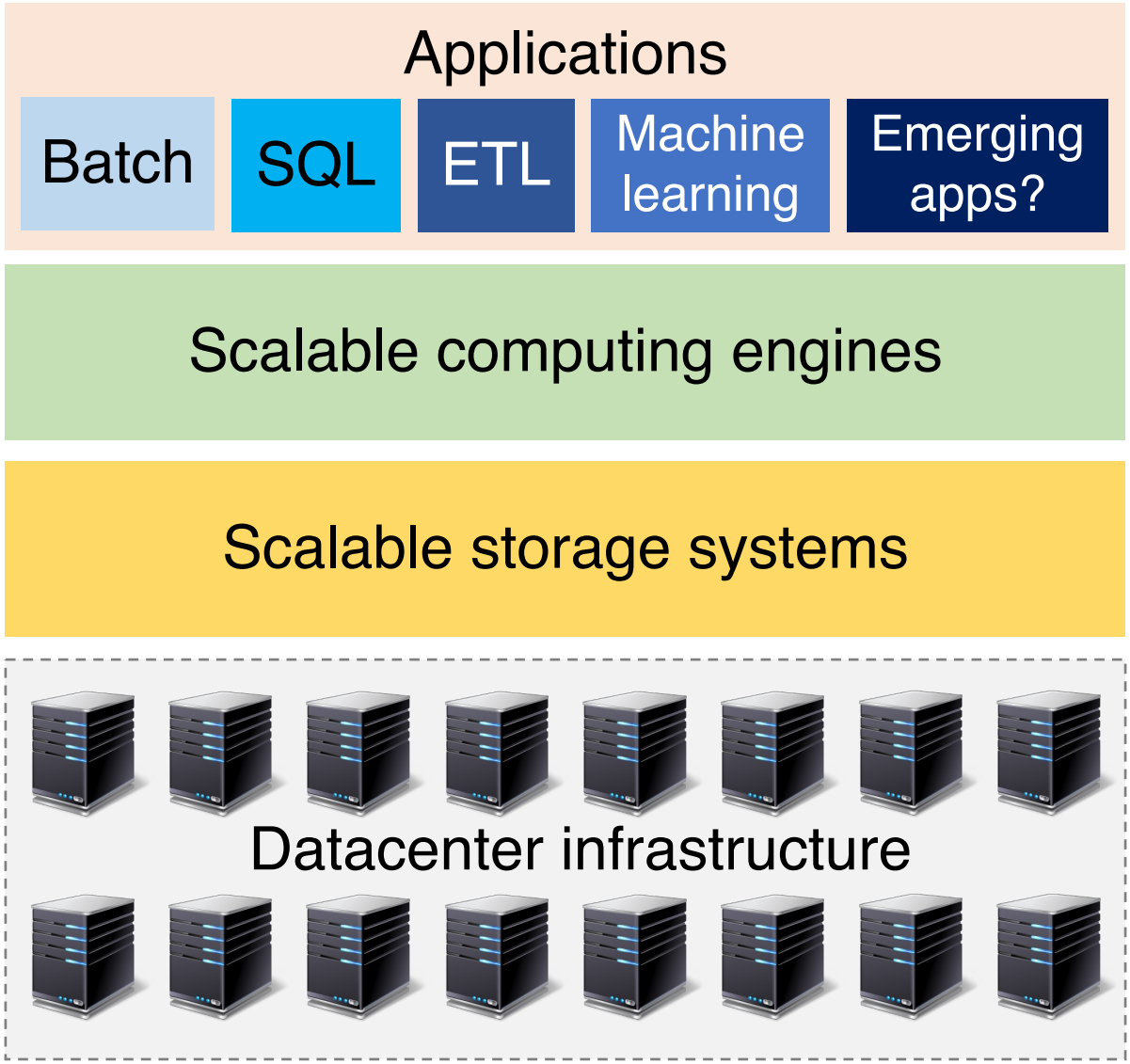
Reinforcement Learning Systems: Ray

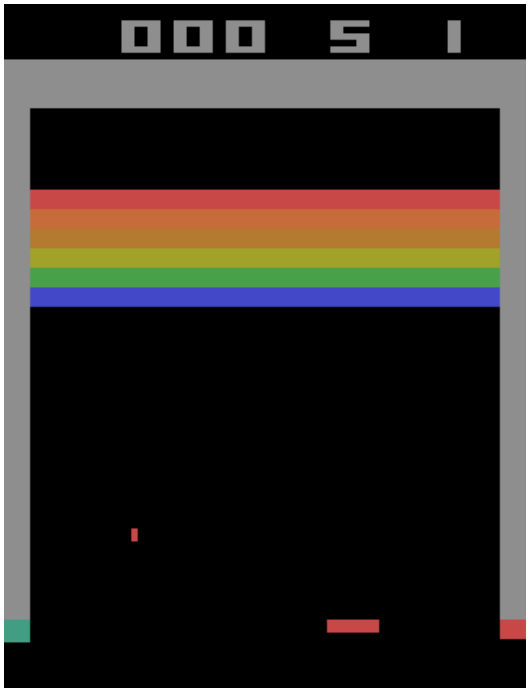
DS 5110: Big Data Systems (Spring 2023)

Lecture 8a

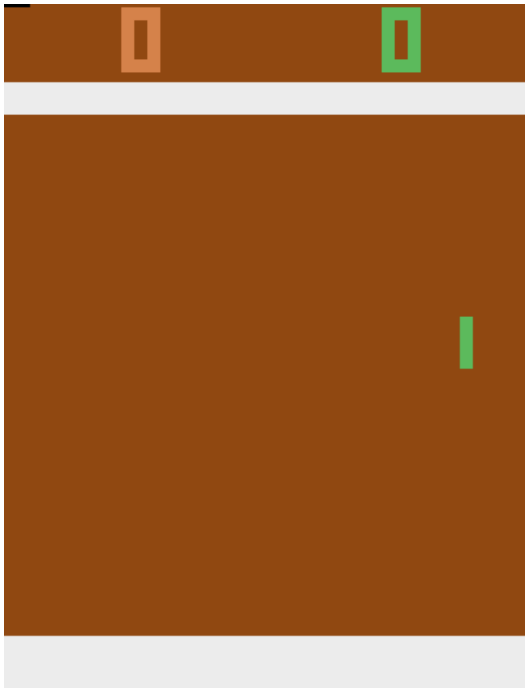
Yue Cheng


Applications
Batch
SQL
ETL
Machine learning
Emerging apps?
Scalable computing engines
Scalable storage systems
Datacenter infrastructure

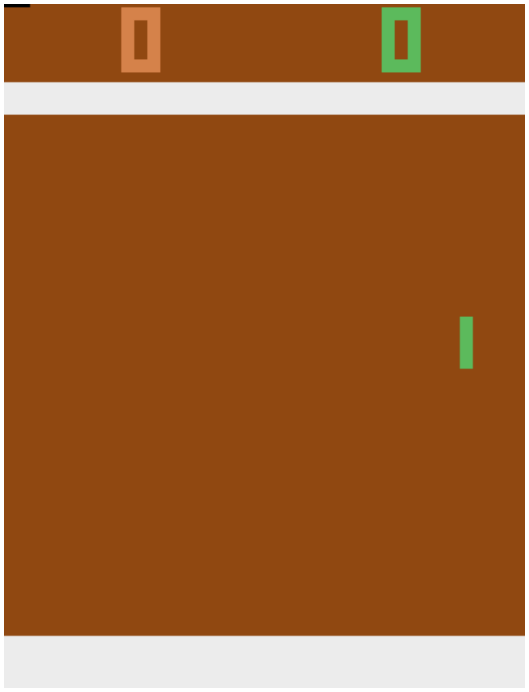
Reinforcement learning



Atari breakout



Pong: after 30 mins of training



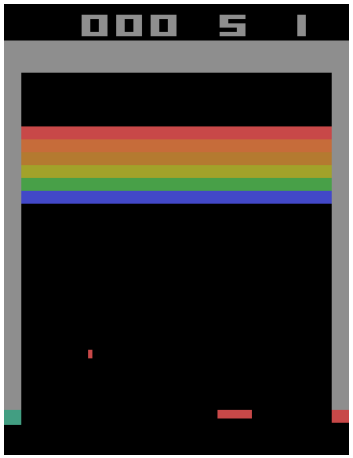
Pong: DQN wins like a boss

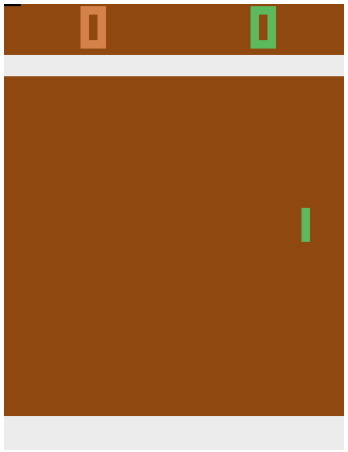
*: Playing Atari with Deep Reinforcement Learning: https://arxiv.org/abs/1312.5602

RL application pattern

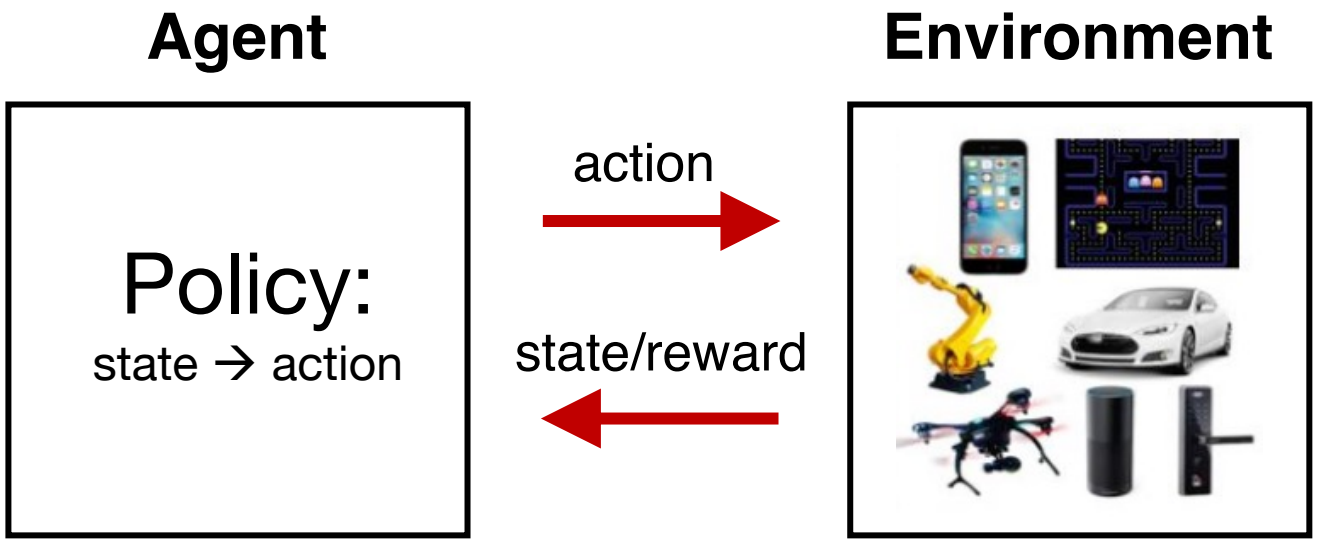
- Process inputs from different sensors in parallel & real-time

- Execute large number of simulations, e.g., up to 100s of millions

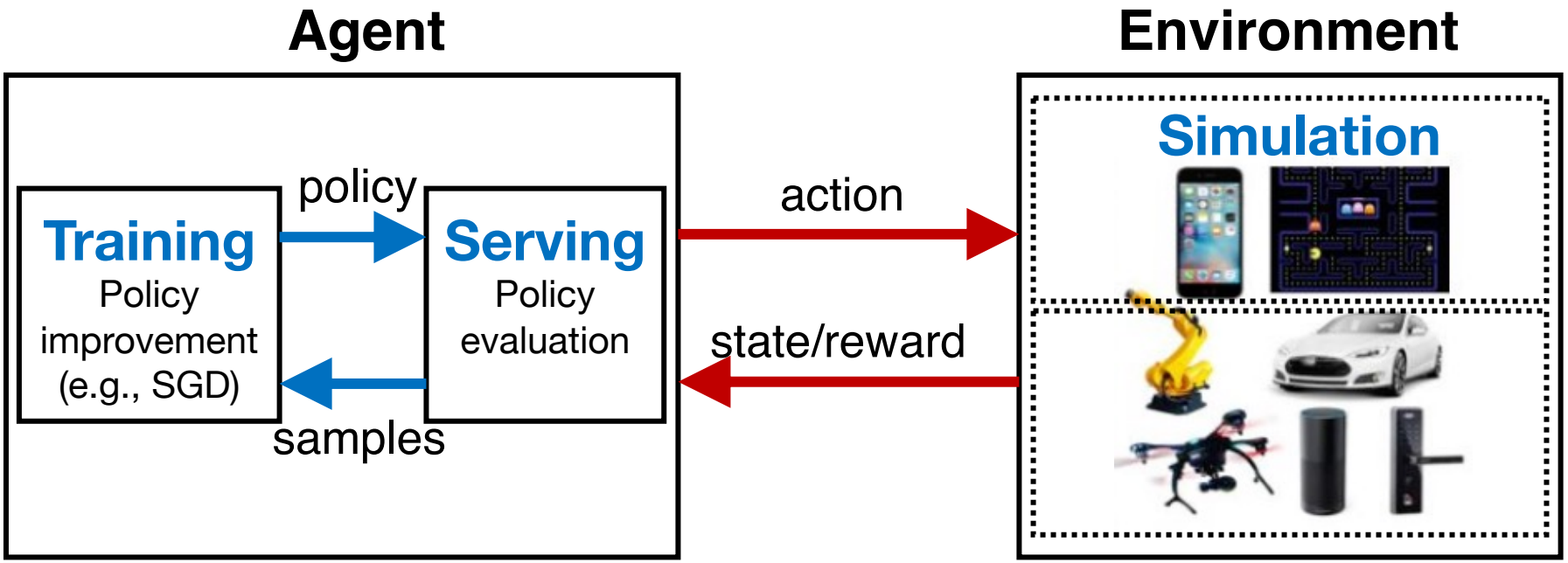




RL setup


Agent
Environment
Policy:
state → action
action
state/reward

RL setup in more detail



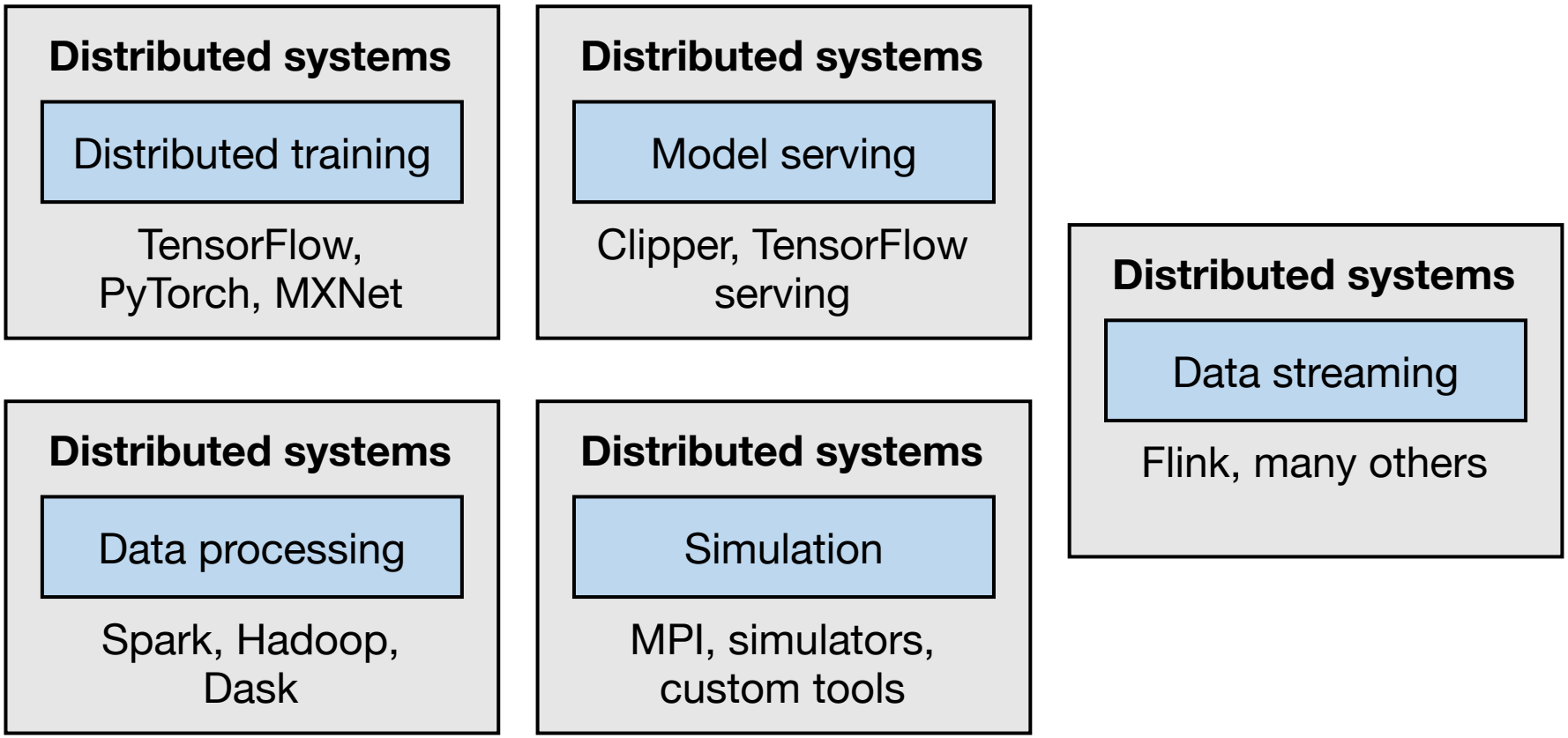
RL application pattern

- Process inputs from different sensors in parallel & real-time
- Execute large number of simulations, e.g., up to 100s of millions
- Rollouts outcomes are used to update policy (e.g., SGD)

RL application requirements

- Need to handle dynamic task graphs, where tasks have
  - Heterogeneous durations
  - Heterogenous computations
- Schedule millions of tasks / sec
- Make it easy to parallelize ML algorithms (often written in Python)

The ML/AI/data ecosystems today



Emerging AI applications require **stitching** together **multiple** disparate systems

Ad hoc integrations are difficult to manage and program!

Ray API

Tasks

```
futures = f.remote(args)
```

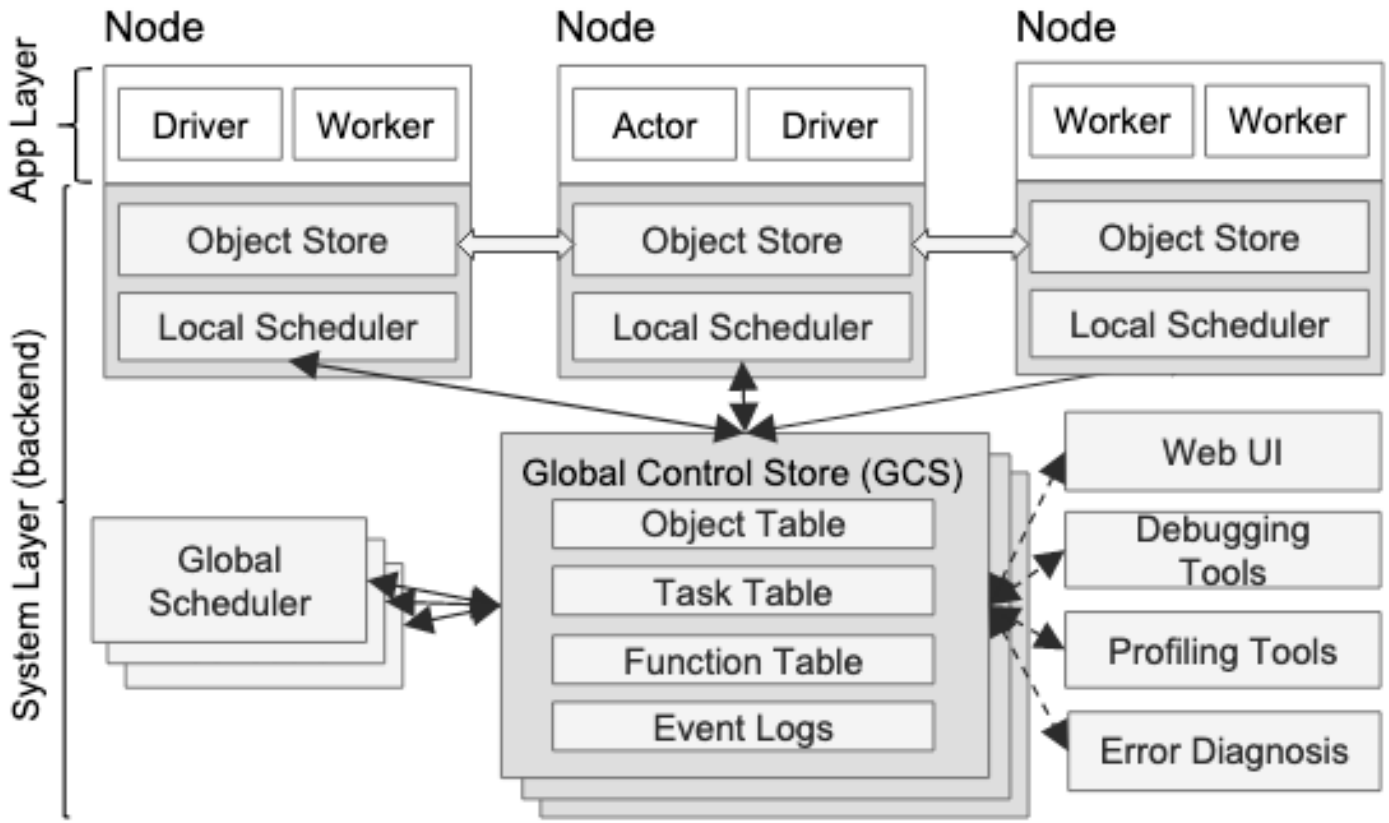
Actors

```
actor = Class.remote(args)
futures = actor.method.remote(args)
```

```
objects = ray.get(futures)
ready_futures = ray.wait(futures, k, timeout)
```

Ray API examples: Demo

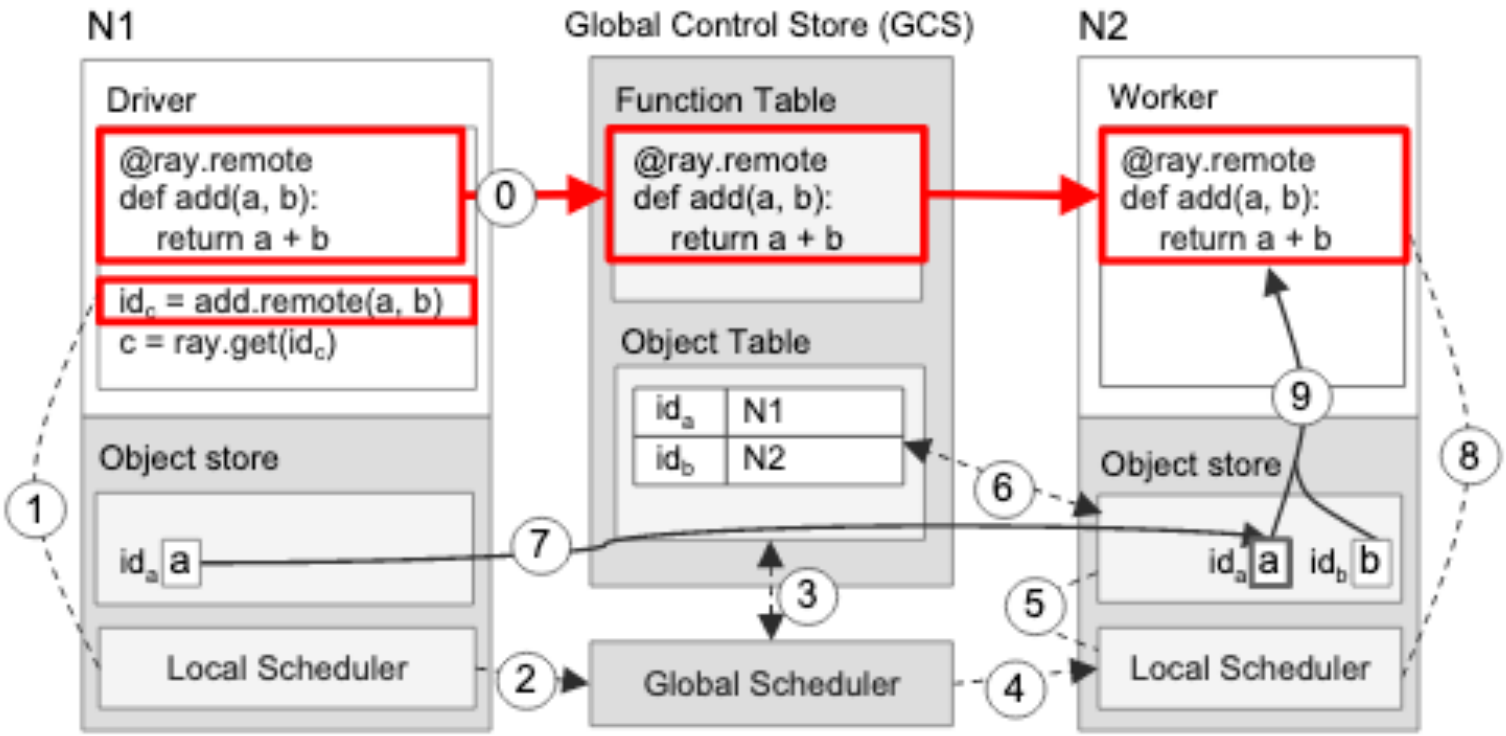
Ray architecture



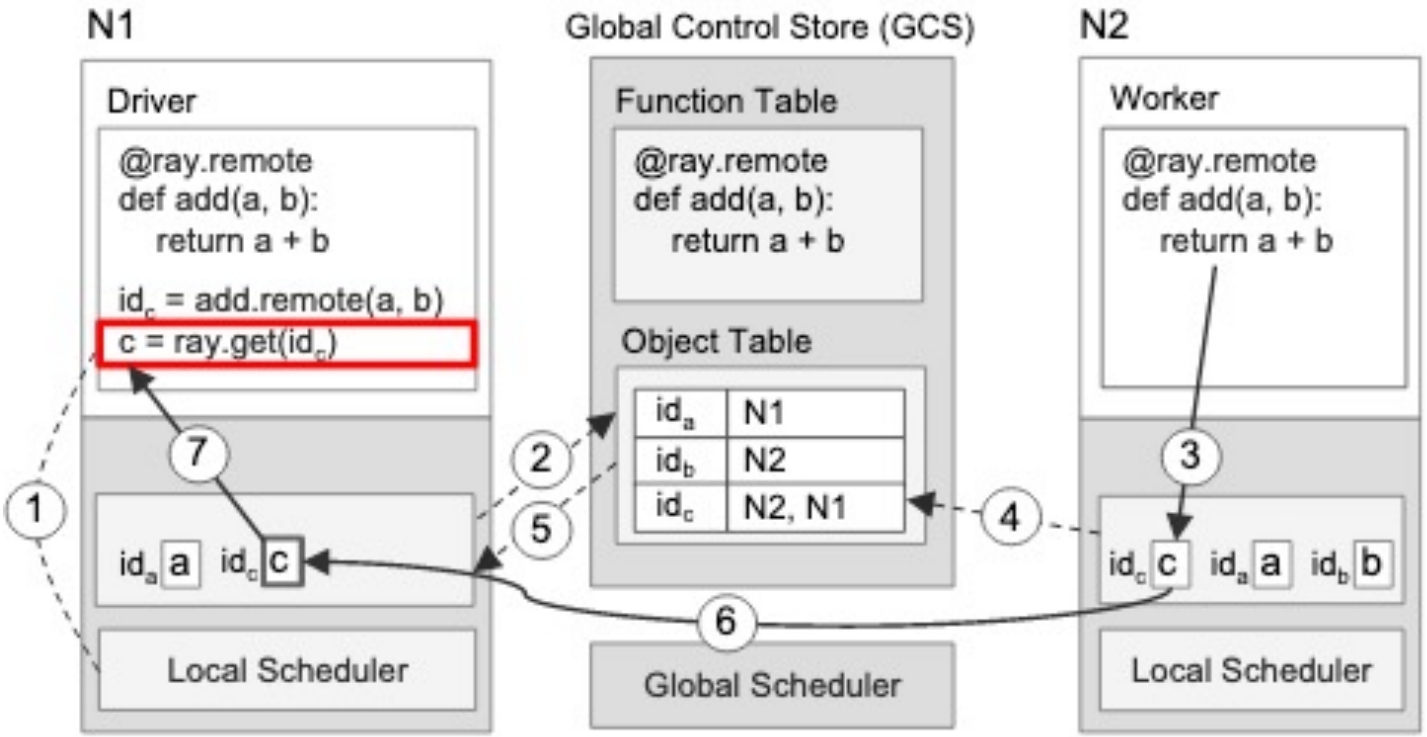
Global control store (GCS)

- Object table
- Task table
- Function table

Executing a task remotely



Returning the results of a remote task



This Wed: Federated Learning Systems