PyData New York City 2022

Distributed Python with Ray: Hands on with the Ray 2.0 APIs for scaling Python Workloads

Jules S. Damji - @2twitme

Nov 11, 2022, New York, NY

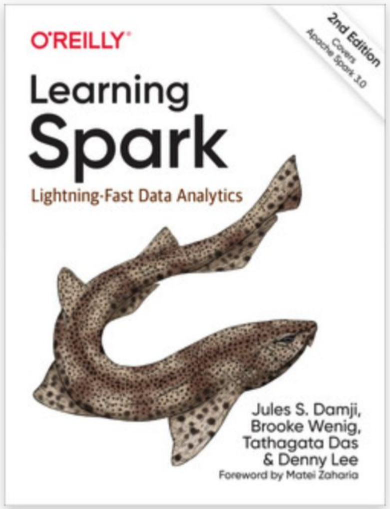
Few Important URLs

Keep these URLs open in your browser tabs

- → Ray Core Class Survey: https://bit.ly/pydata-nyc-2022
- → GitHub: https://bit.ly/pydata-nyc-tutorial-2022
- → 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



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 Patterns & APIs
- Modules [1] Hand-on in class
- Modules [2-3] Extra Curriculum at home

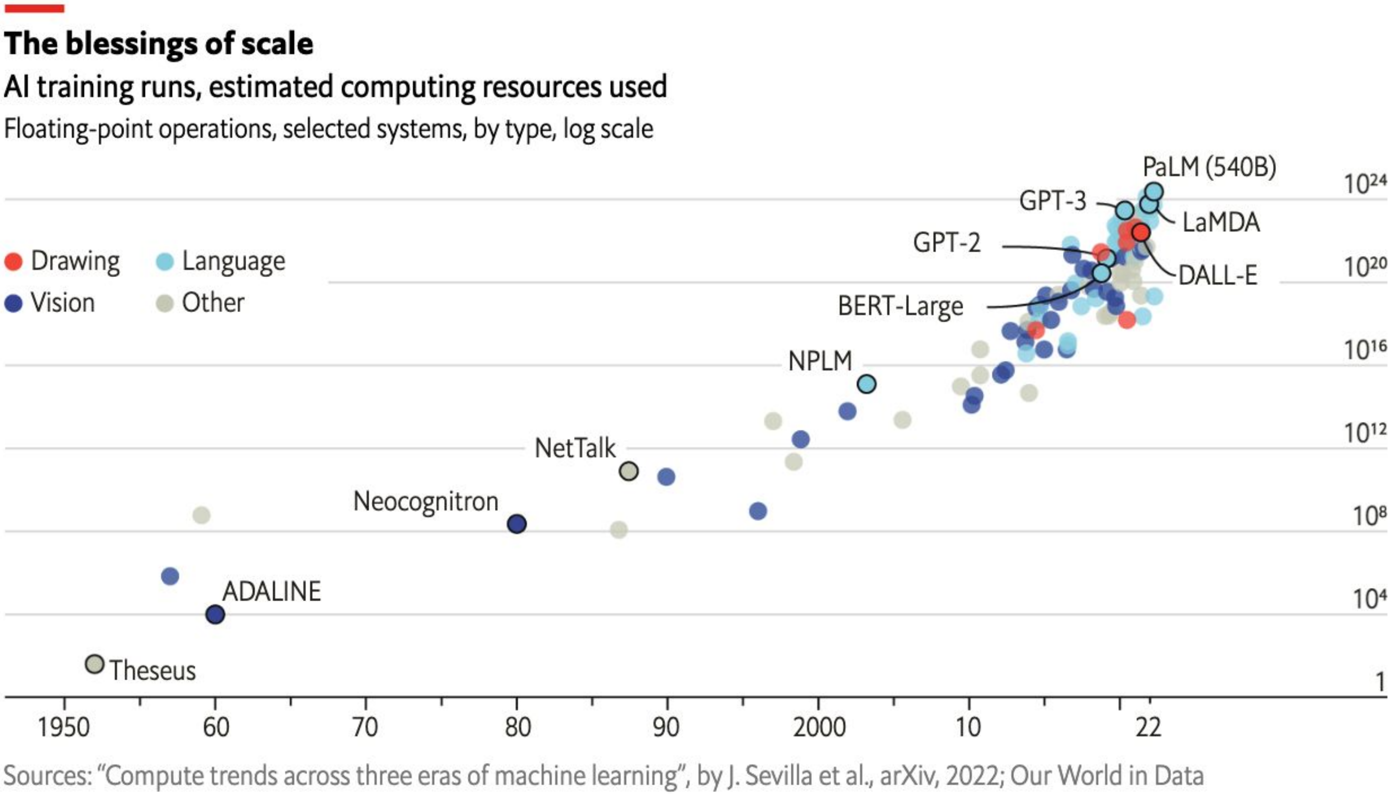
Why Ray

Machine learning is pervasive

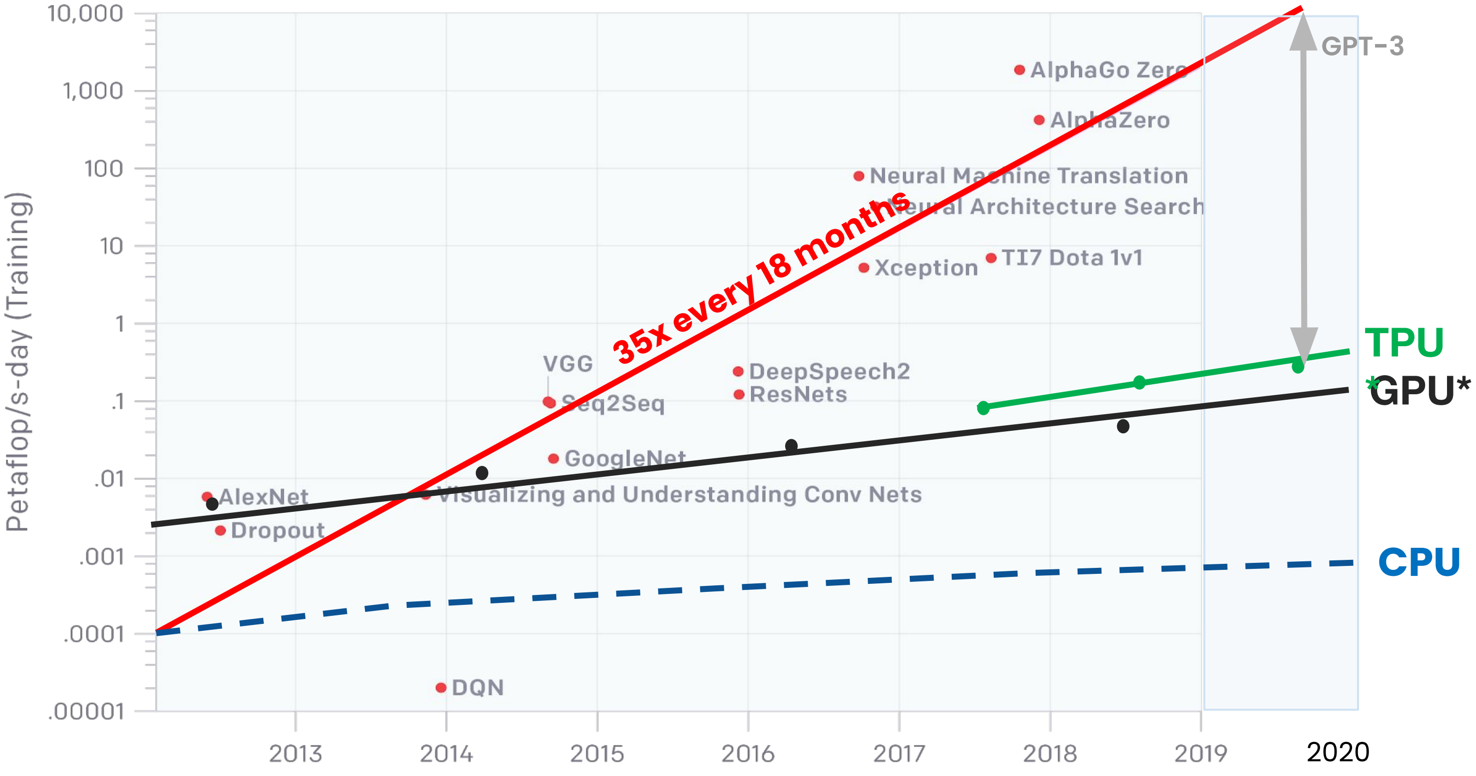
Distributed computing is a necessity

Python is the default language for DS/ML

Blessings of scale …

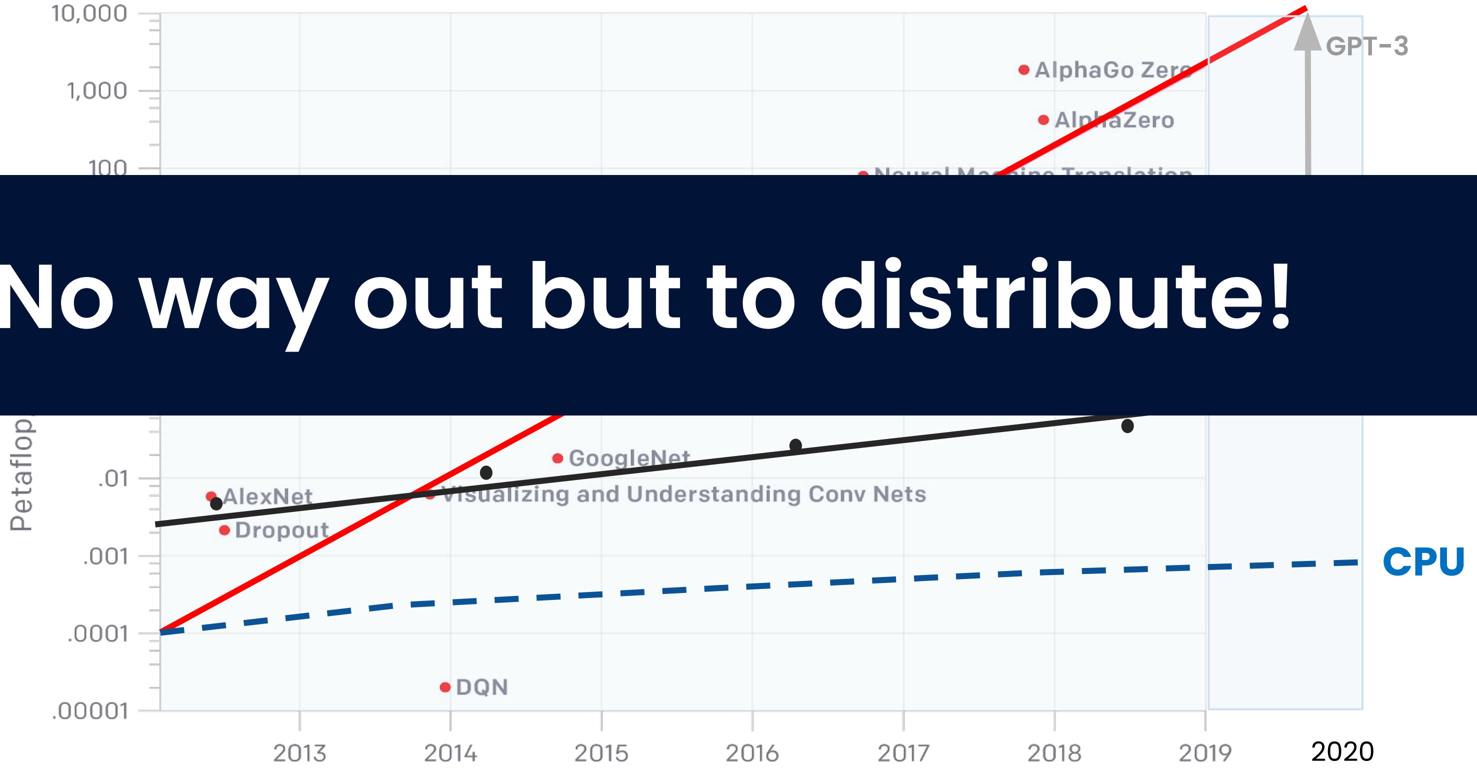


Compute - supply demand problem



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

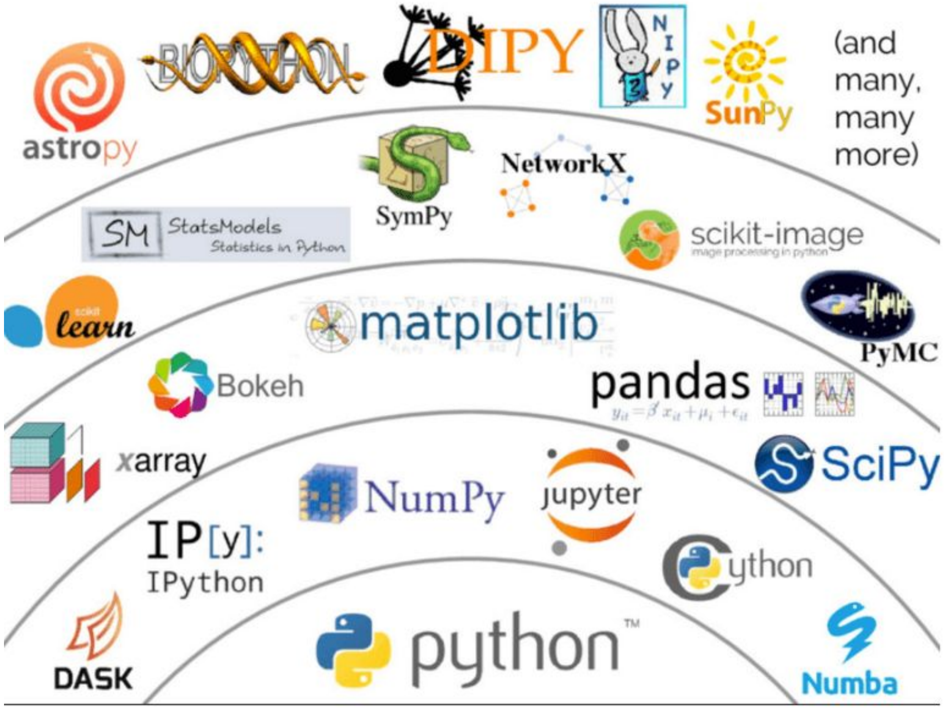
Specialized hardware is not enough



No way out but to distribute!

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

Python data science ecosystem

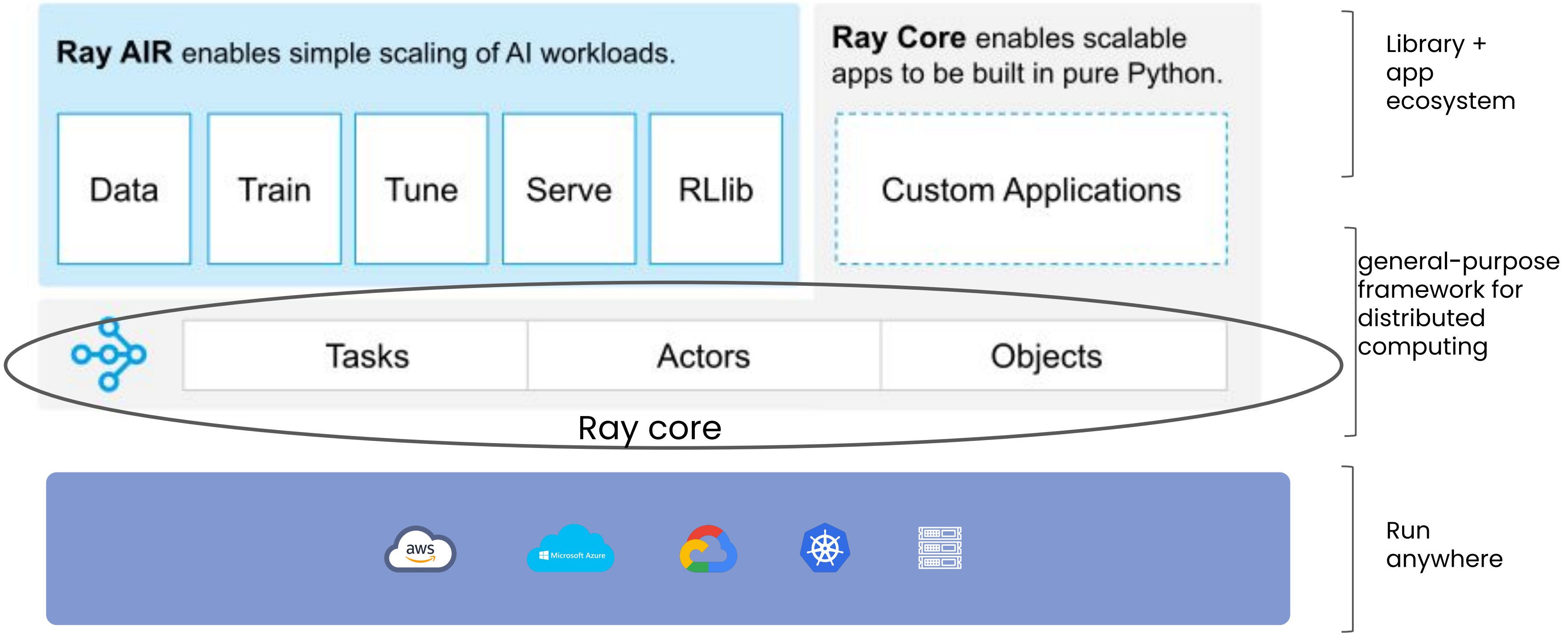


What is Ray

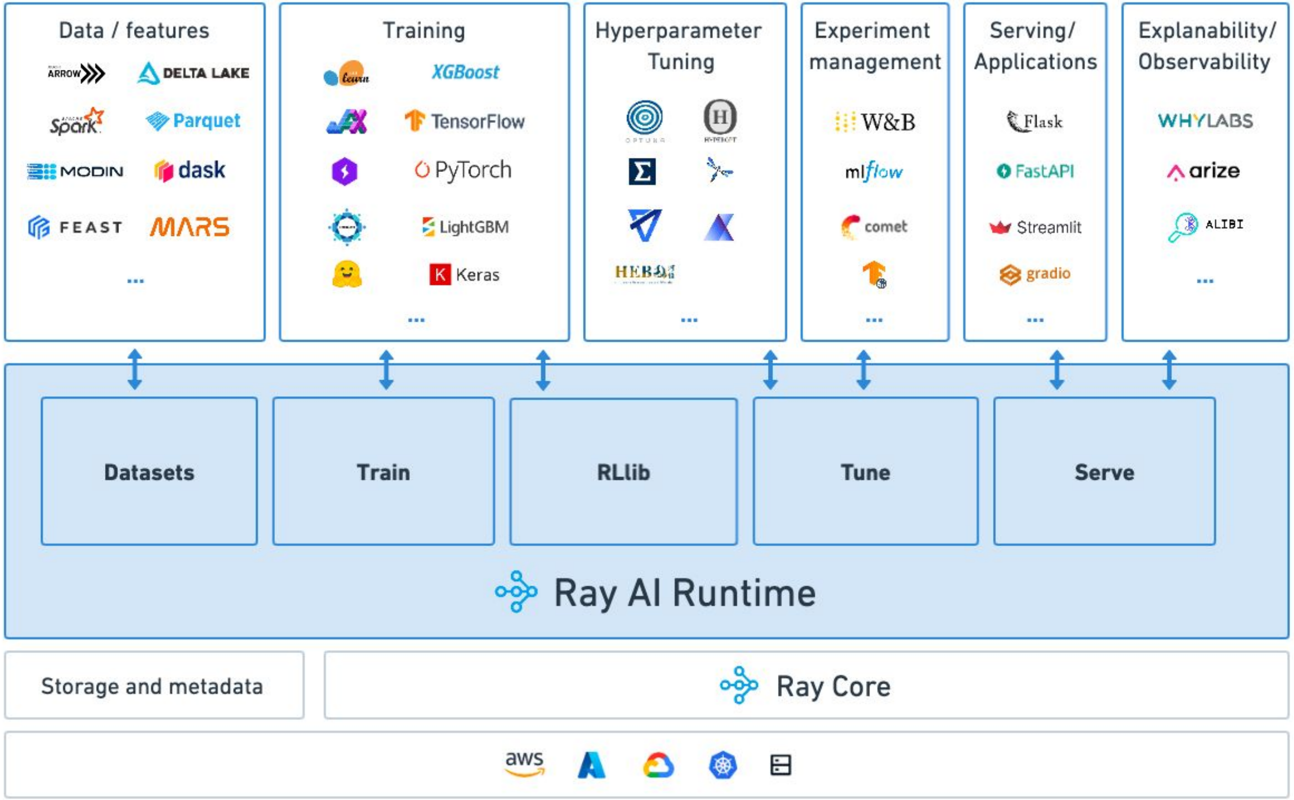
- A simple/general-purpose library for distributed computing
- An ecosystem of Python libraries (for scaling ML and more)
- Runs on laptop, public cloud, K8s, on-premise

A layered cake of functionality and capabilities for scaling ML workloads

A Layered Cake and Ecosystem

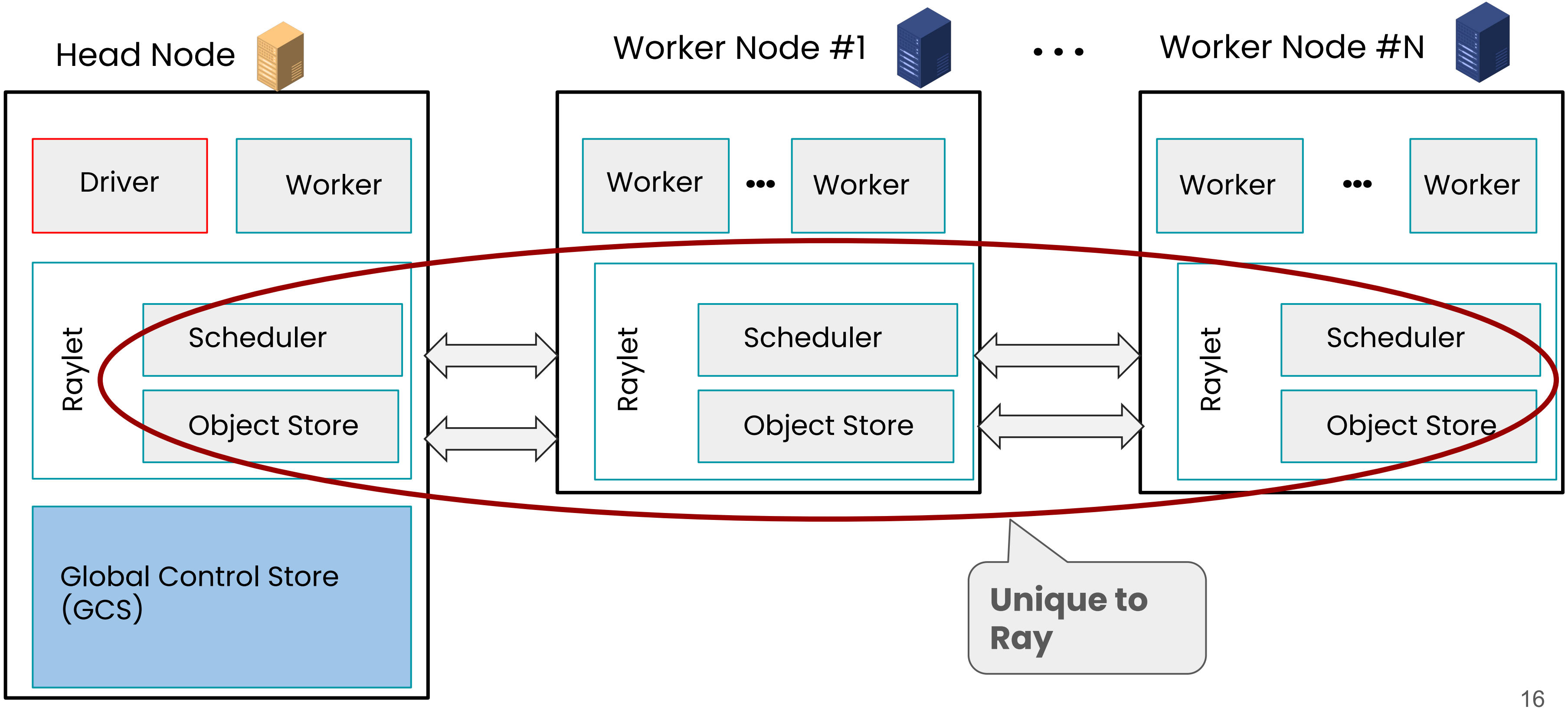


Ray AI Runtime (AIR) is a scalable runtime for end-to-end ML applications



Ray Architecture & Components

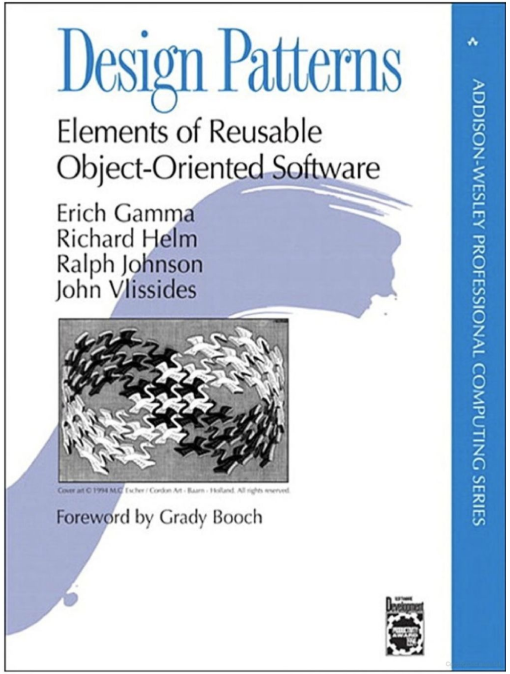
An anatomy of a Ray cluster



Ray distributed design 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]

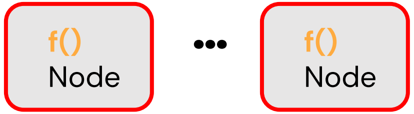
Python → Ray APIs

```
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()
```

Distributed

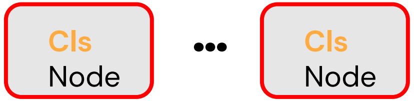



```
class Cls():
  def
__init__(self, x):
  def f(self, a):
      ...
  def g(self, a):
      ...
```

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

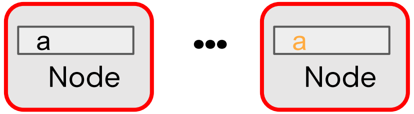



```
import numpy as np
a= np.arange(1, 10e6)
b = a * 2
```

Distributed immutable object

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

Distributed



Function → Task

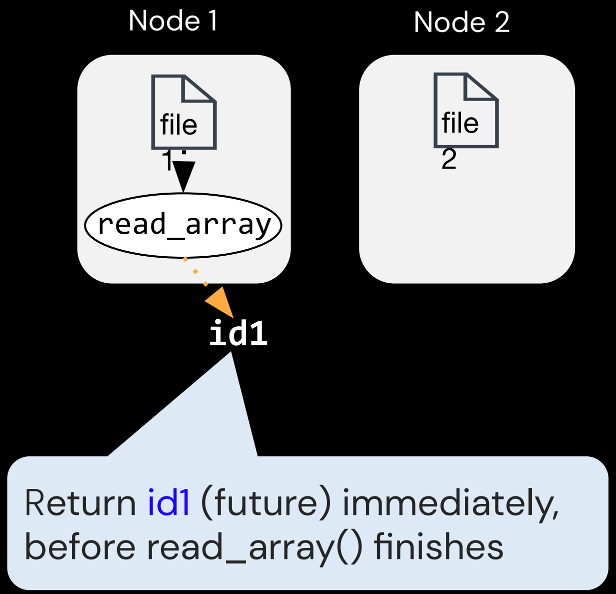
Class → Actor

```
@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.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()
```

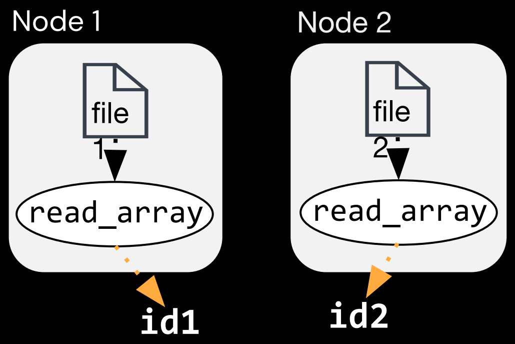
Task API

```
@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)
```

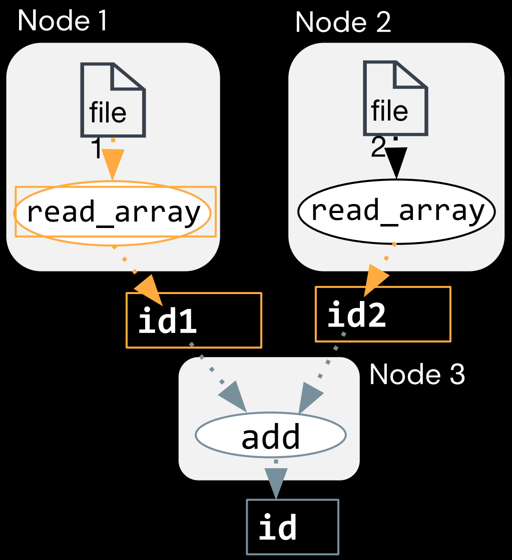


Dynamic task graph: build at runtime

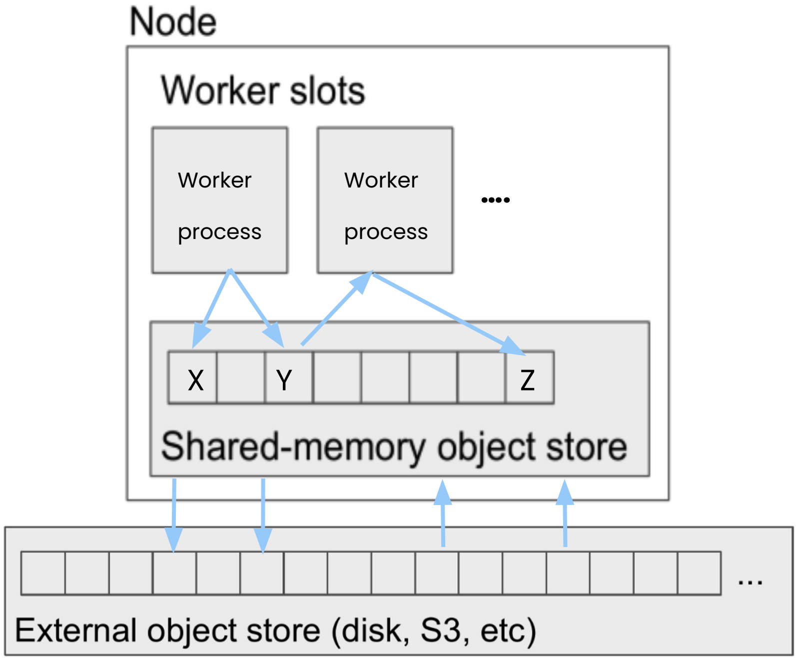
Task API

```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)
```

`ray.get()` block until result available

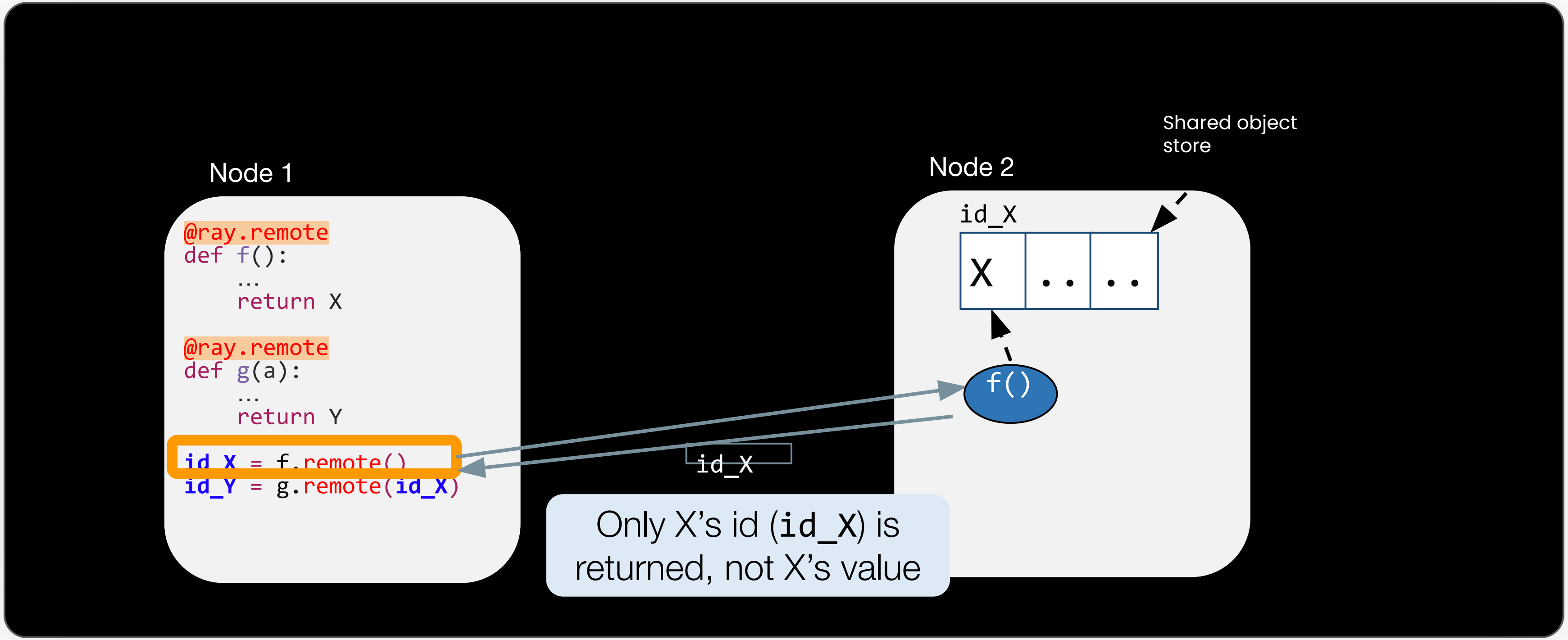


Distributed Immutable object store

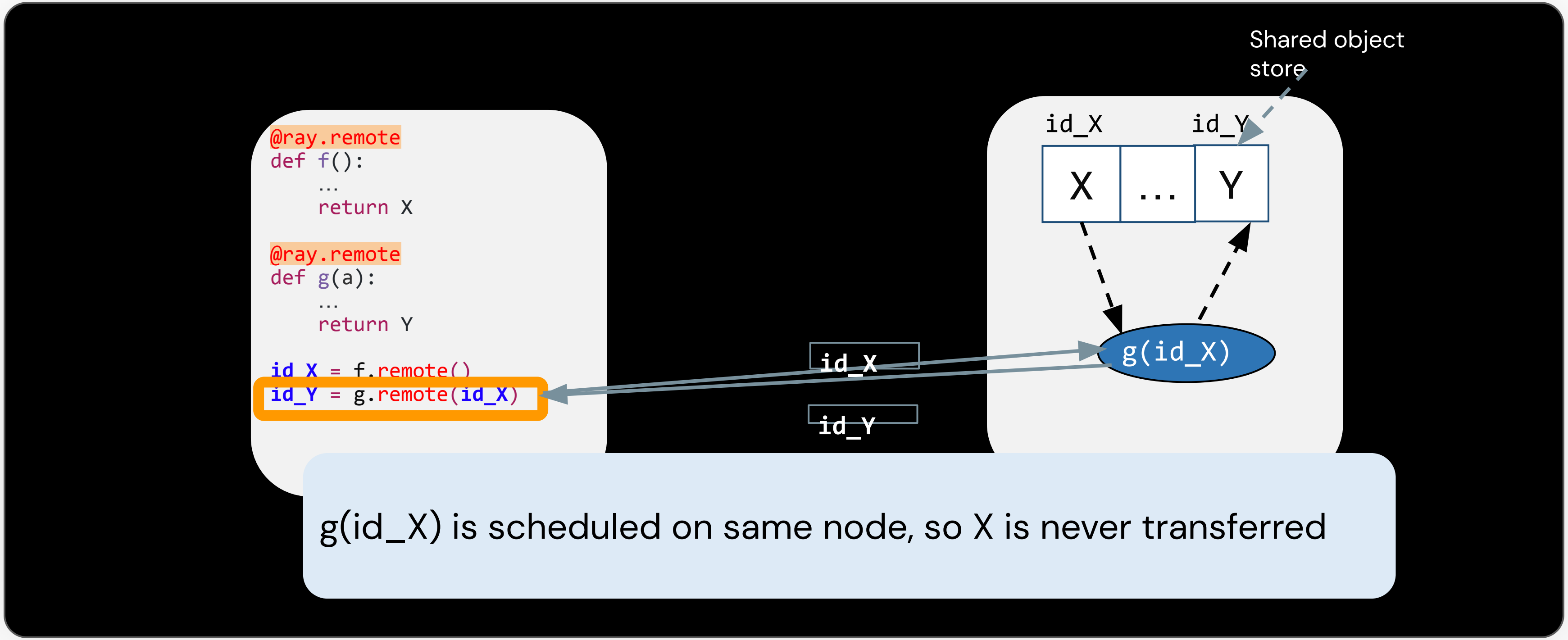


Spill over to external storage

Distributed object store



Distributed object store



Examples of Distributed Applications with Ray

Distributed Applications with Ray

ML Libraries

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

All using Ray core APIs & patterns

Monitoring Services

- WhyLabs
- Arize AI
- W & B

All using Ray core APIs & patterns

ML Platforms & Integrations

- Merlin (Shopify)
- Zero Copy (IBM)
- Meta
- MLflow, Comet
- AirFlow
- HuggingFace
- Pycaret
- Ludwig AI
- Uber
- Instacart
- Shopify
- Spotify

All using Ray core APIs & patterns

Ray Ecosystem: https://docs.ray.io/en/latest/ray-overview/ray-libraries.html

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

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)

Let's go with

https://bit.ly/pydata-nyc-tutorial-2022

Thank you!

Questions?

email: jules@anyscale.com

twitter: @2twitme