Optimizing Storage Performance with Calibrated Interrupts

Amy Tai[‡*]
Igor Smolyar[†*]
Michael Wei[‡]
Dan Tsafrir[†‡]

[†] TECHNION Israel Institute of Technology

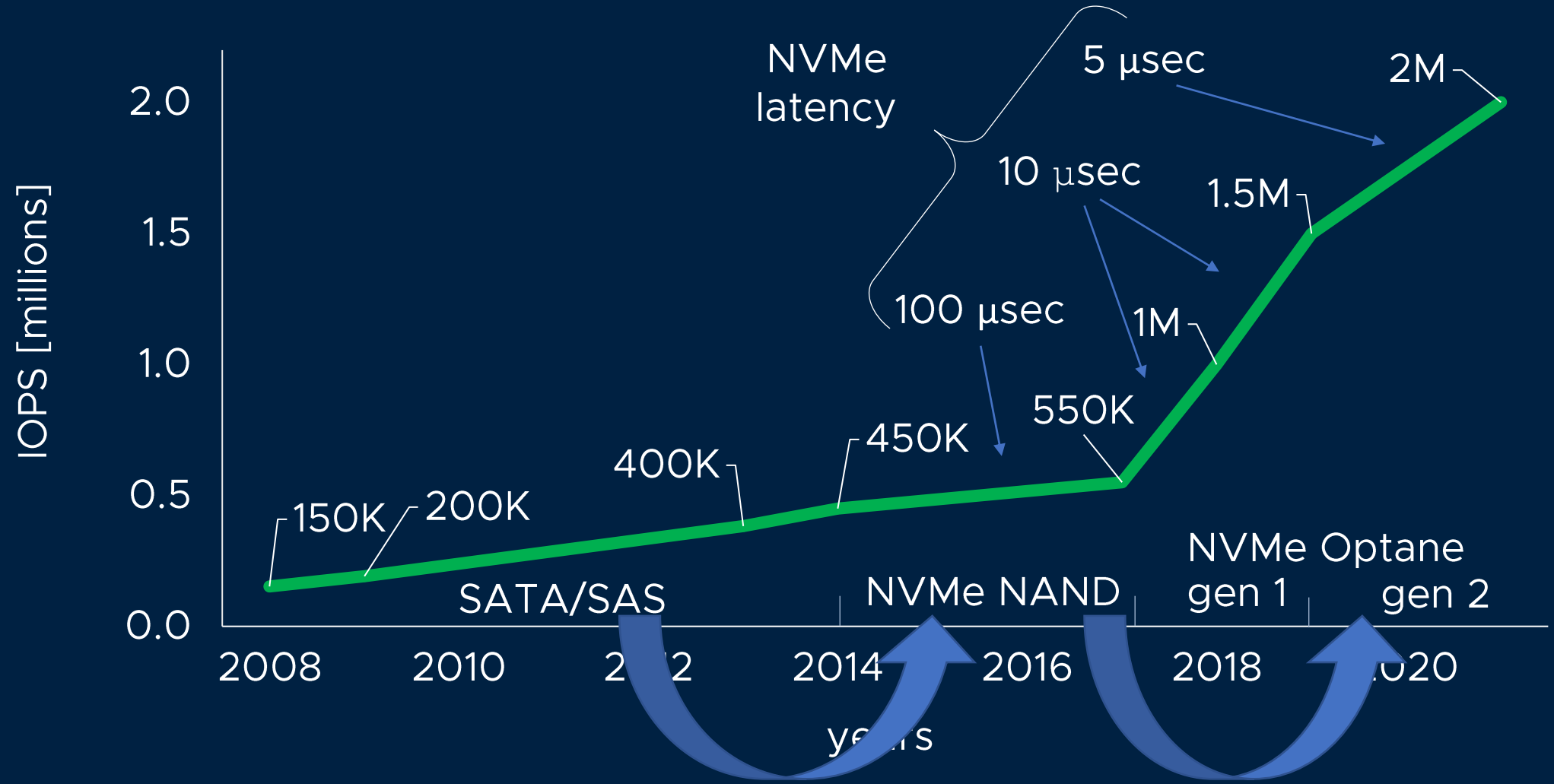
[‡] vmware®

* Denotes co-first authors with equal contribution

OSDI 2021

Presenter: Igor Smolyar

Motivation: rapid increase in storage performance

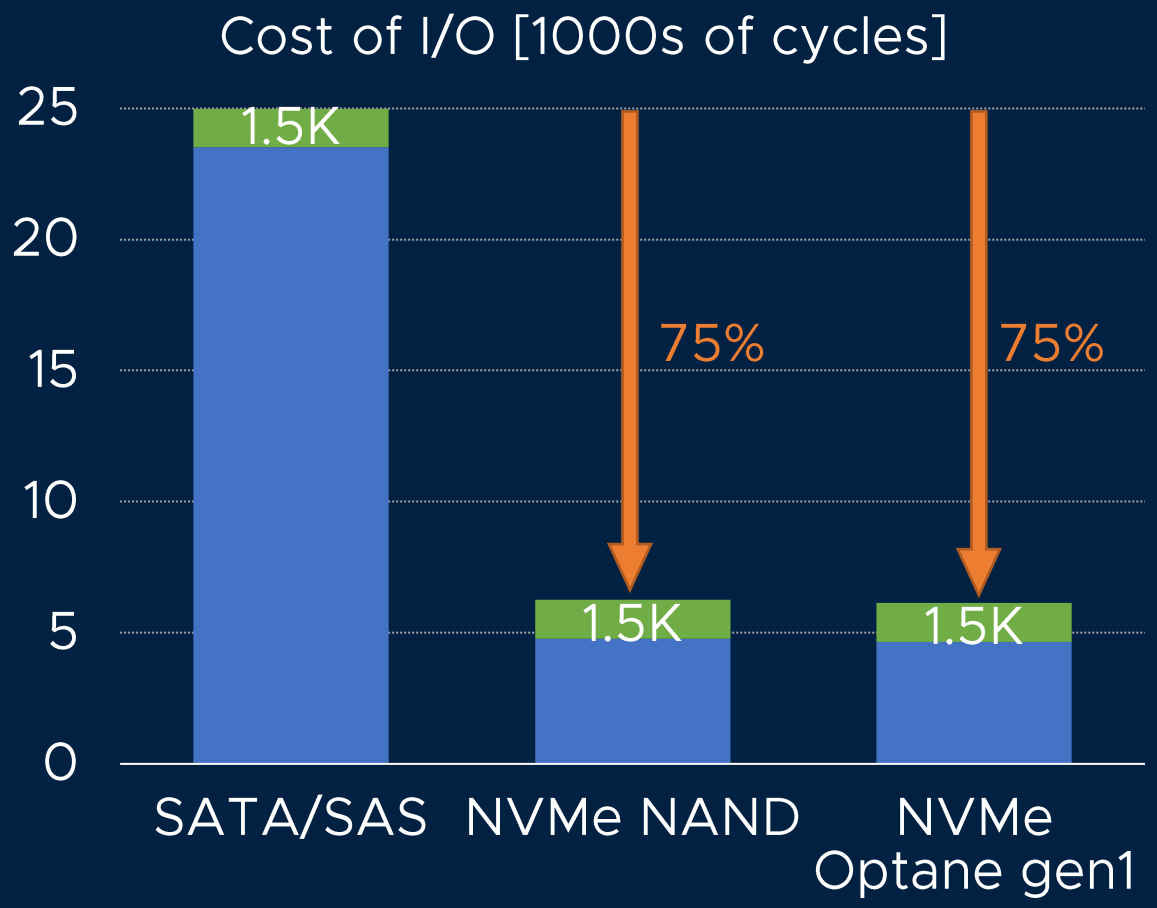


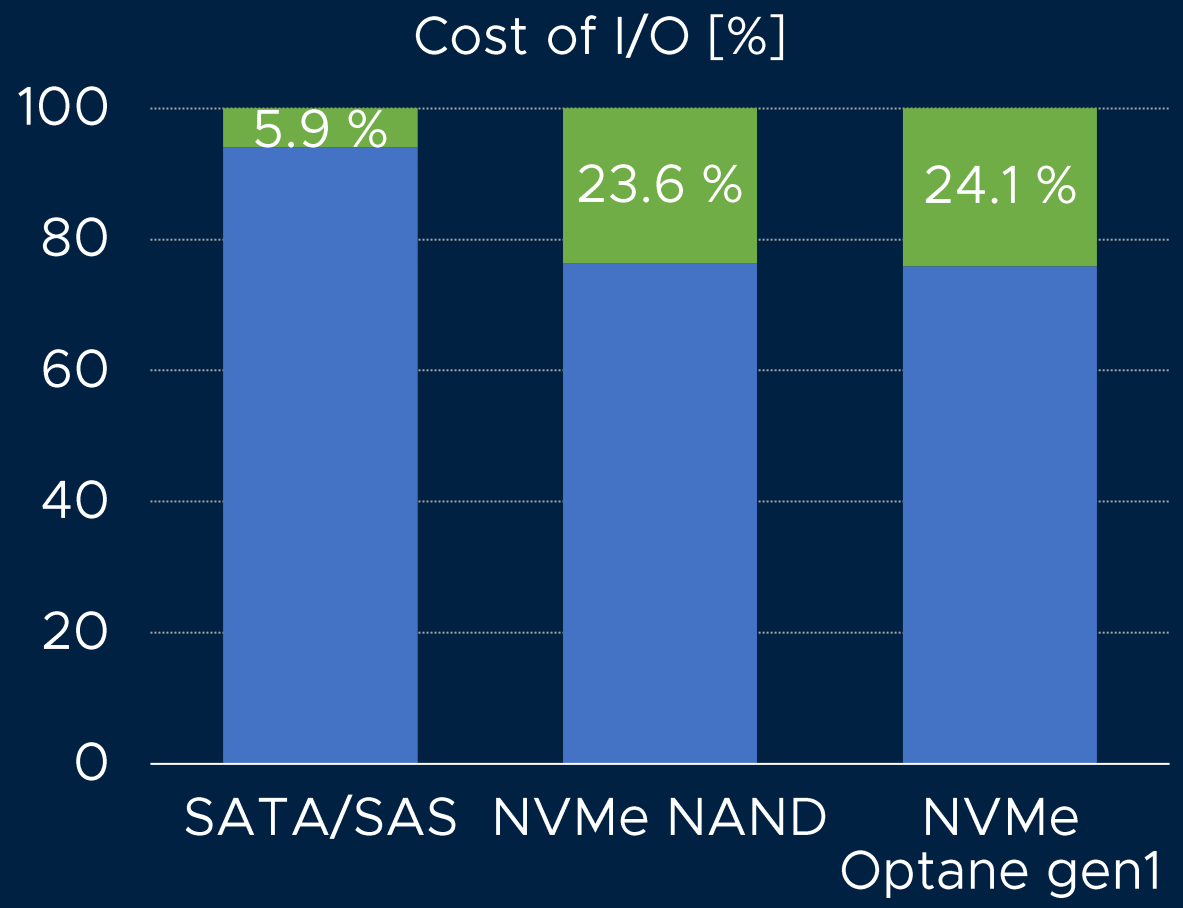
1. More efficient I/O interface + optimized software stack

2. SSDs internals---latency improvements

Motivation 1: faster I/O interface--software and hardware

CPU-bound async workload, io depth 256



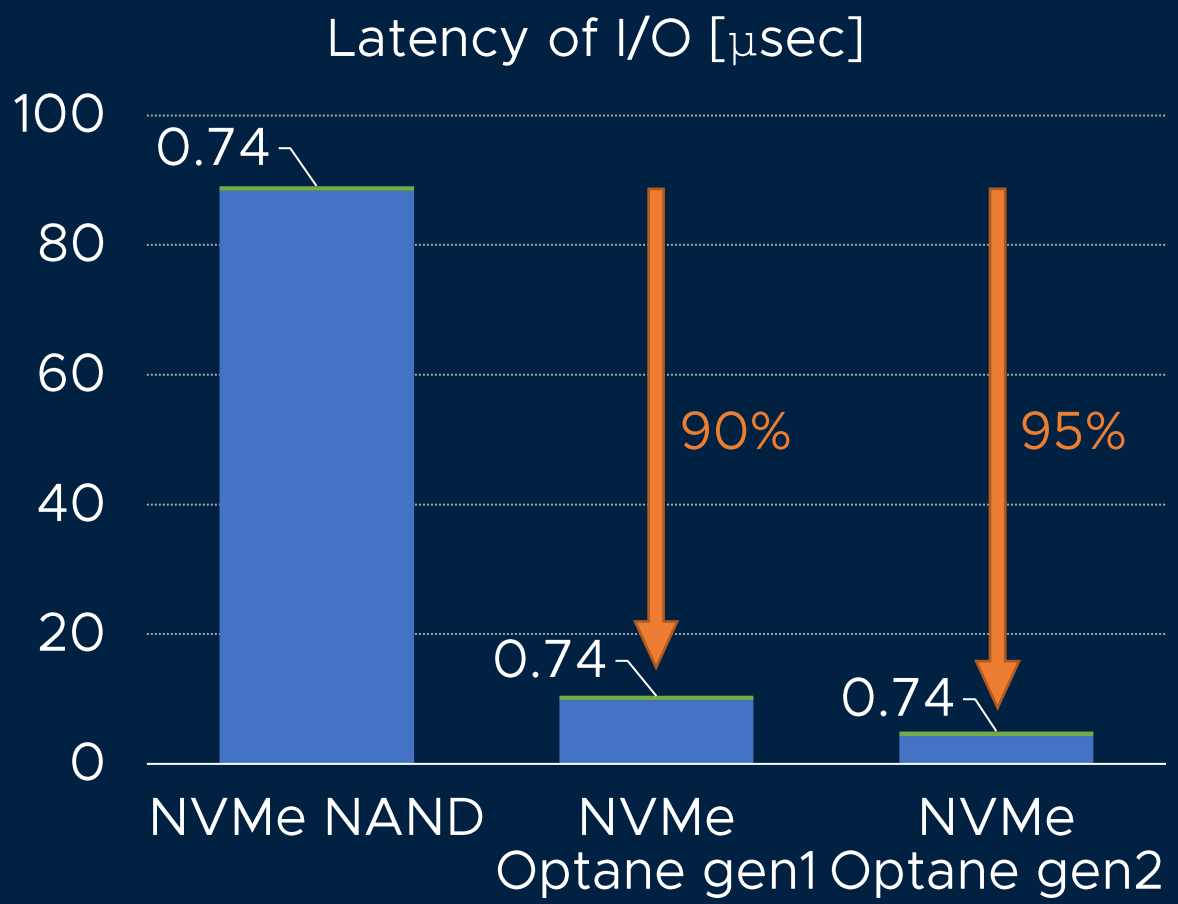


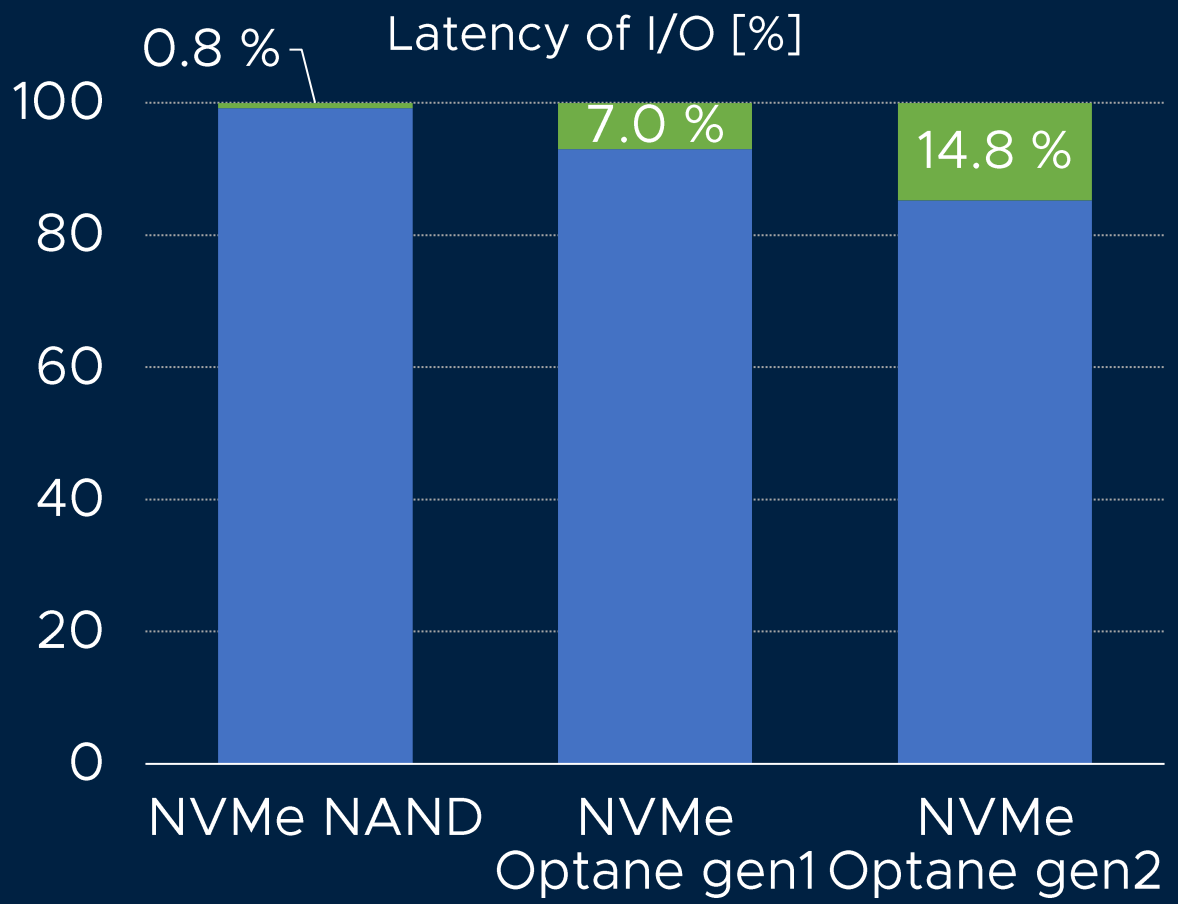
interrupt

other

Motivation 2: SSDs latency improvements

Device-bound sync workload, io depth 1





interrupt

other

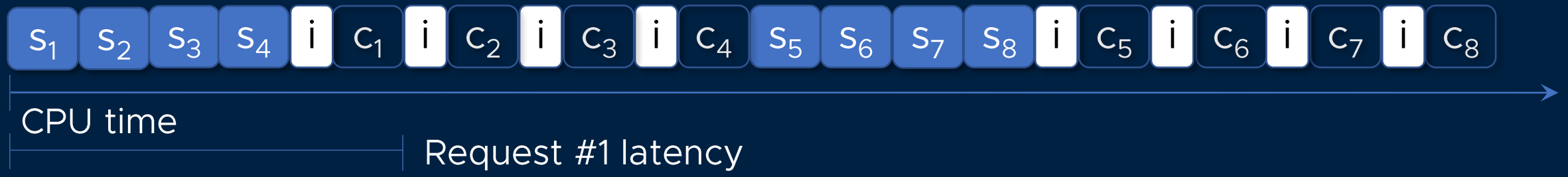
Interrupts are expensive.

Firing an interrupt on each completion hampers performance.

Must Coalesce!

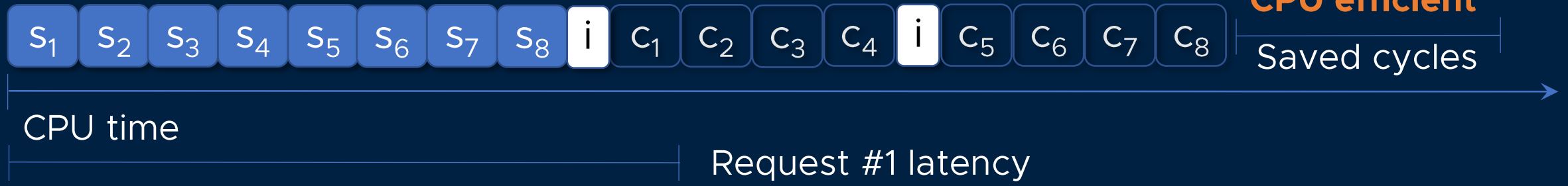
Interrupt coalescing

No coalescing: interrupt on each request



Coalescing: interrupt every 4 requests

Coalescing is CPU efficient



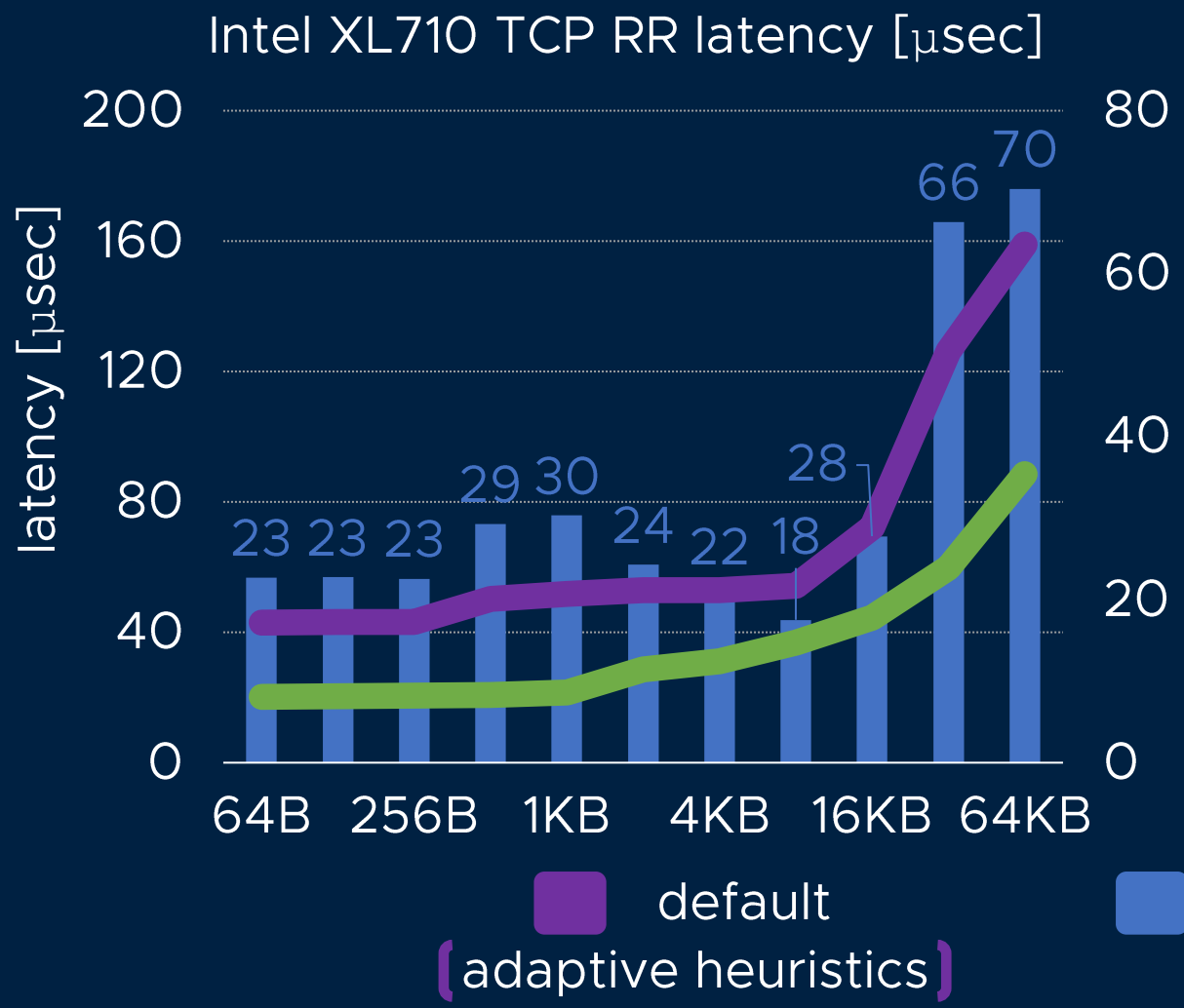
Coalescing problem: latency/throughput trade-off
Which request is latency sensitive?

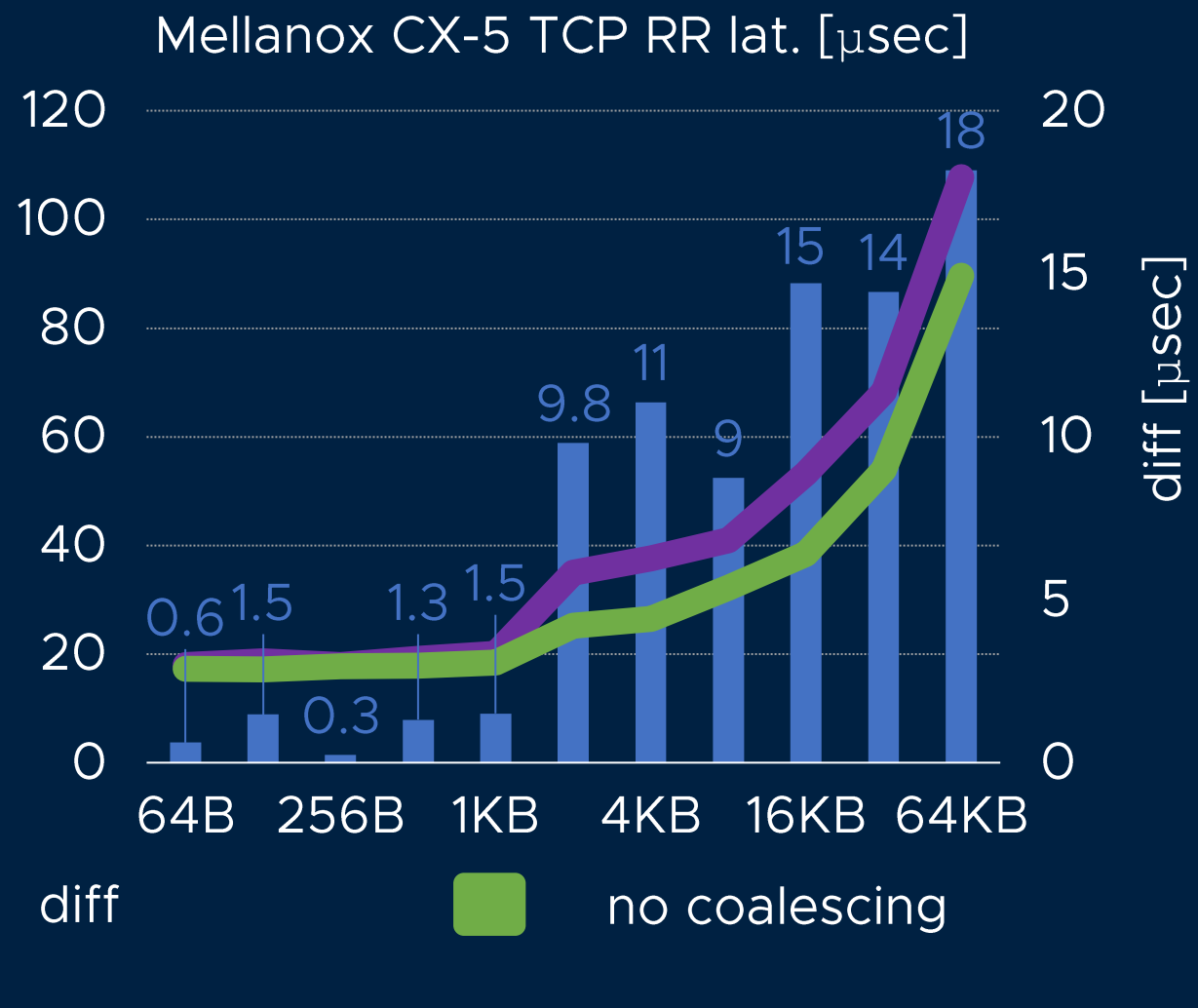
Eternal question: when to generate an interrupt?

- Difficult question!
- Can heuristics help?
  - Sense the workload and act accordingly
- Let’s see how heuristics work in NICs

Eternal question: when to generate an interrupt?

Heuristics are suboptimal





Devices resort to use heuristics,
which is suboptimal for performance.

Problem: Semantic Gap

Missing Semantics:
when to generate an interrupt?

Our Solution

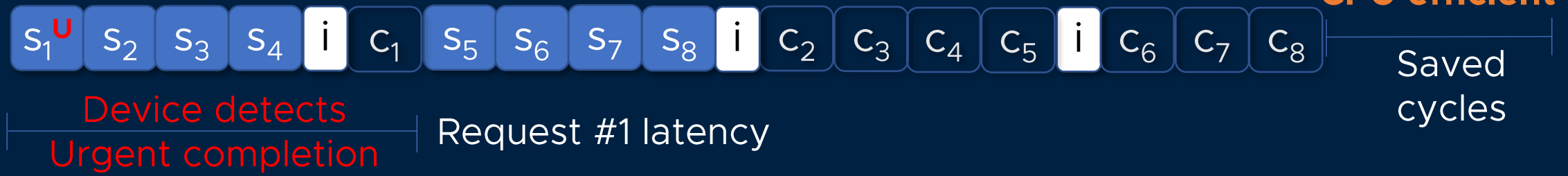
Software has the knowledge when it needs an interrupt.
I/O requests should convey this knowledge to the device.

For NVMe devices we calibrate* interrupts
with Urgent and Barrier flags.

* To calibrate: to adjust precisely for a particular function

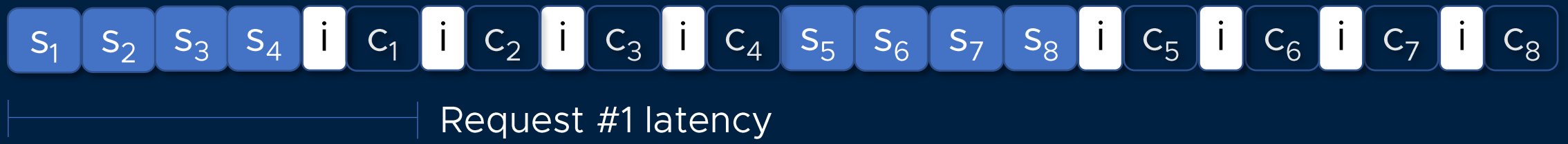
Semantics: Urgent flag for latency sensitive request #1

Cinterrupts: on Urgent fire immediately, coalesce the rest

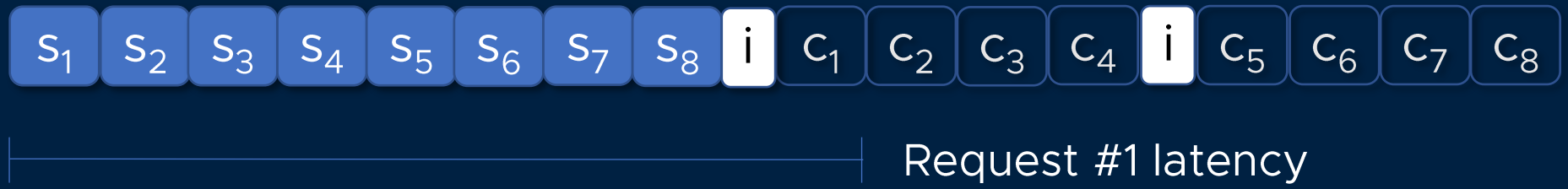


Cinterrupts is CPU efficient

No coalescing: interrupt on each request

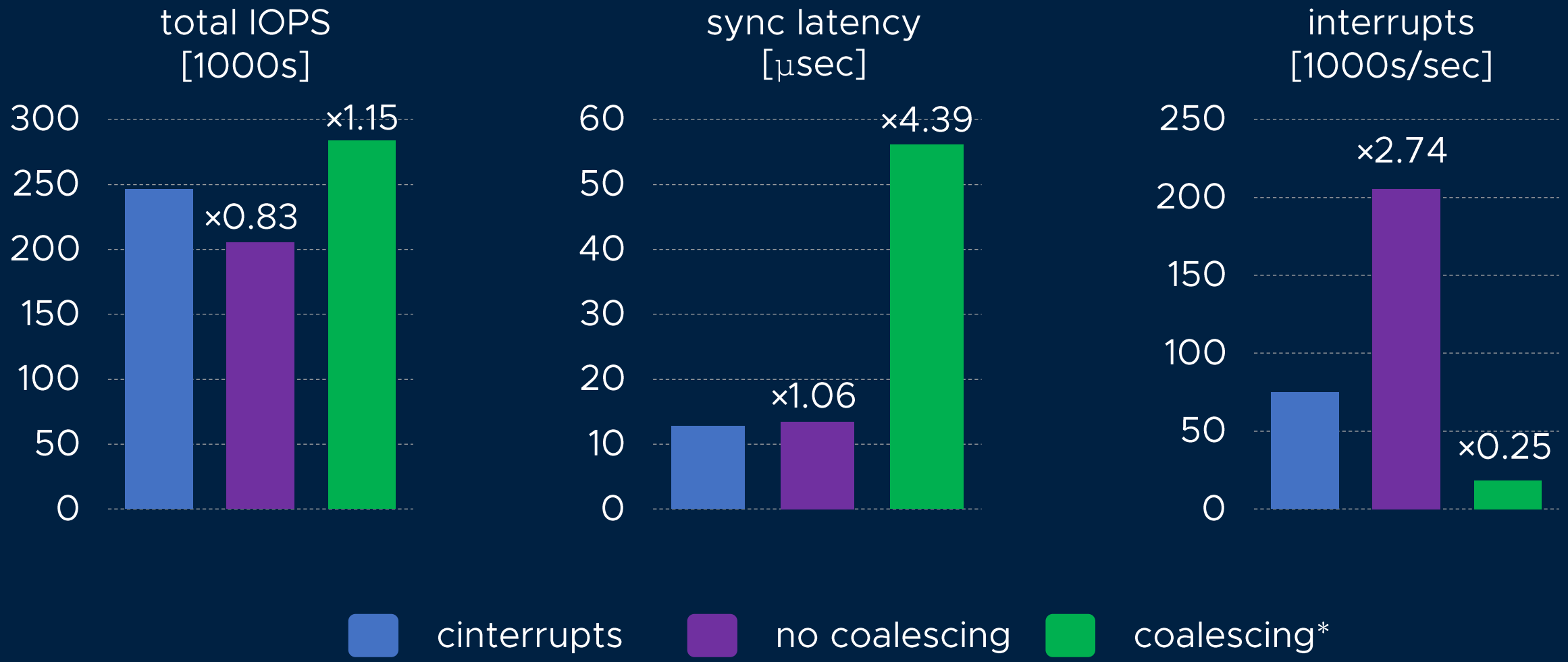


Coalescing: interrupt every 4 requests



Performance benefits of Urgent

Mixed workload, two threads: sync (urgent) and async (throughput)



* Cinterrupts adaptive coalescing, superior to NVMe coalescing

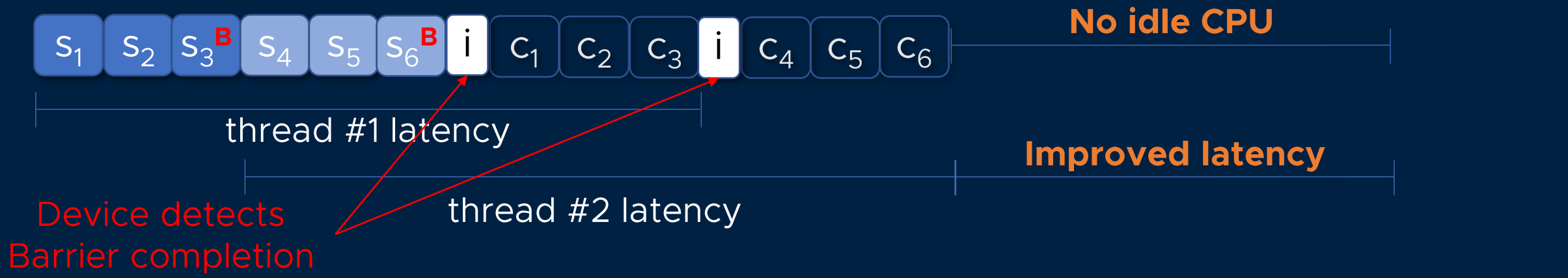
Urgent improves latency.

Can we also optimize throughput?

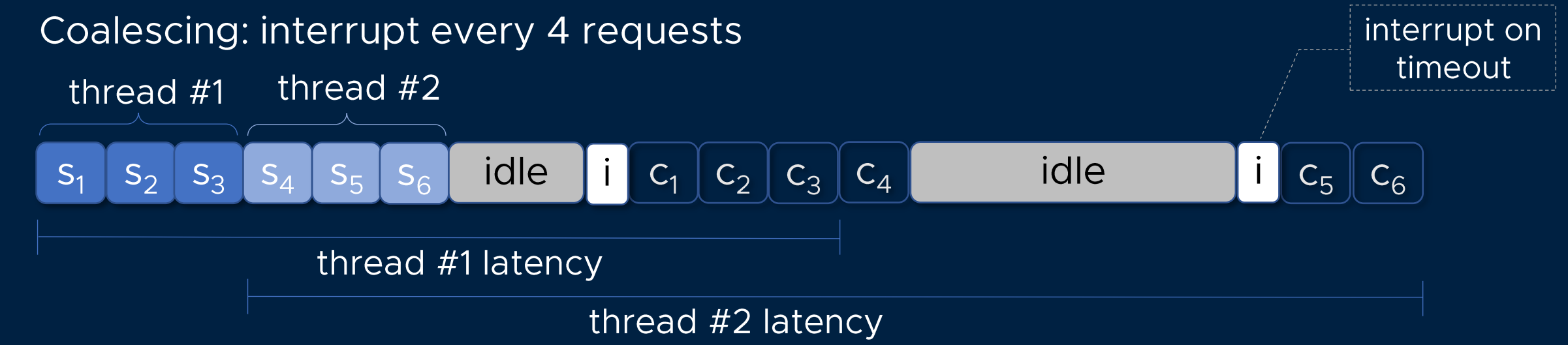
Semantics: Barrier flag means end of a batch

Two threads, each submits batch of 3 requests

Cinterrupts: mark with Barrier the last request in the batch

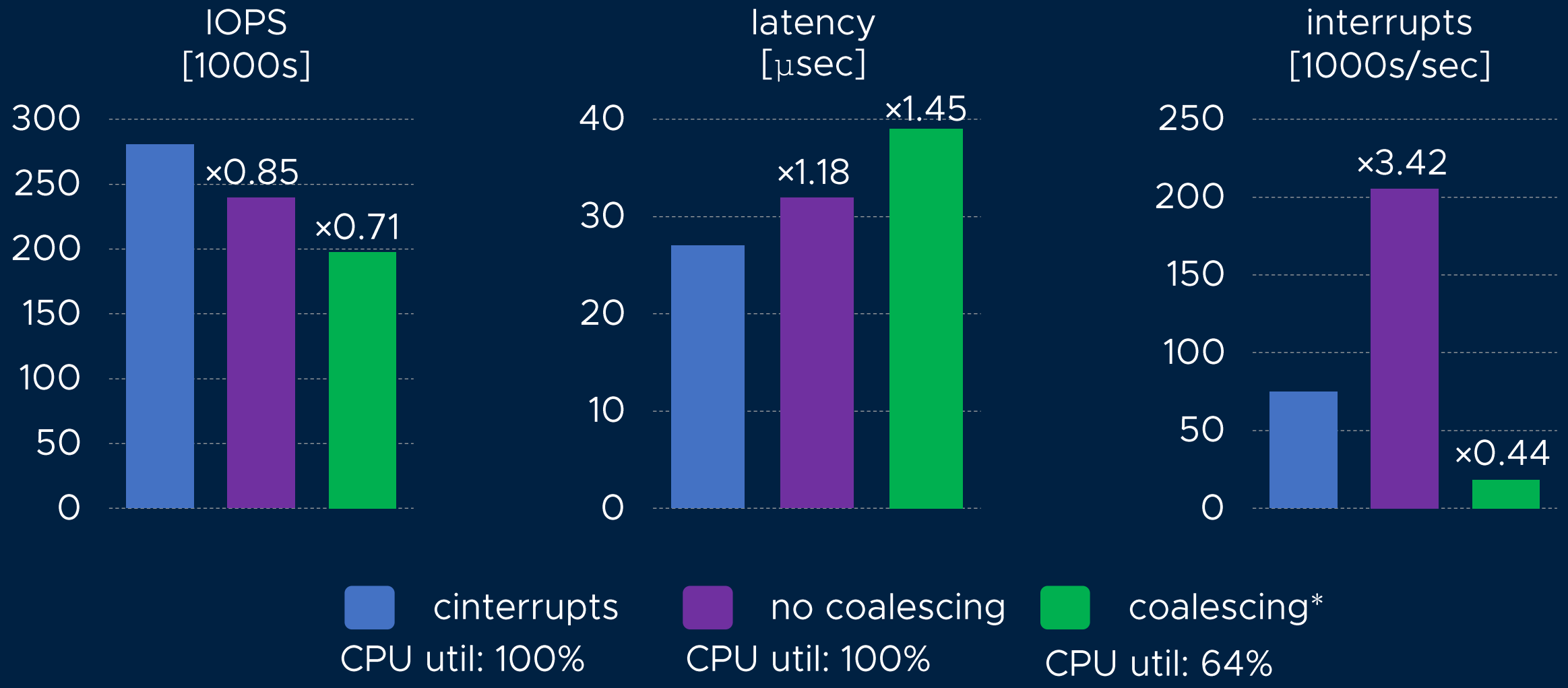


Coalescing: interrupt every 4 requests



Performance benefits of Barrier

Async workload, two threads submit in batches



* Cinterrupts adaptive coalescing, superior to NVMe coalescing

Barrier optimizes throughput.

Generates interrupt exactly when batch is ready.

Implementation: passing flags with Linux kernel support

- RWF_URGENT and RWF_BARRIER syscall flags
- Applications use explicitly in
  - preadv2/pwritev2
  - io_submit
- For legacy apps, kernel sets default annotations
  - mark sync or blocking with Urgent
  - mark async with Barrier

More in the paper

- More macro & micro benchmarks
- Cinterrupts adaptive coalescing
- Emulation details
- Cinterrupts for networking
- and more…

Optimizing Storage Performance with Calibrated Interrupts

Amy Tai, Igor Smolyar, Michael Wei, Dan Tsafrir

Technion – Israel Institute of Technology

VMware Research

Abstract

1 Introduction

Conclusions

Lack of semantics when to generate an interrupt is a problem for high-speed I/O devices

Cinterrupts closes this Semantic Gap for NVMe devices with Urgent and Barrier flags

Cinterrupts improves workloads performance even in a dynamic environment

Thank You!

Have questions? Send us an email.

Amy Tai: taiam@vmware.com

Igor Smolyar: igors@cs.technion.ac.il

Michael Wei: mwei@vmware.com

Dan Tsafrir: dan@cs.technion.ac.il