Ray
A Distributed Execution Framework for Emerging AI Applications

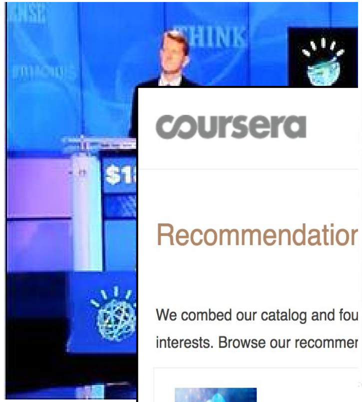
BDD/RISE mini-retreat

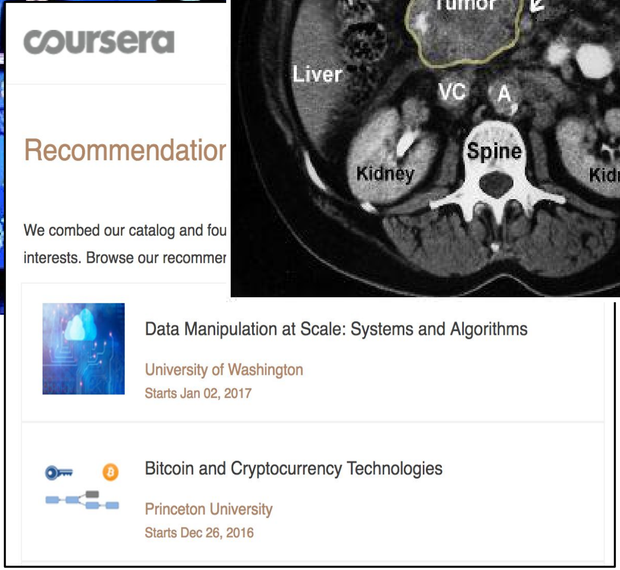
Philipp Moritz

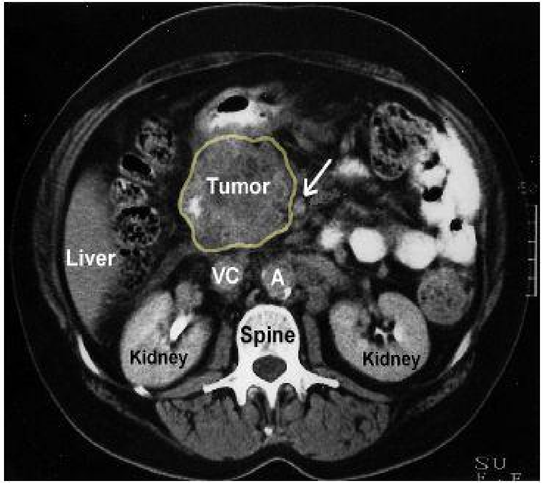

Berkeley
UNIVERSITY OF CALIFORNIA
BERKELEY

Emerging AI applications

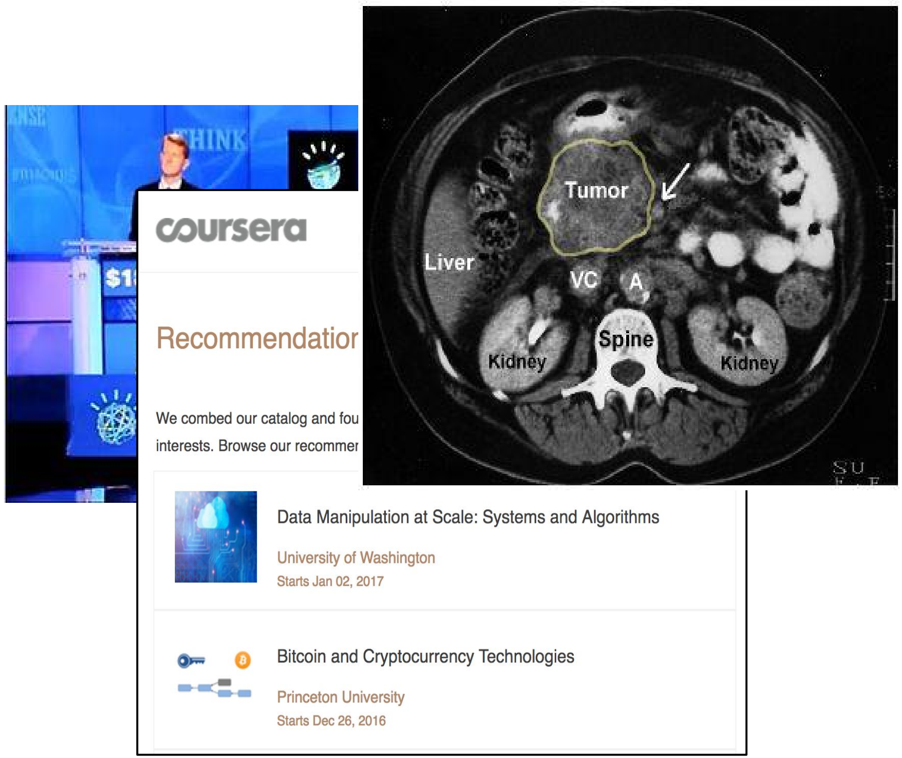
Emerging AI applications

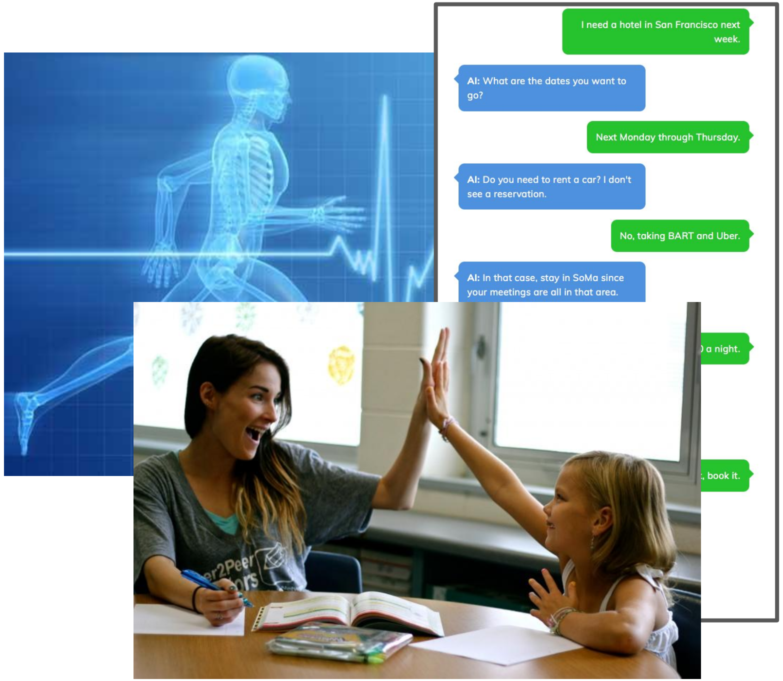




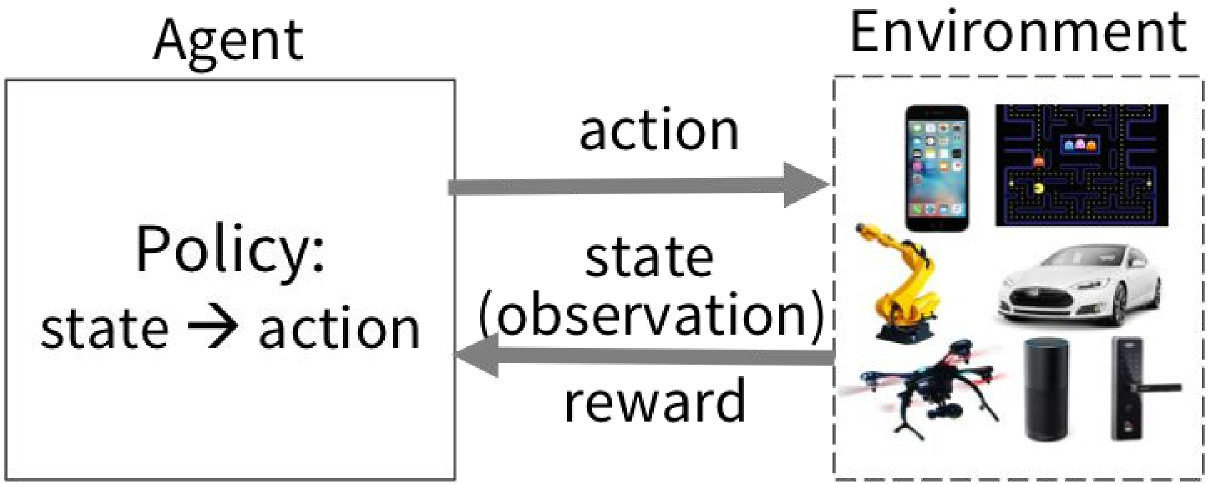


Emerging AI applications





Interacting with an environment



Things that are hard with current distributed systems

Things that are hard with current distributed systems

- Reinforcement learning training
- Fine-grained task parallelism with heterogeneous tasks
- Planning in real-time for a robot

Things that are hard with current distributed systems

- Reinforcement learning training
- Fine-grained task parallelism with heterogeneous tasks
- Planning in real-time for a robot

Requirements

- Low latency tasks
- High throughput tasks
- Adapt computation based on task progress
- Complex task dependencies
- Nested parallelism, dynamic task graph construction
- Tolerance of machine failures
- Seamless usage of GPUs and other accelerators

Example: RL training

```
def train(env, hyperparameters):
    policy = initial_policy()
    for _ in range(1000):
        trajectories = [rollout(policy, env) for _ in range(K)]
        policy.update(trajectories)
    return policy


def rollout(policy, env):
    # alternately evaluate policy and simulate env
```

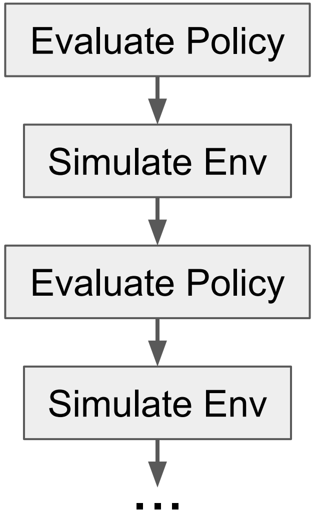
Example: RL training

```
def train(env, hyperparameters):
    policy = initial_policy()
    for _ in range(1000):
        trajectories = [rollout(policy, env) for _ in range(K)]
        policy.update(trajectories)
    return policy


def rollout(policy, env):
    # alternately evaluate policy and simulate env

while True:
    train(env, random_hyperparameters()):
```

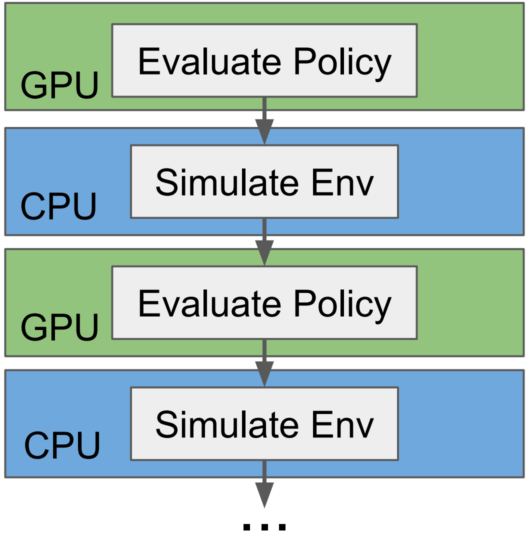
Example: RL training

```
@ray.remote
def train(env, hyperparameters):
    policy = initial_policy()
    for _ in range(1000):
        trajectories = ray.get([rollout.remote(policy, env) for _ in range(K)])
        policy.update(trajectories)
    return policy

@ray.remote
def rollout(policy, env):
    # alternately evaluate policy and simulate env

while True:
    train.remote(env, random_hyperparameters()):
```

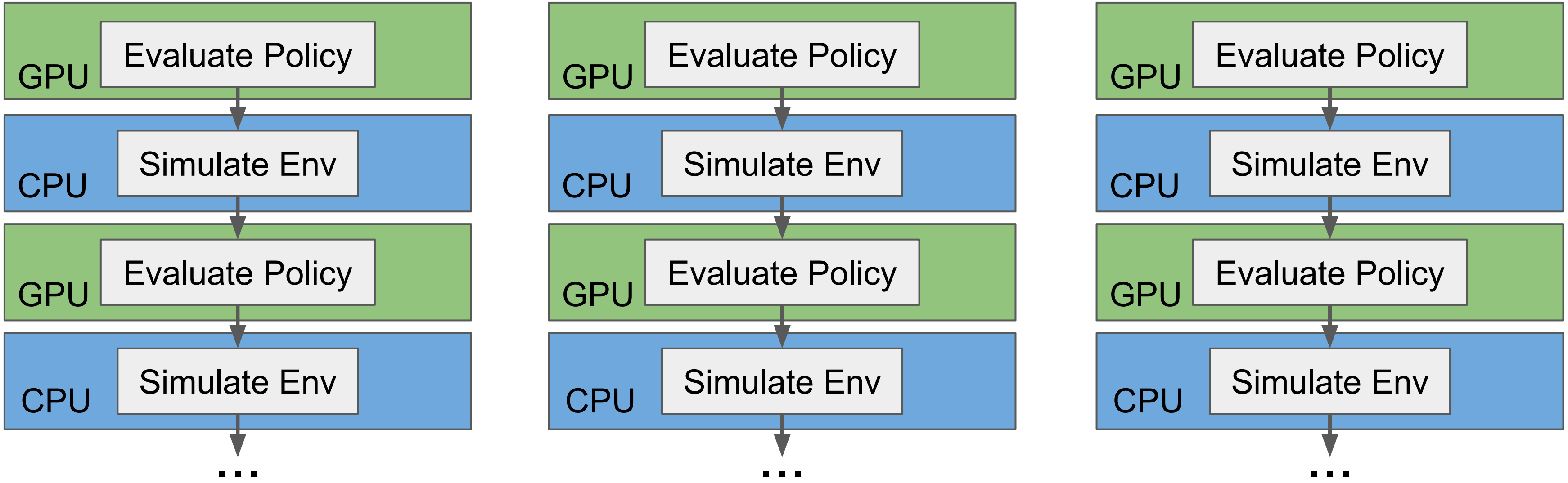
Example: simulators on CPUs, policies on GPUs



Example: simulators on CPUs, policies on GPUs



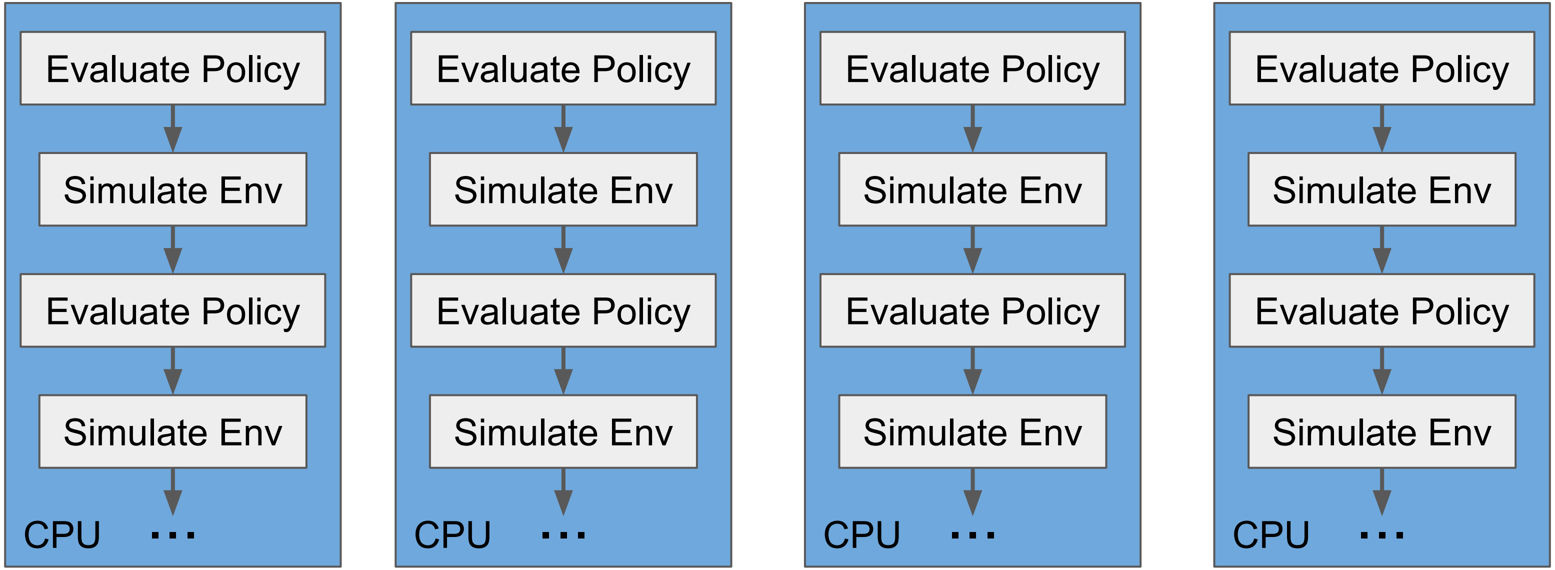
Example: simulators on CPUs, policies on GPUs



Example: simulators on CPUs, policies on GPUs

Options:

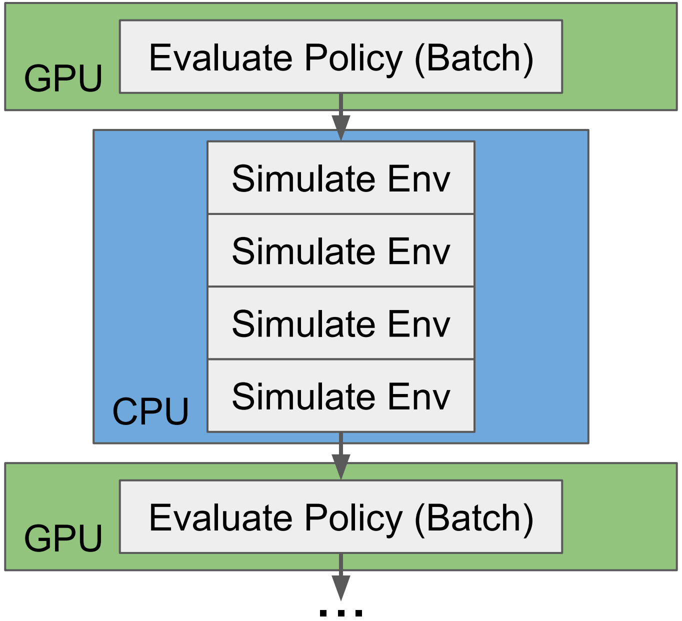
1) **One task per CPU, do everything on CPUs.**
2) One task per GPU (batch policy evaluation on GPU).
3) Many tasks. Policy evaluation on GPUs and simulator on CPUs.



Example: simulators on CPUs, policies on GPUs

Options:

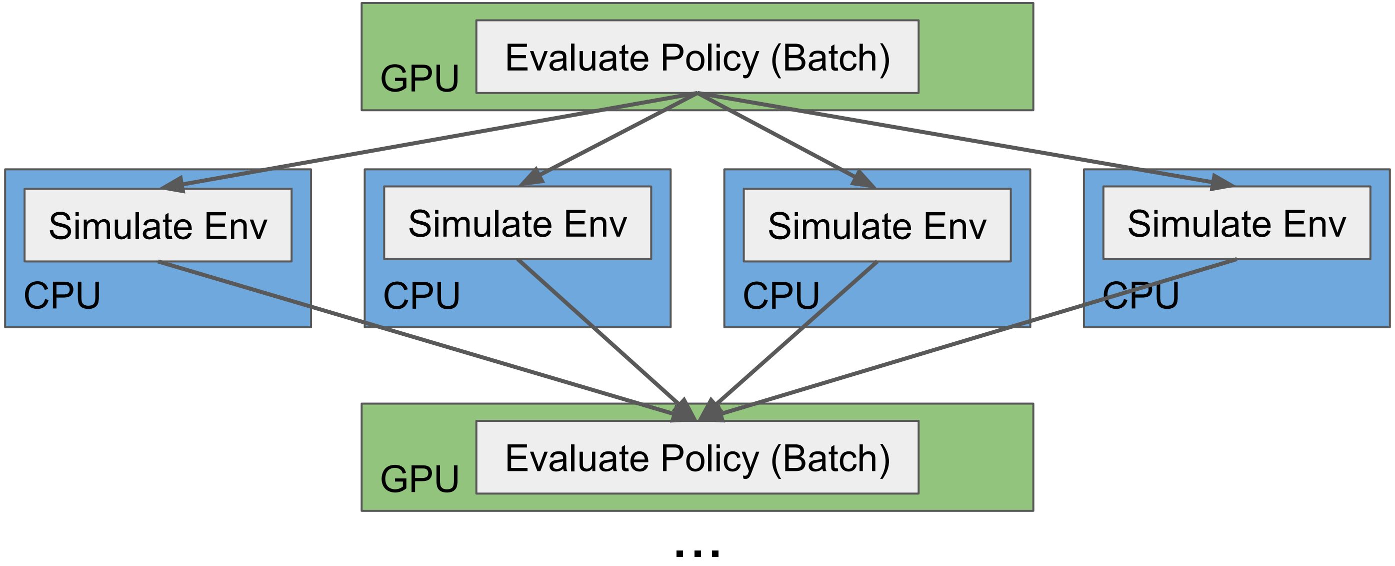
1) One task per CPU, do everything on CPUs.
2) **One task per GPU (batch policy evaluation on GPU).**
3) Many tasks. Policy evaluation on GPUs and simulator on CPUs.



Example: simulators on CPUs, policies on GPUs

Options:

1) One task per CPU, do everything on CPUs.
2) One task per GPU (batch policy evaluation on GPU).
3) **Many tasks. Policy evaluation on GPUs and simulator on CPUs.**



Ray API

Ray API - remote functions

```
def zeros(shape):
    return np.zeros(shape)


def dot(a, b):
  return np.dot(a, b)
```

Ray API - remote functions

```python
@ray.remote
def zeros(shape):
    return np.zeros(shape)

@ray.remote
def dot(a, b):
  return np.dot(a, b)
```

Ray API - remote functions

```python
@ray.remote
def zeros(shape):
    return np.zeros(shape)

@ray.remote
def dot(a, b):
  return np.dot(a, b)

id1 = zeros.remote([5, 5])
id2 = zeros.remote([5, 5])
id3 = dot.remote(id1, id2)
ray.get(id3)
```

Ray API - remote functions

```
@ray.remote
def zeros(shape):
    return np.zeros(shape)

@ray.remote
def dot(a, b):
  return np.dot(a, b)

id1 = zeros.remote([5, 5])
id2 = zeros.remote([5, 5])
id3 = dot.remote(id1, id2)
ray.get(id3)
```

- Blue variables are Object IDs.

Ray API - remote functions

```
@ray.remote
def zeros(shape):
    return np.zeros(shape)

@ray.remote(num_gpus=2)
def dot(a, b):
  return np.dot(a, b)

id1 = zeros.remote([5, 5])
id2 = zeros.remote([5, 5])
id3 = dot.remote(id1, id2)
ray.get(id3)
```

- **Blue** variables are Object IDs.
- Can specify **GPU** requirements

Ray API - actors

```python
class Counter(object):
    def __init__(self):
        self.value = 0
    def inc(self):
        self.value += 1
        return self.value

c = Counter()
c.inc()  # This returns 1
c.inc()  # This returns 2
c.inc()  # This returns 3
```

Ray API - actors

```python
@ray.actor
class Counter(object):
    def __init__(self):
        self.value = 0
    def inc(self):
        self.value += 1
        return self.value

c = Counter.actor()
id1 = c.inc()
id2 = c.inc()
id3 = c.inc()
ray.get([id1, id2, id3])  # This returns [1, 2, 3]
```

Ray API - actors

```python
@ray.actor
class Counter(object):
    def __init__(self):
        self.value = 0
    def inc(self):
        self.value += 1
        return self.value

c = Counter.actor()
id1 = c.inc()
id2 = c.inc()
id3 = c.inc()
ray.get([id1, id2, id3])  # This returns [1, 2, 3]
```

- State is shared between actor methods.
- Actor methods return **Object IDs**.

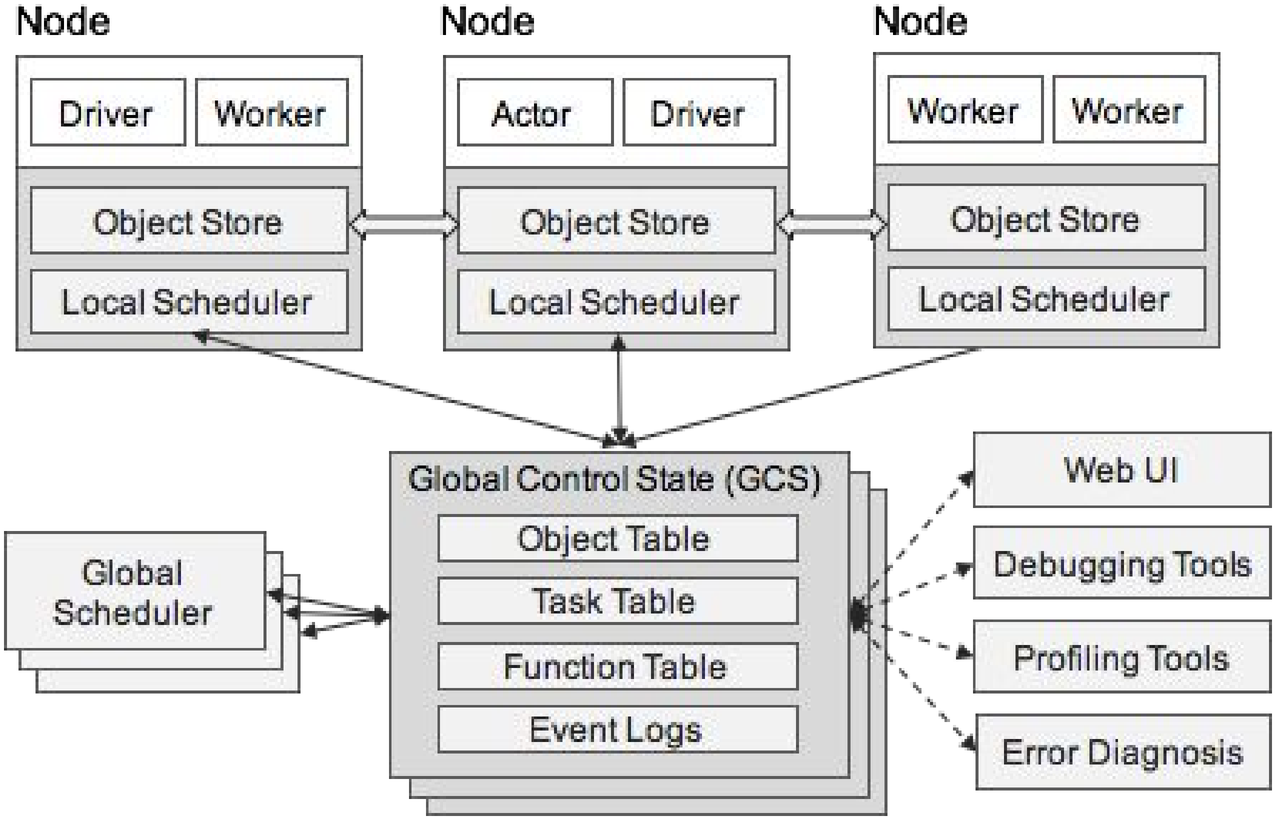
Ray API - actors

```python
@ray.actor(num_gpus=1)
class Counter(object):
    def __init__(self):
        self.value = 0
    def inc(self):
        self.value += 1
        return self.value

c = Counter.actor()
id1 = c.inc()
id2 = c.inc()
id3 = c.inc()
ray.get([id1, id2, id3])  # This returns [1, 2, 3]
```

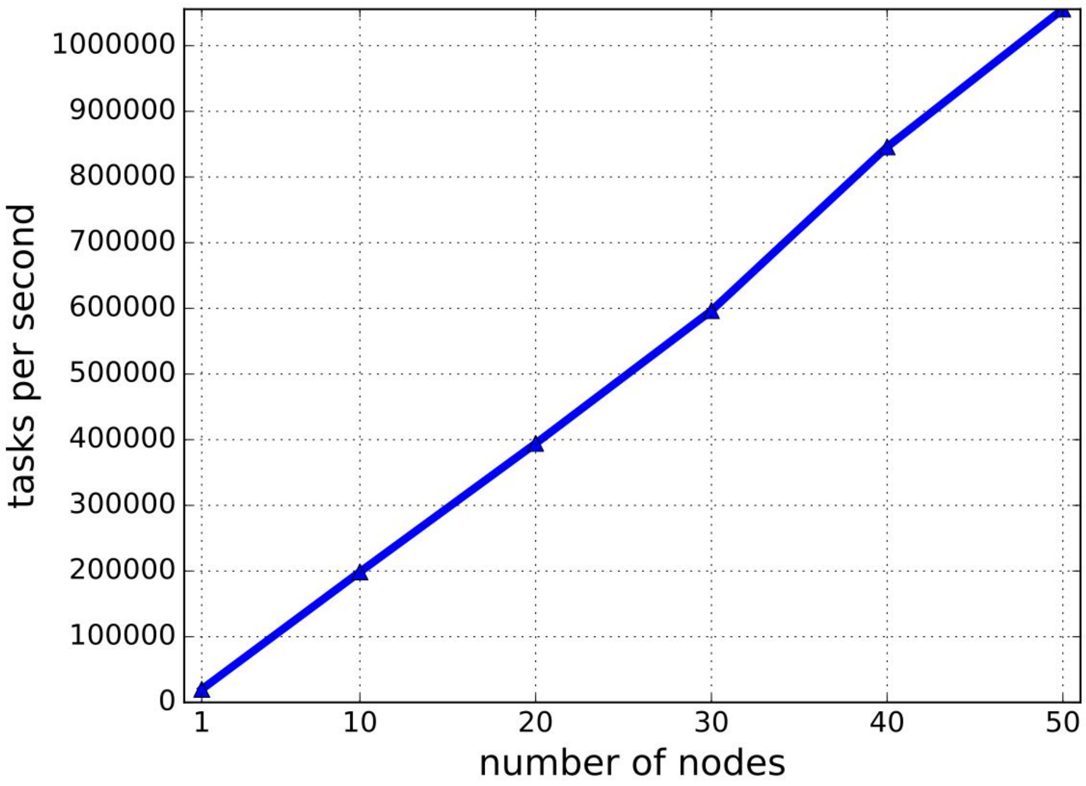
- State is shared between actor methods.
- Actor methods return **Object IDs**.
- Can specify **GPU** requirements

Ray Architecture

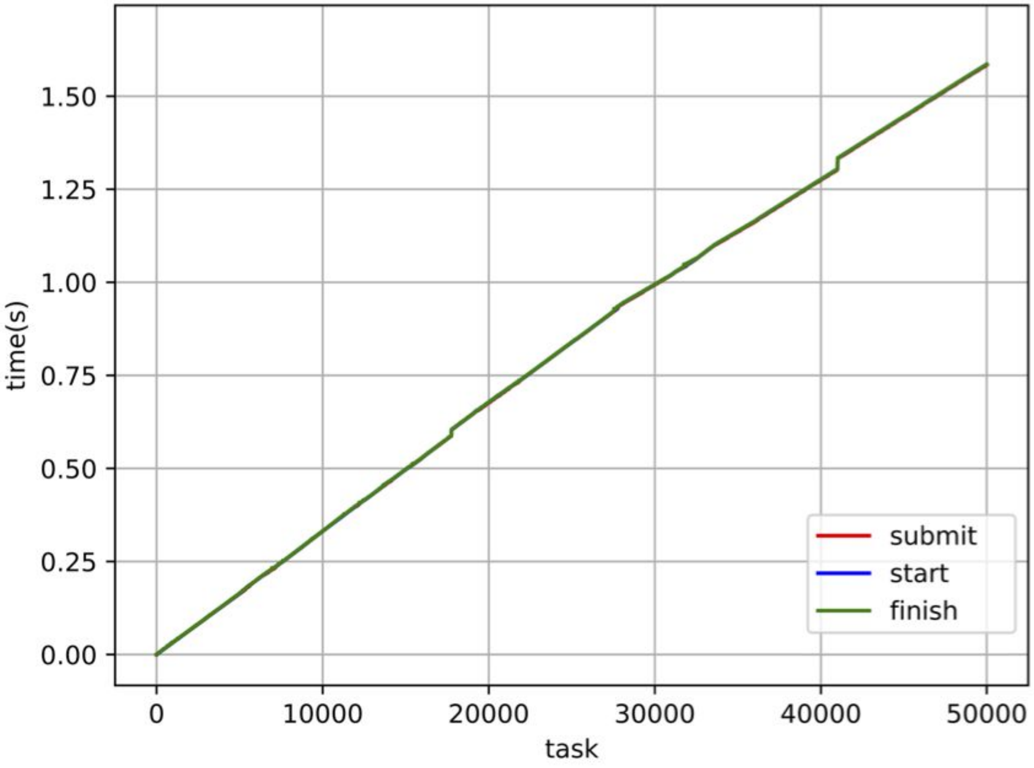
System Architecture



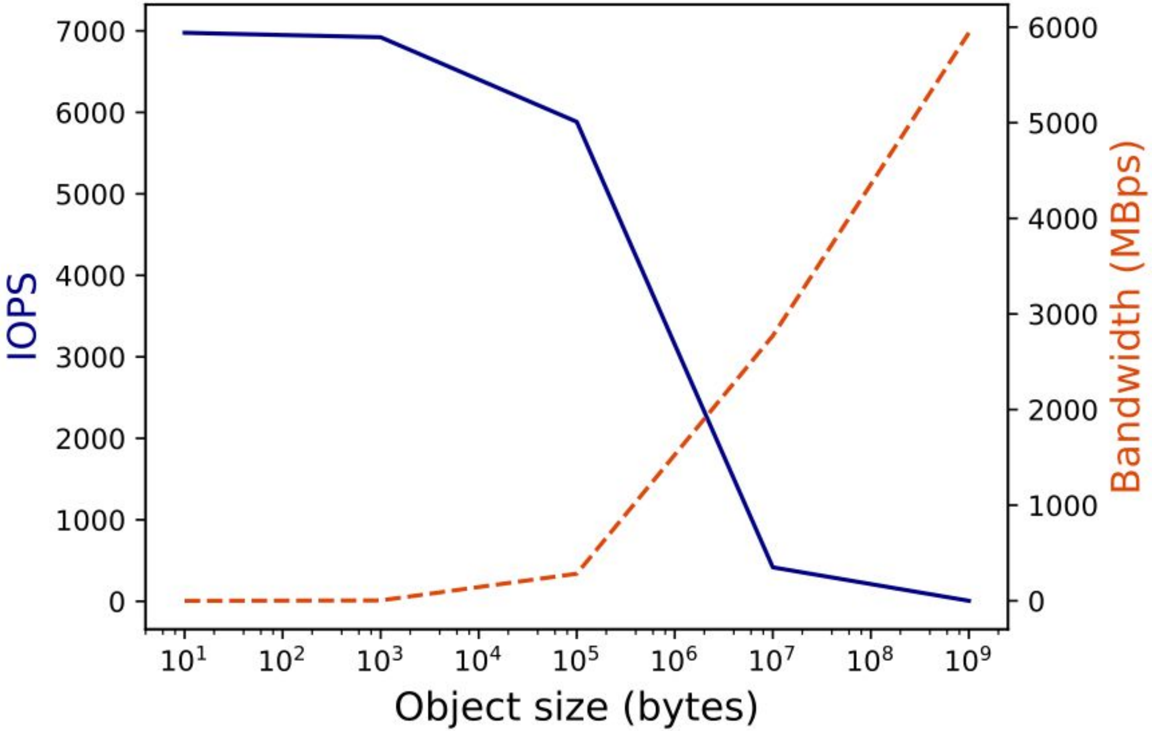
System throughput



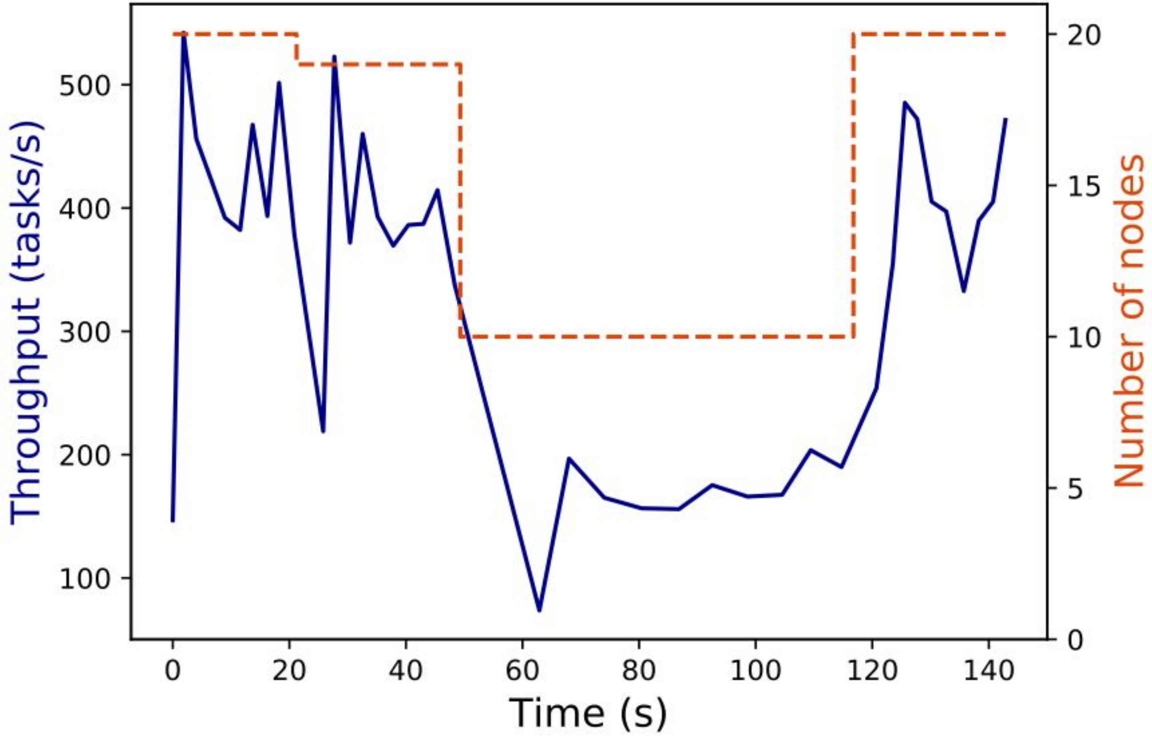
Single machine throughput



Object store performance

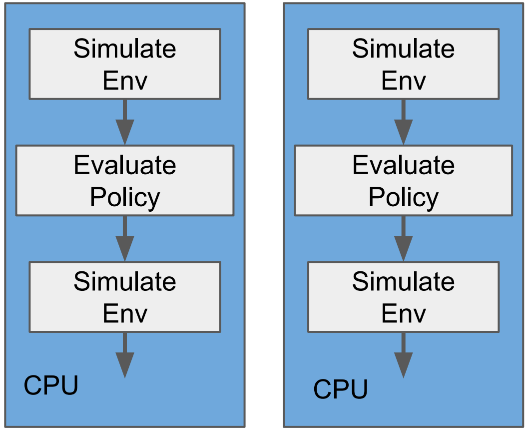


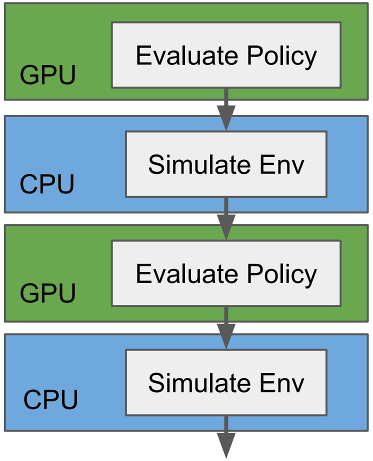
Robustness to node failure

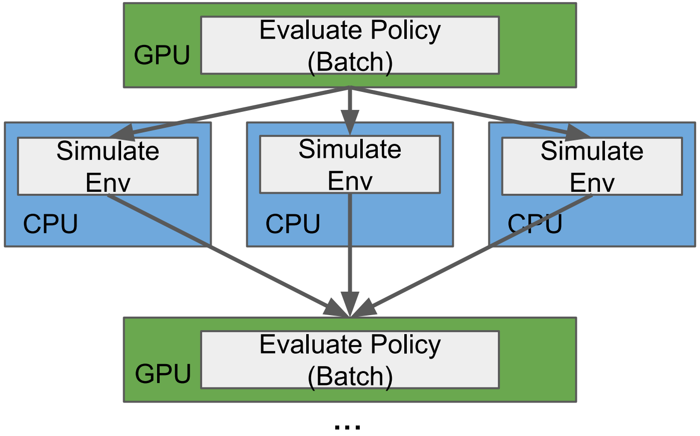


Experiments

Speeding up rollouts for policy gradients







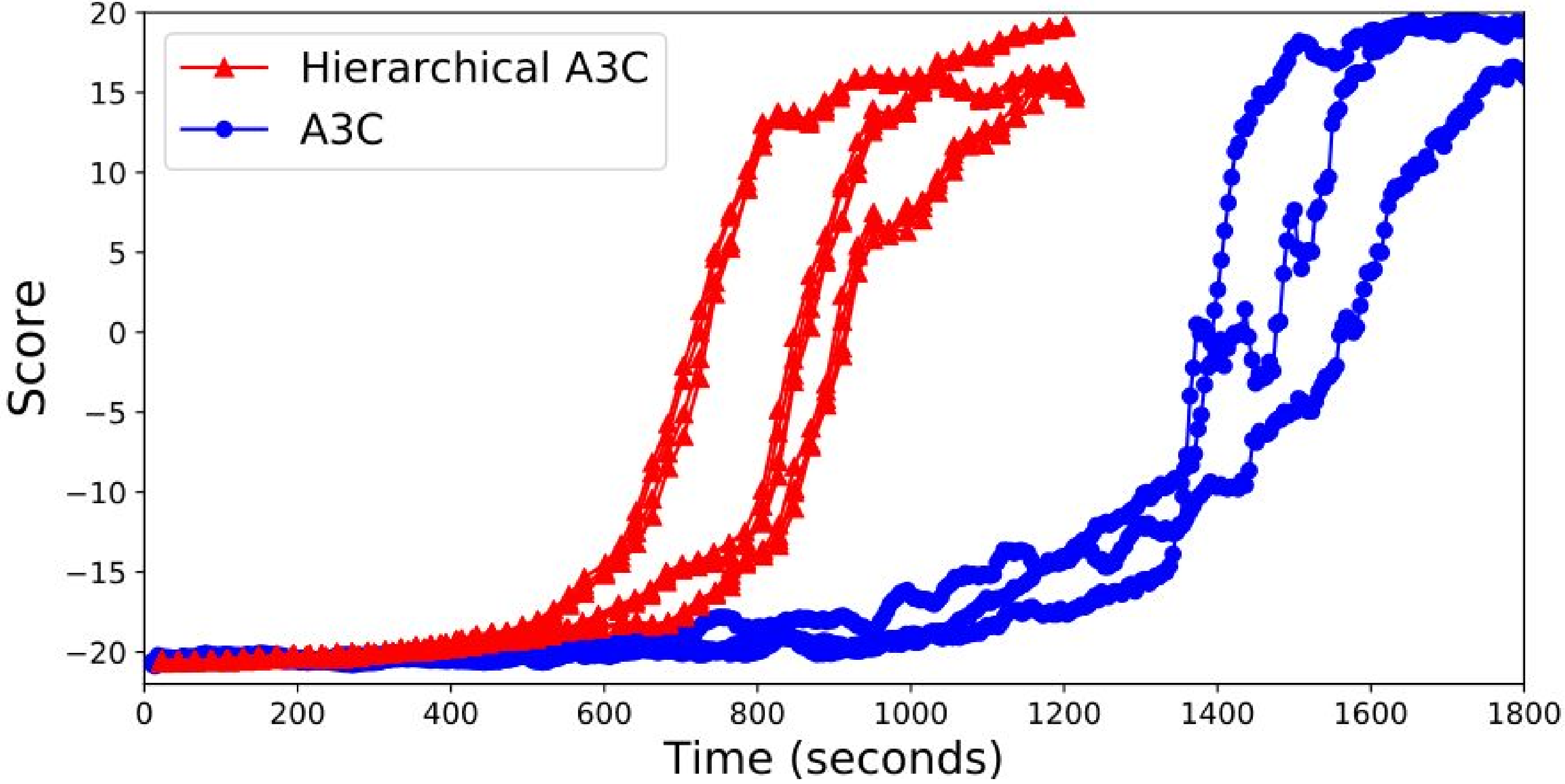
	Parallel Rollouts on CPU	**Policy Evaluation on GPU**	**Fine grained rollouts**
Speedup:	**1.0x**	**1.3x**	**4.1x**

Evolution strategies

	10 nodes	20 nodes	30 nodes	40 nodes	50 nodes
Reference	97K	215K	202K	N/A	N/A
Ray	152K	285K	323K	476K	571K

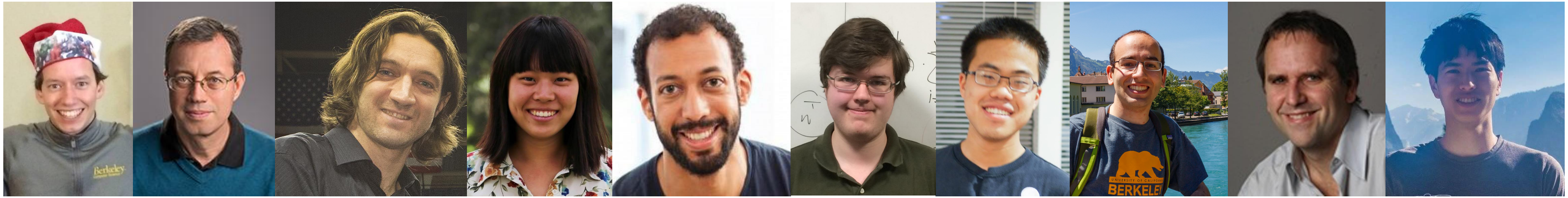
The Ray implementation takes half the amount of code and was implemented in a couple of hours

Hierarchical A3C



Ray is a system for AI Applications

- Ray is open source! https://github.com/ray-project/ray
- We have a pre-release!
- We'd love your feedback.



Philipp Ion Alexey Stephanie Johann William Richard Mehrdad Mike Robert