Towards DBMS-aware Memory Management in Unikernels

Martin Lindbüchl
Advisor: Ilya Meignan--Masson
Examiner: Prof. Dr. Pramod Bhatotthia
Systems Research Group
https://dse.cit.tum.de/

18.11.2024 – 18.03.2025

Context: Movement Towards Cloud

The provider hosts databases for customers (DBaaS)

↓

Feasible to choose custom operating system (OS)

↓

Move towards optimized OSs for database systems

Motivation: DBMS/OS Mismatch

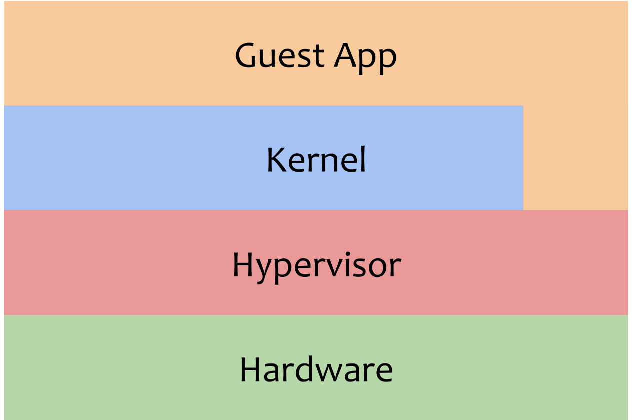
OS Abstractions

- OS hides kernel level functionality from application
  - Scheduling
  - Memory Management

→ OS generality conflicts with DBMS requirements

State-of-the-art

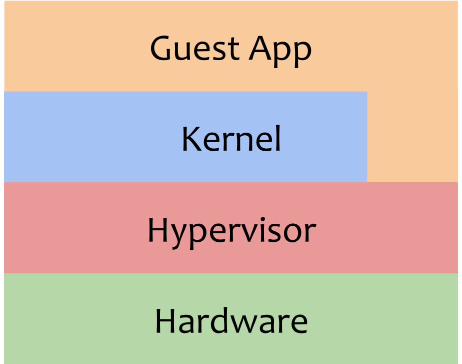
	Efficient	Flexible	Complex
User-space systems	no	yes	no
Kernel modules[1]	yes	no	yes
Dune-based systems[2]	very	yes	very



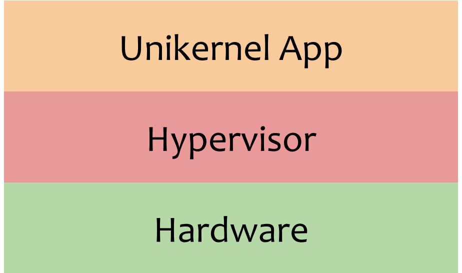
Traditional VM

[1] ExMap[SIGMOD'23], Kreon[TOS'21]
[2] Aquila[EuroSys'21], Libdbos[SIGMOD'25]
Figure inspired by unikraft

Unikernels



Traditional VM



Unikernel

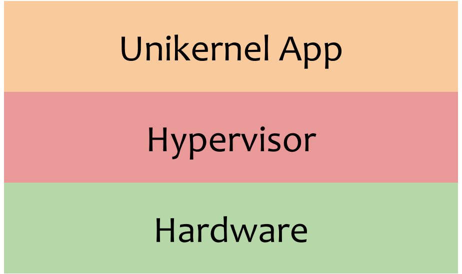
- Heavyweight
- Address space / privilege isolation
- Large codebase

- Lightweight
- No address space / privilege isolation
- Small codebase

Figure inspired by [unikraft]

Unikernel-based Memory Management

	Efficient	Flexible	Complex
User-space systems	no	yes	no
~~Kernel modules~~	yes	no	yes
~~Dune-based systems~~	very	yes	very
Kernel API	yes	very	no



Unikernel

Figure inspired by unikraft

Outline

- ~~Motivation & Background~~
- Overview
- Design & Implementation
- Evaluation
- Conclusion

Problem statement

How can we design a **unikernel's** virtual memory (VM) subsystem to allow **efficient** and **general** DBMS/OS co-design?

Proposal

A new **VM subsystem** design exposing a **kernel API** tailored to DBMS requirements using a **unikernel architecture**

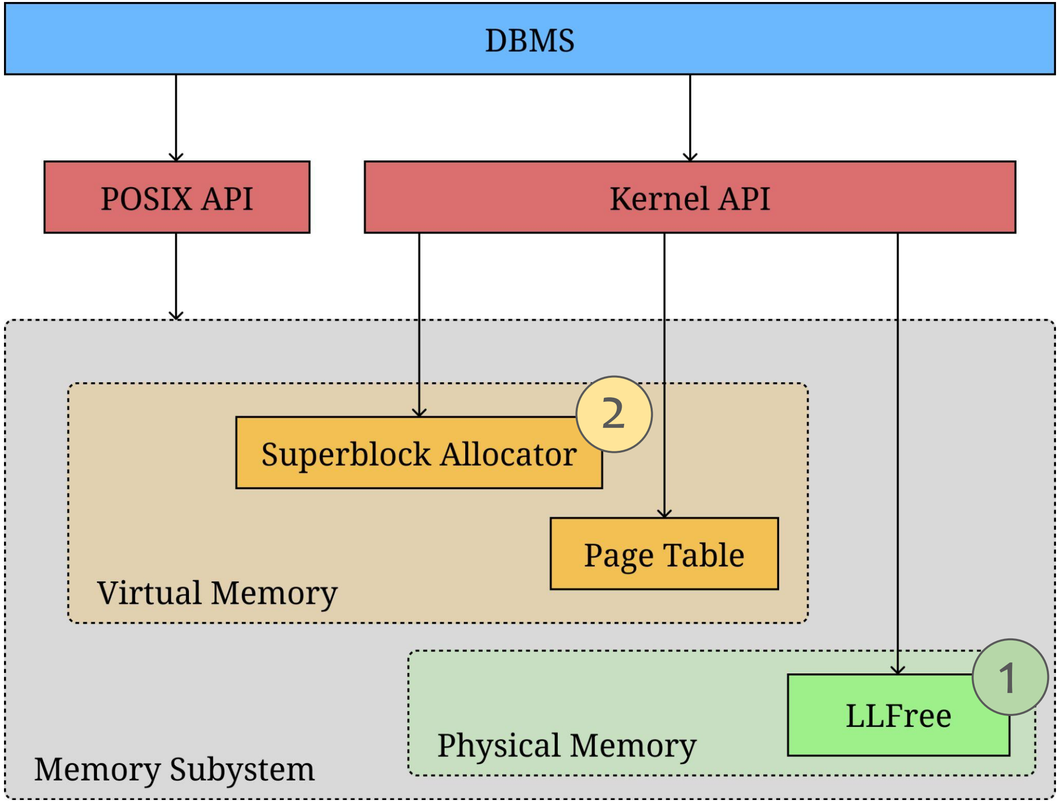
System design goals:

- Flexibility
- Scalability
- Compatibility
- Memory efficiency

Outline

- ~~Motivation & Background~~
- ~~Overview~~
- Design & Implementation
  - Overview
  - System design
  - Algorithms
- Evaluation
- Conclusion

Overview: Unikernel/DBMS Co-design



Physical Memory Allocation

1 LLFree[1]

Properties

- Scalable
- Lockfree
- Anti-fragmenting

Challenges

- Early allocation
- Initialization
- Memory contiguity

Kernel API: Physical memory allocation, Memory metrics

[1] LLFree[ATC'23]

VM Management

② Superblock Allocator

Properties

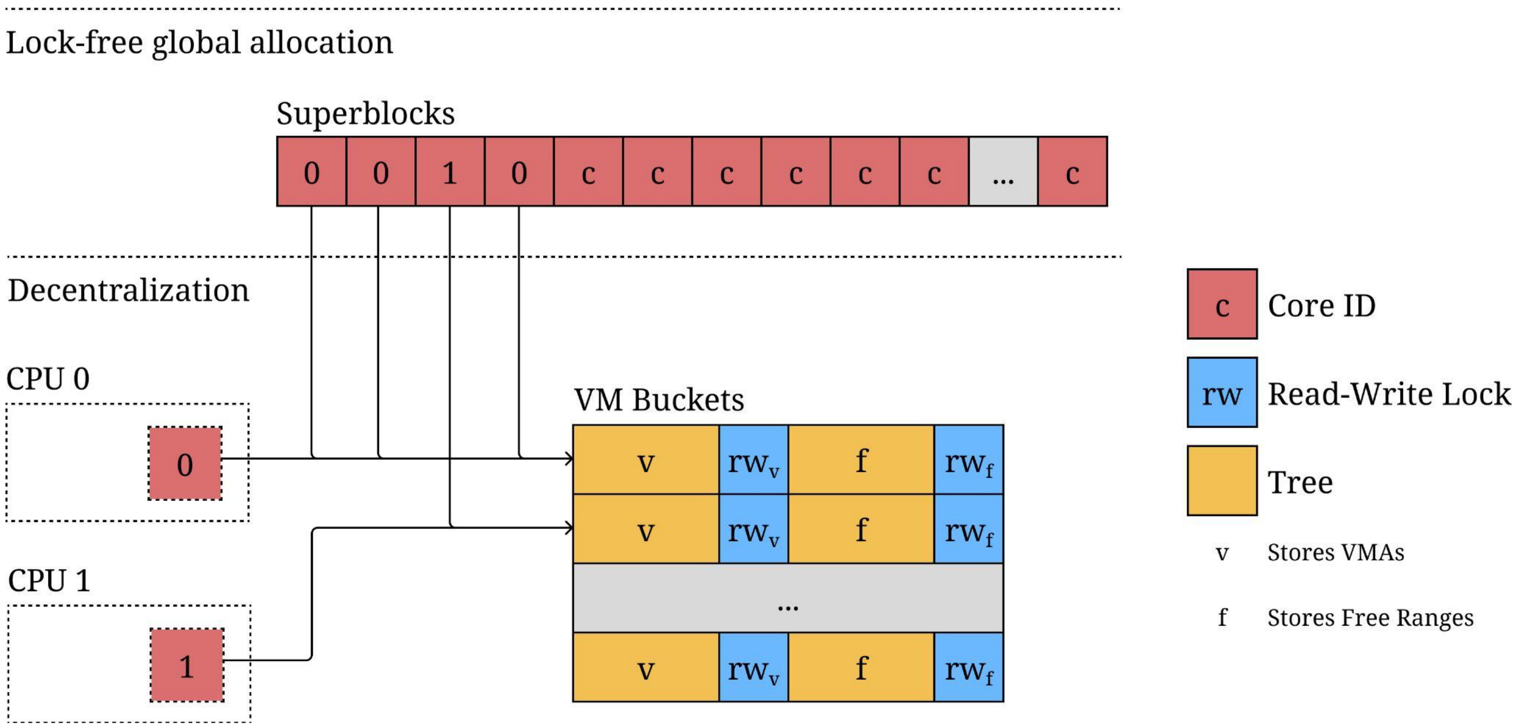
- Scalable
- Core local

Challenges

- Optimistic superblock allocation
- Core distribution implications

Kernel API: VM allocation, VM area manipulation, page table synchronization

Superblock Allocator



Outline

- ~~Motivation & Background~~
- ~~Overview~~
- ~~Design & Implementation~~
- Evaluation
  - Microbenchmarks
  - Macrobenchmarks
- Conclusion

Evaluation: Microbenchmarks

- Boot Time
- Memory Footprint
- VM Scalability
- Bulk (de-)allocation (Best case)
- Pipeline deallocation
- Randomized deallocation (Worst case)

Evaluation: Microbenchmarks

Competitors

- Linux virtual machine +jemalloc
- Original OSv version
- Optimized OSv version +jemalloc

Bulk (Best case)

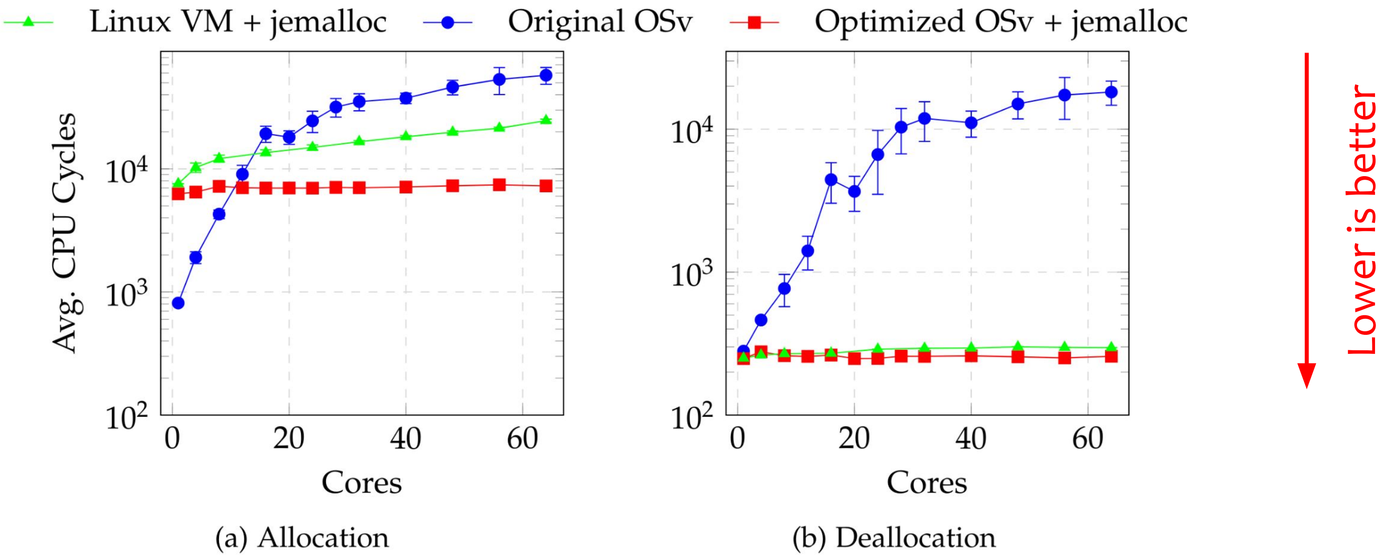
- Simulate allocation pattern of arena allocators
- Measure throughput of overall memory subsystem

Random (Worst case)

- Approximate highly shared workloads
- Constructed to highlight worst case performance

Microbenchmarks: Bulk

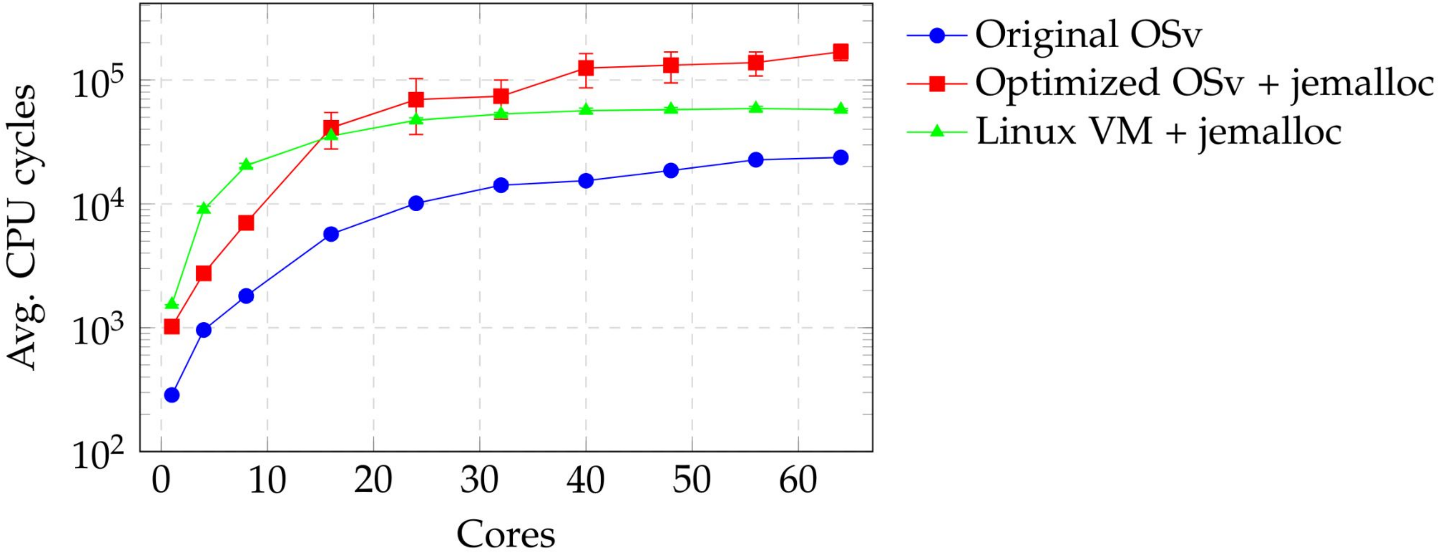
Bulk allocation | 200,000 pages | Core local allocation + deallocation



Superior allocation scalability

Microbenchmarks: Random

Random deallocation | 20,000 pages | Barely core-local deallocations



Limited scalability for highly shared workloads

Evaluation: Macrobenchmarks

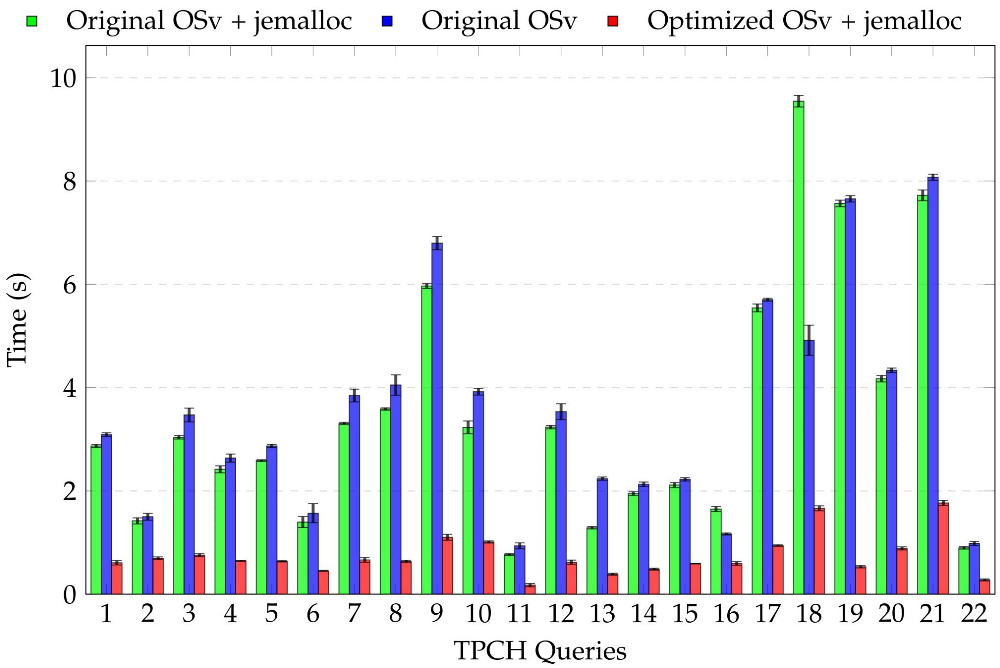
Competitors

- Linux virtual machine +jemalloc
- Original OSv version
- Original OSv version +jemalloc
- Optimized OSv version +jemalloc

TPC-H benchmark on DuckDB

- How does our implementation perform for database workloads?

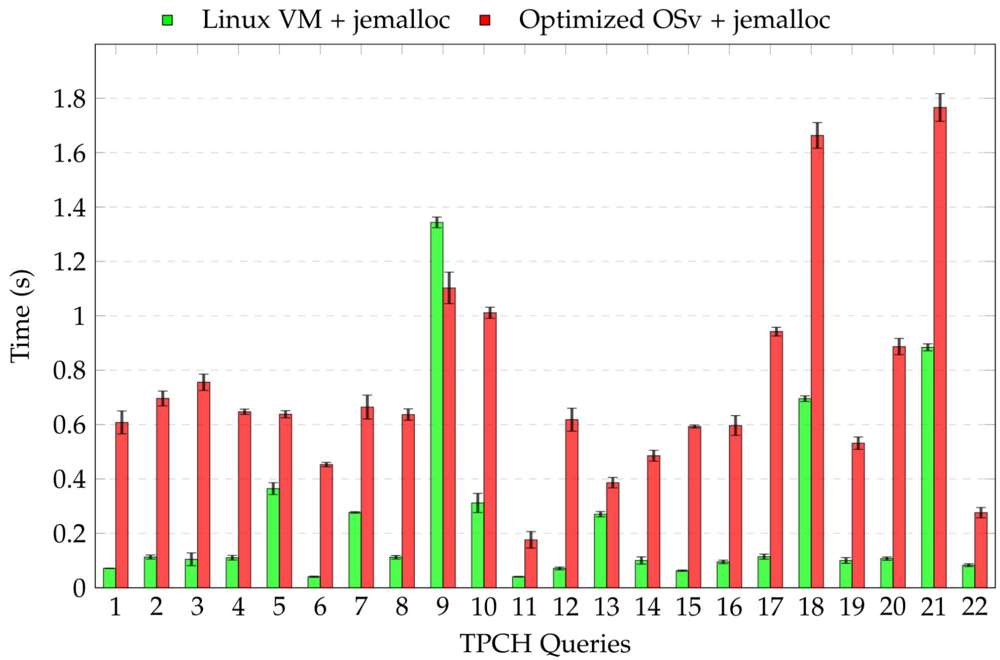
Database workloads: OSv comparison



Lower is better

Superior OLAP performance for all TPC-H queries

Database workloads: Linux comparison



Lower is better

Narrowing OLAP performance gap to Linux

Conclusion

Proposition: Design a new VM subsystem for unikernels to efficiently expose kernel functionality to DBMS

Result: We created a platform for general DBMS/OS co-design that offers

- **Flexible** use of kernel functionality due to Kernel API
- Linux **compatibility** due to maintained POSIX interfaces
- A **scalable** memory subsystem due to unikernel optimizations
  - Narrowing performance gap to linux without Kernel API utilization

Implementation and Benchmarks:
https://github.com/Martin-Lndbl/osv_controller

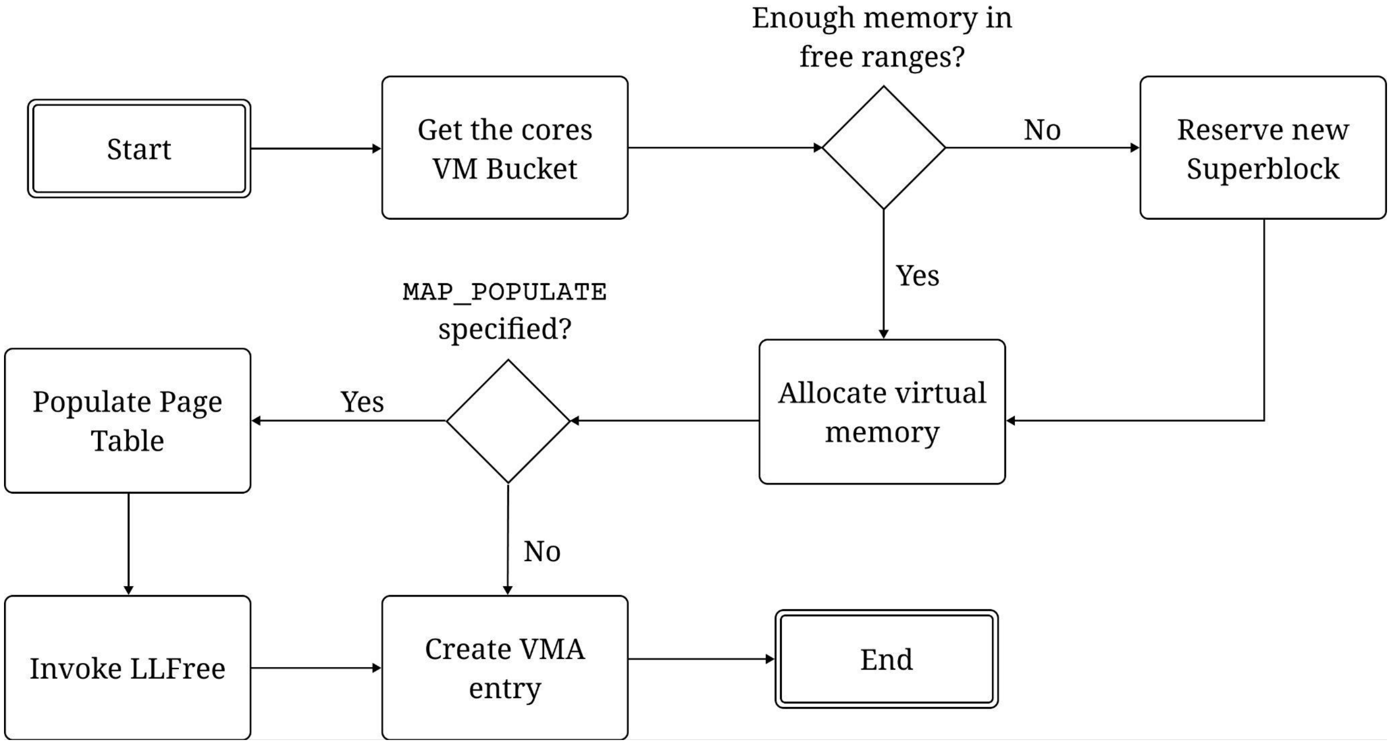
Questions?

Backup

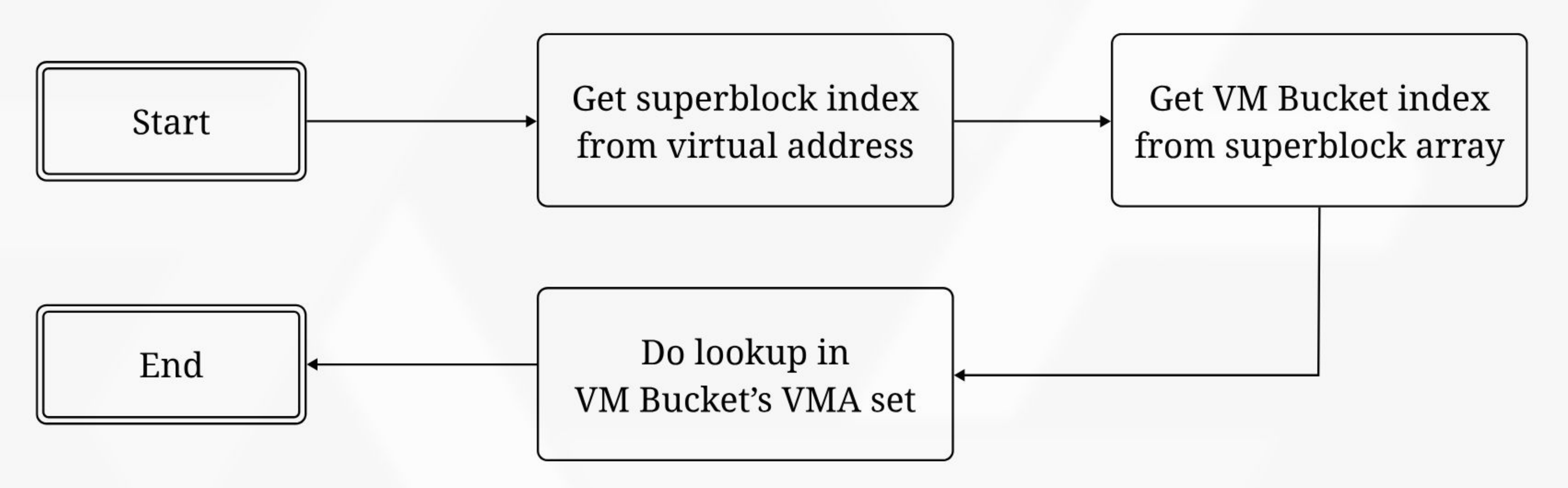
Kernel API table

Interface	Description
void* frames_alloc(order)	Allocate physically contiguous memory of the specified order and return the linearly mapped address.
void frames_free(addr, order)	Free the physically contiguous memory represented by the linearly mapped address.
u64 stat_free_phys_mem()	Get the amount of free physical memory.
u64 stat_total_phys_mem()	Get the amount of total physical memory.
void flush_tlb_all()	Flushes all TLBs in the system.
mutex& pt_high_mutex()	Returns the mutex for higher page table levels for modifications of linear mappings.
rwlock_t& vma_lock(addr)	Returns the read-write lock for an address range containing the VMA intersecting with the specified address.
rwlock_t& free_ranges_lock(addr)	Returns the read-write lock for the memory range set containing the address.
void insert(vma)	Insert the VMA into the kernel's internal state.
void erase(vma)	Erase the VMA from the kernel's internal state.
optional<vma*> find_intersecting_vma(addr)	Find the VMA intersecting with addr, if one exists.
vector<vma*> find_intersecting_vmas(addr, size)	Find all VMAs intersecting with the address range specified.
bool validate(addr, size)	Test if the given region can be allocated.
void allocate_range(addr, size)	Allocate a given virtual memory region.
uintptr_t reserve_range(size)	Allocate any virtual memory range of the given size.
void free_range(addr, size)	Free the given virtual memory range by returning it back to OSv's free lists.

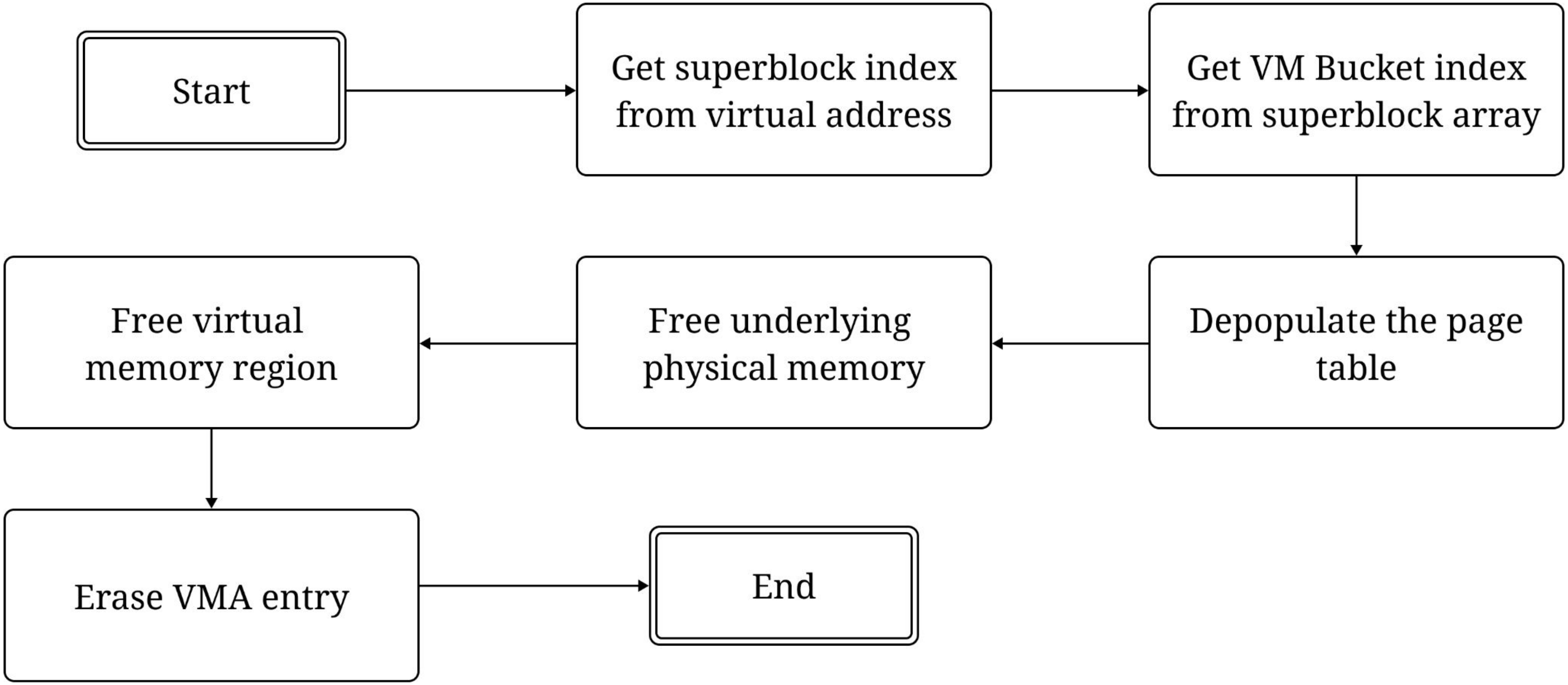
VM Allocation



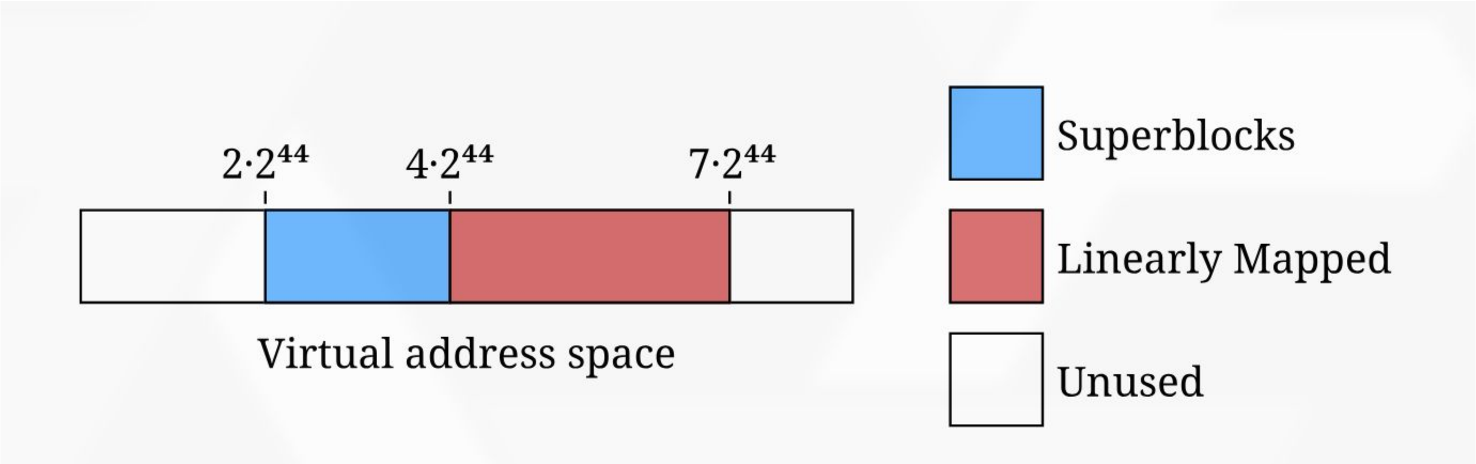
VM Area Lookup



VM Deallocation



VM Layout in OSv



Evaluation: Setup

Host

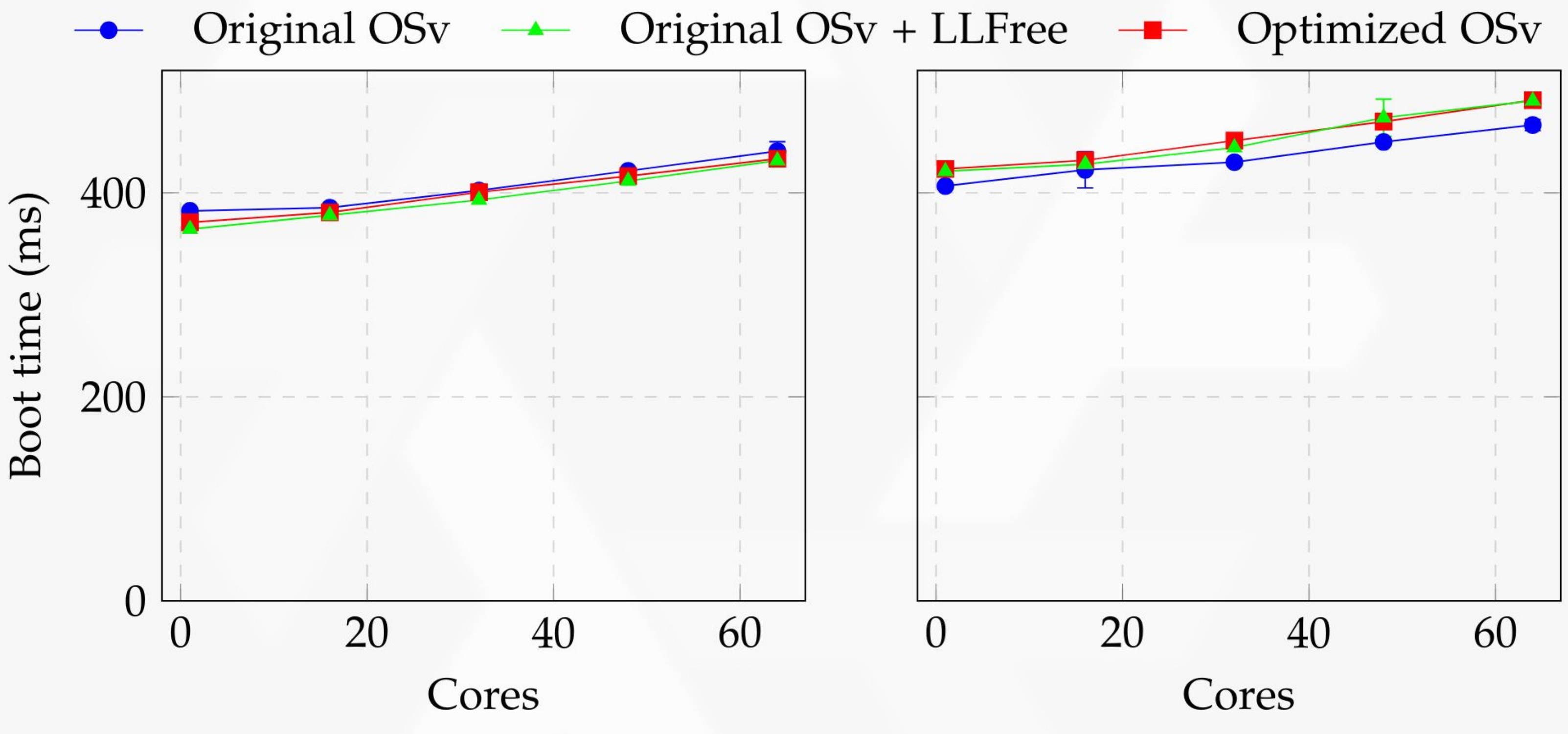
- AMD EPYC 9654P processor (96 cores @ 2.4 GHz, 192 hardware threads)
- 768 GiB DRAM
- Linux 6.12.12 / Qemu 9.2.0

Guest OS

- 60 GB DRAM
- OSv or Linux 6.1.96

Baseline: Original OSv version with native allocation mechanism

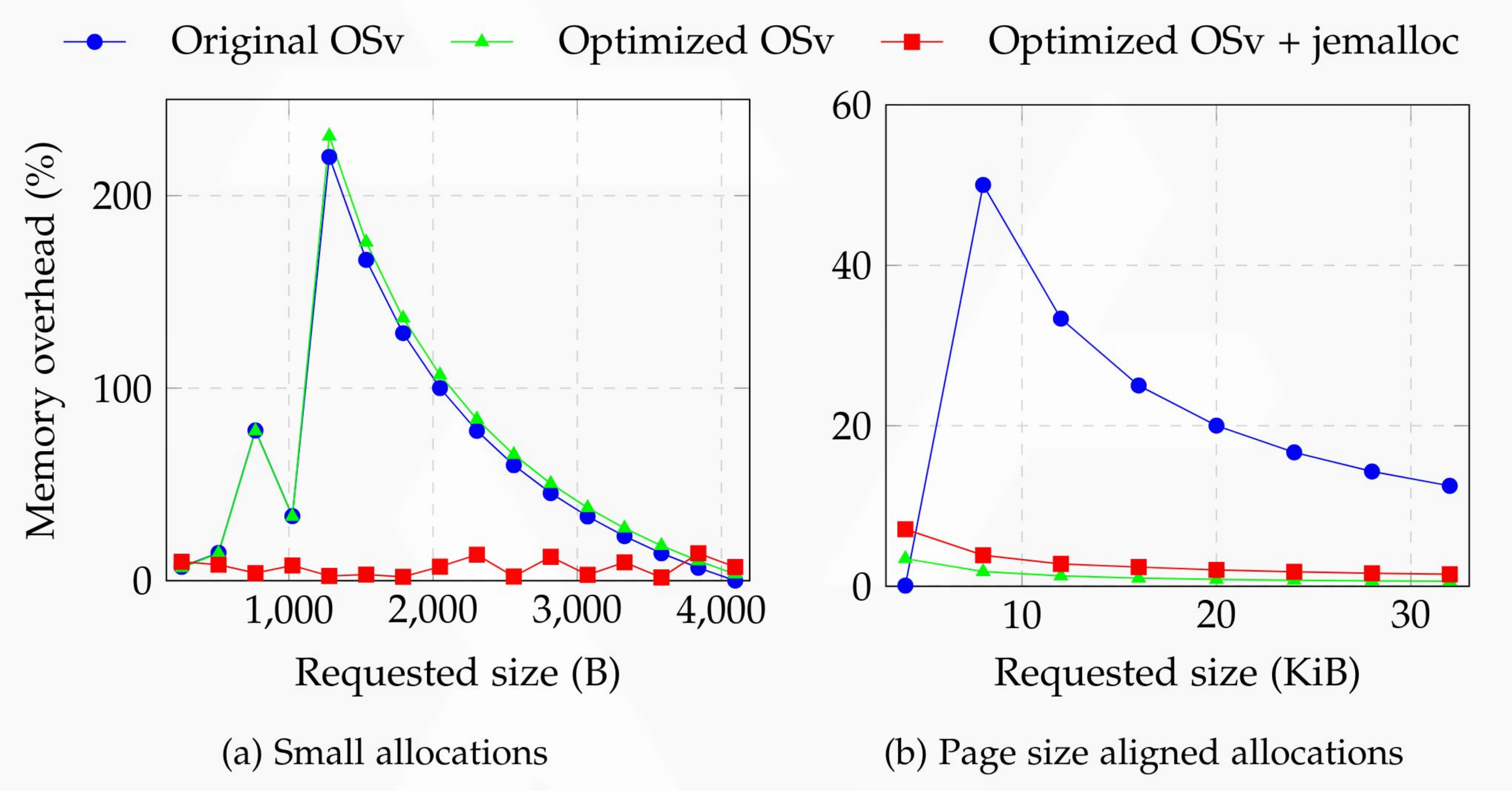
Boot Time



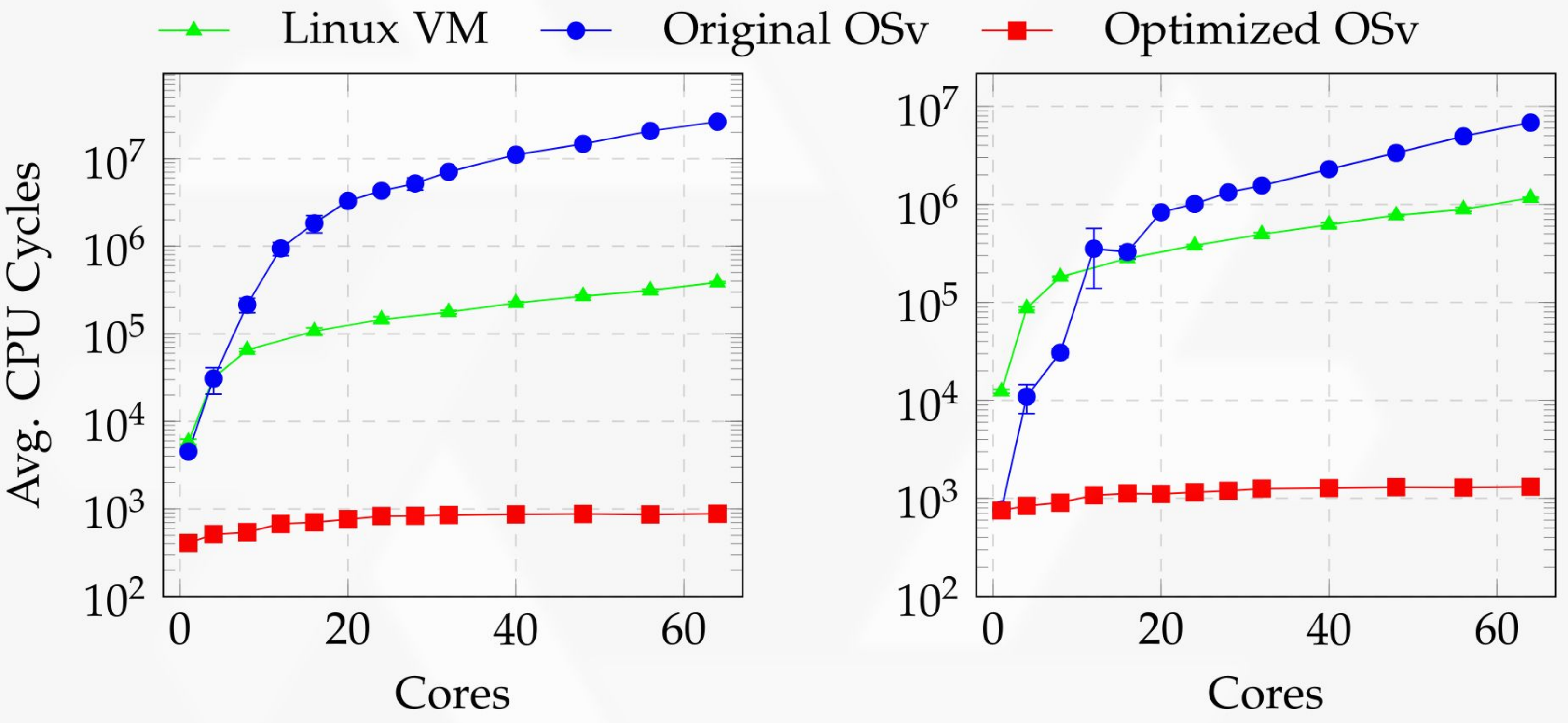
(a) 1GB RAM

(b) 500GB RAM

Memory Footprint



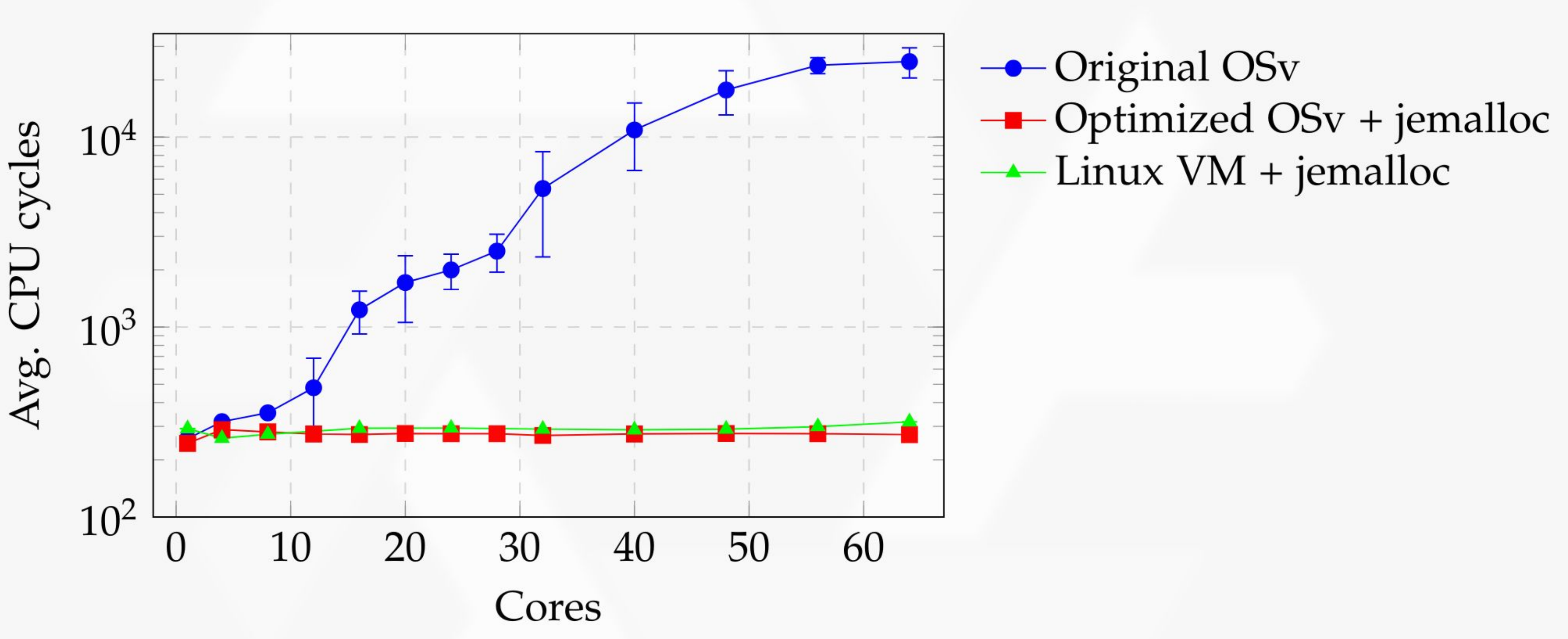
VM Scalability



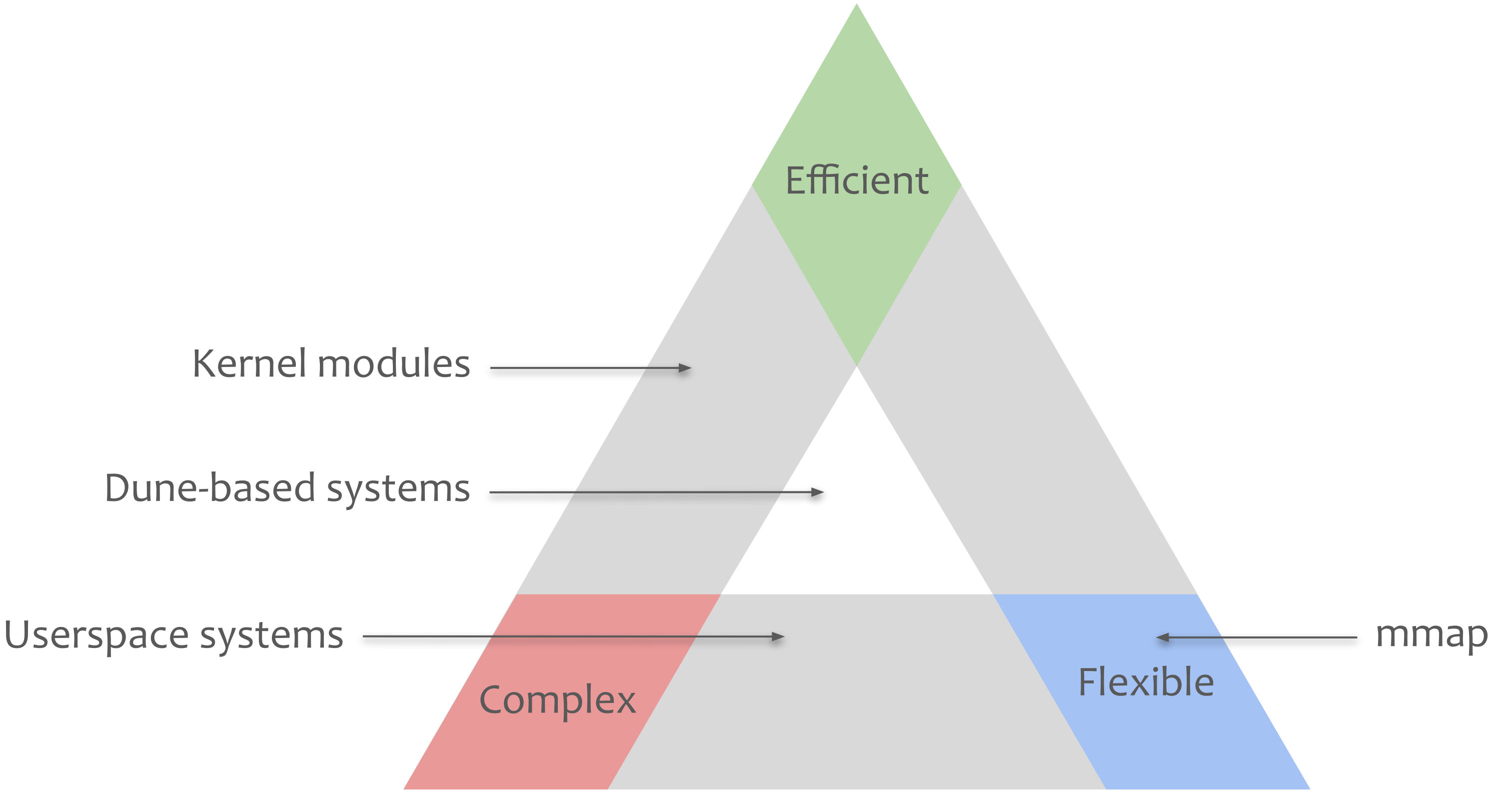
(a) Create virtual mapping

(b) Destroy virtual mapping

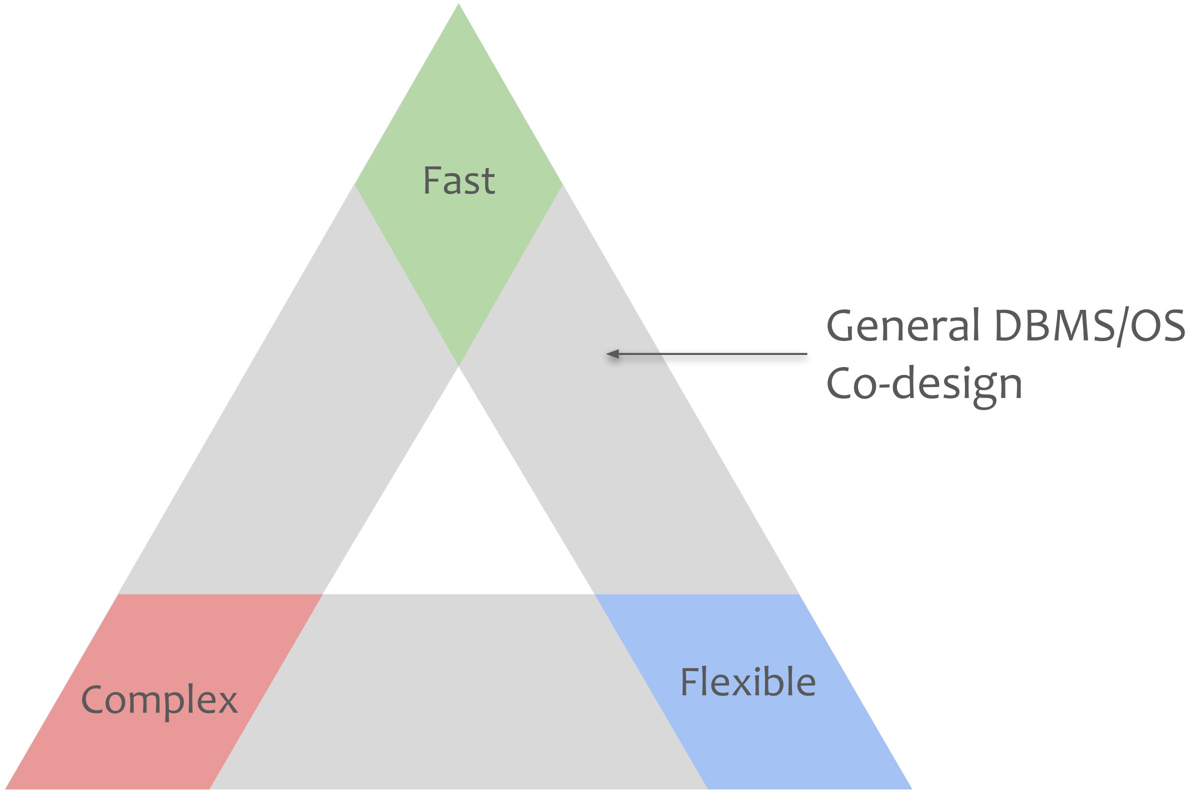
Pipeline deallocation



State-of-the-art



Research Gap



Overview: Unikernel/DBMS Co-design