ESCHER – Expressive Scheduling with Ephemeral Resources

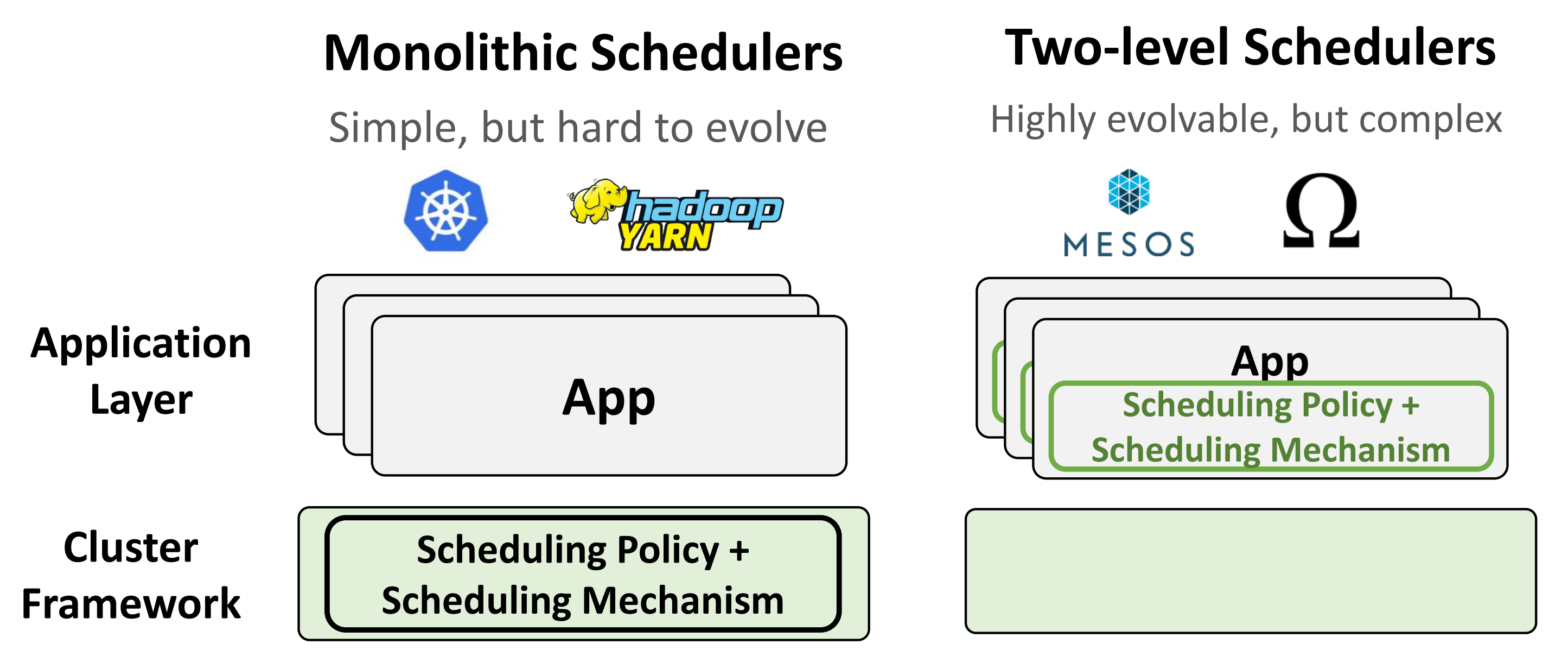
Romil Bhardwaj, Alexey Tumanov, Richard Liaw, Stephanie Wang, Philipp Moritz, Robert Nishihara, Ion Stoica

romilb@cs.berkeley.edu

Berkeley UNIVERSITY OF CALIFORNIA · Georgia Tech · anyscale

Cluster schedulers need to be evolvable

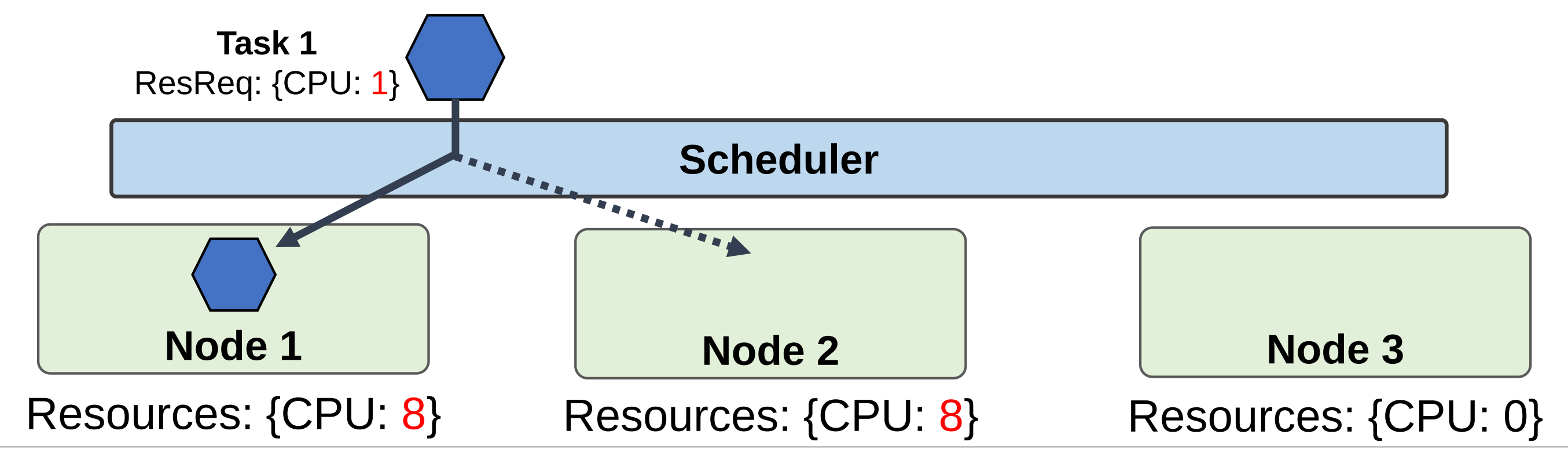
- Scheduling requirements of modern distributed applications are getting increasingly **complex**
  - E.g., Distributed training requires affinity, anti-affinity and gang scheduling – all in the same job
- Cluster frameworks must provide **flexible scheduling control without the complexities of implementing a scheduler.** Existing Schedulers are insufficient:


Monolithic Schedulers
Simple, but hard to evolve
Two-level Schedulers
Highly evolvable, but complex
MESOS
Application Layer
App
App
Scheduling Policy + Scheduling Mechanism
Cluster Framework
Scheduling Policy + Scheduling Mechanism

ESCHER Abstractions

Abstraction 1: Resource matching scheduler

Scheduler matches tasks resource **requirements** to node resource **availabilities**



Abstraction 2: Create resources at runtime

Frameworks provide an **API** for **applications** to **create resources** on nodes at **runtime**

```
def set_resource(label, capacity, node_spec=None)
```

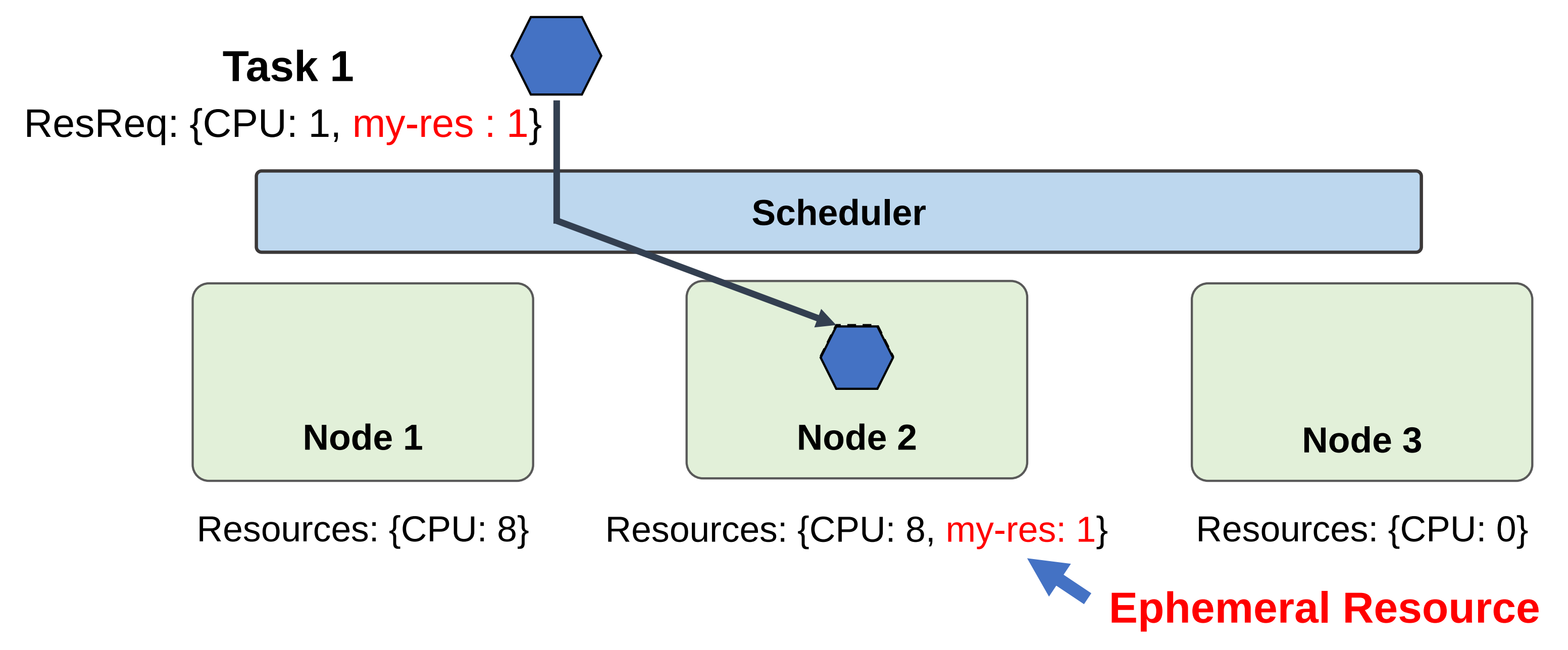
Grants applications control over **resource management**

Can specify **resource availability constraints** for resource creation

Combining the Abstractions

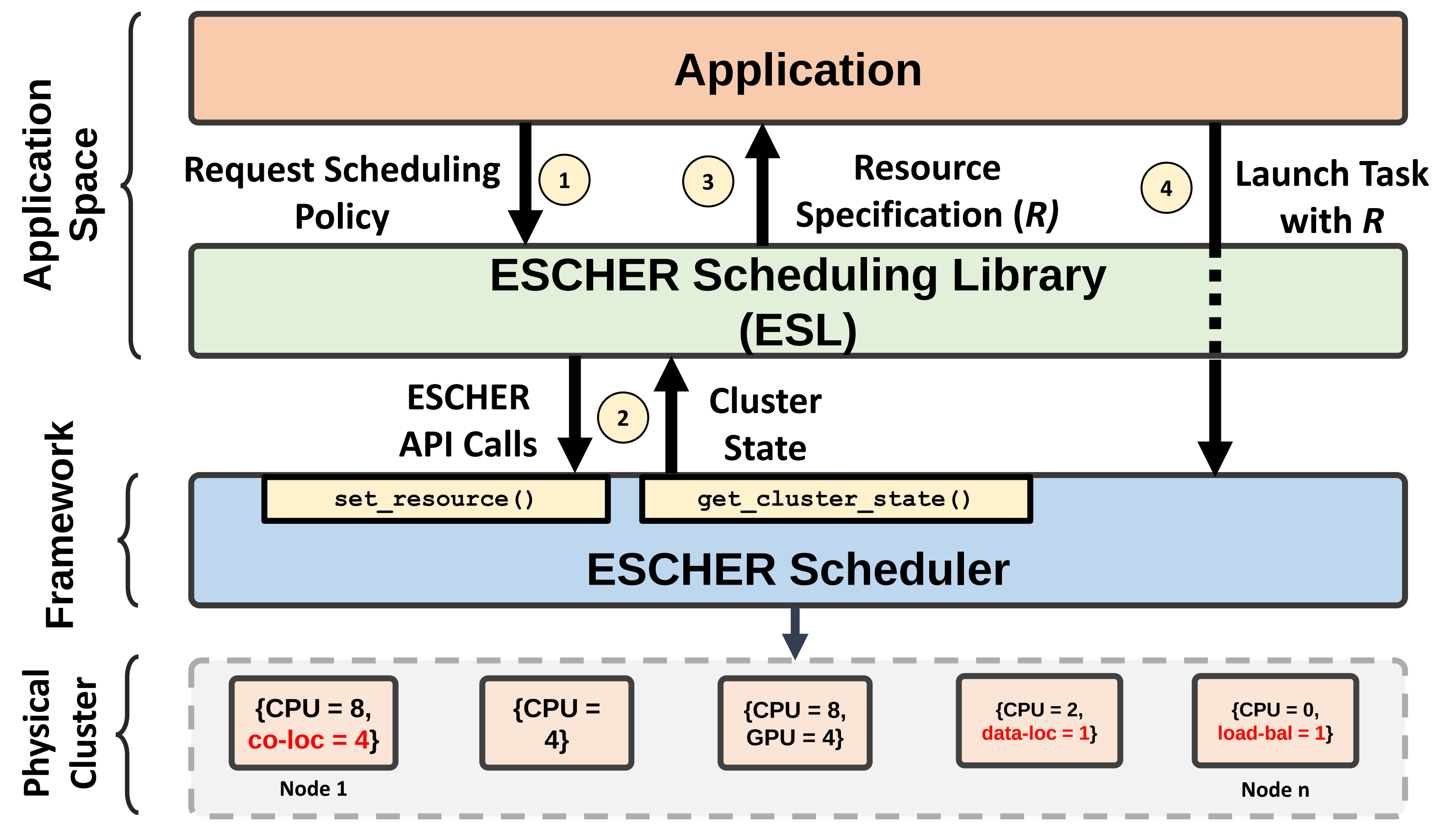
A simple resource matching scheduler can be **induced** to make targeted placement decisions with short-lived **ephemeral** resources

- These abstractions are **sufficient** to allow applications to express any arbitrary scheduling policy
- Applications can use resource management to **declaratively** specify and execute scheduling constraints

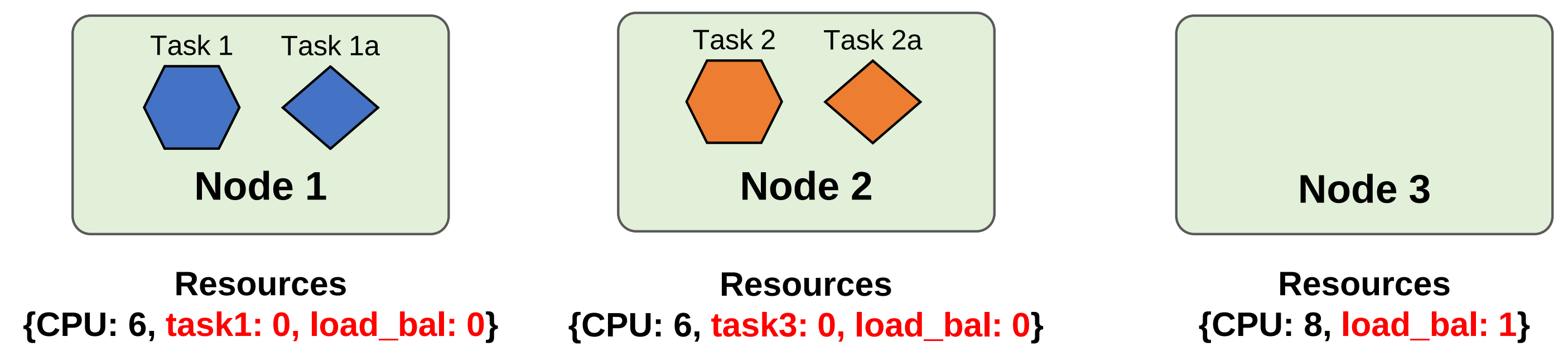


ESCHER Workflow

- ESCHER Scheduling Libraries (ESLs) encapsulate complexity of using ephemeral resources into reusable libraries.



Example - Load Balancing and Co-location



```
def task(task_id):
    set_resource(label=task_id, capacity=1)
    ...

def colocated_task():
    ...

# Create load-balancing resources
set_resource(“load_bal”, 1, where={cpu: 8})

# Create 3 pairs of tasks to co-locate and load balance
for i in range(0, task_count):
    launch(task, id=i, resources={“load_bal”: 1})
    # Launch co-located task
    launch(colocated_task, resources={i: 1})
```

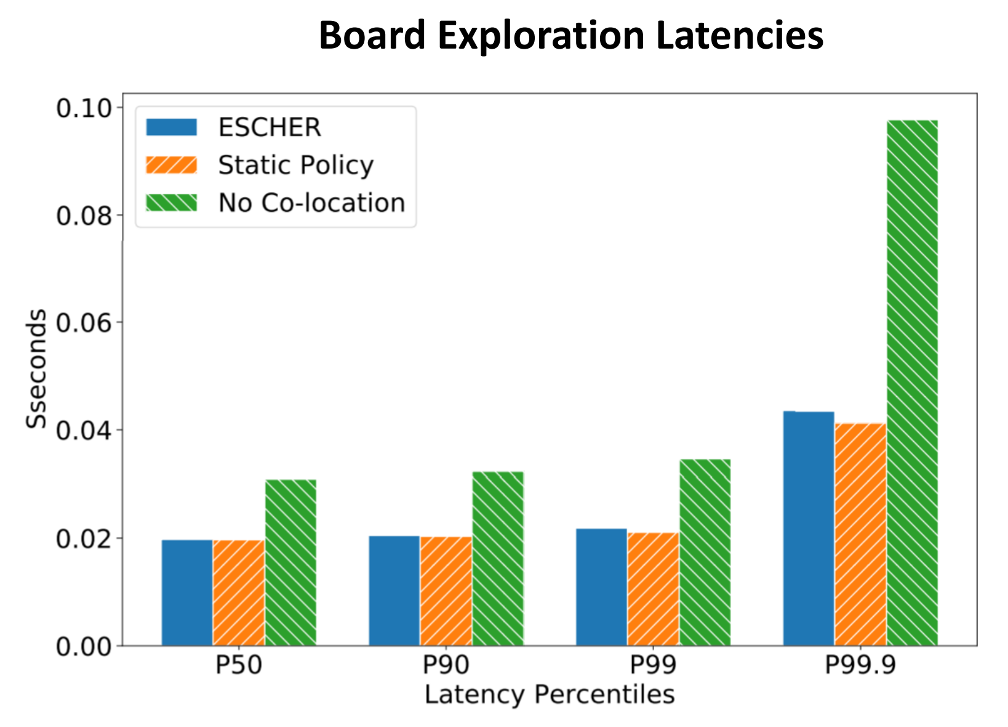
ESCHER has implementations of **data locality**, **bin-packing**, **anti-affinity**, **soft constraints**, **gang scheduling, WFQ** and compositions

Benchmarks

AlphaZero on ESCHER

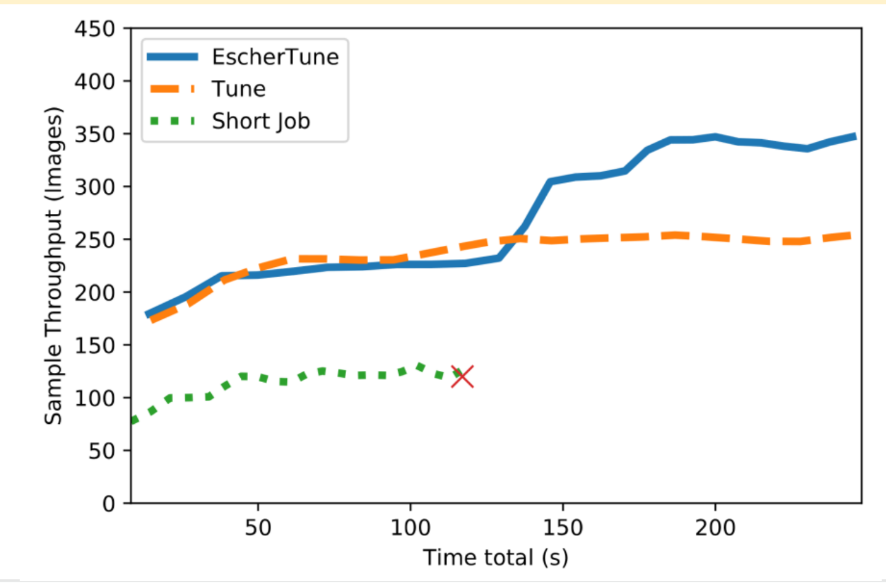
ESCHER is **2x faster** in exploring Go board states than an unaware scheduler

Performs comparably **with a hard-coded policy**, while requiring only **5 lines** of changes

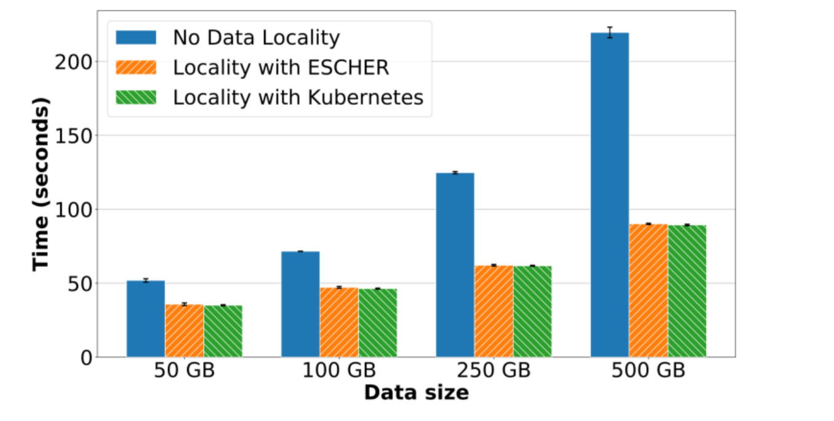


Distributed Training on ESCHER

Ported Gandiva's[1] scheduling policies to Ray Tune - 38% speedup with just 40 lines of code.



Kubernetes MapReduce on ESCHER



Number of Nodes	Scheduler: Generic	Kubernetes	ESCHER
10	183.32 ± 0.51	54.69 ± 0.46	55.24 ± 0.39
50	113.71 ± 0.49	44.02 ± 0.27	44.71 ± 0.44
100	51.90 ± 0.31	35.08 ± 0.31	35.76 ± 0.49

Job Makespan

[1] OSDI 18