AA203 Optimal and Learning-based Control

Lecture 18

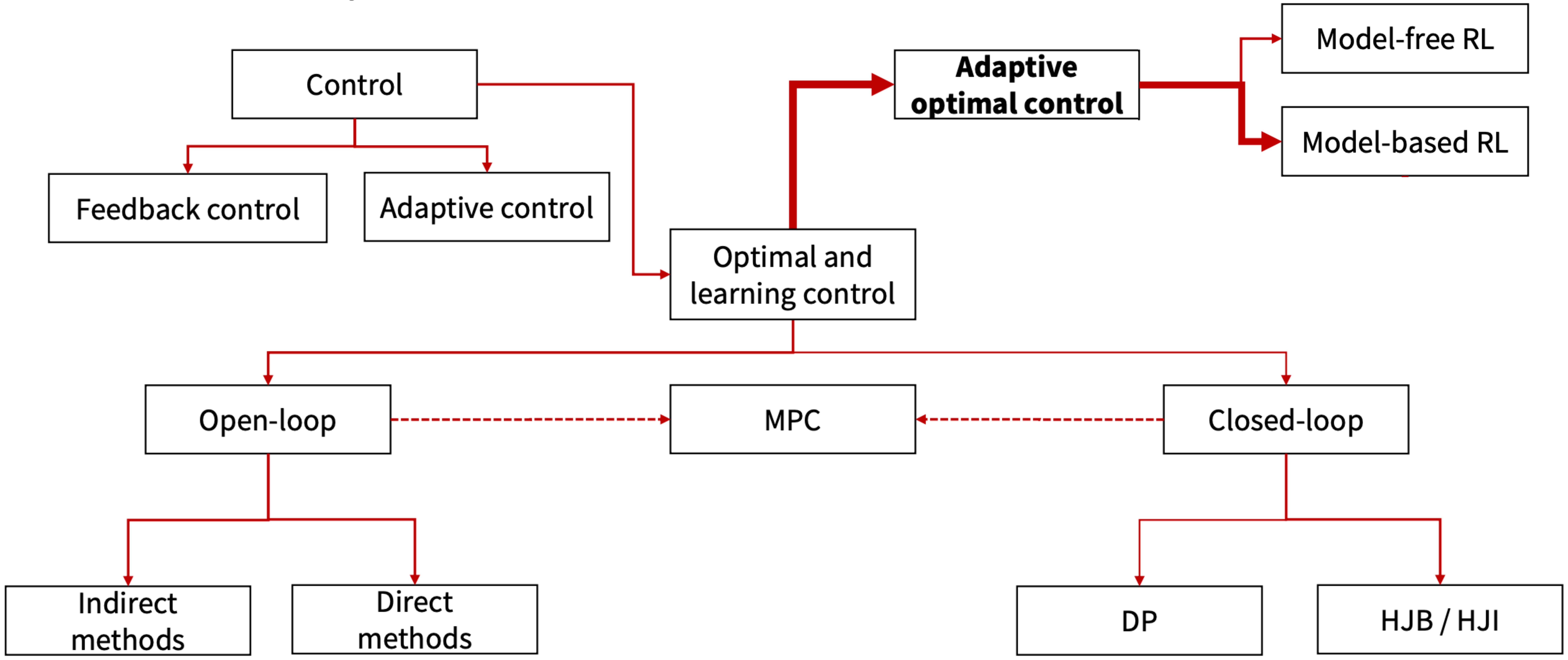
Model-based Policy Learning

Autonomous Systems Laboratory
Daniele Gammelli

Stanford University

Autonomous Systems Laboratory
Stanford Aeronautics & Astronautics

Roadmap


Control
Feedback control
Adaptive control
Optimal and learning control
Adaptive optimal control
Model-free RL
Model-based RL
Open-loop
MPC
Closed-loop
Indirect methods
Direct methods
DP
HJB / HJI

Recap: Model-based RL

- In model-free RL, we discussed different approaches to solve unknown MDPs directly from experience via **policy / value-function learning**
- In model-based RL, we aim to (1) **estimate an approximate model** of the dynamics, and (2) **use it for control**

Approach 1: "learn a model $p(x_{t+1} | x_t, u_t)$ from experience and use it to plan"

1. Run base policy $\pi_0(u_t | x_t)$ in the environment (e.g., random policy, exploration policy) and collect dataset of transitions $\mathcal{D} = \{(x_t, u_t, x_{t+1})_i\}$
2. Fit dynamics model to data to minimize error (or equivalently, maximize (log) likelihood)

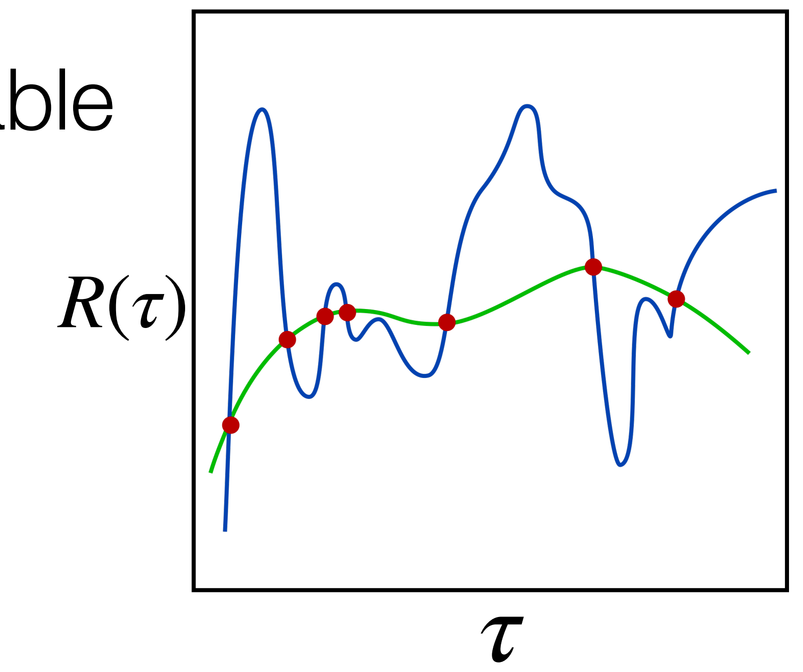
$$\theta^* = \arg\min_\theta \sum_i \left\| f_\theta(x_t, u_t) - x_{t+1} \right\|^2$$

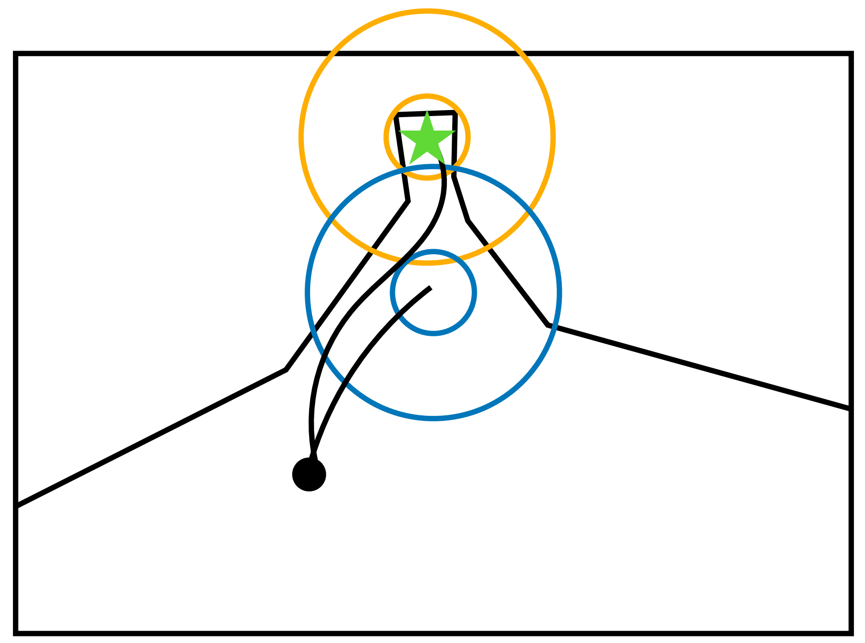
YES	NO
Sys. ID	Distribution mismatch Exploitation of errors

3. Use the learned model to plan a sequence of actions

Problem: we'll likely *erroneously* exploit our model where it is less knowledgeable
(Possible) Solution: consider how "certain" we are our about the prediction

This allows us to reason in terms of expectations under our model



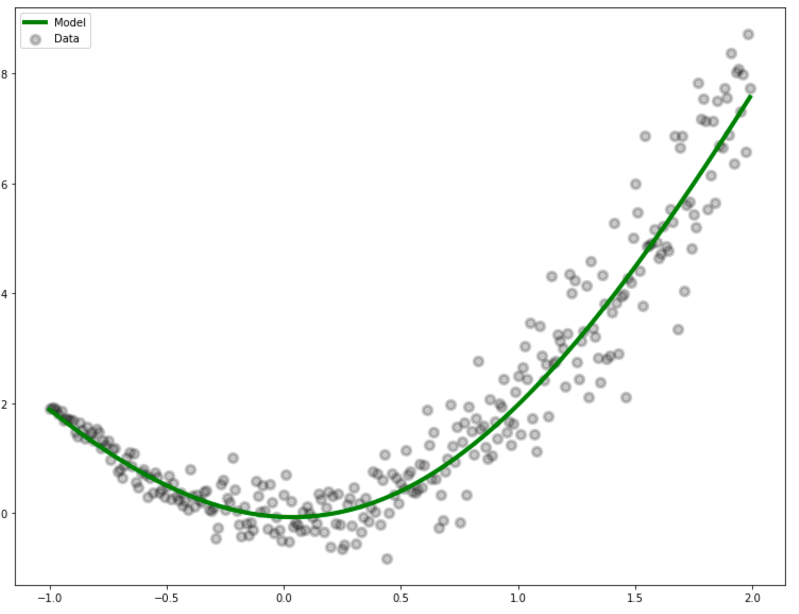


Recap: Model-based RL

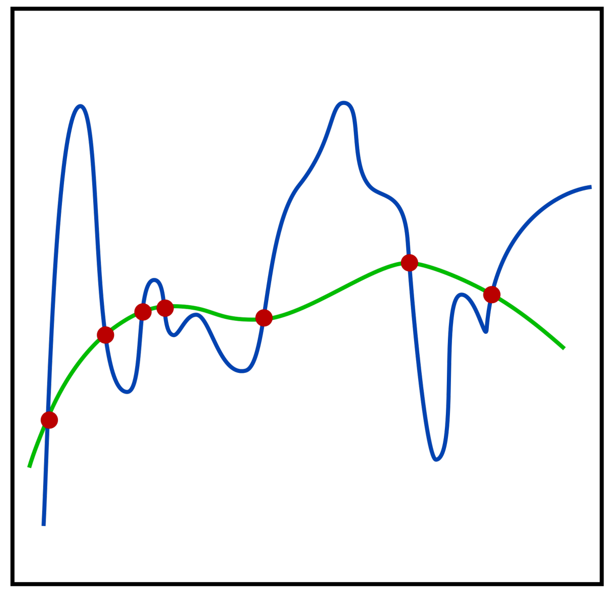
Uncertainty estimation

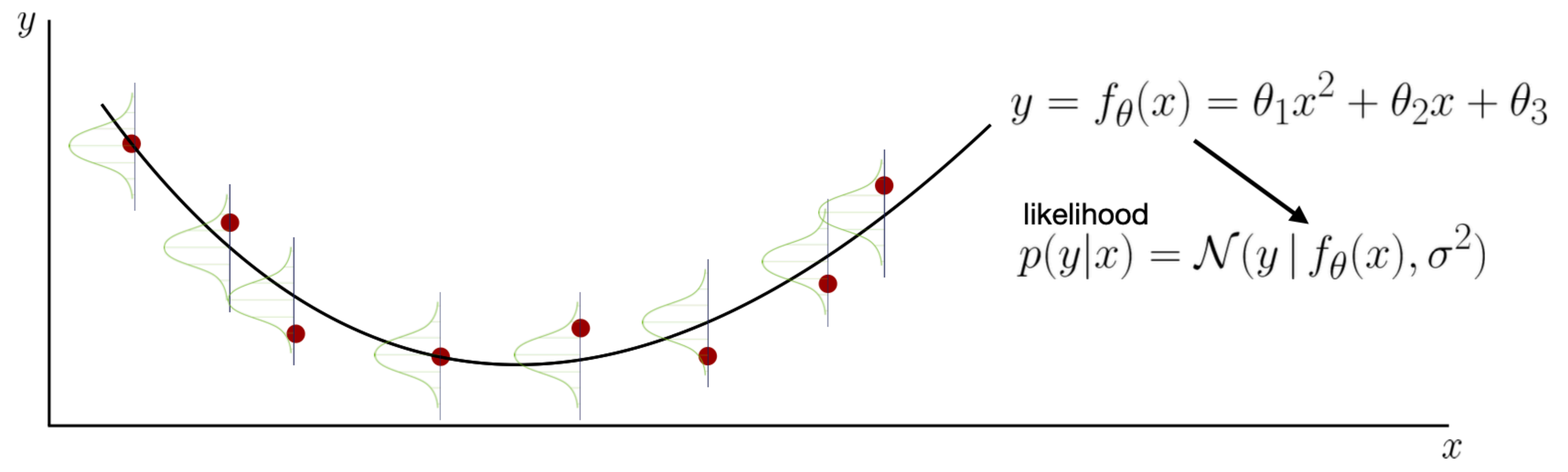
- Learning from a probabilistic standpoint (i.e., deterministic vs probabilistic predictions)
- The importance of estimating *model/epistemic uncertainty*

Aleatoric uncertainty



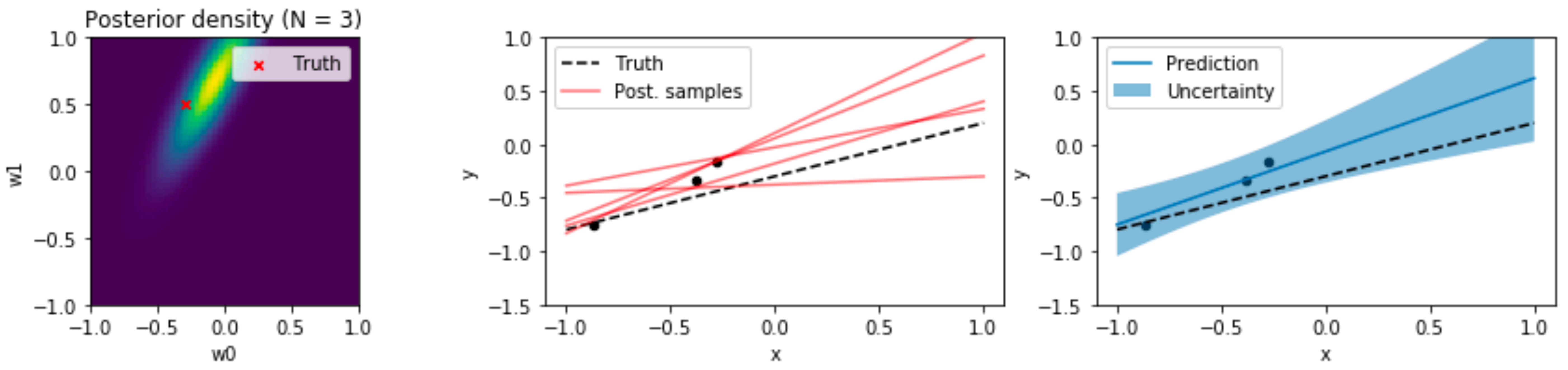
Epistemic uncertainty

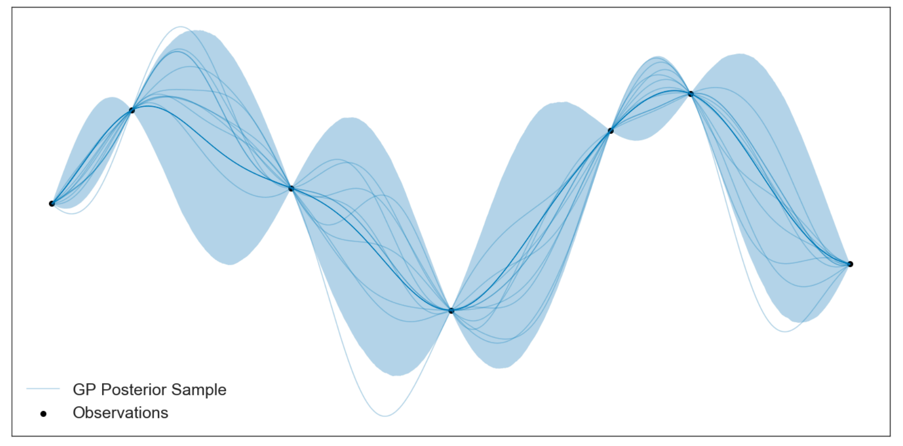


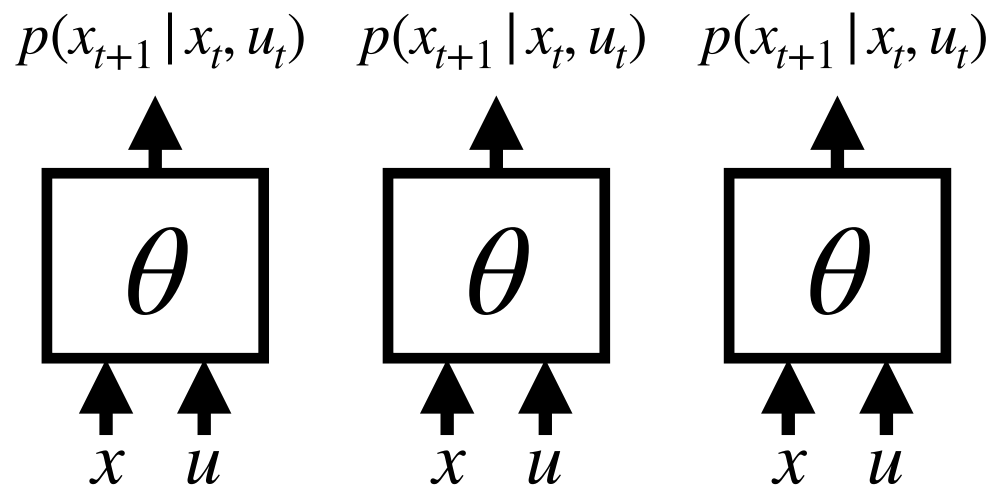


y
x
$y = f_\theta(x) = \theta_1 x^2 + \theta_2 x + \theta_3$
likelihood
$p(y|x) = \mathcal{N}(y \,|\, f_\theta(x), \sigma^2)$

- A structured way to represent uncertainty over a parametric model is through a **posterior distribution** over the parameters $p(\theta \,|\, \mathscr{D})$







$p(x_{t+1} \,|\, x_t, u_t)$
$p(x_{t+1} \,|\, x_t, u_t)$
$p(x_{t+1} \,|\, x_t, u_t)$
θ
θ
θ
x u
x u
x u

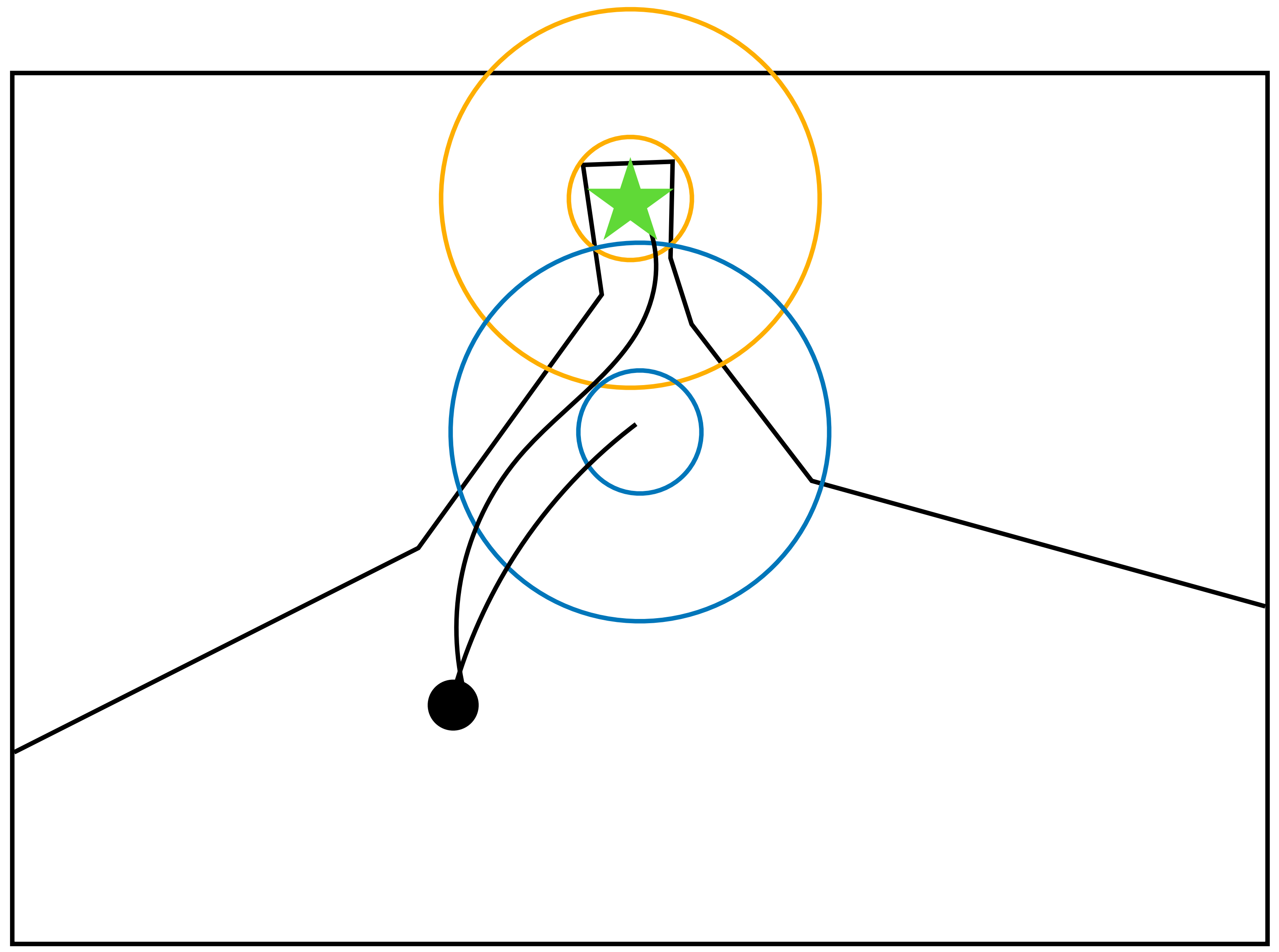
- Two examples:
  - Gaussian Processes (accurate; expensive; limited expressivity)
  - Ensembles (approximate; simple; high-capacity NNs)

Recap: Model-based RL

- How do we use this in planning? A possible idea is the following:

- Given a candidate action sequence $u_1, \dots, u_T$:
    1. Sample $\theta_i \sim p(\theta \mid \mathcal{D})$ (in the case of ensembles, this is equivalent to choosing one among the models)
    2. Propagate forward the learned dynamics according to $x_{t+1} \sim p_{\theta_i}(x_{t+1} \mid x_t, u_t)$, for all t
    3. Compute (predicted) rewards $\sum_t r(x_t, u_t)$
    4. Repeat steps 1-3 and compute the average reward

$$J\left(u_1, \dots, u_T\right) = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{H} r\left(x_{t,i}, u_t\right), \text{ where } x_{t+1,i} \sim p_{\theta_i}\left(x_{t+1,i} \mid x_{t,i}, u_t\right)$$

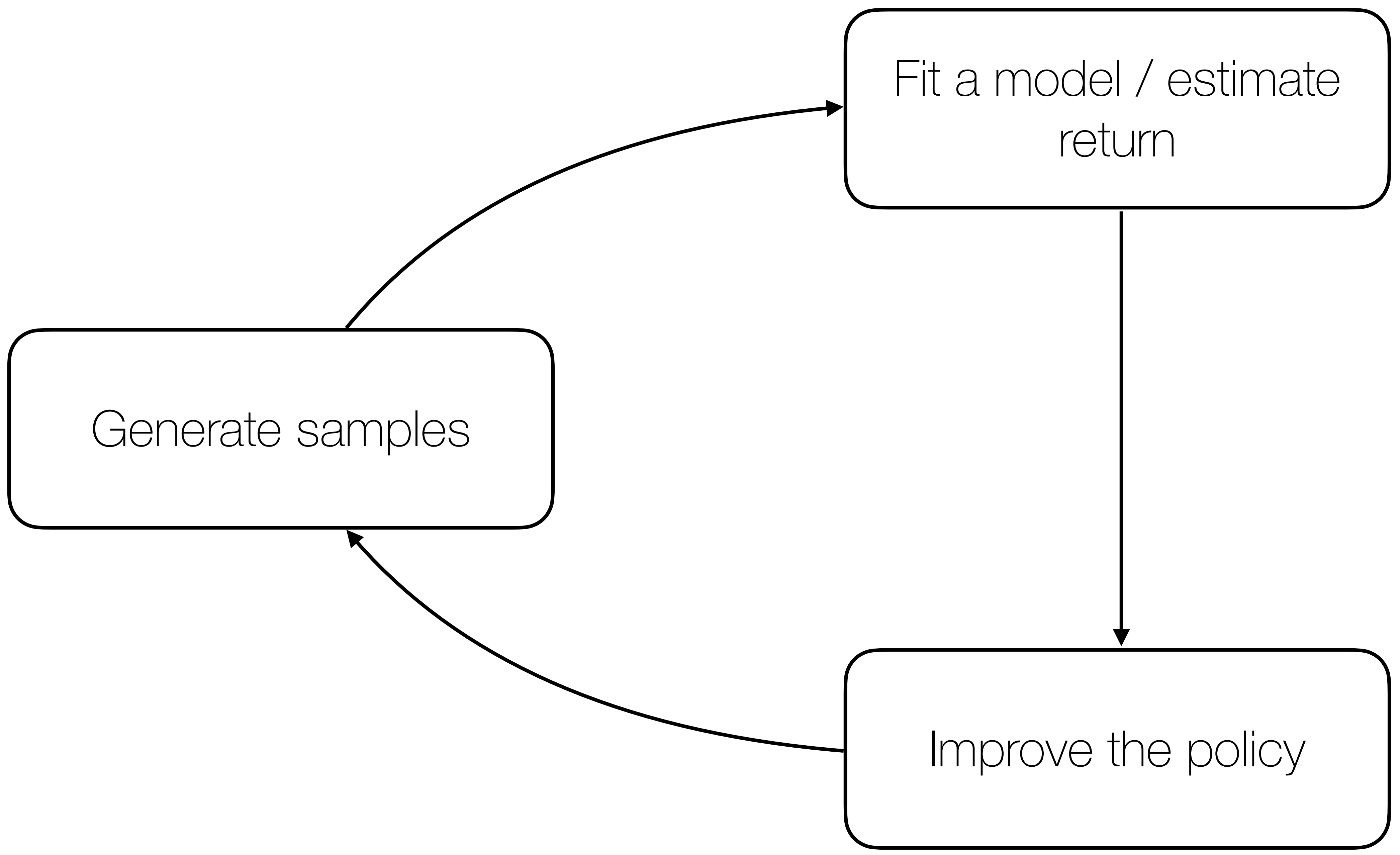


Recap: Model-based RL

$\pi(u_t \mid x_t)$

AGENT

$\tau = (x_0, u_0, \ldots, x_N, u_N)$

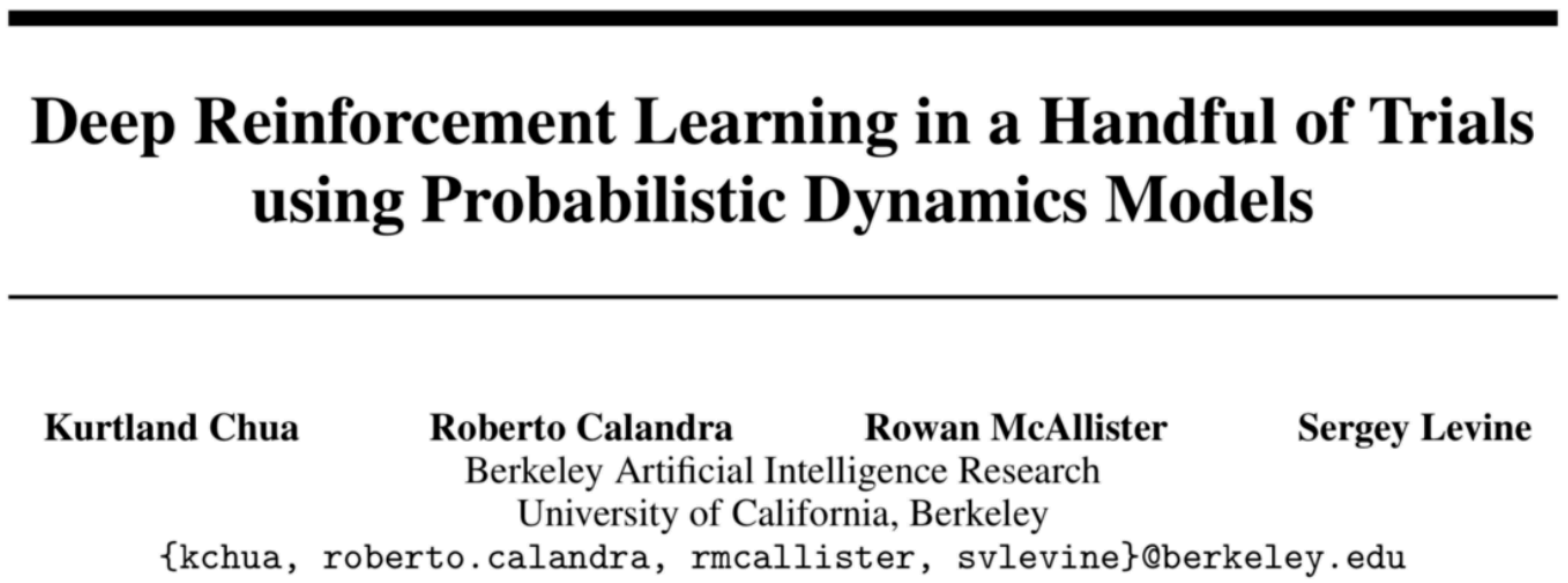


$f_\theta\left(x_t, u_t\right) \approx P\left(x_{t+1} \mid x_t, u_t\right)$

plan through f_θ

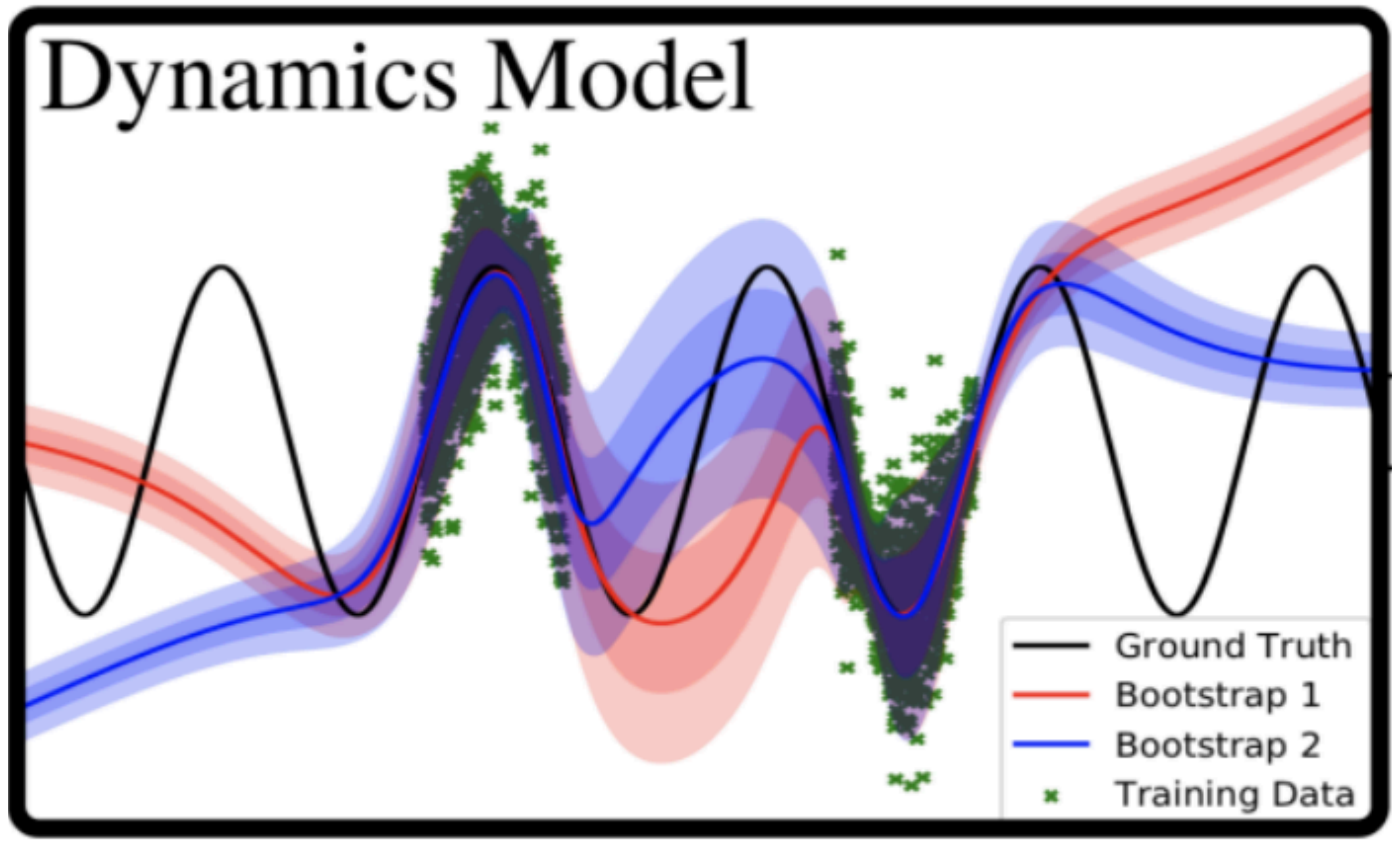
Case study: PETS

- Probabilistic Ensembles with Trajectory Sampling
- Key idea:
  - **Model:** Use ensemble of NNs to approximate posterior over model



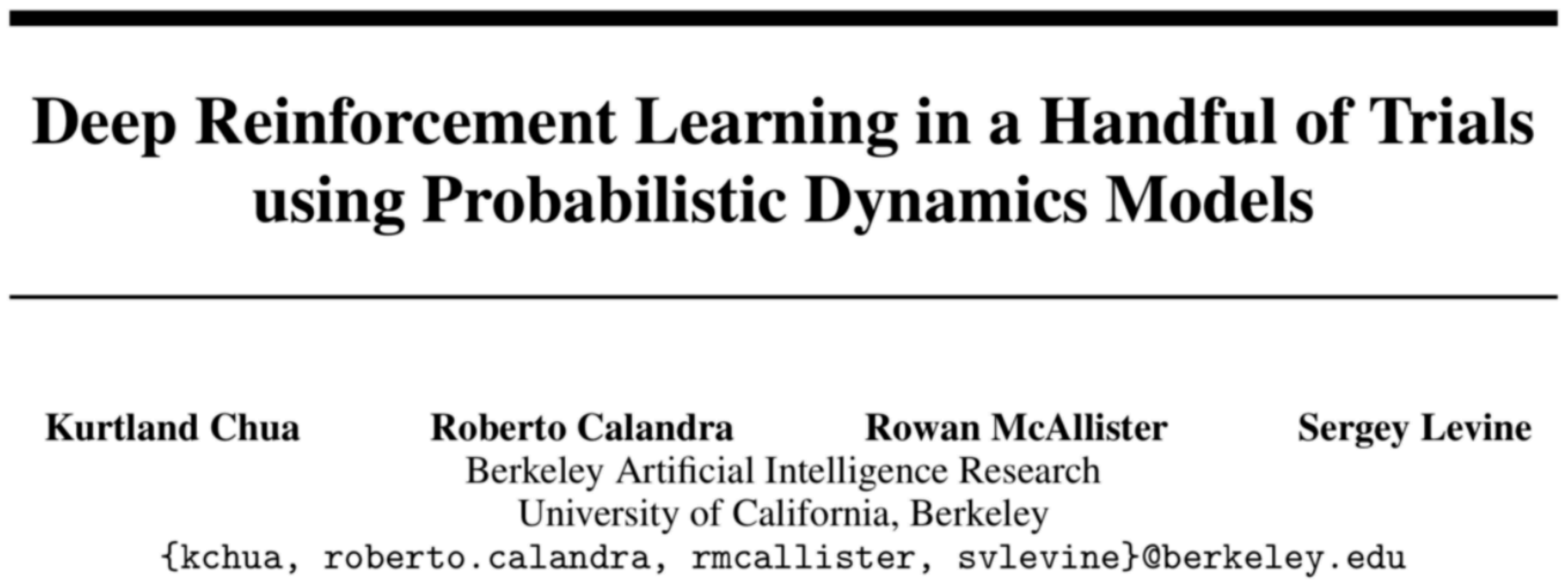
Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models

Kurtland Chua **Roberto Calandra** **Rowan McAllister** **Sergey Levine**
Berkeley Artificial Intelligence Research
University of California, Berkeley
{kchua, roberto.calandra, rmcallister, svlevine}@berkeley.edu



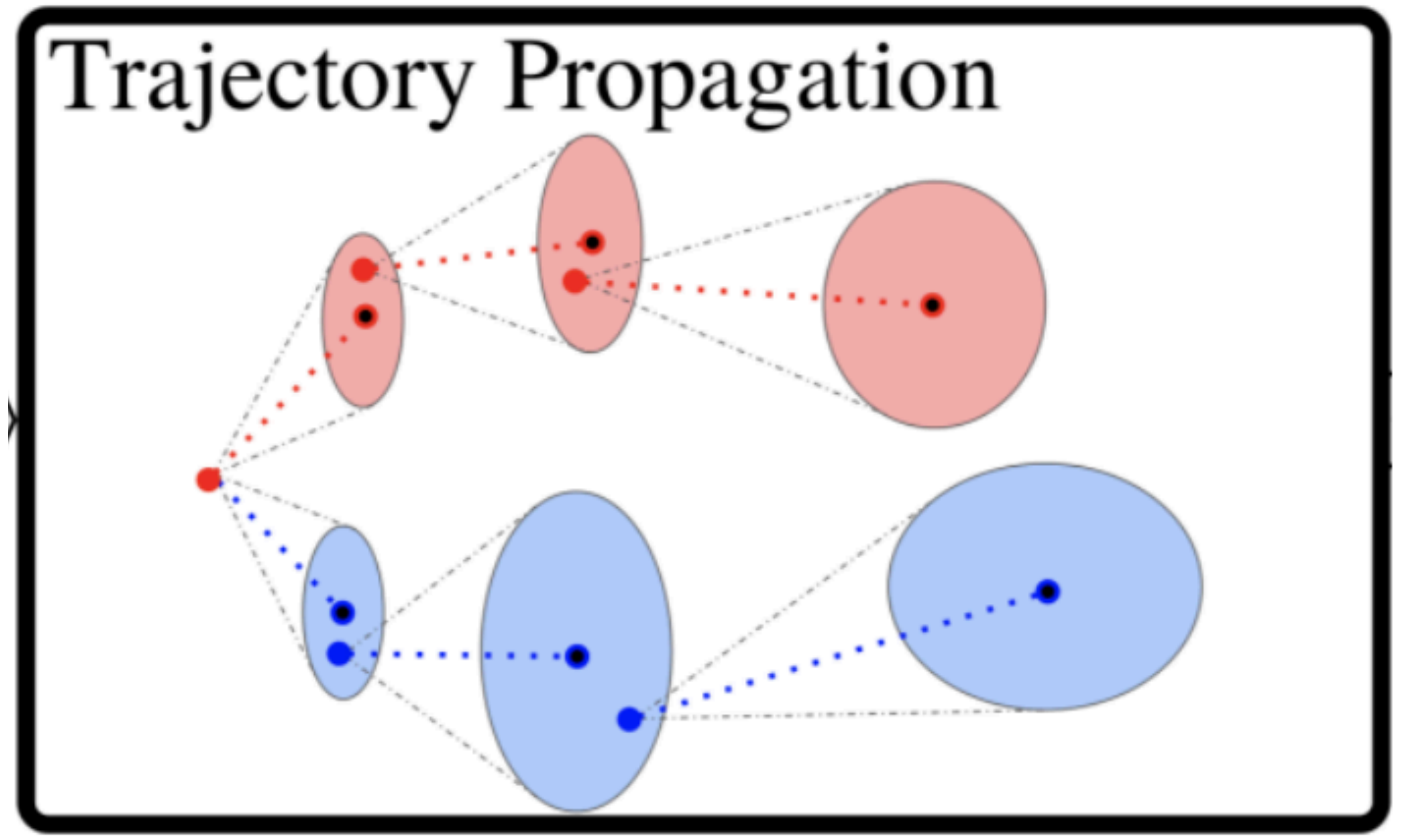
Case study: PETS

- Probabilistic Ensembles with Trajectory Sampling
- Key idea:
  - **Model:** Use ensemble of NNs to approximate posterior over model
  - **Propagation:** sample different models and use them to generate predictions of different "futures"



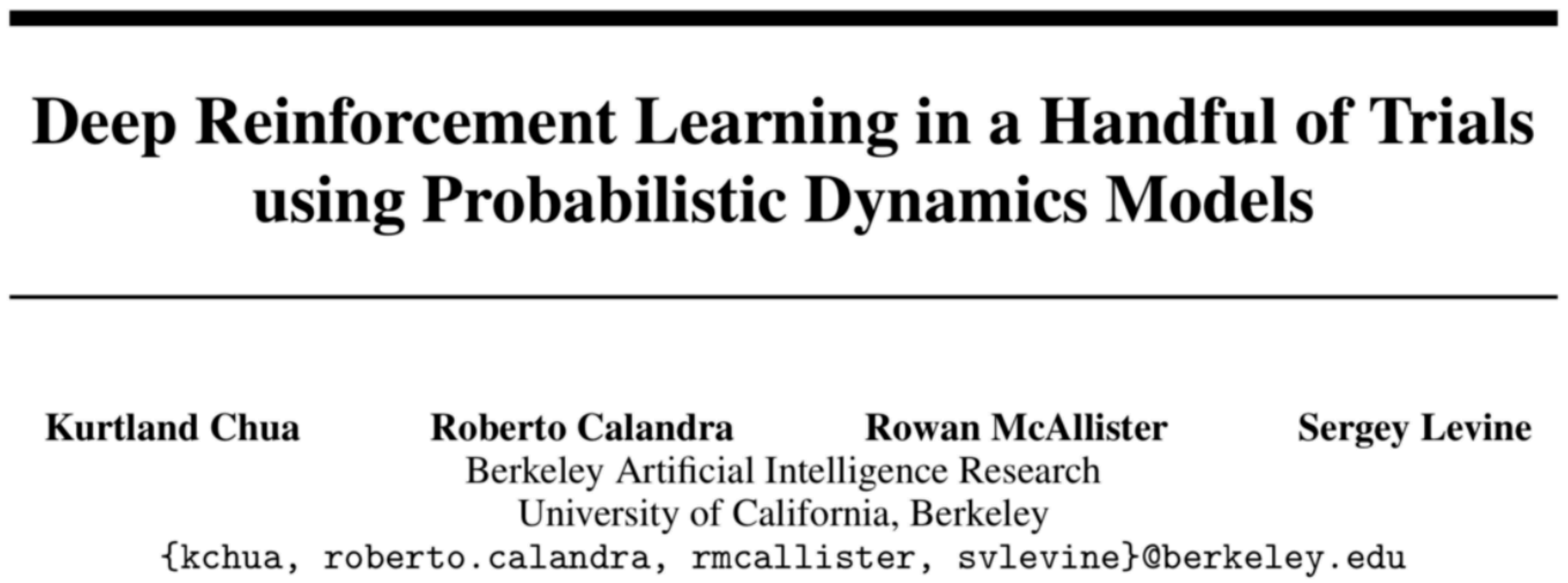
Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models

Kurtland Chua **Roberto Calandra** **Rowan McAllister** **Sergey Levine**
Berkeley Artificial Intelligence Research
University of California, Berkeley
{kchua, roberto.calandra, rmcallister, svlevine}@berkeley.edu



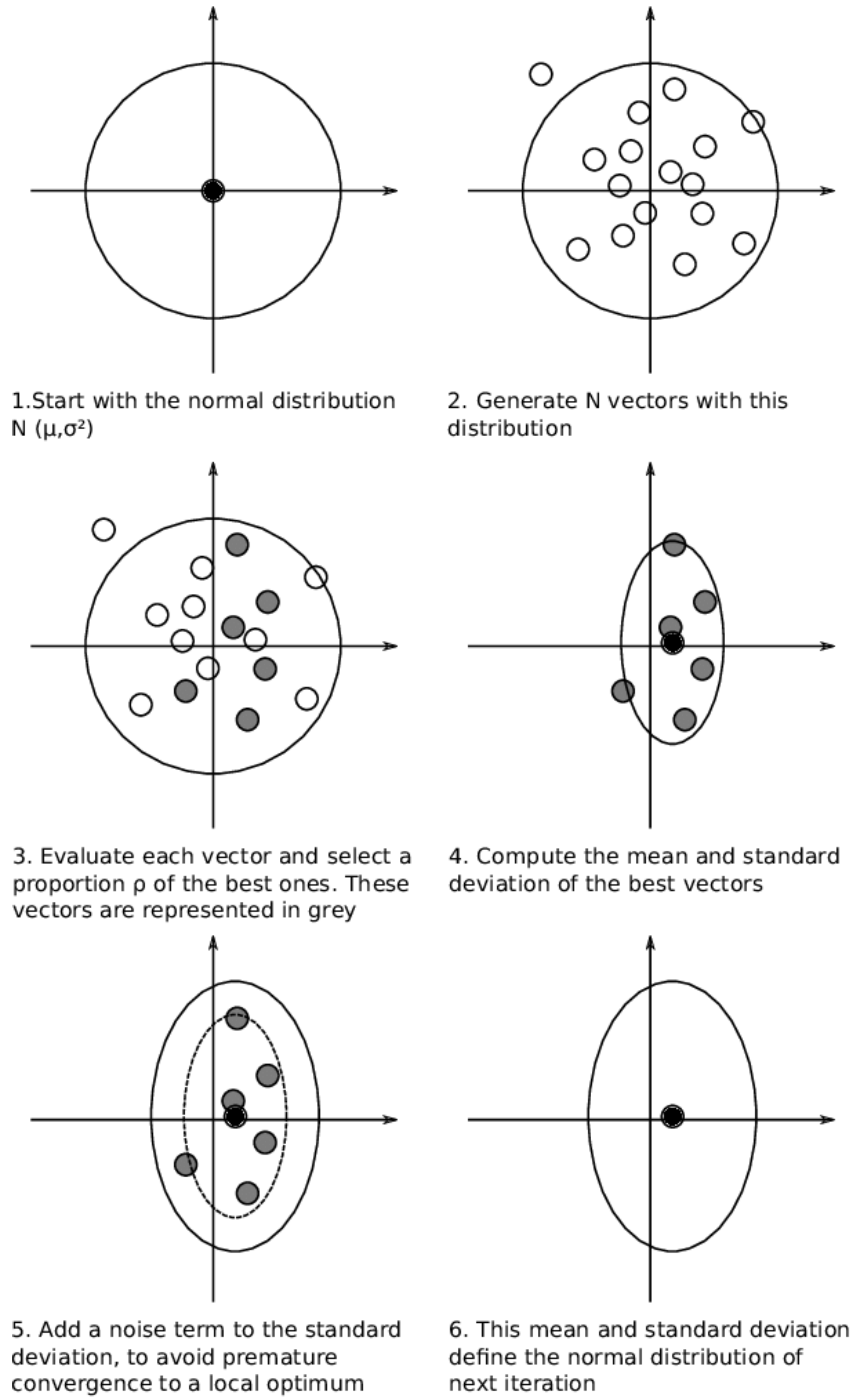
Case study: PETS

- Probabilistic Ensembles with Trajectory Sampling
- Key idea:
    - **Model:** Use ensemble of NNs to approximate posterior over model
    - **Propagation:** sample different models and use them to generate predictions of different “futures”
    - **Planning:** apply MPC (compute action sequence via sampling, i.e., cross-entropy method (CEM))



Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models

Kurtland Chua **Roberto Calandra** **Rowan McAllister** **Sergey Levine**
Berkeley Artificial Intelligence Research
University of California, Berkeley
{kchua, roberto.calandra, rmcallister, svlevine}@berkeley.edu



1.Start with the normal distribution N (μ,σ²)

2. Generate N vectors with this distribution

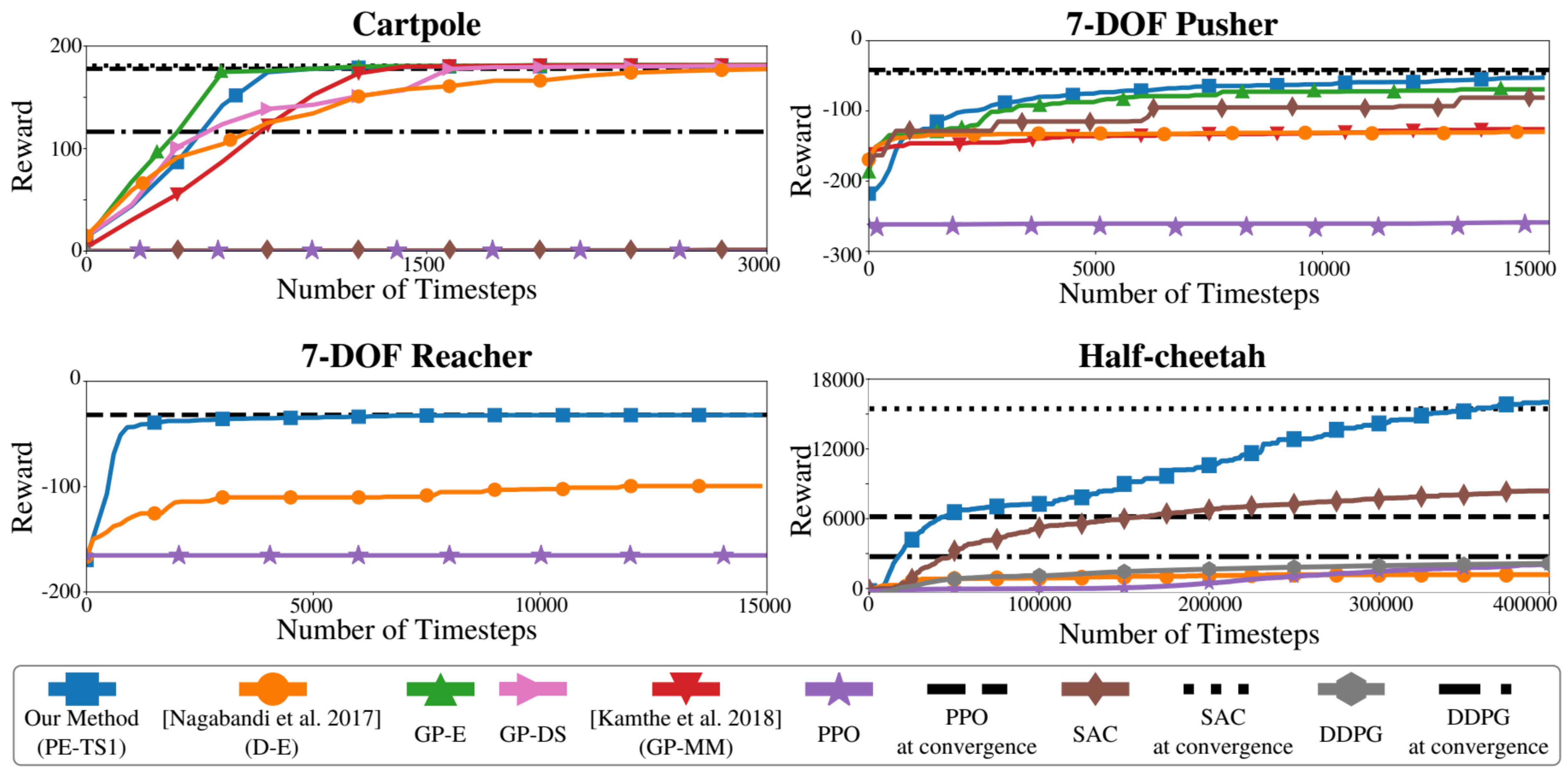
3. Evaluate each vector and select a proportion ρ of the best ones. These vectors are represented in grey

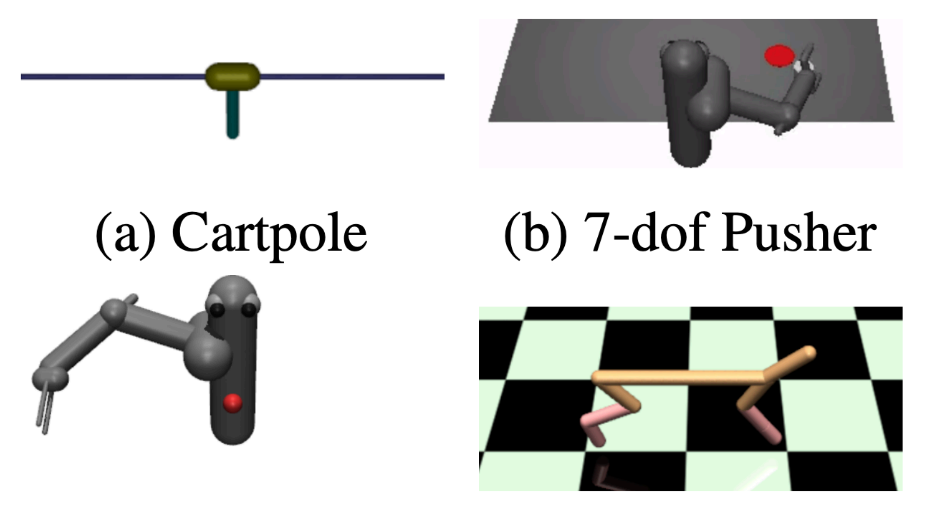
4. Compute the mean and standard deviation of the best vectors

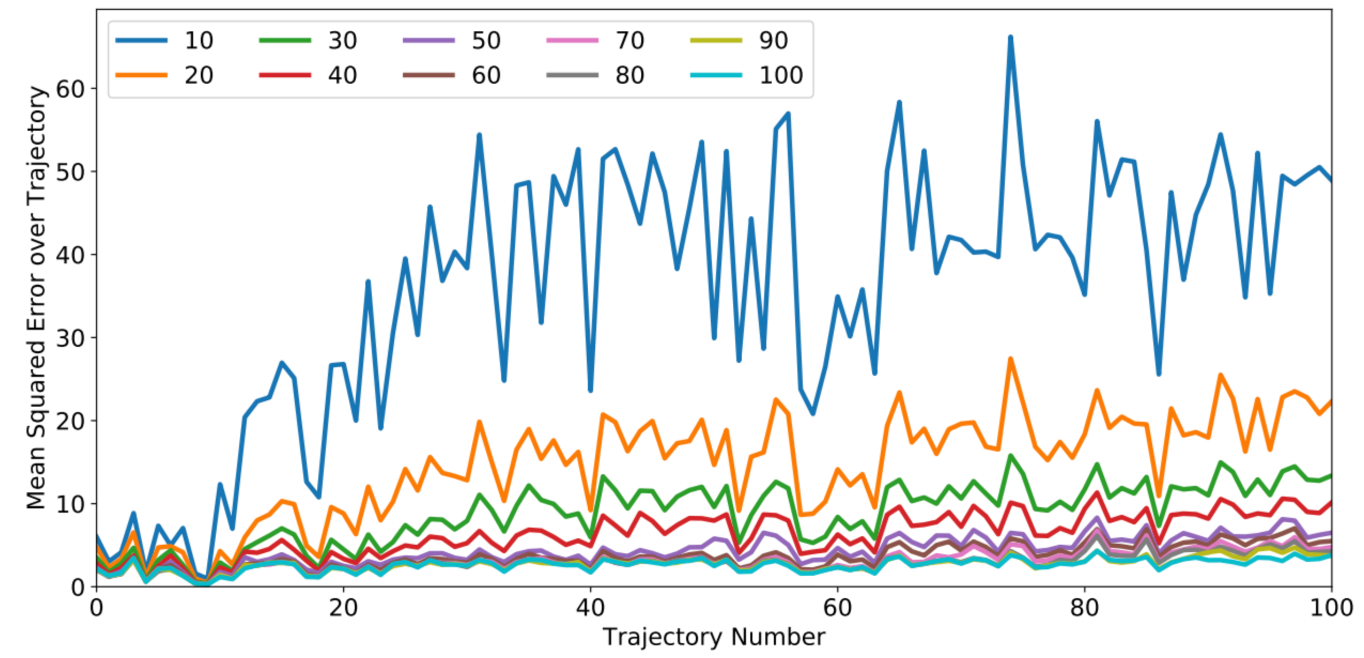
5. Add a noise term to the standard deviation, to avoid premature convergence to a local optimum

6. This mean and standard deviation define the normal distribution of next iteration

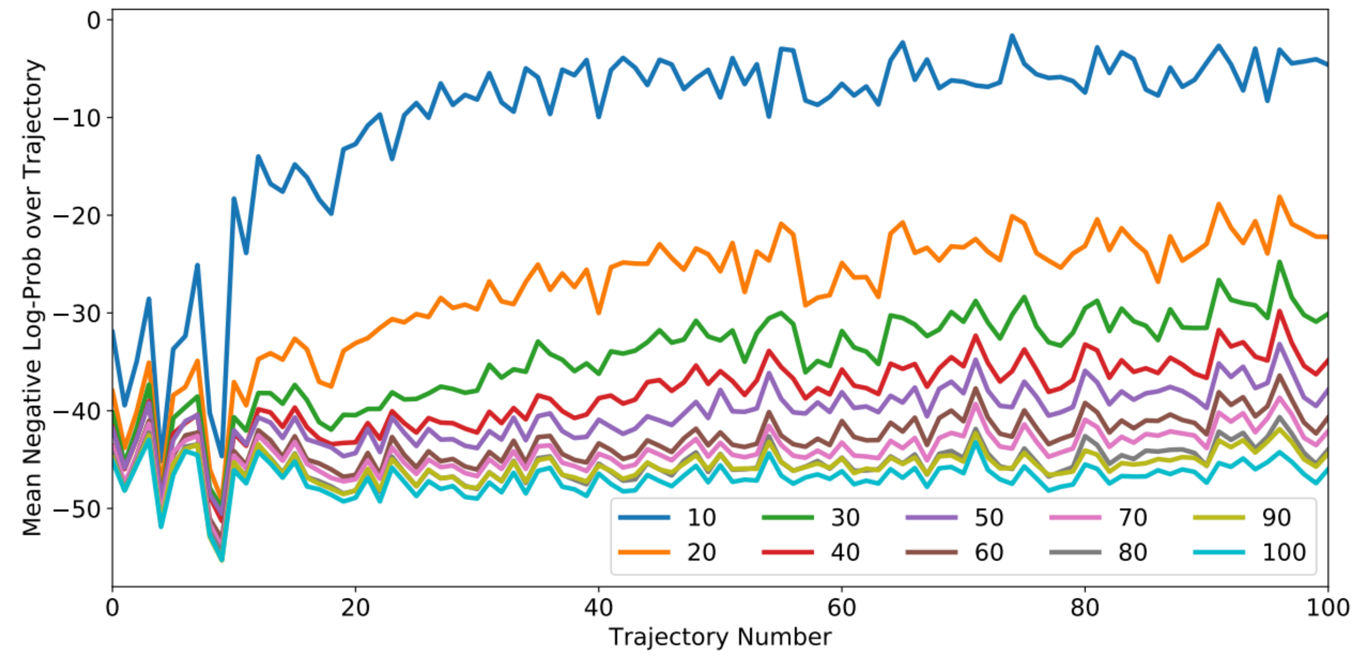
Case study: PETS




(a) Cartpole (b) 7-dof Pusher (c) 7-dof Reacher (d) Half-cheetah



(a) Mean squared error.



(b) Negative log likelihood.

Why model-based?

- Pros:
  - Sample efficiency
  - Improved multi-task performance
  - Transitions provide strong supervision (opposed to e.g., sparse reward)

- Cons:
  - Optimize the wrong objective
  - Can converge to worse performance if model is wrong
  - Can be difficult to train with high-dimensional states/observations (e.g., images)

Outline

Last week

Approach 1:
"Learn a model and use it to plan"

General idea

- Integrating planning and learning

"Dyna-style" algorithms

- Dyna-Q & Extensions

Remarks

Approach 2:
"Learn a model and improve model-free learning"

A bird's eye view of previous lectures

- Value-based methods: learn **value functions** from experience
- Policy optimization: learn **policies** from experience
- Previous lecture: learn a **model** from experience (and **plan** to construct a policy)
- Today: integrate learning and planning into a single architecture

Note:

- Last week we used the term model to describe a dynamics model, i.e.,

$$x_{t+1} \sim p_\theta(x_{t+1} \mid x_t, u_t)$$

- In general, we can assume the model to represent the *unknowns* in our MDP $\mathscr{M} = (X, U, P, R)$

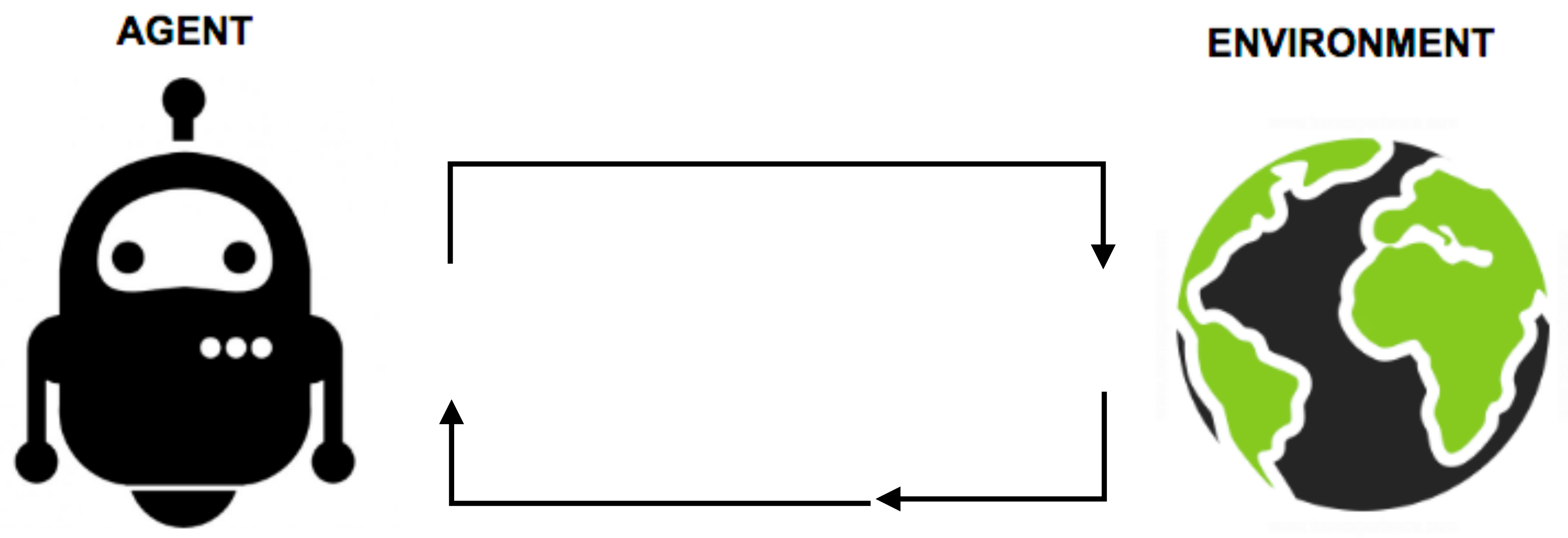
$$x_{t+1} \sim p_\theta(x_{t+1} \mid x_t, u_t)$$
$$R_t = r_\theta(x_t, u_t)$$

Examples of models:

- Table look-up
- Linear
- GP
- Neural network,…

Different sources of experience

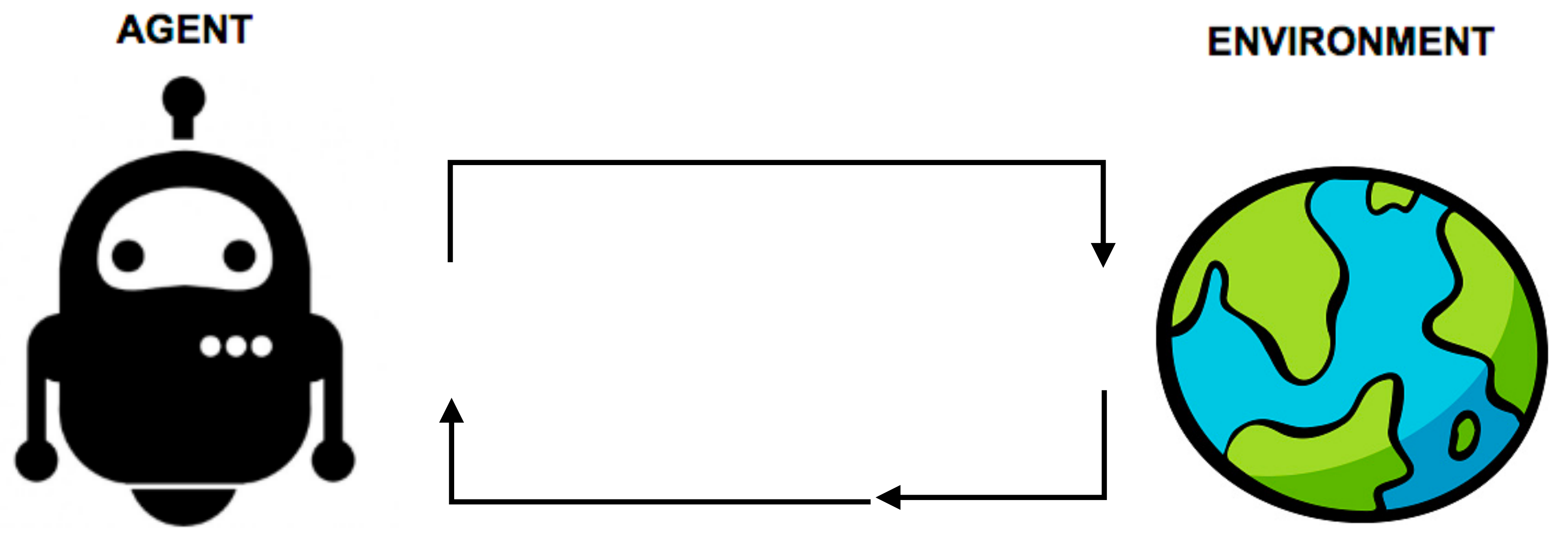
- Having a model enables us to consider two sources of experience



Real experience: sampled from the environment (true MDP)

- Interacting with the environment provides us with samples from the true MDP $\mathscr{M} = (X, U, P, R)$

$$x_{t+1} \sim P(x_{t+1} \,|\, x_t, u_t)$$
$$R_t = R(x_t, u_t)$$



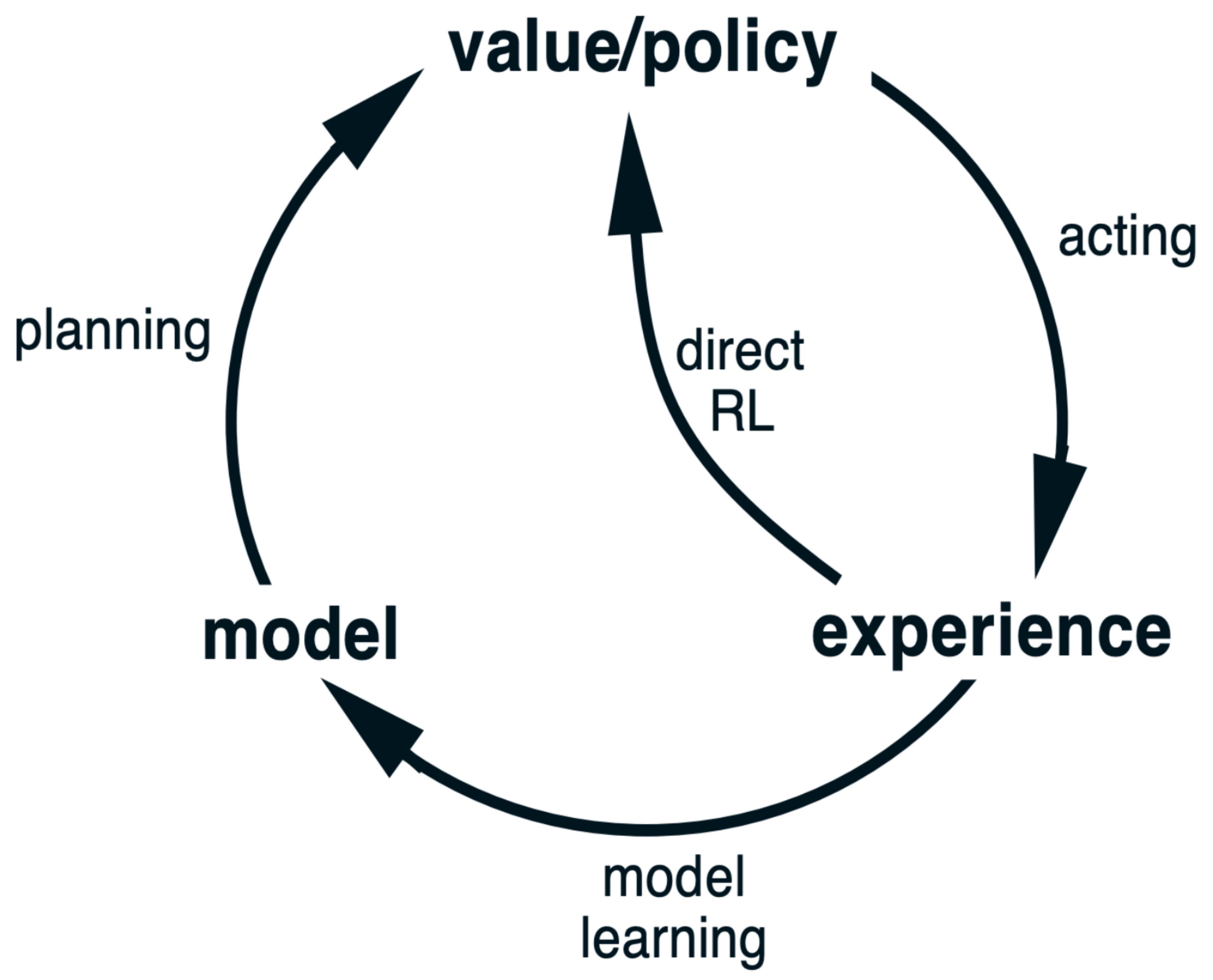
Simulated experience: sampled from the model (approximate MDP)

- Simulating transitions through the model provides us with samples from an approximation of the MDP

$$x_{t+1} \sim p_\theta(x_{t+1} \,|\, x_t, u_t)$$
$$R_t = r_\theta(x_t, u_t)$$

A general recipe for model-based acceleration

1. Interact with the environment to generate a dataset of transitions $\mathscr{D} = \{(x_t, u_t, r_t, x_{t+1})\}$
2. Fit dynamics/reward model to $\mathscr{D}$
3. Generate simulated experience under your model and use model-free algorithms



Arrow = relationship of influence and presumed improvement

Example: MBRL via policy gradient

- In PO, we defined the policy gradient via (variations of) the following equation:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta \left(u_{i,t} \mid x_{i,t}\right) Q_\phi(x_{i,t}, u_{i,t})$$

where used real experience (in the form of trajectories of interactions with the environment) to practically approximate the expectations

- We could consider the following scheme:

1. Run base policy $\pi_0(u_t | x_t)$ in the environment (e.g., random policy, exploration policy) and collect dataset of transitions $\mathscr{D} = \{(x_t, u_t, x_{t+1})_i\}$
2. Fit dynamics model to data to minimize error (or equivalently, maximize (log) likelihood)

$$\theta^* = \arg\min_\theta \sum_i \left\| f_\theta\left(x_t, u_t\right) - x_{t+1} \right\|^2$$

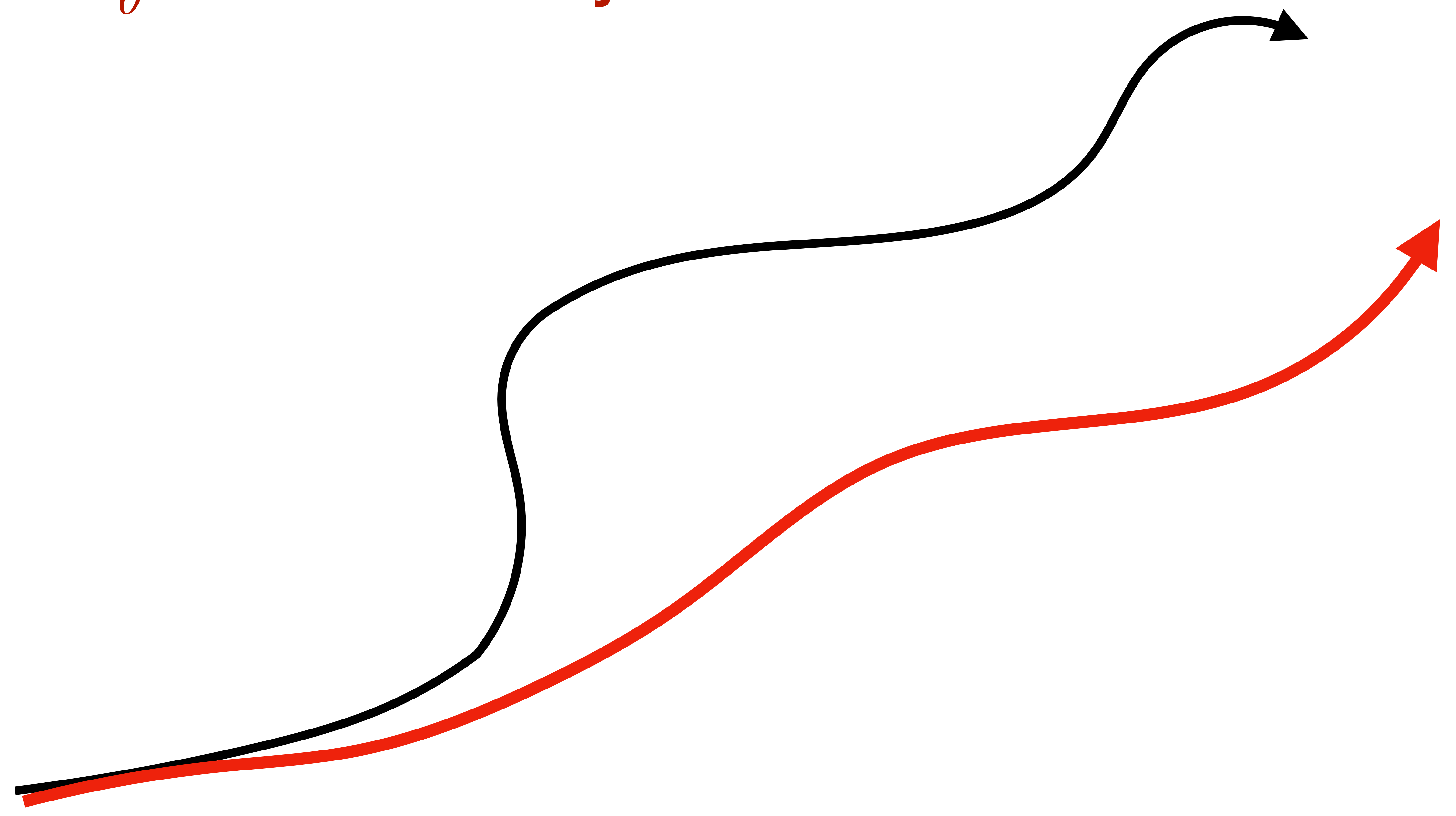
3. Use the learned model to generate simulated trajectories $\{\tau_i\}$ through policy π_θ
4. Use $\{\tau_i\}$ to improve π_θ via policy gradient

Question
What is a potential problem with this?

Issue with (long) model-based rollouts

Run π_θ with real dynamics

Run π_θ with estimated dynamics



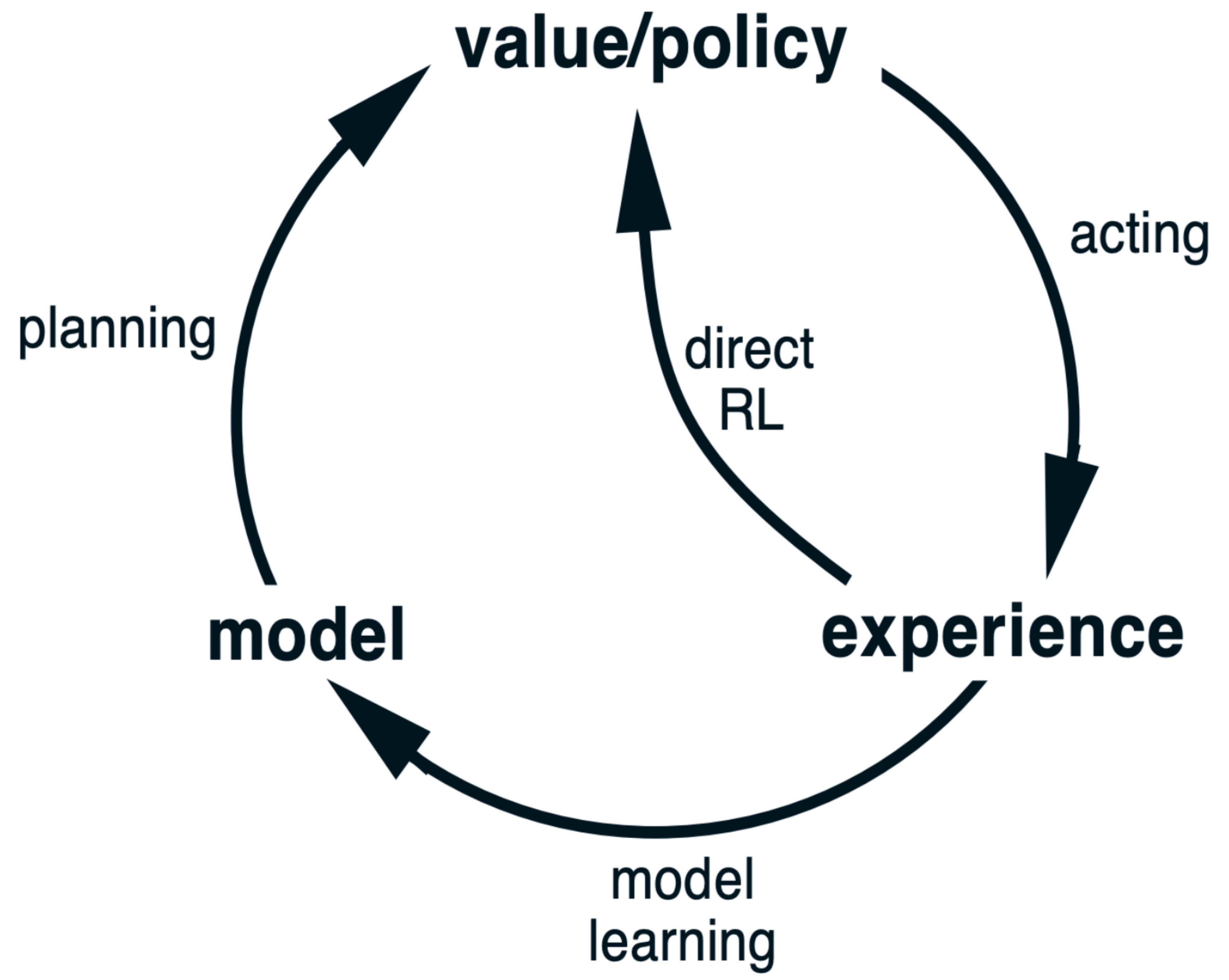
- We want to avoid long model-based rollouts, as these will necessarily incur in accumulating error
- At the same time, short rollouts do not guarantee exploration of "later timesteps"

Dyna-Q

```
Initialize $Q(x,u)$ and $Model(x,u), \forall x \in X, \forall u \in U$,
Repeat (for each episode):
    (a) $x_t \leftarrow$ current (non-terminal) state
    (b) Choose $u_t$ from $x_t$ using policy derived from Q (e.g., $\epsilon$-greedy)
    (c) Take action $u_t$, observe $r_t, x_{t+1}$
    (d) $Q(x_t,u_t) \leftarrow Q(x_t,u_t) + \alpha \left( r_t + \gamma \max_{u'_{t+1}} Q\left(x_{t+1}, u'_{t+1}\right) - Q(x_t,u_t) \right)$
    (e) $Model(x_t,u_t) \leftarrow x_{t+1}, r_t$
    (f) Repeat n times:
        $x_t \leftarrow$ sample random state previously observed
        $u_t \leftarrow$ sample random action previously taken in $x_t$
        $x_{t+1}, r_t \leftarrow Model(x_t,u_t)$ ; predict through model
        $Q(x_t,u_t) \leftarrow Q(x_t,u_t) + \alpha \left( r_t + \gamma \max_{u'_{t+1}} Q\left(x_{t+1}, u'_{t+1}\right) - Q(x_t,u_t) \right)$
    until $x_t$ is terminal
```

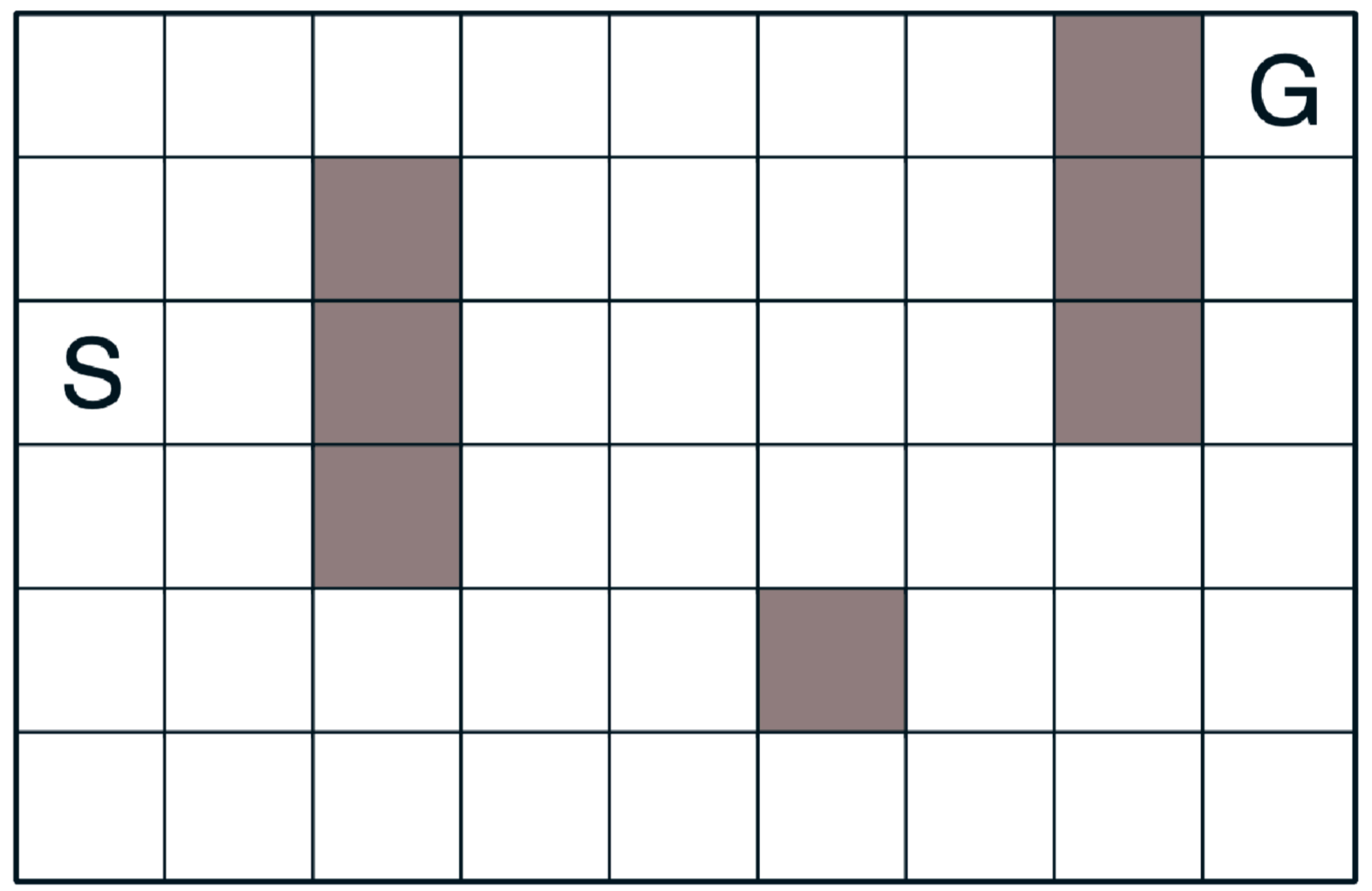
Dyna-Q

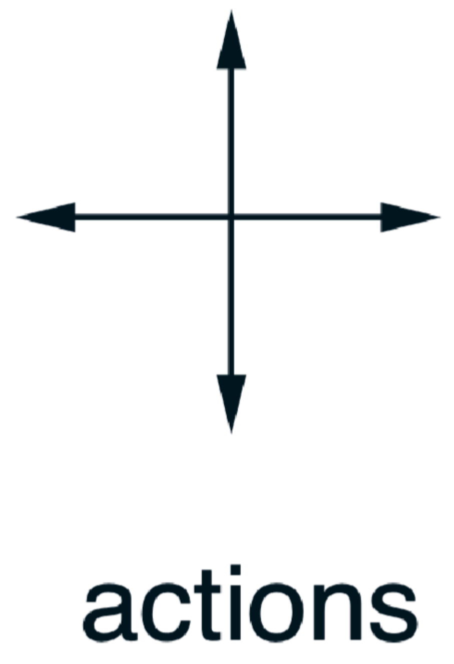
```
Initialize $Q(x,u)$ and $Model(x,u), \forall x \in X, \forall u \in U,$
Repeat (for each episode):
    (a) $x_t \leftarrow$ current (non-terminal) state
    (b) Choose $u_t$ from $x_t$ using policy derived from Q (e.g., $\epsilon$-greedy)
    (c) Take action $u_t$, observe $r_t, x_{t+1}$
    (d) $Q(x_t,u_t) \leftarrow Q(x_t,u_t) + \alpha \left( r_t + \gamma \max_{u'_{t+1}} Q\left(x_{t+1}, u'_{t+1}\right) - Q(x_t,u_t) \right)$
    (e) $Model(x_t,u_t) \leftarrow x_{t+1}, r_t$
    (f) Repeat n times:
        $x_t \leftarrow$ sample random state previously observed
        $u_t \leftarrow$ sample random action previously taken in $x_t$
        $x_{t+1}, r_t \leftarrow Model(x_t,u_t)$ ; predict through model
        $Q(x_t,u_t) \leftarrow Q(x_t,u_t) + \alpha \left( r_t + \gamma \max_{u'_{t+1}} Q\left(x_{t+1}, u'_{t+1}\right) - Q(x_t,u_t) \right)$
    until $x_t$ is terminal
```



- **Model** = Tabular model
- **Direct RL** = Q-learning
- **Planning** = 1-step

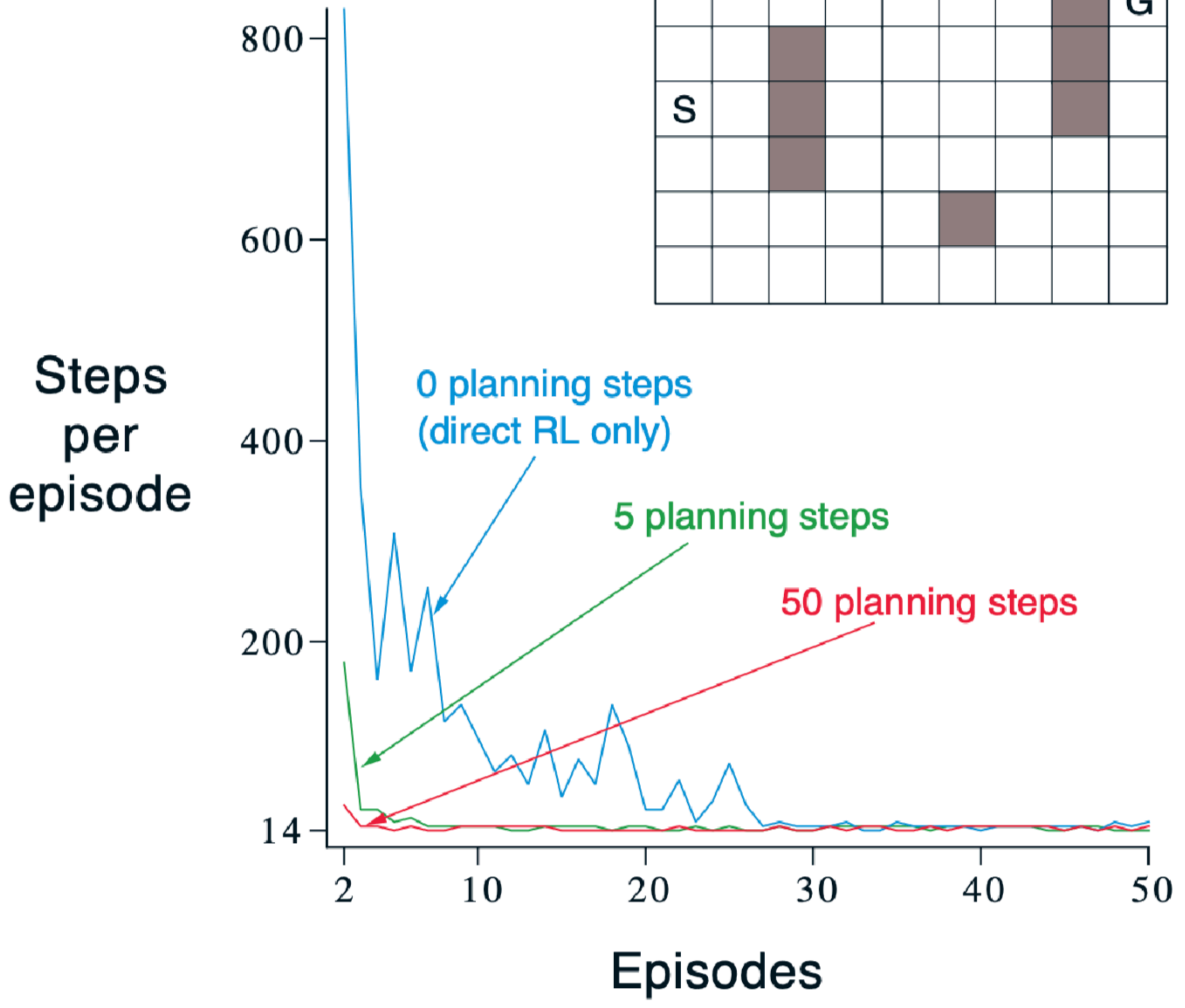
Example: Dyna-maze

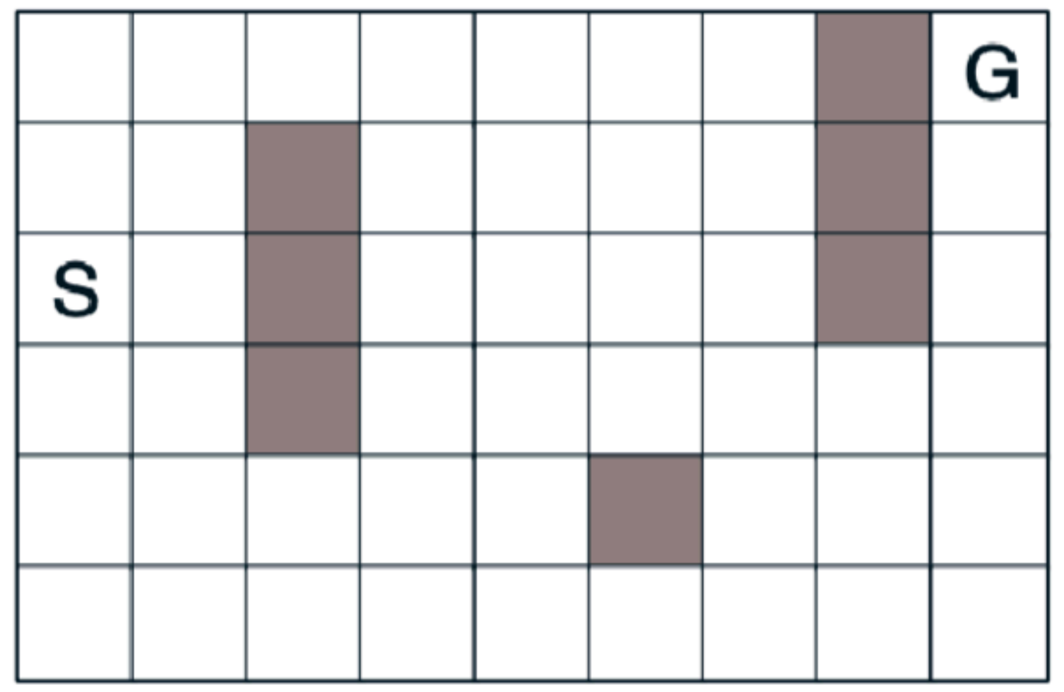


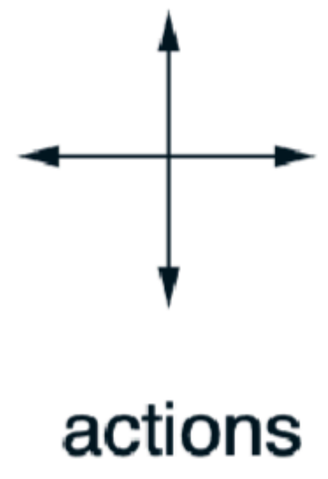


- 47 states, 4 actions
- Deterministic dynamics
- Reward = 0 everywhere, except +1 on *G*
- $\gamma = 0.95$
- Zero-initialized Q, $\alpha = 0.1, \epsilon = 0.1$

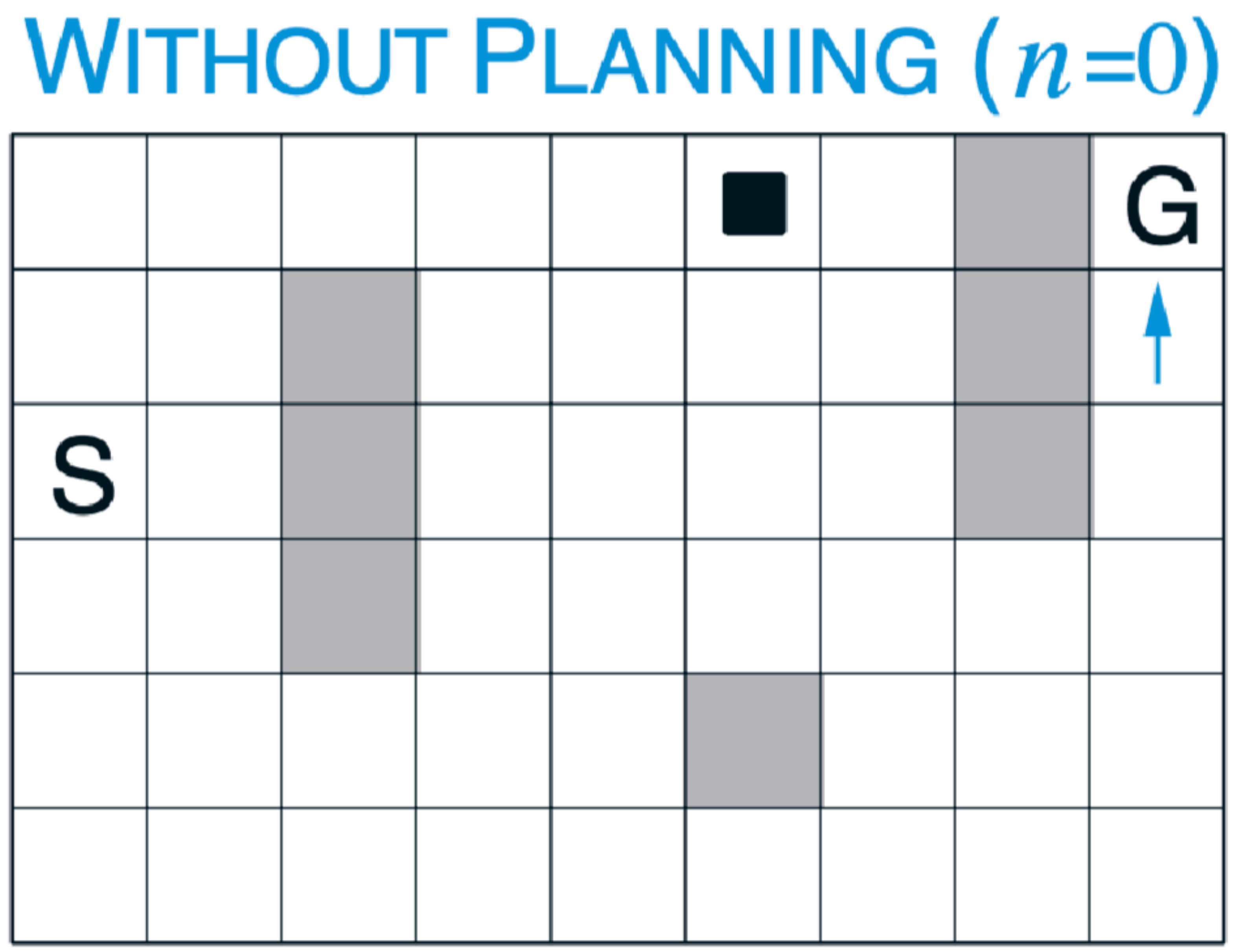
Example: Dyna-maze

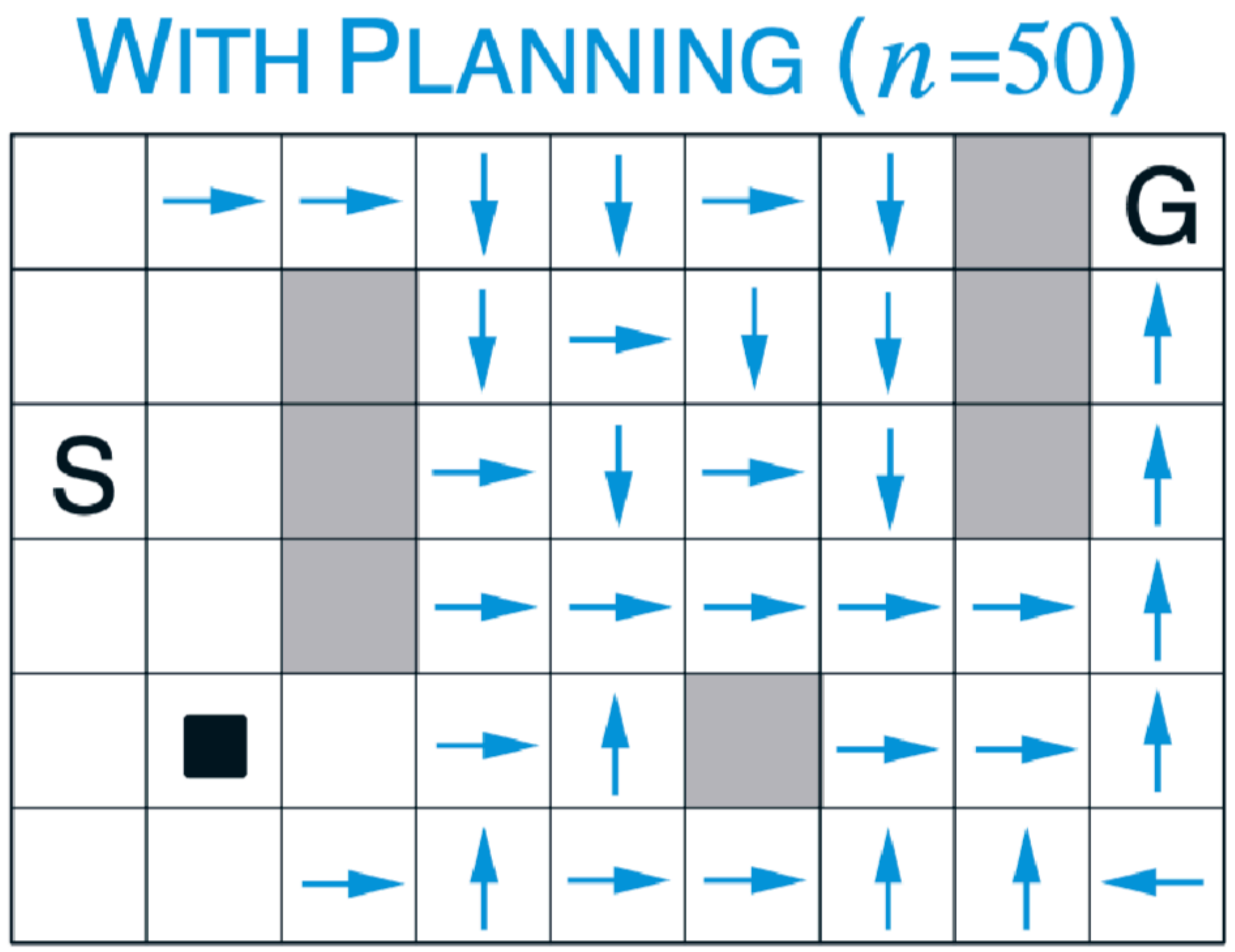






Example: Dyna-maze



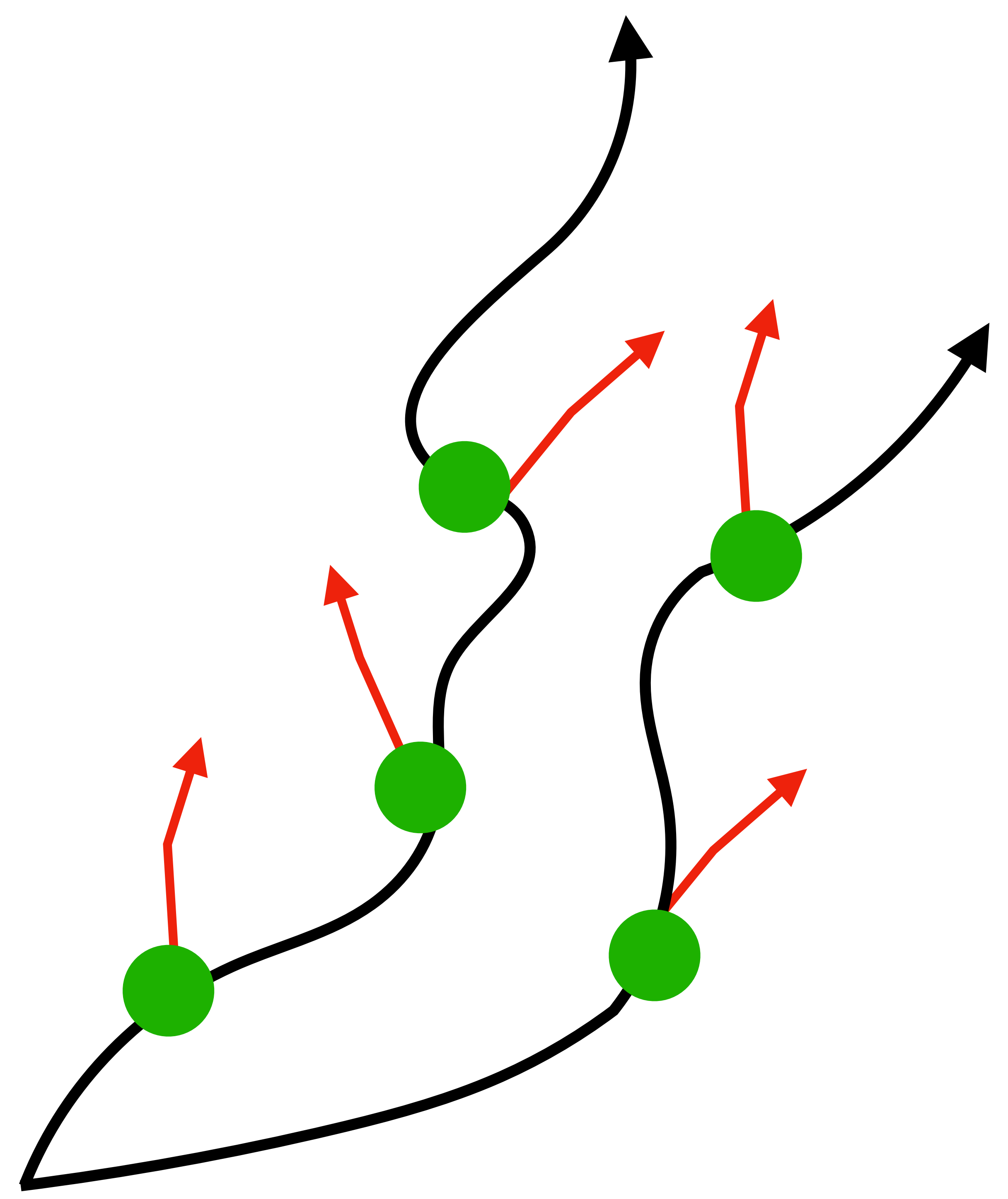


- The plot compares the policies found by Dyna-Q with and without planning, half-way through the second episode
- Without planning (n = 0), each episode adds only one additional step to the policy, and so only one step (the last) has been learned so far
- With planning, again only one step is learned during the first episode, but during the second episode planning allows to develop an extensive policy that will reach almost back to the start state

“Dyna-style” algorithms

- Dyna-Q represents a specific choice of model, planning, direct RL algorithm, etc.
- More generally, we can define the following recipe for “dyna-style” algorithms

1. Collect data $\{(x_t, u_t, r_t, x_{t+1})\}$
2. Learn dynamics / reward model, i.e., $p_\theta(x_{t+1} \mid x_t, u_t), r_\theta(x_t, u_t)$
3. Repeat n times
    1. Sample x_t from buffer
    2. Choose action u_t (from dataset, π, random, exploration, etc.)
    3. Simulate dynamics / reward $\hat{x}_{t+1} \sim p_\theta(x_{t+1} \mid x_t, u_t), \;\; \hat{r}_t = r_\theta(x_t, u_t)$
    4. Train on $\{(x_t, u_t, \hat{r}_t, \hat{x}_{t+1})\}$ via model-free RL
    5. Optionally, take k more model-based steps



- **Only uses short roll-outs**
- **While observing diverse states**

Example: model-based acceleration of DQN

Process 4: Model training

$$\theta^* = \arg\min_{\theta} \sum_i \left\| f_\theta\left(x_t, u_t\right) - x_{t+1} \right\|^2$$

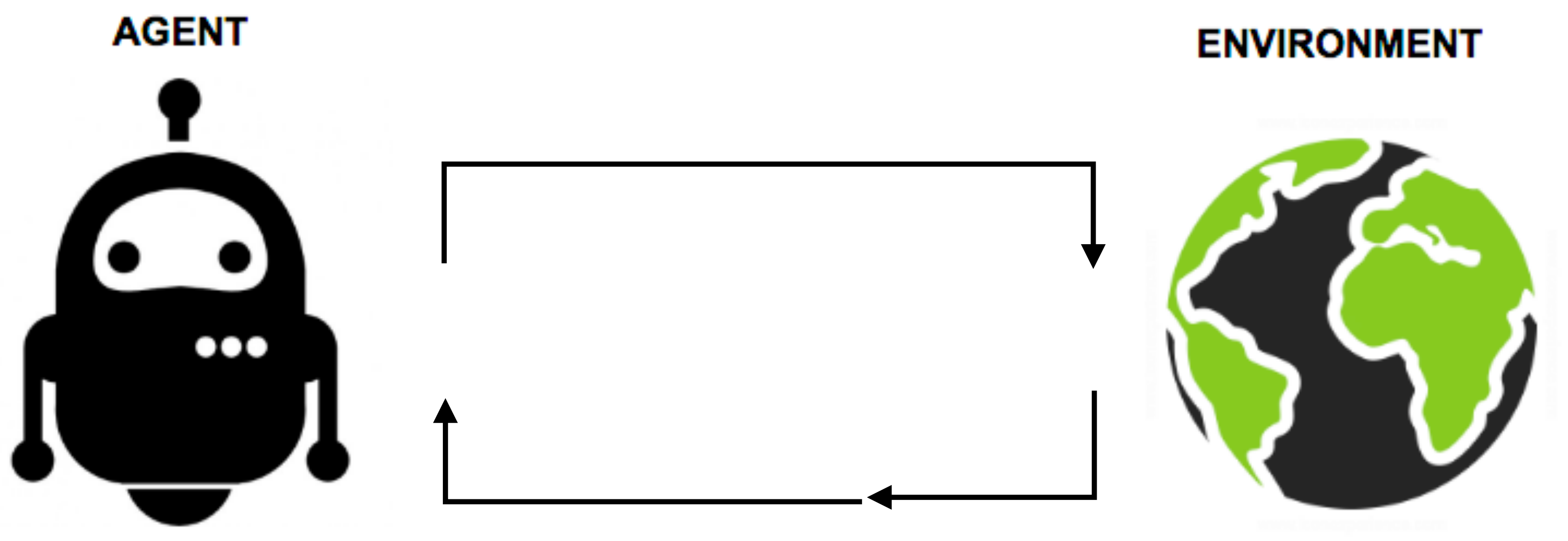
Process 5: Model-data collection

Dataset of transitions

(x_t, u_t, x_{t+1}, r_t)
(x_t, u_t, x_{t+1}, r_t)
(x_t, u_t, x_{t+1}, r_t)
(x_t, u_t, x_{t+1}, r_t)

- Pros:
  - Generally more sample efficient via augmented experience
- Cons:
  - Model errors can affect learning (we could consider ideas from uncertainty estimation)
  - In practice, these models tend to learn faster, but converge to overall worse performance

Process 1: Collect data



Dataset of transitions

(x_t, u_t, x_{t+1}, r_t)
(x_t, u_t, x_{t+1}, r_t)

Process 3: Target network update

$\theta \rightarrow \phi$

Process 2: Q-function regression

TD update

$$\Delta\theta = \alpha\left(r_t + \gamma \max_{u'_{t+1}} Q_\theta\left(x_{t+1}, u'_{t+1}\right) - \hat{Q}_\theta(x_t, u_t)\right)\nabla_\theta \hat{Q}_\theta(x_t, u_t)$$

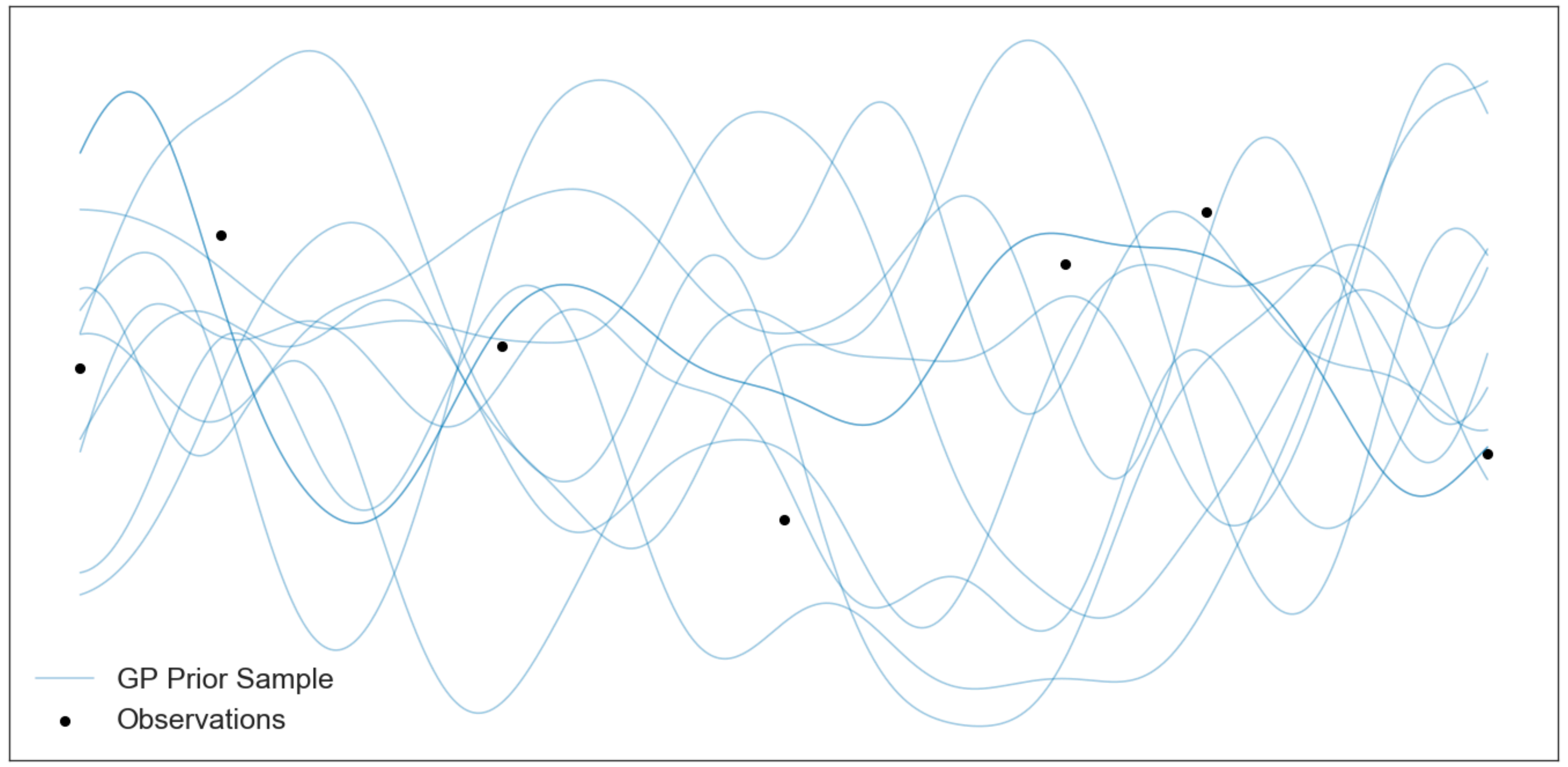
Case study: PILCO

- Deisenroth and Rasmussen, *Probabilistic inference for learning control*, ICML 2011
- Approach: use Gaussian process for dynamics model
  - Gives measure of *epistemic* uncertainty
  - Extremely sample efficient
- Pair with arbitrary (possibly nonlinear) policy
- By propagating the uncertainty in the transitions, capture the effect of small amount of data

Gaussian process reminder

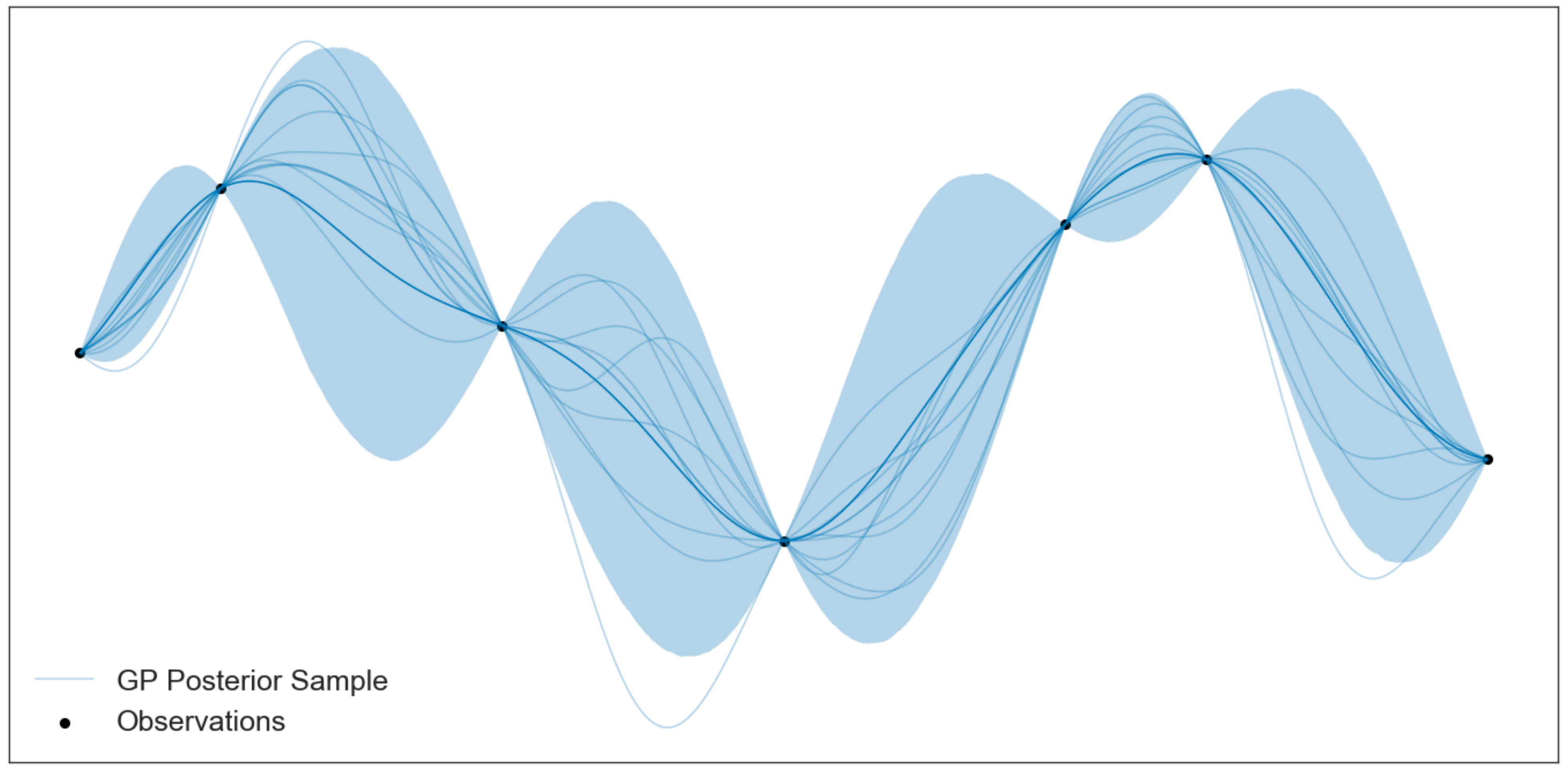
- Represent "distribution over functions"

Samples from **prior** distribution



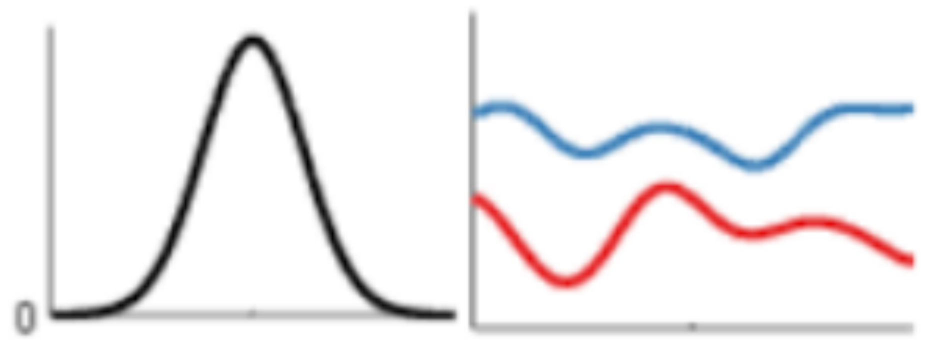
Bayesian inference

Samples from **posterior** distribution



- Typically, initialize with zero mean; behavior determined entirely by kernel
- Standard kernel choice: squared exponential, used in PILCO
  - Has smooth interpolating behavior

Squared Exponential Kernel

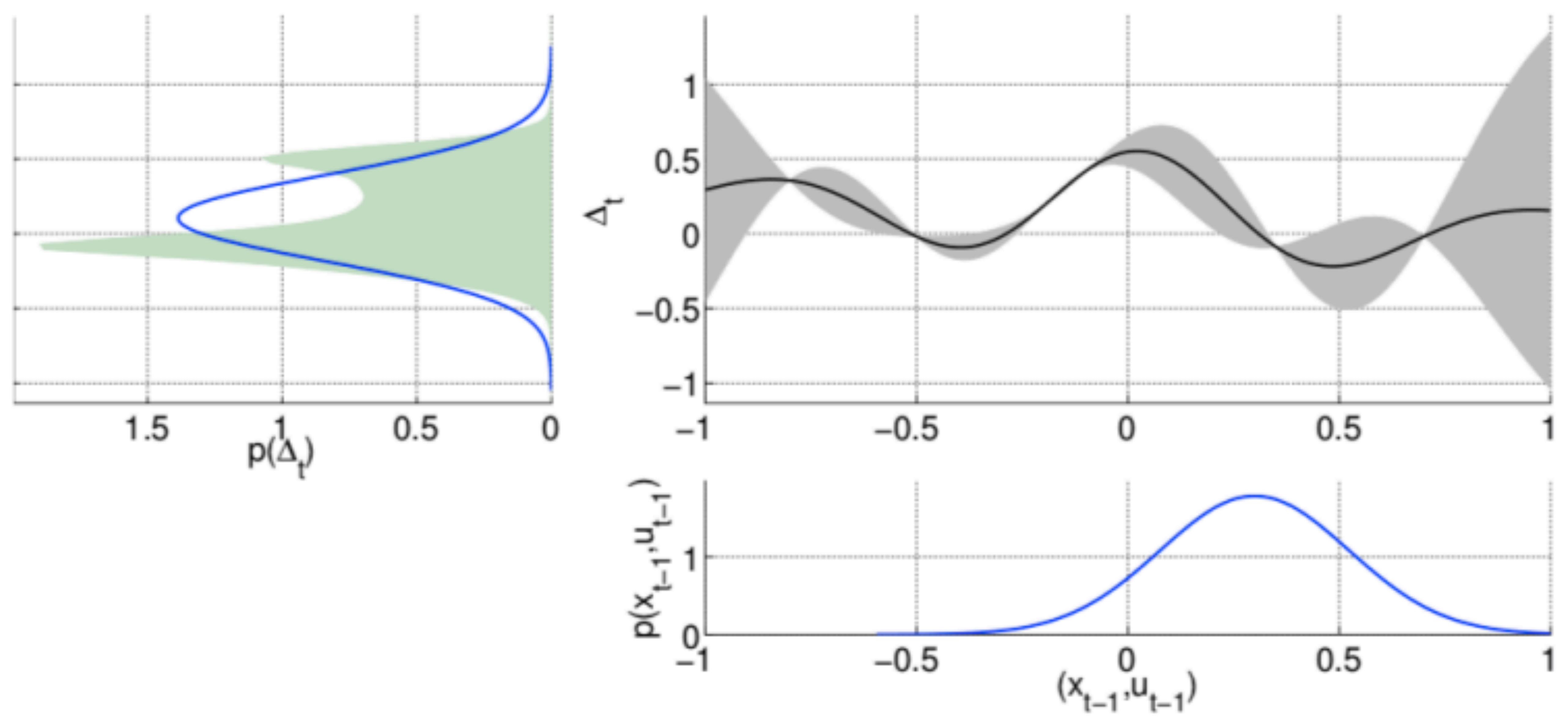


A.K.A. the Radial Basis Function kernel

$$k_{\mathrm{SE}}(x, x') = \sigma^2 \exp\left(-\frac{(x-x')^2}{2\ell^2}\right)$$

Case study: PILCO

- For GP conditioned on data, one step prediction is Gaussian
- But, need to make multistep predictions: so, need to derive multi-step predictive distribution
- Turn to approximating distribution at each time with a Gaussian via moment matching



Case study: PILCO

- All algorithm design choices made to ensure analytical tractability:

- Because of the squared exponential kernel, mean and variance can be computed in closed form
- Choose cost:

$$c(\mathbf{x}) = 1 - \exp\left(- \parallel \mathbf{x} - \mathbf{x}_{\text{target}} \parallel^2 / \sigma_c^2\right)$$

- which is similarly squared exponential; thus expected cost can be computed exactly, factoring in uncertainty

- Choose also radial basis function or linear policy, to enable analytical uncertainty propagation

PILCO (at a high level)

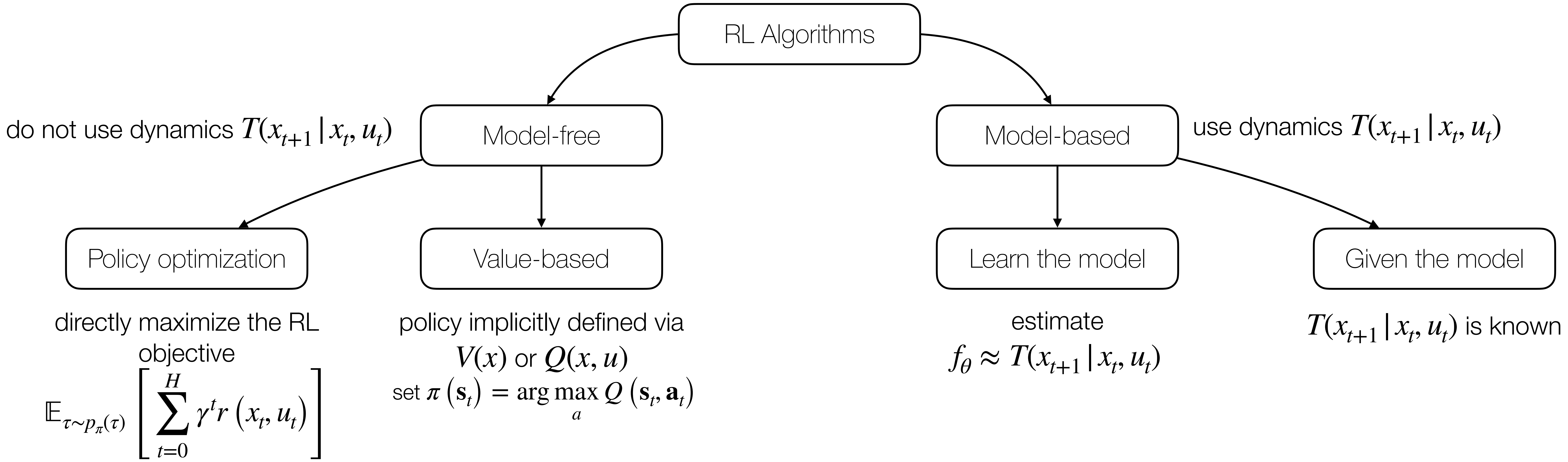
- Uncertainty prop: leverage specific functional forms to derive analytical expressions for mean and variance of trajectory under policy.

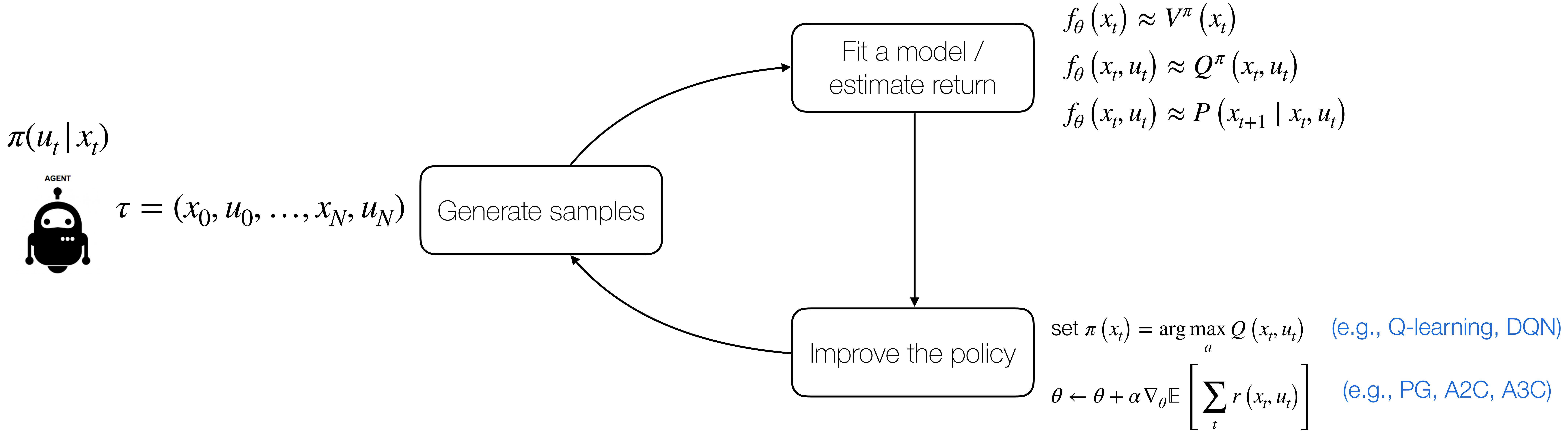
- Can use chain rule (aka backprop through time) to compute the gradient of expected total cost w.r.t. policy parameters

- Algorithm:
  - Roll out policy to get new measurements; update model
  - Compute (locally) optimal policy via gradient descent
    - This policy is “local” in the sense of the data we’ve given it, i.e., it’s tailored to the regions of state space it’s seen before; this is more general than “local” in the sense of linearization
  - Repeat

Combining model and policy learning

- We discussed two possible solutions, but there are infinitely many more!
- Very busy research direction! Many topics not covered here
  - Many possible combinations of planning/control, policies, values, and models
- Quite practical: model learning is data efficient and parameterized policy is cheap to evaluate at run time


RL Algorithms
Model-free
do not use dynamics $T(x_{t+1} | x_t, u_t)$
Model-based
use dynamics $T(x_{t+1} | x_t, u_t)$
Policy optimization
directly maximize the RL objective
$\mathbb{E}_{\tau \sim p_\pi(\tau)} \left[\sum_{t=0}^{H} \gamma^t r\left(x_t, u_t\right) \right]$
Value-based
policy implicitly defined via $V(x)$ or $Q(x, u)$
set $\pi\left(\mathbf{s}_t\right) = \arg\max_a Q\left(\mathbf{s}_t, \mathbf{a}_t\right)$
Learn the model
estimate $f_\theta \approx T(x_{t+1} | x_t, u_t)$
Given the model
$T(x_{t+1} | x_t, u_t)$ is known


$\pi(u_t | x_t)$
AGENT
$\tau = (x_0, u_0, \ldots, x_N, u_N)$
Generate samples
Fit a model / estimate return
$f_\theta\left(x_t\right) \approx V^\pi\left(x_t\right)$
$f_\theta\left(x_t, u_t\right) \approx Q^\pi\left(x_t, u_t\right)$
$f_\theta\left(x_t, u_t\right) \approx P\left(x_{t+1} \mid x_t, u_t\right)$
Improve the policy
set $\pi\left(x_t\right) = \arg\max_a Q\left(x_t, u_t\right)$
(e.g., Q-learning, DQN)
$\theta \leftarrow \theta + \alpha \nabla_\theta \mathbb{E}\left[\sum_t r\left(x_t, u_t\right)\right]$
(e.g., PG, A2C, A3C)

Next time

- Course recap
- Research presentations