OBASE: Object Based Address-Space-Engineering

Vinay Banakar, Suli Yang, Kan Wu
Andrea Arpaci-Dusseau, Remzi Arpaci-Dusseau, Kimberly Keeton

UNIVERSITY OF WISCONSIN–MADISON
Google

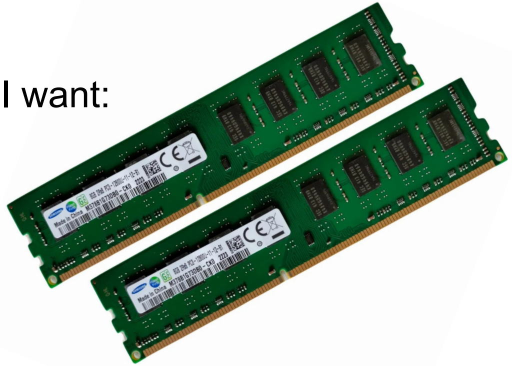
Memory is **expensive**

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

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

DDR5 costs **265%** more [2]

the RAM I want:

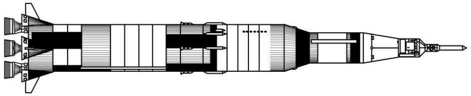


the RAM I can afford:

[1] Pond: CXL-Based Memory Pooling Systems for Cloud Platforms
[2] DRAMeXchange (6 months -Feb 2026)

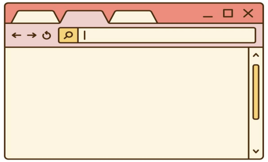
Memory is constrained and wasted

4KB of RAM in 1996:



Landed humanity on moon

32GB of RAM in 2026:



Struggles to run 5 tabs

65% used at Google [1] and Alibaba [2]

50% of accesses touch **1%** data at Alibaba [4]

95-98% used at Meta [3]

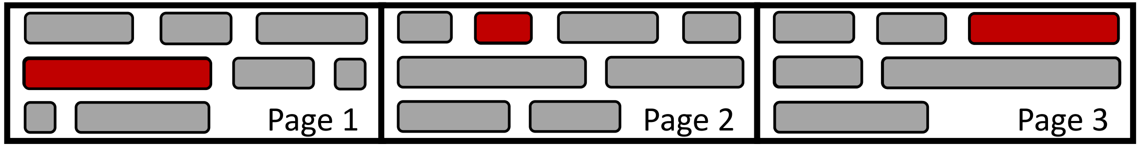
<3% data touched in 24 hours at Meta [5]

[1] Borg: the Next Generation
[2] Imbalance in the Cloud: an Analysis on Alibaba Cluster Trace
[3] TPP: Transparent Page Placement for CXL Tiered-Memory
[4] HotRing: A Hotspot-Aware In-Memory Key-Value Store
[5] 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

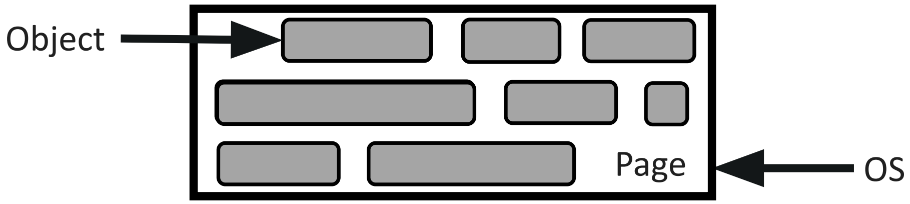
Frequently (hot) and infrequently (cold) accessed objects intermingled on the same page



[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

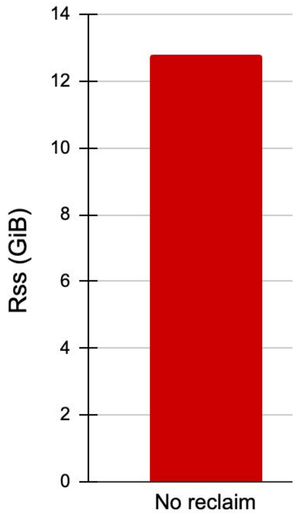
Memory is expensive, constrained, and wasted

Semantic gap between application and OS memory model

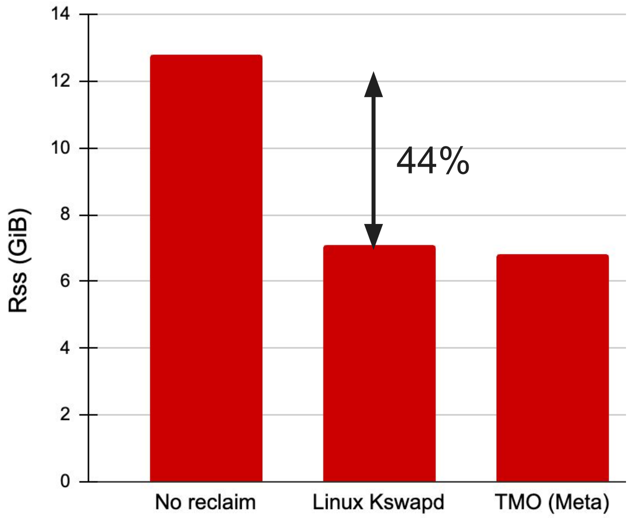


Data is NOT bin-packed based on usage

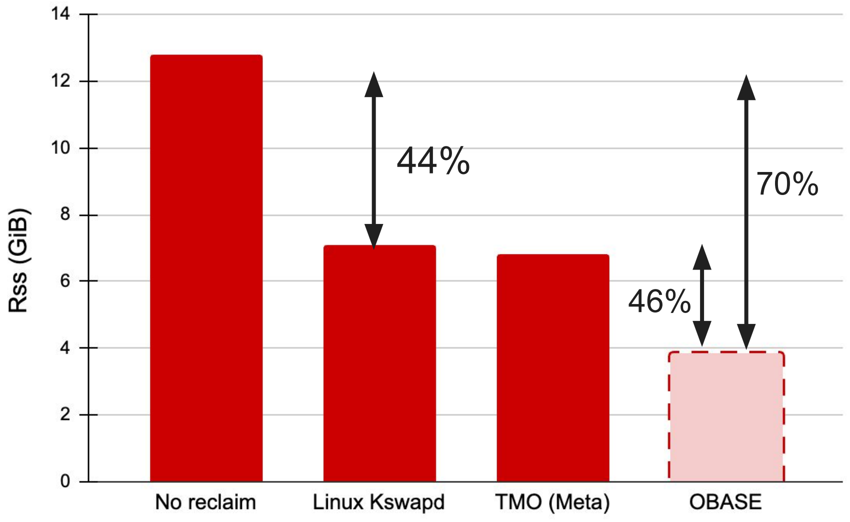
Transparently improve memory efficiency of applications for skewed workloads



Transparently improve memory efficiency of applications for skewed workloads



Transparently improve memory efficiency of applications for skewed workloads



OBASE: **Object-Based Address-Space Engineering to Improve Memory Tiering**

Overview

Data is bin-packed based on usage

- New metric (**Page Utilization**) to quantify a new kind of (**hotness**) fragmentation
- The principles of **Address-Space Engineering**
- A compiler-runtime system that dynamically reorganize the virtual address space for a workload (**OBASE**)
- **OBASE** increases memory savings of swapping solutions without sacrificing performance
- **OBASE** increases performance of page based tiering solutions

Outline

- **Page Utilization**
  A new metric to quantify hotness fragmentation

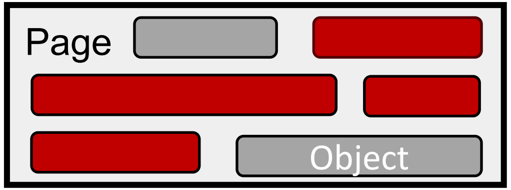
- Address-Space Engineering
  Active management of virtual address space as a resource

- OBASE
  Dynamically optimize application memory layout

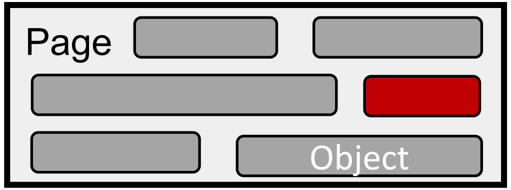
- Evaluation

Quantify Hotness Fragmentation

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



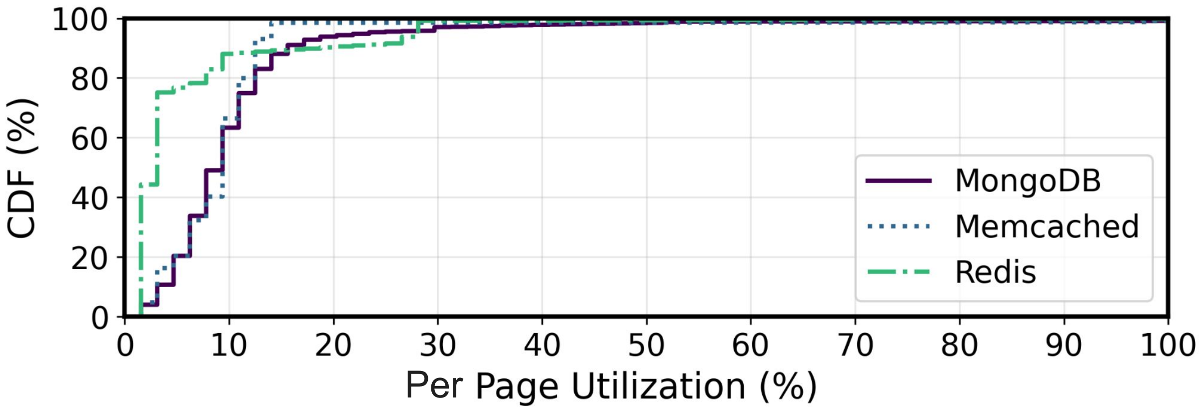
High Page Utilization



Low Page Utilization

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

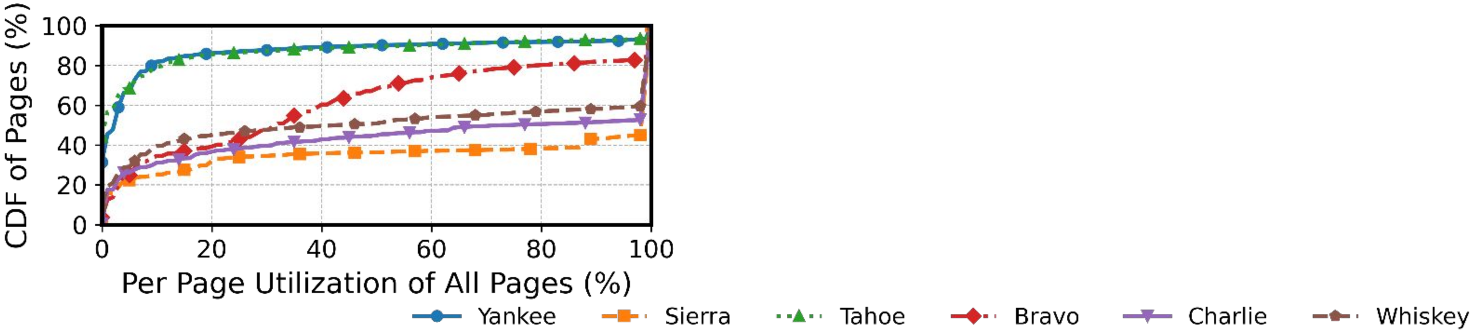
Active pages are mostly cold (open source)



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%**

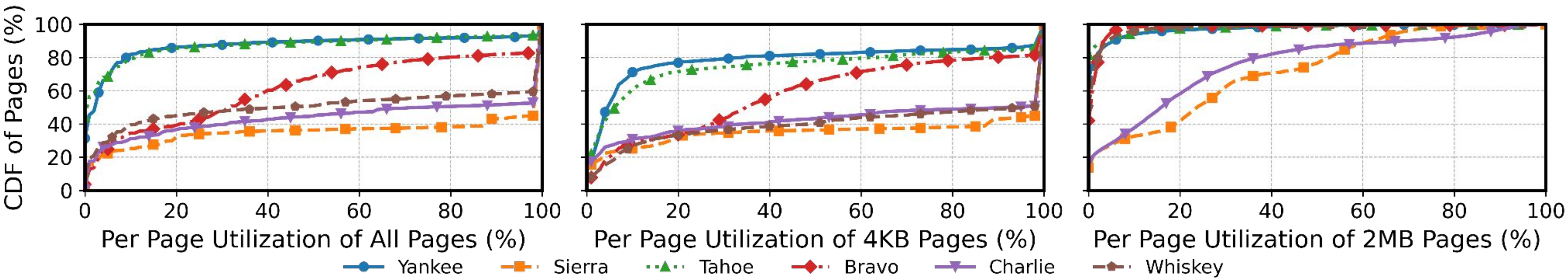
Active pages are mostly **cold** (datacenter)



Page Utilization for upto 30s of Google datacenter workloads

- All pages: **Half** of all pages waste over **70%** of their capacity on cold data

Active pages are mostly **cold** (datacenter)

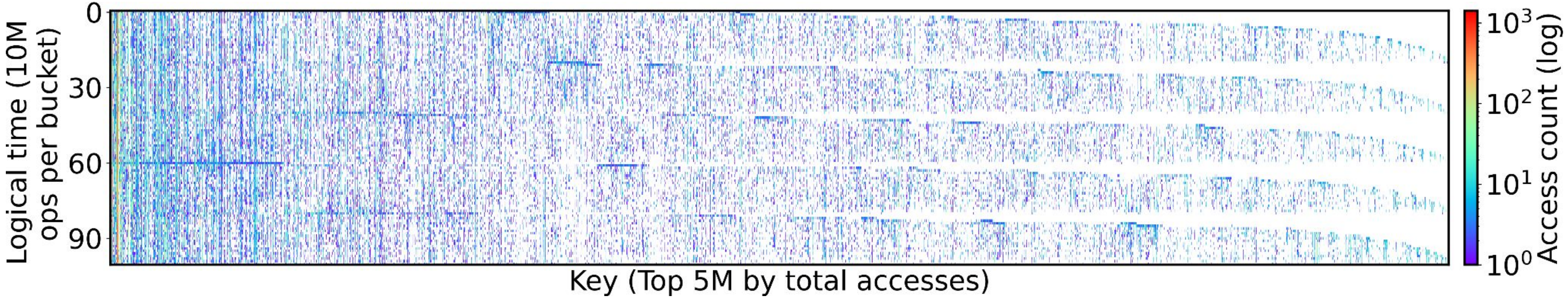


Page Utilization for upto 30s of Google datacenter workloads

- All pages: **Half** of all pages waste over **70%** of their capacity on cold data
- 4KB pages: **50–80%** of pages utilize less than **30%** of their capacity
- 2MB pages: **90%** of huge pages utilize less than **10%** of their capacity

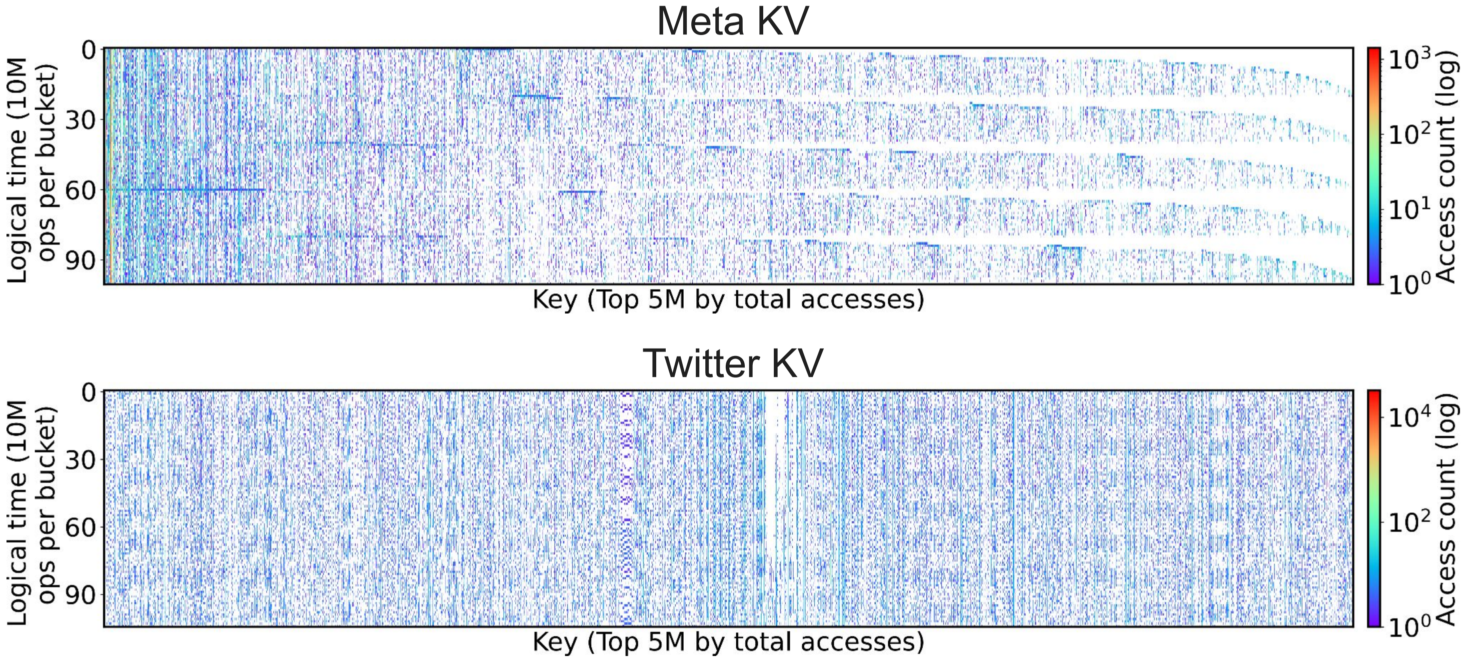
Object Hotness Is Transient

Meta KV Cachelib Trace



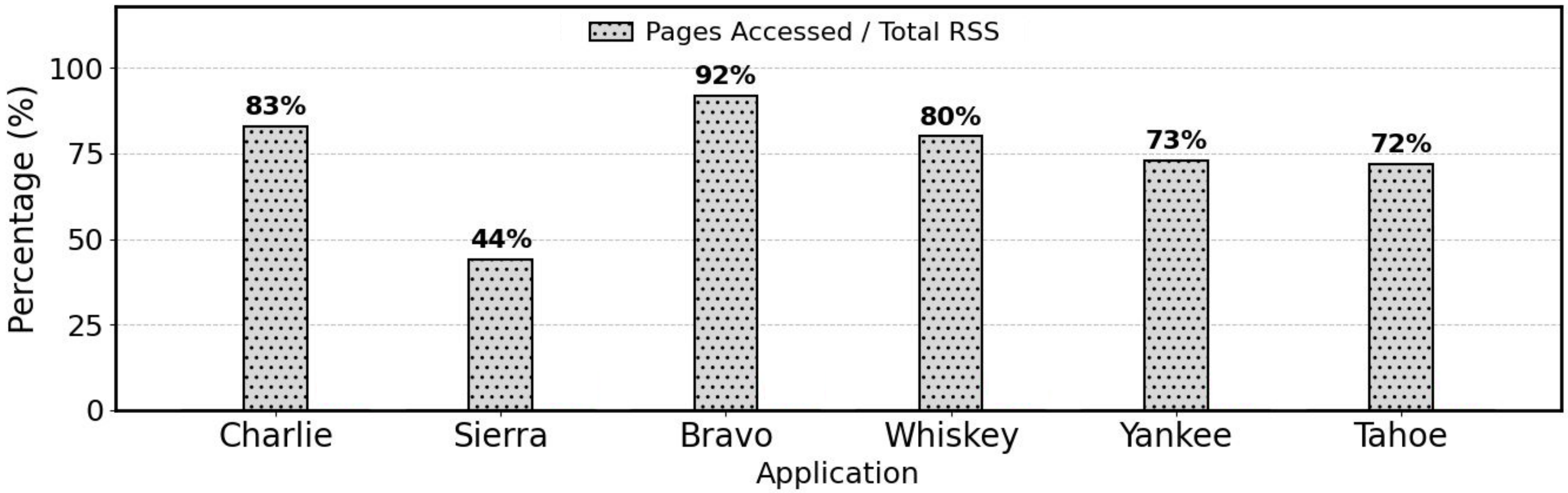
- Bursts of activity followed by long idle gaps
- Object hotness is neither knowable at allocation nor stable over time

Object **Hotness** Is Transient



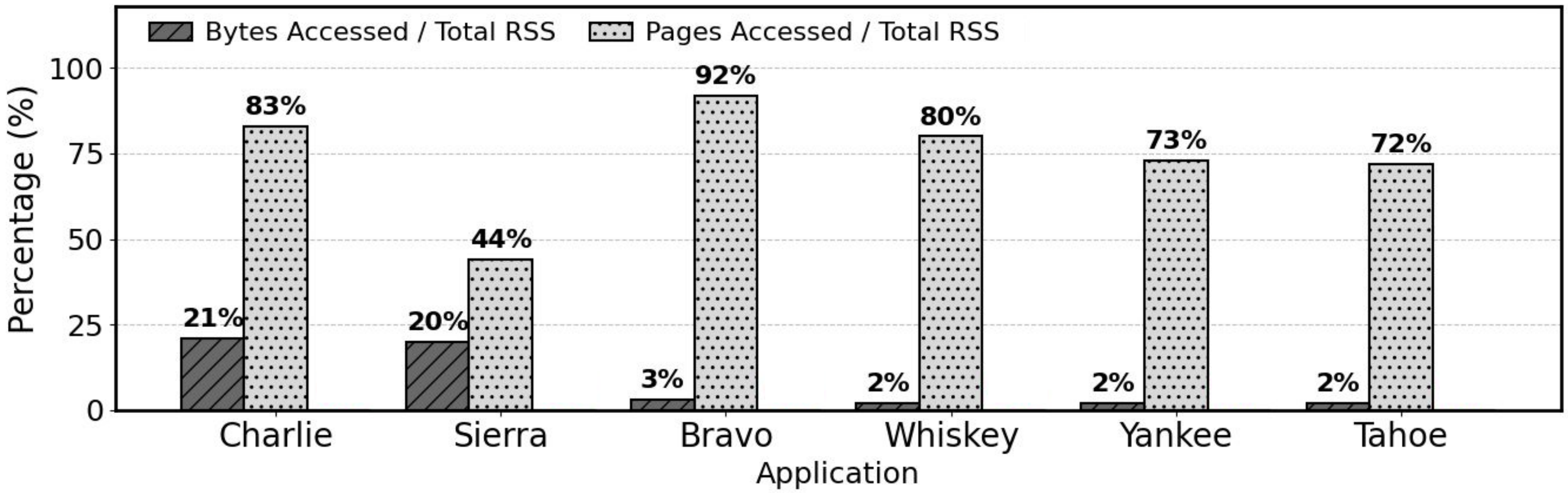
- Bursts of activity followed by long idle gaps
- Object hotness is neither knowable at allocation nor stable over time

Lower Page Utilization == Lower Reclaimable Memory



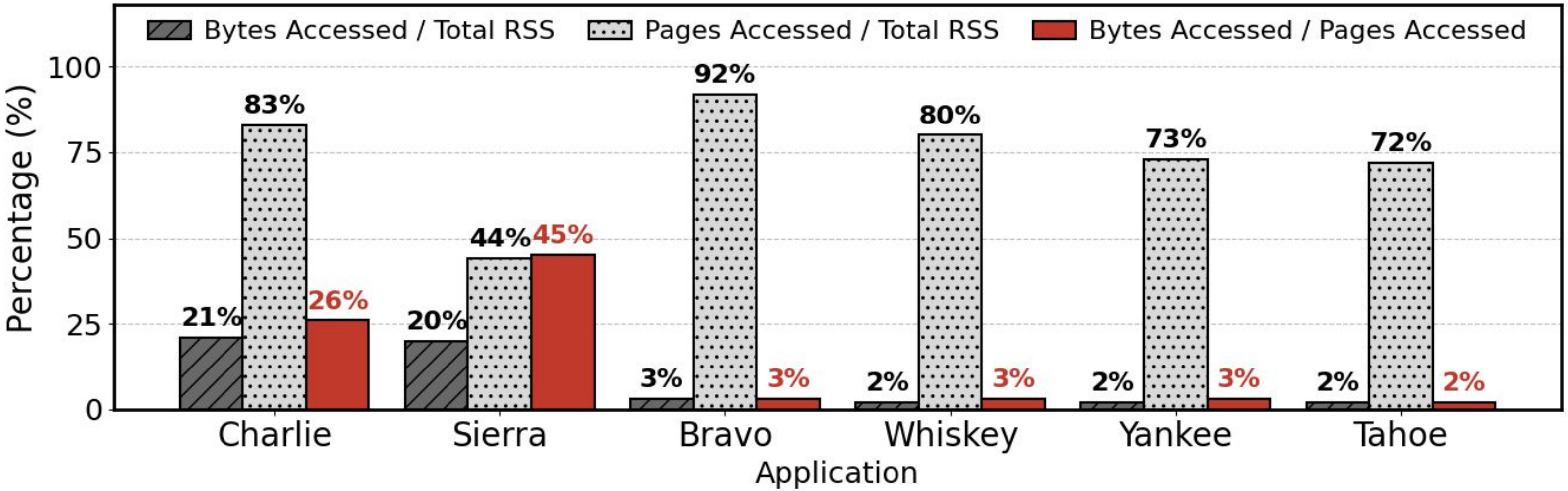
- Fewer pages needed to serve skewed workloads
- 2-21% of bytes are accessed, yet 44%-92% of pages are touched, because a few hot objects pin entire pages in DRAM
- Over 97% of memory within touched pages cannot be reclaimed despite being cold

Lower Page Utilization == Lower Reclaimable Memory



- Fewer pages needed to serve skewed workloads
- 2-21% of bytes are accessed, yet 44%-92% of pages are touched, because a few hot objects pin entire pages in DRAM
- Over 97% of memory within touched pages cannot be reclaimed despite being cold

Lower Page Utilization == Lower Reclaimable Memory



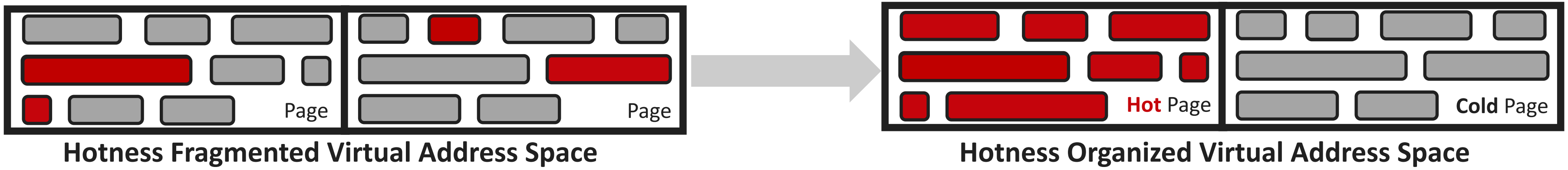
- Fewer pages needed to serve skewed workloads
- 2-21% of bytes are accessed, yet 44%-92% of pages are touched, because a few hot objects pin entire pages in DRAM
- Over 97% of memory within touched pages cannot be reclaimed despite being cold

Outline

- Page Utilization
  A new metric to quantify hotness fragmentation
- **Address-Space Engineering**
  Active management of virtual address space as a resource
- OBASE
  Dynamically optimize application memory layout
- Evaluation

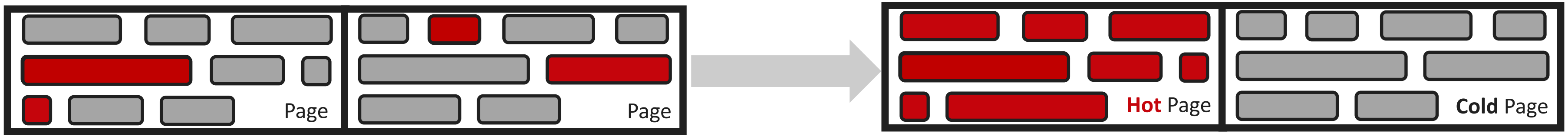
Address-Space-Engineering

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



Hotness Fragmented Virtual Address Space

Hotness Organized Virtual Address Space

Principles of workload-optimized address space:

P1. Decoupling Layout from Tiering

P2. Dynamically Grouping Data by Access Intensity

P3. Enabling Data Mobility

Outline

- Page Utilization
  A new metric to quantify hotness fragmentation
- Address-Space Engineering
  Active management of virtual address space as a resource
- **OBASE**
  Dynamically optimize application memory layout
- Evaluation

Object Based Address-Space-Engineering

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

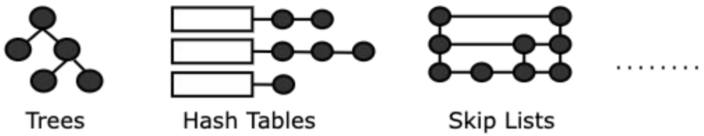
P1. Decoupling Layout from Tiering
Backend Independence

P2. Grouping Objects by Access Intensity
Tracking and Grouping Objects
Dynamic Adaptation

P3. Enabling Object Mobility
Safe Concurrent Migration

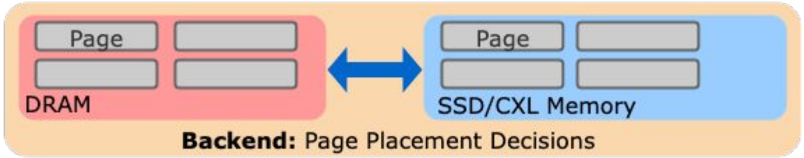
Address-Space-Engineering

P1. Decoupling Layout from Tiering



Requirements:

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

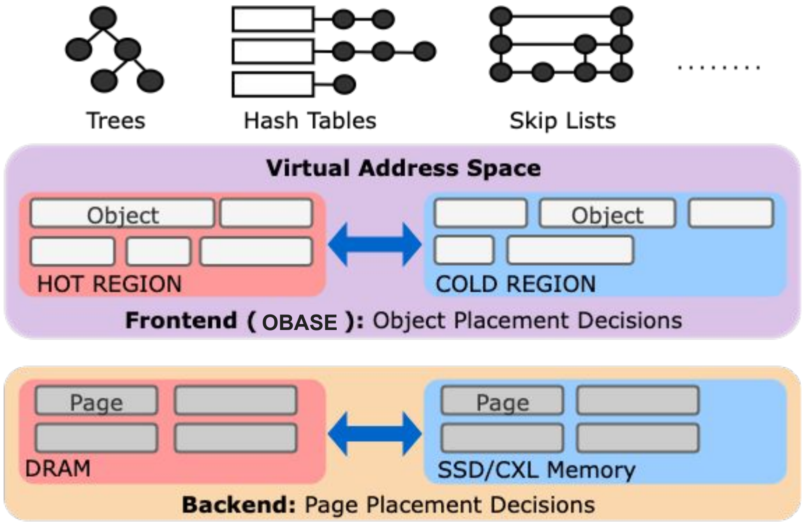


Object Based Address-Space-Engineering

P1. Decoupling Layout from Tiering

Backend Independence

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



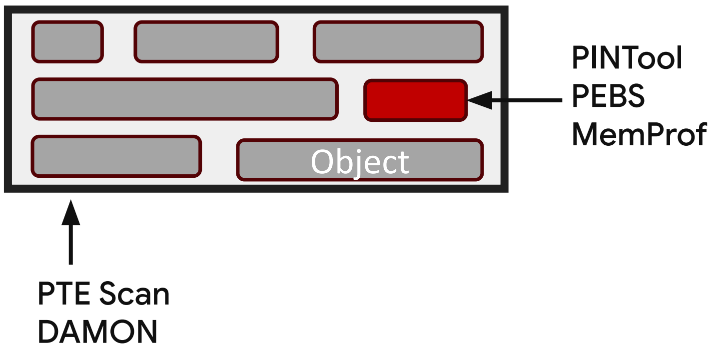
TMO, AutoNuma, TPP, MEMTIS, etc

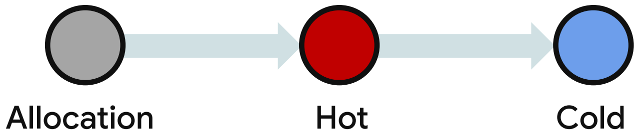
Address-Space-Engineering

P2. Grouping Objects by Access Intensity

Requirements:

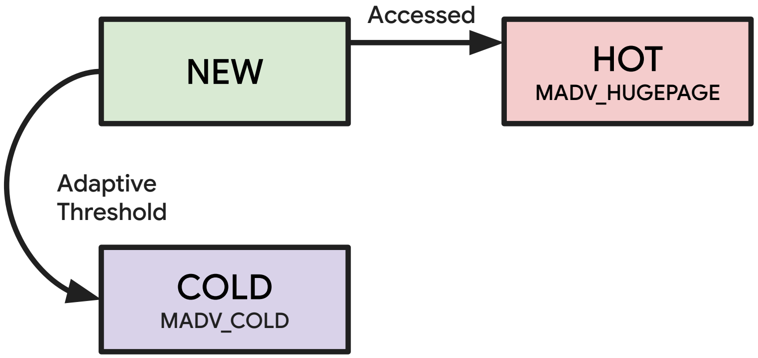
- Tracking activity at allocation granularity
- Activity tracking must have low overhead
- Static hints at allocation-time is insufficient





P2. Grouping Objects by Access Intensity

OC in action every 120s



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 enables **OC** hints and ensures different heap regions are contiguous

Tagged Pointer (Guide):

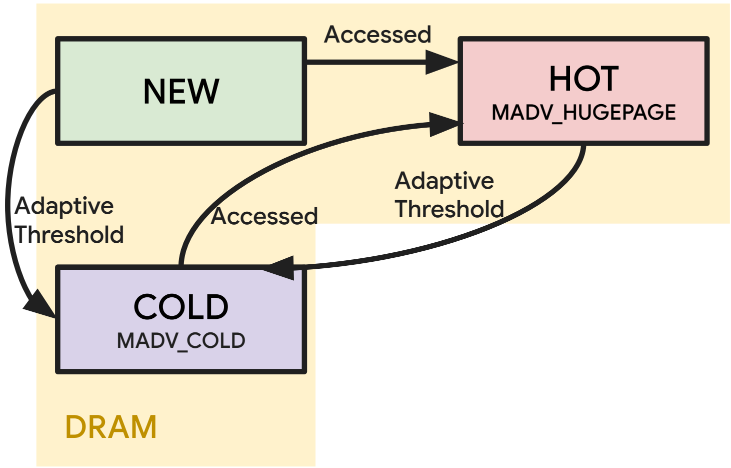
Address	Access	Lock	Heap ID	CIW	ATC

Consecutive **I**nact **W**indow

P2. Grouping Objects by Access Intensity

Adaptive Workload Response

- Promotion Rate (PR) as performance proxy
- Adapting cold threshold to reach target PR
- Proactive reclamation using `MADV_PAGEOUT`



PR = (unique COLD pages accessed / working set size) × (60 / scan interval)

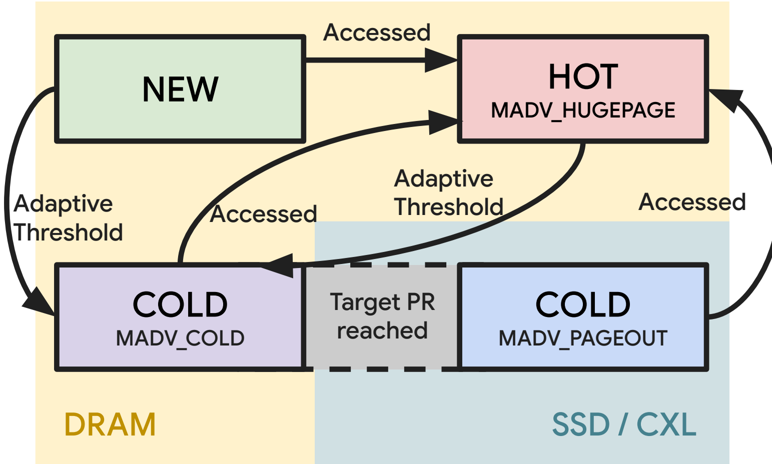
Target Promotion Rate = 1% (based on swap and CXL tiering deployments)

Object Based Address-Space-Engineering

P2. Grouping Objects by Access Intensity

Adaptive Workload Response

- Promotion Rate (PR) as performance proxy
- Linear increase/decrease of cold threshold to reach target PR
- Proactive reclamation using `MADV_PAGEOUT`



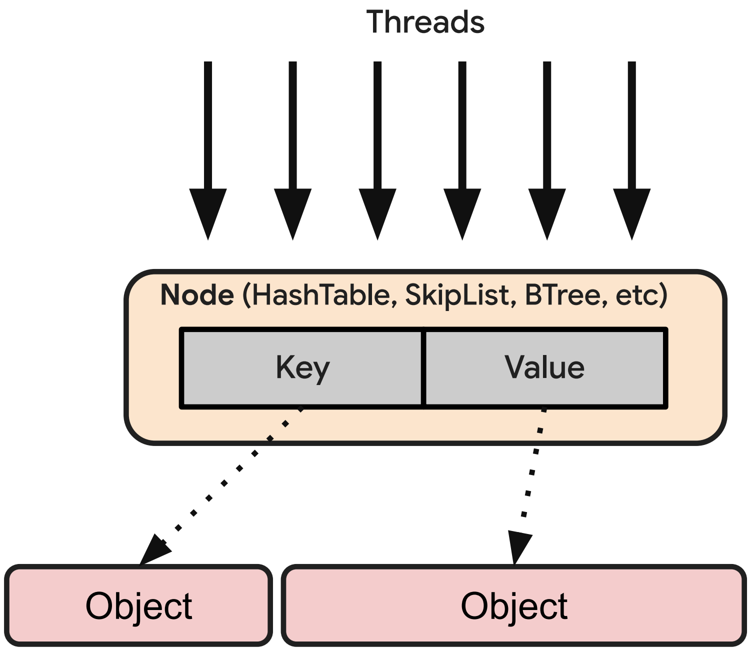
PR = (unique COLD pages accessed / working set size) × (60 / scan interval)

Target Promotion Rate = 1% (based on swap and CXL tiering deployments)

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

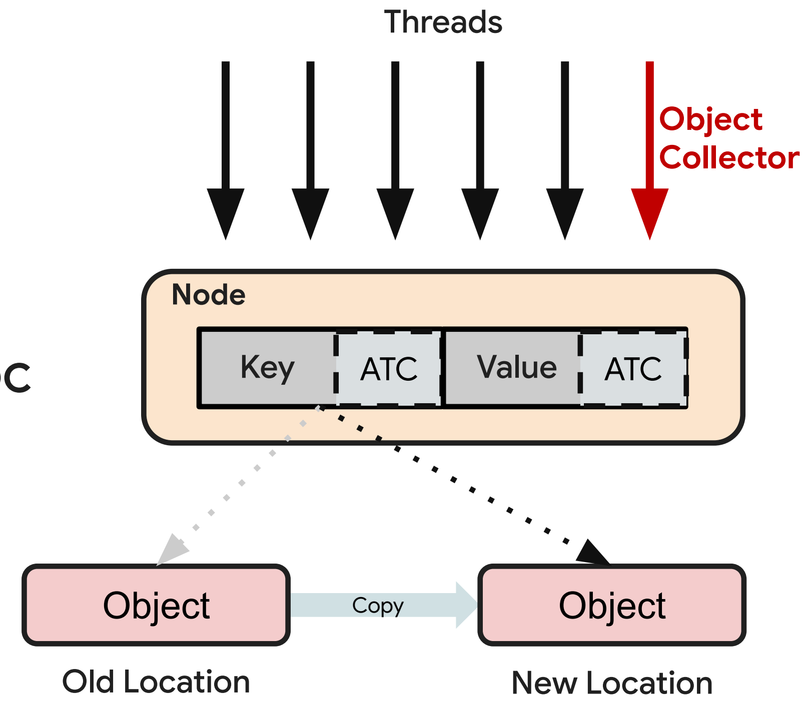


Object Based Address-Space-Engineering

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. **OC** enables ATC during ACTIVE phase.
- Optimistic Object Migration



Outline

- Page Utilization
  A new metric to quantify hotness fragmentation
- Address-Space Engineering
  Active management of virtual address space as a resource
- OBASE
  Dynamically optimize application memory layout
- **Evaluation**

Evaluation

- **Ten** popular pointer based data structures: ART, MassTree, BTree, HashTable, etc
- S… r…
- I… (…
- [illegible] traces from Meta and Twitter
- Overhead, scalability, and adaptive policy studies

Frontend Effectiveness

Does OBASE organize the virtual address space effectively?

Backend Validation

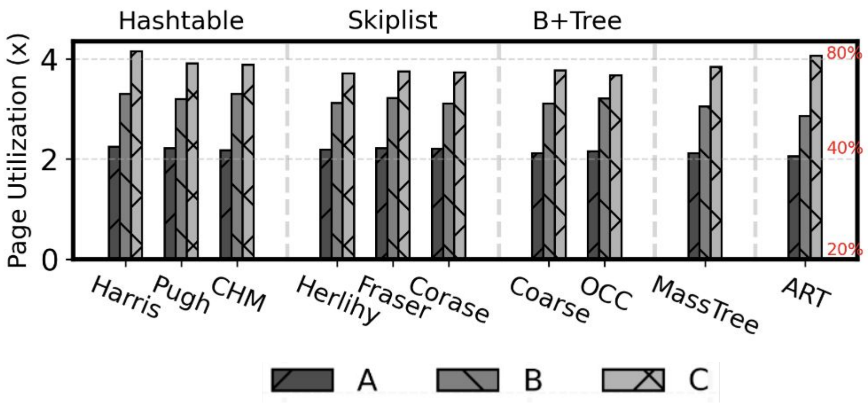
Does OBASE improve memory efficiency of different tiering systems?

	…ores)
SSD	512 GB P4800x SSD
OS	Ubuntu 22.04

Frontend **Effectiveness**

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

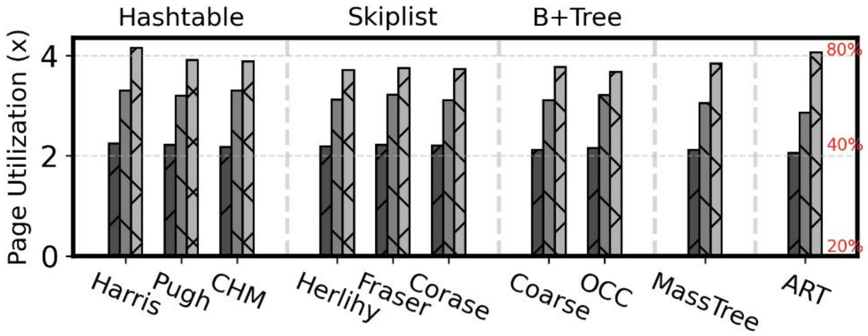
- OBASE **increases page utilization** by
  - **2x** for workload A (50% read 50% write)
  - **3x** for workload B (95% read 5% write)
  - **4x** for workload C (100% read)

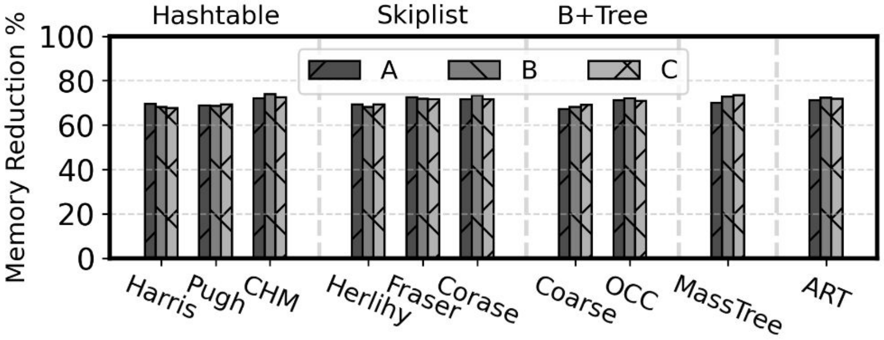


Frontend Effectiveness

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

- OBASE **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)
- OBASE Hinted reduces **memory usage** by up to **70%** through object-level cold detection and heap organization

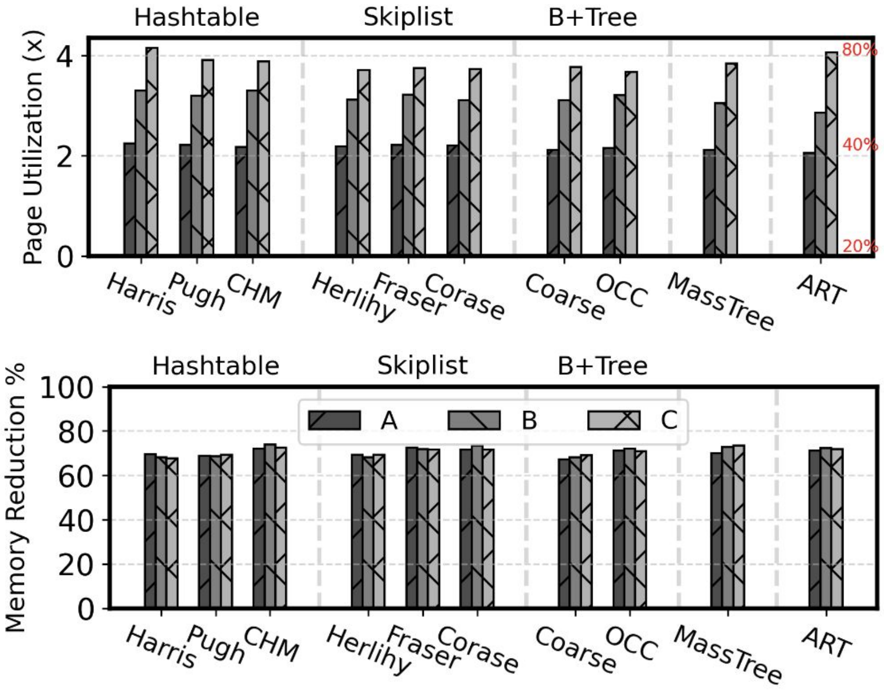




Frontend Effectiveness

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

- OBASE **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)
- OBASE Hinted reduces **memory usage** by up to **70%** through object-level cold detection and heap organization
- OBASE tracking overhead lowers average throughput by **2.5%** and increases P90 latency by **5%**



Backend Validation (Reclamation)

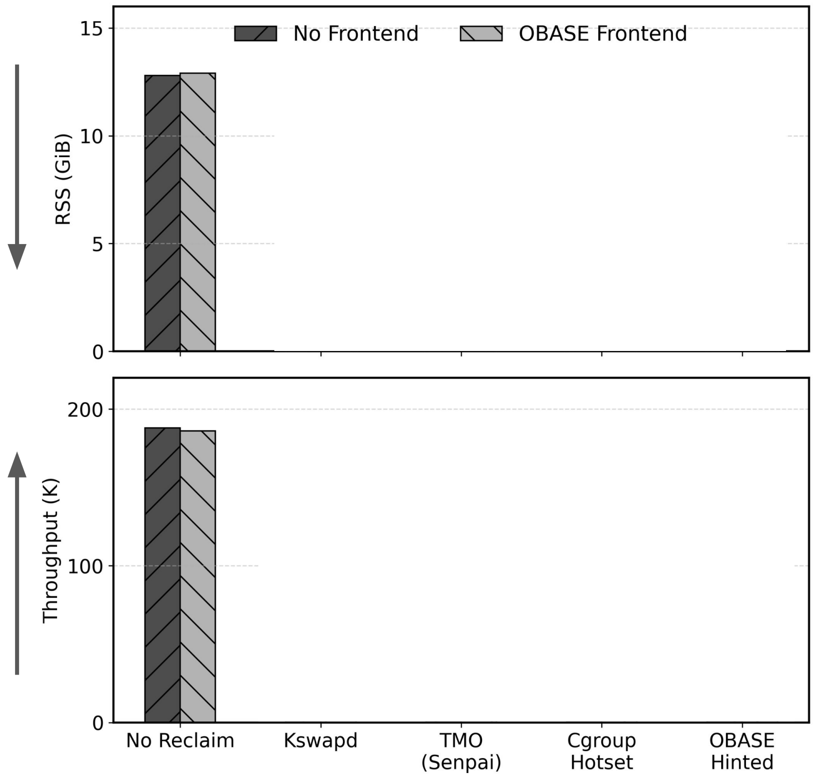
YCSB-C with ~12 GiB footprint and ~4 GiB actively accessed

Reclamation backends:

- **Kswapd**: Reclaims under memory pressure
- **TMO**: Proactively reclaim memory
- **Cgroup hotset**: Cgroup limit set to 4GiB

Standard backends

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



Backend Validation (Reclamation)

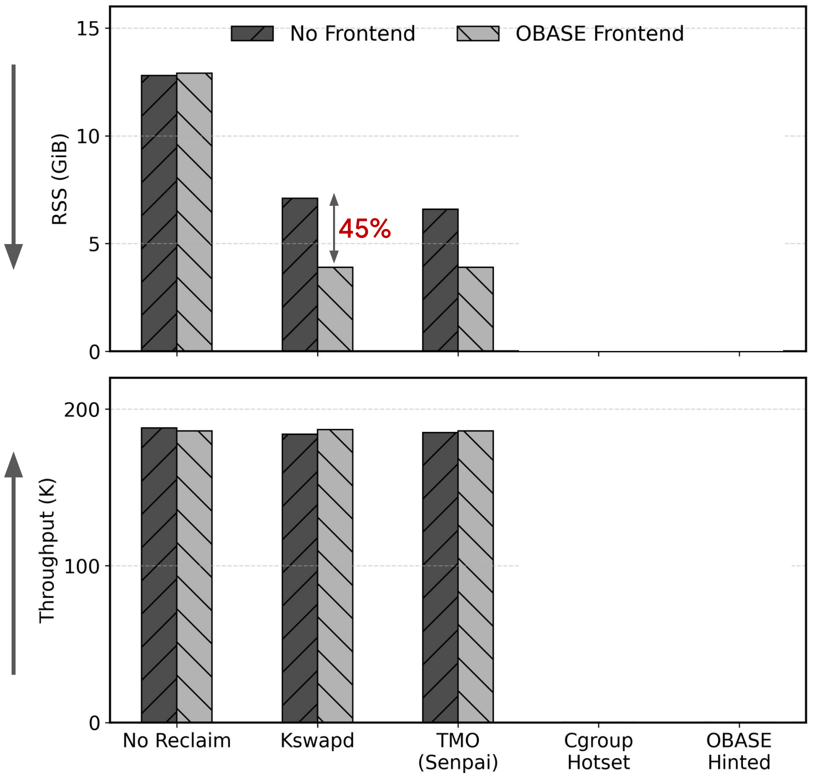
YCSB-C with ~12 GiB footprint and ~4 GiB actively accessed

Reclamation backends:

- **Kswapd**: Reclaims under memory pressure
- **TMO**: Proactively reclaim memory
- **Cgroup hotset**: Cgroup limit set to 4GiB

Standard backends

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



Backend Validation (Reclamation)

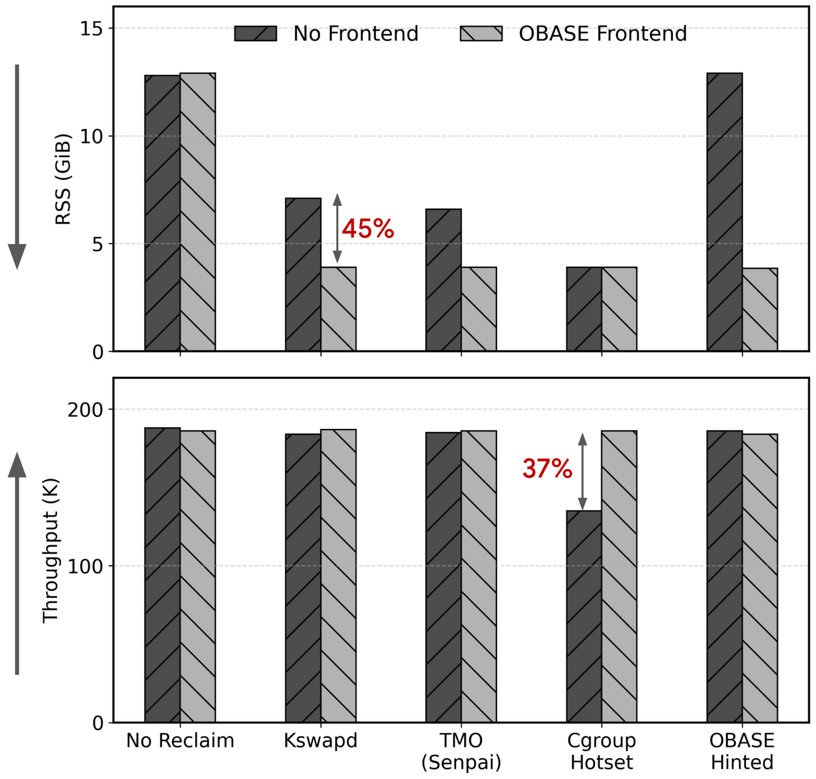
YCSB-C with ~12 GiB footprint and ~4 GiB actively accessed

Reclamation backends:

- **Kswapd**: Reclaims under memory pressure
- **TMO**: Proactively reclaim memory
- **Cgroup hotset**: Cgroup limit set to 4GiB

Standard backends

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



Backend Validation (Reclamation)

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

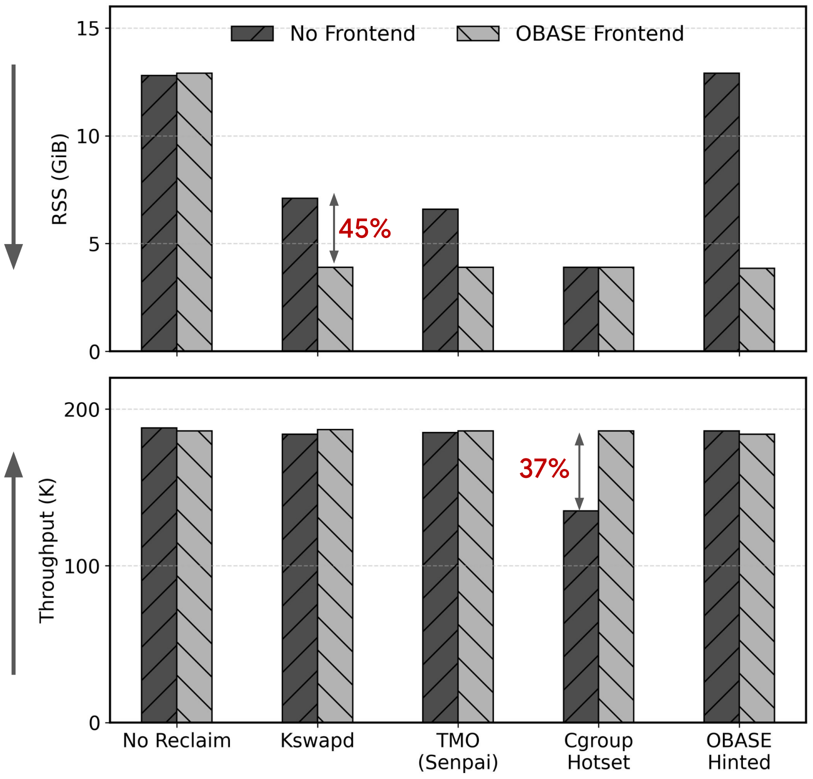
Reclamation backends:

- **Kswapd**: Reclaims under memory pressure
- **TMO**: Proactively reclaim memory
- **Cgroup hotset**: Cgroup limit set to 4GiB

Standard backends

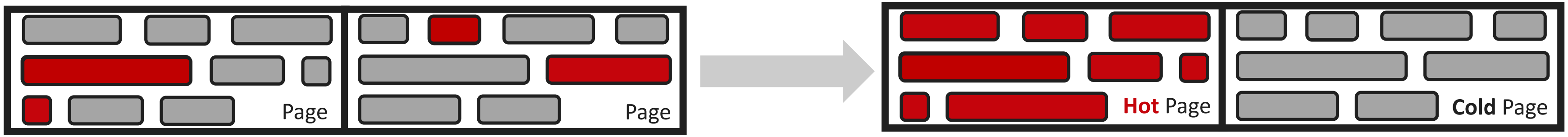
- Sacrifice performance to save memory (cgroup hotset)
- Sacrifice savings to preserve performance (kswapd and TMO)

OBASE achieves max memory savings with no performance degradation



OBASE Summary

- **Semantic gap** between application and OS memory model
- **Hotness fragmentation** is widespread
- Address-Space Engineering for improving page utilization
- **OBASE** as an approach for Address-Space Engineering
- Achieved higher memory savings across diverse data structures and multiple tiering backends on realistic workloads

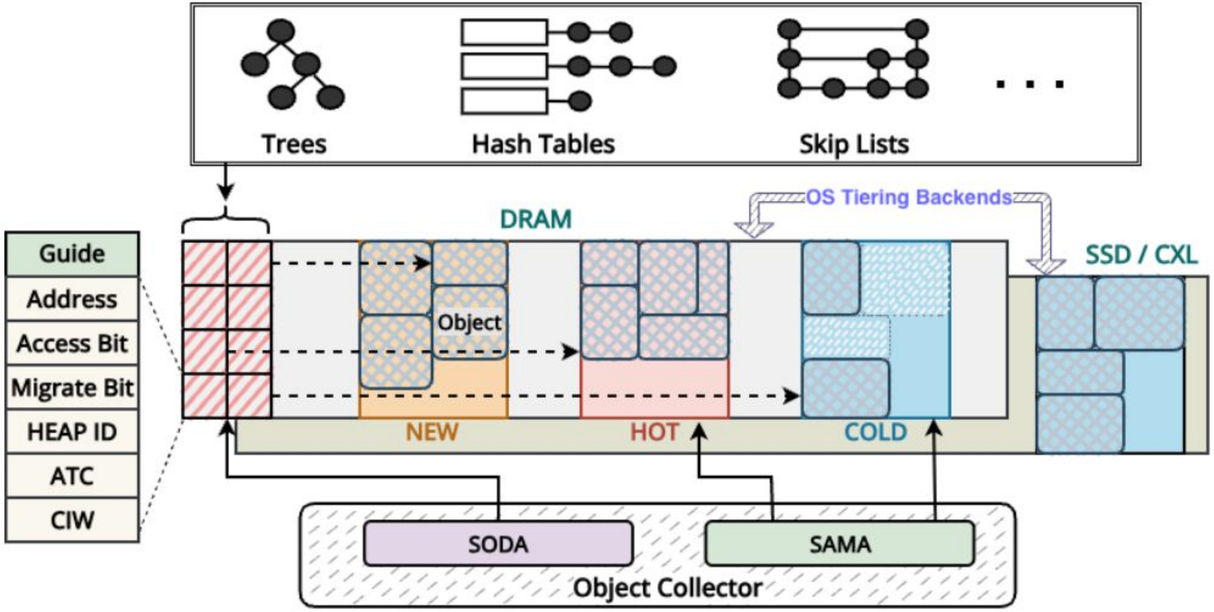


Hotness Fragmented Virtual Address Space

Hotness Organized Virtual Address Space

OBASE Limitations

- **Lack of pointer stability:** Invalidates cached pointers like Abseil and STL structures
- **Unique Object Ownership:** When annotated users implicitly assert that it has exclusive access over the object. Similar to `std::unique_ptr` ownership model
- **Incompatibility with pointer arithmetic:** Unsuitable for data structures likes arrays and matrices
- **Language support restrictions:** Requires dereference overloading
- **Requirement for explicit annotations:** Users must specify which objects are managed



The Object Collector monitors access via Guides and SODA, migrates objects using a spatially-aware allocator, and presents a re-organized address space to the OS backend.

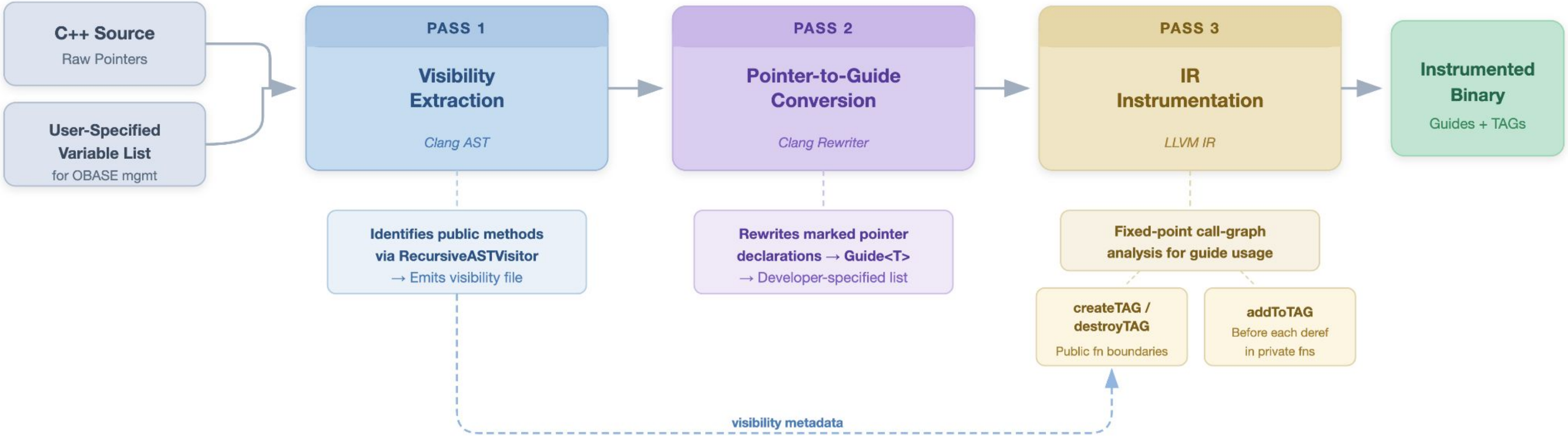
Guide Metadata Encoding

- 48 bits - Address
- 7 bits - Active Thread Count (128 concurrent threads)
- 5 bits - Consecutive Inactive Windows (32 windows): 1hr+ at 120s interval
- 2 bits - Heap ID (3 heaps)
- 2 bits - Access and Migration lock flags

Sparse Object Data Activity (SODA)

- SODA divides the address space into blocks and each block has a bunch of words. Each word is a 64-bit integer, and each bit represents 8 bytes of the address space.
- SODA tracks guide pointers rather than object addresses, it remains valid across relocations - the guide's address does not change when object moves.
- With 128 KiB block size and 2^44 gbits, the lower bound is 40 MiB and the upper bound is 682 GiB.

OBASE Compiler Passes for Guide Management

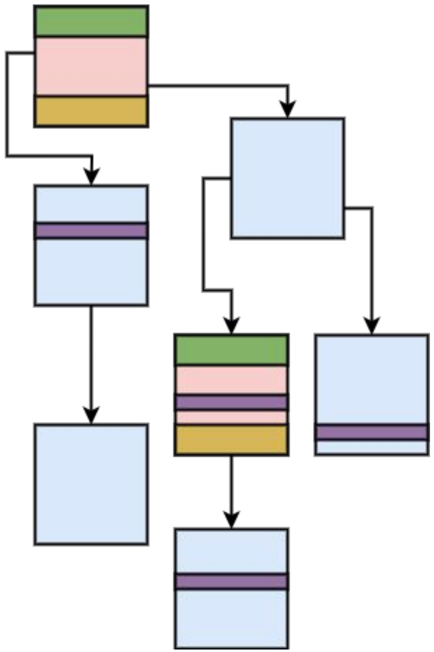


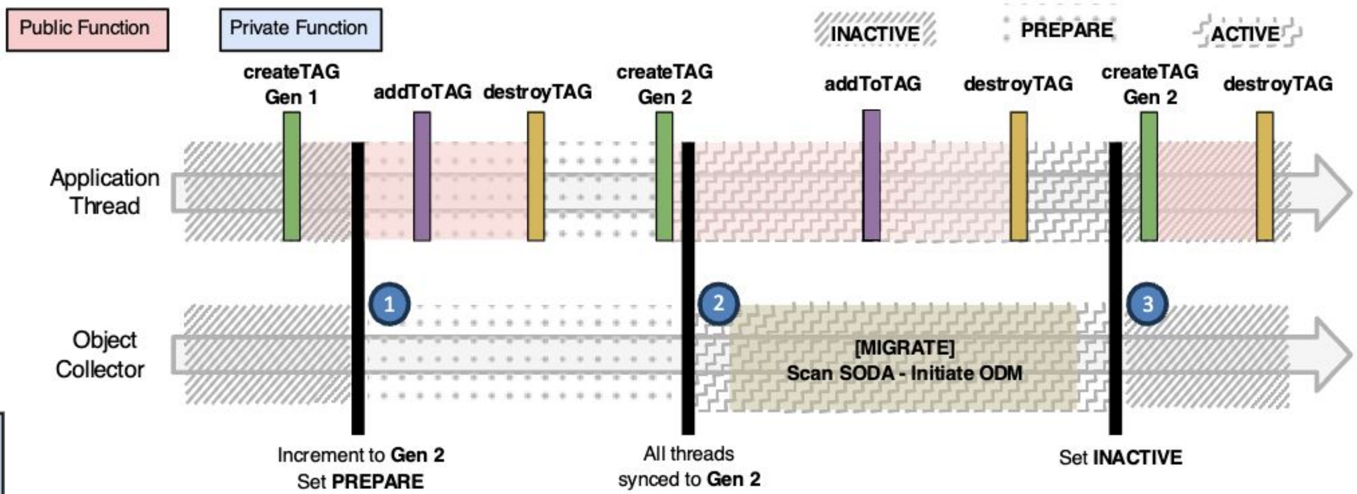
OBASE Compiler Passes for Guide Management

- **Pass 1 - Visibility Extraction (Clang Front-End)**
  - A `RecursiveASTVisitor` inspects C++ class definitions to identify public methods (e.g., get, set, delete) that serve as data-structure entry points.
  - Outputs a visibility file consumed by Pass 3, so TAGs are only created at public boundaries.
- **Pass 2 - Pointer-to-Guide Conversion (Clang Rewriter)**
  - Developers provide a list of pointer field names to convert; the pass rewrites their declarations and usages into `Guide<T>`.
  - Gives precise developer control over which objects OBASE manages while automating all syntactic changes.
- **Pass 3 - IR Instrumentation (LLVM IR)**
  - Performs a fixed-point analysis: first scans for direct guide usage, then propagates through the call graph to discover indirect usage.
  - Combines this with Pass 1's visibility data to classify every function, then inserts:
  - `createTAG` / `destroyTAG` at entry/exit of public functions that touch guides.
  - `addToTAG` before each guide dereference in private functions (no new TAG creation).

OBASE Code Transformation

```
void HashMap::get(int idx) {
    createTAG();                          // public entry
    Guide<char>& v = buckets[idx];        // HashMap member
    addToTAG(&v);                         // before deref
    if (*v > 37) { destroyTAG(); return; }
    addToTAG(&v);   *v = 42;
    addToTAG(&v);
    std::cout << "Value: " << *v << std::endl;
    destroyTAG();                         // public exit
}
```



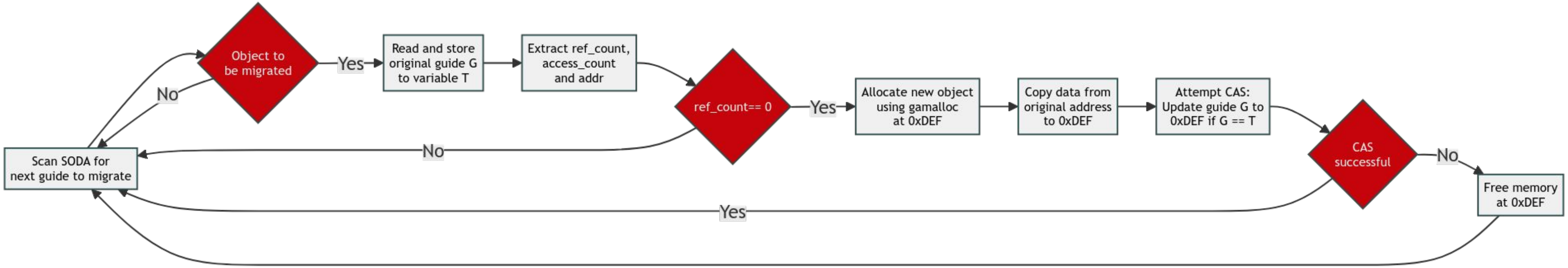


Start of Public Function	Guide Access	End of Public Function

`createTAG:`
`publicNestingLevel++`
`TAI[slot] = 1`
`threadLocalGen = GlobalGen`
	`if (TAG_state != INACTIVE`
	`&& addToTAG(Guide)):`
	`incrementATC(Guide)`
		`destroyTAG:`
		`publicNestingLevel--`
		`if (!publicNestingLevel):`
		`decrementAndClear(TAG)`
		`TAI[slot] = 0`

The left side shows the instrumented call graph, code is inserted in three scenarios as shown. Public functions create and destroy Thread-local Active Scope Guards (TAGs), maintaining nesting levels and registering the thread, in the Thread Activity Index (TAI). Guides increment active reference counts only if added to the TAG. Reference counts are decremented only when the outermost public function exits. Active Thread Count (ATC) is enabled only during PREPARE and ACTIVE states

Optimistic Data Migration - ACTIVE



Algorithm 2 Optimistic Data Migration (ODM)

```
procedure MigrateObject(ptr, targetHeap)   ▷ ptr: address of guide slot
    guide ← AtomicLoad(ptr)   ▷ Load object guide
    if RefCount(guide) > 0 then
        return false   ▷ Object in use
    end if
    srcAddr ← ExtractAddr(guide)
    srcHeap ← ExtractHeapType(guide)   ▷ Get current heap type
    SetMigrationLock(ptr)   ▷ CAS; mark as migrating
    size ← GetSize(srcAddr)
    dstAddr ← samalloc(size,targetHeap)
    if dstAddr = null then
        ClearMigrationLock(ptr)
        return false   ▷ Allocation failed
    end if
    Copy(dstAddr,srcAddr,size)   ▷ Copy data
    newGuide ← CreateGuide(guide,dstAddr,targetHeap)   ▷ Inherits lock from guide metadata
    ClearMigrationLock(newGuide)   ▷ Local clear of inherited lock; commit via CAS below
    success ← AtomicCAS(ptr,guide,newGuide)
    if success then
        Free(srcAddr,srcHeap)   ▷ Release old memory
        return true
    else
        Free(dstAddr,targetHeap)   ▷ Rollback
        ClearMigrationLock(ptr)   ▷ CAS; release lock we set
        return false   ▷ Migration failed, guide changed
    end if
end procedure
```

Time	Actor	Action	ATC	Lock	Outcome
t_0	OC	read guide	0	0	eligible
t_0'	OC	CAS: set lock	0	1	copying begins
t_1	Thread	dereference	1	0	lock cleared
t_2	OC	CAS: commit	*mismatch*		**aborted**
If no thread intervenes between t_0' and t_2:					
t_2'	OC	CAS: commit	0	0	**success**

Table 5.1: **Race between migration and concurrent access.** A dereference at t_1 modifies the guide, causing the OC's commit CAS to fail. When no thread intervenes, both CAS operations succeed and the object moves.

Tiering background

AutoNUMA - *Reactive, recency-based, no demotion*

- Default Linux NUMA balancing; periodically unmaps a sample of pages and uses the resulting minor faults as a hotness signal.
- A single access is enough to trigger promotion - works well when pages are clearly hot or cold, but over-migrates under transient or shifting access patterns.
- Has no proactive demotion path; pages only leave the fast tier when general memory pressure forces standard reclamation.

TPP (Transparent Page Placement) - *Production-tuned, hysteresis filtering*

- Demotes cold pages via migration, preserving byte-addressable access on the slow tier.
- Decouples allocation from reclamation with separate watermarks, keeping headroom in the fast tier for promotions without stalling allocators.
- Adds a hysteresis check on top of NUMA hint faults: a page must be in the active LRU list *and* fault again to be promoted, filtering out one-off accesses that trip up AutoNUMA.

Memtis - *Adaptive threshold, hardware-sampled*

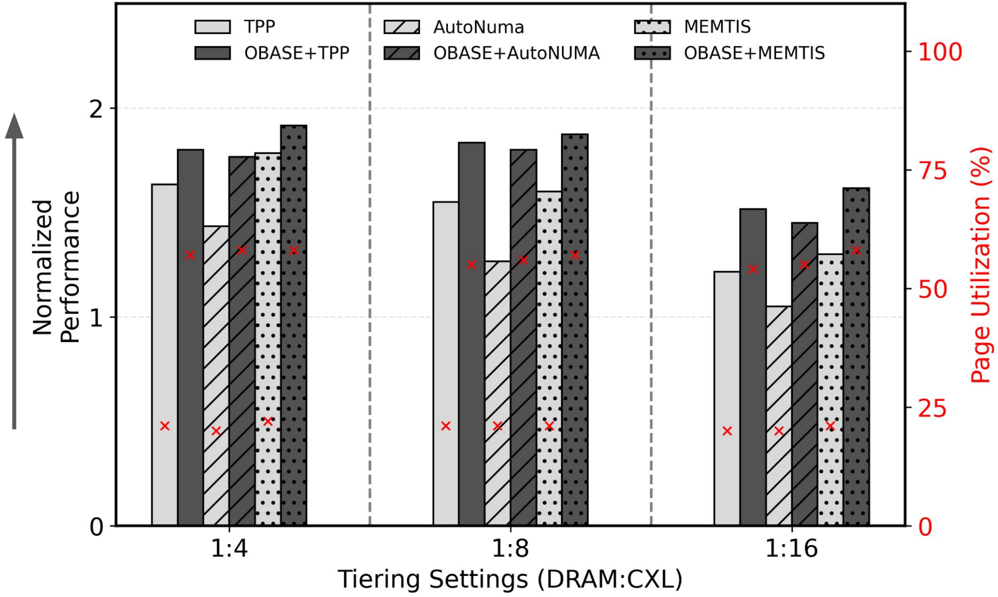
- Uses Intel PEBS (Processor Event-Based Sampling) instead of page faults, giving low-overhead access counts at sub-page granularity.
- Maintains a histogram of access counts in exponential bins and dynamically computes a hot threshold so the fast tier is always filled with exactly the hottest pages - no static "promote after N accesses" cutoff.
- Can split huge pages and promote only the hot 4 KiB subpages, reducing fast-tier waste from mixed-temperature huge pages.

Backend Validation (Tiering)

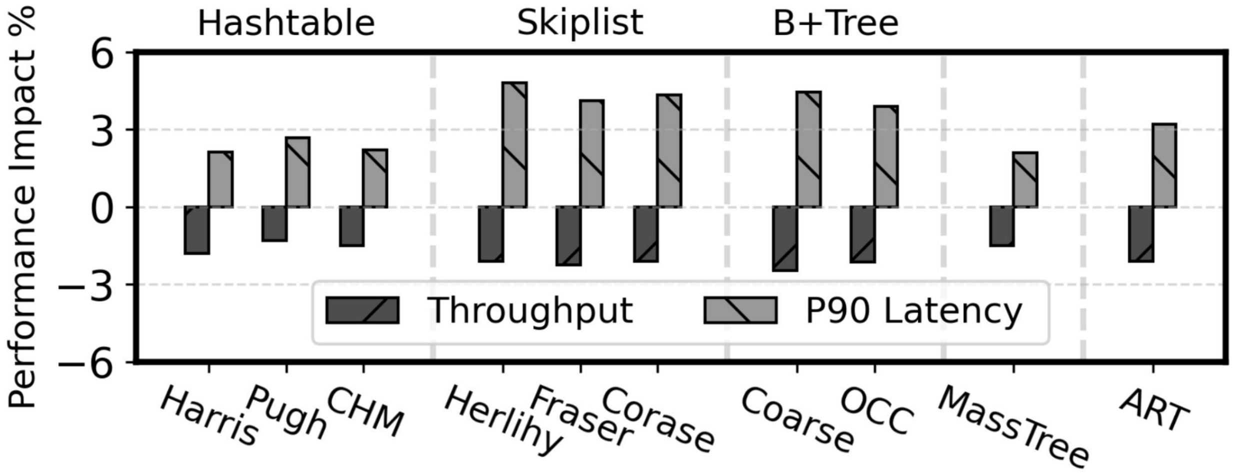
YCSB-B with ~67 GiB footprint (50M - 30B key and 1KiB value)

- Tiering backends: TPP, AutoNuma, MEMTIS
- DRAM:CXL ratio across three configurations: 1:4 (14.8 GiB DRAM), 1:8 (7.4 GiB DRAM), and 1:16 (3.9 GiB DRAM)
- Without OBASE hotset spans 16.3 GiB pages but with OBASE it spans 6.33 GiB

OBASE at ratio 1:X performs comparably to baseline at 1:(X/2)



Performance overhead

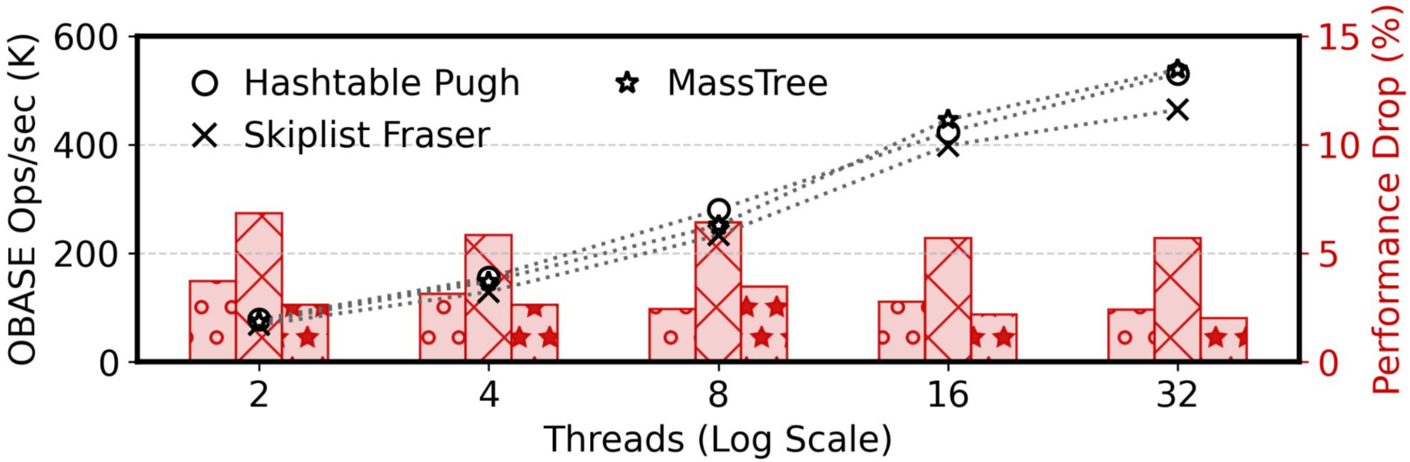


Throughput and p90 latency overhead across data structures

The overhead has two main sources:
(1) A tagged-pointer read-modify-write on each guide dereference
(4-5 ns, comparable to an L1 cache hit)
(2) TAG/ATC bookkeeping during ACTIVE migration epochs.

The Object Collector runs in a dedicated thread and consumes less than 5% of CPU time.

Scalability

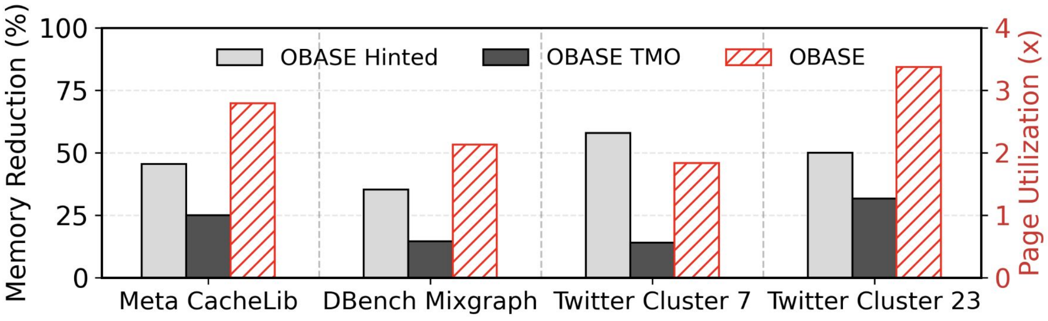


Bars show absolute throughput; markers show overhead relative to baseline.
Overhead remains bounded at 1–8% with no upward trend.

Guide metadata updates target per-object state (no cross-thread contention), the test-and-set optimization skips redundant writes for hot objects, TAGs are thread-local, and ATC increments occur only during brief ACTIVE epochs.

Production trace results

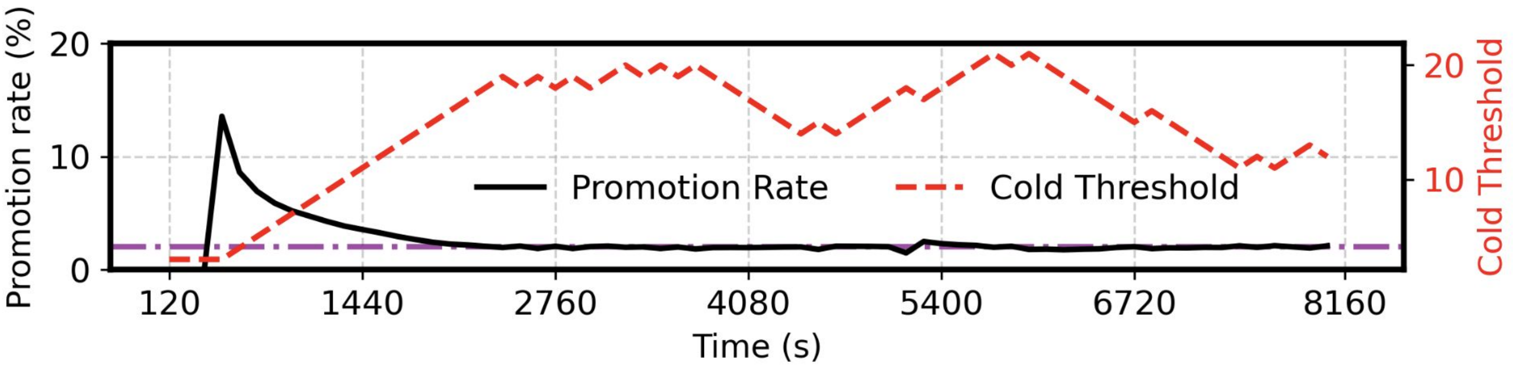
- **Meta CacheLib:** Read-heavy (83% GET) with gradually shifting popular keys.
- **DBench Mixgraph:** Models Meta's ZippyDB with key-range locality across 30 prefixes. Read-heavy (85% GET, 14% PUT, 1% SEEK).
- **Twitter Cluster 7:** High skew (α=1.07) with small, concentrated working set of reads and writes.
- **Twitter Cluster 23:** Write-heavy (31% SET, 30% INCR) with low skew (α=0.274) and deletes.



OBASE Hinted reduces RSS by 36–58% compared to no reclamation

Adding OBASE to TMO provides 15–30% additional savings

Cold threshold adaptation (Meta CacheLib)



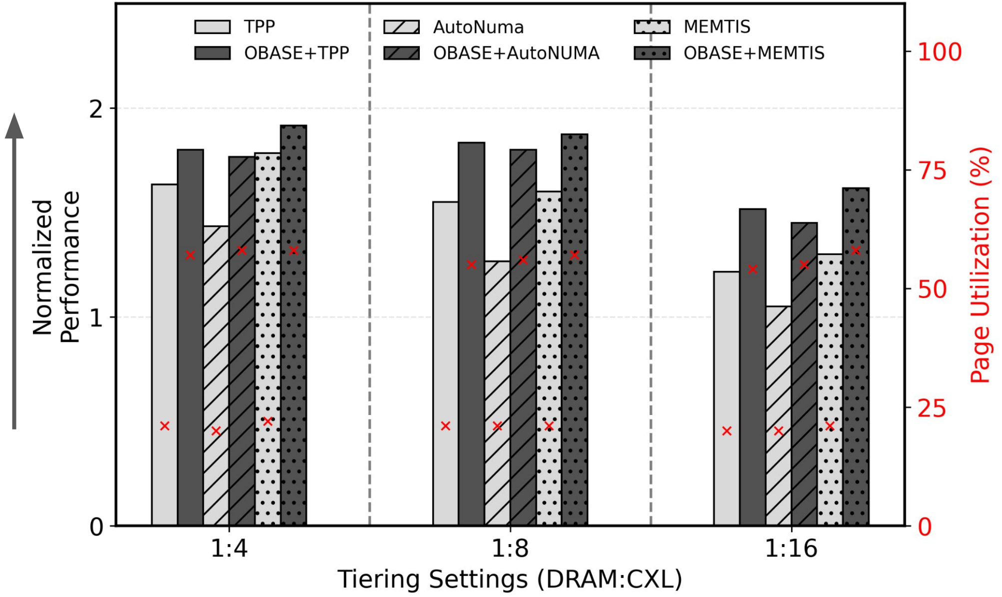
Additive Increase Additive Decrease of Cold Threshold

PR = (unique COLD pages accessed / working set size) × (60 / scan interval)
Target Promotion Rate = 1% (based on CXL tiering deployments)

Backend Validation (Tiering)

YCSB-B with ~67 GiB footprint (50M - 30B key and 1KiB value)

- Tiering backends: TPP, AutoNuma, MEMTIS
- DRAM:CXL ratio across three configurations: 1:4 (14.8 GiB DRAM), 1:8 (7.4 GiB DRAM), and 1:16 (3.9 GiB DRAM)
- Without OBASE hotset spans 16.3 GiB pages but with OBASE it spans 6.33 GiB



OBASE at ratio 1:X performs comparably to baseline at 1:(X/2)

Related Work

- **Object-Level Management:**
  - AIFM and MIRA operate at object granularity but focus 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
- **Cache Line Hardware Tiering:**
  - Only reaches coherent, byte-addressable far memory. FMM caps at a 1:1 local:CXL ratio and never shrinks RSS, so so it offers nothing for the zswap/SSD tiers
  - Memstrata operates on page granular hotness (DAMON scans)