Tidying Up The Address Space

What a mess!

Vinay Banakar[1, 2], Suli Yang[2], Kan Wu[2]
Andrea Arpaci-Dusseau[1], Remzi Arpaci-Dusseau[1], Kimberly Keeton[2]

(1) WISCONSIN UNIVERSITY OF WISCONSIN–MADISON

(2) Google

Memory is expensive, constrained, and wasted

50% of server costs at Azure and Meta[1]

$ per bit flatlined for **10** years[2]

DRAM costs **265%** more[2]

65% used at Google[3] and Alibaba[4]

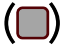
95-98% used at Meta[5]

50% of accesses touch **1%** data at Alibaba[6]

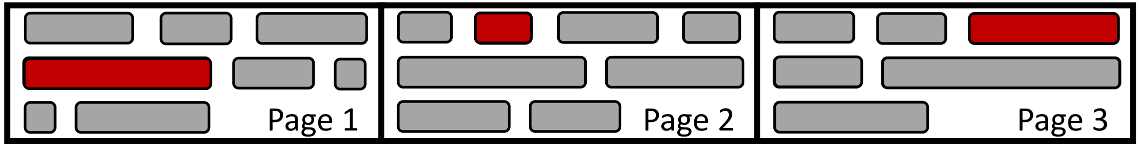
<3% data touched in 24 hours at Meta[7]

[1] Pond: CXL-Based Memory Pooling Systems for Cloud Platforms
[2] DRAMeXchange (6 months -Oct 26 2025)
[3] Borg: the Next Generation
[4] Imbalance in the Cloud: an Analysis on Alibaba Cluster Trace
[5] TPP: Transparent Page Placement for CXL Tiered-Memory
[6] HotRing: A Hotspot-Aware In-Memory Key-Value Store
[7] Benchmarking RocksDB KV Workloads at Facebook

Wasted Memory - Hotness Fragmentation

- Object () creation order determines their location on the virtual address space
- Real world workloads exhibit highly skewed access patterns
- 90% of requests at Meta [1], Twitter [2], and Alibaba [3] access KV objects <1 KiB in size

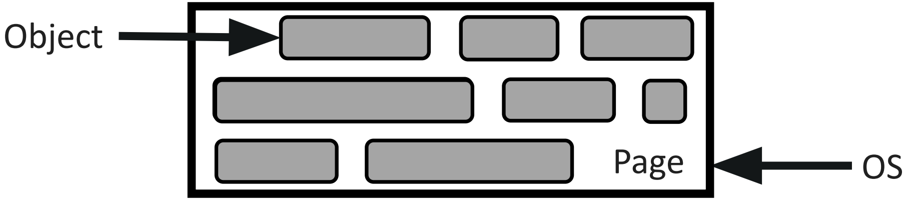
Frequently (hot) and infrequently (cold) accessed objects intermingled on the same page



[1] Characterizing, Modeling, and Benchmarking RocksDB Key-Value Workloads at Facebook, ATC 2020
[2] A large scale analysis of hundreds of in-memory cache clusters at Twitter, OSDI 2020
[3] HotRing: A Hotspot-Aware In-Memory Key-Value Store, FAST 2020

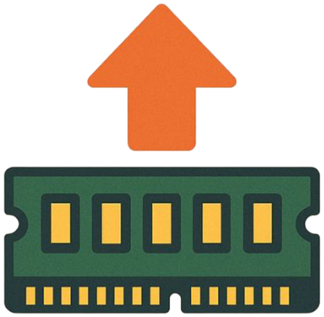
Memory is expensive, constrained, and wasted

Semantic gap between application and OS **memory model**

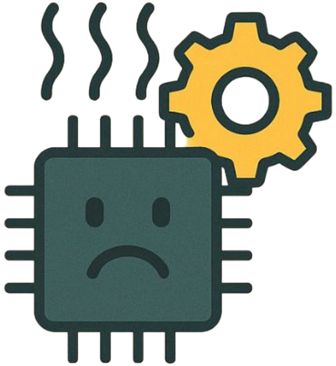


Data is NOT bin-packed based on usage

Data is **NOT bin-packed** based on **usage**



Increased
Memory Footprint



Poor Hardware
Efficiency

Expensive and
Unsustainable
Datacenters

Overview

Data is bin-packed based on usage

- New metric (**Page Utilization**) to quantify hotness fragmentation
- Introduce **Address-Space Engineering**

Overview

Data is bin-packed based on usage

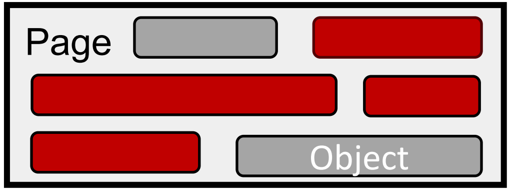
- New metric (**Page Utilization**) to quantify hotness fragmentation
- Introduce **Address-Space Engineering**
- A compiler-runtime system that dynamically reorganize the virtual address space for a workload (**Hierarchically Aware Data structurES**)
- **HADES** increases memory savings of swapping solutions without sacrificing performance
- Demonstrated **70%** memory reduction on YCSB workloads across **10** different popular highly concurrent data structures

Outline

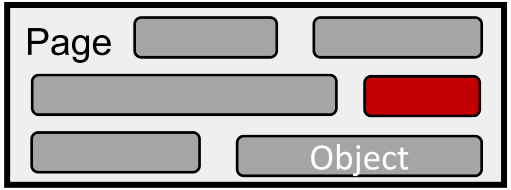
- Page Utilization
- Rewards of Improved Page Utilization
- Address-Space Engineering
- HADES
- Evaluation

Quantify Hotness Fragmentation

$$\text{Page Utilization} = \frac{\text{Total Unique Bytes Accessed}}{\text{Total Unique Pages Accessed x Page Size}}$$



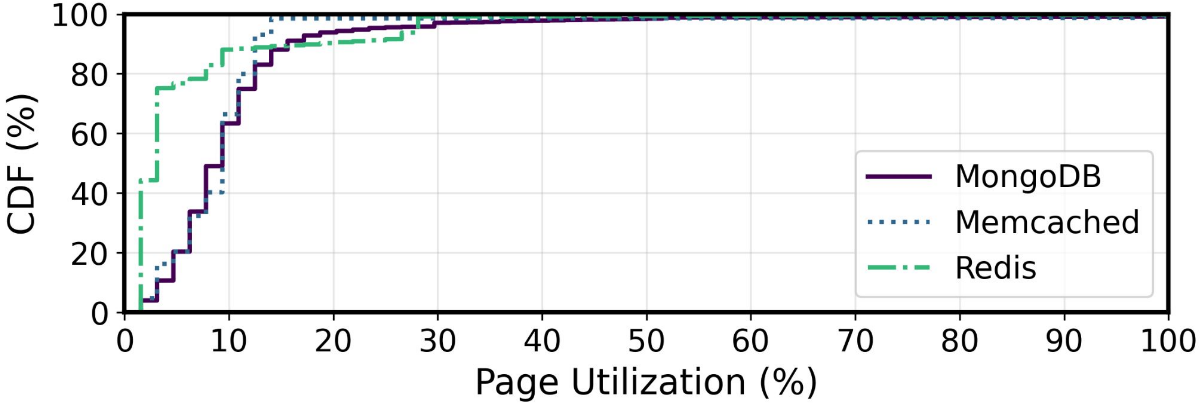
High Page Utilization



Low Page Utilization

Page Utilization directly measures the **hotness fragmentation** of an application for a workload

Poor Page Utilization

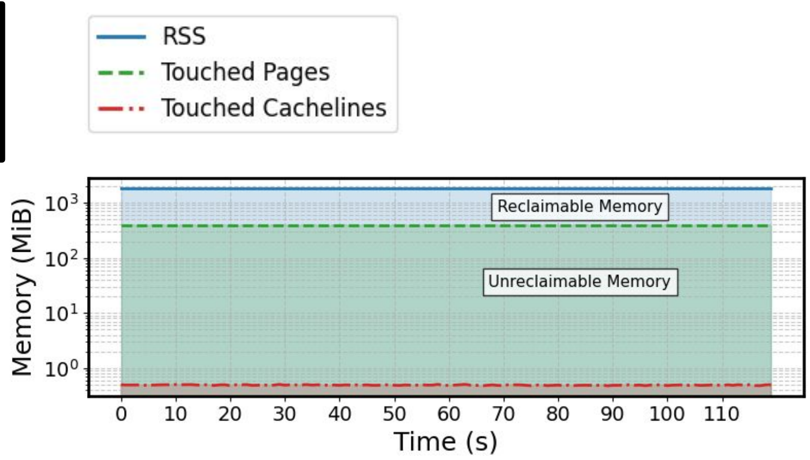


Page Utilization for 360s epochs running YCSB-C with Zipfian distribution

- **75%** of accessed pages in Redis utilize **3%** or less of their capacity
- **95%** of pages in MongoDB and Memcached use less than **15%**

Rewards of Improved Page Utilization

#1 Increased Reclaimable Memory: Fewer pages needed to serve skewed workloads

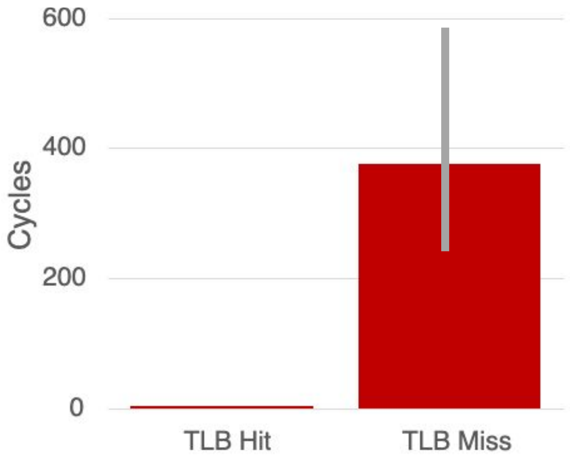


Redis touches ~0.5 MiB of cachelines but 1.2 GiB remains resident for YCSB

Rewards of Improved Page Utilization

#1 Increased Reclaimable Memory: Fewer pages needed to serve skewed workloads

#2 Targeted Huge Pages: Saves CPU cycles without sacrificing reclaimable memory



- 11% of CPU cycles at Google spent on dTLB load misses [1]
- Applying THP indiscriminately increases footprint by 69% [2]

[1] Characterizing a Memory Allocator at Warehouse Scale, ASPLOS 2024
[2] Coordinated and efficient huge page management with ingens, OSDI 2016

Rewards of Improved Page Utilization

#1 Increased Reclaimable Memory: Fewer pages needed to serve skewed workloads

#2 Targeted Huge Pages: Saves CPU cycles without sacrificing reclaimable memory

#3 Require fewer machines: More cost effective and sustainable data centers

- Jobs with small working sets but large allocation footprints are spread across multiple machines [3]
- DRAM produces **12x** more emissions per bit than SSDs [4]

[1] Characterizing a Memory Allocator at Warehouse Scale, ASPLOS 2024
[2] Coordinated and efficient huge page management with ingens, OSDI 2016
[3] Borg: the Next Generation, EuroSys 2020
[4] FairyWREN: A Sustainable Cache for Emerging Write-Read-Erase Flash Interfaces, OSDI 2024

Rewards of Improved Page Utilization

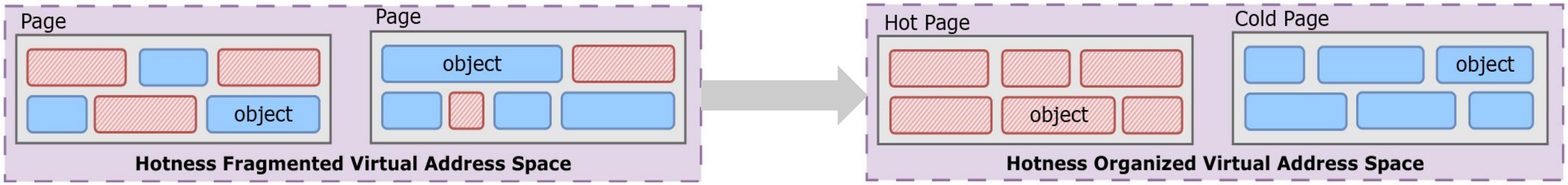
#1 Increased Reclaimable Memory: Fewer pages needed to serve skewed workloads

#2 Targeted Huge Pages: Saves CPU cycles without sacrificing reclaimable memory

#3 Require fewer machines: More cost effective and sustainable data centers

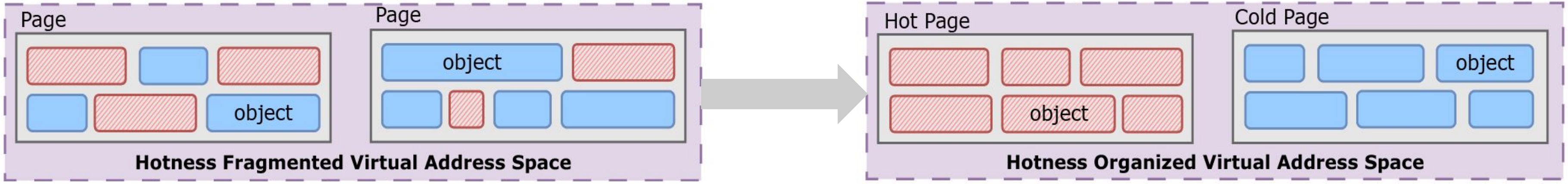
Address-Space Engineering

Engineer the application's virtual address space to be OS-tiering-friendly, adapting to workload access patterns



Address-Space Engineering

Engineer the application's virtual address space to be OS-tiering-friendly, adapting to workload access patterns



Principles of workload-optimized address space:

P1. Decoupling Layout from Reclamation

P2. Grouping Objects by Access Intensity

P3. Enabling Object Mobility

Hierarchically Aware Data structurES

A compiler-runtime frontend that dynamically reorganizes the address space for a workload

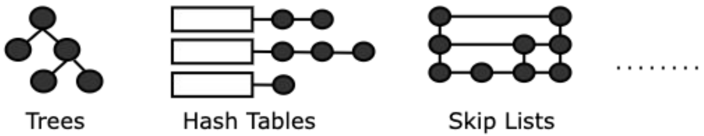
P1. Decoupling Layout from Reclamation
Backend Integration

P2. Grouping Objects by Access Intensity
Tracking and Grouping Objects
Adaptive Workload Response

P3. Enabling Object Mobility
Safe Concurrent Migration

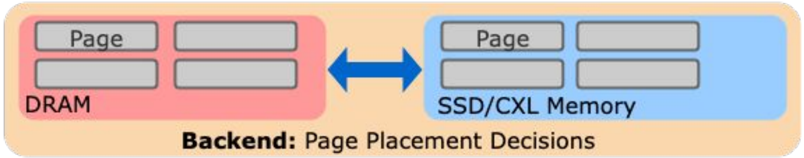
Address-Space Engineering

P1. Decoupling Layout from Reclamation



Requirements:

- Complement tiering / swapping solutions like: Kswapd, TMO, AutoNuma, Zswap, TPP, etc
- Require no new OS abstractions
- Require no specialized hardware knowledge

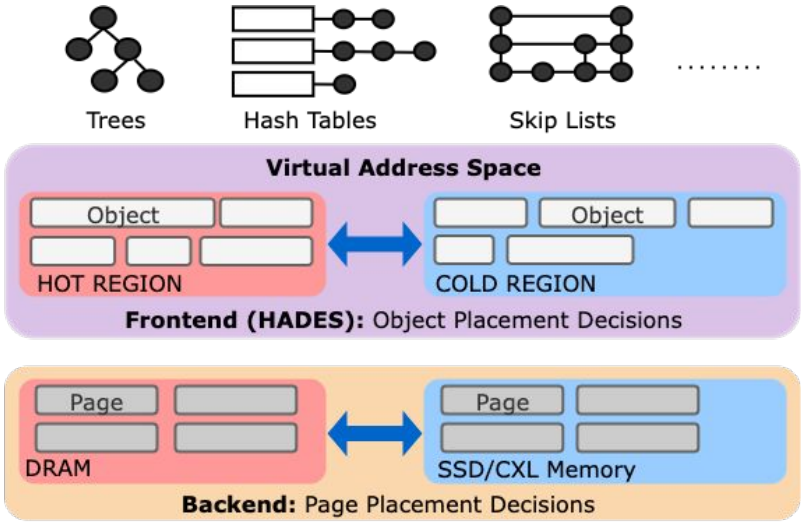


Hierarchically Aware Data structurES

P1. Decoupling Layout from Reclamation

Backend Integration

- Per process user space runtime
- Static analysis for compile time annotations
- **Frontend** organizes address space layout and **backed** acts upon it

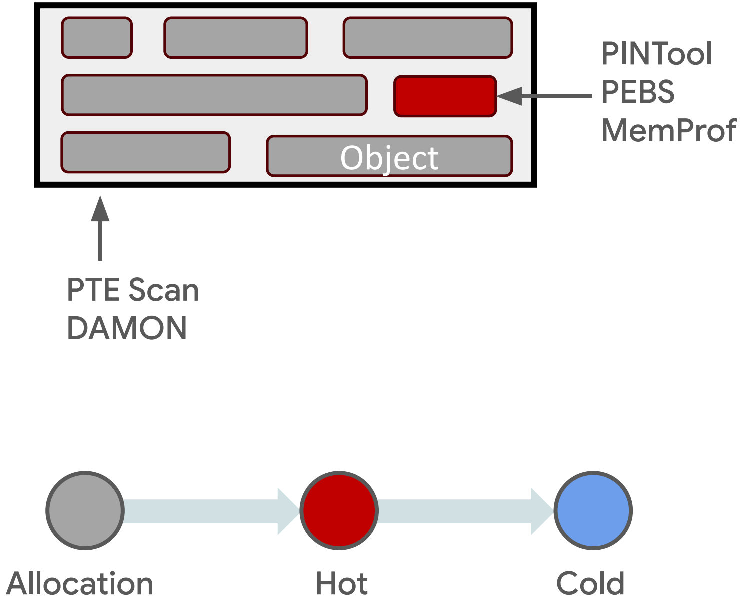


Address-Space Engineering

P2. Grouping Objects by Access Intensity

Requirements:

- Tracking activity at allocation granularity
- Activity tracking must have low overhead
- Static hints at allocation-time is insufficient



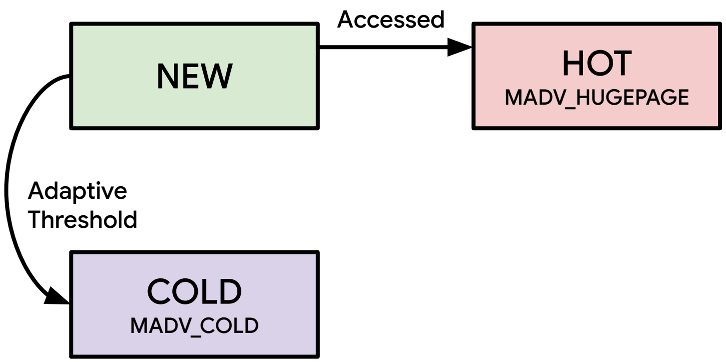
Hierarchically Aware Data structurES

P2. Grouping Objects by Access Intensity

Tracking and Grouping Objects

- Tagged pointers to track activity on dereference
- **Object Collector** (OC) periodically scans managed objects to maintain freshness and group to heaps
- Custom Jemalloc ensures different heap regions are contiguous

OC in action every 120s

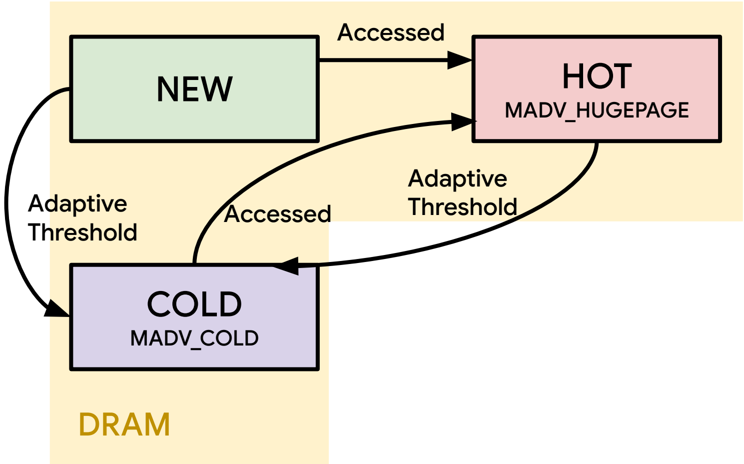


Hierarchically Aware Data structurES

P2. Grouping Objects by Access Intensity

Adaptive Workload Response

- Promotion Rate (PR) as performance proxy
- Adaptive policy to reach target promotion rate
- Proactive reclamation using `MADV_PAGEOUT`

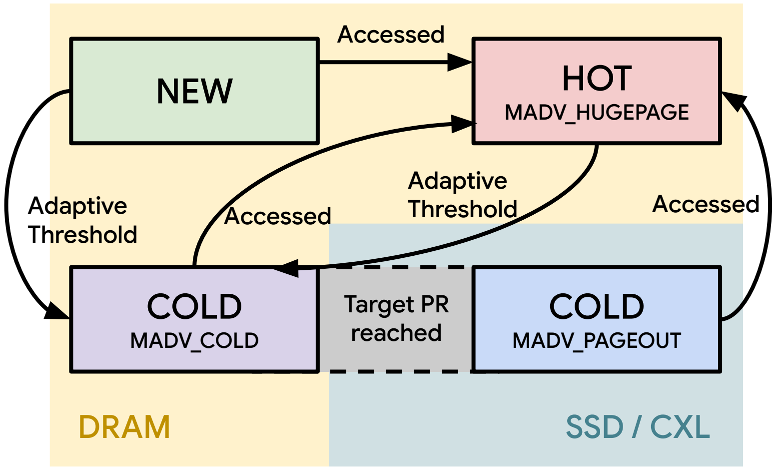


Hierarchically Aware Data structurES

P2. Grouping Objects by Access Intensity

Adaptive Workload Response

- Promotion Rate (PR) as performance proxy
- Adaptive policy to reach target promotion rate
- Proactive reclamation using `MADV_PAGEOUT`

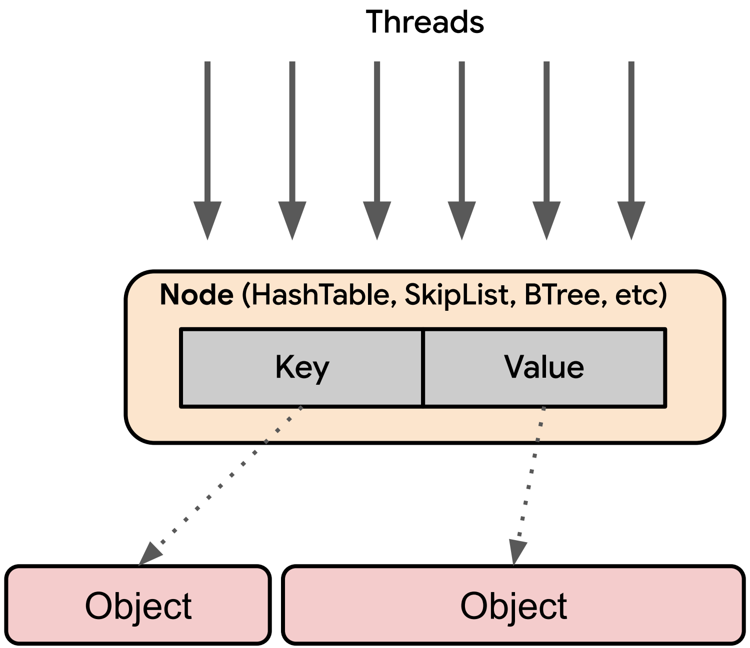


Address-Space Engineering

P3. Enabling Object Mobility

Background and Requirement:

- Unmanaged languages assume object addresses are fixed after allocation
- Focus on **pointer-based data structures**
- Fast but safe in concurrent environments

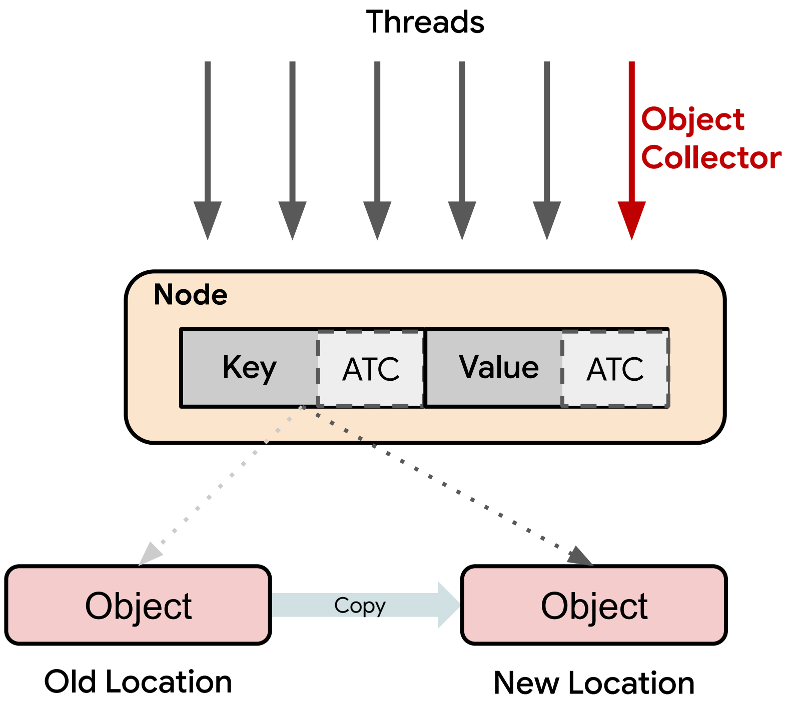


Hierarchically Aware Data structurES

P3. Enabling Object Mobility

Safe Concurrent Migration

- Track objects activity in real-time using Active Thread Count (ATC) embedded in unused bits
- Compiler manages ATC through static analysis
- Optimistic Object Migration Protocol



Evaluation

- **Ten** popular pointer based data structures
  - ART, MassTree, BTree, HashTable, etc
  - Used by Redis, LevelDB, NGINX, DuckDB, etc

- **Six** unique concurrency mechanisms ranging from global locks to lock-free

- **CrestDB** with YCSB for consistency

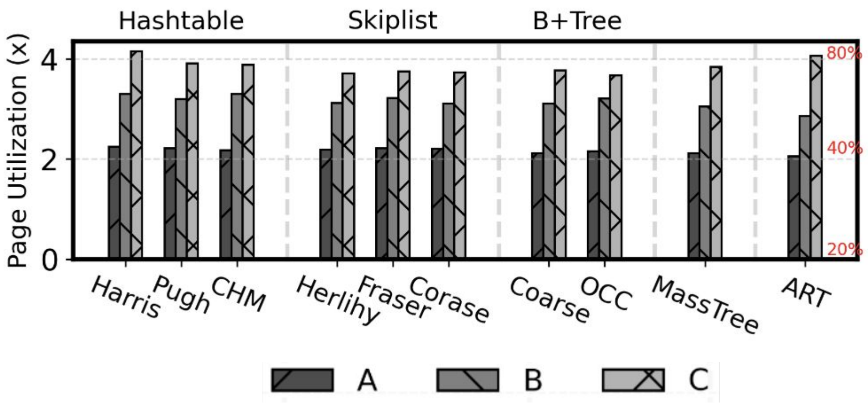
- TLB performance optimization for bulk reclaim in linux kernel

CPU	Intel(R) Xeon Gold 5218 (16 cores)
Memory	2x 16GB @2400MHz DRAM
SSD	512 GB P4800x SSD
OS	Ubuntu 22.04

Frontend Effectiveness

10M KV pairs with 30B Keys and 1024B values

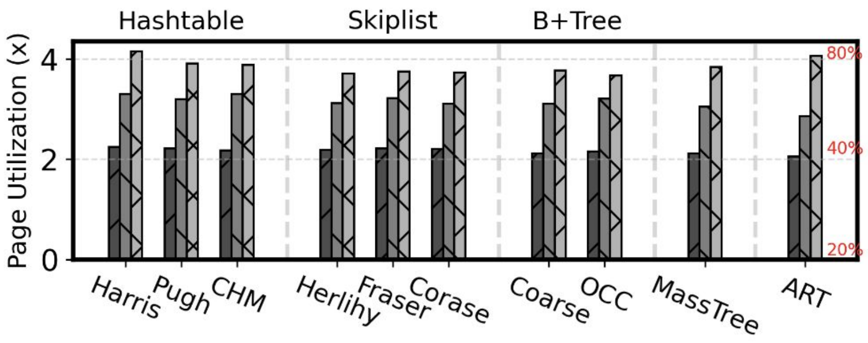
- HADES **increases page utilization** by
  - **2x** for workload A (50% read 50% write)
  - **3x** for workload B (95% read 5% write)
  - **4x** for workload C (100% read)

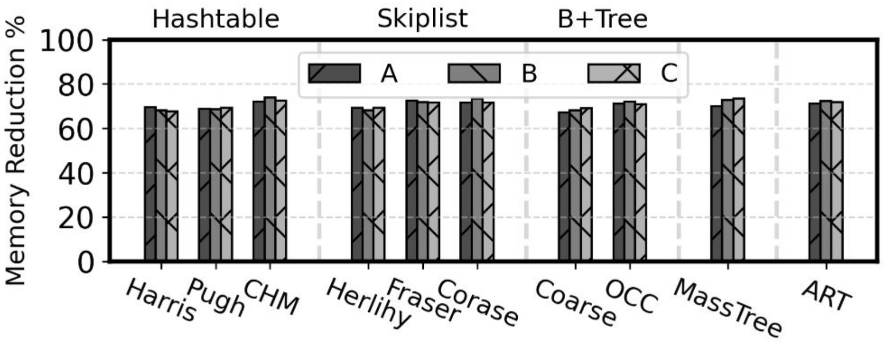


Frontend Effectiveness

10M KV pairs with 30B Keys and 1024B values

- HADES **increases page utilization** by
  - **2x** for workload A (50% read 50% write)
  - **3x** for workload B (95% read 5% write)
  - **4x** for workload C (100% read)
- HADES reduces **memory usage** by up to **70%** through object-level cold detection and heap organization

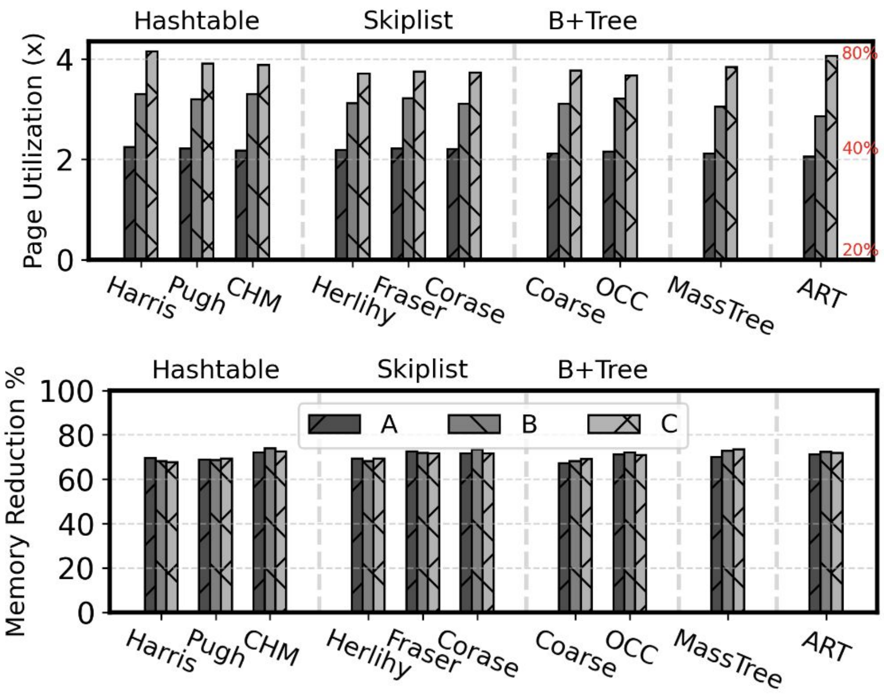




Frontend Effectiveness

10M KV pairs with 30B Keys and 1024B values

- HADES **increases page utilization** by
  - **2x** for workload A (50% read 50% write)
  - **3x** for workload B (95% read 5% write)
  - **4x** for workload C (100% read)
- HADES reduces **memory usage** by up to **70%** through object-level cold detection and heap organization
- HADES tracking overhead lowers average throughput by **2.5%** and increases P90 latency by **5%**



Backend Validation

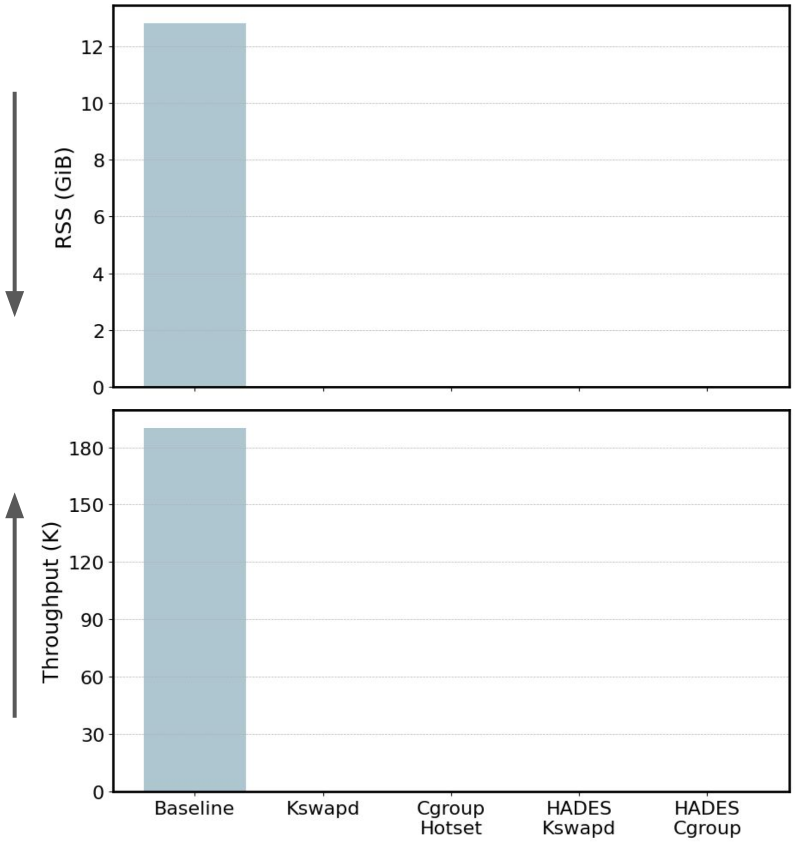
YCSB-C with ~12 GiB footprint and ~4 GiB actively accessed

Tiering backends:

- Kswapd: Reclaims under memory pressure
- Cgroup hotset: Cgroup limit set to 4GiB

Standard backends

- Sacrifice performance to save memory (cgroup hotset)
- Sacrifice savings to preserve performance (kswapd)



Backend Validation

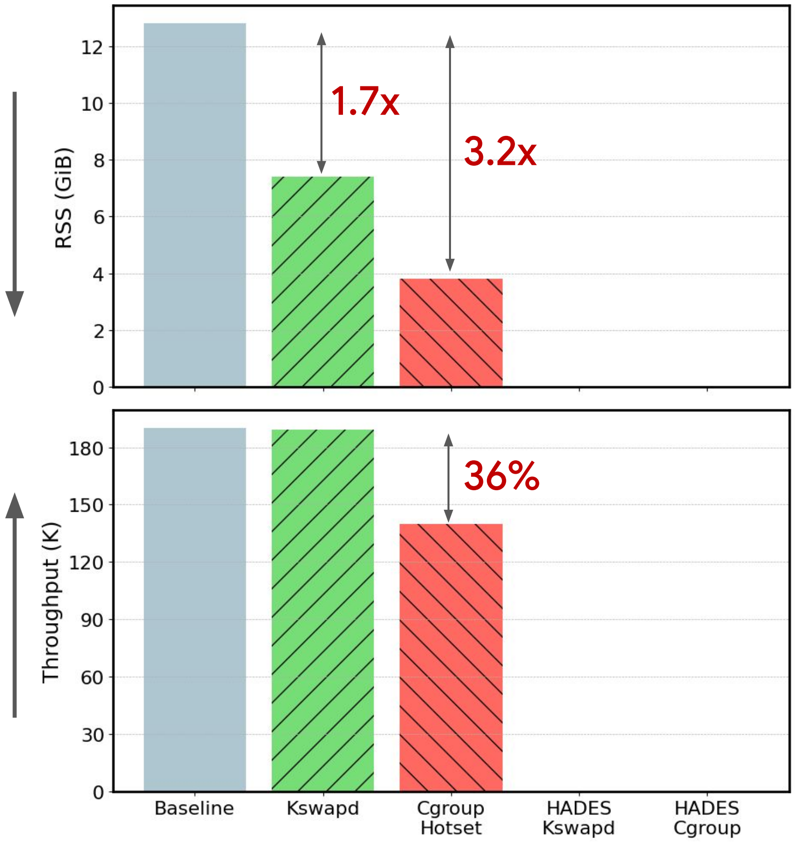
YCSB-C with ~12 GiB footprint and ~4 GiB actively accessed

Tiering backends:

- Kswapd: Reclaims under memory pressure
- Cgroup hotset: Cgroup limit set to 4GiB

Standard backends

- Sacrifice performance to save memory (cgroup hotset)
- Sacrifice savings to preserve performance (kswapd)



Backend Validation

YCSB-C with ~12 GiB footprint and ~4 GiB actively accessed

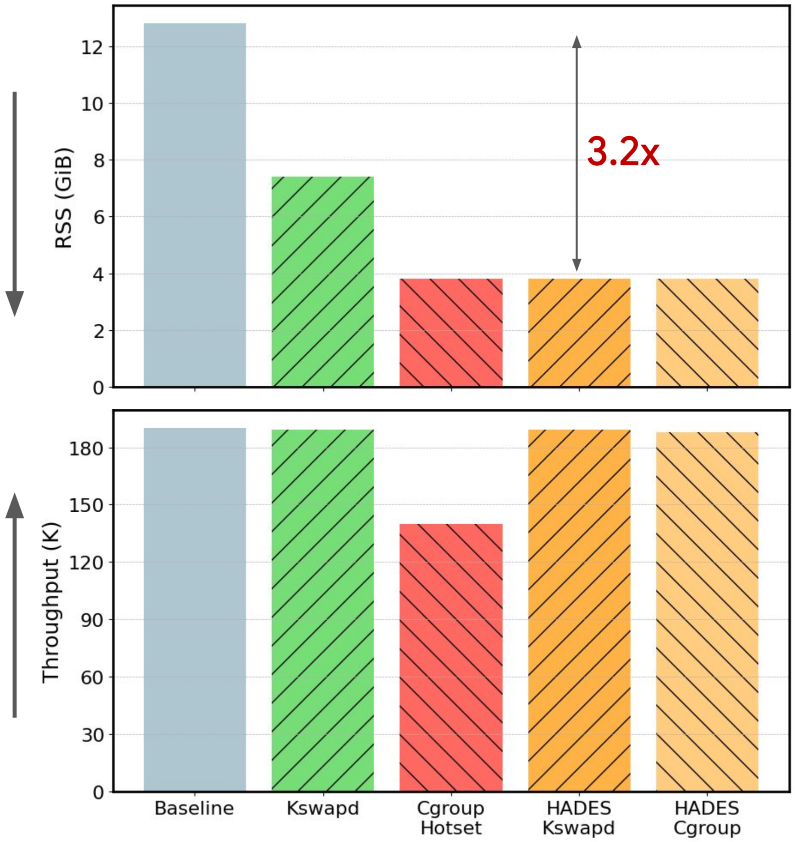
Tiering backends:

- Kswapd: Reclaims under memory pressure
- Cgroup hotset: Cgroup limit set to 4GiB

Standard backends

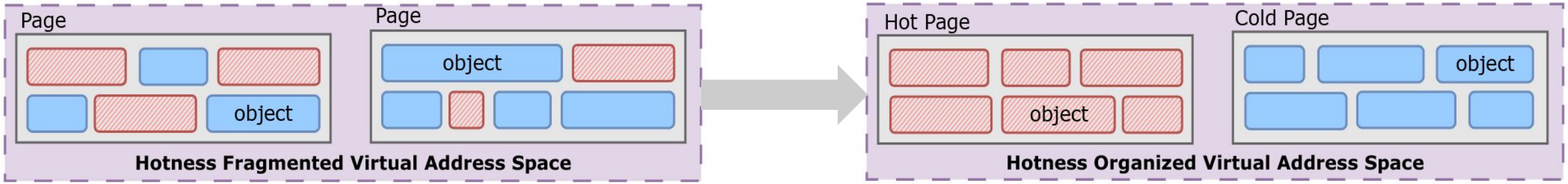
- Sacrifice performance to save memory (cgroup hotset)
- Sacrifice savings to preserve performance (kswapd)

HADES achieves max memory savings with no performance degradation



Summary

- **Semantic gap** between application and OS memory model
- **Hotness fragmentation** is widespread
- Address-Space Engineering for improving page utilization
- **HADES** as an approach for Address-Space Engineering
- Achieved higher memory savings across diverse data structures without degrading performance



Open Questions

- Are there features that we should be adding to managed language to enable easier address-space engineering?

  Garbage collectors divides memory into lifetime generations and not access generations

- What OS abstractions/interfaces to support dynamically reorganize the address space?

- What future applications can we have with dynamically organized address space?

  Address-space engineering could improve multiple aspects of execution: concurrency, isolation, tiering

Limitations

- Lack of pointer stability
- Unique Object Ownership
- Incompatibility with pointer arithmetic
- Language support restrictions
- Requirement for explicit annotations

Related Work

- **Object-Level Management:**
  - AIFM and MIRA operate at object granularity but focus exclusively on far-memory over RDMA, requiring direct hardware access that limits production adoption
  - Alaska uses handle-based indirection to reduce RSS through heap compaction, addressing fragmentation reactively without object hotness classification
- **Runtime vs. Allocation-Time Placement:** Allocation time hinting approaches fail to capture objects transitioning between hot and cold states or distinguish between objects from the same allocation site with different access patterns
- **Page-Level Optimizations:** TPP, HeMEM, TMTS, MEMTIS, etc