Reinforcement Learning: Advanced Model-Based RL (ESE 3990)

Instructor:
Antonio Loquercio

Fall 2026

University of Pennsylvania

Categorizing RL agents

Axis 1: How to make decisions

- Value Based
  - No Policy (Implicit)
  - Value/Q Function
- Policy Based
  - Policy
  - No Value/Q Function
- Actor Critic
  - Policy
  - Value Function

Axis 2: Environment Representation

- Model-based — Today
  - Policy and/or Value/Q Function
  - Model
- Model-free
  - Policy and/or Value/Q Function
  - No Model

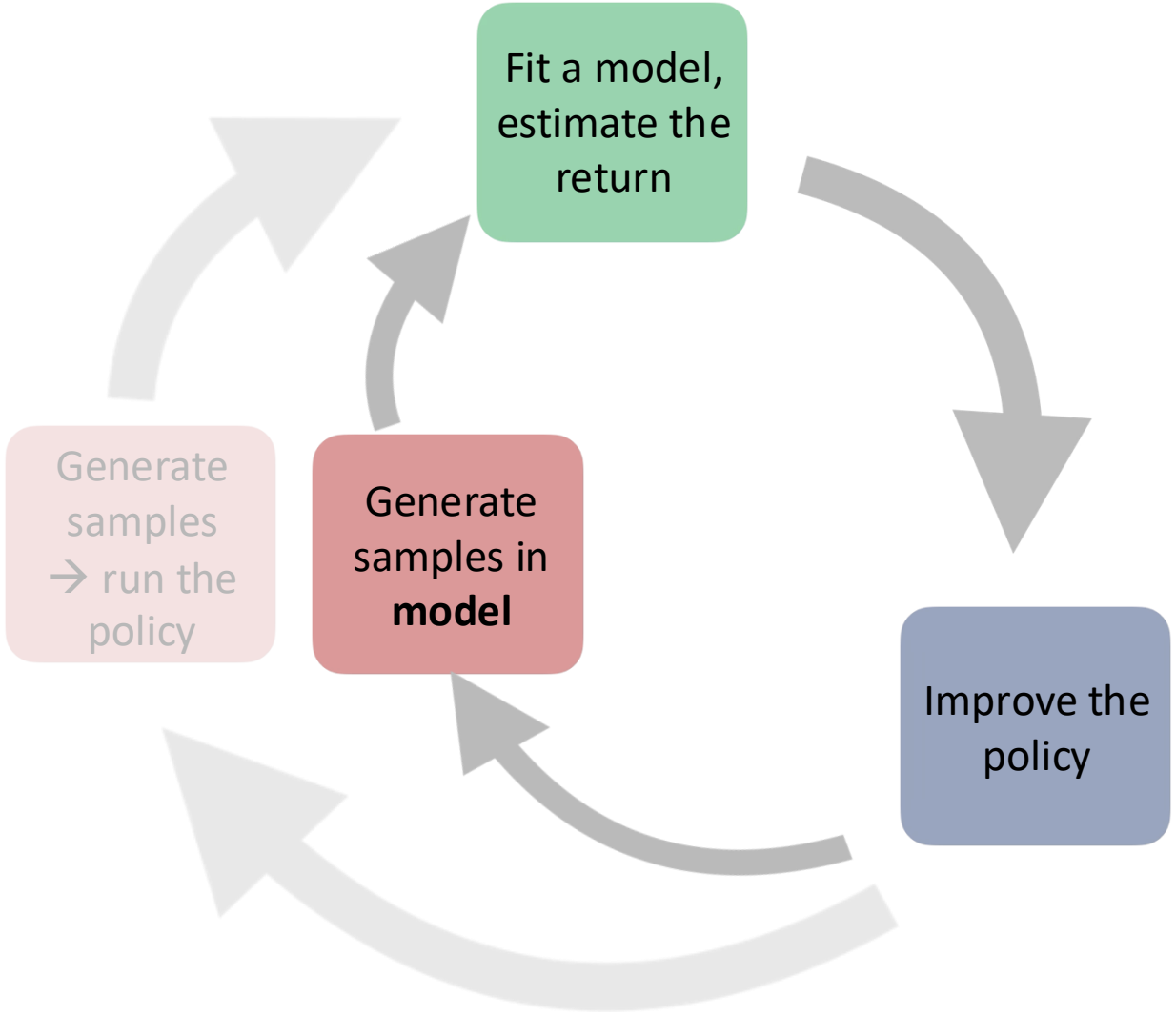
Today’s Agenda

- Understanding the landscape of model-based RL methods
- Ideas to address model inaccuracies
- Understand models that operate in the latent space

Why Model-Based?

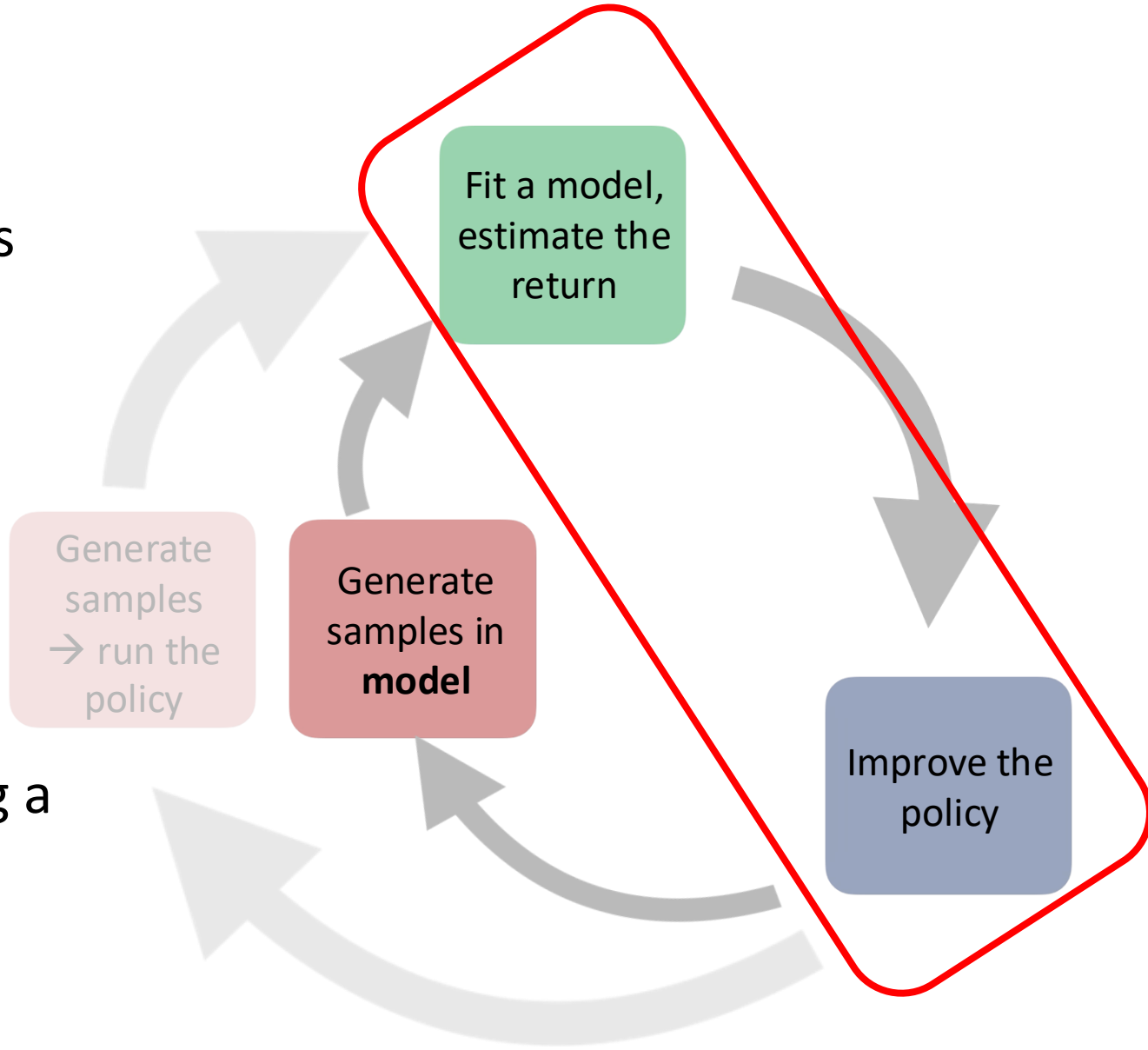
- Generate samples in a “model” of the world.

- The model is a “simulator” of the environment:
  - Predicts the next state given the current state.
  - Predicts the reward given a state.



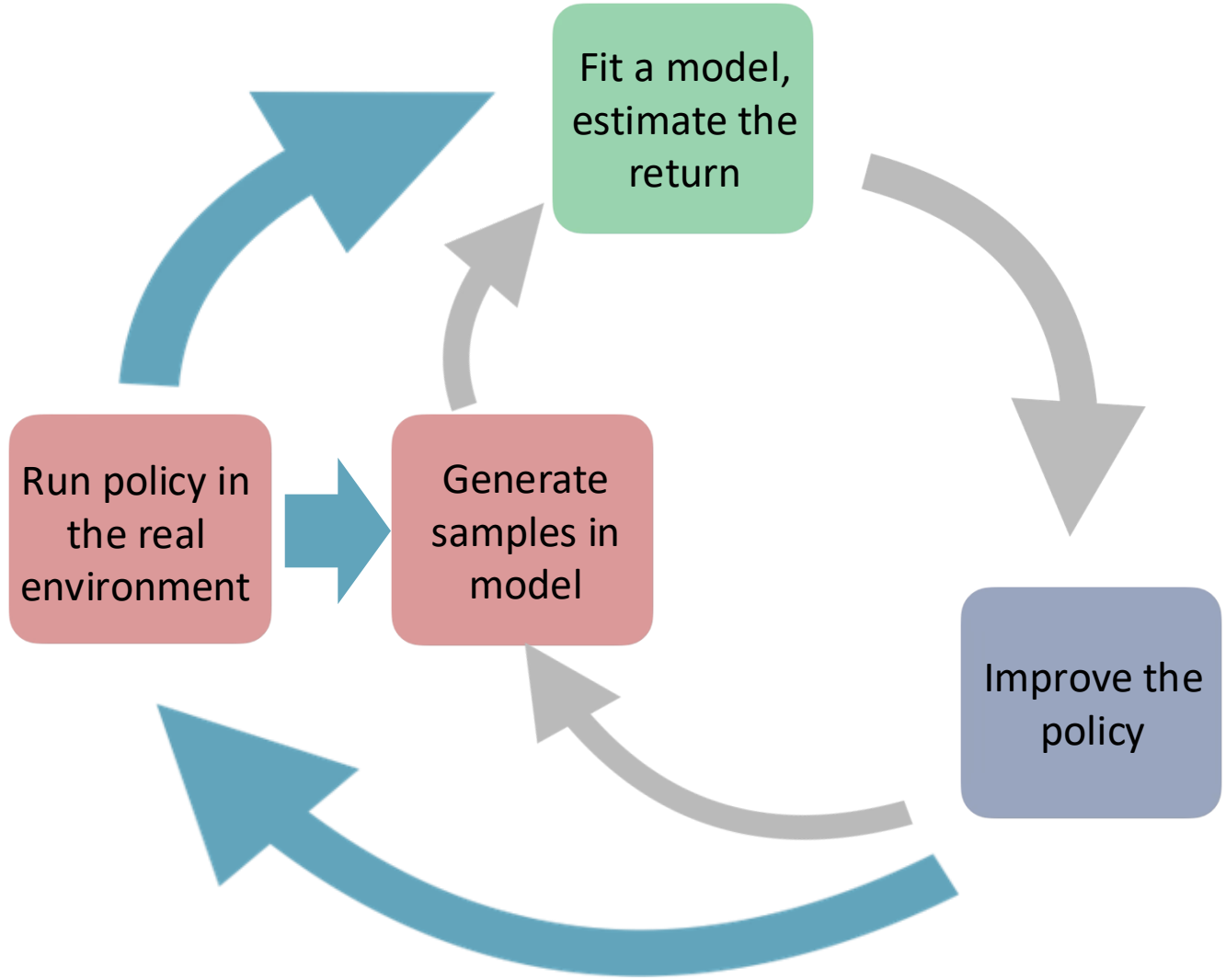
Why Model-Based?

- I can use any of the prior algorithms for policy optimization.
  - Dynamic Programming
  - Model-Free Control
    - With Q-functions
    - Policy-based
    - Actor Critic
- We define the process of evaluating a policy or finding V^*/π^* only using a model *planning*.



Integrated Planning and Learning

- We occasionally run the policy in the real environment:
  - To improve the model.
  - To directly improve the Q function and/or the policy.
- Often called *Dyna (Sutton, '91)*



Model-Based vs Model-Free RL

- Advantages of Model-based:
  - Generally, more sample-efficient than model-free
  - Since we squeeze more out of each sample

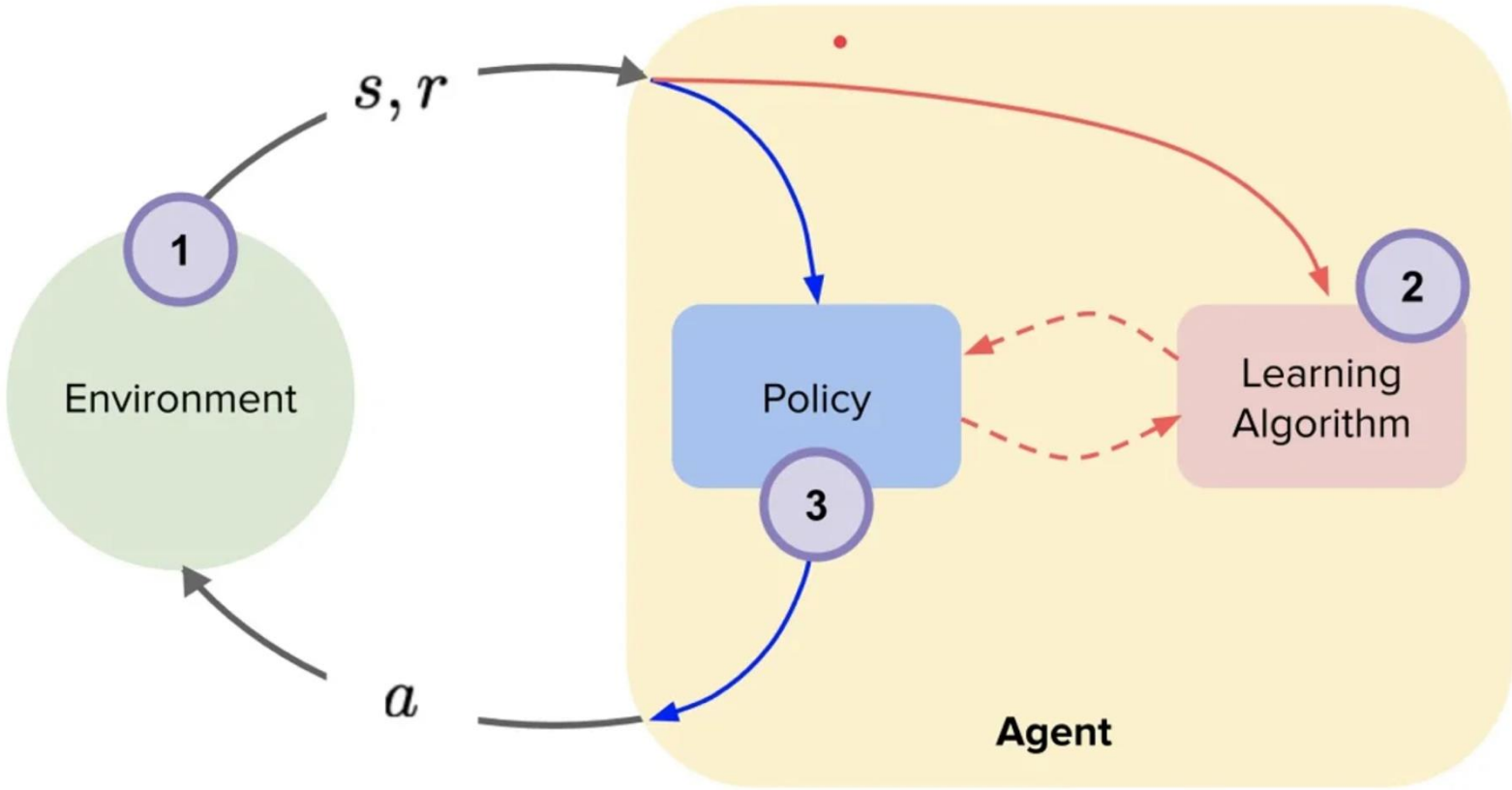
- Disadvantages of Model-based:
  - Compounding errors:
  - In the model
  - In the value function/policy

How to use a model

1. Simulating the environment

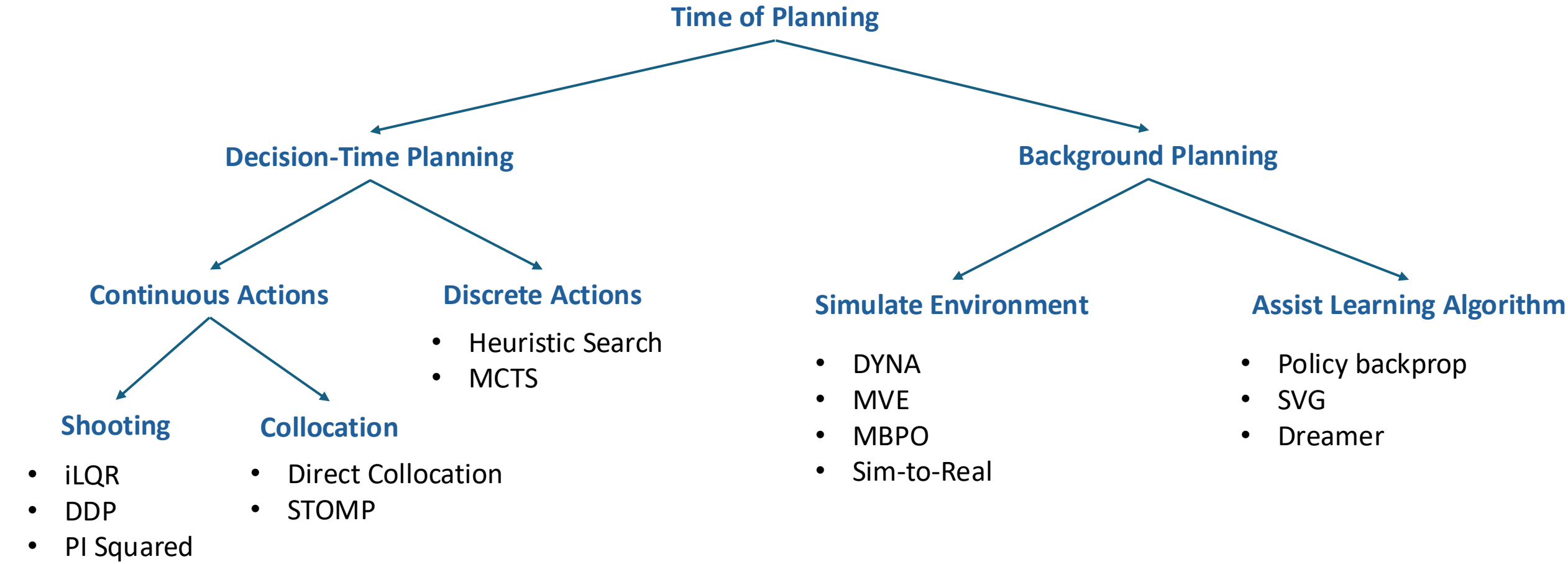
2. Assisting the learning algorithm

3. Strengthening the policy



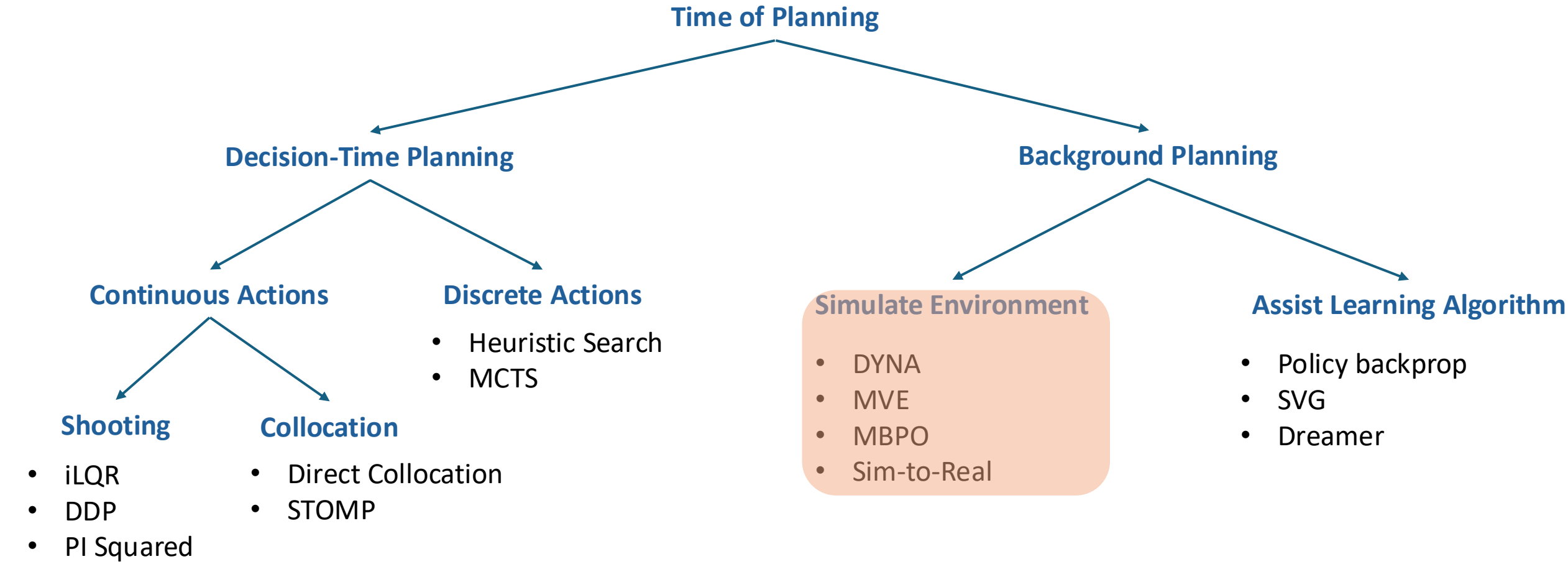
Slide from Igor Mordatch

Landscape of Planning Methods



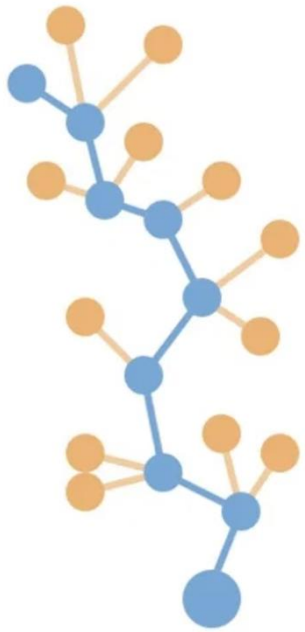
Slide from Igor Mordatch

Landscape of Planning Methods



Simulate Environment

Mix real and/or model-generated experience and apply traditional model-free methods.



The model offers a way:

- Augment the replay buffer
- Collect data safely
  - Particularly good for partially trained policies

Slide from Igor Mordatch

Learning Controllers with Sim-to-Real









- Learning High-Speed Flight in the Wild, Loquercio et al., Science Robotics 2021

The Spectrum of Simulators (or World Models)

Physics-based

Data-Driven





GENESIS

Too many to list





MuJoCo

ManiSkill

...

General / Ease of Customization / Faster

Visual Fidelity / Environment Diversity

Gathering Data To Train the Model

- **Learn only from static datasets**
  - Human demonstrations
  - Manually engineered policy rollouts (system identification)
  - Another (sub-optimal) policy
- Popular topic in research (model-based offline RL)
- What are the issues with static datasets?

Slide from Igor Mordatch

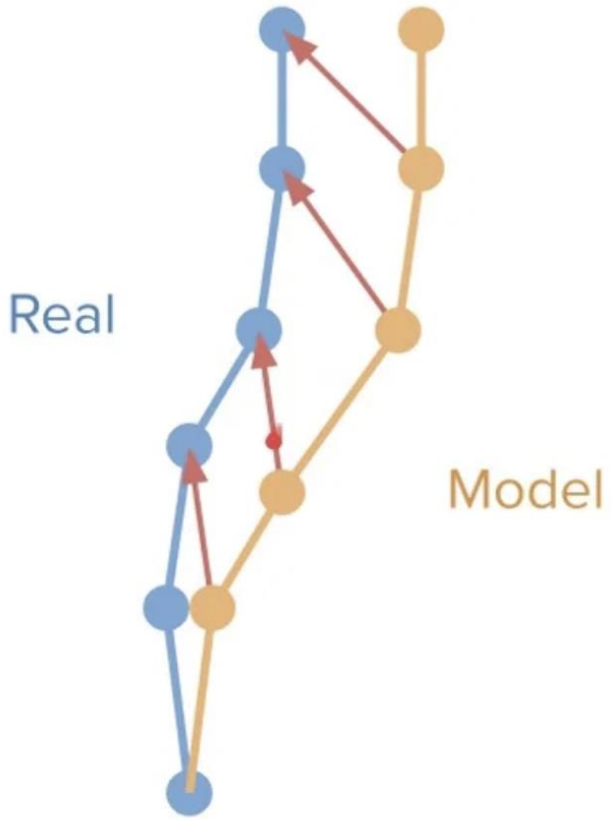
Gathering Data to Train the Model

- **Use the model to generate data to train itself**

- Example:
  - Rollout the model and train it to pull its predicted next states close to the true next states.

- Can be made more efficient with adversarial ways to self-audit the model

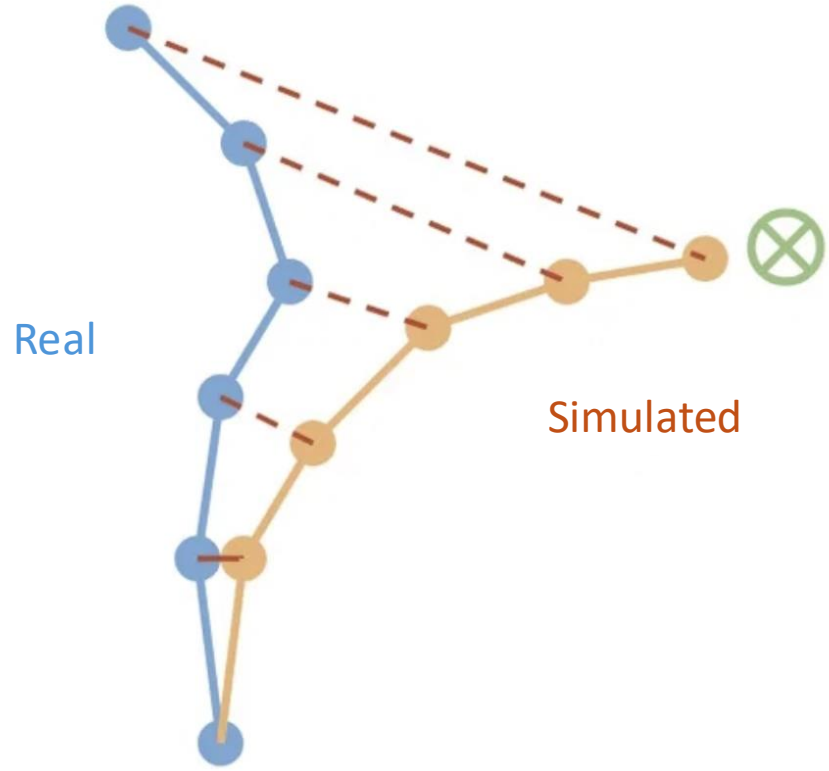
- Improves the model in the areas most relevant to the policy.
  - But requires rolling out a partially trained policy in the real world.



Slide from Igor Mordatch

Models are always wrong!

- The environment can change over time.
- The environment is too complex to be completely captured in a model.
- Small model errors propagate and compound
- Planners may exploit model errors



Slide from Igor Mordatch

Example: Model-Based Walking

- Task: Get as high as possible: $r(s_t, a) = 1 \; if \; h(s_{t+1}) > h(s_t)$



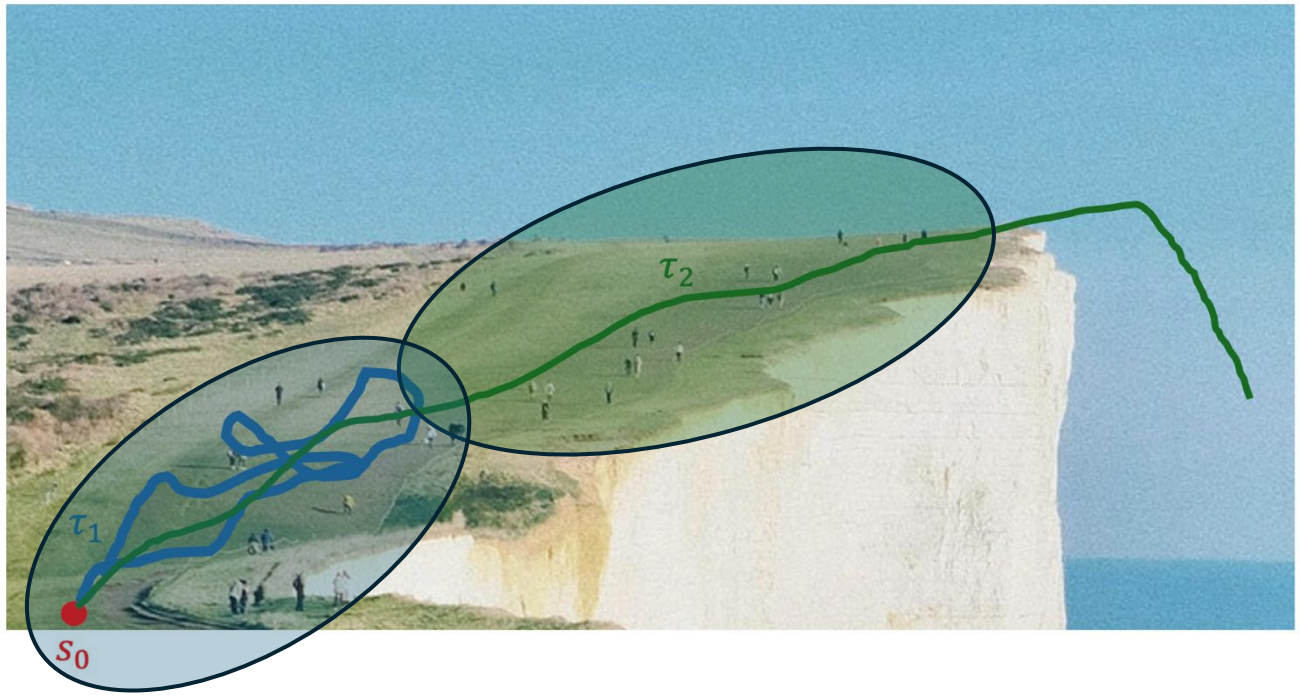
How would you tackle this problem?

When can I trust my model?

- Do not update the policy too much (using similar ideas from prior lectures) after a model update

- Reason about uncertainty
  - When certain, exploit
  - When uncertain, be safe

- Replan at decision time



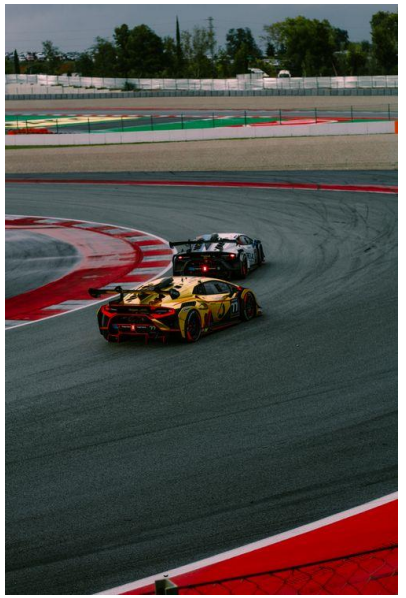
Reason about Uncertainty is… complicated

- How to quantify uncertainty?

- We don’t want to be:
  - Too pessimistic in uncertain states
  - Too optimistic in uncertain states

- This is an exploration/exploitation problem.
  - The tradeoff often depends on the use case





Estimating Uncertainty

- Conceptually helpful to think of two sources of uncertainty:

- **Epistemic Uncertainty**
  - Represents the model's lack of knowledge about the world
  - "Errors due to lack of training data"
  - Reducible (e.g., collect more data)

- **Aleatoric Uncertainty / Risk**
  - World's inherent stochasticity
  - "Errors due to noisy sensors"
  - Irreducible (unless you buy a better sensor)

Epistemic Uncertainty: Lack of Knowledge





Car





???

Aleatoric Uncertainty: Inherent Randomness





Car

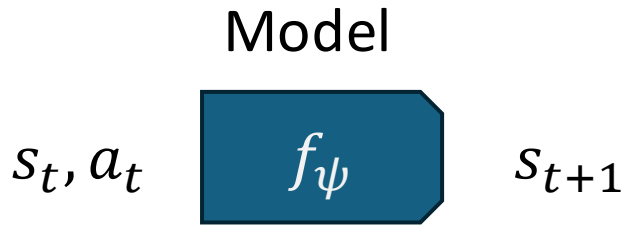




???

Formal Definition of Uncertainty

- When learning a model, we optimize:

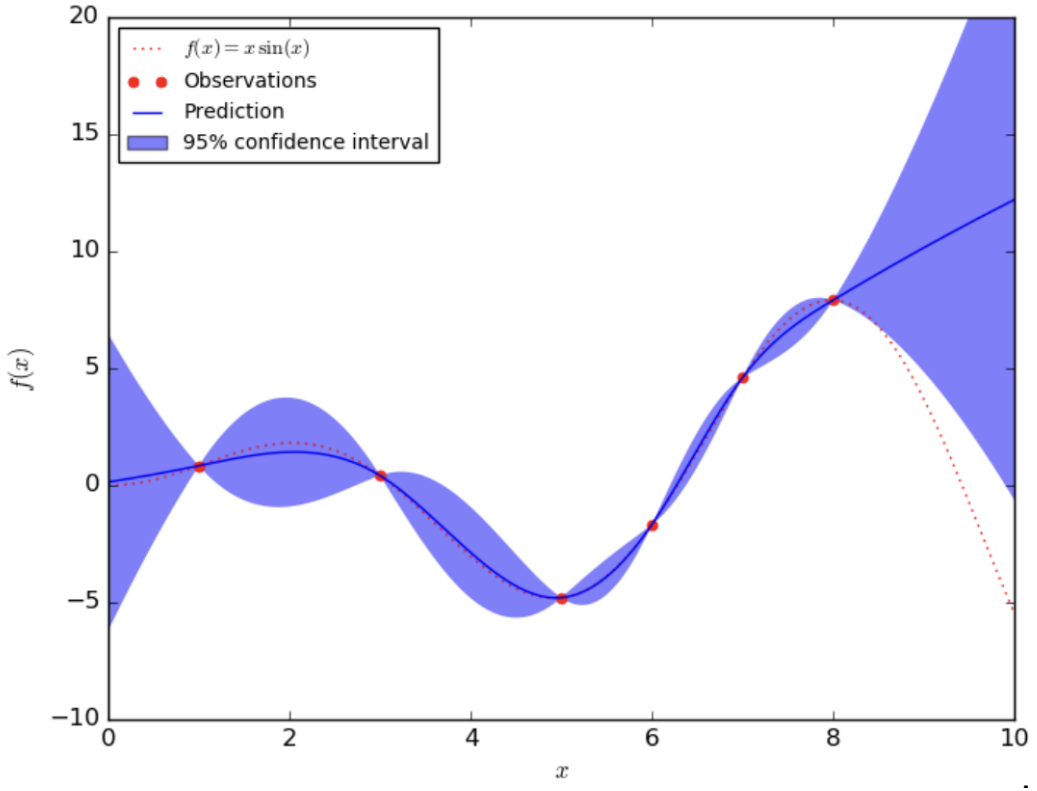


- $\psi^* = \arg\max_\psi \log p(\psi|D)$
- $\psi^* = \arg\min_\psi \sum_{i \in D} \|f_\psi(s_t, a_t) - s_{t+1}\|^2$

- The model uncertainty can be quantified by the entropy of $p(\psi|D)$
  - High entropy means that many models are probable, so I have a higher likelihood of making mistakes.

Estimating Model Uncertainty

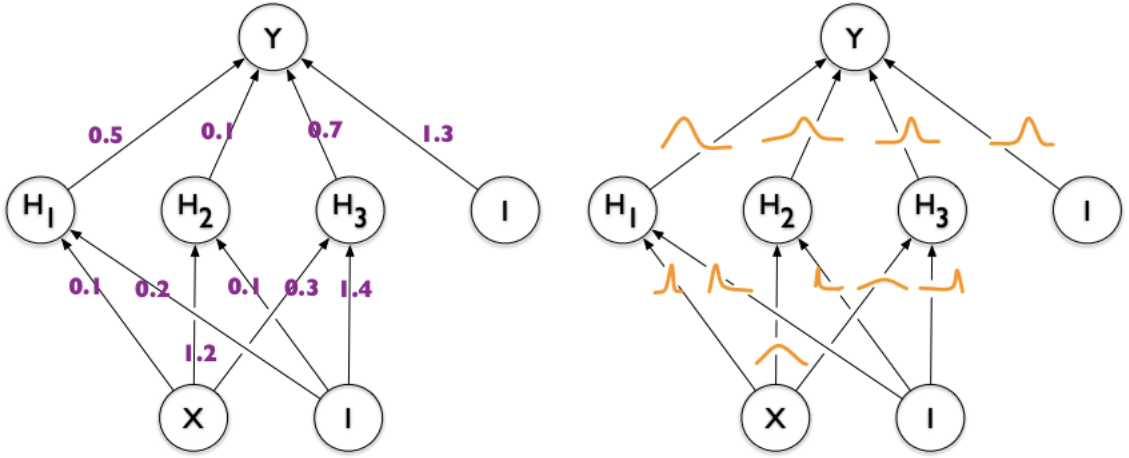
- Parametrize the model with a function approximation method that directly captures uncertainty:
  - Gaussian Processes



Estimating Model Uncertainty

- Parametrize the model with a function approximation method that directly captures uncertainty:
  - Gaussian Processes

- Incorporate uncertainty into the design of neural networks:
  - Assume independent model parameters: $p(\psi|D) = \Pi_i p(\theta_i|D)$
  - Probabilistic/Bayesian neural nets



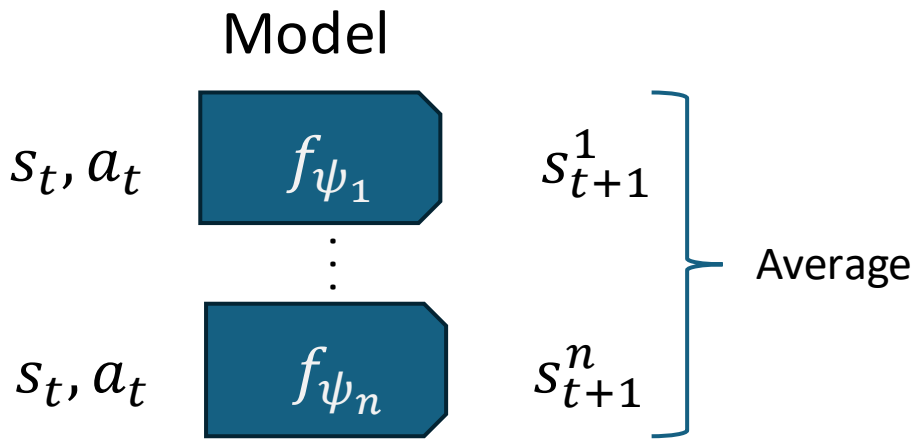
Weight Uncertainty in Neural Networks, Blundell et al.

Estimating Model Uncertainty

- Parametrize the model with a function approximation method that directly captures uncertainty:
  - Gaussian Processes

- Incorporate uncertainty into the design of neural networks:
  - Assume independent model parameters: $p(\psi|D) = \Pi_i p(\theta_i|D)$
  - Probabilistic/Bayesian neural nets

- Ensemble of models
  - Assume that $p(\psi|D) \approx \frac{1}{N}\sum_n \delta(\psi_n)$
  - In practice: average the output of models trained on the same data/different subsets.



What to do with uncertainty?

- Penalize a trajectory if it encounters states with high uncertainty
  - Plan for the worst-case scenario wrt model uncertainty
- Compute a plan over a set of possible models, then average the policy
  - Example: each ensemble member is a model.
- Give a bonus if you encounter a state with high uncertainty
  - Generally called “curiosity-based” learning
  - Great for exploration if you don’t care about safety

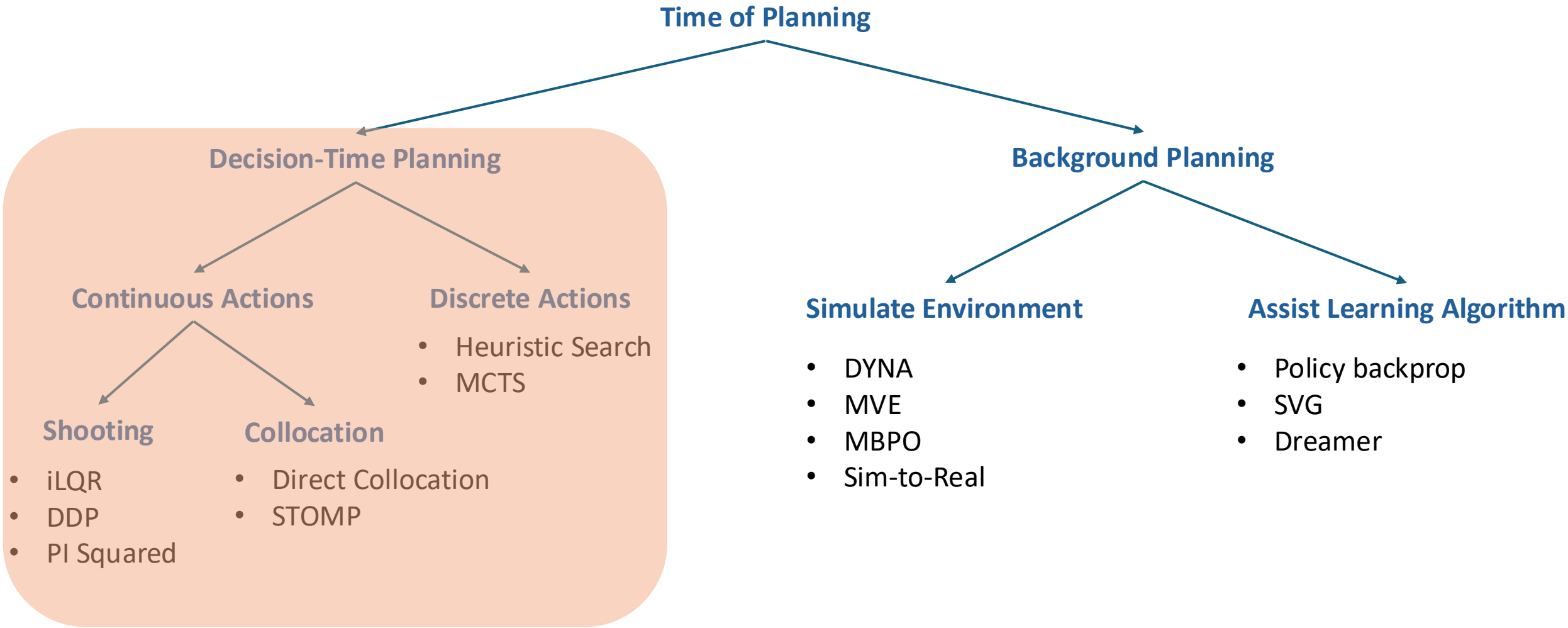
When can I trust my model?

- Do not update the policy too much (using similar ideas from prior lectures) after a model update

- Reason about uncertainty
  - When certain, exploit
  - When uncertain, be safe

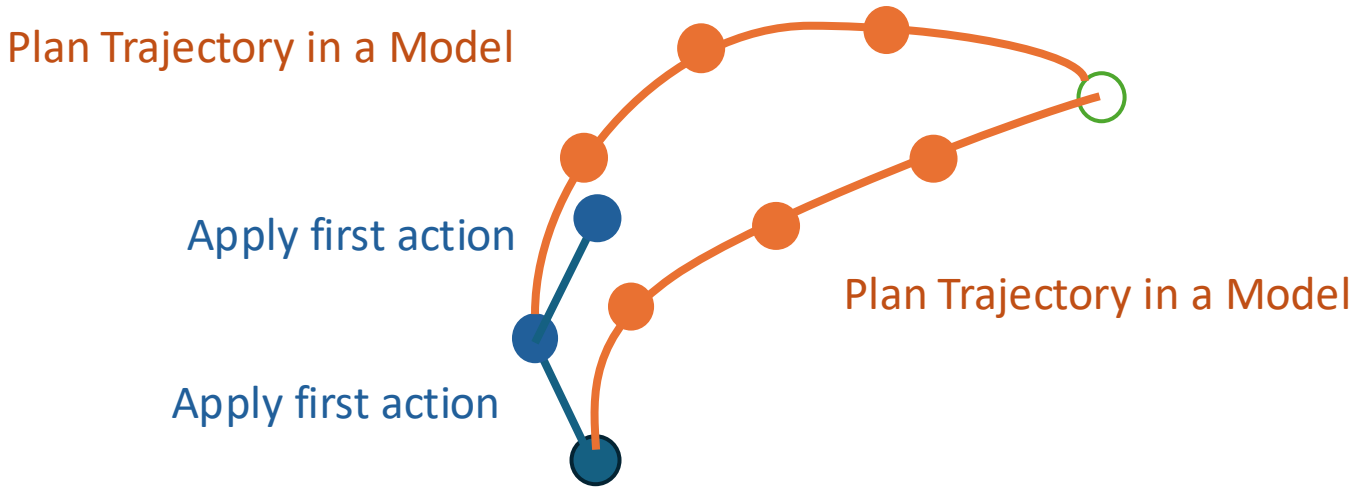
- Replan at decision time

Landscape of Planning Methods



Decision-time Planning

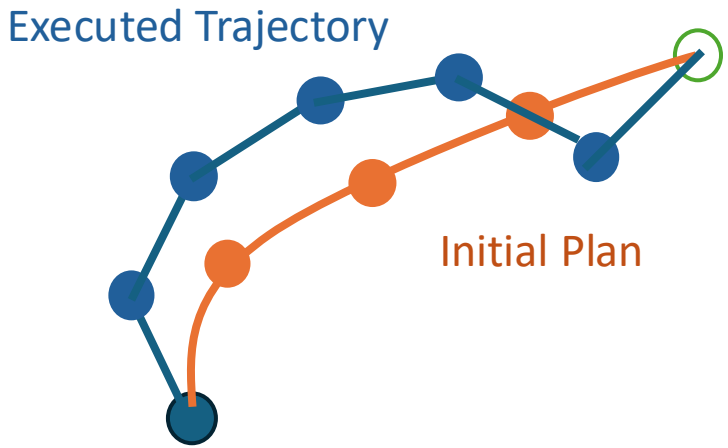
- Don’t commit to a plan, but **continually re-plan**



Slide from Igor Mordatch

Decision-time Planning

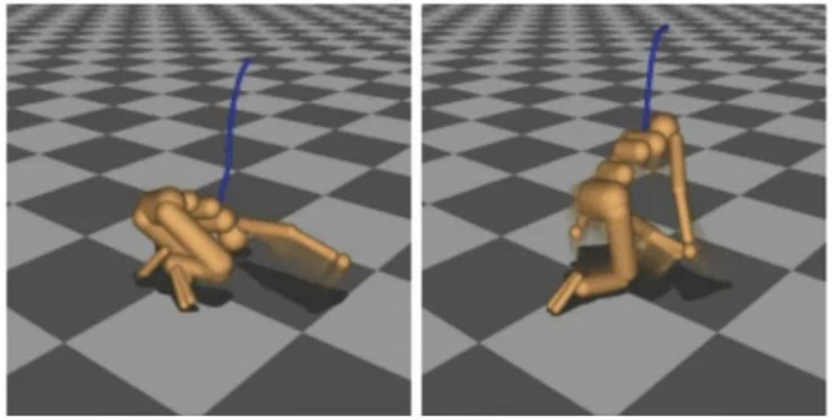
- Helpful when models are not very accurate
  - Don’t need a perfect model, just one pointing in the right direction
  - The worse the model, the shorter the time between replanning



Slide from Igor Mordatch

Decision-time Planning for Real-Time Control

- Isn’t replanning expensive?
  - Reuse the solution from the prior step
  - Make the model simple (e.g., linear) and find analytical solutions



Tassa et al (2012). Synthesis and Stabilization of Complex Behaviors through Online Trajectory Optimization.



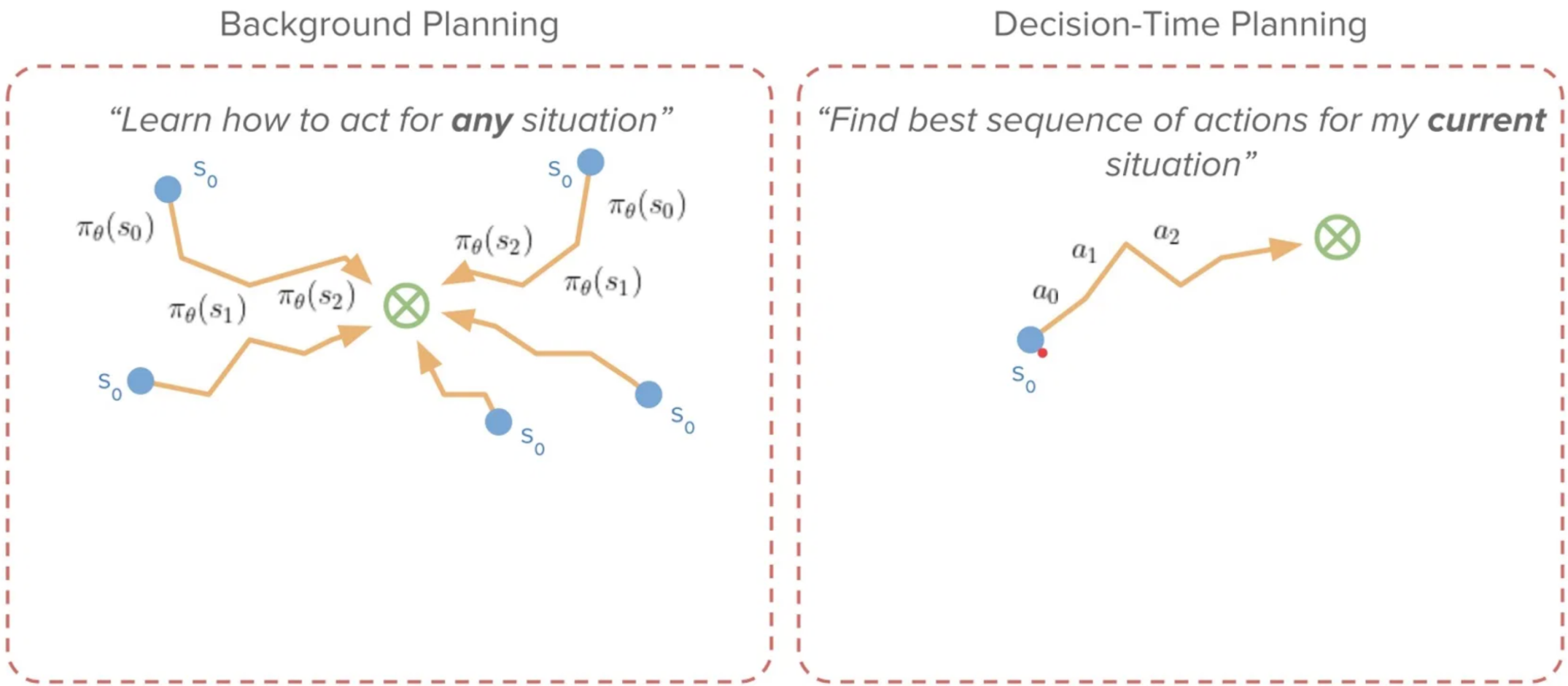
Williams et al (2017). Information Theoretic MPC for Model-Based Reinforcement Learning.



Nagabandi et al (2019). Deep Dynamics Models for Learning Dexterous Manipulation.

Slide from Igor Mordatch

Background vs decision-time planning

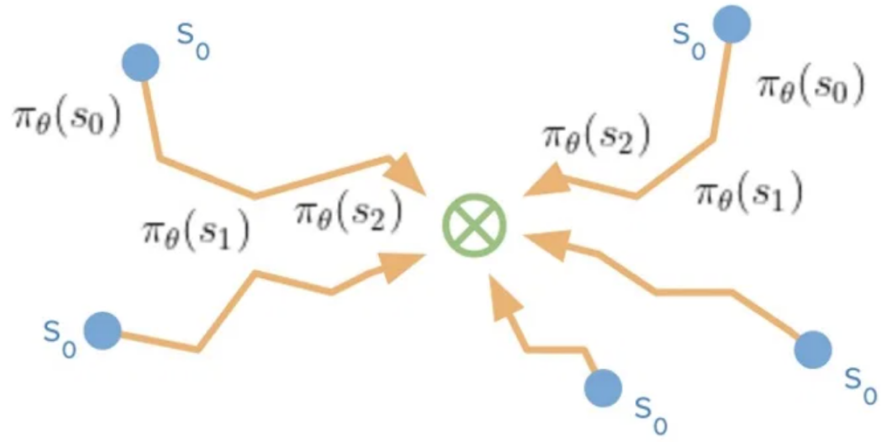


Slide from Igor Mordatch

Background vs decision-time planning

Background Planning

*"Learn how to act for **any** situation"*



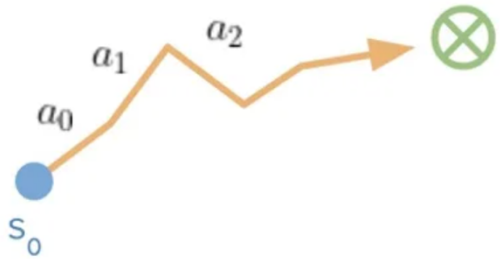
Optimization variables: θ
(parameters of policy, or value, or ...)

ex: $J(\theta) = \mathbb{E}_{s_0}[\sum_{t=0}^{H} \gamma^t r_t], \quad a_t = \pi_\theta(s_t)$

Habit

Decision-Time Planning

*"Find best sequence of actions for my **current** situation"*



Optimization variables: $a_0, a_1, \ldots, a_H$
(sequence of actions and/or states)

ex: $J(a_0, \ldots, a_H) = \sum_{t=0}^{H} \gamma^t r_t$

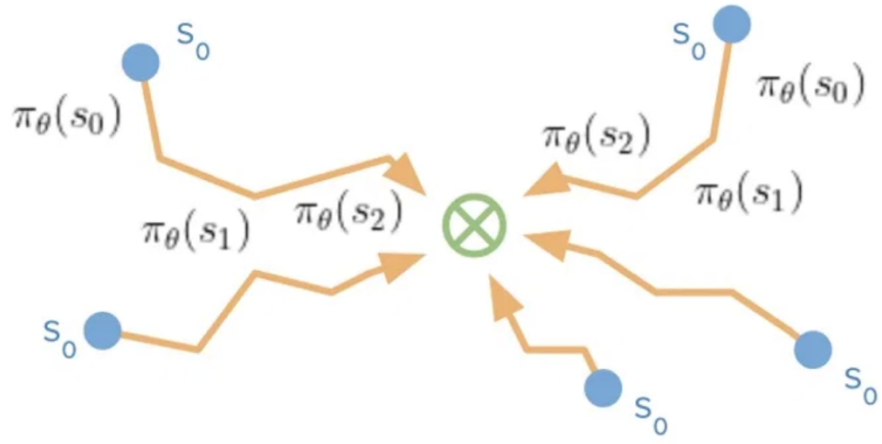
Improvisation

Slide from Igor Mordatch

Background vs decision-time planning

Background Planning

*"Learn how to act for **any** situation"*

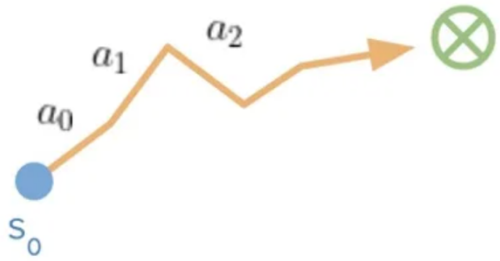


Optimization variables: θ
(parameters of policy, or value, or ...)

ex: $J(\theta) = \mathbb{E}_{s_0}[\sum_{t=0}^{H} \gamma^t r_t], \quad a_t = \pi_\theta(s_t)$

Decision-Time Planning

*"Find best sequence of actions for my **current** situation"*



Optimization variables: $a_0, a_1, \ldots, a_H$
(sequence of actions and/or states)

ex: $J(a_0, \ldots, a_H) = \sum_{t=0}^{H} \gamma^t r_t$

Fast

Kahneman (2011). Thinking Fast and Slow

Slow

Slide from Igor Mordatch

Background vs Decision-Time Planning

Background Planning	Decision-Time Planning	
x	v	Act without any learning
(x)	(v)	Robust in unfamiliar situations
x	v	Independent of Obs. Space

Background

$$J(\theta) = \mathbb{E}_{s_0}[\sum_{t=0}^{H} \gamma^t r_t], \quad a_t = \pi_\theta(s_t)$$

joint angles?
pixels?
graphs?

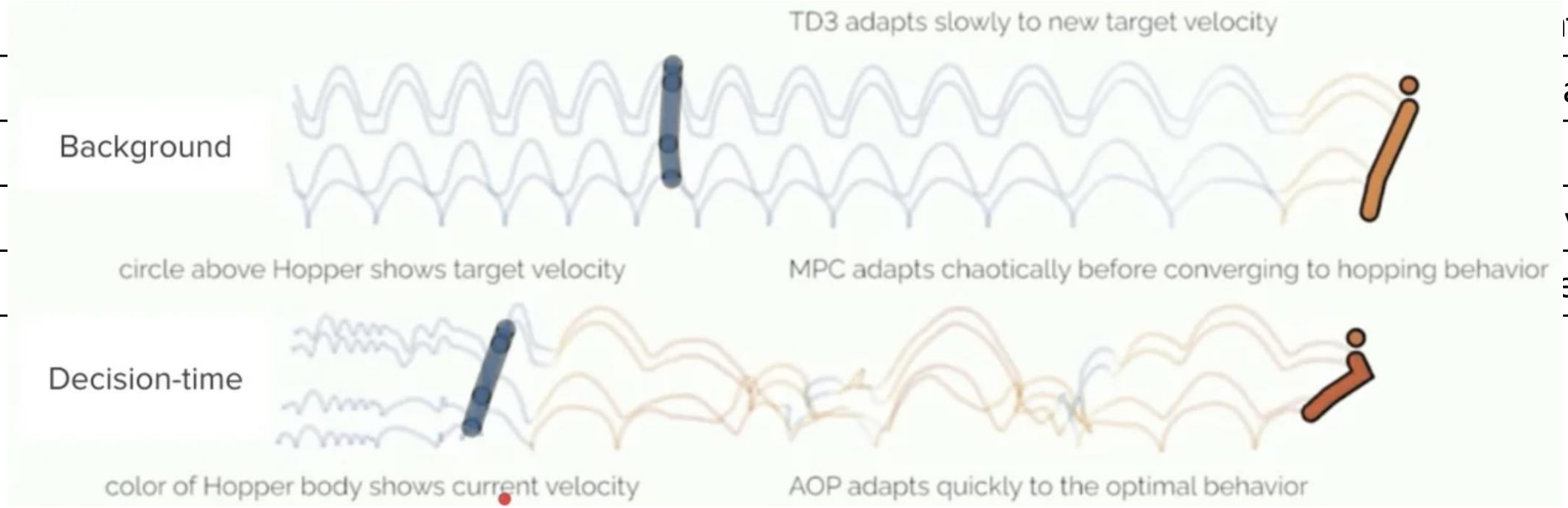
Decision-Time

$$J(a_0, ..., a_H) = \sum_{t=0}^{H} \gamma^t r_t$$

Slide from Igor Mordatch

Background vs Decision-Time Planning

Background Planning	Decision-Time Planning	
x	v	Act without any learning



Adaptive Online Planning for Continual Lifelong Learning
Lu et al (2019)

Slide from Igor Mordatch

Background vs Decision-Time Planning

Background Planning	Decision-Time Planning	
x	v	Act without any learning
(x)	(v)	Robust in unfamiliar situations
x	v	Independent of Obs. Space
v	(v)	Partial Observability
v	x	Fast Computation at Deployment
v	x	Predictability and Coherence
v	x	Similar for Continuous and Discrete Actions

- Mixing them is a good idea, but non-trivial
- When to switch?
- How to switch?

Slide from Igor Mordatch

Continuous vs Discrete Actions

- Not a significant distinction for background planning
  - Learn stochastic policies that sample from discrete or continuous actions
  - The optimization is generally smooth with respect to policy parameters θ

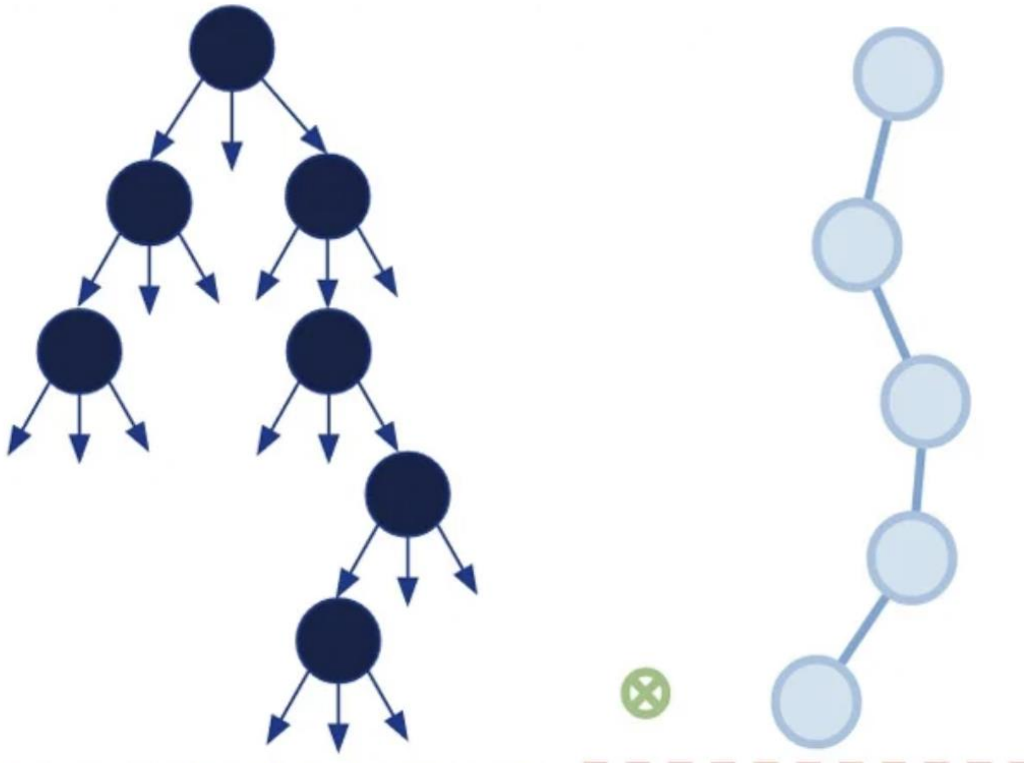
$$J(\theta) = \mathbb{E}_{\pi}\left[\sum_{t} r_t\right], \qquad a_t \sim \pi_\theta(\cdot|s_t)$$

- Leads to specialized methods for decision-time planning
  - Discrete search (as in MCTS) or continuous trajectory optimization

Slide from Igor Mordatch

Continuous vs Discrete Actions

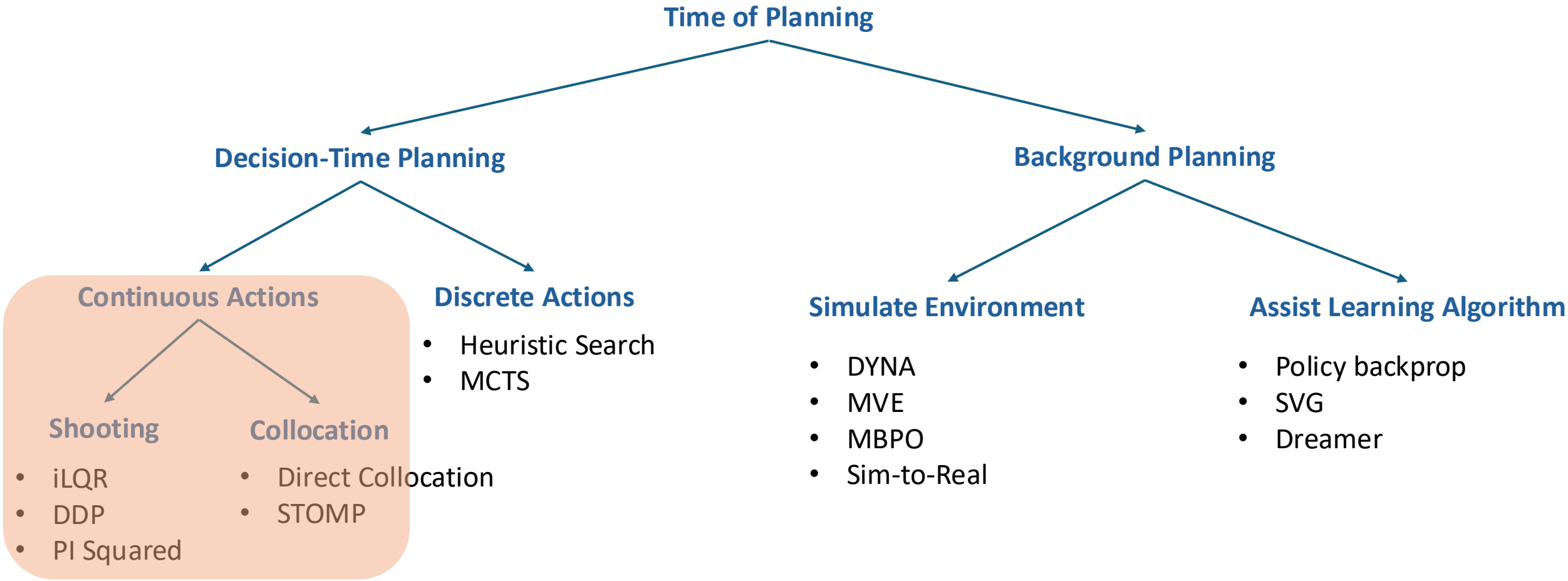
- **Discrete**
  - Example: Monte-Carlo Tree Search
  - Consider multiple action paths and store the returns
- **Continuous**
  - Trajectory Optimization
  - Consider only one action paths and store its return



Discrete **Continuous**

Slide from Igor Mordatch

Landscape of Planning Methods



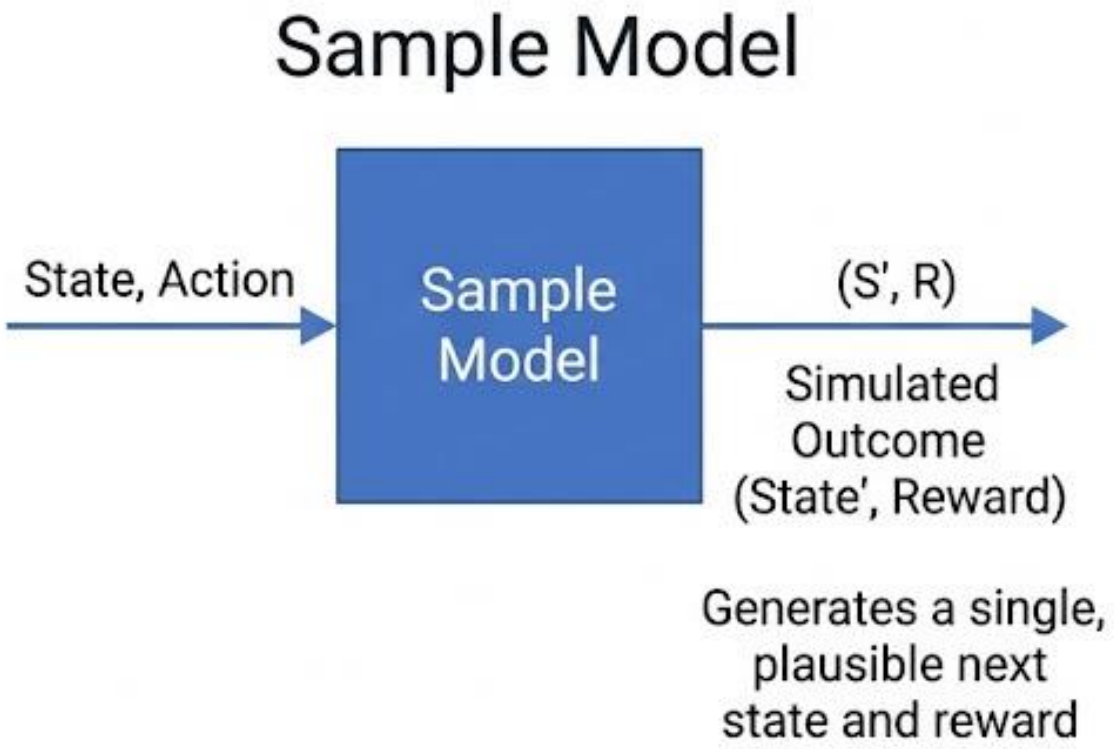
Why so many methods?

- Issues of naïve planning with continuous actions:
- Sensitivity and poor conditioning
- Only reaches a local optimum
- Slow convergence
- Addressing each leads to a different class of methods

Slide from Igor Mordatch

How to Represent the Model?

The traditional state-space model



Observation-space model

Why is this significantly harder than a state-space model?

The challenges of observation-space models

- Fundamentally, an ill-defined problem
  - The current observation might not have enough information to predict the next observation
  - Using a history of observations helps, but does not solve the problem
- Often much more computationally expensive than a state-space model
  - Mostly because of the high dimensionality of observations

Should I model the entire observation space?



Should I model the entire observation space?



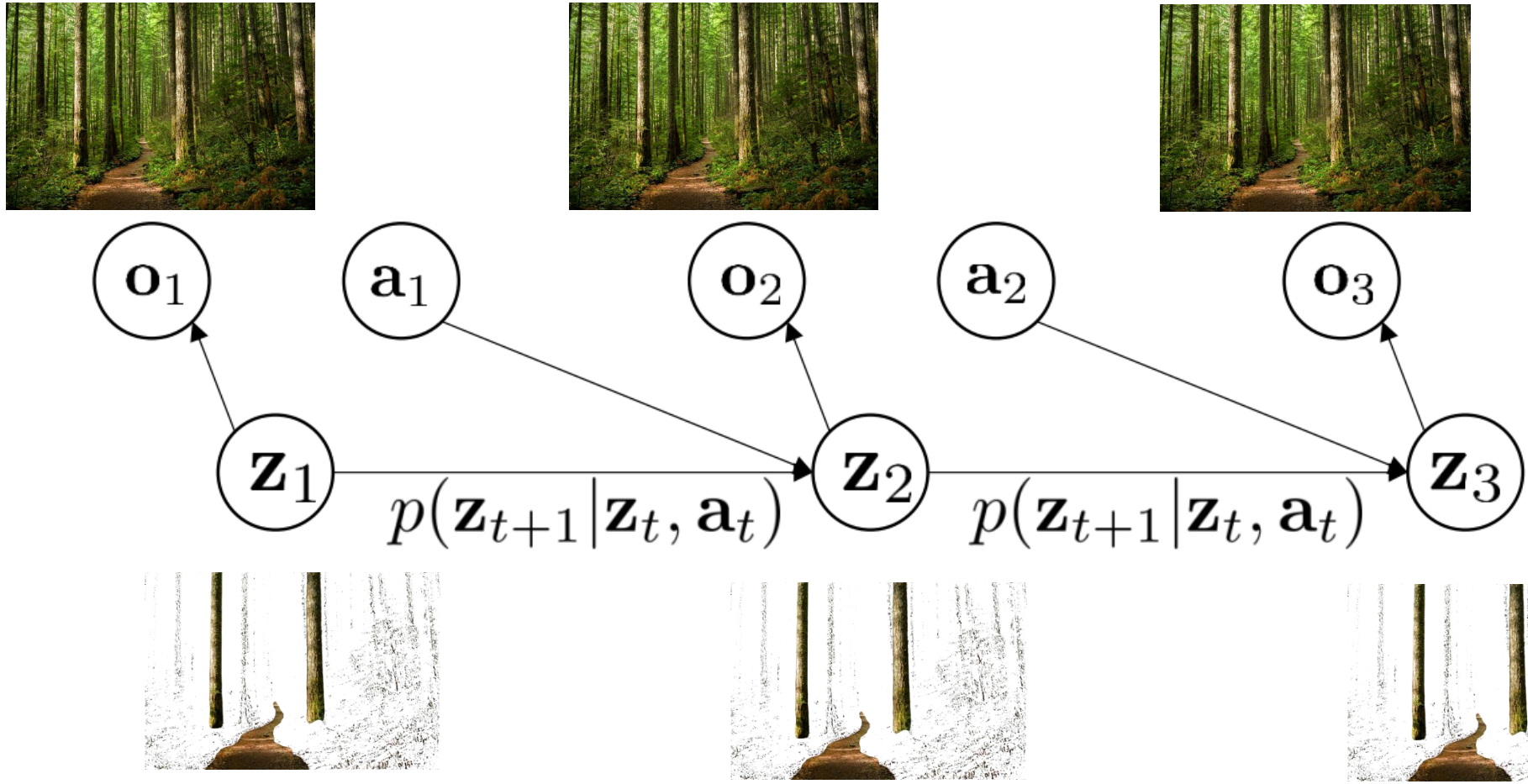
Should I model the entire observation space?

- Can save significant computation by focusing on the part of the observation that is most relevant to the task.

- The prediction problem becomes simpler
  - Better performance for the same data

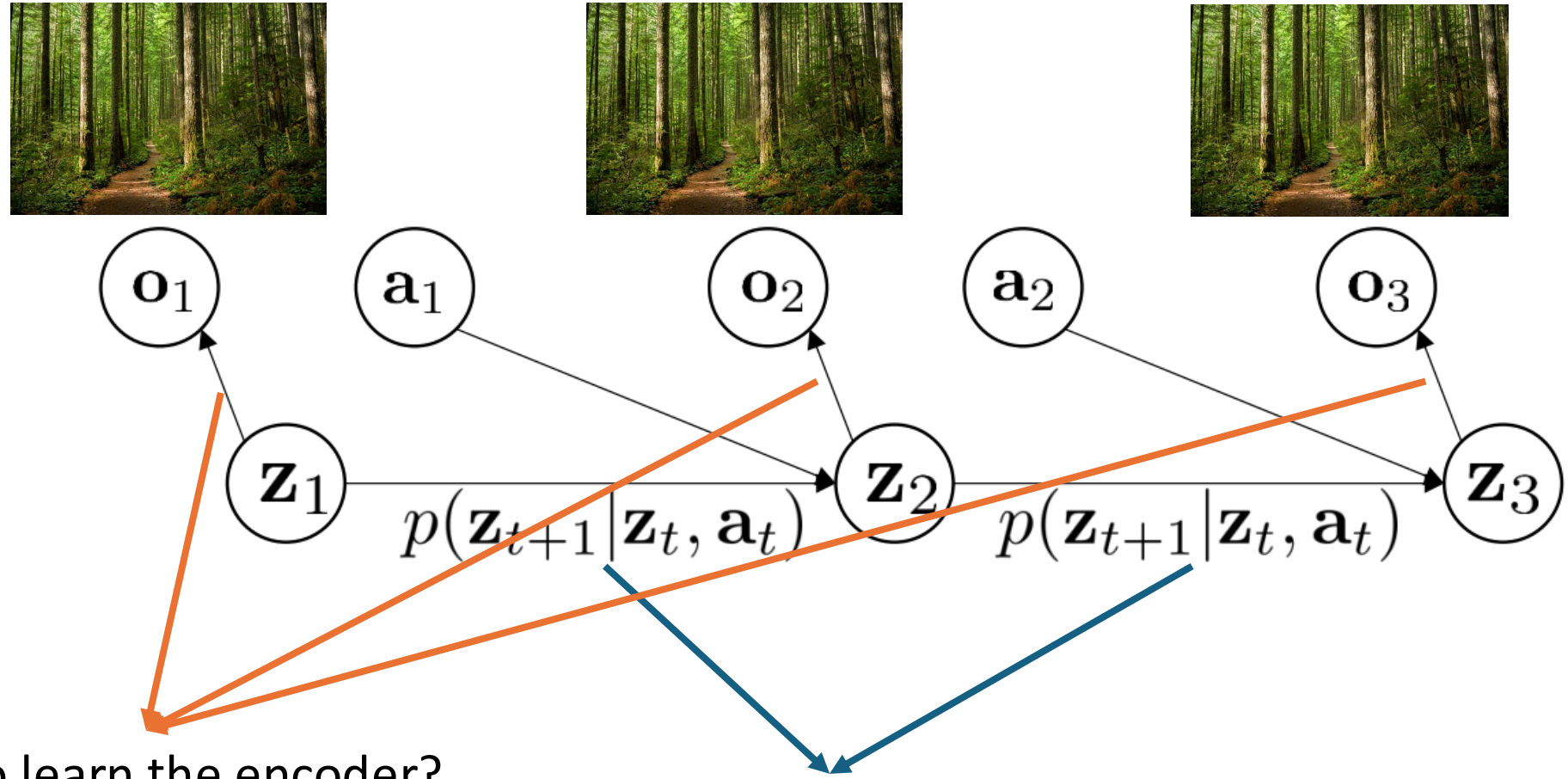
- But how to decide what is important?
  - Decide in advance and build estimators -> back to (quasi) state-space models.
  - Learn from experience -> Latent-space models

Latent Space Models



Adapted from Sergey Levine

Latent Space Models



How to learn the encoder?

How to learn the model?

The components of a latent-space model

- An encoder with parameters ϕ

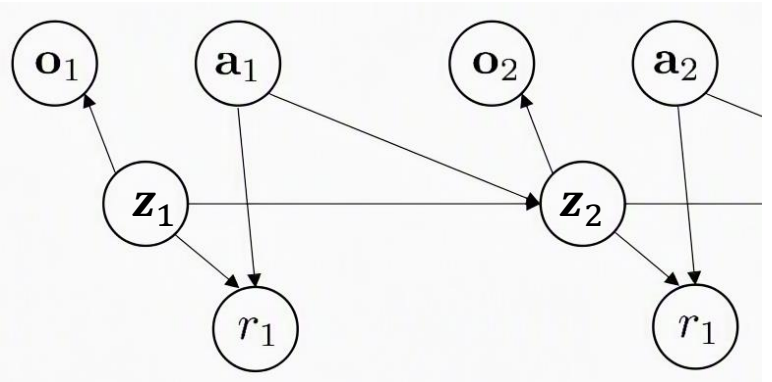
$$z_t = g_\phi(o_t)$$

- (Sometimes) a decoder with parameters δ

$$o_t \approx d_\delta(z_t)$$

- A model with parameters ψ
  - Forward predictor: $z_{t+1} = f_\psi(z_t, a_t)$
  - Reward estimator: $r_t \approx r_\psi(z_t)$

Learning the encoder and model parameters



- A loss to reconstruct the reward:

$$L_{\mathrm{r}}(\phi,\psi) = \left(r_t - r_\psi\left(g_\phi(o_t)\right)\right)^2$$

- A loss to be consistent between time steps:

$$\mathrm{L_d}(\phi,\psi) = \left(g_\phi(o_{t+1}) - f_\psi\left(g_\phi(o_t), a_t\right)\right)^2$$

- (sometimes) A loss to reconstruct the observation (through a decoder d_δ)

$$\mathrm{L_o}(\phi,\delta) = \left(o_t - d_\delta(g_\phi(o_t))\right)^2$$

- *They depend on each other*, so we generally learn them jointly.
  - Alternate between keeping one fixed and computing the gradients for the other.

Learning the encoder and model parameters

- Collect a set of trajectories in the real environment
- And minimize the sum of the prior losses on all transitions (o_t, a_t, r_t, o_{t+1})
- Often, we make both the encoder and model stochastic (e.g., representing them by diffusion models)
  - Helps if the environment is stochastic
  - Helps with multi-modality of “important” things (e.g., two paths in a forest)
  - Helps if some important things are not visible (e.g., a *likely* obstacle far away)

Advantage Actor Critic with a Latent Model

- Initialize policy π_θ, encoder g_ϕ, and model f_ψ, r_ψ
- Until Converged
  - Collect an episode with $\pi_\theta\left(g_\phi(o_t)\right)$ in the real environment. Add it to the replay buffer.
  - Use data from the replay buffer to update the model (slide 53).
  - Plan in the model for n times:
    - Sample an initial state z_0
    - Obtain a latent space trajectory $\left(z_0, a_0 = \pi(z_0), r_\psi(z_0, a_0), \dots\right)$
    - Compute the policy gradient using the simulated trajectory
    - Update policy

- Does it make sense to use off-policy learning (PPO,Q-Learning) in the planning stage?

Notable Examples

DREAM TO CONTROL: LEARNING BEHAVIORS BY LATENT IMAGINATION

Danijar Hafner *
University of Toronto
Google Brain

Timothy Lillicrap
DeepMind

Jimmy Ba
University of Toronto

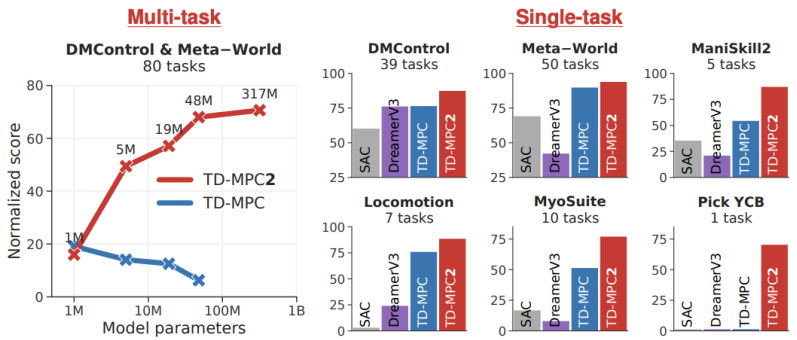
Mohammad Norouzi
Google Brain

Abstract

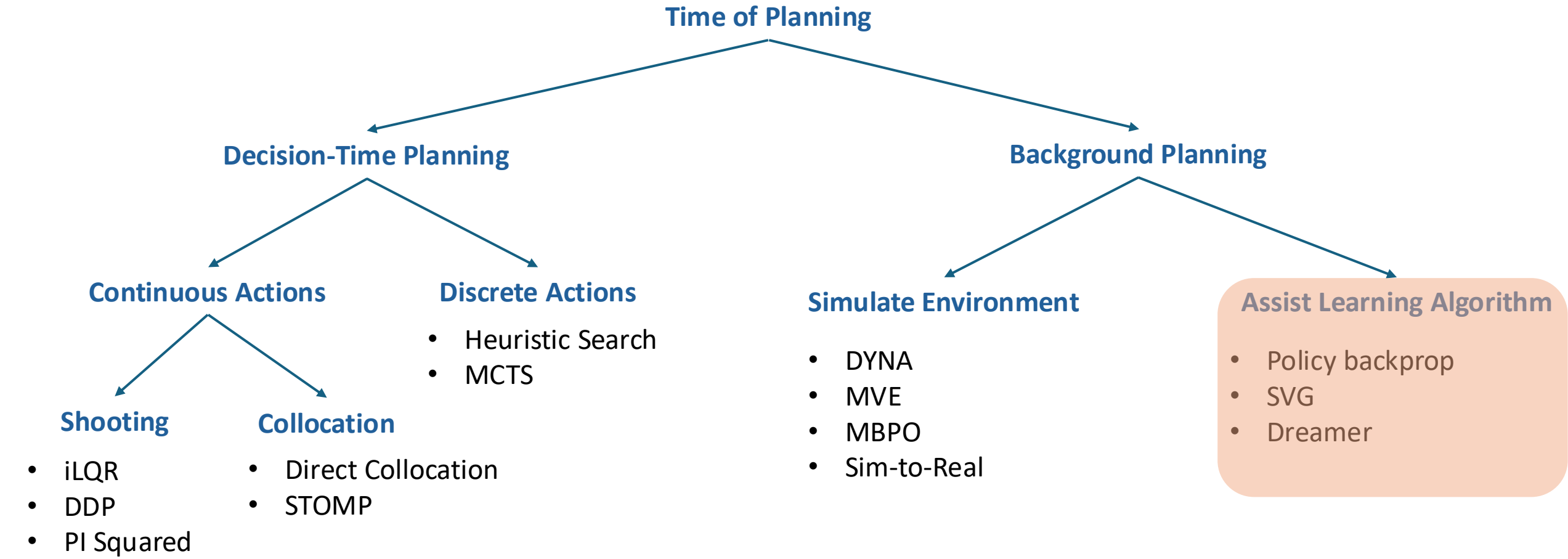
Learned world models summarize an agent's experience to facilitate learning complex behaviors. While learning world models from high-dimensional sensory inputs is becoming feasible through deep learning, there are many potential ways for deriving behaviors from them. We present Dreamer, a reinforcement learning agent that solves long-horizon tasks from images purely by latent imagination. We efficiently learn behaviors by propagating analytic gradients of learned state values back through trajectories imagined in the compact state space of a learned world model. On 20 challenging visual control tasks, Dreamer exceeds existing approaches in data-efficiency, computation time, and final performance.

TD-MPC2: Scalable, Robust World Models for Continuous Control

Nicklas Hansen*, **Hao Su***†, **Xiaolong Wang***†
*University of California San Diego, †Equal advising
{nihansen,haosu,xiw012}@ucsd.edu



Next Class


Time of Planning
Decision-Time Planning
Background Planning
Continuous Actions
Discrete Actions
Simulate Environment
Assist Learning Algorithm
Shooting
Collocation
• Heuristic Search
• MCTS
• DYNA
• MVE
• MBPO
• Sim-to-Real
• Policy backprop
• SVG
• Dreamer
• iLQR
• DDP
• PI Squared
• Direct Collocation
• STOMP