



Symbiosis: The Art of Application and Kernel Cache Cooperation

Yifan Dai, Jing Liu,
Andrea Arpaci-Dusseau and Remzi Arpaci-Dusseau



Database Systems and Storage Engines



Large scale database systems

MongoDB / MySQL / CockroachDB



Database Systems and Storage Engines

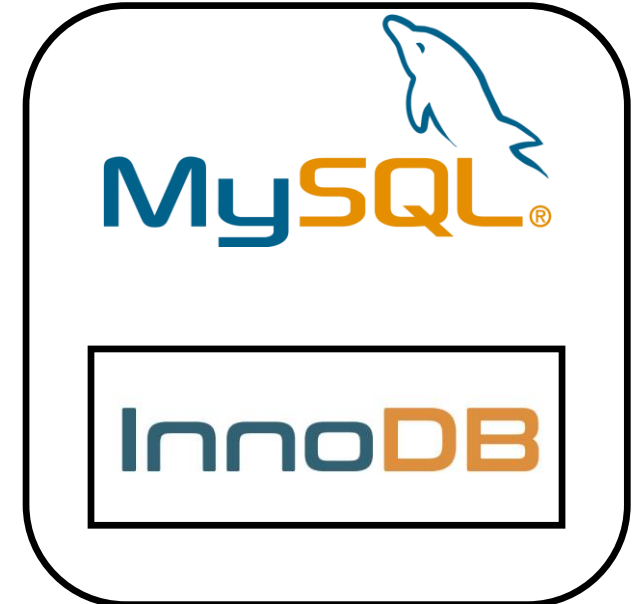
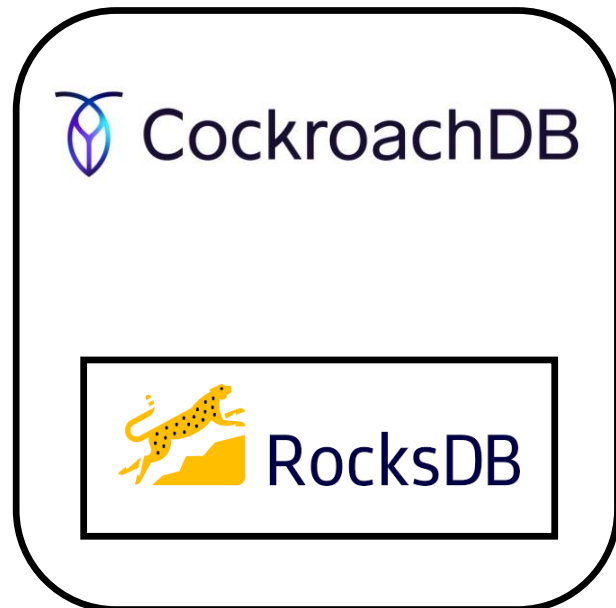


Large scale database systems

MongoDB / MySQL / CockroachDB

Storage engines within DB to interact with filesystems

RocksDB / WiredTiger / InnoDB

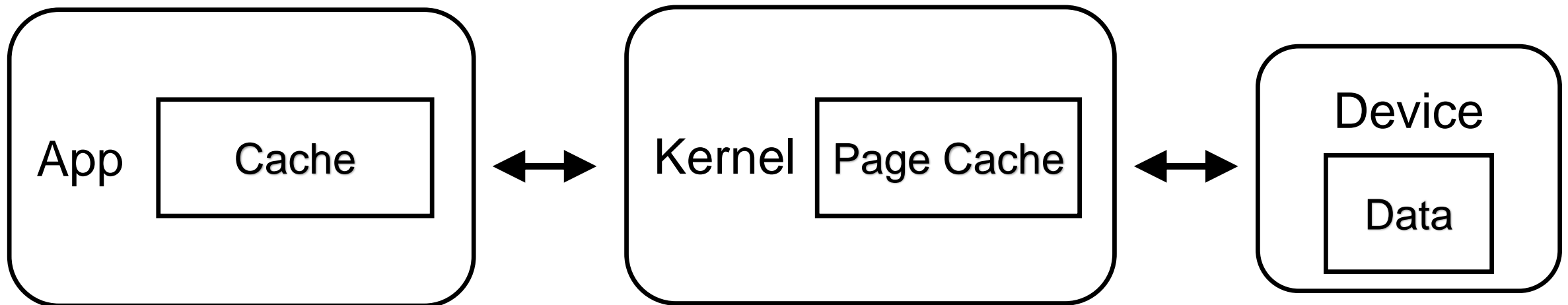


Caching in Storage Engine



Cache within the application

An implicit layer of kernel caching



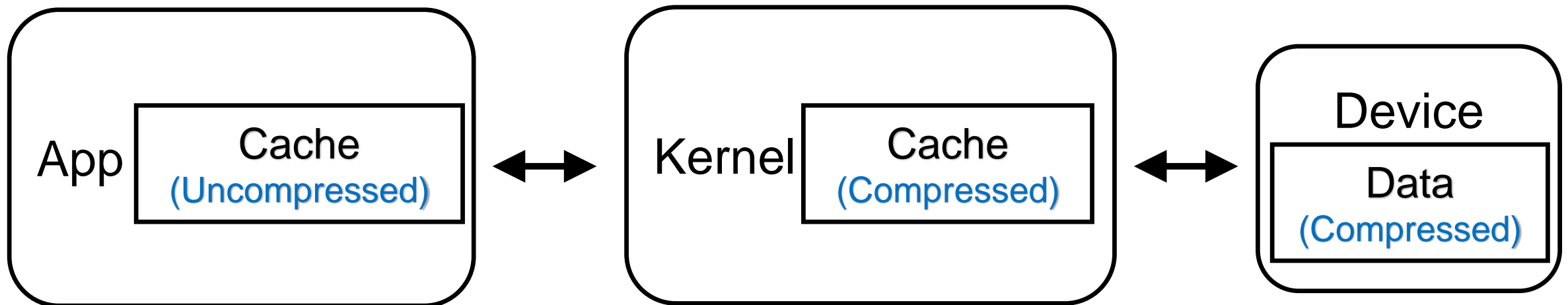
Caching in Storage Engine



Cache within the application

An implicit layer of kernel caching

Stores compressed data





Caching in Storage Engine

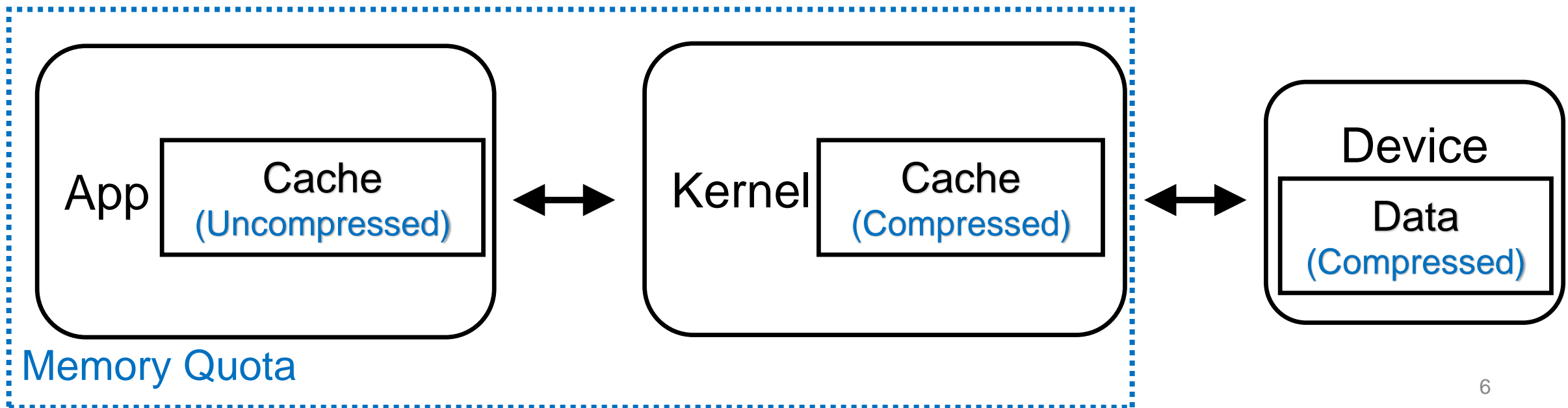
Forms a special two-layer cache structure

Sharing of memory quota

Optimal cache partitioning is important and not trivial

Yields great benefit

Depends on various factors



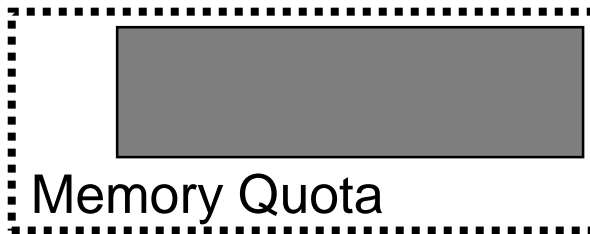
Optimal Cache Partitioning is Hard



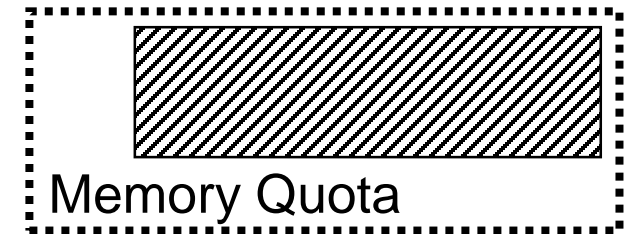
■ App Cache ▨ Kernel Cache

Configuration

#1: All to **app cache**



#2: All to **kernel cache**



Performance with
small working set size

Performance with
large working set size

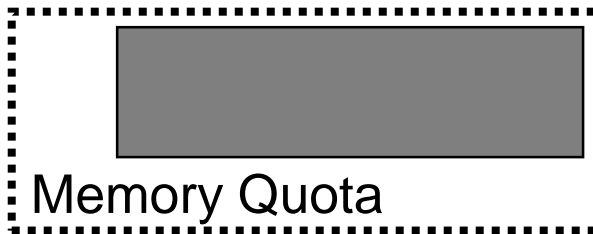
Optimal Cache Partitioning is Hard



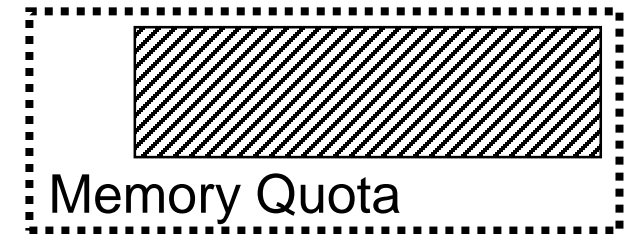
■ App Cache ▨ Kernel Cache

Configuration

#1: All to **app cache**



#2: All to **kernel cache**



Performance with
small working set size

2



1

Performance with
large working set size

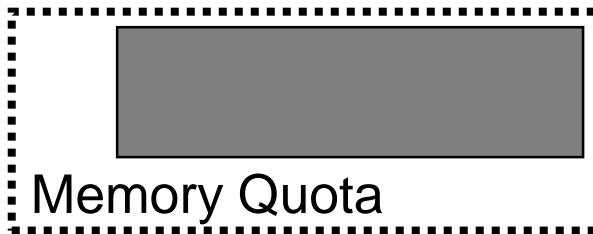
Optimal Cache Partitioning is Hard



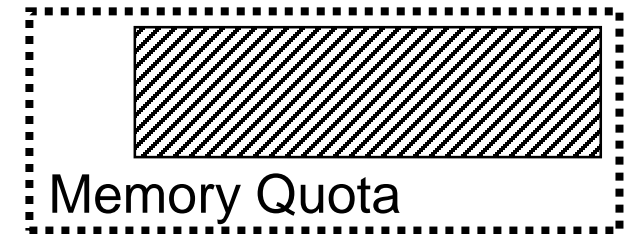
■ App Cache ▨ Kernel Cache

Configuration

#1: All to **app cache**



#2: All to **kernel cache**



Performance with
small working set size

2

•

1

Performance with
large working set size

1

•

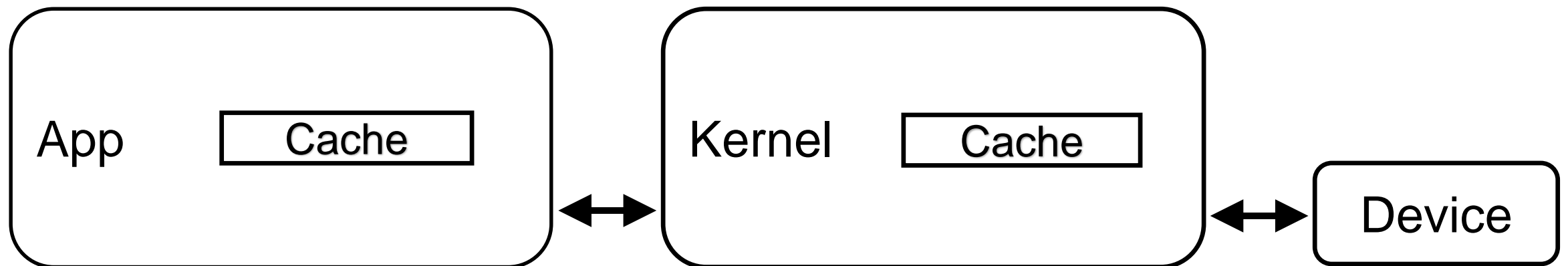
5

The best partition could be arbitrary percentage in the middle

Symbiosis - Adapt Cache Sizes by Simulation



Symbiosis

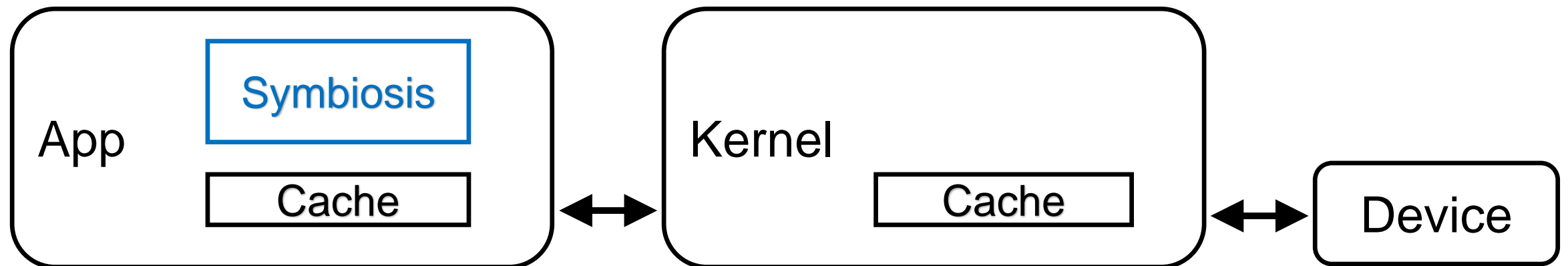


Symbiosis - Adapt Cache Sizes by Simulation



Symbiosis

Embedded in the storage engine



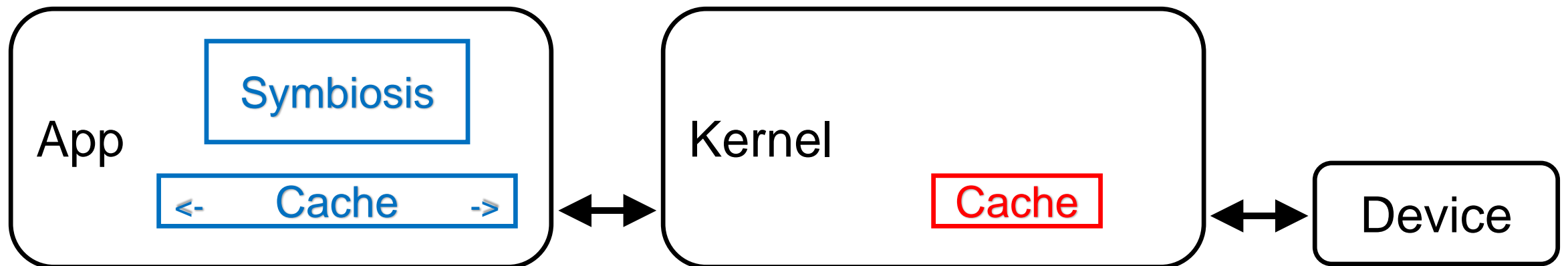
Symbiosis - Adapt Cache Sizes by Simulation



Symbiosis

Embedded in the storage engine

Optimizes cache partitioning automatically



Symbiosis - Adapt Cache Sizes by Simulation

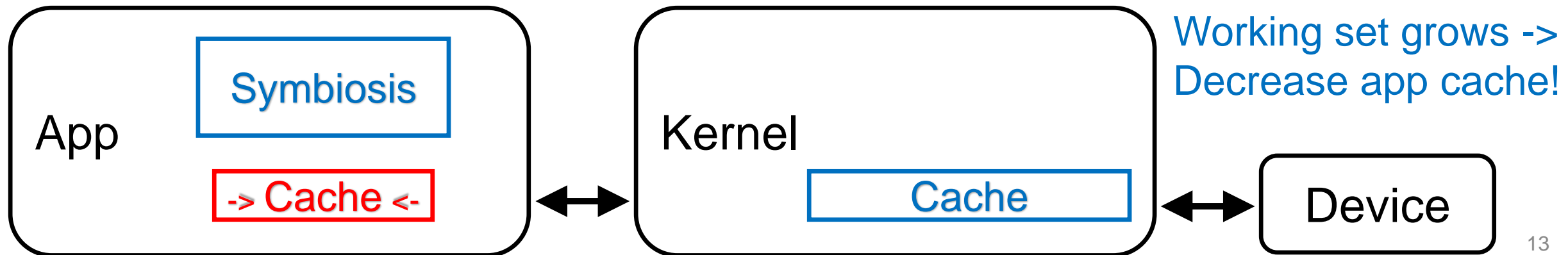


Symbiosis

Embedded in the storage engine

Optimizes cache partitioning automatically

Adapts cache sizes to dynamic workloads



Symbiosis - Adapt Cache Sizes by Simulation



Integrated into production systems with <1000 LoC

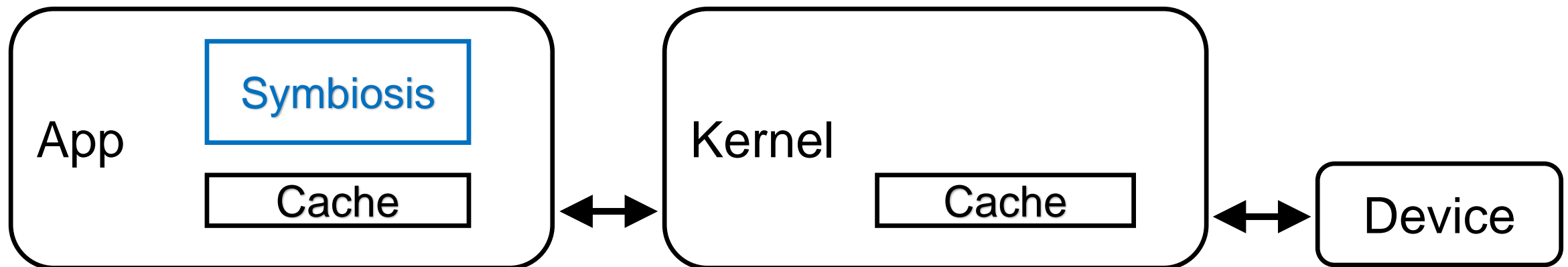
LevelDB, RocksDB, WiredTiger

Performance improvements

1.5x on average for read-heavy workloads

Online cache simulation with high accuracy and negligible overhead

~0.1% space overhead and ~1% time overhead



Outline



Overview of Symbiosis

Key Challenge

- Simulate accurately with low overhead

Optimization Techniques

- Incremental reuse of a single ghost cache

- Misalignment-aware sampling

- Guard against unmodeled

Evaluation

- Static workloads

- Dynamic (changing) workloads

Symbiosis - Overview



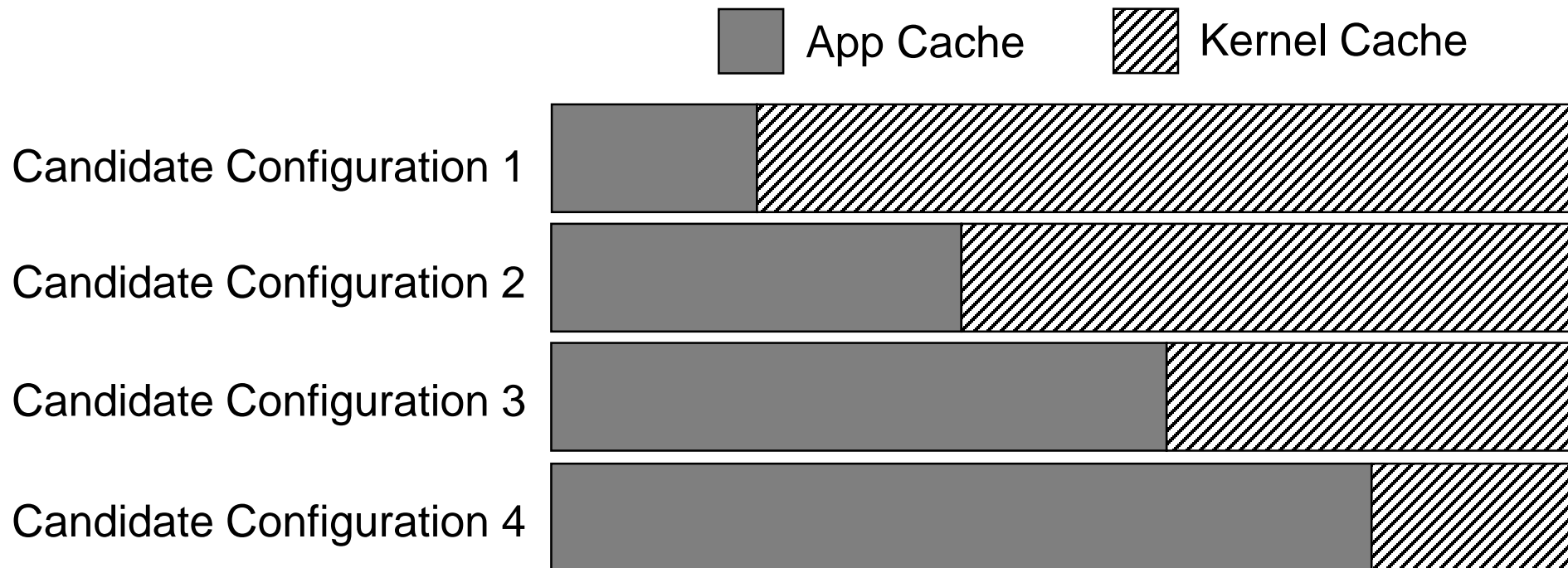
Detects workload change

Symbiosis - Overview



Detects workload change

Simulates multiple candidate cache size configurations



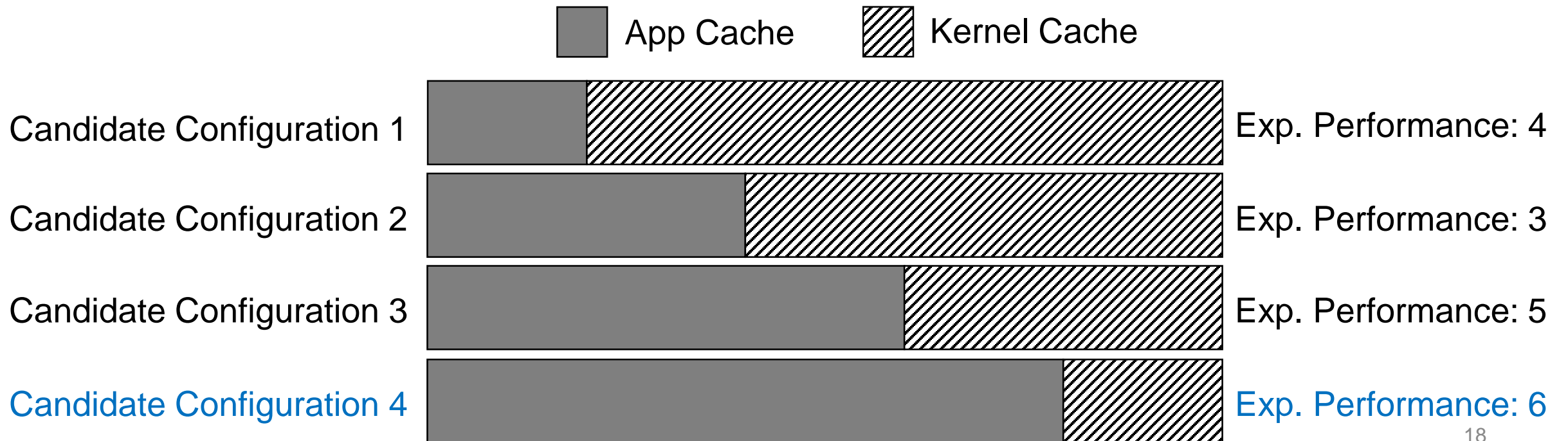
Symbiosis - Overview



Detects workload change

Simulates multiple candidate cache size configurations

Finds the configuration with the best expected performance



Symbiosis - Overview

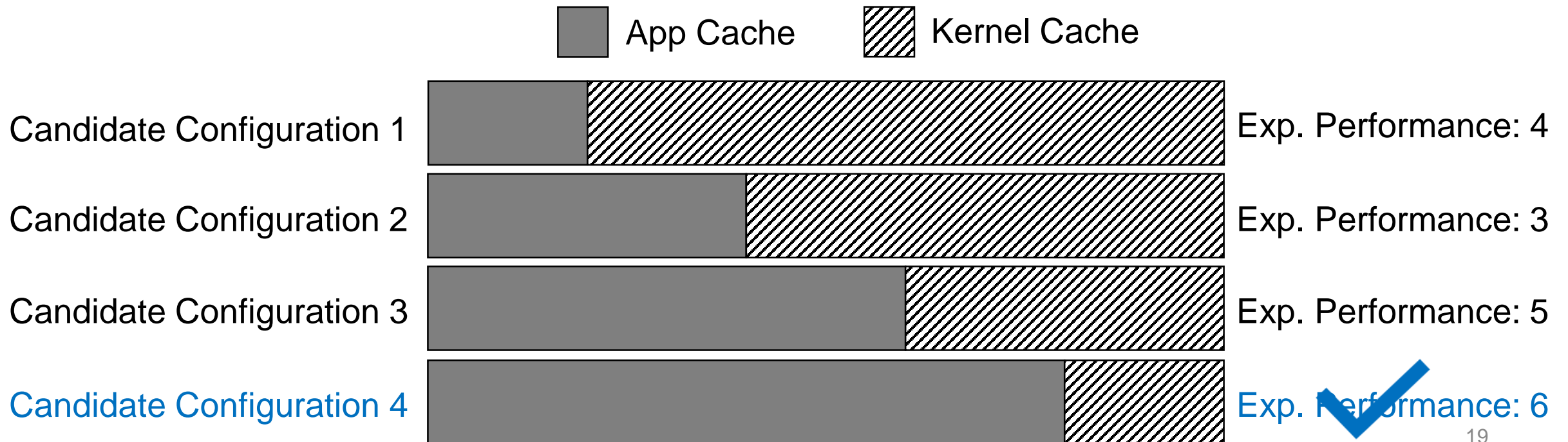


Detects workload change

Simulates multiple candidate cache size configurations

Finds the configuration with the best expected performance

Applies the best configuration if it yields enough benefit



Key Challenge



How to simulate accurately?

How to simulate with negligible overhead?

Key Challenge



How to simulate accurately?



Tension!

How to simulate with negligible overhead?

Key Challenge



How to simulate accurately?



Tension!

How to simulate with negligible overhead?

Optimization Techniques

- Incremental reuse of a single ghost cache

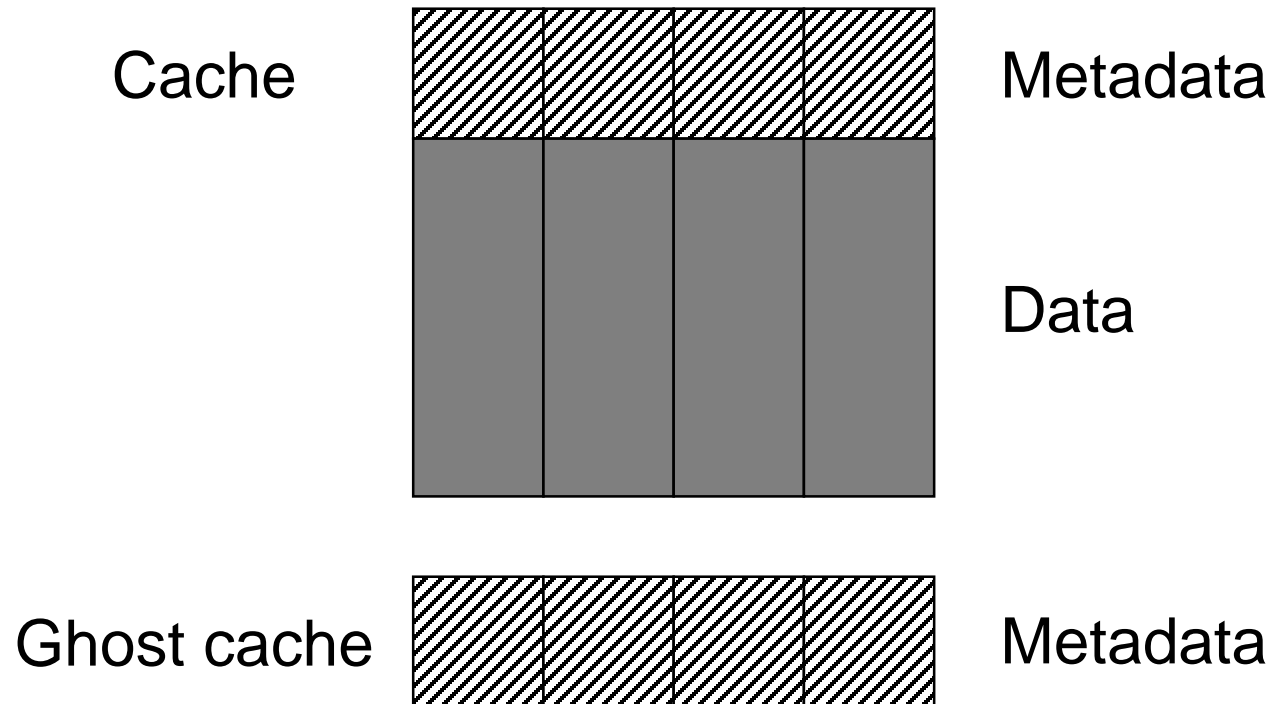
- Misalignment-aware sampling

- Guard against unmodeled

Classical Solution - Ghost Cache Simulation



Ghost cache - maintain only cache access metadata
Useful for cache statistics analysis



Ghost Cache Simulation - Single Layer

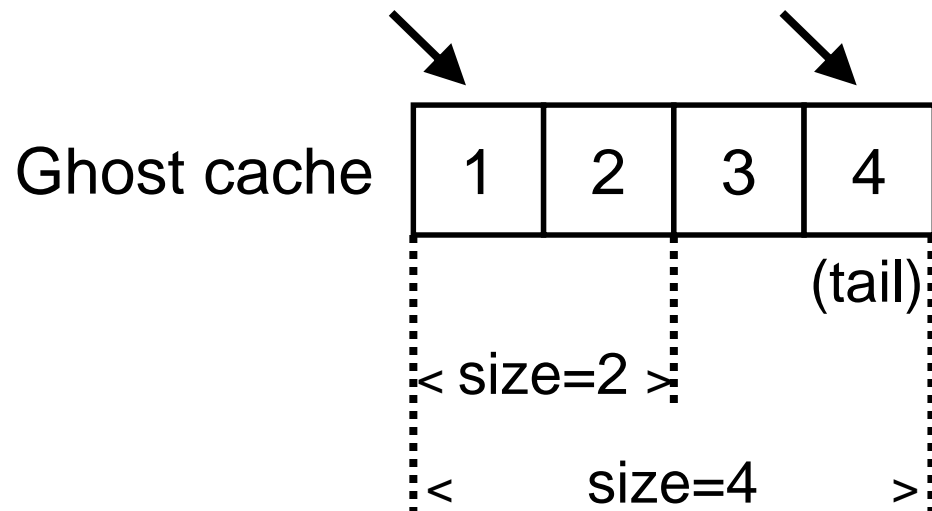


Use a single ghost cache instance

Largely reduces space and time overhead

Simulate caches of multiple sizes simultaneously

When the eviction policy has stack property



Ghost Cache Simulation - Two Layers



Simultaneous simulation of caches with different sizes is **not feasible**
The second layer sees different access patterns

Ghost Cache Simulation - Two Layers



Simultaneous simulation of caches with different sizes is **not feasible**
The second layer sees different access patterns

1st-layer access trace: 2 1 3 2 3 1 3 1 2 4
2 1 3 2 3 1 2 1 3 4
2 1 3 2 1 3 1 3 2 4

1st-layer Ghost cache size = 1

1

Ghost Cache Simulation - Two Layers



Simultaneous simulation of caches with different sizes is **not feasible**
The second layer sees different access patterns

1st-layer access trace: 2 1 3 2 3 1 3 1 2 4
2 1 3 2 3 1 2 1 3 4
2 1 3 2 1 3 1 3 2 4

2nd-layer access trace: 2 1 3 2 3 1 3 1 2 4
2 1 3 2 3 1 2 1 3 4
2 1 3 2 1 3 1 3 2 4

1st-layer Ghost cache size = 1

1

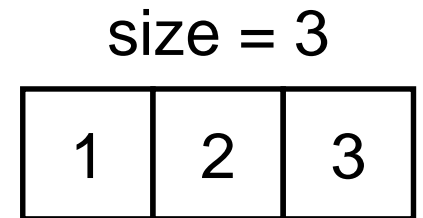
Ghost Cache Simulation - Two Layers



Simultaneous simulation of caches with different sizes is **not feasible**
The second layer sees different access patterns

1st-layer access trace: 2 1 3 2 3 1 3 1 2 4
2 1 3 2 3 1 2 1 3 4
2 1 3 2 1 3 1 3 2 4

1st-layer Ghost cache



2nd-layer access trace: 4 3 4 2 4

Incremental reuse of a single ghost cache



Use a single ghost cache instance

Largely reduces space and time overhead

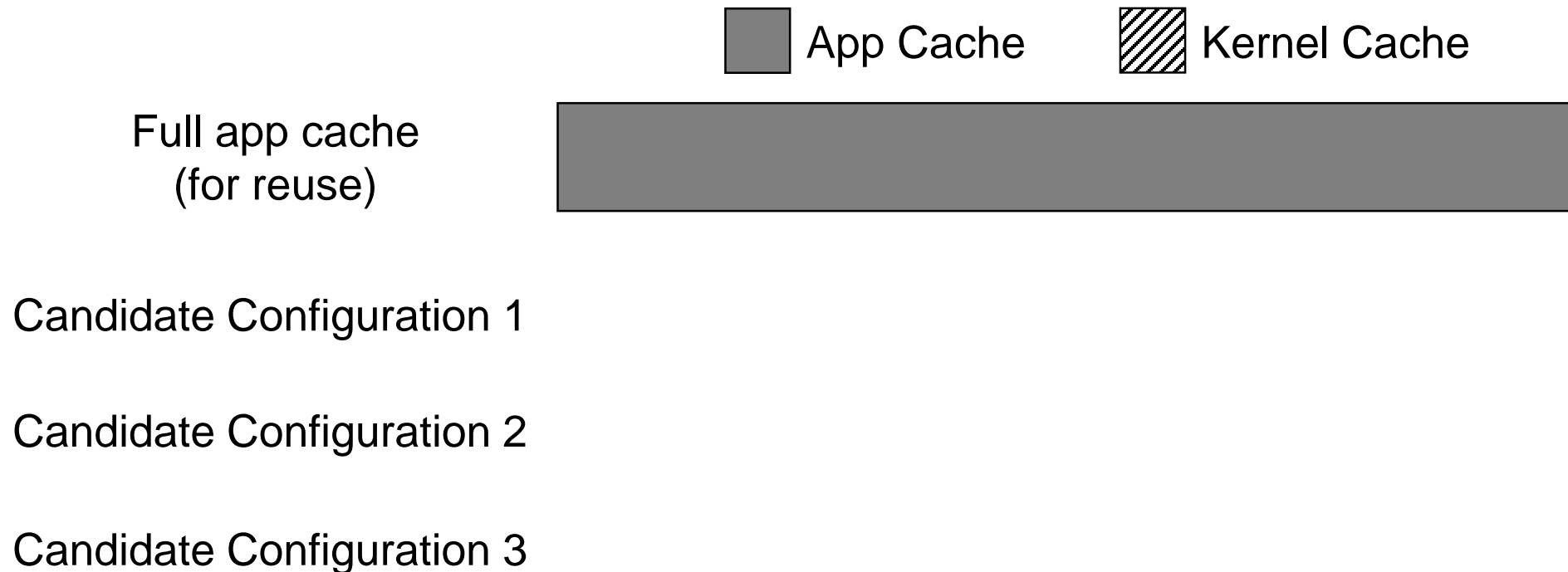
Incremental reuse of a single ghost cache



Use a single ghost cache instance

Largely reduces space and time overhead

Simulate candidates one at a time



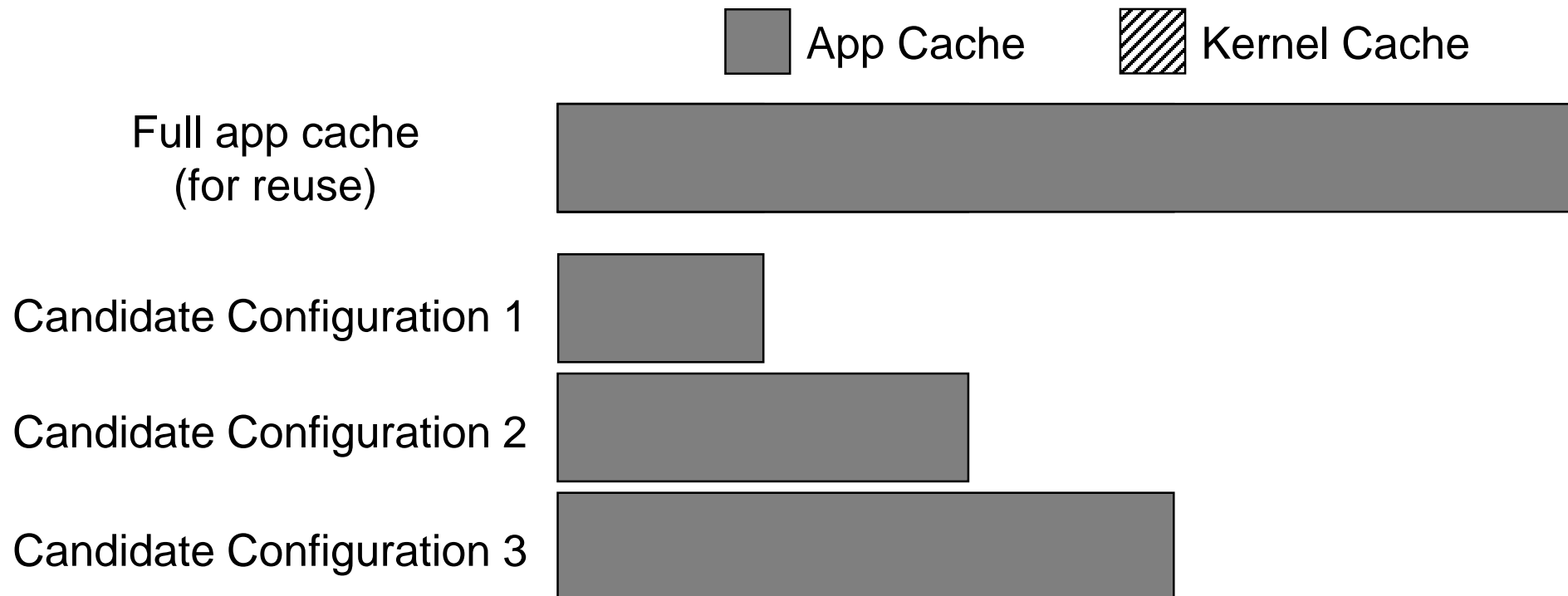
Incremental reuse of a single ghost cache



Use a single ghost cache instance

Largely reduces space and time overhead

Simulate candidates one at a time





Incremental reuse of a single ghost cache

Use a single ghost cache instance

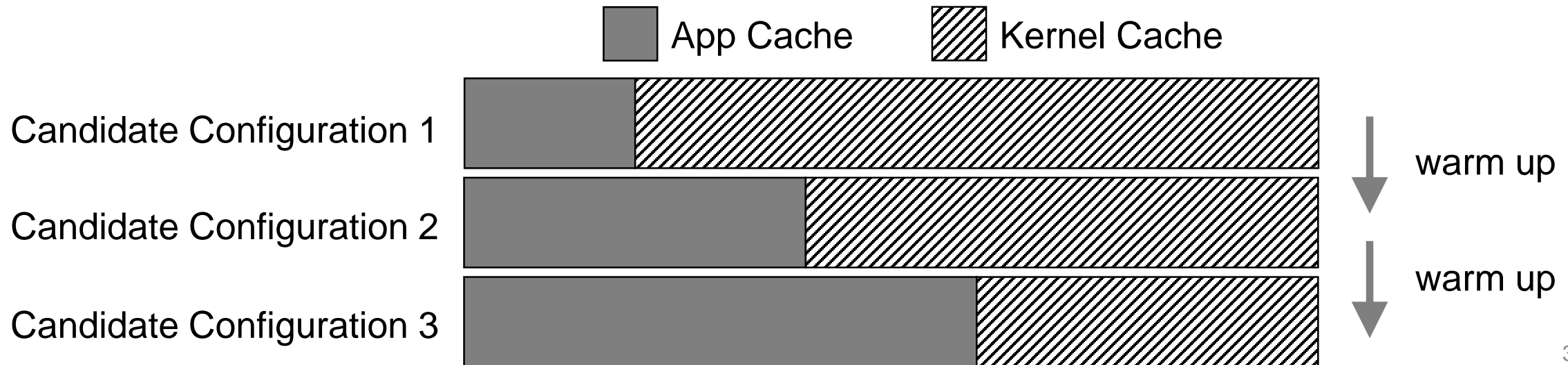
Largely reduces space and time overhead

Simulate candidates one at a time

Minimize warm-up before each simulation

Utilize the stack property for the first layer

Simulate in the order of increasing application cache size





Incremental reuse of a single ghost cache

Use a single ghost cache instance

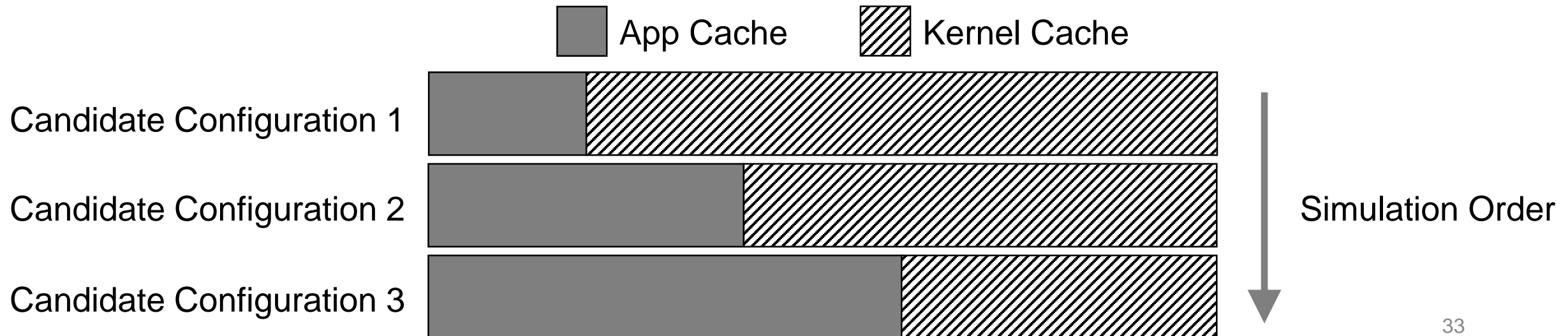
Largely reduces space and time overhead

Simulate candidates one at a time

Minimize warm-up before each simulation

Utilize the stack property for the first layer

Simulate in the order of increasing application cache size

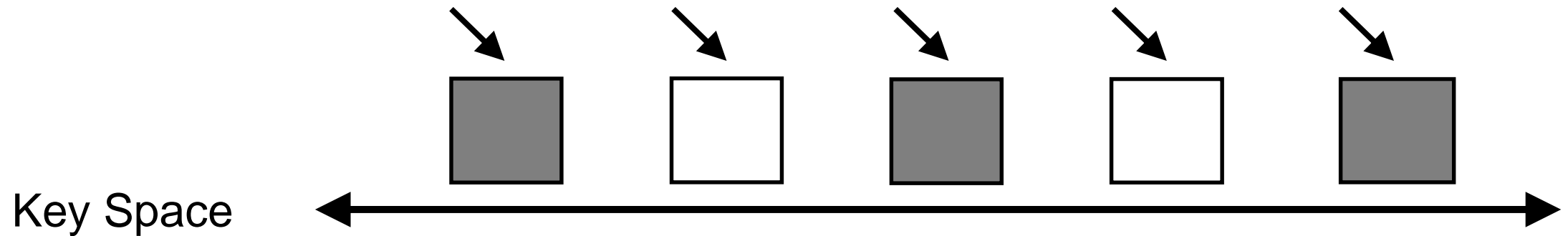


Misalignment-aware Sampling



Key-space sampling

Sample targets in key space and all accesses to the targets



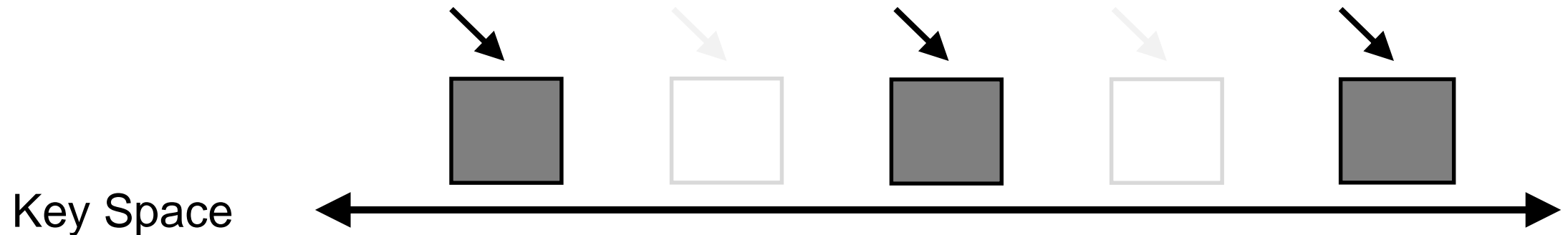
Misalignment-aware Sampling



Key-space sampling

Samples targets in key space and all accesses to the targets

Works in normal multi-layer caching



Misalignment-aware Sampling

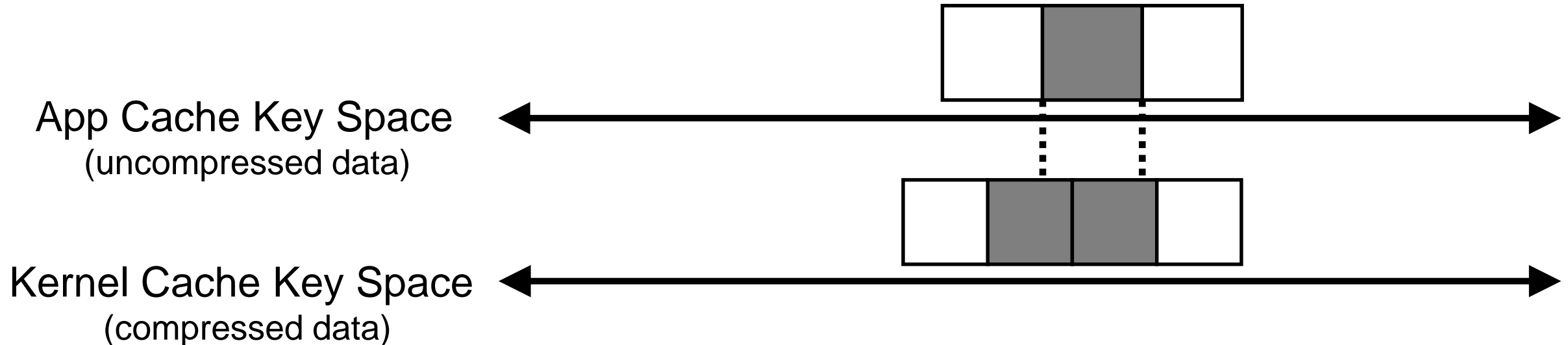


Misalignment - different caching units

The unit of app cache: app-defined blocks

The unit of kernel cache: pages

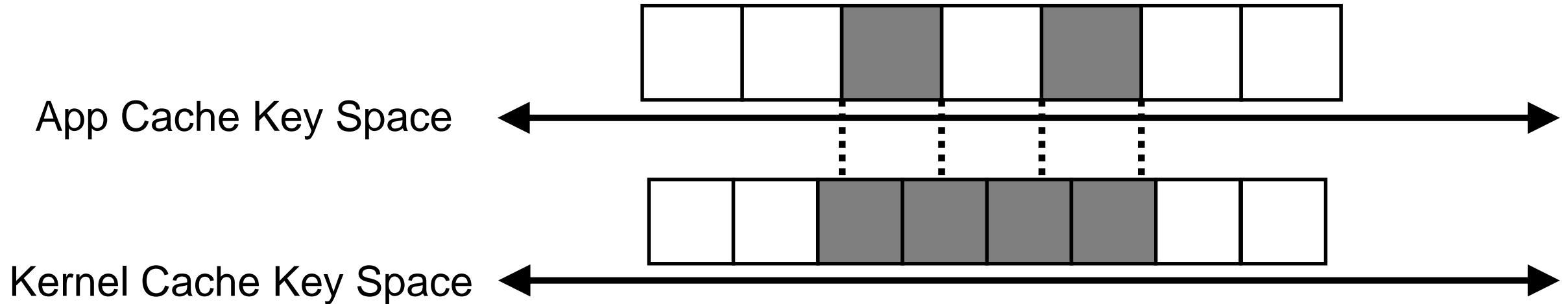
One unit in the 1st layer corresponds to multiple units in the 2nd layer



Misalignment-aware Sampling



Key-space sampling with misalignment

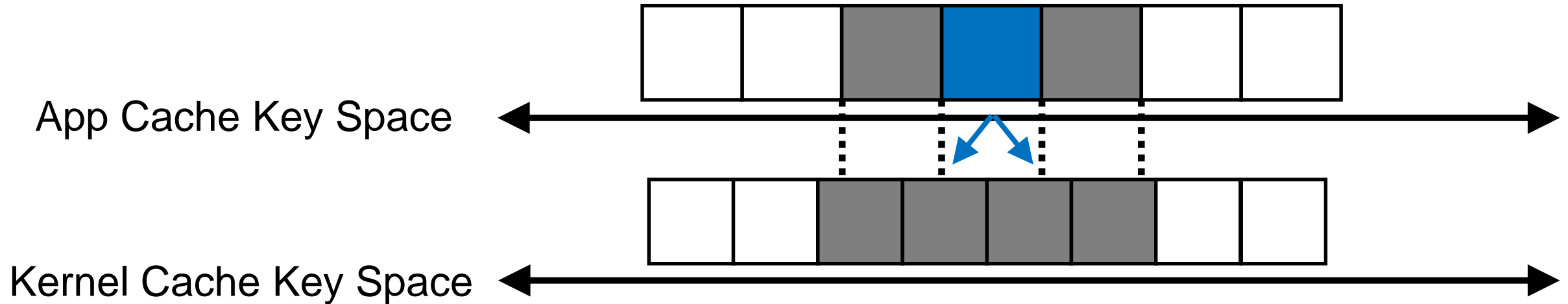


Misalignment-aware Sampling



Key-space sampling with misalignment

Hits in the 2nd layer due to accessed neighboring blocks in the 1st layer



Misalignment-aware Sampling

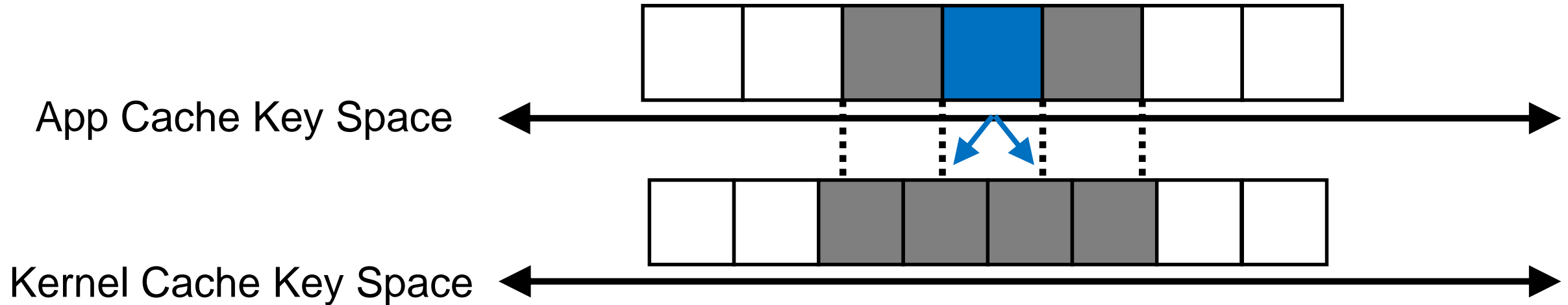


Key-space sampling with misalignment

Hits in the 2nd layer due to accessed neighboring blocks in the 1st layer

Contiguous targets not all sampled so hits are not counted

Loses the effects of accessing contiguous pages

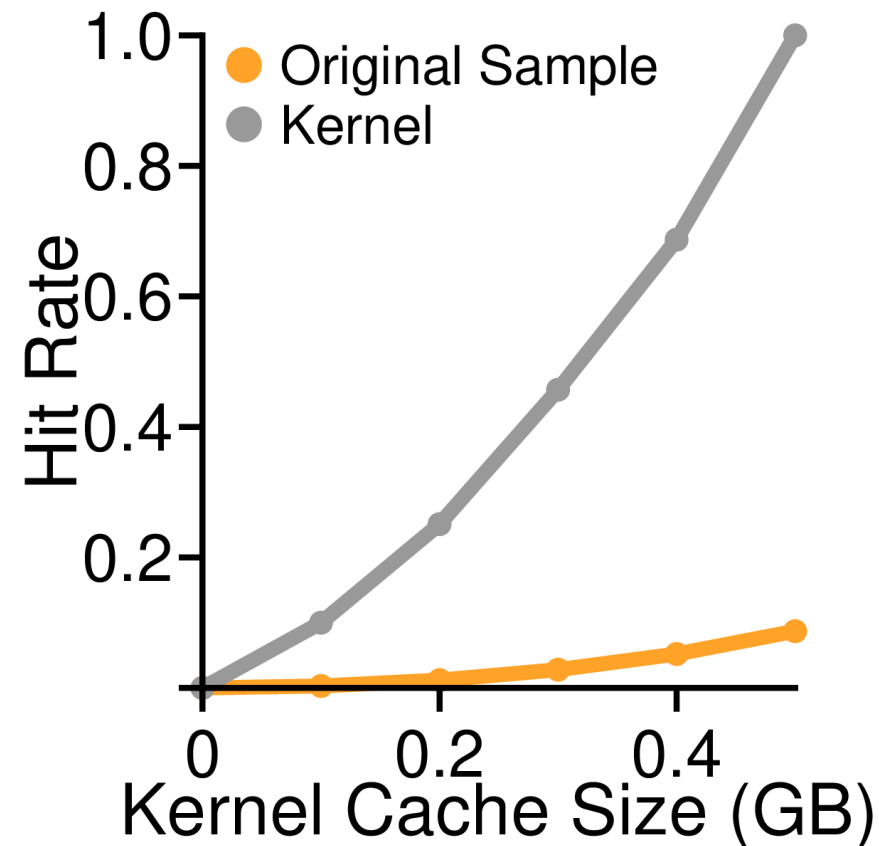


Misalignment-aware Sampling



Key-space sampling with misalignment

Much lower kernel cache hit rate

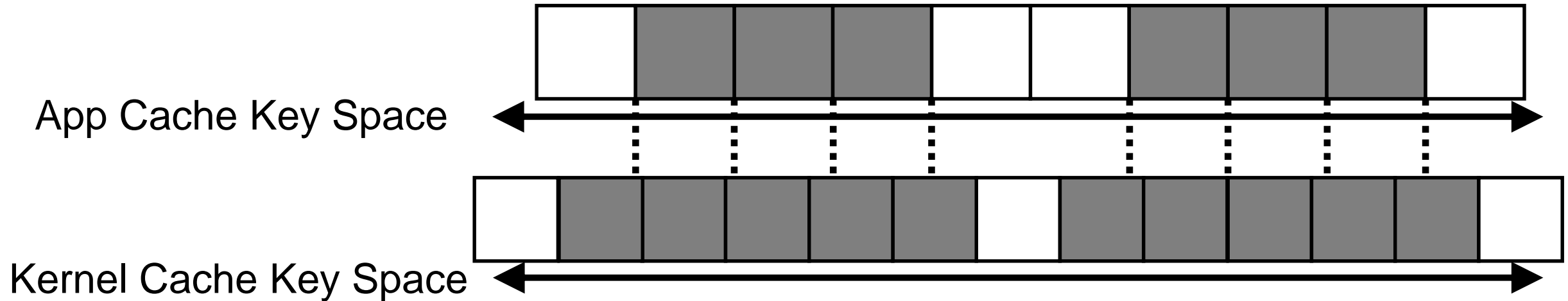


Misalignment-aware Sampling



Misalignment-aware Sampling

Sample contiguous targets together to keep the effect of misalignment



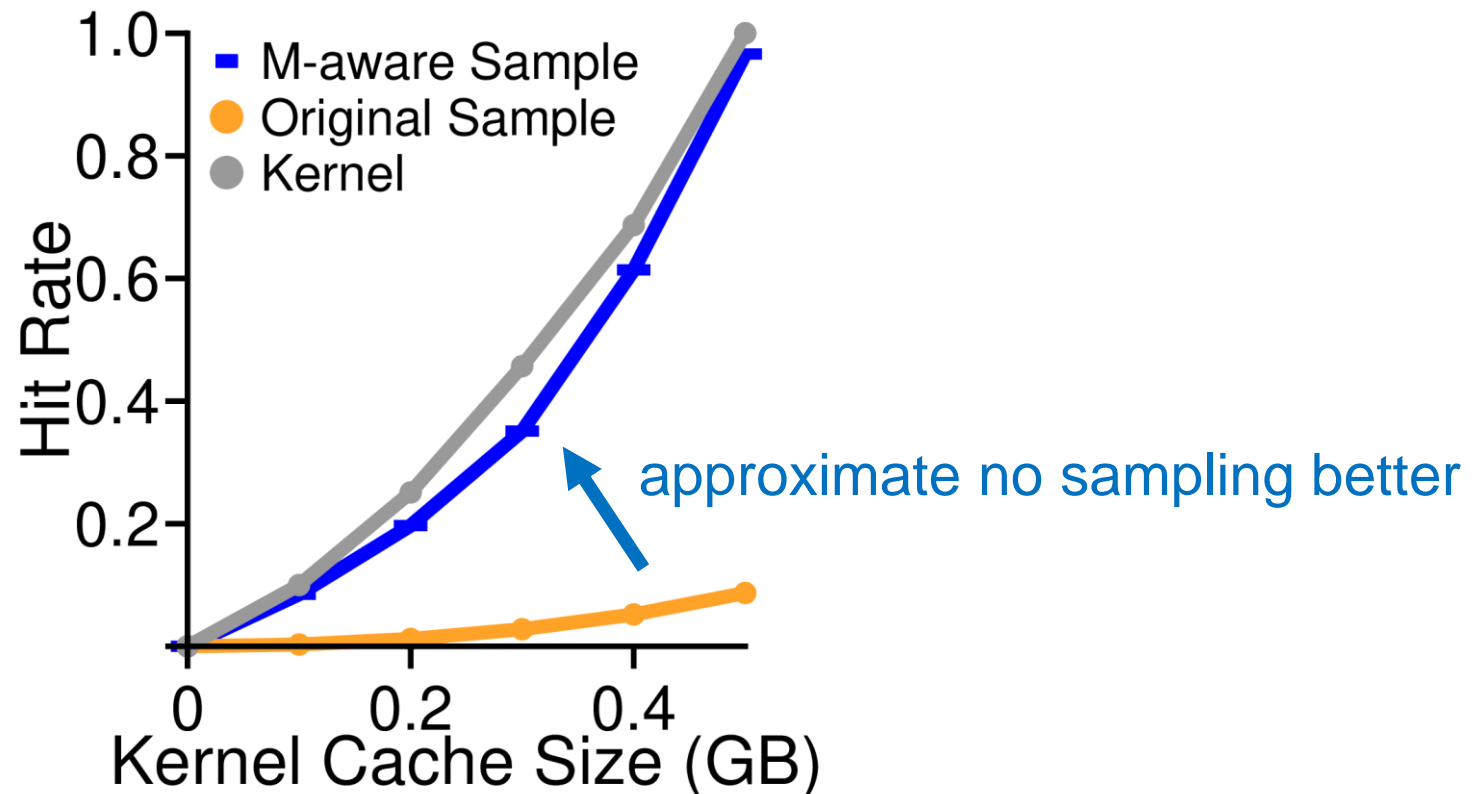
Misalignment-aware Sampling



Misalignment-aware Sampling

Sample contiguous targets together to keep the effect of misalignment
In Symbiosis, we sample in groups of 32 and the sampling rate is 1/64

Misalignment-aware sampling is much more accurate



Guard Against Unmodeled Cases



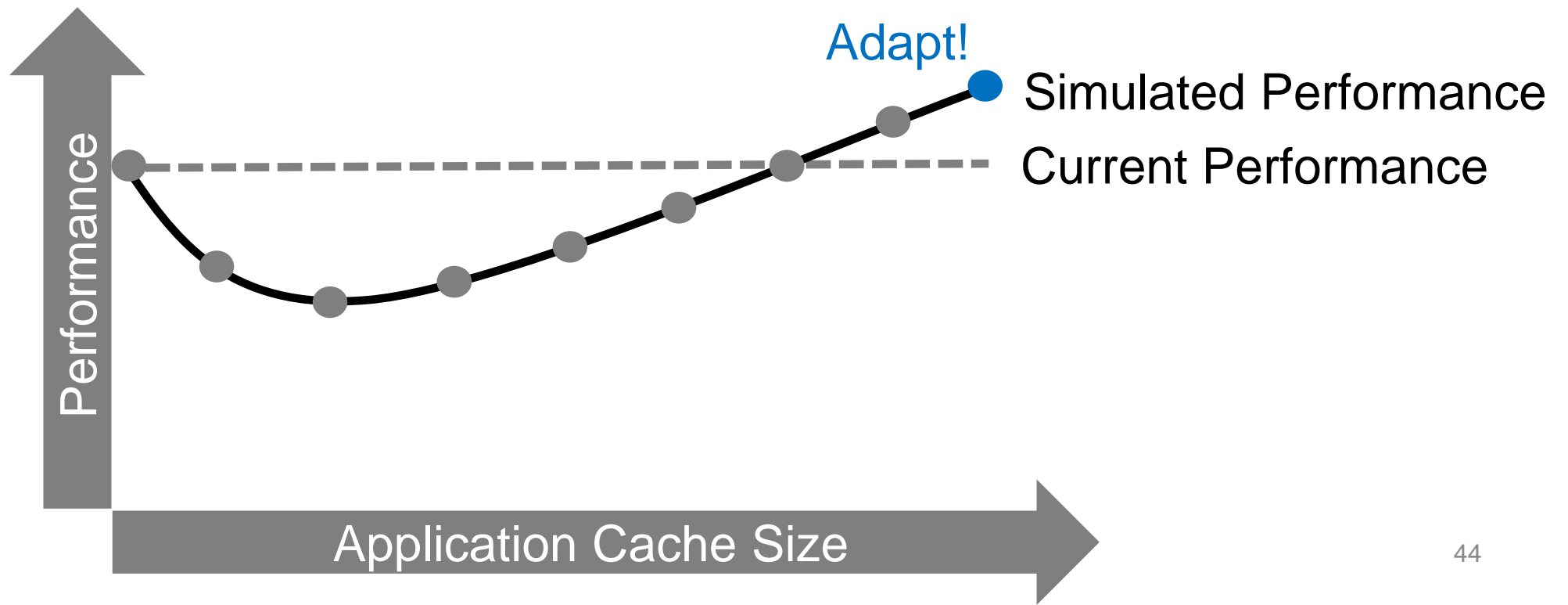
Some kernel features not modeled (e.g., read-ahead)

Guard Against Unmodeled Cases



Some kernel features not modeled (e.g., read-ahead)

When simulation is accurate



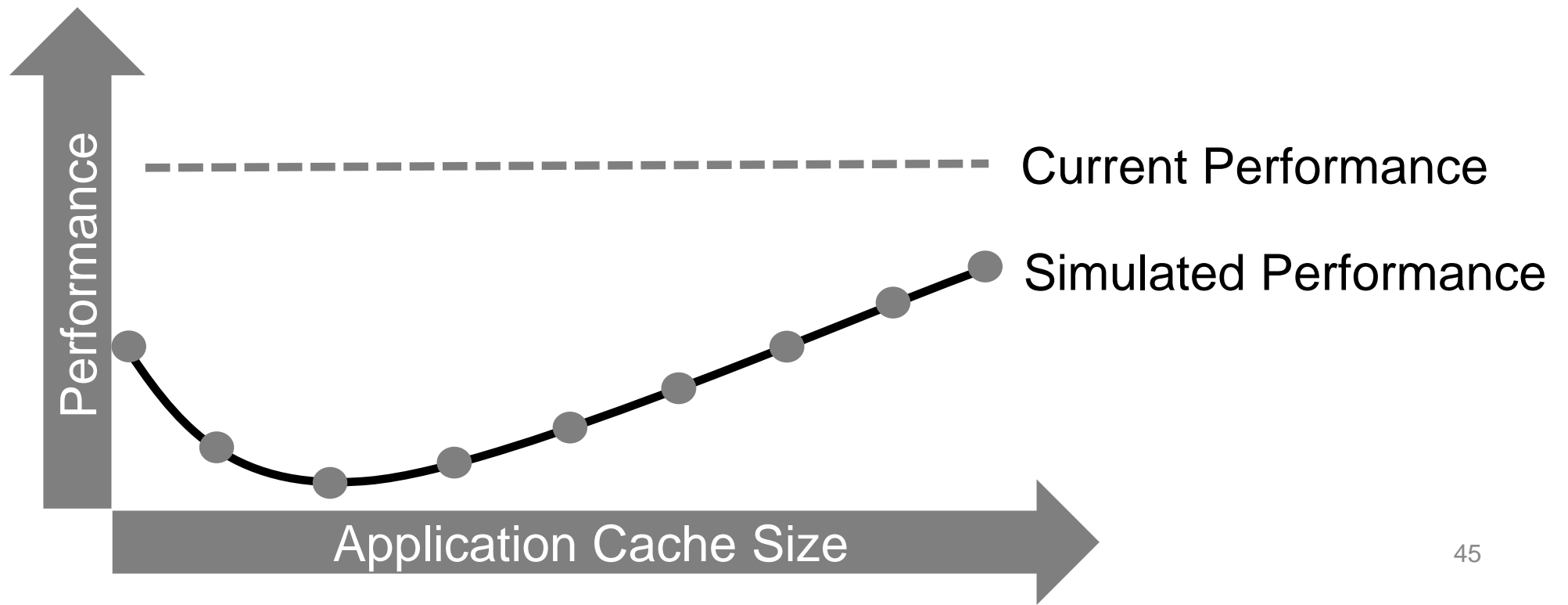
Guard Against Unmodeled Cases



Some kernel features not modeled (e.g., read-ahead)

When simulation is not accurate

Observation: omitting such features usually results in a worse perf.



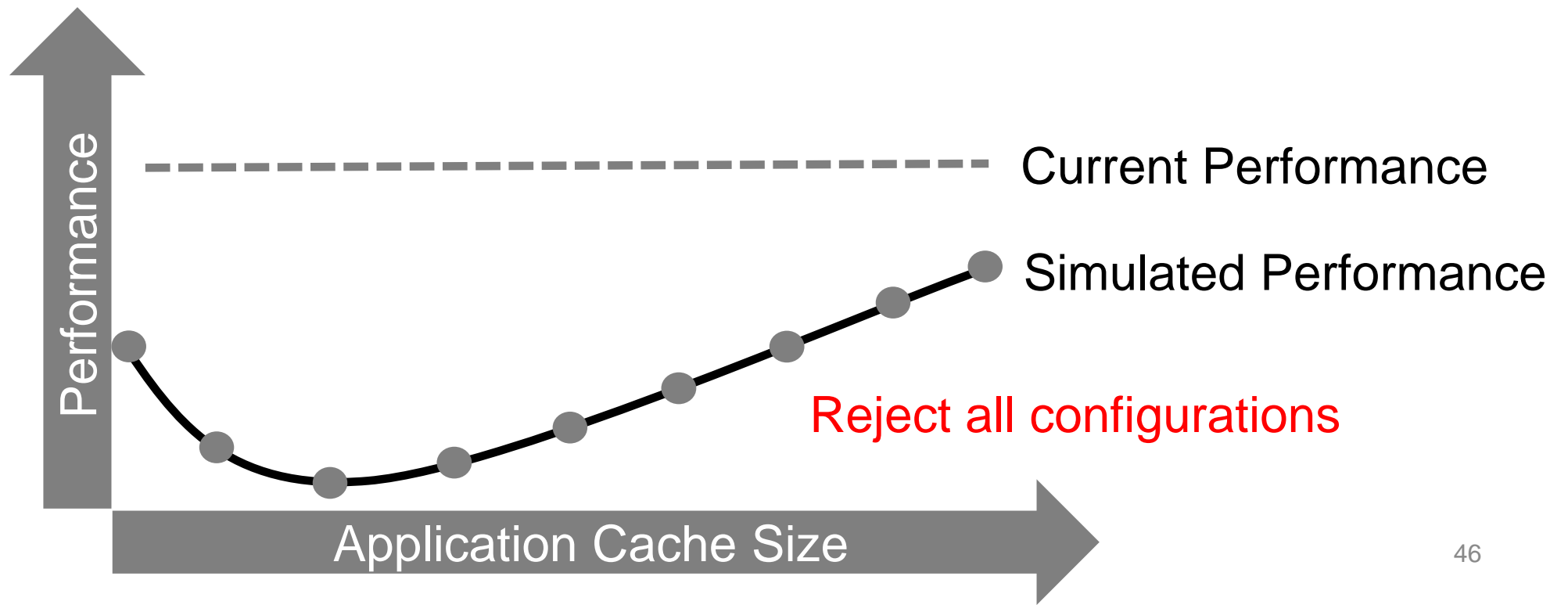
Guard Against Unmodeled Cases



Some kernel features not modeled (e.g., read-ahead)

Observation: omitting such features usually results in a worse perf.

Policy: only adapts when predicted perf. is better than current perf.



Evaluation - Static Workloads



Experiments

Performance with various workloads and environments

Factors		Presented Space	# of parameters
Workloads	Data Set Size	5, 2.5, 1.67, 1.24, 1	4
	Access Pattern	uniform, zipfian, hotspot{30,20,10}	5
Software	Compression Lib.	snappy (default), zstd	2
	Storage Engine	LevelDB, RocksDB, WiredTiger	3
Hardware	CPU Frequency	CPU1: 2.9GHz, CPU2: 2.0GHz	2
	Device Latency	SSD1: ~10us, SSD2: ~70us	2

Results

Improves performance in 98% of 190+ workloads by 1.5x on average

Evaluation - Dynamic Workloads



Experiments

- Performance with abrupt and gradual workload changes

- Performance with real world workloads

- Overhead and tail latency analysis

Results

- Symbiosis properly reacts to all of 38 workloads

- Symbiosis incurs 0.1% space overhead and 1% time overhead

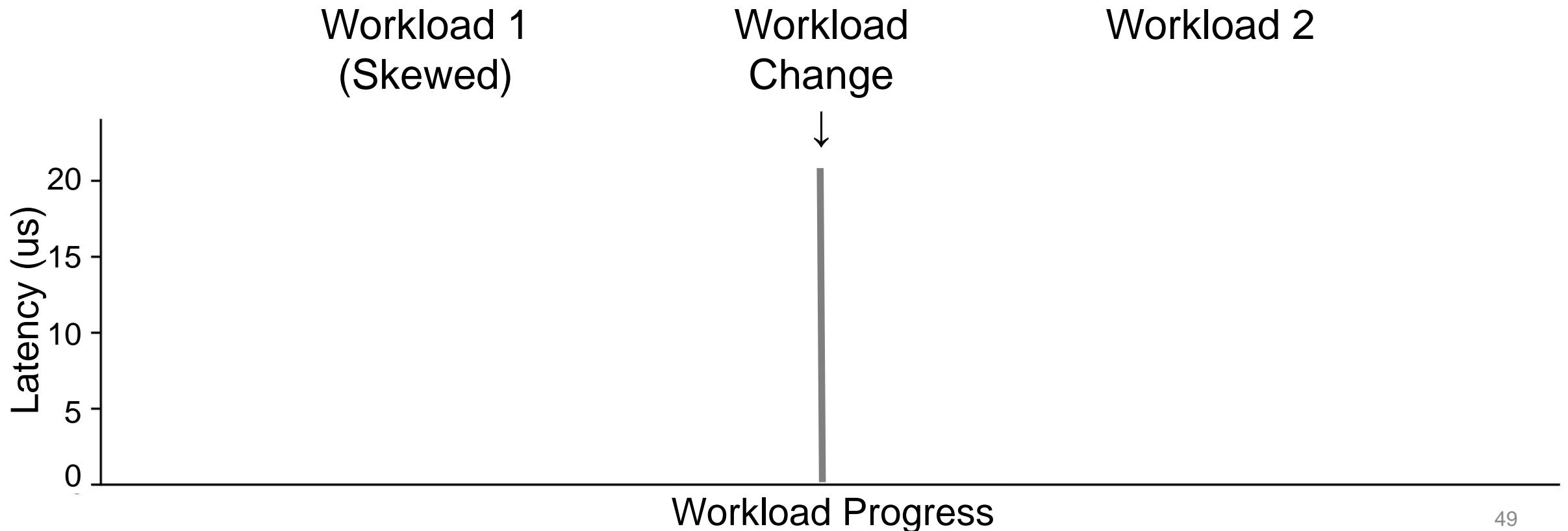
Dynamic Workloads - Example



Consists of 2 static workloads

Workload 1 is very skewed, while workload 2 is less skewed

Data set fits in memory when compressed, but not when uncompressed



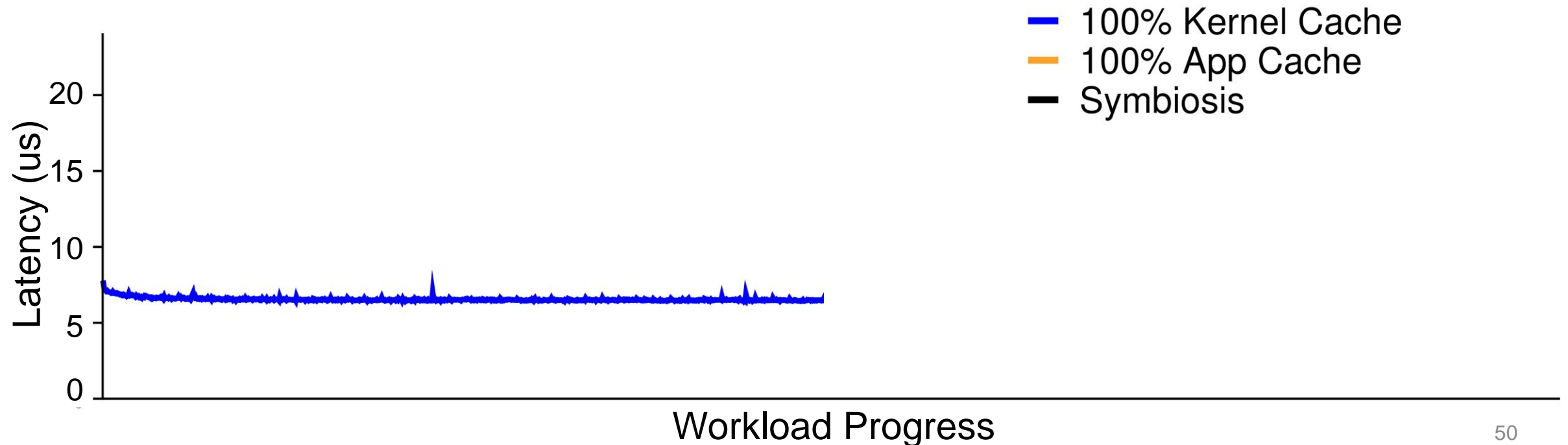
Dynamic Workloads - Example



Workload 1

Workload 1

Workload 2

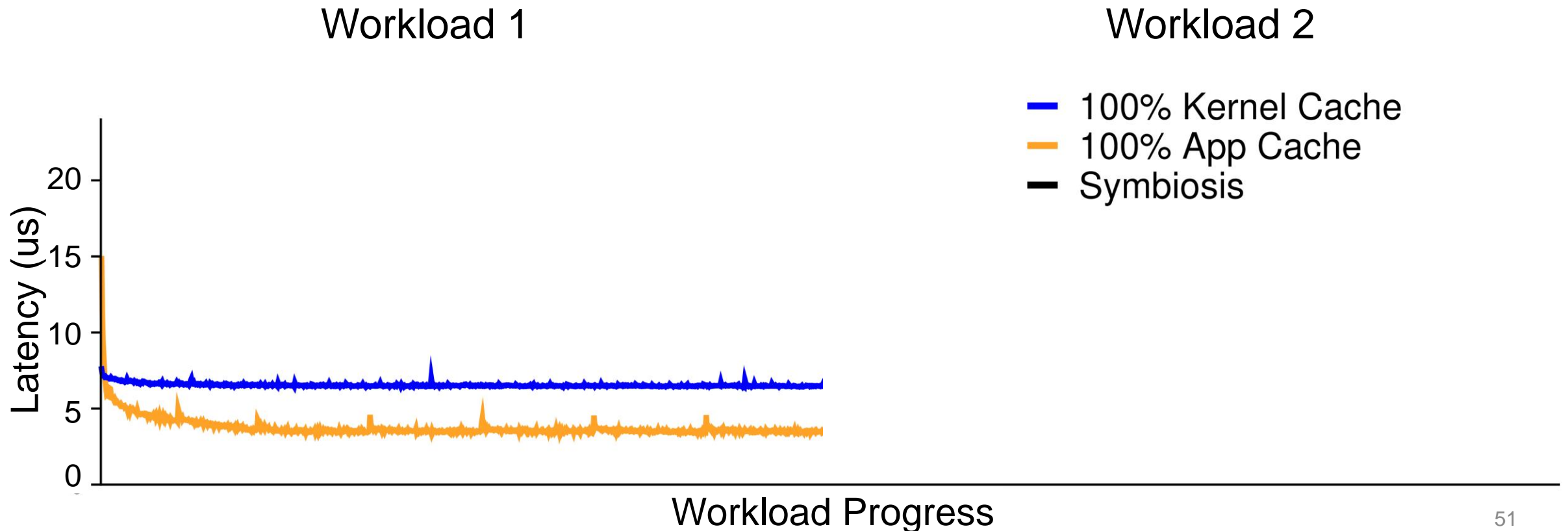


Dynamic Workloads - Example



Workload 1

100% app cache performs better due to high skewedness

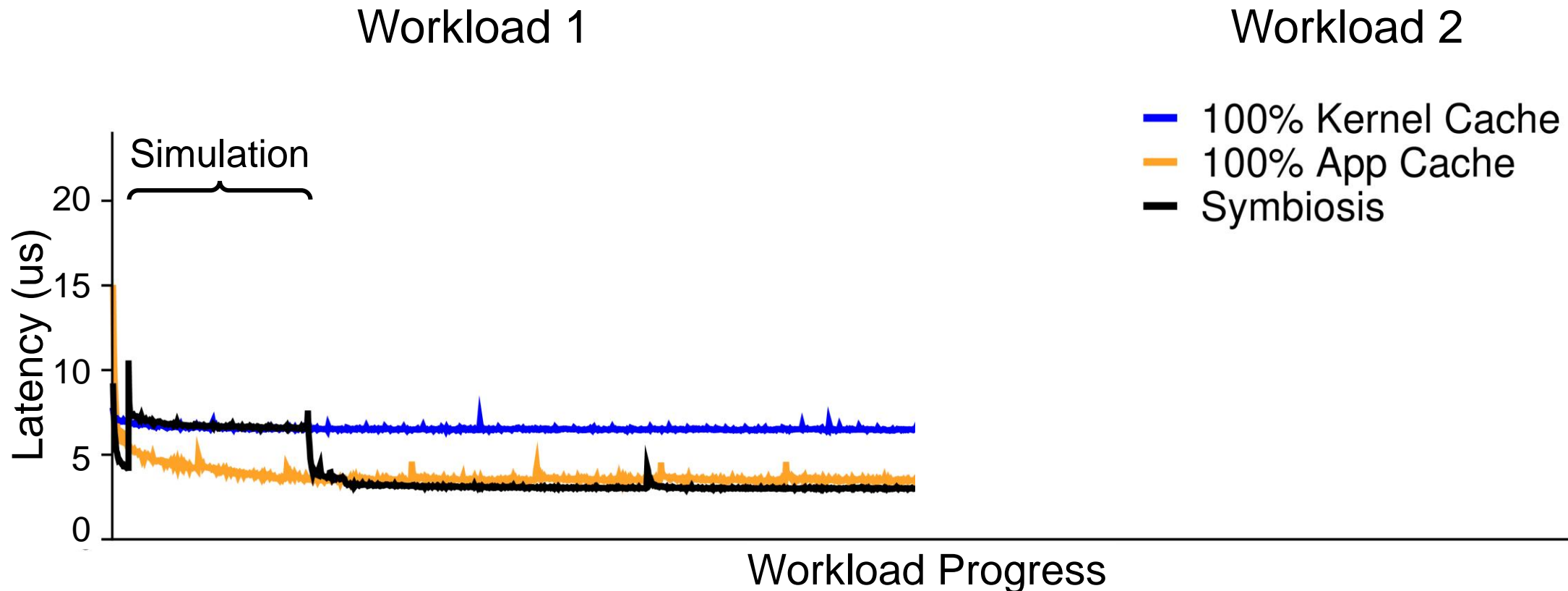


Dynamic Workloads - Example



Workload 1

Symbiosis performs best after simulation

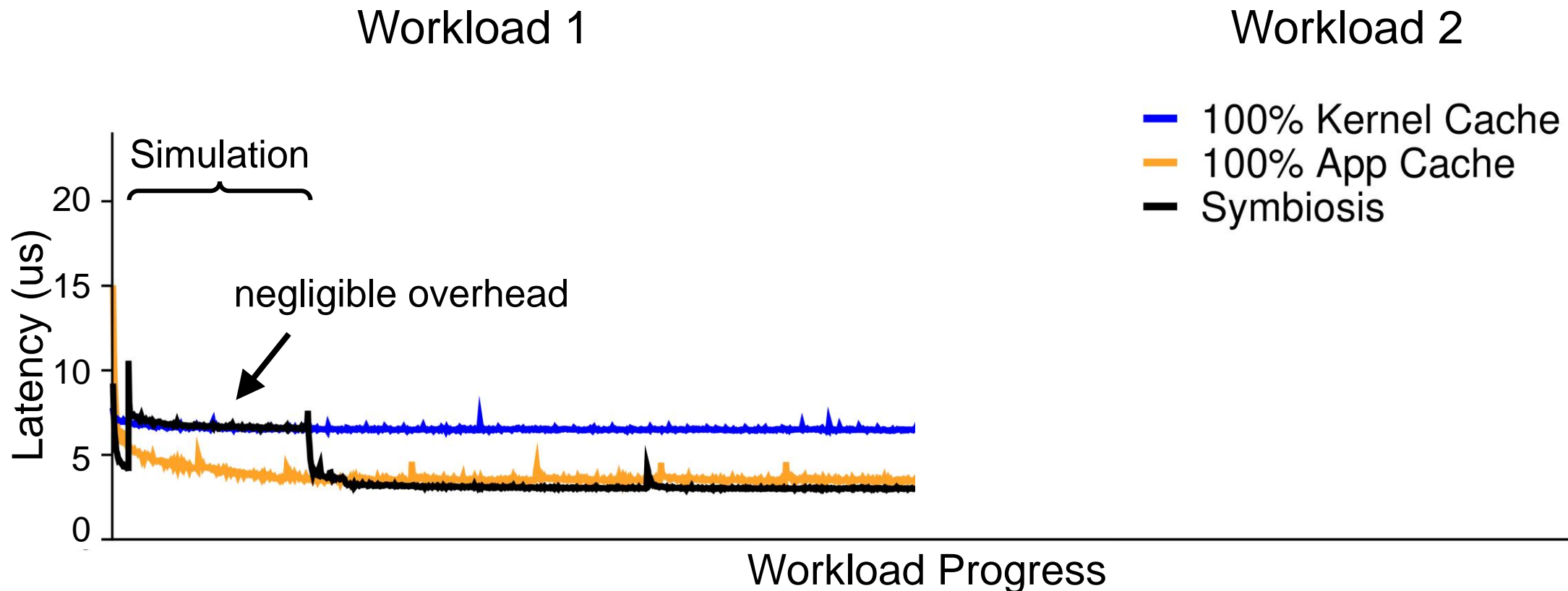


Dynamic Workloads - Example



Workload 1

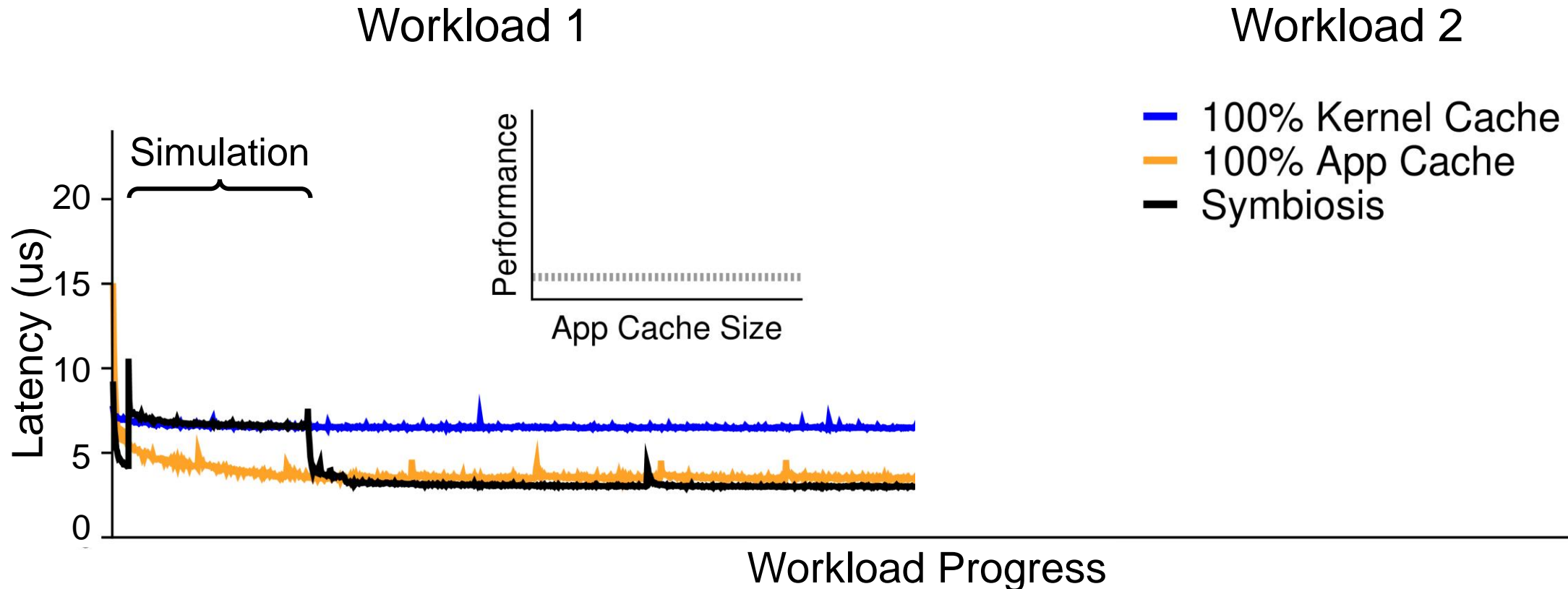
Symbiosis incurs negligible overhead during simulation



Dynamic Workloads - Example



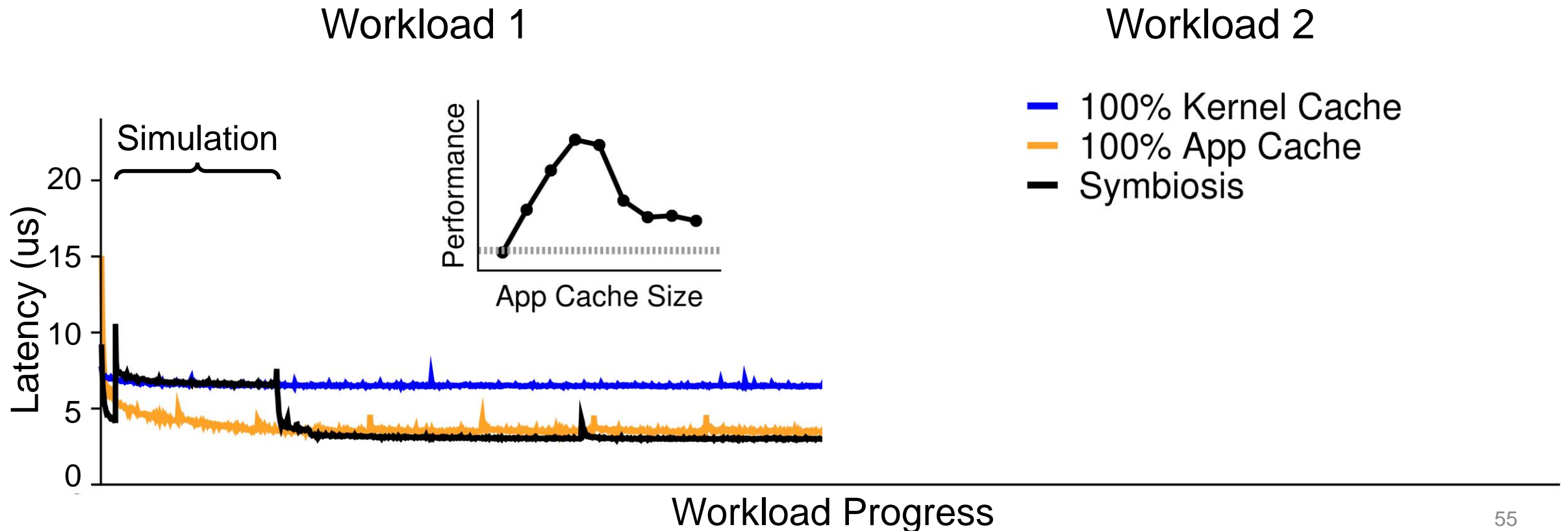
Workload 1



Dynamic Workloads - Example



Workload 1

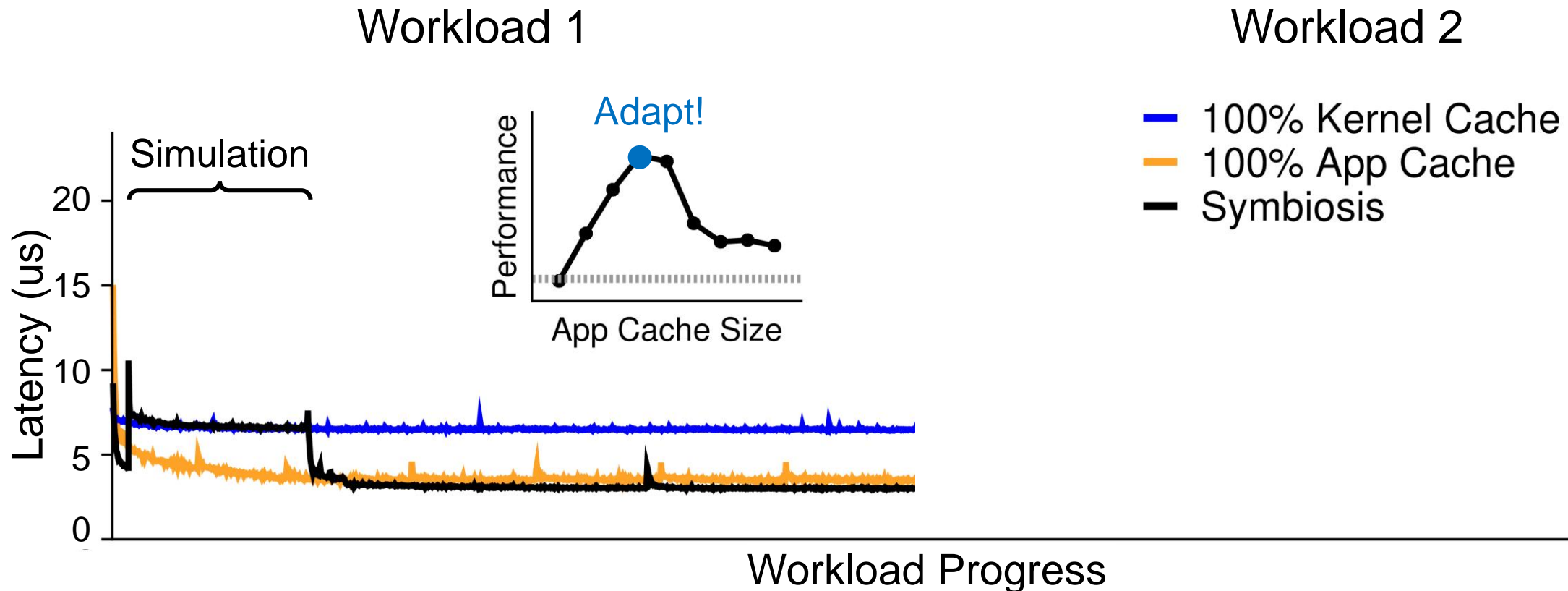


Dynamic Workloads - Example



Workload 1

Simulation result: giving ~40% of memory to app cache is the best

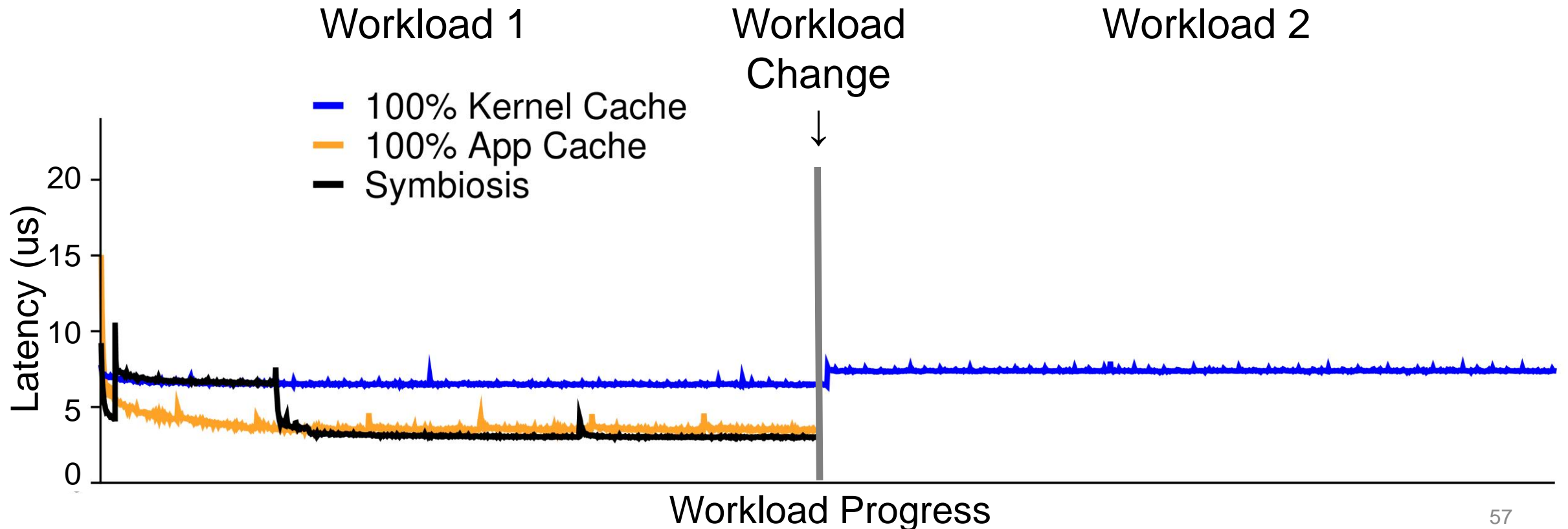


Dynamic Workloads - Example



Workload 2

100% kernel cache performs similarly

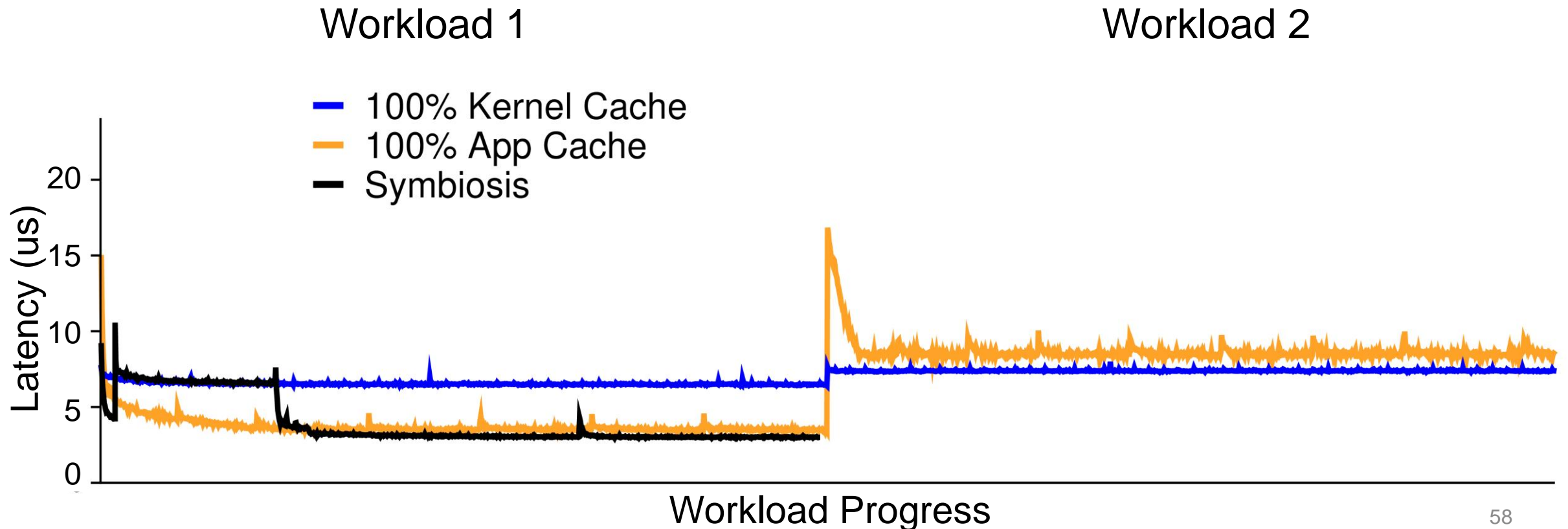


Dynamic Workloads - Example



Workload 2

100% app cache performs worse due to less skewedness

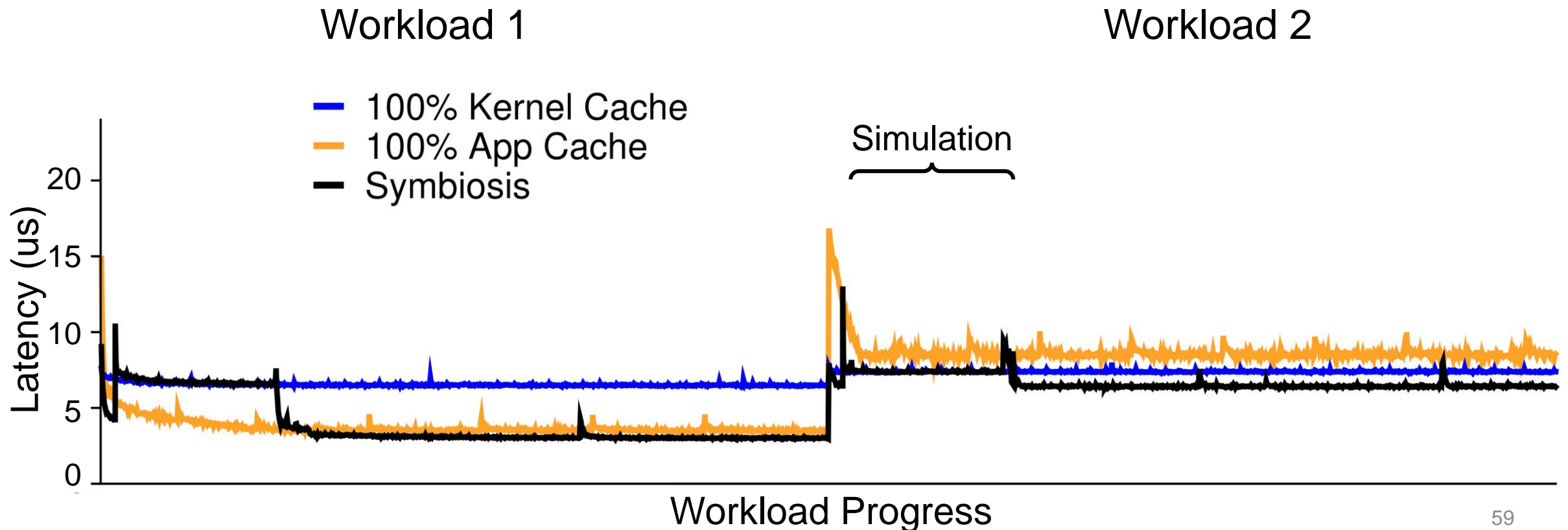


Dynamic Workloads - Example



Workload 2

Symbiosis maintains the best performance after simulation

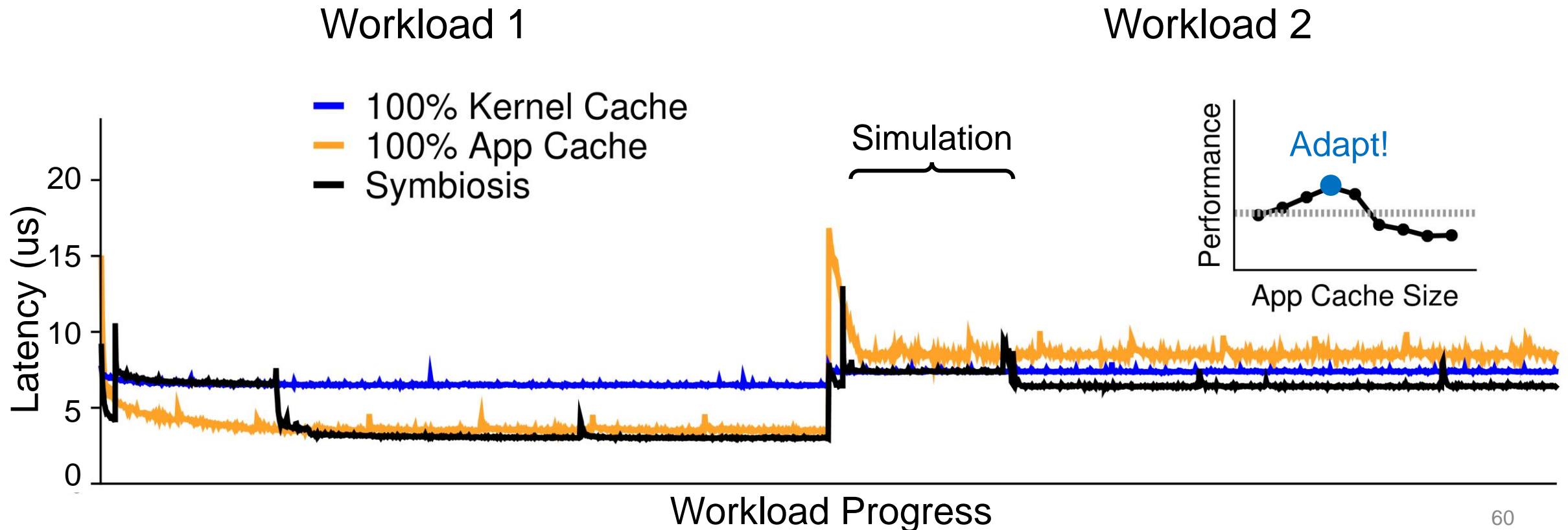


Dynamic Workloads - Example



Workload 2

Symbiosis maintains the best performance after simulation

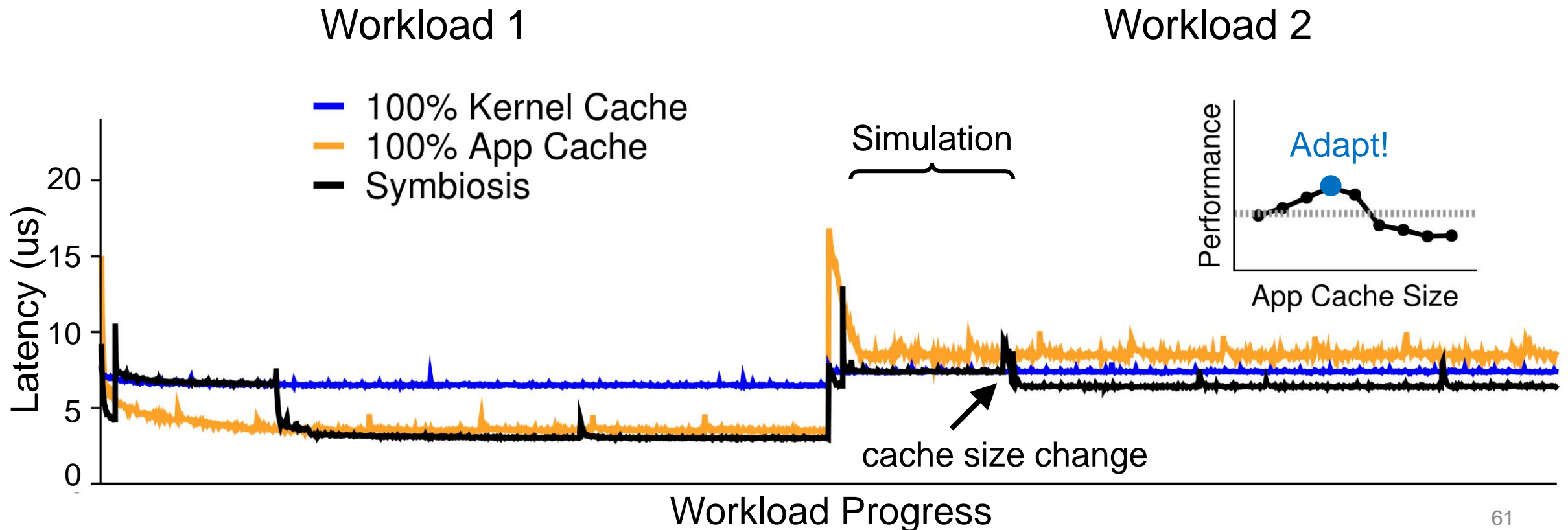


Dynamic Workloads - Example



Workload 2

Symbiosis maintains the best performance after simulation



Conclusion



Symbiosis: dynamic cache size adjustment via online simulation

Integrated into production systems within 1000 LoC

LevelDB, RocksDB, WiredTiger

Adapts to various workloads and reacts to workload changes

Performs well on 190+ static workloads and 38 dynamic workloads

Delivers excellent performance with negligible overhead

1.5x gain on average for read-heavy workloads

0.1% space overhead and 1% time overhead

Conclusion



Simulation-based online performance tuning

A tuning approach that does not rely on machine learning

Deep understanding and careful optimizations are necessary

Potential parameter tuning beyond cache sizes

Thank You for Listening!



See the paper for:

Offline simulation study

Detailed implementation and evaluation

Symbiosis source code

<https://github.com/daiyifandanny/Symbiosis>