



Introduction to Ray for distributed and machine learning applications in Python

Jules S. Damji - [@2twitme](#)
April 26, 2023, Seattle, WA

Few Important URLs

Keep these URLs open in your browser tabs

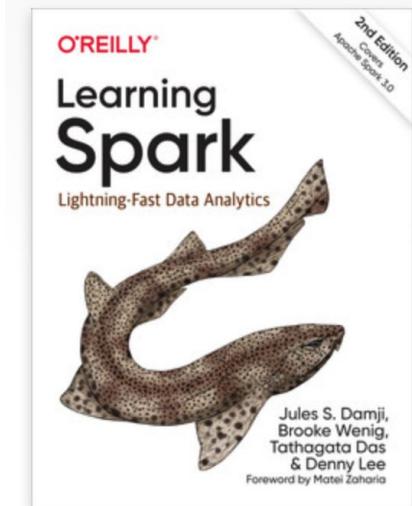
- Ray Core Class Survey: <https://bit.ly/pydata-seattle-2023>
- GitHub: <https://bit.ly/pydata-seattle-tutorial-2023>
- Ray Documentation: <https://bit.ly/ray-core-docs>





\$whoami

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





anyscale

Who we are: Original creators of Ray

What we do: Unified compute platform to develop, deploy, and manage scalable AI & Python applications with Ray

Why do it: Scaling is a necessity, scaling is hard; make distributed computing easy and simple for everyone

Agenda

- Why & What's Ray & Ray Ecosystem
- Ray Architecture & Components
- Ray Core Design & Scaling Patterns & APIs
- Modules [1] Hand-on in class
- Modules [2-3] Extra Curriculum at home

Why Ray



Machine
learning is
pervasive

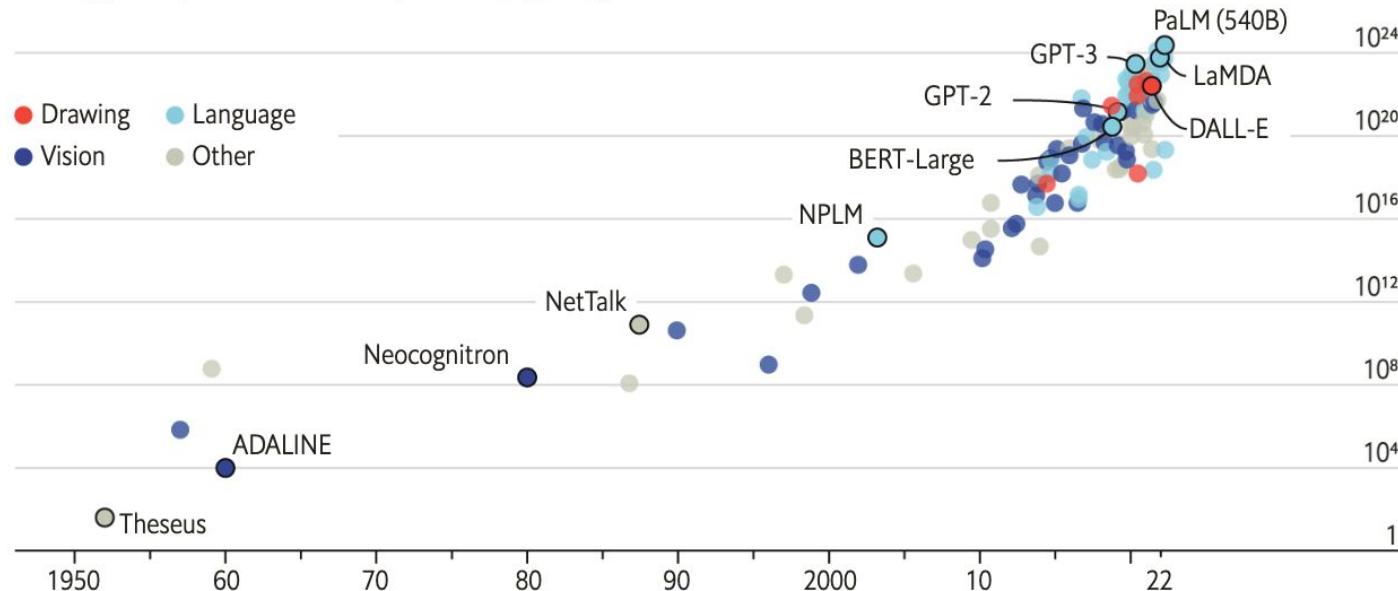
Distributed
computing is a
necessity

Python is the
default
language for
DS/ML

Blessings of scale ...

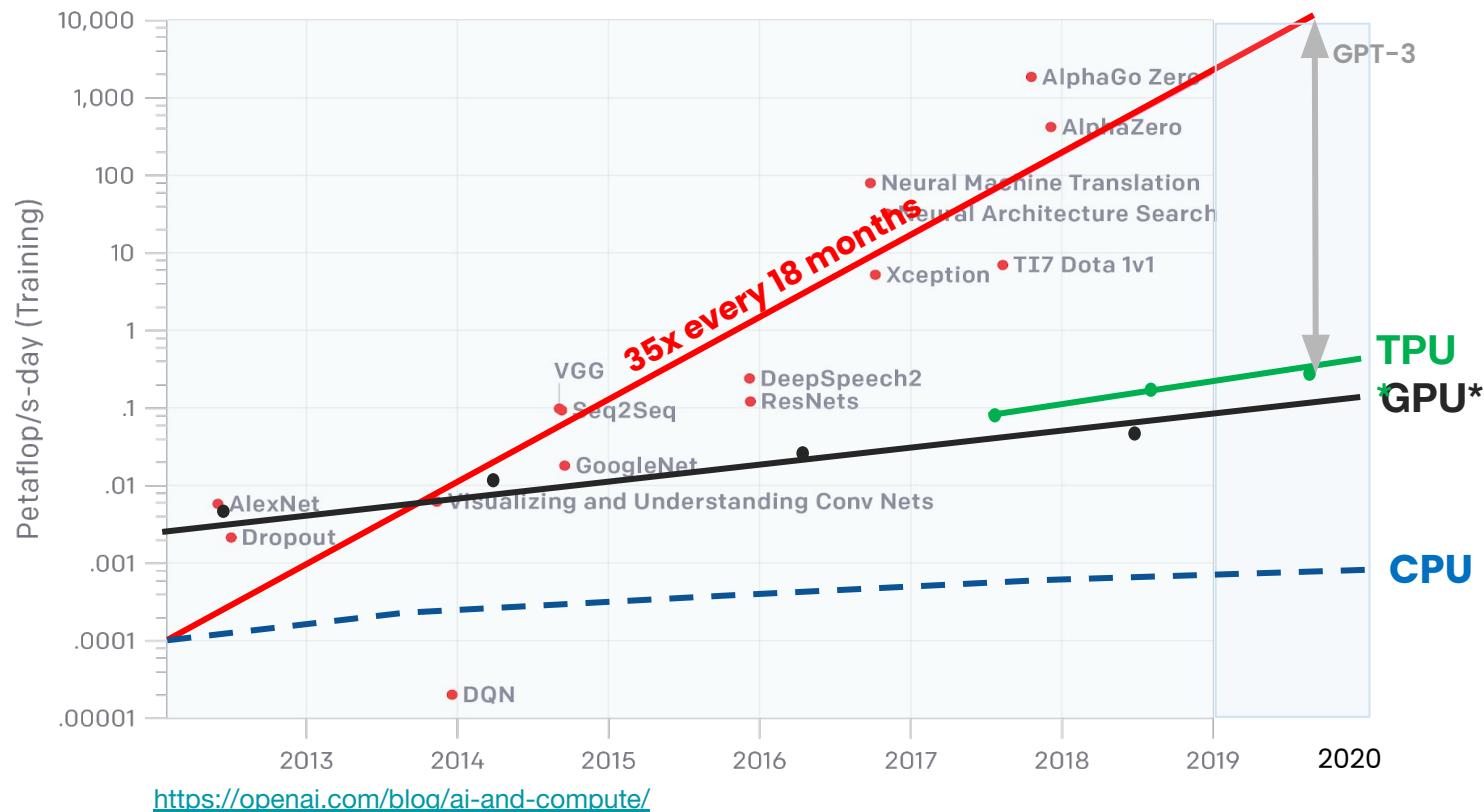
The blessings of scale

AI training runs, estimated computing resources used
Floating-point operations, selected systems, by type, log scale

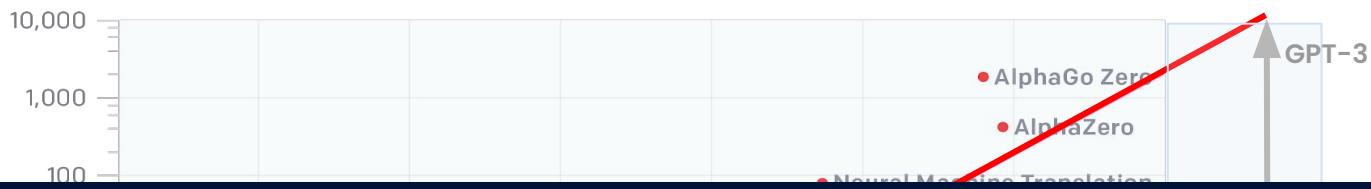


Sources: "Compute trends across three eras of machine learning", by J. Sevilla et al., arXiv, 2022; Our World in Data

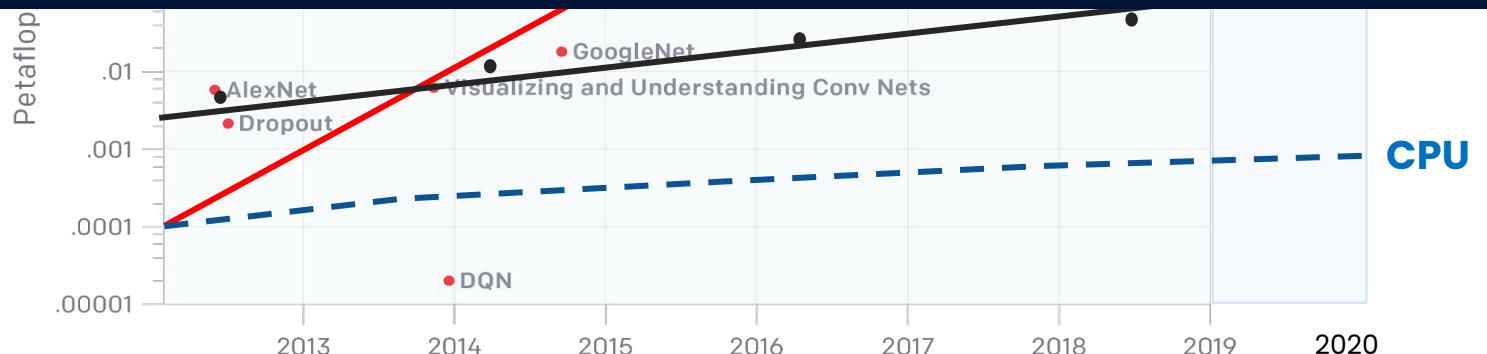
Compute - supply demand problem



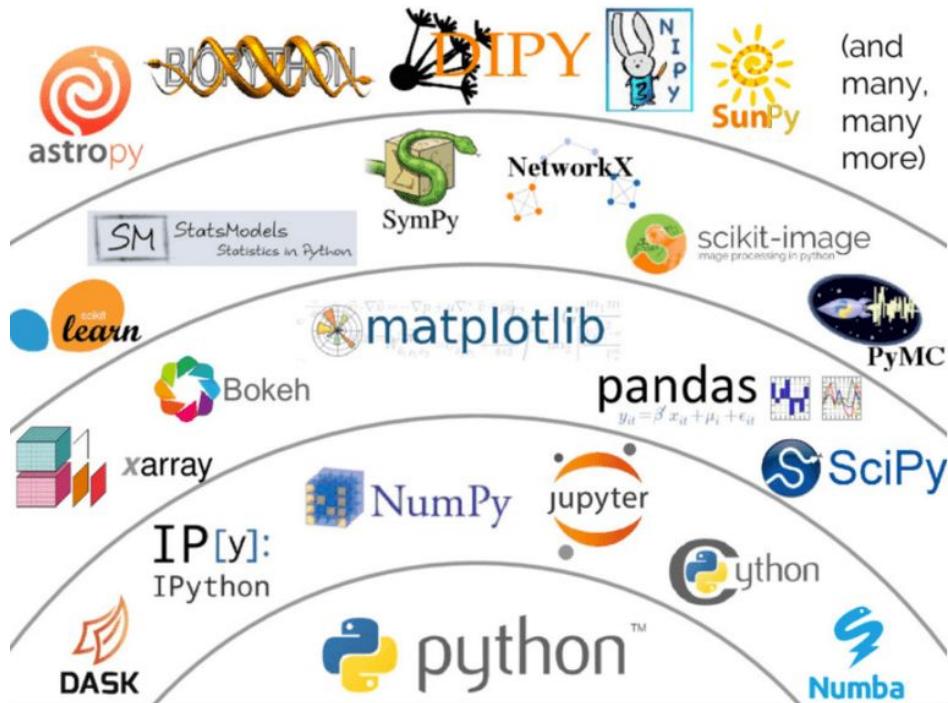
Specialized hardware is not enough



No way out but to distribute!



Python data science ecosystem



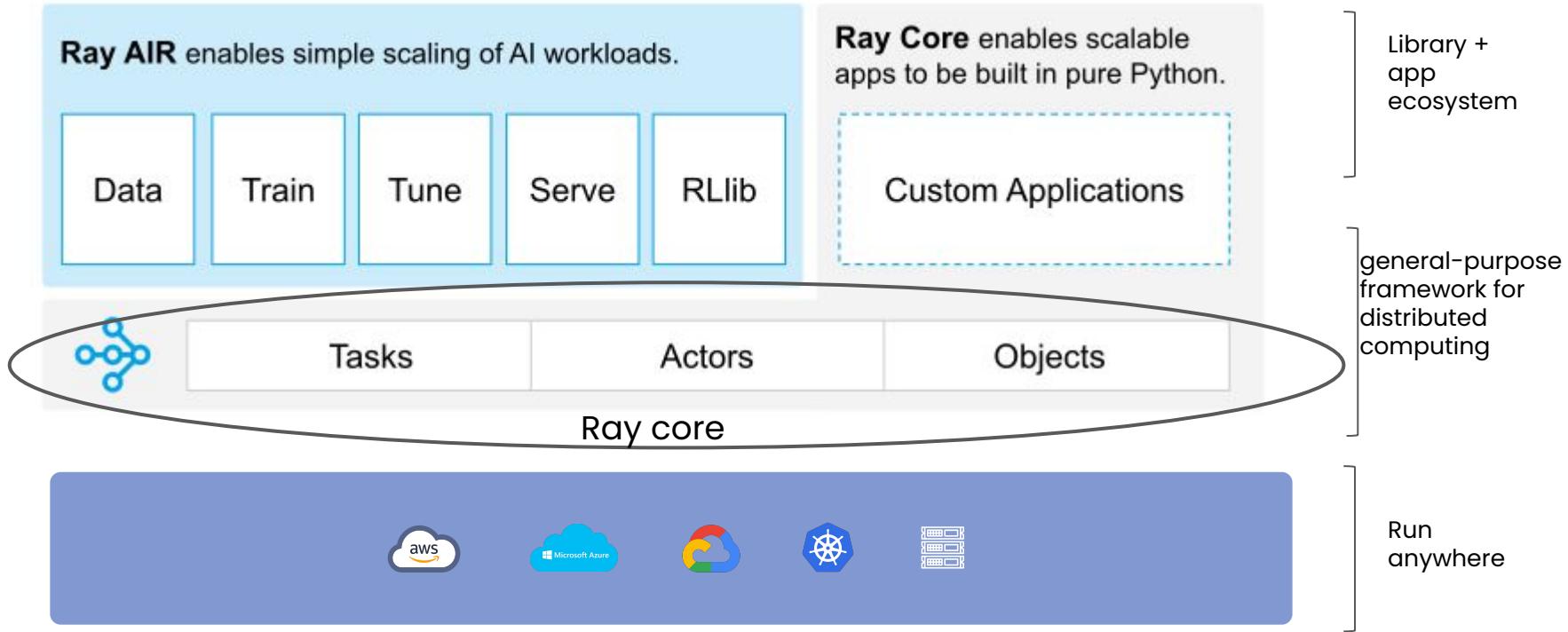
What is Ray



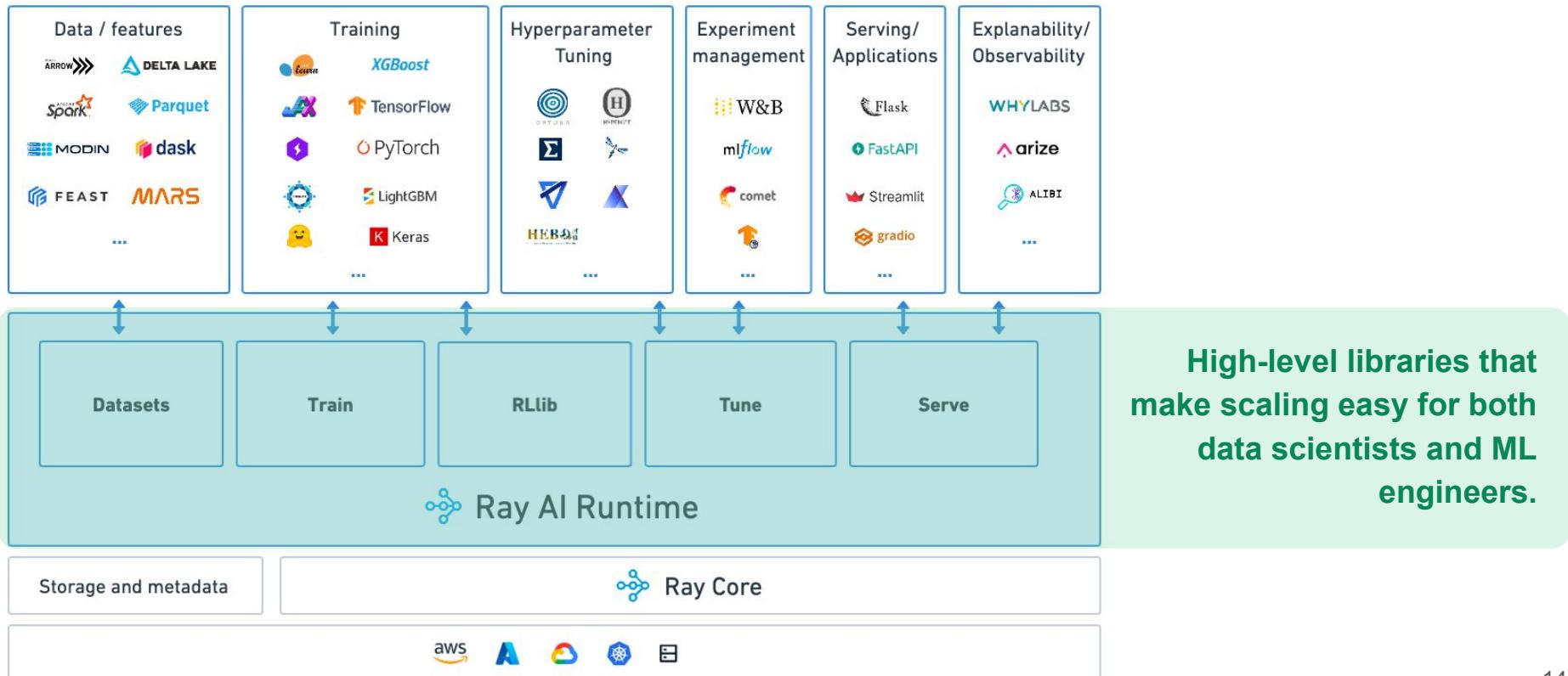
- A *simple/general-purpose* library for distributed computing
- An ecosystem of Python libraries (for scaling ML and more)
- Runs on laptop, public cloud, K8s, on-premise

A layered cake of functionality and capabilities for scaling ML workloads

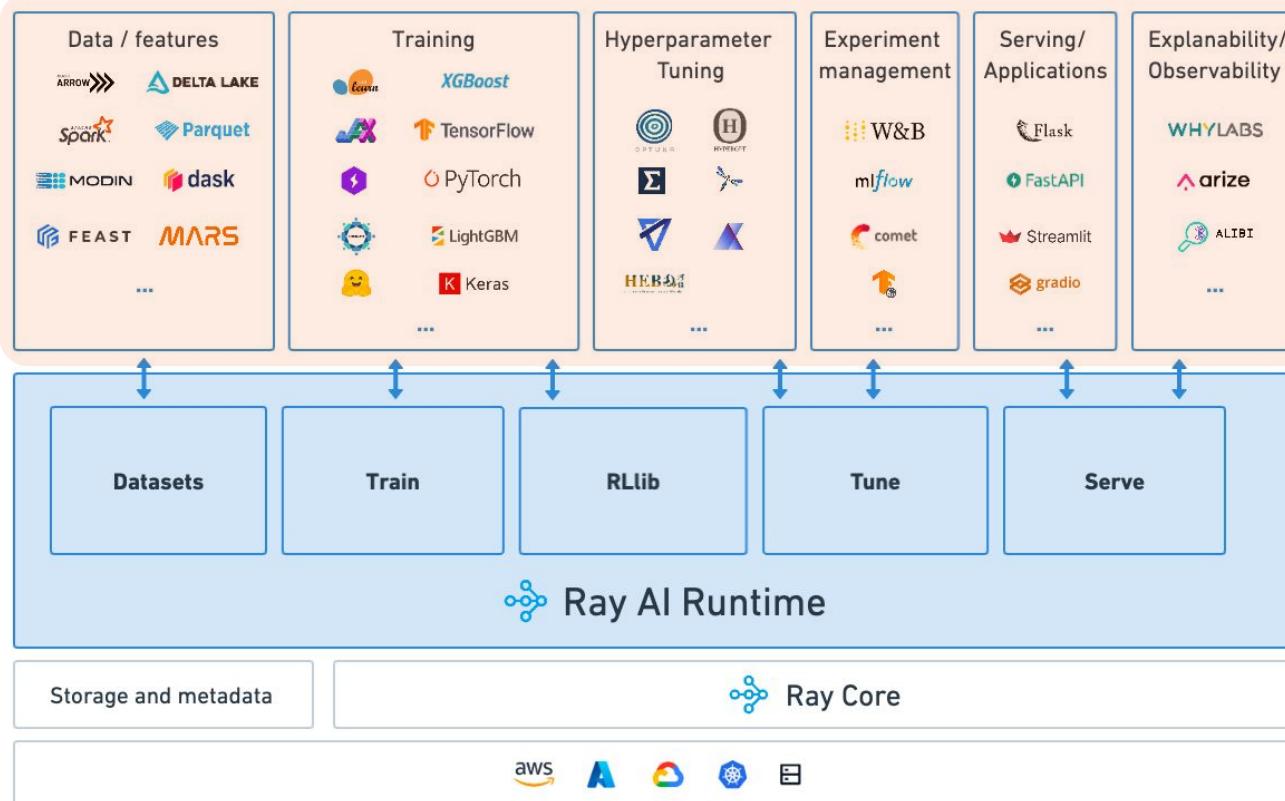
A Layered Cake and Ecosystem



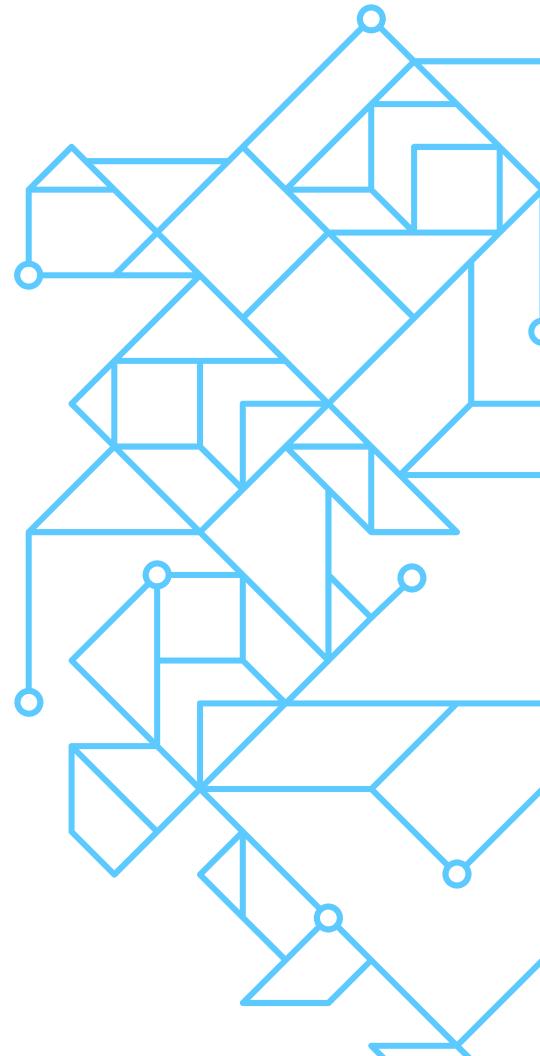
Ray AI Runtime (AIR) is a scalable runtime for end-to-end ML applications



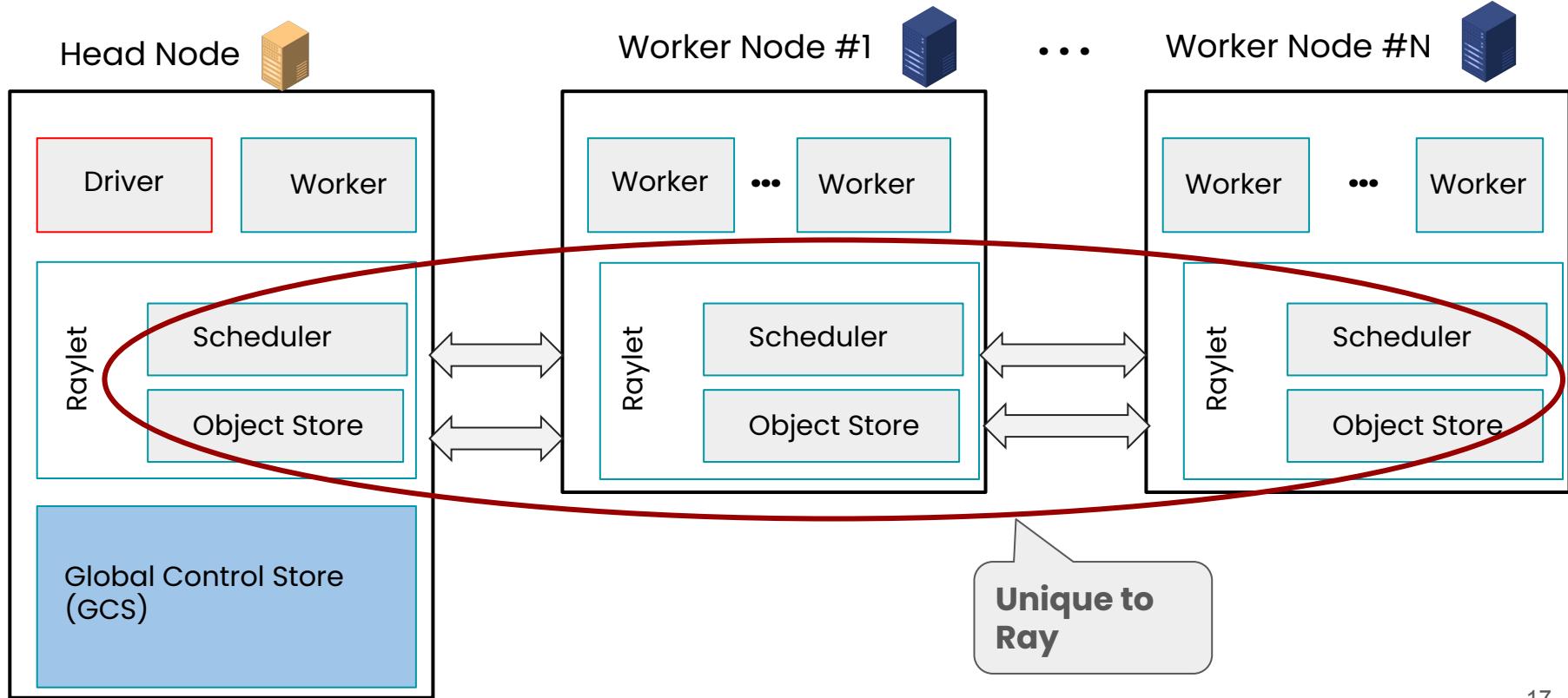
Ray AI Runtime (AIR) is a scalable toolkit for end-to-end ML applications



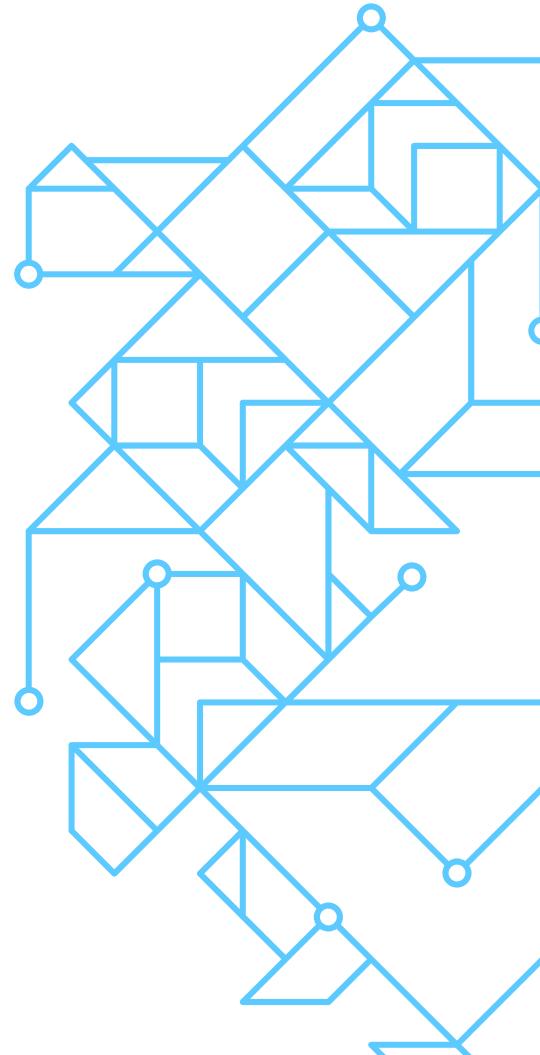
Ray Architecture & Components



An anatomy of a Ray cluster

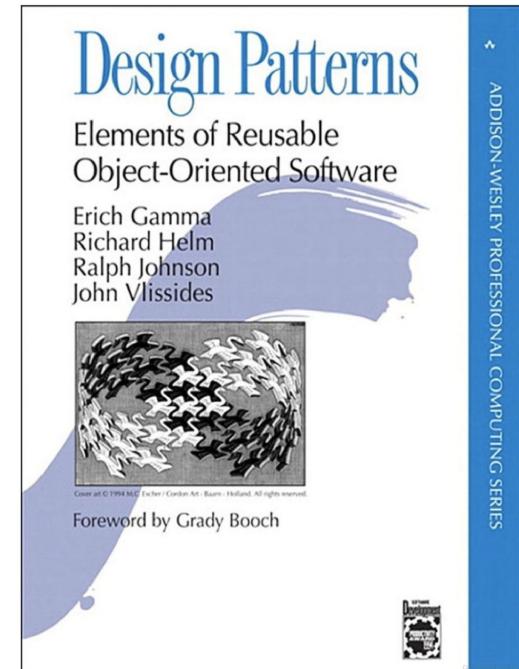


Ray distributed design & scaling patterns & APIs



Ray Basic Design Patterns

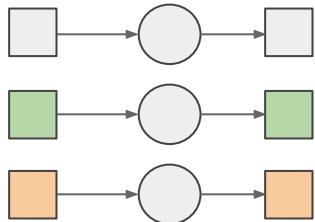
- Ray Parallel Tasks
 - Functions as stateless units of execution
 - Functions distributed across the cluster as tasks
- Ray Objects as Futures
 - Distributed (immutable objects) store in the cluster
 - Fetched when materialized
 - Enable massive asynchronous parallelism
- Ray Actors
 - Stateful service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



1. [Patterns for Parallel Programming](#)
2. [Ray Design Patterns](#)
3. [Ray Distributed Library Integration Patterns](#)

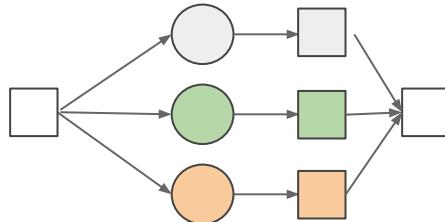
Scaling Design Patterns

Batch Training / Inference



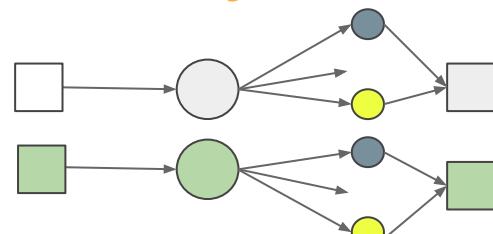
Different data / Same function

AutoML



Same data / Different function

Batch Tuning



Different data / Same function /

 Compute

 Data

Python → Ray APIs



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

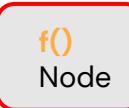
Task

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

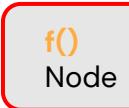


```
@ray.remote
class Cls():
    def __init__(self, x):
        def f(self, a):
            ...
            def g(self, a):
                ...
                cls = Cls.remote()
                cls.f.remote(a)
```

Distributed



...



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

Actor

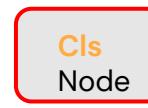
```
import numpy as np
a= np.arange(1, 10e6)
b = a * 2
```

Distributed
immutable
object

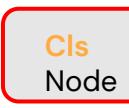


```
import numpy as np
a = np.arange(1, 10e6)
obj_a = ray.put(a)
b = ray.get(obj_a) * 2
```

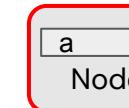
Distributed



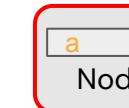
...



Distributed



...

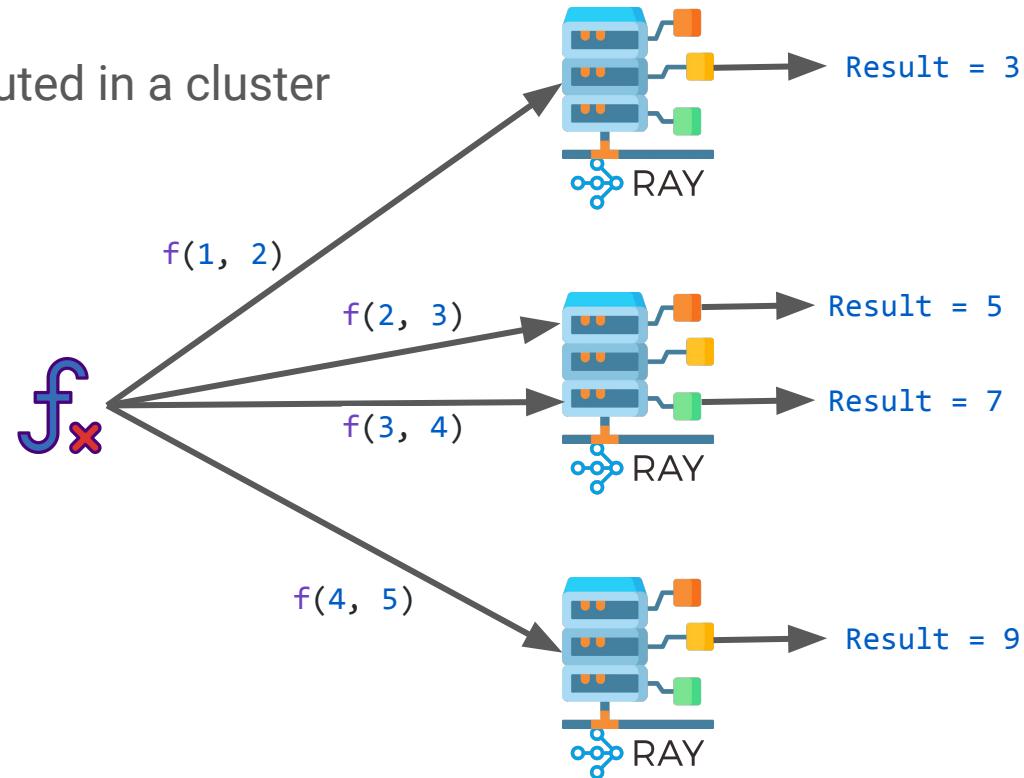


Ray Task

A function f_x remotely executed in a cluster

```
@ray.remote(num_cpus=2)
def f(a, b):
    return a + b

f.remote(1, 2) # returns 3
```



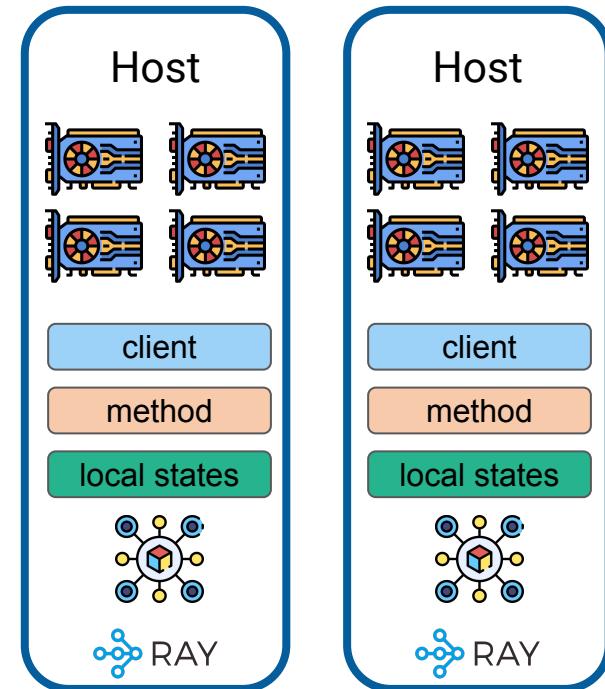
Ray Actor

A **class**  remotely executed in a cluster

```
@ray.remote(num_gpus=4)
class HostActor:
    def __init__(self):
        self.num_devices = os.environ["CUDA_VISIBLE_DEVICES"]

    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"
```



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)
```

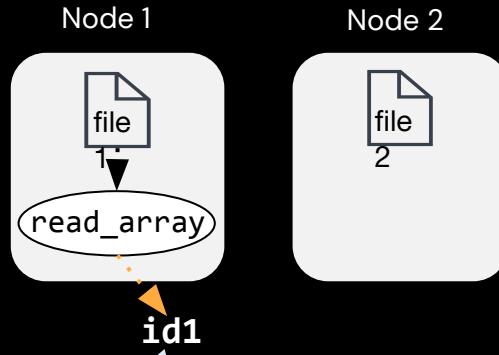
Class → Actor

```
@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

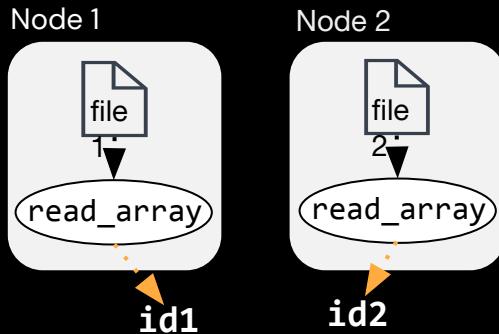
```
@ray.remote  
def read_array(file):  
    # read ndarray "a"  
    # from "file"  
    return a  
  
@ray.remote  
def add(a, b):  
    return np.add(a, b)  
  
id1 = read_array.remote(file1)  
id2 = read_array.remote(file2)  
id = add.remote(id1, id2)  
sum = ray.get(id)
```



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

Task API

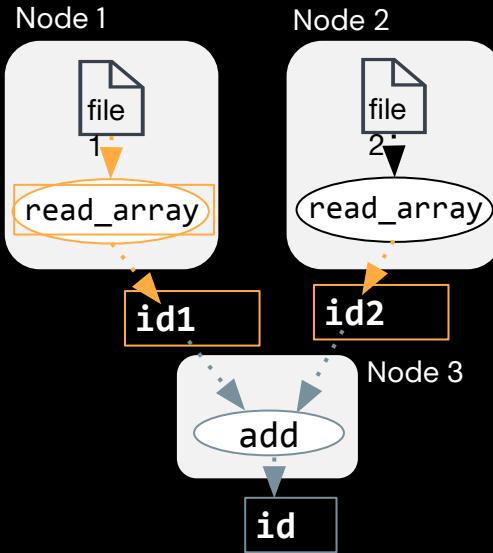
```
@ray.remote  
def read_array(file):  
    # read ndarray "a"  
    # from "file"  
    return a  
  
@ray.remote  
def add(a, b):  
    return np.add(a, b)  
  
id1 = read_array.remote(file1)  
id2 = read_array.remote(file2)  
id = add.remote(id1, id2)  
sum = ray.get(id)
```



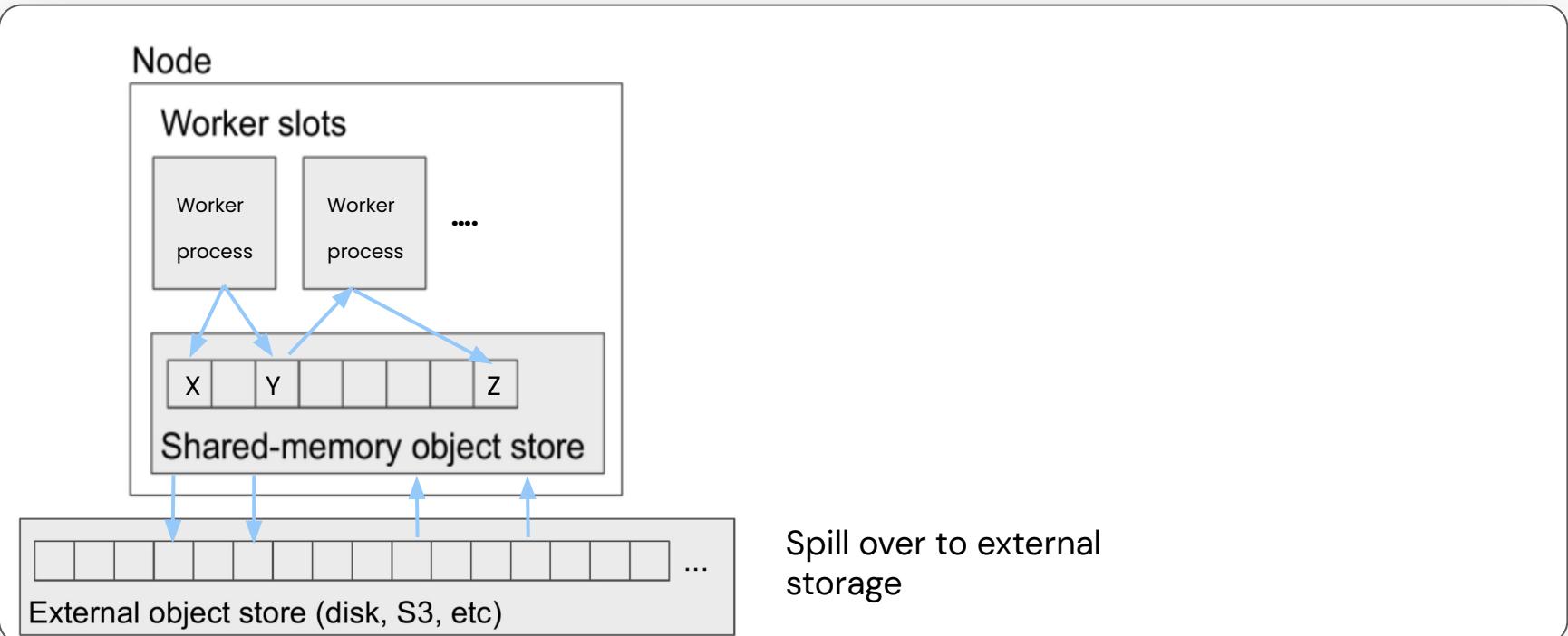
Dynamic task graph:
build at runtime

Task API

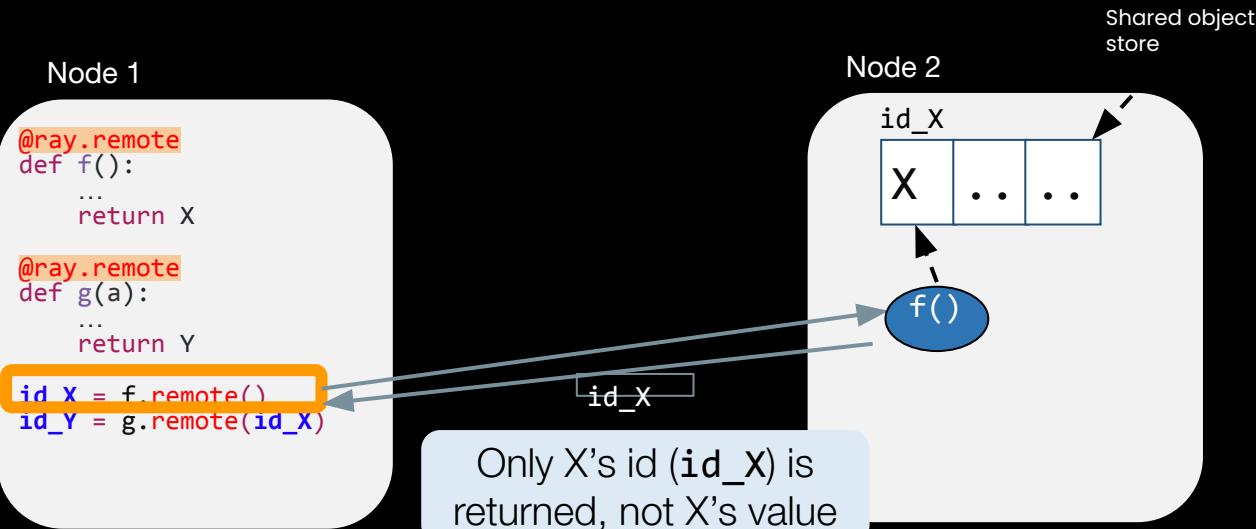
```
@ray.remote  
def read_array(file):  
    # read ndarray "a"  
    # from "file"  
    return a  
  
@ray.remote  
def add(a, b):  
    return np.add(a, b)  
  
id1 = read_array.remote(file1)  
id2 = read_array.remote(file2)  
id = add.remote(id1, id2)  
sum = ray.get(id) → ray.get() block until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



Distributed Immutable object store

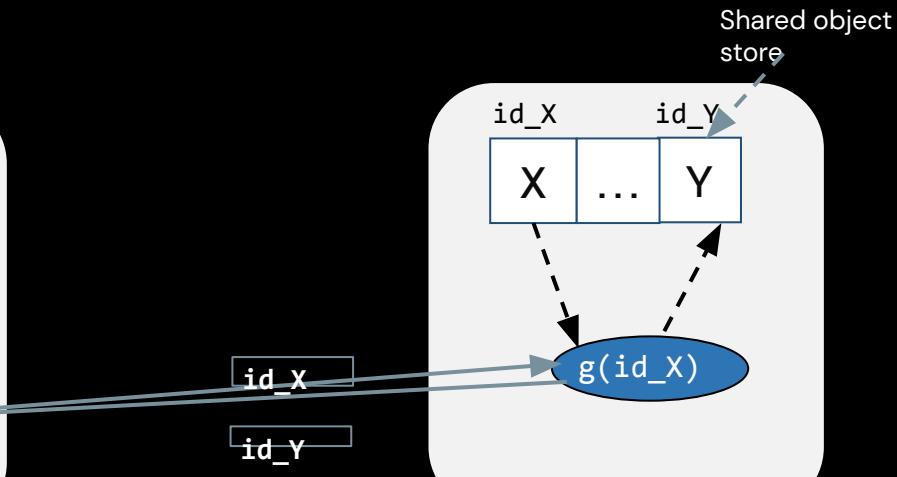


Distributed object store



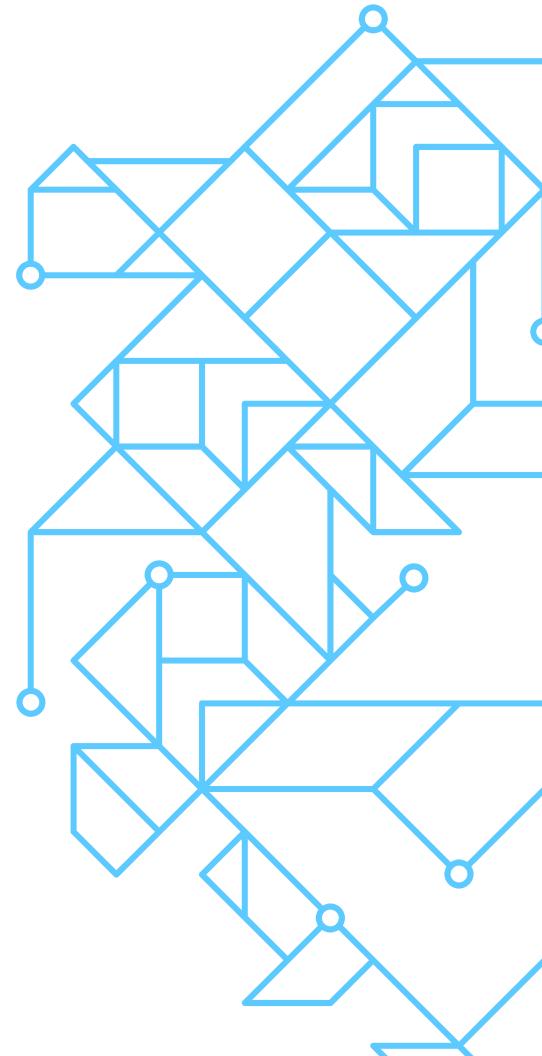
Distributed object store

```
@ray.remote  
def f():  
    ...  
    return X  
  
@ray.remote  
def g(a):  
    ...  
    return Y  
  
id_X = f.remote()  
id_Y = g.remote(id_X)
```



`g(id_X)` is scheduled on same node, so `X` is never transferred

Examples of Distributed Applications with Ray



Distributed Applications with Ray

ML Libraries

- Ray AI Runtime
- Distributed scikit-learn/Joblib
- Distributed XGBoost on Ray
- Ray Multiprocess Pool

All using Ray core APIs & patterns

Experimenting & Monitoring Services

- WhyLabs
- Arize AI
- W & B
- MLflow

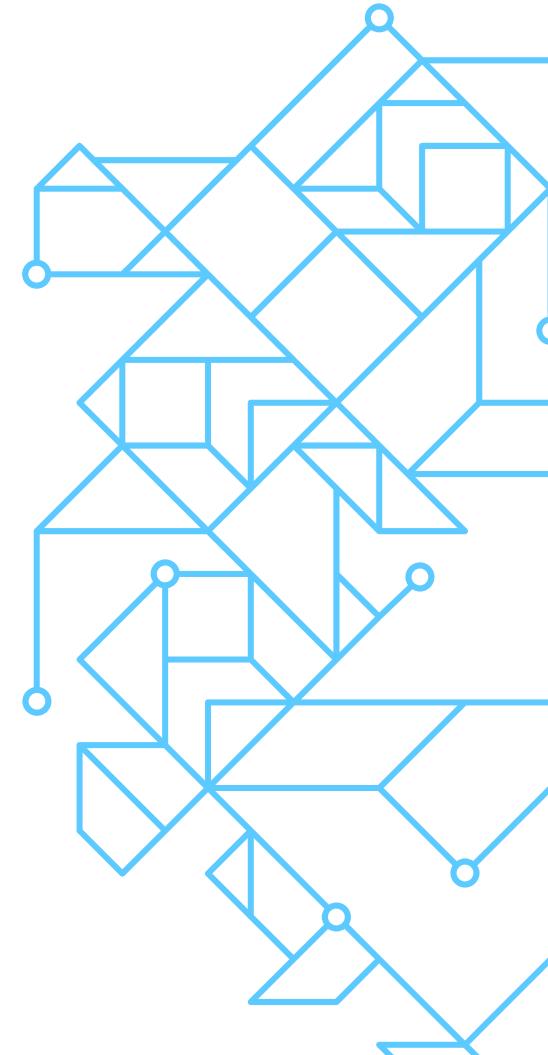
All using Ray core APIs & patterns

ML Platforms & Integrations

- Merlin (Shopify)
- Meta (TorchX)
- AirFlow, Prefect
- HuggingFace
- Ludwig AI
- Uber
- Instacart
- Spotify

All using Ray core APIs & patterns

Generative AI, LLMs & Ray



Current state of the world..

LLM companies provide commercial APIs
(note: co:here and OpenAI both use Ray internally)

co:here



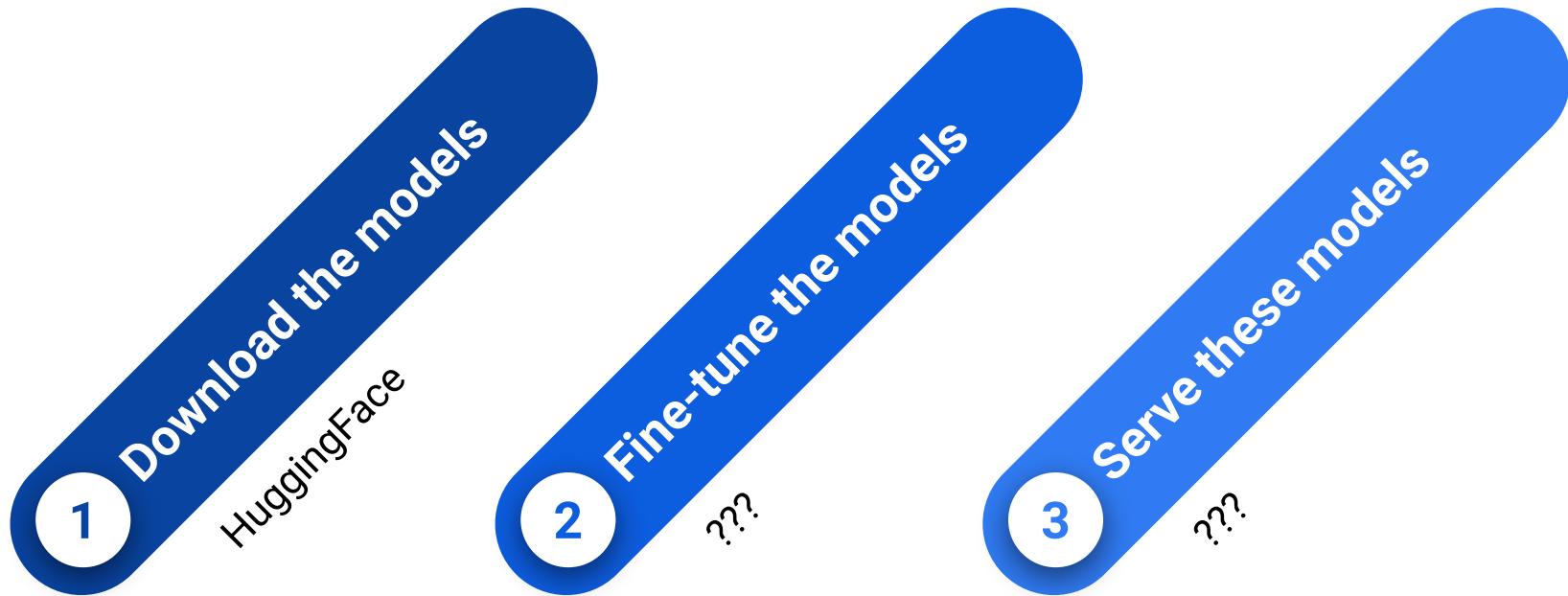
ANTHROPIC



Hugging Face



Options for poor mortals ...



What are pain points in training/serving LLMs?

01

Scaling is costly and hard to manage

- Need spot instance support
- Hard to run distributed workloads
- Hard to optimize CPUs/GPUs

02

Existing serving / inference solutions don't work

- Individual replicas can't be distributed
- Need to be able to integrate business logic

03

Distributed Training hard to get working right

- Hyperparameters need to be tuned
- Need a platform to iterate very quickly at scale

Ray provides generic platform for LLMs

01

Simplify orchestration and scaling

- Spot instance support for data parallel training
- Easily spin up and run distributed workloads on any cloud
- Optimize CPUs/GPUs by pipelining w/ Ray Data

02

Inference and serving

- Ability to support complex pipelines integrating business logic
- Ability to support multiple node serving

03

Training

- Integrates distributed training with distributed hyperparameter tuning w/ ML frameworks

What's the LLM stack for Generative AI?



Technical stacks for Large language models (LLMs)



Model definition



Automatic model parallelization



GPU management and runtime orchestration



Compilation and runtime



GPU accelerator

HuggingFace for models

DeepSpeed for optimized Training

PyTorch for Framework

Ray for Orchestration

GPU or other hardware

Figure 5: Technical integration layered stack for LLM

What about Generative AI?

How Ray solves common production challenges for generative AI infrastructure

By Antoni Baum, Eric Liang, Jun Gong, Kai Fricke and Richard Liaw | March 20, 2023

This is part 1 of our generative AI blog series. In this post, we talk about how to use Ray to productionize common generative model workloads. An upcoming blog will deep dive into why projects like Alpa are using Ray to scale large models.

Training 175B Parameter Language Models at 1000 GPU scale with Alpa and Ray

By Jiao Dong, Hao Zhang, Lianmin Zheng, Jun Gong, Jules S. Damji and Phi Nguyen | March 22, 2023

This is part 2 of our generative AI blog series. Here we cover how Ray empowers large language models (LLM) frameworks such as Alpa. To learn how to use Ray to productionize generative model workloads, see part 1.

<https://anyscale.com/blog>

Faster stable diffusion fine-tuning with Ray AIR

By Kai Fricke | March 28, 2023

This is part 3 of our generative AI blog series that dives into a concrete example of how you can use Ray to scale the training of generative AI models. To learn more using Ray to productionize generative model workloads, see part 1. To learn about how Ray empowers LLM frameworks such as Alpa, see part 2.

How to fine tune and serve LLMs simply, quickly and cost effectively using Ray + DeepSpeed + HuggingFace

By Waleed Kadous, Jun Gong, Antoni Baum and Richard Liaw | April 10, 2023

This is part 4 of our blog series on Generative AI. In the previous blog posts we explained why Ray is a sound platform for Generative AI, we showed how it can push the performance limits, and how you can use Ray for stable diffusion.

Ray and Anyscale emerging as a standard for ML infrastructure

How Ray, a Distributed AI Framework, helps Power ChatGPT

Ray breaks the \$1/TB barrier as the world's most cost-efficient sorting system

By Frank Sifei Luan, UC Berkeley

The Magic of Merlin: Shopify's New Machine Learning Platform

by Isaac Vidas · Data Science & Engineering · Aug 6, 2020 · 11 minute read

Unleashing ML Innovation at Spotify with Ray



Contact Us Support My Account

re:Invent Products Solutions Pricing Documentation Learn More

Blog Home

Category ▾

Edition ▾

AWS Big Data Blog

Introducing AWS Glue for Ray: Scaling your data integration workloads using Python

by Zach Mitchell, Ishan Gaur, Kinsuk Bhakare, and Derek Liu | Oct 28, 2020 · 11 minute read

Google Cloud Blog

Large Scale Deep Learning Training and Tuning with Ray at Uber

Xu Ning, Di Yu, Michael Mui



Han Li

Mar 17 · 9 min read · Listen



Distributed Machine Learning at Instacart

How Instacart uses distributed Machine Learning to efficiently thousands of models in production

Author



Netflix Technology Blog

Feb 13 · 11 min read · Listen



Scaling Media Machine Learning at Netflix

Ray: Fastest Growing Scalable Compute Framework



McKinsey
& Company



ERICSSON



cruise

Morgan Stanley



J.P.Morgan



Uber



RICARDO



verizon[✓]



24,000+

GitHub
stars

700+

Community
Contributors

5,000+

Repositories
Depend on Ray

1,000+

Organizations
Using Ray

Key Takeaways

- Distributed computing is a necessity & norm
- Ray's vision: make distributed computing simple
 - Don't have to be distributed programming expert
- Build your own disruptive apps & libraries with Ray
- Scale your ML workloads with Ray libraries (Ray AIR)
- Ray offers the compute substrate for Generative AI workloads

Ray Summit 2023



SPEAKERS

TRAINING

SPONSORS

WHO ATTENDS

REGISTER NOW

THE PLACE FOR EVERYTHING RAY

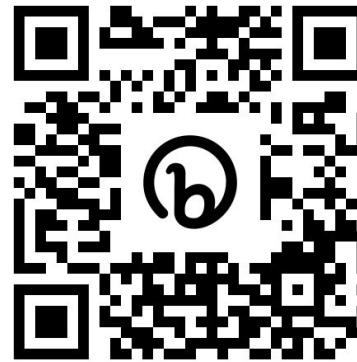
San Francisco Marriott Marquis | September 18-20

AI is moving fast. Get in front of what's next at Ray Summit 2023. Join the global Ray community in San Francisco for keynotes, Ray deep dives, lightning talks and more exploring the future of machine learning and scalable AI.

REGISTRATION IS OPEN



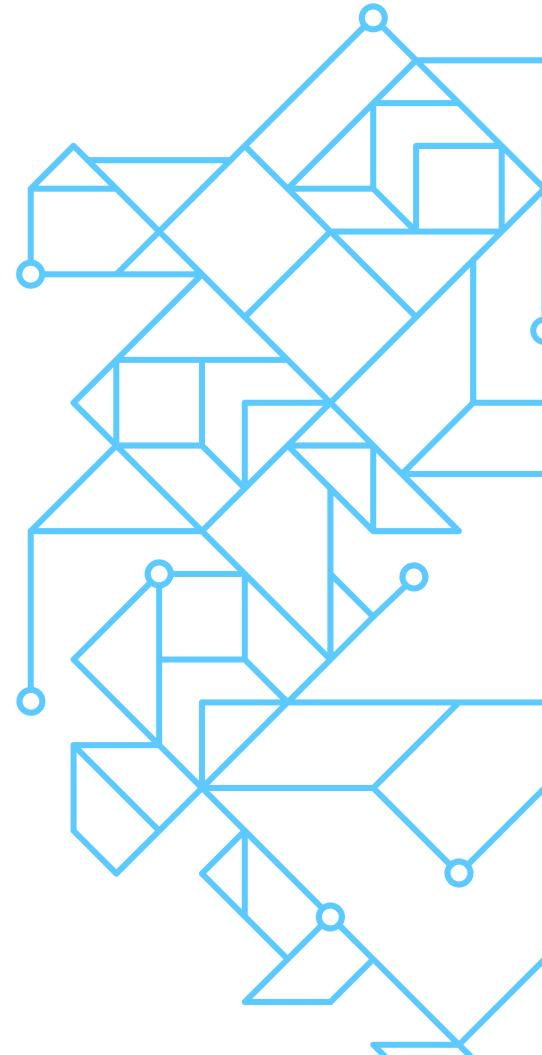
<https://bit.ly/raysummit2023>



Let's go with



<https://bit.ly/pydata-seattle-tutorial-2023>



Thank you!

Questions?

email: jules@anyscale.com

twitter: [@2twitme](https://twitter.com/2twitme)

