PyTorch-Native RL & Agentic Development

At scale

Davide Testuggine

Software Engineer

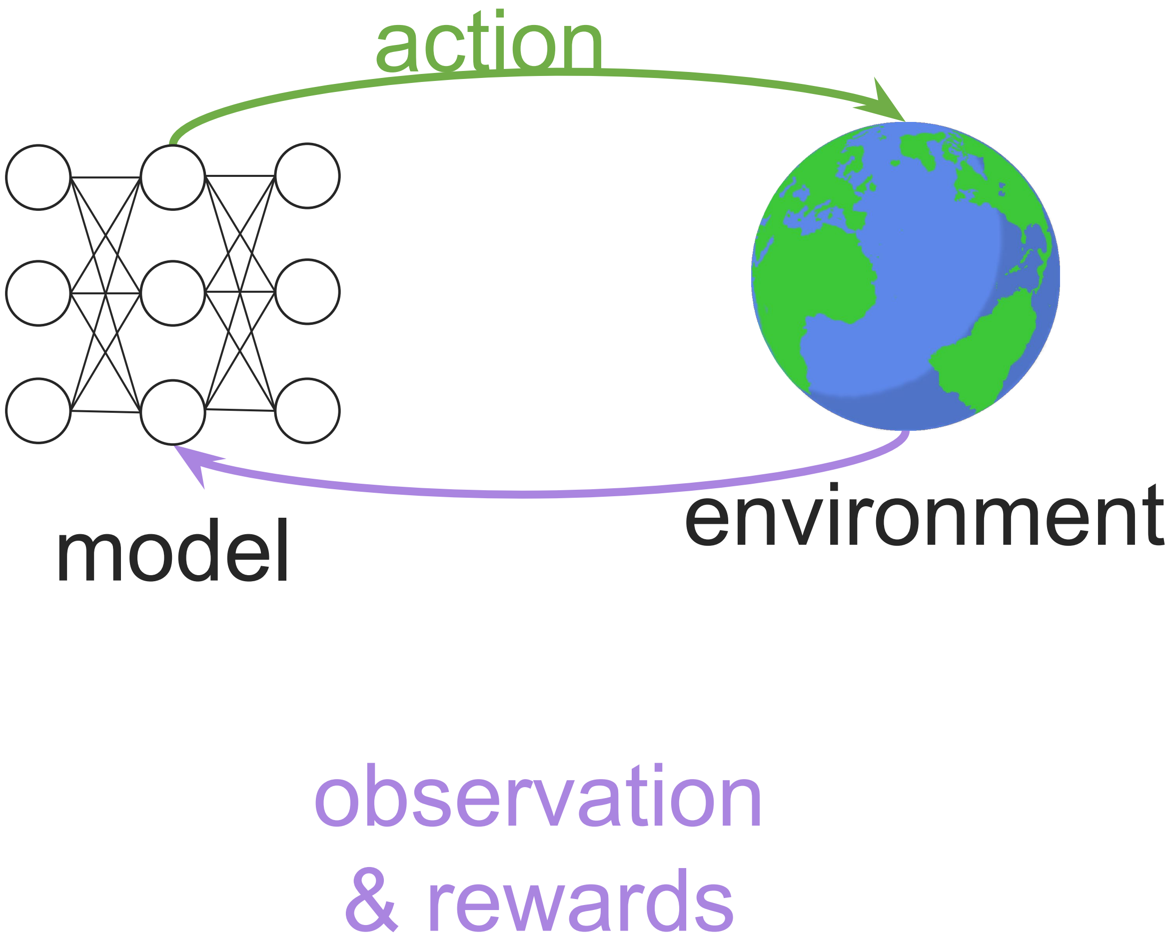
Meta

AGENDA

01 – Our ecosystem: Forge + OpenEnv

02 – Quick TLDR on Forge

(Next talks will go deep on OpenEnv)



trainer

OUR STRATEGY

Building Agents requires a new stack

01 – INFRASTRUCTURE

Maximize RL throughput while still being easy to build/debug

Get infra “out of the way” and focus on algorithms

TORCHFORGE

02 – DATA

Get the model exposed to as many different tasks as possible

Reuse everyone else’s work

OPENENV

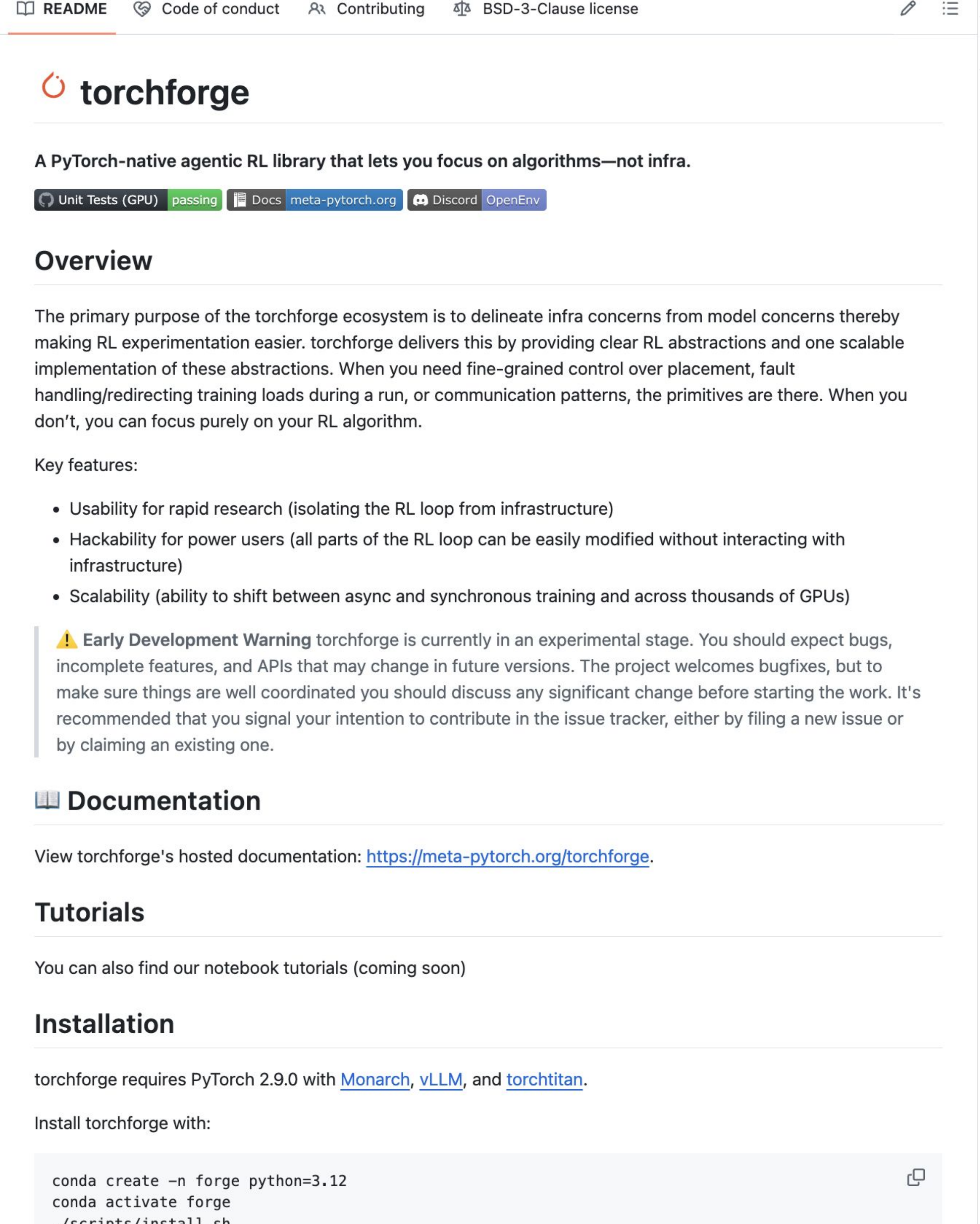
TORCHFORGE

Focus on algorithms — not infra.

- Actor-based programming model
- Separate control and data planes

Coming soon:

- Full agentic loops
- Multi-agent orchestration



README Code of conduct Contributing BSD-3-Clause license

torchforge

A PyTorch-native agentic RL library that lets you focus on algorithms—not infra.

Unit Tests (GPU) passing | Docs meta-pytorch.org | Discord OpenEnv

Overview

The primary purpose of the torchforge ecosystem is to delineate infra concerns from model concerns thereby making RL experimentation easier. torchforge delivers this by providing clear RL abstractions and one scalable implementation of these abstractions. When you need fine-grained control over placement, fault handling/redirecting training loads during a run, or communication patterns, the primitives are there. When you don't, you can focus purely on your RL algorithm.

Key features:

- Usability for rapid research (isolating the RL loop from infrastructure)
- Hackability for power users (all parts of the RL loop can be easily modified without interacting with infrastructure)
- Scalability (ability to shift between async and synchronous training and across thousands of GPUs)

> ⚠️ **Early Development Warning** torchforge is currently in an experimental stage. You should expect bugs, incomplete features, and APIs that may change in future versions. The project welcomes bugfixes, but to make sure things are well coordinated you should discuss any significant change before starting the work. It's recommended that you signal your intention to contribute in the issue tracker, either by filing a new issue or by claiming an existing one.

📖 Documentation

View torchforge's hosted documentation: https://meta-pytorch.org/torchforge.

Tutorials

You can also find our notebook tutorials (coming soon)

Installation

torchforge requires PyTorch 2.9.0 with Monarch, vLLM, and torchtitan.

Install torchforge with:

```
conda create -n forge python=3.12
conda activate forge
./scripts/install.sh
```

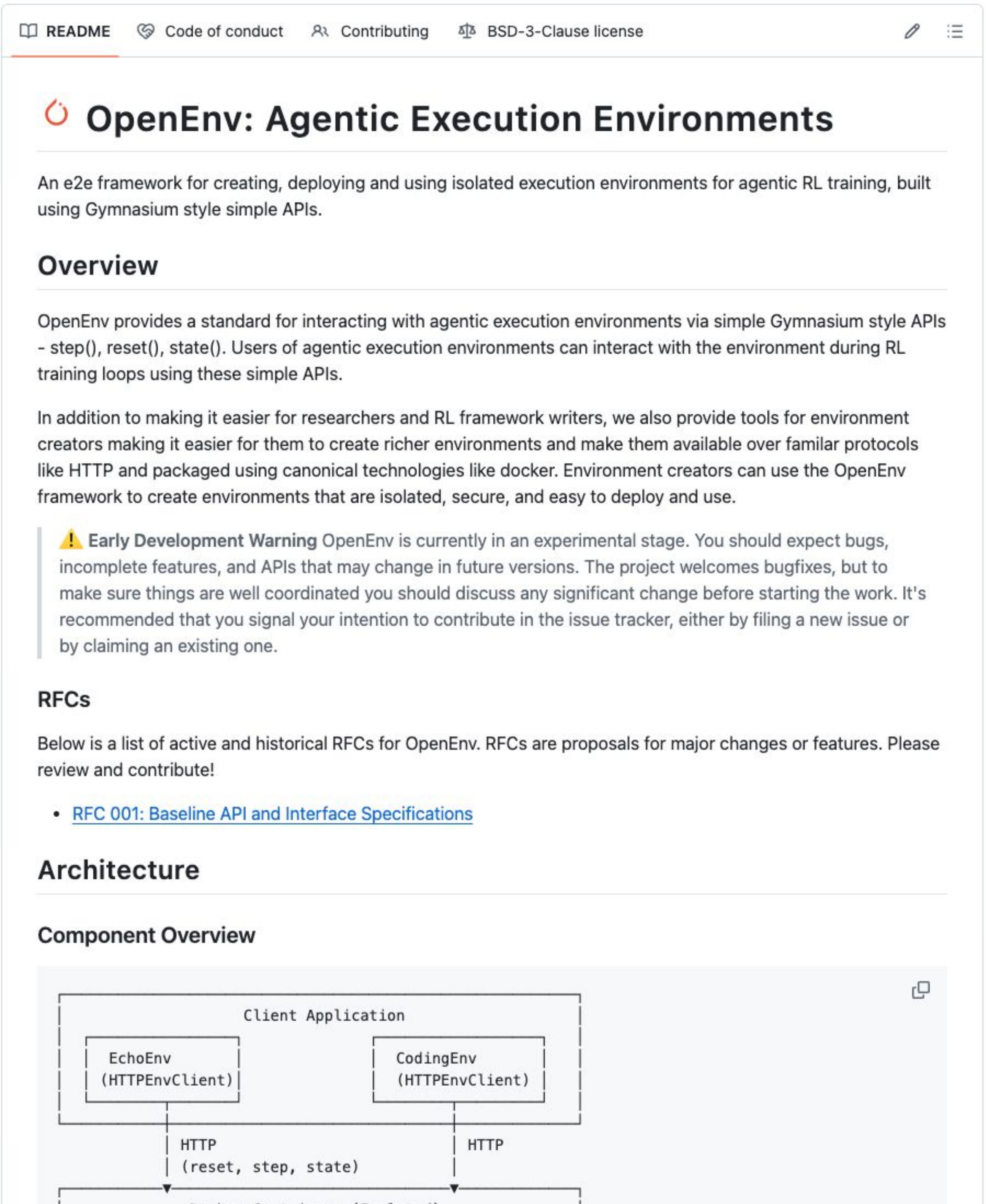
OPENENVS

Standardized spec to building environments.

- Gymnasium-like APIs
- Containers, Local & MCP tools as first-class citizens

Coming soon:

- Reward pipelines
- Evals



README | Code of conduct | Contributing | BSD-3-Clause license

OpenEnv: Agentic Execution Environments

An e2e framework for creating, deploying and using isolated execution environments for agentic RL training, built using Gymnasium style simple APIs.

Overview

OpenEnv provides a standard for interacting with agentic execution environments via simple Gymnasium style APIs - step(), reset(), state(). Users of agentic execution environments can interact with the environment during RL training loops using these simple APIs.

In addition to making it easier for researchers and RL framework writers, we also provide tools for environment creators making it easier for them to create richer environments and make them available over familar protocols like HTTP and packaged using canonical technologies like docker. Environment creators can use the OpenEnv framework to create environments that are isolated, secure, and easy to deploy and use.

> ⚠️ **Early Development Warning** OpenEnv is currently in an experimental stage. You should expect bugs, incomplete features, and APIs that may change in future versions. The project welcomes bugfixes, but to make sure things are well coordinated you should discuss any significant change before starting the work. It's recommended that you signal your intention to contribute in the issue tracker, either by filing a new issue or by claiming an existing one.

RFCs

Below is a list of active and historical RFCs for OpenEnv. RFCs are proposals for major changes or features. Please review and contribute!

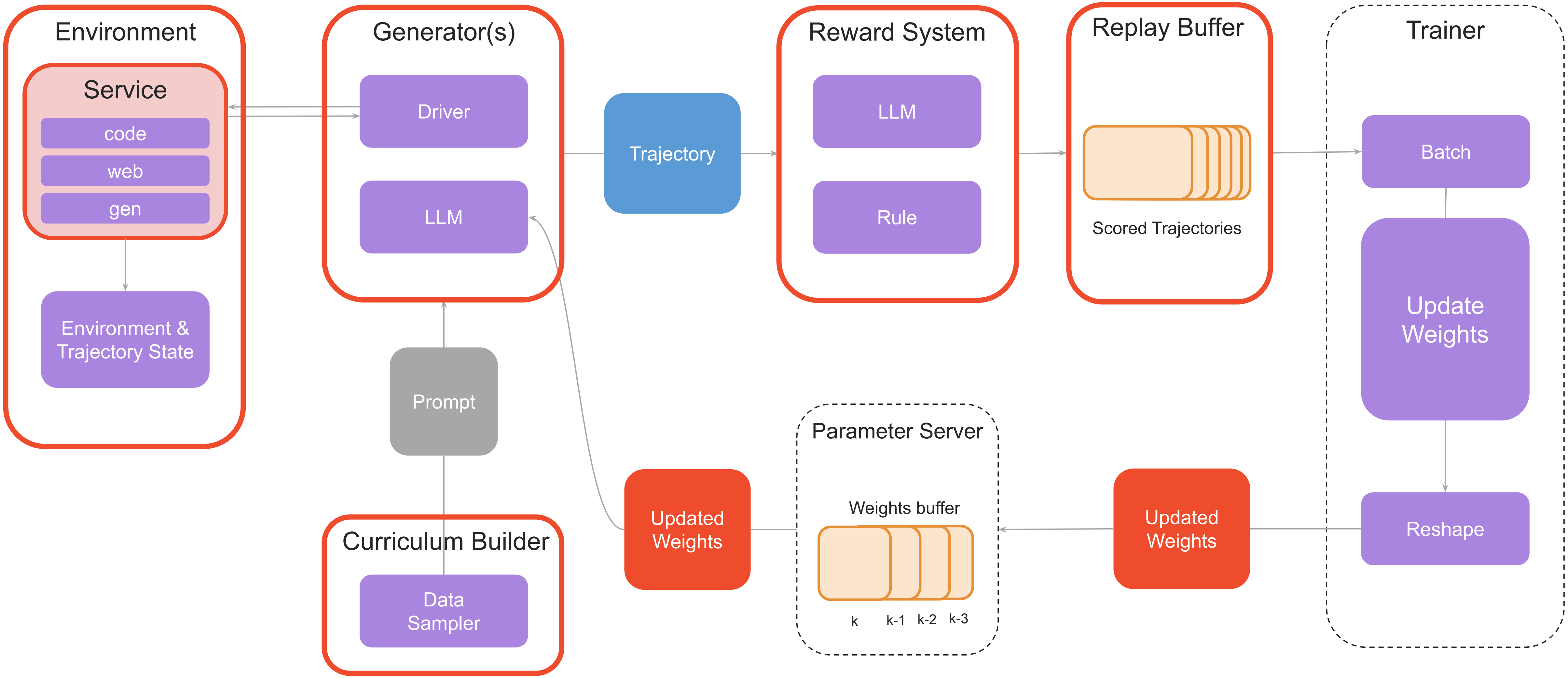
- RFC 001: Baseline API and Interface Specifications

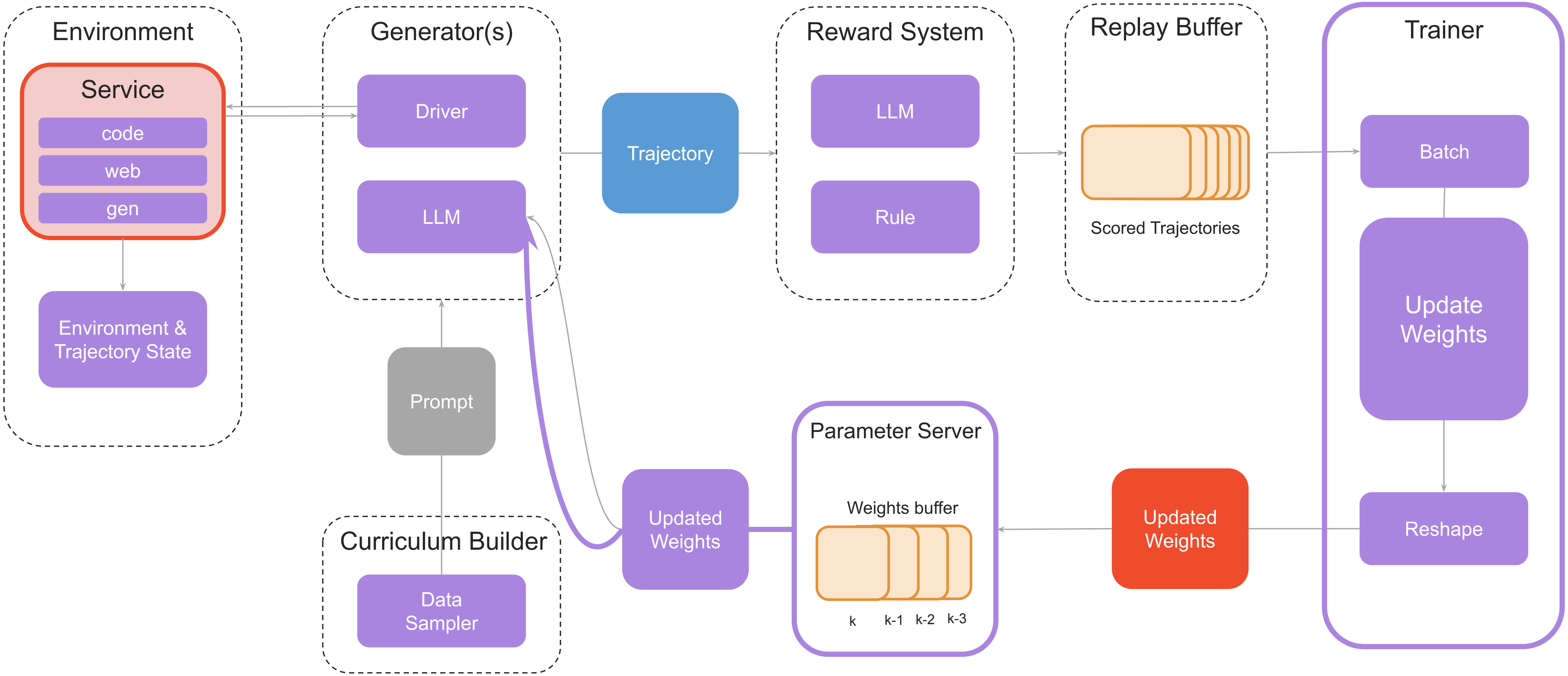
Architecture

Component Overview

```
┌─────────────────────────────────────────────────────────────┐
│                     Client Application                      │
│  ┌────────────────┐                  ┌────────────────┐     │
│  │  EchoEnv       │                  │  CodingEnv     │     │
│  │ (HTTPEnvClient)│                  │ (HTTPEnvClient)│     │
│  └────────┬───────┘                  └────────┬───────┘     │
└───────────┼───────────────────────────────────┼─────────────┘
            │ HTTP                              │ HTTP
            │ (reset, step, state)              │
            ▼                                   ▼
```

RL INFRA


Environment
Service
code
web
gen
Environment & Trajectory State
Generator(s)
Driver
LLM
Prompt
Curriculum Builder
Data Sampler
Trajectory
Reward System
LLM
Rule
Replay Buffer
Scored Trajectories
Trainer
Batch
Update Weights
Reshape
Updated Weights
Parameter Server
Weights buffer
k
k-1
k-2
k-3
Updated Weights


Environment
Service
code
web
gen
Environment & Trajectory State
Generator(s)
Driver
LLM
Prompt
Curriculum Builder
Data Sampler
Trajectory
Reward System
LLM
Rule
Replay Buffer
Scored Trajectories
Trainer
Batch
Update Weights
Reshape
Updated Weights
Parameter Server
Weights buffer
k
k-1
k-2
k-3
Updated Weights

1. Orchestration
2. Programming Model
3. Performance

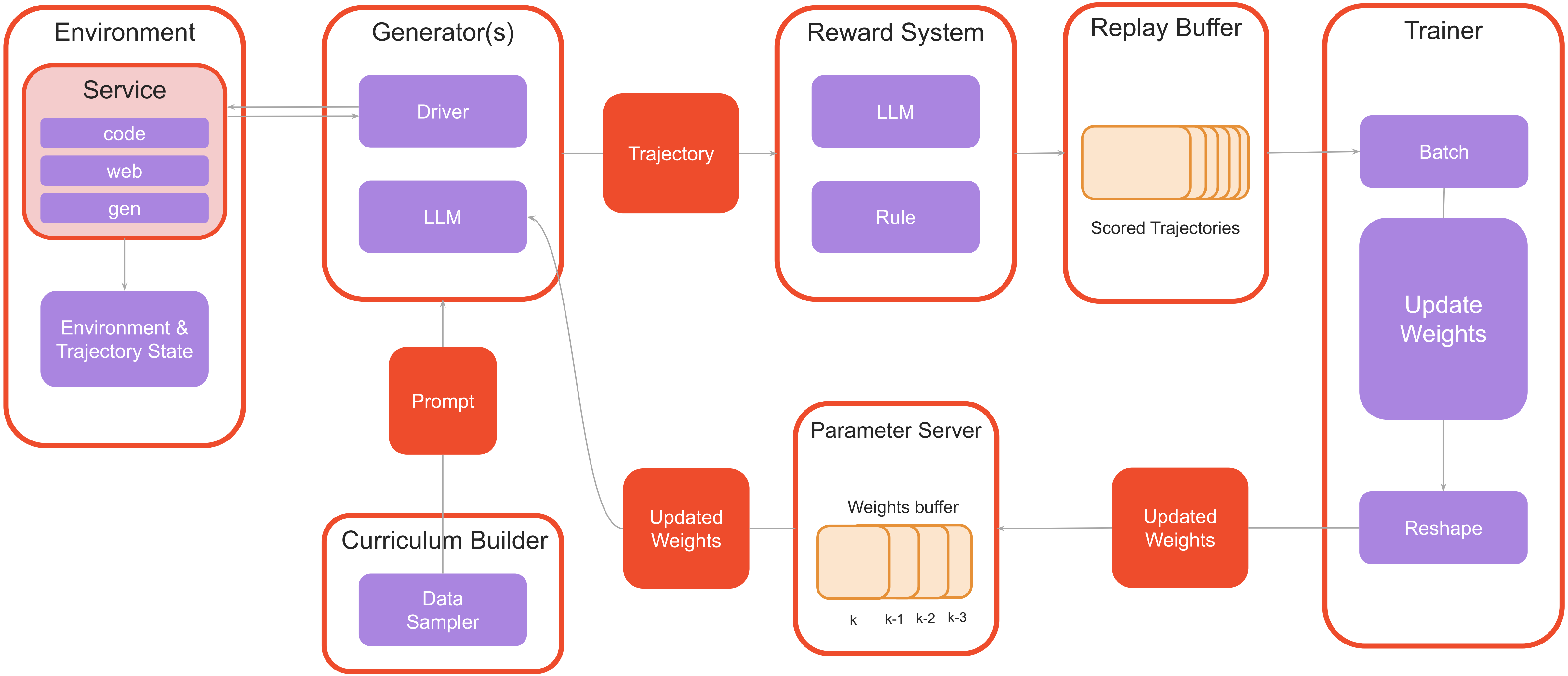
DISTRIBUTED RL CHALLENGES

1 Orchestration

2 Programming Model

3 Performance

RL SYSTEMS - ORCHESTRATION



1. Orchestration
2. **Programming Model**
3. Performance

```python
if rank == 0:
    # Coordinator logic
    gather_trajectories(...)
    broadcast_weights(...)

elif rank in generator_ranks:
    # Generator logic
    trajectories = generate(...)
    send_to_rank_0(trajectories)
    recv_weights_from_rank_0()

elif rank in trainer_ranks:
    # Trainer logic
    recv_trajectories_from_rank_0()
    new_weights = train_step(...)
    send_weights_to_rank_0(new_weights)
```

```python
# Under the hood, typically requires
# complex collective operations
dist.all_gather(tensor_list, tensor)
dist.scatter(output, scatter_list,
src=0)
```

SPMD force you to think about N ranks simultaneously, not logical components

```python
if rank == 0:
    # Coordinator logic
    gather_trajectories(...)
    broadcast_weights(...)

elif rank in generator_ranks:
    # Generator logic
    trajectories = generate(...)
    send_to_rank_0(trajectories)
    recv_weights_from_rank_0()

elif rank in trainer_ranks:
    # Trainer logic
    recv_trajectories_from_rank_0()
    new_weights = train_step(...)
    send_weights_to_rank_0(new_weights)
```

```python
# Under the hood, typically requires
# complex collective operations
dist.all_gather(tensor_list, tensor)
dist.scatter(output, scatter_list,
src=0)
```

SPMD force you to think about N ranks simultaneously, not logical components

```
if rank == 0:
    # Coordinator logic
    gather_trajectories(...)
    broadcast_weights(...)

elif rank in generator_ranks:
    # Generator logic
    trajectories = generate(...)
    send_to_rank_0(trajectories)
    recv_weights_from_rank_0()

elif rank in trainer_ranks:
    # Trainer logic
    recv_trajectories_from_rank_0()
    new_weights = train_step(...)
    send_weights_to_rank_0(new_weights)
```

```
# Under the hood, typically requires
# complex collective operations
dist.all_gather(tensor_list, tensor)
dist.scatter(output, scatter_list,
src=0)
```

SPMD force you to think about N ranks simultaneously, not logical components

```
if rank == 0:
    # Coordinator logic
    gather_trajectories(...)
    broadcast_weights(...)

elif rank in generator_ranks:
    # Generator logic
    trajectories = generate(...)
    send_to_rank_0(trajectories)
    recv_weights_from_rank_0()

elif rank in trainer_ranks:
    # Trainer logic
    recv_trajectories_from_rank_0()
    new_weights = train_step(...)
    send_weights_to_rank_0(new_weights)
```

```
# Under the hood, typically requires
# complex collective operations
dist.all_gather(tensor_list, tensor)
dist.scatter(output, scatter_list,
src=0)
```

SPMD force you to think about N ranks simultaneously, not logical components

```python
if rank == 0:
    # Coordinator logic
    gather_trajectories(...)
    broadcast_weights(...)

elif rank in generator_ranks:
    # Generator logic
    trajectories = generate(...)
    send_to_rank_0(trajectories)
    recv_weights_from_rank_0()

elif rank in trainer_ranks:
    # Trainer logic
    recv_trajectories_from_rank_0()
    new_weights = train_step(...)
    send_weights_to_rank_0(new_weights)
```

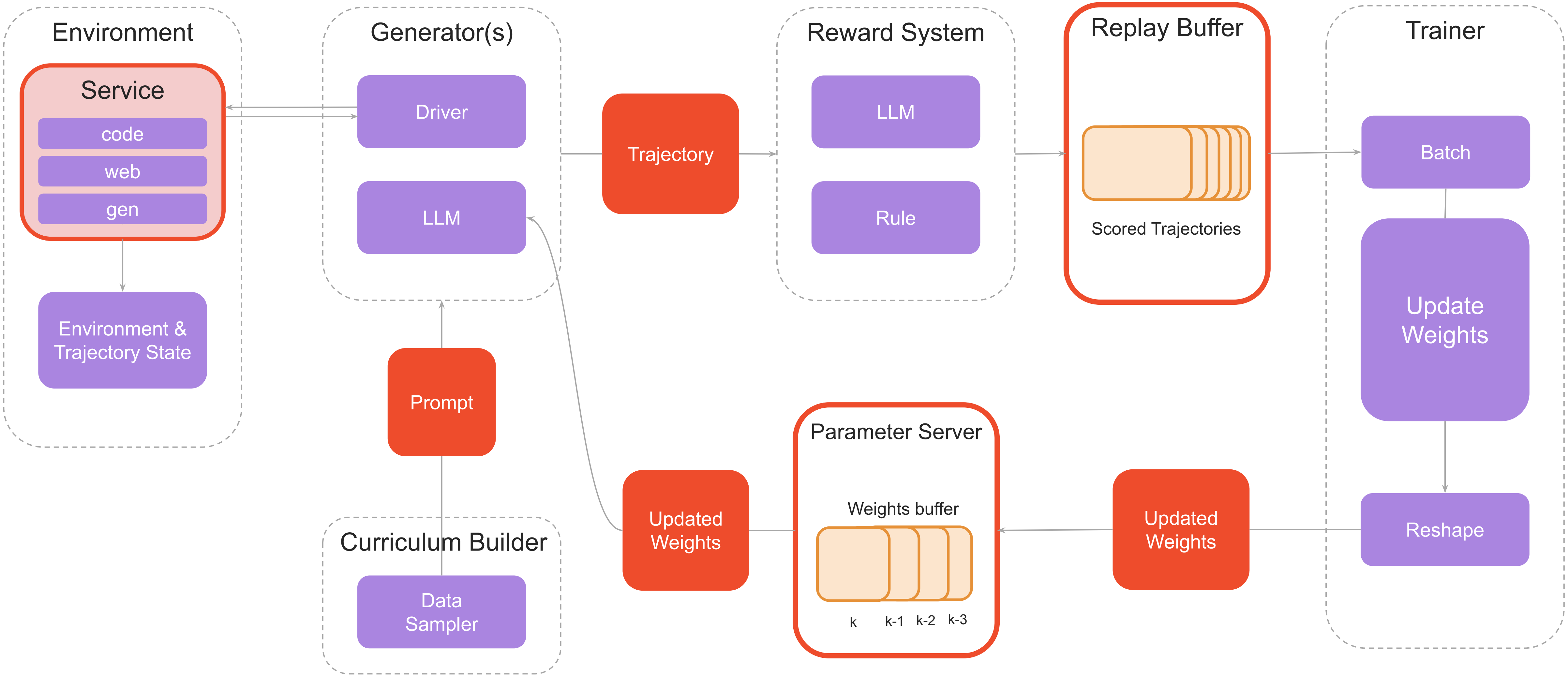
```python
# Under the hood, typically requires
# complex collective operations
dist.all_gather(tensor_list, tensor)
dist.scatter(output, scatter_list,
src=0)
```

SPMD force you to think about N ranks simultaneously, not logical components

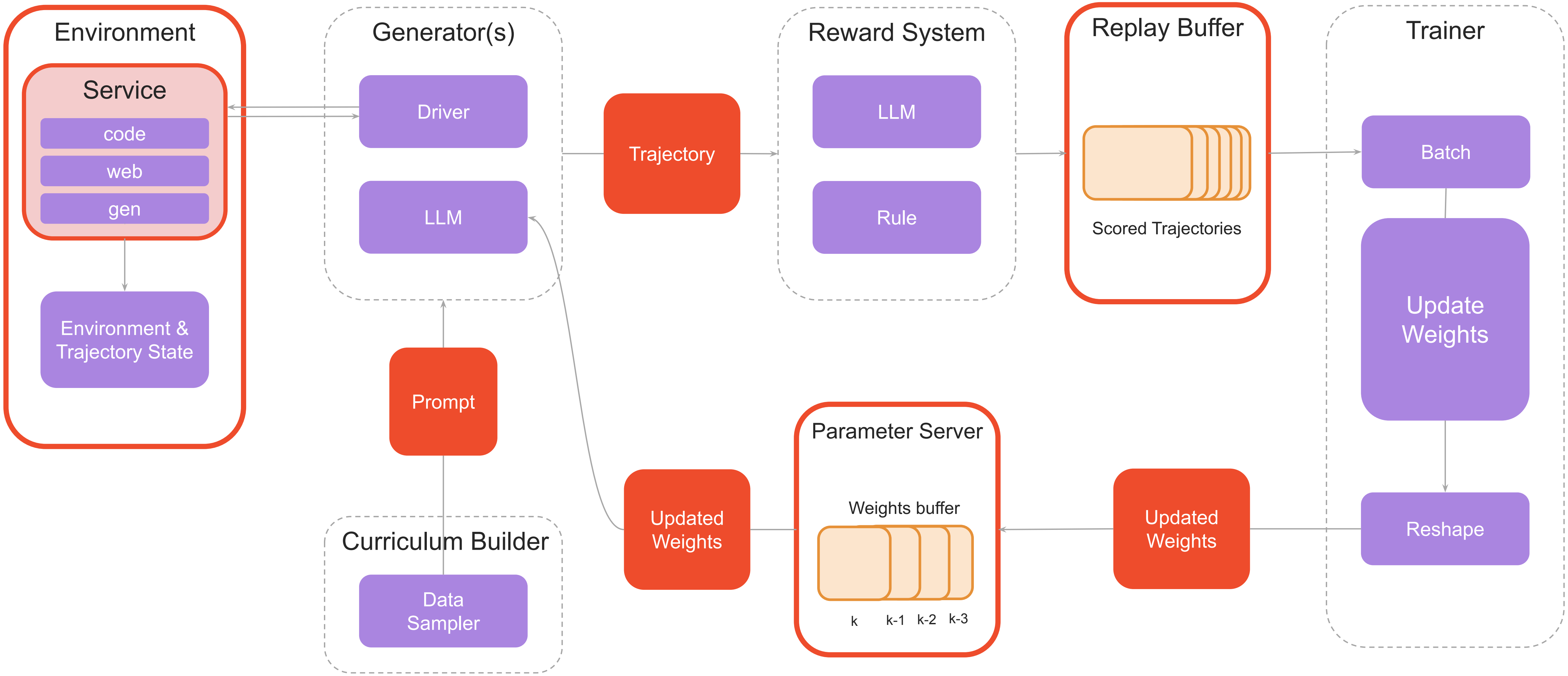
DISTRIBUTED RL CHALLENGES

1. Orchestration
2. Programming Model
3. Performance

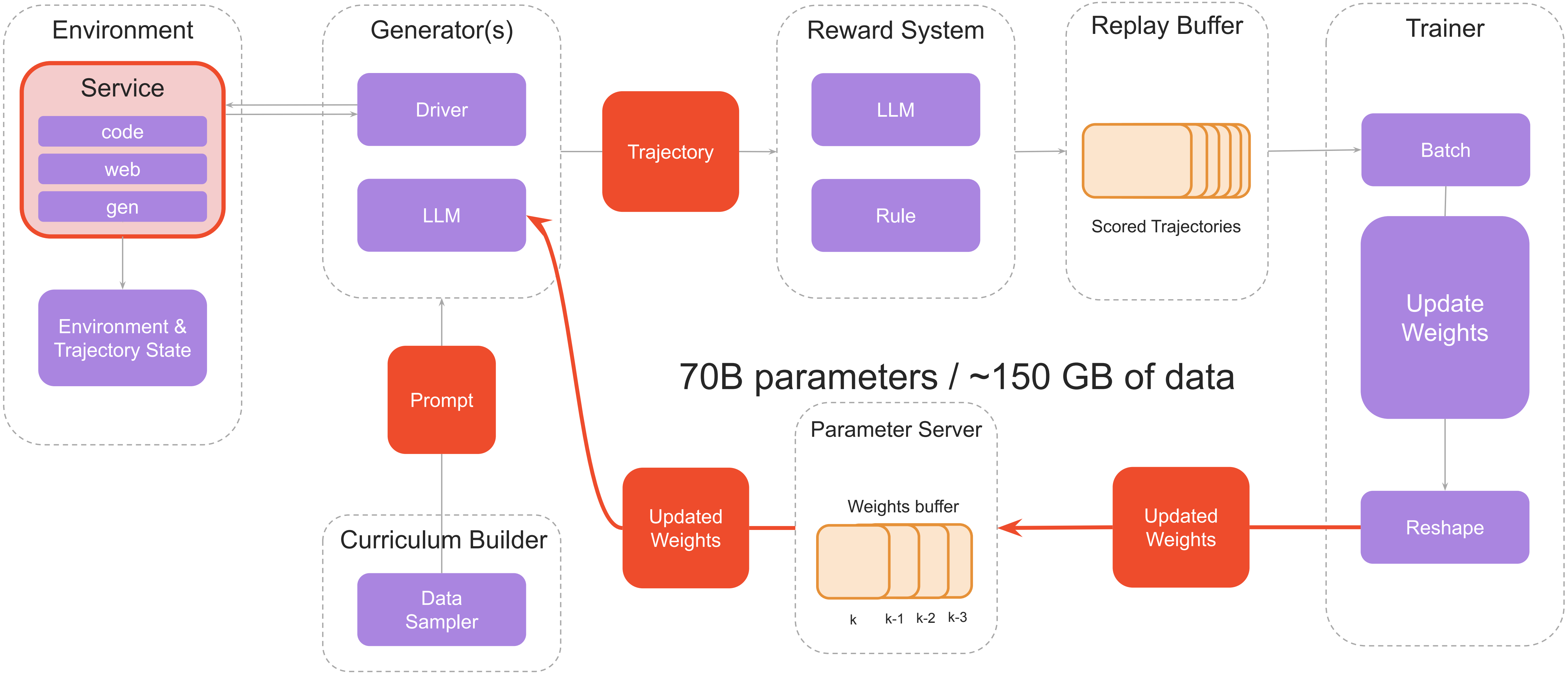
RL SYSTEMS - MANAGING BOTTLENECKS

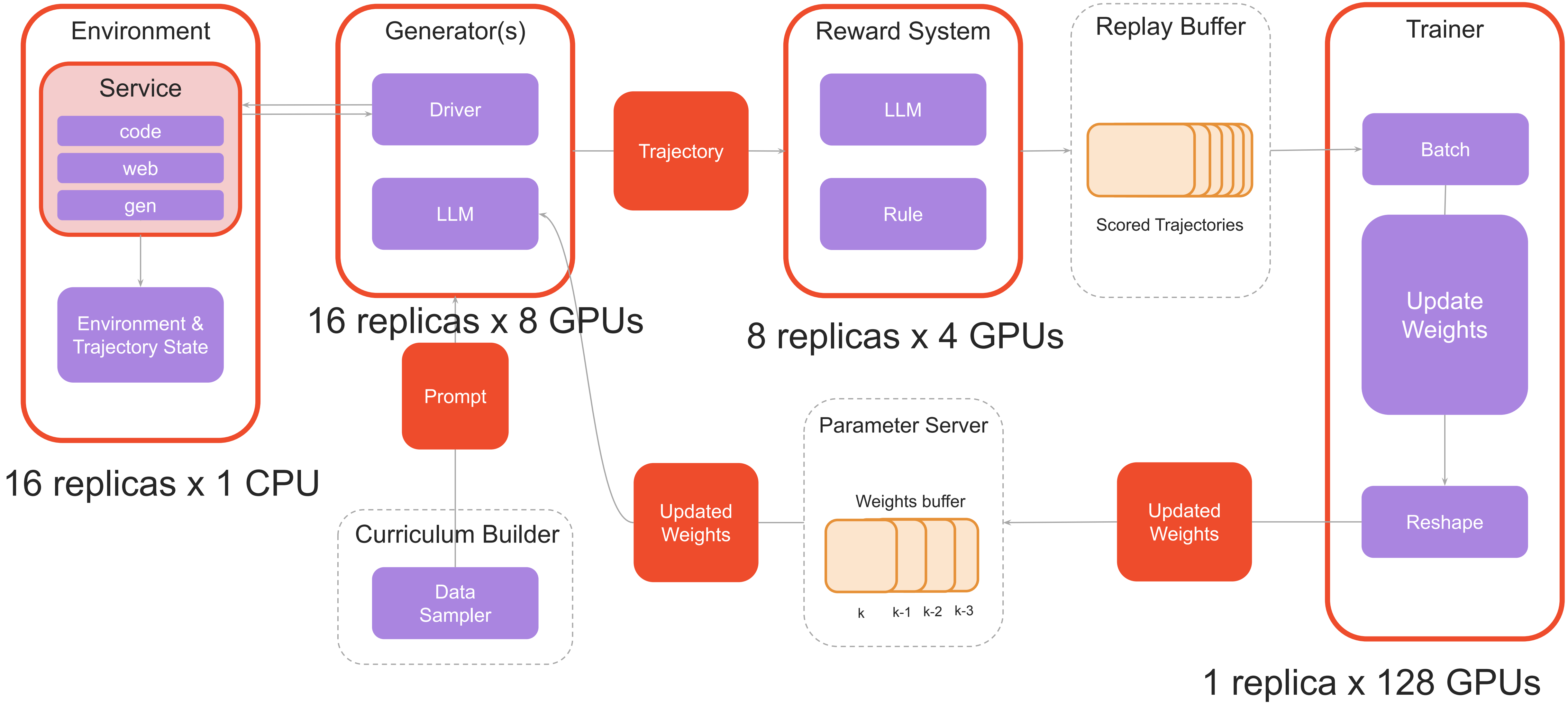


RL SYSTEMS - MANAGING BOTTLENECKS



RL SYSTEMS - WEIGHT SYNCHRONIZATION




Environment
Service
code
web
gen
Environment & Trajectory State
16 replicas x 1 CPU
Generator(s)
Driver
LLM
16 replicas x 8 GPUs
Prompt
Curriculum Builder
Data Sampler
Trajectory
Reward System
LLM
Rule
8 replicas x 4 GPUs
Replay Buffer
Scored Trajectories
Updated Weights
Parameter Server
Weights buffer
k
k-1
k-2
k-3
Updated Weights
Trainer
Batch
Update Weights
Reshape
1 replica x 128 GPUs

INTRODUCING MONARCH

PyTorch-native distributed programming framework based on scalable actor messaging

Actor Meshes

Actors grouped into collections. Broadcast messages to all members with a single call

RDMA Transfers

Fast point-to-point transfers of GPU/CPU memory using one-sided RDMA operations.

Fault Tolerance

Supervision trees provide automatic failure recovery and propagation.

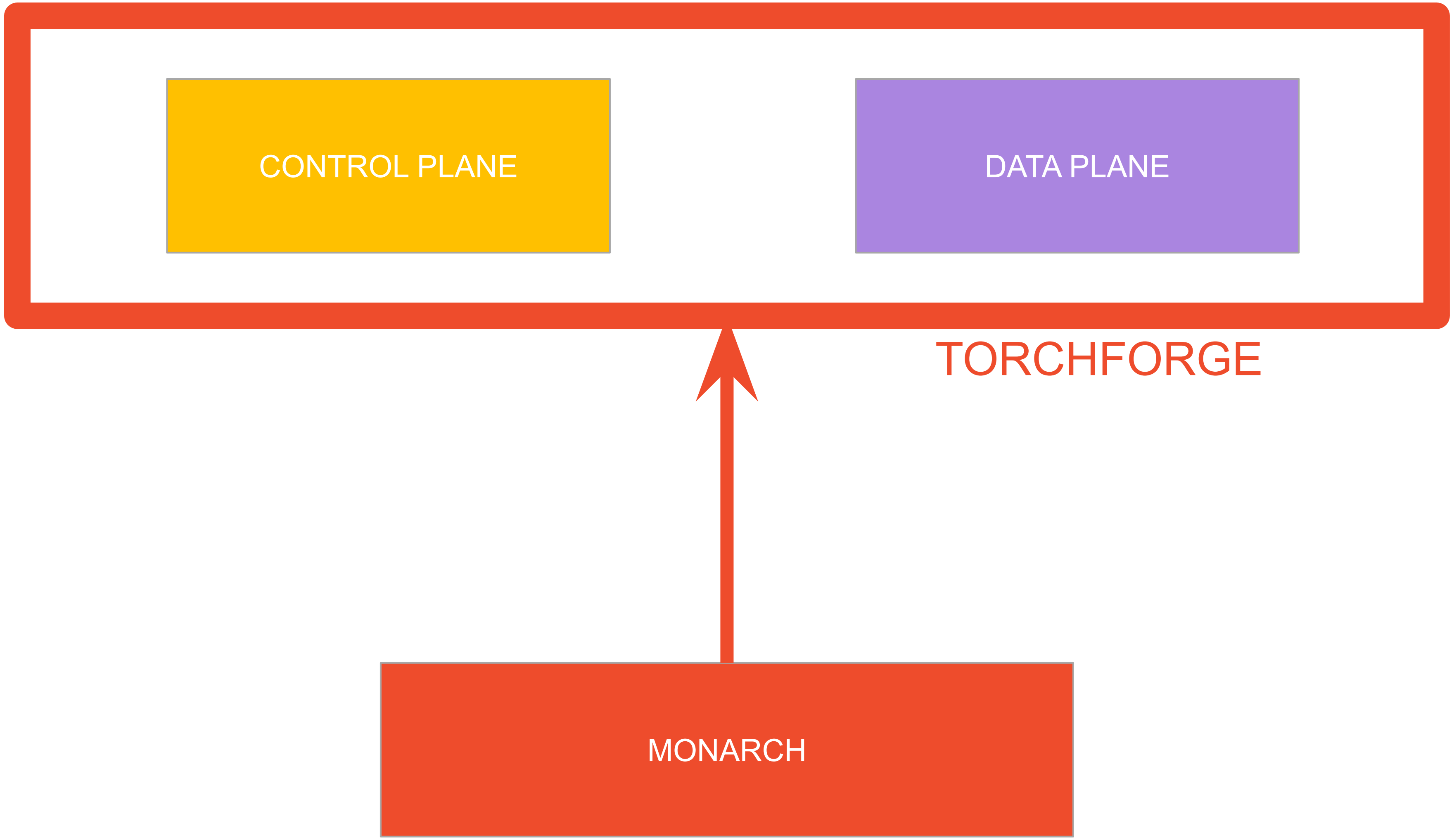
Imperative Python API

```python
# Spawn 8 processes, one per GPU
procs = this_host().spawn_procs({"gpus": 8})

# Define actor and create mesh
class Trainer(Actor):
    @endpoint
    def train(self, step: int): ...

trainers = procs.spawn("trainers", Trainer)
trainers.train.call(step=0).get()  # Broadcast to ALL!
```

Write distributed code imperatively, not with SPMD rank logic

TORCHFORGE CORE

TORCHFORGE CORE



Services	TorchStore
Control Plane of TorchForge Makes decisions about how and where data should flow, managing configuration, routing and orchestration.	Data Plane of TorchForge Actually moves and stores data according to those decisions

Create a Service

```
# Define resource requirements
policy = PolicyActor.options(
    hosts=1,
    procs=8,
    with_gpus=True,
    num_replicas=16
).as_service()

# 16 replicas × 8 GPUs each = 128 GPUs
# Automatic load balancing across
replicas
```

Interact with Services

```
# route() - load balanced to ONE replica
response = await
policy.generate.route(prompt)

# fanout() - broadcast to ALL replicas
await policy.update.fanout(version)
```

What Services Give You

Automatic Load Balancing across replicas

Fault tolerance with auto-restart

Ephemeral life cycle (starts/stops with the job)

Independent scaling per component

Heterogeneous Scaling

Policy
16 replicas x
8 GPUs

Trainer
1 replicas x
128 GPUs

Reward
4 replicas x
4 GPUs

Coder
16 replicas x
0 GPUs

What TorchStore Provides

Lightning-fast updates

In-memory storage + RDMA transfers enable weight sync in seconds, not minutes

Async Support

Storage in widely available DRAM enables weight sync without GPU interruption

Automatic Resharding

DTensor integration seamlessly converts between any distributed topology (FSDP ↔ Tensor Parallel)

Best possible UX

Simple put/get APIs abstract away all distributed complexity

Simple APIs

```
# Store sharded weights from trainer
await ts.put("policy_v42", dtensor)
# DTensor with FSDP sharding

# Fetch with different sharding for
inference
await ts.get("policy_v42", dtensor)
# DTensor with Tensor Parallel sharding

# Resharding happens automatically!
```

```
async def generate_episode(dataloader, policy, reward, replay_buffer):
    # Sample a prompt from the dataset
    prompt, target = await dataloader.sample.route()

    # Generate response using the policy
    response = await policy.generate.route(prompt)

    # Evaluate the response quality
    reward_value = await reward.evaluate.route(prompt, response, target)

    # Store the episode for training
    await replay_buffer.add.route(Episode(response, reward_value))
```

```python
async def synchronous_rl(batch_size):
    version = 0
    while True:
        # Collect full batch
        for _ in range(batch_size):
            await generate_episode(...)

        # Train on the complete batch
        batch = await buffer.sample.route(version, batch_size)
        await trainer.train_step.route(batch)

        # Update weights in lockstep
        await policy.update_weights.fanout(version + 1)
        version += 1
```

```python
async def synchronous_rl(batch_size):
    version = 0
    while True:
        # Collect full batch
        for _ in range(batch_size):
            await generate_episode(...)

        # Train on the complete batch
        batch = await buffer.sample.route(version, batch_size)
        await trainer.train_step.route(batch)

        # Update weights in lockstep
        await policy.update_weights.fanout(version + 1)
        version += 1
```

```python
async def synchronous_rl(batch_size):
    version = 0
    while True:
        # Collect full batch
        for _ in range(batch_size):
            await generate_episode(...)

        # Train on the complete batch
        batch = await buffer.sample.route(version, batch_size)
        await trainer.train_step.route(batch)

        # Update weights in lockstep
        await policy.update_weights.fanout(version + 1)
        version += 1
```

```python
async def synchronous_rl(batch_size):
    version = 0
    while True:
        # Collect full batch
        for _ in range(batch_size):
            await generate_episode(...)

        # Train on the complete batch
        batch = await buffer.sample.route(version, batch_size)
        await trainer.train_step.route(batch)

        # Update weights in lockstep
        await policy.update_weights.fanout(version + 1)
        version += 1
```

```python
async def async_rl_loop(num_rollout_loops: int):
    # Start concurrent rollout loops
    rollouts = [asyncio.create_task(continuous_rollouts())
      for _ in range(num_rollout_loops)]

    # Start continuous training
    training = asyncio.create_task(continuous_training())

    await asyncio.gather(*rollouts, training)
```

```python
async def async_rl_loop(num_rollout_loops: int):
    # Start concurrent rollout loops
    rollouts = [asyncio.create_task(continuous_rollouts())
      for _ in range(num_rollout_loops)]

    # Start continuous training
    training = asyncio.create_task(continuous_training())

    await asyncio.gather(*rollouts, training)
```

```python
async def async_rl_loop(num_rollout_loops: int):
    # Start concurrent rollout loops
    rollouts = [asyncio.create_task(continuous_rollouts())
      for _ in range(num_rollout_loops)]

    # Start continuous training
    training = asyncio.create_task(continuous_training())

    await asyncio.gather(*rollouts, training)
```

THANK YOU