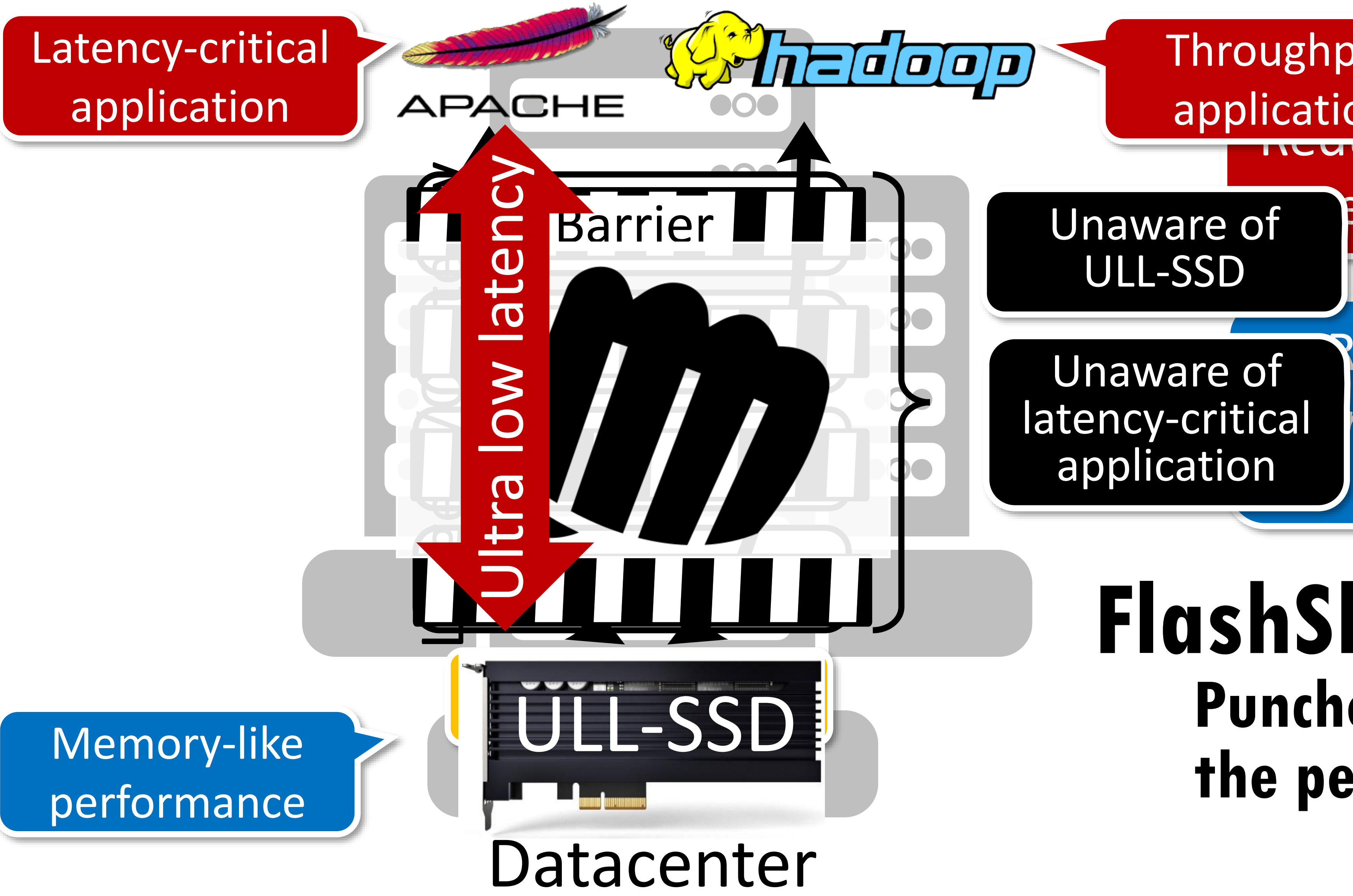
FlashShare: Punching Through Server Storage Stack from Kernel to Firmware for Ultra-Low Latency SSDs

Jie Zhang, Miryeong Kwon, Donghyun Gouk, Sungjoon Koh, Changlim Lee, Mohammad Alian, Myoungjun Chun, Mahmut Kandemir, Nam Sung Kim, Jihong Kim and Myoungsoo Jung

CAMELab.org
Computer Architecture and Memory Systems Laboratory

PennState
ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Executable Summary



Reduce average *turnaround* esponse times by 22%

Reduce 99th-percentile *naround* response times by 31%

FlashShare

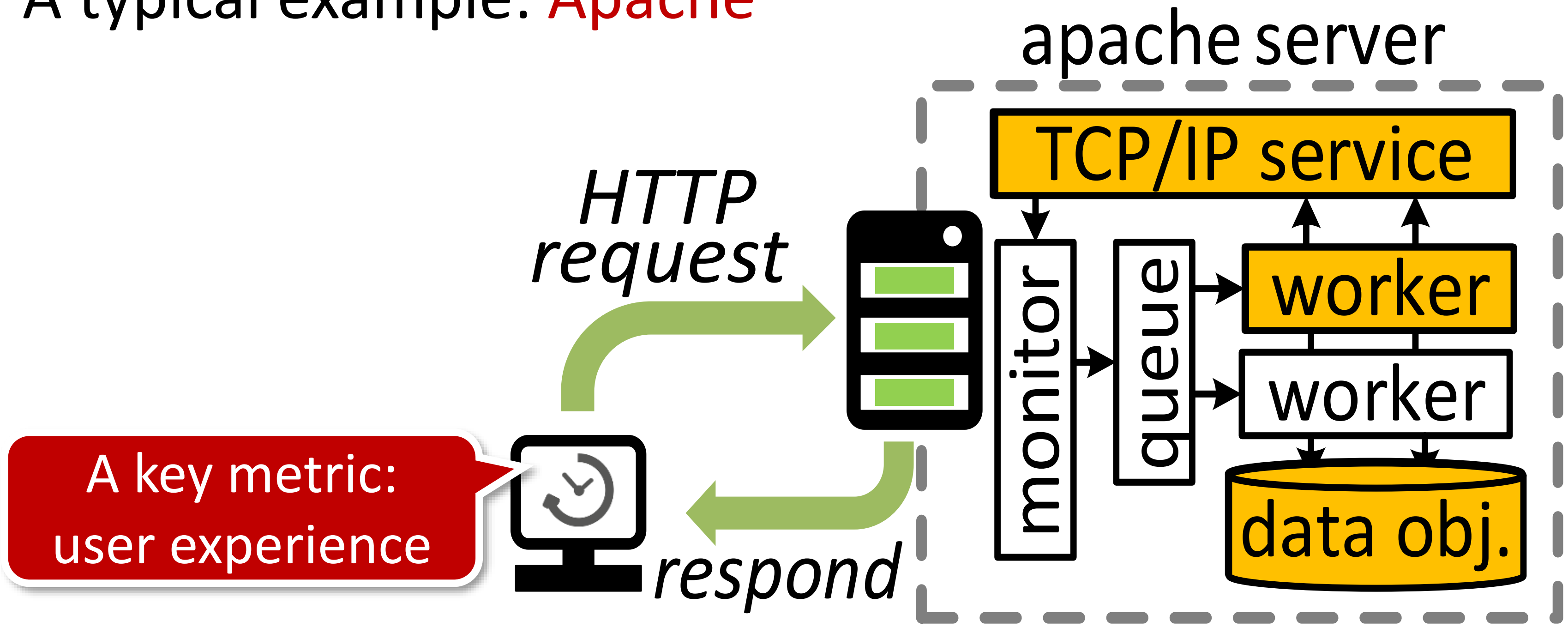
Punches through the performance barriers

Motivation: applications in datacenter

Datacenter executes a wide range of latency-critical workloads.

- Driven by the market of social media and web services;
- Required to satisfy a certain level of service-level agreement;
- Sensitive to the latency (i.e., turn-around response time);

A typical example: Apache



Motivation: applications in datacenter

- Latency-critical applications exhibit varying loads during a day.
- Datacenter overprovisions its server resources to meet the SLA.
- However, it results in a low utilization and low energy efficiency.

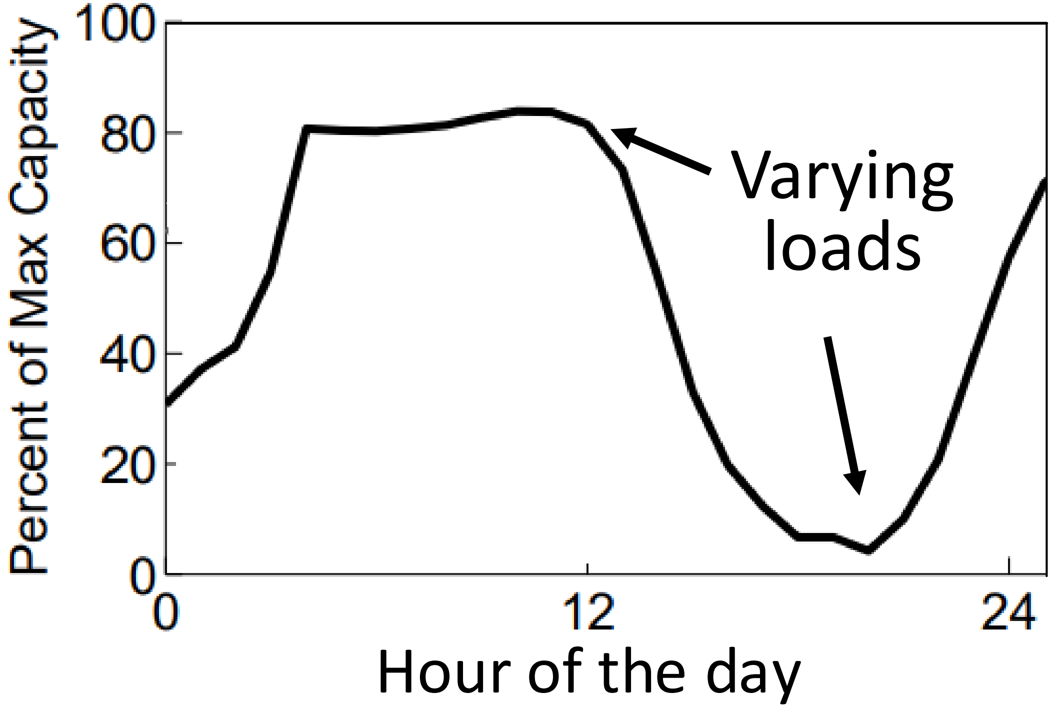


Figure 1. Example diurnal pattern in queries per second for a Web Search cluster[1].

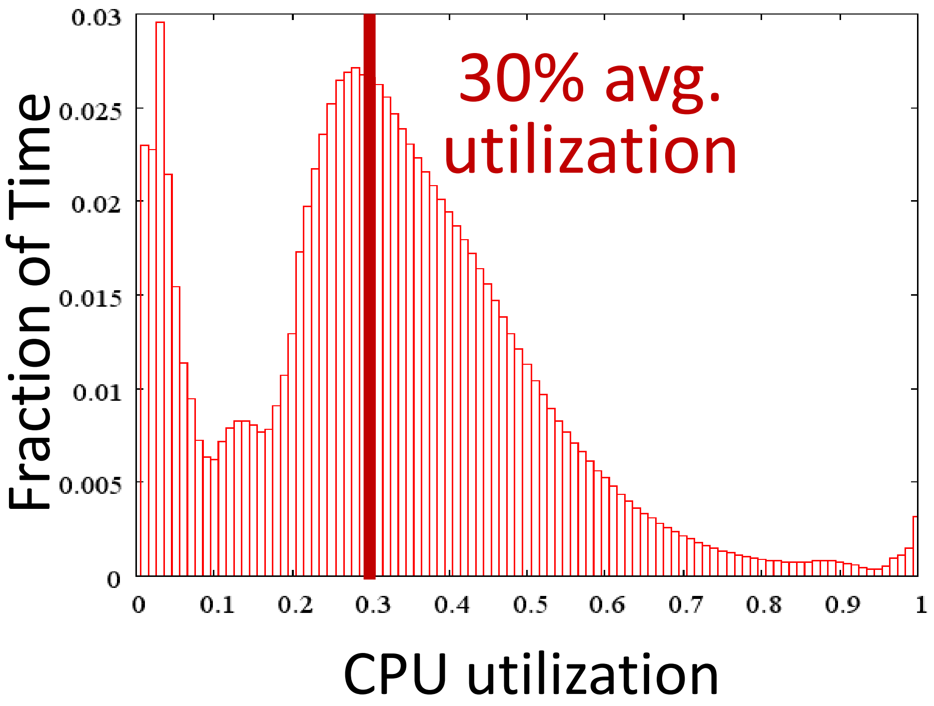


Figure 2. CPU utilization analysis of Google server cluster[2].

[1]Power Management of Online Data-Intensive Services.

[2]The Datacenter as a Computer.

Motivation: applications in datacenter

Popular solution: co-locating latency-critical and throughput workloads.

Micro'11

Bubble-Up: Increasing Utilization in Modern Warehouse Scale Computers via Sensible Co-locations

Jason Mars
University of Virginia
jom5x@cs.virginia.edu

Lingjia Tang
University of Virginia
lt8f@cs.virginia.edu

Robert Hundt
Google
rhundt@google.com

Kevin Skadron
University of Virginia
skadron@cs.virginia.edu

Mary Lou Soffa
University of Virginia
soffa@cs.virginia.edu

ABSTRACT

As much of the world's computing continues to move into the cloud, the overprovisioning of computing resources to ensure the performance isolation of *latency-sensitive* tasks, such as web search, in modern datacenters is a major contributor to low machine utilization. Being unable to accu-

ISCA'15

Heracles: Improving Resource Efficiency at Scale

David Lo[†], Liqun Cheng[‡], Rama Govindaraju[‡], Parthasarathy Ranganathan[‡] and Christos Kozyrakis[†]
Stanford University[†] Google, Inc.[‡]

Abstract

User-facing, latency-sensitive services, such as websearch, underutilize their computing resources during daily periods of low traffic. Reusing those resources for other tasks is rarely done in production services since the contention for shared resources can cause latency spikes that violate the service-level objectives

But, to amortize the much larger capital expenses, an increased emphasis on the *effective use of server resources* is warranted.

Several studies have established that the average server utilization in most datacenters is low, ranging between 10% and 50% [14, 74, 66, 7, 19, 13]. A primary reason for the low utilization is the popularity of latency-critical (LC) services such as social media, search engines, software-as-a-service, online maps, webmail, machine translation, online shopping and advertising. These user-facing services are typically scaled across thousands of servers and access distributed state stored in memory or Flash across these servers. While their load varies significantly due to diurnal patterns and unpredictable spikes in user accesses, it is

Eurosys'14

Reconciling High Server Utilization and Sub-millisecond Quality-of-Service

Jacob Leverich Christos Kozyrakis
Computer Science Department, Stanford University
{leverich,christos}@cs.stanford.edu

Abstract

The simplest strategy to guarantee good quality of service (QoS) for a latency-sensitive workload with sub-millisecond latency in a shared cluster environment is to never run other workloads concurrently with it on the same server. Unfortunately, this inevitably leads to low server utilization, reducing both the capability and cost effectiveness of the cluster.

provements. In the past, we could also rely on deploying new servers with processors offering higher performance at the same power consumption; in effect, by replacing servers year after year, we could achieve greater compute capability without having to invest in new power or cooling infrastructure. However, the end of voltage scaling has resulted in a significant slowdown in processor performance scaling [9]. A datacenter operator can still increase capability by build-

CAMEL

Challenge: applications in datacenter

Experiment: *Apache+PageRank* vs. *Apache only*

Server configuration:

Components	Spec.	Components	Spec.
CPU	i7-4790	Memory	32GB
	3.6GHz		DDR3
	8 cores	Chipset	H97

Applications:

Apache – Online latency-critical application;

PageRank – Offline throughput application;

Performance metrics:

SSD device latency;

Response time of latency-critical application;

Challenge: applications in datacenter

Experiment: *Apache+PageRank* vs. *Apache only*

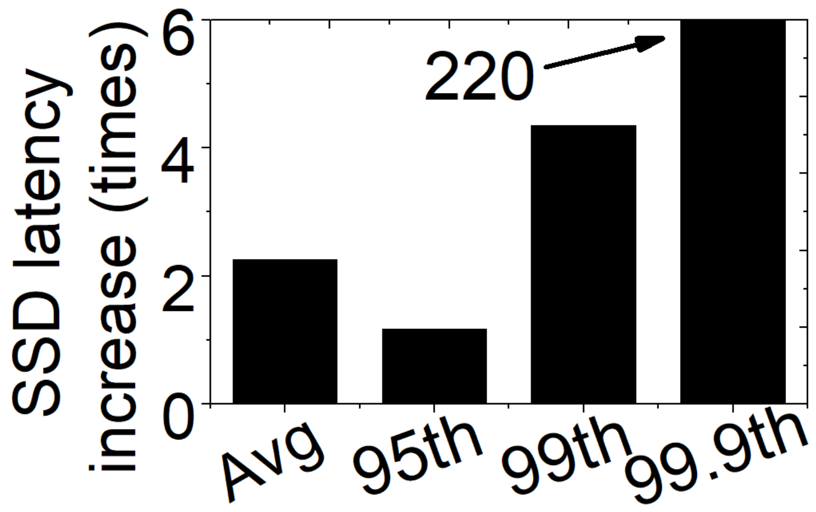


Fig 1: Apache SSD latency increases due to PageRank.

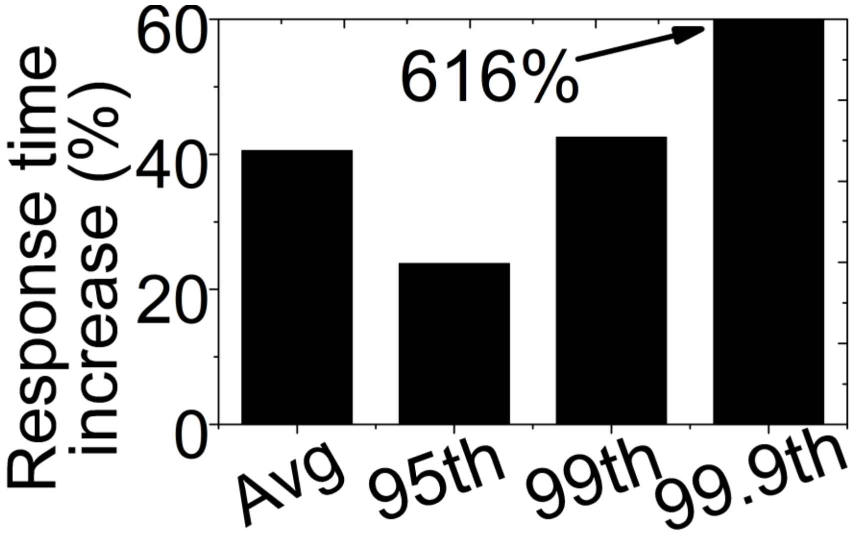


Fig 2: Apache response time increases due to PageRank.

- The throughput-oriented application drastically increases the I/O access latency of the latency-critical application.
- This latency increase deteriorates the turnaround response time of the latency-critical application.

Challenge: ULL-SSD

There are emerging **U**ltra **L**ow-**L**atency SSD (ULL-SSD) technologies, which can be used for faster I/O services in the datacenter.

	Optane	nvNitro	ZNAND	XL-Flash
Technique	Phase change RAM	MRAM	New NAND Flash	
Vendor	Intel	Everspin	Samsung	Toshiba
Read	10us	6us	3us	N/A
Write	10us	6us	100us	N/A

Challenge: ULL-SSD

In this work, we use engineering sample of Z-SSD.

Z-NAND[1]	
Technology	SLC based 3D NAND 48 stacked word-line layer
Capacity	64Gb/die
Page Size	2KB/Page

Z-NAND based archives ***"Z-SSD"***

[1] Cheong, Wooseong, et al. "A flash memory controller for 15μs ultra-low-latency SSD using high-speed 3D NAND flash with 3μs read time." *2018 IEEE International Solid-State Circuits Conference-(ISSCC)*, 2018.

Challenge: Datacenter server with ULL-SSD

Unfortunately, the short latency characteristics of ULL-SSD cannot be exposed to users (in particular, for the latency-critical applications).

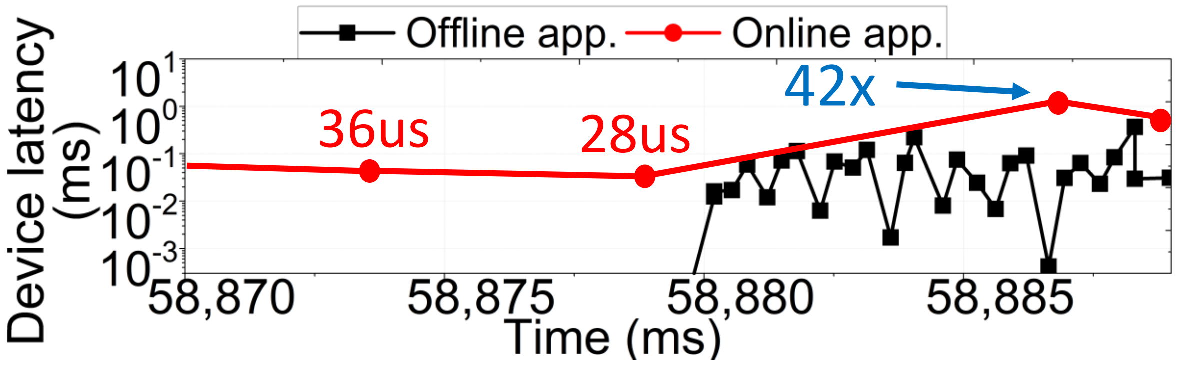
Server configuration:

Components	Spec.	Components	Spec.
CPU	i7-4790	Memory	32GB
	3.6GHz		DDR3
	8 cores	Chipset	H97

Applications:

Apache – online latency-critical application;

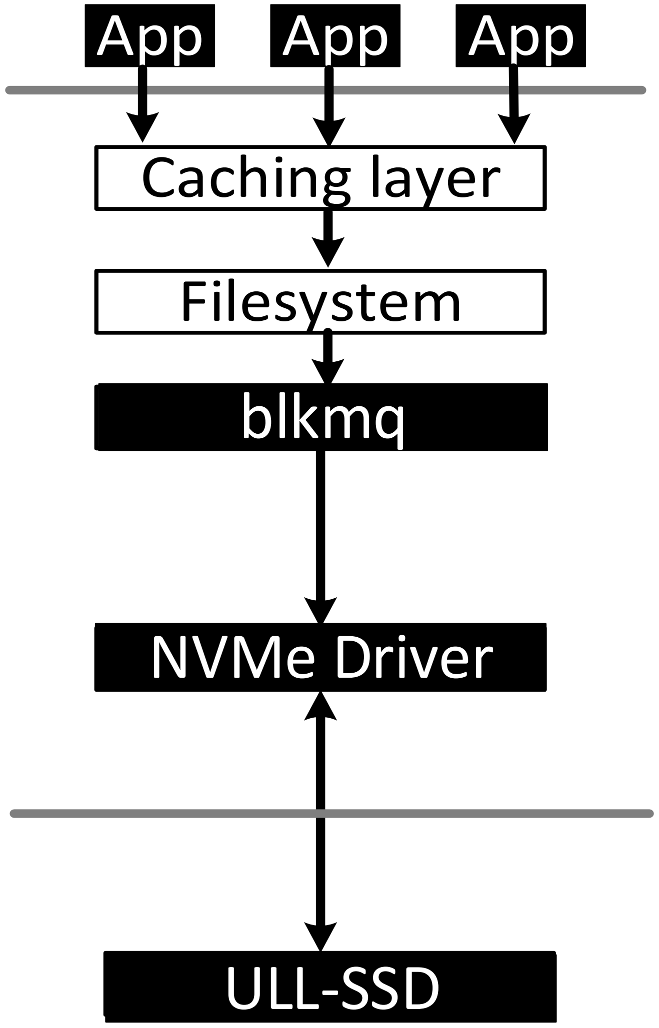
PageRank – offline throughput application;



Device latency analysis

Challenge: Datacenter server with ULL-SSD

ULL-SSD fails to bring short latency, because of the storage stack.

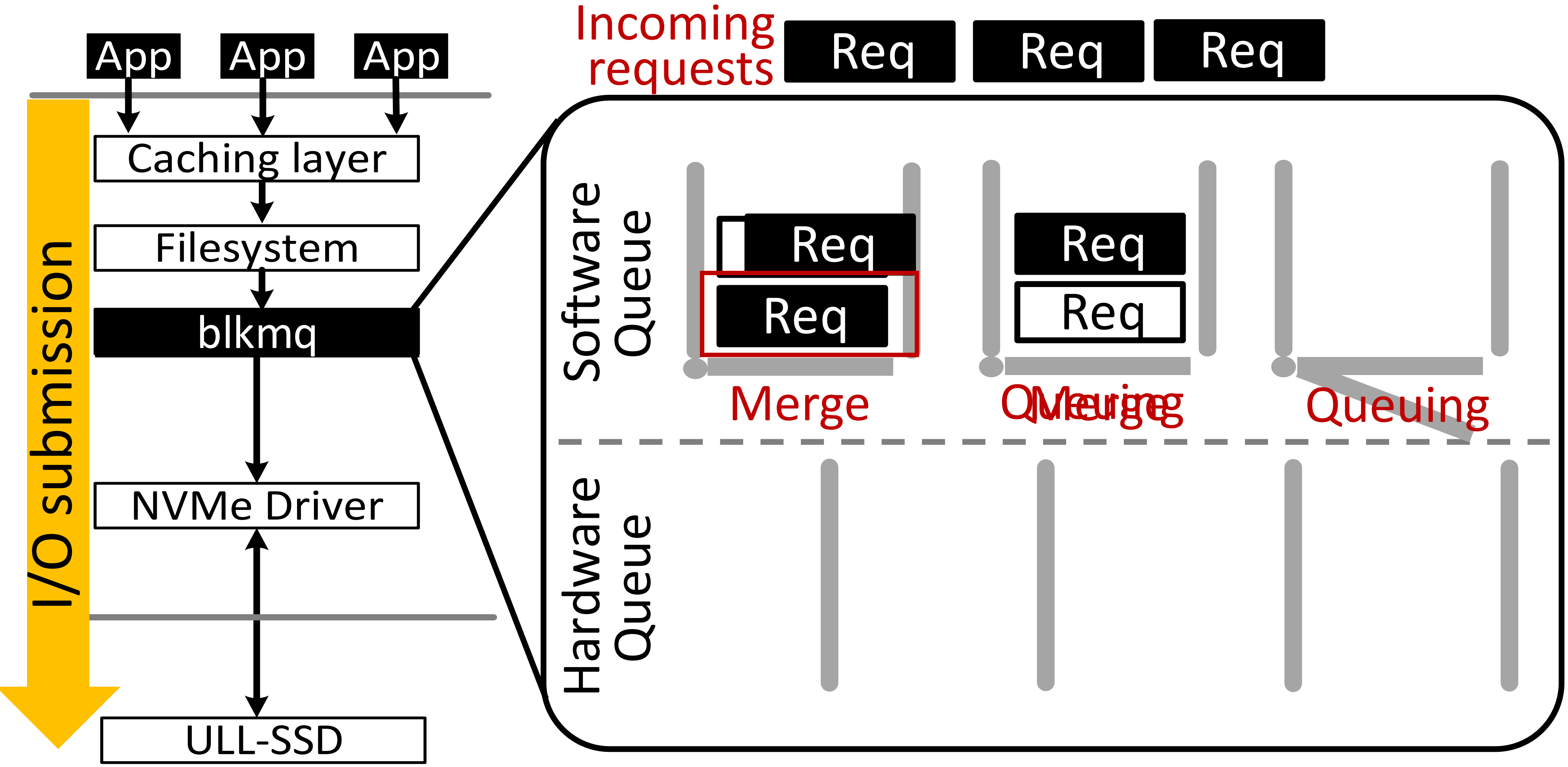


The storage stack is unaware of the characteristics of both latency-critical workload and ULL-SSD

The current design of blkmq layer, NVMe driver, and SSD firmware can hurt the performance of latency-critical applications.

Blkmq layer: challenge

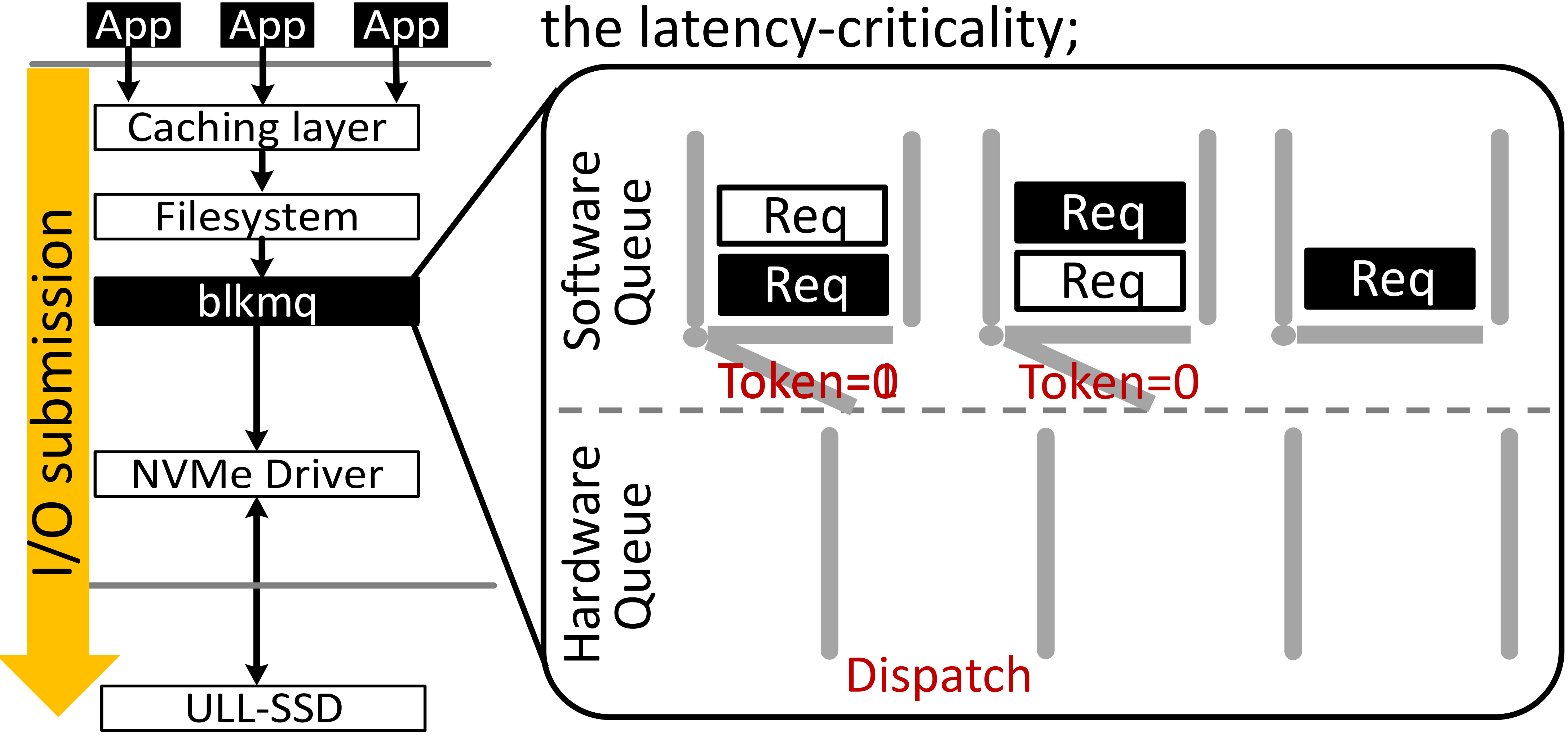
Software queue: holds latency-critical I/O requests for a long time;



Blkmq layer: challenge

Software queue: holds latency-critical I/O requests for a long time;

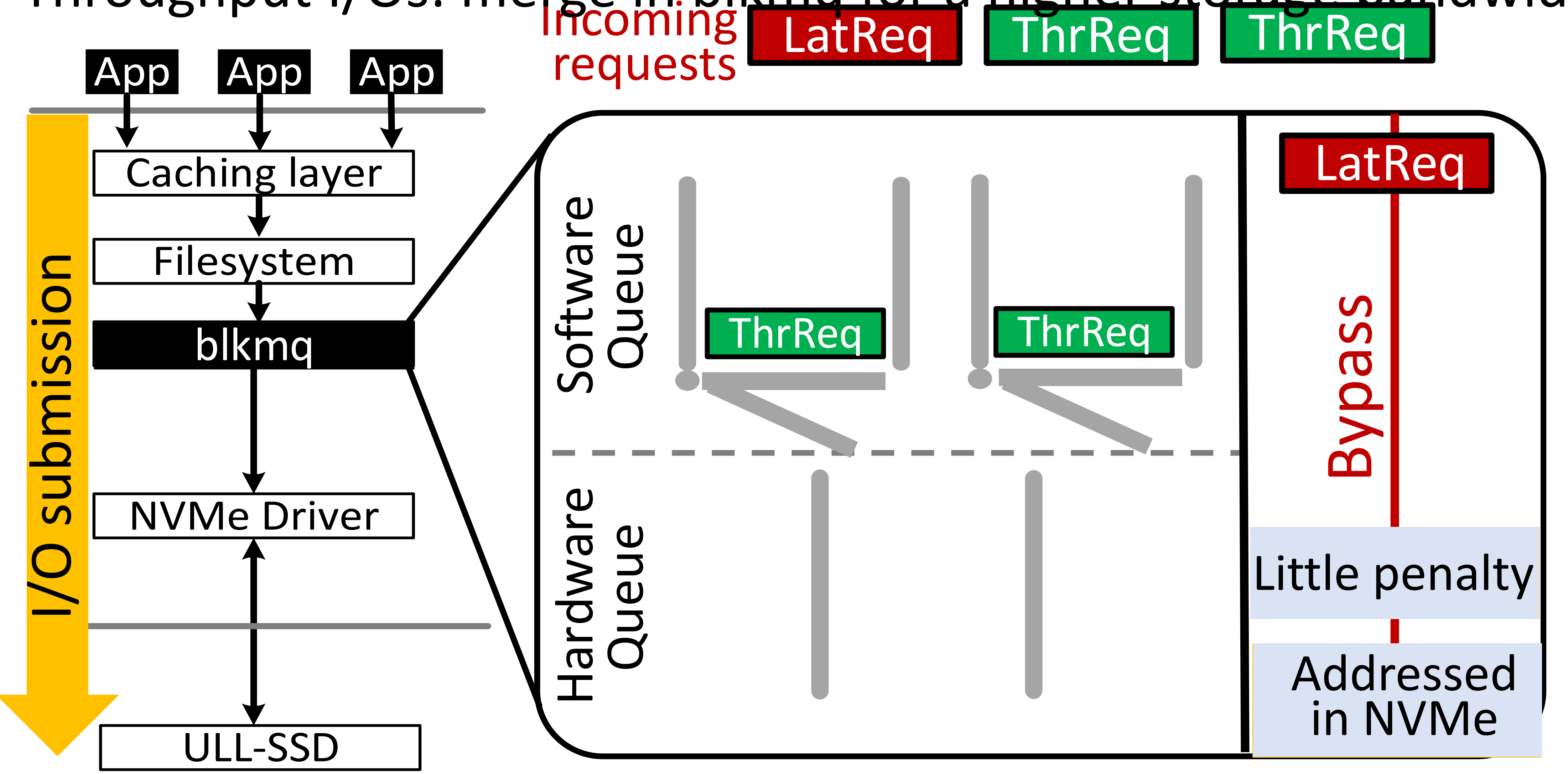
Hardware queue: dispatches an I/O request without a knowledge of the latency-criticality;



Blkmq layer: optimization

Our solution: bypass.

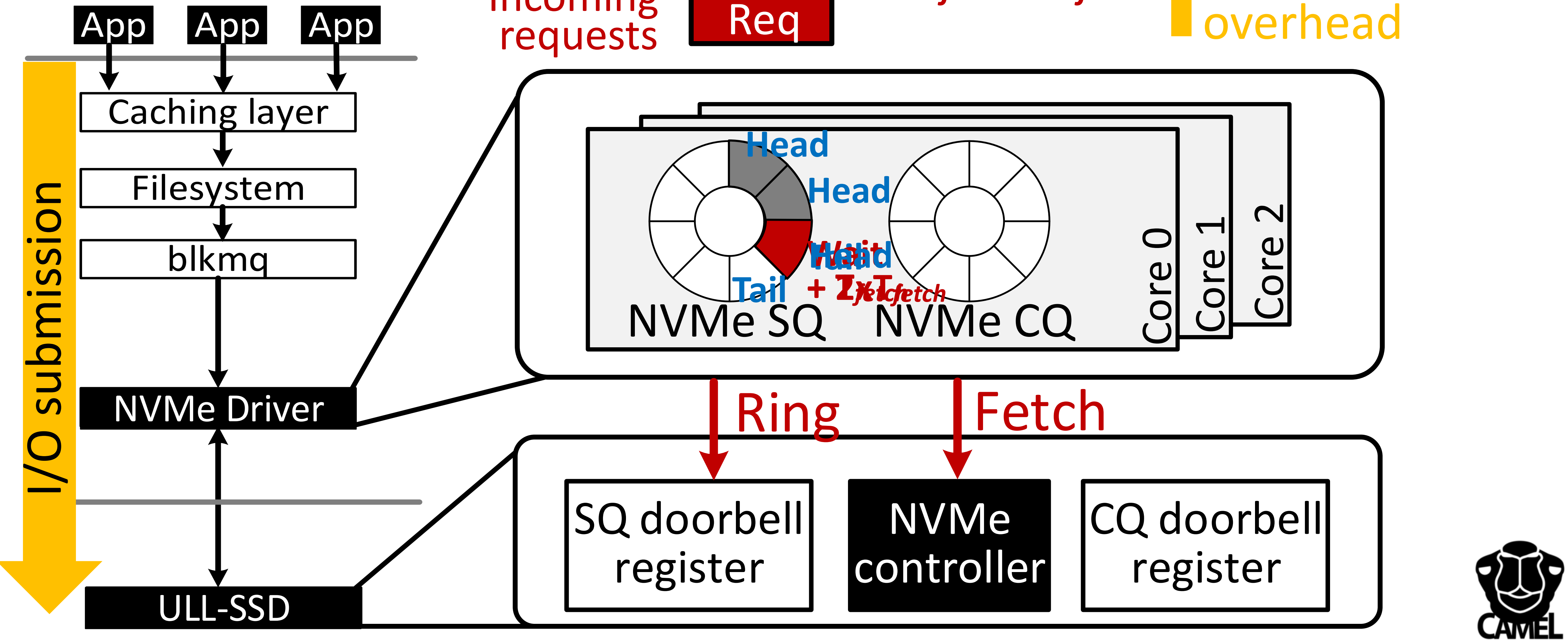
Latency-critical I/Os: bypass blkmq for a faster response;
Throughput I/Os: merge in blkmq for a higher storage bandwidth.



NVMe SQ: challenge (bypass is not simple enough)

NVMe protocol-level queue: a latency-critical I/O request can be blocked by prior I/O requests;

Time Cost = $T_{fetch\text{-}self}$ + 2xT_{fetch} ↑ >200% overhead



NVMe SQ: optimization

Target: Designing towards a responsiveness-aware NVMe submission.

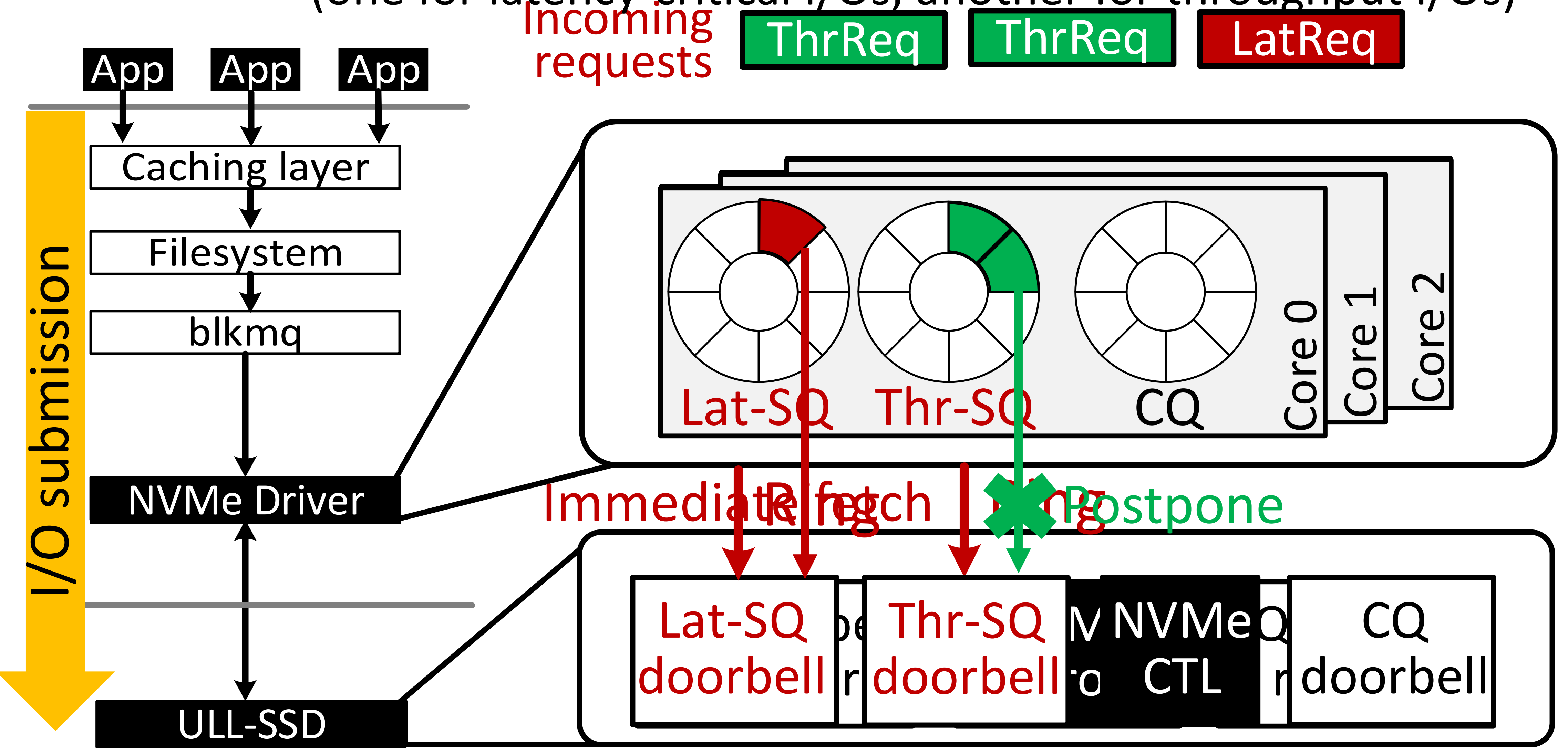
Key Insight:

- Conventional NVMe controller(s) allow to customize the standard arbitration strategy for different NVMe protocol-level queue accesses.
- Thus, we can make the NVMe controller to decide which NVMe command to fetch by sharing a hint for the I/O urgency.

NVMe SQ: optimization

Our Solution: 1. Double SQ doorbell register (one for latency-critical I/Os, another for throughput I/Os)

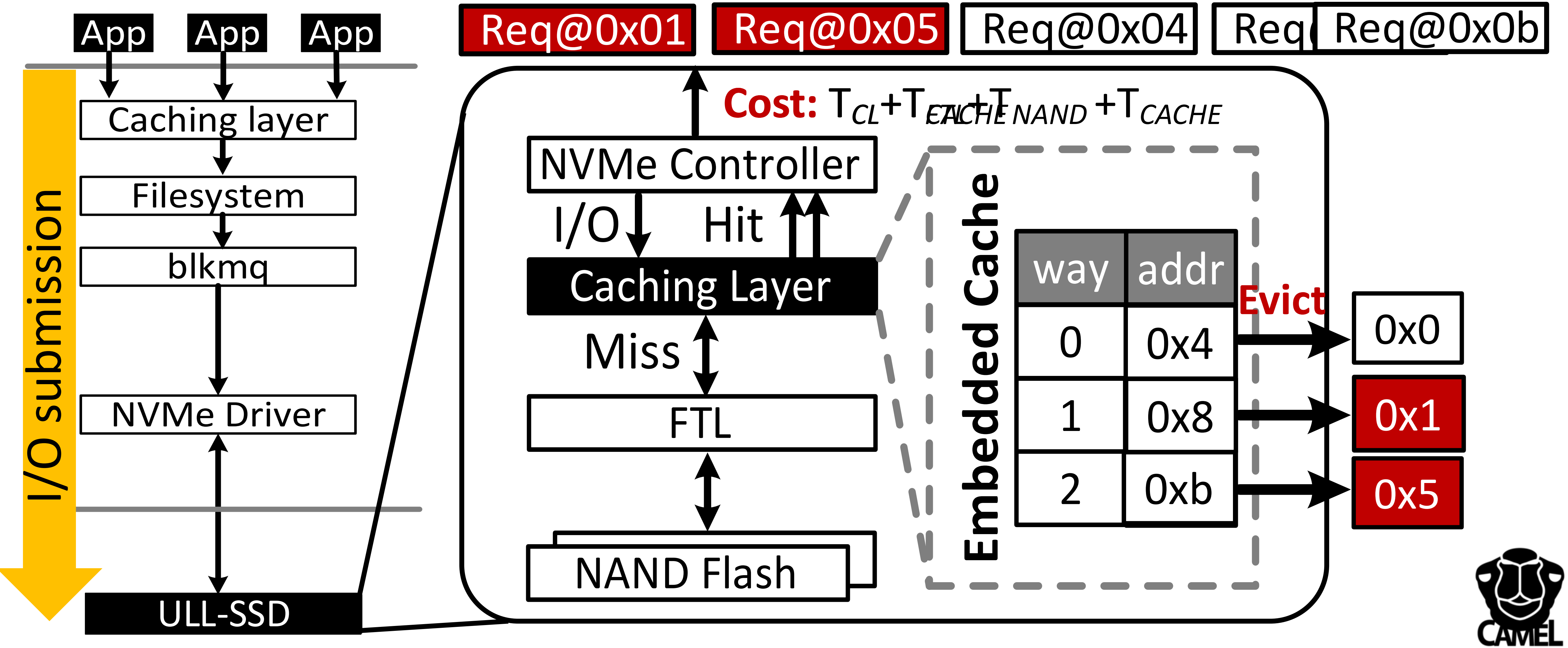
2. New arbitration strategy: gives the highest priority to the Lat-SQ



SSD firmware: challenge

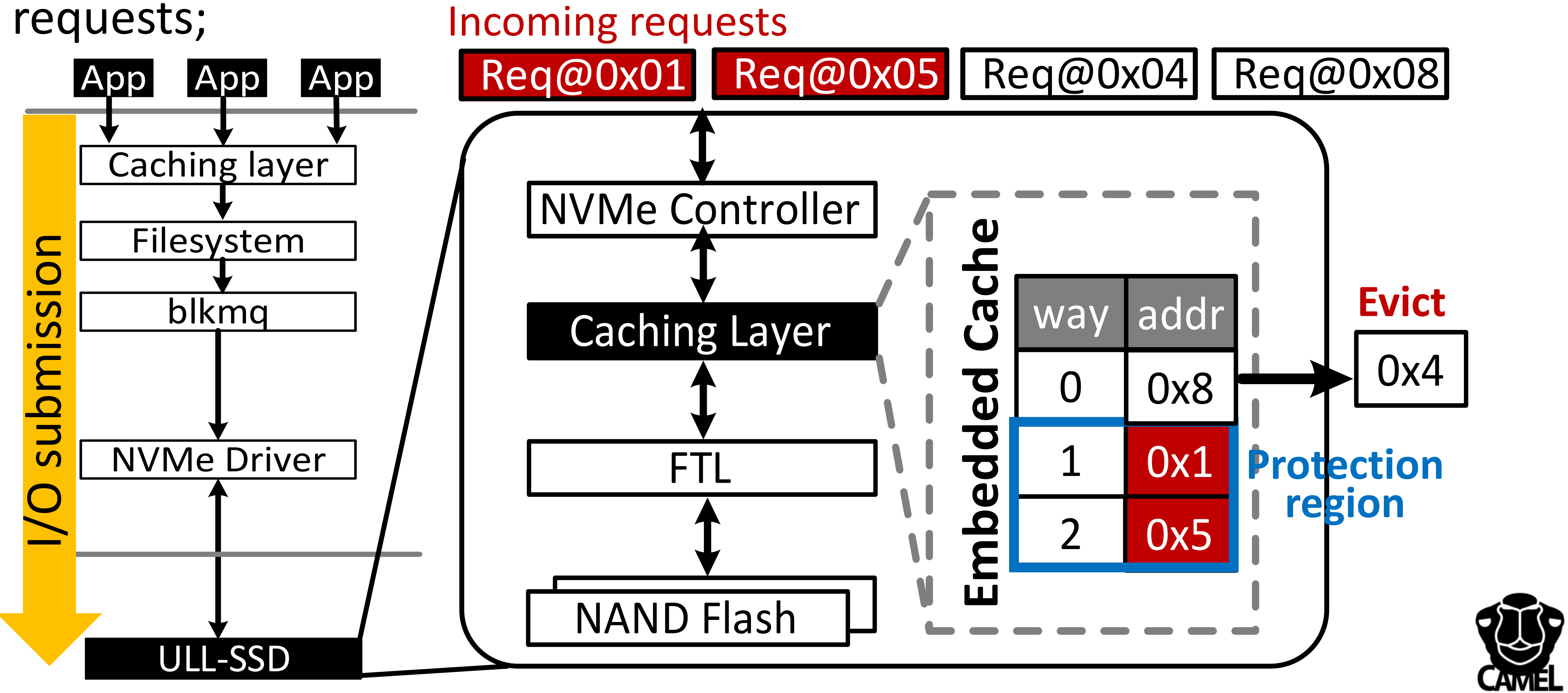
Embedded cache provides the fastest response (DRAM service)

Embedded cache can be polluted by the throughput I/O requests; from an eviction;



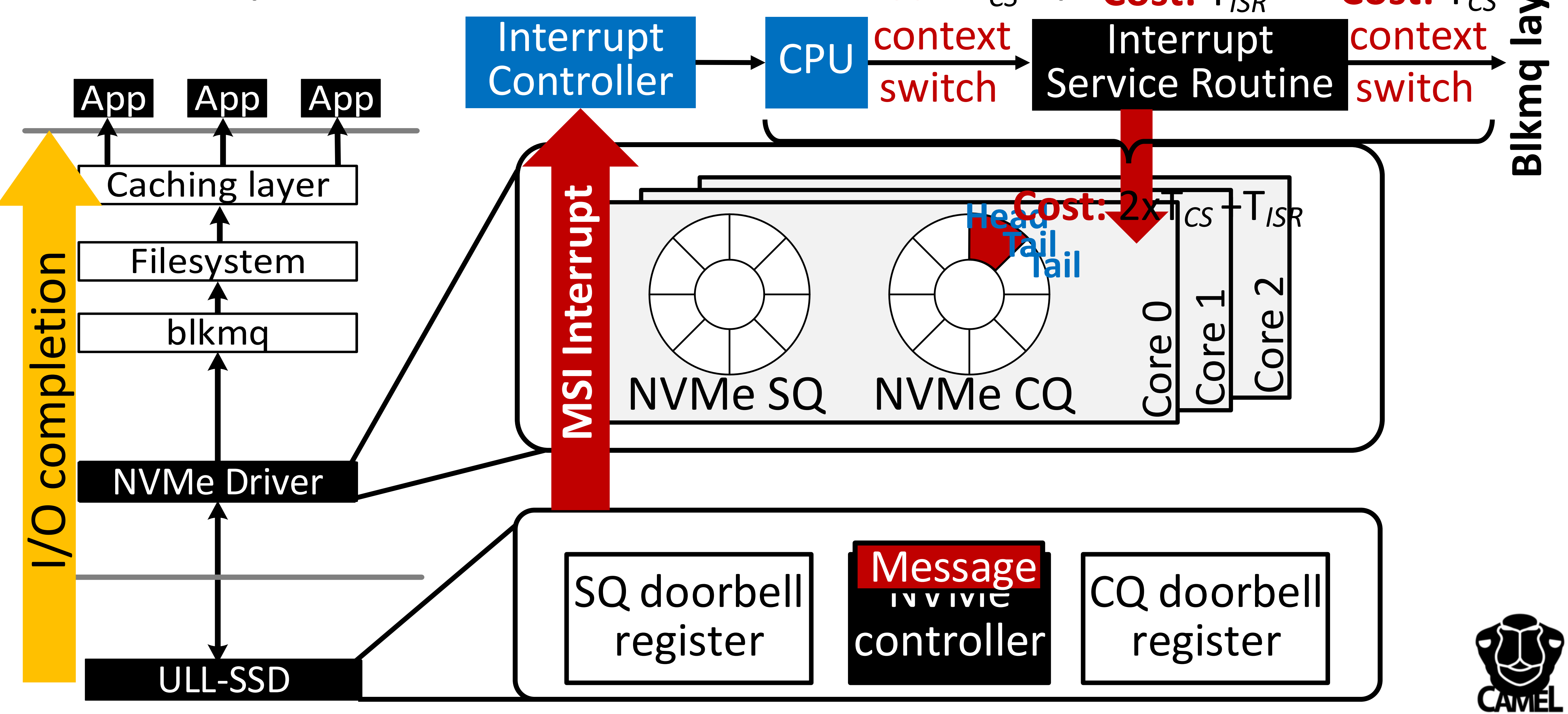
SSD firmware: optimization

Our design: splits the internal cache space to protect latency-critical I/O requests;



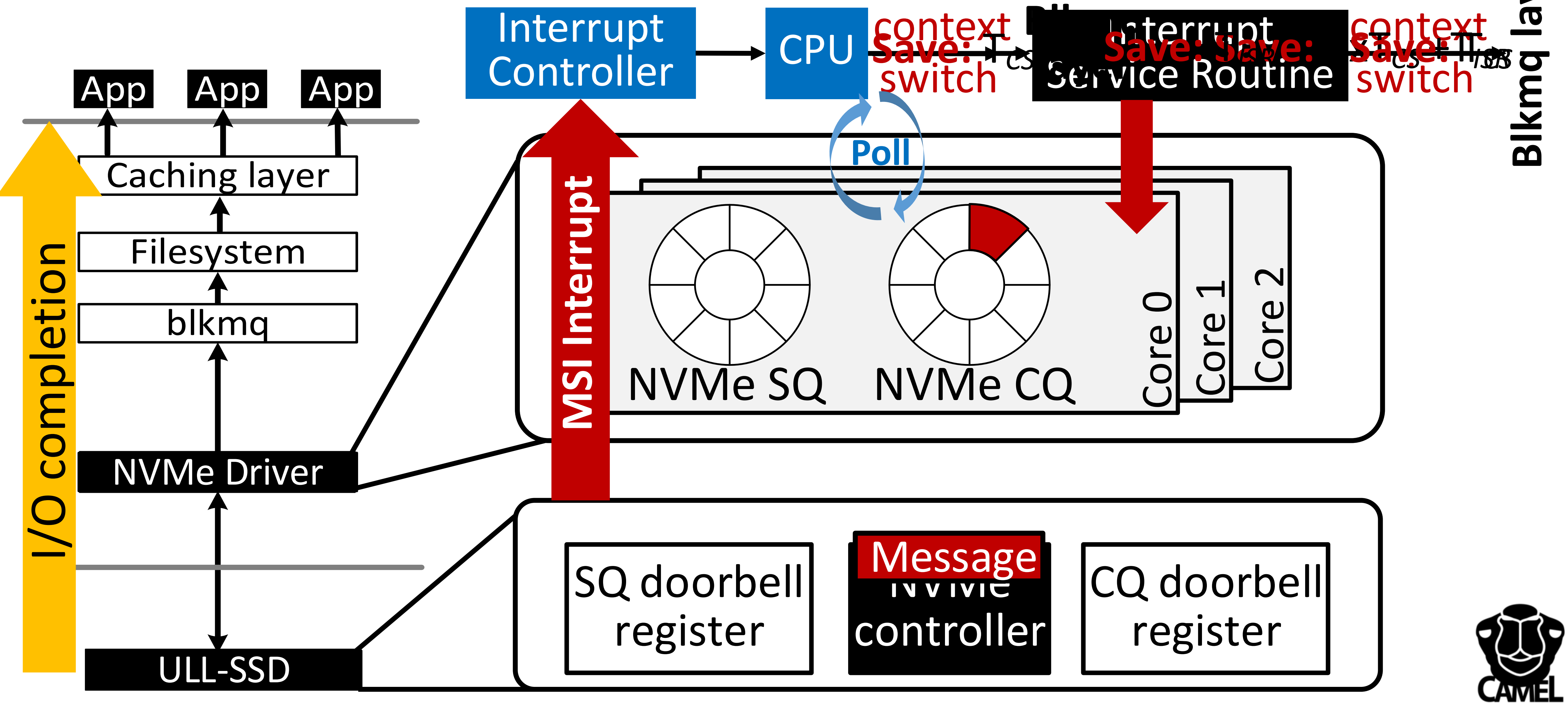
NVMe CQ: challenge

NVMe completion: MSI overhead for each I/O request:



NVMe CQ: optimization

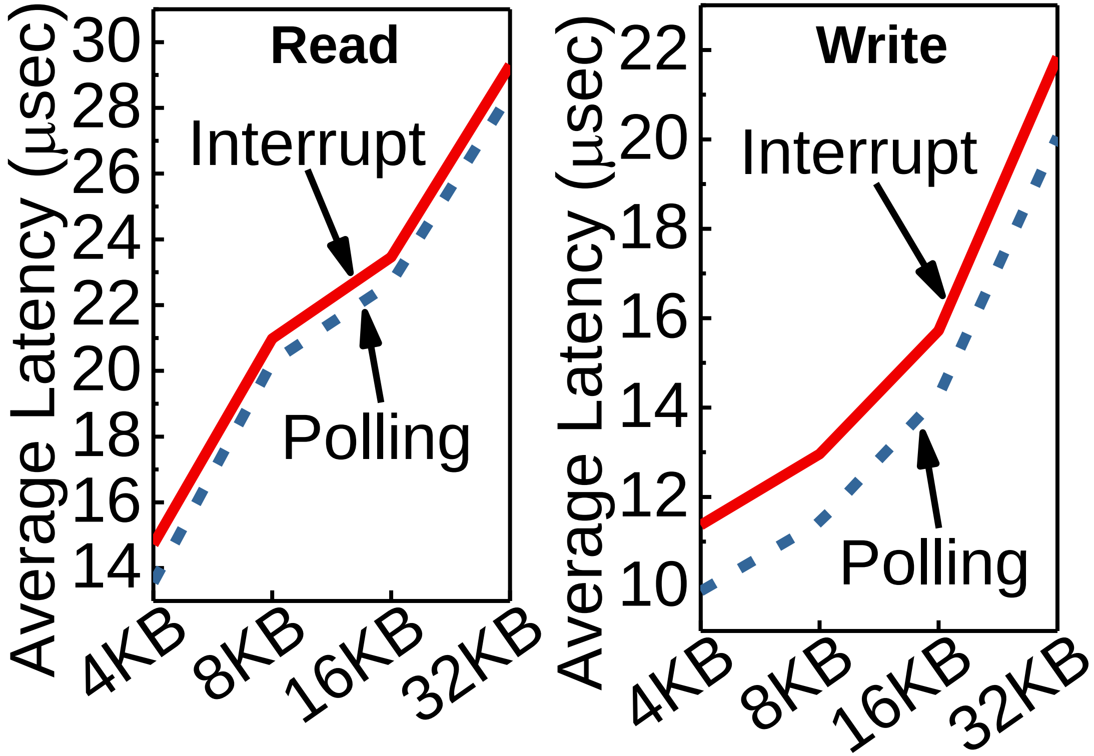
Key insight: state-of-the-art Linux supports a poll mechanism;



NVMe CQ: optimization

Poll mechanism can bring benefits to fast storage device.

ULL SSD

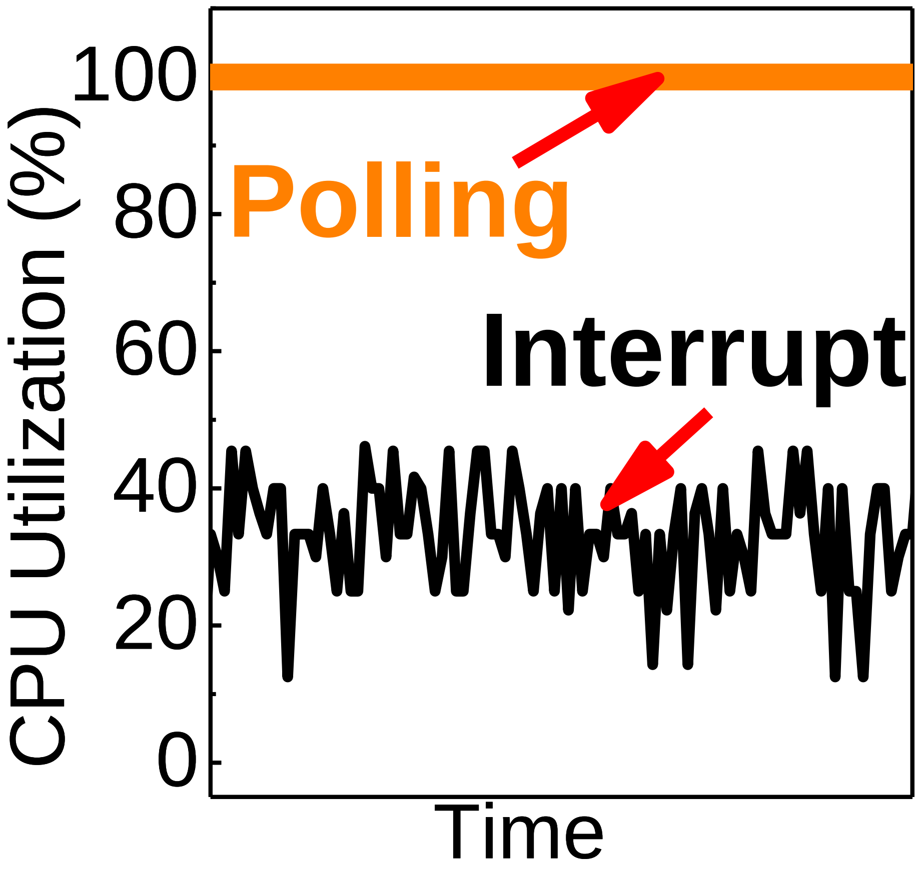


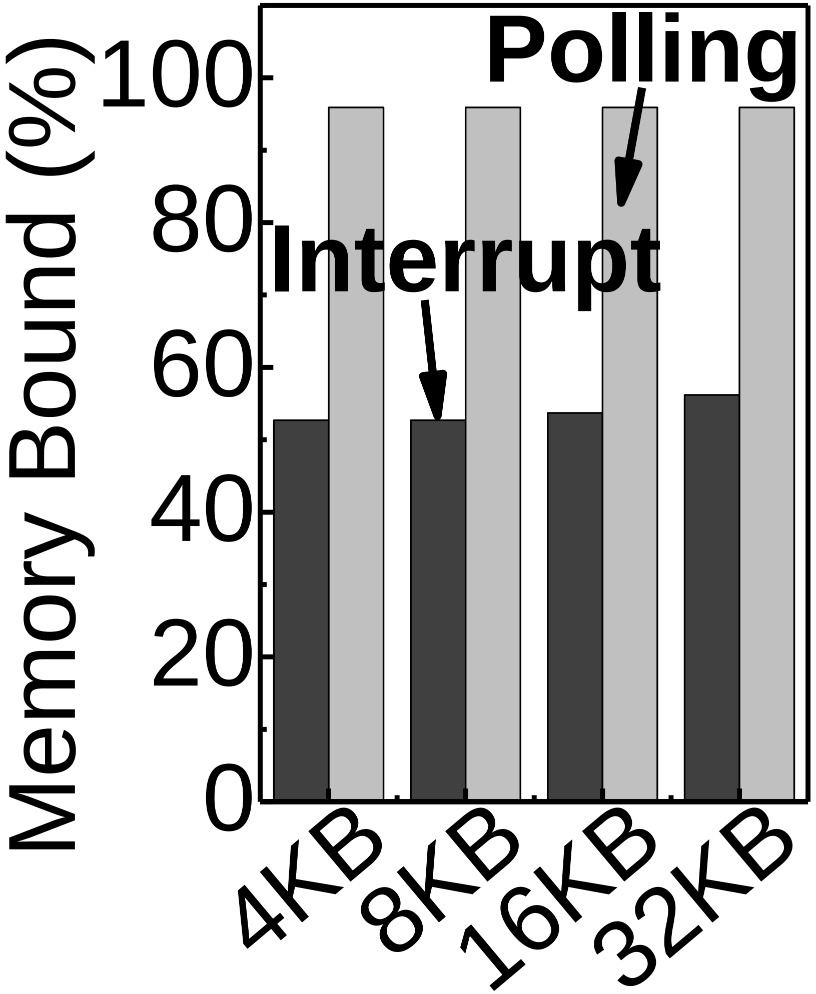
Decreases by

Read: ***7.5%*** & Write: ***13.2%***

NVMe CQ: optimization

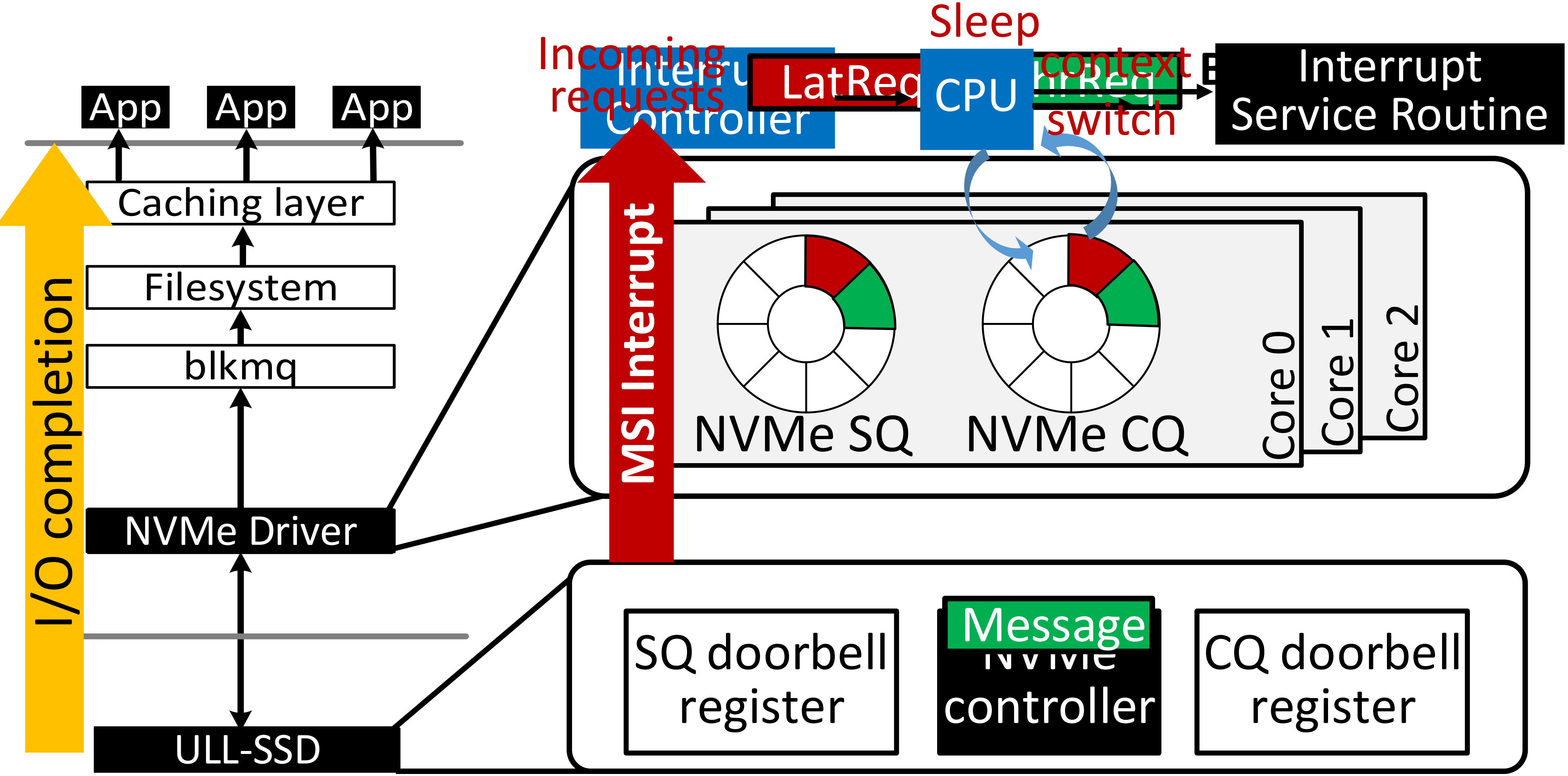
However, the poll-based I/O services consume most host resources.





NVMe CQ: optimization

<u>**Our solution**</u>: selective interrupt service routine (***Select-ISR***).



Design: Responsiveness Awareness

Key Insight: users have a better knowledge of I/O responsiveness (i.e., latency critical/throughput).

Our Approach:

- Open a set of APIs to users, which pass the workload attribute to Linux PCB.

chworkload_attr

USAGE: chworkload_attr -t type [-p process_id] [user_program]

DESCRIPTION: set application type

Call a new utility: *chworkload_attr*	Invoke new system call	Modify Linux PCB data structure

```
int sched_setworkload_attr(){
...
return syscall(
   sys_sched_setworkload_attr, pid, attr);
}
int sched_getworkload_attr(){
...
return syscall(
   sys_sched_getworkload_attr, pid, attr);
}
```

```
SYSCALL_DEFINE2(
   sched_setworkload_attr, pid_t, pid,
                           int, workload_attr)

int sched_setworkload_attr
         (struct task_struct* p,
          int workload_attr){
...
      p->workload_attr =
            workload_attr;
}
```

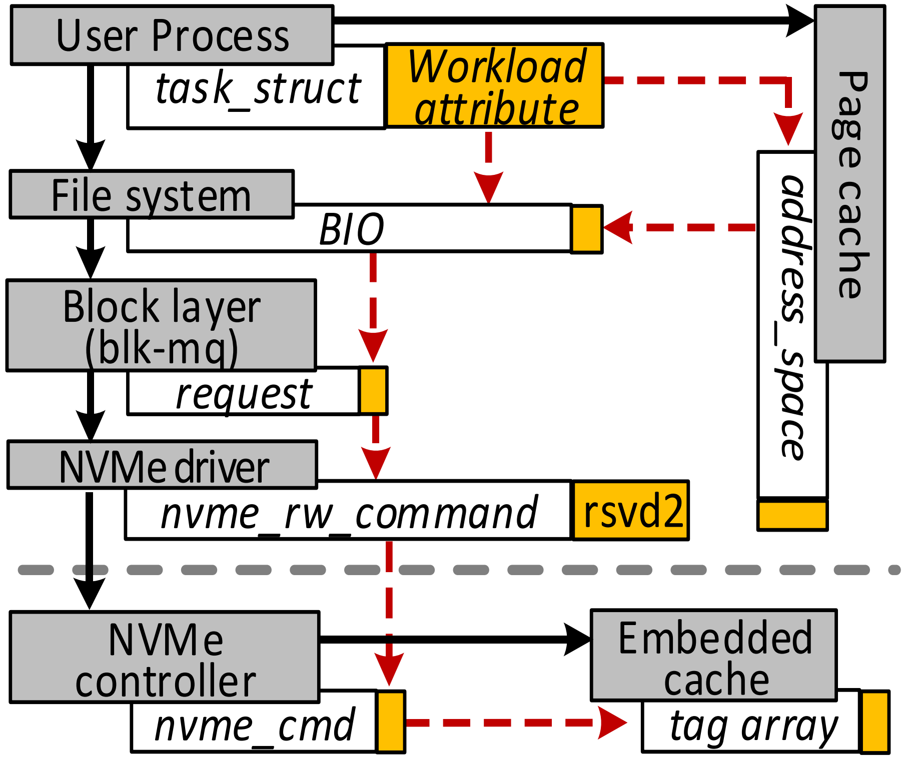
```
struct task_struct {
      ....
   volatile long state;
   unsigned int policy;
   unsigned int
         workload_attr;
      ....
}
```

Design: Responsiveness Awareness

Key Insight: users have a better knowledge of I/O responsiveness (i.e., latency critical/throughput).

Our Approach:

- Open a set of APIs to users, which pass the workload attribute to Linux PCB.
- Deliver the workload attribute to each layer of storage stack.



More optimizations

Advanced caching layer designs:

- Dynamic cache split scheme: to maximize cache hits in various request patterns;
- Read prefetching: better utilize SSD internal parallelism;
- Adjustable read prefetching with ghost cache: adaptive to different request patterns;

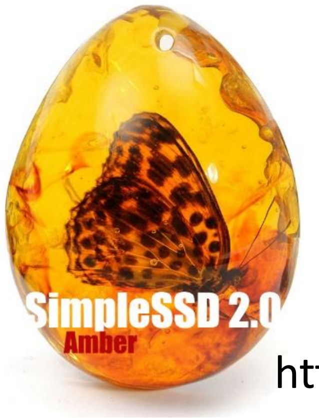
Hardware accelerator designs:

- Conduct simple but timing-consuming tasks such as I/O poll and I/O merge;
- Simplify the design of blkmq and NVMe driver.

Experiment Setup

Test Environment

gem5		SimpleSSD	
parameters	**value**	**parameters**	**values**
core	64-bit ARM, 8, 2GHz	**read/write/erase**	3us/100us/1ms
L1D$/L1I$	64KB, 64KB	**channel/package**	16/1
mem ctrler	1	**die/plane**	8/8
memory	DDR3, 2GB	**page size**	2KB
Kernel	4.9.30	**DMA/PCIe**	800MHz,3.0, x4
Image	Ubuntu 14.04	**DRAM cache**	1.5GB

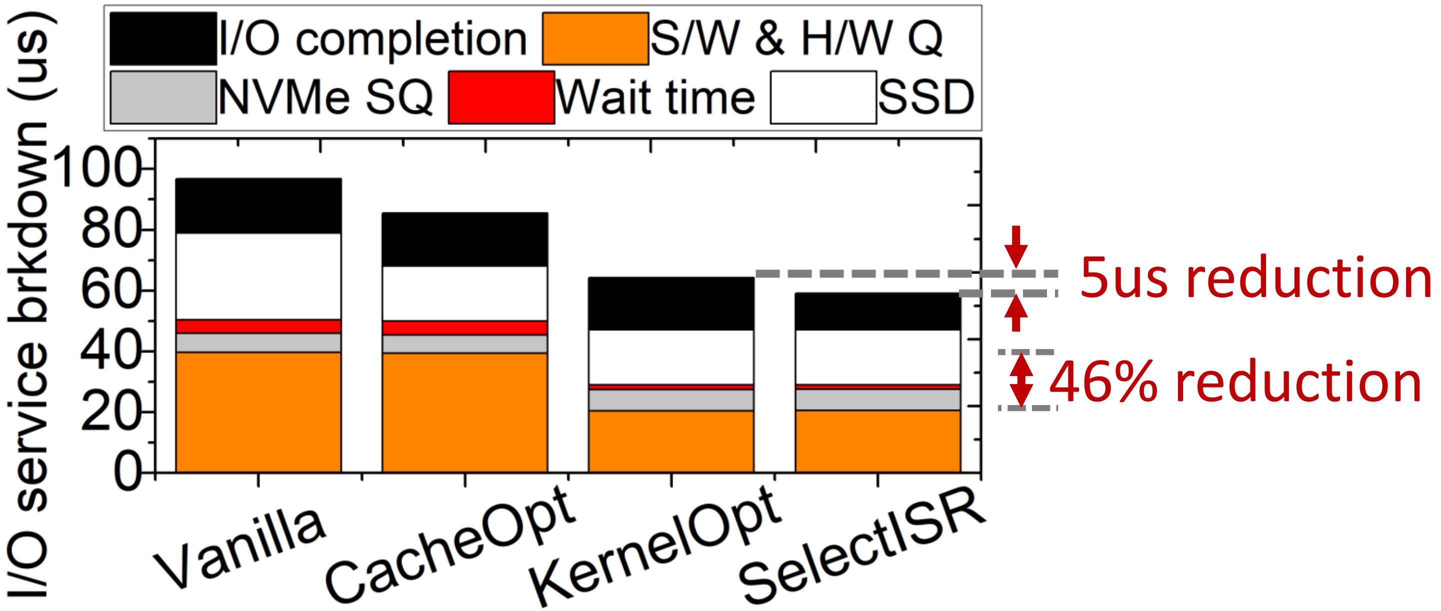


http://simplessd.org

System configurations:

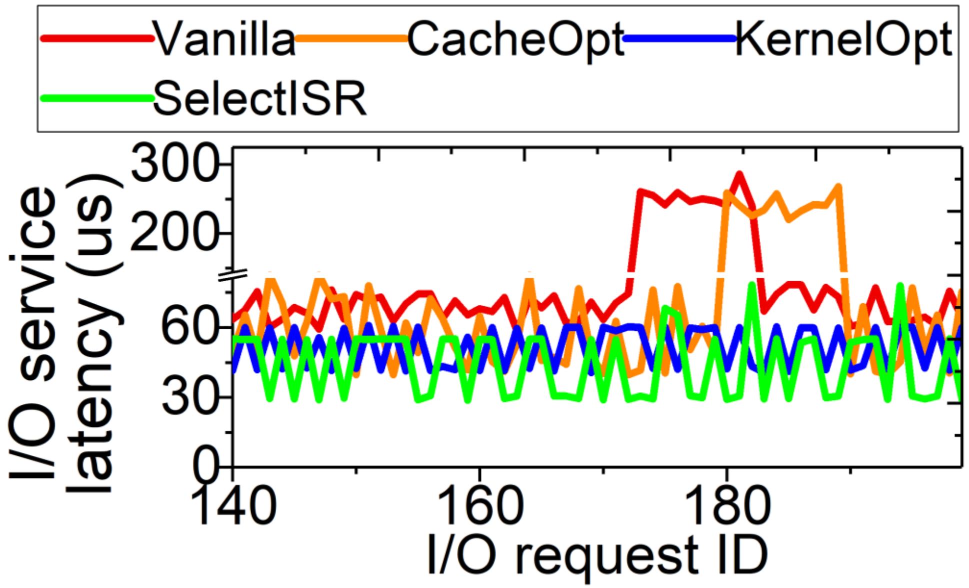
- Vanilla – a vanilla Linux-based computer system running on ZSSD;
- CacheOpt – compared to Vanilla, it optimizes the cache layer of SSD firmware;
- KernelOpt – it optimizes blkmq layer and NVMe I/O submission;
- SelectISR – compared to KernelOpt, it adds the optimization of selective ISR;

Evaluation: latency breakdown



- **KernelOpt** reduces the time cost of blkmq layer by 46% thanks to no queuing time;
- As latency-critical I/Os are fetched by NVMe controller immediately, **KernelOpt** drastically reduces the waiting time;
- **CacheOpt** better utilizes the embedded cache layer and reduces the SSD access delays by 38%;
- By selectively using polling mechanism, **SelectISR** can reduce the I/O completion time by 5us.

Evaluation: online I/O access



- **CacheOpt** reduces the average I/O service latency, but it cannot eliminate the long tails;
- **KernelOpt** can remove the long tails, because it can avoid long queuing time and prevents throughput I/Os from blocking latency-critical I/Os;
- **SelectISR** reduces the average latency further, thanks to selectively using poll mechanism.

Conclusion

Observation

The ultra-low latency of new memory-based SSDs is not exposed to latency-critical application and have no benefit from user-experience angle;

Challenge

Piecemeal reformations of the current storage stack won't work due to multiple barriers; the storage stack is unaware of all behaviors of ULL-SSD and latency-critical applications;

Our solution

FlashShare: We expose different levels of I/O responsiveness to the key components in the current storage stack and optimize the corresponding system layers to make ULL visible to users (latency-critical applications).

Major results

- Reducing average turnaround response times by 22%;
- Reducing 99th-percentile turnaround response times by 31%.

Thank you