

Overview of Ray

Kamil Kaczmarek

Emmy Li



Technical training team

Kamil



Emmy



Overview of Ray

Ray project

Key Ray characteristics

Ray libraries

Example use cases

Summary



Overview of Ray



Ray project

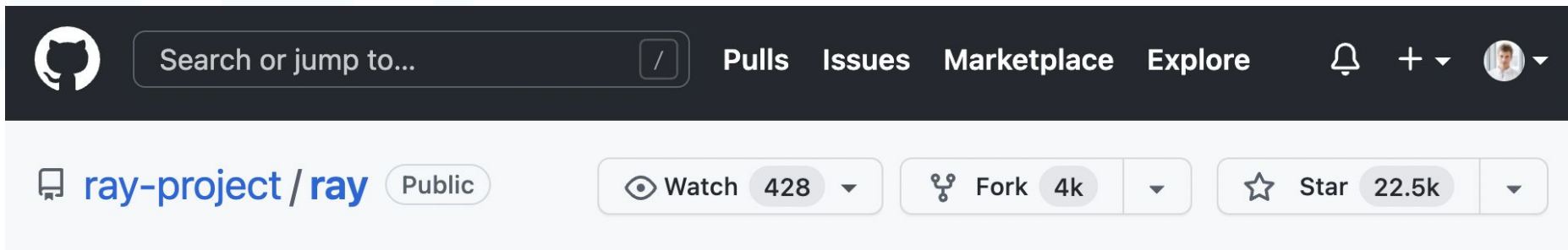
Key Ray characteristics

Ray libraries

Example use cases

Summary

Ray project

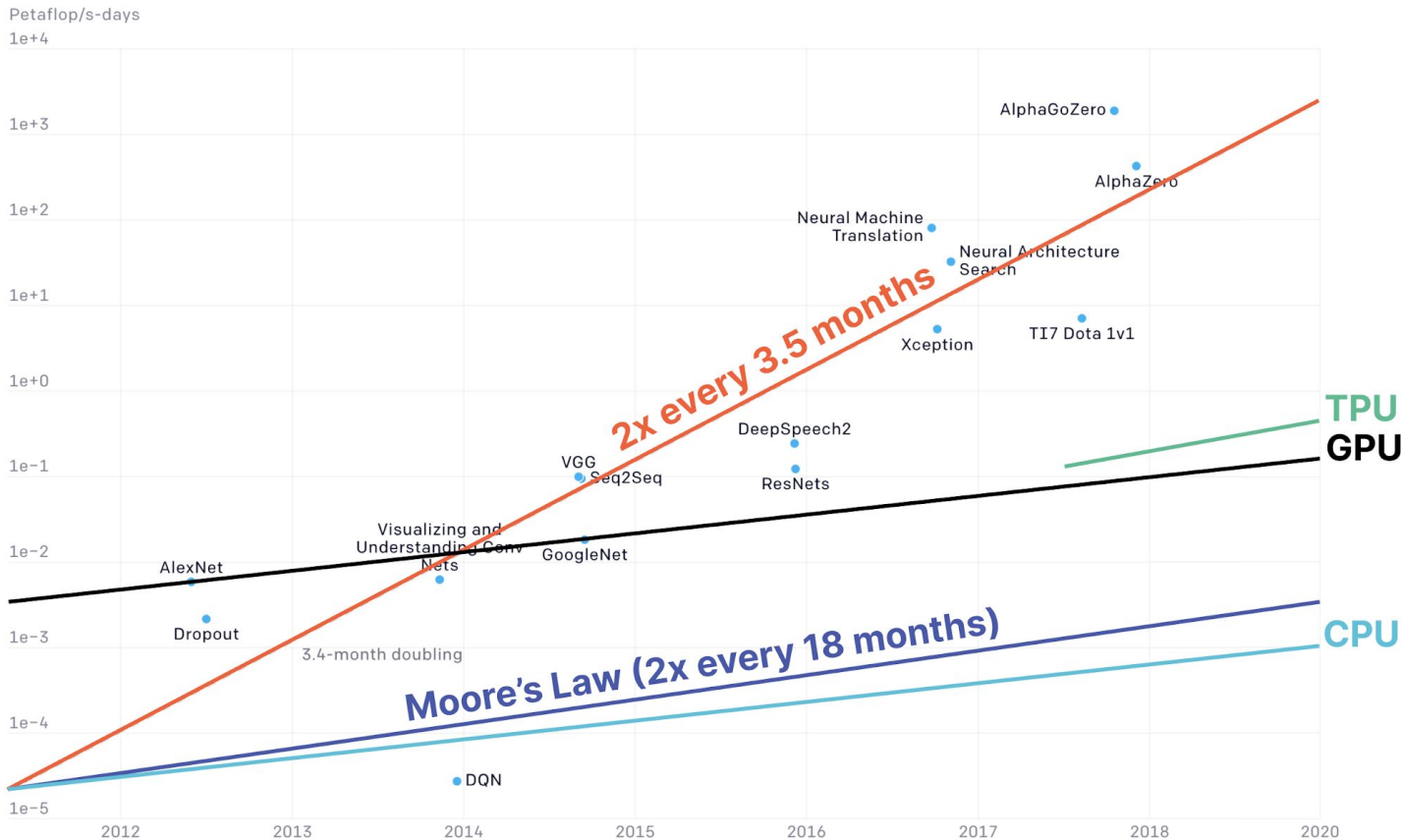


Ray is:
an open-source unified compute framework
that makes it easy to scale AI and Python workloads.

Ray handles
orchestration, scheduling, fault tolerance, auto-scaling and more
so that you can scale your apps without becoming a distributed
systems expert.



Distributed computing: a bit of context



Overview of Ray

Ray project



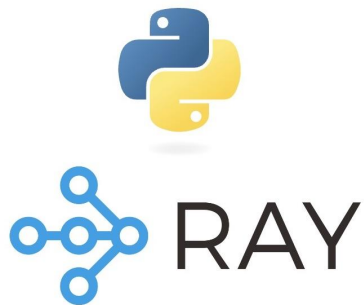
Key Ray characteristics

Ray libraries

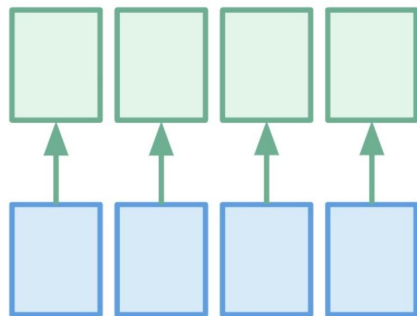
Example use cases

Summary

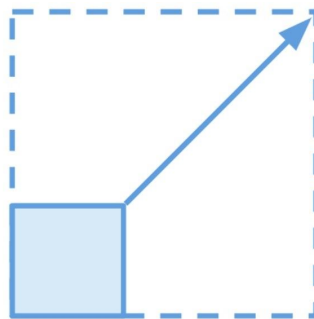
Key Ray characteristics



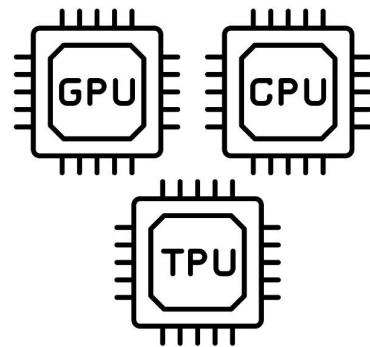
Python first approach



Simple and flexible API



Scalability



Support for heterogeneous
hardware

Python first approach



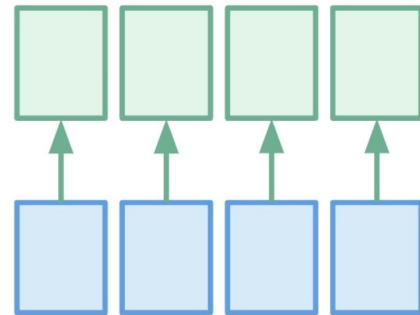
Ray Core, key abstractions:

- Tasks: remote, stateless Python functions
- Actors: remote, stateful Python classes
- Objects: tasks and actors create and compute on objects that can be stored and accessed anywhere in the cluster

Simple and flexible API

Ray Core is:

an open-source, Python, **general purpose**, distributed computing library that enables ML engineers and Python developers to scale Python applications and accelerate machine learning workloads.

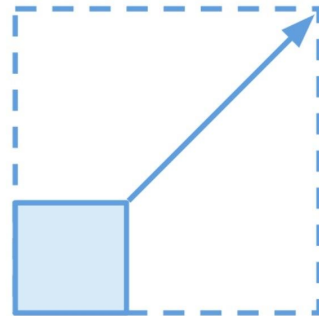


Ray AI Runtime (AIR) is:

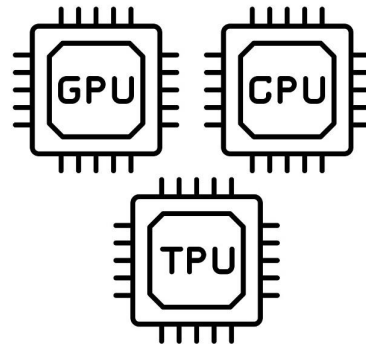
an open-source, Python, **domain-specific** set of libraries that equips ML engineers, data scientists, and researchers with a scalable and unified toolkit for ML applications.

Scalability

- Utilize large compute cluster
- Work with single, unified pool of resources
- Autoscaler (scale up and scale down)



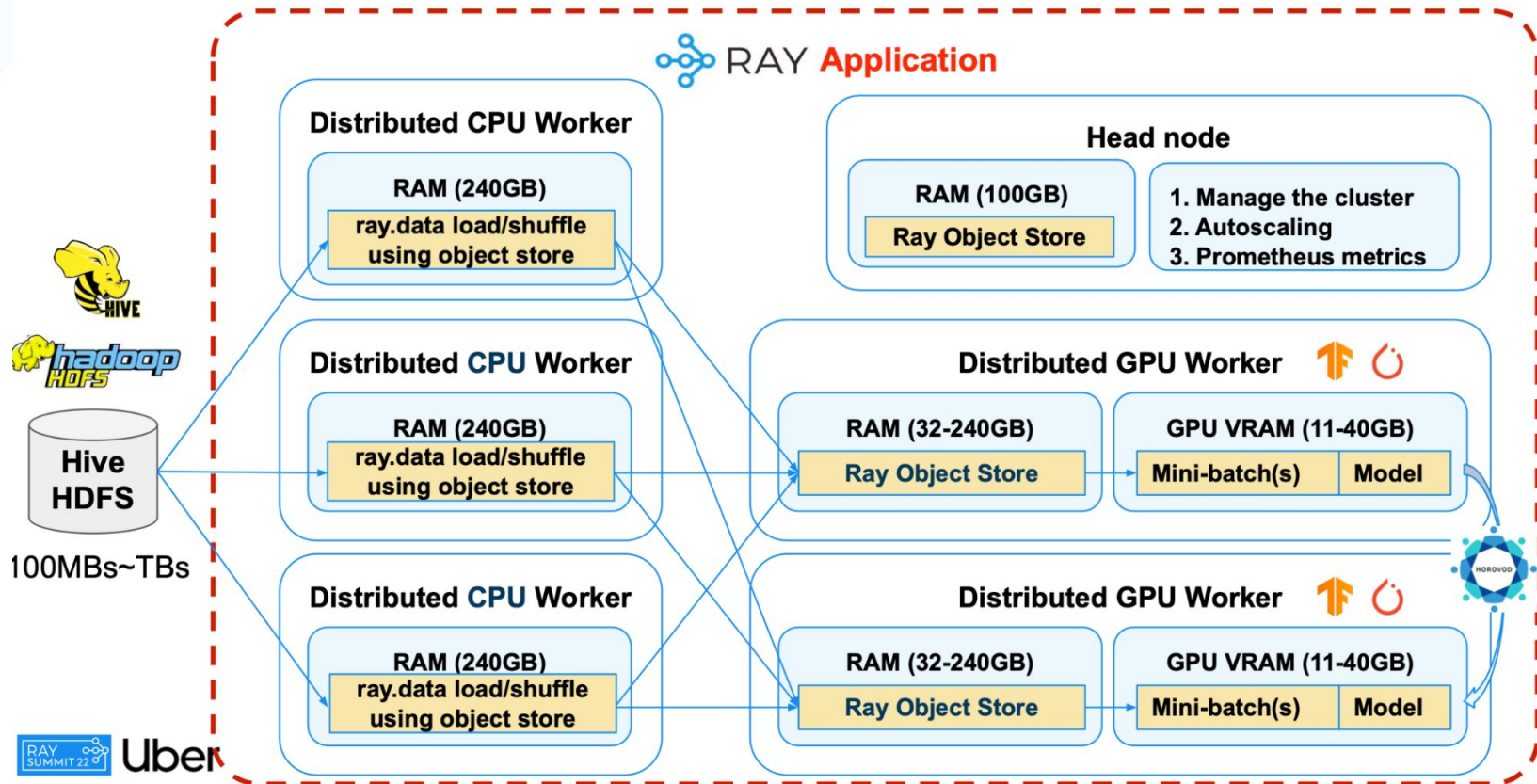
Support for heterogen. hardware



```
1 # specify resources when starting a cluster
2 ray.init(num_cpus=128)
3
4 # specify resources for a compute task
5 @ray.remote(num_cpus=36, num_gpus=8)
6 def train_and_score_model():
7     RoBERTa_pretrained = ...
8     ...
9     return score
10
11 # specify fractional GPUs
12 @ray.remote(num_gpus=0.5)
13 def tiny_ml():
14     EfficientNet = load()
15     ...
16     return acc
17
18 # specify custom resources
19 @ray.remote(resources={"custom": 1})
20 def train_optimized():
21     load_data = ...
22     ...
23     return model_val
```



Deep learning pipeline at Uber



Overview of Ray

Ray project

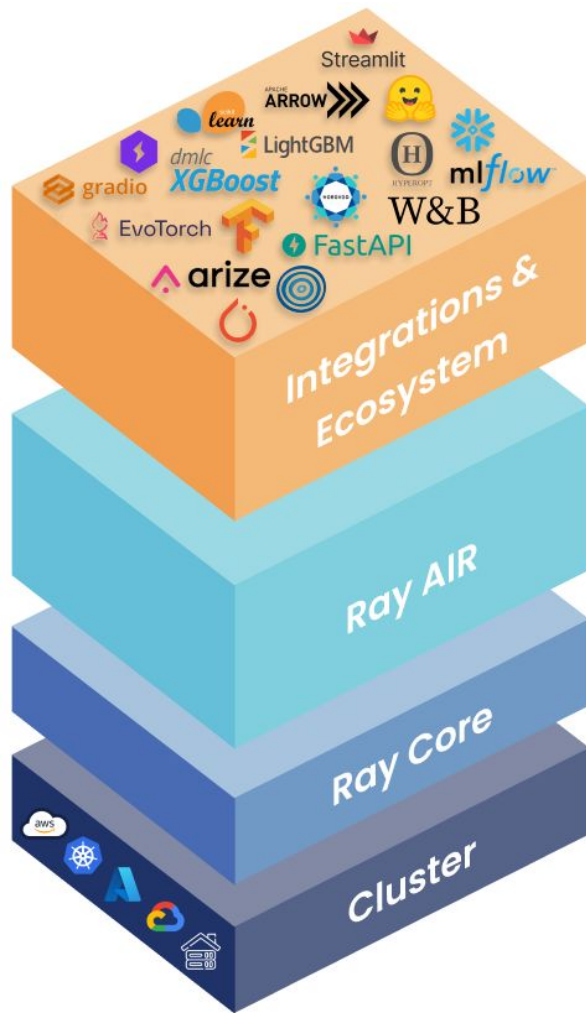
Key Ray characteristics

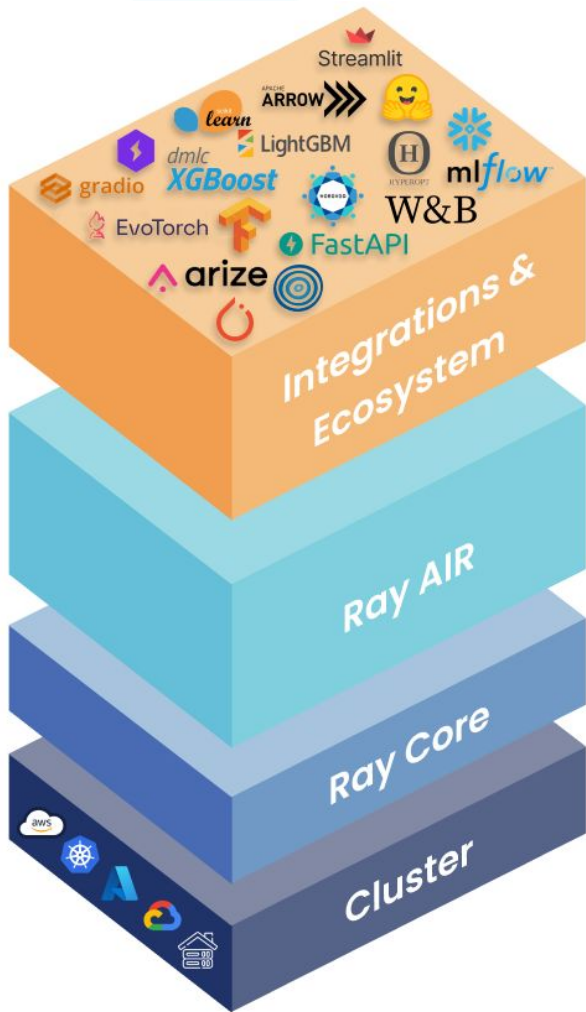
→ Ray libraries

Example use cases

Summary

Ray libraries





Growing ecosystem of **integrations** with popular **ML** and **MLOps** projects.

Ray AI Runtime (AIR) is:

an open-source, Python, **domain-specific** set of libraries that equips ML engineers, data scientists, and researchers with a scalable and unified toolkit for ML applications.

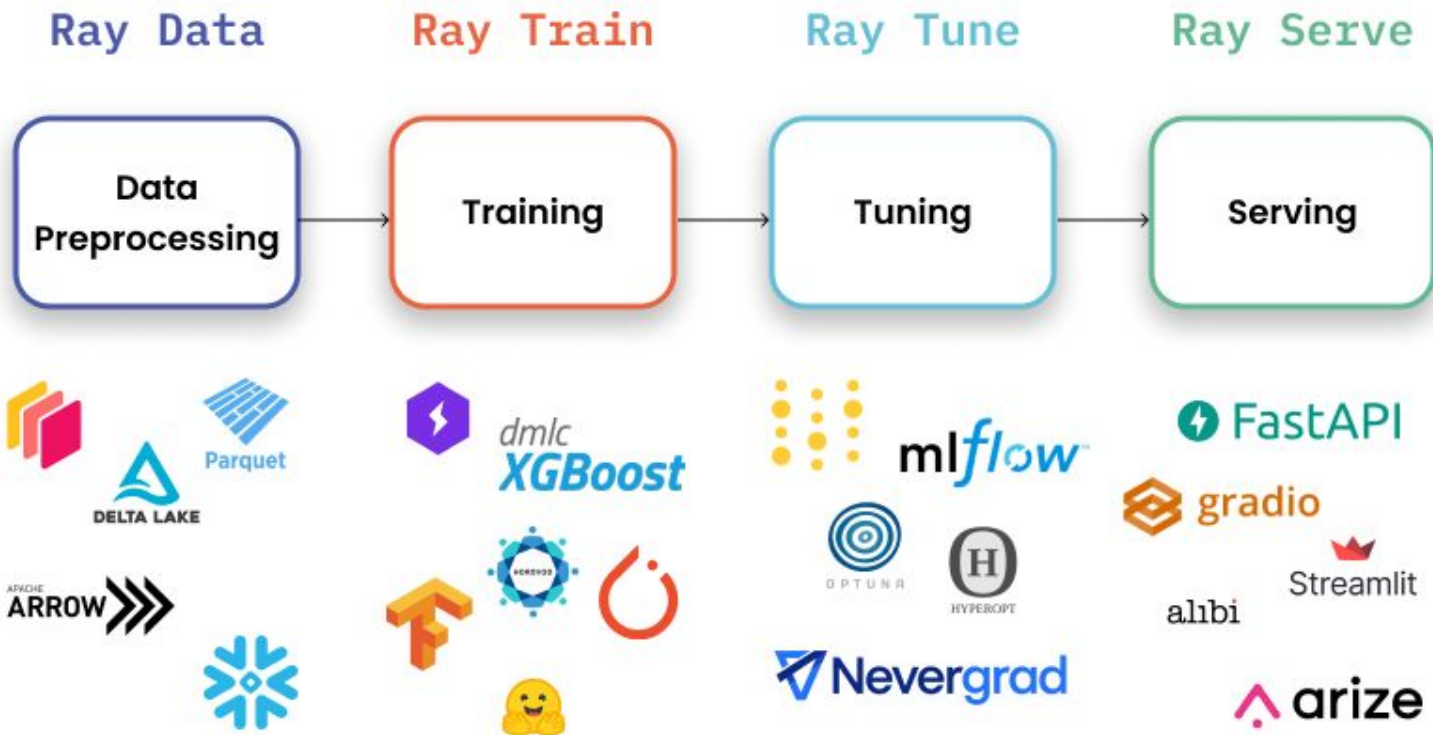
Ray Core is:

an open-source, Python, **general purpose**, distributed computing library that enables ML engineers and Python developers to scale Python apps and accelerate machine learning workloads.

Ray cluster is:

a set of **worker nodes connected to** a common Ray **head node**. Ray clusters can be fixed-size, or they can autoscale up and down according to the resources requested by applications running on the cluster.

Ray AIR and integrations



Overview of Ray

Ray project

Key Ray characteristics

Ray libraries



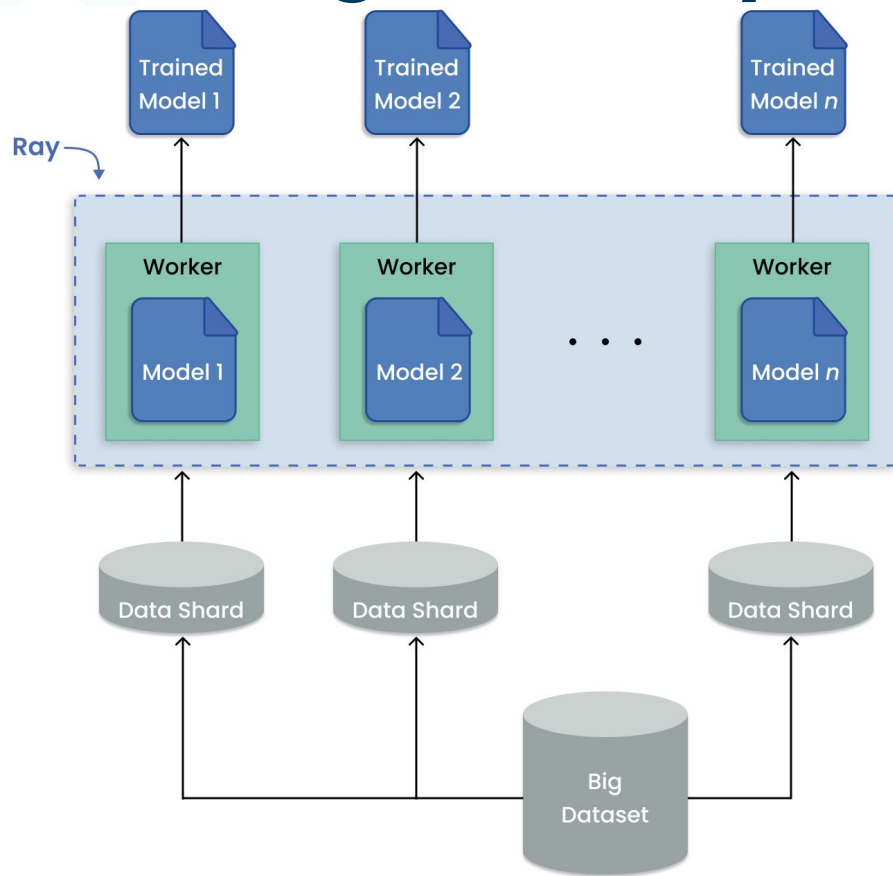
Example use cases

Summary

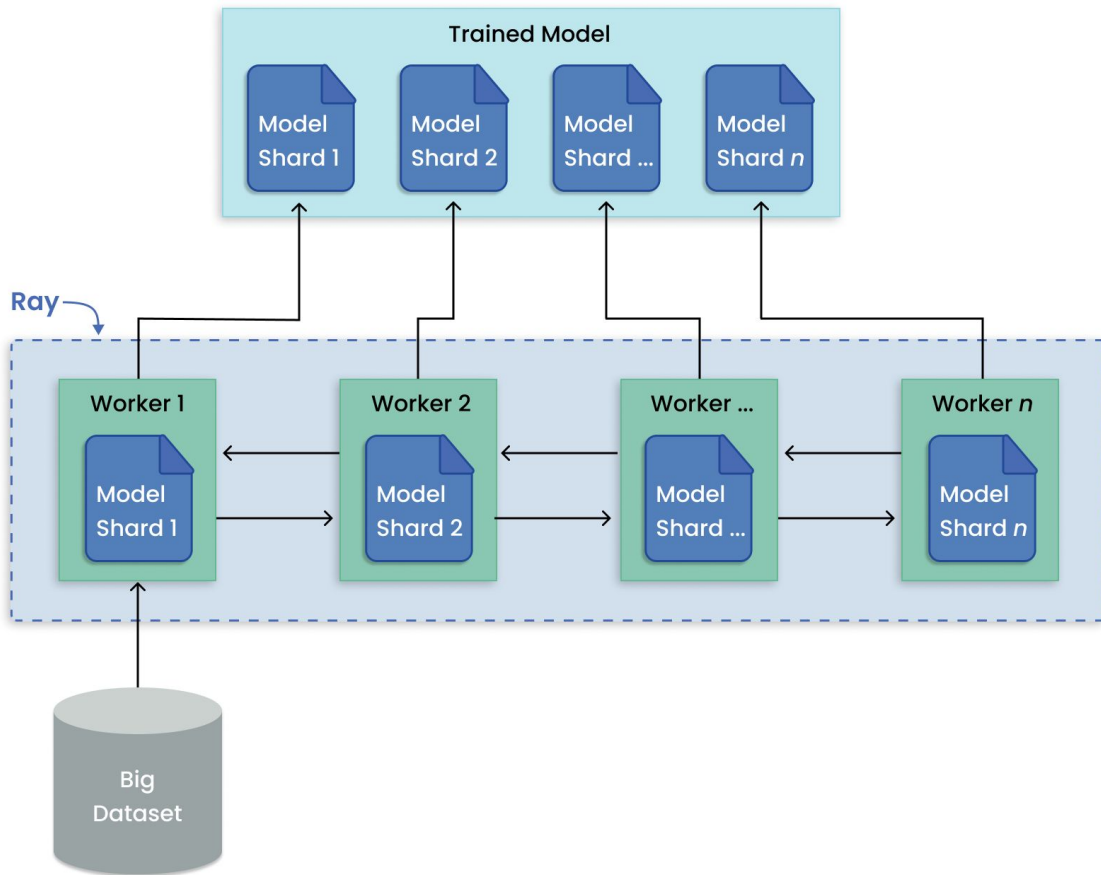
Ray use cases

Scaling ML workloads

Parallel training of many models

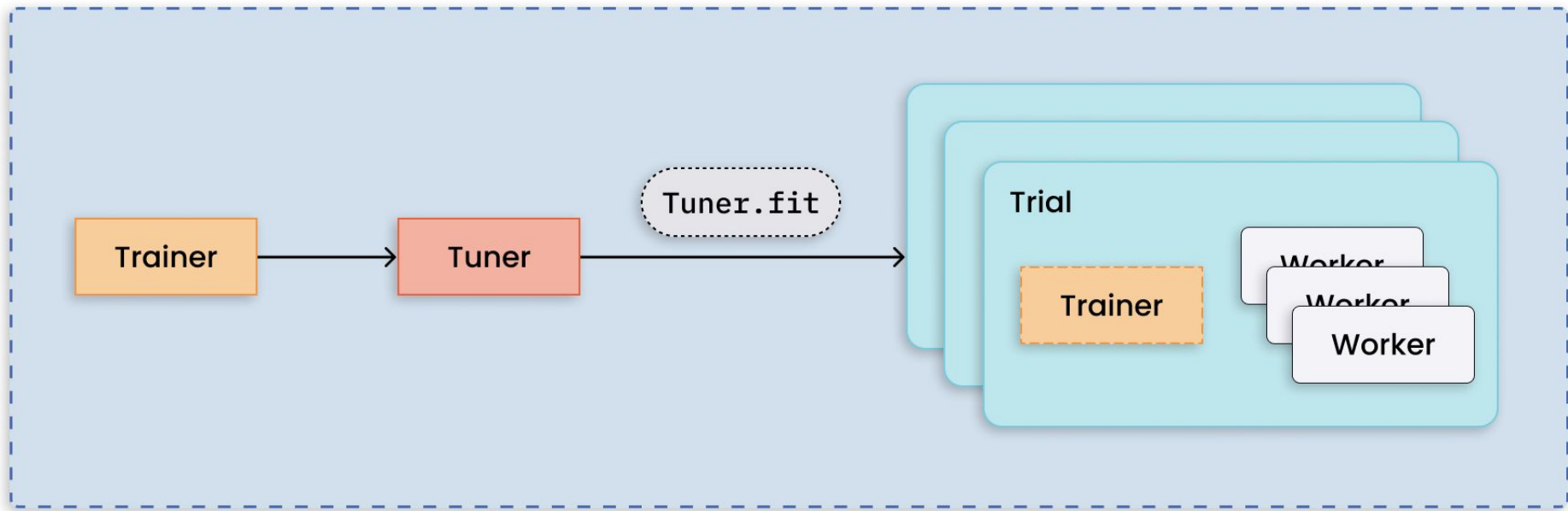


Distributed training of large models



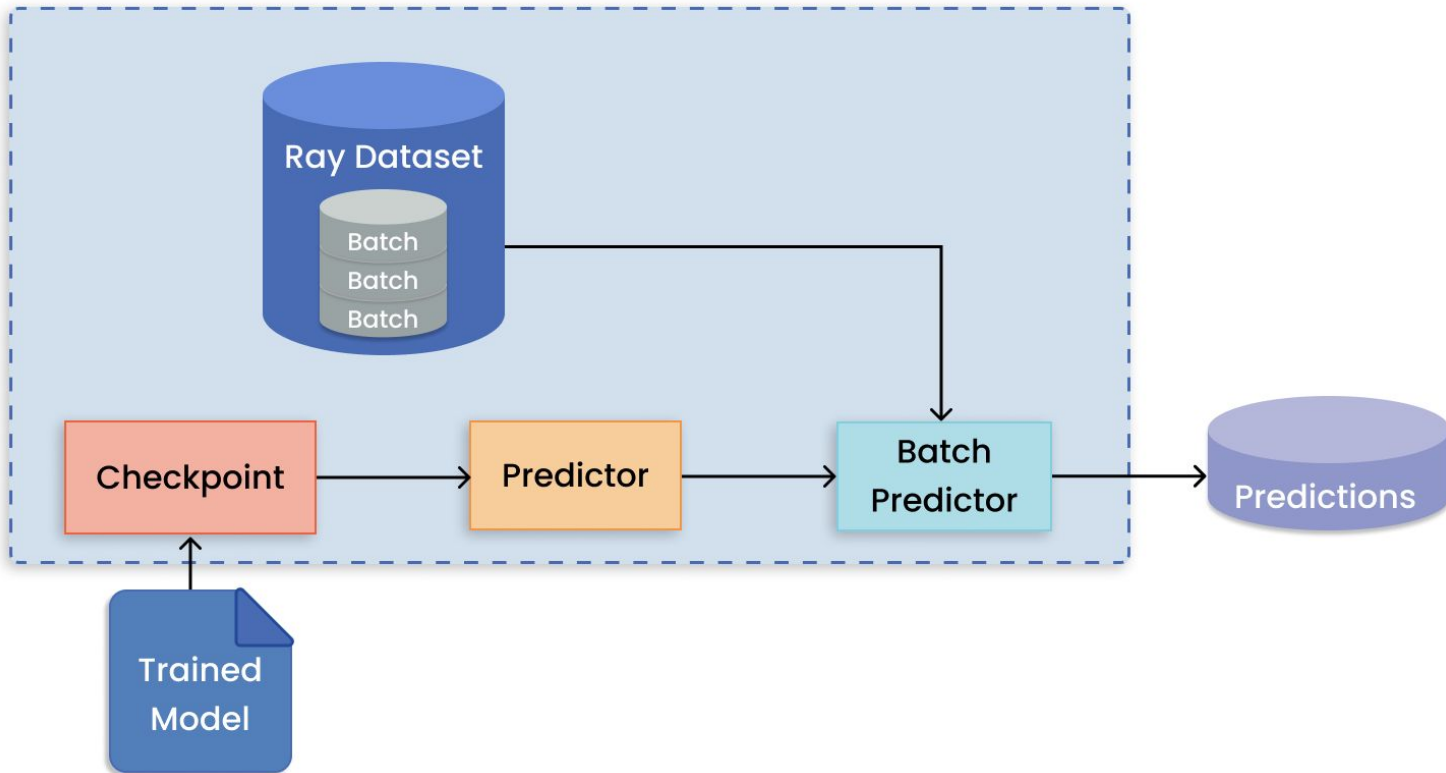
Managing parallel hyperparameter tuning experiments

Ray



Batch inference on CPUs and GPUs

Ray

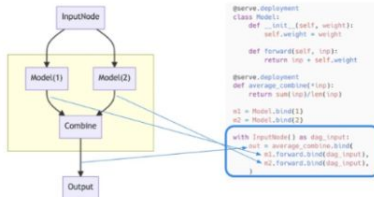


Multi-model composition for model serving

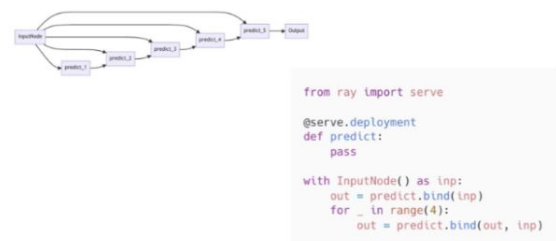
Chaining



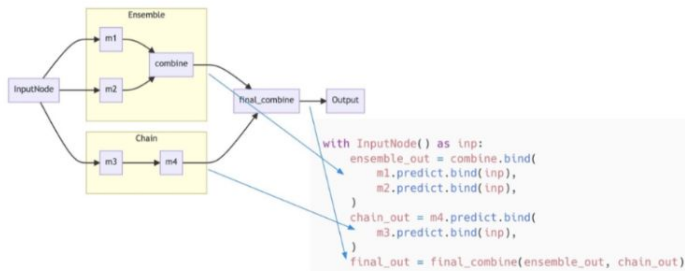
Ensemble



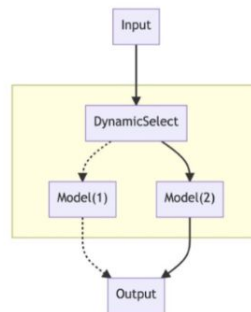
Tree



Ensemble + Chaining



Dynamic Selection



Embed **imperative** API
within **declarative** dataflow

```
@serve.deployment
class Model:
    def __init__(self, weight):
        self.weight = weight

    def forward(self, inp):
        return inp + self.weight

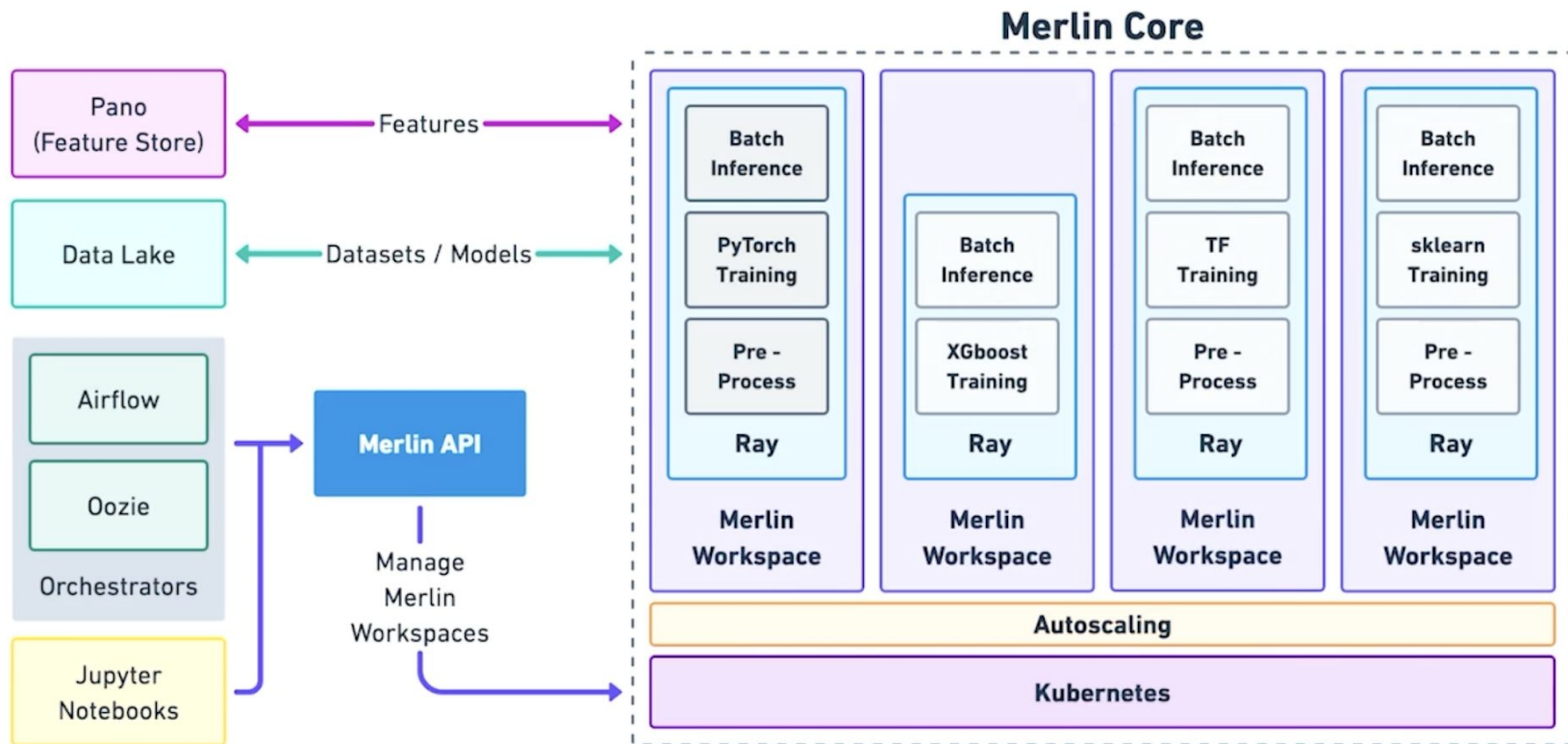
@serve.deployment
class RandomSelection:
    def __init__(self, m1, m2):
        (self.m1, self.m2) = m1, m2

    def forward(self, inp):
        if random.random() < 0.5:
            return self.m1.forward.remote(inp)
        else:
            return self.m2.forward.remote(inp)

m1 = Model.bind(1)
m2 = Model.bind(2)
select = RandomSelection.bind(m1, m2)

with InputNode() as dag_input:
    out = select.forward.remote(dag_input)
```


ML platform



Overview of Ray



Ray project

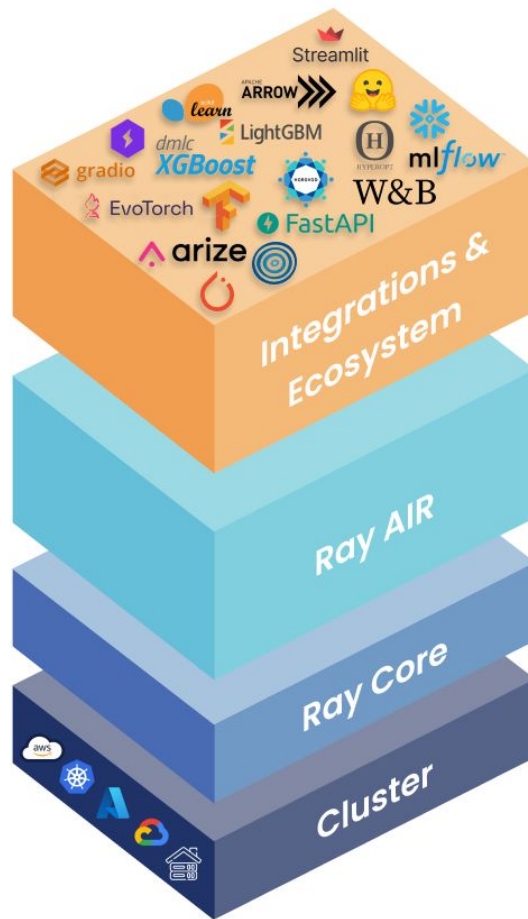
Key Ray characteristics

Ray libraries

Example use cases

Summary

Ray libraries

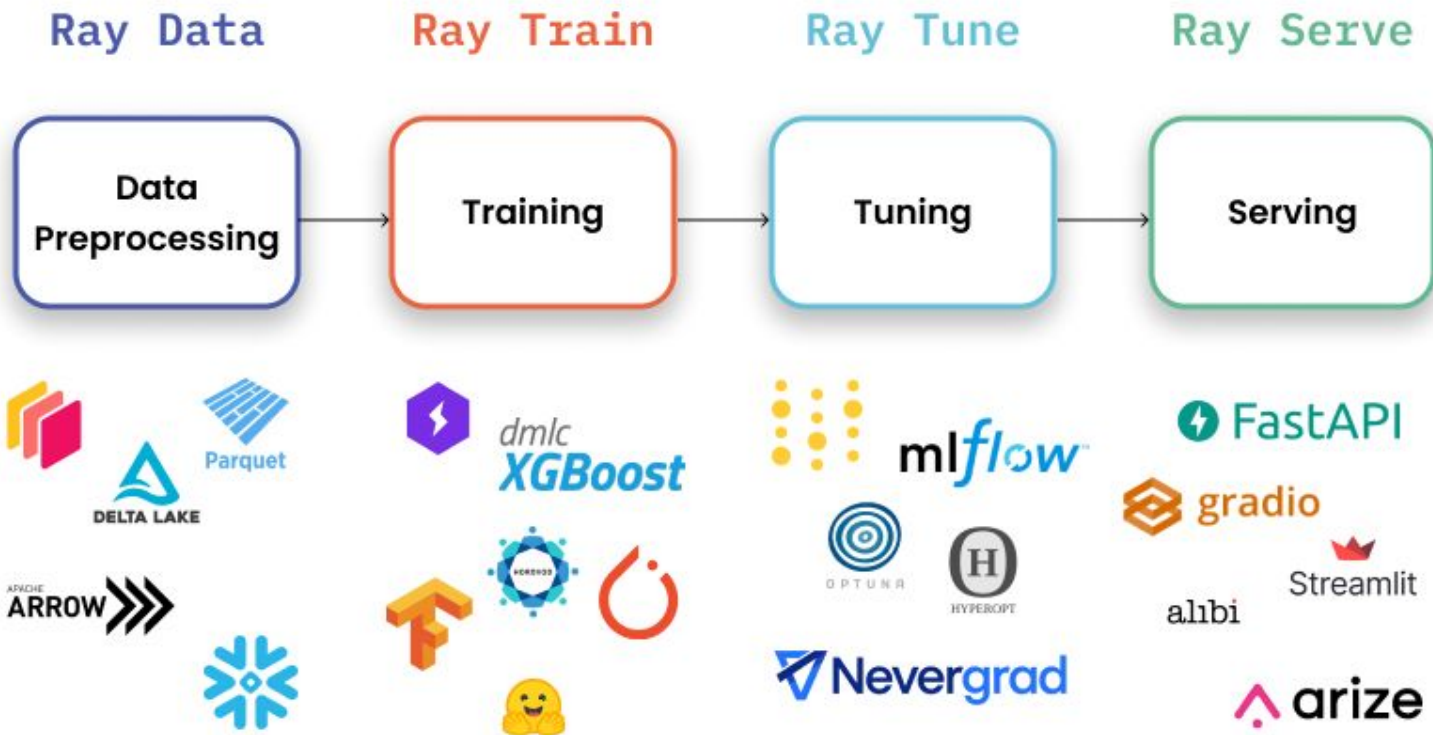


— a growing ecosystem of integrations and partnerships with popular projects

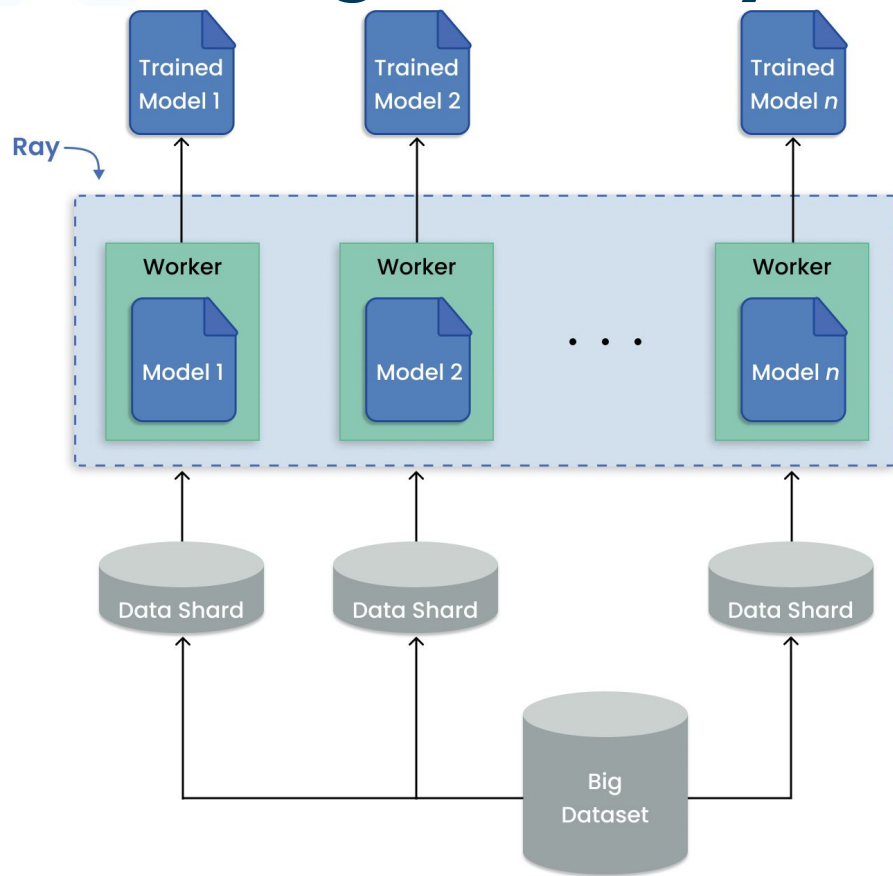
— high-level libraries which enable simple scaling of AI workloads

— a low-level distributed computing framework with a concise core, Python-first API

Ray AIR and integrations



Parallel training of many models



Thank you

Any questions?

