Raythena: *job scheduling on next generation HPCs*

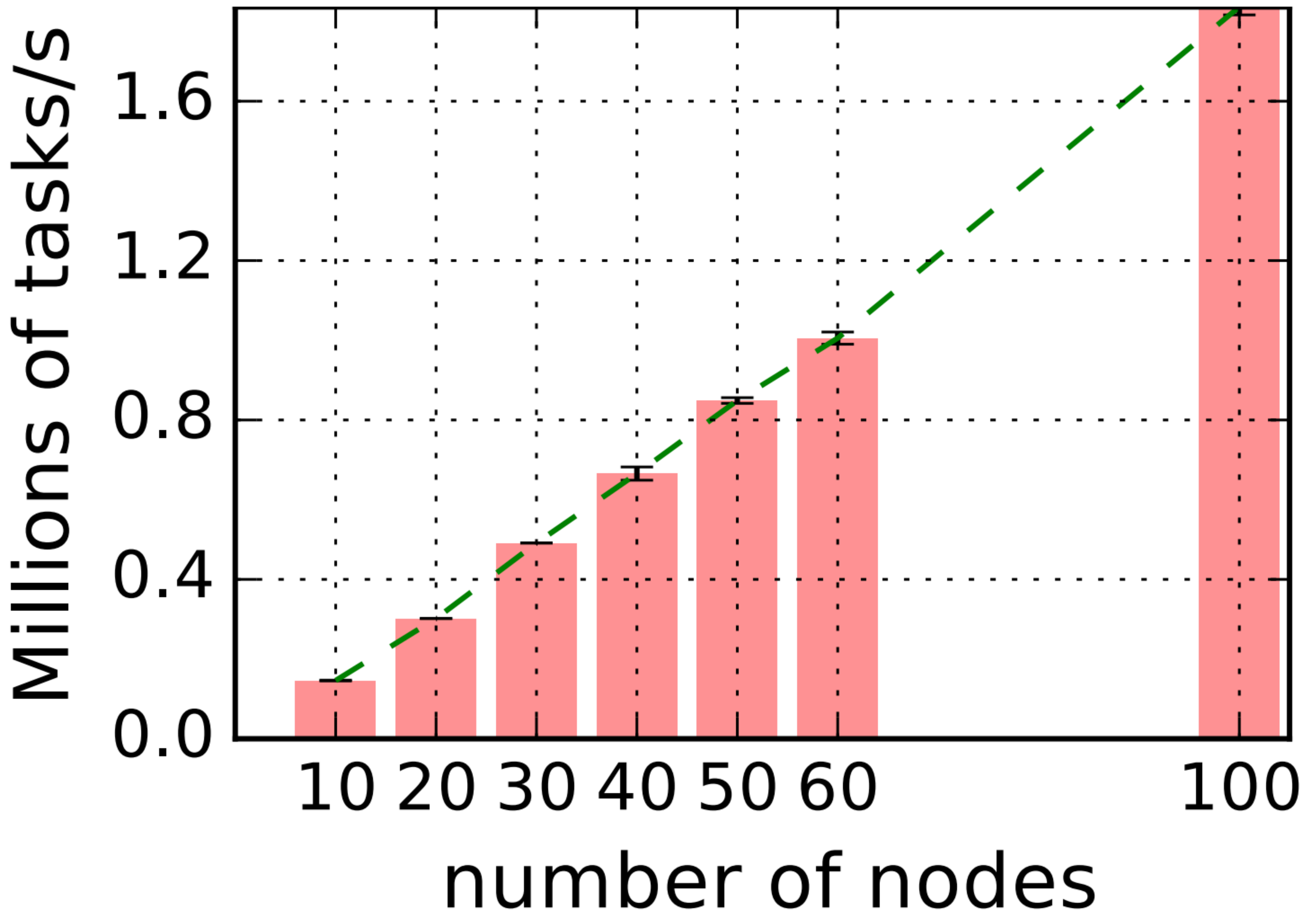
Paolo Calafiura, Vakho Tsulaia, Charles Leggett,
Illya Shapoval, Julien Esseiva, Rollin Thomas,
Miha Muškinja

4th US ATLAS HPC Meeting
Thursday 26 September 2019
LBNL

Introduction

- In view of the increasing availability of heterogeneous HPCs we are exploring the applicability of a modern *distributed execution framework* for ATLAS workflows— Ray[1],
- Ray was originally designed for end-to-end large-scale AI applications and aims to simplify complex parallel systems using a simple Python interface,
  - It allows the user to easily express parallelism while also capturing data dependencies,
  - Efficiently handles processing of large datasets across many compute nodes.
- New project for us and it was presented for the first time at the ATLAS S&C Week in NY this year in June.

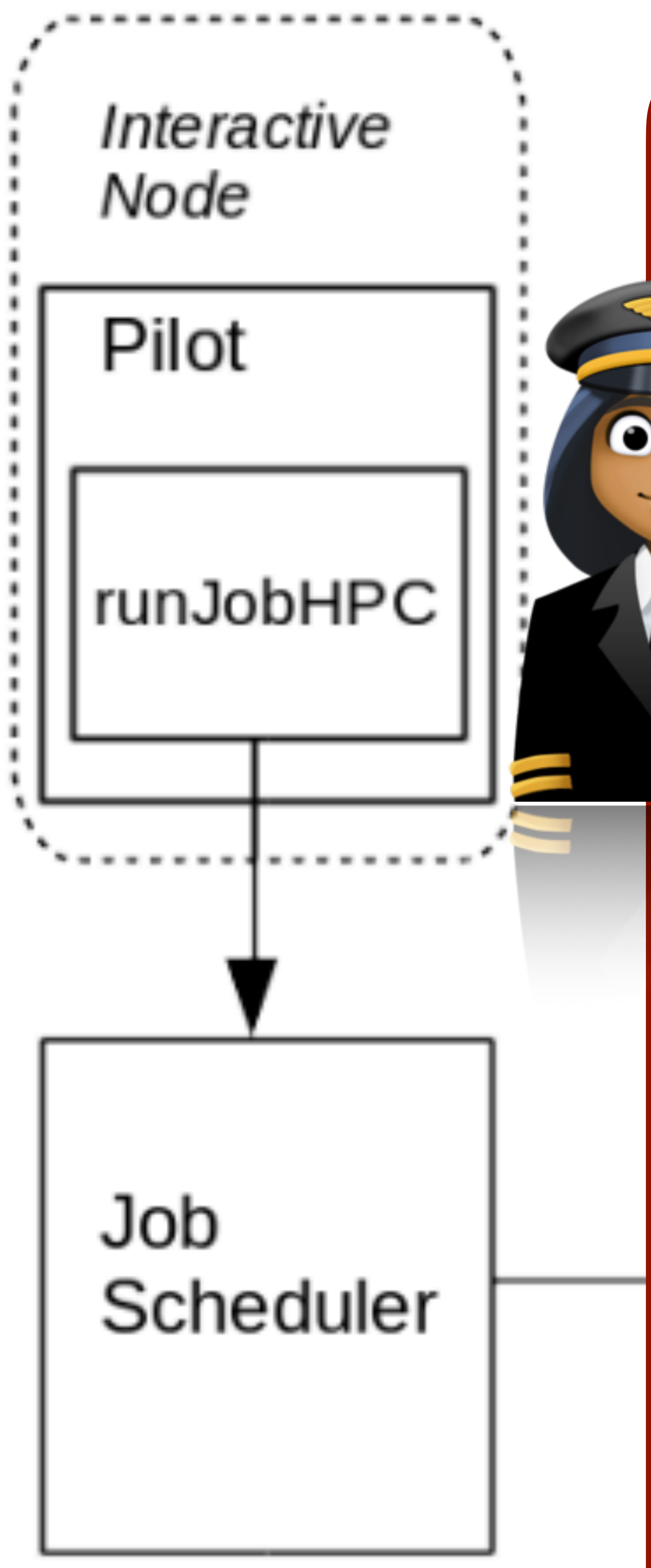
[1]: https://github.com/ray-project/ray

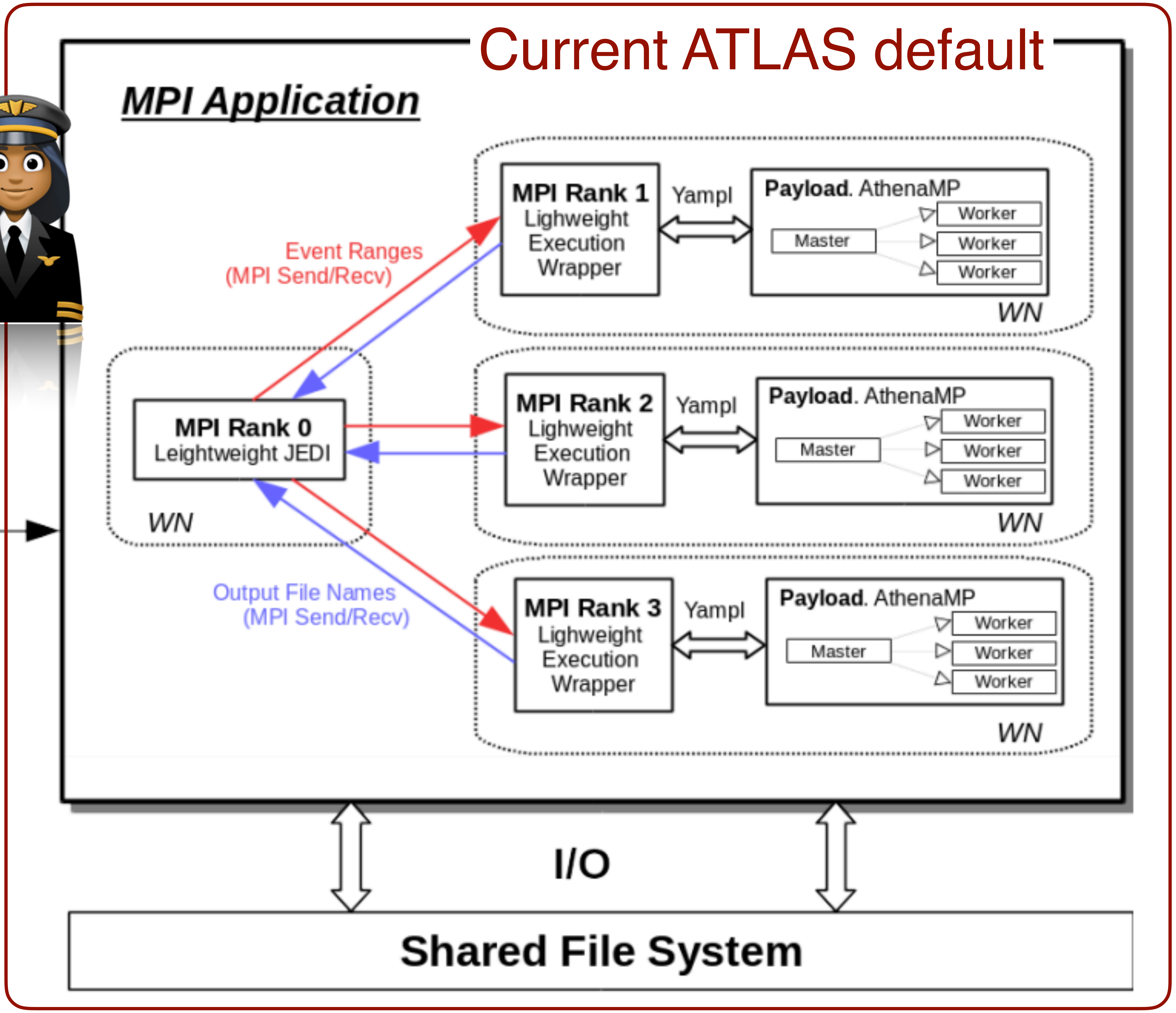


Advantages of using Ray for ATLAS workflows

- Ray is widely used by the broader community and centrally maintained. Using Ray would eliminate the need of supporting some of the ATLAS home-built software,
- Developed by RiseLab at UC Berkeley. We are collaborating directly with the developers (Ion Stoica, et. al.),
- Proven to be scalable on HPCs,
- Ray is lightweight and easy to install (e.g. as a module on an HPC),
- Ray fits well into the modular scheme of Run 3 job scheduling as an intermediate layer. Once we have the Ray workflow, we could also replace Ray with any other *task parallel* framework (e.g. Dask, Spark).

First application: ATLAS Event Service with Ray— Raythena



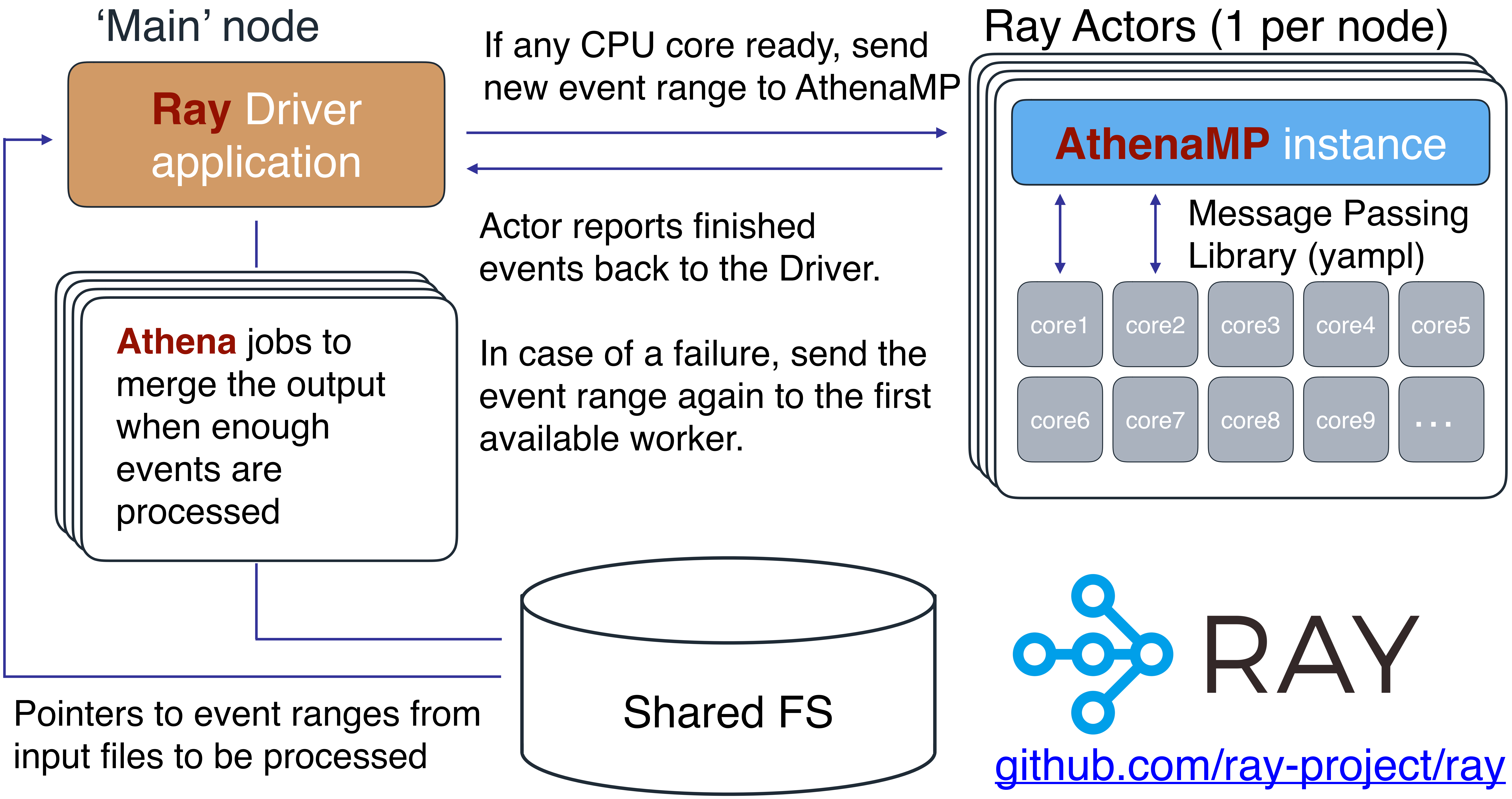

Current ATLAS default
MPI Application
Event Ranges (MPI Send/Recv)
Output File Names (MPI Send/Recv)
MPI Rank 0 Leightweight JEDI
WN
MPI Rank 1 Lighweight Execution Wrapper
Yampl
Payload. AthenaMP
Master
Worker
Worker
Worker
WN
MPI Rank 2 Lighweight Execution Wrapper
Yampl
Payload. AthenaMP
Master
Worker
Worker
Worker
WN
MPI Rank 3 Lighweight Execution Wrapper
Yampl
Payload. AthenaMP
Master
Worker
Worker
Worker
WN
I/O
Shared File System

We implemented a prototype of this scheme with **Ray**.

Figure 1. Schematic view of Yoda

iopscience/10.1088/1742-6596/664/9/092025/pdf

Ray-based ATLAS Event Service: Raythena



If any CPU core ready, send new event range to AthenaMP

Actor reports finished events back to the Driver.

In case of a failure, send the event range again to the first available worker.

Message Passing Library (yampl)

Athena jobs to merge the output when enough events are processed

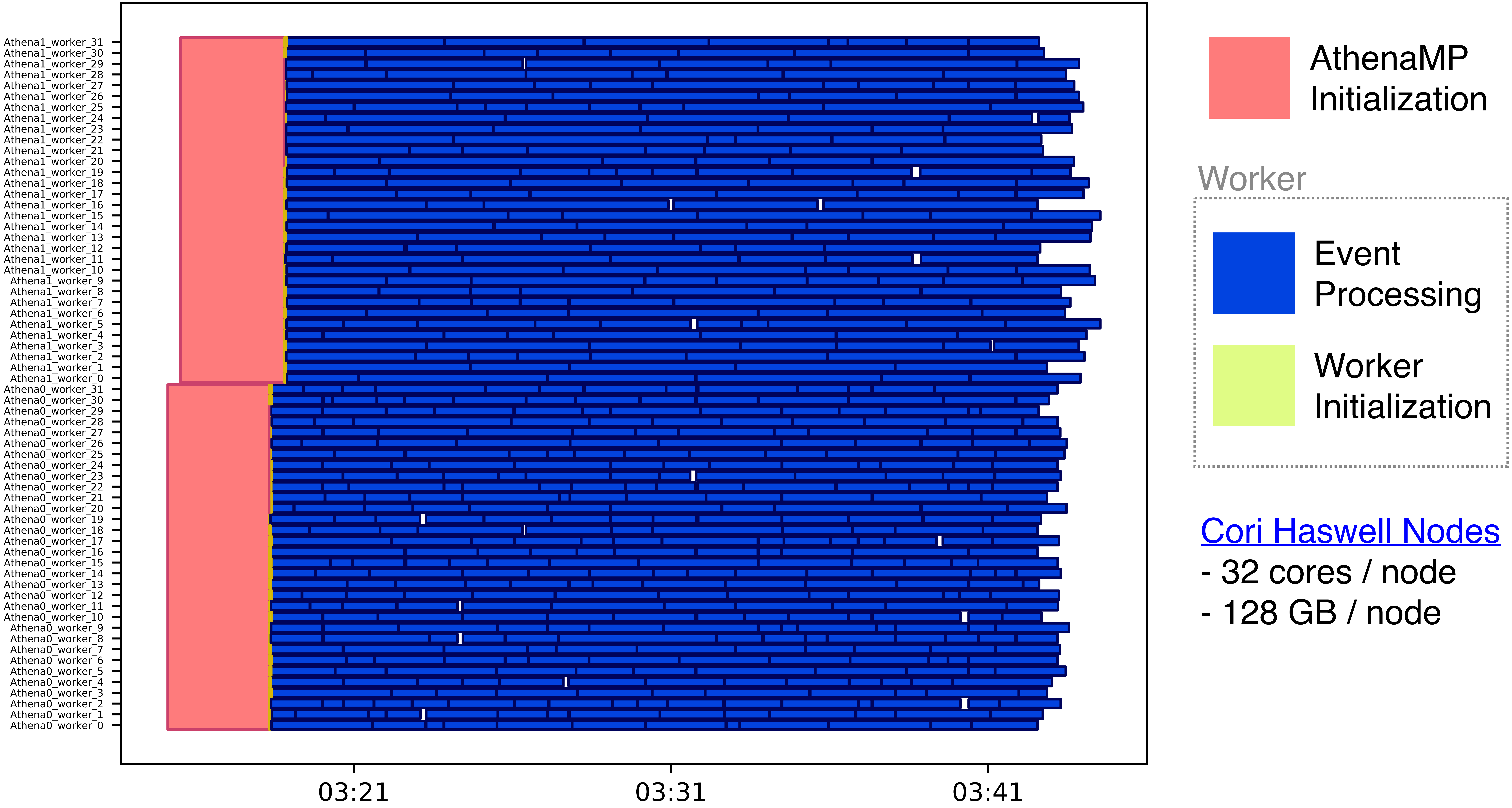
Pointers to event ranges from input files to be processed

github.com/ray-project/ray

Working example on Cori at NERSC

- Successfully tested the Raythena workflow on Cori **Haswell** and **KNL** nodes at NERSC,
- We are running ID-only simulation jobs (for faster turnaround) with recent master nightlies,
- Athena merge jobs are spawned on-the-fly with `HITSMerge_tf`,
- Largest test that we tried so far:
  - 60 Haswell nodes with 32 cores each,
  - 100k events to process with 1 event pre ‘Event Range’,
  - Spawn merge jobs every 100 events to form 1000 merged HITS files.
- No bottlenecks found so far in Ray.

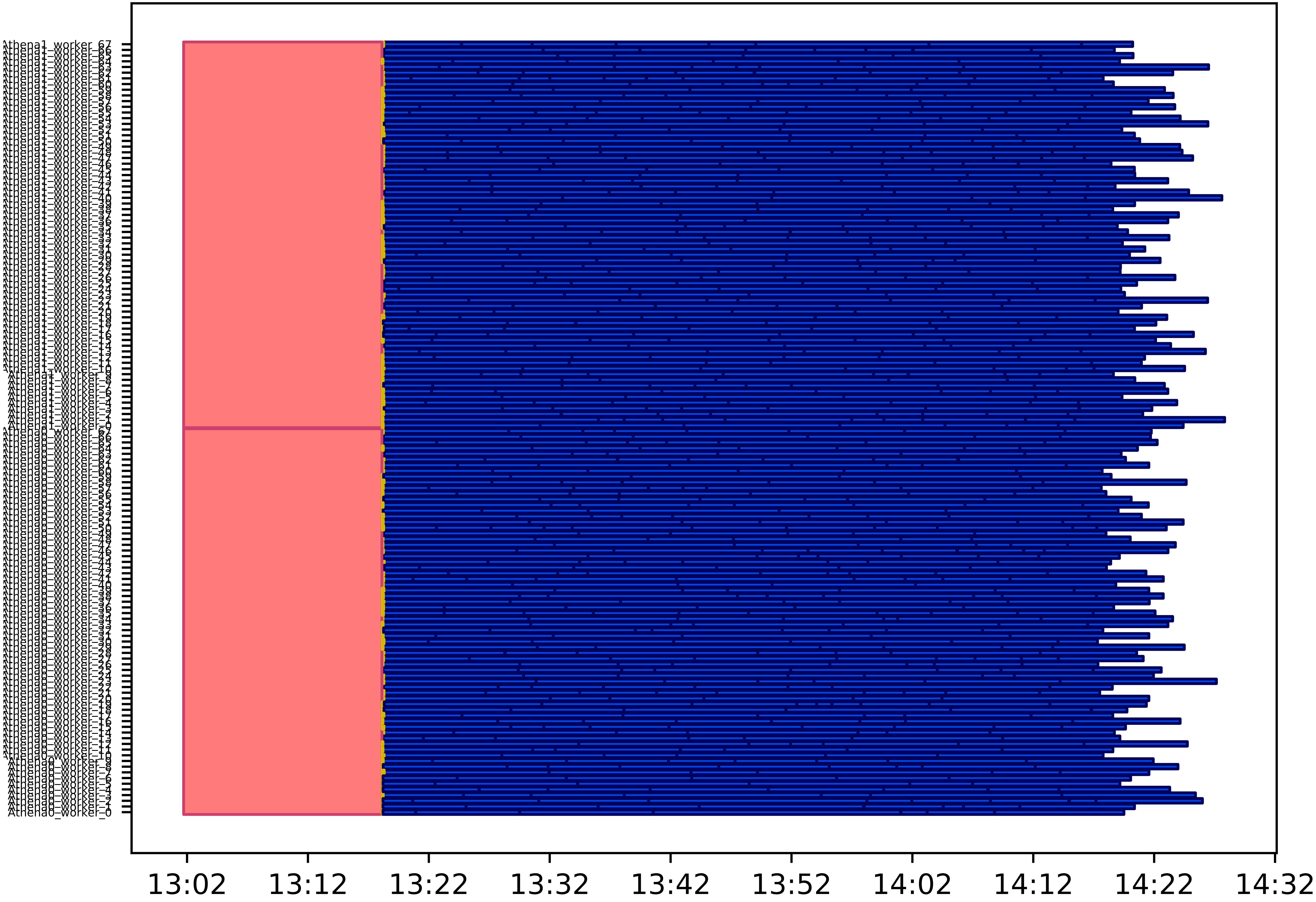
Close-up — two AthenaMP instances on Haswell nodes



Cori Haswell Nodes

- 32 cores / node
- 128 GB / node

Close-up — two AthenaMP instances on KNL nodes



🟥 AthenaMP Initialization

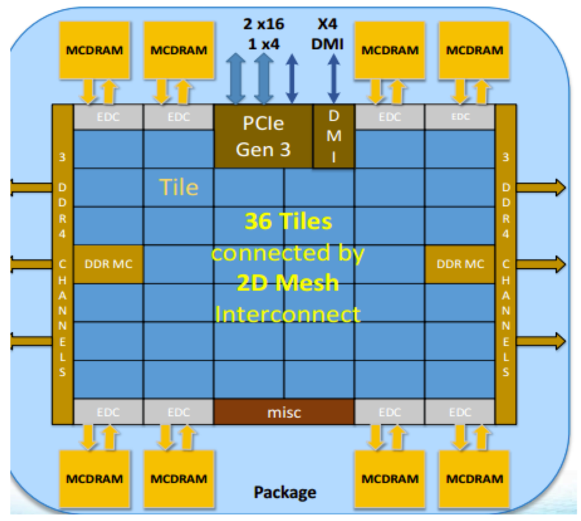
Worker

🟦 Event Processing

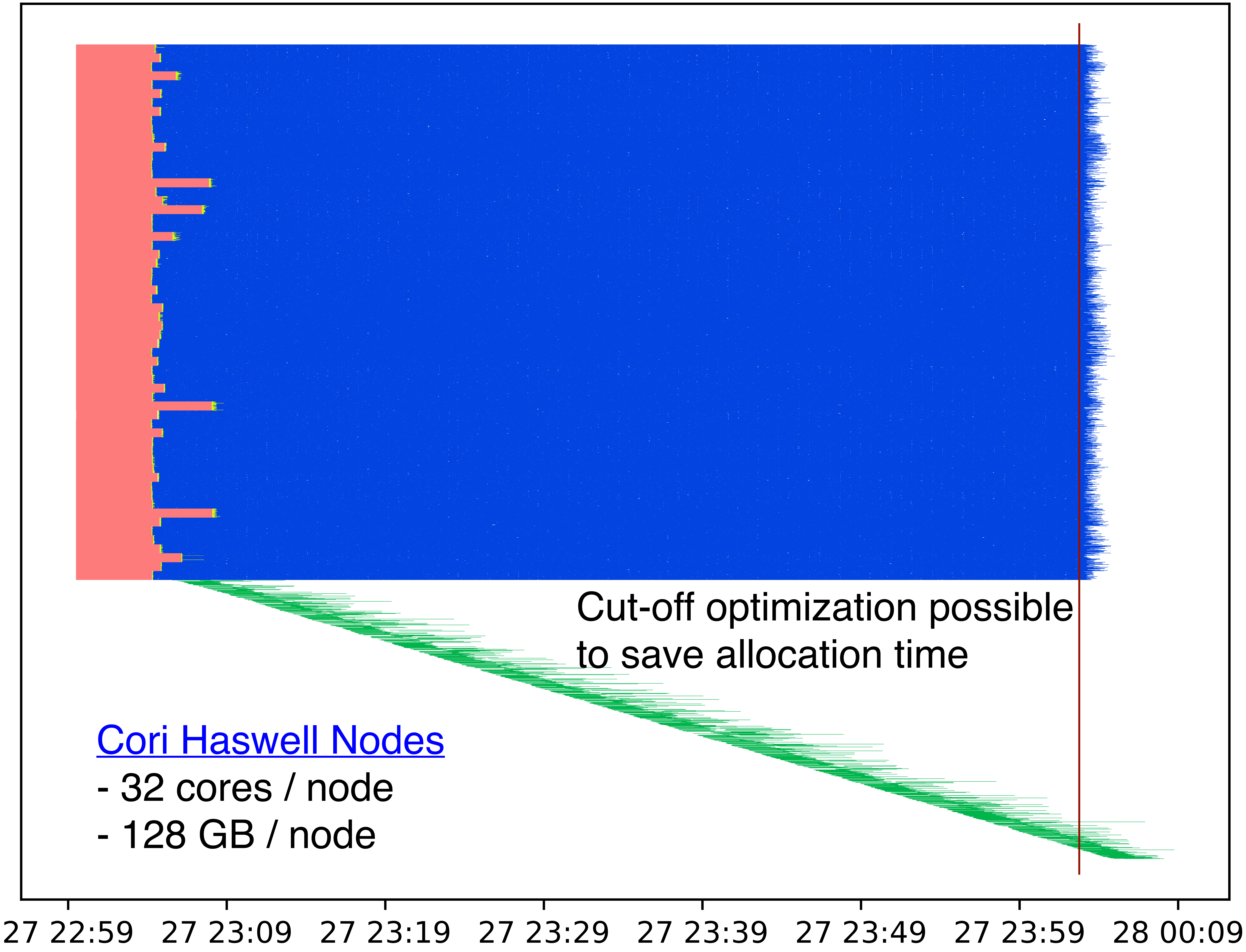
🟨 Worker Initialization

Cori KNL Nodes

- 68 cores / node



Large Haswell job example (60 AthenaMP instances, 100k events)



Cut-off optimization possible to save allocation time

Cori Haswell Nodes
- 32 cores / node
- 128 GB / node

🟥 AthenaMP Initialization

Worker

🟦 Event Processing

🟨 Worker Initialization

🟩 Merge Job

On-the-fly merging
1000 merge jobs,
100 events per job

Raythena running scheme

```
$ sbatch —image mmuskinj/centos7-atlasos-ray:1.0.0 —module=cvmfs
```

```
     $ shifter ./ray_start_head.sh

$ srun shifter ./ray_start_other.sh &

     $ shifter ./run_raythena.sh
```

- Ray, Raythena, and Athena are all running in a container on all nodes,
- At NERSC we are using Shifter containers which are built from Docker images,
- Can be ported to other HPCs without too much effort.

Future plans

- We present a proof-of-concept Ray application for the ATLAS Event Service,
- Our next steps include two main projects:
  - *Expand horizontally*: turn this into a ‘production quality’ application with full error handling and connection to the outside world (i.e. Panda, Harvester, Pilot v2, …),
  - *Vertical integration*: reproduce other layers of ATLAS workflow with Ray,
- Available person-power at LBL:
  - Paolo, Vakho, Charles, Illya, Miha, Julien Esseiva (MSc at HES-SO Switzerland, with us until Feb 2020), Rollin Thomas (NERSC).

Connecting to the outside world (*horizontal expansion*)

- Our goal for this workshop is to work together with the experts (Doug, Paul, Tadashi, et. al.) to figure out the details and benefits of using Ray for Run 3 job scheduling,
- We have a designated hands-on session tomorrow: https://indico.physics.lbl.gov/indico/event/955/#b-293-breakouts-ray-event-serv

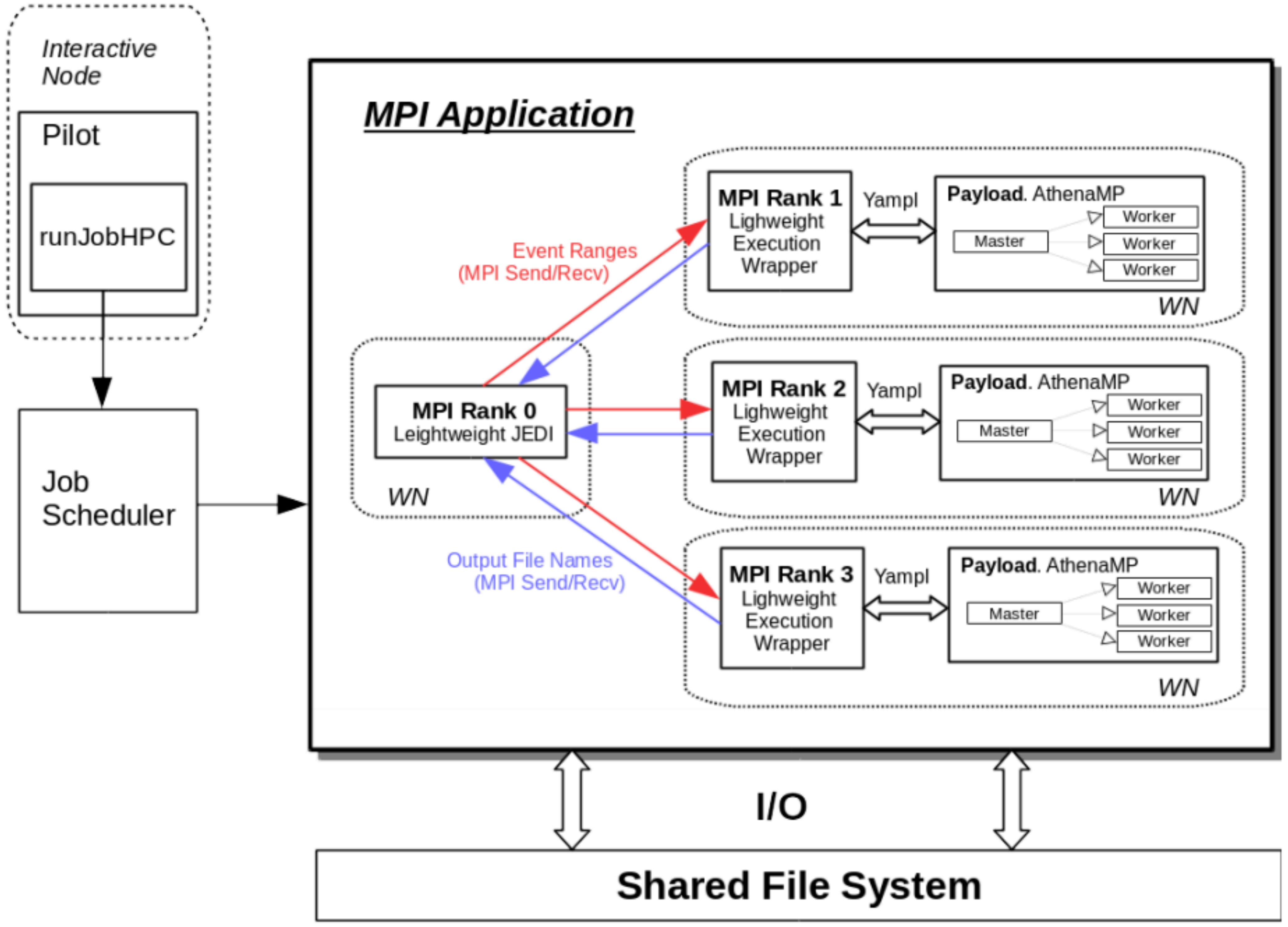
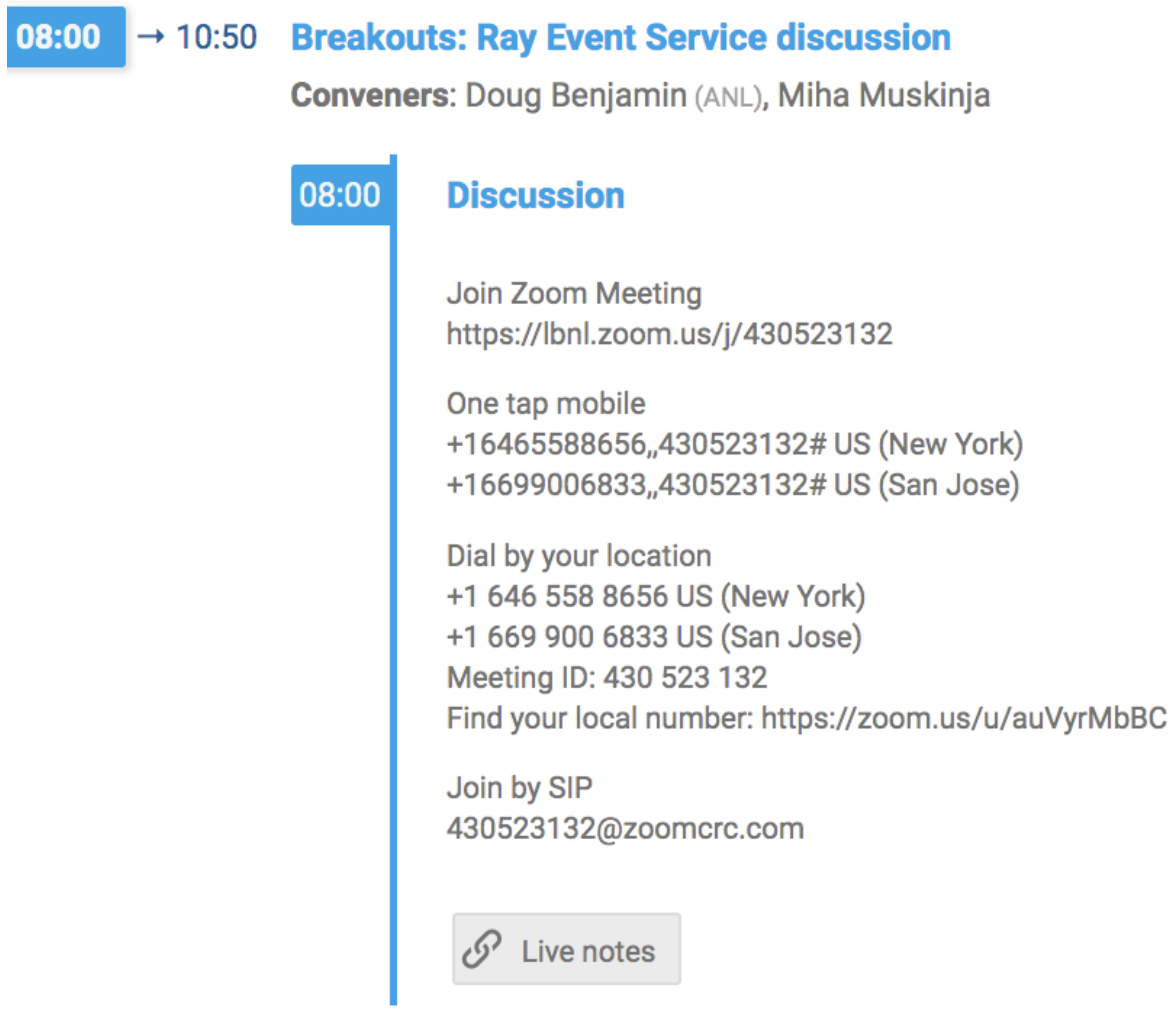


Figure 1. Schematic view of Yoda


08:00 → 10:50 **Breakouts: Ray Event Service discussion**

Conveners: Doug Benjamin (ANL), Miha Muskinja

08:00 **Discussion**

Join Zoom Meeting
https://lbnl.zoom.us/j/430523132

One tap mobile
+16465588656,,430523132# US (New York)
+16699006833,,430523132# US (San Jose)

Dial by your location
+1 646 558 8656 US (New York)
+1 669 900 6833 US (San Jose)
Meeting ID: 430 523 132
Find your local number: https://zoom.us/u/auVyrMbBC

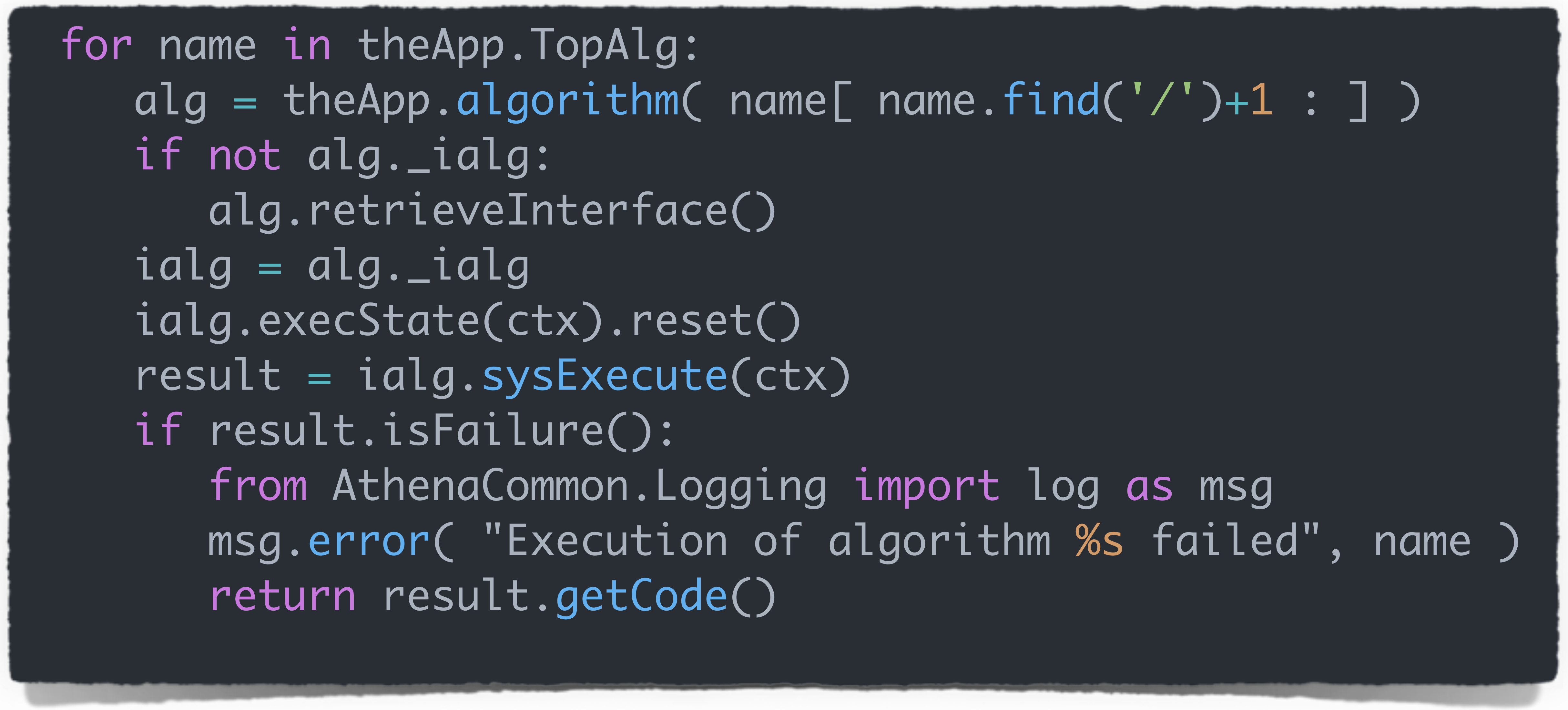
Join by SIP
430523132@zoomcrc.com

Live notes

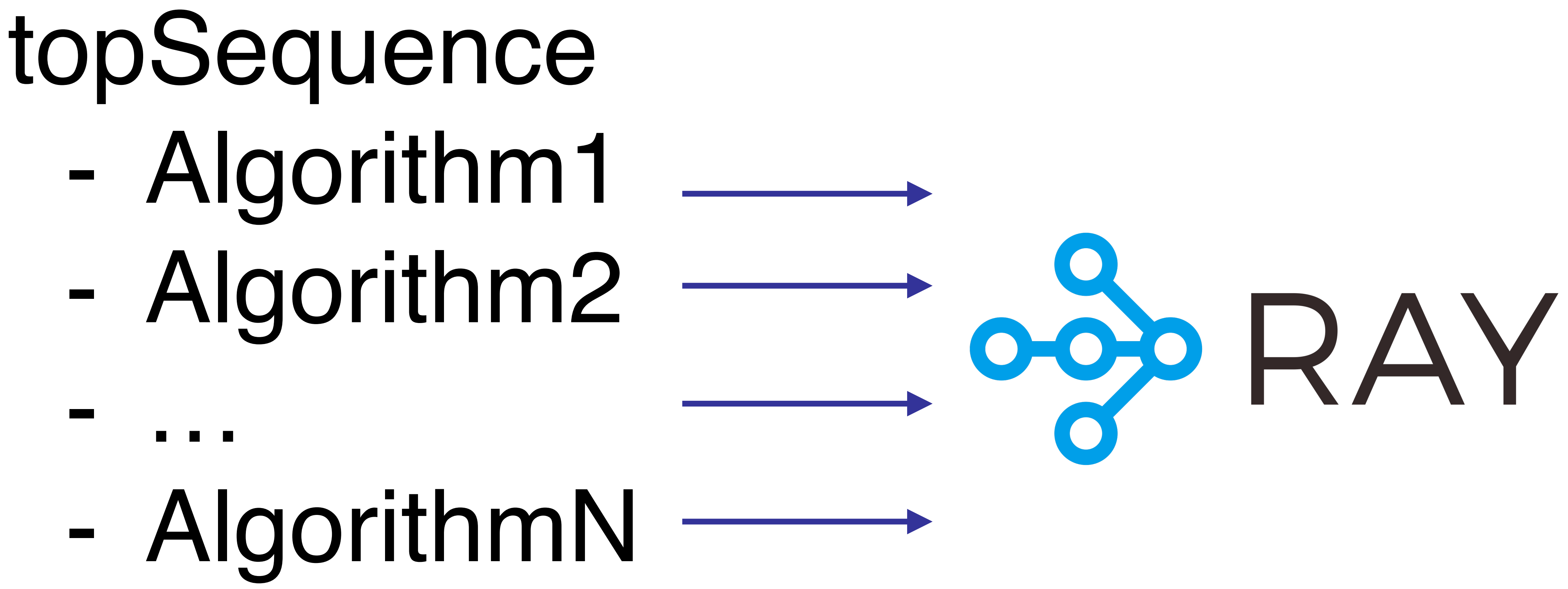
Digging deeper into Athena (*vertical expansion*)

- The long-term project is to interface Athena/Gaudi algorithms directly to Ray for a much finer control over scheduling the workload,
- This would replace the current event loop with Ray and enable scheduling of a single event across several nodes:
  - *Advantage*: maximize throughput by more efficient/tailored scheduling of algorithms to computing resources (e.g., CPU vs GPU),
- Our most promising idea is to use Athena's python EventLoopManager and rewrite it with Ray wrappers.

PyAthenaEventLoopMgr.py



```
for name in theApp.TopAlg:
   alg = theApp.algorithm( name[ name.find('/')+1 : ] )
   if not alg._ialg:
      alg.retrieveInterface()
   ialg = alg._ialg
   ialg.execState(ctx).reset()
   result = ialg.sysExecute(ctx)
   if result.isFailure():
      from AthenaCommon.Logging import log as msg
      msg.error( "Execution of algorithm %s failed", name )
      return result.getCode()
```



topSequence
- Algorithm1
- Algorithm2
- …
- AlgorithmN

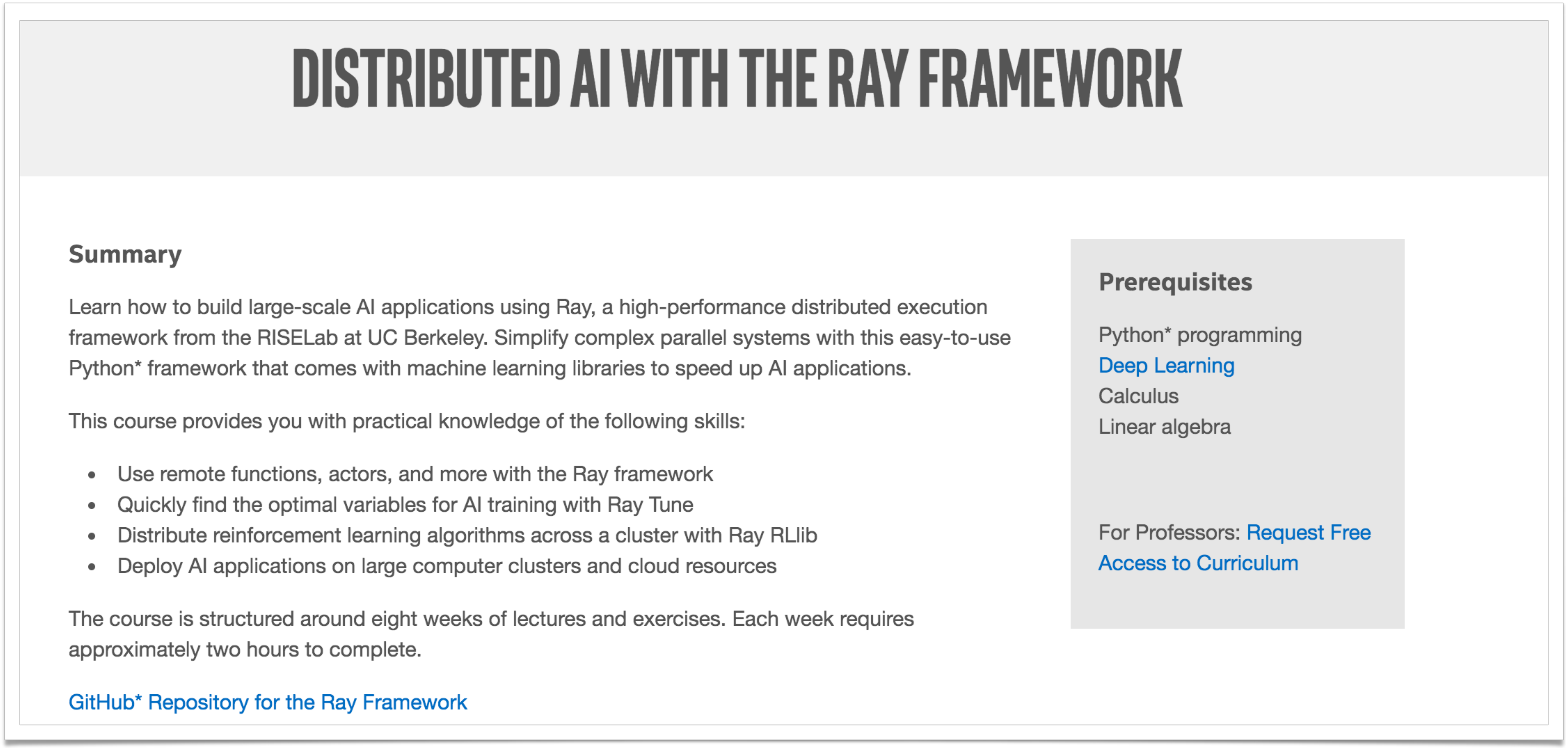
Conclusion

- We are exploring applicability of a distributed execution framework (Ray) to ATLAS workflows,
- So far we have a working prototype of a Ray-based ATLAS Event Service,
  - We are working with the experts on integrating this system in the ATLAS' global job scheduling system via Panda,
  - A stand-alone version successfully tested on Cori Haswell and KNL nodes,
  - Can be fully run from containers and is therefore portable to other systems,
- Longer-term-plan is to divide the ATLAS workflow into base components (Algorithms) and interface them directly to Ray.

BACKUP

Ray documentation and tutorials

- Ray has a very rich documentation hosted on readthedocs:
  - https://ray.readthedocs.io/en/latest/index.html,
- Hands-on tutorials with exercises available in form of jupyter notebooks,
- Since Feb 2019, Intel hosts an 8-week course about distributed AI computation with Ray: https://software.intel.com/en-us/ai/courses/distributed-AI-ray.



DISTRIBUTED AI WITH THE RAY FRAMEWORK

Summary

Learn how to build large-scale AI applications using Ray, a high-performance distributed execution framework from the RISELab at UC Berkeley. Simplify complex parallel systems with this easy-to-use Python* framework that comes with machine learning libraries to speed up AI applications.

This course provides you with practical knowledge of the following skills:

- Use remote functions, actors, and more with the Ray framework
- Quickly find the optimal variables for AI training with Ray Tune
- Distribute reinforcement learning algorithms across a cluster with Ray RLlib
- Deploy AI applications on large computer clusters and cloud resources

The course is structured around eight weeks of lectures and exercises. Each week requires approximately two hours to complete.

GitHub* Repository for the Ray Framework

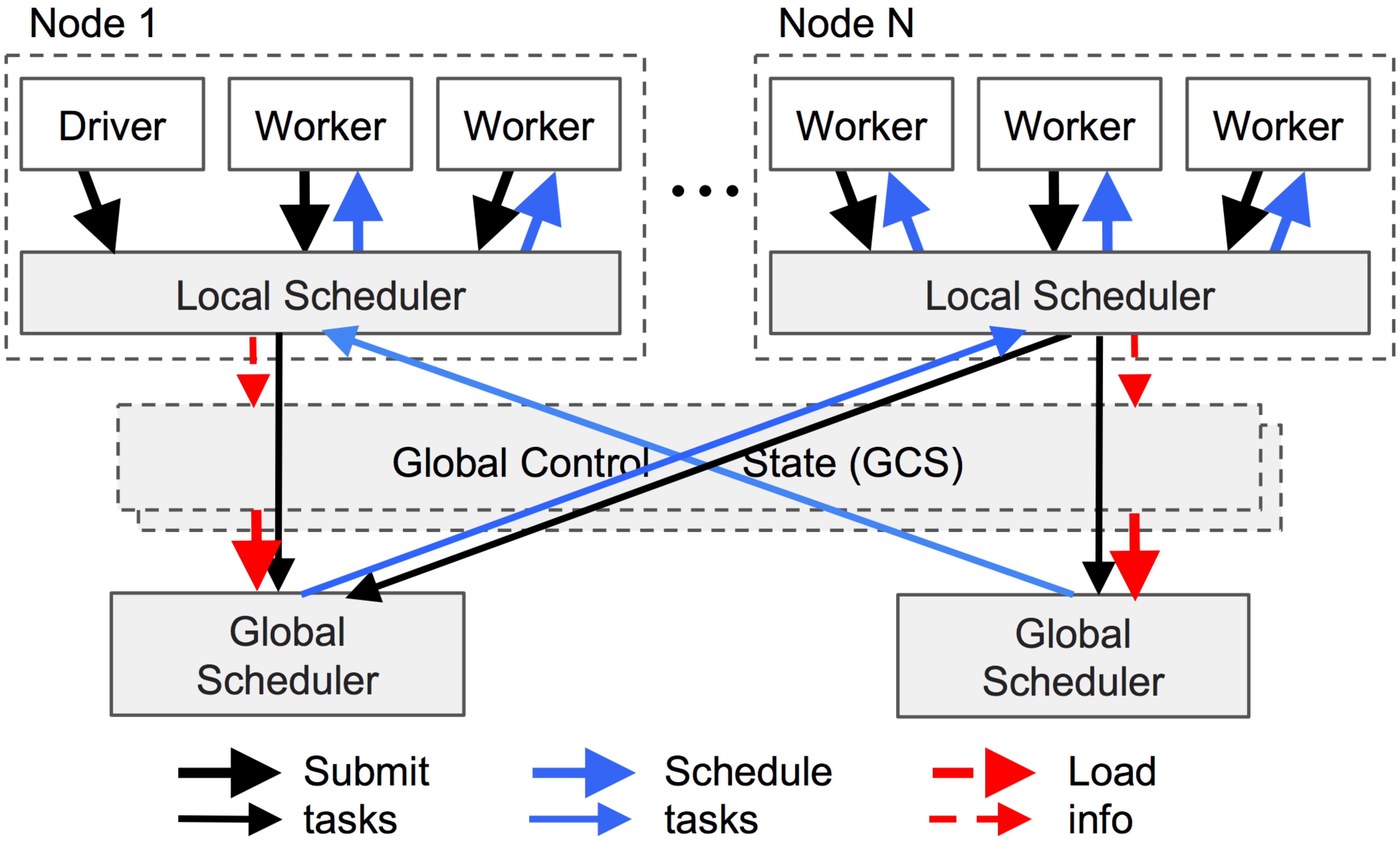
Prerequisites

Python* programming
Deep Learning
Calculus
Linear algebra

For Professors: Request Free Access to Curriculum

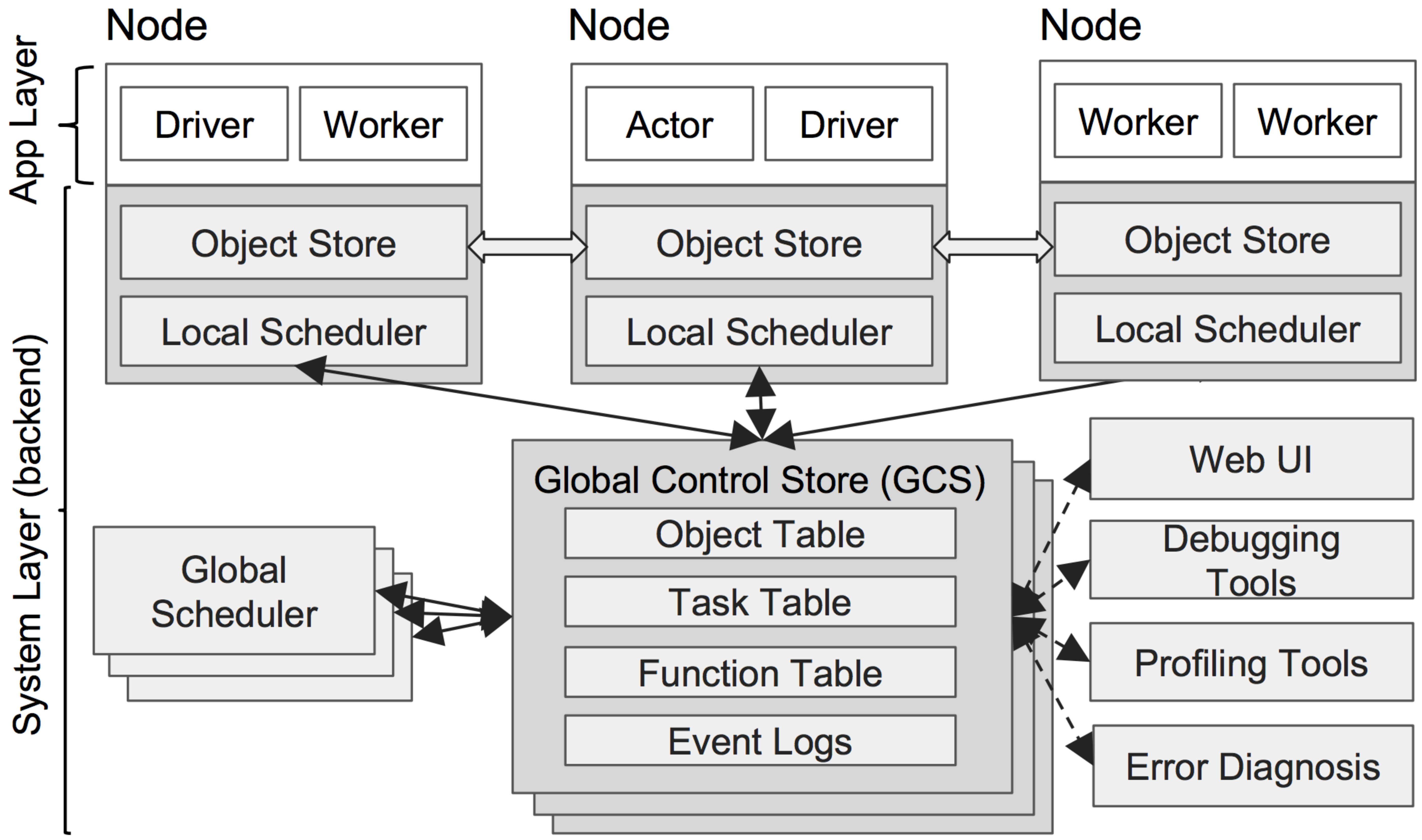
Ray 101

- One driver application (running on any compute node) controls all nodes in a cluster (HPC) that are connected via TCP to a redis server,
- Tasks are first scheduled locally (Local Scheduler) if resources are available, otherwise they are scheduled globally via the Global Scheduler.



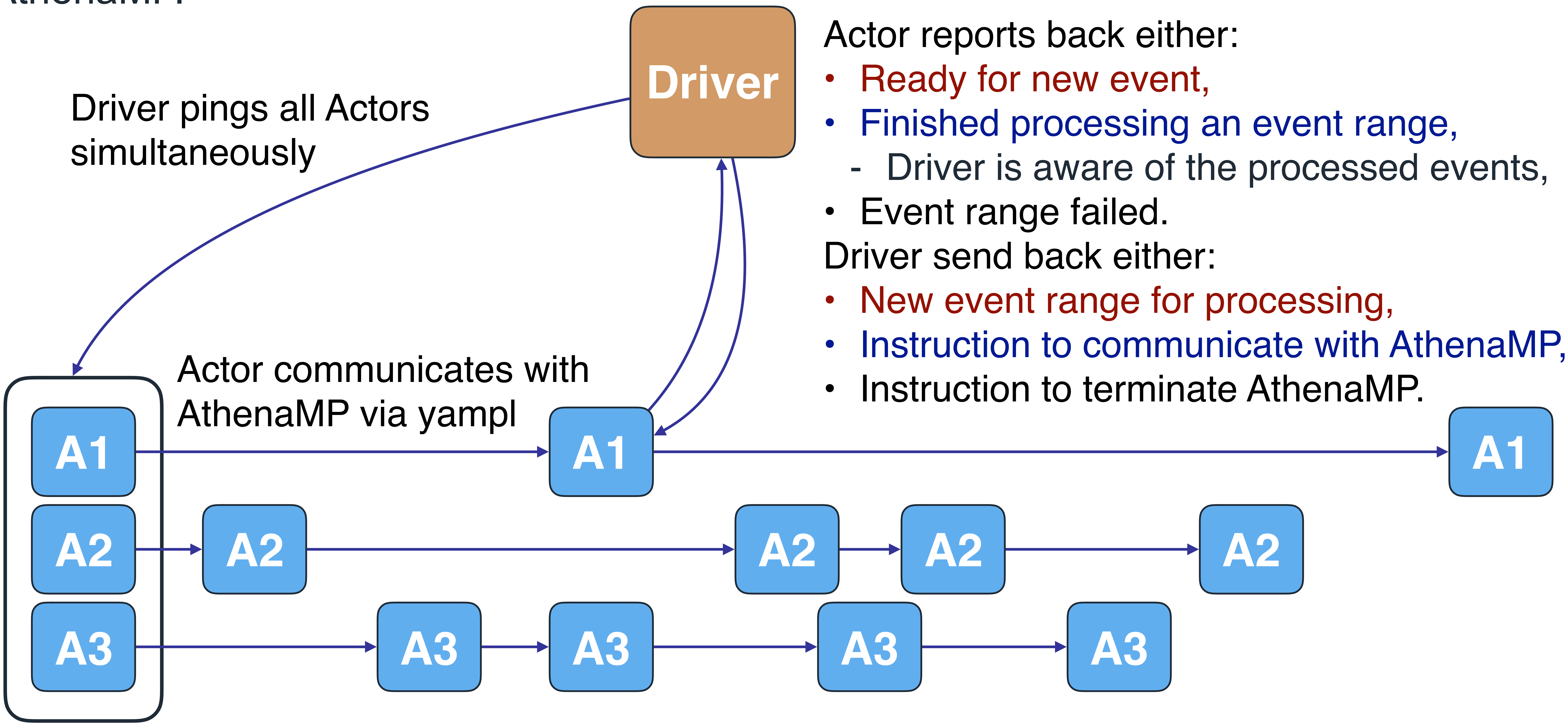
Ray Application Layer

- Ray maintains three types of processes:
  - *Driver*: a process executing the user program,
  - *Worker*: a stateless process that executes tasks invoked by the driver or another worker. Workers are started automatically and execute tasks serially without maintaining a local state,
  - *Actor*: a stateful process that executes only the method it exposes. They execute methods serially and each method depends on the state resulting from the previous execution.



Asynchronous communication between the Driver and Actors

- Asynchronous communication is implemented in a few 100 python lines using Ray explicit parallelism expressions,
- There is no redundant communication; Actors independently communicate with AthenaMP.


Driver
Driver pings all Actors simultaneously
Actor communicates with AthenaMP via yampl
A1
A2
A3
A1
A2
A3
A3
A2
A3
A2
A3
A2
A1

Actor reports back either:

- Ready for new event,
- Finished processing an event range,
  - Driver is aware of the processed events,
- Event range failed.

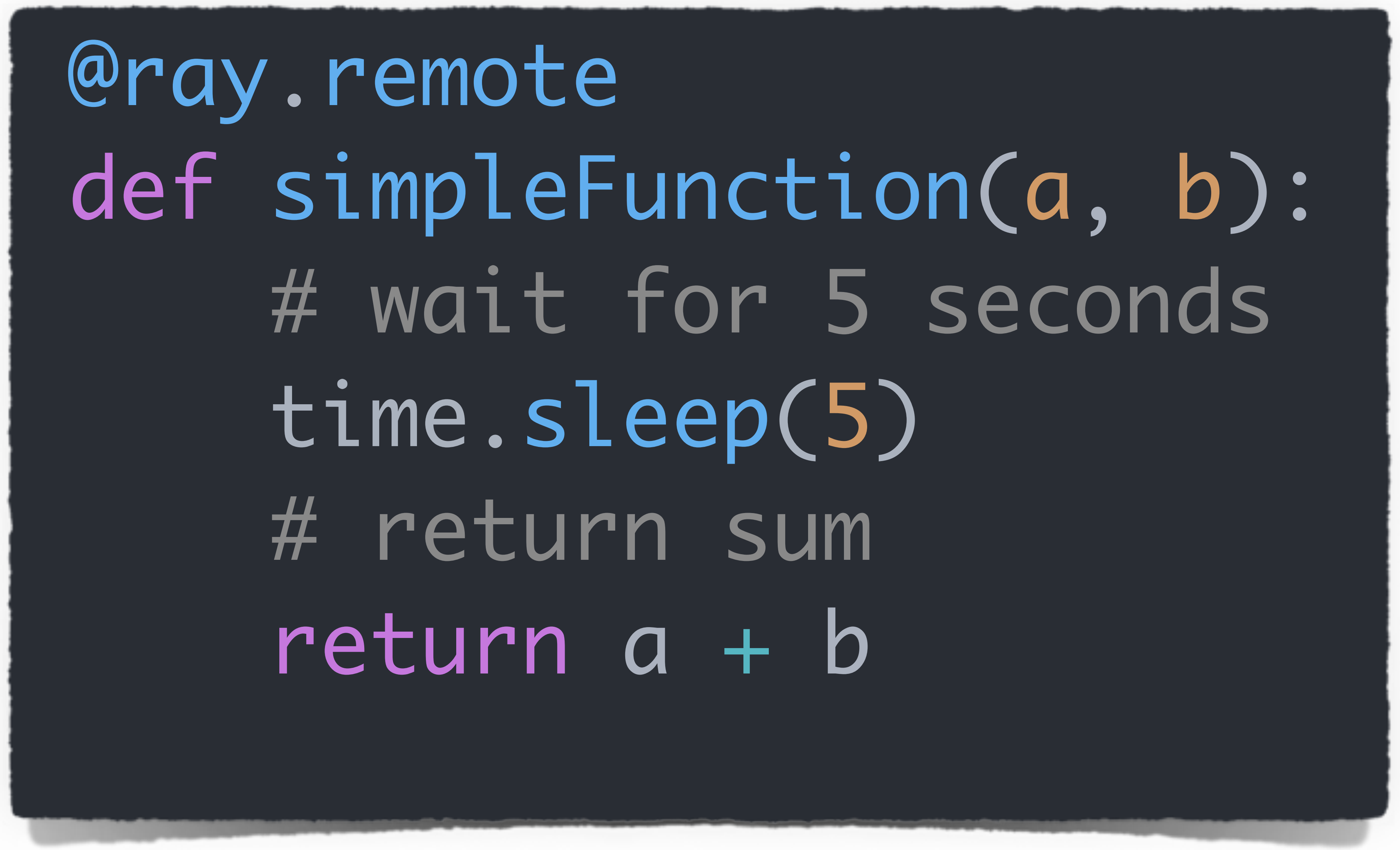
Driver send back either:

- New event range for processing,
- Instruction to communicate with AthenaMP,
- Instruction to terminate AthenaMP.

Ray Functions and Actors

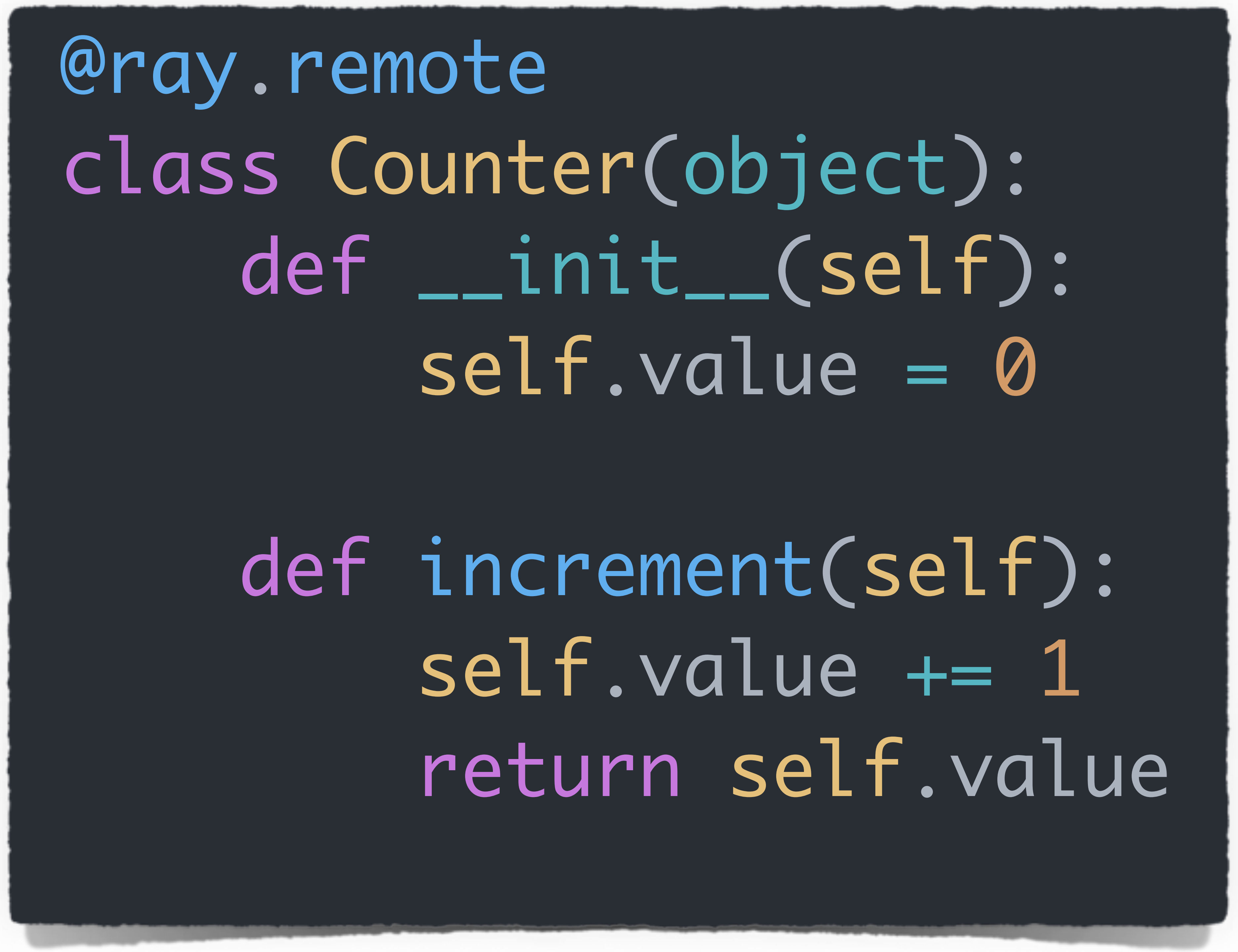
- A Ray parallel application is constructed with python decorations:

Task executed at a worker



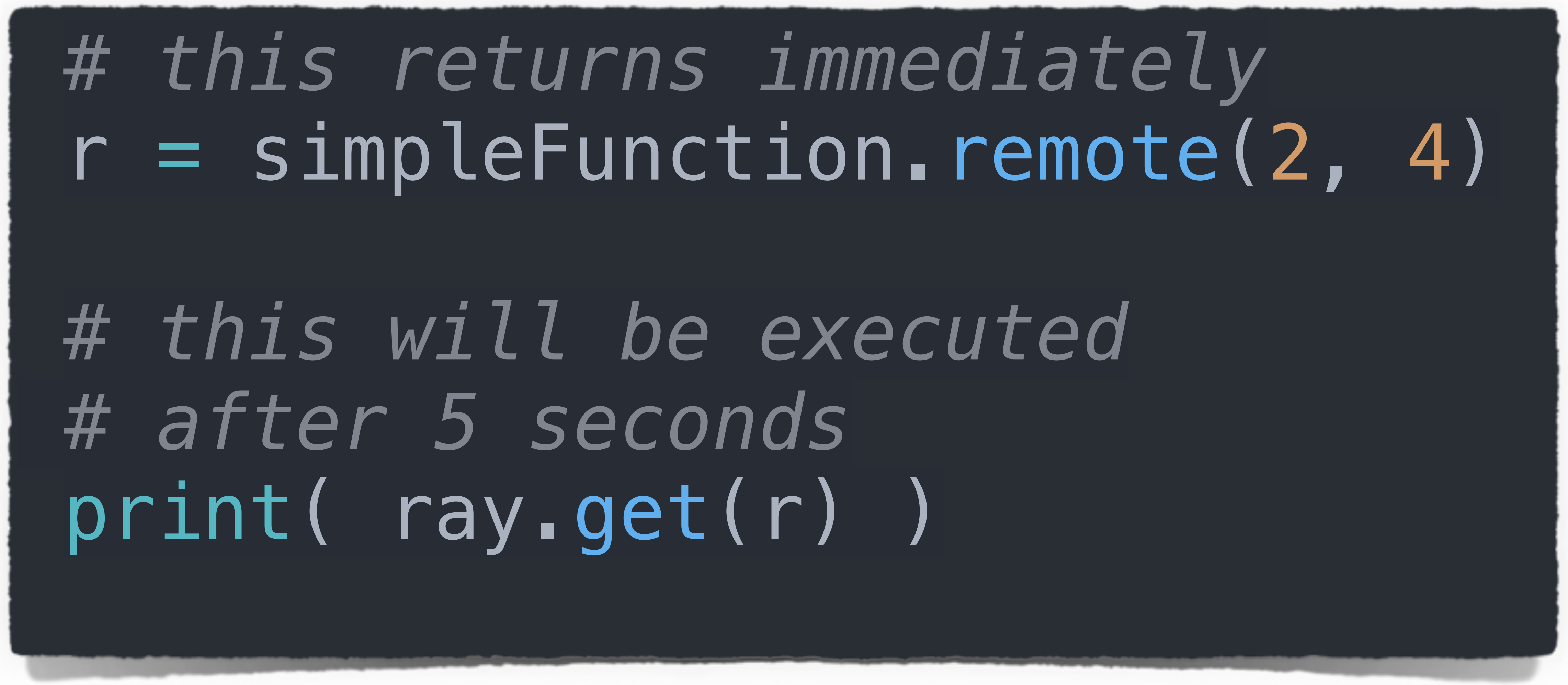
```python
@ray.remote
def simpleFunction(a, b):
    # wait for 5 seconds
    time.sleep(5)
    # return sum
    return a + b
```

Actor process



```python
@ray.remote
class Counter(object):
    def __init__(self):
        self.value = 0

    def increment(self):
        self.value += 1
        return self.value
```



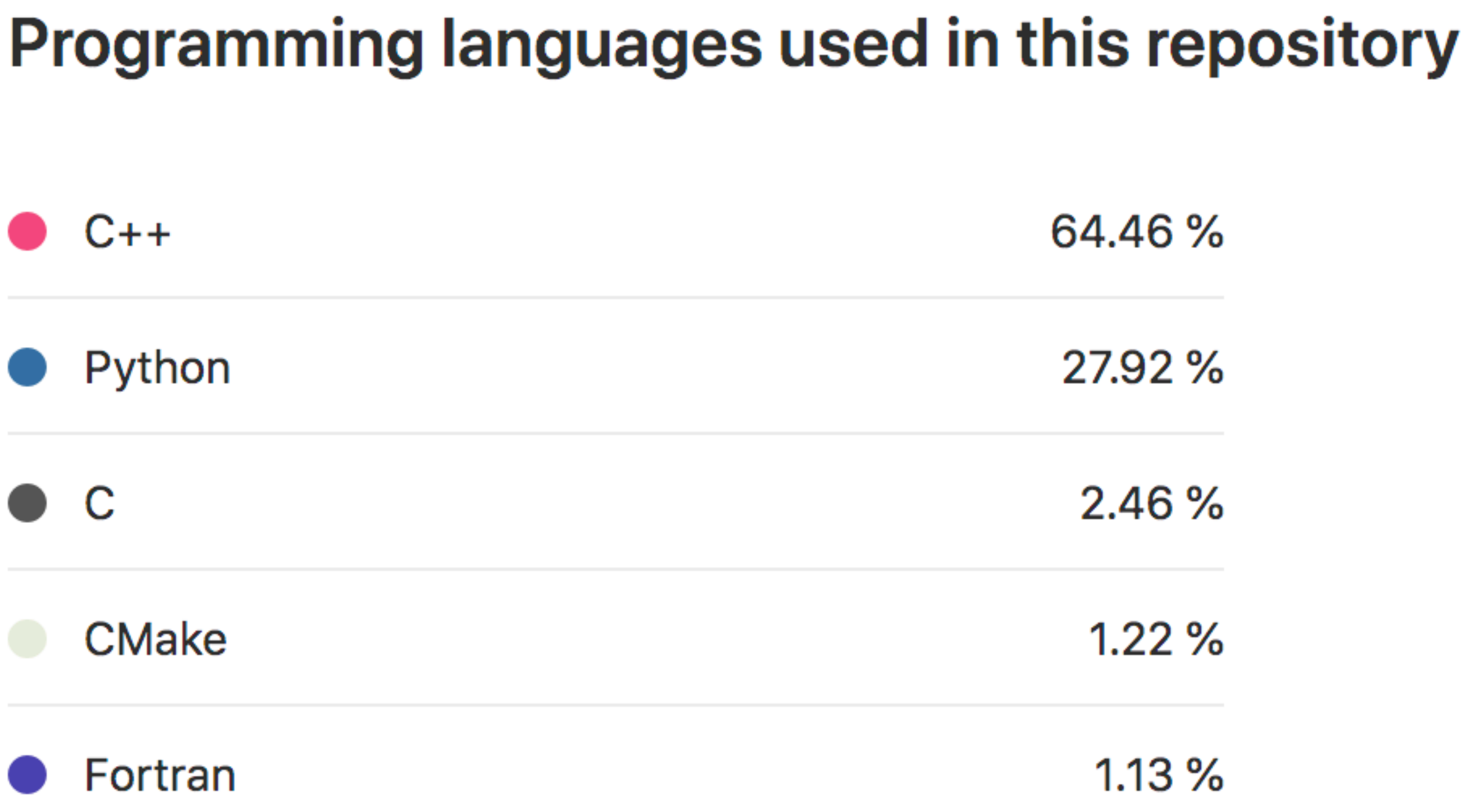
```python
# this returns immediately
r = simpleFunction.remote(2, 4)

# this will be executed
# after 5 seconds
print( ray.get(r) )
```

Driver application

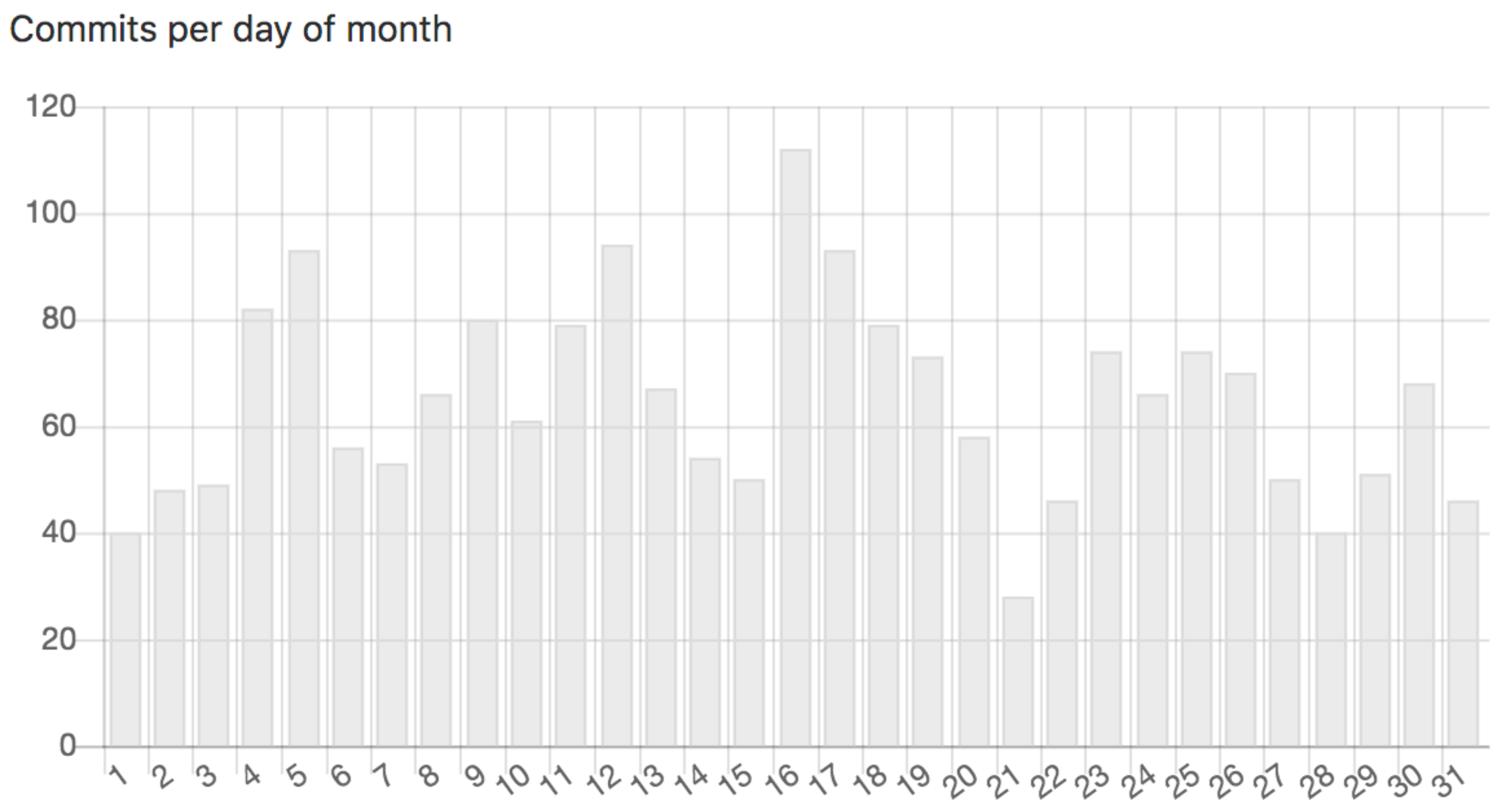
Athena framework in ATLAS

- Athena is the main software framework in ATLAS used for all data analysis steps: event generation, simulation, digitization, reconstruction, user analysis,
- Athena is based on the common Gaudi framework that is used by ATLAS, LHCb and FCC,
- Software that we are designing and covering in this talk is tailored to efficiently run Athena applications on HPCs.



Programming languages used in this repository

●	C++	64.46 %
●	Python	27.92 %
●	C	2.46 %
●	CMake	1.22 %
●	Fortran	1.13 %


Commits per day of month