Generative World Models v/s Sim2Real

Sanjiban Choudhury

Cornell Bowers CIS Computer Science

Today's class

- Practical MBRL
- Learning Models: The DREAMER algorithm
- Leveraging Physics: Sim2Real

  Case study: OpenAI Dactyl Hand

Modelling complex tasks from video input

Challenges with learning complex models

Challenge 1: Can't see state, only get high-dimensional observations

Challenge 2: Planning with complex dynamics

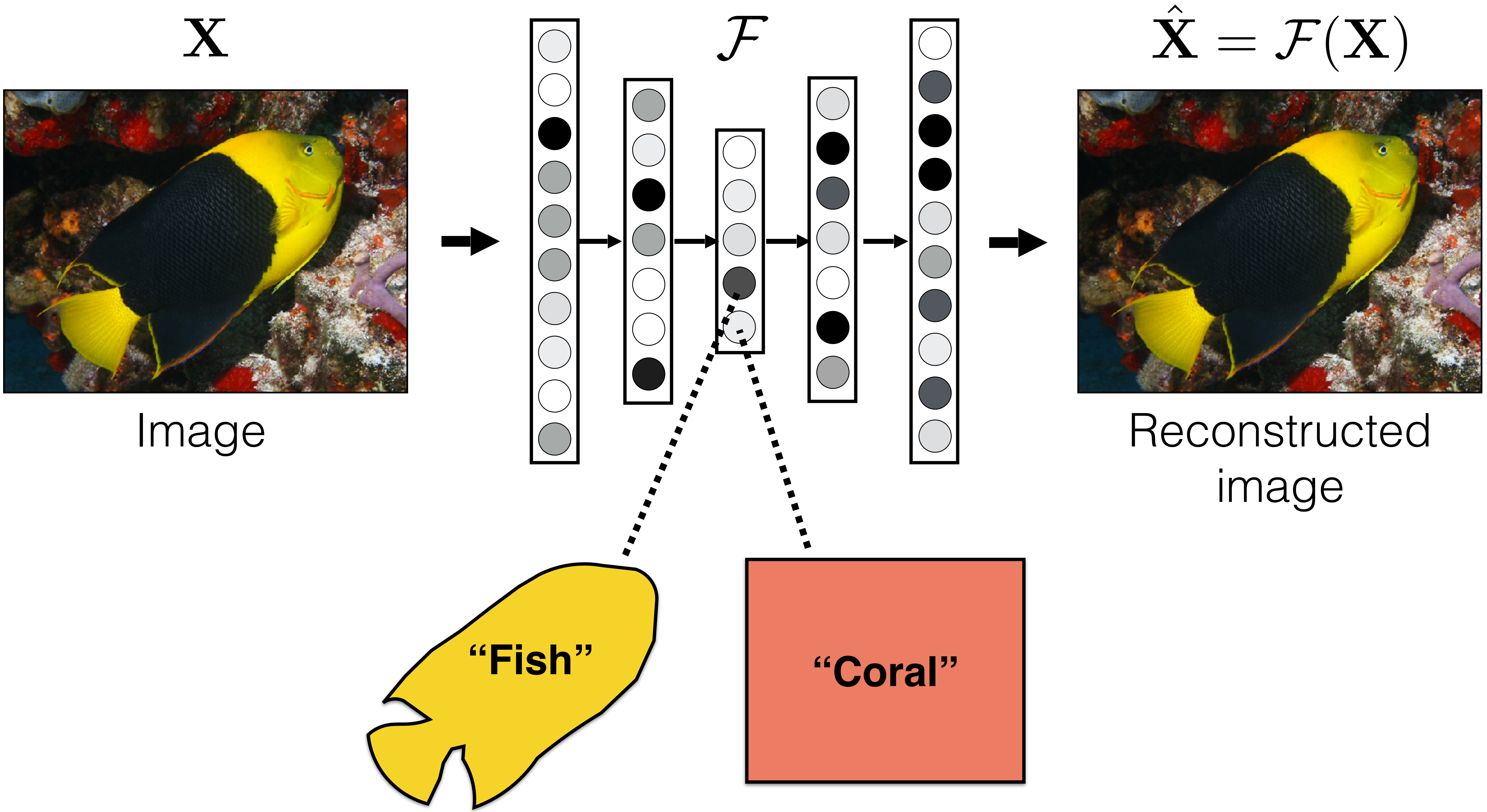
Two strategies

“Physics the sh*t out it!”

“Be lazy and use ML”

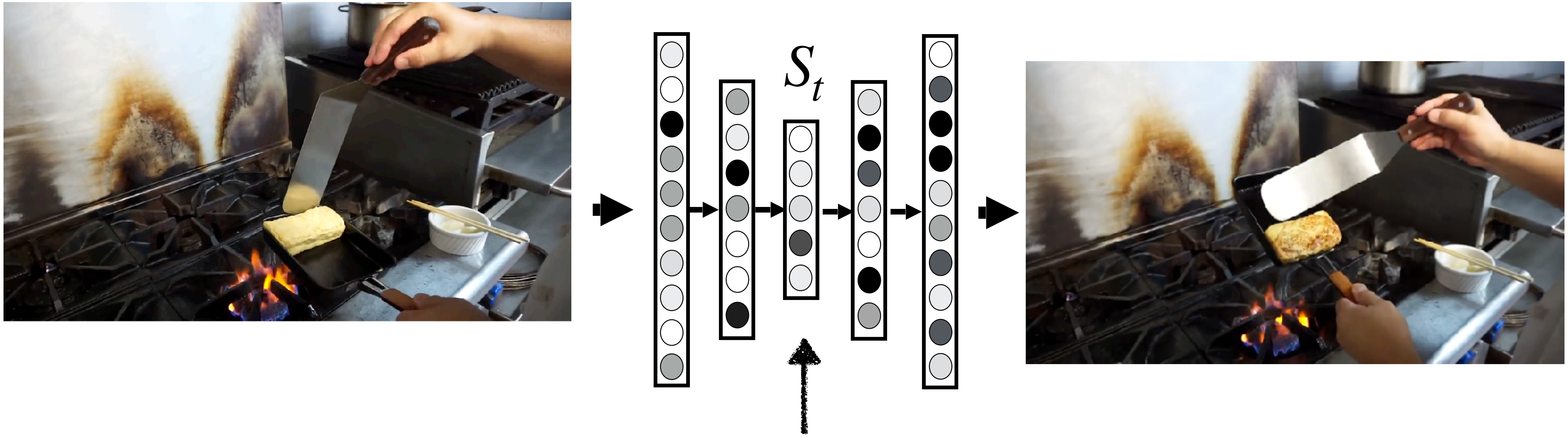
How can we learn latent low-dimensional state from high-dimensional observations?

Idea: Use "auto-encoder" trick from computer vision



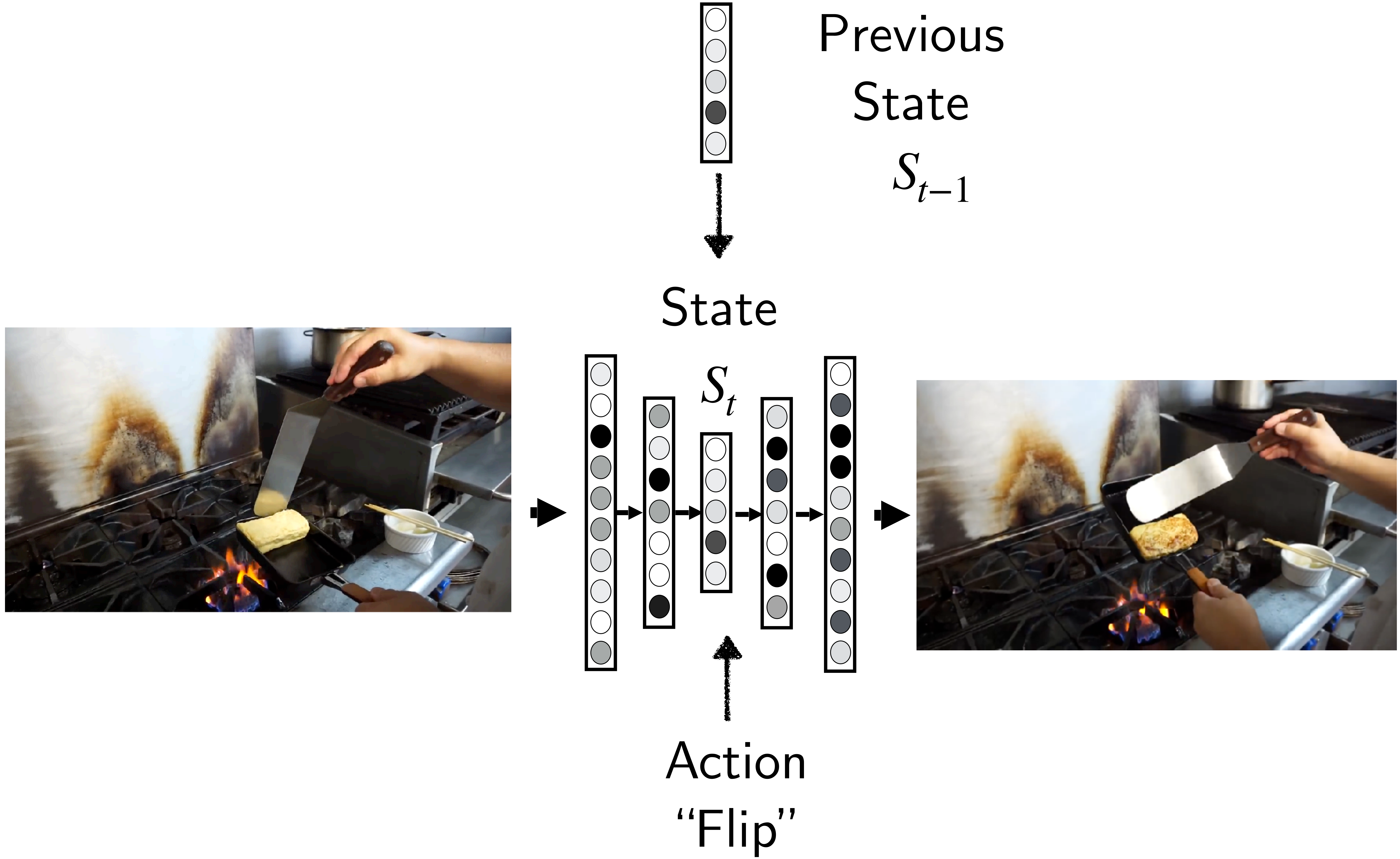
$\mathbf{X}$
$\mathcal{F}$
$\hat{\mathbf{X}} = \mathcal{F}(\mathbf{X})$
Image
Reconstructed image
"Fish"
"Coral"

State



Action

“Flip”


Previous
State
S_{t-1}
State
S_t
Action
“Flip”

Today's class

- ☑ Practical MBRL

  (Only observations, complex dynamics)

- ☐ Learning Models: The DREAMER algorithm

- ☐ Leveraging Physics: Sim2Real

  Case study: OpenAI Dactyl Hand

The DREAMER Algorithms

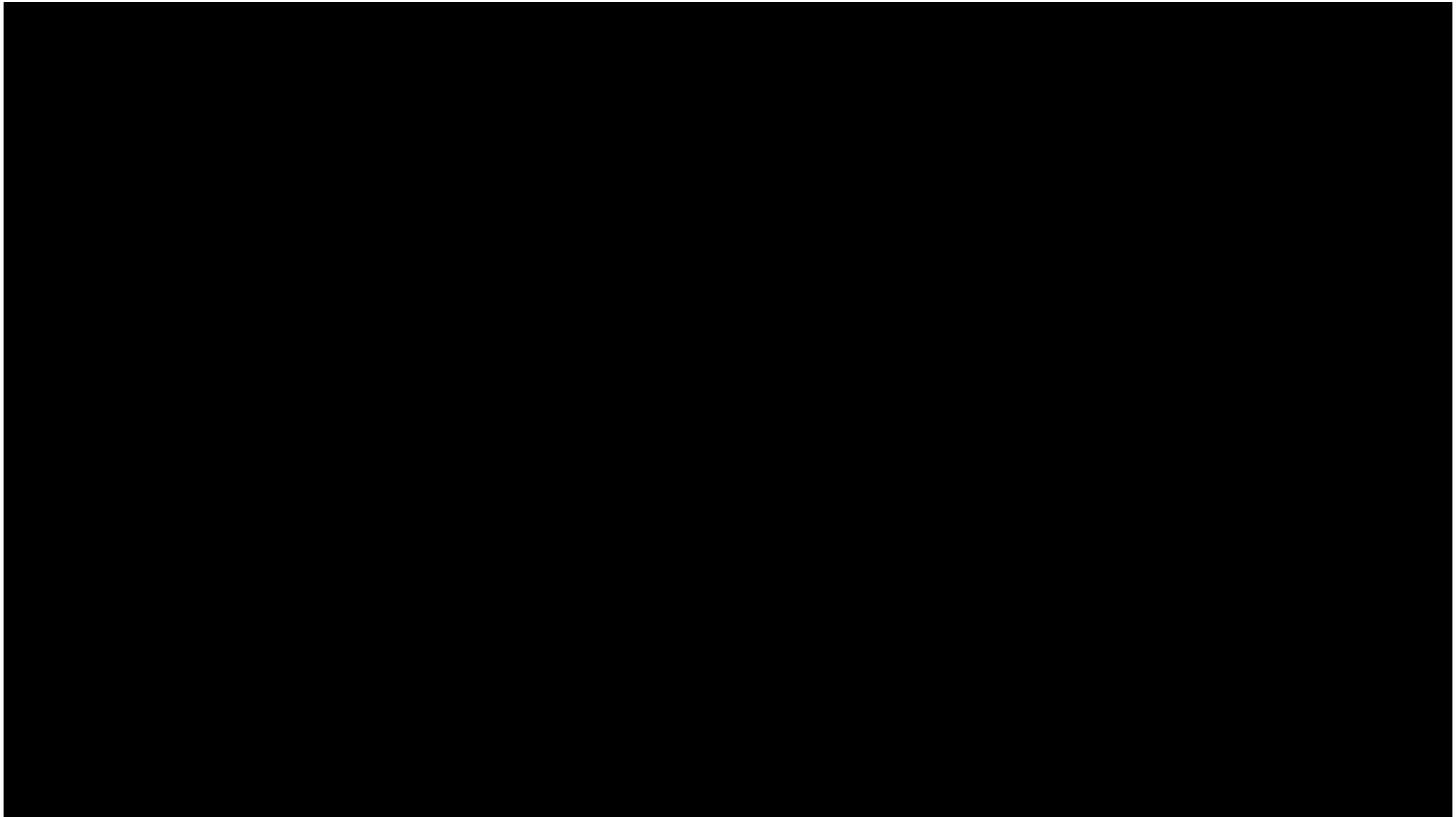
Mastering Diverse Domains through World Models

2023

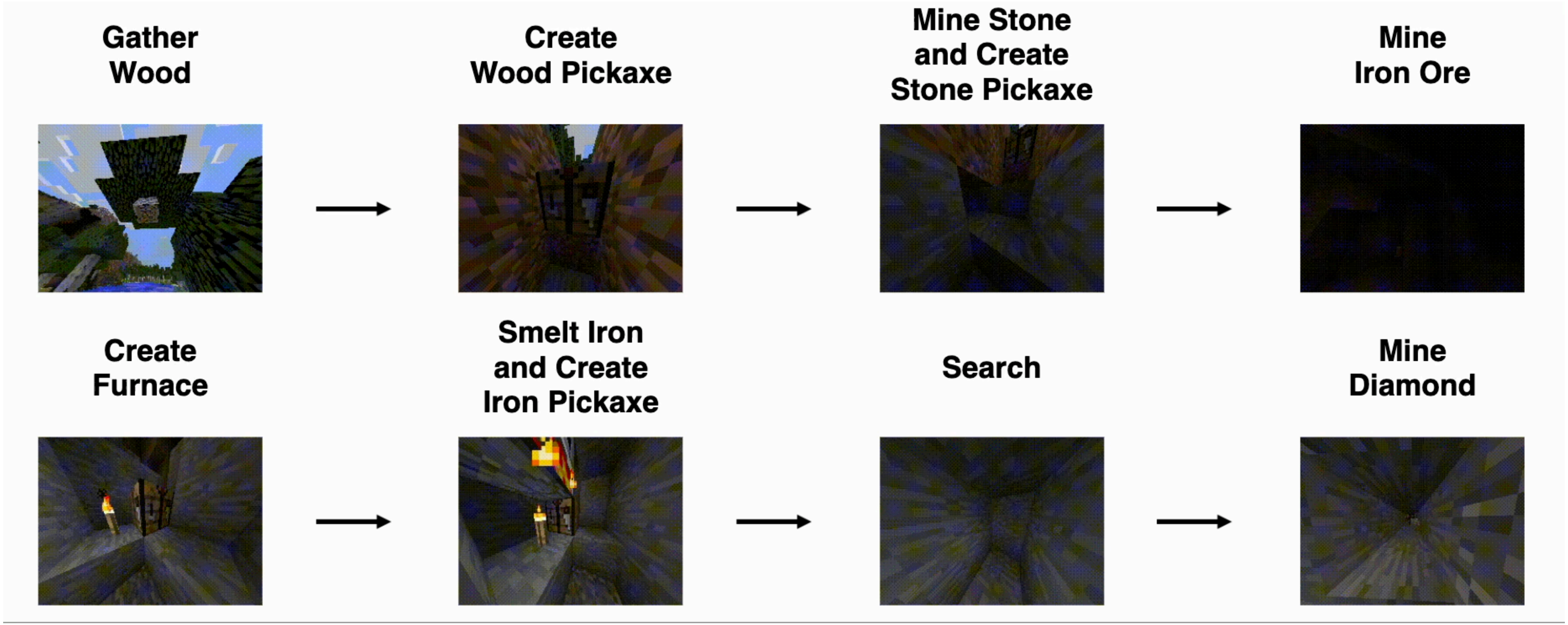
Danijar Hafner,[1,2] Jurgis Pasukonis,[1] Jimmy Ba,[2] Timothy Lillicrap[1]

[1]DeepMind [2]University of Toronto

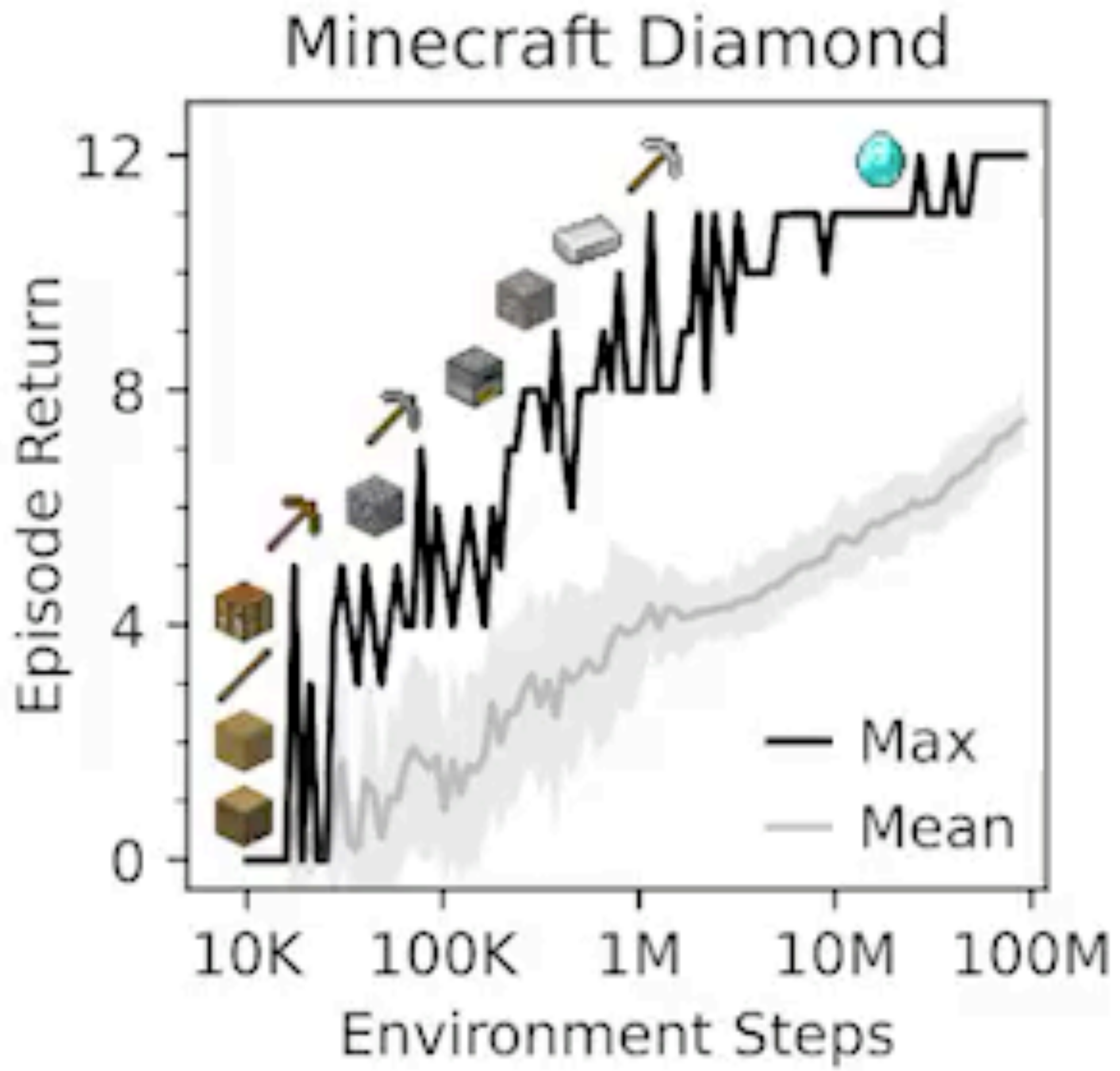
MineRL Diamond Challenge

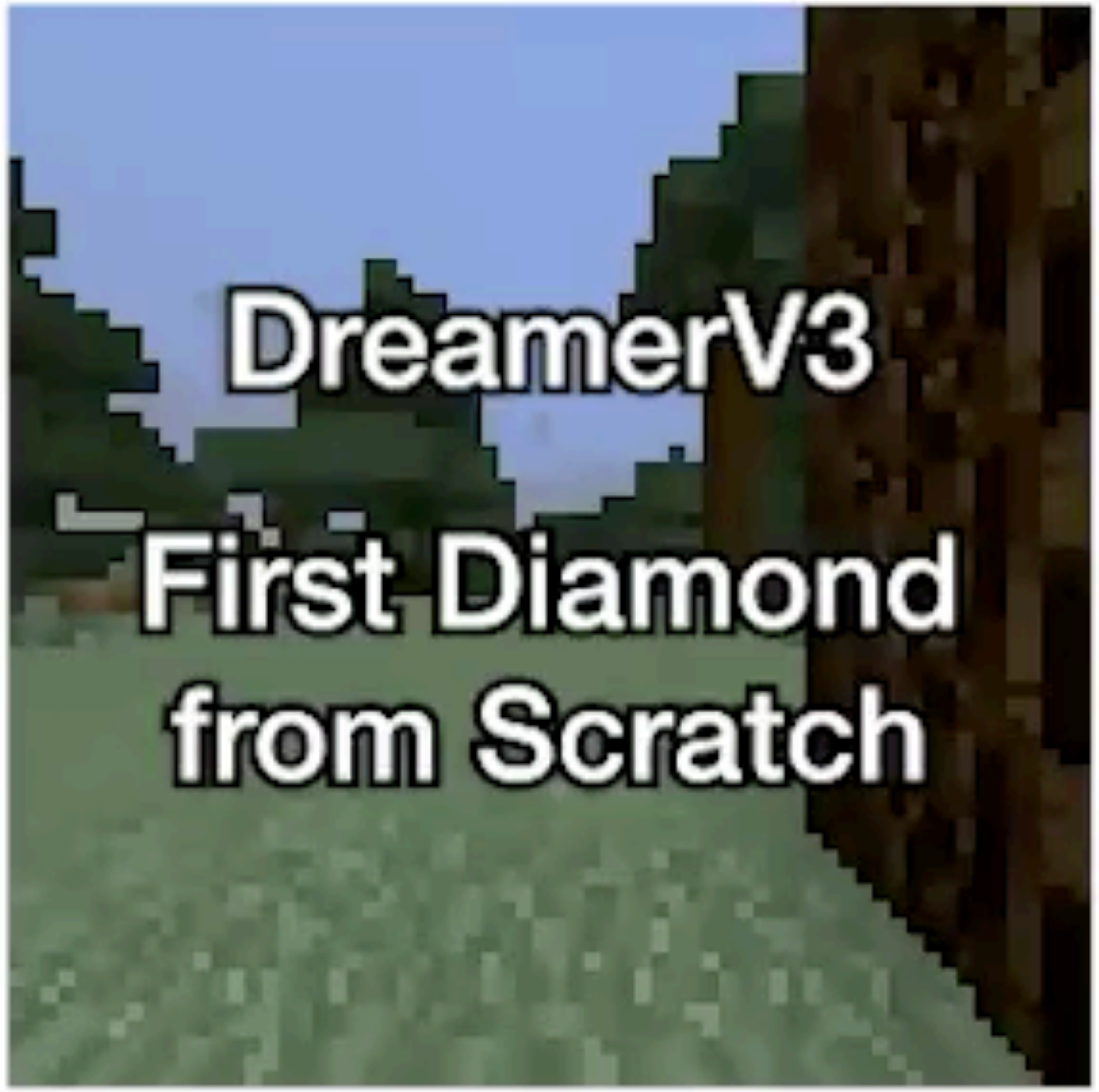


MineRL Diamond Challenge



DreamerV3 solved this task!





The DREAMER Algorithm

Dream to Control: Learning Behaviors by Latent Imagination

Danijar Hafner *
University of Toronto
Google Brain

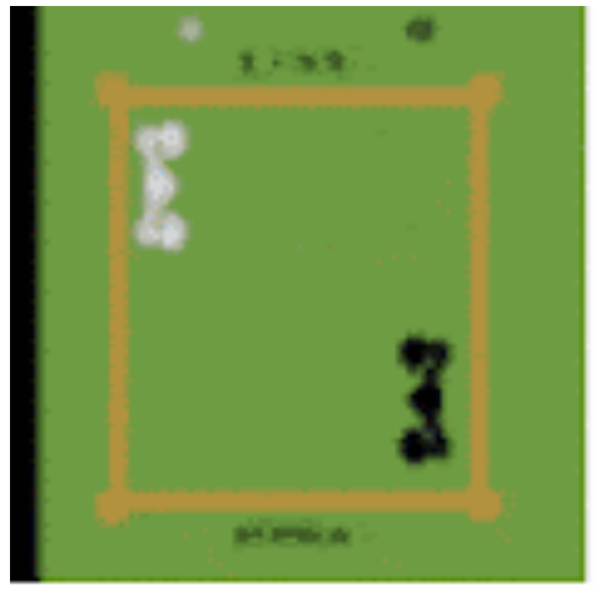
Timothy Lillicrap
DeepMind

Jimmy Ba
University of Toronto

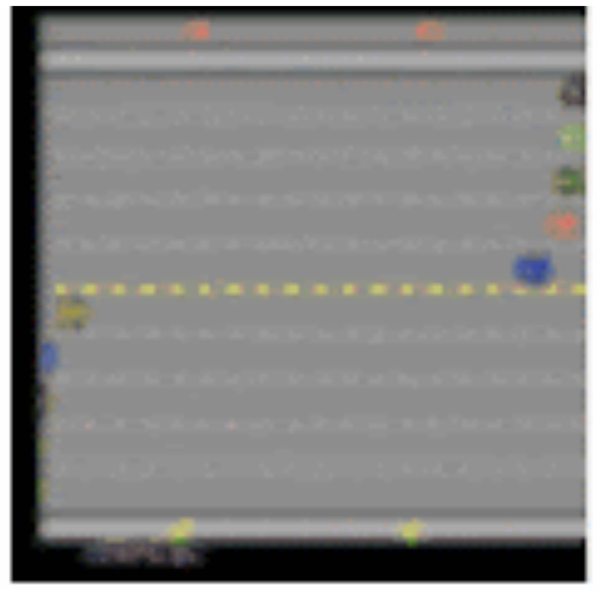
Mohammad Norouzi
Google Brain

2020

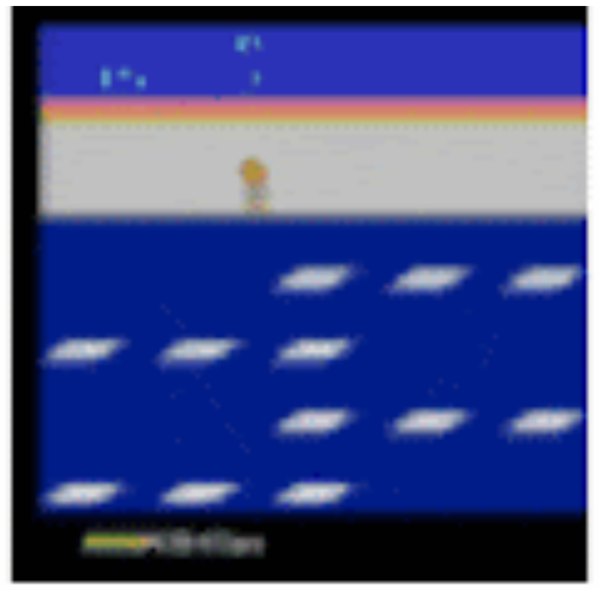
Look at the videos below



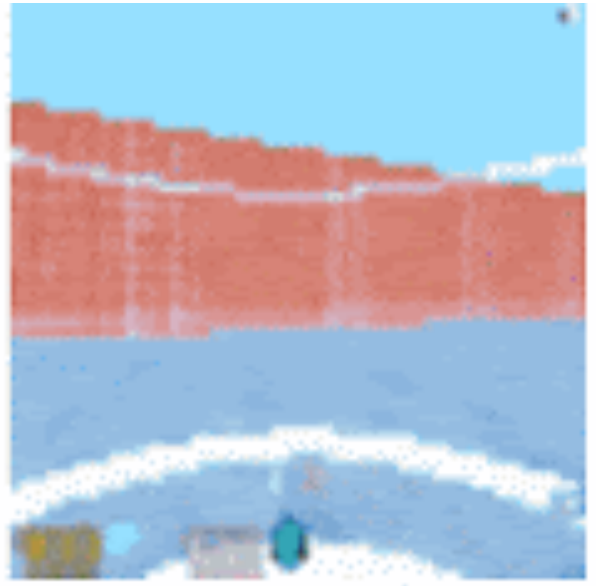
Boxing



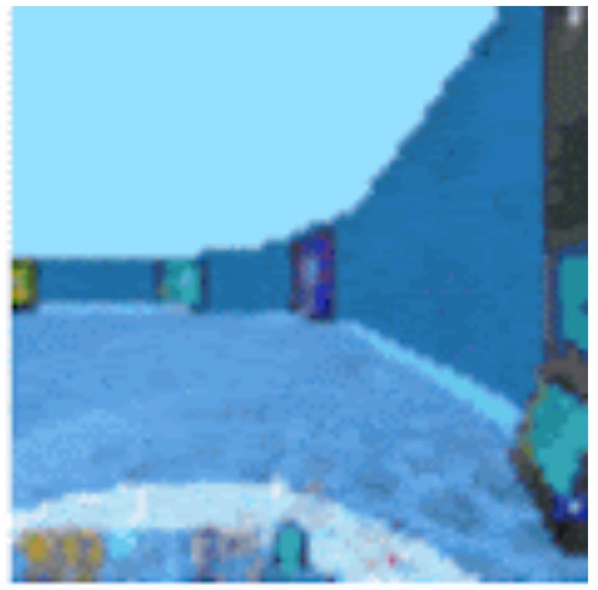
Freeway



Frostbite



Collect Objects



Watermaze

Sparse Cartpole

Acrobot Swingup

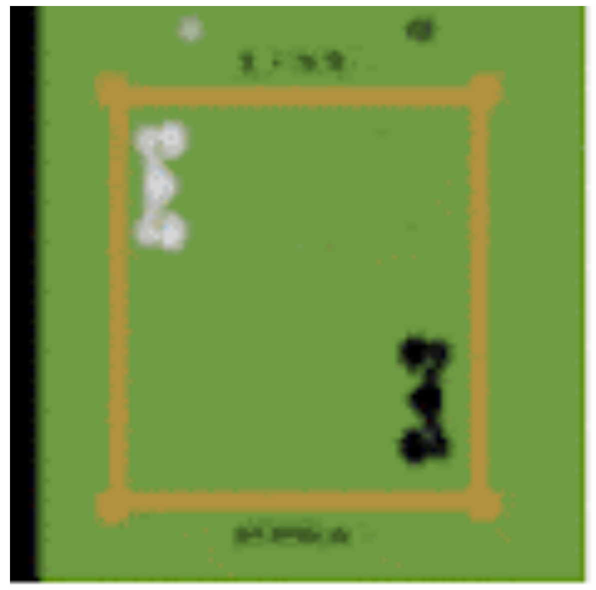
Hopper Hop

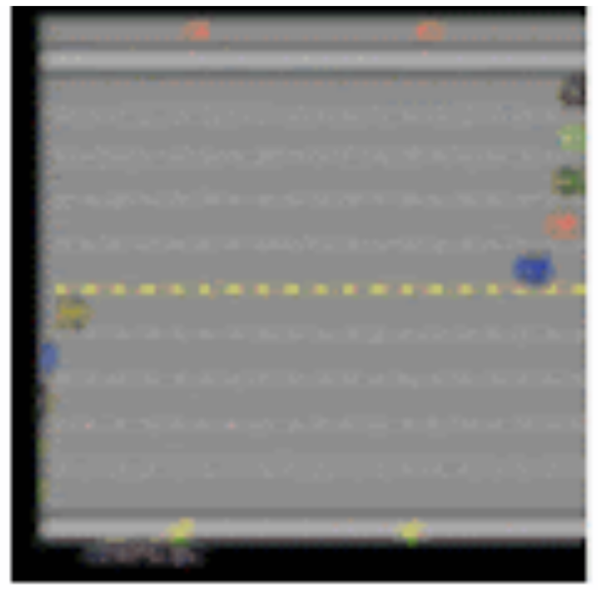
Walker Run

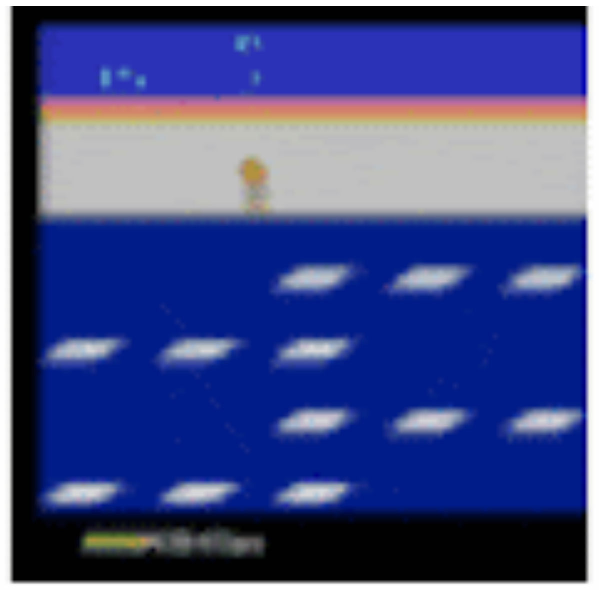
Quadruped Run

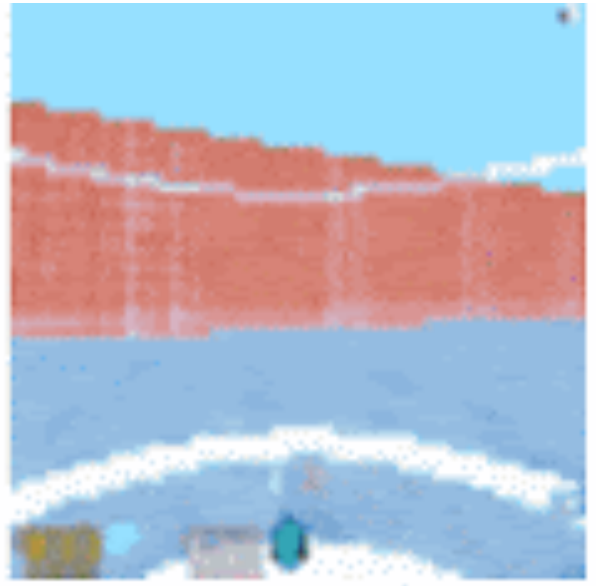
Which of these are real vs model?

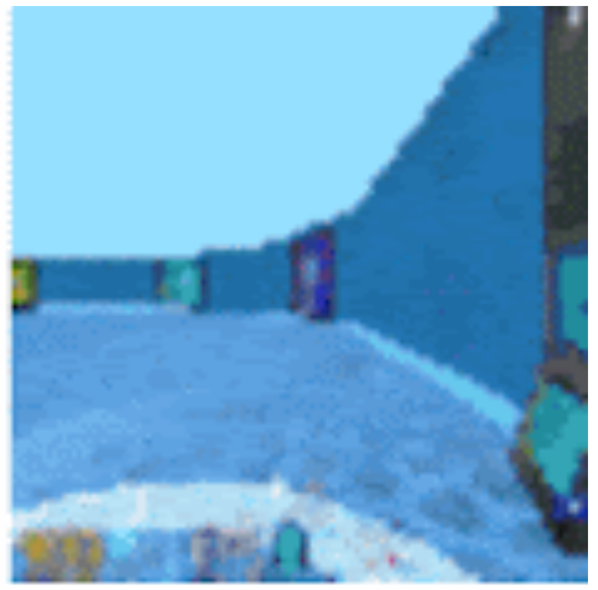
Look at the videos below


Boxing


Freeway


Frostbite


Collect Objects


Watermaze


Sparse Cartpole


Acrobot Swingup


Hopper Hop

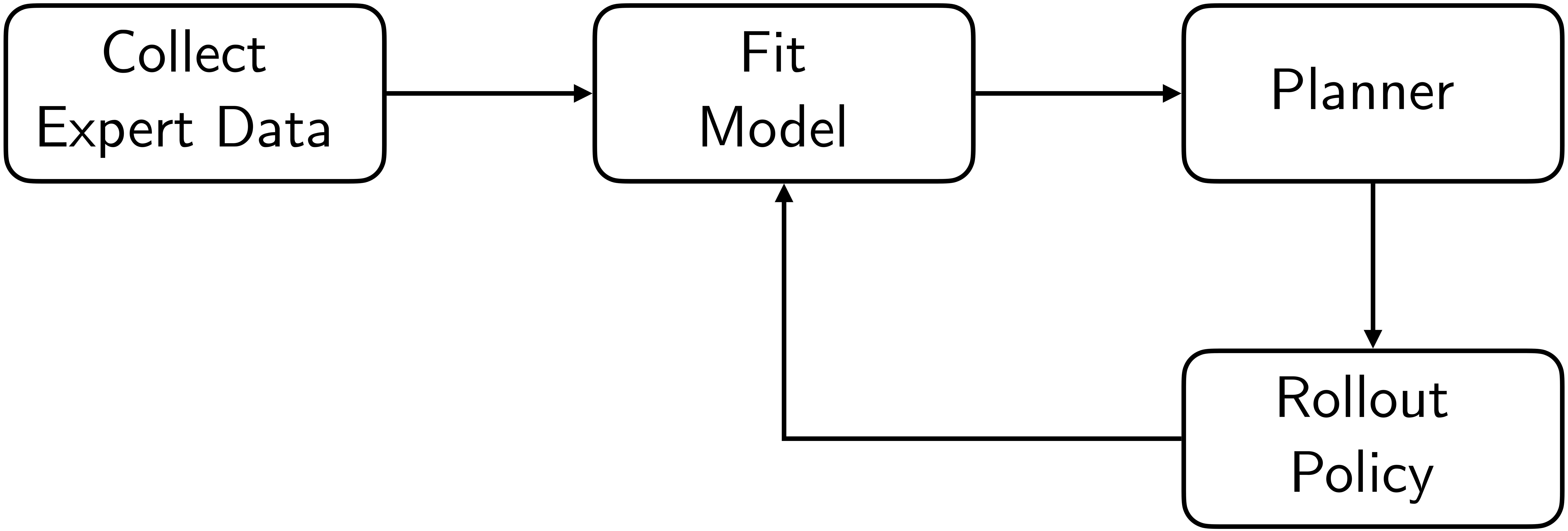

Walker Run


Quadruped Run

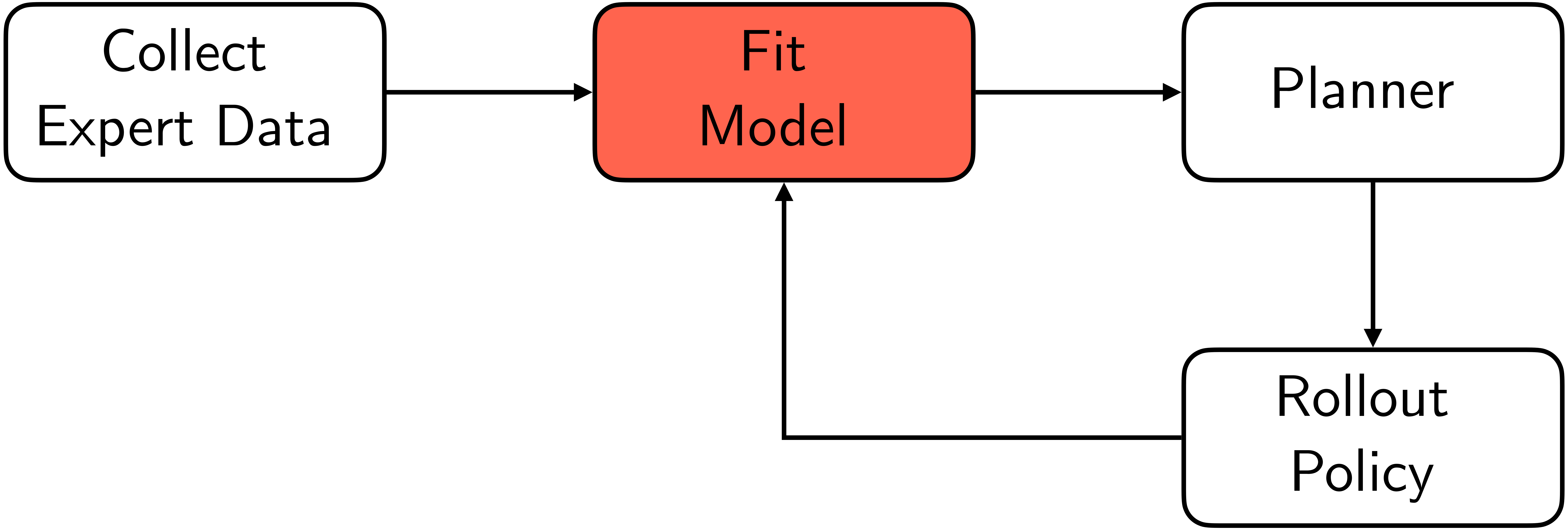
They are all from a model!

Recap: Model-based RL

(Ross & Bagnell, 2012)



How does DREAMER fit a model?



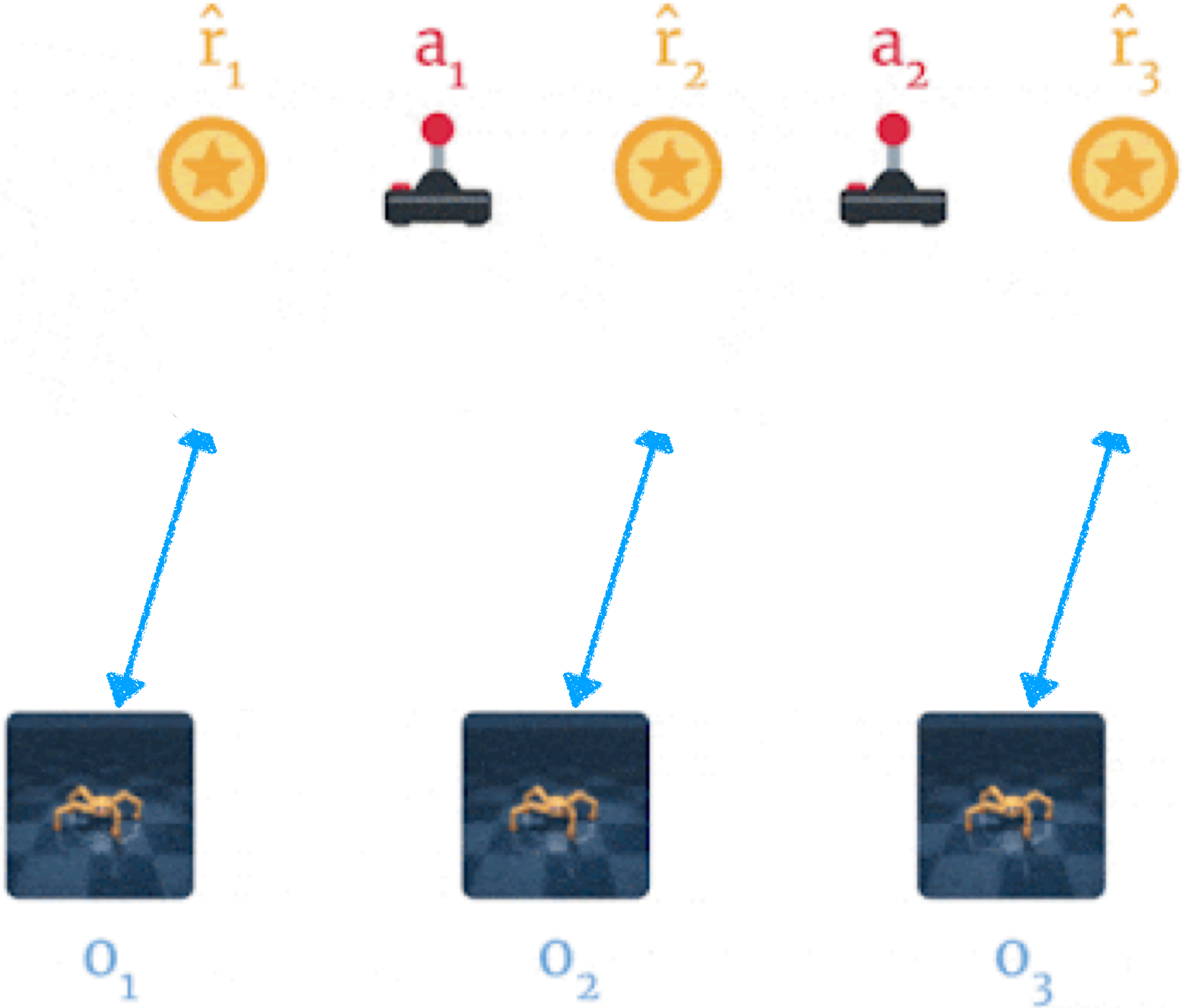
Goal: Fit a Model given data

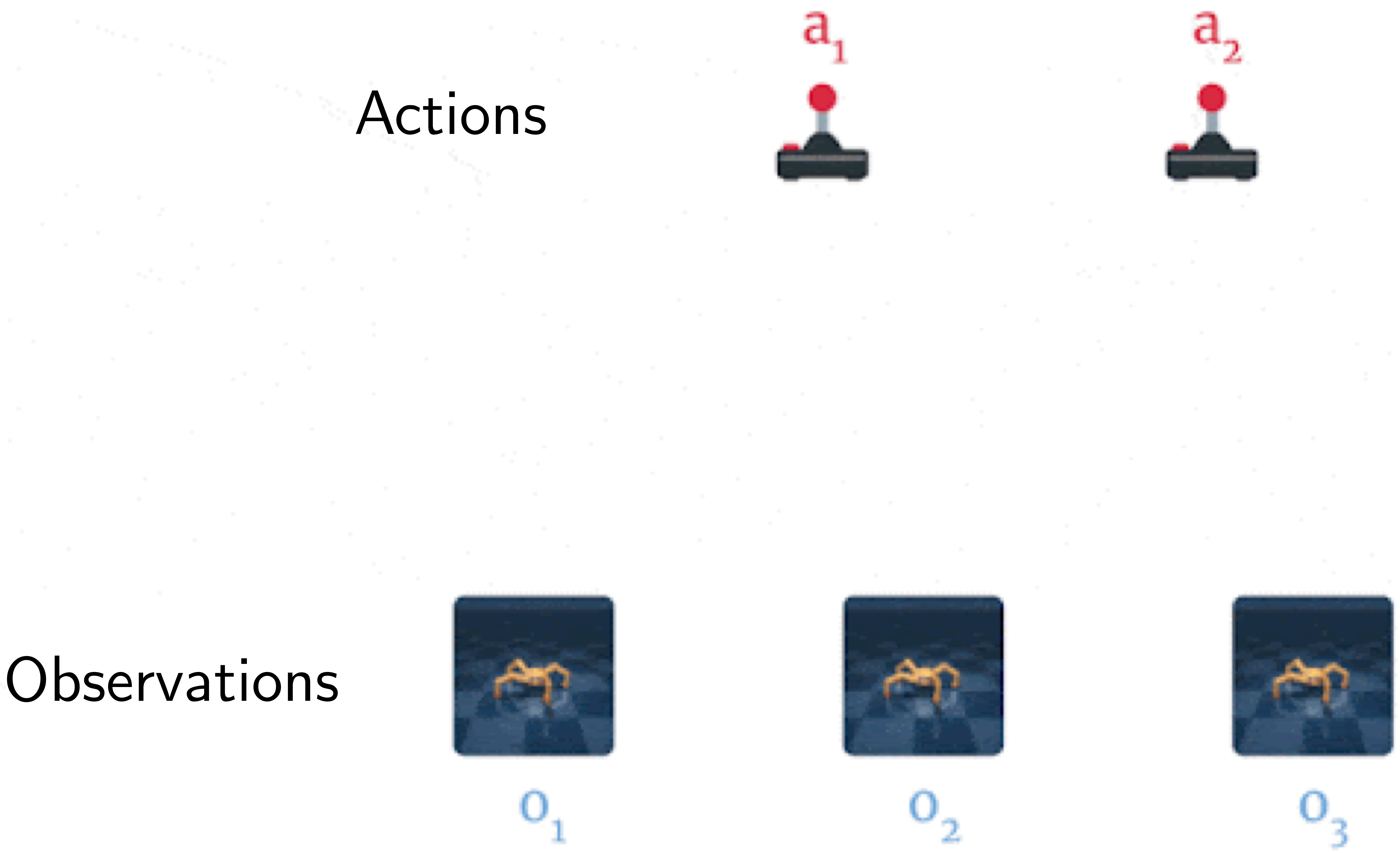
Given Data:
Observations, rewards,
actions

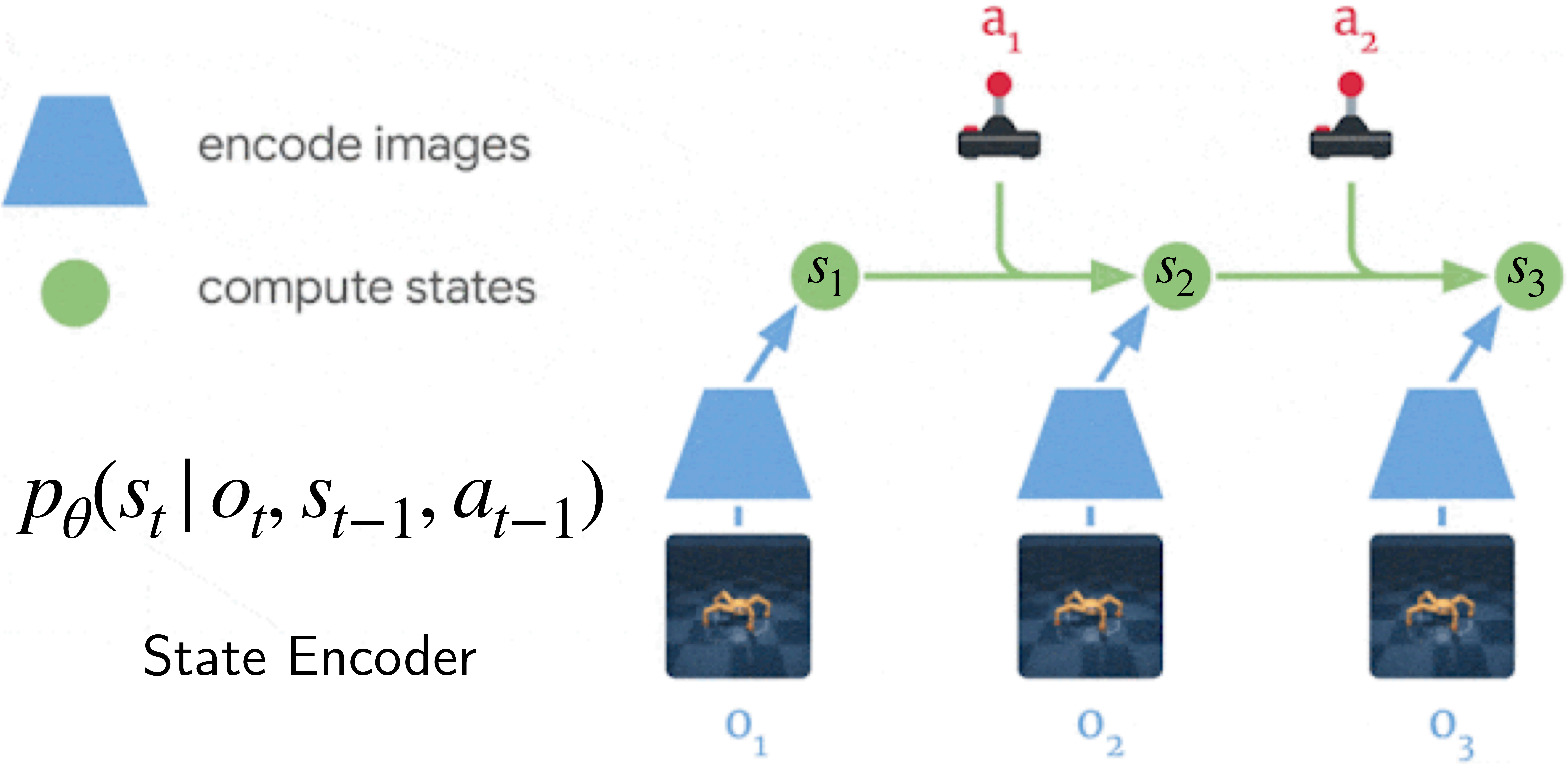
Goal: Fit a Model given data

Given Data:
Observations, rewards,
actions

Predict:
States,
Dynamics Function,
Reward Function




Actions
a_1
a_2
Observations
o_1
o_2
o_3



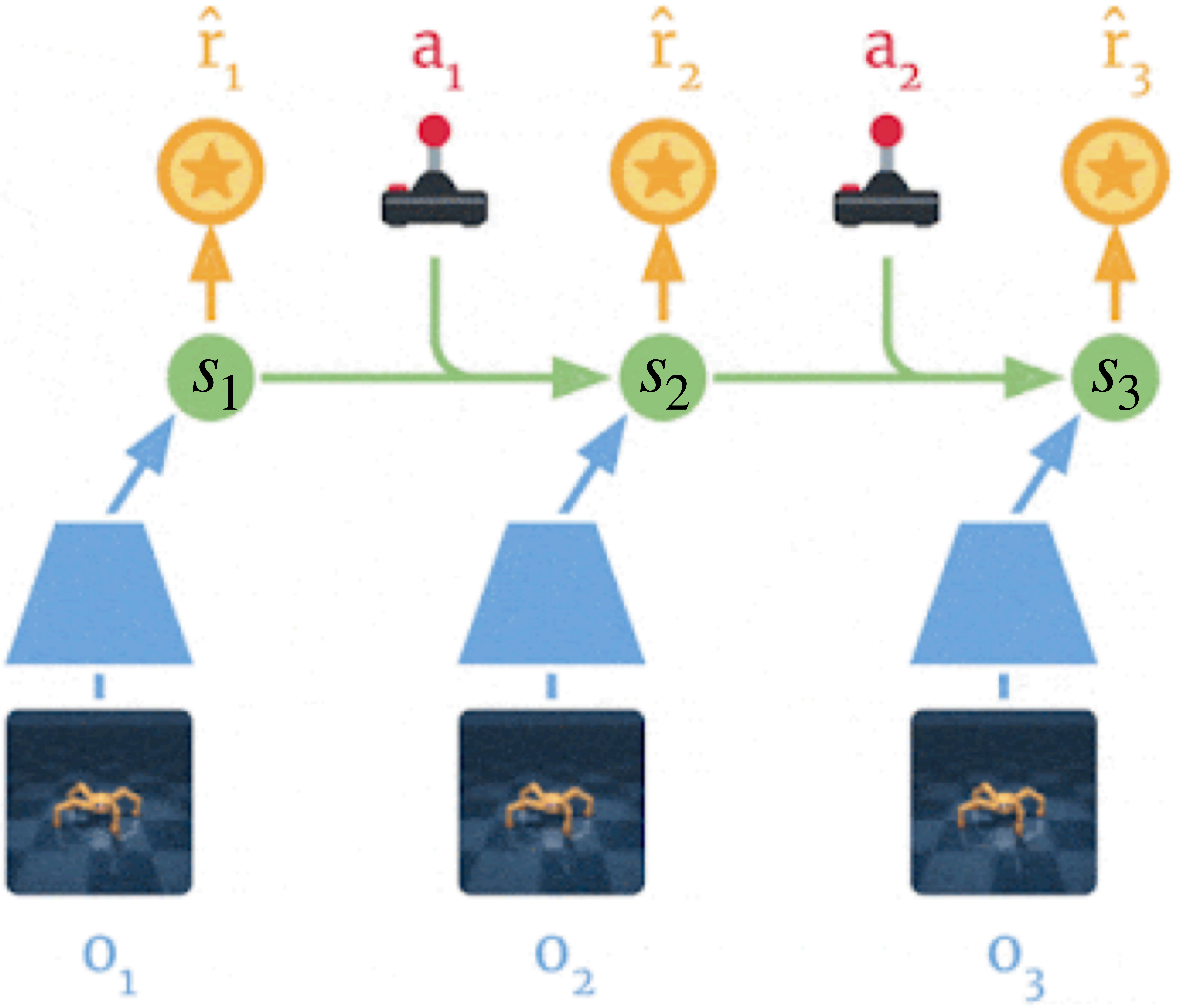
$$p_\theta(s_t \mid o_t, s_{t-1}, a_{t-1})$$

State Encoder

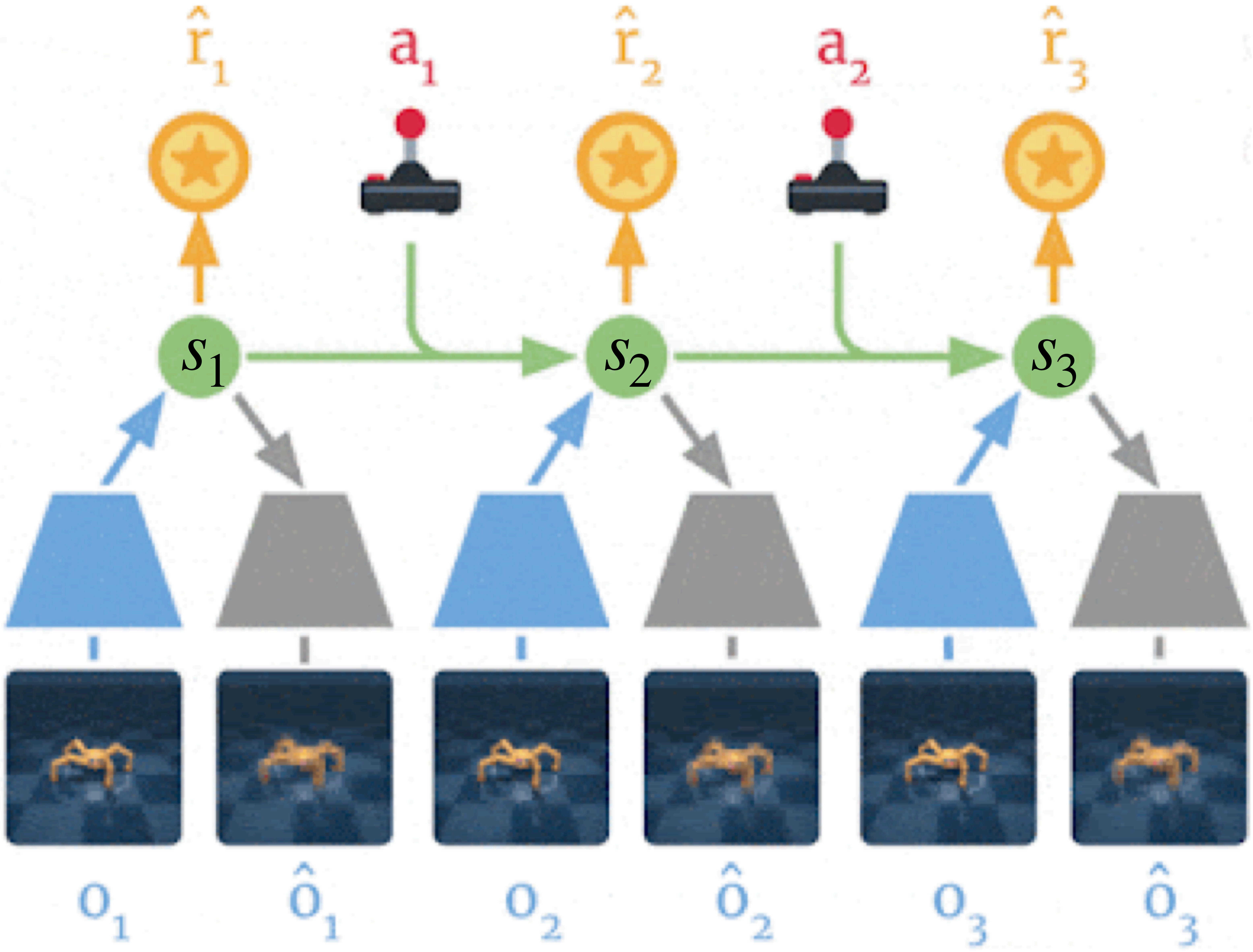
$$\ell = (r_t - \hat{r}_t)^2$$

$$q_\theta(r_t \mid s_t)$$

Reward Decoder

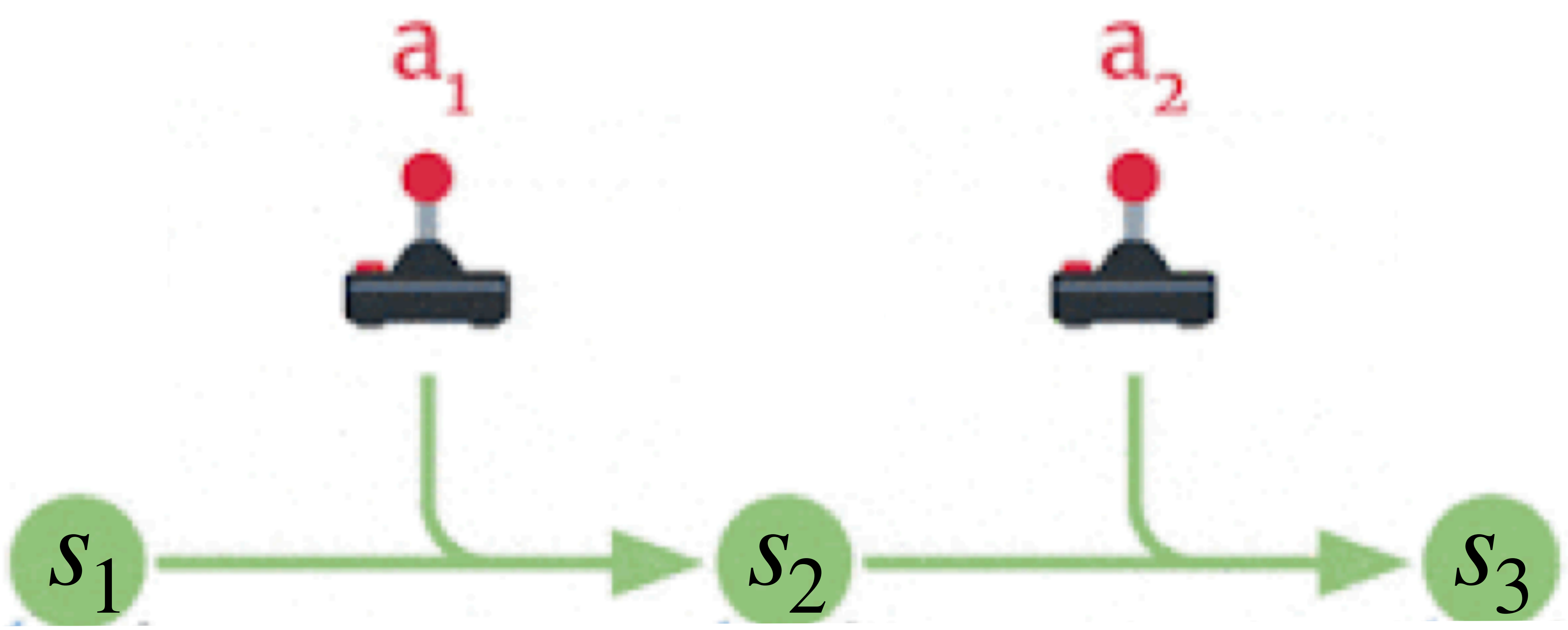


$$\ell = (o_t - \hat{o}_t)^2$$



$$q_\theta(o_t \,|\, s_t)$$

Observation Decoder



$$q_\theta(s_{t+1} \mid s_t, a_t)$$

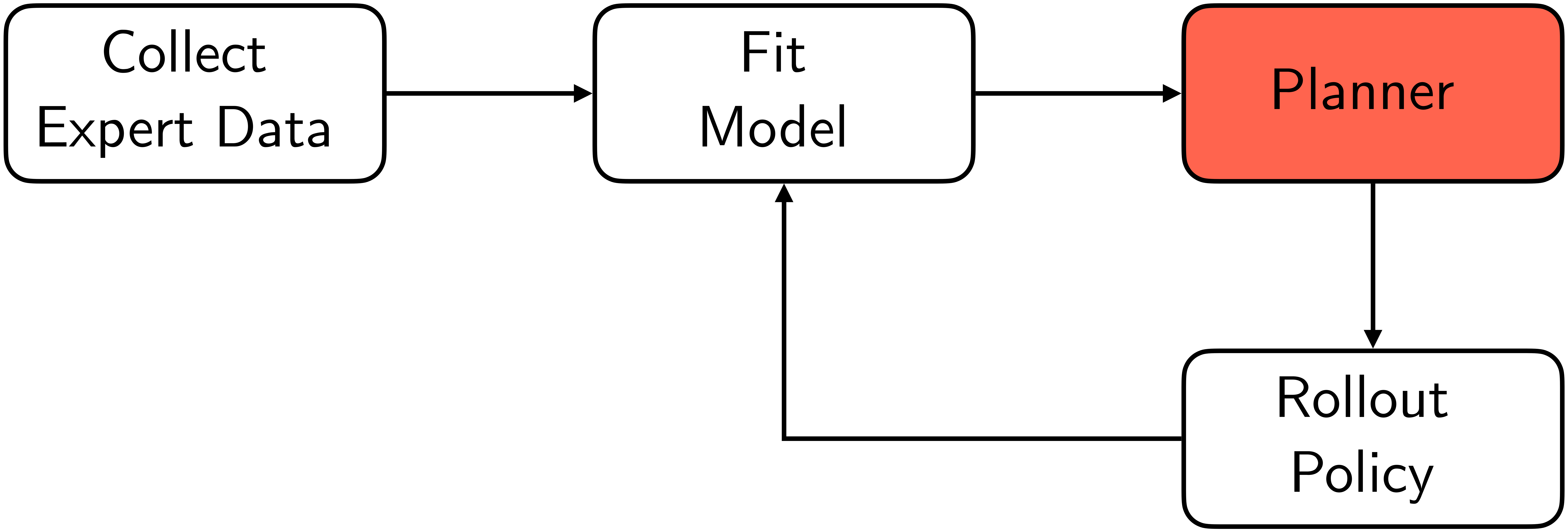
Dynamics
Function

$$\ell = KL(\underbrace{p_\theta(s_t \mid o_t, s_{t-1}, a_{t-1})}_{\text{Freeze gradients}} \mid\mid q_\theta(s_{t+1} \mid s_t, a_t))$$

Results: Learning World Model

Results: Learning World Model

How does DREAMER do planning?



Goal: Learn a Policy using Actor-Critic

$\pi_\phi(a_t \mid s_t)$

Actor

$V_\psi(s_t)$

Critic

From rollouts in the model

$q_\theta(s_t \mid s_{t-1}, a_{t-1})$

Recall: Actor-Critic for model-free RL

Start with an arbitrary initial policy $\pi_\phi(a \mid s)$

while *not converged* **do**

Roll-out $\pi_\phi(a \mid s)$ to collect trajectories $D = \{s^i, a^i, r^i, s^i_+\}_{i=1}^N$

Fit value function $V_\psi(s^i)$ using TD, i.e. minimize $(r^i + \gamma V_\psi(s^i_+) - V_\psi(s^i))^2$

Compute advantage $\hat{A}(s^i, a^i) = r(s^i, a^i) + \gamma V_\psi(s^i_+) - V_\psi(s^i)$

Compute gradient

$$\nabla_\phi J(\phi) = \frac{1}{N}\left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\phi(a^i_t \mid s^i_t)\, \hat{A}(s^i, a^i)\right]$$

Update parameters $\phi \leftarrow \phi + \alpha \nabla_\phi J(\phi)$

Actor-Critic in Model-based RL

Start with an arbitrary initial policy $\pi_\phi(a \mid s)$

while *not converged* **do**

Roll-out $\pi_\phi(a \mid s)$ in the model $q_\theta(s' \mid s, a)$ to collect trajectories $D = \{s^i, a^i, r^i, s_+^i\}_{i=1}^N$

Fit value function $V_\psi(s^i)$ using TD, i.e. minimize $(r^i + \gamma V_\psi(s_+^i) - V_\psi(s^i))^2$

Compute advantage $\hat{A}(s^i, a^i) = r(s^i, a^i) + \gamma V_\psi(s_+^i) - V_\psi(s^i)$

Compute gradient

$$\nabla_\phi J(\phi) = \frac{1}{N}\left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\phi(a_t^i \mid s_t^i)\, \hat{A}(s^i, a^i)\right]$$

Update parameters $\phi \leftarrow \phi + \alpha \nabla_\phi J(\phi)$

Actor-Critic in Model-based RL

Start with an arbitrary initial policy $\pi_\phi(a \mid s)$

while *not converged* **do**

Roll-out $\pi_\phi(a \mid s)$ in the model $q_\theta(s' \mid s, a)$ to collect trajectories $D = \{s^i, a^i, r^i, s^i_+\}_{i=1}^N$

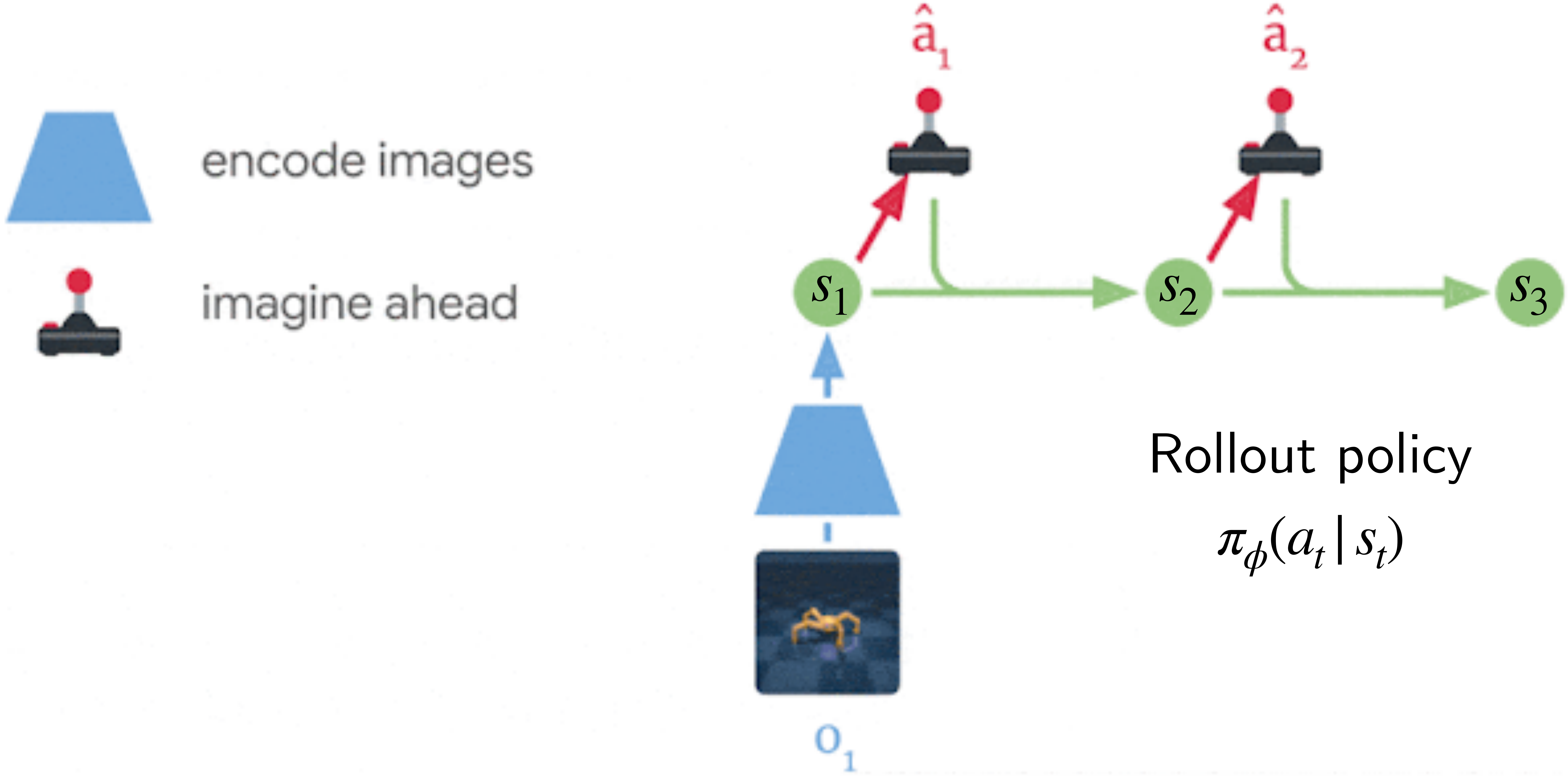
Fit value function $V_\psi(s^i)$ using TD, i.e. minimize $(r^i + \gamma V_\psi(s^i_+) - V_\psi(s^i))^2$

Directly backprop gradients through model to update policy!

$$\phi \leftarrow \phi + \alpha \nabla_\phi V_\psi(s^i)$$

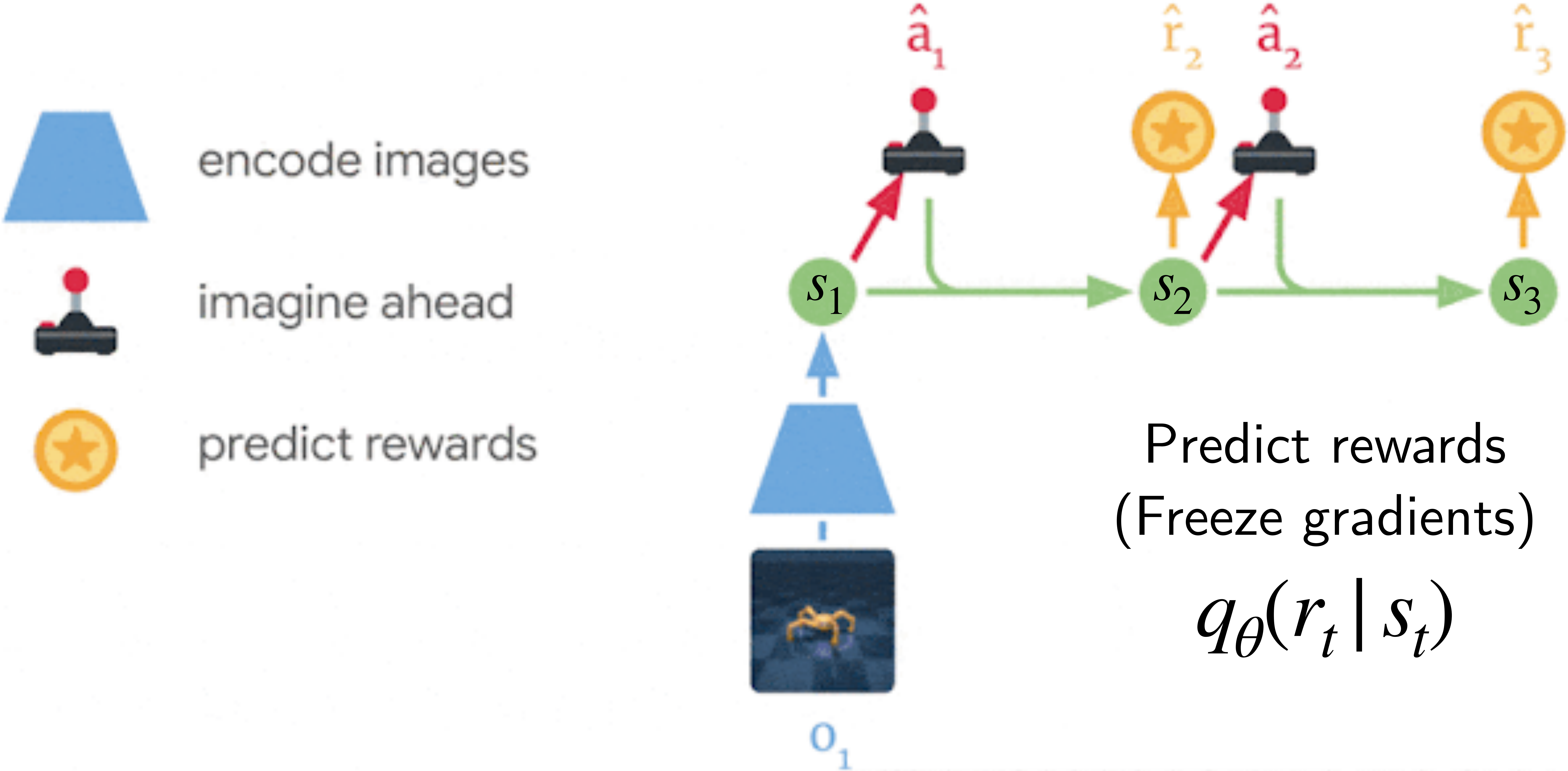
o_1

encode images



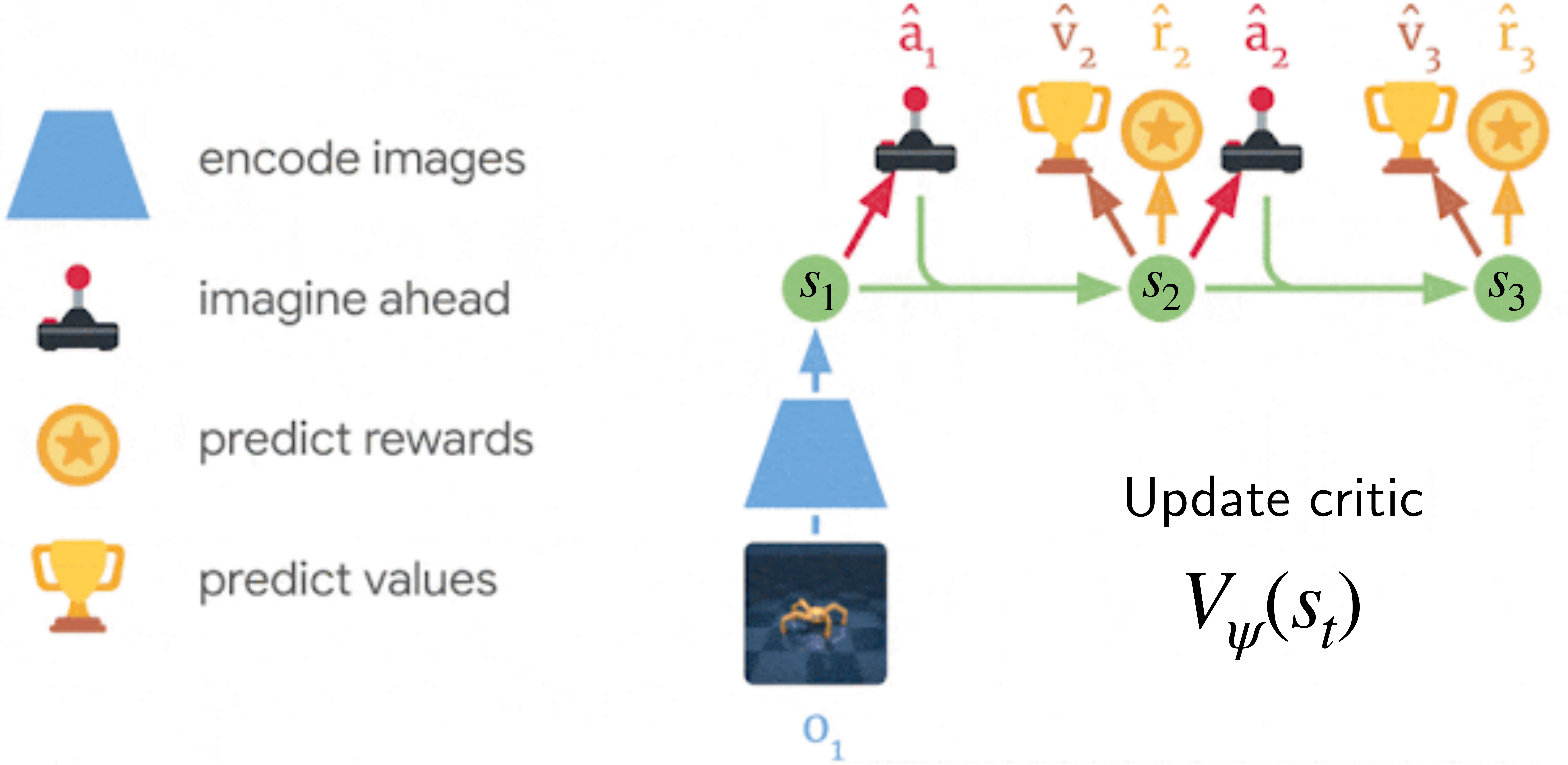
Rollout policy

$$\pi_\phi(a_t \,|\, s_t)$$



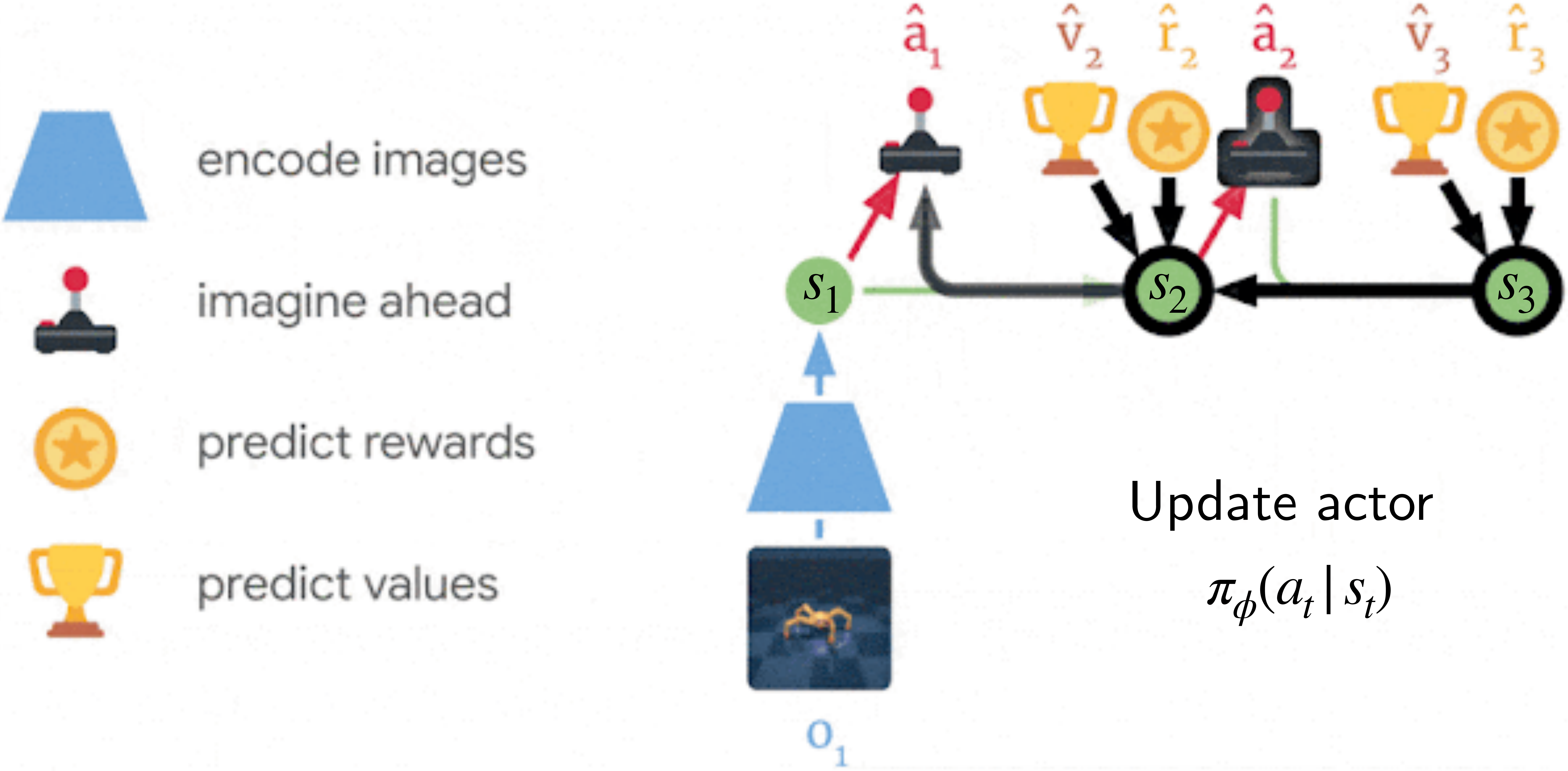
Predict rewards
(Freeze gradients)

$$q_\theta(r_t \mid s_t)$$



Update critic

$$V_\psi(s_t)$$



Update actor

$$\pi_\phi(a_t \mid s_t)$$

DREAMER: Results

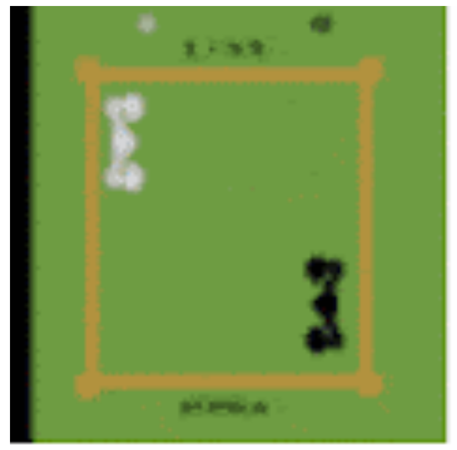

Sparse Cartpole

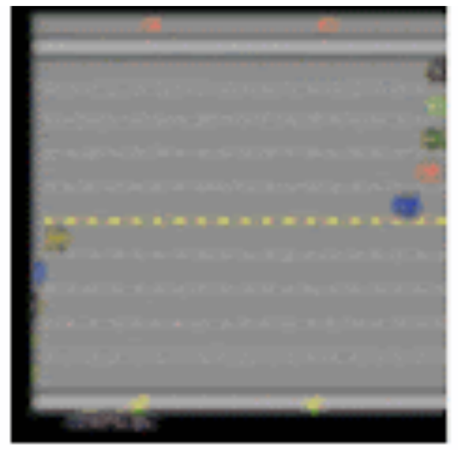

Acrobot Swingup

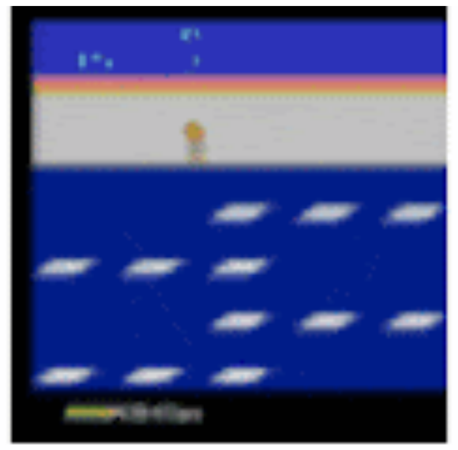

Hopper Hop

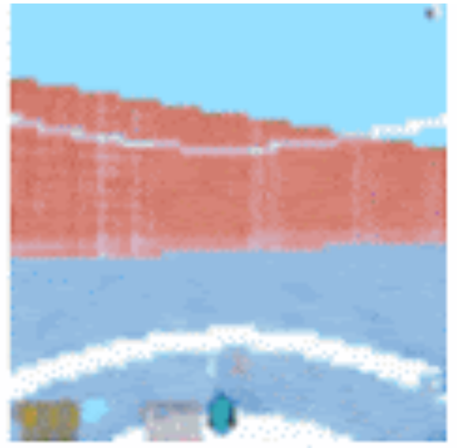

Walker Run


Quadruped Run

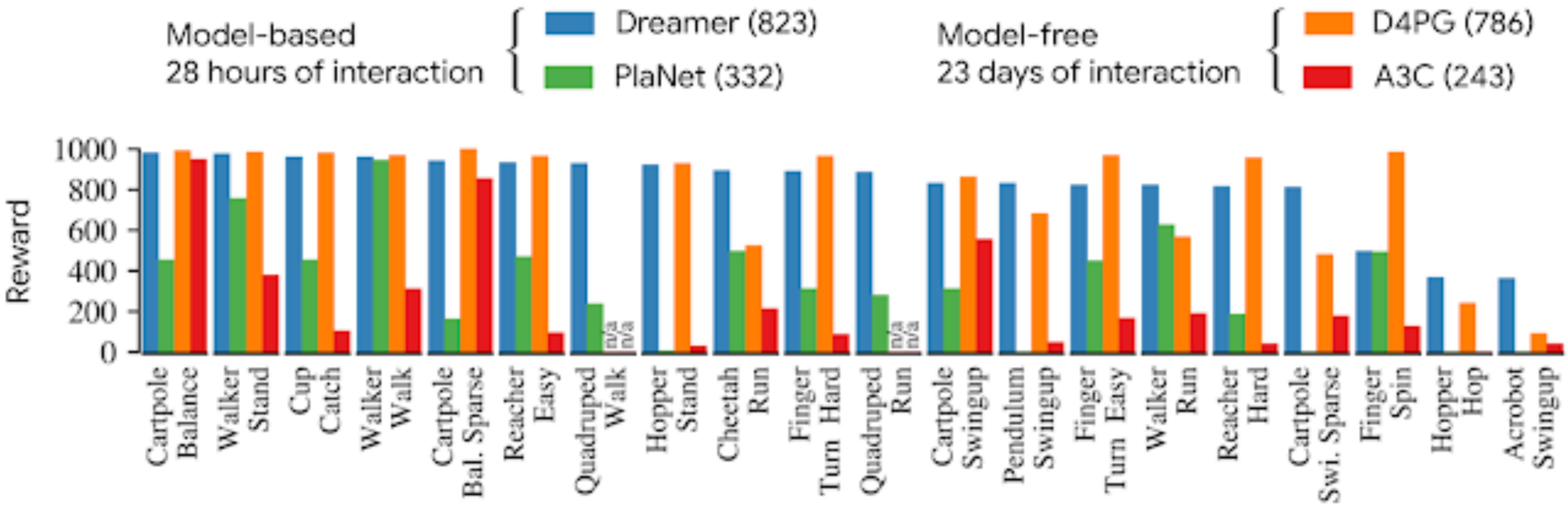

Boxing


Freeway


Frostbite


Collect Objects


Watermaze



DREAMER is a template
for Model-based RL

But there are many challenges as we
scale to harder real-world applications

DREAMER V2, V3, etc

Today's class

- ☑ Practical MBRL

(Only observations, complex dynamics)

- ☑ Learning Models: The DREAMER algorithm

- ☐ Leveraging Physics: Sim2Real

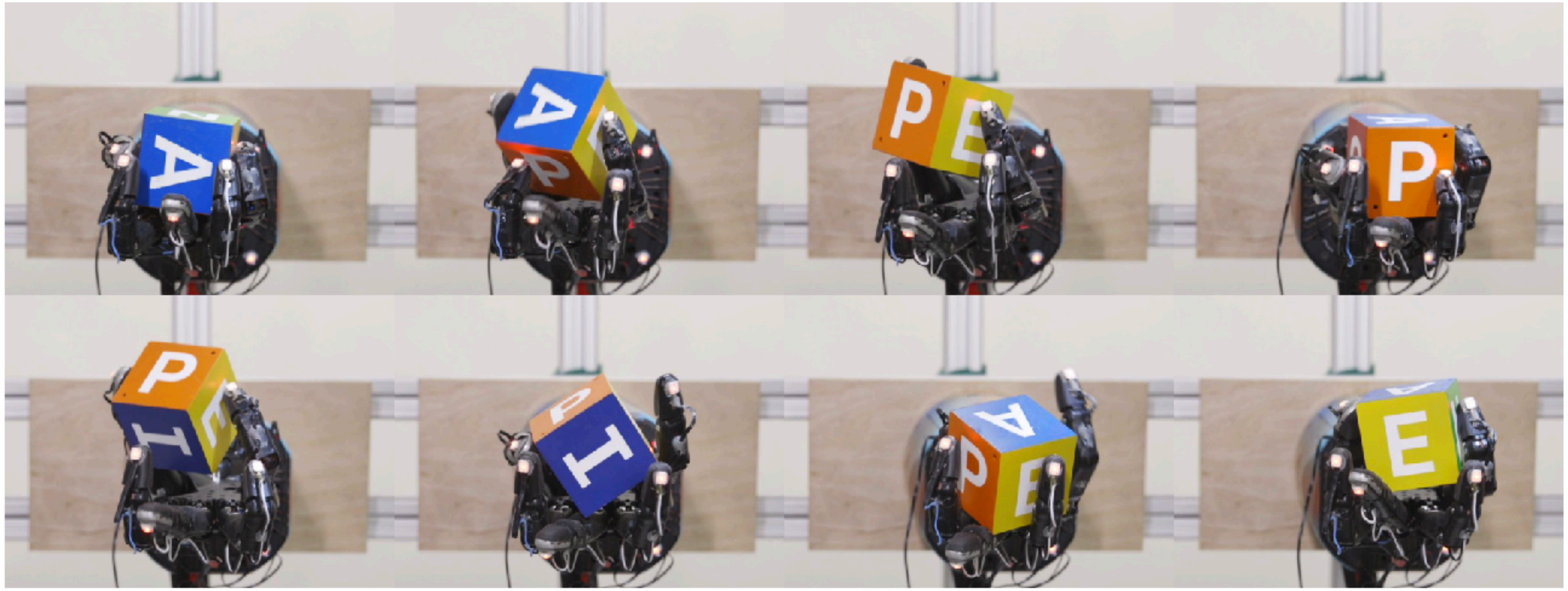
  Case study: OpenAI Dactyl Hand

Learning Dexterity

(Open AI)

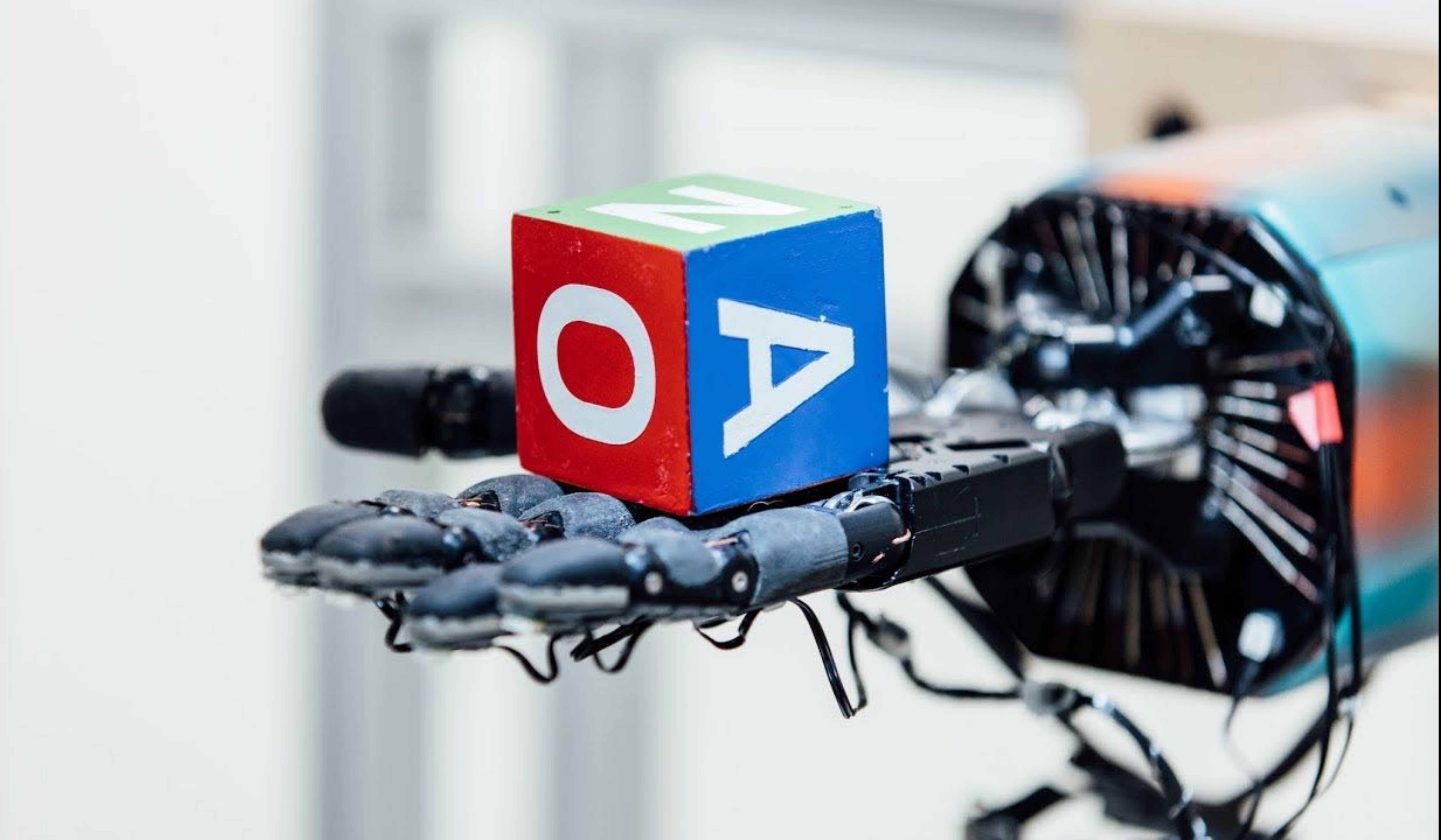
Learning Dexterous In-Hand Manipulation

OpenAI,* Marcin Andrychowicz, Bowen Baker, Maciek Chociej,
Rafał Józefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron,
Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor,
Josh Tobin, Peter Welinder, Lilian Weng, Wojciech Zaremba



Initial configuration

Goal configuration


O
A

Sim

Train a policy in simulation

(RL)

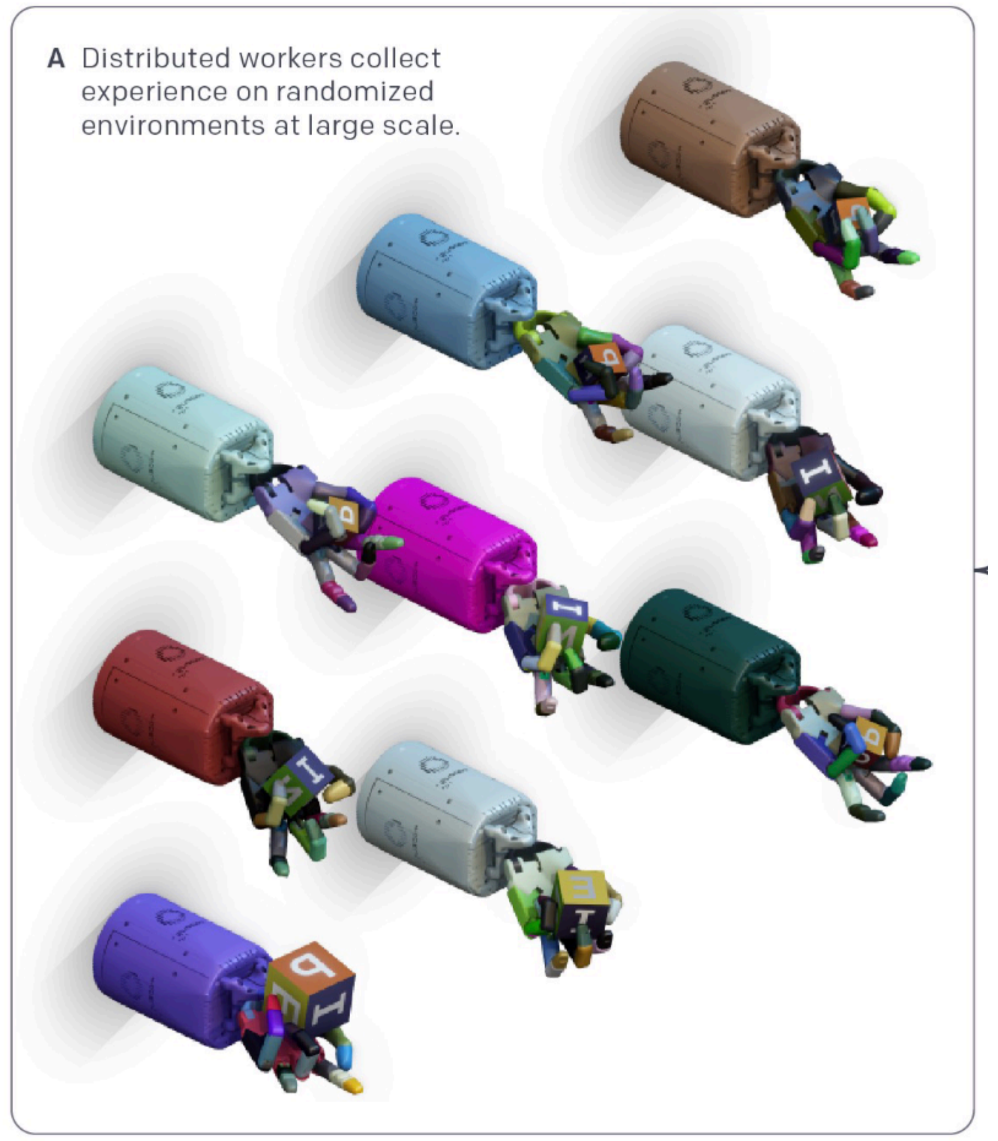
Real

Test in real world

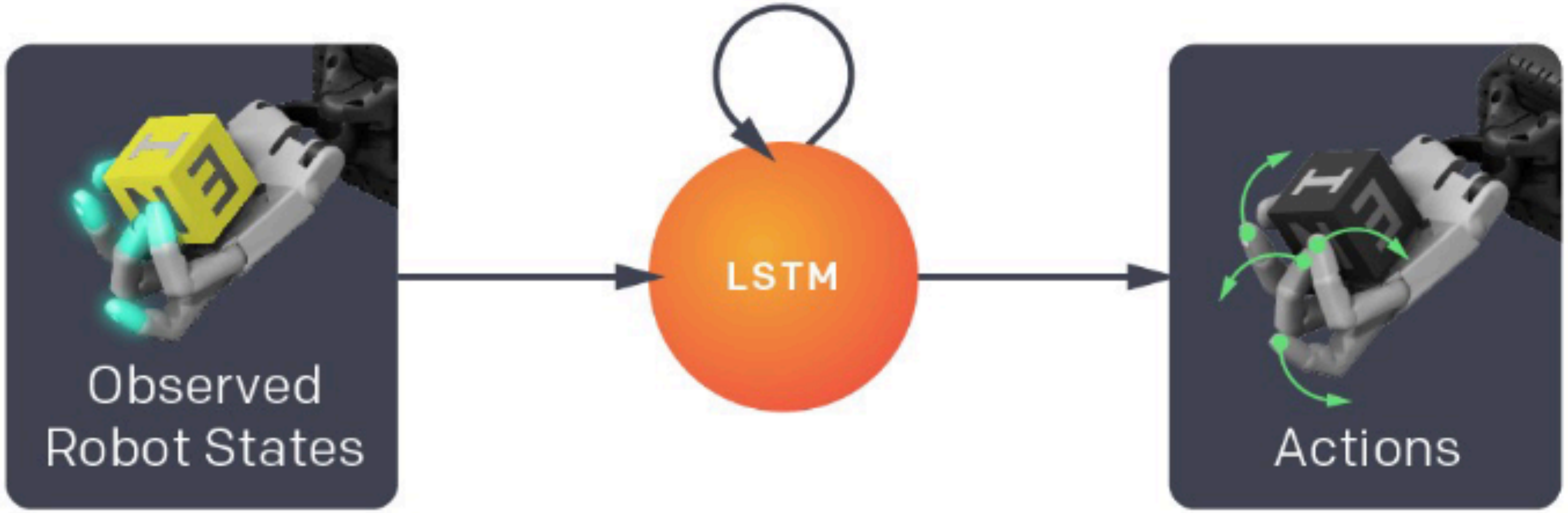
Sim

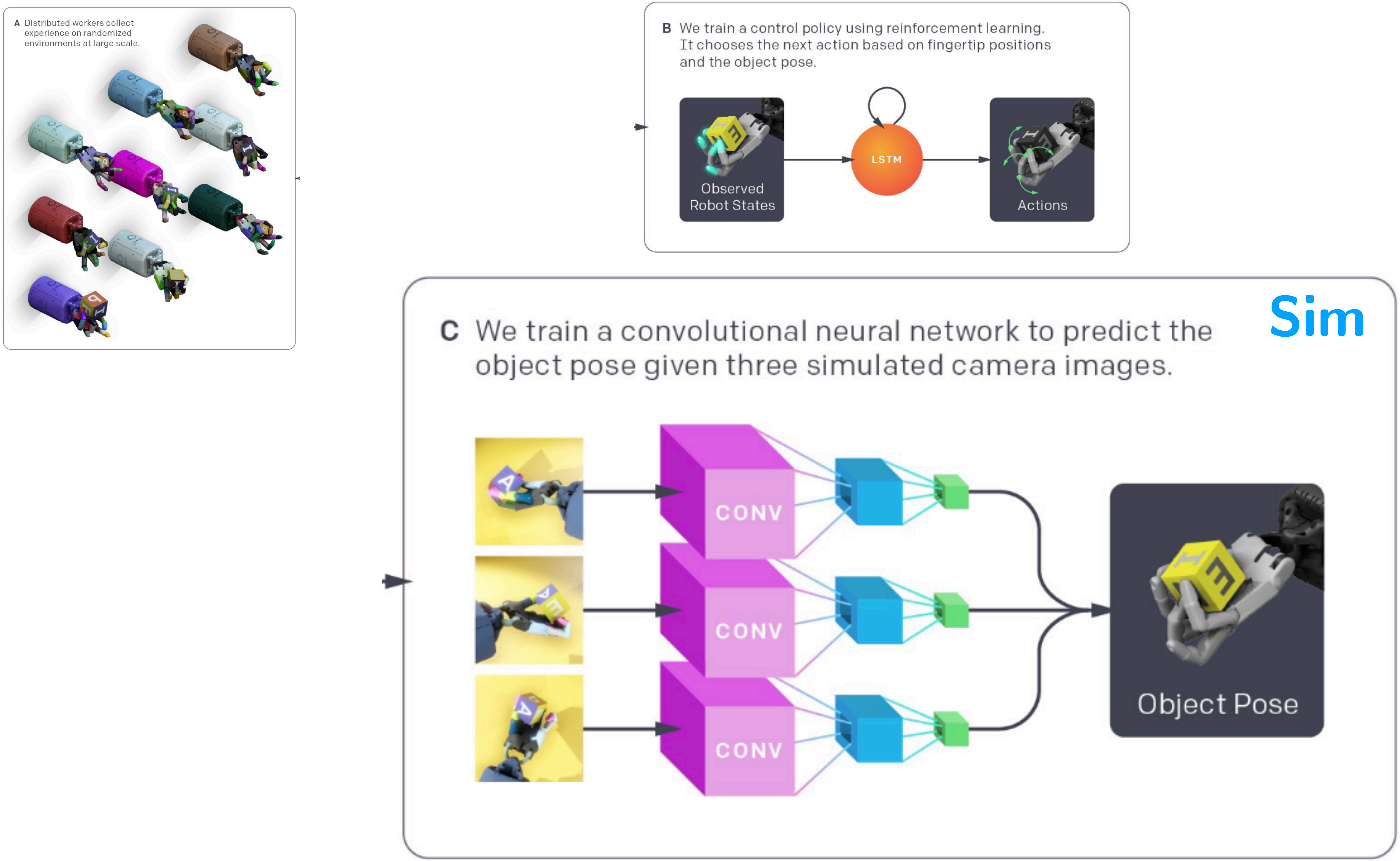

A Distributed workers collect experience on randomized environments at large scale.

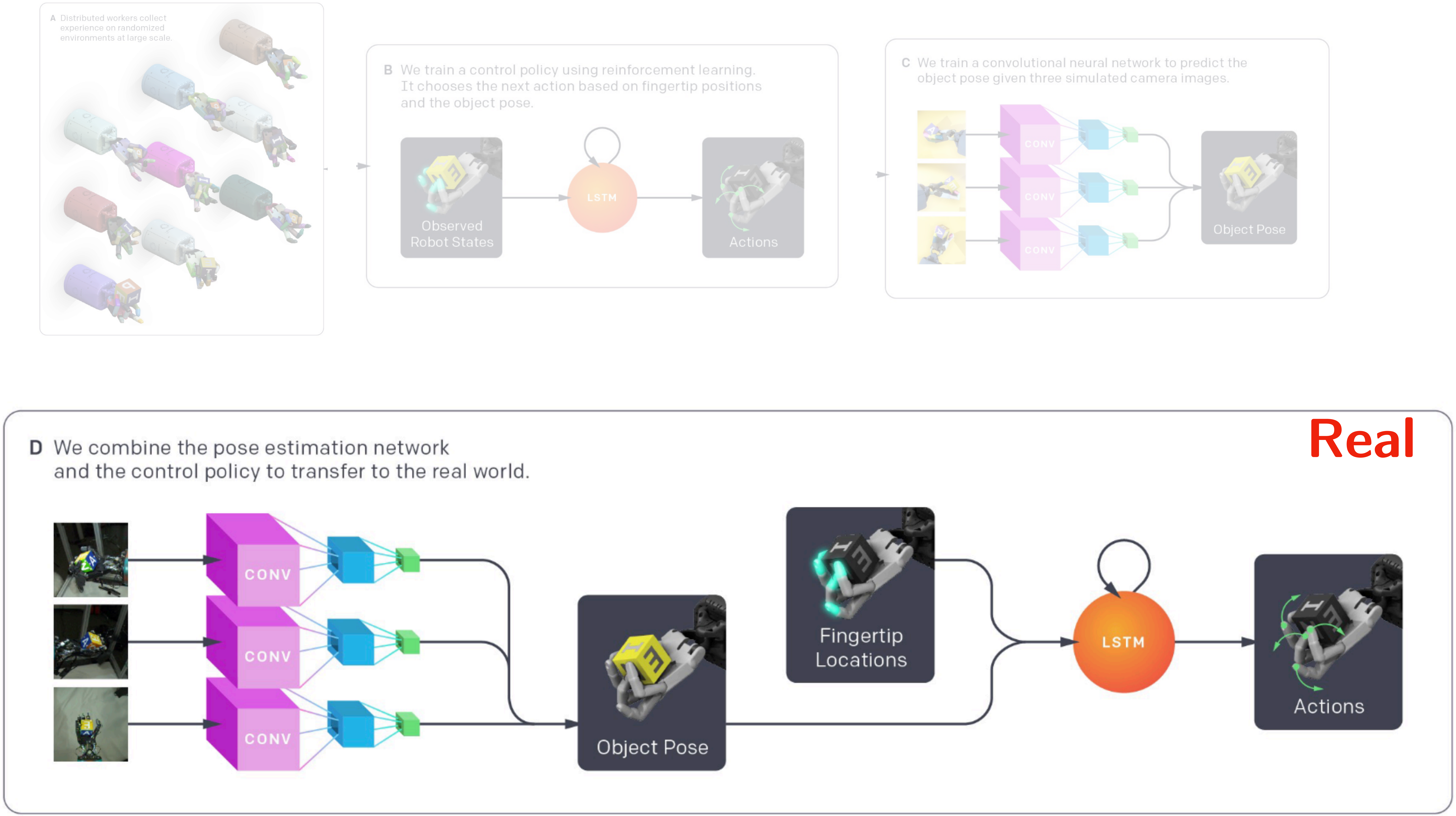
Sim



B We train a control policy using reinforcement learning. It chooses the next action based on fingertip positions and the object pose.

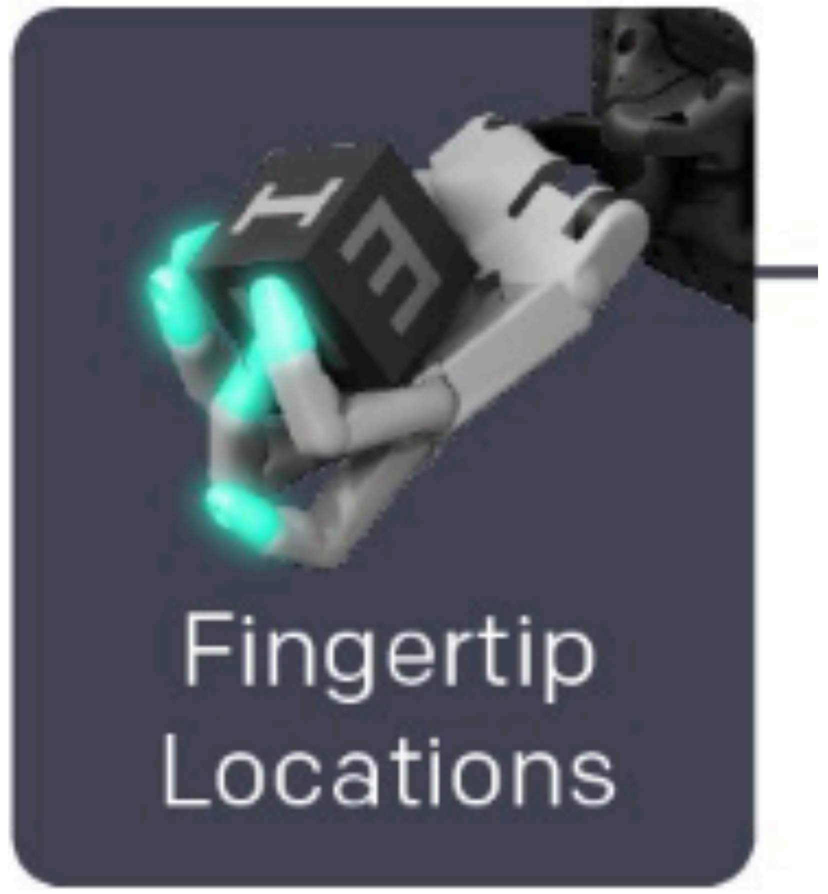



A Distributed workers collect experience on randomized environments at large scale.
B We train a control policy using reinforcement learning. It chooses the next action based on fingertip positions and the object pose.
Observed Robot States
LSTM
Actions
C We train a convolutional neural network to predict the object pose given three simulated camera images.
Sim
CONV
CONV
CONV
Object Pose


A Distributed workers collect experience on randomized environments at large scale.
B We train a control policy using reinforcement learning. It chooses the next action based on fingertip positions and the object pose.
Observed Robot States
LSTM
Actions
C We train a convolutional neural network to predict the object pose given three simulated camera images.
CONV
CONV
CONV
Object Pose
D We combine the pose estimation network and the control policy to transfer to the real world.
Real
CONV
CONV
CONV
Object Pose
Fingertip Locations
LSTM
Actions

$$\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{R}, \mathscr{T}$$

$$\boxed{\mathcal{S}}, \mathcal{A}, \mathcal{R}, \mathcal{T}$$

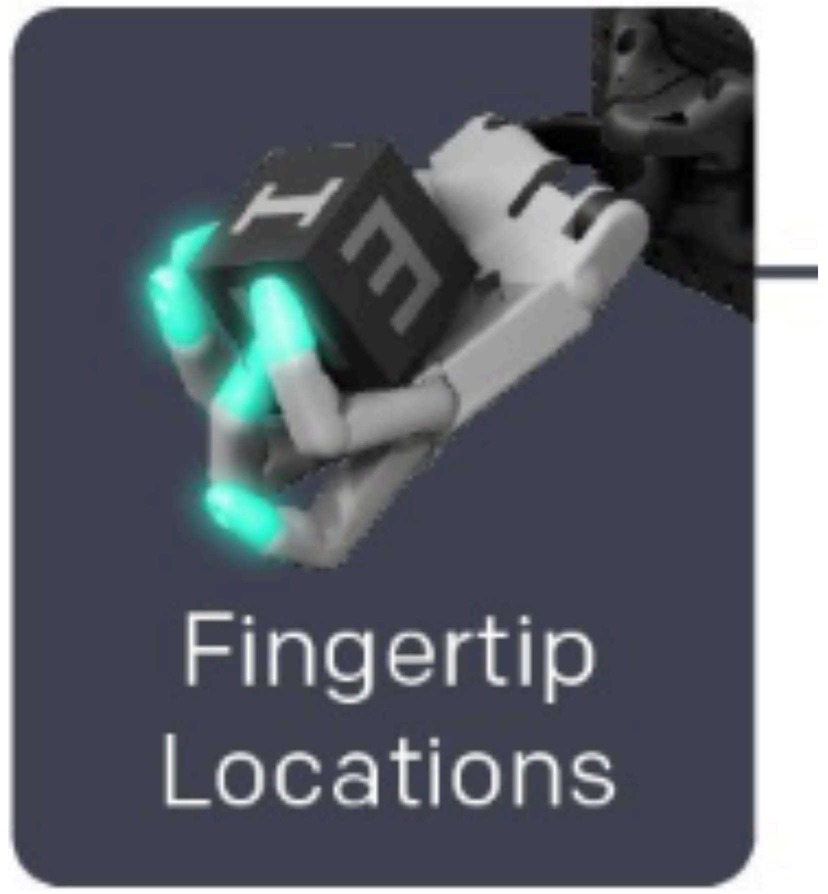


Question: Is the current object pose and fingertip location sufficient to capture state?

$$\boxed{S}, A, R, \mathcal{T}$$

No!

This is merely the current observation of a POMDP



Need to keep a HISTORY

E.g. History of observations can reveal the weight of the object or how fast the index finger can move.

$$S\ ,\ \boxed{A}\ ,\ R\ ,\ \mathcal{T}$$

$$S\,,\,A\,,\,\boxed{R}\,,\,\mathcal{T}$$

The reward given at timestep t is $r_t = d_t - d_{t+1}$, where d_t and d_{t+1} are the rotation angles between the desired and current object orientations before and after the transition, respectively. We give an additional reward of 5 whenever a goal is achieved and a reward of -20 (a penalty) whenever the object is dropped. More information about the simulation environment can be found in Appendix C.1.

Sim2Real as Transferring MDPs

$$\hat{S}, A, R, \hat{\mathcal{T}} \rightarrow S, A, R, \mathcal{T}$$

Sim

Real

There will be a mismatch in state representations and transition

Our policy needs to be robust to this mismatch

Key Idea: Add in Randomization in Sim

1. Randomize the observation

Observation noise. To better mimic the kind of noise we expect to experience in reality, we add Gaussian noise to policy observations. In particular, we apply a correlated noise which is sampled once per episode as well as an uncorrelated noise sampled at every timestep.

Key Idea: Add in Randomization in Sim

1. Randomize the observation

2. Randomize the physics

Physics randomizations. Physical parameters like friction are randomized at the beginning of every episode and held fixed. Many parameters are centered on values found during model calibration in an effort to make the simulation distribution match reality more closely. Table 1 lists all physics parameters that are randomized.

Key Idea: Add in Randomization in Sim

1. Randomize the observation
2. Randomize the physics
3. Unmodeled effects

Unmodeled effects. The physical robot experiences many effects that are not modeled by our simulation. To account for imperfect actuation, we use a simple model of motor backlash and introduce action delays and action noise before applying them in simulation. Our motion capture setup sometimes loses track of a marker temporarily, which we model by freezing the position of a simulated marker with low probability for a short period of time in simulation. We also simulate marker occlusion by freezing its simulated position whenever it is close to another marker or the object. To handle additional unmodeled dynamics, we apply small random forces to the object. Details on the concrete implementation are available in Appendix C.2.

Key Idea: Add in Randomization in Sim

Visual appearance randomizations. We randomize the following aspects of the rendered scene: camera positions and intrinsics, lighting conditions, the pose of the hand and object, and the materials and textures for all objects in the scene. Figure 4 depicts some examples of these randomized environments. Details on the randomized properties and their ranges are available in Appendix C.2.

1. Randomize the observation
2. Randomize the physics
3. Unmodeled effects
4. Visual randomization

Today's class

- ☑ Practical MBRL

  (Only observations, complex dynamics)

- ☑ Learning Models: The DREAMER algorithm

- ☑ Leveraging Physics: Sim2Real

  Case study: OpenAI Dactyl Hand