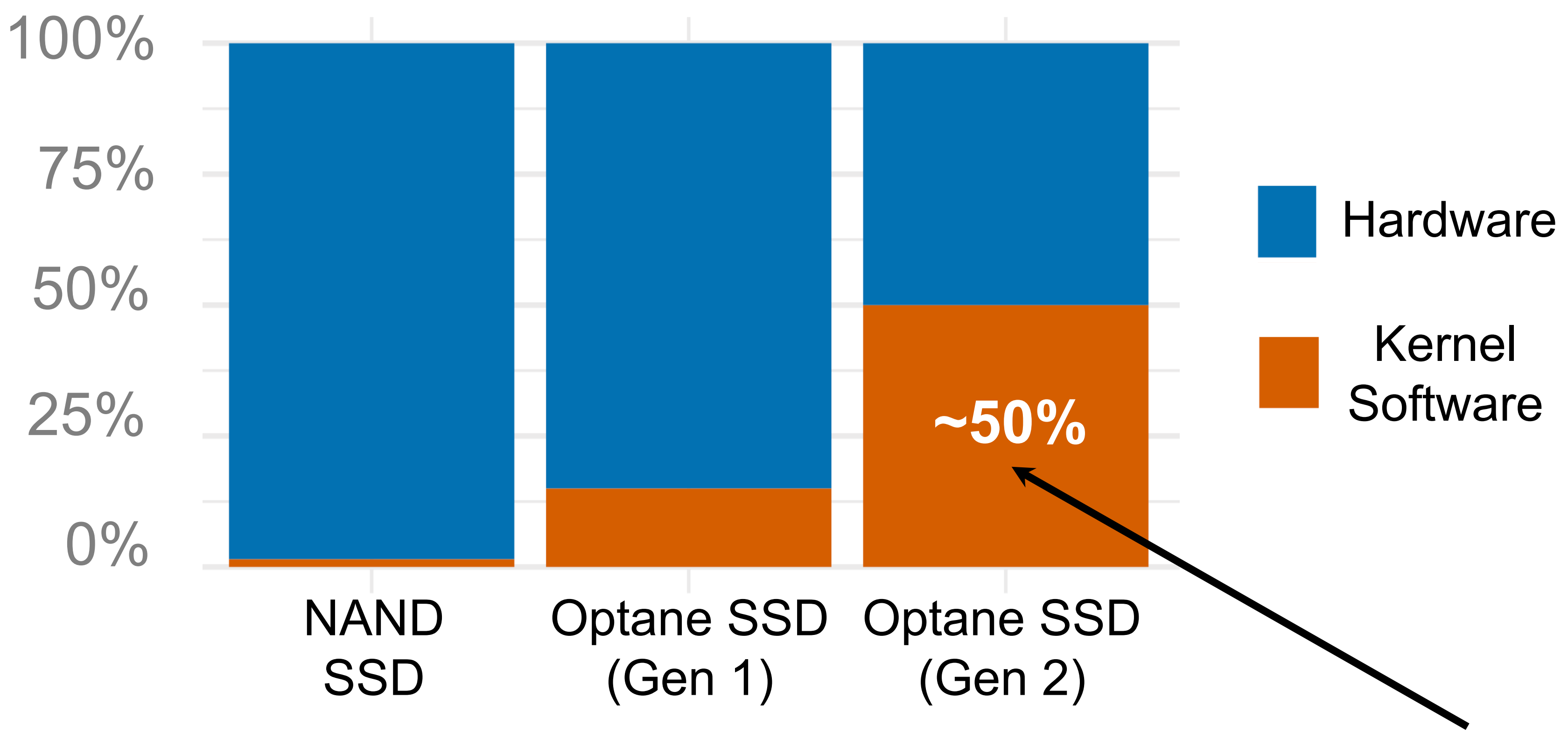
XRP: In-Kernel Storage Functions with eBPF

Yuhong Zhong[1], Haoyu Li[1], Yu Jian Wu[1], Ioannis Zarkadas[1], Jeffrey Tao[1], Evan Mesterhazy[1], Michael Makris[1], Junfeng Yang[1] Amy Tai[2], Ryan Stutsman[3], and Asaf Cidon[1]

1 Columbia University in the City of New York
2 Google
3 The University of Utah

Kernel Software is Becoming the Bottleneck for Storage

Average Read Latency Breakdown

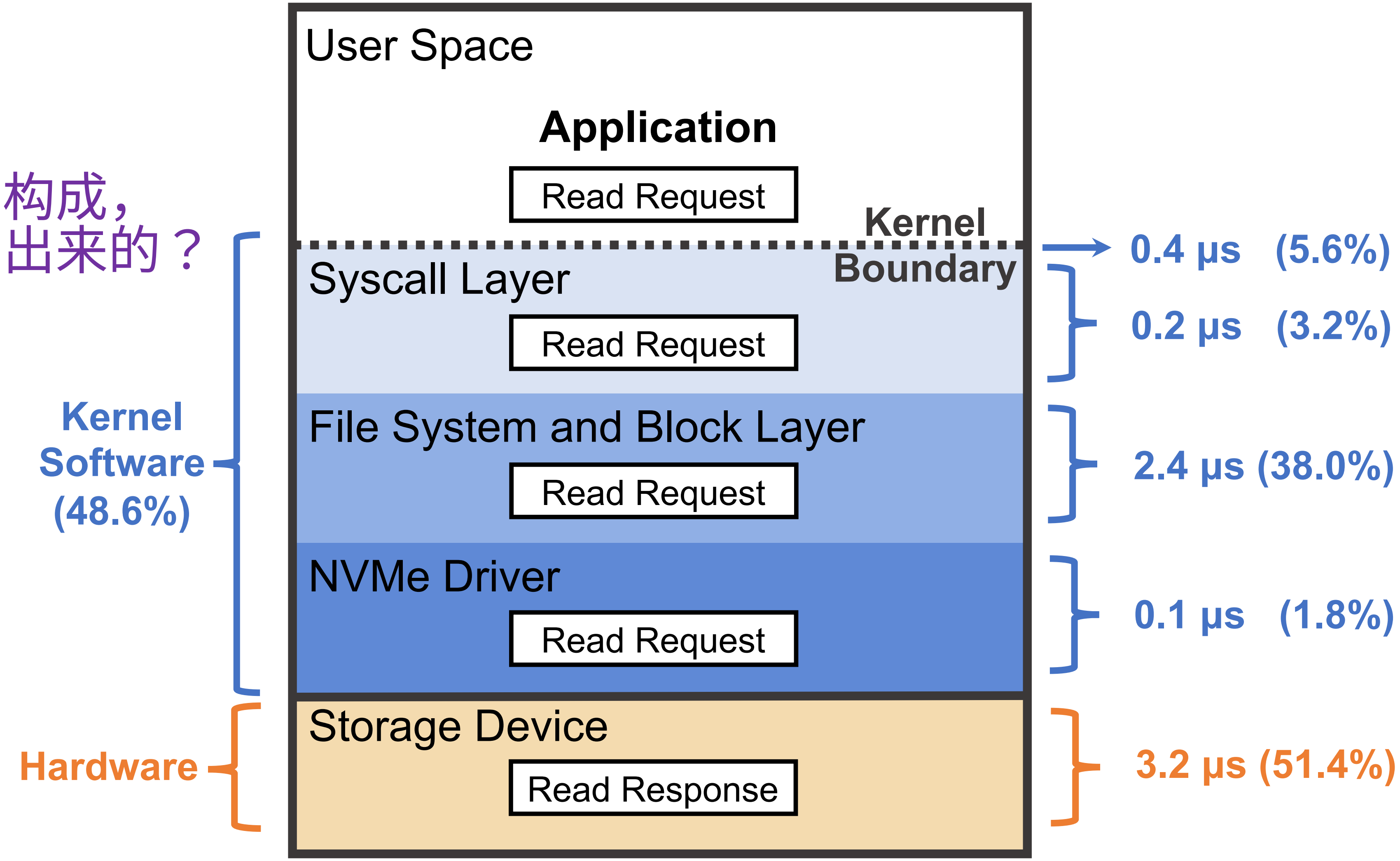


Kernel software overhead accounts for ~50% of read latency on Optane SSD Gen 2

Workload: Random 512B Read

Where Does the Latency Come From?

具体分析延迟的构成，
不过这是怎么测出来的？

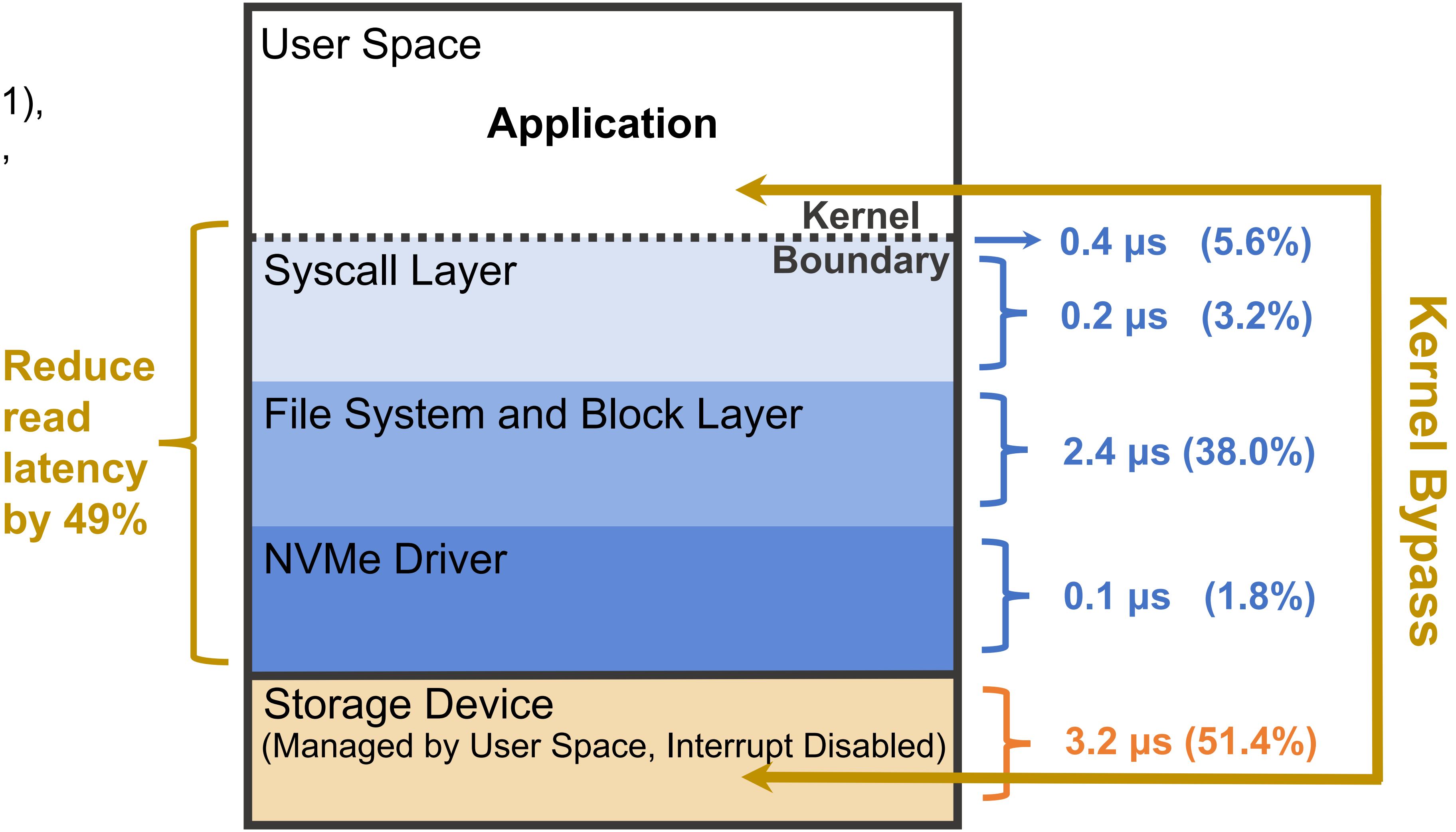


Workload: Random 512B Read, Disk: Optane SSD P5800X

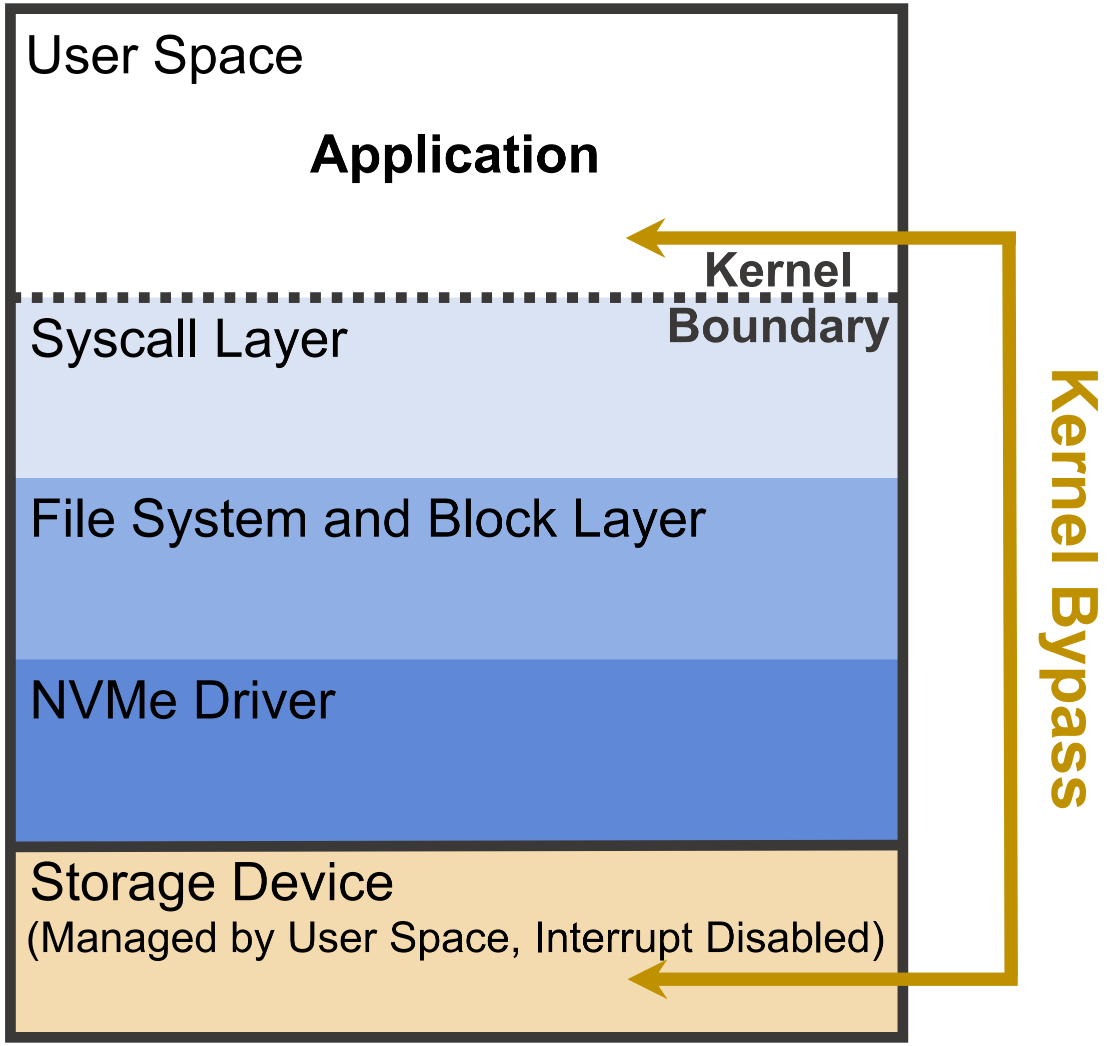
Bypass Kernel to Eliminate Overhead

Academic Work
Demikernel (SOSP ’21),
Shenango (NSDI ’19),
Snap (SOSP ’19),
IX (SOSP ’17),
Arrakis (OSDI ’14),
mTCP (NSDI ’14),
...

In industry, the most common library is SPDK



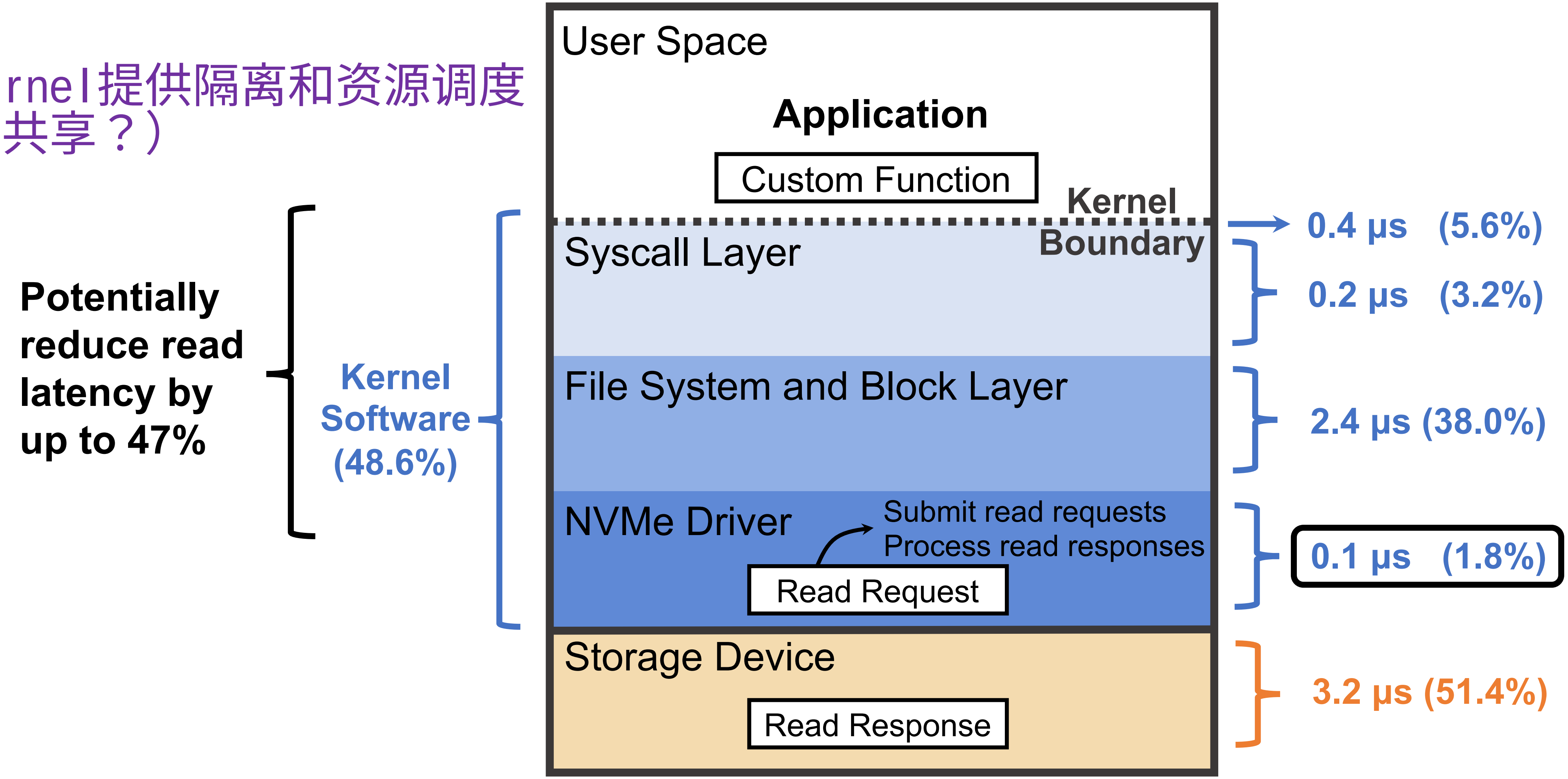
Kernel Bypass is Not a Panacea



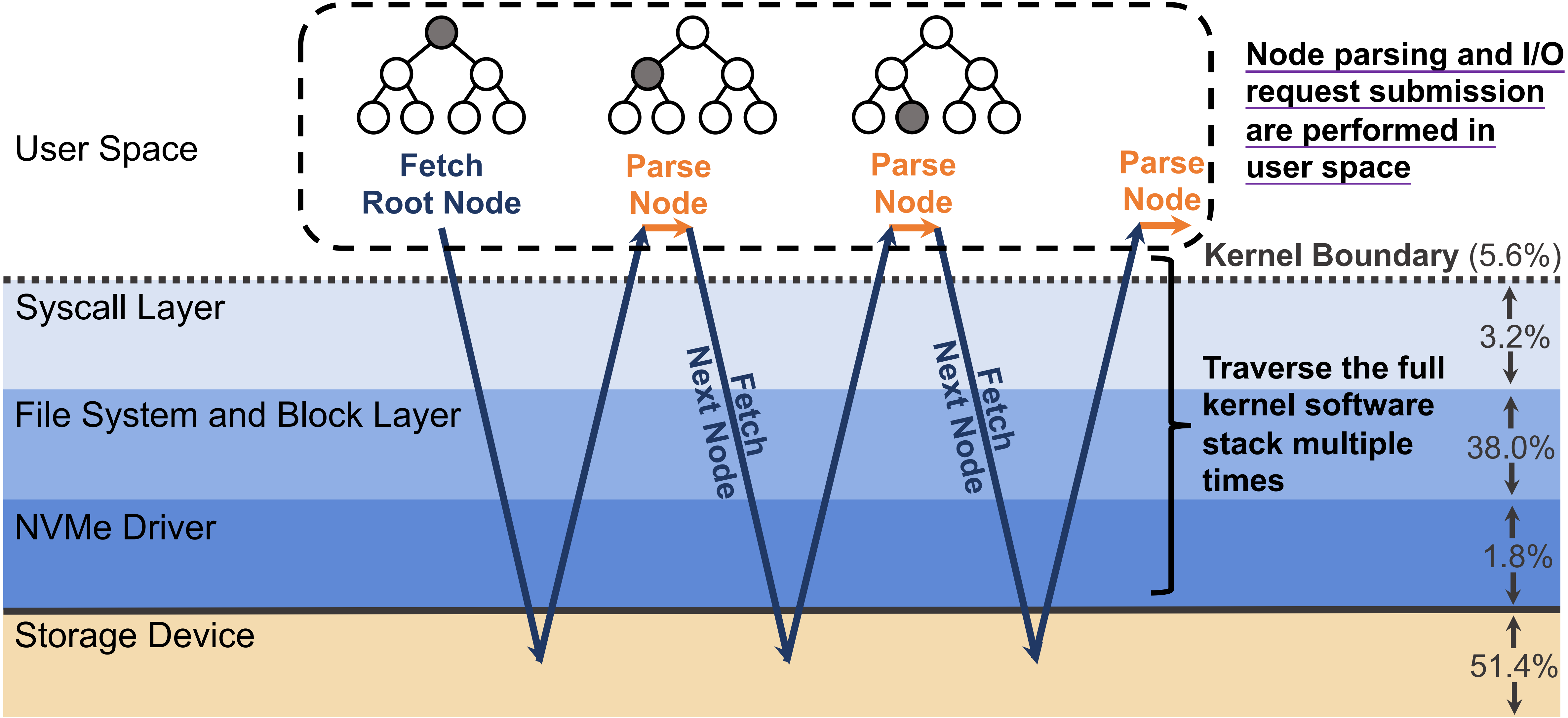
- ✅ Does not incur the overhead of the kernel storage stack
- ❌ No fine-grained access control
- ❌ Requires busy polling for completion
  - ❌ Processes cannot yield CPU when waiting for I/O
    - ❌ CPU cycles are wasted when I/O utilization is low
    - ❌ CPU cannot be shared efficiently among multiple processes

Move Application Logic Into the Kernel

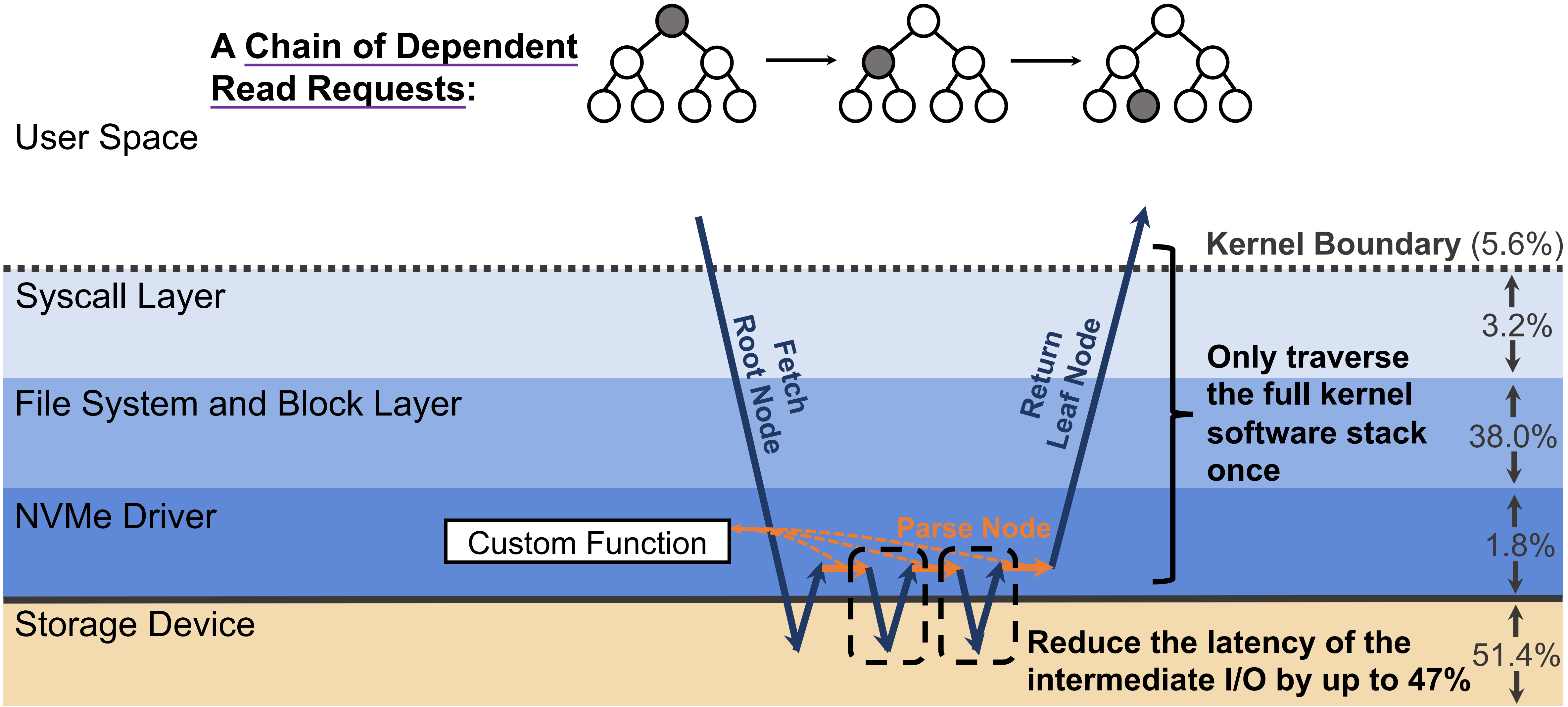
kernel提供隔离和资源调度（共享？）



B+ Tree Index Lookup from User Space



B+ Tree Index Lookup With an In-Kernel Function



Chains of Dependent Read Requests are Very Common

B-Tree **LSM Tree**

$$\underbrace{\hspace{20em}}_{\text{Issue dependent read requests to perform lookups}}$$

Goal: Build a framework for storage engines to accelerate dependent read requests using in-kernel functions

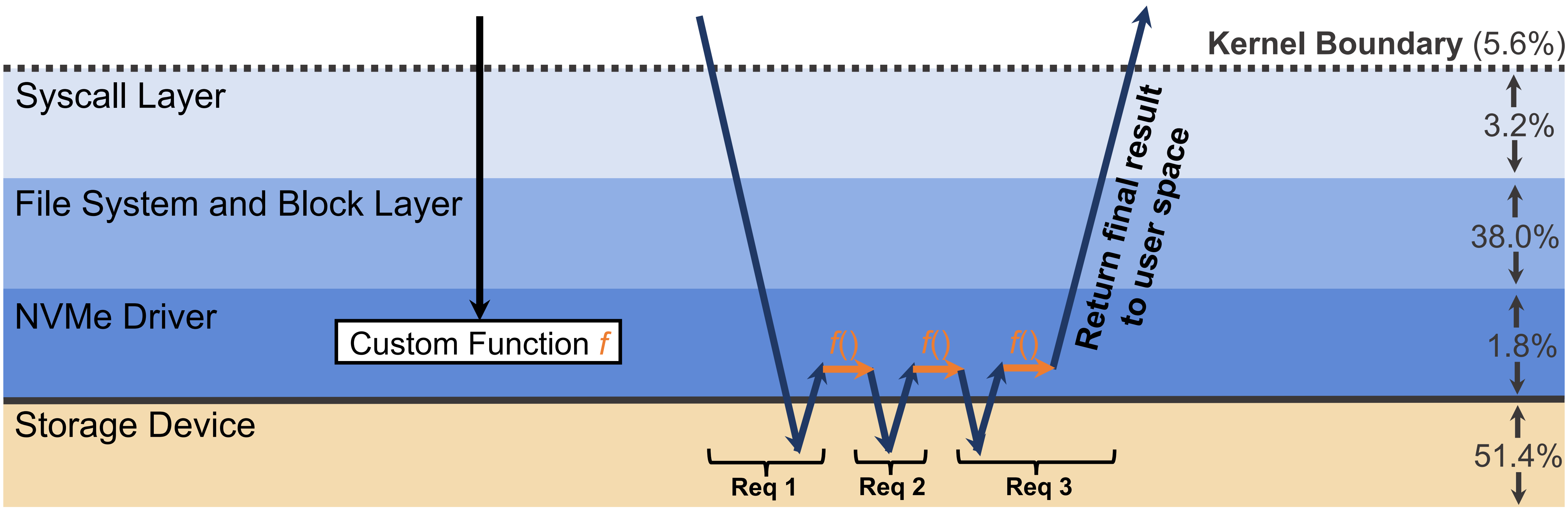
XRP: A Framework for In-Kernel Storage Functions

XRP can accelerate many types of operations such as index lookups, range queries, and aggregations

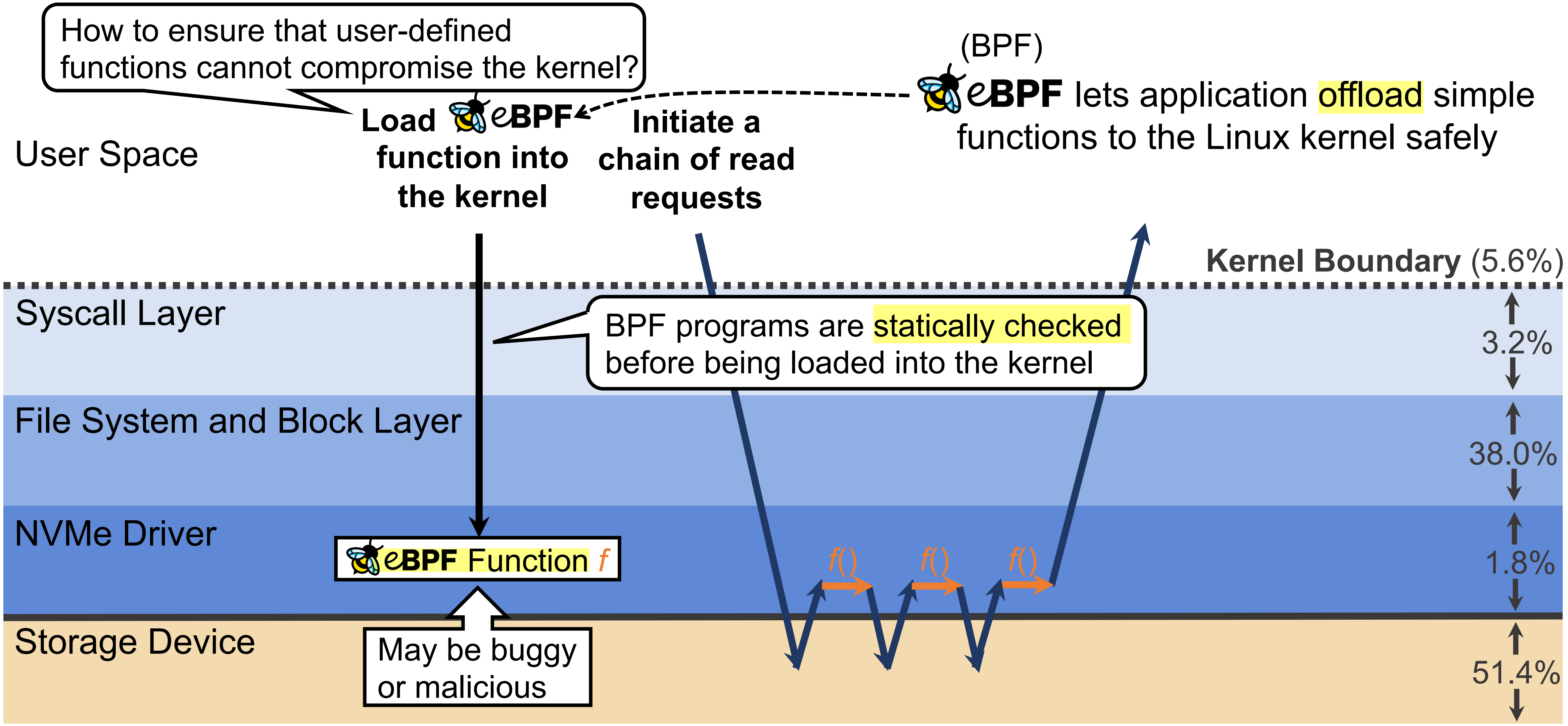
User Space

Load a custom function into the kernel

Initiate a chain of read requests



Using BPF to Offload Custom Functions Safely



BPF is Widely Used in Networking

eBPF

- Packet Filtering
- Packet Forwarding
- Packet Tracing
- Network Scheduling

A BPF program can operate on each packet independently

However, a storage BPF program needs to traverse a large on-disk data structure in a stateful way

Adopting BPF in Storage is Challenging

XRP is the first system that adopts BPF to reduce the kernel software overhead for storage

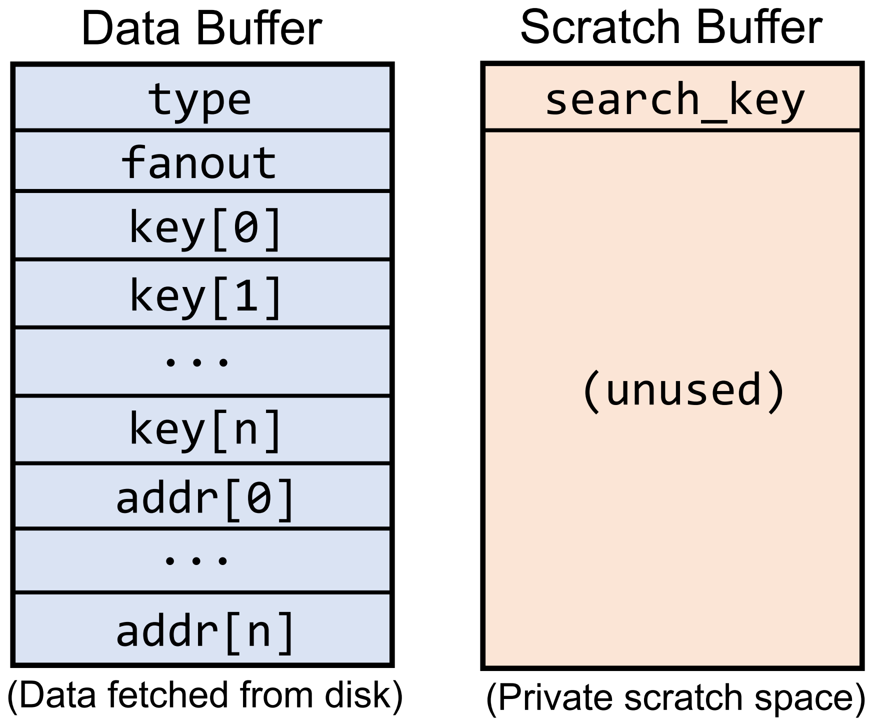
Key research challenges:

- Translating file offsets in the NVMe driver
- Augmenting the BPF verifier to support storage use cases
- Resubmitting NVMe requests
- Interaction with application-level caches

BPF Can Traverse Different Types of Data Structures

```c
u32 btree_lookup(struct bpf_xrp *context) {
    struct node *n = (struct node *) context->data;
    u64 search_key = *(u64 *) context->scratch;
    if (node->type == LEAF) {
        context->done = true;
        return 0;
    }
    int i;
    for (i = 1; i < MIN(n->fanout, MAX_FANOUT); ++i) {
        if (search_key < n->key[i]) break;
    }
    context->done = false;
    context->next_addr[0] = n->addr[i - 1];
    return 0;
}
```

MAX_FANOUT ensures for loop is bounded



XRP: In-Kernel Storage Functions with eBPF

XRP $\xrightarrow{\text{Integrate}}$

BPF-KV

A simple B+ tree key-value store

mongoDB®

WIREDTIGER (LSM Tree)

A popular production key-value store

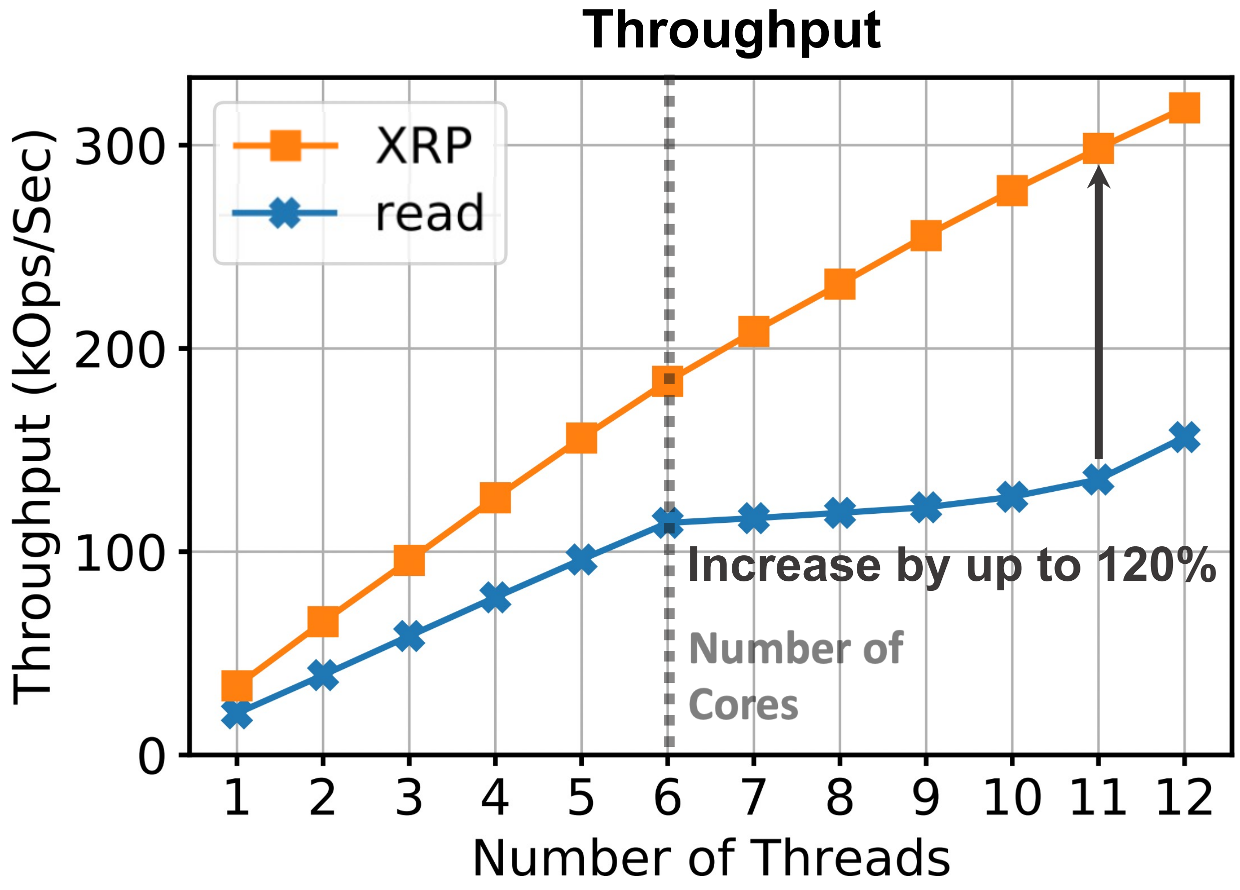
Evaluation

- What is the performance benefit of XRP?
- How does XRP compare to kernel bypass?
- What types of operations can XRP support?
- Can XRP accelerate a production key-value store?

See the paper

XRP Nearly Eliminates the Kernel Software Overhead

Multi-threaded throughput in BPF-KV with uniform random 512B read:

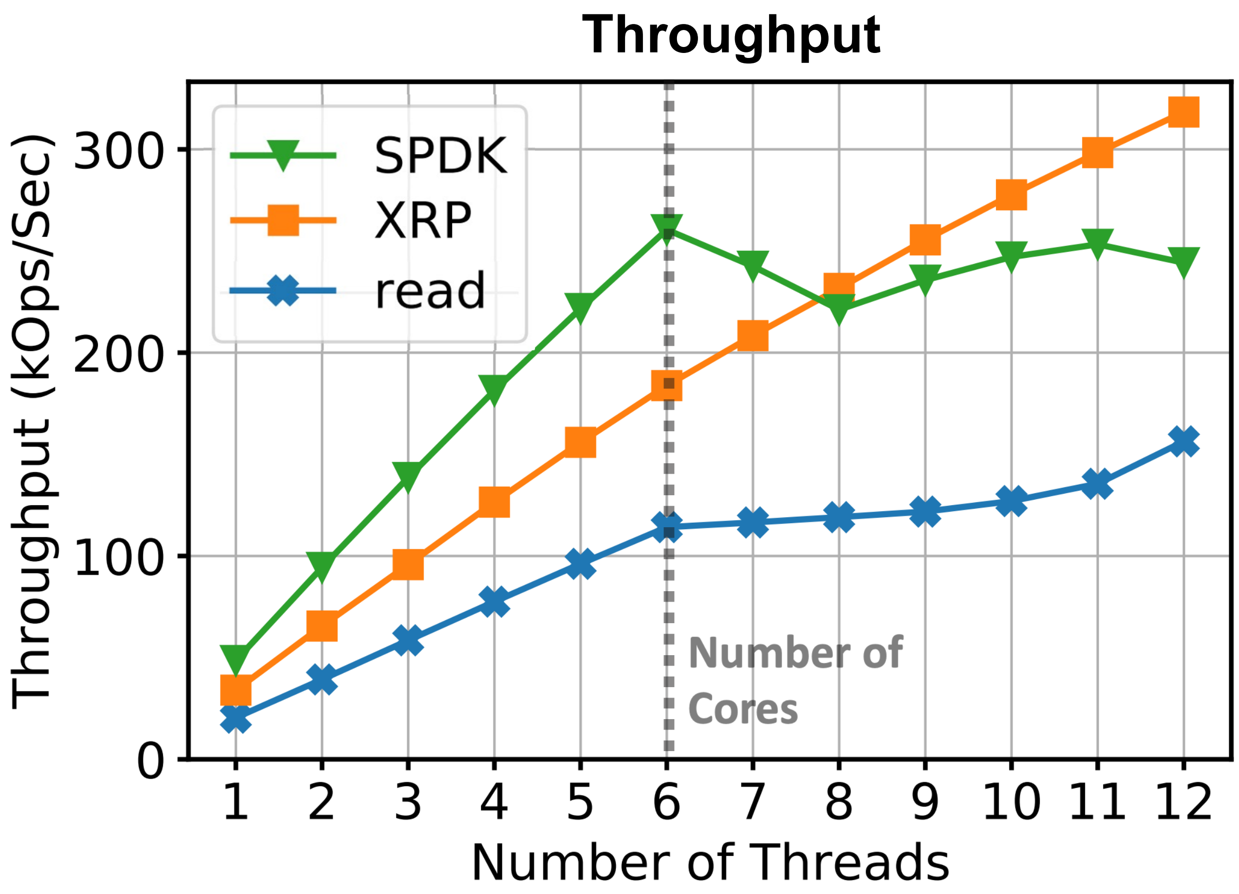


XRP can scale well even if the number of threads exceeds the number of cores

This is because XRP alleviates the CPU contention by reducing the CPU overhead per IO request

XRP Handles CPU Contention, SPDK Not So Much

Multi-threaded throughput in BPF-KV with uniform random 512B read:



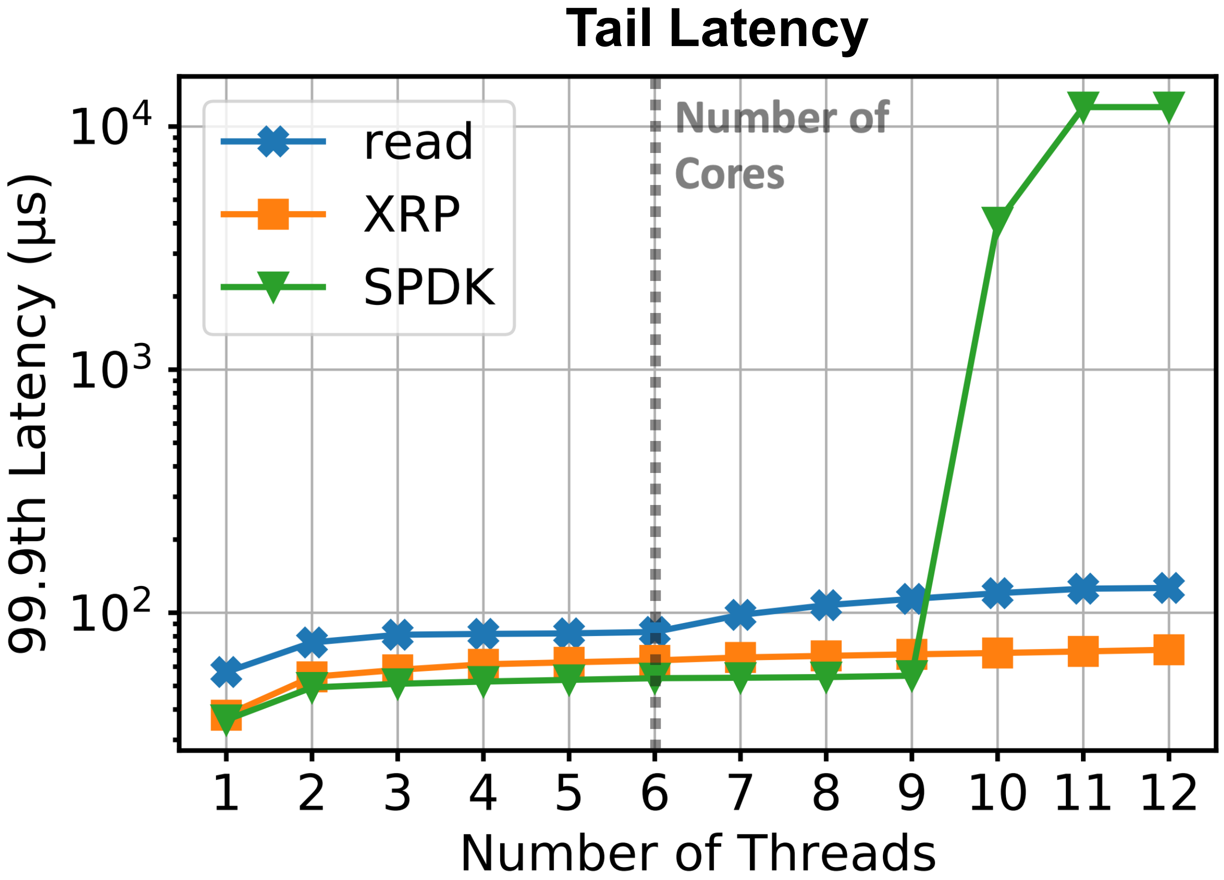
SPDK fails to scale beyond 6 threads because SPDK threads cannot yield CPU when waiting for I/O to complete

XRP provides performance that is close to/better than SPDK without sacrificing isolation and CPU efficiency

Each thread represents a different storage application on the same machine

XRP Handles CPU Contention, SPDK Not So Much

Multi-threaded tail latency in BPF-KV with uniform random 512B read:



Compared to read, XRP improves tail latency by up to 45%

Tail latency of SPDK spikes to ~10 ms when the number of threads is greater than the number of cores by more than 50%

Conclusions

ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED

- XRP is the first system to use BPF to accelerate common storage functions
- XRP captures most of the performance benefit of kernel bypass, without sacrificing CPU utilization and access control

We are actively integrating XRP with other popular key-value stores including RocksDB

Try it out: http://xrp-project.com/

yuhong.zhong@columbia.edu