Ray - Scalability from a Laptop to a Cluster

Dean Wampler - April 8, 2020
dean@anyscale.com
@deanwampler
https://ray.io
https://anyscale.com

Checkout our online events this Summer:

https://anyscale.com/events

RAY

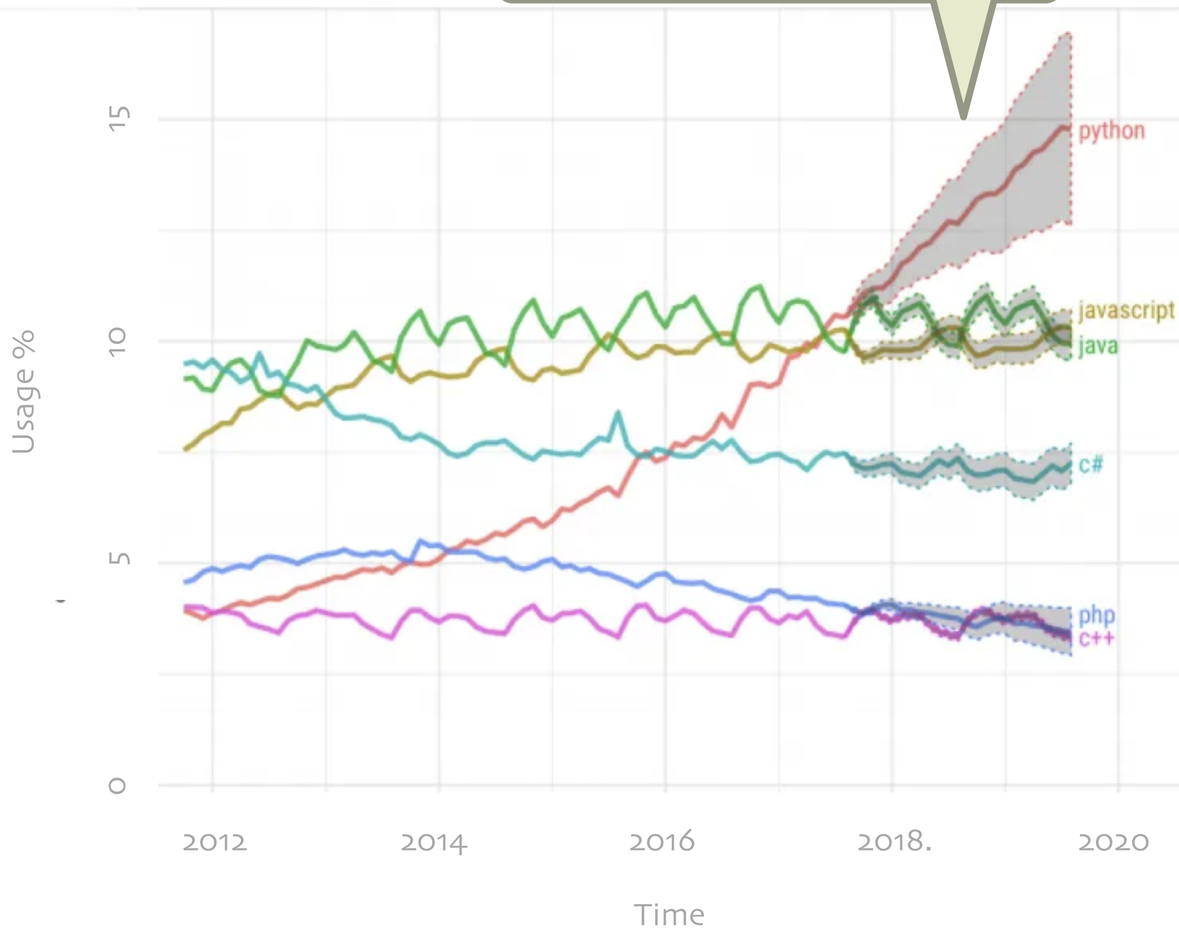
anyscale

Two Major Trends

Model sizes and therefore compute requirements outstripping Moore's Law

Hence, there is a pressing need for robust, easy to use solutions for distributed Python

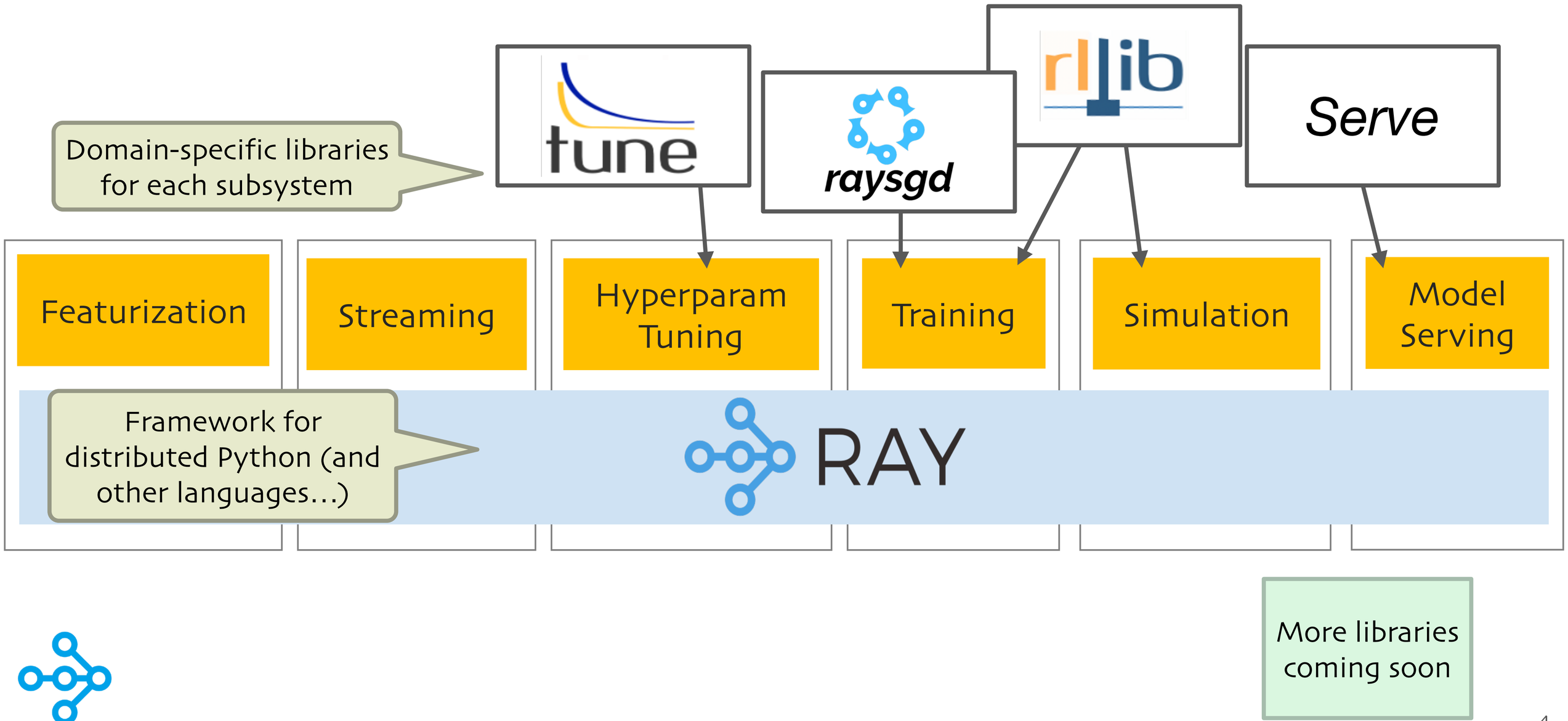
Python growth driven by ML/AI and other data science workloads



The ML Landscape Today

All require distributed implementations to scale

The Ray Vision: Sharing a Common Framework



API - Designed to Be Intuitive and Concise

Functions -> Tasks

```
def make_array(…):
    a = … # Construct a NumPy array
    return a


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

The Python you already know…

API - Designed to Be Intuitive and Concise

Functions -> Tasks

```
@ray.remote
def make_array(…):
    a = … # Construct a NumPy array
    return a

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

Now these functions are remote “tasks"

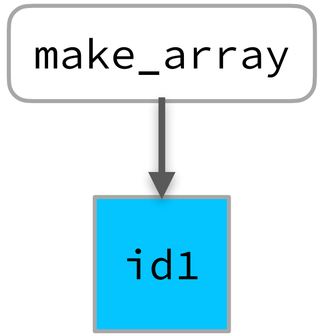
For completeness, add these first:

```
import ray
import numpy as np
ray.init()
```

API - Designed to Be Intuitive and Concise

Functions -> Tasks

```
@ray.remote
def make_array(…):
    a = … # Construct a NumPy array
    return a

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

id1 = make_array.remote(…)
```



API - Designed to Be Intuitive and Concise

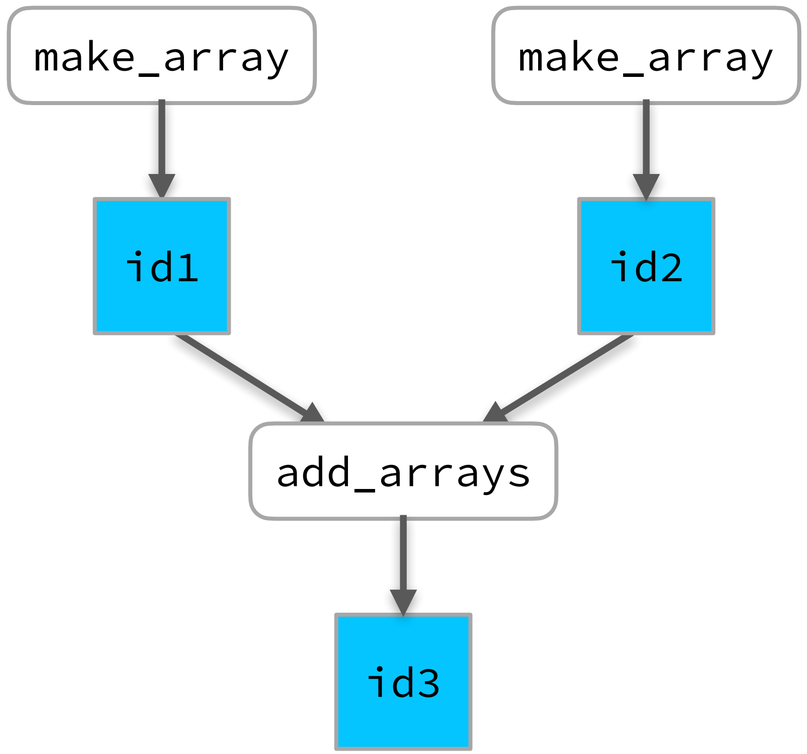
Functions -> Tasks

```
@ray.remote
def make_array(…):
    a = … # Construct a NumPy array
    return a

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

id1 = make_array.remote(…)
id2 = make_array.remote(…)
```

API - Designed to Be Intuitive and Concise

Functions -> Tasks

```
@ray.remote
def make_array(…):
    a = … # Construct a NumPy array
    return a

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

id1 = make_array.remote(…)
id2 = make_array.remote(…)
id3 = add_arrays.remote(id1, id2)
```



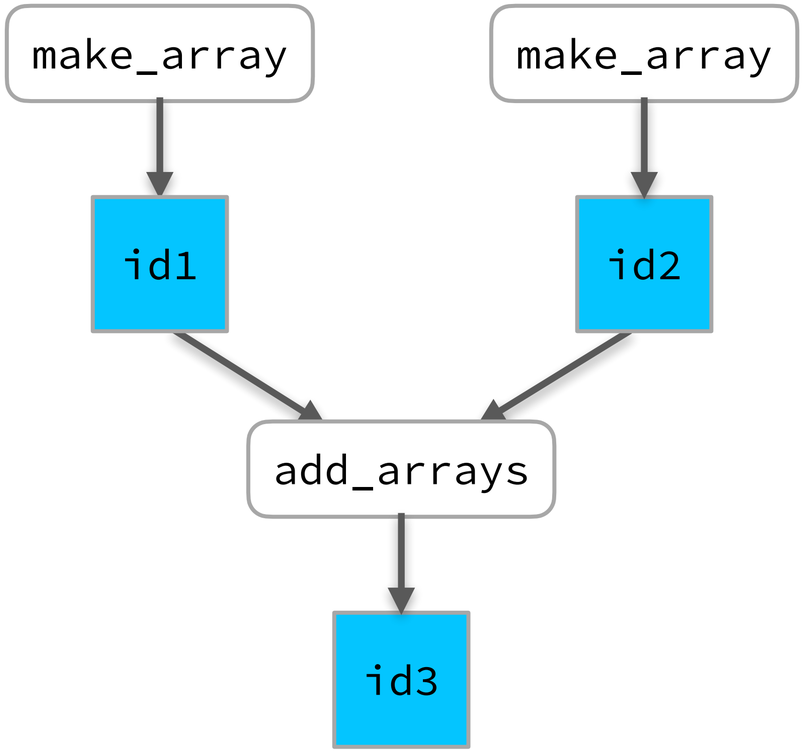
API - Designed to Be Intuitive and Concise

Functions -> Tasks

```
@ray.remote
def make_array(…):
    a = … # Construct a NumPy array
    return a

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

id1 = make_array.remote(…)
id2 = make_array.remote(…)
id3 = add_arrays.remote(id1, id2)
ray.get(id3)
```

Ray handles extracting the arrays from the object ids

Ray handles sequencing of async dependencies



API - Designed to Be Intuitive and Concise

What about distributed state?

Functions -> Tasks

```
@ray.remote
def make_array(…):
    a = … # Construct a NumPy array
    return a

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

id1 = make_array.remote(…)
id2 = make_array.remote(…)
id3 = add_arrays.remote(id1, id2)
ray.get(id3)
```

API - Designed to Be Intuitive and Concise

Functions -> Tasks

```
@ray.remote
def make_array(…):
    a = … # Construct a NumPy array
    return a

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

id1 = make_array.remote(…)
id2 = make_array.remote(…)
id3 = add_arrays.remote(id1, id2)
ray.get(id3)
```

Classes -> Actors

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

The Python classes you love…

API - Designed to Be Intuitive and Concise

Functions -> Tasks

```
@ray.remote
def make_array(…):
    a = … # Construct a NumPy array
    return a

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

id1 = make_array.remote(…)
id2 = make_array.remote(…)
id3 = add_arrays.remote(id1, id2)
ray.get(id3)
```

Classes -> Actors

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

… now a remote "actor"

You need a "getter" method to read the state.

API - Designed to Be Intuitive and Concise

Functions -> Tasks

```
@ray.remote
def make_array(…):
    a = … # Construct a NumPy array
    return a

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

id1 = make_array.remote(…)
id2 = make_array.remote(…)
id3 = add_arrays.remote(id1, id2)
ray.get(id3)
```

Classes -> Actors

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

c = Counter.remote()
id4 = c.increment.remote()
id5 = c.increment.remote()
ray.get([id4, id5]) # [1, 2]
```

API - Designed to Be Intuitive and Concise

Functions -> Tasks

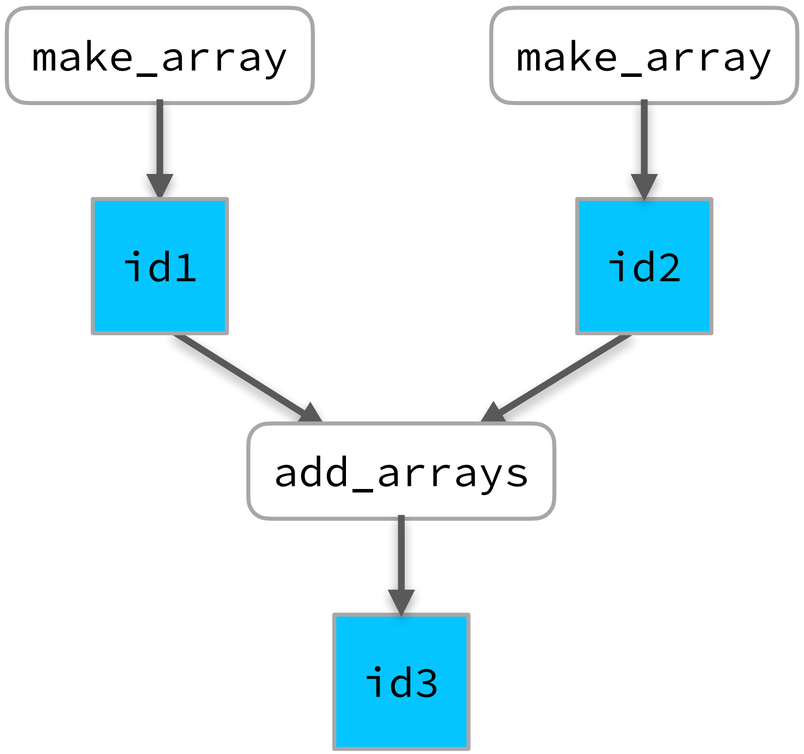
```
@ray.remote
def make_array(…):
    a = … # Construct a NumPy array
    return a

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

id1 = make_array.remote(…)
id2 = make_array.remote(…)
id3 = add_arrays.remote(id1, id2)
ray.get(id3)
```

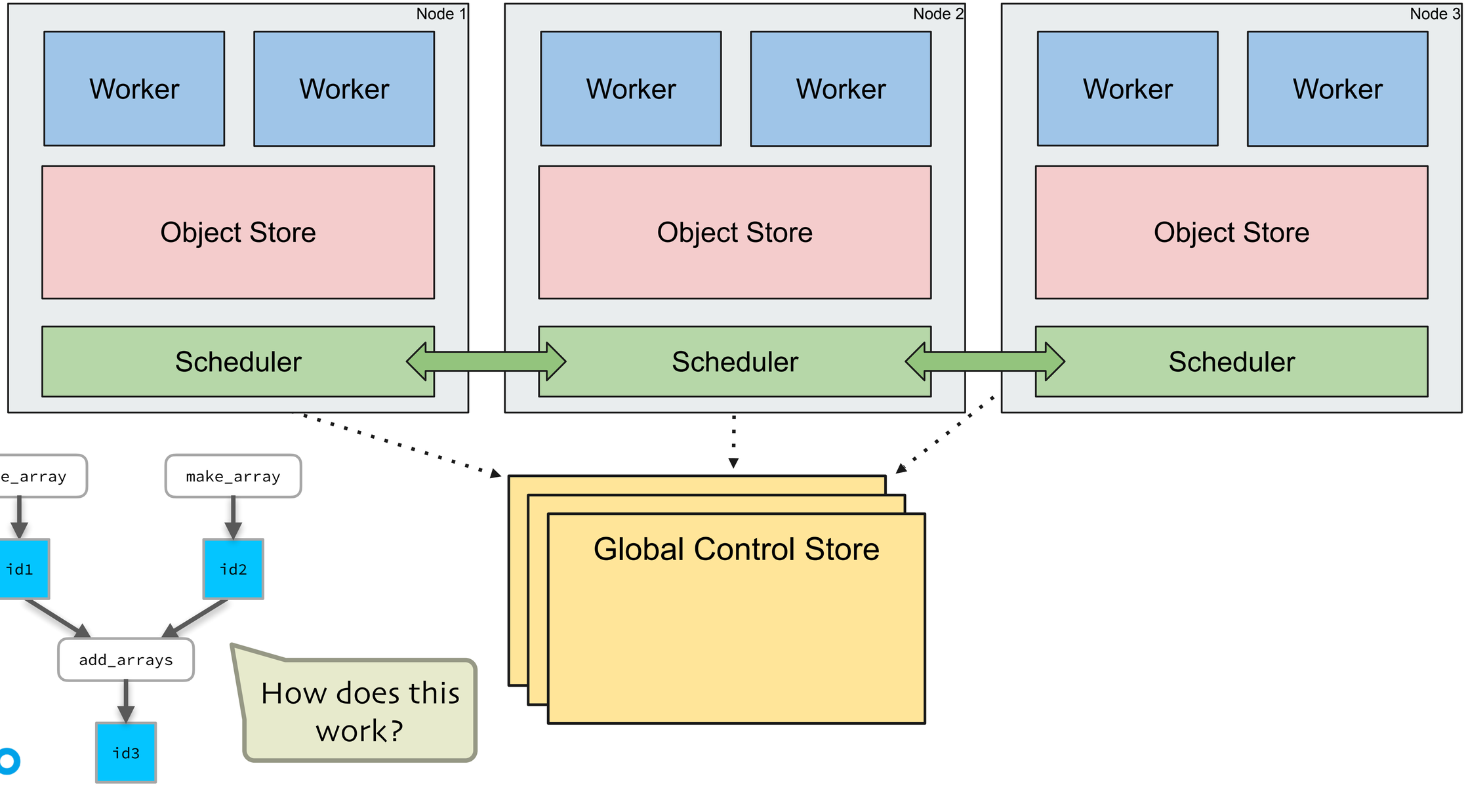
Classes -> Actors

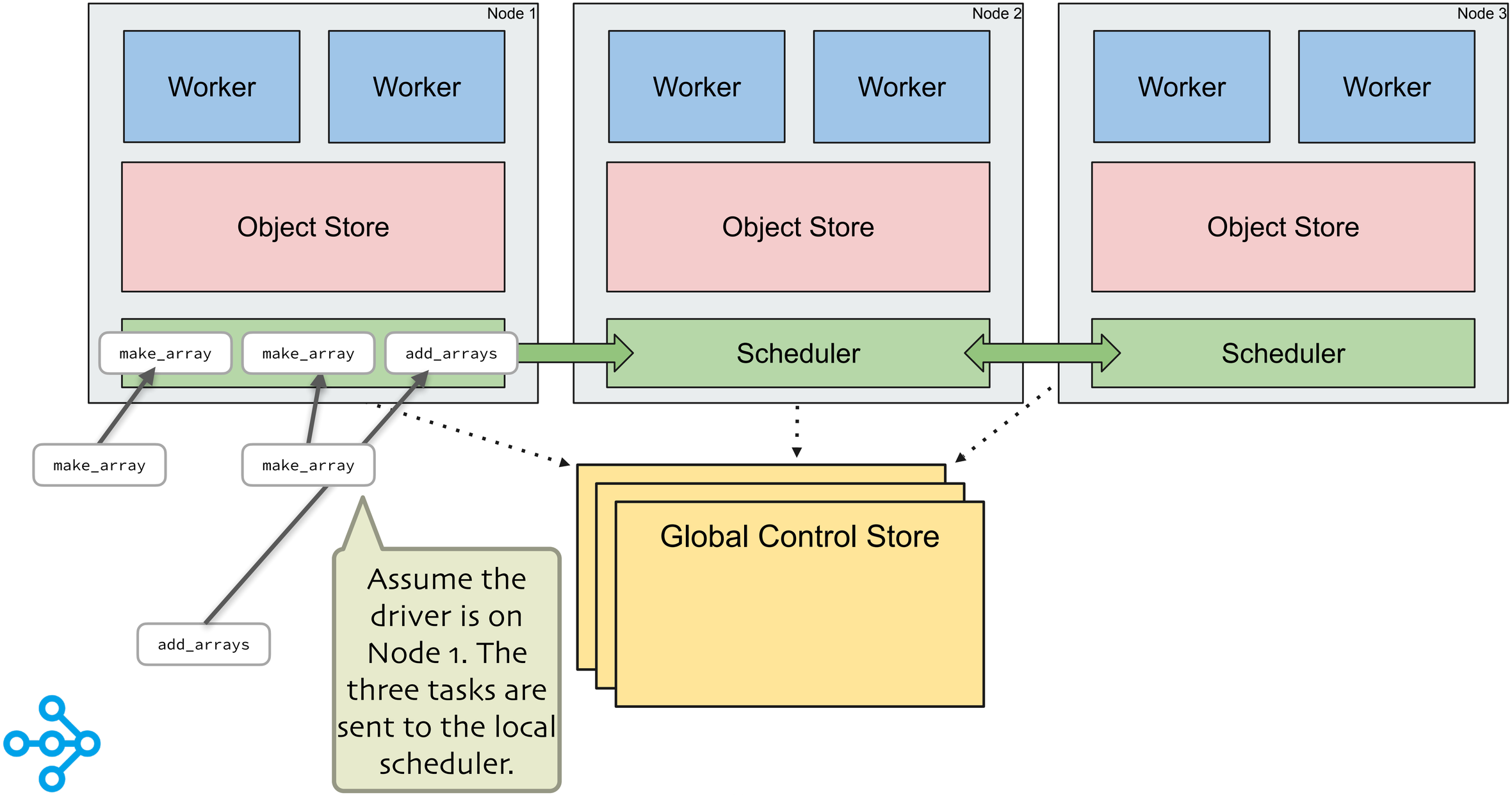
```
@ray.remote(num_gpus=1)
class Counter(object):
    def __init__(self):
        self.value = 0
    def increment(self):
        self.value += 1
        return self.value
    def get_count(self):
       return self.value

c = Counter.remote()
id4 = c.increment.remote()
id5 = c.increment.remote()
ray.get([id4, id5]) # [1, 2]
```

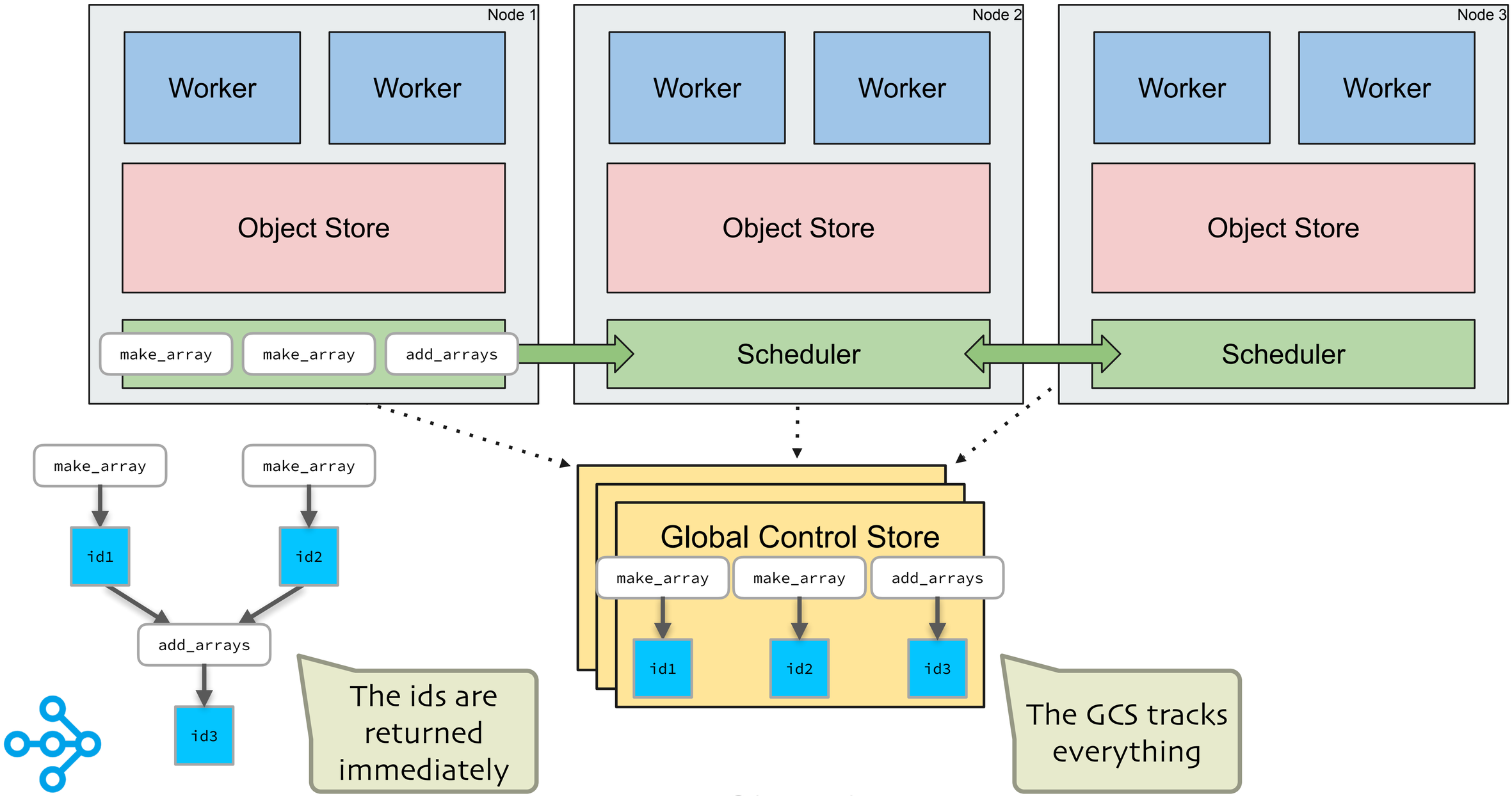
Optional configuration specifications

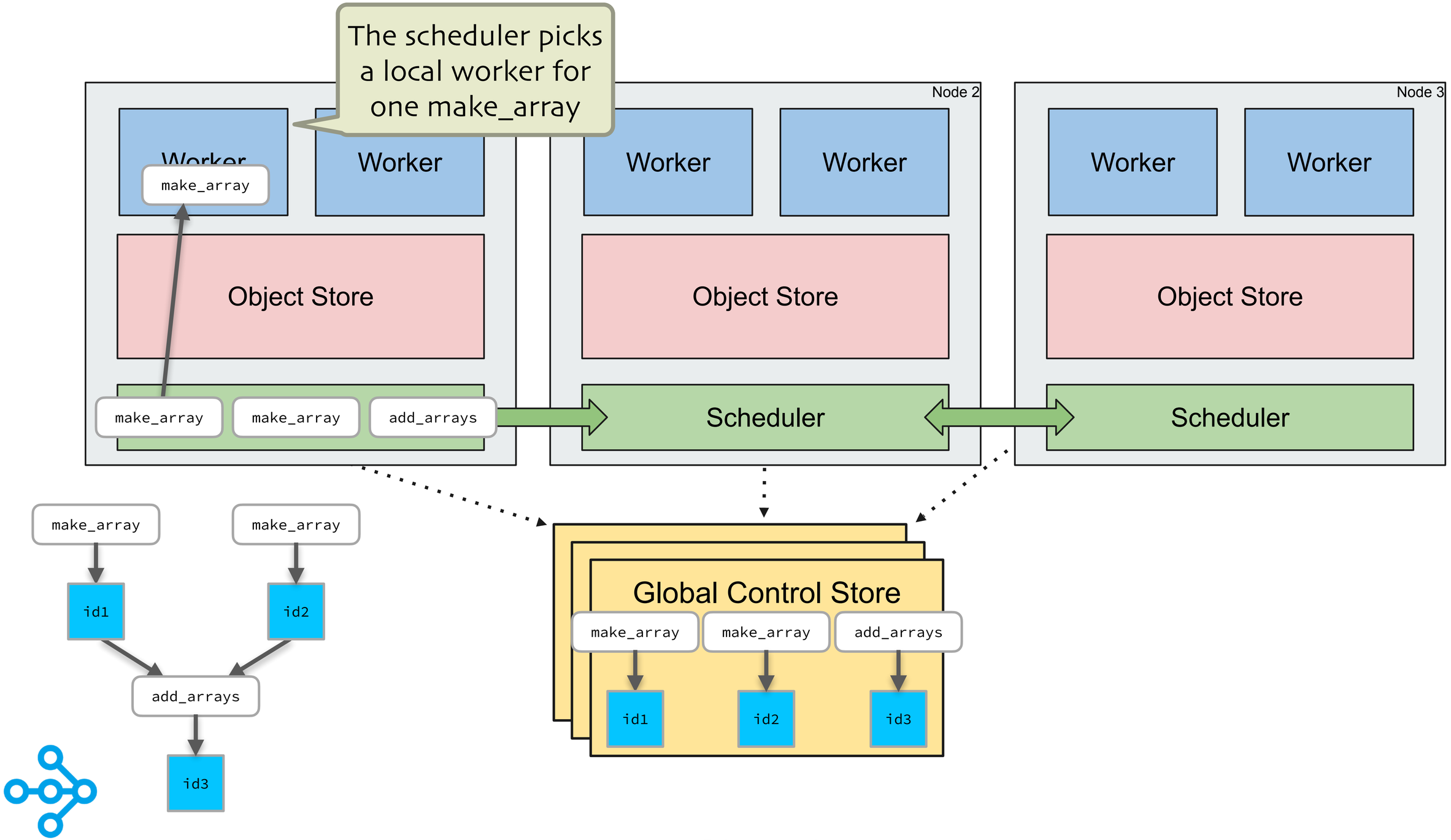
How Does This Work?

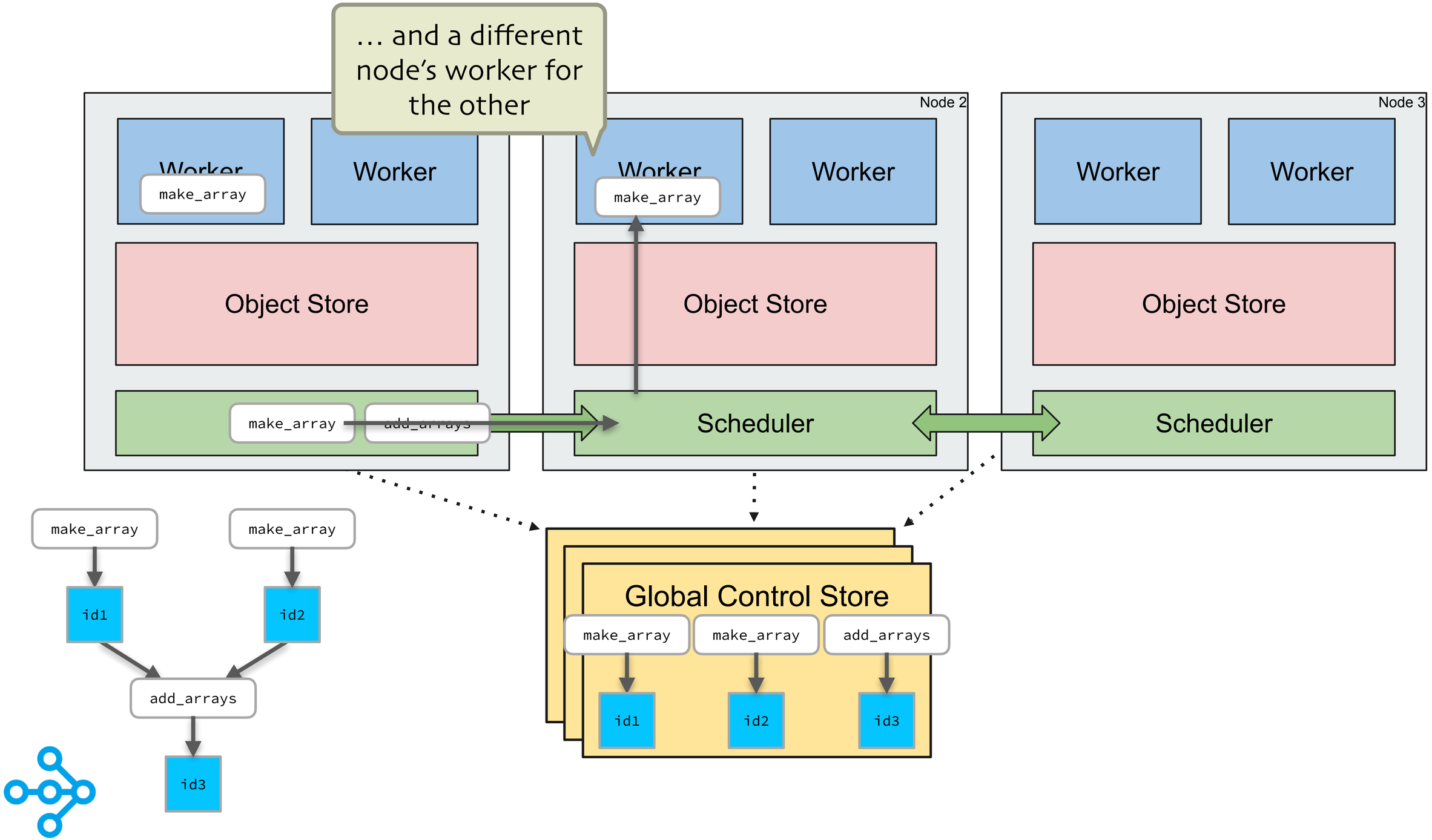



Node 1
Worker
Worker
Object Store
Scheduler
Node 2
Worker
Worker
Object Store
Scheduler
Node 3
Worker
Worker
Object Store
Scheduler
Global Control Store
make_array
make_array
id1
id2
add_arrays
id3
How does this work?


Node 1
Worker
Worker
Object Store
make_array
make_array
add_arrays
Node 2
Worker
Worker
Object Store
Scheduler
Node 3
Worker
Worker
Object Store
Scheduler
make_array
make_array
add_arrays
Global Control Store
Assume the driver is on Node 1. The three tasks are sent to the local scheduler.

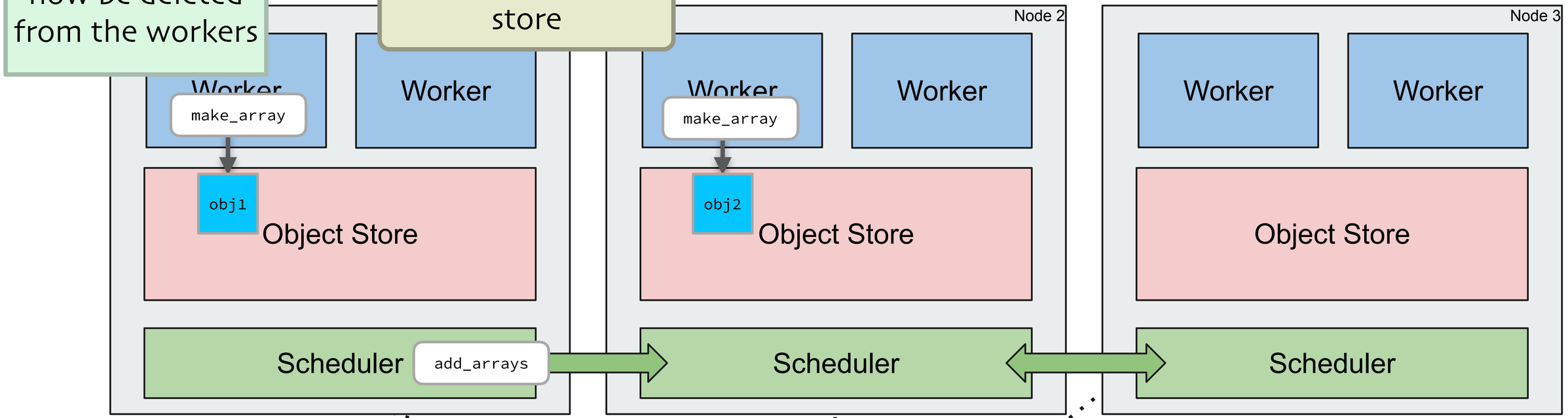

Node 1
Worker
Worker
Object Store
make_array
make_array
add_arrays
Node 2
Worker
Worker
Object Store
Scheduler
Node 3
Worker
Worker
Object Store
Scheduler
make_array
make_array
id1
id2
add_arrays
id3
Global Control Store
make_array
make_array
add_arrays
id1
id2
id3
The ids are returned immediately
The GCS tracks everything

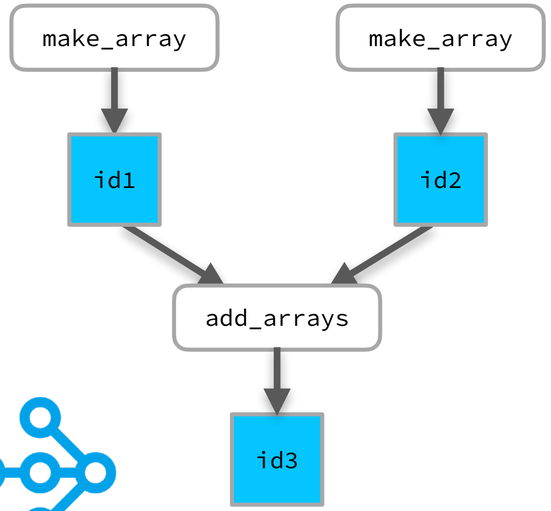

The scheduler picks a local worker for one make_array
Node 2
Node 3
Worker
Worker
Worker
Worker
Worker
Worker
make_array
Object Store
Object Store
Object Store
make_array
make_array
add_arrays
Scheduler
Scheduler
make_array
make_array
id1
id2
add_arrays
id3
Global Control Store
make_array
make_array
add_arrays
id1
id2
id3

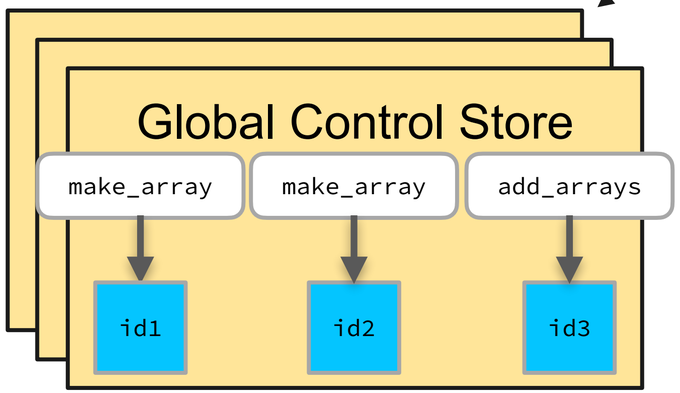

… and a different node's worker for the other
Node 2
Node 3
Worker
Worker
Worker
Worker
Worker
Worker
make_array
make_array
Object Store
Object Store
Object Store
make_array
add_arrays
Scheduler
Scheduler
make_array
make_array
id1
id2
add_arrays
id3
Global Control Store
make_array
make_array
add_arrays
id1
id2
id3

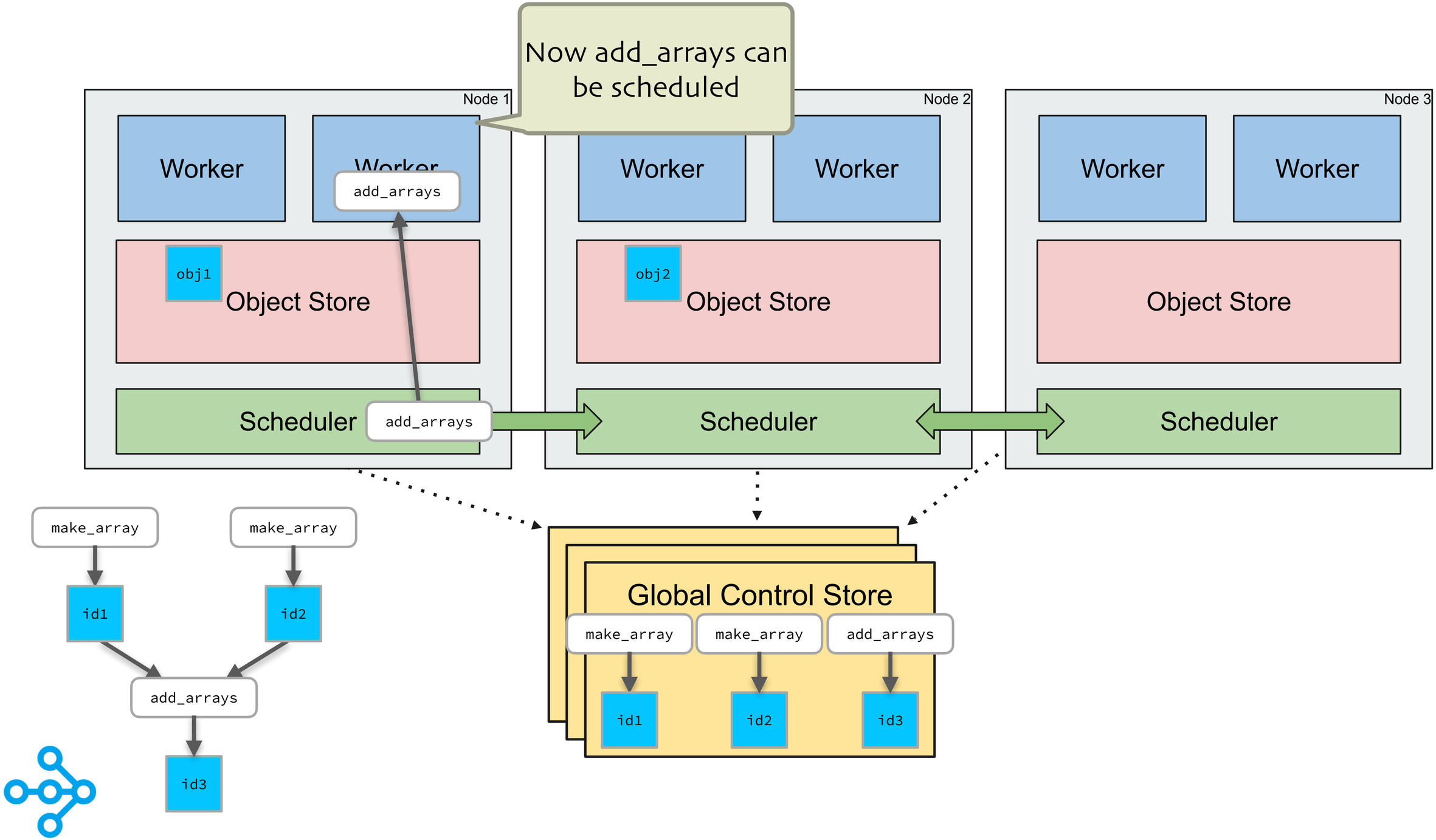
The tasks can now be deleted from the workers

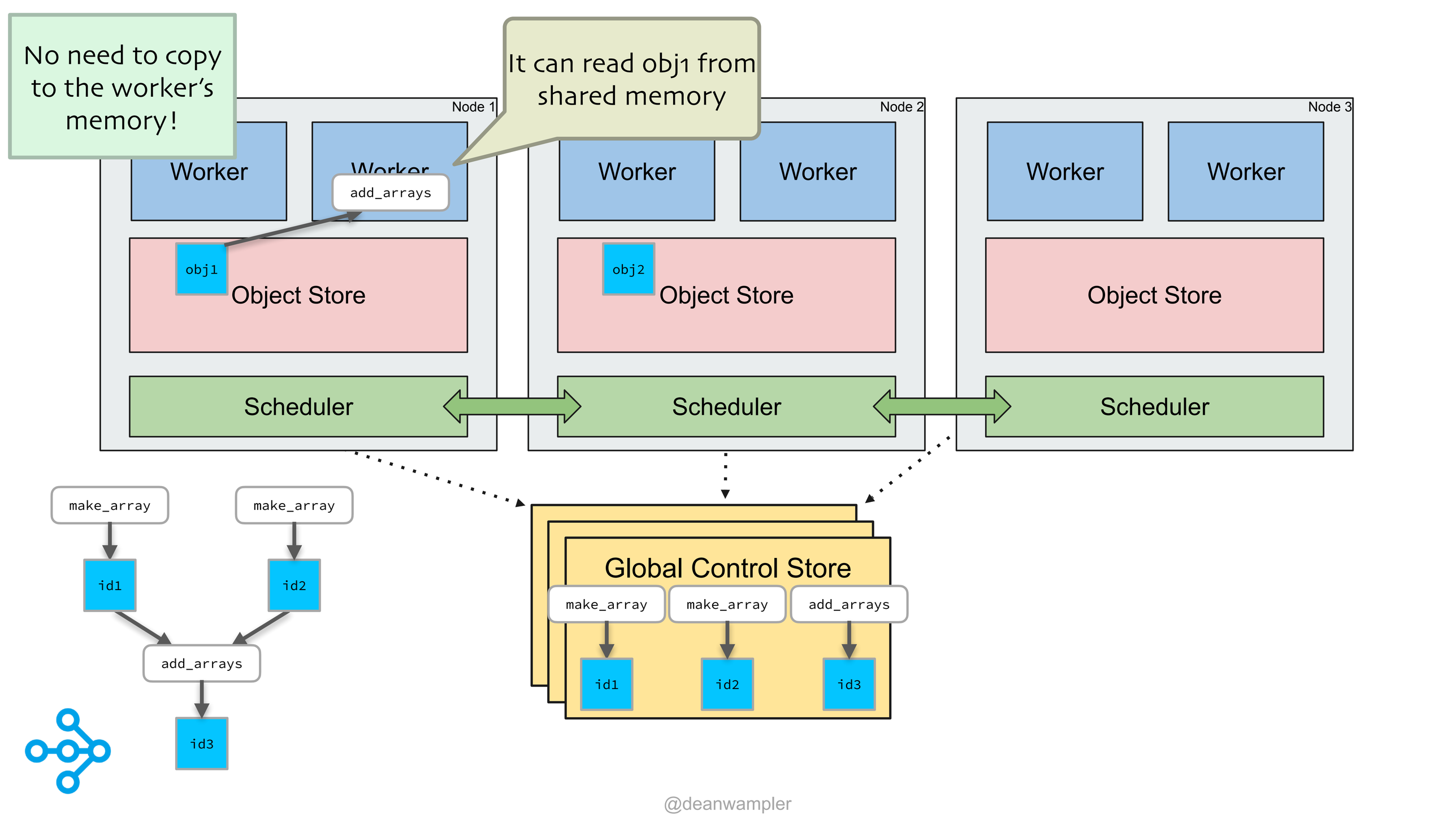
Each task writes its objects to the object store

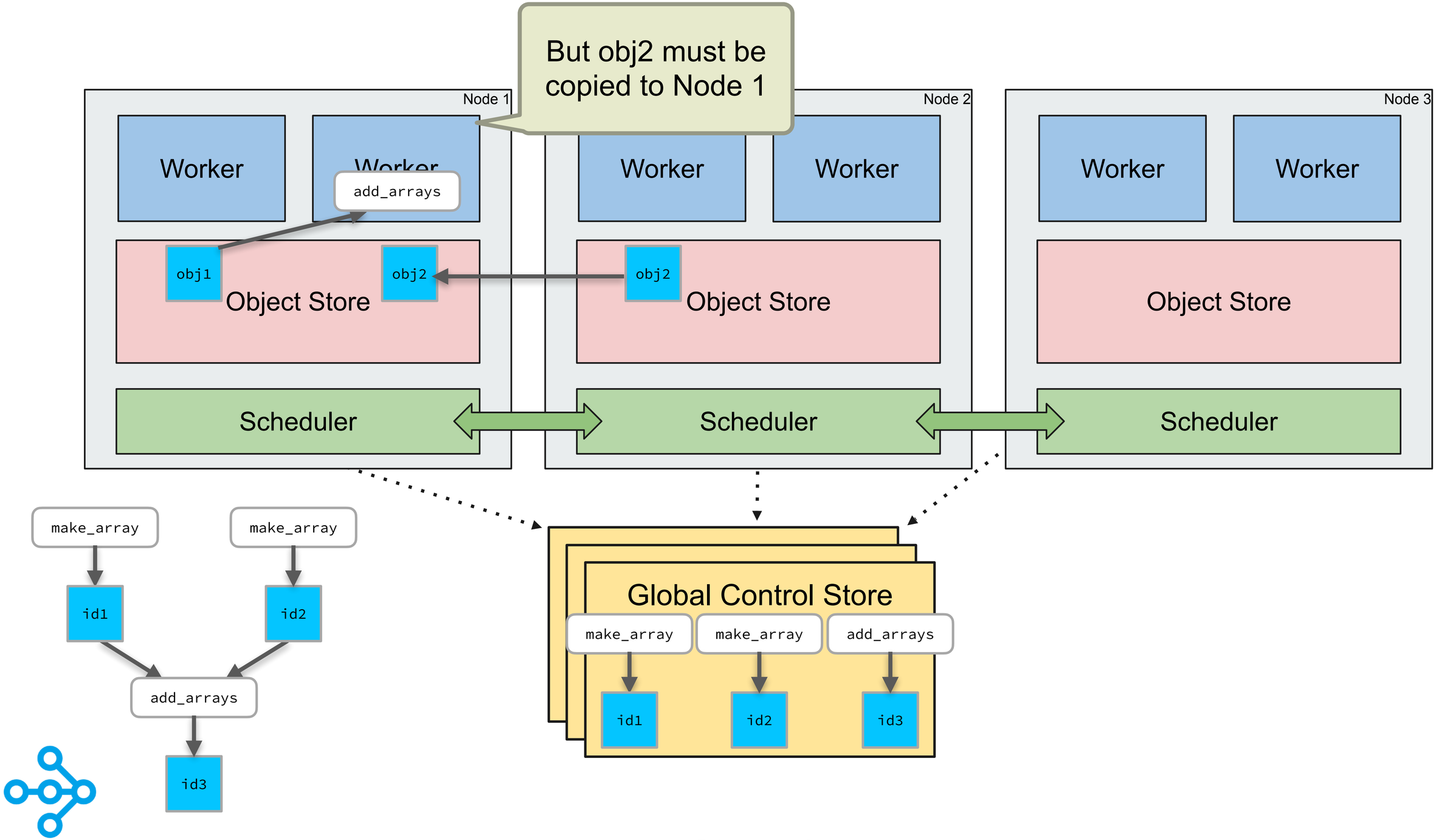


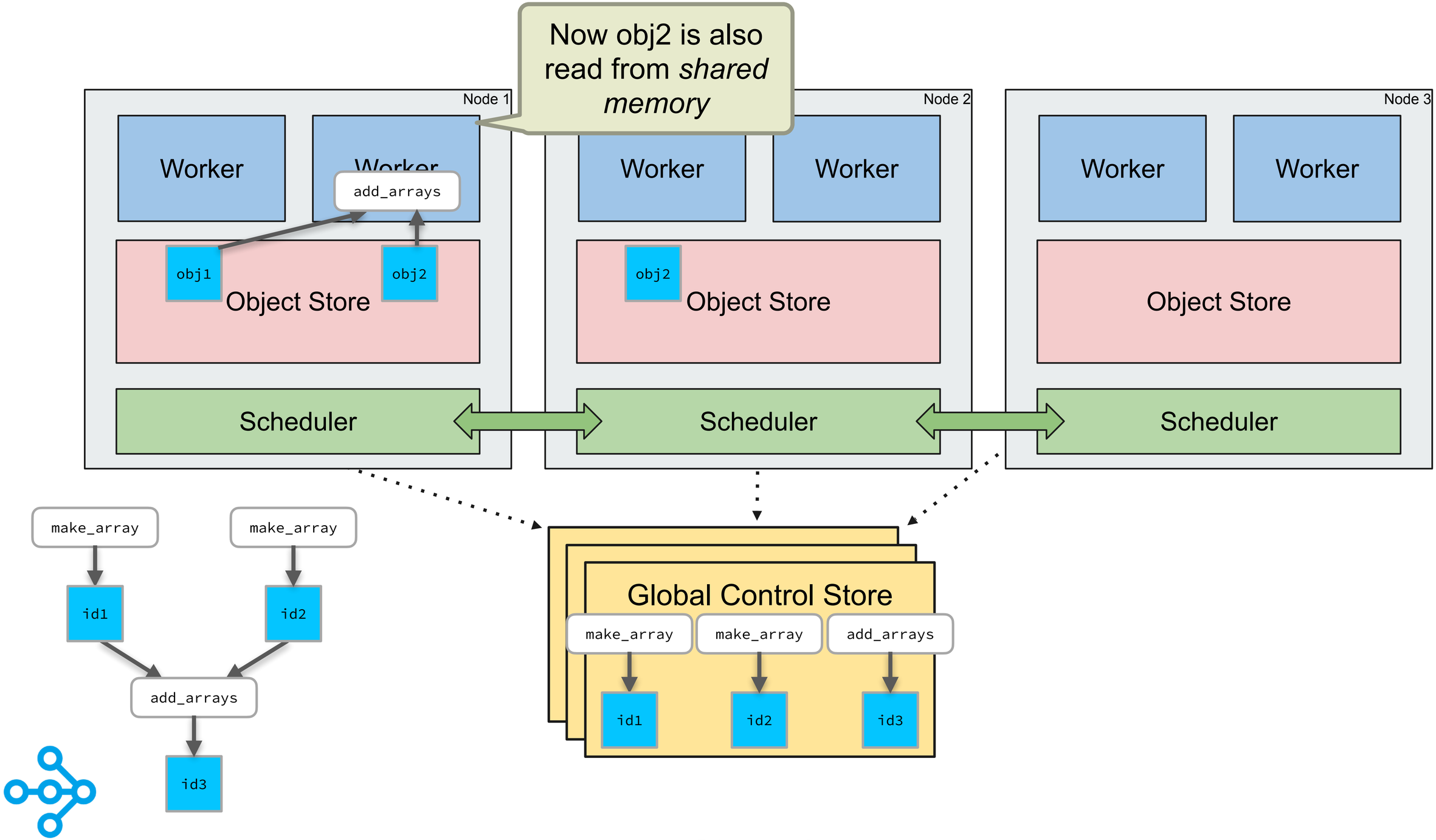


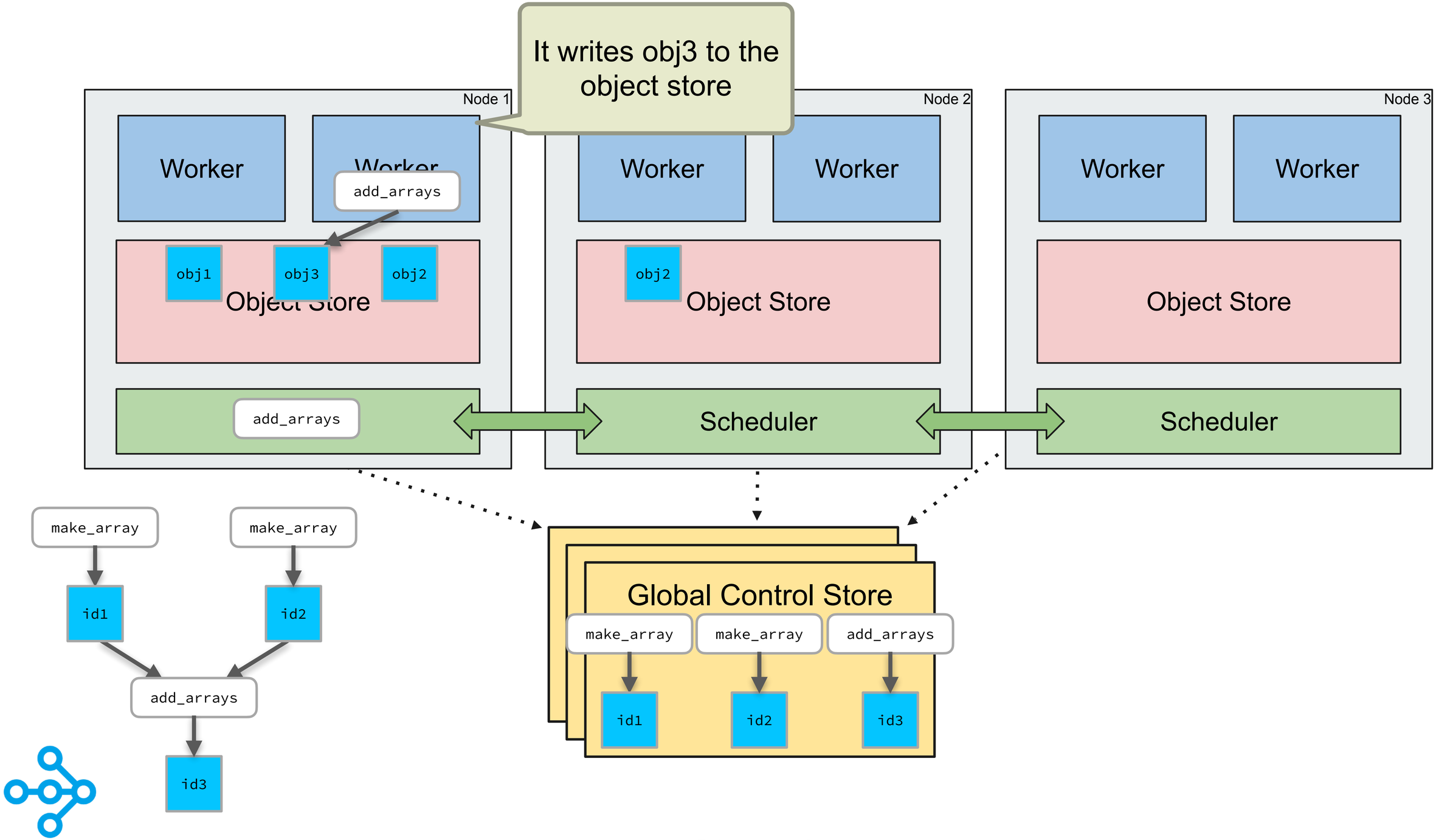


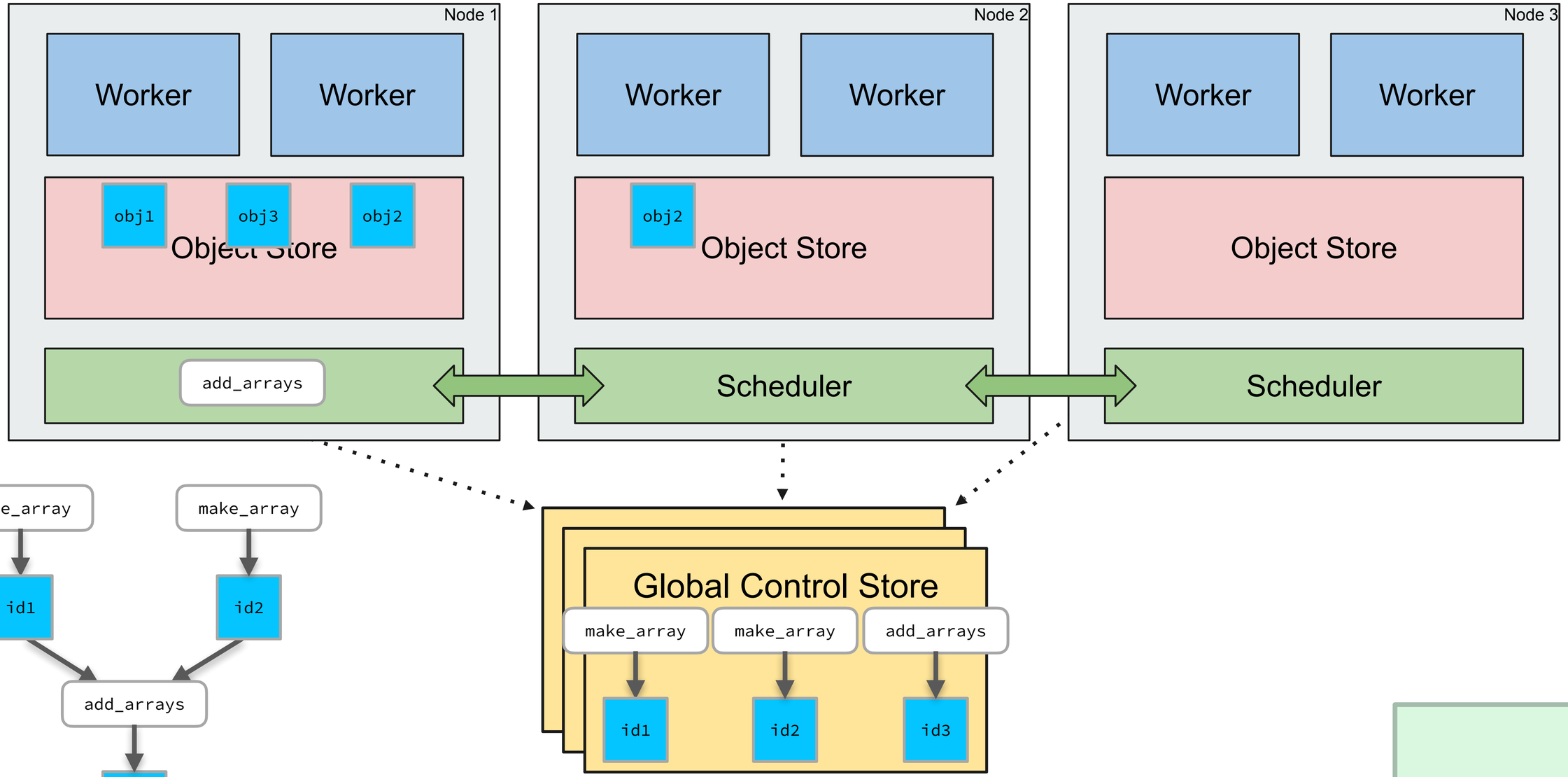

Now add_arrays can be scheduled
Node 1
Node 2
Node 3
Worker
Worker
Worker
Worker
Worker
add_arrays
obj1
Object Store
obj2
Object Store
Object Store
Scheduler
add_arrays
Scheduler
Scheduler
make_array
make_array
id1
id2
add_arrays
id3
Global Control Store
make_array
make_array
add_arrays
id1
id2
id3


No need to copy to the worker's memory!
It can read obj1 from shared memory
Node 1
Node 2
Node 3
Worker
Worker
Worker
Worker
Worker
Worker
add_arrays
obj1
obj2
Object Store
Object Store
Object Store
Scheduler
Scheduler
Scheduler
make_array
make_array
id1
id2
add_arrays
id3
Global Control Store
make_array
make_array
add_arrays
id1
id2
id3


But obj2 must be copied to Node 1
Node 1
Node 2
Node 3
Worker
Worker
Worker
Worker
Worker
Worker
add_arrays
obj1
obj2
obj2
Object Store
Object Store
Object Store
Scheduler
Scheduler
Scheduler
make_array
make_array
id1
id2
add_arrays
id3
Global Control Store
make_array
make_array
add_arrays
id1
id2
id3


Now obj2 is also read from *shared memory*
Node 1
Node 2
Node 3
Worker
Worker
Worker
Worker
Worker
Worker
add_arrays
obj1
obj2
Object Store
obj2
Object Store
Object Store
Scheduler
Scheduler
Scheduler
make_array
make_array
id1
id2
add_arrays
id3
Global Control Store
make_array
make_array
add_arrays
id1
id2
id3


It writes obj3 to the object store
Node 1
Node 2
Node 3
Worker
Worker
Worker
Worker
Worker
Worker
add_arrays
obj1
obj3
obj2
Object Store
obj2
Object Store
Object Store
add_arrays
Scheduler
Scheduler
make_array
make_array
id1
id2
add_arrays
id3
Global Control Store
make_array
make_array
add_arrays
id1
id2
id3




```
ray.get(id3)
returns obj3
```

And we're done!

Ray Community

Community and Resources

- ray.io
- ray.readthedocs.io/en/latest/
- Tutorials (free): Anyscale Academy
- github.com/ray-project/ray.git
- Need help?
  - Ray Slack: ray-distributed.slack.com
  - ray-dev group

Migrating to Ray

If you're already using…

- asyncio
- joblib
- multiprocessing.Pool

For example, from this:

```
from multiprocessing.pool import Pool
```

To this:

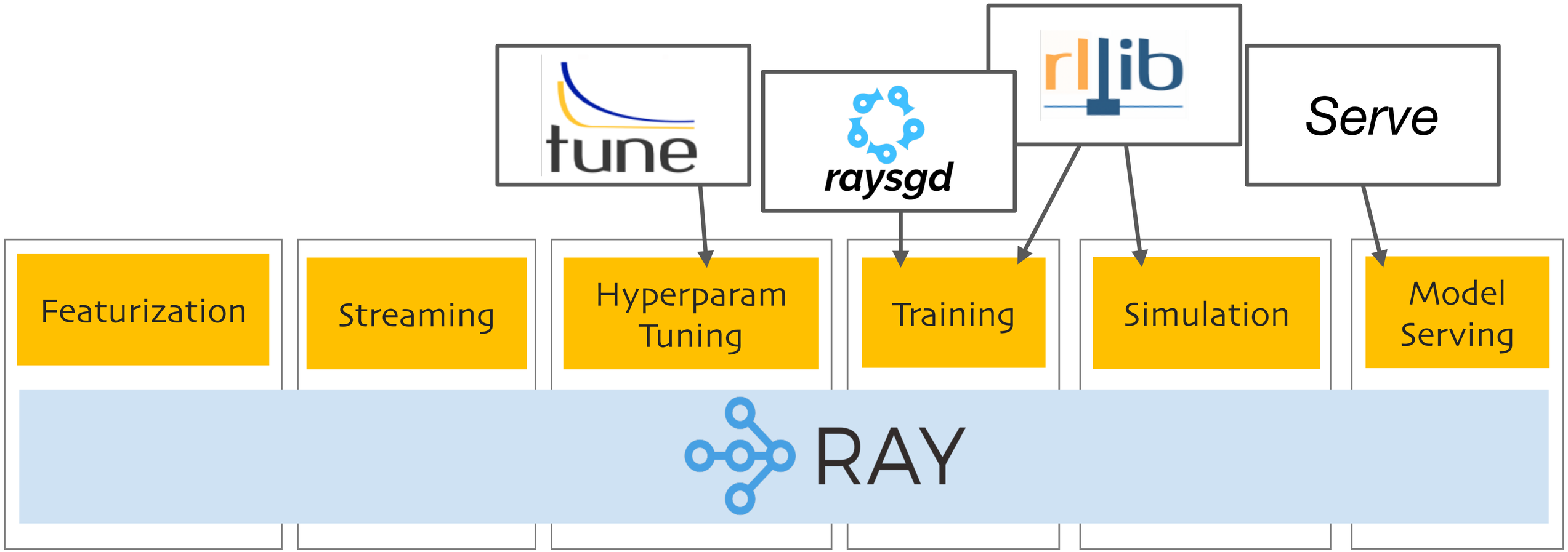
```
from ray.util.multiprocessing.pool import Pool
```

- Use Ray's implementations
  - Drop-in replacements
  - Change import statements
  - Break the one-node limitation!

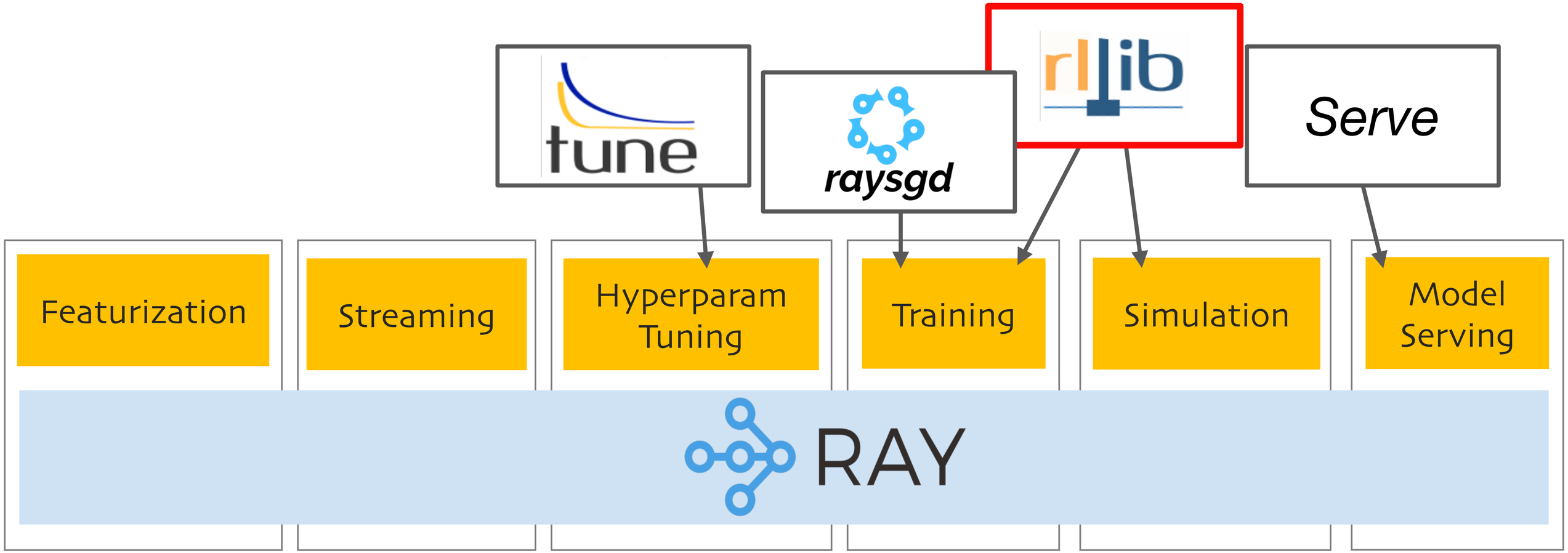
See these blog posts:
https://medium.com/distributed-computing-with-ray/how-to-scale-python-multiprocessing-to-a-cluster-with-one-line-of-code-d19f242f60ff
https://medium.com/distributed-computing-with-ray/easy-distributed-scikit-learn-training-with-ray-54ff8b643b33

Machine Learning with Ray-based Libraries

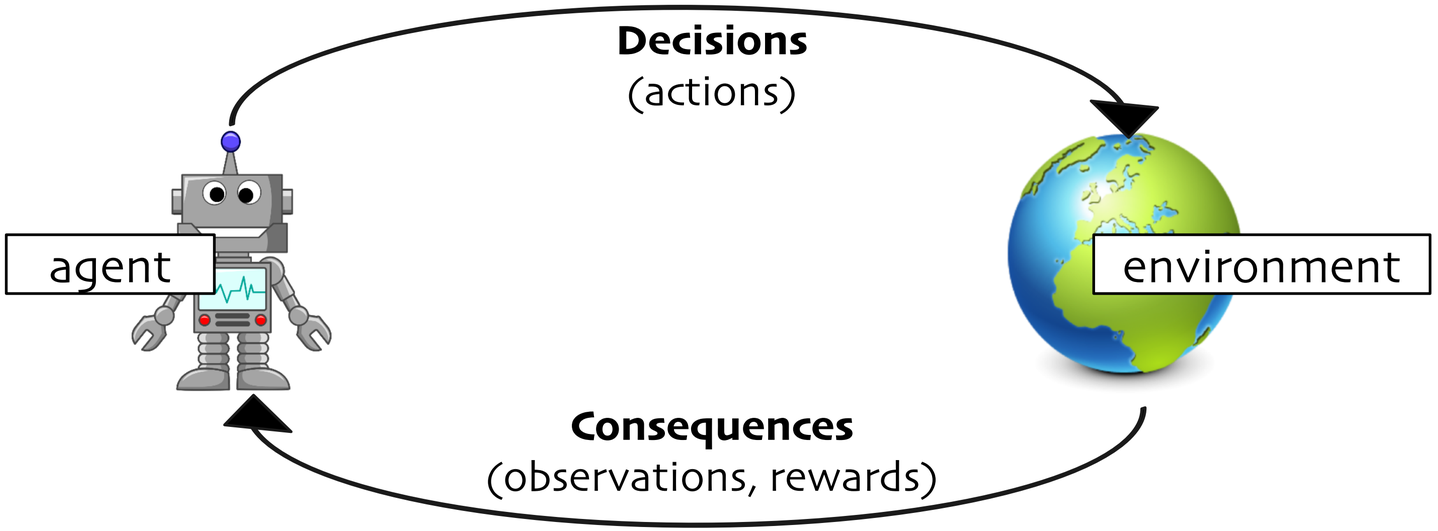
Ray Libraries

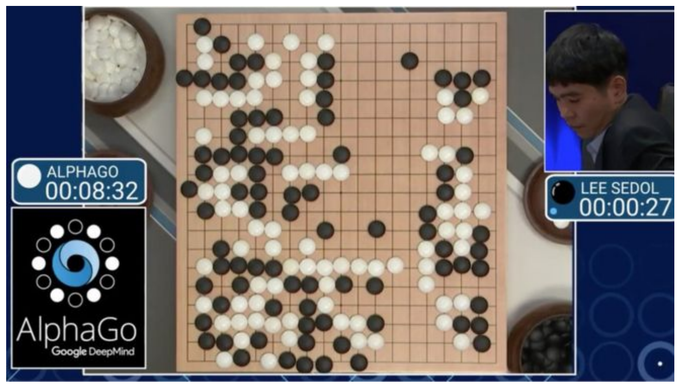


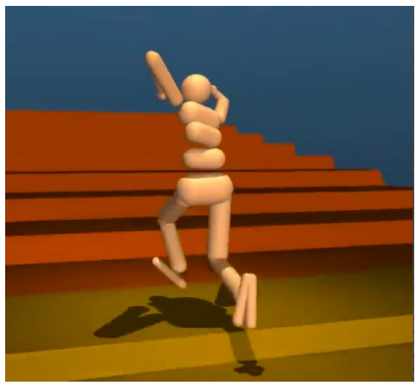
Reinforcement Learning - Ray RLlib



Background: Reinforcement Learning



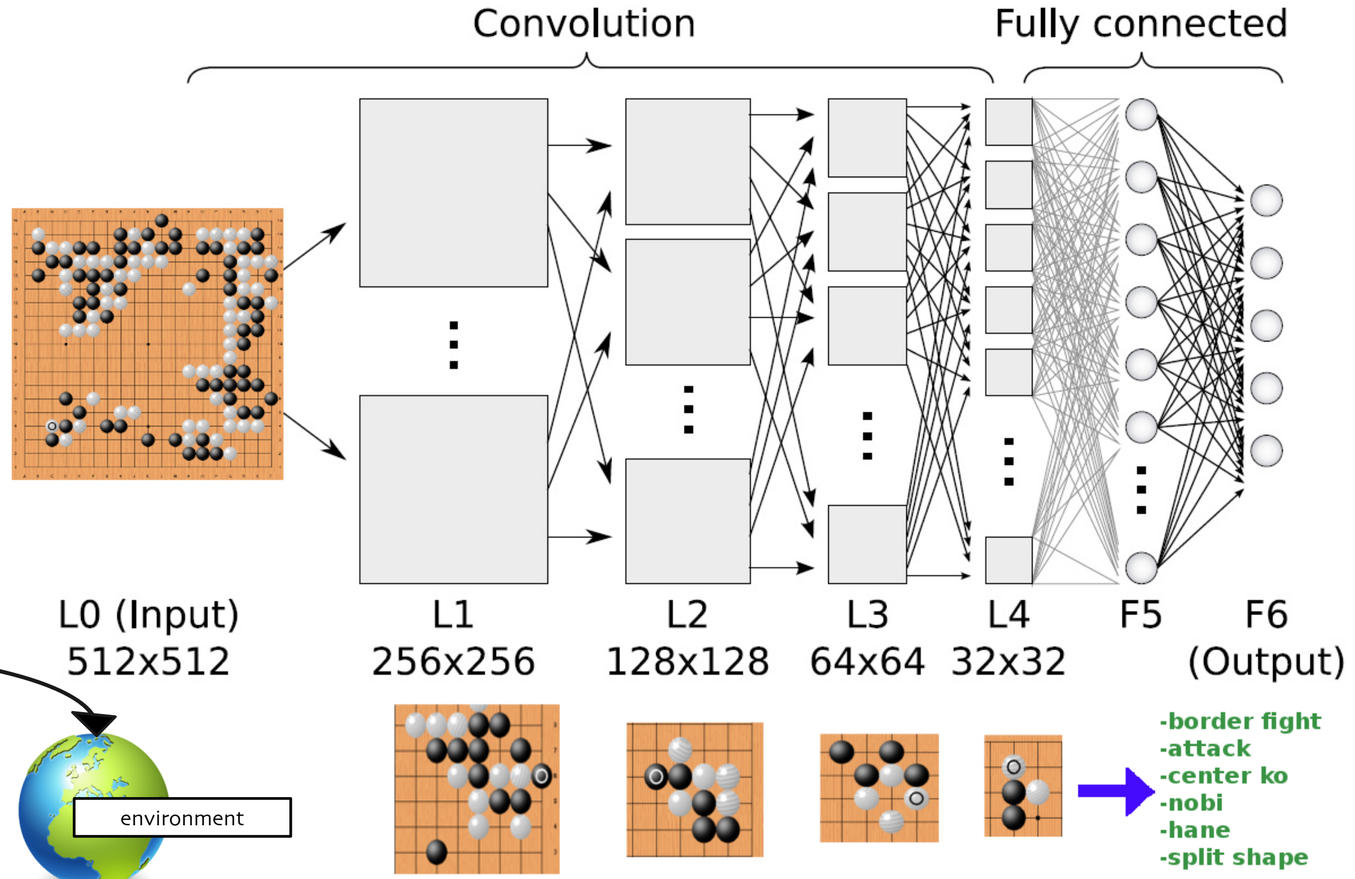


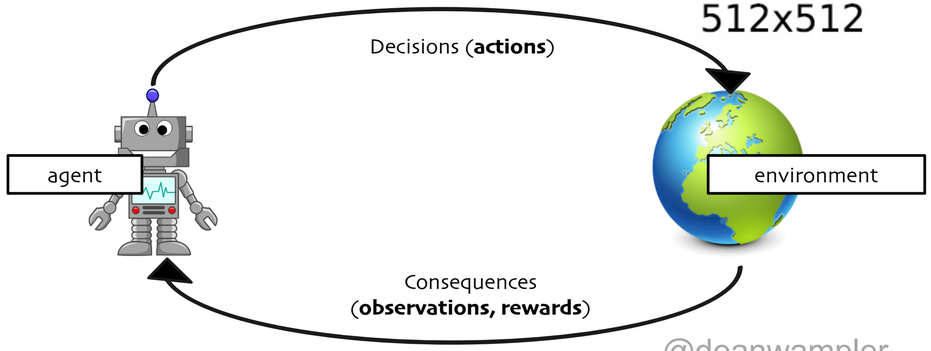


Go as a Reinforcement Learning Problem

AlphaGo (Silver et al. 2016)

- **Observations**:
  - board state
- **Actions**:
  - where to place the stones
- **Rewards**:
  - 1 if win
  - 0 otherwise





Growing Number of RL Applications

Industrial Processes

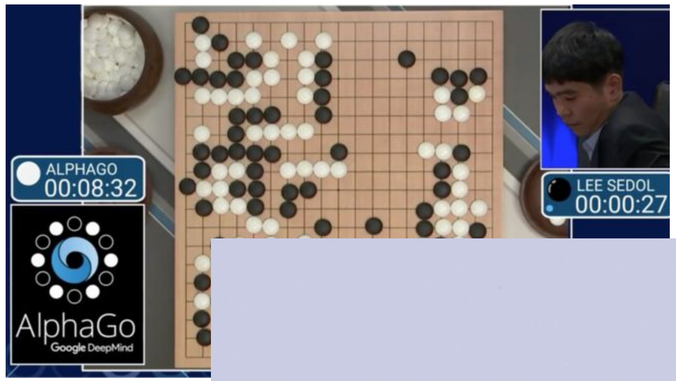
System Optimization

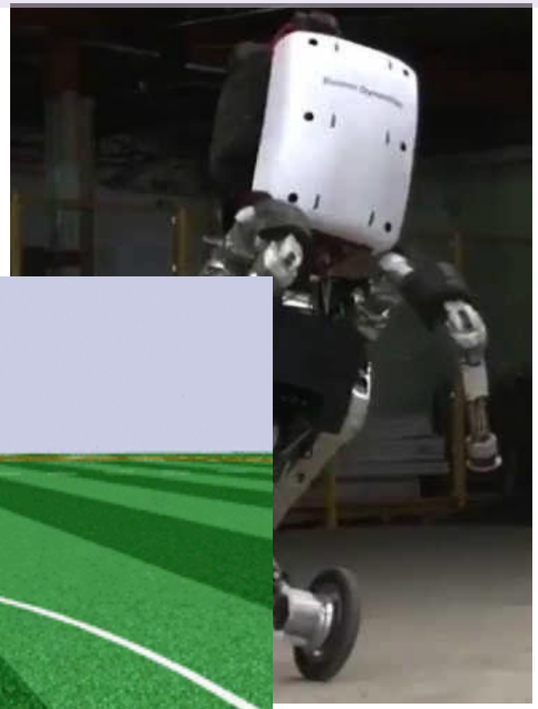
Advertising

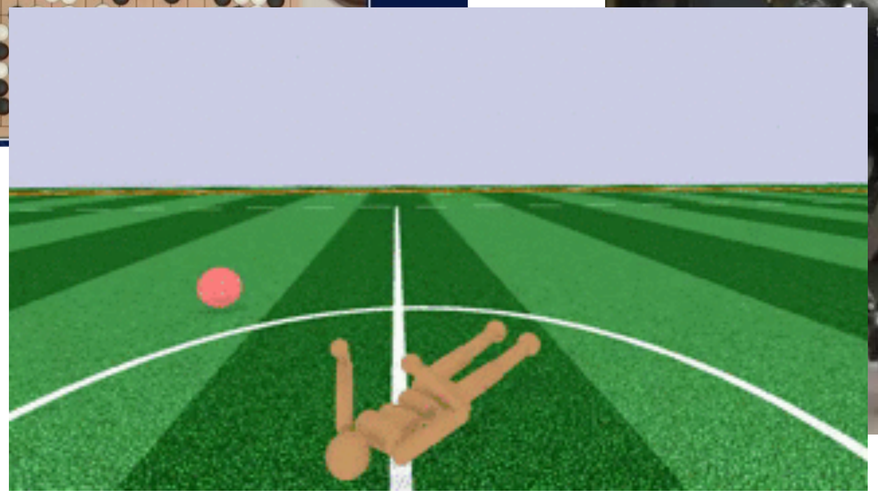
Recommendations

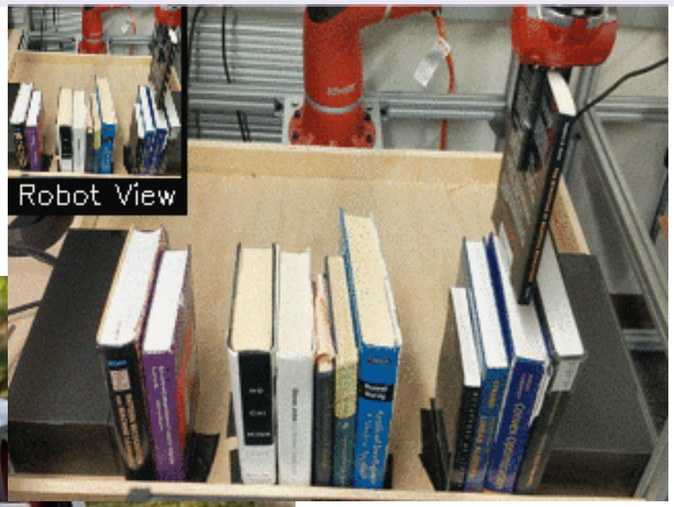
Finance

RL applications

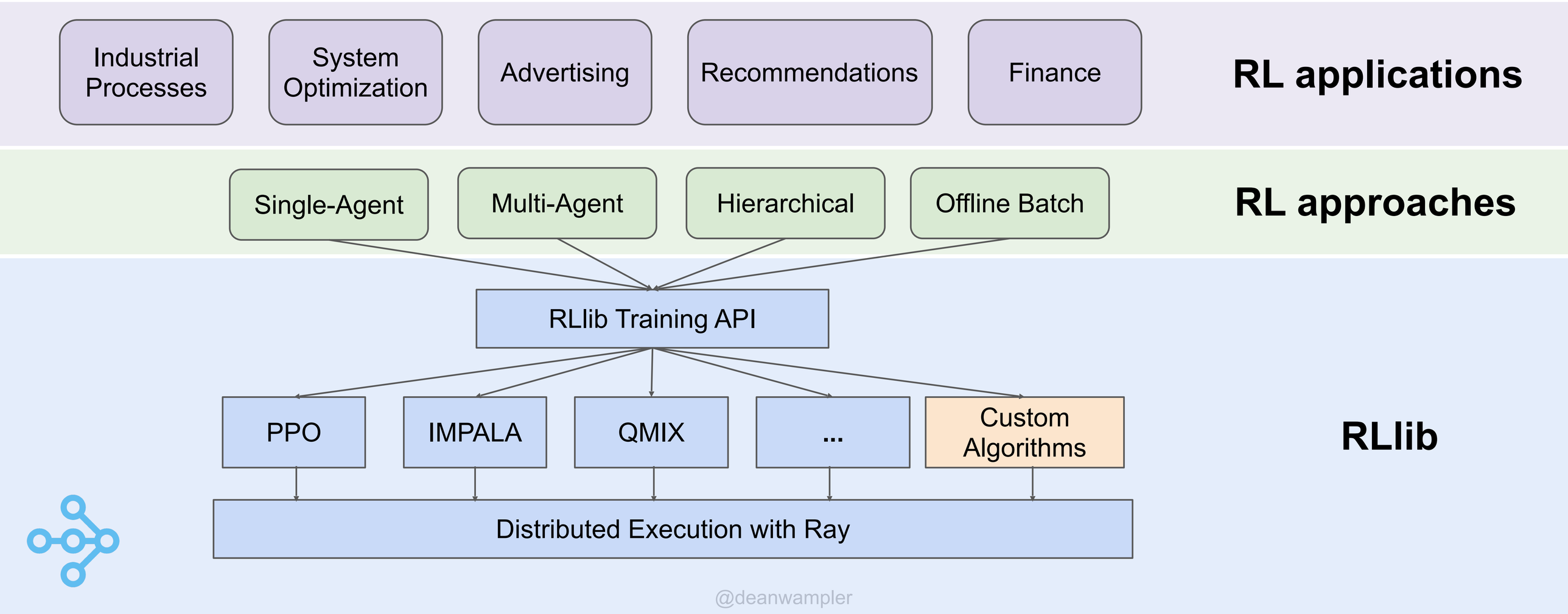








RLlib: A Scalable, Unified Library for RL



Broad Range of Scalable Algorithms

- High-throughput architectures
  - Distributed Prioritized Experience Replay (Ape-X)
  - Importance Weighted Actor-Learner Architecture (IMPALA)
  - Asynchronous Proximal Policy Optimization (APPO)

- Gradient-based
  - Soft Actor-Critic (SAC)
  - Advantage Actor-Critic (A2C, A3C)
  - Deep Deterministic Policy Gradients (DDPG, TD3)
  - Deep Q Networks (DQN, Rainbow, Parametric DQN)
  - Policy Gradients
  - Proximal Policy Optimization (PPO)

- gradient-free
  - Augmented Random Search (ARS)
  - Evolution Strategies

- Multi-agent specific
  - QMIX Monotonic Value Factorisation (QMIX, VDN, IQN)

- Offline
  - Advantage Re-Weighted Imitation Learning (MARWIL)

Amazon SageMaker RL

End-to-end examples for classic RL and real-world RL applications

Robotics | Industrial Control | HVAC | Autonomous Vehicles | Operations | Finance | Games | NLP

RL Environments to model real-world problems

- **AWS Simulation Environments**: Amazon Sumerian | AWS RoboMaker
- **Open Source Environments**: EnergyPlus | RoboSchool | PyBullet | ...
- **Custom Environments**: Bring Your Own
- **Commercial simulators**: MATLAB & Simulink

Open AI Gym

RL Toolkits that provide RL agent algorithm implementations

- **RL-Coach**: DQN | PPO | HER | Rainbow | ...
- **RL-Ray RLLib**: APEX | ES | IMPALA | A3C | ...
- **Open AI Baselines**: TRPO | GAIL | ...
- ...

SageMaker Deep Learning Frameworks

TensorFlow | MxNet | PyTorch | Chainer

Training Options

Single Machine / Distributed | Local / Remote simulation | CPU / GPU Hardware

SageMaker supported | Customer BYO

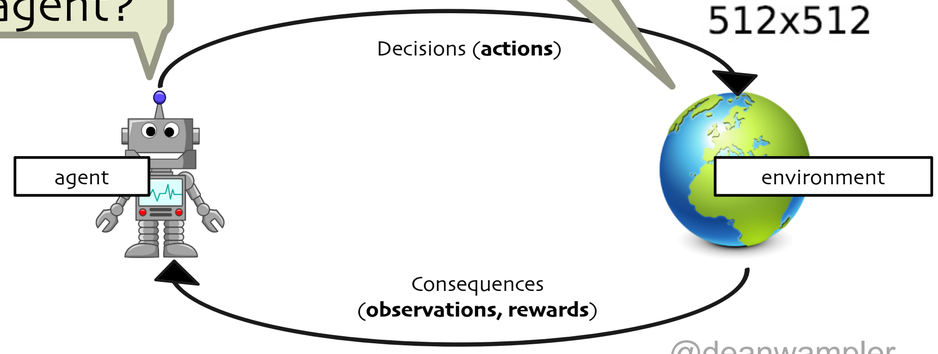
Diverse Compute Requirements Motivated Creation of Ray!

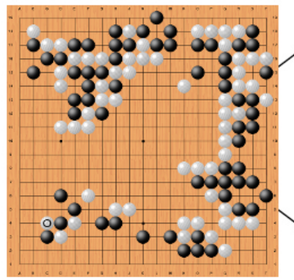
And repeated play, over and over again, to train for achieving the best reward

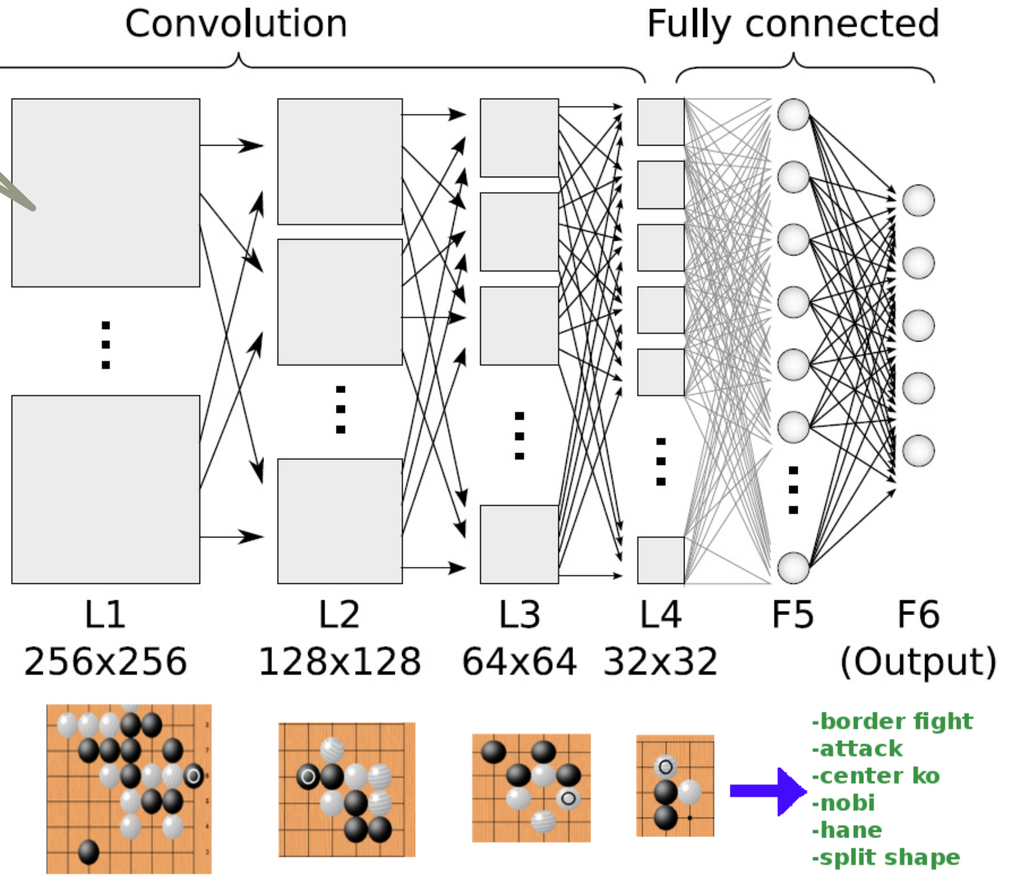
Neural network "stuff"

Simulator (game engine, robot sim, factory floor sim…)

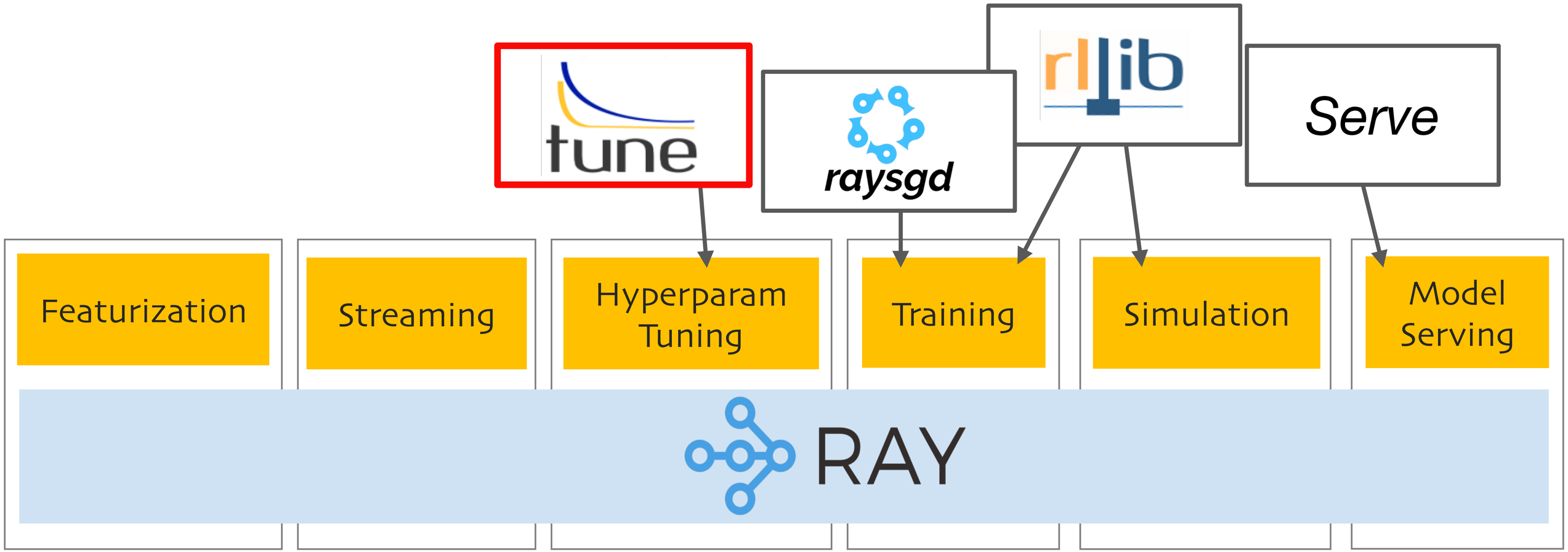
Complex agent?







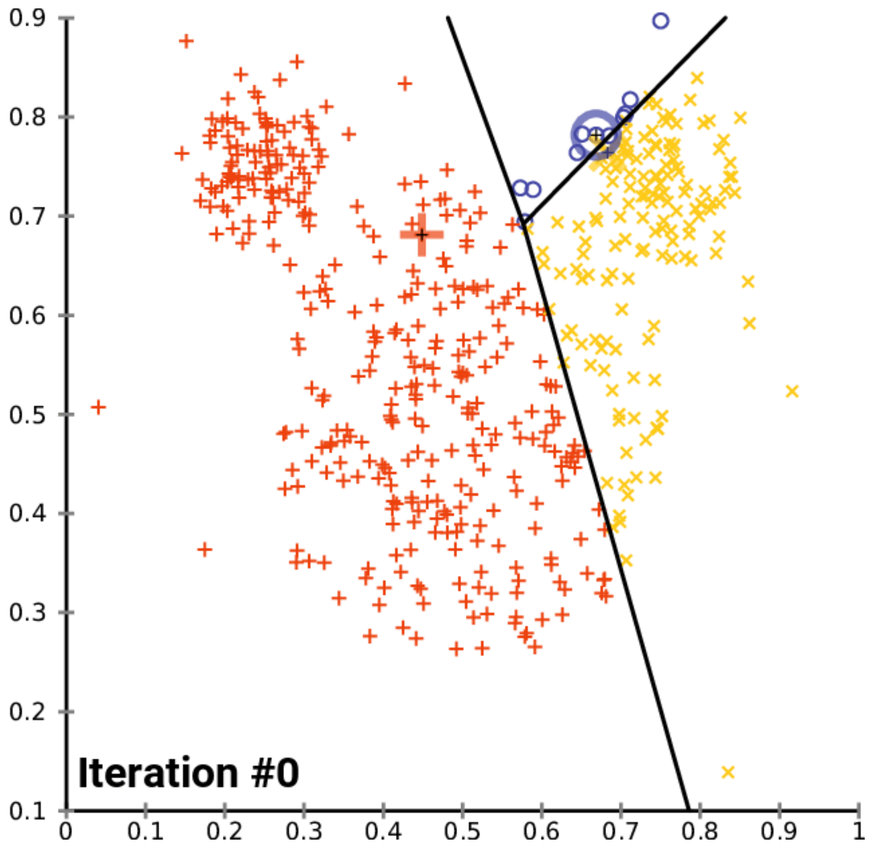
Hyperparameter Tuning - Ray Tune



What Is Hyperparameter Tuning?

Trivial example:

- What's the best value for "k" in k-means??
  - k is a "hyperparameter"
    - The resulting clusters are defined by "parameters"

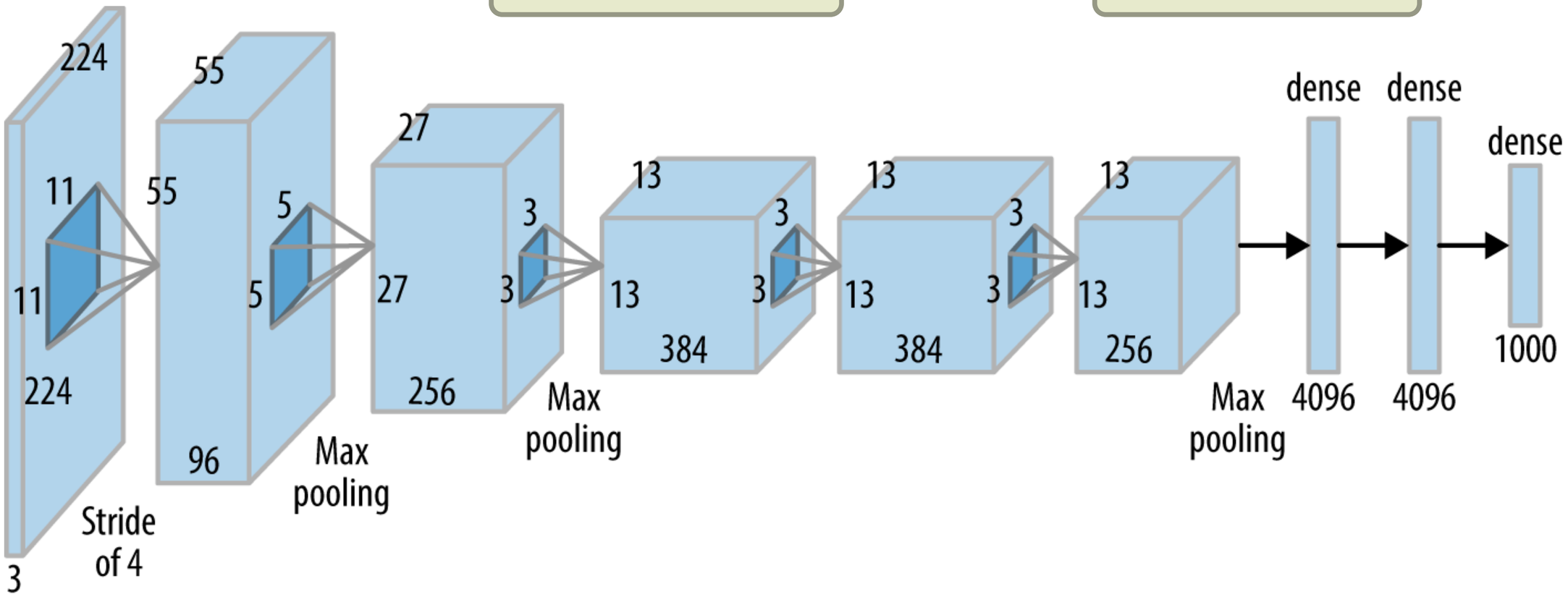


Source: https://commons.wikimedia.org/wiki/File:K-means_convergence.gif

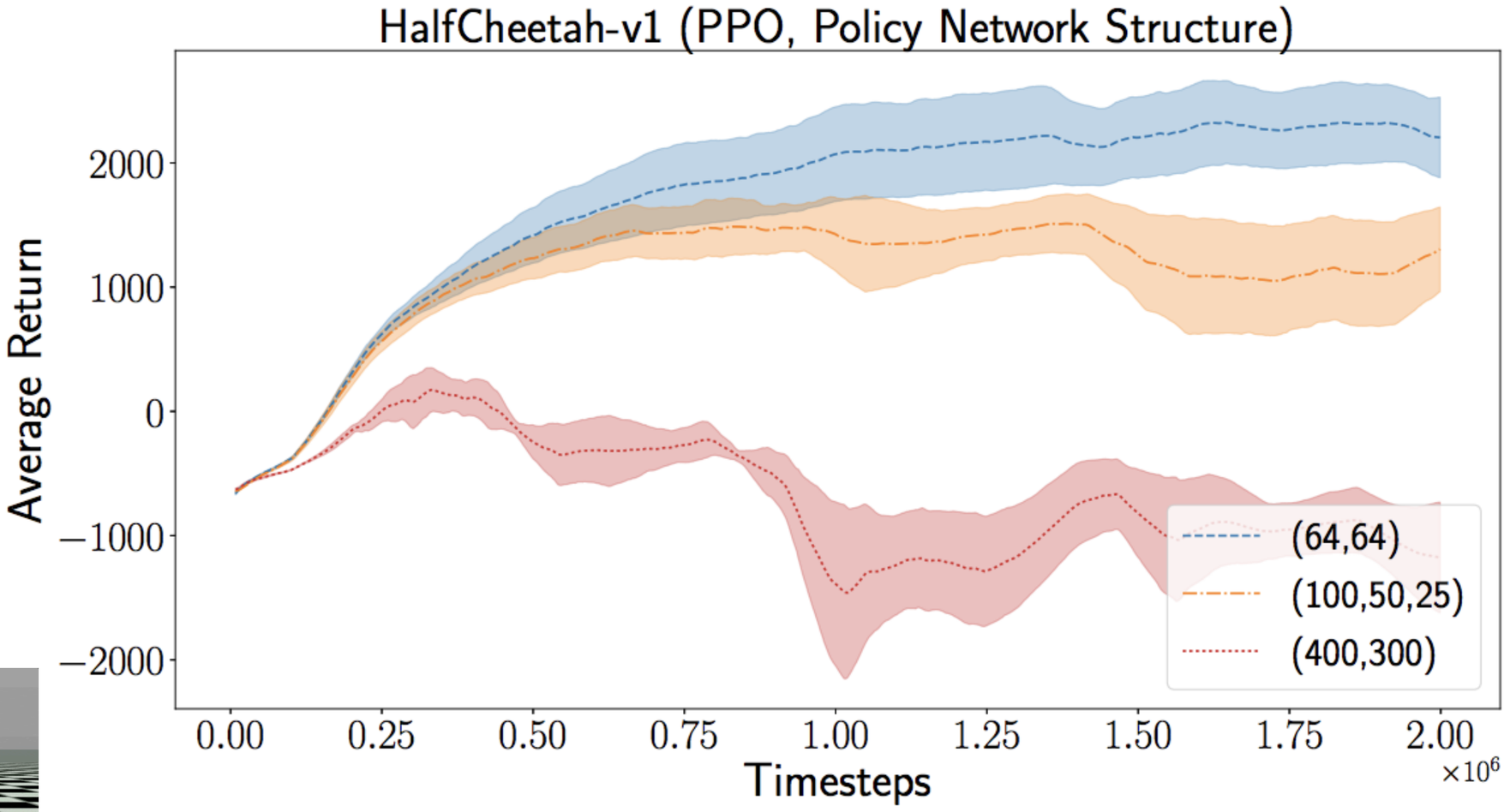
Nontrivial Example - Neural Networks

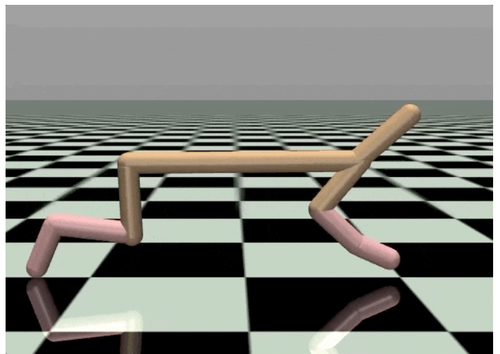
How many layers?
What kinds of layers?

Every number shown is a hyperparameter!



Hyperparameters Are Important for Performance



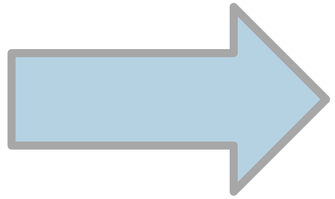


Why We Need a Framework for Tuning Hyperparameters

We want the best model

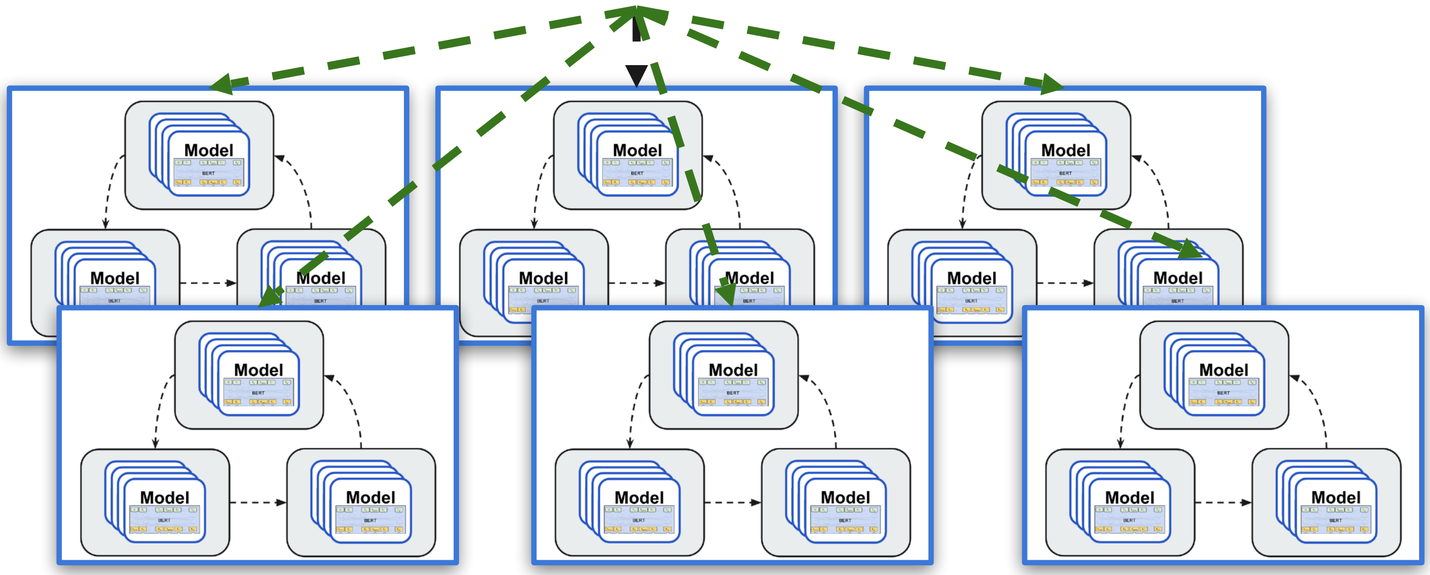
Resources are expensive

Model training is time-consuming

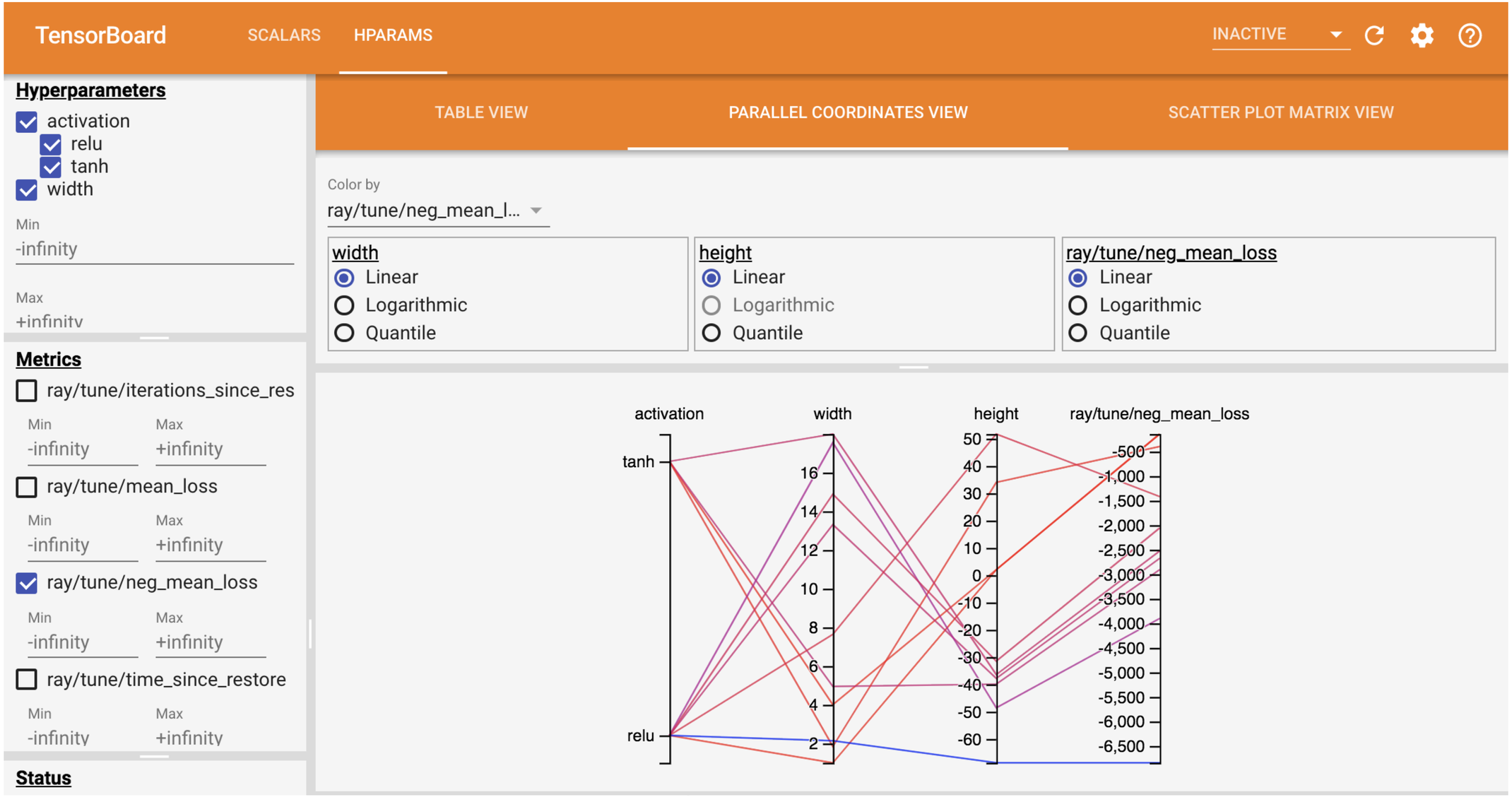


Tuning + Distributed Training

```
tune.run(PytorchTrainable,
        config={
            "model_creator": PretrainBERT,
            "data_creator": create_data_loader,
            "use_gpu": True,
            "num_replicas": 8,
            "lr": tune.uniform(0.001, 0.1)
        },
        num_samples=100,
        search_alg=BayesianOptimization()
)
```



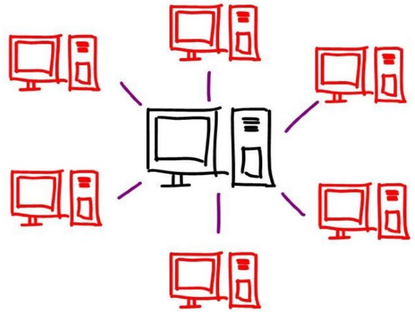
Native Integration with TensorBoard HParams



Tune is Built with Deep Learning as a Priority

Resource Aware Scheduling

Seamless Distributed Execution



Simple API for new algorithms

```
class TrialScheduler:
    def on_result(self, trial, result): ...
    def choose_trial_to_run(self): ...
```

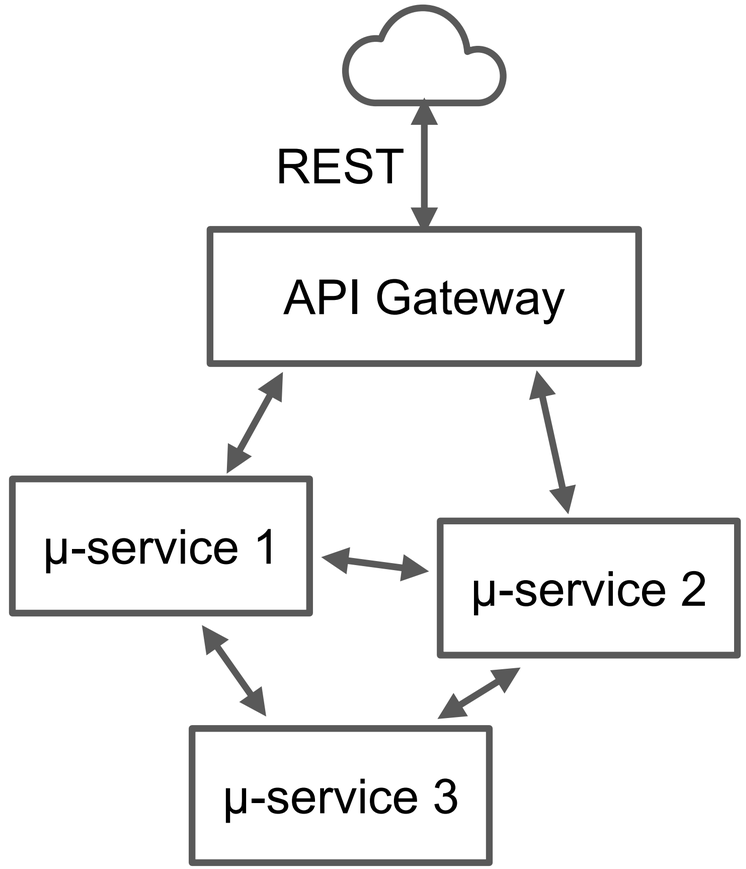
Framework Agnostic

ray.readthedocs.io/en/latest/tune.html

What about Ray for Microservices?

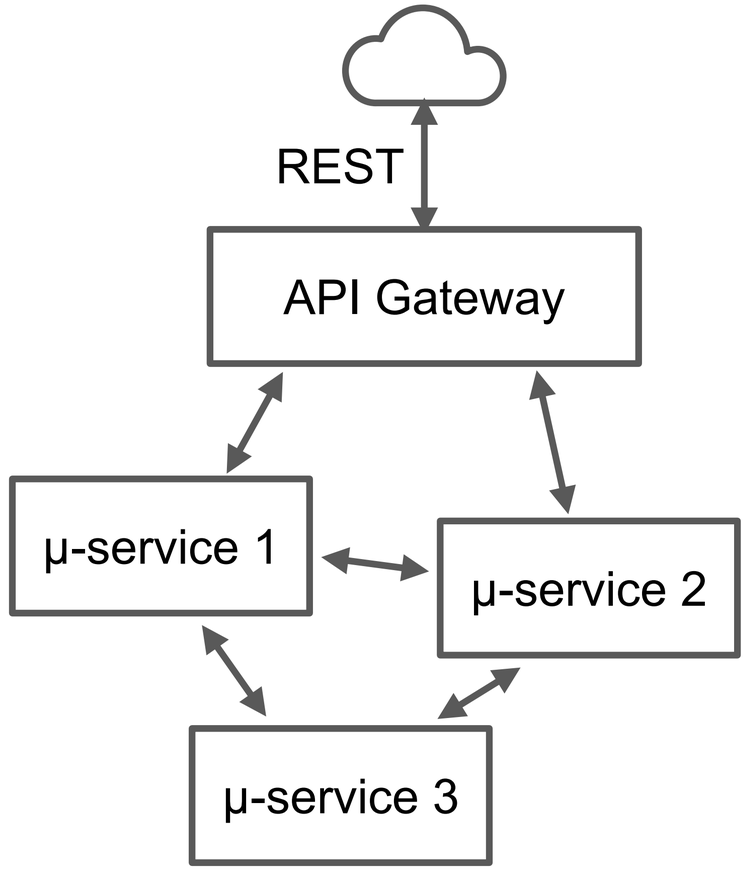
What Are Microservices?

- They partition the domain
  - Conway's Law - Embraced
  - Separate responsibilities
- Separate management



Conway's Law - Embraced

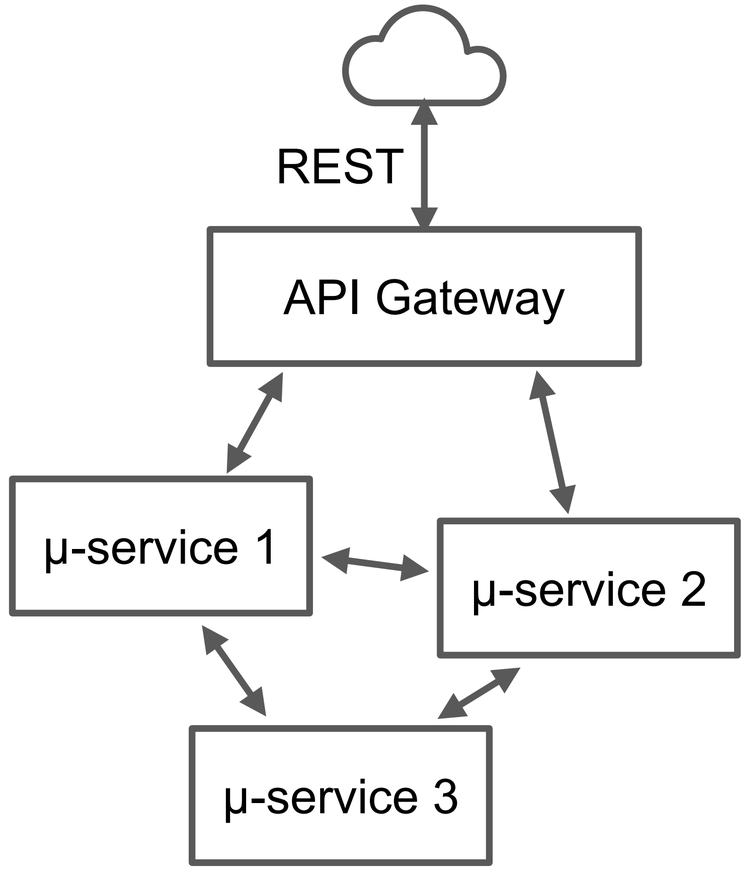
- "Any organization that designs a system will produce a design whose structure is a copy of the organization's communication structure"
- Let each team own and manage the services for its part of the domain



en.wikipedia.org/wiki/Conway's_law

Separate Responsibilities

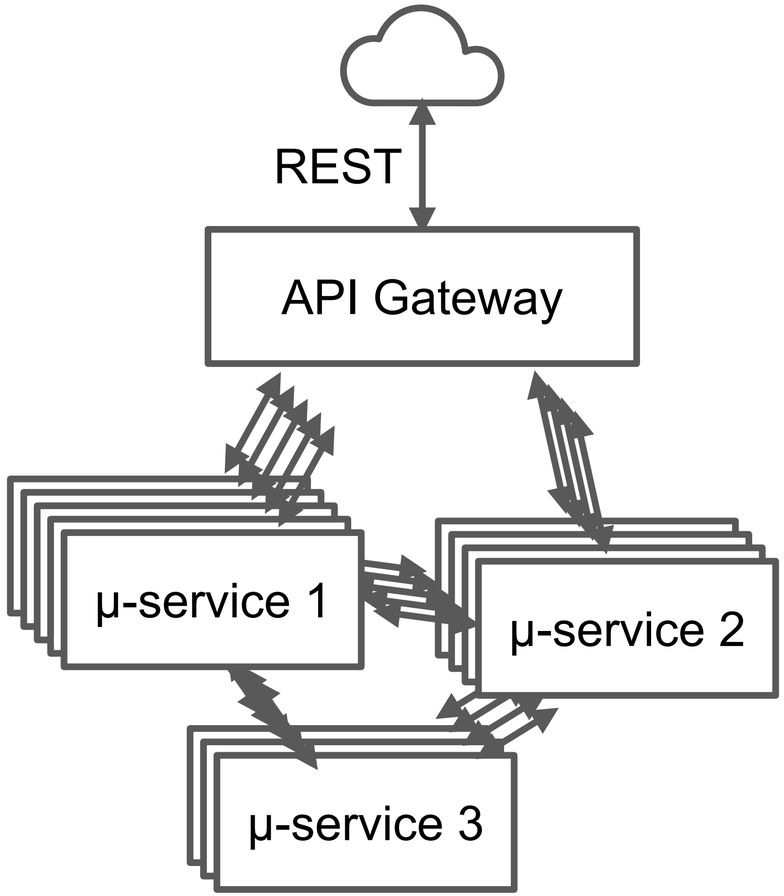
- Each microservice does "one thing", a single responsibility with minimal coupling to the other microservices
- (Like, hopefully, the teams are organized, too…)



wikipedia.org/wiki/Single-responsibility_principle

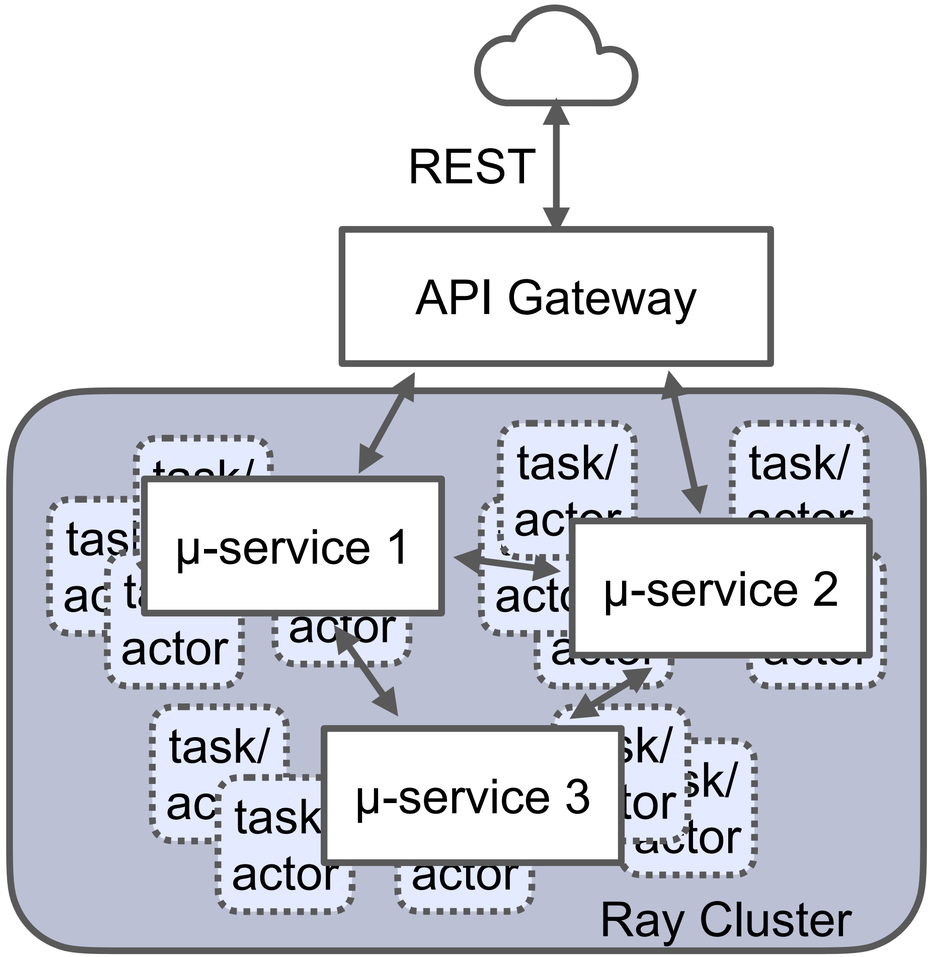
Separate Management

- Each team manages its own instances
- Each microservice has a different number of instances for scalability and resiliency
- But they have to be managed **explicitly**



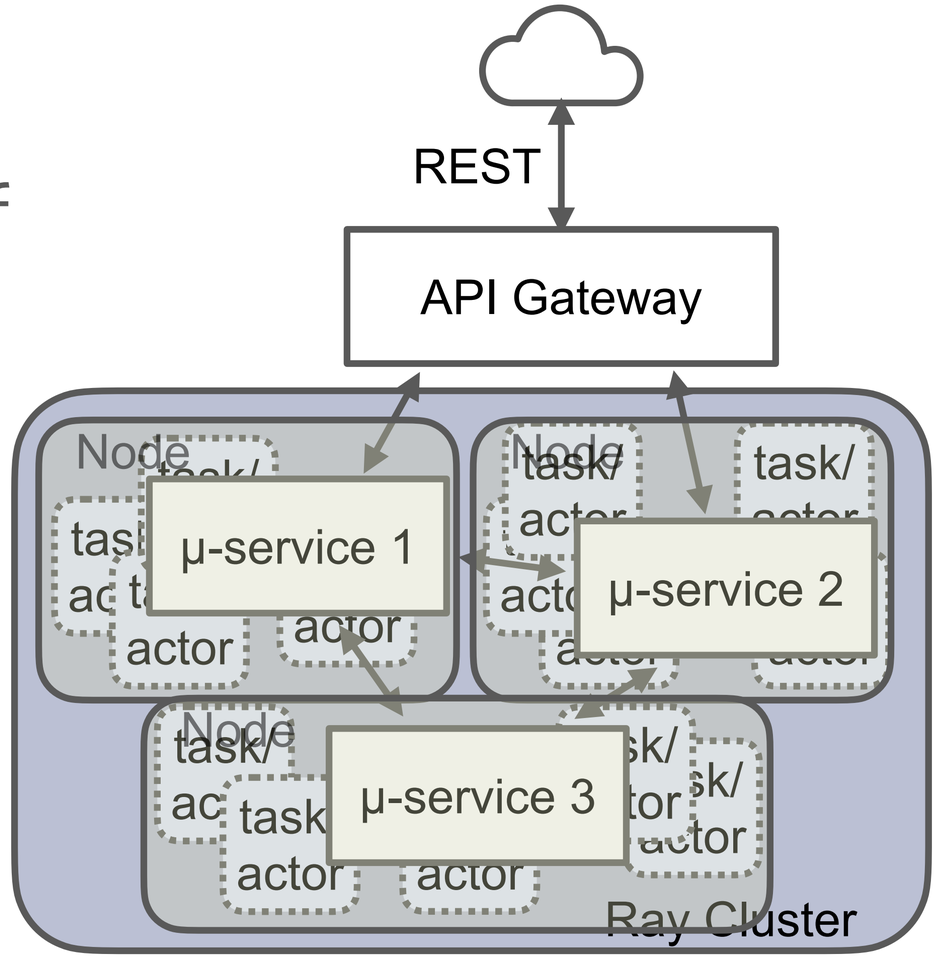
Management - Simplified

- With Ray, you have one "logical" instance to manage and Ray does the cluster-wide scaling for you.



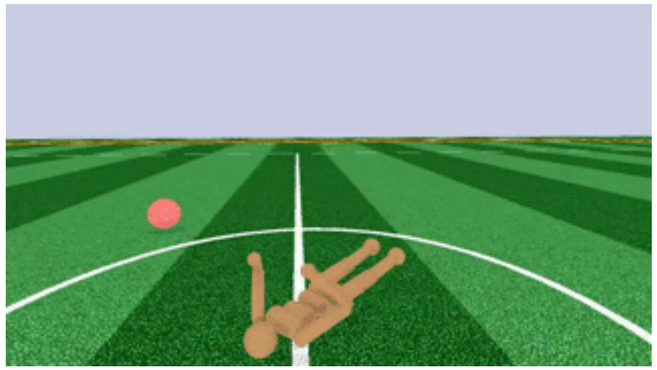
What about Kubernetes (and others…)?

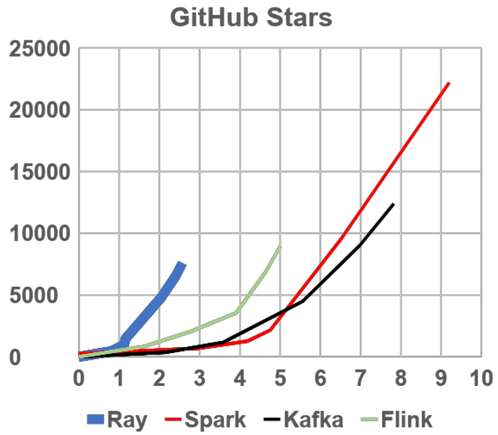
- Ray scaling is very fine grained.
- It operates within the "nodes" of coarse-grained managers
  - Containers, VMs, or physical machines

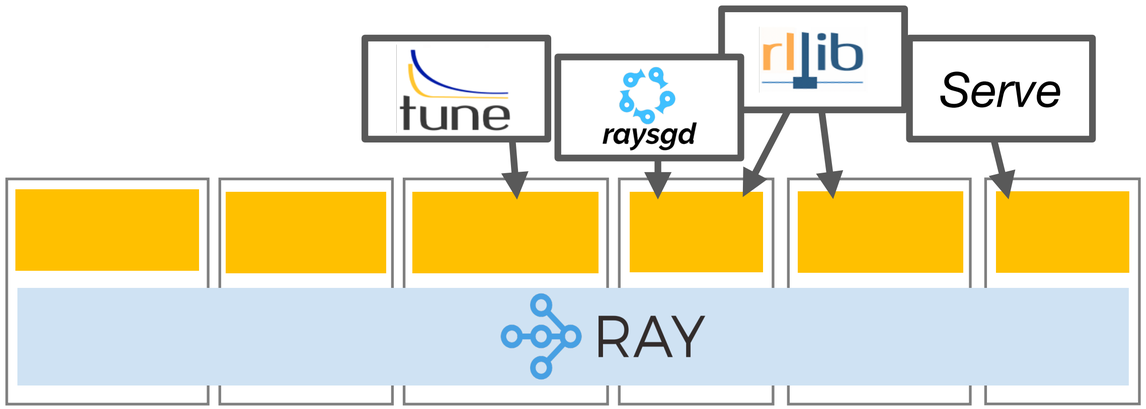


Conclusion

- Ray is the new state-of-the-art for distributed computing
  - The shortest path from your laptop to the cloud
  - Run complex distributed tasks on large clusters from simple code on your laptop







About Anyscale, Inc

- Spun out of U.C. Berkeley
- Making Ray the standard for distributed computing
- We are hiring!
- https://anyscale.com

Questions?

ray.io
anyscale.com - We're Hiring!
anyscale.com/events
raysummit.org
dean@anyscale.com

RAY

anyscale