Ray: The Next Generation Compute Runtime for ML Applications

October 24, 2022

Zhe Zhang, Head of Open Source Engineering, Anyscale

anyscale

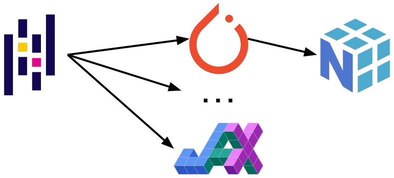
How do you manage your compute resources for Machine Learning?

anyscale

Scalability

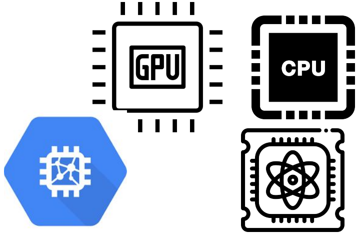
Simple & Flexible API

Unification



RAY

Heterogeneous Hardware



Simple yet Flexible API for Distributed Computing

anyscale

Minimalist API

RAY

ray.init()	Initialize Ray context.
@ray.remote	Function or class decorator specifying that the function will be executed as a task or the class as an actor in a different process.
.remote	Postfix to every remote function, remote class declaration, or invocation of a remote class method. Remote operations are *asynchronous*.
ray.put()	Store object in object store, and return its ID. This ID can be used to pass object as an argument to any remote function or method call. This is a *synchronous* operation.
ray.get()	Return an object or list of objects from the object ID or list of object IDs. This is a *synchronous* (i.e., blocking) operation.
ray.wait()	From a list of object IDs returns (1) the list of IDs of the objects that are ready, and (2) the list of IDs of the objects that are not ready yet.

Ray in a nutshell

Functions → Tasks : stateless computations (essentially RPC)

Classes → Actors : stateful computations

Distributed futures : enables parallelism

In-memory object store: enables passing args/results by reference

Extension of existing languages, rather than a new language

Function

```
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)
```

Class

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

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

Function → **Task**

```python
@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)
```

Class → **Actor**

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

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

Function → **Task**

```python
@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)
```

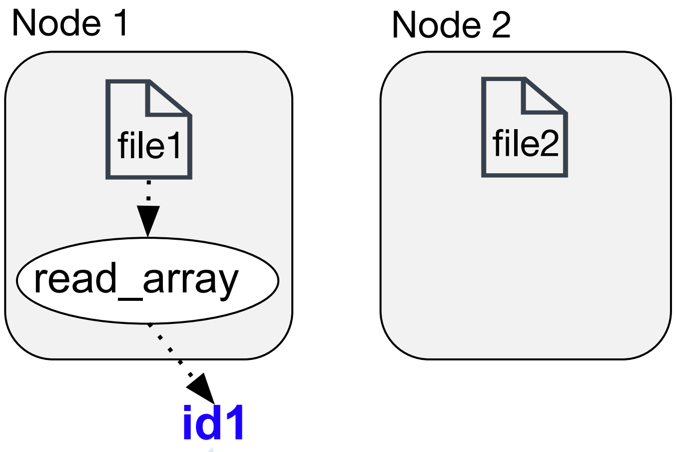
Object → **Actor**

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

c = Counter.remote()
id4 = c.inc.remote()
id5 = c.inc.remote()
```

Task API

id1: distributed future (object id)

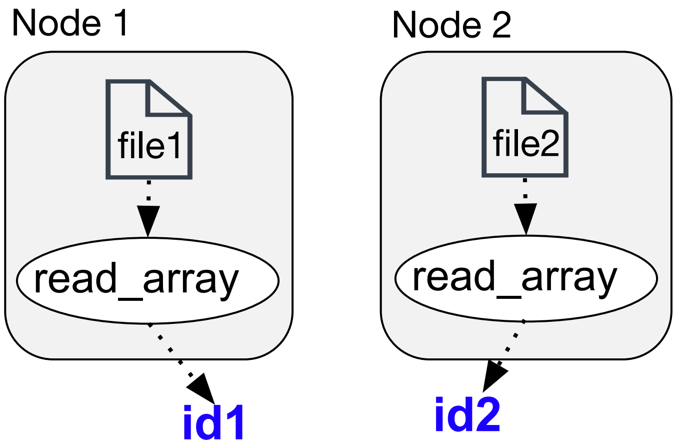
```
@ray.remote
def read_array(file):
    # read ndarray “a”
    # from “file”
    return a

@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)
```



Return future id1; before read_array() finishes

Task API

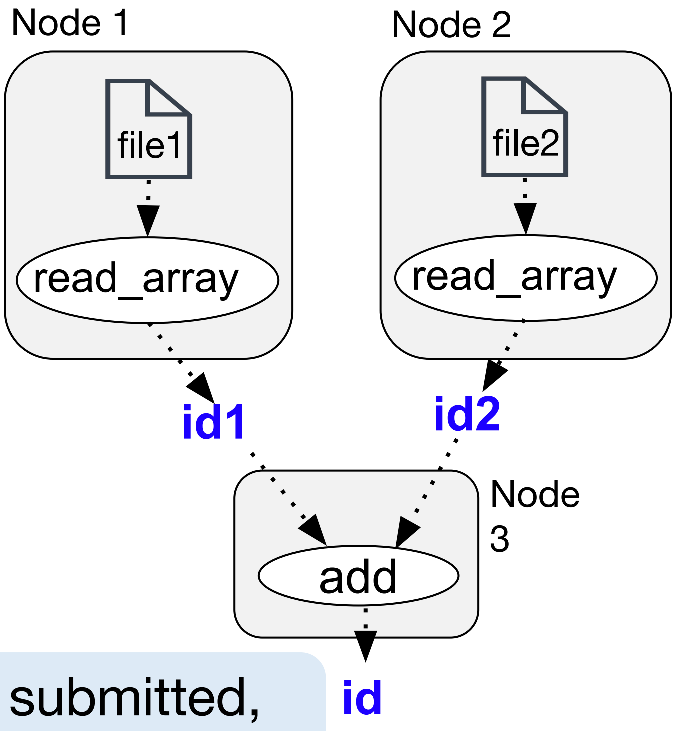
```
@ray.remote
def read_array(file):
    # read ndarray “a”
    # from “file”
    return a

@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

Task API

```
@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)
```

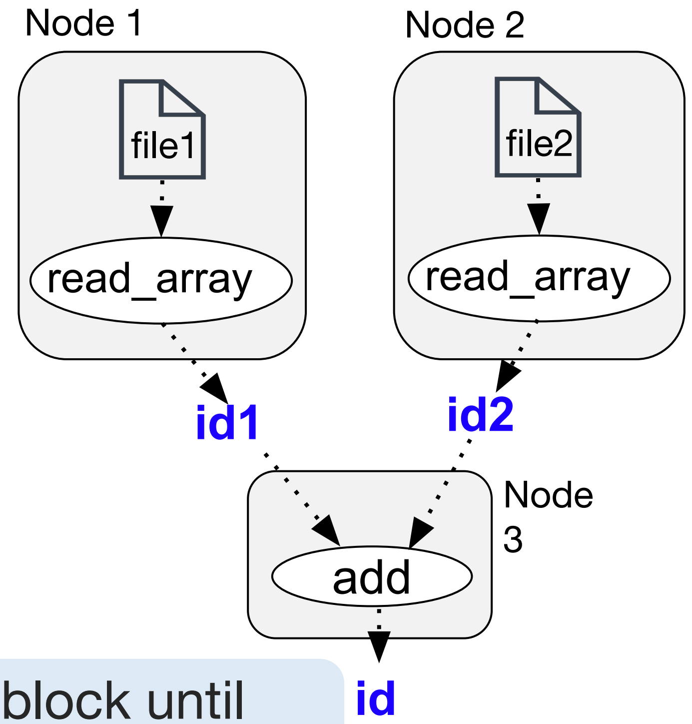


Every task submitted, but not finished yet

Task API

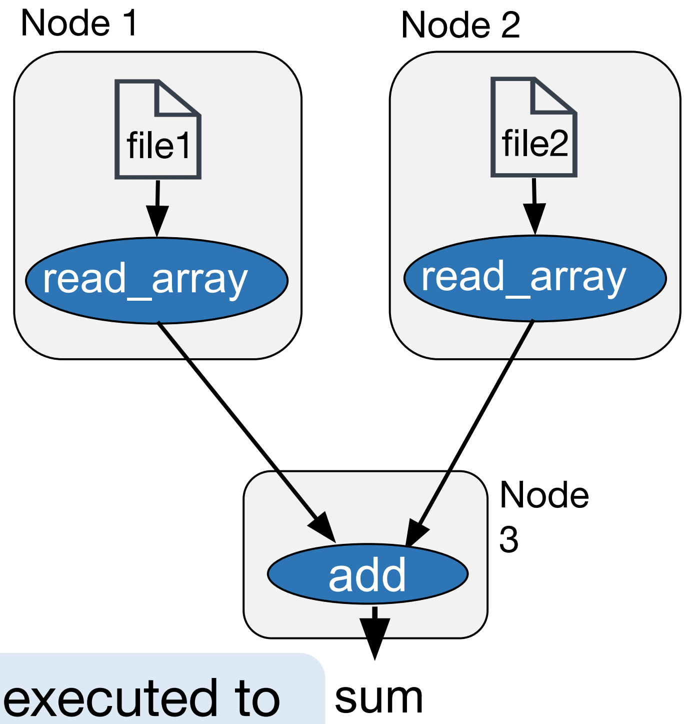
```
@ray.remote
def read_array(file):
    # read ndarray "a"
    # from "file"
    return a

@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



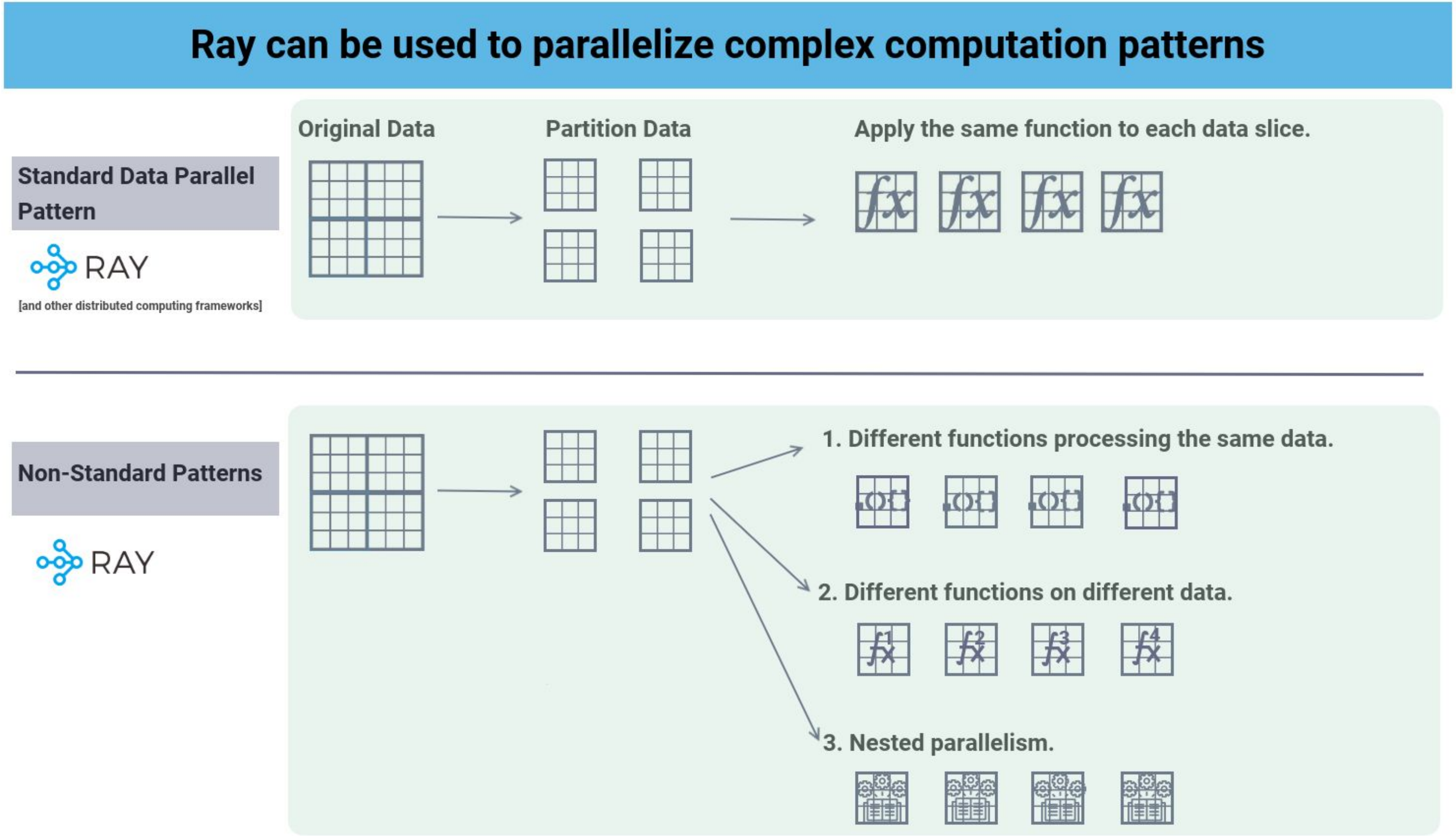
Task API

```
@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)
```

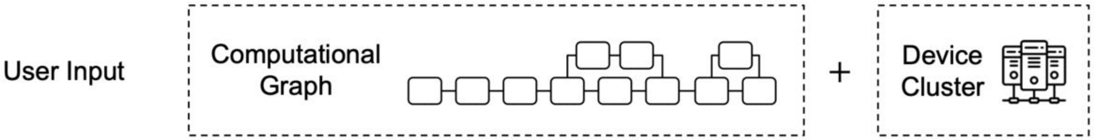


Task graph executed to compute sum

Using Ray's API for Data Parallelism



Advanced Usage of Ray’s API



Alpa

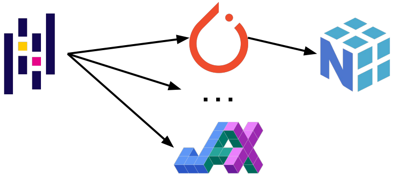
Source:
https://ai.googleblog.com/2022/05/alpa-automated-model-parallel-deep.html

Scalability

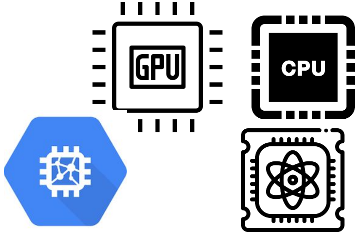
Simple & Flexible API

RAY

Unification



Heterogeneous Hardware

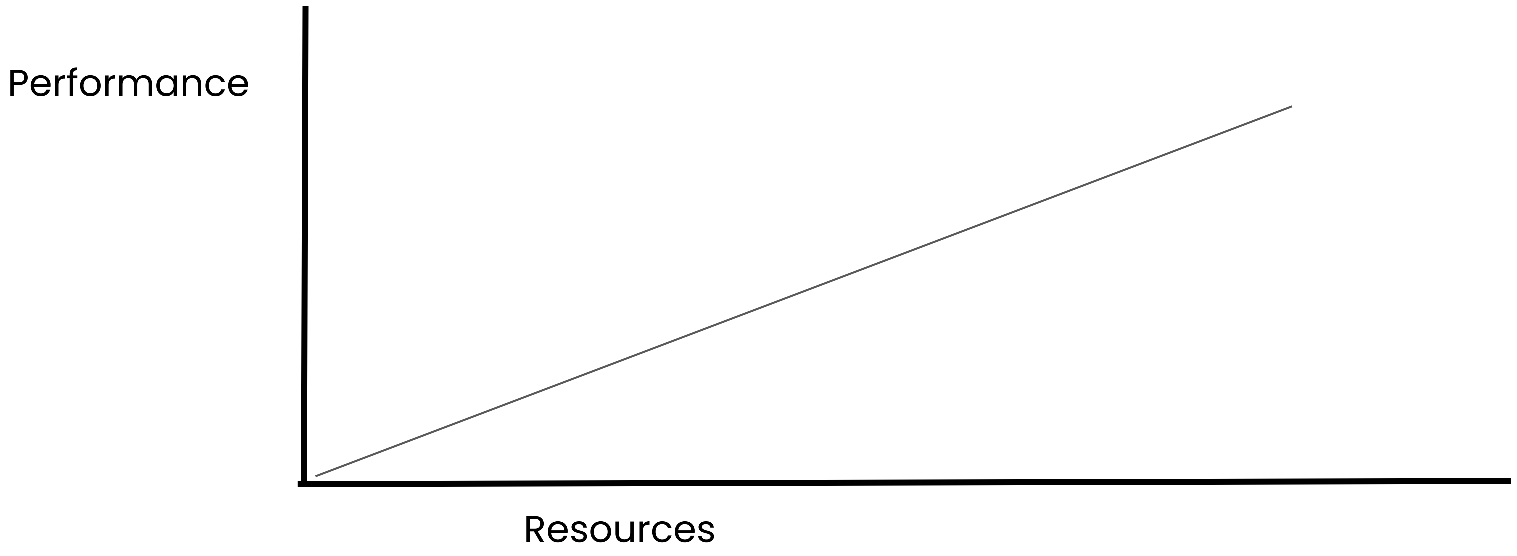


Scalability

Scalability

=

Removing non-linear factors



“We looked at a half-dozen distributed computing projects, and Ray was by far the winner. … We are **using Ray to train our largest models**. It’s been very helpful to us to be able to scale up to unprecedented scale. … **Ray owns a whole layer, and not in an opaque way**.”

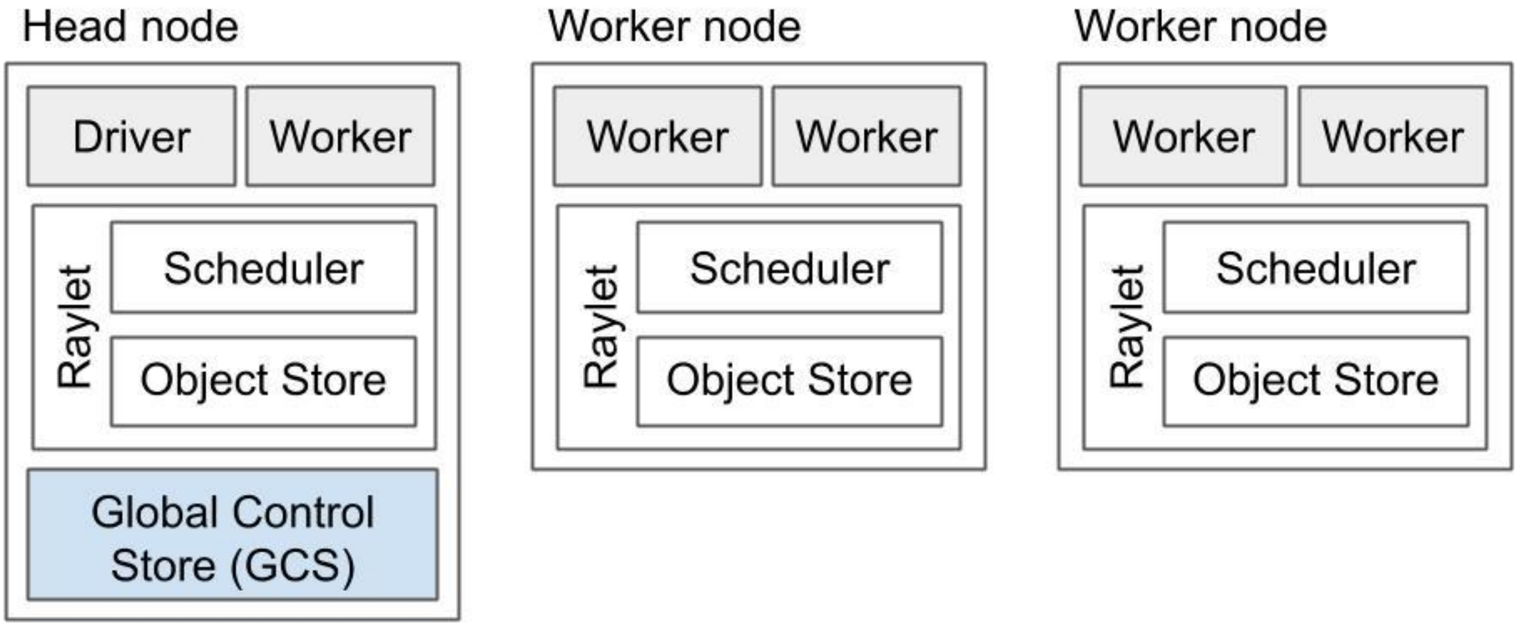
– Greg Brockman, President, OpenAI

Removing Non-linear Factors

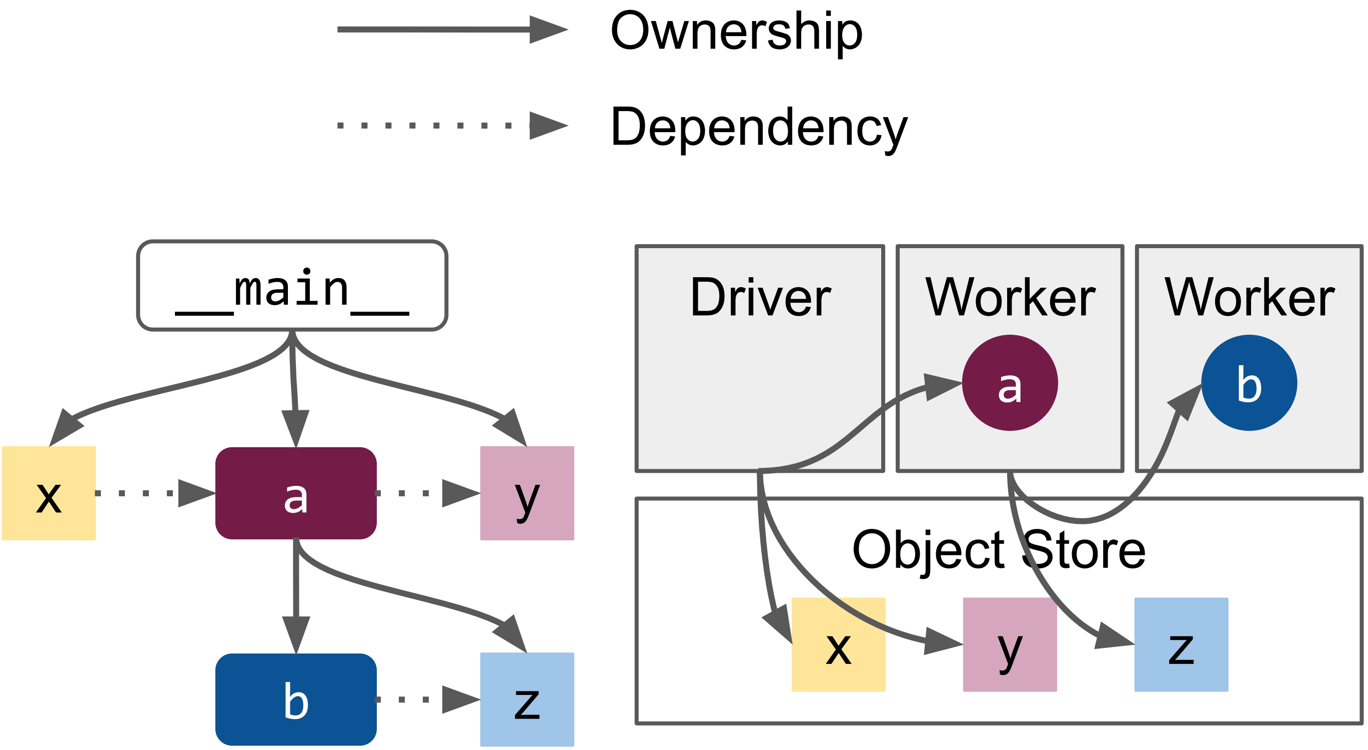
Flexible Fault Tolerance

Global Distributed Scheduling

Scalability – A Ray Cluster Looks Like...



```python
@ray.remote
def b():
  return

@ray.remote
def a(dep):
  z = b.remote()

x = ray.put(...)
y = a.remote(x)
```



Program **Task graph** **Physical execution**

Ray’s Fault Tolerance Model

Tasks: by default retry 3 times

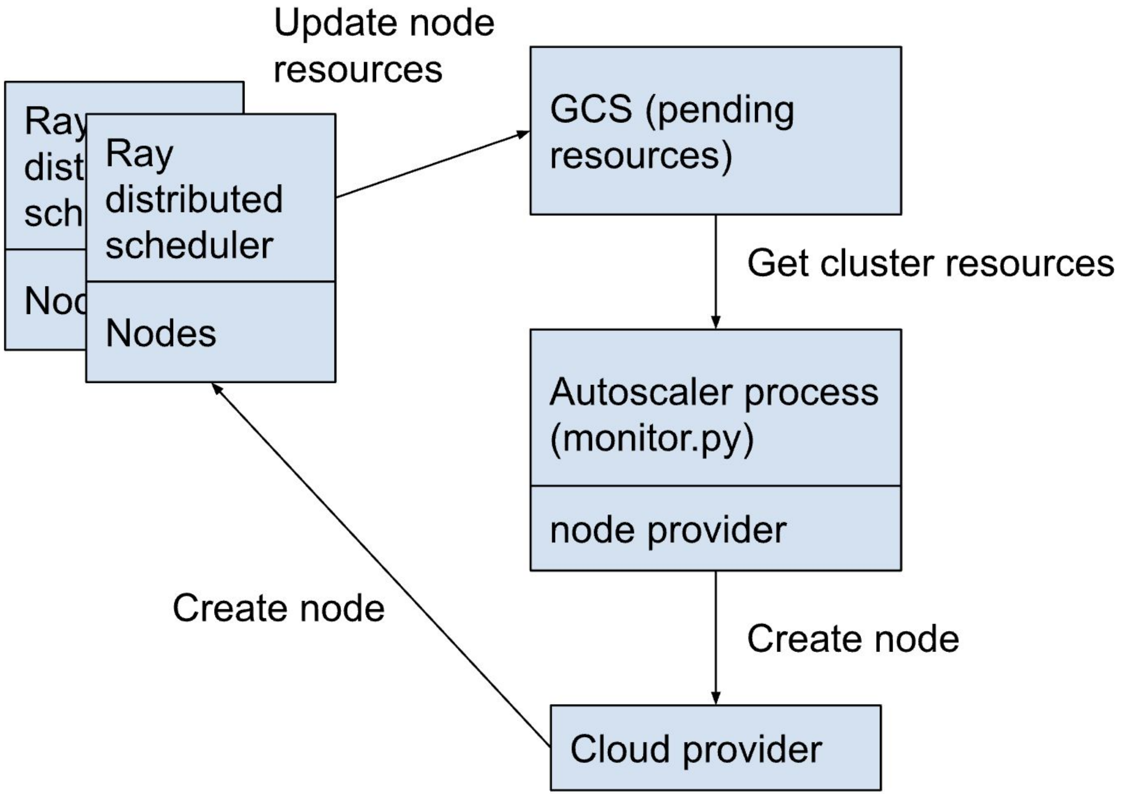
```
@ray.remote (max_retries=1)
def read_array(file):
```

Actors: by default not restarted. But behavior can be customized.

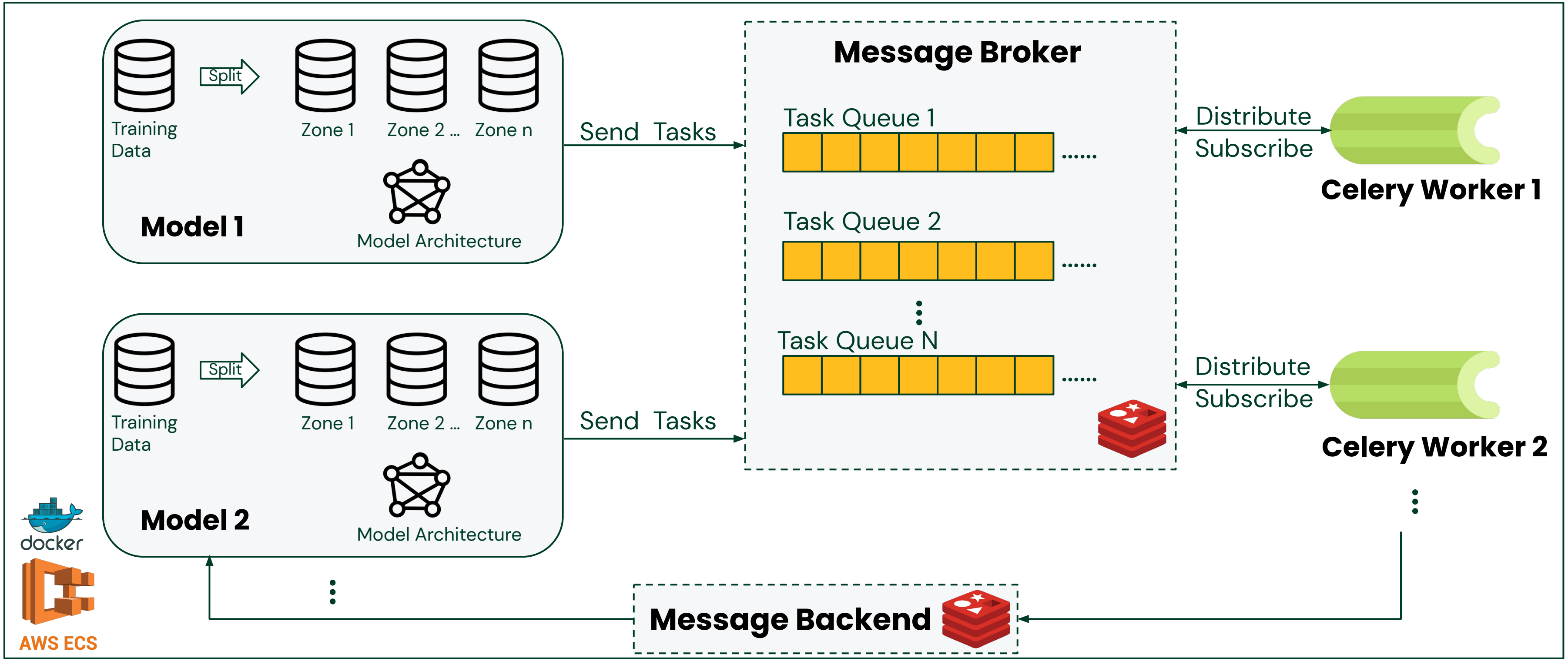
```
@ray.remote (max_restarts=5)
class Counter(object):
```

Objects: lineage-based reconstruction

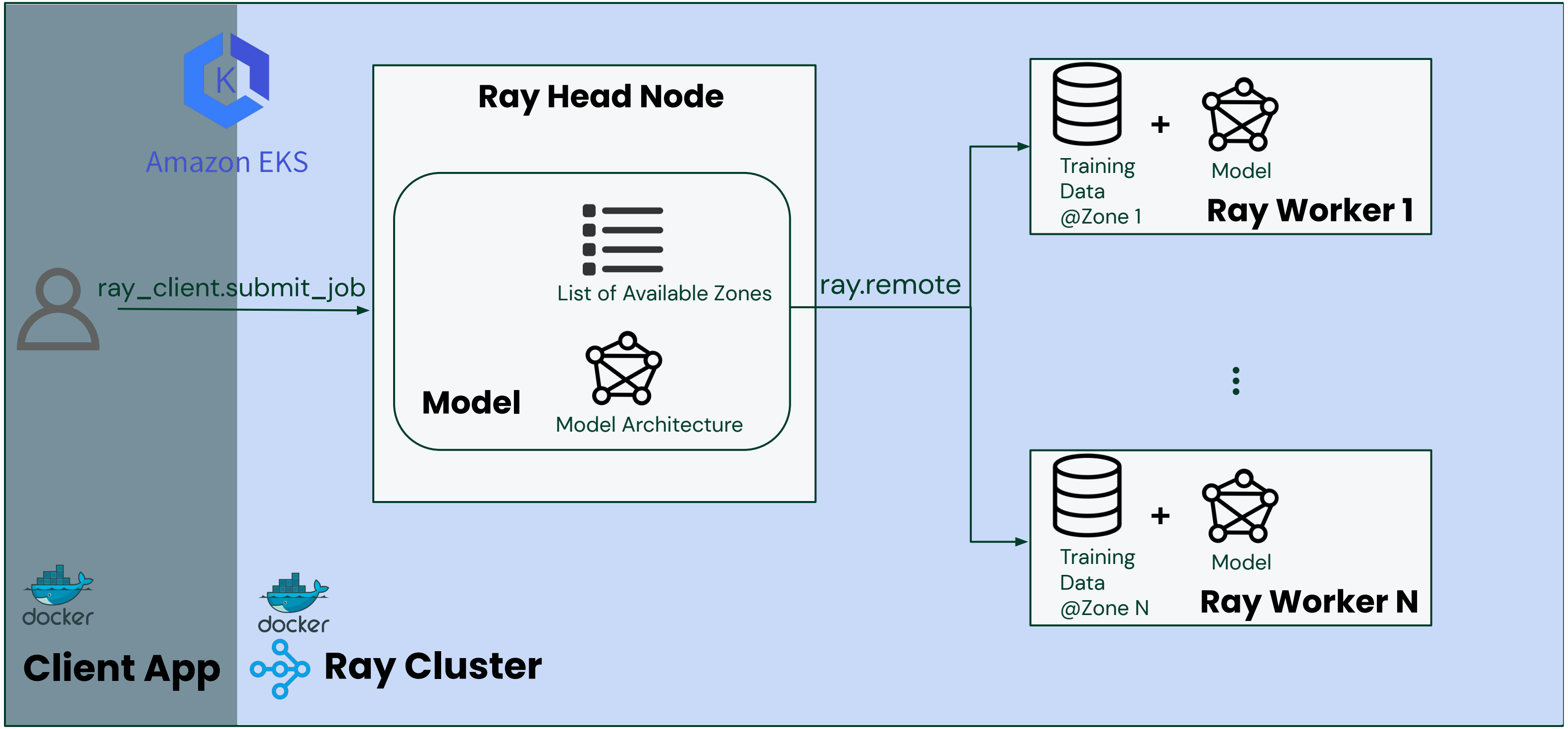
Ray’s Autoscaler



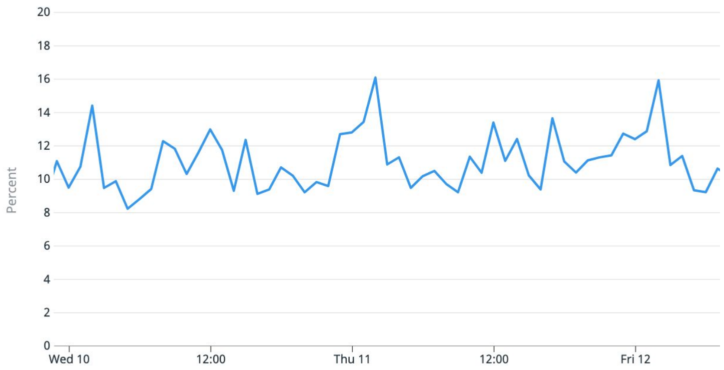
Existing Solution: Celery



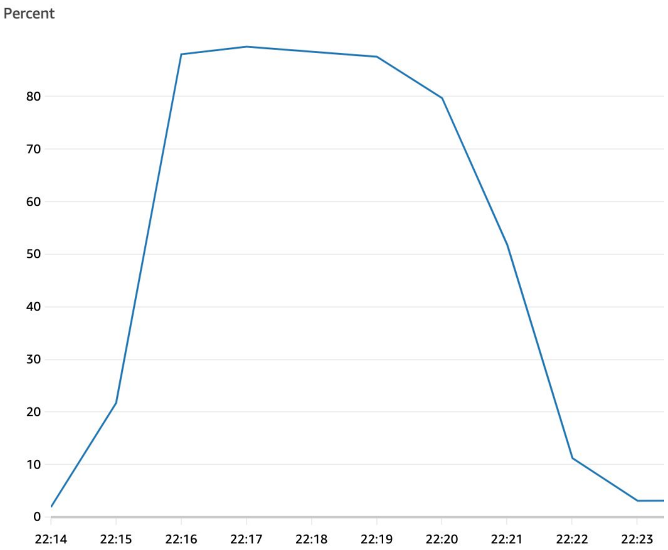
Proposed Solution: Ray



Results: Better Utilization



Before

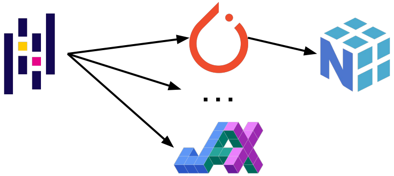


After

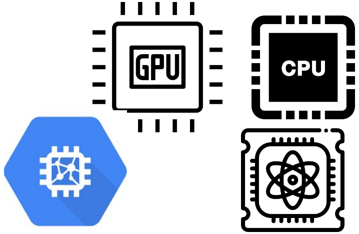
Scalability

Simple & Flexible API

Unification



Heterogeneous Hardware

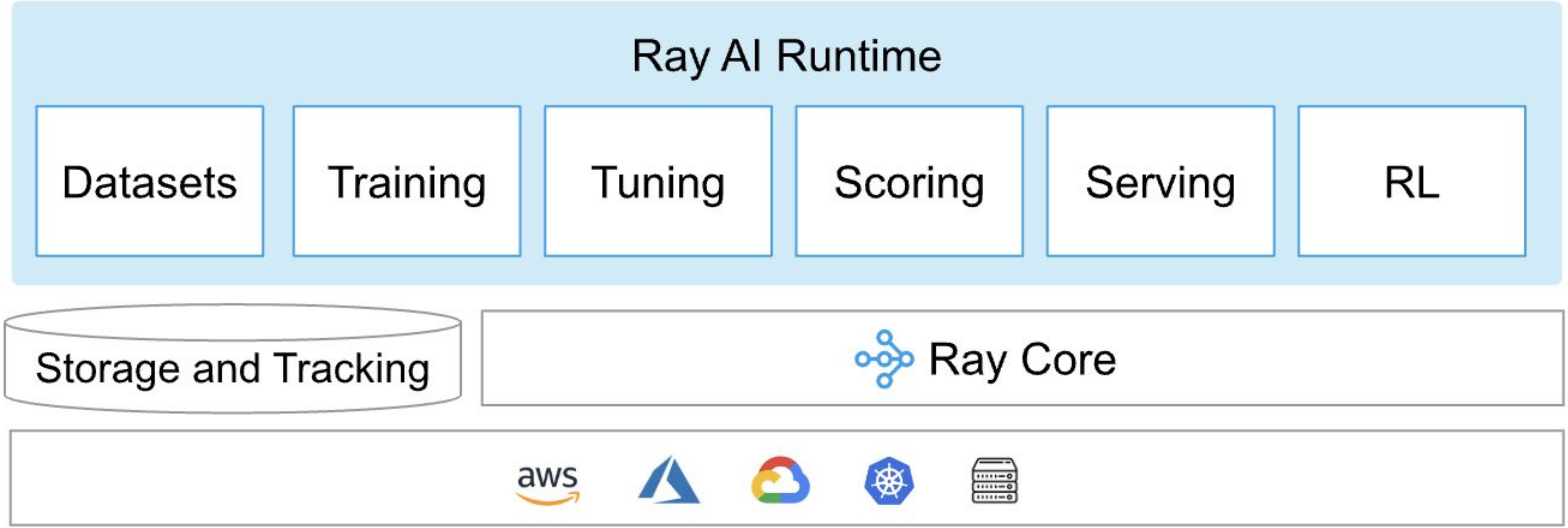


RAY

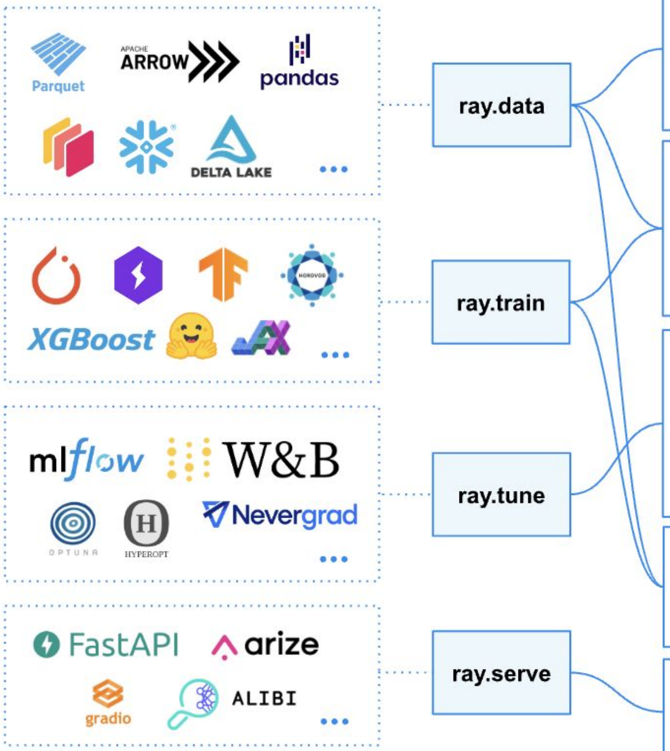
Unification – One Substrate for ML Compute

anyscale

Ray AI Runtime



Ray AI Runtime




```
dataset = ray.data.read_datasource(...)
train_ds, valid_ds = train_test_split(
    dataset, test_size=0.3)
test_ds = valid_ds.drop_columns(["target"])
preprocessor = StandardScaler(columns=["mean radius"])
```

```
trainer = ray.train.xgboost.XGBoostTrainer(
   scaling_config=ScalingConfig(num_workers=128),
   label_column="target",
   datasets=dict(train=train_ds, valid=valid_ds},
   preprocessor=preprocessor)
result = trainer.fit()
```

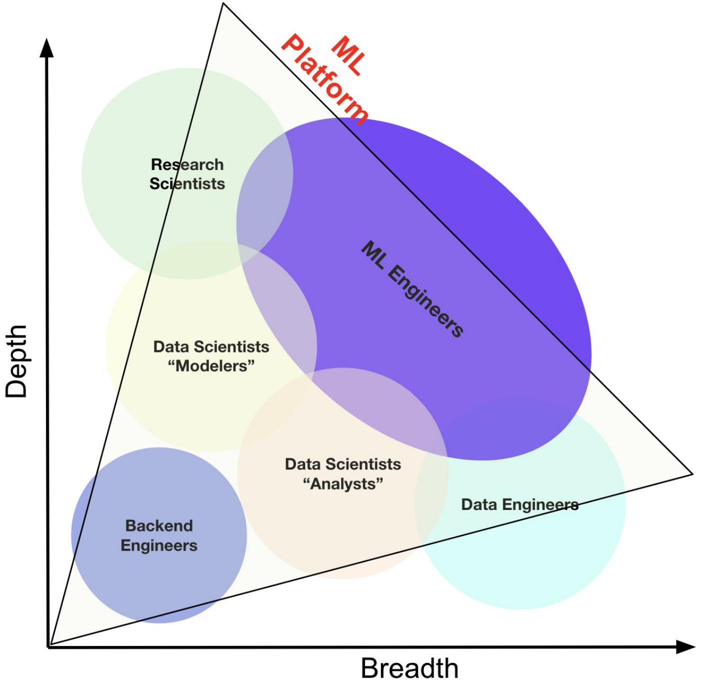
```
tuner = ray.tune.Tuner(
   trainer,
   param_space={"params": {"depth": tune.randint(1, 9)}},
   tune_config=TuneConfig(
      num_samples=5, metric="logloss", mode="min"))
checkpoint = tuner.fit().get_best_result().checkpoint
```

```
batch_predictor = BatchPredictor.from_checkpoint(
    checkpoint, XGBoostPredictor)
predicted_probabilities = batch_predictor.predict(test_ds)
predicted_probabilities.show()
```

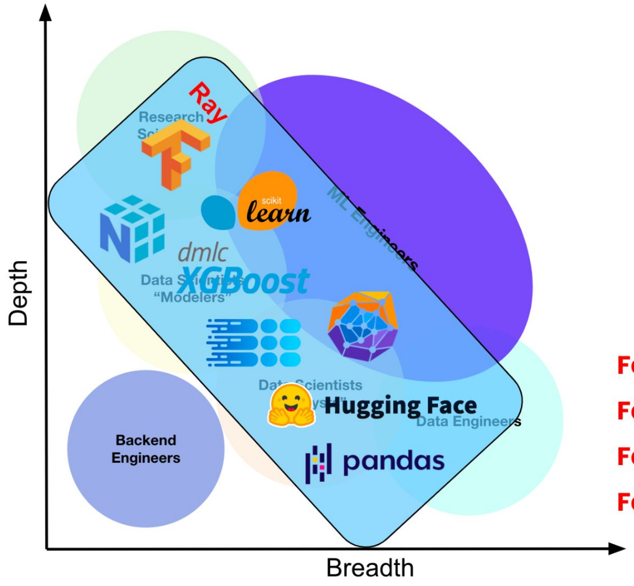
```
deployment = PredictorDeployment.options(name="XGB_Serve")
deployment.deploy(XGBoostPredictor, checkpoint, ...)
print(deployment.url)
```

Using Ray at Spotify

Where we want to be



What Ray's Ecosystem can help



Familiar tools

Faster project start time

Faster prototyping

Faster experimentation

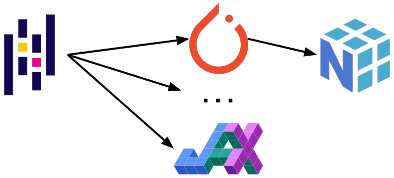
Sample of Companies Who Use Ray in their Machine Learning Platform

RIDECELL	lyft	Uber	cruise	instacart
Meta	Google	OpenAI	Microsoft	
ANT GROUP	shopify	amazon	STITCH FIX	
RIOT GAMES	NETFLIX	Spotify	ByteDance	
DOW	intel	IBM	Hugging Face	co:here

Scalability

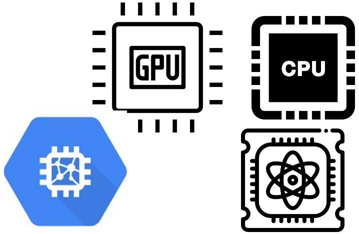
Simple & Flexible API

Unification



RAY

Heterogeneous Hardware



Heterogeneous Hardware

anyscale

Function → **Task**

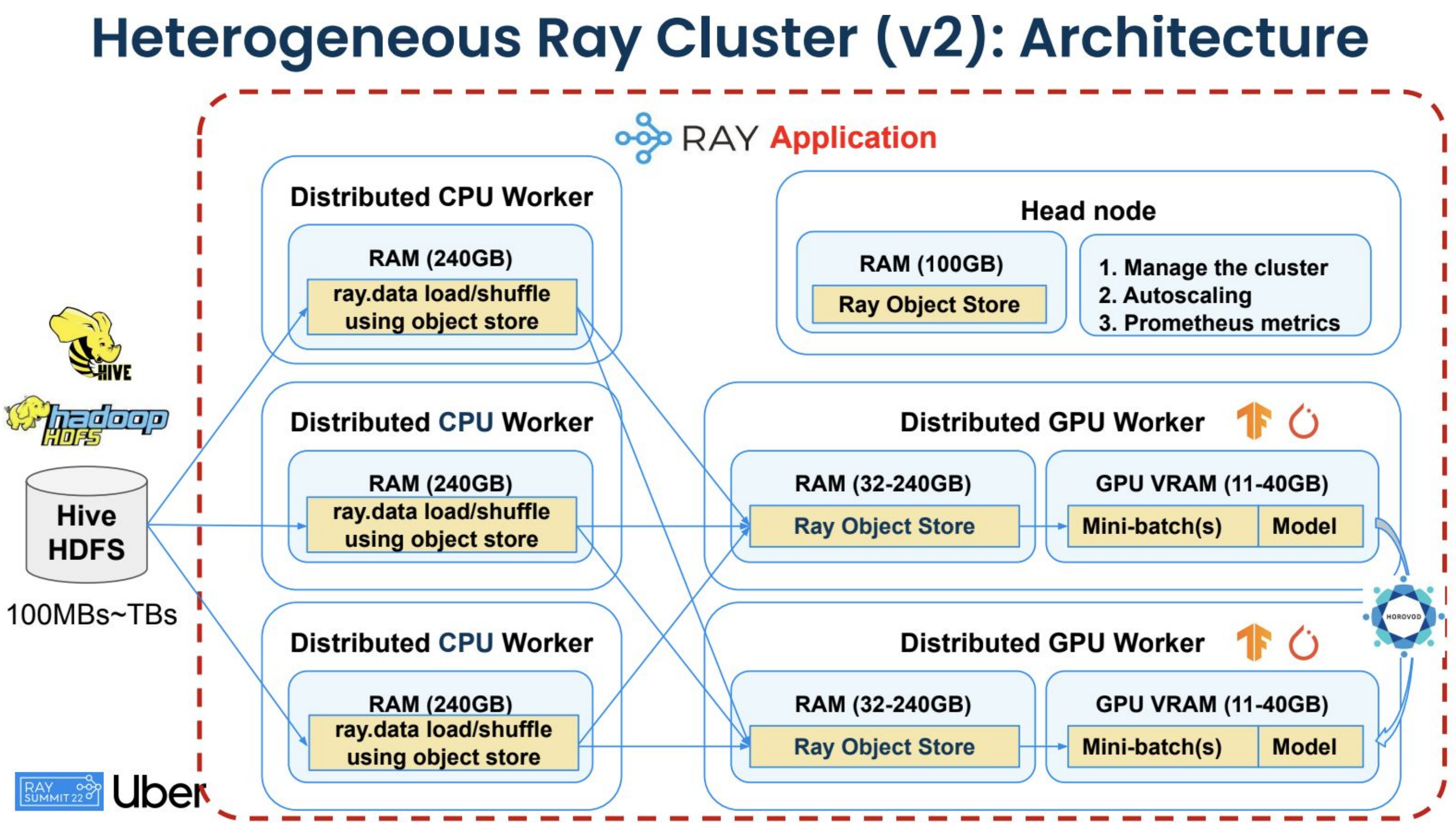
```python
@ray.remote
def read_array(file):
    # read ndarray "a"
    # from "file"
    return a

@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)
```

Class → **Actor**

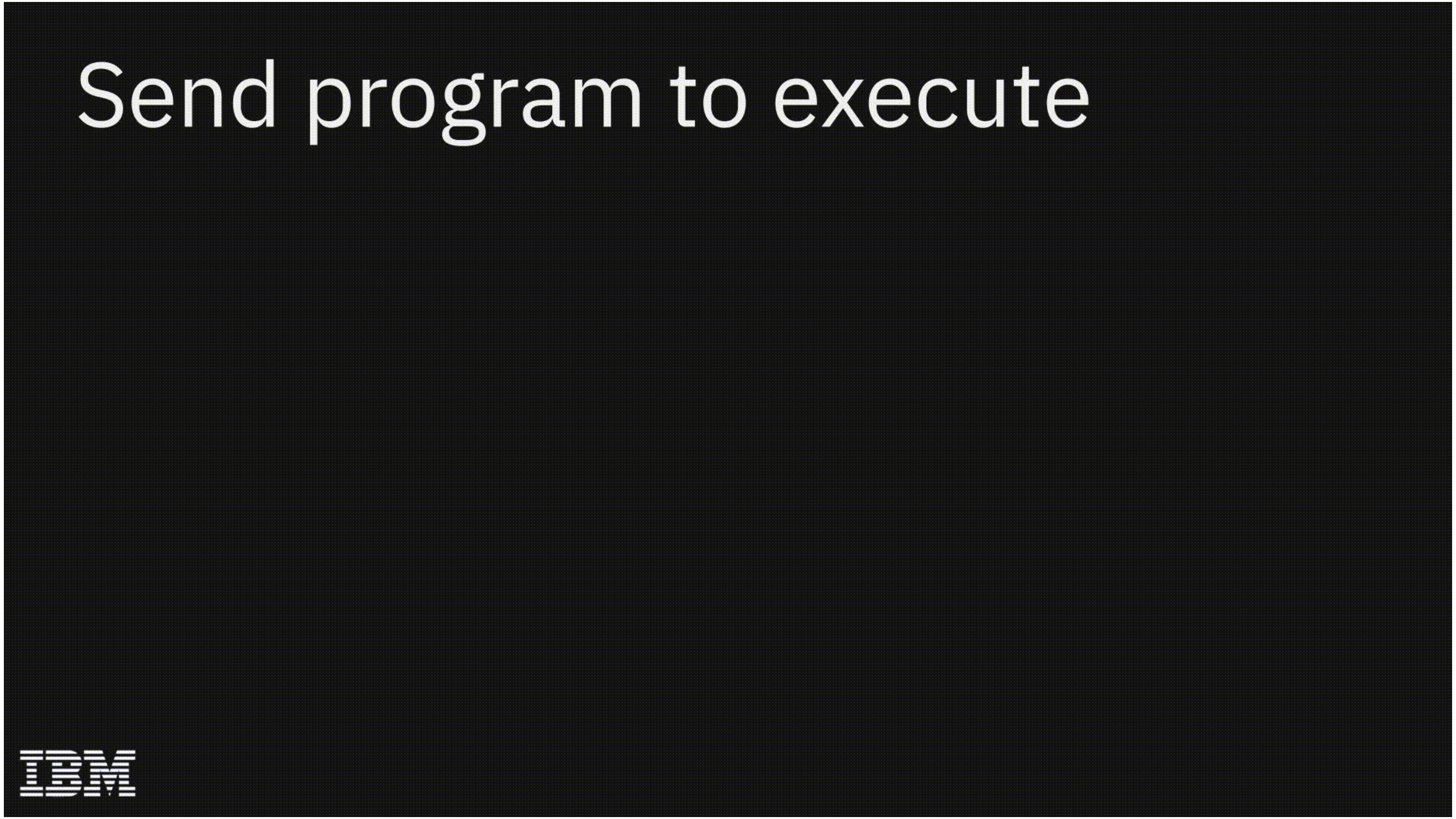
```python
@ray.remote(num_cpus=2, num_gpus=1)
class Counter(object):
        self.value += 1
        return self.value

c = Counter.remote()
id1 = c.inc.remote()
id2 = c.inc.remote()
val = ray.get(id2)
```

can specify
resource demands;
support heterogeneous hardware

Deep Learning Cluster at Uber



IBM: Quantum + Classical Computing



Summary

Ray's **simple and flexible APIs** provides **easy access** to:

Scalable and **unified** distributed computing on **heterogeneous** hardware.

Call to action

- Get started at docs.ray.io
- Dive deeper on https://github.com/ray-project/ray/
- See the Ray Summit Program
- Stay tuned on our meetup series