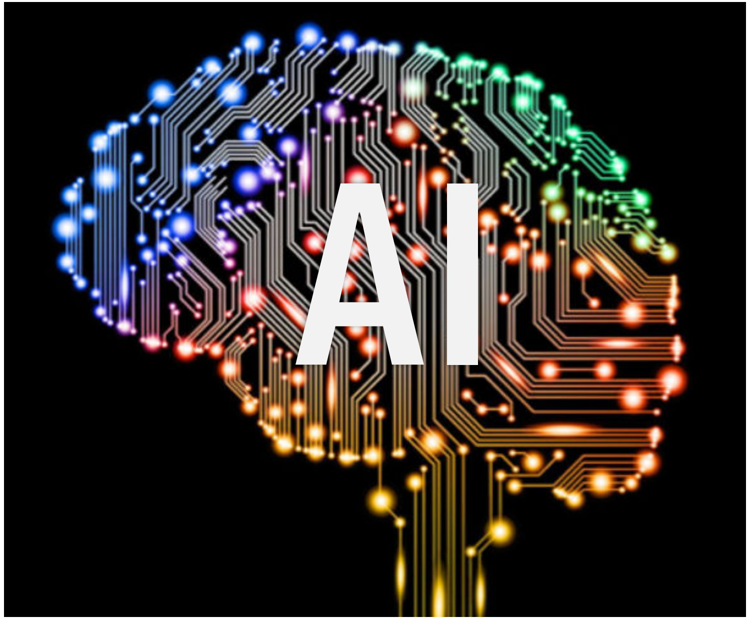
RL Systems @ RISELab

Ion Stoica

UC Berkeley and Databricks

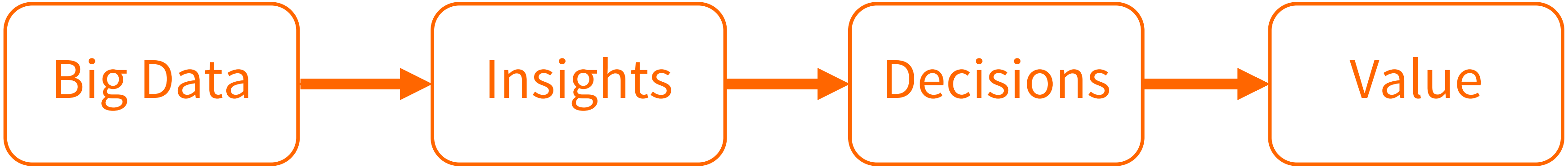
March 24, 2018

The two major trends of the past decade



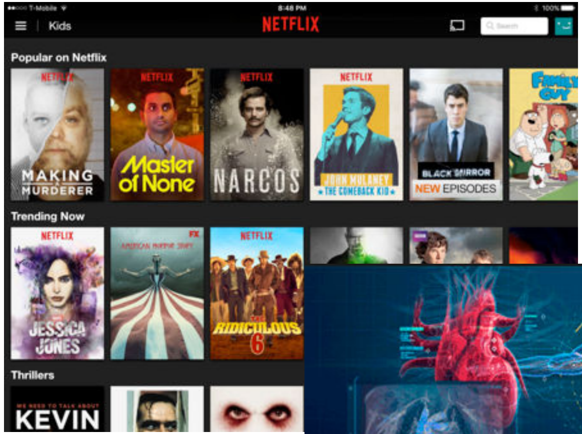
Harnessing data "revolution"

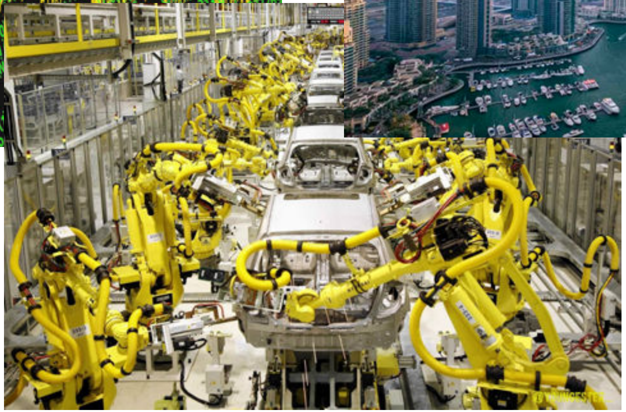
Harnessing data "revolution"



"Data is only as valuable as the decisions it enables"

Complex decisions powered by AI





Challenge

mission-critical apps in adversarial, continually changing environments

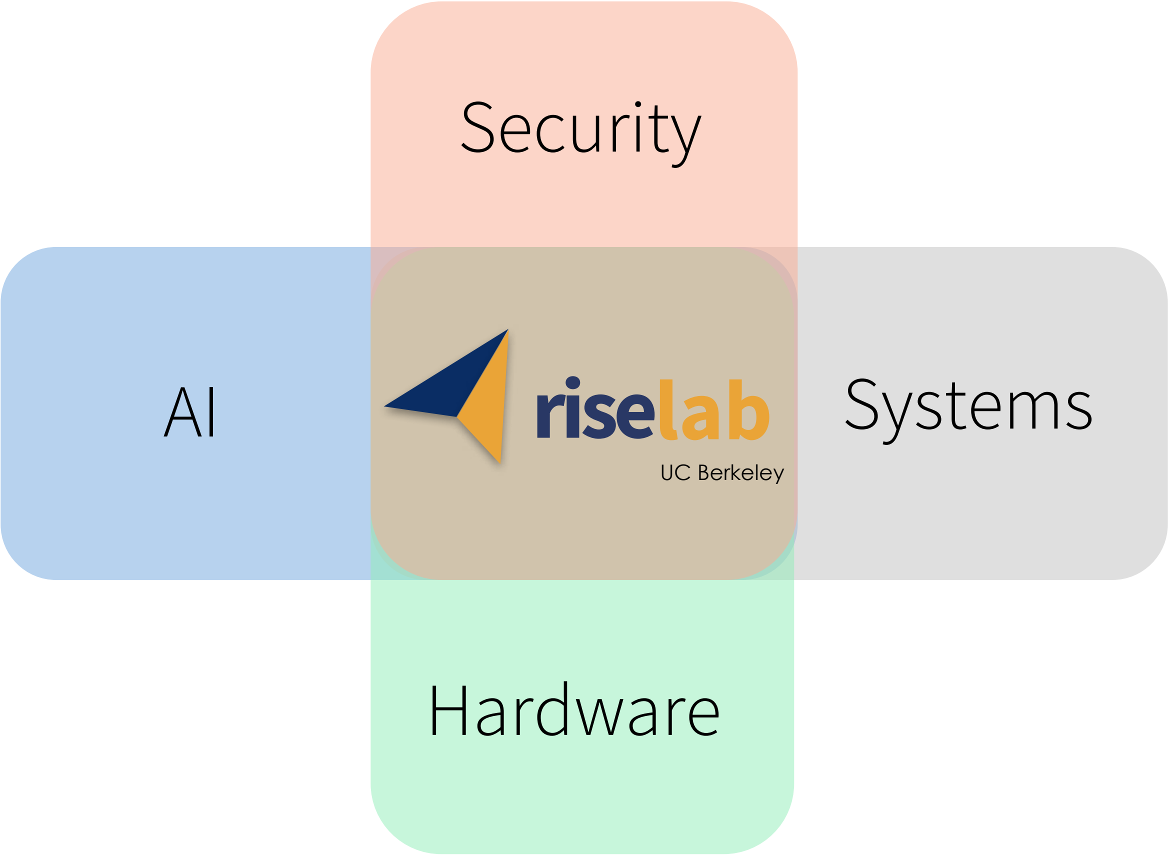
RISELab Goal

Develop open source platforms, tools, and algorithms for **r**eal-time **i**ntelligent decisions on live-data which are **s**ecure and **e**xplainable

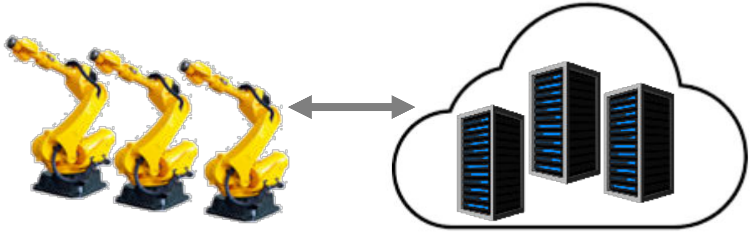
RISELab Goal

Develop open source platforms, tools, and algorithms for **r**eal-time **i**ntelligent decisions on live-data which are **s**ecure and **e**xplainable

UC Berkeley


Security
AI
riselab
UC Berkeley
Systems
Hardware

Example: Robotics



AI

- Reinforcement learning (RL)
- Control hierarchies

Security

- Shared learning
- Adversarial learning

Systems

- Systems for RL
- Cloud-edge systems

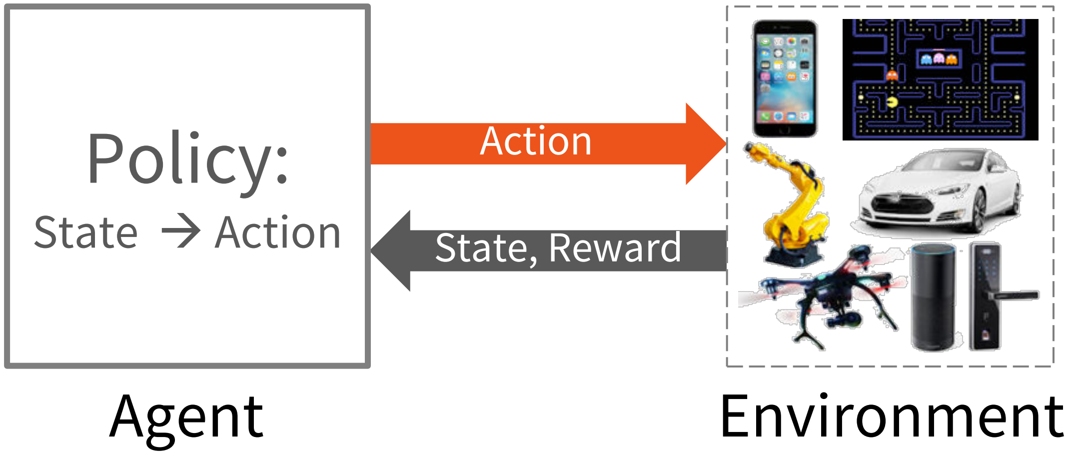
Hardware

- AI accelerators
- Hardware enclaves

Reinforcement Learning (RL)

Agent continually learning by interacting with environment

Compute **policy** (i.e., **state → action**) to maximize **reward**



RL significant benefit from scale

Published as a conference paper at ICLR 2017

REINFORCEMENT LEARNING THROUGH ASYNCHRONOUS ADVANTAGE ACTOR-CRITIC ON A GPU

Mohammad Babaeizadeh
Department of Computer Science
University of Illinois at Urbana-Champaign, USA
mb2@uiuc.edu

Iuri Frosio, Stephen Tyree, Jason Clemons, Jan Kautz
NVIDIA, USA

A Distributed PPO

A.1 Algorithm details

Pseudocode for the Distributed PPO algorithm is provided in Algorithm Boxes 2 and 3. W is the number of workers; D sets a threshold for the number of workers whose gradients must be available to update the parameters. M, B is the number of sub-iterations with policy and baseline updates given a batch of datapoints. T is the number of data points collected per worker before parameter updates are computed. K is the number of time steps for computing K-step returns and truncated backprop through time (for RNNs)

Algorithm 2 Distributed Proximal Policy Optimization (chief)

EFFICIENT PARALLEL METHODS FOR DEEP REINFORCEMENT LEARNING

Alfredo V. Clemente
Department of Computer and Information Science
Norwegian University of Science and Technology
Trondheim, Norway
alfredvc@stud.ntnu.no

Arjun Chandra
Telenor Research
Trondheim, Norway
arjun.chandra@telenor.com

Humberto N. Castejón
Telenor Research
Trondheim, Norway
humberto.castejon@telenor.com

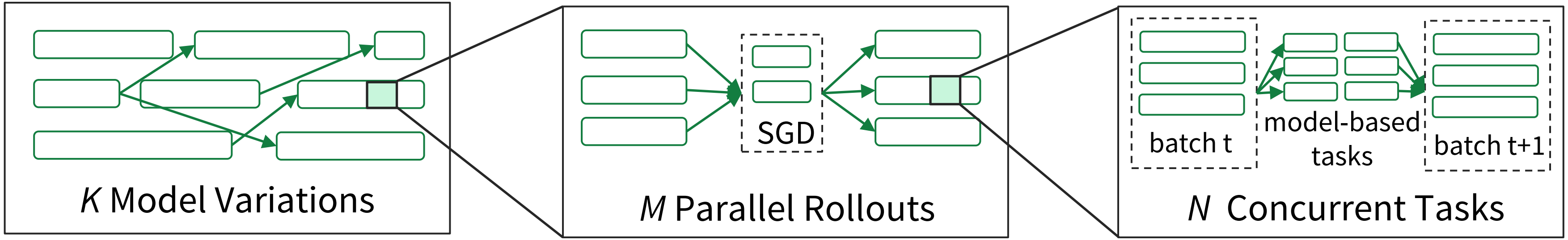
DISTRIBUTED PRIORITIZED EXPERIENCE REPLAY

ABSTRACT

ropose a distributed architecture for deep reinforcement learning at scale, enables agents to learn effectively from orders of magnitude more data than viously possible. The algorithm decouples acting from learning: the actors eract with their own instances of the environment by selecting actions according a shared neural network, and accumulate the resulting experience in a shared

RL systems requirements

Nested parallelism



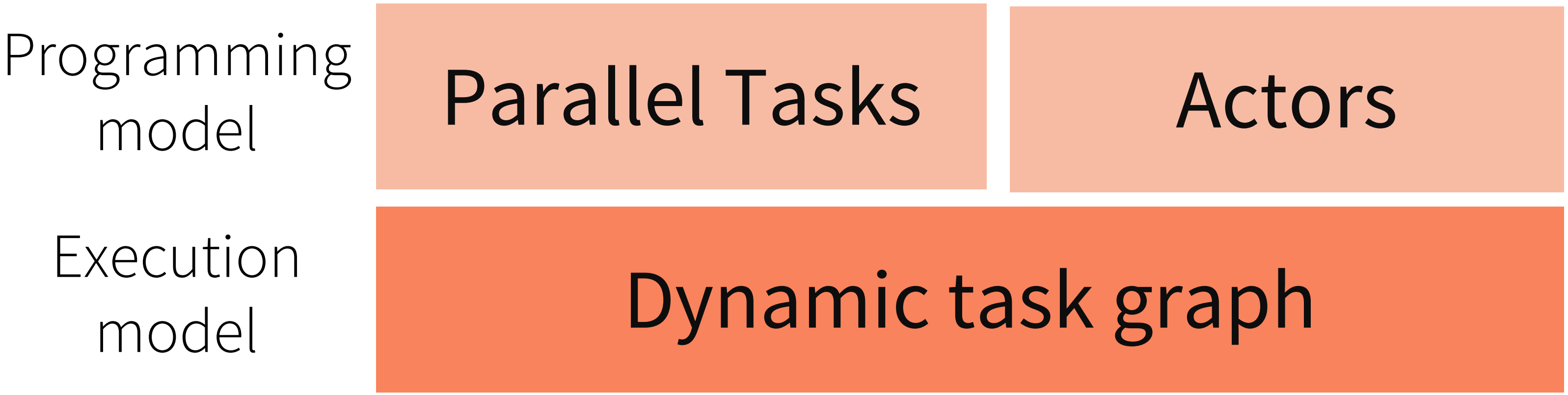
Heterogeneity

- Different task durations
- Different resource requirements (e.g., GPUs, TPUs, CPUs)

Real-time decisions

Ray: A system for distributed AI

Architecture



Parallel Tasks

```python
def read_array(file):
    # read ndarray “a” from “file”
    return a


def add(a, b):
    return np.add(a, b)

a = read_array(file1)
b = read_array(file2)
sum = add(a, b)
```

Parallel Tasks

```
@ray.remote
def read_array(file):
    # read ndarray “a” from “file”
    return a

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

a = read_array(file1)
b = read_array(file2)
sum = add(a, b)
```

Parallel Tasks

```
@ray.remote
def read_array(file):
    # read ndarray “a” from “file”
    return a

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

id1 = read_array.remote(file1)
id2 = read_array.remote(file2)
id = add.remote(id1, id2)
sum = ray.get(id)
```

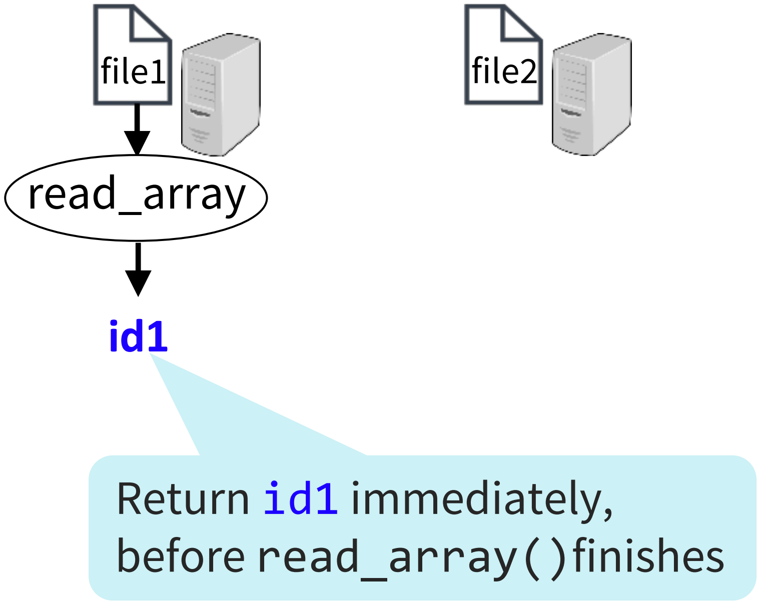
- Blue variables are **Object IDs**
- Similar to futures

Parallel Tasks

```
@ray.remote
def read_array(file):
    # read ndarray “a” from “file”
    return a

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

id1 = read_array.remote(file1)
id2 = read_array.remote(file2)
id = add.remote(id1, id2)
sum = ray.get(id)
```

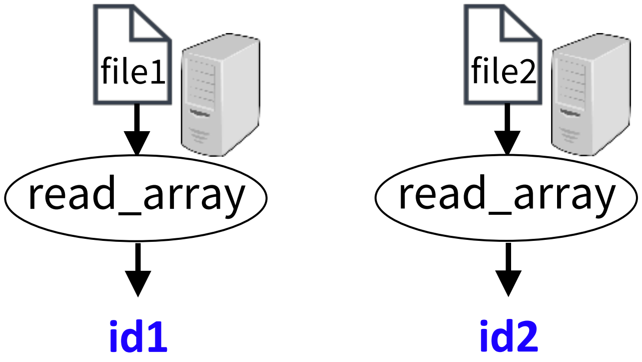
Parallel Tasks

```
@ray.remote
def read_array(file):
    # read ndarray “a” from “file”
    return a

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

id1 = read_array.remote(file1)
id2 = read_array.remote(file2)
id = add.remote(id1, id2)
sum = ray.get(id)
```



Parallel Tasks

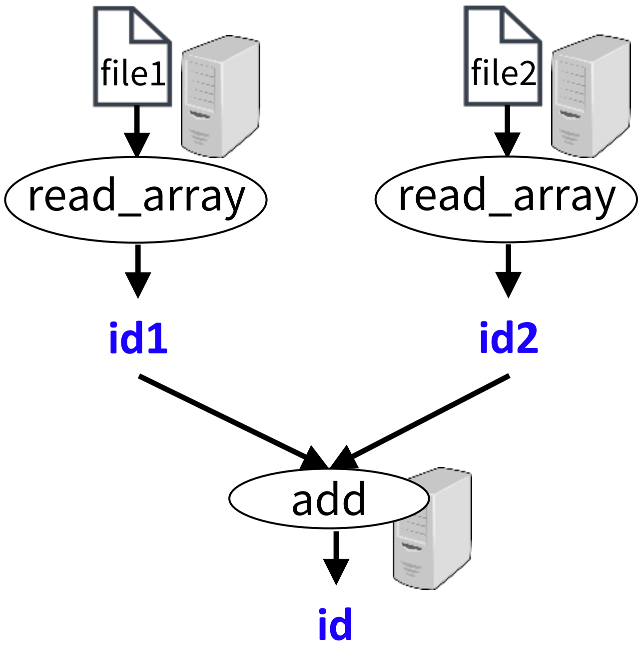
```
@ray.remote
def read_array(file):
    # read ndarray “a” from “file”
    return a

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

id1 = read_array.remote(file1)
id2 = read_array.remote(file2)
id = add.remote(id1, id2)
sum = ray.get(id)
```



Dynamic task graph:
build at runtime

Parallel Tasks

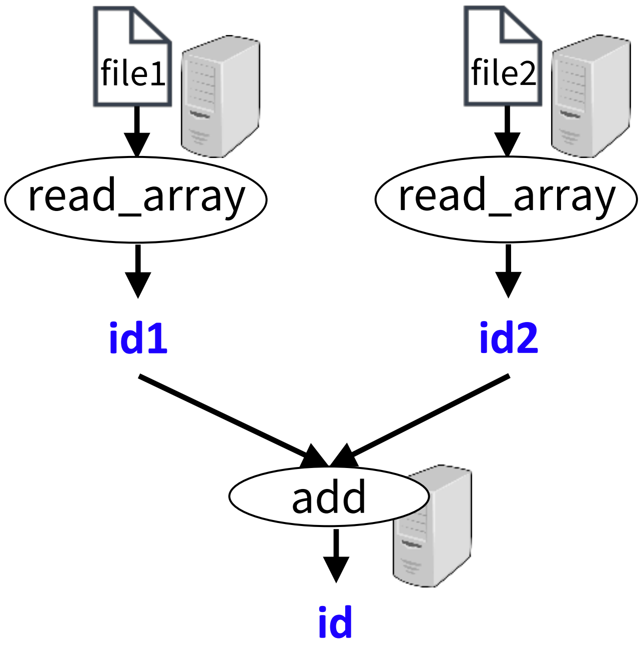
```
@ray.remote
def read_array(file):
    # read ndarray “a” from “file”
    return a

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

id1 = read_array.remote(file1)
id2 = read_array.remote(file2)
id = add.remote(id1, id2)
sum = ray.get(id)
```



Every task scheduled, but not finished yet

Parallel Tasks

```
@ray.remote
def read_array(file):
    # read ndarray "a" from "file"
    return a

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

id1 = read_array.remote(file1)
id2 = read_array.remote(file2)
id = add.remote(id1, id2)
sum = ray.get(id)
```



ray.get() block until result available

Tasks, not enough!

Might not have access to simulator state, can't do

```
state = simulator.initialize()
action = policy.compute(state)
```

Some state expensive to create (e.g., DNN on GPUs)

- Better to create it once and then reinitialize for each task

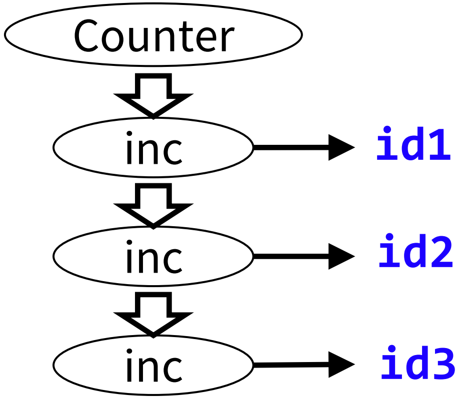
Actors

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

c = Counter()
c.inc()
c.inc()
c.inc()
```

Actors

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

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

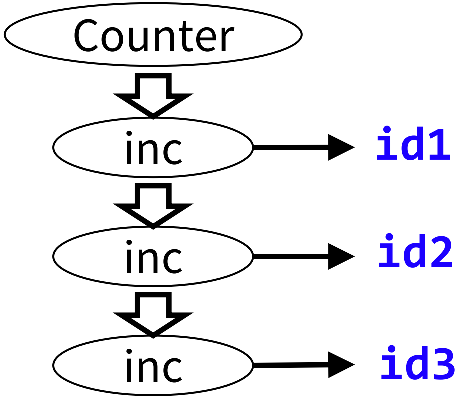
- State shared across actor's methods
- Actor methods return **object IDs**



Actors

```
@ray.remote(num_gpus = 4)
class Counter(object):
    def __init__(self):
        self.value = 0
    def inc(self):
        self.value += 1
        return self.value

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

- State shared across actor's methods
- Actor methods return **object IDs**
- Can specify # of GPUs



Example

Evolution Strategies as a Scalable Alternative to Reinforcement Learning

Tim Salimans [1] Jonathan Ho [1] Xi Chen [1] Ilya Sutskever [1]

Abstract

We explore the use of Evolution Strategies, a class of black box optimization algorithms, as an alternative to popular RL techniques such as Q-learning and Policy Gradients. Experiments on MuJoCo and Atari show that ES is a viable solution strategy that scales extremely well with the number of CPUs available: By using hundreds to thousands of parallel workers, ES can solve 3D humanoid walking in 10 minutes and obtain competitive results on most Atari games after one hour of training time. In addition, we highlight several advantages of ES as a black box

Try lots of different policies and see which work best!

1. Introduction

Developing agents that can accomplish challenging tasks

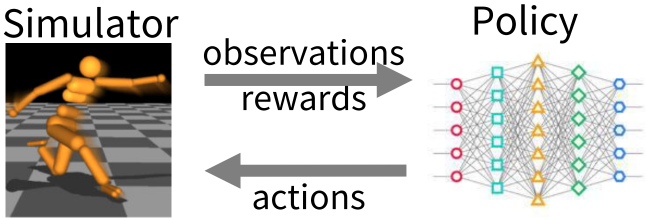
In this paper, we investigate the effectiveness of evolution strategies in the context of controlling robots in the MuJoCo physics simulator (Todorov et al., 2012) and playing Atari games with pixel inputs (Mnih et al., 2015). Our key findings are as follows:

1. We found specific network parameterizations that cause evolution strategies to reliably succeed, which we elaborate on in section 2.2.

2. We found the evolution strategies method to be highly parallelizable: we observe linear speedups in run time even when using over a thousand workers. In partic-

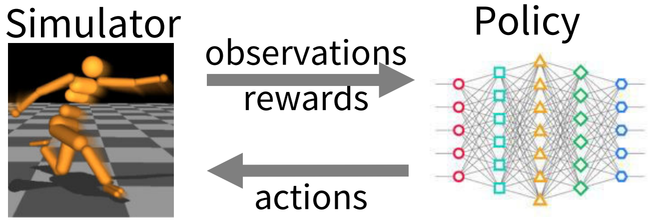
ing between 3x and 10x as much data. The slight decrease in data efficiency is partly offset by a reduction

Pseudocode




```
class Worker(object):
  def do_simulation(policy, seed):
    # perform simulation and return reward
```

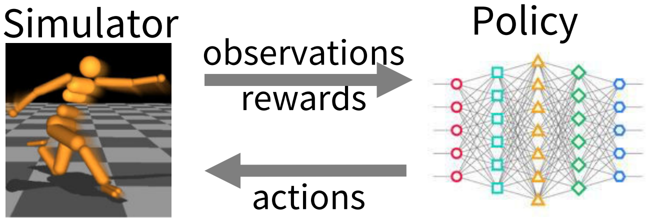
Pseudocode




```
class Worker(object):
  def do_simulation(policy, seed):
    # perform simulation and return reward
```

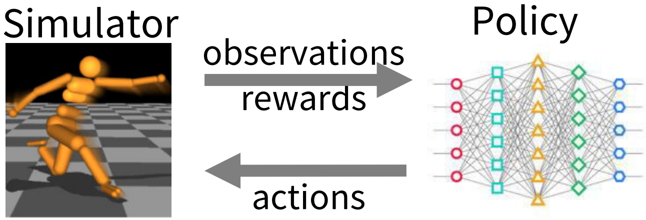
```
workers = [Worker() for i in range(20)]
policy = initial_policy()
```

Pseudocode



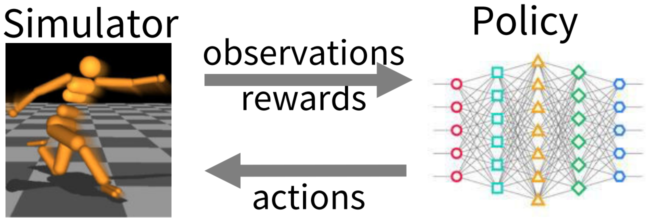

```
class Worker(object):
  def do_simulation(policy, seed):
    # perform simulation and return reward

workers = [Worker() for i in range(20)]
policy = initial_policy()

for i in range(200):
  seeds = generate_seeds(i)
  rewards = [workers[j].do_simulation(policy, seeds[j])
             for j in range(20)]
  policy = compute_update(policy, rewards, seeds)
```

Pseudocode



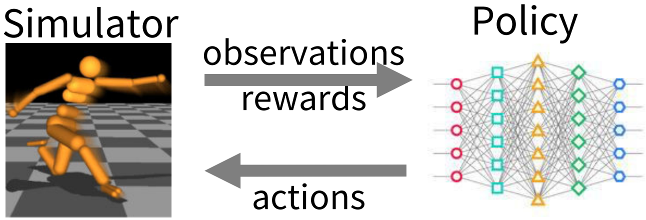

```
@ray.remote
class Worker(object):
  def do_simulation(policy, seed):
    # perform simulation and return reward

workers = [Worker() for i in range(20)]
policy = initial_policy()

for i in range(200):
  seeds = generate_seeds(i)
  rewards = [workers[j].do_simulation(policy, seeds[j])
              for j in range(20)]
  policy = compute_update(policy, rewards, seeds)
```

Pseudocode



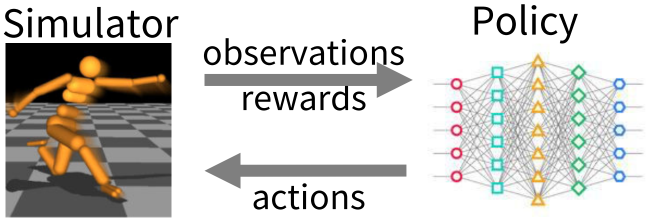

```
@ray.remote
class Worker(object):
  def do_simulation(policy, seed):
    # perform simulation and return reward

workers = [Worker.remote() for i in range(20)]
policy = initial_policy()

for i in range(200):
  seeds = generate_seeds(i)
  rewards = [workers[j].do_simulation(policy, seeds[j])
             for j in range(20)]
  policy = compute_update(policy, rewards, seeds)
```

Pseudocode




```
@ray.remote
class Worker(object):
  def do_simulation(policy, seed):
    # perform simulation and return reward

workers = [Worker.remote() for i in range(20)]
policy = initial_policy()

for i in range(200):
  seeds = generate_seeds(i)
  rewards = [workers[j].do_simulation.remote(policy, seeds[j])
             for j in range(20)]
  policy = compute_update(policy, rewards, seeds)
```

Pseudocode

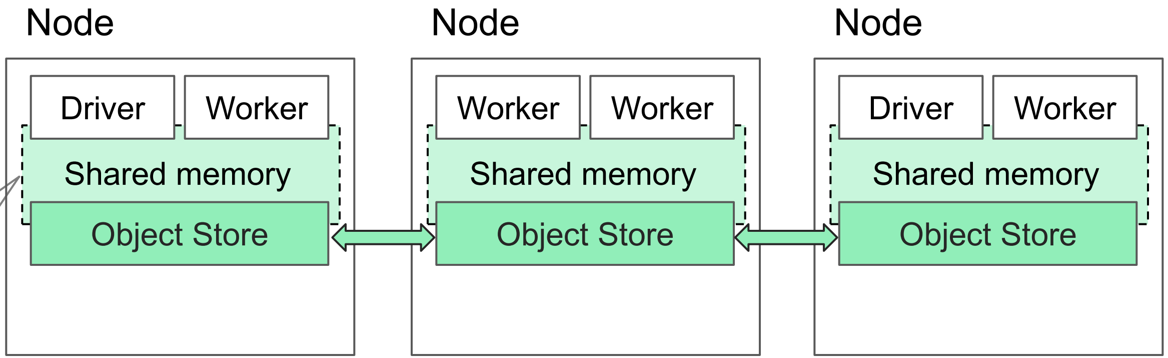



```
@ray.remote
class Worker(object):
  def do_simulation(policy, seed):
    # perform simulation and return reward

workers = [Worker.remote() for i in range(20)]
policy = initial_policy()

for i in range(200):
  seeds = generate_seeds(i)
  rewards = [workers[j].do_simulation.remote(policy, seeds[j])
             for j in range(20)]
  policy = compute_update(policy, ray.get(rewards), seeds)
```

Ray Architecture

In-memory object store

- Immutable objects



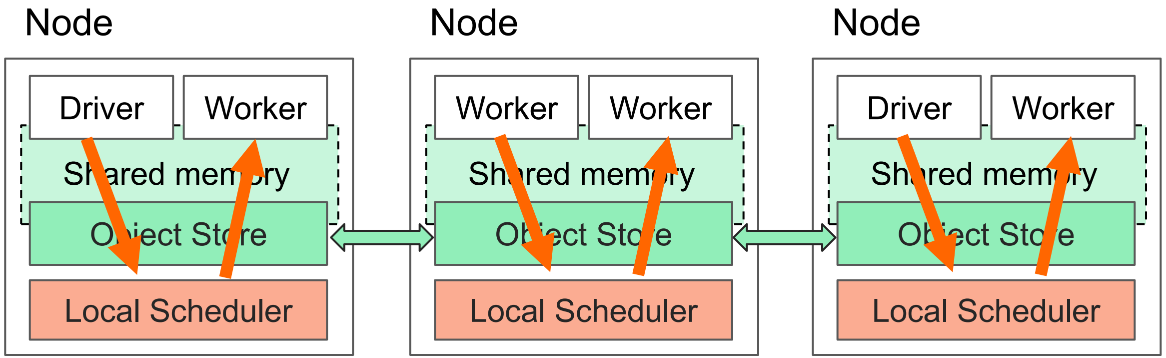
Serialization using Apache Arrow

Ray Architecture

In-memory object store

- Immutable objects

Distributed scheduler

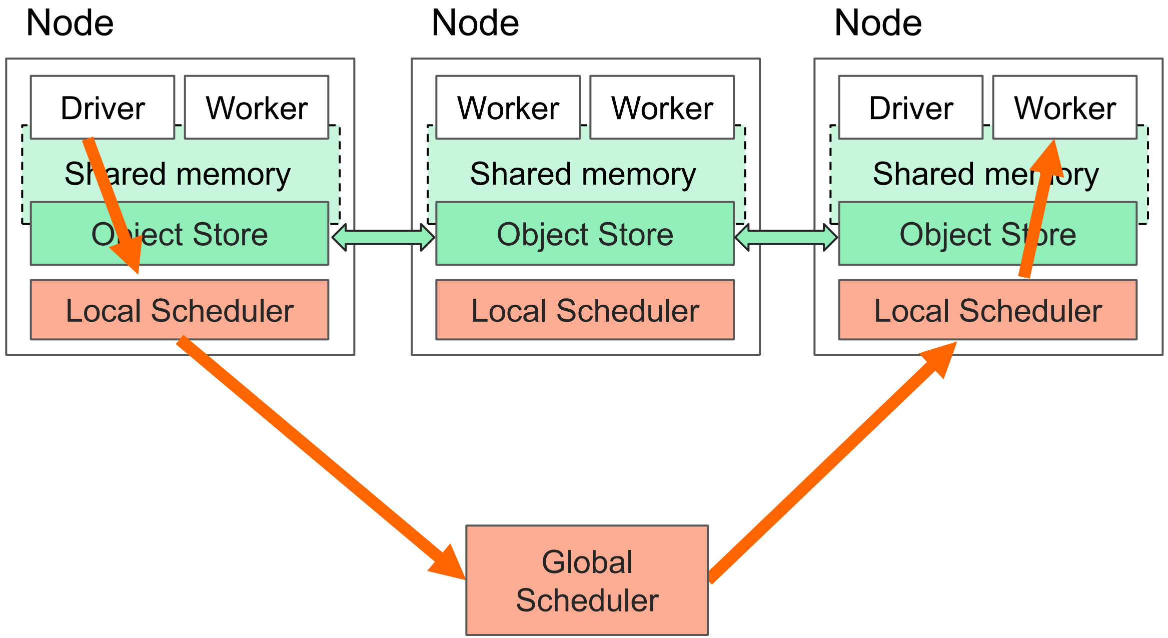


Ray Architecture

In-memory object store

- Immutable objects

Distributed scheduler



Ray Architecture

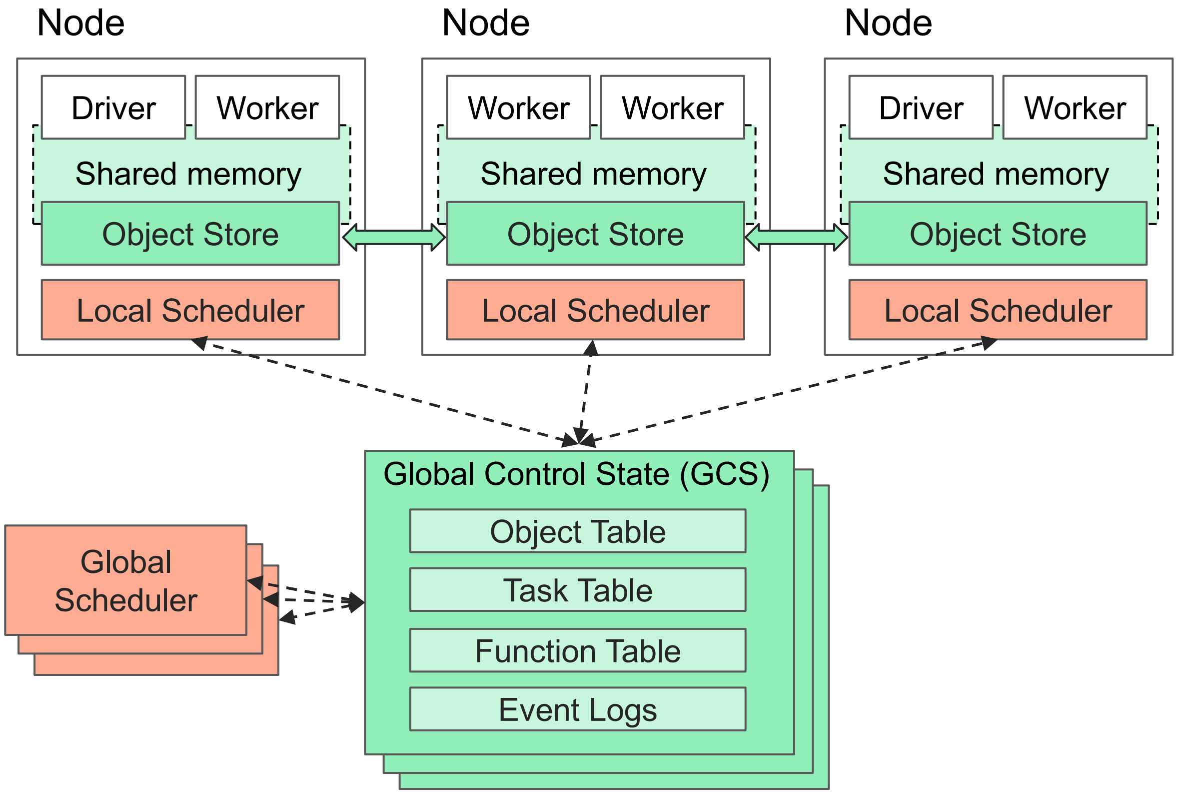
In-memory object store

- Immutable objects

Distributed scheduler

Centralized control store

- Stateless components



Ray Architecture

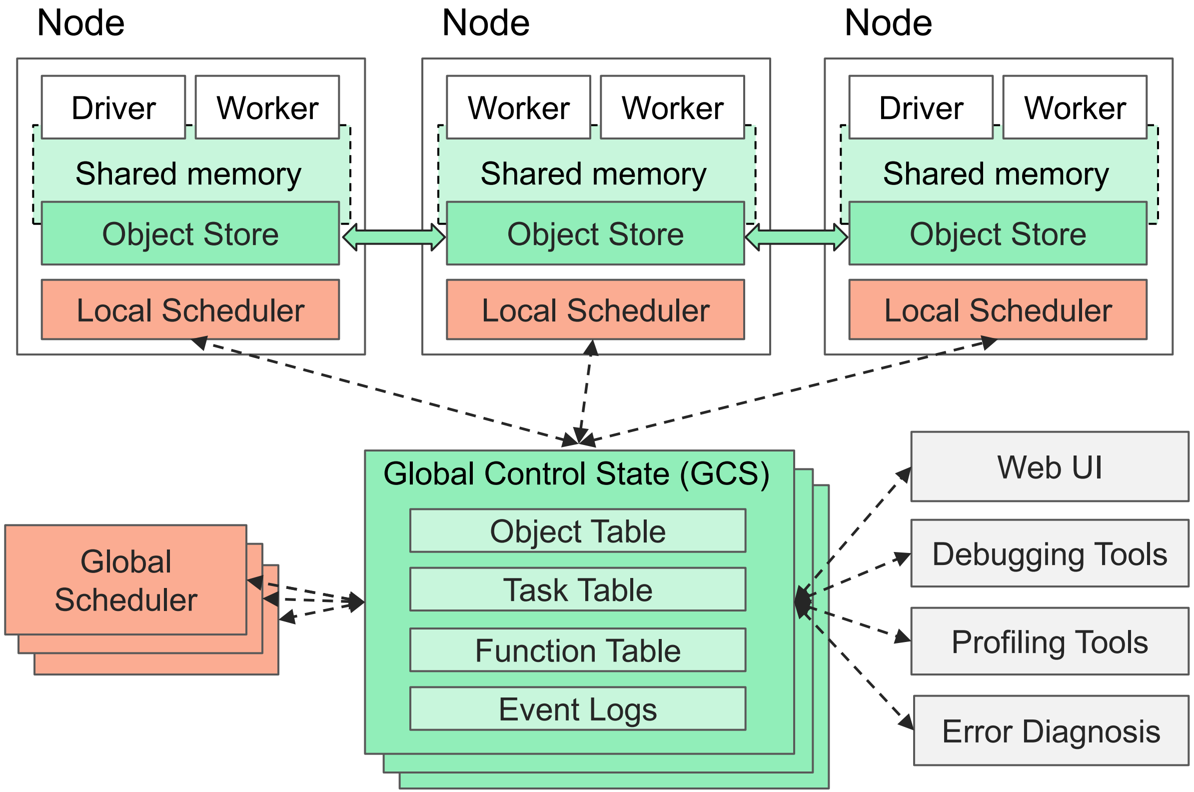
In-memory object store

- Immutable objects

Distributed scheduler

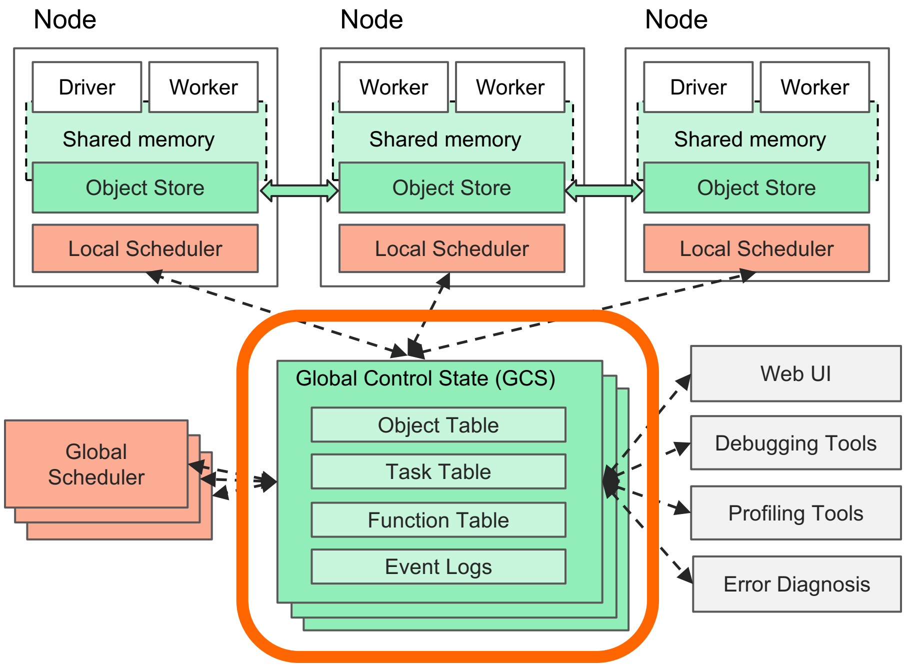
Centralized control store

- Stateless components



Highly Scalable

GCS: sharded database

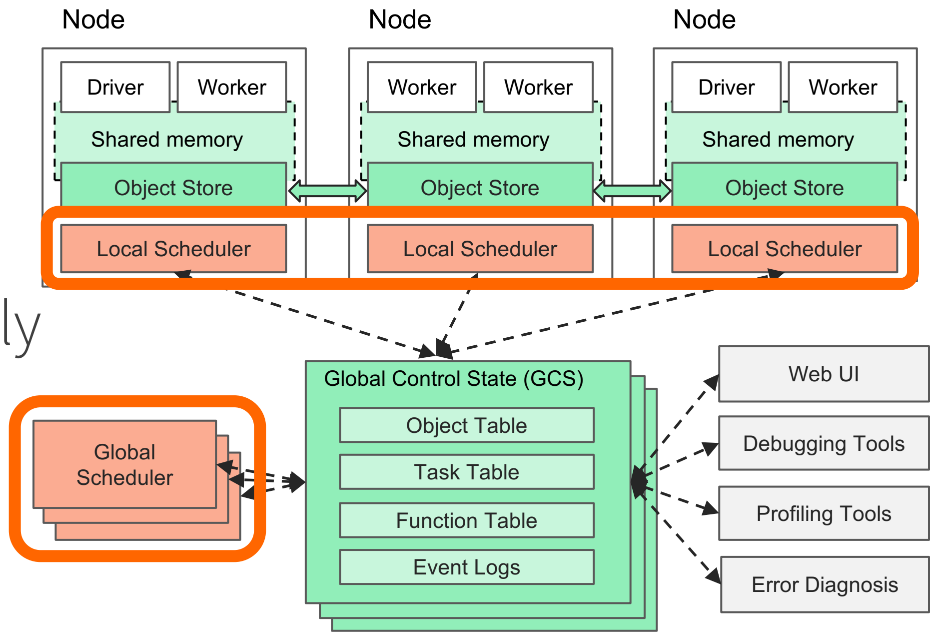


Highly Scalable

GCS: sharded database

Distributed scheduler

- Most tasks are scheduled locally
- Global scheduler plays role of load balancer



Highly Scalable

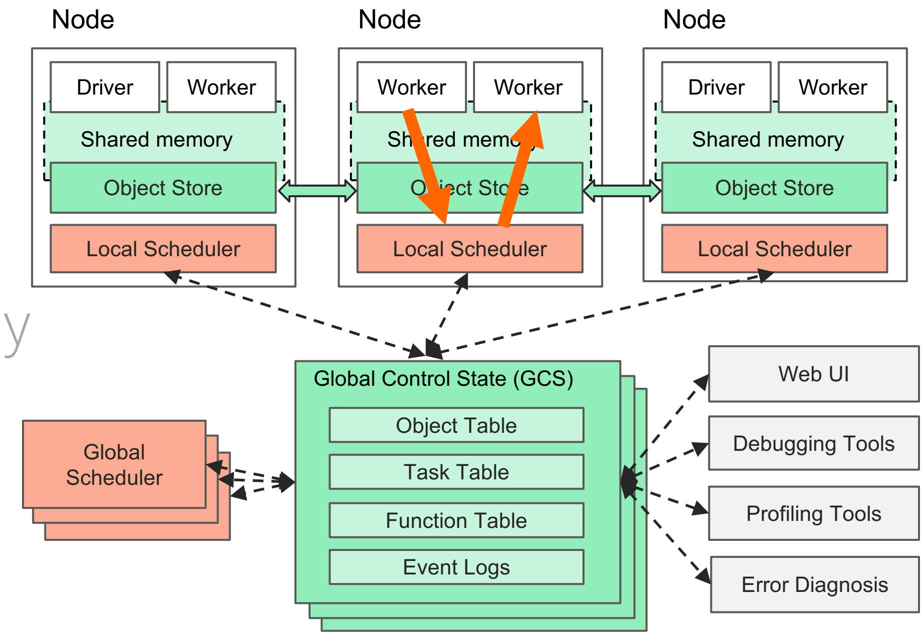
GCS: sharded database

Distributed scheduler

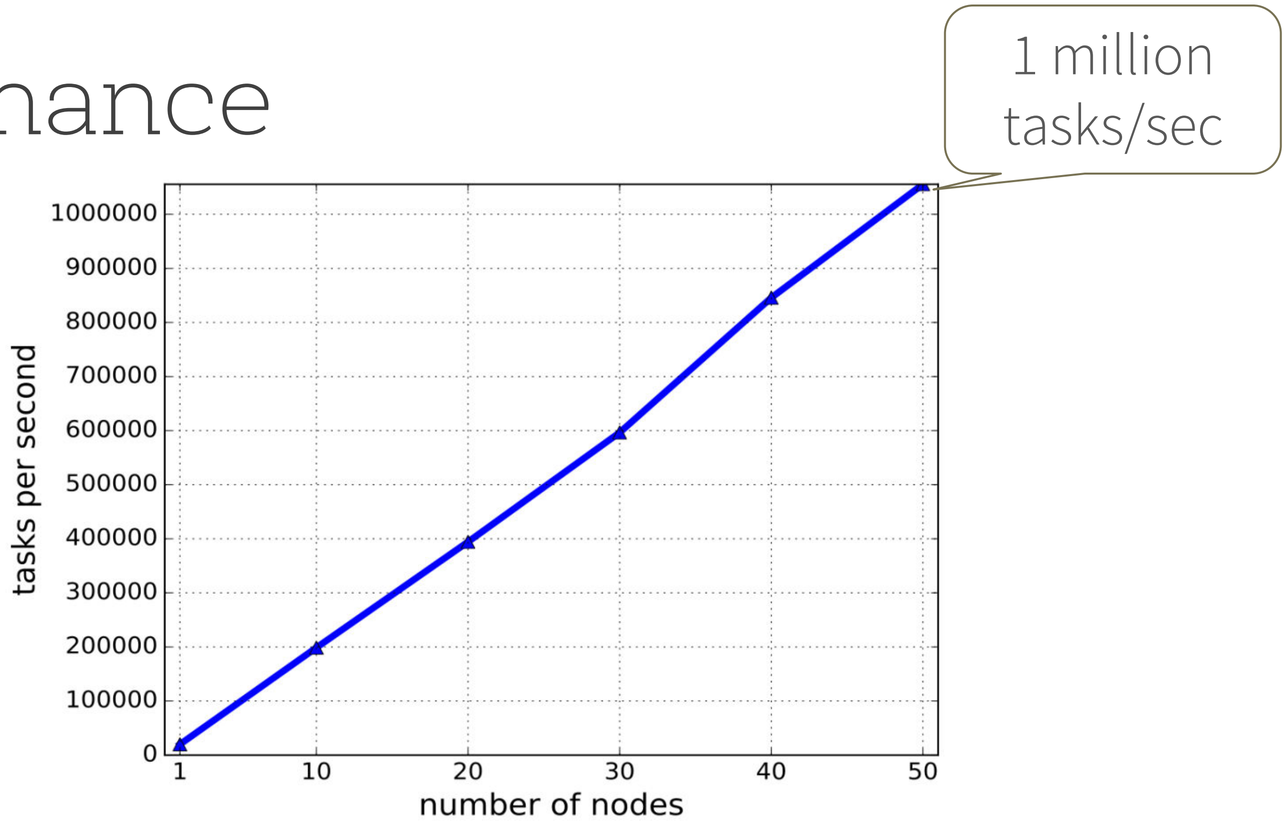
- Most tasks are scheduled locally
- Global scheduler plays role of load balancer

Tasks can spawn other tasks

- Driver not bottleneck



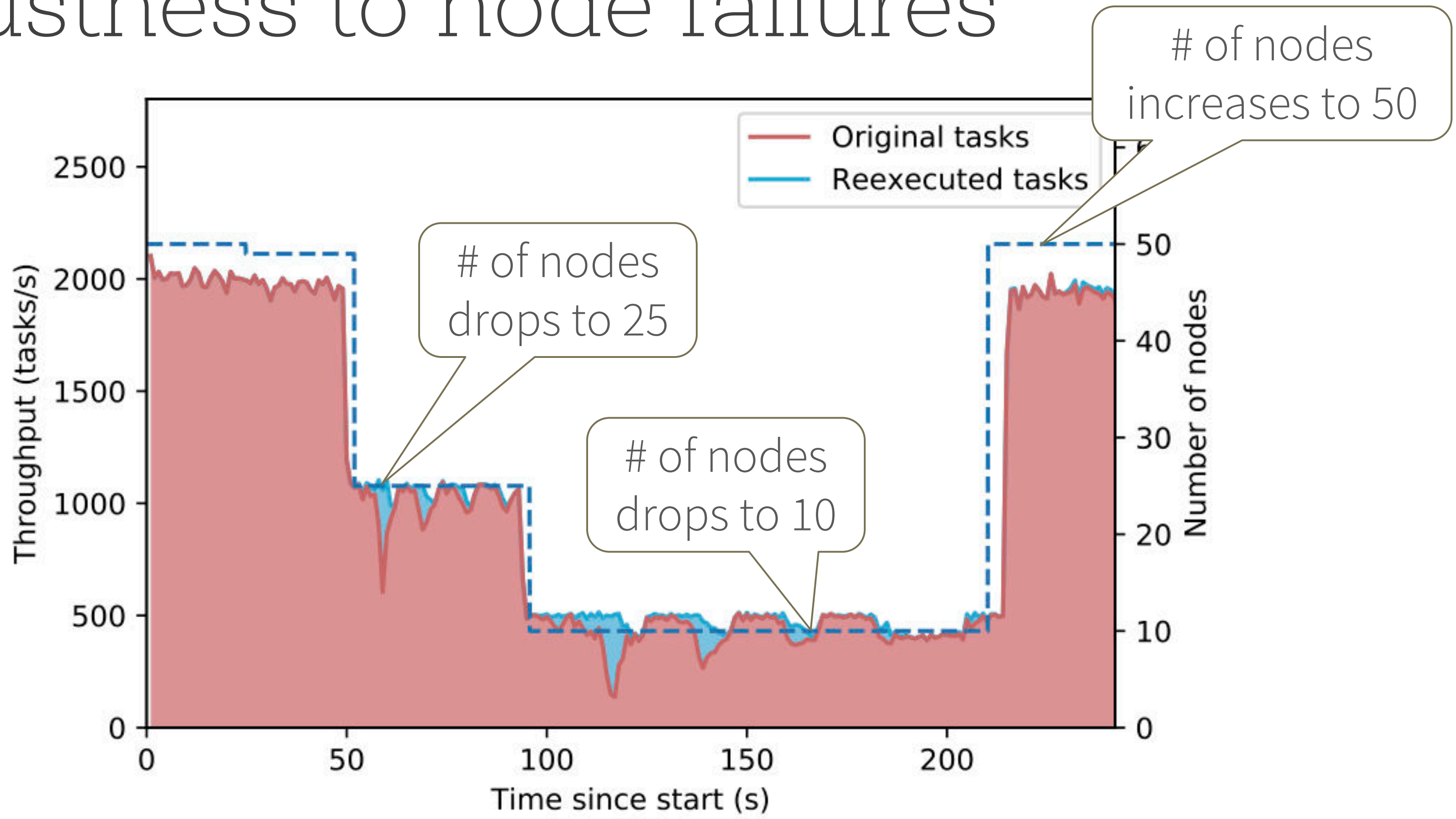
Performance



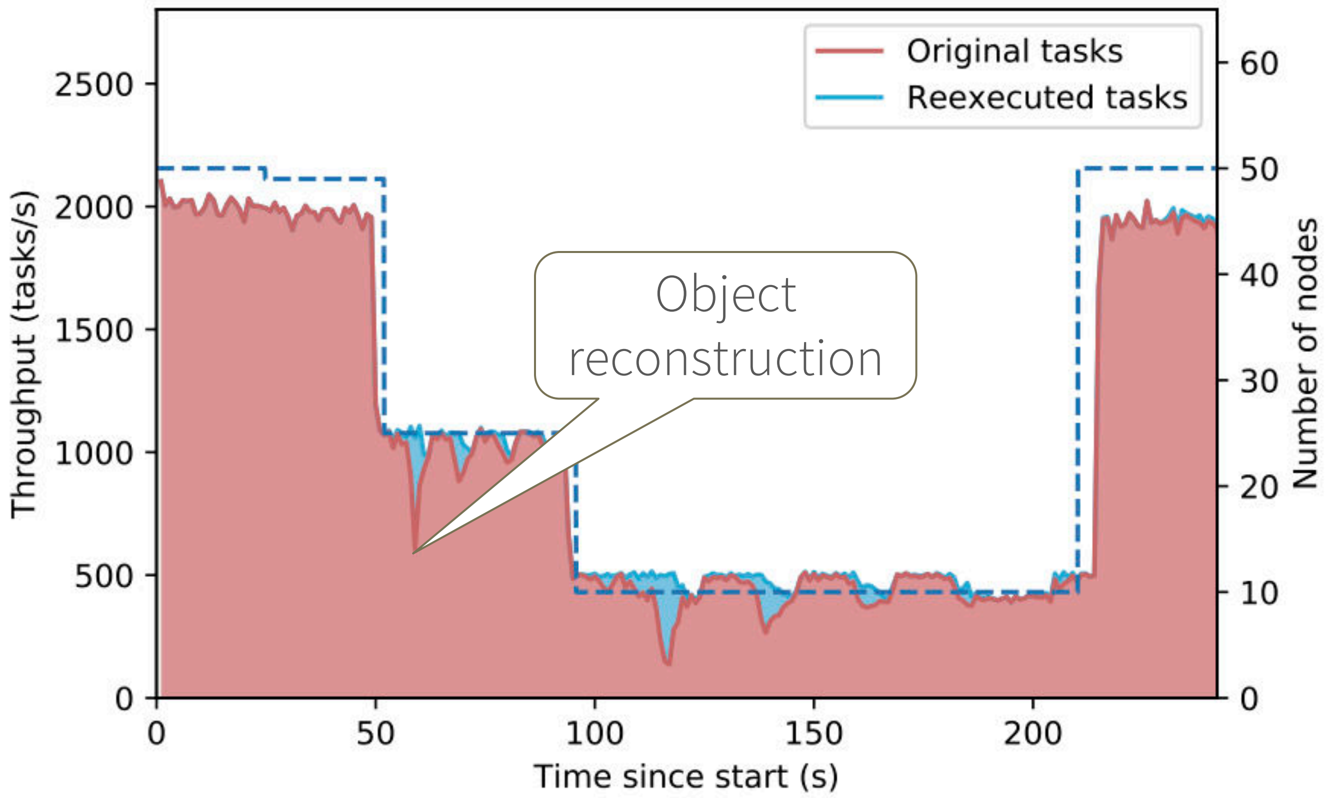
Latency of local task execution: ~300 us

Latency of remote task execution: ~1ms

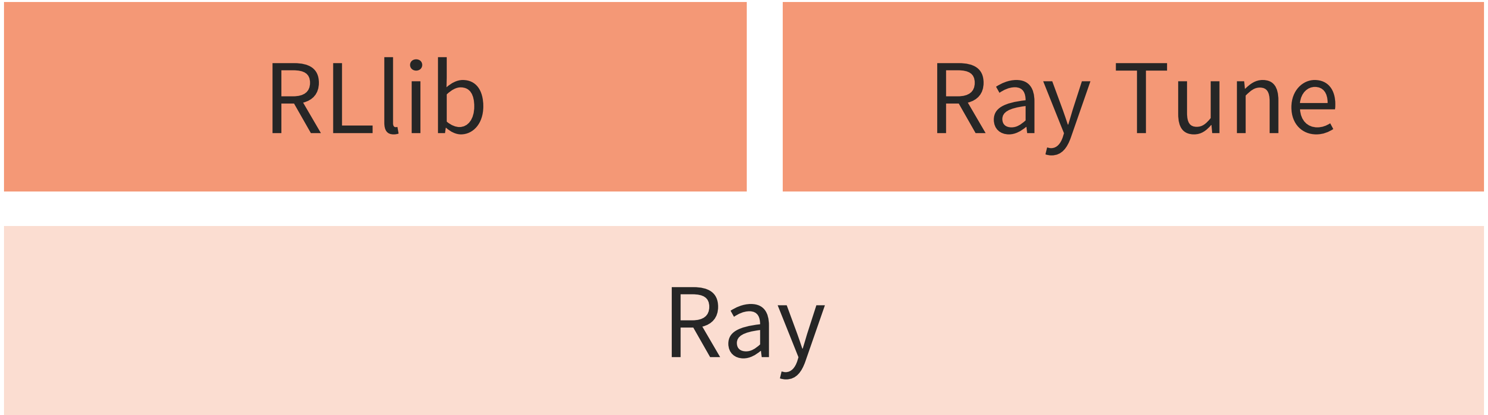
Robustness to node failures



Robustness to node failures



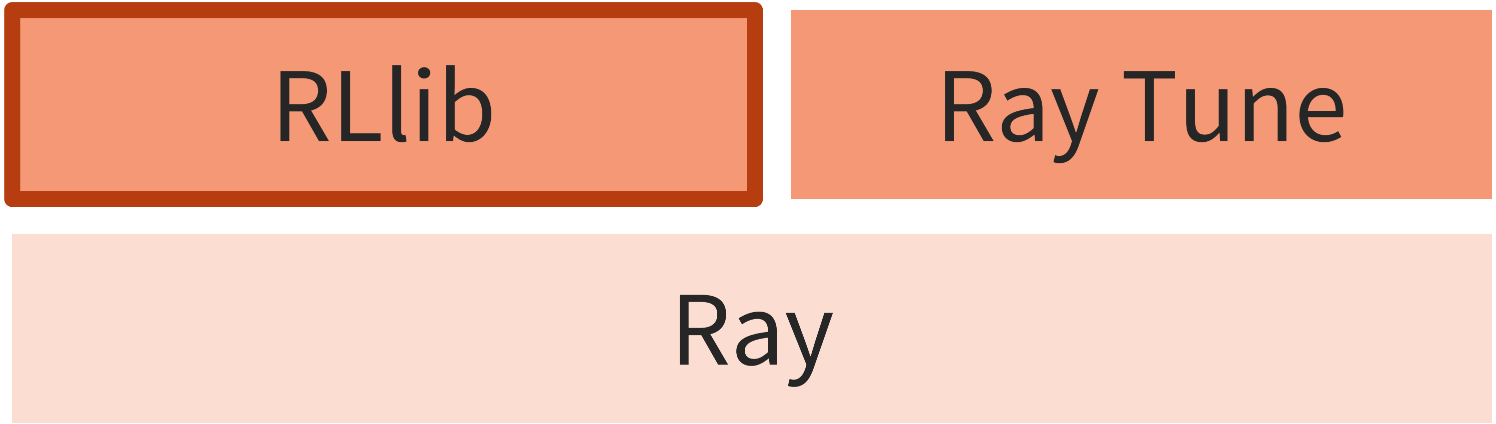
Ray libraries



RLlib: a scalable and composable RL library

Ray Tune: a flexible hyper-parameter search library

Ray libraries



RLlib: a scalable and composable RL library

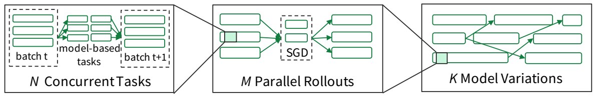
Ray Tune: a flexible hyper-parameter search library

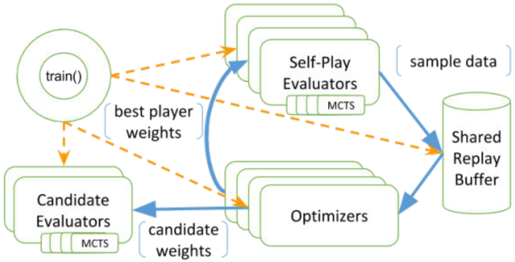
Many open source libraries for RL

Library	Framework	Components	Scaling
ChainerRL (Preferred Networks)	Chainer	Reusable Components	Multiprocessing
Baselines (OpenAI)	Tensorflow	Few reusable Components	MPI / Distributed
Coach (Intel)	Neon, TF	Reusable Components	Param-server
ELF (Facebook)	PyTorch	Reusable Components	Multiprocessing
rllab (OpenAI)	Keras, TF	Reusable Components	Multiprocessing
TensorForce	Tensorflow	Reusable Components	Param-server

None provides both **scalability** and **composability**

Rllib: scalable and composable RL Library





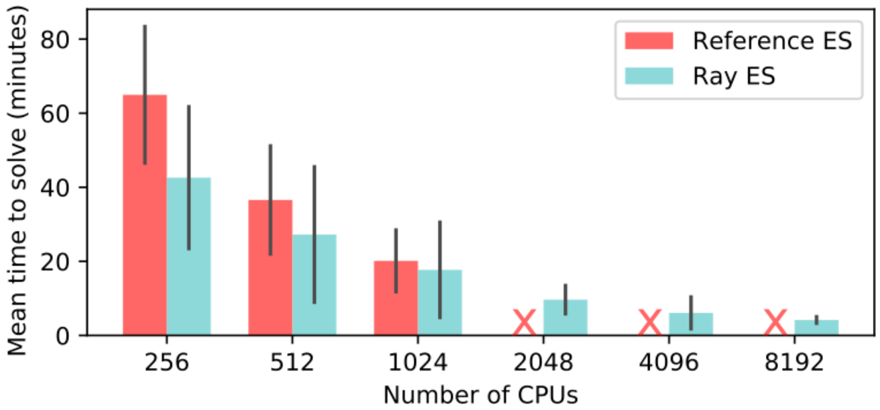
PYTORCH

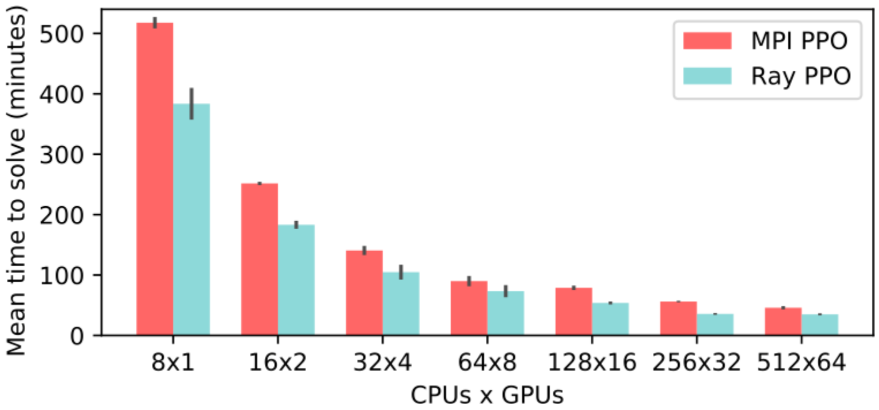
Support for nested parallelism: Distributed primitives usable within other meta-algorithms

Easy to compose distributed RL algorithms such as AlphaGo Zero

Broadly compatible with deep learning frameworks and third-party libraries

RLlib performance

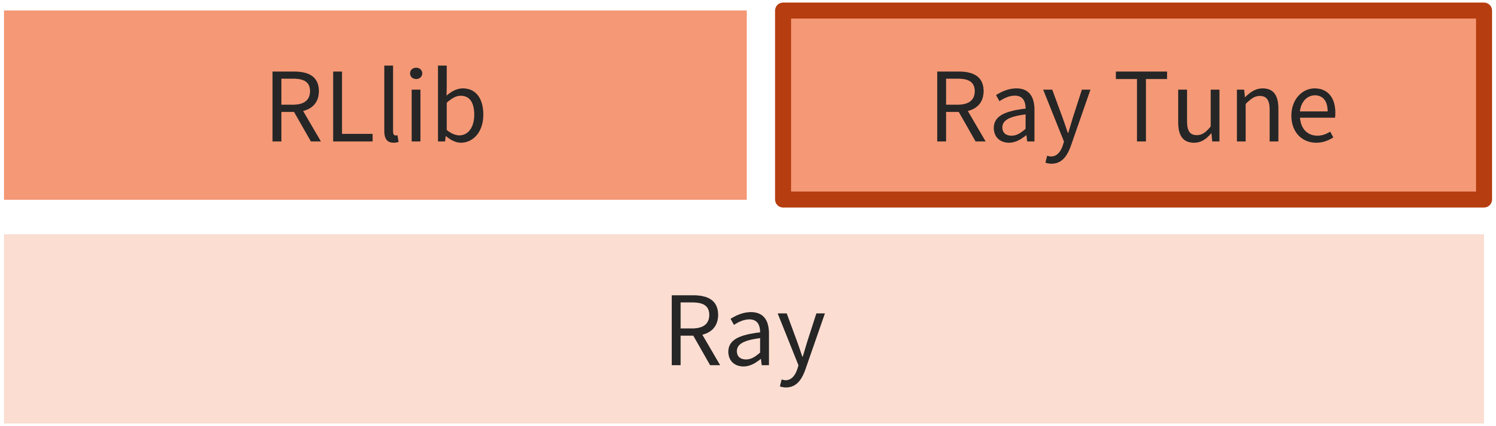




RLlib vs Redis-based
ES implementation

RLlib vs OpenAI
PPO implementation

Ray libraries



RLlib: a scalable and composable RL library

Ray Tune: a flexible hyper-parameter search library

Ray Tune

Implements a variety of search strategies

Simple API to define trainable models

```
# Trainable function
def train(config, reporter):
  for _ in range(N):
    reporter(...)
```

```
# Trainable class
class MyModel(Trainable):
  def _setup(); def _train();
  def _save(); def _restore();
```

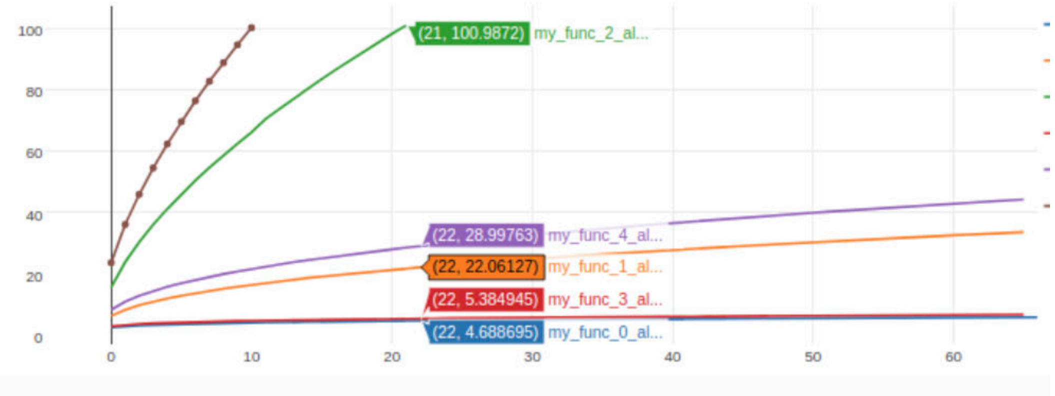
Two simple APIs for defining Trainable models

Ray Tune API

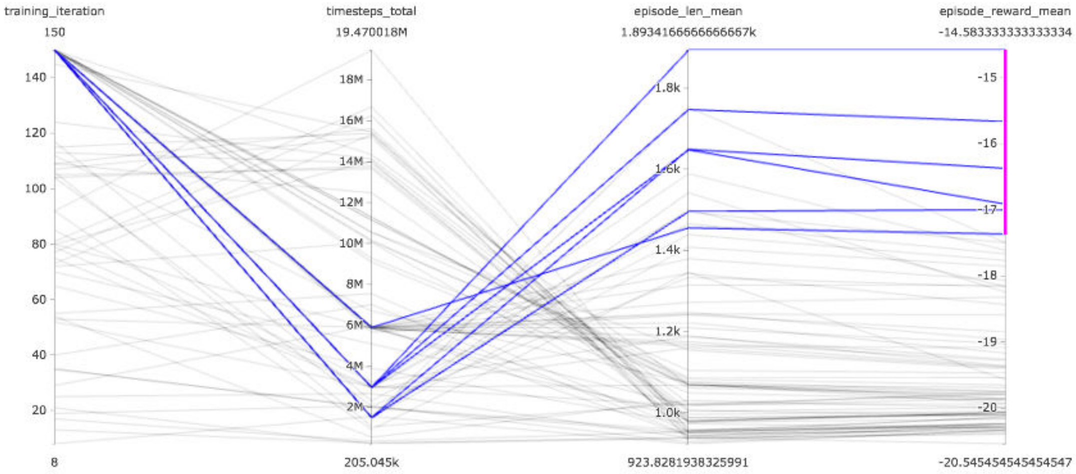
HyperBand | Grid Search | ... | Bayesian Optimization | Population Based Training

Trial schedulers implement strategies for distributed optimization

Rich visualization



rllab's VisKit



Google Vizir's parallel coordinates visualization

RL Applications

Mixed-autonomy traffic

SQL query optimization

Control hierarchies:

- Program synthesis
- Robotics manipulation

RL Applications

Mixed-autonomy traffic

SQL query optimization

Control hierarchies:

- Program synthesis
- Robotics manipulation

Mixed-autonomy traffic

Collaboration with PATH (IEOR, Berkeley)

Challenges:

- Highway accounts for 75% of transportation energy usage[1]
- Commuters waste a full week in traffic each year[2]

Question: How might a small fraction of autonomous vehicles affect traffic dynamics?

[1]https://www.nap.edu/read/12794/chapter/5
[2]https://www.cnbc.com/2016/08/09/commuters-waste-a-full-week-in-traffic-each-year.html

Single-lane experiment

Real experiment : Sugiyama, et al, 2008

230m ring road

22 human drivers

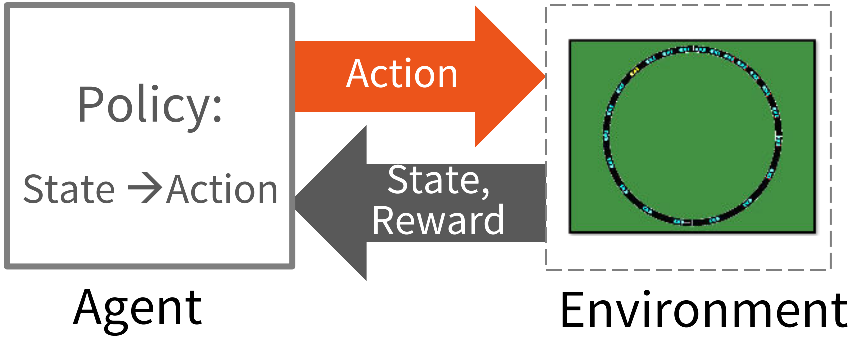
Instructions: drive at
30 km/h around the ring

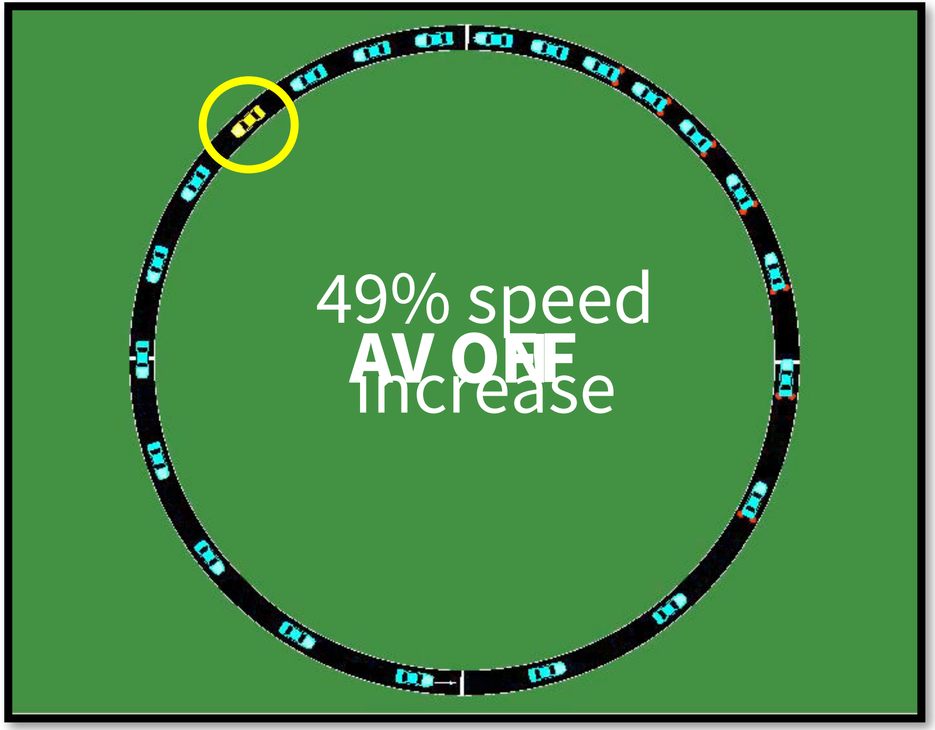
Result: traffic jams

Mixed-autonomy as RL

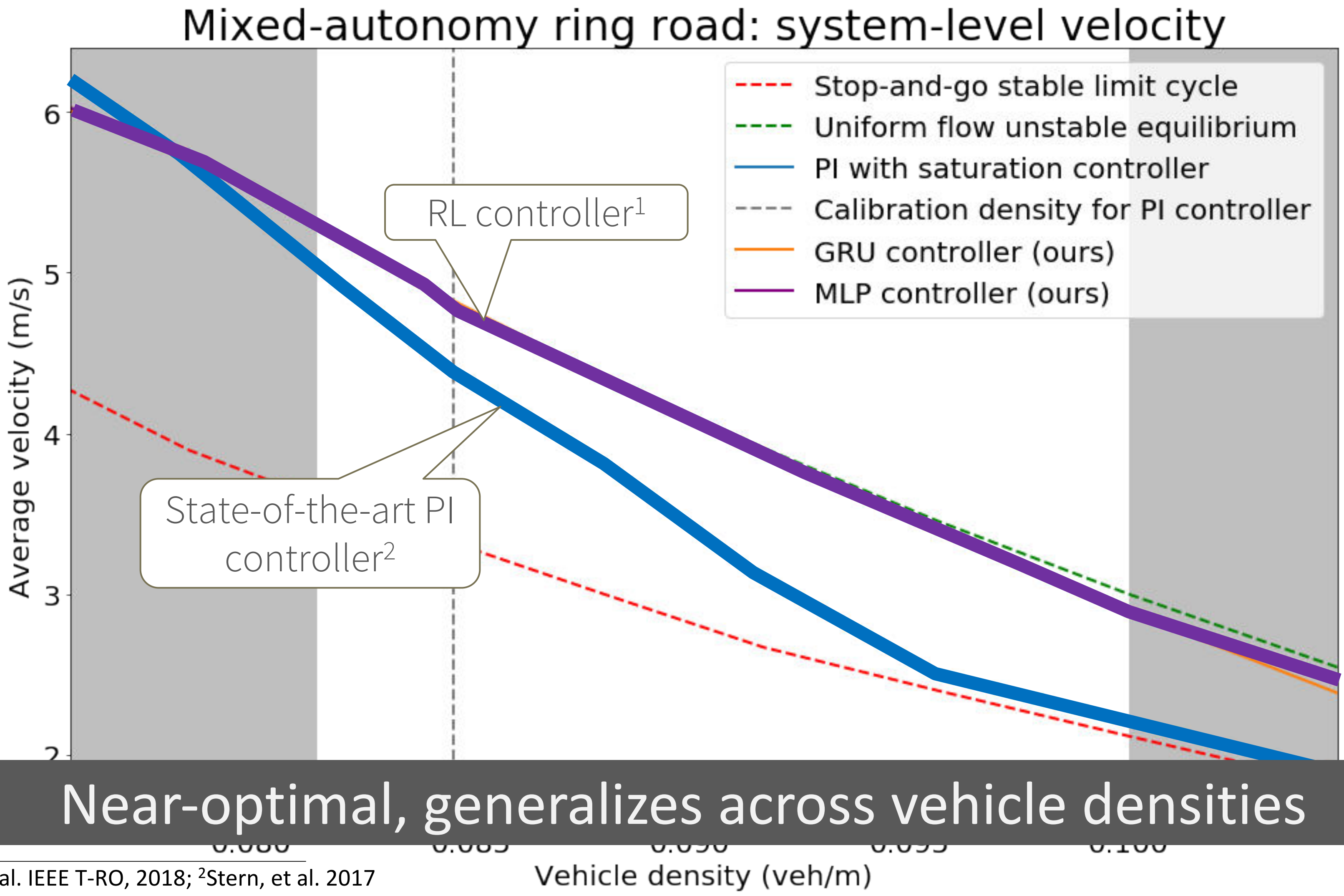
RL formulation

- **State**: car positions
- **Action**: acceleration, lane change
- **Policy**: TRPO, 3 hidden layers
- **Reward**: average velocity





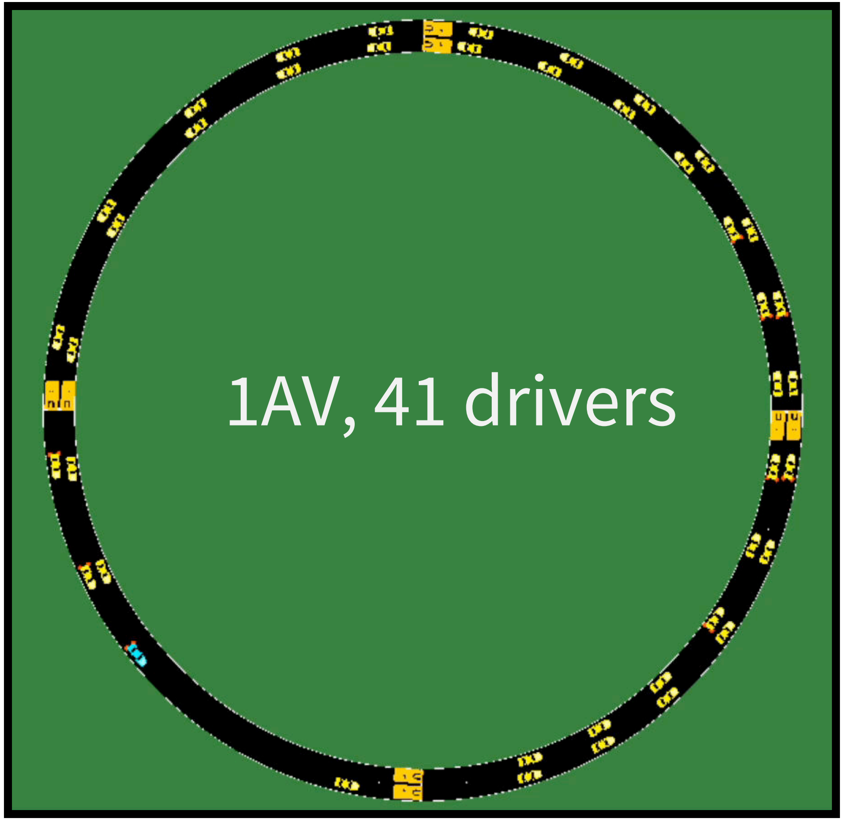
Simulation: Wu, et al. IEEE T-RO, 2018



Near-optimal, generalizes across vehicle densities

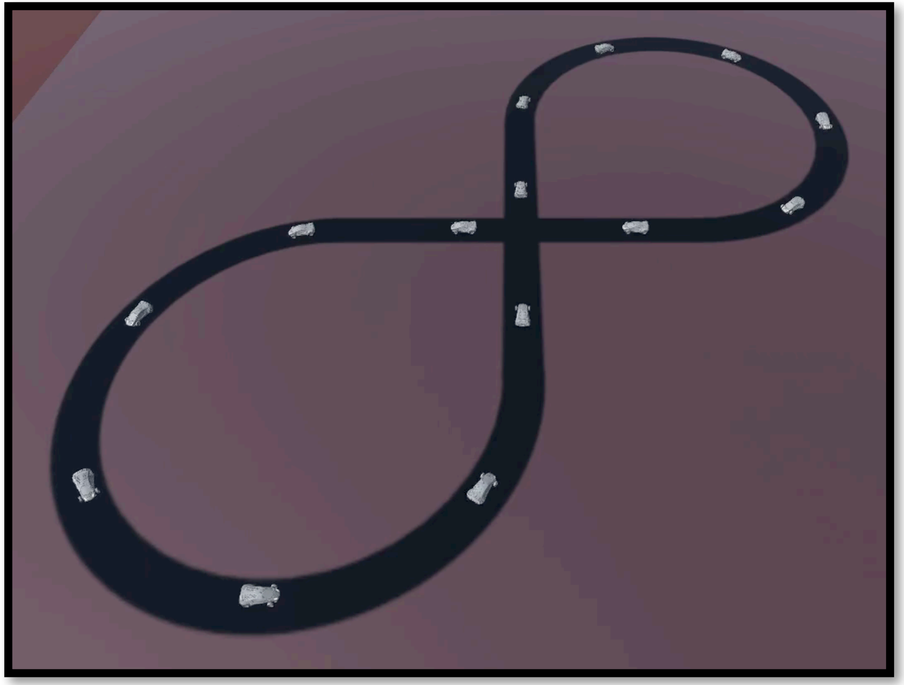
[1]**Wu**, et al. IEEE T-RO, 2018; [2]Stern, et al. 2017

Multi-lane traffic

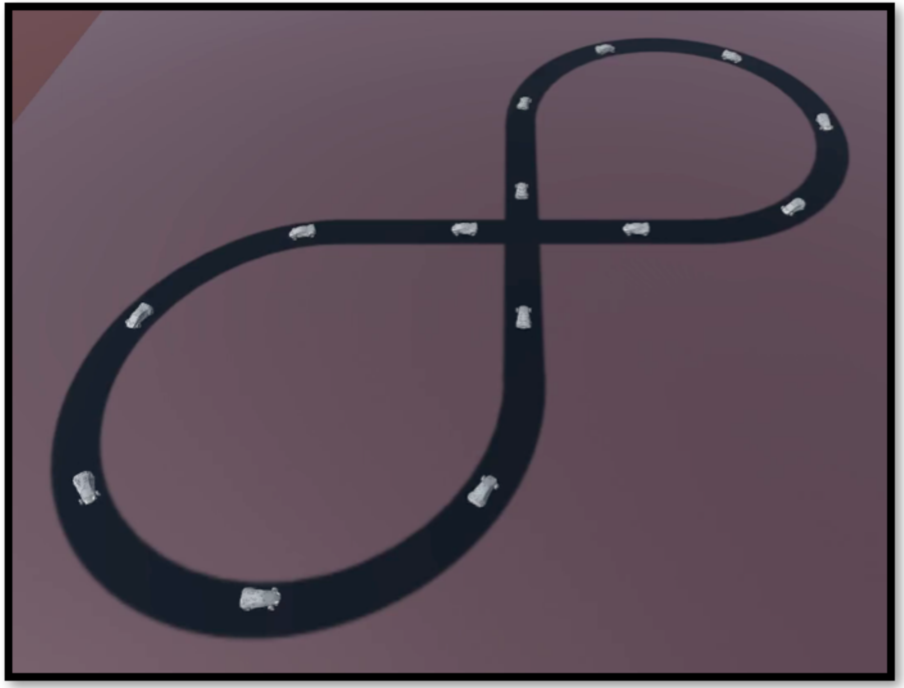


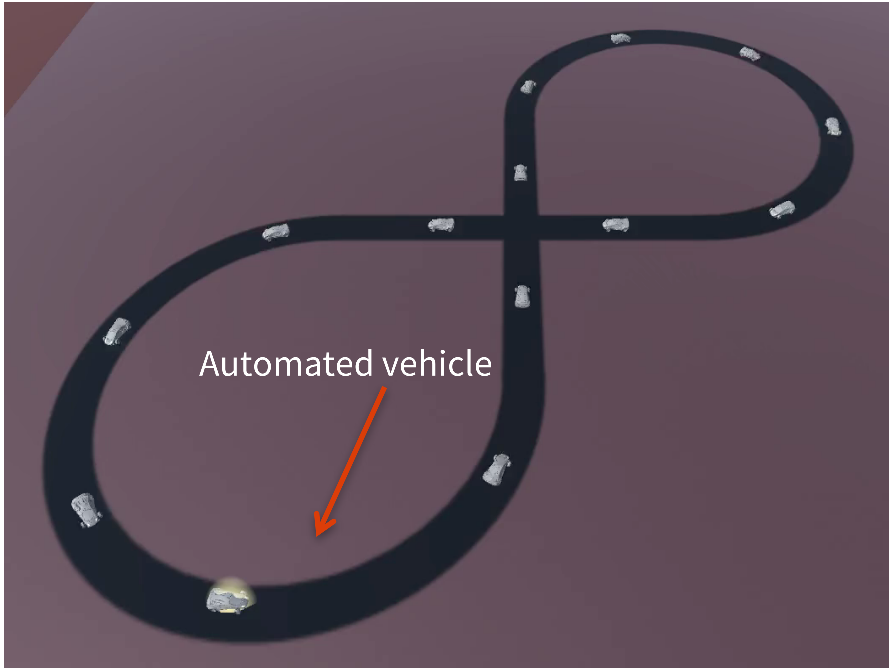
Simulation: Wu, et al. IEEE T-RO, 2018

Intersection

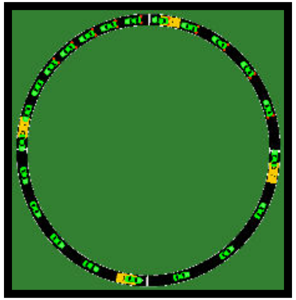


Intersection





RLlib vs rllab: preliminary results

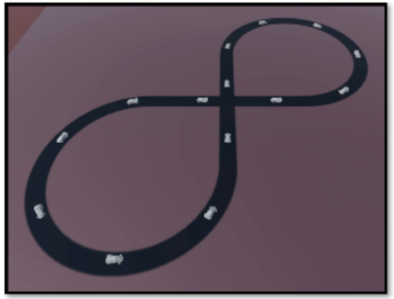


Task: stabilizing a single-lane ring
Batch: 144 trajectories
Measure: time to collect rollouts

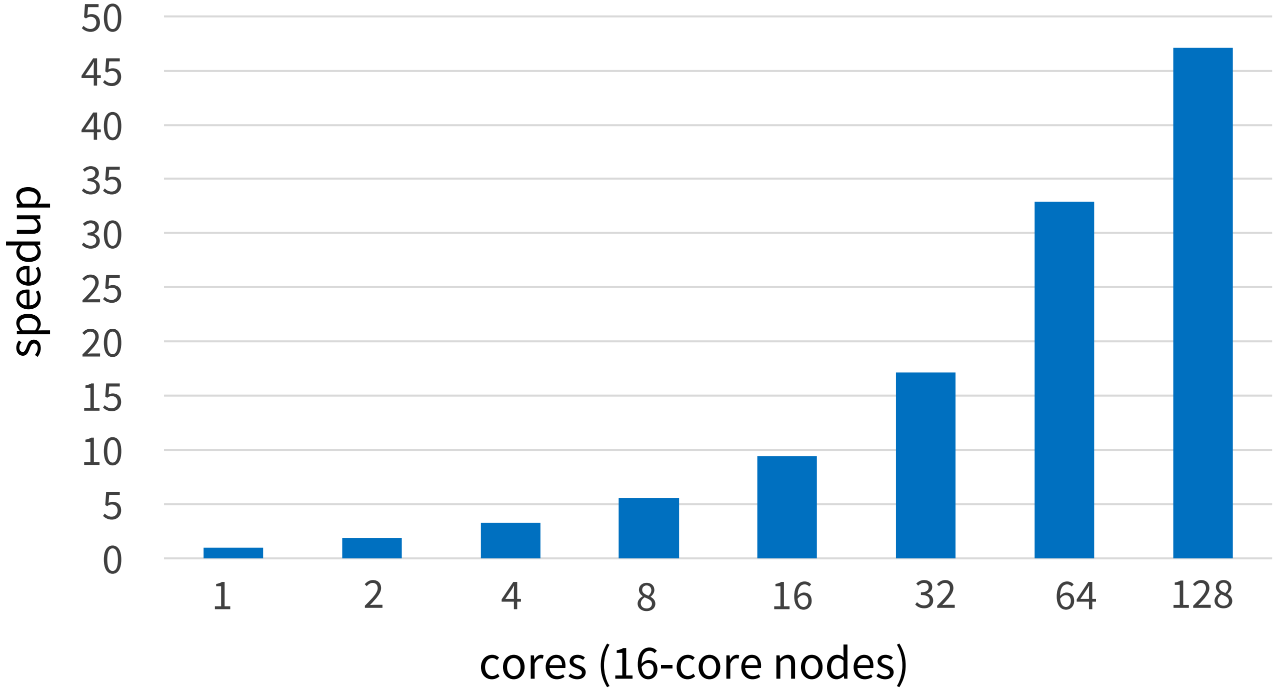
# nodes	# CPUs	rllab	RLlib (PPO)	RLlib speedup	$$*
1	16	205s	191s	1.00x	$0.24
1	72	79s	74s	2.58x	$1.08
2	32	N/A	98s	1.95x	$0.48
4	64	N/A	62s	3.08x	$0.96
8	128	N/A	42s	4.55x	$1.92

* AWS EC2 Spot Pricing (per hour)

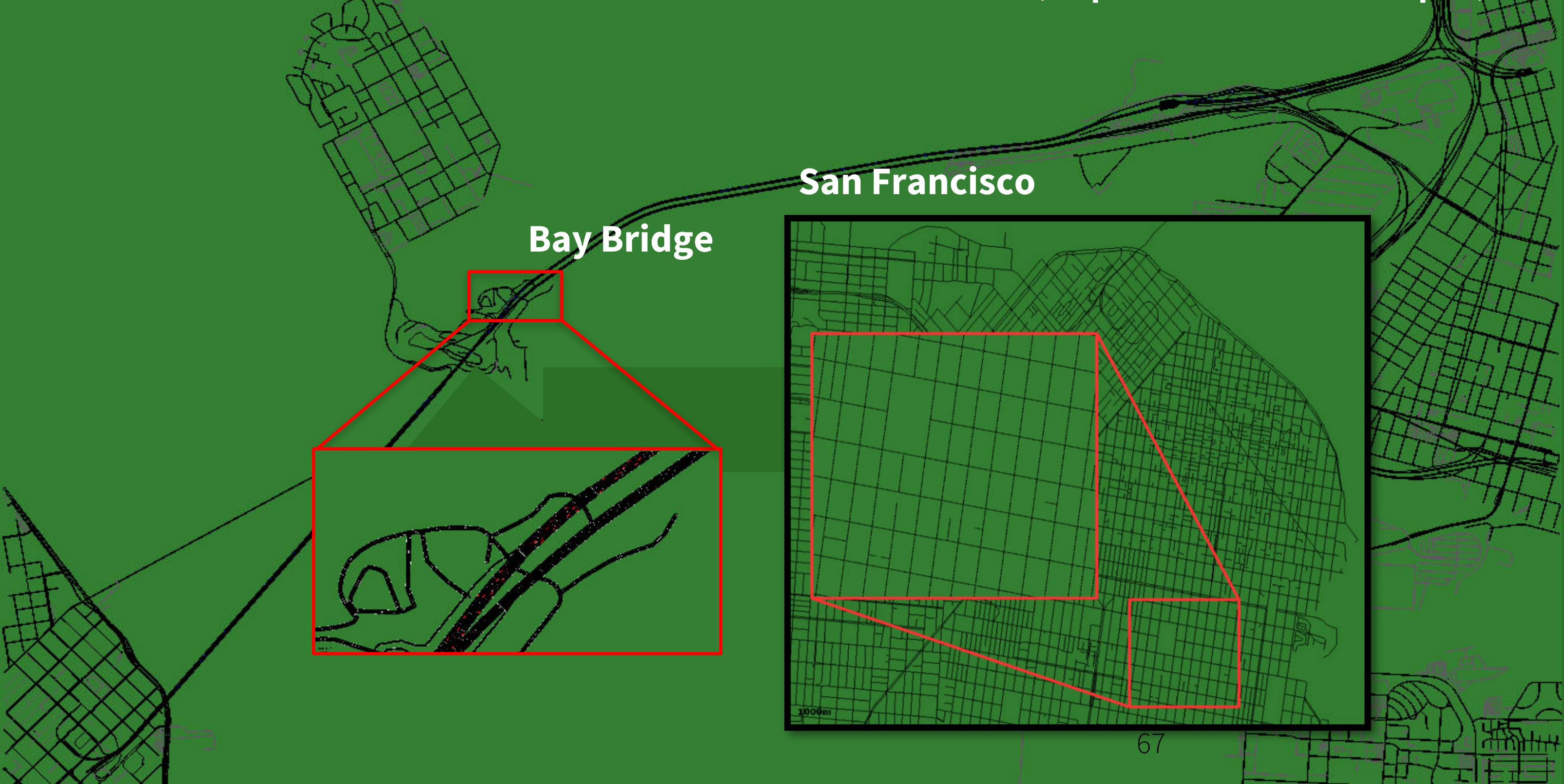
RLlib results



Task: stabilizing loopy intersection
Measure: training time



Future work: full networks (OpenStreetMaps)



RL Applications

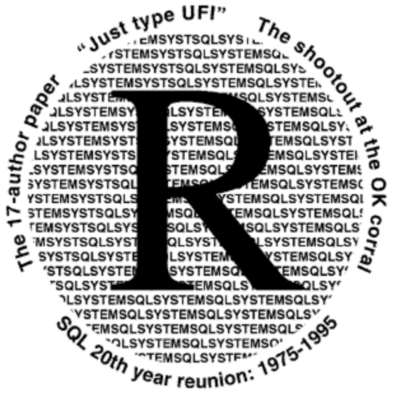
Mixed-autonomy traffic

SQL query optimization

Control hierarchies:

- Program synthesis
- Robotics manipulation

SQL query optimization (preliminary)



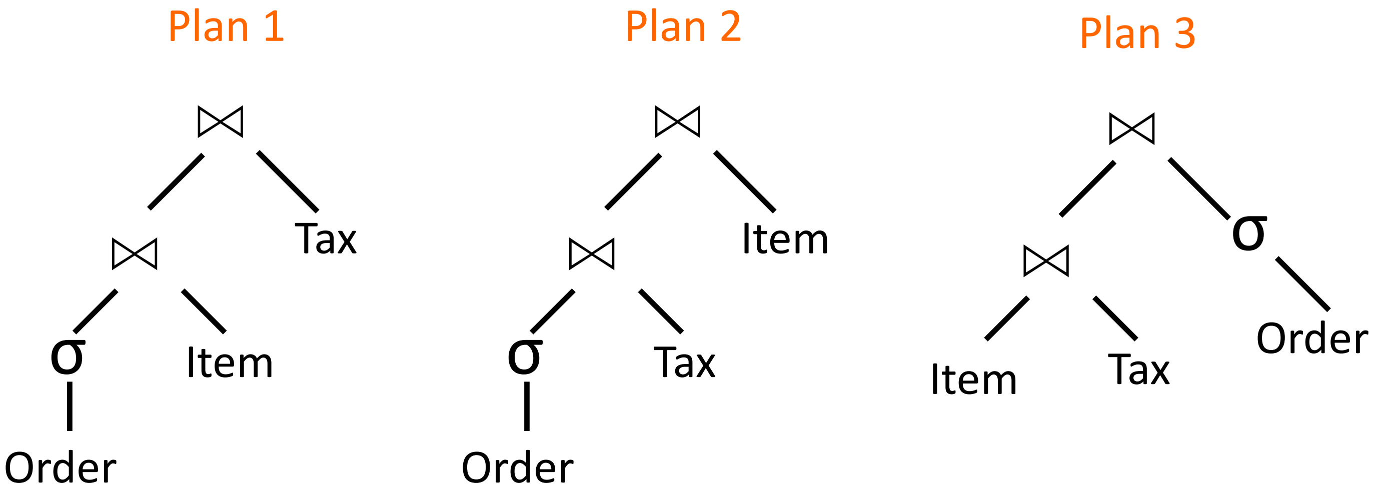
40+ years of research and 1,000s papers later…

Still hard – query optimizer highly sensitive to:

- Inaccuracies in cost model, e.g.,
  - Don't know how many distinct countries are in the database
- Dynamic execution environments, e.g.,
  - Don't know how much memory will be available during execution
  - Increasing challenge with multitenancy, e.g., BigQuery, Athena
- Heuristics, e.g., always "push down" the filter "USA"

Examples of query plans

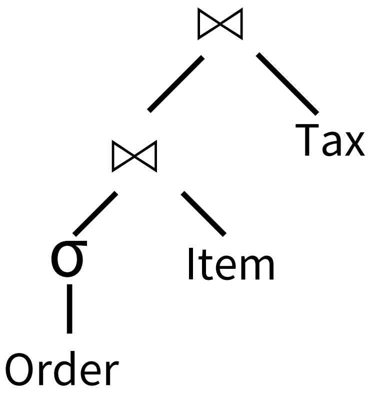
```
SELECT *
  FROM Order, Item, Tax
  WHERE Order.sku = Item.sku   AND
        Item.code = Tax.code   AND
        Order.ctry = Tax.ctry AND
        Order.ctry = 'USA'
```



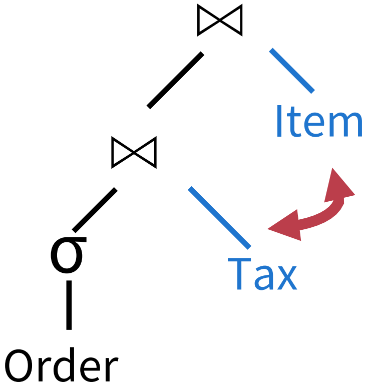
Examples of query plan

```
SELECT *
  FROM Order, Item, Tax
  WHERE Order.sku = Item.sku   AND
        Item.code = Tax.code   AND
        Order.ctry = Tax.ctry AND
        Order.ctry = 'USA'
```

Query optimization as RL



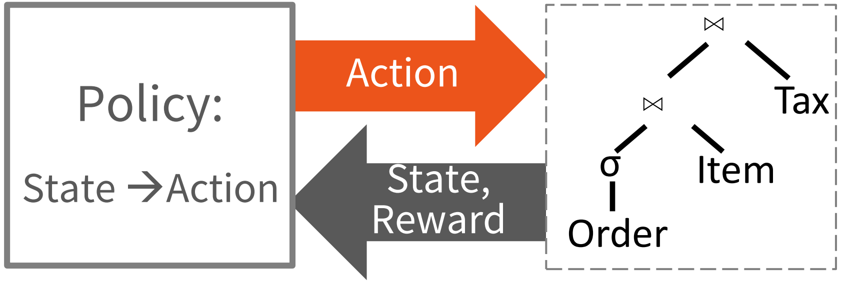
State: query plan



Action: xform

Policy: DQN, 2 hidden layers

Reward: -Runtime



Interesting results

`R(a,b,c):` 1k Records
`S(b,d):` 10k Records (multiplicity in b, no index)
`T(c,e):` 10k Records (multiplicity in c, no index)

500 training queries randomly sampled

Test Query:
```
SELECT count(1)
  FROM R,S,T
  WHERE R.b = S.b  AND
        R.c = T.c
  GROUP BY b,c
```

Interesting results

`R(a,b,c):` 1k Records
`S(b,d):` 10k Records (multiplicity in b, no index)
`T(c,e):` 10k Records (multiplicity in c, no index)

Postgres Plan: $\gamma((R \bowtie S) \bowtie T)$ Aggregates after all joins
Learned Plan: $\gamma(T \times S) \bowtie R$ Aggregates before R join

Postgres Plan: 18.3 s
Learned Plan: **3.9 s**
4.7x

Avoiding cartesian products is a common heuristic

Interesting results

`R(a,b,c)`: 1k Records

`udf(b)`: Expensive UDF

500 training queries randomly sampled

Test Query:

```sql
SELECT count(1)
  FROM R
  WHERE UDF(b)
  GROUP BY b
```

Interesting results

`R(a,b,c)`: 1k Records
`udf(b)`: Expensive UDF

Postgres Plan: $\gamma(\sigma(R))$ — Push down predicate

Learned Plan: $\sigma(\gamma(R))$ — Aggregate First

Postgres Plan: 1.27 s
Learned Plan: **0.212 s**

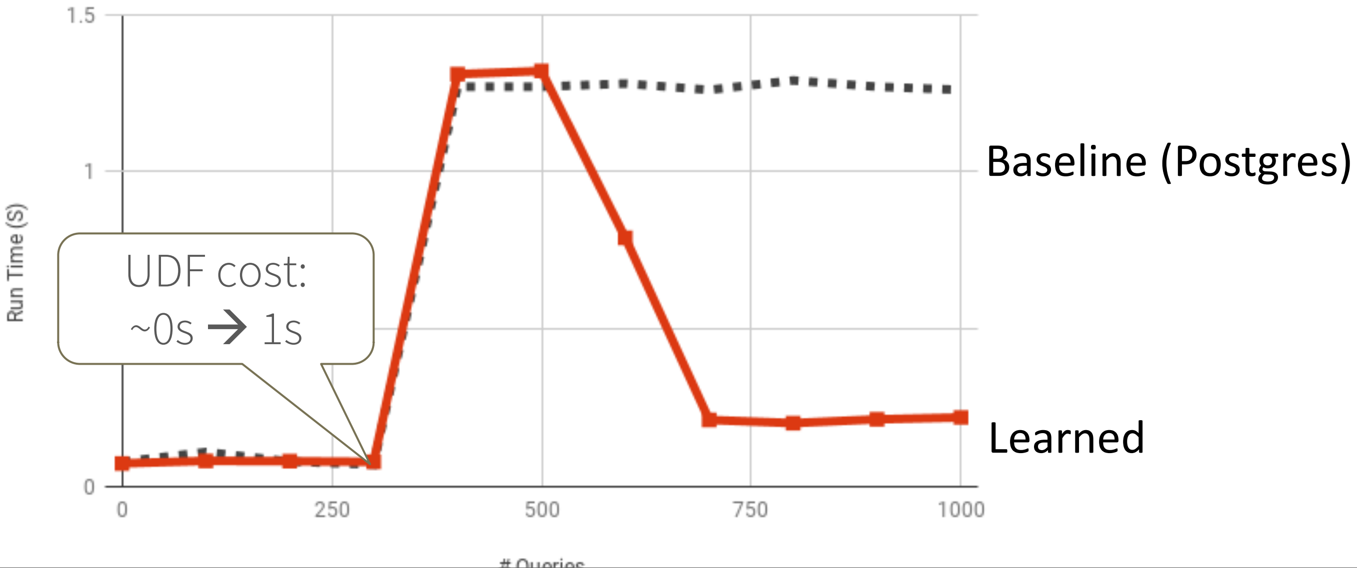
8x

Cost models often fail to consider UDFs

Interesting Results

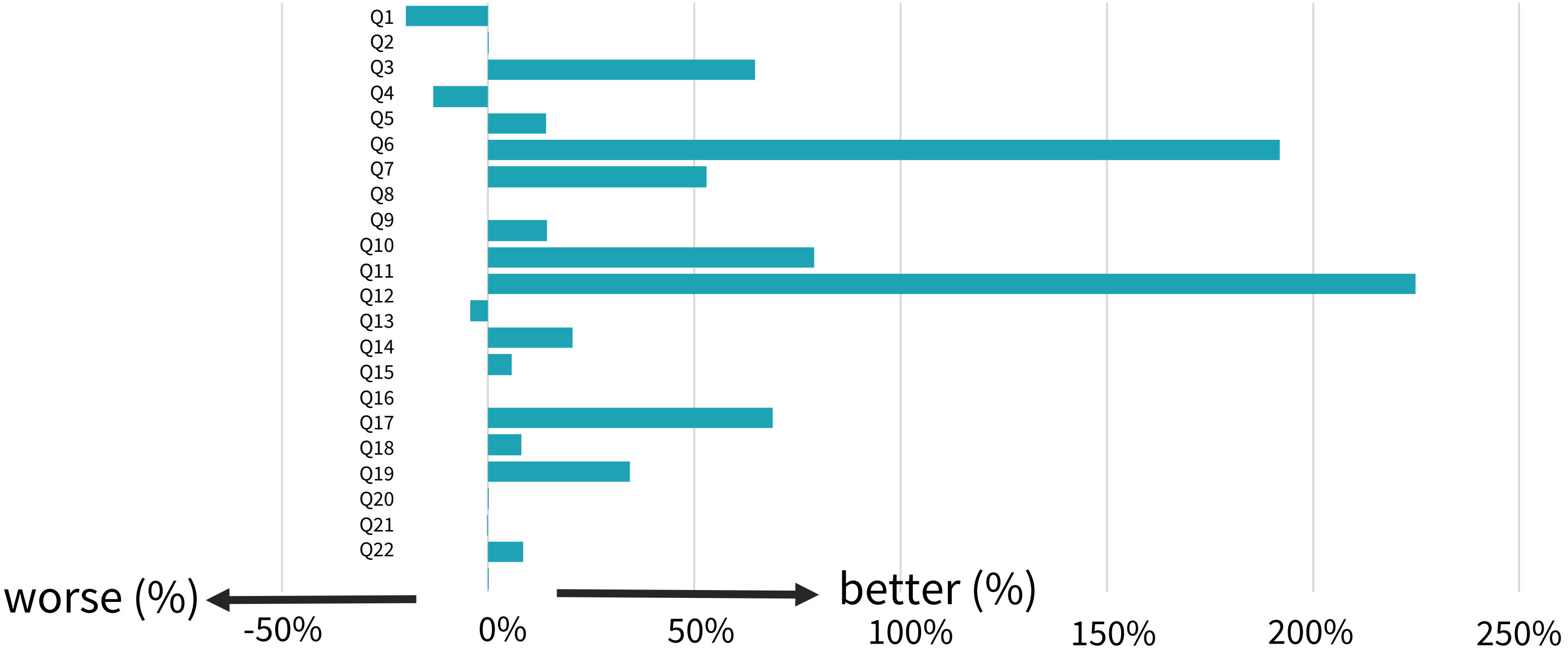
`R(a,b,c):` 1k Records

`udf(b):` UDF becomes more expensive to execute



Can adapt to dynamic environments

TPCH: preliminary results



Improves Postgres query optimizer performance

(10k training queries, 100K test queries)

Future work

Improve generality, e.g.

- Nested queries, sort orders, more complex rewriting rules
- Select physical operator

Apache SparkSQL integration

RL Applications

Mixed-autonomy traffic

SQL query optimization

Control hierarchies:

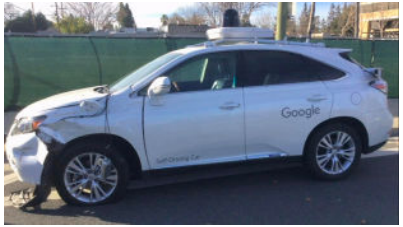
- Program synthesis
- Robotics manipulation

Sample efficiency

One of the big challenges of RL

Best case, it can take long time to converge

Worst case, can be very expensive, even unsafe to do many experiments



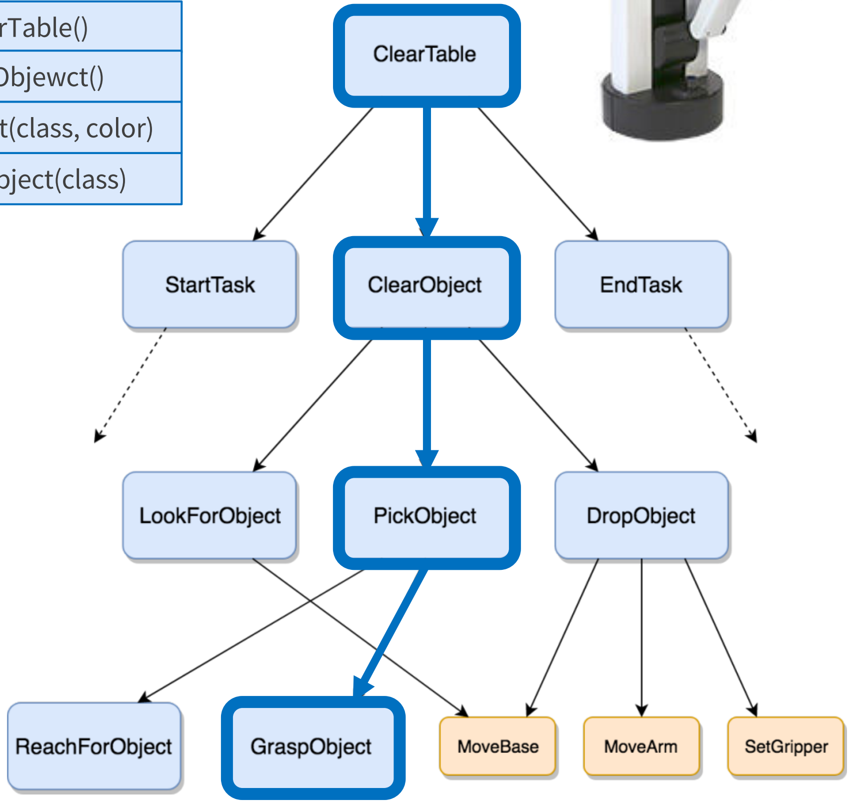
Control hierarchies

Aggregate low level action in higher level procedures

- Each proc. can
  - Call sub-procedures
  - Take actions
  - Terminate
- Procedures take arguments
- State is entire call-stack

call stack

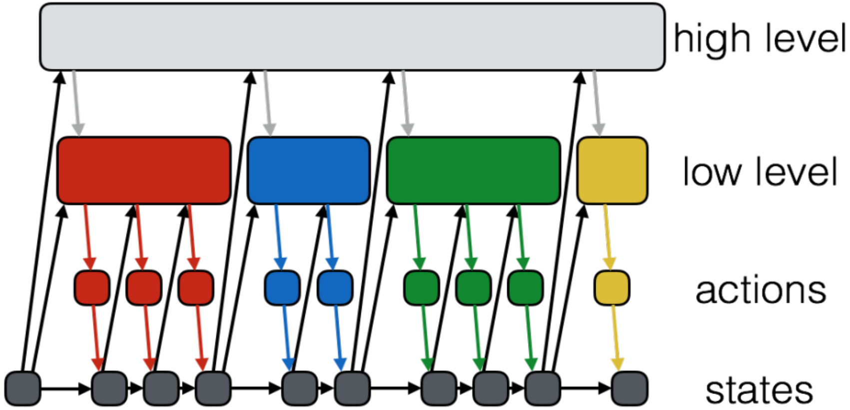
ClearTable()
ClearObjewct()
PickObject(class, color)
GraspObject(class)



Imitation learning

Complete demonstration includes:

- States / observations
- Elementary actions
- Procedure calls and terminations
- Call stack



Strong vs. weak supervision

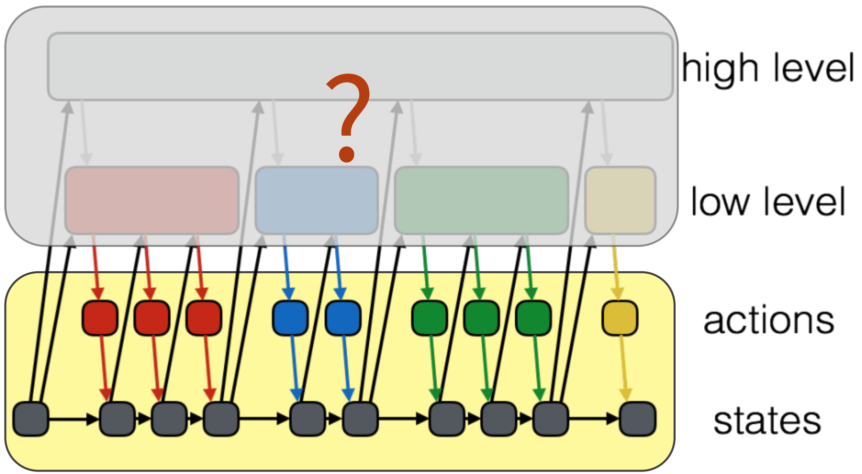
Call stack visible in demonstration?

Yes → strong supervision

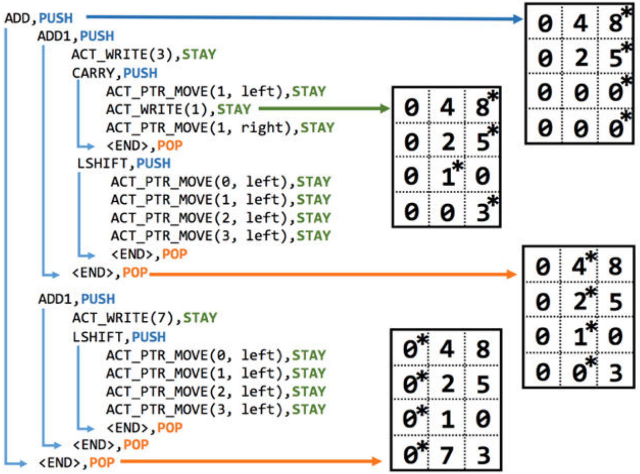
- Imitate hierarchical structure

No → weak supervision

- Need to fill in hierarchy



Experiment long-hand addition*



Model	Strongly-supervised traces	Accuracy for input length	
		500	1000
NPI (Reed & de Freitas, 2016)[2]	160	<100%	<100%
NPL (Li et al., 2017)	10	**100%**	<100%
PHP	3	**100%**	**100%**

* Fox et al., ICLR 2018

Shared Imitation Learning of Hierarchical Skills for Robot Control

Roy Fox*, Ron Berenstein*, Sanjay Krishnan,
Pieter Abbeel, Ion Stoica, Ken Goldberg

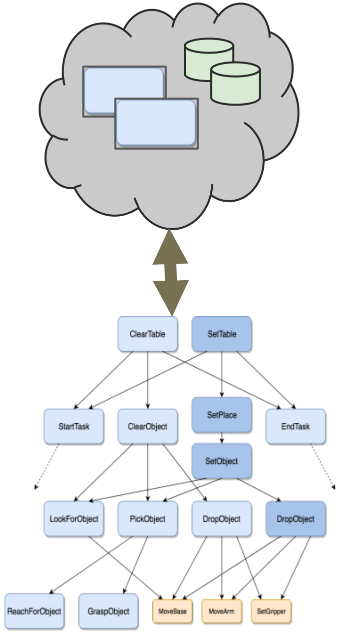
riselab UC Berkeley
AUTOLAB
Berkeley UNIVERSITY OF CALIFORNIA
BAIR BERKELEY ARTIFICIAL INTELLIGENCE RESEARCH

Future work

Imitation learning using weak supervision

Shared learning – a procedure skill can be:

- **Reuse**: apply current procedure to new task
- **Retrain**: train procedure to work for all tasks
- **Recreate**: train separate procedure for new task



Summary

RISELab goal: Develop open source platforms, tools and algorithms for intelligent real-time decisions on live data that are secure and explainable

Many decisions leverage AI/RL

Ray: a system for distributed AI

- RLlib and Ray Tune support highly scalable RL apps
- Open source: https://github.com/ray-project/ray
- Install: **`pip install ray`**

Promising RL apps