

# MATLAB과 함께하는 딥러닝 4주 완성 부트캠프

김종남 부장

Application Engineer @ MathWorks

[calebkim@mathworks.com](mailto:calebkim@mathworks.com)



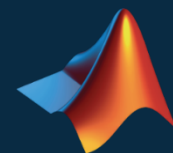
# 세션4. MATLAB으로 시작하는 강화학습

MATLAB과 함께하는 딥러닝 4주 완성 부트캠프

김종남 부장

Application Engineer @ MathWorks

[calebkim@mathworks.com](mailto:calebkim@mathworks.com)



MathWorks®

*Accelerating the pace of engineering and science*

# Agenda

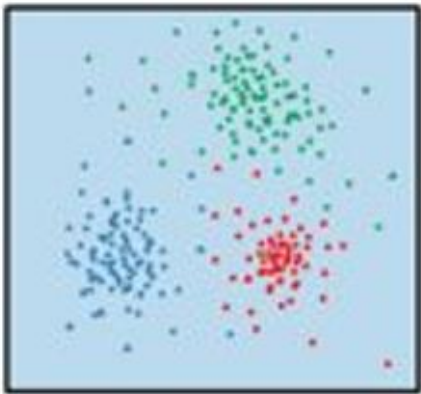
- **How to Train a Robot to Walk**
  - What is reinforcement learning?
  - Overview of using a traditional controls approach
  - Applying the reinforcement learning workflow to train the robot with Reinforcement Learning Designer

# Reinforcement Learning: A Subset of Machine Learning

## machine learning

unsupervised learning

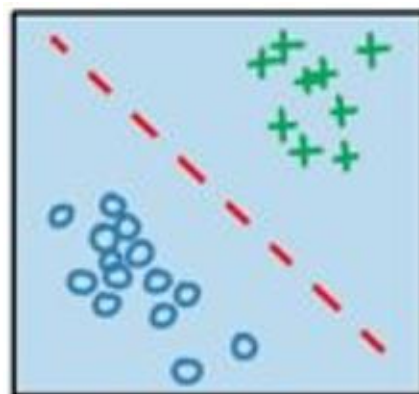
[unlabeled data]



clustering

supervised learning

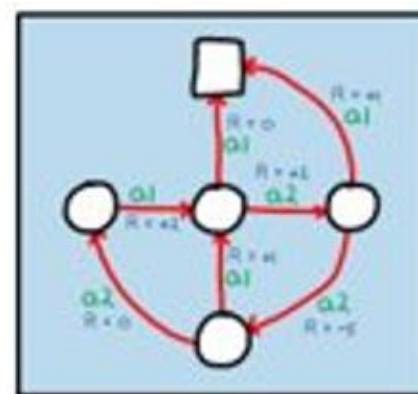
[labeled data]



classification and regression

reinforcement learning

[interaction data]

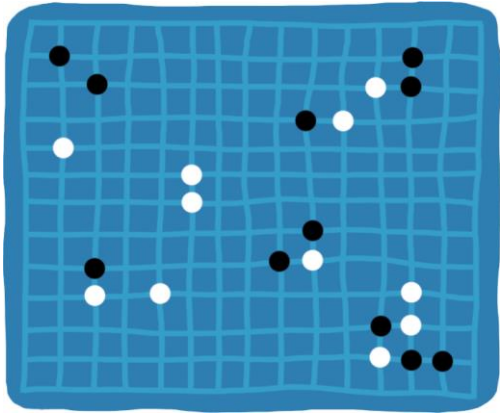


control and decision making

## Reinforcement learning:

- Learning a **behavior** or accomplishing a **task** through trial & error  
[*interaction*]
- Complex problems typically need deep models  
[*Deep Reinforcement Learning*]

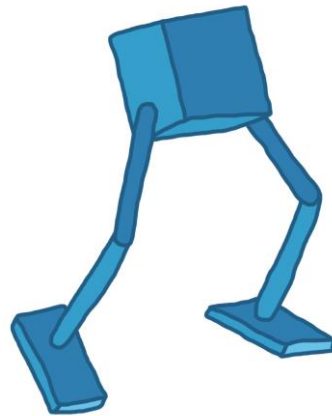
# Reinforcement Learning Applications



video games



autonomous vehicles



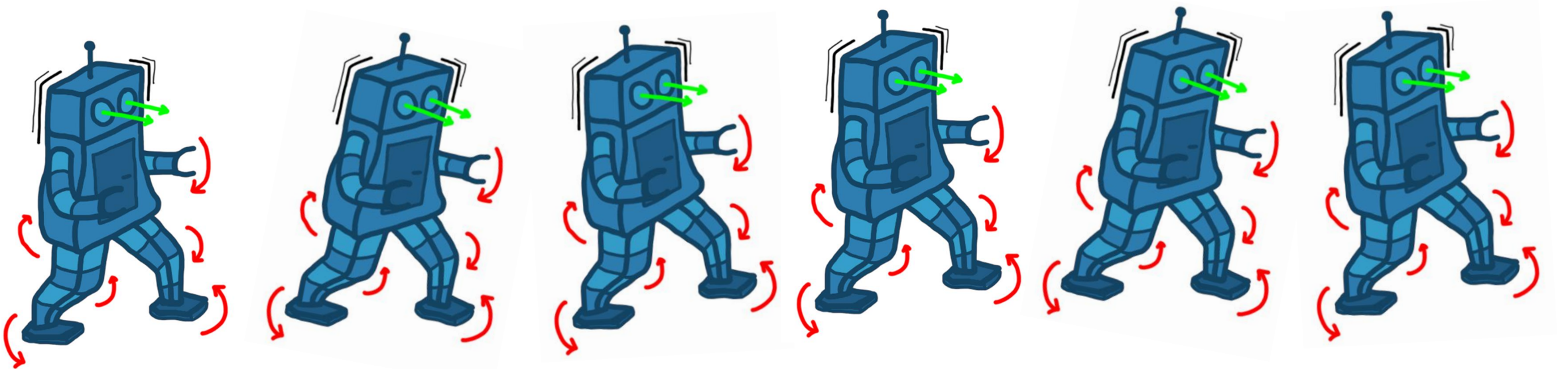
robotics



controls

# How do We Train a Robot to Walk

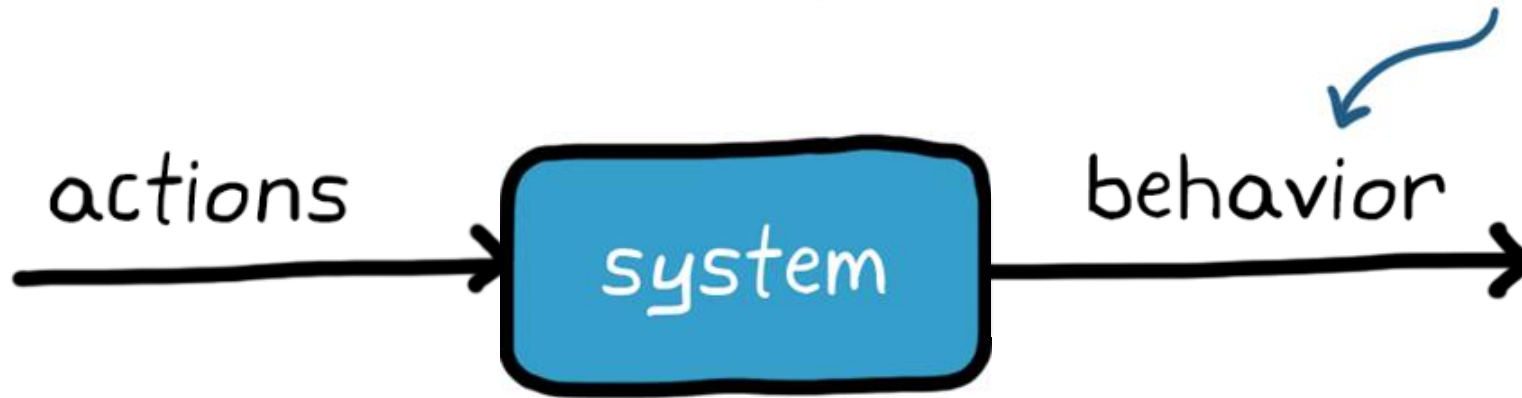
Goal: Train a robot to walk a straight line



What sequence of motor commands do we need to make the robot walk?

# The goal of control

which actions generate the desired behavior?

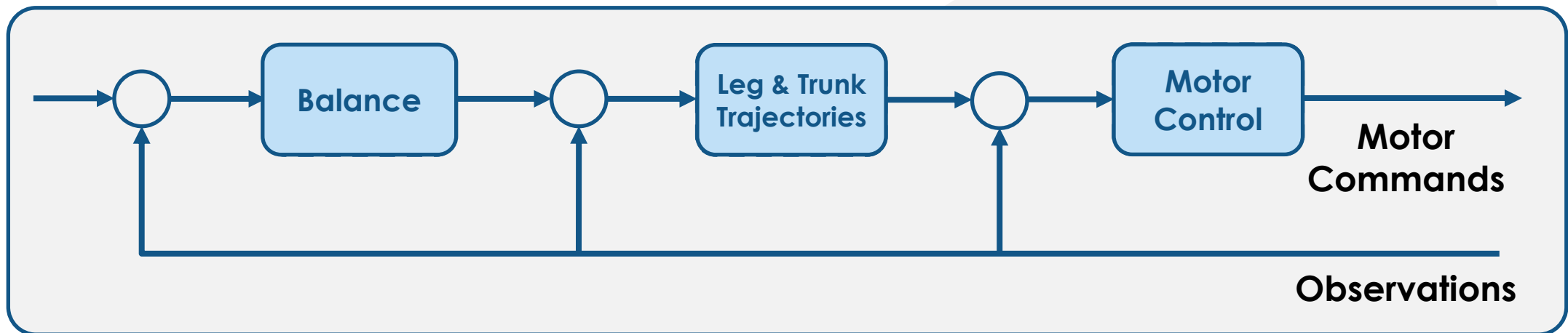
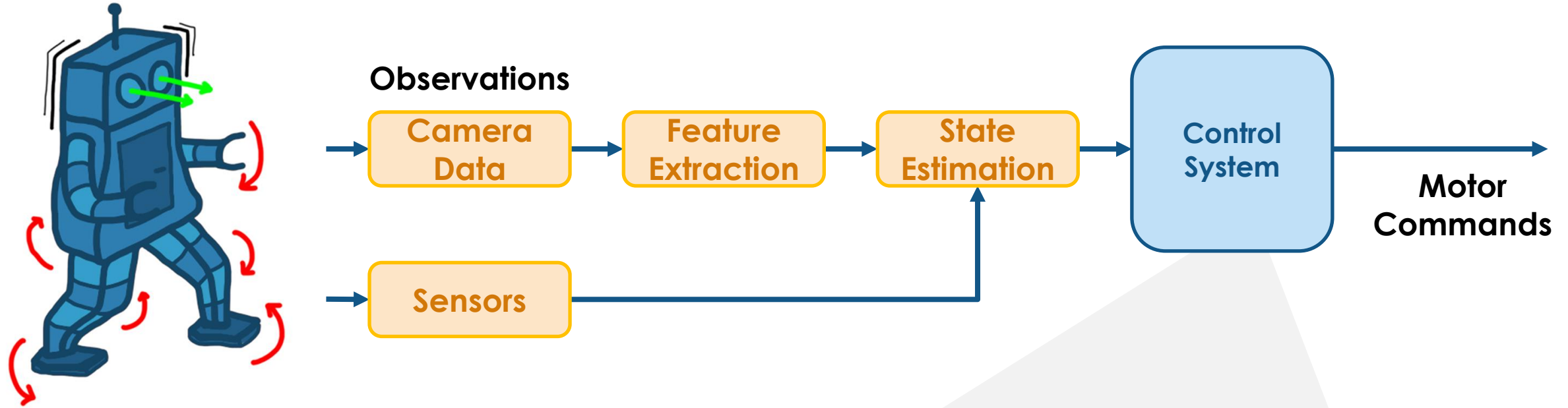


# The goal of control

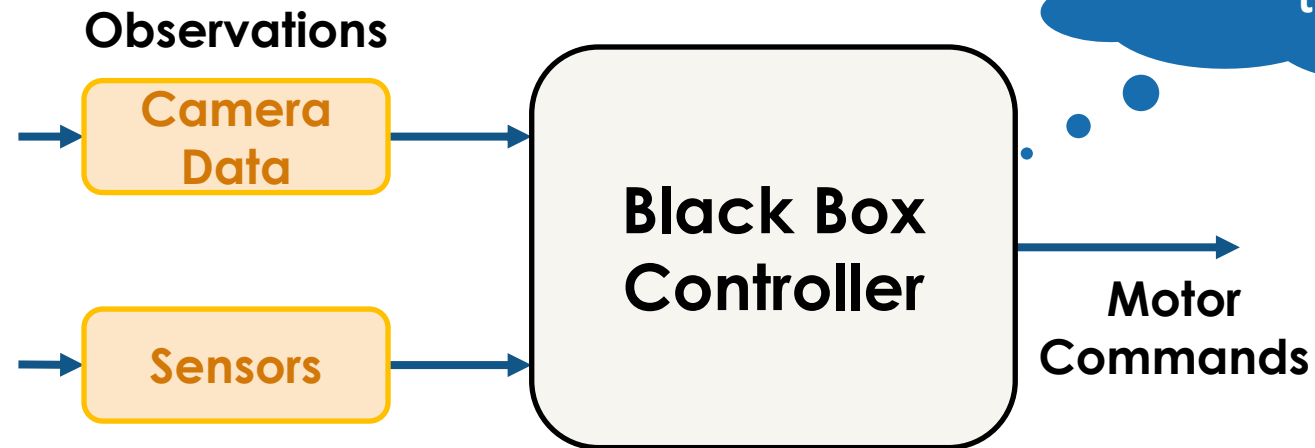
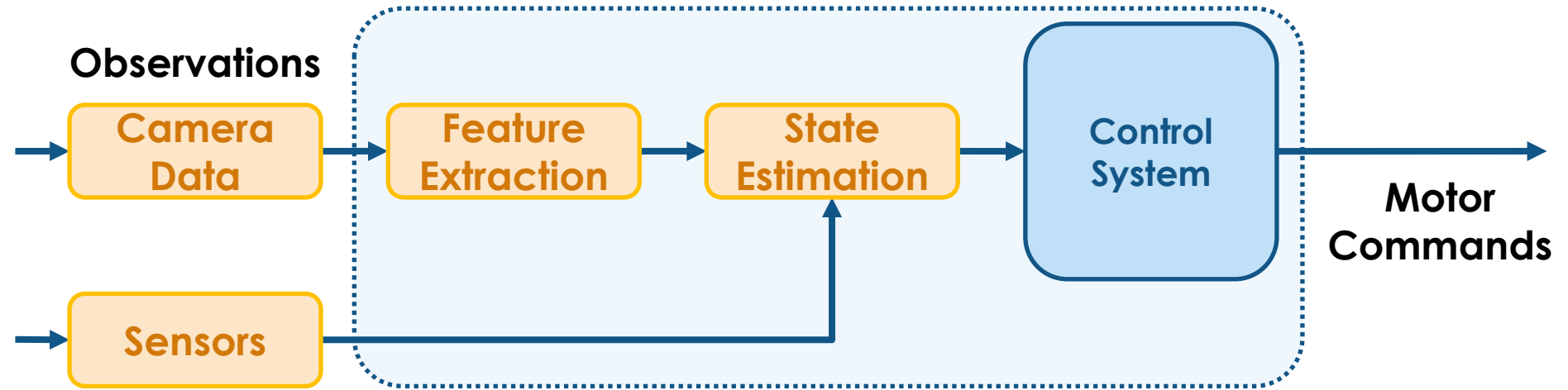




# A walking robot – a traditional controls approach



# A walking robot – an alternative approach



# What Is Reinforcement Learning?

“

Reinforcement learning is learning what to do—how to map situations to actions—so as to maximize a numerical reward signal.

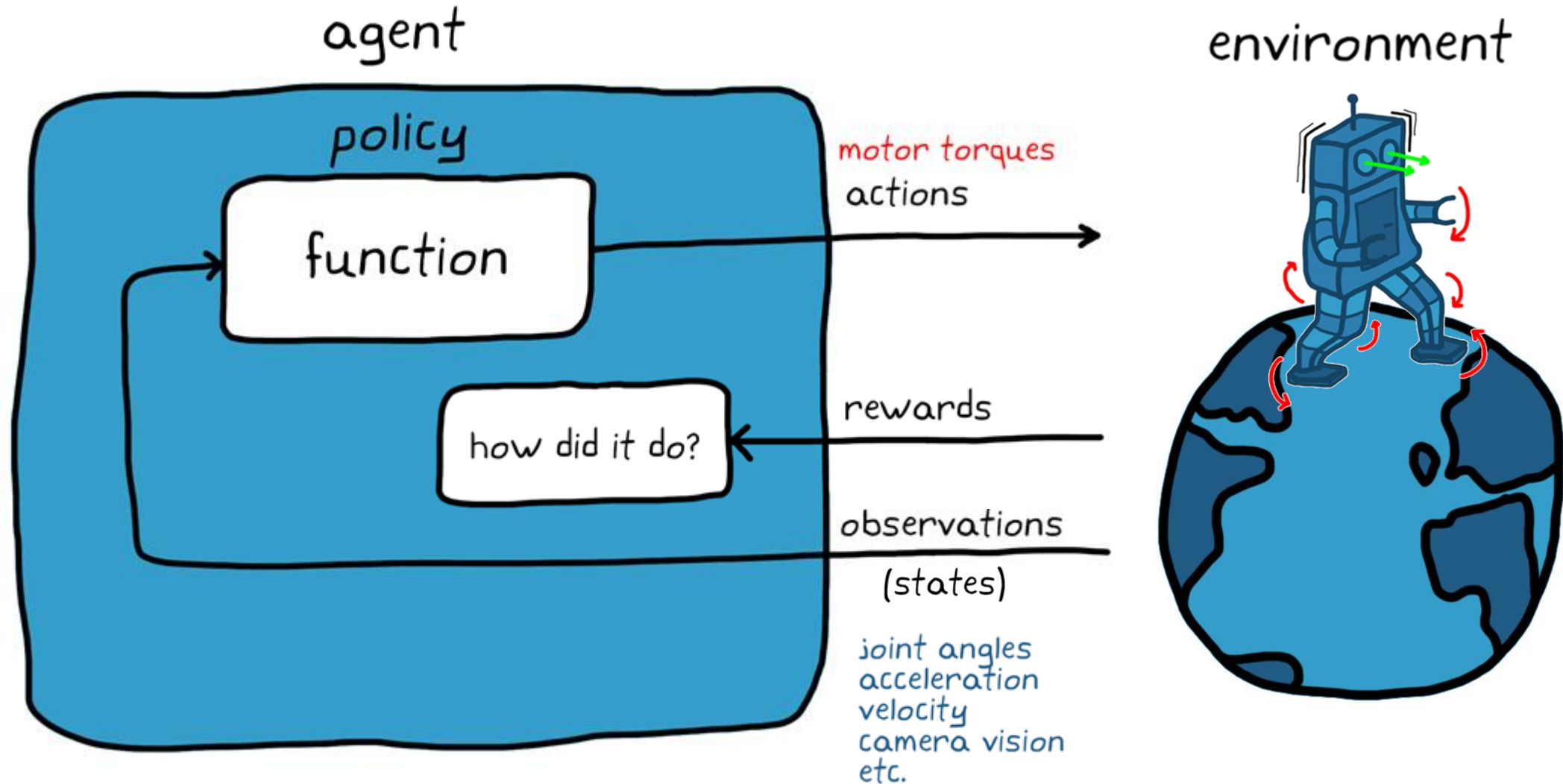
The learner is not told which actions to take, but instead must discover which actions yield the most reward by trying them.

”



Sutton and Barto,  
[\*Reinforcement Learning: An Introduction\*](#)

# Some Reinforcement Learning Terminology



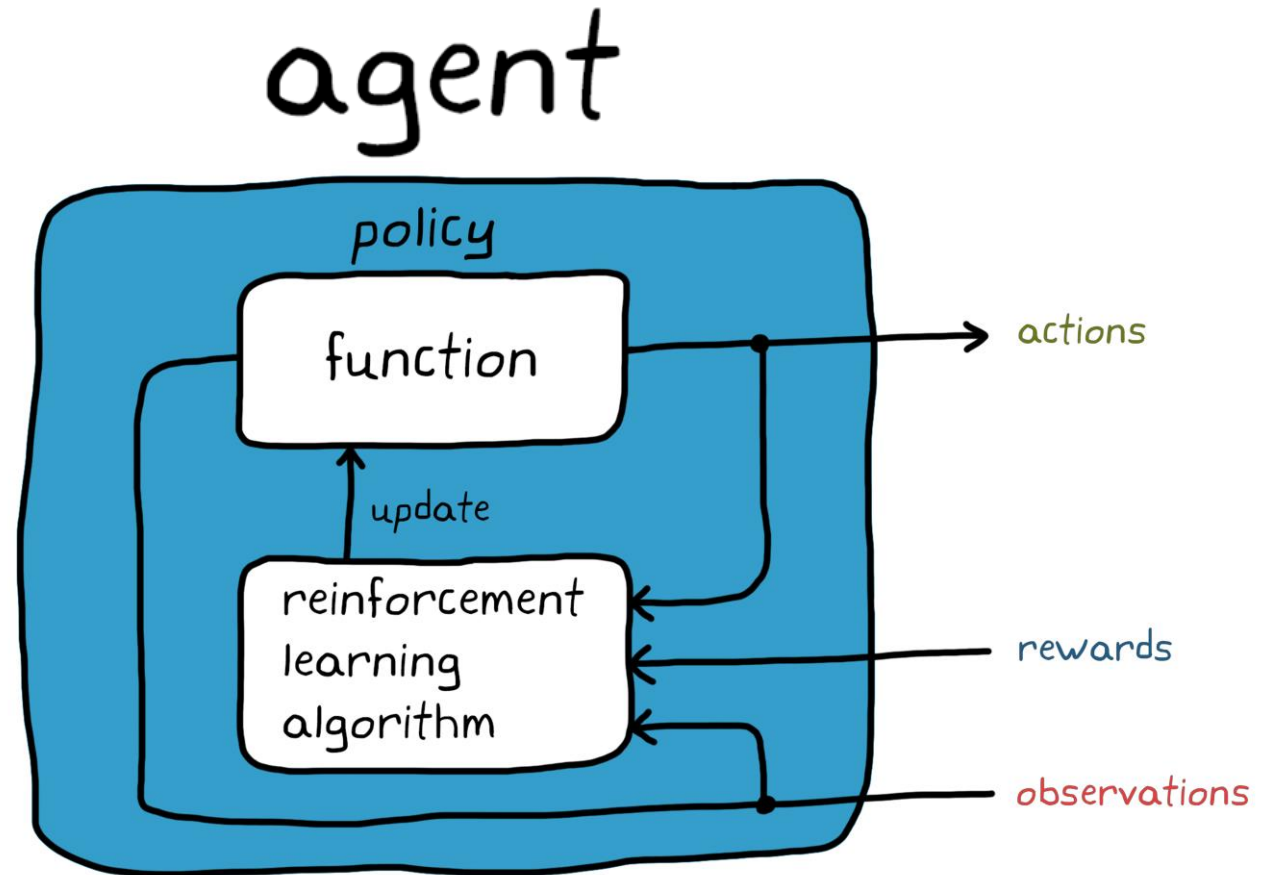
# Learning the Optimal Policy

## Policy

function that maps  
**observations** to **actions**

## Reinforcement Learning Algorithm

optimization method used to find the optimal policy that maximizes accumulative long-term reward



# Reinforcement Learning Workflow

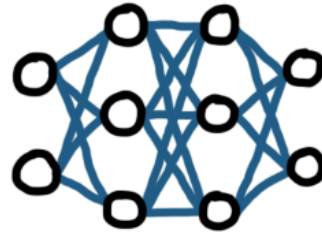
environment



reward



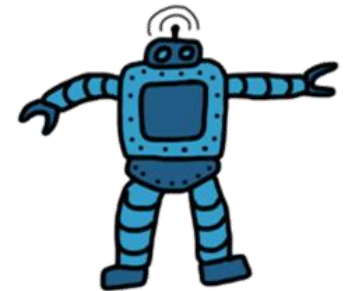
policy



training



deploy



# Reinforcement Learning Workflow

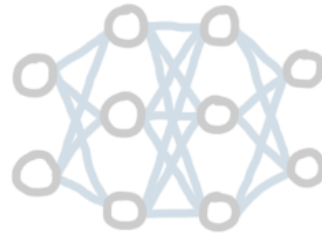
environment



reward



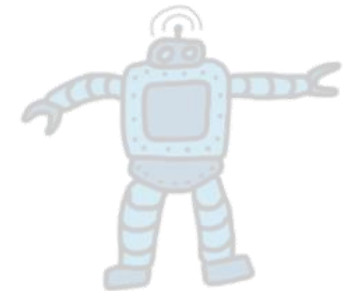
policy



training



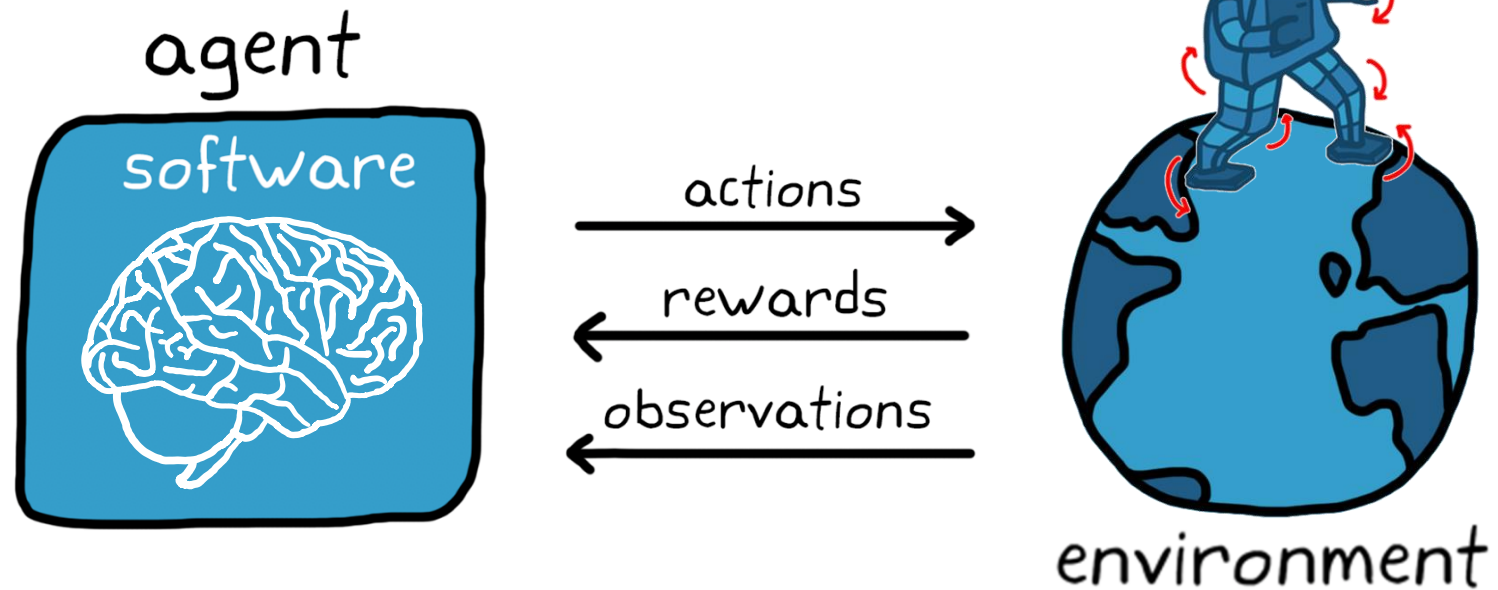
deploy



# Environment



- Everything outside of an agent

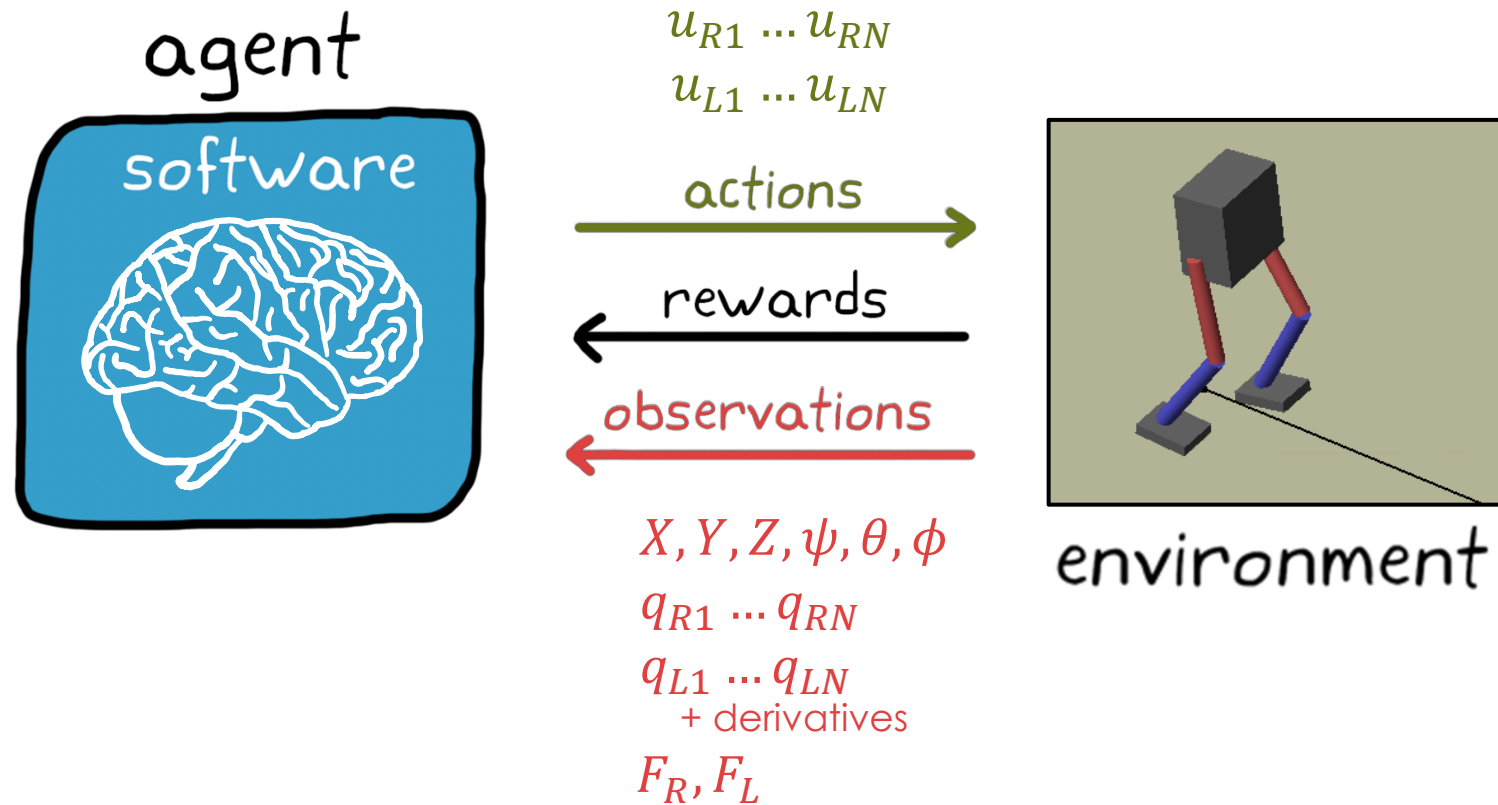




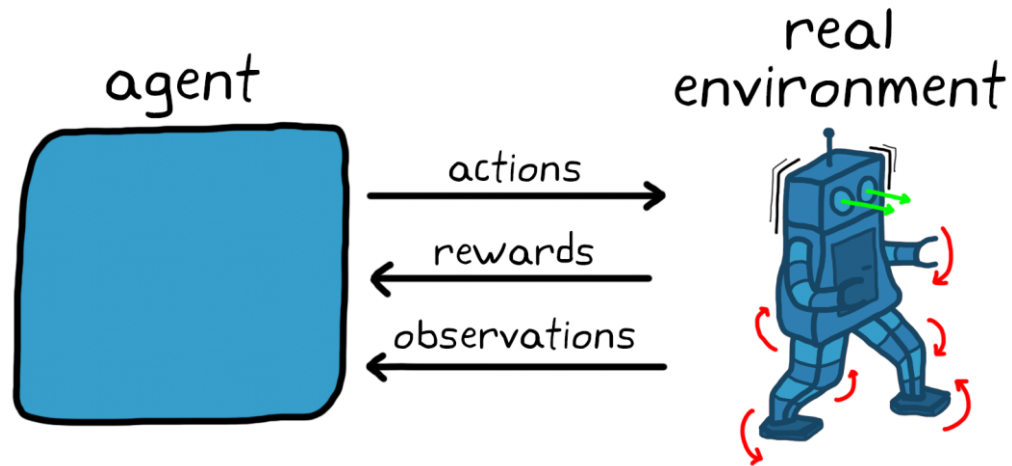
# Environment



- Everything outside of an agent

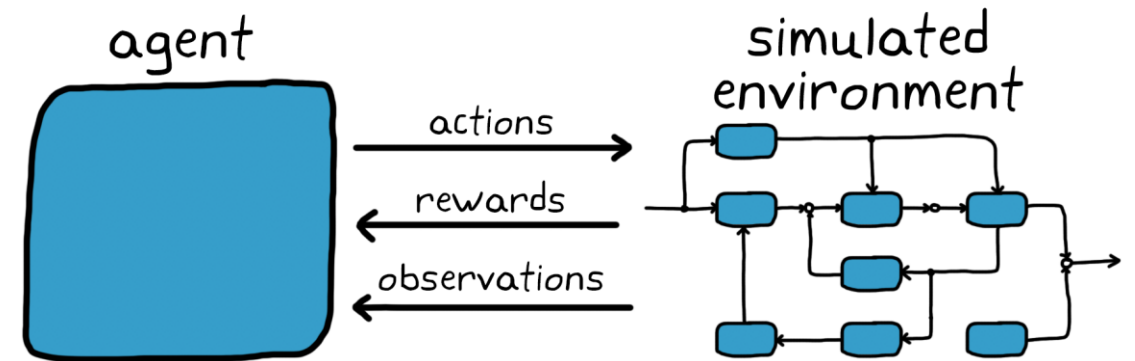


# Real vs Simulated Environments



😊 Accuracy

😞 Risk



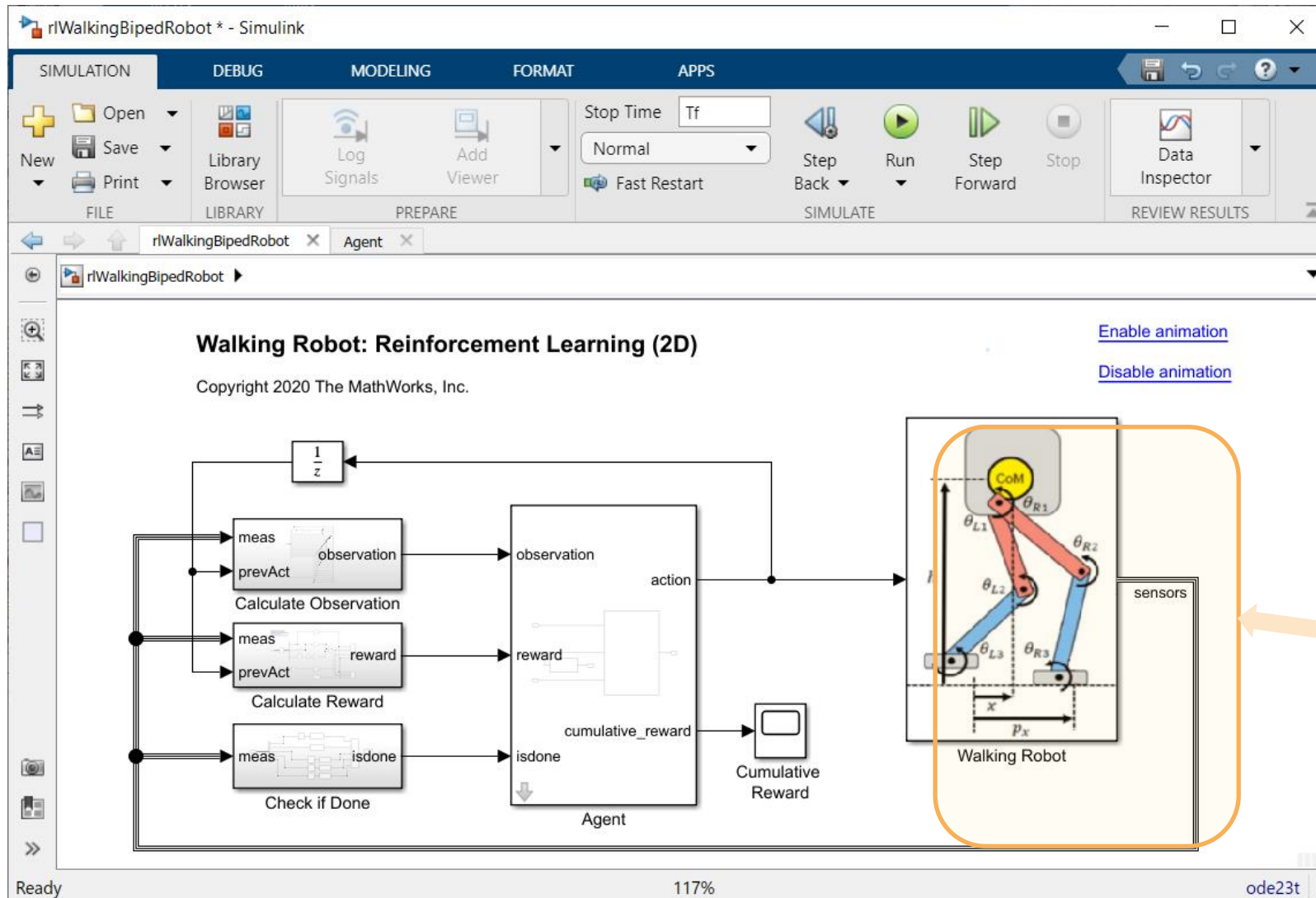
😊 Training speed

😊 Flexible simulated conditions

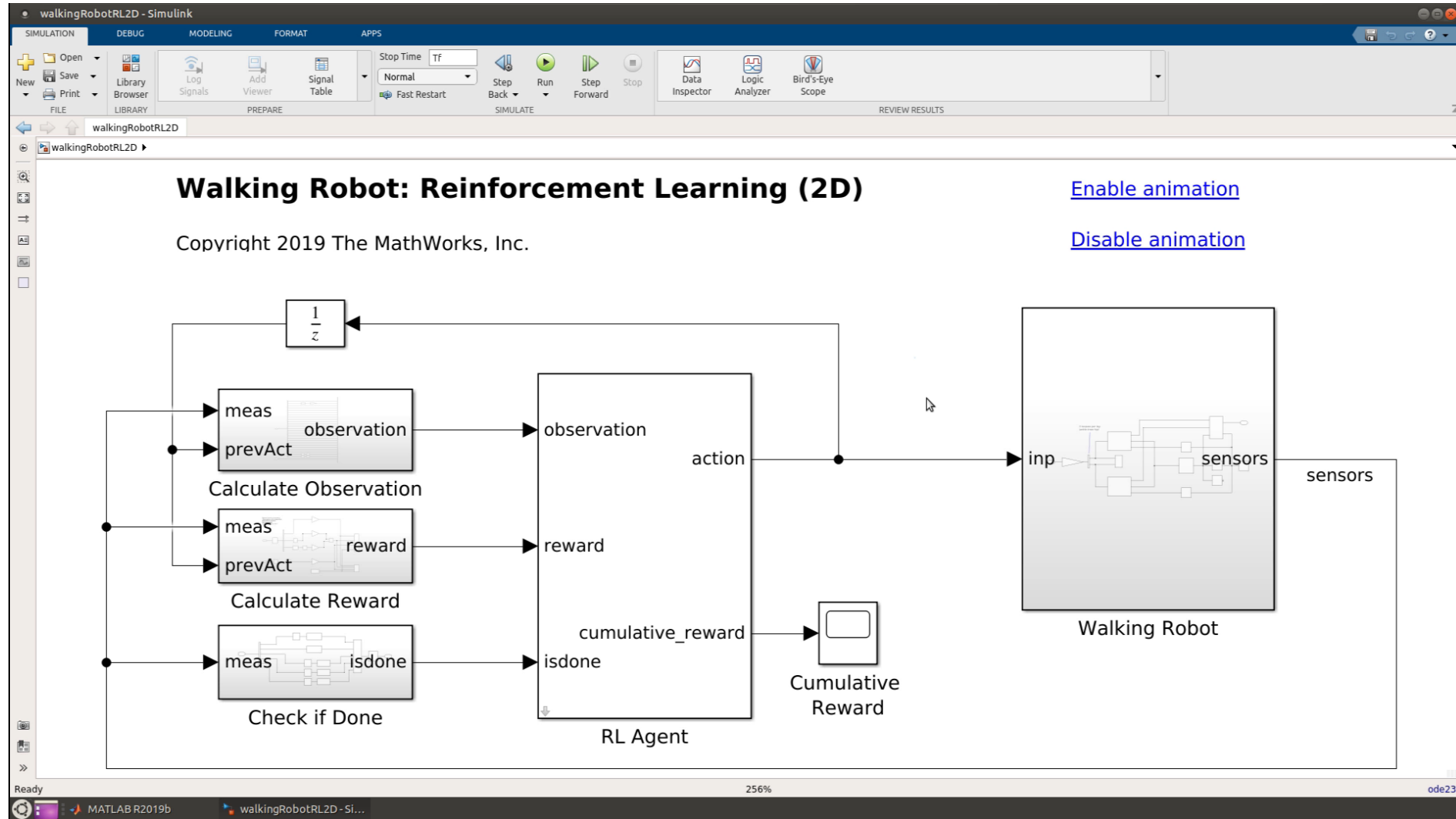
😊 Safety

😞 Model inaccuracies

# Define Simulated Environment



# Environment - Simulink



# Reinforcement Learning Workflow

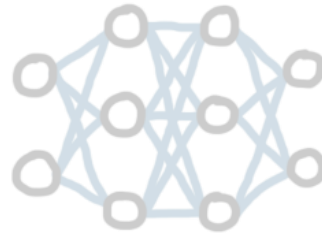
environment



reward



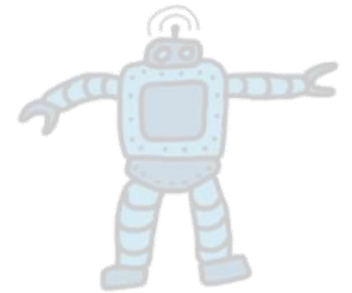
policy



training



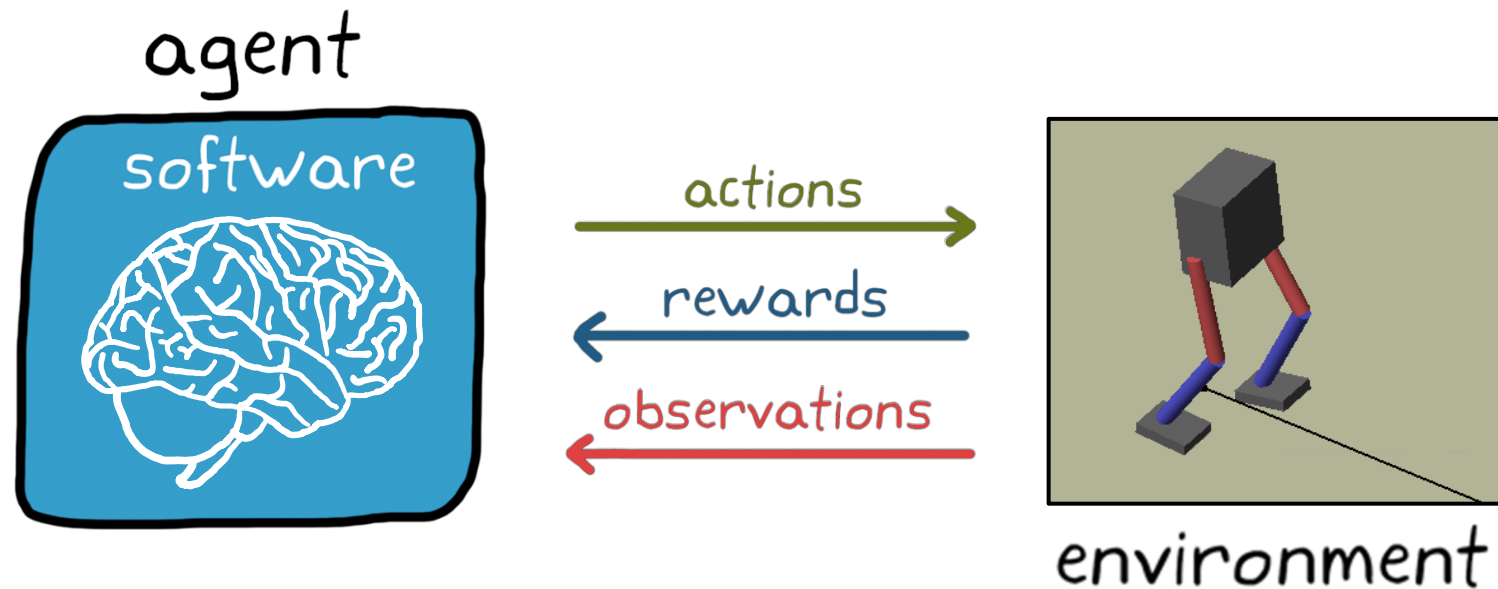
deploy



# Reward

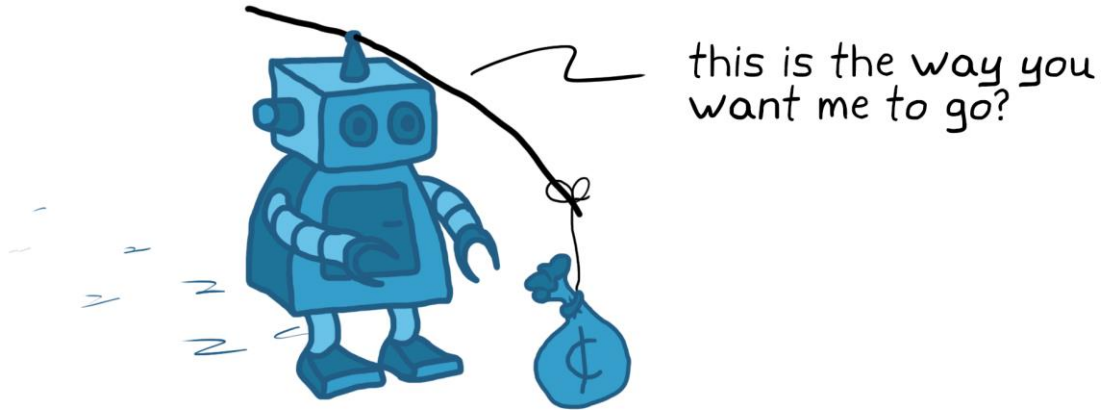


A function that outputs a **scalar number** that represents the immediate "goodness" of an agent being in a particular **state** and taking a particular **action**.



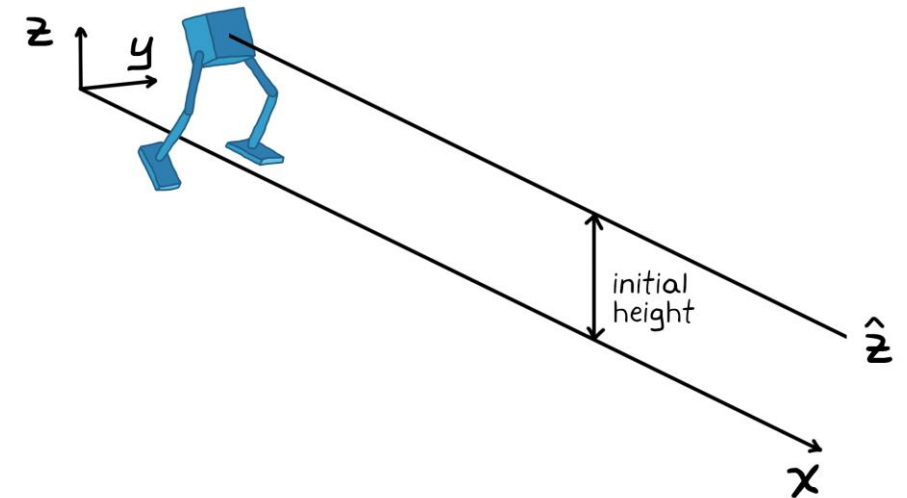
reward = function (state, action)

# Defining the Reward

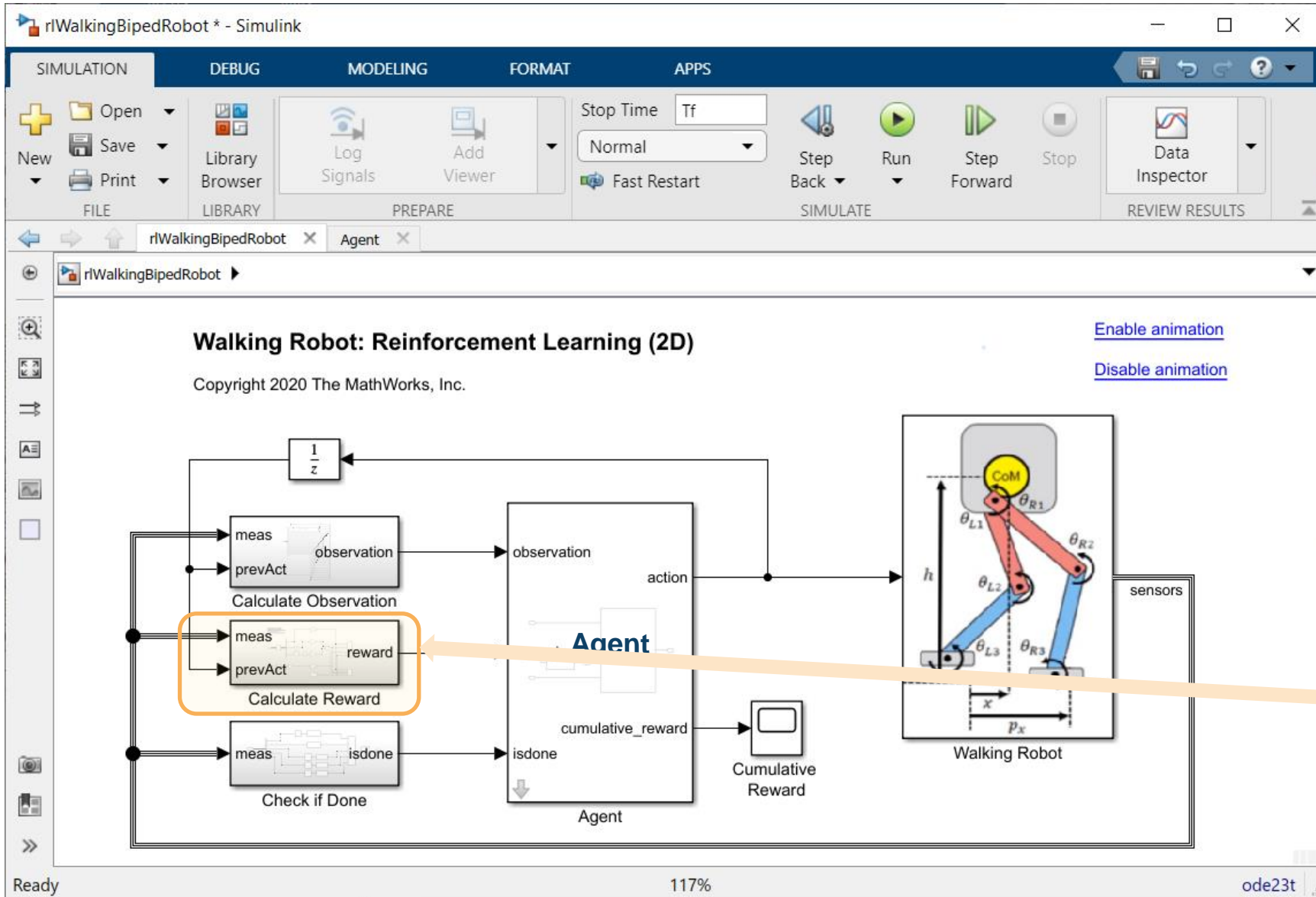


$$r_t = v_x - 3y^2 - 50\hat{z}^2 + 25\frac{T_s}{T_f} - 0.02\sum_i u_{t-1}^i{}^2$$

forward velocity  
 don't stray from path  
 keep trunk at initial height  
 walk as long as possible  
 minimize actuator effort



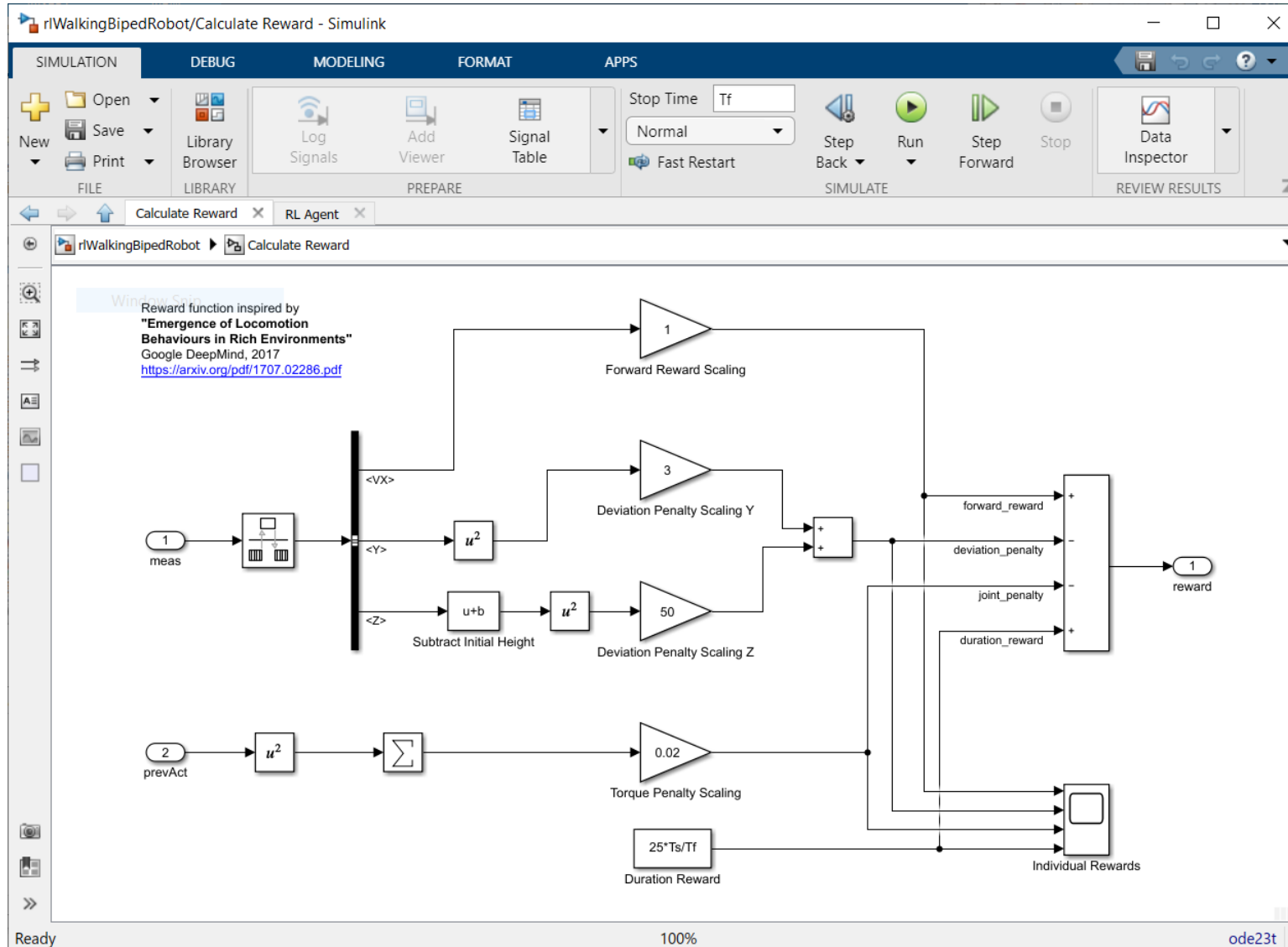
# Defining the Reward



Reward defines task to learn



# Defining the Reward



$$r_t = v_x - 3y^2 - 50z^2 + 25 \frac{T_s}{T_f} - 0.02 \sum_i u_{t-1}^2$$

forward velocity  
 don't stray from path  
 keep trunk at initial height  
 walk as long as possible  
 minimize actuator effort

# Reinforcement Learning Workflow

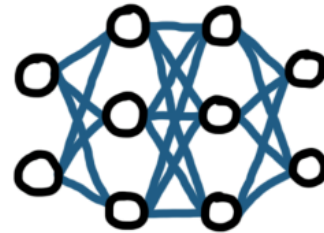
environment



reward



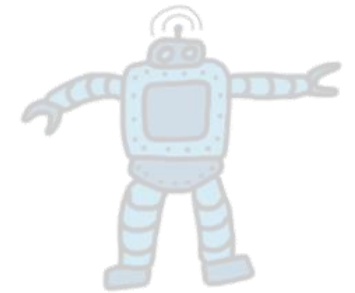
policy



training



deploy



# The Agent



# Learning the Optimal Policy

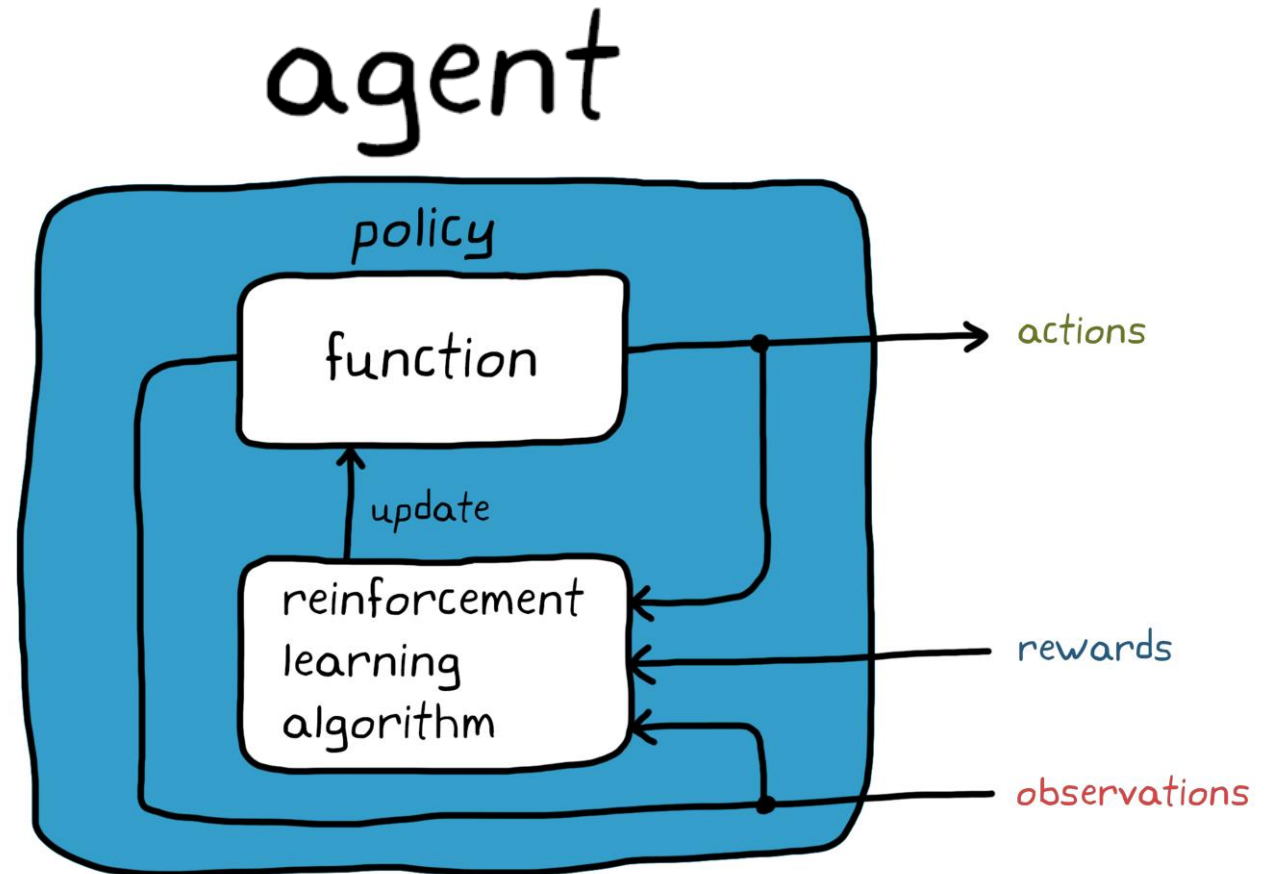


## Policy

function that maps  
**observations** to **actions**

## Reinforcement Learning Algorithm

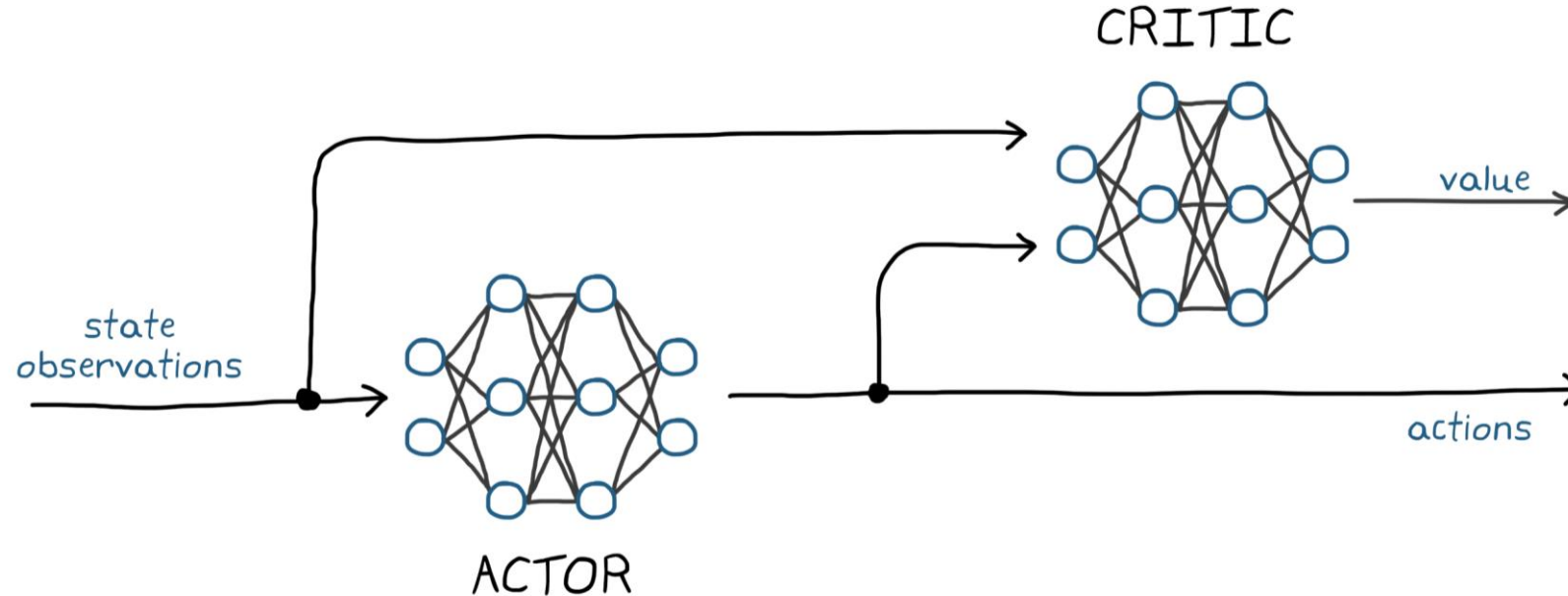
optimization method used  
to find the optimal policy



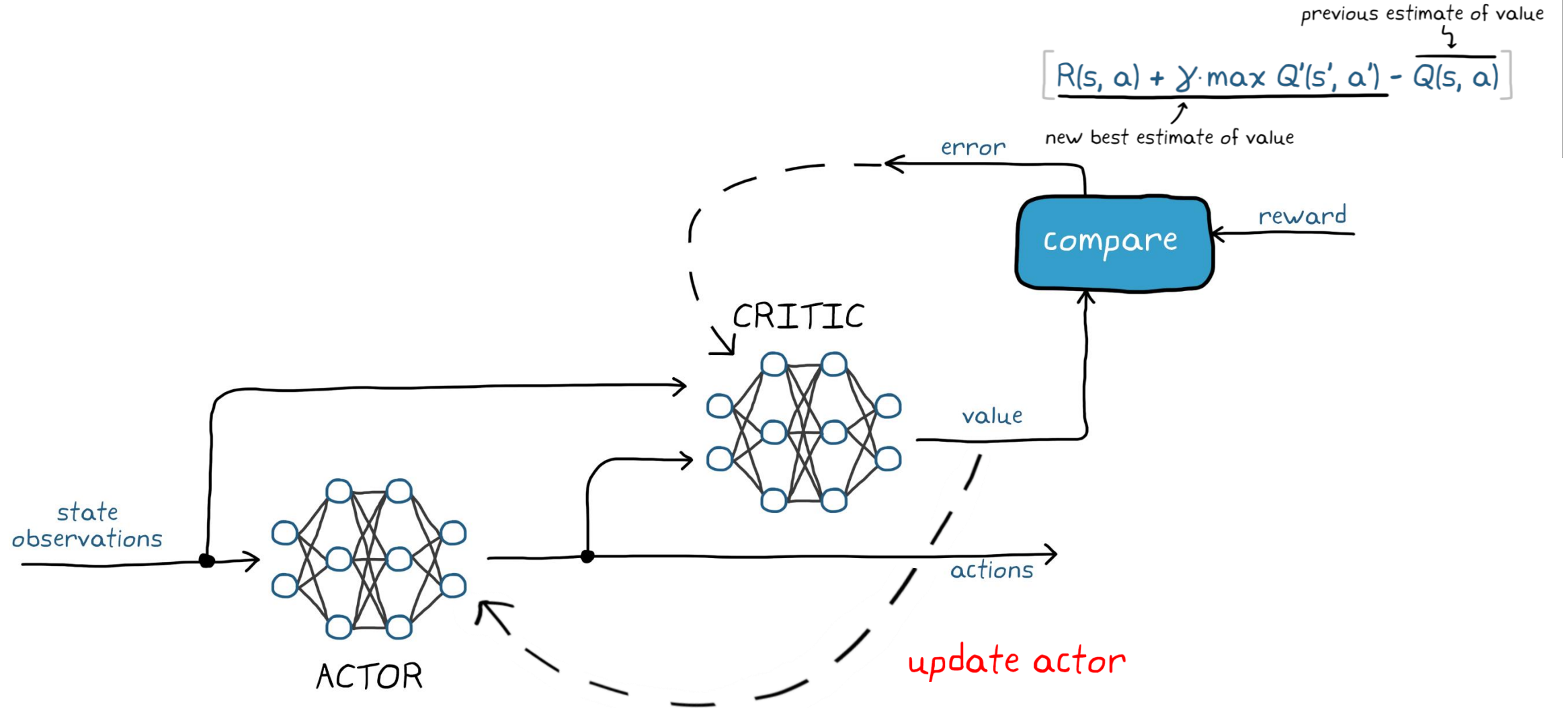
# Actor-Critic Methods



- **Two** neural networks (typically) are **simultaneously tuned** during training:
  - The actor tries to learn the best action at each state
  - The critic tries to
    - estimate the value of each state/state & action the actor takes
    - critique/guide the actor's choices



# Actor-Critic Training Cycle



# Creating the Agent



- Constructing a DDPG (Deep Deterministic Policy Gradient) Agent
  - Create the critic network
  - Create the policy network
  - Create the agent with actor and critic network and set hyperparameters

# Creating the Agent with Reinforcement Learning Designer

The screenshot displays the Reinforcement Learning Designer (RLD) software interface. The main workspace shows MATLAB code for configuring a training environment for a biped robot. The code includes parameters for observations, actions, and environment simulation.

```

4 numObs = 29;
5 obsInfo = r1NumericSpec([numObs 1]);
6 obsInfo.Name = 'observations';

Create the action specification.

7 numAct = 6;
8 actInfo = r1NumericSpec([numAct 1, 'LowerLimit', -1, 'UpperLimit', 1]);
9 actInfo.Name = 'foot_torque';

Create the environment interface for the walking robot model.

10 blk = [mdl, '/RL Agent'];
11 env = r1SimulinkEnv(mdl, blk, obsInfo, actInfo);
12 env.ResetFcn = @(in) walkerResetFcn(in, upper_leg_length/100, lower_leg_length/100, h/100);

Select and Create Agent for Training
This example provides the option to train the robot either using either a DDPG or TD3 agent. To simulate the robot with the agent of your choice, set the AgentSelection flag accordingly.

13 AgentSelection = 'DDPG';
14 switch AgentSelection
15     case 'DDPG'
16         agent = createDDPGAgent(numObs, obsInfo, numAct, actInfo, Ts);
17     case 'TD3'
18         agent = createTD3Agent(numObs, obsInfo, numAct, actInfo, Ts);

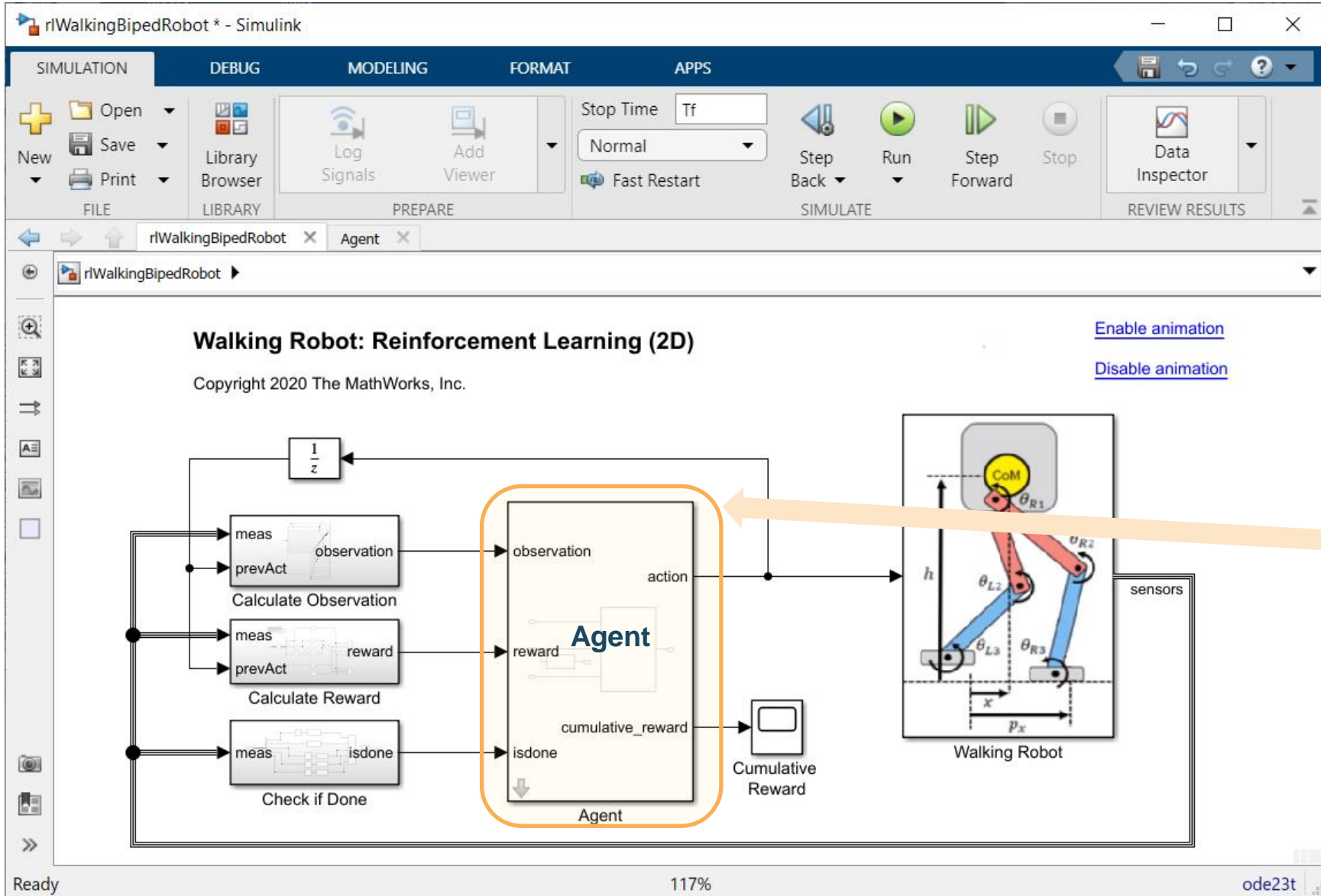
```

The right-hand pane shows a list of parameters for the robot model, including:

- actInfo: 1x1 rlNur
- actuatorType: 1
- blk: 'rlWalking'
- contact\_damping: 50
- contact\_point\_radius: 1.0000e-0
- contact\_stiffness: 500
- density: 500 [1x1 double]
- env: 1x1 Simuli
- foot\_density: 1000
- foot\_offset: [-1,0,0]
- foot\_x: 5
- foot\_y: 4
- foot\_z: 1
- g: 9.8067
- h: 18
- height\_plane: 0.0250
- init\_angs\_L: [-0.4510,0
- init\_angs\_R: [-0.4510,0
- init\_height: 21.0125
- joint\_damping: 1
- joint\_limit\_damping: 10
- joint\_limit\_stiffness: 10000
- joint\_stiffness: 0
- leftinit: [0,0;-0.18
- leg\_radius: 0.7500
- lower\_leg\_length: 10
- mass: 0.1600
- max\_torque: 3
- mdl: 'rlWalking'
- motion\_time\_constant: 0.0100



# Defining the Agent



Define the agent

# Reinforcement Learning Workflow

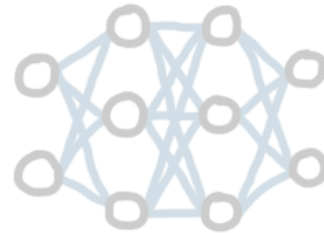
environment



reward



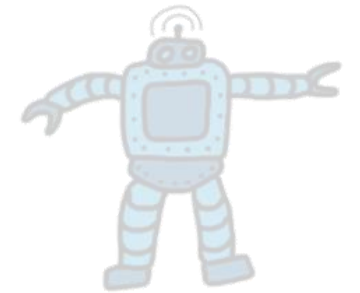
policy



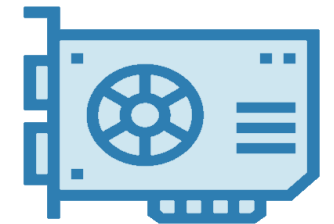
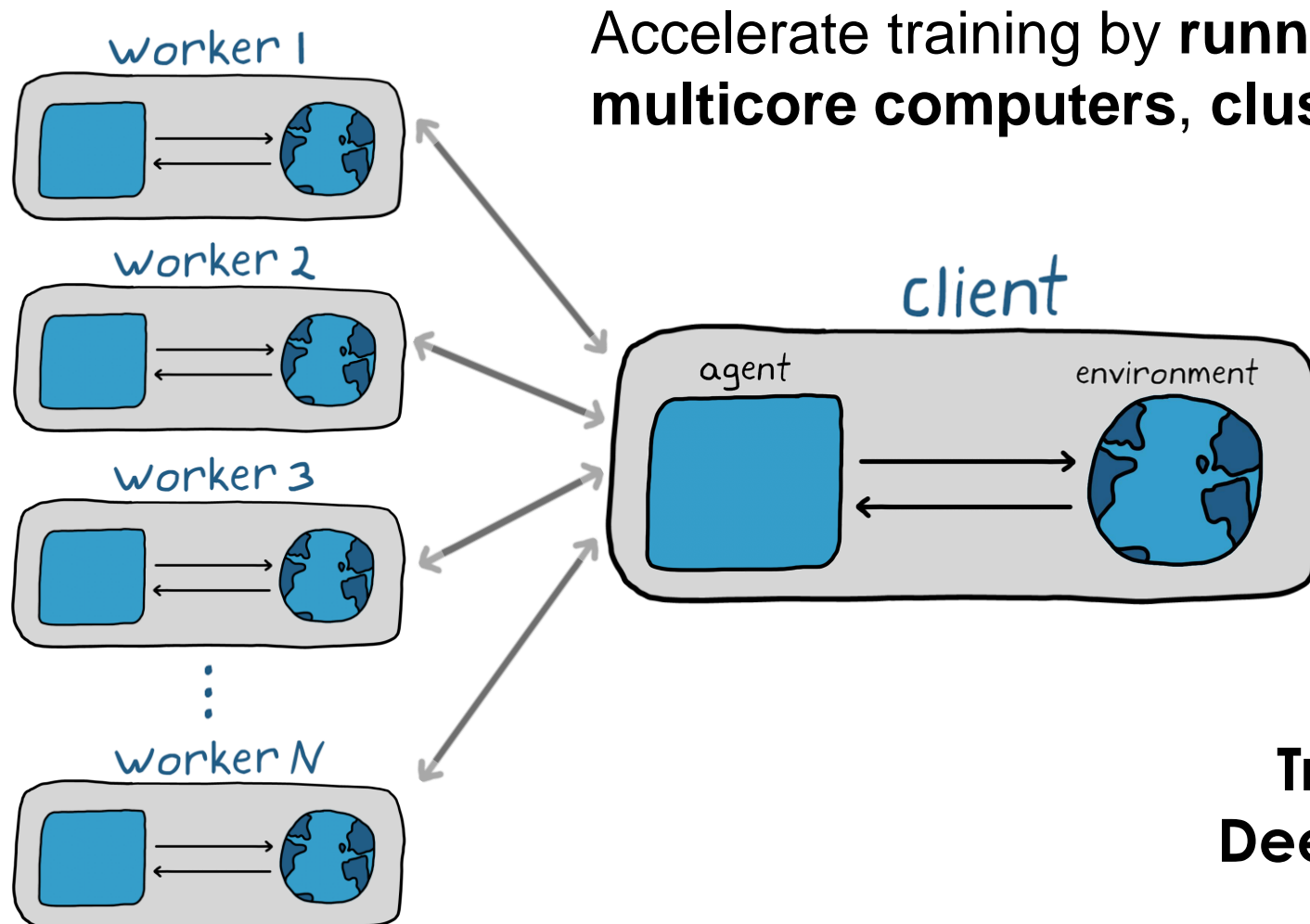
training



deploy



# Training Our Deep Reinforcement Learning Agent



**Train with the GPU** when using **Deep Neural Networks** for Actor or Critic representations

# Training the Agent

REINFORCEMENT LEARNING TRAIN SIMULATE DDPG AGENT

agentDDPG Import View Actor Model View Critic Model Train Simulate Export

NAME IMPORT REPRESENTATION NEXT STEPS

Agents: agentDDPG

Observation Specification

Observation Name	Domain	Dimension	Data Type
observations	continuous	[29 1]	double

Action Specification

Action Name	Domain	Dimension	Data Type
foot_torque	continuous	[6 1]	double

Hyperparameters

Agent Options

Sample time: 0.025  
 Discount factor: 0.99  
 Execution environment:  CPU  GPU  
 Batch size: 128  
 Experience buffer length: 1e+05

Actor Options

Learn rate: 0.0001  
 Gradient threshold: 1

Critic Options

Learn rate: 0.001  
 Gradient threshold: 1

More Options

Optimizer: adam  
 Denominator offset: 1e-08  
 Gradient decay: 0.9  
 Squared gradient decay: 0.999  
 Gradient threshold method: l2norm  
 L2 regularization: 0.0001

Exploration

Ornstein Uhlenbeck Noise Options

Standard deviation: 0.3  
 Mean: 0

Plot Options

X-axis limit: 1000

Ornstein Uhlenbeck Noise

Value

Steps

14 Agent opened: "agentDDPG"

# Testing the Agent

REINFORCEMENT LEARNING TRAIN SIMULATE **DDPG AGENT**

Environment: env Number of Episodes: 10  
 Agent: agentDDPG Max Episode Length: 400  
 Stop on Error Use Parallel Simulate

SYSTEM SIMULATION OPTIONS

Agents: agentDDPG x agentDDPG\_Trained

Environments: env

Results: experience1 trainStats1

Preview: agentDDPG\_Trained =  
 Type DDPG  
 Observation Dimension observations : [29 1]  
 Action Dimension foot\_torque : [6 1]  
 Actor Learn Rate 0.0001  
 Critic Learn Rate 0.001

**Overview**  
 The DDPG algorithm is an actor-critic reinforcement learning method which computes an optimal policy that maximizes the long-term reward.  
[Learn more](#)

**Observation Specification**

Observation Name	Domain	Dimension	Data Type
observations	continuous	[29 1]	double

**Action Specification**

Action Name	Domain	Dimension	Data Type
foot_torque	continuous	[6 1]	double

**Hyperparameters**

**Agent Options**  
 Sample time: 0.025  
 Discount factor: 0.99  
 Execution environment:  CPU  GPU  
 Batch size: 128  
 Experience buffer length: 1e+05

**Actor Options**  
 Learn rate: 0.0001  
 Gradient threshold: 1

**Critic Options**  
 Learn rate: 0.001  
 Gradient threshold: 1

**More Options**  
 Reset buffer  Save buffer  
 Target smooth factor: 0.001  
 Target update frequency: 1

**More Options**  
 Optimizer: adam  
 Denominator offset: 1e-08  
 Gradient decay: 0.9  
 Squared gradient decay: 0.999  
 Gradient threshold method: l2norm  
 L2 regularization: 0.0001

**Exploration**

**Ornstein Uhlenbeck Noise Options**  
 Standard deviation: 0.3  
 Mean: 0

**Ornstein Uhlenbeck Noise**

# Reinforcement Learning Workflow



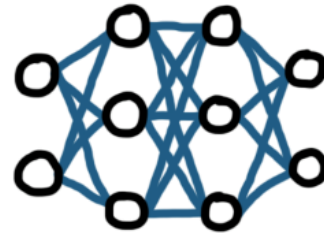
environment



reward



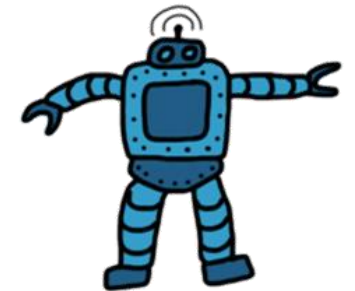
policy



training



deploy

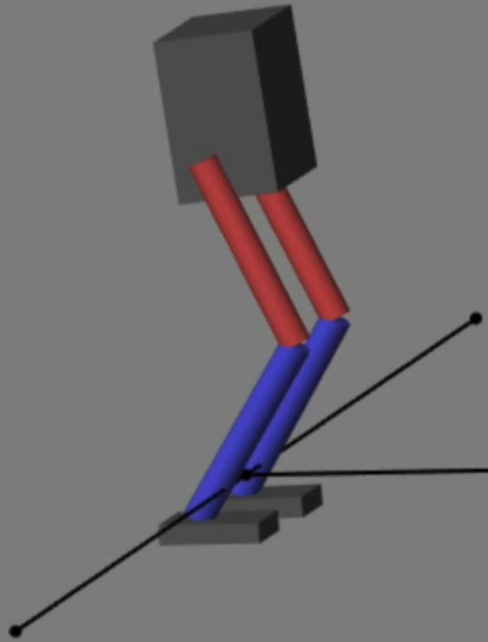


reward shaping

# Reward Function Design Matters



$$r_t = v_x$$



I want my robot to walk forward. Let's set the reward to be the robot's forward velocity.

# Reward Shaping to Improve Learning



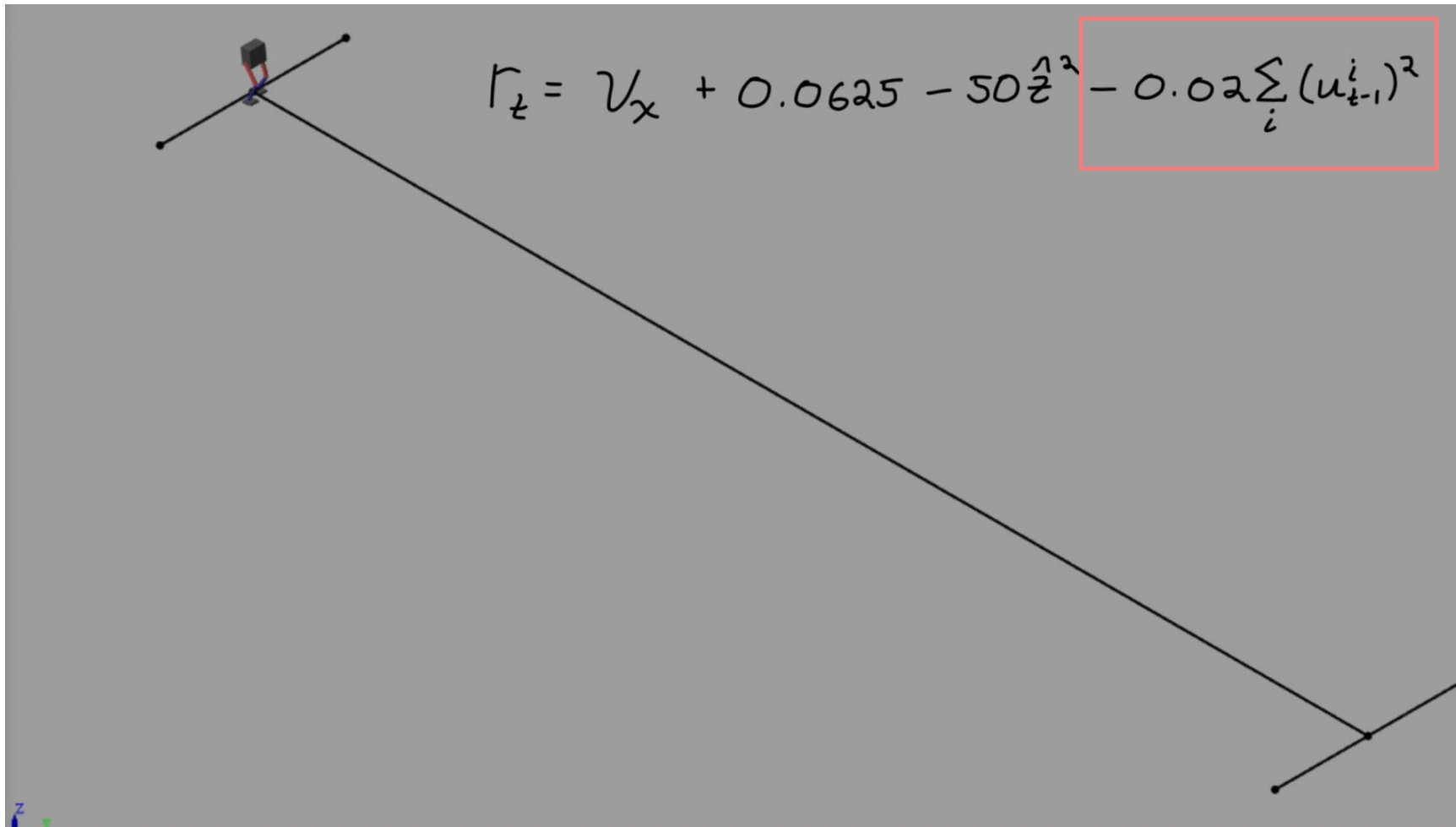
$$r_t = v_x + 0.0625 - 50z^2$$



Let's add a reward term for each time step it remains upright and a penalty for not maintaining a torso height

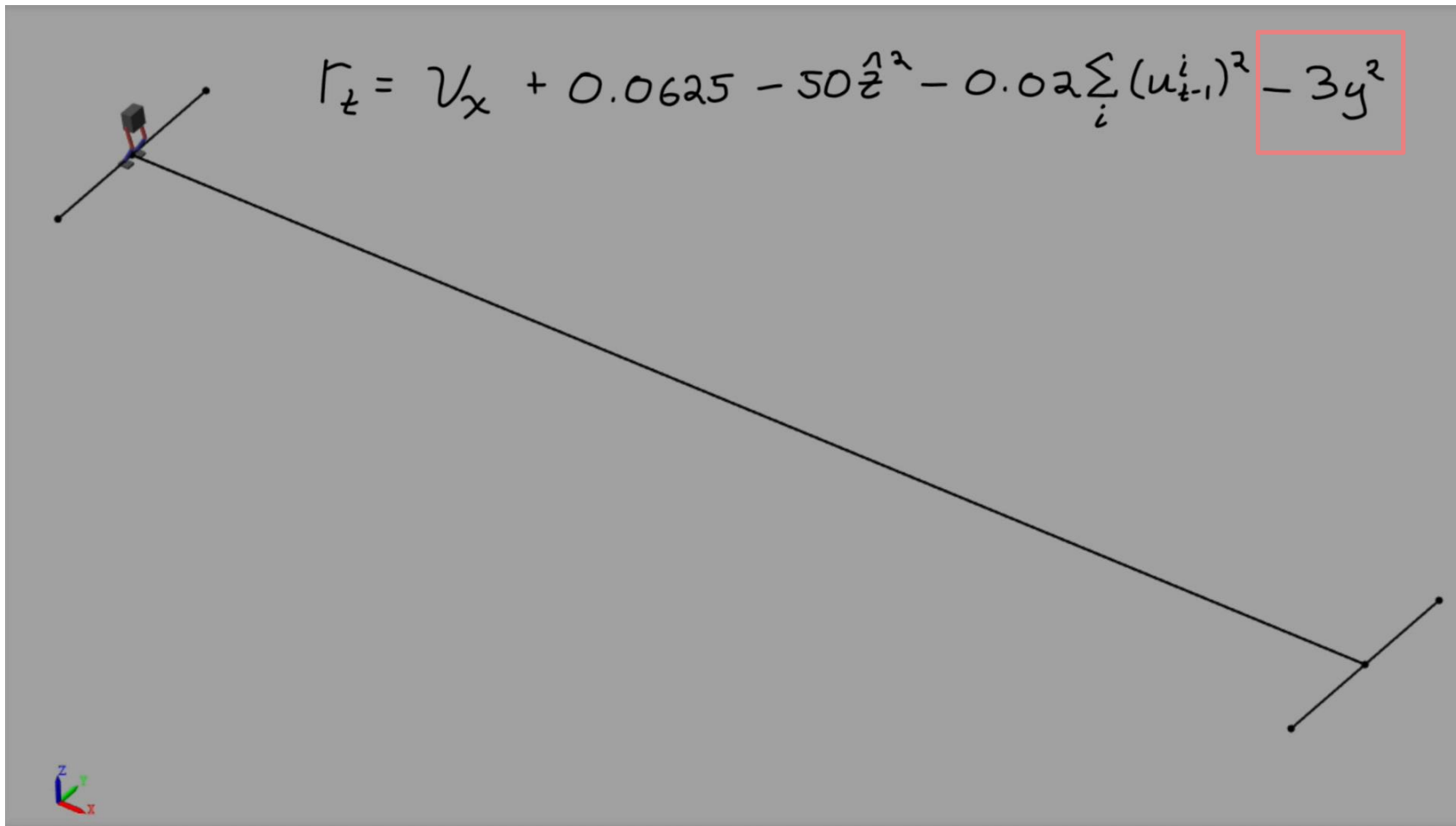


# Reward Shaping to Improve Learning



Let's add a penalty for the energy used for actuation

# Reward Shaping to Improve Learning



Let's add a penalty for deviating from the line

# Reinforcement Learning Workflow

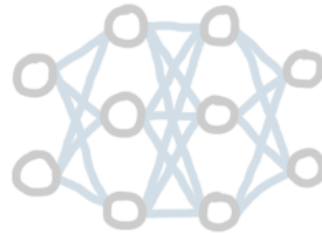
environment



reward



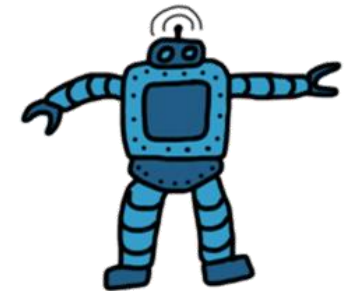
policy



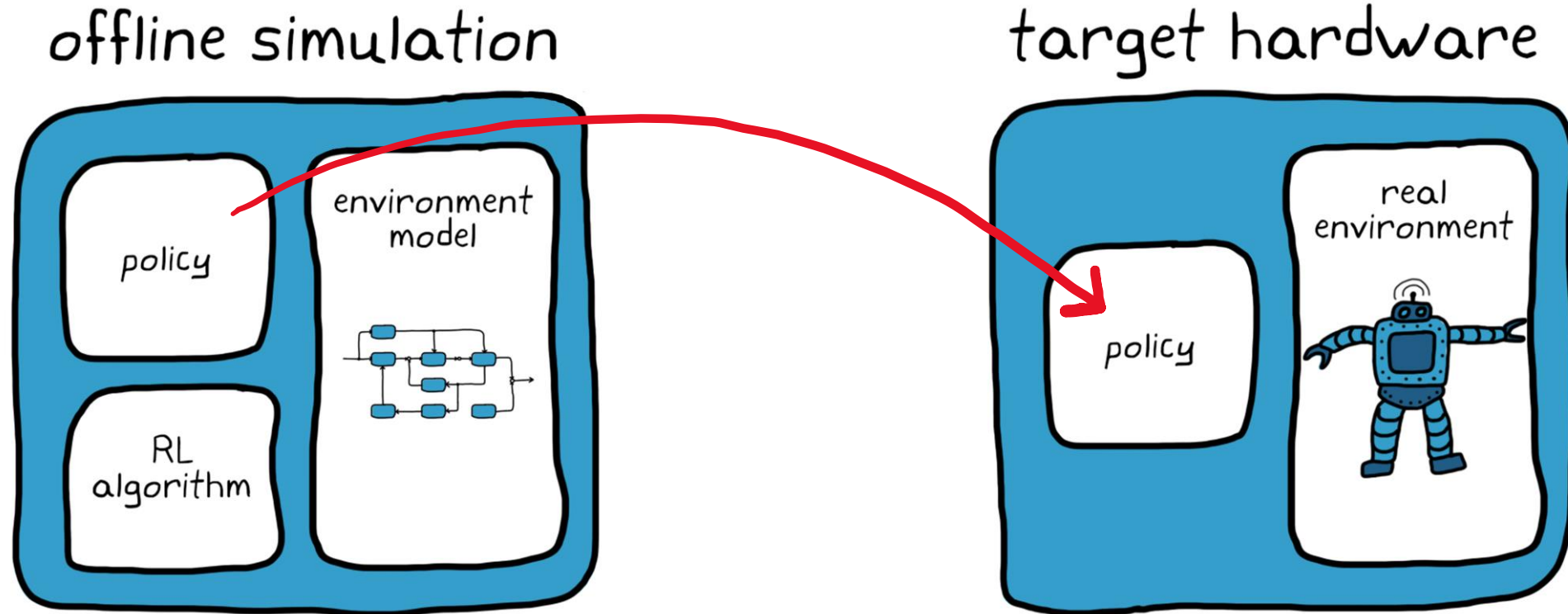
training



deploy

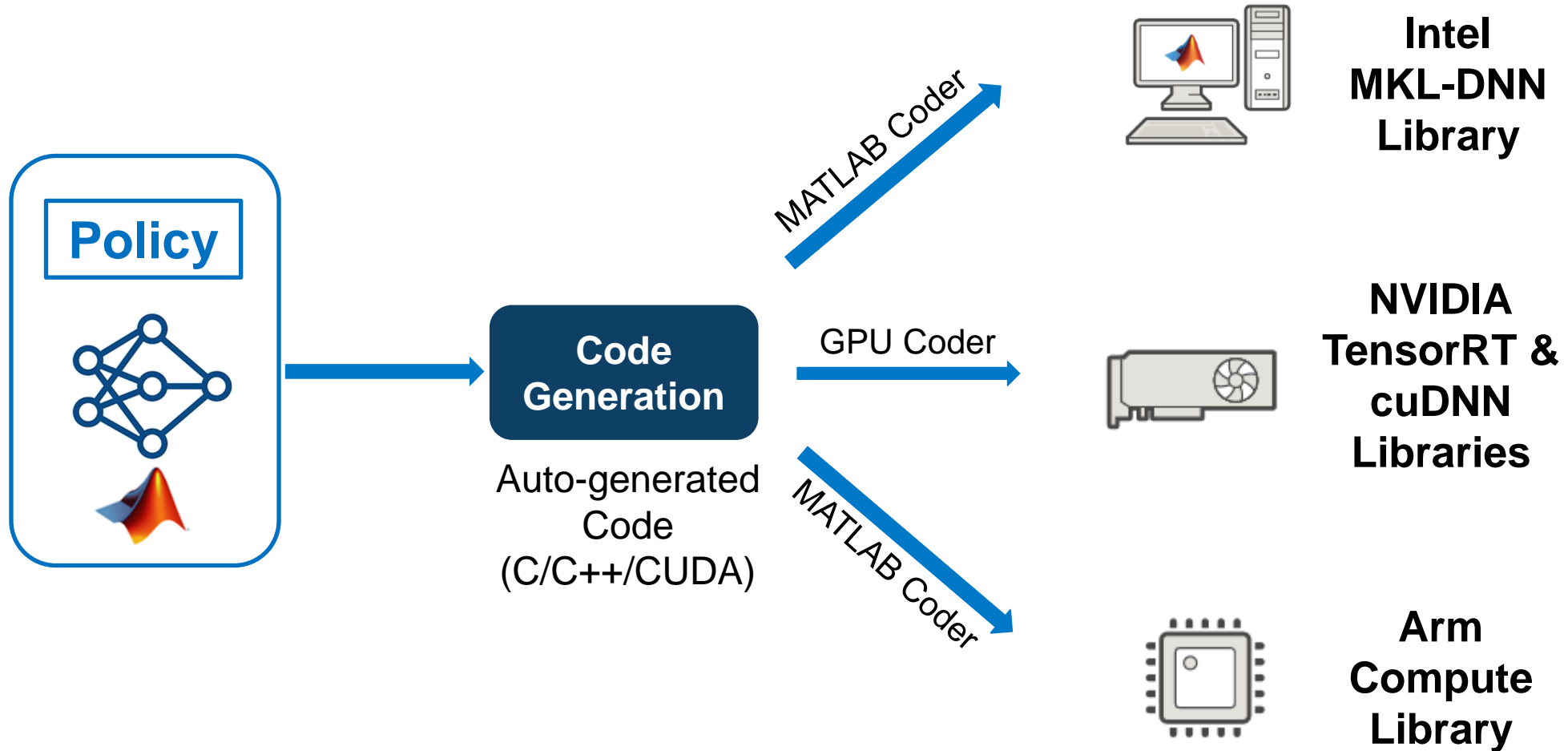


# Deploy policy to the target hardware



Automatically generate **C/C++** or **CUDA** code  
to run the **policy** on an **embedded system**

# Policy Deployment to Hardware Platforms



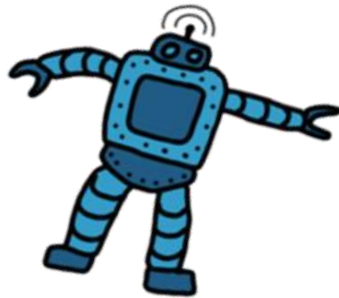
Intel® is a trademark of Intel Corporation

NVIDIA® and TensorRT® are registered trademarks of NVIDIA Corporation

Arm® is a registered trademark of Arm Limited (or its subsidiaries)

## Key takeaways

- **Reinforcement Learning** can solve complicated control and decision problems
- **Reward Shaping** can improve learning outcomes
- **MATLAB and Simulink** provide a **complete workflow for Deep Reinforcement Learning**





감사합니다