DATA+AI SUMMIT 2022

Scaling ML/AI Workloads with Ray Ecosystem

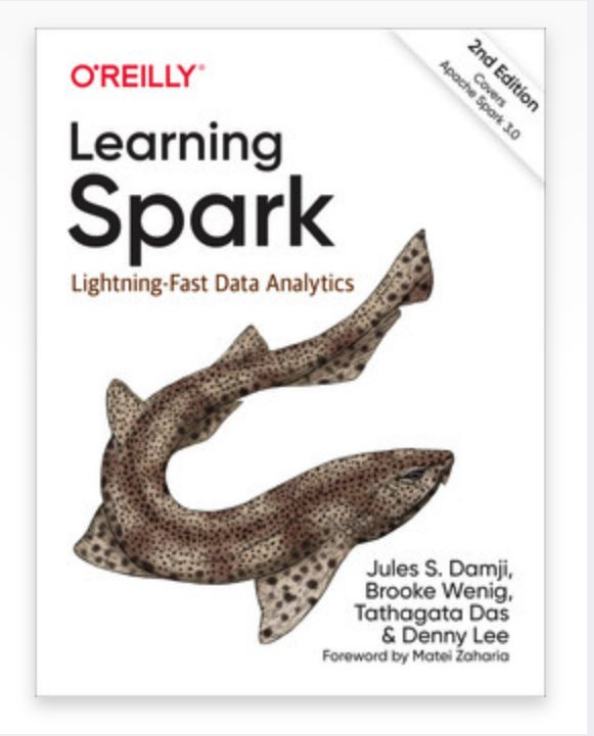
Jules S. Damji @2twitme

Lead Developer Advocate,
Anyscale, Ray Team

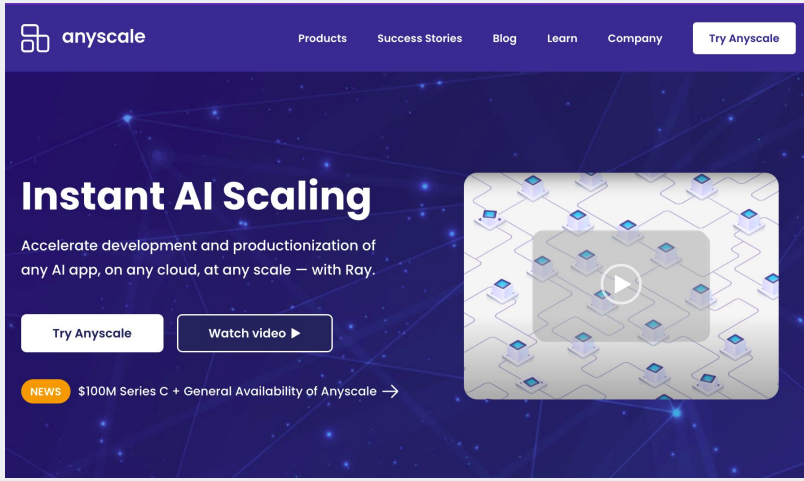
ORGANIZED BY databricks

$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 are we: Original Creators of Ray

What we do: Provide cloud-managed service for Ray

Why do it: Make distributed computing easy and simple

Scale AI/ML workloads

Overview

- Why & What Ray & Ray Ecosystem
- Ray Architecture & Components
- Ray Core APIs
- Ray Native ML Libraries
  - Ray Tune
- Demo
  - Scaling ML workloads (train, tune, inference)
- Q & A

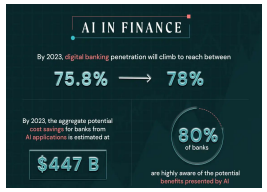
Why Ray 

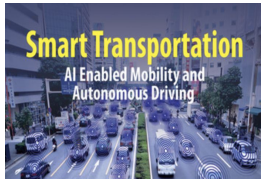
Machine Learning is Pervasive

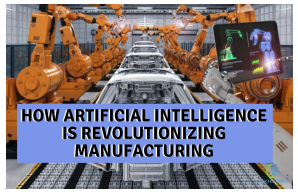
Distributed Computing is a necessity

Python is the language of DS/ML

AI/ML in industries …

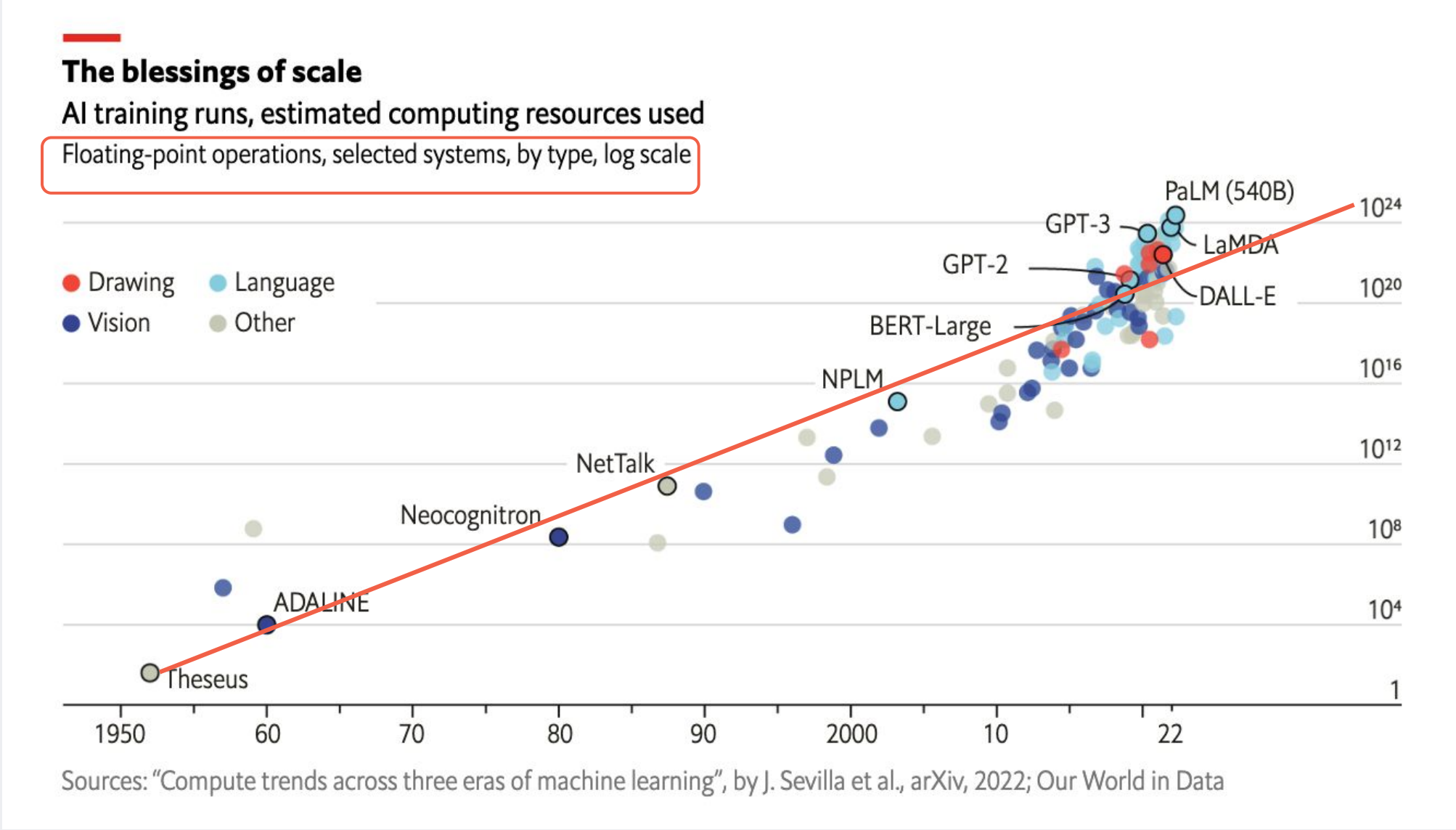




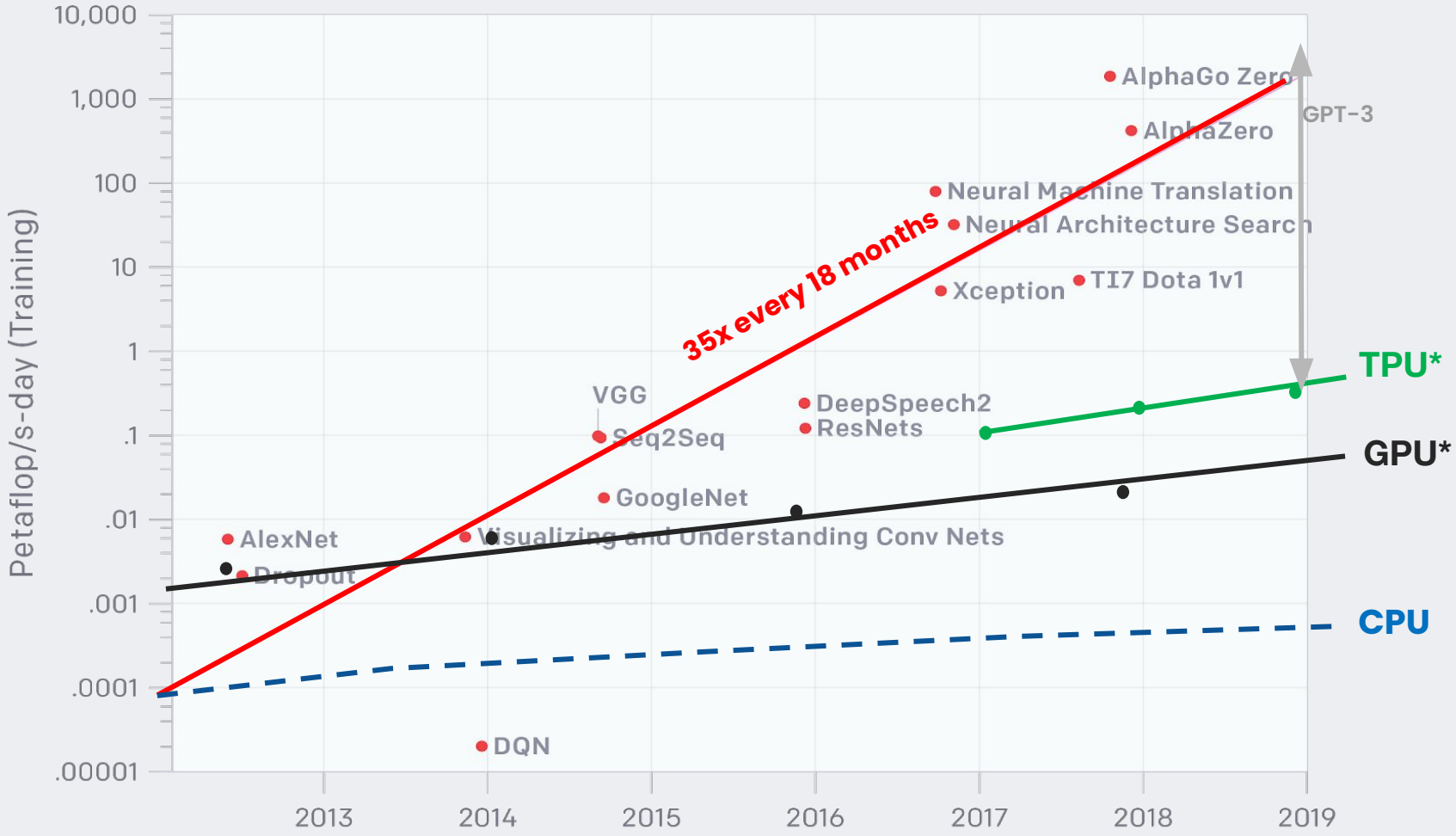


Proliferated & Incorporated across Industries

Blessings of scale ....

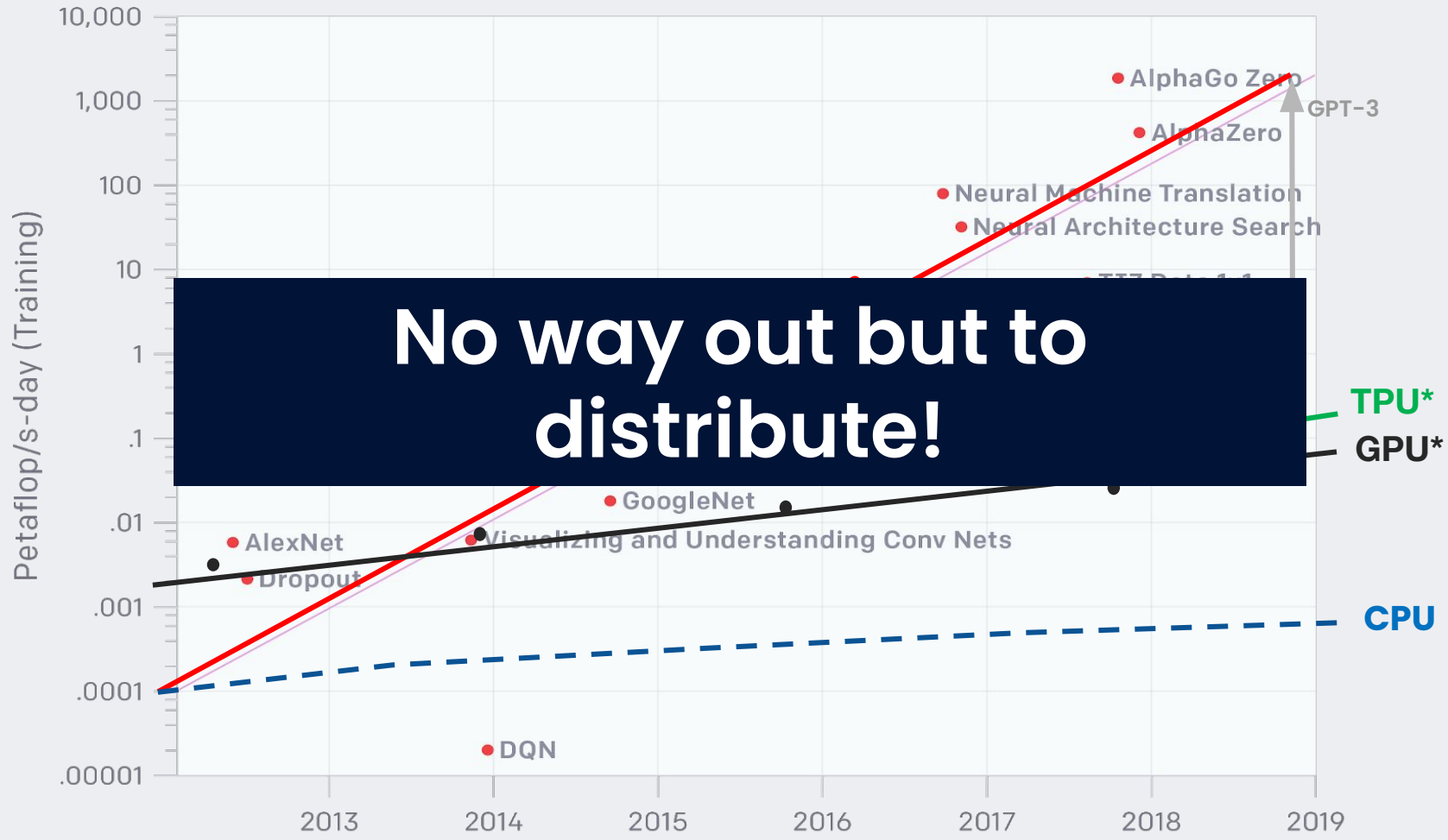


Compute demand – supply problem



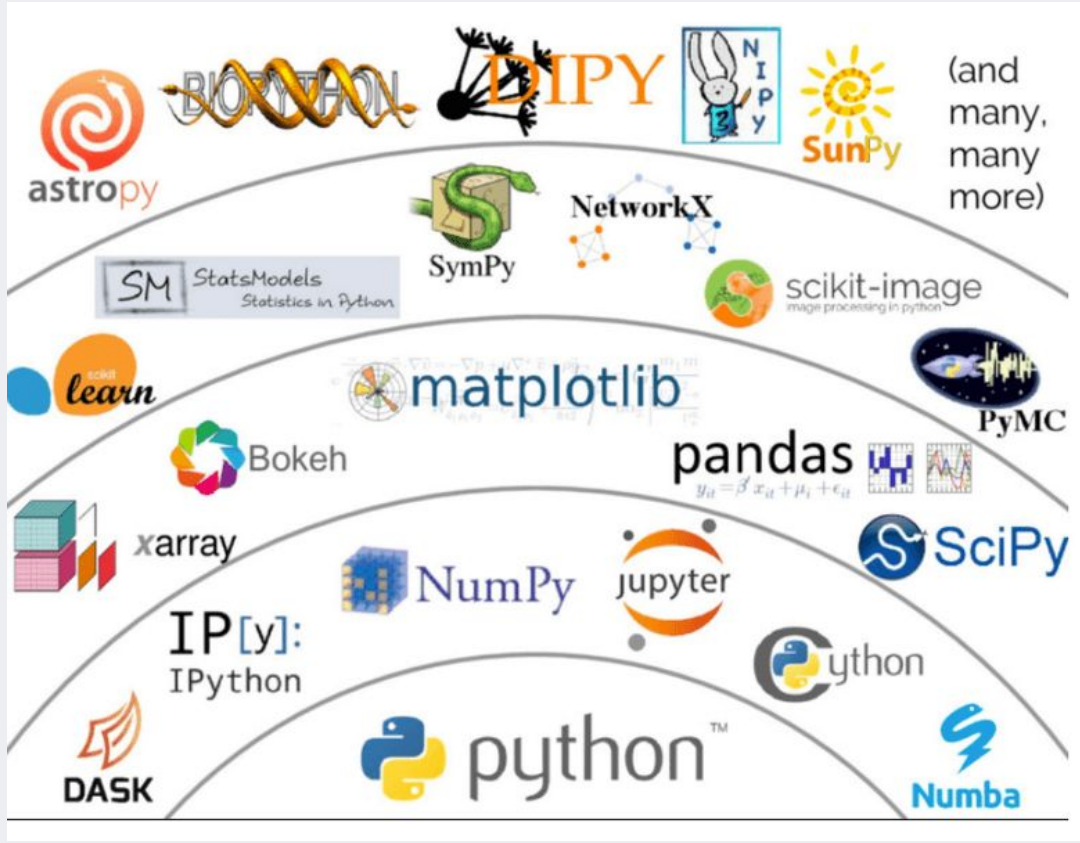
Source: https://openai.com/blog/ai-and-compute/

Specialized hardware is not enough



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

Python data science/ML ecosystem dominating



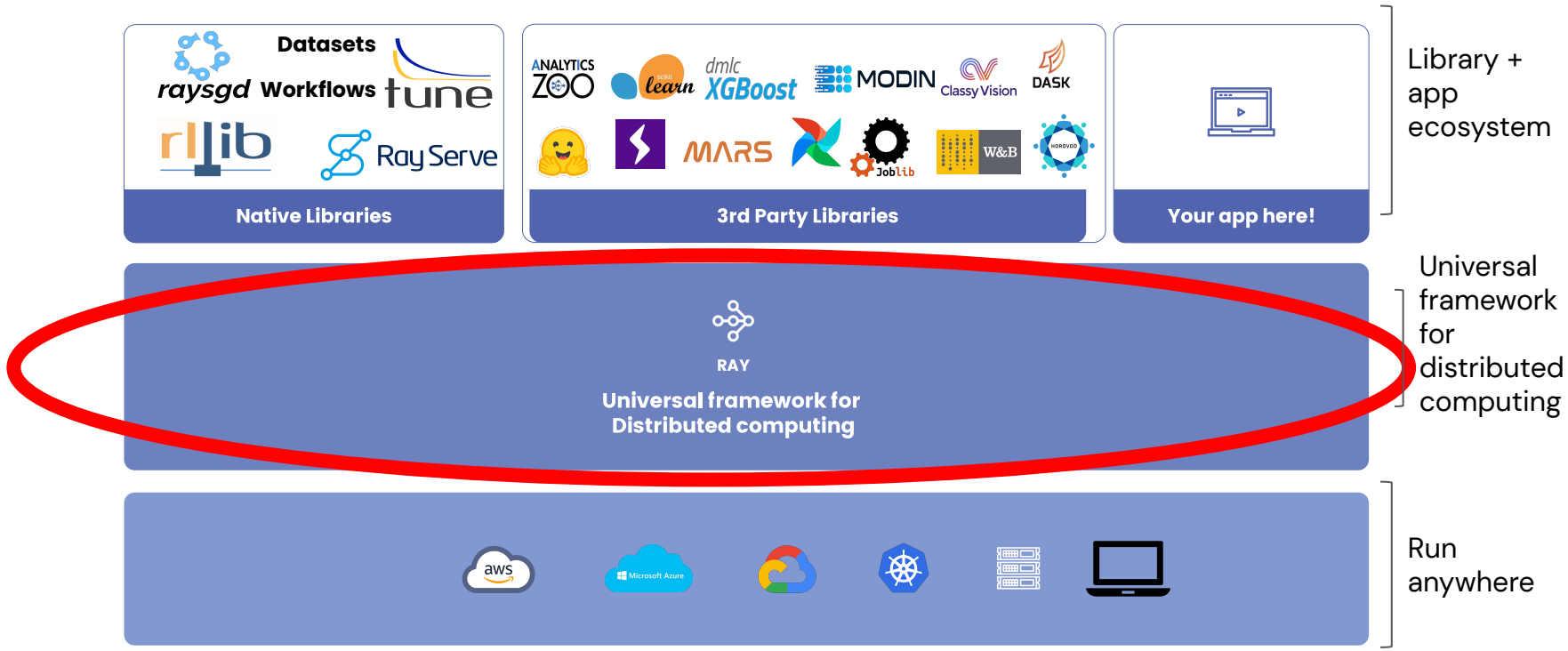
Origins of Ray

What is Ray?

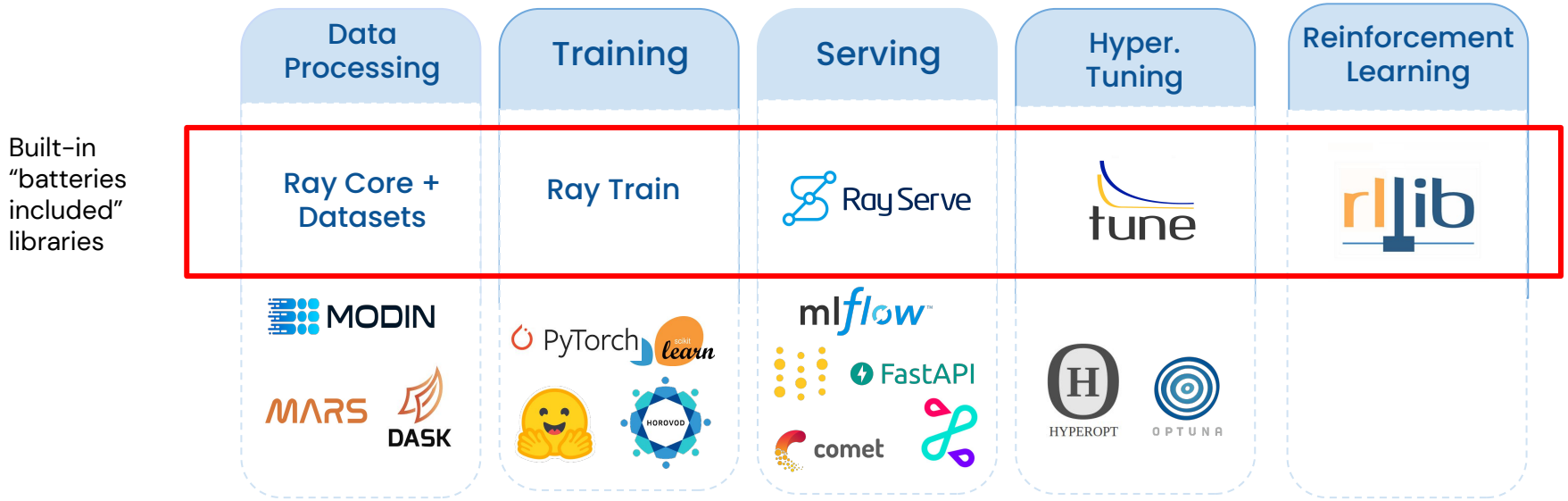
- A ***general-purpose*** library for distributed computing
- An ecosystem of ***native libraries*** to scale ML workloads
- Runs ***anywhere*****:** laptop, public cloud, K8s, on-premise

A layered cake of functionality & capabilities for scaling ML workloads

The Layered Cake and Ecosystem

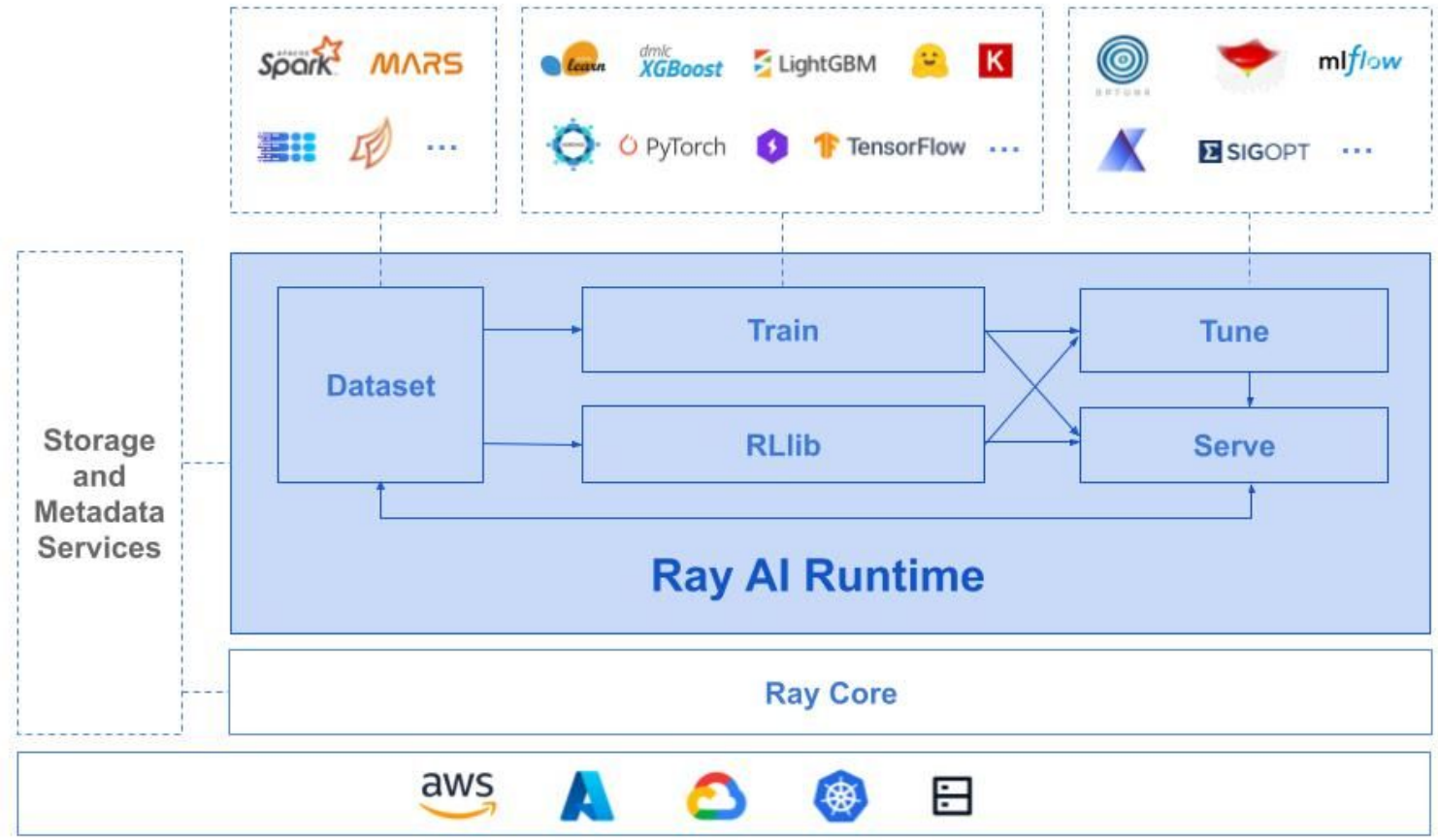


Libraries for scaling ML workloads



Only use the libraries you need!

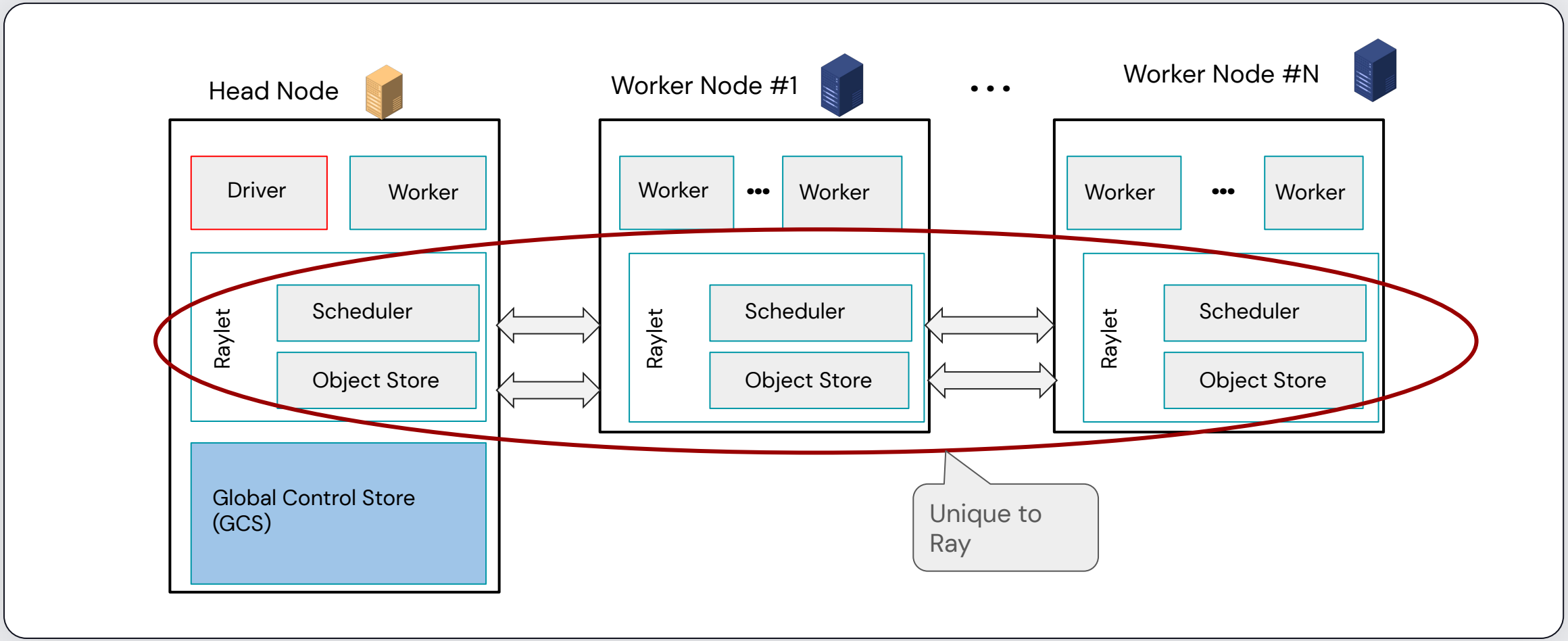
Ray AIR in Ray 2.0



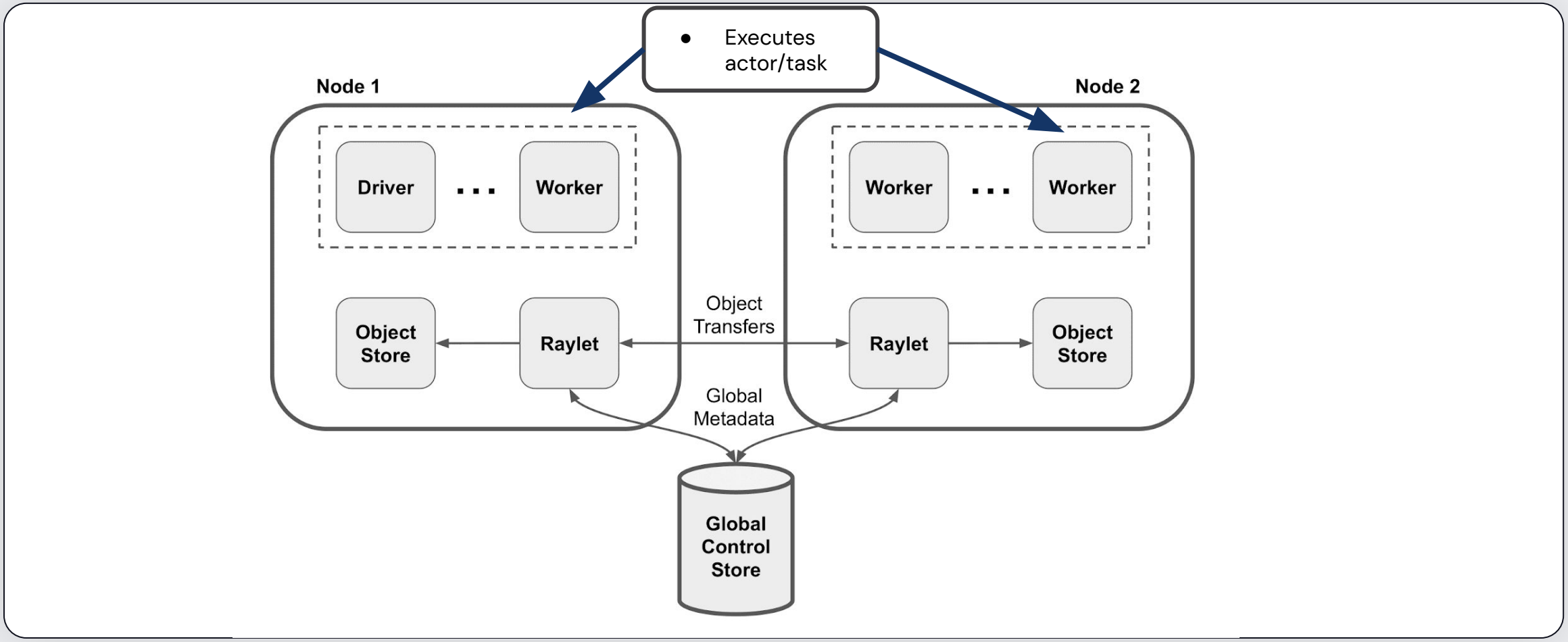
Who Using Ray?

Ray Architecture and Components

Anatomy of a Ray cluster



Anatomy of a Ray cluster



Ray Distributed Design Patterns and APIs

Ray Design Patterns

Ray Parallel Tasks

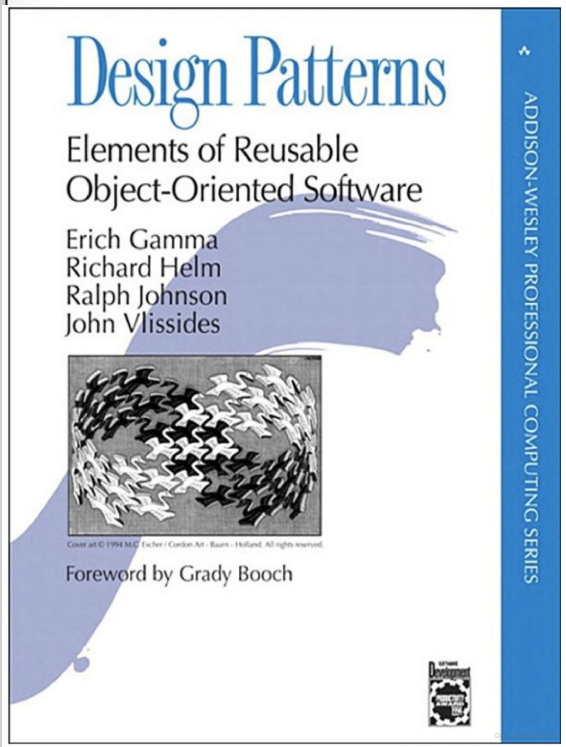
- Functions as stateless units of execution
- Functions distributed across a cluster as task

Ray Objects as Futures

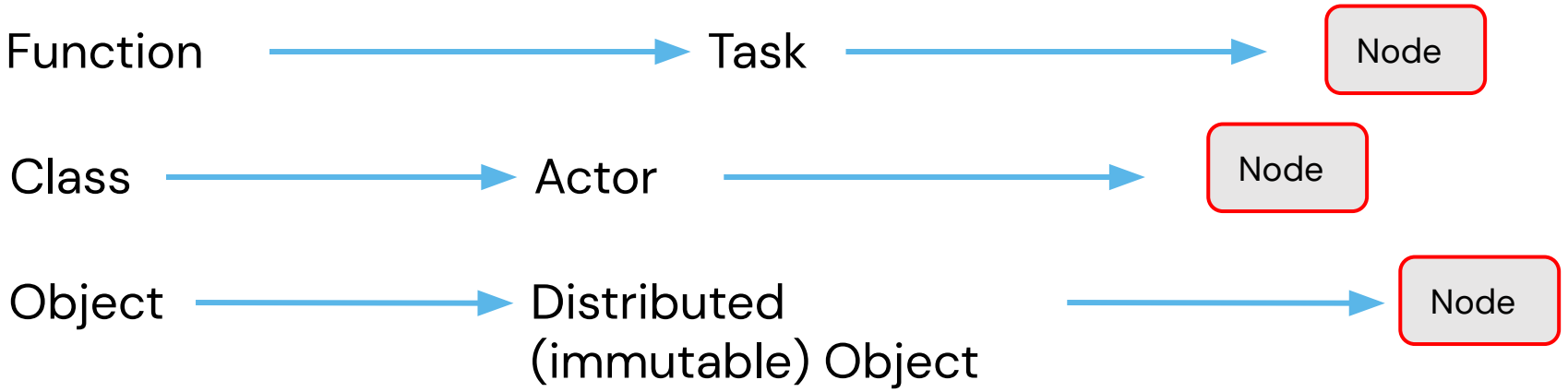
- Distributed immutable objects
- Fetched when available
- Enable asynchronous execution

Ray Actors

- Stateful service on a cluster
- Enables message passing and maintains state



Python → Ray Basic Patterns



Function → Task

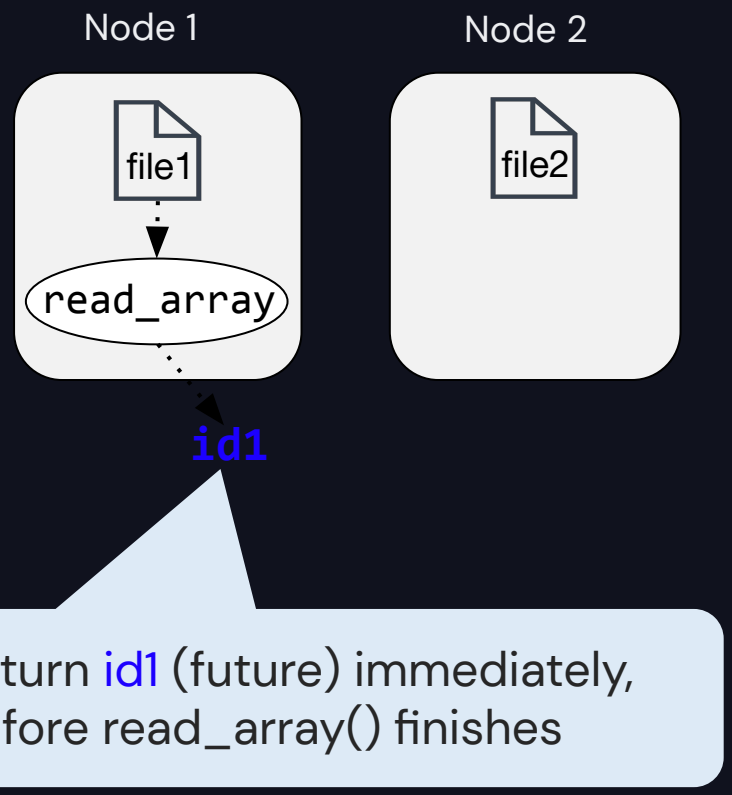
```
@ray.remote
def read_array(file):
    # read ndarray “a”
    # from “file”
    return a

@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()
```

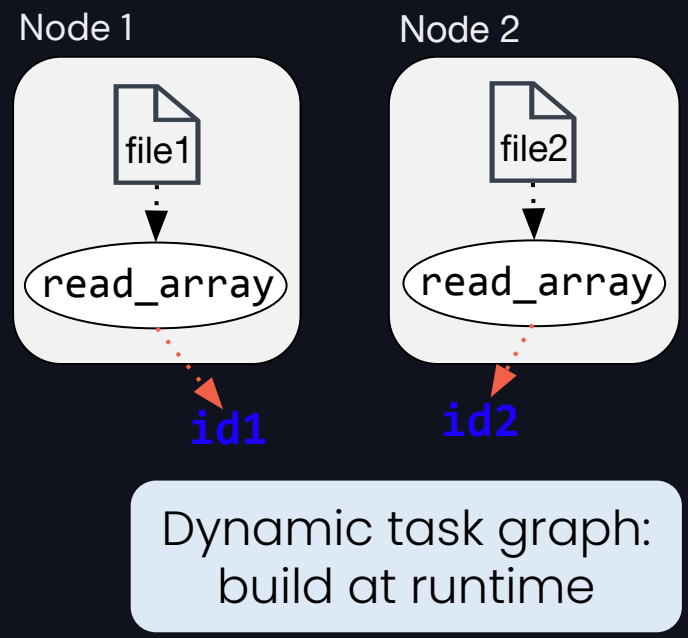
Task API

```
@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

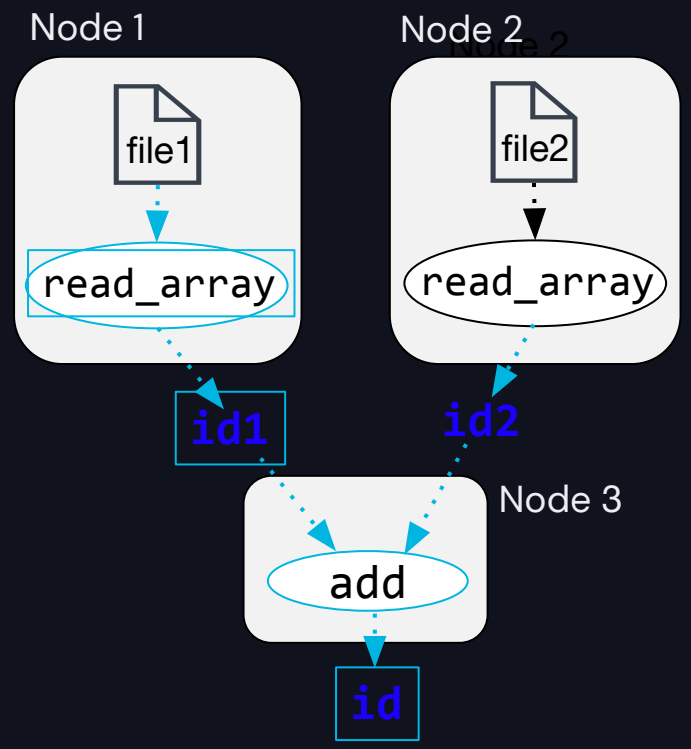
```
@ray.remote
def read_array(file):
    # read ndarray “a”
    # from “file”
    return a

@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)
```



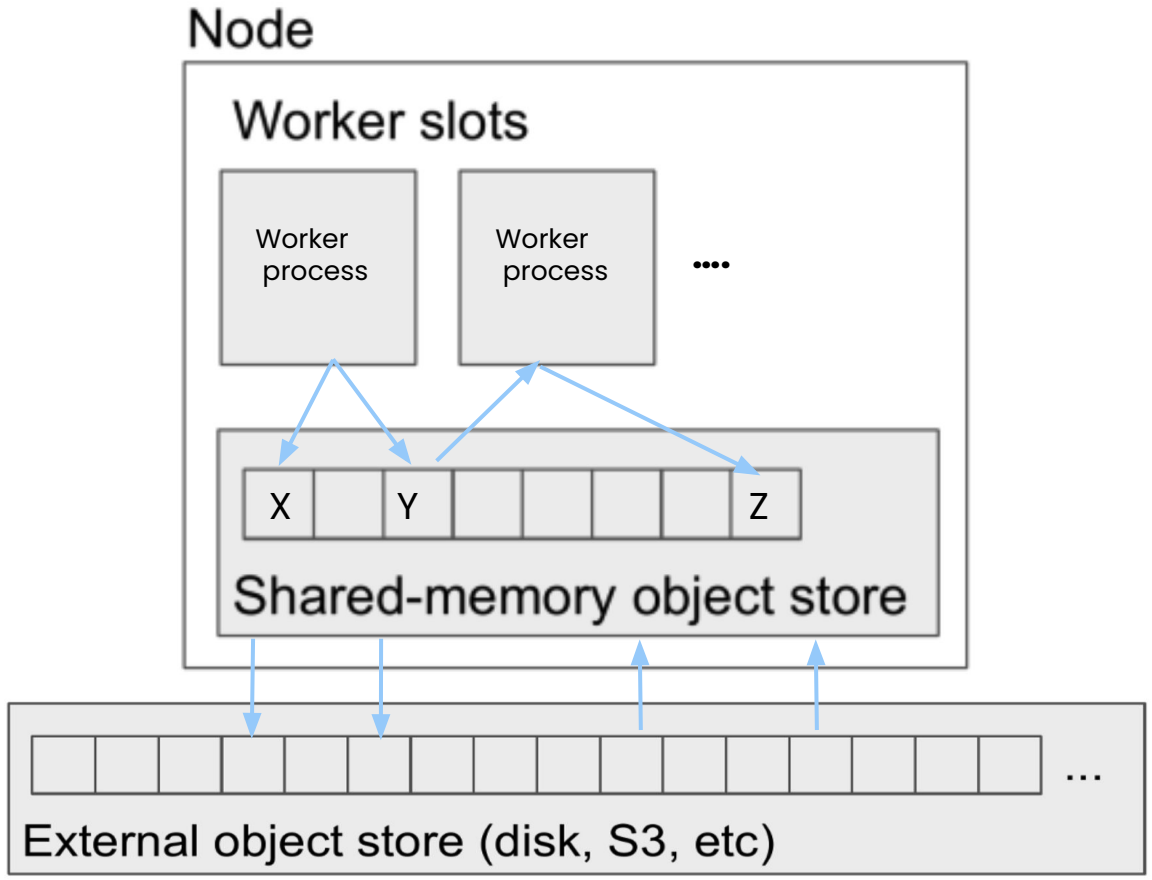
Task API

```
@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

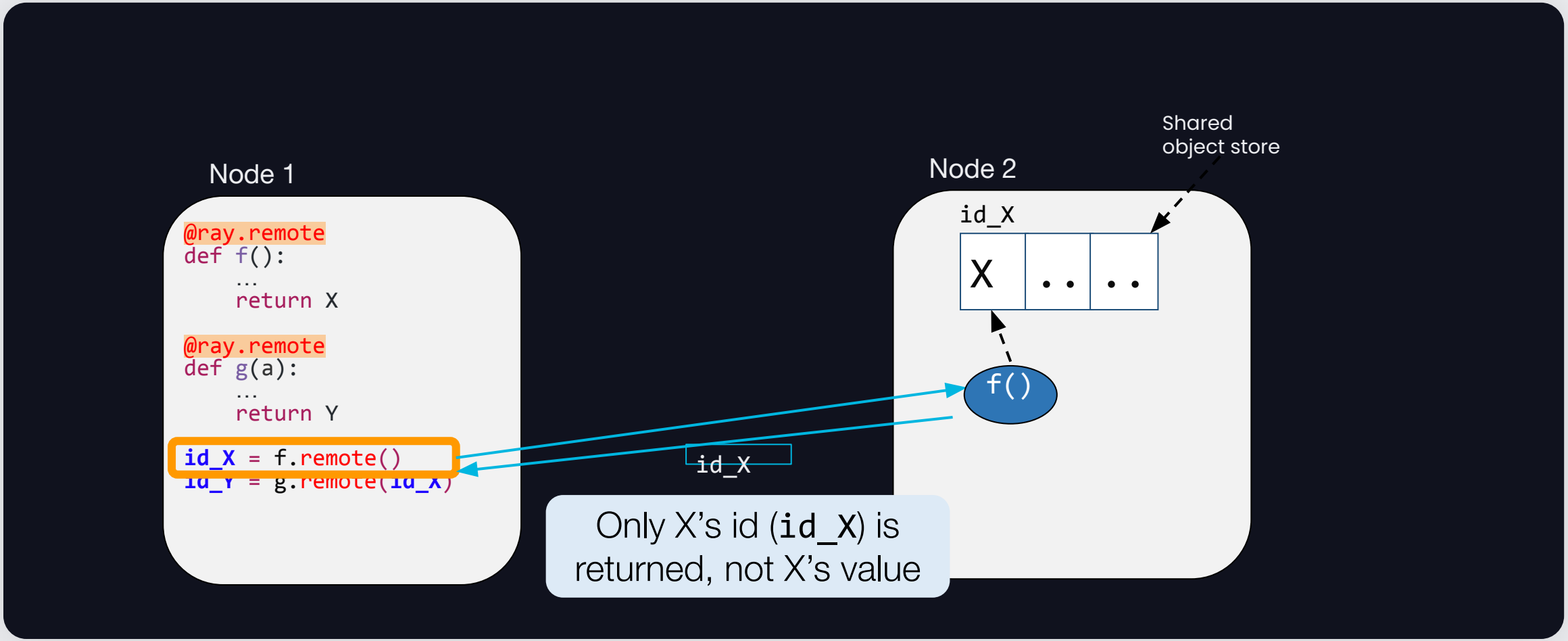


Distributed Immutable object store

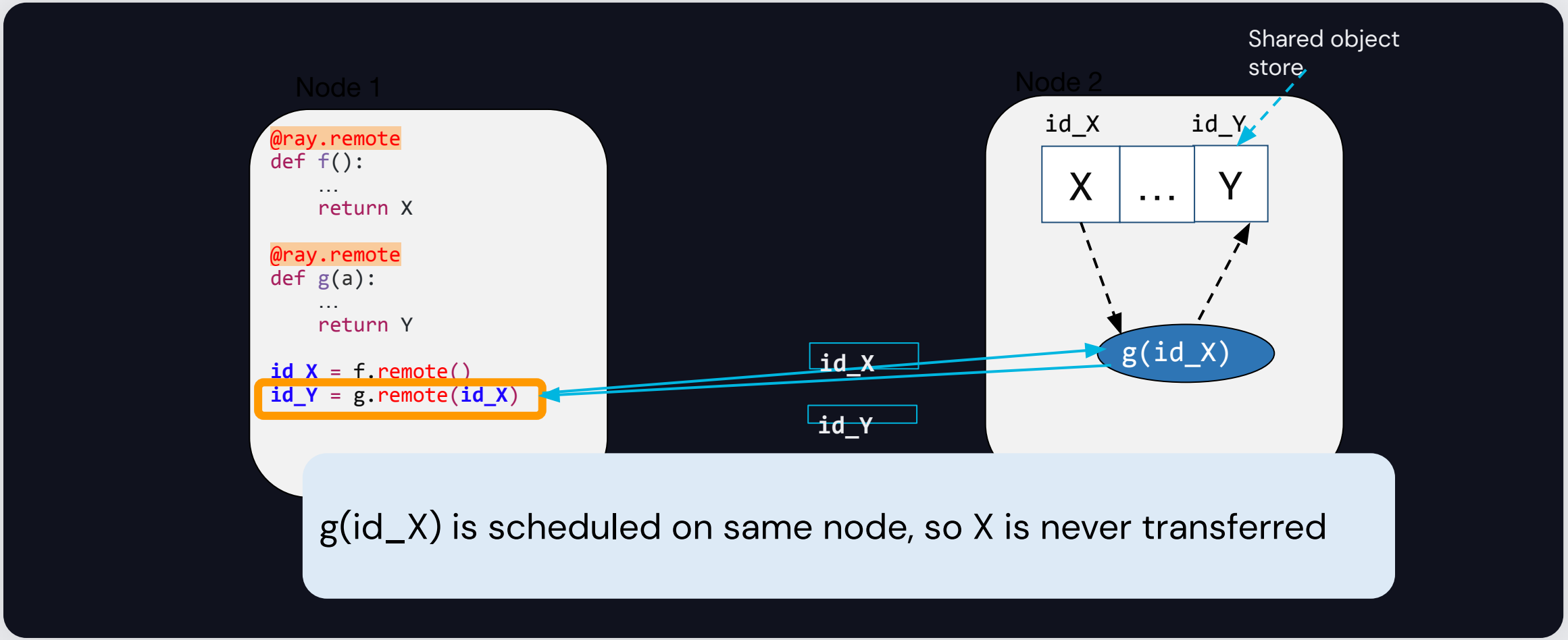


Spill over to external storage

Distributed object store



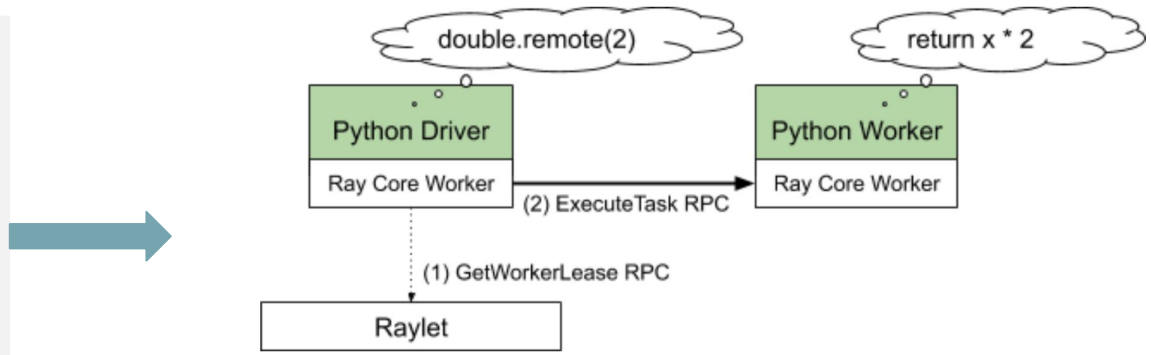
Distributed object store



How Raylet schedules Tasks

Basic Ray Task Call

```
@ray.remote
def double(x):
    return x * 2


fut1 = double.remote(2)
assert ray.get(fut1) == 4
```

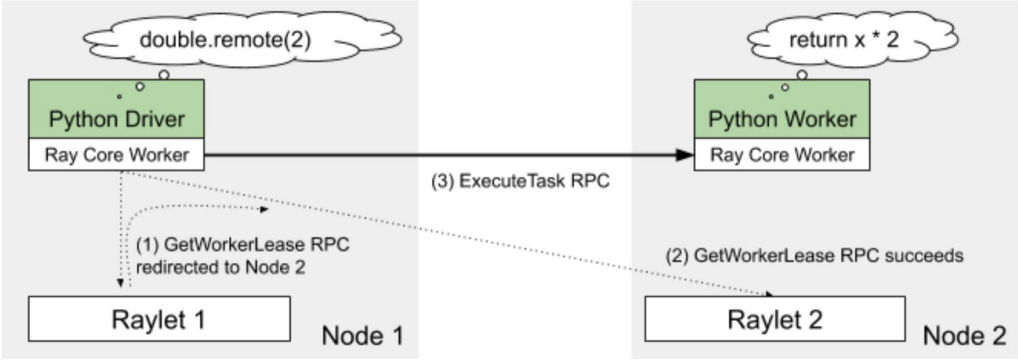


Components in green boxes represent Python code. Components in white boxes are part of the Ray common runtime written in C++. Joined boxes represent a process. Any Python driver or worker can call into the Ray C++ core worker library to execute further tasks. In this figure, all processes are running on the same machine. Ray uses gRPC as a unified communication layer for both local and remote procedure calls.

How Raylet schedules Tasks

Scaling to Multiple Nodes

1. The driver asks Raylet 1 for a worker to execute `double`. It has no free workers, but Raylet 1 knows Raylet 2 has free resources, and redirects the request to Raylet 2.
2. The driver sends ExecuteTask to the remote Python worker leased from Raylet 2 over gRPC.



Tasks are sent to remote workers if there are no local resources available, transparently scaling Ray applications out to multiple nodes.

Ray Native Library: Ray Tune

Ray Tune for distributed HPO

Why use Ray tune?

- Efficient algorithms for parallel trials
- Effective orchestration of distributed trials
- Easy APIs
- Interoperable with Ray Train and Ray Datasets

Ray Tune supports SOTA

Search Algorithms

Search Algorithms (tune.suggest)

Tune's Search Algorithms are wrappers around open-source optimization libraries for efficient hyperparameter selection. Each library has a specific way of defining the search space - please refer to their documentation for more details.

You can utilize these search algorithms as follows:

```
from ray.tune.suggest.hyperopt import HyperOptSearch
tune.run(my_function, search_alg=HyperOptSearch(...))
```

Summary

SearchAlgorithm	Summary	Website	Code Example
Random search/grid search	Random search/grid search		tune_basic_example
AxSearch	Bayesian/Bandit Optimization	[Ax]	ax_example
BlendSearch	Blended Search	[Bs]	blendsearch_example
CFO	Cost-Frugal hyperparameter Optimization	[Cfo]	cfo_example
DragonflySearch	Scalable Bayesian Optimization	[Dragonfly]	dragonfly_example
SkoptSearch	Bayesian Optimization	[Scikit-Optimize]	skopt_example
HyperOptSearch	Tree-Parzen Estimators	[HyperOpt]	hyperopt_example
BayesOptSearch	Bayesian Optimization	[BayesianOptimization]	bayesopt_example
TuneBOHB	Bayesian Opt/HyperBand	[BOHB]	bohb_example
NevergradSearch	Gradient-free Optimization	[Nevergrad]	nevergrad_example
OptunaSearch	Optuna search algorithms	[Optuna]	optuna_example
ZOOptSearch	Zeroth-order Optimization	[ZOOpt]	zoopt_example
SigOptSearch	Closed source	[SigOpt]	sigopt_example
HEBOSearch	Heteroscedastic Evolutionary Bayesian Optimization	[HEBO]	hebo_example

Trial Schedulers

Trial Schedulers (tune.schedulers)

In Tune, some hyperparameter optimization algorithms are written as "scheduling algorithms". These Trial Schedulers can early terminate bad trials, pause trials, clone trials, and alter hyperparameters of a running trial.

All Trial Schedulers take in a `metric`, which is a value returned in the result dict of your Trainable and is maximized or minimized according to `mode`.

```
tune.run( ... , scheduler=Scheduler(metric="accuracy", mode="max"))
```

Summary

Tune includes distributed implementations of early stopping algorithms such as Median Stopping Rule, HyperBand, and ASHA. Tune also includes a distributed implementation of Population Based Training (PBT) and Population Based Bandits (PB2).

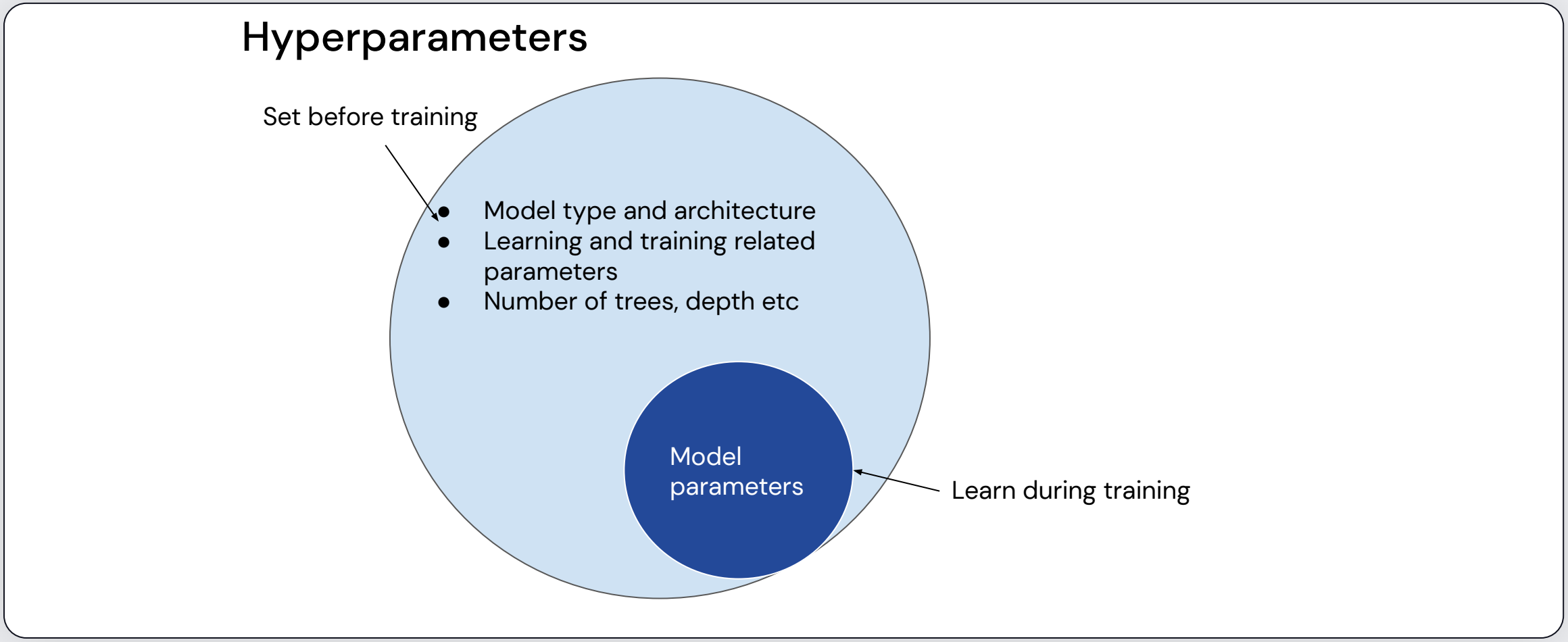
> **Tip**
>
> The easiest scheduler to start with is the `ASHAScheduler` which will aggressively terminate low-performing trials.

When using schedulers, you may face compatibility issues, as shown in the below compatibility matrix. Certain schedulers cannot be used with Search Algorithms, and certain schedulers are require checkpointing to be implemented.

Schedulers can dynamically change trial resource requirements during tuning. This is currently implemented in `ResourceChangingScheduler`, which can wrap around any other scheduler.

Scheduler	Need Checkpointing?	SearchAlg Compatible?	Example
ASHA	No	Yes	Link
Median Stopping Rule	No	Yes	Link
HyperBand	Yes	Yes	Link
BOHB	Yes	Only TuneBOHB	Link
Population Based Training	Yes	Not Compatible	Link
Population Based Bandits	Yes	Not Compatible	Basic Example, PPO example

What are hyperparameters?

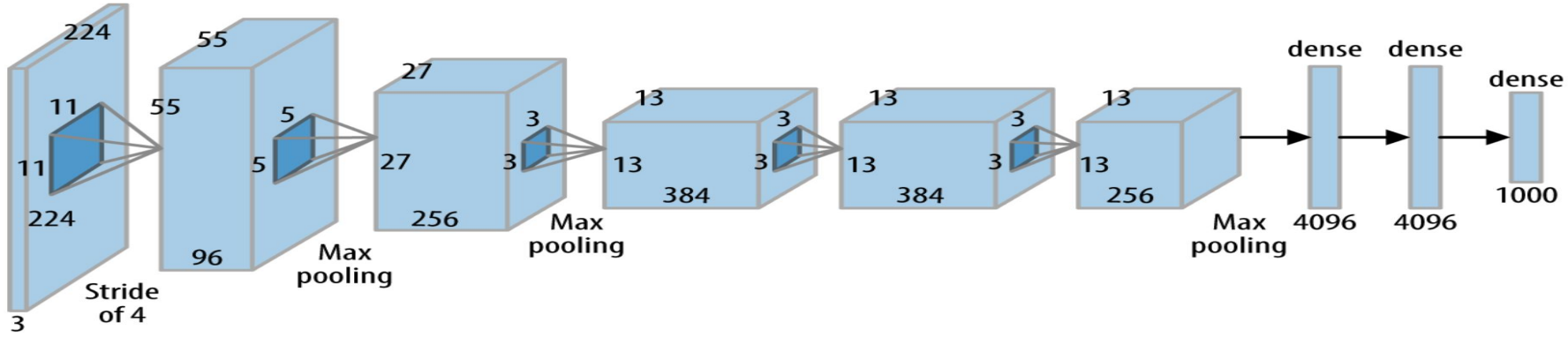


What are hyperparameters

A concrete example for NN

Hyperparameter tuning

"choosing a set of optimal hyperparameters for a learning algorithm"



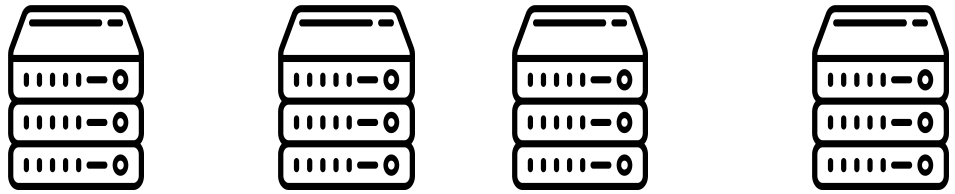
Example: what network structure is best for your binary classification problem?

How many layers? What kinds of layers? Learning rate schedule?
Every number here is a hyperparameter!

Challenges of HPO

- Time consuming
- Costly over time
  - Uses resources (CPU/GPU)
- Fault-tolerance and elasticity

$$$



Ray Tune HPO algorithms

Host of algorithms

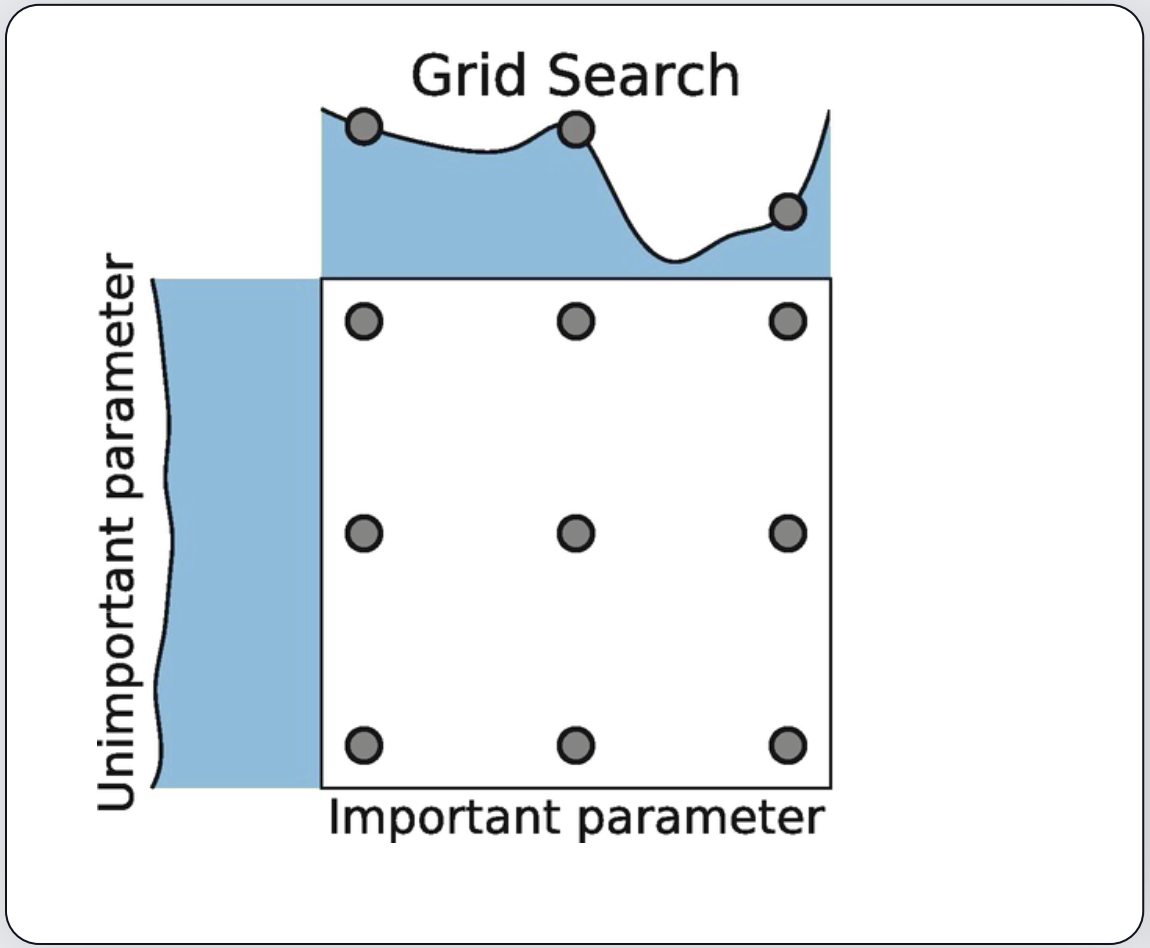
- Over 15+ algorithms natively provided or integrated
- Easy to swap out different algorithms with minimal code change

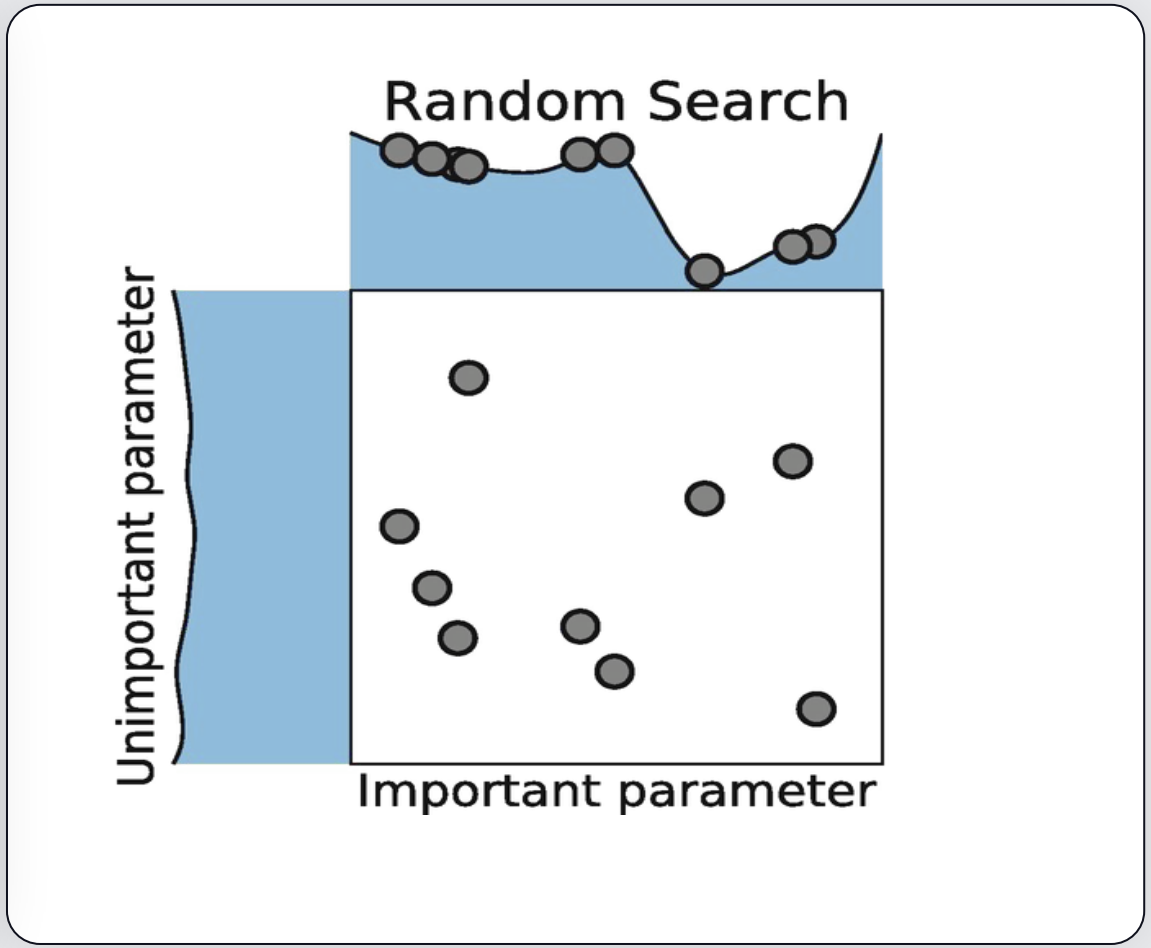
Three strategies

1. Exhaustive search
2. Bayesian optimization
3. Advanced scheduling

1. Exhaustive Search

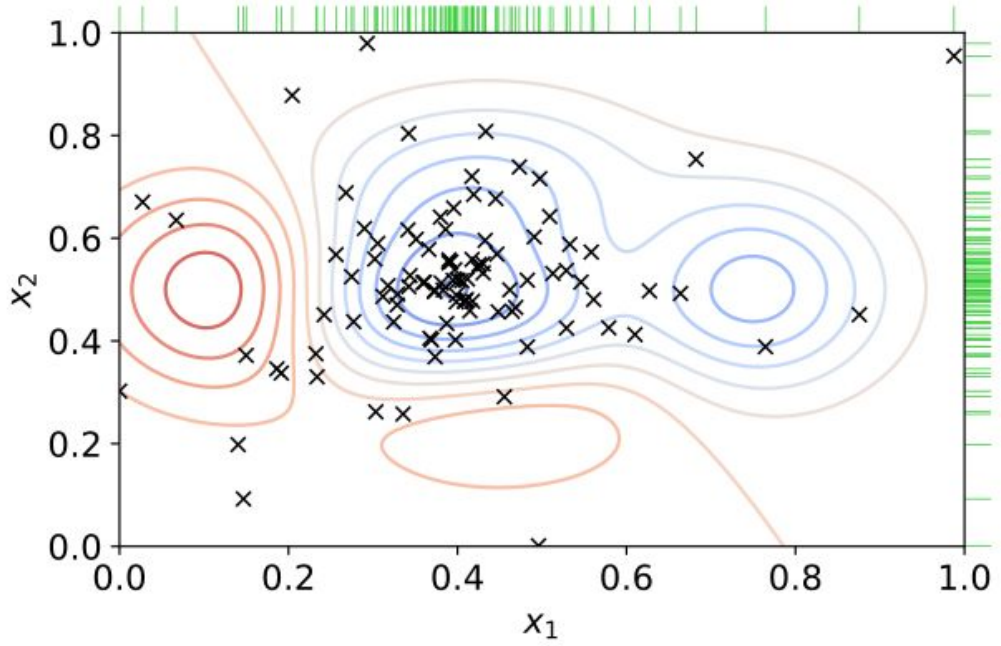
- Easily parallelizable and easy to use





2. Bayesian Optimization

- Uses results from previous combinations for the next trial
- Inherently sequential
- Popular libraries support :
  - Hyperopt
  - Optuna
  - Scikit-optimize
  - Nevergrad

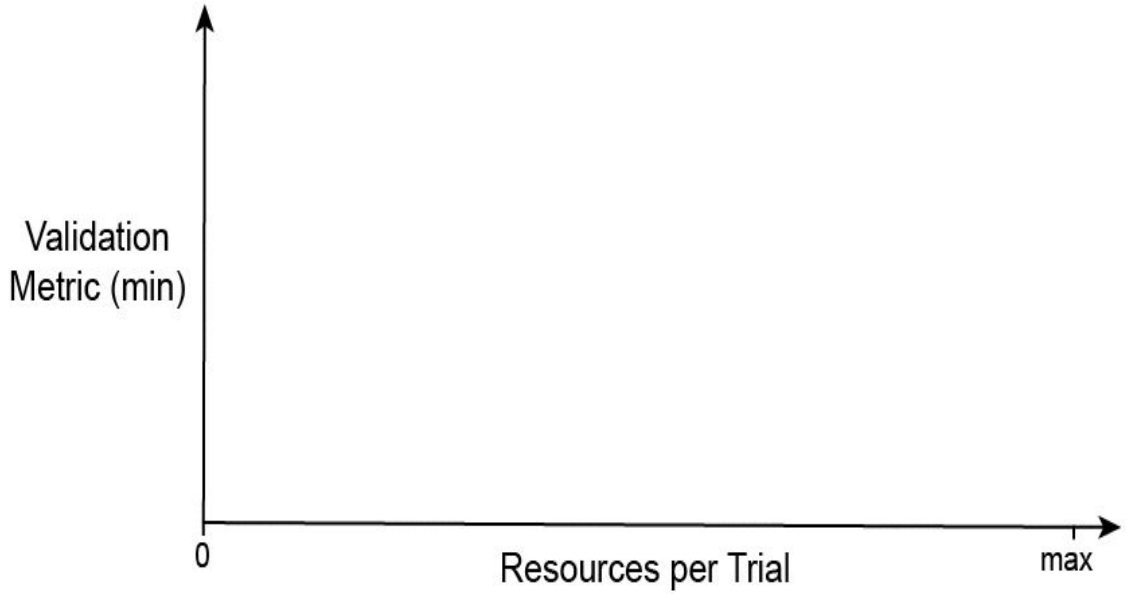


https://www.wikiwand.com/en/Hyperparameter_optimization

3. Advanced Scheduling

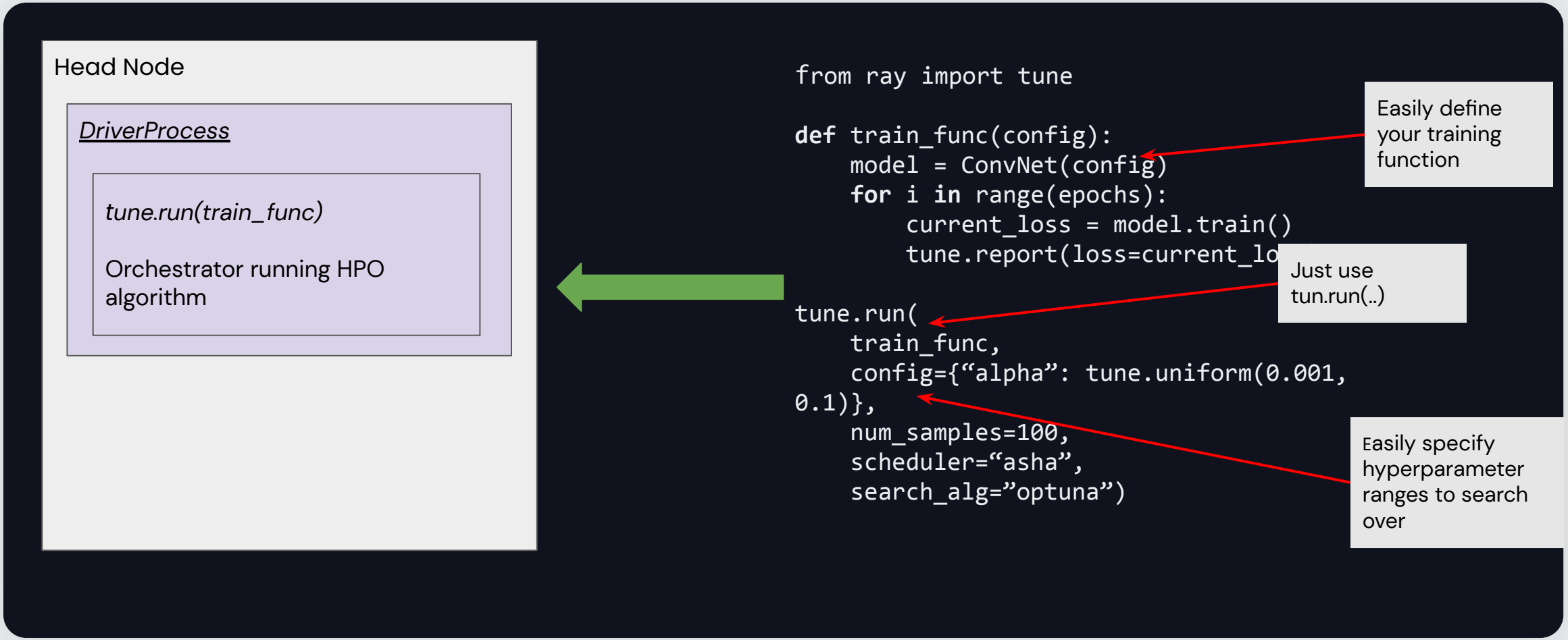
Early Stopping

- Fan out parallel trials and observe
- Use intermediate results (epochs, tree sample) to prune underperforming trials
- Median stopping, ASHA/Hyperband



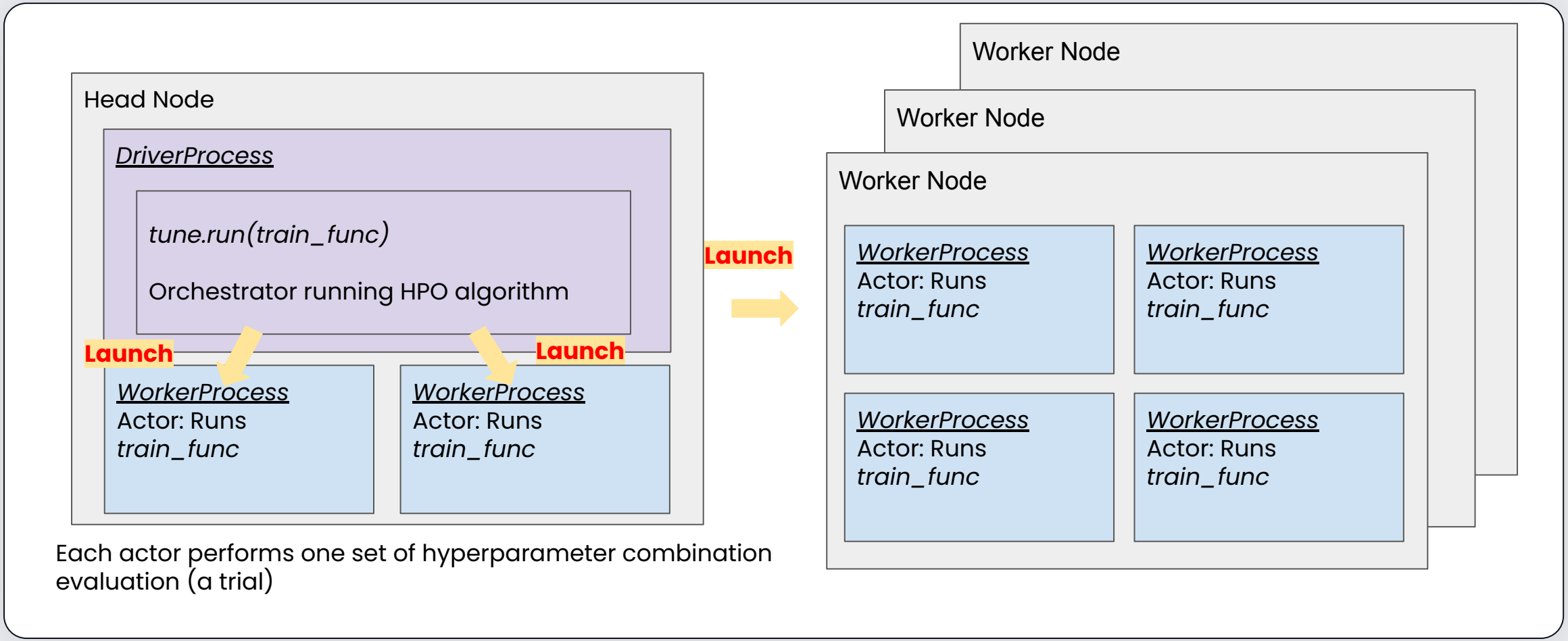
Ray Tune – Distribute HPO Example

Architecture and components



Ray Tune – Distributed HPO

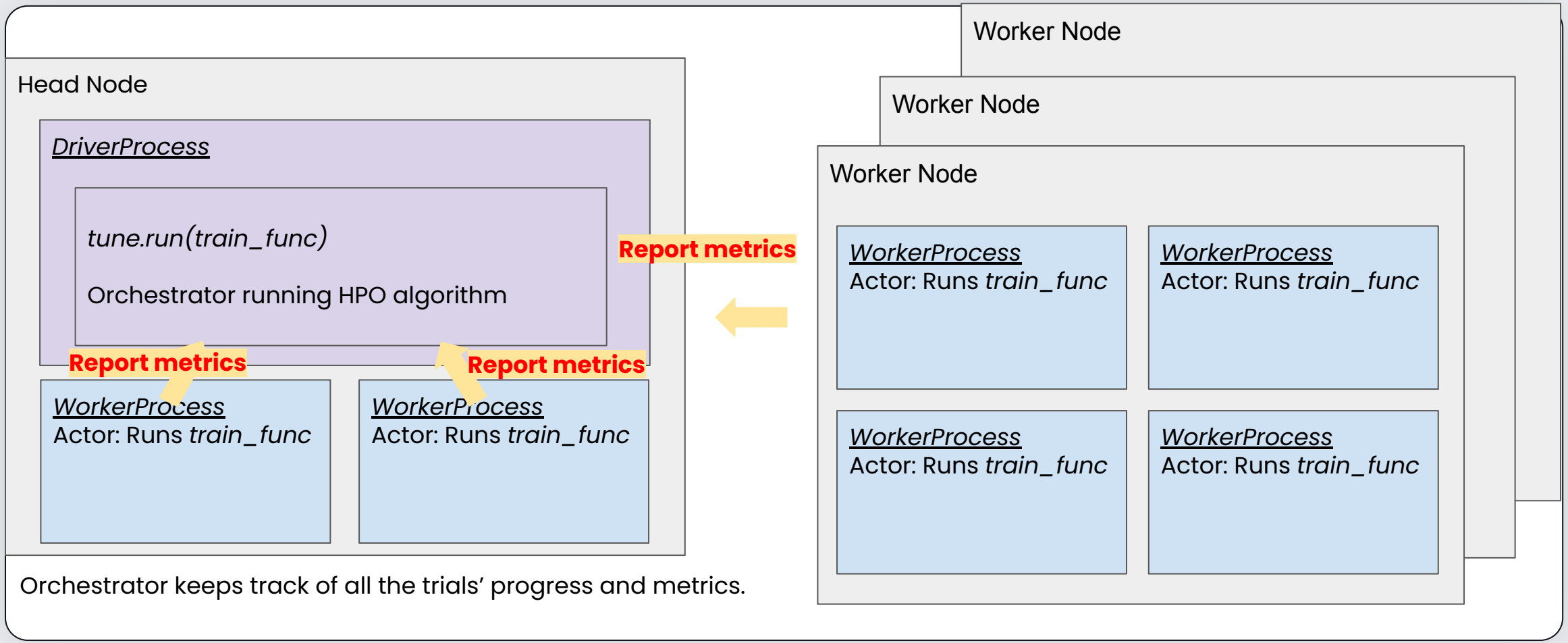
Architecture and components



Each actor performs one set of hyperparameter combination evaluation (a trial)

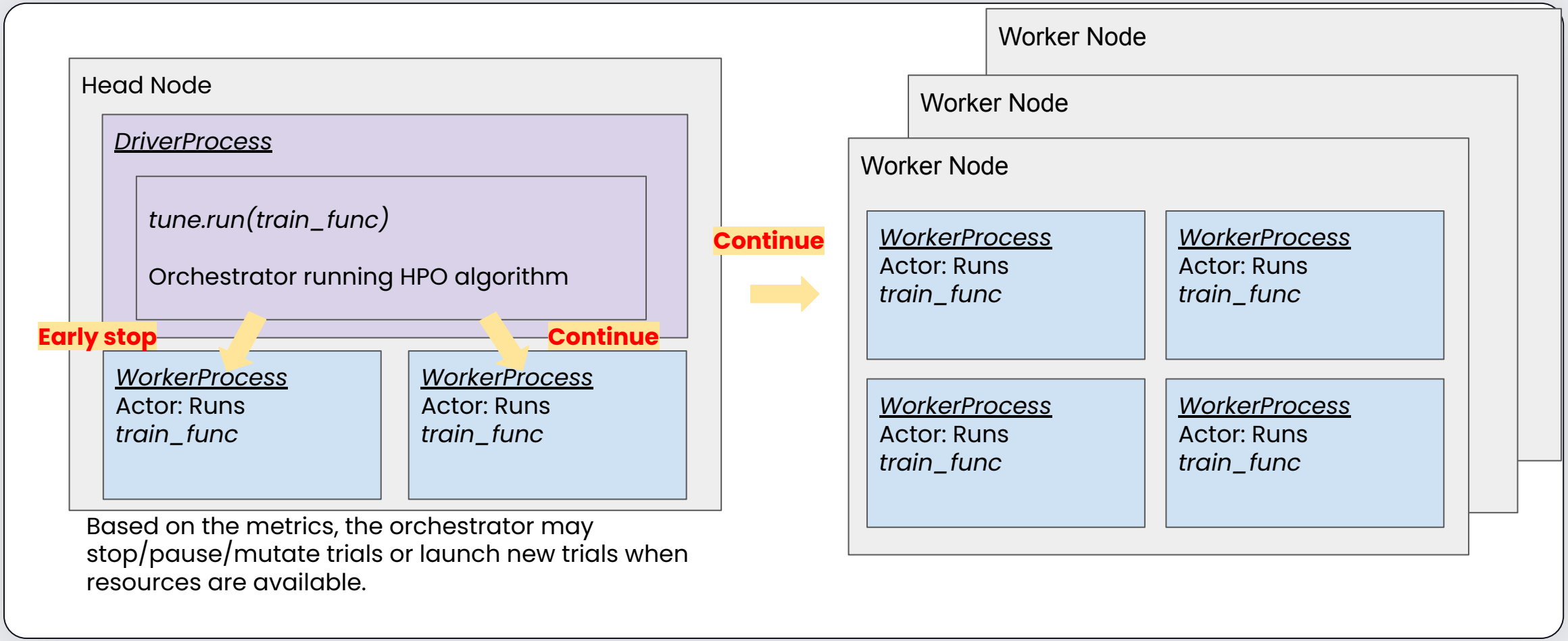
Ray Tune – Distributed HPO

Architecture and components



Ray Tune – Distributed HPO

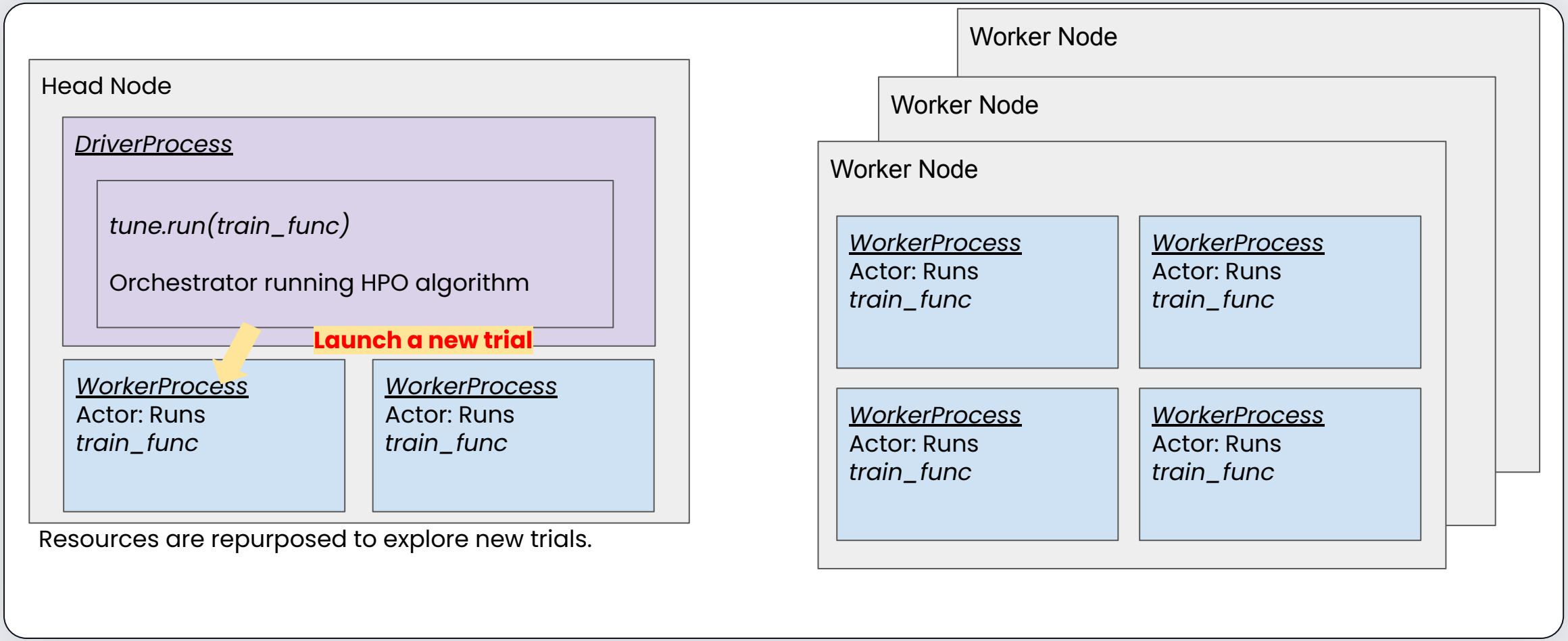
Architecture and components



Based on the metrics, the orchestrator may stop/pause/mutate trials or launch new trials when resources are available.

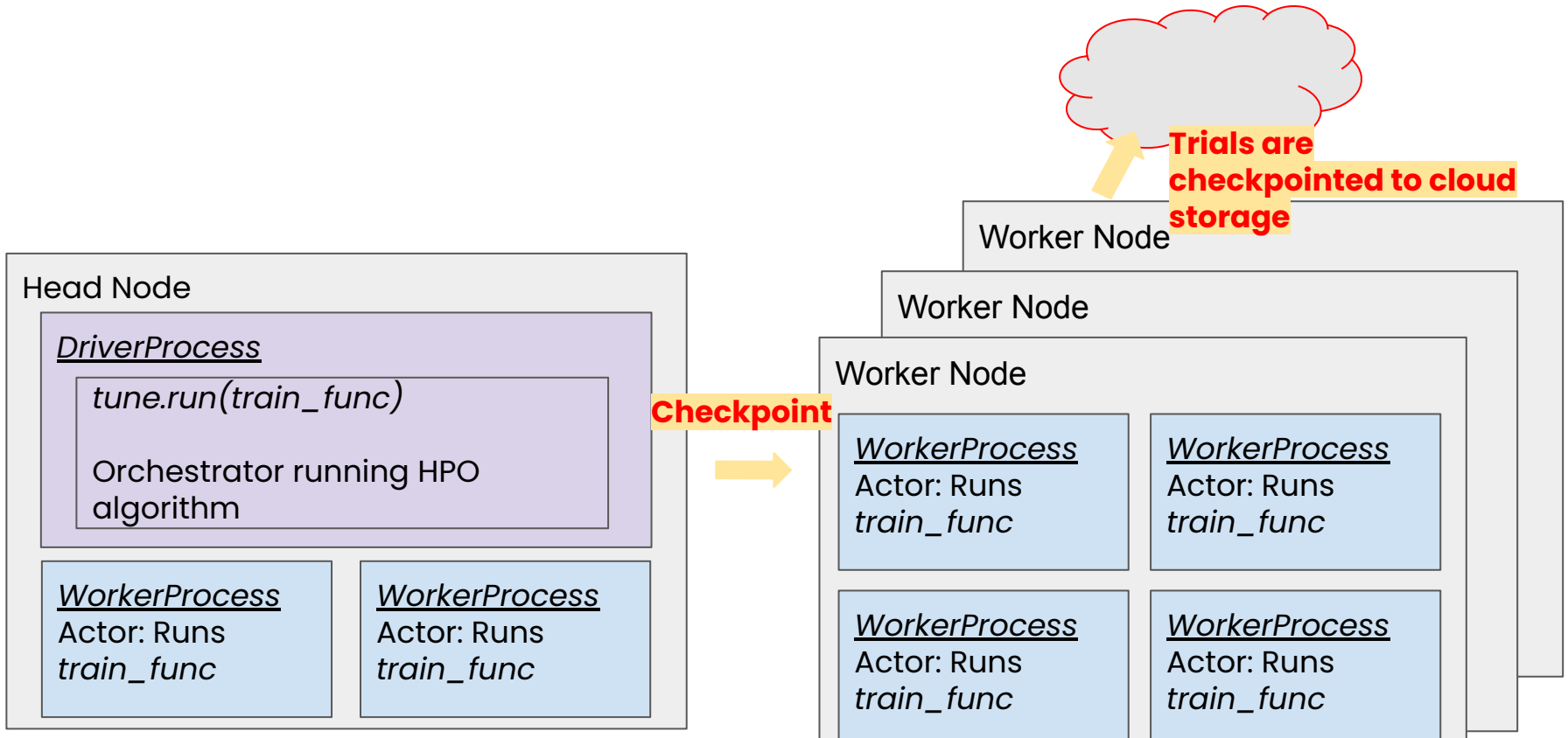
Ray Tune – Distributed HPO

Architecture and components



Ray Tune – Distributed HPO

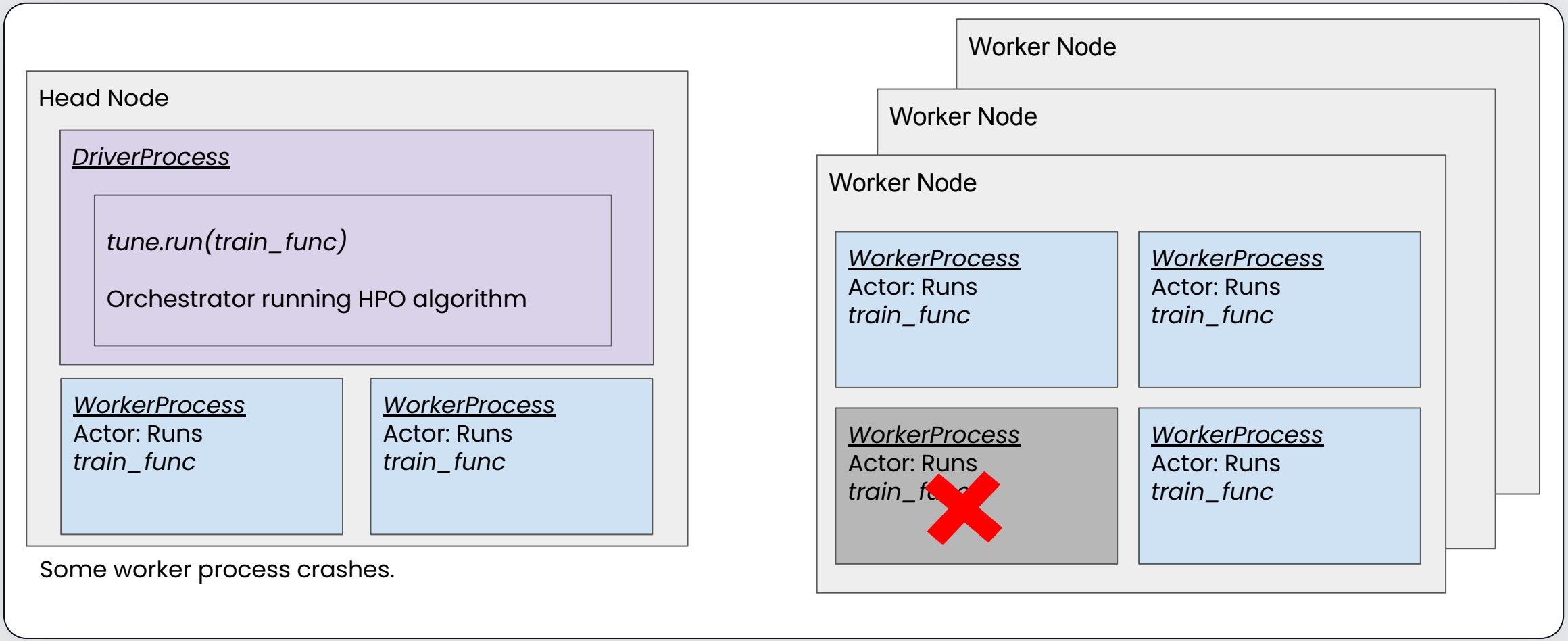
Architecture and components



Orchestrator also manages checkpoint state.

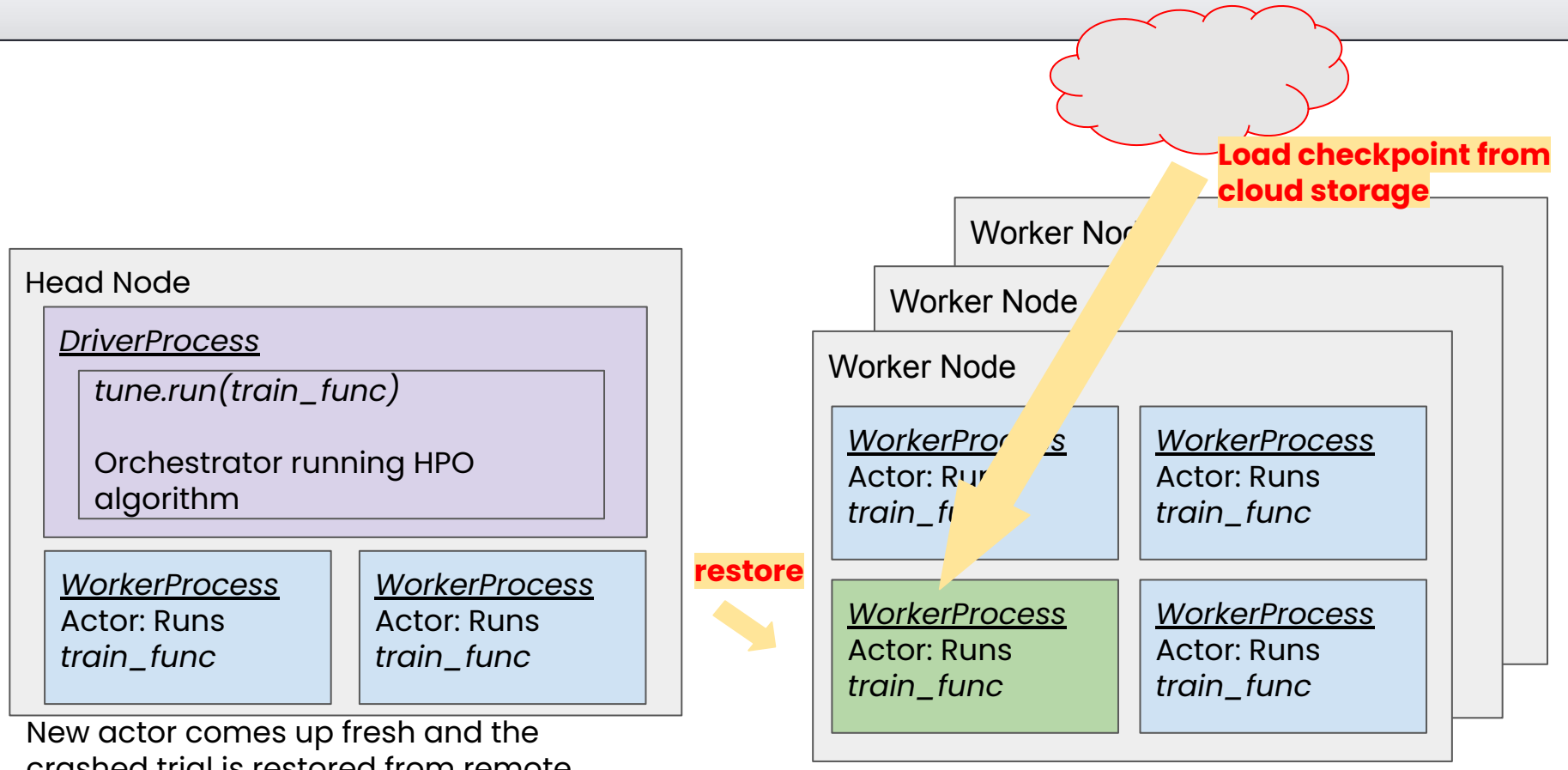
Ray Tune – Distributed HPO

Architecture and components



Ray Tune – Distributed HPO

Architecture and components



New actor comes up fresh and the crashed trial is restored from remote checkpoint.

Takeaways

- Distributed computing is a necessity & norm
- Ray's vision: make distributed programming simple
  - Don't have to be distributed systems expert. Just use @ray.remote :)
- Scale your ML workloads with Ray Libraries

RAY Libraries Documentation Learn Ecosystem Community 20,541

Get involved and stay informed

Live and on-demand. Virtual and in-person. Ask questions and share learnings. File bugs and submit code. Whatever your preference, there are lots of ways to connect with the global Ray community, get involved with the project, and stay informed on the latest and greatest.

ray.io/community

GitHub

Follow the project, track issues, file bugs and feature requests, or contribute code.

Follow the project →

Discussion forum

Join the forum to get technical help and share best practices and tips with the Ray community.

Join the forum →

Slack

Connect with other users and project maintainers on the official Ray Slack channel.

Chat with other users →

Newsletter

Subscribe to the monthly Ray newsletter to get curated updates delivered to your inbox.

Sign up for updates →

Twitter

Follow @raydistributed on Twitter to stay informed on the latest news and updates.

Follow us on Twitter →

DATA+AI
SUMMIT 2022

Thank you!

Let's stay in touch:

 @2twitme

jules@anyscale.com

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

Demo: Scaling ML workloads

File Edit View Run Kernel Tabs Settings Help

/

Name

- images
- lightgbm_ray
- model.xgb
- xgboost_demo.ipynb

xgboost_demo.ipynb

Markdown

Python 3 (ipykernel)

2. Hyperparameter tuning
3. Inference

First, we try on a local node with a data set, time it, and then try on a Ray cluster with multiple nodes and multiple cores.

We should observe noticiable difference.