HardHarvest: Hardware-Supported Core Harvesting for Microservices

ISCA '25

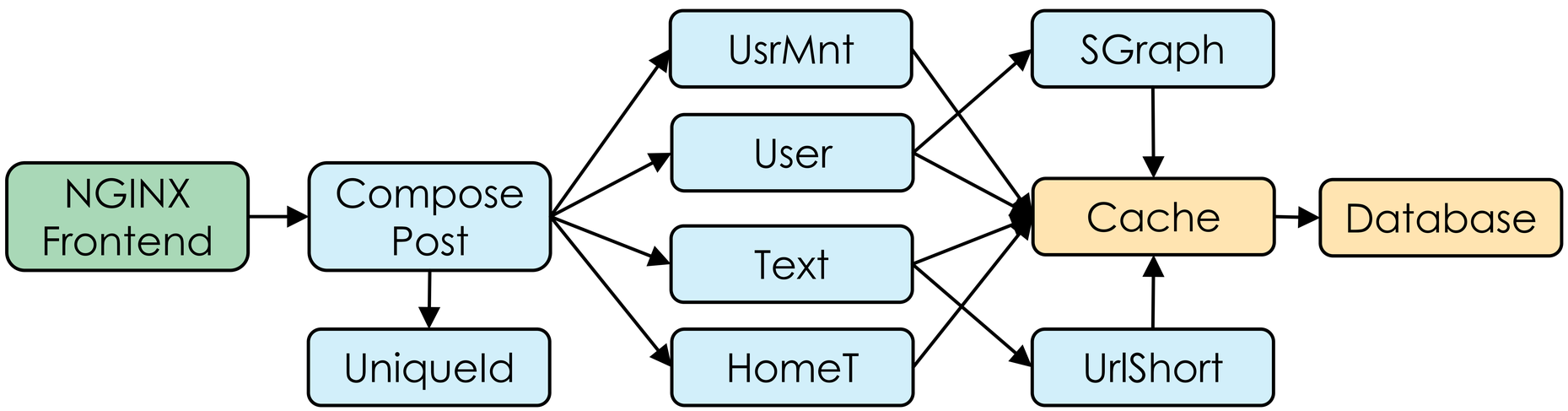
Jovan Stojkovic, Chunao Liu*, Muhammad Shahbaz*, Josep Torrellas

University of Illinois at Urbana-Champaign, *Purdue University

J. Stojkovic to start at UT Austin

Microservices

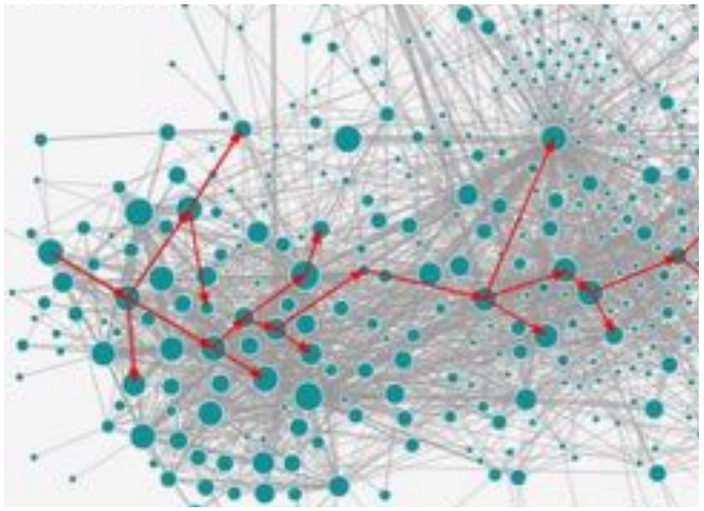
- Large monolithic applications decomposed into many small interdependent services
  - Each service implements separate functionality

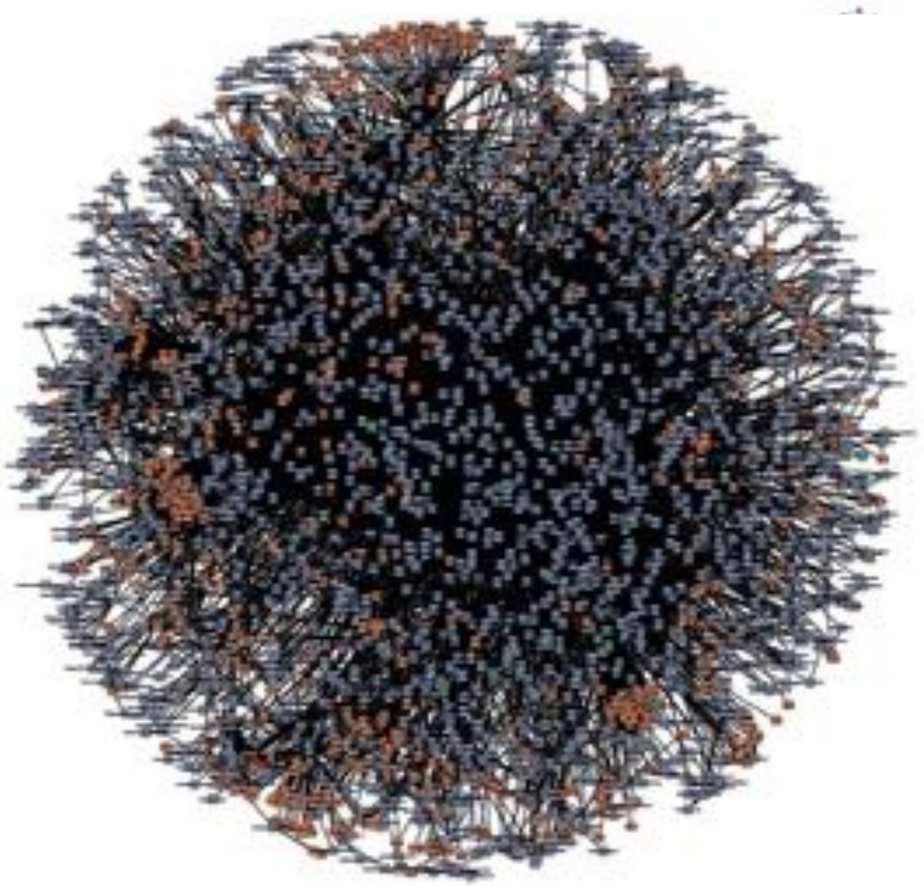


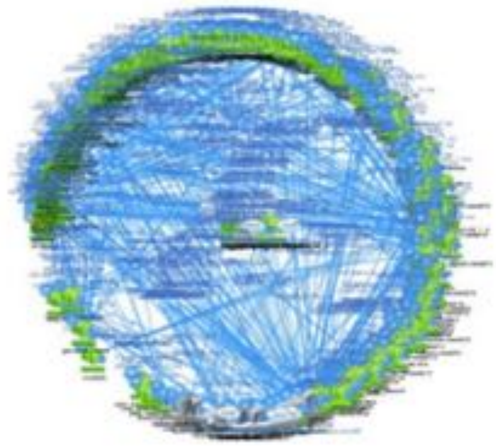
Benefits of Microservices

- Scalability ☑
- Design simplicity ☑
- HW management ☑

Microservices are Widely Used







ıre

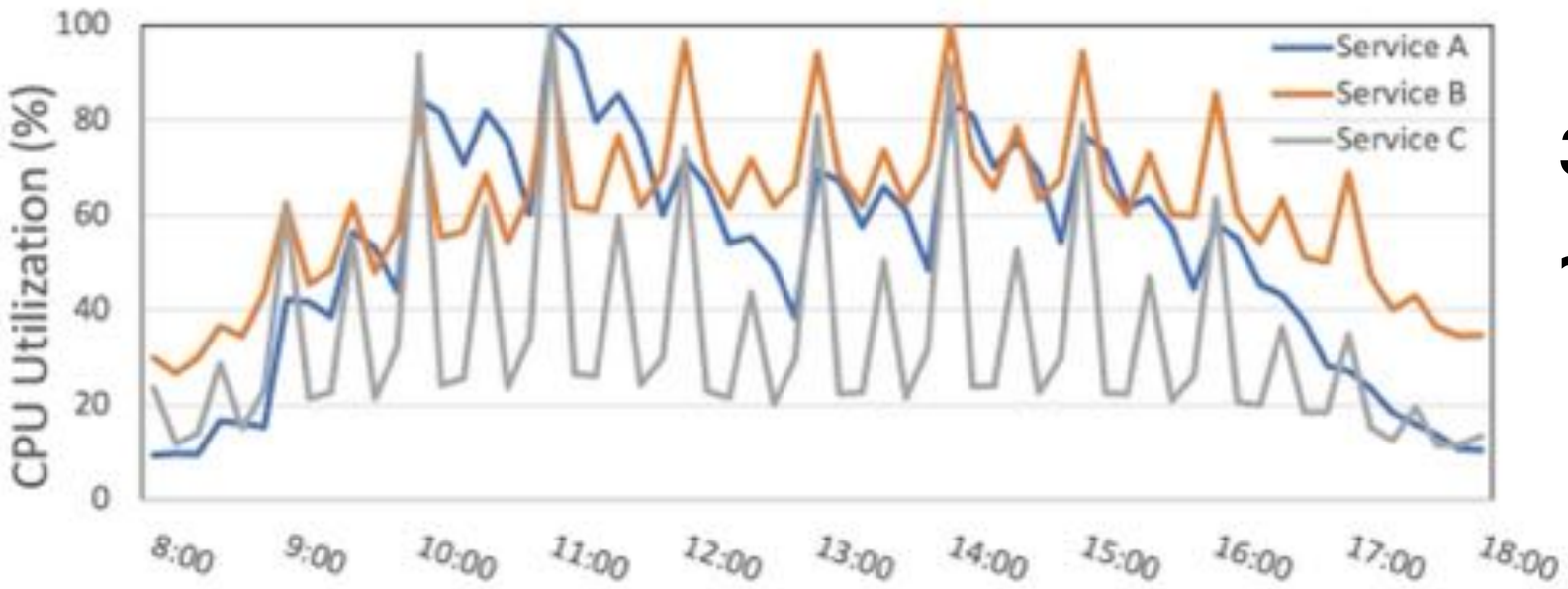
Actual Architecture

: simplified and actual scheme (source)

Structure of microservices at Amazon. Looks almost like a Death Star but is way more powerful.

Low Core Utilization with Microservices

- To accommodate the peak load, microservices typically overprovisioned

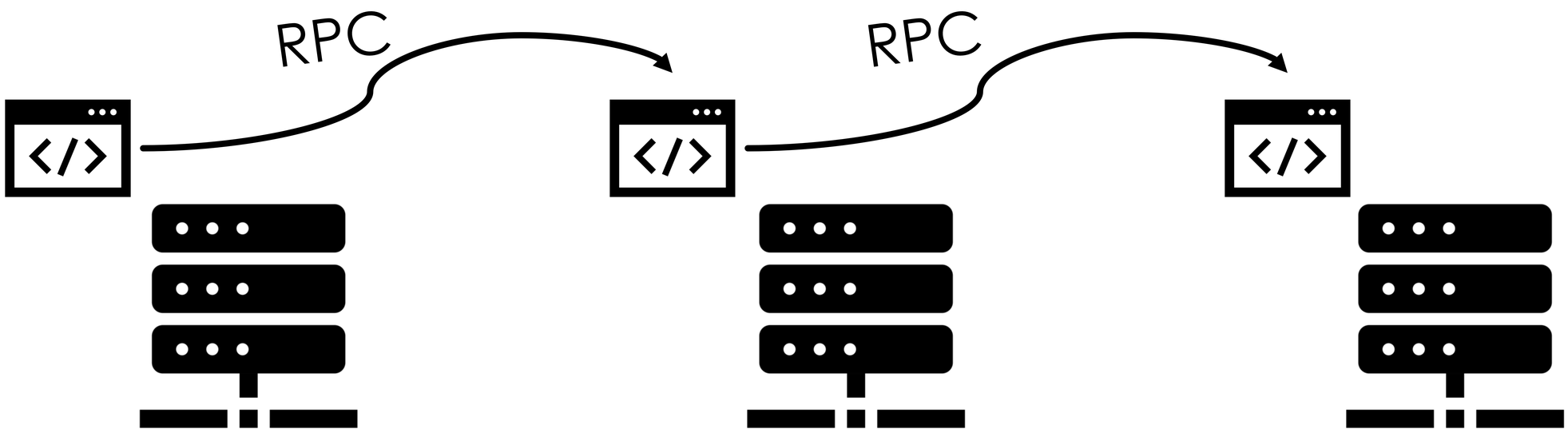


3 large Microsoft services

~1M virtual cores

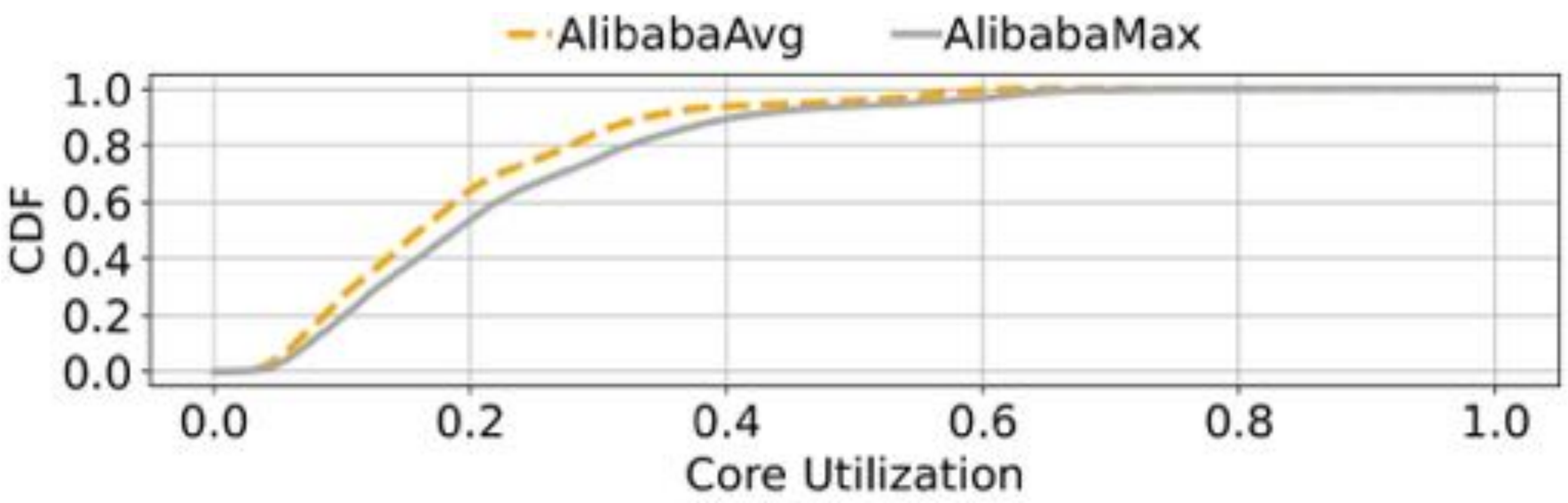
Low Core Utilization with Microservices

- To accommodate the peak load, microservices typically overprovisioned
- Requests often stalled on synchronous RPCs to read/write to/from remote storage, or to invoke other microservices



Low Core Utilization with Microservices

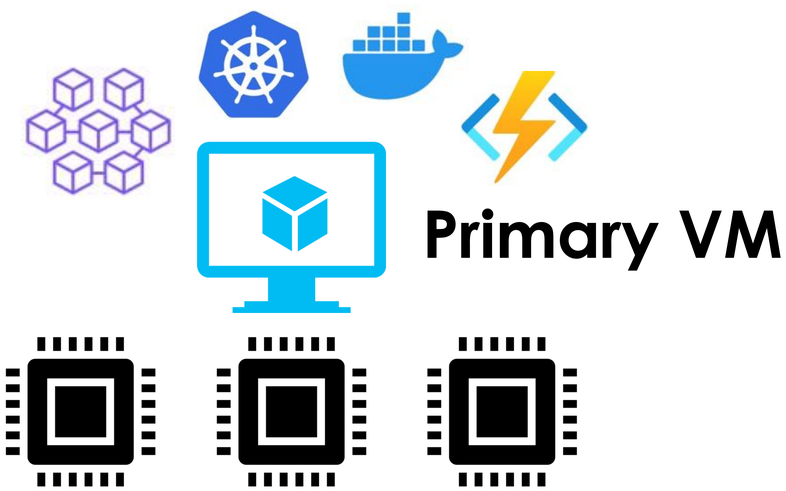
- To accommodate the peak load, microservices typically overprovisioned
- Requests often stalled on synchronous RPCs to read/write to/from remote storage, or to invoke other microservices
- As a result → **cores are heavily underutilized**



90% of services have maximum core utilization less than 40%!

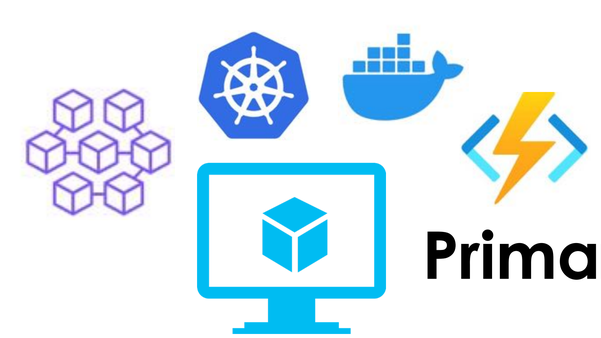
Core Harvesting to the Rescue

- Collocate latency-critical microservices with compute-heavy batch apps
- **Primary VMs**
  - Run latency-critical workloads
  - Users request a number of cores



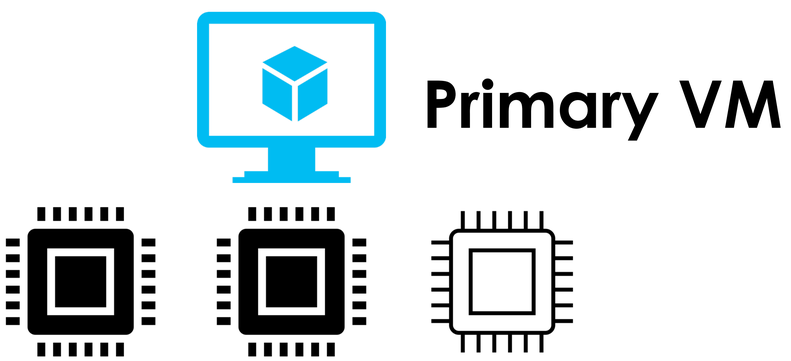
Core Harvesting to the Rescue

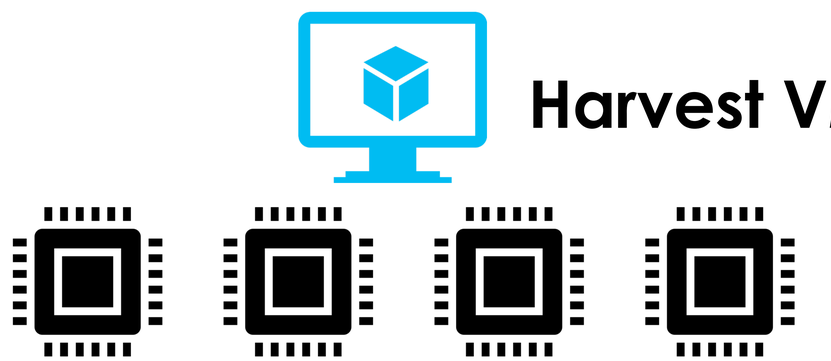
- Collocate latency-critical microservices with compute-heavy batch apps
- **Primary VMs**
  - Run latency-critical workloads
  - Users request a number of cores
- **Harvest VMs**
  - Run batch workloads in the background
  - Steal idle Primary VM's cores & return them



Core Harvesting to the Rescue

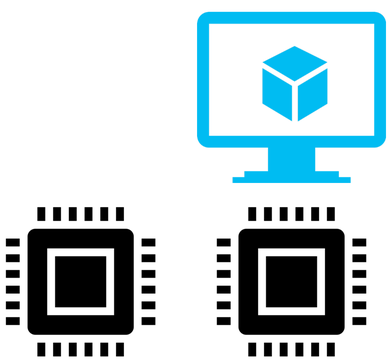
- Collocate latency-critical microservices with compute-heavy batch apps
- **Primary VMs**
  - Run latency-critical workloads
  - Users request a number of cores
- **Harvest VMs**
  - Run batch workloads in the background
  - Steal idle Primary VM's cores & return them

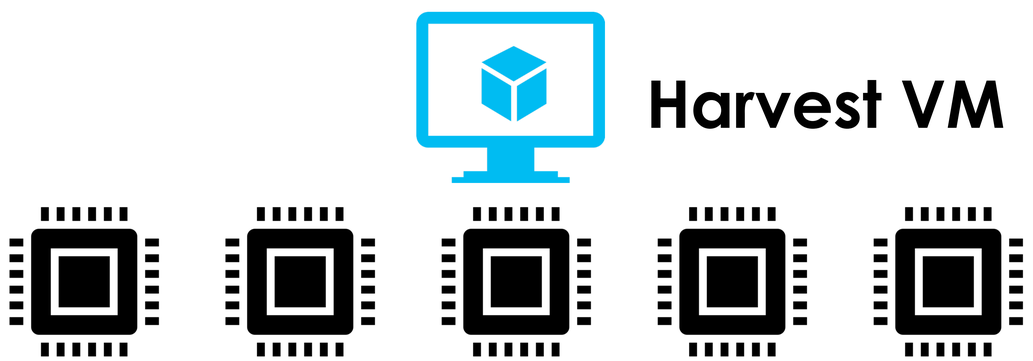




Core Harvesting to the Rescue

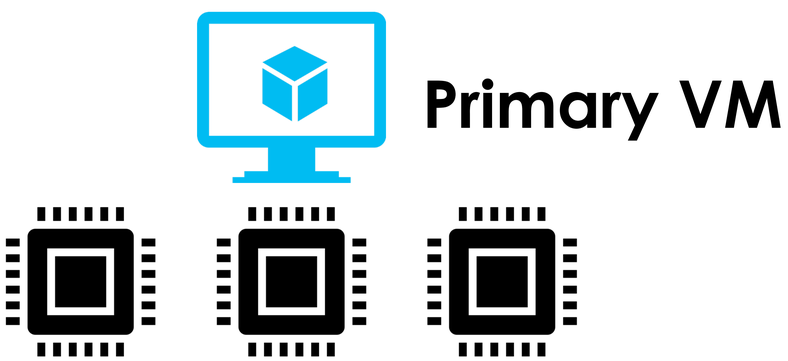
- Collocate latency-critical microservices with compute-heavy batch apps
- **Primary VMs**
  - Run latency-critical workloads
  - Users request a number of cores
- **Harvest VMs**
  - Run batch workloads in the background
  - Steal idle Primary VM's cores & return them

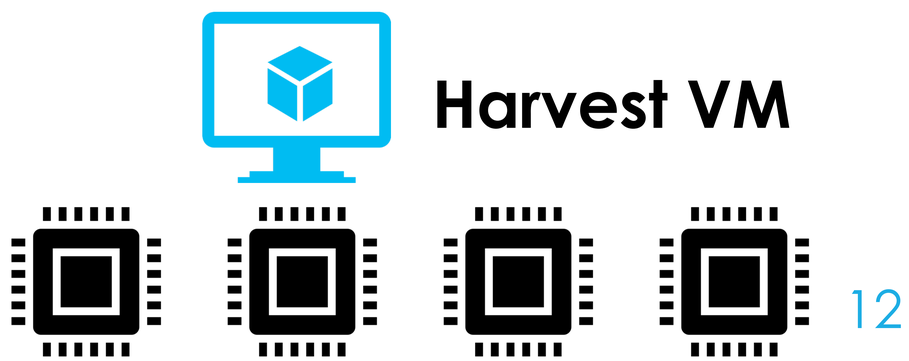




Core Harvesting to the Rescue

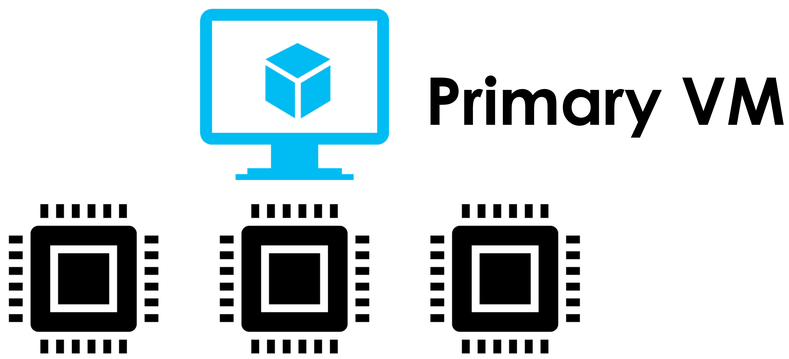
- Collocate latency-critical microservices with compute-heavy batch apps
- **Primary VMs**
  - Run latency-critical workloads
  - Users request a number of cores
- **Harvest VMs**
  - Run batch workloads in the background
  - Steal idle Primary VM's cores & return them





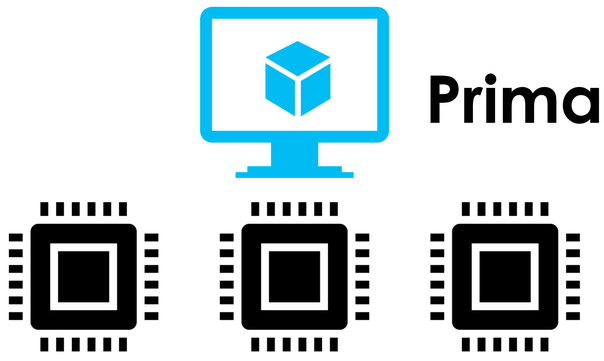
Core Harvesting to the Rescue

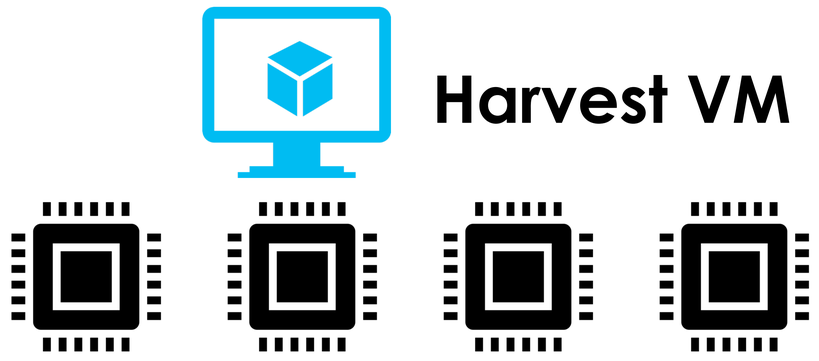
- Improved resource utilization
- Higher throughput for Harvest VMs



Core Harvesting to the Rescue

- Improved resource utilization
- Higher throughput for Harvest VMs
- Software core harvesting → increase tail latency for Primary VMs!



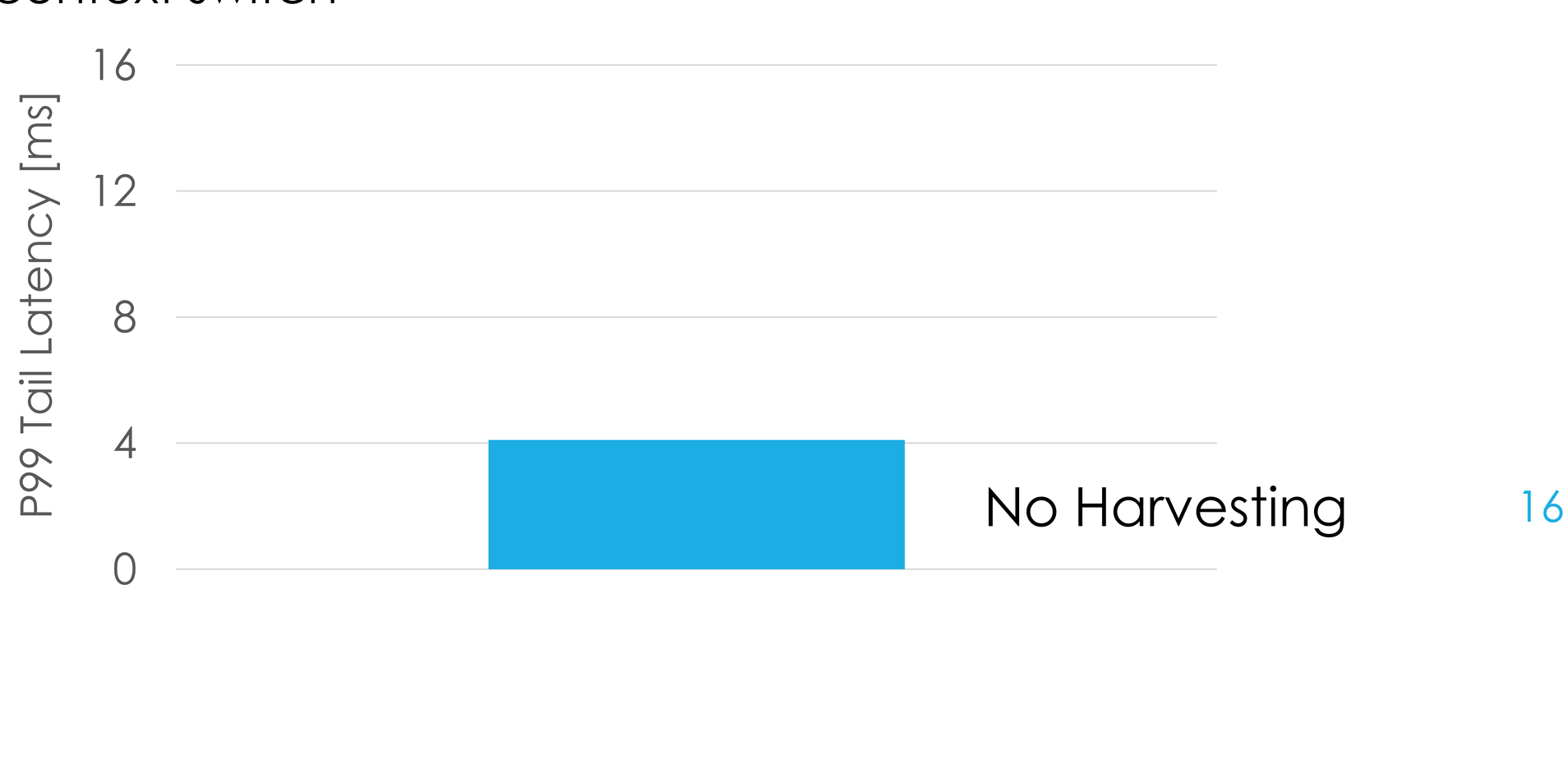


Problem #1: Expensive Core Re-Assignment

- Moving core from one VM to another is expensive
  - Need for two hypervisor calls (detach and attach)
  - Cross-VM context switch

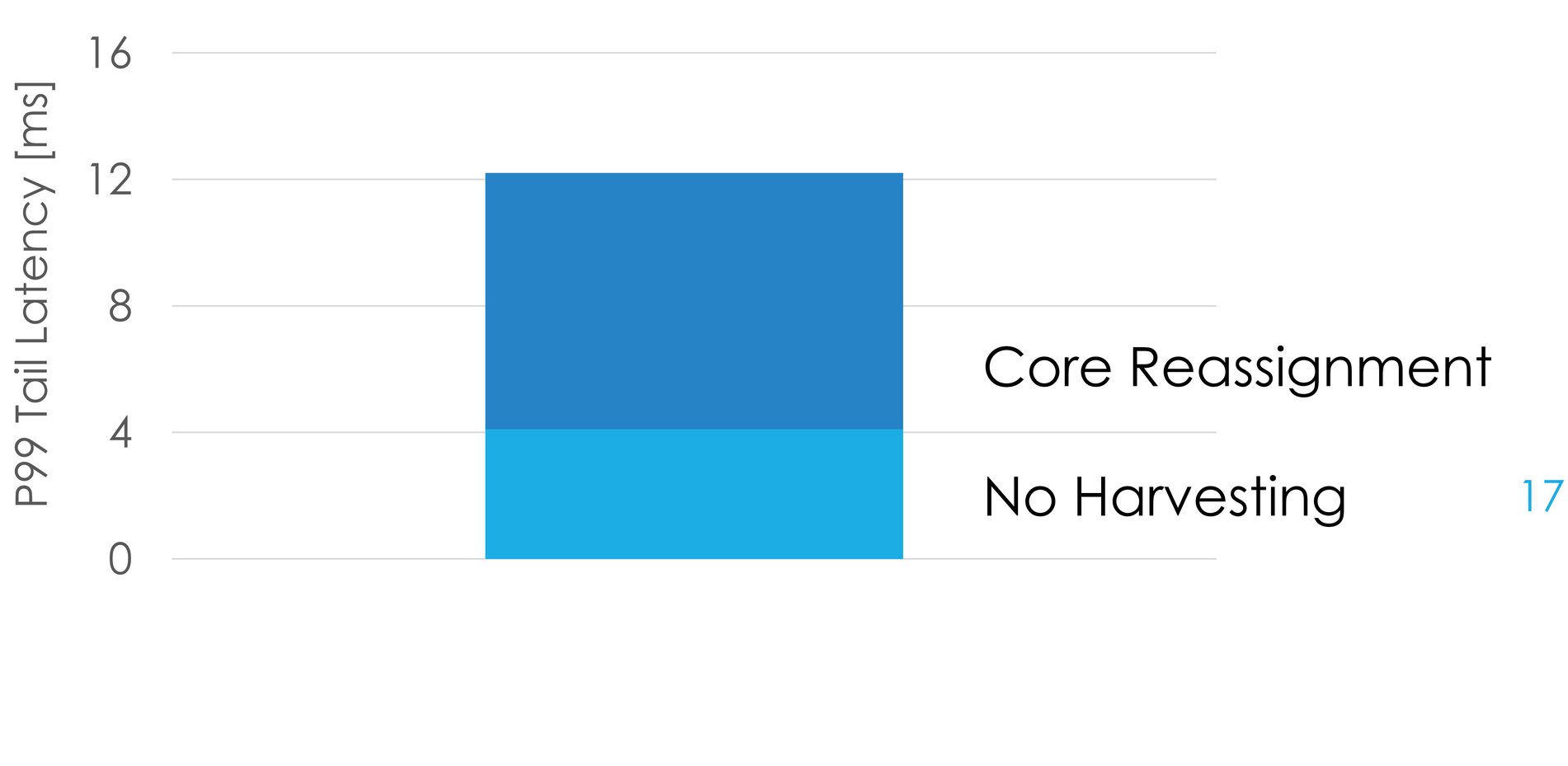
Problem #1: Expensive Core Re-Assignment

- Moving core from one VM to another is expensive
  - Need for two hypervisor calls (detach and attach)
  - Cross-VM context switch



Problem #1: Expensive Core Re-Assignment

- Moving core from one VM to another is expensive
  - Need for two hypervisor calls (detach and attach)
  - Cross-VM context switch

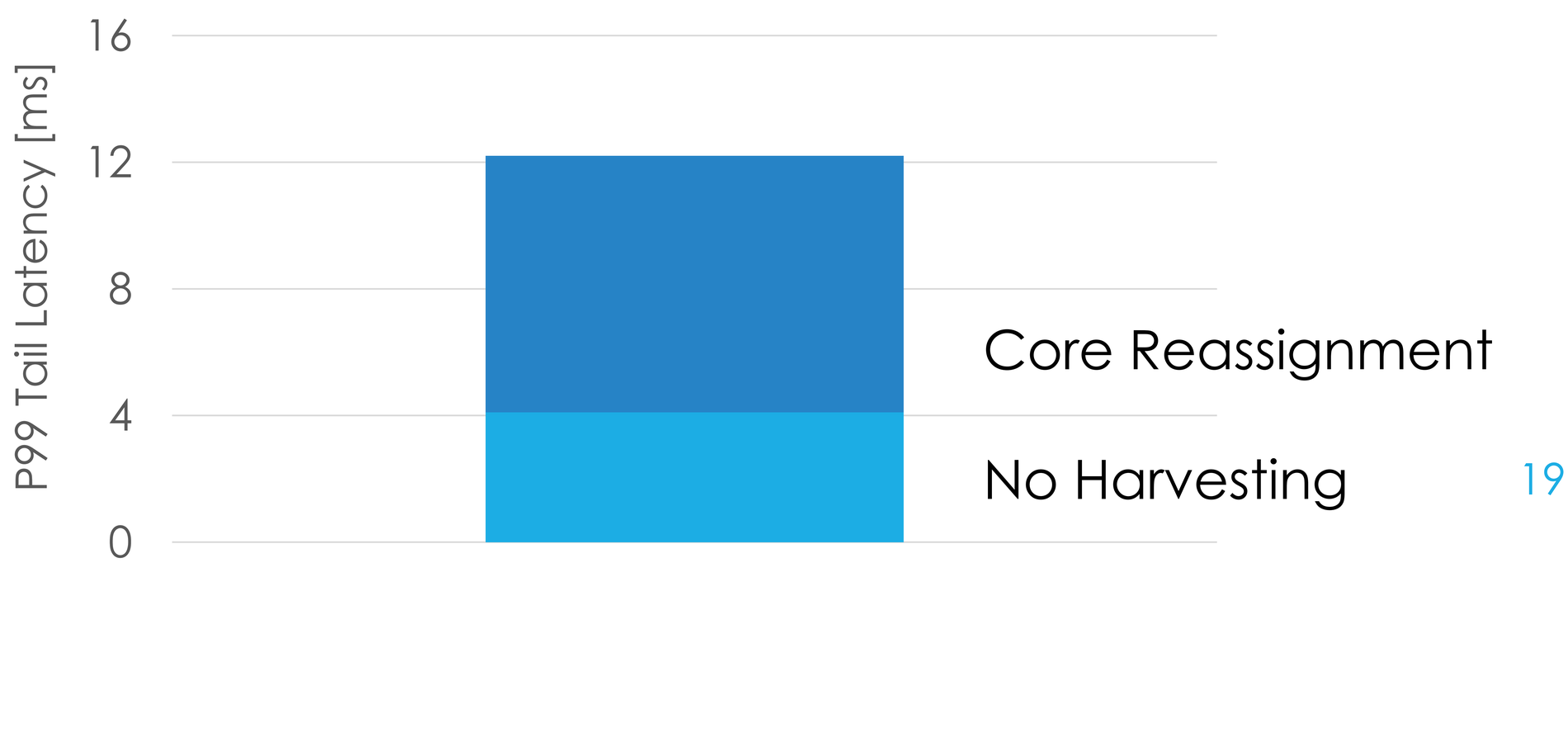


Problem #2: Expensive Micro-arch Flushing

- For security → different VMs should not observe any state left in caches/TLBs
- On a cross-VM context switch, need to flush and invalidate caches/TLBs

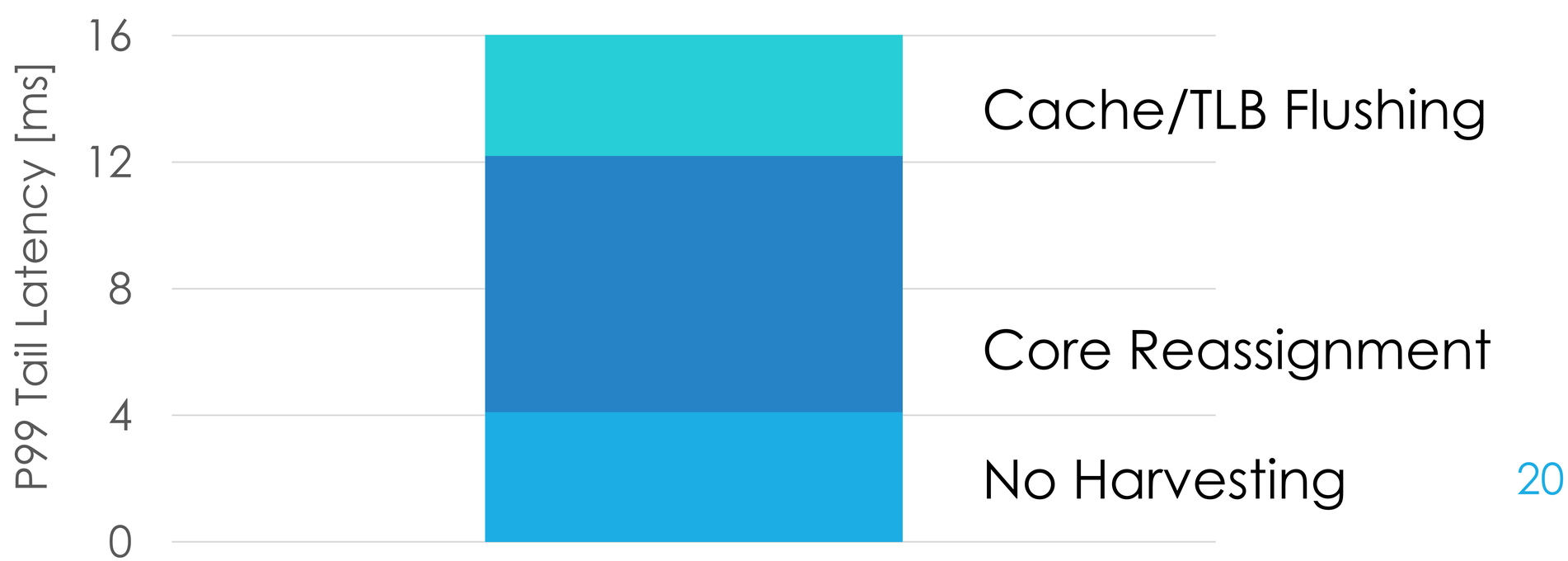
Problem #2: Expensive Micro-arch Flushing

- For security → different VMs should not observe any state left in caches/TLBs
- On a cross-VM context switch, need to flush and invalidate caches/TLBs



Problem #2: Expensive Micro-arch Flushing

- For security → different VMs should not observe any state left in caches/TLBs
- On a cross-VM context switch, need to flush and invalidate caches/TLBs



Goals

- High core utilization
- High Harvest VM throughput
- Low Primary VM tail latency

Contributions

- Characterize the opportunities of supporting hardware-based core harvesting in microservice environments
- Propose **HardHarvest**
  - The first architecture for hardware core harvesting
- Compared to state-of-the-art software core harvesting
  - **Higher core utilization:** 1.5x
  - **Higher Harvest VM throughput:** 1.8x
  - **Lower Primary VM tail latency:** 6.0x

Opportunity: Small Services' Working Set Sizes

- Microservices sensitive to cache misses, but tolerate Cache/TLB downsizing
  - Tail latency remains nearly unchanged even when LLC is reduced to its ¼

Opportunity: Small Services' Working Set Sizes

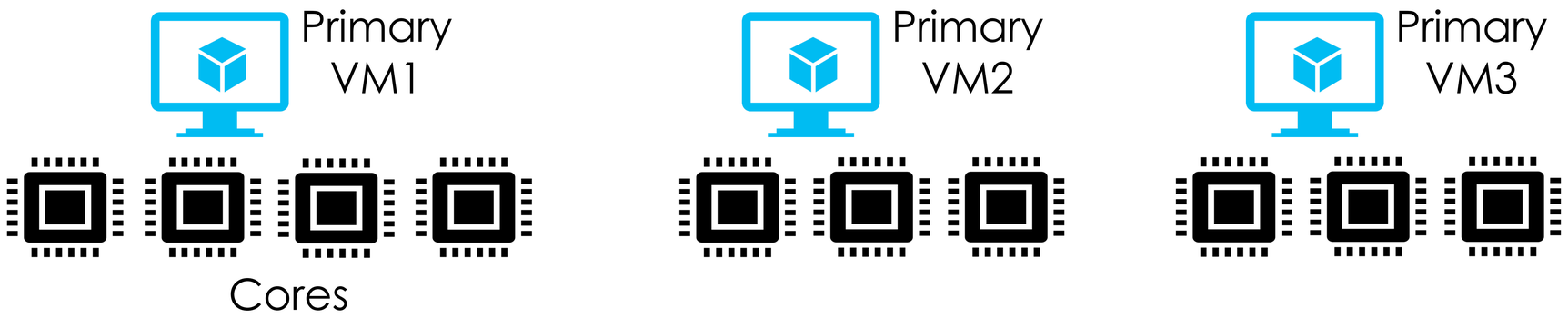
- Microservices sensitive to cache misses, but tolerate Cache/TLB downsizing
  - Tail latency remains nearly unchanged even when LLC is reduced to its ¼
- Enables selective flushing and partitioning
  - Reserve cache/TLB space for the Primary VM and preserve useful data

HardHarvest: Hardware-Supported Core Harvesting for Microservices

1. Software overheads of core reassigning high → propose a hardware solution
2. Cache/TLB flushing expensive but needed for security → partition caches/TLBs with smart replacement policy

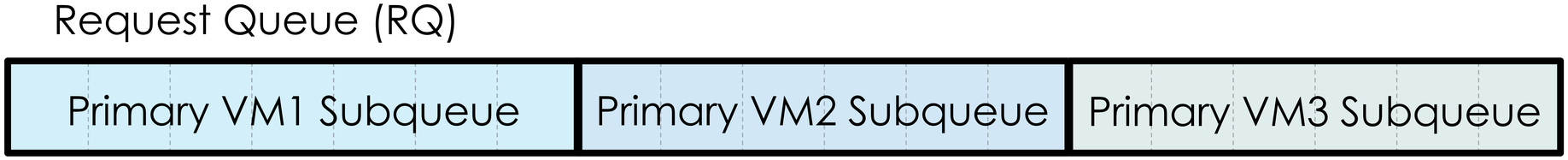
Hardware Schedulers

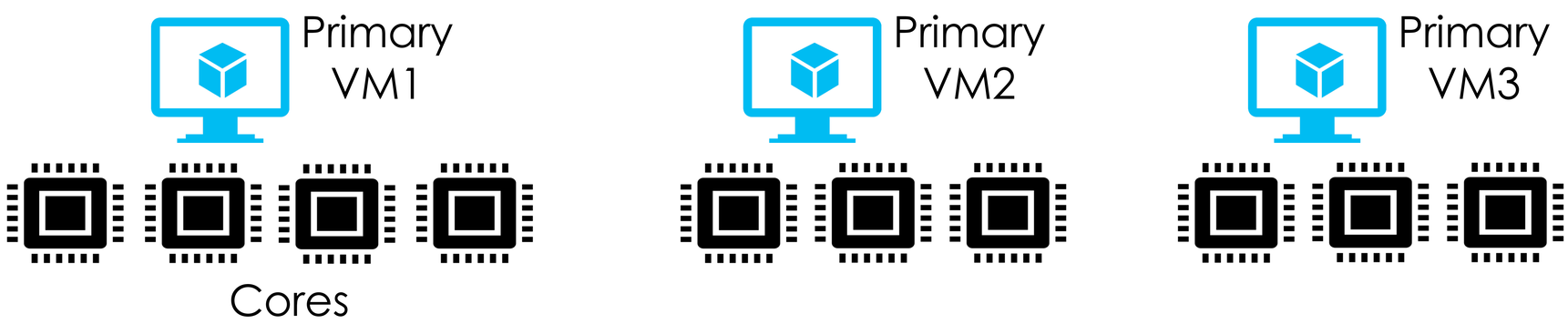
- Hardware schedulers to schedule VM's incoming requests
  - Avoid hypervisor calls and software locks



Hardware Schedulers

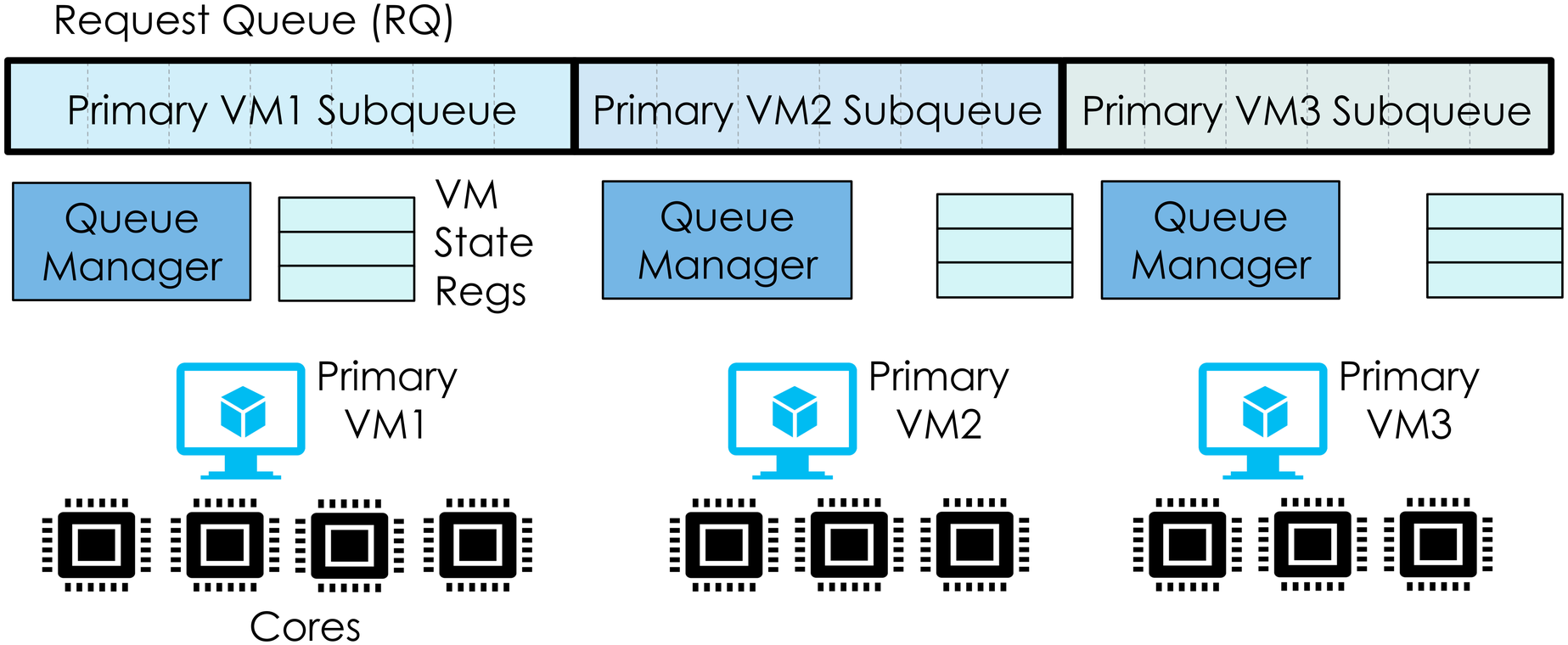
- Hardware schedulers to schedule VM's incoming requests
  - Avoid hypervisor calls and software locks





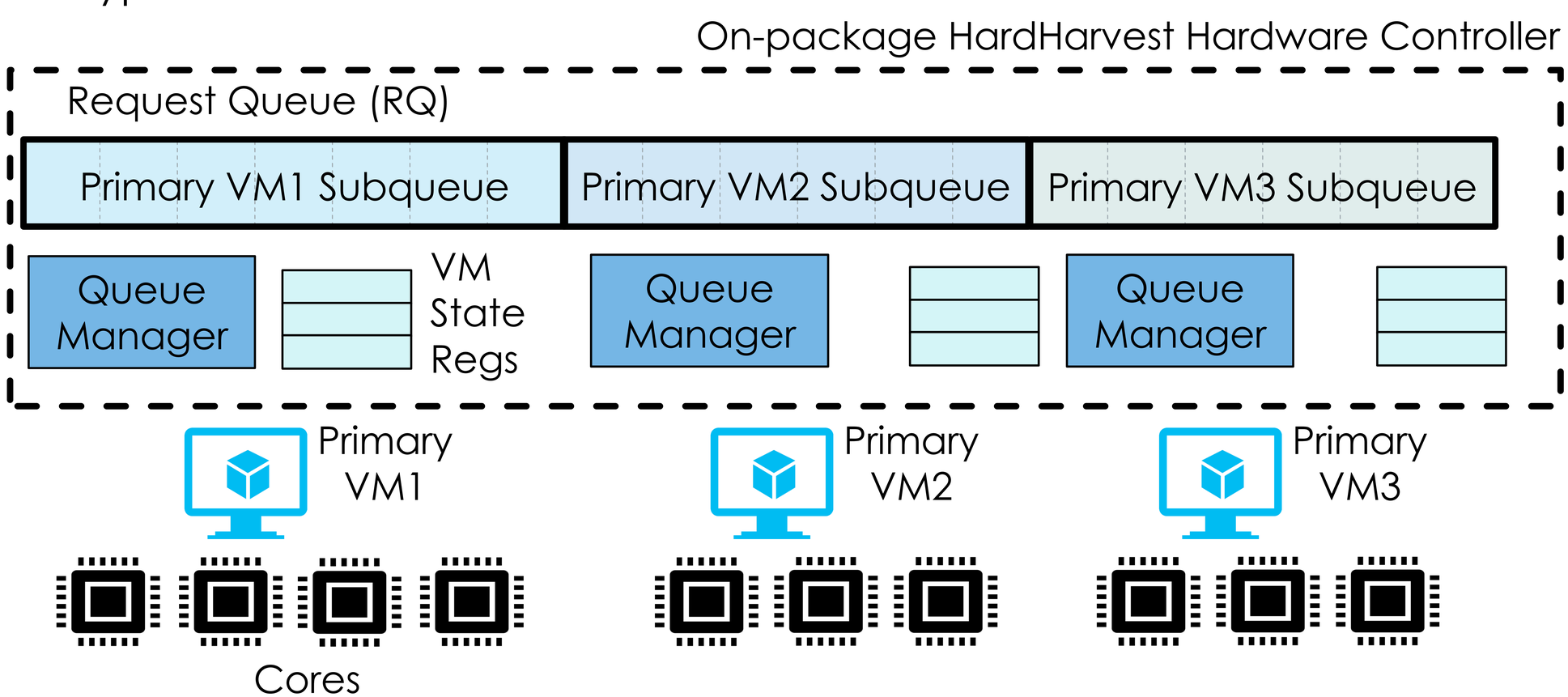
Hardware Schedulers

- Hardware schedulers to schedule VM's incoming requests
  - Avoid hypervisor calls and software locks



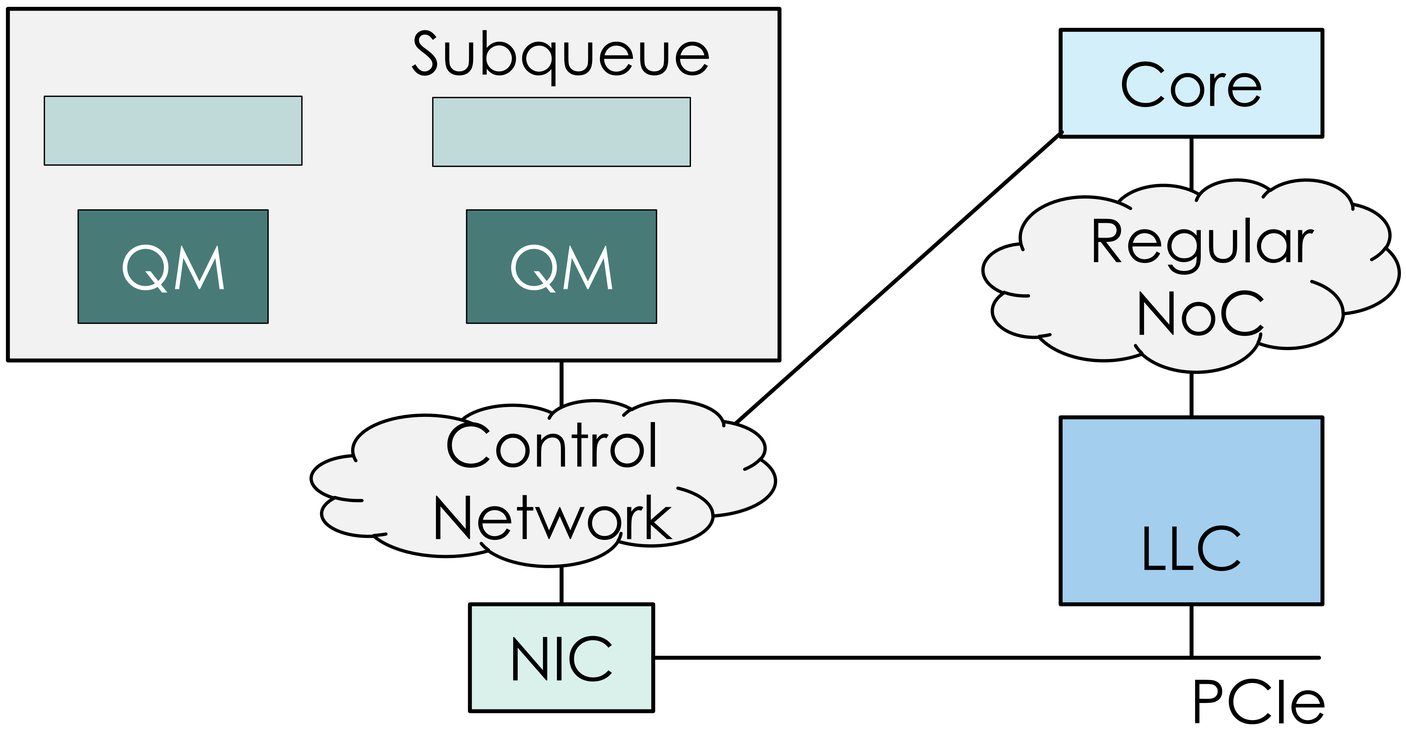
Hardware Schedulers

- Hardware schedulers to schedule VM's incoming requests
  - Avoid hypervisor calls and software locks



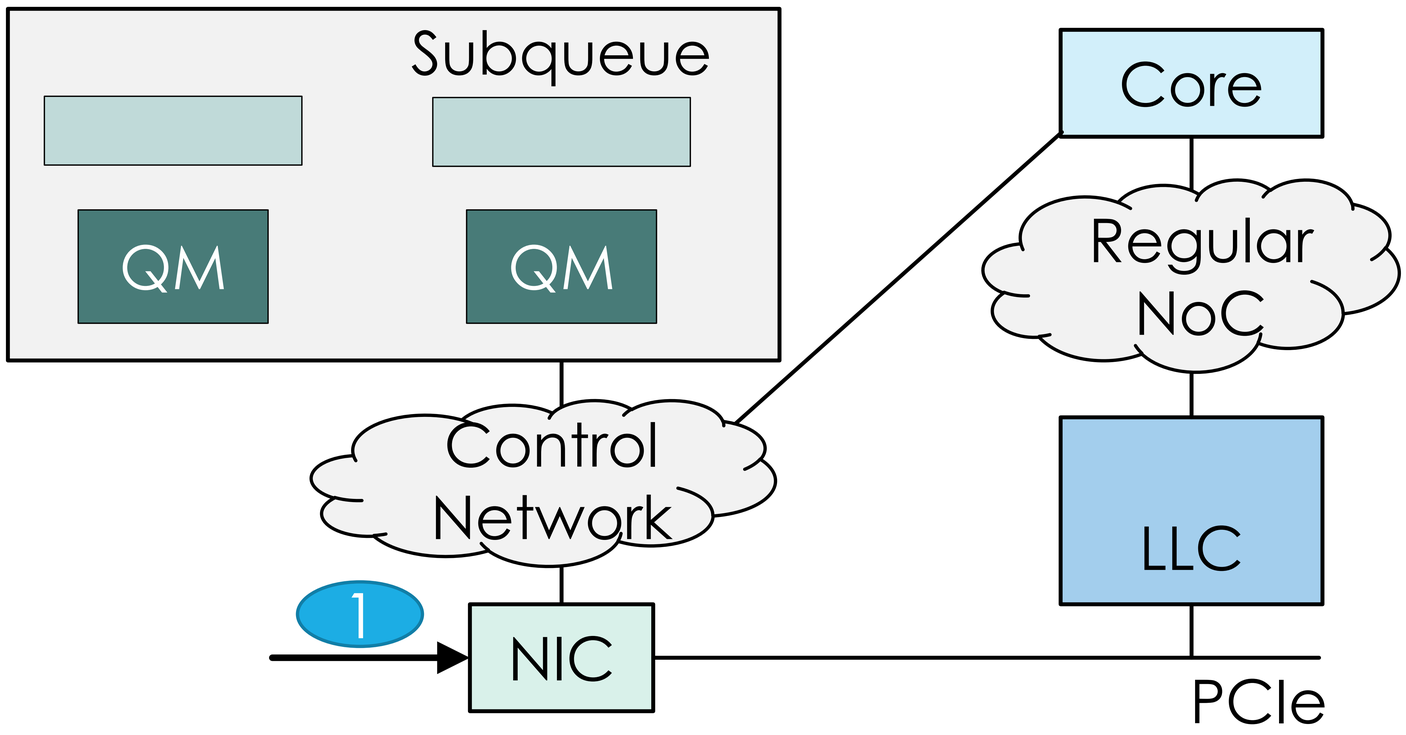
Handling New Incoming Requests

- Hardware schedulers to schedule VM's incoming requests
  - Avoid hypervisor calls and software locks



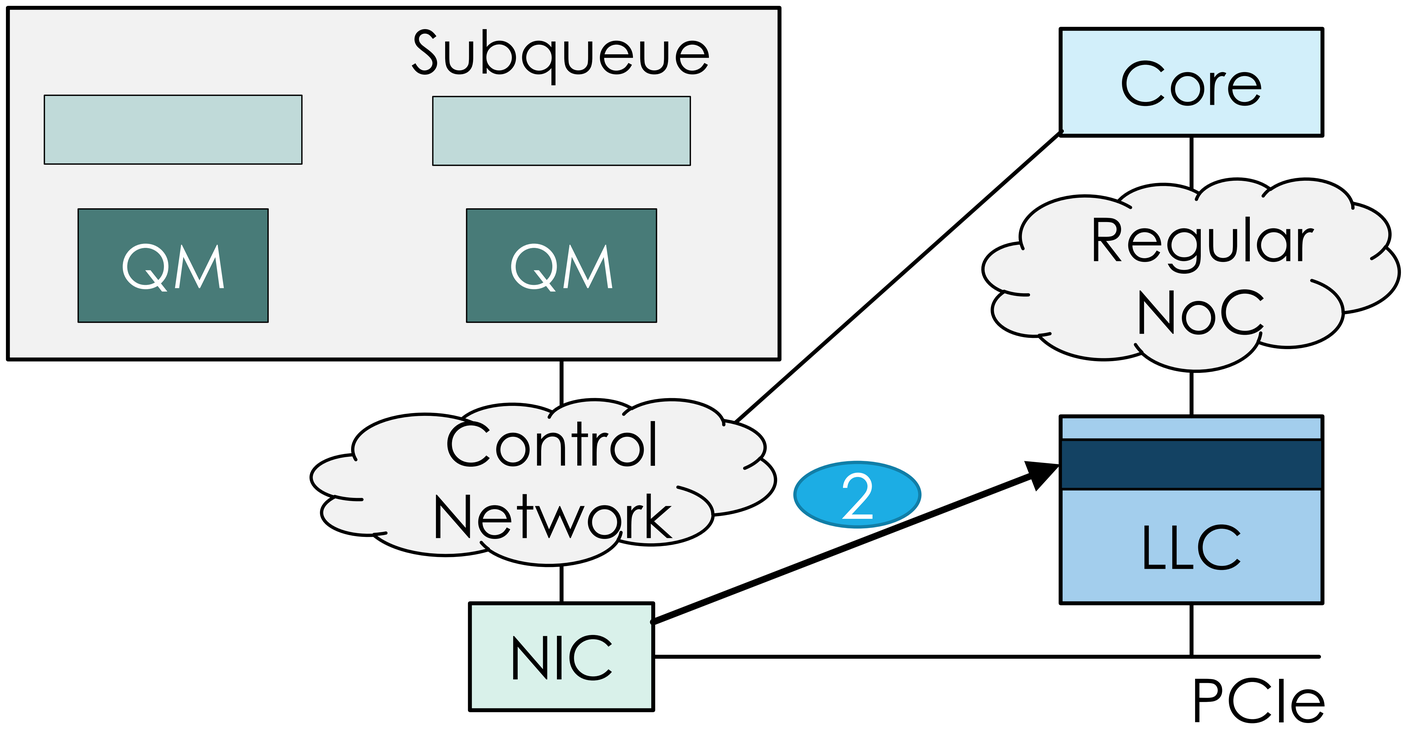
Handling New Incoming Requests

- Hardware schedulers to schedule VM's incoming requests
  - Avoid hypervisor calls and software locks



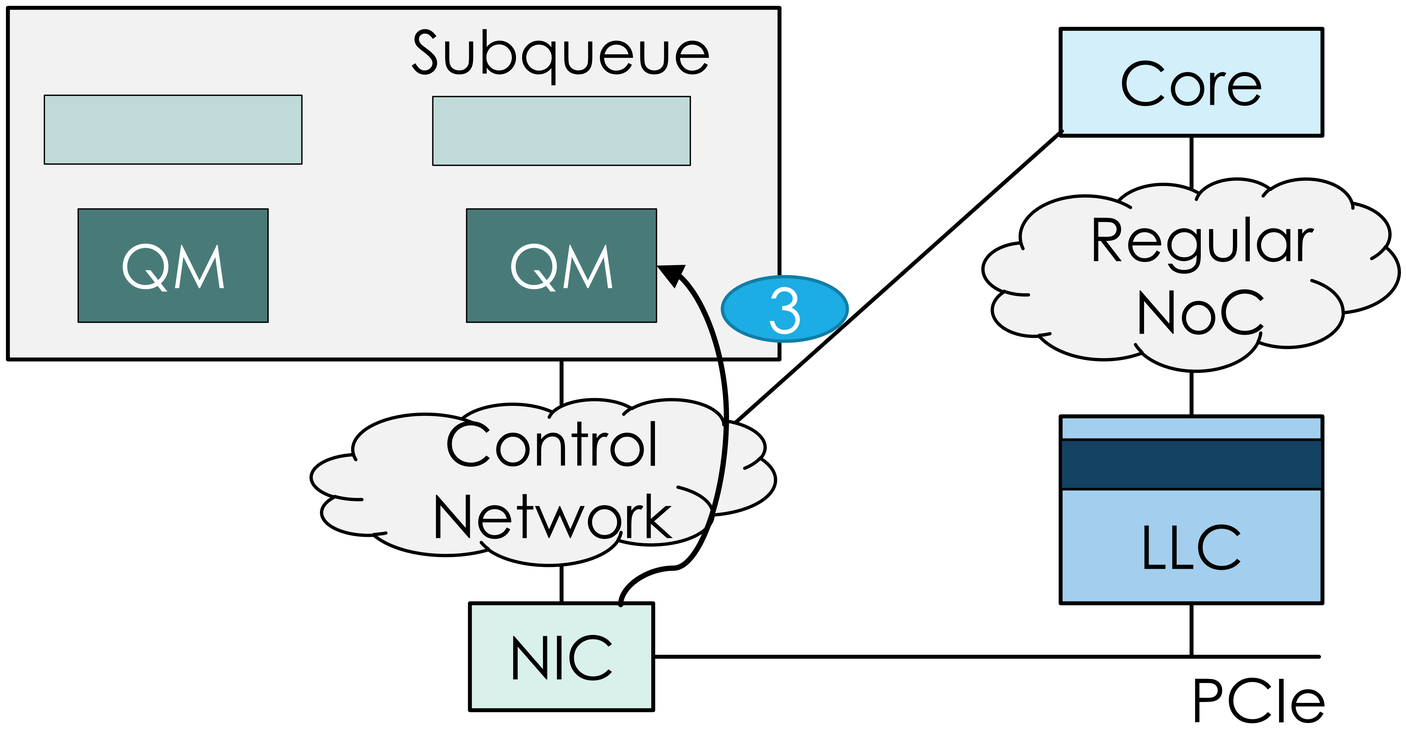
Handling New Incoming Requests

- Hardware schedulers to schedule VM's incoming requests
  - Avoid hypervisor calls and software locks



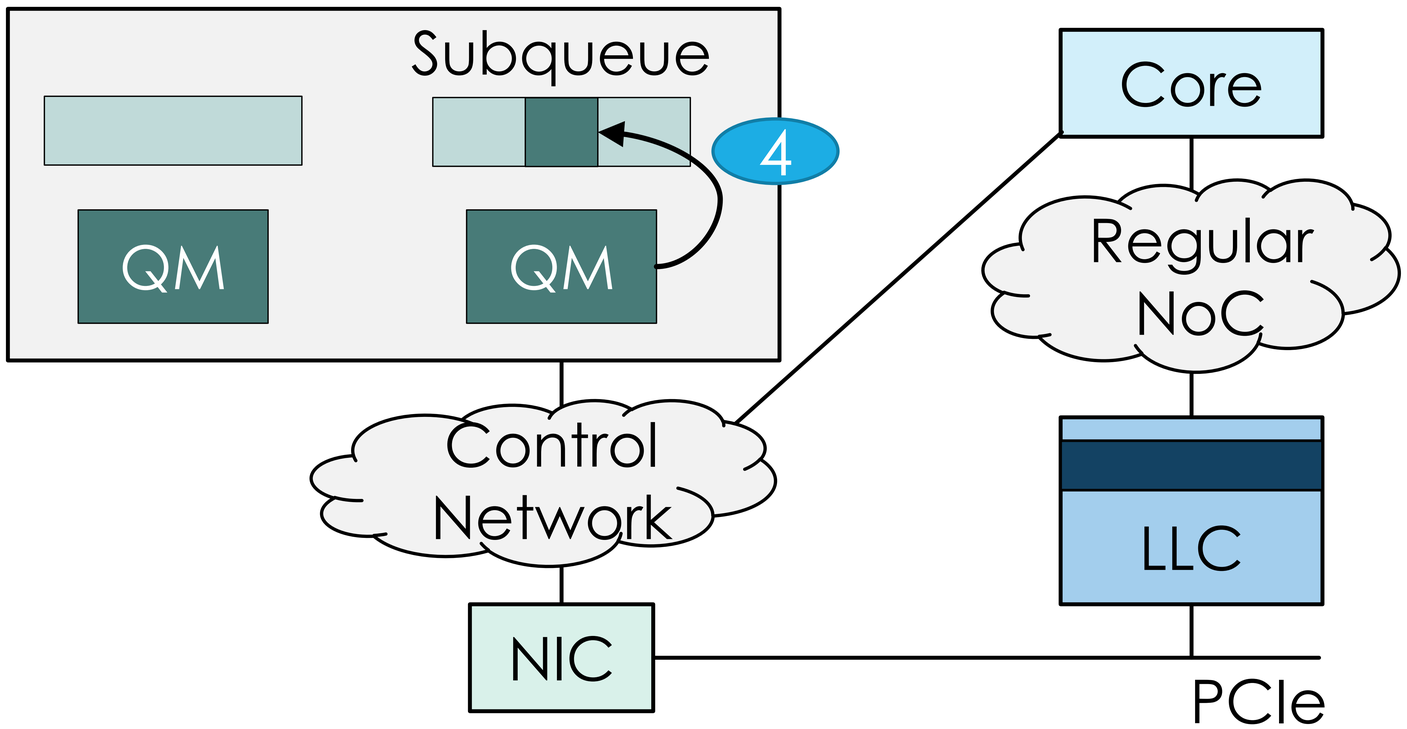
Handling New Incoming Requests

- Hardware schedulers to schedule VM's incoming requests
  - Avoid hypervisor calls and software locks



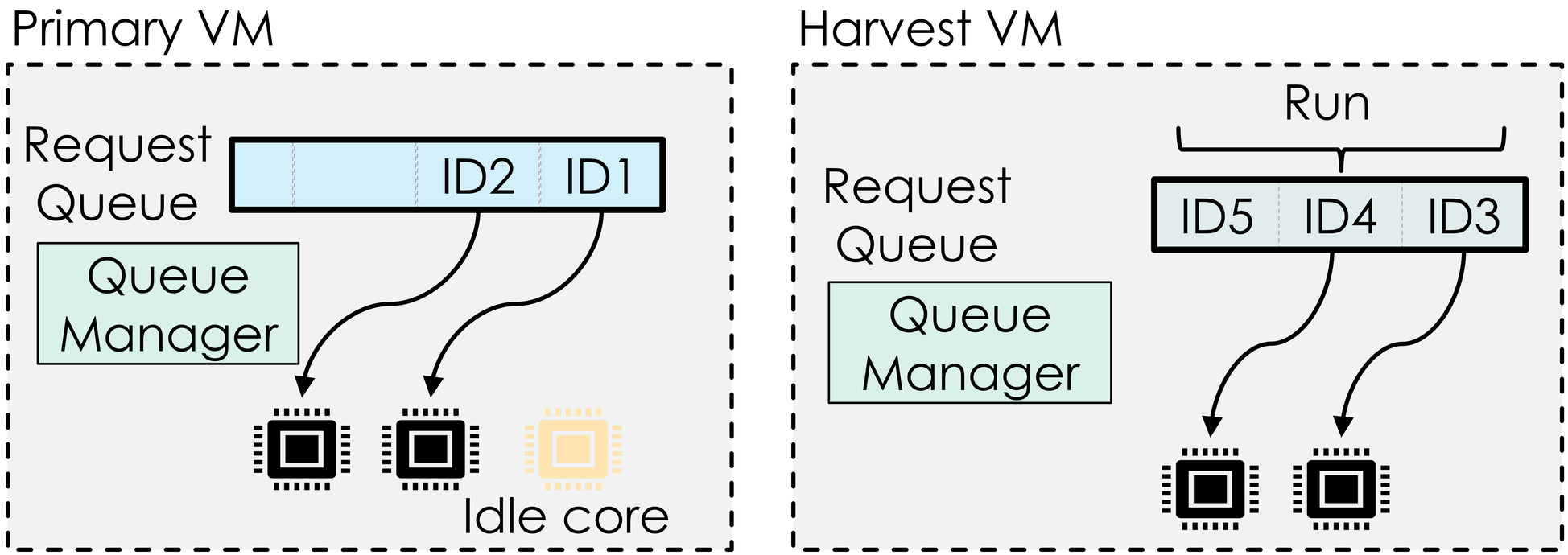
Handling New Incoming Requests

- Hardware schedulers to schedule VM's incoming requests
  - Avoid hypervisor calls and software locks



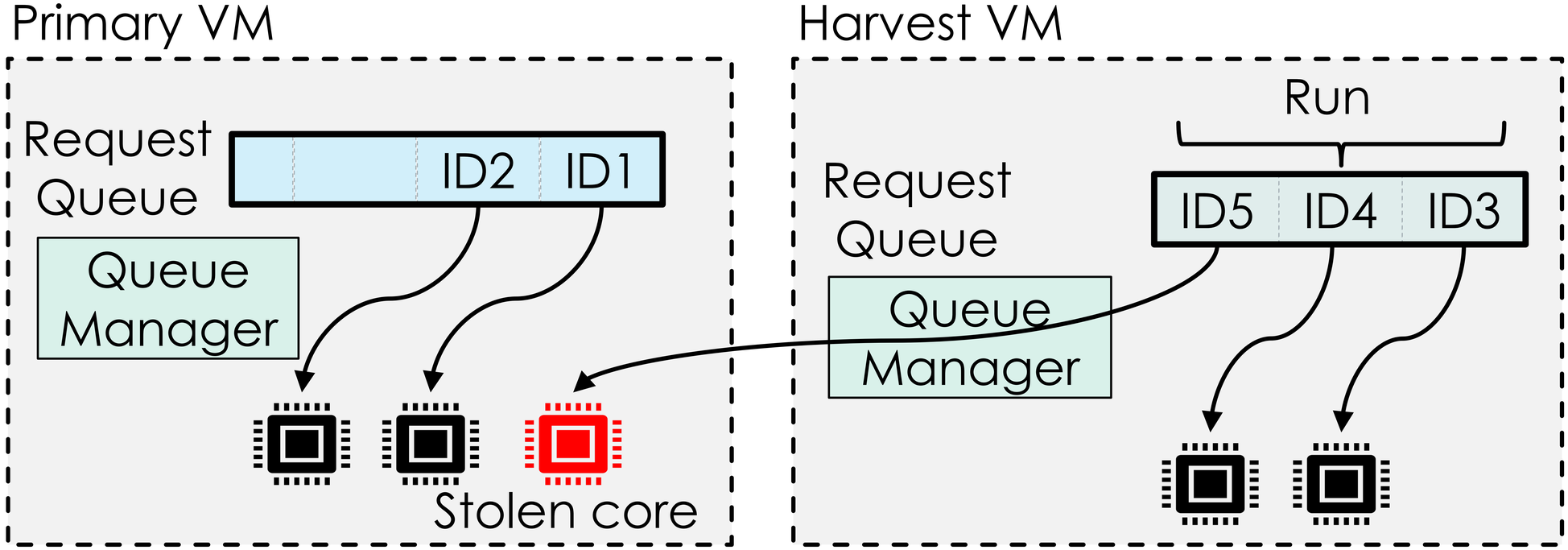
Minimizing Core Reassignment Overheads

- Hardware schedulers to schedule VM's incoming requests
  - Avoid hypervisor calls and software locks



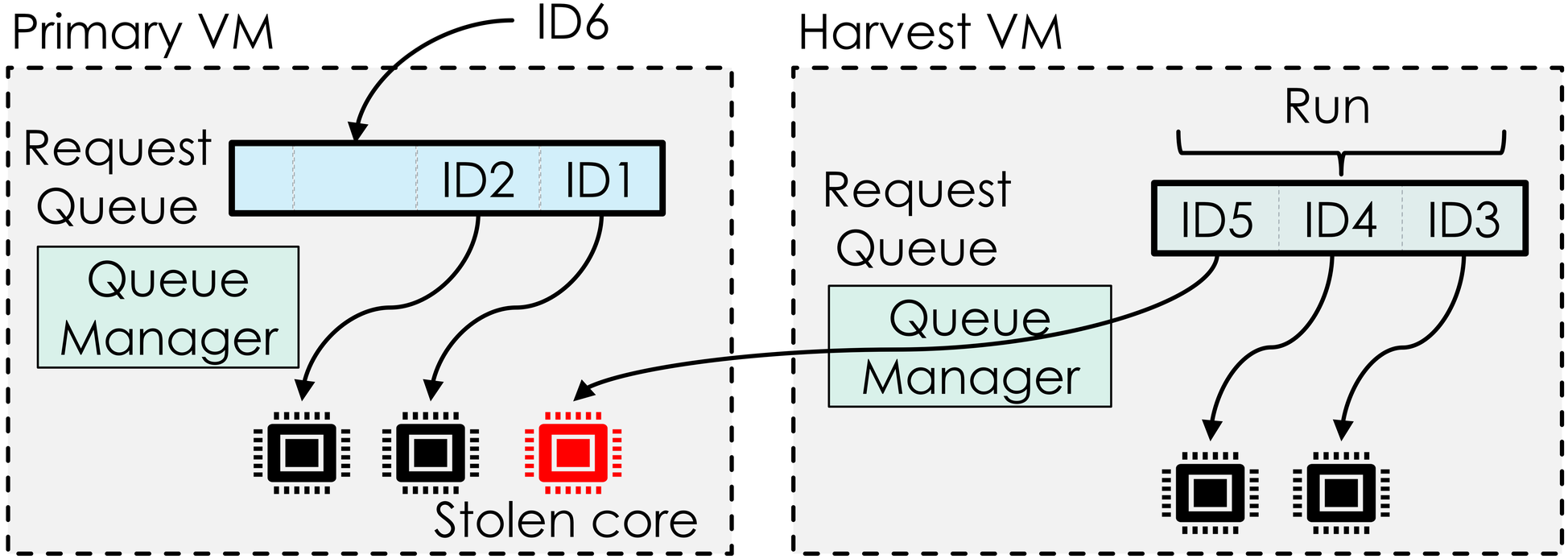
Minimizing Core Reassignment Overheads

- Hardware schedulers to schedule VM's incoming requests
  - Avoid hypervisor calls and software locks



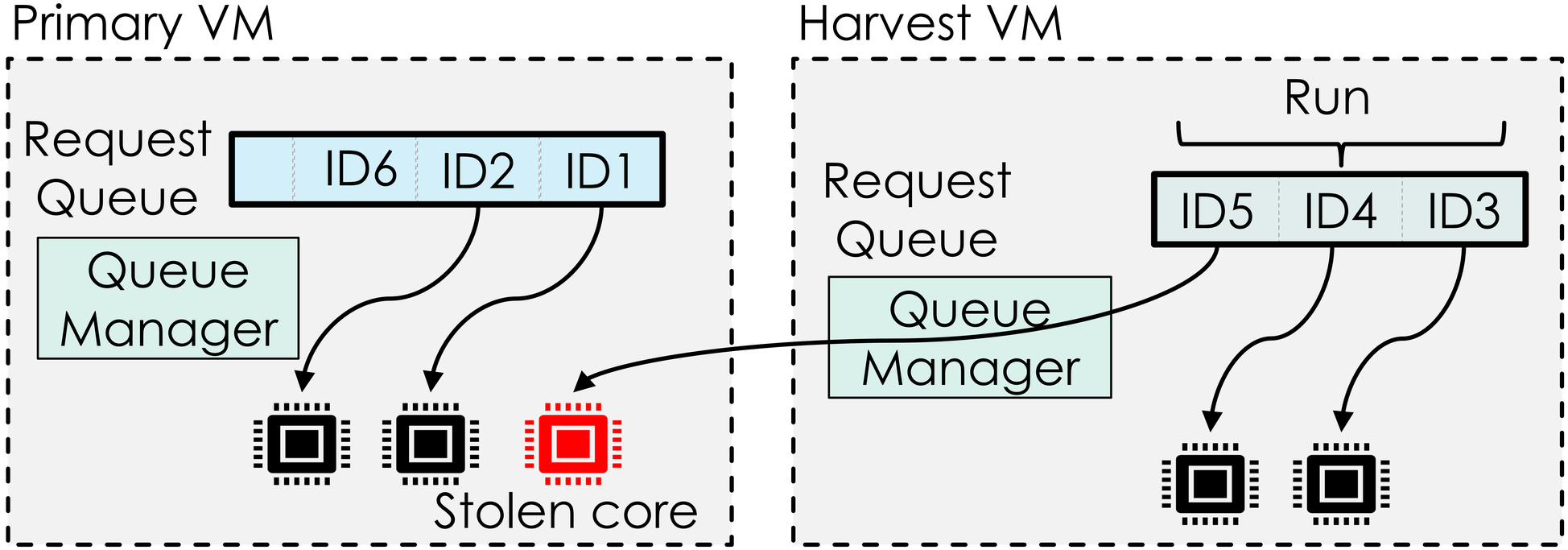
Minimizing Core Reassignment Overheads

- Hardware schedulers to schedule VM's incoming requests
  - Avoid hypervisor calls and software locks



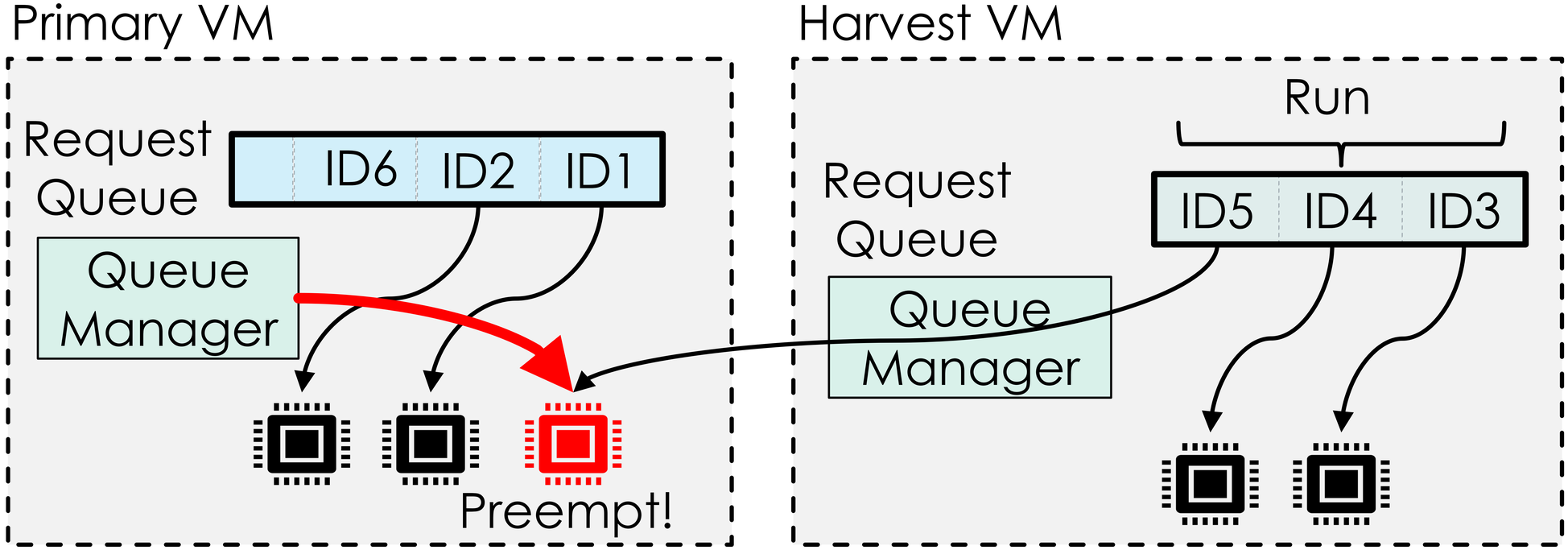
Minimizing Core Reassignment Overheads

- Hardware schedulers to schedule VM's incoming requests
  - Avoid hypervisor calls and software locks



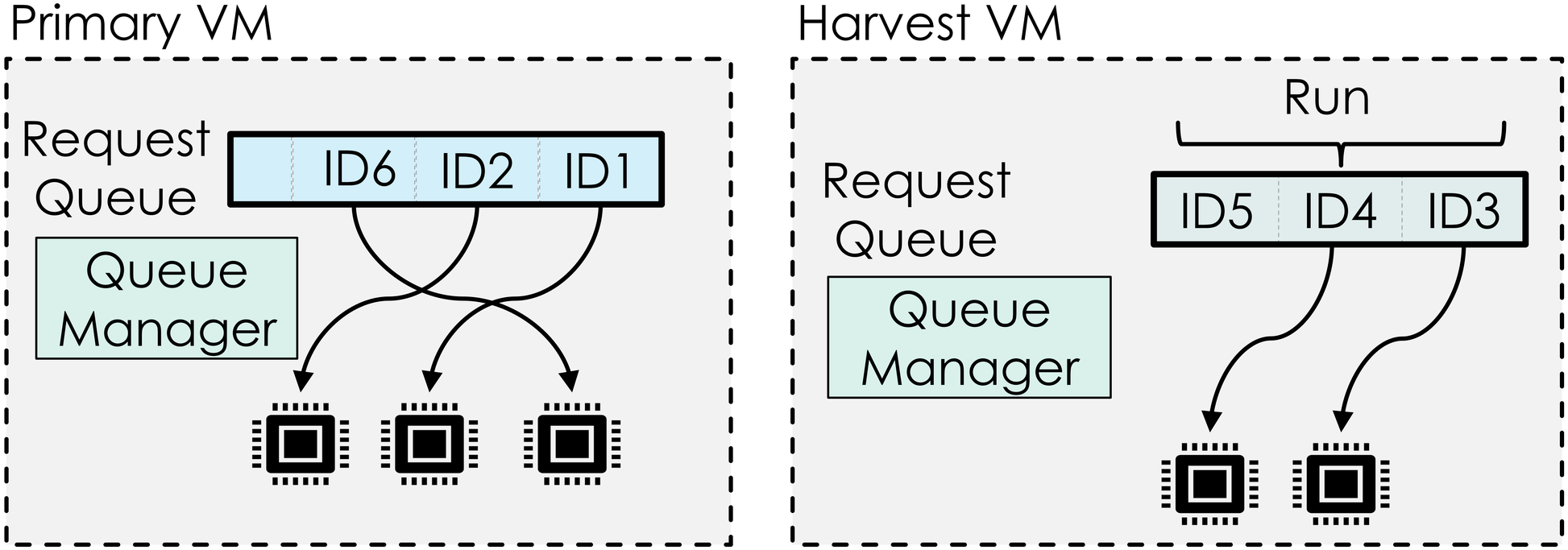
Minimizing Core Reassignment Overheads

- Hardware schedulers to schedule VM's incoming requests
  - Avoid hypervisor calls and software locks



Minimizing Core Reassignment Overheads

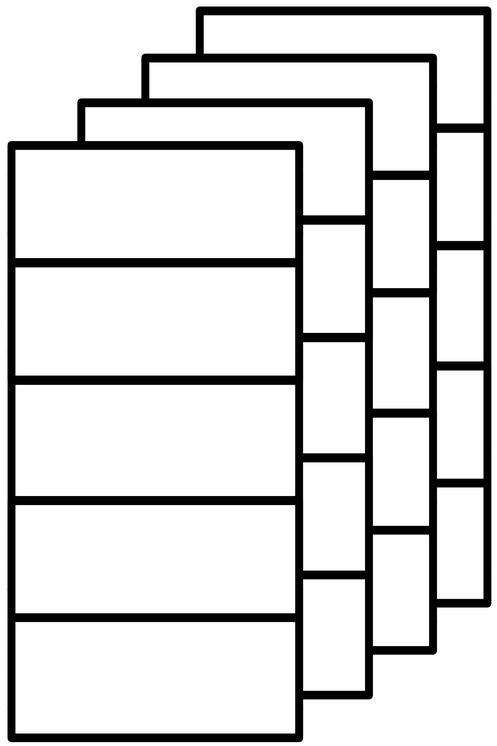
- Hardware schedulers to schedule VM's incoming requests
  - Avoid hypervisor calls and software locks



TLB/Cache Partitioning to Preserve State

- For security, stateful HW structures flushed+inv on cross-VM context switch
  - L1 (D+I) caches and TLBs, and L2 cache and TLB
- Microservices have relatively small instructions and data working sets

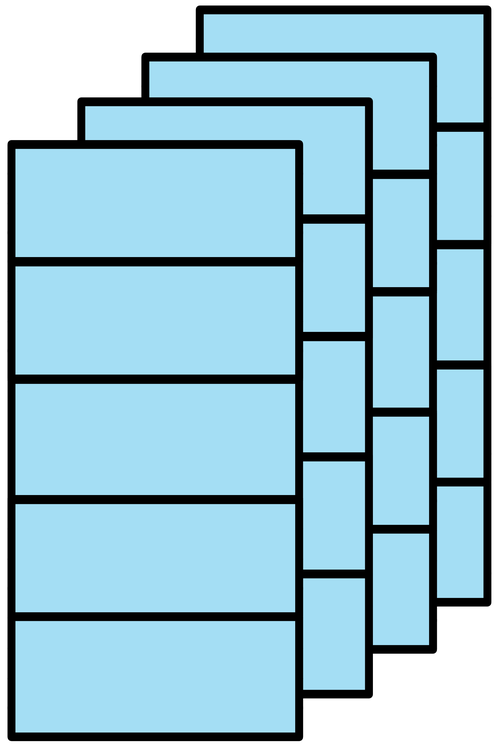
TLB/Cache



TLB/Cache Partitioning to Preserve State

- For security, stateful HW structures flushed+inv on cross-VM context switch
  - L1 (D+I) caches and TLBs, and L2 cache and TLB
- Microservices have relatively small instructions and data working sets

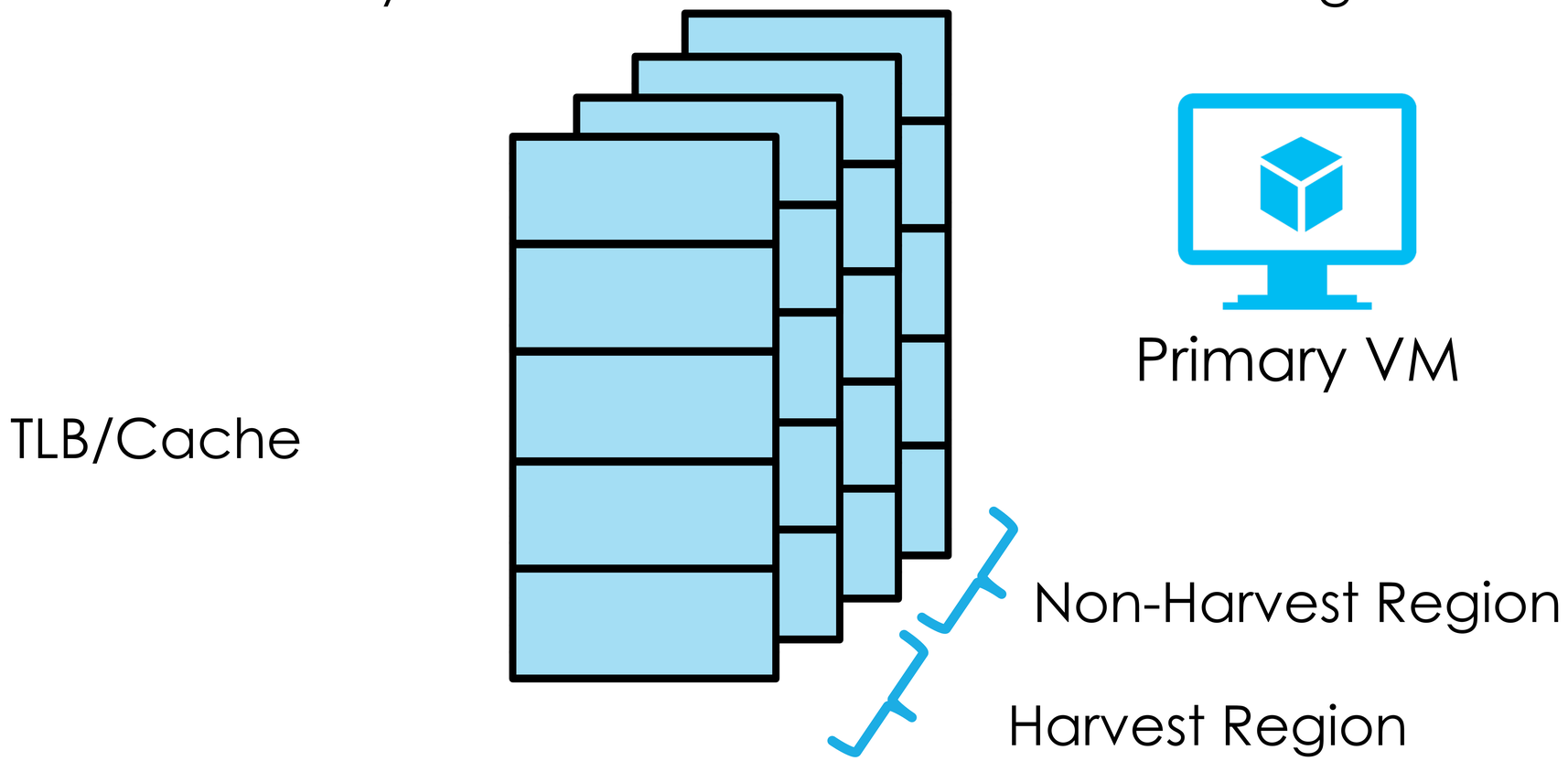
TLB/Cache



Primary VM

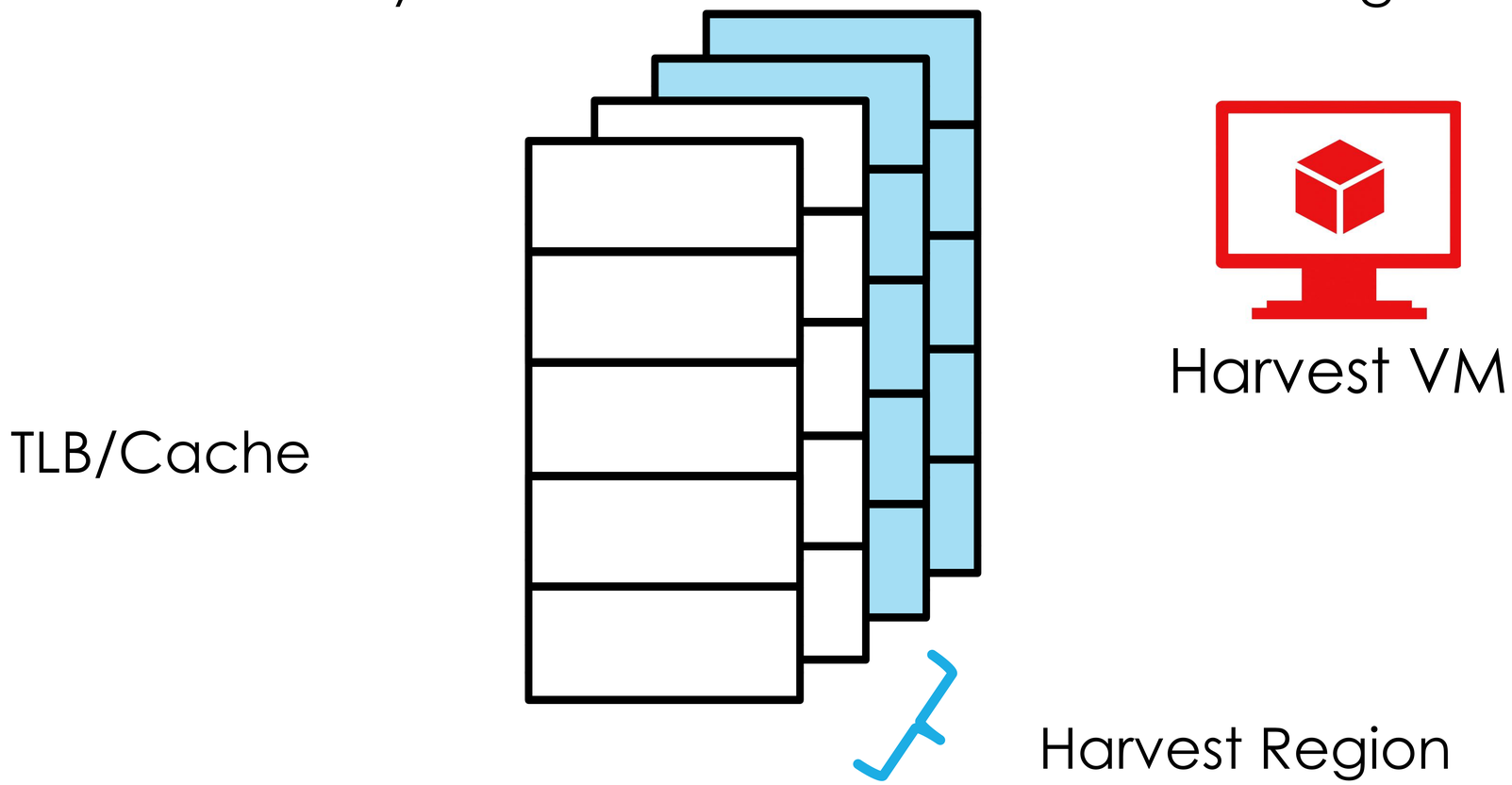
TLB/Cache Partitioning to Preserve State

- For security, stateful HW structures flushed+inv on cross-VM context switch
  - L1 (D+I) caches and TLBs, and L2 cache and TLB
- Microservices have relatively small instructions and data working sets



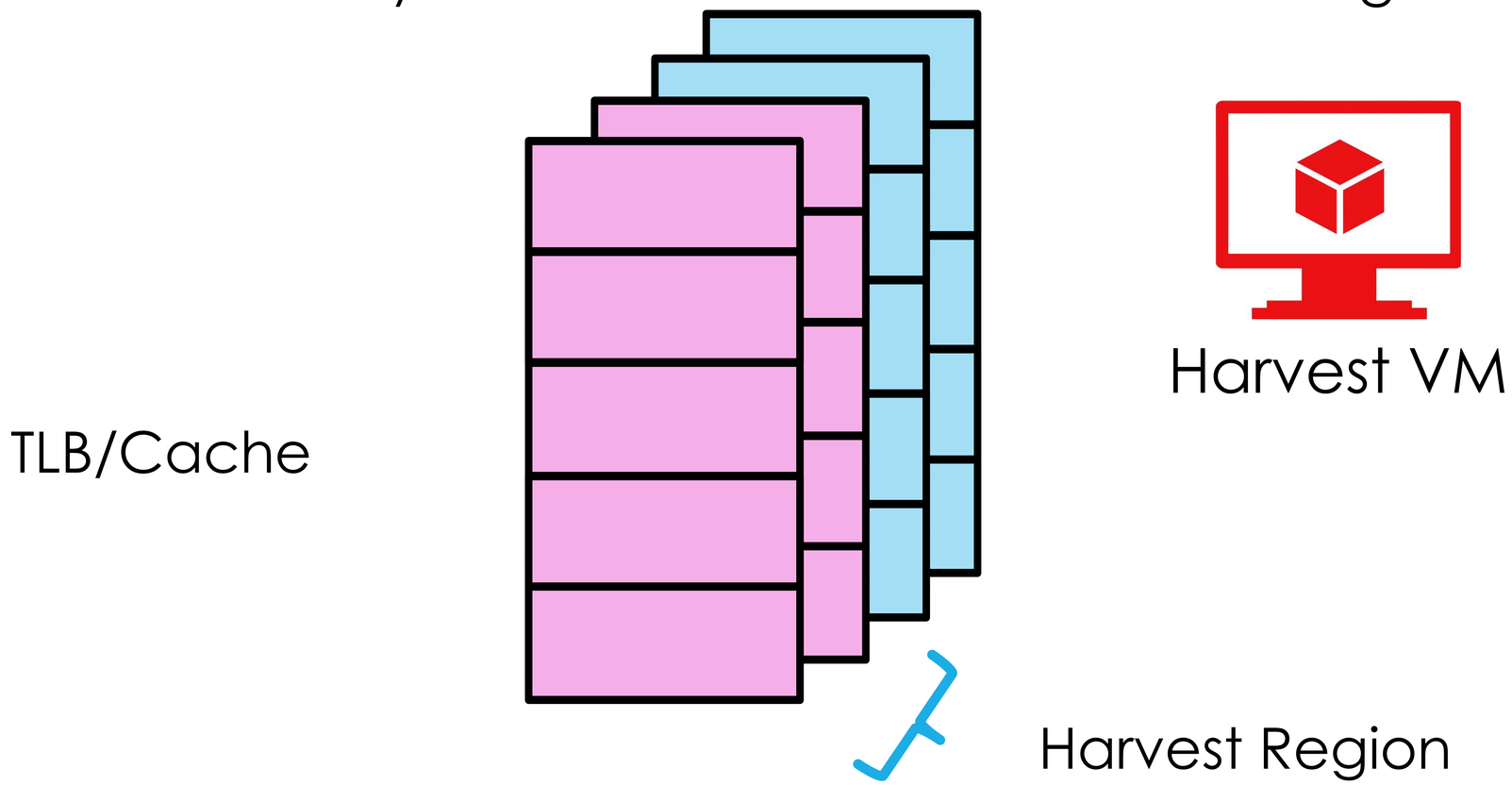
TLB/Cache Partitioning to Preserve State

- For security, stateful HW structures flushed+inv on cross-VM context switch
  - L1 (D+I) caches and TLBs, and L2 cache and TLB
- Microservices have relatively small instructions and data working sets



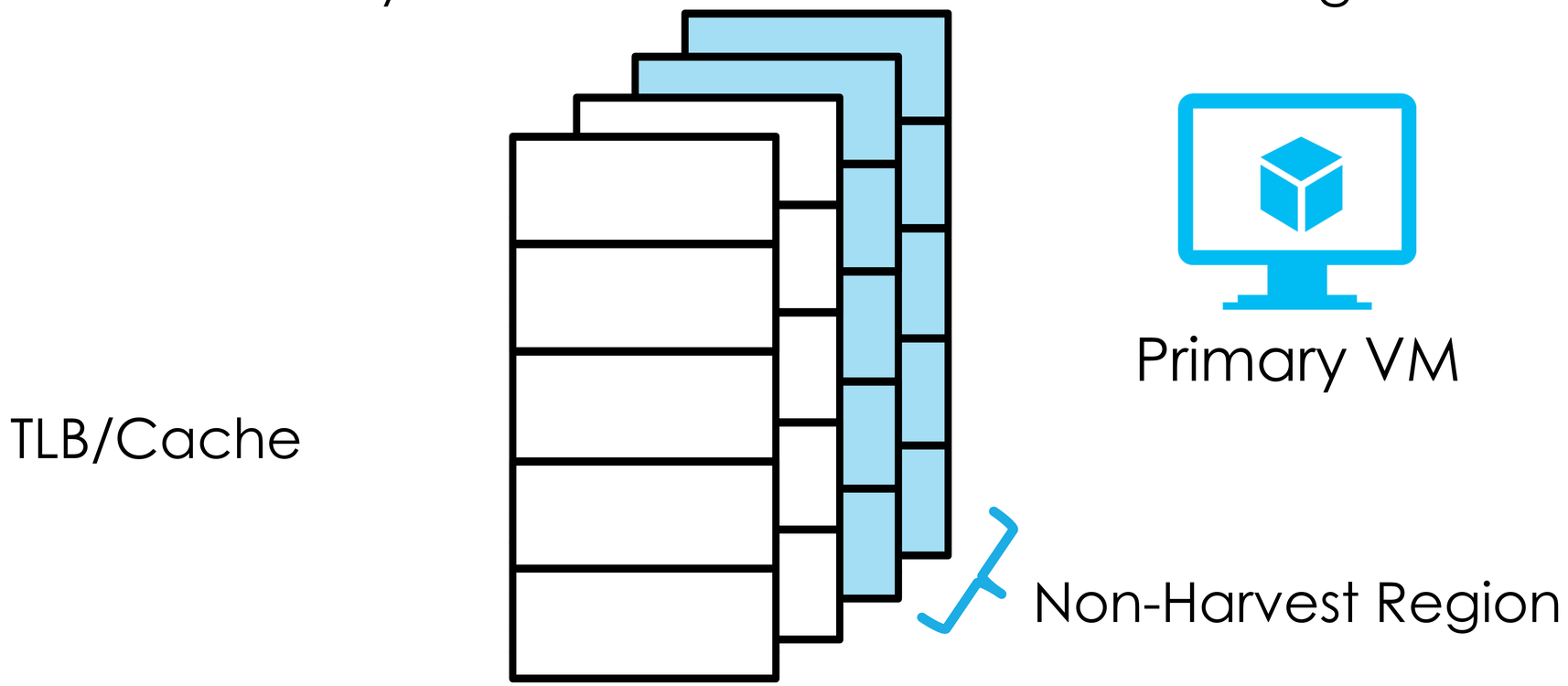
TLB/Cache Partitioning to Preserve State

- For security, stateful HW structures flushed+inv on cross-VM context switch
  - L1 (D+I) caches and TLBs, and L2 cache and TLB
- Microservices have relatively small instructions and data working sets



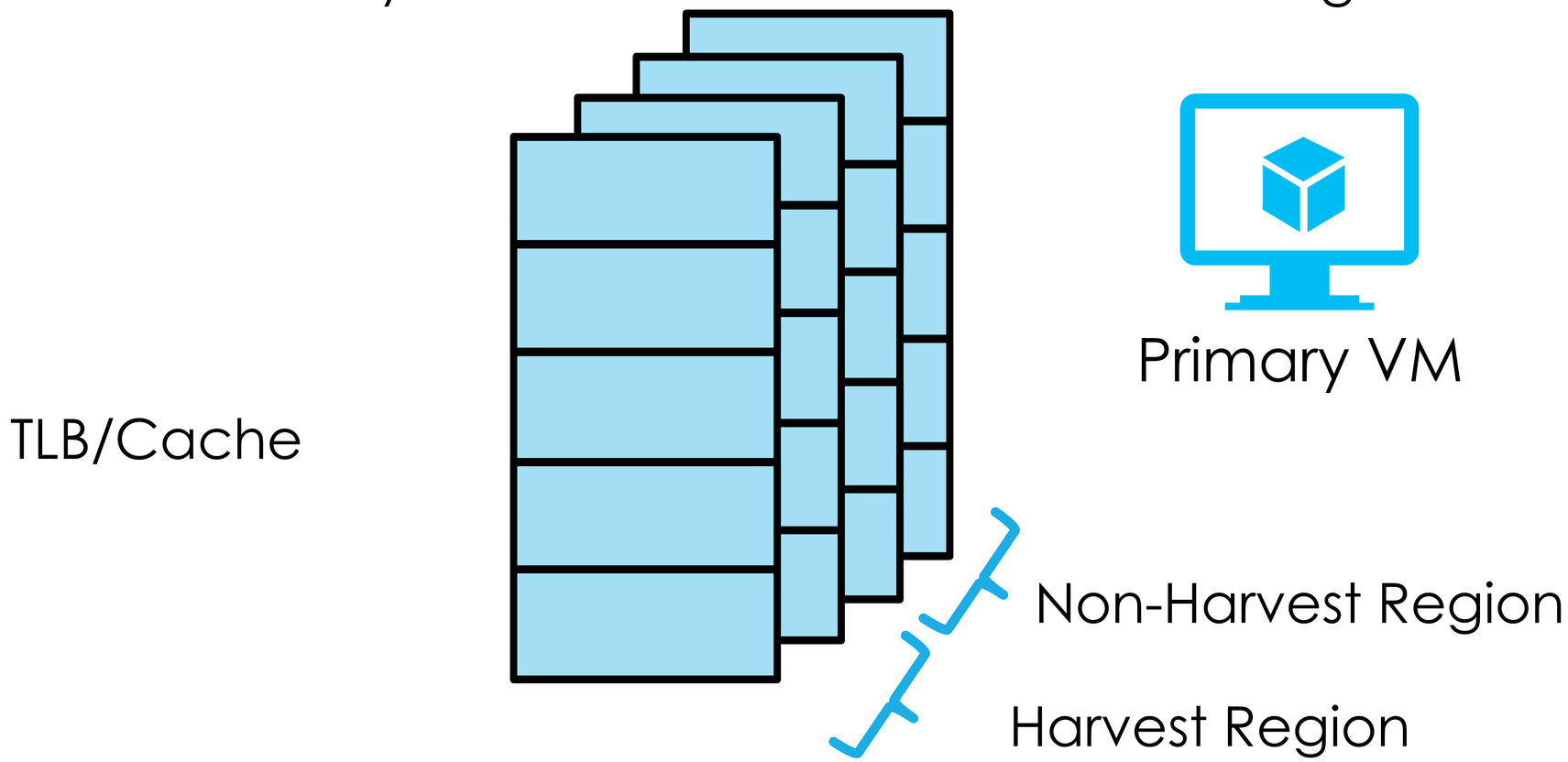
TLB/Cache Partitioning to Preserve State

- For security, stateful HW structures flushed+inv on cross-VM context switch
  - L1 (D+I) caches and TLBs, and L2 cache and TLB
- Microservices have relatively small instructions and data working sets



TLB/Cache Partitioning to Preserve State

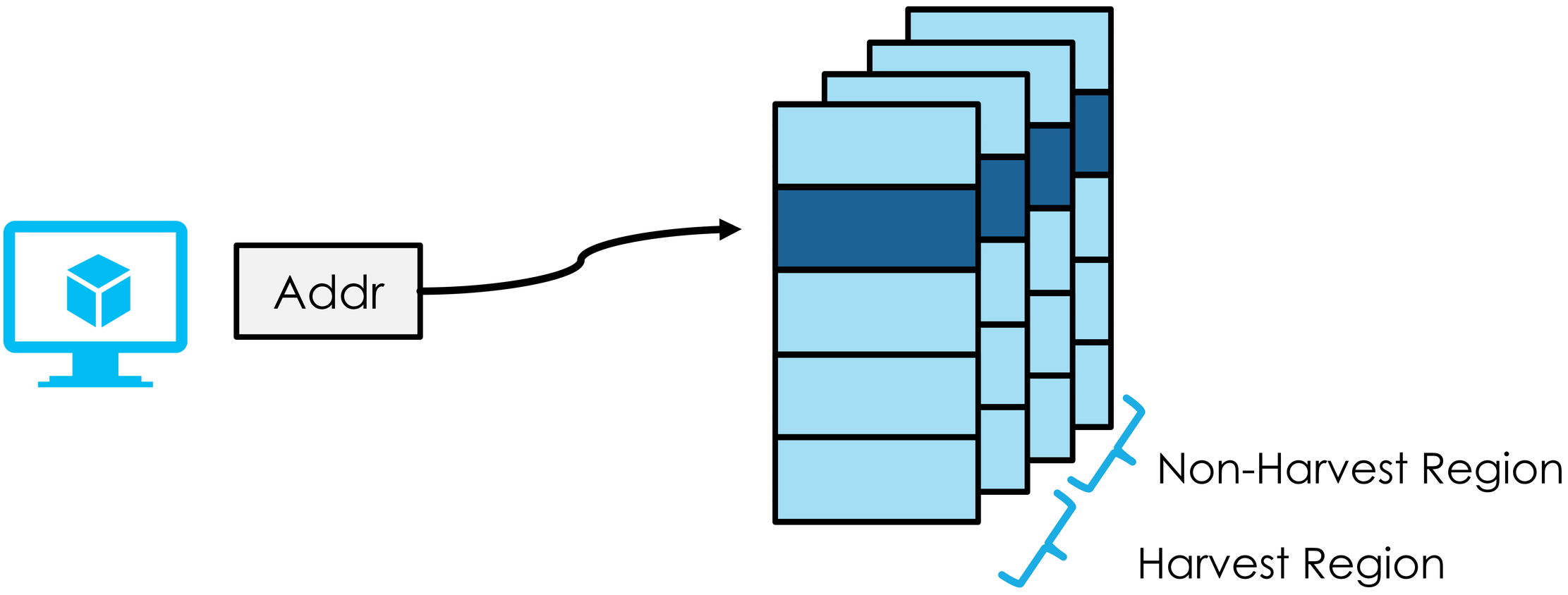
- For security, stateful HW structures flushed+inv on cross-VM context switch
  - L1 (D+I) caches and TLBs, and L2 cache and TLB
- Microservices have relatively small instructions and data working sets



Smart Replacement Policy

- For performance, HardHarvest keeps in Non-Harvest region the data that is most likely to be reused in the future
- *Shared pages*: program code, libraries, read-only input data
  → shared across requests
- *Private pages*: allocated after a microservice request has arrived
  → private per-invocation
- Shared pages more likely to be reused in the future

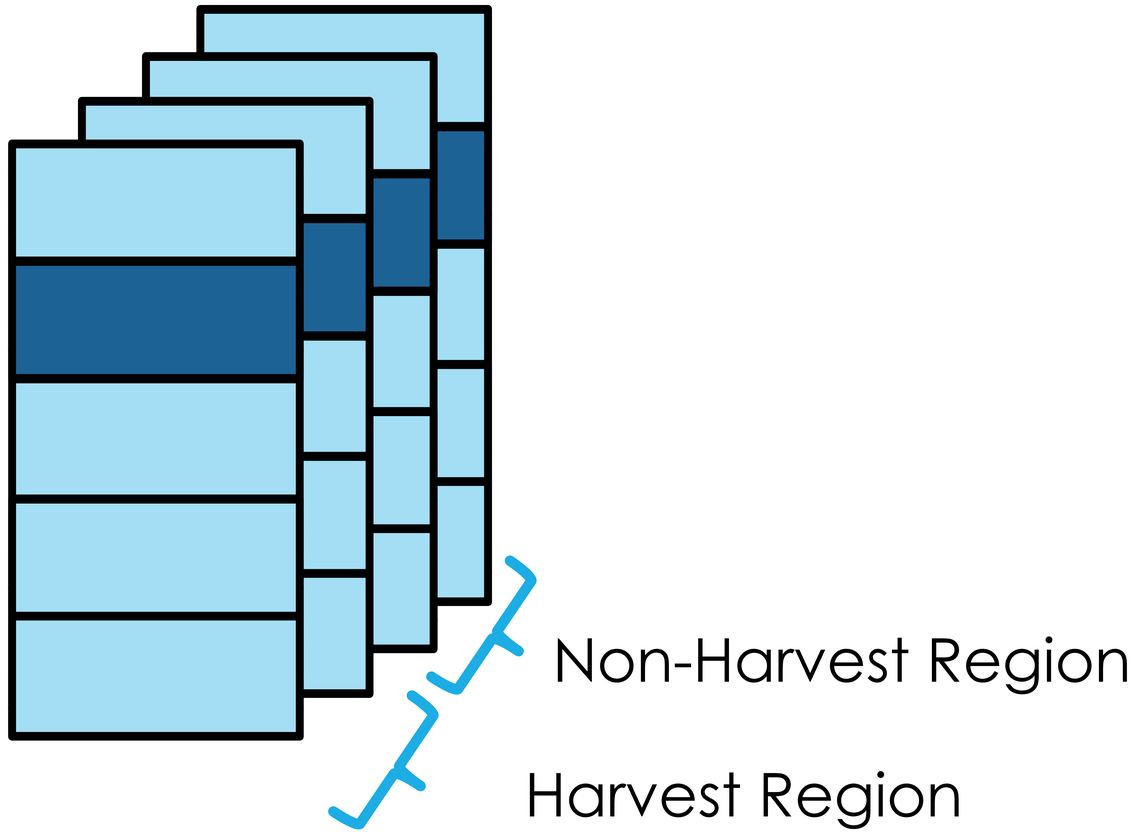
Smart Replacement Policy



Smart Replacement Policy

If any slot empty
→ Pick that slot

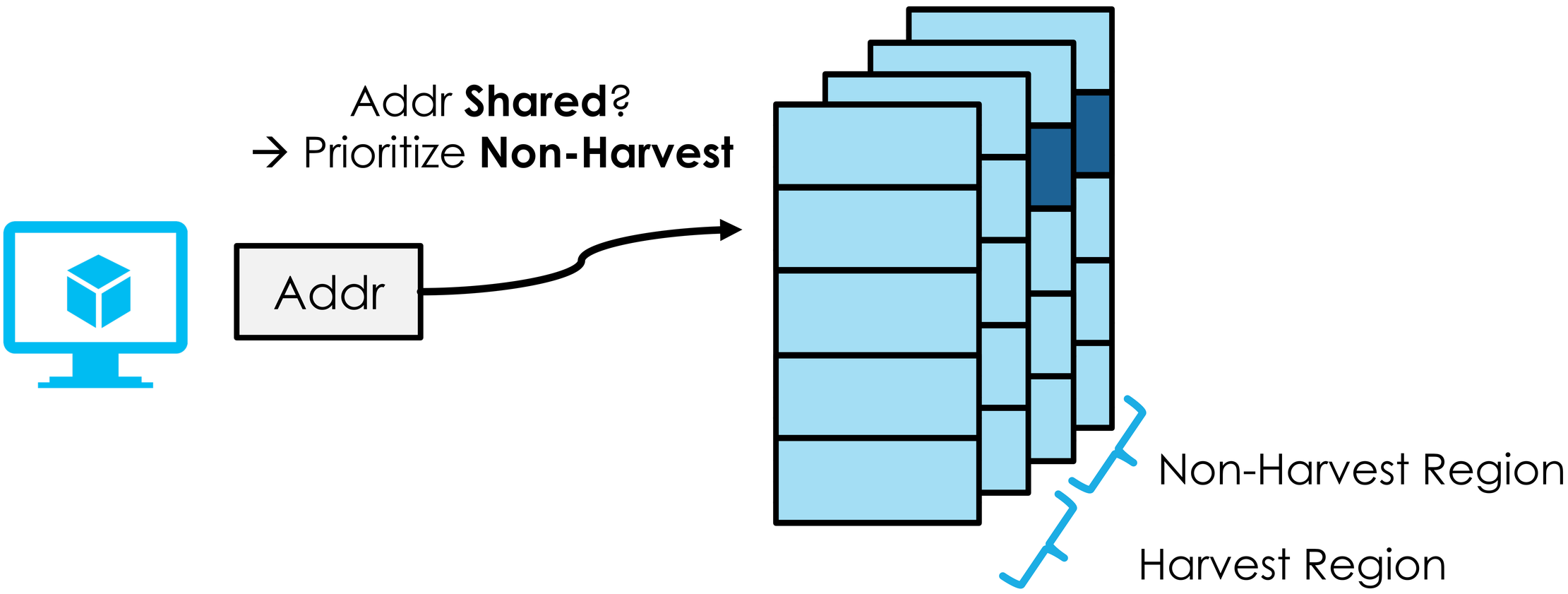
Addr



Non-Harvest Region

Harvest Region

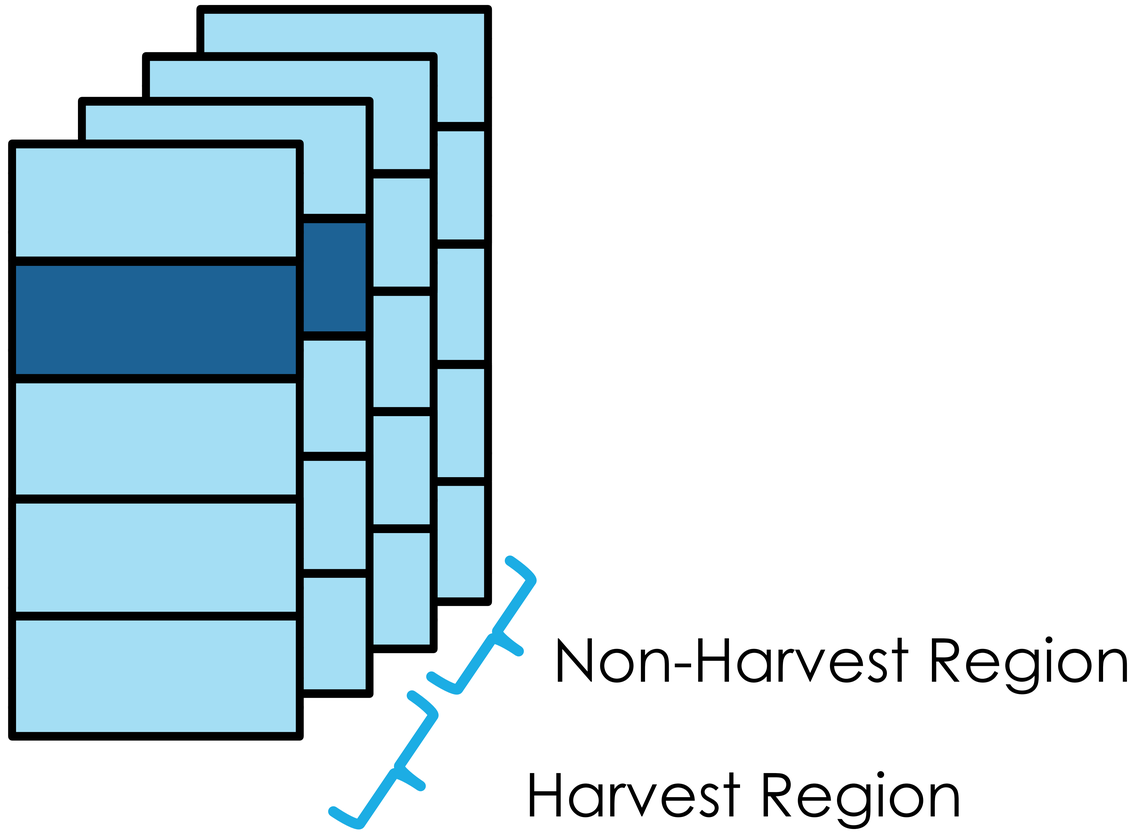
Smart Replacement Policy



Smart Replacement Policy

Addr **Private**?
→ Prioritize **Harvest**

Addr



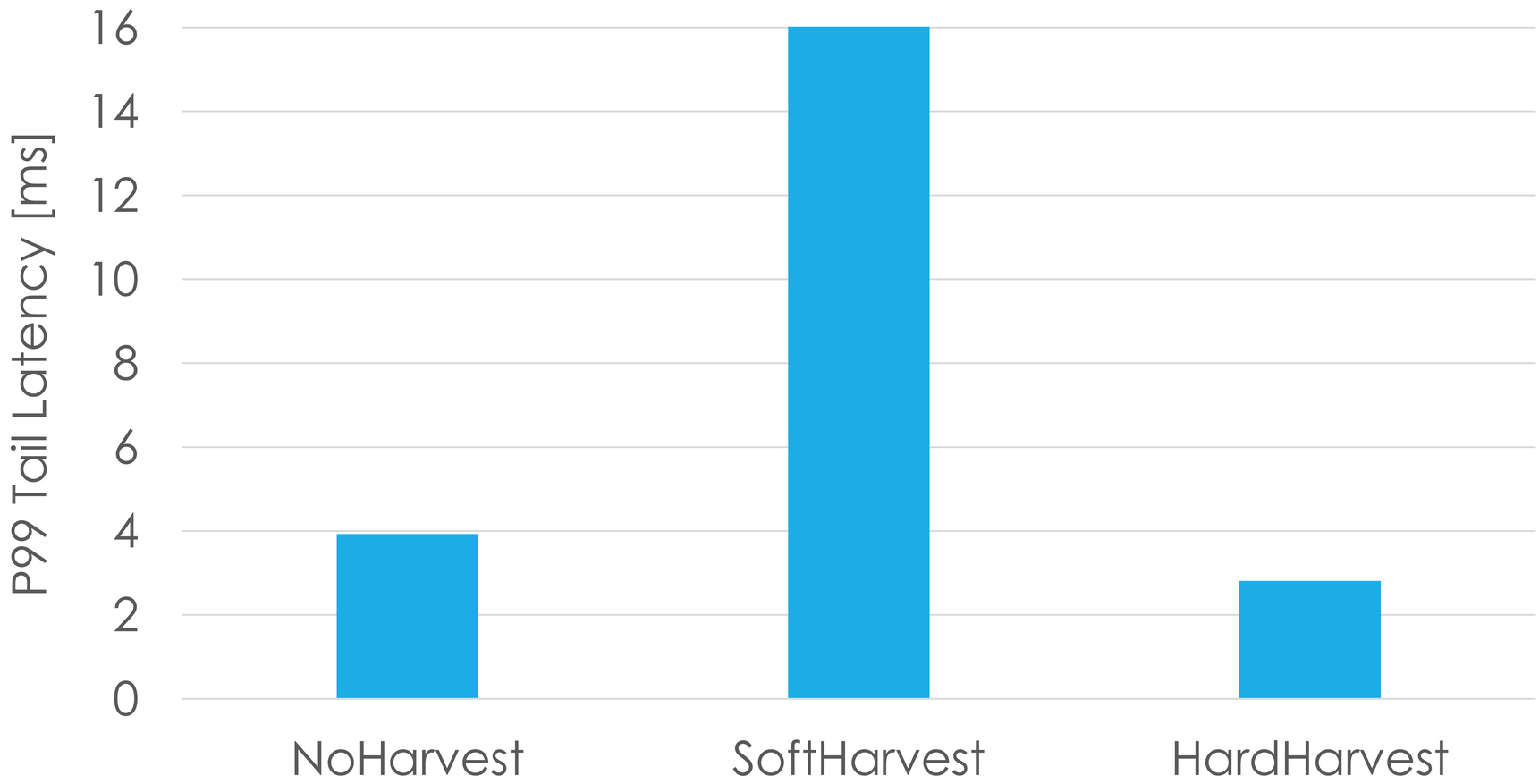
Non-Harvest Region

Harvest Region

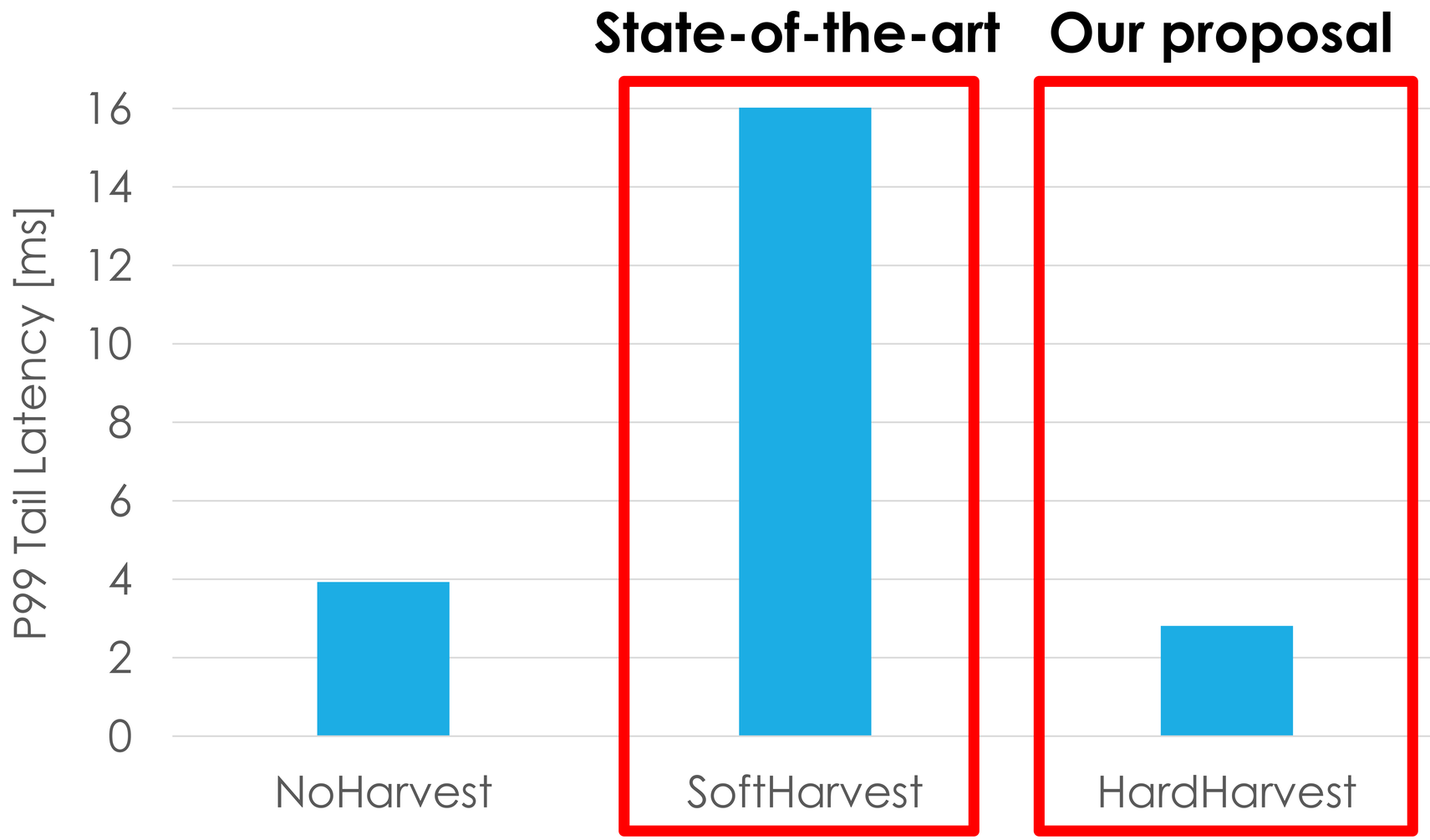
Evaluation Methodology

- Cycle-accurate full-system simulations: SST + QEMU
- DeathStarBench microservices with Alibaba's invocation traces
- Systems evaluated
  - **NoHarvest:** conventional system where no VM performs core harvesting
  - **SoftHarvest (EuroSys'21):** software core harvesting
  - **HardHarvest:** our proposal

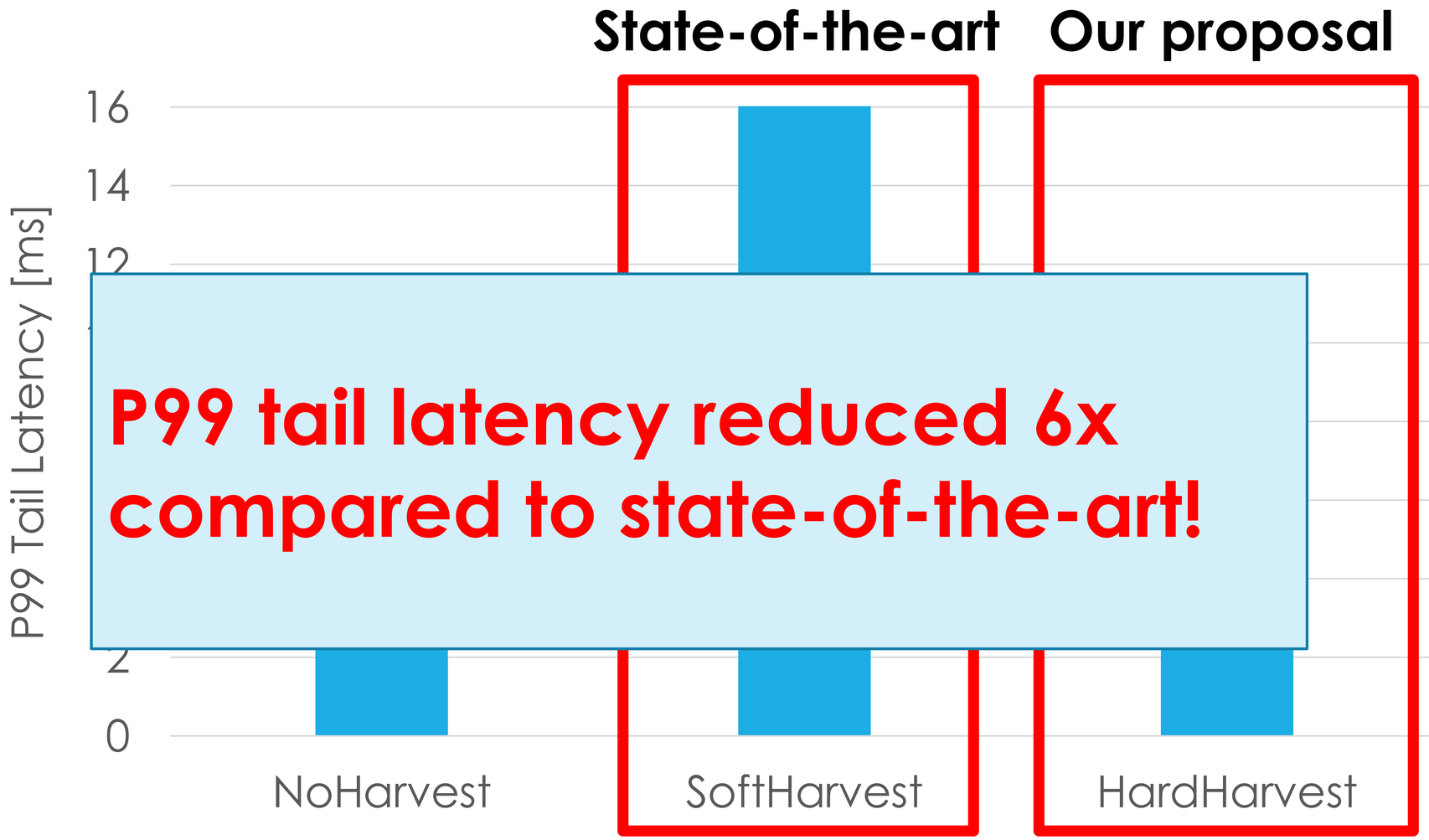
HardHarvest Reduces Primary VMs' Tail Latency



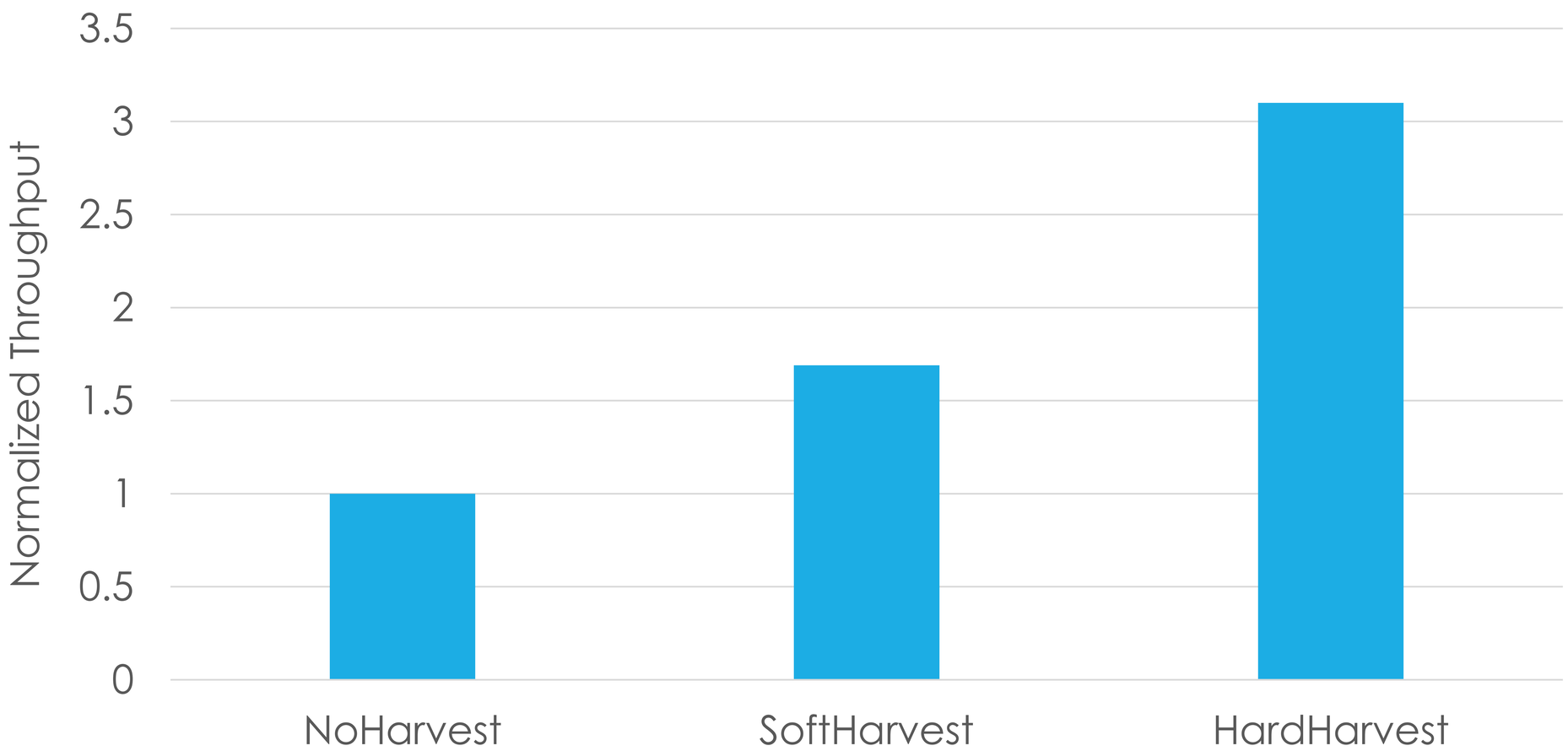
HardHarvest Reduces Primary VMs' Tail Latency



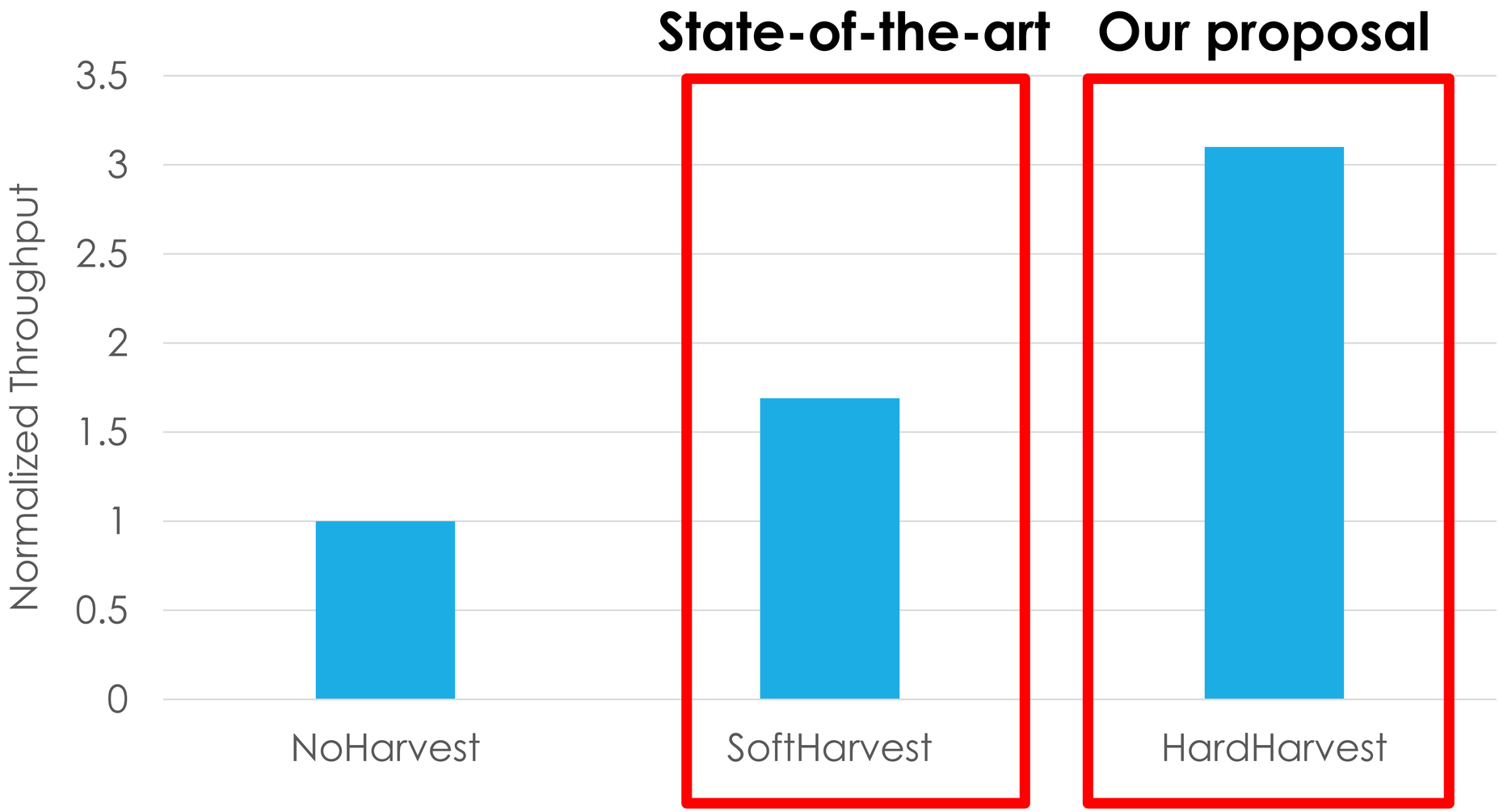
HardHarvest Reduces Primary VMs' Tail Latency



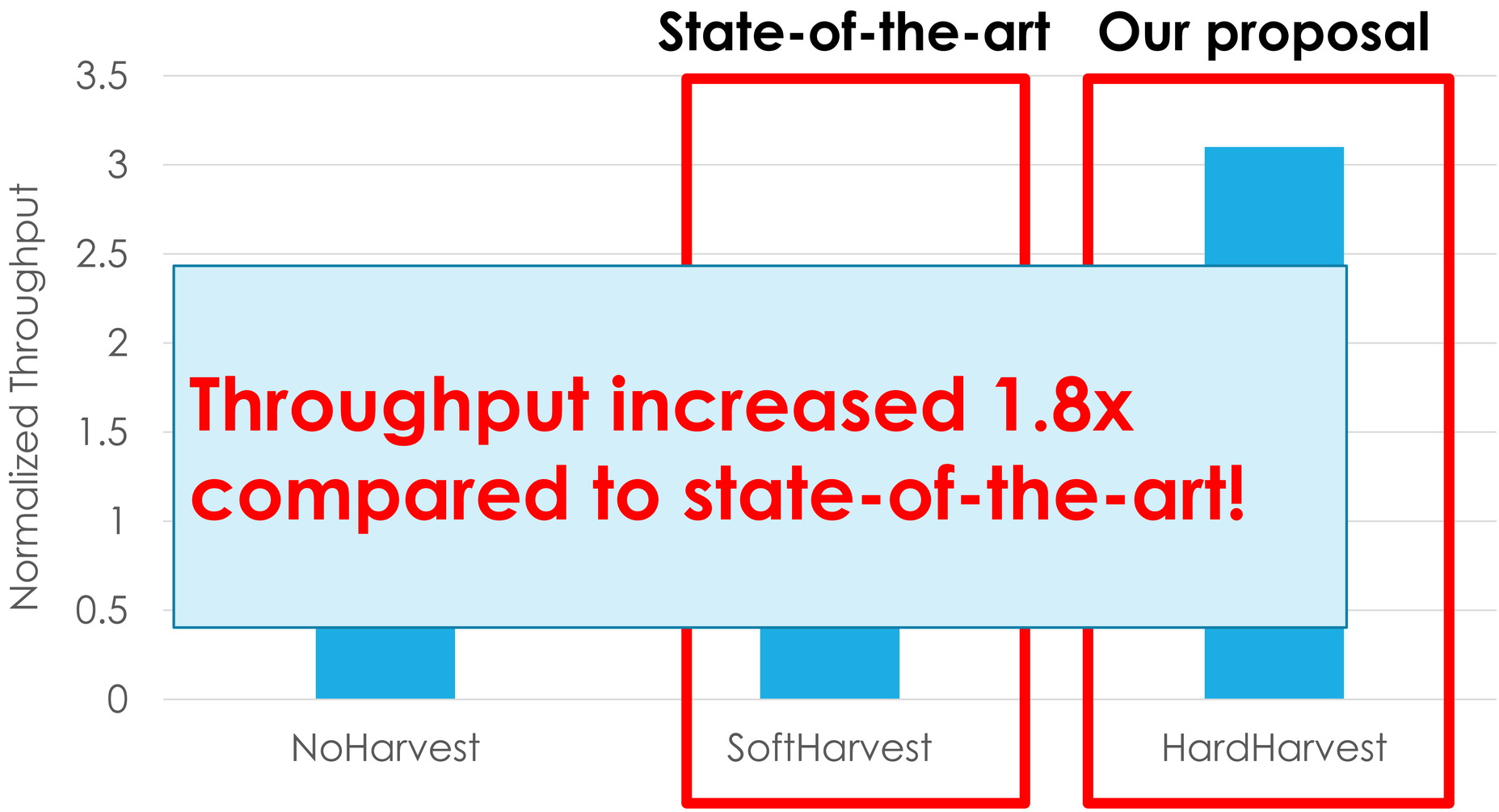
HardHarvest Increases Harvest VMs' Throughput



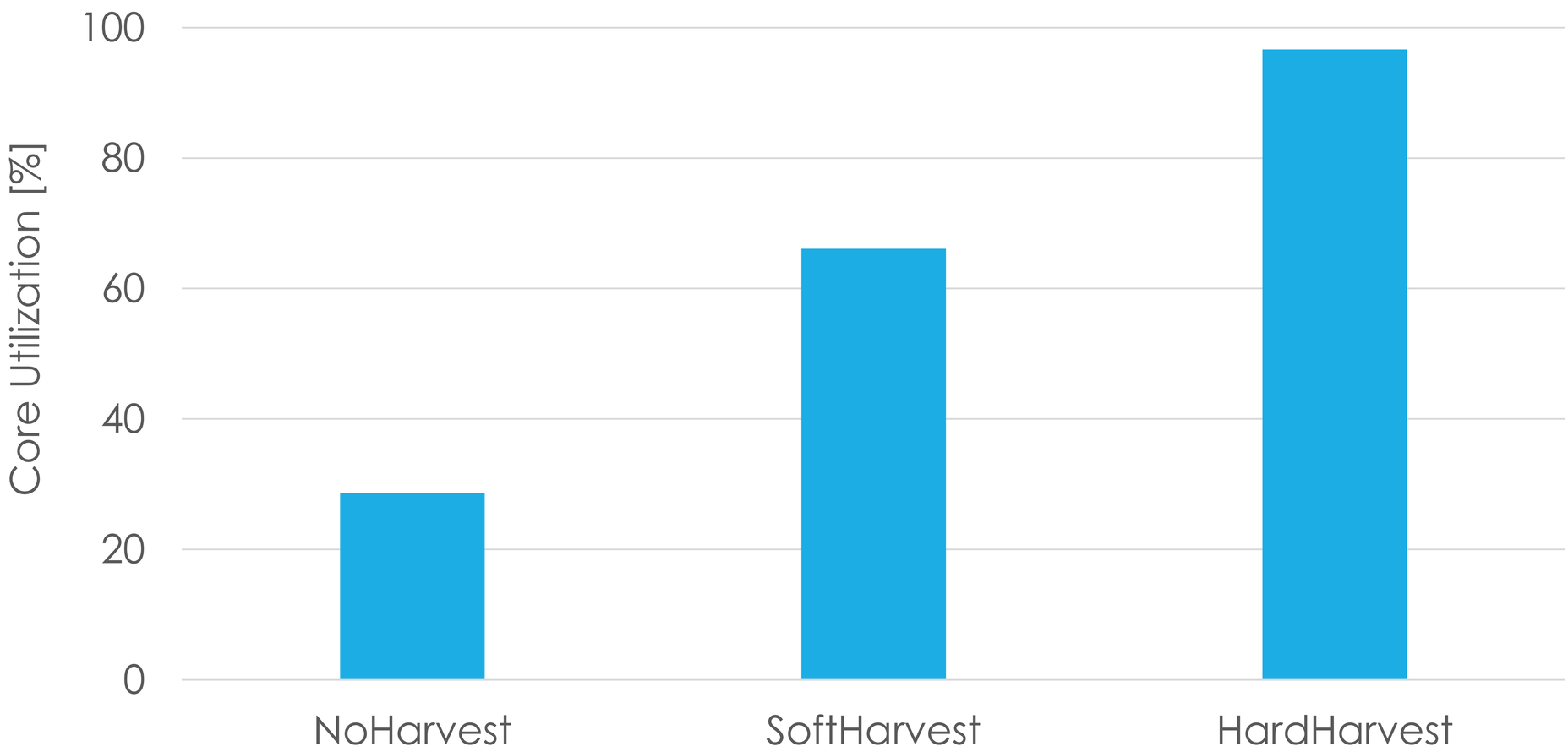
HardHarvest Increases Harvest VMs' Throughput



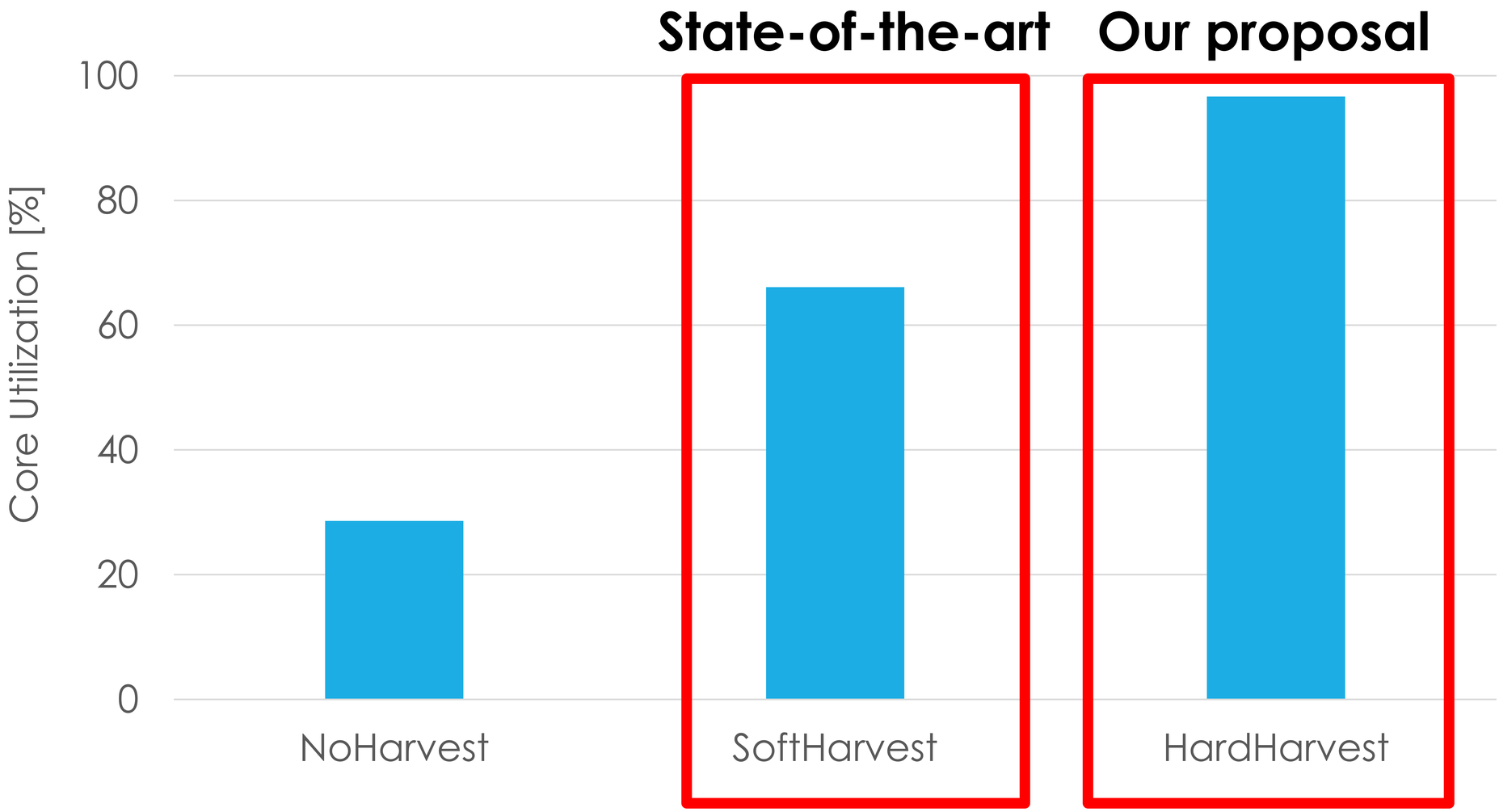
HardHarvest Increases Harvest VMs' Throughput



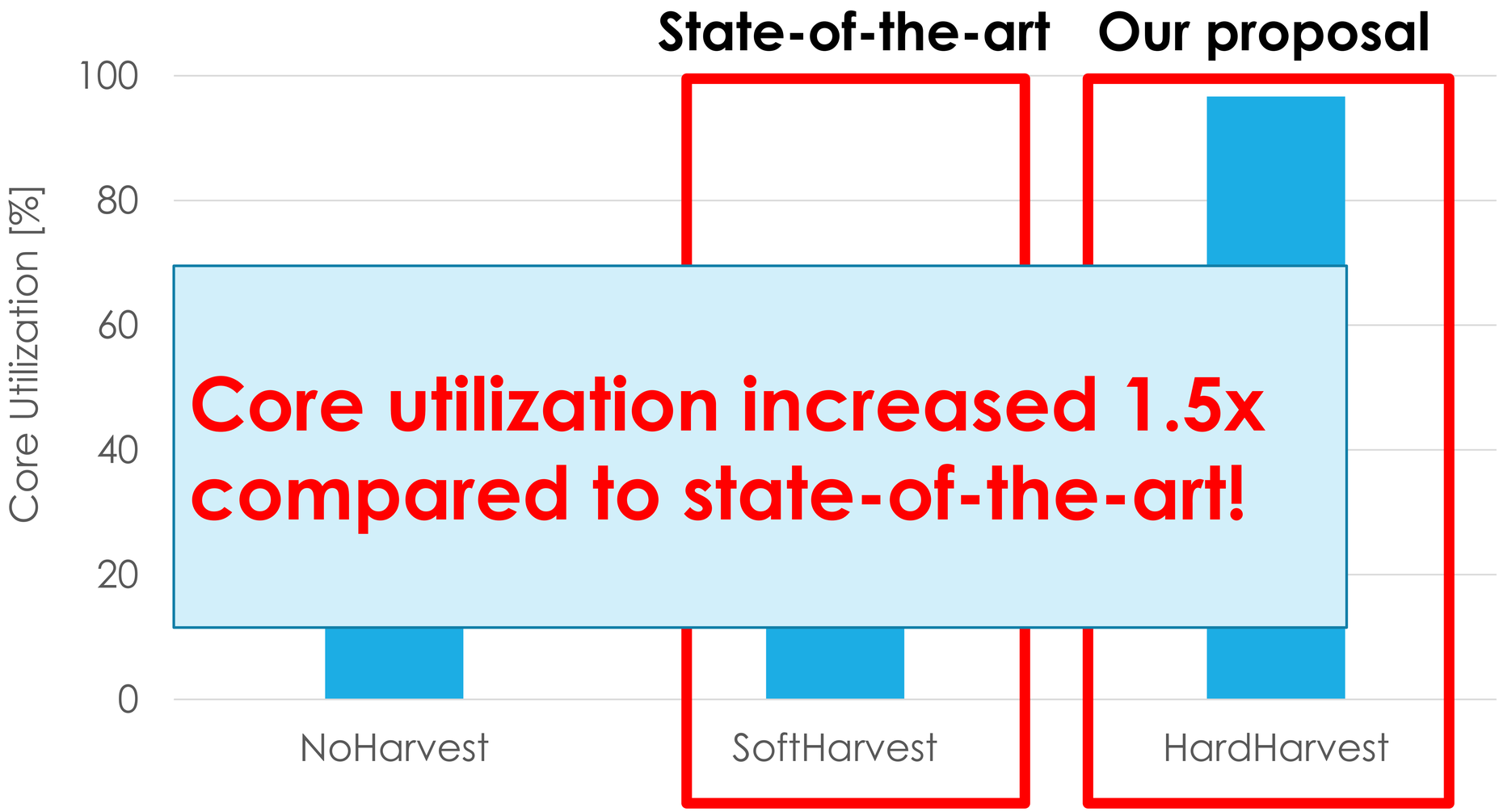
HardHarvest Increases Core Utilization



HardHarvest Increases Core Utilization



HardHarvest Increases Core Utilization



Conclusions

- Microservices beneficial, but have low core utilization
- How to design efficient core harvesting scheme for microservices?
- HardHarvest: the first architecture for in-hardware core harvesting
  - 6x lower P99 tail latency of Primary VMs, 1.8x higher throughput of Harvest VMs, and 1.5x higher core utilization

HardHarvest: Hardware-Supported Core Harvesting for Microservices

ISCA '25

Jovan Stojkovic, Chunao Liu*, Muhammad Shahbaz*, Josep Torrellas

University of Illinois at Urbana-Champaign, *Purdue University