SJTU SAIL (Software Architecture and Infrastructure Lab)

SHANGHAI JIAO TONG UNIVERSITY

Machine Learning Frameworks

Yalun Lin Hsu

2020-10-18

SAIL

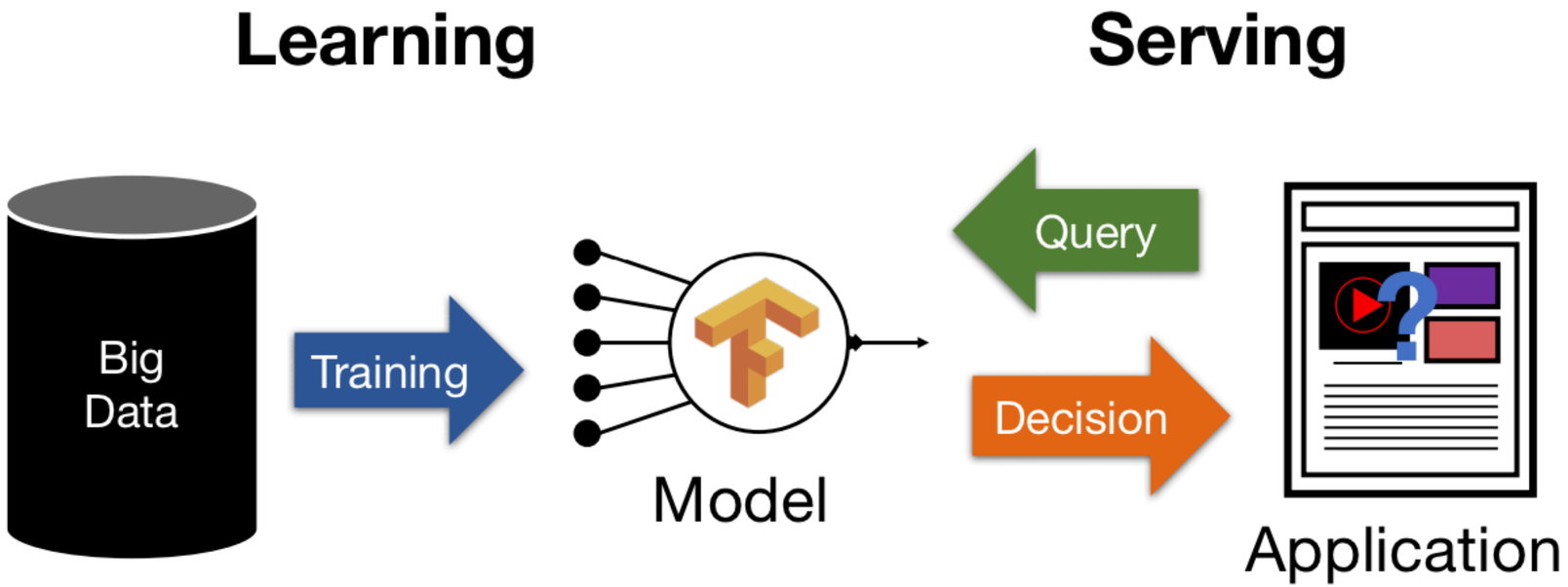
NSDI 2017

Clipper

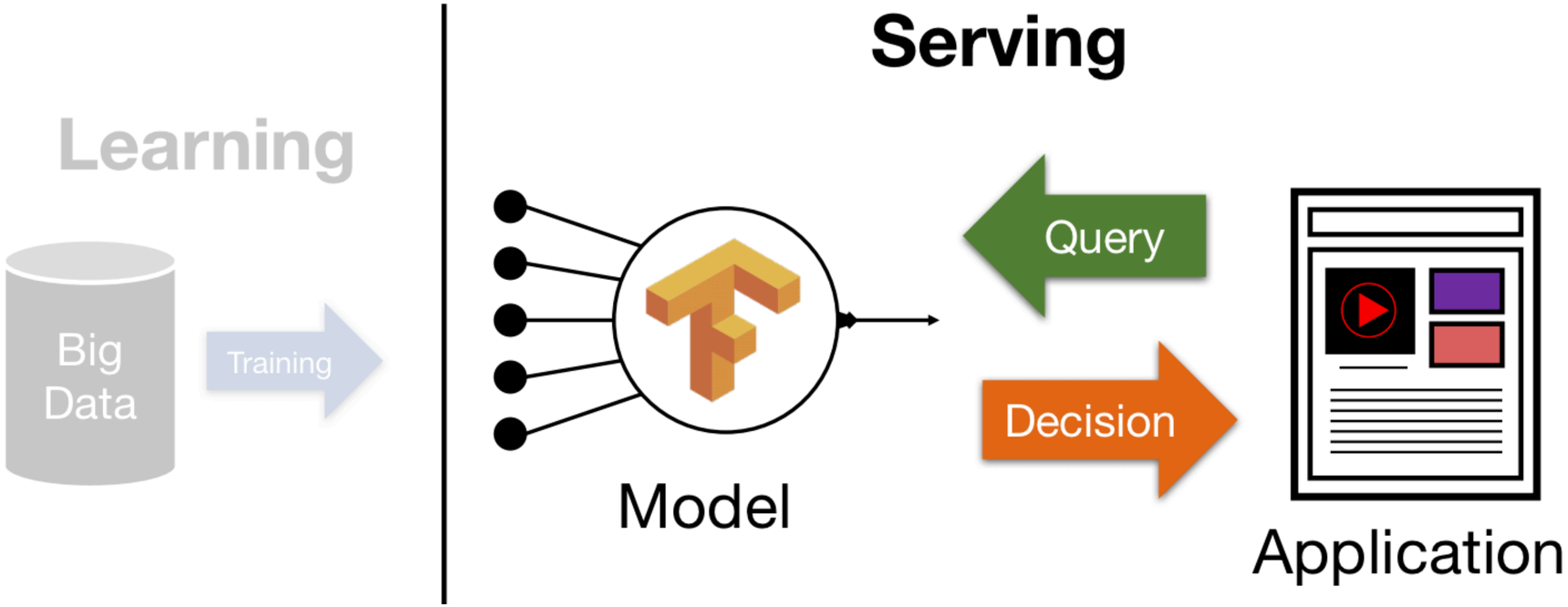
A Low-Latency Online Prediction Serving System

https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/crankshaw

Learning Produces a Trained Model



Serving Stage



Prediction-Serving for interactive applications

Timescale: ~10s of milliseconds

Prediction-Serving Raises New Challenges

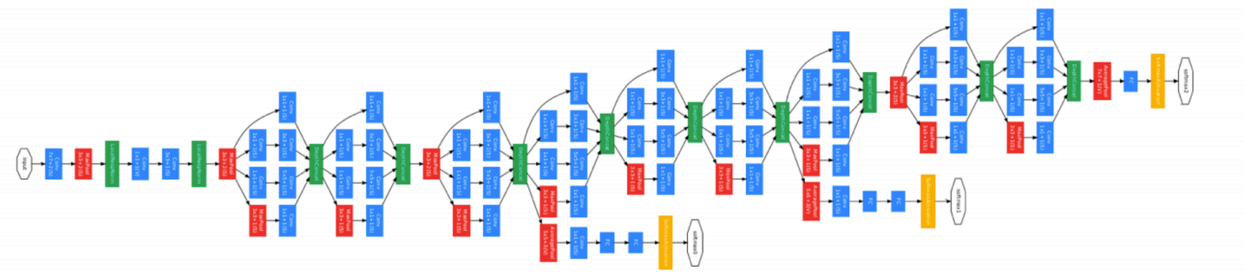
Prediction-Serving Challenge

- Support low-latency, high throughput serving workloads

- Large and growing ecosystem of ML models and frameworks

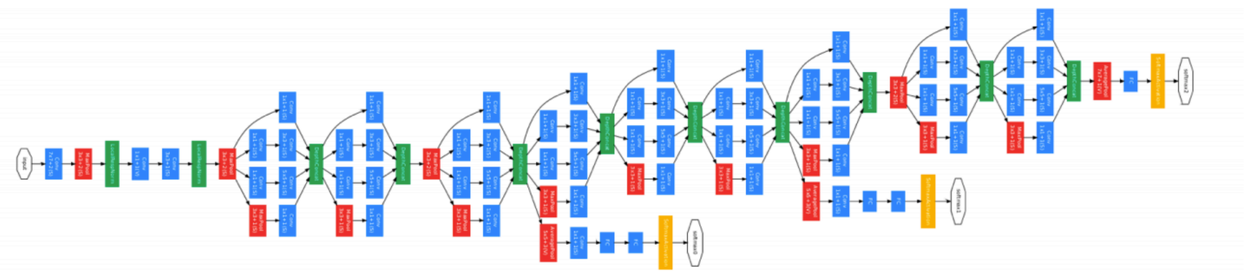
Support low-latency, high throughput serving workloads

- Models getting more complex
  - 10s of GFLOPs
  - Using specialized hardware for predictions

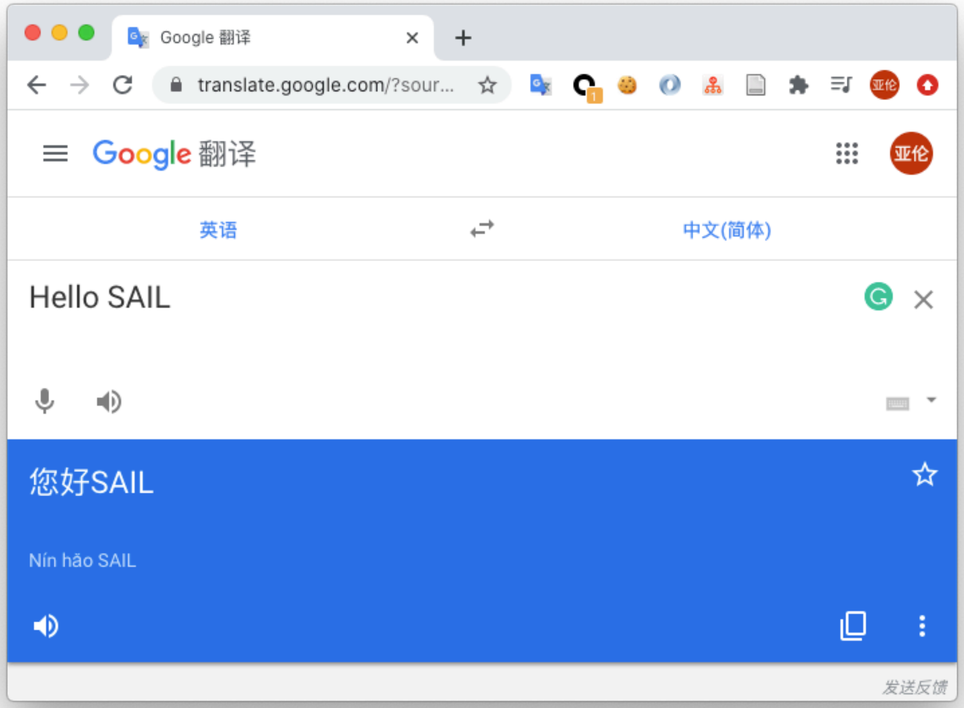


Support low-latency, high throughput serving workloads

- Models getting more complex
  - 10s of GFLOPs
  - Using specialized hardware for predictions
- Deployed on critical path
  - Maintain SLOs under heavy load



Google Translate



Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi
yonghui,schuster,zhifengc,qvl,mnorouzi@google.com

Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, Jeffrey Dean

140 billion words a day — 82,000 GPUs running 24/7

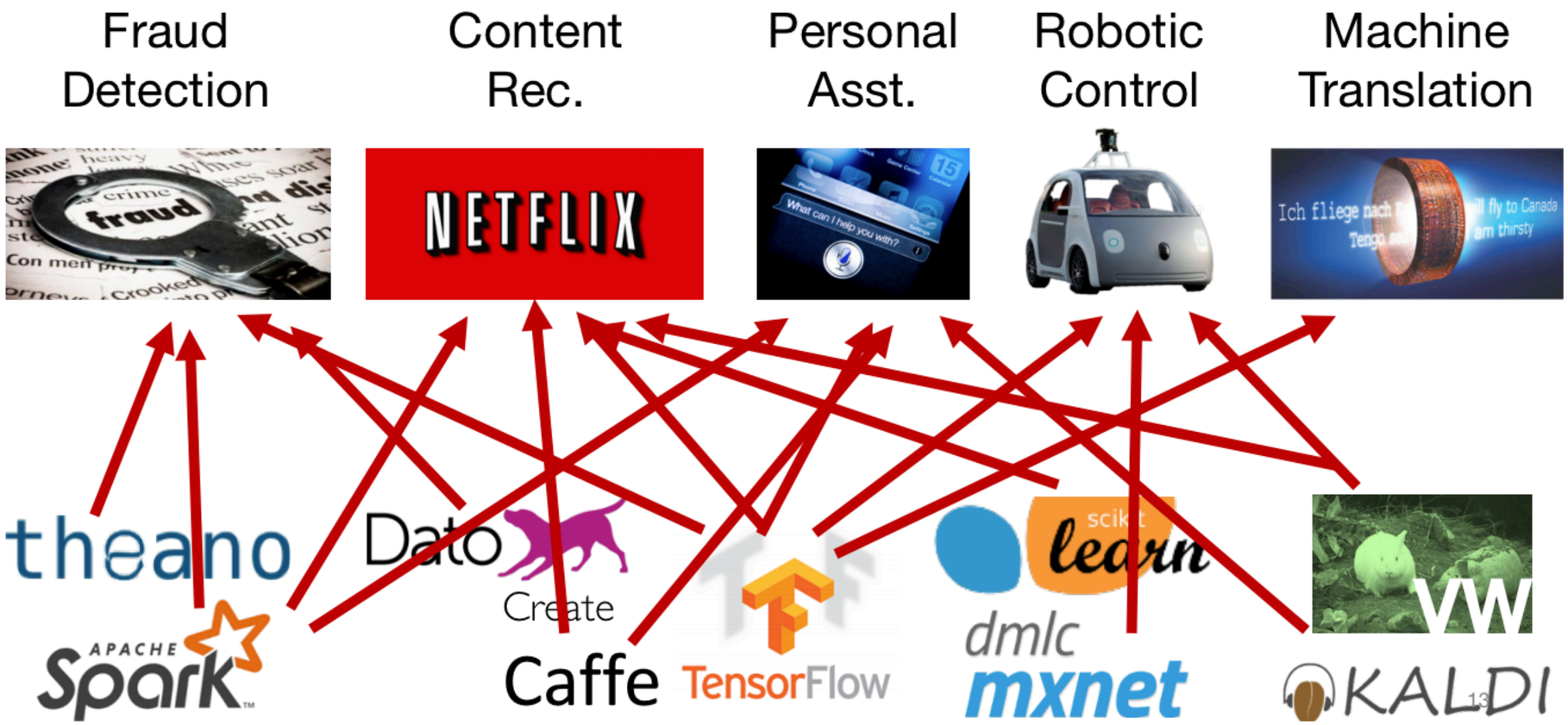
Invented new hardware — Tensor Processing Unit (TPU)

Big Companies Build One-off Systems

Problems:

- Models getting more complexExpensive to build and maintain
  - Highly specialized and require ML and systems expertise
- Tightly-coupled model and application
  - Difficult to change or update model
- Only supports single ML framework

Large and growing ecosystem of ML models and frameworks

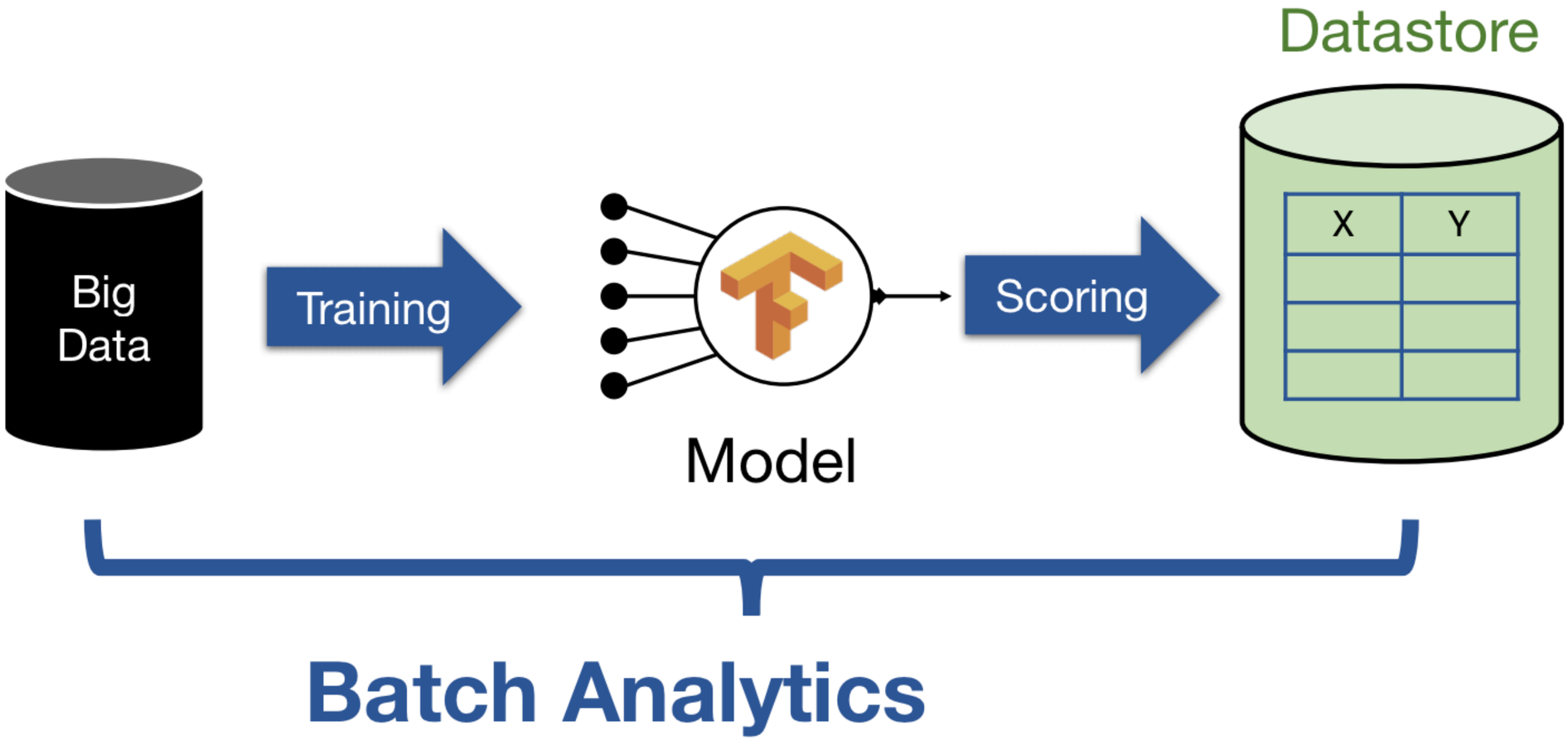


Large and growing ecosystem of ML models and frameworks

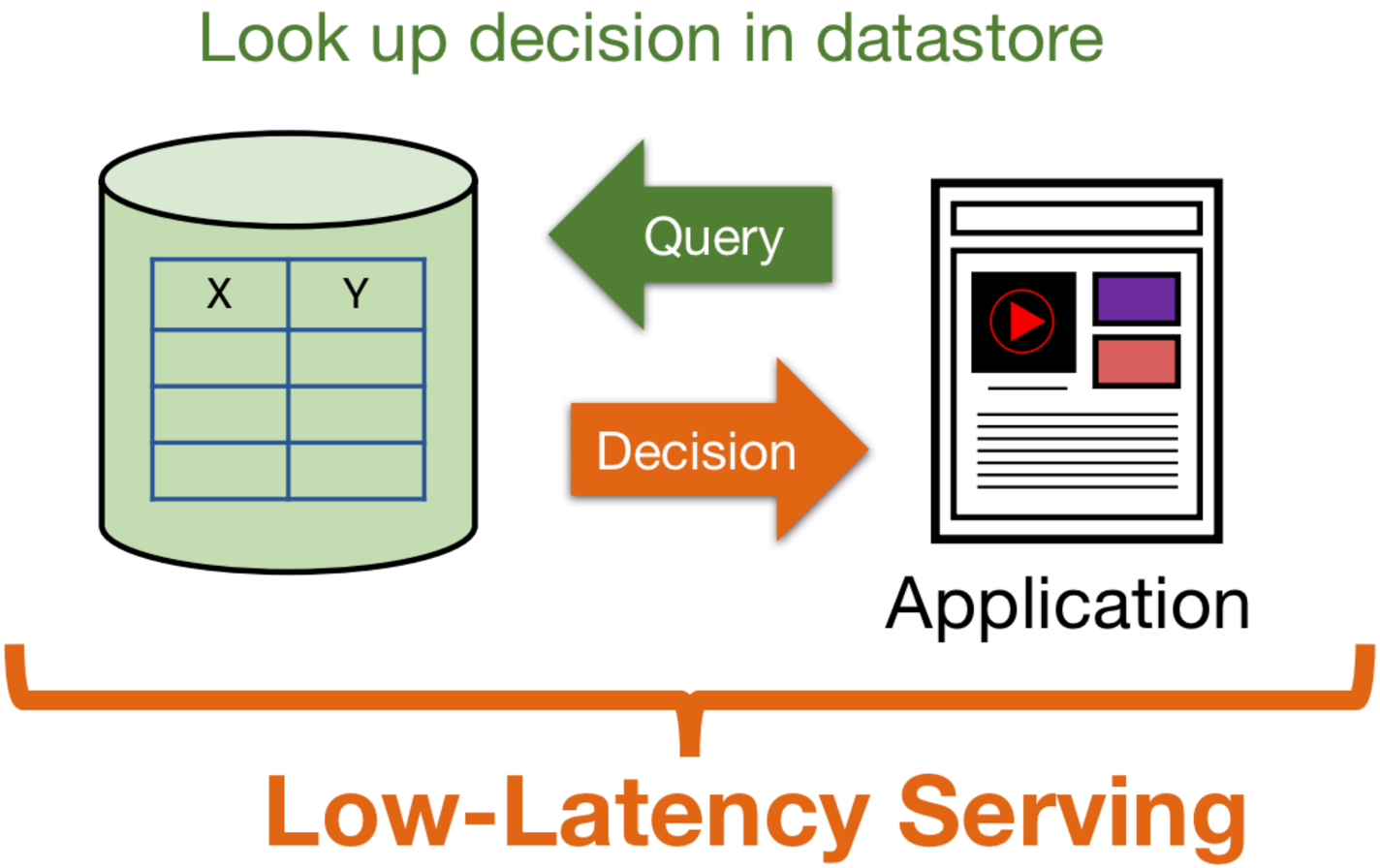
- **Difficult to deploy and brittle to manage**
- **Varying physical resource requirements**

But companies can’t build new serving systems…

Use existing systems: Offline Scoring



Use existing systems: Offline Scoring



Use existing systems: Offline Scoring

Problems:

- Requires full set of queries ahead of time
  - Small and bounded input domain
- Wasted computation and space
  - Can render and store unneeded predictions
- Costly to update
  - Re-run batch job

Prediction-Serving Challenge

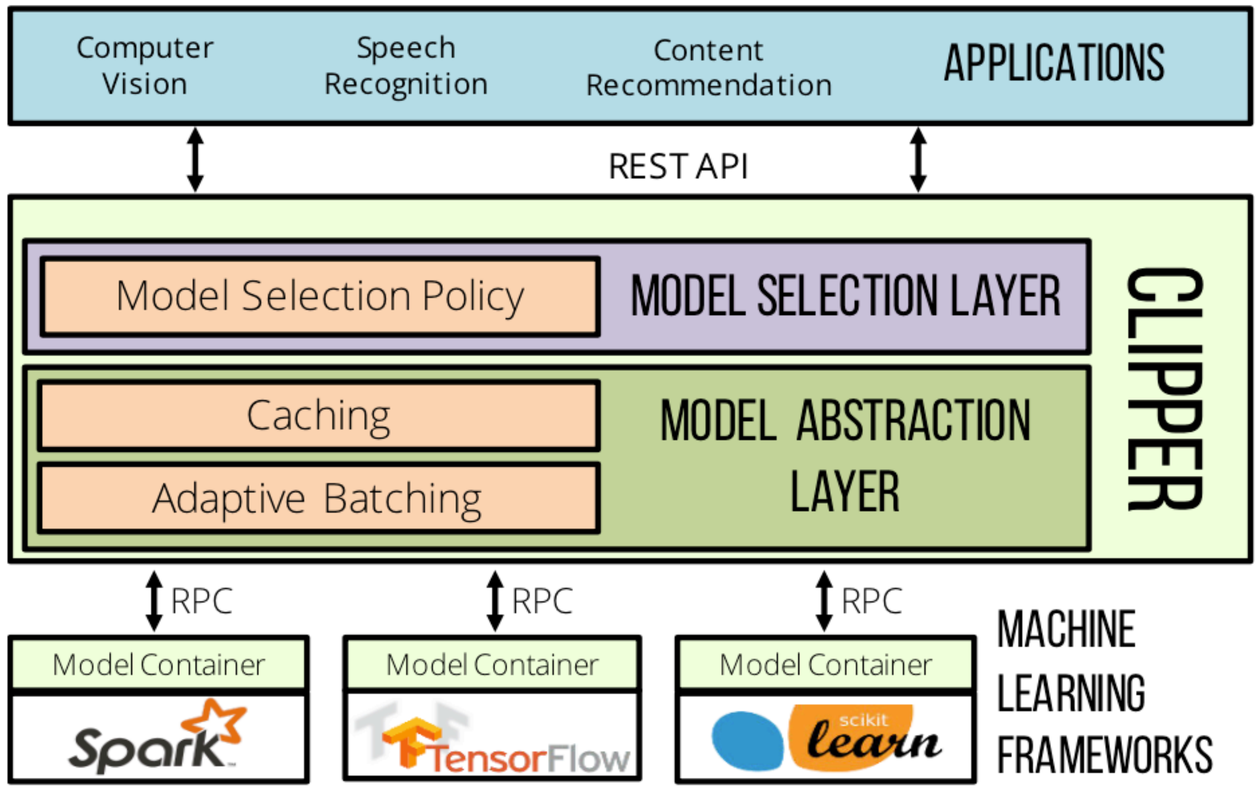
- Support low-latency, high throughput serving workloads

- Large and growing ecosystem of ML models and frameworks

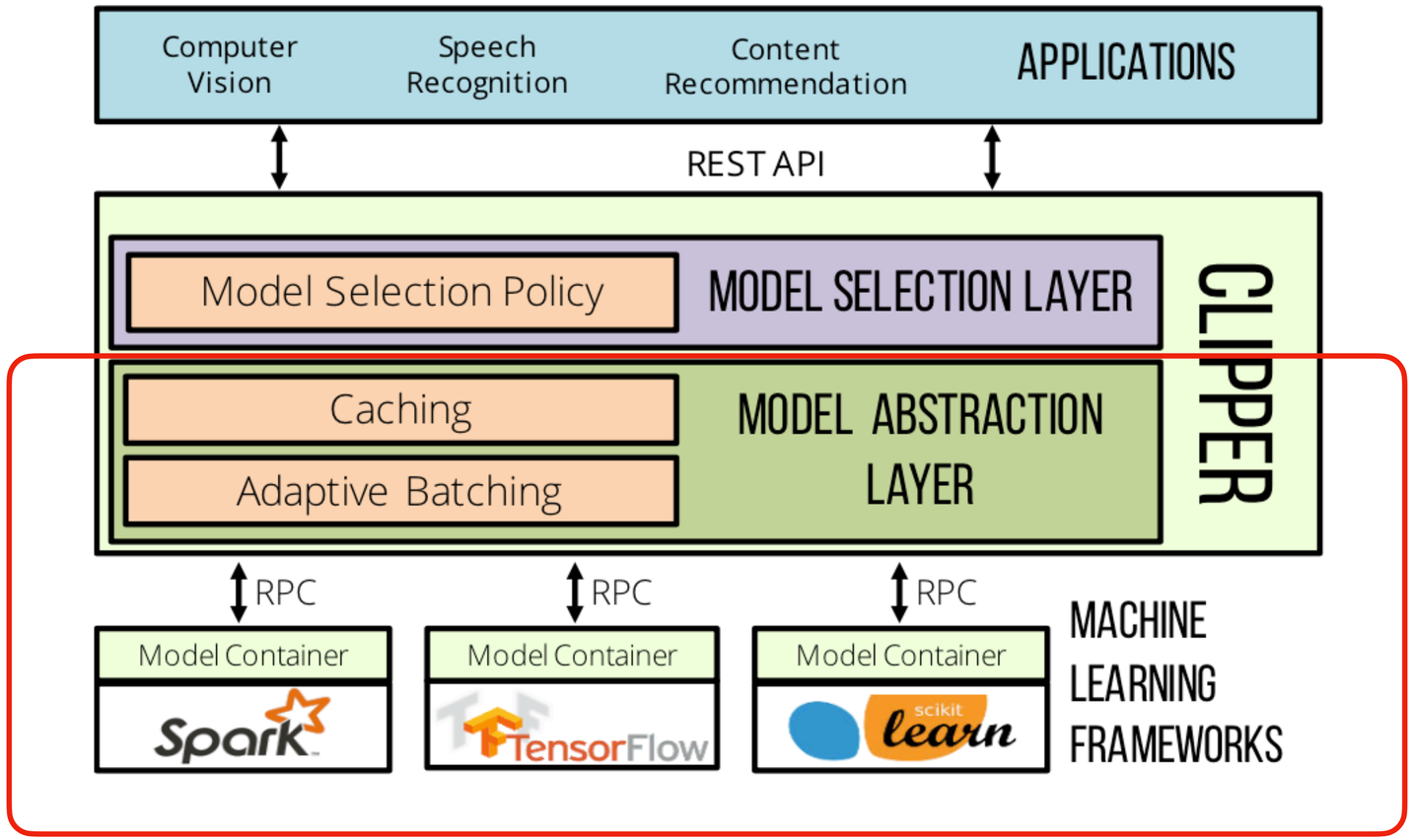
Clipper Solution

- Simplifies deployment through layered architecture
- Serves many models across ML frameworks concurrently
- Employs caching, batching, scale-out for high-performance serving

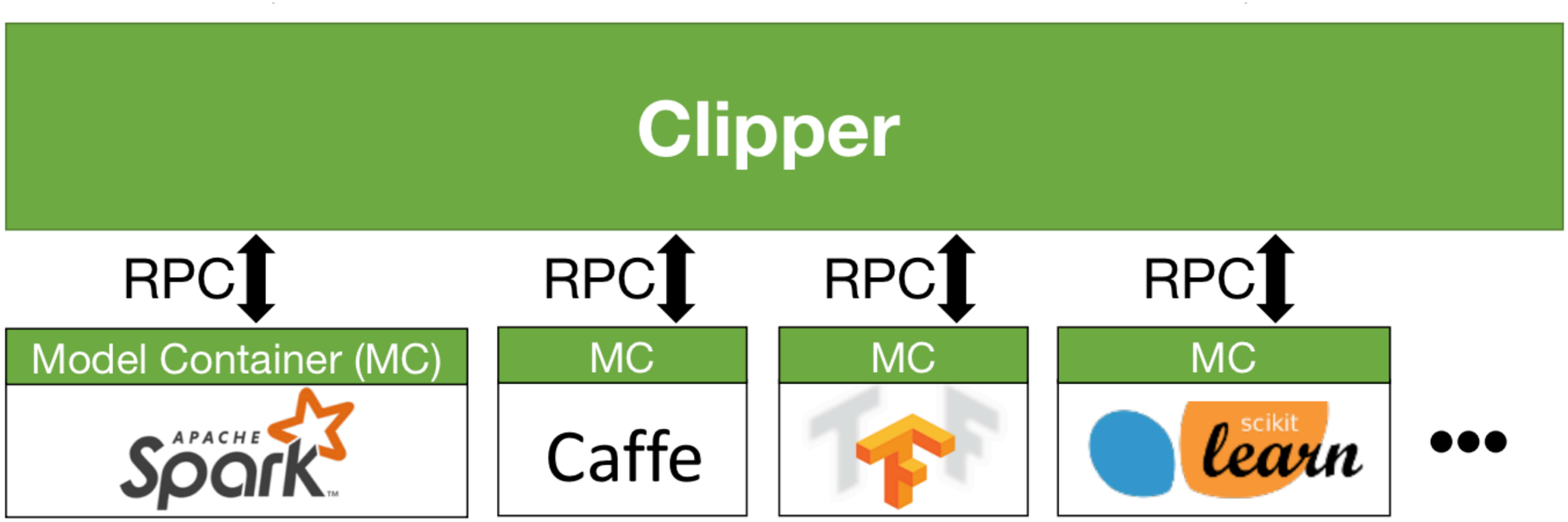
Clipper Architecture



Clipper Architecture



Container-based Model Deployment



Container-based Model Deployment

Common Interface —> Simplifies Deployment:

- Evaluate models using original code & systems

- Models run in separate processes as Docker containers
  - Resource Isolation
  - Scale-out

Container-based Model Deployment

Common Interface —> Simplifies Deployment:

- Evaluate models using original code & systems

- Models run in separate processes as Docker containers
  - Resource Isolation
  - Scale-out

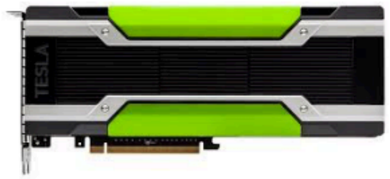
Problem: frameworks optimized for **batch processing** not **latency**

Batching to Improve Throughput

Why batching helps

A single page load may generate many queries

Hardware Acceleration



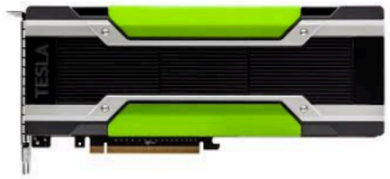
Helps amortize system overhead

Batching to Improve Throughput

Why batching helps

A single page load may generate many queries

Hardware Acceleration



GRPC

Helps amortize system overhead

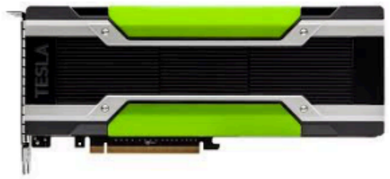
- **Optimal batch depends on**:
  - hardware configuration
  - model and framework
  - system load

Adaptive Batching — AIMD

Why batching helps

A single page load may generate many queries

Hardware Acceleration



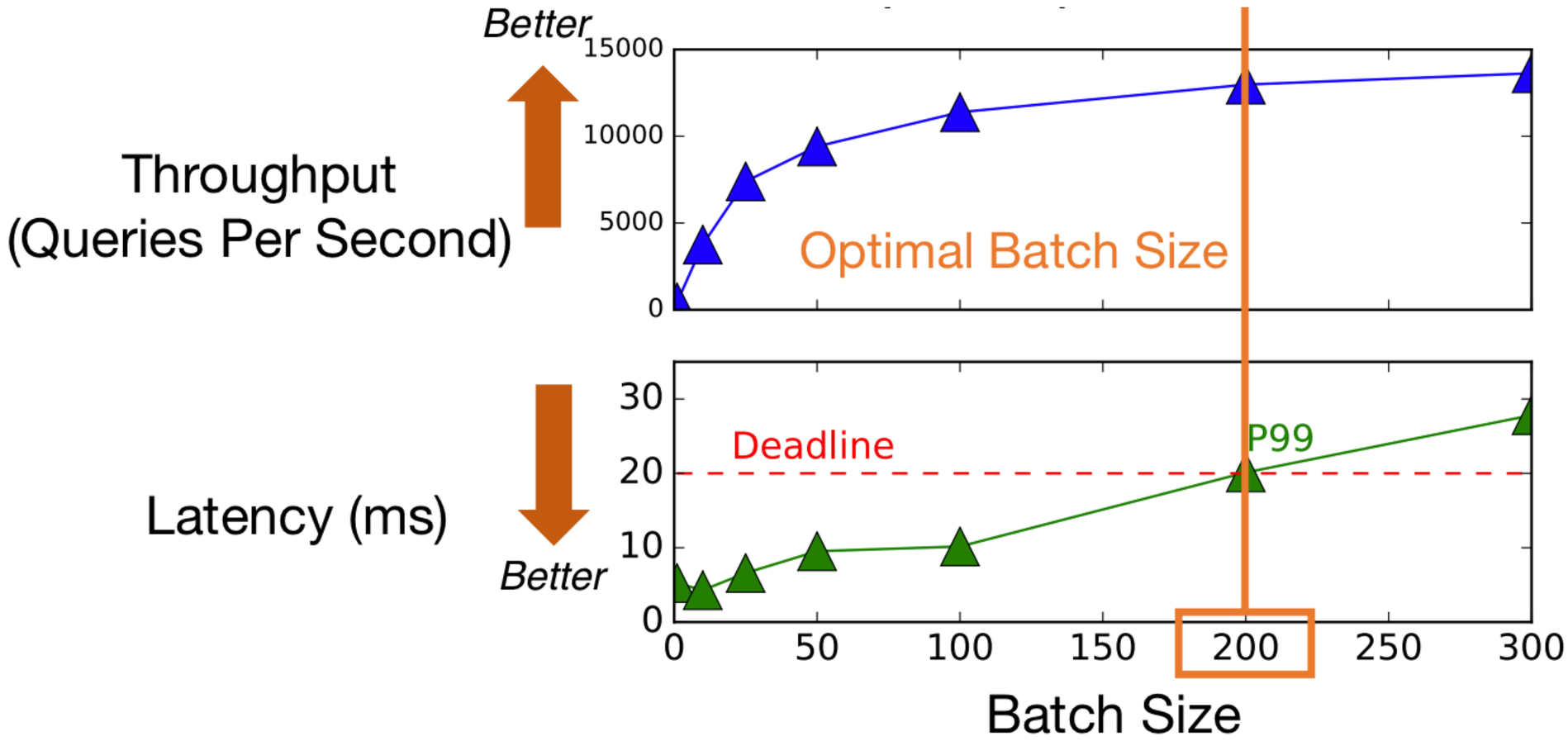
GRPC

Helps amortize system overhead

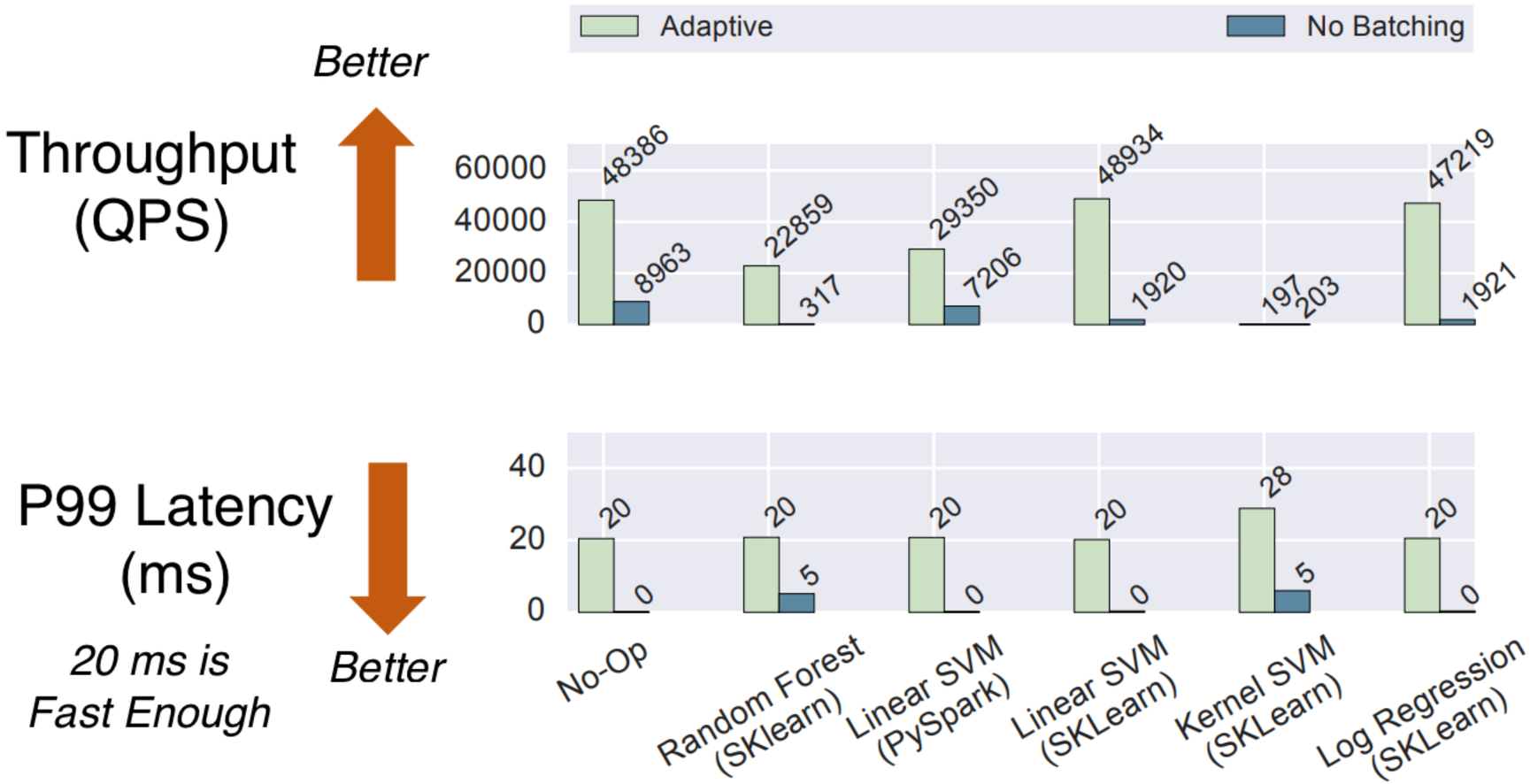
- **Optimal batch depends on**:
  - hardware configuration
  - model and framework
  - system load

- Clipper **Adaptively tradeoff latency and throughput…**
  - Inc. batch size until the latency objective is exceeded (**Additive Increase**)
  - If latency exceeds SLO cut batch size by a fraction (**Multiplicative Decrease**)

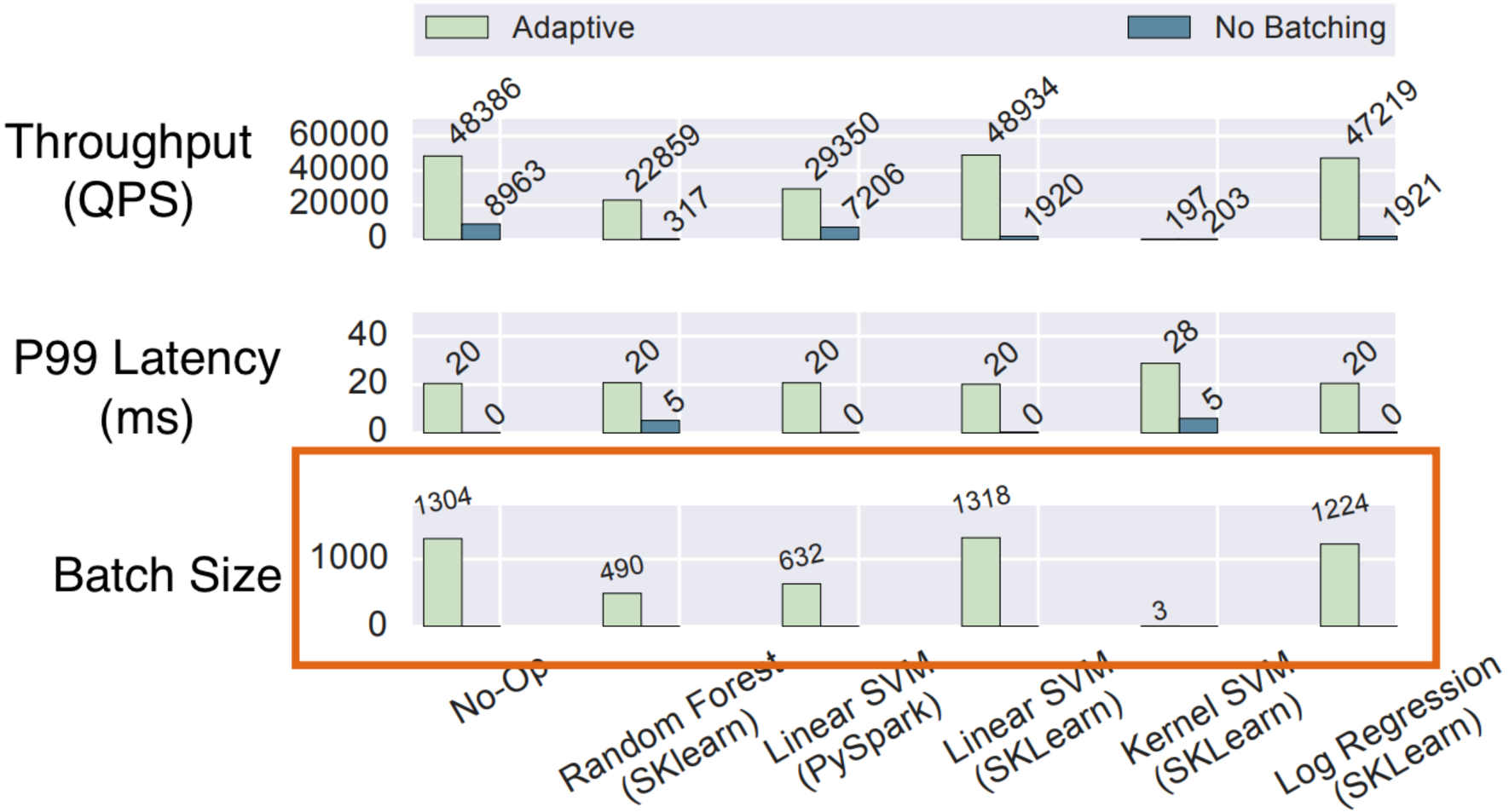
Tensor Flow Conv. Net (GPU)



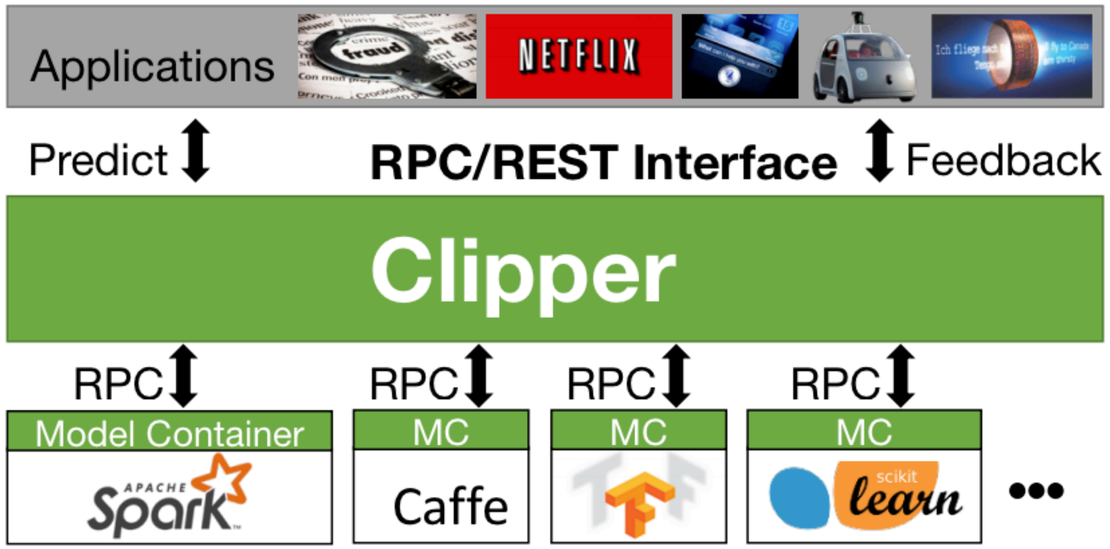
Different Frameworks

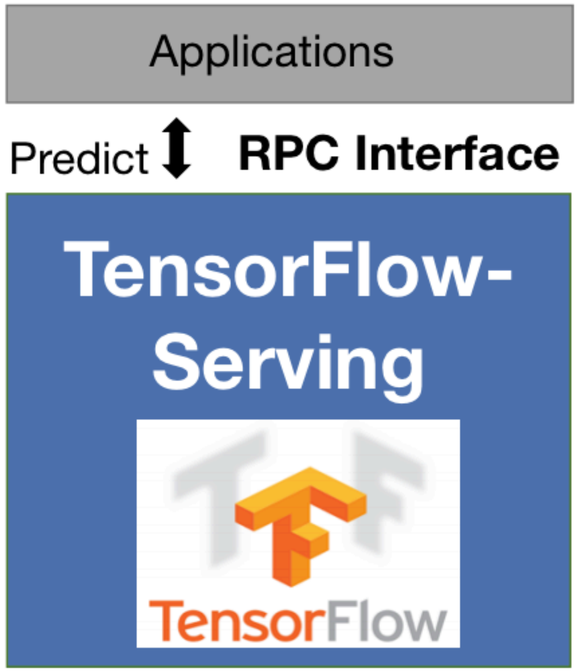


Different Frameworks

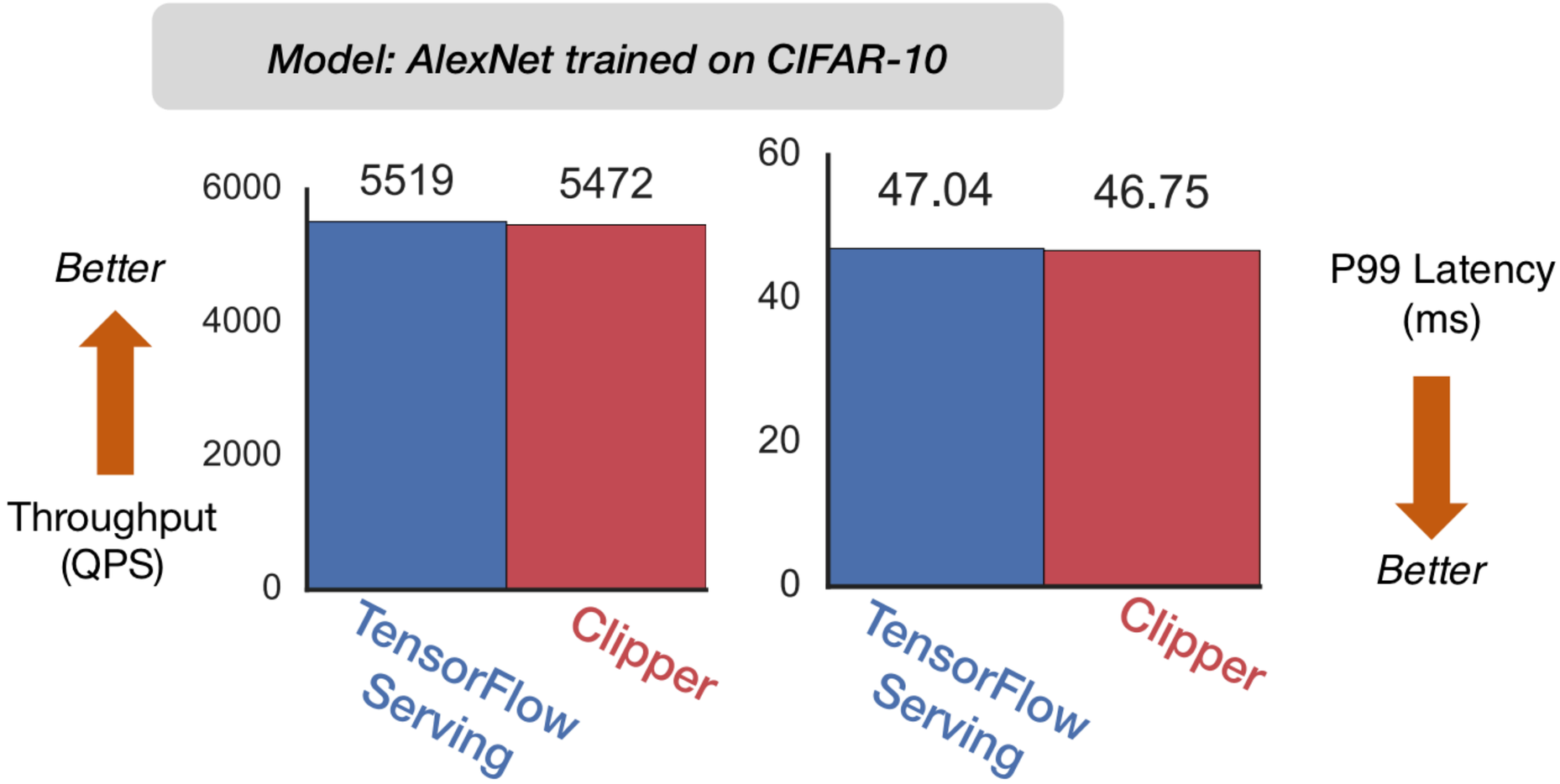


Overhead of Decoupled Architecture

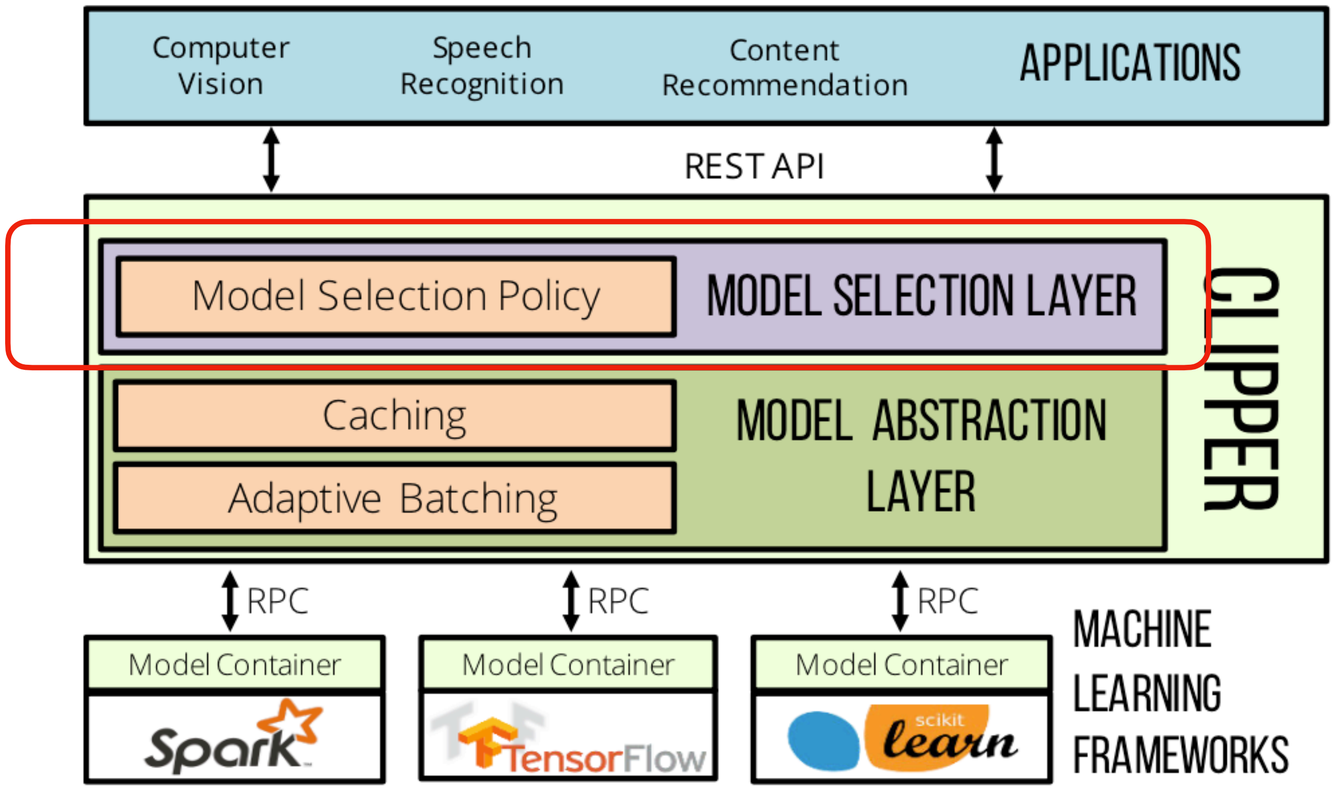




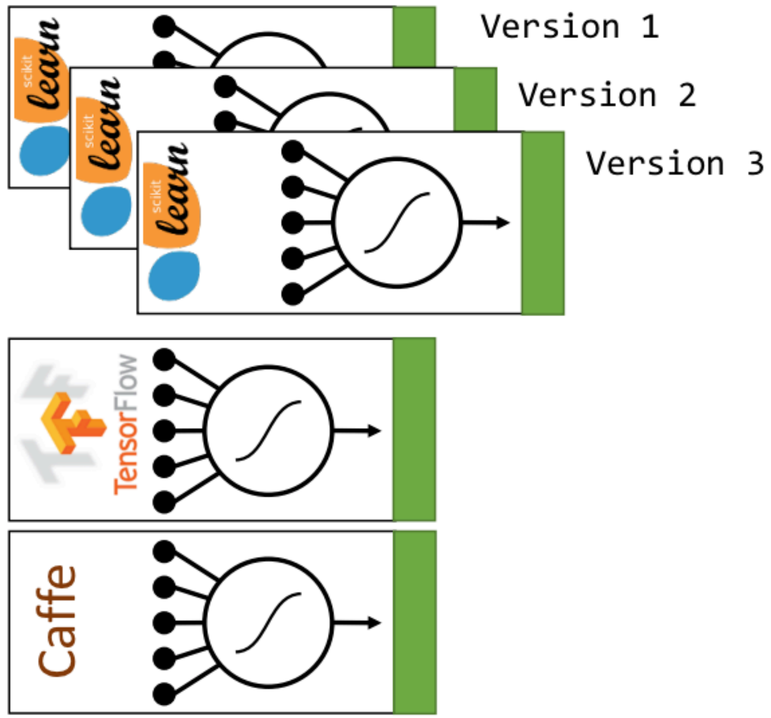
Overhead of Decoupled Architecture



Model Selection Layer



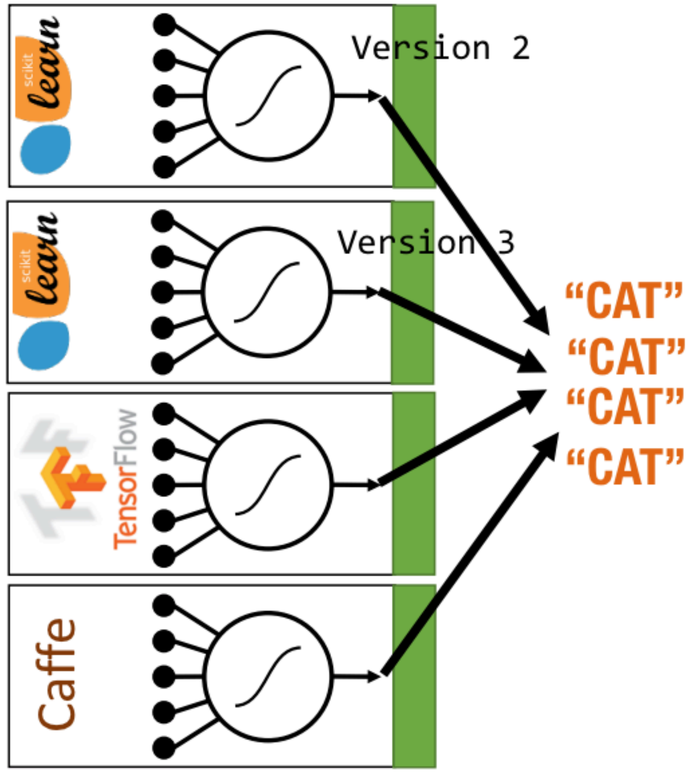
Model Selection



Periodic retraining

Experiment with new models and frameworks

Selection Policy: Estimate confidence

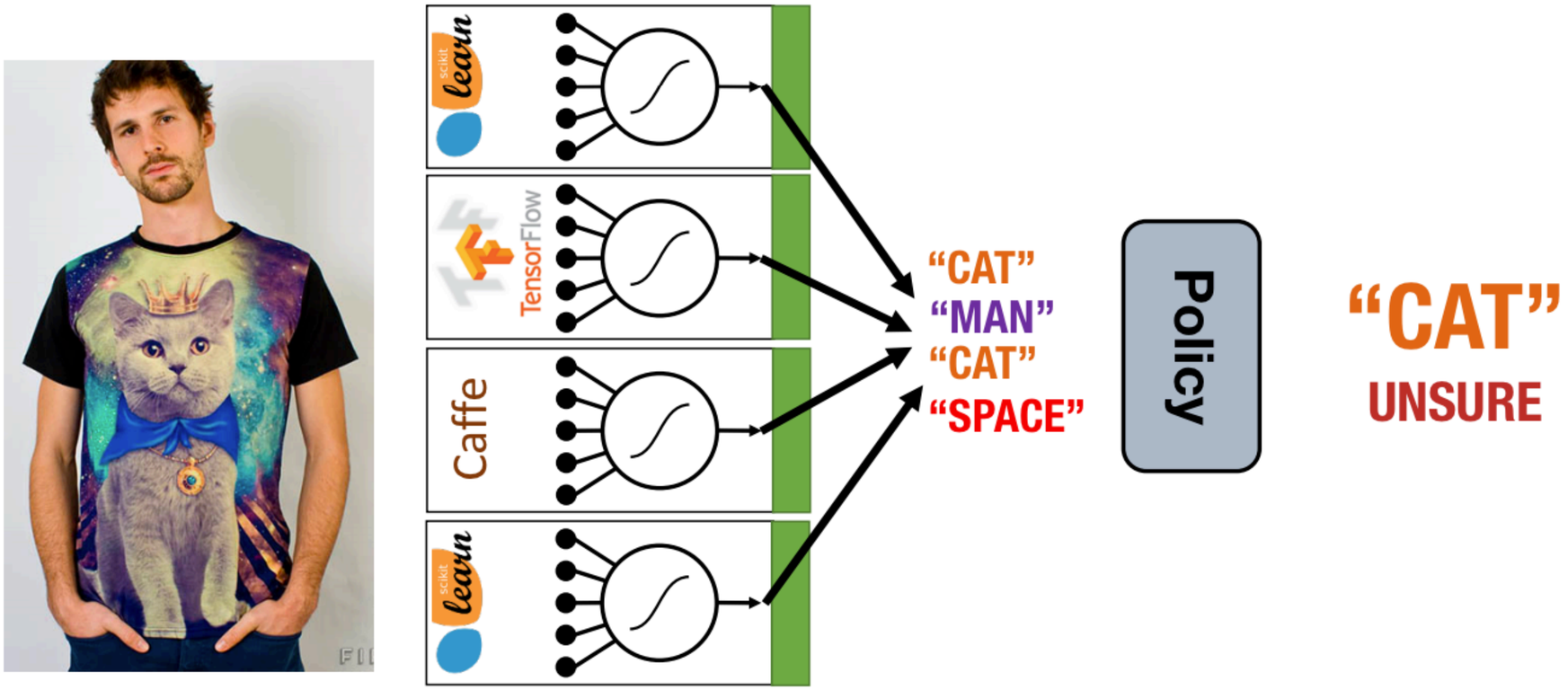


Policy

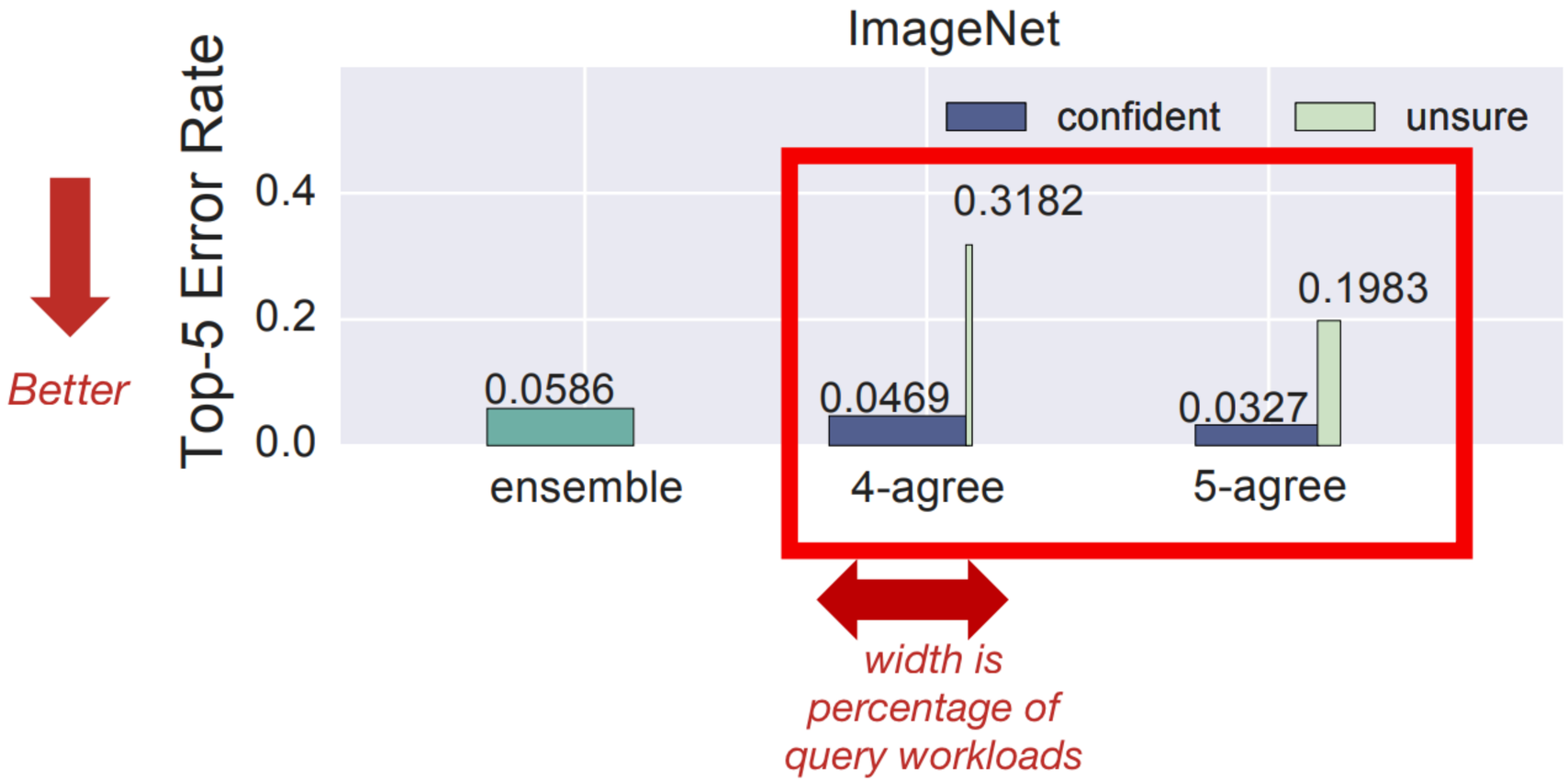
“CAT”

CONFIDENT

Selection Policy: Estimate confidence



Selection Policy: Estimate confidence



Summary

- **Prediction-serving** is an important and **challenging** area for systems research
  - Support low-latency, high-throughput serving workloads
  - Serve large and growing ecosystem of ML frameworks

- **Clipper** is a first **step** towards addressing these challenges
  - Simplifies deployment through layered architecture
  - Serves many models across ML frameworks concurrently
  - Employs caching, adaptive batching, container scale-out to meet interactive serving workload demands

OSDI 2018

Ray

A Distributed Framework for Emerging AI Applications

https://www.usenix.org/conference/osdi18/presentation/nishiharaw

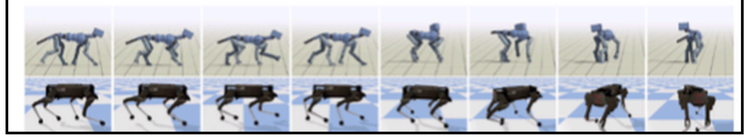
Emerging AI Applications

Learning Agile Robotic Locomotion Skills by Imitating Animals

arXiv Preprint 2020

Xue Bin Peng (1,2) Erwin Coumans (1) Tingnan Zhang (1) Tsang-Wei Lee (1) Jie Tan (1) Sergey Levine (1,2)

(1) Google Research (2) University of California, Berkeley



Virtual-Taobao: Virtualizing Real-world Online Retail Environment for Reinforcement Learning

Jing-Cheng Shi[1,2], Yang Yu[1], Qing Da[2], Shi-Yong Chen[1], An-Xiang Zeng[2]

[1]National Key Laboratory for Novel Software Technology, Nanjing University, China

[2]Alibaba Group, China

Wuji: Automatic Online Combat Game Testing Using Evolutionary Deep Reinforcement Learning

Yan Zheng[1,*], Xiaofei Xie[2,*], Ting Su[2], Lei Ma[3], Jianye Hao[1,✉], Zhaopeng Meng[1], Yang Liu[2], Ruimin Shen[4], Yingfeng Chen[4], Changjie Fan[4]

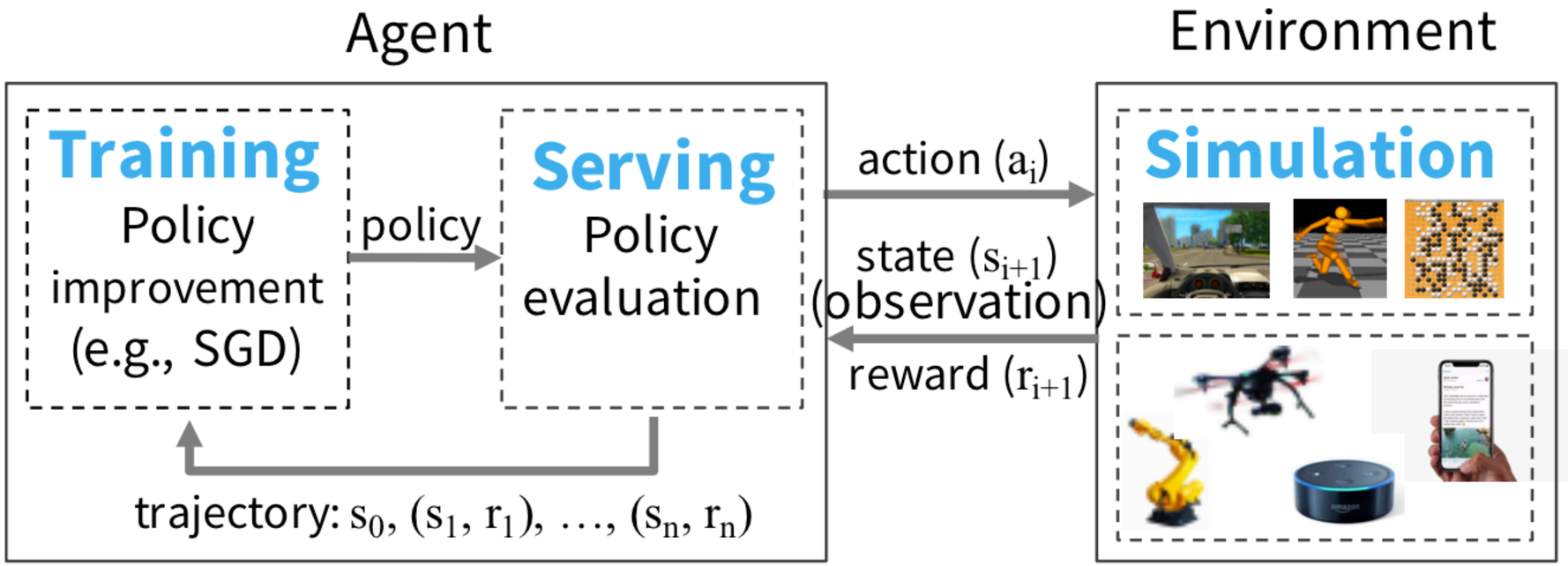
[1] *College of Intelligence and Computing, Tianjin University, China.*

[2] *Nanyang Technological University, Singapore.*

[3] *Kyushu University, Japan.*

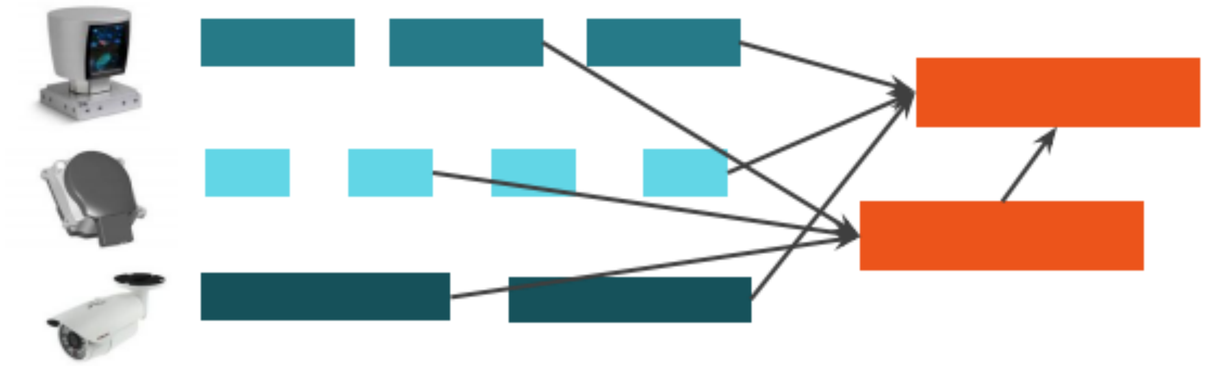
[4] *Fuxi AI Lab, Netease, Inc., Hangzhou, China.*

Example of an RL system



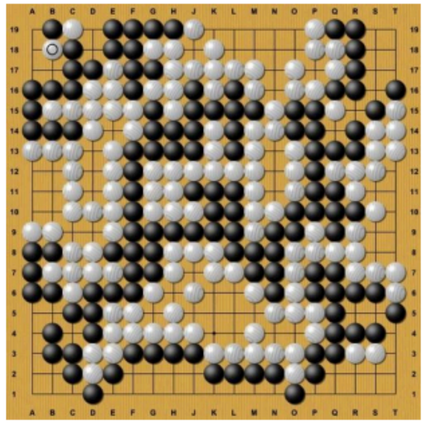
RL Application Pattern

- Process inputs from different sensors in parallel & real-time



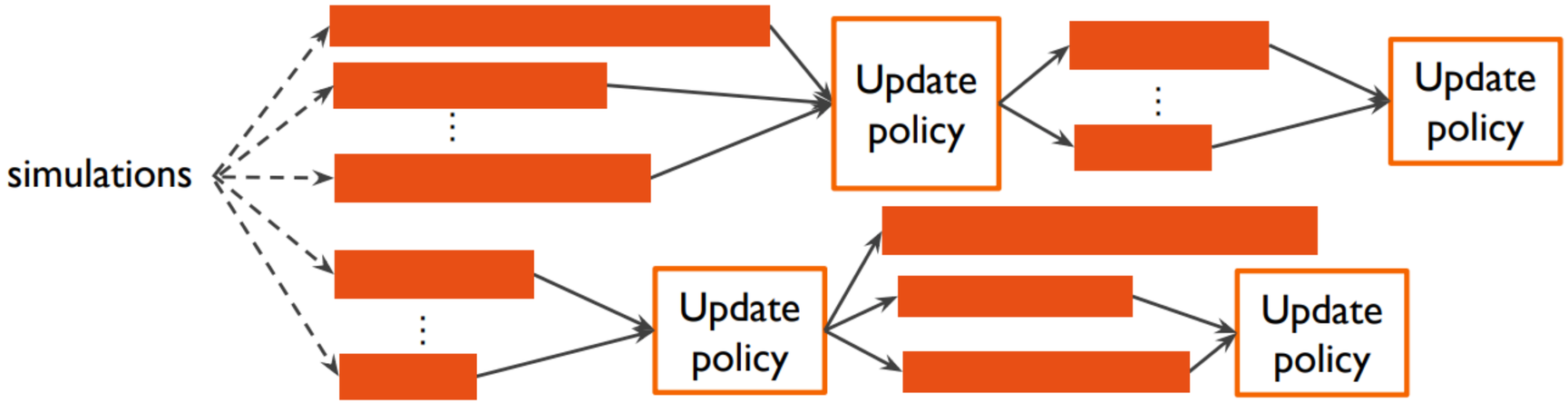
RL Application Pattern

- Process inputs from different sensors in parallel & real-time
- Execute large number of simulations, e.g., up to 100s of millions



RL Application Pattern

- Process inputs from different sensors in parallel & real-time
- Execute large number of simulations, e.g., up to 100s of millions
- Rollouts outcomes are used to update policy (e.g., SGD)



RL Application Pattern

- Process inputs from different sensors in parallel & real-time
- Execute large number of simulations, e.g., up to 100s of millions
- Rollouts outcomes are used to update policy (e.g., SGD)
- Often policies implemented by DNNs
- Most RL algorithms developed in Python

RL Application Requirements

- Need to handle dynamic task graphs, where tasks have
  - Heterogeneous durations
  - Heterogeneous computations
- Schedule millions of tasks/sec
- Make it easy to parallelize ML algorithms written in Python

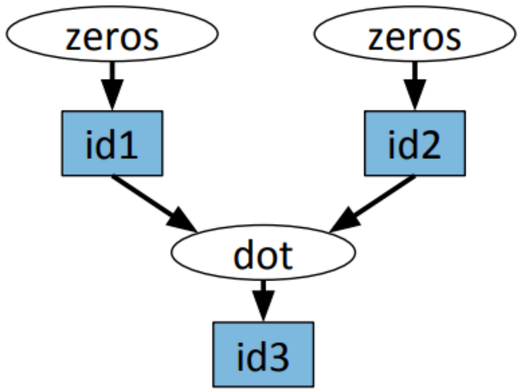
Ray API — remote functions

```
@ray.remote
def zeros(shape):
    return np.zeros(shape)

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

Ray API — remote functions

```
@ray.remote
def zeros(shape):
    return np.zeros(shape)

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

id1 = zeros.remote([5, 5])
id2 = zeros.remote([5, 5])
id3 = dot.remote(id1, id2)
ray.get(id3)
```

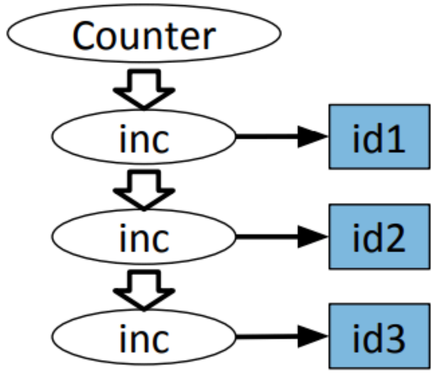
- **Blue** variables are Object IDs.



Ray API — actors

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

c = Counter.remote()
id1 = c.inc.remote()
id2 = c.inc.remote()
id3 = c.inc.remote()
ray.get([id1, id2, id3])  # This returns [1, 2, 3]
```

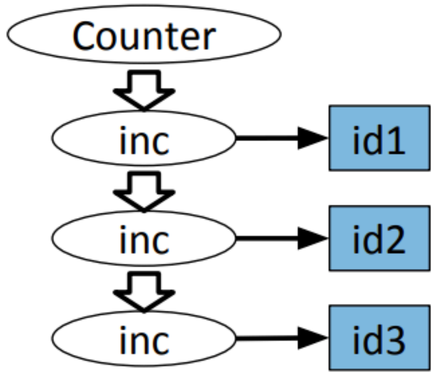
- State is shared between actor methods.
- Actor methods return **Object IDs**.



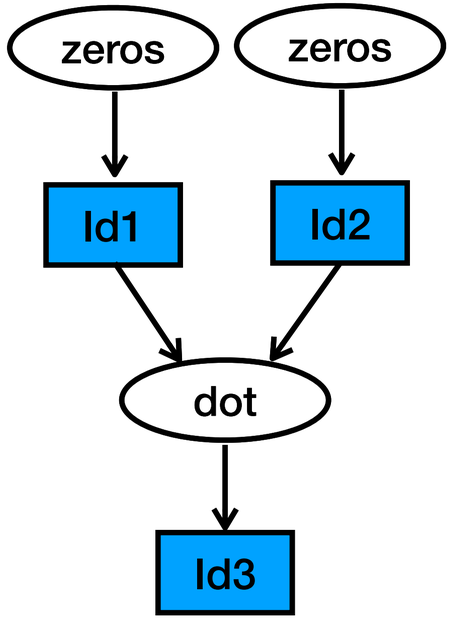
Ray API — actors

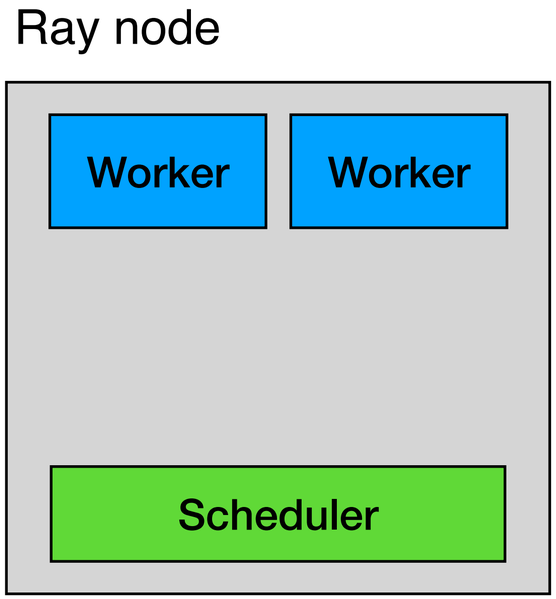
```
@ray.remote(num_gpus=1)
class Counter(object):
    def __init__(self):
        self.value = 0
    def inc(self):
        self.value += 1
        return self.value

c = Counter.remote()
id1 = c.inc.remote()
id2 = c.inc.remote()
id3 = c.inc.remote()
ray.get([id1, id2, id3])  # This returns [1, 2, 3]
```

- State is shared between actor methods.
- Actor methods return **Object IDs**.
- Can specify **GPU** requirements

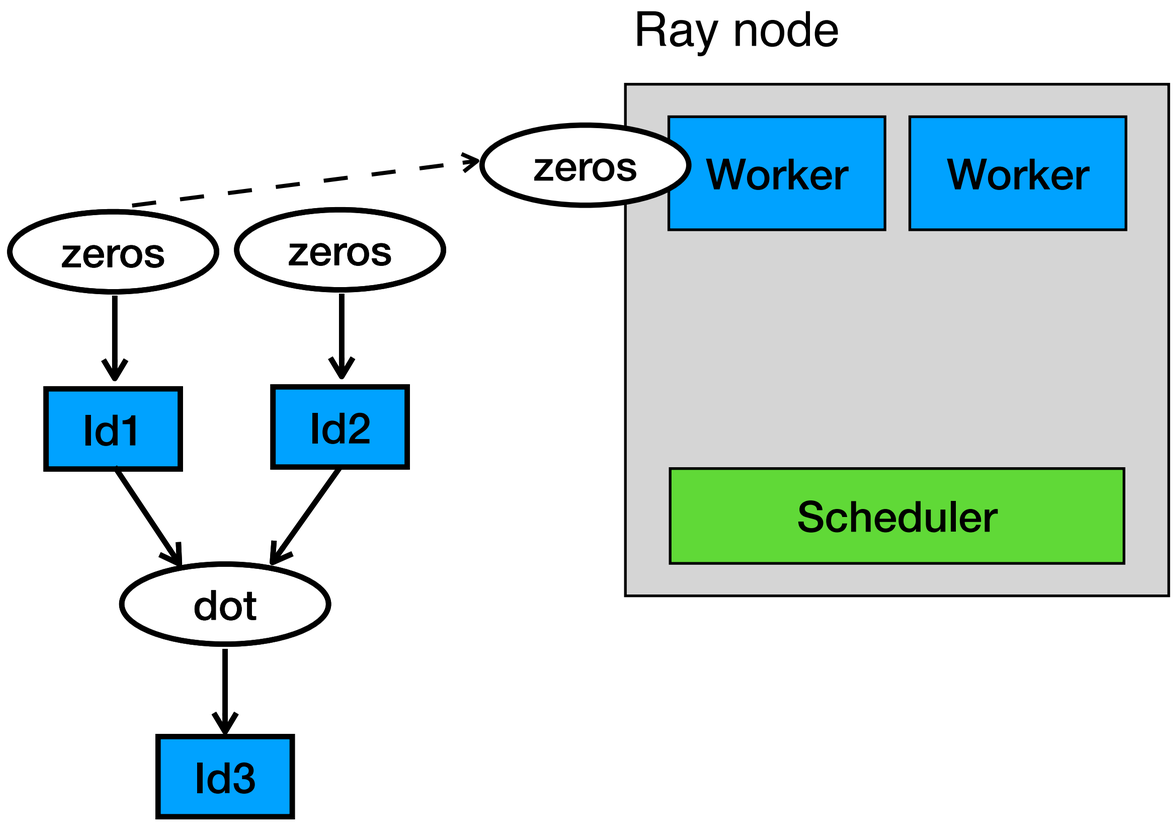


Architecture

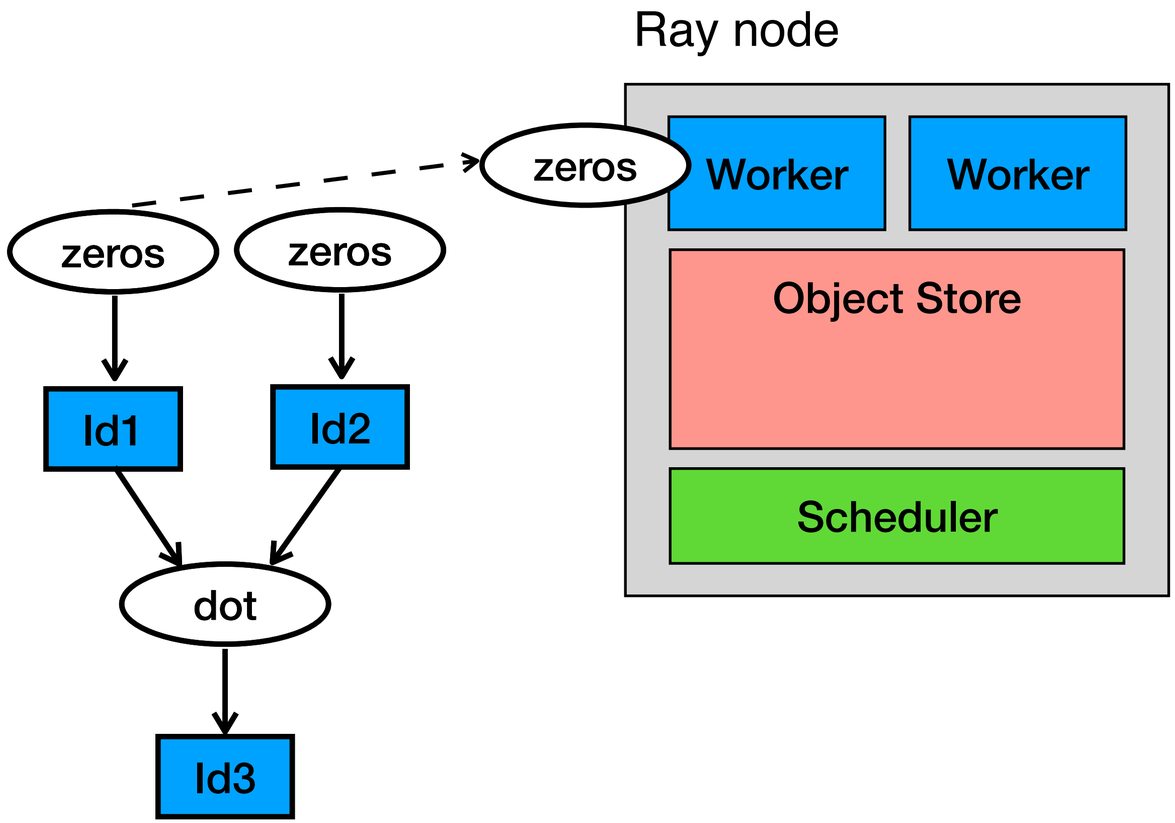




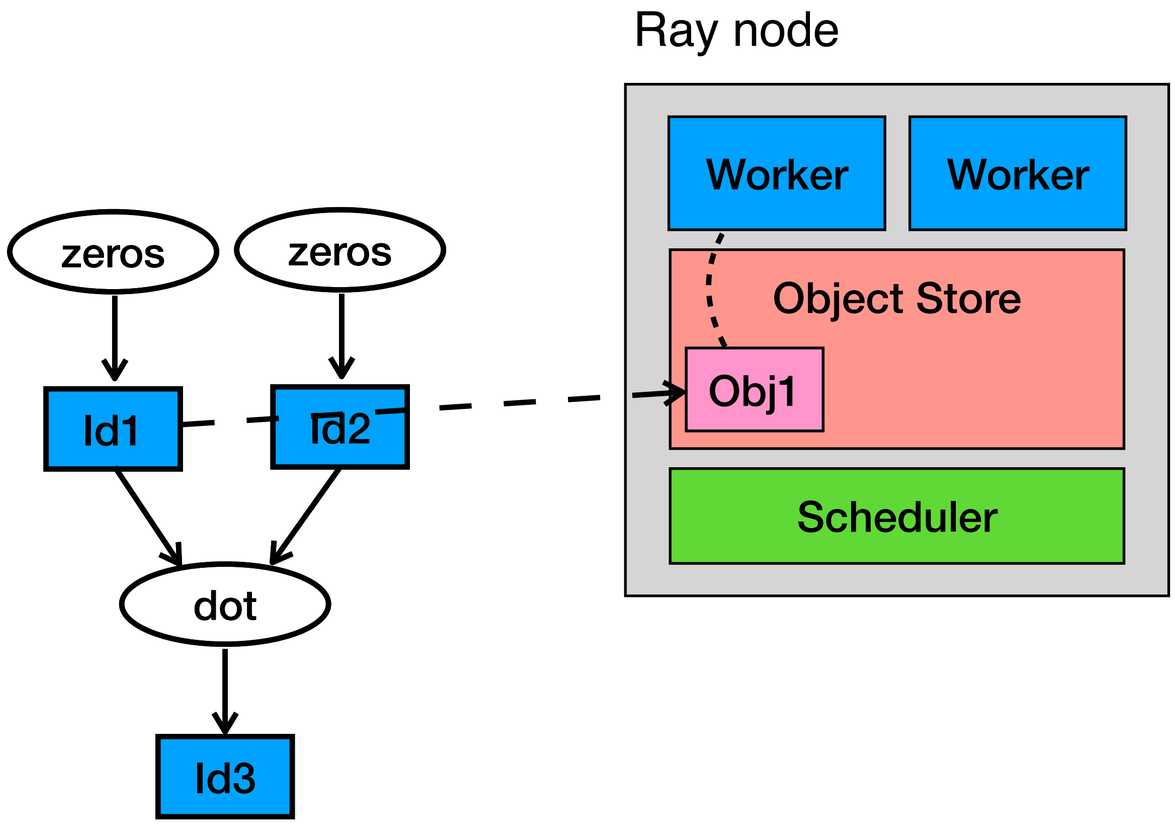
Architecture



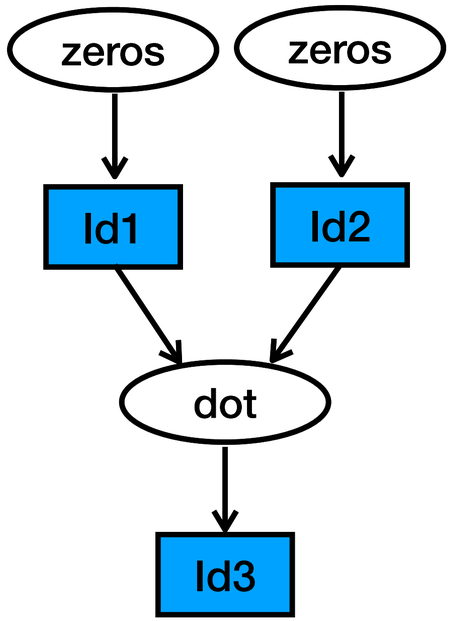
Architecture

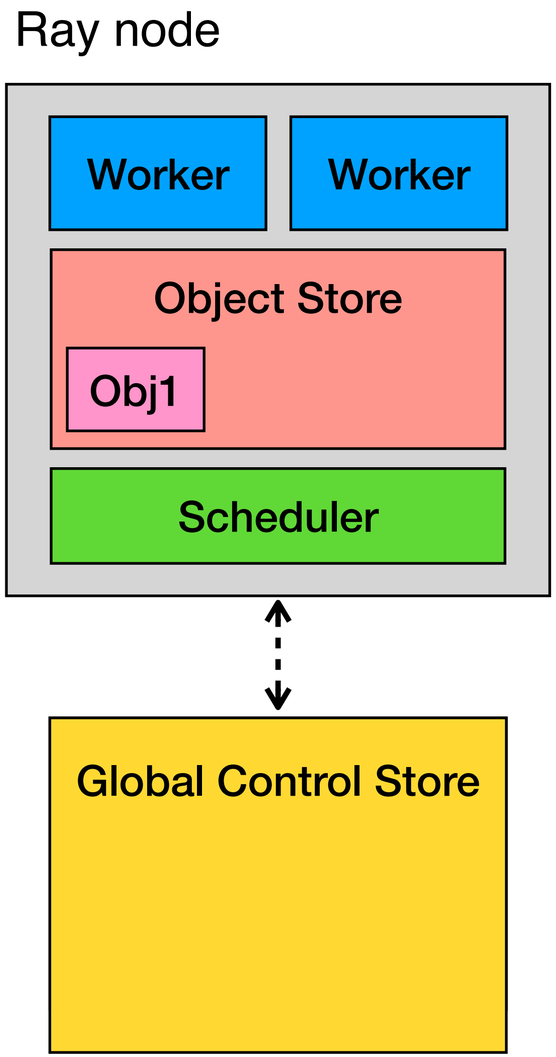


Architecture

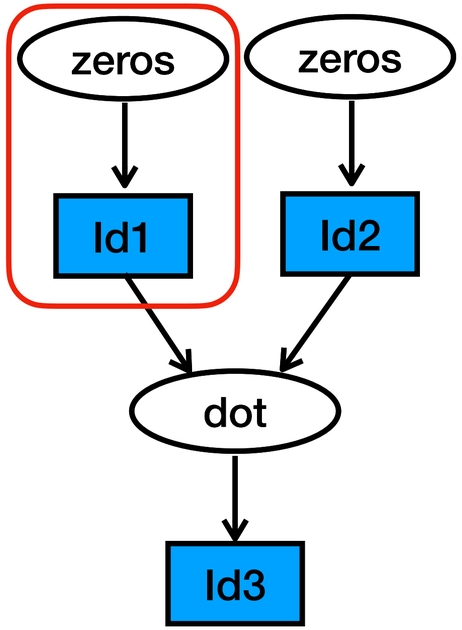


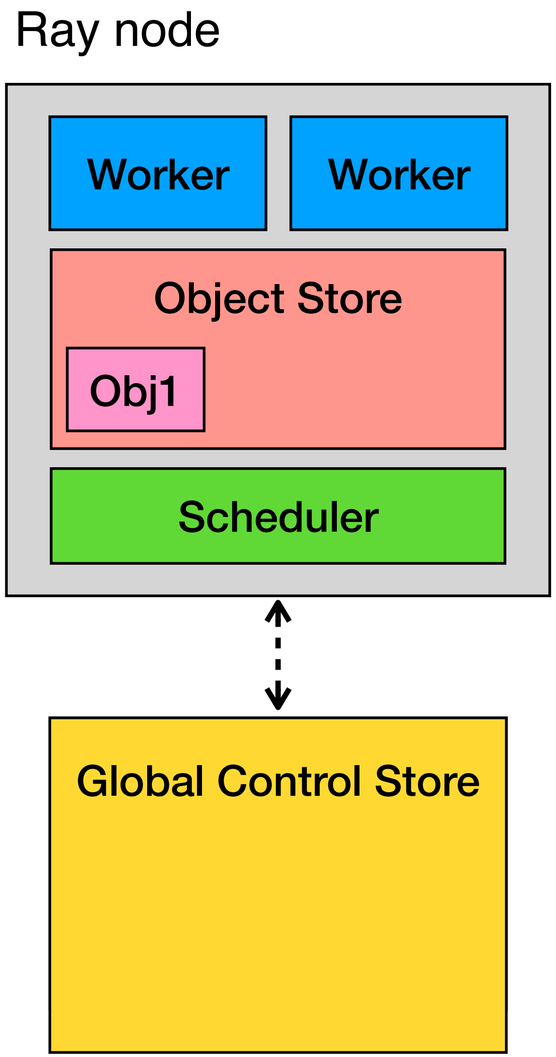
Architecture



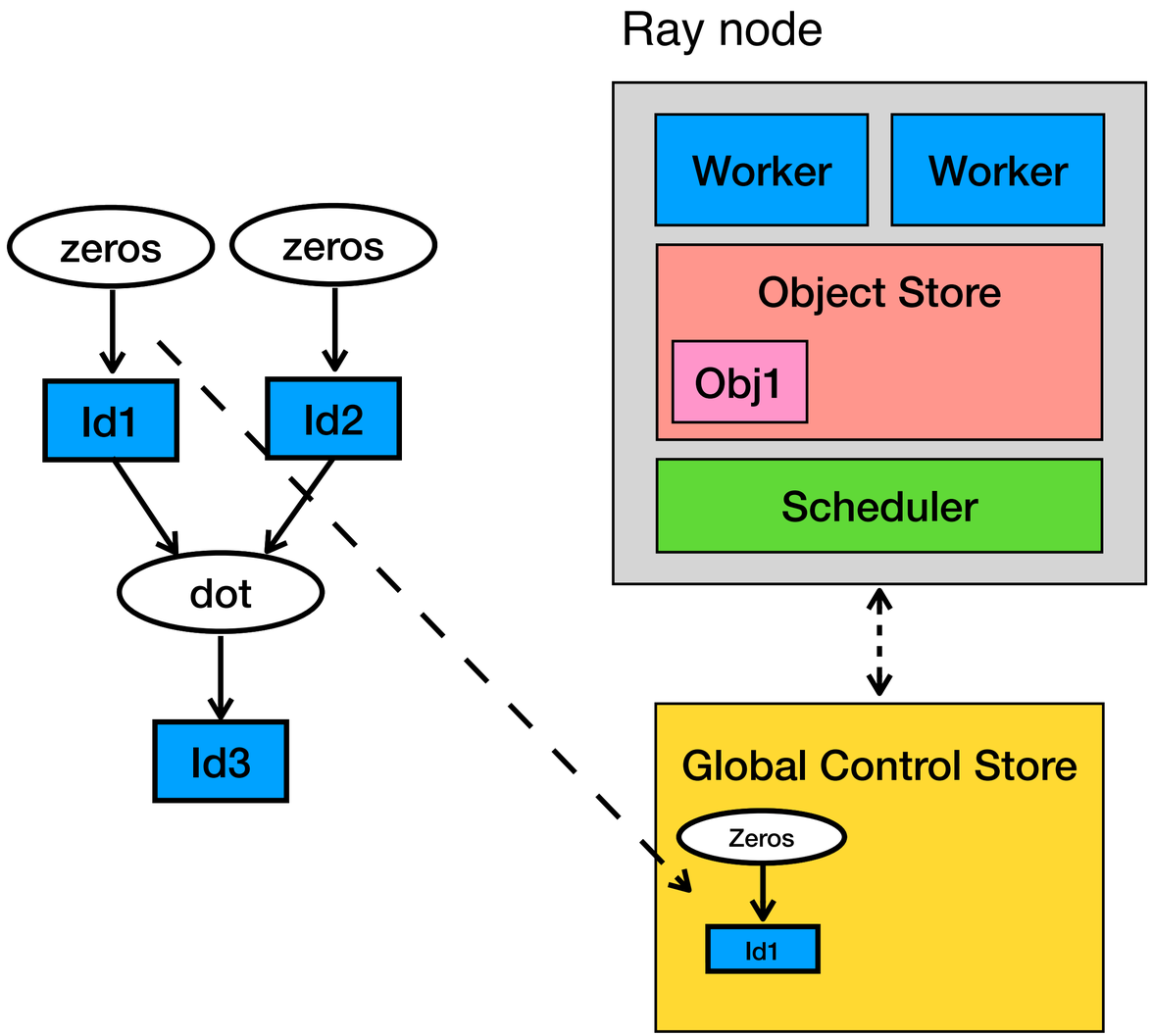


Architecture

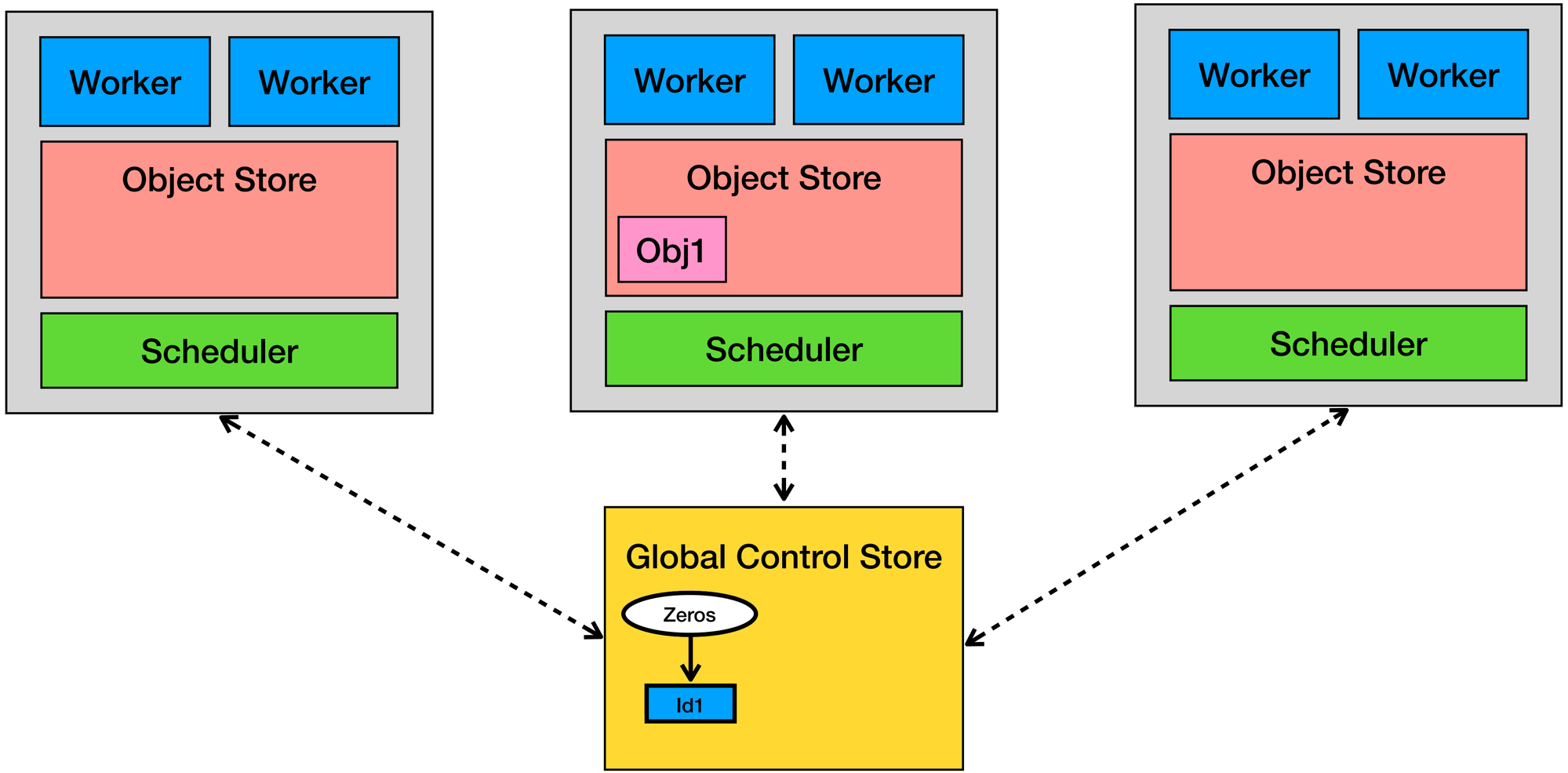




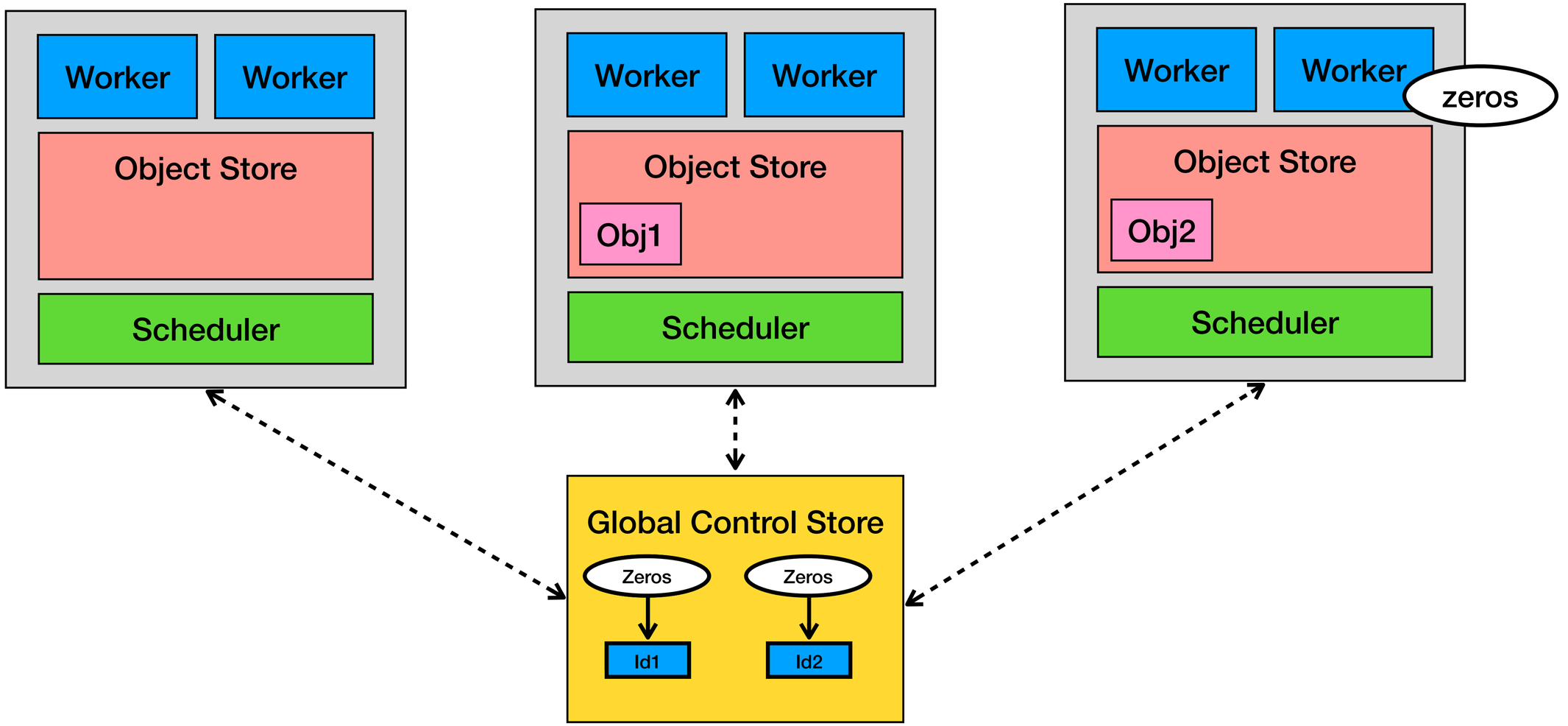
Architecture



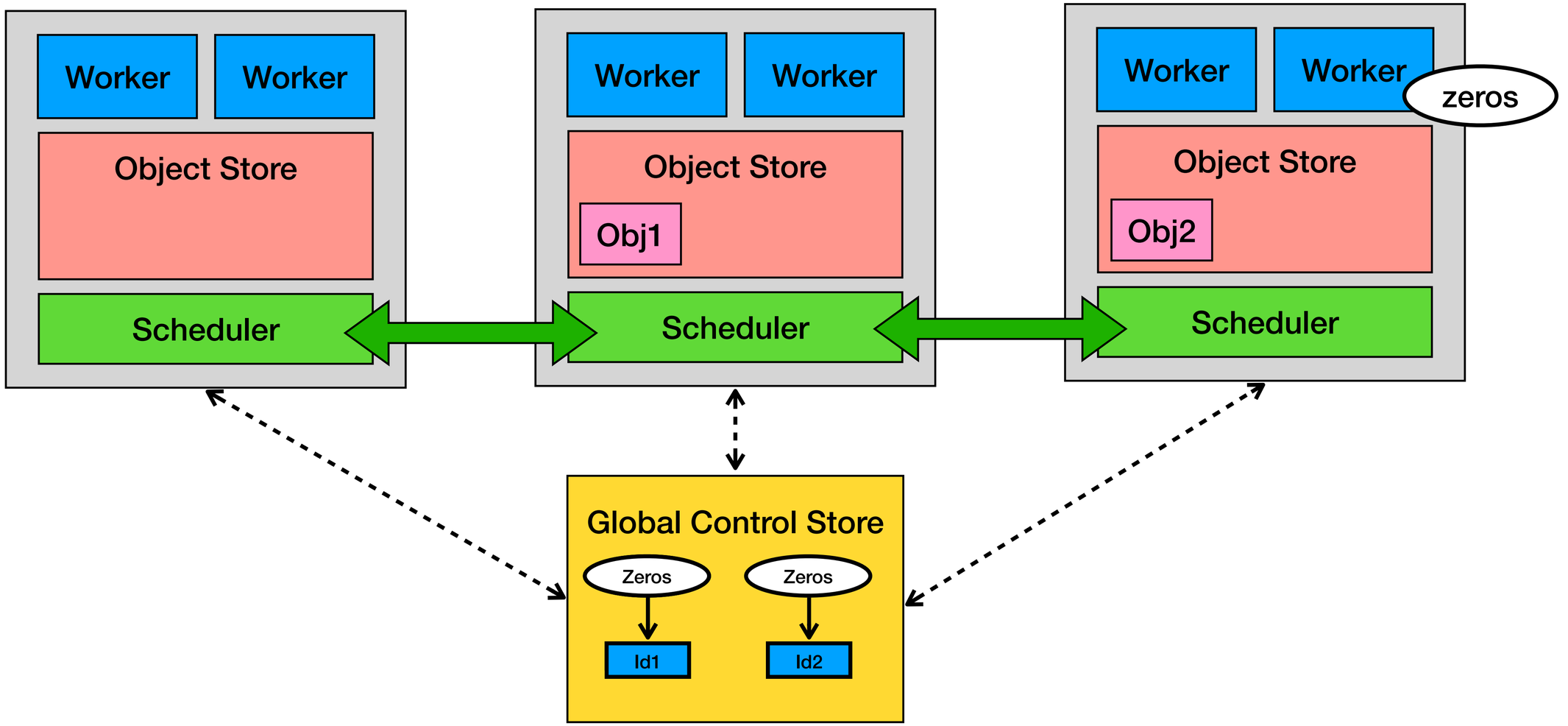
Architecture



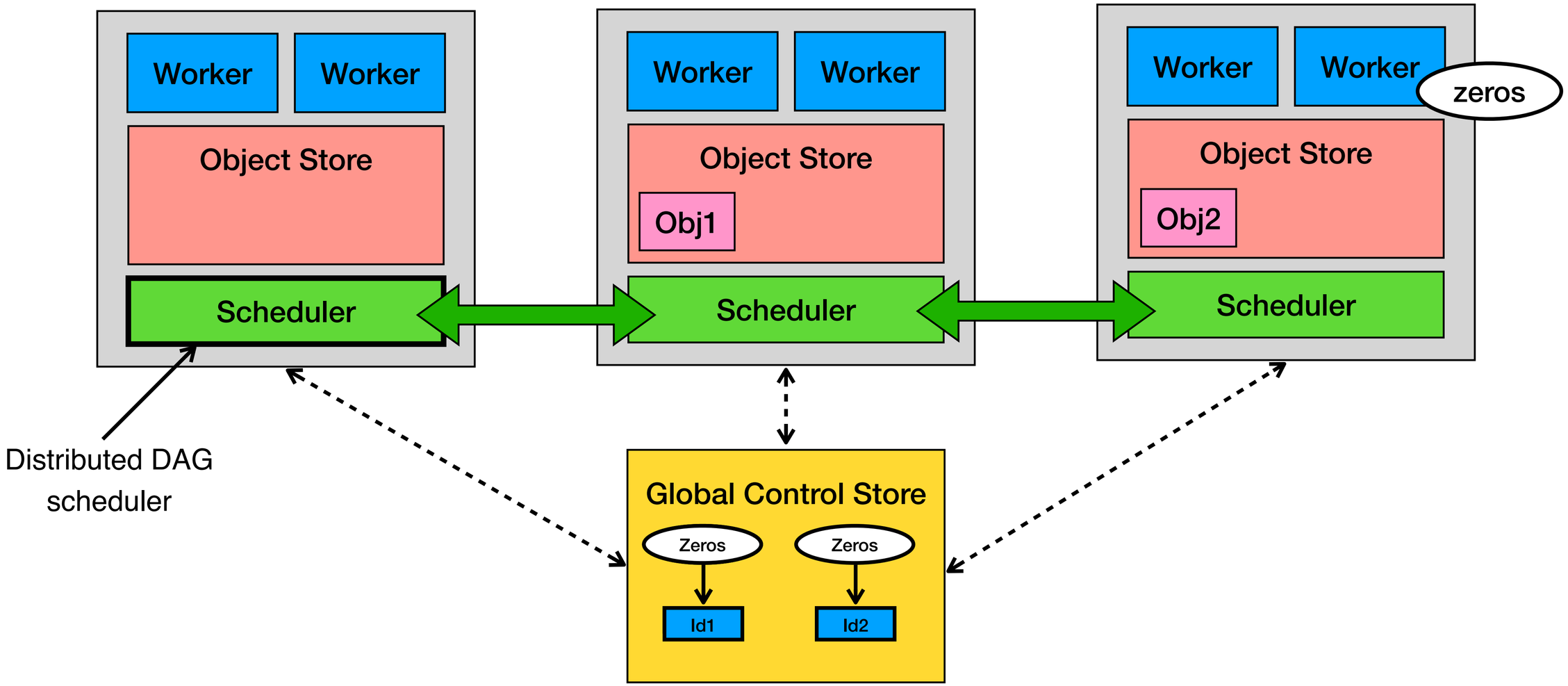
Architecture



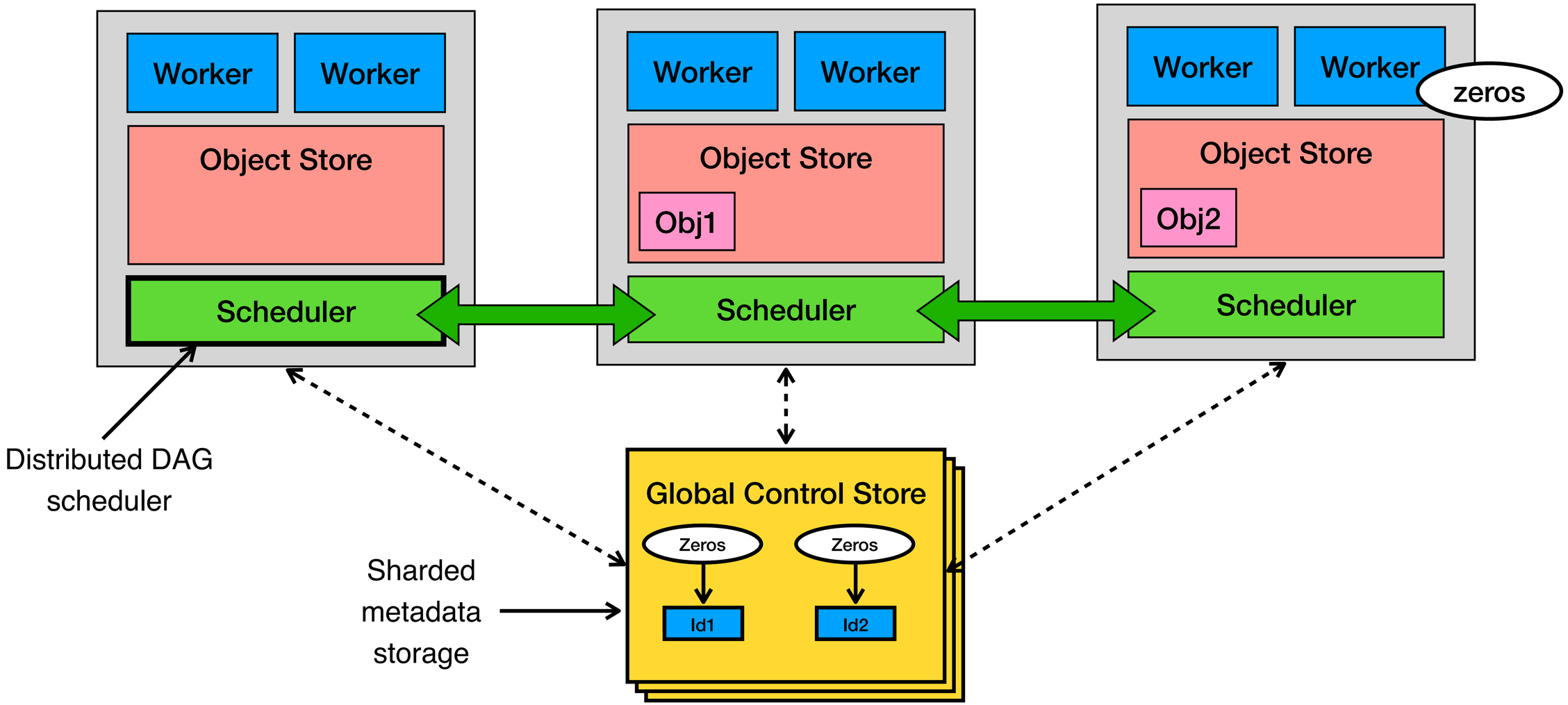
Architecture



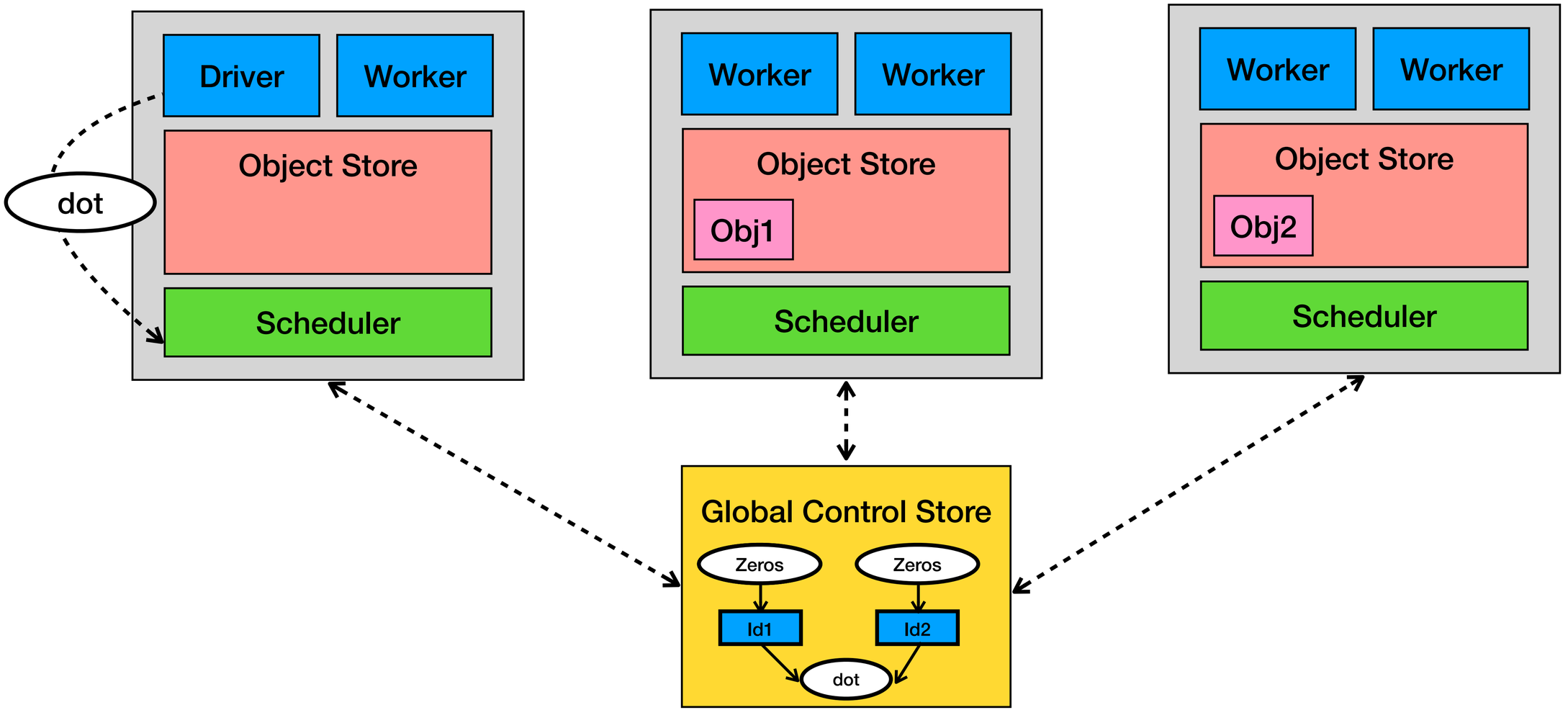
Architecture



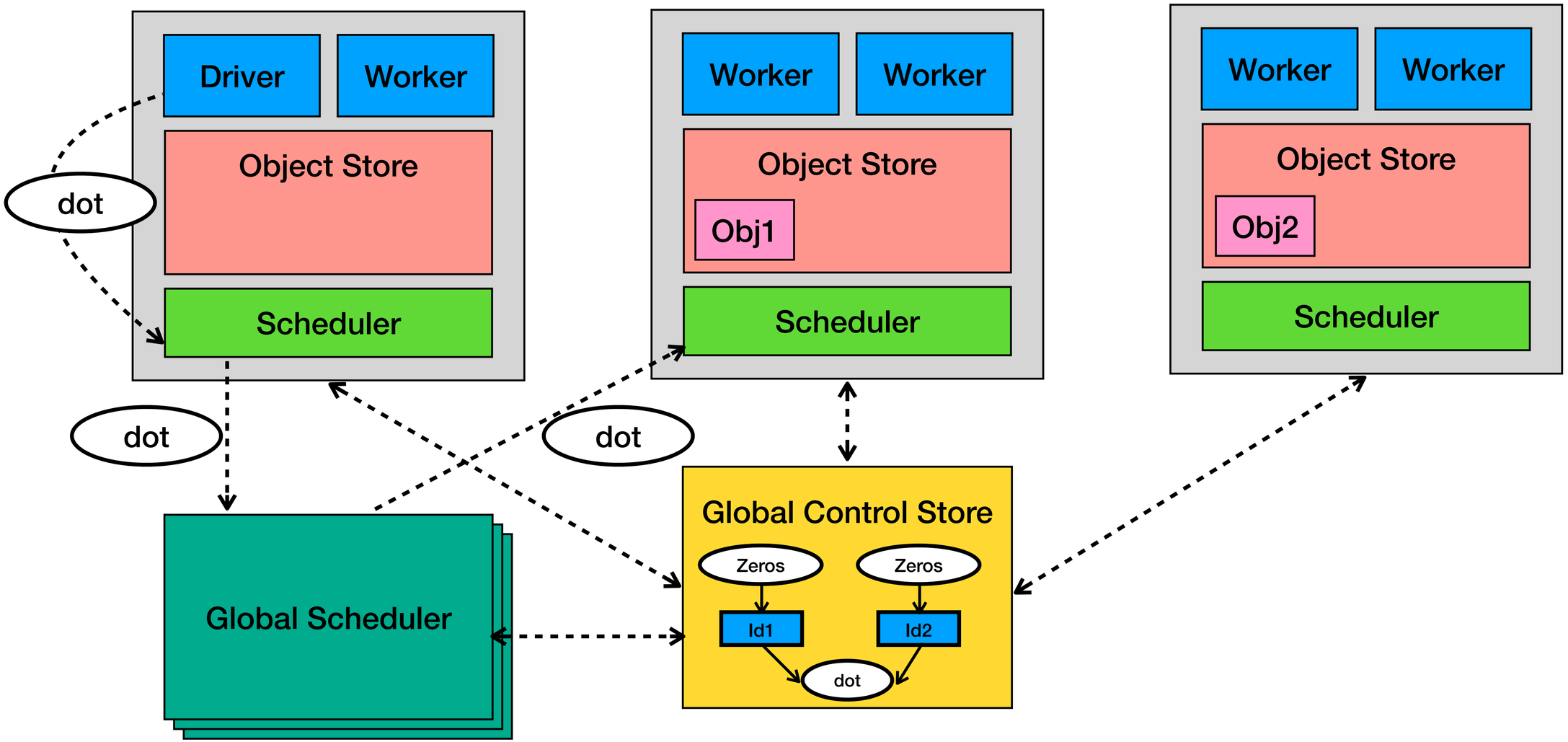
Architecture



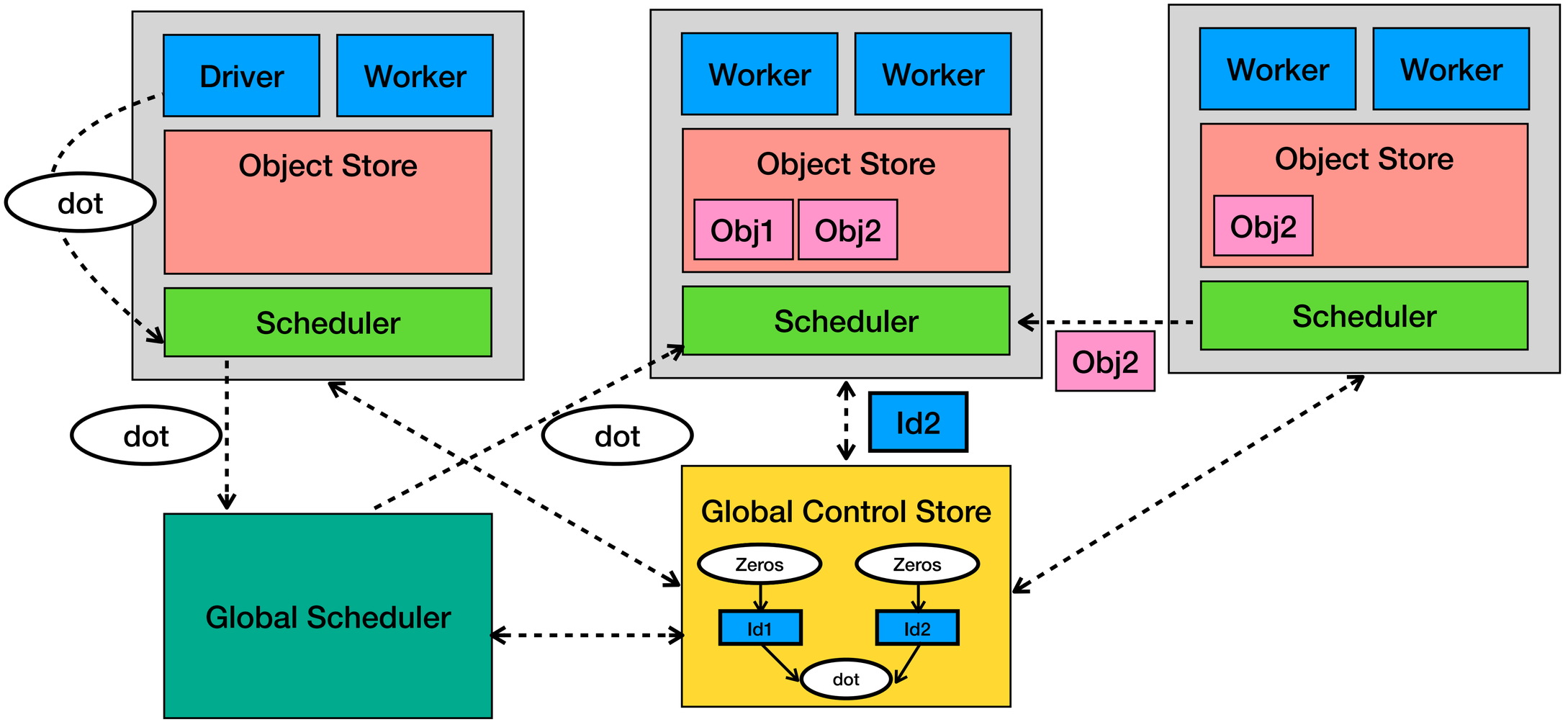
Architecture



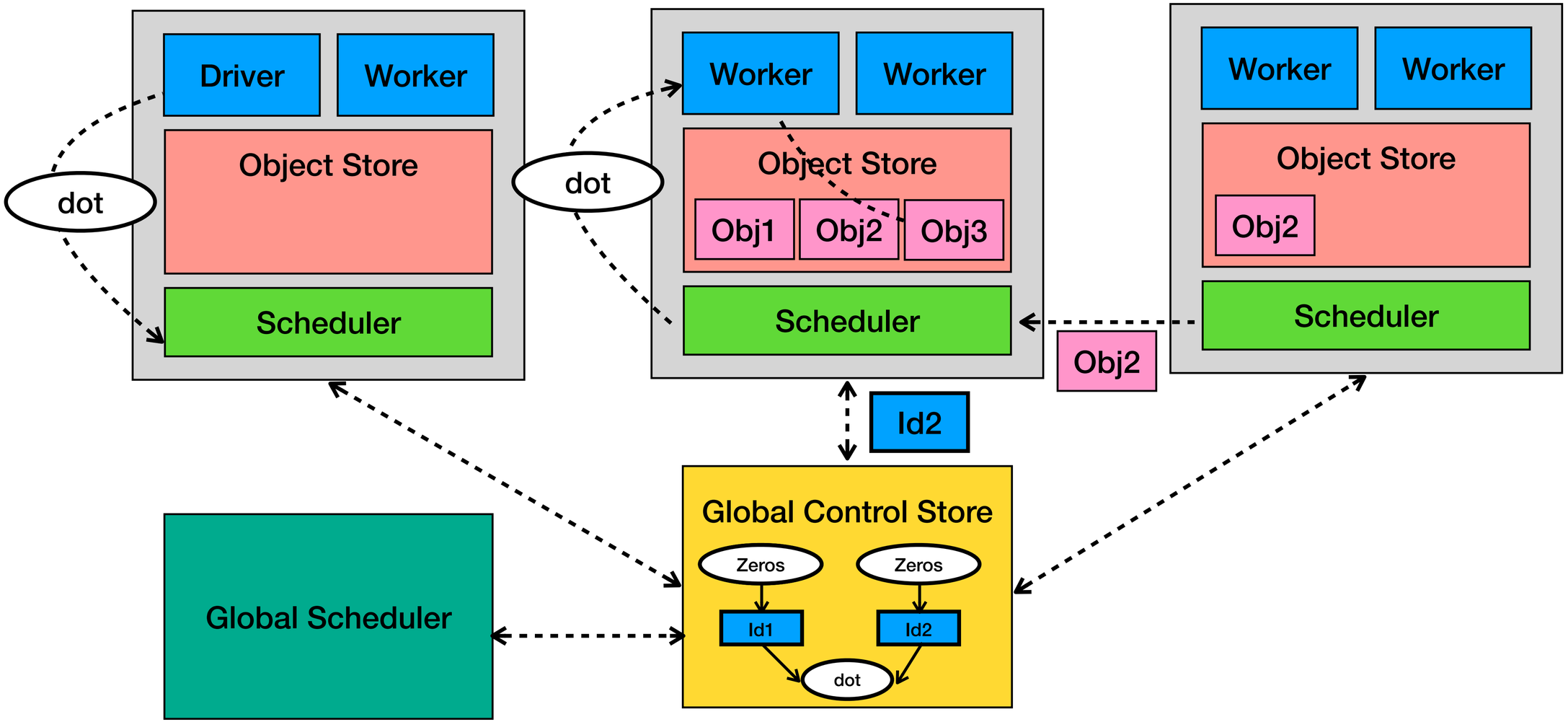
Architecture



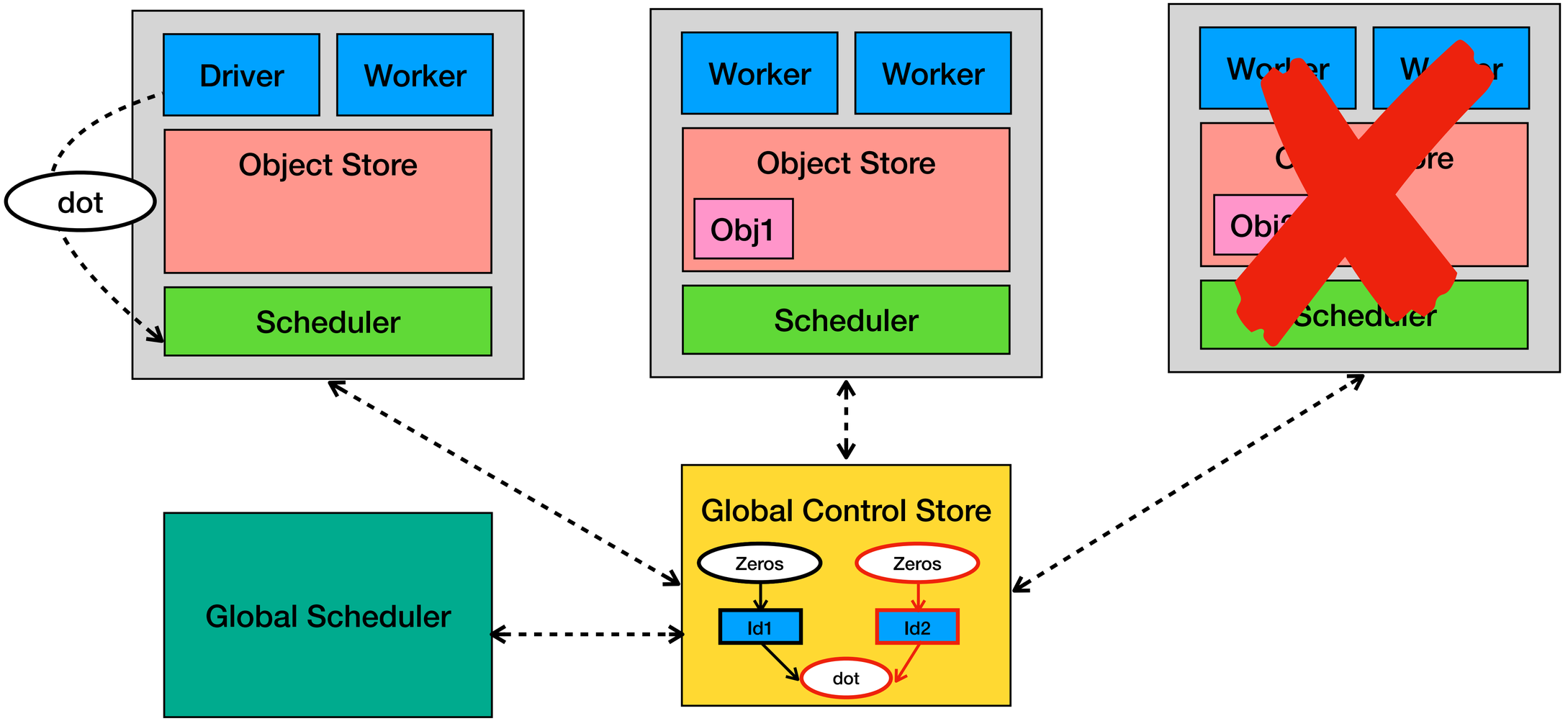
Architecture



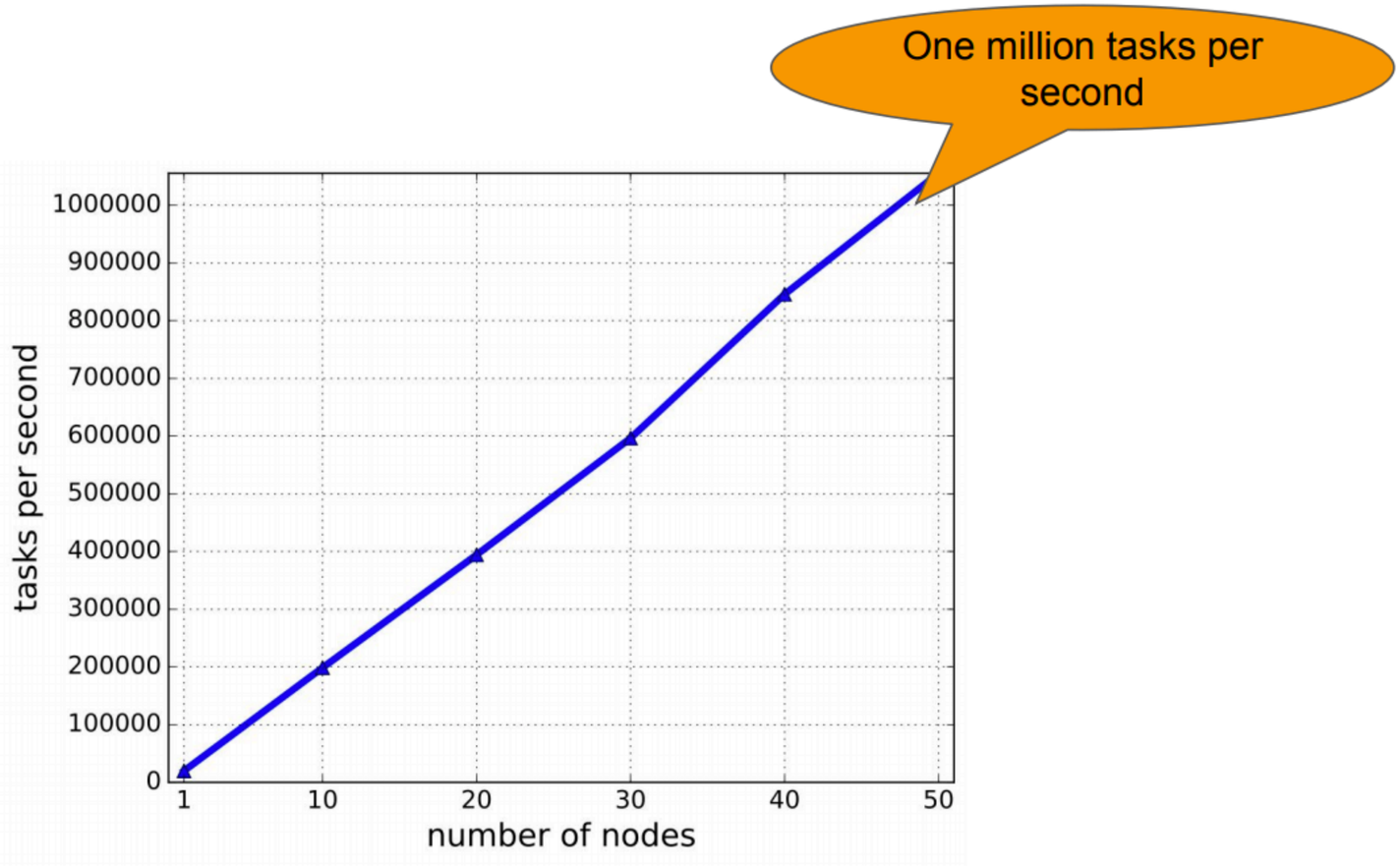
Architecture



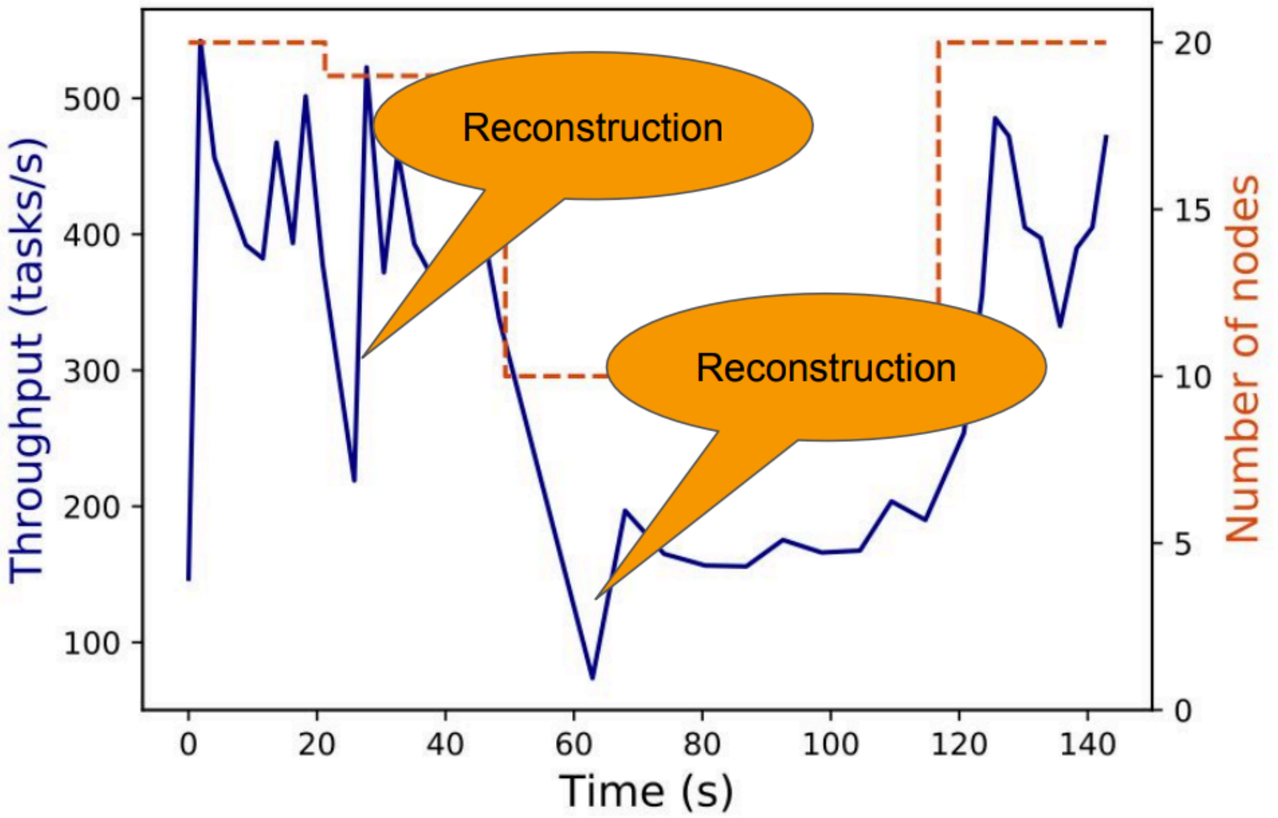
Fault Tolerance



Performance



Performance



Summary

- Ray unifies task-parallel and actor programming models
- Global control store and a bottom-up distributed scheduler

Thanks