UCI University of California, Irvine

CS 277: Control and Reinforcement Learning

Winter 2026

Lecture 12: Model-Based Methods

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

Logistics

assignments

- Quiz 6 due Friday
- We'll only have 8 quizzes
- Exercise 4 to be published soon, due next Friday

Today's lecture

Model-based learning

Model-free RL with a model

Model-predictive control

Modeling POMDPs

Learning vs. planning

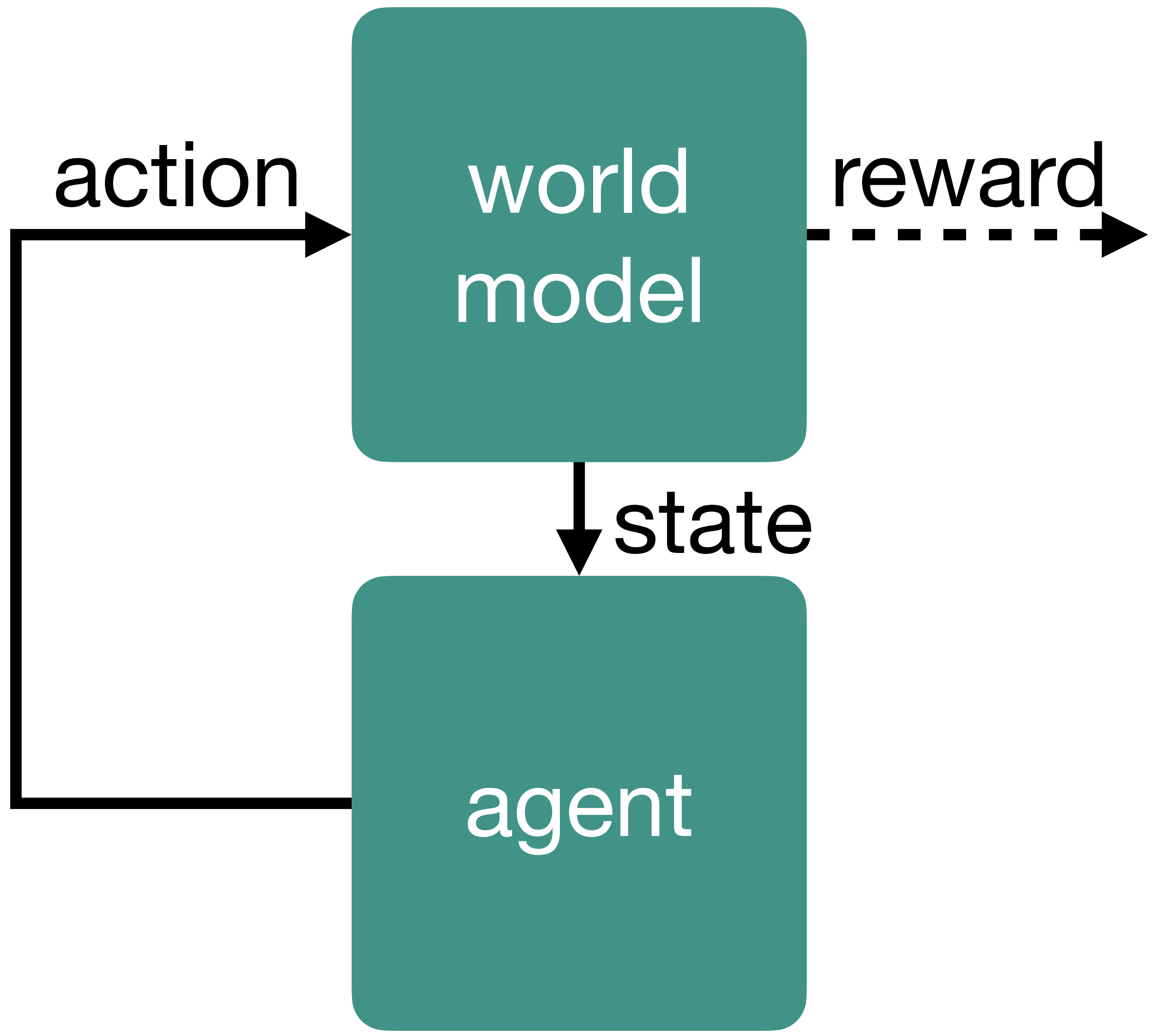
- Model = dynamics + reward function
  - Planning = finding a good policy with access to a model
- Learning = improving performance using data
  - Are rollouts from the model considered “data”?
    - If yes, planning can involve learning
- Model-based learning = methods that explicitly learn the model
  - Unlike planning, access to a model is not given; it is learned
  - Usually, focus on dynamics p, because reward function r is simulated

Model-based learning

- Is a learning algorithm $\mathscr{A}$ model-based?
  - Not to be confused with ML terminology calling anything learned a "model"
- In tabular representation — just count parameters:
  - Model-free = $O(|\mathcal{S}| \cdot |\mathcal{A}|)$ (to represent $\pi(a|s)$ or $Q(s,a)$)
  - Model-based = $\Omega(|\mathcal{S}|^2 \cdot |\mathcal{A}|)$ (to represent $p(s'|s,a)$)
- In CogSci: model-based = learning / inference about non-current instances
- Function approximation: $Q_\theta(s,a)$ is informative of s', is this model-based?

What is a world model?

- Imagination of counterfactual actions
  - and their effects on rewards, future states
- Why model the world?
  - Can be more data-efficient to learn
  - A (low-dim) simulator can make RL easier, be transferable, interpretable, etc.
  - As a memory process for agent deployment (more on this later)



Why model the world? Data efficiency

- Sample efficiency
  - Despite estimating “more” parameters
  - Supervision signal $(s_t, a_t) \mapsto s_{t+1}$ is much more informative per sample
- Generalization
  - Optimal value / policy is a global property; model is local
  - But the model needs to be good in all states; policy only in states it reaches
- Data abundance
  - Web-scale trajectory data; actions / rewards scarcer; can use other supervision

Why model the world? It's extra useful

- Fast simulation: MFRL / planning in imagination
- Arbitrary reset ("teleporting robot") simulation: adversarial training
- Differentiable simulation: locally-counterfactual value information
- Search / MPC: more compute in on-policy states
- Transferability: multi-task, non-stationarity, multi-agent
- Low-dimensional latent state: interpretable, debuggable, explainable
- And more: safety, causal inference, uncertainty quantification

- Fast
- Resettable
- Differentiable

How to learn a model

- Interact with (fully observable) environment to get trajectory data
  - Deterministic continuous dynamics / reward: minimize MSE loss

$$\mathcal{L}_\phi(s, a, r, s') = \|s' - f_\phi(s, a)\|_2^2 + (r - r_\phi(s, a))^2$$

  - Stochastic dynamics: minimize NLL loss

$$\mathcal{L}_\phi(s, a, s') = -\log p_\phi(s' | s, a)$$

- Data can be off-policy ⇒ unbiased estimate, but with covariate shift
  - Random policy is often used

Policy Gradient through the model

- Model is often learned with SGD ⇒ must be differentiable

$$\hat{J}_\theta = \sum_t \gamma^t \hat{c}(x_t, u_t) = \sum_t \gamma^t \hat{c}(\hat{f}(\cdots \hat{f}(x_0, \pi_\theta(x_0))\cdots, \pi_\theta(x_{t-1})), \pi_\theta(x_t))$$

- Just do Policy Gradient over $\hat{J}_\theta$?
  - Chain rule ⇒ back-propagation through time (BPTT)
- $\nabla_\theta \hat{J}_\theta$ can be bad approximation of $\nabla_\theta J_\theta$; also, $\hat{J}_\theta$ is ill-conditioned for SGD:
  - Perturbing one action individually may change $\hat{J}_\theta$ unreasonably little / much
    - Vanishing / exploding gradients
  - Second-order methods can help, but Hessian is even nastier — for the same reason

- Fast
- Resettable
- Differentiable

PG with a model

- Luckily, we have the Policy Gradient Theorem

$$\nabla_\theta \hat{J}_\theta = \mathbb{E}_{\xi \sim p_\theta}\left[\sum_t \gamma^t \hat{Q}_{\bar{\theta}}(s_t, a_t) \nabla_\theta \log \pi_\theta(a_t | s_t)\right]$$

- Idea: use the model as a fast simulator just to estimate $\hat{Q}_{\bar{\theta}}(s_t, a_t)$
  - E.g., by MC or TD
  - Avoids complications of gradients through the model
    - Only backprop through single-step $\log \pi_\theta(a_t | s_t)$
  - Only the policy evaluation / critic is model-based

- Fast
- Resettable
- Differentiable

Today's lecture

Model-based learning

Model-free RL with a model

Model-predictive control

Modeling POMDPs

Model-free RL with a model

- General scheme for using a model for model-free RL:

Algorithm Model-free RL with a model

Collect data ← interaction with environment (random policy)

Train model $\hat{p}, \hat{r}$ ← supervised learning

repeat

Sample s from the replay buffer ← seeded by initial interaction; may interact more as learner improves

Sample $(a|s) \sim \pi_\theta$

Simulate $r = \hat{r}(s,a)$ and $(s'|s,a) \sim \hat{p}$ ← use model as simulator

Perform model-free RL with (s,a,r,s')

- Benefit: get diverse off-policy s, and fresh on-policy a

- Fast
- Resettable
- Differentiable

Model-free RL with a model

- On-policy actions ⇒ allows n-step estimation without bias:

Algorithm Multi-step RL with a model

Collect data
Train model $\hat{p}, \hat{r}$
repeat
  Sample s from the replay buffer
  Roll out the learner's policy for n steps in the simulator
  Perform n-step model-free RL

- $\hat{r}(s_t, a_t) + \gamma \hat{r}(\hat{s}_{t+1}, a_{t+1}) + \cdots + \gamma^{n-1} \hat{r}(\hat{s}_{t+n-1}, a_{t+n-1})$ is unbiased
  - Except for model inaccuracy

- Fast
- Resettable
- Differentiable

Dyna

Algorithm Dyna

Collect data
Train model $\hat{p}, \hat{r}$
repeat
  Sample (s, a) from the replay buffer
  $Q(s,a) \to \hat{r}(s,a) + \gamma\, \mathbb{E}_{(s'|s,a)\sim\hat{p}}[\max_{a'} Q(s',a')]$

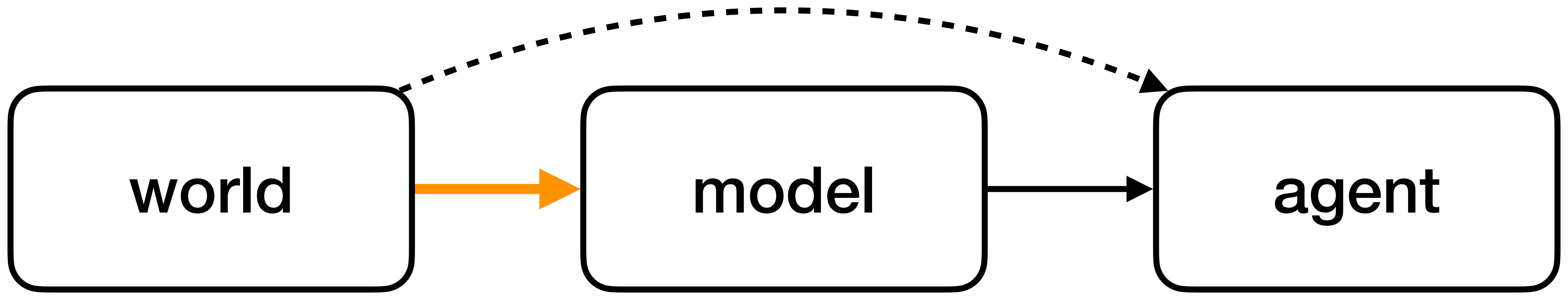
use model as simulator to estimate

- Another idea: also mix in samples generated from learner interactions
  - Benefit: keep training the model to be good for states that learner sees
  - Function approximation: feed the replay buffer and reduce covariate shift

- Fast
- Resettable
- Differentiable

[Sutton, Dyna, an Integrated Architecture for Learning, Planning, and Reacting, ACM Sigart 1991]

Optimal exploration for model learning



- How to explore optimally for learning the model?
- Explicit Explore or Exploit (E[3]):
  - Maintain set S of sufficiently explored states
  - The model $\hat{M}$ has the empirical transitions and rewards on S
  - Other states collapsed to absorbing state with reward 0 (in $\hat{M}$) or $r_{\max}$ (in $\hat{M}'$)
- Principle of optimism under uncertainty

[Kearns and Signh, Near-optimal reinforcement learning in polynomial time, ML 2002]

Explicit Explore or Exploit (E^3)

Algorithm E^3

$S \leftarrow \emptyset$
repeat
  $\pi \leftarrow$ optimal plan in $\hat{M}$ ← pessimistic model
  if $\Pr(\pi$ reaches absorbing state$) < \epsilon$ **then**
    Terminate
  else
    Execute optimal plan in $\hat{M}'$ ← optimistic model
    if $s \notin S$ reached **then**
      Take least tried action
      if each action tried K times **then**
        Empirically estimate $\hat{p}(\cdot|s,\cdot), \hat{r}(s,\cdot)$
        Add s to S

- When probability to explore is low, optimal policy in $\hat{M}$ is truly near-optimal
- For provable guarantees, ϵ and K can be determined from real number of states
  - Or updated every time the number of visited states is doubled

R-max

- E^3 takes different actions when it explores or exploits
  - $\Rightarrow$ needs to know which at start of episode, many steps ahead
- Instead, plan only in optimistic $\hat{M}'$
  - Implicit explore or exploit: either

Algorithm R-MAX

mark all states *unknown*
repeat
  Execute $\pi \leftarrow$ optimal plan in $\hat{M}'$
  Record (s, a, r, s') in *unknown* states
  if $n(s) = K$ **then**
    Empirically estimate $\hat{p}(\cdot|s, \cdot)$, $\hat{r}(s, \cdot)$
    Mark s *known*

[Brafman and Tennenholtz, R-max – A General Polynomial Time Algorithm for Near-Optimal Reinforcement Learning, JMLR 2002]

Today's lecture

Model-based learning

Model-free RL with a model

Model-predictive control

Modeling POMDPs

Issues with approximate models (1)

- In large state / action spaces, we can only approximate the dynamics
- No guarantees outside of training distribution
  - We shouldn't step too far off-policy
- Solution: keep interacting using learner policy and updating the model

Issues with approximate models (2)

- Model inaccuracy accumulates
  - If $|p_\phi(s'|s,a) - p(s'|s,a)|_1 \leq \epsilon$ then $|p_\phi(s_t) - p(s_t)|_1 \leq \epsilon t$
  - We have to plan far enough ahead to realize the consequences of actions
  - But we don't have to execute those plans far ahead!

Algorithm Model-Predictive Control (MPC)

$\mathcal{D} \leftarrow$ collect data
repeat
  $\hat{M} \leftarrow$ train model $\hat{p}, \hat{r}$ from $\mathcal{D}$
  repeat
    $\pi \leftarrow$ plan in $\hat{M}$ from current state s to horizon H
    Take *one action* a according to π
    Add empirical (s, a, r, s') to $\mathcal{D}$

How to use a differentiable model

- Suppose we have differentiable $x_{t+1} = f(x_t, u_t)$ and $c(x_t, u_t)$

- Taylor expansion for ϵ-perturbation $(\delta x, \delta u)$ around a trajectory $(\hat{x}, \hat{u})$:

interesting dependence on x_t and u_t

$$\hat{f}(x_t, u_t) = \hat{f}(\hat{x}_t, \hat{u}_t) + O(\epsilon)$$

- Fast
- Resettable
- Differentiable

How to use a differentiable model

- Suppose we have differentiable $x_{t+1} = f(x_t, u_t)$ and $c(x_t, u_t)$
- Taylor expansion for ϵ-perturbation $(\delta x, \delta u)$ around a trajectory $(\hat{x}, \hat{u})$:

captures linear dependence on x_t and u_t

$$\hat{f}(x_t, u_t) = \hat{f}(\hat{x}_t, \hat{u}_t) + \delta x_t \nabla_x \hat{f}_t + \delta u_t \nabla_u \hat{f}_t + O(\epsilon^2)$$

- Fast
- Resettable
- Differentiable

How to use a differentiable model

- Suppose we have differentiable $x_{t+1} = f(x_t, u_t)$ and $c(x_t, u_t)$

- Taylor expansion for ϵ-perturbation $(\delta x, \delta u)$ around a trajectory $(\hat{x}, \hat{u})$:

$$\hat{f}(x_t, u_t) = \hat{f}(\hat{x}_t, \hat{u}_t) + \delta x_t \nabla_x \hat{f}_t + \delta u_t \nabla_u \hat{f}_t + O(\epsilon^2)$$

$$\hat{c}(x_t, u_t) = \hat{c}(\hat{x}_t, \hat{u}_t) + O(\epsilon)$$

interesting dependence on x_t and u_t

How to use a differentiable model

- Suppose we have differentiable $x_{t+1} = f(x_t, u_t)$ and $c(x_t, u_t)$
- Taylor expansion for ϵ-perturbation $(\delta x, \delta u)$ around a trajectory $(\hat{x}, \hat{u})$:

$$\hat{f}(x_t, u_t) = \hat{f}(\hat{x}_t, \hat{u}_t) + \delta x_t \nabla_x \hat{f}_t + \delta u_t \nabla_u \hat{f}_t + O(\epsilon^2)$$
$$\hat{c}(x_t, u_t) = \hat{c}(\hat{x}_t, \hat{u}_t) + \delta x_t \nabla_x \hat{c}_t + \delta u_t \nabla_u \hat{c}_t + O(\epsilon^2)$$

linear dependence on x_t and u_t

optimal control: ∞

How to use a differentiable model

- Suppose we have differentiable $x_{t+1} = f(x_t, u_t)$ and $c(x_t, u_t)$
- Taylor expansion for ϵ-perturbation $(\delta x, \delta u)$ around a trajectory $(\hat{x}, \hat{u})$:

NOW we can neglect these

$$\hat{f}(x_t, u_t) = \hat{f}(\hat{x}_t, \hat{u}_t) + \delta x_t \nabla_x \hat{f}_t + \delta u_t \nabla_u \hat{f}_t + O(\epsilon^2)$$

$$\hat{c}(x_t, u_t) = \hat{c}(\hat{x}_t, \hat{u}_t) + \delta x_t \nabla_x \hat{c}_t + \delta u_t \nabla_u \hat{c}_t$$

$$+\frac{1}{2}(\delta x_t^\intercal (\nabla_x^2 \hat{c}_t)\delta x_t + \delta u_t^\intercal (\nabla_u^2 \hat{c}_t)\delta u_t + 2\delta x_t^\intercal (\nabla_{xu} \hat{c}_t)\delta u_t) + O(\epsilon^3)$$

Iterative LQR (iLQR)

Algorithm iLQR

Initialize $\hat{x}, \hat{u}$

repeat

Set $A, B \leftarrow \nabla_x \hat{f}_t, \nabla_u \hat{f}_t$ — linearize dynamics around current trajectory $(\hat{x}, \hat{u})$

Set $Q, R, N, q, r \leftarrow \nabla_x^2 \hat{c}_t, \nabla_u^2 \hat{c}_t, \nabla_{xu} \hat{c}_t, \nabla_x \hat{c}_t, \nabla_u \hat{c}_t$ — quadratic cost approximation around $(\hat{x}, \hat{u})$

$\hat{L}_t, \hat{\ell}_t \leftarrow$ LQR on $\delta x_t = x_t - \hat{x}_t$, $\delta u_t = u_t - \hat{u}_t$ — place "origin" at $(\hat{x}, \hat{u})$

$\delta\hat{x}, \delta\hat{u} \leftarrow$ execute policy $\delta u_t = \hat{L}_t \delta x_t + \hat{\ell}_t$ in env — roll out to get new trajectory $(\hat{x}, \hat{u})$

$\hat{x} \leftarrow \hat{x} + \delta\hat{x}$, $\hat{u} \leftarrow \hat{u} + \delta\hat{u}$

- Fast
- Resettable
- Differentiable

Local models

- Can we use a learned model for iLQR?
  - Idea 1: learn global model, linearize locally ⇒ wasteful
  - Idea 2: directly learn local linearizations:

Algorithm Local Models

Initialize a policy $\pi(u_t|x_t)$
repeat
  Roll out π to horizon T for N trajectories
  Fit $p(x_{t+1}|x_t, u_t)$
  Plan new policy π

- Fast
- Resettable
- Differentiable

[Kumar et al., Optimal Control with Learned Local Models: Application to Dexterous Manipulation, ICRA 2016]

How to fit local dynamics

- Idea 1: linear regression
  - Find $(A_t, B_t)_{t=0}^{T-1}$ such that $x_{t+1} \approx A_t x_t + B_t u_t$
  - Do we care about the process noise ω_t?
    - If we assume it's Gaussian, doesn't affect policy; but could help evaluate the method
- Idea 2: Bayesian linear regression
  - Learn global model, use it as prior for local model
  - More data efficient across time steps and across iterations

How to plan with local models

- Idea 1: as in iLQR, find optimal control sequence $\hat{u}$ and its trajectory $\hat{x}$
  - Problem: model errors will cause actual trajectory to diverge from $\hat{x}$
- Idea 2: find $\hat{x}$ by executing the optimal policy directly in the environment
  - Problem: need spread for linear regression, dynamics may be too deterministic
- Idea 3: make control stochastic by injecting Gaussian noise
  - E.g., have $\epsilon_t \sim \mathcal{N}(0, R^{-1})$, shaped by the control cost
    - Optimal for the incurred costs, not for the spread needed for regression

Today's lecture

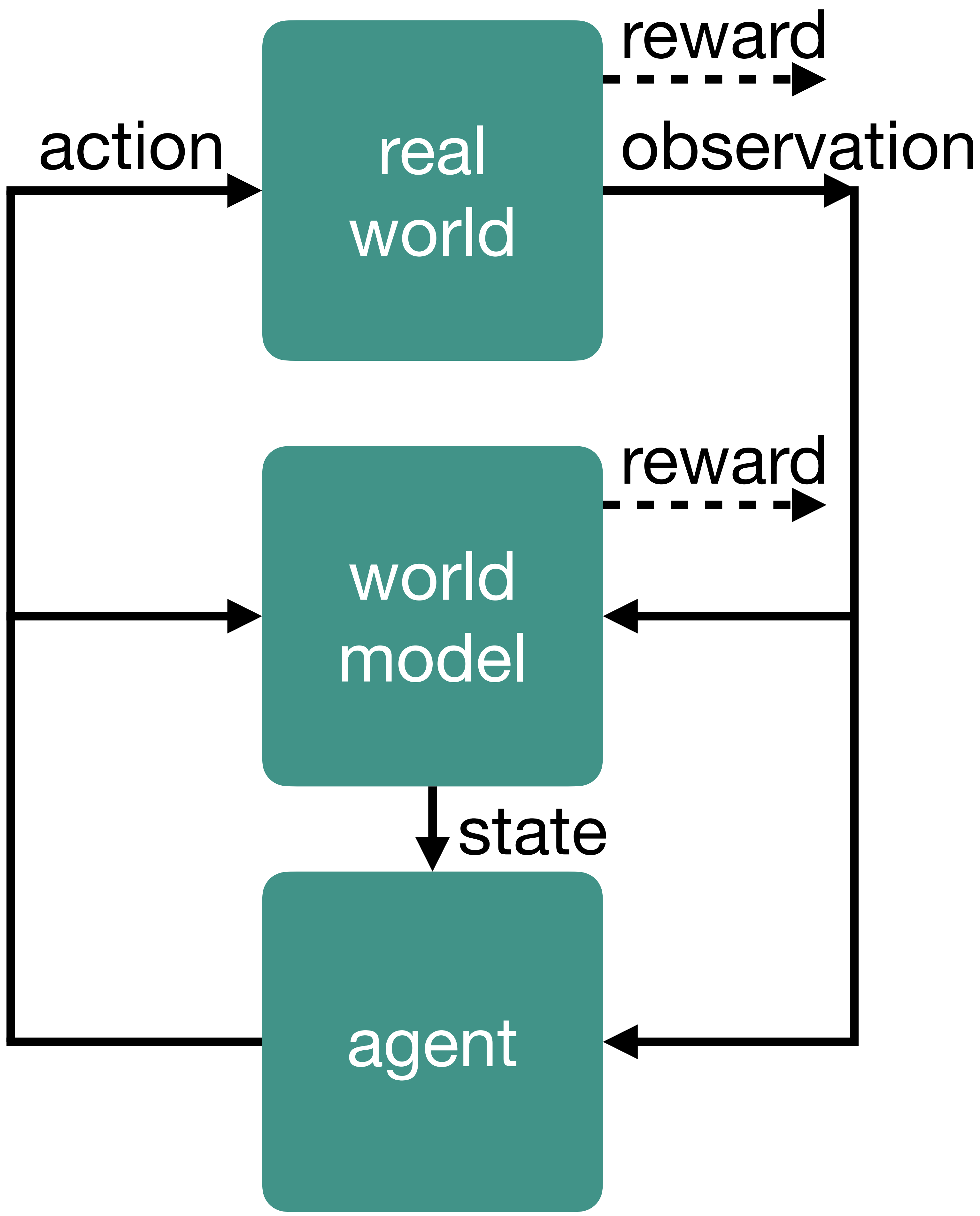
Model-based learning

Model-free RL with a model

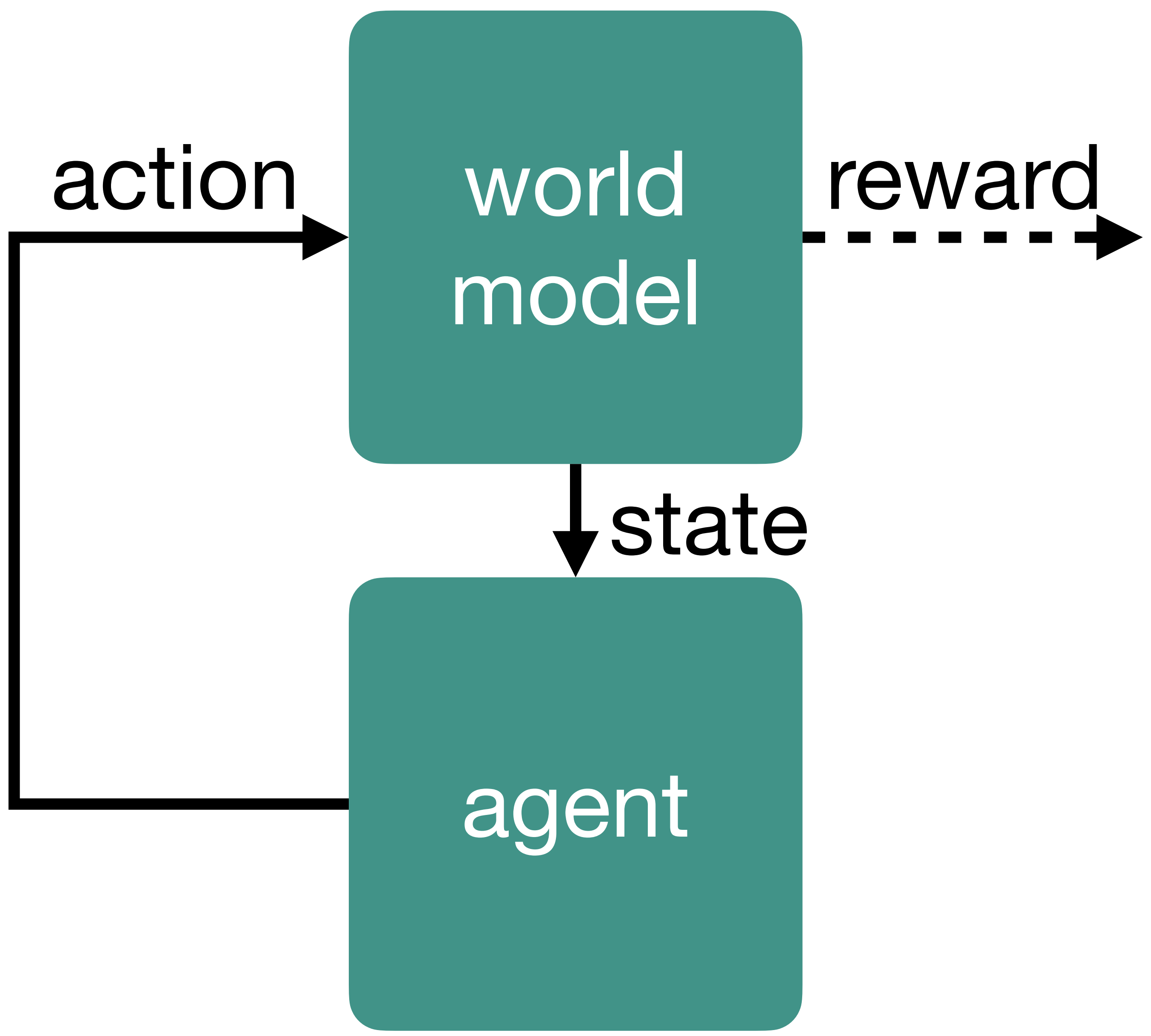
Model-predictive control

Modeling POMDPs

World modeling under partial observability



Imagination of counterfactual actions



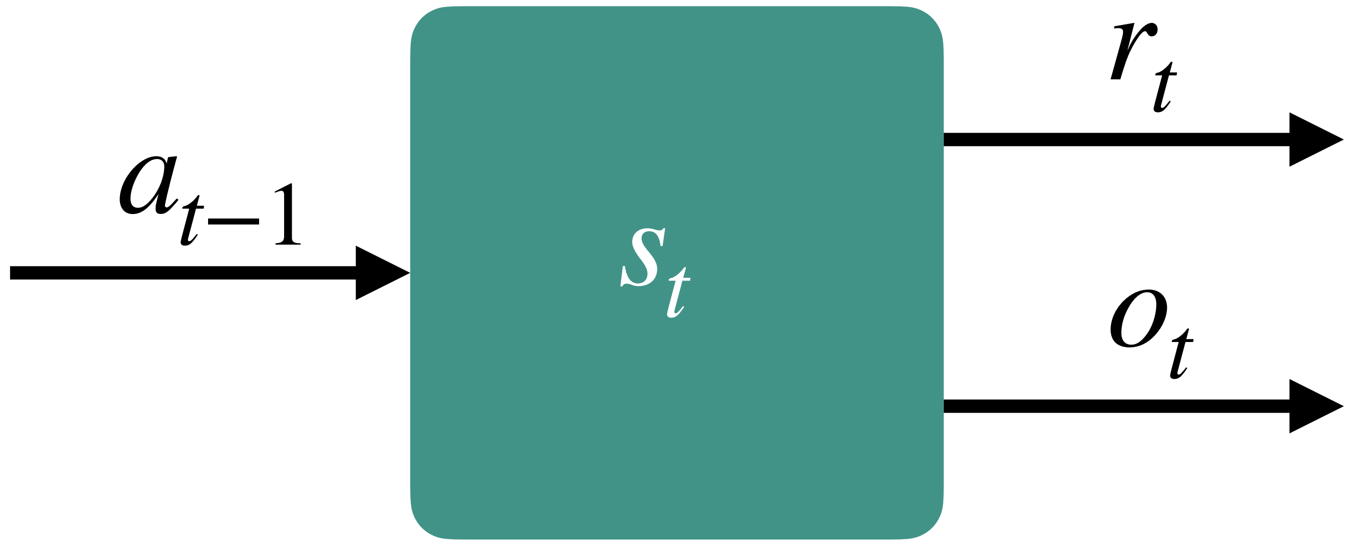
- Same reward distr. after every history
  - Same return for each policy ← **bisimulation**
  - The modes must be aligned

Why model the world? As agent memory

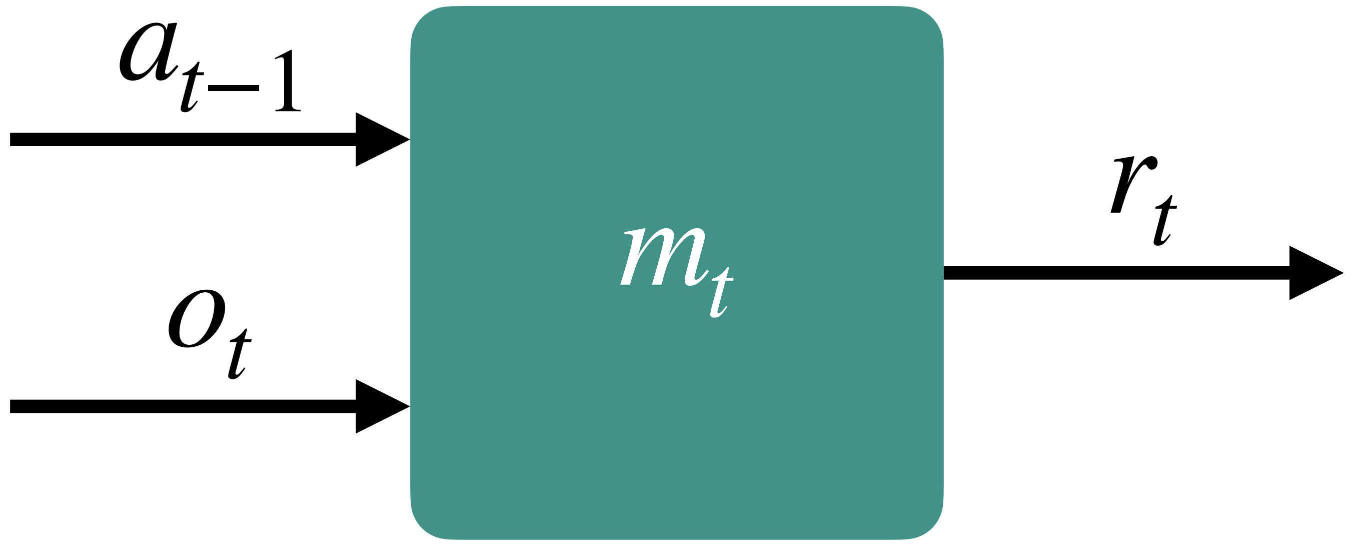
- The model is an MDP, even if the world is a POMDP
  - It can be used as an RNN to process the observable history
  - The latent state is fully observable to the agent, which simplifies RL
  - Supervising the world modeling is much easier than RL of RNN-based policies
- Many MDPs are equivalent to a given POMDP
  - Most known is the Belief MDP, which is sufficient but generally not minimal
  - World modeling finds a different equivalent MDP that may not represent beliefs
    - We never see the state, and we may not even care about the full state

What is a POMDP world model?

- In RL: how my past actions affect my future rewards
  - … given my past observations

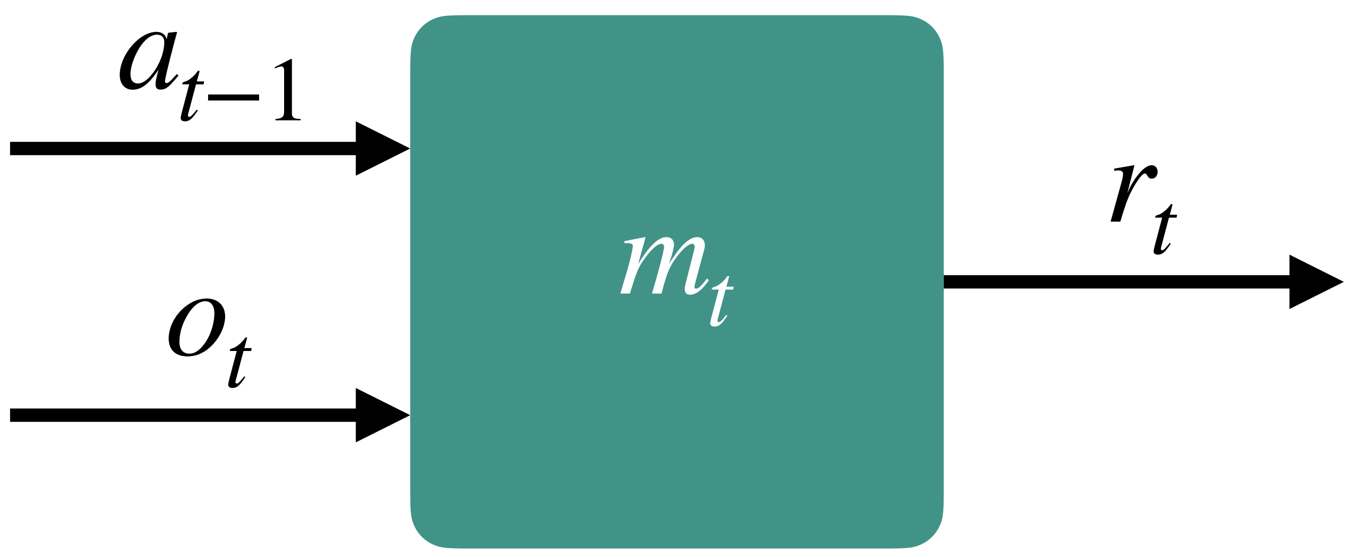


- A bisimulation lemma: if $p(r_t | h_t) = q(r_t | h_t)$ for any history $h_t = (o_{\leq t}, a_{<t})$
  - then $\mathbb{E}_{\xi \sim p_\pi}[R(\xi)] = \mathbb{E}_{\xi \sim q_\pi}[R(\xi)]$ for any policy $\pi(a_t | h_t)$ (with $R(\xi) = \sum_t \gamma^t r_t$)
  - If q is a good model of p, then any good policy in q is also good in p
  - But how do we find a good policy in q?
    - Model-free RL!

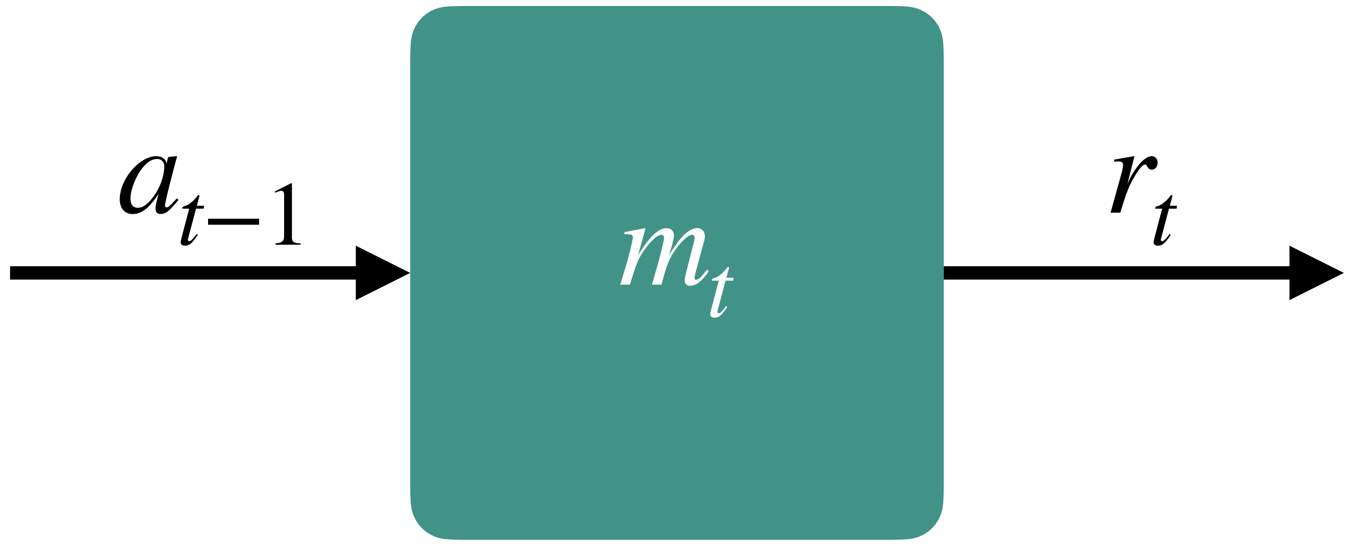


How to use a POMDP world model?

- Interaction mode: feed environment observations, base action on model state
  - Model step: $q(m_t \mid m_{t-1}, a_{t-1}, o_t)$, can be deterministic
  - $q(m_t \mid h_t)$ embeds the history (like an RNN or Transformer)

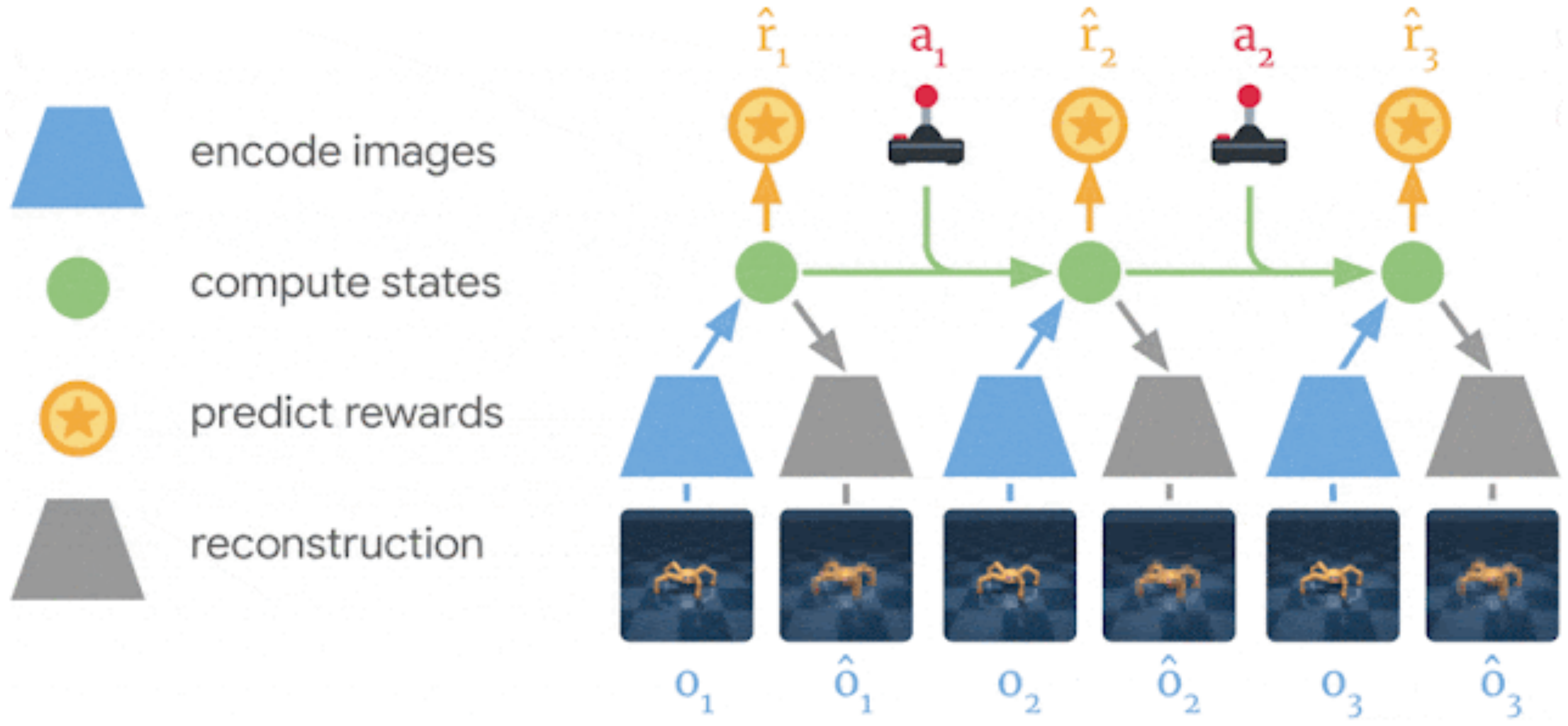


- Imagination mode: no observation needed, can be used as simulator
  - Model step: $q(m_t \mid m_{t-1}, a_{t-1})$
  - $\pi(a_t \mid m_t)$ can then transfer to interaction mode



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]

Recap

- Model-based RL schemes:
  - Plan in a learned model
  - Improve model-free RL using a learned model
- Good theory for how to explore optimally for learning a model
- Potentially huge benefits under partial observability