UNIVERSITY OF CRETE Australian National University ISOVALENT Red Hat FORTH INSTITUTE OF COMPUTER SCIENCE

TeraHeap: Reducing Memory Pressure in Managed Big Data Frameworks

Iacovos G. Kolokasis
kolokasis@ics.forth.gr

Giannos Evdorou
evdorou@ics.forth.gr

Shoaib Akram
shoaib.akram@anu.edu.au

Christos Kozanitis
kozanitis@ics.forth.gr

Anastasios Papagiannis
anastasios@isovalent.com

Foivos Zakkak
fzakkak@redhat.com

Polyvios Pratikakis
polyvios@ics.forth.gr

Angelos Bilas
bilas@ics.forth.gr

Analytics frameworks need large heaps

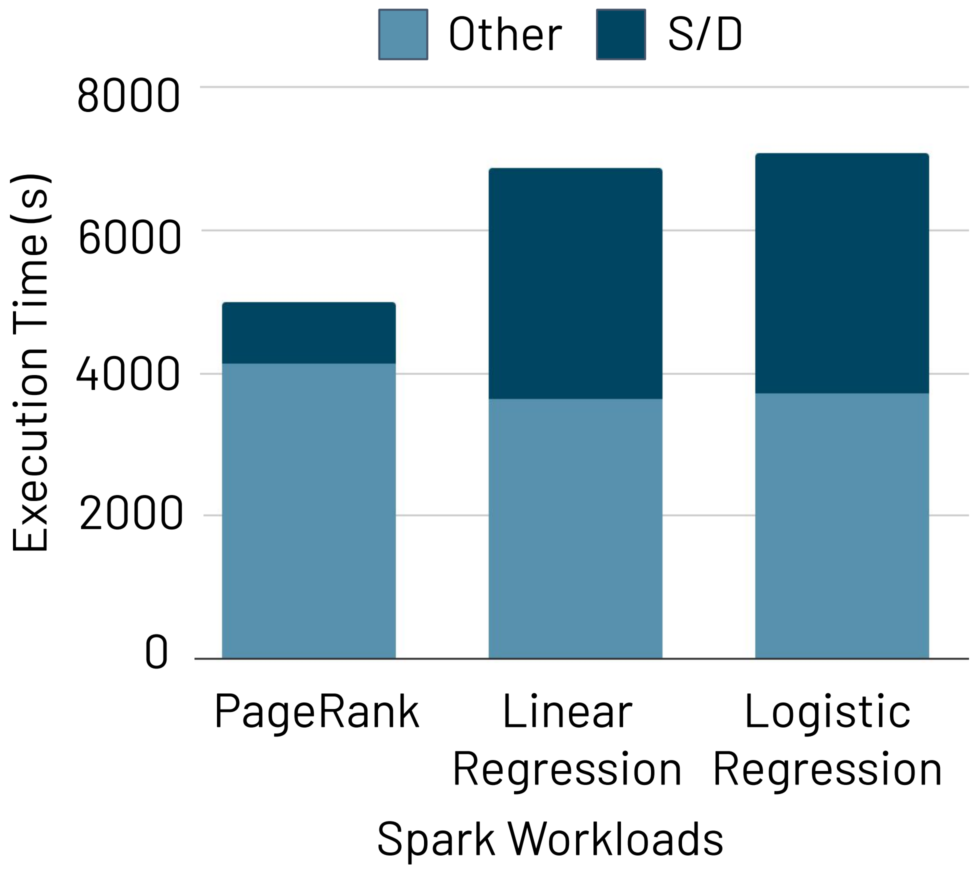
- Analytics frameworks use managed runtimes
- To process **large amounts of data** they need **large heaps**
- Large heaps are **expensive (DRAM)** and **increase GC cost!**
    - DRAM is expensive in dollar cost, energy, and power
    - GC requires expensive scans over large heaps
- For these reasons analytics frameworks avoid large heaps

Common practice: Move objects off-heap

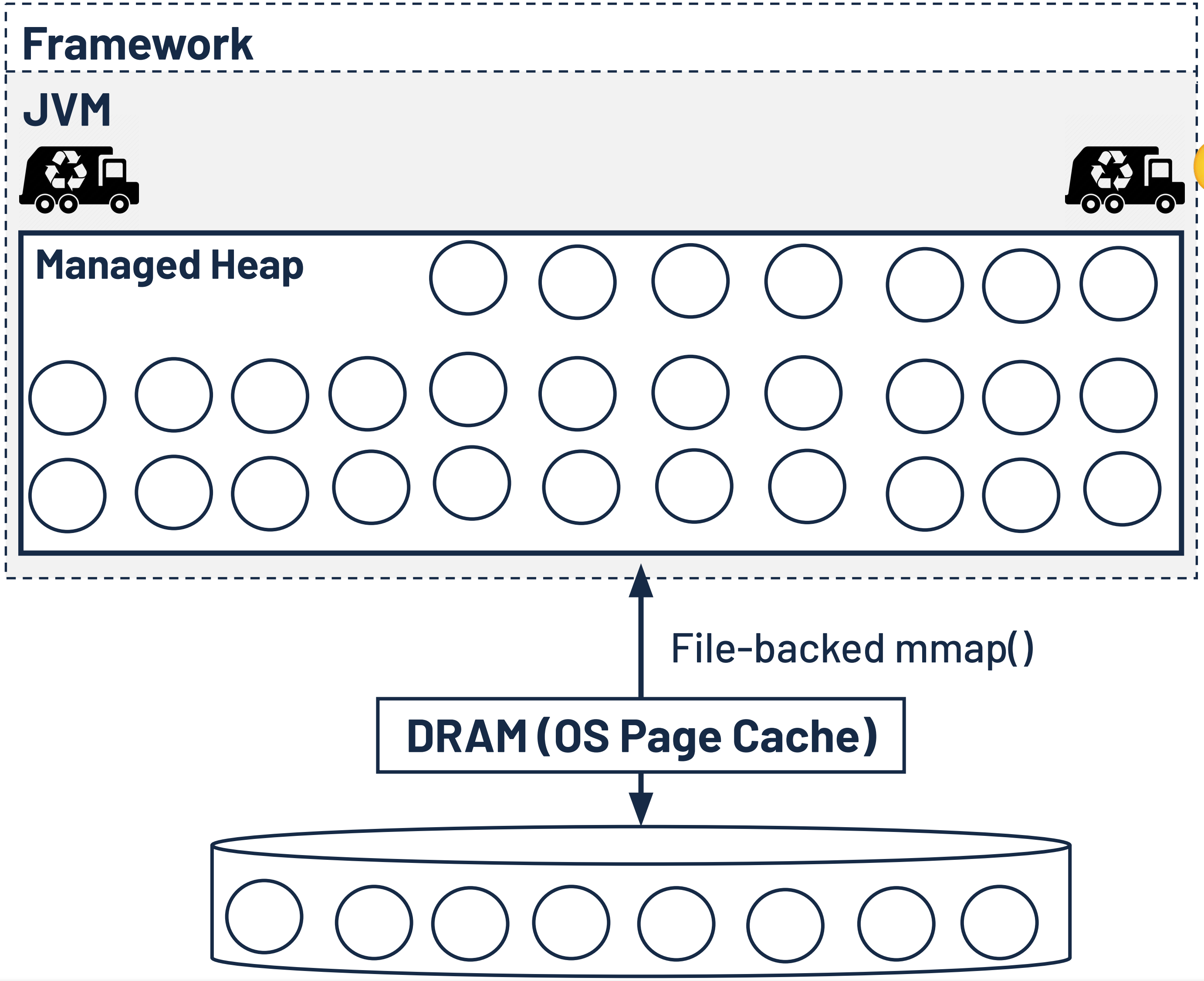
- Off-heap storage in this context means
  - Off DRAM → on fast storage
  - Unmanaged → no GC scans

- Off-heap demands serialization/deserialization (S/D)
  - Transform object closure into byte streams

- S/D is significant problem!
  - Takes up to 47% in Spark workloads
  - Not everything is serializable!
  - Off-heap can be unsafe



Eliminate S/D: Extend the heap over storage

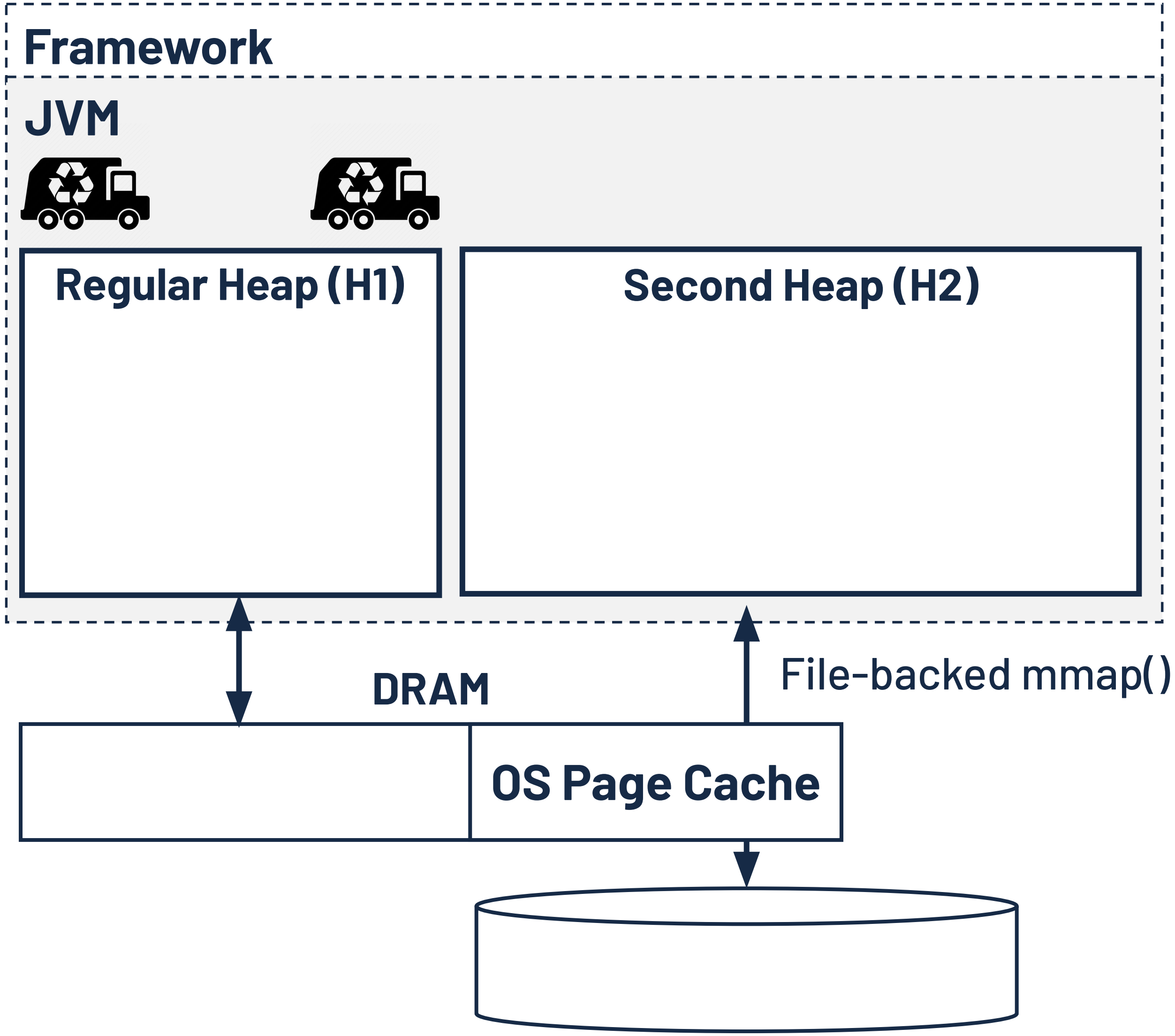


- Today OpenJDK naively uses mmap()

☹️ GC cost increases dramatically!

  - **Random accesses** over storage
  - **Object compaction** over storage
  - High I/O traffic

TeraHeap: Eliminate S/D without increasing GC cost



- Provides the illusion of a single heap
- Avoid GC scans over the device heap
- Custom management for the device heap
  - Lazy GC due to high storage capacity
  - Minimizing I/O traffic

Outline

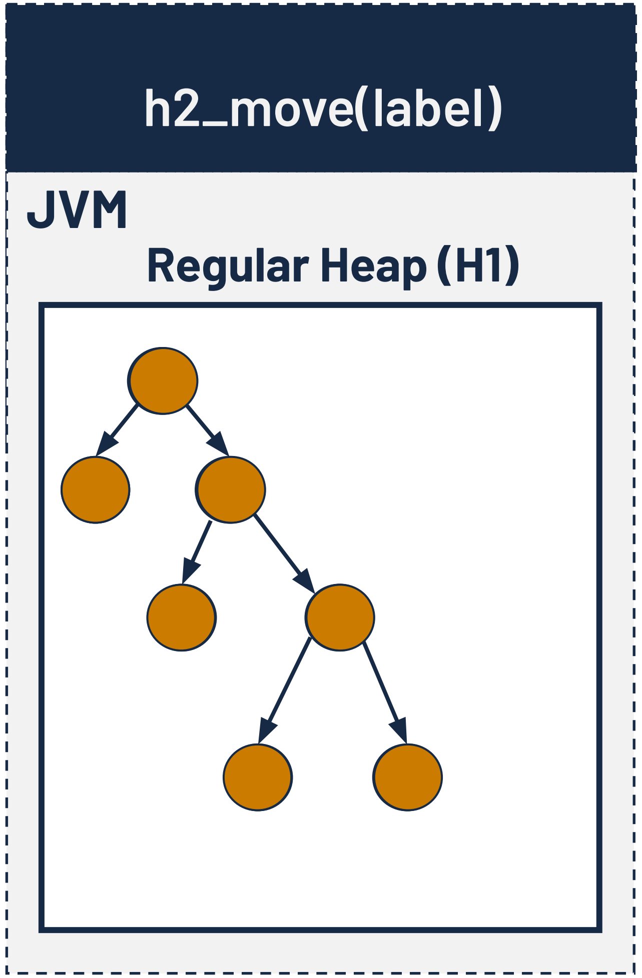
- Motivation

- Design
    - Identify objects for moving to H2
    - Reclaim objects in H2 without GC scans
    - Update cross-heap references with low I/O cost

- Evaluation

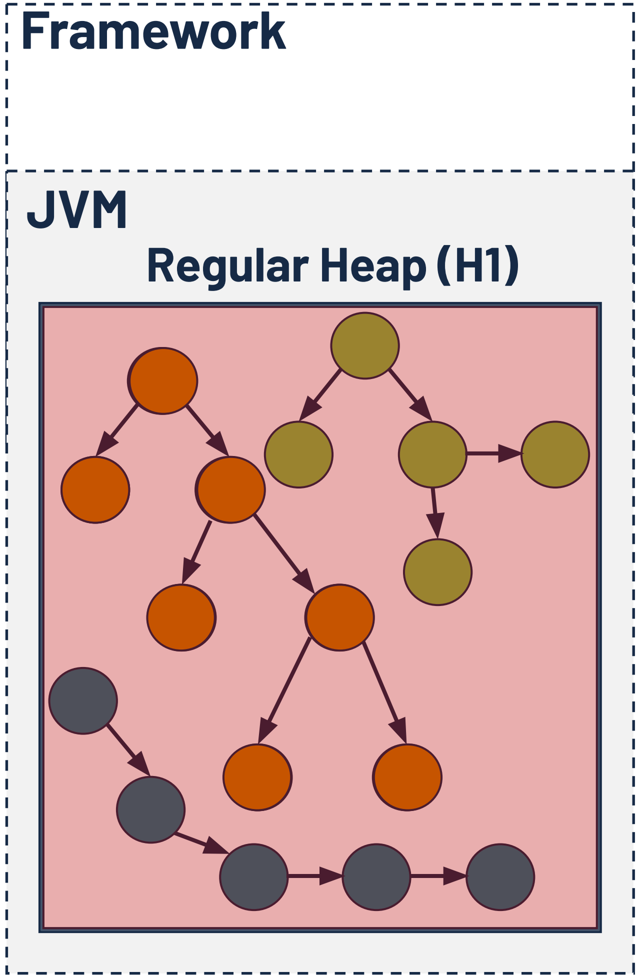
- Conclusions

Move off-heap objects to H2



- Goal: Find **large clusters** of objects with **similar lifetime**
- Frameworks **move** partitions **off-heap**
- Frameworks have **eventually immutable objects**
- TeraHeap provides two hints
  - h2_mark_root(): Mark key object with a label
  - h2_move(): Advice when to move objects to H2
- Move objects to H2 during GC
- GC propagates the label from key object to all reachable objects

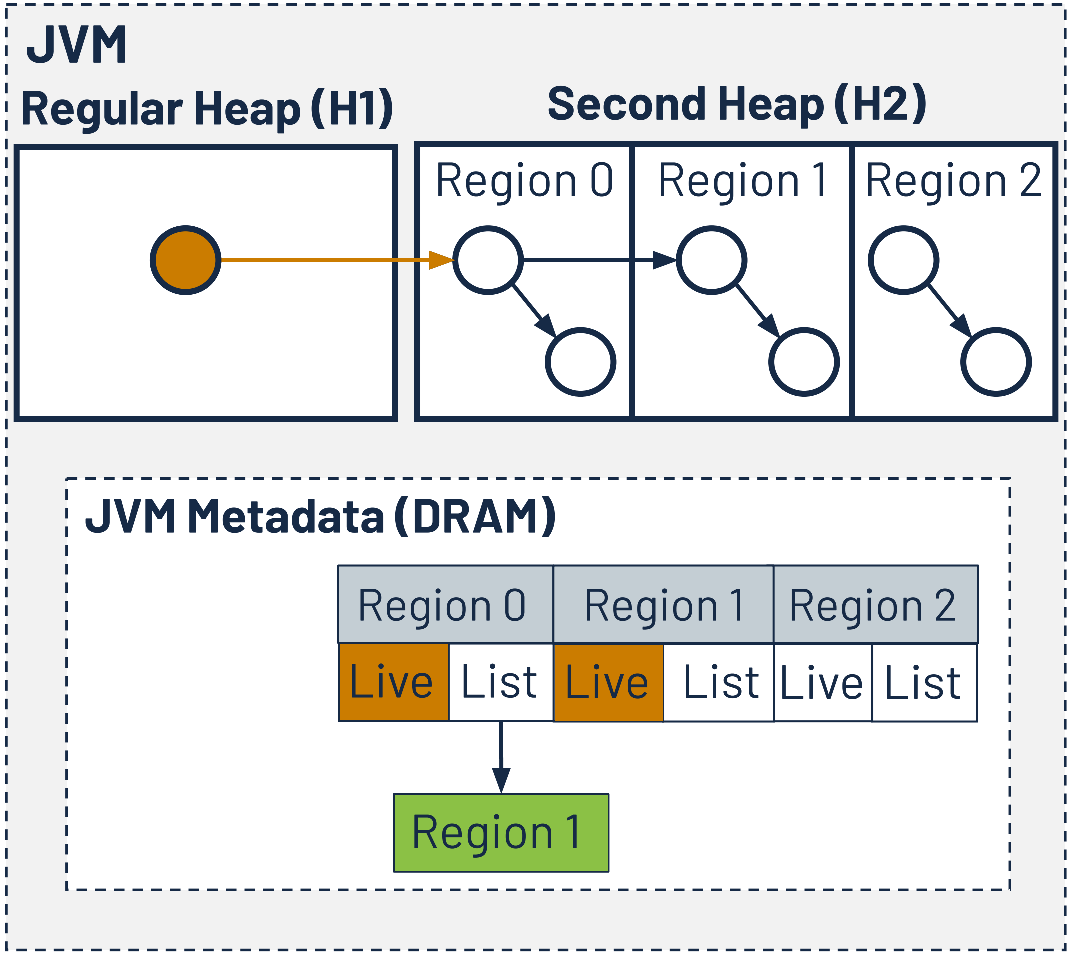
Can move objects to H2 eagerly



☐ Goal: Reduce **memory pressure in H1**

- Increased memory pressure before transfer hint?
- Eager transfers to H2 → decrease memory pressure in H1
- Use a high threshold to identify memory pressure
- Bypass transfer hint
- Move only a few marked objects to H2
  - Reduce **read-modify-write operations** in storage

Leverage storage capacity to free objects lazily



Region 0		Region 1		Region 2	
Live	List	Live	List	Live	List

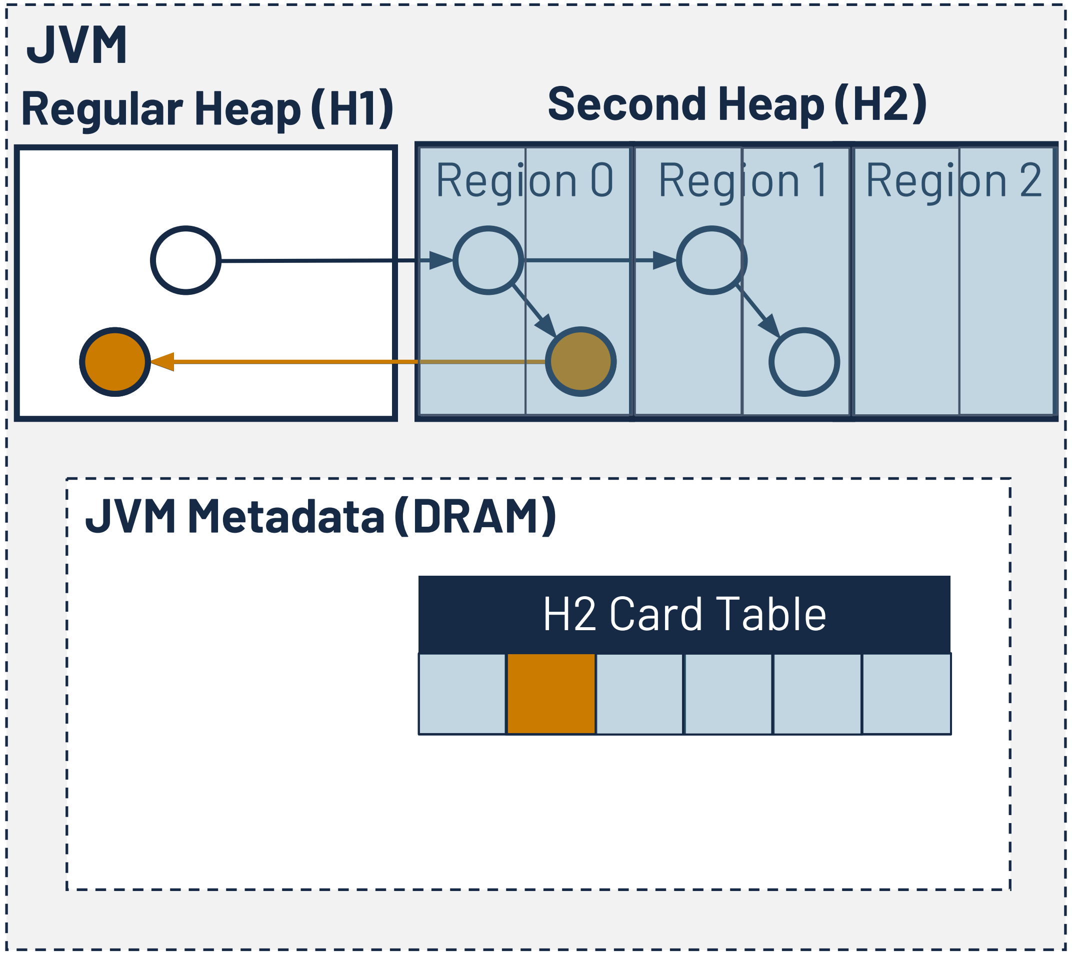
- ☐ Goal: **Reclaim** dead objects **without GC scans**

- TeraHeap organizes H2 in fixed-sized regions
  - Objects with same label in the same region
  - Reclaim whole regions **(bulk free)**

- Per region DRAM metadata **(avoid object access)**
  - Live bit → region liveness
  - Dependency list → cross-region references

- GC identifies H2 live regions
  - Free regions by zeroing regions metadata

Preserve correctness of object liveness



☐ Goal: **Track** H2 to H1 references with **low I/O cost**

- Card table (byte array in DRAM)
  - One byte per fixed-size H2 segments
  - Large segments to reduce card table size
- Categorize cards to scan less segments
- Based on GC type, we scan specific segments

Testbed

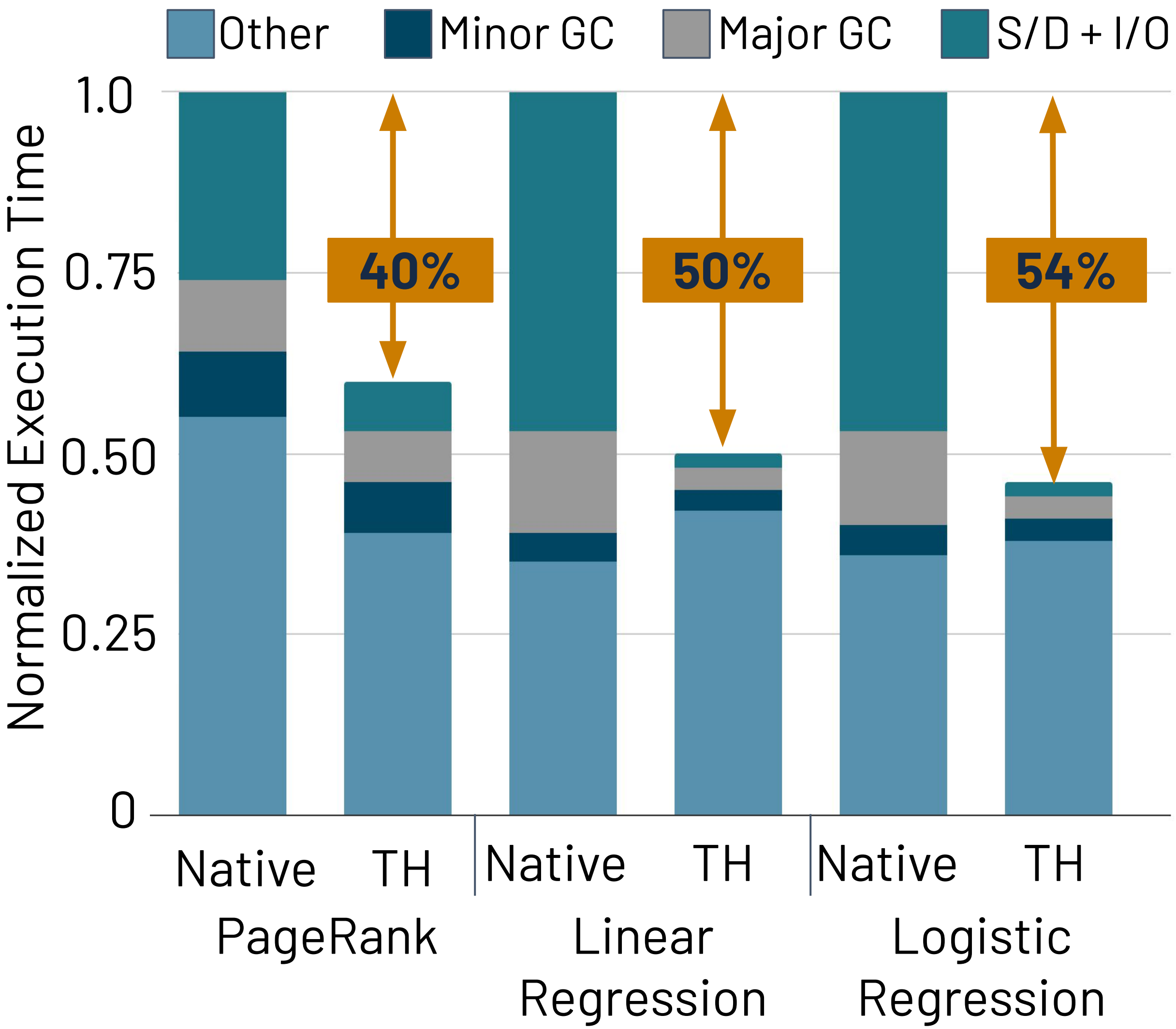
- We implement TeraHeap in OpenJDK 8 (we now support OpenJDK 17)
    - Extend Parallel Scavenge garbage collector
    - Extend interpreter, C1 and C2 (JIT) compilers to support updates in H2

- We use one servers with 2 TB NVMe SSD and 256 GB DRAM
    - Also, we evaluate TeraHeap with NVM

- Real world applications
    - Spark with SparkBench suite
    - Giraph with Graphalytics benchmark suite

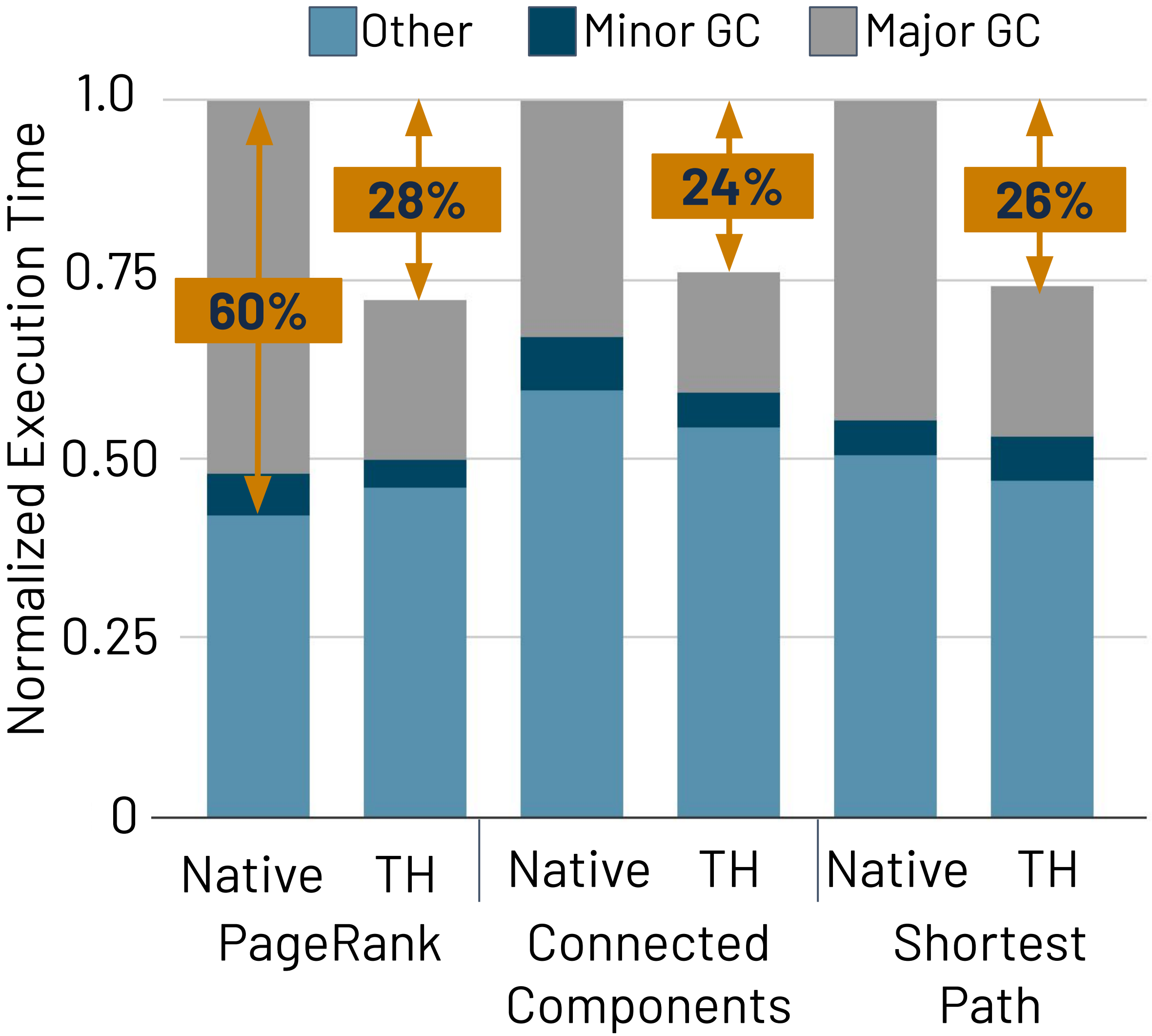
- Limit DRAM capacity using cgroups

TeraHeap outperforms native Spark by up to 54%



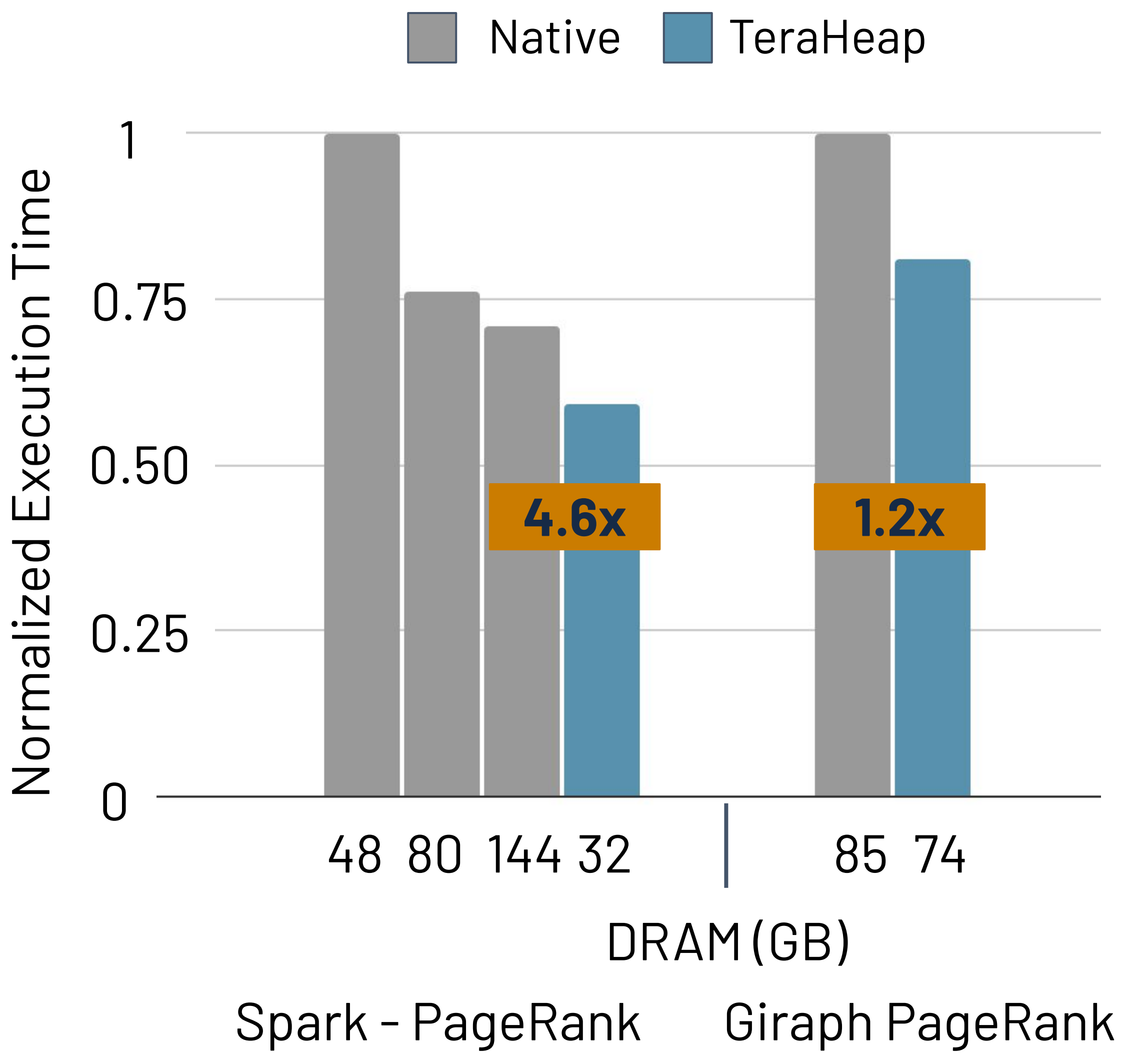
- Teraheap reduces S/D overhead
- S/D in TeraHeap is due to shuffling

TeraHeap outperforms native Giraph by up to 28%



- Main performance improvement
  - Reduction of major GC (up to 50%)
- Off-heap **reduces** heap pressure **temporarily**
  - Giraph processes objects **only on-heap**
  - Increases heap pressure → Increased GC!

TeraHeap reduces DRAM requirements



- Provide direct access to H2 objects
- Outperforms native Spark using 4.6x less DRAM
- Outperforms native Giraph using 1.2x less DRAM

Key Takeaways

- Analytics frameworks deal with large datasets using S/D

- TeraHeap provides the illusion of single managed heap
    - No S/D and no GC scans in the device heap for freeing space

- Improves native Spark and Giraph performance by up to 54% and 28%

- TeraHeap requires up to 4.6x less DRAM

Future work

- Eliminate hints by dynamically determining which objects to move to H2

TeraHeap: Reducing Memory Pressure for Managed Big Data Frameworks

Artifacts Available acm V1.1

Artifacts Evaluated acm Functional V1.1

github.com/CARV-ICS-FORTH/teraheap

We thankfully acknowledge the support of the European Commision projects
EVOLVE (GA No 825061) and Eupex (GA No 101033975)
Iacovos G. Kolokasis is supported by the Meta Research PhD Fellowship (2022 – 2024)