OPPerTune

Post-Deployment Configuration Tuning of Services Made Easy

Gagan Somashekar* [1,3], **Karan Tandon*** [2], Anush Kini[2], Chieh-Chun Chang[4], Petr Husak[4],
Ranjita Bhagwan[2], Mayukh Das[3], Anshul Gandhi[1], Nagarajan Natarajan[2]

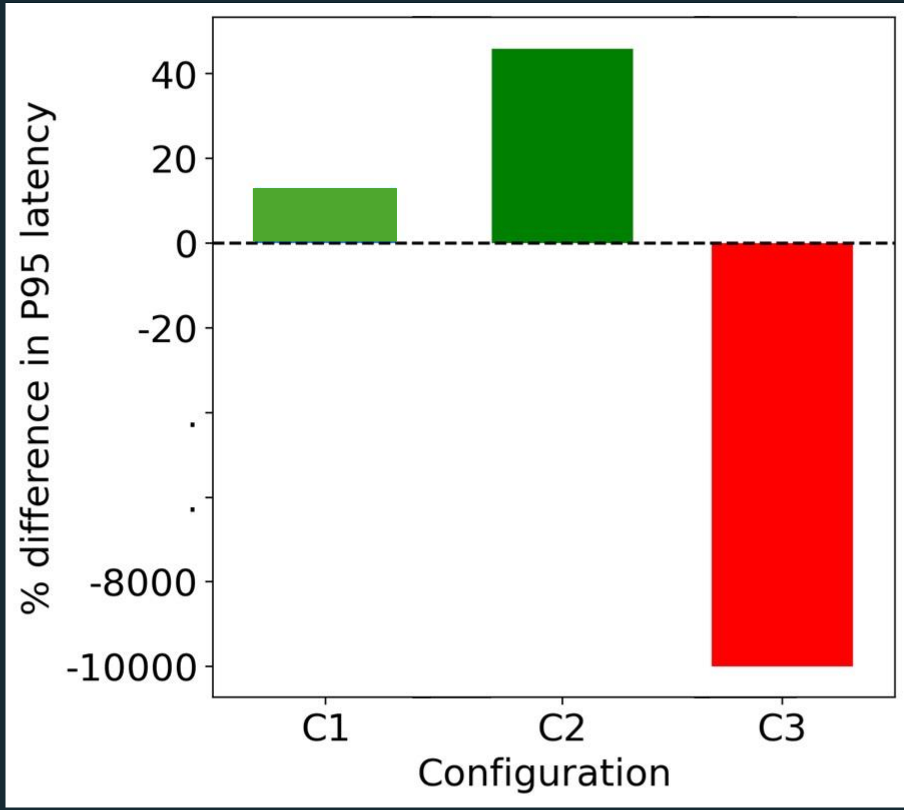
1 PACE Stony Brook University 2 Microsoft Research 3 Microsoft 365 4 Microsoft

Motivation

- Application performance depends on its configuration

Motivation

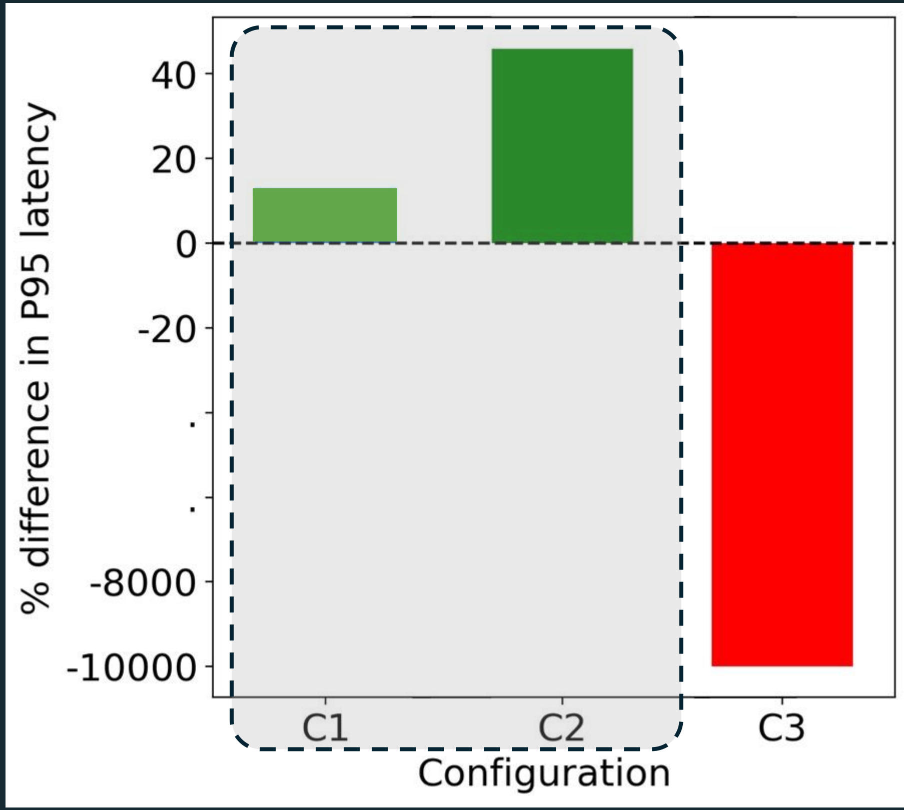
- Application performance depends on its configuration



Microservices-based cloud app

Motivation

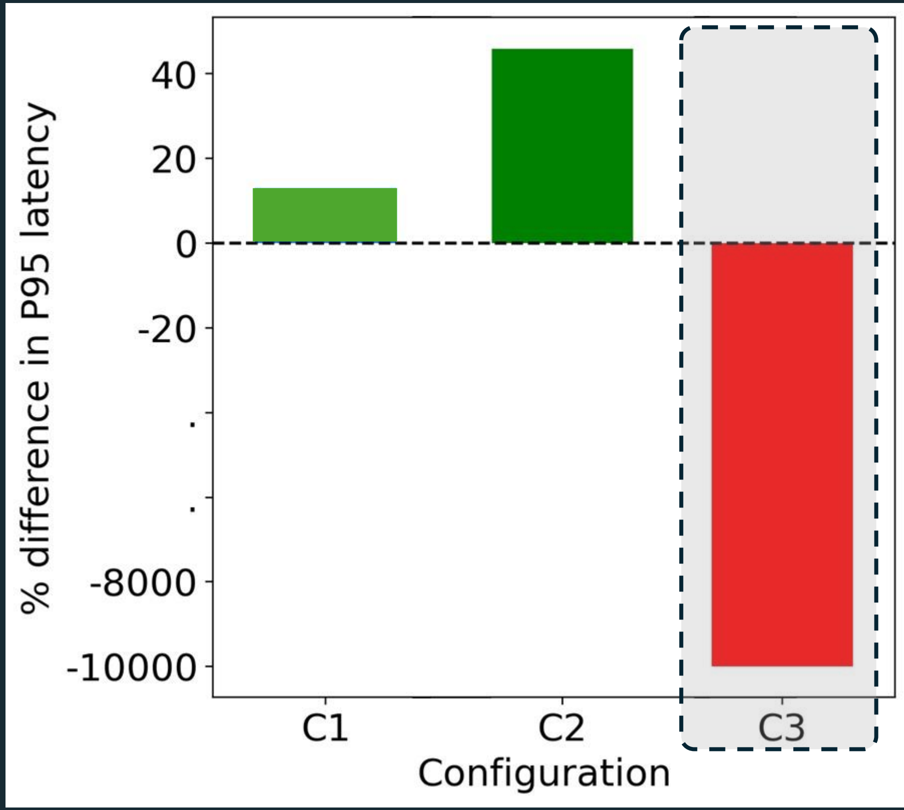
- Application performance depends on its configuration



Microservices-based cloud app

Motivation

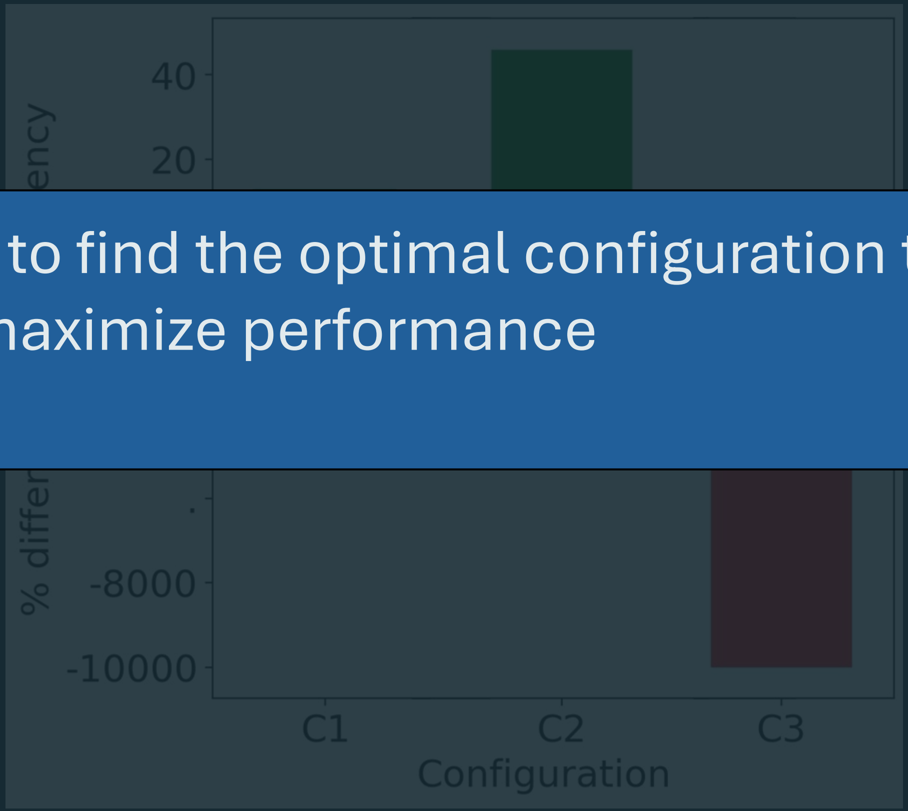
- Application performance depends on its configuration



Microservices-based cloud app

Motivation

- Application performance depends on its configuration

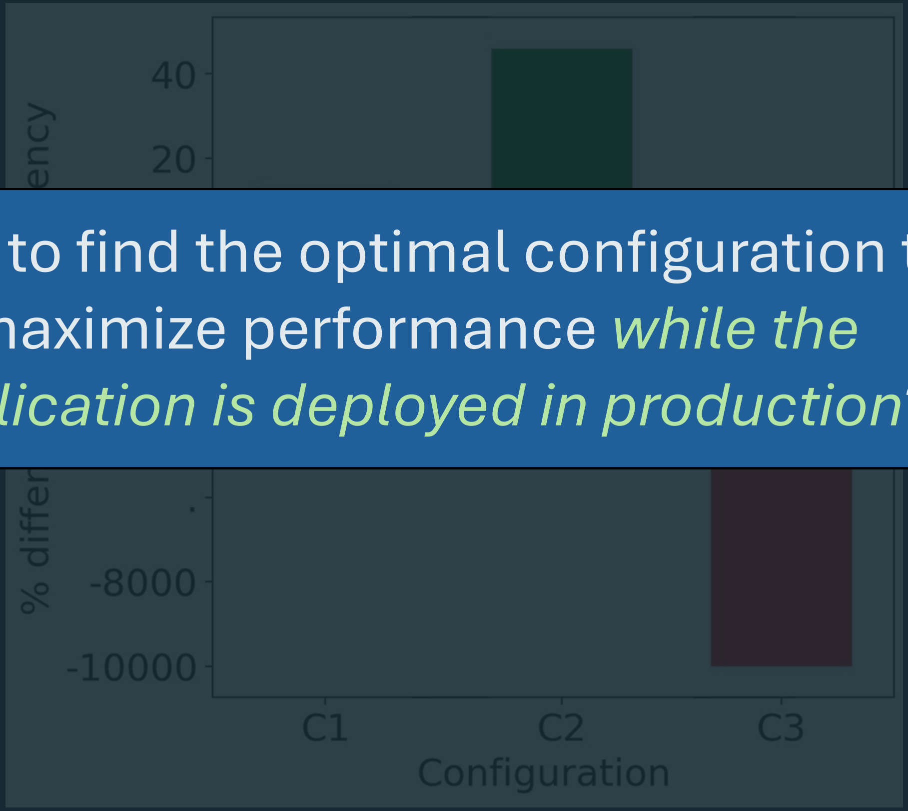


Microservices-based cloud app

How to find the optimal configuration to maximize performance

Motivation

- Application performance depends on its configuration



Microservices-based cloud app

How to find the optimal configuration to maximize performance *while the application is deployed in production*?

Motivation

- Very large configuration space
  - N = Number of microservices
  - P = Parameters per microservice
  - C = Number of parameter values
  - **Total possible configurations** ≈ C^{N*P}

Motivation

- Very large configuration space
  - N = Number of microservices
  - P = Parameters per microservice
  - C = Number of parameter values
  - **Total possible c**

How to find the optimal configuration in minimum steps?

Need for a new framework

	CherryPick (NSDI '17)	µTune (OSDI '18)	OPTIMUS CLOUD (ATC '20)	KEA (SIGMOD'21)	SelfTune (NSDI '23)
Handles numerical and categorical parameters	✓	✓	✓	✓	X
Can scope the problem	X	X	X	X	X
Filter parameters to tune	X	X	✓	X	X
Low Sample Complexity	✓	X	X	X	✓
Application-independent	✓	✓	✓	X	✓
Supports online learning	✓	X	X	✓	✓
Handles dynamic objective function	X	✓	✓	✓	✓
End-to-end framework*	X	X	✓	✓	X

Need for a new framework

	CherryPick (NSDI '17)	µTune (OSDI '18)	OPTIMUS CLOUD (ATC '20)	KEA (SIGMOD'21)	SelfTune (NSDI '23)
Handles numerical and categorical parameters	✓	✓	✓	✓	X
Can scope the problem	X	X	X	X	X
Filter parameters to tune	X	X	✓	X	X
Low Sample Complexity	✓	X	X	X	✓
Application-independent	✓	✓	✓	X	✓
Supports online learning	✓	X	X	✓	✓
Handles dynamic objective function	X	✓	✓	✓	✓
End-to-end framework*	X	X	✓	✓	X

Need for a new framework

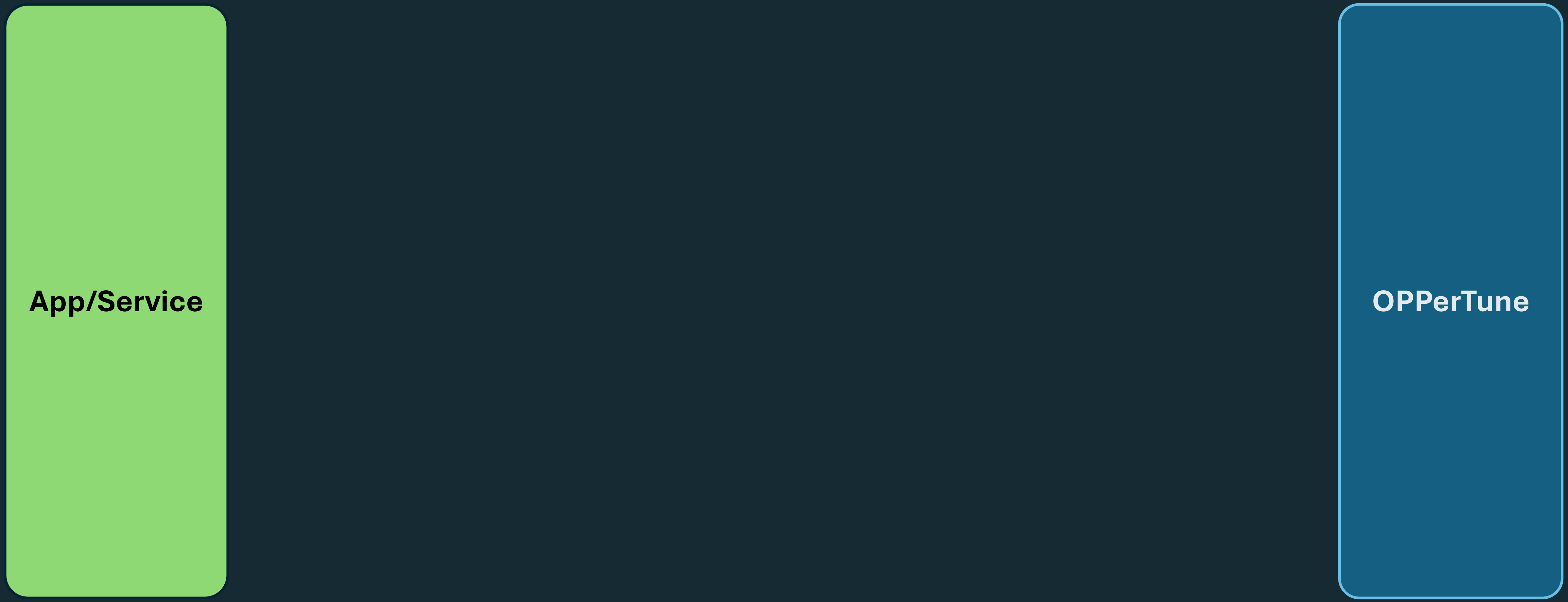
	CherryPick (NSDI '17)	µTune (OSDI '18)	OPTIMUS CLOUD (ATC '20)	KEA (SIGMOD'21)	SelfTune (NSDI '23)	OPPerTune
Handles numerical and categorical parameters	✓	✓	✓	✓	X	✓
Can scope the problem	X	X	X	X	X	✓
Filter parameters to tune	X	X	✓	X	X	✓
Low Sample Complexity	✓	X	X	X	✓	✓
Application-independent	✓	✓	✓	X	✓	✓
Supports online learning	✓	X	X	✓	✓	✓
Handles dynamic objective function	X	✓	✓	✓	✓	✓
End-to-end framework*	X	X	✓	✓	X	✓

OPPerTune (Optimal Post-deployment Performance Tuner)

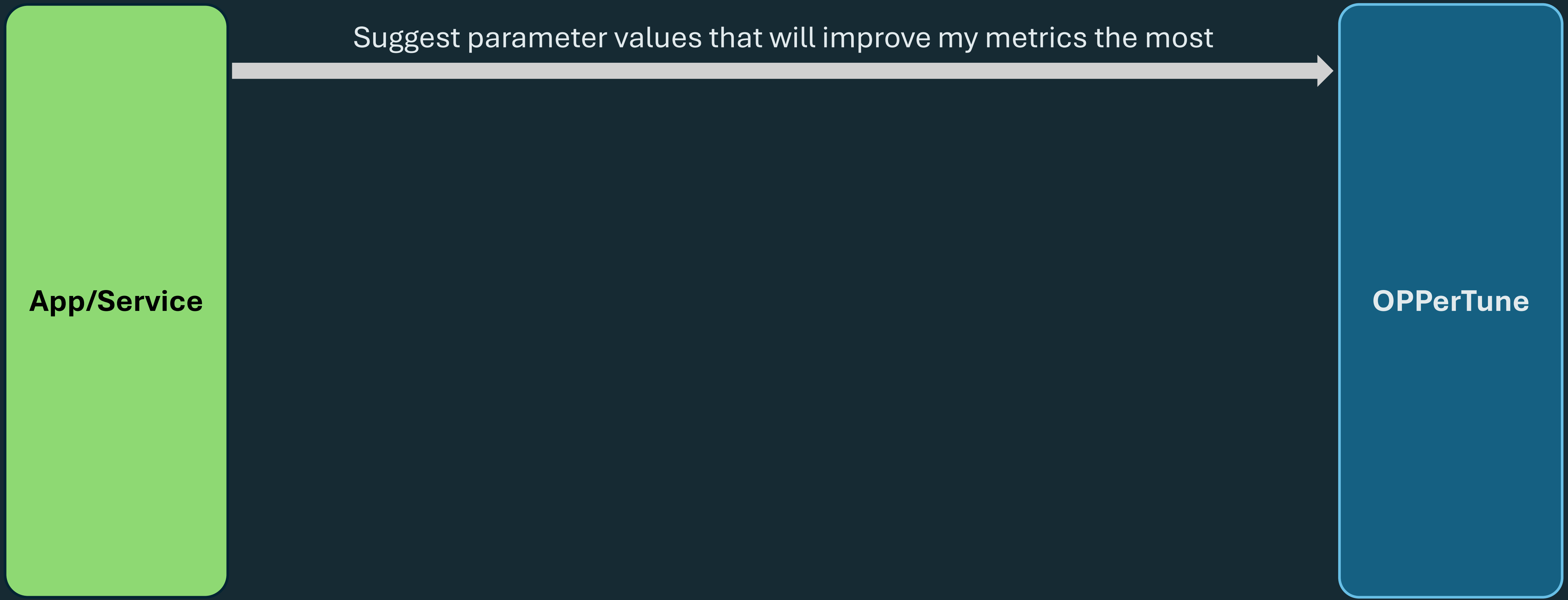
OPPerTune (Optimal Post-deployment Performance Tuner)

App/Service

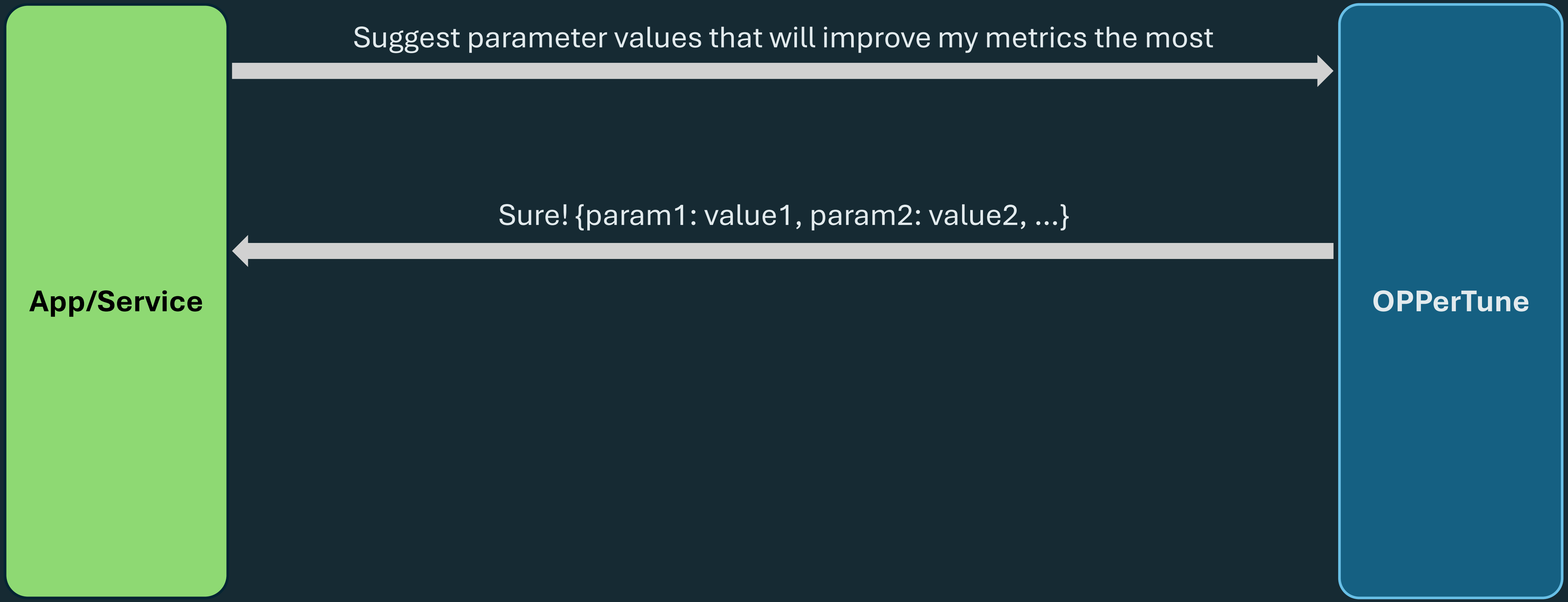
OPPerTune (Optimal Post-deployment Performance Tuner)



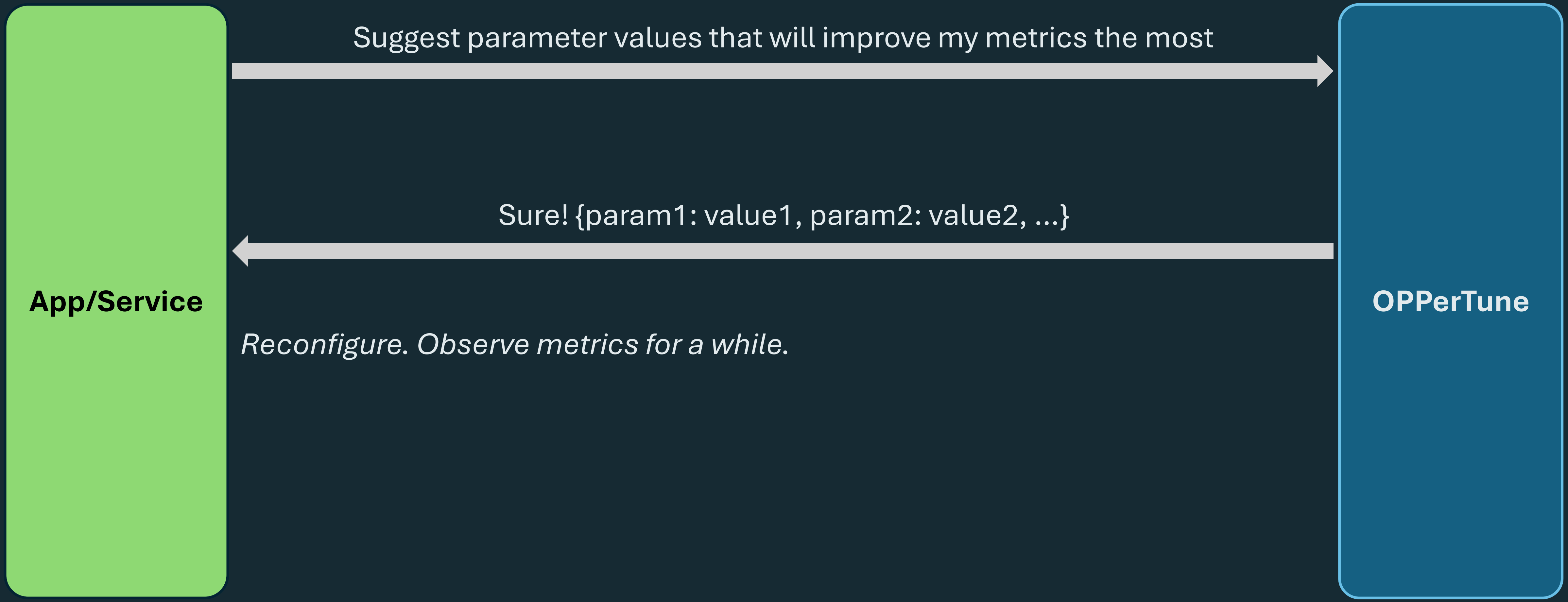
OPPerTune (Optimal Post-deployment Performance Tuner)



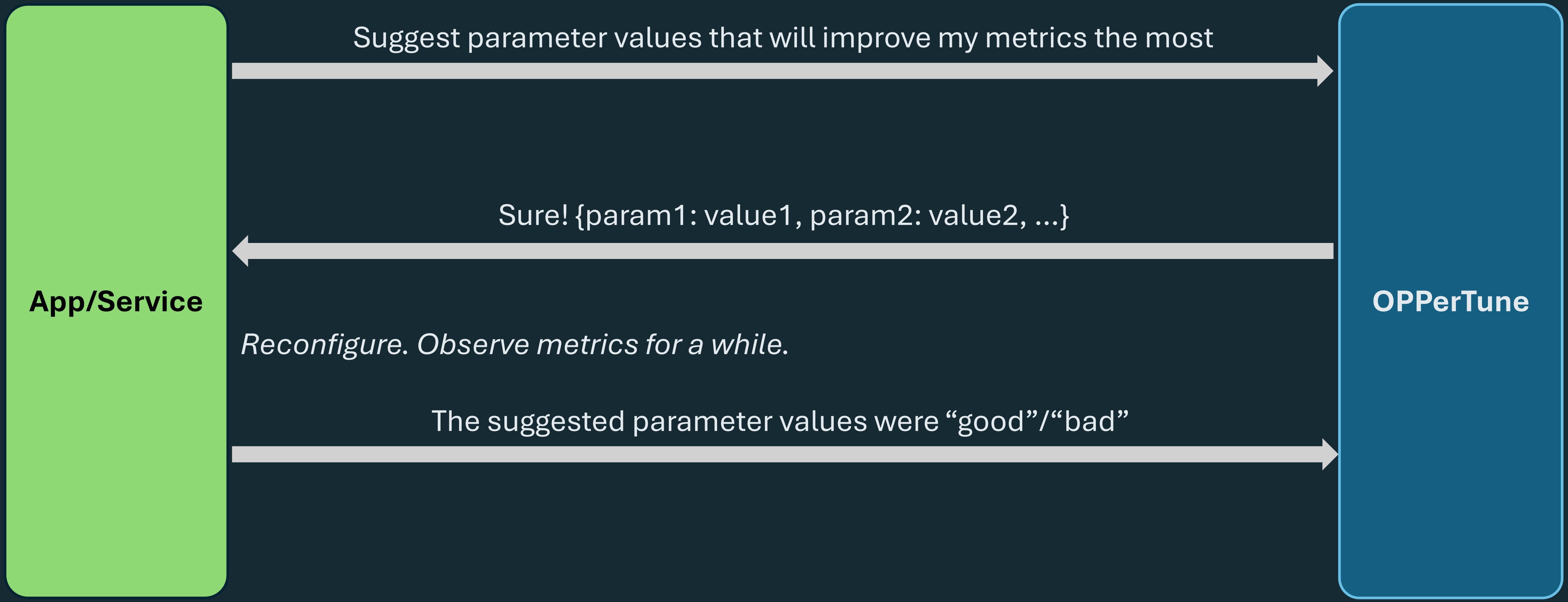
OPPerTune (Optimal Post-deployment Performance Tuner)



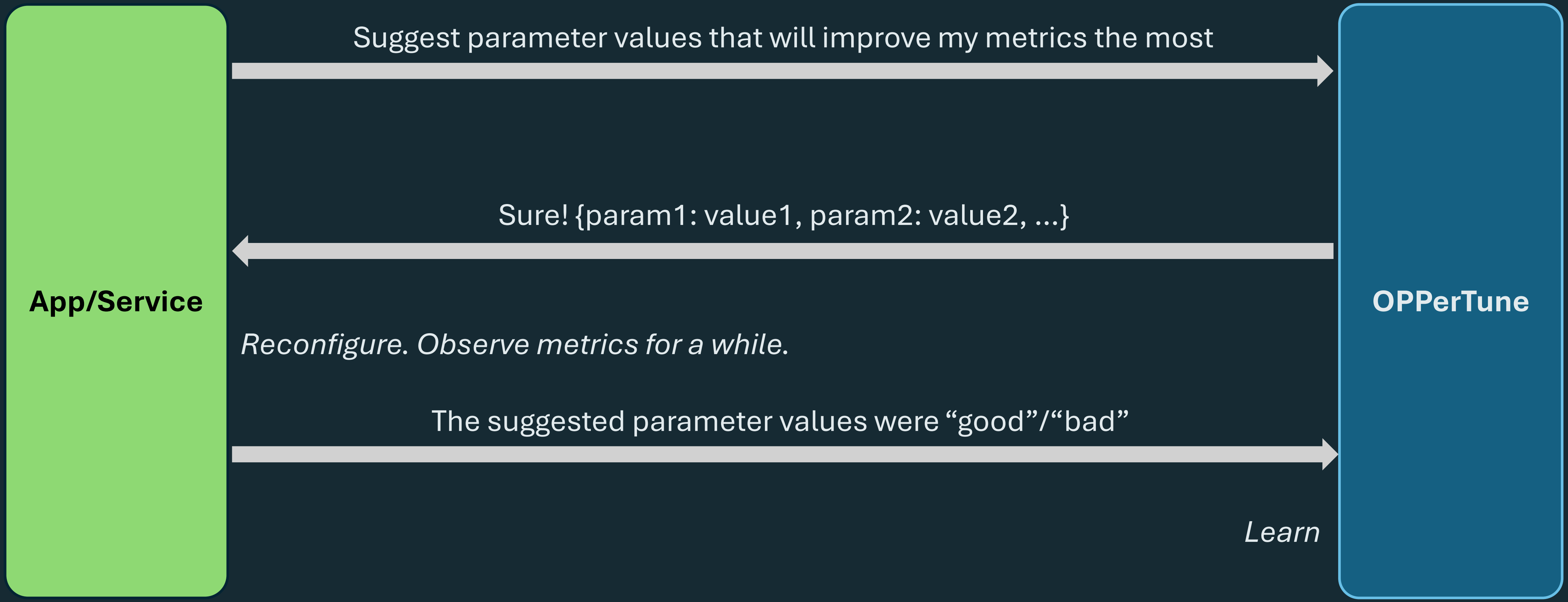
OPPerTune (Optimal Post-deployment Performance Tuner)



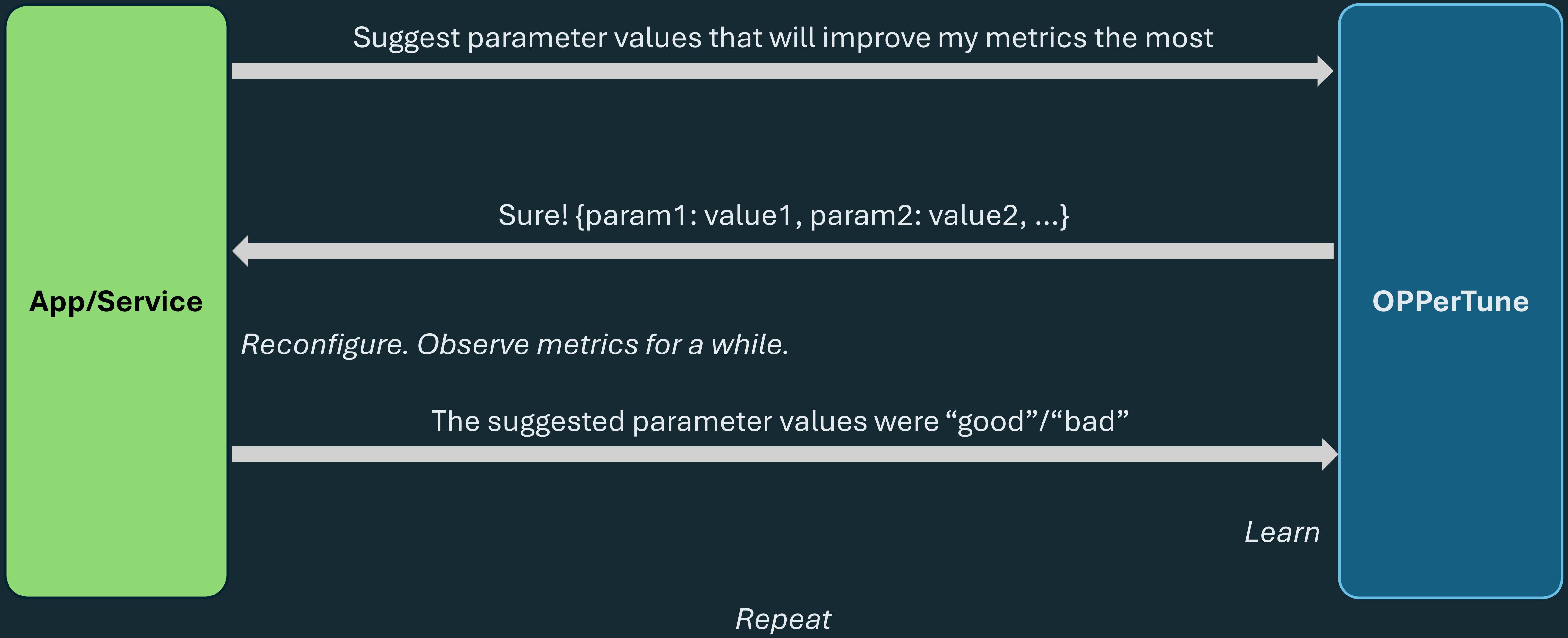
OPPerTune (Optimal Post-deployment Performance Tuner)



OPPerTune (Optimal Post-deployment Performance Tuner)

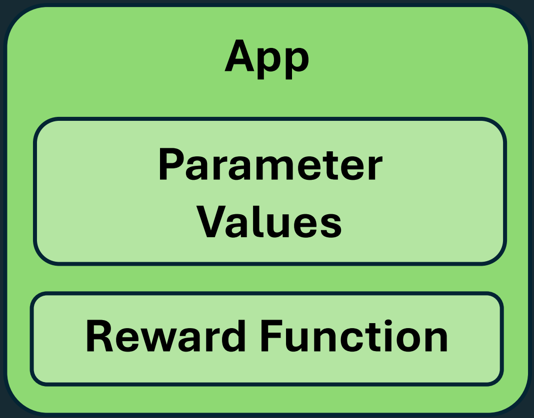


OPPerTune (Optimal Post-deployment Performance Tuner)

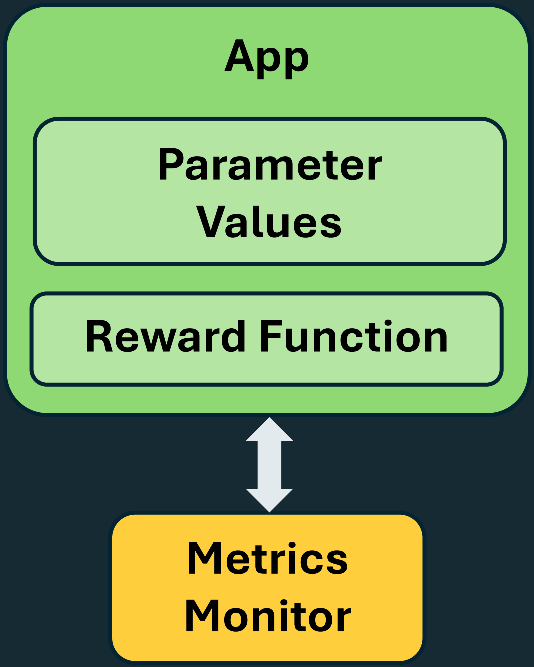


OPPerTune – Tuning Iteration

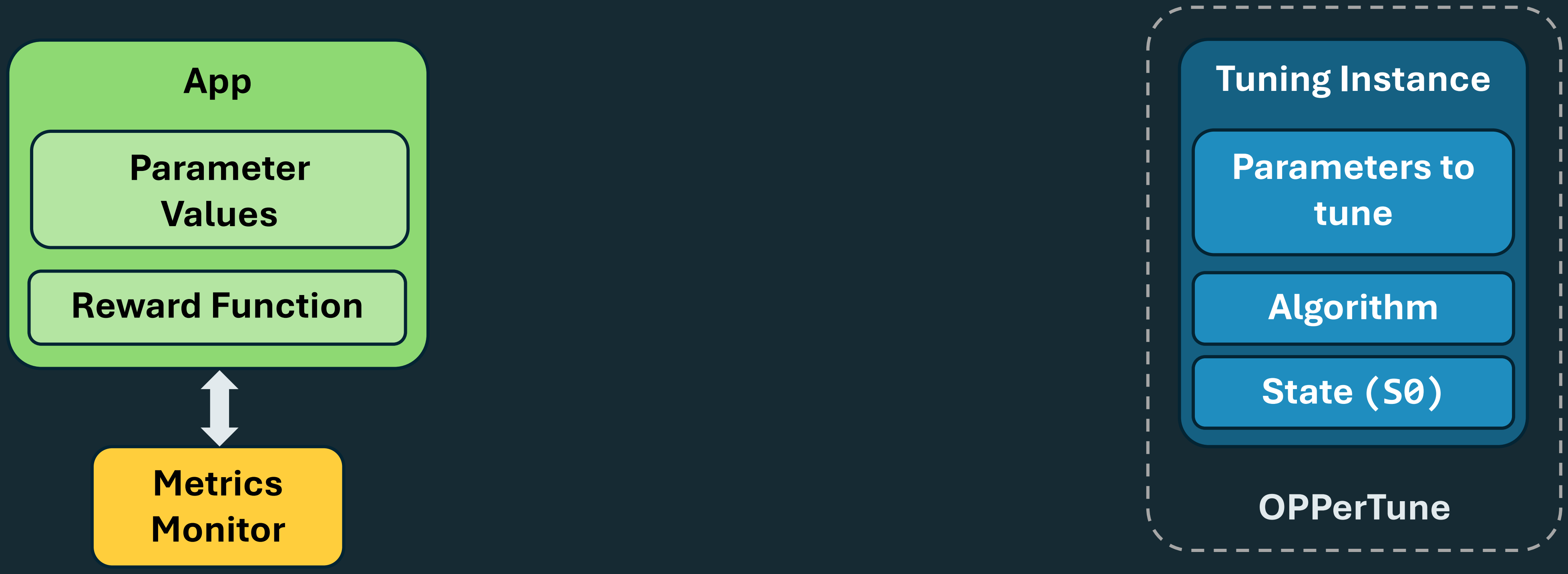
OPPerTune – Tuning Iteration



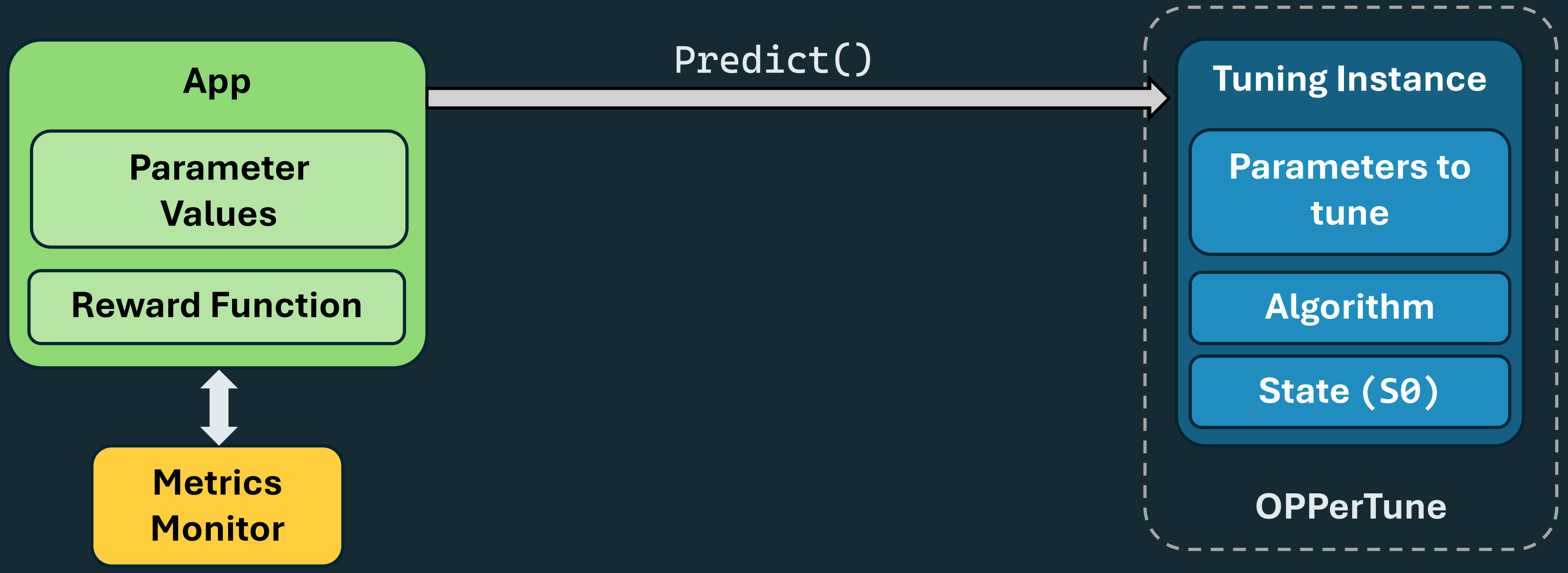
OPPerTune – Tuning Iteration



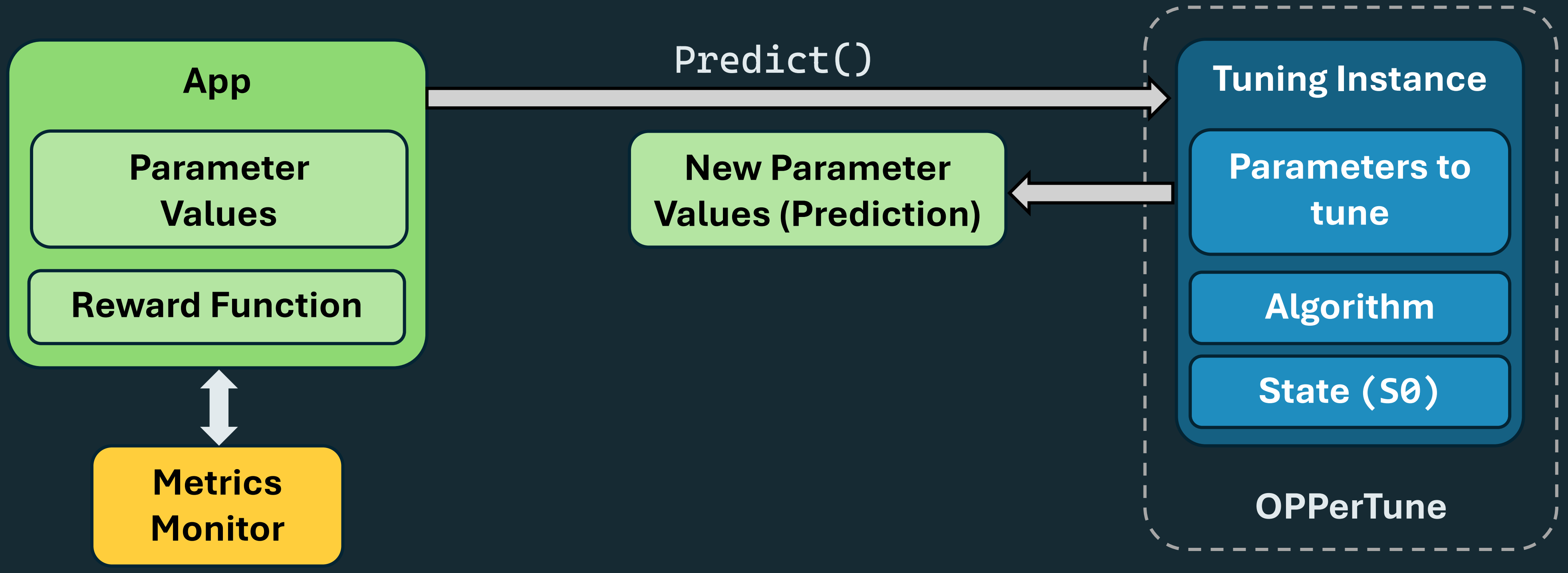
OPPerTune – Tuning Iteration



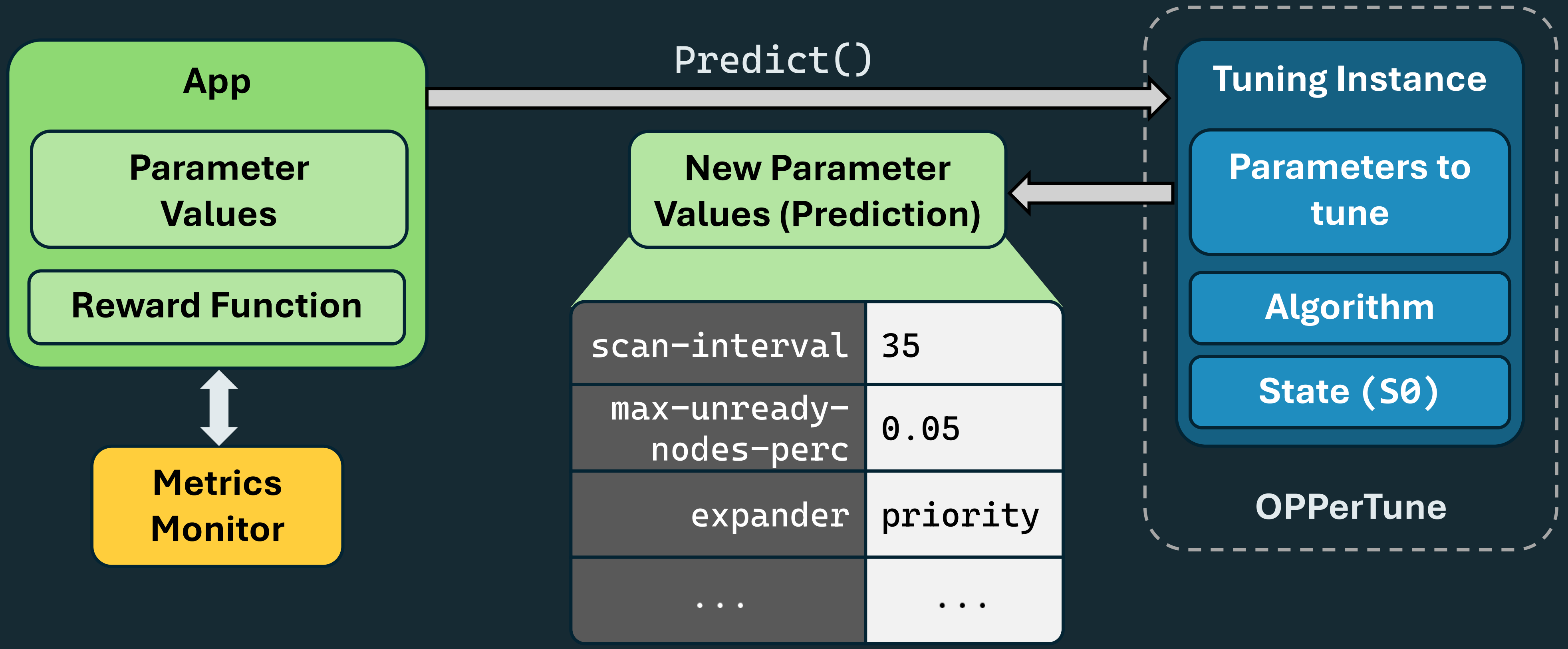
OPPerTune – Tuning Iteration



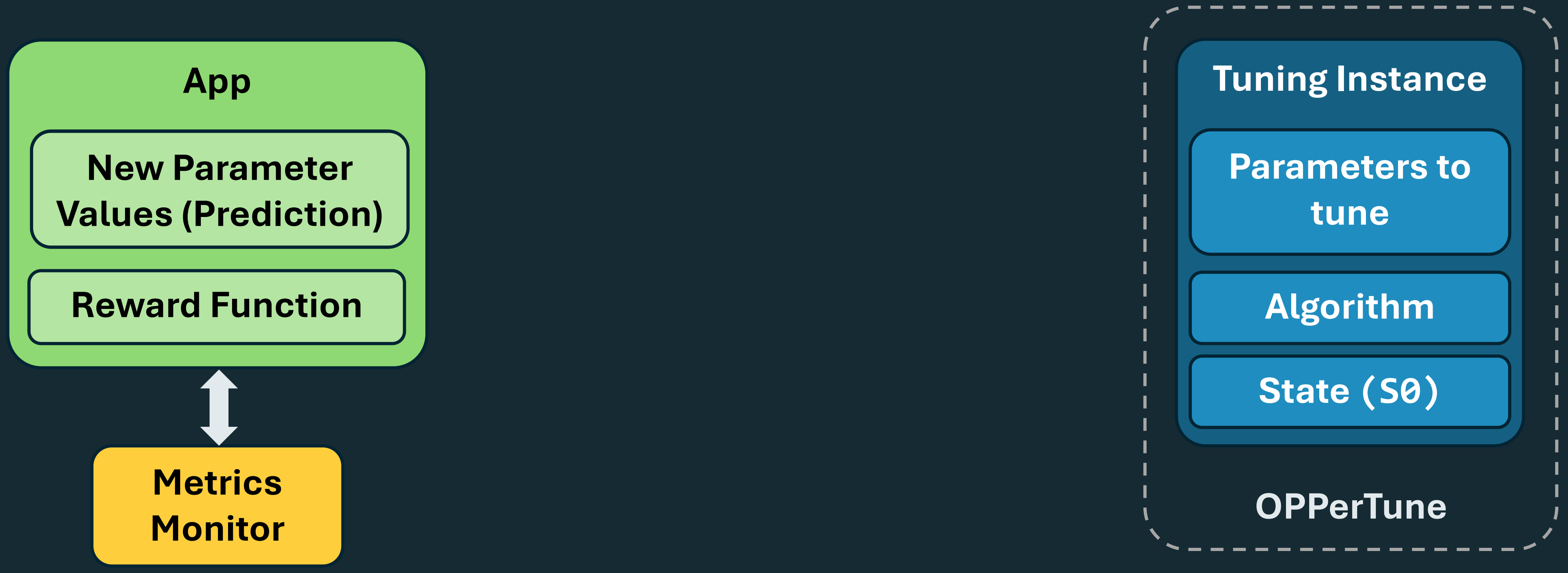
OPPerTune – Tuning Iteration



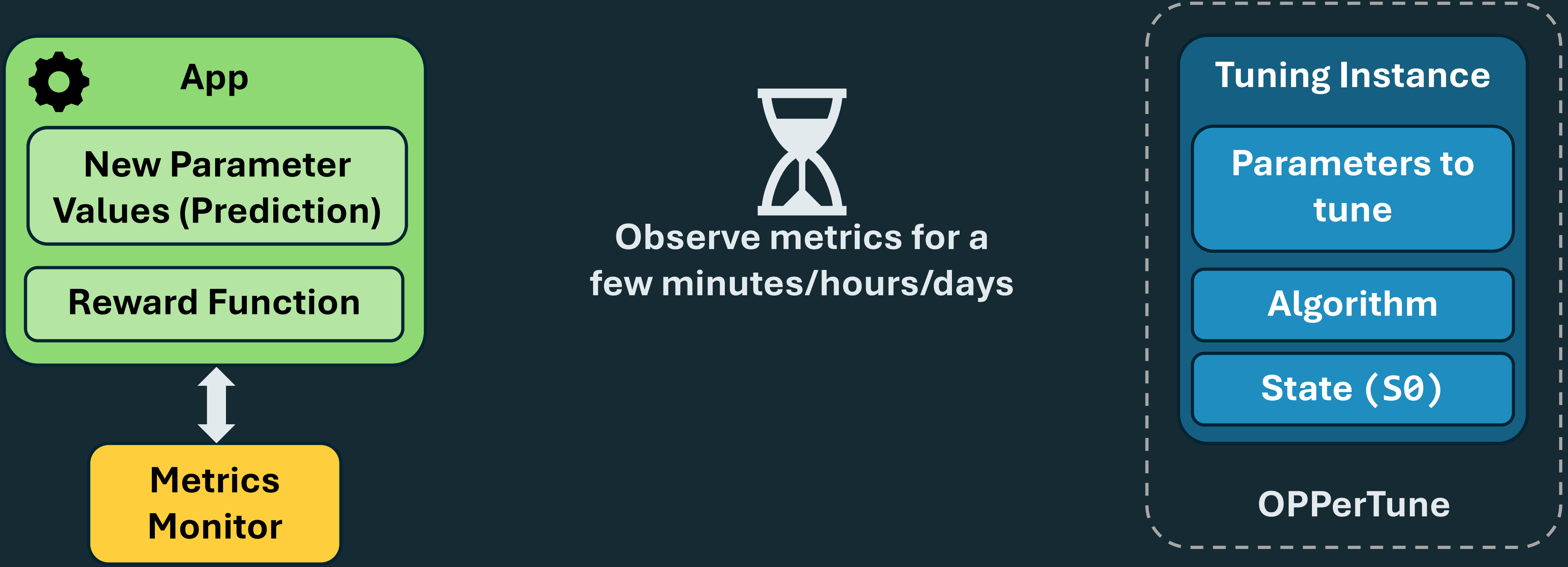
OPPerTune – Tuning Iteration



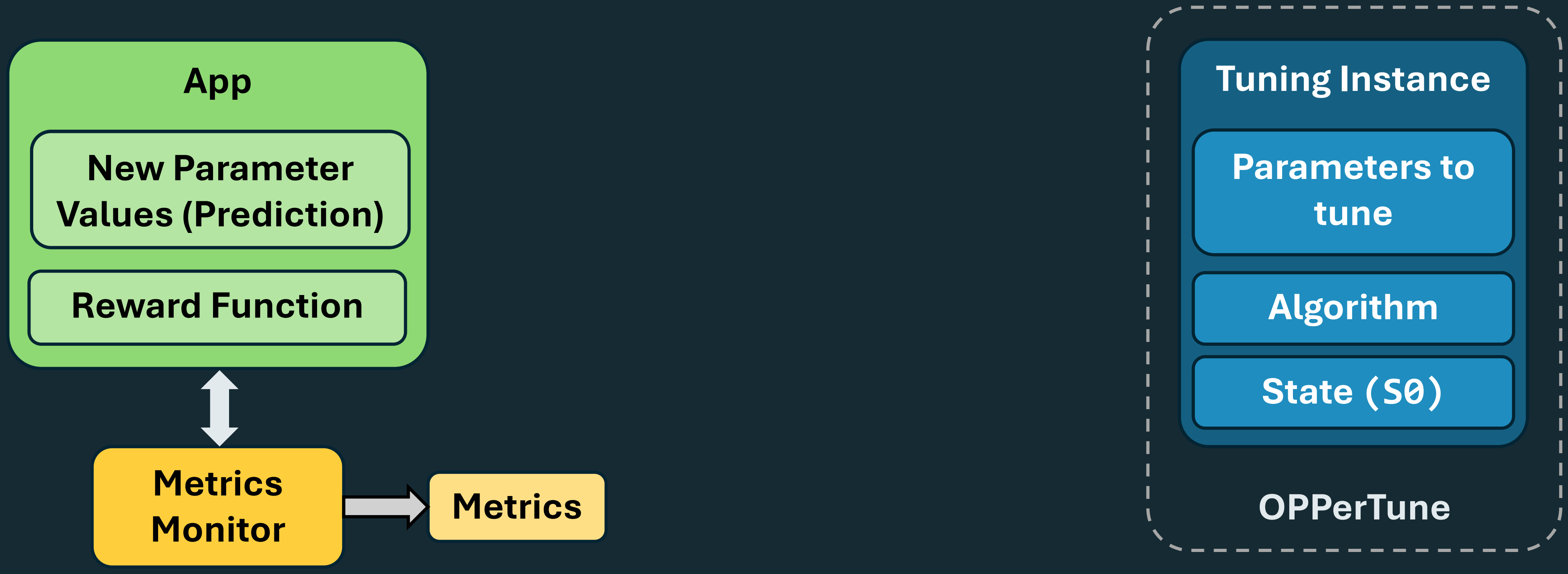
OPPerTune – Tuning Iteration



OPPerTune – Tuning Iteration

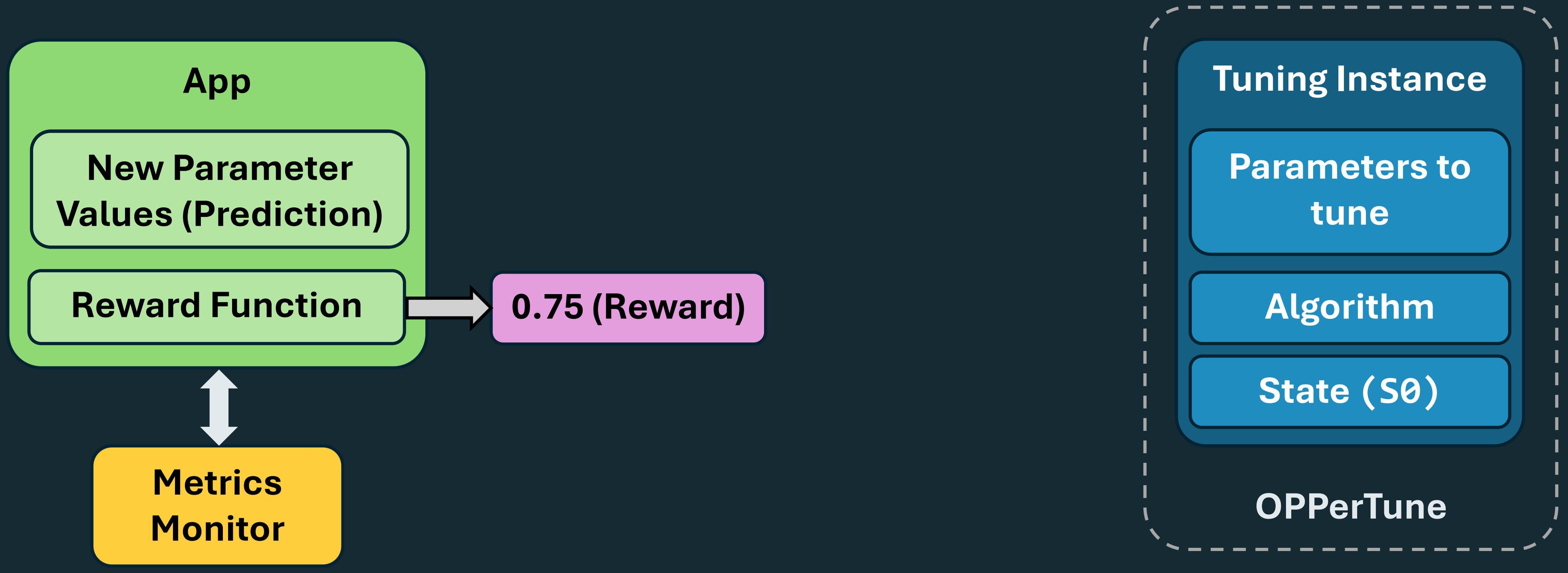


OPPerTune – Tuning Iteration



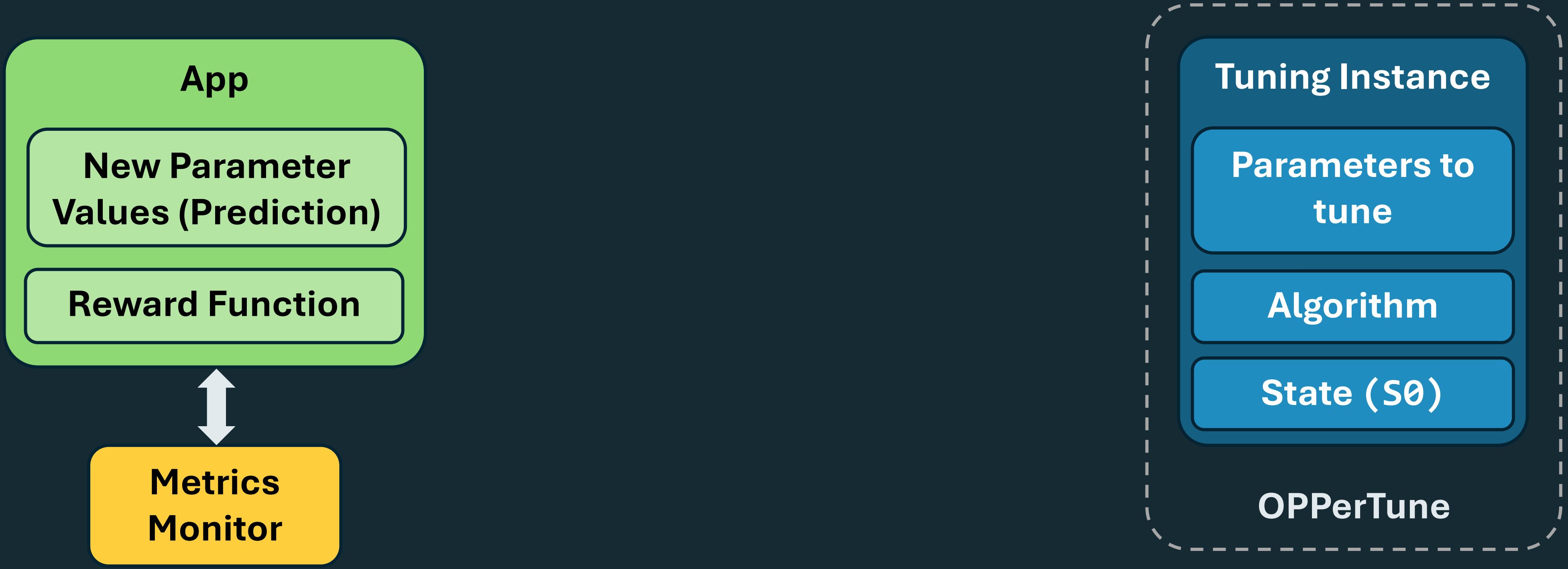
OPPerTune – Tuning Iteration

OPPerTune – Tuning Iteration

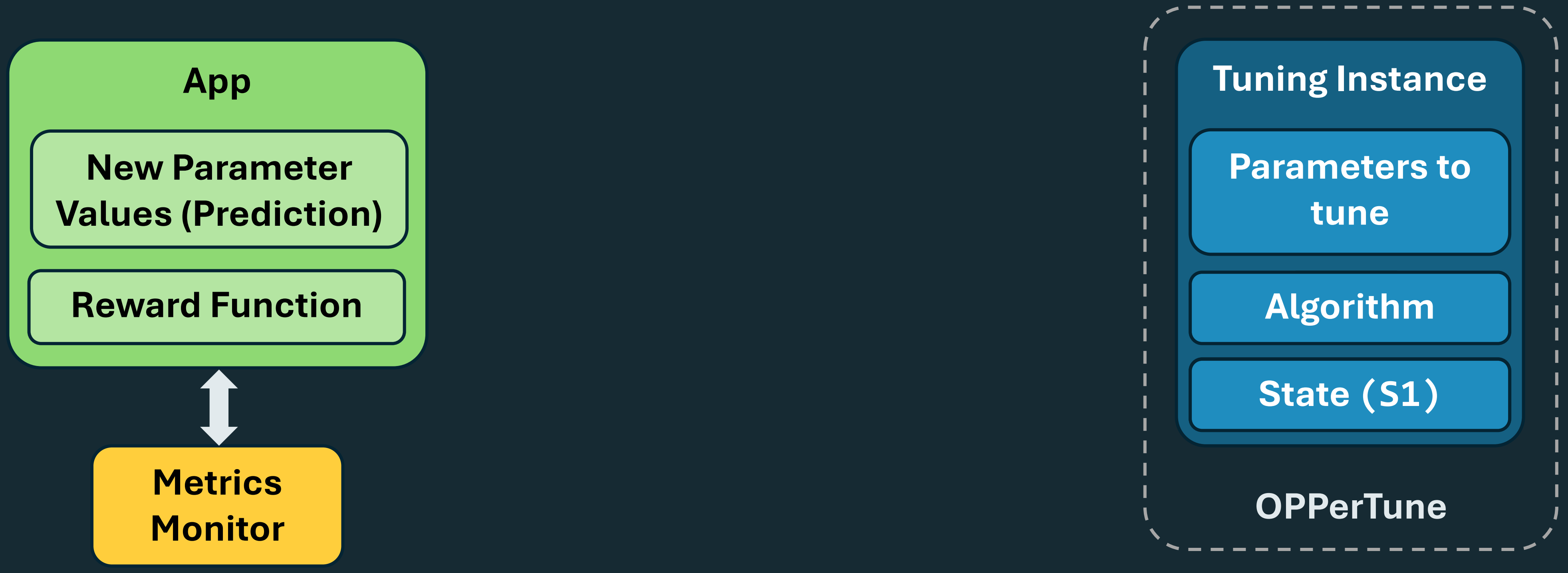


OPPerTune – Tuning Iteration

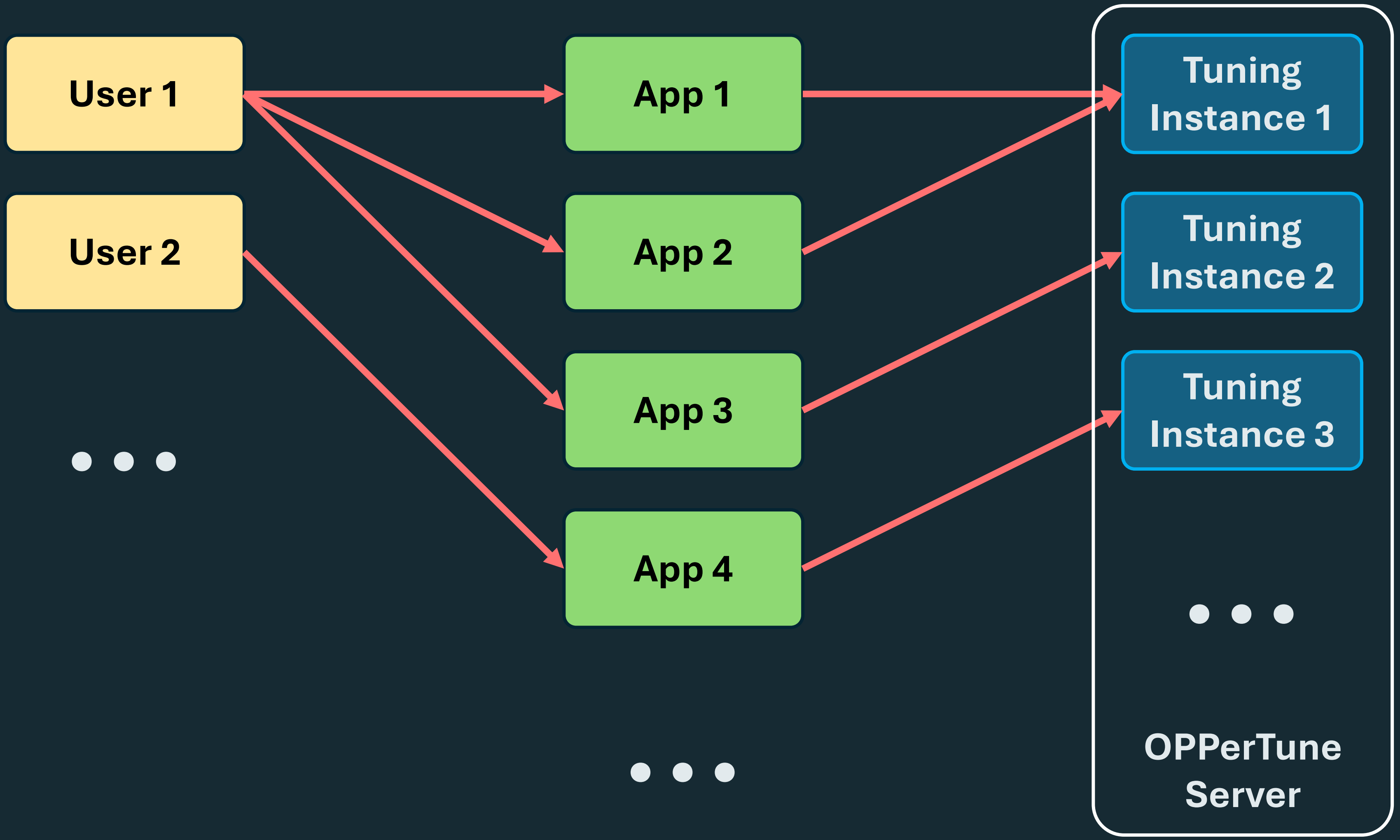
OPPerTune – Tuning Iteration



OPPerTune – Tuning Iteration



Tuning at scale



Challenge 1
Hybrid parameter space

Hybrid parameter space

- Jointly tuning numerical and categorical parameters for optimal performance

Hybrid parameter space

- Jointly tuning numerical and categorical parameters for optimal performance

What about existing algorithms?

Hybrid parameter space

- Jointly tuning numerical and categorical parameters for optimal performance

What about existing algorithms?

- Gradient-based algorithms: Inapplicable due to lack of continuity

Hybrid parameter space

- Jointly tuning numerical and categorical parameters for optimal performance

What about existing algorithms?

- Gradient-based algorithms: Inapplicable due to lack of continuity
- Traditional bandit algorithms & Deep RL techniques: Sample-inefficient

Hybrid parameter space

- Jointly tuning numerical and categorical parameters for optimal performance

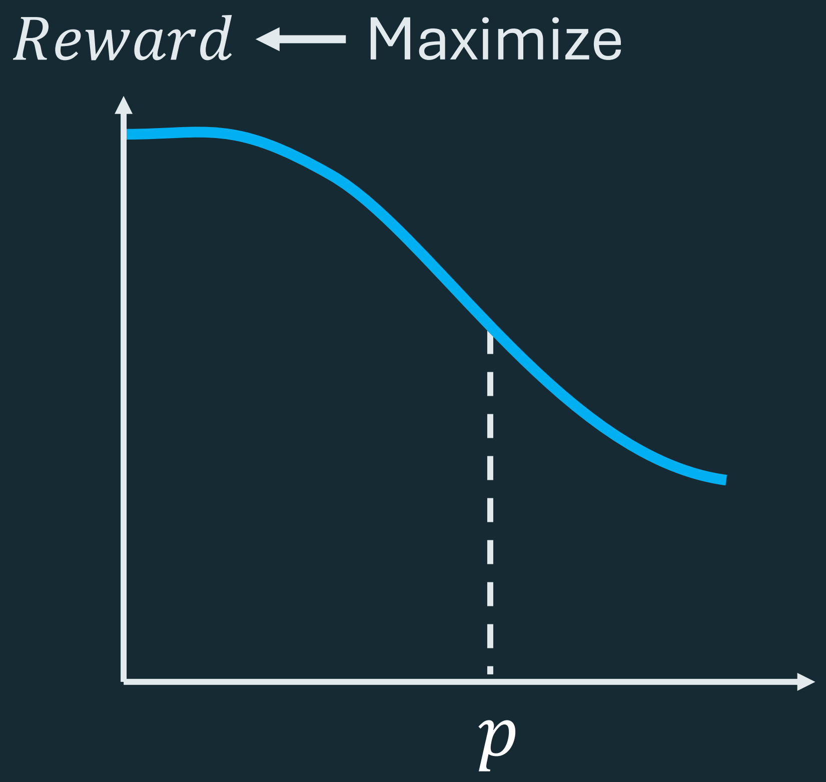
What about existing algorithms?

- Gradient-based algorithms: Inapplicable due to lack of continuity
- Traditional bandit algorithms & Deep RL techniques: Sample-inefficient
- Bayesian optimization: Needs **multiple** samples for the **same** reward function

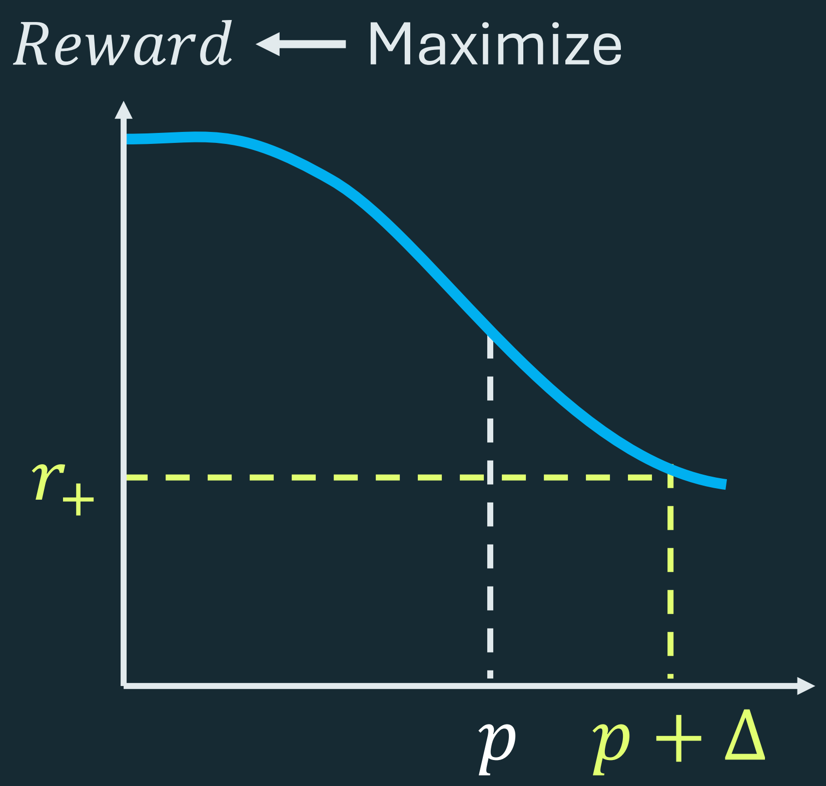
Our solution - HybridBandits

- Scales linearly with number of numerical parameters and total categorical combinations
- Combines two algorithms – one for numerical and another for categorical parameters

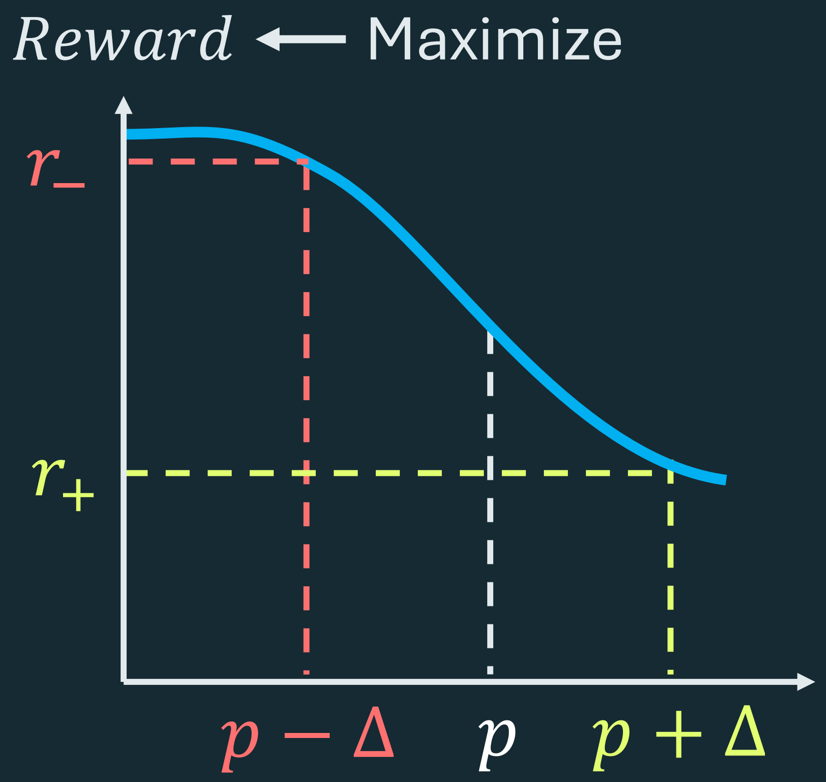
HybridBandits (Numerical Parameters)



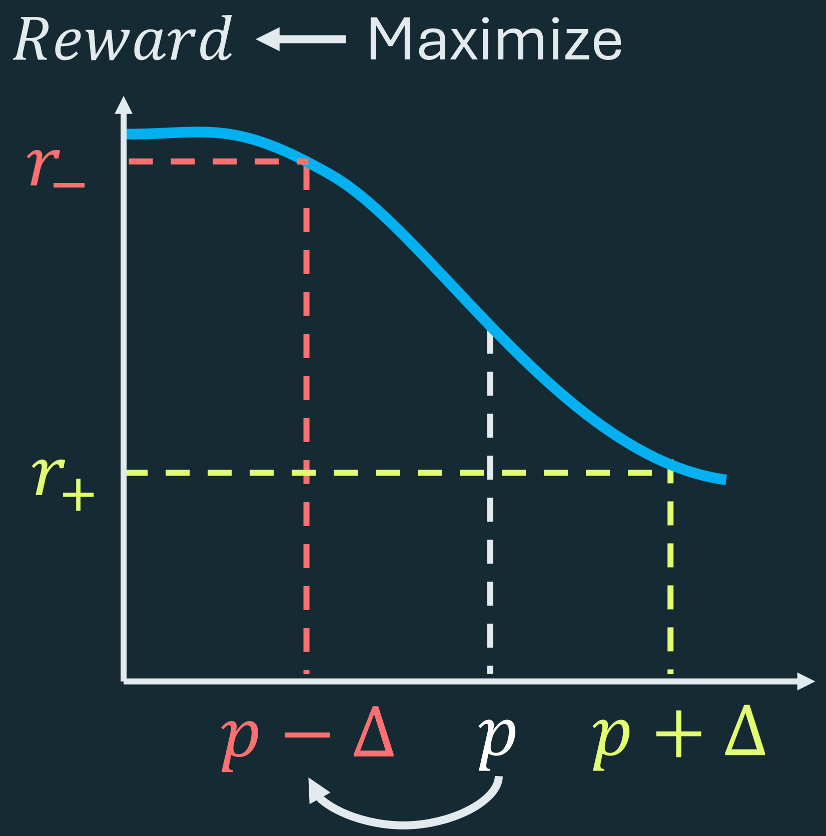
HybridBandits (Numerical Parameters)



HybridBandits (Numerical Parameters)



HybridBandits (Numerical Parameters)



HybridBandits (Categorical Parameters)

App

HybridBandits (Categorical Parameters)

App

Name	`a`
Value	`a2`
Categories	`0. a1` `1. a2`

Name	`b`
Value	`b1`

Categories	`0. b1`
	`1. b2`
	`2. b3`

HybridBandits (Categorical Parameters)

App

Name	a
Value	a2
Categories	0. a1 1. a2

Name	b
Value	b1
Categories	0. b1 1. b2 2. b3

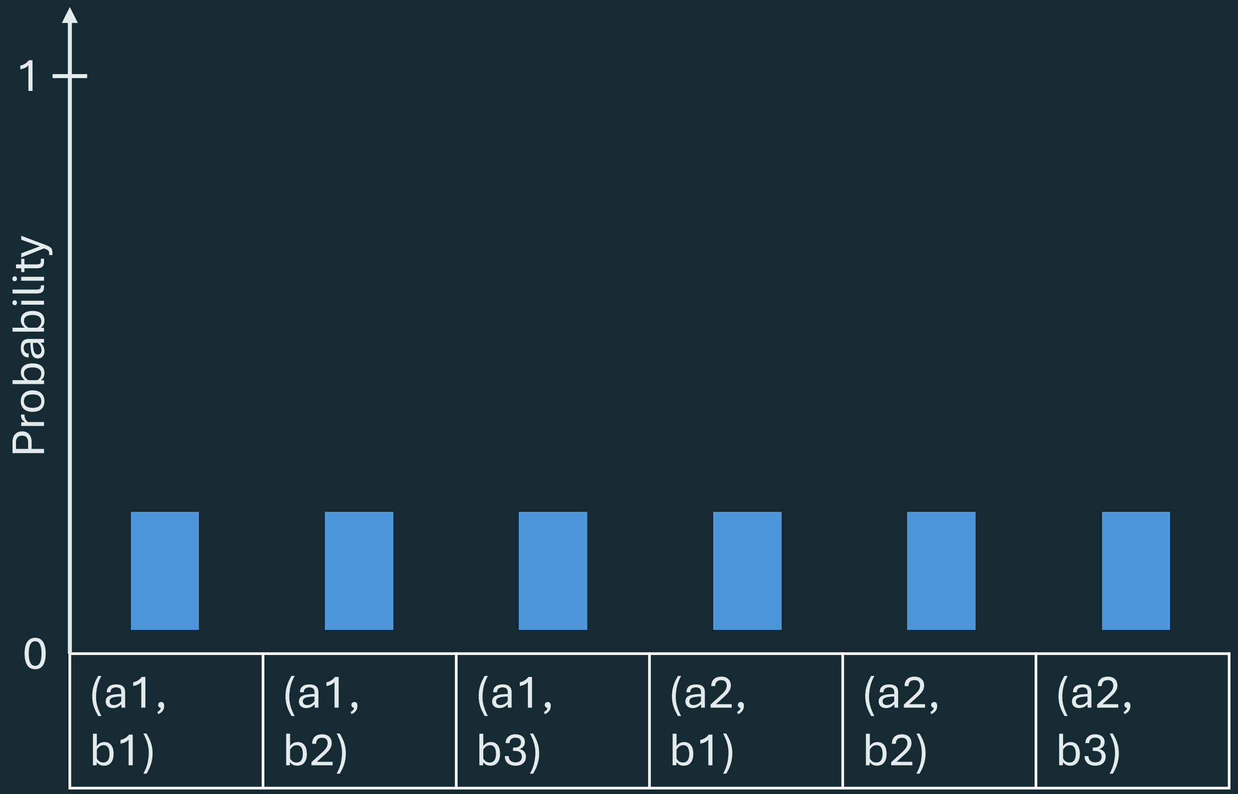
(a1, b1)	(a1, b2)	(a1, b3)	(a2, b1)	(a2, b2)	(a2, b3)

HybridBandits (Categorical Parameters)

App

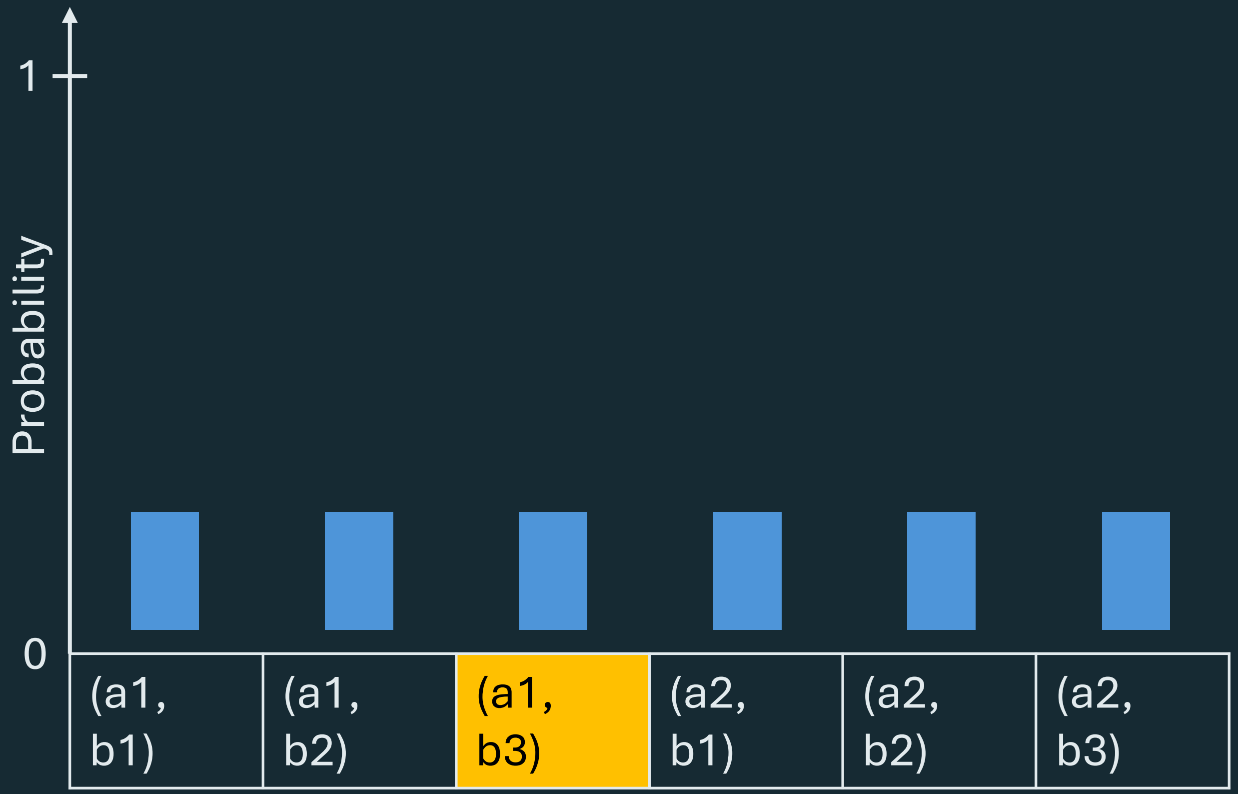
Name	a
Value	a2
Categories	0. a1 1. a2

Name	b
Value	b1
Categories	0. b1 1. b2 2. b3

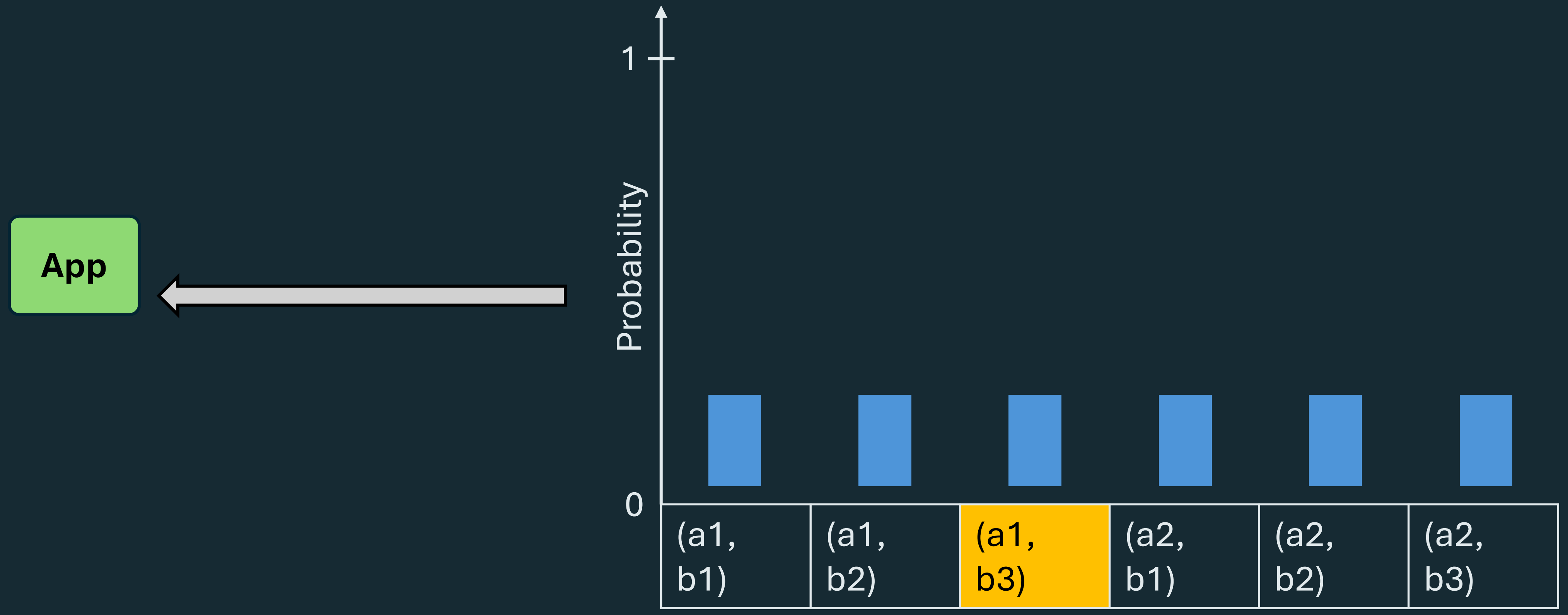


OPPerTune – HybridBandits (Categorical Parameters)

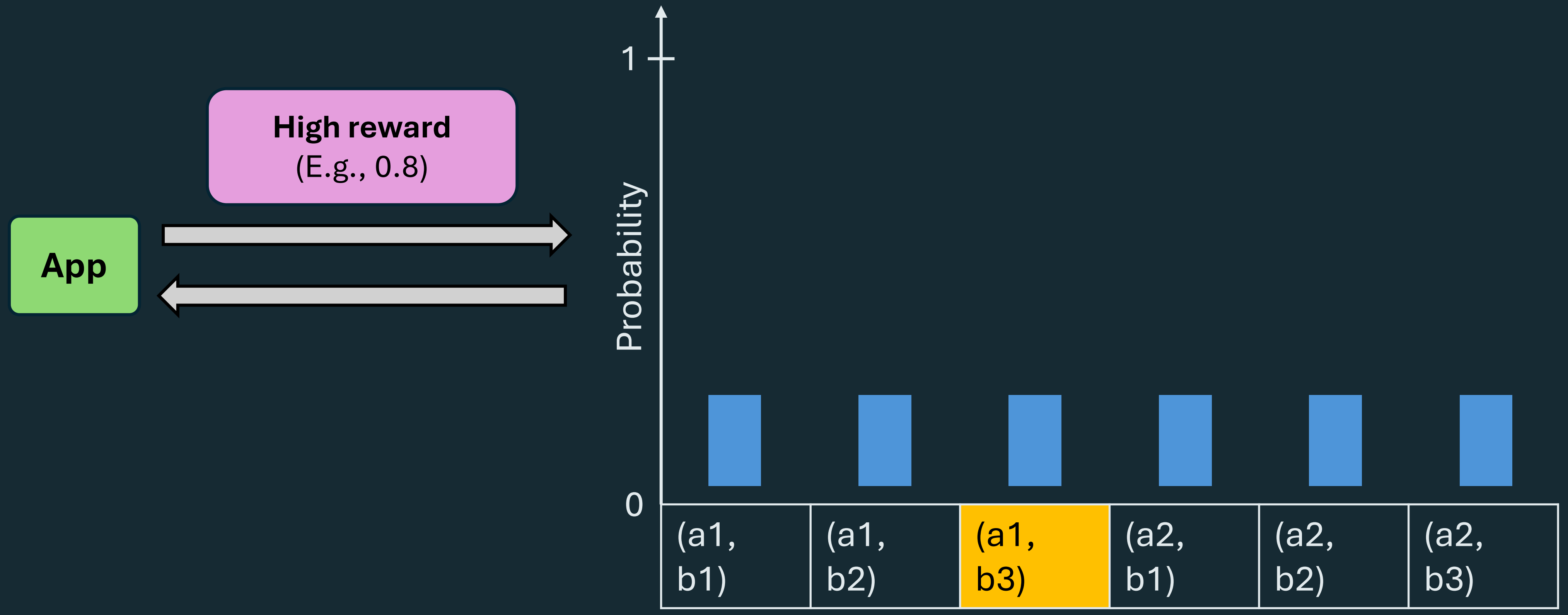
App



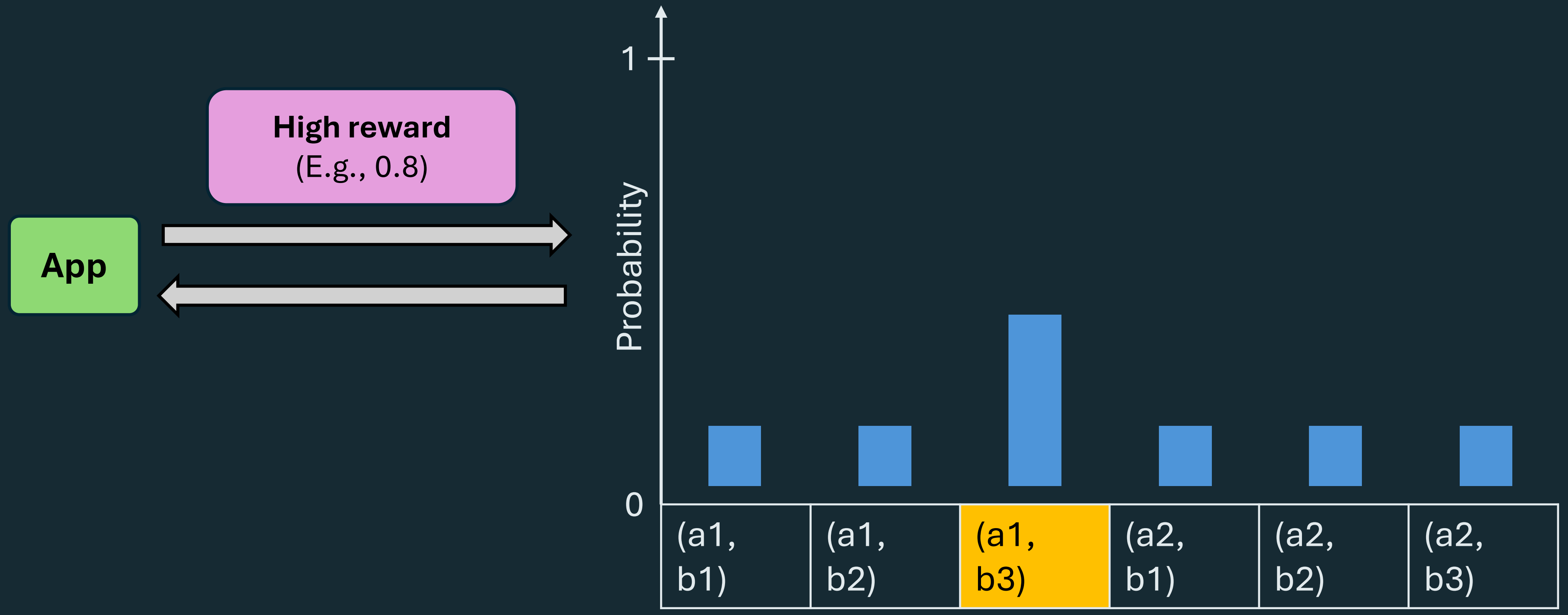
OPPerTune – HybridBandits (Categorical Parameters)



OPPerTune – HybridBandits (Categorical Parameters)

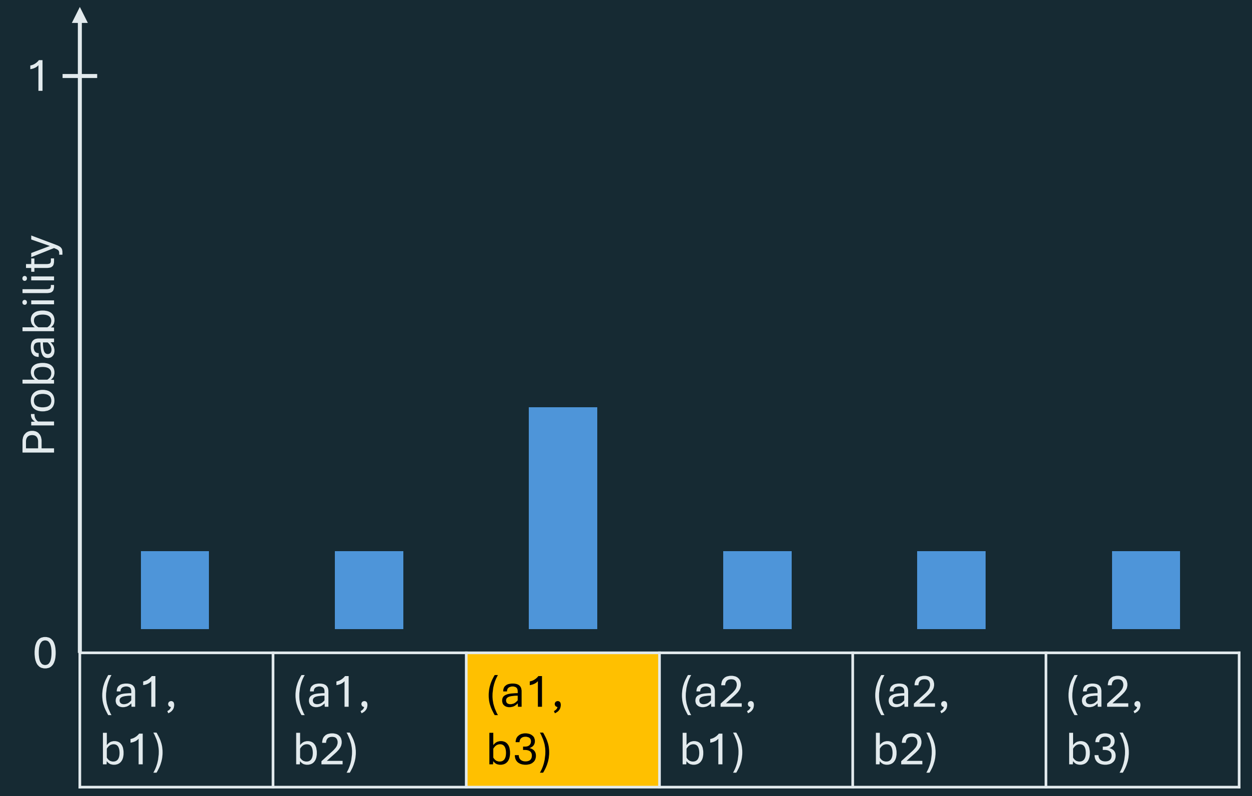


OPPerTune – HybridBandits (Categorical Parameters)



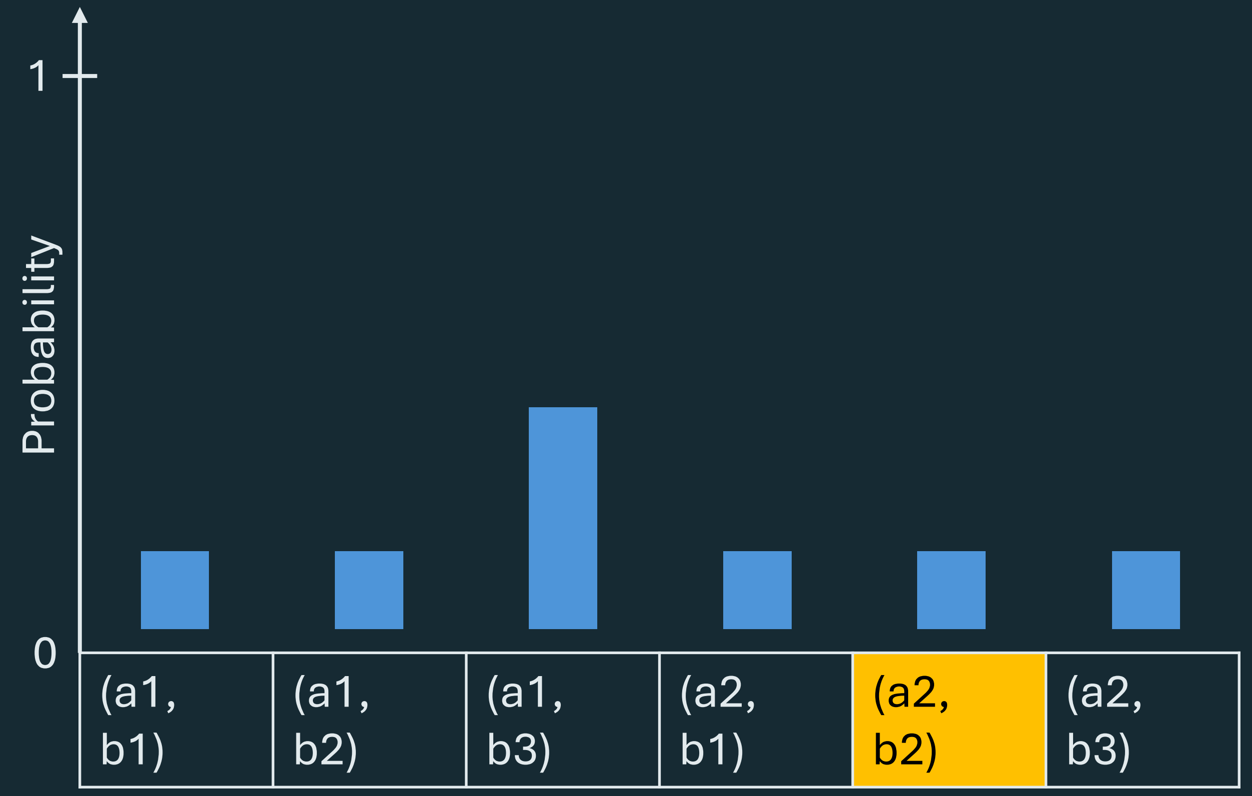
OPPerTune – HybridBandits (Categorical Parameters)

App

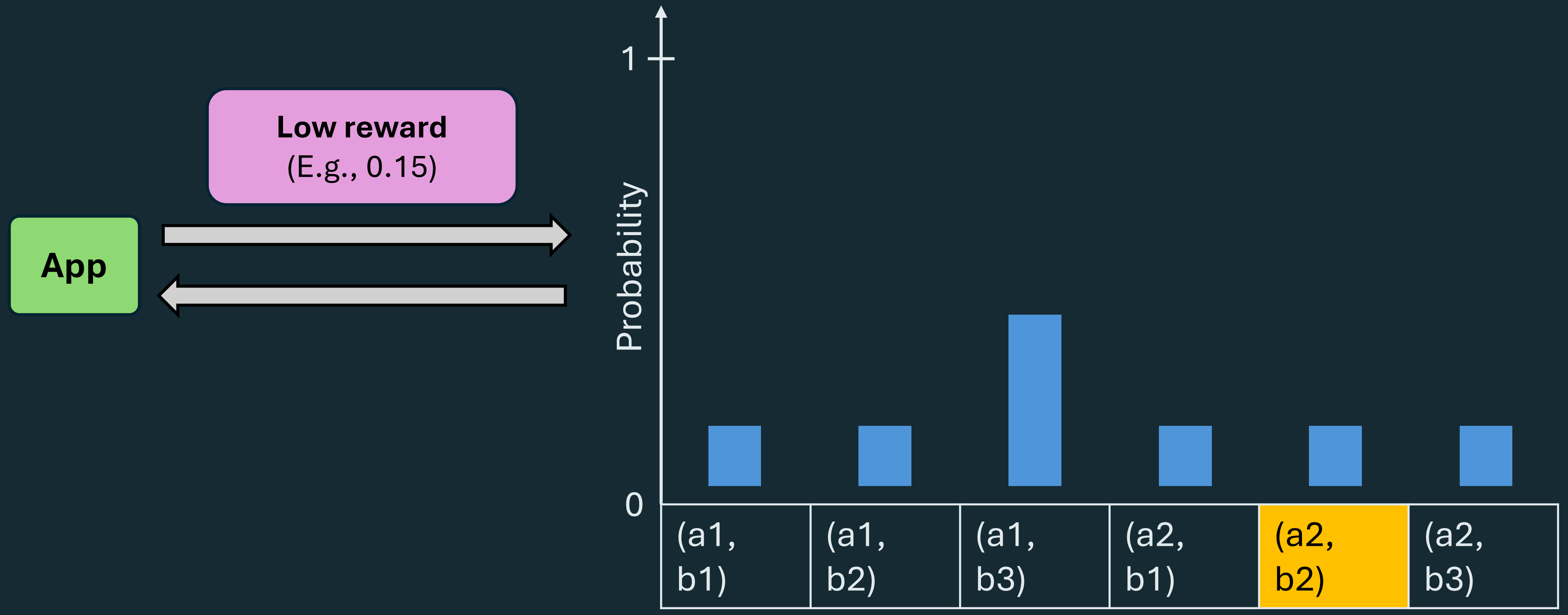


OPPerTune – HybridBandits (Categorical Parameters)

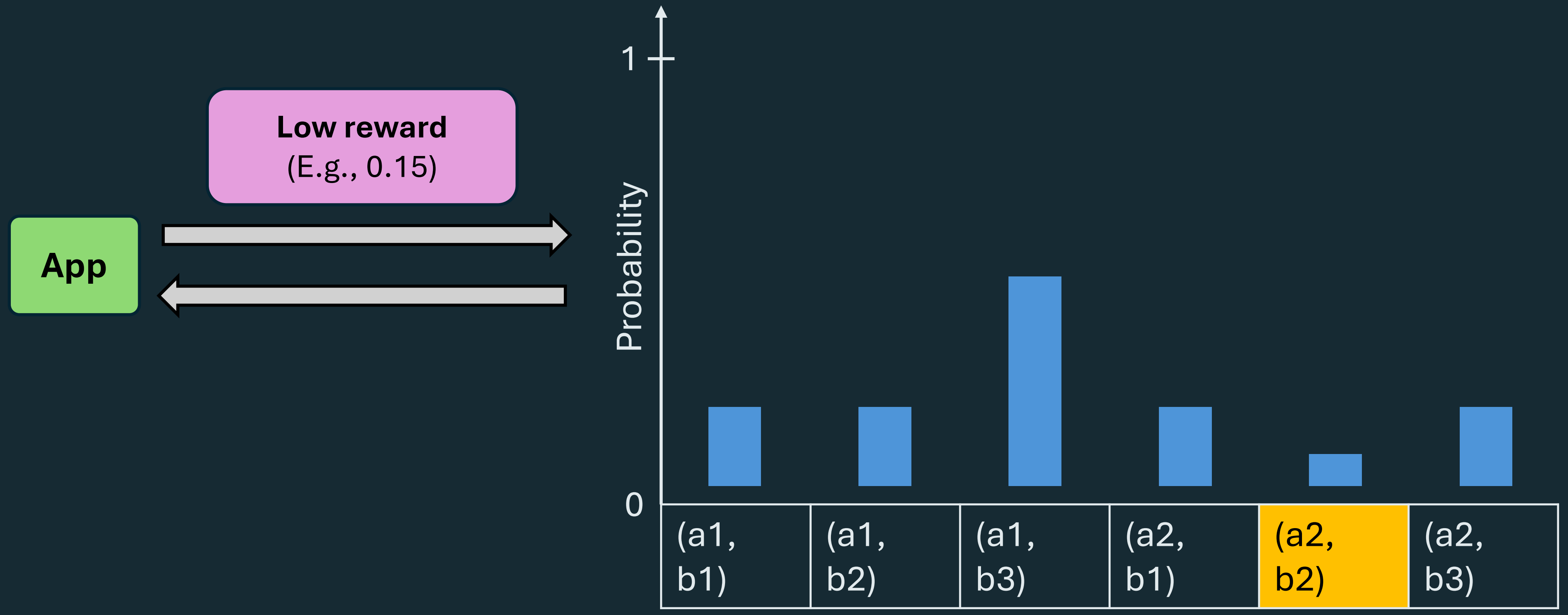
App



OPPerTune – HybridBandits (Categorical Parameters)

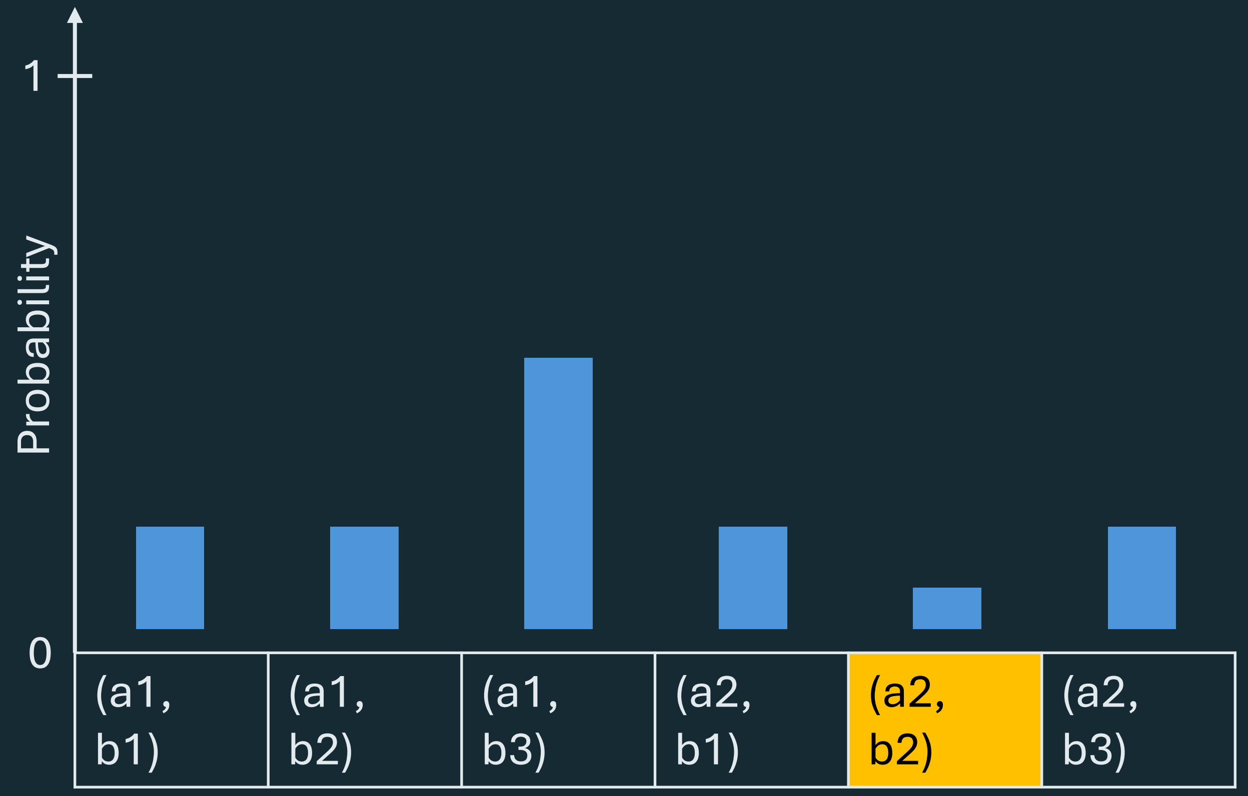


OPPerTune – HybridBandits (Categorical Parameters)



OPPerTune – HybridBandits (Categorical Parameters)

App



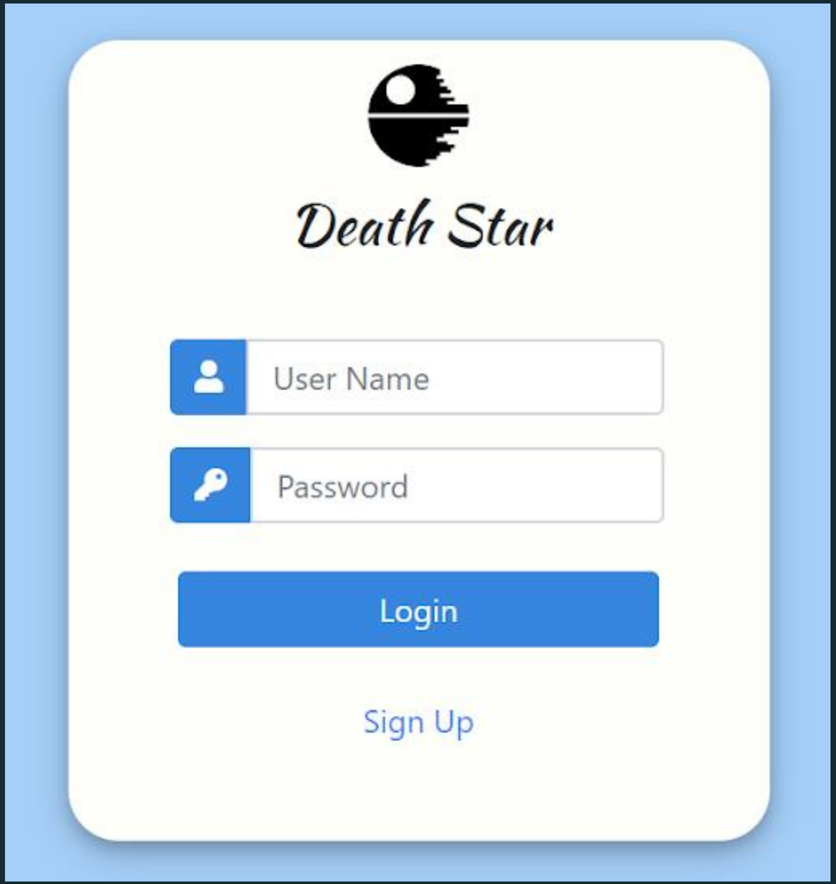
Our solution - HybridBandits

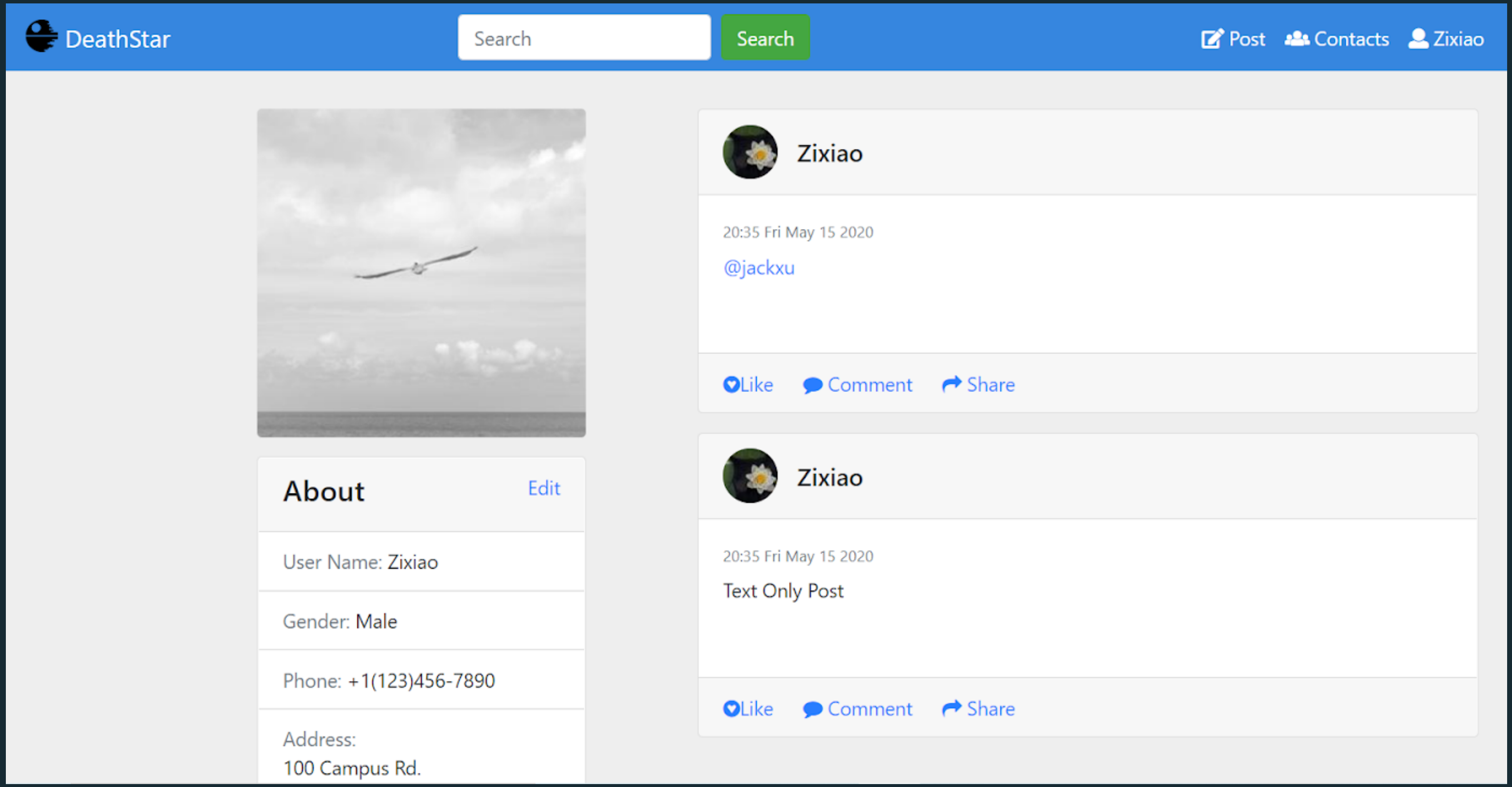
- Combines two algorithms – one for numerical and another for categorical parameters
- Scales linearly with number of numerical parameters and total categorical combinations

Challenges

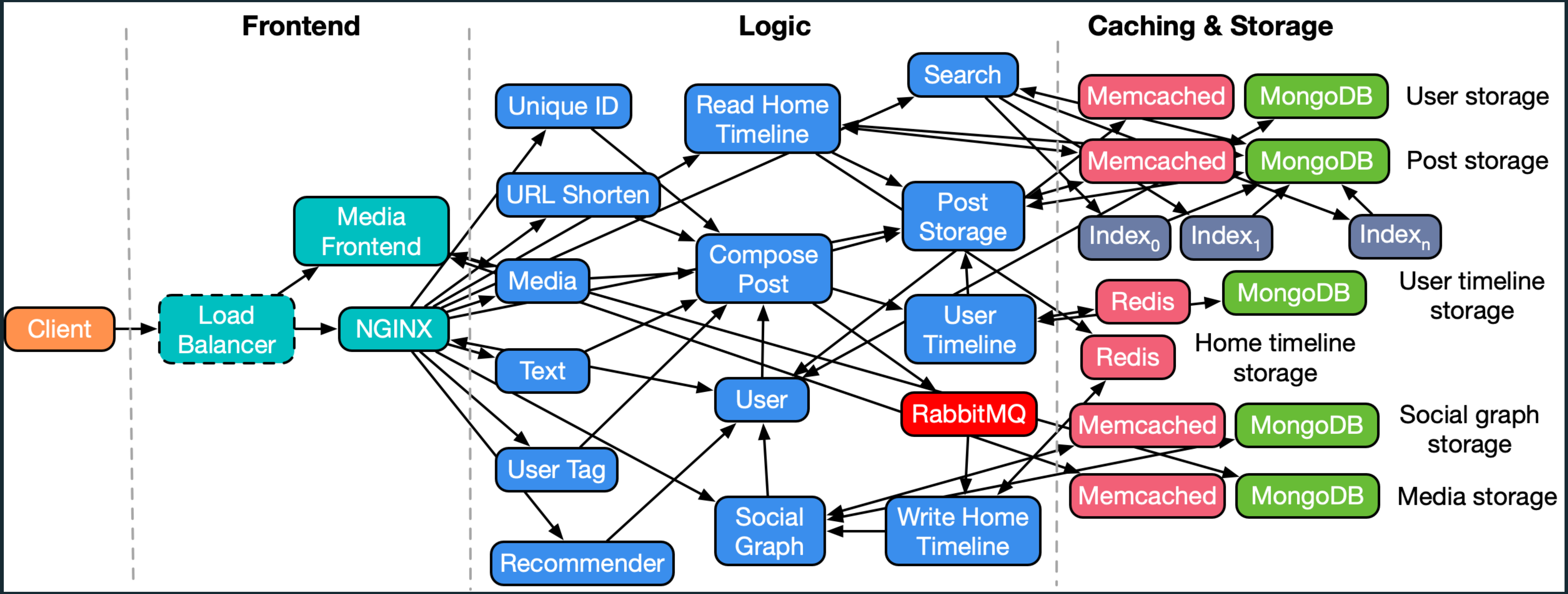
- Two different algorithms tuning different types of parameters
- Need to work with a single reward value

Evaluation – DeathStarBench (Social Network app)



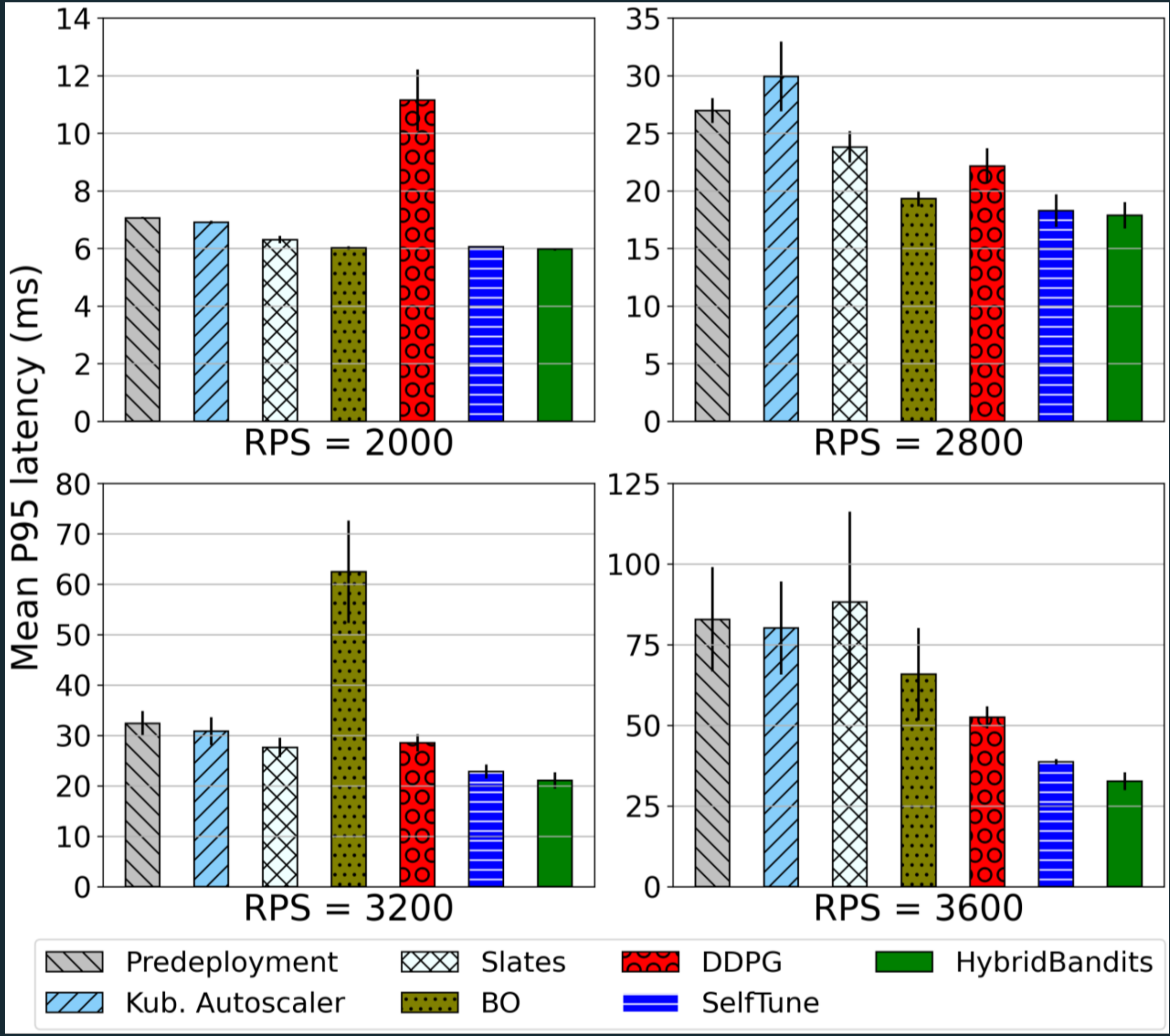


Evaluation – DeathStarBench (Social Network app)



Effectiveness of HybridBandits

- **Workload:** Constant RPS on the DeathStarBench Social Network app
- Fixed budget of 50 iterations
- An iteration involves
    1. Configuring the app with the parameters returned by the algorithm
    2. Running workload for 20 minutes
    3. Measuring the P95 latency

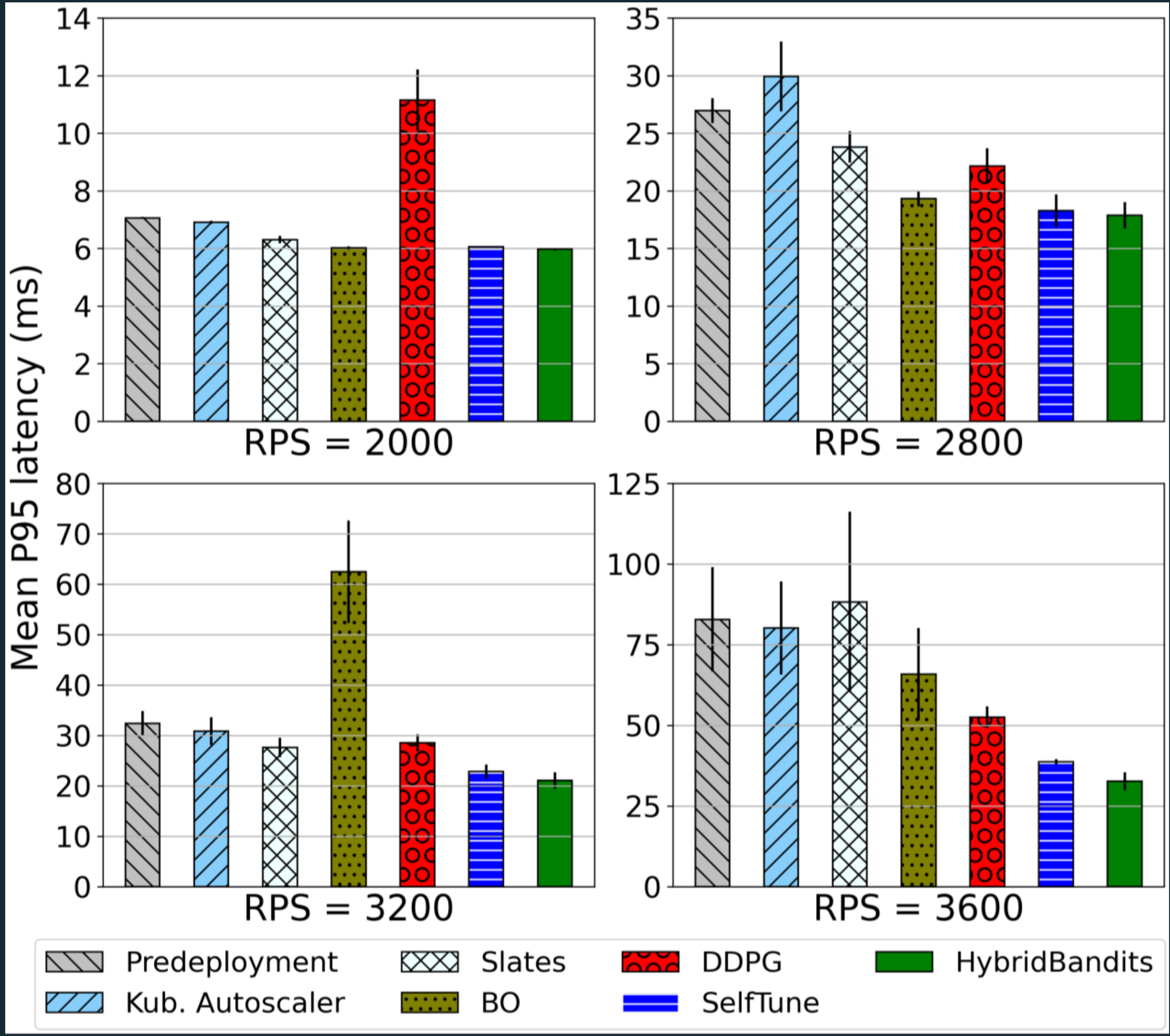


Effectiveness of HybridBandits

- **Workload:** Constant RPS on the DeathStarBench Social Network app
- Fixed budget of 50 iterations
- An iteration involves
    1. Configuring the app with the parameters returned by the algorithm
    2. Running workload for 20 minutes
    3. Measuring the P95 latency

Takeaway

About 15%-20% reduction in the tail latency of the application, relative to SelfTune, especially at higher workloads (when the shared resources in the VMs are strained).



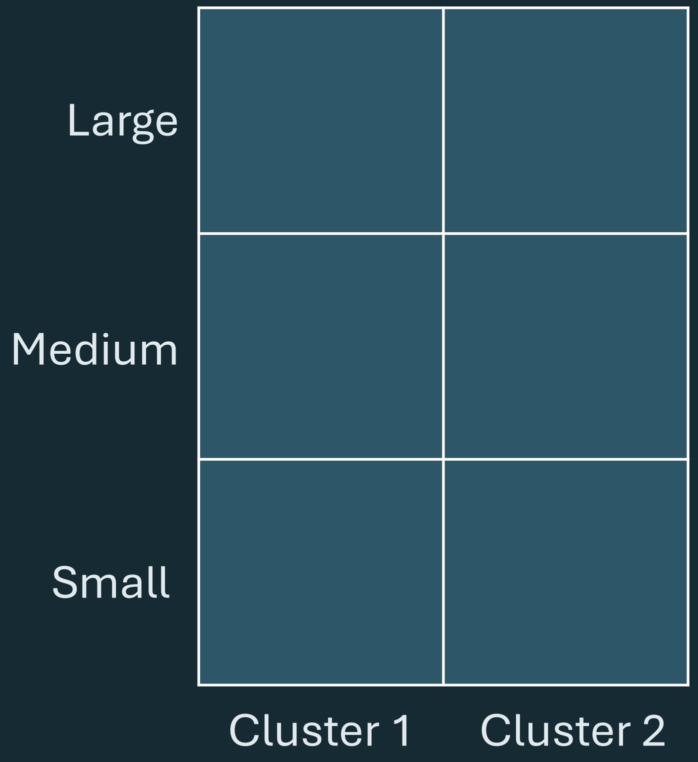
Challenge 2
Scoping the tuning problem

Scoping the tuning problem

- Workload characteristics (context)
  - Workload size = {Small, Medium, Large}
  - Cluster ID = {1, 2}

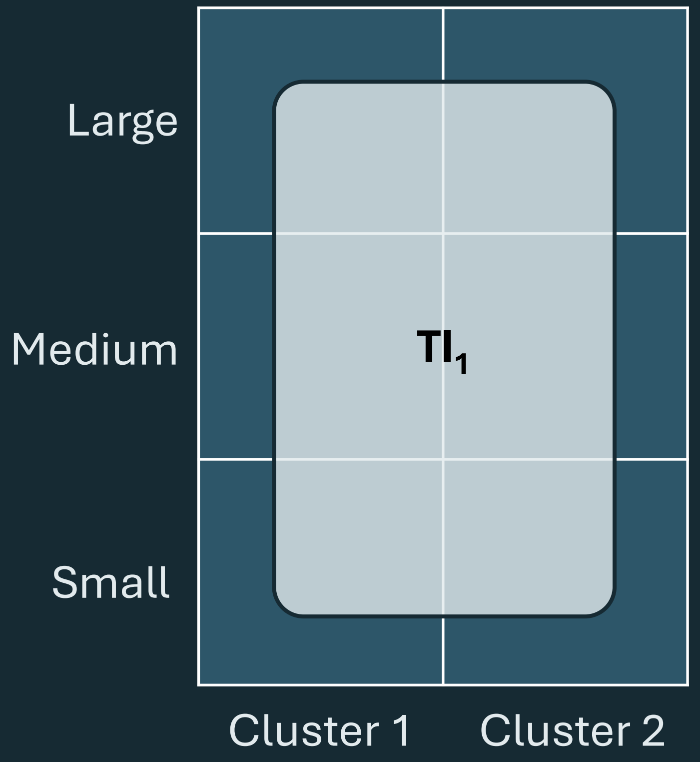
Scoping the tuning problem

- Workload characteristics (context)
  - Workload size = {Small, Medium, Large}
  - Cluster ID = {1, 2}



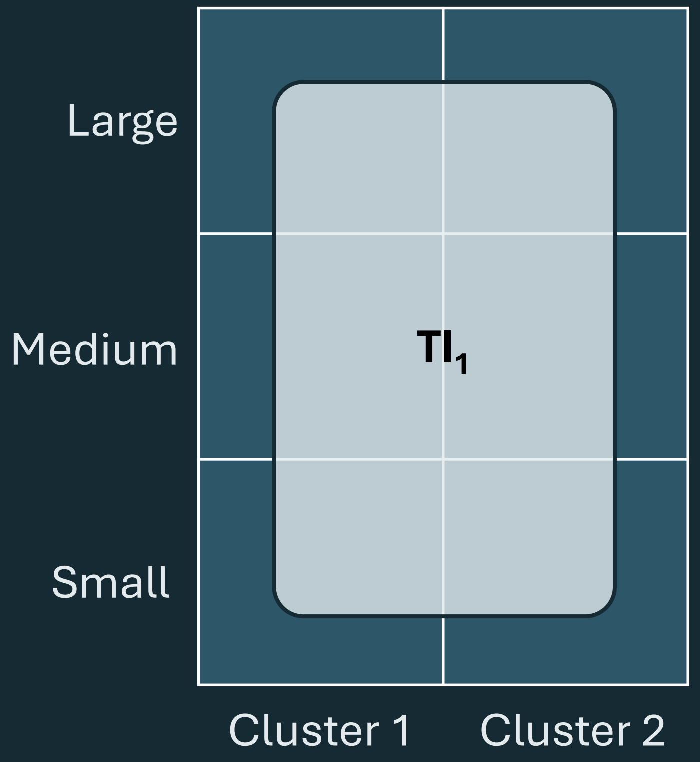
Scoping the tuning problem

- Workload characteristics (context)
  - Workload size = {Small, Medium, Large}
  - Cluster ID = {1, 2}



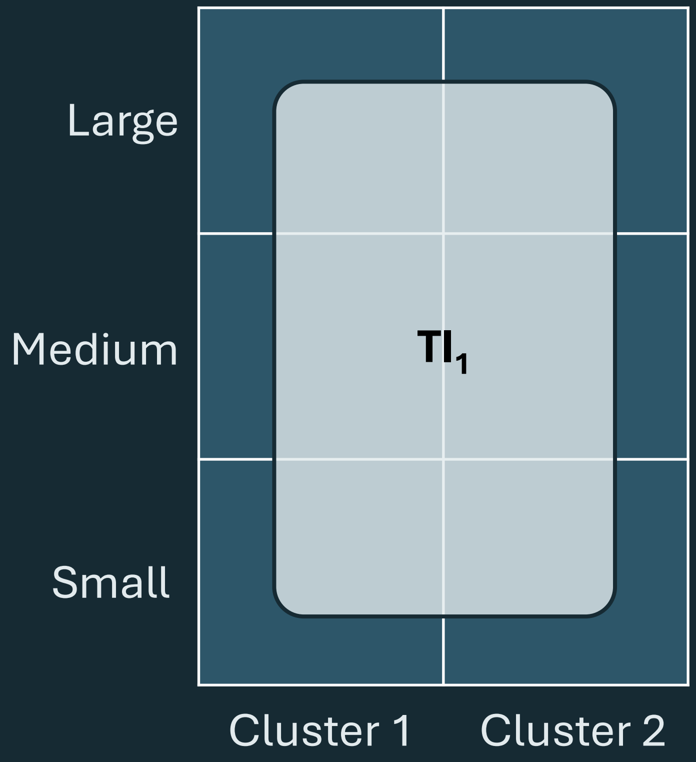
Scoping the tuning problem

- Workload characteristics (context)
  - Workload size = {Small, Medium, Large}
  - Cluster ID = {1, 2}

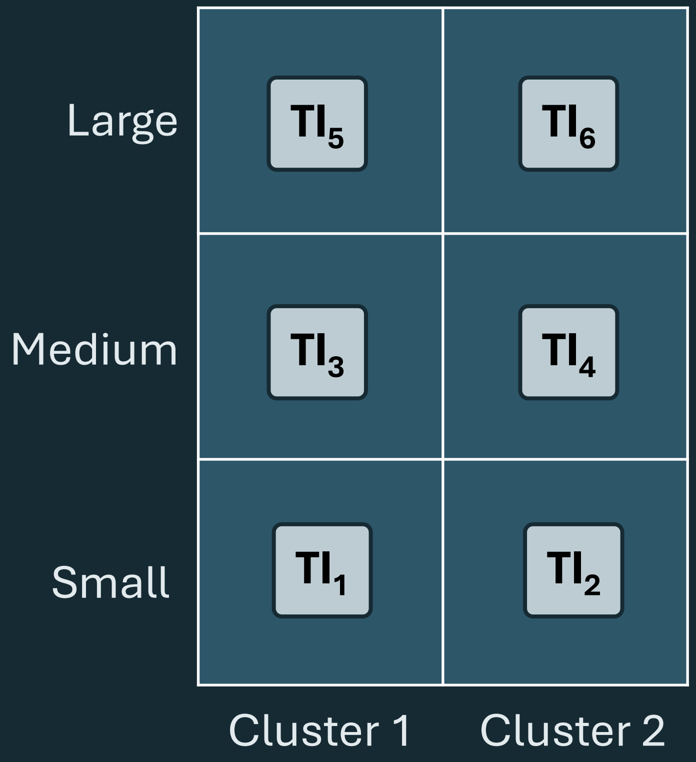


Scoping the tuning problem

- Workload characteristics (context)
  - Workload size = {Small, Medium, Large}
  - Cluster ID = {1, 2}

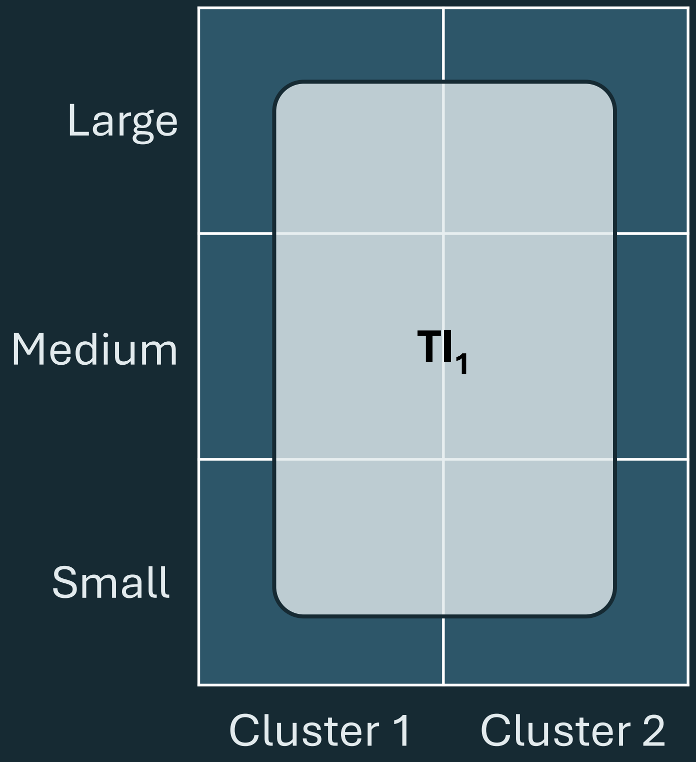


?



Scoping the tuning problem

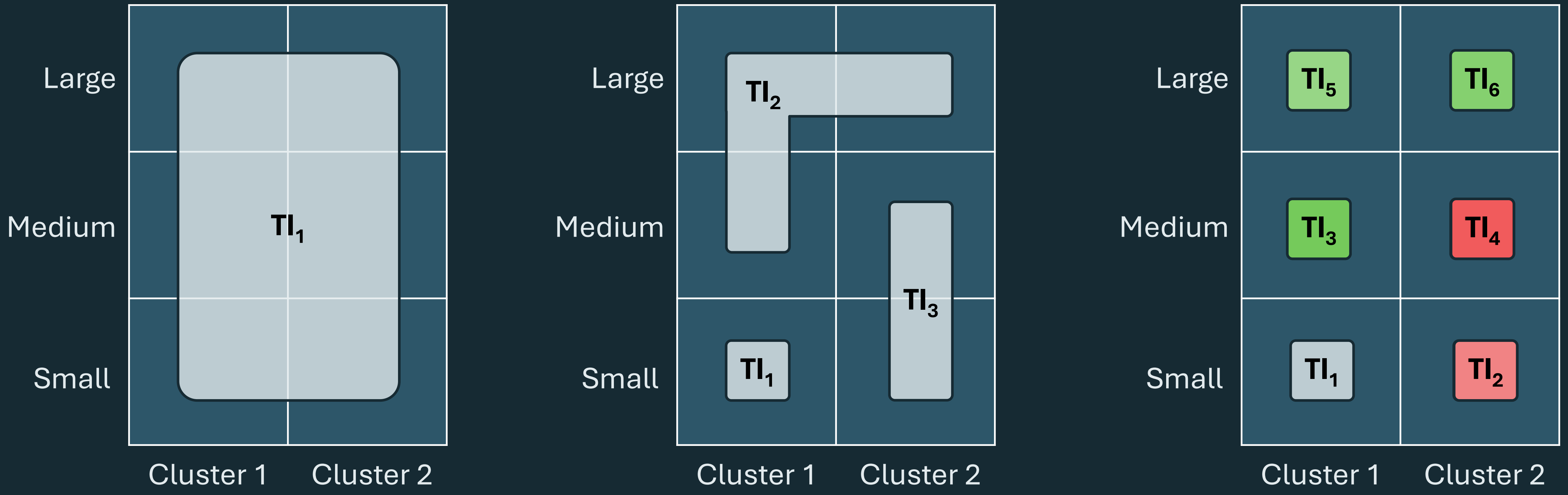
- Workload characteristics (context)
  - Workload size = {Small, Medium, Large}
  - Cluster ID = {1, 2}



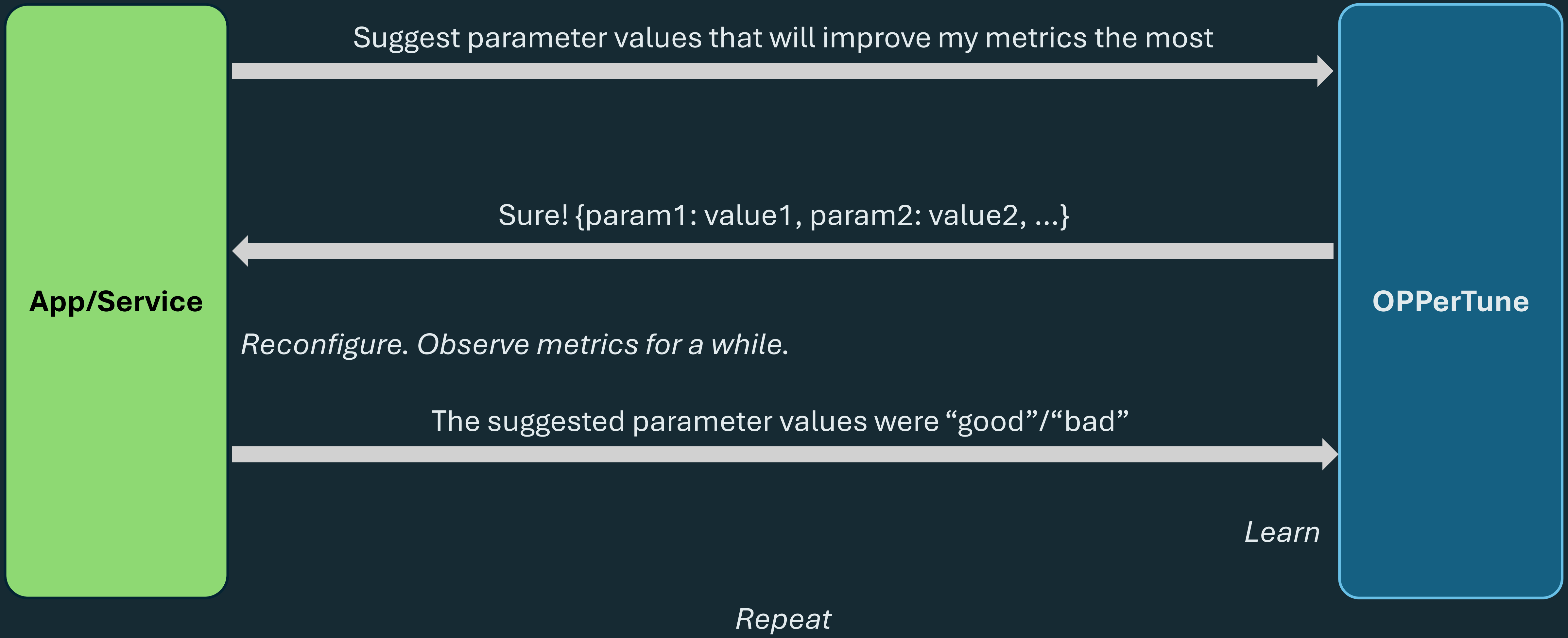
?

Scoping the tuning problem

- Workload characteristics (context)
  - Workload size = {Small, Medium, Large}
  - Cluster ID = {1, 2}



AutoScope


App/Service
Suggest parameter values that will improve my metrics the most
Sure! {param1: value1, param2: value2, ...}
Reconfigure. Observe metrics for a while.
The suggested parameter values were “good”/“bad”
Learn
OPPerTune
Repeat

AutoScope

App/Service

Suggest parameter values that will improve my metrics the most

Also, here is some information (context) about the workload/environment

Sure! {param1: value1, param2: value2, ...}

Reconfigure. Observe metrics for a while.

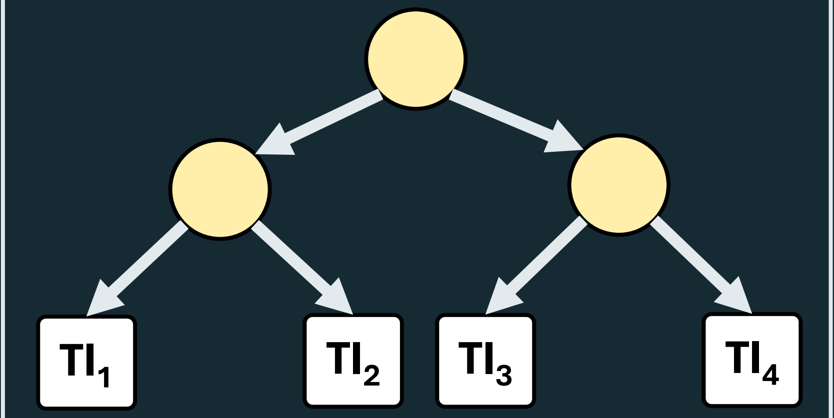
The suggested parameter values were “good”/“bad”

Learn

OPPerTune

Repeat

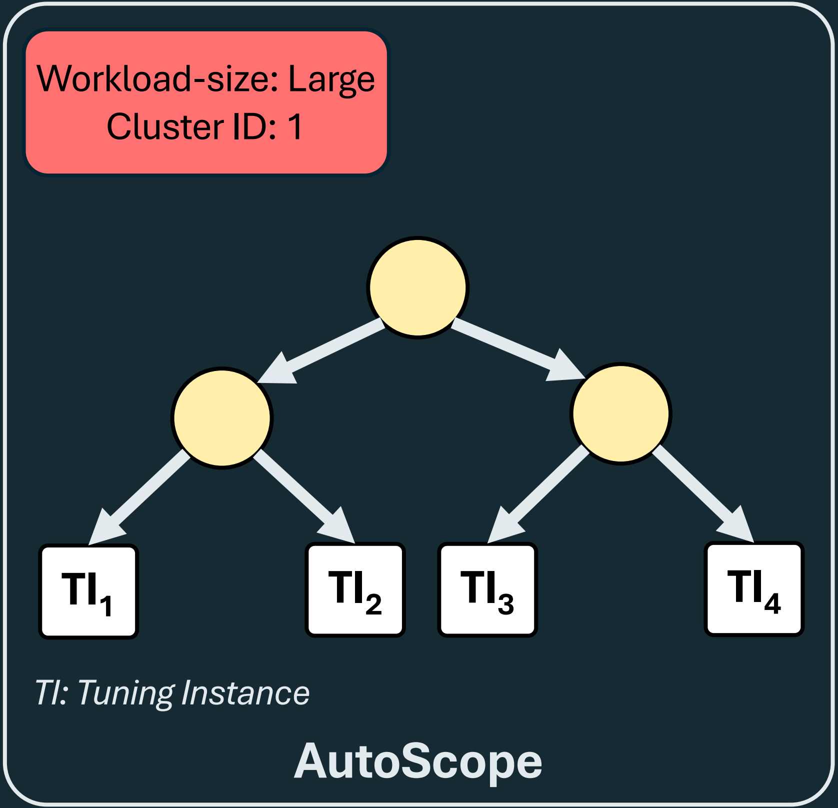
AutoScope



TI: Tuning Instance

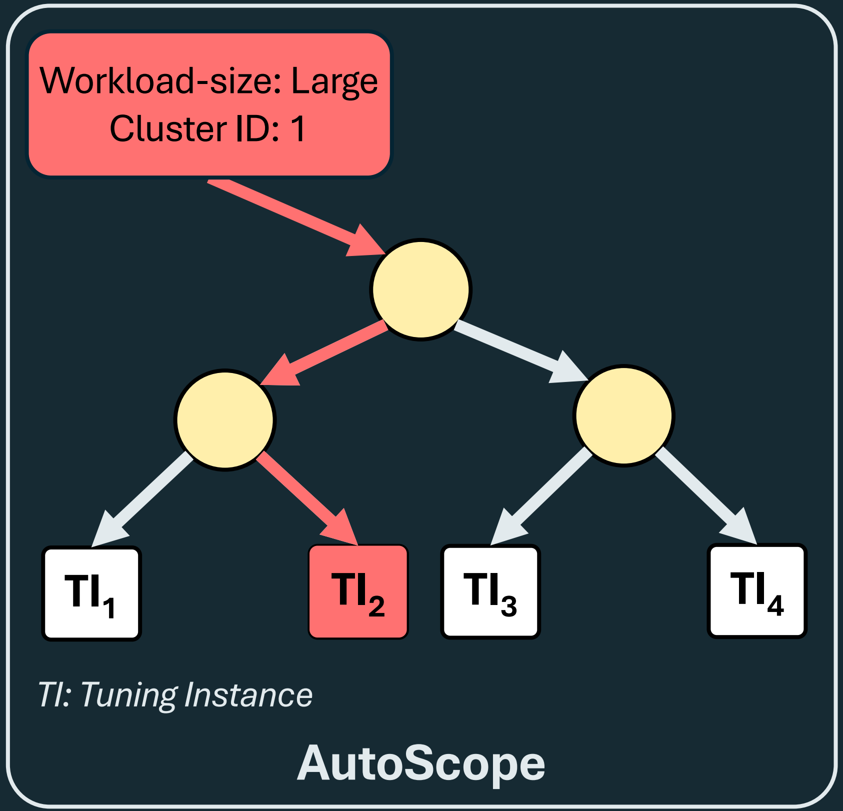
AutoScope

AutoScope



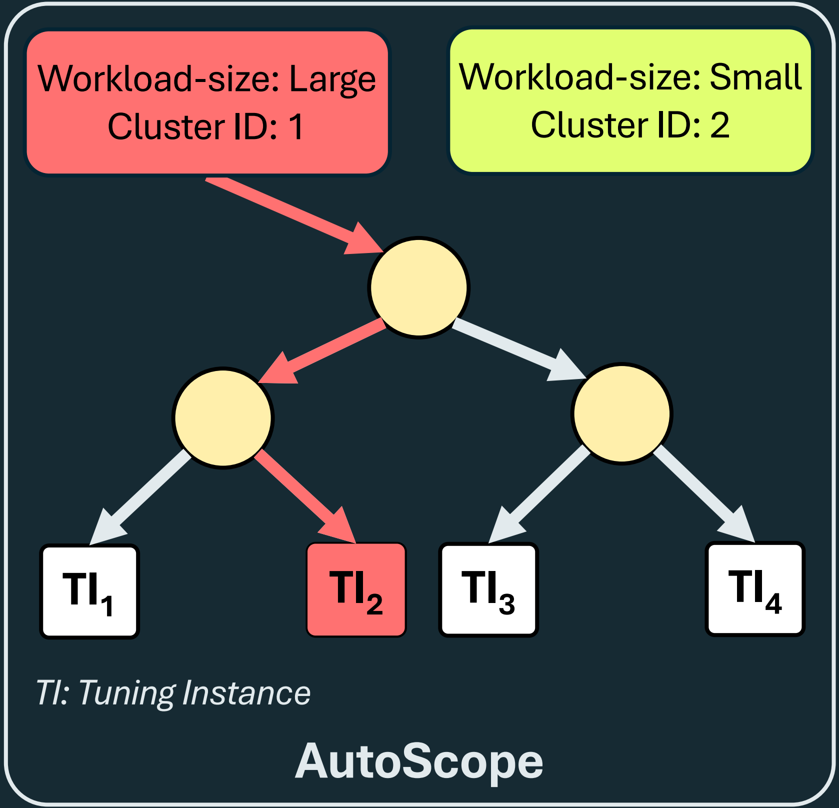

```python
if workload_size == “large” and cluster_id == 1:
```

AutoScope




```
if workload_size == “large” and cluster_id == 1:
    return tuning_instance_2.predict()
    # scan_interval=40, expander=“priority”
```

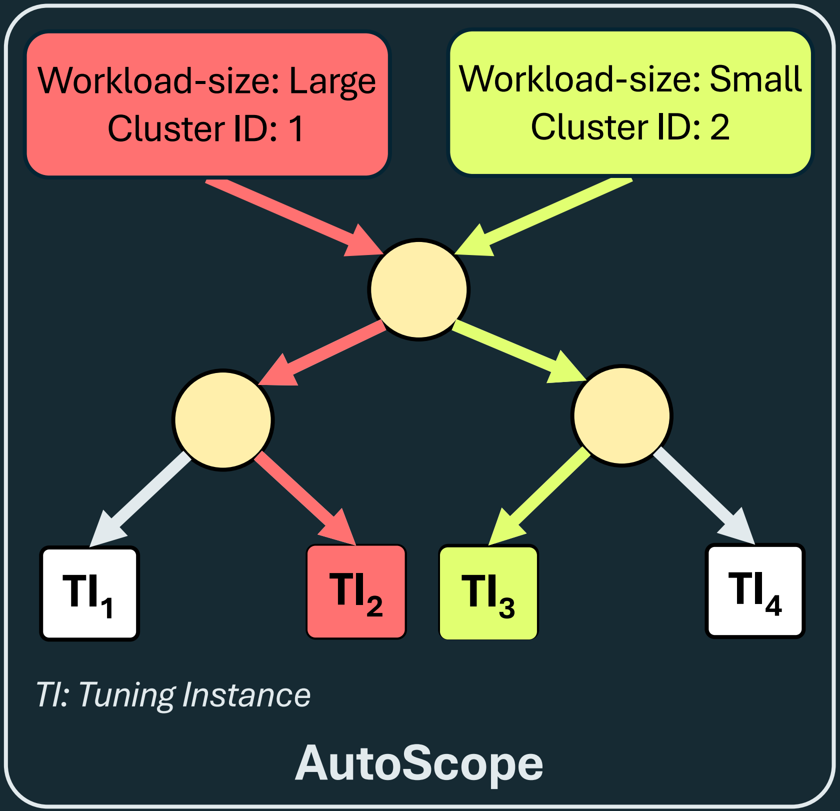
AutoScope




```
if workload_size == “large” and cluster_id == 1:
    return tuning_instance_2.predict()
    # scan_interval=40, expander=“priority”

elif workload_size == “small” and cluster_id == 2:
```

AutoScope




```
if workload_size == “large” and cluster_id == 1:
    return tuning_instance_2.predict()
    # scan_interval=40, expander=“priority”

elif workload_size == “small” and cluster_id == 2:
    return tuning_instance_3.predict()
    # scan_interval=30, expander=“random”

elif ...
```

Effectiveness of AutoScope

- Deployed SelfTune and OPPerTune for 2 weeks in 2 production clusters (workloads could be re-run on the clusters, so we could test both the frameworks in the timeframe)
- **1 week of training** followed by **1 week of testing**

Method	Experiment Completion Time (in minutes)		Sample Complexity (#rewards)	
	Cluster 1	Cluster 2	Cluster 1	Cluster 2
Pre-deployment choices	105.85 ± 16.75	36.66 ± 1.60	-	-
Expert choices	42.41 ± 5.28	34.46 ± 4.72	-	-
$\text{SelfTune}_{\text{cluster, type, size}}$	38.56 ± 6.55	30.79 ± 0.52	94	313
AutoScope	38.98 ± 5.90	32.71 ± 0.26	29	61

Effectiveness of AutoScope

- Deployed SelfTune and OPPerTune for 2 weeks in 2 production clusters (workloads could be re-run on the clusters, so we could test both the frameworks in the timeframe)
- **1 week of training** followed by **1 week of testing**

Method	Experiment Completion Time (in minutes)		Sample Complexity (#rewards)	
	Cluster 1	Cluster 2	Cluster 1	Cluster 2
Pre-deployment choices	105.85 ± 16.75	36.66 ± 1.60	-	-
Expert choices	42.41 ± 5.28	34.46 ± 4.72	-	-
$\text{SelfTune}_{\text{cluster, type, size}}$	38.56 ± 6.55	30.79 ± 0.52	94	313
AutoScope	38.98 ± 5.90	32.71 ± 0.26	29	61

Takeaway

AutoScope reduces completion times significantly with less than a third of samples (i.e., in less than a third of time)

Effectiveness of AutoScope

- Deployed SelfTune and OPPerTune for 2 weeks in 2 production clusters (workloads could be re-run on the clusters, so we could test both the frameworks in the timeframe)
- **1 week of training** followed by **1 week of testing**

Method	Experiment Completion Time (in minutes)		Sample Complexity (#rewards)	
	Cluster 1	Cluster 2	Cluster 1	Cluster 2
Pre-deployment choices	105.85 ± 16.75	36.66 ± 1.60	-	-
Expert choices	42.41 ± 5.28	34.46 ± 4.72	-	-
$\text{SelfTune}_{\text{cluster, type, size}}$	38.56 ± 6.55	30.79 ± 0.52	94	313
AutoScope	38.98 ± 5.90	32.71 ± 0.26	29	61

Takeaway

AutoScope reduces completion times significantly with less than a third of samples (i.e., in less than a third of time)

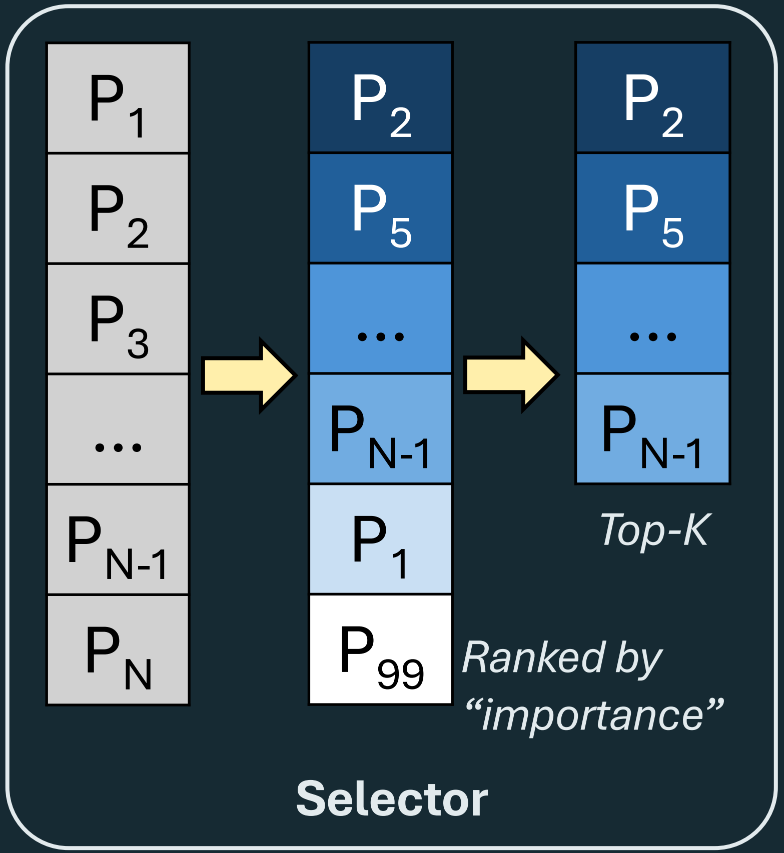
Challenge 3
Filtering parameters to tune

Microbenchmarking – Mitigating the cost of tuning

- Apps can have 1000s of parameters
- Tuning all of them will be slow
- Can also be costly
  - E.g., service disruptions

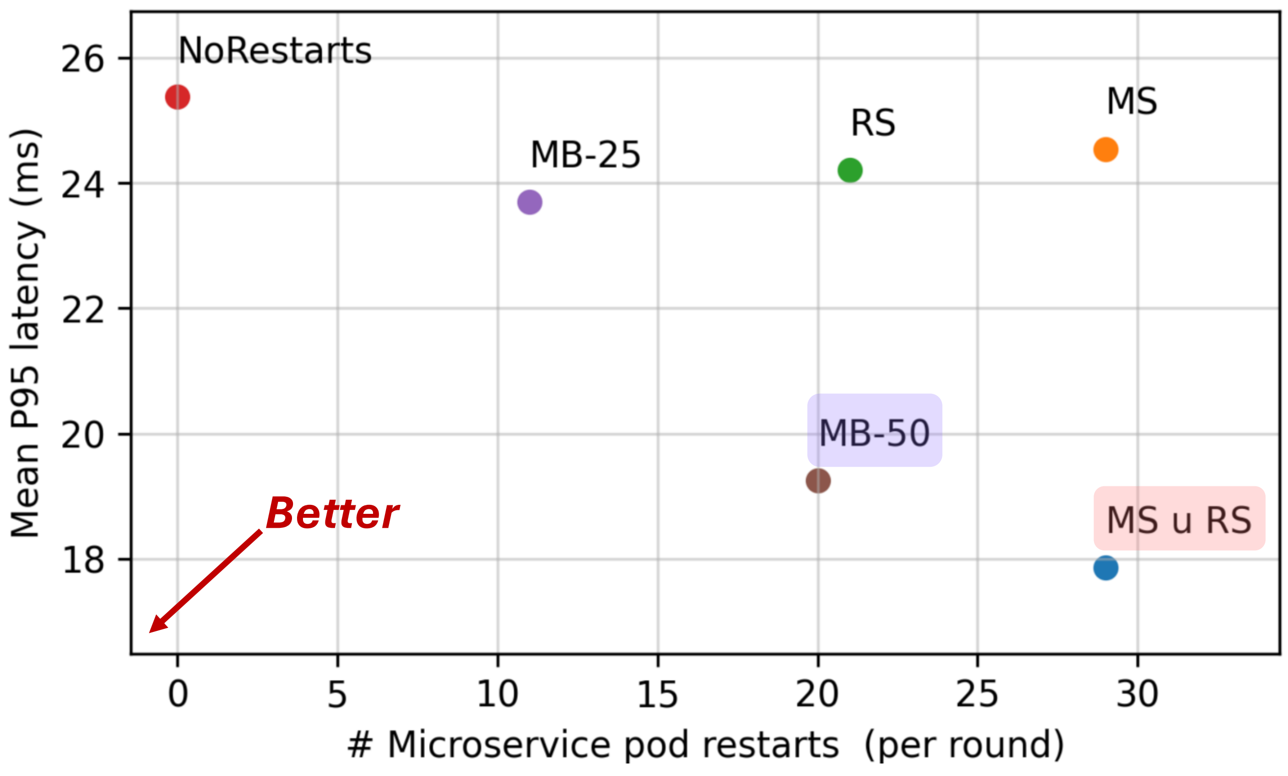
Microbenchmarking – Mitigating the cost of tuning

- Apps can have 1000s of parameters
- Tuning all of them will be slow
- Can also be costly
  - E.g., service disruptions



Effectiveness of Microbenchmarking

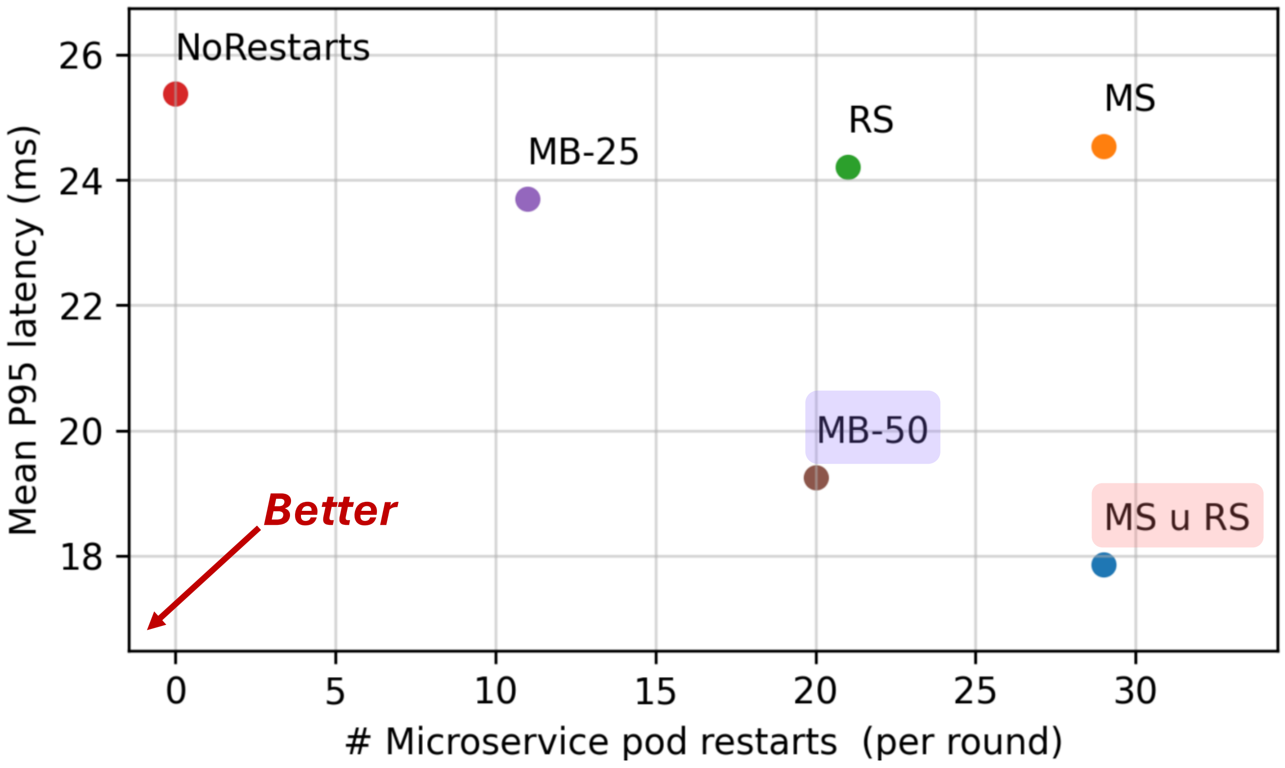
- **Workload:** Constant RPS (here, 2800) on the DeathStarBench Social Network app
- Fixed budget of 50 rounds
- 1 round means
  - Configuring the app with the parameters returned by the algorithm
  - Running workload for 20 minutes
  - Measuring the P95 latency



MS=Microservices, RS=Rightsizing, MB=Microbenchmarking

Effectiveness of Microbenchmarking

- **Workload:** Constant RPS (here, 2800) on the DeathStarBench Social Network app
- Fixed budget of 50 rounds
- 1 round means
  - Configuring the app with the parameters returned by the algorithm
  - Running workload for 20 minutes
  - Measuring the P95 latency



MS=Microservices, RS=Rightsizing, MB=Microbenchmarking

Takeaway

Microbenchmarking achieves a good tradeoff between performance and the cost of tuning

Summary

Summary

- Parameter tuning for deployed apps is hard

Summary

- Parameter tuning for deployed apps is hard
  - Large parameter space

Summary

- Parameter tuning for deployed apps is hard
  - Large parameter space
  - Hybrid parameters across the software stack

Summary

- Parameter tuning for deployed apps is hard
  - Large parameter space
  - Hybrid parameters across the software stack
  - Determining the right scope

Summary

- Parameter tuning for deployed apps is hard
  - Large parameter space
  - Hybrid parameters across the software stack
  - Determining the right scope
  - Deciding which parameters to tune

Summary

- Parameter tuning for deployed apps is hard
  - Large parameter space
  - Hybrid parameters across the software stack
  - Determining the right scope
  - Deciding which parameters to tune
- OPPerTune helps address all these

Summary

- Parameter tuning for deployed apps is hard
  - Large parameter space
  - Hybrid parameters across the software stack
  - Determining the right scope
  - Deciding which parameters to tune
- OPPerTune helps address all these
  - Propose novel algorithms to tackle these challenges

Summary

- Parameter tuning for deployed apps is hard
  - Large parameter space
  - Hybrid parameters across the software stack
  - Determining the right scope
  - Deciding which parameters to tune
- OPPerTune helps address all these
  - Propose novel algorithms to tackle these challenges
  - General end-to-end tuning service

Summary

- Parameter tuning for deployed apps is hard
  - Large parameter space
  - Hybrid parameters across the software stack
  - Determining the right scope
  - Deciding which parameters to tune
- OPPerTune helps address all these
  - Propose novel algorithms to tackle these challenges
  - General end-to-end tuning service
  - Easy to setup and integrate, and is scalable

Summary

- Parameter tuning for deployed apps is hard
  - Large parameter space
  - Hybrid parameters across the software stack
  - Determining the right scope
  - Deciding which parameters to tune
- OPPerTune helps address all these
  - Propose novel algorithms to tackle these challenges
  - General end-to-end tuning service
  - Easy to setup and integrate, and is scalable
  - Is open-sourced (github.com/microsoft/oppertune)

Systems Innovation – hiring!

- Systems Innovation – Microsoft Research - **aka.ms/systems-innovation**
- We are always on the lookout for motivated candidates for Researcher, PostDoc and Internship positions to work on cutting edge research at the intersection of AI and Systems.
- If you are interested, please reach out to: **m365research-careers@microsoft.com**

Resources

- Project homepage
  - aka.ms/oppertune
- Repo
  - github.com/microsoft/oppertune
- Related papers
  1. SelfTune: Tuning Cluster Managers *(NSDI 2023)*
  2. Learning Accurate Decision Trees with Bandit Feedback via Quantized Gradient Descent (*TMLR 2022*)
  3. Optimal regret algorithm for Pseudo-1d Bandit Convex Optimization (*PMLR 2021*)

aka.ms/oppertune