

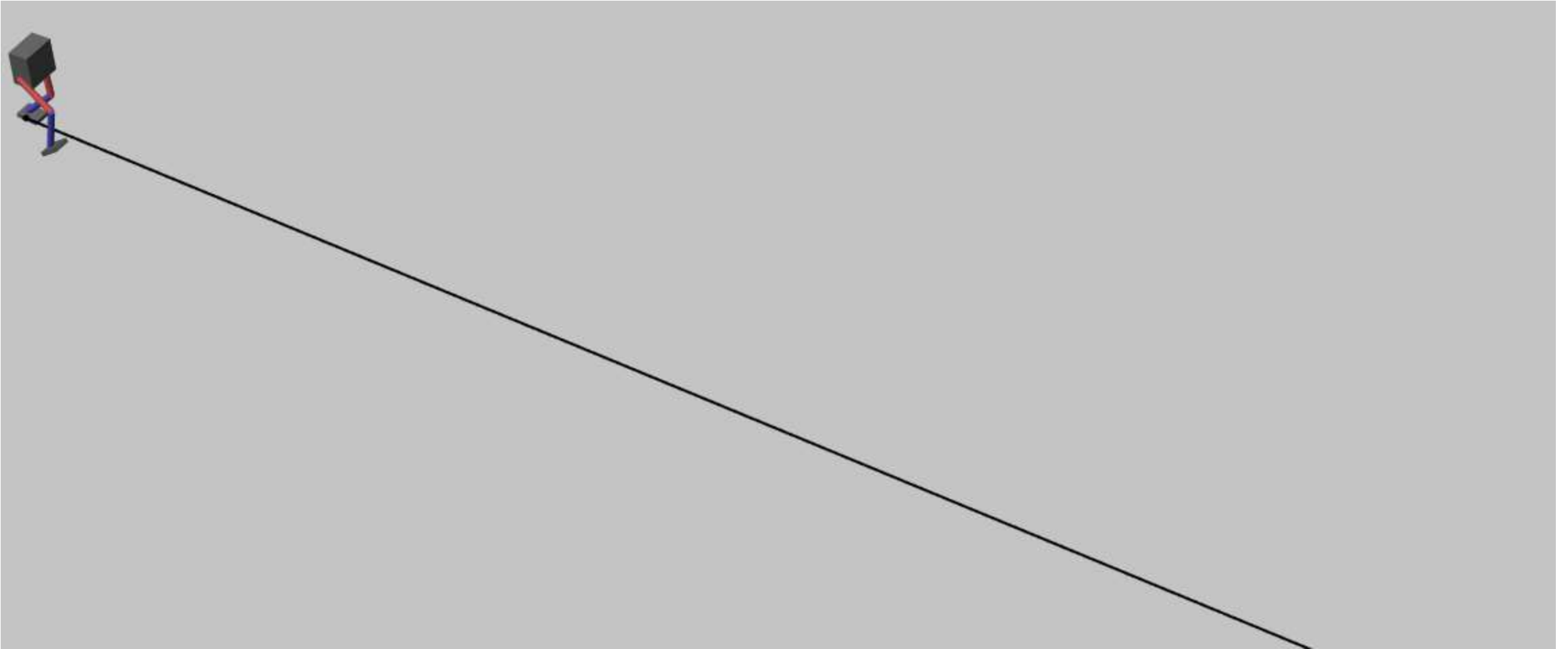
Reinforcement Learning: Leveraging Deep Learning for Controls

Aditya Baru

Goal: We hope you walk away knowing the answer to these questions

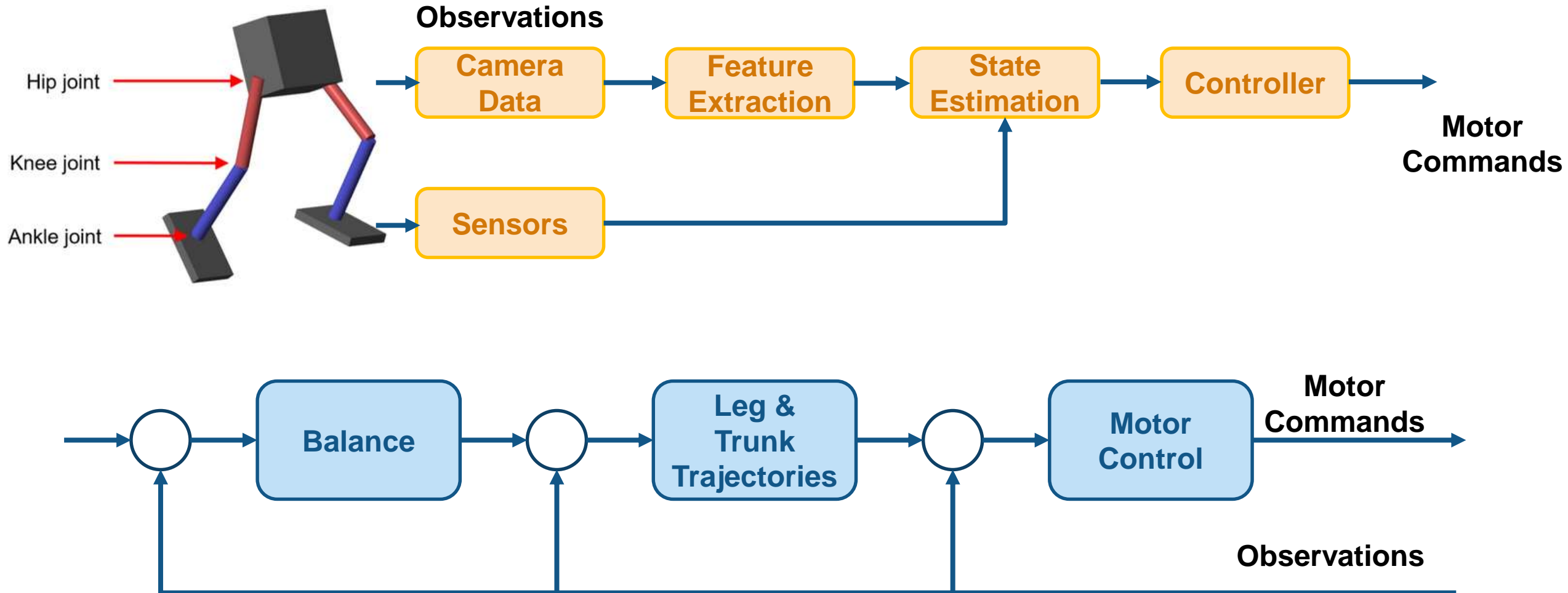
- What is reinforcement learning and why should I care about it?
- How do I set it up and solve it? [*from an engineer's perspective*]
- What are some benefits and drawbacks?

Why should you care about reinforcement learning?

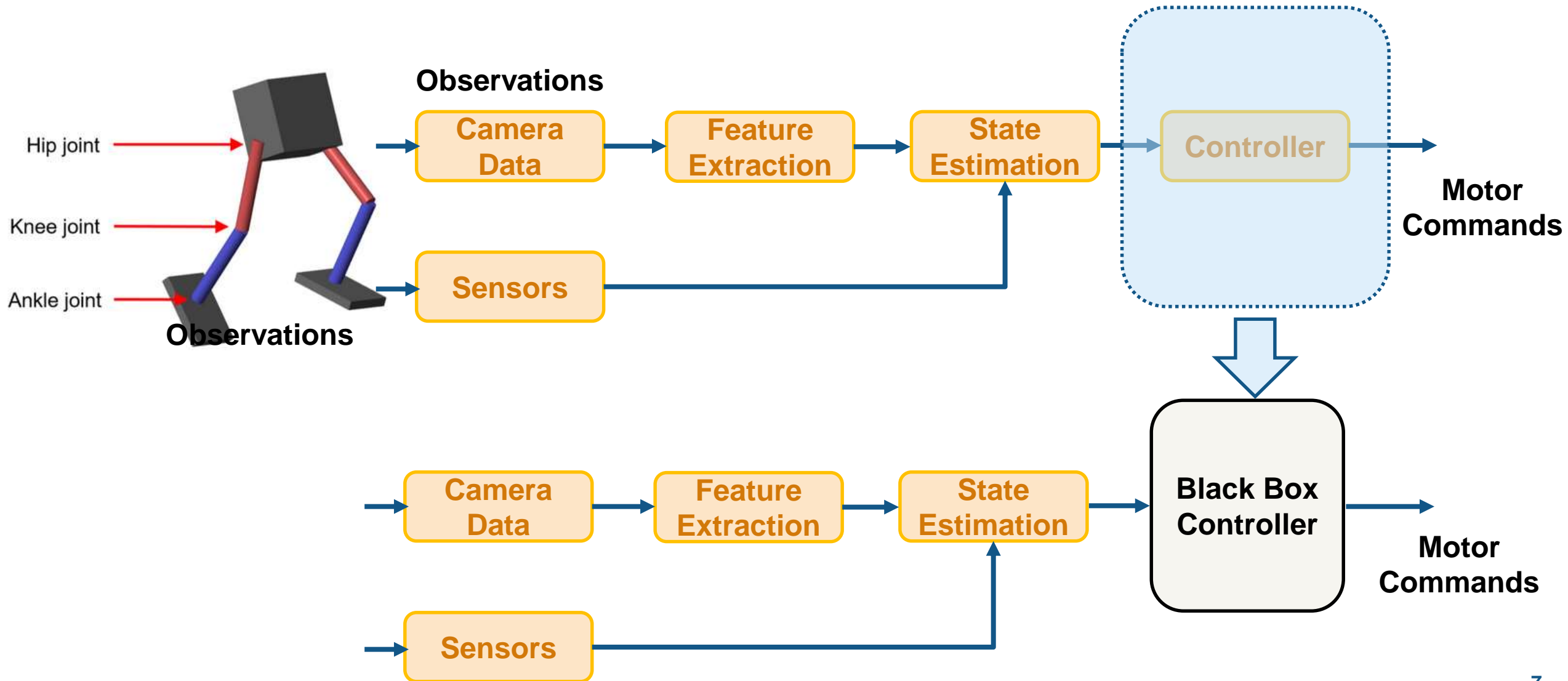


Teach a robot to follow a straight line using camera data

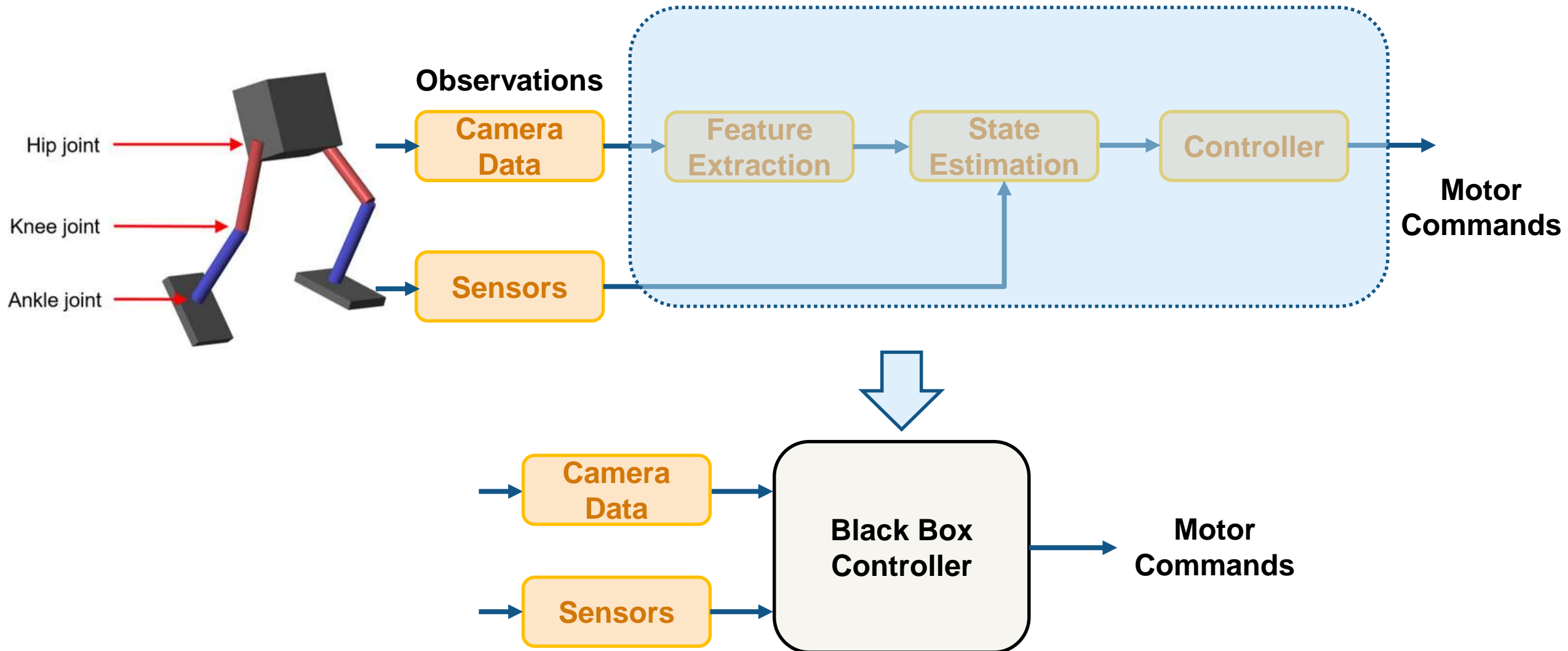
Let's try to solve this problem the traditional way



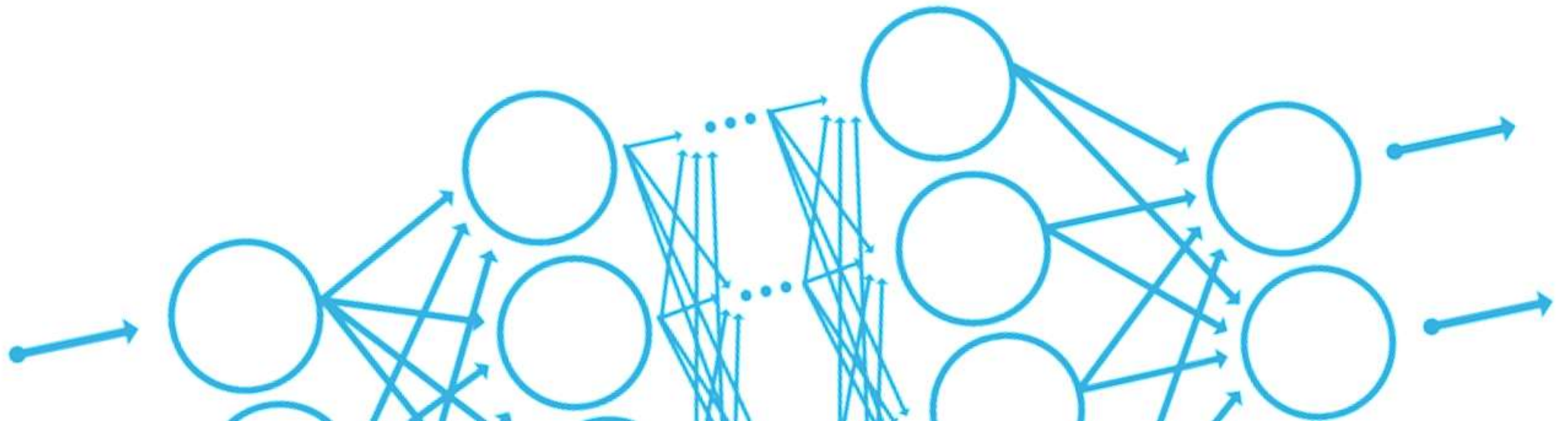
What is the alternative approach?



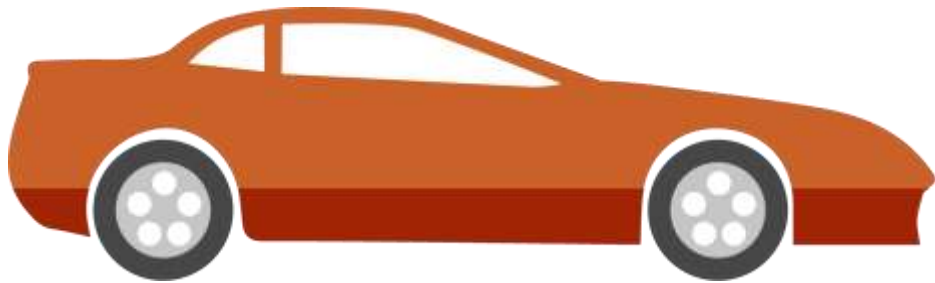
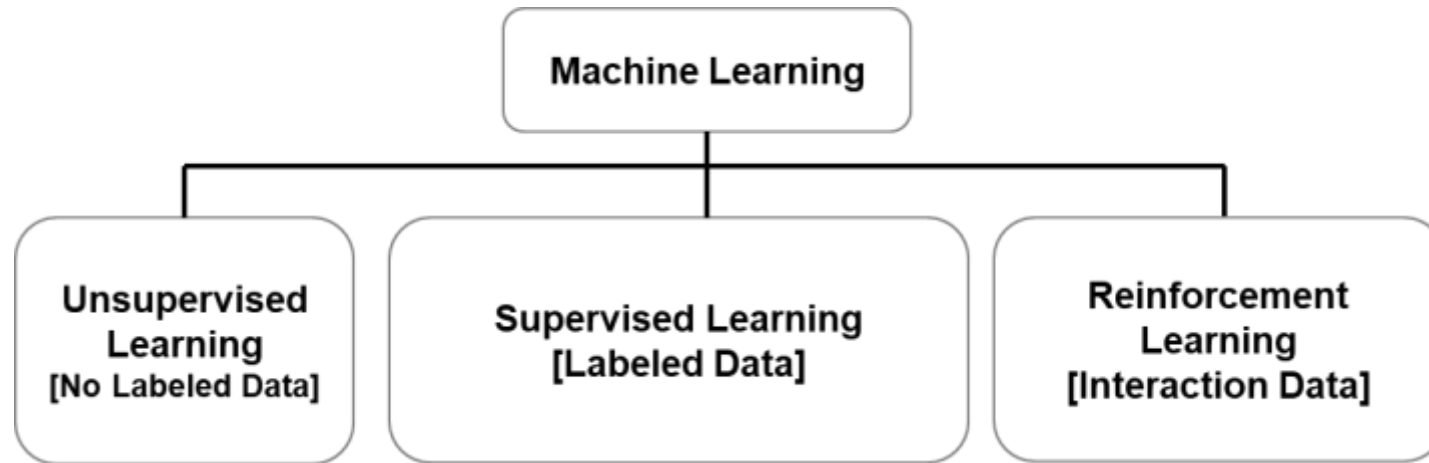
What is the alternative approach?



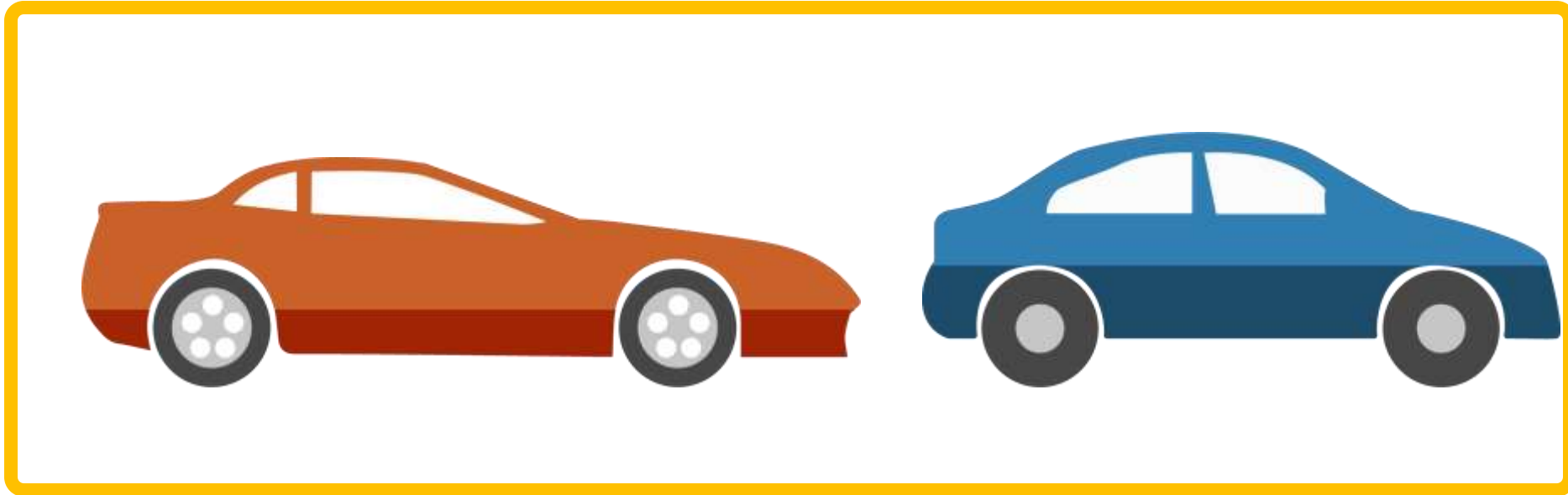
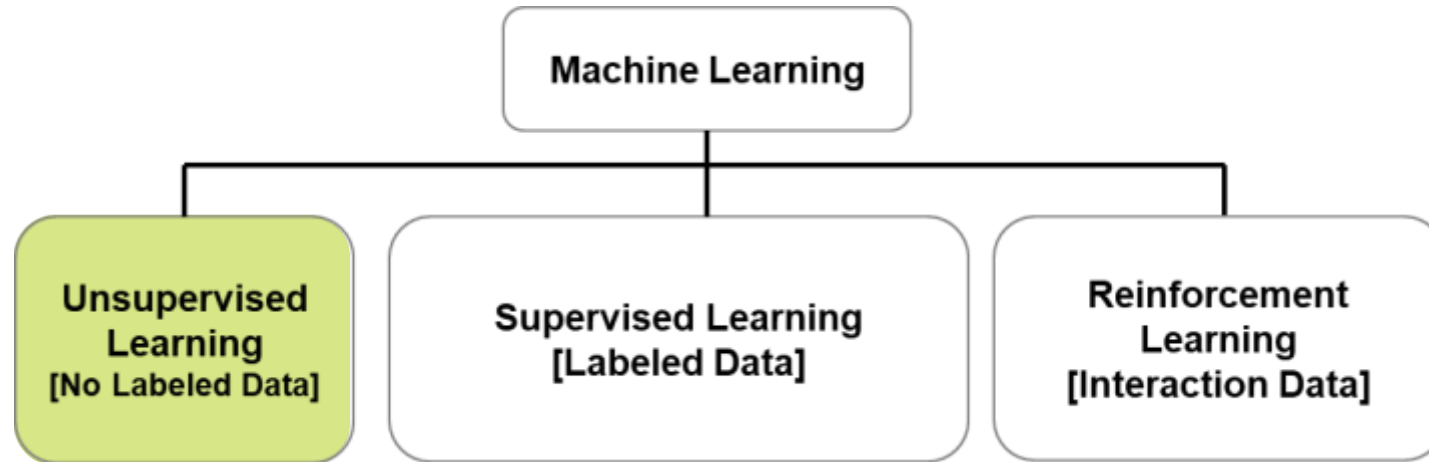
What is reinforcement learning?



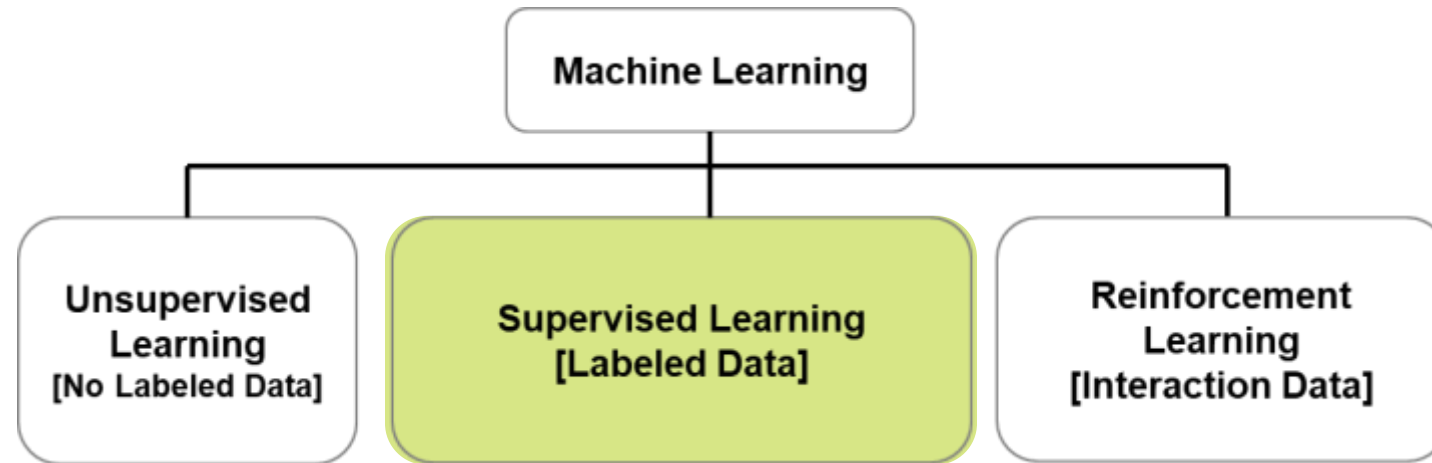
Reinforcement Learning vs Machine Learning vs Deep Learning



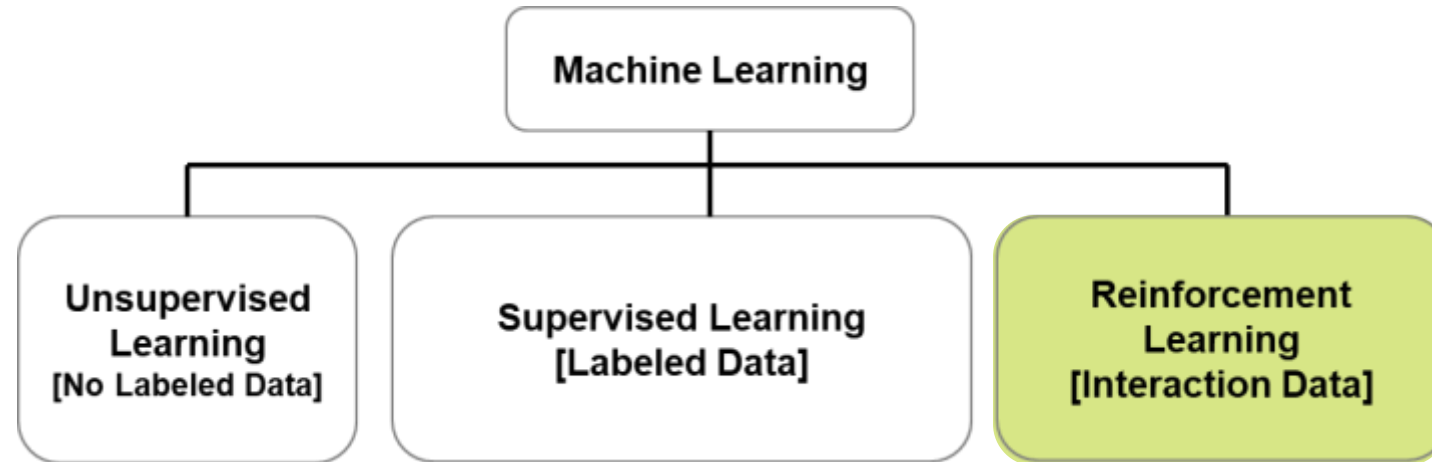
Reinforcement Learning vs Machine Learning vs Deep Learning



Reinforcement Learning vs Machine Learning vs Deep Learning

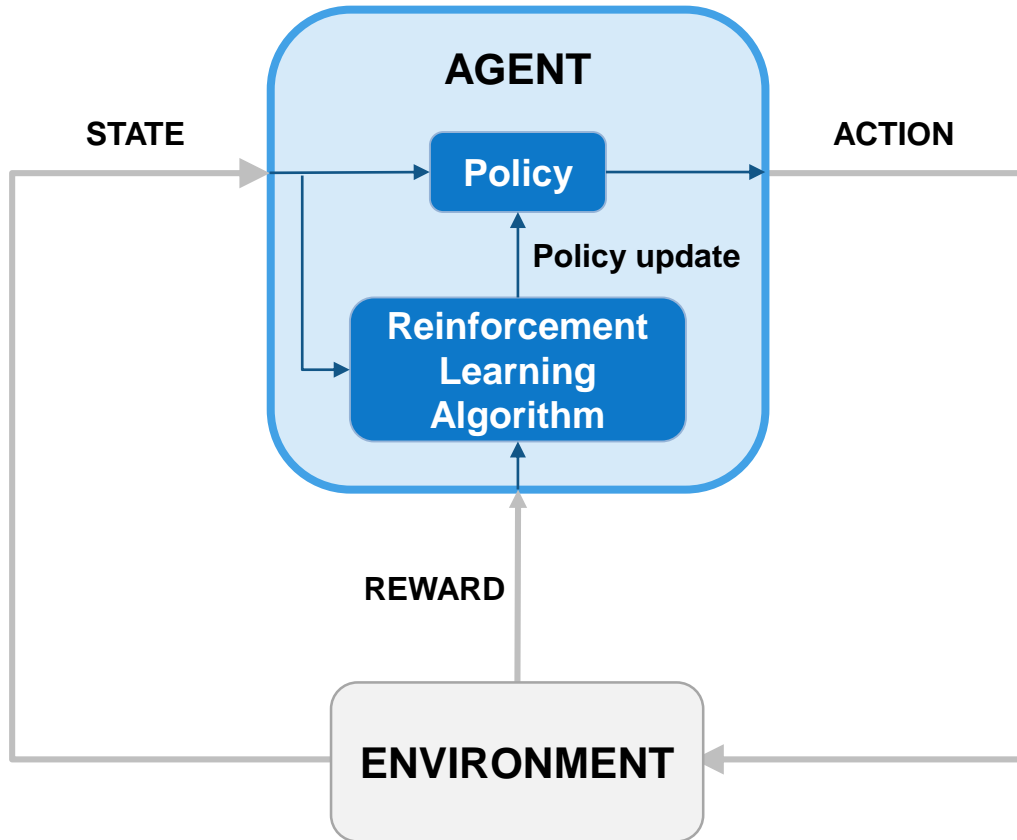


Reinforcement Learning vs Machine Learning vs Deep Learning



A Practical Example of Reinforcement Learning

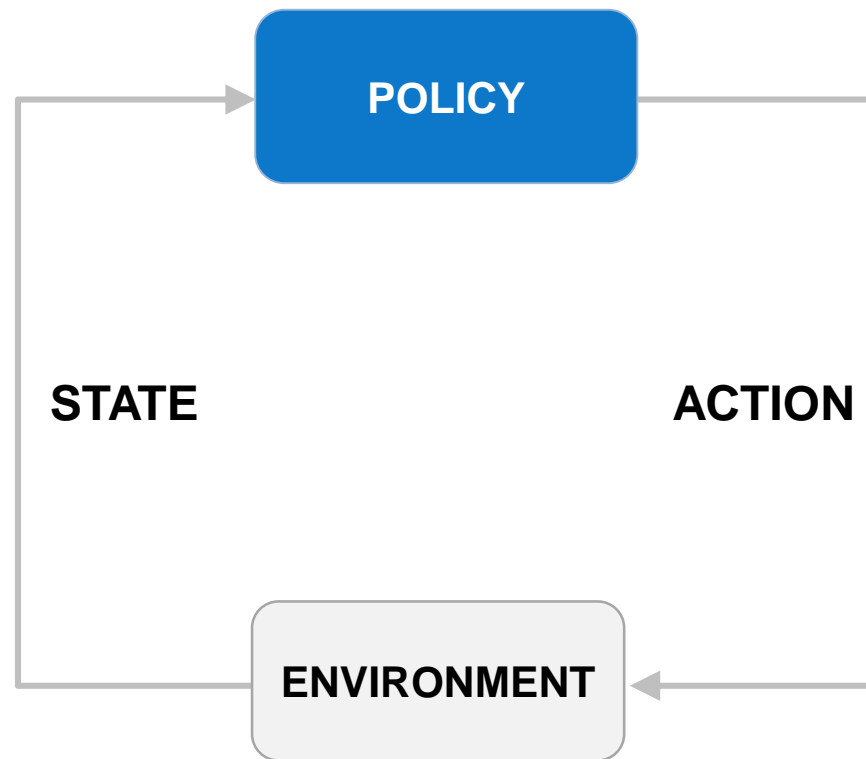
Training a Self-Driving Car



- Vehicle's computer learns how to drive...
(**agent**)
- using sensor readings from LIDAR, cameras,...
(**state**)
- that represent road conditions, vehicle position,...
(**environment**)
- by generating steering, braking, throttle commands,...
(**action**)
- based on an internal state-to-action mapping...
(**policy**)
- that tries to optimize driver comfort & fuel efficiency...
(**reward**).
- The policy is updated through repeated trial-and-error by a **reinforcement learning algorithm**

A Practical Example of Reinforcement Learning

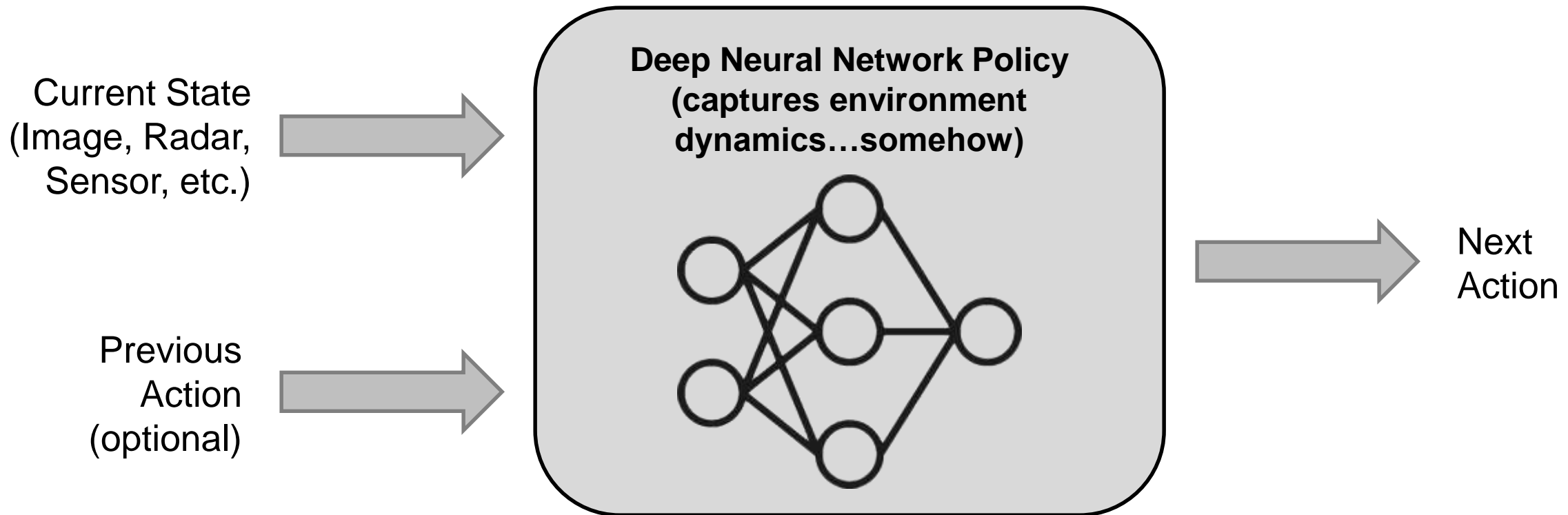
A Trained Self-Driving Car Only Needs A Policy To Operate



- Vehicle's computer uses the final state-to-action mapping... (**policy**)
- to generate steering, braking, throttle commands,... (**action**)
- based on sensor readings from LIDAR, cameras,... (**state**)
- that represent road conditions, vehicle position,... (**environment**)

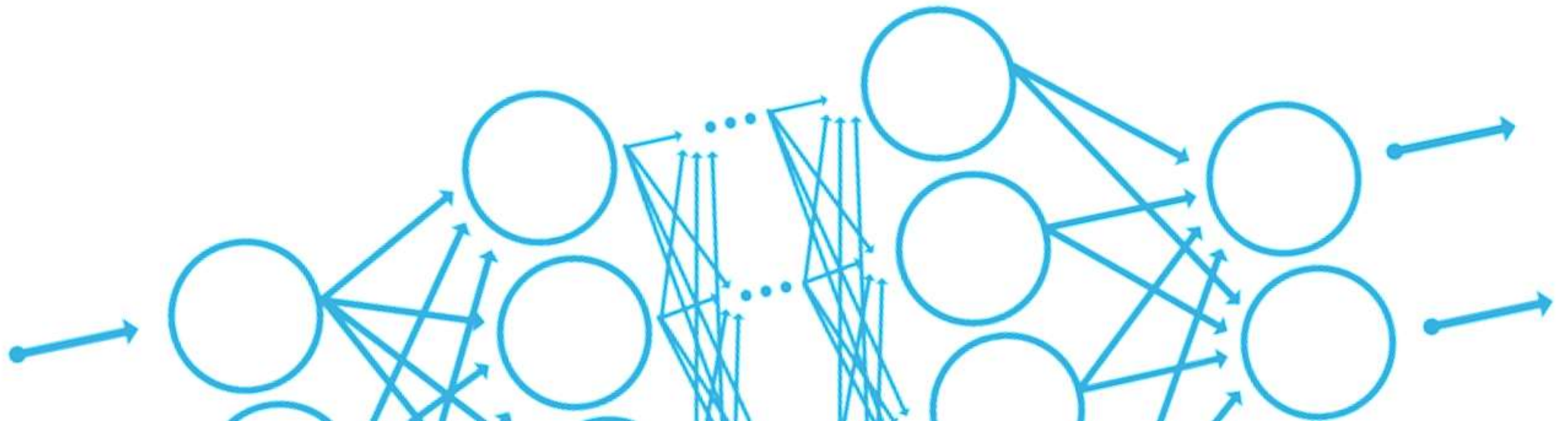
By definition, this trained policy is optimizing driver comfort & fuel efficiency

A deep neural network trained using reinforcement learning is a black-box model that determines the best possible action



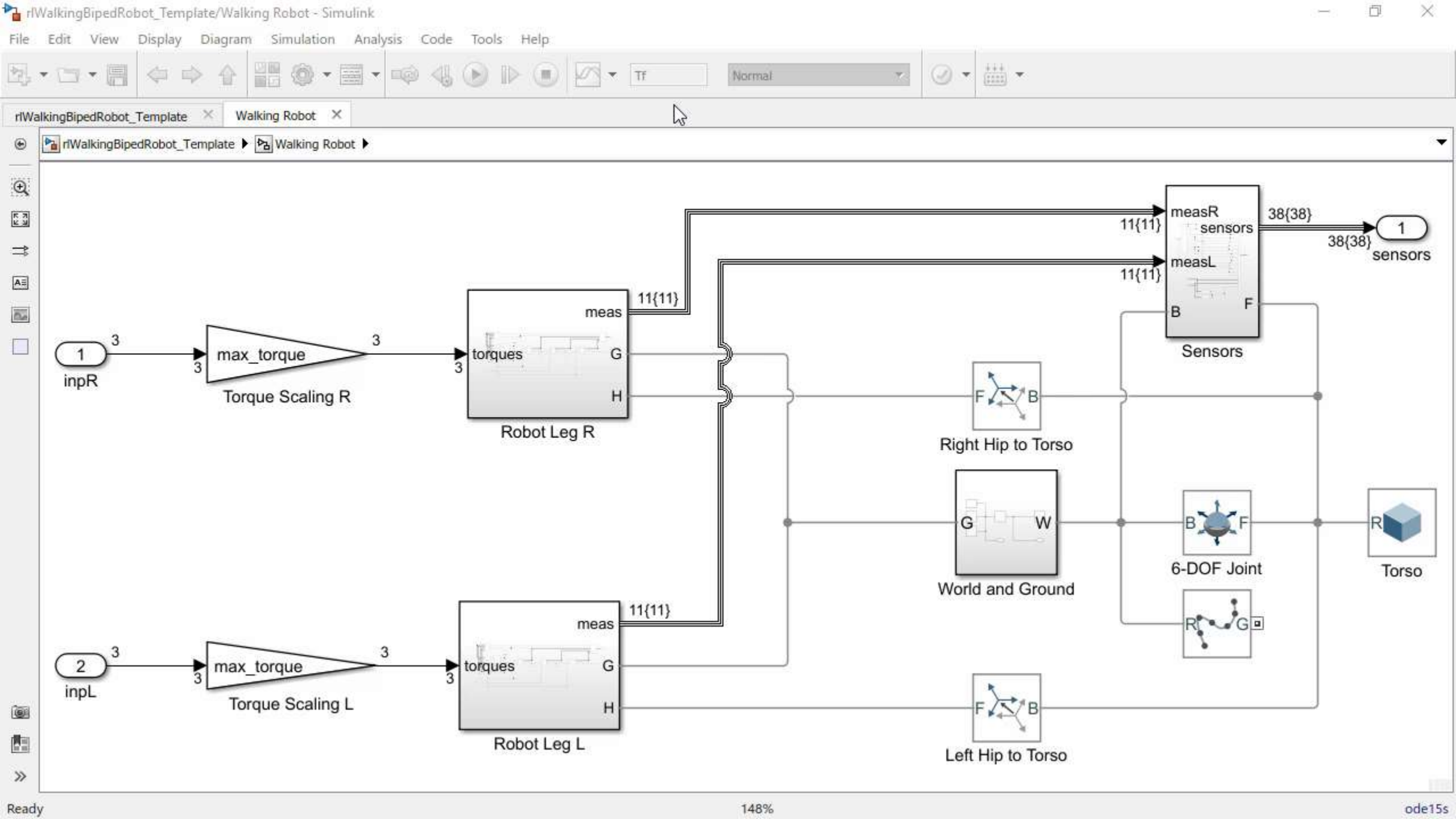
By representing policies using deep neural networks, we can solve problems for **complex, non-linear systems** (continuous or discrete) by directly using **data that traditional approaches cannot use easily**

How do I set it up and solve it?

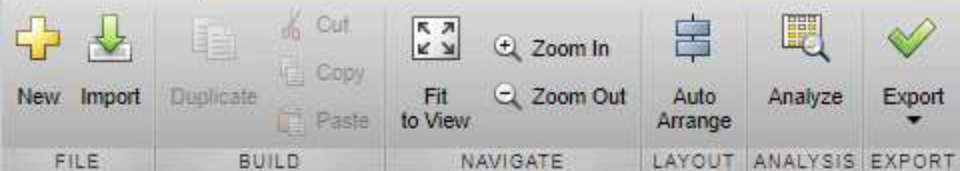


Reinforcement Learning Workflow





DESIGNER



LAYER LIBRARY

INPUT

- imageInputLayer
- image3dInputLayer
- sequenceInputLayer
- roiInputLayer

CONVOLUTION AND FULLY CONNECTED

- convolution2dLayer
- convolution3dLayer
- groupedConvolution2dLayer
- transposedConv2dLayer
- transposedConv3dLayer
- fullyConnectedLayer

SEQUENCE

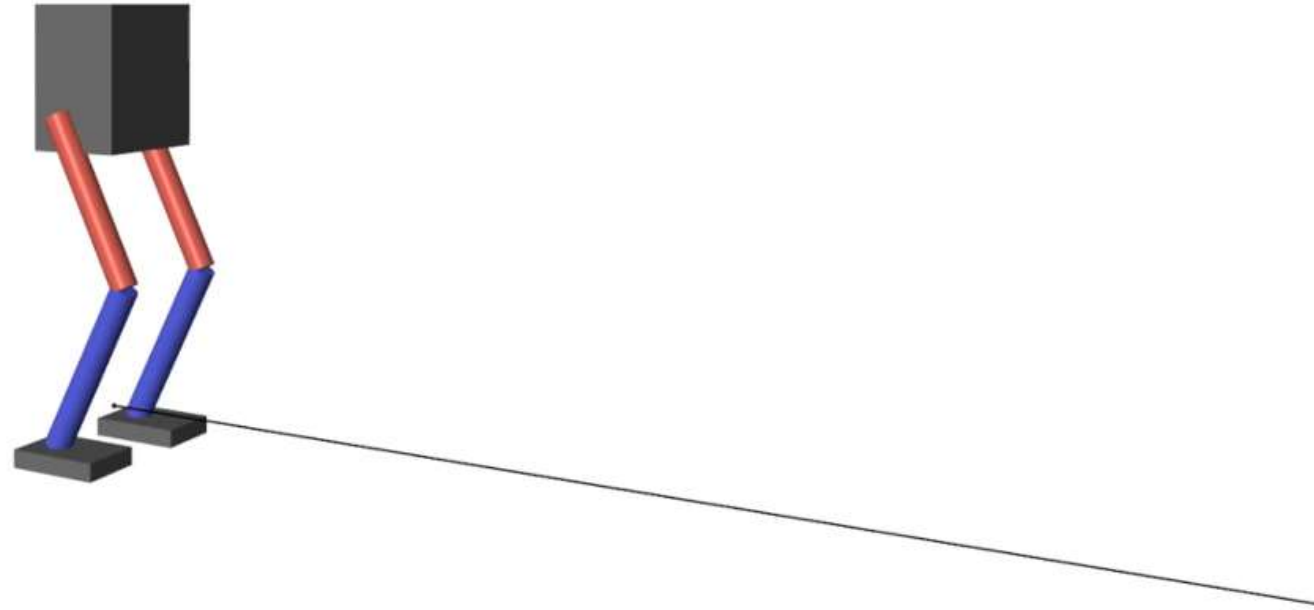
- IstmLayer
- hiIstmLayer

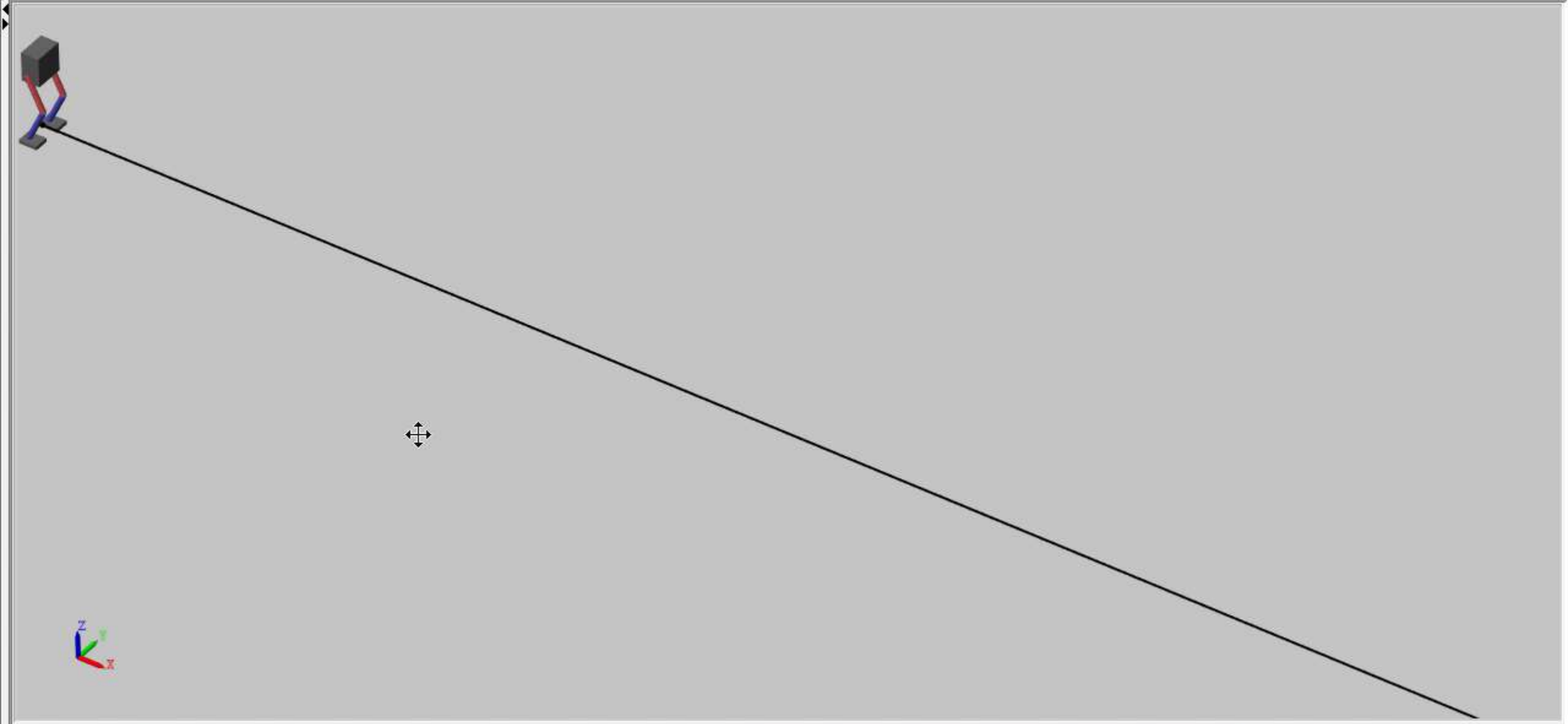


PROPERTIES

Number of layers	7
Number of connections	6
Input type	Image
Output type	None

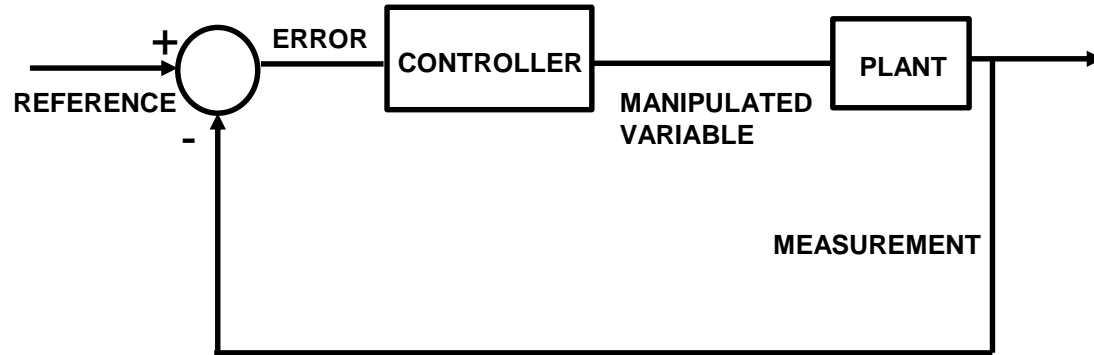
Steps in the Reinforcement Learning Workflow





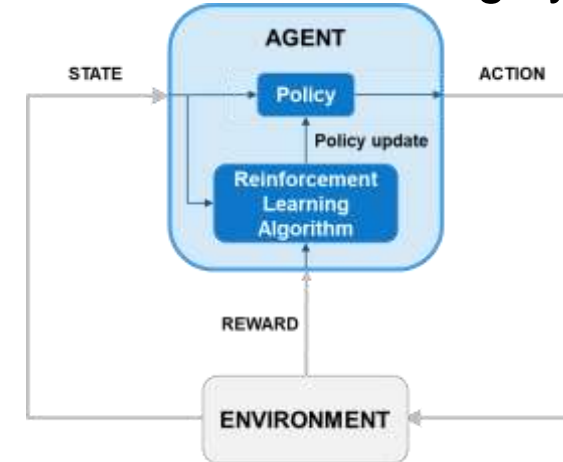
Reinforcement Learning vs Controls

Control system



- Adaptation mechanism
- Error/Cost function
- Manipulated variable
- Measurement
- Plant
- Controller

Reinforcement learning system



- RL Algorithm
- Reward
- Action
- Observation
- Environment
- Policy

Reinforcement learning has parallels to **control system design**



Pop Quiz: When would you use Reinforcement Learning?

	Controller Capability	Computational Cost in Training/Tuning	Computational Cost in Deployment
PID	Low	Low	Low
Model Pred Control	High	Low	High
Reinforcement Learning	High	High	Medium

Reinforcement learning might be a good fit if

- An environment model is available (trial & error on hardware can be expensive), and
- Training/tuning time is **not** critical for the application, and
- Uncertain environments or nonlinear environments

Automotive Applications

- Controller Design
- Lane Keep Assist
- Adaptive Cruise Control
- Path Following Control
- Trajectory Planning

Train DDPG Agent to Control Flying Robot

Train a reinforcement learning agent to control a flying robot model.

Train Biped Robot to Walk Using DDPG Agent

Train a reinforcement learning agent to control a biped walking robot modeled in Simscape Multibody.

Train DDPG Agent for Adaptive Cruise Control

Train a reinforcement learning agent for an adaptive cruise control application.

Train DQN Agent for Lane Keeping Assist

Train a reinforcement learning agent for a lane keeping assist application.

Train DDPG Agent for Path Following Control

Train a reinforcement learning agent for a lane following application.

Reinforcement Learning Toolbox

New in R2019a

- Built-in and custom algorithms for reinforcement learning
- Environment modeling in MATLAB and Simulink
- Deep Learning Toolbox support for designing policies
- Training acceleration through GPUs and cloud resources
- Deployment to embedded devices and production systems
- Reference examples for getting started



Reinforcement Learning Toolbox™ provides functions and blocks for training policies using reinforcement learning algorithms including DQN, A2C, and DDPG. You can use these policies to implement controllers and decision-making algorithms for complex systems such as robots and autonomous systems. You can implement the policies using deep neural networks, polynomial, or look-up tables.

The toolbox lets you train policies by enabling them to interact with environments represented by MATLAB® or Simulink® models. You can evaluate algorithms, experiment with hyperparameter settings, and monitor training progress. To improve training performance, you can run simulations in parallel on the cloud, computer clusters, and GPUs (with Parallel Computing Toolbox™ and MATLAB Parallel Server™).

Through the ONNX™ model format, existing policies can be imported from deep learning frameworks such as TensorFlow™, Keras, and PyTorch (with Deep Learning Toolbox™). You can generate optimized C, C++, and CUDA code to deploy trained policies on microcontrollers and GPUs.

The toolbox includes reference examples for using reinforcement learning to design controllers for robotics and automated driving applications.

TRAINING AND TRAINERS

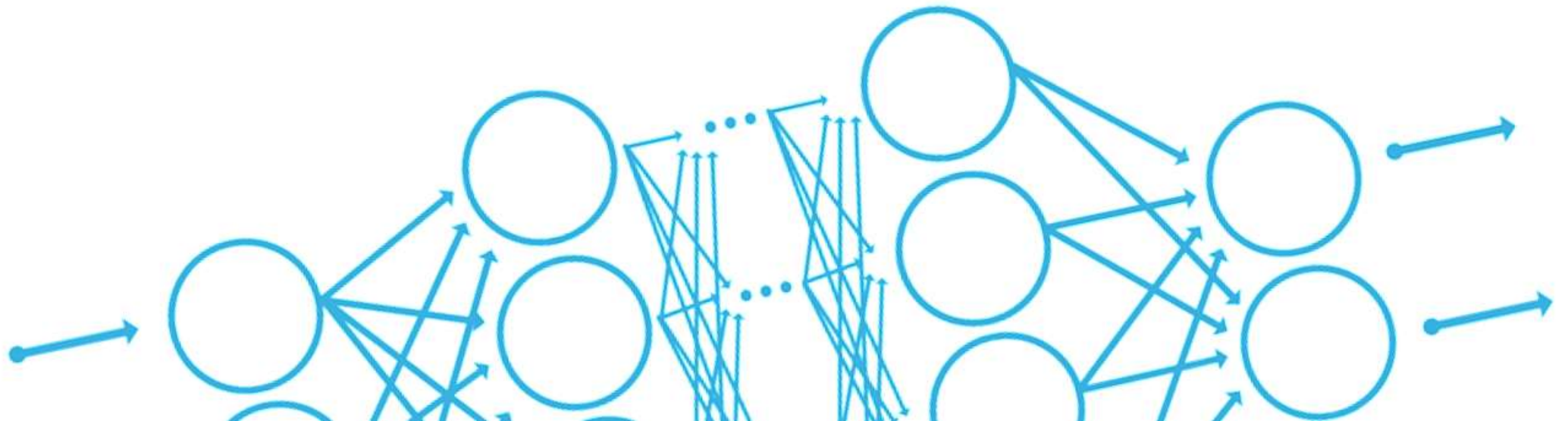
Train and simulate reinforcement learning agents

Policy Deployment

Code generation and deployment of trained policies

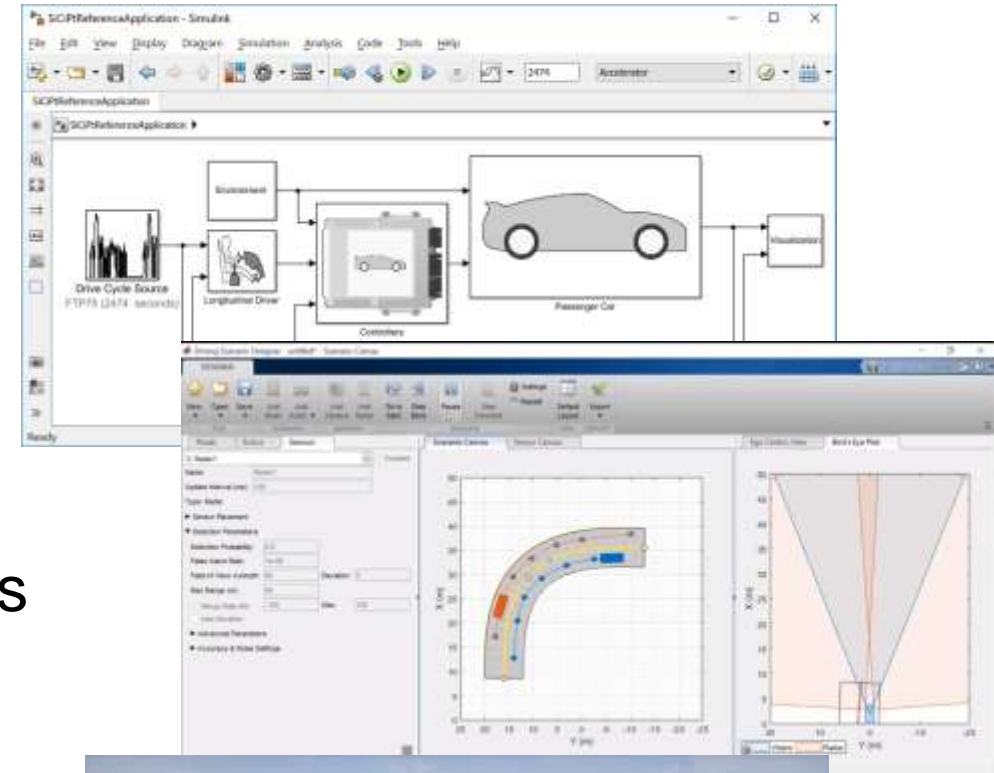


Takeaways



Simulation and Virtual Models are a Key Aspect of Reinforcement Learning

- Reinforcement learning needs **a lot** of data (*sample inefficient*)
 - Training on hardware can be prohibitively expensive and dangerous
- Virtual models allow you to simulate conditions hard to emulate in the real world
 - This can help develop a more robust solution
- Many of you have already developed MATLAB and Simulink models that can be reused



Pros & Cons of Reinforcement Learning

Pros	Cons
No need to collect data before training	A lot of simulation trials required
Opens up AI applications intractable today	Training may not converge
Complex end-to-end solutions can be developed (e.g. camera input → car steering wheel)	Reward signal design, network layer structure & hyperparameter tuning can be challenging
Suitable for uncertain, nonlinear environments	No performance guarantees
Virtual models allow simulations of varying conditions and training parallelization	Further training might be necessary after deployment on real hardware
<p>Everyone is excited about it as it appears to be a silver bullet for all problems</p>	

Resources

- Examples for automotive and autonomous system applications
- Documentation written for engineers and domain experts
- Tech Talk video series on reinforcement learning concepts for engineers

Train DDPG Agent to Control Flying Robot
Train a reinforcement learning agent to control a flying robot model.

Train Biped Robot to Walk Using DDPG Agent
Train a reinforcement learning agent to control a biped walking robot modeled in Simulink.

Train DDPG Agent for Adaptive Cruise Control
Train a reinforcement learning agent to control an adaptive cruise control application.

Train DDPG Agent for Lane Assist
Train a reinforcement learning agent to control a lane assist application.

Train DDPG Agent for Path Following Control
Train a reinforcement learning agent to control a path following application.

Reinforcement Learning Toolbox
Design and train policies using reinforcement learning.

Reinforcement Learning Toolbox™ provides functions and blocks for training policies using reinforcement learning algorithms including DQN, A2C, and DDPG. You can use these policies to implement controllers and decision-making algorithms for complex systems such as robots and autonomous systems. You can implement the policies using deep neural networks, polynomials, or look-up tables.

The toolbox lets you train policies by enabling them to interact with environments represented by MATLAB® or Simulink® models. You can evaluate algorithms, experiment with hyperparameter settings, and monitor training progress. To improve training performance, you can run simulations in parallel on the cloud, computer clusters, and GPUs (with Parallel Computing Toolbox™ and MATLAB Parallel Server™).

Through the ONNX™ model format, existing policies can be imported from deep learning frameworks such as TensorFlow™, Keras, and PyTorch (with Deep Learning Toolbox™). You can generate optimized C, C++, and CUDA code to deploy trained policies on microcontrollers and GPUs.

The toolbox includes reference examples for using reinforcement learning to design controllers for robotics and automated driving applications.

Getting Started
Learn the basics of Reinforcement Learning Toolbox

MATLAB Environments
Model reinforcement learning environment dynamics using MATLAB

Simulink Environments
Model reinforcement learning environment dynamics using Simulink models

Policies and Value Functions
Define policy and value function representations, such as deep neural networks and Q tables

Agents
Create and configure reinforcement learning agents using common algorithms, such as SARSA

Training and Validation
Train and simulate reinforcement learning agents

Policy Deployment
Code generation and deployment of trained policies

Agent-Environment Interaction Diagram:
The diagram shows the interaction between an Agent and an Environment. The Agent contains a Policy block and Reinforcement learning algorithms. The Environment contains a robot arm. Arrows indicate the flow of information: 'actuator commands' (Action) from Agent to Environment, 'Observation (state)' from Environment to Agent, and 'Reward' from Environment to Agent.

Extra Slides

