Ray: A distributed framework for scaling AI & Python workloads

UC San Diego

Jules S. Damji - @2twitme
May 18, 2023, Mandeville Auditorium, UCSD

Few Important URLs

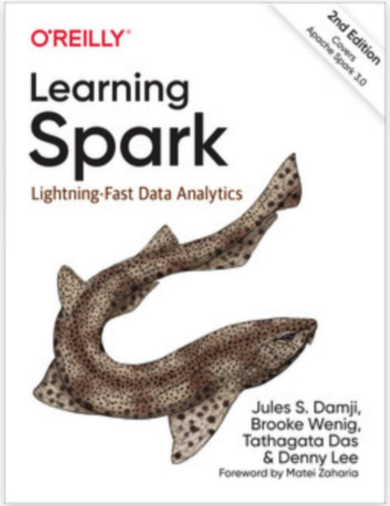
Keep these URLs open in your browser tabs

→ GitHub: https://github.com/dmatrix/ray-core-tutorial

→ 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

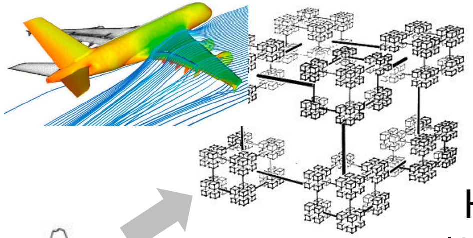
Agenda

- Evolution of Distributed & Cloud Computing
- Distributed Computing: *Necessity not a norm*
- Why Ray & what is Ray
- Ray Ecosystem: Libraries & Integrations
- *Ray & LLMs*
- Ray core module - 1
- Q & A

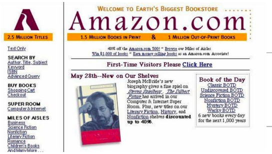
The future of computing is distributed

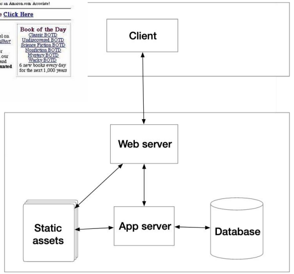
Distributed systems not new…

ARPA Network (1970s)

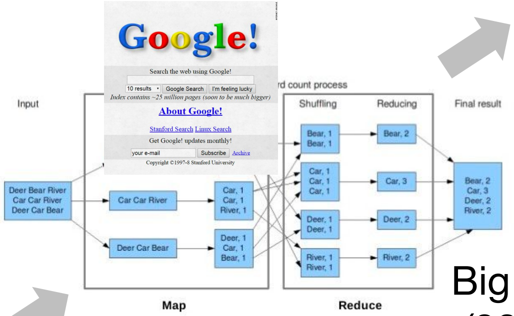


HPC (1980s)



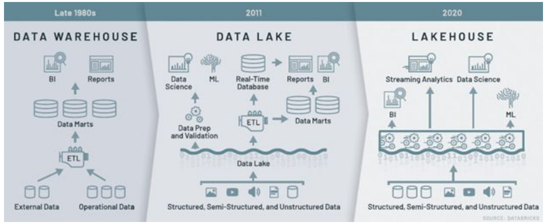


Web (1990s)

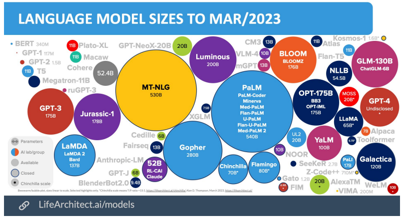


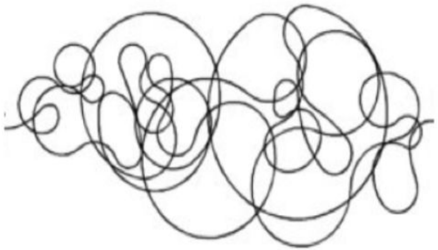
Big Data (2000s)

Big Data (2010s)

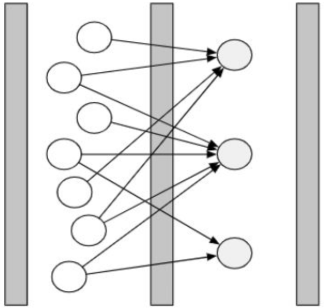


Cluster usage, by generation

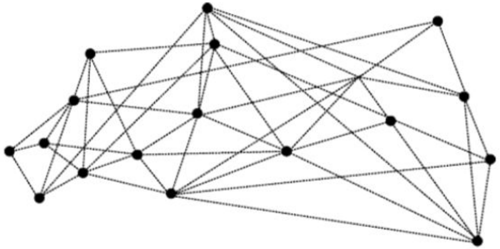




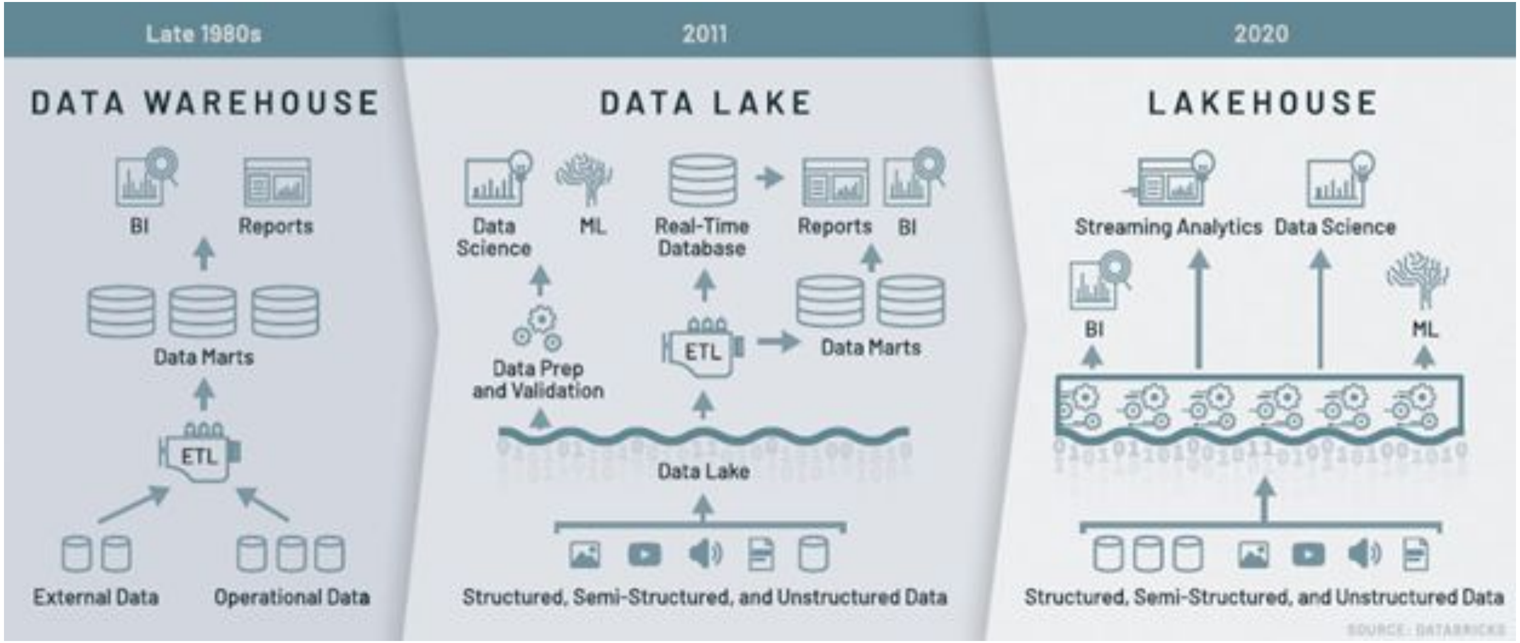
1990s



2000s



current



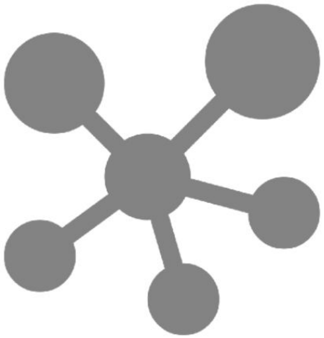
How to scale AI applications using massive data in Lakehouse in the cloud?

The Evolution and trend in cloud computing

Too Big To Fit, scale-up vs. scale-out

When an application becomes too big or too complex to run efficiently on a single server, there are some options:

- migrate to a larger server, and buy bigger licenses – that's called *vertical scale-up*
- distribute data+compute across multiple servers – that's called *horizontal scale-out*



The histories of **MPI**, **Hadoop**, **Spark**, **Dask**, etc., represent generations of scale-out, which imply **trade-offs** both for the risks (losing partitions, split-brain, etc.) as well as the inherent overhead costs

Too Big To Fit, scale-up vs. scale-out

When an application becomes too big or too complex to run efficiently on a single server, there are some options:

- migrate to a
  that's called
- distribute da
  that's called

Cloud Computing has arguably been an embodiment of distributed systems practices for the past 15 years, whether for scale-up or scale-out

The histories of MPI, Hadoop, Spark, Dask, etc., represent generations of scale-out, which imply **trade-offs** both for the risks (losing partitions, split-brain, etc.) as well as the inherent overhead costs

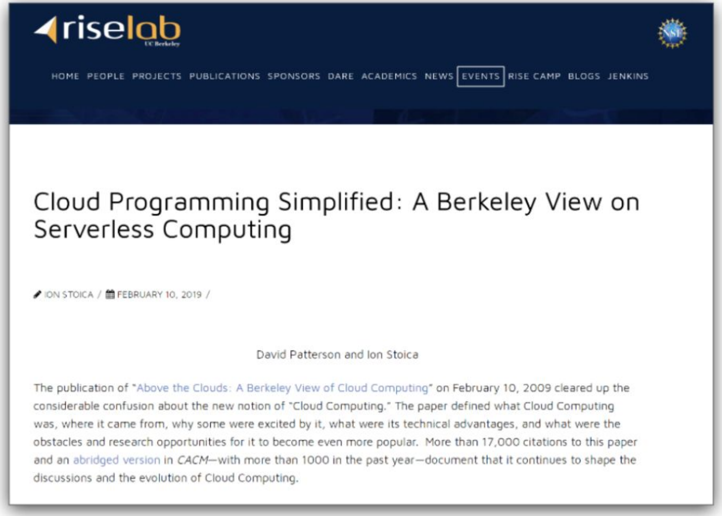
Formal definition of Cloud Computing

Professors **Ion Stoica** and **David Patterson** led EECS grad students to define cloud computing **formally** in 2009

"More than 17,000 citations to this paper…"

2019 follow-up:

"We now predict … that *Serverless Computing* will grow to dominate the future of cloud computing in the next decade"



riselab

HOME PEOPLE PROJECTS PUBLICATIONS SPONSORS DARE ACADEMICS NEWS EVENTS RISE CAMP BLOGS JENKINS

Cloud Programming Simplified: A Berkeley View on Serverless Computing

ION STOICA / FEBRUARY 10, 2019 /

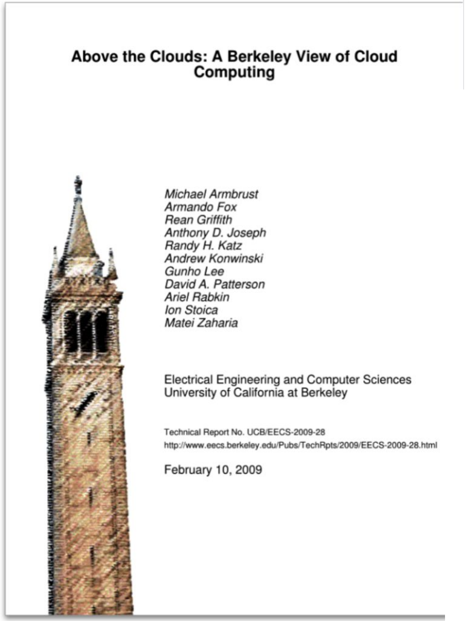
David Patterson and Ion Stoica

The publication of "Above the Clouds: A Berkeley View of Cloud Computing" on February 10, 2009 cleared up the considerable confusion about the new notion of "Cloud Computing." The paper defined what Cloud Computing was, where it came from, why some were excited by it, what were its technical advantages, and what were the obstacles and research opportunities for it to become even more popular. More than 17,000 citations to this paper and an abridged version in *CACM*—with more than 1000 in the past year—document that it continues to shape the discussions and the evolution of Cloud Computing.

Evolution of cloud patterns

Initially, cloud services were simplified to make them more recognizable for IT staff accustomed to VMware

- Patterson, et al., developed industry research methodology and eventually also a **pattern language** to describe distributed systems
- **AMPlab** foresaw how cloud use cases would progress in industry over the next decade, which greatly informed **Apache Spark**, etc.



Above the Clouds: A Berkeley View of Cloud Computing

Michael Armbrust
Armando Fox
Rean Griffith
Anthony D. Joseph
Randy H. Katz
Andrew Konwinski
Gunho Lee
David A. Patterson
Ariel Rabkin
Ion Stoica
Matei Zaharia

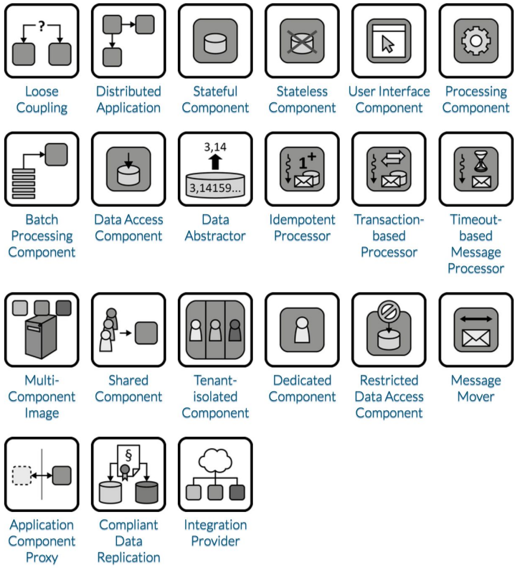
Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2009-28
http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html

February 10, 2009

2009

Cloud patterns & architectures

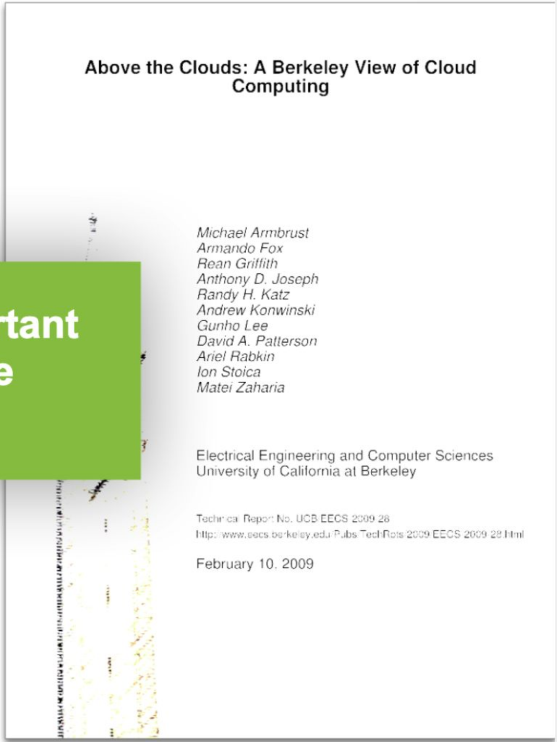


Evolution of cloud patterns

Initially, cloud services were simplified to make them more recognizable for IT staff accustomed to VMware

- Patterson, et al., research method also a **pattern la** distributed syste
- **AMPlab** foresaw how cloud use cases would progress in industry over the next decade, which greatly informed **Apache Spark**, etc.

Apache Spark was an important outcome from this collective area of research



Above the Clouds: A Berkeley View of Cloud Computing

Michael Armbrust
Armando Fox
Rean Griffith
Anthony D. Joseph
Randy H. Katz
Andrew Konwinski
Gunho Lee
David A. Patterson
Ariel Rabkin
Ion Stoica
Matei Zaharia

Electrical Engineering and Computer Sciences
University of California at Berkeley

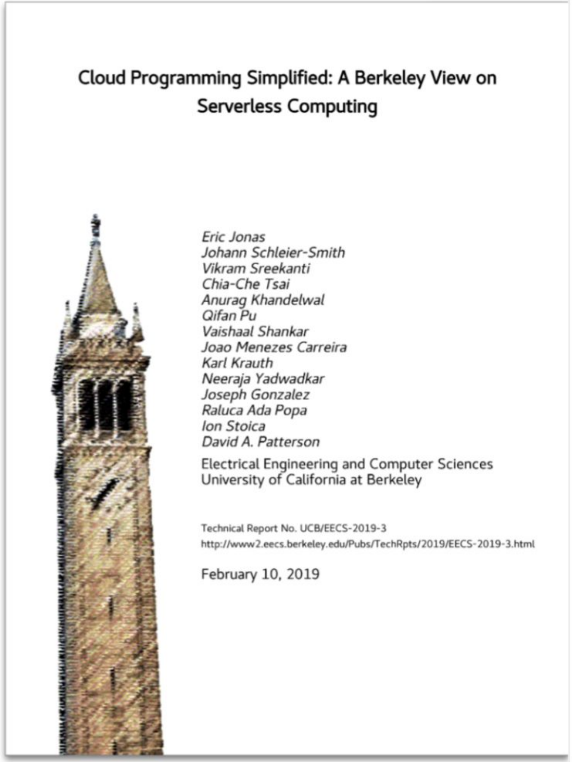
Technical Report No. UCB/EECS-2009-28
http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html

February 10, 2009

Evolution of cloud patterns

Eric Jonas noted >50% of **RISElab** grad students had never used Spark; also, how cloud was evolving in its second decade:

- “Decoupling of computation and storage; they scale separately and are priced independently”
- “The abstraction of executing a piece of code instead of allocating resources on which to execute that code”
- “Paying for the code execution instead of paying for resources you have allocated toward executing the code”



Cloud Programming Simplified: A Berkeley View on Serverless Computing

Eric Jonas
Johann Schleier-Smith
Vikram Sreekanti
Chia-Che Tsai
Anurag Khandelwal
Qifan Pu
Vaishaal Shankar
Joao Menezes Carreira
Karl Krauth
Neeraja Yadwadkar
Joseph Gonzalez
Raluca Ada Popa
Ion Stoica
David A. Patterson

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2019-3
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2019/EECS-2019-3.html

February 10, 2019

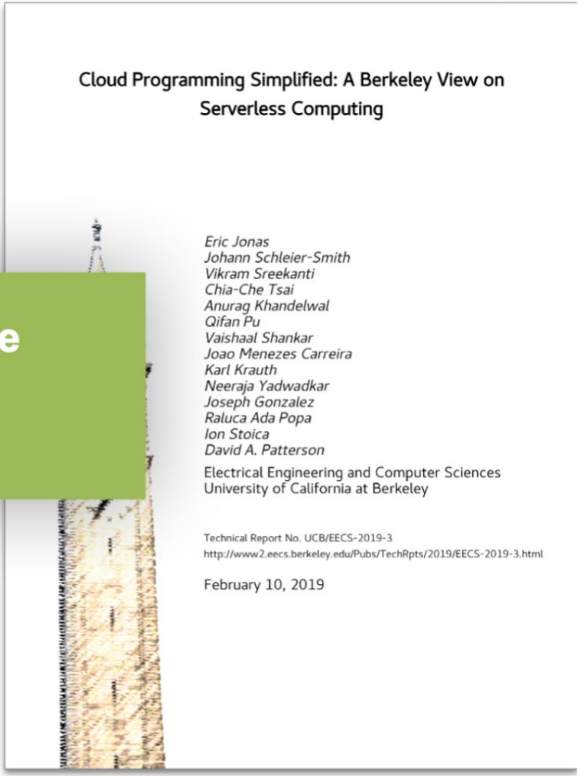
2019

Evolution of cloud patterns

Eric Jonas noted >50% of RISElab grad students had never used Spark, plus how cloud was evolving in its second decade:

- "Decoupling of c… they scale sepa… independently"
- "The abstraction… code instead of allocating resources on which to execute that code"
- "Paying for the code execution instead of paying for resources you have allocated toward executing the code"

Ray is an important outcome from this collective area of research


Cloud Programming Simplified: A Berkeley View on Serverless Computing

Eric Jonas
Johann Schleier-Smith
Vikram Sreekanti
Chia-Che Tsai
Anurag Khandelwal
Qifan Pu
Vaishaal Shankar
Joao Menezes Carreira
Karl Krauth
Neeraja Yadwadkar
Joseph Gonzalez
Raluca Ada Popa
Ion Stoica
David A. Patterson

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2019-3
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2019/EECS-2019-3.html

February 10, 2019

2019

Why and what's Ray?

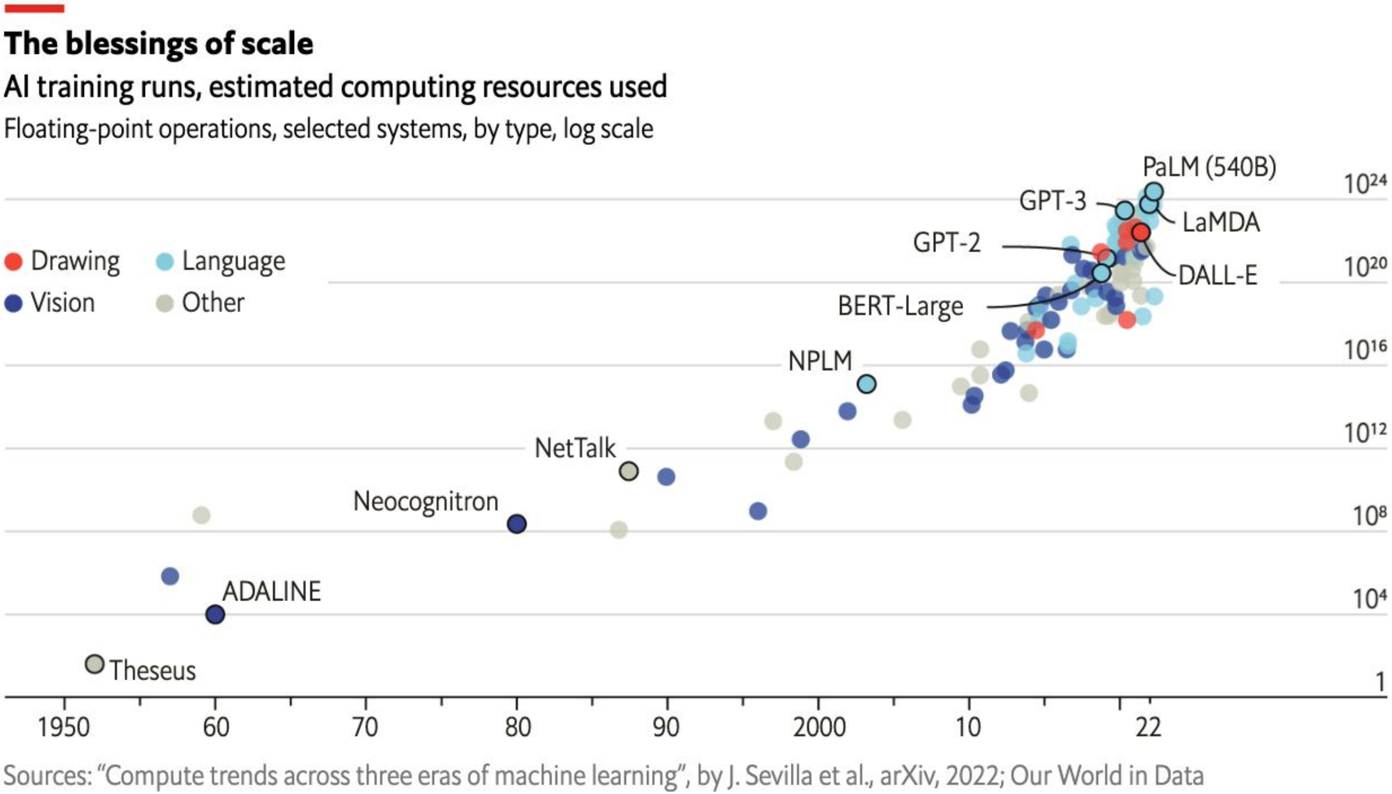
Why Ray

Machine learning is pervasive

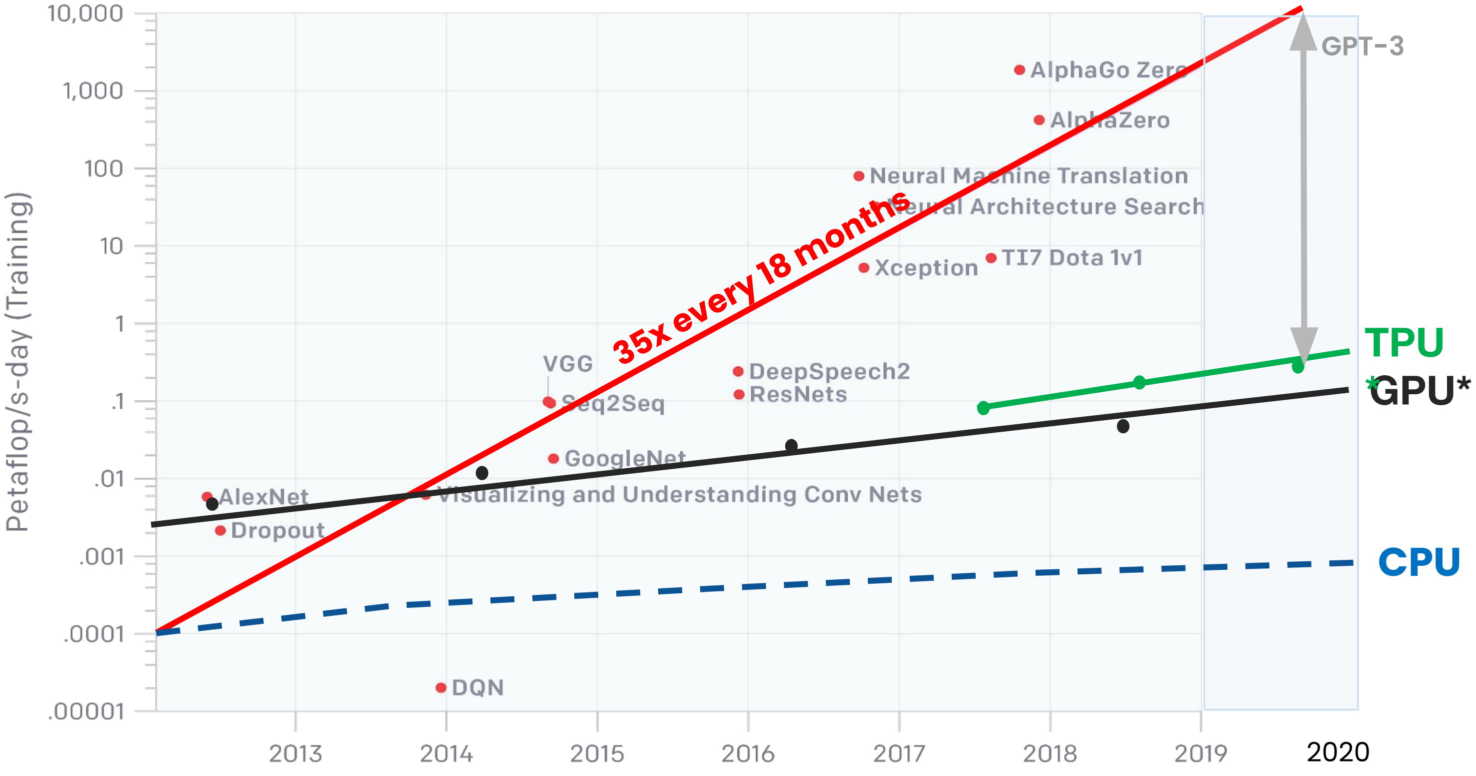
Distributed computing is a necessity

Python is the default language for DS/ML

Blessings of scale …

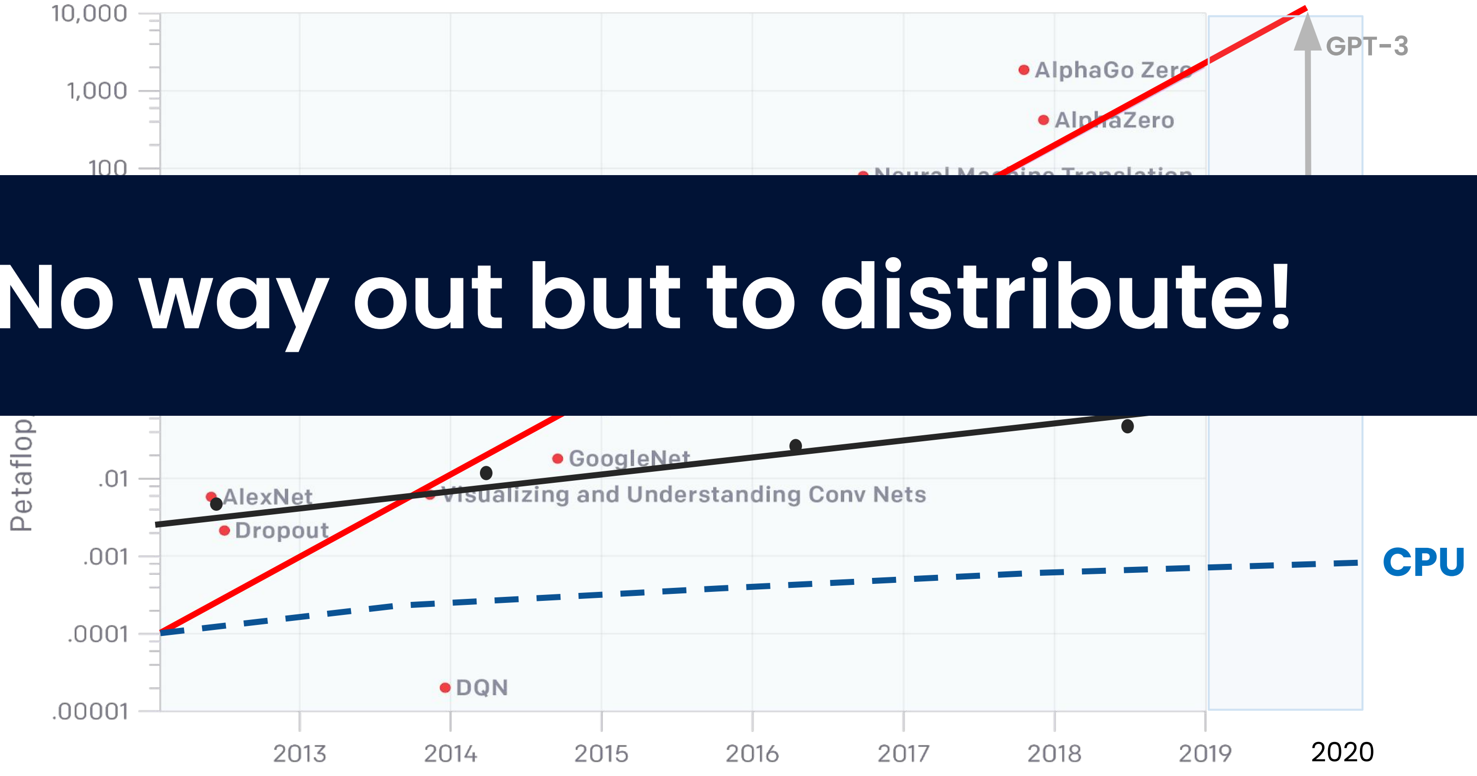


Compute - supply demand problem



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

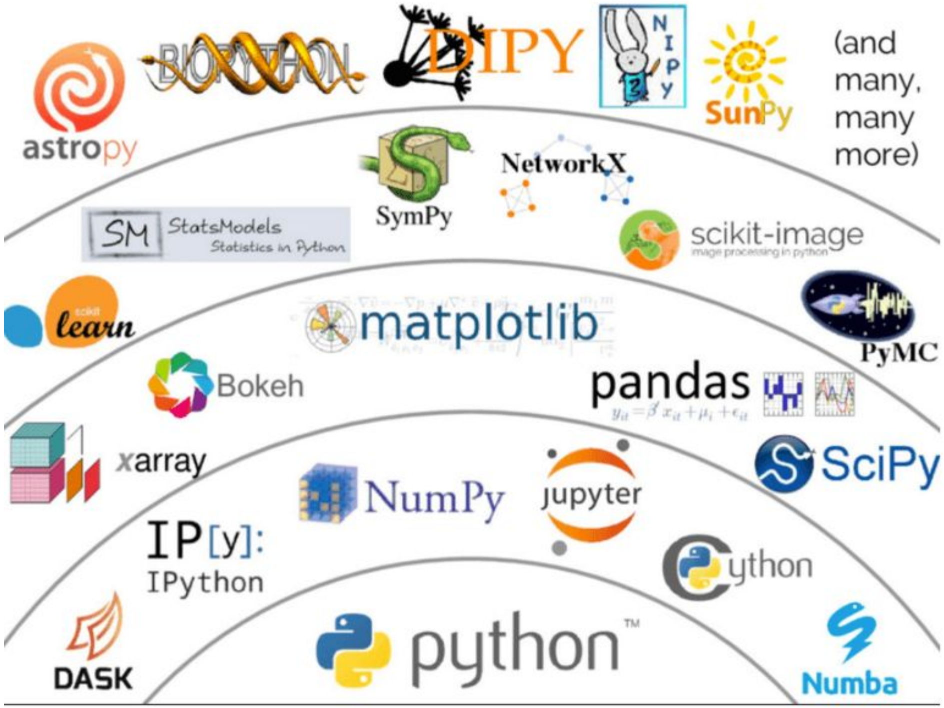
Specialized hardware is not enough



No way out but to distribute!

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

Python data science ecosystem

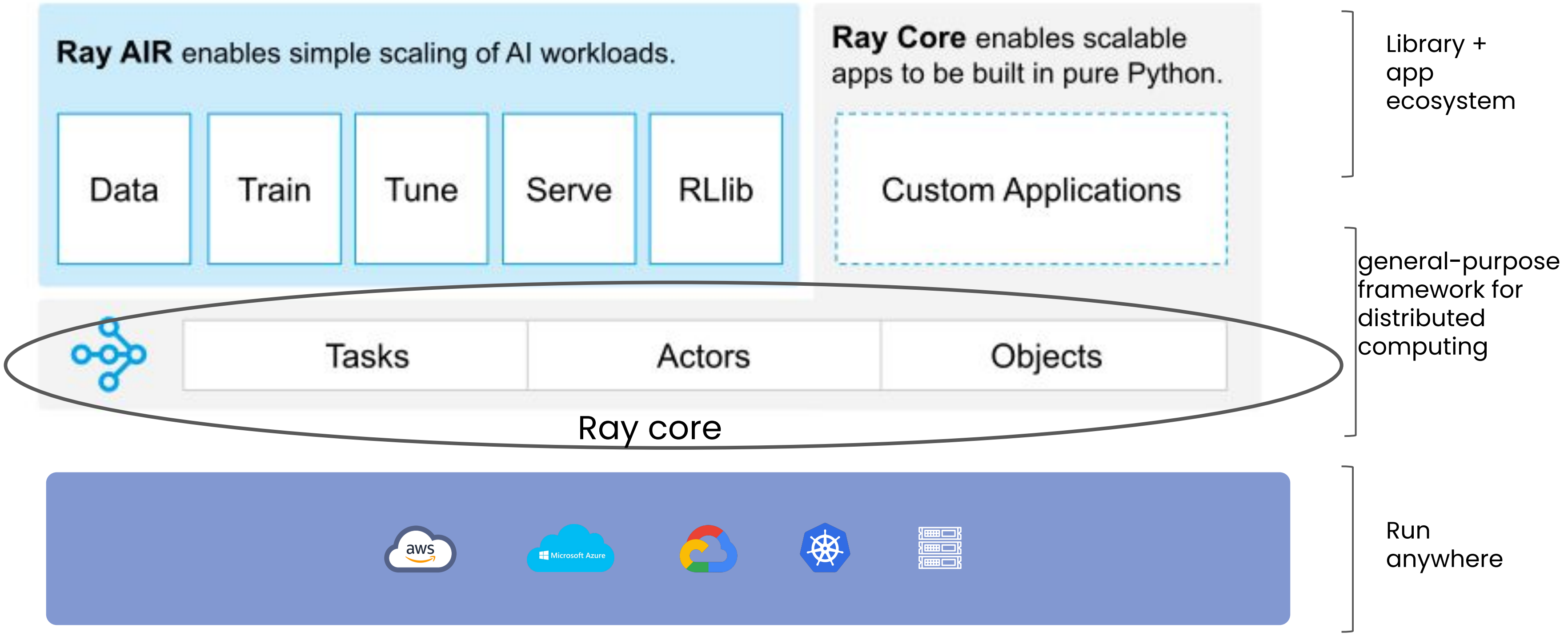


What is Ray

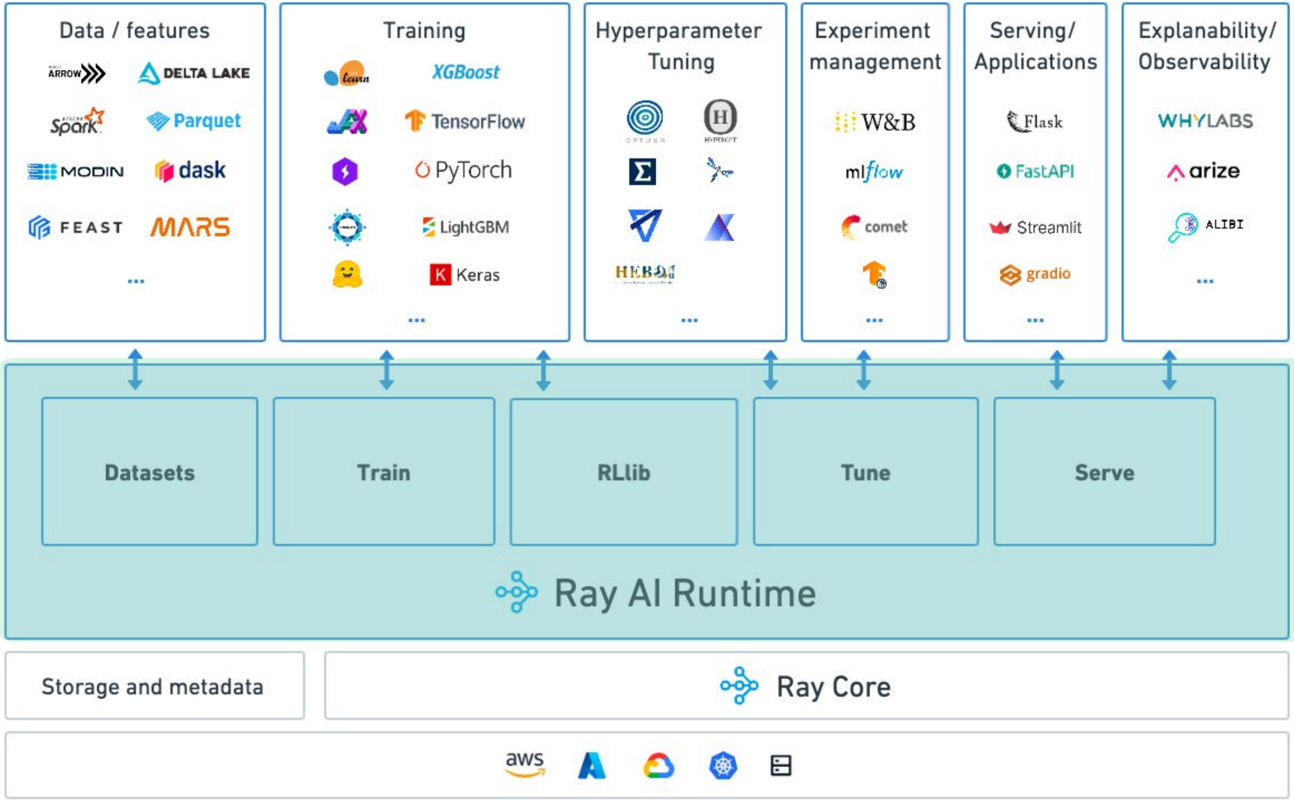
- A ***simple/general-purpose*** library for distributed computing
- An ecosystem of Python libraries (for scaling ML and more)
- Runs on laptop, public cloud, K8s, on-premise

A layered cake of functionality and capabilities for scaling ML workloads

A Layered Cake and Ecosystem

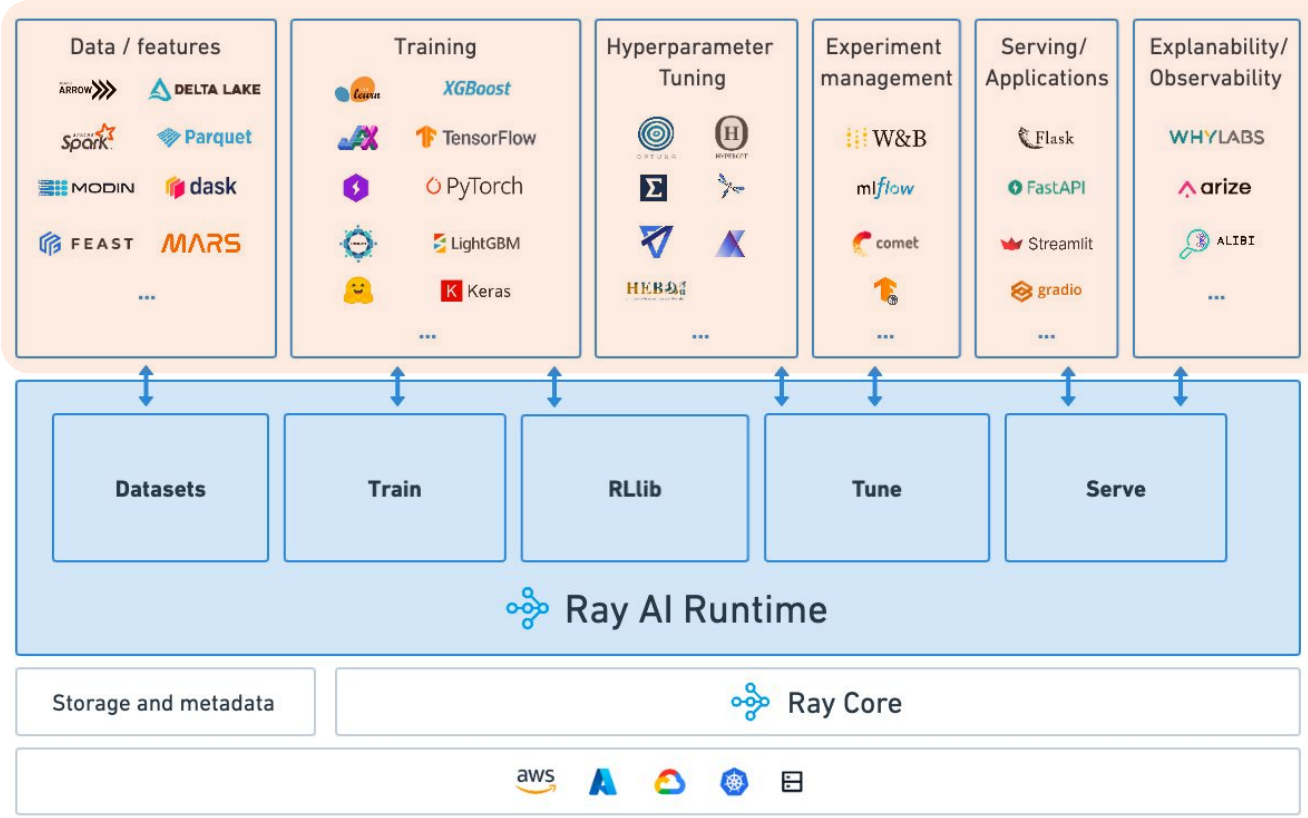


Ray AI Runtime (AIR) is a scalable runtime for end-to-end ML applications



High-level libraries that make scaling easy for both data scientists and ML engineers.

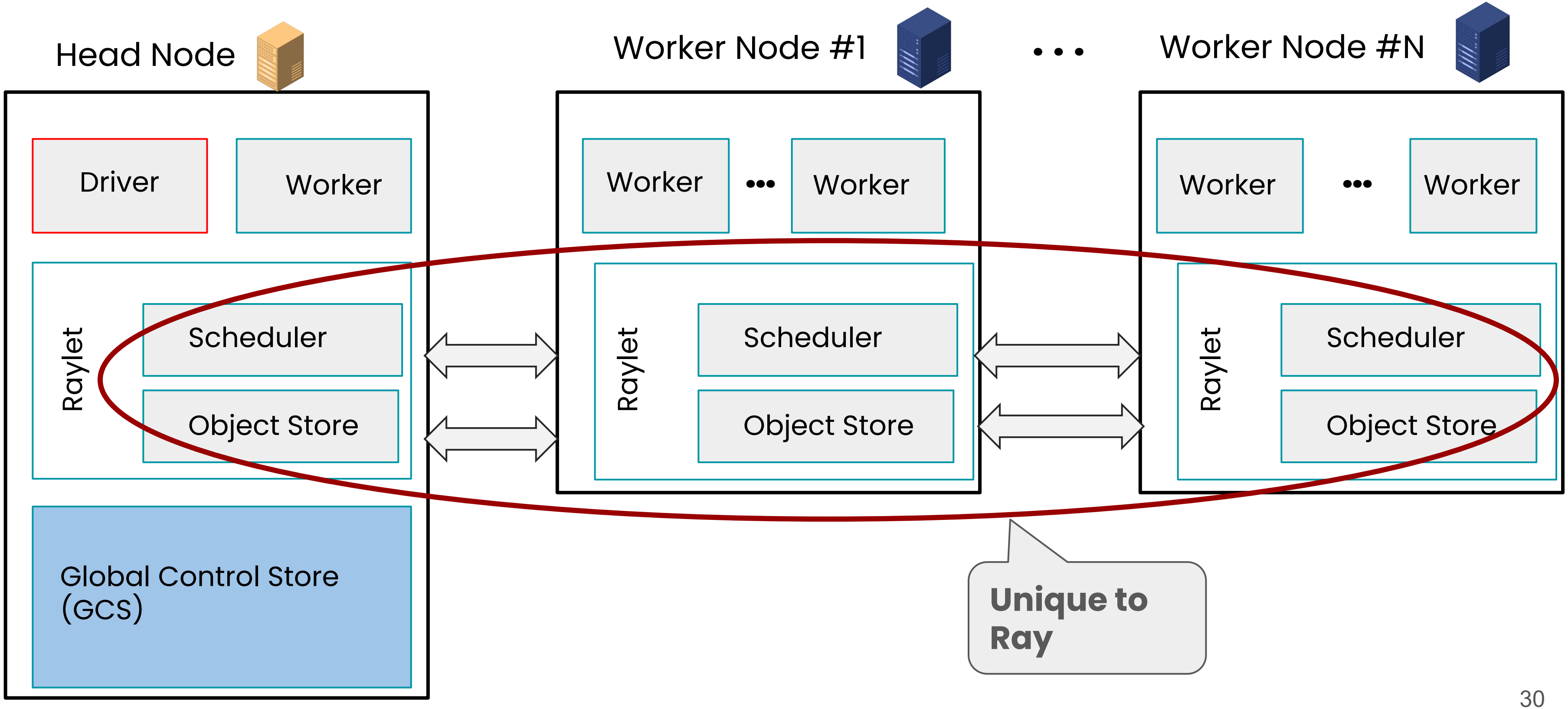
Ray AI Runtime (AIR) is a scalable toolkit for end-to-end ML applications



Scalable integrations with best-of-breed libraries/MLOps tools

Ray Architecture & Components

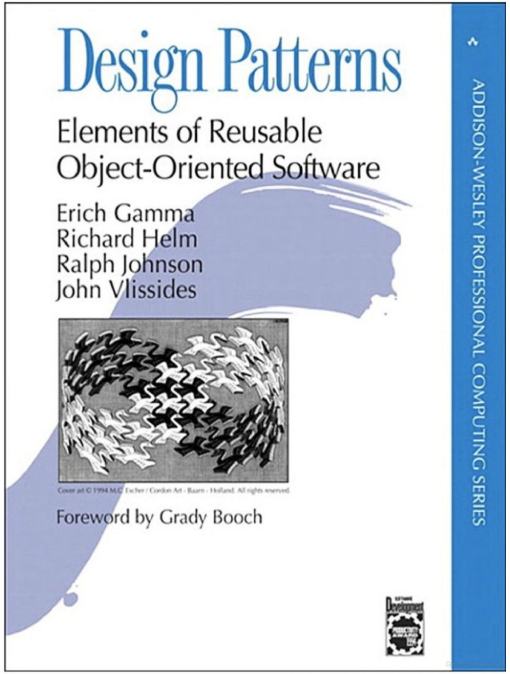
An anatomy of a Ray cluster



Ray distributed design & scaling patterns & APIs

Ray Basic Design Patterns

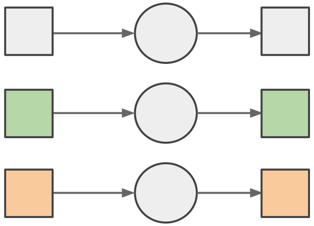
- Ray Parallel Tasks
    - Functions as stateless units of execution
    - Functions distributed across the cluster as tasks
- Ray Objects as Futures
    - Distributed (immutable objects) store in the cluster
    - Fetched when materialized
    - Enable massive asynchronous parallelism
- Ray Actors
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



1. Patterns for Parallel Programming
2. Ray Design Patterns
3. Ray Distributed Library Integration Patterns

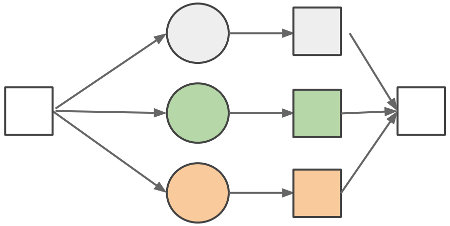
Scaling Design Patterns

Batch Training / Inference



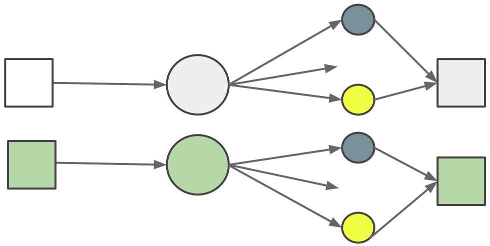
Different data / Same function

AutoML

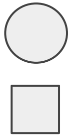


Same data / Different function

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()
```

Distributed

f() Node ••• f() Node

```
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

Cls Node ••• Cls Node

```
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
```

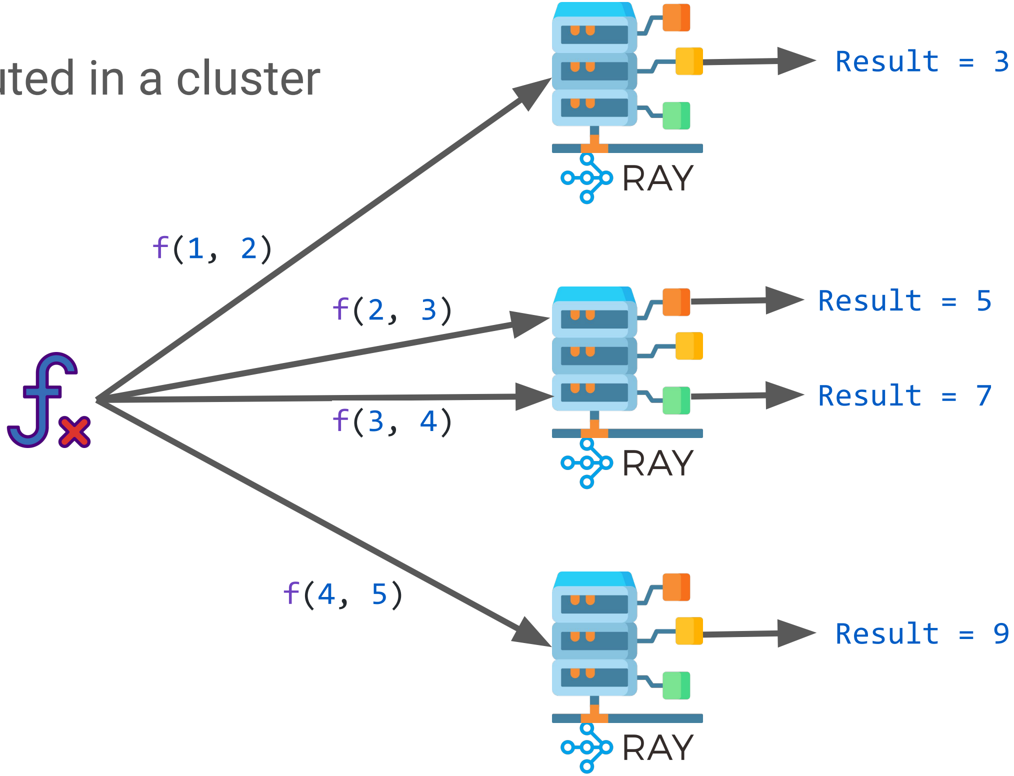
Distributed

a Node ••• a Node

Ray Task

A function remotely executed in a cluster

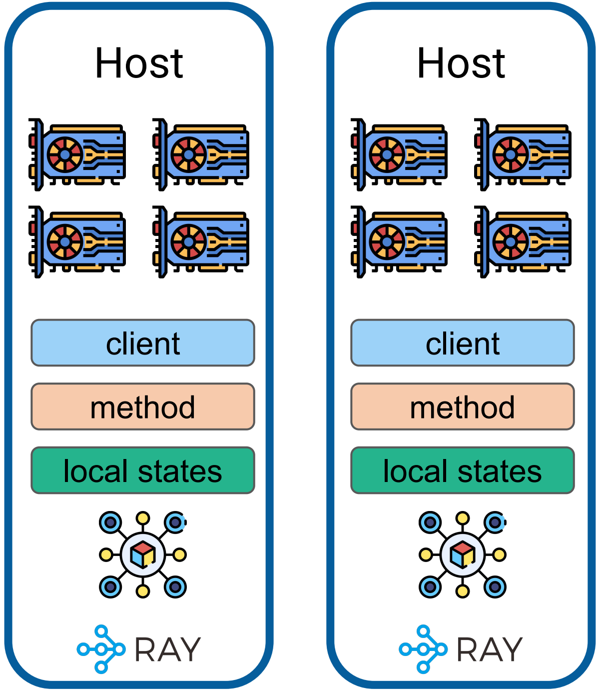
```python
@ray.remote(num_cpus=2)
def f(a, b):
    return a + b


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



Ray Actor

A class remotely executed in a cluster

```python
@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

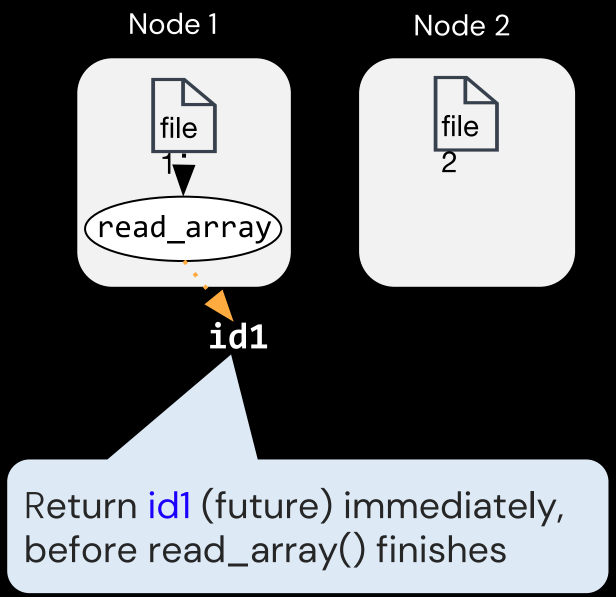
Class → Actor

```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)
```

```python
@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()
```

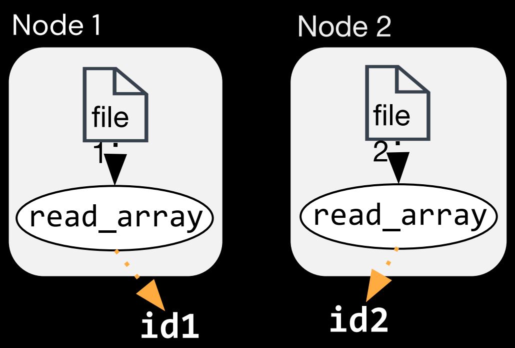
Task API

```
@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)
```

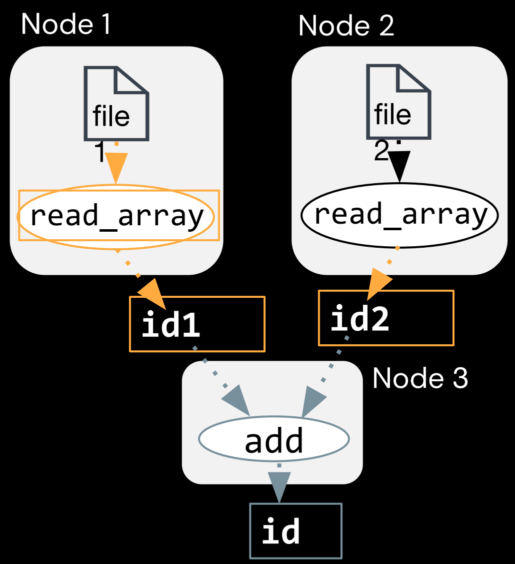


Dynamic task graph: build at runtime

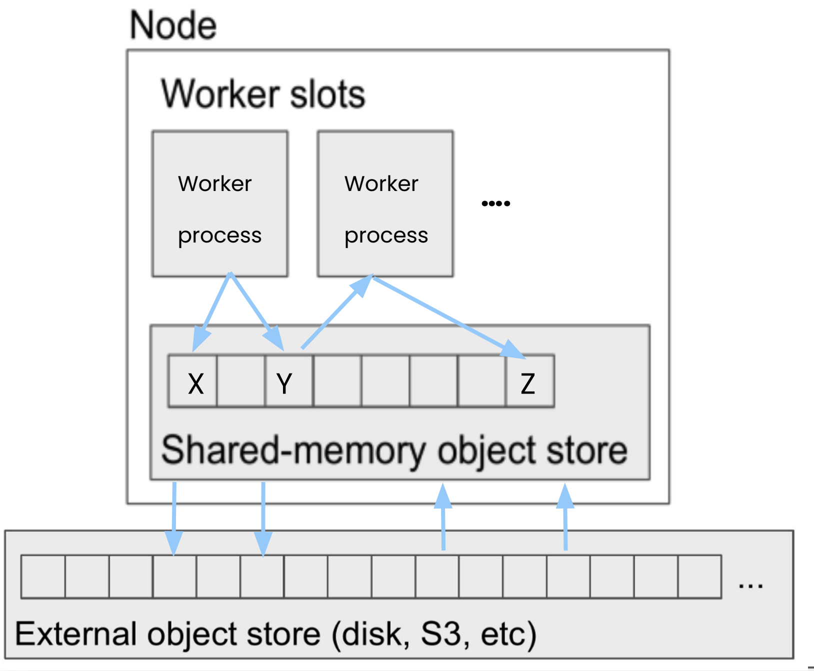
Task API

```
@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

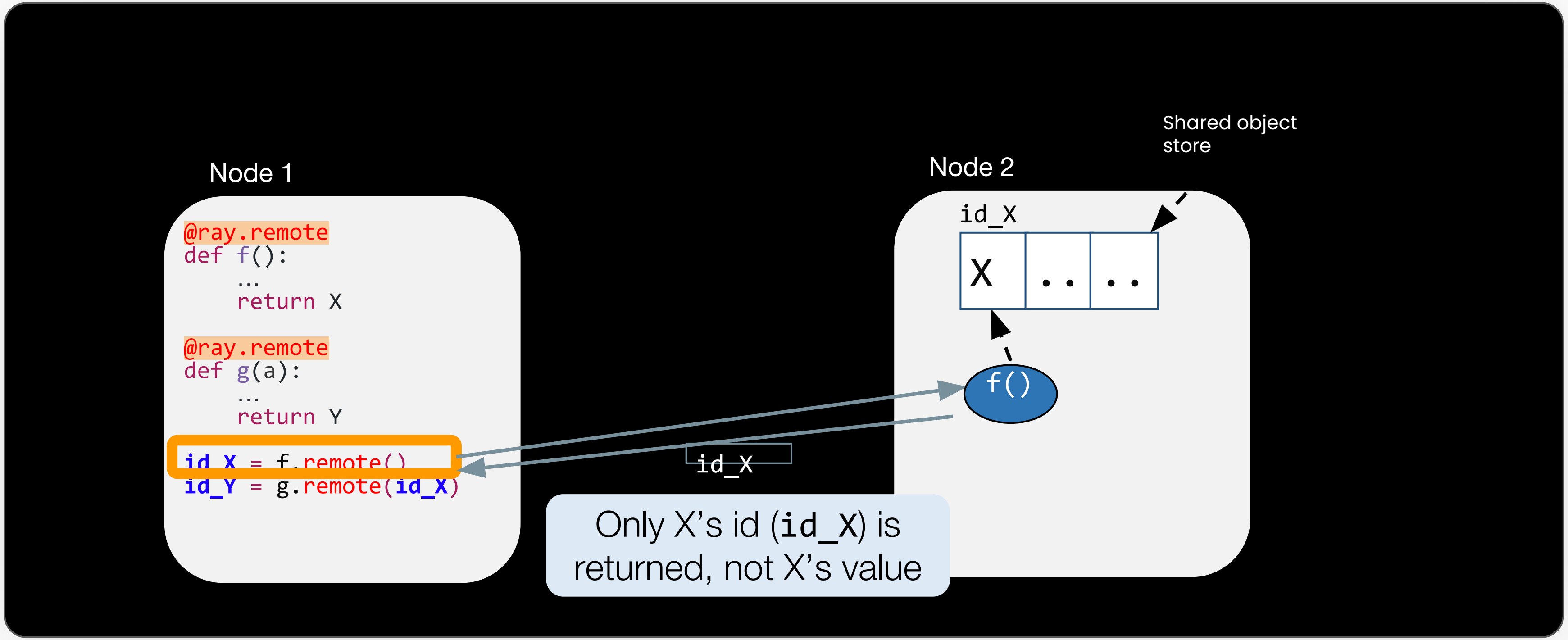


Distributed Immutable object store

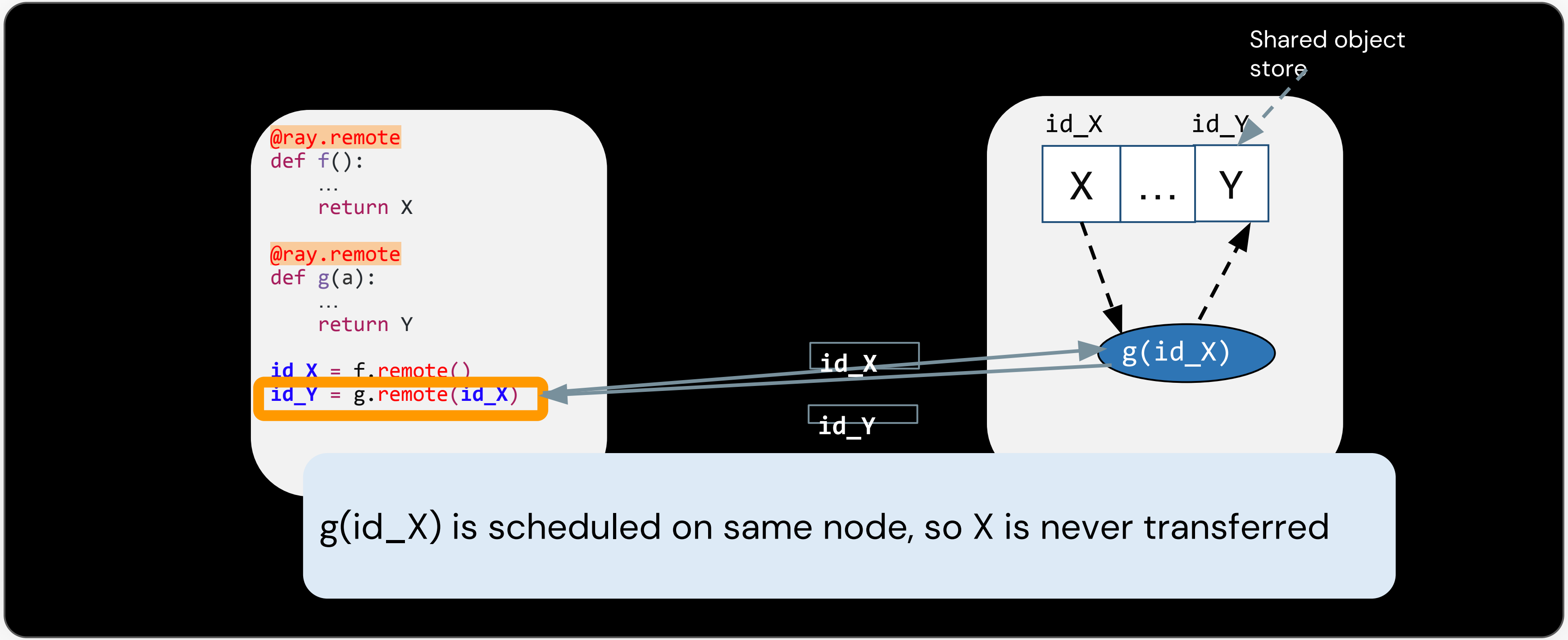


Spill over to external storage

Distributed object store



Distributed object store



Examples of Distributed Applications with Ray

Distributed Applications with Ray

ML Libraries

- Ray AI Runtime
- 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

- DoorDash ML platform
- AirFlow, PrefectePredibase AI
- Uber, Lyft
- Instacart
- Spotify

All using Ray core APIs & patterns

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

Ray: Fastest Growing Scalable Compute Framework

25,000+
GitHub
stars

820+
Community
Contributors

5,000+
Repositories
Depend on Ray

1,000+
Organizations
Using Ray

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

OpenAI

ANTHROP\C

Hugging Face

stability.ai

Options for poor mortals …

1 **Download the models**

HuggingFace

2 **Fine-tune the models**

???

3 **Serve these models**

???

anyscale

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 scale	• 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

anyscale

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?

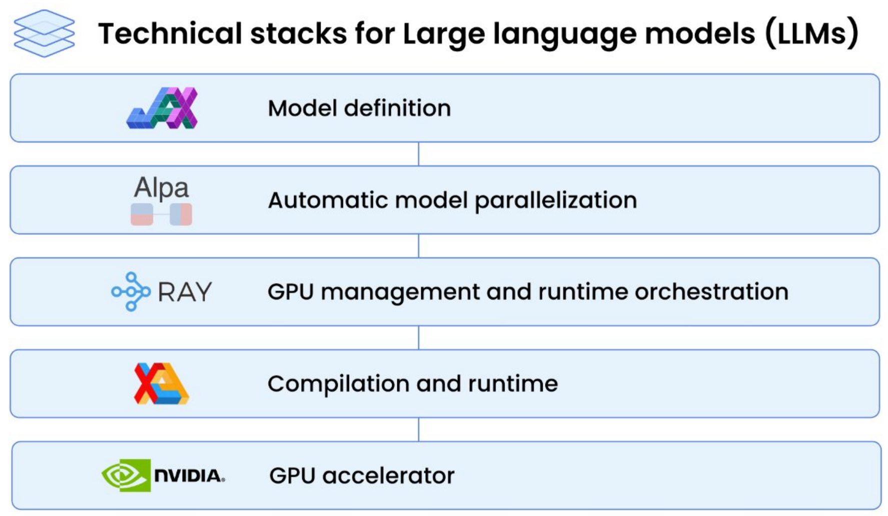


Figure 5: Technical integration layered stack for LLM

- HuggingFace for models
- DeepSpeed for optimized Training
- PyTorch for Framework
- Ray for Orchestration
- GPU or other hardware

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, ...ps Power ChatGPT

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

By Frank Sifei Luan, UC Be...

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

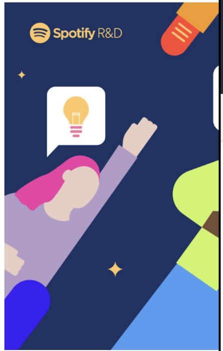
by Isaac Vidas · Data Science & Engineering

Google Cloud Blog

Large Scale Deep Learning Training and Tuning with Ray at Uber

Xu Ning, Di Yu, Michael Mui

Unleashing ML Innovation at Spotify with Ray



aws

Contact Us Support My Acc...

re:Invent Products Solutions Pricing Documentation Le...

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, Kinshuk Pahare, and Derek Liu | on 28 NOV...

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

Autho...

Netflix Technology Blog

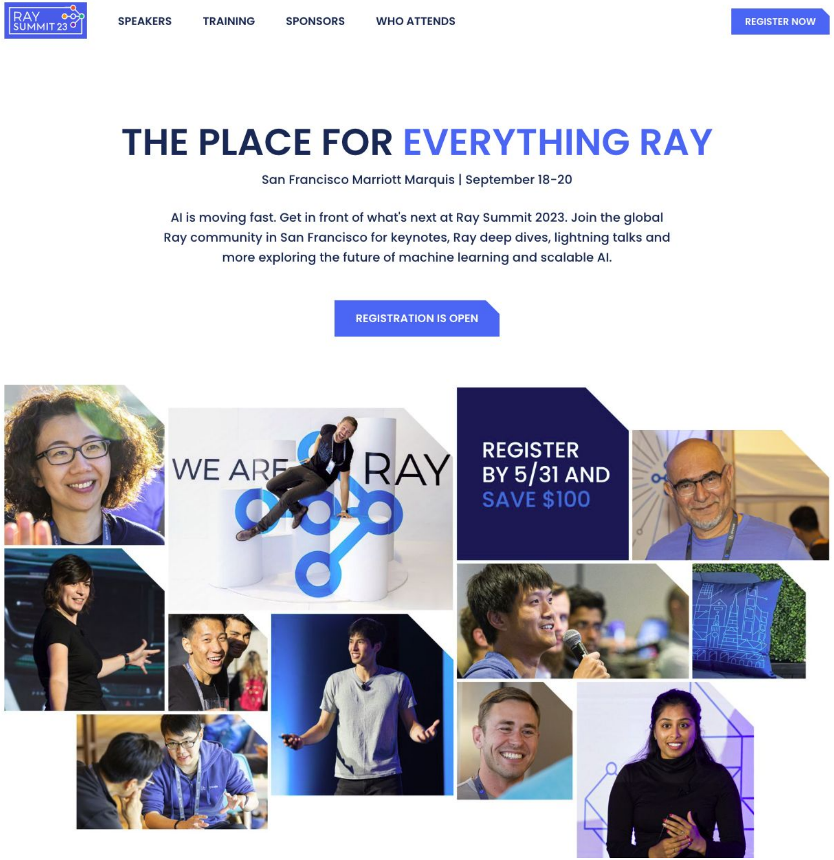
Feb 13 · 11 min read · Listen

Scaling Media Machine Learning at Netflix

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



https://bit.ly/raysummit2023

RAY

Let’s go with

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

Thank you!

Questions?

email: jules@anyscale.com

twitter: @2twitme