One Cluster to Rule Them All

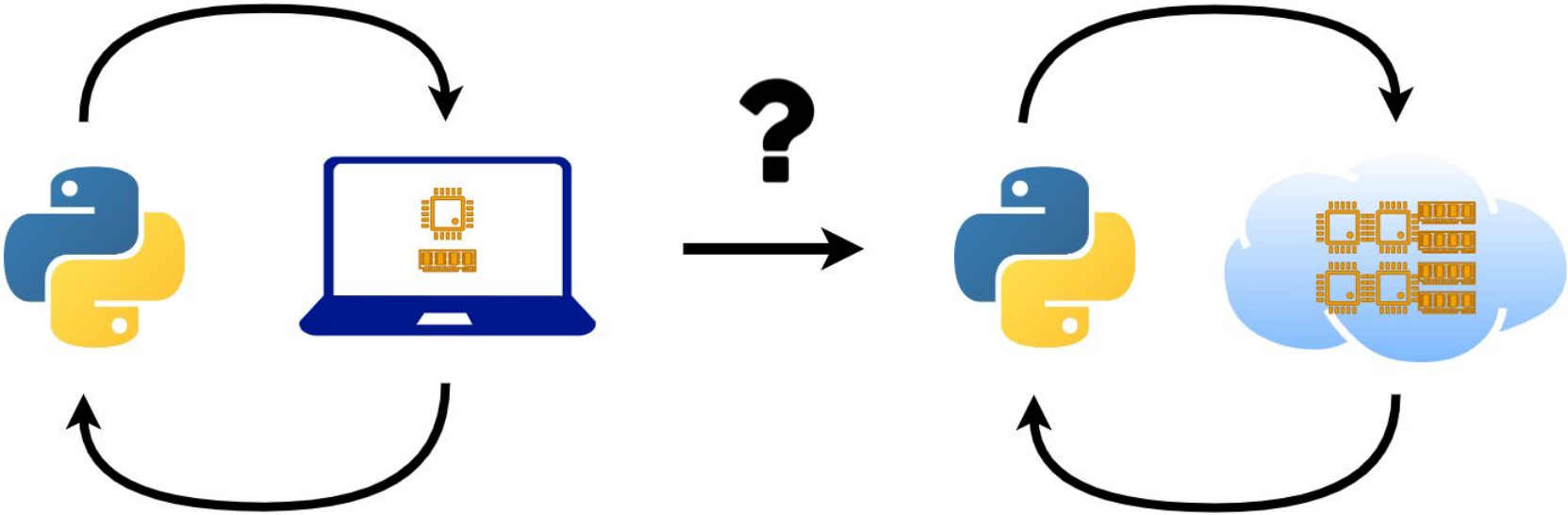
ML on the Cloud

Victor Yap

MLOps Engineer

Rev.com / Rev.ai

Speech Recognition for Customers



Autoscaling

Rightsizing

Spot Instances

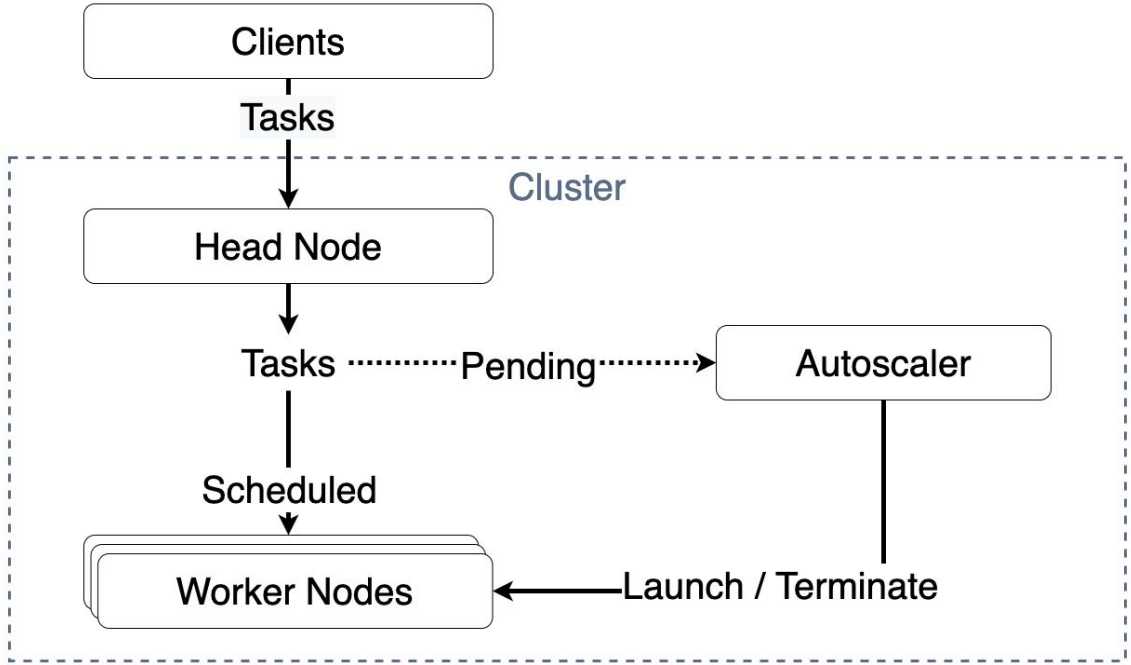
Environment Management

Ray

Kubernetes

Karpenter

Ray - “a simple, universal API for building distributed applications”



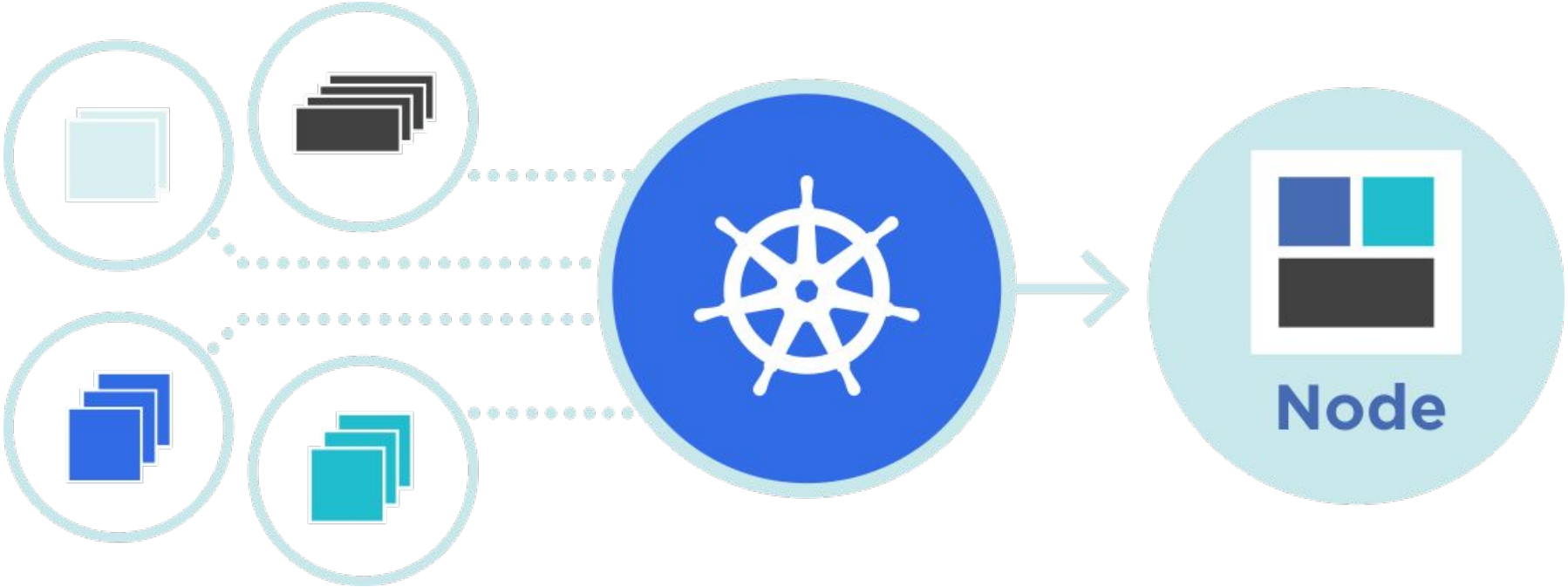
AWS Batch?

Slurm?

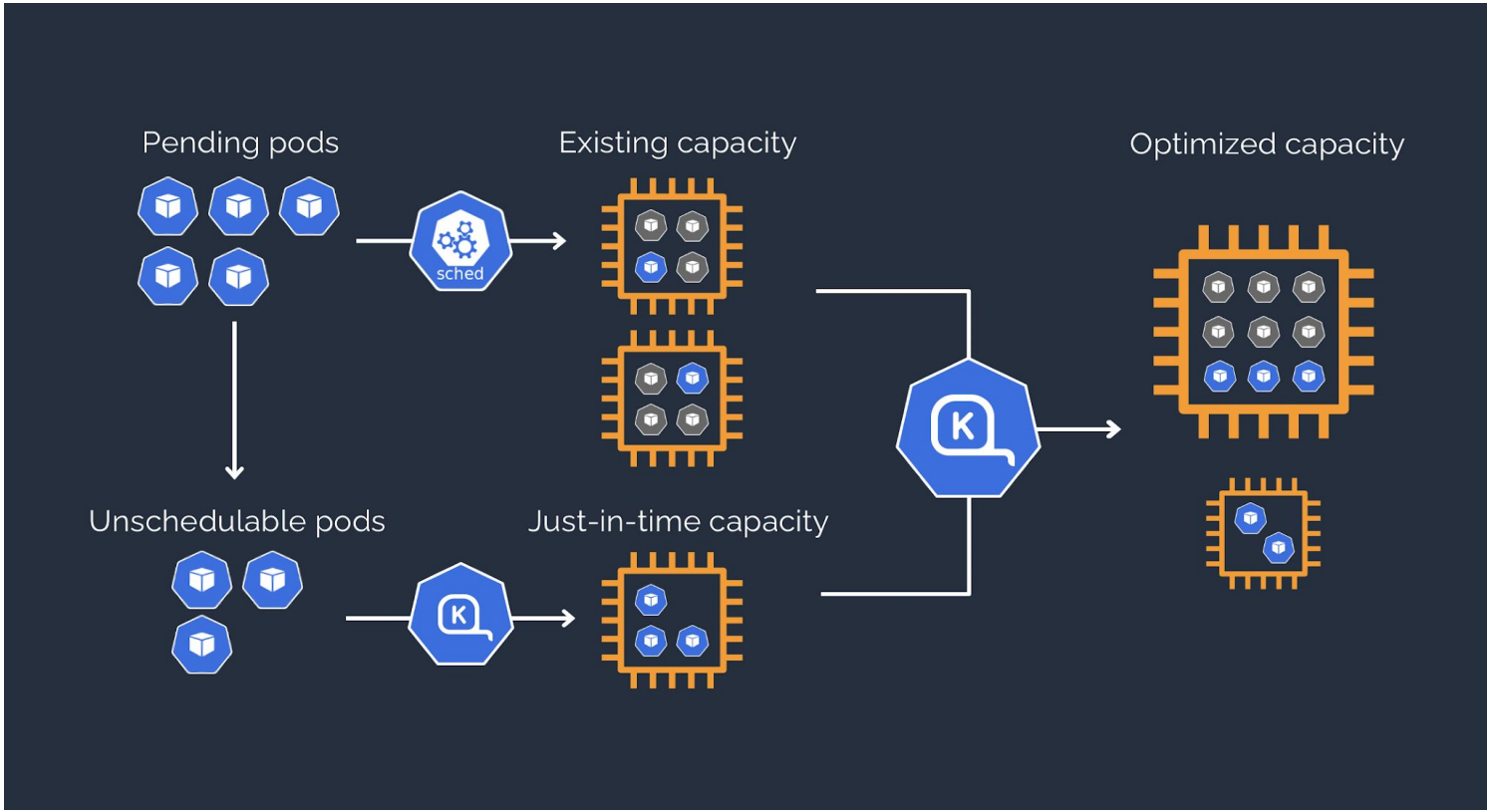
Ray is Python-Friendly

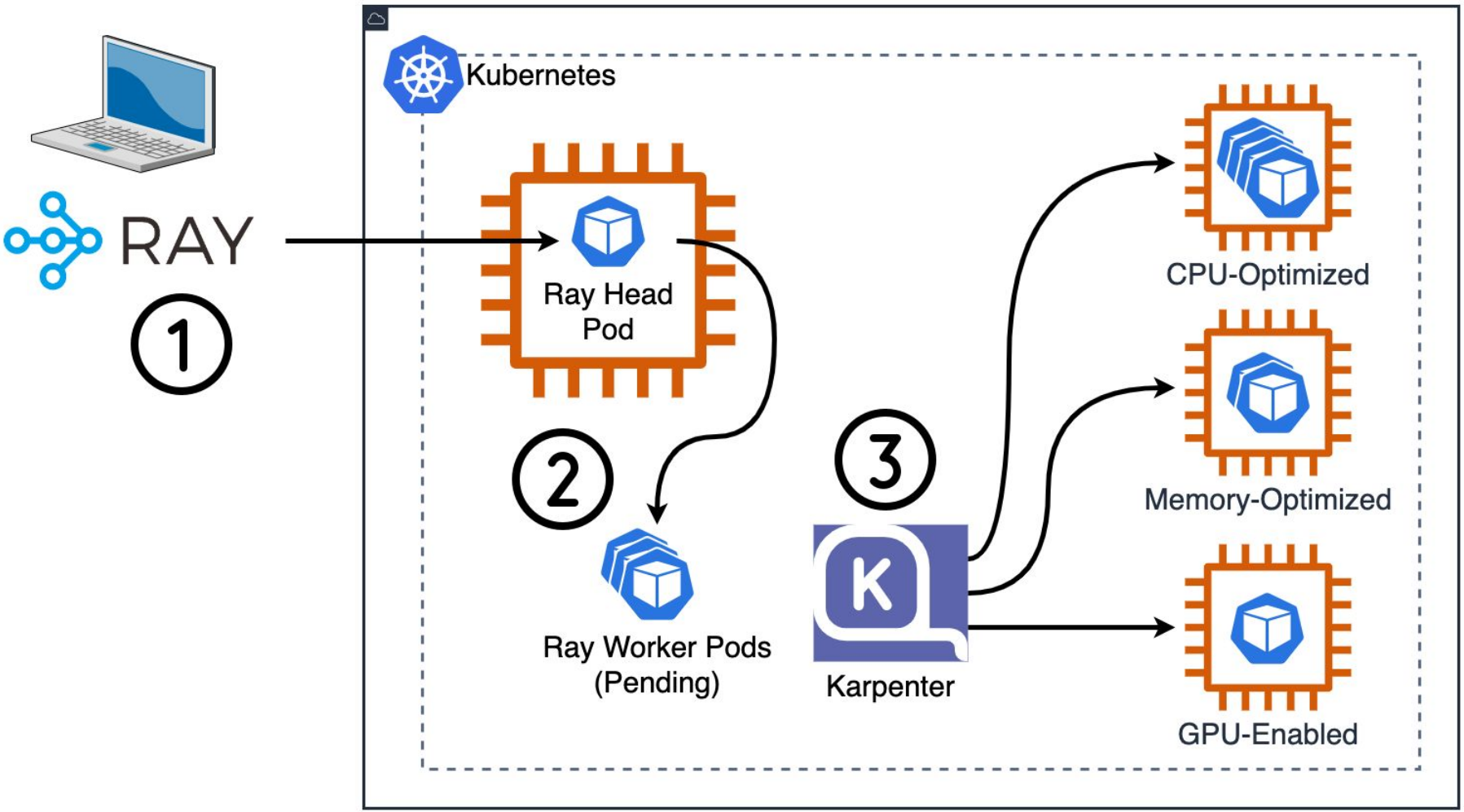
```
ray.init()  # starts a local cluster in one line

@ray.remote(num_cpus=1, memory=1024 ** 3)
def preprocess(data):
    pass  # do stuff

@ray.remote(num_gpus=1, accelerator_type="p2")
def train():
    pass  # do stuff
```

Kubernetes



Karpenter


Pending pods
sched
Existing capacity
Optimized capacity
Unschedulable pods
Just-in-time capacity


RAY
1
Kubernetes
Ray Head
Pod
2
Ray Worker Pods
(Pending)
3
Karpenter
CPU-Optimized
Memory-Optimized
GPU-Enabled

Demo

```
import ray

# ray.init() will use the local system to run a cluster
ray.init()

# ray.init accepts an address to connect to a cluster
ray.init("ray://127.0.0.1:10001")
```

```python
import time
from contextlib import contextmanager

@contextmanager
def timer():
    """Context manager to measure running time of code."""
    start = time.time()
    yield
    time_elapsed = round(time.time() - start)
    print(f"timer: took {time_elapsed} seconds")
```

```
def get_instance_type():
    """Returns what instance type this function is running on."""
    import requests
    token = requests.put(
        "http://169.254.169.254/latest/api/token",
        headers={"X-aws-ec2-metadata-token-ttl-seconds": "21600"}
    ).text
    instance_type = requests.get(
        "http://169.254.169.254/latest/meta-data/instance-type",
        headers={"X-aws-ec2-metadata-token": token},
    ).text
    return instance_type
```

```
def print_cluster_resources():
    """Prints the CPUs, memory and GPUs of the current Ray cluster."""
    cluster_resources = ray.cluster_resources()
    CPUs = int(cluster_resources["CPU"])
    memory = round(cluster_resources["memory"] / (1000 ** 3))
    GPUs = round(cluster_resources.get("GPU", 0))
    print(f"CPUs = {CPUs}, memory = {memory}G, GPUs = {GPUs}")
```

```python
@ray.remote(num_cpus=1, memory=1000 ** 3)
def preprocess(data):
    time.sleep(1)
    return get_instance_type()

with timer():
    print(ray.get(preprocess.remote("data")))
```

```
t3.2xlarge
timer: took 2 seconds
```

```
print_cluster_resources()

from collections import Counter
with timer():
    print(Counter(
        ray.get([preprocess.remote(x) for x in range(60)])
    ))
```

```
CPUs = 4, memory = 9G, GPUs = 0
Counter({'t3.2xlarge': 60})
timer: took 16 seconds
```

```
from collections import Counter
with timer():
    print(Counter(
        ray.get([preprocess.remote(x) for x in range(6000)])
    ))

print_cluster_resources()
```

```
Counter({'m6a.48xlarge': 4443, 'm6a.32xlarge': 1362, 'c6id.4xlarge': 195})
timer: took 50 seconds
CPUs = 292, memory = 442G, GPUs = 0
```

```
@ray.remote(memory=100 * 1000 ** 3)
def preprocess_big_data():
    return get_instance_type()

print(ray.get(preprocess_big_data.remote()))
print_cluster_resources()
```

```
i4i.8xlarge
CPUs = 34, memory = 197G, GPUs = 0
```

```
@ray.remote(num_gpus=4, accelerator_type="p2")
def train():
    return get_instance_type()

with timer():
    print(ray.get(train.remote()))
print_cluster_resources()
```

```
p2.xlarge
timer: took 178 seconds
CPUs = 37, memory = 239G, GPUs = 1
```

```
@ray.remote(num_gpus=4, accelerator_type="p2")
def train():
    return get_instance_type()

with timer():
    print(ray.get(train.remote()))
print_cluster_resources()
```

(issues in Karpenter prevented this from actually working)

Should You Try This?

Or

Choose Proven Technologies?

Demo Code:

github.com/vicyap/mlops-world-2022