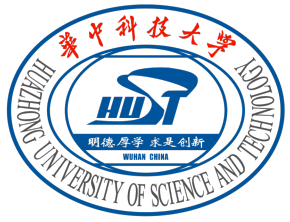
ScalaCache: Scalable User-Space Page Cache Management with Software-Hardware Coordination

Li Peng[1], Yuda An[1], You Zhou[3], Chenxi Wang[4], Qiao Li[5], Chuanning Cheng[6], Jie Zhang[1,2]

Peking University[1], Zhongguancun Laboratory[2], HUST[3], UCAS[4], Xiamem University[5], Huawei[6]

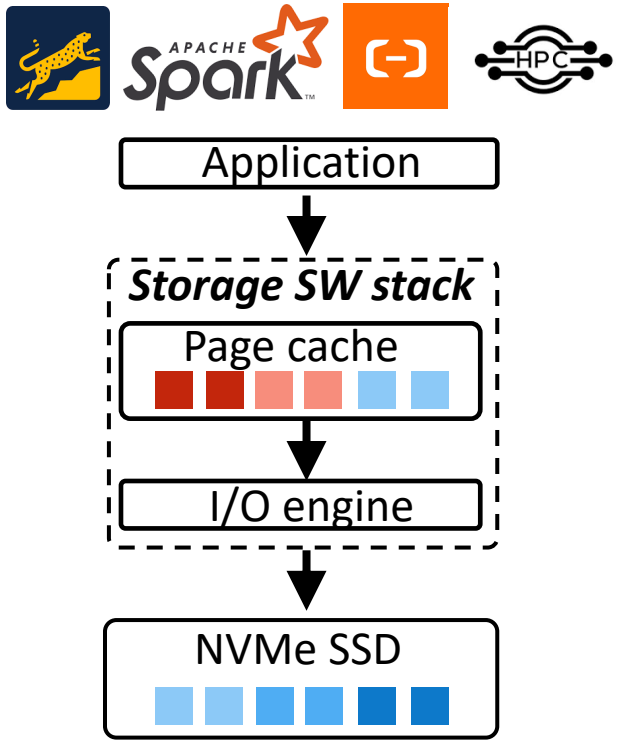
Computer Hardware And System Evolution Laboratory

PEKING UNIVERSITY



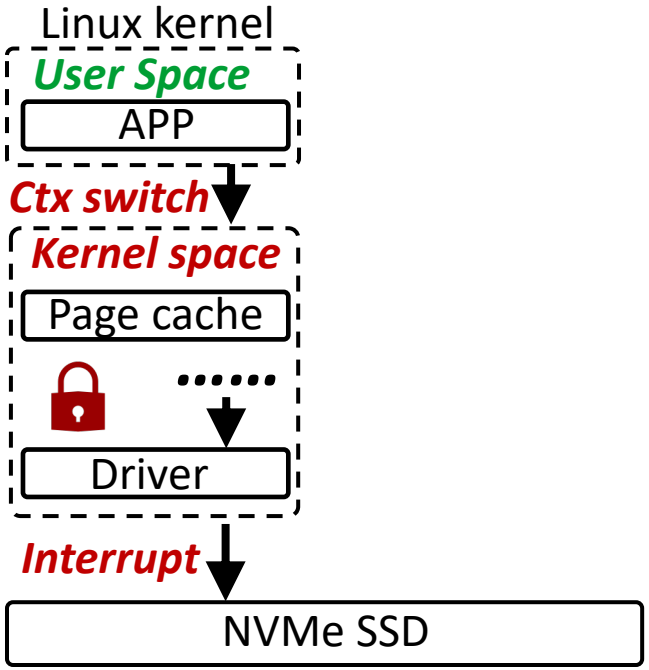
Background: Storage Software Stack

- **Adopted in diverse computing domains**
  - Databases, cloud computing, and HPC
- **Components**
  - **Page cache manager**: buffer hot data in main memory
  - **I/O engine**: concurrently access data residing in SSD
  - Narrow down performance gap between processors and storage devices



Background: Existing Page Cache Manager Design

- **Linux kernel page cache**
  - Kernel space implementation
  - Fails to follow up on SSD performance boost
  - Heavy overhead (e.g., global locking)
- **Hardware trend -> High-performance SSD**
  - High bandwidth: surpass 14GB/s
  - Low latency: ~10us

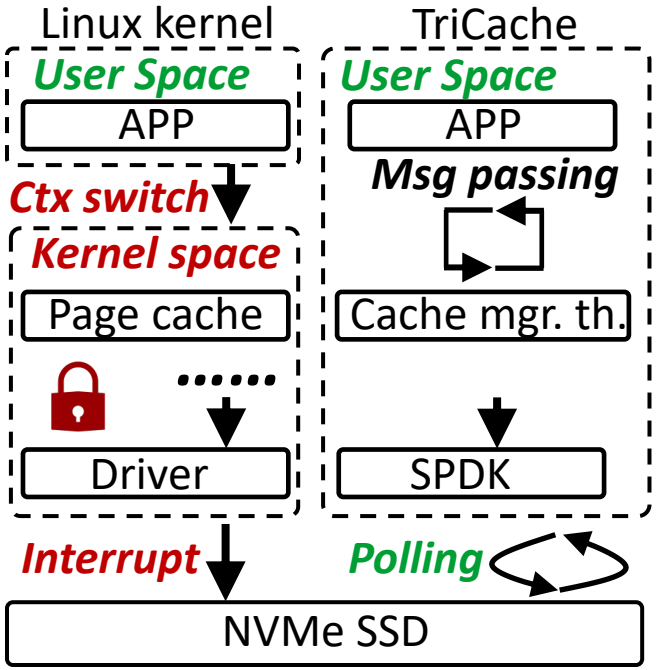


Intel Optane SSD

Samsung PM1743

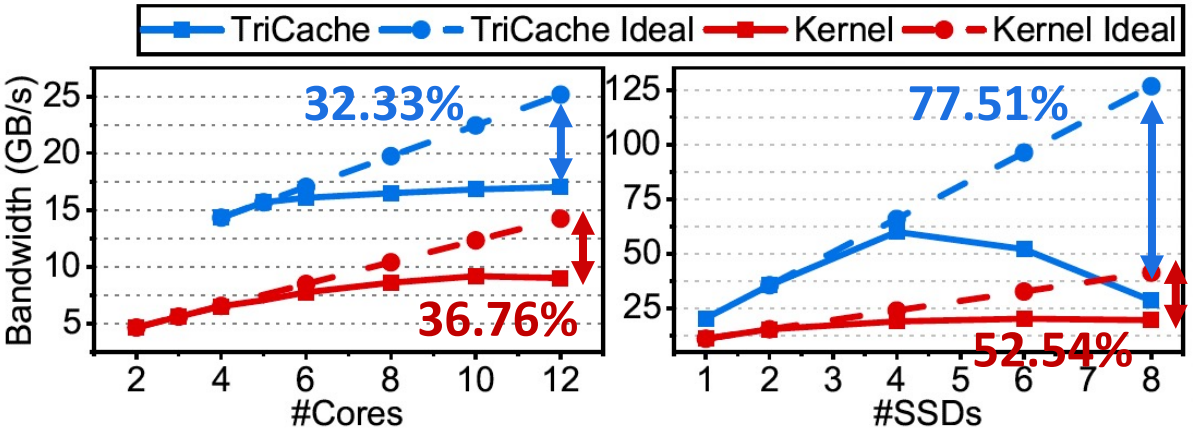
Background: Existing Page Cache Manager Design

- **Linux kernel page cache**
  - Kernel space implementation
  - Fails to follow up on SSD performance boost
  - Heavy overhead (e.g., global locking)
- **User-space page cache (TriCache [OSDI'22])**
  - Efficient user-space SPDK I/O engine
  - Multiple threads manage cache without lock
  - Message passing between cache mgr. and APPs



Preliminary Study

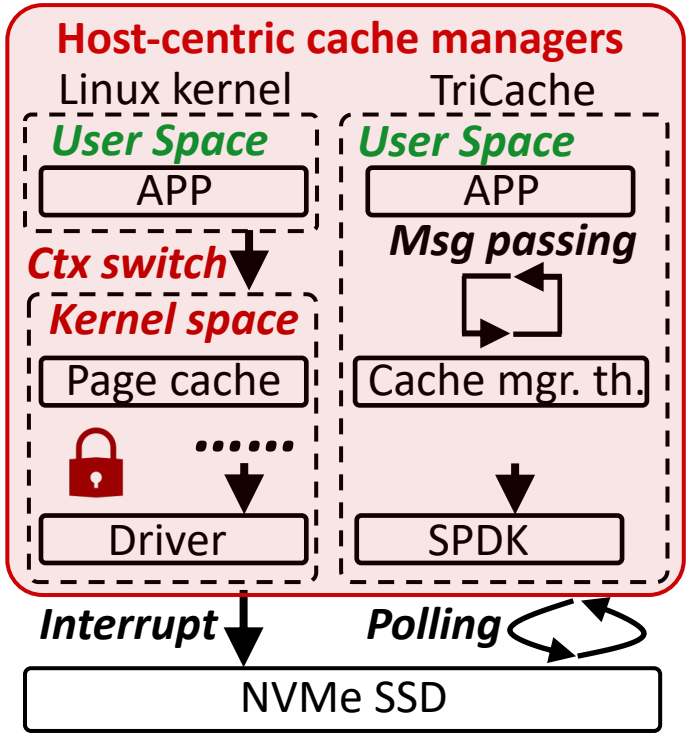
- Performance analysis
  - Macro-benchmark
  - Compare with ideal cases
- Poor scalability with CPU cores
  - Kernel: **36.76%** degradation
  - TriCache: **32.33%** degradation
- Cannot scale with SSDs
  - Kernel: **52.54%** performance gap
  - TriCache: **77.51%** performance gap



(a) Performance of various cores. (b) Performance of various SSDs.

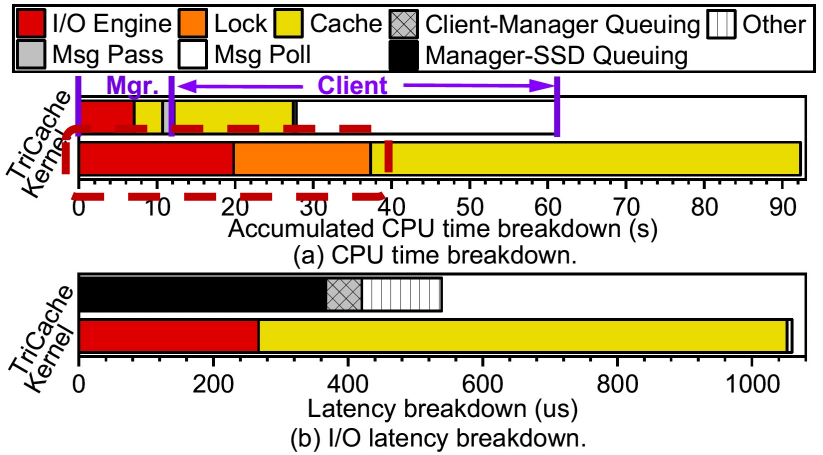
Motivation: Heavy Storage Tax

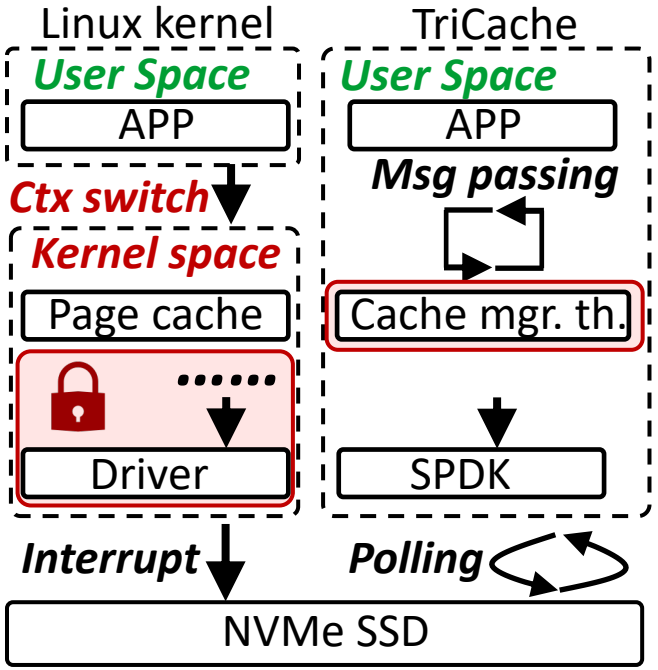
- **Root cause:** host-centric designs
  - Both designs exclusively reside on the host
- **Levy heavy storage tax**
  - CPU tax
  - Communication tax
  - Interference tax



Motivation: Heavy Storage Tax

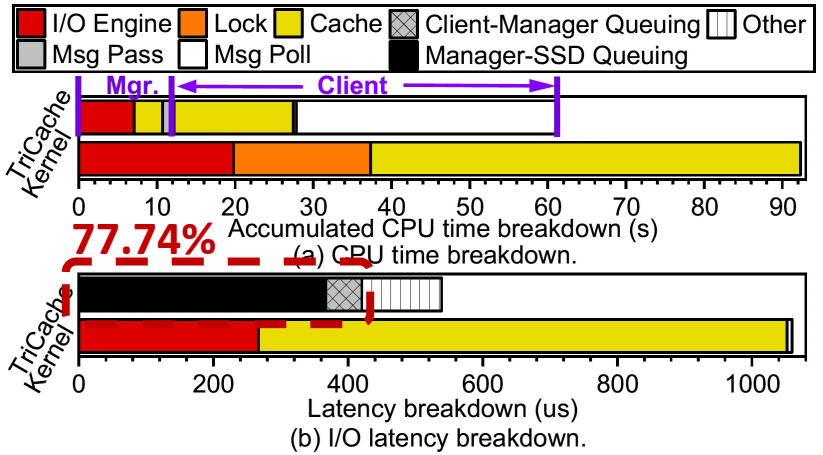
- **Root cause:** host-centric designs
  - Both designs exclusively reside on the host
- **CPU tax**
  - Kernel page cache: **locking (18.98%)** and **heavy I/O engine (21.45%)**
  - TriCache:
    - Dedicate multiple host CPU threads per SSD for cache mgnt.
    - Exacerbate as the number of SSDs scales up
  - **Deprive applications of precious computing resources**

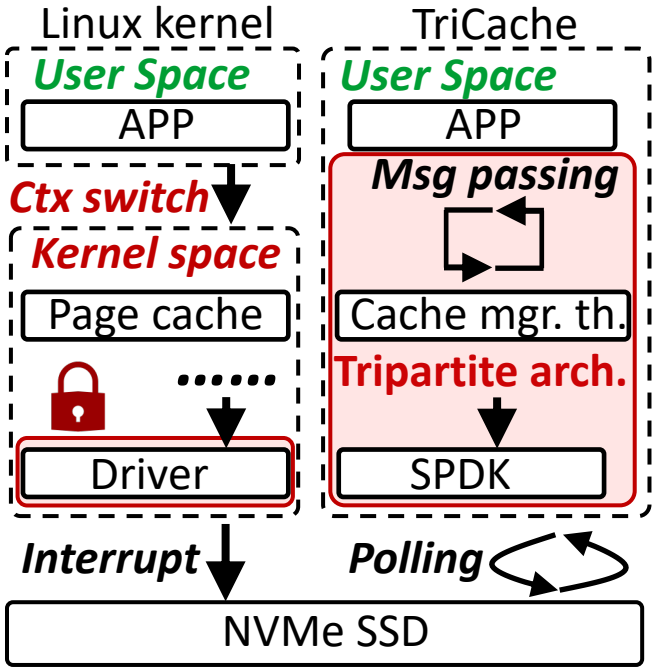




Motivation: Heavy Storage Tax

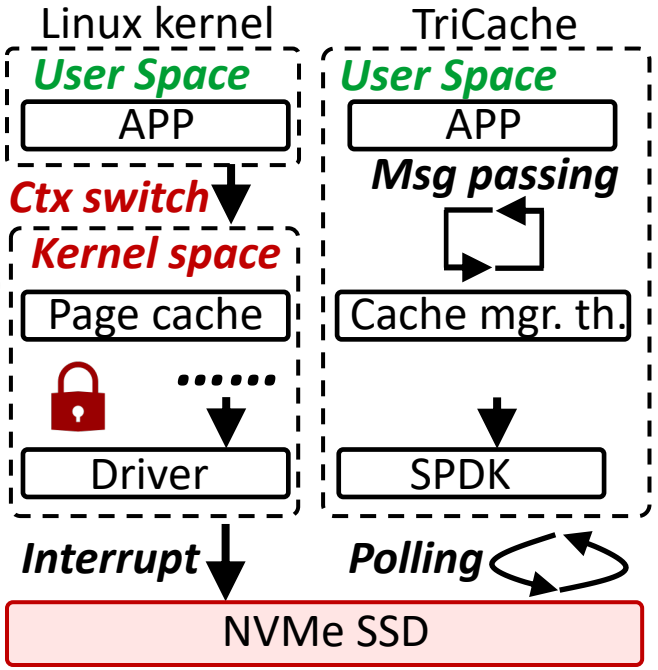
- **Root cause:** host-centric designs
  - Both designs exclusively reside on the host
- **Communication tax**
  - Kernel page cache: **heavy I/O engine**
  - TriCache:
    - **Tripartite structure**: APP <-> Cache mgr. <-> SSD
    - Prolongs communication path
    - **77.74%** queuing latency due to communication





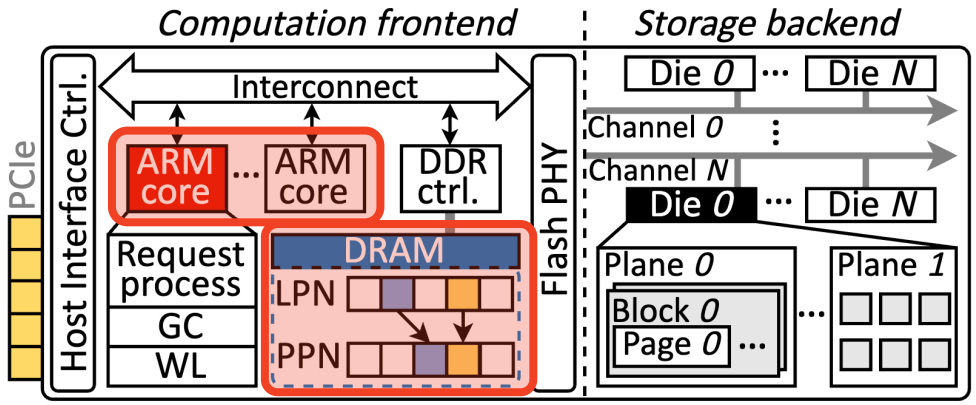
Motivation: Heavy Storage Tax

- **Root cause:** host-centric designs
  - Both designs exclusively reside on the host
- **Interference tax**
  - Host-centric designs cannot detect SSD internal activities (**e.g., GC**)
    - multiple software layers sit between the host-centric manager and the SSD
  - Interference between GC and regular I/O requests
  - **Compromise performance stability**

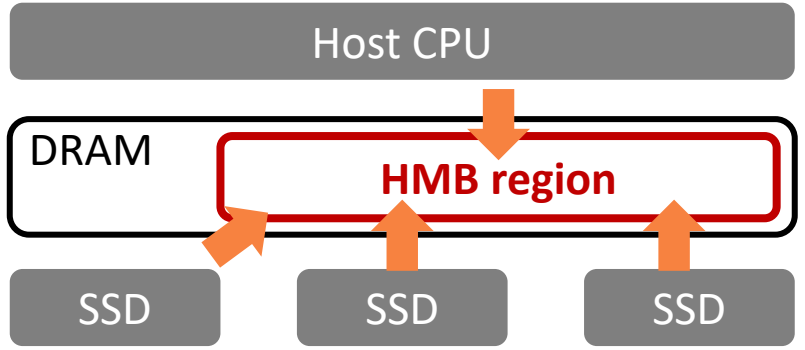


Key insights

- **Emerging computational SSDs**
  - Multi-core ARM processor (4-16 cores)
  - DRAM capacity (4-16GB)
  - Process offloaded tasks from host



- **NVMe host memory buffer (HMB) feature**
  - A DMA-able region in host memory
  - Allows SSDs to directly manage data in the region
  - Ensure rapid data accessibility for applications
  - SSD-controlled page cache with the data cached on the host side



Offload cache management to CSDs!

Our Solution

- Overcome CPU tax by
  - ✓Offloading cache management into CSDs
- Overcome communication and interference taxes by
  - ✓Coordinate software (cache management) and hardware (SSD)

ScalaCache

ScalaCache: Outline

- Overview
- Design
- Evaluation

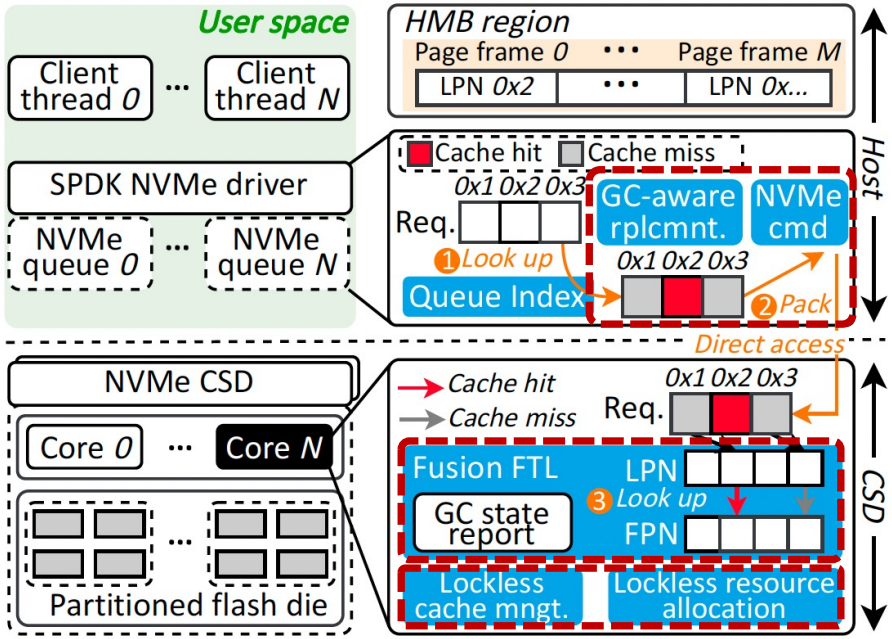
ScalaCache: SW-HW coordination for cache mngt.

➢Overview

- **Lightweight:** high-performance cache index structure
- **Scalability:** lockless cache mngt. and resource allocation

} **Remove CPU tax**

- **Efficiency and stability:** trimmed communication and GC-aware rplcmnt.
  - →**Reduce communication and interference taxes**

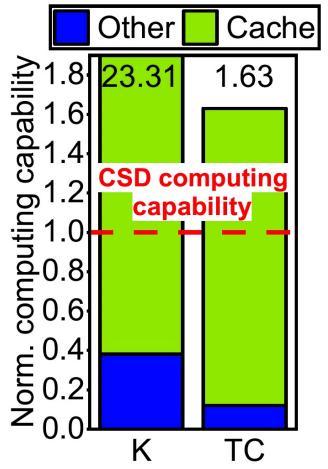


ScalaCache: Outline

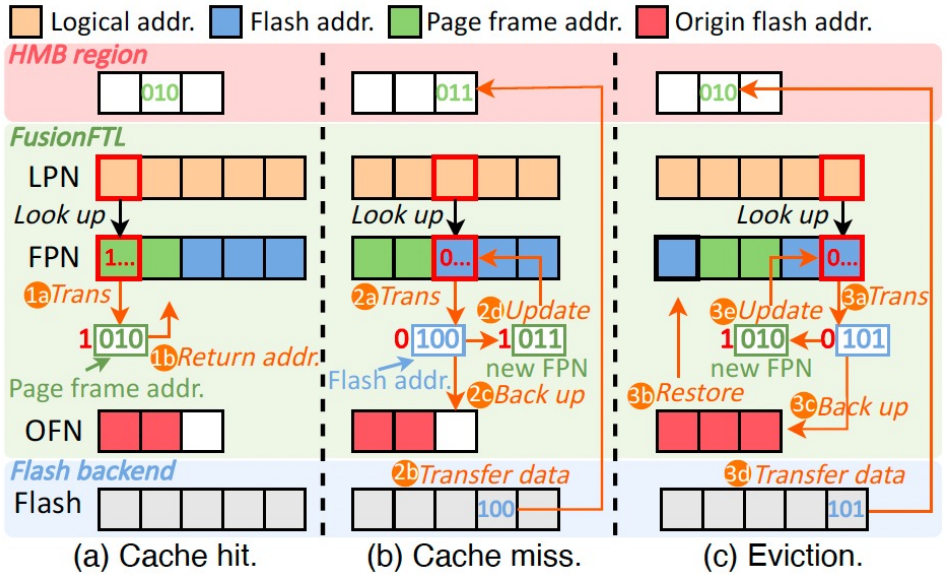
- Overview
- **Design**
- Evaluation

Cache Management Offloading

- **Challenge**: cannot directly offload existing cache management
- **Observation**: CSD internal FTL mapping similar to cache indexing
- **FusionFTL:** consolidates their indirection layers
  - Translate LPN to page frame or flash address based on a flag bit
  - Simplify redundant address translations



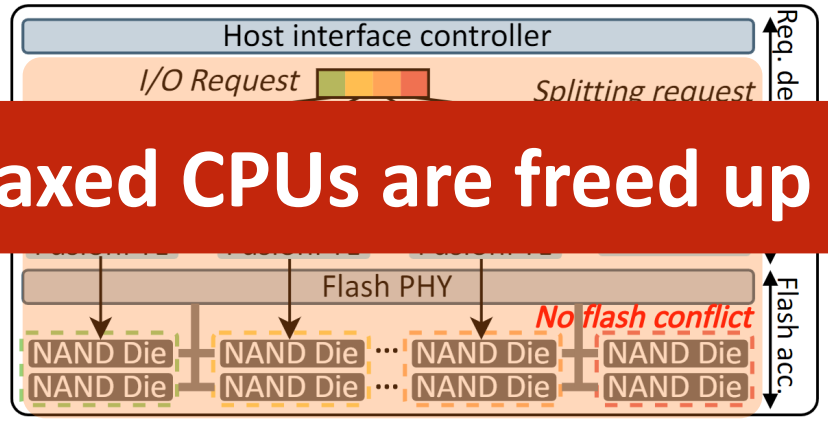
(c) Required computing capability.



(a) Cache hit. (b) Cache miss. (c) Eviction.

Concurrent I/O processing inside CSDs

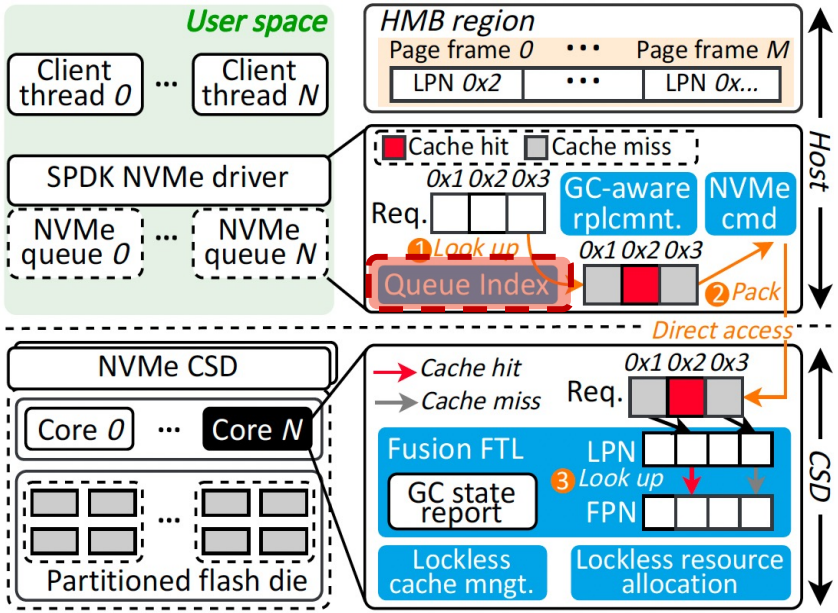
- **Potential bottleneck**: Multiple CSD cores compete for critical resources throughout the I/O path (e.g., FusionFTL, free page frames, and flash)
- **Concurrent processing model within CSD**:
  - **Resource partitioning: address space, page frames, and flash**
    - Assigns them to CSD cores as private resources
  - Split I/O request based on address space division to exploit each CSD core
  - Each core access private resources without contention, which enables lock-free I/O processing



Removing CPU tax: taxed CPUs are freed up for applications' use

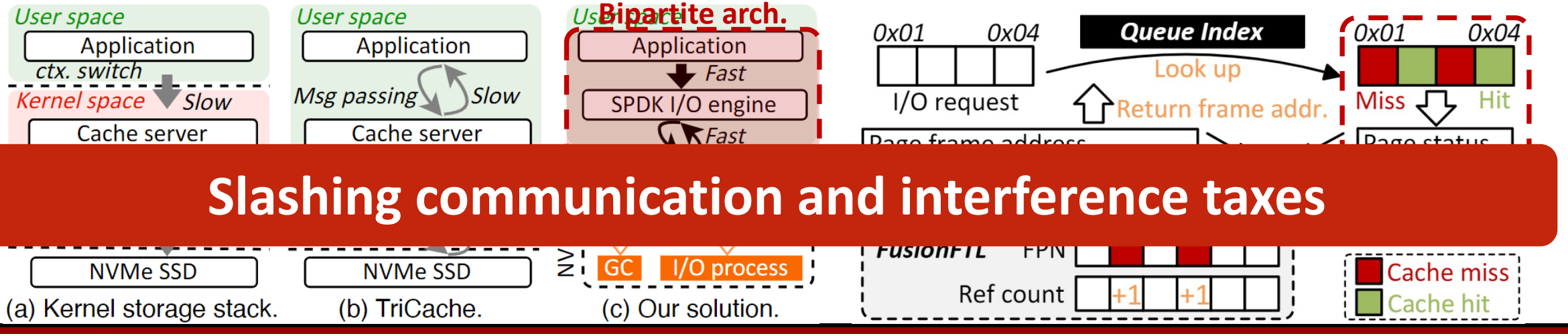
Cache Access on the Host Side

- **Overloading**:
  - Computing capability of the CSD is still limited
  - Processing all requests by CSDs leads to overloading issue
- **Goal:** Avoid overloading CSDs and shorten hit path
- **QueueIndex:**
  - Within each client thread
  - Buffer frame address to accelerate cache lookup
  - Balance the load between the CSD and the host



Coordination between Host and CSDs

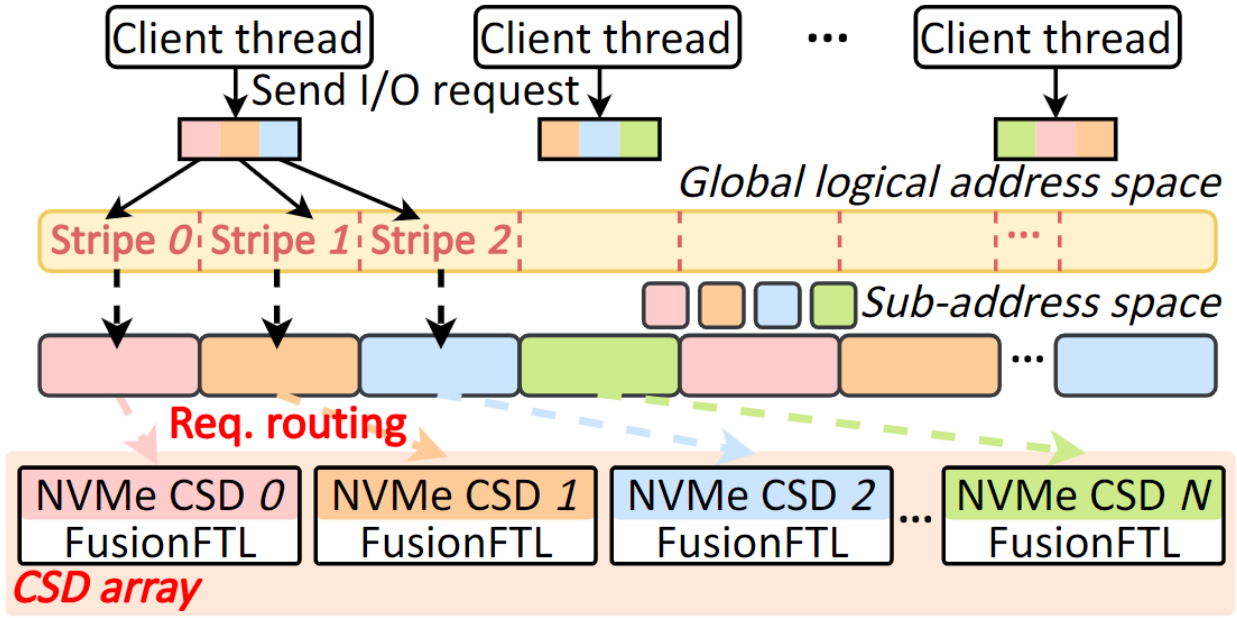
- **Trimmed communication:**
  - By offloading cache mgmt., clients can directly access the cache and flash
    - Transform tripartite architecture into **bipartite architecture**
  - Bundle missing pages with discontinuous addr. into a single NVMe cmd.
- **Reduced GC interference:**
  - GC report: share the internal GC state to host
  - **GC-aware replacement policy** to prioritize the reclamation of clean pages



Slashing communication and interference taxes

Concurrent cache built on a CSD array

- **Goal**: Achieve scalability across multiple CSDs
- **Parallel processing model**: Organizes multiple CSDs into a CSD array
  - Distribute I/O requests to multiple CSDs
  - Leverage multiple CSDs to handle requests concurrently
  - Aggregate computing power of multiple CSDs to deliver scalable perf.



ScalaCache: Outline

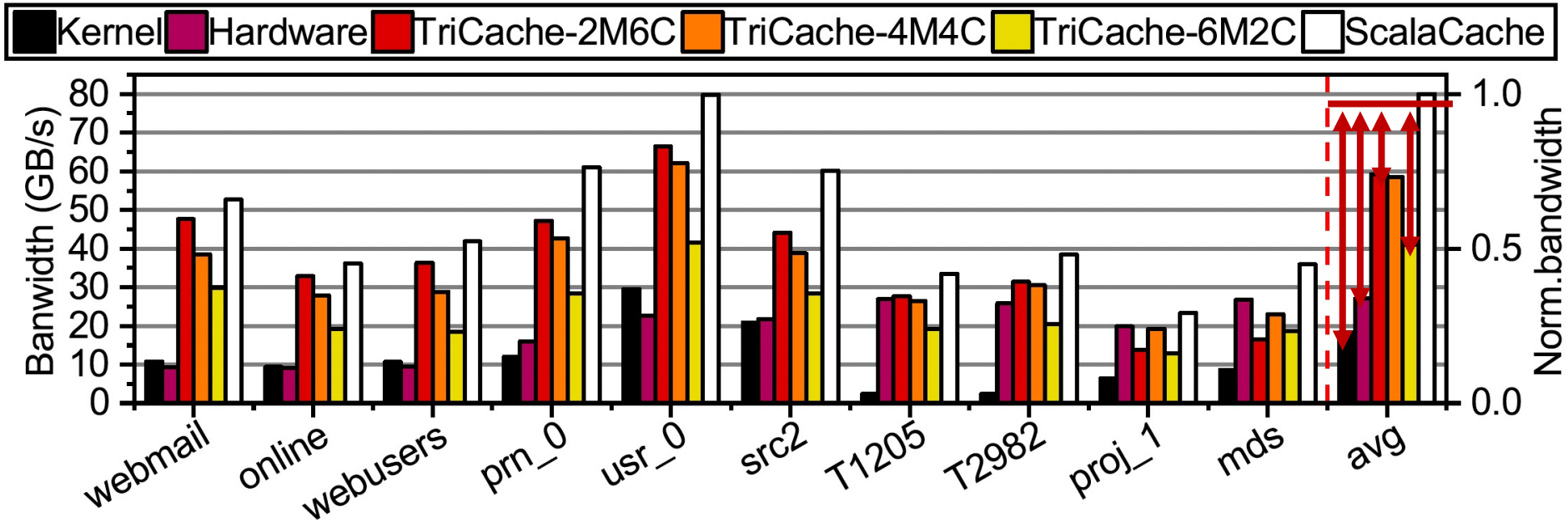
- Overview
- Design
- **Evaluation**

Evaluation: Setup

- **Implementation:**
  - Build our ScalaCache design based on FEMU emulator
- **Platform:**
  - **Kernel**: traditional page cache implemented in Linux kernel
  - **TriCache**: state-of-the-art user-space cache management
  - **Hardware**: simply offloads cache management into CSDs
  - **ScalaCache**: hardware-software coordinated user-space cache mgmt.
- **Real-world workloads –** MSR, FIU, and Tencent block trace

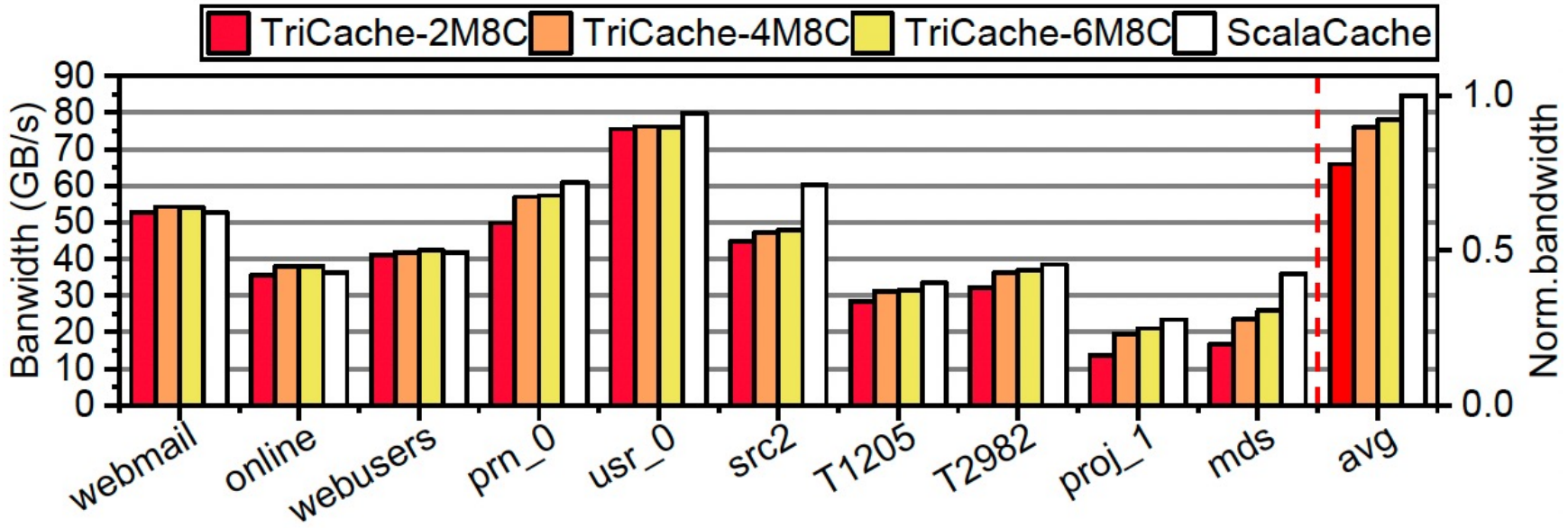
Evaluation: Overall

- Bandwidth comparison with fixed 8 host CPU cores
- **5.12×** and **1.95×bandwidth improvement** compared to Kernel and Hardware
- **35.30%** and **94.78% bandwidth improvement** compared to TriCache which employs 2 and 6 manager threads (i.e., TriCache-2M and TriCache-6M)
  - Frees up taxed CPU for more client th. and benefits from lightweight design



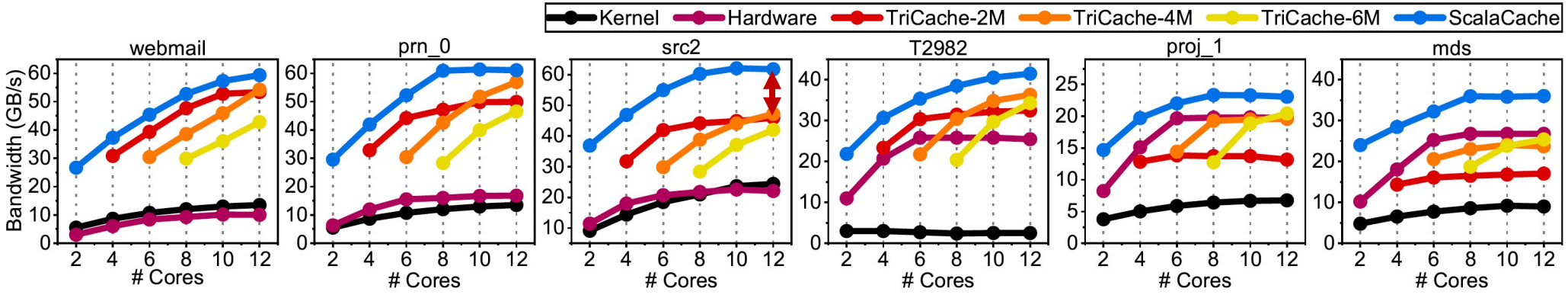
Evaluation: Overall

- Bandwidth comparison with fixed 8 client threads
  - Relax the number of cores in TriCache (i.e., 8 cores for client threads) while allocating extra cores as cache manager threads
- ScalaCache still outperforms TriCache (e.g., outperforms 2M8C by **29%**)
  - More manager threads in TriCache increase the communication cost
  - ScalaCache removes this cost



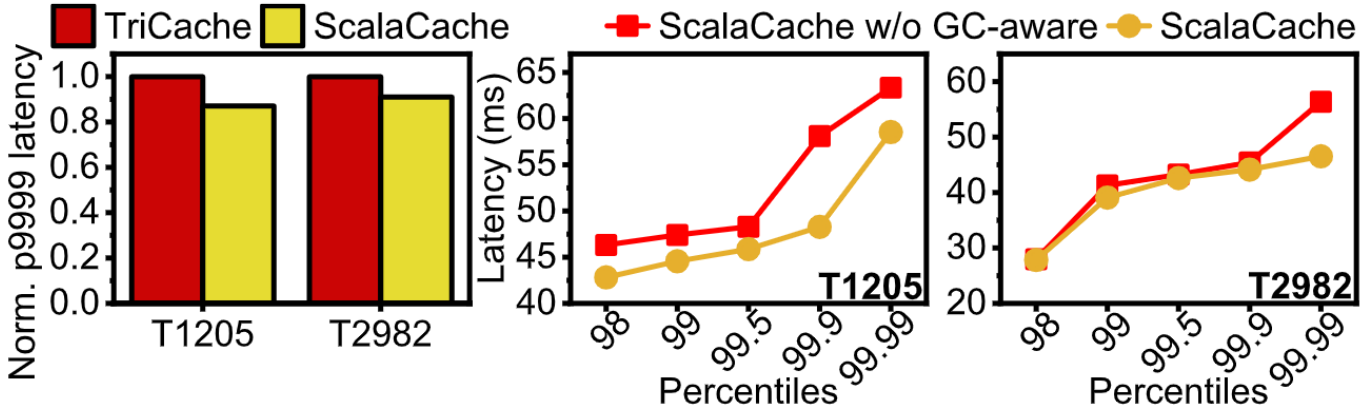
Evaluation: Scalability with host CPU cores

- ScalaCache consistently shows improved scalability in all workloads
  - E.g., surpasses TriCache-2M by **35.17%** in src2
- Due to lightweight and lockless designs, including
  - Lightweight cache management
  - Lockless resource allocation framework
  - Concurrent I/O processing



Evaluation: Tail Latency

- Compare tail latency in write-intensive workloads
  - **11%** 99.99th latency reduction compared to TriCache
  - Unattainable with host-centric cache manager designs like TriCache
- Breakdown:
  - Evaluate the tail latency of ScalaCache with and without GC awareness
  - E.g., **17.44%** 99.9th latency in T1205
  - Software-hardware coordinated fashion alleviates GC impact



(a) Tail latency comparison. (b) Improvement of GC-aware replacement policy.

Conclusion

- **Challenges**
  - **Host-centric cache manager designs**
  - **Heavy storage taxes**: CPU tax, communication tax, and interference tax
- **Key insights**
  - Cache management offloading and software-hardware coordination
- **ScalaCache designs**
  - Lightweight cache mgmt in CSD + Trimmed communication + GC avoidance

→ **Successfully reduce heavy storage taxes**

Thanks for attending!
Q&A

Computer Hardware And System Evolution Laboratory

CHASELab

PEKING UNIVERSITY

HUAWEI