RAY

Distributed Systems and AI

Ray: Programming at Any Scale

https://github.com/ray-project/ray

Robert Nishihara

riselab
UC Berkeley

A Growing Number of Use Cases

蚂蚁金服
ANT FINANCIAL

Berkeley
UNIVERSITY OF CALIFORNIA

BONSAI AI

ERICSSON

intel

Alibaba.com™

facebook

J.P.Morgan

Morgan Stanley

PRIMER

Microsoft

The Machine Learning Ecosystem

Machine Learning Ecosystem

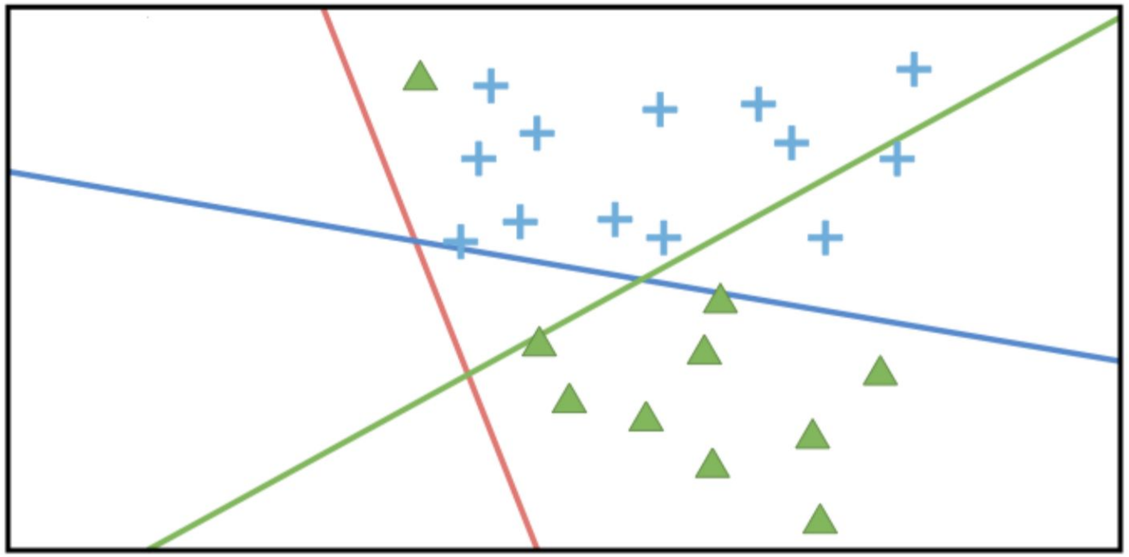
The Machine Learning Ecosystem

The Machine Learning Ecosystem

Training

Multiclass (more than 2) Classification

- **One-vs-rest** train separate binary classifiers for each class

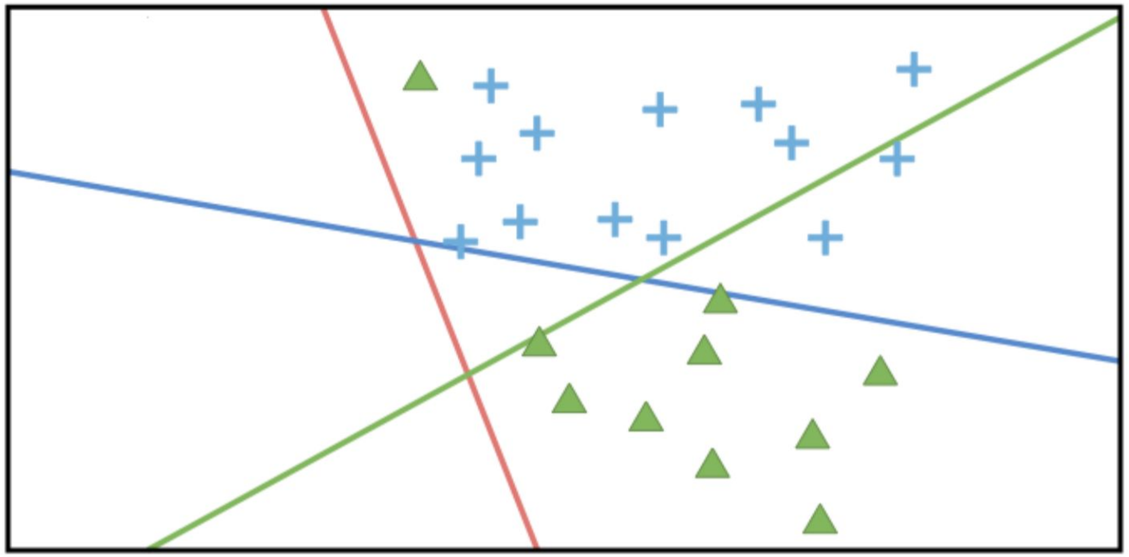


Machine Learning Ecosystem

The Machine Learning Ecosystem

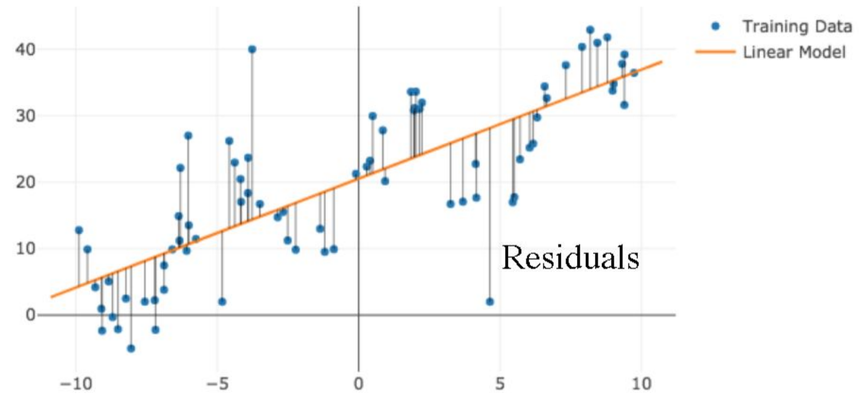
Multiclass (more than 2) Classification

- **One-vs-rest** train separate binary classifiers for each class



Training

Diagnosing Fit
→ The Residuals



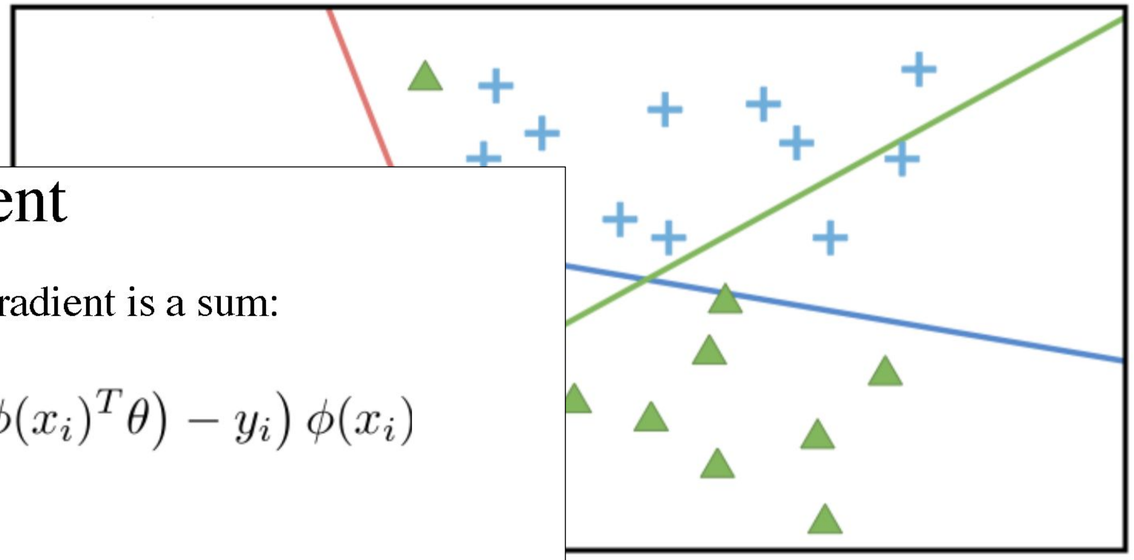
achine Learning Ecosystem

The Machine Learning Ecosystem

Training

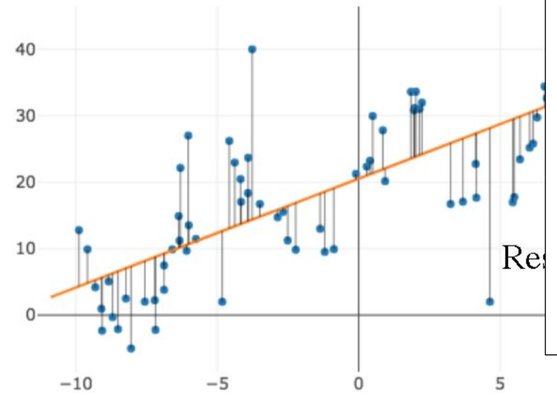
Multiclass (more than 2) Classification

- **One-vs-rest** train separate binary classifiers for each class



Diagnosing Fit

→ The Residu



Stochastic Gradient Descent

- For many learning problems the gradient is a sum:

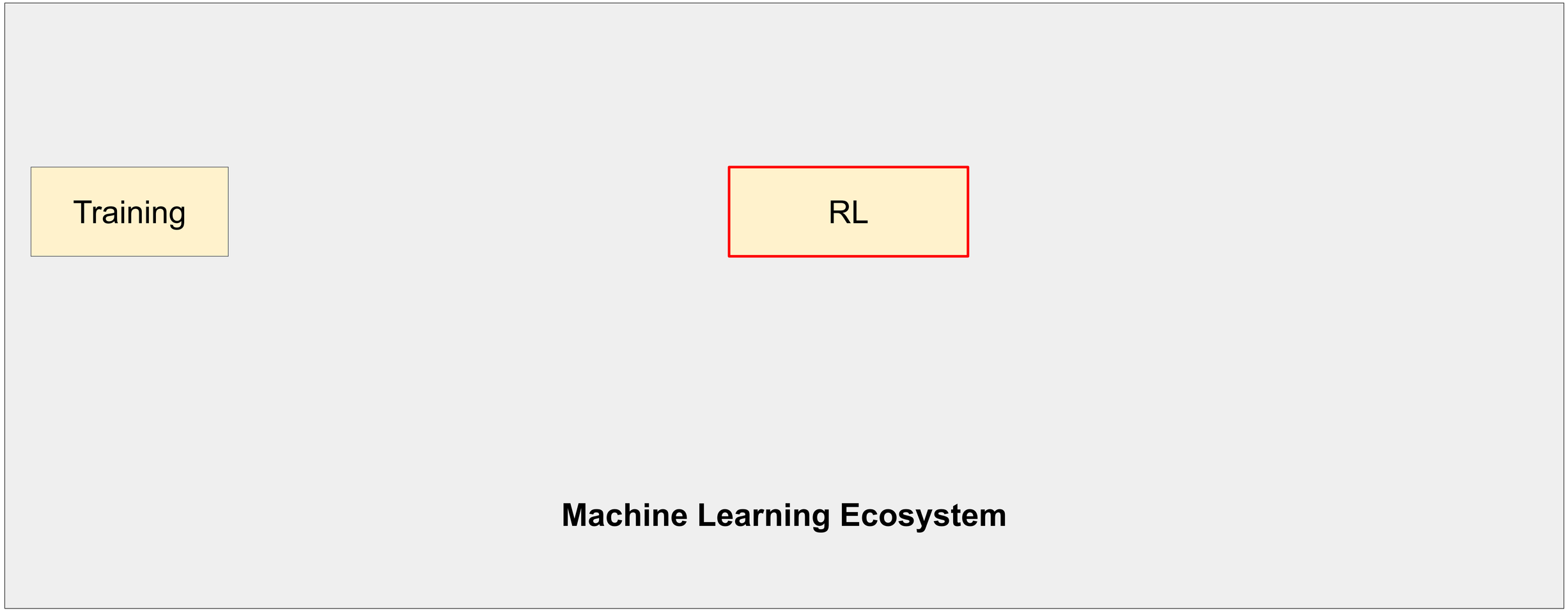
$$\nabla_\theta \mathbf{L}(\theta) = \frac{1}{n}\sum_{i=1}^{n}\left(\sigma\left(\phi(x_i)^T\theta\right) - y_i\right)\phi(x_i)$$

  - For large n this can be costly

- What if we approximated the gradient by looking at a few random points:

$$\nabla_\theta \mathbf{L}(\theta) \approx \frac{1}{|\mathcal{B}|}\sum_{i\in\mathcal{B}}\left(\sigma\left(\phi(x_i)^T\theta\right) - y_i\right)\phi(x_i)$$

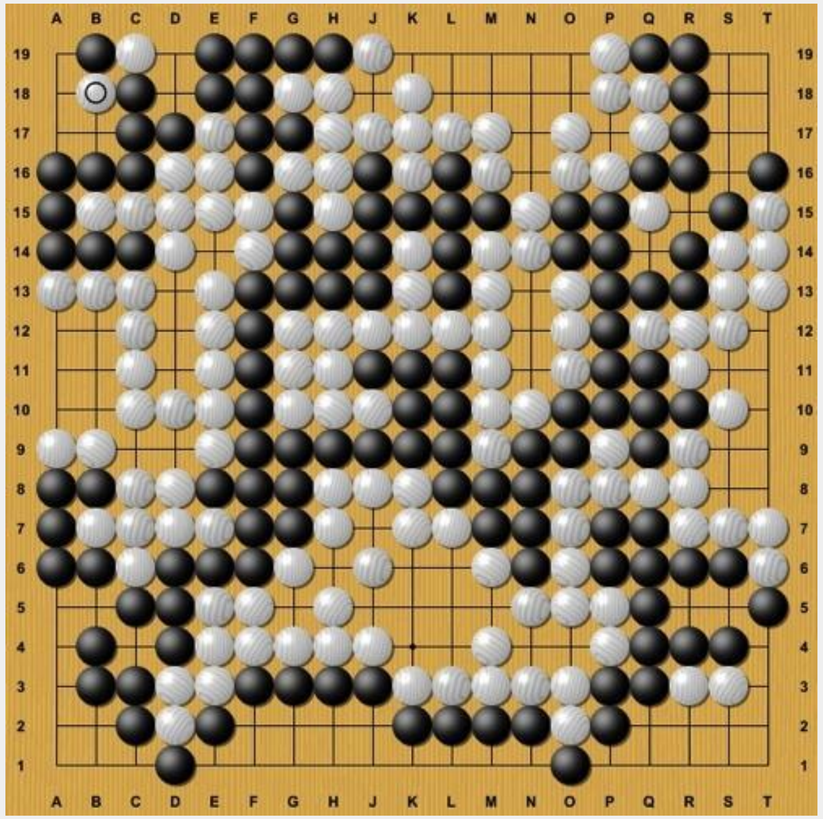
The Machine Learning Ecosystem



The Machine Learning Ecosystem

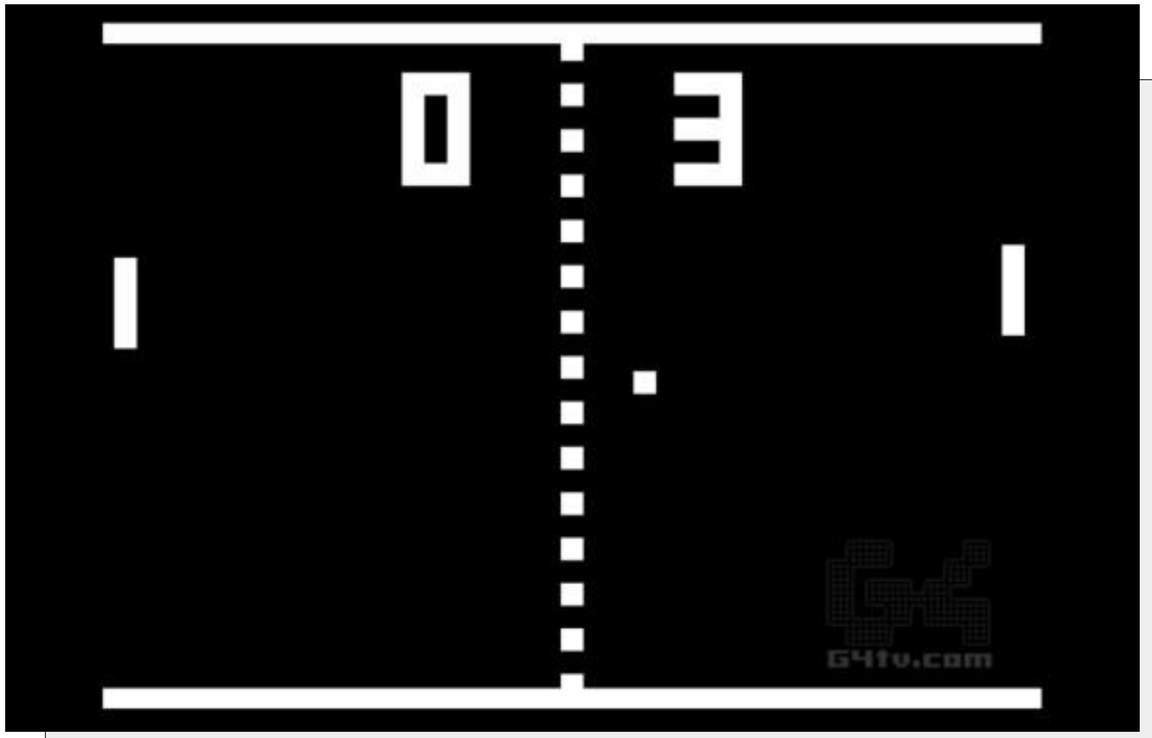
Training

RL

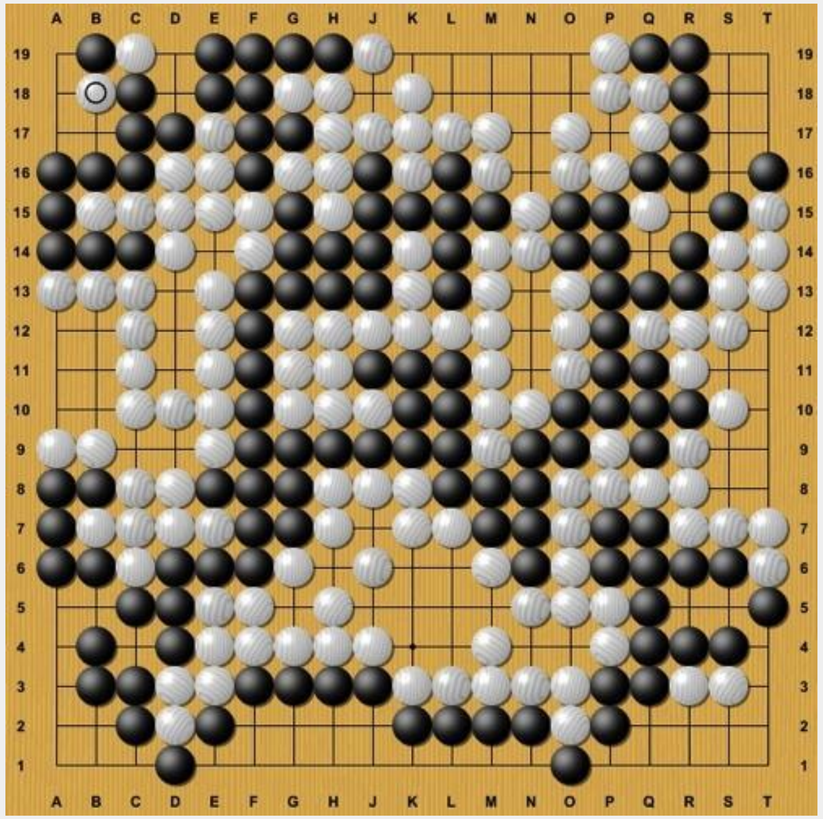


Machine Learning Ecosystem

The Machine Learning Ecosystem

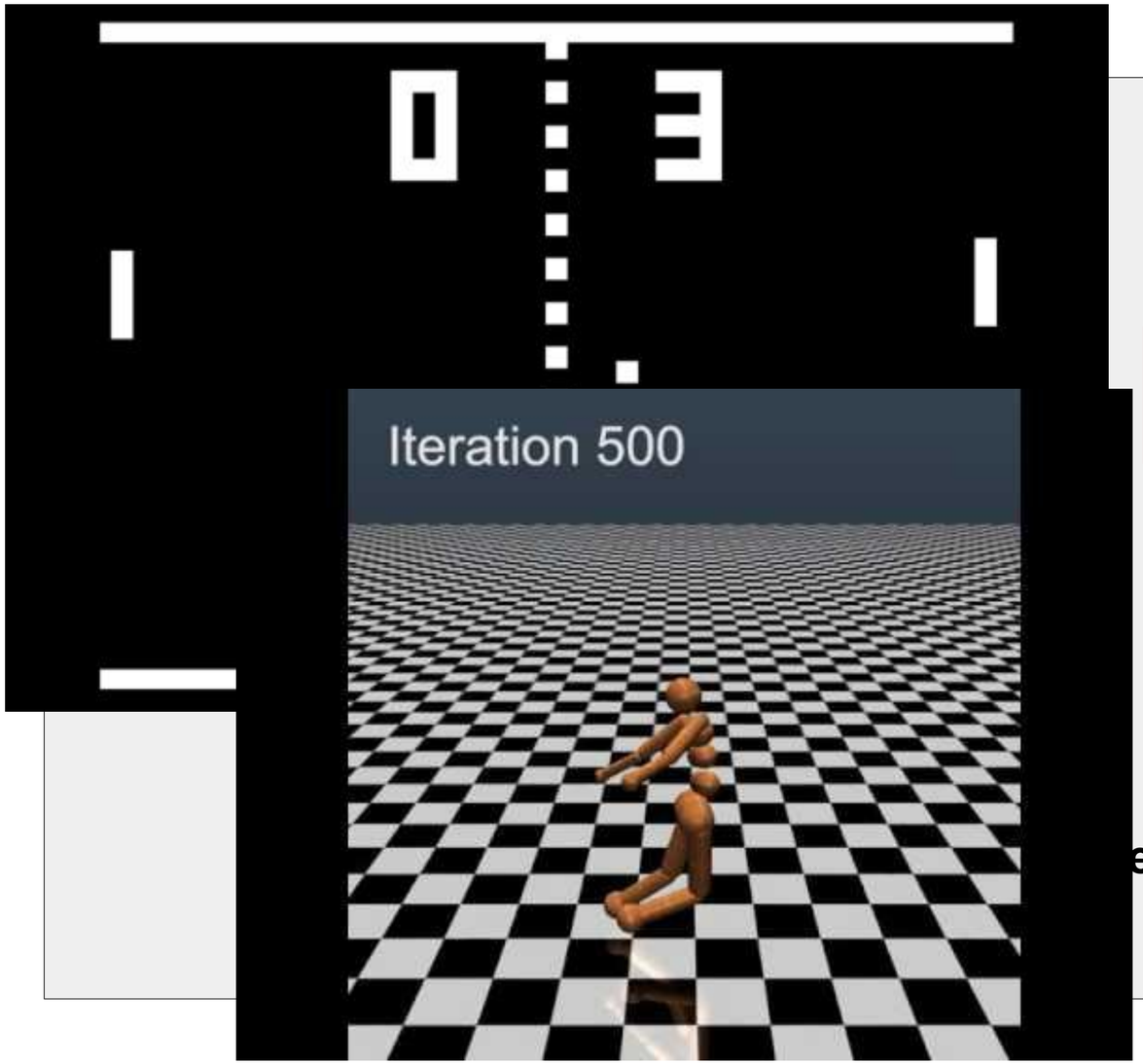


RL



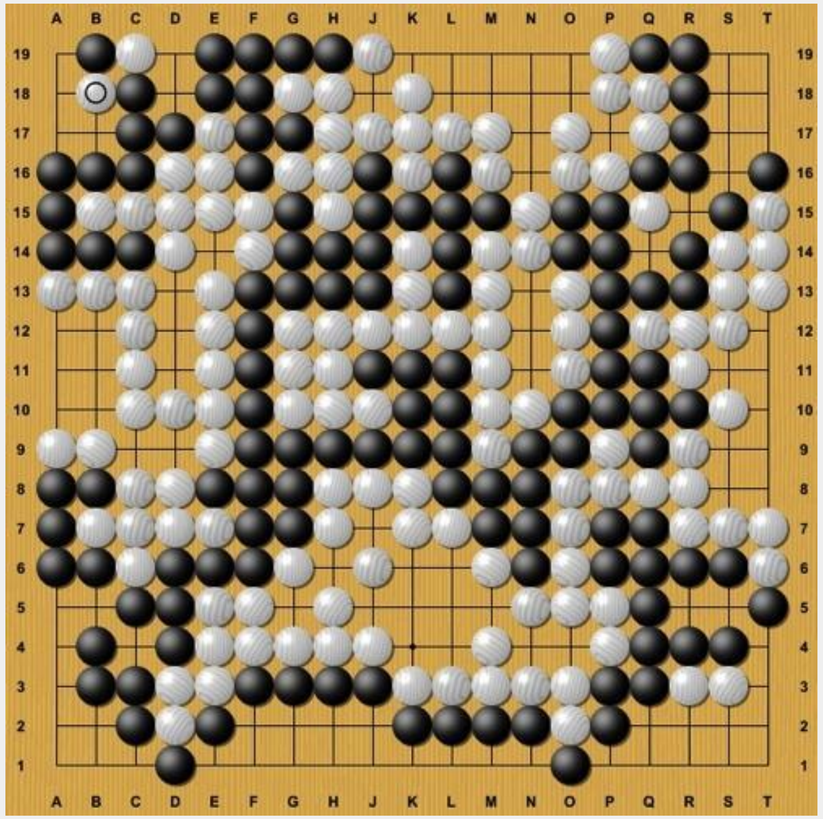
Machine Learning Ecosystem

The Machine Learning Ecosystem

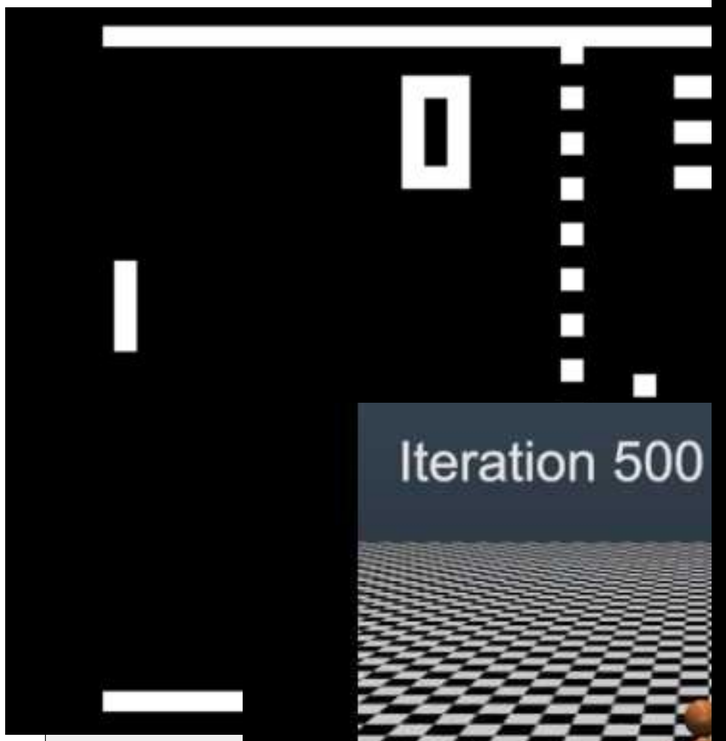


RL

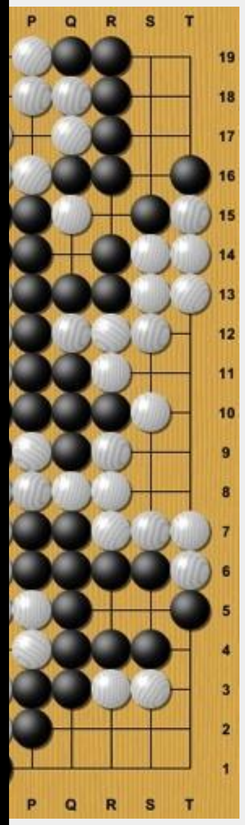
earning Ecosystem



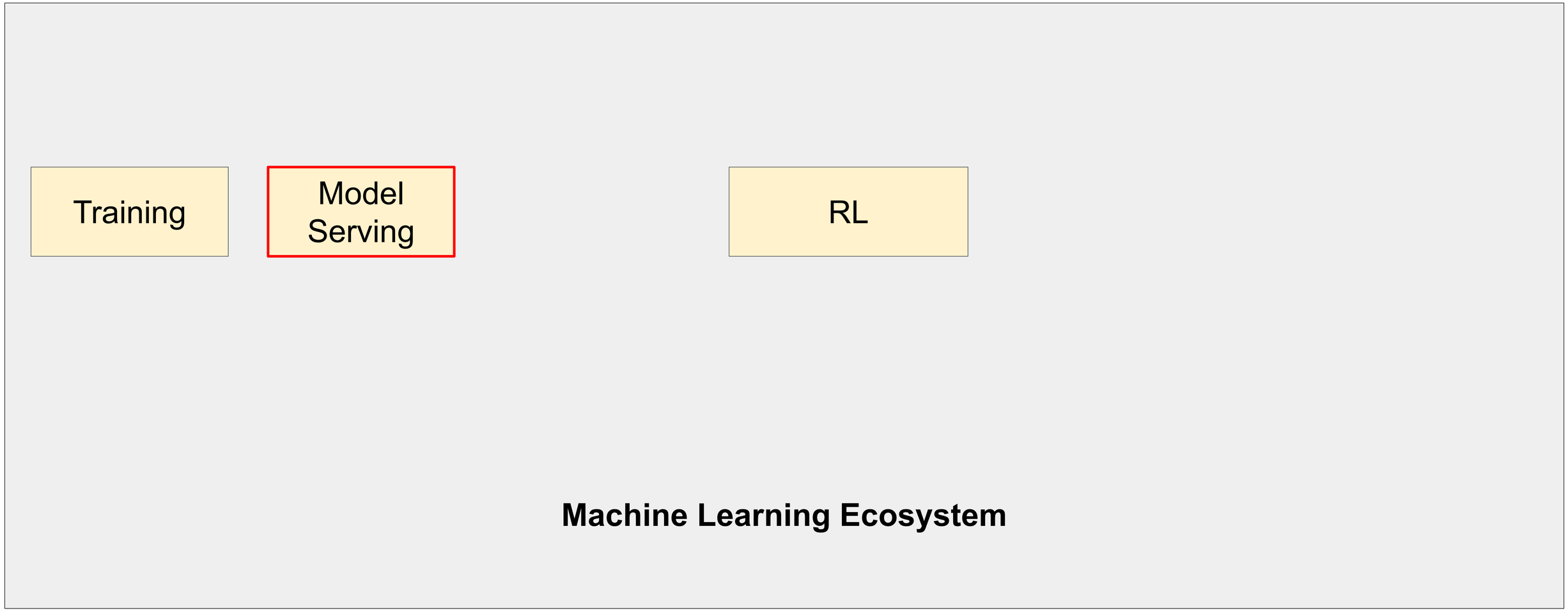
The Machine



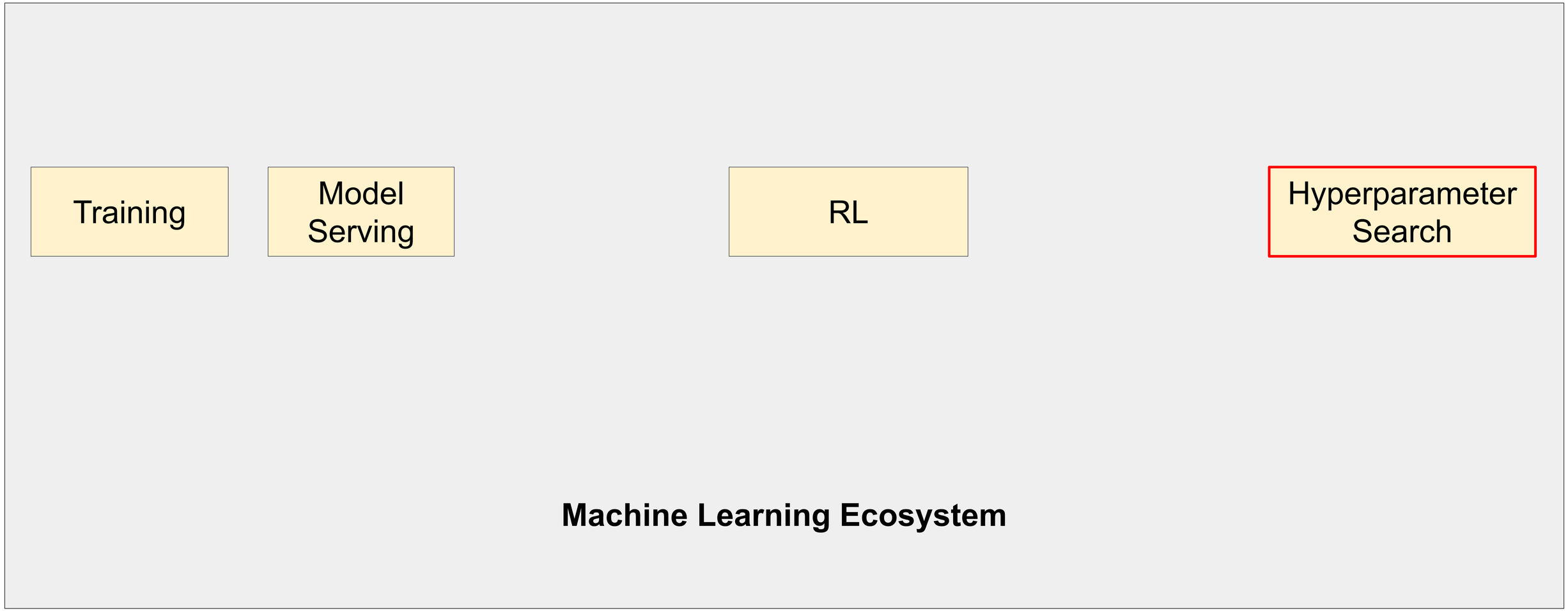

Iteration 500


P Q R S T
19
18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
P Q R S T

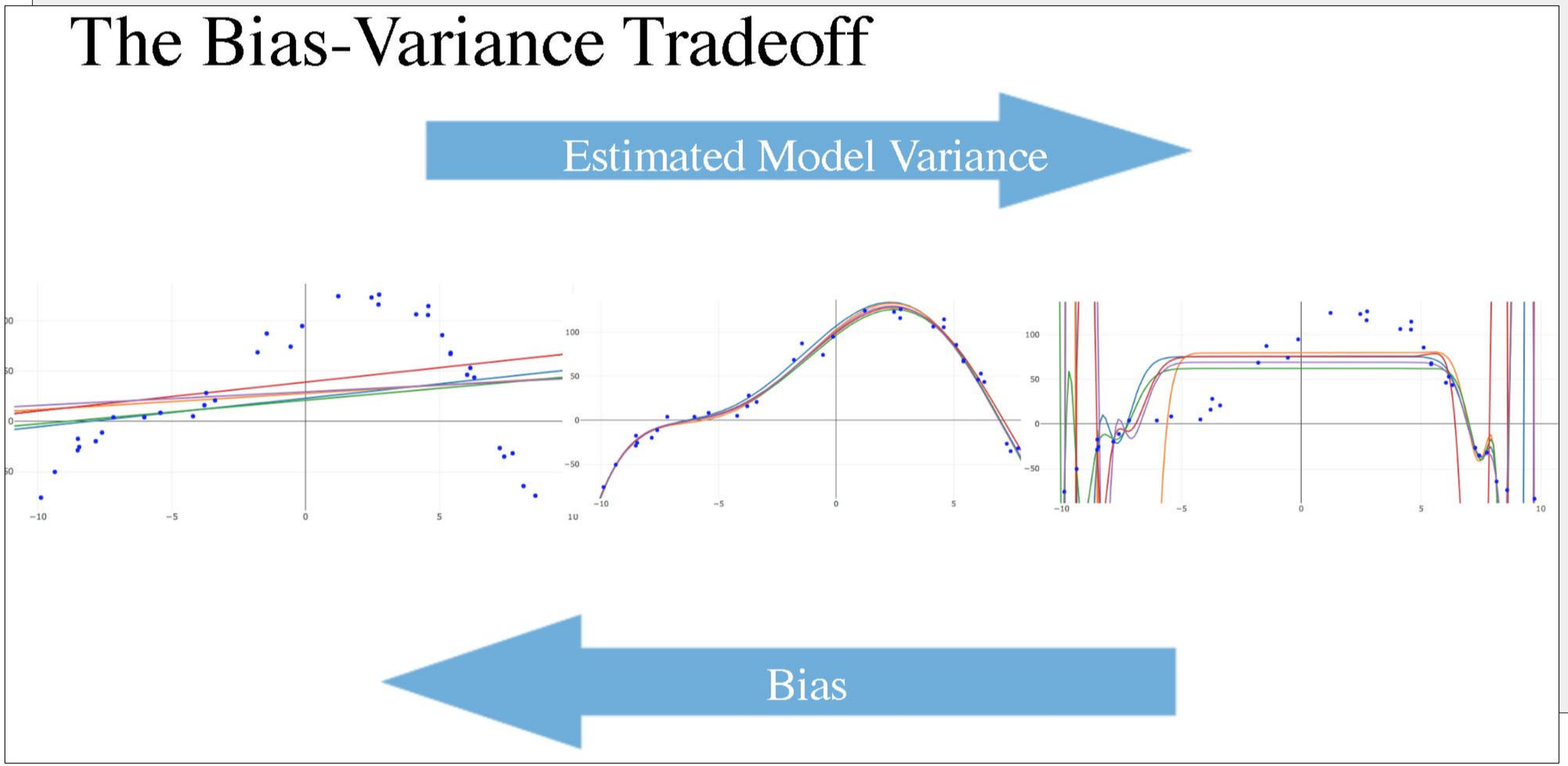
The Machine Learning Ecosystem



The Machine Learning Ecosystem

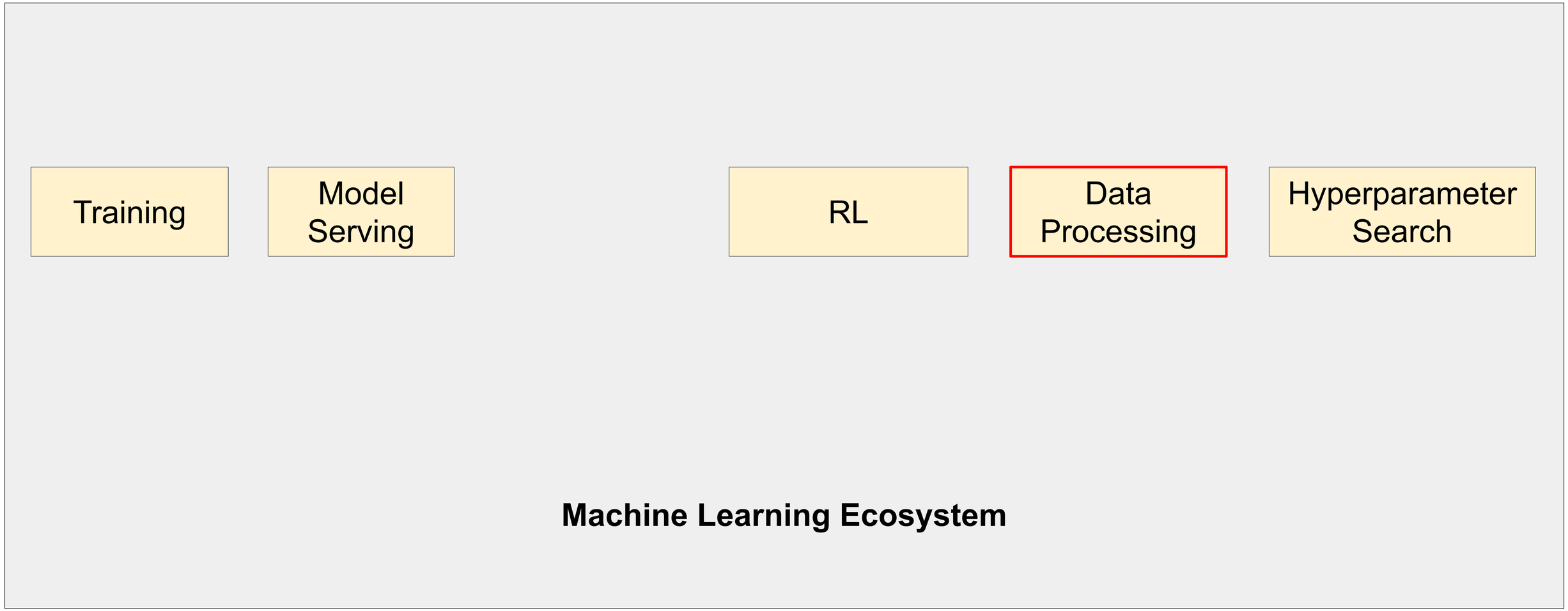


The Machine Learning Ecosystem

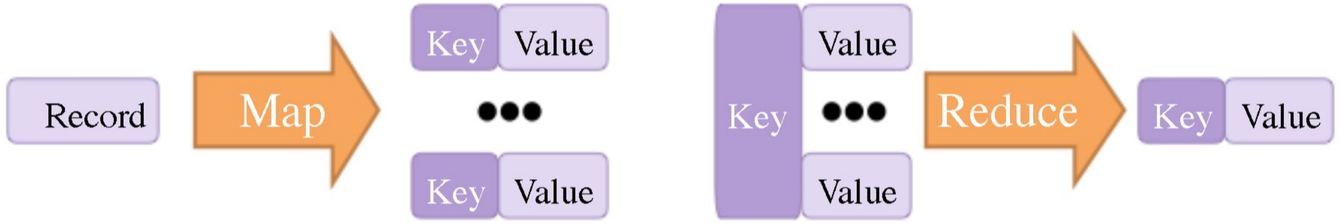


Hyperparameter Search

The Machine Learning Ecosystem



The Map Reduce Abstraction (Simpler)



Example: *Word-Count*

```
Map(book):
 for (word in book):
  emit (word, 1)
```

Key | Value

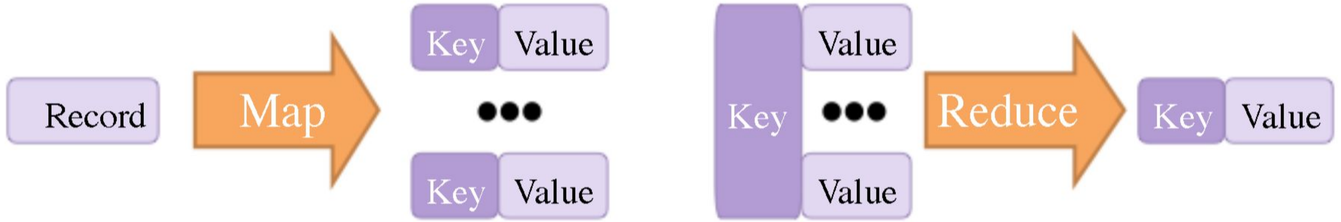
```
Reduce(word, counts) {
 sum = 0
 for count in counts:
    sum += count
 emit (word,
SUM(counts))
}
```

Data Processing

Hyperparameter Search

Machine Learning Ecosystem

The Map Reduce Abstraction (Simpler)



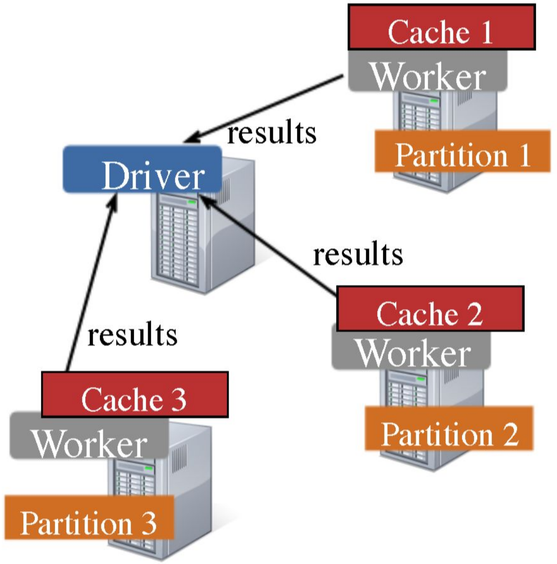
Example: *Word-Count*

Reduce(word, counts) {

Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

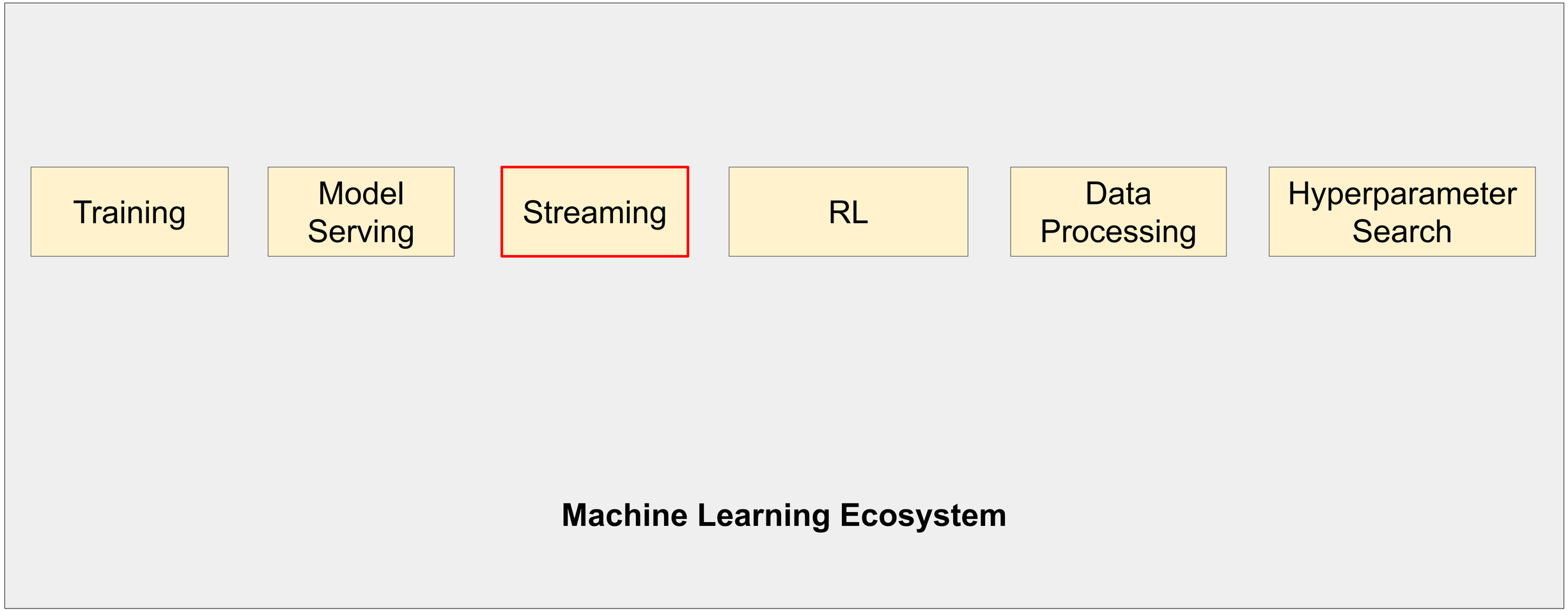
```
lines = spark.textFile("hdfs://file.txt")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()

messages.filter(lambda s: "mysql" in s).count()
messages.filter(lambda s: "php" in s).count()
```



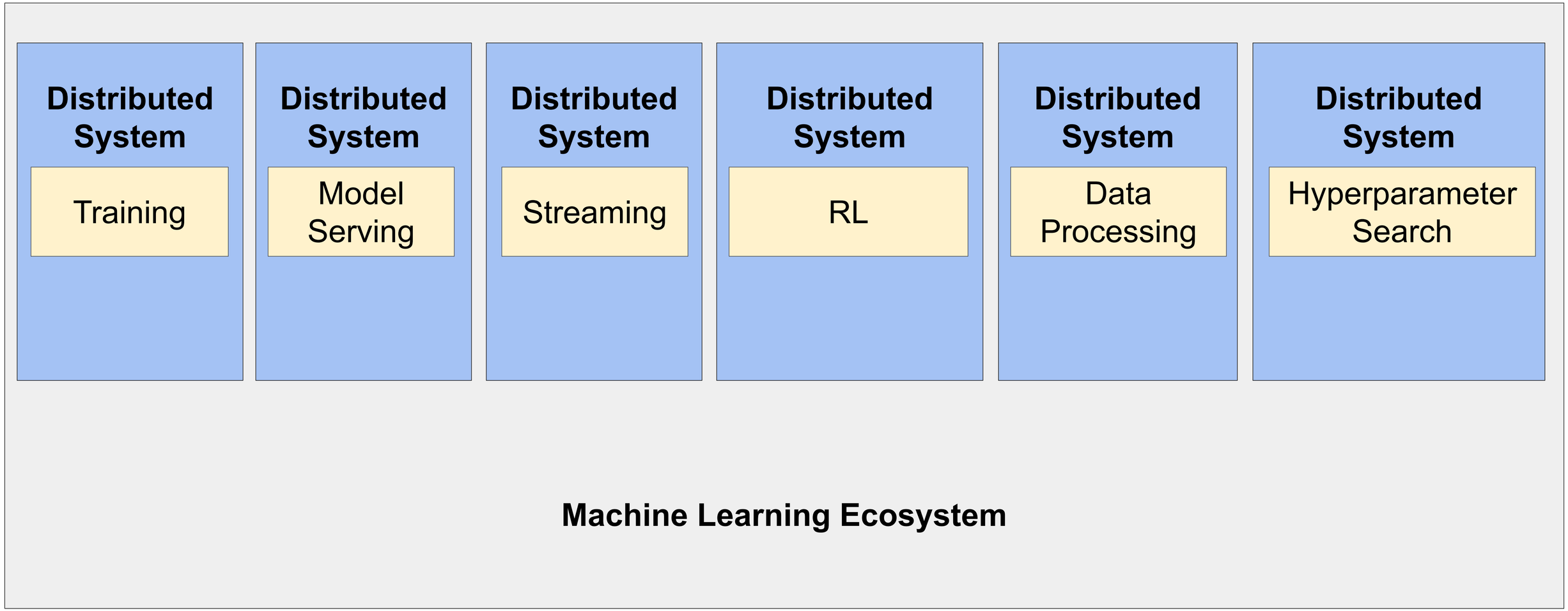
Data Processing

Hyperparameter Search

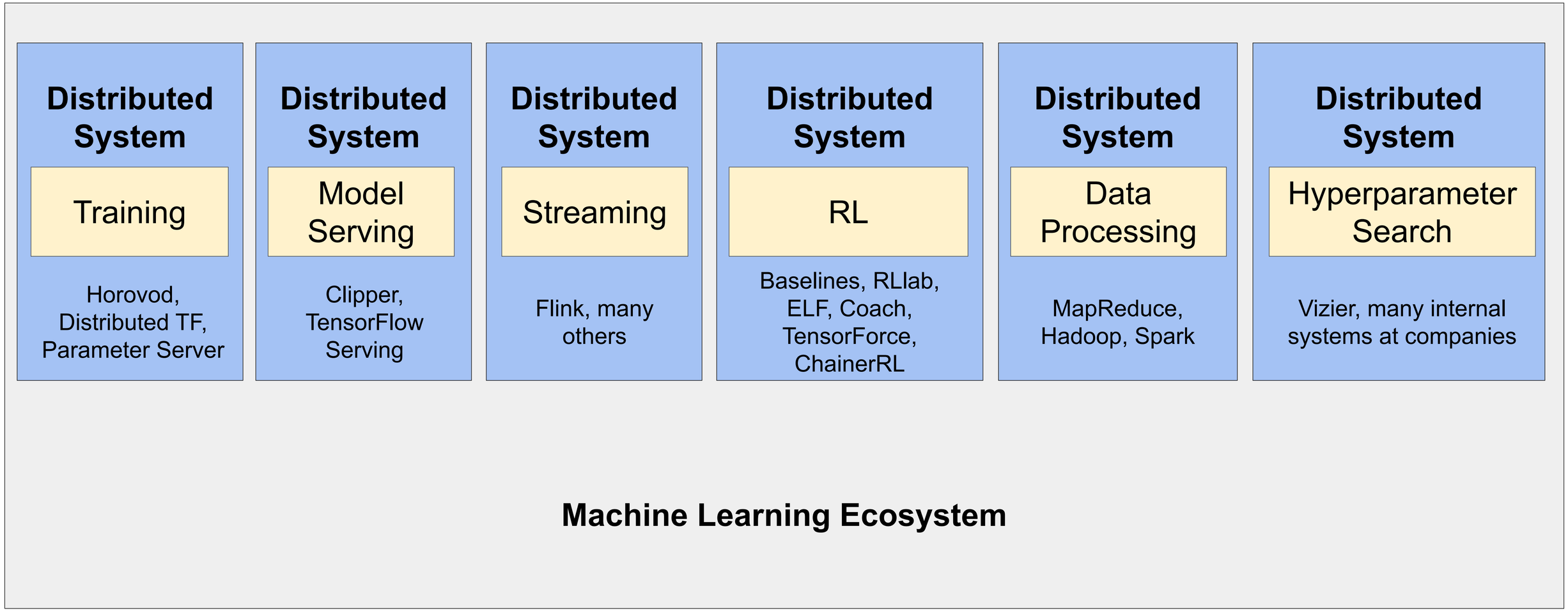
The Machine Learning Ecosystem



The Machine Learning Ecosystem



The Machine Learning Ecosystem



The Machine Learning Ecosystem

Distributed System

Training

Horovod, Distributed TF, Parameter Server

Distributed System

Model Serving

Clipper, TensorFlow Serving

Distributed System

Streaming

Flink, many others

Distributed System

RL

Baselines, RLlab, ELF, Coach, TensorForce, ChainerRL

Distributed System

Data Processing

MapReduce, Hadoop, Spark

Distributed System

Hyperparameter Search

Vizier, many internal systems at companies

<u>Why distributed systems?</u>

The Machine Learning Ecosystem

Distributed System	Distributed System	Distributed System	Distributed System	Distributed System	Distributed System
Training	Model Serving	Streaming	RL	Data Processing	Hyperparameter Search
Horovod, Distributed TF, Parameter Server	Clipper, TensorFlow Serving	Flink, many others	Baselines, RLlab, ELF, Coach, TensorForce, ChainerRL	MapReduce, Hadoop, Spark	Vizier, many internal systems at companies

<u>Why distributed systems?</u>

- More **work** (computation) than one machine can do in a reasonable amount of time.
- More **data** than can fit in one machine.

The Machine Learning Ecosystem

Distributed System	Distributed System	Distributed System	Distributed System	Distributed System	Distributed System
Training	Model Serving	Streaming	RL	Data Processing	Hyperparameter Search
Horovod, Distributed TF, Parameter Server	Clipper, TensorFlow Serving	Flink, many others	Baselines, RLlab, ELF, Coach, TensorForce, ChainerRL	MapReduce, Hadoop, Spark	Vizier, many internal systems at companies

<u>Aspects of a distributed system</u>

<u>Why distributed systems?</u>

- More **work** (computation) than one machine can do in a reasonable amount of time.
- More **data** than can fit in one machine.

The Machine Learning Ecosystem

Distributed System	Distributed System	Distributed System	Distributed System	Distributed System	Distributed System
Training	Model Serving	Streaming	RL	Data Processing	Hyperparameter Search
Horovod, Distributed TF, Parameter Server	Clipper, TensorFlow Serving	Flink, many others	Baselines, RLlab, ELF, Coach, TensorForce, ChainerRL	MapReduce, Hadoop, Spark	Vizier, many internal systems at companies

<u>Aspects of a distributed system</u>

- Units of work “tasks” executed in parallel
- Scheduling (which tasks run on which machines and when)
- Data transfer
- Failure handling
- Resource management (CPUs, GPUs, memory)

<u>Why distributed systems?</u>

- More **work** (computation) than one machine can do in a reasonable amount of time.
- More **data** than can fit in one machine.

The Machine Learning Ec

Why is this a problem?

Distributed System	Distributed System	Distributed System	Distributed System	Distributed System	Distributed System
Training	Model Serving	Streaming	RL	Data Processing	Hyperparameter Search
Horovod, Distributed TF, Parameter Server	Clipper, TensorFlow Serving	Flink, many others	Baselines, RLlab, ELF, Coach, TensorForce, ChainerRL	MapReduce, Hadoop, Spark	Vizier, many internal systems at companies

<u>Aspects of a distributed system</u>

- Units of work “tasks” executed in parallel
- Scheduling (which tasks run on which machines and when)
- Data transfer
- Failure handling
- Resource management (CPUs, GPUs, memory)

<u>Why distributed systems?</u>

- More **work** (computation) than one machine can do in a reasonable amount of time.
- More **data** than can fit in one machine.

What is Ray?

Training | Model Serving | Streaming | RL | Data Processing | Hyperparameter Search

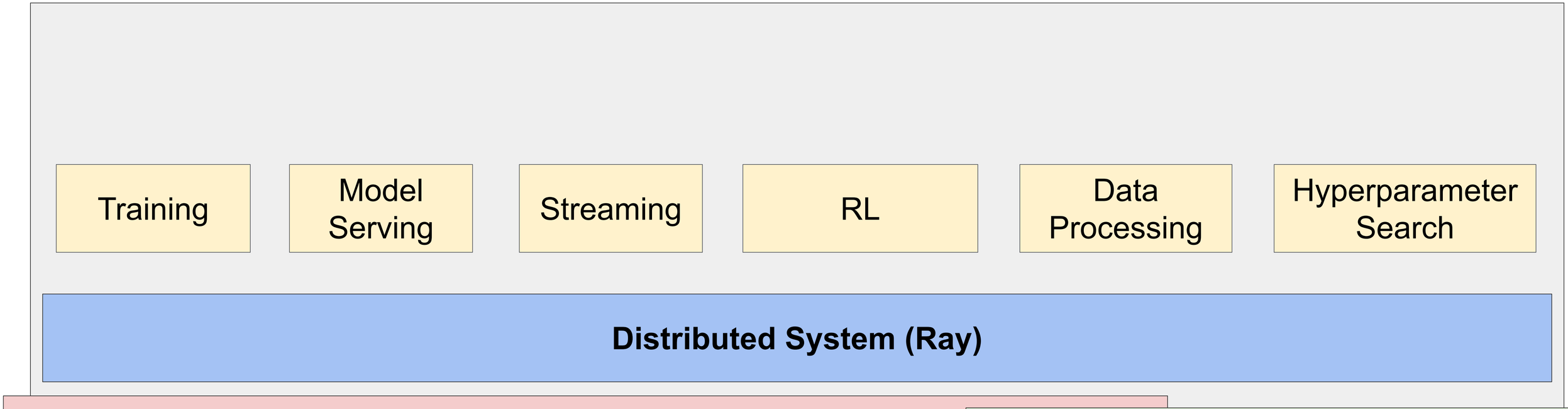
<u>Aspects of a distributed system</u>

- Units of work “tasks” executed in parallel
- Scheduling (which tasks run on which machines and when)
- Data transfer
- Failure handling
- Resource management (CPUs, GPUs, memory)

<u>Why distributed systems?</u>

- More **work** (computation) than one machine can do in a reasonable amount of time.
- More **data** than can fit in one machine.

What is Ray?



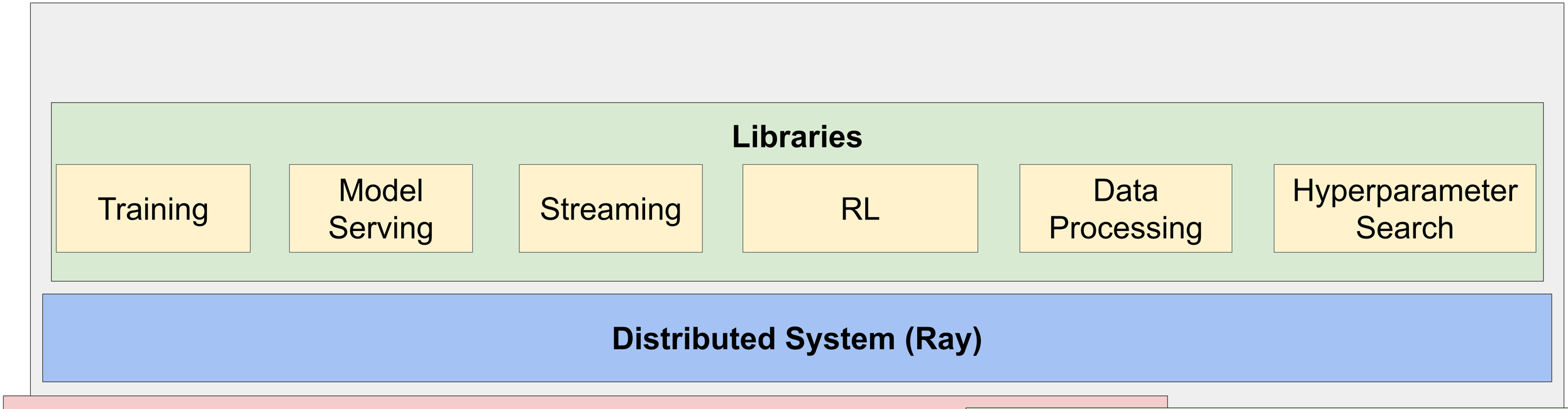
<u>Aspects of a distributed system</u>

- Units of work “tasks” executed in parallel
- Scheduling (which tasks run on which machines and when)
- Data transfer
- Failure handling
- Resource management (CPUs, GPUs, memory)

<u>Why distributed systems?</u>

- More **work** (computation) than one machine can do in a reasonable amount of time.
- More **data** than can fit in one machine.

What is Ray?



<u>Aspects of a distributed system</u>

- Units of work “tasks” executed in parallel
- Scheduling (which tasks run on which machines and when)
- Data transfer
- Failure handling
- Resource management (CPUs, GPUs, memory)

<u>Why distributed systems?</u>

- More **work** (computation) than one machine can do in a reasonable amount of time.
- More **data** than can fit in one machine.

Distributed System

Training

Horovod, Distributed TF, Parameter Server

Distributed System

Model Serving

Clipper, TensorFlow Serving

Distributed System

Streaming

Flink, many others

Distributed System

RL

Baselines, RLlab, ELF, Coach, TensorForce, ChainerRL

Distributed System

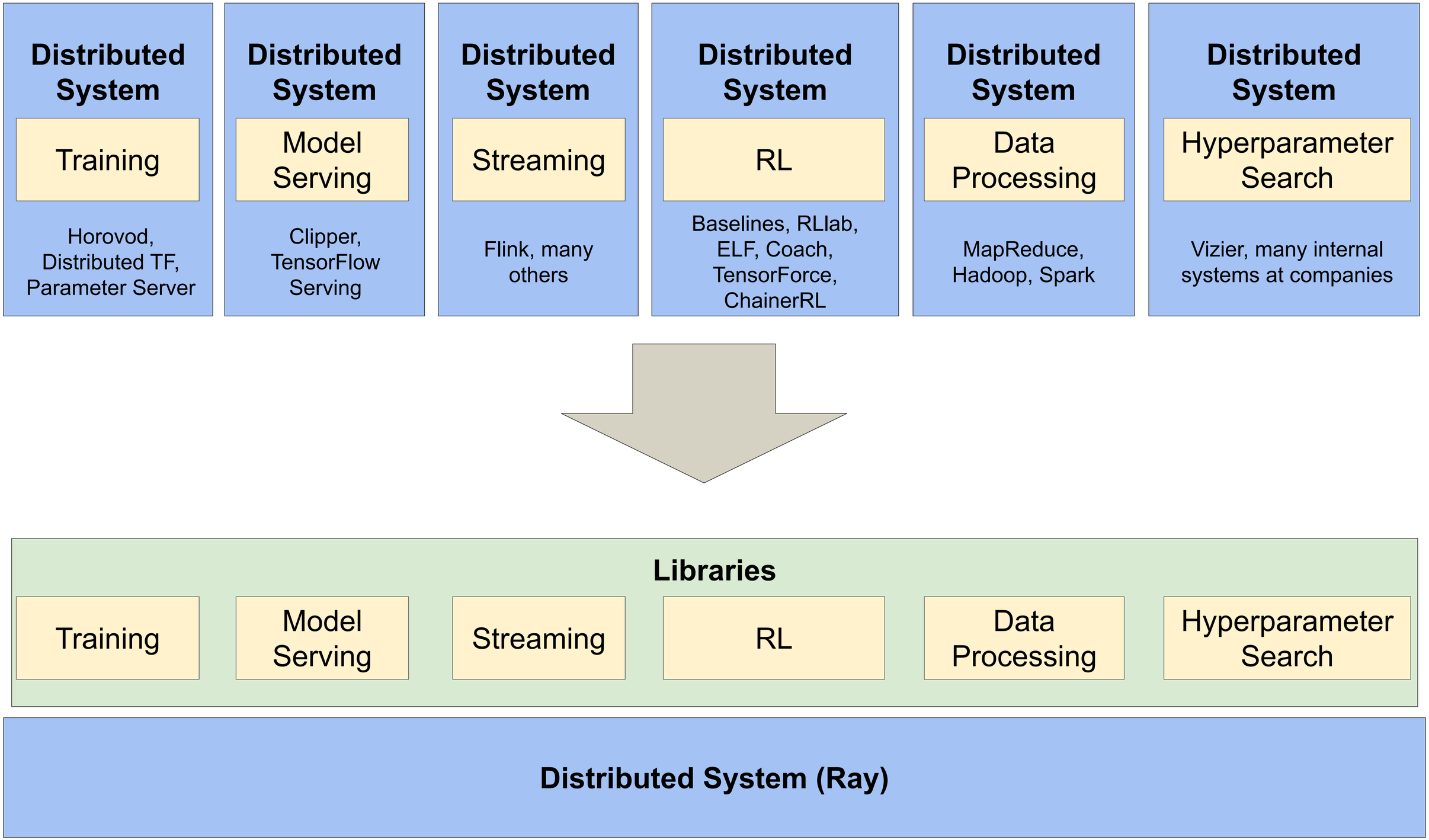
Data Processing

MapReduce, Hadoop, Spark

Distributed System

Hyperparameter Search

Vizier, many internal systems at companies


Distributed System
Training
Horovod, Distributed TF, Parameter Server
Distributed System
Model Serving
Clipper, TensorFlow Serving
Distributed System
Streaming
Flink, many others
Distributed System
RL
Baselines, RLlab, ELF, Coach, TensorForce, ChainerRL
Distributed System
Data Processing
MapReduce, Hadoop, Spark
Distributed System
Hyperparameter Search
Vizier, many internal systems at companies
Libraries
Training
Model Serving
Streaming
RL
Data Processing
Hyperparameter Search
Distributed System (Ray)

Distributed System	Distributed System	Distributed System	Distributed System	Distributed System	Distributed System
Training	Model Serving	Streaming	RL	Data Processing	Hyperparameter Search
Horovod, Distributed TF, Parameter Server	Clipper, TensorFlow Serving	Flink, many others	Baselines, RLlab, ELF, Coach, TensorForce, ChainerRL	MapReduce, Hadoop, Spark	Vizier, many internal systems at companies

This requires a very **general** underlying distributed system.

Libraries

Training	Model Serving	Streaming	RL	Data Processing	Hyperparameter Search

Distributed System (Ray)

Distributed System	Distributed System	Distributed System	Distributed System	Distributed System	Distributed System
Training	Model Serving	Streaming	RL	Data Processing	Hyperparameter Search
Horovod, Distributed TF, Parameter Server	Clipper, TensorFlow Serving	Flink, many others	Baselines, RLlab, ELF, Coach, TensorForce, ChainerRL	MapReduce, Hadoop, Spark	Vizier, many internal systems at companies

This requires a very **general** underlying distributed system.

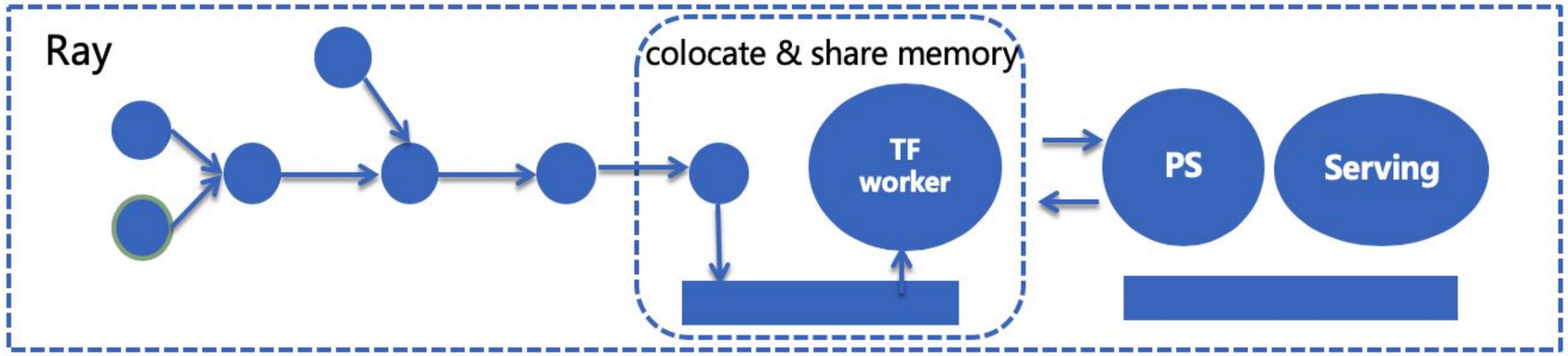
Generality comes from **tasks (functions)** and **actors (classes).**

Libraries

Training	Model Serving	Streaming	RL	Data Processing	Hyperparameter Search

Distributed System (Ray)

Use Case: Online Machine Learning



- 3 min, streaming + model training, from feature / label to model output
- 5 min, streaming + training + serving, from feature / label to model deploy
- 5% CTR improvement comparing to offline model; 1% CTR improvement comparing to blink solution

Ray API

Functions -> Tasks

```
def read_array(file):
    # read array “a” from “file”
    return a


def add(a, b):
    return np.add(a, b)
```

Ray API

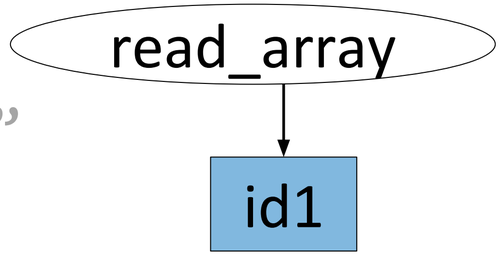
Functions -> Tasks

```
@ray.remote
def read_array(file):
    # read array “a” from “file”
    return a

@ray.remote
def add(a, b):
    return np.add(a, b)
```

Ray API

Functions -> Tasks

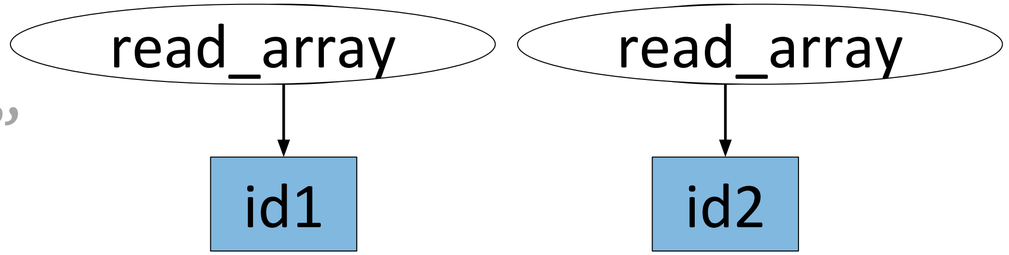
```
@ray.remote
def read_array(file):
    # read array “a” from “file”
    return a

@ray.remote
def add(a, b):
    return np.add(a, b)

id1 = read_array.remote([5, 5])
```



Ray API

Functions -> Tasks

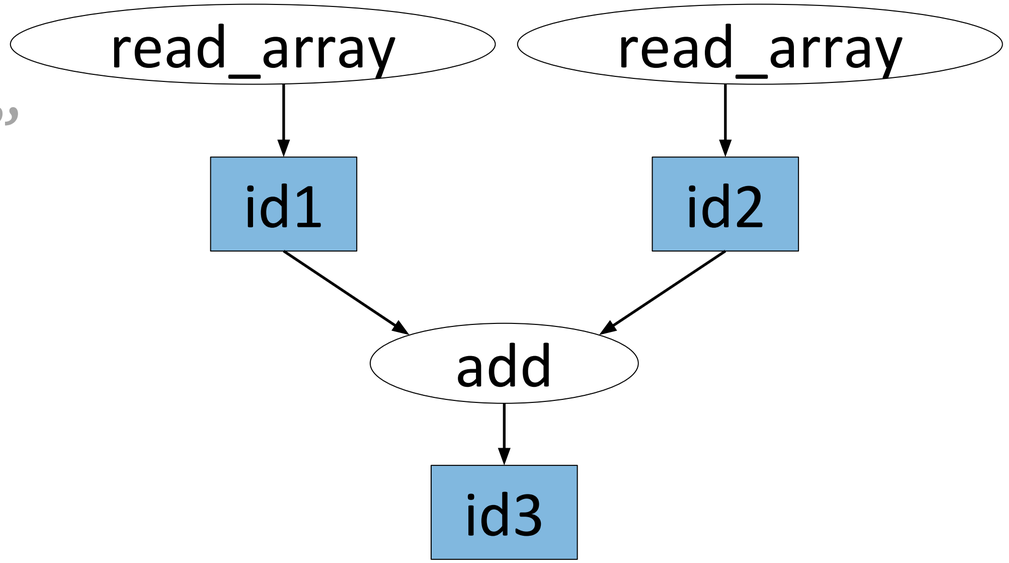
```
@ray.remote
def read_array(file):
    # read array “a” from “file”
    return a

@ray.remote
def add(a, b):
    return np.add(a, b)

id1 = read_array.remote([5, 5])
id2 = read_array.remote([5, 5])
```



Ray API

Functions -> Tasks

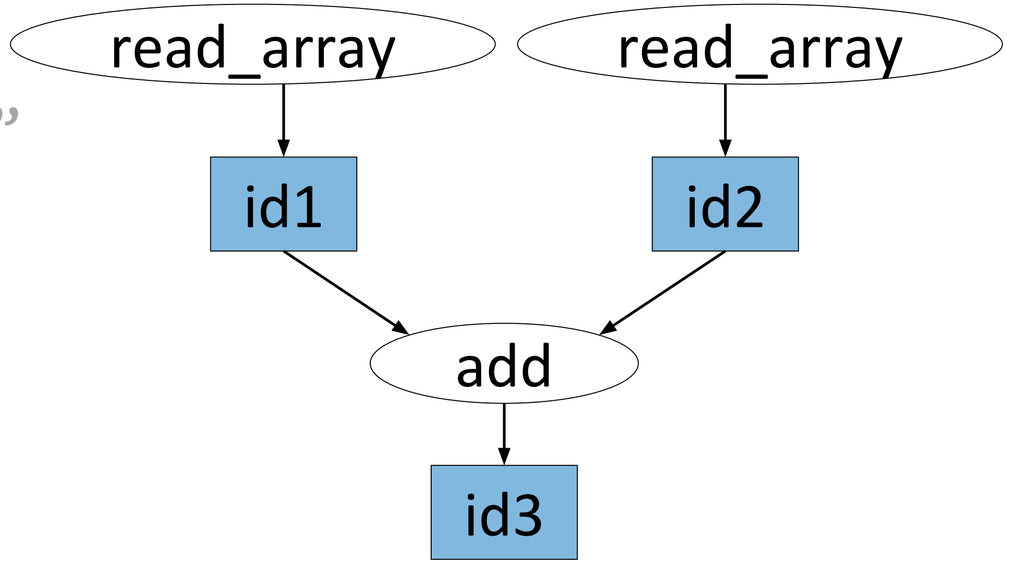
```
@ray.remote
def read_array(file):
    # read array “a” from “file”
    return a

@ray.remote
def add(a, b):
    return np.add(a, b)

id1 = read_array.remote([5, 5])
id2 = read_array.remote([5, 5])
id3 = add.remote(id1, id2)
```



Ray API

Functions -> Tasks

```
@ray.remote
def read_array(file):
    # read array “a” from “file”
    return a

@ray.remote
def add(a, b):
    return np.add(a, b)

id1 = read_array.remote([5, 5])
id2 = read_array.remote([5, 5])
id3 = add.remote(id1, id2)
ray.get(id3)
```



Ray API

Functions -> Tasks

```
@ray.remote
def read_array(file):
    # read array “a” from “file”
    return a

@ray.remote
def add(a, b):
    return np.add(a, b)

id1 = read_array.remote([5, 5])
id2 = read_array.remote([5, 5])
id3 = add.remote(id1, id2)
ray.get(id3)
```

Classes -> Actors

Ray API

Functions -> Tasks

```
@ray.remote
def read_array(file):
    # read array “a” from “file”
    return a

@ray.remote
def add(a, b):
    return np.add(a, b)

id1 = read_array.remote([5, 5])
id2 = read_array.remote([5, 5])
id3 = add.remote(id1, id2)
ray.get(id3)
```

Classes -> Actors

```
@ray.remote(num_gpus=1)
class Counter(object):
    def __init__(self):
        self.value = 0
    def inc(self):
        self.value += 1
        return self.value
```

Ray API

Functions -> Tasks

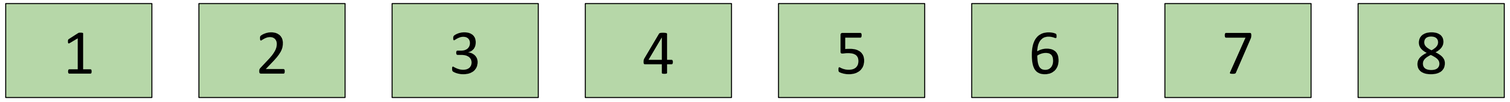
```
@ray.remote
def read_array(file):
    # read array “a” from “file”
    return a

@ray.remote
def add(a, b):
    return np.add(a, b)

id1 = read_array.remote([5, 5])
id2 = read_array.remote([5, 5])
id3 = add.remote(id1, id2)
ray.get(id3)
```

Classes -> 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])
```

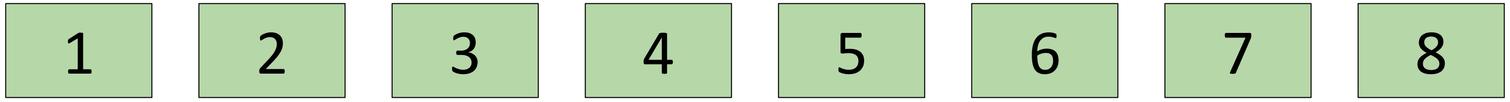
Parallelism Example: Tree Reduction

Parallelism Example: Tree Reduction



Add a bunch of values (2 at a time).

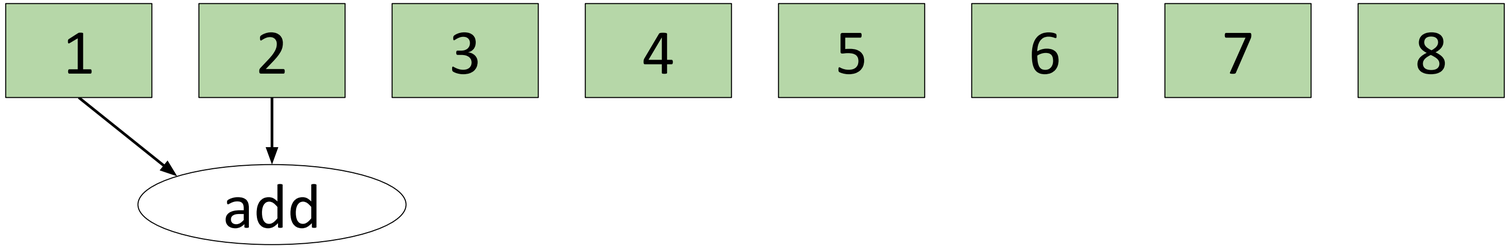
Parallelism Example: Tree Reduction




```
@ray.remote
def add(x, y):
    time.sleep(0.5)
    return x + y
```

Add a bunch of values (2 at a time).

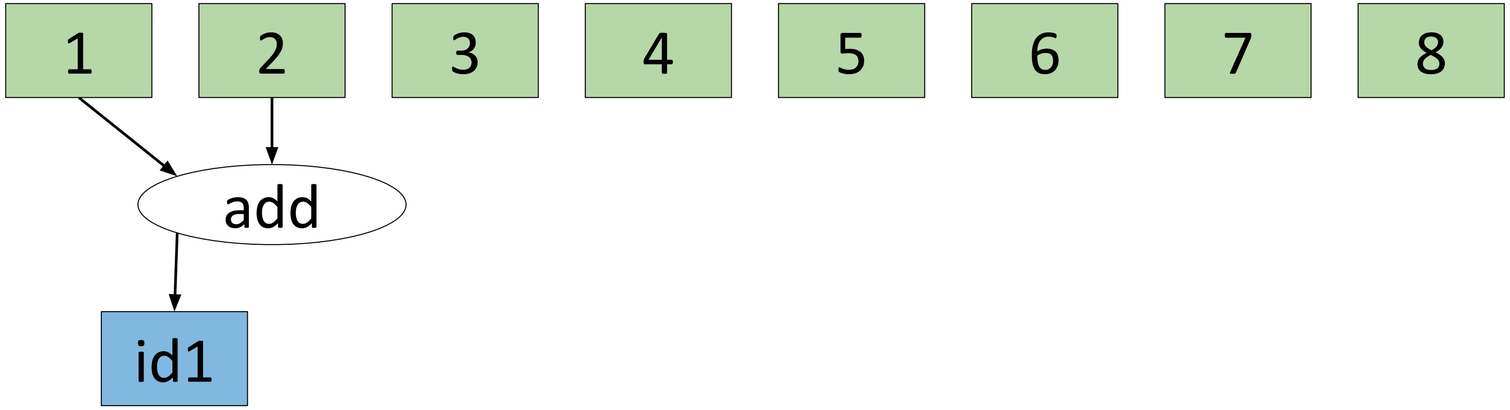
Parallelism Example: Tree Reduction




```
@ray.remote
def add(x, y):
    time.sleep(0.5)
    return x + y
```

Add a bunch of values (2 at a time).

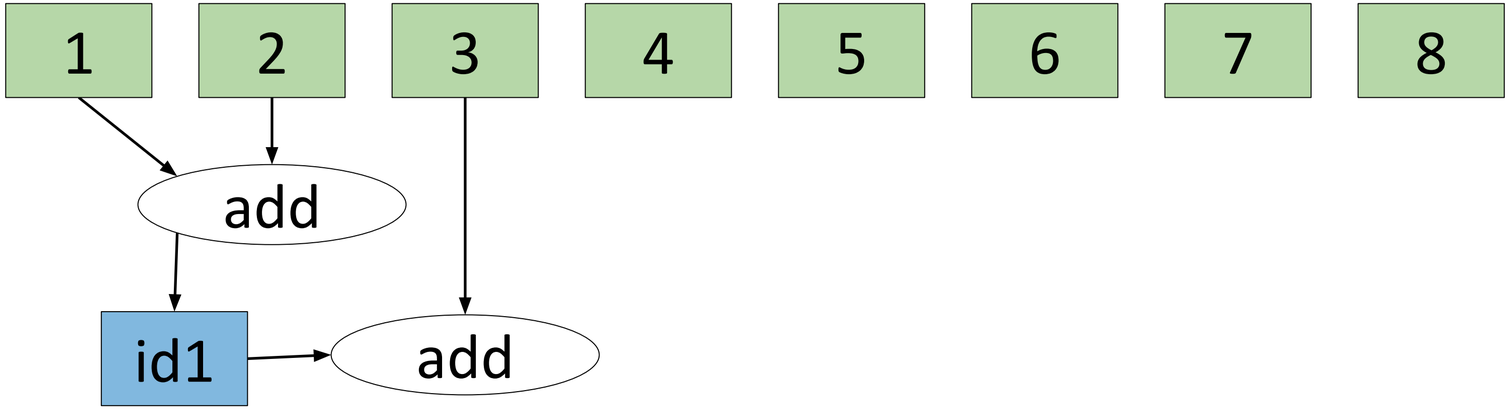
Parallelism Example: Tree Reduction




```
@ray.remote
def add(x, y):
    time.sleep(0.5)
    return x + y
```

Add a bunch of values (2 at a time).

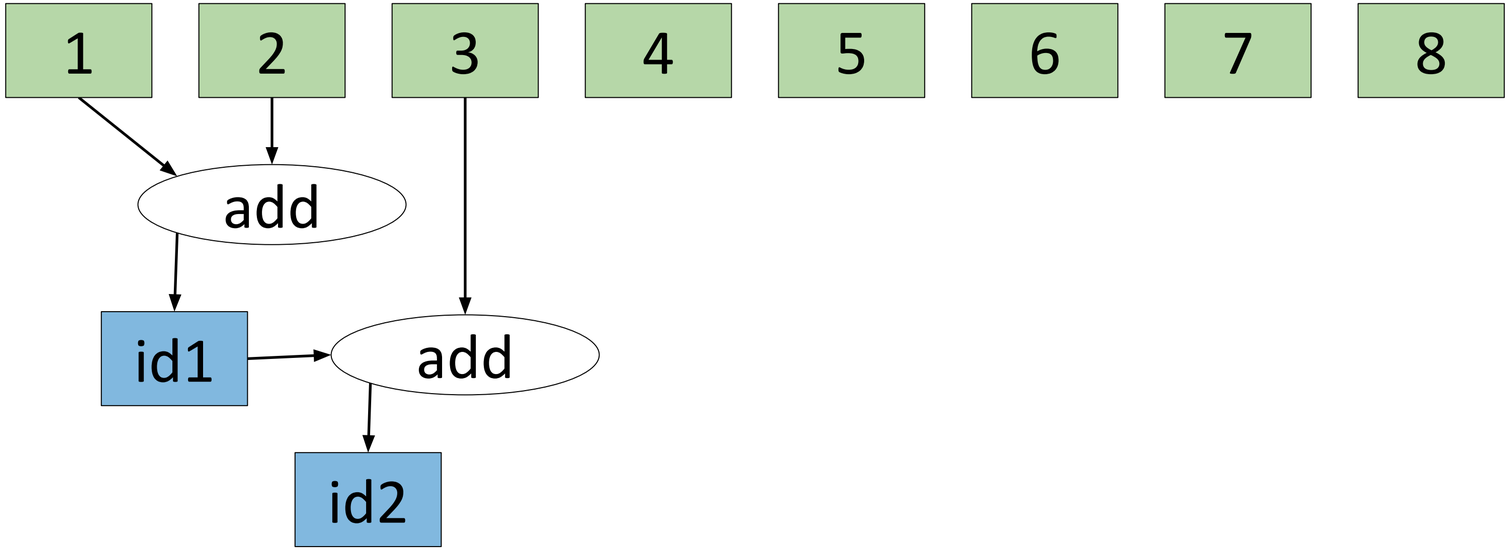
Parallelism Example: Tree Reduction




```
@ray.remote
def add(x, y):
    time.sleep(0.5)
    return x + y
```

Add a bunch of values (2 at a time).

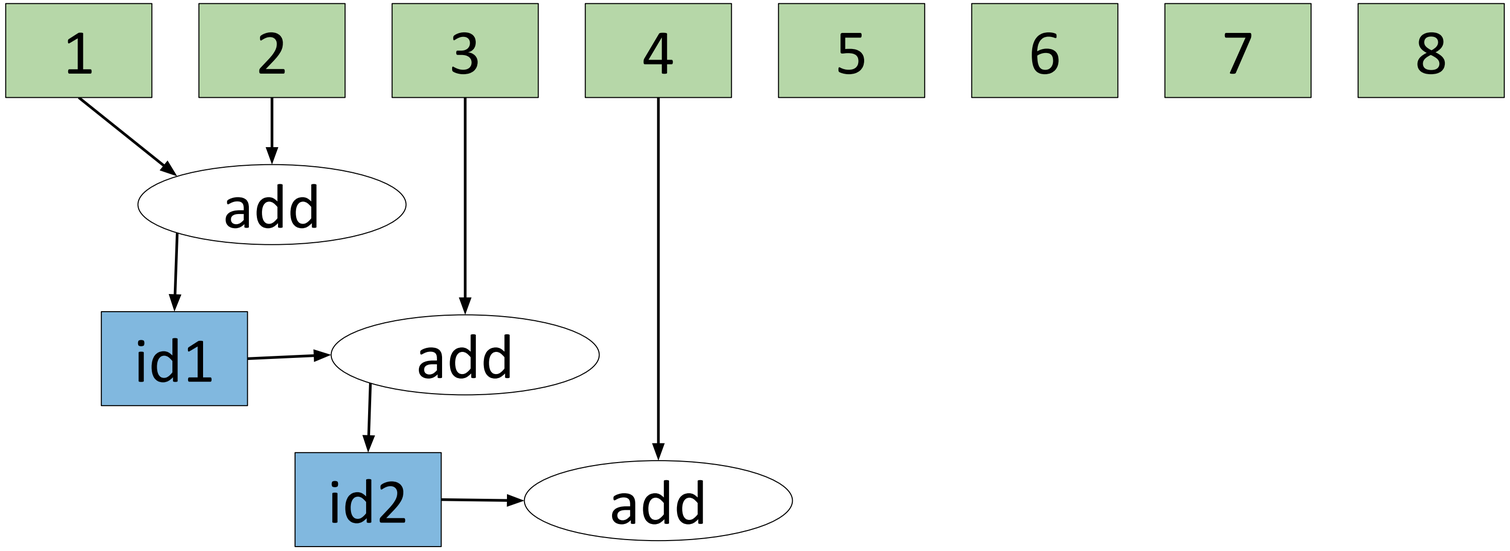
Parallelism Example: Tree Reduction




```
@ray.remote
def add(x, y):
    time.sleep(0.5)
    return x + y
```

Add a bunch of values (2 at a time).

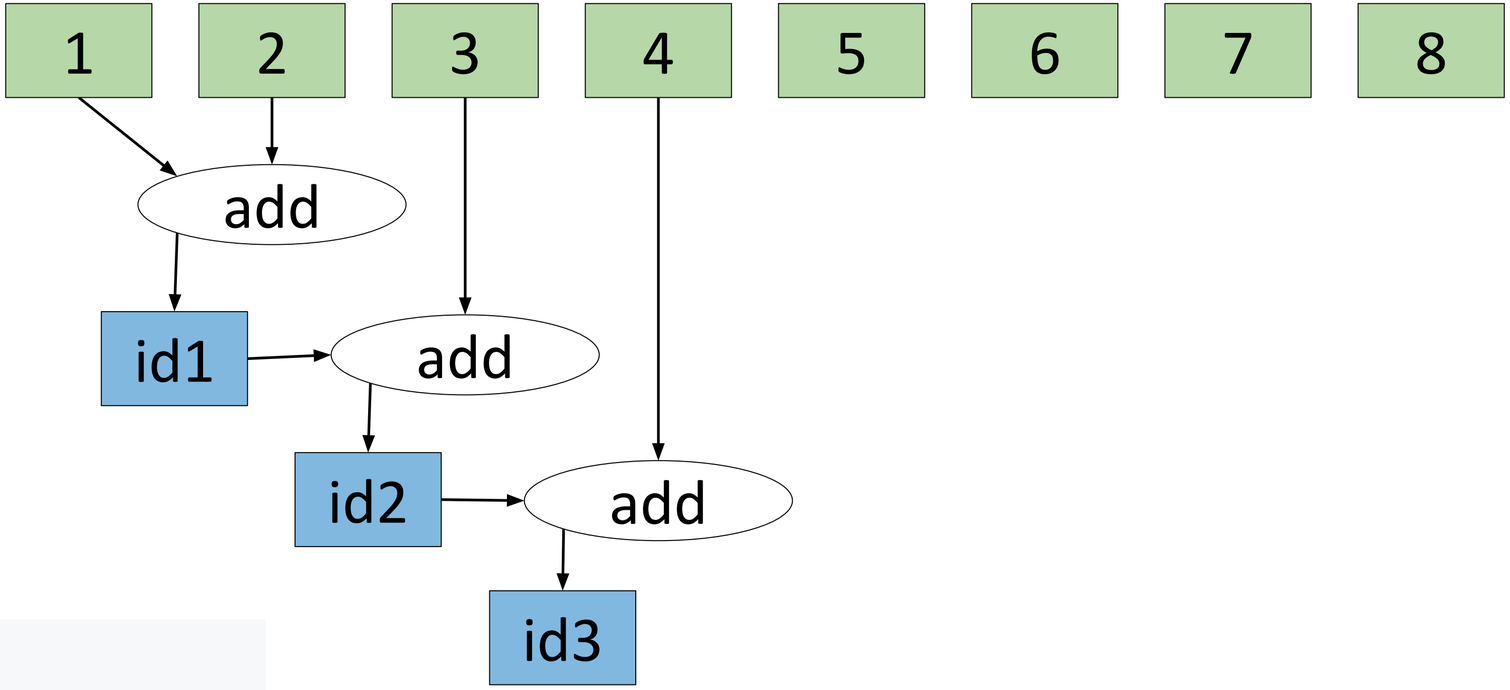
Parallelism Example: Tree Reduction




```
@ray.remote
def add(x, y):
    time.sleep(0.5)
    return x + y
```

Add a bunch of values (2 at a time).

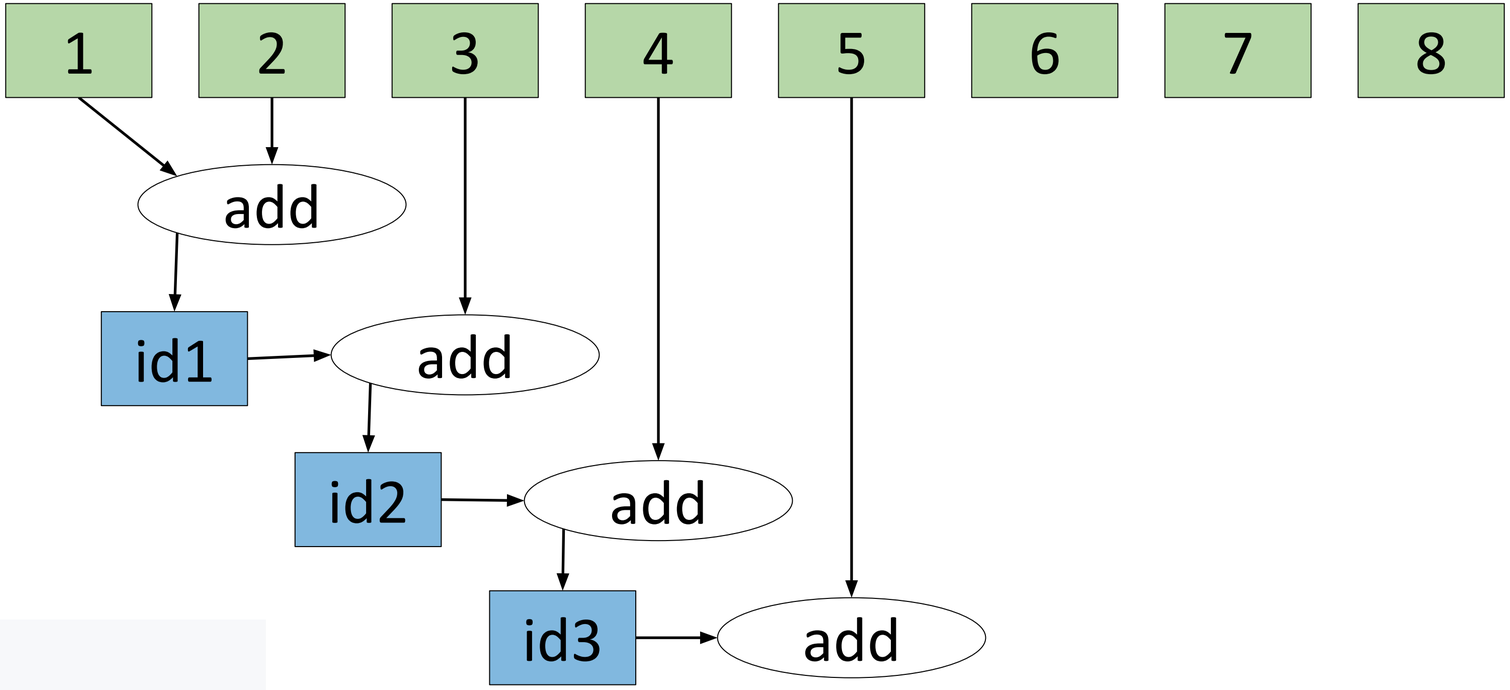
Parallelism Example: Tree Reduction




```python
@ray.remote
def add(x, y):
    time.sleep(0.5)
    return x + y
```

Add a bunch of values (2 at a time).

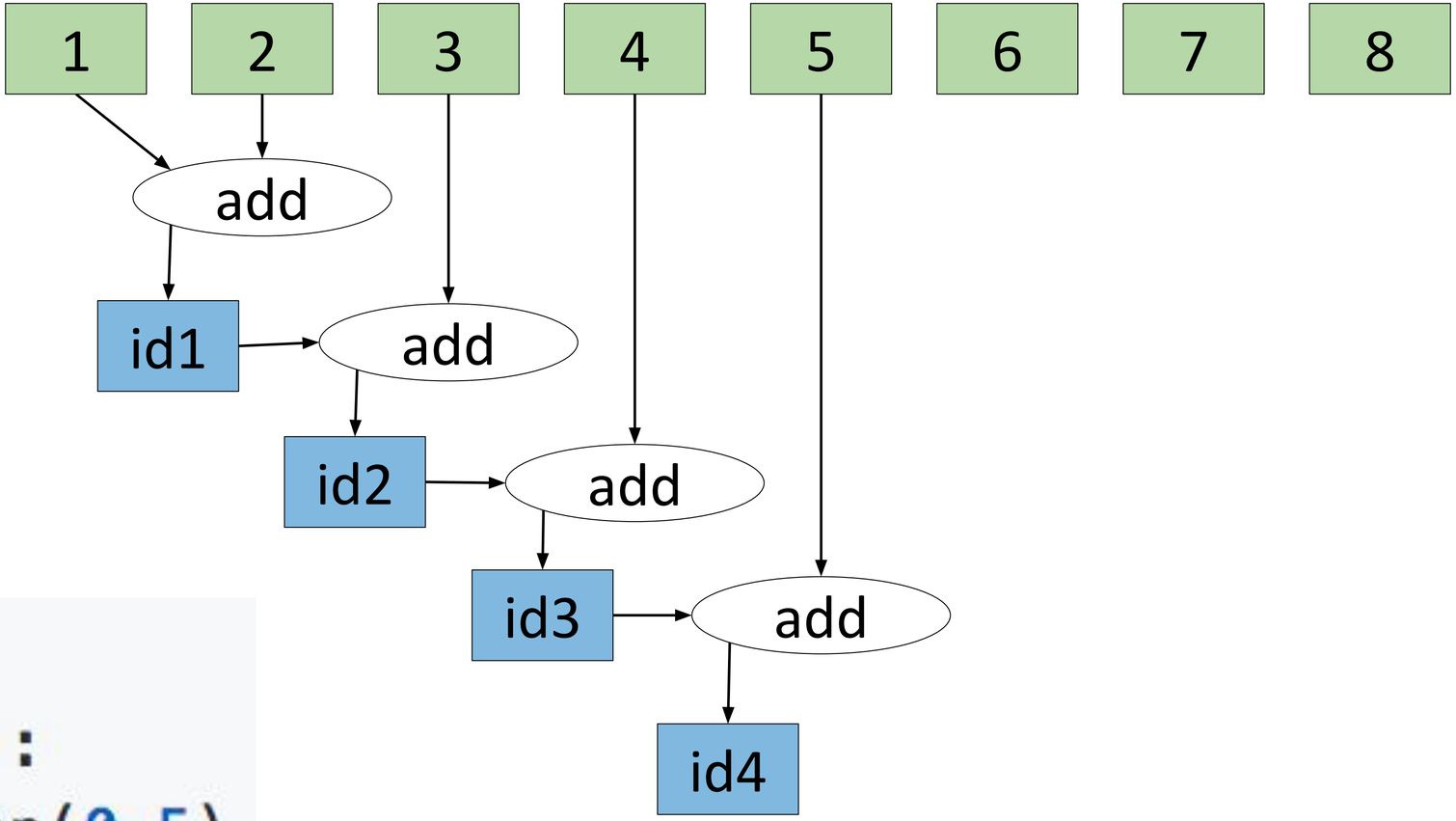
Parallelism Example: Tree Reduction




```python
@ray.remote
def add(x, y):
    time.sleep(0.5)
    return x + y
```

Add a bunch of values (2 at a time).

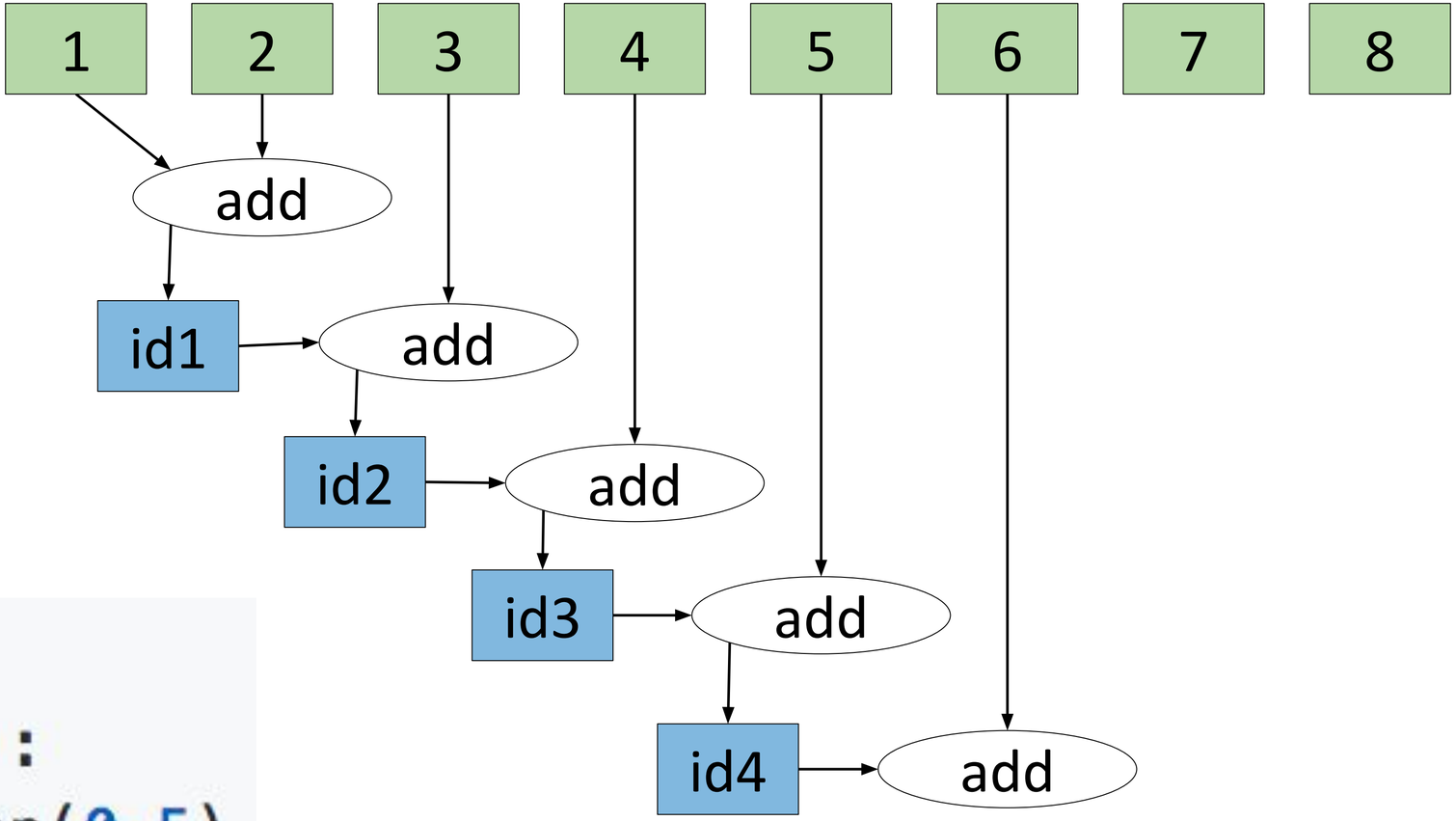
Parallelism Example: Tree Reduction




```python
@ray.remote
def add(x, y):
    time.sleep(0.5)
    return x + y
```

Add a bunch of values (2 at a time).

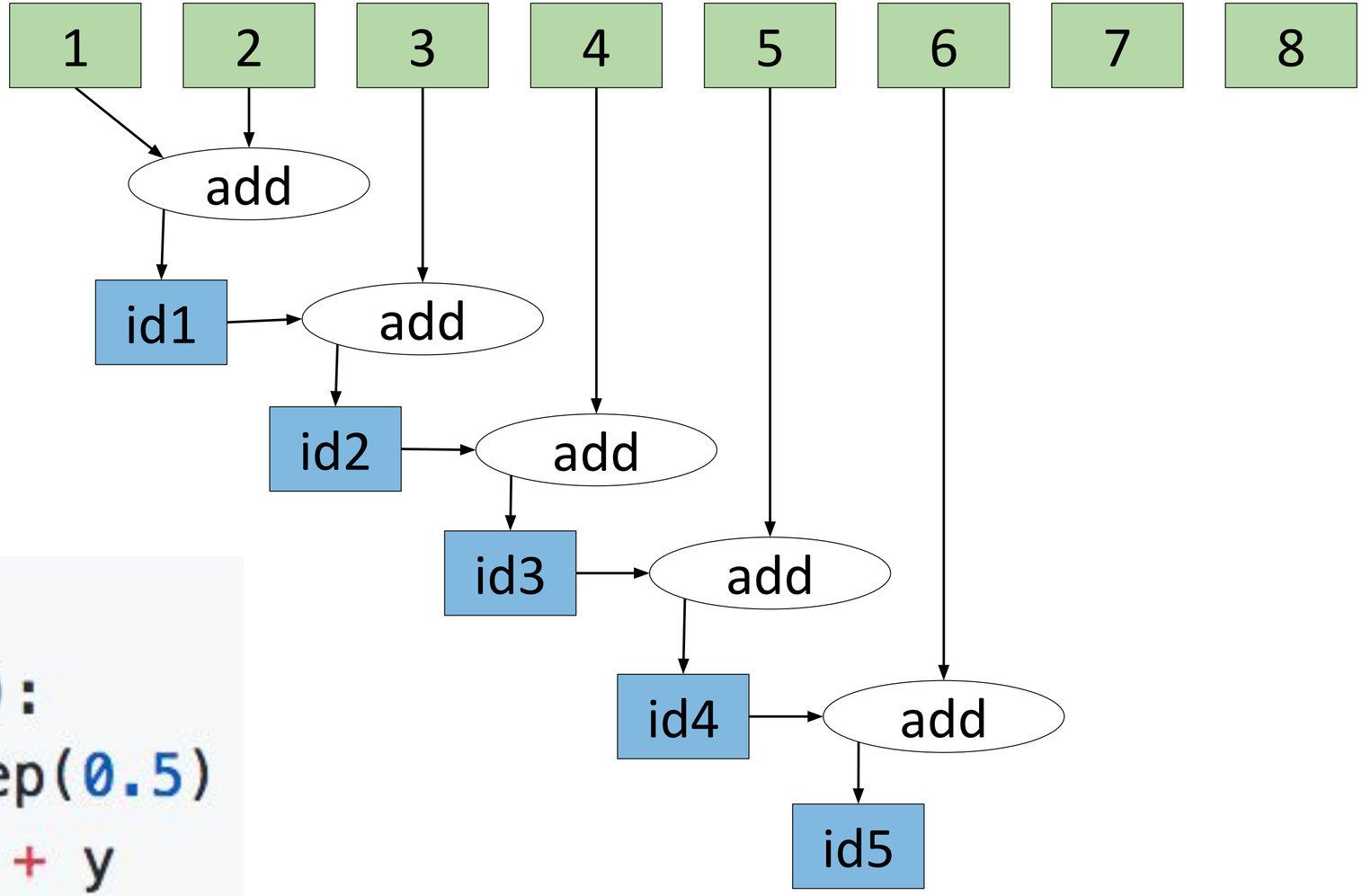
Parallelism Example: Tree Reduction




```python
@ray.remote
def add(x, y):
    time.sleep(0.5)
    return x + y
```

Add a bunch of values (2 at a time).

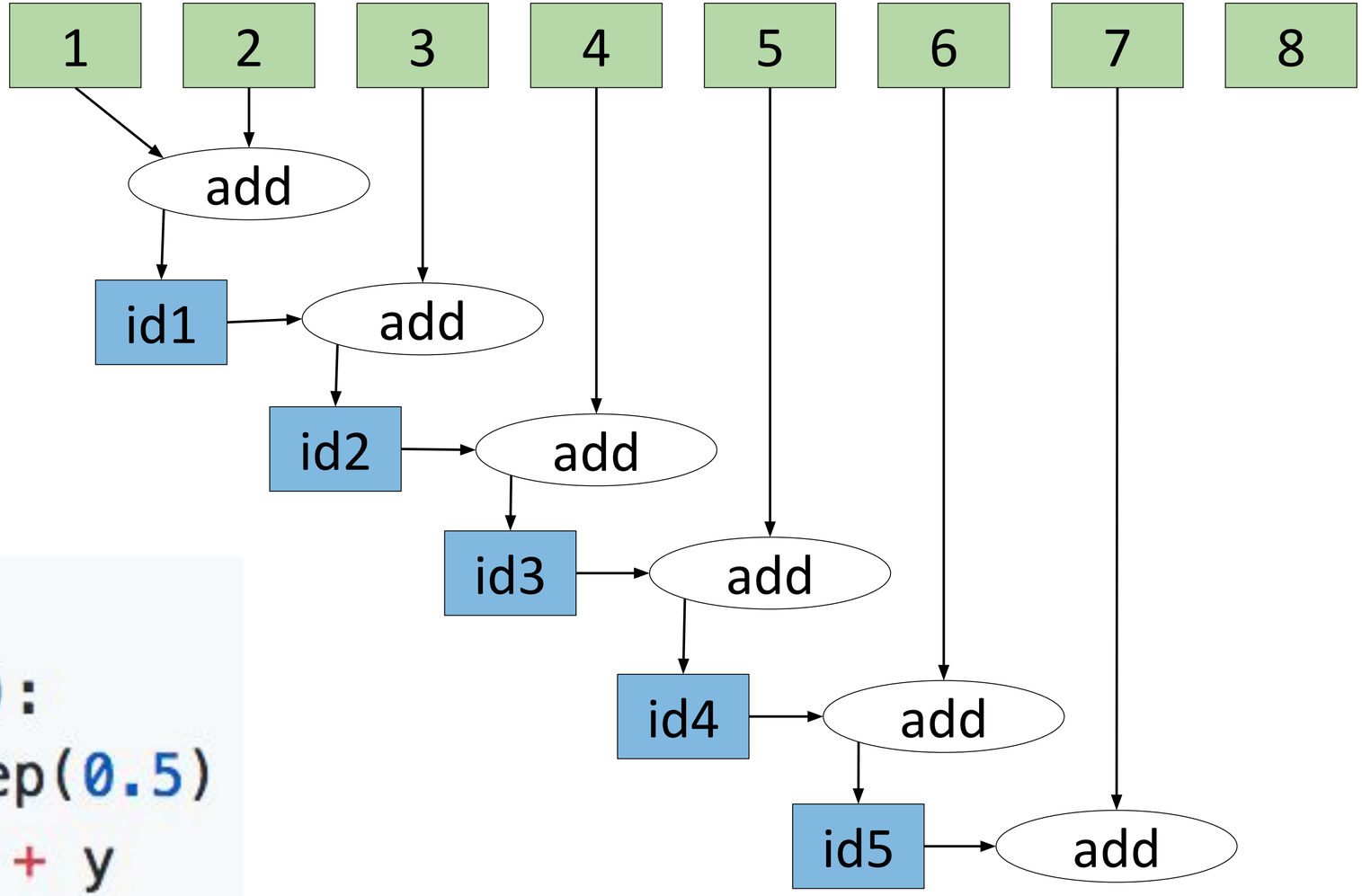
Parallelism Example: Tree Reduction




```python
@ray.remote
def add(x, y):
    time.sleep(0.5)
    return x + y
```

Add a bunch of values (2 at a time).

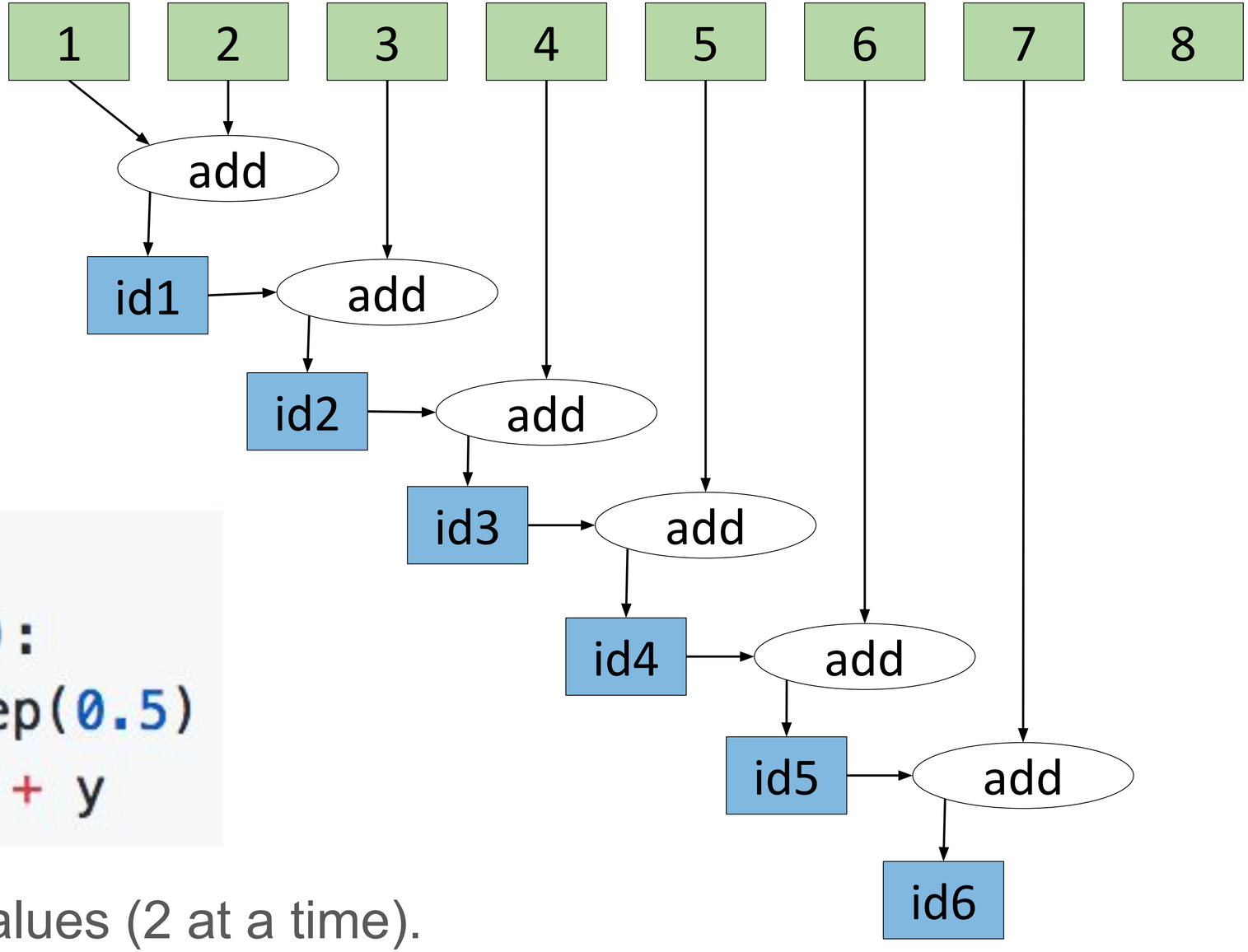
Parallelism Example: Tree Reduction




```python
@ray.remote
def add(x, y):
    time.sleep(0.5)
    return x + y
```

Add a bunch of values (2 at a time).

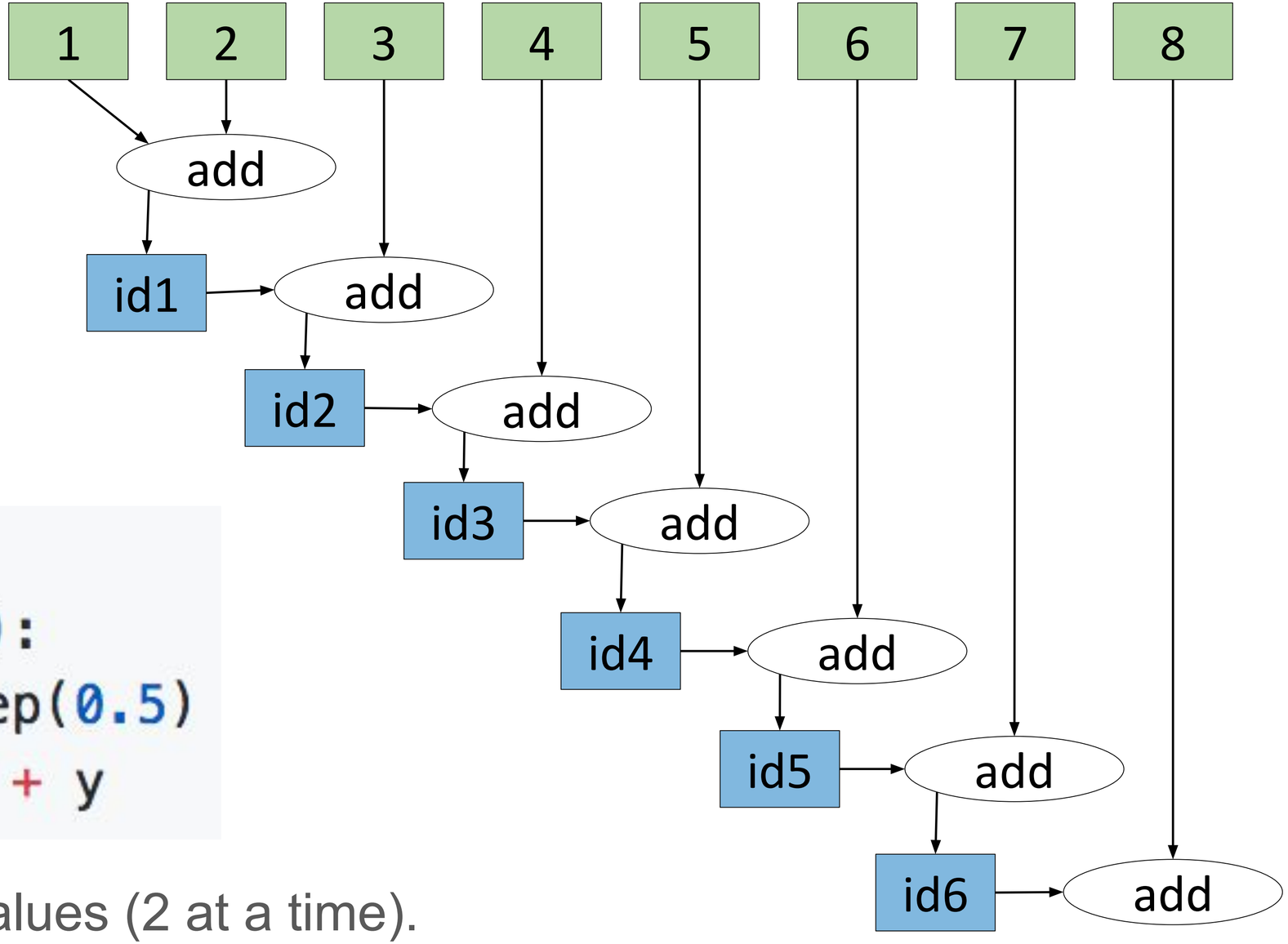
Parallelism Example: Tree Reduction




```python
@ray.remote
def add(x, y):
    time.sleep(0.5)
    return x + y
```

Add a bunch of values (2 at a time).

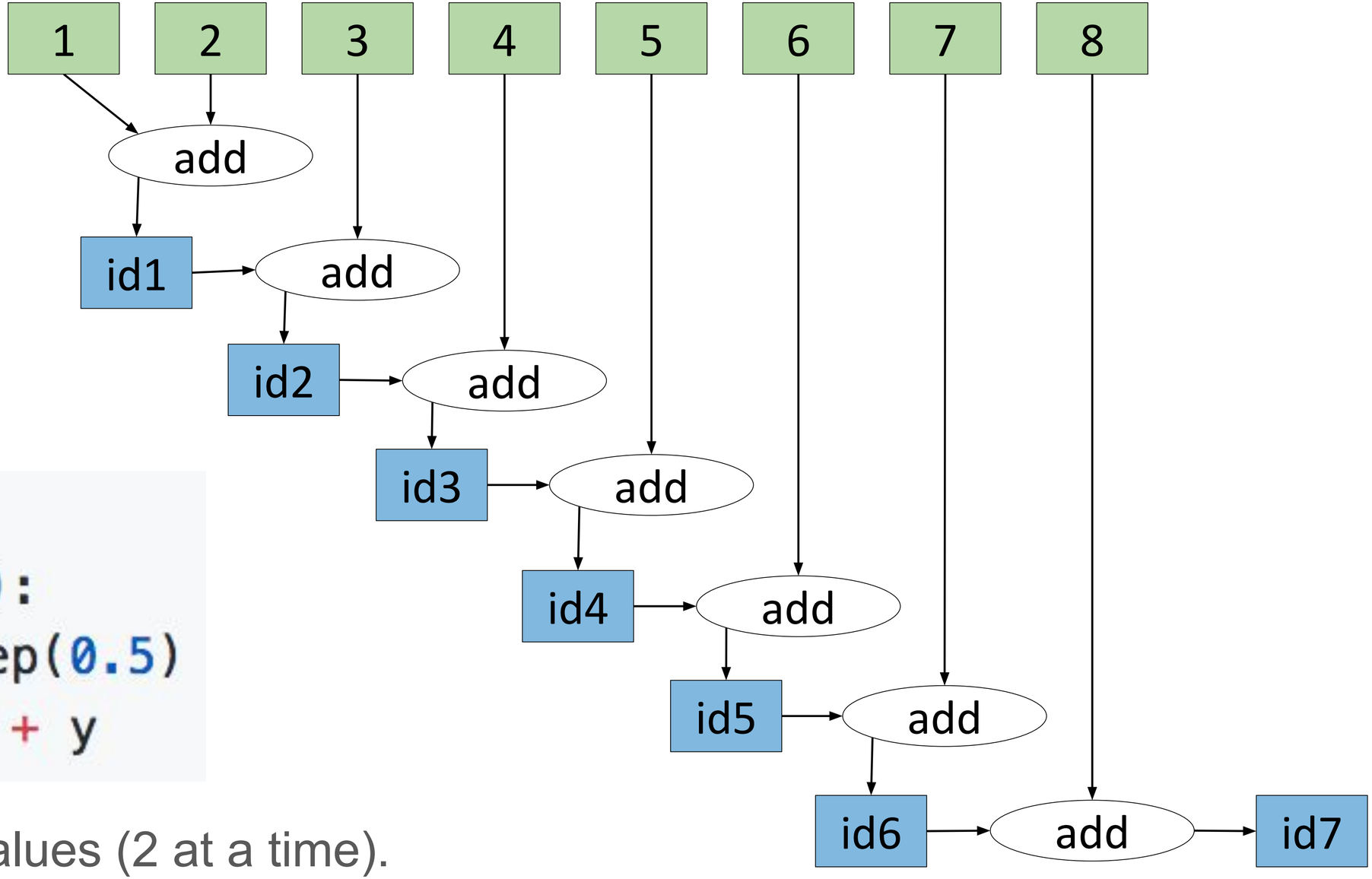
Parallelism Example: Tree Reduction




```python
@ray.remote
def add(x, y):
    time.sleep(0.5)
    return x + y
```

Add a bunch of values (2 at a time).

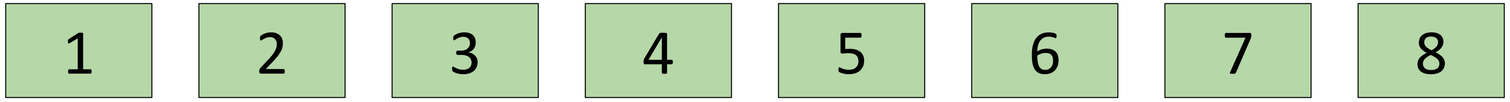
Parallelism Example: Tree Reduction




```python
@ray.remote
def add(x, y):
    time.sleep(0.5)
    return x + y
```

Add a bunch of values (2 at a time).

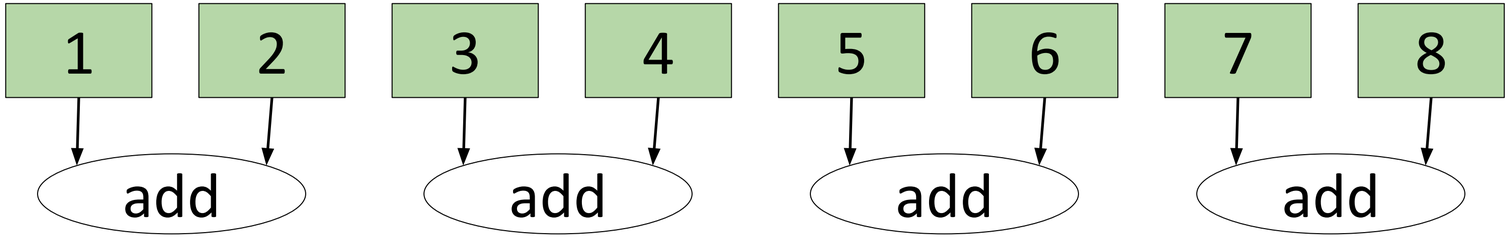
Parallelism Example: Tree Reduction




```
@ray.remote
def add(x, y):
    time.sleep(0.5)
    return x + y
```

Add a bunch of values (2 at a time).

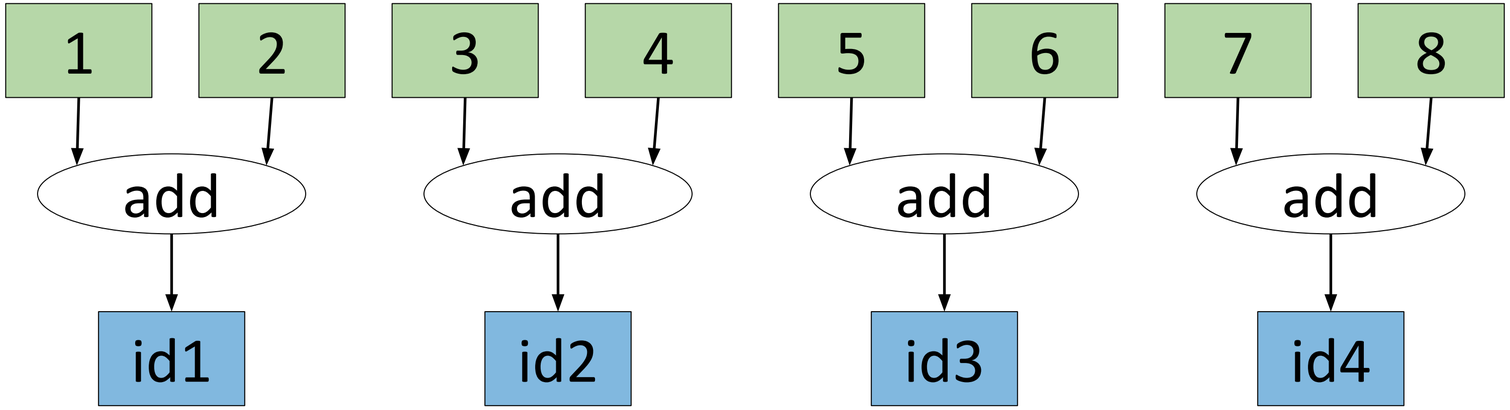
Parallelism Example: Tree Reduction




```
@ray.remote
def add(x, y):
    time.sleep(0.5)
    return x + y
```

Add a bunch of values (2 at a time).

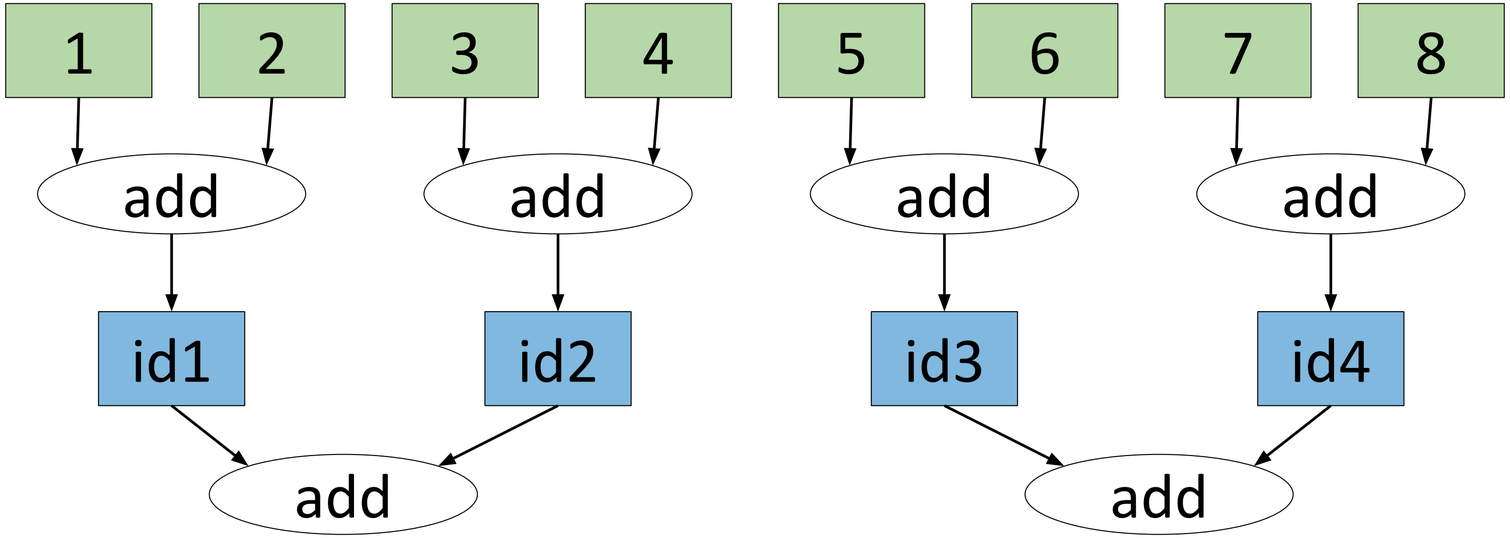
Parallelism Example: Tree Reduction




```
@ray.remote
def add(x, y):
    time.sleep(0.5)
    return x + y
```

Add a bunch of values (2 at a time).

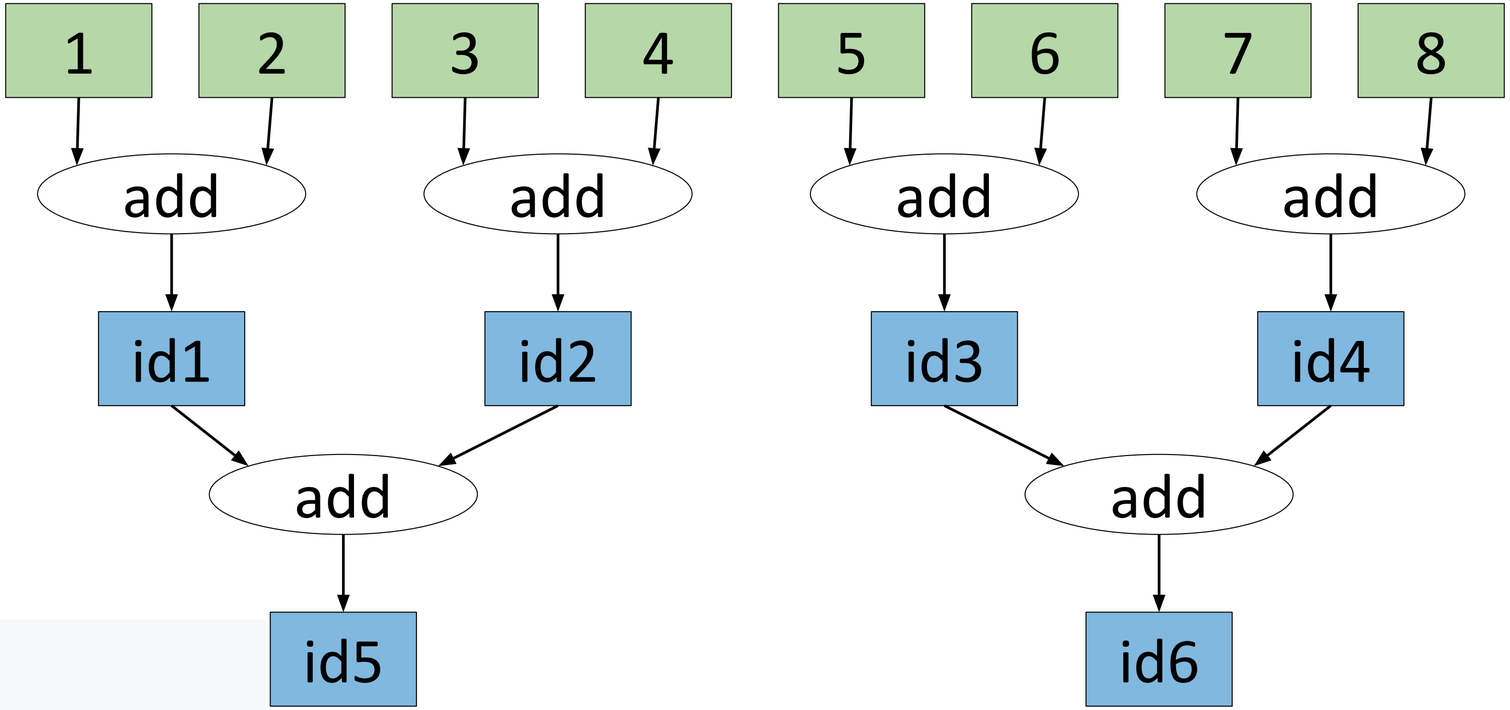
Parallelism Example: Tree Reduction




```python
@ray.remote
def add(x, y):
    time.sleep(0.5)
    return x + y
```

Add a bunch of values (2 at a time).

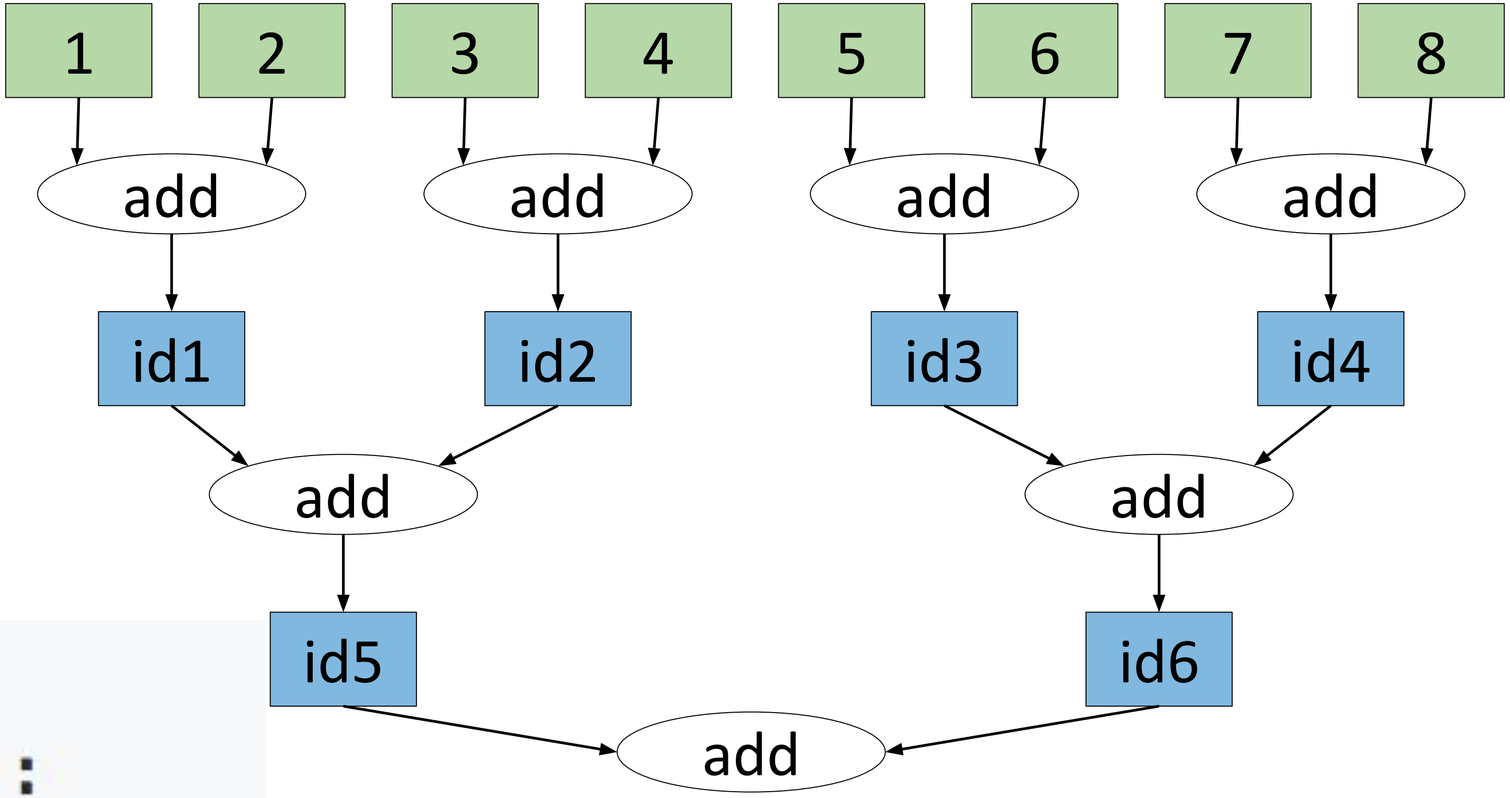
Parallelism Example: Tree Reduction




```python
@ray.remote
def add(x, y):
    time.sleep(0.5)
    return x + y
```

Add a bunch of values (2 at a time).

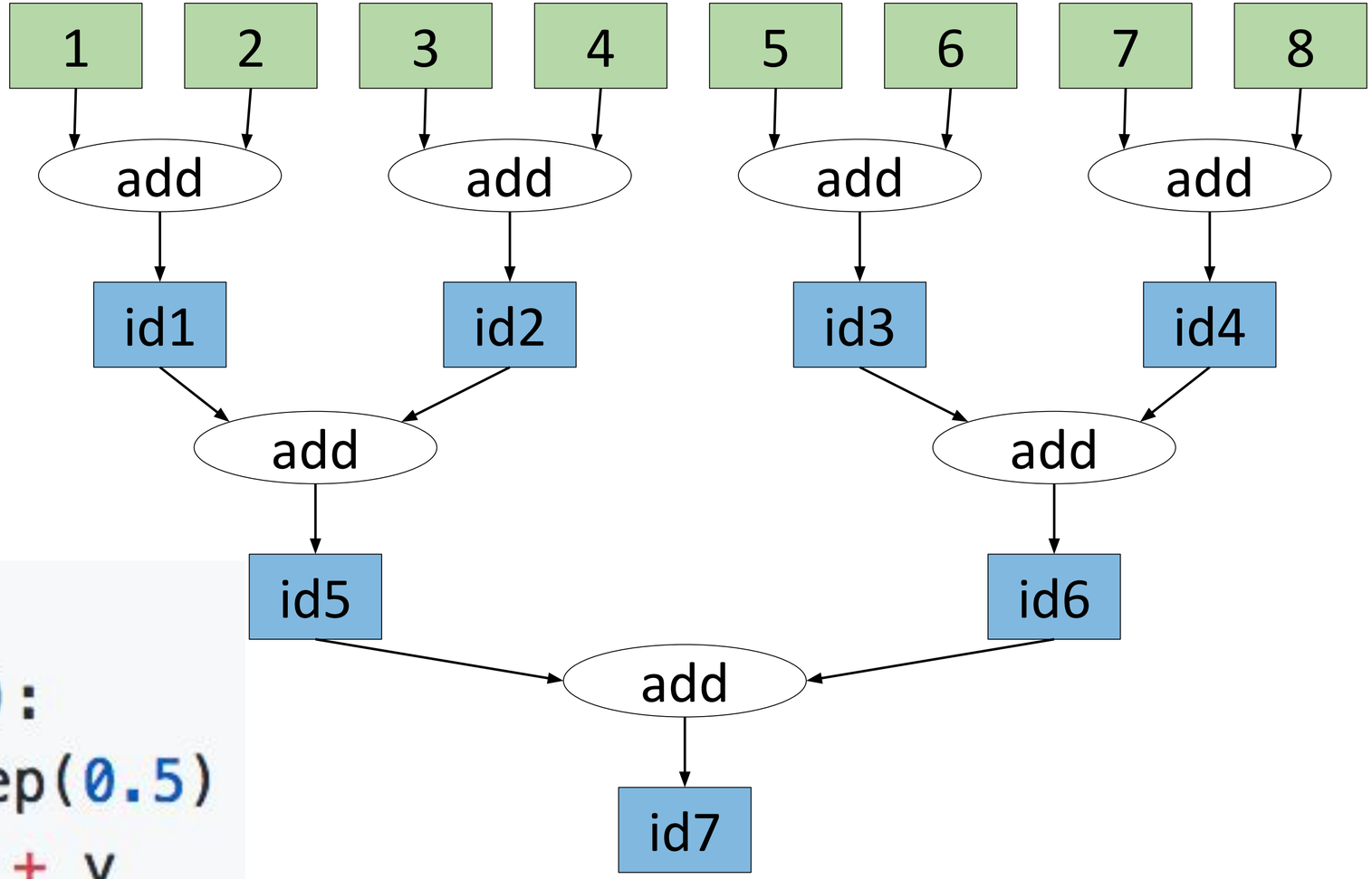
Parallelism Example: Tree Reduction




```python
@ray.remote
def add(x, y):
    time.sleep(0.5)
    return x + y
```

Add a bunch of values (2 at a time).

Parallelism Example: Tree Reduction




```python
@ray.remote
def add(x, y):
    time.sleep(0.5)
    return x + y
```

Add a bunch of values (2 at a time).

Parallelism Example: Tree Reduction

```
@ray.remote
def add(x, y):
    time.sleep(0.5)
    return x + y
```

Add a bunch of values (2 at a time).

Parallelism Example: Tree Reduction

What does the difference look like in code?

```
@ray.remote
def add(x, y):
    time.sleep(0.5)
    return x + y
```

Add a bunch of values (2 at a time).

Parallelism Example: Tree Reduction

What does the difference look like in code?

```
@ray.remote
def add(x, y):
    time.sleep(0.5)
    return x + y
```

```
vals = [1, 2, 3, 4, 5, 6, 7, 8]
while len(vals) > 1:
    new_val = add.remote(vals[0], vals[1])
    vals = [new_val] + vals[2:]
```

```
vals = [1, 2, 3, 4, 5, 6, 7, 8]
while len(vals) > 1:
    new_val = add.remote(vals[0], vals[1])
    vals = vals[2:] + [new_val]
```

Add a bunch of values (2 at a time).

Parallelism Example: Tree Reduction

What does the difference look like in code?

```
@ray.remote
def add(x, y):
    time.sleep(0.5)
    return x + y
```

```
vals = [1, 2, 3, 4, 5, 6, 7, 8]
while len(vals) > 1:
    new_val = add.remote(vals[0], vals[1])
    vals = [new_val] + vals[2:]
```

```
vals = [1, 2, 3, 4, 5, 6, 7, 8]
while len(vals) > 1:
    new_val = add.remote(vals[0], vals[1])
    vals = vals[2:] + [new_val]
```

Add a bunch of values (2 at a time).

Actors: Parameter Server Example

```
@ray.remote
class ParameterServer(object):
    def __init__(self):
        self.params = np.zeros(10)
    def get_params(self):
        return self.params
    def update_params(self, grad):
        self.params -= grad
```

Actors: Parameter Server Example

```
@ray.remote
class ParameterServer(object):
    def __init__(self):
        self.params = np.zeros(10)
    def get_params(self):
        return self.params
    def update_params(self, grad):
        self.params -= grad
```

```
@ray.remote(num_gpus=1)
def worker(ps):
    while True:
        params = ray.get(ps.get_params.remote())
        grad = ...  # Use TensorFlow
        ps.update_params.remote(grad)
```

Actors: Parameter Server Example

```
@ray.remote
class ParameterServer(object):
    def __init__(self):
        self.params = np.zeros(10)
    def get_params(self):
        return self.params
    def update_params(self, grad):
        self.params -= grad
```

```
@ray.remote(num_gpus=1)
def worker(ps):
    while True:
        ...
```

```
@ray.remote(num_gpus=1)
def worker(ps):
    while True:
        ...
```

```
@ray.remote(num_gpus=1)
def worker(ps):
    while True:
        params = ray.get(ps.get_params.remote())
        grad = ...  # Use TensorFlow
        ps.update_params.remote(grad)
```

Actors: Parameter Server Example

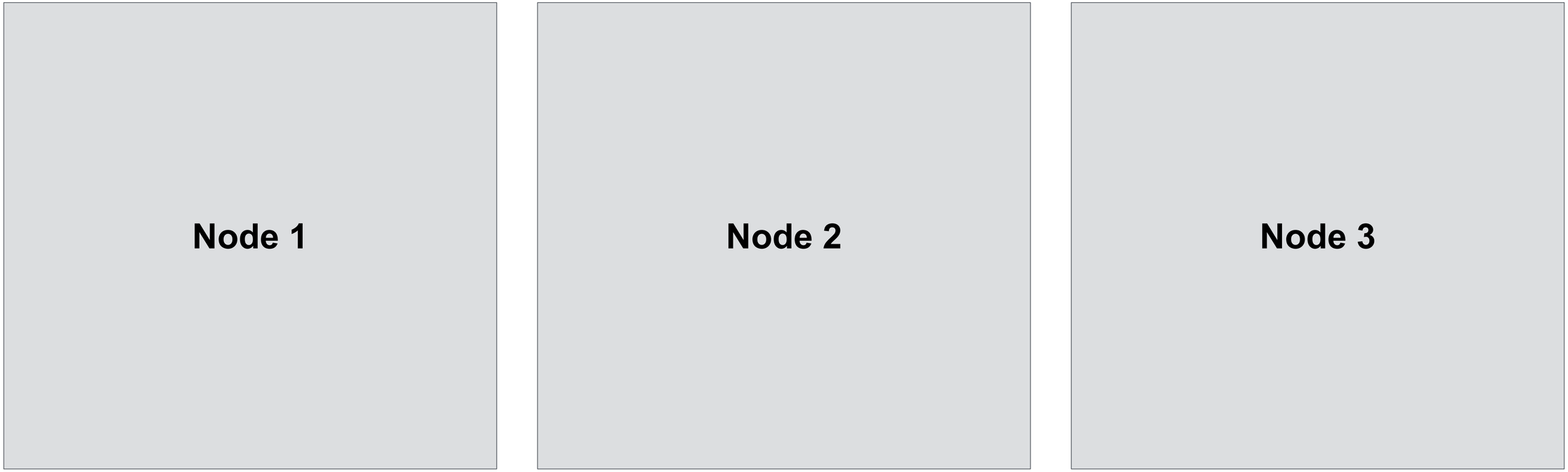
```
@ray.remote
class ParameterServer(object):
    def __init__(self):
        self.params = np.zeros(10)
    def get_params(self):
        return self.params
    def update_params(self, grad):
        self.params -= grad
```

```
@ray.remote(num_gpus=1)
def worker(ps):
    while True:
        ...
```

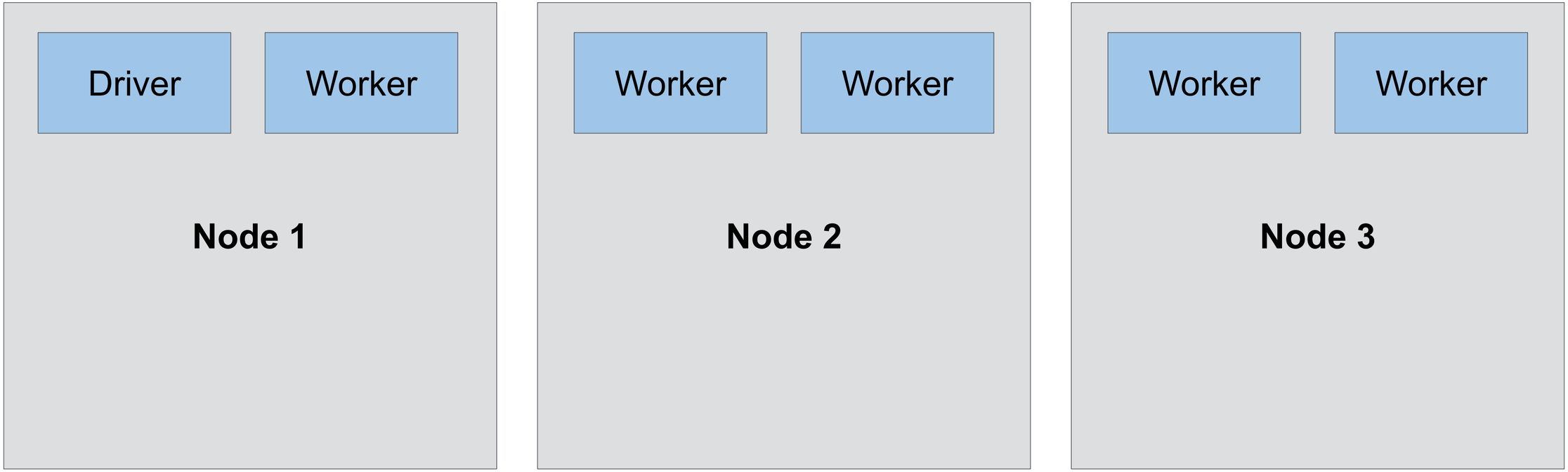
```
@ray.remote(num_gpus=1)
def worker(ps):
    while True:
        ...
```

```
@ray.remote(num_gpus=1)
def worker(ps):
    while True:
        params = ray.get(ps.get_params.remote())
        grad = ...  # Use TensorFlow
        ps.update_params.remote(grad)
```

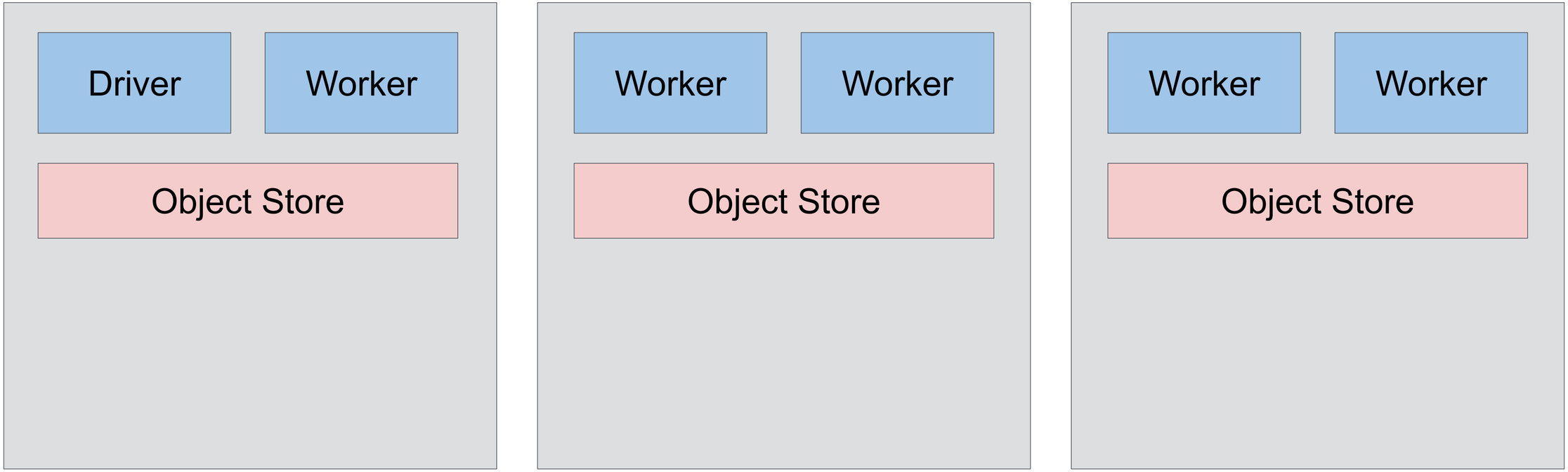
Ray Architecture



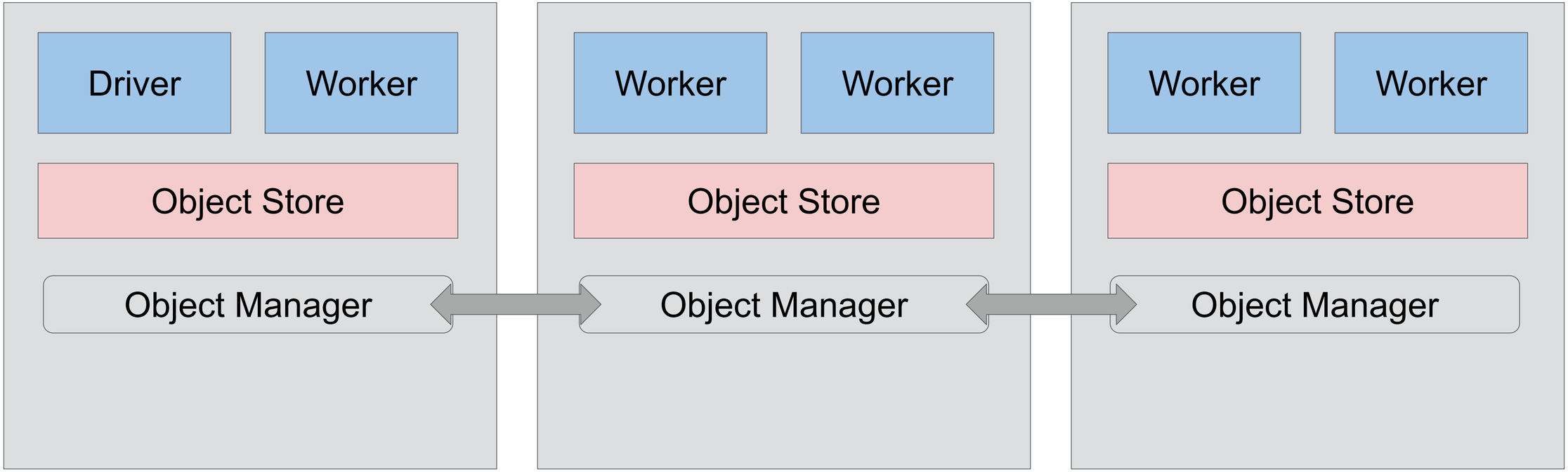
Ray Architecture



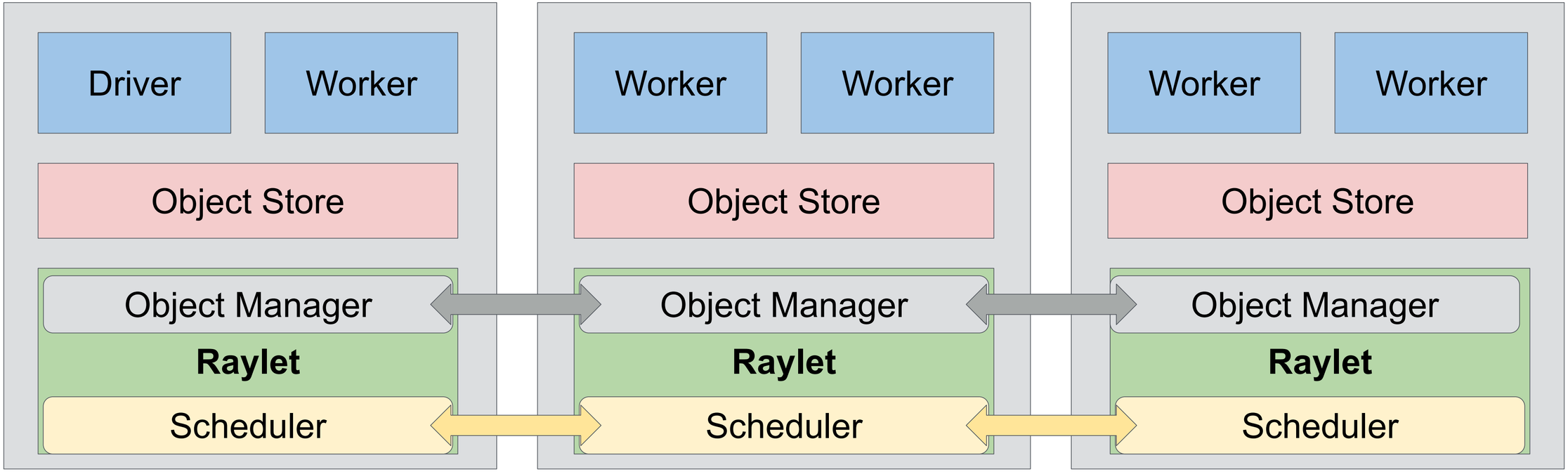
Ray Architecture



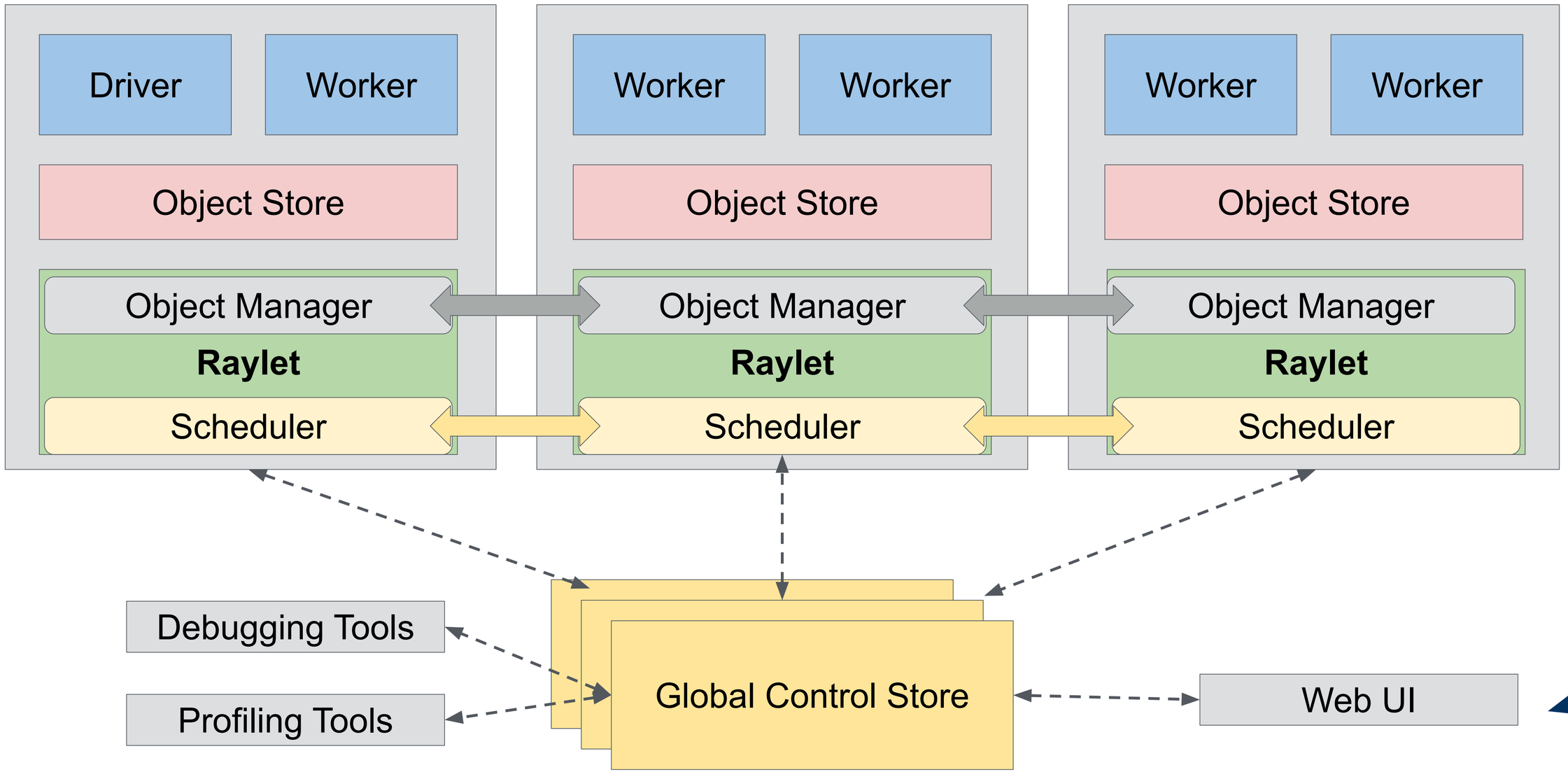
Ray Architecture



Ray Architecture



Ray Architecture

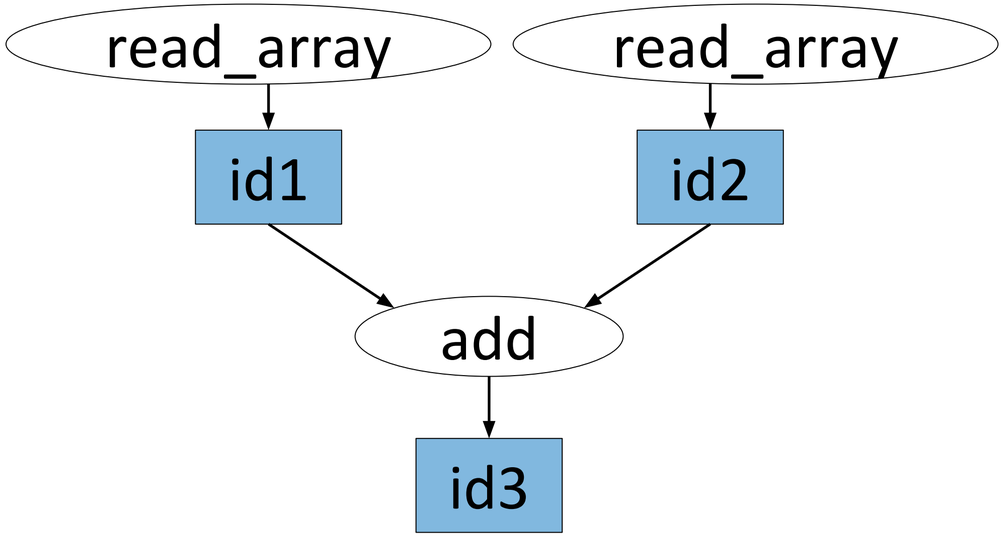


How does this work under the hood?

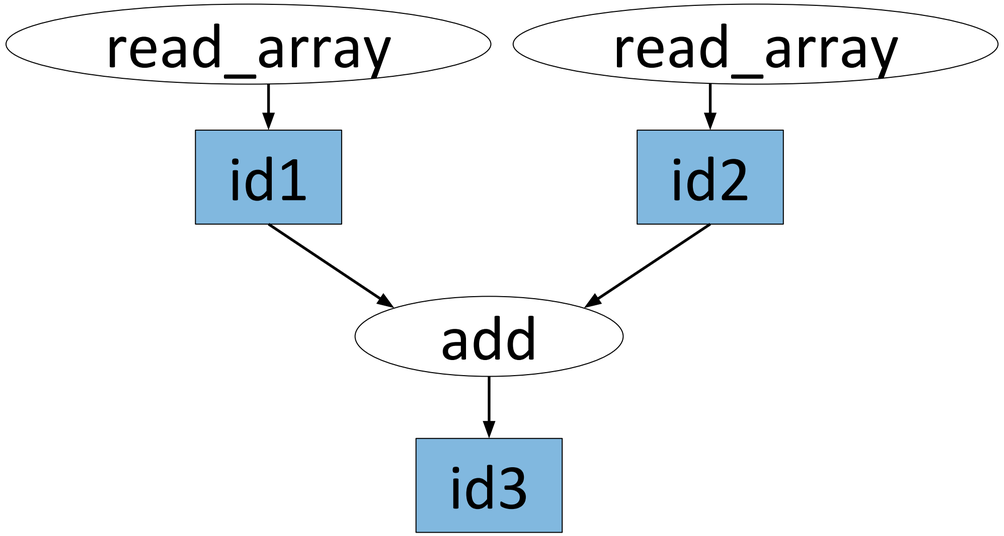
How does this work under the hood?

Tasks

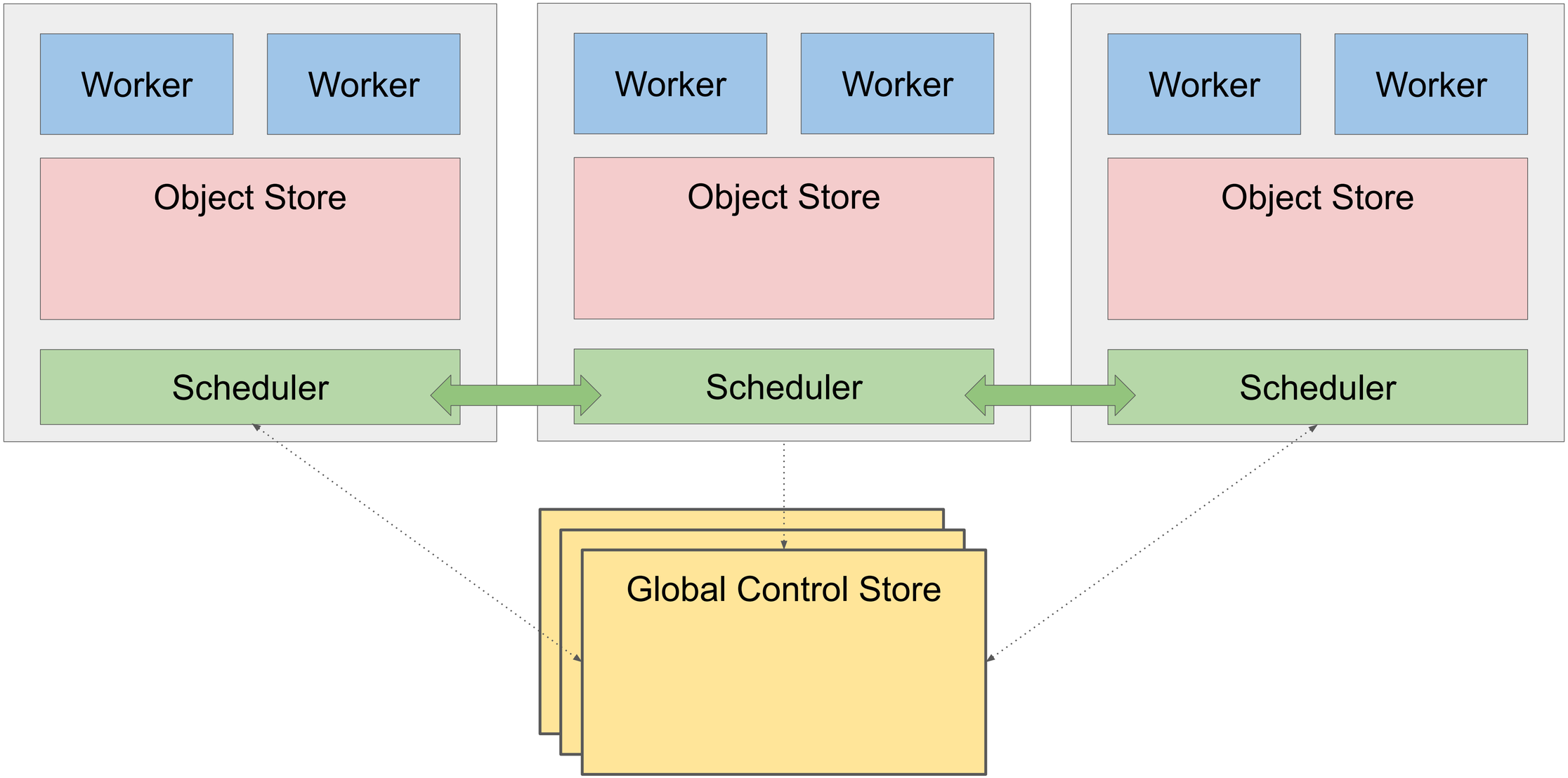
```
@ray.remote
def read_array(file):
    # read array “a” from “file”
    return a

@ray.remote
def add(a, b):
    return np.add(a, b)

id1 = read_array.remote([5, 5])
id2 = read_array.remote([5, 5])
id3 = add.remote(id1, id2)
ray.get(id3)
```



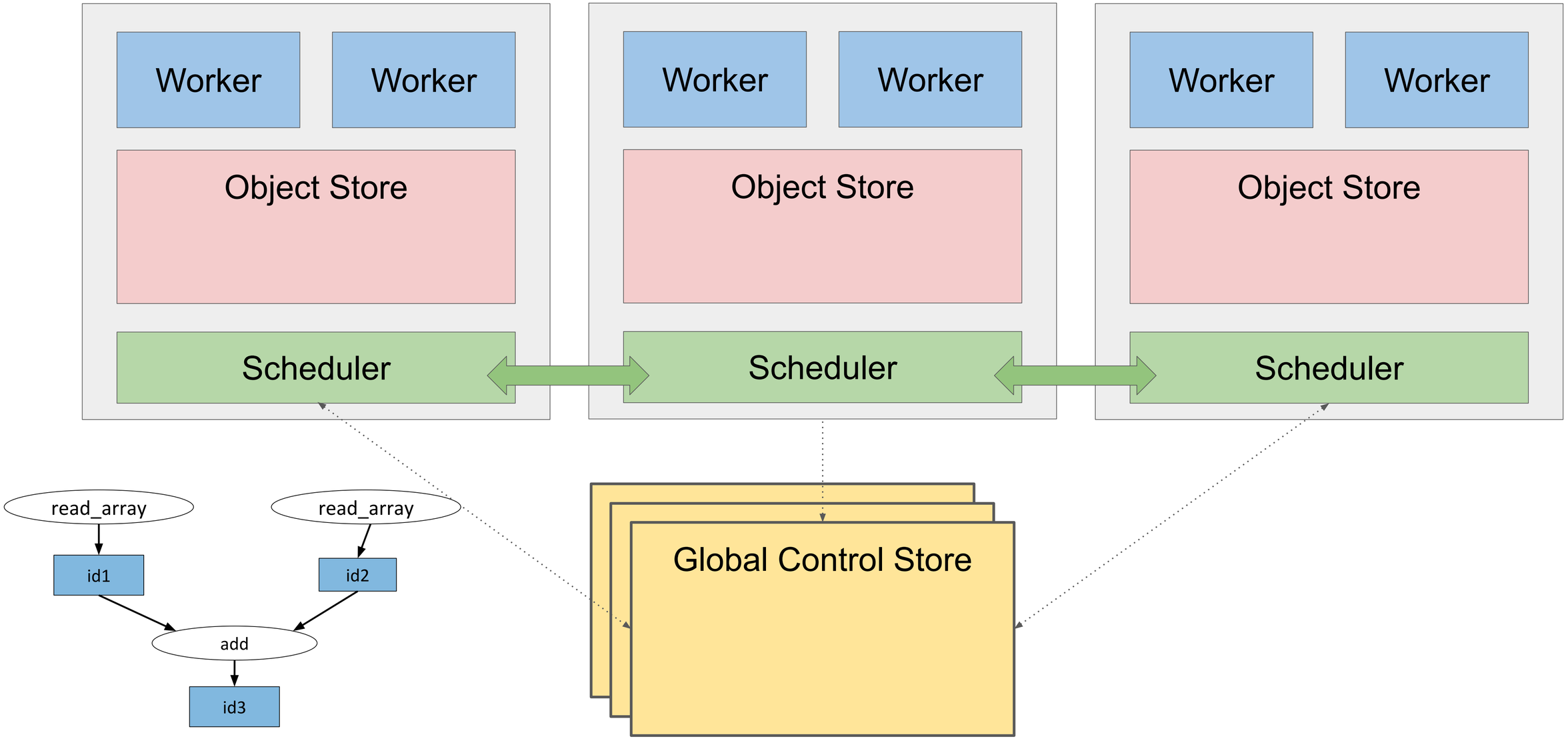
How does this work under the hood?



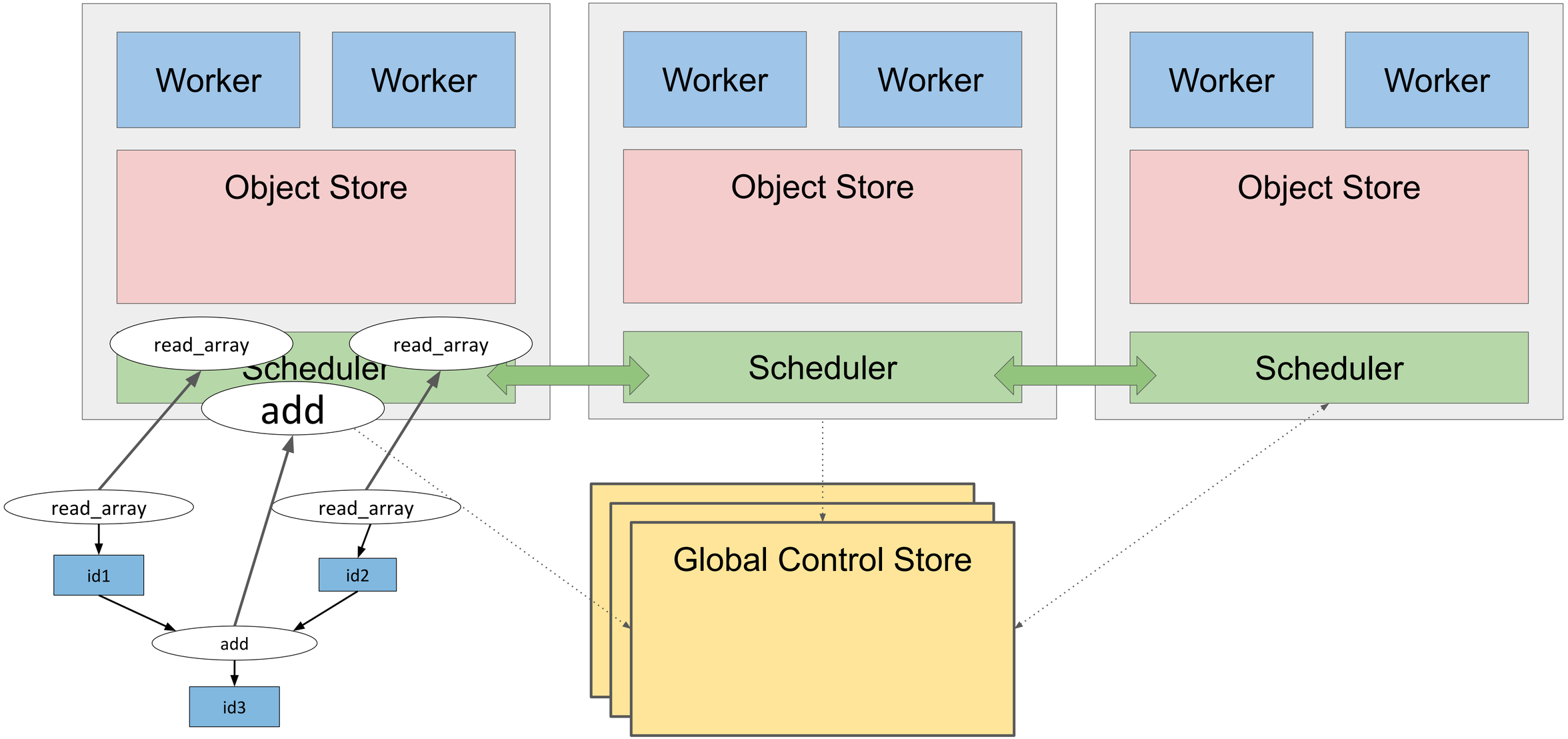
The Ray Architecture



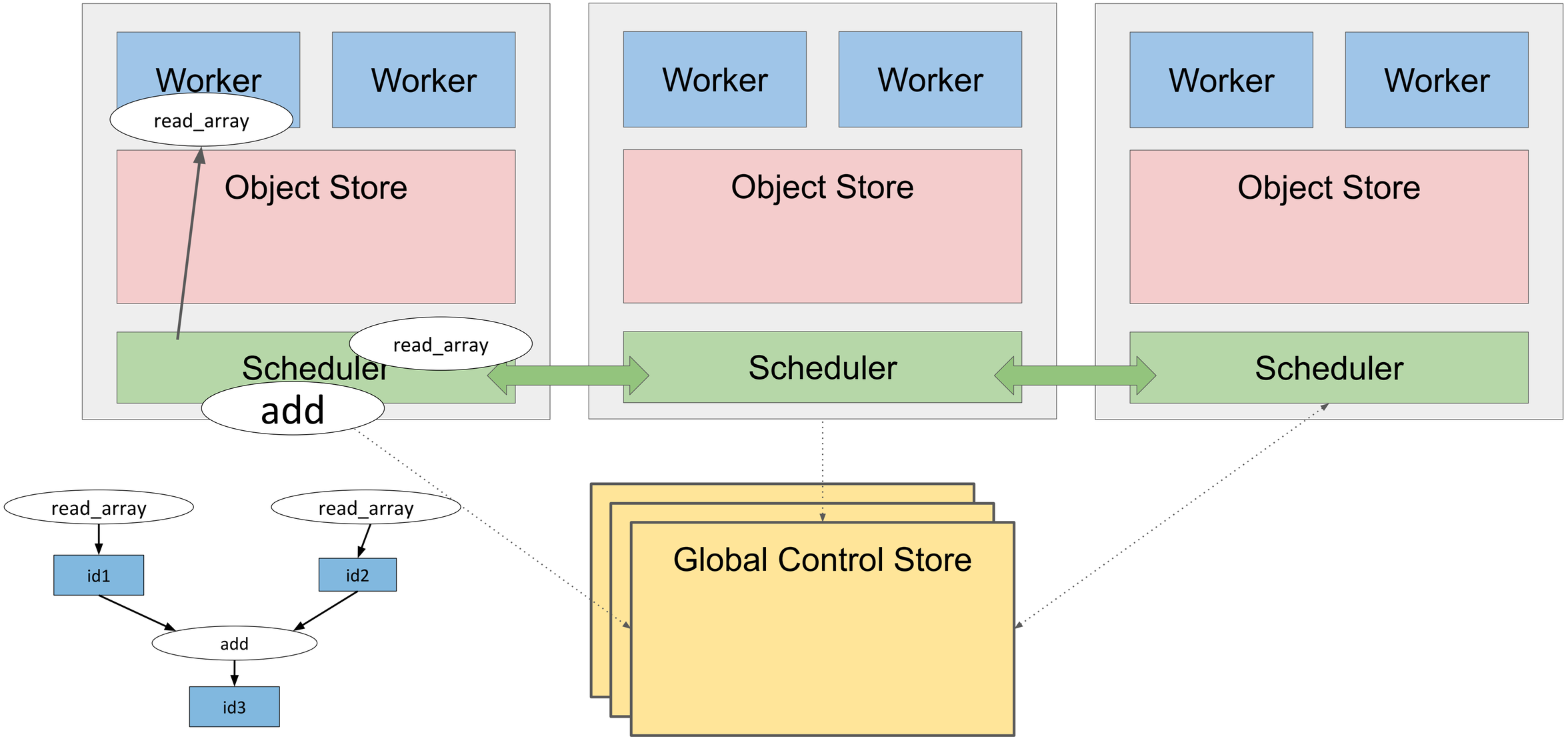
The Ray Architecture



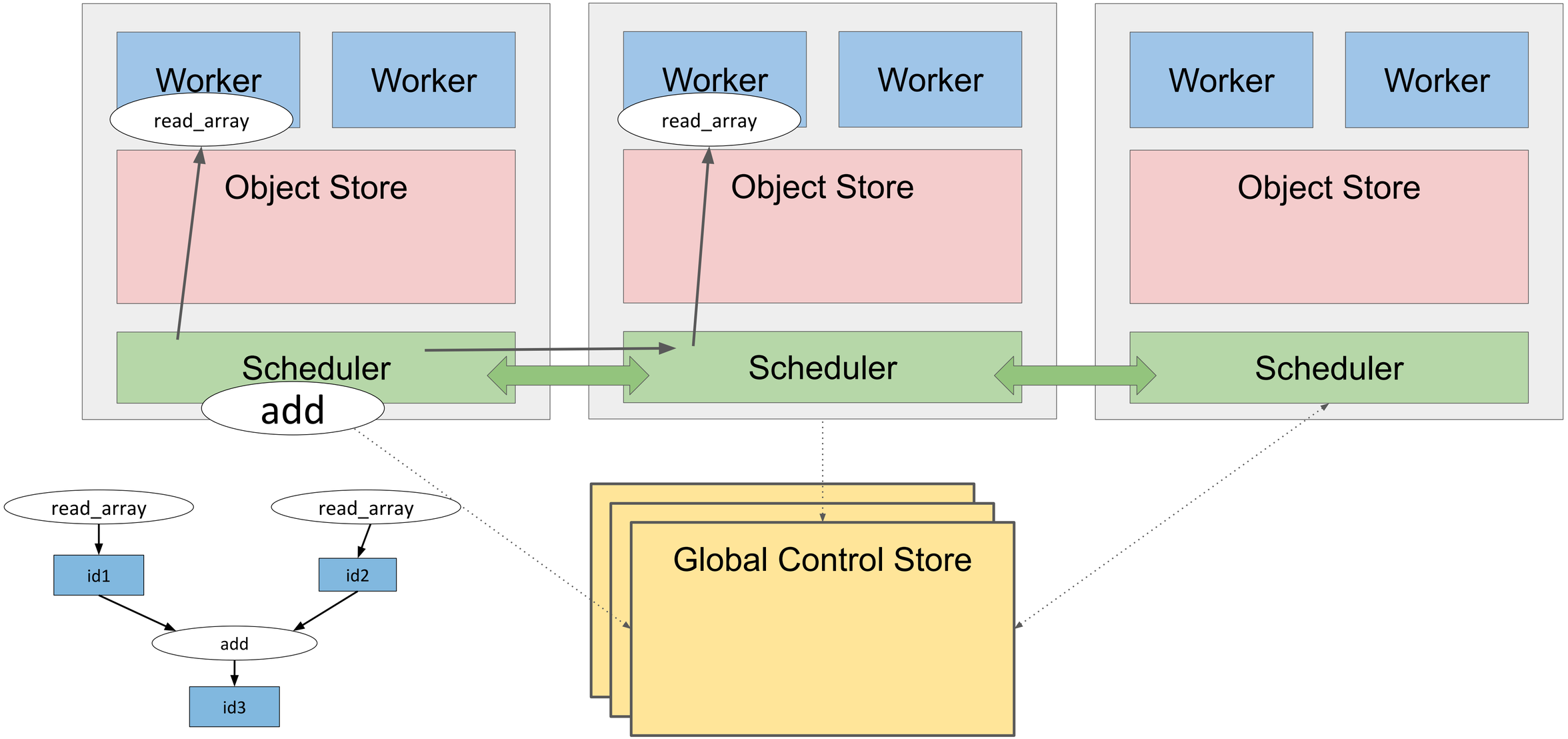
The Ray Architecture



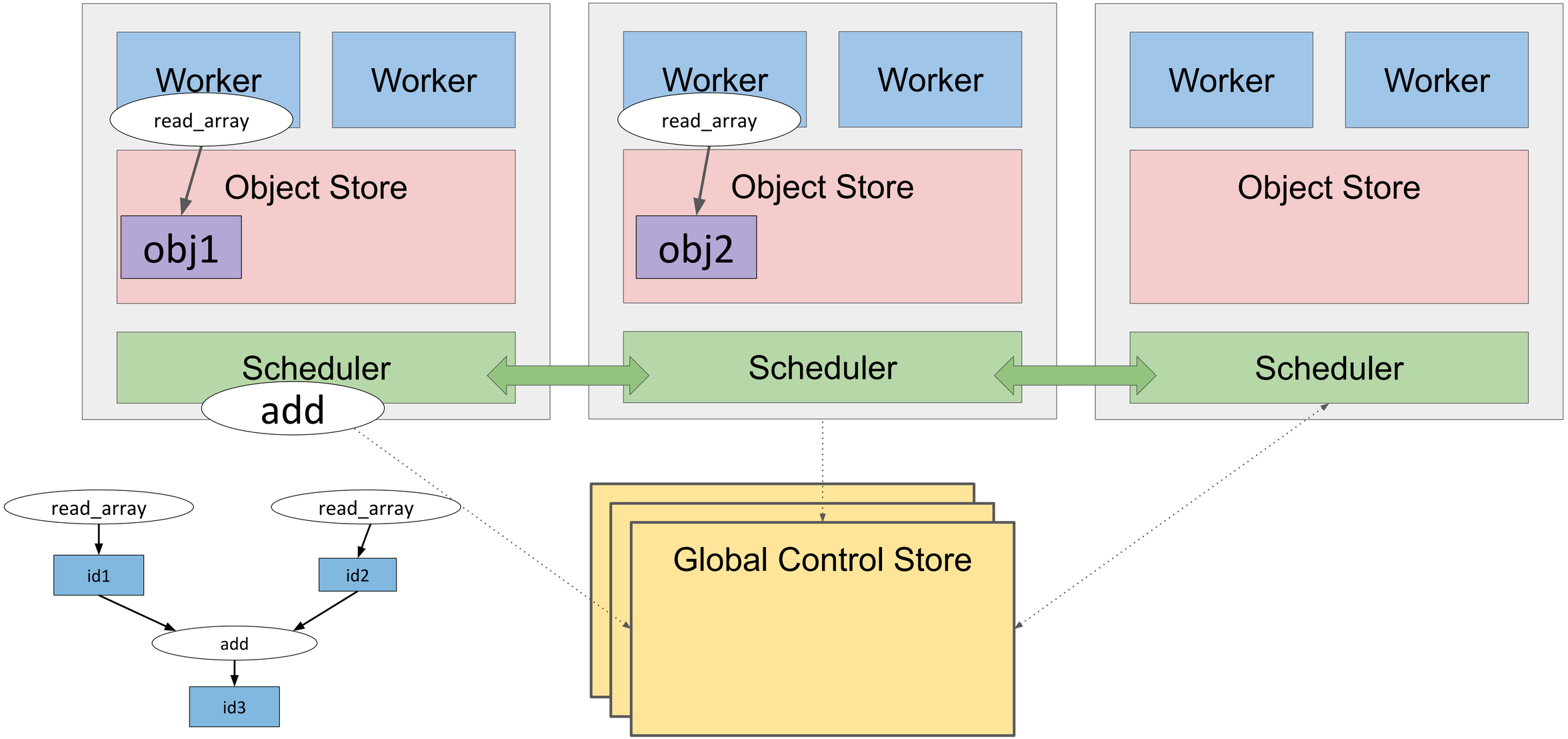
The Ray Architecture



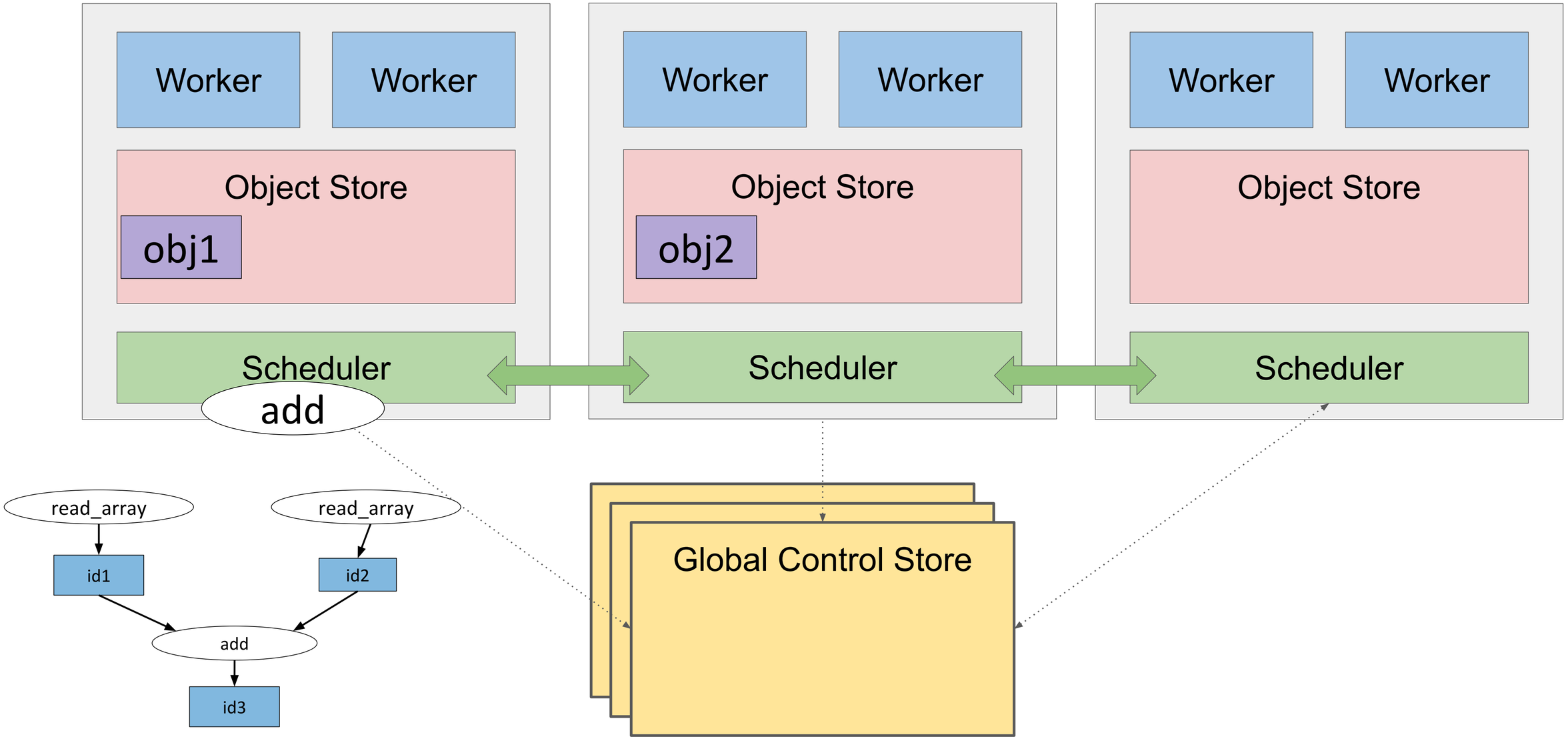
The Ray Architecture



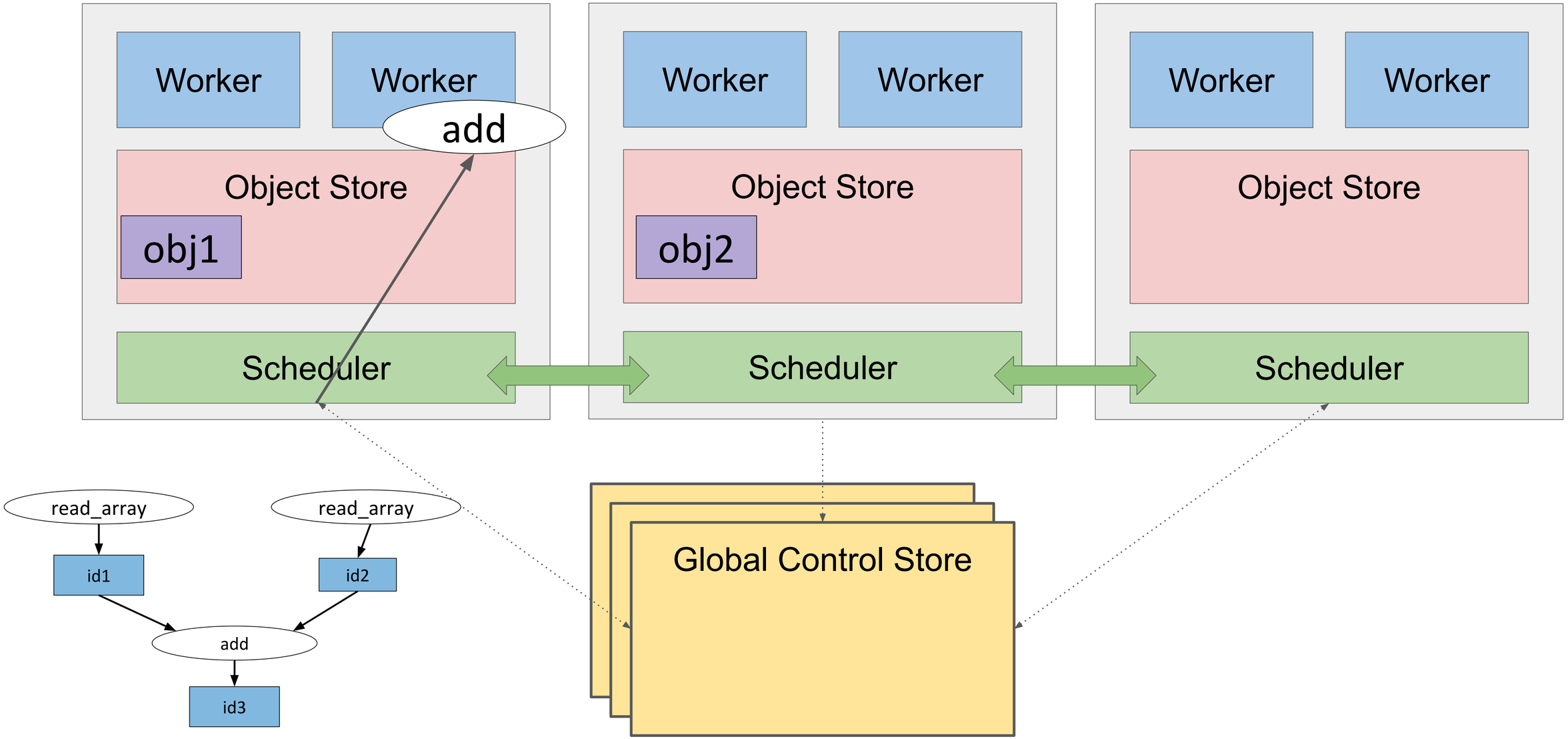
The Ray Architecture



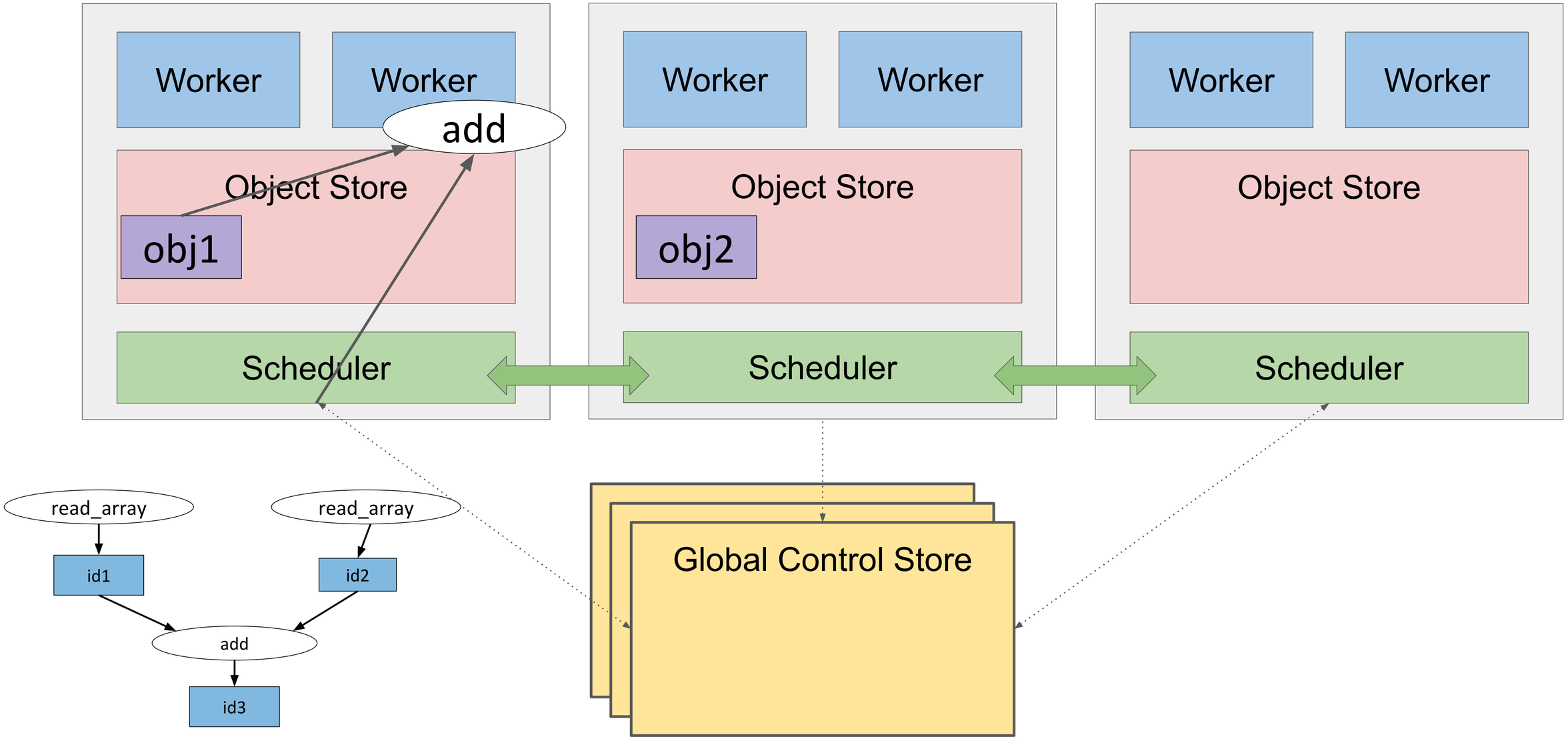
The Ray Architecture



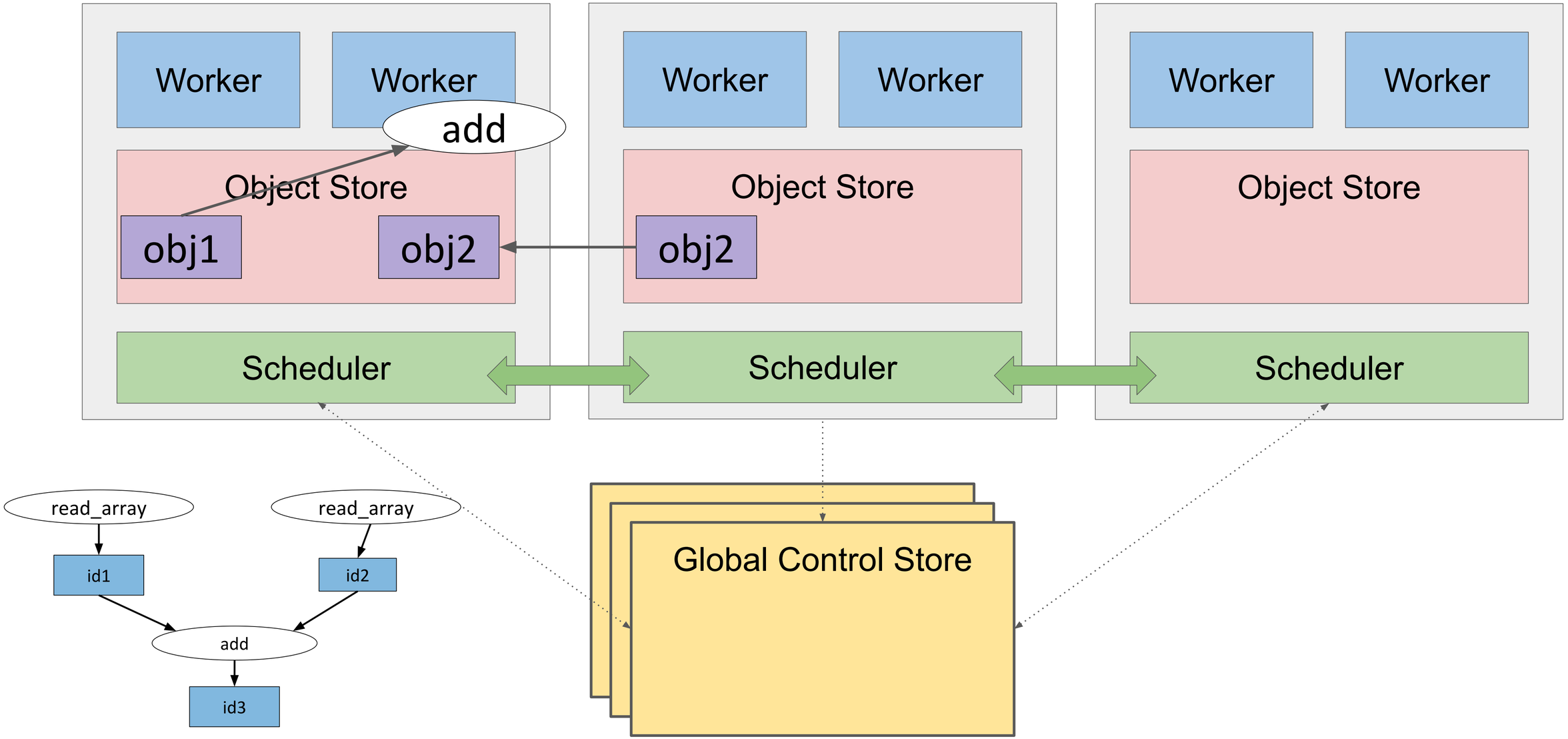
The Ray Architecture



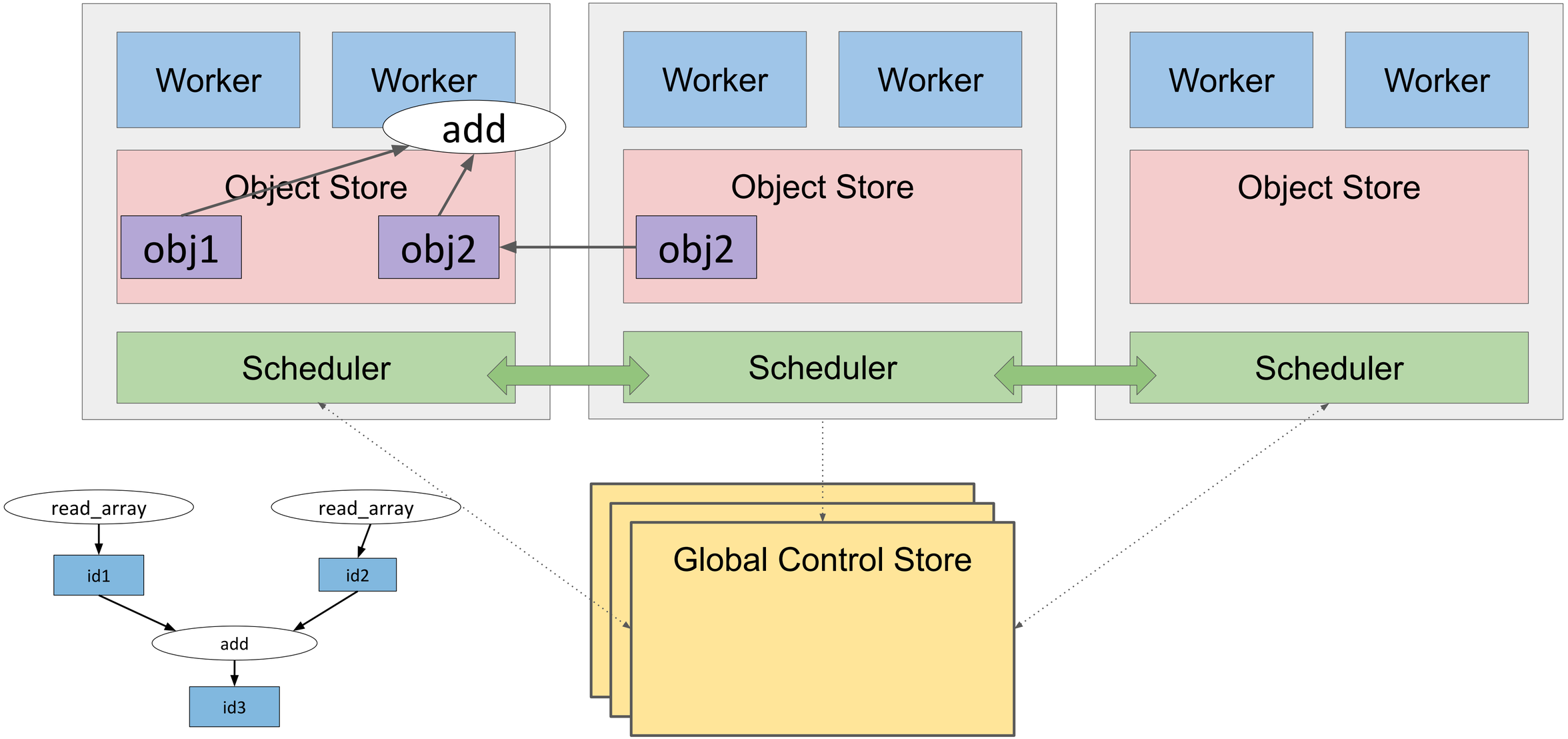
The Ray Architecture



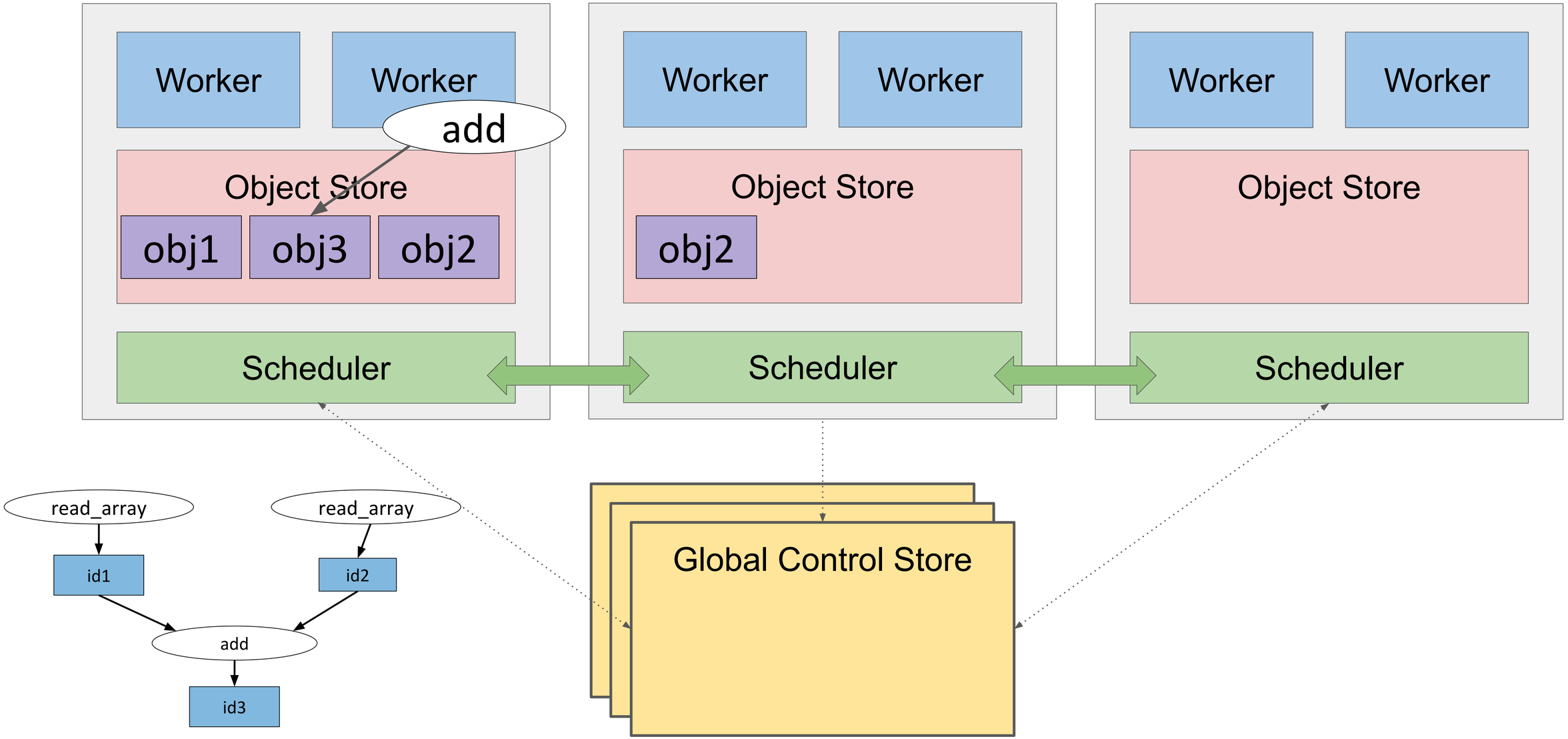
The Ray Architecture



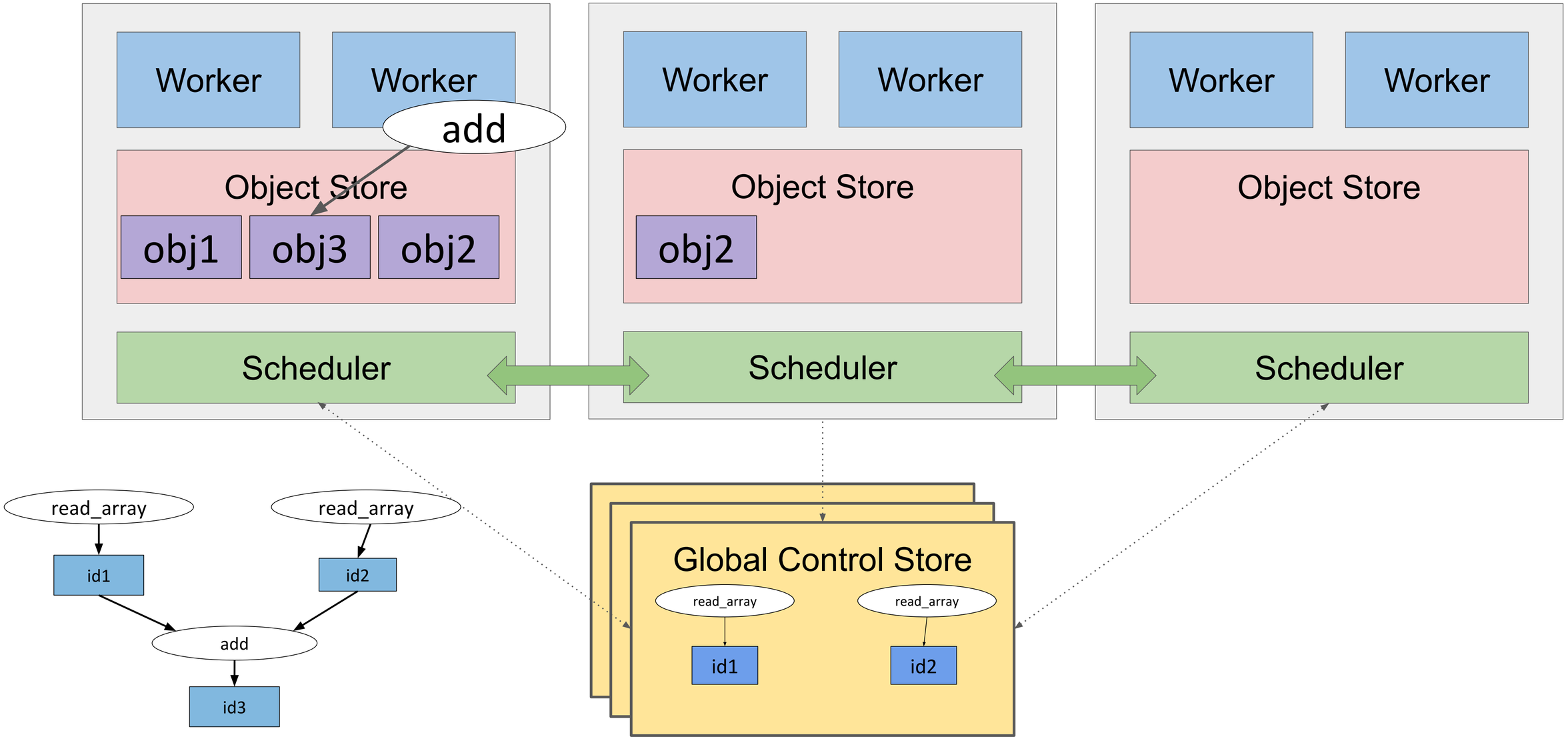
The Ray Architecture



The Ray Architecture



The Ray Architecture



Conclusion

- Ray is an open source project for distributed computing
- **special-purpose** distributed systems -> **general-purpose** distributed system
- Support for the full ML lifecycle (data collection, training, simulation, serving)

Conclusion

- Ray is an open source project for distributed computing
- **special-purpose** distributed systems -> **general-purpose** distributed system
- Support for the full ML lifecycle (data collection, training, simulation, serving)