Introduction to Ray for distributed & machine learning in Python

Jules S. Damji, Ray Team @Anyscale

X: @2twitme

LinkedIn: https://www.linkedin.com/in/dmatrix/

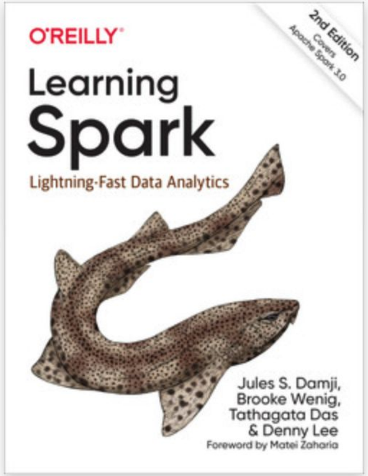
RAY

A quick poll...

RAY

$whoami

- Lead Developer Advocate, Anyscale & Ray Team
- Sr. Developer Advocate, Databricks, Apache Spark/MLflow Team
- Led Developer Advocacy, Hortonworks
- Held SWE positions:
    - Sun Microsystems
    - Netscape
    - @Home
    - Loudcloud/Opsware
    - Verisign



RAY

A few important URLs

Keep these URLs open in a browser tab:

- GitHub Learning Material: https://bit.ly/devaiworld23
- Ray Documentation: https://bit.ly/ray-core-docs

📅 Today's agenda

- Why & What's Ray & Ray Ecosystem
- Ray Architecture & Components
- Ray Core Design & Scaling Patterns & APIs
- Modules [1]: Hands-on in class
- Wrap up…

Why Ray + What's Ray

RAY

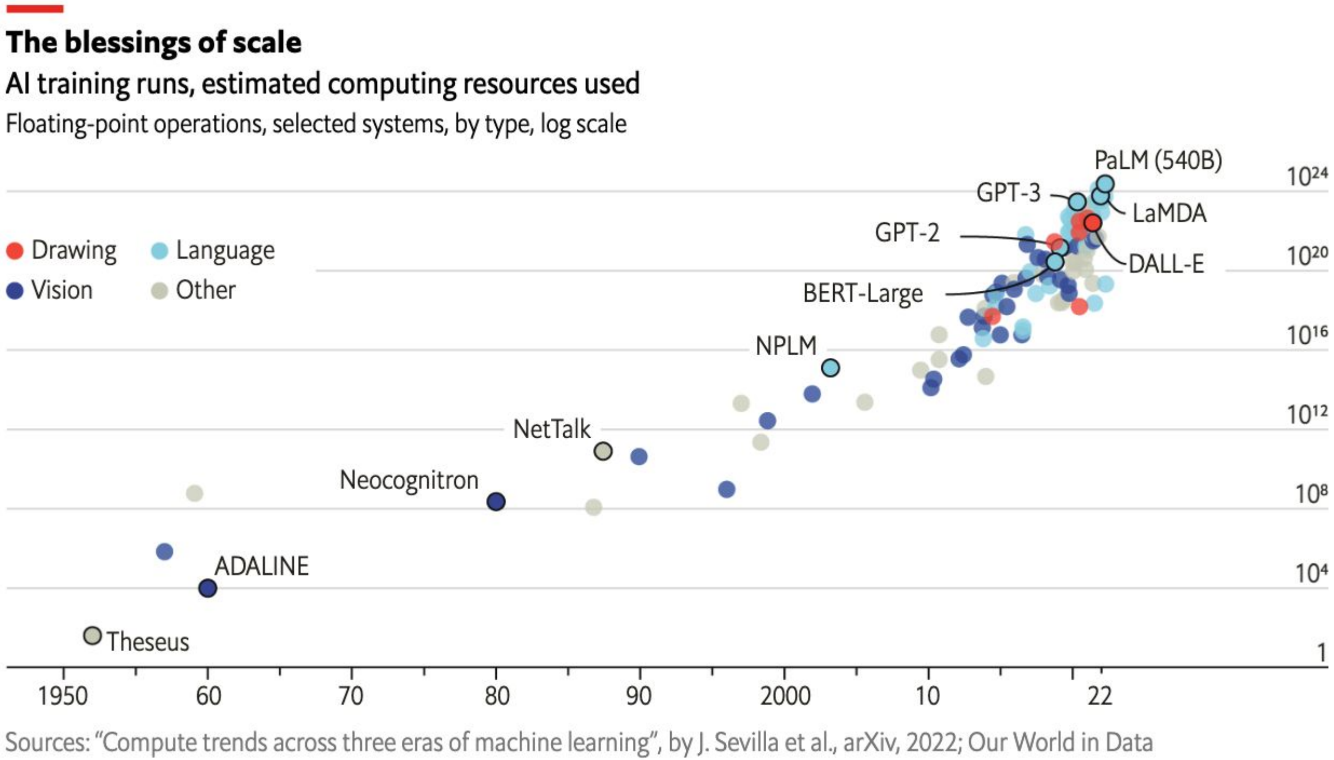
Why Ray?

Machine learning is pervasive

Distributed computing is a necessity

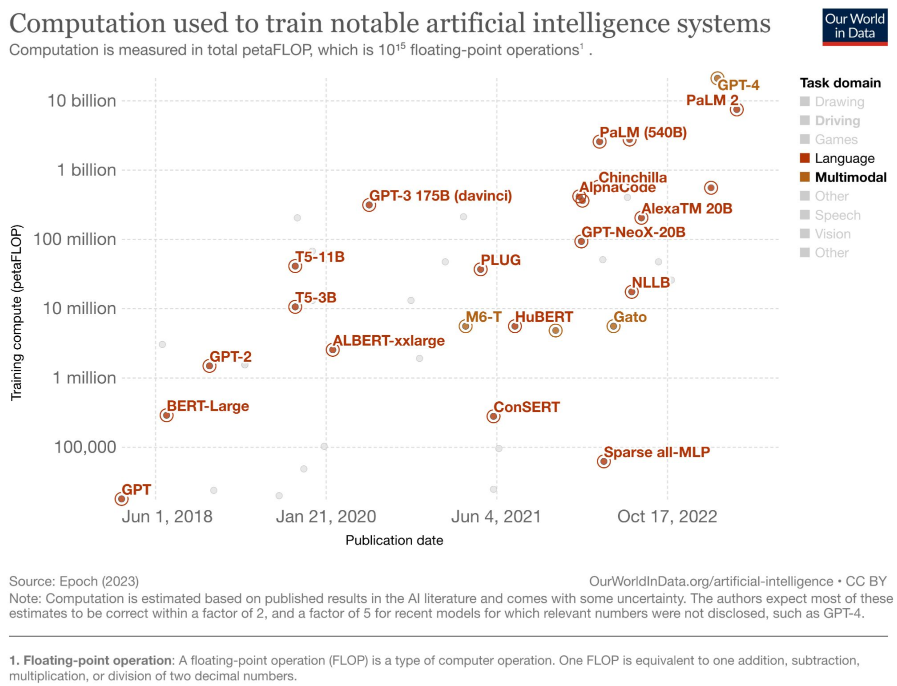
Python is the default language for DS/ML

Blessings of scale ....



RAY

Blessings of scale ….



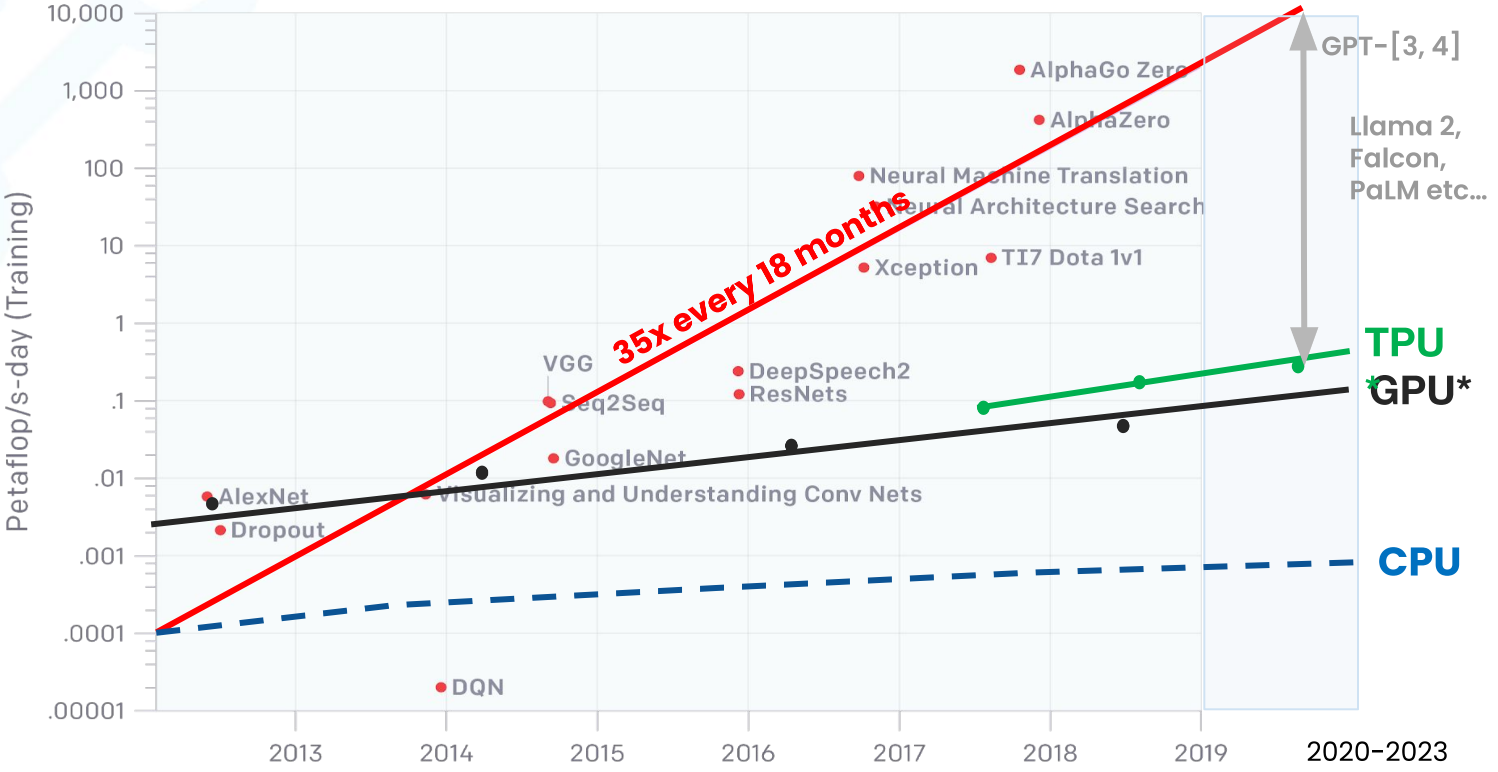
1. Model size are getting larger

- Model size is exponentially increasing.
- Models are too large to into a single GPU.
- We need to shard the models across multiple GPUs for training
  - e.g. ZeRO, Model Parallel, Pipeline Parallel

BERT(2019): 336M params(1.34GB)
Llama-2 (2023): 70B params(280GB) ~20x
GPT-4: ~1800B >5,000x

RAY

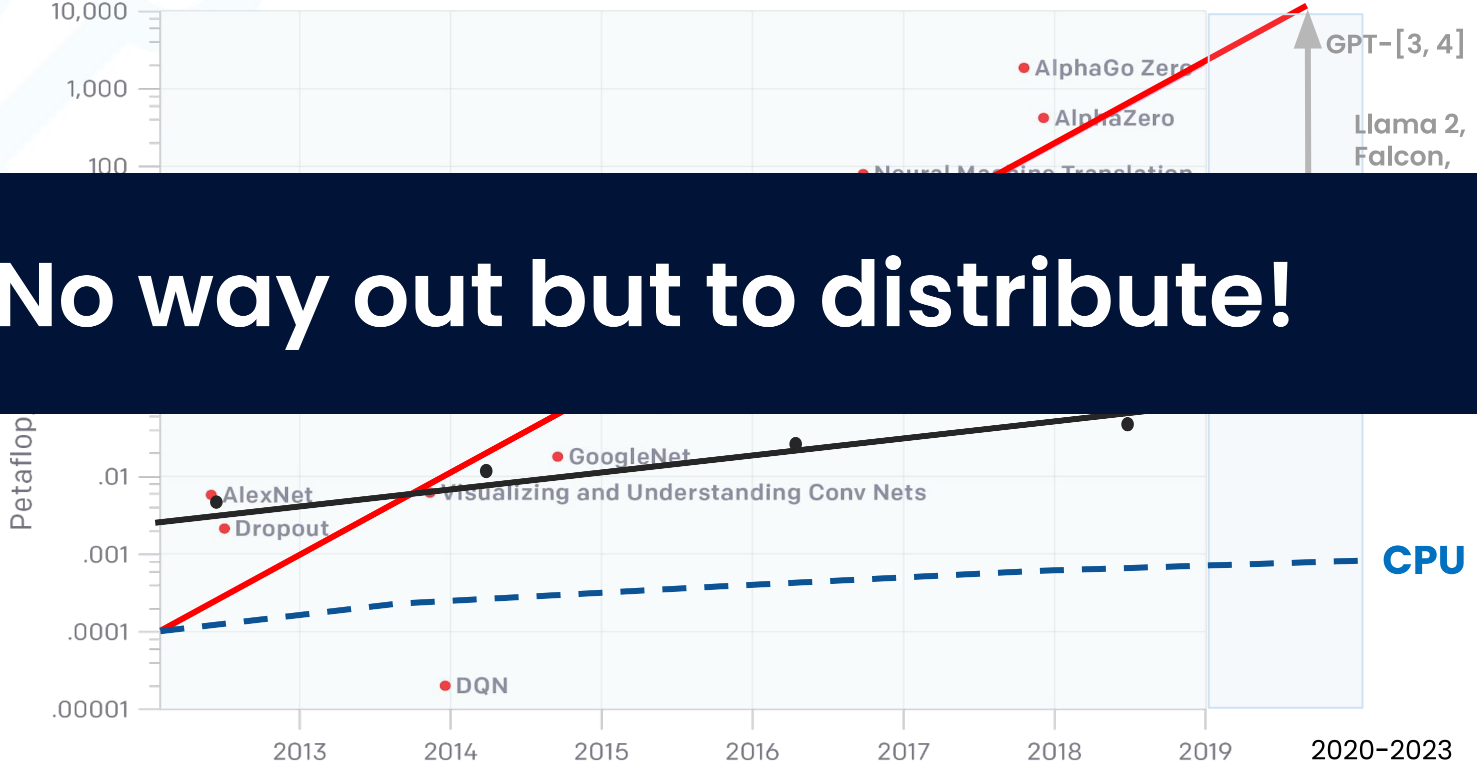
Supply demand-problem



https://openai.com/blog/ai-and-compute/

RAY

Supply demand-problem

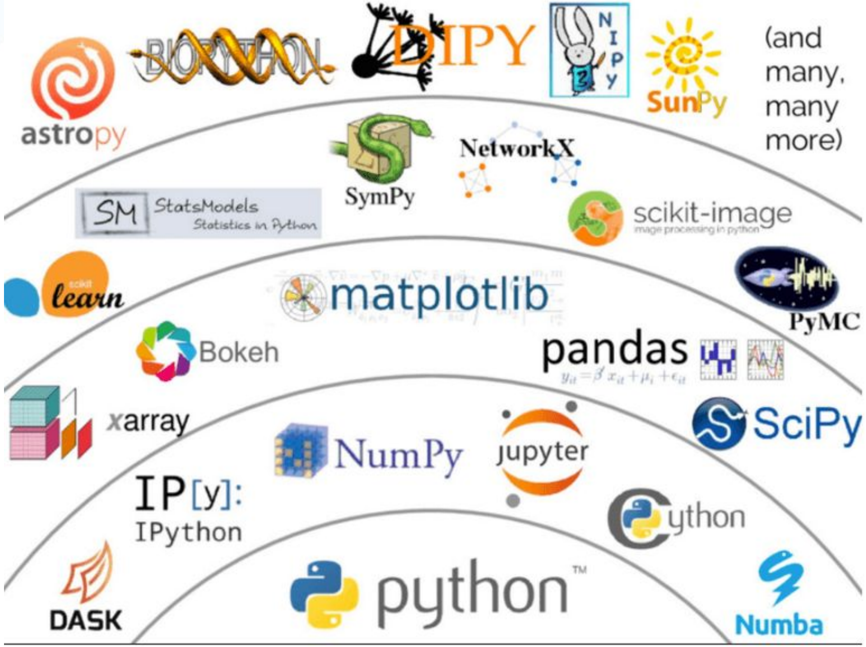


No way out but to distribute!

https://openai.com/blog/ai-and-compute/

RAY

Python DS/ML Ecosystem

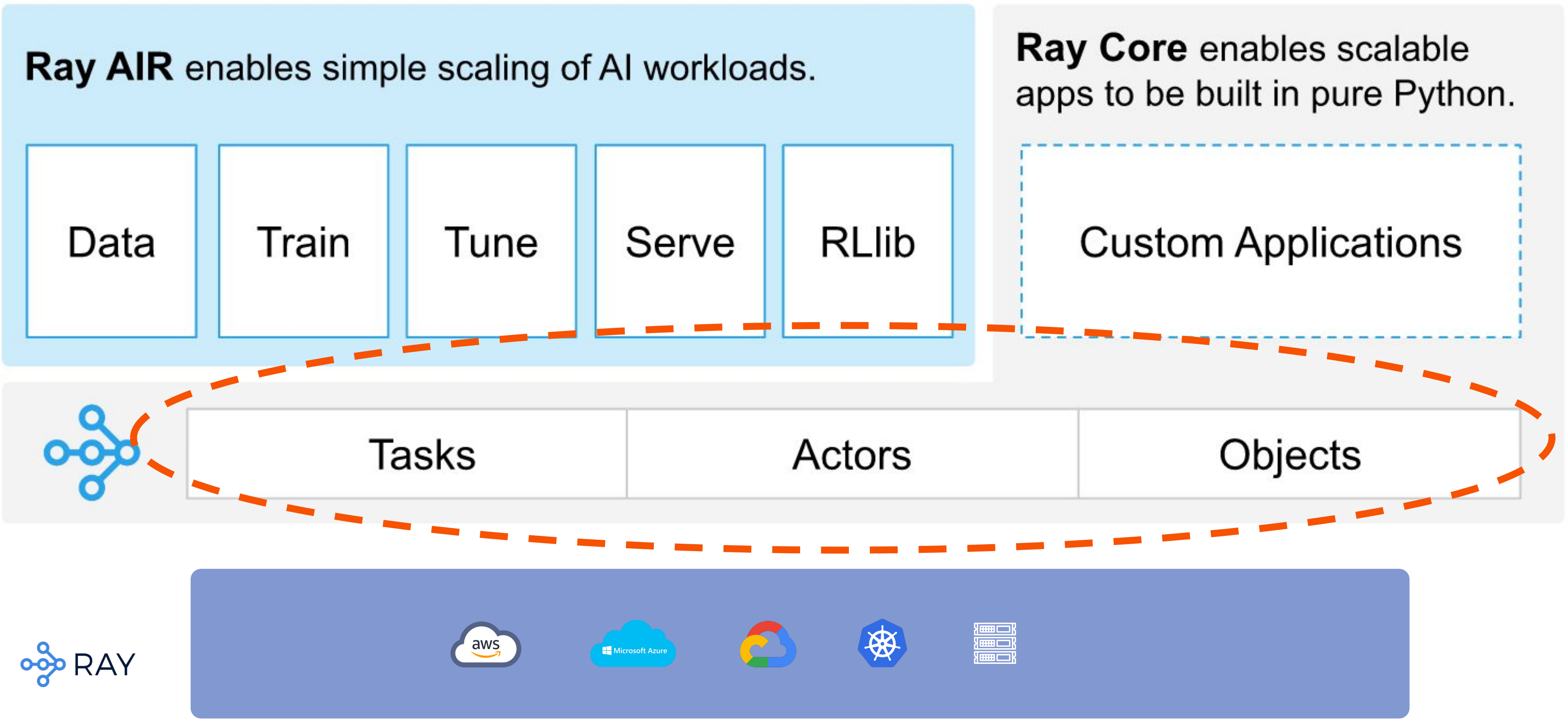


What's Ray ?

- A ***simple/general-purpose*** library for distributed computing
- An ecosystem of Python Ray AI libraries (for scaling ML & more)
- Runs on laptop, public cloud, K8s, on-premise
- Easy to install and get started …. *pip install ray[default]*

A layered cake of functionality and capabilities for scaling ML workloads

A layered cake and ecosystem



RAY

A Layered Cake and Ecosystem

Ray AI Libraries enable simple scaling of AI workloads.

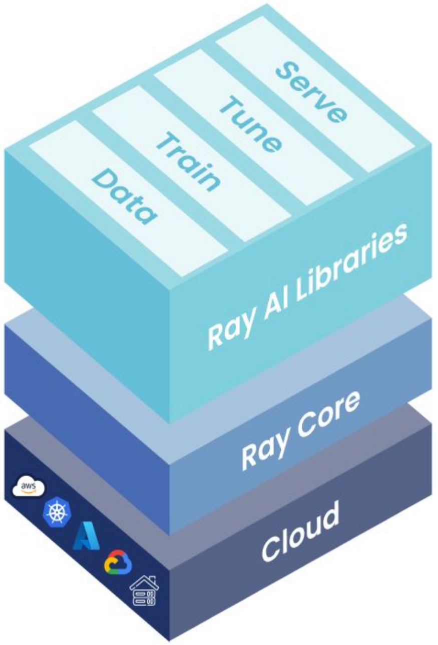
Data	Train	Tune	Serve	RLlib

Ray Core enables scalable apps to be built in pure Python.

Custom Applications

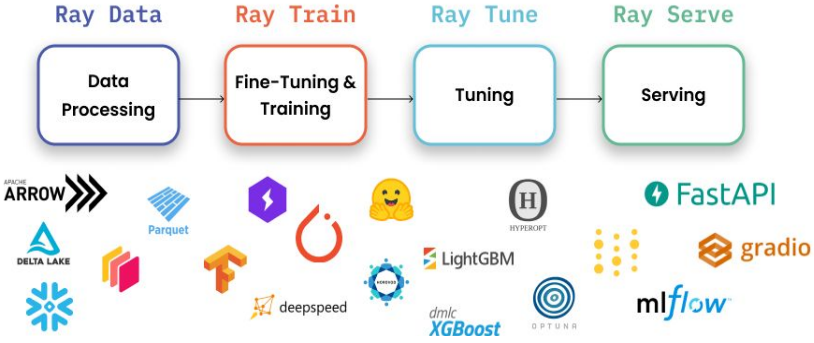
Tasks	Actors	Objects

AI libraries



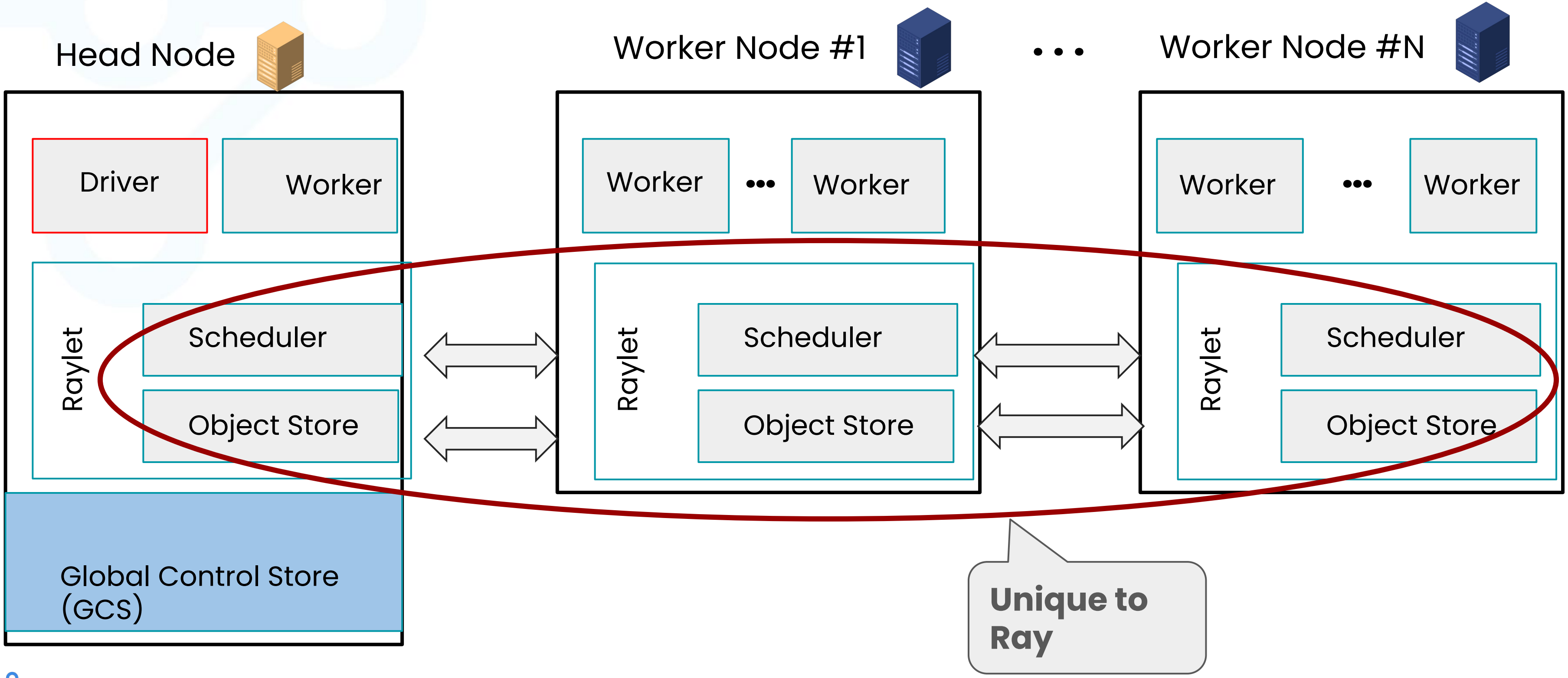
high-level libraries that enable simple scaling of AI workloads

a low-level distributed computing framework with a concise core and Python-first API



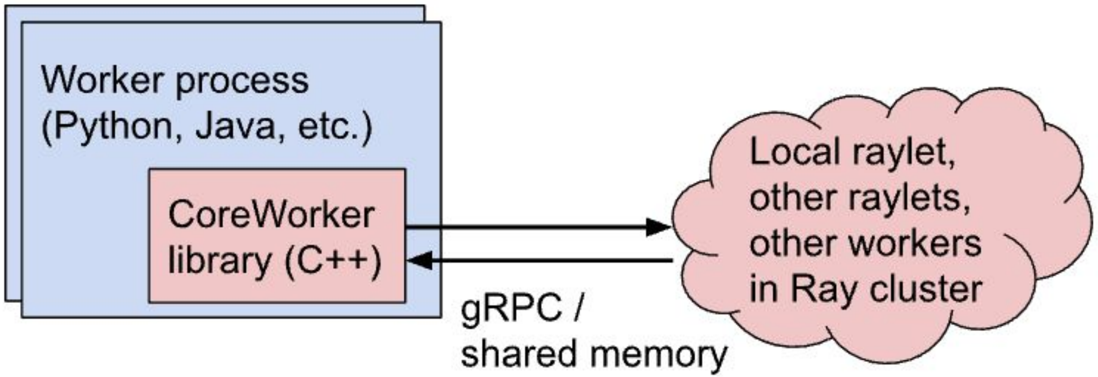
Ray architecture & components

Anatomy of a Ray cluster



RAY

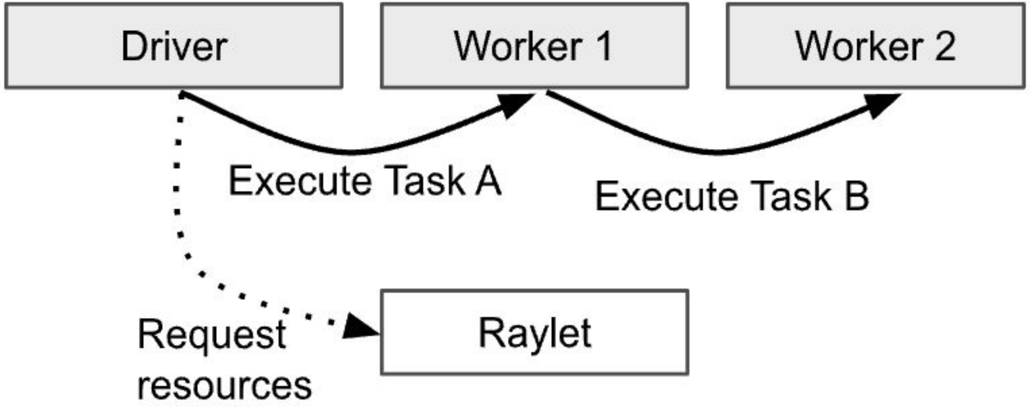
Anatomy of a Ray worker process



Ray workers interact with other Ray processes through the CoreWorker library.

Lifetime of a Ray task ...

```
# Driver code
...
A_ref = A.remote()
```

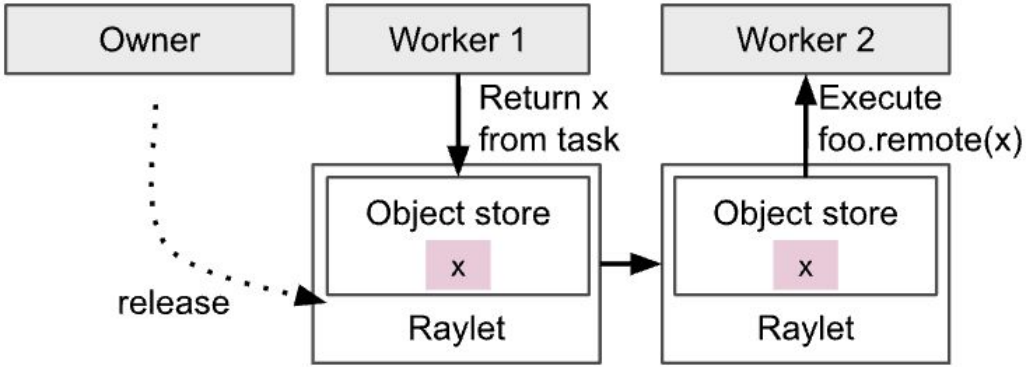



```
# Worker 1 code
...
B = B.remote()
return B
```

The process that submits a task is considered to be the owner of the result and is responsible for acquiring resources from a raylet to execute the task. Here, the driver owns the result of `A`, and `Worker 1` owns the result of `B`.

RAY

Lifetime of a Ray object ...

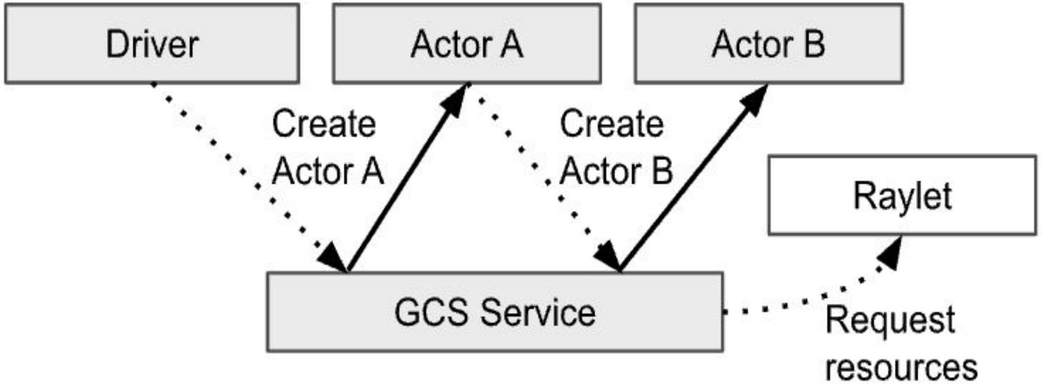


Distributed memory management in Ray. Workers can create and get objects. The owner is responsible for determining when the object is safe to release.

Lifetime of a Ray Actor …

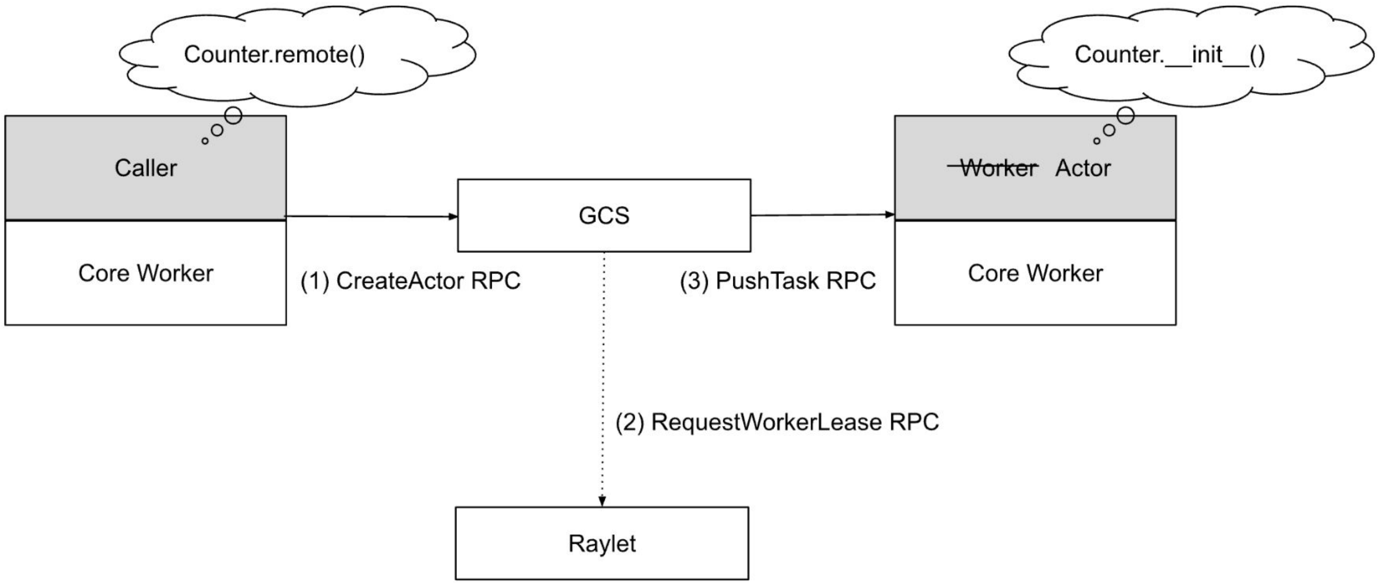
handle_A = ActorA.remote()

handle_B= ActorB.remote()



Unlike task submission, which is fully decentralized and managed by the owner of the task, actor lifetimes are managed centrally by the GCS service.

Actor creation sequence …



Actor creation tasks are scheduled through the centralized GCS service.

RAY

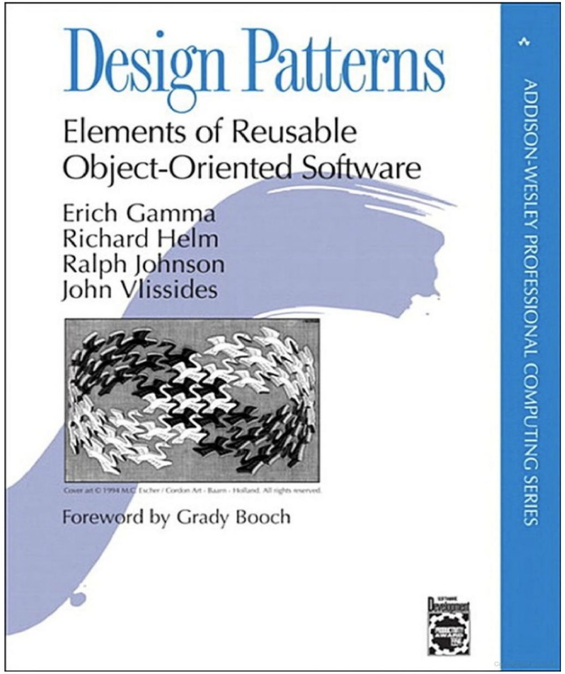
Ray core design & scaling patterns

RAY

Ray basic design pattern

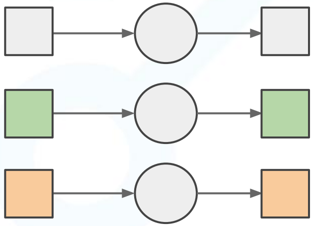
- Ray Parallel Tasks
  - Functions as stateless units of execution
  - Functions distributed across the cluster as tasks
- Ray Objects as Futures
  - Distributed (immutable objects) store in the cluster
  - Fetched when materialized
  - Enable massive asynchronous parallelism
- Ray Actors
  - Stateful service on a cluster
  - Enable Message passing

- [Patterns for Parallel Programming]
- [Ray Distributed Library Integration Patterns]



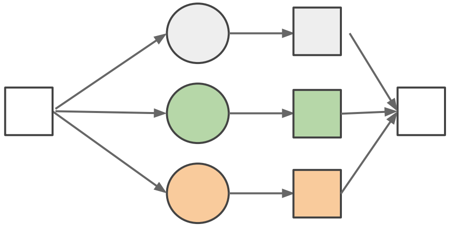
Scaling design patterns

Batch Training / Inference



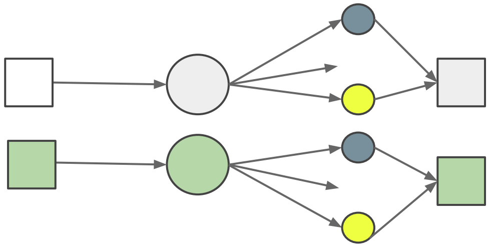
Different data / Same function

AutoML

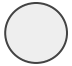


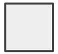
Same data / Different function

Batch Tuning



Different data / Same function

 Compute

 Data

RAY

Python → Ray API

```python
def f(x):
# do something with x:
   y = ...
   return y
v = f(x)
```

Task

```python
@ray.remote
def f(x):
# do something with x:
   y = ...
   return y
v_ref= f.remote(x)
```

Distributed

f()
Node

•••

f()
Node

```python
class Cls():
  def
__init__(self, x):
  def f(self, a):
       ...
  def g(self, a):
       ...
```

Actor

```python
@ray.remote
class Cls():
  def
__init__(self, x):
  def f(self, a):
       ...
  def g(self, a):
       ...
cls = Cls.remote()
cls.f.remote(a)
```

Distributed

Cls
Node

•••

Cls
Node

```python
import numpy as np
a= np.arange(1, 10e6)
b = a * 2
```

Distributed immutable object

```python
import numpy as np
a = np.arange(1, 10e6)
obj_a = ray.put(a)
b = ray.get(obj_a) * 2
```

Distributed

a
Node

•••

a
Node

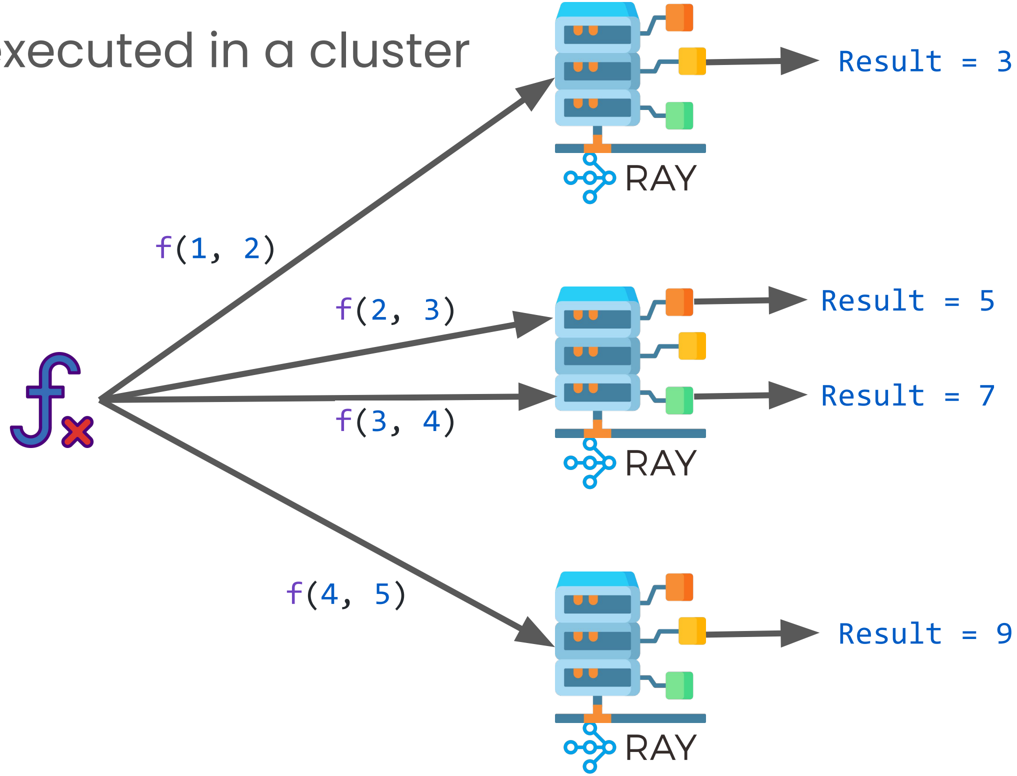
RAY

27

Ray Task

A function remotely executed in a cluster

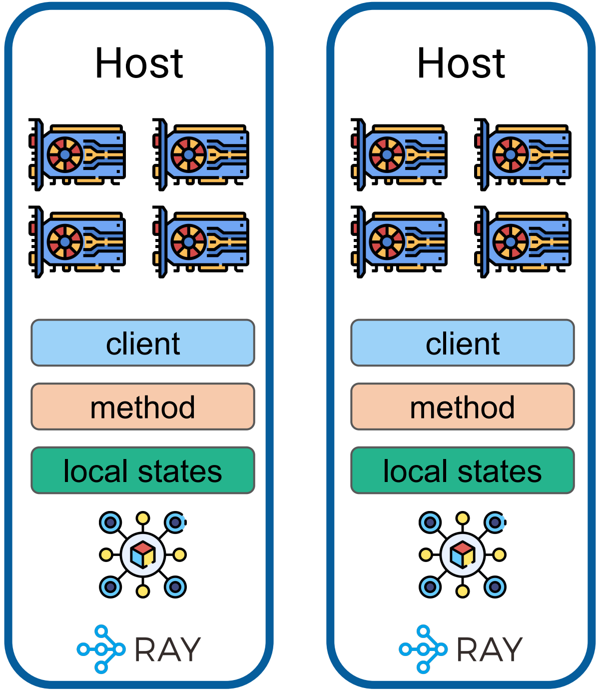
```python
@ray.remote(num_cpus=3)
def f(a, b):
    return a + b

f.remote(1, 2) # returns 3
f.remote(2, 3) # returns 5
f.remote(3, 4) # returns 7
f.remote(4, 5) # returns 9
```



Ray Actor

A class remotely executed in a cluster

```python
@ray.remote(num_gpus=4)
class HostActor:
    def __init__(self):
        self.model  = load_model(“s3://model_checkpoint”)
        self.num_devices = os.environ["CUDA_VISIBLE_DEVICES"]
    def inference(self, data):
         return self.model(data)
    def f(self, output):
        return f"{output} {self.num_devices}"

actor = HostActor.remote()  # Create an actor
actor.f.remote("hi")  # returns "hi 0,1,2,3"
actor.inference(input) # returns predictions…
```



RAY

Function → Task

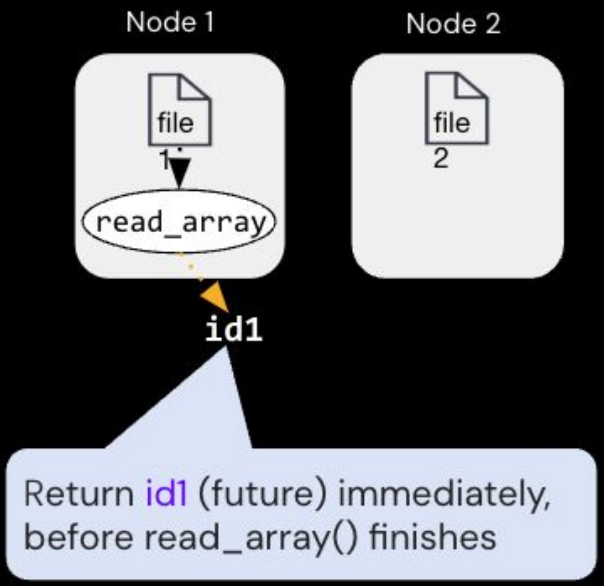
```
@ray.remote
def read_array(file):
    # read ndarray "a"
    # from "file"
    return a

@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

```
@ray.remote(num_gpus=1)
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()
```

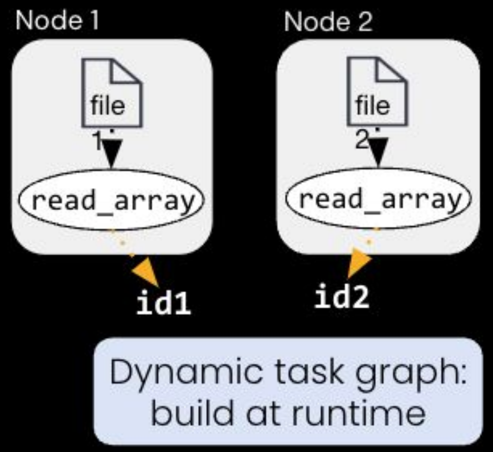
Task API

```
@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 API

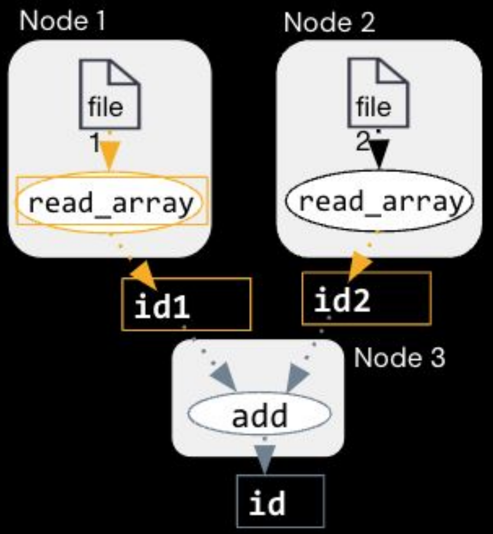
```
@ray.remote
def read_array(file):
    # read ndarray “a”
    # from “file”
    return a

@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 API

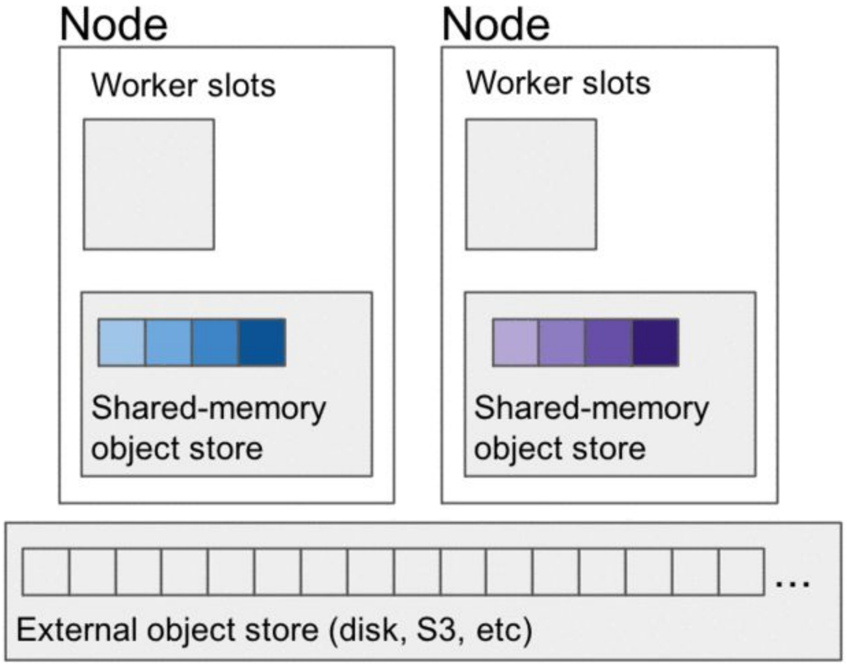
```
@ray.remote
def read_array(file):
    # read ndarray "a"
    # from "file"
    return a

@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



RAY

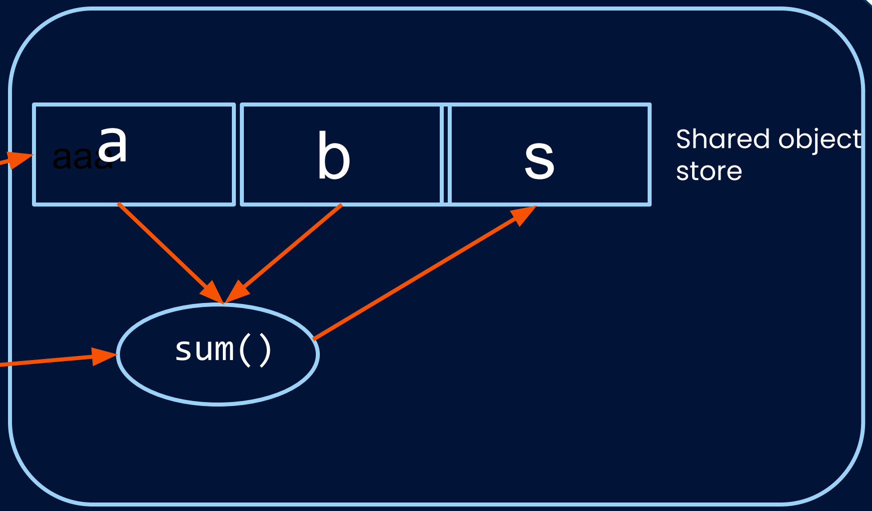
Distributed objects



RAY

Distributed object store

```
a = read_array(file1)
b = read_array(file2)

a_x = ray.put(a)
b_x = ray.put(b)

s_x = sum.remote(a_x, b_x)
val = ray.get(s_x)
print(val)
```



Who's using Ray ….

Sample of Companies Who Use Ray in their Machine Learning Platform

27,000+ GitHub stars	870+ Community Contributors	5,000+ Repositories Depend on Ray	1,000+ Organizations Using Ray

Let’s go with Ray

bitly

RAY

🍎 Recap: Today we learned…

🚂 Why Ray & What's Ray Ecosystem

- Architecture components
  - Role in distributed systems

🔬 Ray Design & Scaling Patterns & APIs

- Ray Tasks, Actors, Objects

RAY

🍏 Sneak Peek: Self-Paced Ray & Anyscale Training

📧 Online at training.anyscale.com

🔍 Preview special technical content releases from the whole team!

Reading list.

[Ray Education GitHub]

Access bonus notebooks and scripts about Ray.

[Ray documentation]

API references and user guides.

[Anyscale Blogs]

Real world use cases and announcements.

[YouTube Tutorials]

Video walkthroughs about learning LLMs with Ray.

RAY

Resources

- How to fine tune and serve LLMs simply, quickly and cost effectively using Ray + DeepSpeed + HuggingFace
- Get started with DeepSpeed and Ray
- Training 175B Parameter Language Models at 1000 GPU scale with Alpa and Ray
- Fast, flexible, and scalable data loading for ML training with Ray Data
- Ray Serve: Tackling the cost and complexity of serving AI in production
- Scaling Model Batch Inference in Ray: Using Actors, ActorPool, and Ray Data
- Fine-Tuning Llama-2: A Comprehensive Case Study for Tailoring Models to Unique Applications (part-1)
- Fine-Tuning LLMs: LoRA or Full-Parameter? An in-depth Analysis with Llama 2 (part-2)

Thank you!

Any questions?

RAY