

uCache

A Customizable Unikernel-based IO Cache

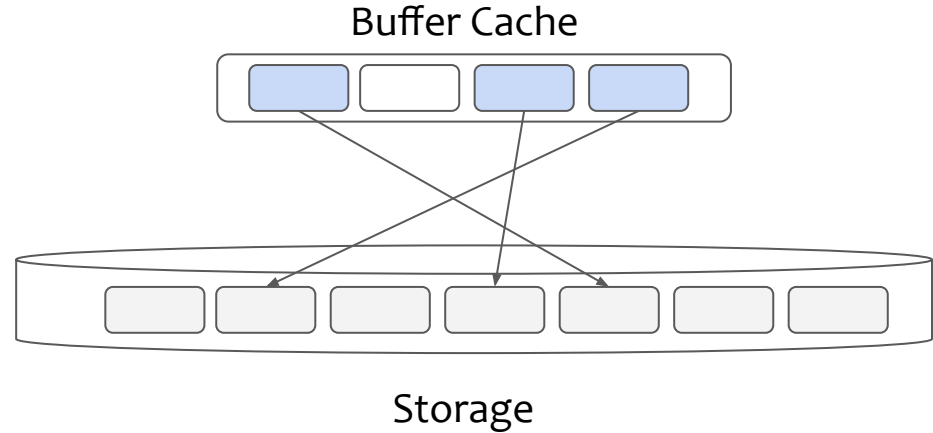
Ilya Meignan--Masson, Masanori Misono, Viktor Leis,
Pramod Bhatotia



- Basic problem: Map locations on storage to memory pointers

- Goals:

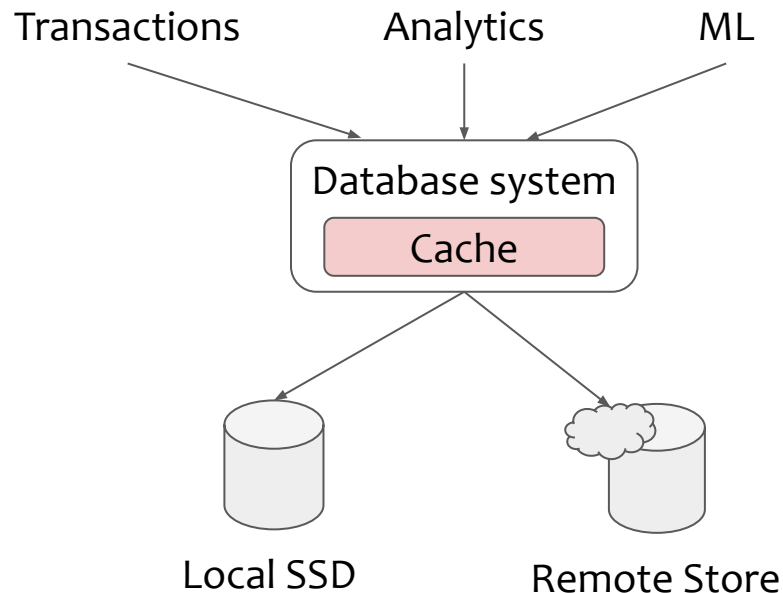
- ① hit: as fast as memory
- ② miss: as fast as storage



Why is it challenging?

Caching in the cloud

- Many workloads
 - OLTP/OLAP
 - Graphs
 - ML
- Many IO backends
 - NVMe SSDs, block and object stores
 - 10s of GiB/s in bandwidth



What do database systems expect from the cache?

Requirements

- ③ Customizable to the workload
 - Guarantees (e.g. transactional safety)
 - Arbitrary buffer size
 - Replacement policy (eviction, prefetching)
- ④ Interact with local and remote IO backends

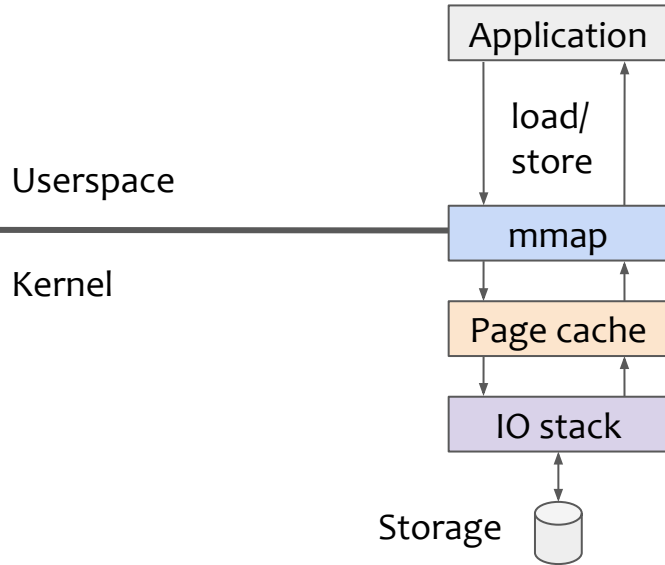


- ① hit: as fast as memory
- ② miss: as fast as storage

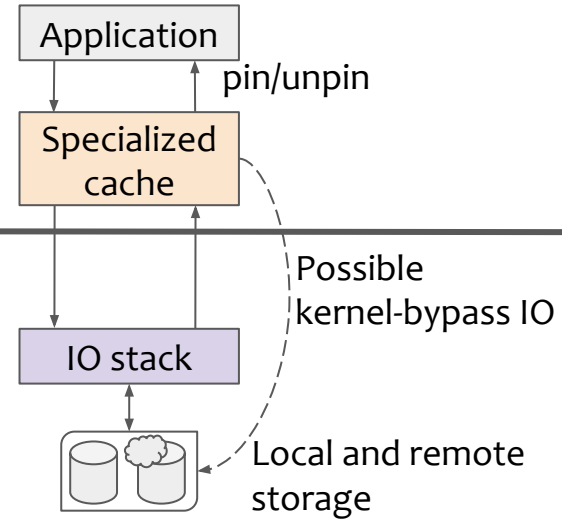
What are the options for caching?

Existing approaches

Traditional: OS-level caching



State-of-the-art: Explicit userspace caching



Existing approaches - comparison

	Performance		Customizable		Simplicity
	① Hit	② Miss	③ Workloads	④ IO backends	
mmap	++	~ (*)	-	--	++
Explicit userspace cache <i>e.g., vmcache [SIGMOD'23]</i>	~ (*)	++	++	++	--

(*) can be fixed

Can the OS provide a simple, flexible and performant IO cache?

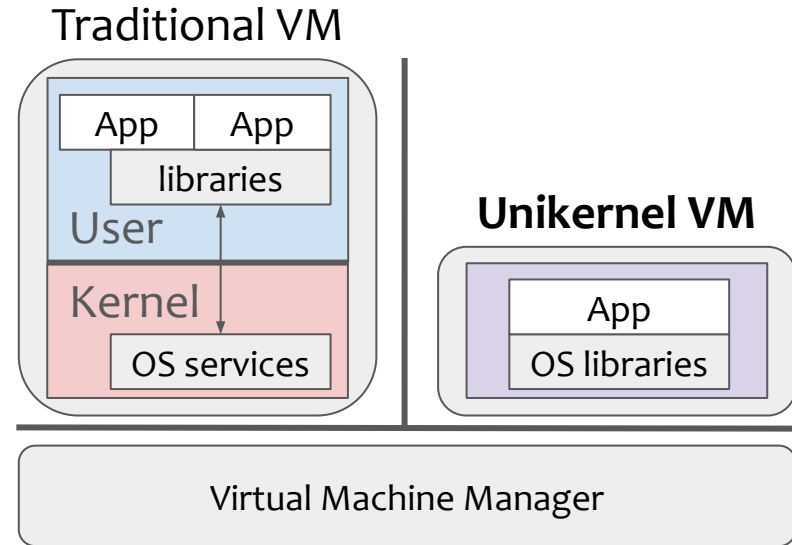
A customizable memory-mapped IO caching framework

Design Goals:

- Flexibility, adapt to the multiple workloads and storage solutions
- Simplicity, the application decide the level of control
- Performance, enable applications to leverage high-performance IO devices

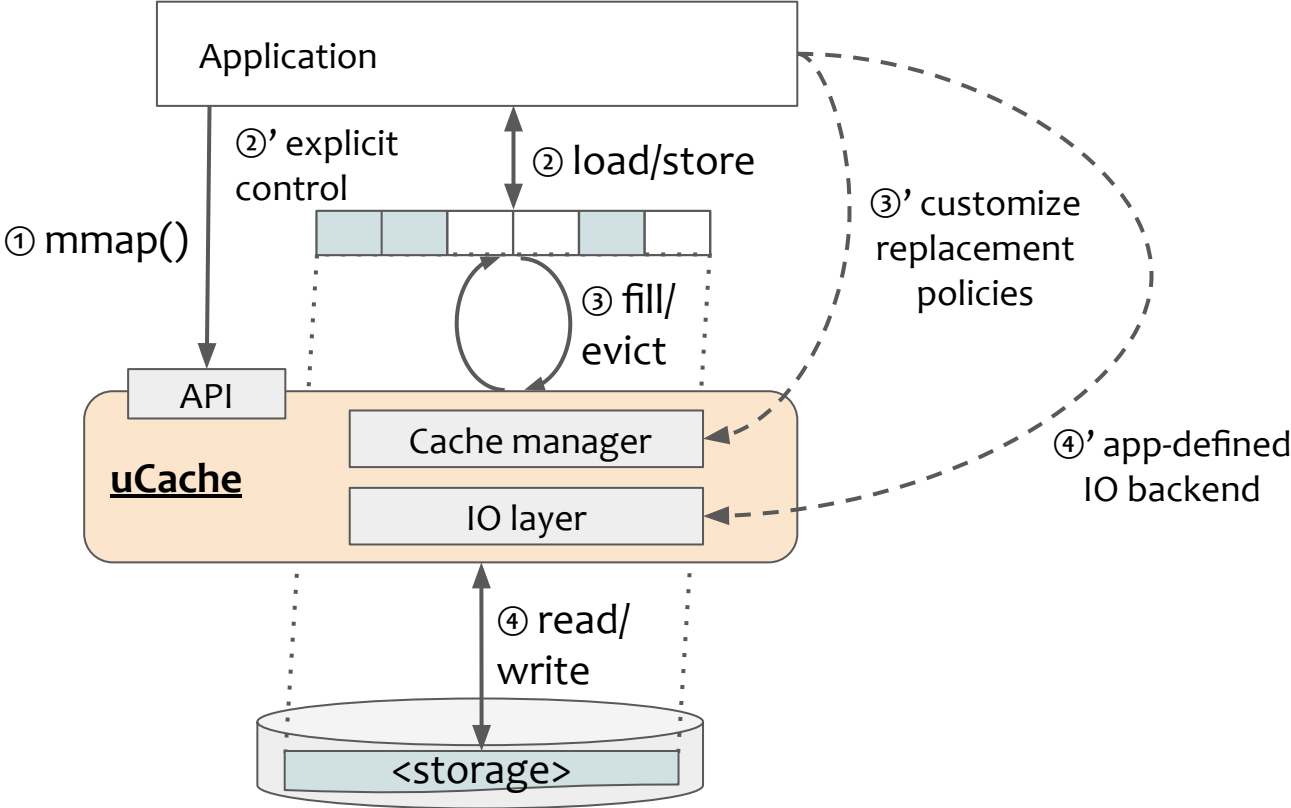
Key idea: A unikernel architecture

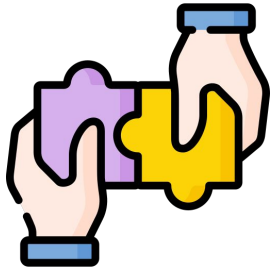
1. **Single** address-space
2. **Library** OS
3. Built into a **VM image**



The unikernel architecture enables OS/App co-design

Overview

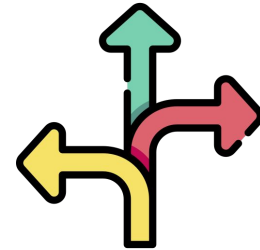




Application/OS
coordination



Scalable cache
operations



Flexible IO
abstraction

Challenge #1: Application/OS coordination

**Completely
transparent**

*e.g., mmap +
load/store*

**Customizable
policies**

*e.g., eviction,
prefetching, ...*

**Explicit
control**

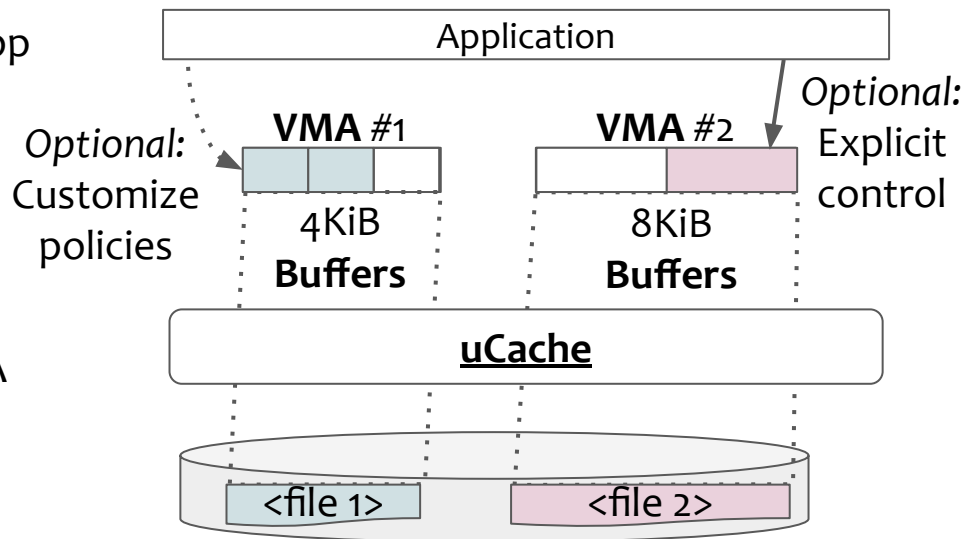
*e.g., pin/unpin, evict,
writeback, ...*



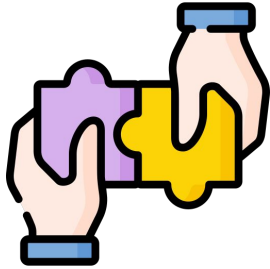
How to organize the **communication** between the app and the OS?

Solution #1: uCache memory abstractions

- Virtual memory area (**VMA**)
 - Shared between the OS and the app
- **Buffer**
 - Granularity for primitives
- **Customizable** policies:
 - Attach function callbacks to a VMA



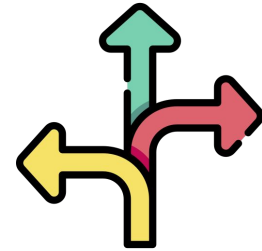
uCache enables OS/App coordination through shared VMA and Buffer abstractions



Application/OS
coordination



Scalable cache
operations

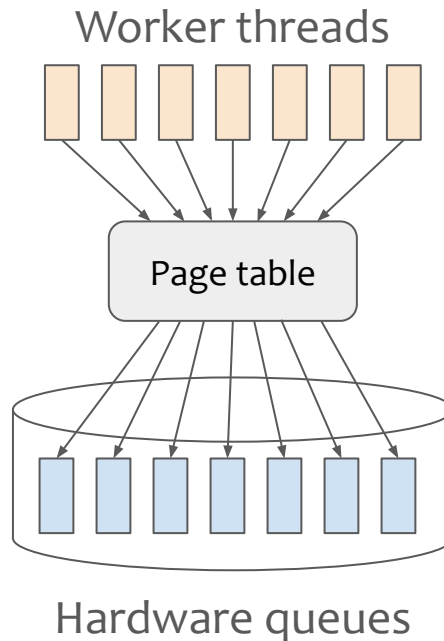


Flexible IO
abstraction

Challenge #2: Scalable cache operations

Lock-free operations

*Avoid synchronization
for sparse accesses*



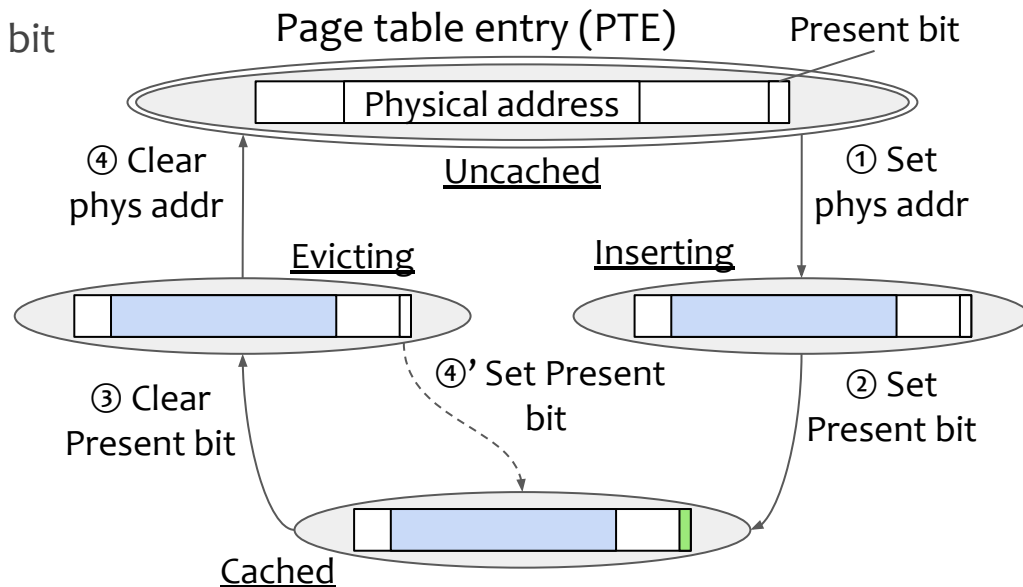
Batch eviction

*Mitigate costly TLB
invalidation*

How to modify the page table in a **lock-free** manner with **batching**?

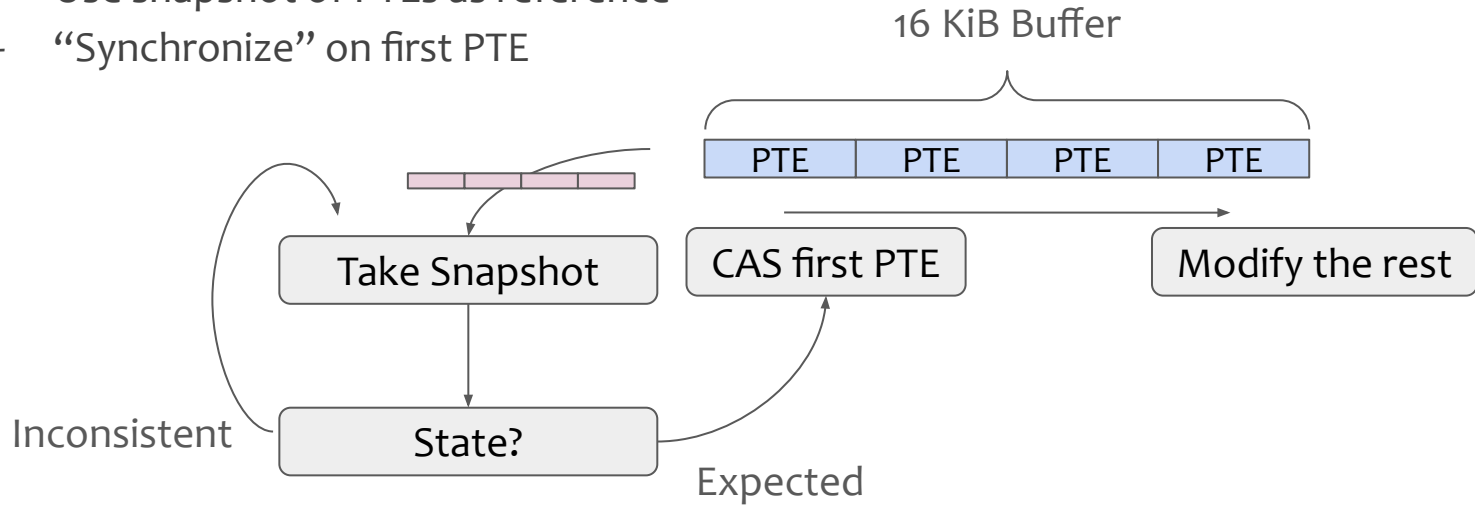
Solution #2: Optimistic cache operations

- **Two-step** CAS operations
 - Separate phys addr and present bit
- During eviction:
 - Accumulate candidates
 - Accesses **cancel** eviction

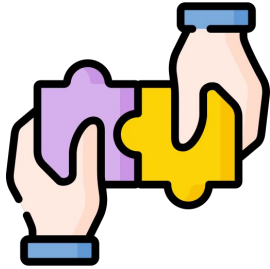


Solution #2: Optimistic cache operations

- **Big pages**
 - Use snapshot of PTEs as reference
 - “Synchronize” on first PTE



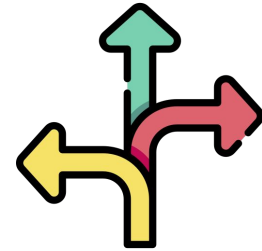
uCache uses **optimistic operations** to limit synchronization even when batching



Application/OS
coordination

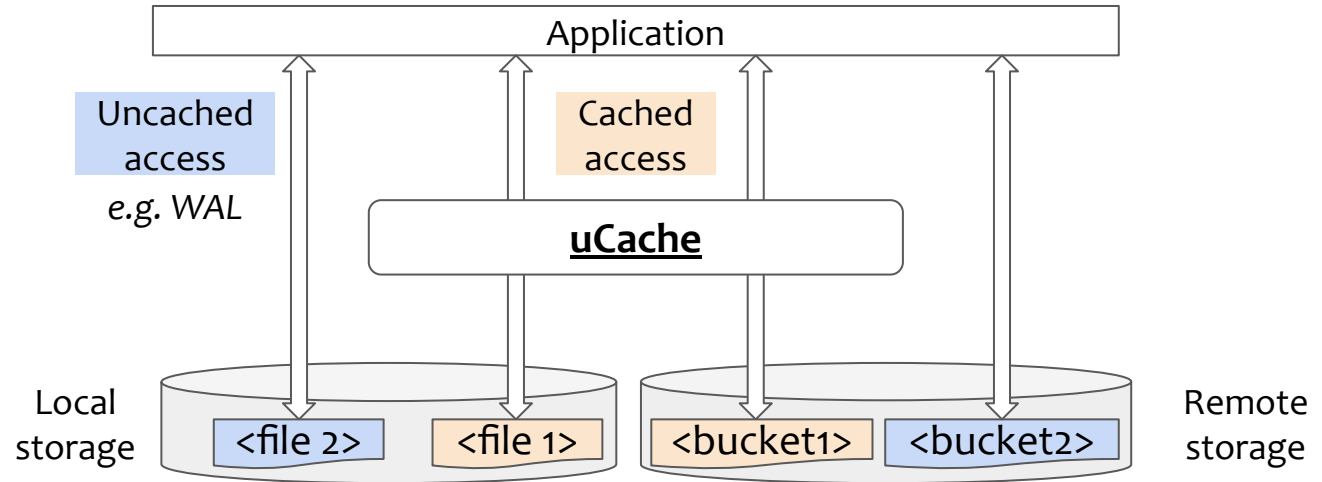


Scalable cache
operations



Flexible IO
abstraction

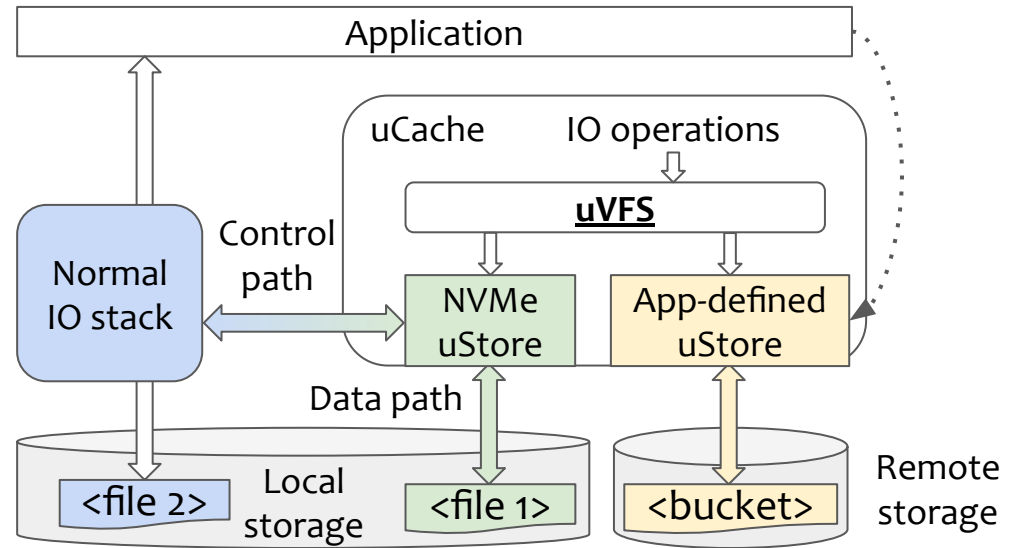
Challenge #3: Flexible IO abstraction



How to enable flexible storage access to applications?

Solution #3: uCache IO abstractions

- **uVFS: Abstracts IO stack**
 - dispatches to uStores
 - App-defined uStores
- **NVMe uStore**
 - performant data path
 - compatible control path



uCache enables performant and customizable IO without hindering compatibility

Implementation

- Built on the **OSv** unikernel
 - ~2K LoC
- Cache manager
 - Default: **CLOCK** replacement
 - Physical memory allocation: **LLFree**[ATC'23]
- IO layer
 - **ext4**-compatible NVMe uStore



Evaluation

1. How performant are cache management operations?
- ~~2. How much footprint does uCache introduce?~~
- ~~3. What is the performance of the IO layer?~~
4. Use cases
 - a. Is uCache a compelling replacement to mmap?
 - b. Can uCache be used for the buffer manager of database systems?
 - c. Can uCache be used to improve the performance of accessing data files?

More evaluation in the paper

Experimental setup

Setup:

- AMD EPYC 9654 (96C), 768GiB RAM
- Kioxia CM-7 NVMe SSD (PCI 5.0)

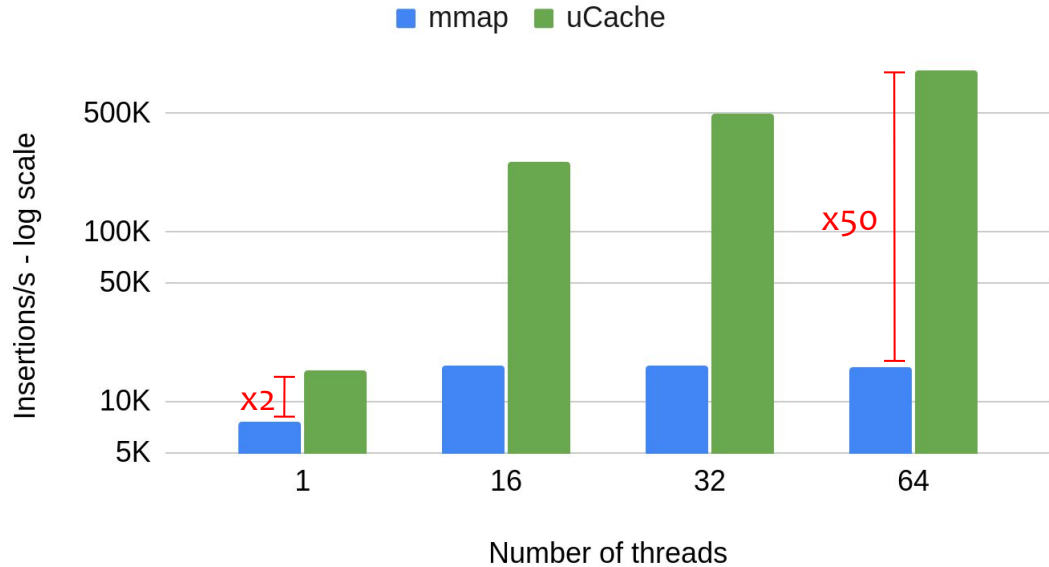
Variants:

- mmap in Linux VM
- Specialized userspace cache
- uCache

Higher is better ↑

Insertions per second

- focus on out-of-memory case
- includes evictions (no writes)



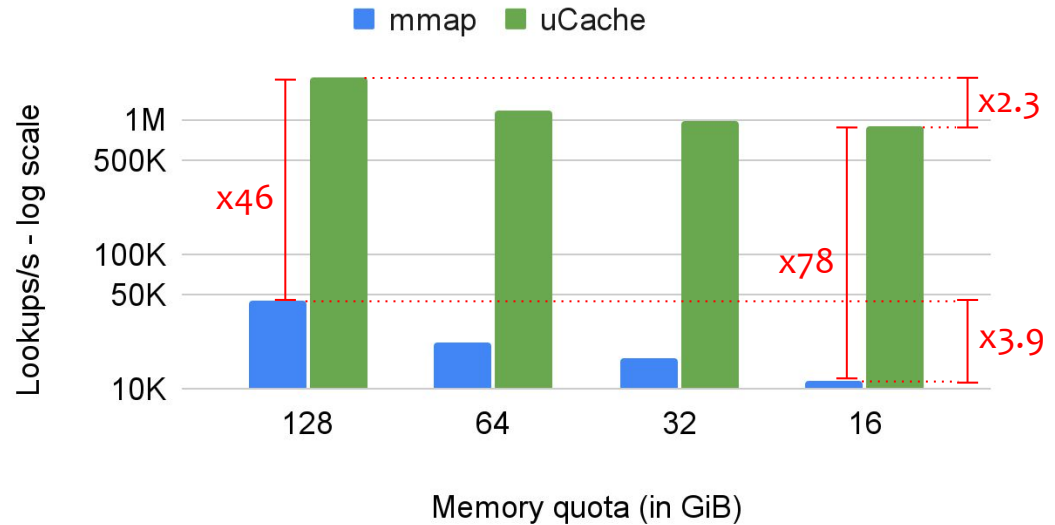
uCache scales linearly with the number of threads

End-to-end performance

Lookups per second:

- random access pattern
- 200GiB data size
- 64 threads

Higher is better ↑



uCache is a compelling replacement to mmap performance-wise

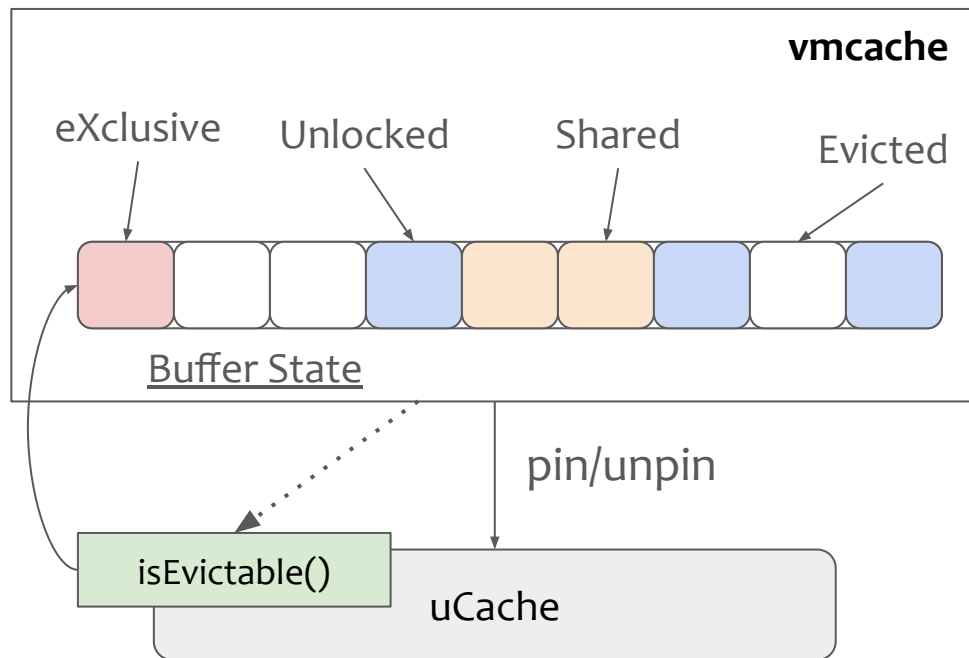
Use case: Database systems buffer management

Buffer cache: vmcache[SIGMOD'23]

Baselines:

- With POSIX (madvise)
- With kernel module (exmap)

- Ported to uCache
#LoC: 100 (+), 400 (-)

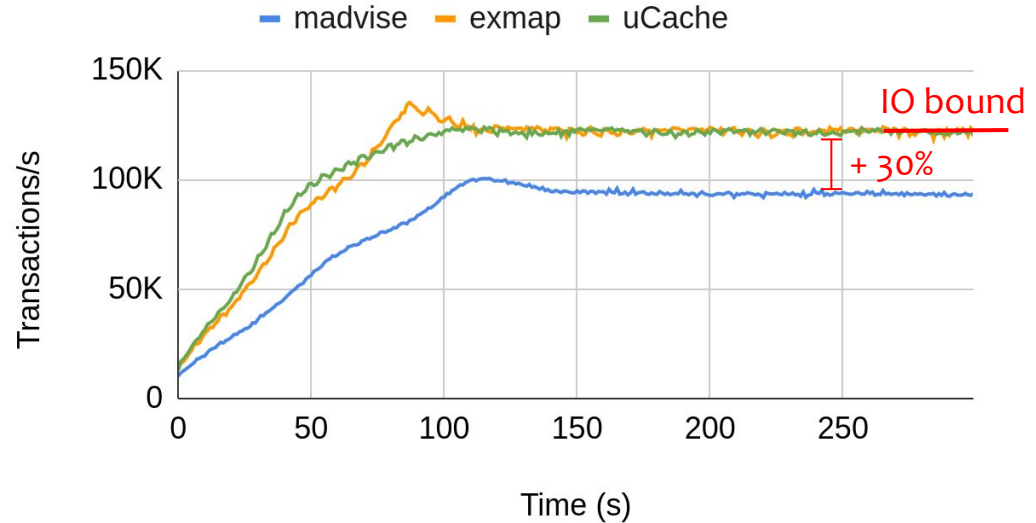


Use case: Database systems buffer management

TPC-C:

- 5000 warehouses (~1TiB of data)
- cache 128GiB
- 64 threads

Higher is better ↑



uCache can integrate complex application semantics without incurring overheads

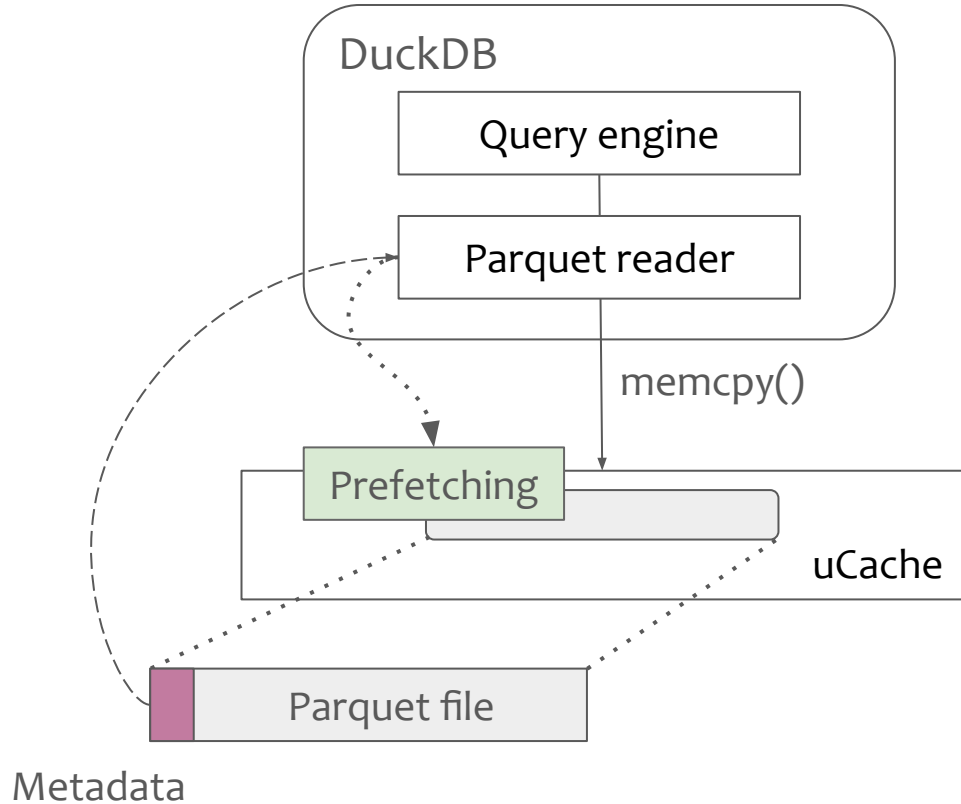
Use case: External file caching

DuckDB: Parquet caching

Baseline:

- With Linux block cache

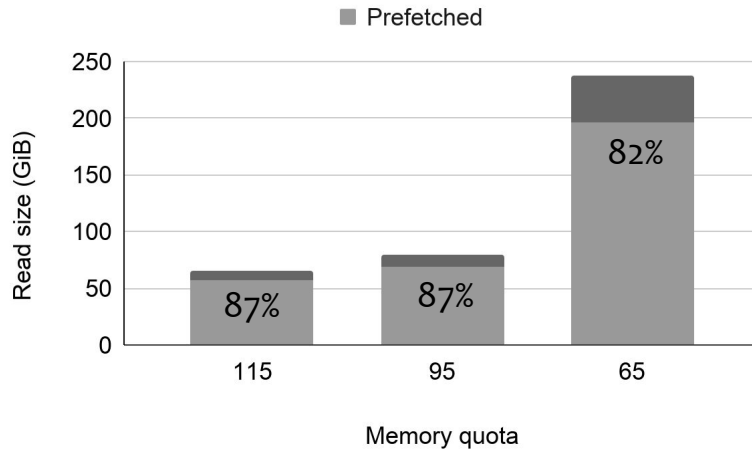
- Ported to uCache
#LoC: 230 (+), 35 (-)



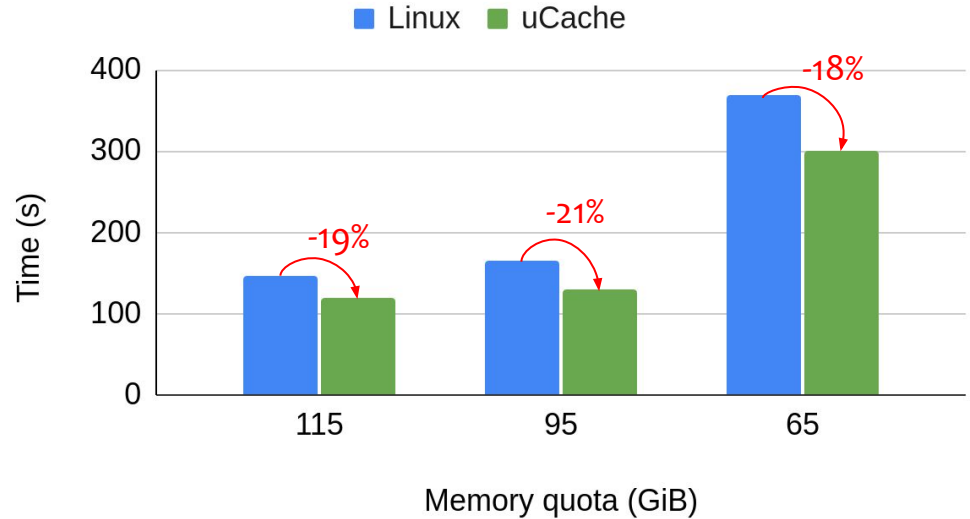
Use case: External file caching

TPC-H:

- Workload: all queries sequentially
- Scale Factor 300 (~95G datasize)
- 64 threads



Lower is better ↓



Can the OS provide simple, flexible and performant memory-mapped IO caching?

uCache

- Aims to be convenient like OS-level caches while performant and flexible
- Customizable and scalable caching framework through the unikernel architecture
- Outperforms mmap and customizable like specialized caches

Try it out !



[TUM-DSE/uCache](https://github.com/TUM-DSE/uCache)

ARTIFACT EVALUATED	ARTIFACT EVALUATED	ARTIFACT EVALUATED
 USENIX	 USENIX	 USENIX
AVAILABLE	FUNCTIONAL	REPRODUCED