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

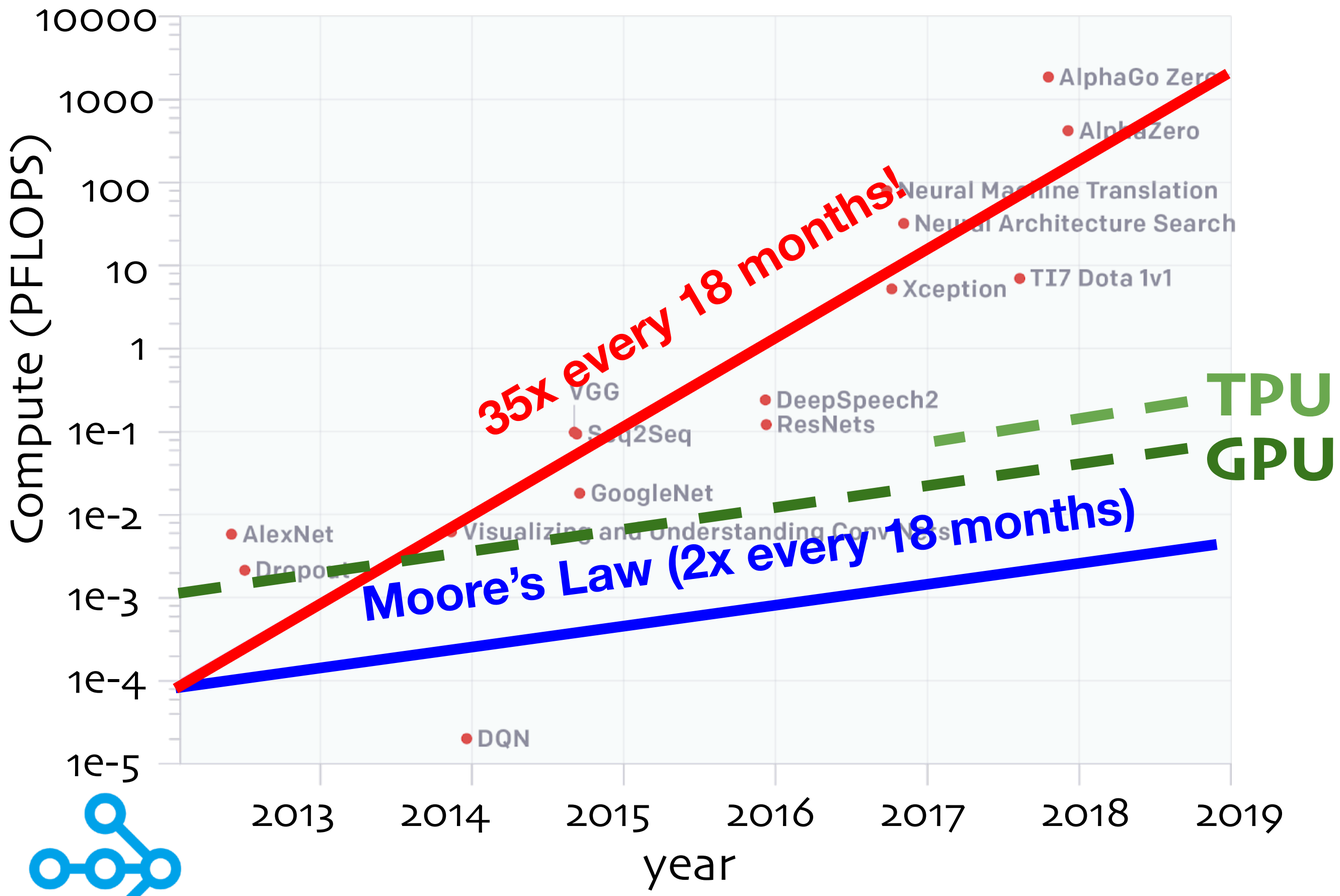
RAY

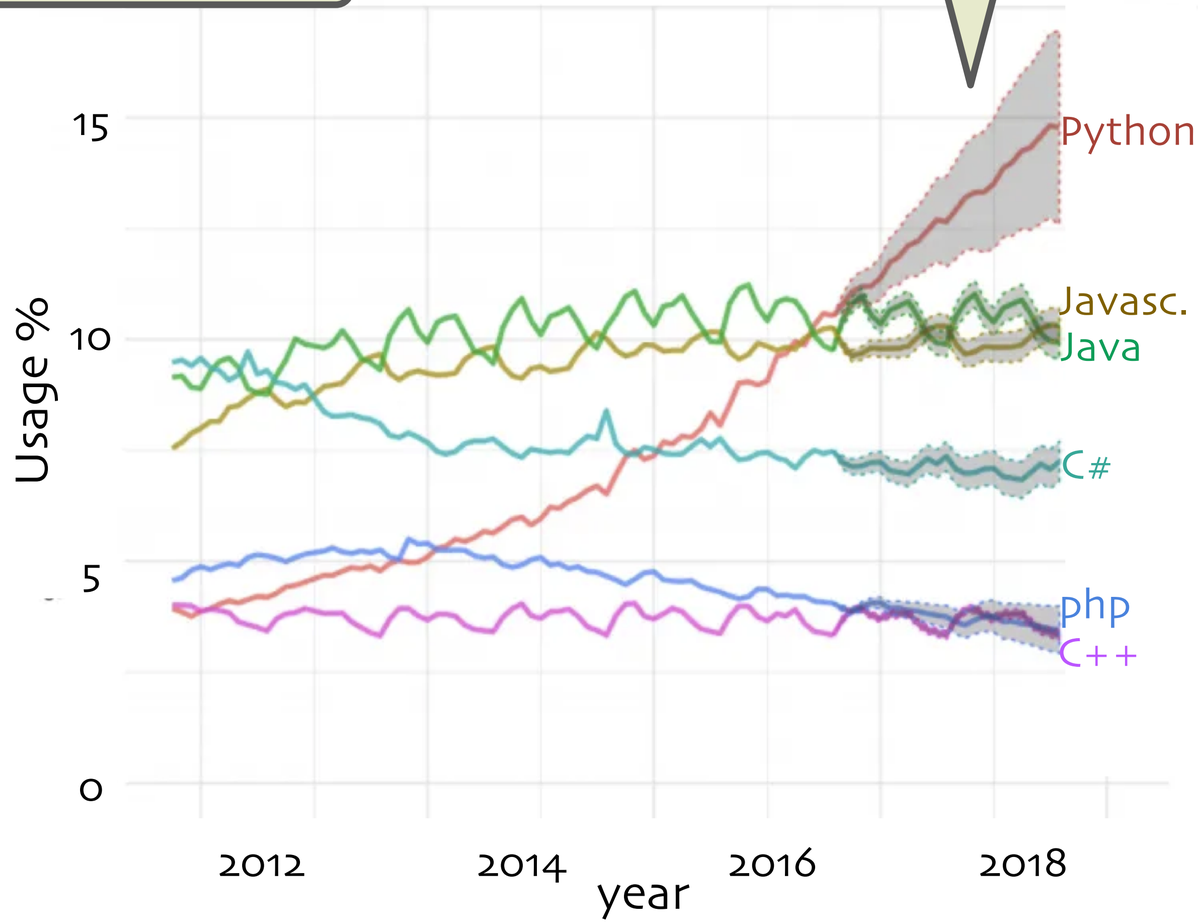
anyscale

Two Major Trends

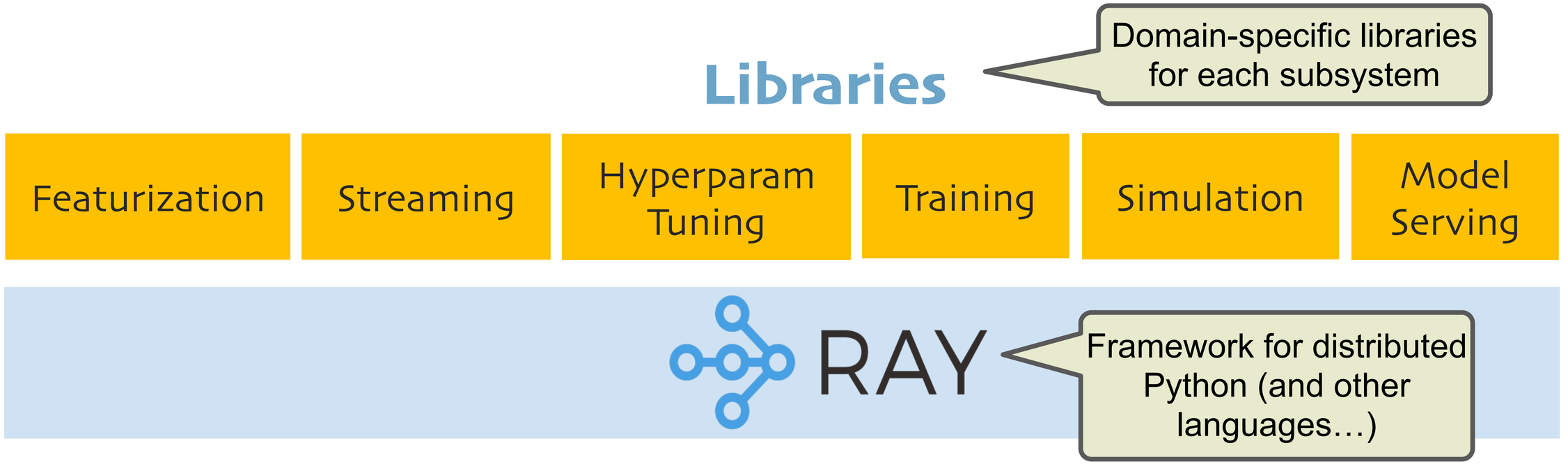
There is a pressing need for robust, easy to use solutions for distributed Python

Driven in large part by ML/AI and other data science workloads





The Ray Vision: Sharing a Common Framework



API

Functions -> Tasks

Designed to intuitive and concise

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


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

The Python you know already…

API

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”

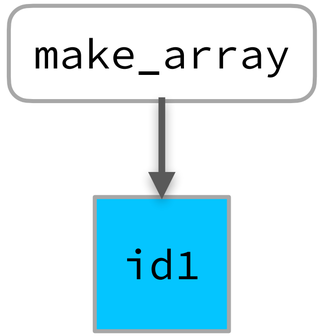
For completeness, add these First:

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

API

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

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(…)
```

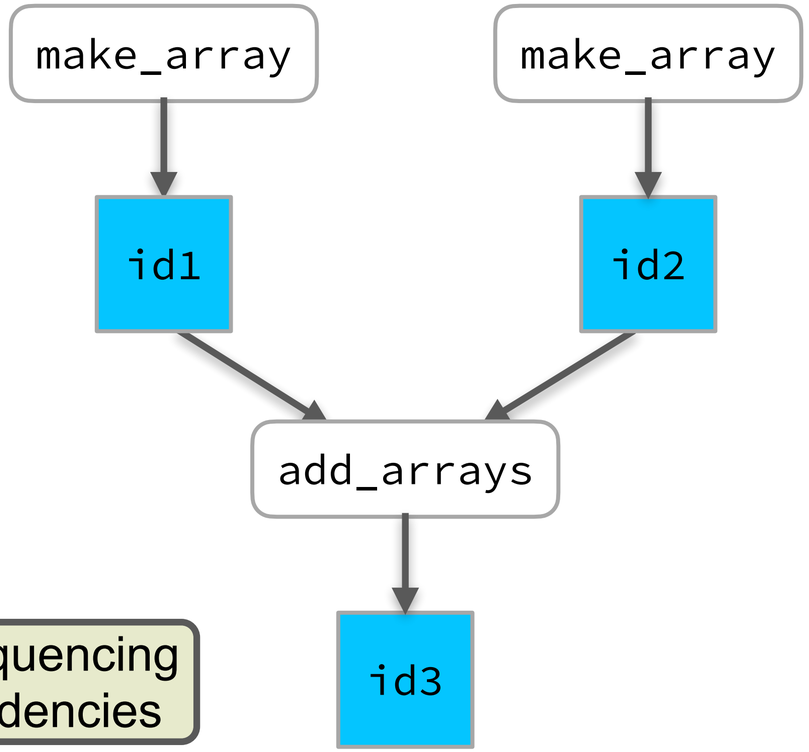
API

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 handles extracting the arrays from the object ids

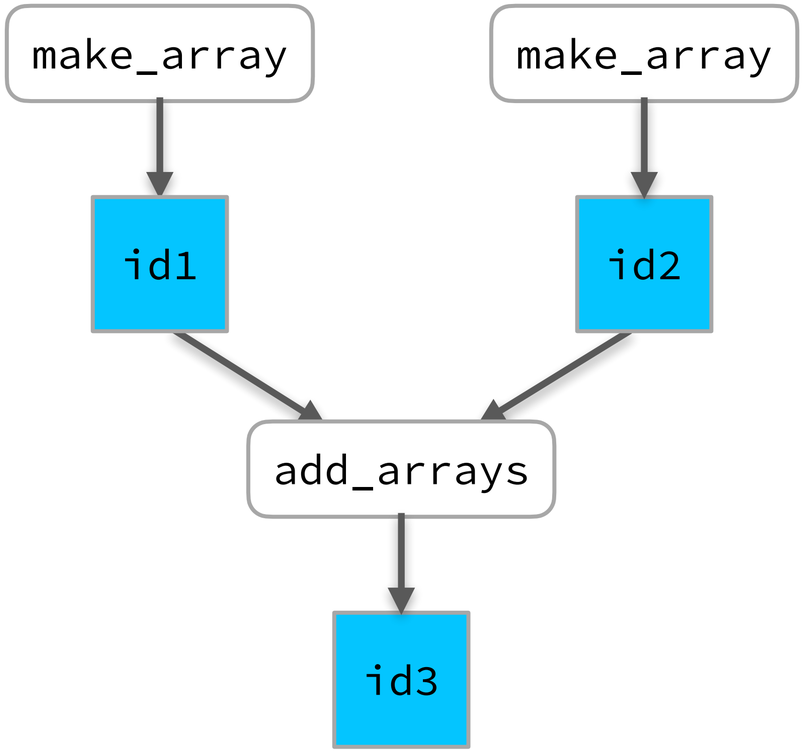
Ray handles sequencing of async dependencies



API

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

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)
```

What about distributed state?

API

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…

What about distributed state?

API

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
```

… now a remote "actor"

API

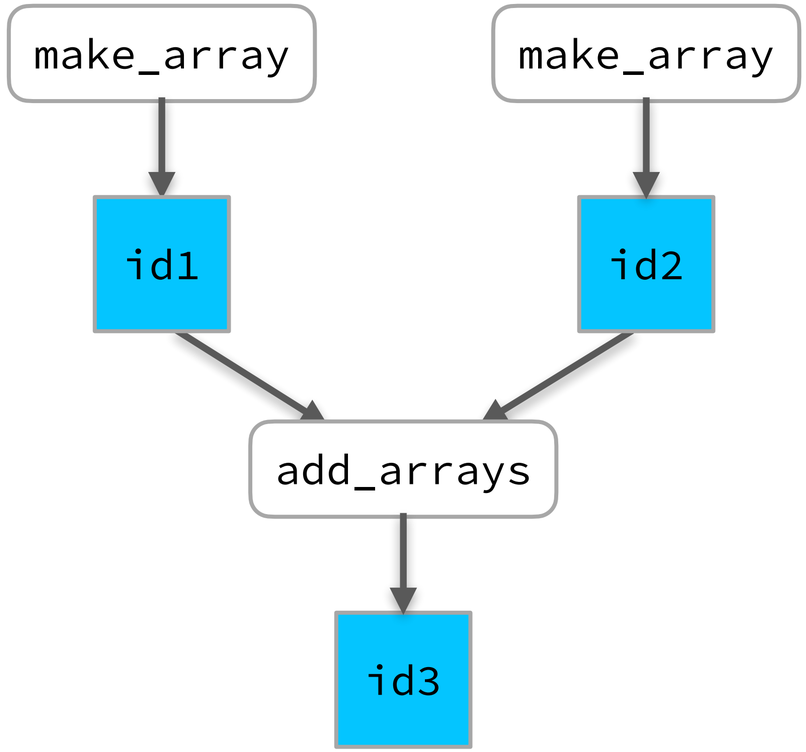
Functions -> Tasks

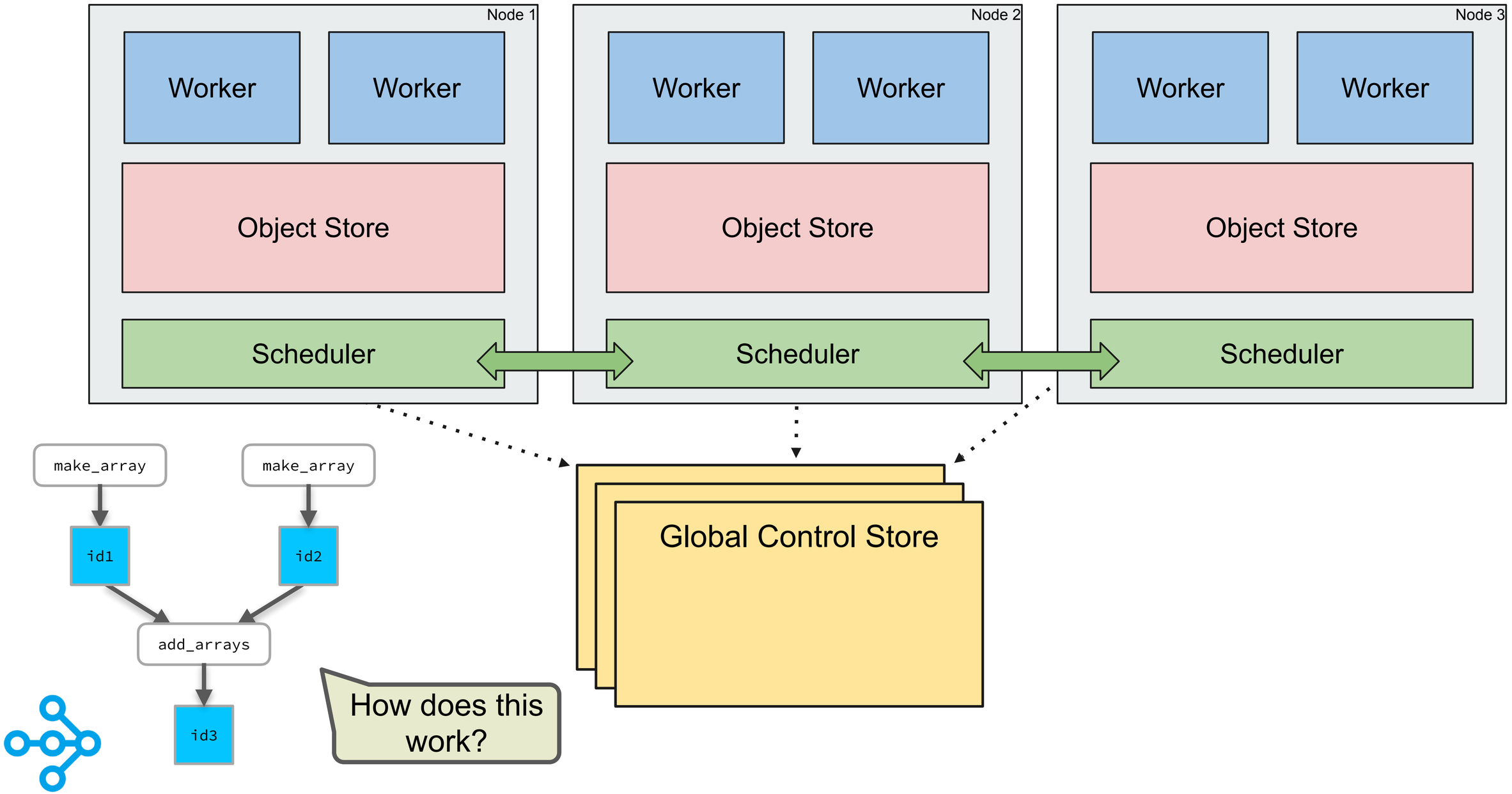
```
@ray.remote
def make_array(…):
    a = … # Construct a NumPy array
    return a

@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)
```

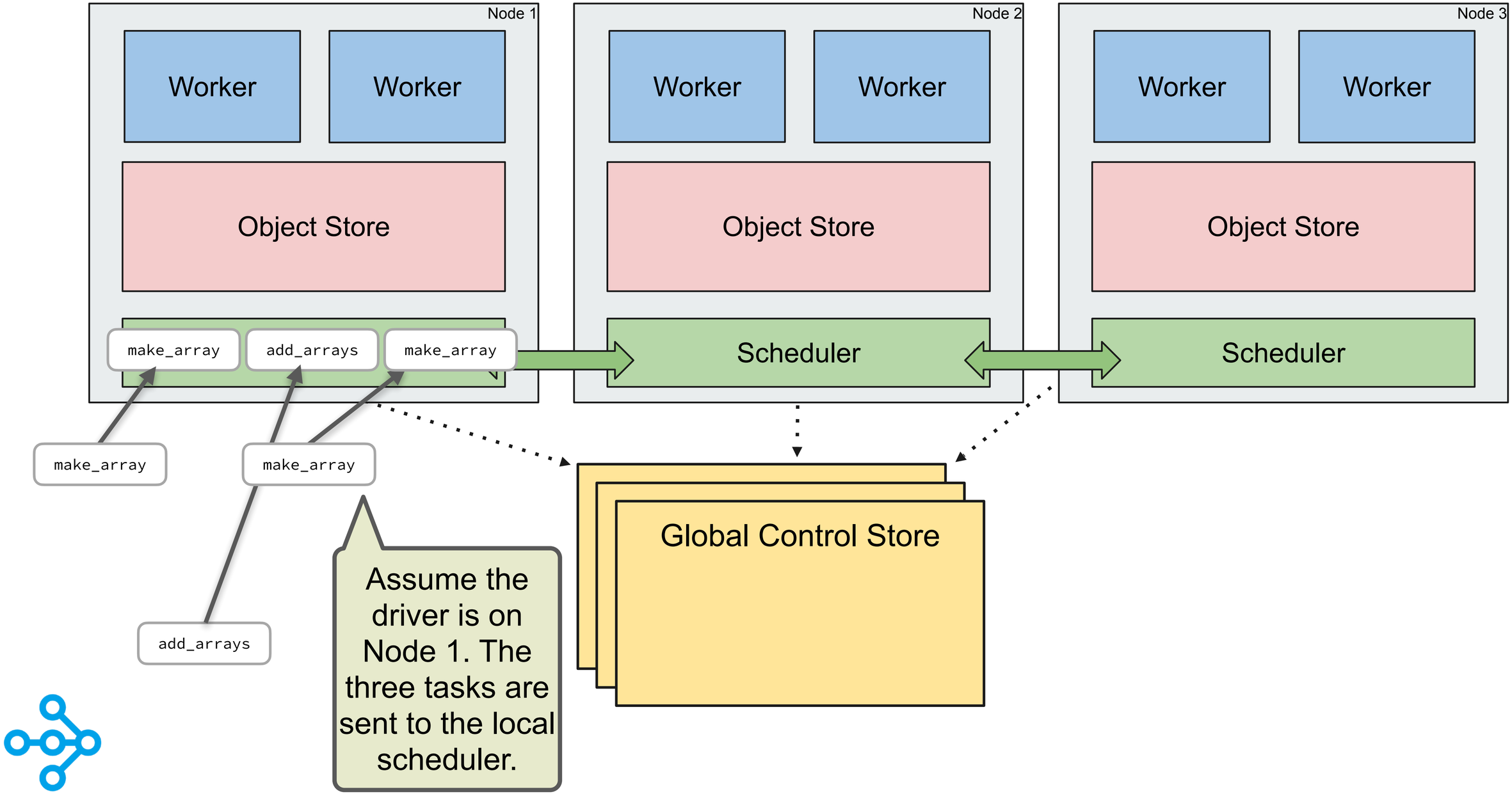
Classes -> Actors

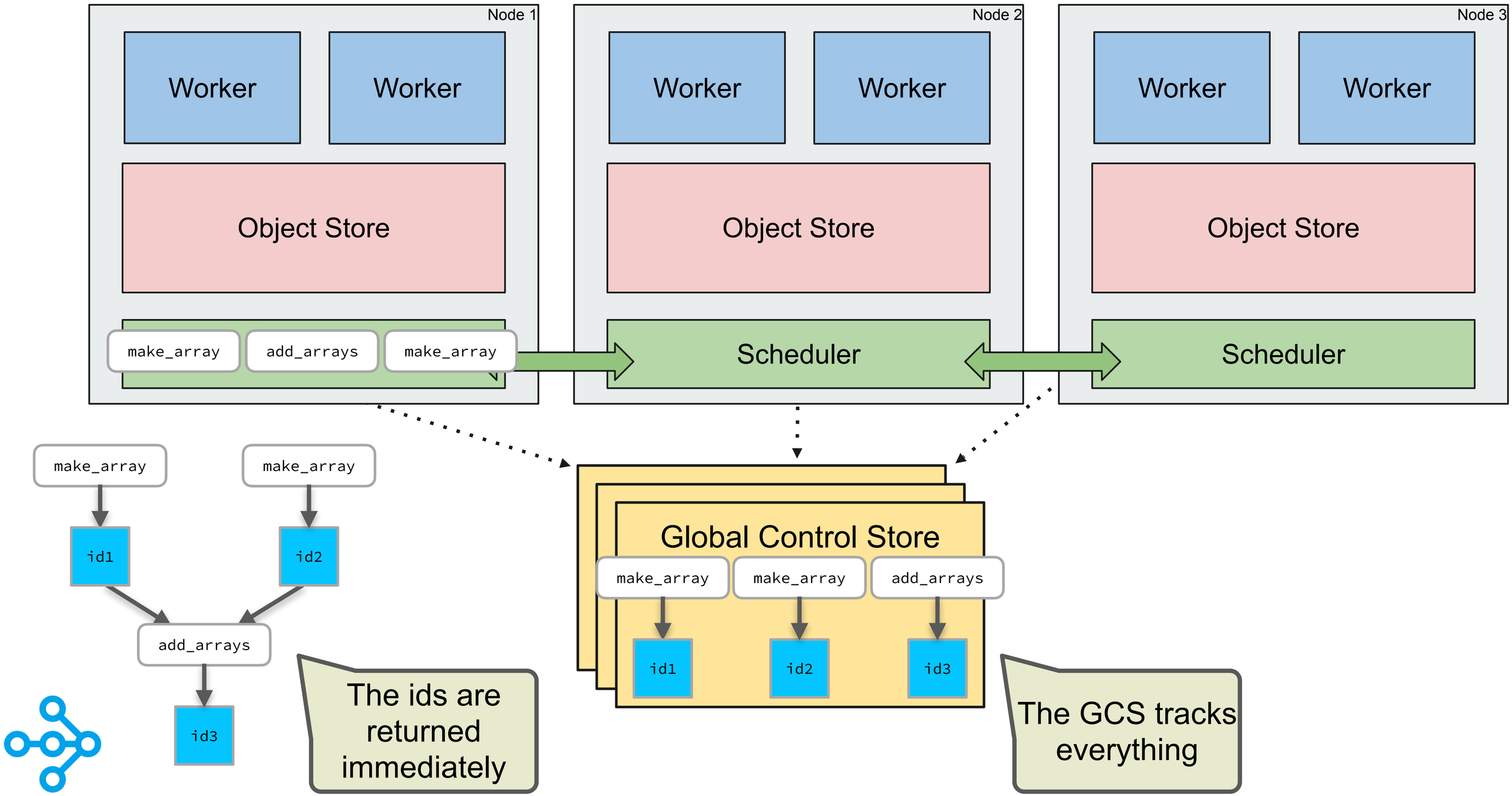
```
@ray.remote
class Counter(object):
    def __init__(self):
        self.value = 0
    def increment(self):
        self.value += 1
        return self.value

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

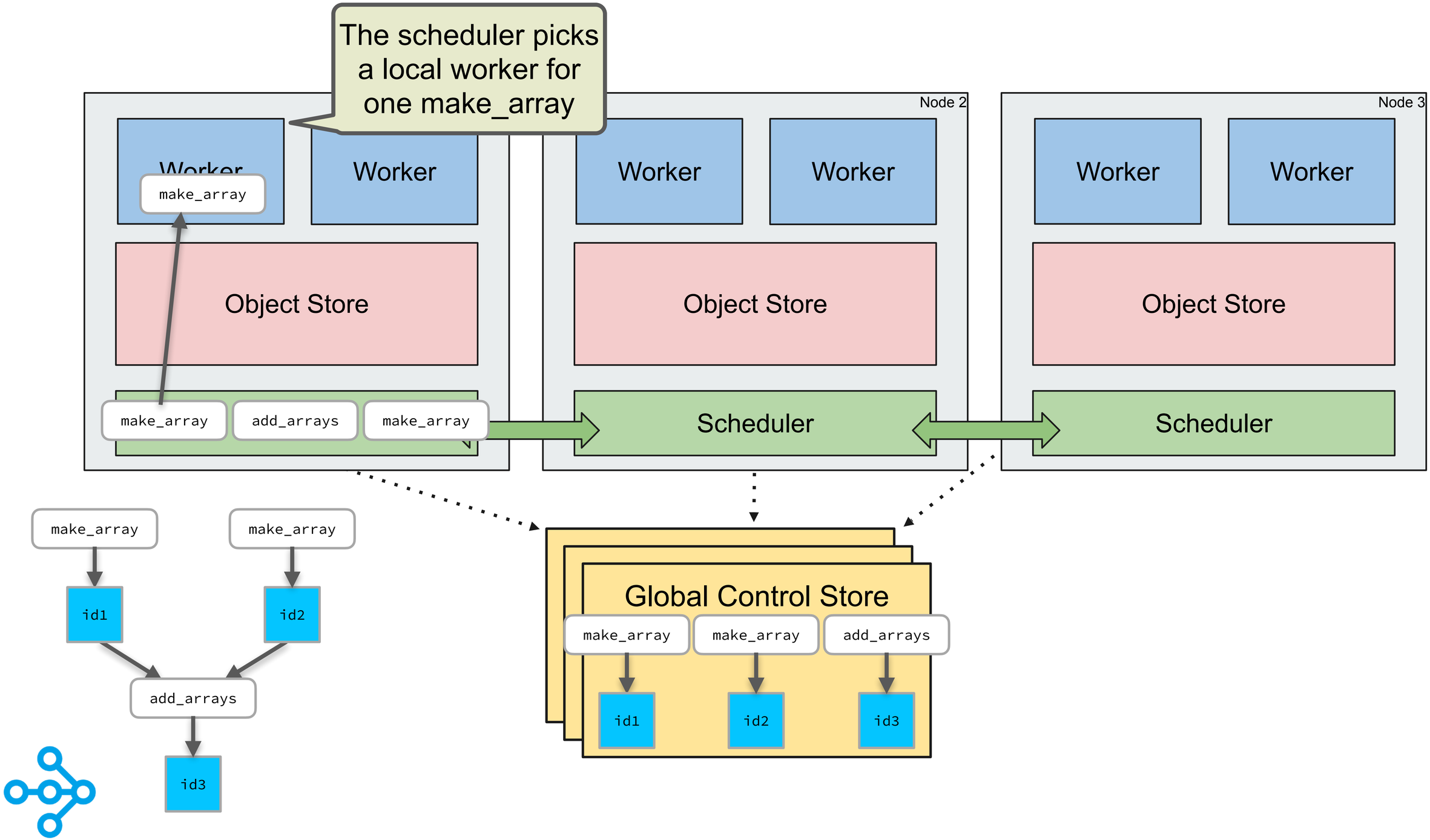
How does this work behind the scenes?

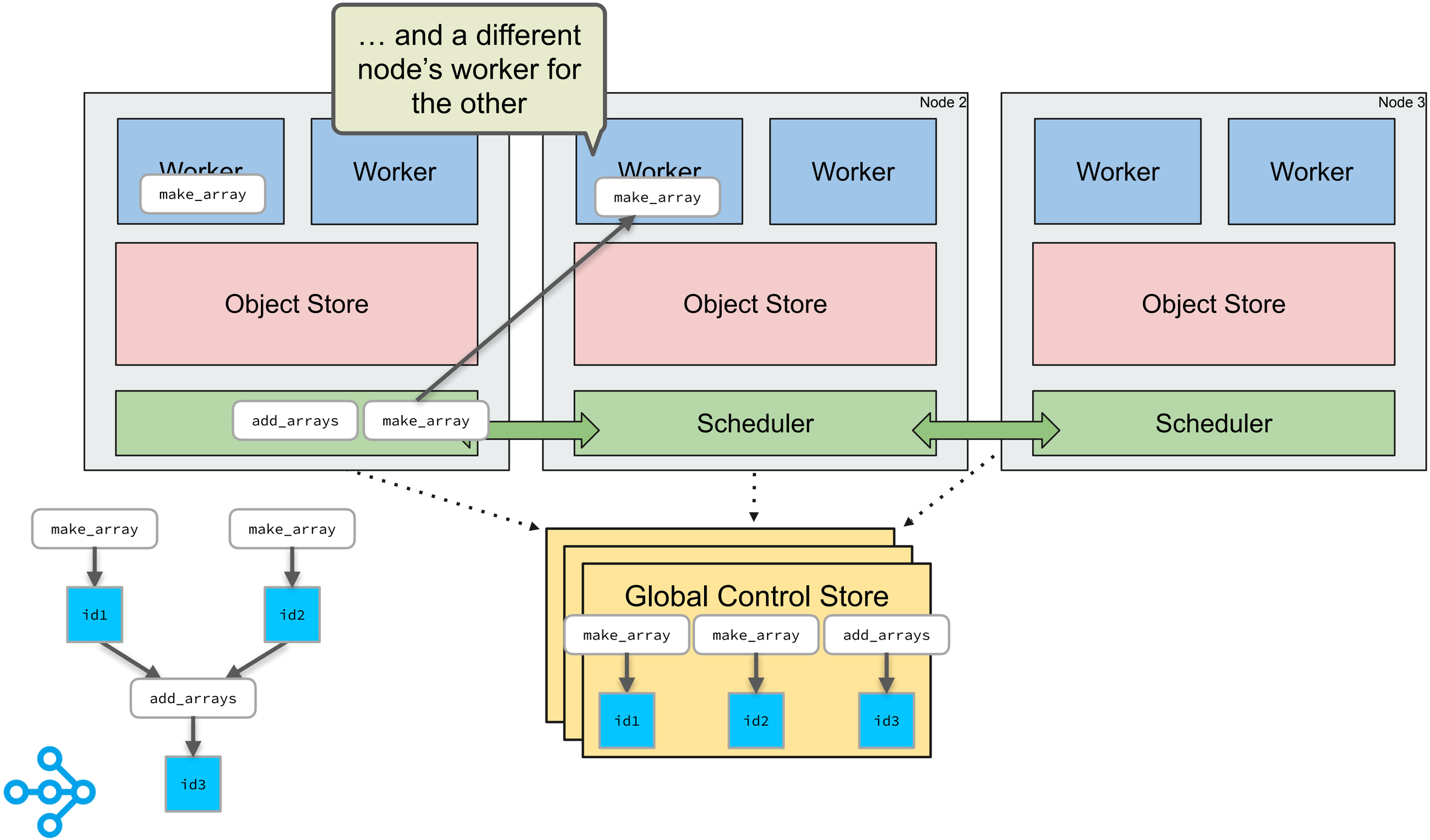


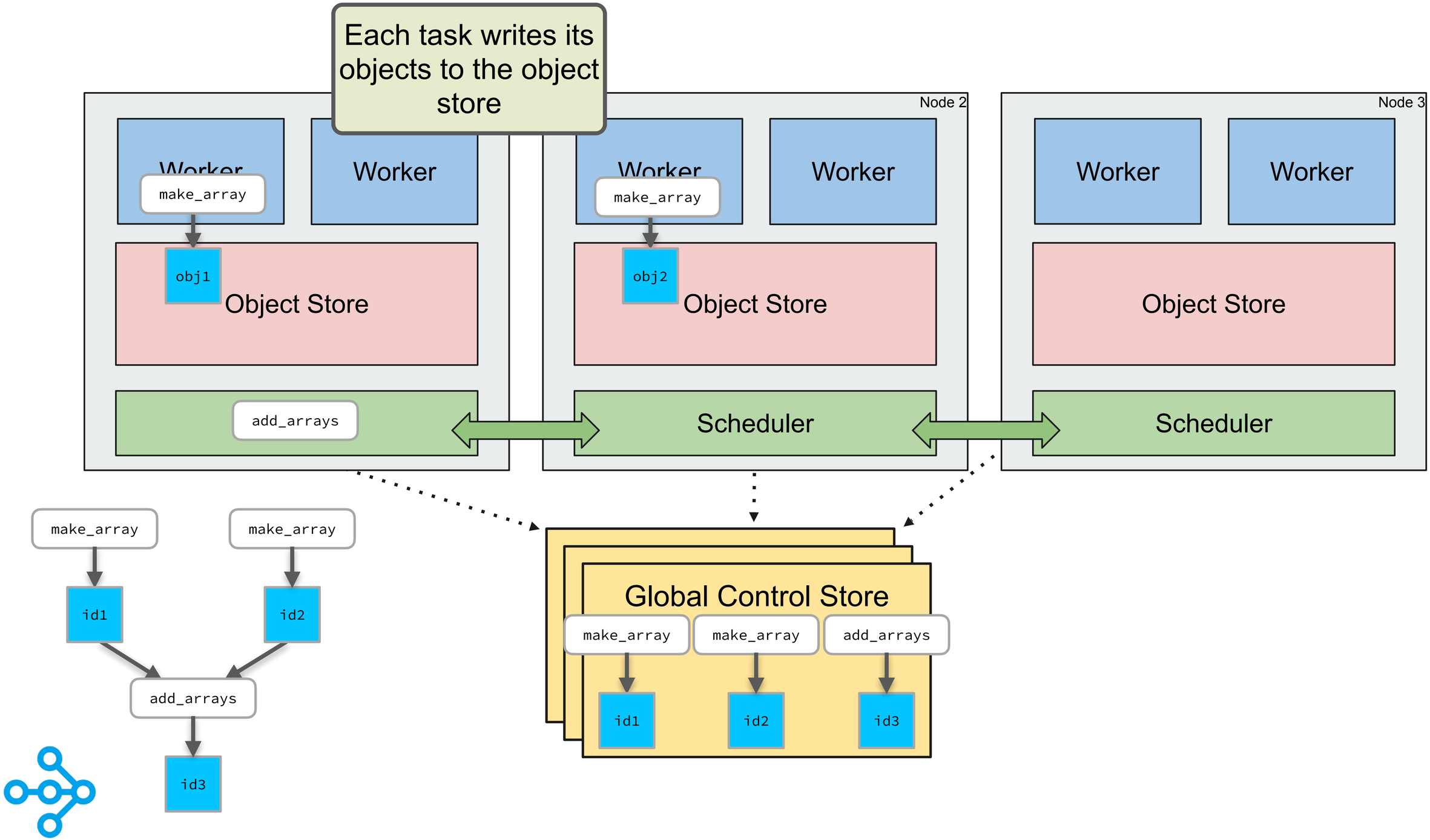

Node 1
Worker
Worker
Object Store
Scheduler
Node 2
Worker
Worker
Object Store
Scheduler
Node 3
Worker
Worker
Object Store
Scheduler
Global Control Store
make_array
make_array
id1
id2
add_arrays
id3
How does this work?

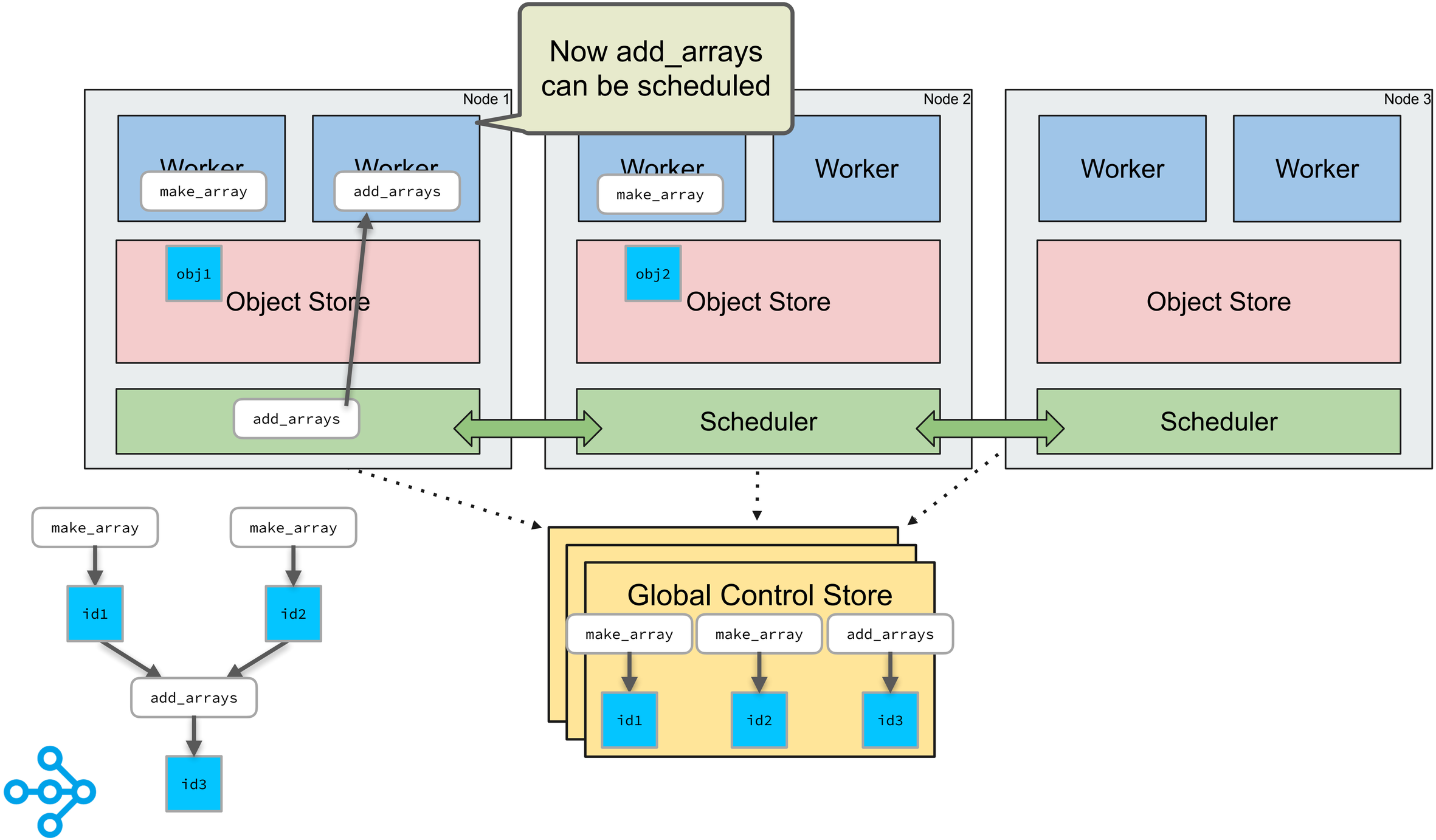

Node 1
Worker
Worker
Object Store
Scheduler
Node 2
Worker
Worker
Object Store
Scheduler
Node 3
Worker
Worker
Object Store
Scheduler
make_array
add_arrays
make_array
make_array
make_array
add_arrays
Global Control Store
Assume the driver is on Node 1. The three tasks are sent to the local scheduler.

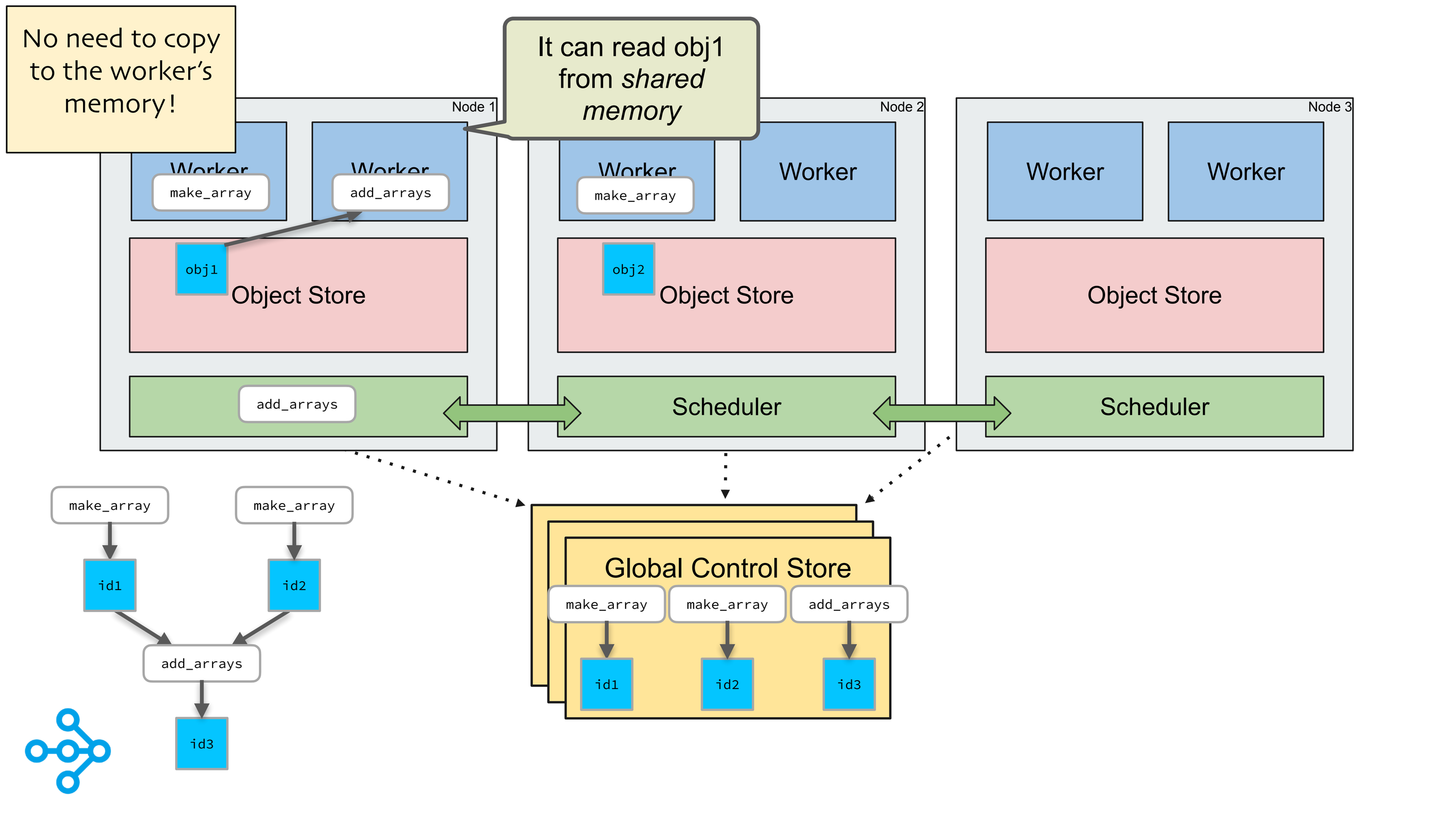

Node 1
Worker
Worker
Object Store
make_array
add_arrays
make_array
Node 2
Worker
Worker
Object Store
Scheduler
Node 3
Worker
Worker
Object Store
Scheduler
make_array
make_array
id1
id2
add_arrays
id3
Global Control Store
make_array
make_array
add_arrays
id1
id2
id3
The ids are returned immediately
The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

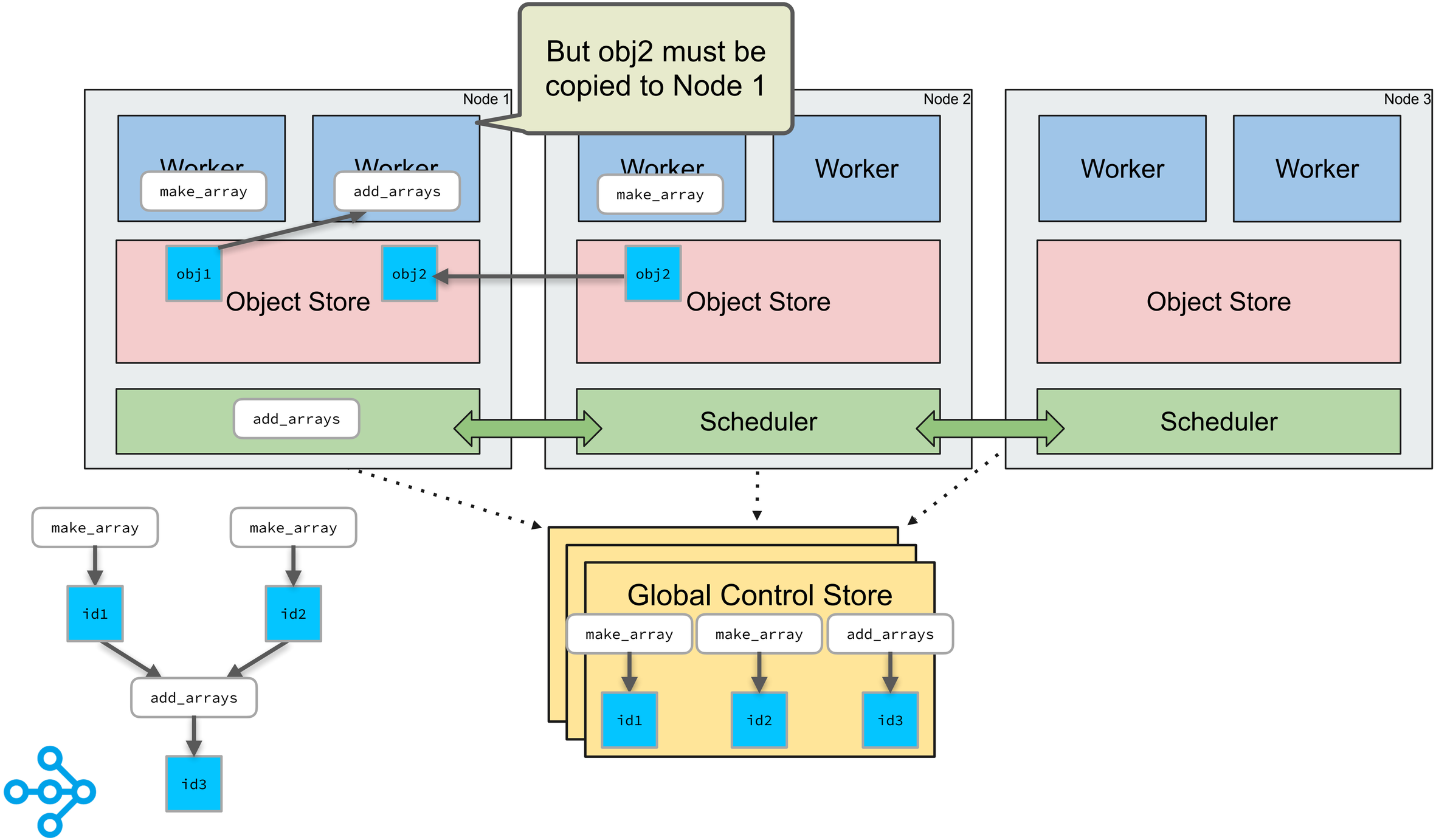

The scheduler picks a local worker for one make_array
Node 2
Node 3
Worker
Worker
Worker
Worker
Worker
Worker
make_array
Object Store
Object Store
Object Store
make_array
add_arrays
make_array
Scheduler
Scheduler
make_array
make_array
id1
id2
add_arrays
id3
Global Control Store
make_array
make_array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

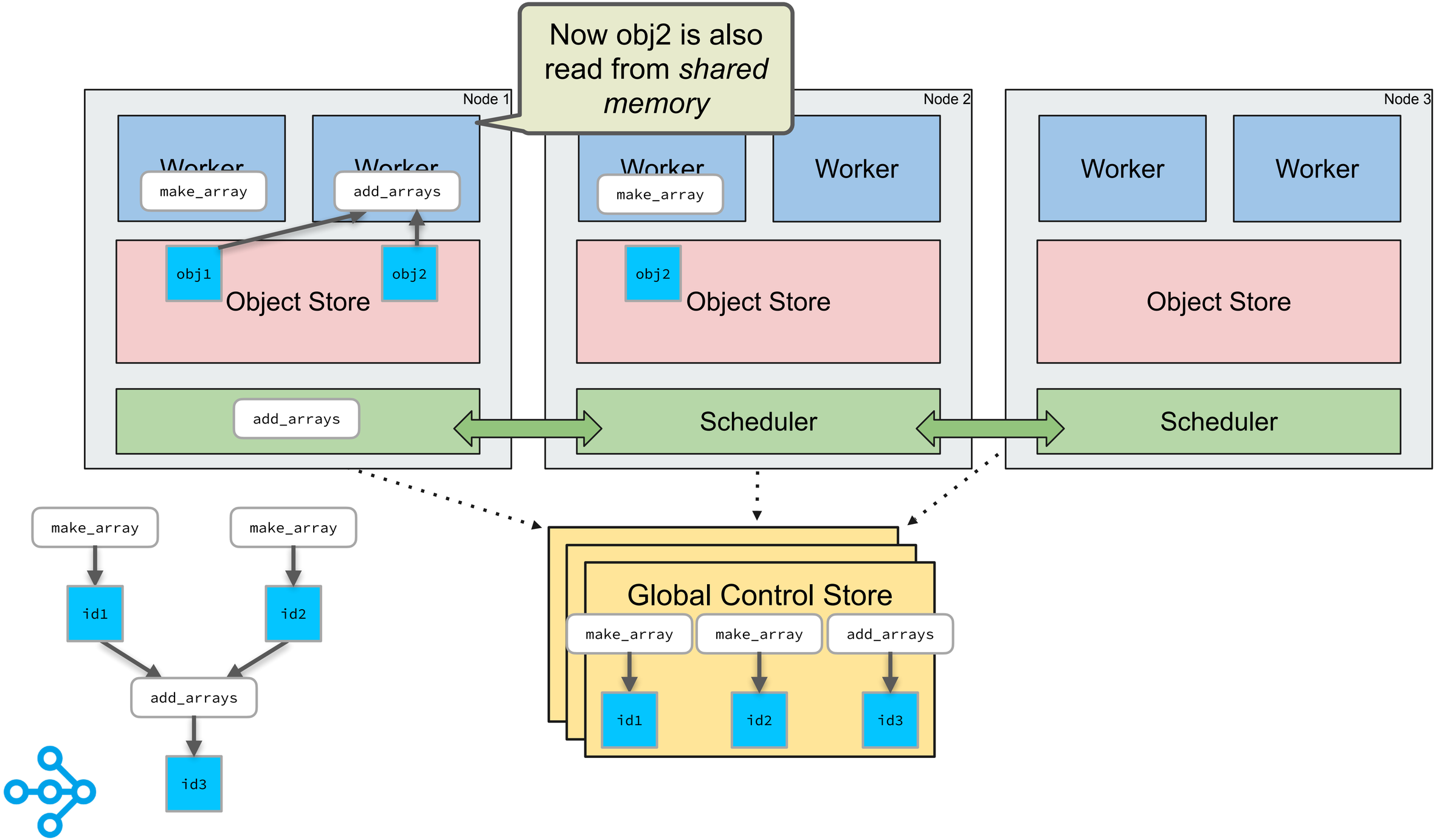

… and a different node's worker for the other
Node 2
Node 3
Worker
Worker
Worker
Worker
Worker
Worker
make_array
make_array
Object Store
Object Store
Object Store
add_arrays
make_array
Scheduler
Scheduler
make_array
make_array
id1
id2
add_arrays
id3
Global Control Store
make_array
make_array
add_arrays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

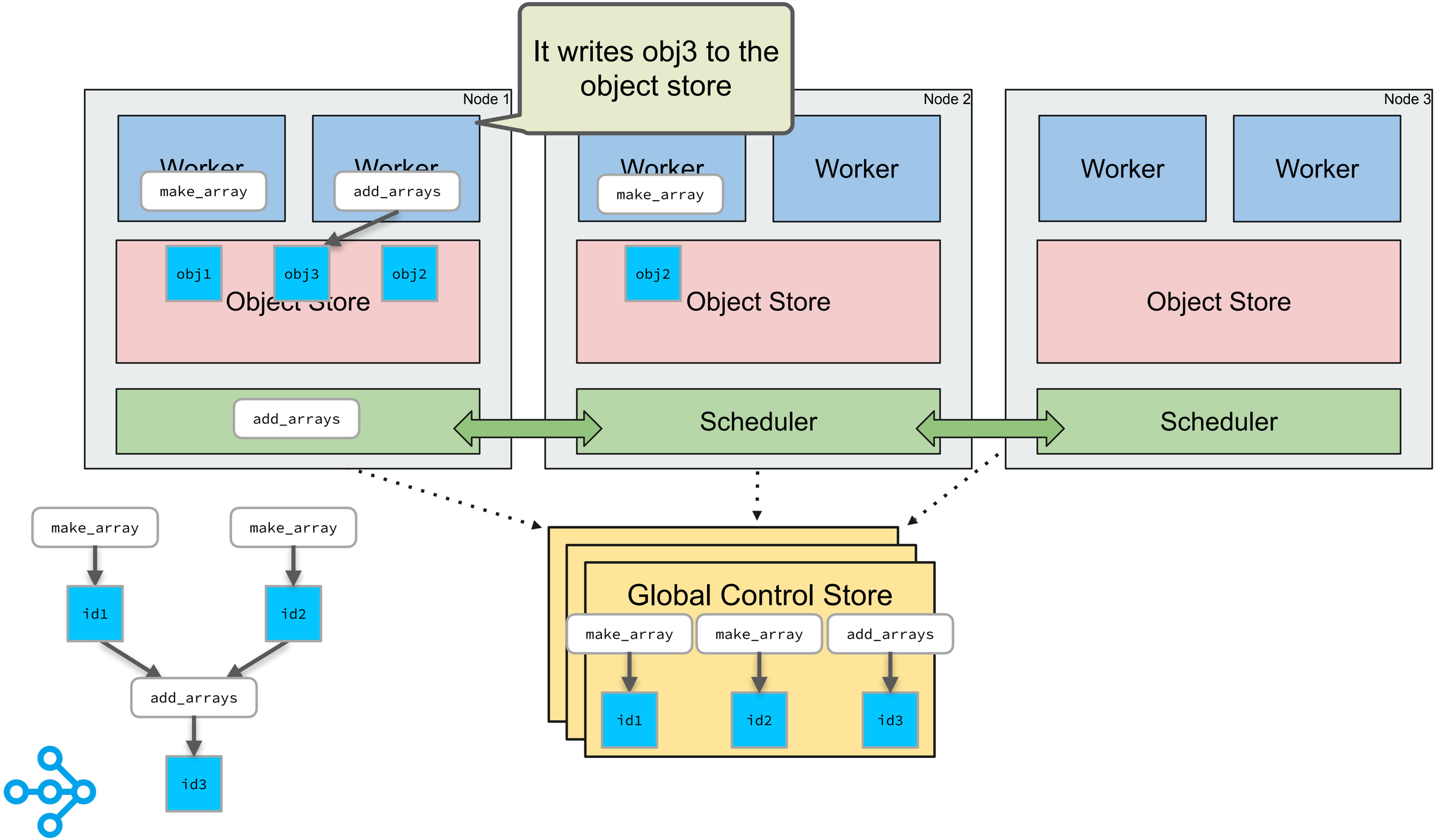

Each task writes its objects to the object store
Node 2
Node 3
Worker
Worker
Worker
Worker
Worker
Worker
make_array
make_array
obj1
obj2
Object Store
Object Store
Object Store
add_arrays
Scheduler
Scheduler
make_array
make_array
id1
id2
add_arrays
id3
Global Control Store
make_array
make_array
add_arrays
id1
id2
id3

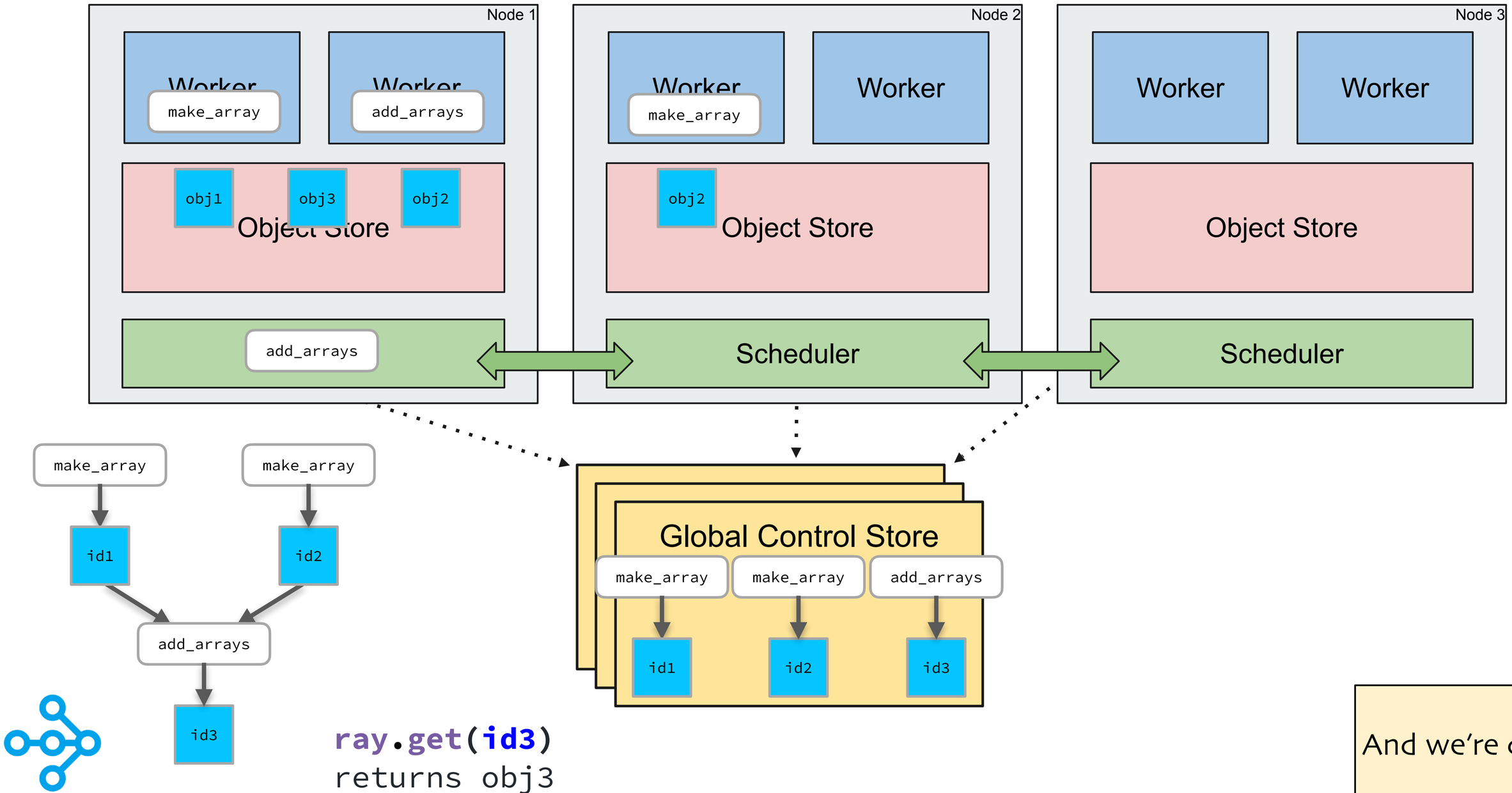

Now add_arrays can be scheduled
Node 1
Node 2
Node 3
Worker
Worker
Worker
Worker
Worker
make_array
add_arrays
make_array
obj1
Object Store
obj2
Object Store
Object Store
add_arrays
Scheduler
Scheduler
make_array
make_array
id1
id2
add_arrays
id3
Global Control Store
make_array
make_array
add_arrays
id1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


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
make_array
add_arrays
make_array
obj1
obj2
Object Store
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


But obj2 must be copied to Node 1
Node 1
Node 2
Node 3
Worker
Worker
Worker
Worker
Worker
make_array
add_arrays
make_array
obj1
obj2
obj2
Object Store
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


Now obj2 is also read from *shared memory*
Node 1
Node 2
Node 3
Worker
Worker
Worker
Worker
Worker
make_array
add_arrays
make_array
obj1
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


It writes obj3 to the object store
Node 1
Node 2
Node 3
Worker
Worker
Worker
Worker
Worker
make_array
add_arrays
make_array
obj1
obj3
obj2
obj2
Object Store
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


Node 1
Worker
Worker
make_array
add_arrays
obj1
obj3
obj2
Object Store
add_arrays
Node 2
Worker
Worker
make_array
obj2
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
ray.get(id3)
returns obj3
And we're done!

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
```

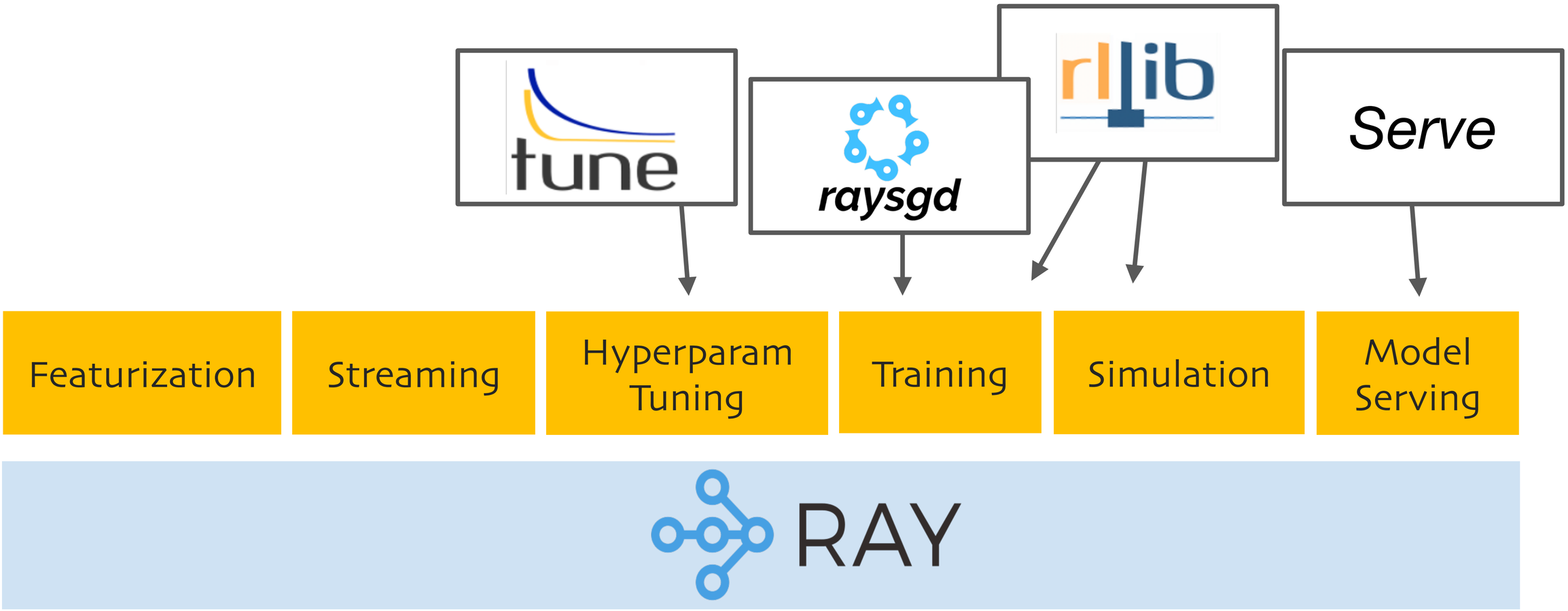
- Use Ray's implementations
  - Drop-in replacements
  - Change import statements
  - Break the one-node limitation!

See this blog post:
https://medium.com/distributed-computing-with-ray/how-to-scale-python-multiprocessing-to-a-cluster-with-one-line-of-code-d19f242f60ff

Machine Learning with Ray-based Libraries

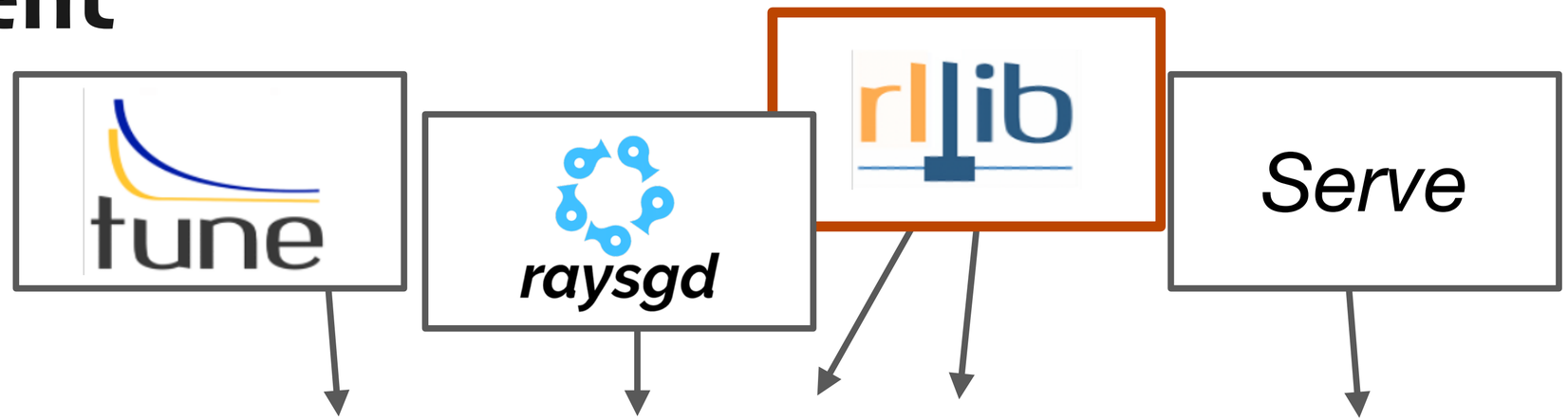
@deanwampler

Ray Libraries



More libraries coming soon

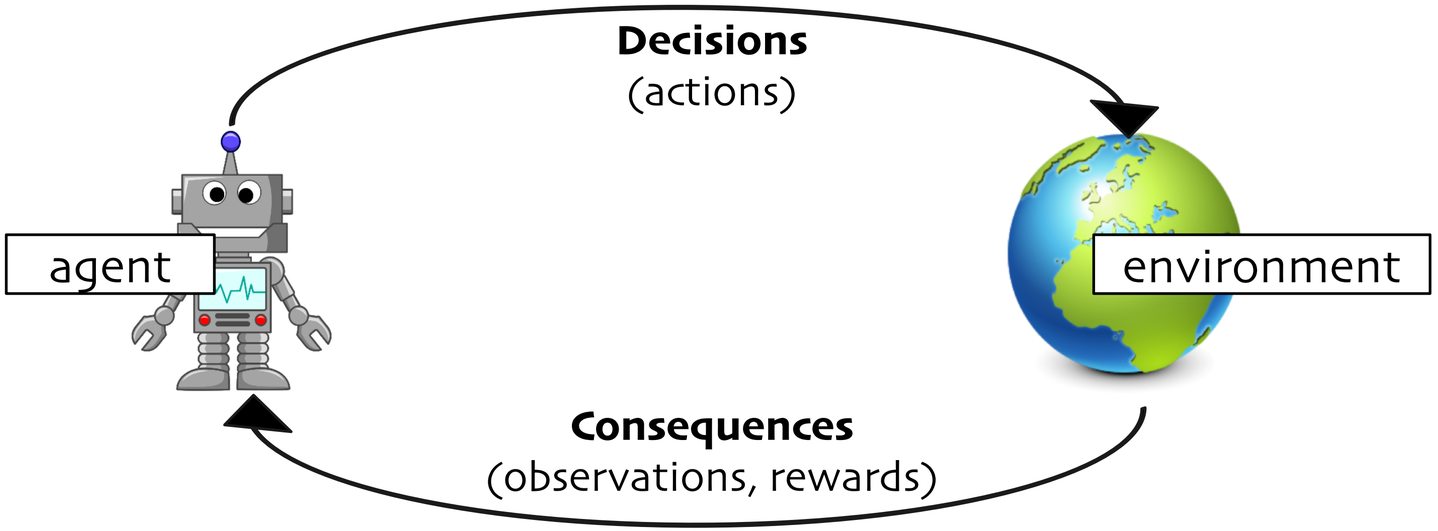
Reinforcement Learning

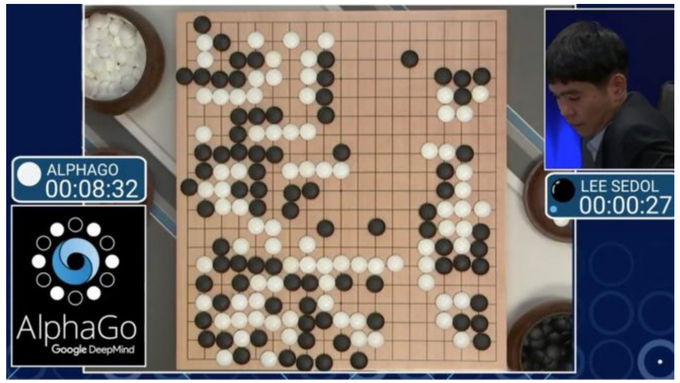


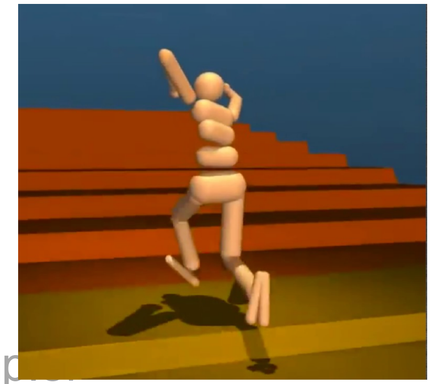
Featurization | Streaming | Hyperparam Tuning | Training | Simulation | Model Serving

RAY

Background: Reinforcement Learning



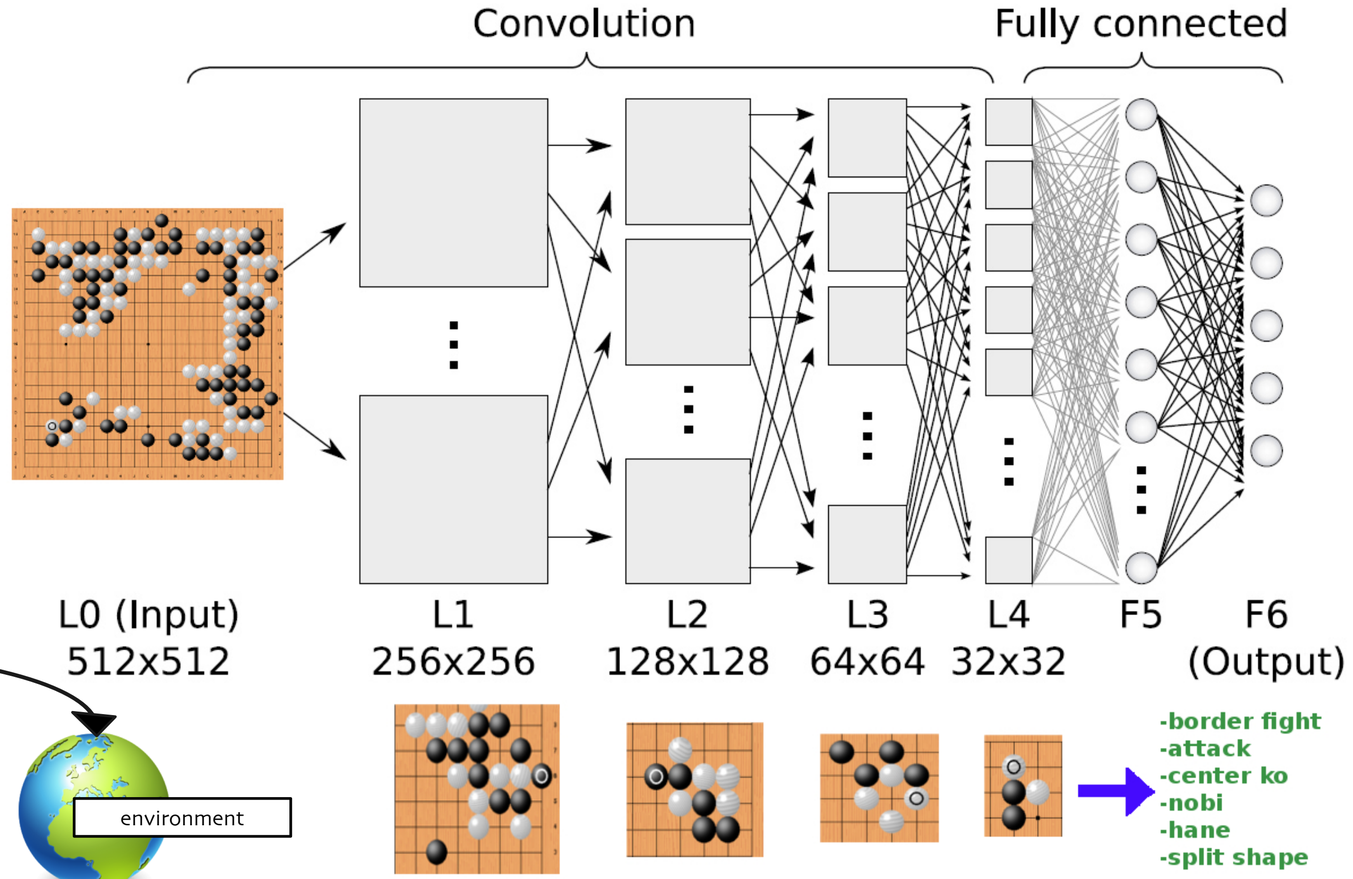


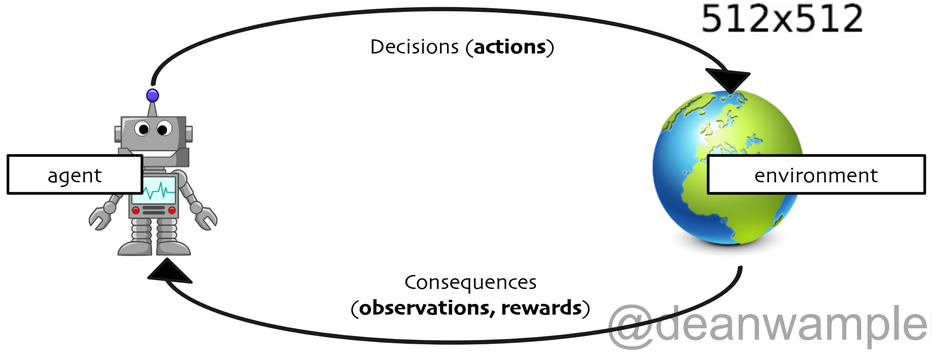


Go as a Reinforcement Learning Problem

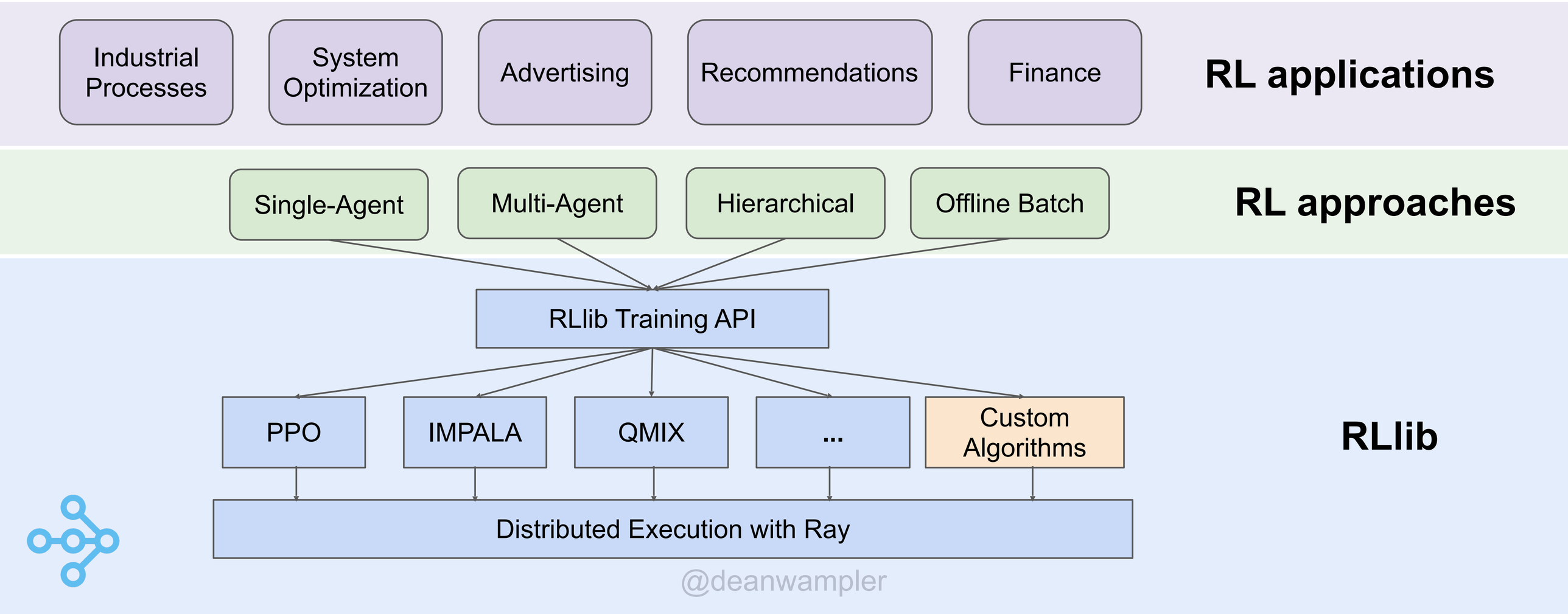
AlphaGo (Silver et al. 2016)

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

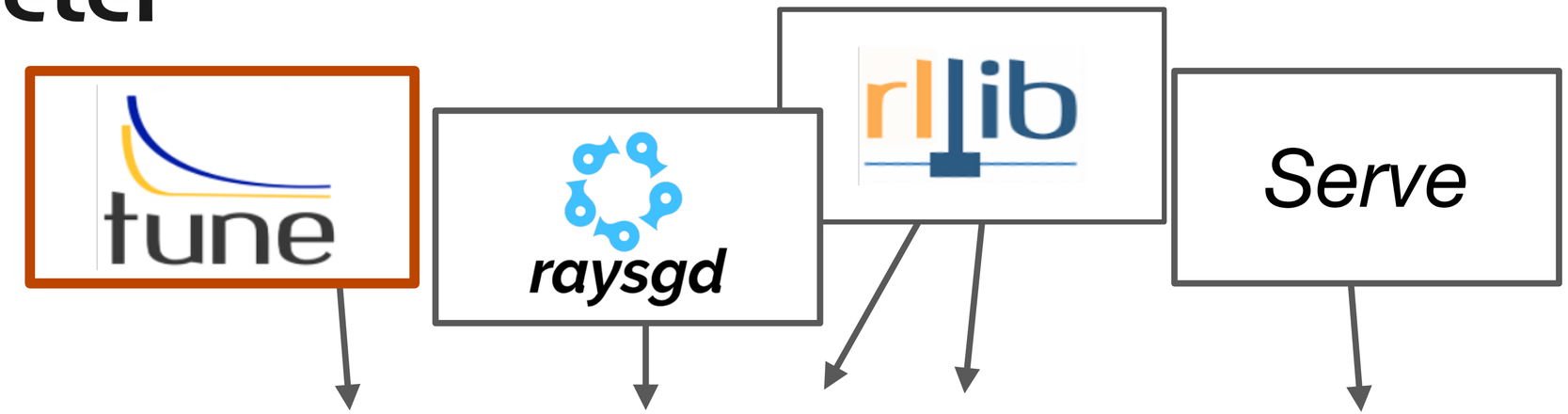




RLlib: A Scalable, Unified Library for RL



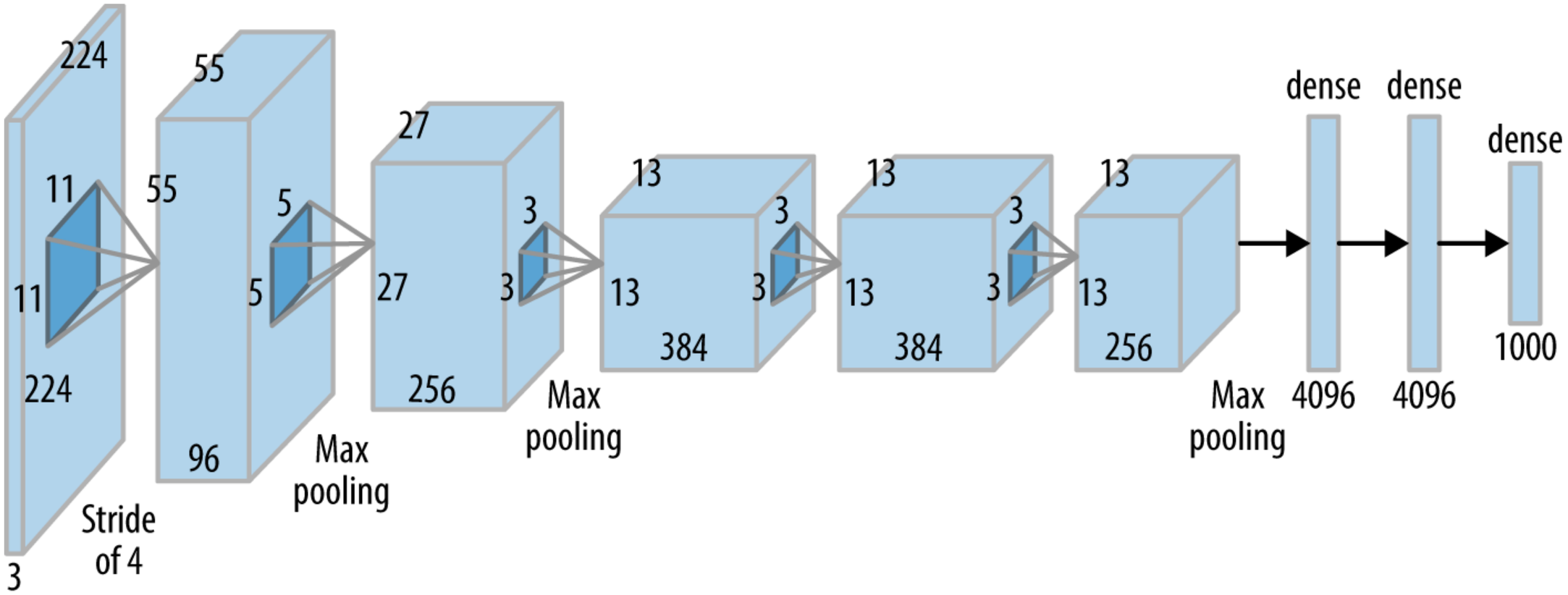
Hyperparameter Tuning



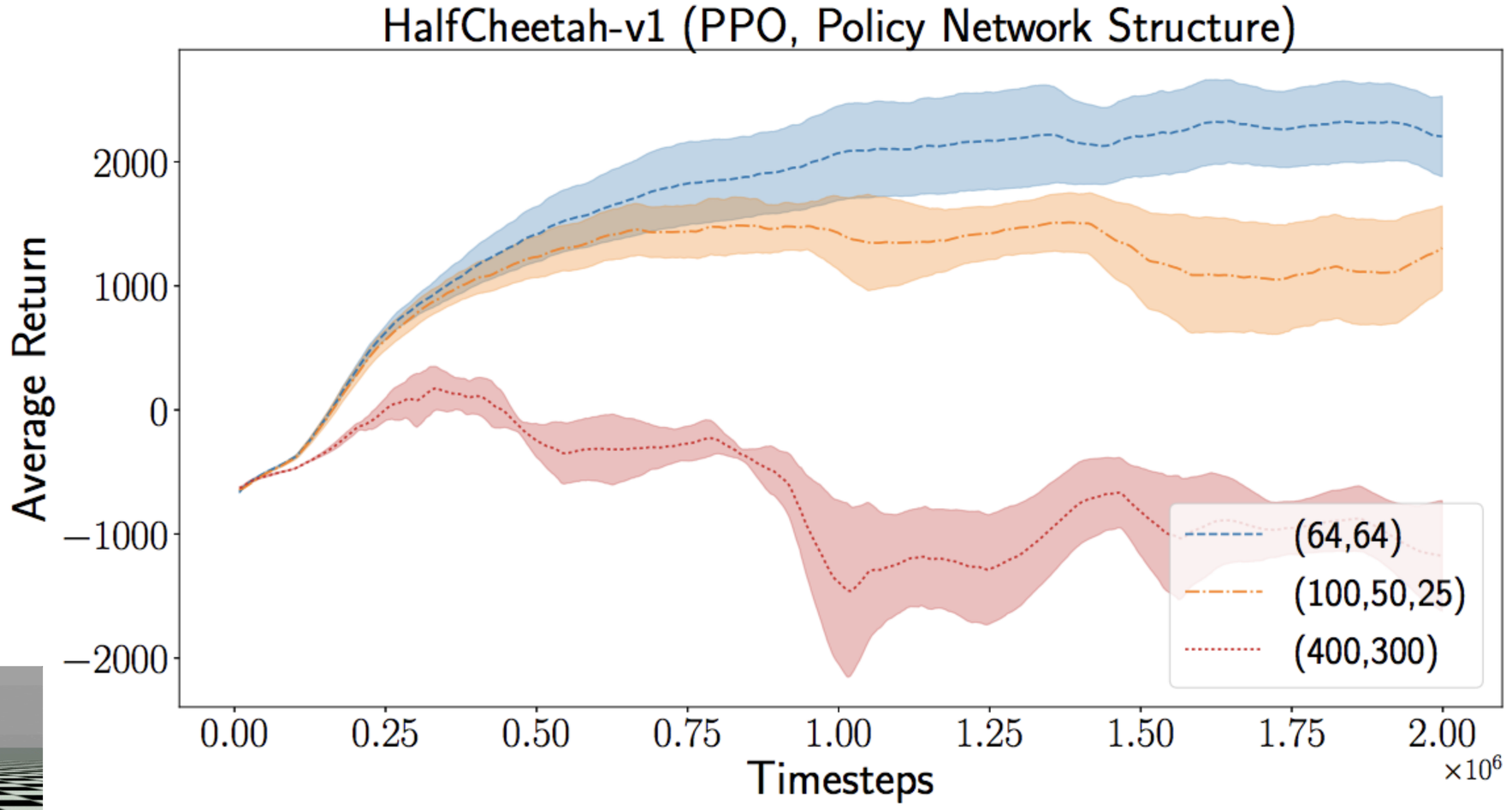
Featurization	Streaming	Hyperparam Tuning	Training	Simulation	Model Serving

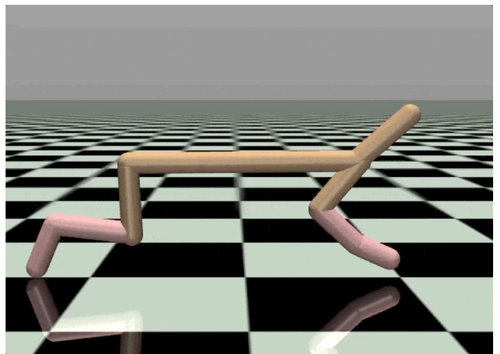
RAY

Tune - Hyperparameter Tuning



Hyperparameters Are Important for Performance



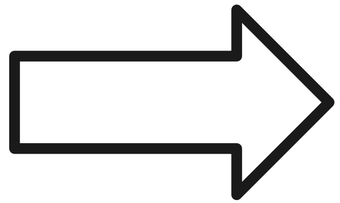


We Need a Framework for Tuning Hyperparameters

We want the best model

Resources are expensive

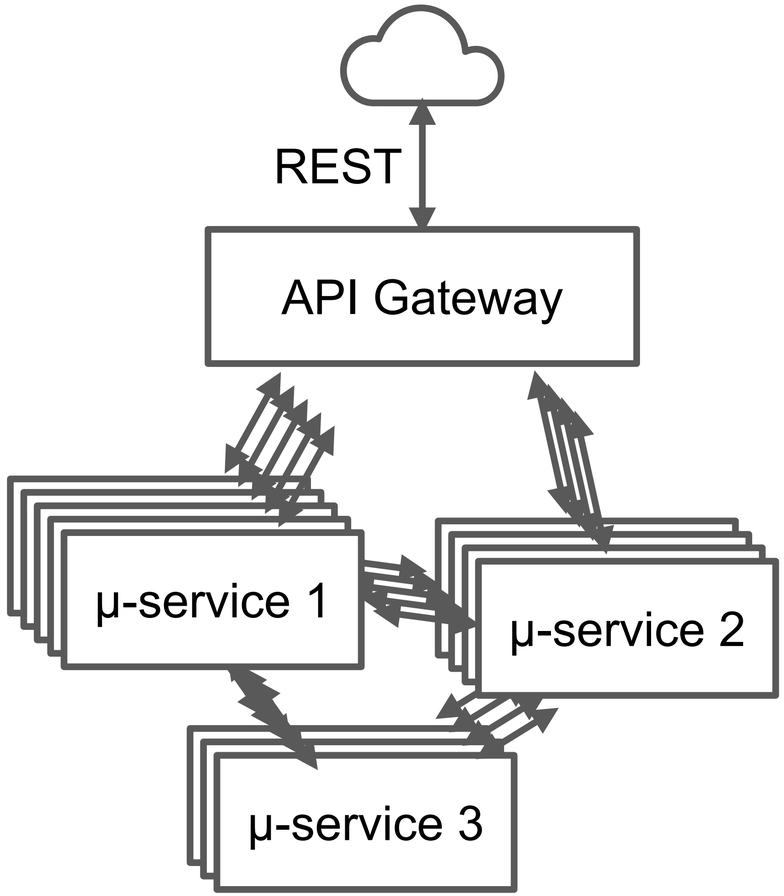
Model training is time-consuming



Ray for Microservices?

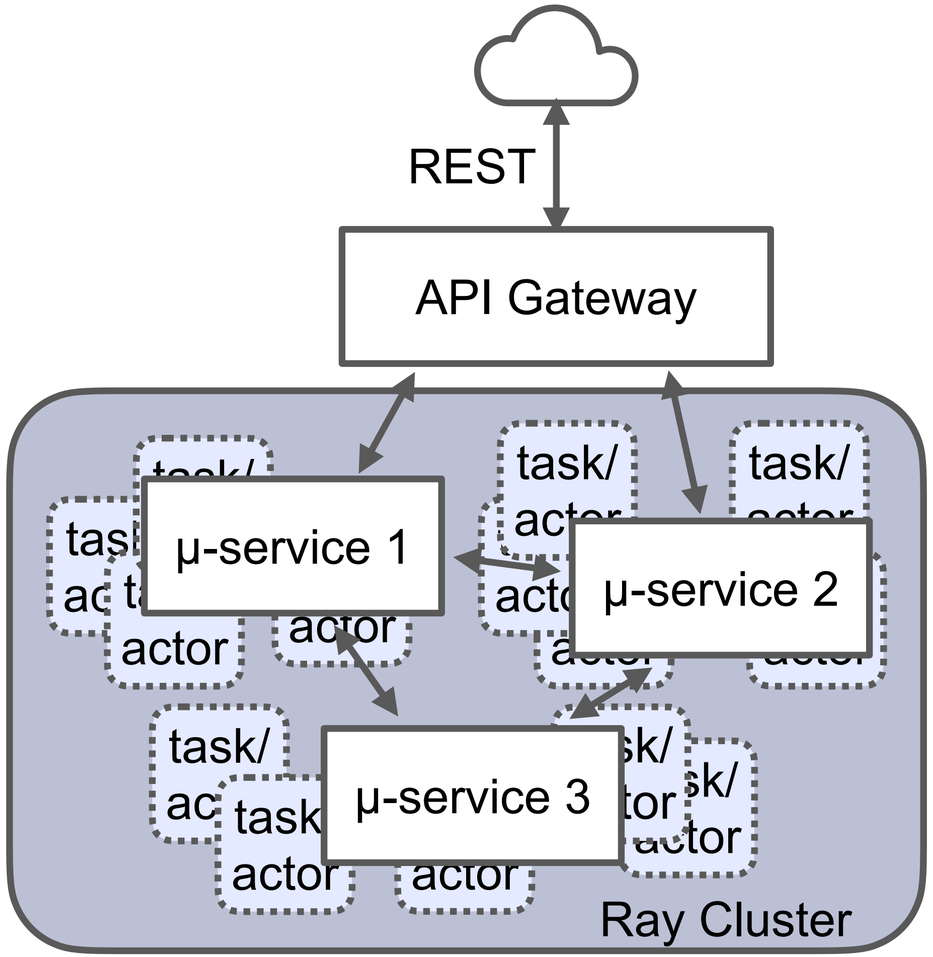
Separate Management

- Each team manages its own instances
- Each microservice has a different number of instances…
  - that have to be managed **explicitly**



Simplified Management

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



Questions?

ray.io

anyscale.com - We're Hiring!

anyscale.com/events

raysummit.org

dean@anyscale.com

anyscale