Distributed Systems
CS 15-440
Ray

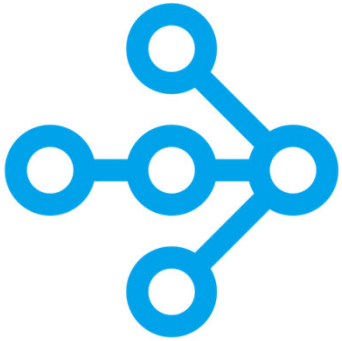
Lecture 19, November 14, 2023

Hend Gedawy

جامعة كارنيجي ميلون في قطر
Carnegie Mellon University Qatar

Outline

- Introduction
- Ray Programming & Computation Model
- Ray Cluster Architecture
- Ray Scheduling
- Lifetime of a Ray Task

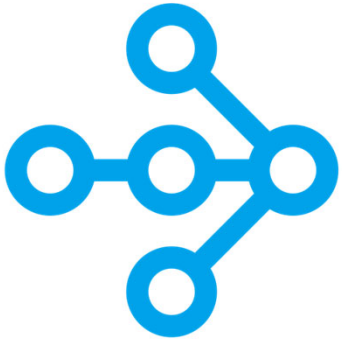


RAY

جامعة كارنيجي ميلون في قطر
Carnegie Mellon University Qatar

Outline

- **Introduction**
- Ray Programming & Computation Model
- Ray Cluster Architecture
- Ray Scheduling
- Lifetime of a Ray Task



RAY

جامعة كارنيجي ميلون في قطر
Carnegie Mellon University Qatar

Ever Growing Data Quantities

A forecast by International Data Corporation (IDC) estimates that there will be **41.6 billion IoT devices** in 2025, capable of generating **79.4 zettabytes (79.4 * 10^{21} bytes) of data.**

Data Analysis & Machine Learning

جامعة كارنيجي ميلون في قطر
Carnegie Mellon University Qatar

State of the Art Models & the Need to Scale

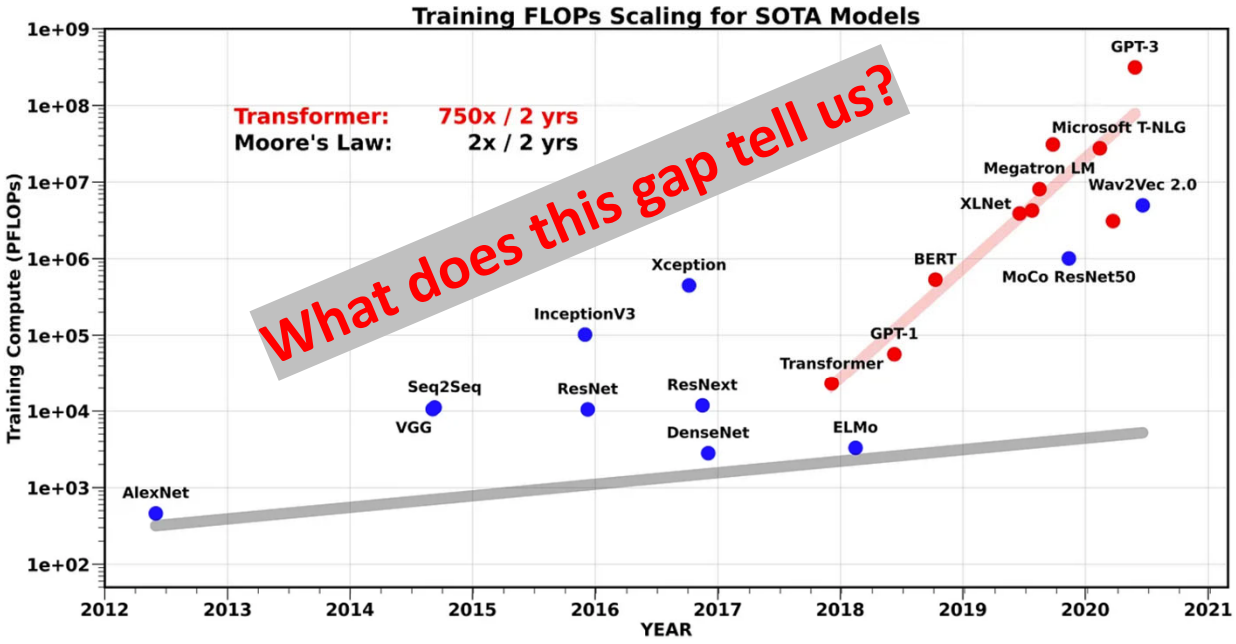


Figure 1: The amount of compute, measured in Peta FLOPs, needed to train SOTA models, for different CV, NLP, and Speech models, along with the different scaling of Transformer models (750x/2yrs)*1 [Download This Image]

According to unverified information leaks, **GPT-4** was **trained** on about 25,000 Nvidia A100 GPUs for **90–100 days**.

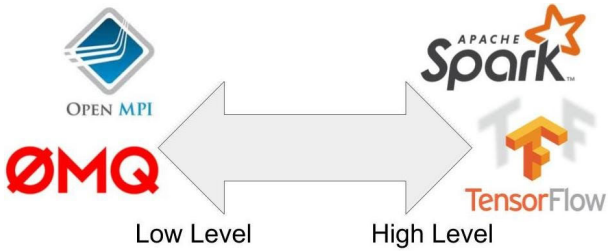
Assuming that the GPUs were installed in Nvidia HGX servers which can host 8 GPUs each, meaning 25,000 / 8 = **3,125 servers** were needed.

source

https://medium.com/riselab/ai-and-memory-wall-2cb4265cb0b8

جامعة كارنيجي ميلون في قطر
Carnegie Mellon University Qatar

Platforms/Tools for Distributed Data Analysis & ML

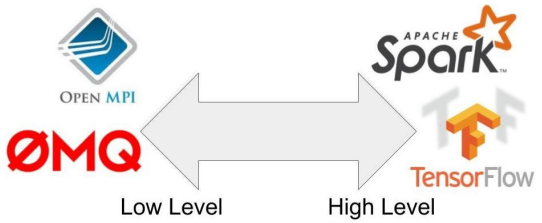


- On one end of the spectrum, we have tools like OpenMPI, Python multiprocessing, and ZeroMQ,
- Provide low-level primitives for sending and receiving messages.
- These tools are very powerful
- They provide a different abstraction and so single-threaded applications must be rewritten from scratch to use them.
- You do the data splitting, distribution, and collect results

Platforms/Tools for Distributed Data Analysis & ML

On the other end of the spectrum, there is **domain-specific distributed computing tools** for different machine learning components in the ML ecosystem

These tools are very powerful and specialized

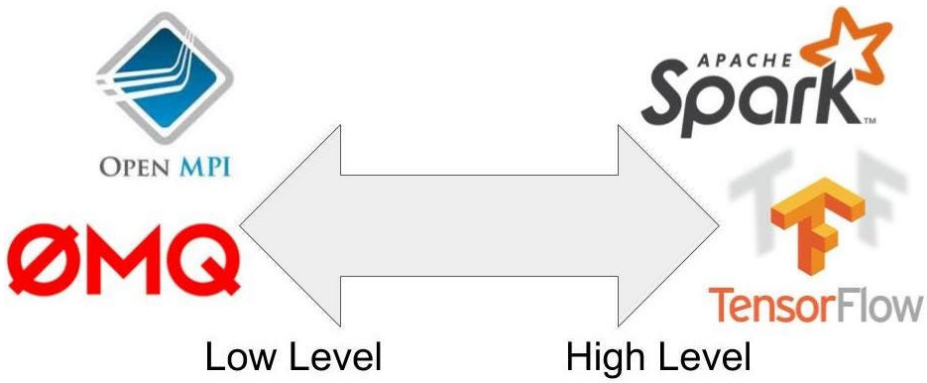


Data Processing	Training	Serving	Hyper. Tuning	Reinforcement Learning
Map Reduce, Hadoop, Spark	Horovod, Distributed TensorFlow, PyTorch	Clipper, MLFlow, FastAPI, Comet	Vizier, HyperOpt, Optuna	Baselines, Rllab, ELF, coach, TensorForce , ChainerRL

To build a **cross-cutting application** that needs tight coupling between different ML components (e.g. training and serving);
Either **glow** together these different systems
Or **Build** a new system **from scratch** (e.g. Alpha Go by DeepMind and DoTa by OpenAI)

جامعة كارنيجي ميلون في قطر
Carnegie Mellon University Qatar

Ray Novelty



1. Occupies a unique **middle ground** (general purpose but takes care of scaling, performance, scheduling, elasticity, etc.)
2. **Powerful API** that builds on existing concepts (i.e. functions and classes) and allows a serial applications to be parallelized and run on a cluster with relatively **few additional lines of code**

Ray Novelty

Ray simplifies the process of developing and deploying large-scale ML models.

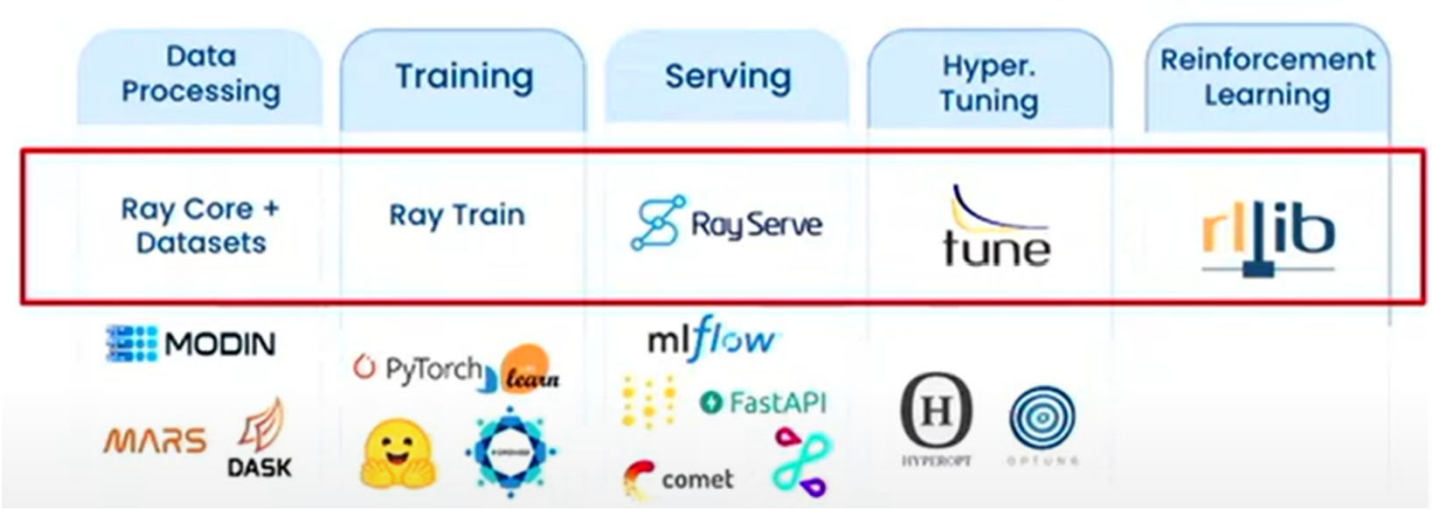


Photo Credit: https://www.youtube.com/watch?v=iuSbCoe34cw

What is Ray

Ray is an active open-source project developed at the University of California, Berkeley. (First Released 2018)

- Ray provides a general-purpose distributed compute framework for
  - scaling **ML models or workloads**, allowing developers to train and deploy models faster and more efficiently.
  - writing **parallel and distributed Python application**
- It provides fundamental primitive abstractions that allow taking the existing program and turn it into a distributed one

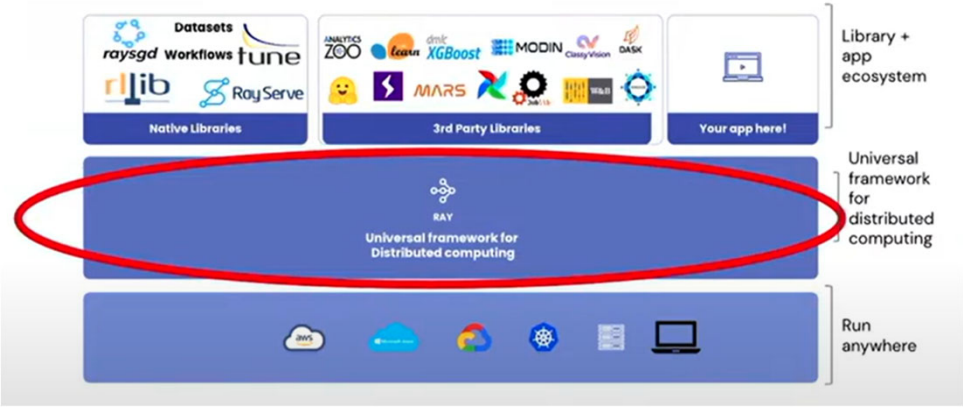
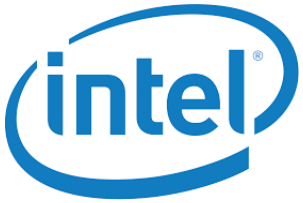


Photo Credit: https://www.youtube.com/watch?v=iuSbCoe34cw

جامعة كارنيجي ميلون في قطر
Carnegie Mellon University Qatar

Companies Using Ray - Examples



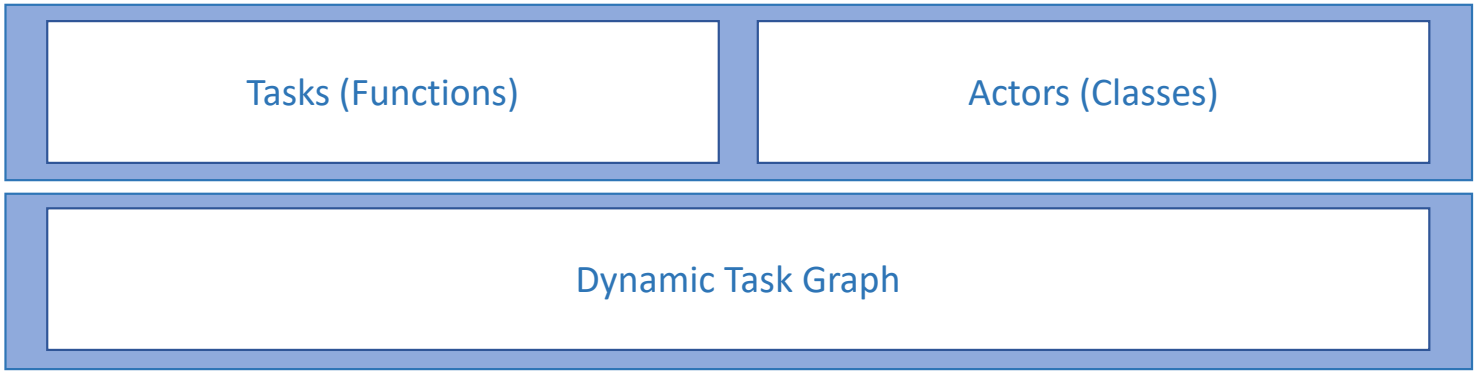
جامعة كارنيجي ميلون في قطر
Carnegie Mellon University Qatar

Outline

- Introduction
- **Ray Programming & Computation Model**
- Ray Cluster Architecture
- Ray Scheduling
- Lifetime of a Ray Task

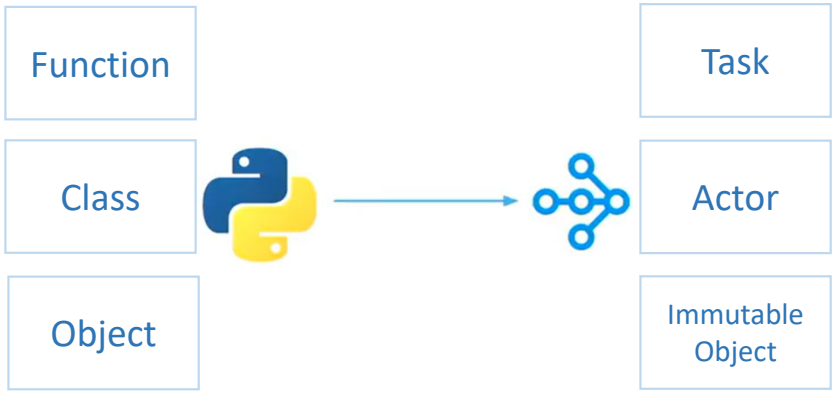
جامعة كارنيجي ميلون في قطر
Carnegie Mellon University Qatar

Programming & Computation Model



Python to Ray

Ray API allows serial applications to be parallelized without major modifications.



Ray takes the existing concepts of **functions** and **classes** and translates them to the distributed setting as **tasks** and **actors**.

Both Operate on **Immutable objects**

Tasks

A python **function** can **be turned into parallel Task by adding @ray.remote** decorater

```
@ray.remote
def zeros(size):
    return np.zeros(size)
```

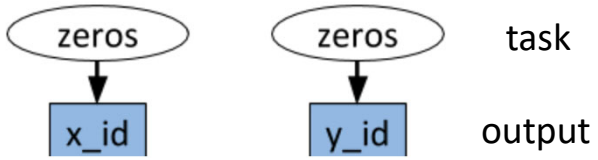
When the **function is invoked by adding (.remote)** to the function name

- This generates a task
- The task is handed off to the backend
- The backend schedules it on the machine(s)/worker(s) to execute it asynchronously

```
x_id = zeros.remote((100, 100))
y_id = zeros.remote((100, 100))
```

The **method invocation returns immediately** (no locking)

- It returns an object (future or object ID) representing the eventual output of the computation



How would using Futures be helpful?

جامعة كارنيجي ميلون في قطر
Carnegie Mellon University Qatar

Ray Objects as futures

- **Futures** are objects can contain a value sometime in the future
  - The value is fetched When available
  - Enable Asynchronous Execution

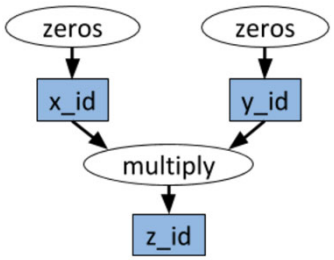
- Ray objects are known as **remote objects** because they can be stored anywhere within a Ray cluster.
  - can exist on one or multiple nodes

Tasks Dependency & Blocking

- **Tasks Dependency:**
  - A task that depends on other tasks, won't execute until they finish
  - Although it returns immediately

- **Blocking** and waiting for the results: Ray.get

A dynamic graph is created in the backend as tasks are invoked




```
# Define two remote functions. Invocations of these functions create tasks
# that are executed remotely.

@ray.remote
def multiply(x, y):
    return np.dot(x, y)

@ray.remote
def zeros(size):
    return np.zeros(size)

# Start two tasks in parallel. These immediately return futures and the
# tasks are executed in the background.
x_id = zeros.remote((100, 100))
y_id = zeros.remote((100, 100))

# Start a third task. This will not be scheduled until the first two
# tasks have completed.
z_id = multiply.remote(x_id, y_id)
```

```
# Get the result. This will block until the third task completes.
z = ray.get(z_id)
```

Photo Credit

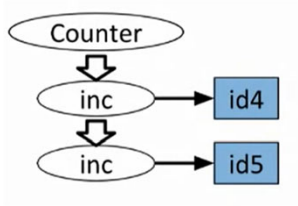
The need for Actors

- What if the application **requires multiple tasks operating on the same shared mutable state.**
- This comes up in multiple contexts in machine learning where the shared state may be:
  - The state of a simulator,
  - The weights of a neural network,
  - Encapsulation of some interaction with the real world
- This **can't be done with** the remote functions and **tasks**
- Ray uses an **actor abstraction** to **encapsulate mutable state** shared between multiple tasks.

Actors

- Stateful service on cluster that enables message passing
- Creating an object of the actor, starts **a new process** in the cluster with that object state
- **Method invocations** on the actor object translates into **tasks** assigned to and executed on that actor
- Tasks invoked on the actor share the state of the process and can mutate it
  - The methods invoked in an actor can be used to send state/messages
- **Each task invoked** on the actor is **implicitly dependent** on the **one** that executed **before it**
- **Ray.get()** allows blocking on actor tasks
- You can specify the **logical resources** (number of CPUs/GPUs) to be **allocated** for the task or actor
  - These are **hard** requirements

Note: Execution of both remote tasks and actor methods is automatically triggered by the system when their inputs become available.



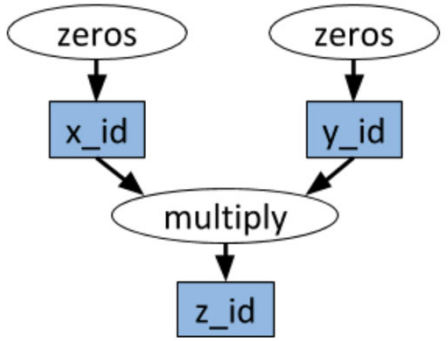
Actors

```
@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()
ray.get([id4, id5])
```

Dynamic Task Graph

- Represents the **entire application** and is only executed a single time.
- It is not known up front. It is **constructed dynamically** as the application runs
  - The execution of one task may trigger the creation of more tasks.
- Both **Task and Actor** abstractions are working on top of the **same dynamic task graph abstraction**
  - The mutable state is encoded in the graph abstraction

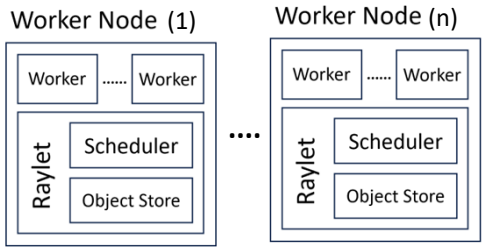



```
# Define two remote functions. Invocations of these functions create tasks
# that are executed remotely.

@ray.remote
def multiply(x, y):
    return np.dot(x, y)

@ray.remote
def zeros(size):
    return np.zeros(size)

# Start two tasks in parallel. These immediately return futures and the
# tasks are executed in the background.
x_id = zeros.remote((100, 100))
y_id = zeros.remote((100, 100))

# Start a third task. This will not be scheduled until the first two
# tasks have completed.
z_id = multiply.remote(x_id, y_id)

# Get the result. This will block until the third task completes.
z = ray.get(z_id)
```

Outline

- Introduction
- Ray Programming & Computation Model
- **Ray Cluster Architecture**
- Ray Scheduling
- Lifetime of a Ray Task

Ray Cluster – Worker Nodes



A set of worker nodes each of which consists of the following physical processes:

One or more **worker processes**

- Python processes waiting in a loop to be assigned tasks
- A worker process is either *stateless* (execute any function) or an *actor* (only executes methods from a class).
- The default number of initial workers is equal to *the number of CPUs on the machine*.

A **raylet:** a C++ program and has two main that communicate across the entire cluster

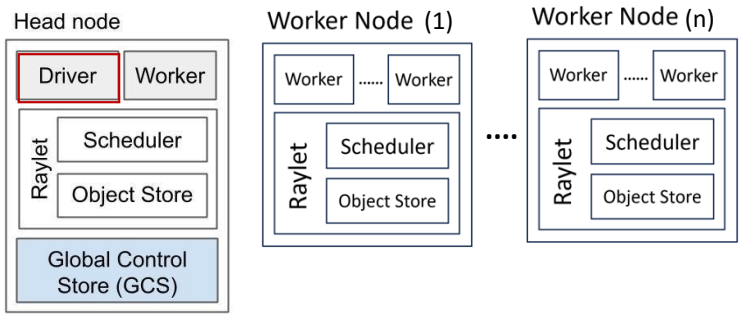
- A **scheduler**. Responsible for resource management to fulfill task requirements
- A **shared-memory object store** responsible for storing and transferring large objects.
  - Once a worker competes a task, it talks to the object store directly and stores the input and output value of each task
  - *No IPC* going on between worker processes on the same node & *No context switching*

جامعة كارنيجي ميلون في قطر
Carnegie Mellon University Qatar

Ray Cluster – Head Node

One of the worker nodes is designated as the **head node**.
In addition to the worker processes, the head node also hosts:

- The **Driver process(es):** is a special worker process that executes the top-level application (e.g., `__main__` in Python).
  - can submit tasks, but cannot execute any itself.
  - can run on any node, but by default are located on the head node

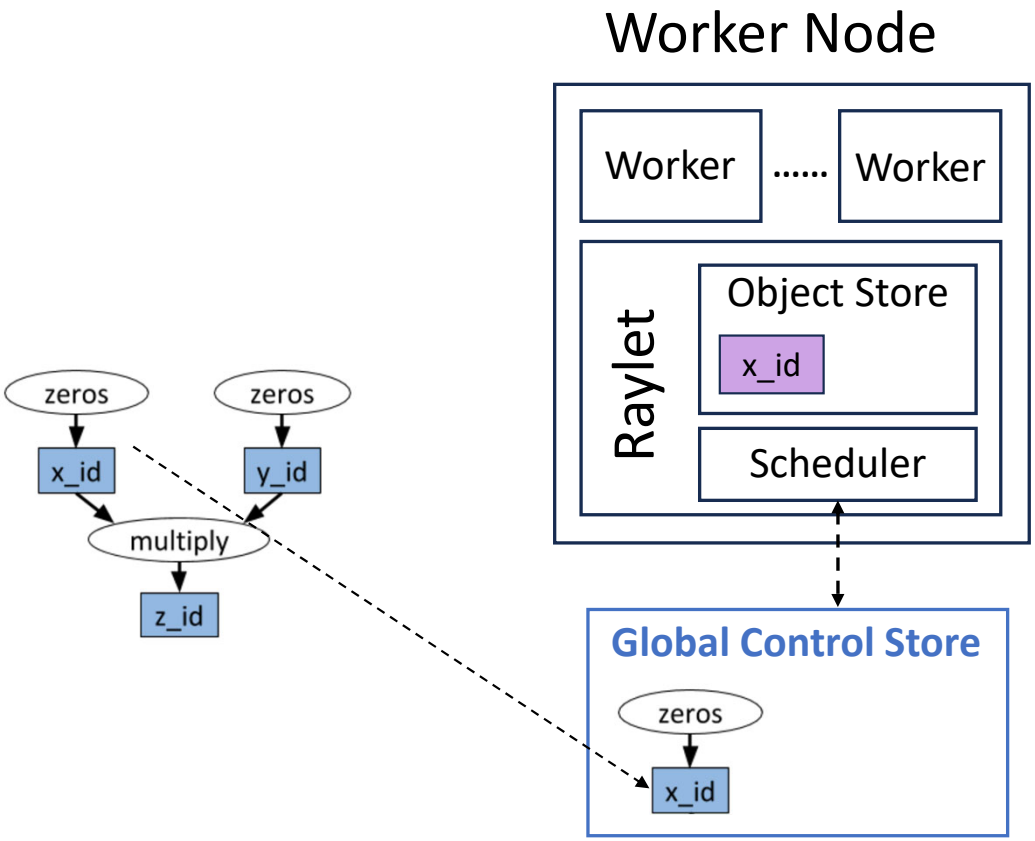


- The **Global Control Store (GCS)** is the brain that has all the metadata about tasks.
  - Maintains the entire control state
  - The GCS is a fault-tolerant key-value server
  - Updated based on heartbeats
  - Can run anywhere

Scheduling is unique because it is distributed & Object store is distributed across the entire cluster and allows sharing objects between nodes

جامعة كارنيجي ميلون في قطر
Carnegie Mellon University Qatar

Ray Scalable Architecture

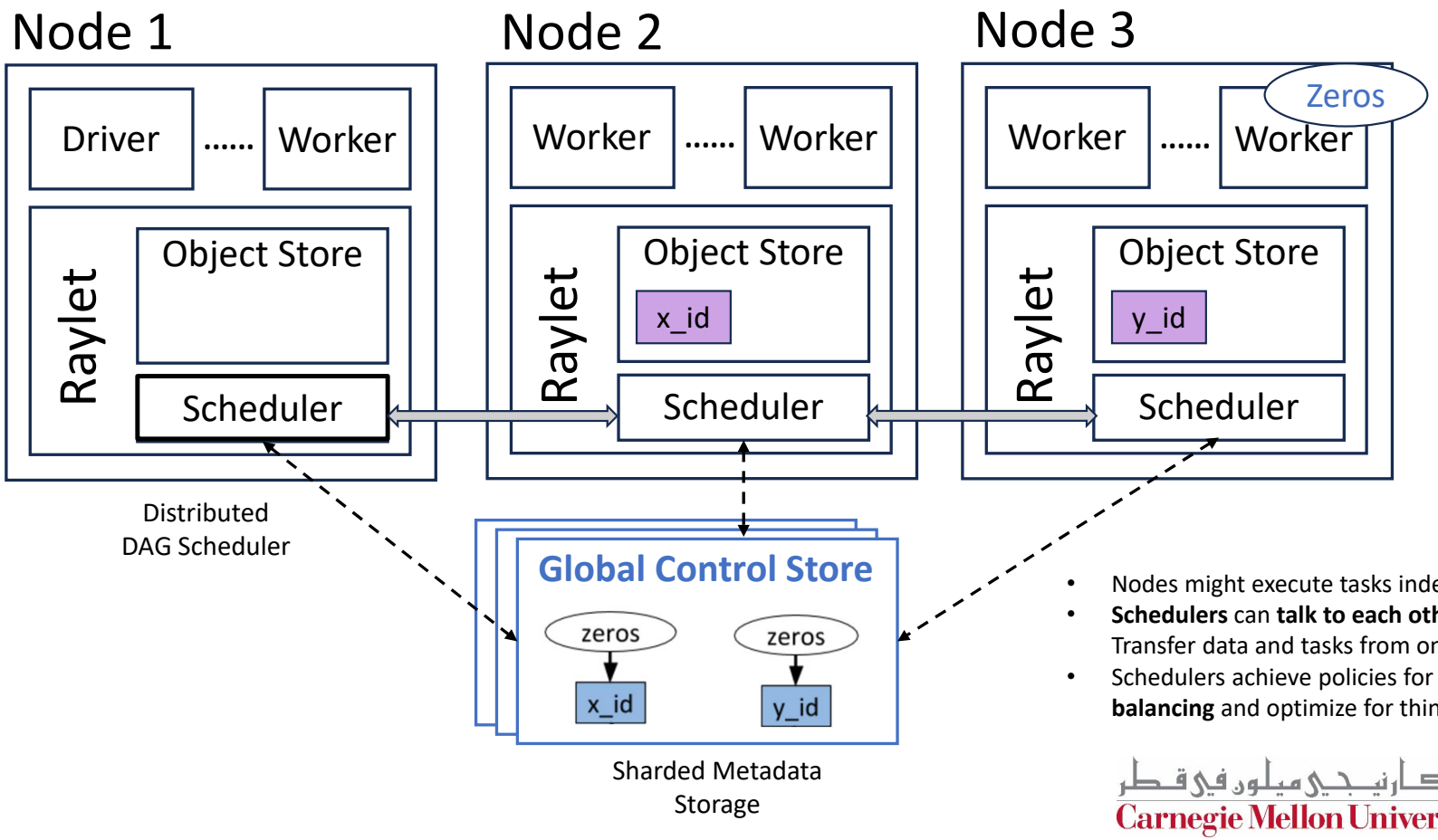


How is the Dynamic task graph (DTG) executed in a system?

- The scheduler logs the (lineage of the) task to the GCS
- the scheduler assigns the task to one of the worker processes
- The worker process will return the value and store it in the object store
- The output value can be retrieved now using ray.get()

جامعة كارنيجي ميلون في قطر
Carnegie Mellon University Qatar

Ray Scalable Architecture



- Nodes might execute tasks independently
- **Schedulers** can **talk to each other directly** to Transfer data and tasks from one node to another
- Schedulers achieve policies for **better load balancing** and optimize for things like **data locality**

جامعة كارنيجي ميلون في قطر
Carnegie Mellon University Qatar

Outline

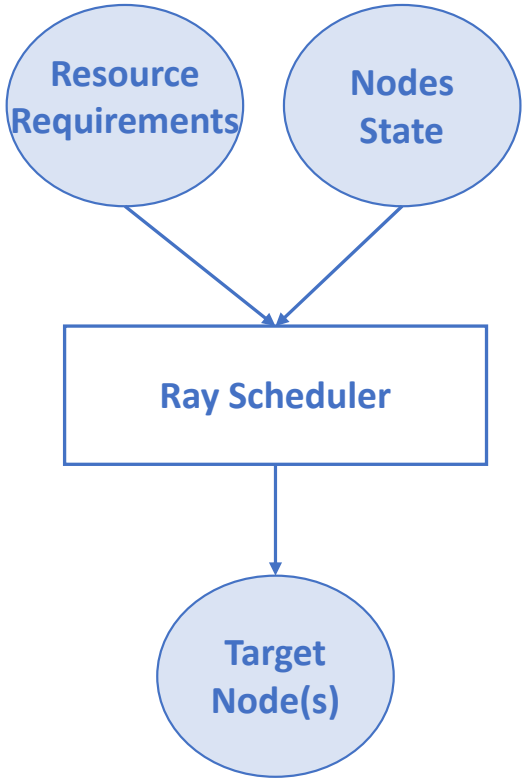
- Introduction
- Ray Programming & Computation Model
- Ray Cluster Architecture
- **Ray Scheduling**
- Lifetime of a Ray Task

جامعة كارنيجي ميلون في قطر
Carnegie Mellon University Qatar

Ray Scheduling

Given task or actor with specific resource requirements , what is the best node(s) to run this task/actor.

- A Node State:
  - Feasible: Node has the required resources:
    - Available: resources are free now
    - Unavailable: resources are currently used by another (task or actor)
  - Infeasible: Node doesn't have required resources (GPU required nut only CPU available)
- Task or actor can only be scheduled if there are feasible nodes (even if unavailable), otherwise Ray waits for feasible nodes are added to the cluster.
- Ray has different scheduling strategies; can be specified with @ray.remote decorator
  - E.g. `@ray.remote(scheduling_strategy="SPREAD")`



Ray Scheduling Strategies

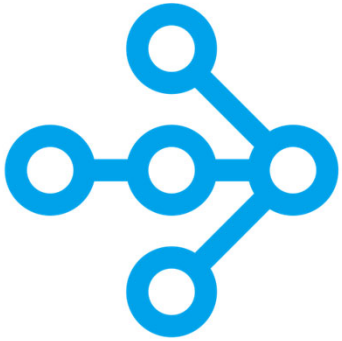
- **Default Strategy:** Schedules tasks or actors onto a group of the top k nodes.
  - Top:
    - (1) have large task arguments (input needed for the task) local
    - (2) have low resource utilization (for load balancing)
  - Within the top k group, nodes are chosen randomly
  - **K:** default is 20% of the total number of nodes.
- **Spread Strategy**: spread the tasks or actors among available nodes.
- **PlacementGroupSchedulingStrategy:** schedules the task or actor to where the placement group is located.
  - Placement groups allow users to atomically reserve groups of resources across multiple nodes (i.e., gang scheduling).
- **NodeAffinitySchedulingStrategy:** Schedules on a particular node specified by its node id
  - low-level strategy prevents optimizations by a smart scheduler
  - If node is Alive and feasible (even if not available currently), the task/actor will be scheduled to that node (when available)
  - Otherwise, the Soft parameter is checked:
    - Soft = True: schedule on another feasible node
    - Soft=False: task or actor will fail with **TaskUnschedulableError** or **ActorUnschedulableError**

```
# Only run the task on the local node.
node_affinity_func.options( scheduling_strategy=
ray.util.scheduling_strategies.NodeAffinitySchedulingStrategy(
node_id=ray.get_runtime_context().get_node_id(), soft=False, ) ).remote()
```

جامعة كارنيجي ميلون في قطر
Carnegie Mellon University Qatar

Outline

- Introduction
- Ray Programming & Computation Model
- Ray Cluster Architecture
- Ray Scheduling
- **Lifetime of a Ray Task**



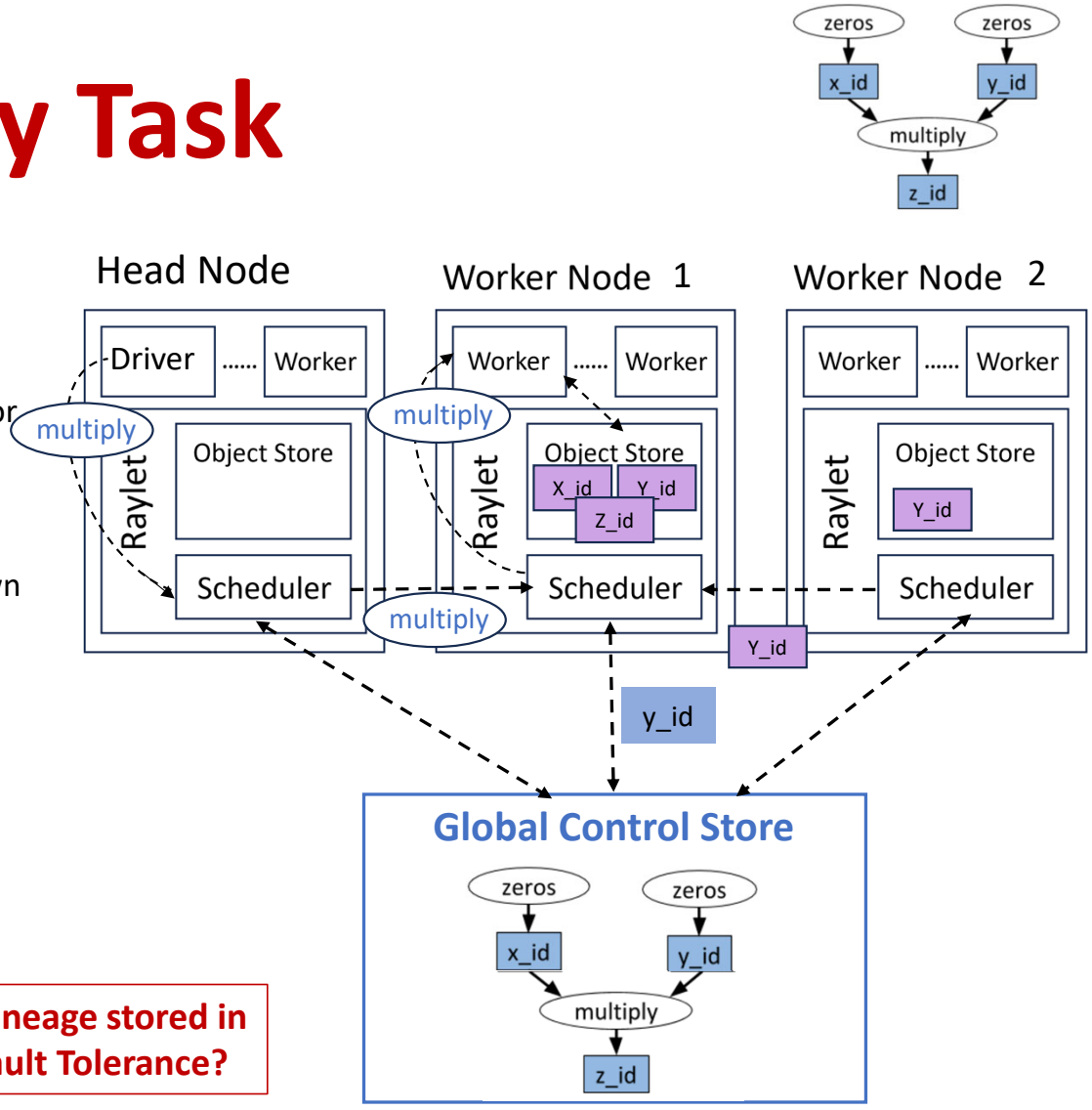
RAY

جامعة كارنيجي ميلون في قطر
Carnegie Mellon University Qatar

Lifetime of a Ray Task

- Driver submits the task to the local scheduler
- The scheduler logs the task to the GCS
- If the scheduler needs to retrieve the object data for one of the dependencies of the task, it will ask the GCS for the location of that object
- Then it is able to request that object data directly from the other node and receive the value in its own object store
- When all data dependencies are available at the node, the scheduler assigns the task to one of the workers
- The worker will return the value and store it in the object store
- The output value can be retrieved now using ray.get()

How do you think Tasks Lineage stored in the GCS can help with Fault Tolerance?



Recap

- Ray provides a general-purpose distributed compute framework for ML and distributed/parallel python applications
- Ray has an efficient API that builds on existing concepts (functions and classes) to allow turning a sequential program into a distributed one with relatively few lines of code
- Ray creates a dynamic task graph that represents the entire application
- Ray allows specifying resource requirements for tasks and allows different scheduling strategies
- Achieves scalability through different things such as:
  - Replicating global scheduler if it becomes a bottleneck
  - Distributed Object Store

جامعة كارنيجي ميلون في قطر
Carnegie Mellon University Qatar

Credit

- https://docs.ray.io/en/latest/
- Ray: a distributed framework for emerging AI applications
- https://www.usenix.org/system/files/osdi18-moritz.pdf
- https://bair.berkeley.edu/blog/2018/01/09/ray/
- https://www.usenix.org/system/files/osdi18-moritz.pdf
- https://venturebeat.com/ai/ray-the-machine-learning-tech-behind-openai-levels-up-to-ray-2-0/#:~:text=Over%20the%20last%20two%20years,OpenAI%20to%20Shopify%20and%20Instacart .
- https://medium.com/juniper-team/ray-distributed-computing-framework-for-ai-ml-applications-4b40617be4a3
- https://bair.berkeley.edu/blog/2018/01/09/ray/