# Dual-Density

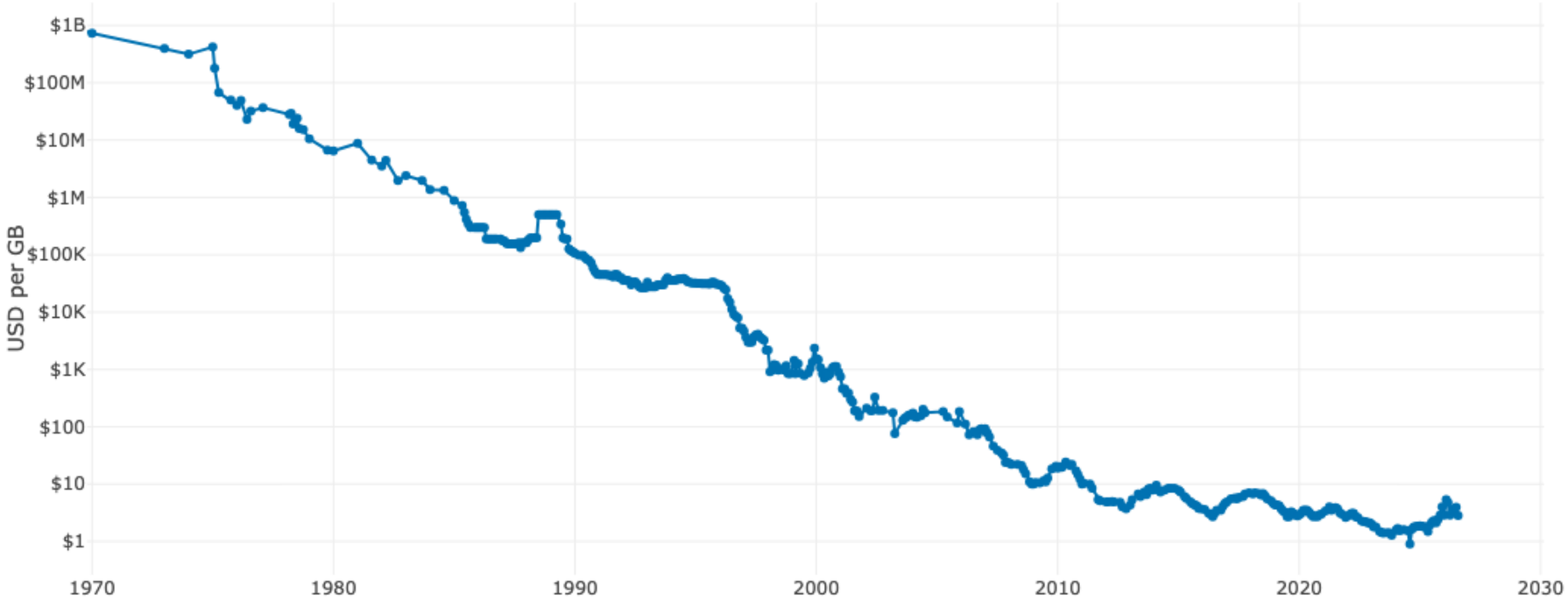
## Trading Cycles for Memory in the Buffer Pool

**Christos Laspias**, Andrew Pavlo, Jignesh M. Patel

under review for VLDB 2027

# Motivation

## The 40-Year Assumption




DRAM Price per GiB over Time

source : https://dam.stanford.edu/memory-prices.html

# Motivation

## The DRAM Scarcity

- DRAM capacity scaling has stagnated (physical limits)
- DRAM is getting more expensive (AI-driven demand)
- Data keeps growing exponentially
- **Purely in-memory systems are often expensive or impossible to deploy**

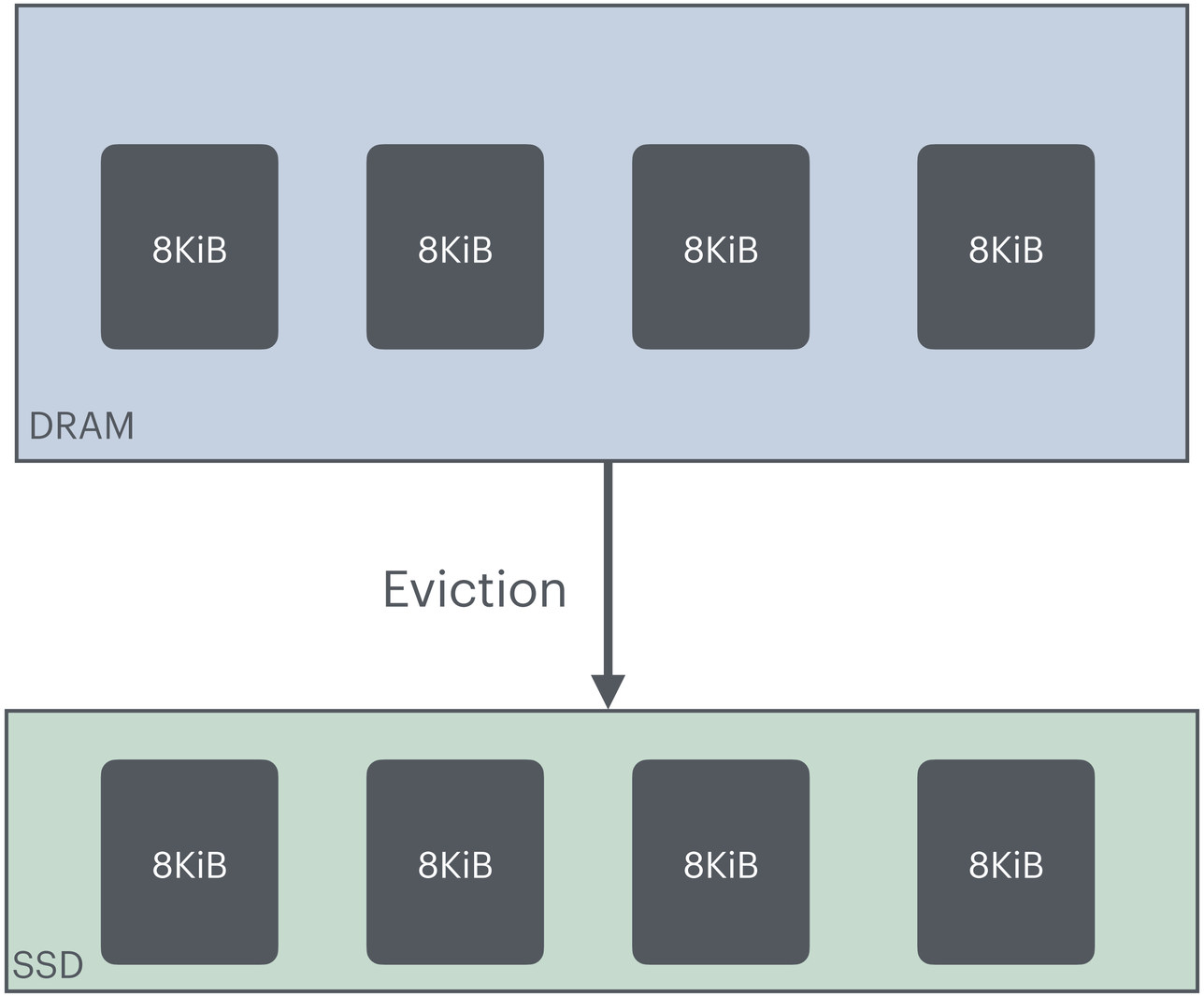
# OLTP

## The Cost of Going to Disk

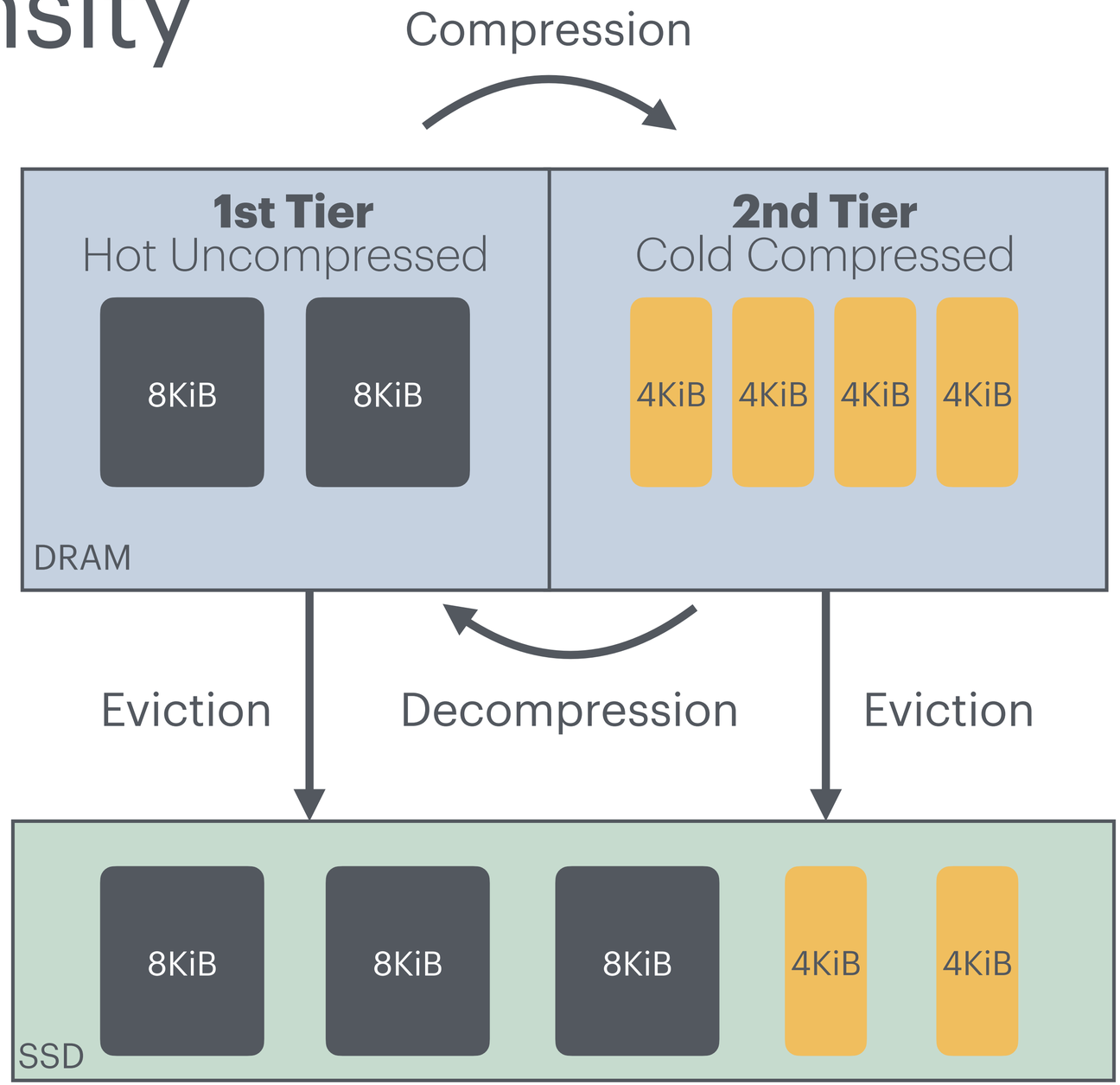
- OLTP comprises of point lookups & random-access updates
- Performance is sensitive to available DRAM
- A page miss triggers synchronous fetch from NVMe, on a transaction's critical path
- SSD is 1000x slower than DRAM
- The CPU sits idle - Core counts increase every year
- How can the DBMS exploit that?

Use **CPU cycles** to keep more data **in memory** in **compressed form**

# Buffer Manager



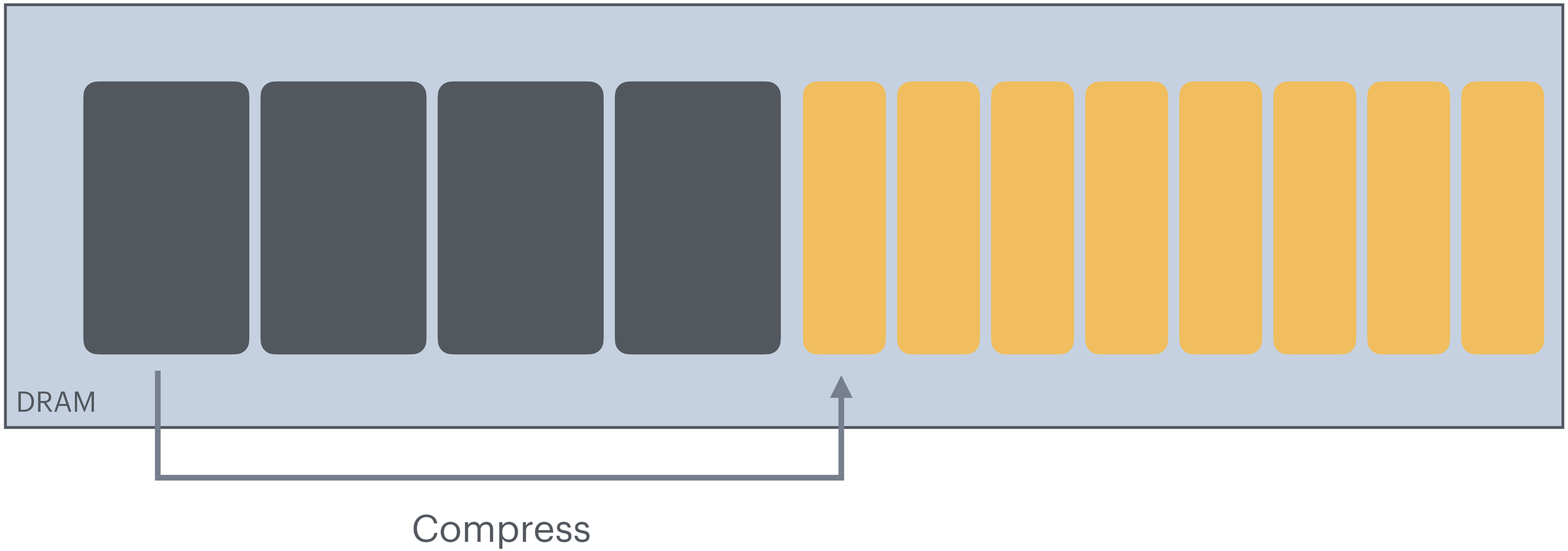
# Dual-Density



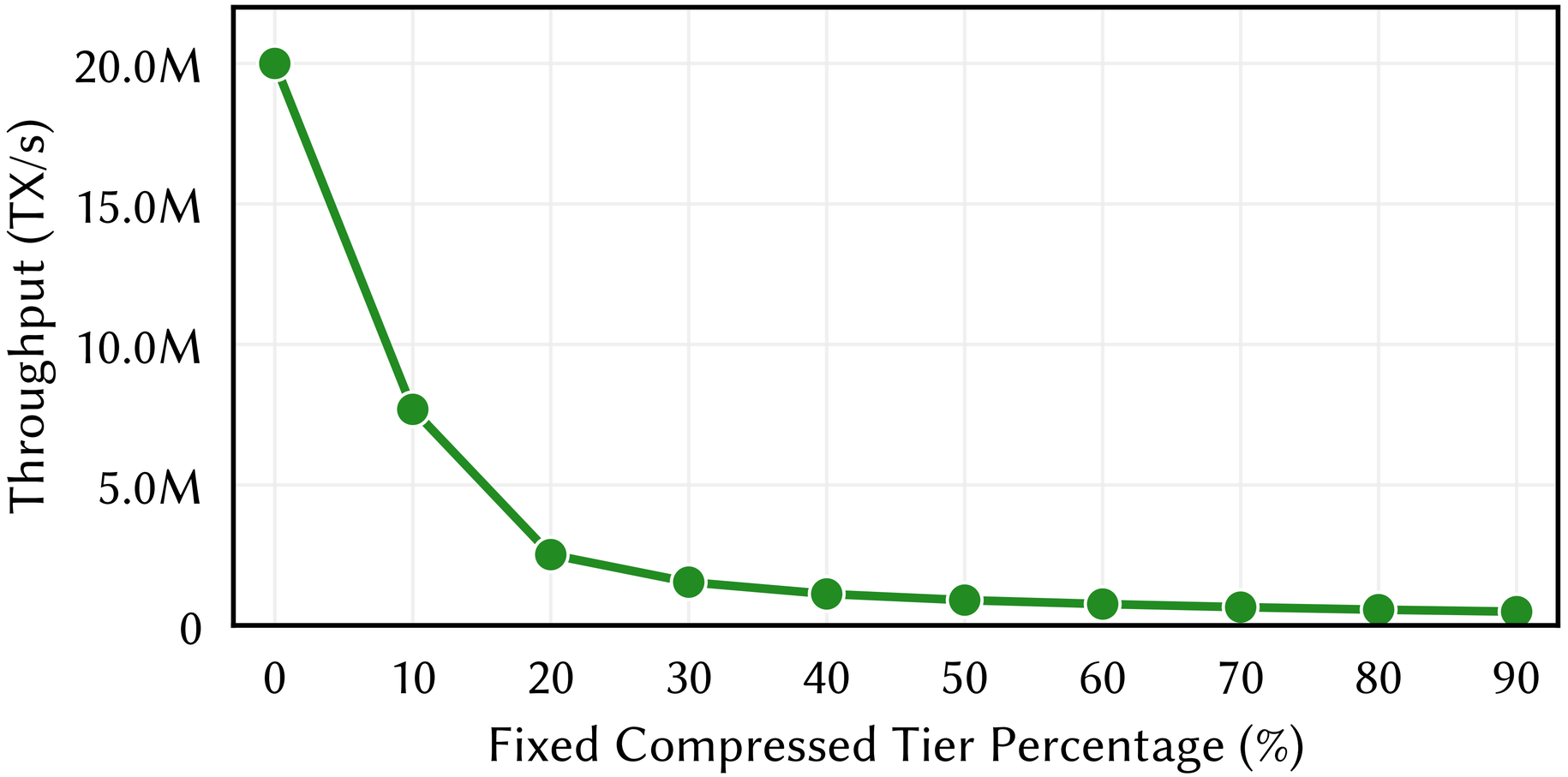
# Existing Buffer Manager

# Existing Buffer Manager

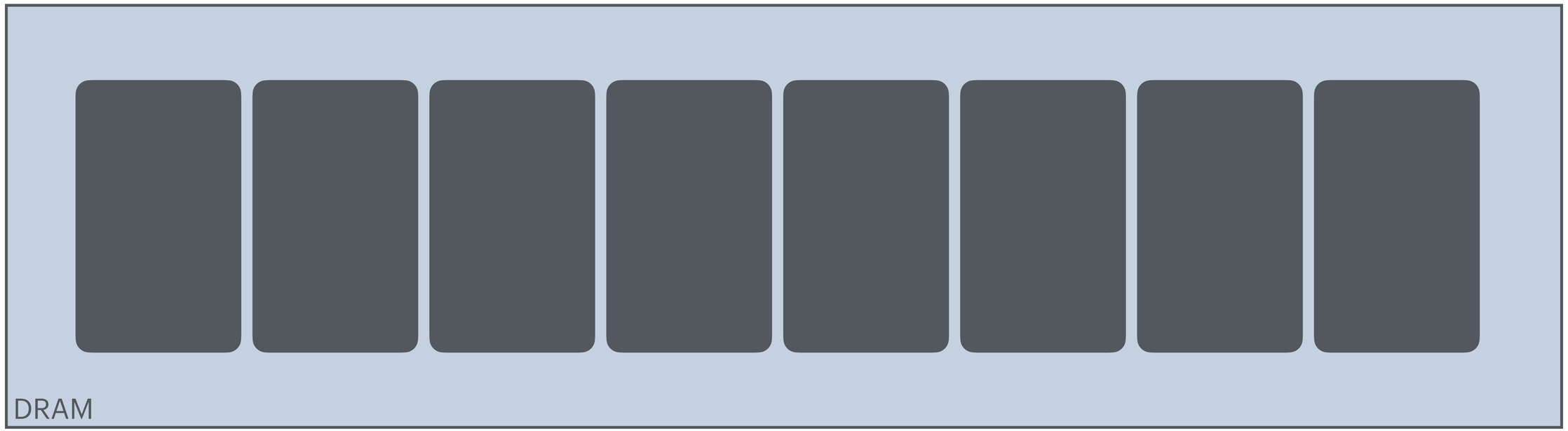
# Static Tier Split



# The Problem with Static Tier Sizing

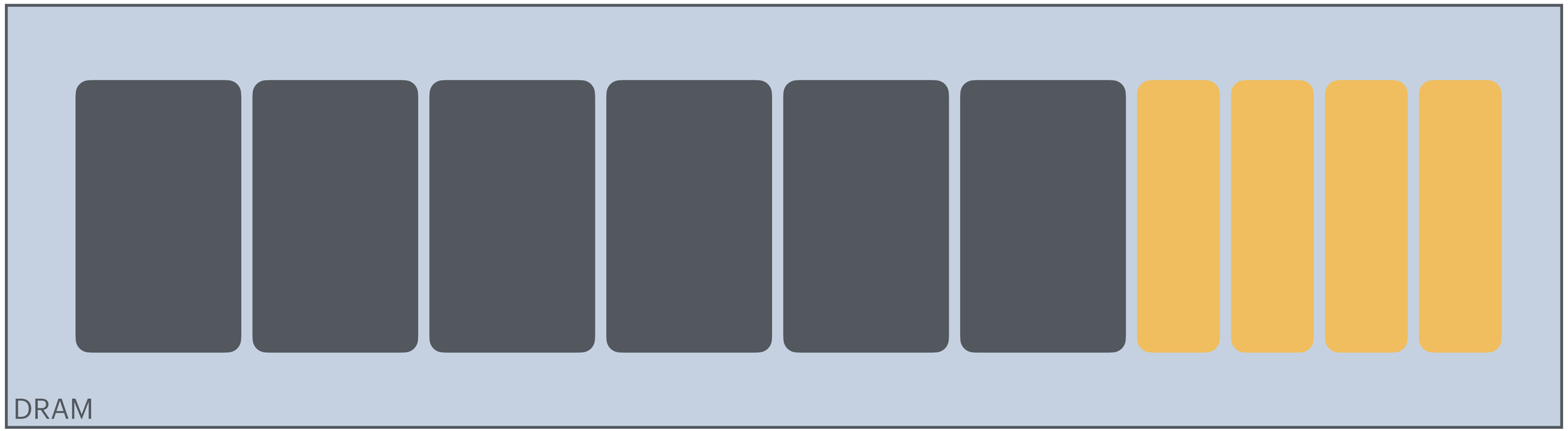


# Adaptive Tier Sizing



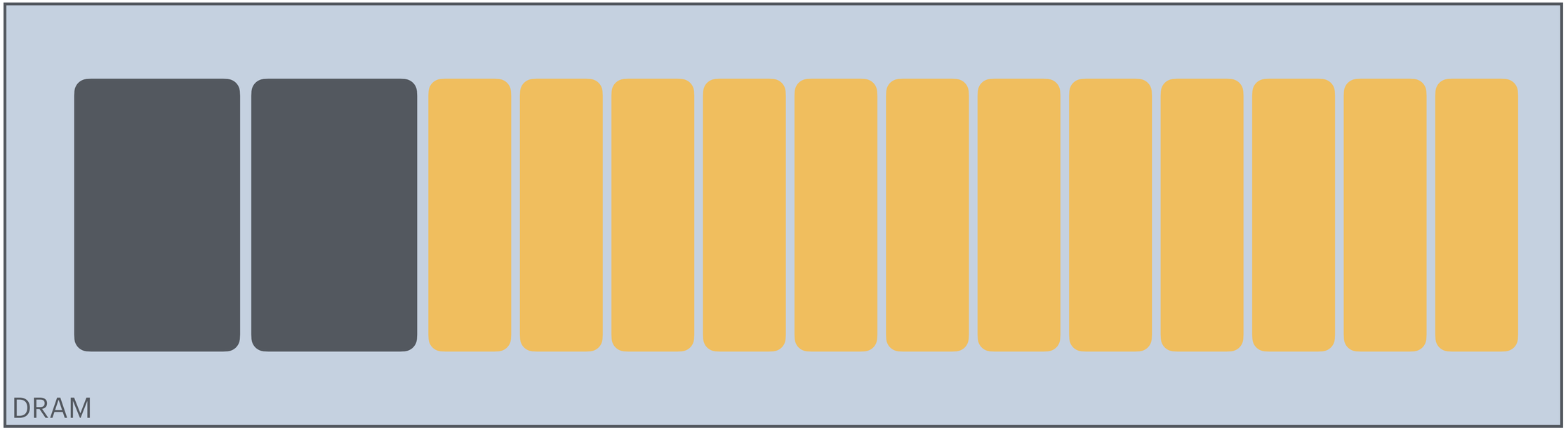

Number of Cached Pages : **8**

# Adaptive Tier Sizing



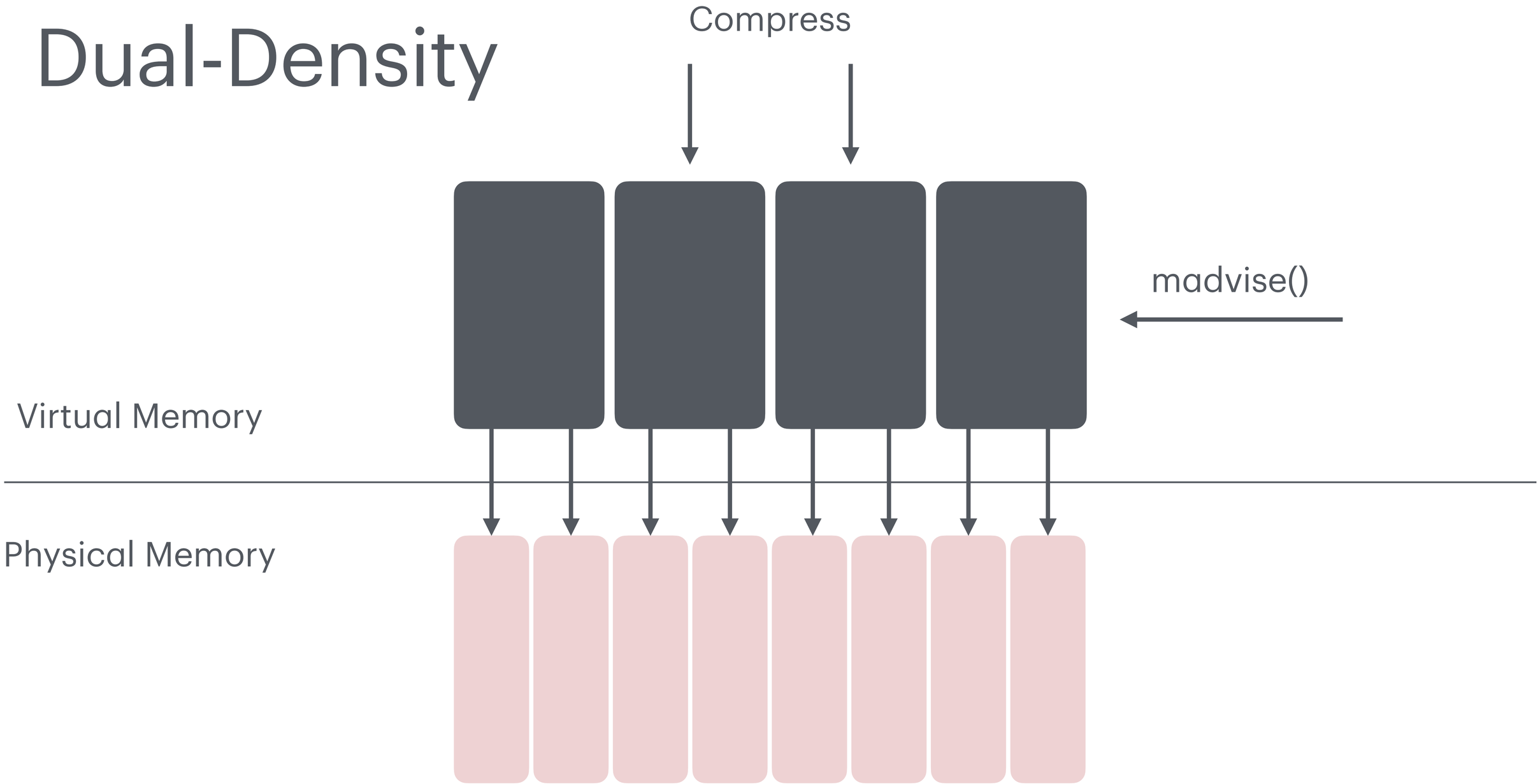

Number of Cached Pages : **10**

# Adaptive Tier Sizing

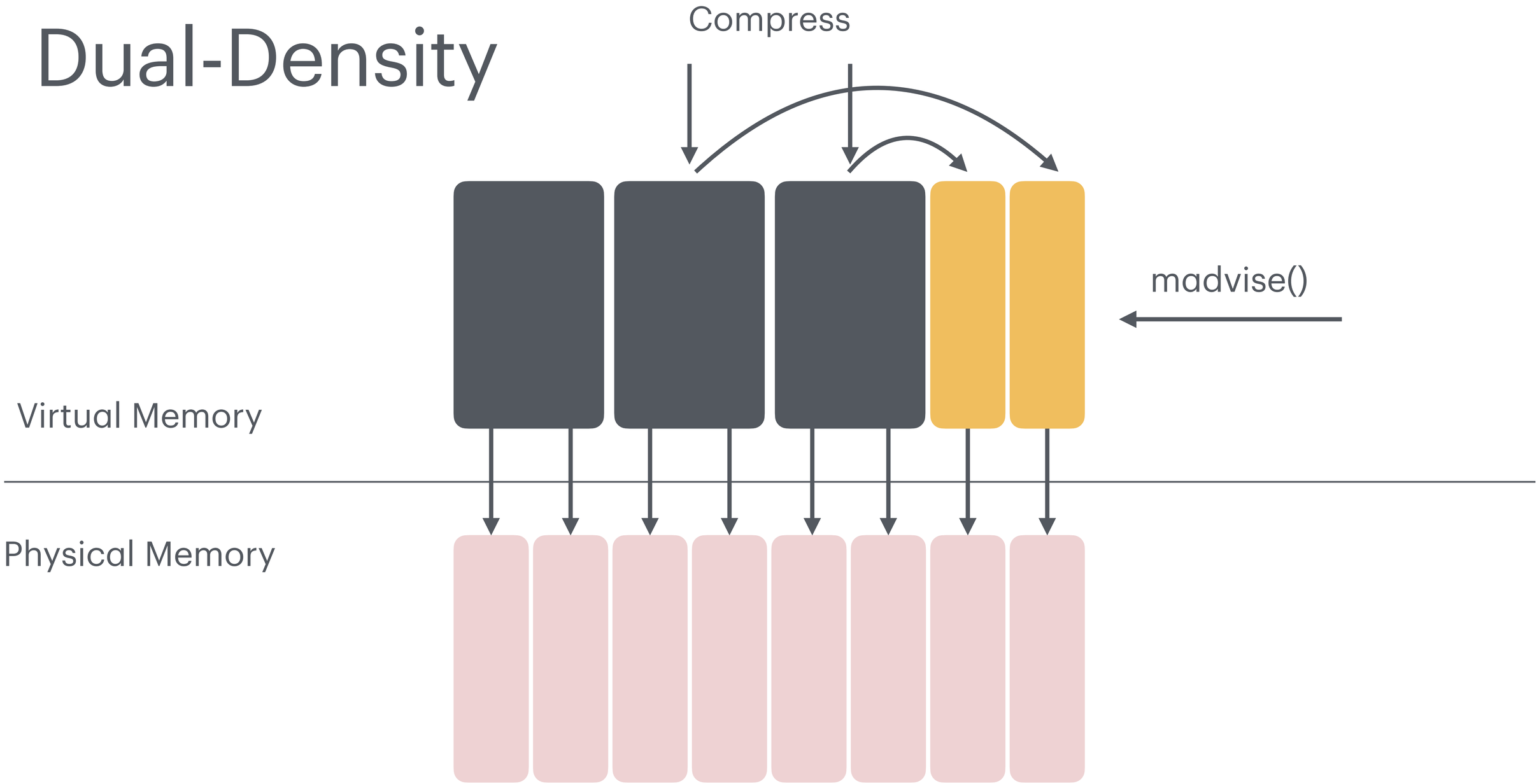



Number of Cached Pages : **14**

# Dual-Density



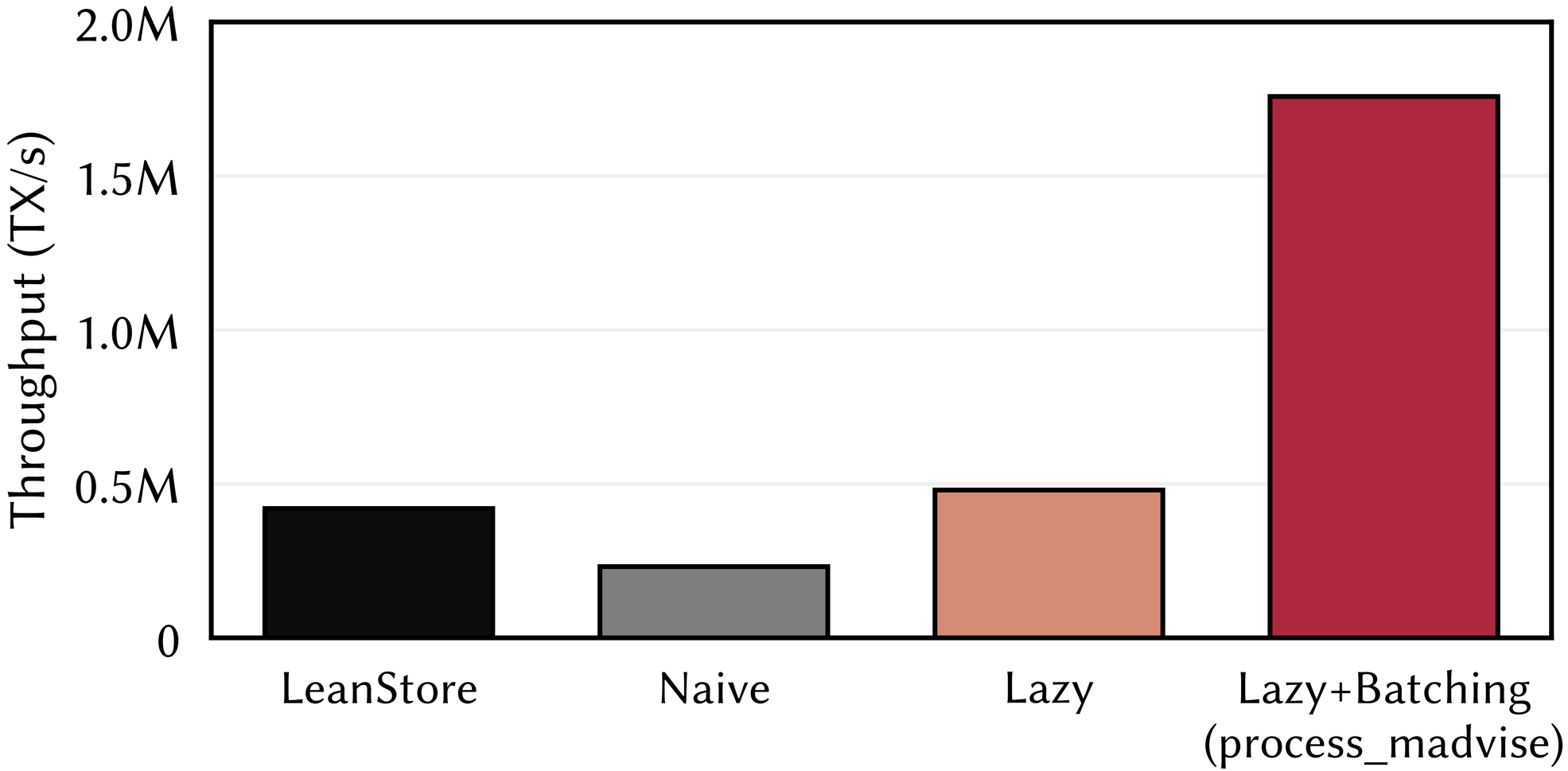
# Dual-Density



# Efficient Tier Rebalancing

- **madvise()** is expensive ~1us per call
- **madvise()** triggers TLB Shootdowns
- Dual-Density needs to call **madvise() THOUSANDS** of times per second
- Naively unmapping wastes all CPU cycles in page-table work
- Two optimizations :
  - Lazy unmapping
  - Batching

# Impact of Memory Optimizations



# Additional Details

- **Hotness tracking** mechansim to identify **hot** and **cold** pages
- Page state is managed through **pointer-swizzling**
- Compression is **latch-free** without blocking other transactions

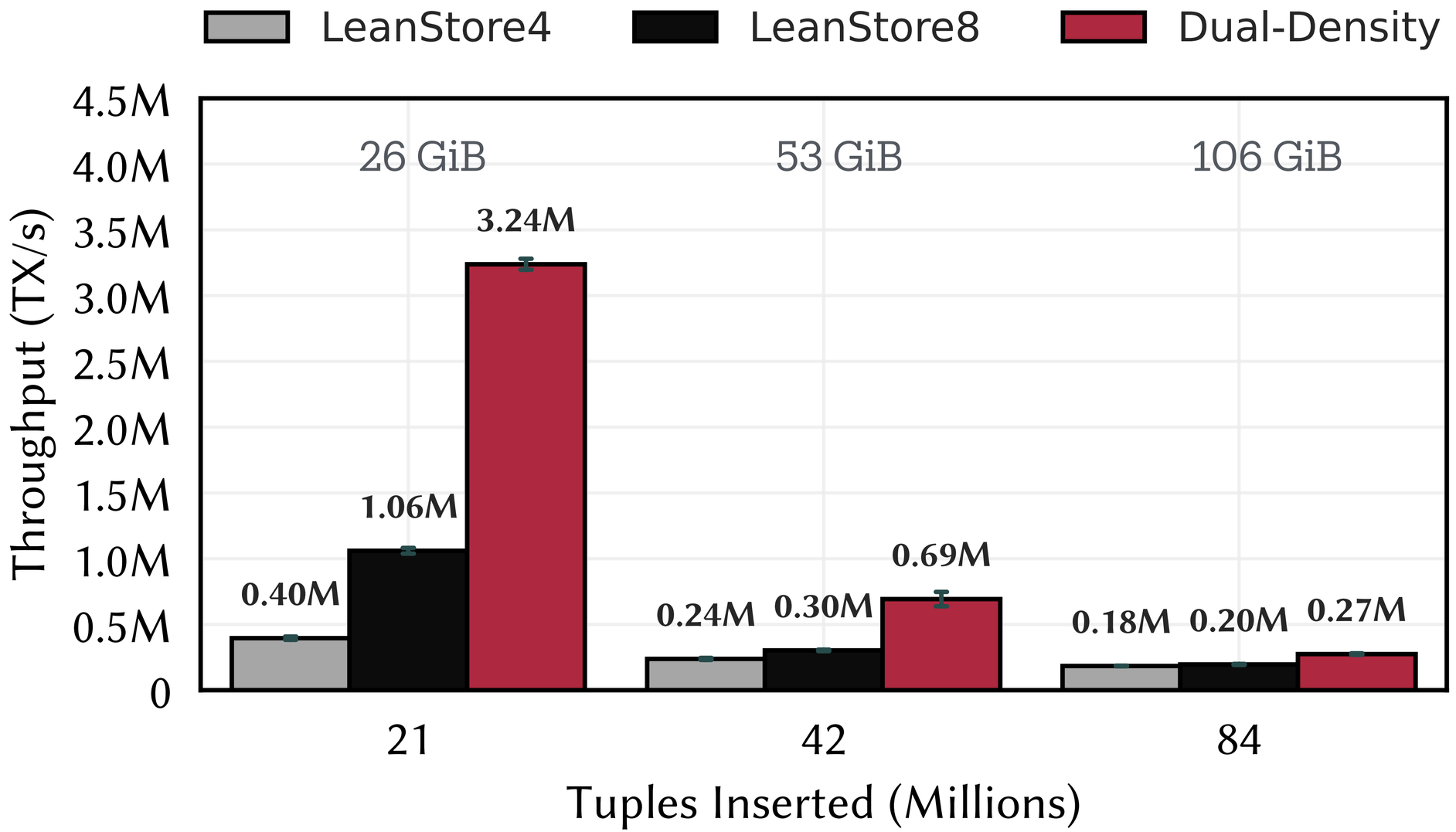
# Evaluation Setup

- AWS i4i.16xlarge - Optimized for OLTP
- Ubuntu 24.04 (kernel v6.17)
- Intel Xeon Platinum  32-core (64 Threads)
- 512GB of DRAM
- 4 locally attached NVMe SSDs (bypassing filesystem and OS page cache)
- RAID0 - distribute reads and writes to SSDs
- Buffer Pool is set to use 20GiB

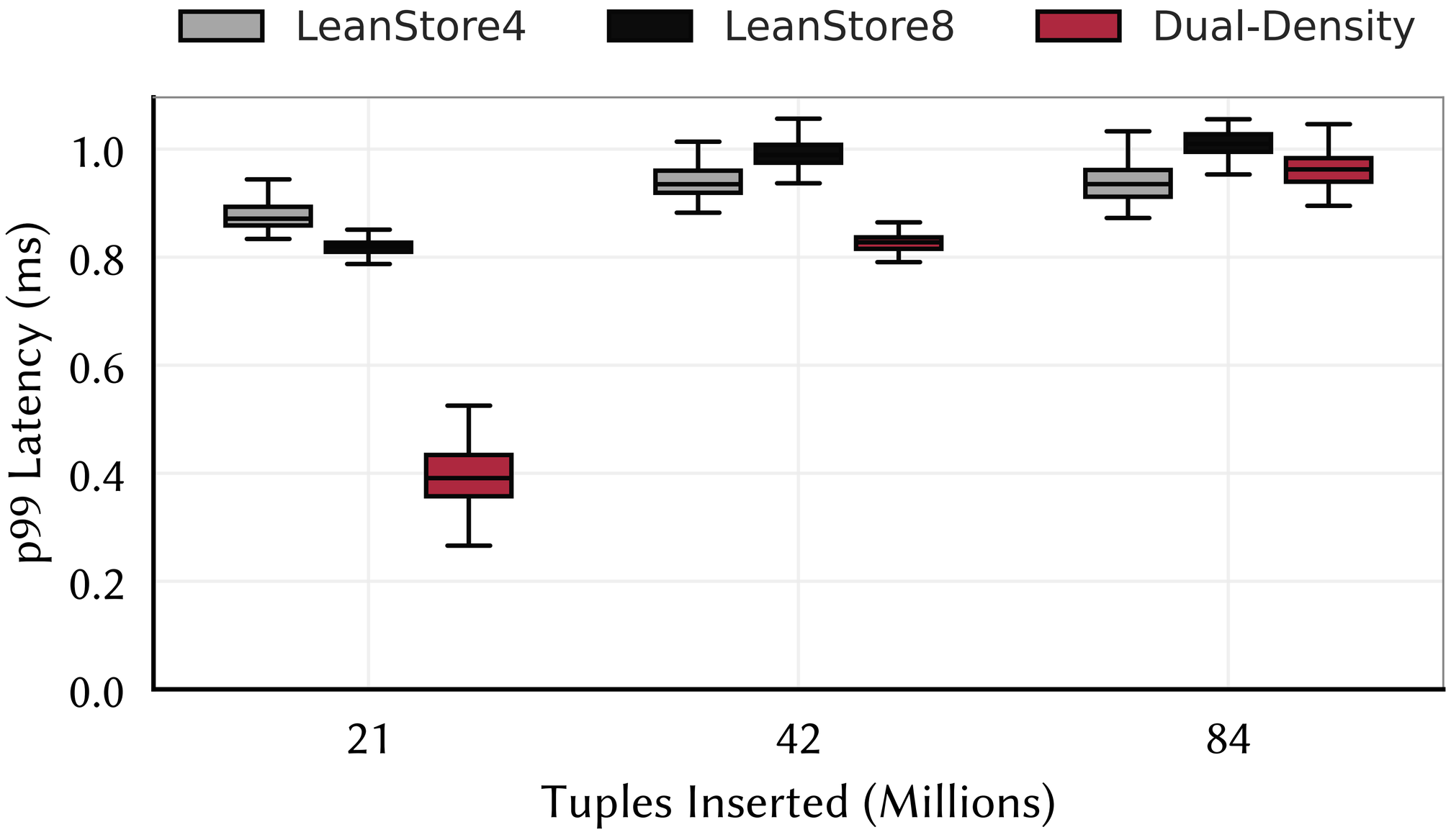
# Competitors and Workloads

- Workloads:
  - YCSB - Real World Paylod
  - TPC-C - Random Strings
- Competitors :
  - LeanStore4 - LeanStore using 4KiB pages
  - LeanStore8 - LeanStore using 8KiB pages
  - Dual-Density - LeanStore with dual-density (8KiB uncompressed - 4KiB compressed)

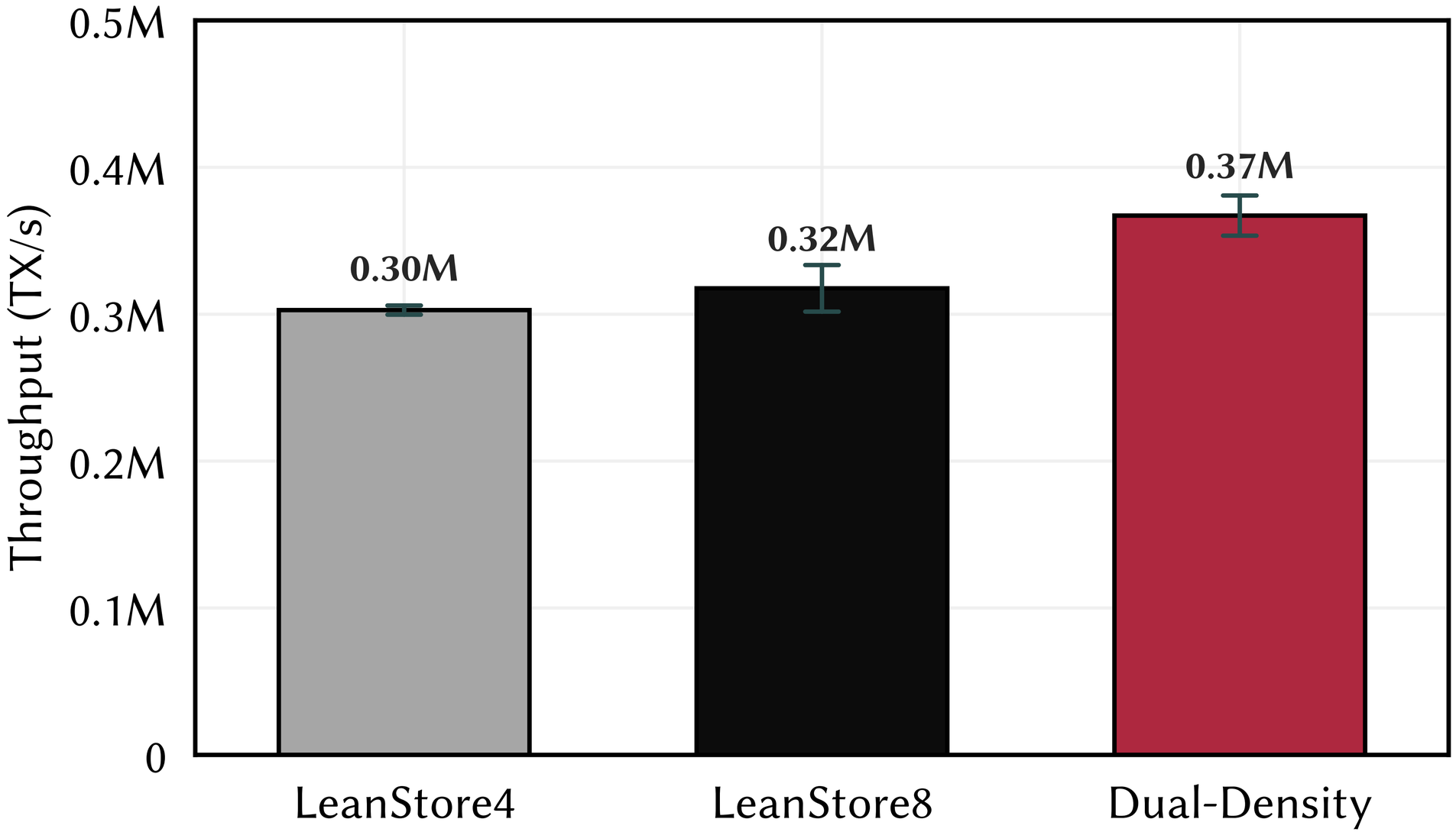
# YCSB Throughput - Scaling Database Size



# YCSB P99 Latency - Scaling Database Size



# TPC-C Throughput



# Summary

- Dual-Density - two-tier buffer pool with transparent in-memory page compression
- Trade CPU cycles for artificial memory expansion
- Efficient adaptive sizing between tiers
- Up to 3x throughput in memory-constrained OLTP workloads
- Code : https://github.com/ChrisLaspias/dual-density
- Would love to hear your questions and feedback!

# Thank You!