# Actor-based Distributed Programming

Marek Konieczny

marekko@agh.edu.pl,

Room 4.43, Spring 2023

AGH

AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA
W KRAKOWIE

# Class Logistics

- Laboratory classes during week 28-31.4
- We will use desktops from Computer Networks Laboratory
- Preliminary plan homework assignment
  - Elastic schedule on week 25-29.4
  - There will be separate tasks for different grades: 5-7, 8, 9-10

# Class Logistics

- Log in to Windows SR images
- Install all required components
  - *pip install -U "ray[default]"*

# Actors

- Actors→ primitives for concurrency/parallelism
- Actors → Entities having a message queue and associated behavior → And isolated state!!
- Actors→ Can exchange messages between each-other
- Actors → 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

- 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, no race conditions anymore

# Actors

- Mathematical model of concurrent computation proposed by Carl Hewitt in 1973
- The actor is an object that encapsulates state and behavior

Session 8 Formalisms for Artificial Intelligence

A Universal Modular ACTOR Formalism for Artificial Intelligence

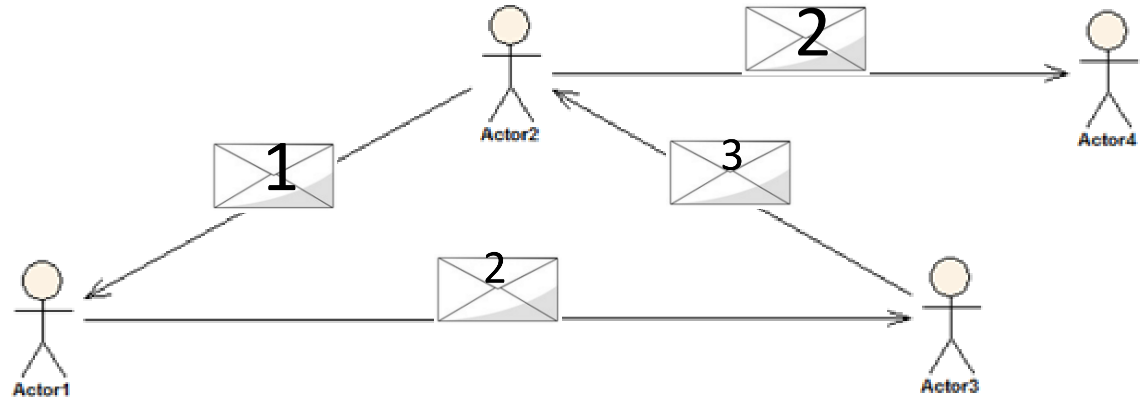
Carl Hewitt
Peter Bishop
Richard Steiger

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



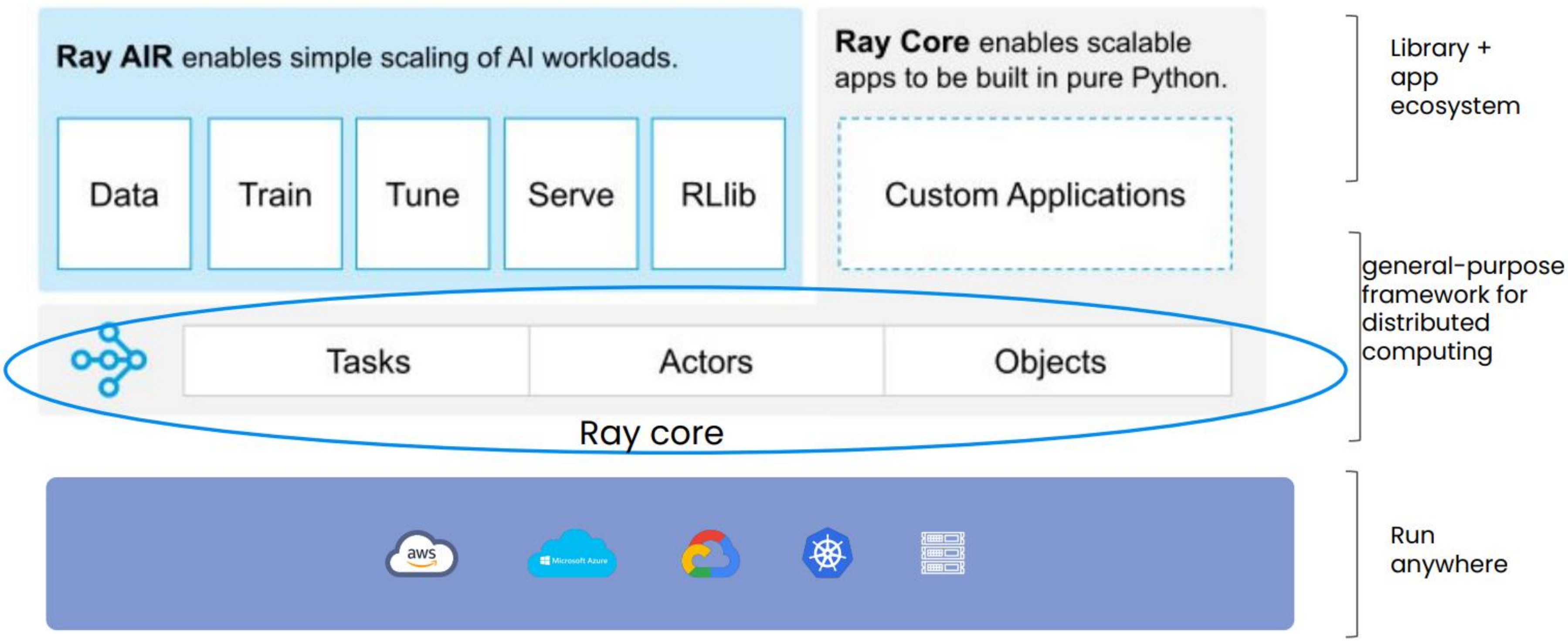

- When…
  - The problem to be solved can be decomposed into a set of independent tasks
  - The problem to be solved can be decomposed into a series of tasks linked by a clear flow
- In short words... when the problem can be parallelized

# Advantages and Drawbacks

- Extends the benefits of object-oriented programming by splitting control flow and business logic
- Allows to decompose a system into interactive, autonomous and independent components that work asynchronously

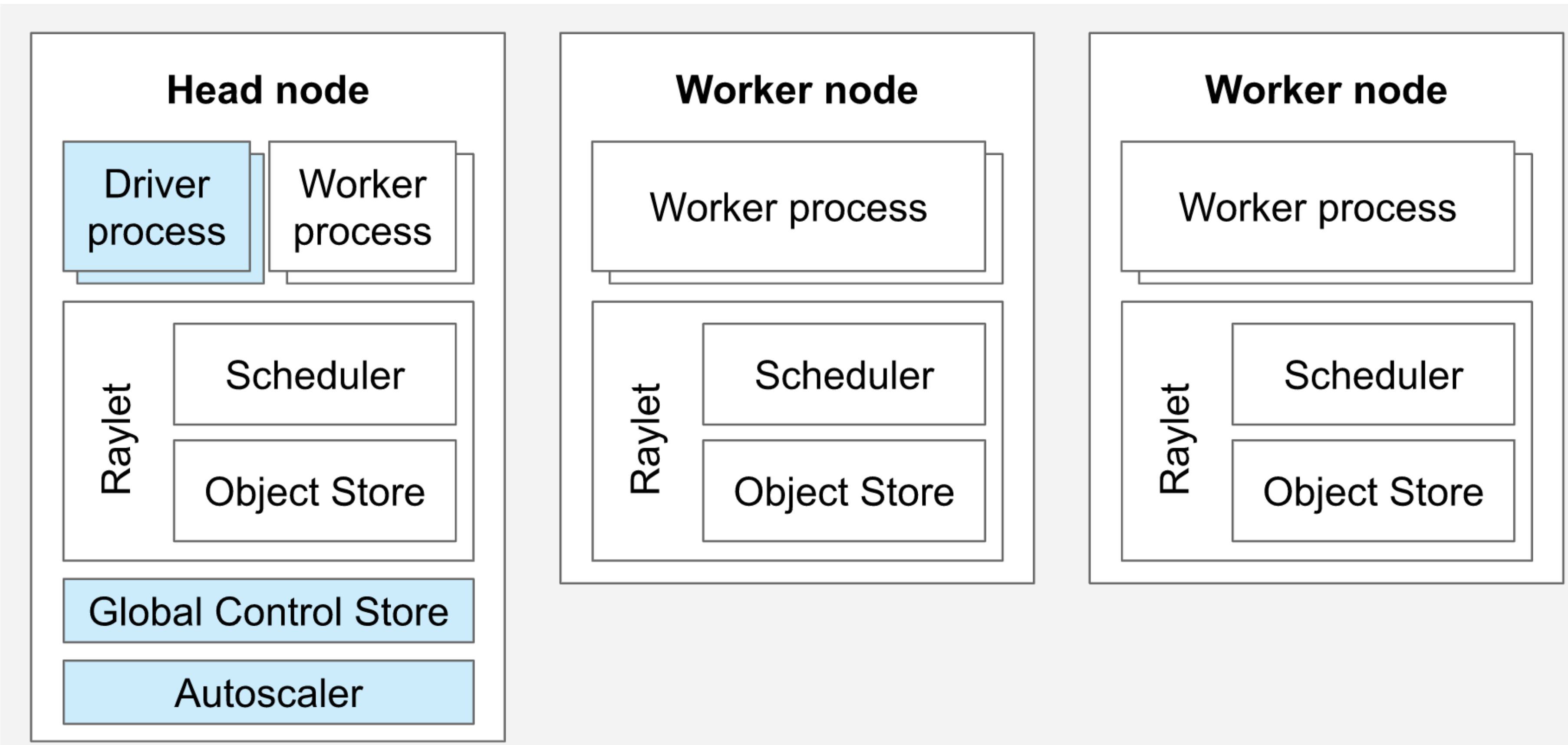
- Sometimes creating actors may dramatically affect the system's responsiveness
- The decision of where to store and run the new actors requires to archive a series of records, so if it is not done well it can lead to performance penalties in highly distributed systems

# 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

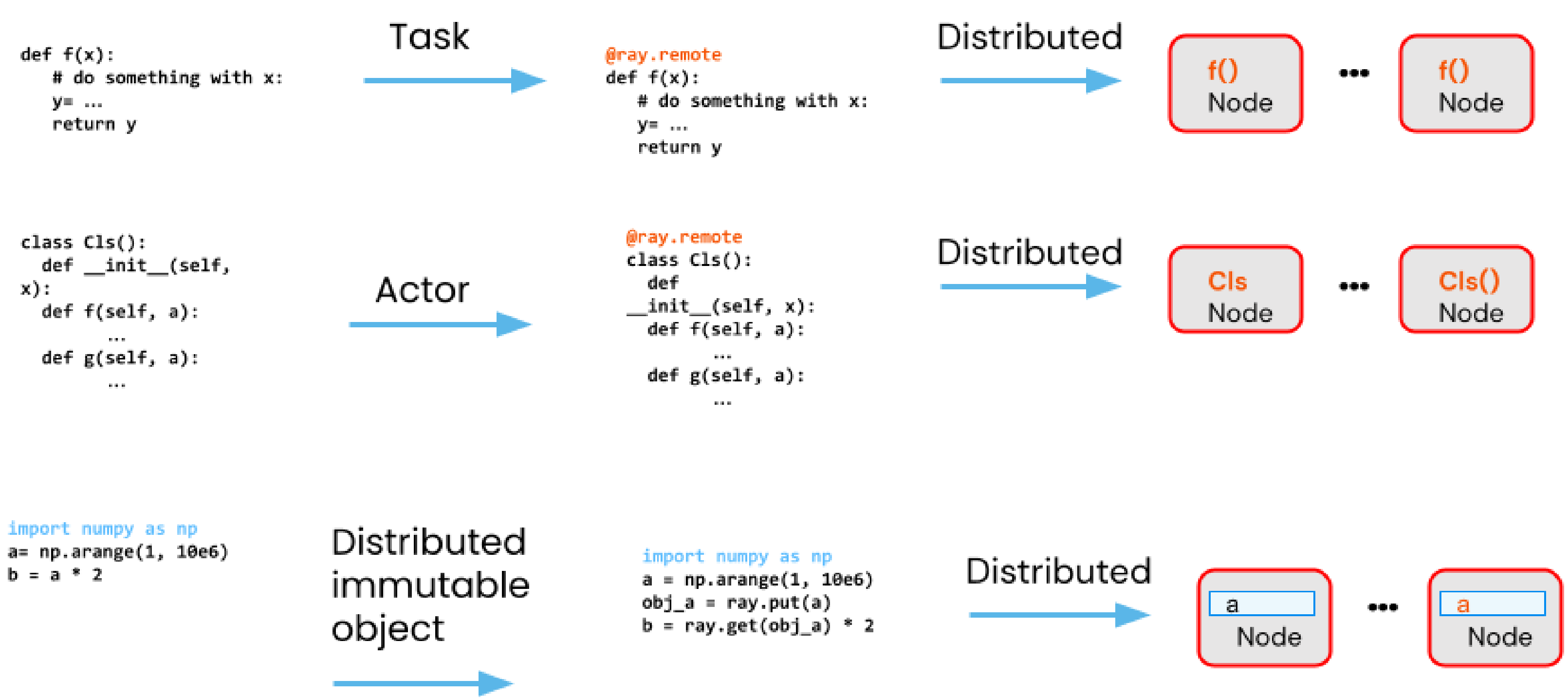
# What is Ray ?



Head node
Driver process
Worker process
Raylet
Scheduler
Object Store
Global Control Store
Autoscaler
Worker node
Worker process
Raylet
Scheduler
Object Store
Worker node
Worker process
Raylet
Scheduler
Object Store

# What is Ray ?

## Python → Ray APIs

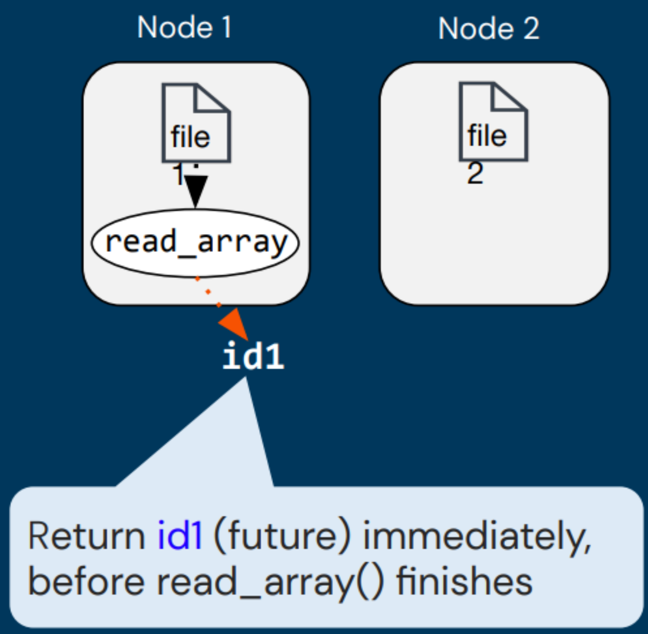


# Task API

```
@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)
```

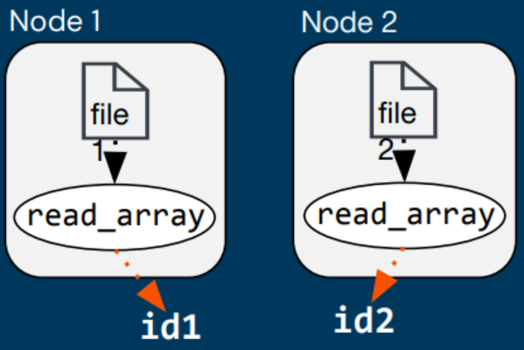


# Task API

```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)
```




Dynamic task graph: build at runtime

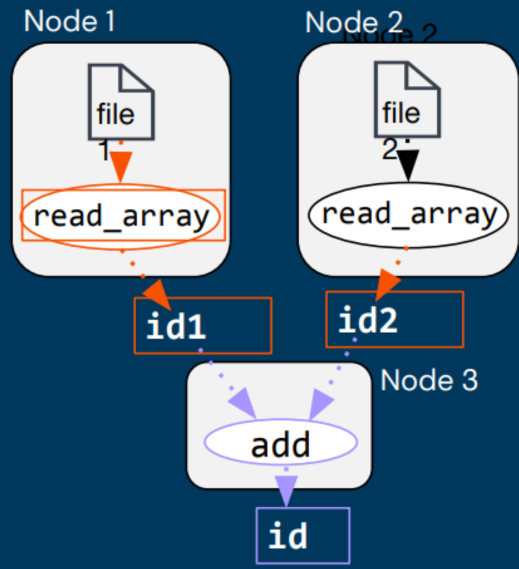
# Task API

```
@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 API

## Function → Task

```
@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

```
@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()
```

# Task 0

- Connect to head of cluster
  – *ray start --address=‘<cluster-head>:6379' --node-ip-address=<your-ip-address>*
- For environment init add following params
  – *ray.init(address='auto', ignore_reinit_error=True, logging_level=logging.ERROR)*
- Env setup:
  – http://172.17.144.218:8265/

# Task 1

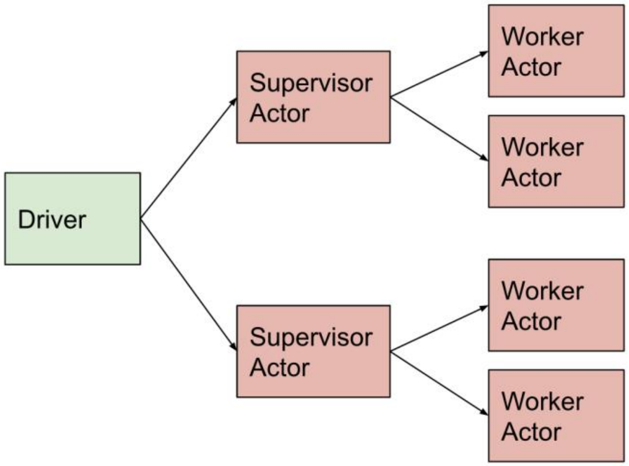
- Perform and analyze all task related exercises from
  - *ray-lab-1-task.py*
- Perform parallel bubble sort in ray

# Task 2

- Perform and analyze all task related exercises from
  - *ray-lab-2-objects.py*
- Perform sample modification of objects in ray

# Task 3

- Perform and analyze all task related exercises from
  - *ray-lab-3-actors.py*
- Perform exercises from the end of the file
- Add Pi counting with actors



# Homework

- Tier 1 (5-7 pts)
  – Prepare and present all exercises from labs
- Tier 2 (8 pts)
  – Prepare and present all exercises from lab and execute them ion the local ray cluster based on docker compose

# Homework

- Tier 3 (9-10 pts)
  - Prepare sample project with other  Ray modules
  - Select 3 different modules
  - Prepare tests form your solution
  - Execute them on top of distributed environment.