

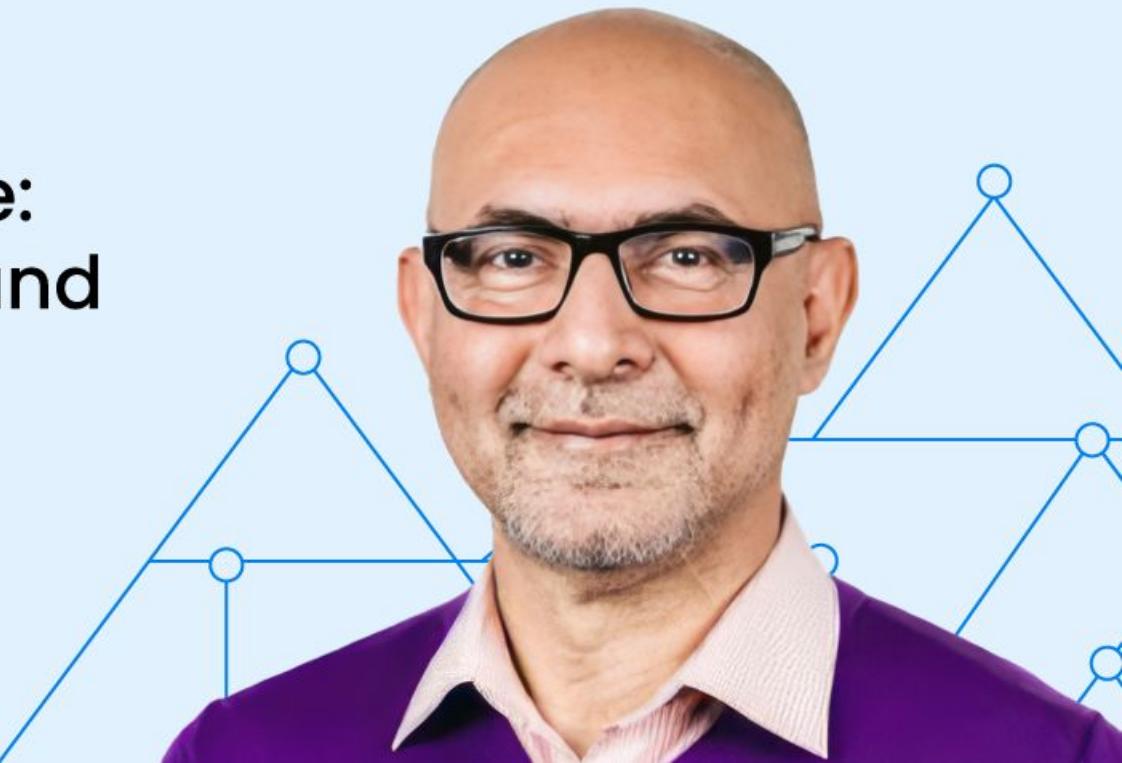


TRAINING

Ray Core Deep Dive: API, Architectures and Best Practices

Jules Damji
Anyscale

presented by  anyscale





Meet the TAs!



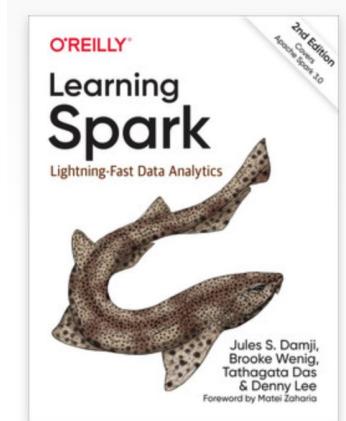
Ruiyang
Wang



Ricky
Xu

\$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



A few important URLs:

Keep these URLs open in a browser tab:

- GitHub Learning Material: https://bit.ly/ray_summit_2023
- Ray Documentation: <https://bit.ly/ray-core-docs>
- Ray Core Class Survey: bit.ly/ray-summit-feedback





Today's agenda.

- Tech check (slido app, Anyscale clusters, etc)
- Why & What's Ray & Ray Ecosystem
- Ray Architecture & Components
- Ray Core Design & Scaling Patterns & APIs
- Modules [1-3]: Hands-on in class 
- Break 2:15 pm - 2:45 PM   
- Modules [4-5]: Hands-on in class 
- Wrap up...
- Happy Hour Meetup 



Tech check.

Participating via app.sli.do

- Join with code **#ray-core**
- Ask questions.
 - Pose your own and upvote others.
 - TAs will be answering questions on a rolling basis.



Tech check.



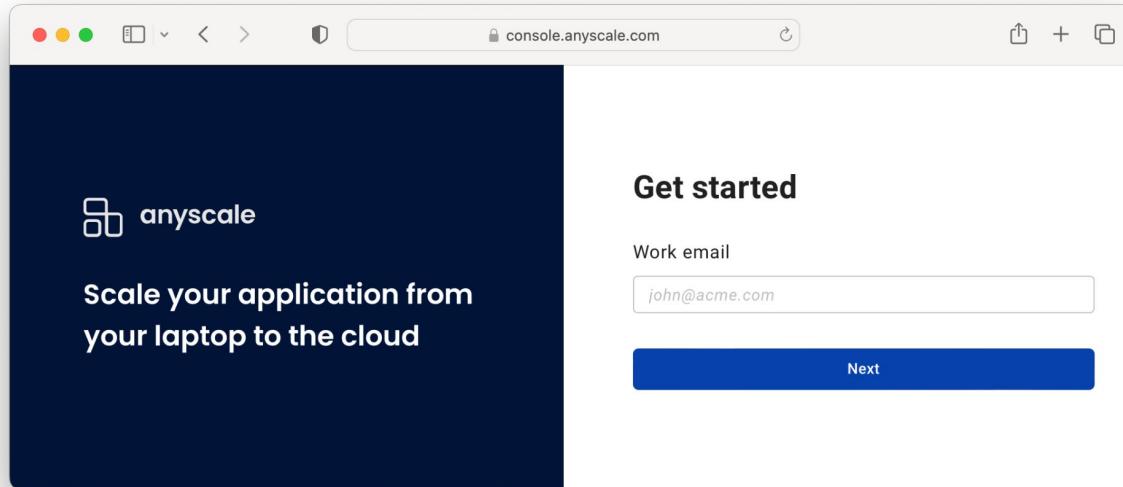
Accessing Anyscale clusters.

- All work will be in Anyscale provisioned clusters.
- Our GitHub repo will be mounted automatically.
- Access begins now.
 - Check your email for login information.
 - Step-by-step instructions to follow.

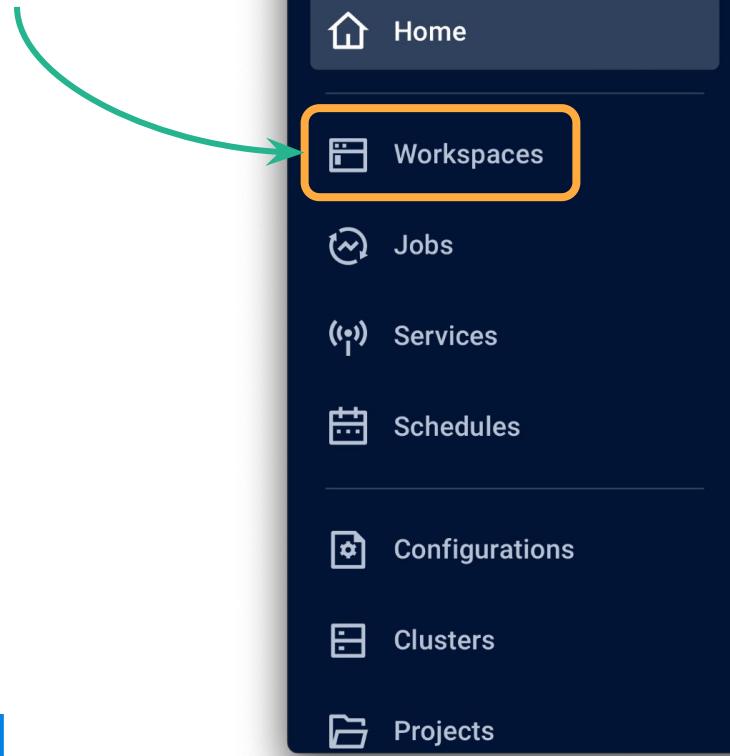


Anyscale login

Link to Anyscale cluster: console.anyscale.com



1. Select Workspaces



The main content area shows the following sections:

Home

Examples to get started

Introduction to Anyscale & Ray Launch ▼

Learn about Anyscale and Ray in this introductory tutorial. This template runs a simple Ray program on a distributed Ray cluster then deploys an Anyscale Job based on the Ray program.

Ray task Anyscale Job

Many Model Training Launch ?

2. Select Your Workspace

The image shows two screenshots of the Anyscale console interface. The left screenshot is a dark-themed sidebar menu titled 'anyscale' with the following items:

- Home
- Workspaces** (highlighted with a teal arrow)
- Jobs
- Services
- Schedules
- Configurations
- Clusters

The right screenshot shows the 'Workspaces' page with the title 'my-workspace'. It includes buttons for '+ Create', 'Start', 'Terminate', and 'Delete'. A search bar and a filter button 'Created by is me' are also present. A table lists one workspace entry:

Name	Status
my-workspace	Active

A teal arrow points from the 'Workspaces' item in the sidebar to the 'my-workspace' entry in the list. A blue question mark icon is in the bottom right corner.

4. Find the content for your class here.

The screenshot shows a JupyterLab interface with a file browser on the left and a launcher on the right. The file browser has a search bar and a list of files. A folder named "ray-summit-2023-training" is highlighted with an orange border and a green arrow points to it from the text above. The launcher on the right lists "Notebook", "Python 3 (ipykernel)", "Console", and another "Python 3" entry. The bottom right corner shows a notification bell with the number 1.

File Edit View Run Kernel Tabs Settings Help

my-workspace

Launcher

Notebook

Python 3 (ipykernel)

Console

Python 3

Launcher 1

Why Ray + What's Ray



Why Ray



Machine learning is pervasive

Distributed computing is a necessity

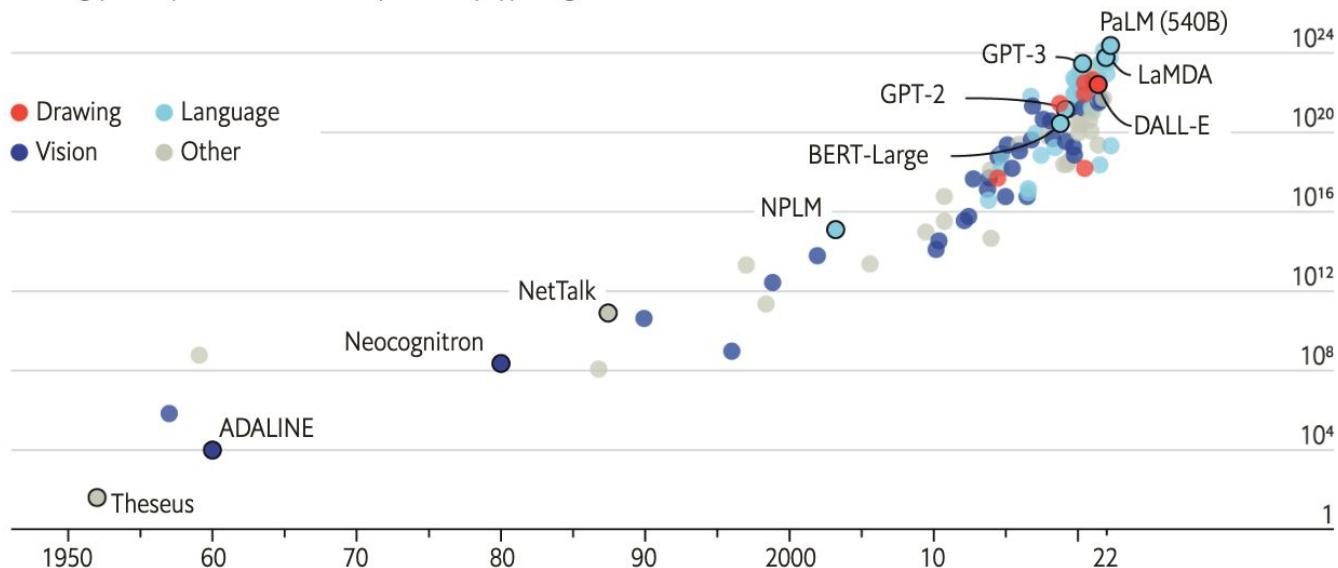
Python is the default language for DS/ML

Blessings of scale ...

The blessings of scale

AI training runs, estimated computing resources used

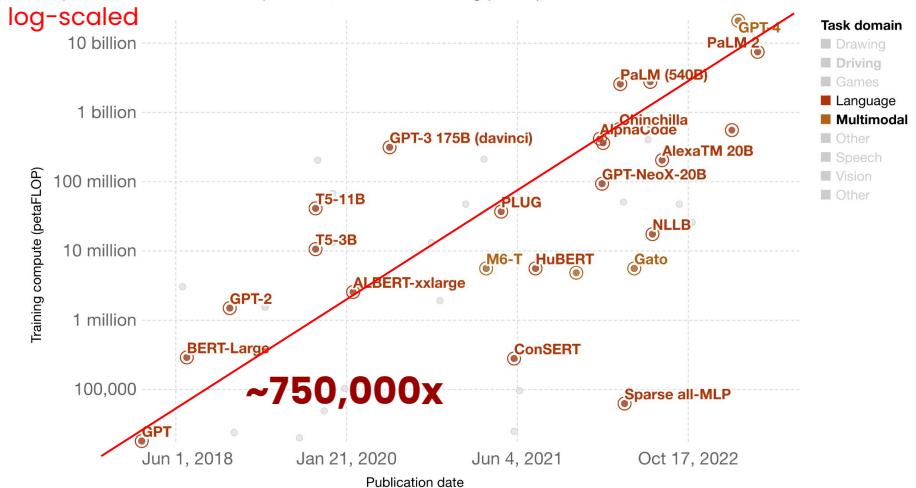
Floating-point operations, selected systems, by type, log scale



Sources: "Compute trends across three eras of machine learning", by J. Sevilla et al., arXiv, 2022; Our World in Data

Scaling is a Necessity

Computation used to train notable artificial intelligence systems
Computation is measured in total petaFLOP, which is 10^{15} floating-point operations¹.



Source: Epoch (2023)

Note: Computation is estimated based on published results in the AI literature and comes with some uncertainty. The authors expect most of these estimates to be correct within a factor of 2, and a factor of 5 for recent models for which relevant numbers were not disclosed, such as GPT-4.

1. Floating-point operation: A floating-point operation (FLOP) is a type of computer operation. One FLOP is equivalent to one addition, subtraction, multiplication, or division of two decimal numbers.

1. Model size are getting larger

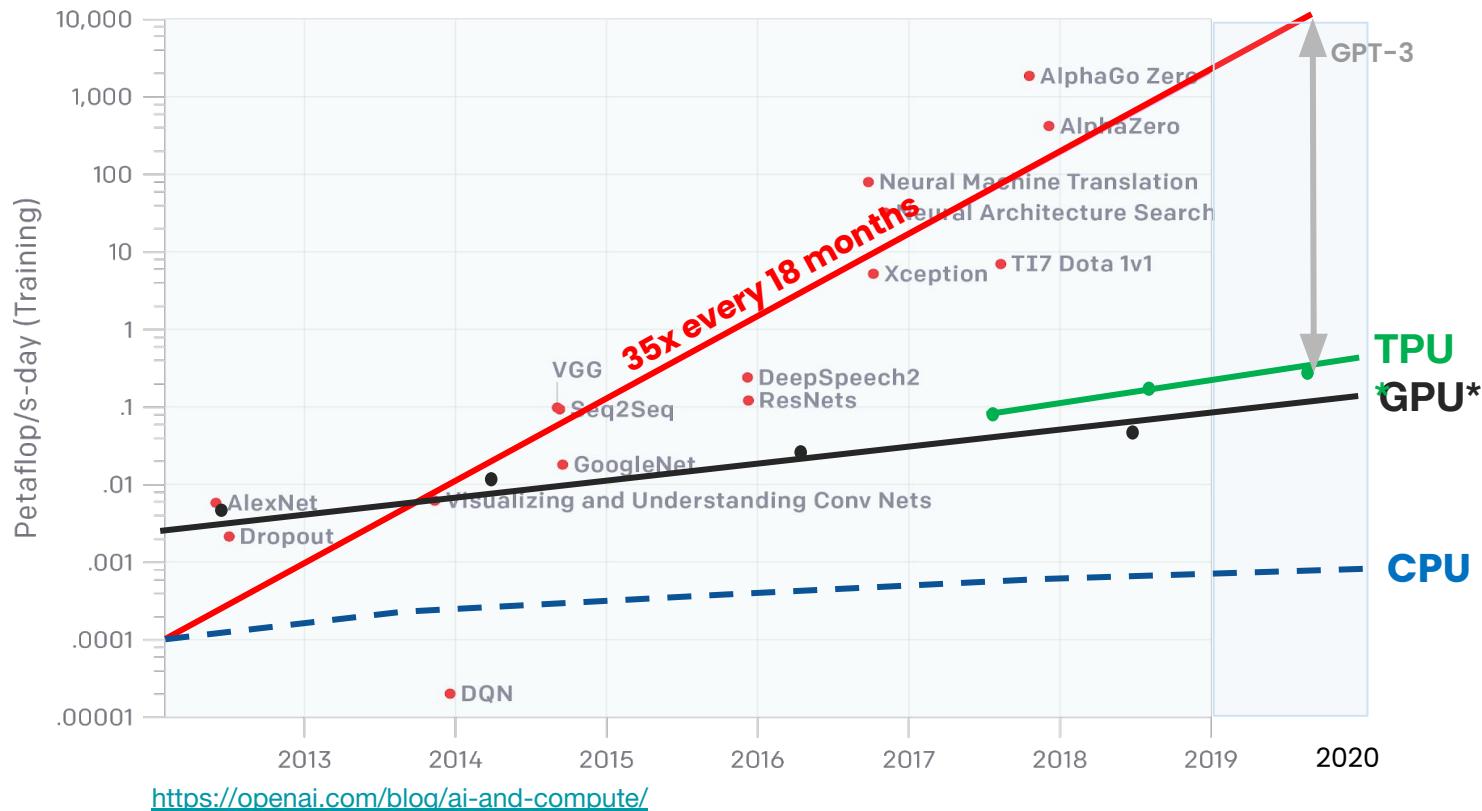
- Model size is **exponentially** increasing.
- Models are too large to fit into a single GPU.
- We need to shard the models across multiple GPUs for training
 - e.g. ZeRO, Model Parallel, Pipeline Parallel

BERT (2019): 336M params(1.34GB)

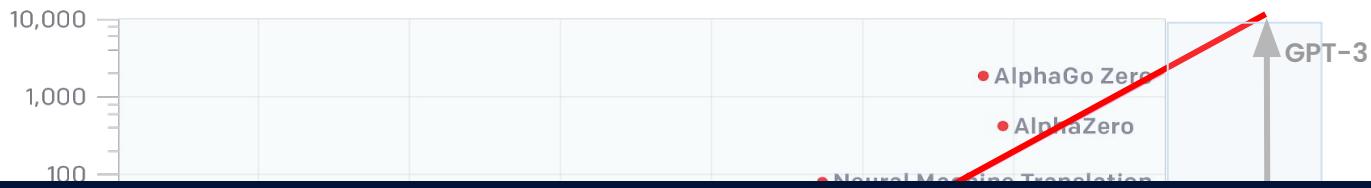
Llama-2 (2023): 70B params(280GB) ~20x

GPT-4: ~1800B >5,000x

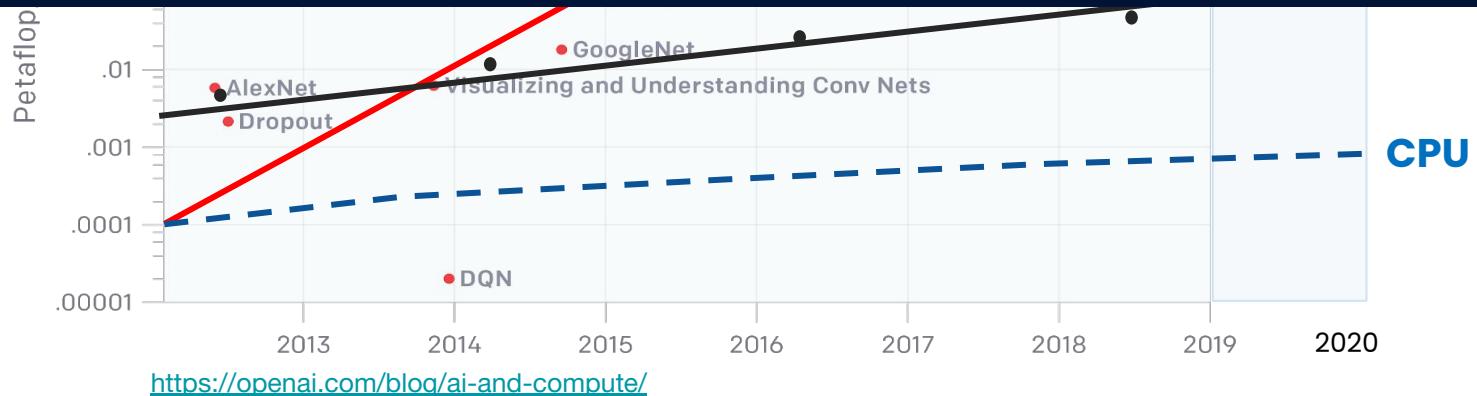
Supply-demand problem



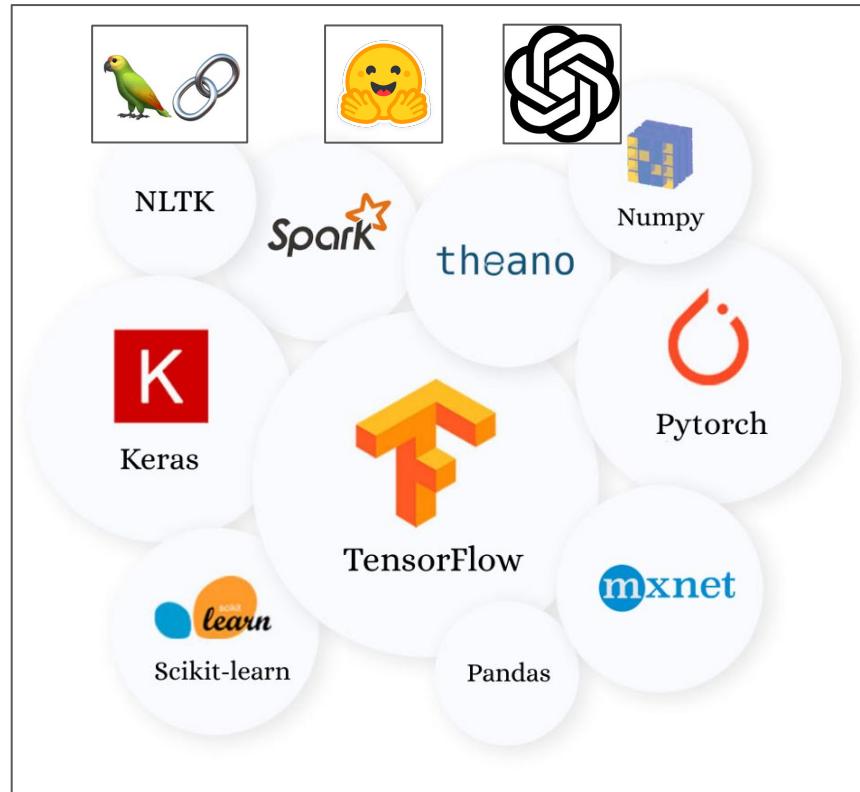
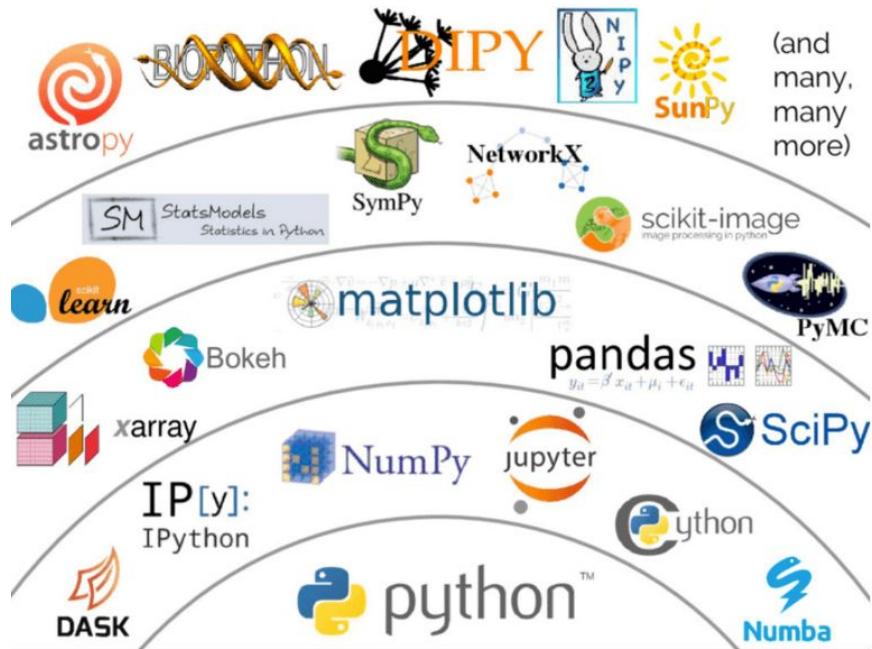
Specialized hardware no enough!



No way out but to distribute!



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 AI Libraries enable simple scaling of AI workloads.



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

Custom Applications



Tasks

Actors

Objects

A Layered Cake and Ecosystem

Ray AI Libraries enable simple scaling of AI workloads.



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

Custom Applications

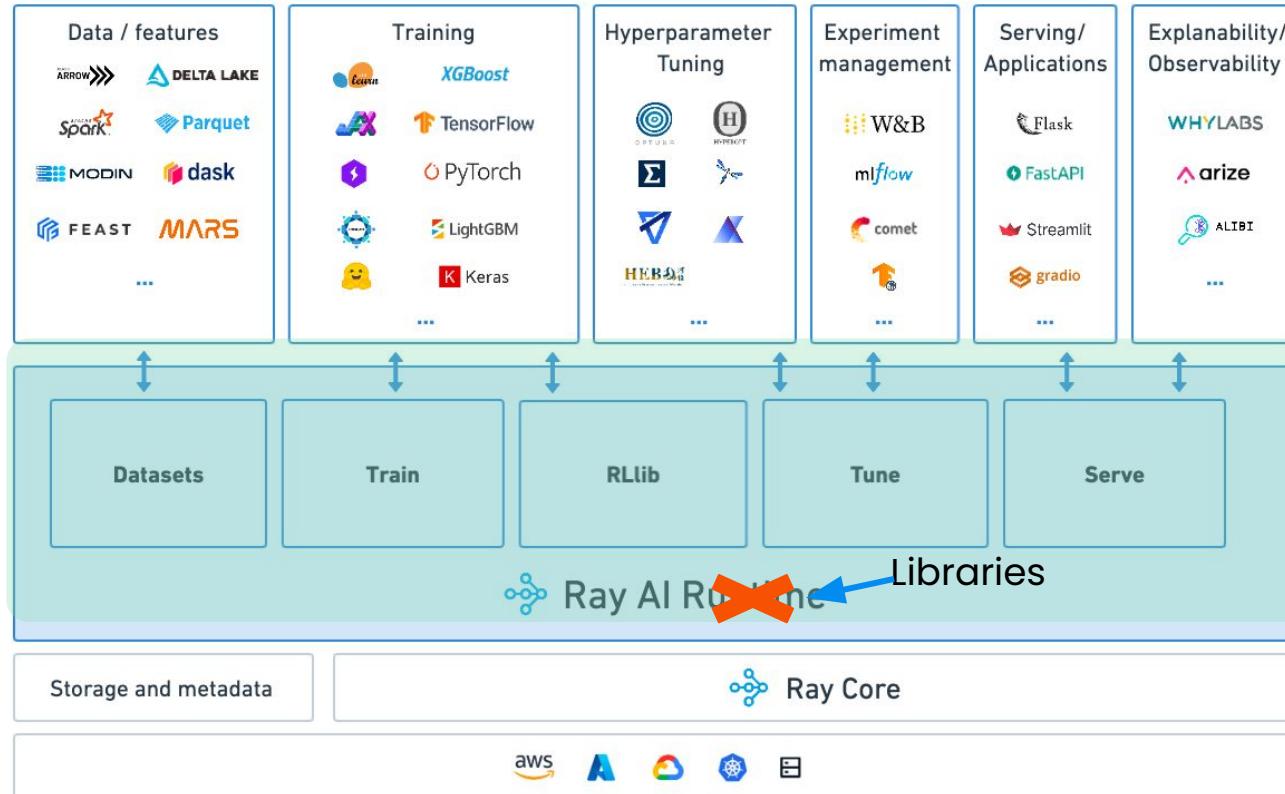


Tasks

Actors

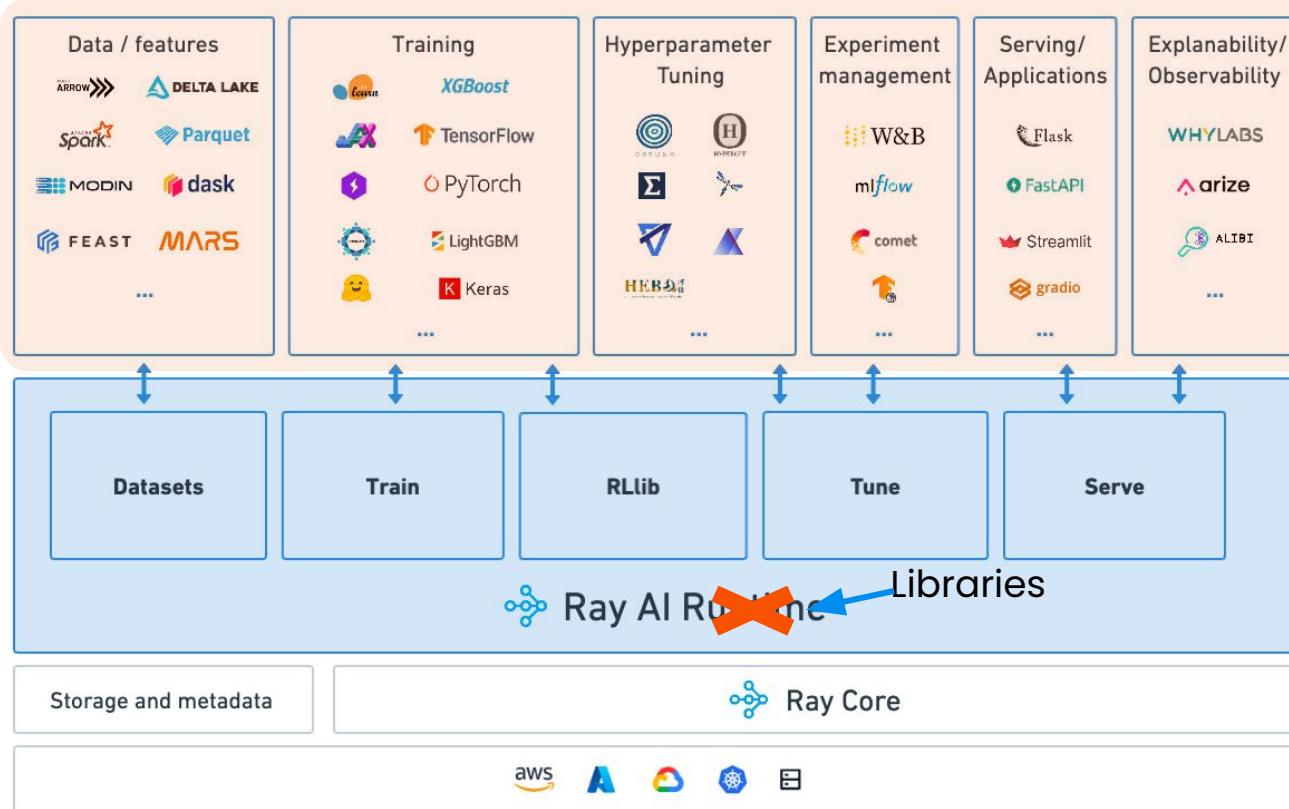
Objects

Ray AI Libraries

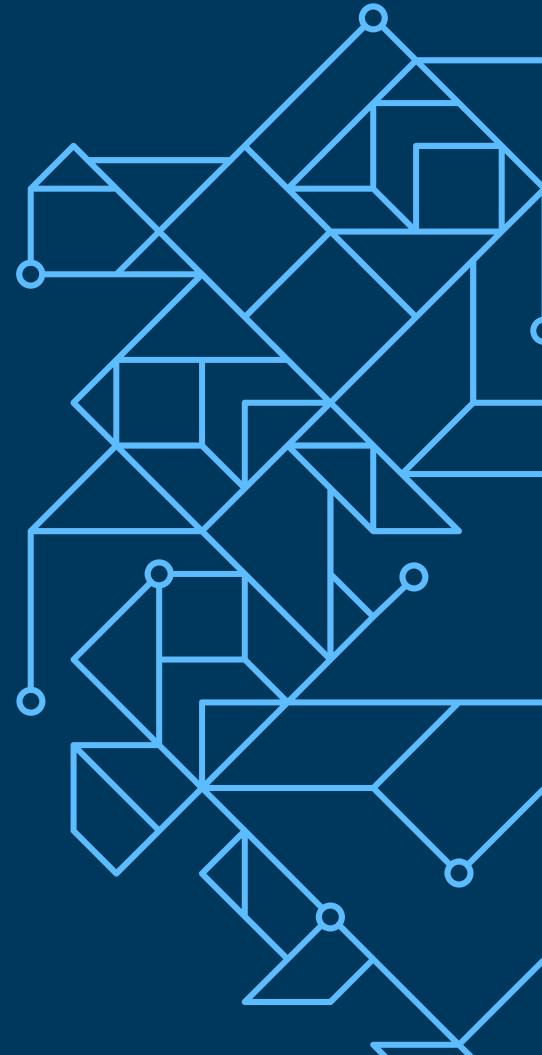


High-level Ray AI Libraries
that make scaling easy for
both data scientists and
ML engineers.

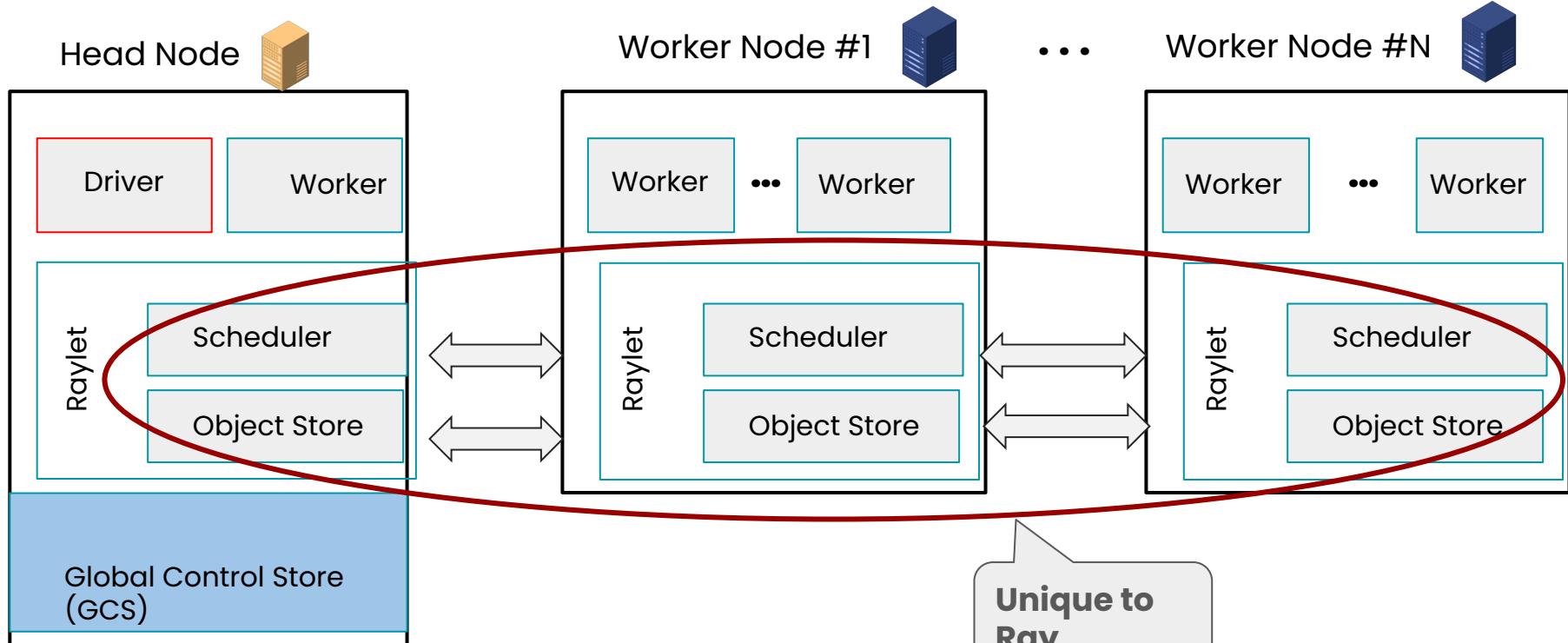
Ray AI Libraries



Ray Architecture & Components

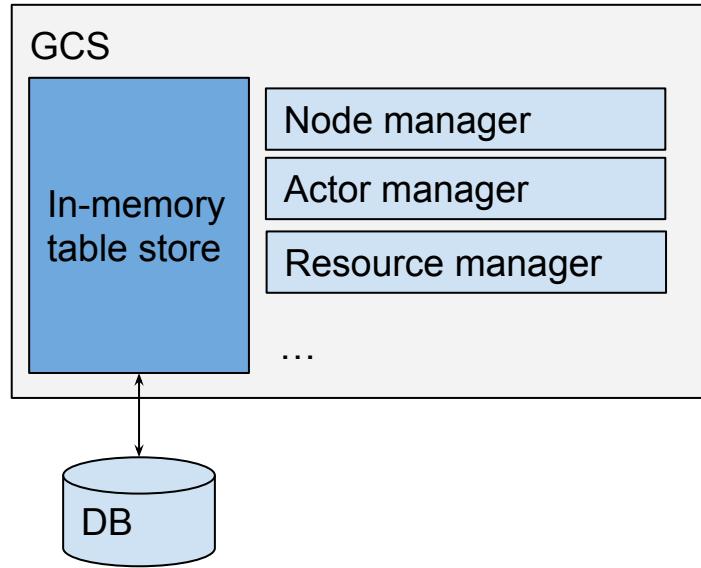


Anatomy of a Ray Cluster



Ray components – GCS

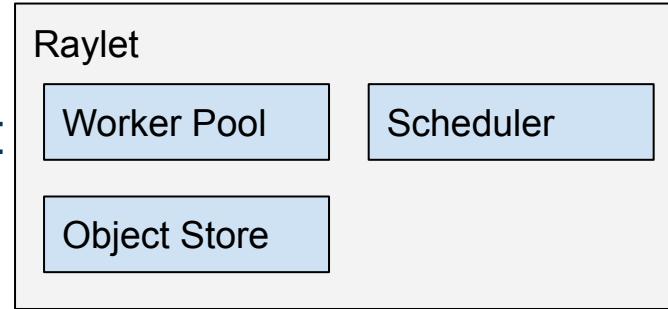
- Node management
 - Monitor liveness
 - Maintain membership
- Actor management
 - Actor creation
 - Actor destruction
 - Actor recovery
- Resource management
 - Keep raylets' cluster resource view updated.



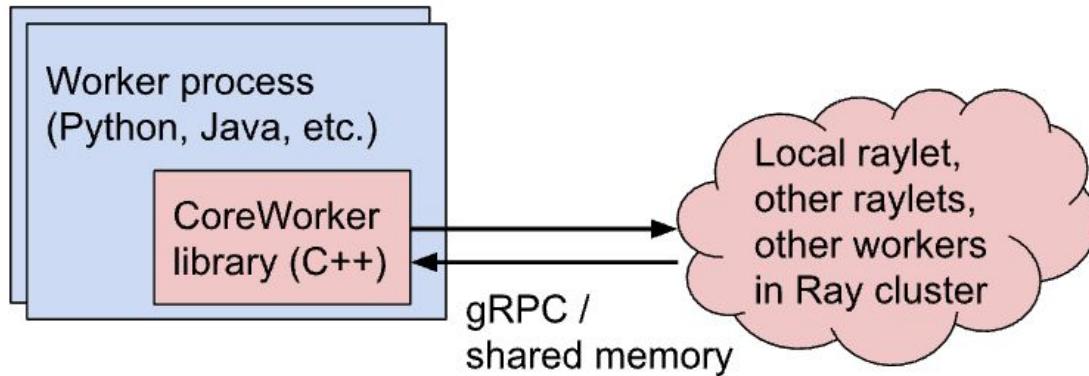
Ray Cluster, from Ray 2.0 White paper

Ray components – Raylet

- Manage workers (worker pool)
 - Prepare worker environment
 - Monitor worker's liveness
 - Failure recovery
- Object store
 - Store ray object data
 - Data transfer between nodes
- Task Scheduler
 - Distributed scheduler for tasks

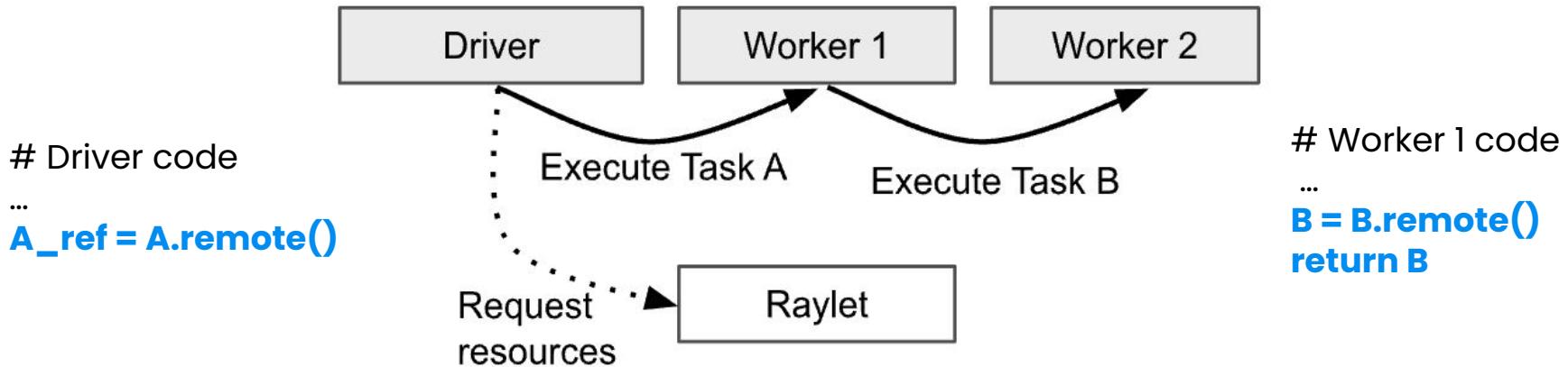


Anatomy of a Ray worker process



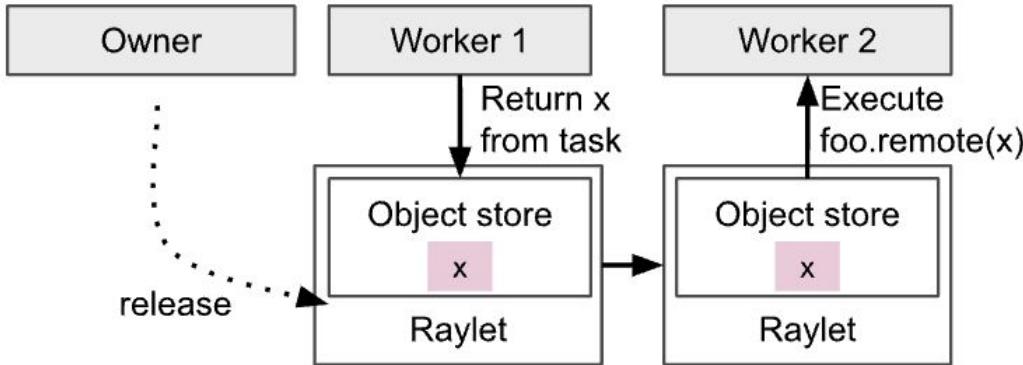
Ray workers interact with other Ray processes through the CoreWorker library.

Lifetime of a Ray Task



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'.

Lifetime of a Ray Object

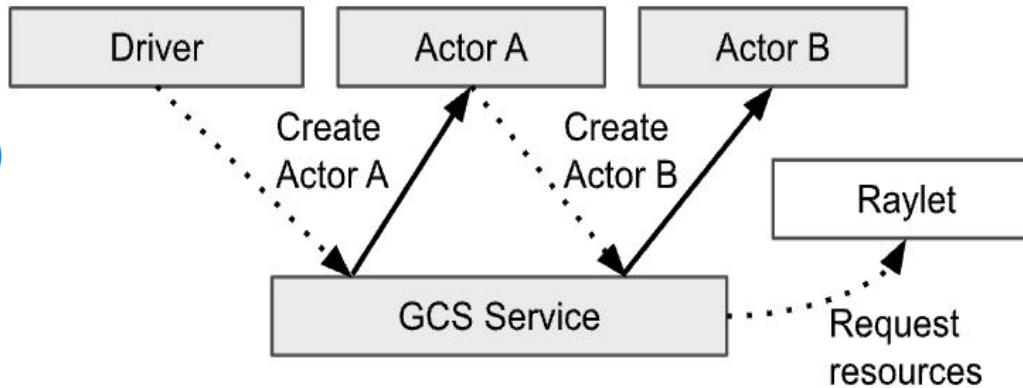


Distributed memory management in Ray. Workers can create and get objects. The owner is responsible for determining when the object is safe to release.

Lifetime of a Ray Actor ...

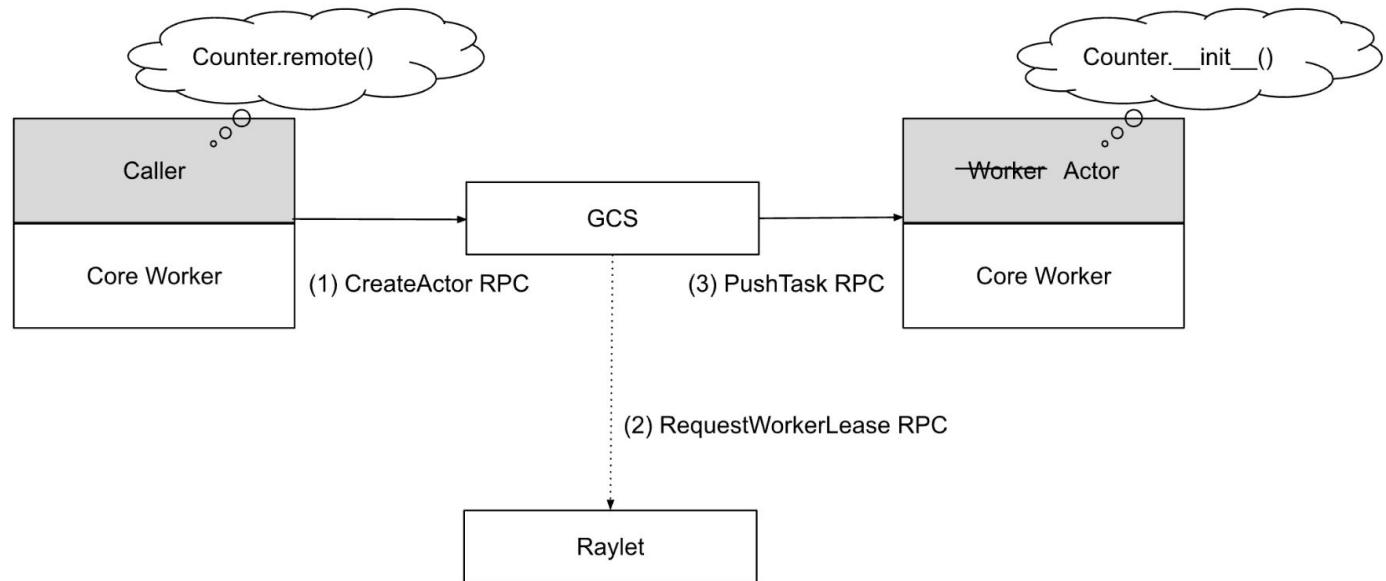
`handle_A =
ActorA.remote()`

`handle_B=
ActorB.remote()`



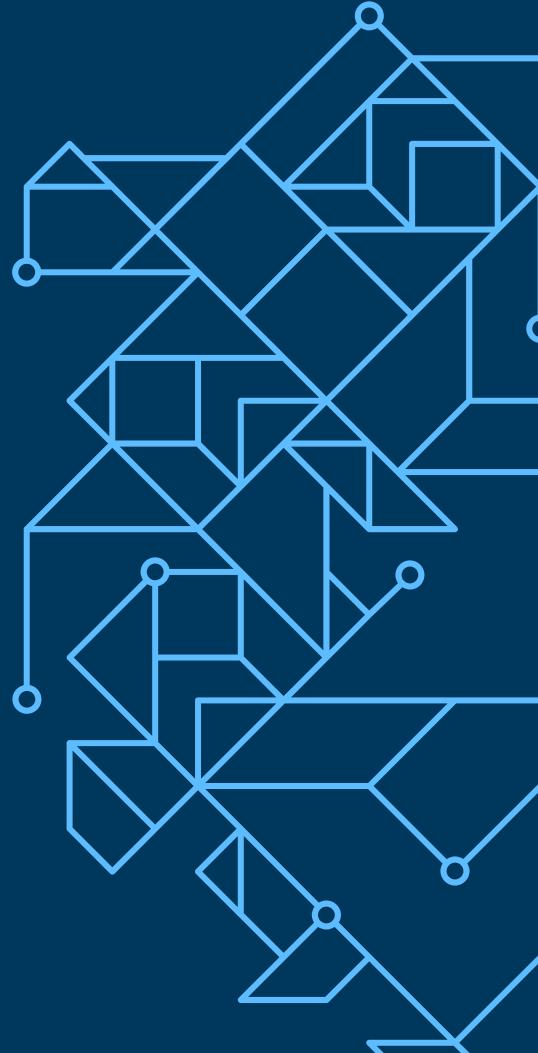
Unlike task submission, which is fully decentralized and managed by the owner of the task, actor lifetimes are managed centrally by the GCS service.

Actor creation sequence



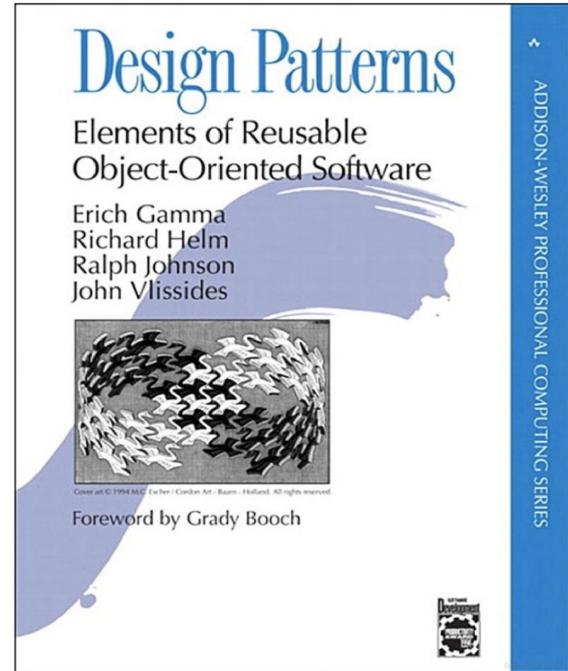
Actor creation tasks are scheduled through the centralized GCS service.

Ray Core Design & Scaling Patterns & APIs



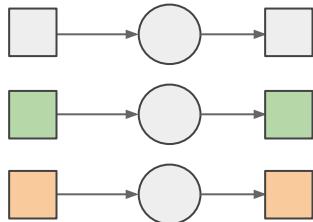
Ray Basic Design Patterns

- **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

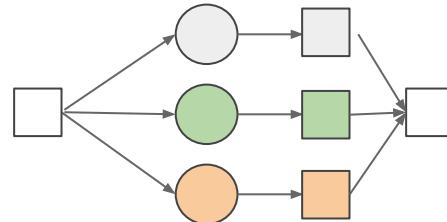


Different data / Same function

Compute

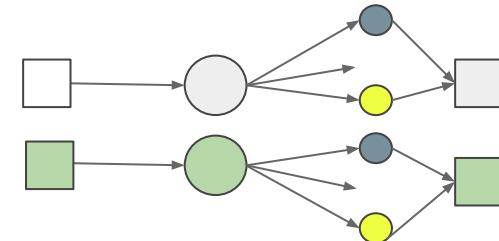
Data

AutoML



Same data / Different function

Batch Tuning



Different data / Same function

Python → Ray API

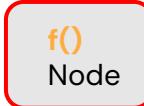


```
def f(x):
# do something with x:
    y = ...
    return y
v = f(x)
```

Task

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

Distributed



...

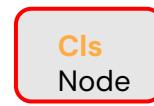


```
class Cls():
    def
    __init__(self, x):
        def f(self, a):
            ...
            def g(self, a):
                ...
                ...
```

Actor

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

Distributed



...



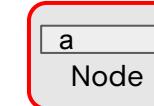
```
import numpy as np
a= np.arange(1, 10e6)
b = a * 2
```

Distributed immutable object



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

Distributed



...

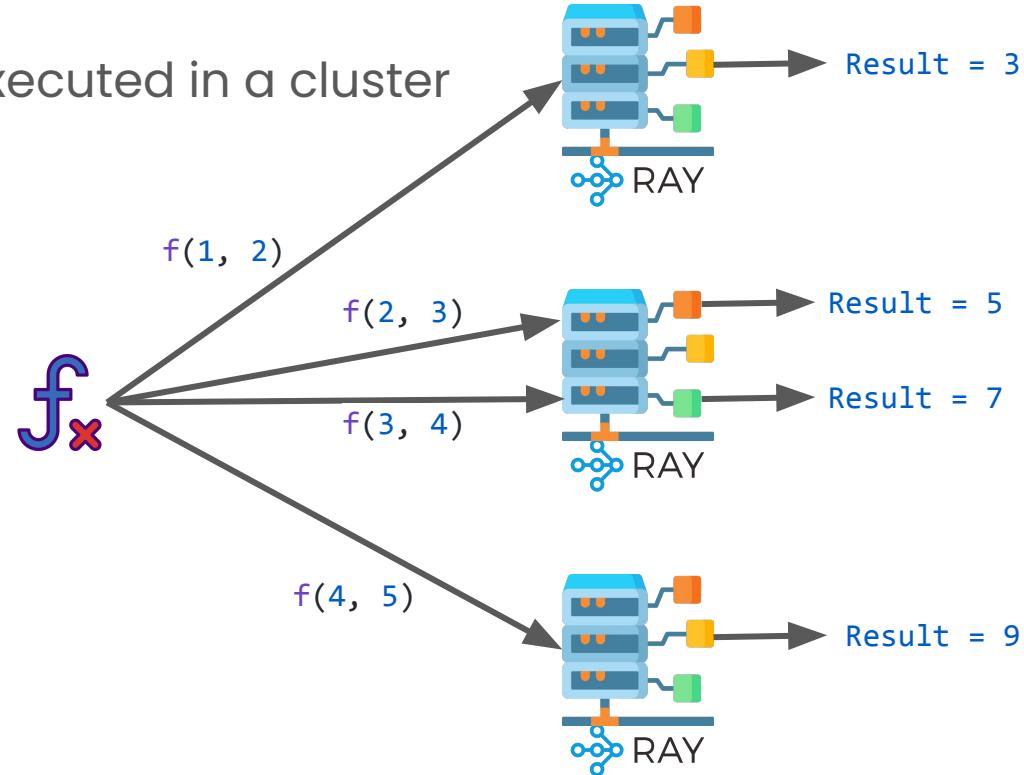


Ray Task

A function f_x remotely executed in a cluster

```
@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

```
@ray.remote(num_gpus=4)

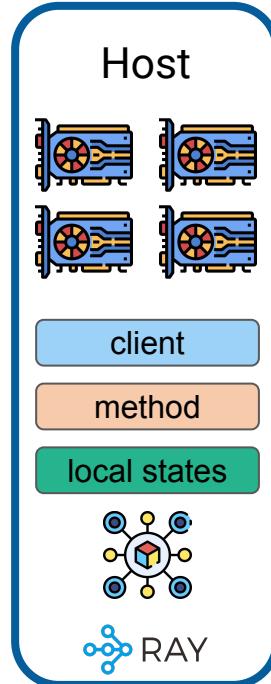
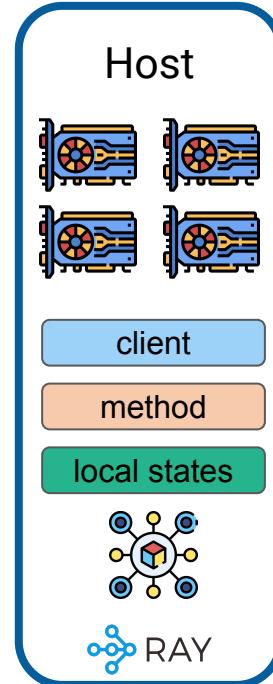
class HostActor:

    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.remote(input) # returns predictions...
```



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)
```

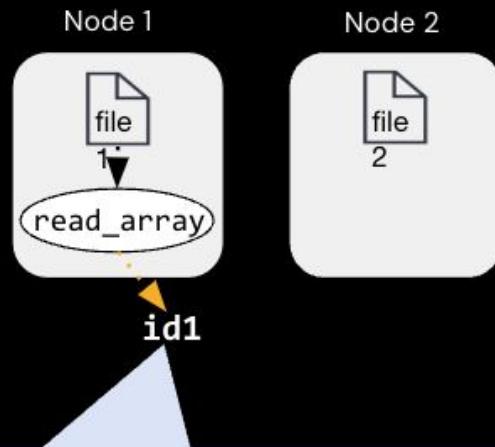
Class → Actor

```
@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

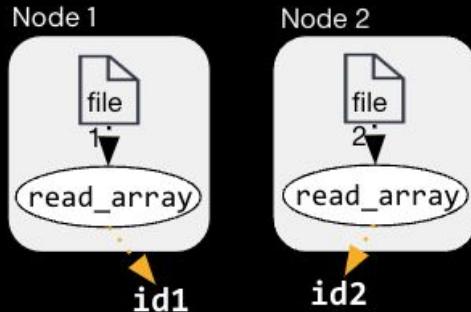
```
@ray.remote  
def read_array(file):  
    # read ndarray "a"  
    # from "file"  
    return a  
  
@ray.remote  
def add(a, b):  
    return np.add(a, b)  
  
id1 = read_array.remote(file1)  
id2 = read_array.remote(file2)  
id = add.remote(id1, id2)  
sum = ray.get(id)
```



Return `id1` (future) immediately,
before `read_array()` finishes

Task API

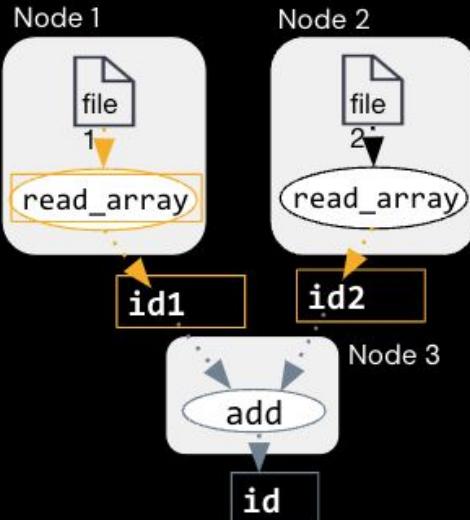
```
@ray.remote  
def read_array(file):  
    # read ndarray "a"  
    # from "file"  
    return a  
  
@ray.remote  
def add(a, b):  
    return np.add(a, b)  
  
id1 = read_array.remote(file1)  
id2 = read_array.remote(file2)  
id = add.remote(id1, id2)  
sum = ray.get(id)
```



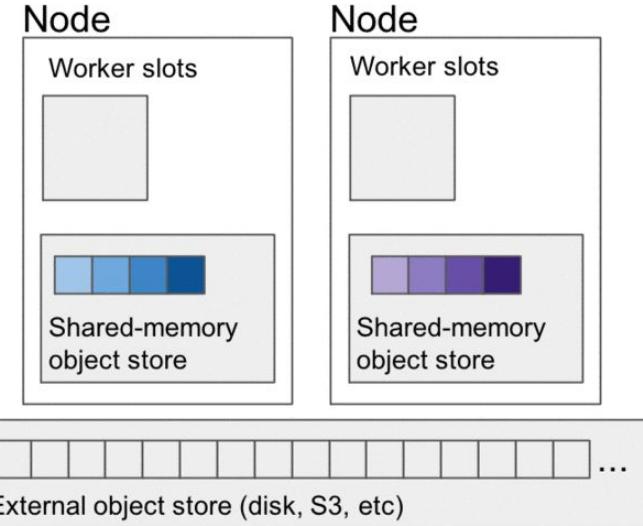
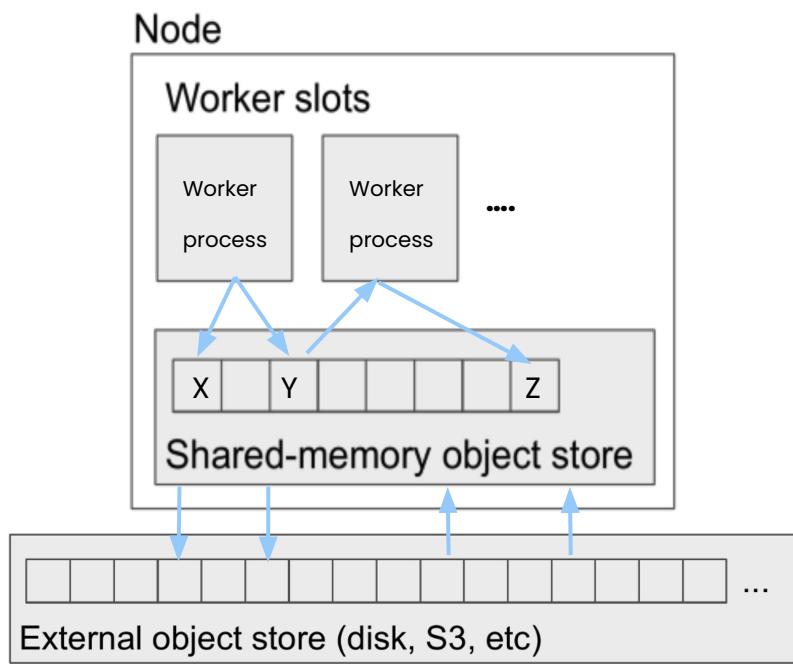
Dynamic task graph:
build at runtime

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) ← ray.get() block until  
result available
```



Distributed object store

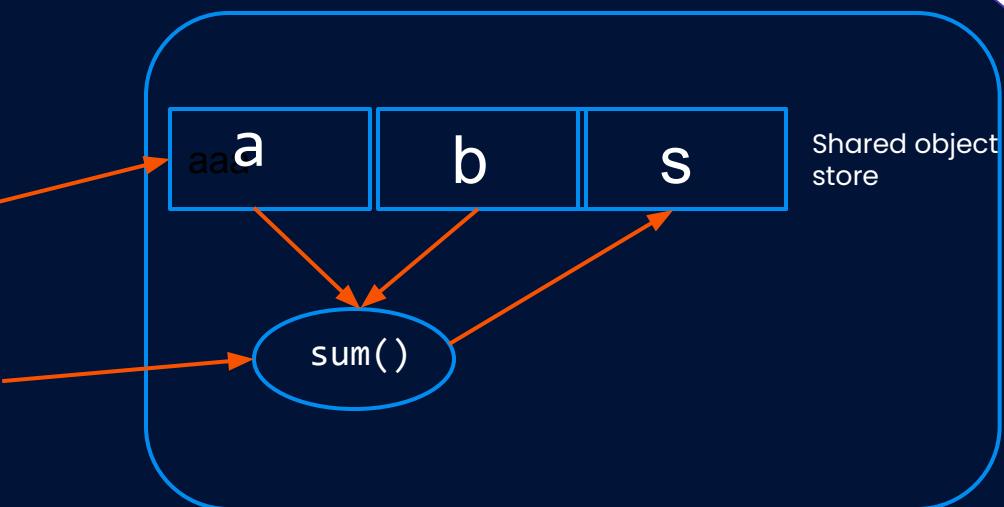


Distributed object store

```
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)
```



Let's go with Ray





Time for a Break!



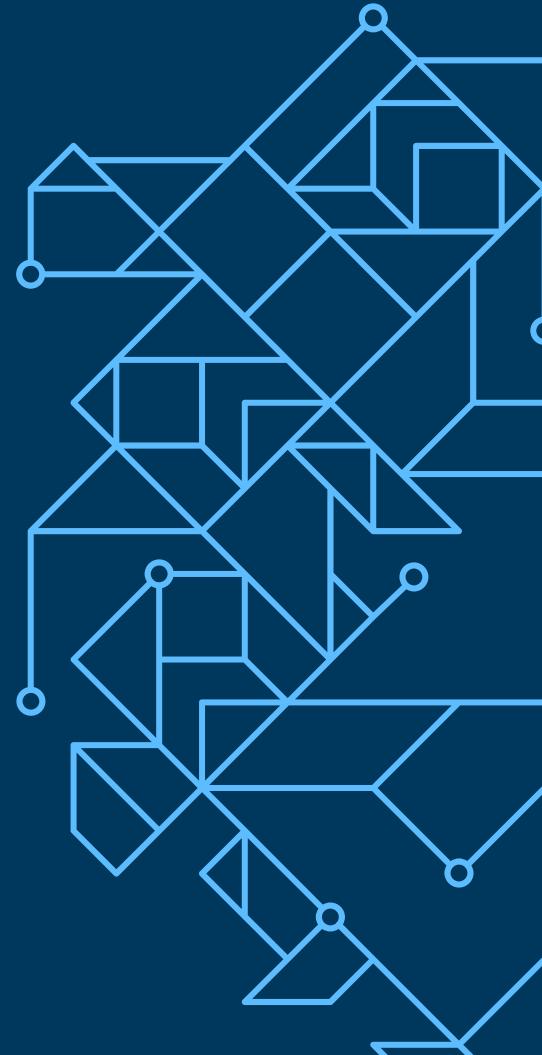


Time for a Break!



Recap & More Resources

For further exploration with Ray, Anyscale, and LLMs.





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



Patterns and Antipatterns

- How to avoid pitfalls
- Use best practices



Sneak Peek: Self-Paced Ray & Anyscale Training



Online at training.anyscale.com



Preview special technical content releases from the whole team!



Fill out the survey.



Go to bit.ly/ray-summit-feedback





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.

Bay Area AI + Ray Summit Meetup & Happy Hour

Bay Area AI + Ray Summit Meetup



Hosted By
Alexy K. and 4 others



Details

Our next in-person meetup is held on the last day of the Ray Summit 2023.

We focus on Ray use in AI, such as LLM, and deployment of ML workloads in various use cases. Weights & Biases will host and present, and planned talks include IBM Research and Open-Source Science. We'll also host lightning talks and networking.

 Bay Area AI
Public group ?

Wednesday, September 20, 2023 at 5:00 PM to Wednesday, September 20, 2023 at 7:00 PM PDT
[Add to calendar](#)

Microsoft
555 California St #200 · San Francisco, CA



Report this event

https://bit.ly/bayai_ray_meetup



Thank you!



Email: jules@anyscale.com
X: [@2twitme](https://twitter.com/@2twitme)

