Actor-Based Distributed Systems

Marek Konieczny

marekko@agh.edu.pl,

Room 4.43, Spring 2025

AGH
Akademia Górniczo-Hutnicza
im. Stanisława Staszica
w Krakowie

Class Logistics

- If you want to use your laptop during classes please install Ray Framework according to instructions:
  - *https://docs.ray.io/en/latest/ray-overview/installation.html*
  - Python 3.13 has alpha support
- Install all required components
  - *pip install -U "ray[default]”*
- Important *pip install ray* might not enough !!!

Class Logistics

- Laboratory classes during week 14-25.4
  - There is no meeting on Easter break
- We will use desktops from Computer Networks Laboratory
  - Select Ubuntu image
- Preliminary plan homework assignment
  - Elastic schedule on week 9-13.5 (?)

Class Logistics

- Laboratory grading
  - Showing up => 1 pt
  - UPEL task upload => 1 pt
  - Hard work on lab meeting => 2 pt
    - You need to show me logs of your tasks execution on Ray Dashboard
  - Extra activity, labs improvement => 1 pt

Actors (theory)

- Mathematical model of concurrent computation proposed by Carl Hewitt in 1973

Session 8 Formalisms for Artificial Intelligence

A Universal Modular ACTOR Formalism for Artificial Intelligence

Carl Hewitt
Peter Bishop
Richard Steiger

Abstract

This paper proposes a modular ACTOR architecture and definitional method for artificial intelligence that is conceptually based on a single kind of object: actors [or, if you will, virtual processors, activation frames, or streams]. The formalism makes no presuppositions about the representation of primitive data structures and control structures. Such structures can be programmed, micro-coded, or hard wired in a uniform modular fashion. In fact it is impossible to determine whether a given object is "really" represented as a list, a vector, a hash table, a function, or a process. The architecture will efficiently run the coming generation of PLANNER-like artificial intelligence languages including those requiring a high degree of parallelism. The efficiency is gained without loss of programming generality because it only makes certain actors more efficient; it does not change their behavioral characteristics. The architecture is general with respect to control structure and does not have or need goto, interrupt, or semaphore primitives. The formalism achieves the goals that the disallowed constructs are intended to achieve by other more structured methods.

PLANNER Progress

Actors (theory)

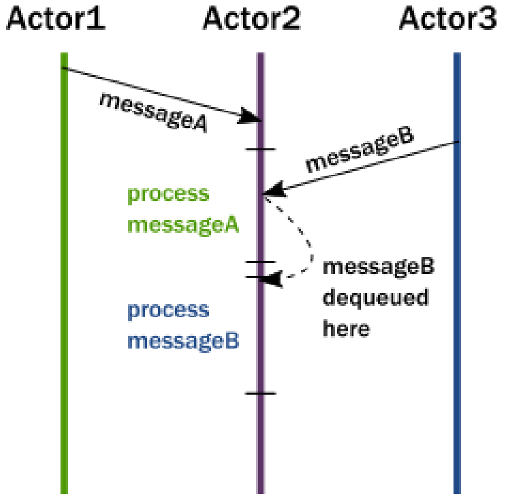
- Primitives for concurrency/parallelism
- Entities having a message queue and associated behavior → And isolated state!!
- Can exchange messages between each-other
- When an actor receives a message it can:
  - Send a finite number of messages to other actors
  - Create a finite number of new actors
  - Modify its interval behavior on receiving messages

Actors (theory)

- Messages between actors are always sent asynchronously
- No requirement on order of message arrival
- Queuing and dequeuing of messages in an actor mailbox are atomic operations
- Fire and Forgot approach, no race conditions anymore

Actors (theory)

- The actor is an object that encapsulates state and behavior
- Originated from
  - Smalltalk, Petri Nets, Channels ...
  - More details on theory in attached document
- More recent
  - *Scheme* made them concrete
  - *Erlang* made them useful
  - *Akka* made them cool
  - *Ray* makes them easy



Origins

- Originated on UC Berkeley
- Group led by Ion Stoica:
  – Apache Messos, Apache Spark (Databricks), Apache Alluxio (Alluxio)
  – Ray is managed by Anyscale

Ray: A Distributed Framework for Emerging AI Applications

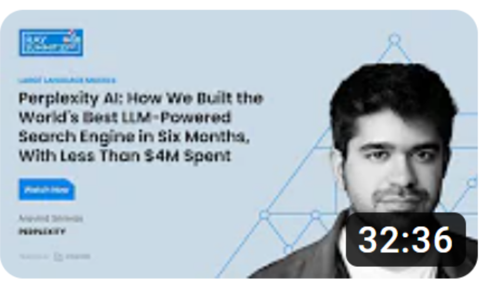
Philipp Moritz,* Robert Nishihara,* Stephanie Wang, Alexey Tumanov, Richard Liaw,
Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I. Jordan, Ion Stoica
University of California, Berkeley

Origins

The next generation of AI infrastructure is being built and scaled on Ray

OpenAI Uber Spotify cohere instacart DOORDASH

NETFLIX samsara LinkedIn NIANTIC



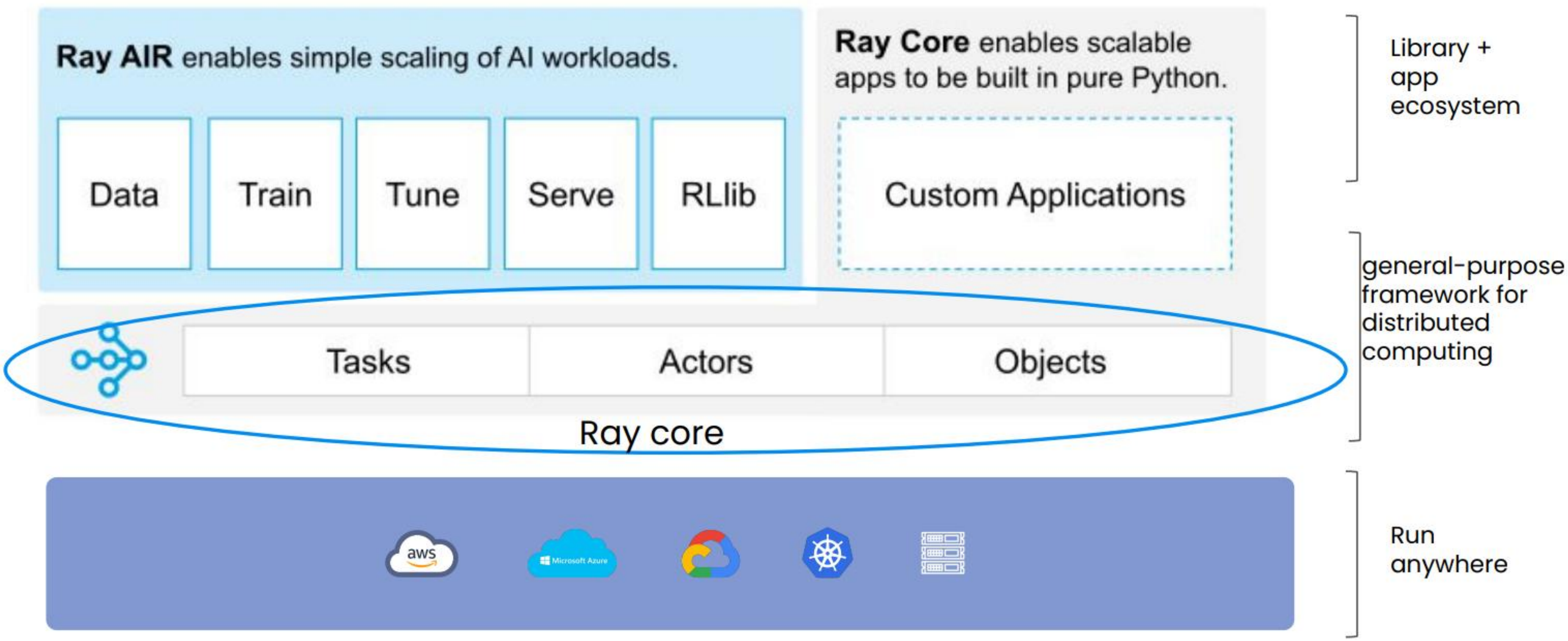
Perplexity AI: How We Built the World's Best LLM-Powered Search Engine in 6 Months, w/ Less Than $4M

Anyscale • 18K views • 5 months ago

The FAST Compute Model

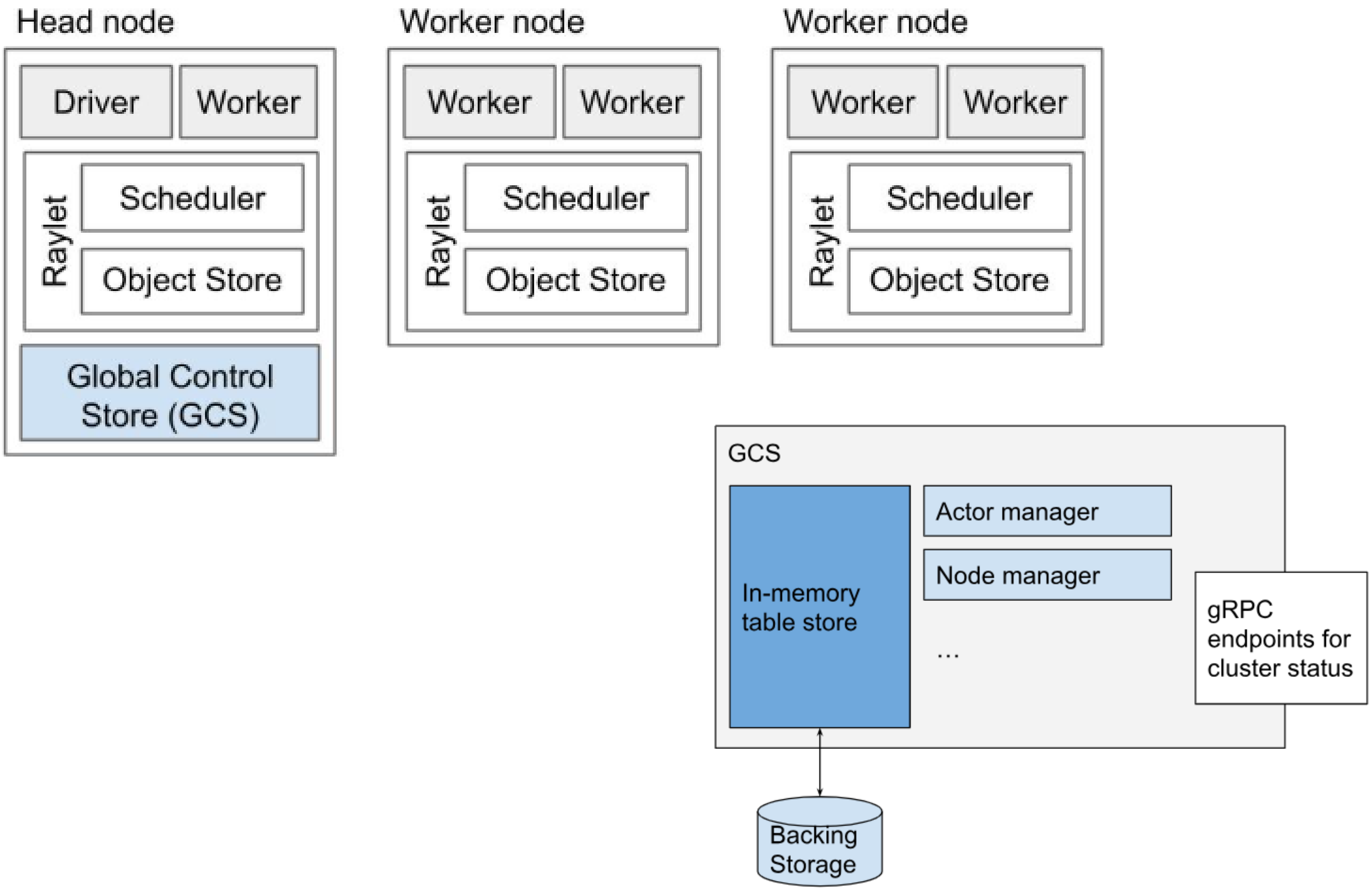
- Implementation of serverless approach in computing (including multi-cloud)
- Most important features
  - **F**utures: references to objects
  - **A**ctors: remote class instances (objects)
  - **S**hared in-memory distributed object store
  - **T**asks: remote functions

What is Ray ?



Ray AIR enables simple scaling of AI workloads.
Data
Train
Tune
Serve
RLlib
Ray Core enables scalable apps to be built in pure Python.
Custom Applications
Tasks
Actors
Objects
Ray core
aws
Microsoft Azure
Library + app ecosystem
general-purpose framework for distributed computing
Run anywhere

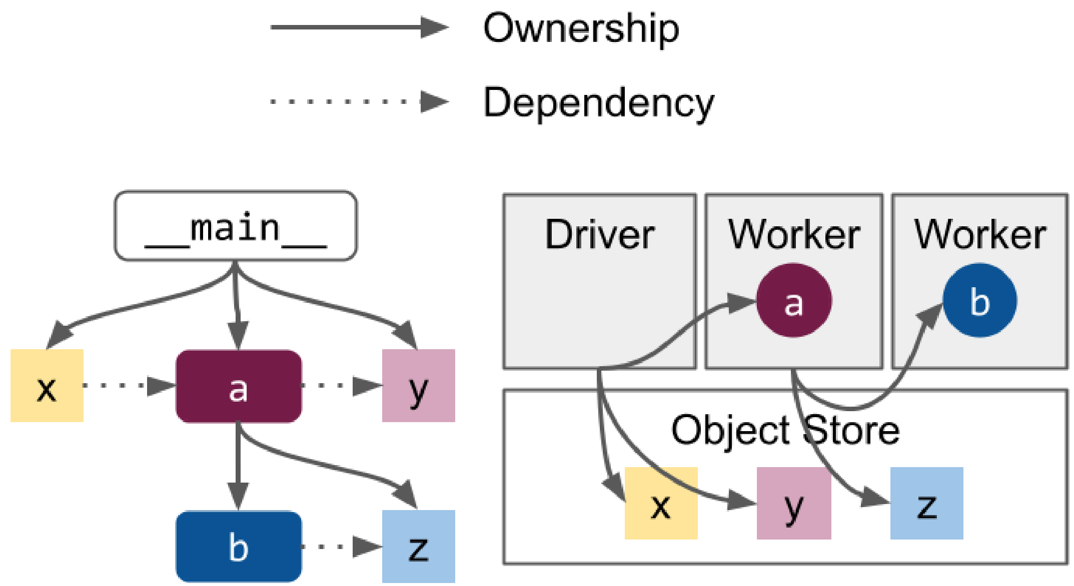
What is Ray ?


Head node
Driver
Worker
Raylet
Scheduler
Object Store
Global Control Store (GCS)
Worker node
Worker
Worker
Raylet
Scheduler
Object Store
Worker node
Worker
Worker
Raylet
Scheduler
Object Store
GCS
In-memory table store
Actor manager
Node manager
...
gRPC endpoints for cluster status
Backing Storage

What is Ray ?

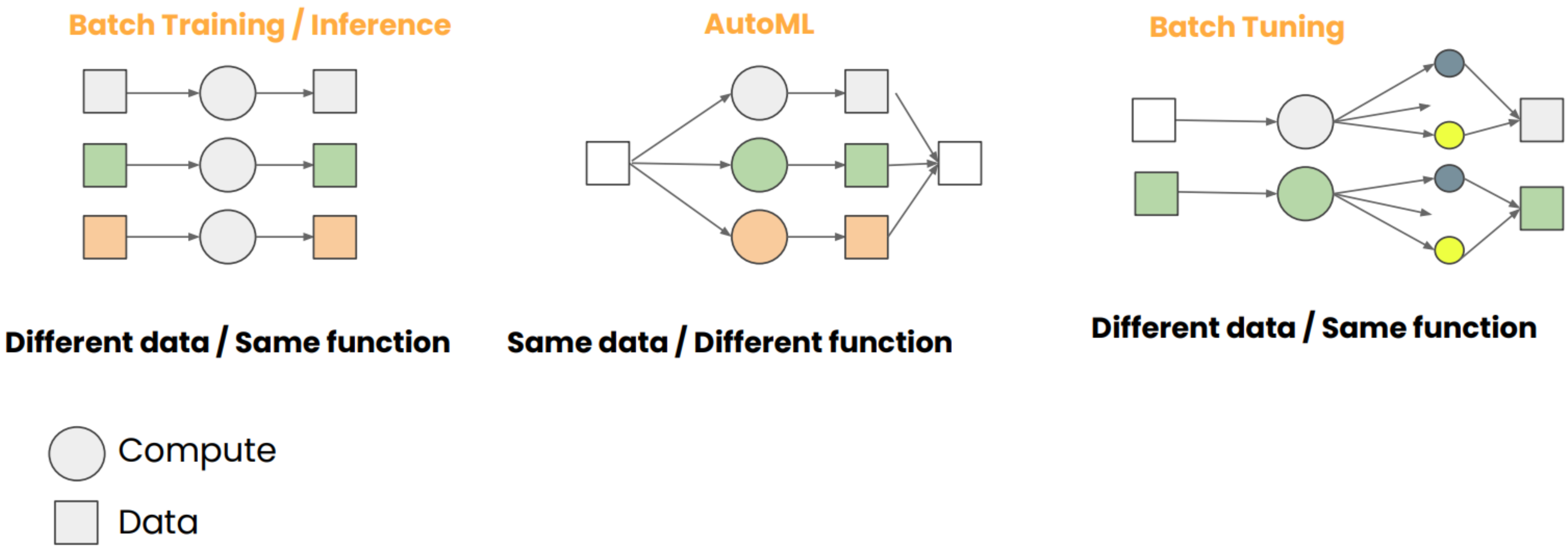
```
@ray.remote
def b():
  return

@ray.remote
def a(dep):
  z = b.remote()

x = ray.put(...)
y = a.remote(x)
```

Program



Task graph **Physical execution**

Scaling patterns



What is Ray ?

Python → Ray APIs

```
def f(x):
    # do something with x:
    y= ...
    return y
```

Task

```
@ray.remote
def f(x):
    # do something with x:
    y= ...
    return y
```

Distributed

f() Node ••• f() Node

```
class Cls():
  def __init__(self,
x):
  def f(self, a):
        ...
  def g(self, a):
        ...
```

Actor

```
@ray.remote
class Cls():
  def
__init__(self, x):
  def f(self, a):
        ...
  def g(self, a):
        ...
```

Distributed

Cls Node ••• Cls() Node

```
import numpy as np
a= np.arange(1, 10e6)
b = a * 2
```

Distributed immutable object

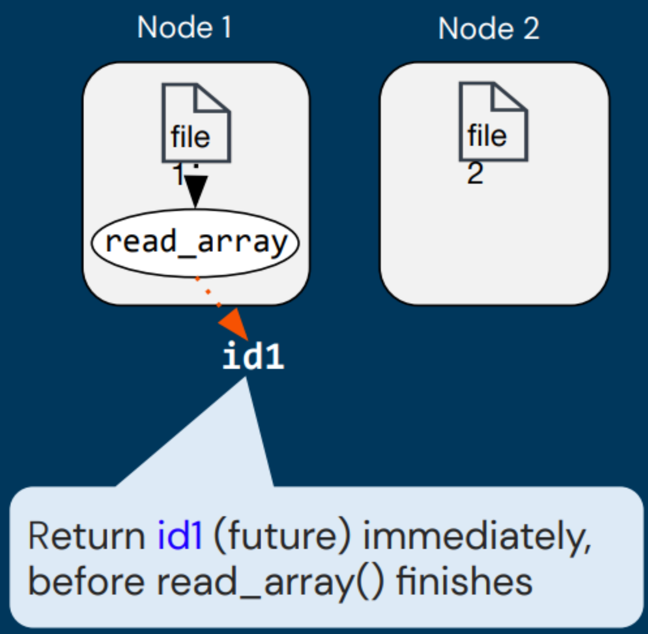
```
import numpy as np
a = np.arange(1, 10e6)
obj_a = ray.put(a)
b = ray.get(obj_a) * 2
```

Distributed

a Node ••• a Node

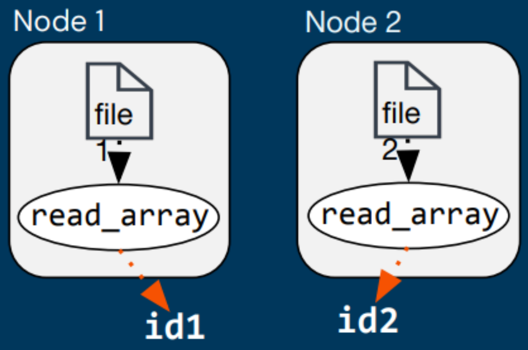
Task API

```
@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)
```

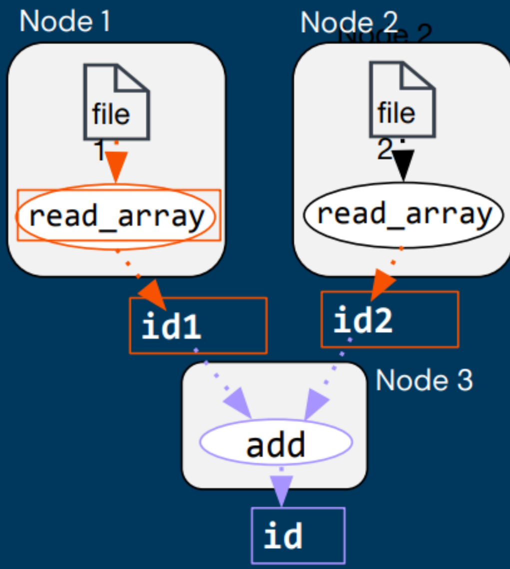


Dynamic task graph: build at runtime

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

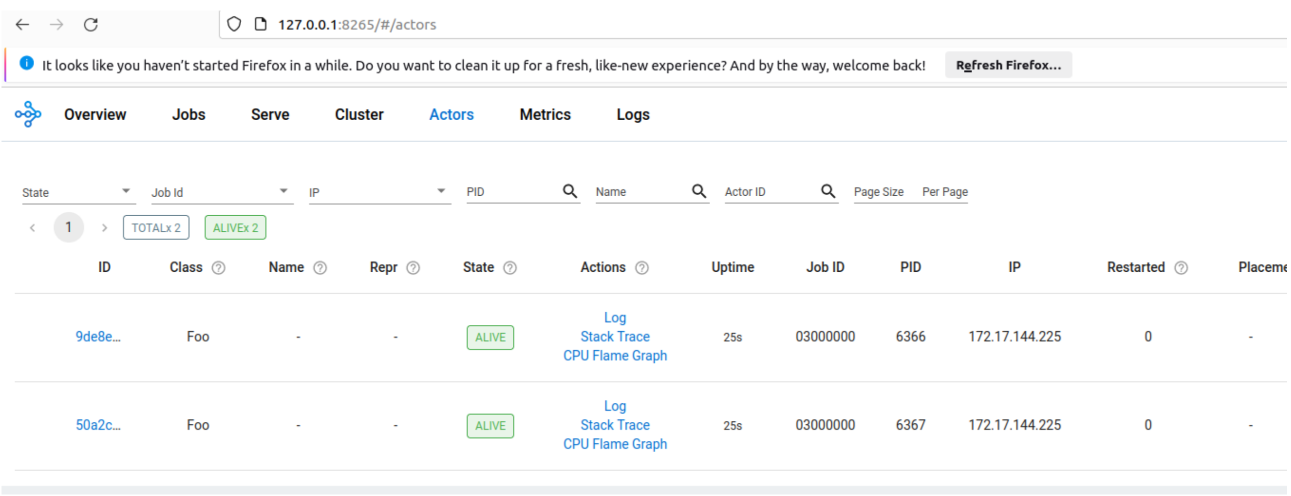


Task 0

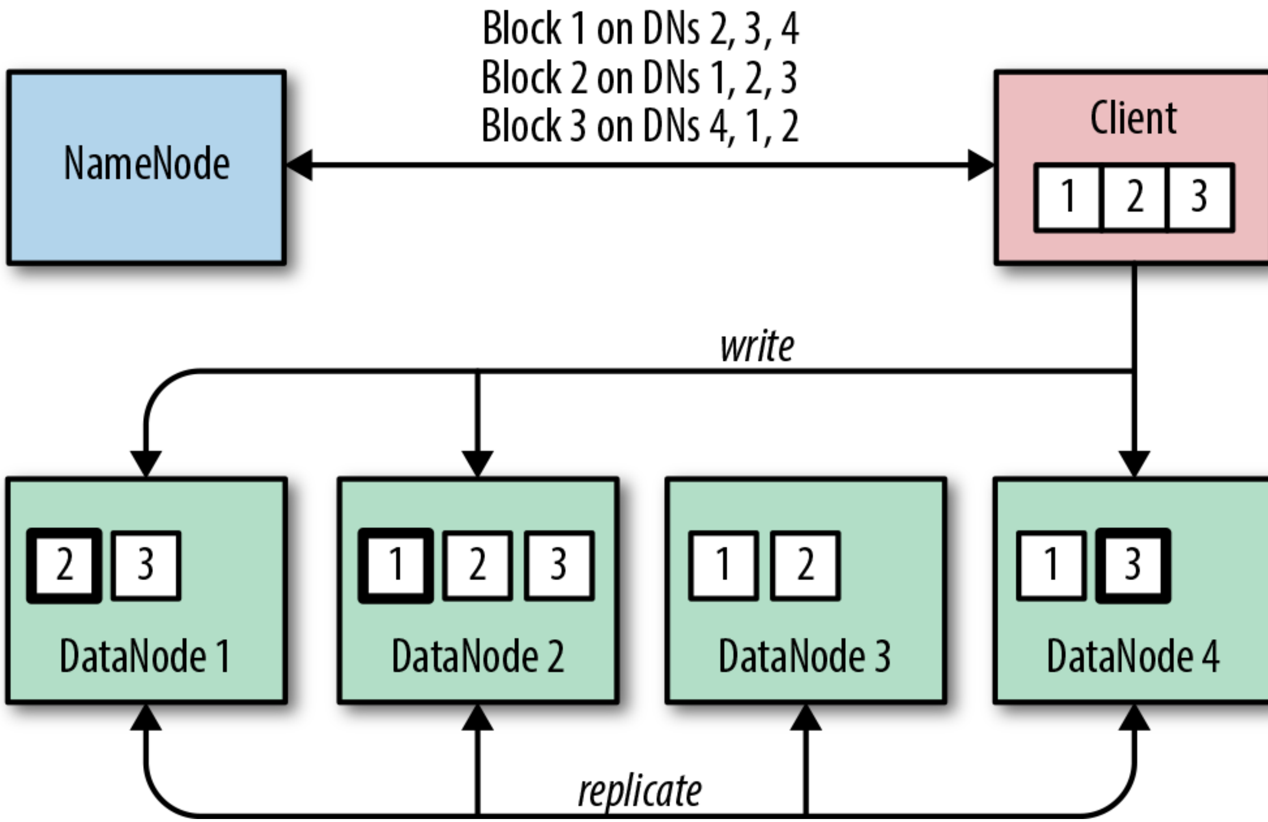
- Start a node
  - *ray start --head --port=6379*
- Check the dashboard on :
  - http://127.0.0.1:8265/
- Copy notebook to folder and start Jupyter
  - *jupyter notebook*

Task 1

- Open and fill all the gaps in attached notebook
  - All important aspects of ray framework are covered
  - At the end there is main pi computation task
- Your dashboard should look like:



Homework


Block 1 on DNs 2, 3, 4
Block 2 on DNs 1, 2, 3
Block 3 on DNs 4, 1, 2
NameNode
Client
1
2
3
write
2
3
DataNode 1
1
2
3
DataNode 2
1
2
DataNode 3
1
3
DataNode 4
replicate

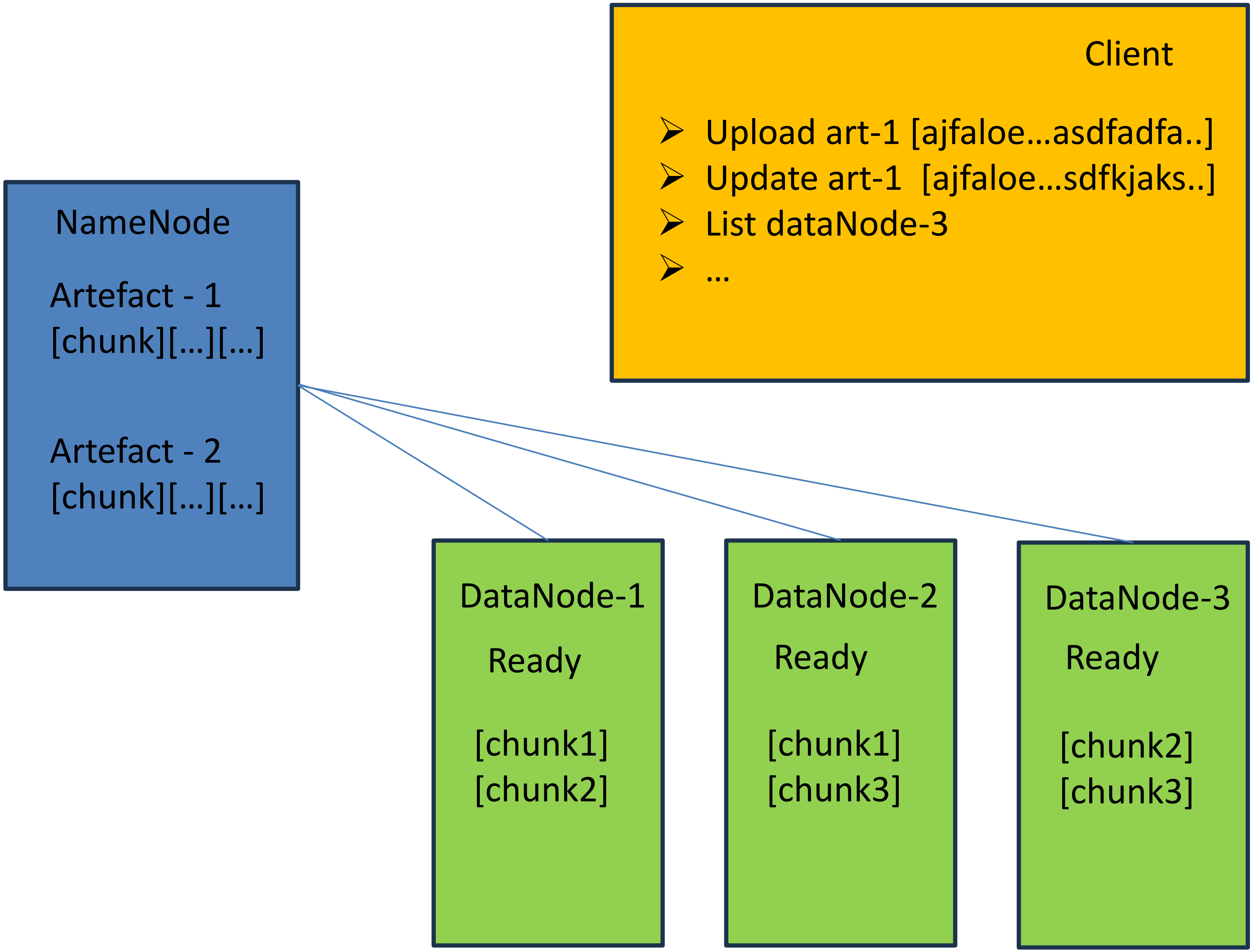
Homework

- Implement a Distributed Artifacts Storage system:
  - there is one name node and x storage nodes (all actors)
  - execute it in a separate script in the system terminal/console
  - name node distributes artifacts to storage nodes in chunks/blocks and keeps track of them
  - there are few copies of chunks in the system to improve high availability and speed (similarly as in HDFS)
  - randomly, data storage nodes can fail; the name node keeps track of all chunks and manages the copies (everything should be consistent)
  - provide a list operation of the most important data (e.g. where chunks/blocks are stored)

Homework

- User can perform the following operations
  - Artifact consist of name and content (stored as string)
  - upload - by providing a new artifact
  - update the stored artifact content
  - user can delete artifacts present in the storage
  - user can get/download the artifacts from the storage
  - user can list the status of the cluster

Homework


Client
Upload art-1 [ajfaloe…asdfadfa..]
Update art-1 [ajfaloe…sdfkjaks..]
List dataNode-3
…
NameNode
Artefact - 1
[chunk][…][…]
Artefact - 2
[chunk][…][…]
DataNode-1
Ready
[chunk1]
[chunk2]
DataNode-2
Ready
[chunk1]
[chunk3]
DataNode-3
Ready
[chunk2]
[chunk3]

Homework

- Verification tests:
  - upload an artifact, list the status
  - update an artifact, list the status
  - get an artifact, list the status
  - delete an artifact, list the status

Homework

- Tier 1 (3 pts)
  - Complete and present all exercises from Task 1 => 3 pts
- Tier 2 (6-10 pts)
  - Prepare implementation of the Task 2 from lab
  - Present demo solution and test scenarios => 6 pts
  - Add some additional advanced functionality, select anything from items not discussed on labs (https://docs.ray.io/en/latest/ray-core/actors.html) => 7-9 pts
  - Add distribution of environment (e.g. execute it on cluster based on K8s, docker …) => 10 pts

Homework

- Tier 3 (9 - 10 pts)
  - Prepare/migrate sample ML project with other Ray modules, use 3 different modules
  - Please use unique data sets:
    - No MIST, Iris …
  - Prepare tests for your solution
  - Describe how you move it from regular implementation to demo on top of Ray
- Execute them on top of distributed environment (+ 1 pt) => 10 pts