Hi!
Good morning!

CS 744: RAY

Shivaram Venkataraman

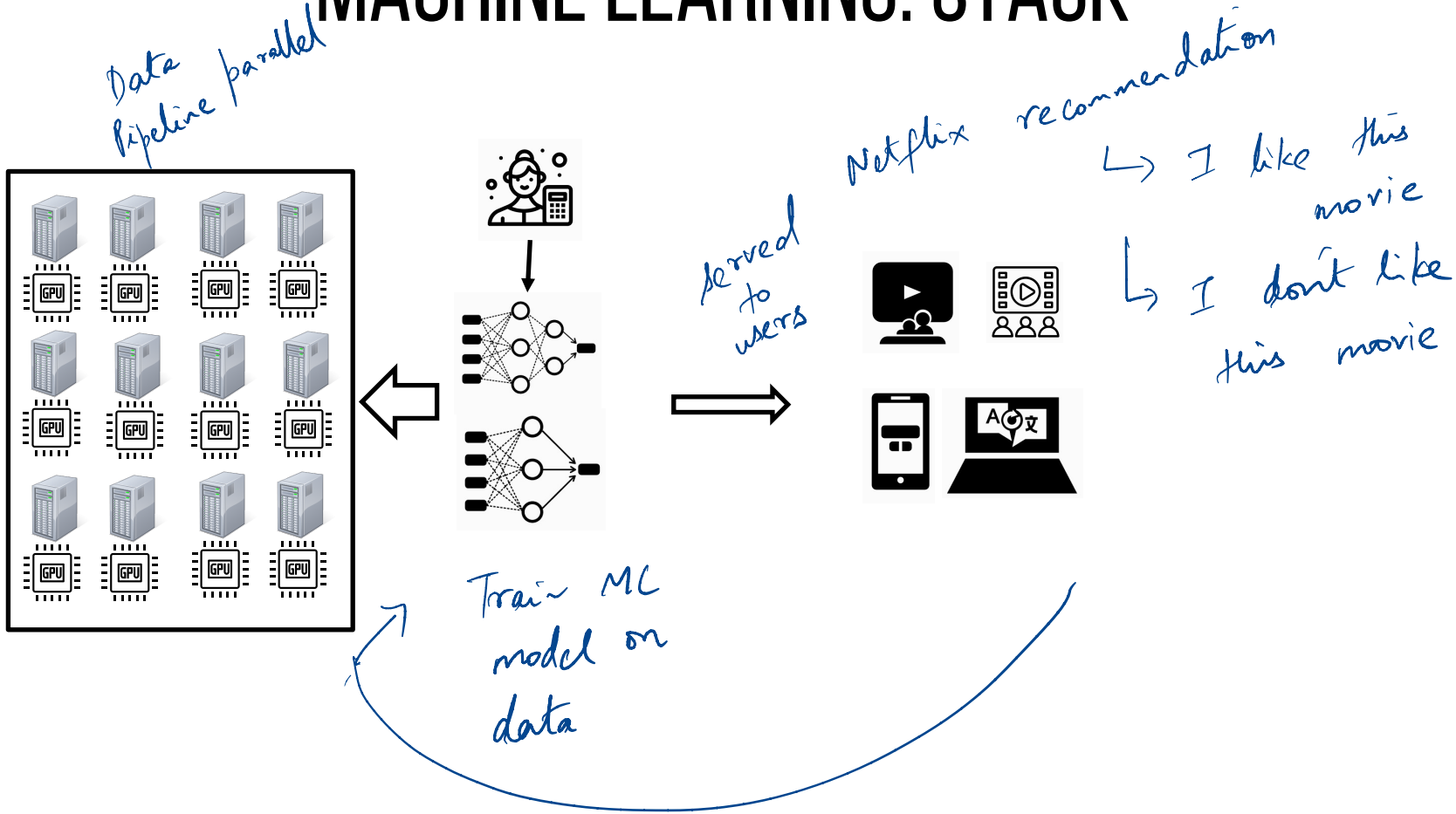
Fall 2021

ADMINISTRIVIA

- Assignment Two: Due tonight!
- Project proposal aka Introduction (10/25)
  - Introduction
  - Related Work
  - Timeline (with eval plan)

Confirm project title in couple of days

MACHINE LEARNING: STACK

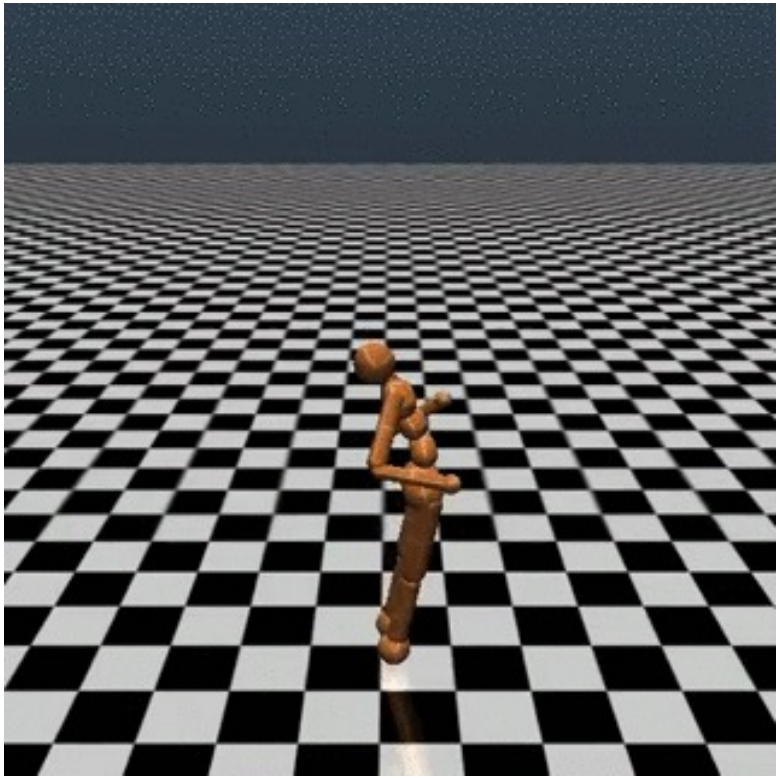


REINFORCEMENT LEARNING: APPLICATIONS

Robots

Game playing

Chess/ Go



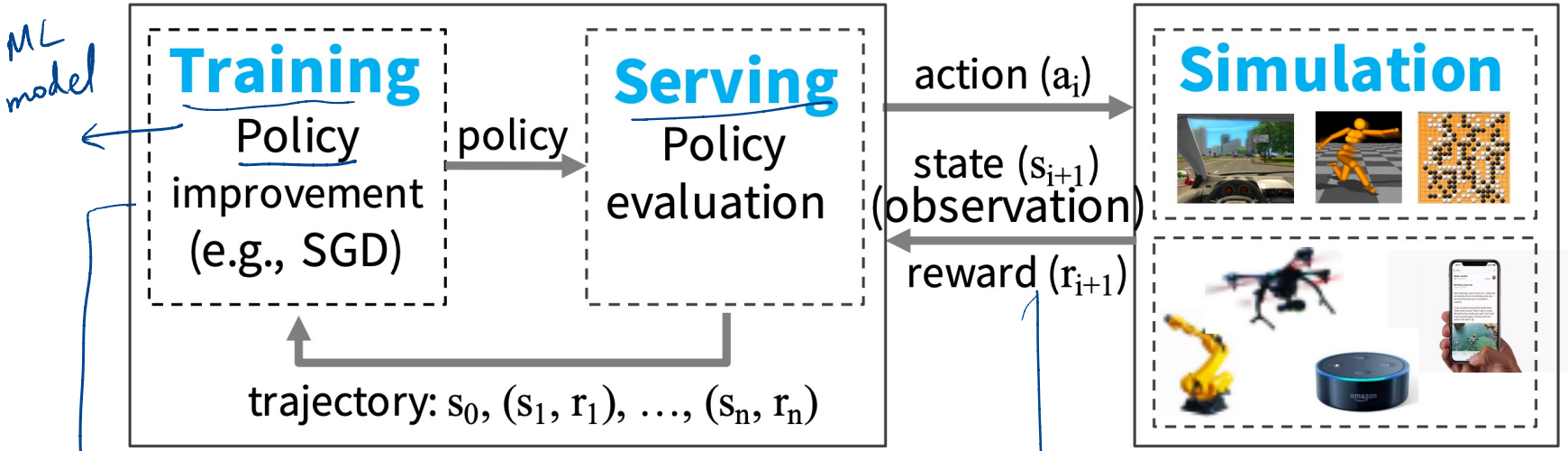
RL SETUP

entity that can perform an action

Agent

Environment

game console or simulator



given your current state what should your action be?

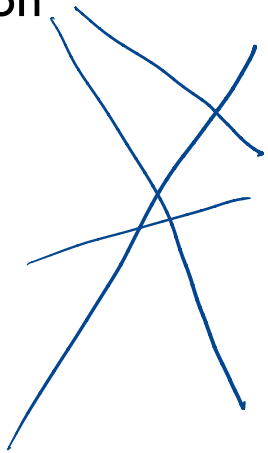
number of points

RL REQUIREMENTS

Simulation

Training

Serving



low latency for serving your model/policy

Dynamic execution

resources can also vary
↑ GPUs, CPUs etc.

Compute time can be variable
↳ milliseconds to hours

Stateful computation → model training needs prev. iteration model

Stateless → serving → simulation

RAY API

stateless and side-effect free

Tasks → similar to ideas in Spark etc.

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

f: local function

remote function

args: arguments to this function

futures: handle or future that can be used to access results

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

Nested remote functions

```
def f():
    ft = g.remote(a, b)
    return ft
```

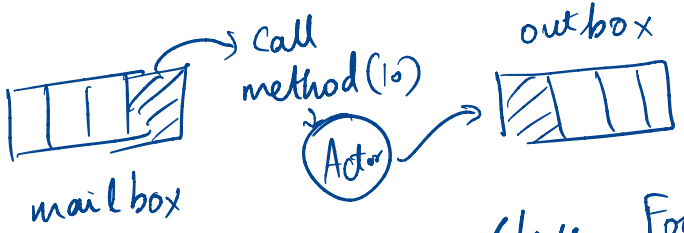
Pass futures as arguments

Actors

Class.remote

actor.method.remote

RAY API



Tasks

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

Class Foo {
int var = 10;
def method():
y

Instantiate to create Actor

Actors

Erlang, fault tolerant distributed computing

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

constructor

stateful computation
state is inside an actor

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

COMPUTATION MODEL

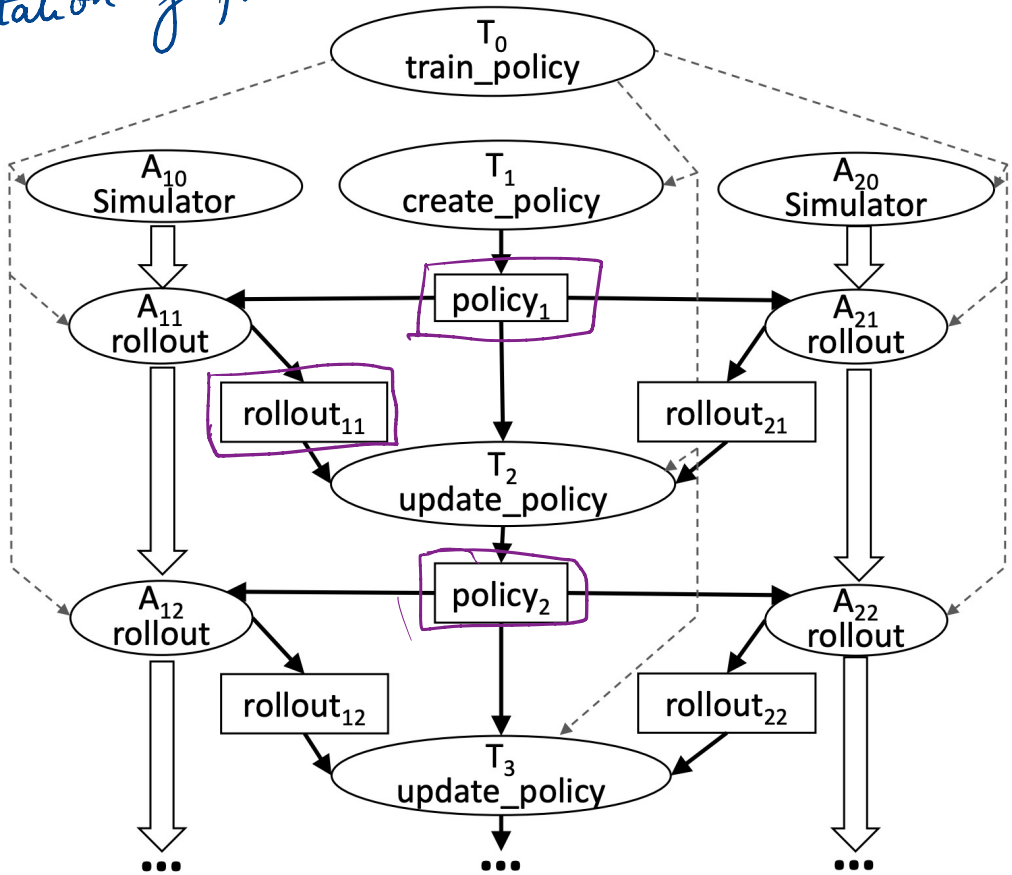
remote tasks and actor methods induce computation graph

Dotted lines → control edges

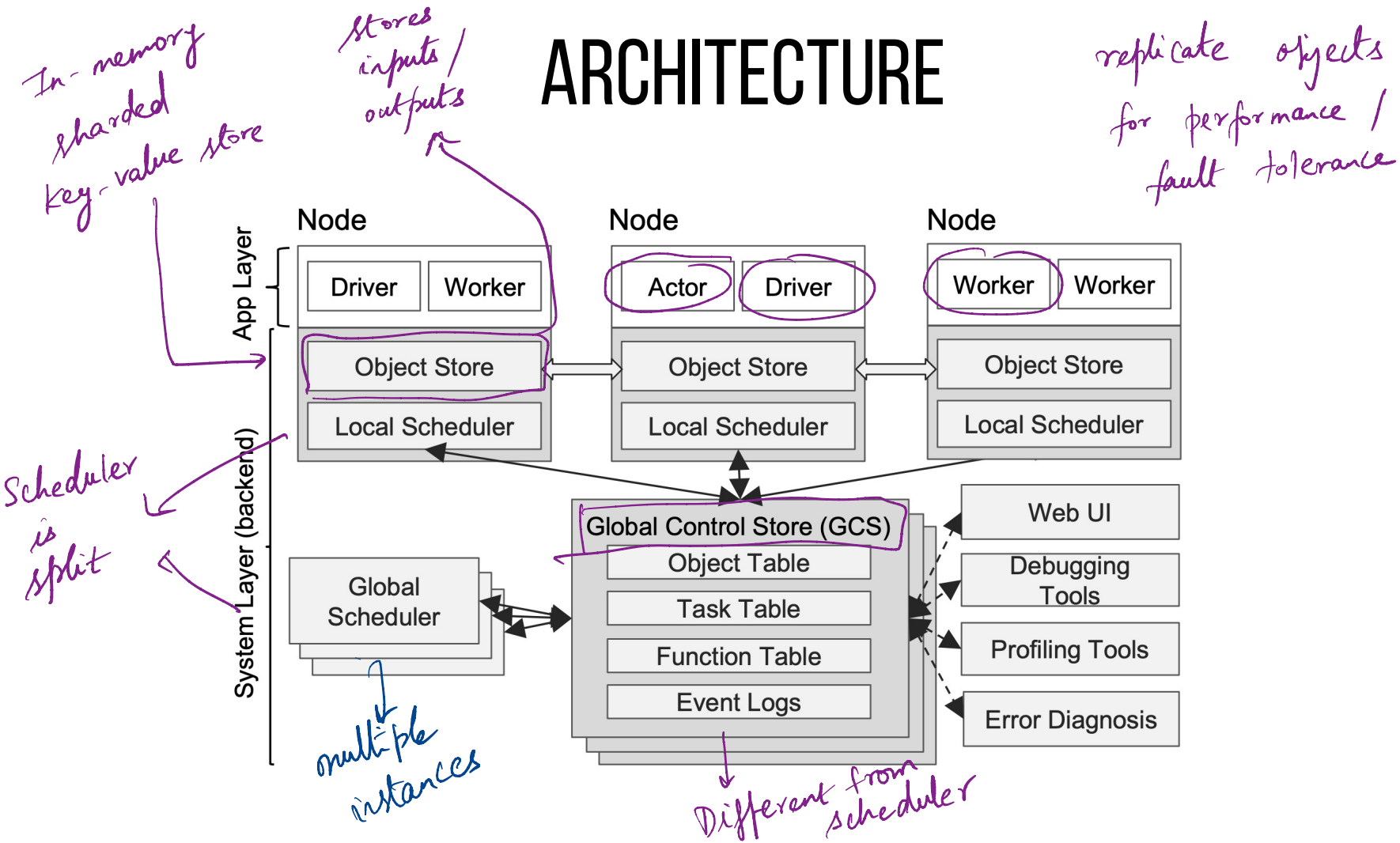
Black arrows → stateful edges (actor methods)

Solid lines → data dependencies

Lineage !



ARCHITECTURE



"Database" used by the scheduler ←

GLOBAL CONTROL STORE

Object table → metadata for the object store
which objects are stored where

Task table → lineage of tasks
which tasks are active, where

Function table → code blocks for functions
in the system

---

shard, replicate ⇒ scale, fault tolerance

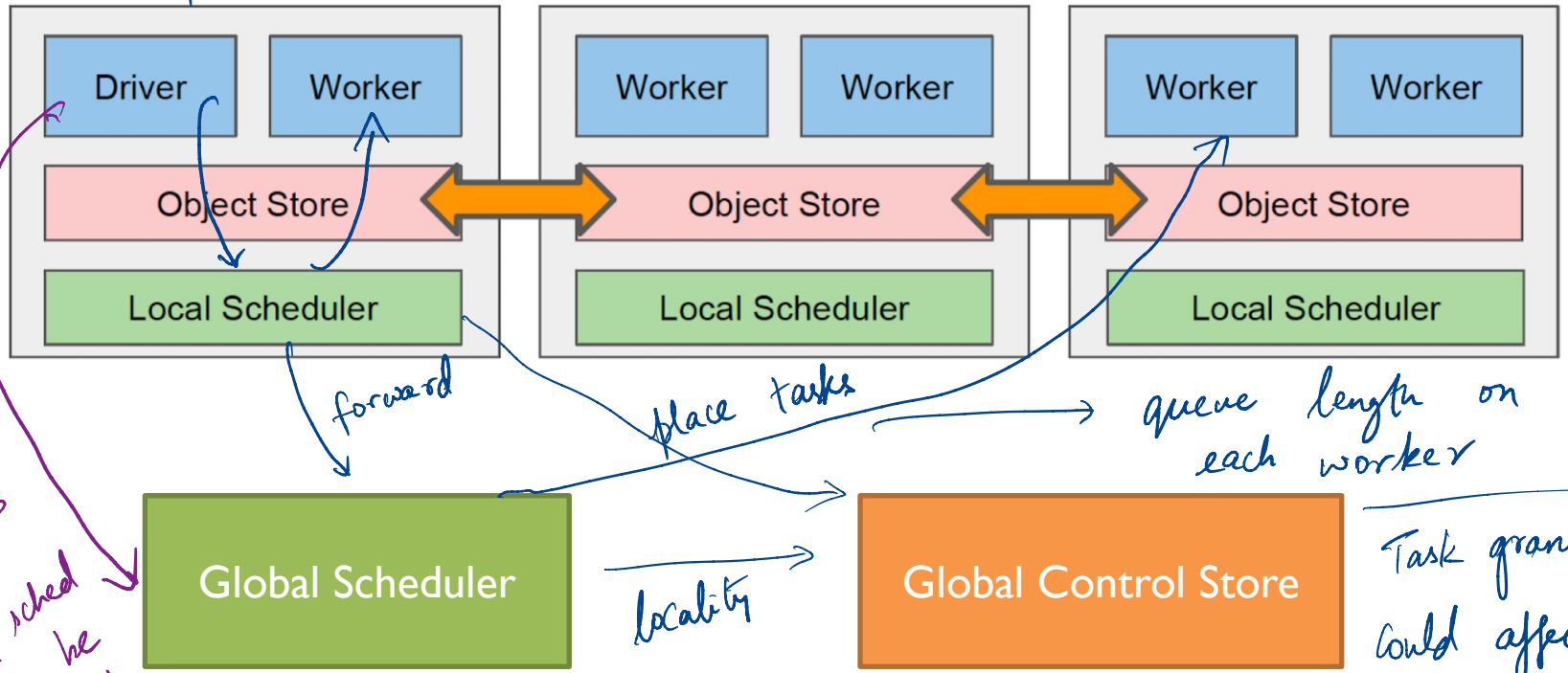
Objective:
Run tasks quickly
↳ tasks can be very small!

Separate out scheduler state from scheduler logic

RAY SCHEDULER

local scheduler can launch tasks on local workers

① queue time on worker > timeout forward to global sched.



FAULT TOLERANCE

Tasks → lineage, replay or re-execute tasks
side-effect free

Actors → check point the actor state periodically

GCS → chain replication → handle replica failures

Global
Scheduler → stateless ⇒ respawn a new scheduler

SUMMARY

Ray: Unified system for ML training, serving, simulation

Flexible API with support for

- Stateless tasks
- Stateful Actors

Distributed scheduling, Global control store

DISCUSSION

https://forms.gle/MnsCJA87CVhMmShs8

Consider you are implementing two apps: a deep learning model training and a sorting application. When will use tasks vs actors and why ?

Deep learning model training

- Use actors, have model as state. Iteration is a method call
- Mix actors & tasks
  Data parallel - each local gradient
  all reduce → Actor - optimizer.step()

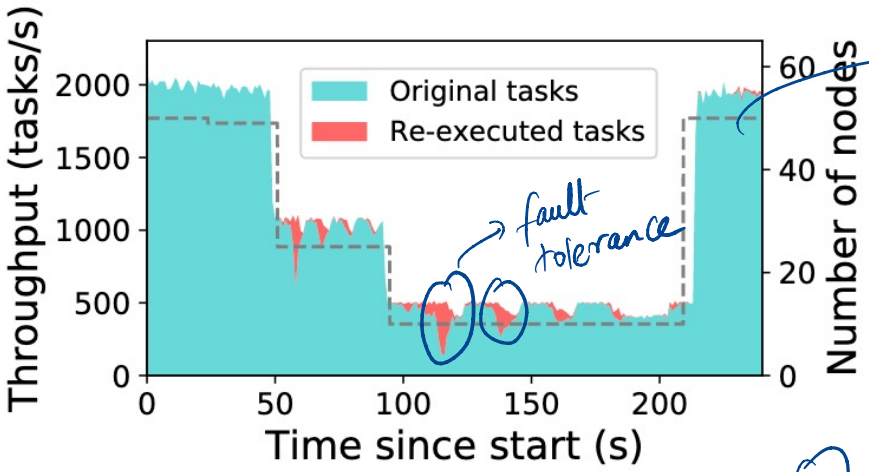
Sorting

- Merge sort
  → can we have data sorted so far as state ?
  → Partition to many actors ?
- Data, partition it, object store

Stateless tasks
intermediate data on object store !

At t = 50, 100s num workers decreases.

Per-worker tput remains same

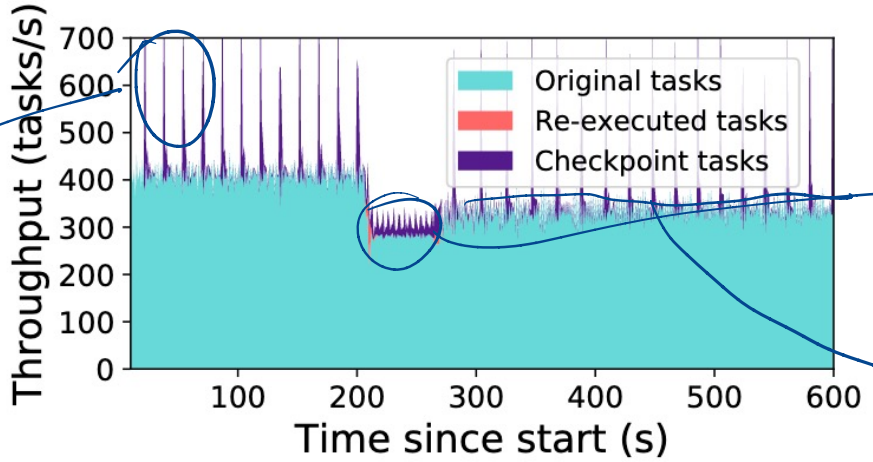


(a) Task reconstruction

Add workers back, tput goes up again "elastic"

T1 → T2 → T3 ... X

checkpointing happens



(b) Actor reconstruction

nodes fail reconstruction

tput recovers after that

NEXT STEPS

Next class: Pollux