Model Based RL

Deep Reinforcement Learning

Mohammad Hossein Rohban, Ph.D.

Courtesy: Some slides are adopted from CS 285 Berkeley, and CS 234 Stanford, and Pieter Abbeel's compact series on RL.

RIML

Deep RL Course

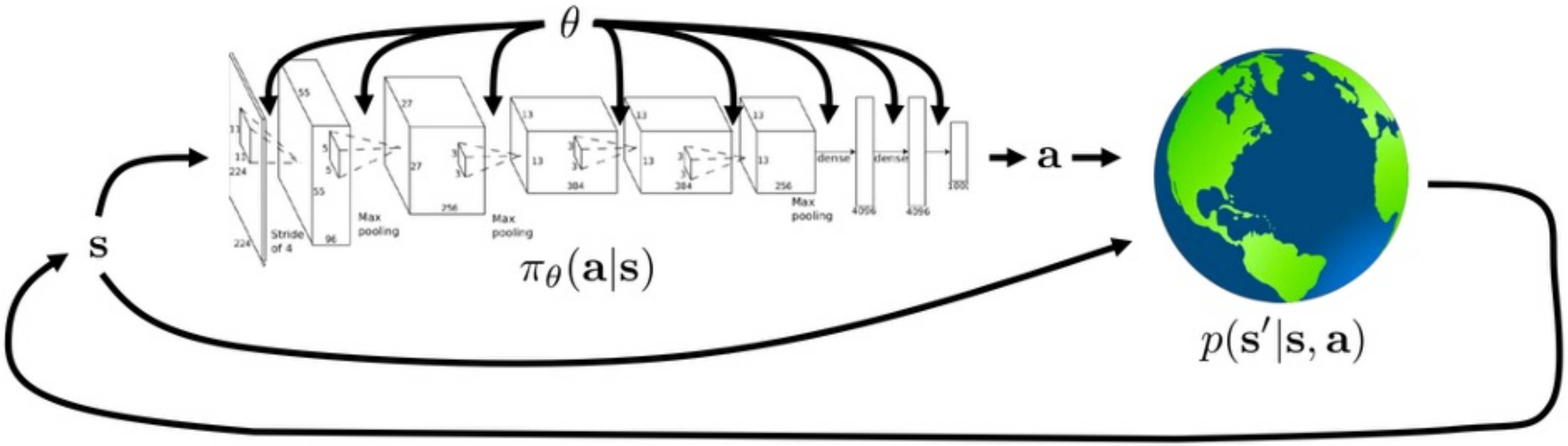
RIML Lab, CE Department, Sharif University of Technology

Spring 2026

Overview

- Introduction to model-based reinforcement learning
- What if we know the dynamics? How can we make decisions?
- Stochastic optimization methods
- Monte Carlo tree search (MCTS)
- Trajectory optimization
- Goal: Understand how we can perform planning with known dynamics models in discrete and continuous spaces

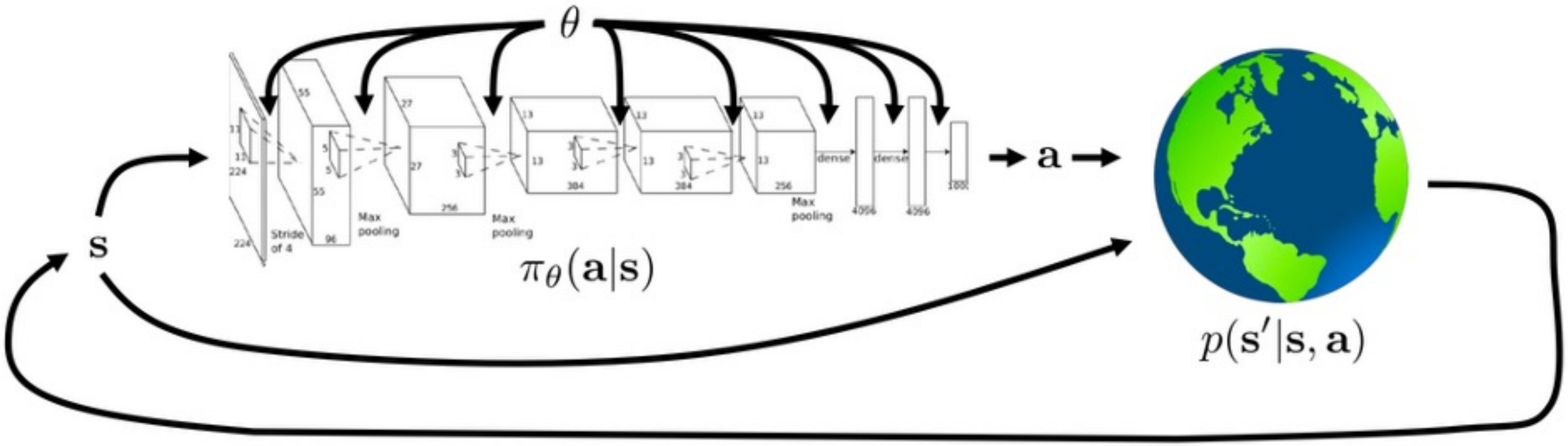
Recap: Model-Free RL



$$\underbrace{p_\theta(\mathbf{s_1},\mathbf{a_1},\ldots,\mathbf{s_T},\mathbf{a_T})}_{\pi_\theta(\tau)} = p(\mathbf{s_1})\prod_{t=1}^{T}\pi_\theta(\mathbf{a_t}|\mathbf{s_t})p(\mathbf{s_{t+1}}|\mathbf{s_t},\mathbf{a_t})$$

$$\theta^\star = \arg\max_\theta E_{\tau\sim p_\theta(\tau)}\left[\sum_t r(\mathbf{s_t},\mathbf{a_t})\right]$$

Recap: Model-Free RL



$$\underbrace{p_\theta(\mathbf{s_1}, \mathbf{a_1}, \dots, \mathbf{s_T}, \mathbf{a_T})}_{\pi_\theta(\tau)} = p(\mathbf{s_1}) \prod_{t=1}^{T} \pi_\theta(\mathbf{a_t}|\mathbf{s_t}) \cancel{p(\mathbf{s_{t+1}}|\mathbf{s_t}, \mathbf{a_t})}$$

$$\theta^\star = \arg\max_\theta E_{\tau \sim p_\theta(\tau)} \left[\sum_t r(\mathbf{s_t}, \mathbf{a_t}) \right]$$

What if we knew the transition dynamics?

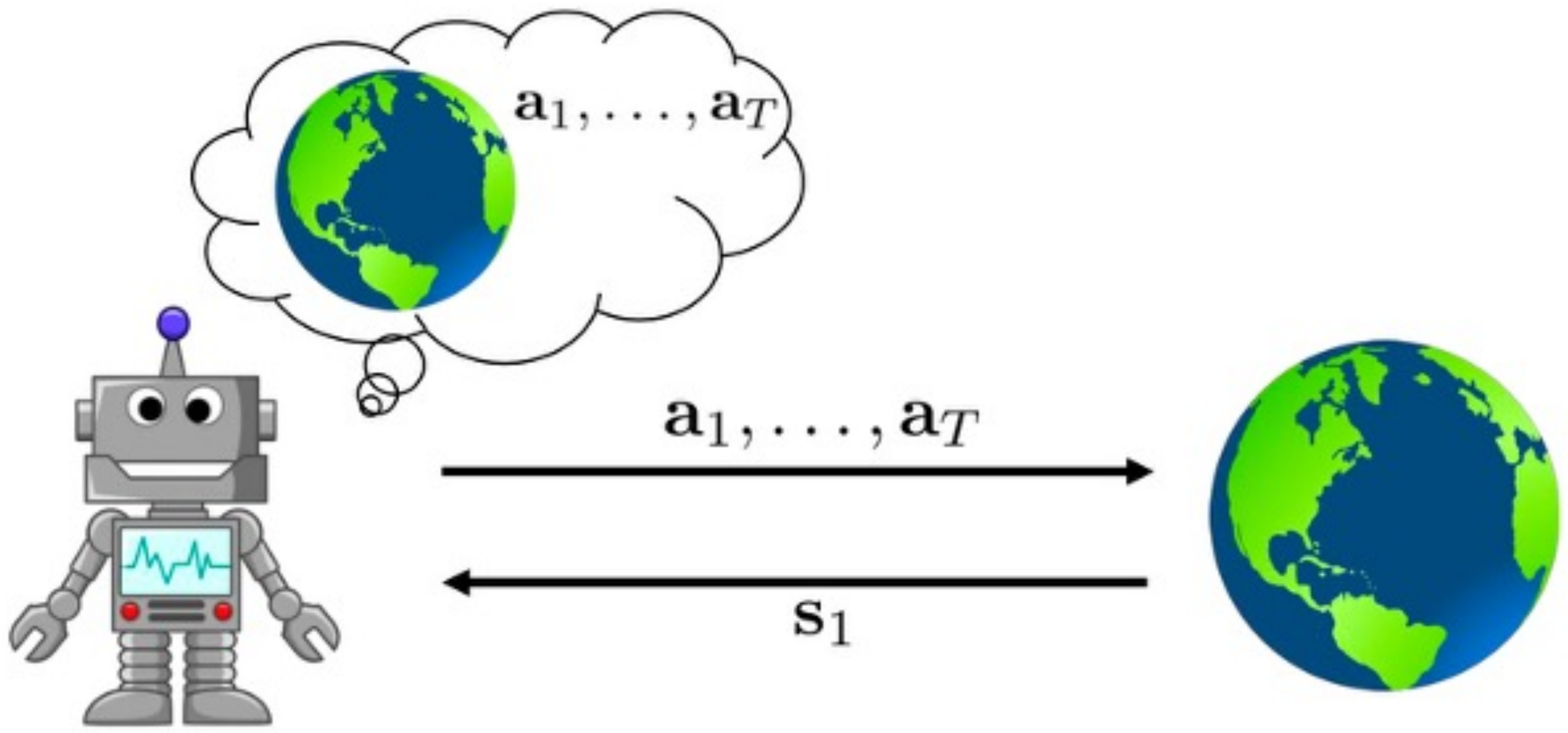
- Often we do know the dynamics
  - Games (e.g., Atari games, chess, Go)
  - Easily modeled systems (e.g., navigating a car)
  - Simulated environments (e.g., simulated robots, video games)
- Often we can learn the dynamics
  - System identification – fit unknown parameters of a known model
  - Learning – fit a general-purpose model to observed transition data

Does knowing the dynamics make things easier? Often, yes!

Model-Based RL

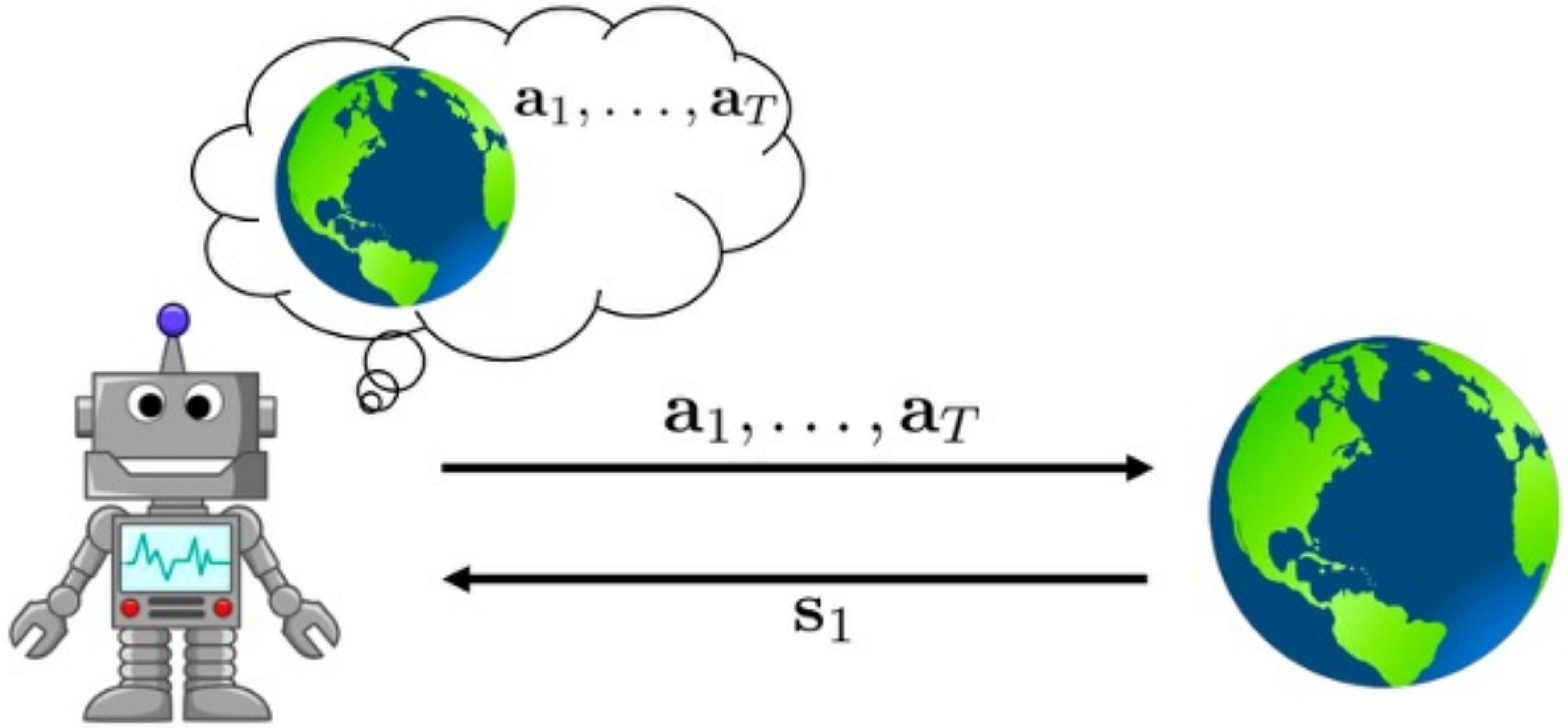
- Model-based reinforcement learning: learn the transition dynamics, then figure out how to choose actions.
- Today: how can we make decisions if we know the dynamics?
  - a. How can we choose actions under perfect knowledge of the system dynamics?
  - b. Optimal control, trajectory optimization, planning

The deterministic case



$$\mathbf{a_1}, \ldots, \mathbf{a_T} = \arg\max_{\mathbf{a_1},\ldots,\mathbf{a_T}} \sum_{t=1}^{T} r(\mathbf{s_t}, \mathbf{a_t}) \text{ s.t. } \mathbf{a_{t+1}} = f(\mathbf{s_t}, \mathbf{a_t})$$

The stochastic open-loop case



$$p_\theta(\mathbf{s_1}, \ldots, \mathbf{s_T} | \mathbf{a_1}, \ldots, \mathbf{a_T}) = p(\mathbf{s_1}) \prod_{t=1}^{T} p(\mathbf{s_{t+1}} | \mathbf{s_t}, \mathbf{a_t})$$

$$\mathbf{a}_1, \ldots, \mathbf{a}_T = \arg\max_{\mathbf{a}_1, \ldots, \mathbf{a}_T} E\left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) | \mathbf{a}_1, \ldots, \mathbf{a}_T\right]$$

The stochastic open-loop case (cont.)

کری می خواست به عیادت بیماری برود.اندیشید که هنگام احوال پرسی ممکن است صدای اورانشنوم وپاسخی ناشایسته بدهم.ازاین رودرپی چاره برآمدوبالاخره باخود گفت:بهتراست پرسشهارا پیش ازرفتن بسنجم وپاسخ رانیزبرآورد کنم تادچاراشتباه نشوم.

بنابراین پرسشهای خودراچنین پیش بینی کرد:

-ابتداازاومی پرسم حالت بهتراست؟ اوخواهد گفت "آری" من درجواب می گویم:خدا را شکر

-بعدازاومی پرسم چه خورده ای؟ لابد نام غذایی راخواهد آورد.من می گویم گوارا باد.

-درپایان می پرسم پزشکت کیست؟ نام پزشکی رامی گویدومن پاسخ می دهم:مقدمش مبارک باد.

....

چون به خانه ی بیماررسید همان گونه که ازپیش آماده شده بودبه احوال پرسی پرداخت:

-کر گفت:"چگونه ای؟"

بیمارگفت: مُردم

کر گفت: خدارا شکر

بیمارازاین سخن بیجا برآشفت.

-بعدازآن پرسید:"چه خورده ای؟"

بیمارگفت: زهر

کر گفت: گواراباد.داروی خوبی است.

بیمار ازاین پاسخ نیزبیشتربه خود پیچید.

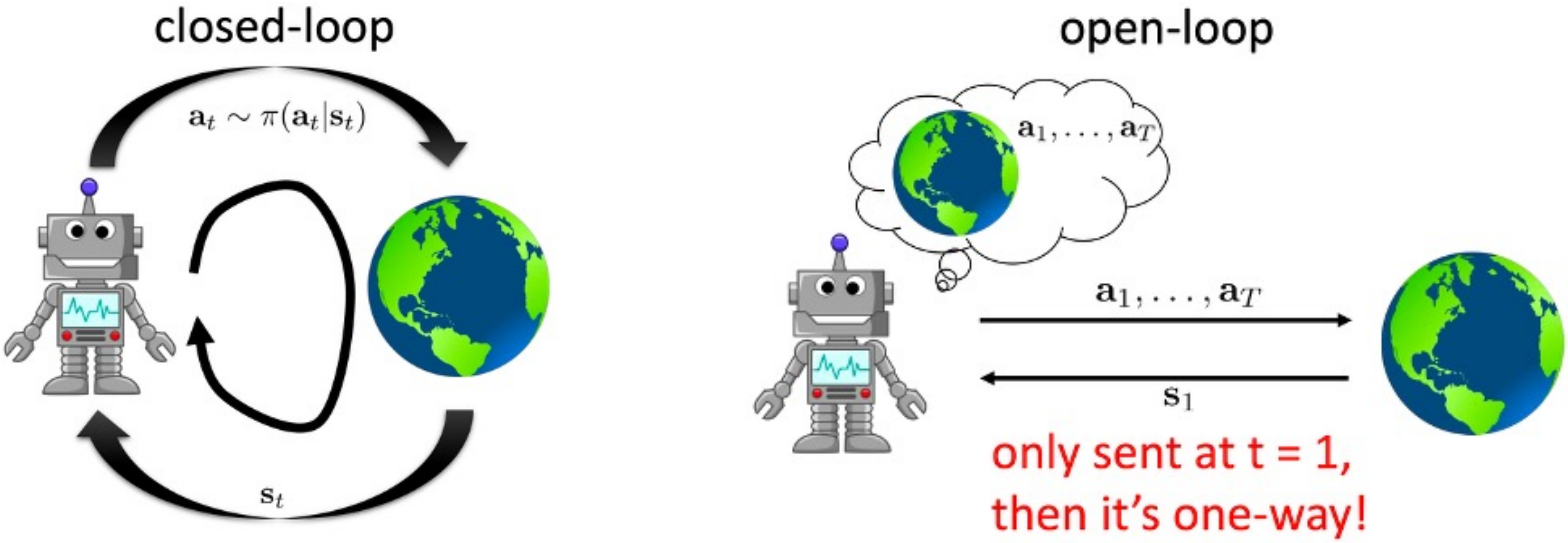
-بعد ازآن کر گفت:" ازطبیبان کیست او   کاوهمی آید به چاره پیش تو؟"

بیمار که آشفتگی وناراحتی اش به نهایت رسیده بود در پاسخ گفت:

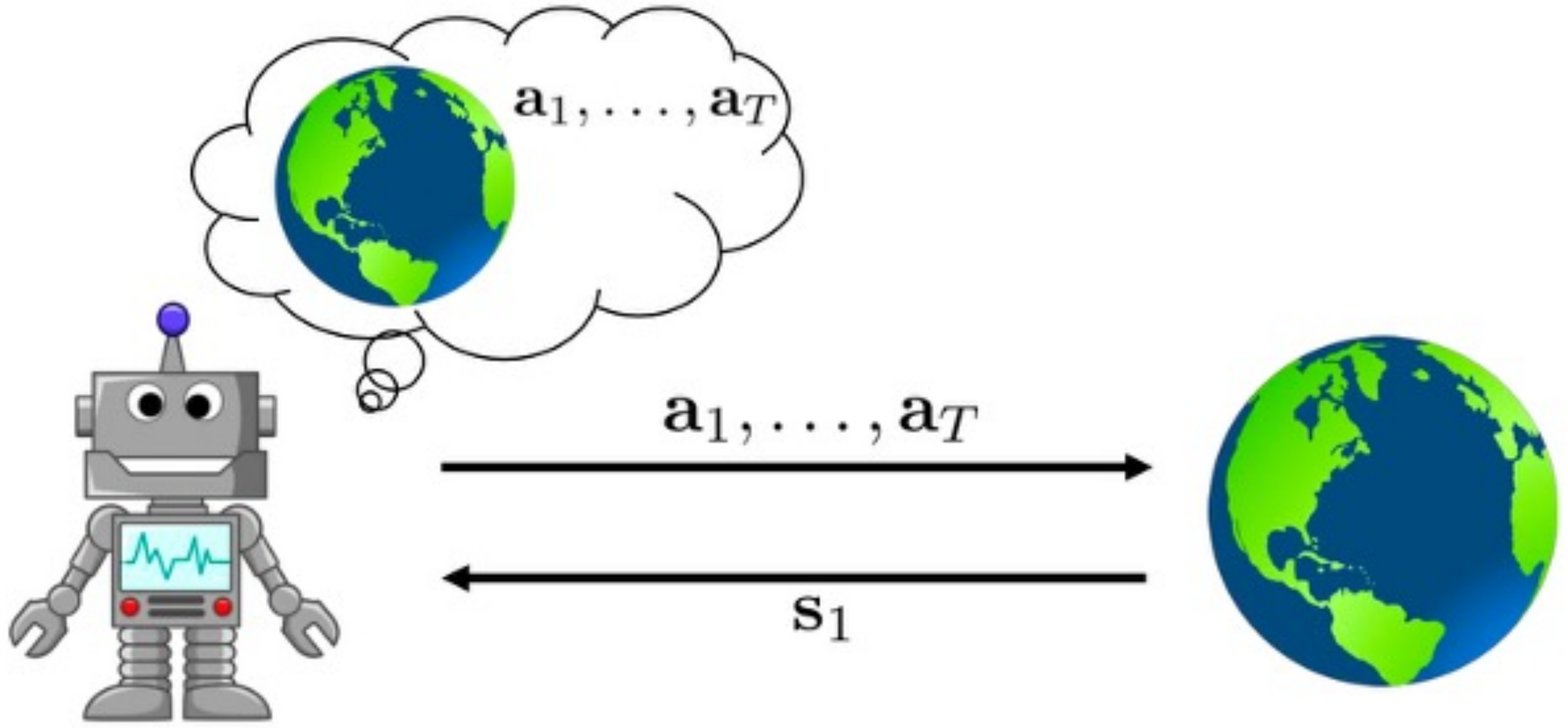
عزرائیل می آید, برو.

کر گفت: پایش بس مبارک.شاد شو!

The stochastic open-loop case (cont.)



The stochastic open-loop case (cont.)



Form of π?
Neural Net GLOBAL

Time varying linear
$K_t s_t + k_t$ LOCAL

$$p(\mathbf{s_1}, \mathbf{a_1}, \ldots, \mathbf{s_T}, \mathbf{a_T}) = p(\mathbf{s_1}) \prod_{t=1}^{T} \pi(\mathbf{a_t}|\mathbf{s_t}) p(\mathbf{s_{t+1}}|\mathbf{s_t}, \mathbf{a_t})$$

$$\pi = \arg\max_{\pi} E_{\tau \sim p(\tau)} \left[\sum_t r(\mathbf{s_t}, \mathbf{a_t}) \right]$$

Stochastic Optimization

✦ abstract away optimal control/planning:

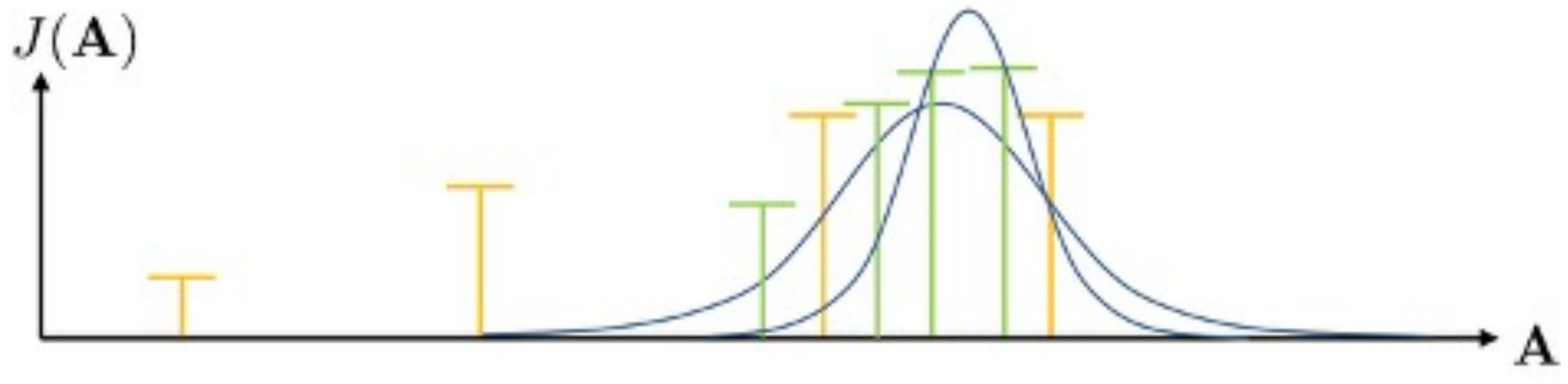
$$\mathbf{a_1}, \dots, \mathbf{a_T} = \arg \max_{\mathbf{a_1},\dots,\mathbf{a_T}} \underbrace{J(\mathbf{a_1}, \dots, \mathbf{a_T})}_{\text{don't care what this is}} \qquad \mathbf{A} = \arg\max_{\mathbf{A}} J(\mathbf{A})$$

✦ simplest method: guess & check "random shooting method"

1. pick $\mathbf{A}_1, \dots, \mathbf{A}_N$ from some distribution (e.g., uniform)
2. choose $\mathbf{A}_i$ based on $\arg max_i J(\mathbf{A}_i)$

CEM (Cross Entropy Method)

1. pick $\mathbf{A}_1, \dots, \mathbf{A}_N$ from some distribution (e.g., uniform)
2. choose $\mathbf{A}_i$ based on $\text{arg}max_i J(\mathbf{A}_i)$



Can we do better?

cross-entropy method with continuous-valued inputs:

1. sample $\mathbf{A}_1, \dots, \mathbf{A}_N$ from $p(\mathbf{A})$
2. evaluate $J(\mathbf{A}_1), \dots, J(\mathbf{A}_N)$
3. pick the elites $\mathbf{A}_{i_1}, \dots, \mathbf{A}_{i_M}$ with the highest value, where $M < N$
4. refit $p(\mathbf{A})$ to the elites $\mathbf{A}_{i_1}, \dots, \mathbf{A}_{i_M}$

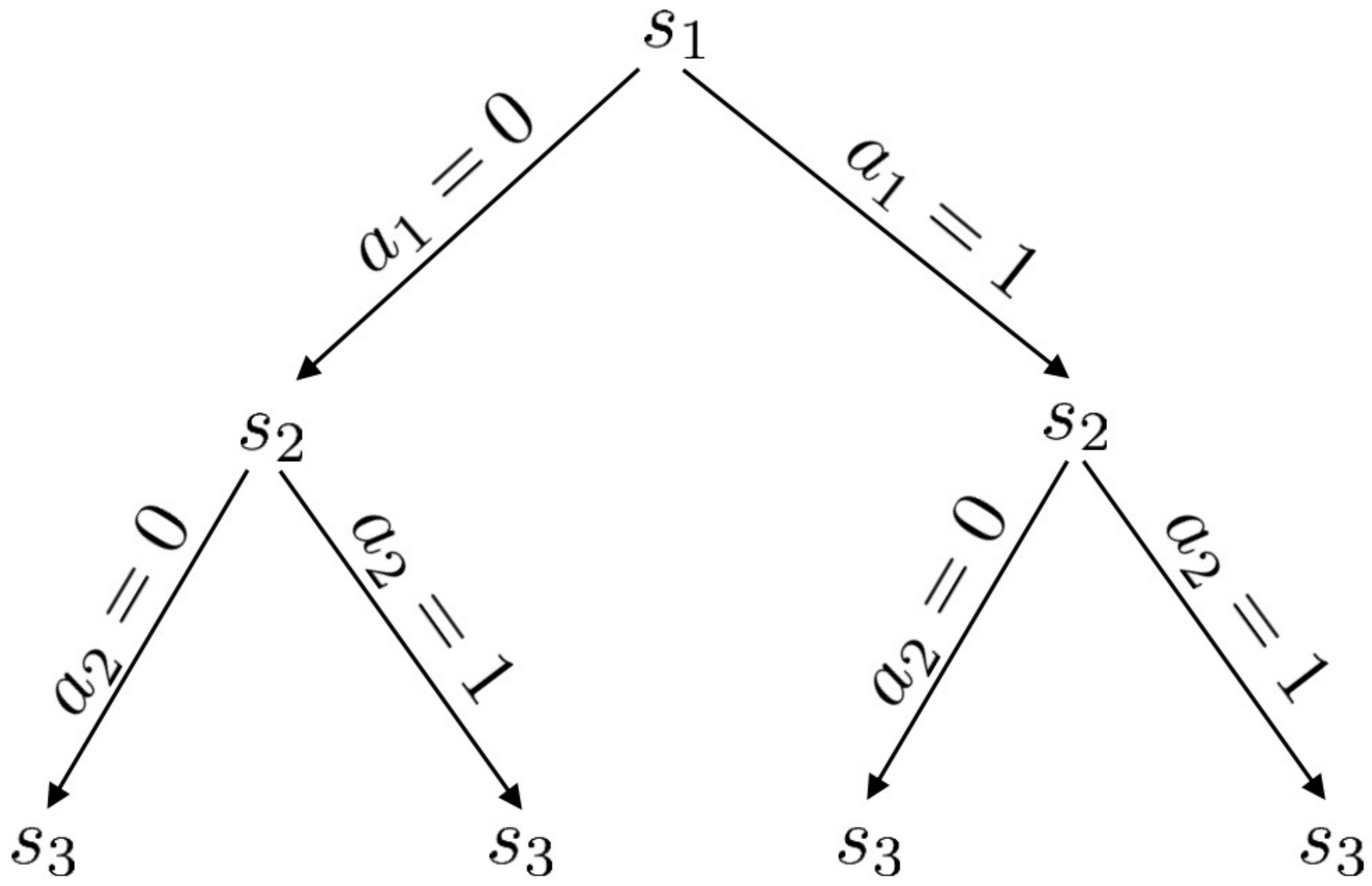
Pros and Cons!

- Pros
  - Could be very fast (Parallelizable)
  - Extremely simple
- Cons
  - Very harsh dimensionality limit
  - Only open-loop planning

Discrete Case: Monte Carlo Tree Search

s_t a_t

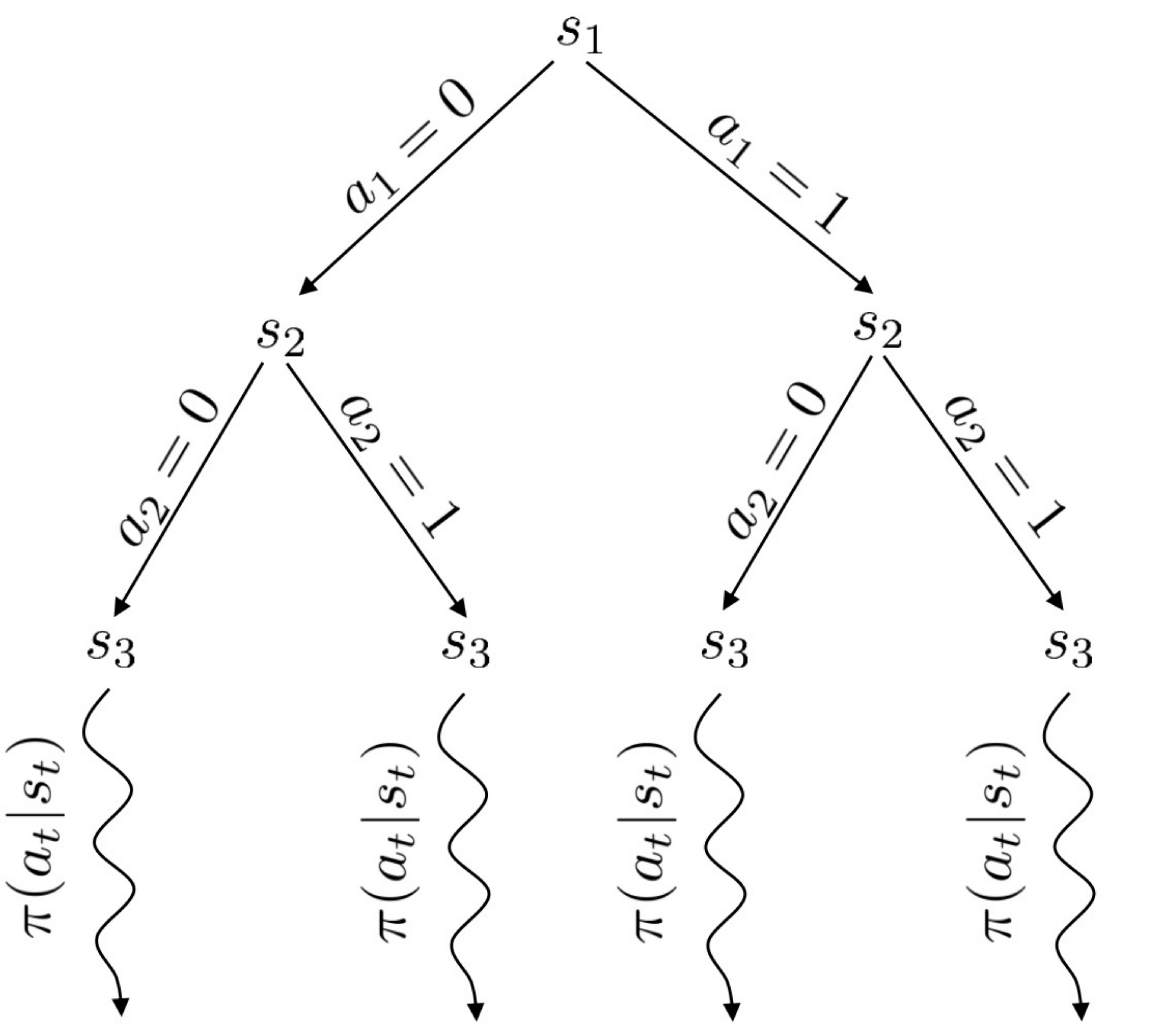
discrete planning as a search problem



Discrete Case: Monte Carlo Tree Search (cont.)

how to approximate value without full tree?

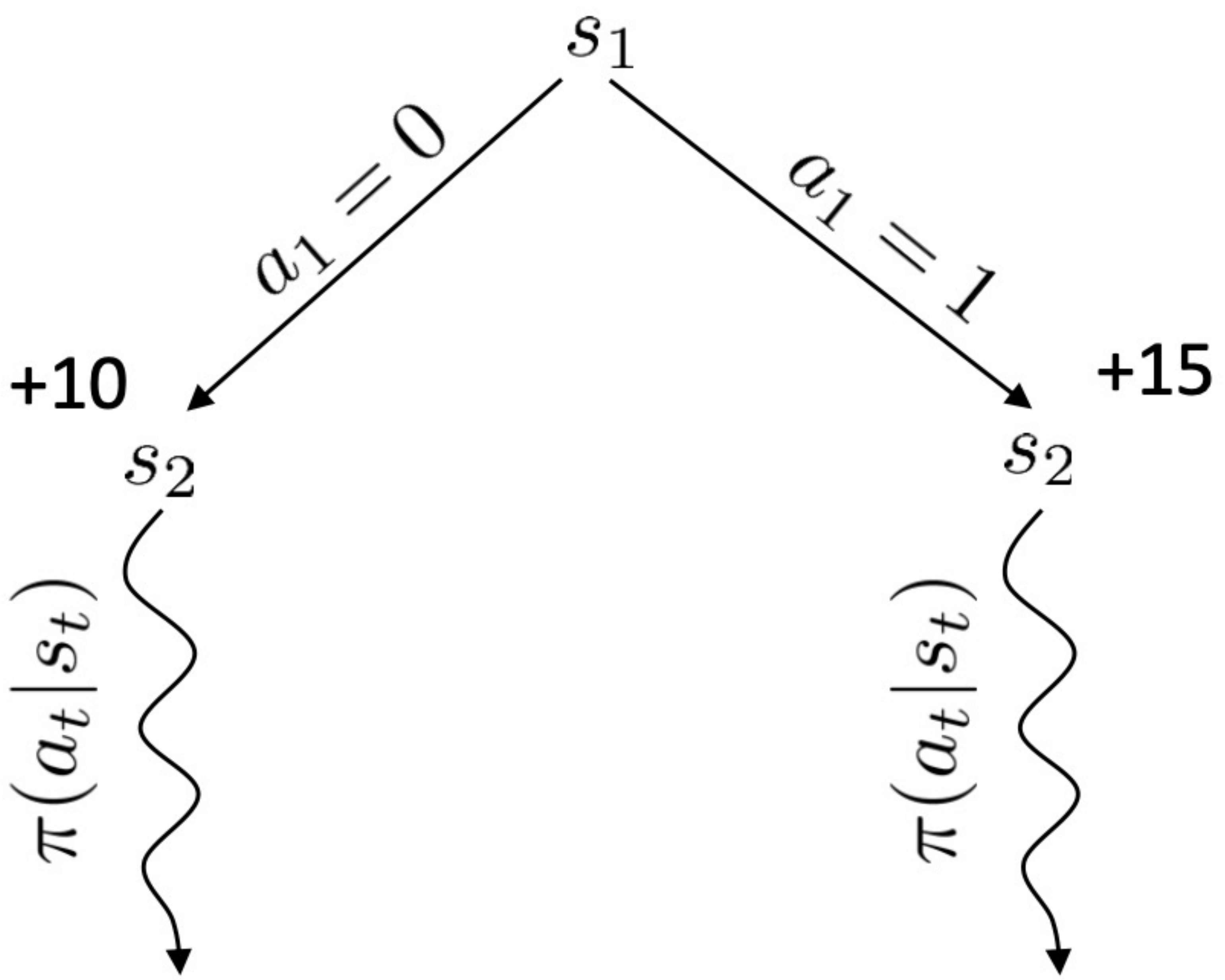
s_t a_t



e.g., random policy

Discrete Case: Monte Carlo Tree Search (cont.)

can't search all paths – where to search first?



intuition: choose nodes with best reward, but also prefer rarely visited nodes

Discrete Case: Monte Carlo Tree Search (cont.)

Tree Search Algorithm

generic MCTS sketch

1. find a leaf s_l using TreePolicy(s_1)
2. evaluate the leaf using $DefaultPolicy(s_l)$
3. update all values in tree between s_1 and s_l

take best action from s_1

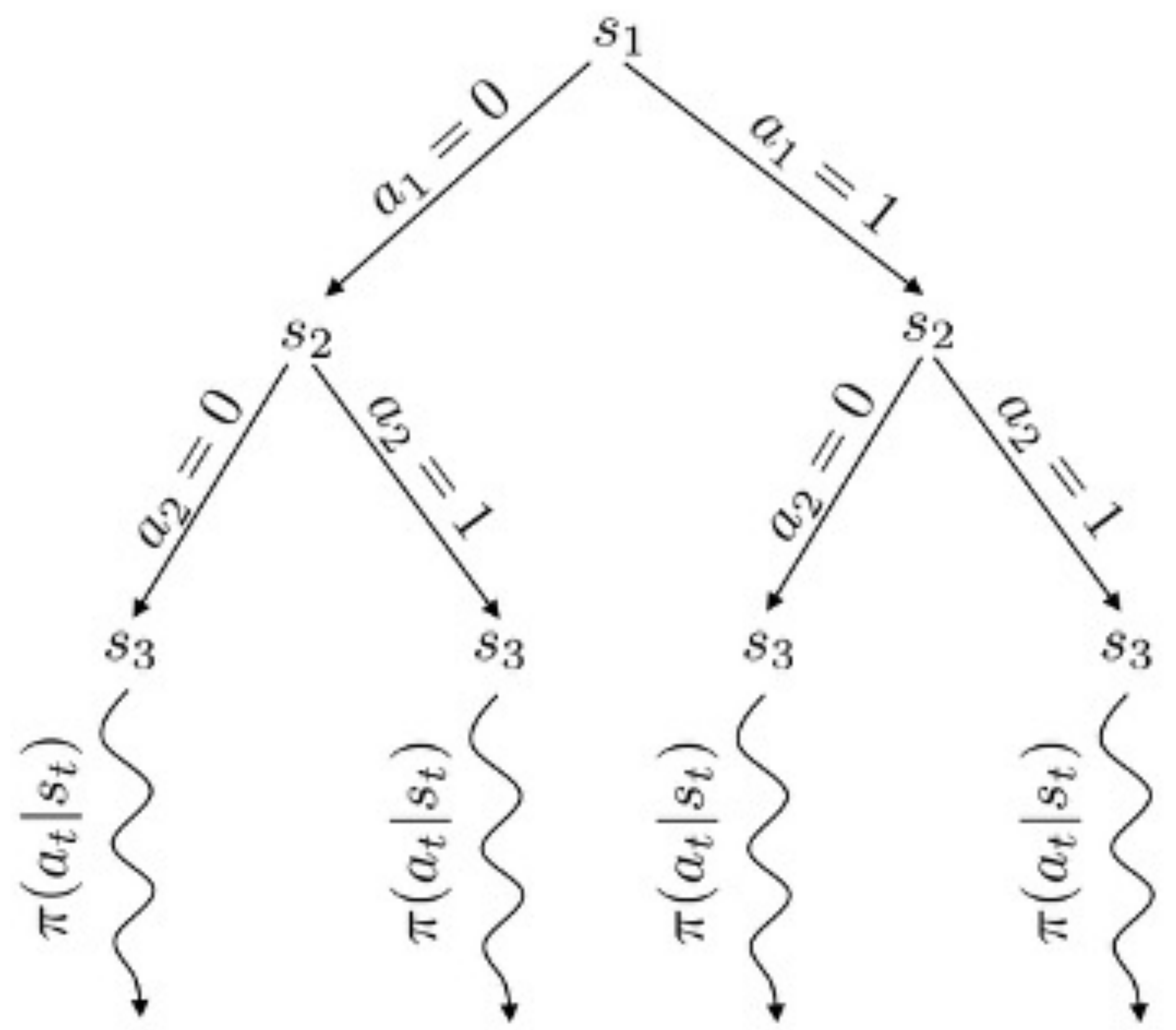
TreePolicy Function

UCT TreePolicy(s_t)

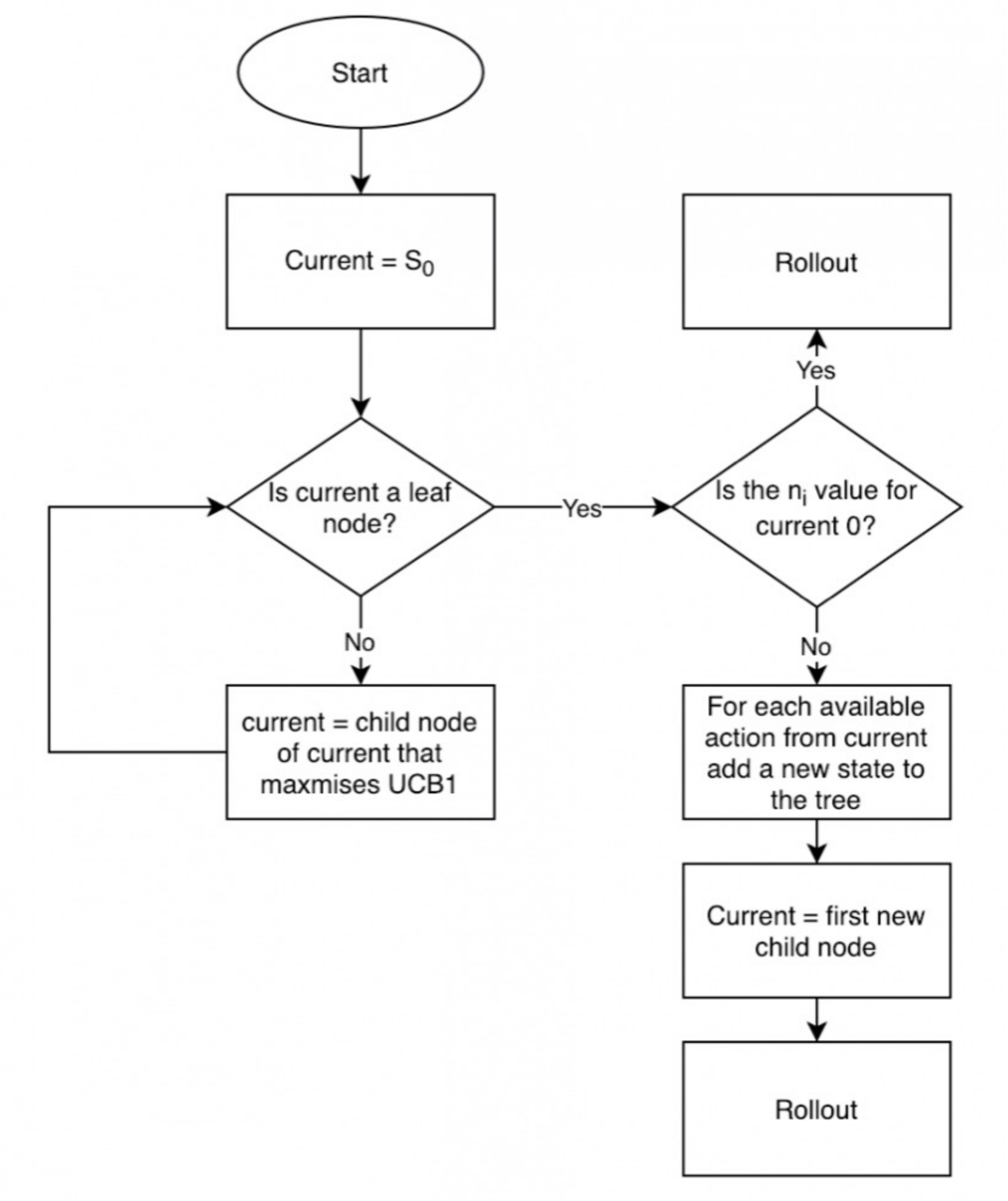
if s_t not fully expanded, choose new a_t

else choose child with best $Score(s_{t+1})$

$$Score(s_t) = \frac{Q(s_t)}{N(s_t)} + 2C\sqrt{\frac{2\ln N(s_{t-1})}{N(s_t)}}$$



Discrete Case: Monte Carlo Tree Search (cont.)



Additional Reading

- Browne, Powley, Whitehouse, Lucas, Cowling, Rohlfshagen, Tavener, Perez, Samothrakis, Colton. (2012). A Survey of Monte Carlo Tree Search Methods.
  - Survey of MCTS methods and basic summary.

Today's Lecture

1. Basics of model-based RL: learn a model, use model for control
   - ☑ Why does naïve approach not work?
   - ☑ The effect of distributional shift in model-based RL
2. Uncertainty in model-based RL
3. Model-based Policy Learning

Goals:

- ☑ Understand how to build model-based RL algorithms
- ☑ Understand the important considerations for model-based RL
- ☑ Understand the tradeoffs between different model class choices

Why learn the Model?

If we knew $f(\mathbf{s}_t, \mathbf{a}_t) = \mathbf{s}_{t+1}$, we could use the tools from last week.

(or $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ in the stochastic case)

So let's learn $f(\mathbf{s}_t, \mathbf{a}_t)$ from data, and <u>then</u> plan through it!

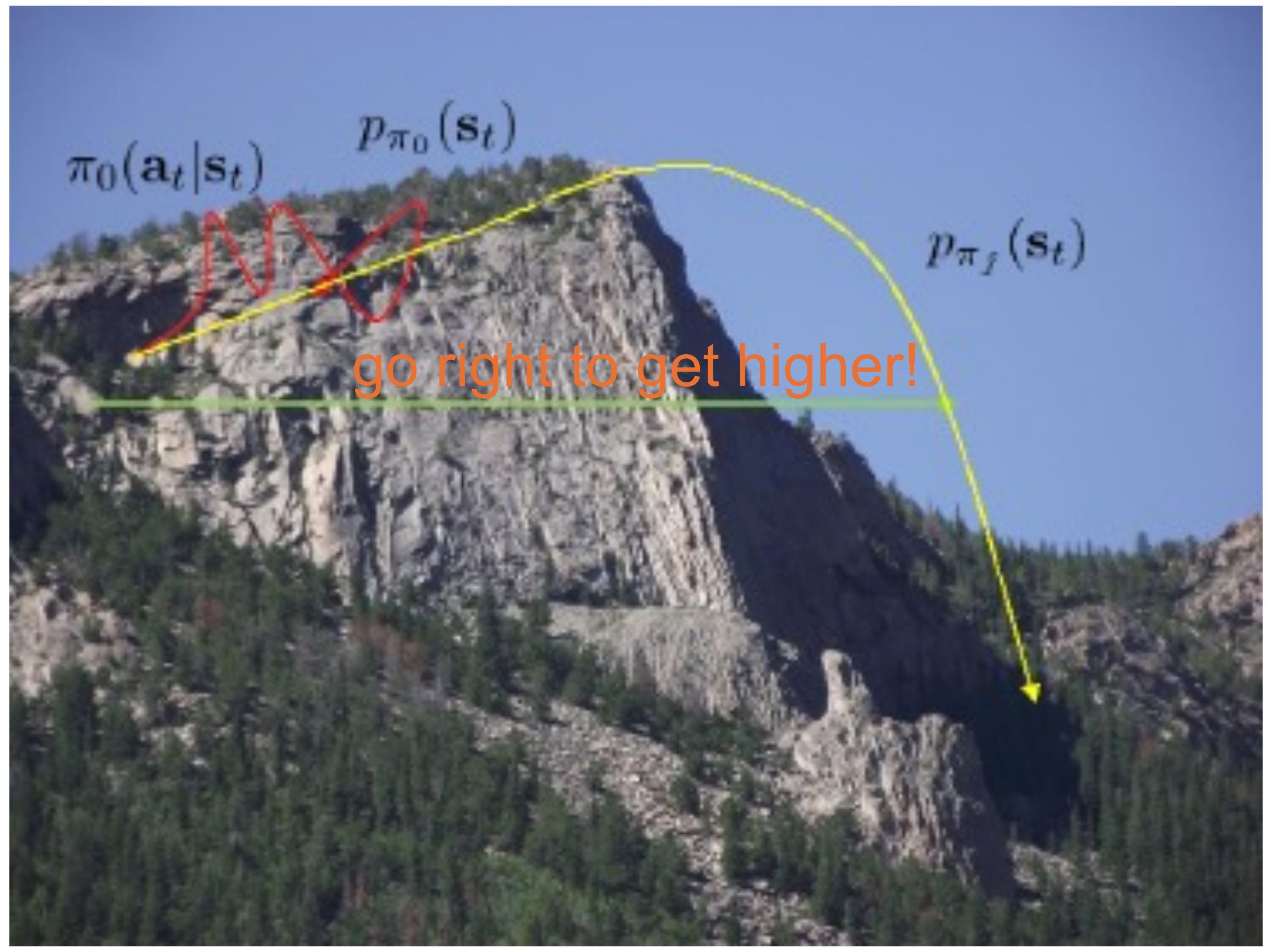
Model-Based reinforcement learning v0.5

1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$
2. learn dynamics model $f(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i \| f(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i \|^2$
3. plan through $f(\mathbf{s}, \mathbf{a})$ to choose actions

Does it Work?   YES

- Essentially how system identification works in classical robotics
- Some care should be taken to design a good base policy
- Particularly effective if we can hand-engineer a dynamics representation using our knowledge of physics, and fit just a few parameters

Does it Work? (cont.)



Model-Based reinforcement learning v0.5

1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$
2. learn dynamics model $f(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i \| f(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}_i' \|^2$
3. plan through $f(\mathbf{s}, \mathbf{a})$ to choose actions

$$P_{\pi_f}(s_t) \neq P_{\pi_0}(s_t)$$

- Distribution mismatch problem becomes exacerbated as we use more expressive model classes

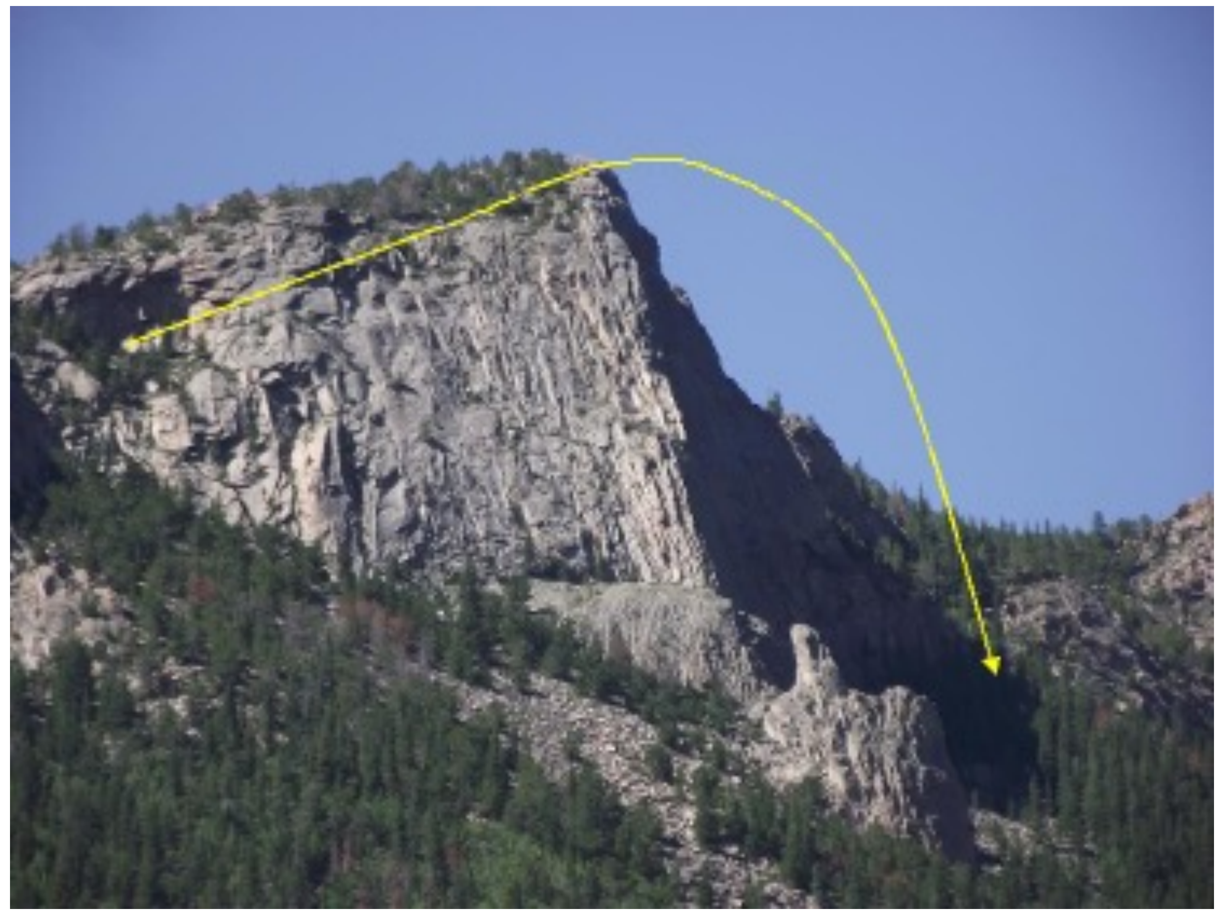
Can we do better?

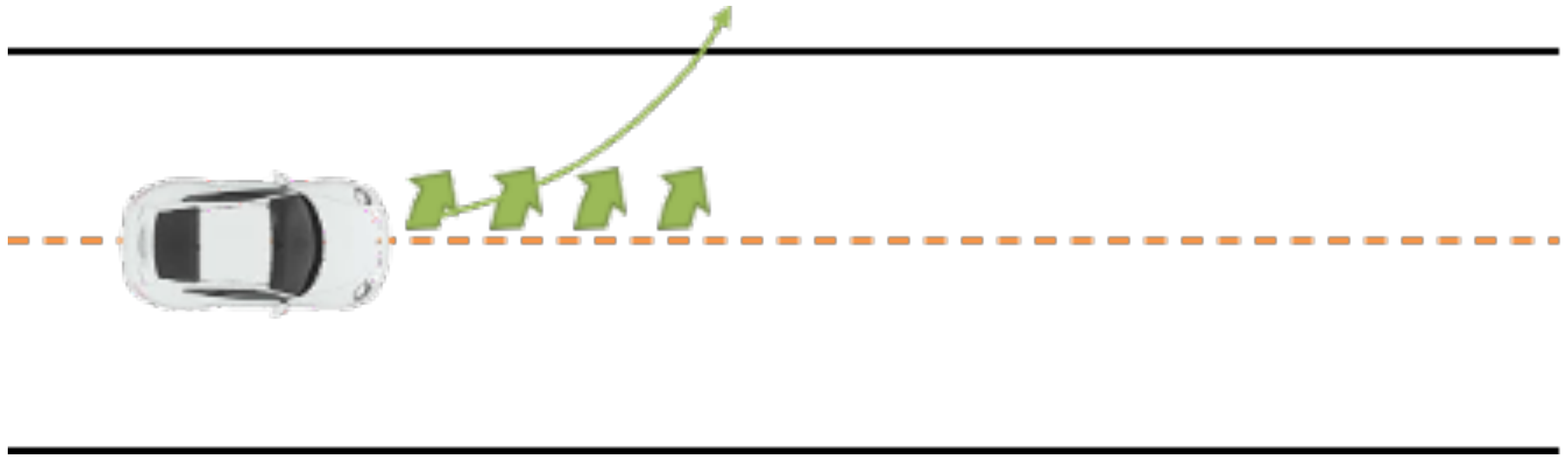
- can we make $p_{\pi_0}(\mathbf{s}_t) = p_{\pi_f}(\mathbf{s}_t)$?
- where have we seen that before? need to collect data from $p_{\pi_f}(\mathbf{s}_t)$

Model-Based reinforcement learning v1

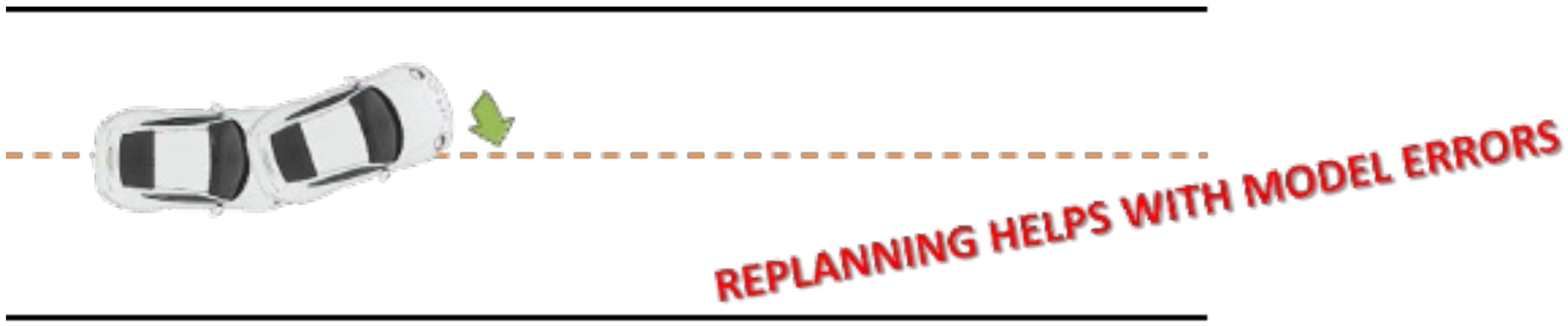
1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$
2. learn dynamics model $f(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i \| f(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i \|^2$
3. plan through $f(\mathbf{s}, \mathbf{a})$ to choose actions
4. execute those actions and add the resulting data $\{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_j\}$ to $\mathcal{D}$

What if we make a MISTAKE?





Can we do better?



Model-Based reinforcement learning v1.5

1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$
2. learn dynamics model $f(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i \| f(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}_i' \|^2$
3. plan through $f(\mathbf{s}, \mathbf{a})$ to choose actions
4. execute the first planned action, observe resulting state $\mathbf{s}'$ (MPC)
5. append $(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ to dataset $\mathcal{D}$

every N steps

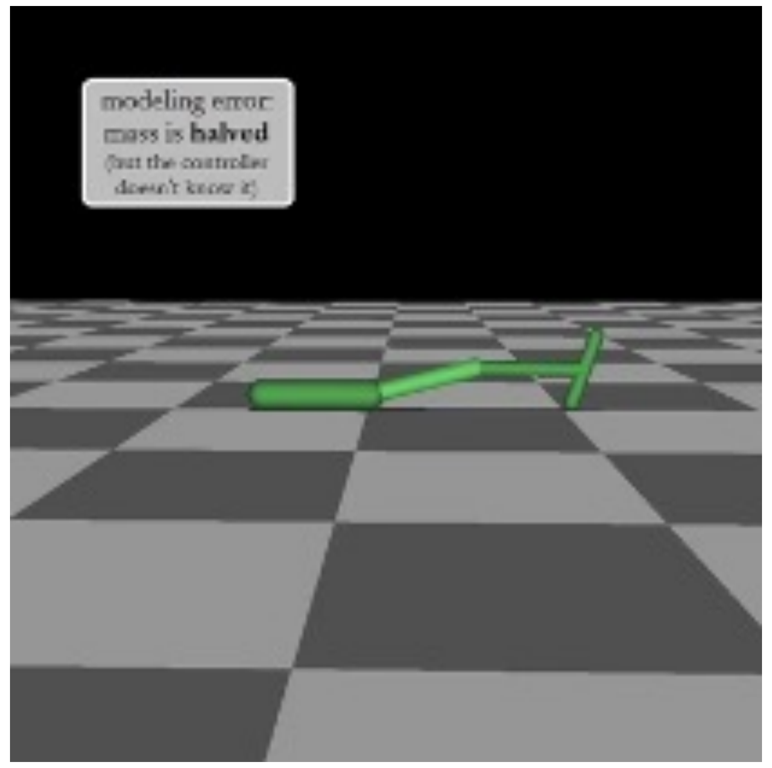
How to Replan?

Model-Based reinforcement learning v1.5

1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$
2. learn dynamics model $f(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i \| f(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i \|^2$
3. plan through $f(\mathbf{s}, \mathbf{a})$ to choose actions
4. execute the first planned action, observe resulting state $\mathbf{s}'$ (MPC)
5. append $(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ to dataset $\mathcal{D}$

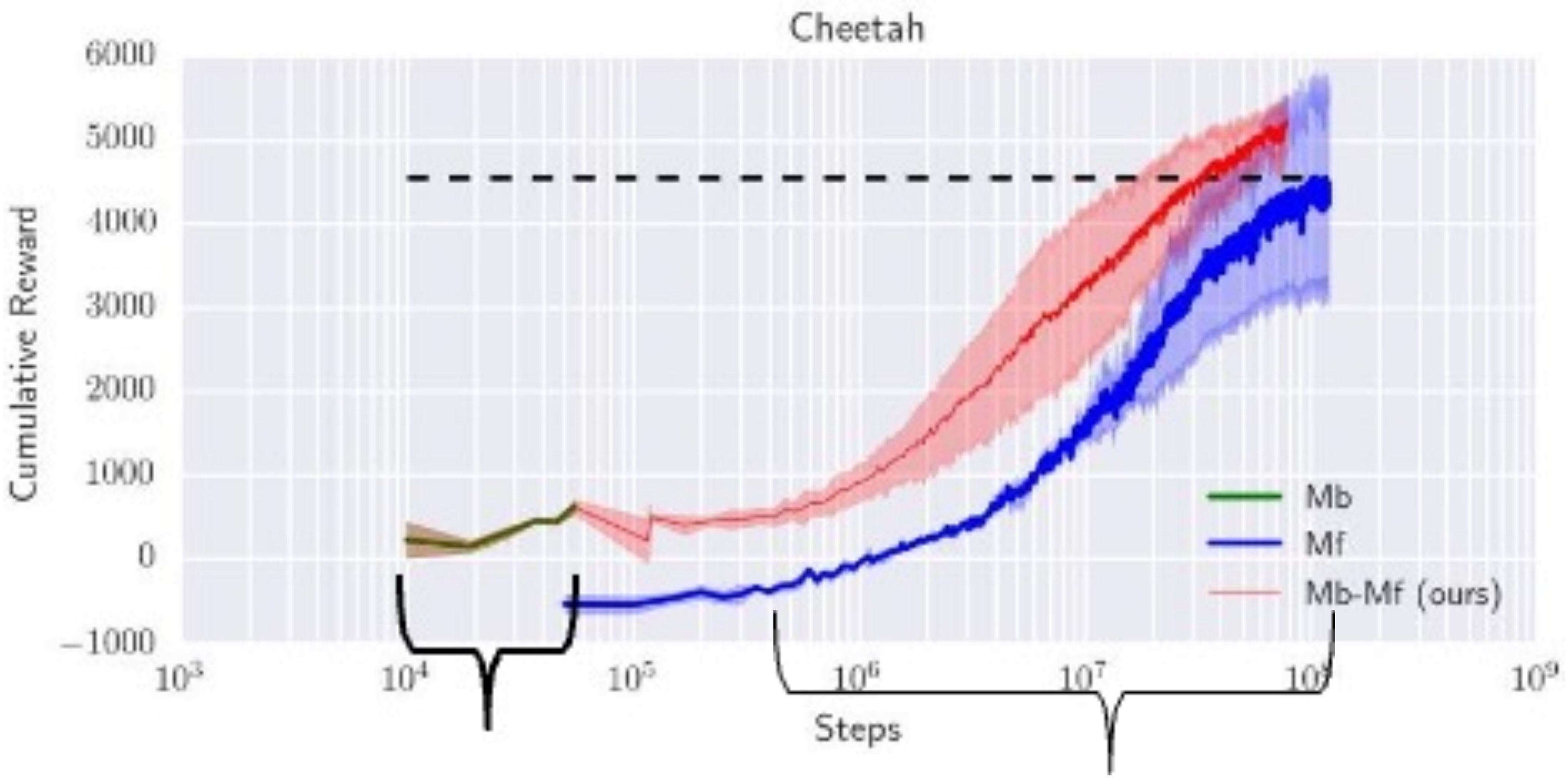
every N steps

- ✦The more you replan, the less perfect each individual plan needs to be
- ✦Can use shorter horizons
- ✦Even random sampling can often work well here!



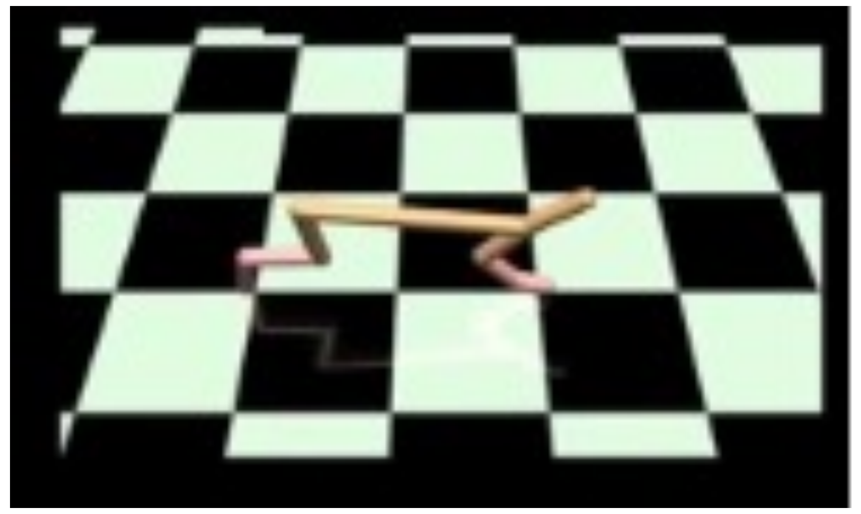
Uncertainty in Model-Based RL

A Performance gap in Model-Based RL

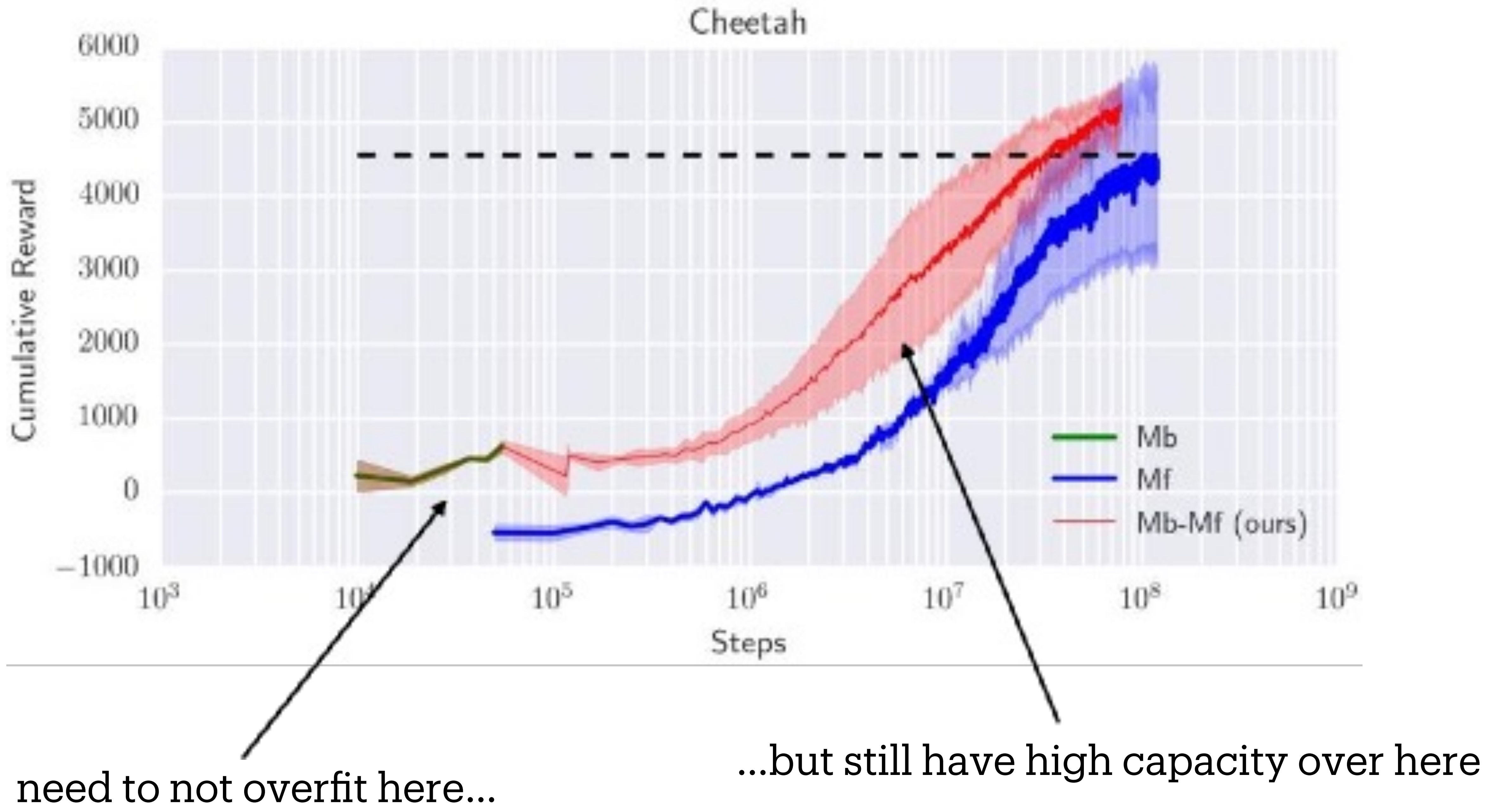


pure model-based
(about 10 minutes real time)

model-free training
(about 10 days...)



Why the Performance gap?

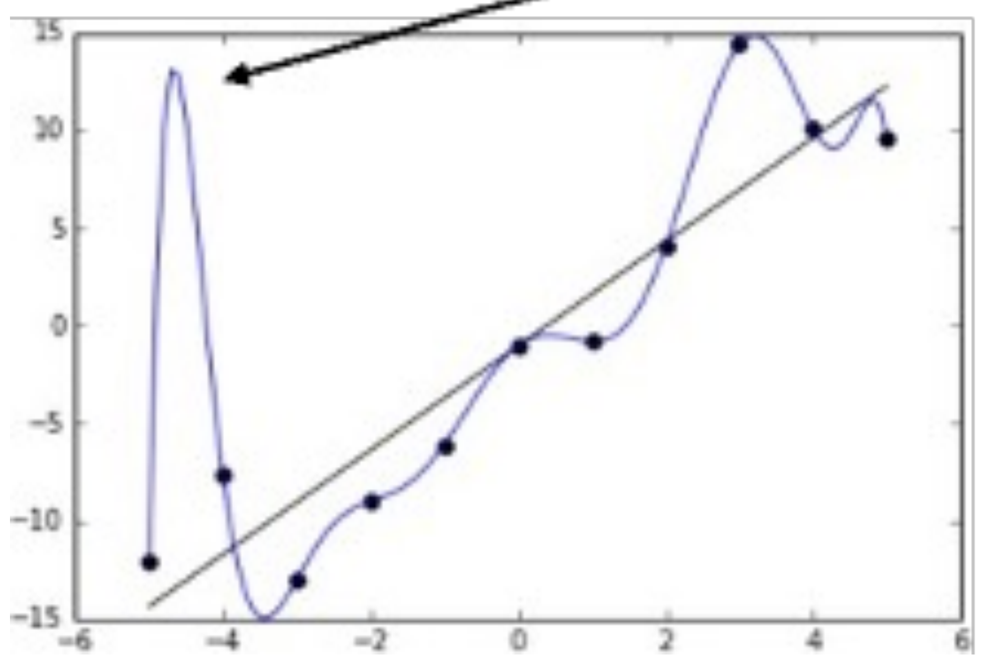


Why the Performance gap? (cont.)

Model-Based reinforcement learning v1.5

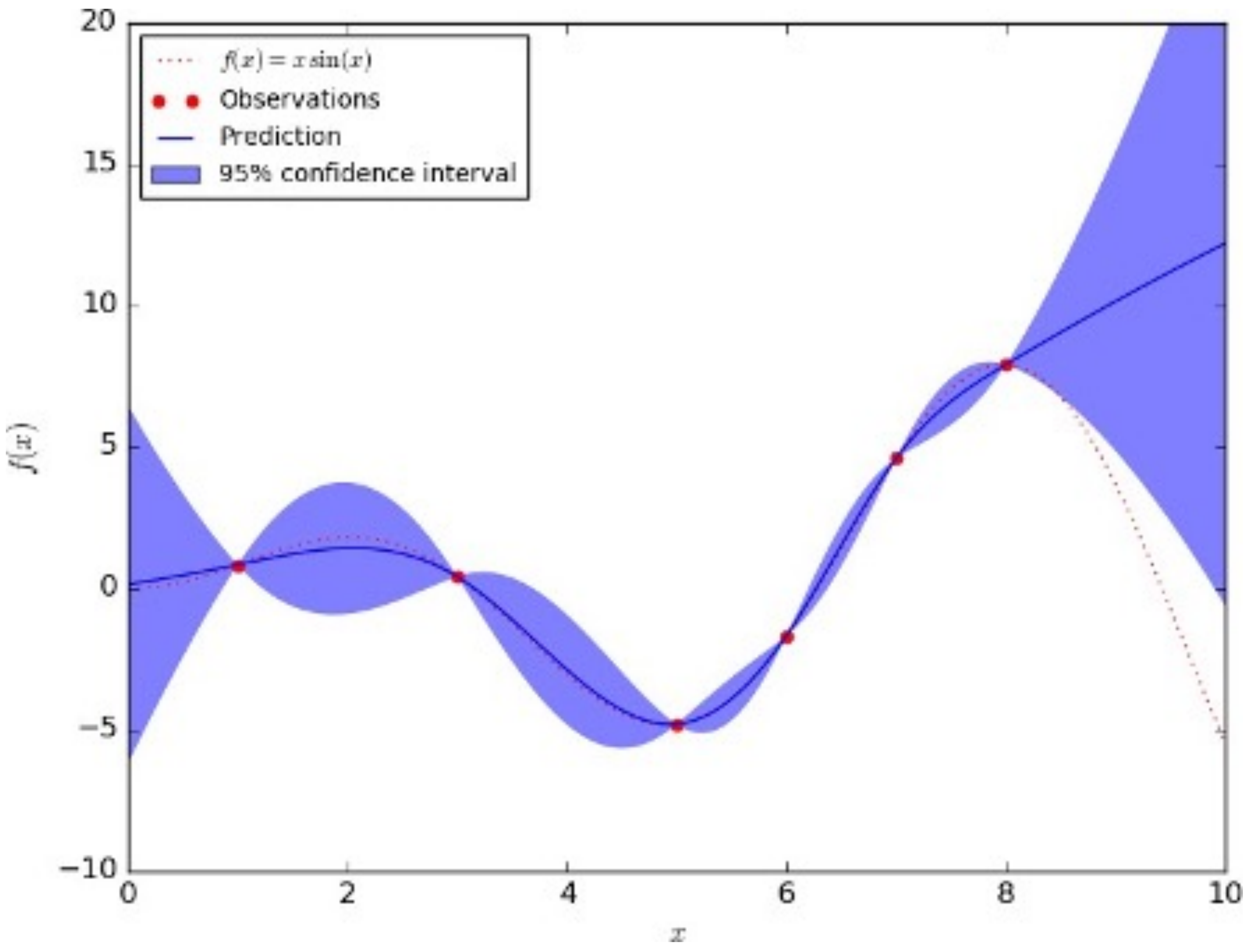
1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$
2. learn dynamics model $f(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i \| f(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i \|^2$
3. plan through $f(\mathbf{s}, \mathbf{a})$ to choose actions
4. execute the first planned action, observe resulting state $\mathbf{s}'$ (MPC)
5. append $(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ to dataset $\mathcal{D}$

every N steps



very tempting to go here...

How can uncertainty estimation help?



$$P_{\pi_f}(s_t) \neq P_{\pi_0}(s_t)$$

expected reward under high-variance prediction is very low, even though mean is the same!

33

Intuition behind uncertainty-aware RL

Model-Based reinforcement learning v1.5

1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$
2. learn dynamics model $f(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i \| f(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i \|^2$
3. plan through $f(\mathbf{s}, \mathbf{a})$ to choose actions
4. execute the first planned action, observe resulting state $\mathbf{s}'$ (MPC)
5. append $(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ to dataset $\mathcal{D}$

every N steps

only take actions for which we think we'll get high reward in expectation (w.r.t. uncertain dynamics)

This avoids "exploiting" the model

The model will then adapt and get better

There are a few caveats...

Need to explore to get better

Expected value is not the same as pessimistic value

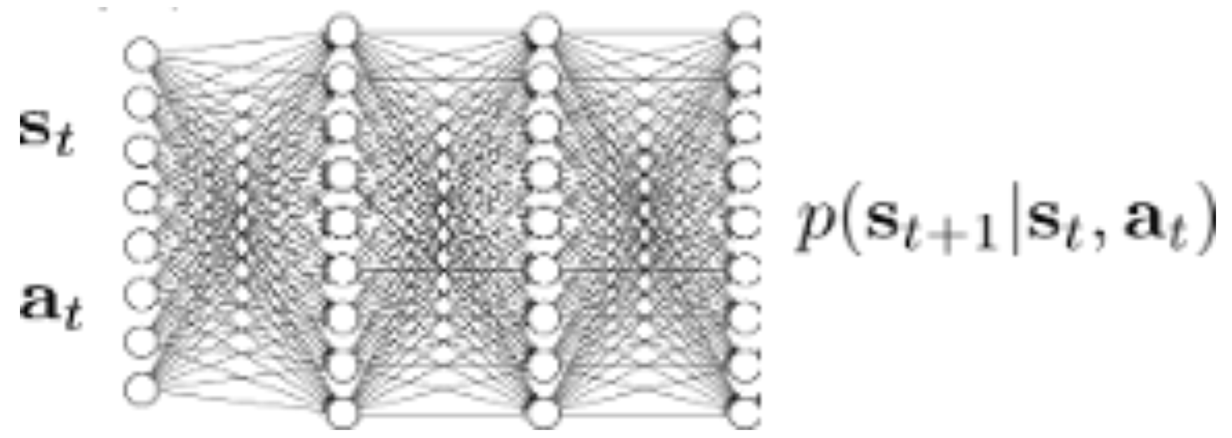
Expected value is not the same as optimistic value

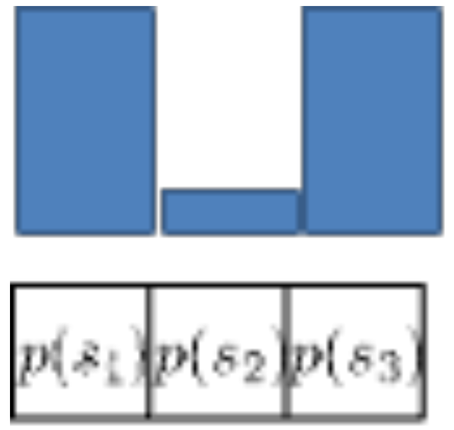
…but expected value is often a good start

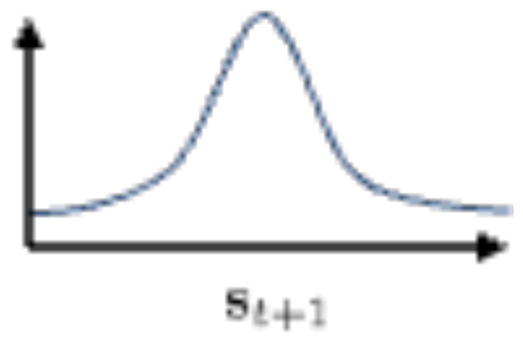
Uncertainty-Aware Neural Net Models

How can we have uncertainty-aware Models?

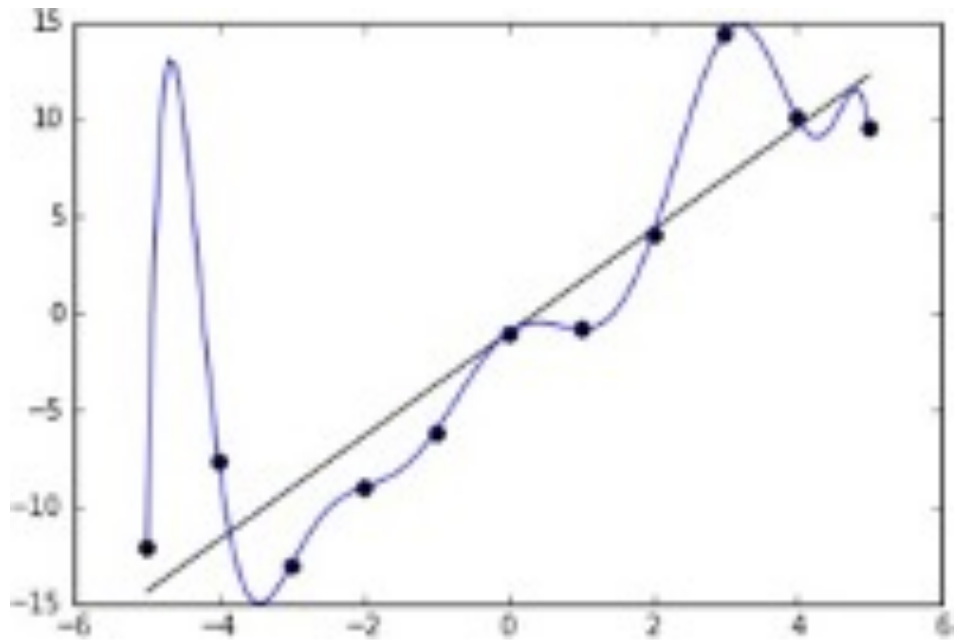
Idea 1: use output entropy







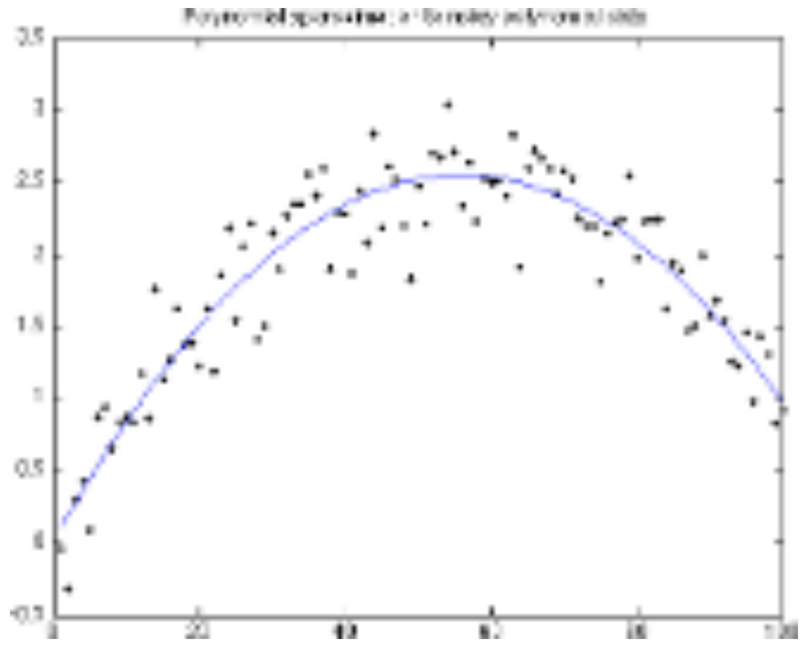
why is this not enough?



Two types of uncertainty:

aleatoric or statistical uncertainty

epistemic or model uncertainty

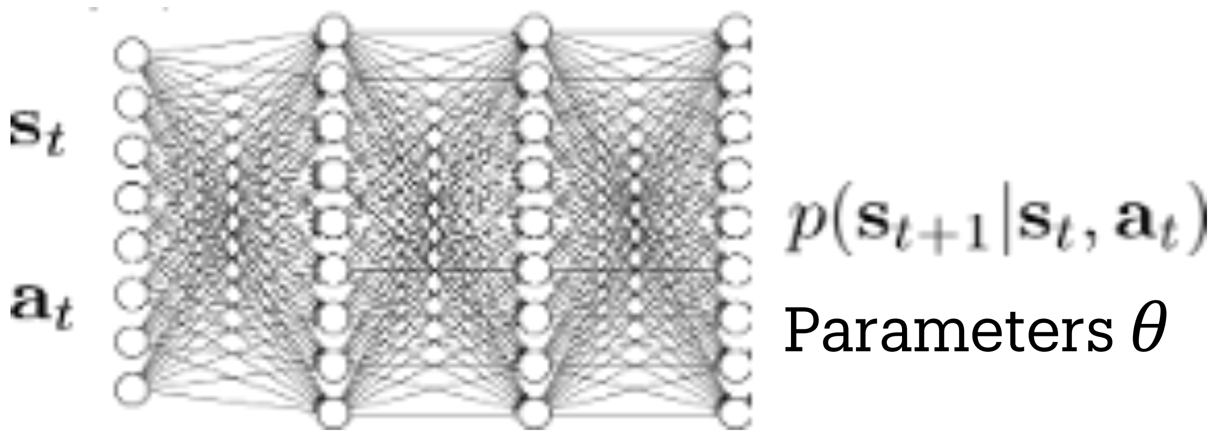


what is the variance here?

“the model is certain about the data, but we are not certain about the model”

How can we have uncertainty-aware Models? (cont.)

Idea 2: estimate model uncertainty



"the model is certain about the data, but we are not certain about the model"

Usually, we estimate

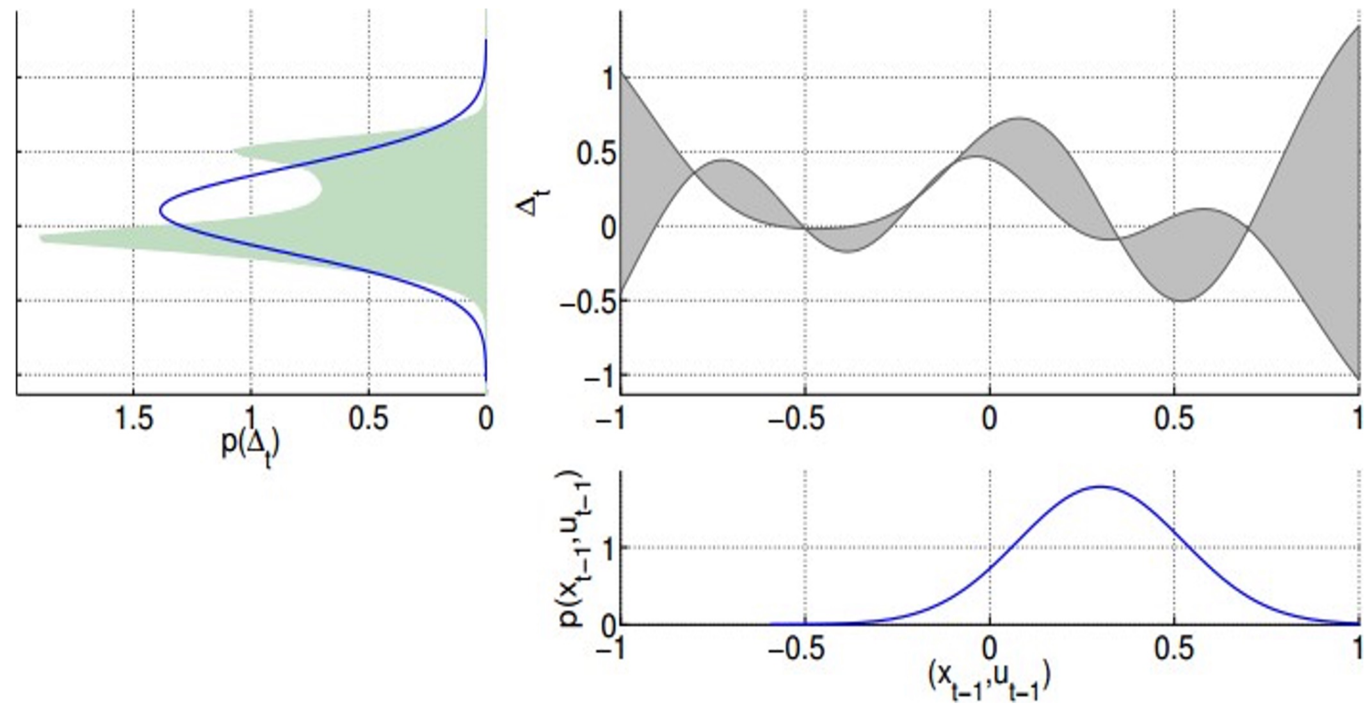
$$\arg\max_\theta \log p(\theta|\mathcal{D}) = \arg\max_\theta \log p(\mathcal{D}|\theta)$$

Can we instead estimate $P(\theta|D)$?

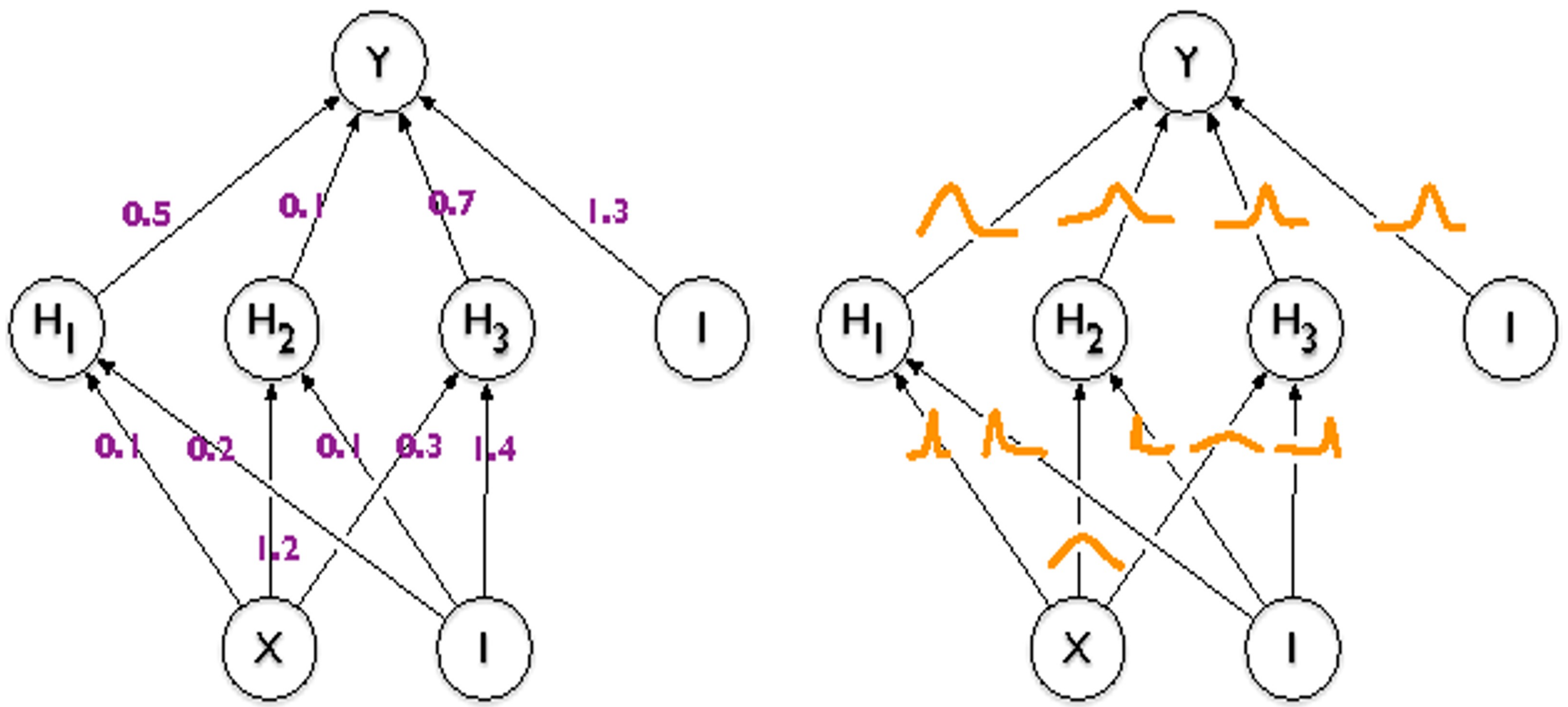
the entropy of this tells us the model uncertainty!

Predict according to:

$$\int p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \theta)p(\theta|\mathcal{D})d\theta$$



Quick overview of Bayesian Neural Networks!



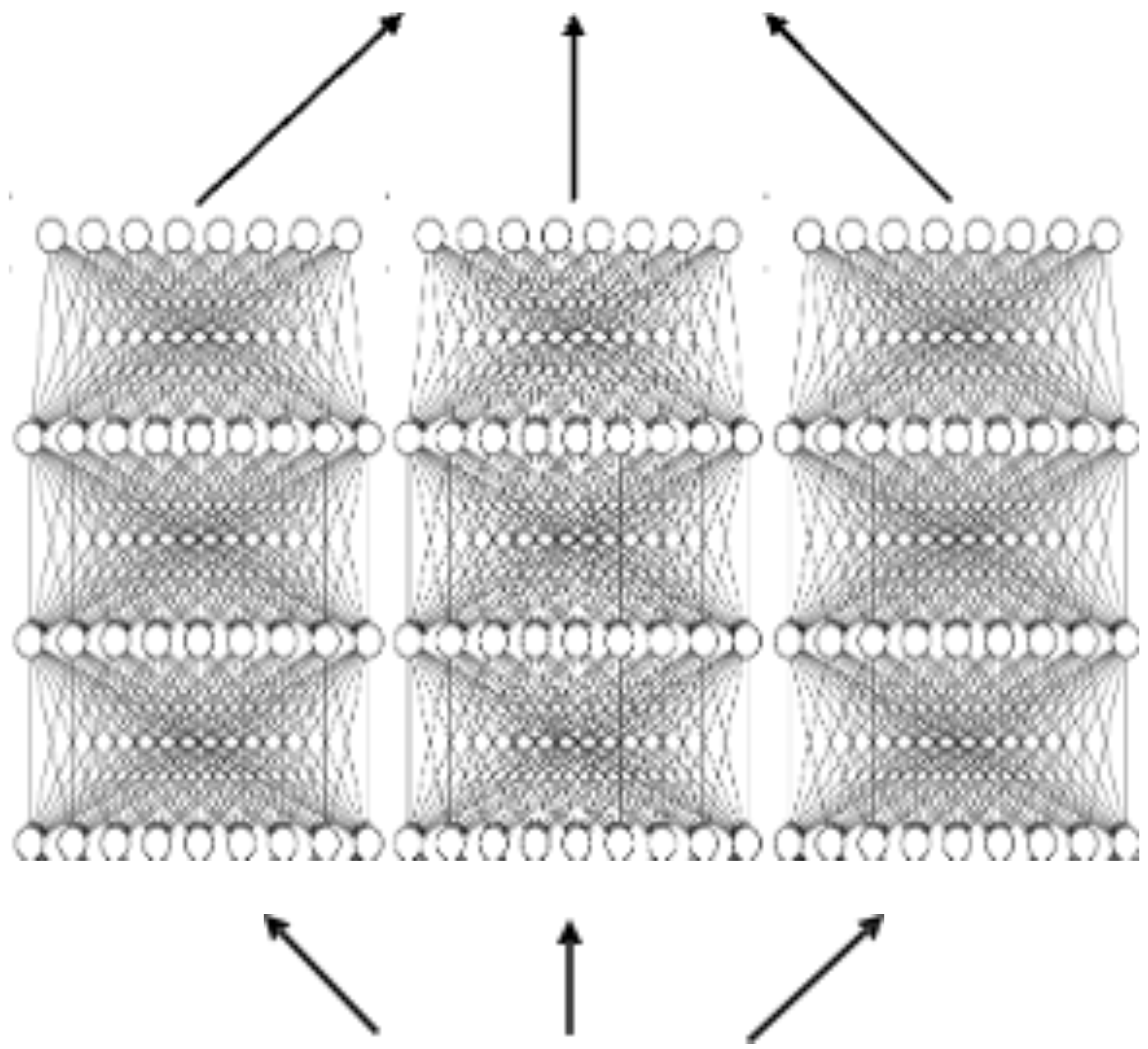
Common approximation:

$$p(\theta|\mathcal{D}) = \prod_i p(\theta_i|\mathcal{D})$$

$$p(\theta_i|\mathcal{D}) = \mathcal{N}(\mu_i, \sigma_i)$$

For more, see:
Blundell et al., Weight Uncertainty in Neural Networks Gal et al., Concrete Dropout

Bootstrap ensembles



- Train multiple models and see if they agree!

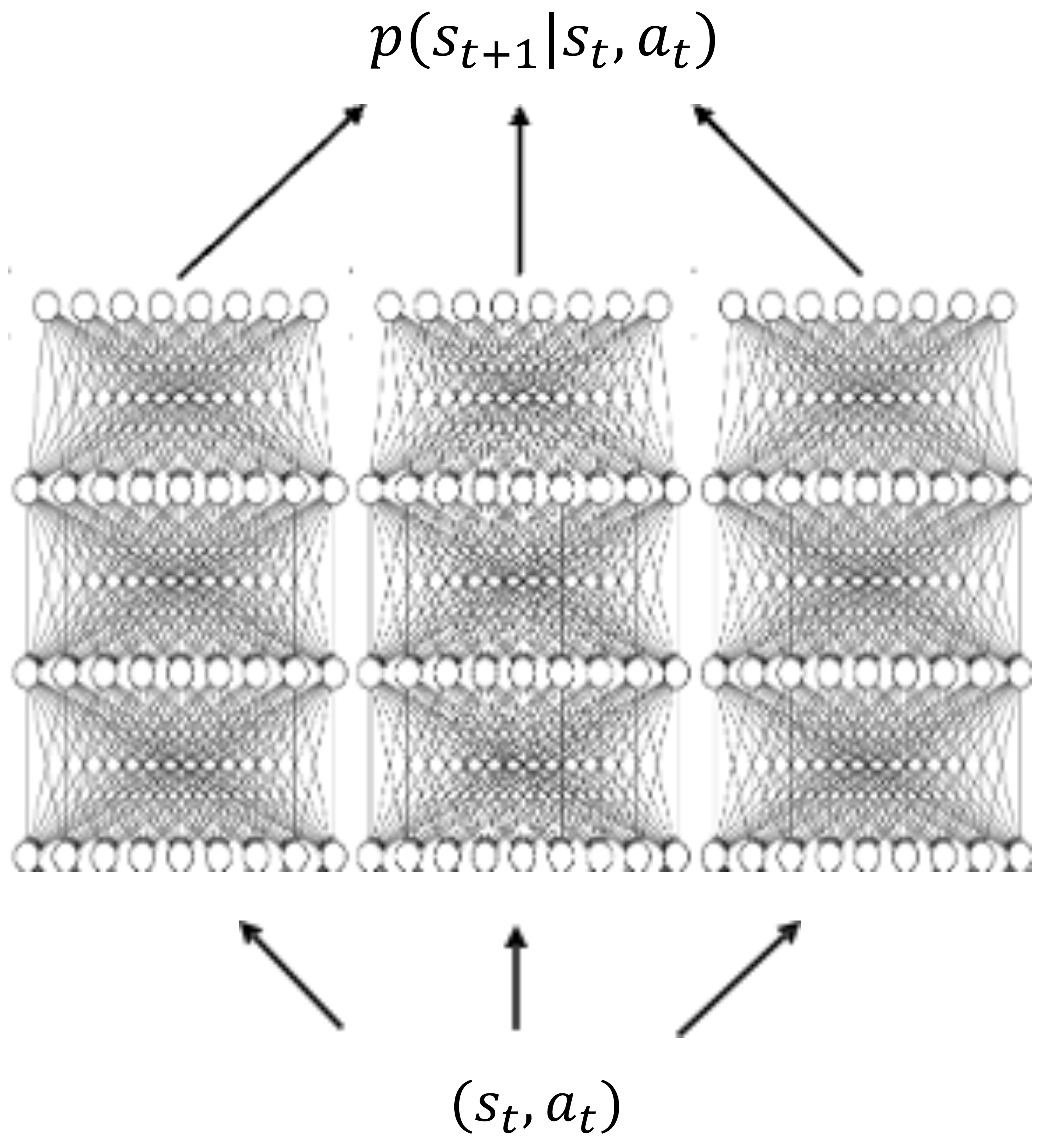
$$\text{formally: } p(\theta|\mathcal{D}) \approx \frac{1}{N}\sum_i \delta(\theta_i)$$

$$\int p(\mathbf{s_{t+1}}|\mathbf{s_t}, \mathbf{a_t}, \theta)p(\theta|\mathcal{D})d\theta \approx \frac{1}{N}\sum_i p(\mathbf{s_{t+1}}|\mathbf{s_t}, \mathbf{a_t}, \theta_i)$$

- How to train?
  - Main idea: need to generate "independent" datasets to get "independent" models

θ_i is trained on D_i, sampled with replacement from D

Bootstrap ensembles in Deep Learning



- This basically works

Very crude approximation, because the number of models is usually small (< 10)

Resampling with replacement is usually unnecessary, because SGD and random initialization usually makes the models sufficiently independent

Planning with Uncertainty, Examples

How to Plan with Uncertainty?

Before: $J(\mathbf{a_1}, \dots, \mathbf{a_H}) = \sum_{t=1}^{H} r(\mathbf{s_t}, \mathbf{a_t})$, where $\mathbf{s_{t+1}} = f(\mathbf{s_t}, \mathbf{a_t})$

Now: $J(\mathbf{a_1}, \dots, \mathbf{a_H}) = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{H} r\left(\mathbf{s_{t,i}}, \mathbf{a_t}\right)$, where $\mathbf{s_{t+1,i}} = f_i\left(\mathbf{s_{t,i}}, \mathbf{a_t}\right)$

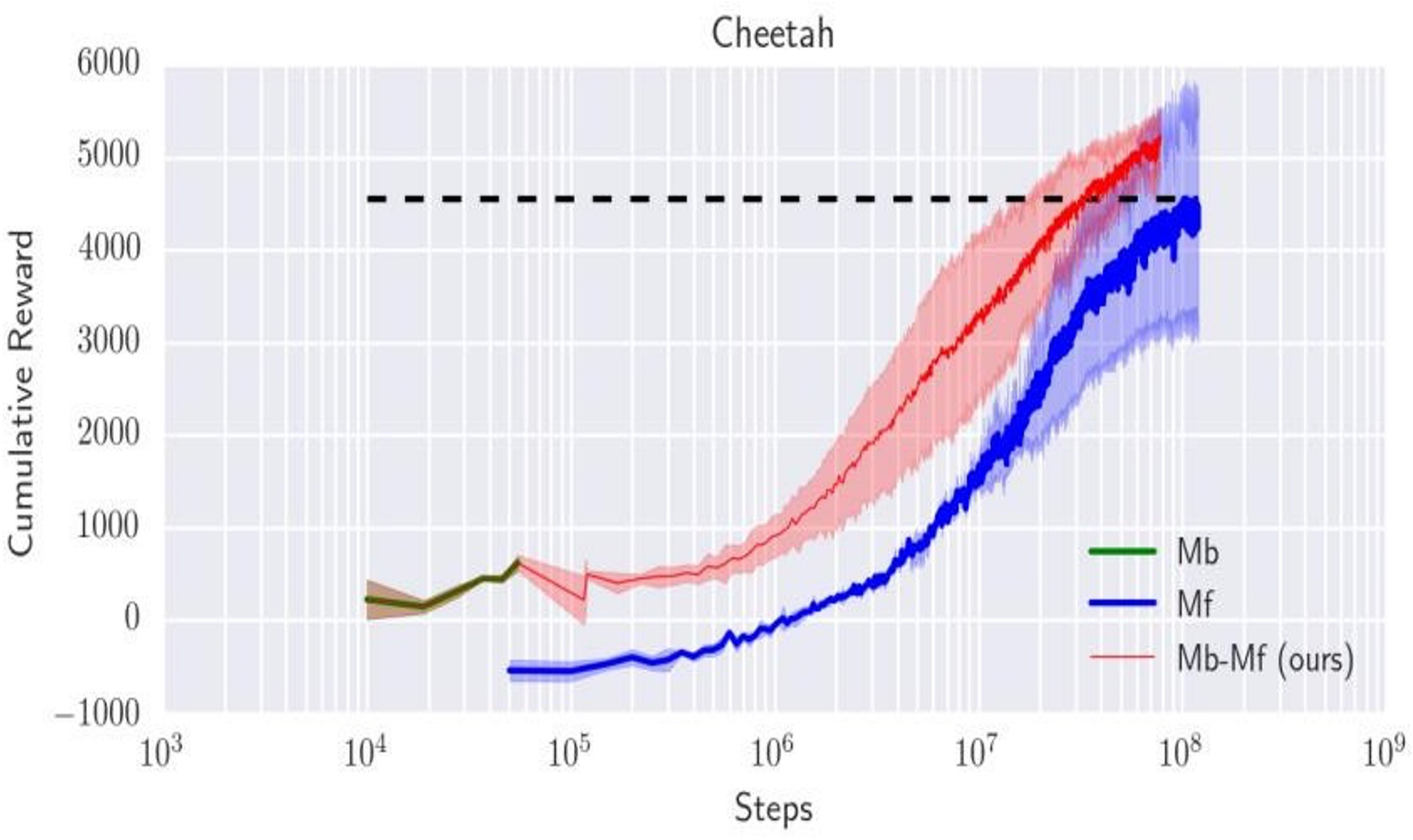
In general, for candidate action sequence $\mathbf{a}_1, \dots, \mathbf{a}_H$:

1. Step 1: sample $\theta \sim p(\theta|\mathcal{D})$
2. Step 2: at each time step t, sample $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \theta)$
3. Step 3: calculate $R = \sum_t r(\mathbf{s}_t, \mathbf{a}_t)$
4. Step 4: repeat steps 1 to 3 and accumulate the average reward

Other options: moment matching, more complex posterior estimation with BNNs, etc.

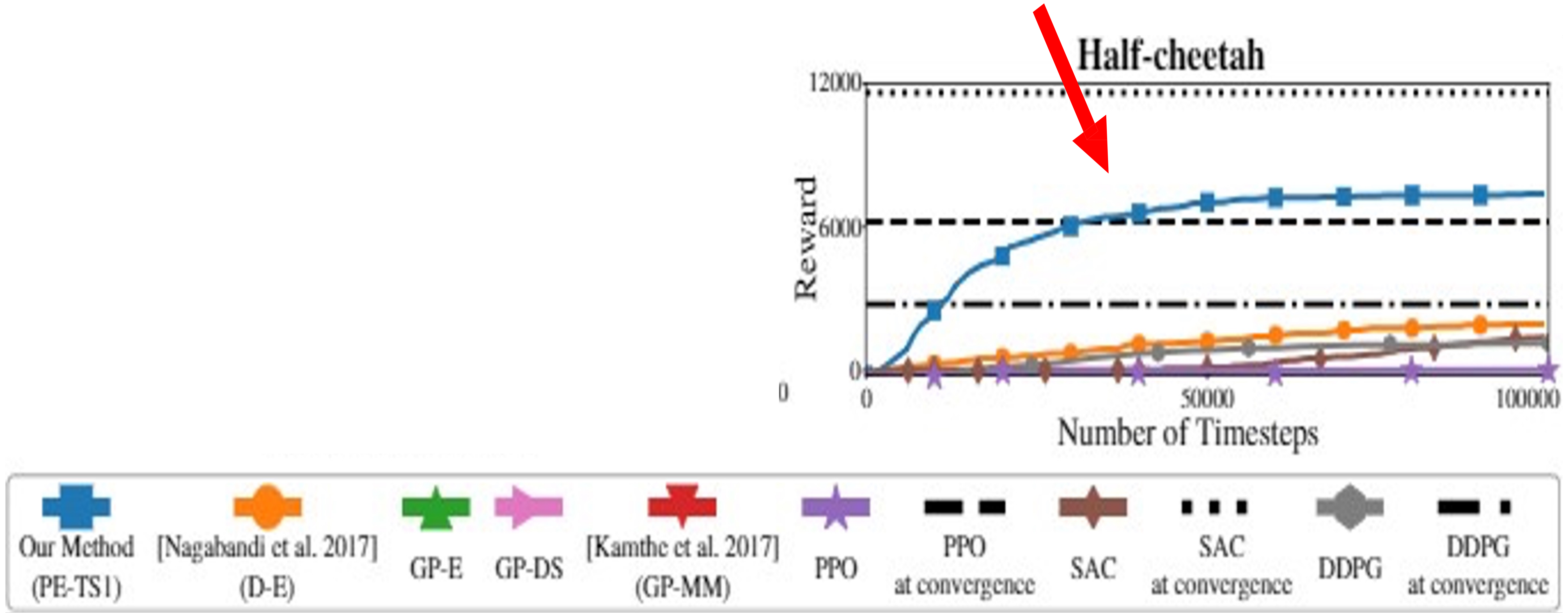
Example: Model-Based RL with ensemble!

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



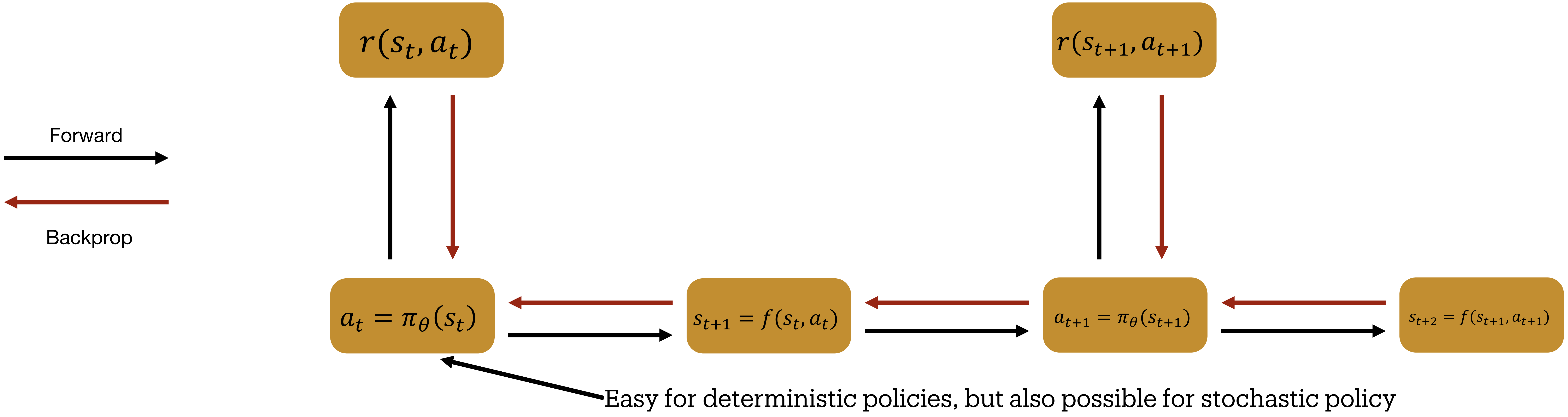
before

exceeds performance of model-free after 40k steps (about 10 minutes of real time)



after

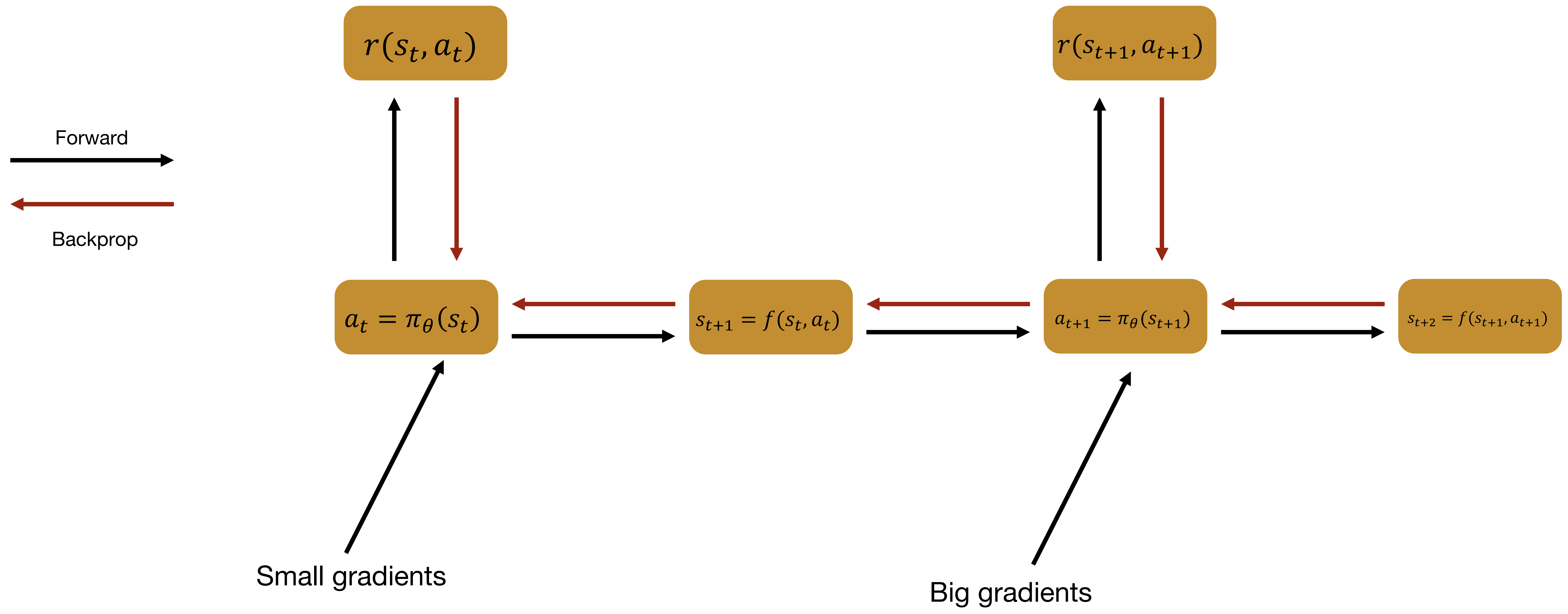
Backpropagate directly into the policy?



Model-Based reinforcement learning v2

1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$
2. learn dynamics model $f(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i \| f(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i \|^2$
3. backpropagate through $f(\mathbf{s}, \mathbf{a})$ into the policy to optimize $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$
4. run $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$, appending the visited tuples $(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ to $\mathcal{D}$

What's the problem with backprop into Policy?



What's the problem with backprop into Policy? (cont.)

- Similar parameter sensitivity problems as shooting methods
  - But no longer have convenient second order LQR-like method, because policy parameters couple all the time steps, so no dynamic programming

- Similar problems to training long RNNs with BPTT
  - Vanishing and exploding gradients
  - Unlike LSTM, we can't just "choose" a simple dynamics, dynamics are chosen by nature

What's the problem with backprop into Policy? (cont.)

- Use derivative-free ("model-free") RL algorithms, with the model used to generate synthetic samples
- Seems weirdly backwards
  - Actually works very well
  - Essentially "model-based acceleration" for model-free RL

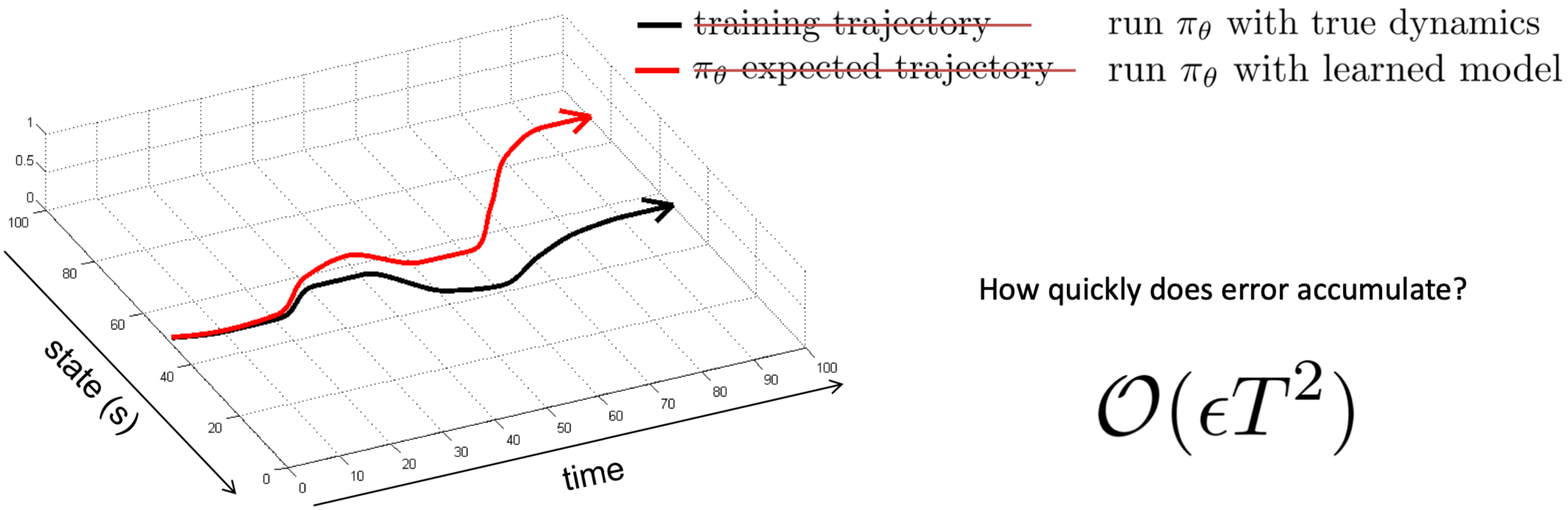
Model-Based RL with policy gradinet

Model-Based reinforcement learning v2.5

1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$
2. learn dynamics model $f(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i \| f(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i \|^2$
3. use $f(\mathbf{s}, \mathbf{a})$ to generate trajectories $\{\tau_i\}$ with policy $\pi_\theta(\mathbf{a}|\mathbf{s})$
4. use $\{\tau_i\}$ to improve $\pi_\theta(\mathbf{a}|\mathbf{s})$ via policy gradient
5. run $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$, appending the visited tuples $(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ to $\mathcal{D}$

What's a potential problem with this approach?

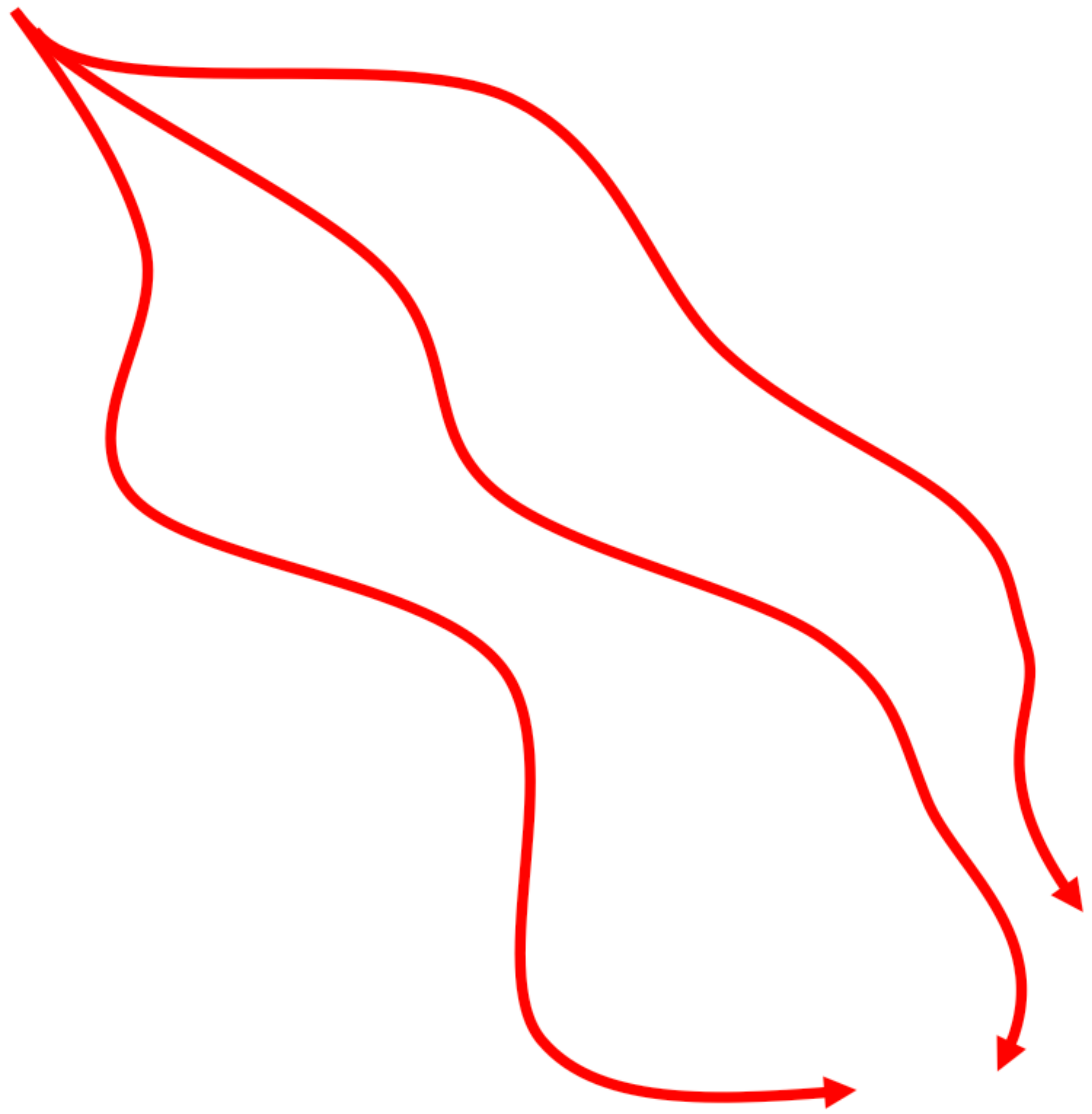
The curse of long Model-Based rollouts



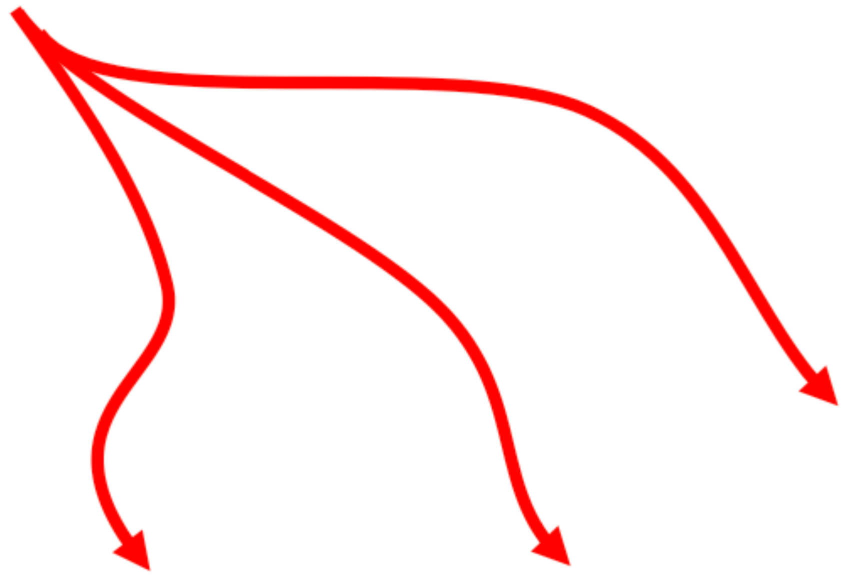
How quickly does error accumulate?

$$\mathcal{O}(\epsilon T^2)$$

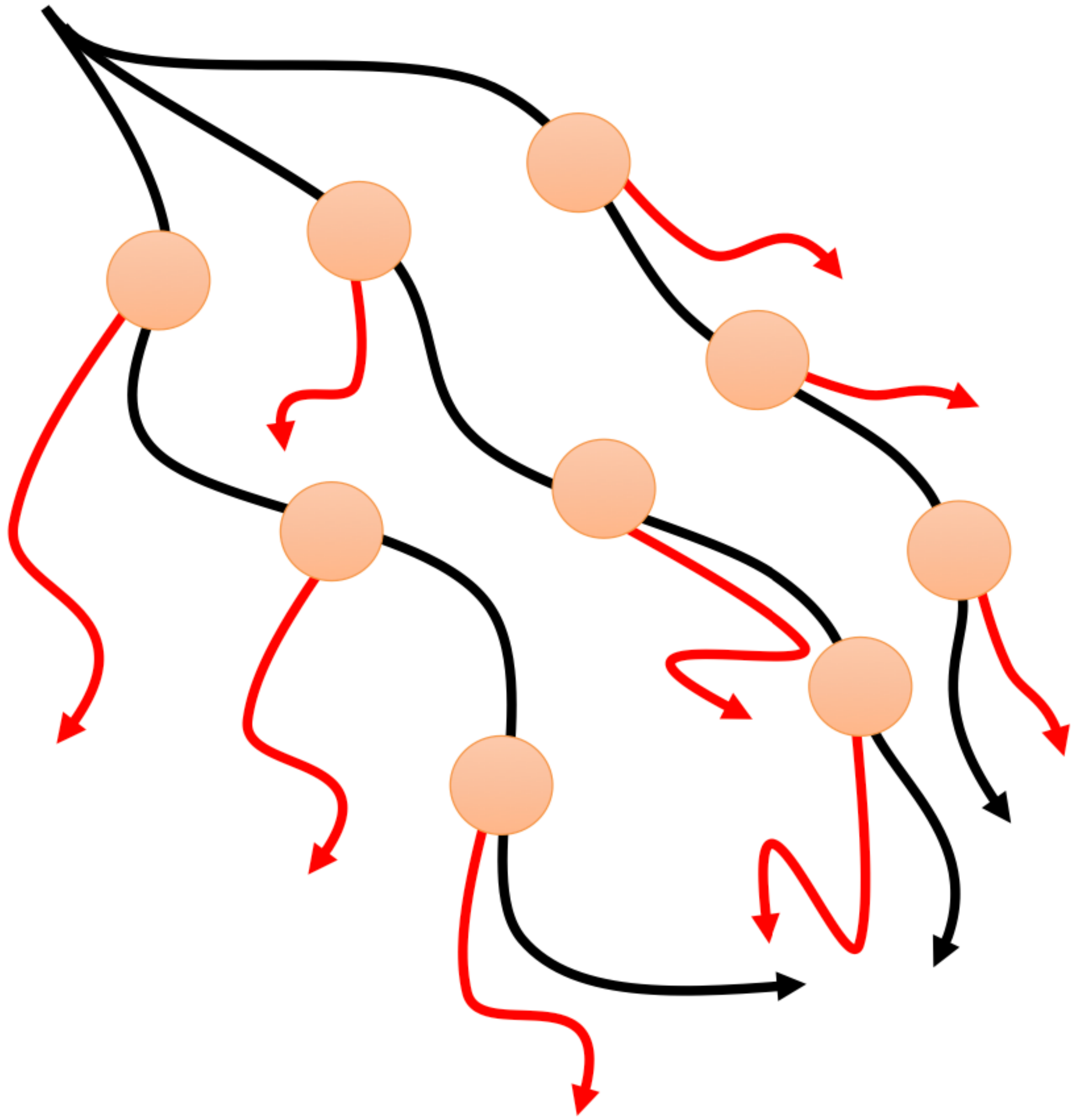
How to get away with accumulated errors?



- huge accumulating error



+ much lower error
- never see later time steps



+ much lower error
+ see all time steps
- wrong state distribution

Model-Based RL with short rollouts

Model-Based reinforcement learning v3

1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$
2. learn dynamics model $f(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i \| f(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i \|^2$
3. pick states $\mathbf{s}_i$ from $\mathcal{D}$, use $f(\mathbf{s}, \mathbf{a})$ to make short rollouts from them
4. use both real and model data to improve $\pi_\theta(\mathbf{a}|\mathbf{s})$ with off-policy RL
5. run $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$, appending the visited tuples $(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ to $\mathcal{D}$

Dyna

Dyna-Q Algorithm

1. given state s, pick action a using exploration policy
2. observe s' and r, to get transition (s, a, s', r)
3. update model $\hat{p}(s'|s, a)$ and $\hat{r}(s, a)$ using (s, a, s')
4. Q-update: $Q(s, a) \leftarrow Q(s, a) + \alpha E_{s',r}[r + max_{a'} Q(s', a') - Q(s, a)]$
5. repeat K times:
   6. sample $(s, a) \sim \mathcal{B}$ from buffer of past states and actions
   7. Q-update: $Q(s, a) \leftarrow Q(s, a) + \alpha E_{s',r}[r + max_{a'} Q(s', a') - Q(s, a)]$

Richard S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming.

Dyna (cont.)

Dyna-Q Algorithm - Direct RL & Planning

Loop forever:

(a) $S \leftarrow$ *current (nonterminal) state*

(b) $A \leftarrow \epsilon\text{–}greedy(S, Q)$

(c) *Take action A; observe resultant reward, R, and state, S'*

(d) $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma max_a Q(S', a) - Q(S, A)]$

(e) $Model(S, A) \leftarrow R, S'$ *(assuming deterministic environment)*

(f) *Loop repeat k times:*

$S \leftarrow$ *random previously observed state*

$A \leftarrow$ *random action previously taken in S*

$R, S' \leftarrow Model(S, A)$

$Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma max_a Q(S', a) - Q(S, A)]$

General "Dyna-Style" Model-Based RL recipe

Dyna Algorithm

1. collect some data, consisting of transitions (s, a, s', r)
2. learn model $\hat{p}(s'|s, a)$ (and optionally, $\hat{r}(s, a)$)
3. repeat K times:
    4. sample $s \sim \mathcal{B}$ from buffer
    5. choose action a (from $\mathcal{B}$, from π, or random)
    6. simulate $s' \sim \hat{p}(s'|s, a)$ (and $r = \hat{r}(s, a)$)
    7. train on (s, a, s', r) with model-free RL
    8. (optional) take N more model-based steps

\+ only requires short (as few as one step) rollouts from model

\+ still sees diverse states

Instantiations

1. Model-Based Acceleration (MBA)
2. Model-Based Value Expansion (MVE)
3. Model-Based Policy Optimization (MBPO)

Model-Based Rollout Q-Learning

1. take some action $\mathbf{a}_i$ and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$, add it to $\mathcal{B}$
2. sample mini-batch $\{\mathbf{s}_j, \mathbf{a}_j, \mathbf{s}'_j, r_j\}$ from $\mathcal{B}$ uniformly
3. use $\{\mathbf{s}_j, \mathbf{a}_j, \mathbf{s}'_j\}$ to update model $\hat{p}(\mathbf{s}'|\mathbf{s}, \mathbf{a})$
4. sample $\{\mathbf{s}_j\}$ from $\mathcal{B}$
5. for each $\mathbf{s}_j$, perform model-based rollout with $\mathbf{a} = \pi(\mathbf{s})$
6. use all transitions $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$ along rollout to update Q-function

+ why is this a good idea?

- why is this a bad idea?

Gu et al. Continuous deep Q-learning with model-based acceleration. ‘16

Feinberg et al. Model-based value expansion. ‘18

Janner et al. When to trust your model: model-based policy optimization. ‘19