Scaling AI/ML Workloads with Ray Ecosystem

Jules S. Damji, @2twitme

Lead Developer Advocate, Ray Team @ Anyscale

Data Council, Austin, TX March 23, 2022

anyscale

Overview

- **Why & What Ray & Ray Ecosystem**
- **Ray Architecture & Components**
- **Ray Core APIs**
- **Ray Native ML Libraries**
  - Ray Tune, XGBoost-Ray
- **Demo**
  - Scaling ML workloads
- **Q & A**

Why Ray?

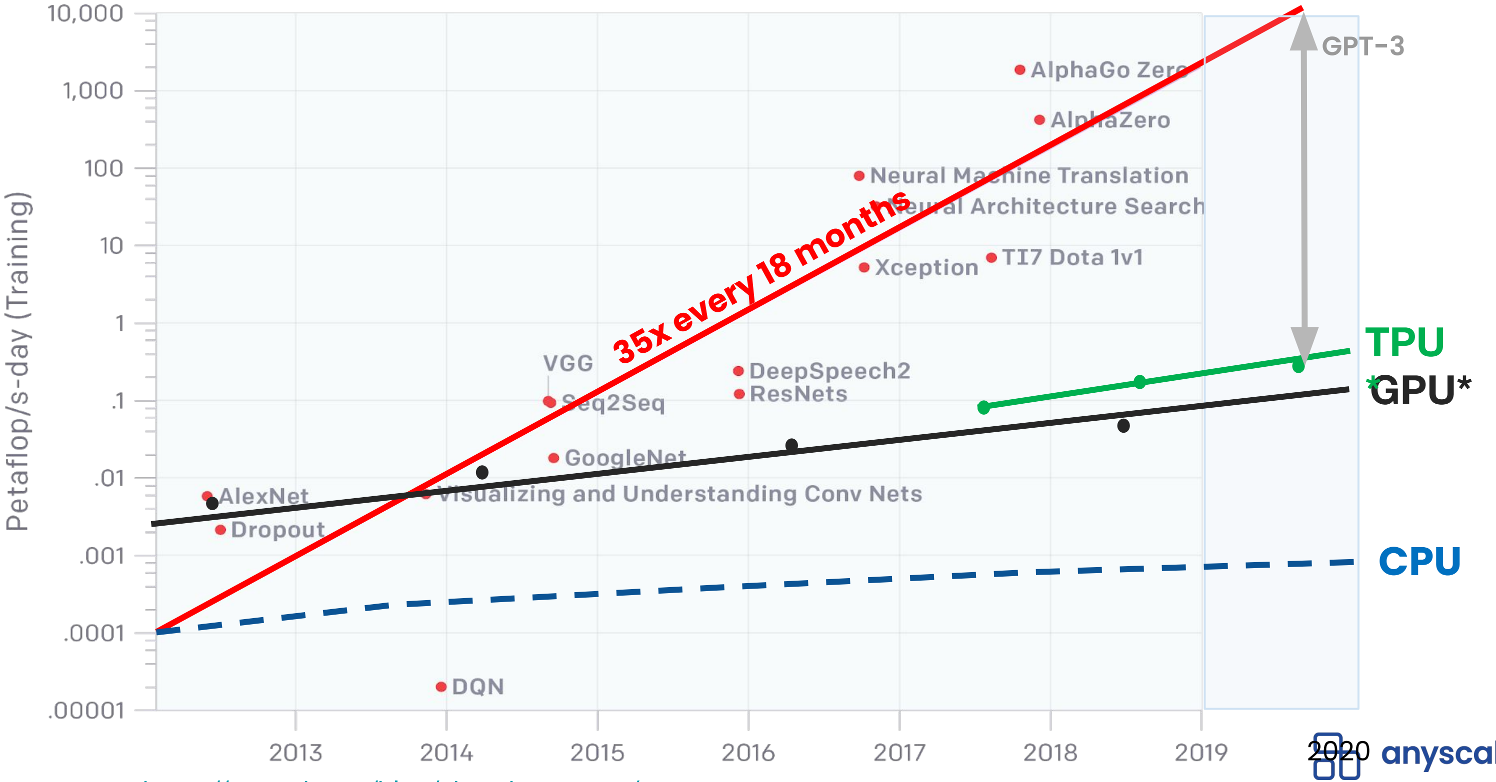
- **Machine learning is pervasive in every domain**
- Distributed machine learning is becoming a necessity
- Distributed systems is notoriously hard

Why Ray?

- Machine learning is pervasive in every domain
- **Distributed machine learning is becoming a necessity**
- Distributed systems is notoriously hard

anyscale

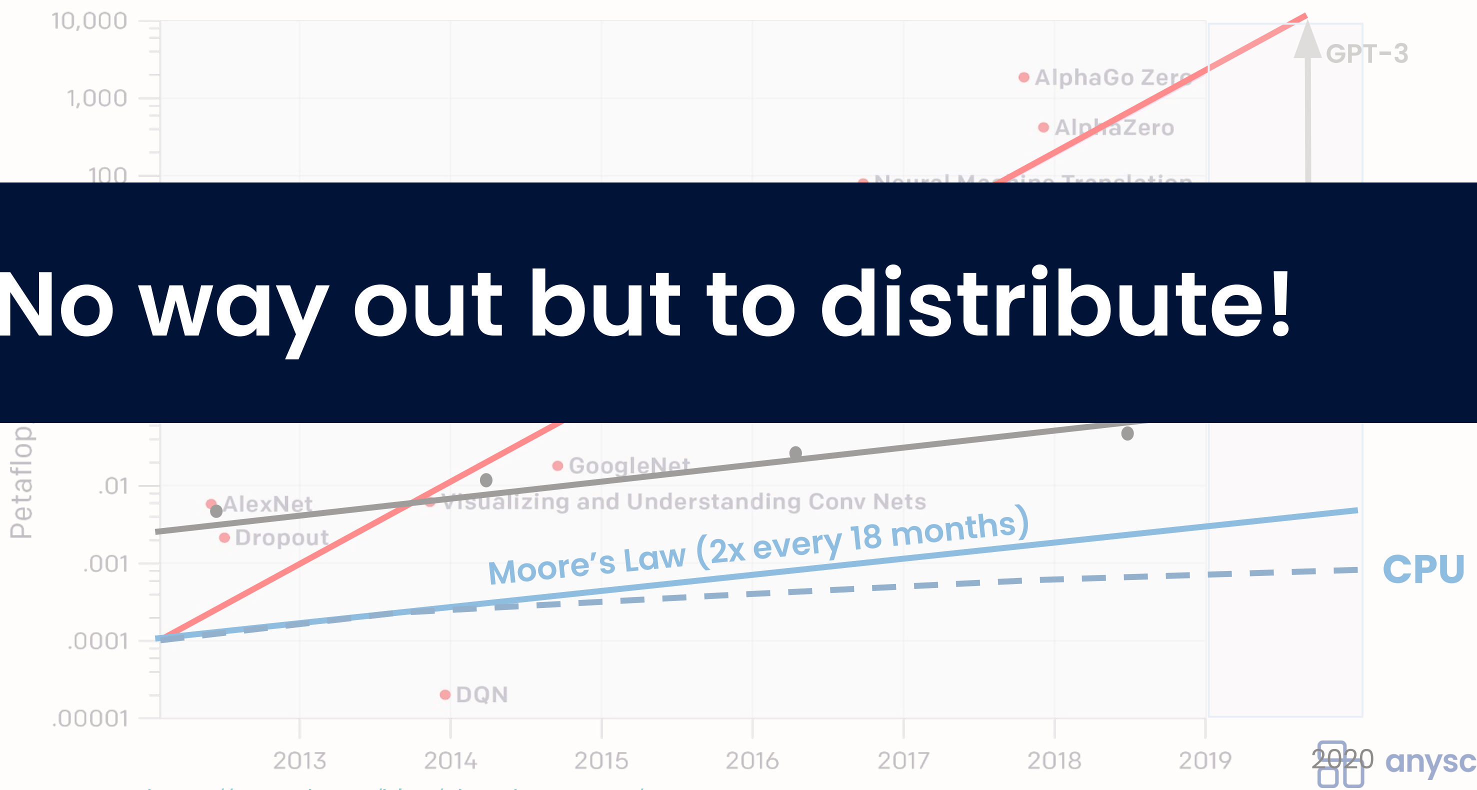
Specialized hardware is also not enough



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

RAY anyscale

Specialized hardware is also not enough



No way out but to distribute!

RAY

anyscale

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

Why Ray?

- Machine learning is pervasive in every domain
- Distributed machine learning is becoming a necessity
- **Distributed systems and programming are notoriously hard**

Existing solutions have may tradeoffs

Ease of development

Serverless

Limitations

01. Cloud specific
02. Stateless only
03. No GPUs/TPUs
04. Runtime limit

Stitch together existing frameworks

Hard to

01. Develop
02. Deploy
03. Manage

Clean slate

Expensive to develop

01. Time
02. People

Generality

RAY

anyscale

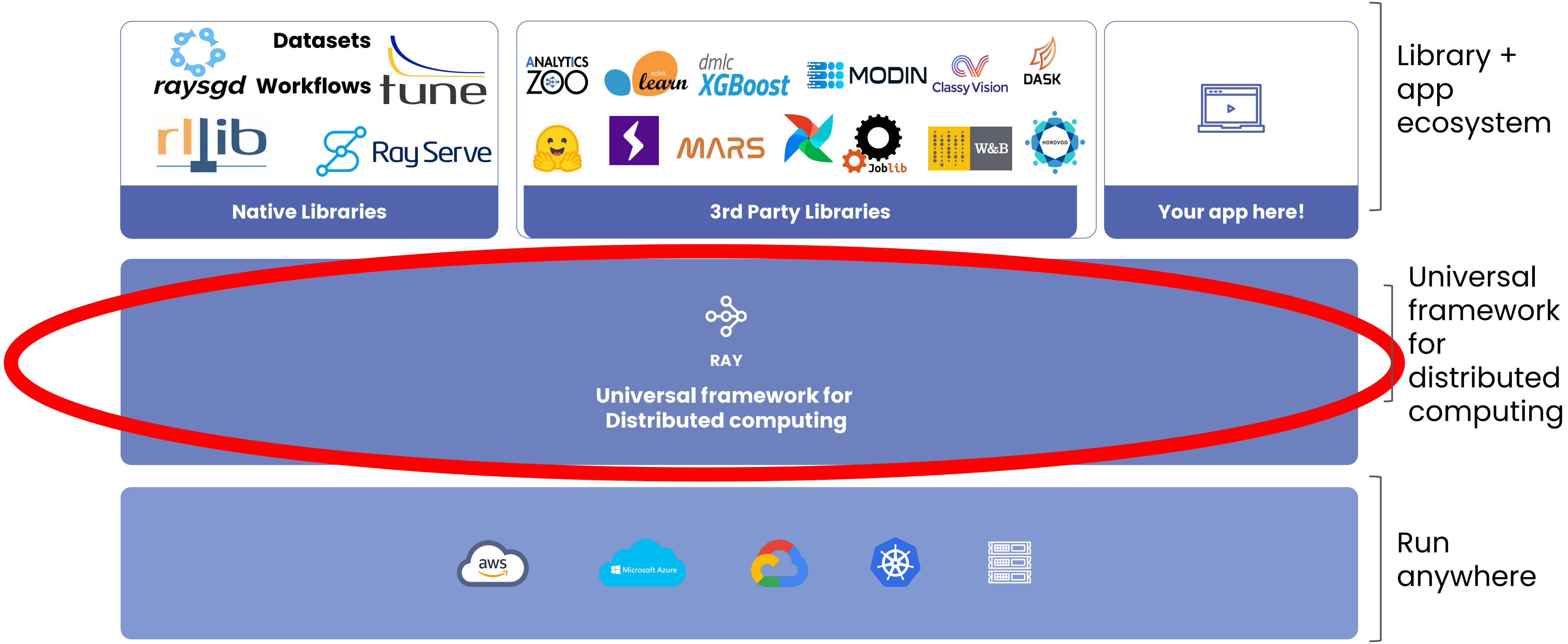
Why Ray?

- Machine learning is pervasive in every domain
- Distributed machine learning is becoming a necessity
- Distributed systems are notoriously hard

Ray's vision:

Make distributed computing accessible to every developer

The Ray Layered Cake and Ecosystem



Rich ecosystem for scaling ML workloads

	Data Processing	Training	Serving	Hyper. Tuning	Reinforcement Learning
Built-in **"batteries included"** libraries	Ray Core + Datasets	Ray Train	Ray Serve	tune	rllib
	MODIN, MARS, DASK	PyTorch, scikit learn, HOROVOD	SELDON	HYPEROPT, OPTUNA	

Only use the libraries you need!

Companies scaling ML with Ray

amazon
intel
dendra SYSTEMS
Uber
Microsoft
VISA
McKinsey & Company
ANT GROUP
rbi restaurant brands international
WILD LIFE
robinhood
NVIDIA
shopify
IBM
TWO SIGMA
Alibaba.com

Companies scaling ML with Ray

Data Processing	Training	Serving	Hyper. Tuning	Reinforcement Learning
amazon	rbi restaurant brands international	WILD LIFE	dendra SYSTEMS	McKinsey & Company
ANT GROUP	Uber	IkigaiLabs	shopify	WILD LIFE

- https://eng.uber.com/horovod-ray/
- https://www.anyscale.com/blog/wildlife-studios-serves-in-game-offers-3x-faster-at-1-10th-the-cost-with-ray
- https://www.ikigailabs.com/blog/how-ikigai-labs-serves-interactive-ai-workflows-at-scale-using-ray-serve

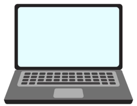
Ray's approach for scaling ML

(Traditional, non-parallelized vanilla Python)

(with as few code changes as possible)

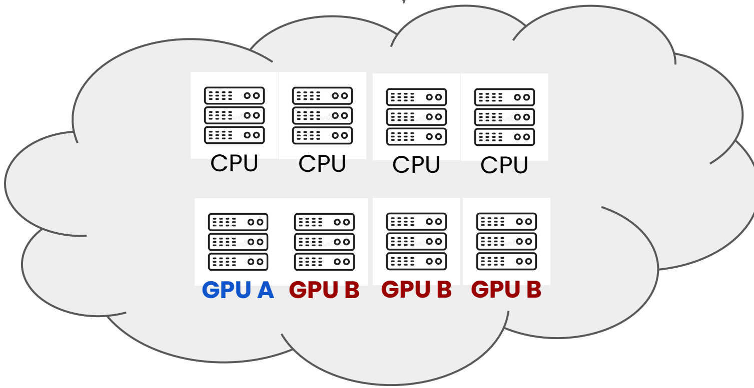
Ray-ified Python

Runs on



Same Python code runs on laptop as infinite cloud!

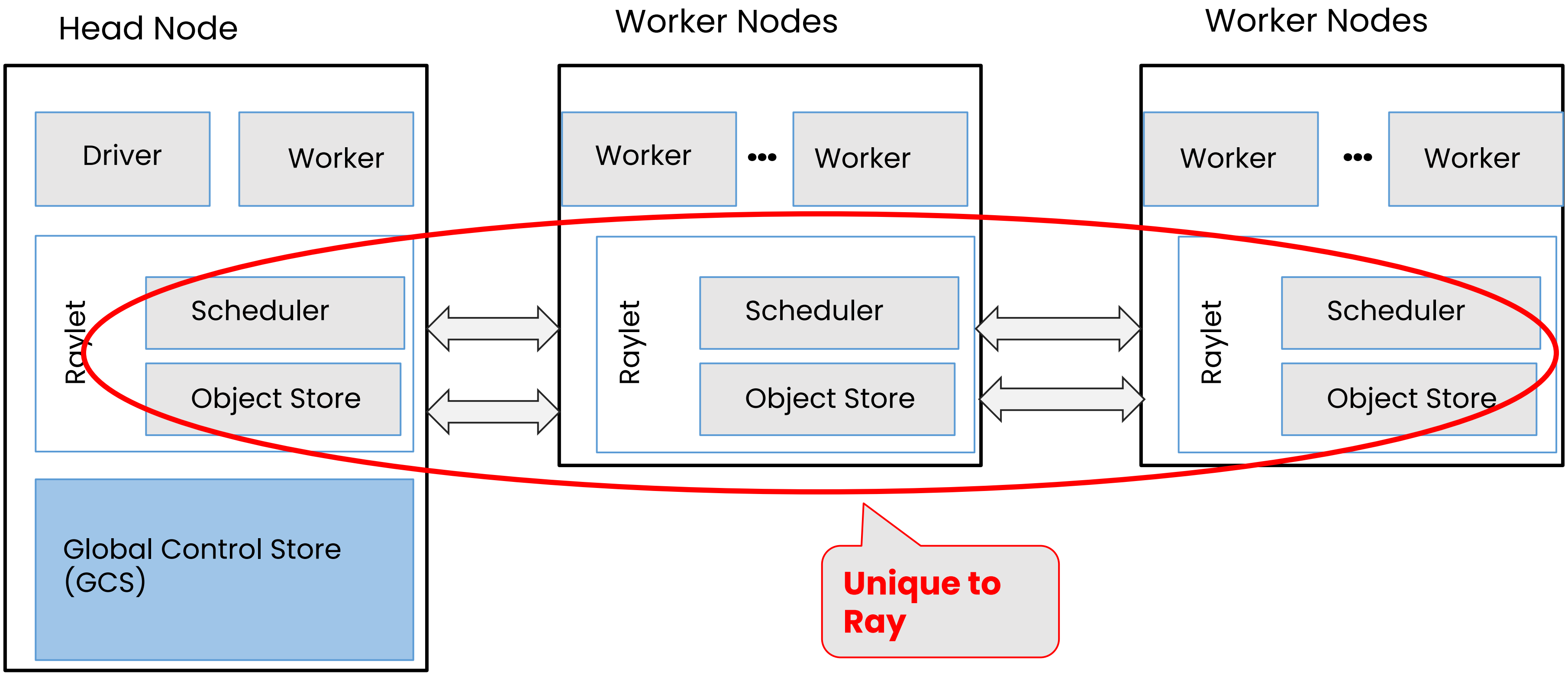
Runs on

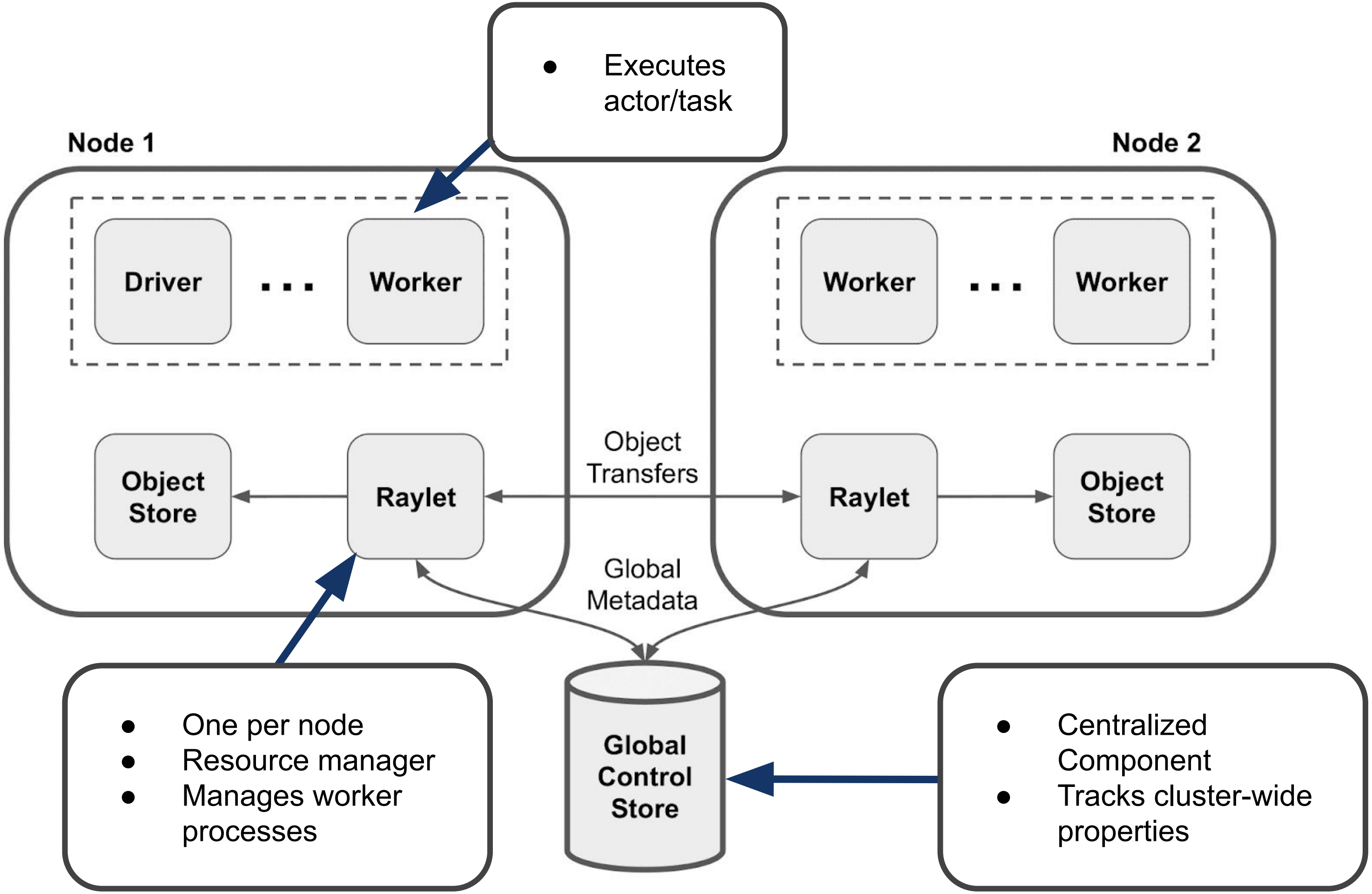


RAY

Ray Architecture & Components

What does Ray Cluster Looks Like …





Executes actor/task
Node 1
Node 2
Driver
Worker
Worker
Worker
Object Store
Raylet
Object Transfers
Raylet
Object Store
Global Metadata
One per node
Resource manager
Manages worker processes
Global Control Store
Centralized Component
Tracks cluster-wide properties

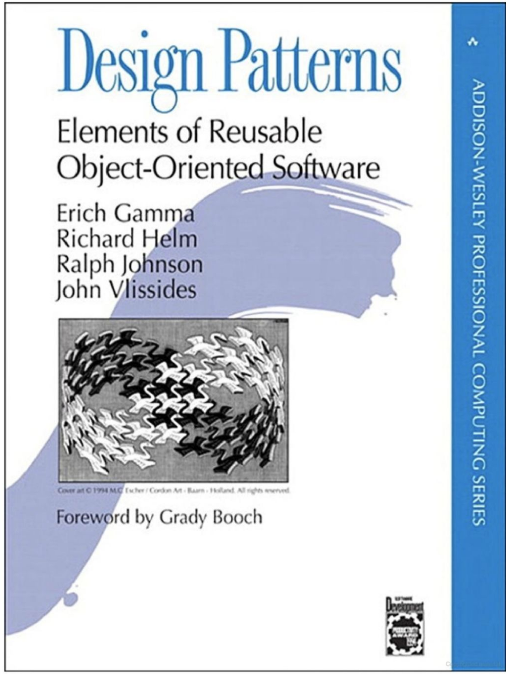
anyscale

Ray Distributed Design Patterns & APIs

Ray Basic Design Patterns

- Ray Parallel Tasks
  - Functions as stateless units of execution
  - Functions distributed across a clusters as tasks
- Ray Objects or Futures
  - Distributed (immutable) Object stored in cluster
  - Retrievable when available
  - Enable asynchronous execution of
- Ray Actors
  - Stateful service on a cluster
  - Message passing and maintains state

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



RAY

anyscale

Python → Ray Basic Patterns

Function → Task → Node

Class → Actor → Node

Object → (Distributed immutable) Object → Node

Function → Task

```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)
```

Class → Actor

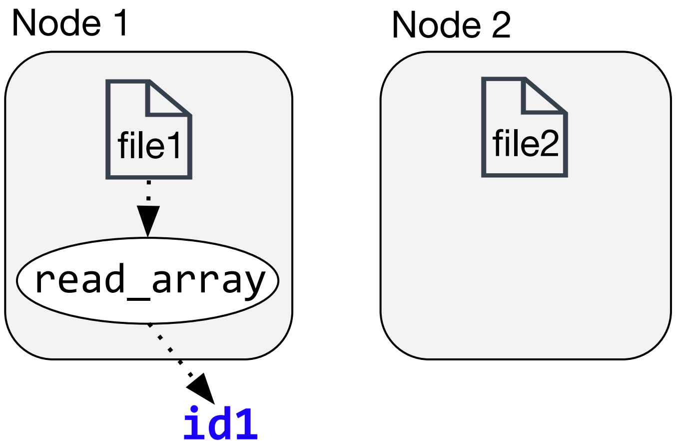
```python
@ray.remote(num_gpus=1)
class Counter(object):
    def __init__(self):
        self.value = 0
    def inc(self):
        self.value += 1
        return self.value

c = Counter.remote()
id4 = c.inc.remote()
id5 = c.inc.remote()
```

anyscale

Task API

Blue variables are ObjectRef IDs (similar to futures)

```
@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)
```

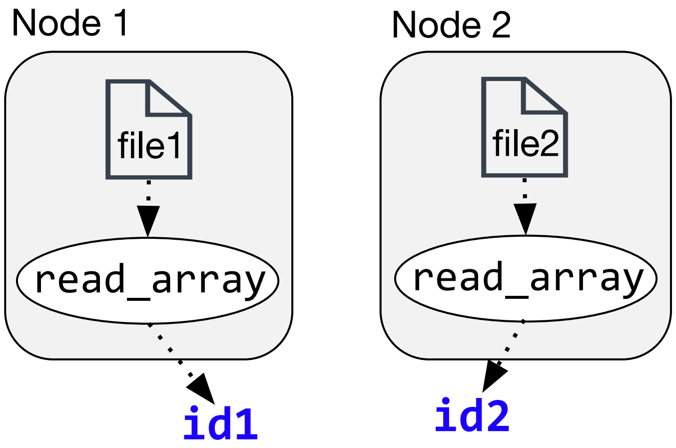


Return id1 (future) immediately, before **read_array()** finishes

Task API

Blue variables are Object IDs
(similar to futures)

```
@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)
```



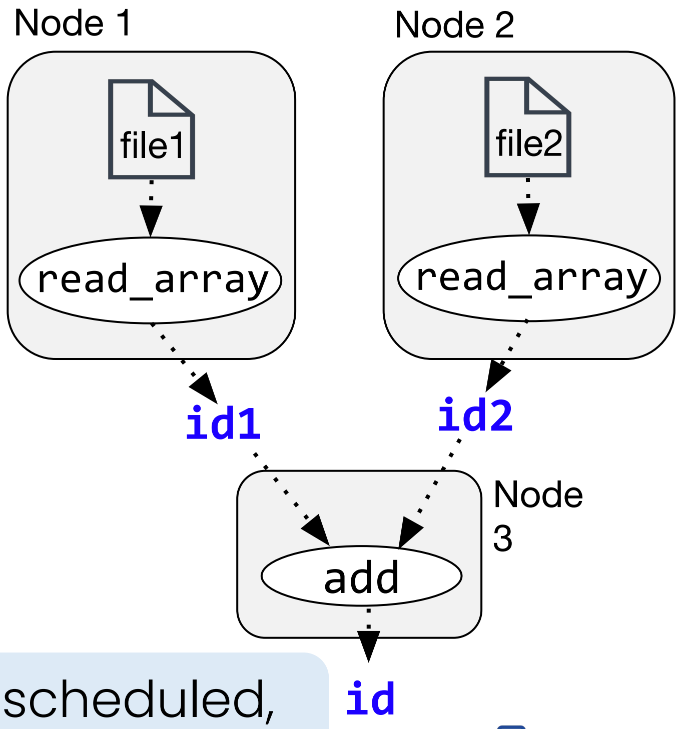
Dynamic task graph:
build at runtime

anyscale

Task API

Blue variables are Object IDs
(similar to futures)

```
@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)
```



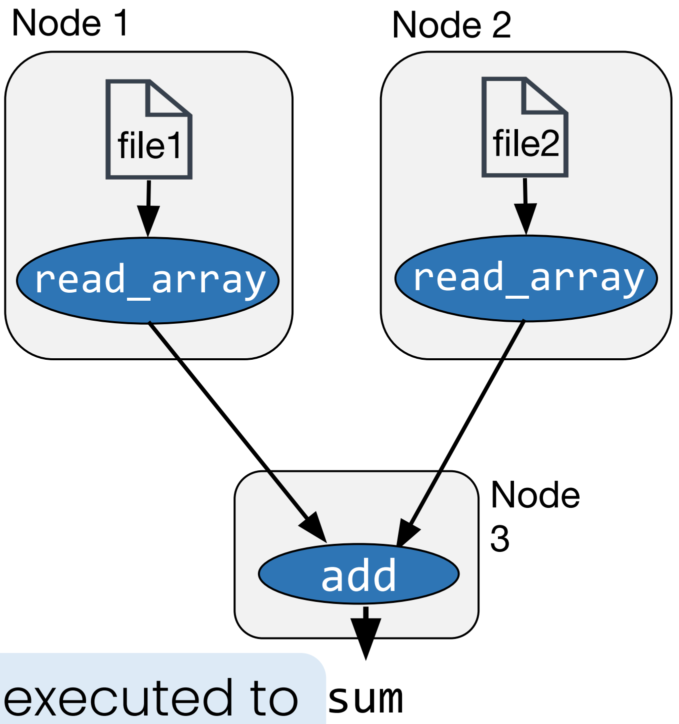
Every task scheduled,
but not finished yet

anyscale

Task API

Blue variables are Object IDs
(similar to futures)

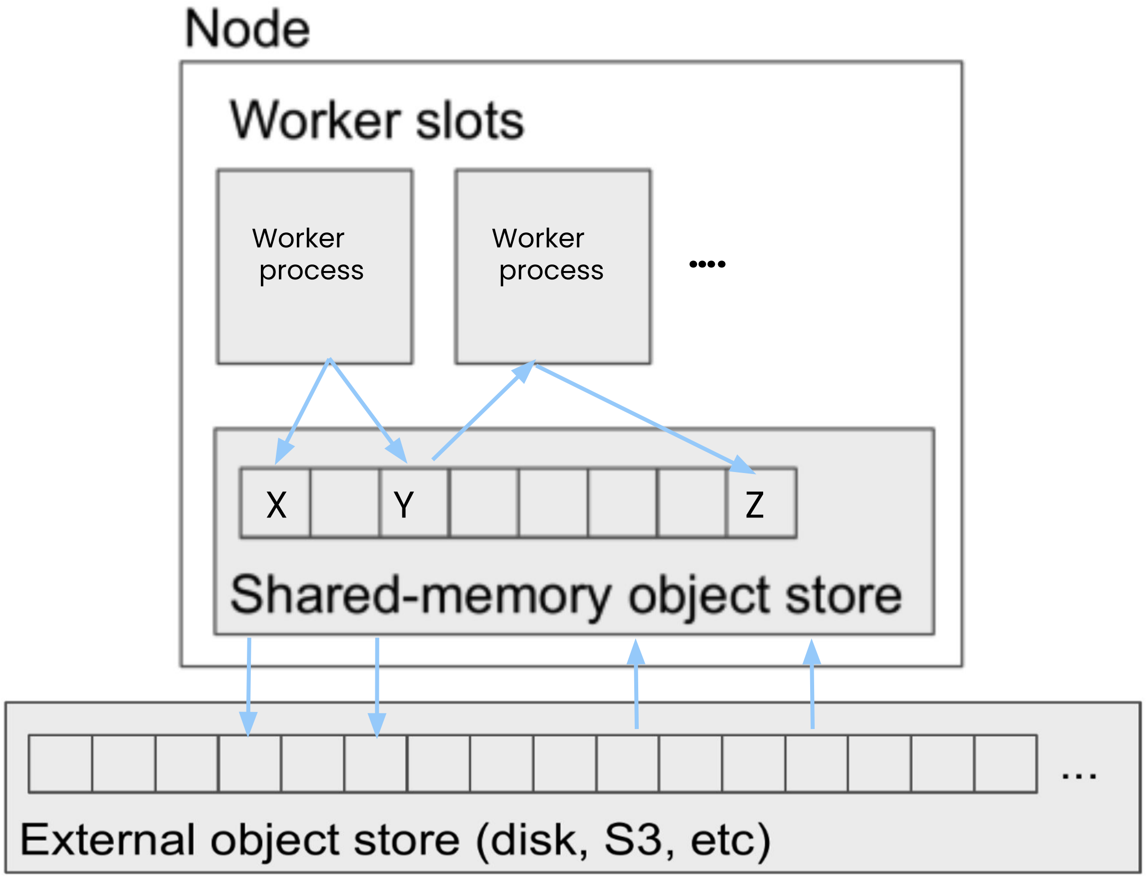
```
@ray.remote
def read_array(file):
    # read ndarray "a"
    # from "file"
    return a

@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 graph executed to compute sum

anyscale

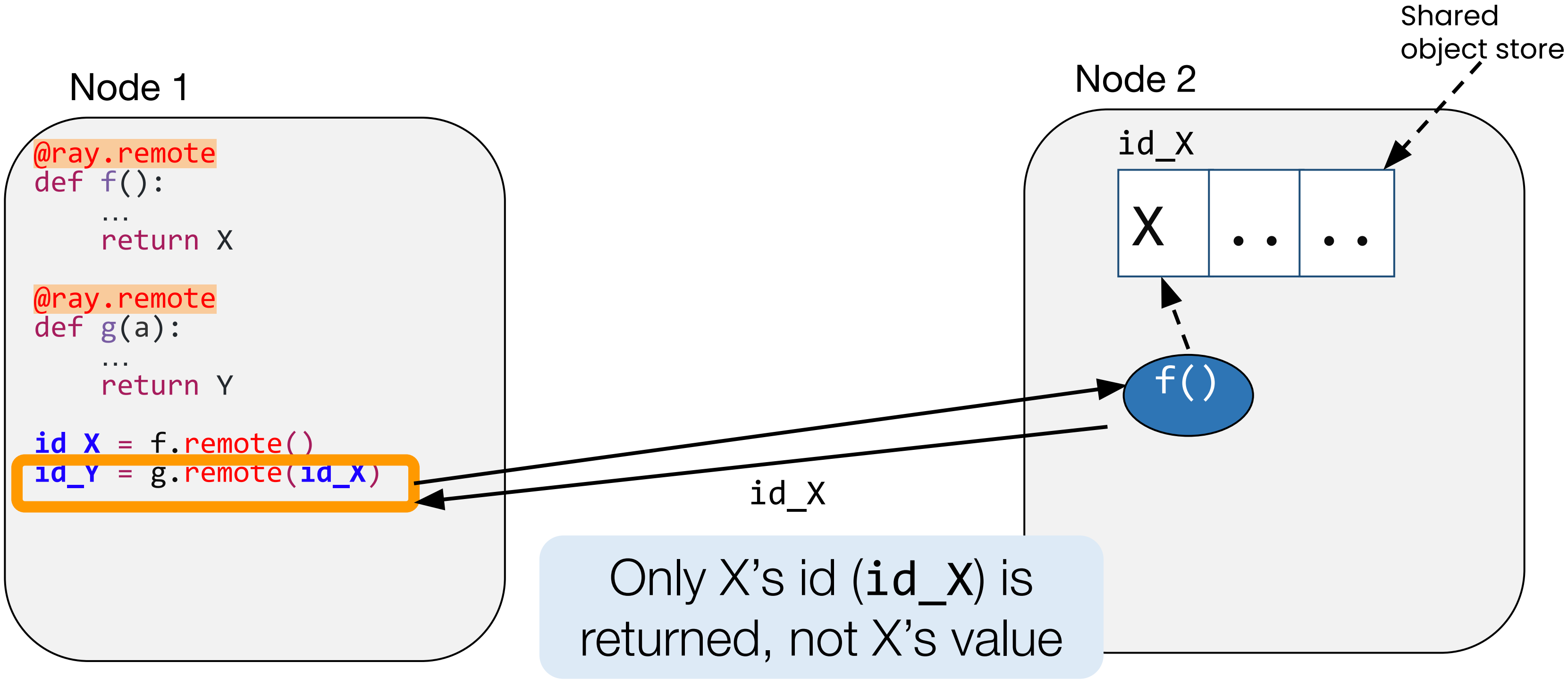
Distributed Immutable object store



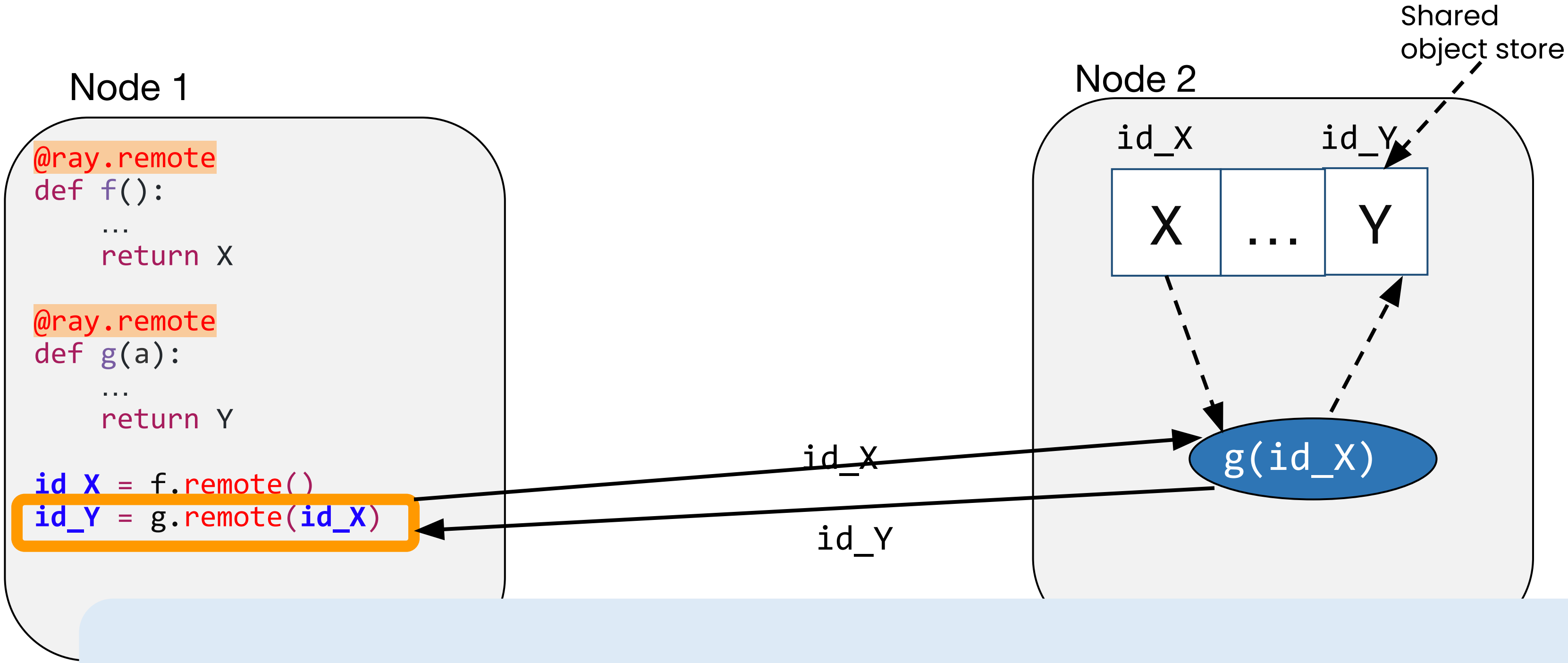
Spill over to external storage

anyscale

Distributed Immutable object store



Distributed object store

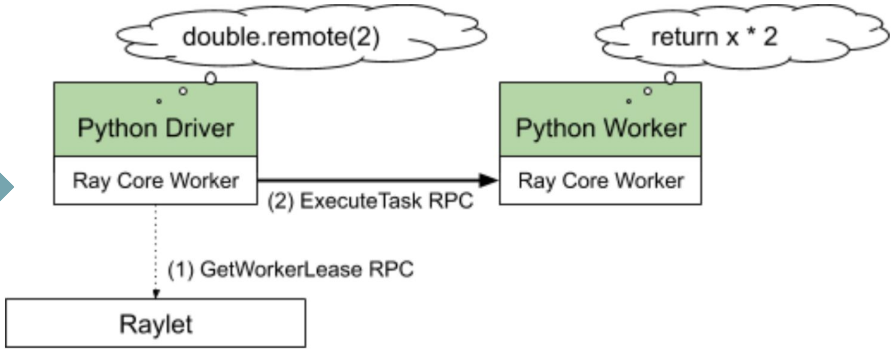


g(id_X) is scheduled on same node, so X is never transferred

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
```

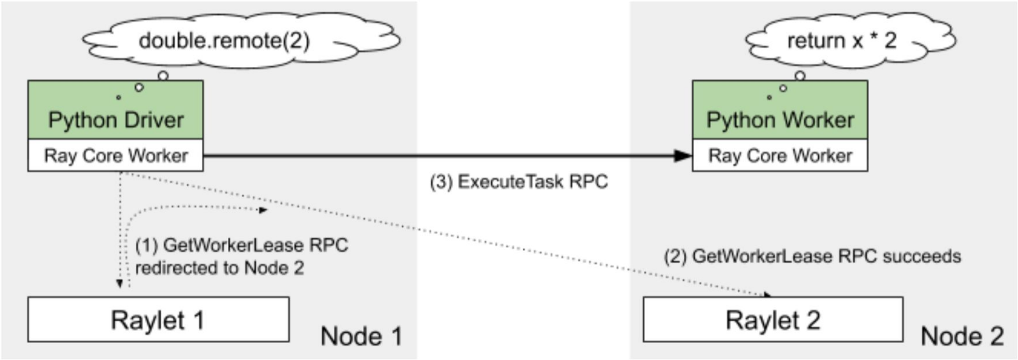


Components in green boxes represent Python code. Components in white boxes are part of the Ray common runtime written in C++. Joined boxes represent a process. Any Python driver or worker can call into the Ray C++ core worker library to execute further tasks. In this figure, all processes are running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

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.

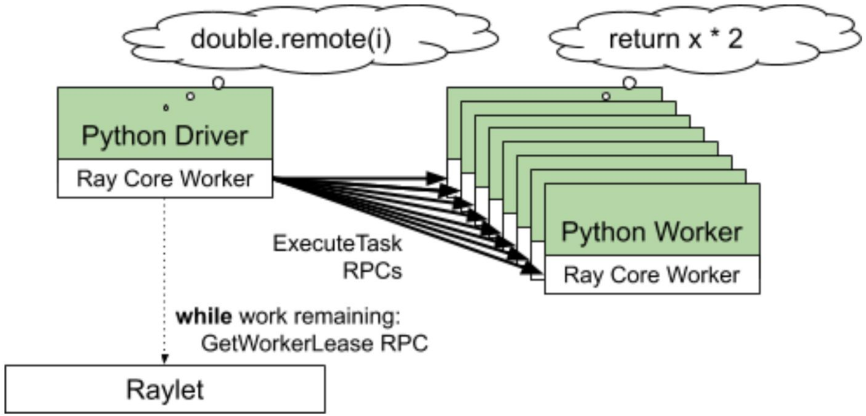


Tasks are sent to remote workers if there are no local resources available, transparently scaling Ray applications out to multiple nodes.

anyscale

Caching Scheduling Decisions

```
futures = [double.remote(i)
for i in range(10000)]
ray.get(futures)

# [0, 2, 4, 6 ...]
```



Once a scheduling decision is made by the Raylet, the worker returned can be reused for other tasks with the same resource requirements and input dependencies. This amortizes scheduling RPC overhead when executing many similar tasks. To avoid unfair monopolization of workers when there are multiple processes trying to submit tasks, callers are only allowed to reuse workers within a few hundred milliseconds of initial grant.

Ray Ecosystem

- Ray Tune
- XGBoost-Ray

anyscale

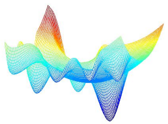
Ray Tune

anyscale

Ray Tune - For distributed HPO

- Efficient algorithms that enable running trials in parallel
- Effective orchestration of distributed trials
- Easy to use APIs
- Interoperable with Ray Train and Ray Datasets
- Saves cost (early stopping bad trials)

Cutting edge optimization algorithms



Compatible with ML ecosystem

```
tune.run(train_model)
```

Minimal code changes to work in distributed settings

- Single Process
- Multi-process/ Multi-GPU
- Multi-Node

Ray tune

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

https://docs.ray.io/en/latest/tune/api_docs/suggestion.html#tune-search-alg

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.

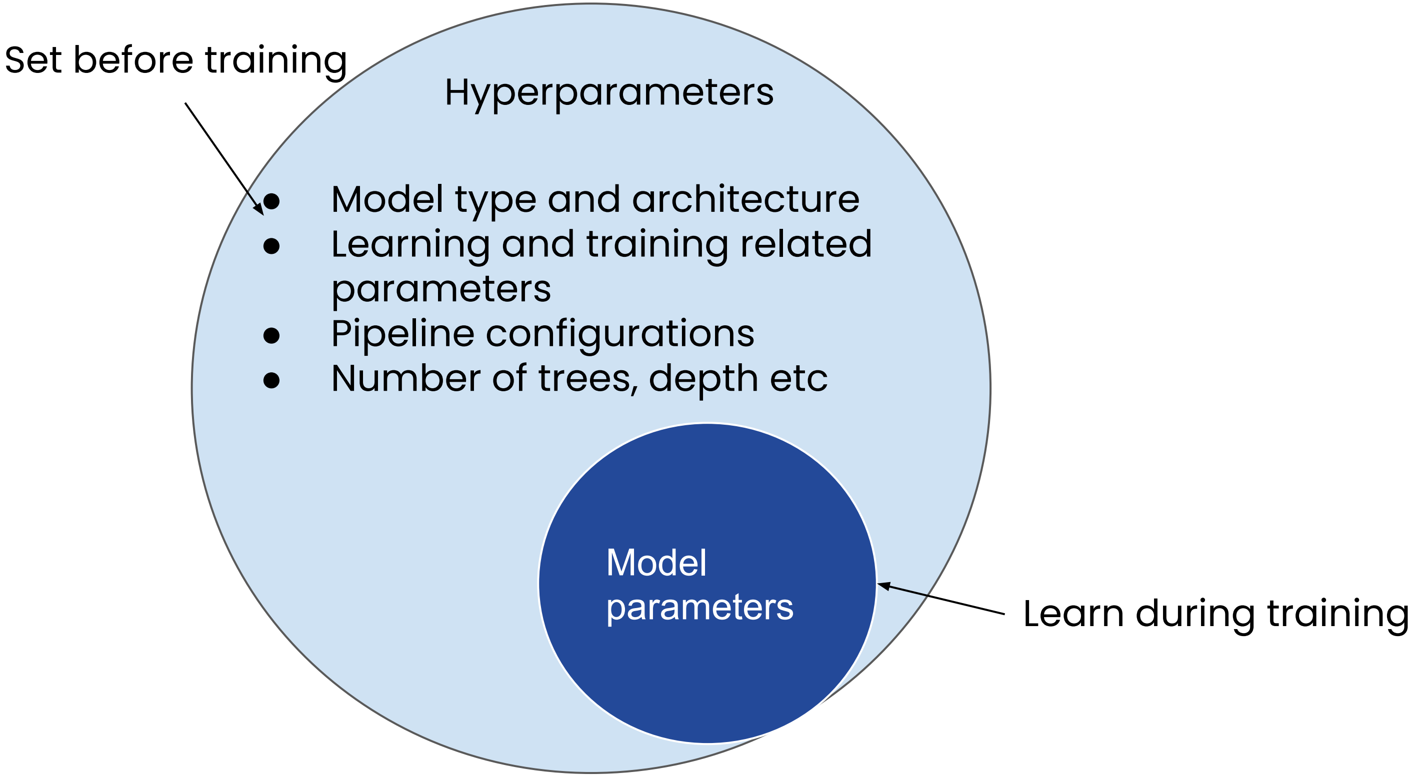
Schedulers can dynamically change trial resource requirements during tuning. This is currently implemented in `ResourceChangingScheduler`, which can wrap around any other scheduler.

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

https://docs.ray.io/en/latest/tune/api_docs/schedulers.html#tune-schedulers

anyscale

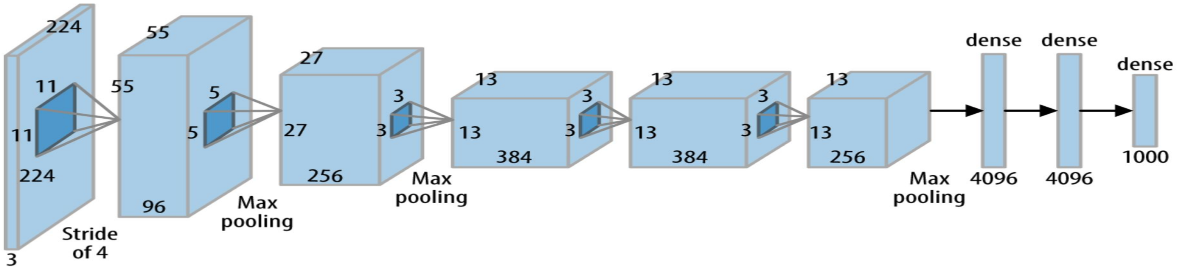
Hyperparameters



anyscale

Hyperparameter tuning

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



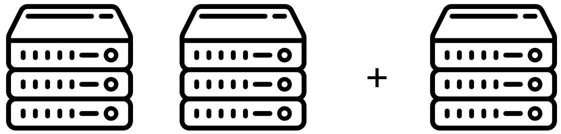
<u>Example</u>: what network structure is best for your binary classification problem?

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

HPO Challenges at scale

- Time consuming and costly
- Use Resources (GPUs/CPUs) at lower costs
- Fault-tolerance and elasticity

$$$



anyscale

Ray Tune - HPO algorithms

- Over 15+ algorithms natively provided or integrated
- Easy to swap out different algorithms with no code change

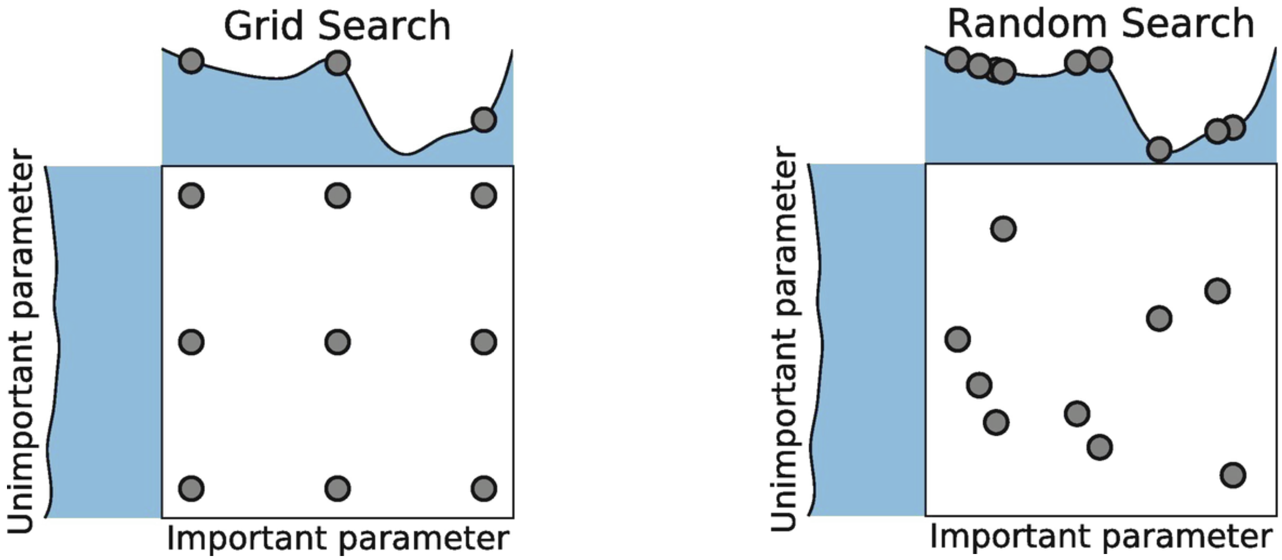
01 **Exhaustive Search**

02 **Bayesian Optimization**

03 **Advanced Scheduling**

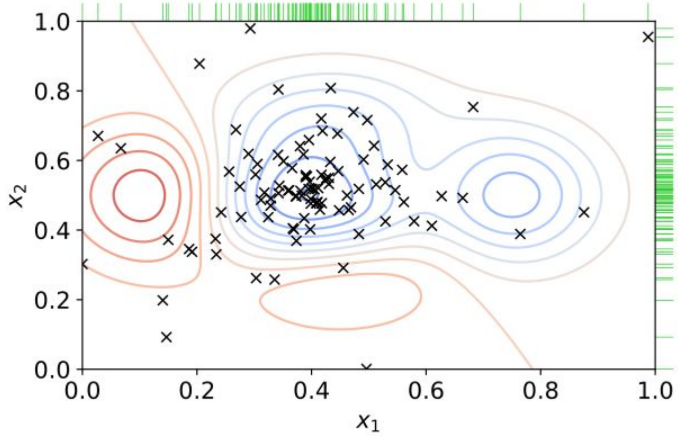
Exhaustive Search

- Easily parallelizable, easy to implement
- Inefficient, compute intensive



Bayesian optimization

- Uses results from previous combinations (trials) to decide which trial to try next
- Inherently sequential
- Popular libraries:
  - hyperopt
  - Optuna
  - Scikit-optimize
  - Nevergrad

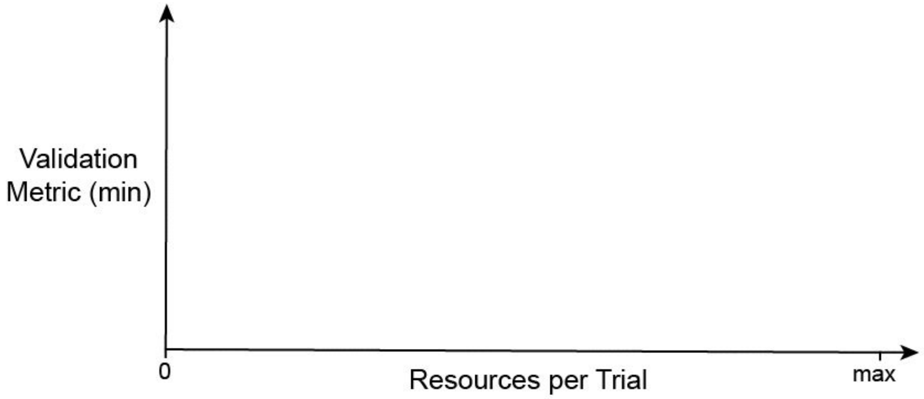


https://www.wikiwand.com/en/Hyperparameter_optimization

anyscale

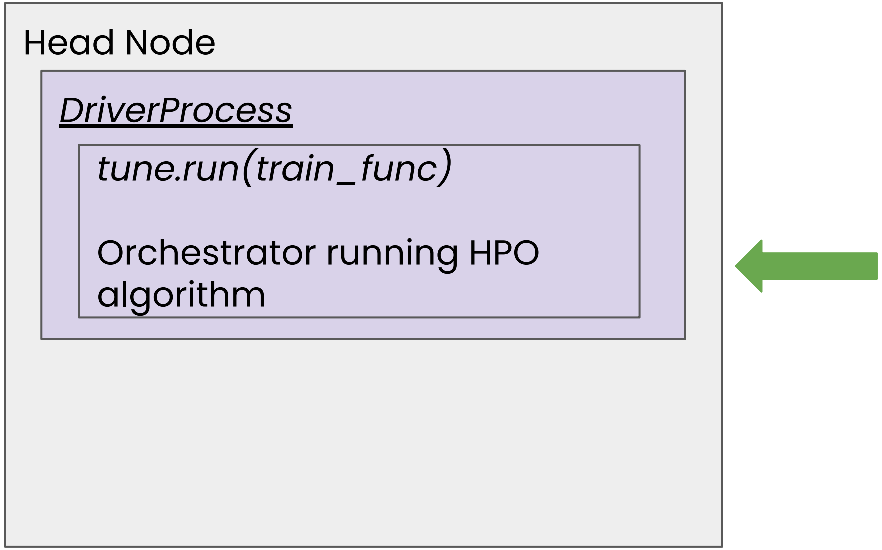
Advanced Scheduling - Early stopping

- Fan out parallel trials during the initial exploration phase
- Use intermediate results (epochs, trees, samples) to prune underperforming trials, saving time and computing resources
- Median stopping, ASHA/Hyperband
- Can be combined with Bayesian Optimization (BOHB)



anyscale

Ray Tune - *distributed* HPO




```
from ray import tune

def train_func(config):
    model = ConvNet(config)
    for i in range(epochs):
        current_loss = model.train()
        tune.report(loss=current_loss)

tune.run(
    train_func,
    config={“alpha”: tune.uniform(0.001, 0.1)},
    num_samples=100,
    scheduler=“asha”,
    search_alg=”optuna”)
```

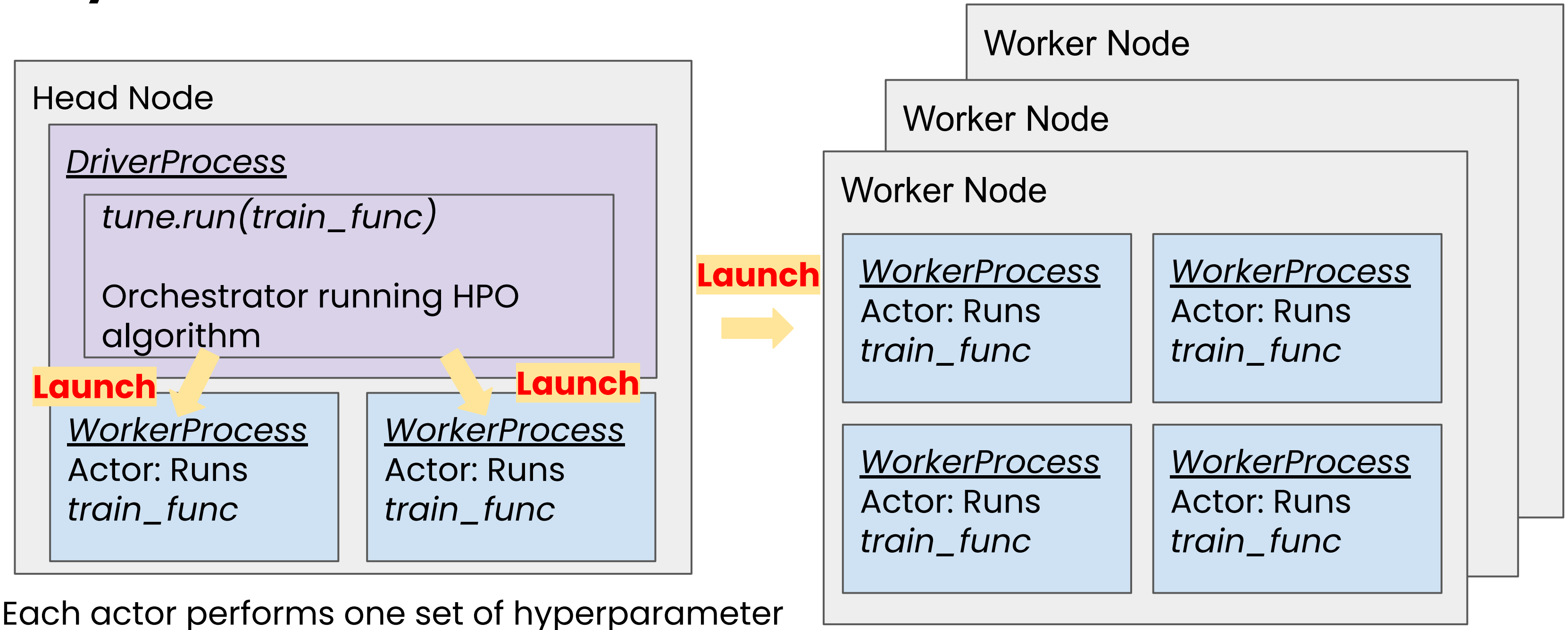
Easily define your training function

Just use tun.run(..)

Easily specify hyperparameter ranges to search over

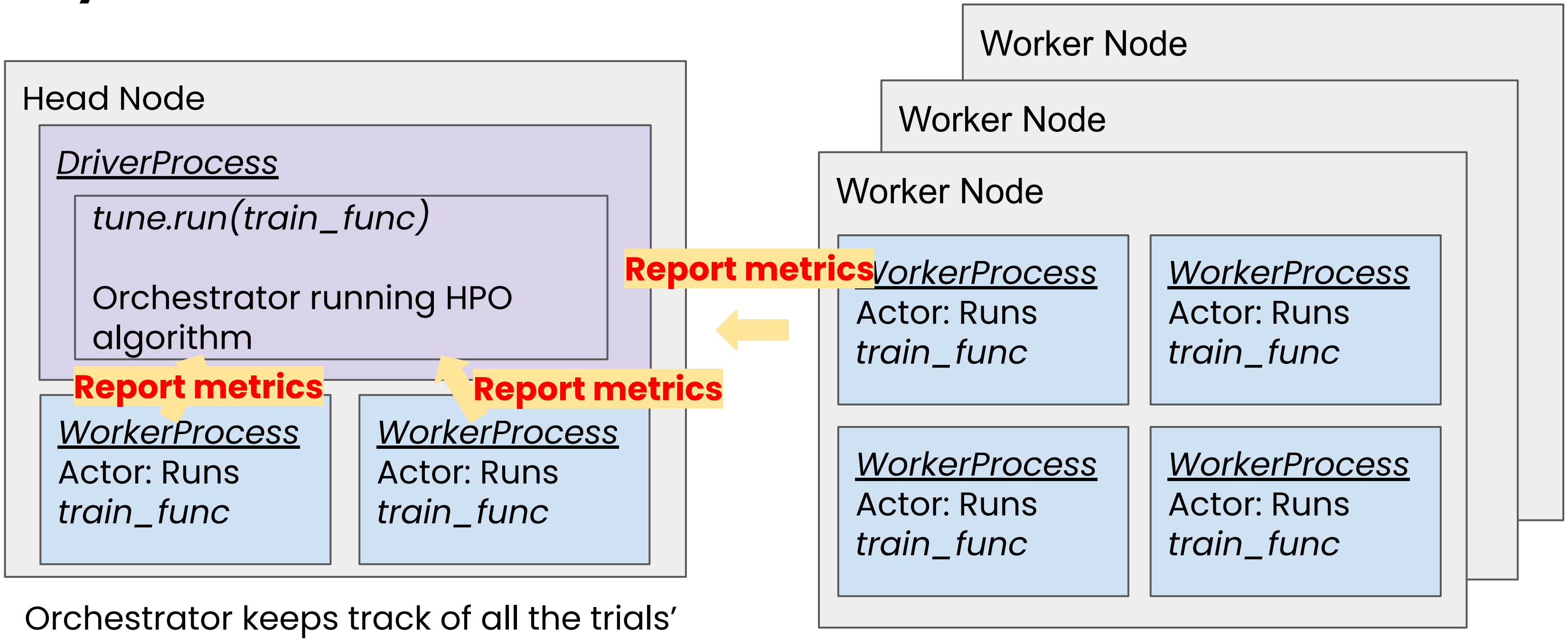
anyscale

Ray Tune - *distributed* HPO



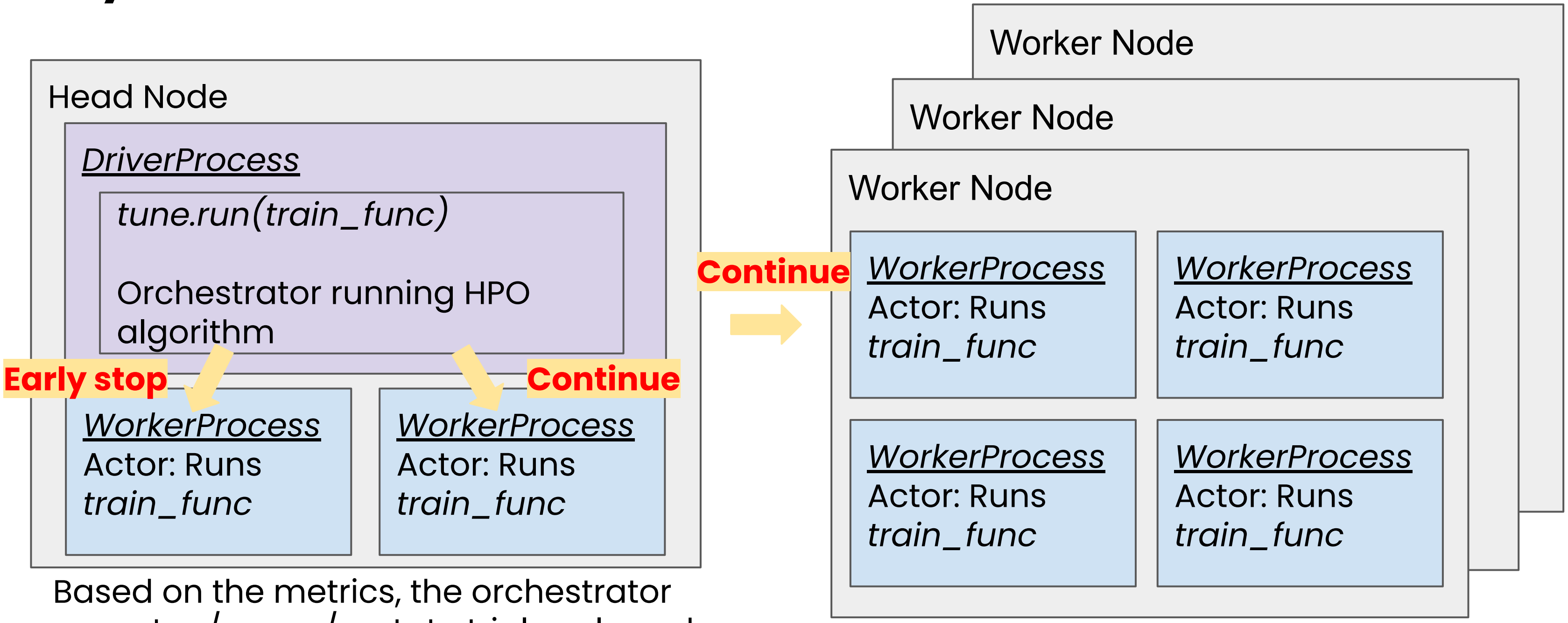
Each actor performs one set of hyperparameter combination evaluation (a trial)

Ray Tune - *distributed* HPO



Orchestrator keeps track of all the trials' progress and metrics.

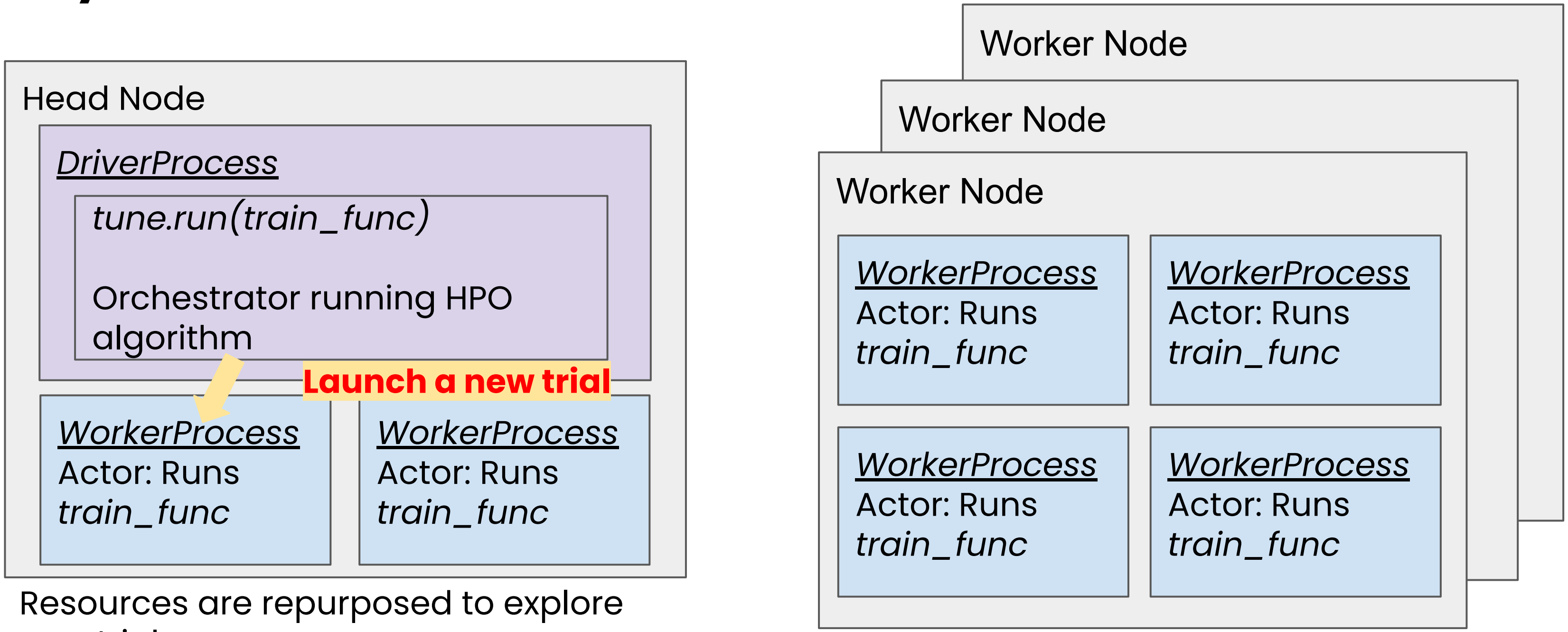
Ray Tune - *distributed* HPO



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

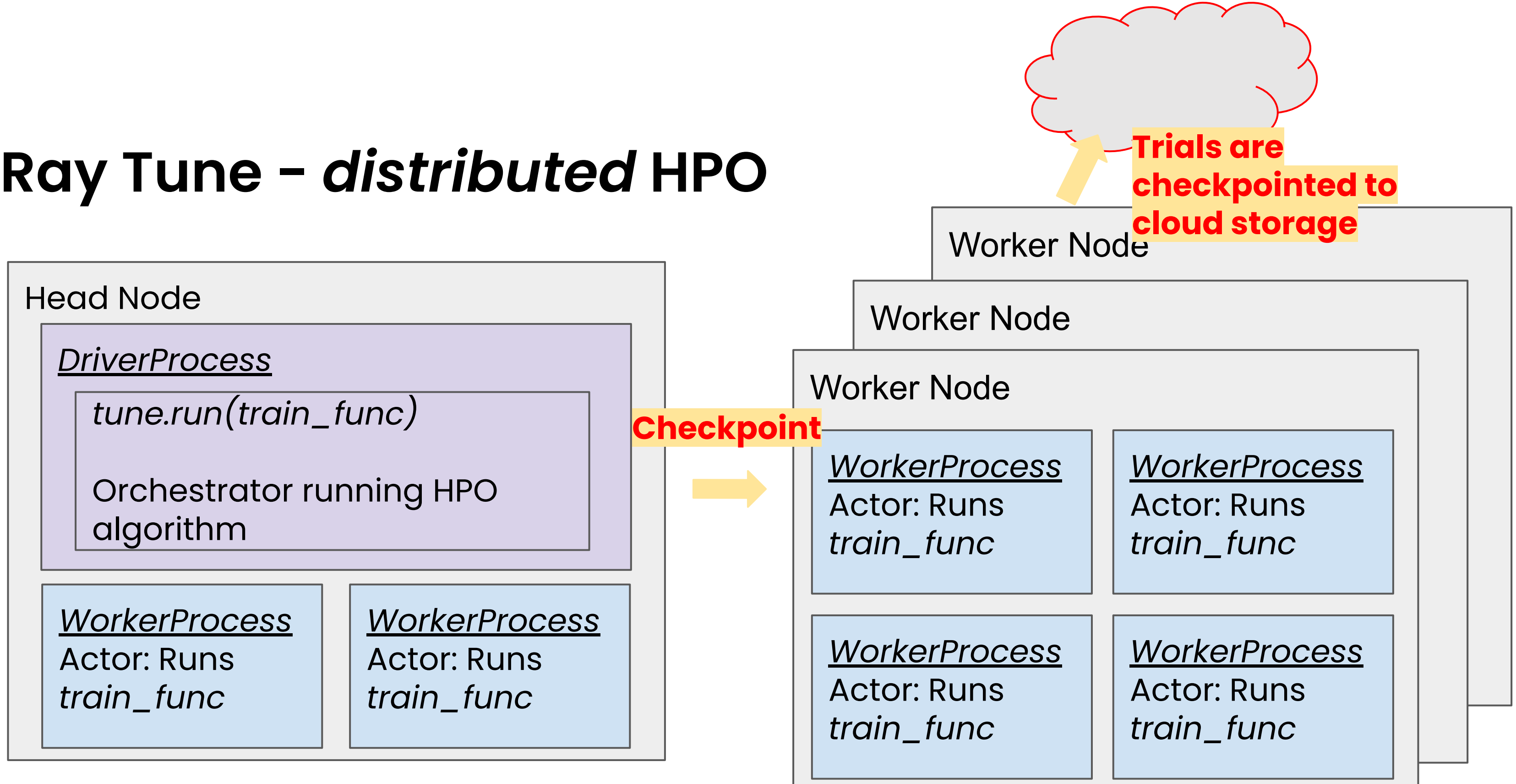
anyscale

Ray Tune - *distributed* HPO



Resources are repurposed to explore new trials.

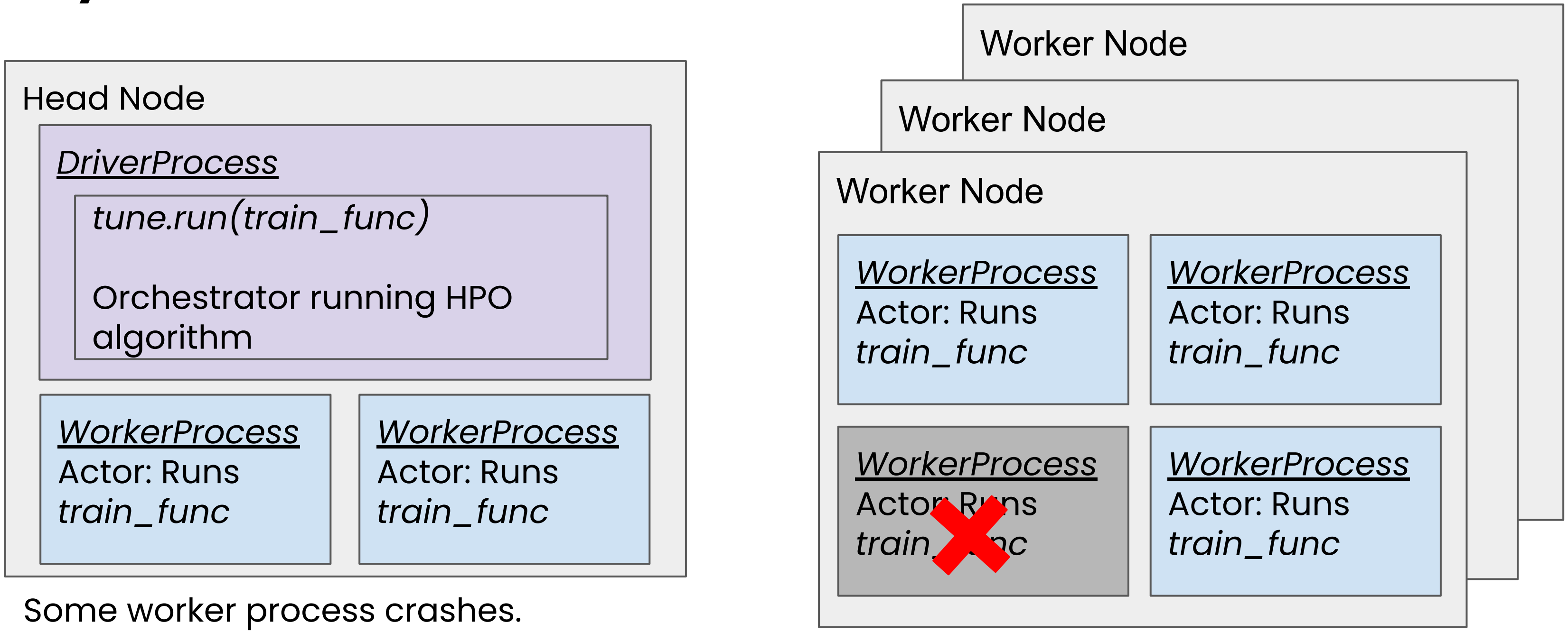
Ray Tune - *distributed* HPO



Orchestrator also manages checkpoint state.

anyscale

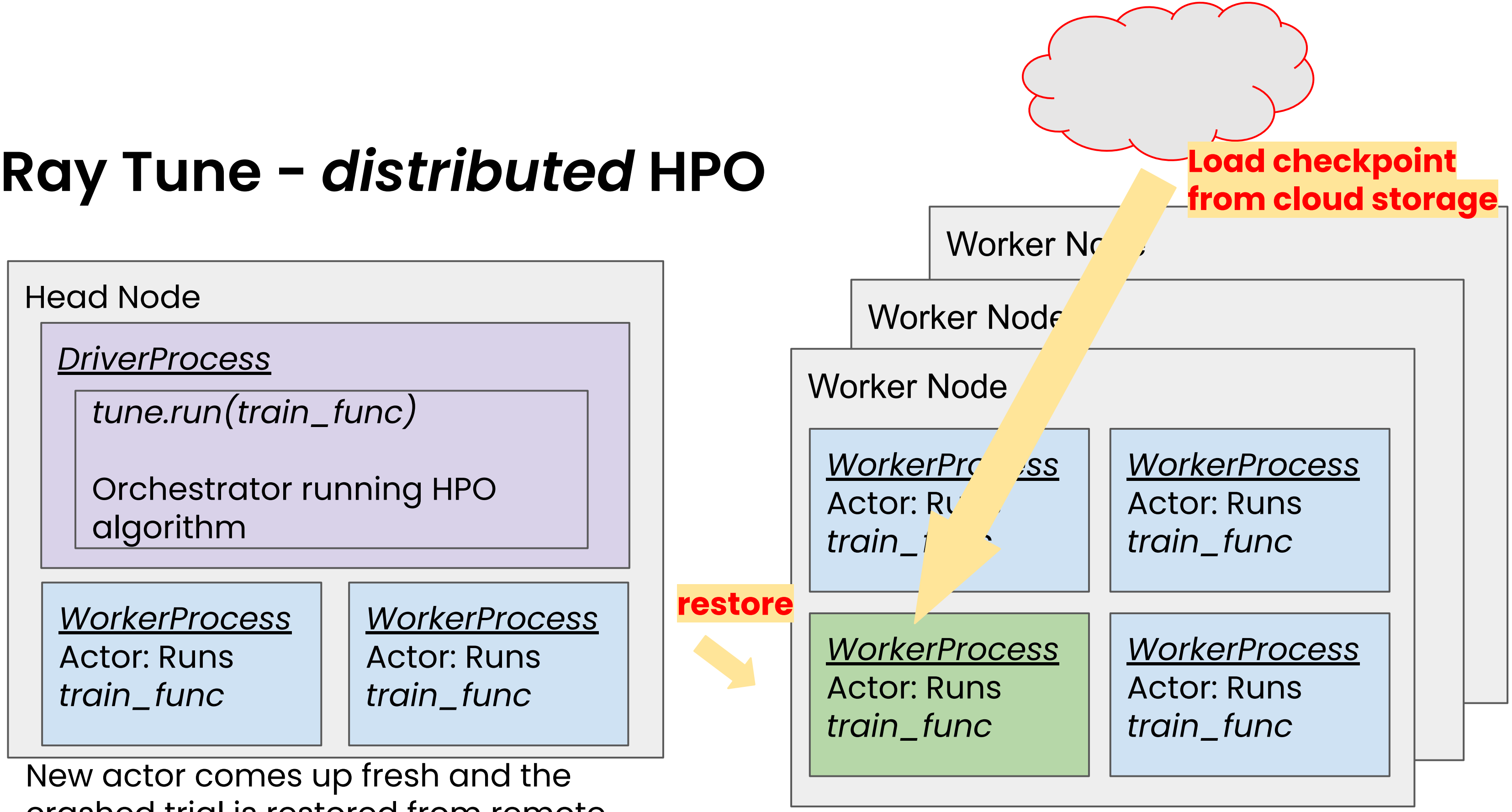
Ray Tune - *distributed* HPO



Some worker process crashes.

anyscale

Ray Tune - *distributed* HPO



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

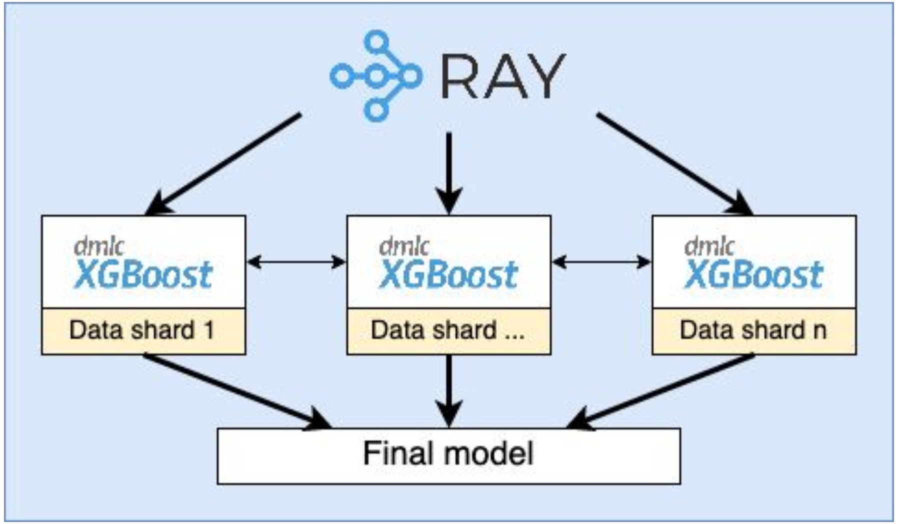
XGBoost-Ray

- Design & Features

anyscale

XGBoost-Ray

- Distributed XGBoost-Ray - Drop-in replacement for XGBoost
- Fault tolerance & Elastic training
- Integration with Ray Datasets and Ray Tune



- https://github.com/ray-project/xgboost_ray
- https://docs.ray.io/en/latest/xgboost-ray.html

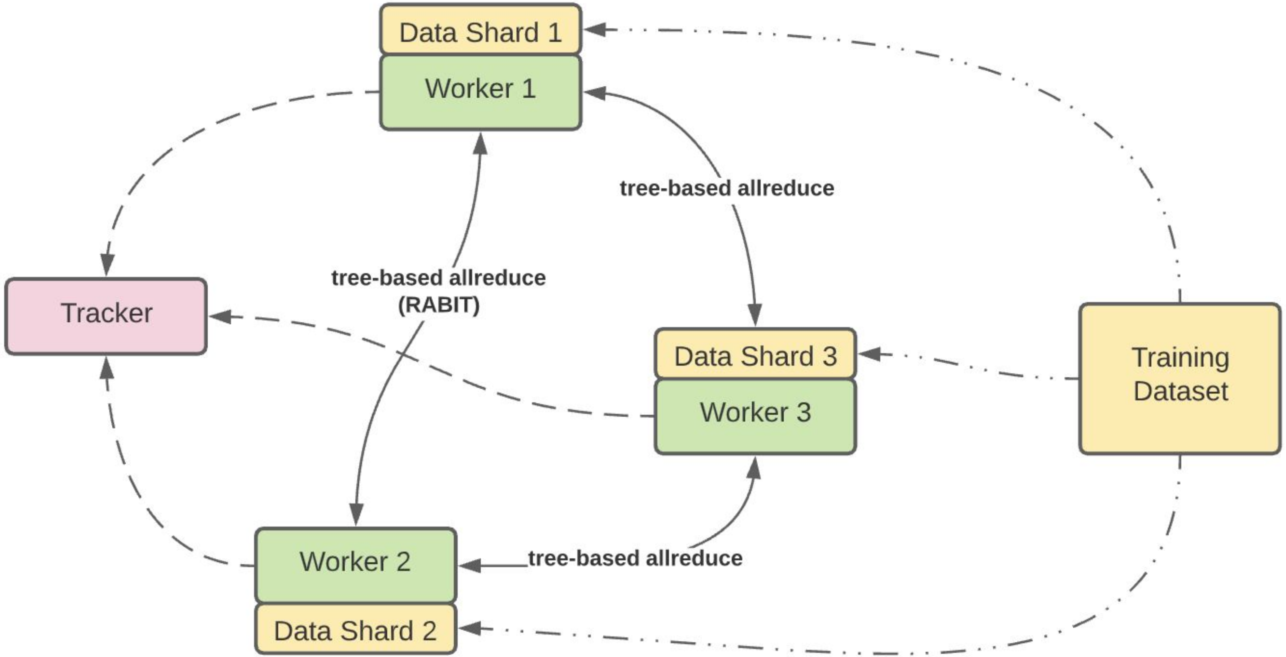
Motivation

- There are existing solutions for distributed XGBoost
  - E.g. Apache Spark, Dask, Kubernetes etc
- But most existing solutions have shortcomings:
  - Dynamic computation graphs
  - Fault tolerance handling
  - GPU support
  - Integration with hyperparameter tuning libraries

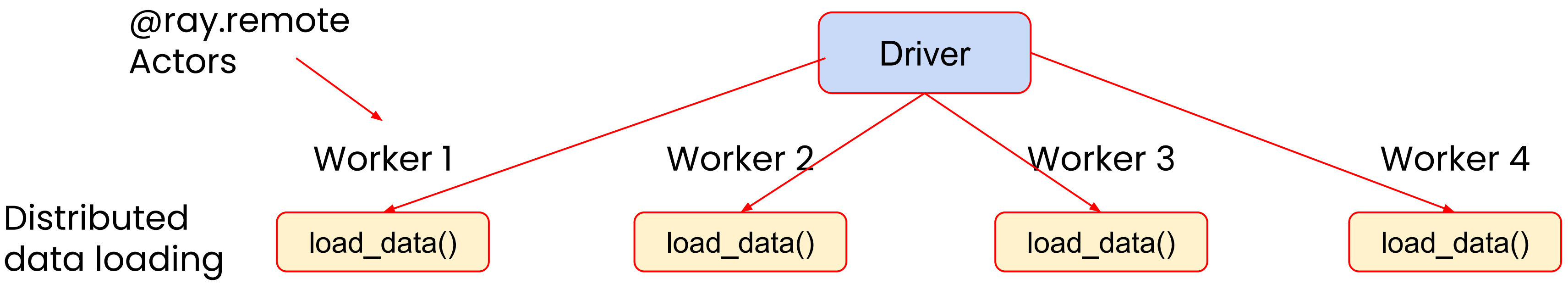
XGBoost-Ray

- Ray actors for stateful training workers
- Advanced fault tolerance mechanisms
- Full (multi) GPU support
- Locality-aware distributed data loading
- Integration with Ray Tune

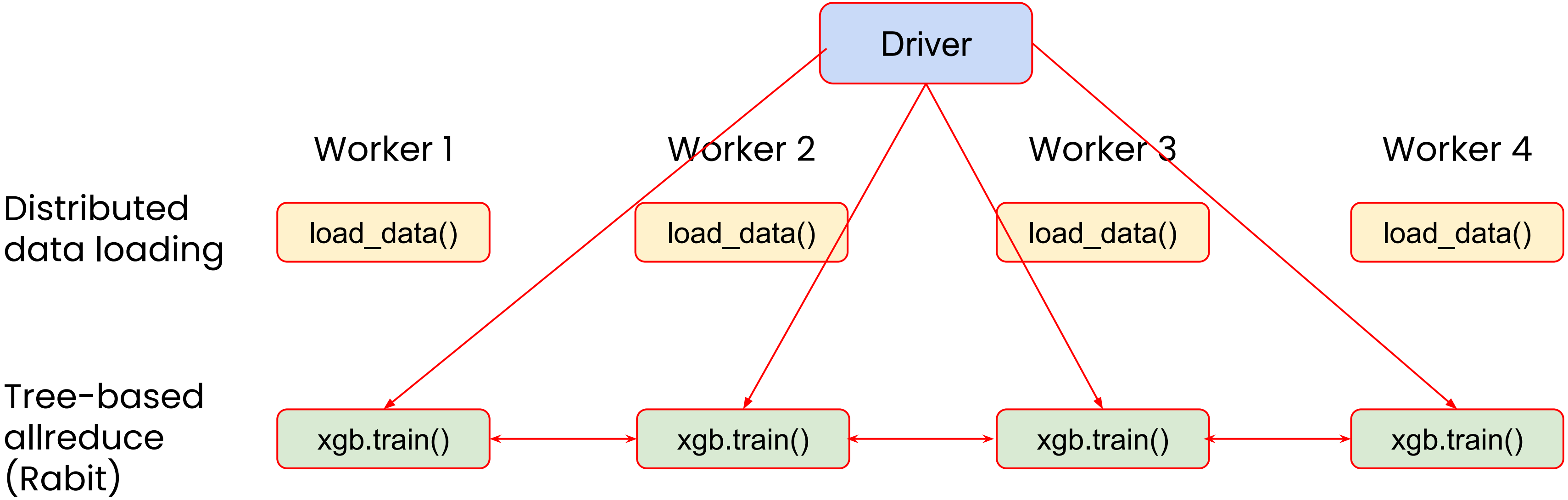
Distributed XGBoost Architecture



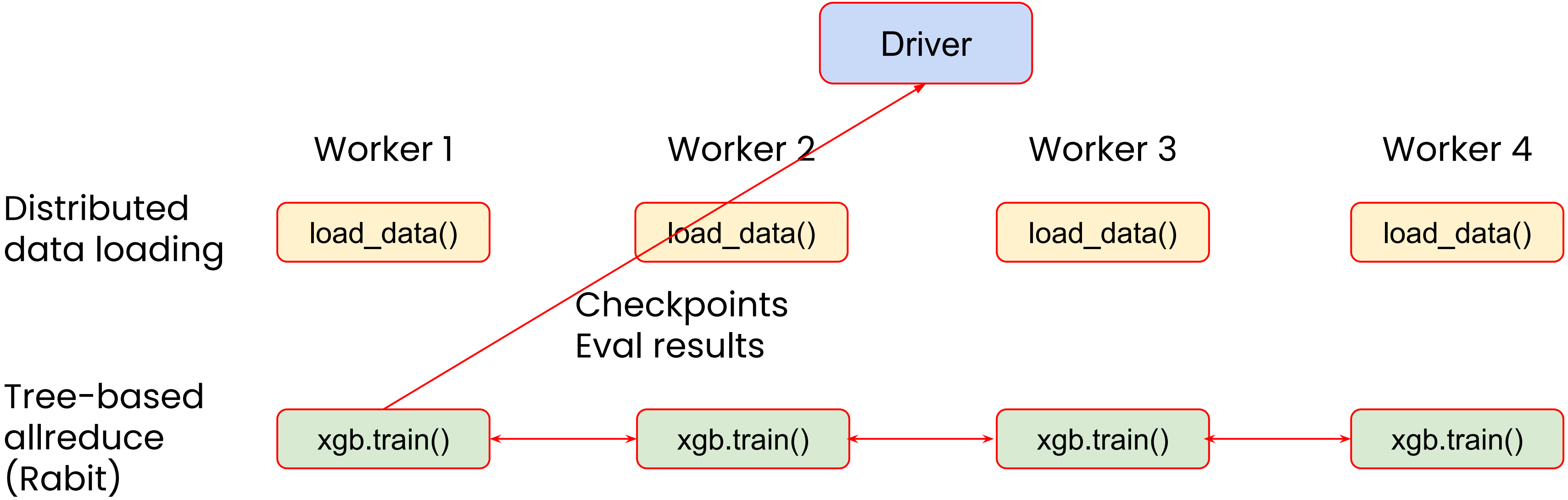
Architecture



Architecture

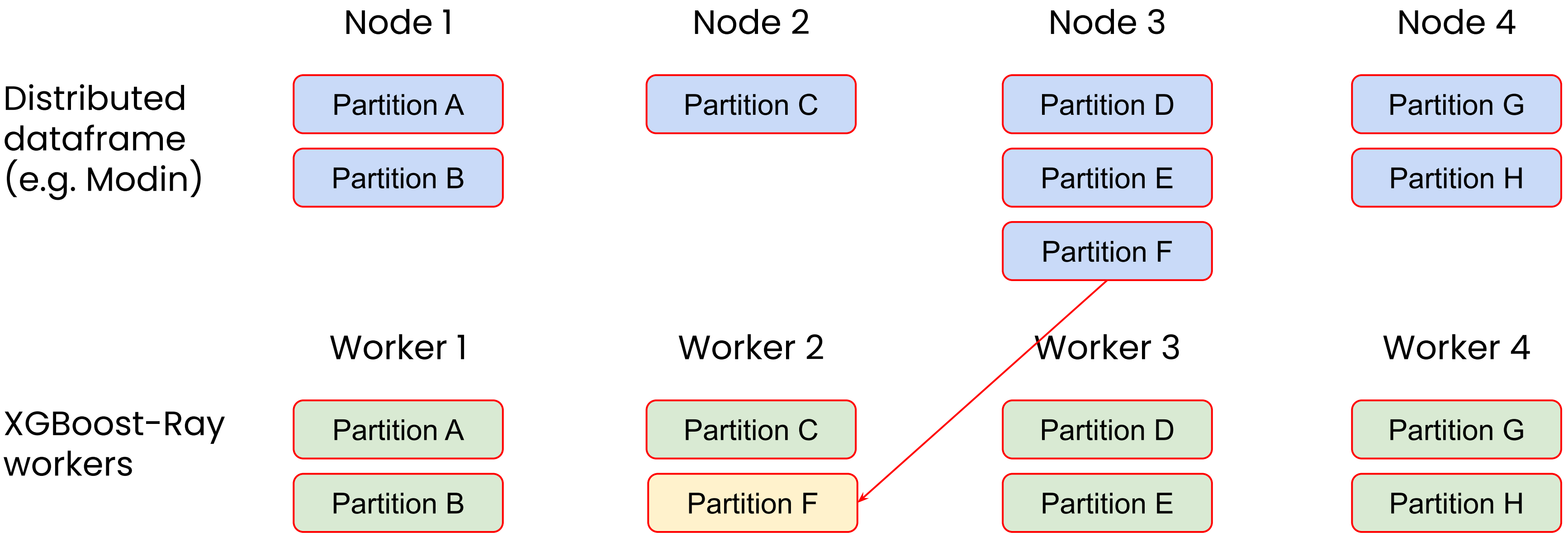


Architecture



anyscale

Distributed data loading

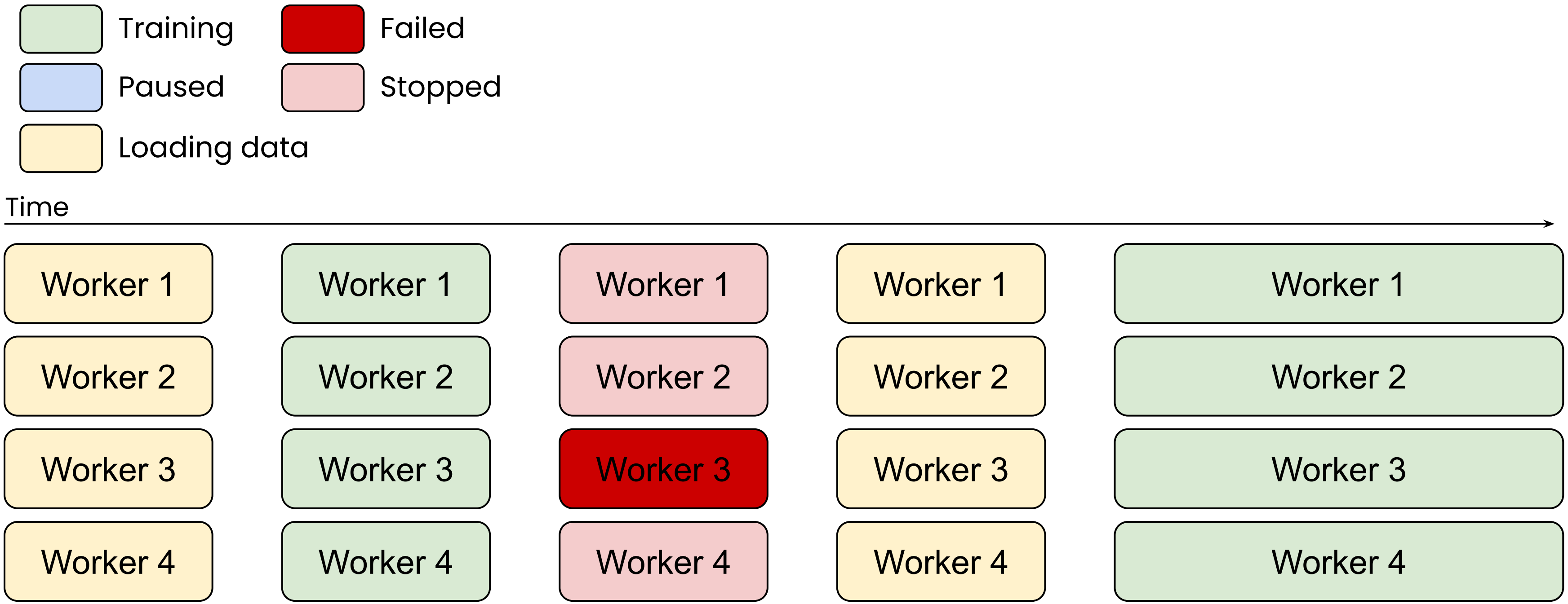


anyscale

Fault tolerance strategies

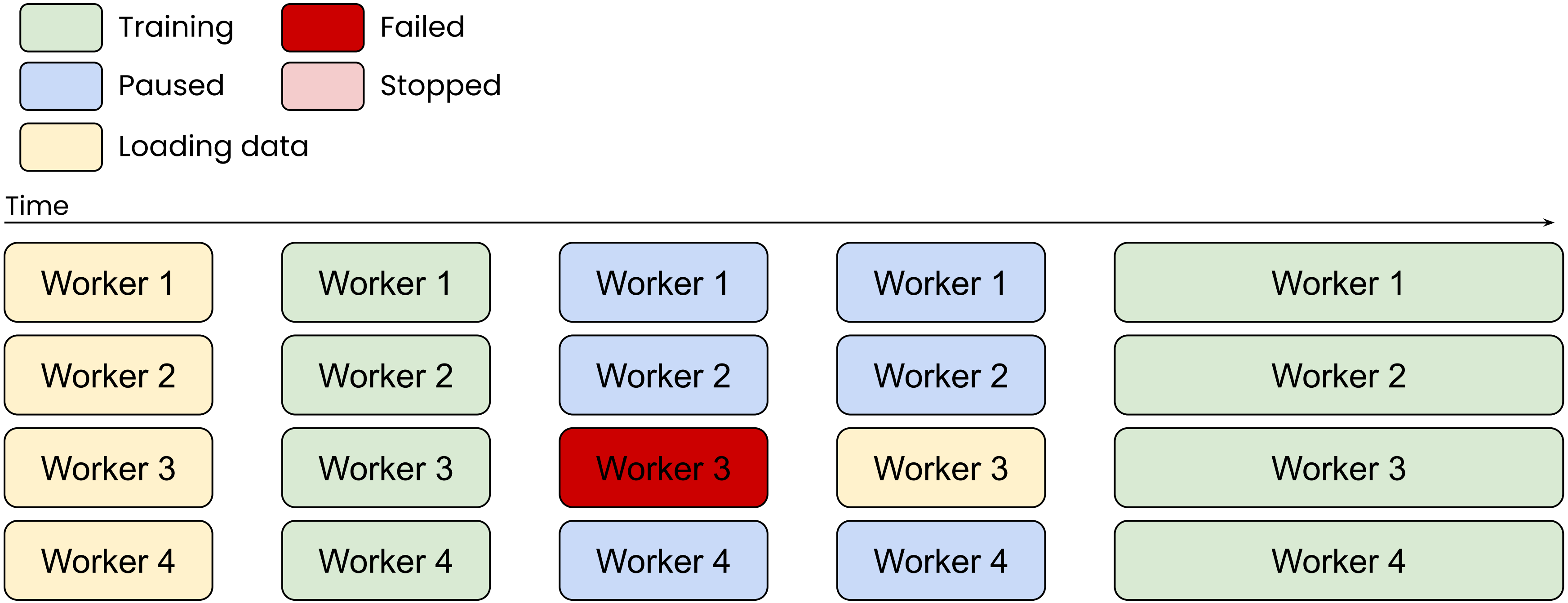
- In distributed training, some worker nodes are bound to fail eventually
- **Default**: Simple (cold) restart from last checkpoint
- **Non-elastic** training (warm restart):
  Only failing worker restarts
- **Elastic training:** Continue training with fewer workers until failed actor is back

Fault tolerance: Simple (cold) restart



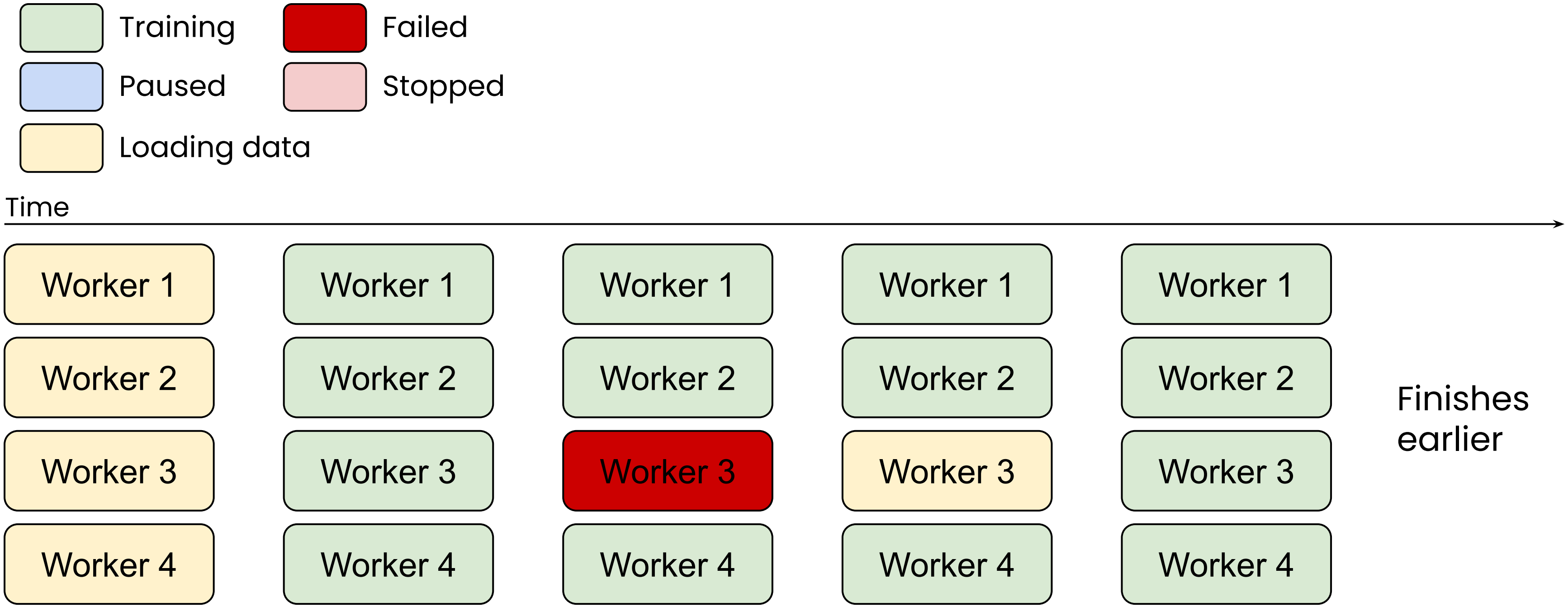
anyscale

Fault tolerance: Non-elastic training (warm restart)



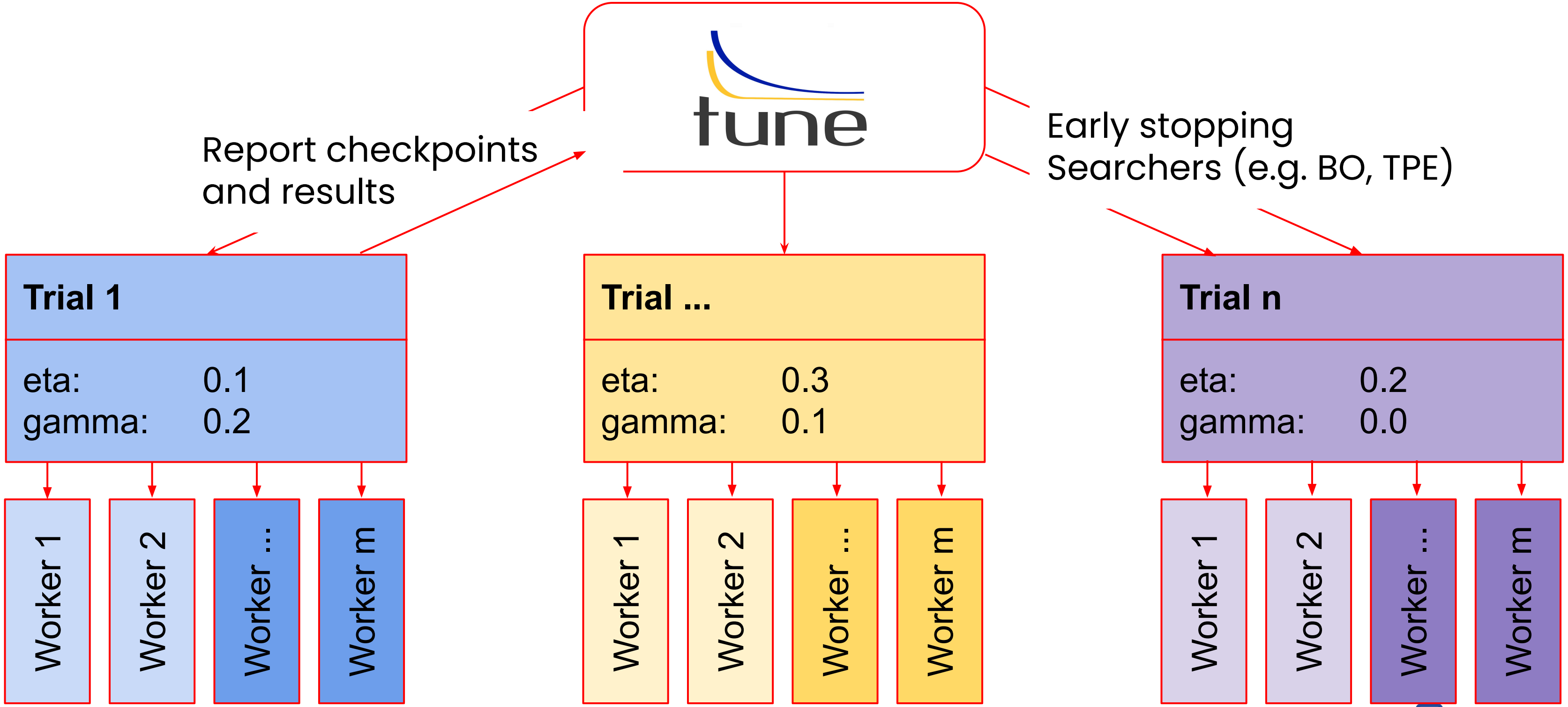
anyscale

Fault tolerance: Elastic training



anyscale

Hyperparameter tuning



anyscale

Simple API example

```python
from sklearn.datasets import load_breast_cancer
from xgboost_ray import RayDMatrix, RayParams, train

train_x, train_y = load_breast_cancer(return_X_y=True)
train_set = RayDMatrix(train_x, train_y)

bst = train(
    {"objective": "binary:logistic"},
    train_set,
    ray_params=RayParams(num_actors=2)
)
bst.save_model("trained.xgb")
```

anyscale

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

MARCH 29 - VIRTUAL - FREE

Production RL Summit

A reinforcement learning event for practitioners

Register: https://tinyurl.com/mr9rd32h

ORGANIZED BY anyscale

Sergey Levine

Berkeley
UNIVERSITY OF CALIFORNIA

Ben Kasper

RIOT GAMES

Sumitra Ganesh

J.P.Morgan

Adam Kelloway

DOW

Marc Weber

SIEMENS

Volkmar Sterzing

SIEMENS

MARCH 29 - VIRTUAL

Production RL Summit

A reinforcement learning event for practitioners

Register: https://tinyurl.com/mr9rd32h

~~$75~~ **$30**
Use code **DCRL2022**

ORGANIZED BY anyscale

HANDS-ON TUTORIAL

Contextual Bandits & RL with RLlib

Learn how to apply cutting edge RL in production with RLlib.

Tutorial covers:

- Brief overview of RL concepts.
- Train and tune contextual bandits and SlateQ algorithm
- Offline RL using cutting-edge algos
- Deploy RL models into a live service

~~$75~~ **$30** (use code **DCRL2022**)

Instructor:

Sven Mika, Lead maintainer, RLlib

RAY SUMMIT 2021 • JUNE 22 - 24

RAY SUMMIT 2022

AUGUST 23 & 24TH | SAN FRANCISCO

Call for Papers is Now Open!

Submit your talk at anyscale.com/ray-summit-2022

DON'T WAIT!
CFP closes April 11th

anyscale

Start learning Ray and contributing …

Getting Started: `pip install ray`

Documentation (docs.ray.io)
Quick start example, reference guides, etc

Join Ray Meetup
Revived in Jan 2022. Next meetup March 2nd.
Meetup each month and publish recording to the members
https://www.meetup.com/Bay-Area-Ray-Meetup/

Forums (discuss.ray.io)
Learn / share with broader Ray community, including core team

Ray Slack
Connect with the Ray team and community

Social Media (@raydistrtibuted, @anyscalecompute)
Follow us on Twitter and linkedIn

GitHub
*Check out sources, file an issue, become a contributor, give us a **Star** :)*
https://github.com/ray-project/ray

Thank you!

Let's stay in touch:

jules@anyscale.com
https://www.linkedin.com/in/dmatrix/

@2twitme

anyscale

VIDEO

anyscale