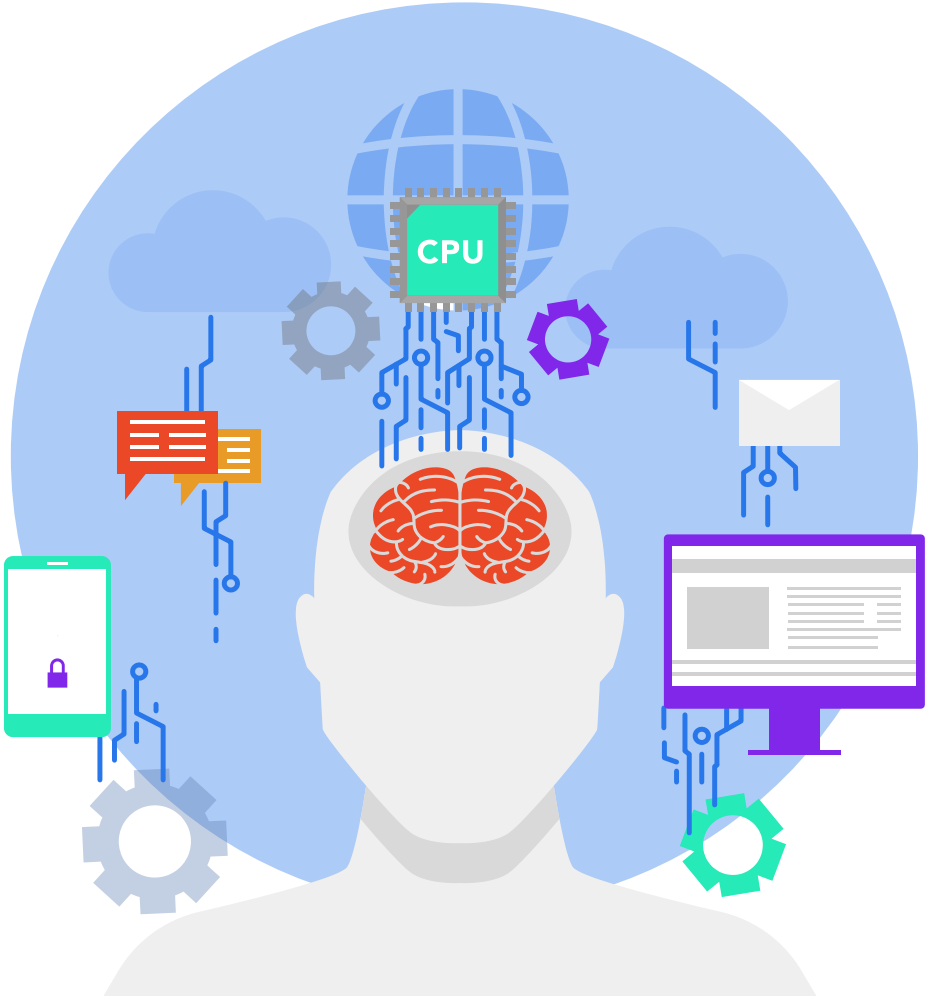


TensorFlow Agents

Intelligent Systems II

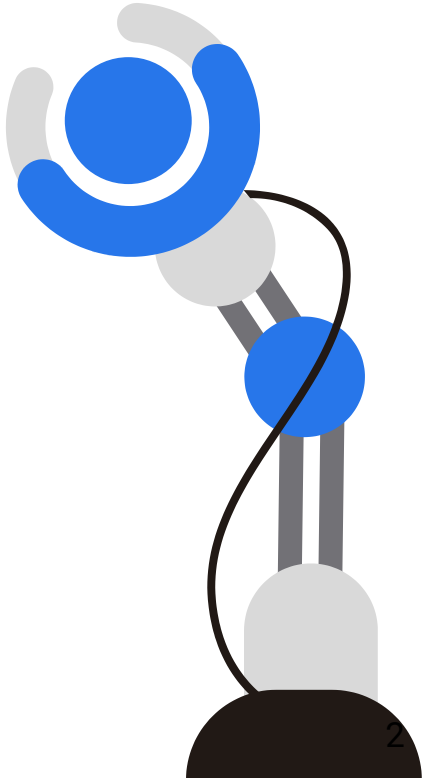
Luís Seabra Lopes

2023

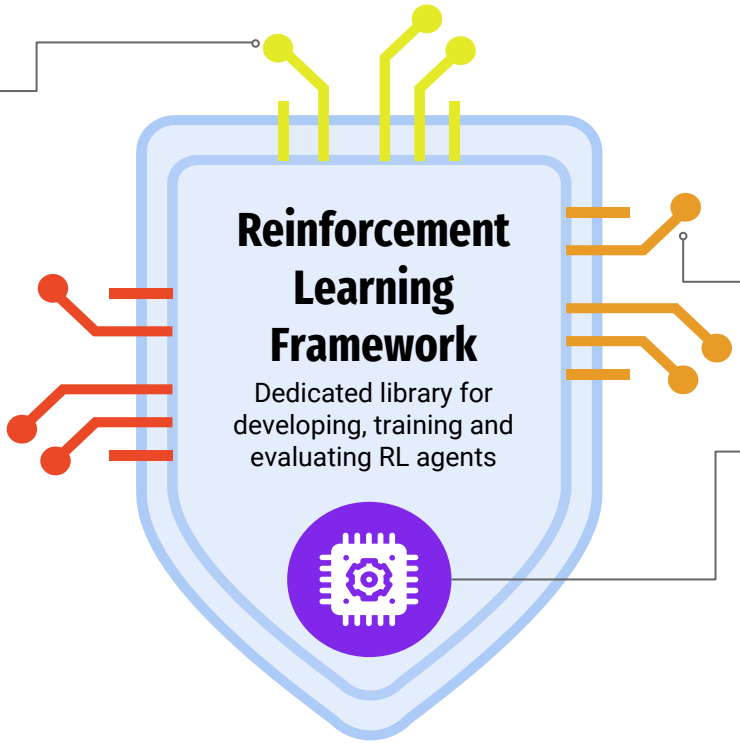
MSc in Software Engineering and MSc in Robotics and Intelligent Systems

Artur Romão, 98470
João Reis, 98474
Paulo Pereira, 98430

Contents



What is TF-Agents?



01

Supported RL Algorithms

Deep Q-Networks, Proximal Policy Optimization, Soft Actor-Critic, etc

02

Environment Compatibility

Such as OpenAI Gym and DeepMind Lab

03

Customizable Components

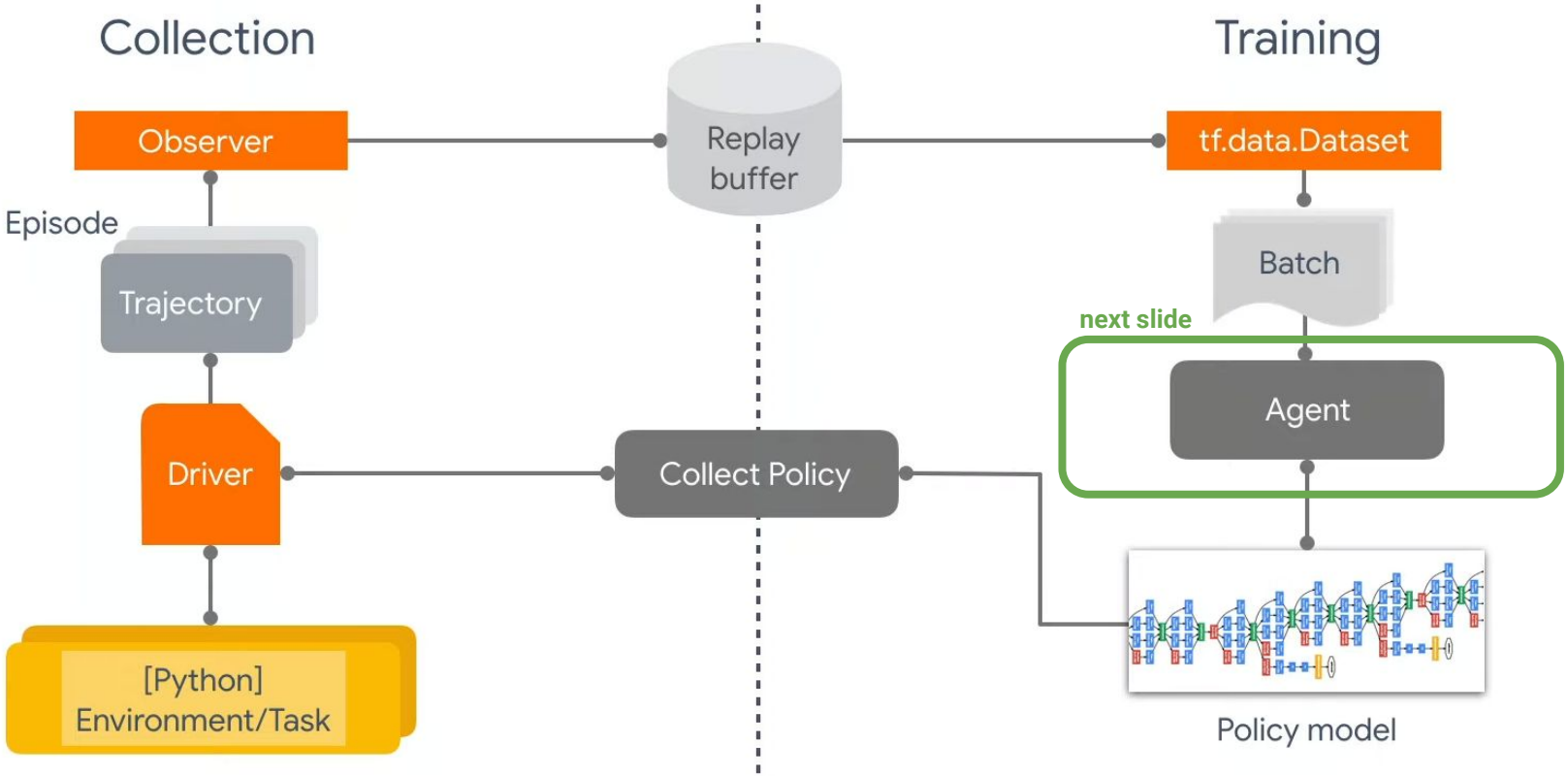
Ease to create custom algorithms, environments, and network architectures

04

Scalability and Distributed Training

Ability to scale RL experiments for faster learning and deployment

TF-Agents System Overview



Agent

01 Core Component

Responsible for learning and decision-making

02 Observation

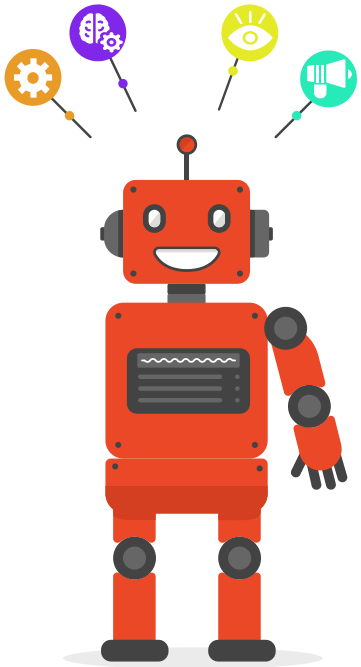
Observes the environment

03 Actions

Takes actions based on its internal policy

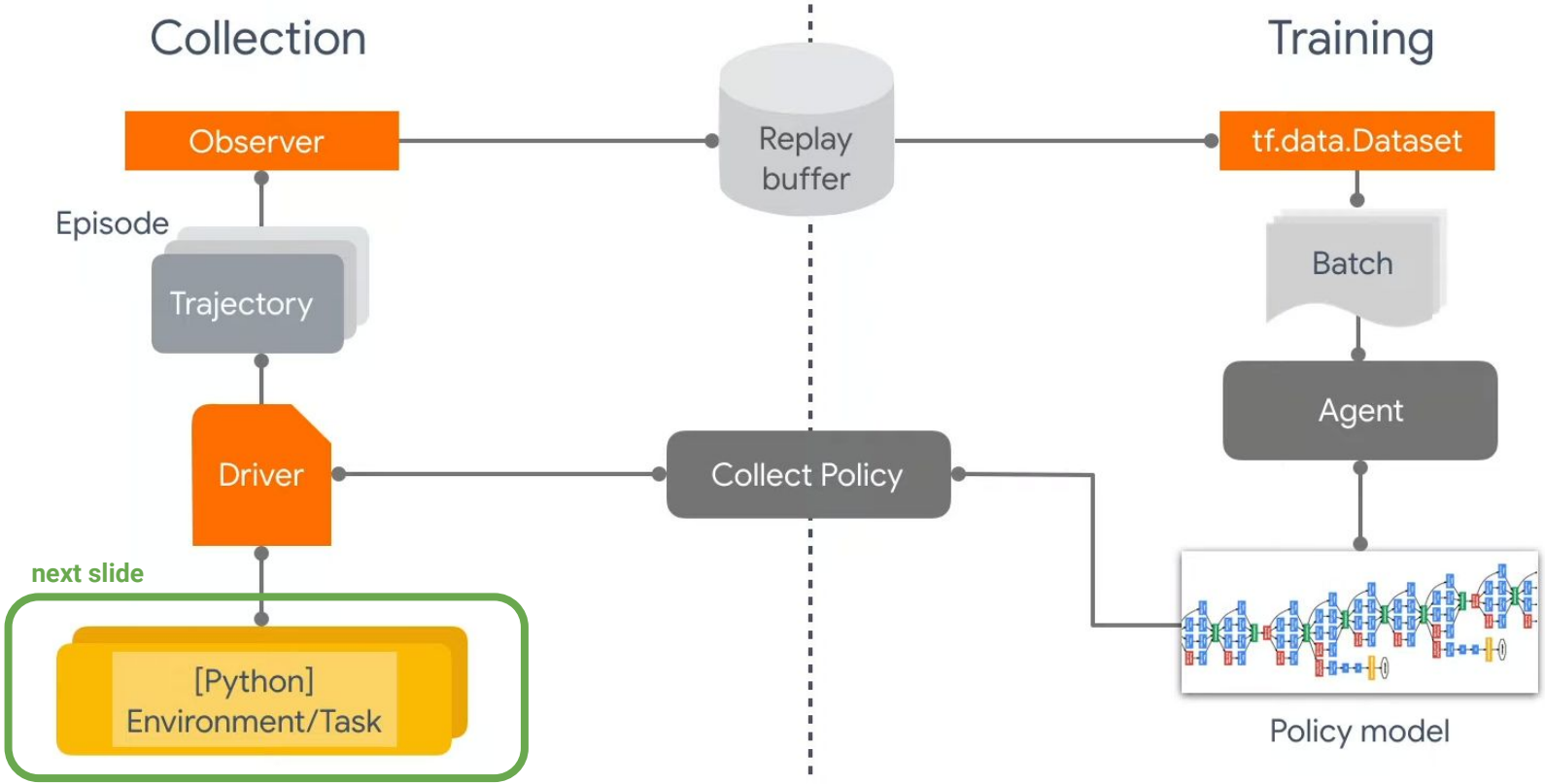
04 Learn

Learns from the rewards it receives



TF-Agents provides a collection of pre-implemented agents: DQN, SAC, and PPO, which can be used for different reinforcement learning algorithms.

TF-Agents System Overview



Environment

1

Agent **interacts** with the environment

- environment returns a **reward** and a **new observation**
- **goal:** maximize the reward

2

Usually, two environments are created

- one for **training** and one for **evaluation**

3

Implemented using **Python** or **TensorFlow**

- **Python**: easier to implement, understand and debug
- **TensorFlow**: more efficient and allows natural parallelization

4

Or implemented by using **standard environments**

- OpenAI gym
- Atari
- DeepMind-Control

5

Provide wrappers to **convert** Python environments into TensorFlow environments

```
# Load the CartPole environment from the
OpenAI gym
env = suite_gym.load("CartPole-v0")

# Test and validate the environment
utils.validate_py_environment (env,
episodes=5)
```

Python vs TensorFlow Environments

Python Environments (PyEnvironment)

Easier to implement, understand and debug

- Use numpy arrays

Both are **abstract base classes** for RL environments

Both have a method **step()** that returns a TimeStep(step_type, reward, discount, observation)

TensorFlow Environments (TFEnvironment)

More efficient and allow natural parallelization

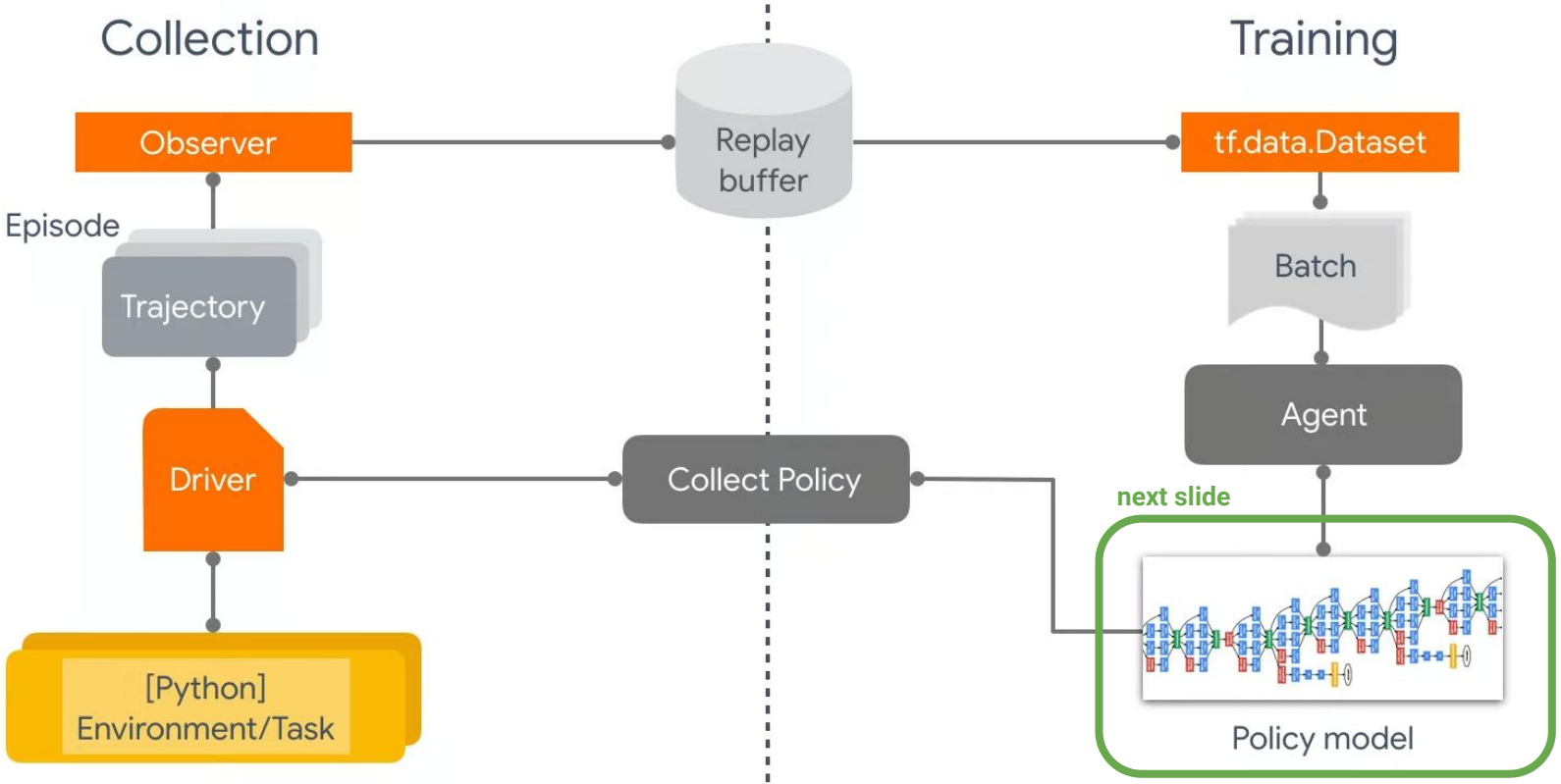
- Generate and use tensor objects

most common workflow: implement an Python environment and use a wrapper to automatically convert it into TensorFlow environment

How about creating your own Python Environment? make sure the observations and time_steps generated follow the correct shapes and types as defined in your specs

```
class CardGameEnv(py_environment.PyEnvironment ):
```

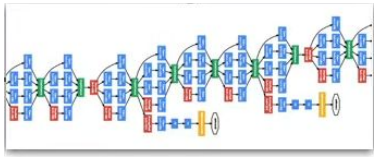
TF-Agents System Overview



Policies

01 What is a policy?

Defines how an agent selects actions based on its observations or states. It encapsulates the **decision-making process** of an agent.



Policy model

02 NN

Most policies have a **neural network** to compute actions.

03 Wrappers

Converting between Python and TensorFlow policies. Can be used to wrap and modify a given policy, e.g. add noise.

→ **Tensorflow Policies**

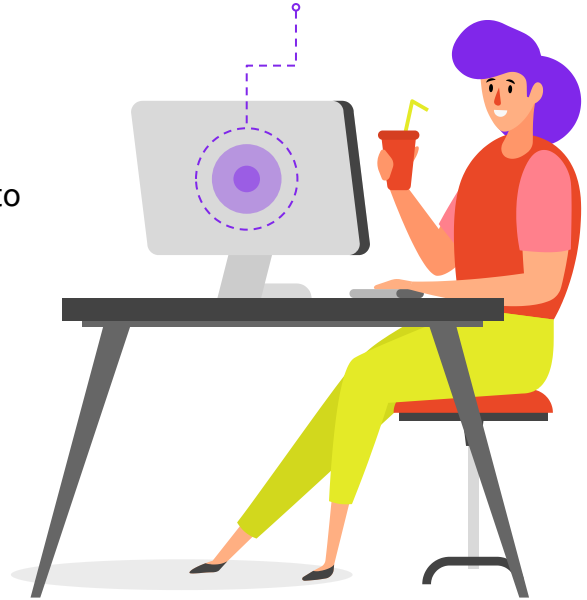
- Random TF Policy
- Actor Policy
- Q Policy

04 Multiple policies

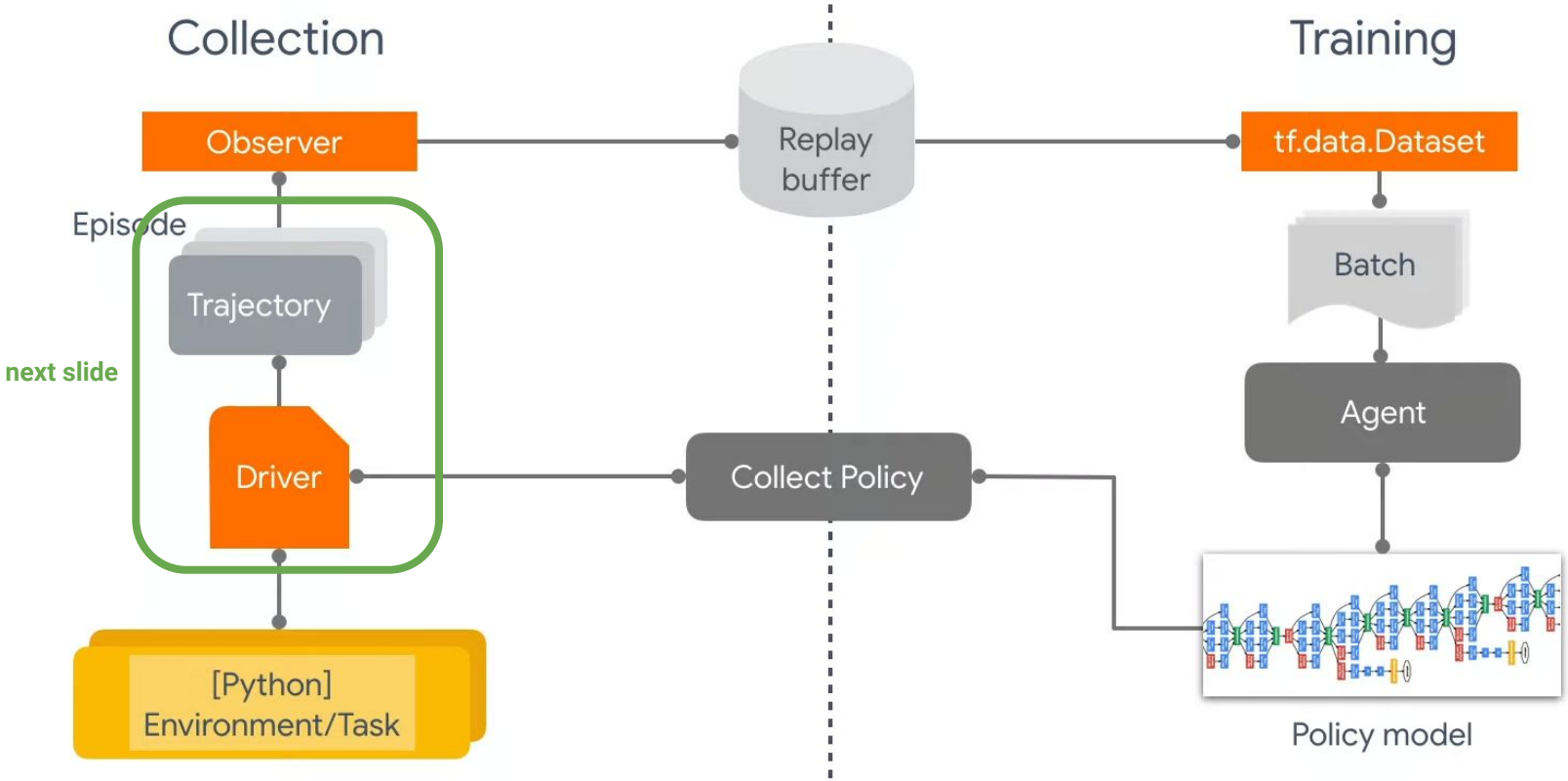
Agents can contain one or more policies for different purposes.

→ **Example**:

- a **main** policy that is being trained for **deployment**
- a noisy policy for **data collection**



TF-Agents System Overview



Driver

Component

Interacts with the environment using a policy to collect experience data

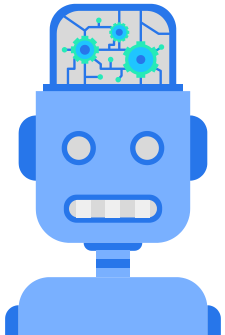
Abstraction

A driver encapsulates the RL loop in its *run()* method

Nature

Can be either Python-based or TensorFlow-based

```
Trajectory(
 {
  "action": array(0),
  "discount": array(1., dtype=float32),
  "next_step_type": array(1, dtype=int32),
  "observation": array([-0.02338193,
-0.01357831,  0.04843839,  0.02565479],
dtype=float32),
  "policy_info": (),
  "reward": array(1., dtype=float32),
  "step_type": array(0, dtype=int32)
 }
)
```



Storage

Data is stored in a tuple called Trajectory and is broadcast to a set of observers

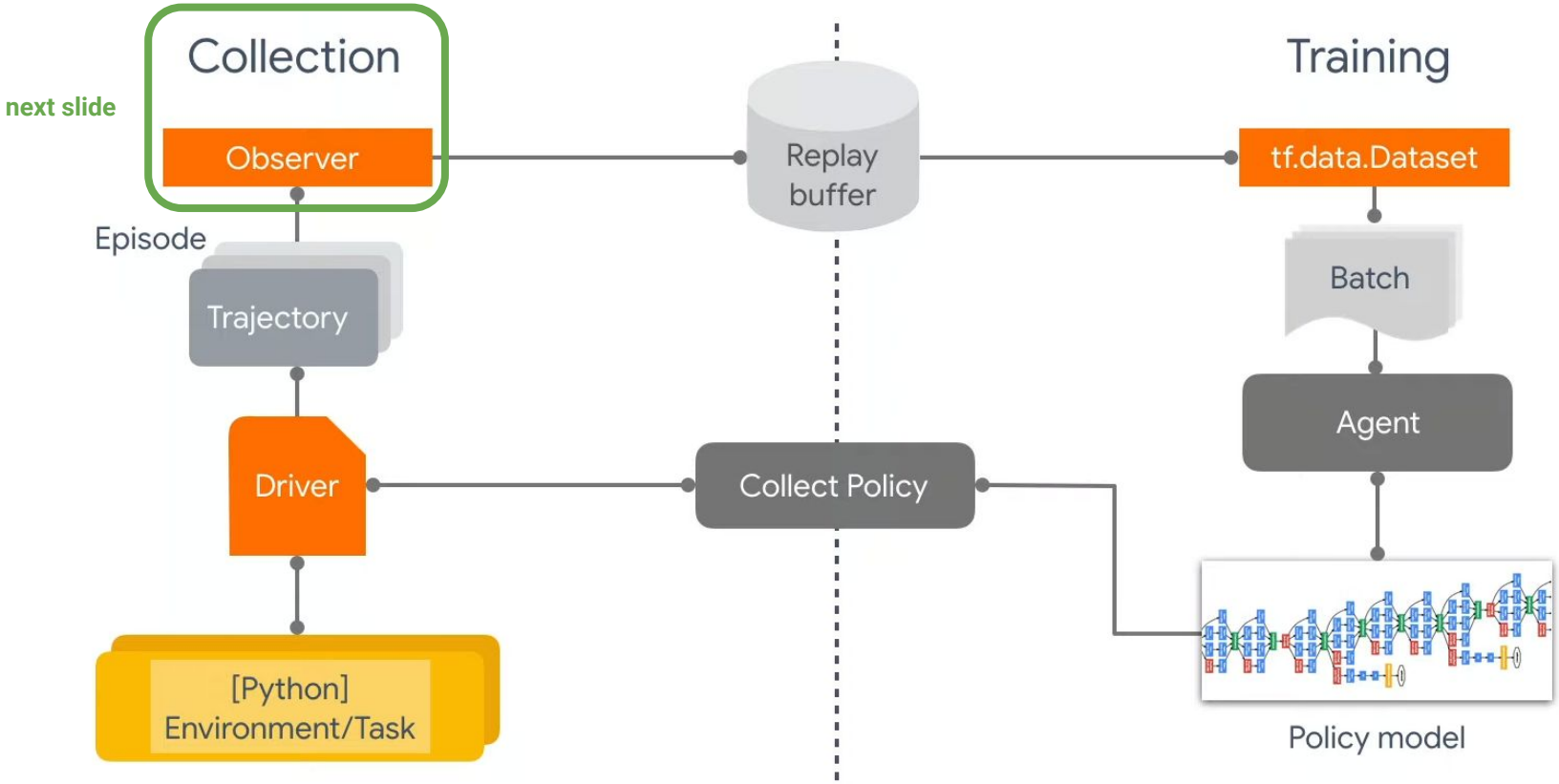
Data collected

Observations, actions, rewards, step type, etc

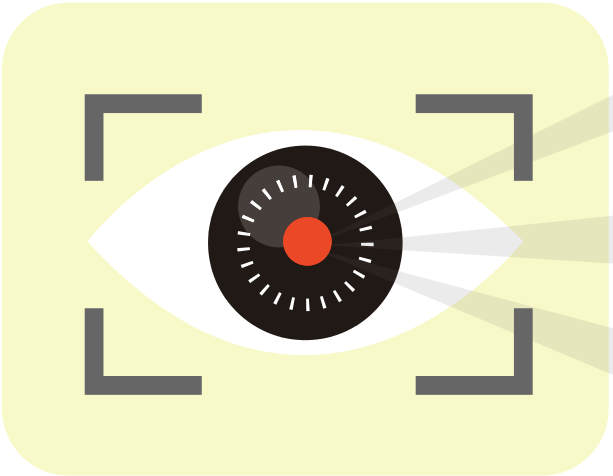
Efficiency

TF-based drivers are more efficient

TF-Agents System Overview



Observer



Trajectory Processing

Process the trajectories generated from an interaction agent-environment

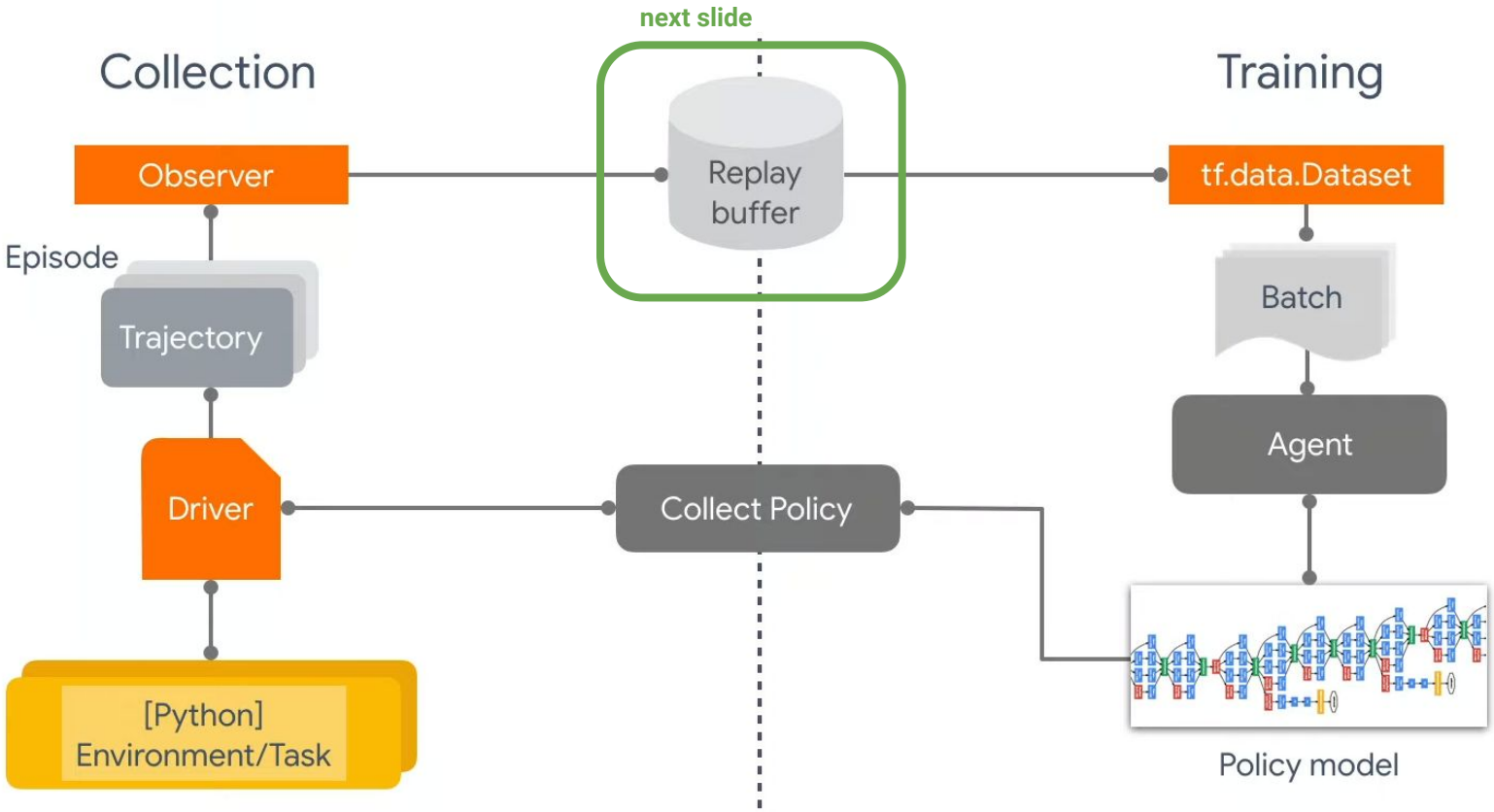
Function or Callable Object

High flexibility, observers are any function that takes a Trajectory as argument

Incorporation in the Driver

Driver's *run()* function takes a list of observers as argument

TF-Agents System Overview



Replay Buffers

01 Store data collection

- Store trajectories of experience observed by an agent while interacting with an environment. Trajectories consist of an observation, action, reward, next observation, and other optional elements, depending on the specific implementation.

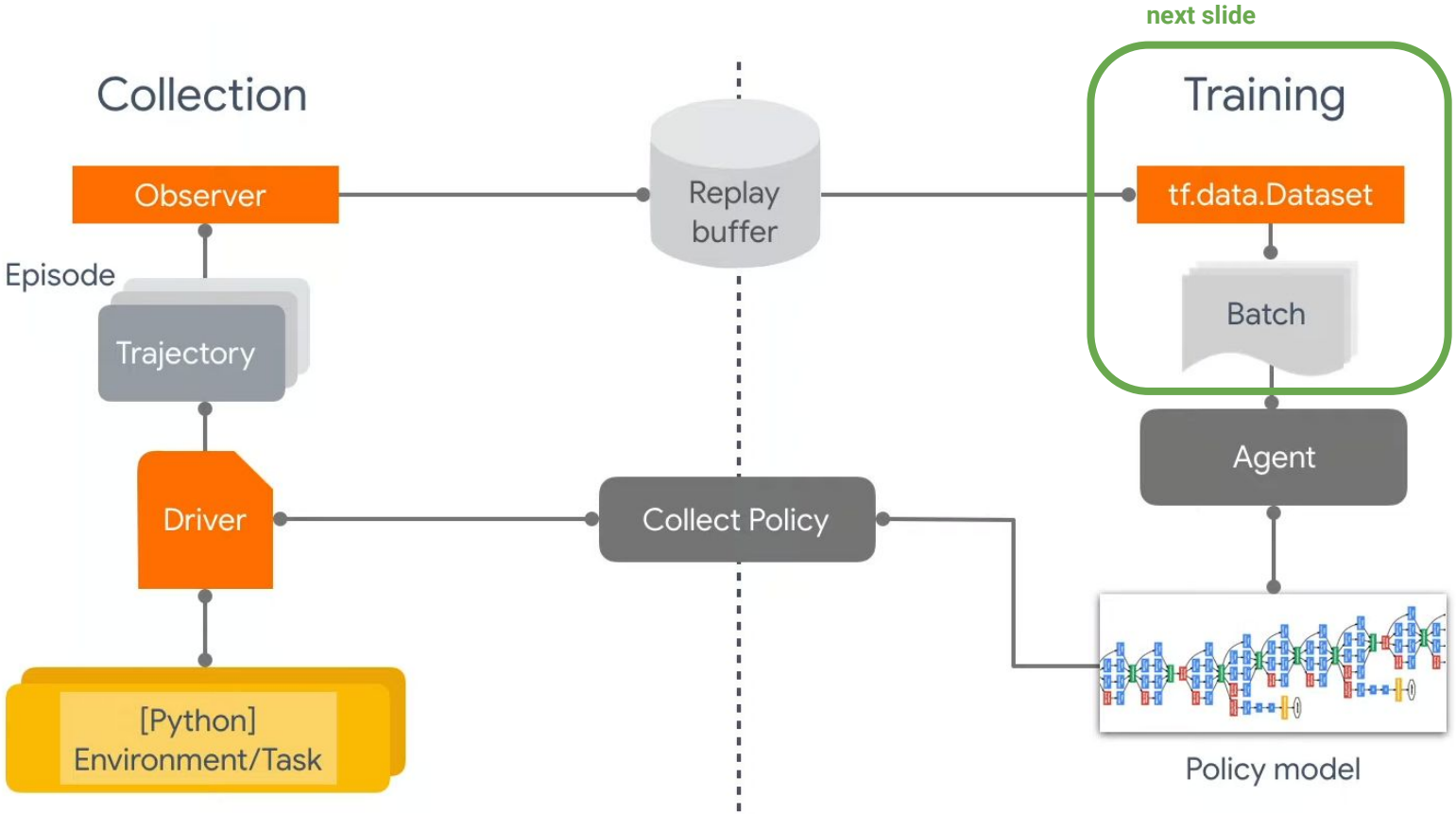
02 Used for replay the experience

- During training, replay buffers are **queried** for a subset of the trajectories (either a sequential subset or a sample) to "**replay" the agent's experience**

03 Replay Buffer API

- The *ReplayBuffer* class allow **create, write and read** from the buffer. *TFUniformReplayBuffer* is the most commonly used replay buffer in TF-Agents.

TF-Agents System Overview



Networks

Define the model that is trained by agents

Main networks

01

QNetwork

Using Qlearning, maps an observation to value estimates for each possible action

02

CriticNetworks

Estimate how good the state the agent is currently in is

03

ActorNetworks

04

ActorDistributionNetwork

Helper networks

05

EncodingNetwork

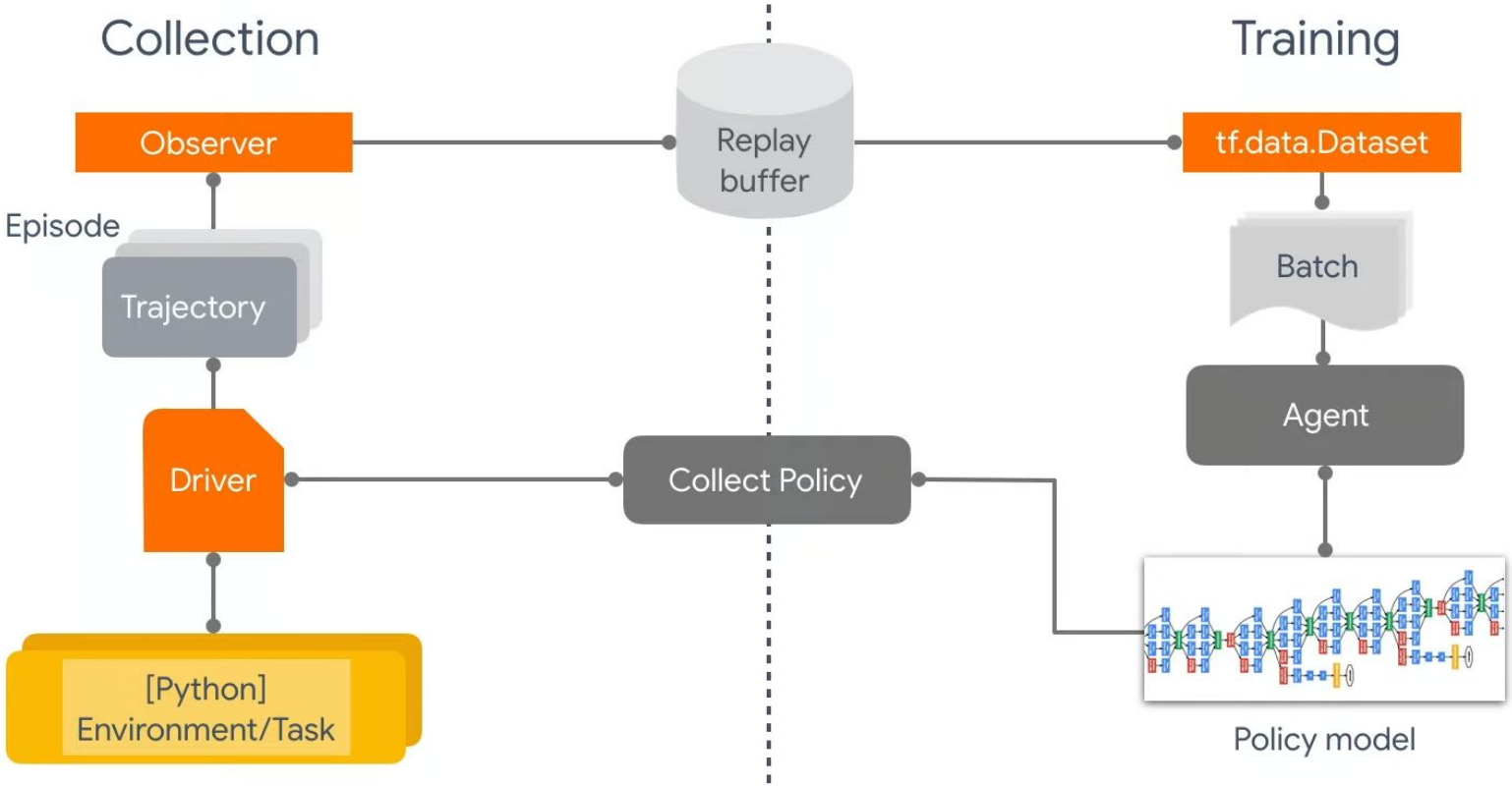
Allows users to easily define a mapping of pre-processing layers to apply to a network's input

06

DynamicUnrollLayer

Automatically resets the network's state on episode boundaries as it is applied over a time sequence.

TF-Agents System Resume



Demonstration