# User Space Buffer Cache with the Bells and Whistles of a Kernel Page Cache

Riwei Pan, **Yu Liang**, Sam H. Noh, Lei Li, Nan Guan, Tei-Wei Kuo, Chun Jason Xue











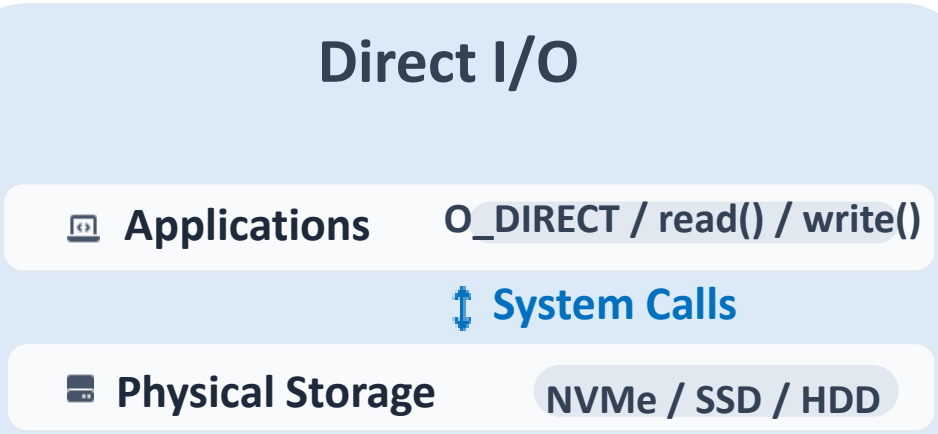



MOHAMED BIN ZAYED
UNIVERSITY OF
ARTIFICIAL INTELLIGENCE

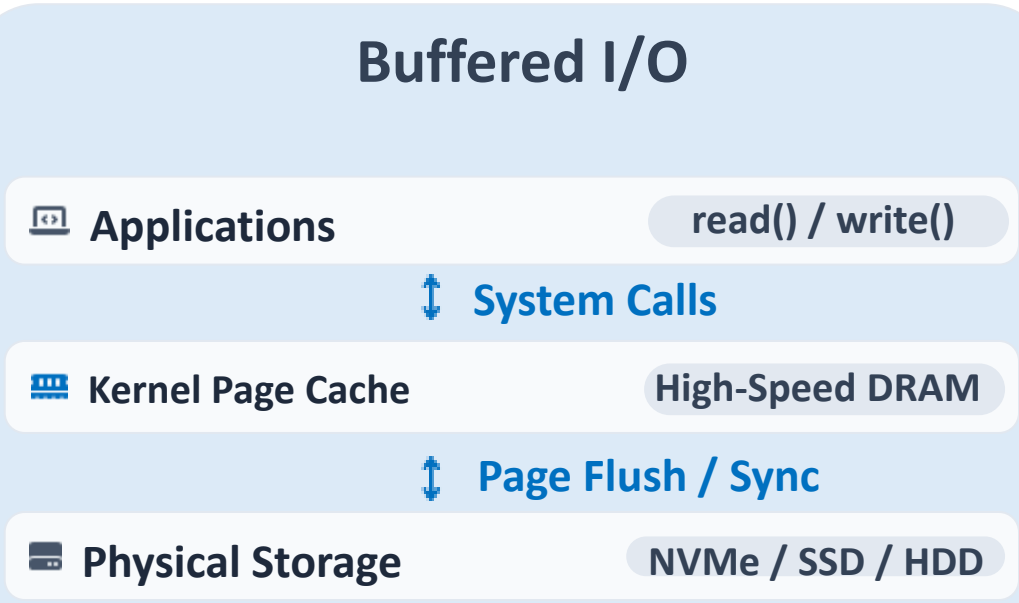
# Two I/O modes: Direct I/O and Buffered I/O

香港城市大學
City University of Hong Kong

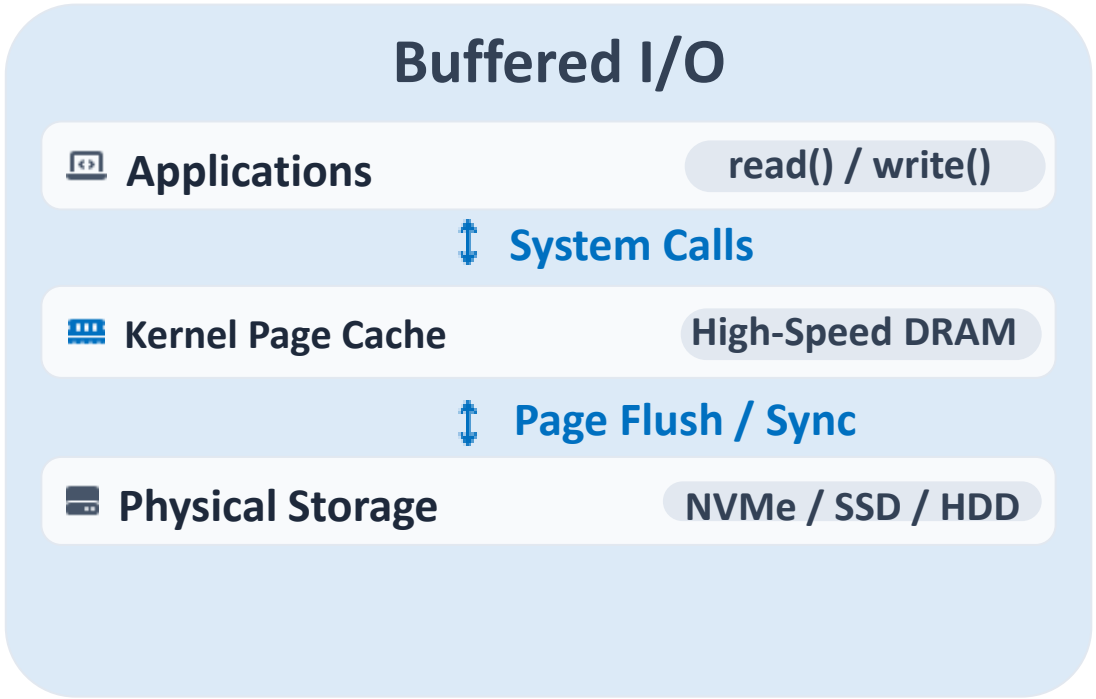
## Direct I/O




## Buffered I/O



# Buffered I/O



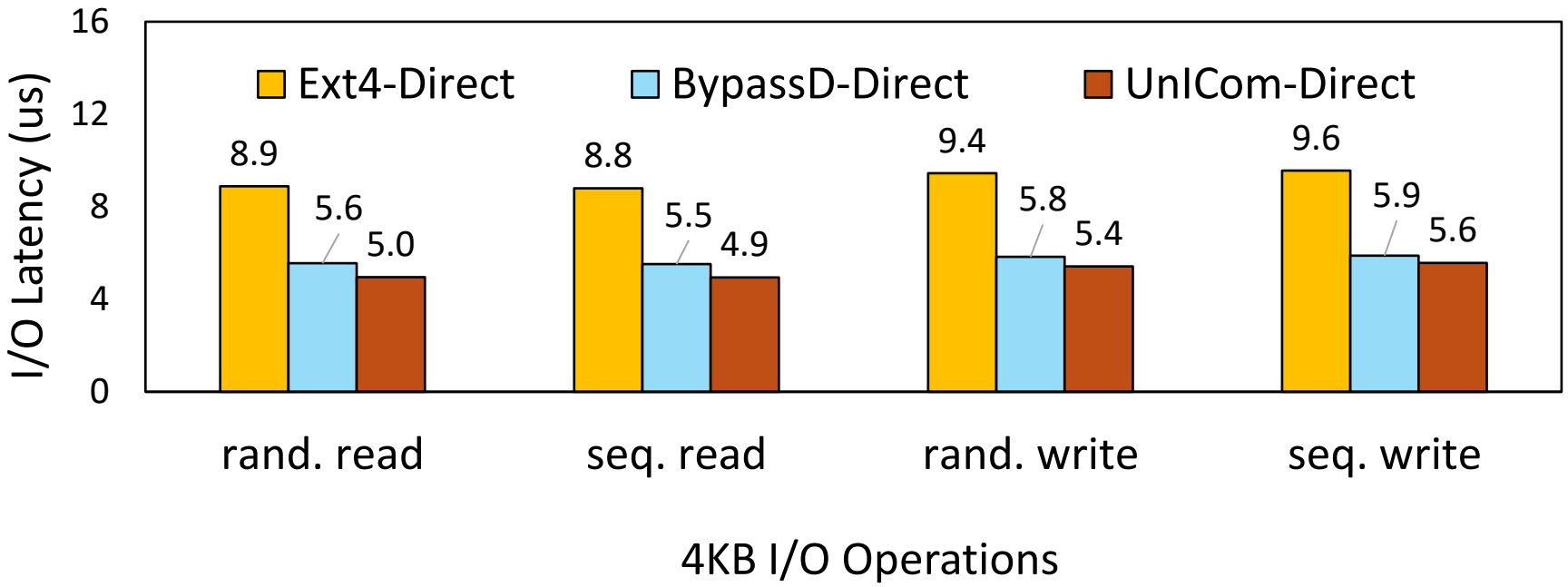

## Two Key Benefits:

- **Achieve high performance**:
  - Handle requests in high-speed DRAM in most cases (cache hit).
- **Reduce I/O number**:
  - Merge small writes into efficient page-sized flushes.

Buffered I/O is the default I/O mode of system calls.

# Overlooked Buffered I/O

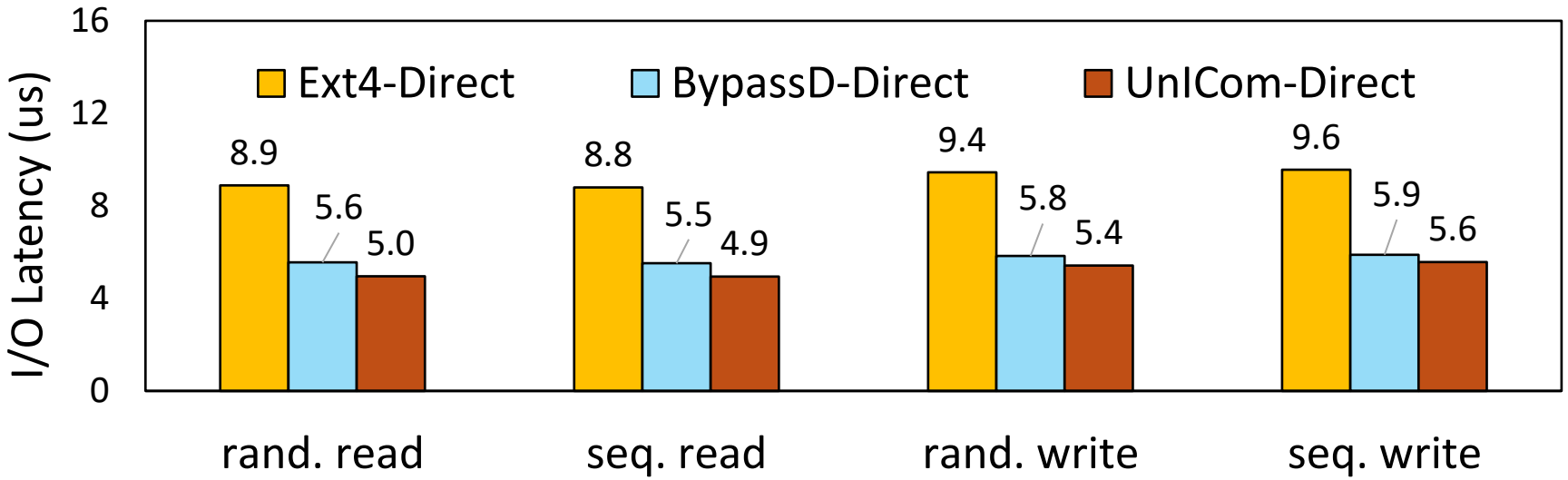
- The SOTA I/O stack optimizations (e.g., BypassD [ASPLOS'24] and UnICom[FAST'26]) focus on direct I/O, **overlooking buffered I/O**.



# Overlooked Buffered I/O

香港城市大學
City University of Hong Kong

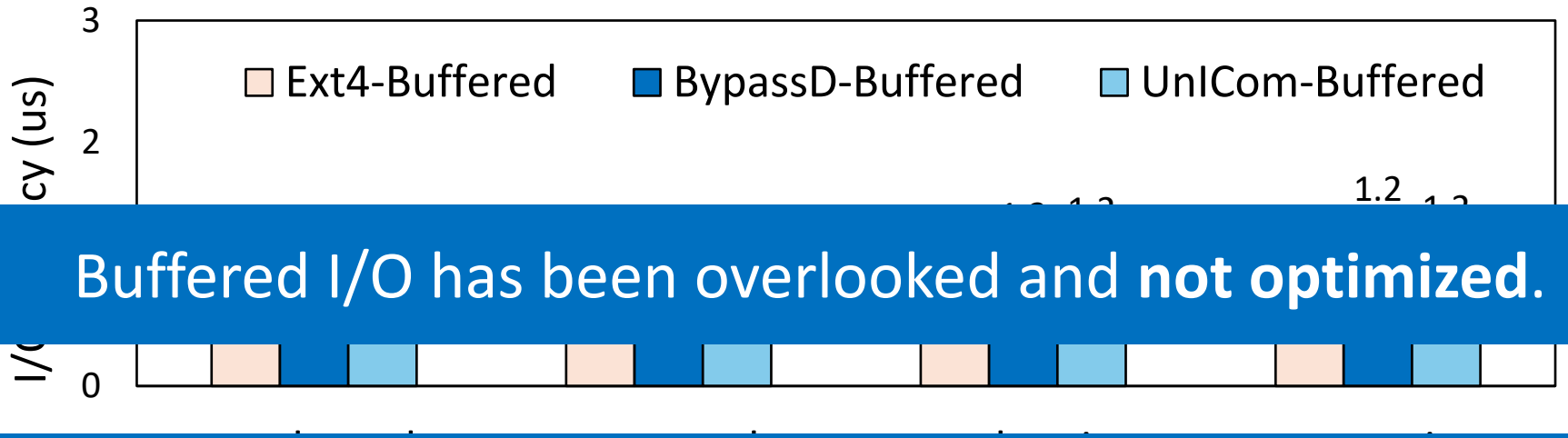
- The SOTA I/O stack optimizations (e.g., BypassD [ASPLOS'24] and UnICom[FAST'26]) focus on direct I/O, **overlooking buffered I/O**.




Direct I/O has been optimized significantly.

# Overlooked Buffered I/O

- The SOTA I/O stack optimizations (e.g., BypassD [ASPLOS'24] and UnICom[FAST'26]) focus on direct I/O, **overlooking buffered I/O.**

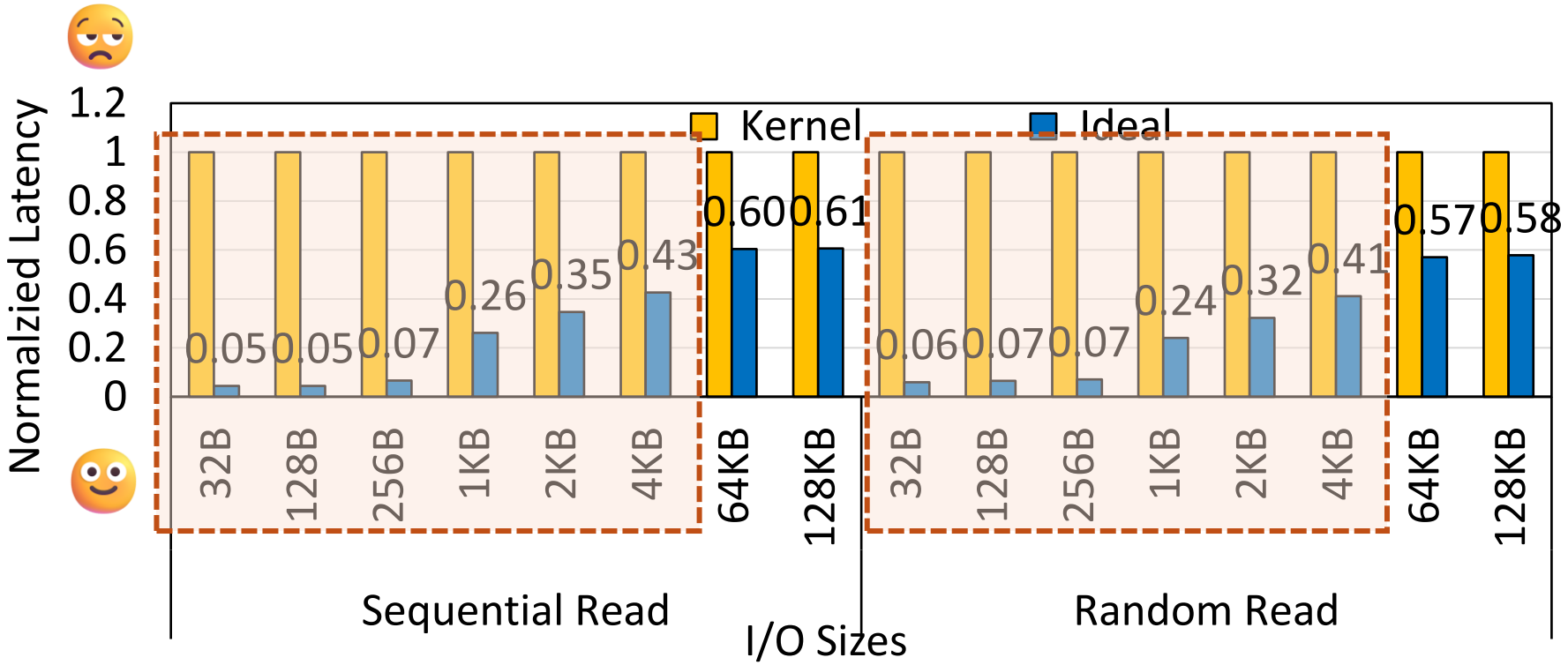



Buffered I/O has been overlooked and **not optimized**.

Buffered I/O is the **default I/O mode** of system calls and has **better performance**.

# Improvement Room for Buffered I/O

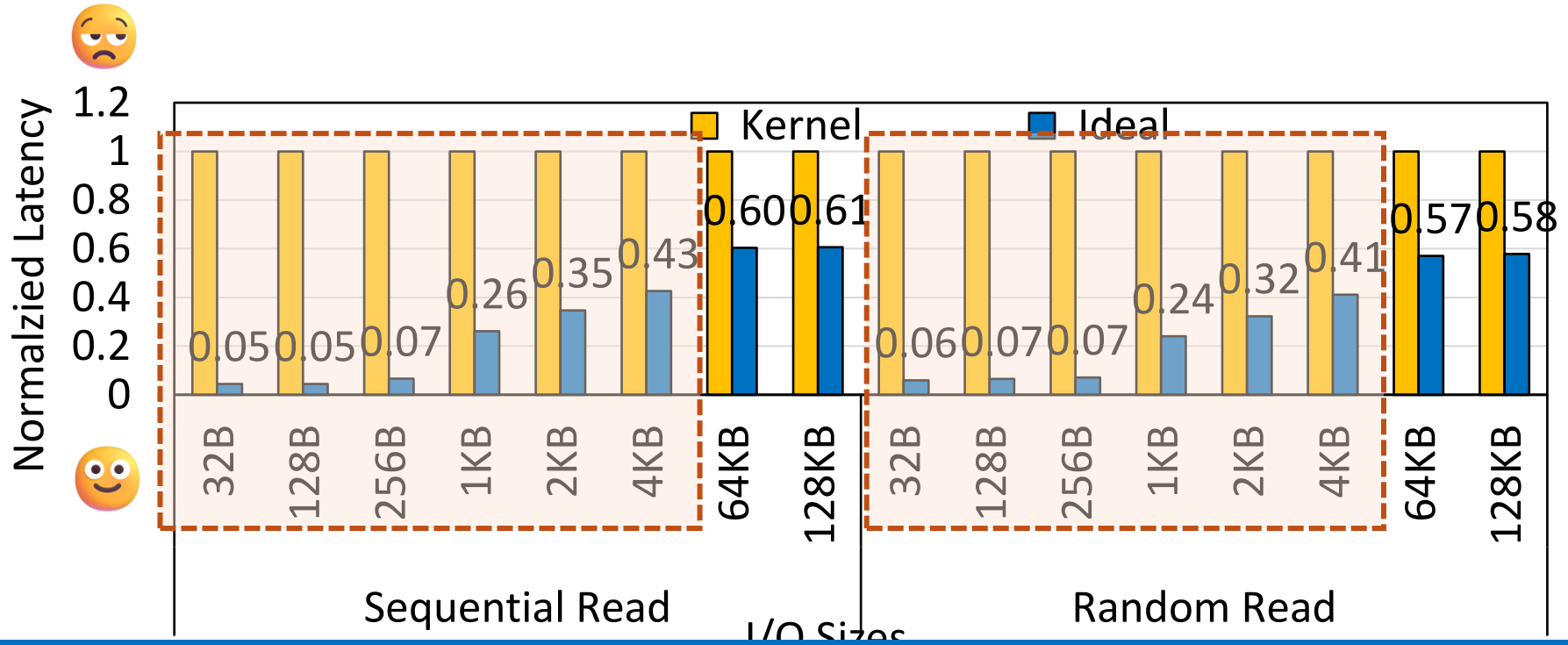
- Put all the things in the user space (Ideal).




**Ideal:** maintaining a radix tree in the user space without kernel locking and VFS retrievers.

# Improvement Room for Buffered I/O

- Put all the things in the user space (Ideal).




Large room to improve buffered I/Os, especially for small I/Os

**Ideal:** maintaining a radix tree in the user space without kernel locking and VFS retrievers.

# Small Buffered I/O Prevalence

香港城市大學
City University of Hong Kong

- Real-world workloads are dominated by small I/O requests (≤ 4KB).

## DATABASE STORAGE WORKLOADS

### RocksDB (KV Store)

YCSB / FB Workloads

**99%**
Reads ≤ 4KB (Run A~F)

**99%**
Writes < 256B (Append)

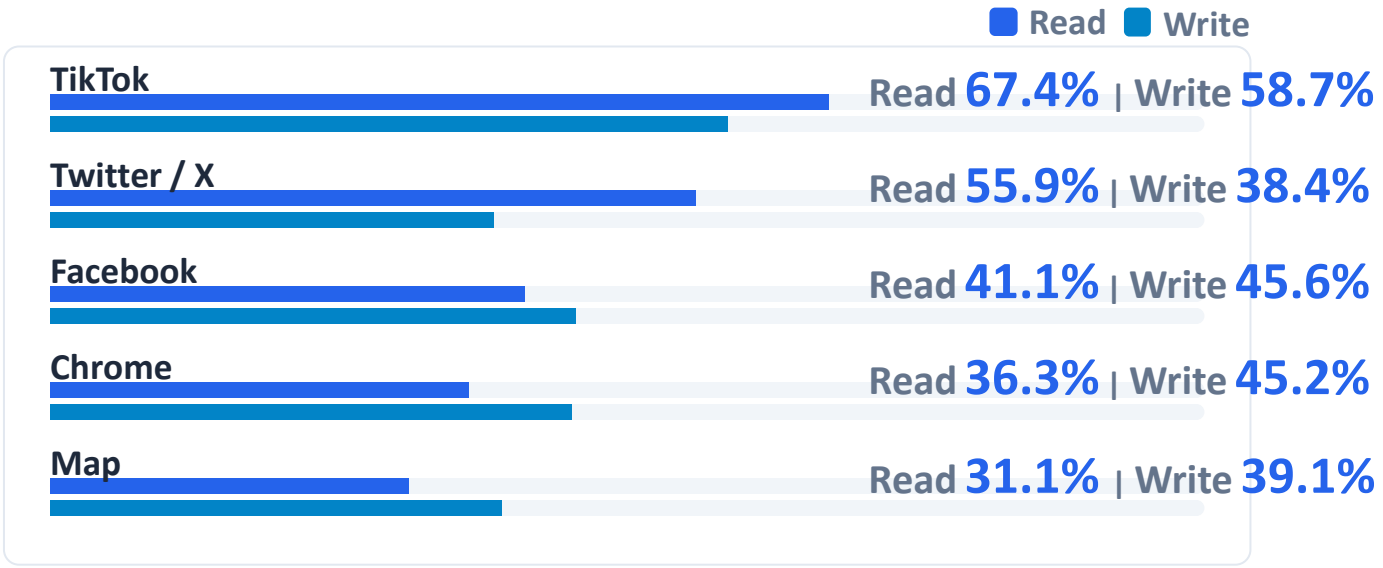
### SQLite (Relational)

TPC-C Workload

**99%**
Reads exactly 4KB

**99%**
Writes ≤ 4KB
(38% <128B by WAL)

## MOBILE APP (< 32B)



# Small Buffered I/O Prevalence

香港城市大學
City University of Hong Kong

- Real-world workloads are dominated by small I/O requests (≤ 4KB).

## DATABASE STORAGE WORKLOADS

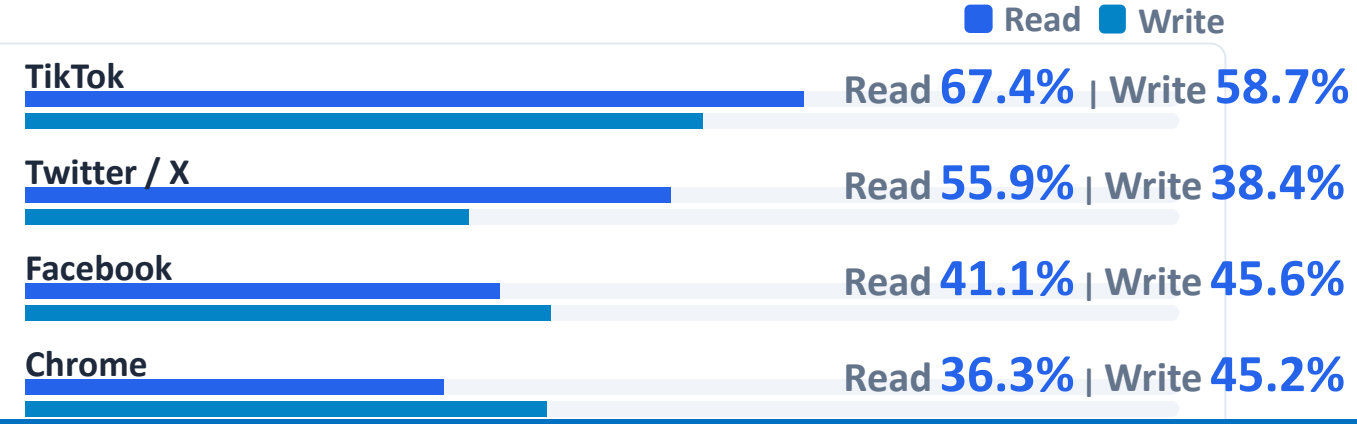
**RocksDB (KV Store)** — YCSB / FB Workloads

- **99%** Reads ≤ 4KB (Run A~F)
- **99%** Writes < 256B (Append)

**SQLite (Relational)** — TPC-C Workload

- **99%**
- **99%**

## MOBILE APP (< 32B)




Optimizing small buffered I/Os is important.

# Small Buffered I/O Prevalence

香港城市大學
City University of Hong Kong

- Real-world workloads are dominated by small I/O requests (≤ 4KB).

## DATABASE STORAGE WORKLOADS

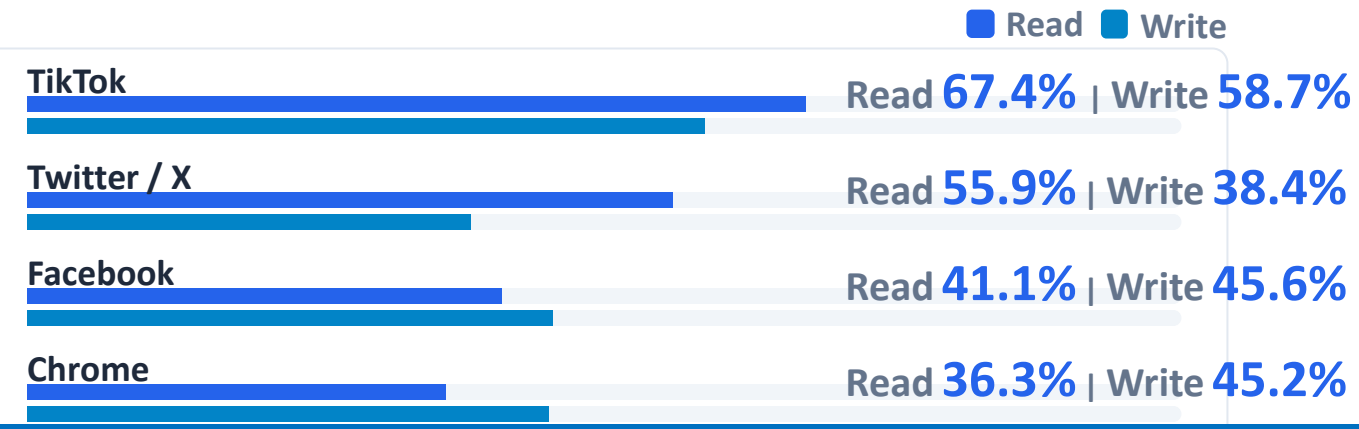
**RocksDB (KV Store)** — YCSB / FB Workloads

- 99% Reads ≤ 4KB (Run A~F)
- 99% Writes < 256B (Append)

**SQLite (Relational)** — TPC-C Workload

- 99%
- 99%

## MOBILE APP (< 32B)




Optimizing small buffered I/Os is important.

Let’s do user space buffer cache!

# User Space Buffer Cache with the Bells and Whistles of a Kernel Page Cache

香港城市大學
City University of Hong Kong

- Retain the **core properties** of kernel page cache and **achieve high performance**

### 1. File-Level Caching

Operates at file granularity to support the default optimizations, including **readahead** and **asynchronous writeback**.

### 2. Global User-Space Memory

Enables **cross-process** allocation and cold-page eviction, ensuring high memory efficiency without stranded capacity.

### 3. Hardware Independence

Ensures **broad deployment** without requiring specialized CPU/hardware extensions.

### 4. Low Access Overhead

**Accelerates common cases** (cache hit) by eliminating software costs—executing **without system calls or IPC**.

# Limitations of Existing User-space Page Caching

香港城市大學
City University of Hong Kong

- Existing user-space caching fall short in these four properties.

- **TriCache (OSDI'21)** and **ScalaCache (ATC'24)** use per-process memory pool and target on block-level cache, without exploiting file locality.

| System | File-Level Cache | High Mem. Efficiency | Hardware Agnostic | Low-Cost Access |
|---|---|---|---|---|
| TriCache | ✗ | ✗ | ✓ | ✓ |
| ScalaCache | ✗ | ✗ | ✗ | ✓ |
| uFS | ✓ | ✗ | ✓ | ✗ |
| Aeolia | ✓ | ✗ | ✗ | ✓ |
| Our Goal | ✓ | ✓ | ✓ | ✓ |

✓ Satisfied ✗ Deficient / Unmet

# Limitations of Existing User-space Page Caching

香港城市大學
City University of Hong Kong

- Existing user-space caching fall short in these four properties.

- **uFS (SOSP'21)** uses fixed memory pool to store and access cached pages through high-cost IPC.

| System | File-Level Cache | High Mem. Efficiency | Hardware Agnostic | Low-Cost Access |
|---|---|---|---|---|
| TriCache | ✗ | ✗ | ✓ | ✓ |
| ScalaCache | ✗ | ✗ | ✗ | ✓ |
| uFS | ✓ | ✗ | ✓ | ✗ |
| Aeolia | ✓ | ✗ | ✗ | ✓ |
| Our Goal | ✓ | ✓ | ✓ | ✓ |

✓ Satisfied ✗ Deficient / Unmet

# Limitations of Existing User-space Page Caching

香港城市大學
City University of Hong Kong

- Existing user-space caching fall short in these four properties.

- **Aeolia (SOSP'21)** employs per-process memory pool and relies on specific CPU instructions.

| System | File-Level Cache | High Mem. Efficiency | Hardware Agnostic | Low-Cost Access |
|---|---|---|---|---|
| TriCache | ✗ | ✗ | ✓ | ✓ |
| ScalaCache | ✗ | ✗ | ✗ | ✓ |
| uFS | ✓ | ✗ | ✓ | ✗ |
| Aeolia | ✓ | ✗ | ✗ | ✓ |
| Our Goal | ✓ | ✓ | ✓ | ✓ |

✓ Satisfied ✗ Deficient / Unmet

# Limitations of Existing User-space Page Caching

香港城市大學
City University of Hong Kong

- Existing user-space caching fall short in these four properties.

| System | File-Level Cache | High Mem. Efficiency | Hardware Agnostic | Low-Cost Access |
|---|---|---|---|---|
| TriCache | ✗ | ✗ | ✓ | ✓ |
| ScalaCache | ✗ | ✗ | ✗ | ✓ |
| uFS | ✓ | ✗ | ✓ | ✗ |
| Aeolia | ✓ | ✗ | ✗ | ✓ |
| **Our Goal** | ✓ | ✓ | ✓ | ✓ |

✓ Satisfied ✗ Deficient / Unmet

Our goal is to achieve **all** four core properties.

# Key Challenges

香港城市大學
City University of Hong Kong

To Achieve ALL FOUR CORE PROPERTIES:

- File-Level Cache
- High Memory Efficiency
- Hardware Independency
- Low-Cost Access

**CHALLENGE 1** Low-Cost Flexible User-Space Memory Management

- High Mem. Efficiency
- Low-Cost Access

**CHALLENGE 2** Cross-Process Page Reclamation

- High Mem. Efficiency
- Low-Cost Access

**CHALLENGE 3** Transparent to I/O Path

- File-Level Cache
- Hardware Independency

# Preliminary Solution

香港城市大學
City University of Hong Kong

## Key idea: Decoupling data and control planes

Delegating per-process data I/O path to the user space
while maintaining the global memory management to the kernel.

**Nexus**

DESIGN 1 · REDUCE KERNEL OVERHEAD

### Two-Level Dynamic Allocation

- **Kernel Level:** Manage coarse-grained memory segments allocated to user space.
- **User Level:** User processes allocate fine-grained pages in segments.

DESIGN 2 · INCREASE MEMORY EFFICIENCY

### Global Cold-Page Reclamation

- **Kernel Selector:** Drives global eviction policies and decide reclaim which pages.
- **User Executor:** User processes perform the actual reclamation operation to ensure synchronization.

DESIGN 3 · EFFICIENT I/O STACK

### Unified Kernel-Bypass Path

- **When cache miss:** cold read and writeback also exploit previous bypass optimizations to minimize I/O path overhead.

# Experimental Platform

香港城市大學
City University of Hong Kong

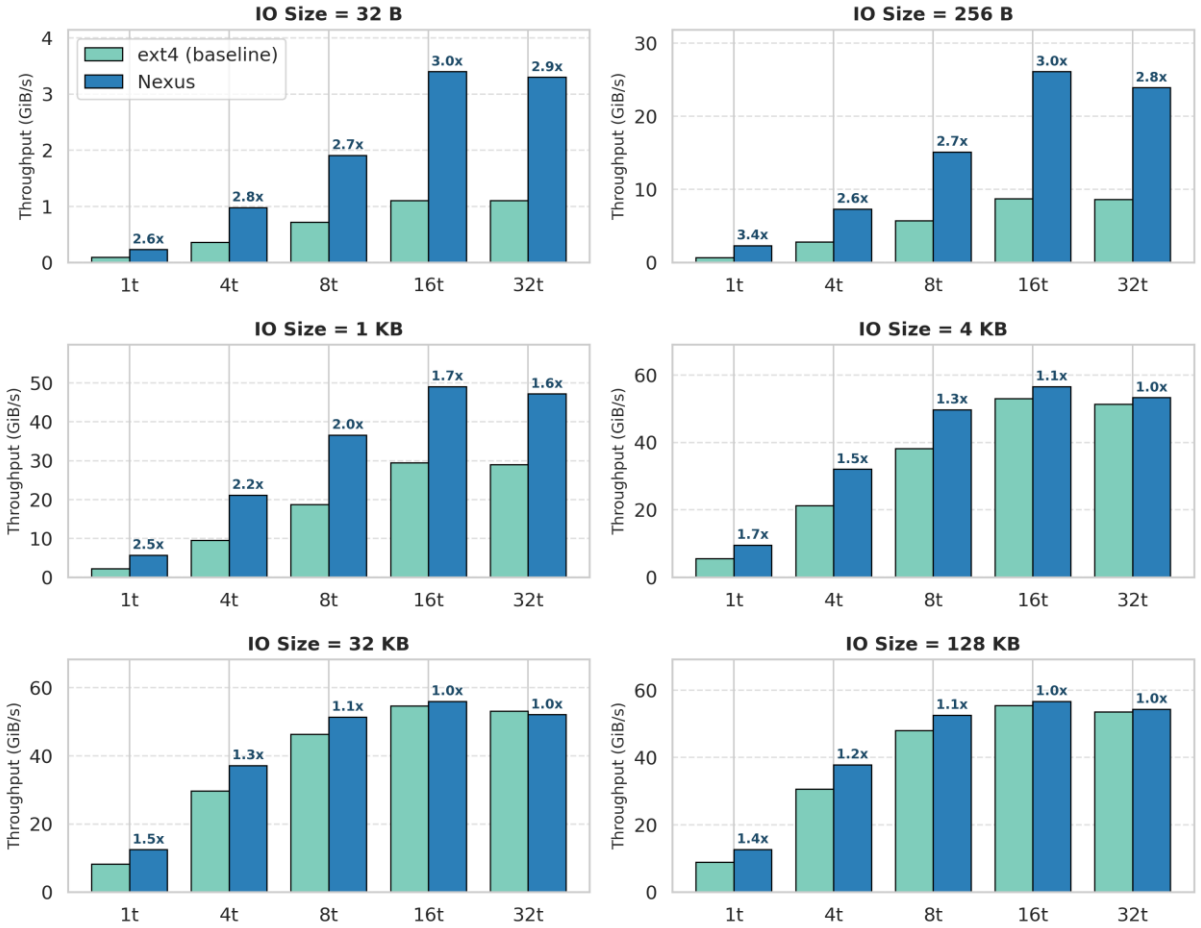
## HARDWARE & OS SPECIFICATIONS

| | |
|---|---|
| CPU | **Intel Core i9-14900K**<br>8 P-cores @ 3.2 GHz + 16 E-cores @ 2.4 GHz |
| DRAM | **32 GB System Memory**<br>High-speed main memory |
| SSD | **400 GB Intel Optane SSD P5801x**<br>Ultra-low latency NVMe storage |
| OS / Kernel | **Ubuntu 20.04 LTS**<br>Linux Kernel 6.5.1 |

# Evaluation Results

香港城市大學
City University of Hong Kong

## Sequential Read Performance for FIO Workloads




Thread Number (1 thread ~ 32 threads)

**Small I/O (< 1 KB)** **2.6 – 3.4×**

Substantial throughput gain, remaining stable across 1–32 threads.

**Medium I/O (4 KB)** **1.1 – 1.7×**

Maintains consistent speedups over ext4 in multi-threaded workloads.

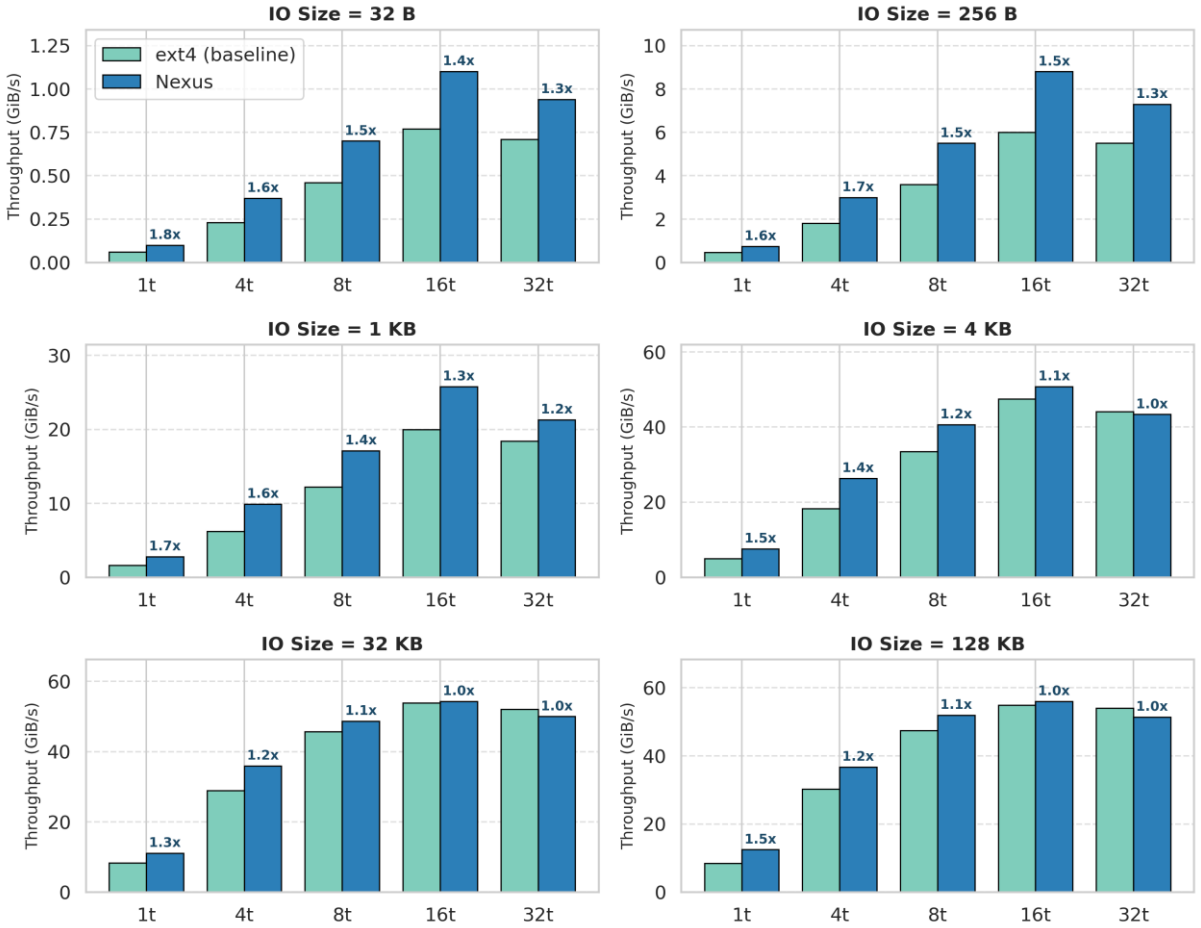
**Large I/O (> 4 KB)** **1.0 – 1.3×**

Data transfer time dominates, shifting the bottleneck to hardware bandwidth.

# Evaluation Results

香港城市大學
City University of Hong Kong

## Random Read Performance for FIO Workloads




### Small I/O (< 1 KB) — 1.3 – 1.7×

Outperforms ext4 steadily without relying on sequential prefetching.

### Medium I/O (4 KB) — 1.1 – 1.5×

Delivers solid speedups without degradation under multi-threaded concurrency.

### Large I/O (> 4 KB) — 1.0 – 1.3×

Speedups converge as device random access latency becomes primary limit.

# Conclusion

香港城市大學
City University of Hong Kong

### Problem

**Small buffered I/O requests dominate** real-world workloads but suffer **high kernel software overheads** from VFS retrievers, kernel locking, and context switching.

### Existing Work & Limitations

- The SOTA I/O optimizations (e.g., BypassD and UnICom) **focus on Direct I/O**.
- Current user-space caches **fail to offer** file-level locality, incur high IPC costs, dynamic memory management, or rely on specialized hardware.

### Goal & Design

Combine **all four** kernel page cache benefits in user space **by decoupling data and control planes**.

### Experimental Proof

Evaluation results show significant performance gains over ext4, reaching up to **3.4× speedup** for small sequential reads (<1 KB) and **1.7×** for random reads.

# Thank you for listening!

# User Space Buffer Cache with the Bells and Whistles of a Kernel Page Cache

Riwei Pan, **Yu Liang**, Sam H. Noh, Lei Li, Nan Guan, Tei-Wei Kuo, Chun Jason Xue

(Contact: riweipan-c@my.cityu.edu.hk)














MOHAMED BIN ZAYED
UNIVERSITY OF
ARTIFICIAL INTELLIGENCE