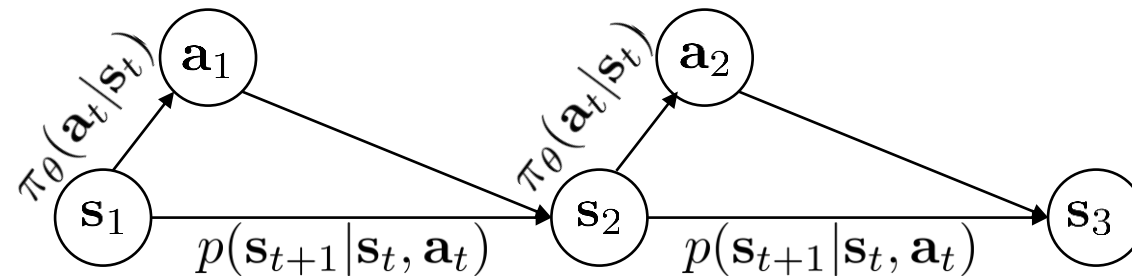
$$\theta^* = \arg \max_{\theta} E_{\tau \sim p_{\theta}(\tau)} \left[ \sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right]$$

# The goal of reinforcement learning

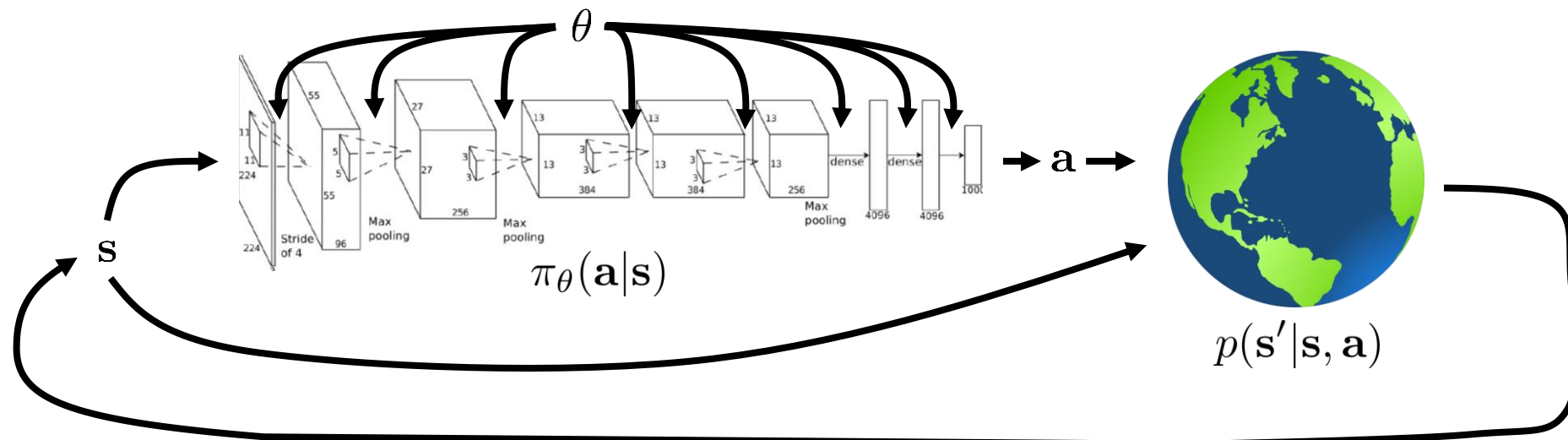


$$p_{\theta}(\mathbf{s}_1, \mathbf{a}_1, \dots, \mathbf{s}_T, \mathbf{a}_T) = p(\mathbf{s}_1) \prod_{t=1}^T \underbrace{\pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t) p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t)}_{\text{Markov chain on } (\mathbf{s}, \mathbf{a})}$$

$p_{\theta}(\tau)$



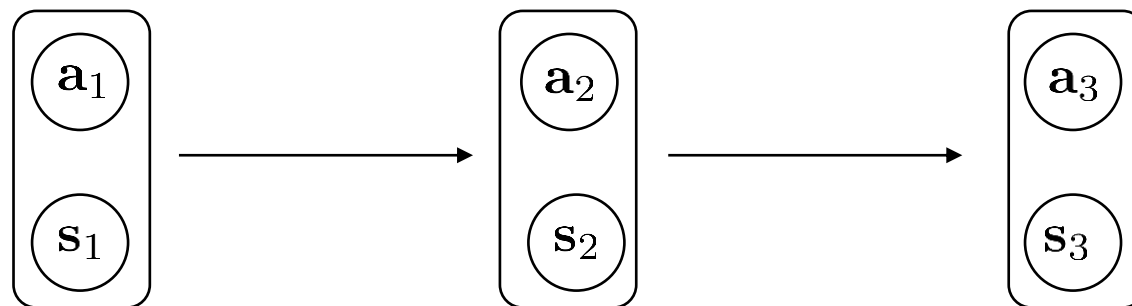
# The goal of reinforcement learning



$$p_{\theta}(\mathbf{s}_1, \mathbf{a}_1, \dots, \mathbf{s}_T, \mathbf{a}_T) = p(\mathbf{s}_1) \prod_{t=1}^T \underbrace{\pi_{\theta}(\mathbf{a}_t|\mathbf{s}_t) p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)}_{\text{Markov chain on } (\mathbf{s}, \mathbf{a})}$$

$p_{\theta}(\tau)$

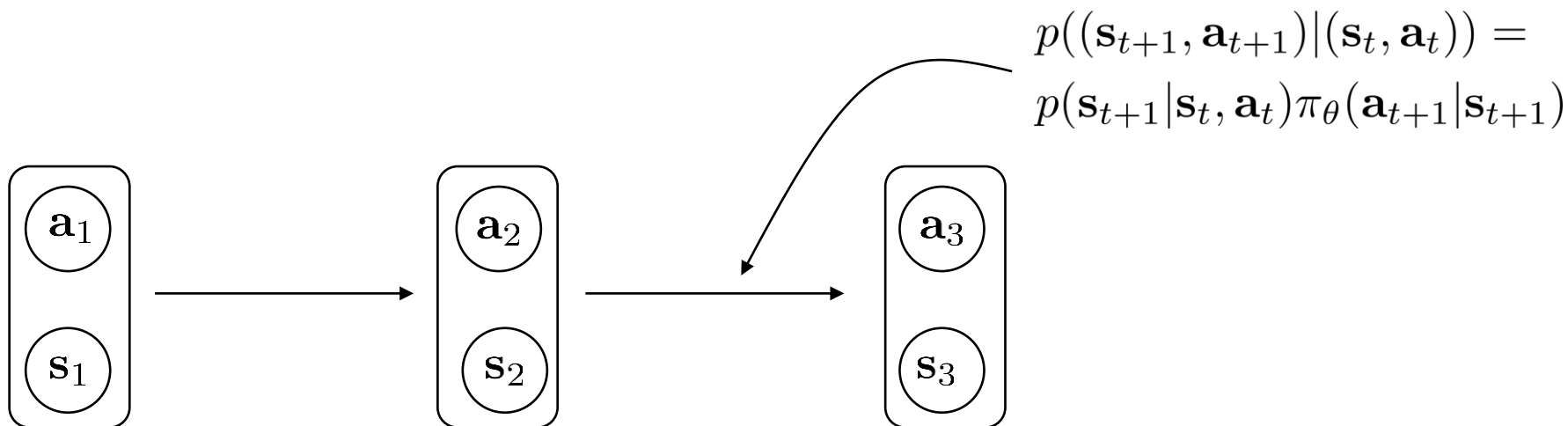
$$p((\mathbf{s}_{t+1}, \mathbf{a}_{t+1}) | (\mathbf{s}_t, \mathbf{a}_t)) = p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t) \pi_{\theta}(\mathbf{a}_{t+1} | \mathbf{s}_{t+1})$$



# Finite horizon case: state-action marginal

$$\theta^* = \arg \max_{\theta} E_{\tau \sim p_{\theta}(\tau)} \left[ \sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right]$$

$$= \arg \max_{\theta} \sum_{t=1}^T E_{(\mathbf{s}_t, \mathbf{a}_t) \sim p_{\theta}(\mathbf{s}_t, \mathbf{a}_t)} [r(\mathbf{s}_t, \mathbf{a}_t)] \quad p_{\theta}(\mathbf{s}_t, \mathbf{a}_t) \quad \text{state-action marginal}$$



# Infinite horizon case: stationary distribution

$$\theta^* = \arg \max_{\theta} \sum_{t=1}^T E_{(\mathbf{s}_t, \mathbf{a}_t) \sim p_{\theta}(\mathbf{s}_t, \mathbf{a}_t)} [r(\mathbf{s}_t, \mathbf{a}_t)]$$

what if  $T = \infty$ ?

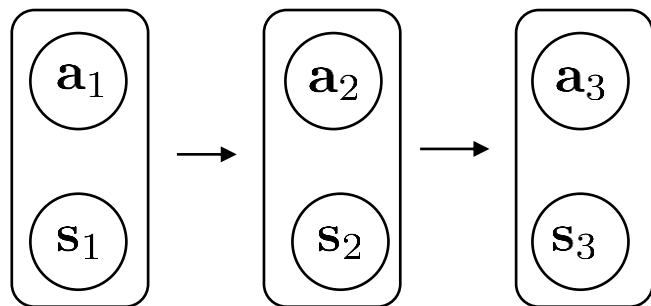
does  $p(\mathbf{s}_t, \mathbf{a}_t)$  converge to a *stationary* distribution?

$$\mu = \mathcal{T} \mu \quad (\mathcal{T} - \mathbf{I}) \mu = 0 \quad \mu = p_{\theta}(\mathbf{s}, \mathbf{a}) \quad \text{stationary distribution}$$

stationary = the same before and after transition

$\mu$  is eigenvector of  $\mathcal{T}$  with eigenvalue 1!

(always exists under some regularity conditions)



*state-action* transition operator

$$\begin{pmatrix} \mathbf{s}_{t+1} \\ \mathbf{a}_{t+1} \end{pmatrix} = \mathcal{T} \begin{pmatrix} \mathbf{s}_t \\ \mathbf{a}_t \end{pmatrix} \quad \begin{pmatrix} \mathbf{s}_{t+k} \\ \mathbf{a}_{t+k} \end{pmatrix} = \mathcal{T}^k \begin{pmatrix} \mathbf{s}_t \\ \mathbf{a}_t \end{pmatrix}$$

# Infinite horizon case: stationary distribution

$$\theta^* = \arg \max_{\theta} \frac{1}{T} \sum_{t=1}^T E_{(\mathbf{s}_t, \mathbf{a}_t) \sim p_{\theta}(\mathbf{s}_t, \mathbf{a}_t)} [r(\mathbf{s}_t, \mathbf{a}_t)] \rightarrow E_{(\mathbf{s}, \mathbf{a}) \sim p_{\theta}(\mathbf{s}, \mathbf{a})} [r(\mathbf{s}, \mathbf{a})]$$

(in the limit as  $T \rightarrow \infty$ )

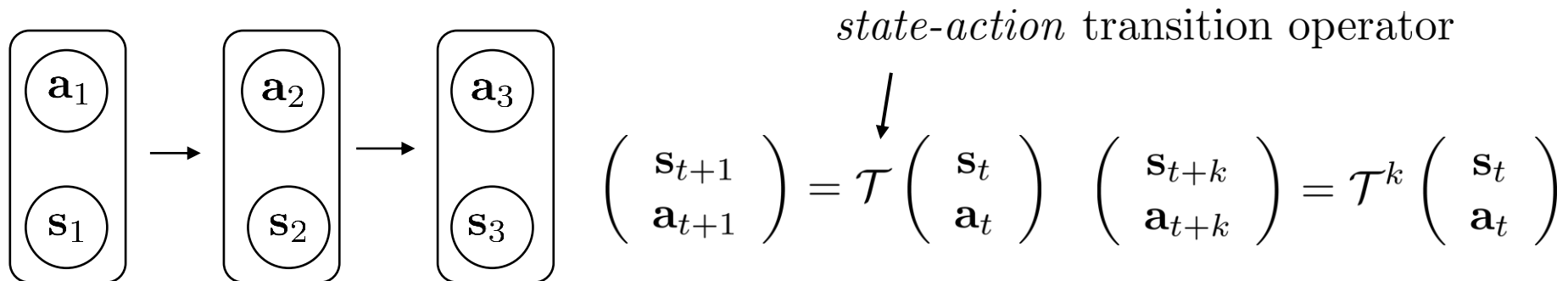
what if  $T = \infty$ ?

does  $p(\mathbf{s}_t, \mathbf{a}_t)$  converge to a *stationary* distribution?

$$\mu = \mathcal{T} \mu \quad (\mathcal{T} - \mathbf{I})\mu = 0 \quad \mu = p_{\theta}(\mathbf{s}, \mathbf{a}) \quad \text{stationary distribution}$$

stationary = the same before and after transition

$\mu$  is eigenvector of  $\mathcal{T}$  with eigenvalue 1!  
(always exists under some regularity conditions)



# Expectations and stochastic systems

$$\theta^* = \arg \max_{\theta} E_{(\mathbf{s}, \mathbf{a}) \sim p_{\theta}(\mathbf{s}, \mathbf{a})} [r(\mathbf{s}, \mathbf{a})]$$

infinite horizon case

$$\theta^* = \arg \max_{\theta} \sum_{t=1}^T E_{(\mathbf{s}_t, \mathbf{a}_t) \sim p_{\theta}(\mathbf{s}_t, \mathbf{a}_t)} [r(\mathbf{s}_t, \mathbf{a}_t)]$$

finite horizon case

In RL, we almost always care about *expectations*



$r(\mathbf{x})$  – *not* smooth

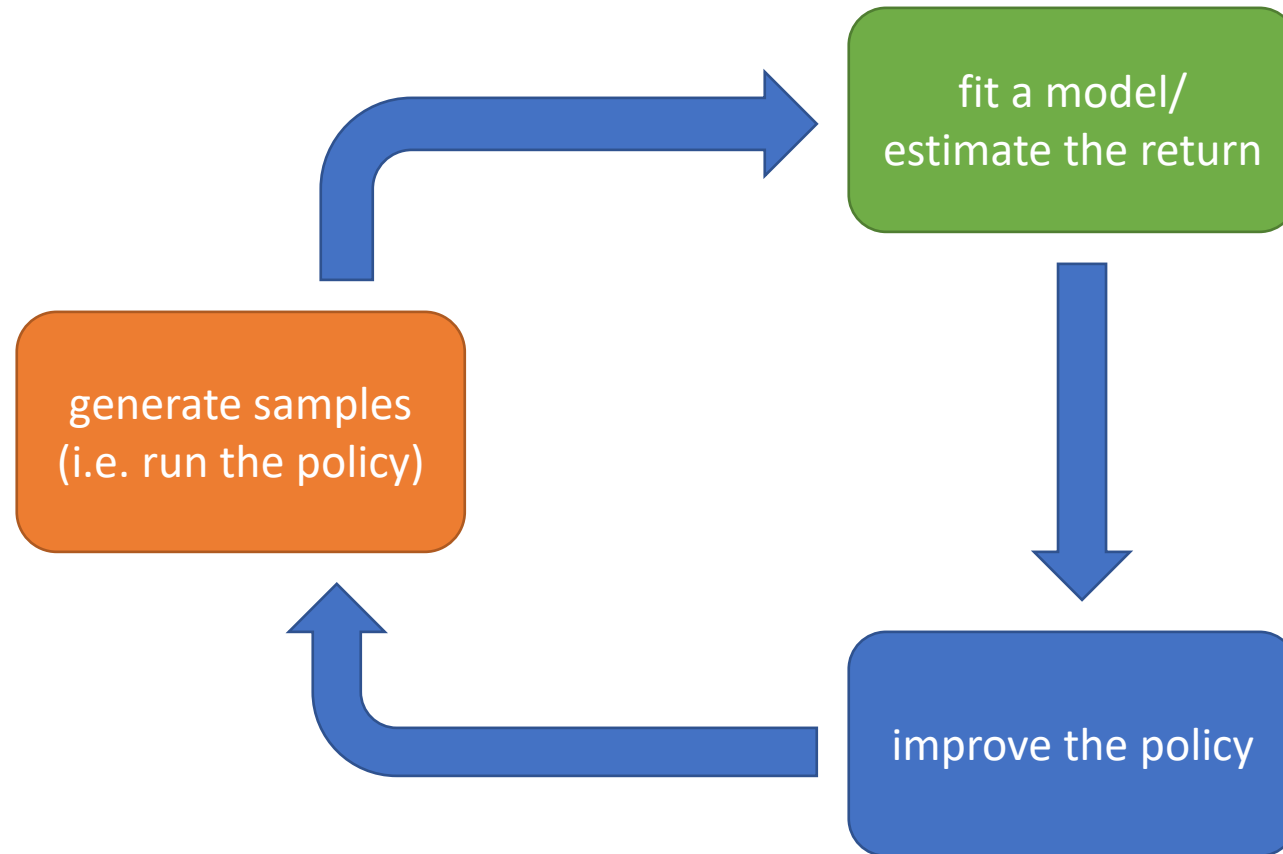
$\pi_{\theta}(\mathbf{a} = \text{fall}) = \theta$

$E_{\pi_{\theta}}[r(\mathbf{x})]$  – *smooth* in  $\theta$ !

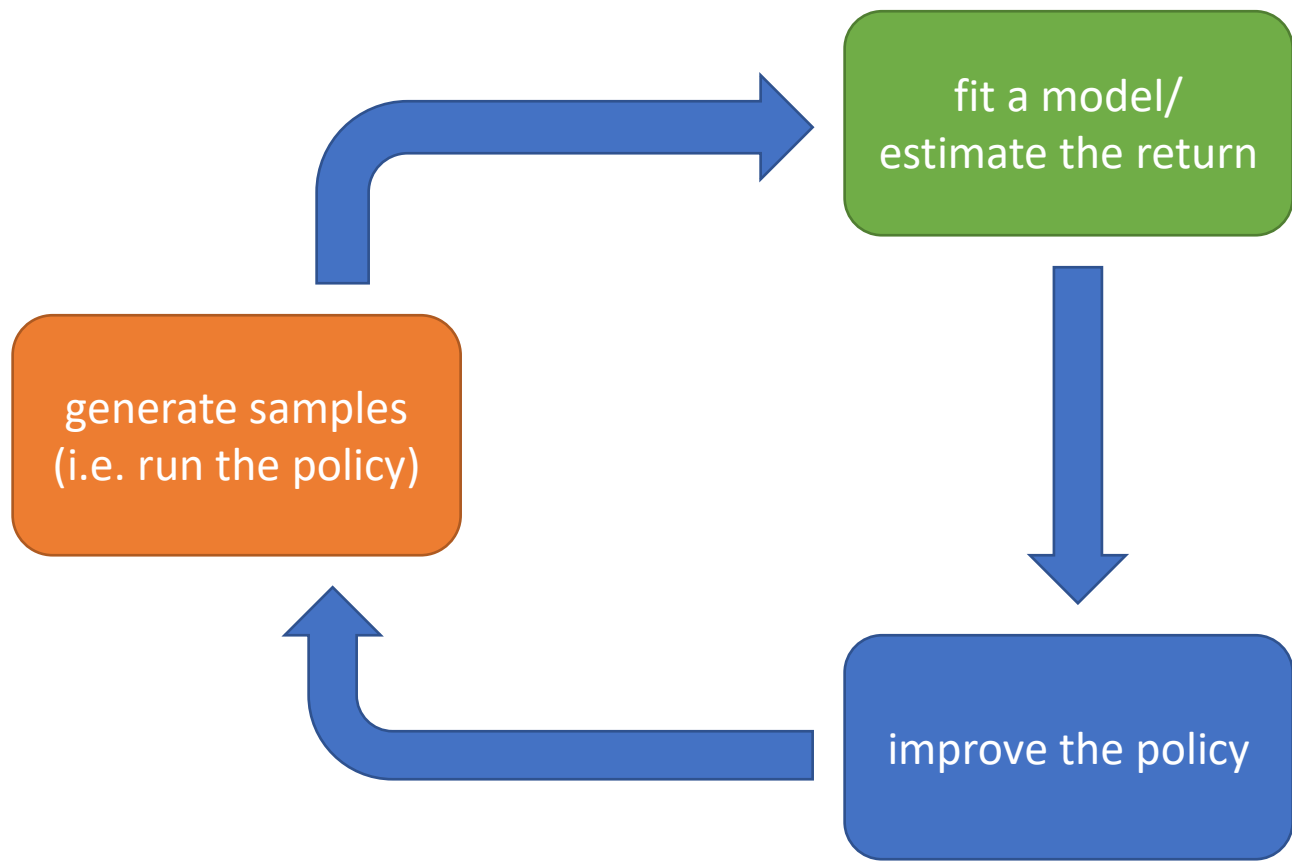
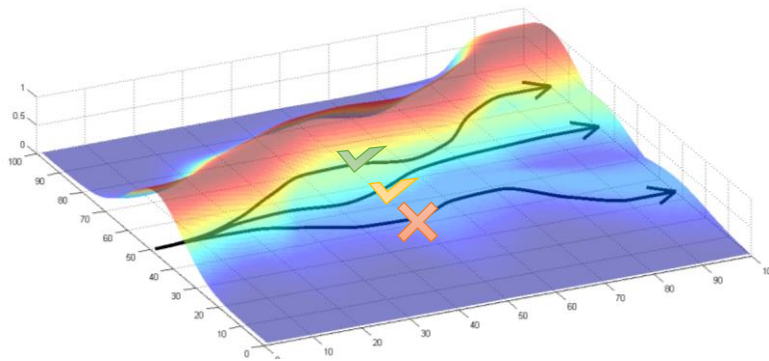


# Algorithms

# The anatomy of a reinforcement learning algorithm



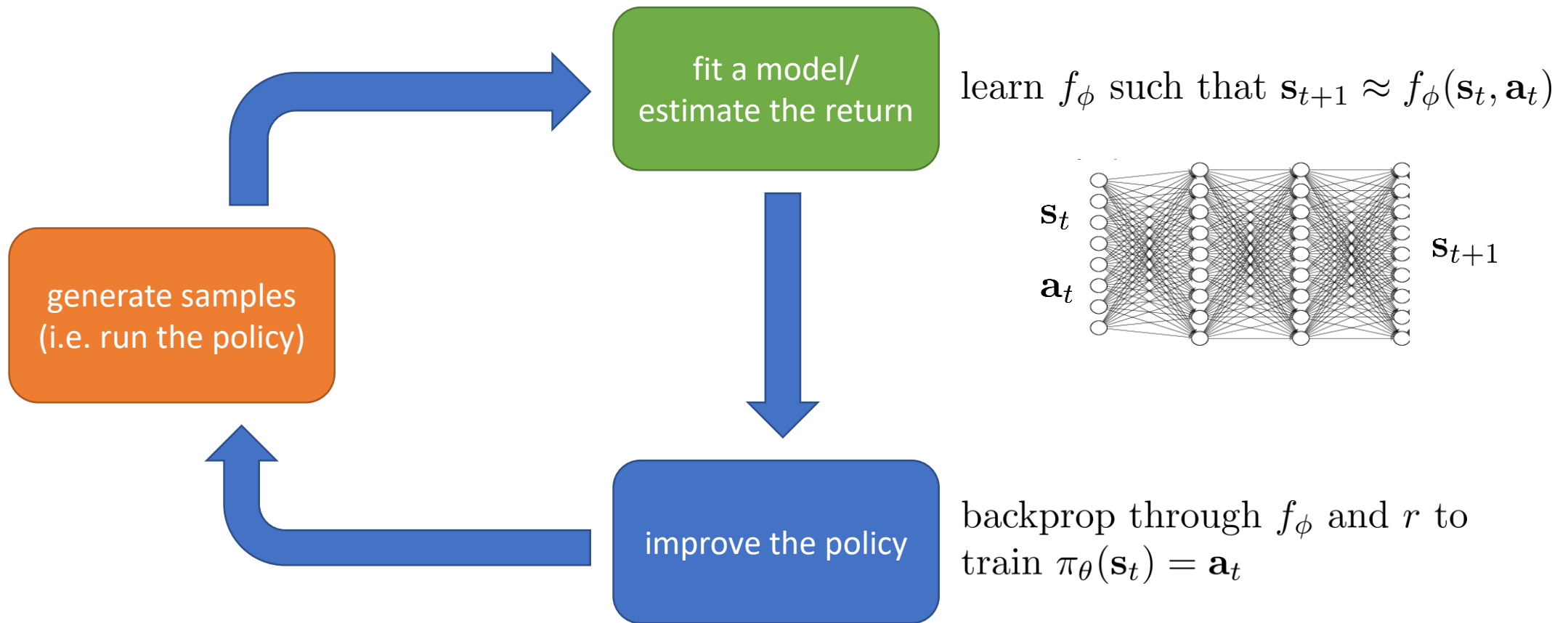
# A simple example



$$J(\theta) = E_{\pi} \left[ \sum_t r_t \right] \approx \frac{1}{N} \sum_{i=1}^N \sum_t r_t^i$$

$$\theta \leftarrow \theta + \alpha \nabla_{\theta} J(\theta)$$

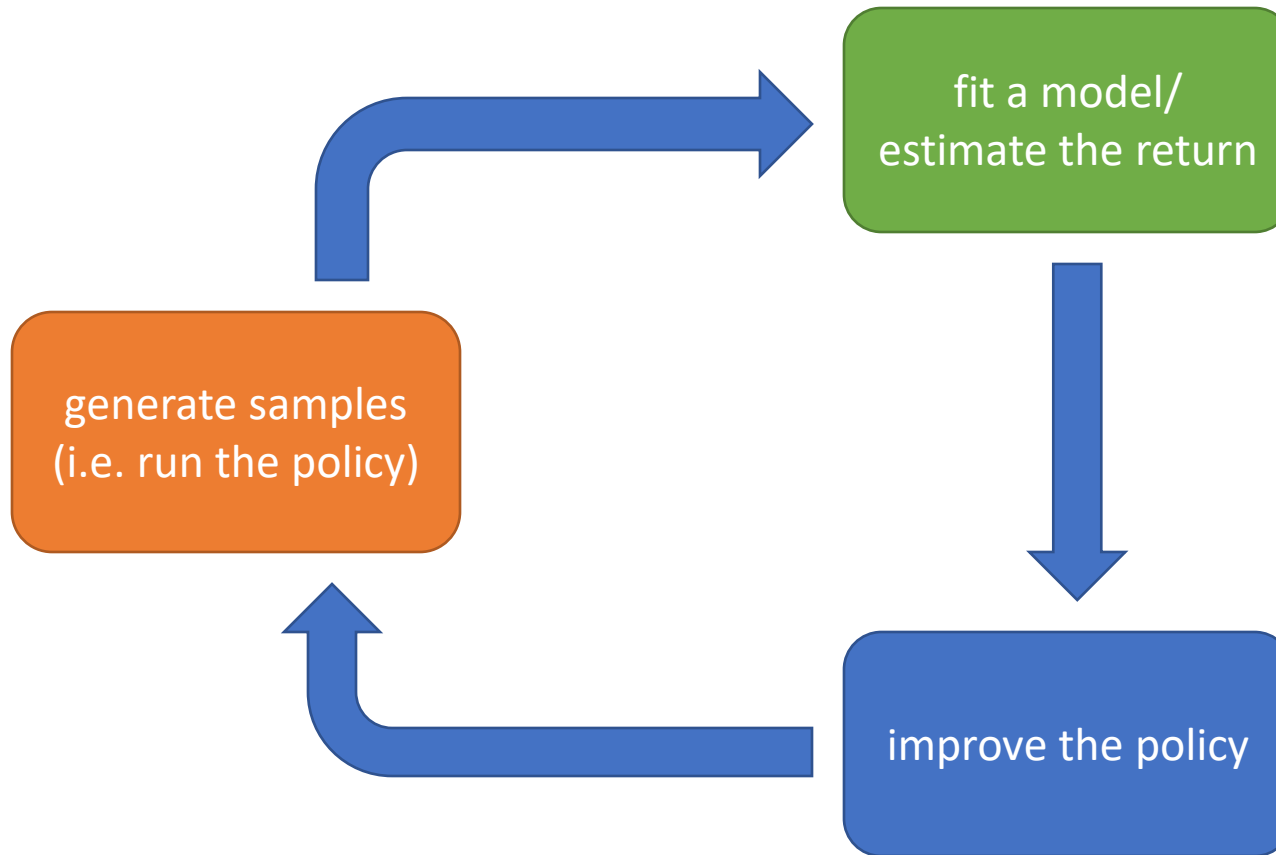
# Another example: RL by backprop



# Which parts are expensive?

real robot/car/power grid/whatever:  
1x real time, until we invent time travel

MuJoCo simulator:  
up to 10000x real time



$$J(\theta) = E_{\pi} \left[ \sum_t r_t \right] \approx \frac{1}{N} \sum_{i=1}^N \sum_t r_t^i$$

trivial, fast

learn  $\mathbf{s}_{t+1} \approx f_{\phi}(\mathbf{s}_t, \mathbf{a}_t)$   
expensive

$$\theta \leftarrow \theta + \alpha \nabla_{\theta} J(\theta)$$

backprop through  $f_{\phi}$  and  $r$  to train  $\pi_{\theta}(\mathbf{s}_t) = \mathbf{a}_t$

# Value Functions

# How do we deal with all these expectations?

$$E_{\tau \sim p_{\theta}(\tau)} \left[ \sum_{t=1}^T r(\mathbf{s}_t, \mathbf{a}_t) \right]$$

$$E_{\mathbf{s}_1 \sim p(\mathbf{s}_1)}$$

what if we knew this part?

$$Q(\mathbf{s}_1, \mathbf{a}_1) = r(\mathbf{s}_1, \mathbf{a}_1) + E_{\mathbf{s}_2 \sim p(\mathbf{s}_2 | \mathbf{s}_1, \mathbf{a}_1)} [E_{\mathbf{a}_2 \sim \pi(\mathbf{a}_2 | \mathbf{s}_2)} [r(\mathbf{s}_2, \mathbf{a}_2) + \dots | \mathbf{s}_2] | \mathbf{s}_1, \mathbf{a}_1]$$

$$E_{\tau \sim p_{\theta}(\tau)} \left[ \sum_{t=1}^T r(\mathbf{s}_t, \mathbf{a}_t) \right] = E_{\mathbf{s}_1 \sim p(\mathbf{s}_1)} [E_{\mathbf{a}_1 \sim \pi(\mathbf{a}_1 | \mathbf{s}_1)} [Q(\mathbf{s}_1, \mathbf{a}_1) | \mathbf{s}_1]]$$

easy to modify  $\pi_{\theta}(\mathbf{a}_1 | \mathbf{s}_1)$  if  $Q(\mathbf{s}_1, \mathbf{a}_1)$  is known!

example:  $\pi(\mathbf{a}_1 | \mathbf{s}_1) = 1$  if  $\mathbf{a}_1 = \arg \max_{\mathbf{a}_1} Q(\mathbf{s}_1, \mathbf{a}_1)$

# Definition: Q-function

$Q^\pi(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^T E_{\pi_\theta} [r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) | \mathbf{s}_t, \mathbf{a}_t]$ : total reward from taking  $\mathbf{a}_t$  in  $\mathbf{s}_t$

# Definition: value function

$V^\pi(\mathbf{s}_t) = \sum_{t'=t}^T E_{\pi_\theta} [r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) | \mathbf{s}_t]$ : total reward from  $\mathbf{s}_t$

$V^\pi(\mathbf{s}_t) = E_{\mathbf{a}_t \sim \pi(\mathbf{a}_t | \mathbf{s}_t)} [Q^\pi(\mathbf{s}_t, \mathbf{a}_t)]$

$E_{\mathbf{s}_1 \sim p(\mathbf{s}_1)} [V^\pi(\mathbf{s}_1)]$  is the RL objective!



# Using Q-functions and value functions

Idea 1: if we have policy  $\pi$ , and we know  $Q^\pi(\mathbf{s}, \mathbf{a})$ , then we can *improve*  $\pi$ :

set  $\pi'(\mathbf{a}|\mathbf{s}) = 1$  if  $\mathbf{a} = \arg \max_{\mathbf{a}} Q^\pi(\mathbf{s}, \mathbf{a})$

this policy is at least as good as  $\pi$  (and probably better)!

and it doesn't matter what  $\pi$  is

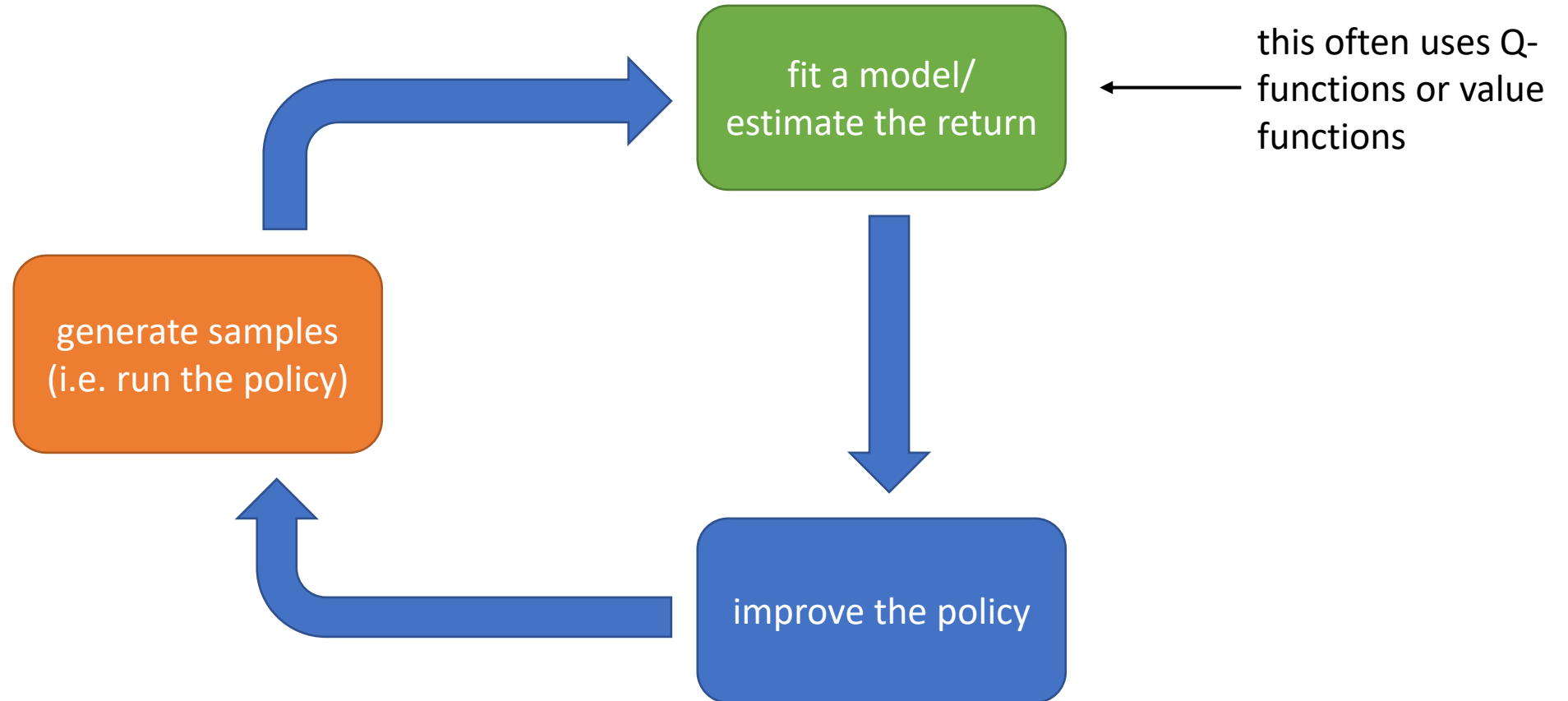
Idea 2: compute gradient to increase probability of good actions  $\mathbf{a}$ :

if  $Q^\pi(\mathbf{s}, \mathbf{a}) > V^\pi(\mathbf{s})$ , then  $\mathbf{a}$  is *better than average* (recall that  $V^\pi(\mathbf{s}) = E[Q^\pi(\mathbf{s}, \mathbf{a})]$  under  $\pi(\mathbf{a}|\mathbf{s})$ )

modify  $\pi(\mathbf{a}|\mathbf{s})$  to increase probability of  $\mathbf{a}$  if  $Q^\pi(\mathbf{s}, \mathbf{a}) > V^\pi(\mathbf{s})$

These ideas are *very* important in RL; we'll revisit them again and again!

# The anatomy of a reinforcement learning algorithm



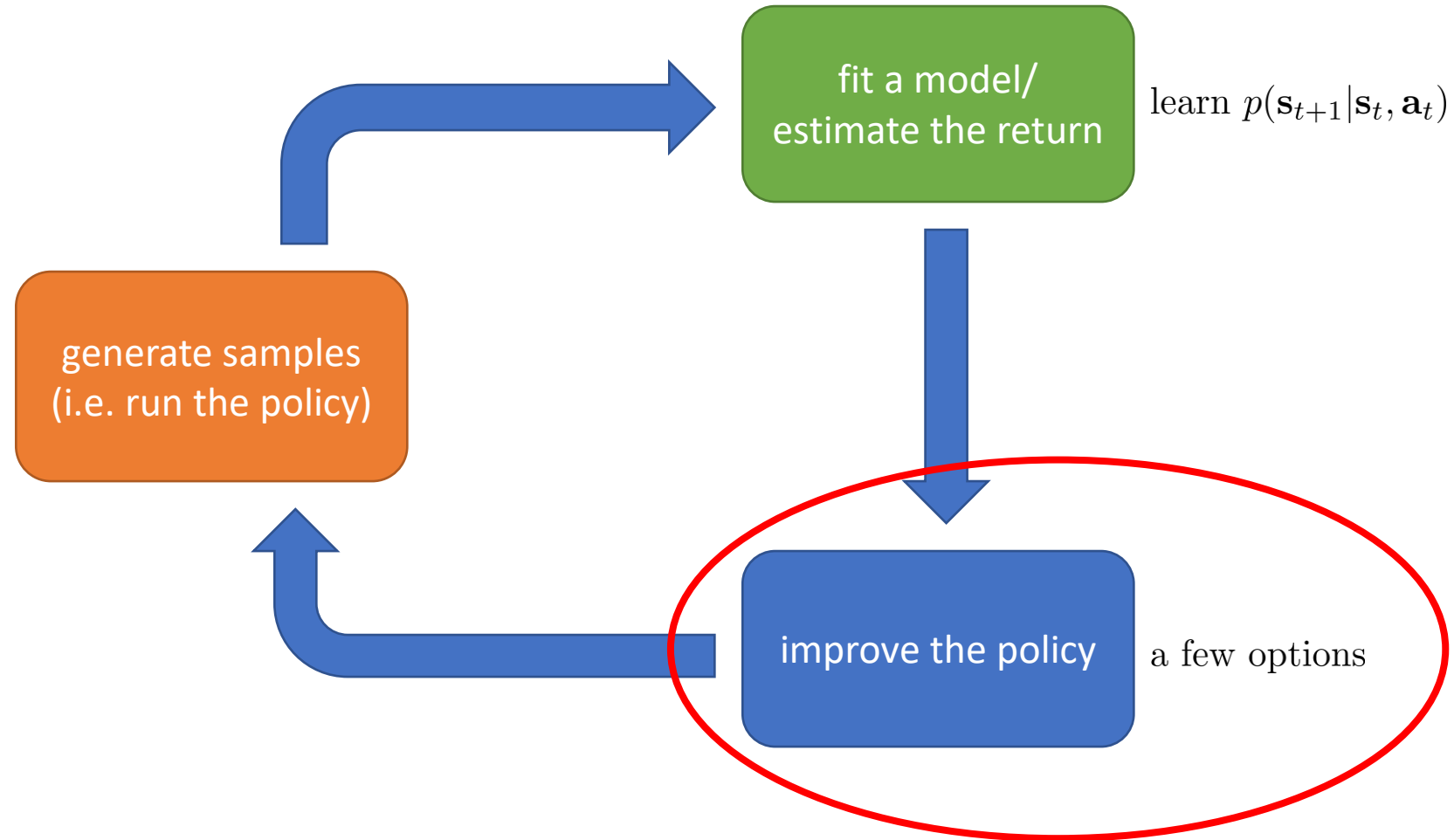
# Types of Algorithms

# Types of RL algorithms

$$\theta^* = \arg \max_{\theta} E_{\tau \sim p_{\theta}(\tau)} \left[ \sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right]$$

- Policy gradients: directly differentiate the above objective
- Value-based: estimate value function or Q-function of the optimal policy (no explicit policy)
- Actor-critic: estimate value function or Q-function of the current policy, use it to improve policy
- Model-based RL: estimate the transition model, and then...
  - Use it for planning (no explicit policy)
  - Use it to improve a policy
  - Something else

# Model-based RL algorithms



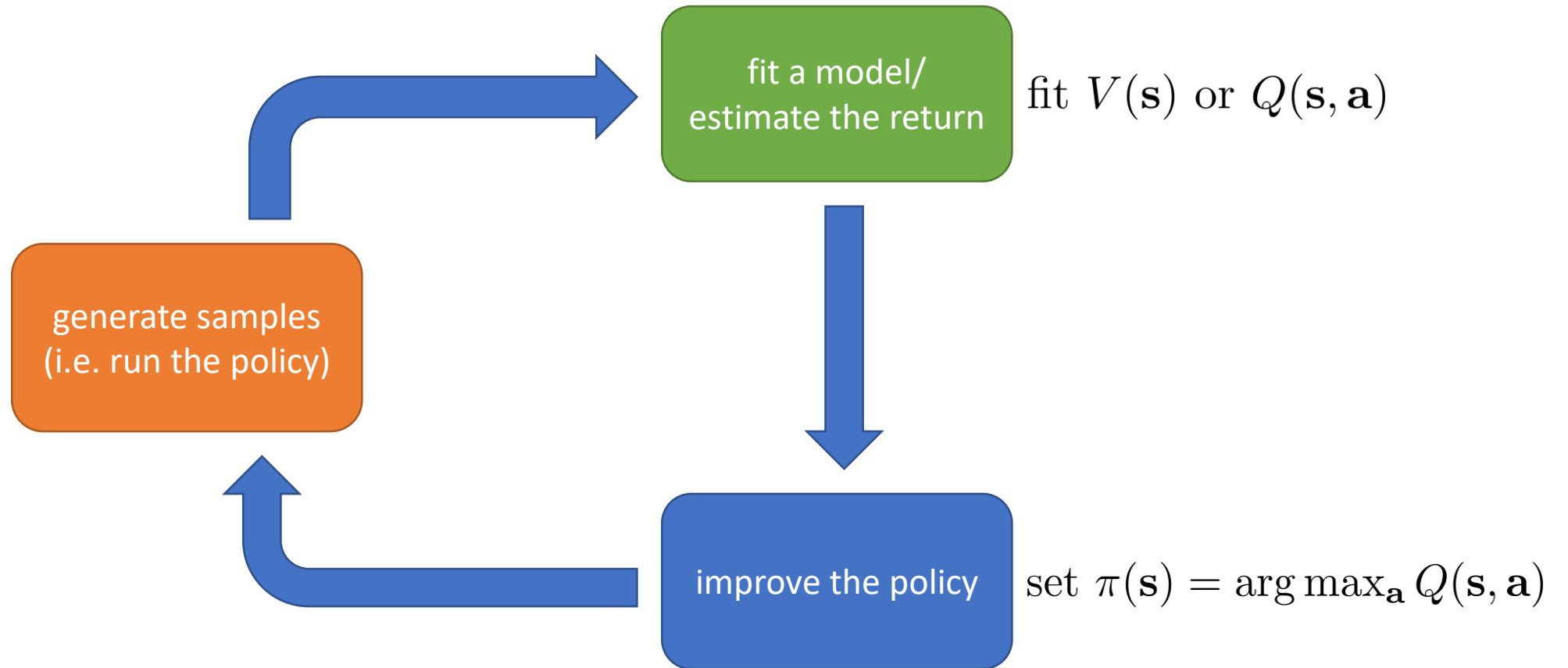
# Model-based RL algorithms

improve the policy

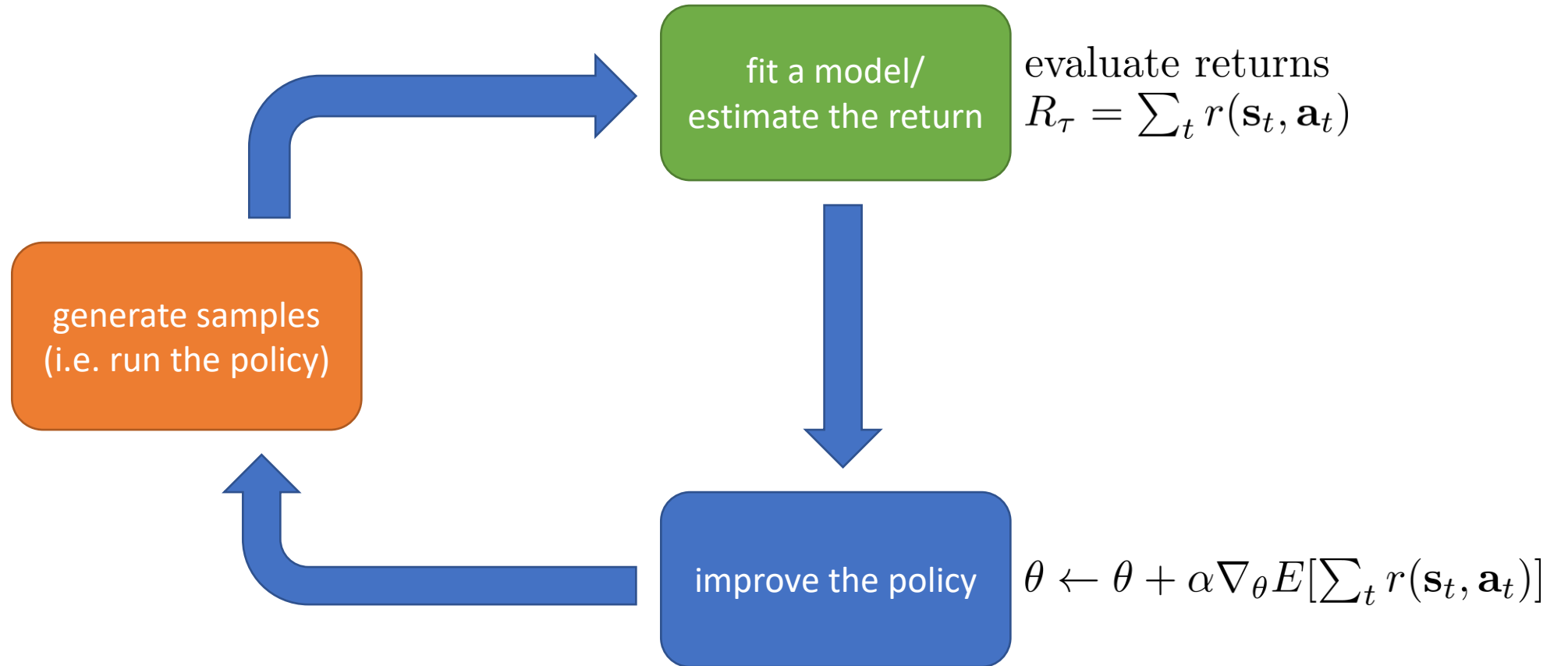
a few options

1. Just use the model to plan (no policy)
  - Trajectory optimization/optimal control (primarily in continuous spaces) – essentially backpropagation to optimize over actions
  - Discrete planning in discrete action spaces – e.g., Monte Carlo tree search
2. Backpropagate gradients into the policy
  - Requires some tricks to make it work
3. Use the model to learn a value function
  - Dynamic programming
  - Generate simulated experience for model-free learner

# Value function based algorithms

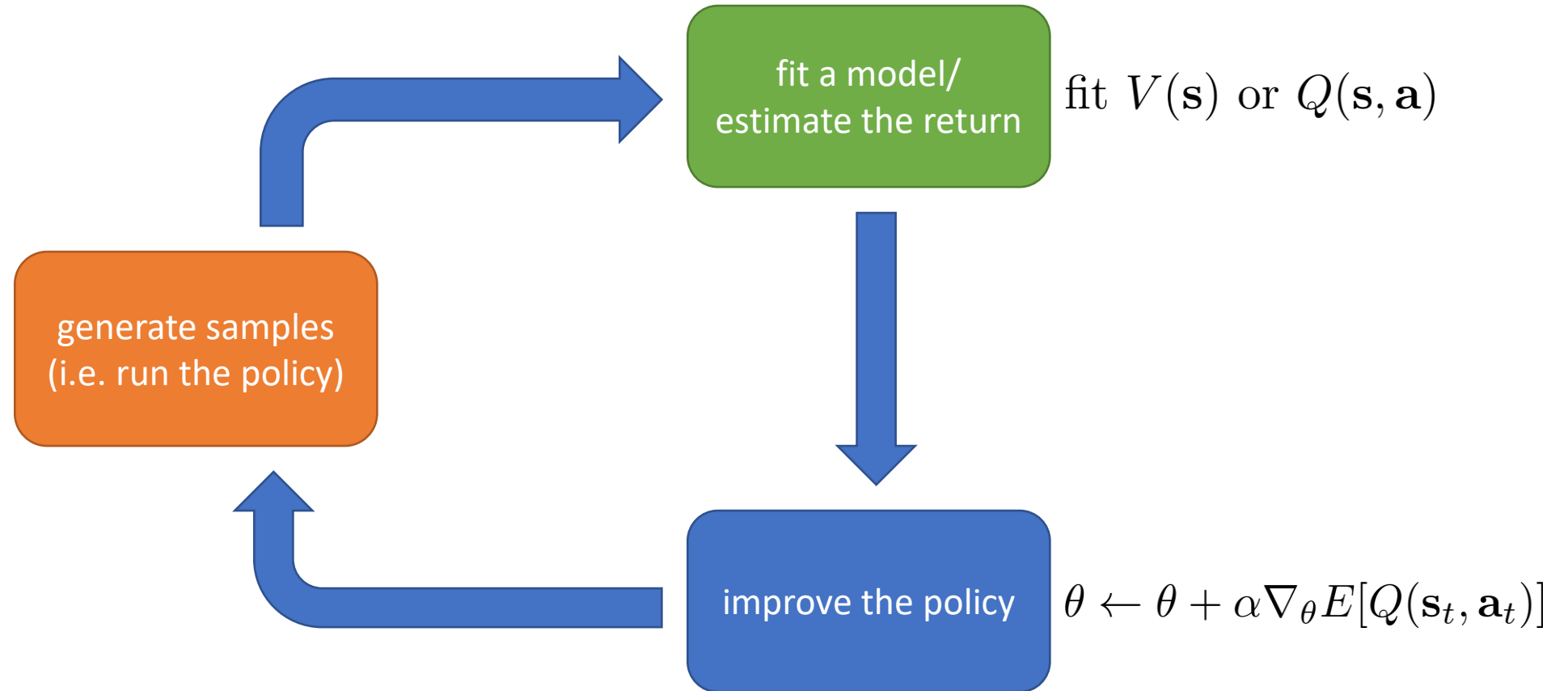


# Direct policy gradients





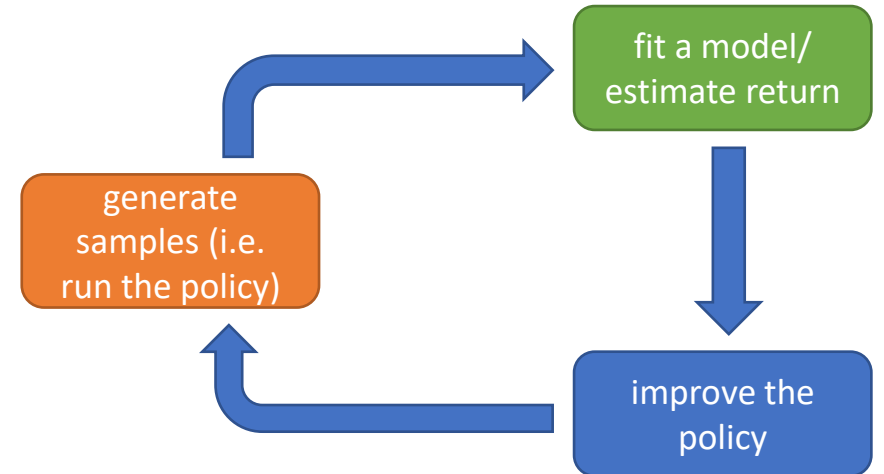
# Actor-critic: value functions + policy gradients



# Tradeoffs Between Algorithms

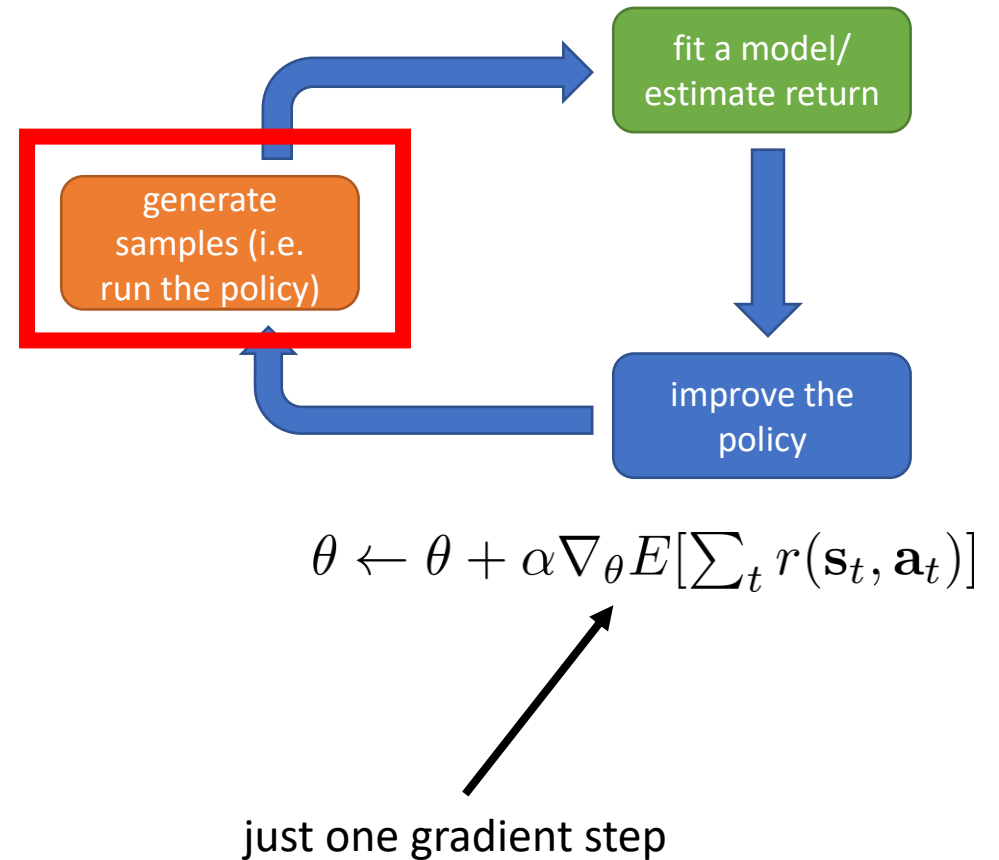
# Why so many RL algorithms?

- Different tradeoffs
  - Sample efficiency
  - Stability & ease of use
- Different assumptions
  - Stochastic or deterministic?
  - Continuous or discrete?
  - Episodic or infinite horizon?
- Different things are easy or hard in different settings
  - Easier to represent the policy?
  - Easier to represent the model?

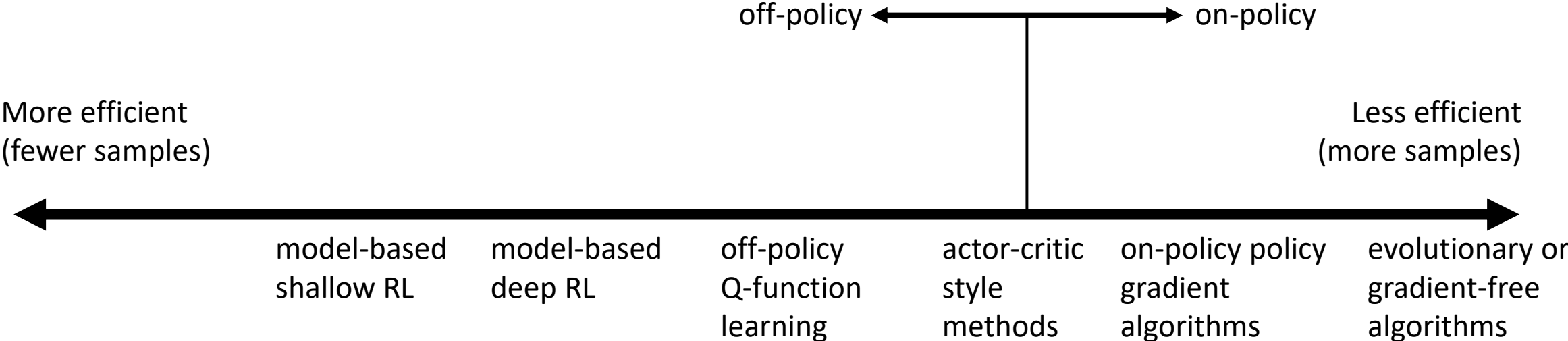


# Comparison: sample efficiency

- Sample efficiency = how many samples do we need to get a good policy?
- Most important question: is the algorithm *off policy*?
  - Off policy: able to improve the policy without generating new samples from that policy
  - On policy: each time the policy is changed, even a little bit, we need to generate new samples



# Comparison: sample efficiency



Why would we use a *less* efficient algorithm?

Wall clock time is not the same as efficiency!

# Comparison: stability and ease of use

- Does it converge?
- And if it converges, to what?
- And does it converge every time?

Why is any of this even a question???

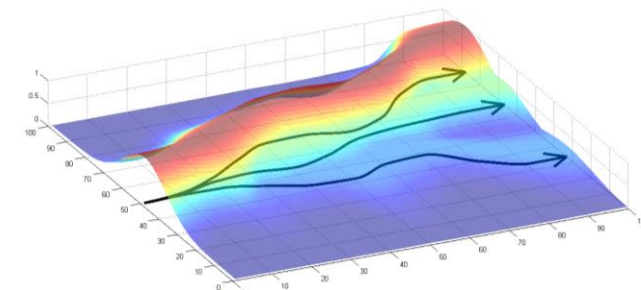
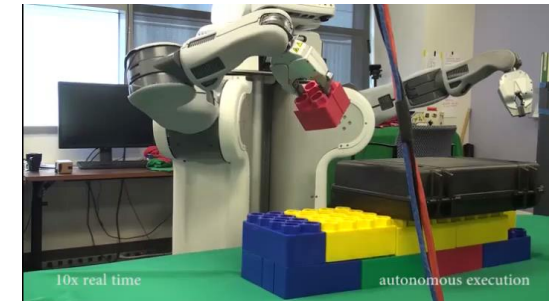
- Supervised learning: almost *always* gradient descent
- Reinforcement learning: often *not* gradient descent
  - Q-learning: fixed point iteration
  - Model-based RL: model is not optimized for expected reward
  - Policy gradient: *is* gradient descent, but also often the least efficient!

# Comparison: stability and ease of use

- Value function fitting
  - At best, minimizes error of fit (“Bellman error”)
    - Not the same as expected reward
  - At worst, doesn’t optimize anything
    - Many popular deep RL value fitting algorithms are not guaranteed to converge to *anything* in the nonlinear case
- Model-based RL
  - Model minimizes error of fit
    - This will converge
  - No guarantee that better model = better policy
- Policy gradient
  - The only one that actually performs gradient descent (ascent) on the true objective

# Comparison: assumptions

- Common assumption #1: full observability
  - Generally assumed by value function fitting methods
  - Can be mitigated by adding recurrence
- Common assumption #2: episodic learning
  - Often assumed by pure policy gradient methods
  - Assumed by some model-based RL methods
- Common assumption #3: continuity or smoothness
  - Assumed by some continuous value function learning methods
  - Often assumed by some model-based RL methods





# Examples of Algorithms

# Examples of specific algorithms

- Value function fitting methods
  - Q-learning, DQN
  - Temporal difference learning
  - Fitted value iteration
- Policy gradient methods
  - REINFORCE
  - Natural policy gradient
  - Trust region policy optimization
- Actor-critic algorithms
  - Asynchronous advantage actor-critic (A3C)
  - Soft actor-critic (SAC)
- Model-based RL algorithms
  - Dyna
  - Guided policy search

We'll learn about most of these in the next few weeks!

# Example 1: Atari games with Q-functions

- Playing Atari with deep reinforcement learning, Mnih et al. '13
- Q-learning with convolutional neural networks



# Example 2: robots and model-based RL

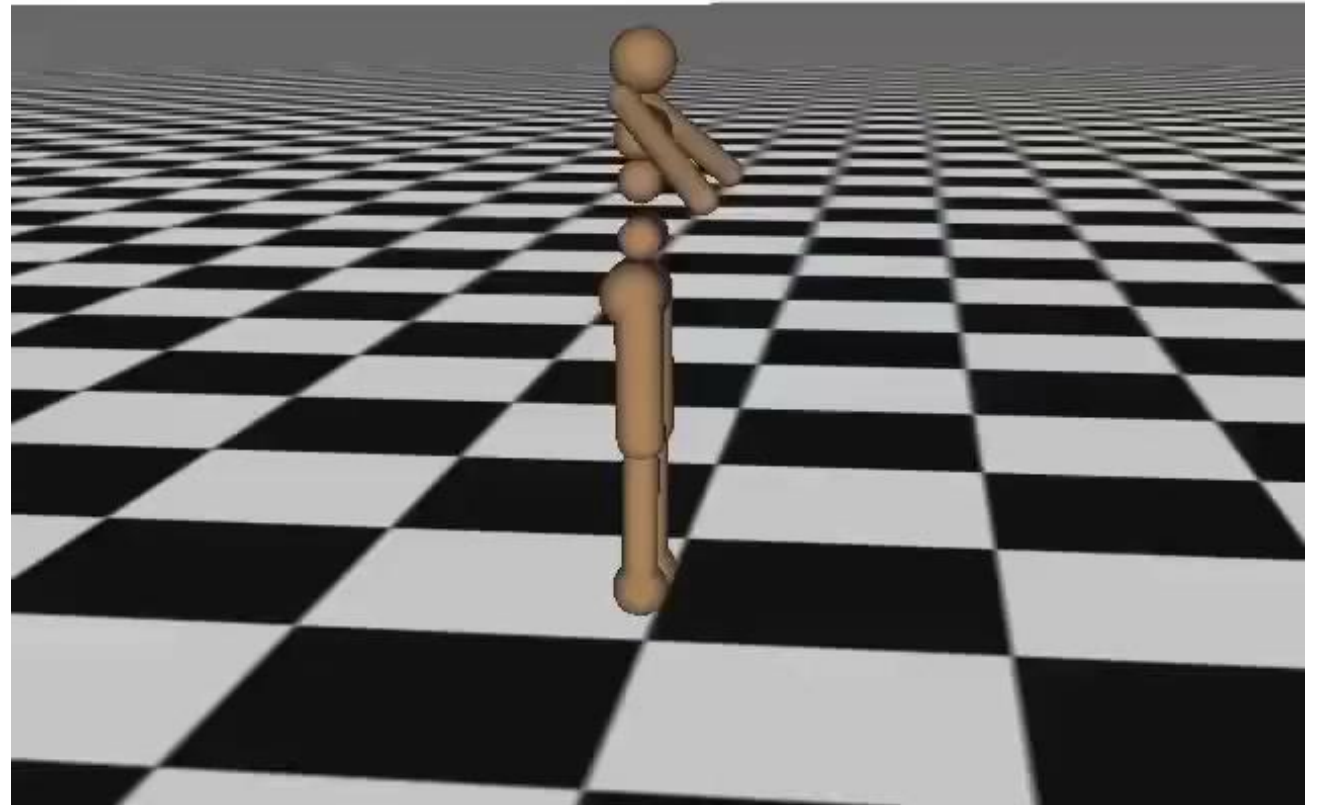
- End-to-end training of deep visuomotor policies, L.\* , Finn\* '16
- Guided policy search (model-based RL) for image-based robotic manipulation

**Various Experiments**  
Including the policy input

# Example 3: walking with policy gradients

- High-dimensional continuous control with generalized advantage estimation, Schulman et al. '16
- Trust region policy optimization with value function approximation

Iteration 0



# Example 4: robotic grasping with Q-functions

- QT-Opt, Kalashnikov et al. '18
- Q-learning from images for real-world robotic grasping

