SmartSweep: Efficient Space Reclamation in Tiered Managed Heaps

Iacovos G. Kolokasis
kolokasis@ics.forth.gr

Konstantinos Delis
konstdelis@ics.forth.gr

Shoaib Akram
shoaib.akram@anu.edu.au

Foivos S. Zakkak
fzakkak@redhat.com

Polyvios Pratikakis
polyvios@ics.forth.gr

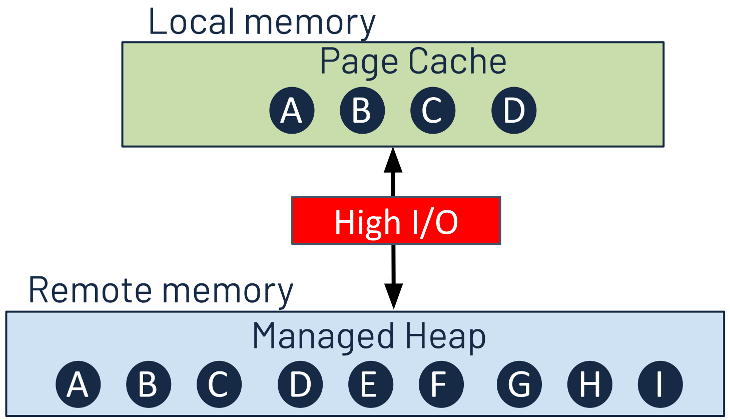
Angelos Bilas
bilas@ics.forth.gr

Analytics frameworks need large heaps

- Popular big data frameworks running on managed runtimes
- To process **large amount of data** they need **large heaps**
- However, scaling DRAM in a single server is **costly and impractical**
  - DRAM is expensive in dollar cost, energy, and power
- Remote memory offers a **scalable, cost-efficient way** to extend the Java heap
  - Use idle memory across remote servers

Garbage collection over remote memory is expensive

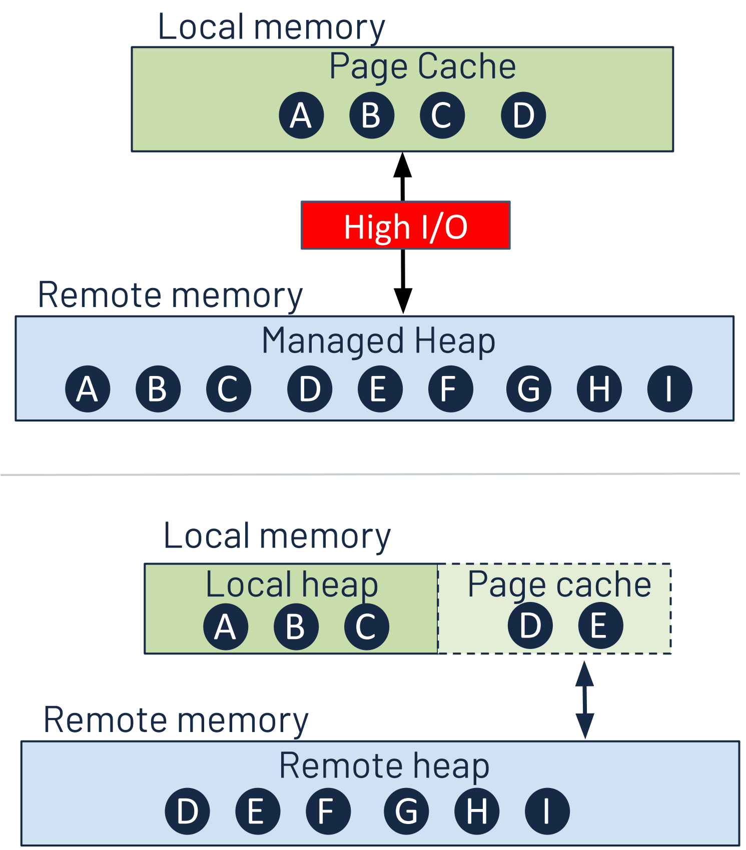
- Large remote heaps make GC operations slower and costly
  - Remote scans and compactions amplifies GC overhead
  - Significant network traffic [MemLiner OSDI'22]



Garbage collection over remote memory is expensive

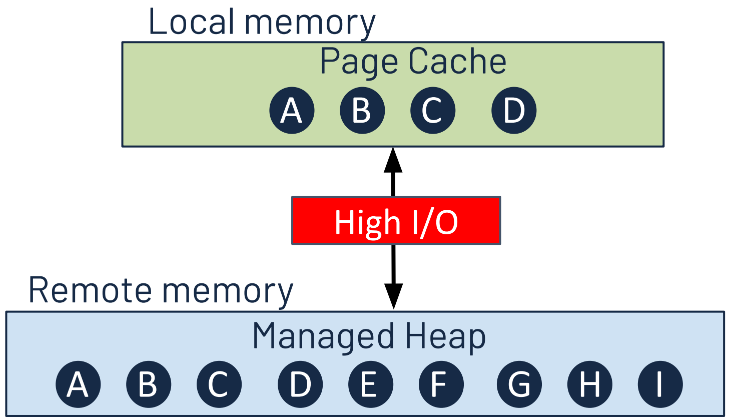
- Large remote heaps make GC operations slower and costly
  - Remote scans and compactions amplifies GC overhead
  - Significant network traffic [MemLiner OSDI'22]

- Our approach: Divide heap to **local** (H1) and **remote** (H2)
  - "Dual-heap" architecture
  - Limit GC operations in **local memory only**
  - **No full scans** and **compactions** over **remote heap**
  - Up to **177x less network I/O traffic** than single-heap architectures

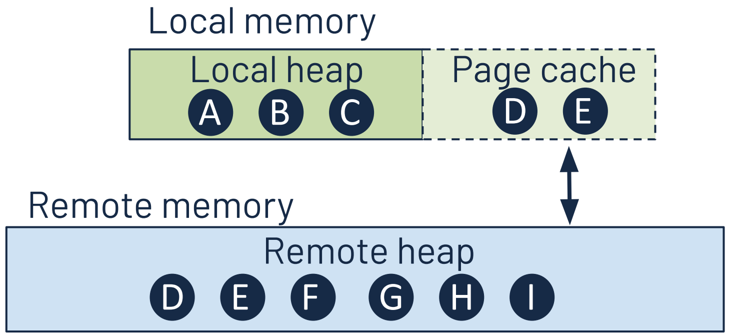


Garbage collection over remote memory is expensive

- Large remote heaps make GC operations slower and costly
  - Remote scans and compactions amplifies GC overhead
  - Significant network traffic [MemLiner OSDI'22]

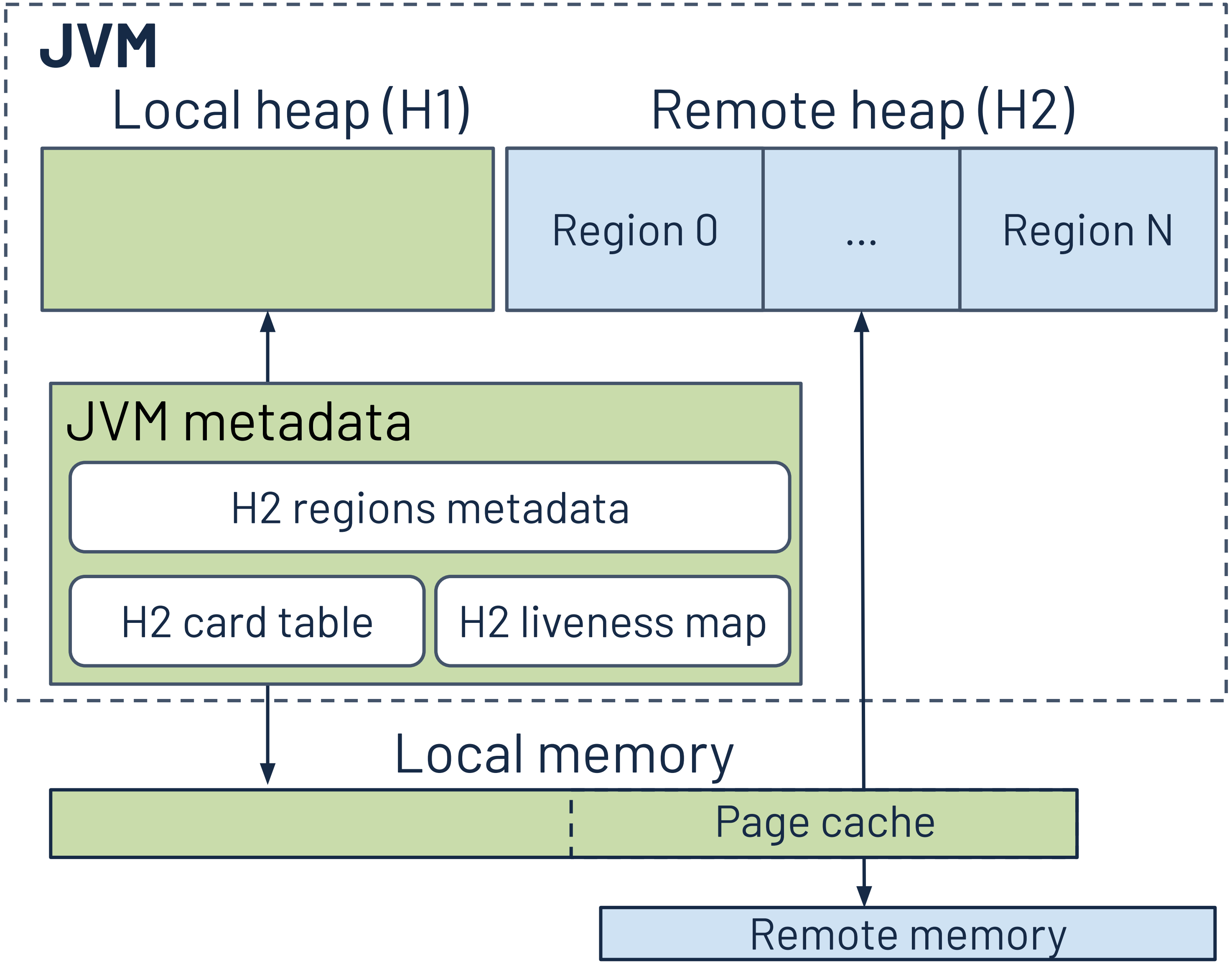


- Our approach: Divide heap to **local** (H1) and **remote** (H2)
  - "Dual-heap" architecture
  - Limit GC operations in **local memory only**
  - **No full scans** and **compactions** over **remote heap**
  - Up to **177x less network I/O traffic** than single-heap architectures



- **Challenge:Reclaim space in remote heap promptly**
  - Otherwise, wasted memory, OOM errors

SmartSweep: Space reclamation without remote GC scans



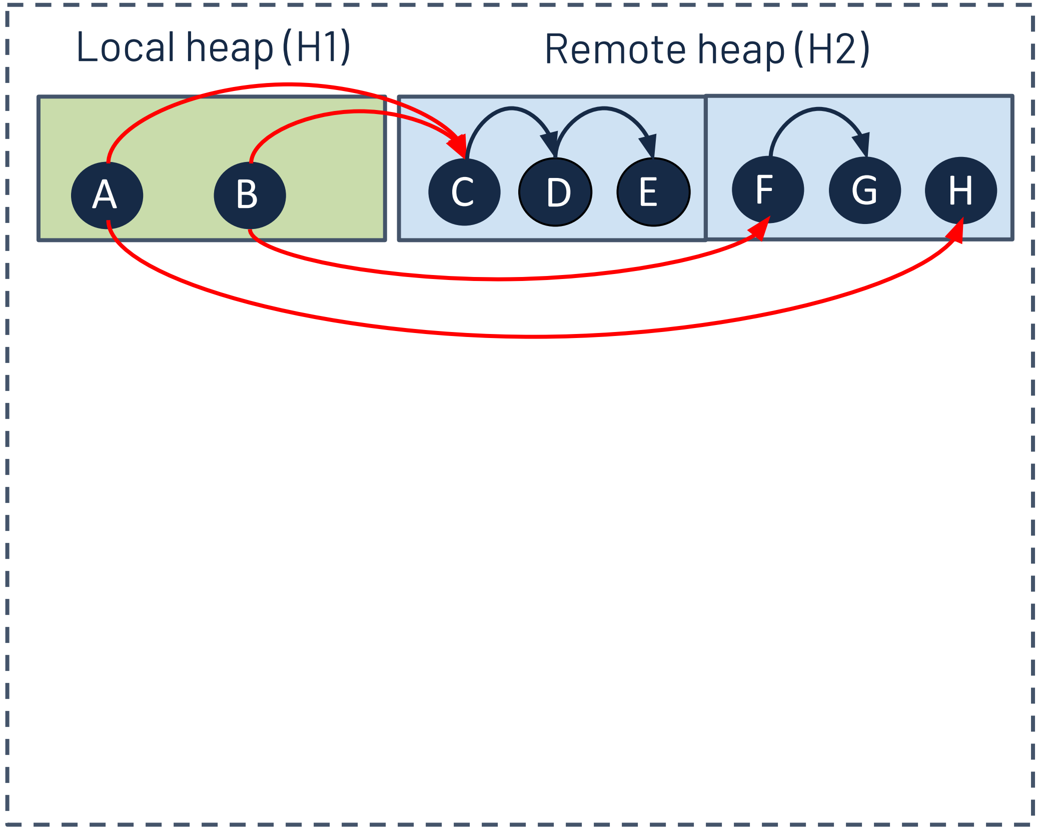
- Remote heap is region-based
  - Treat all objects in a region as a single unit
- No remote GC scans
  - Estimate live H2 objects
  - **Metadata** for each H2 region (local memory)
- Reclaim H2 regions with garbage
  - **Transfer** H2 regions to H1
  - GC reclaims dead H2 objects (local memory)

Outline

- Motivation
- Preliminary design
  - Finding dead objects without scanning the remote heap
  - Reclaiming dead objects in the remote heap
  - Maintaining object references in the remote heap
- Preliminary evaluation
- Conclusions & Future work

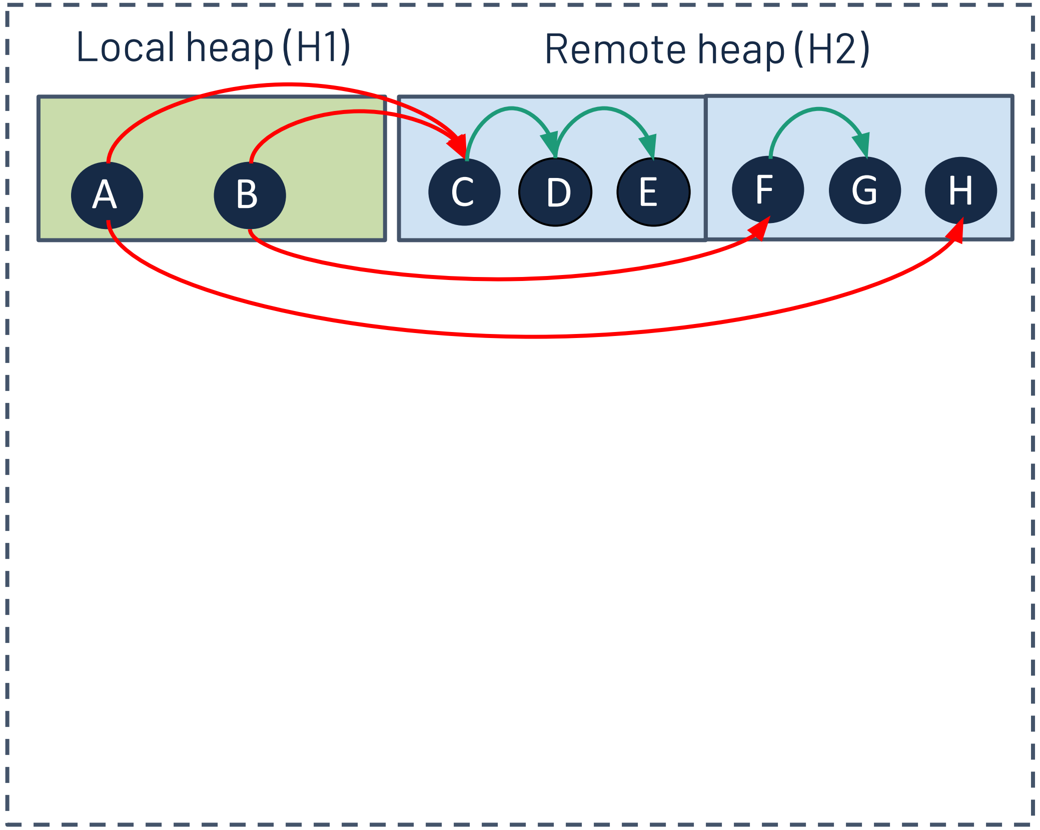
Finding dead objects without scanning the remote heap

- Estimating live objects for each H2 region
  - Track forward references (H1 → H2)



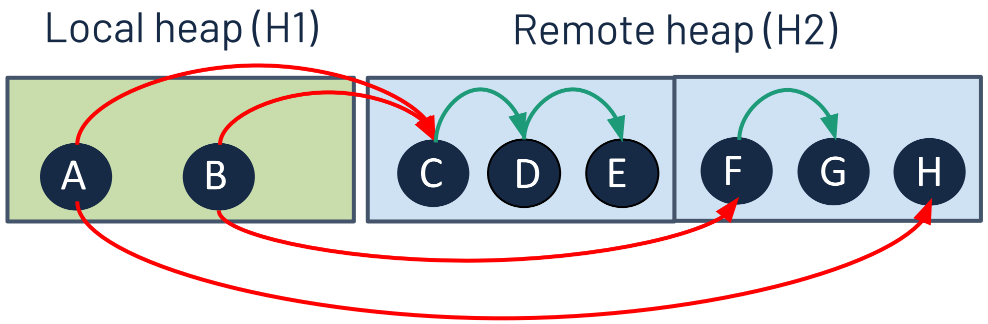
Finding dead objects without scanning the remote heap

- Estimating live objects for each H2 region
  - Track forward references (H1 → H2)
  - Detect changes in references inside regions



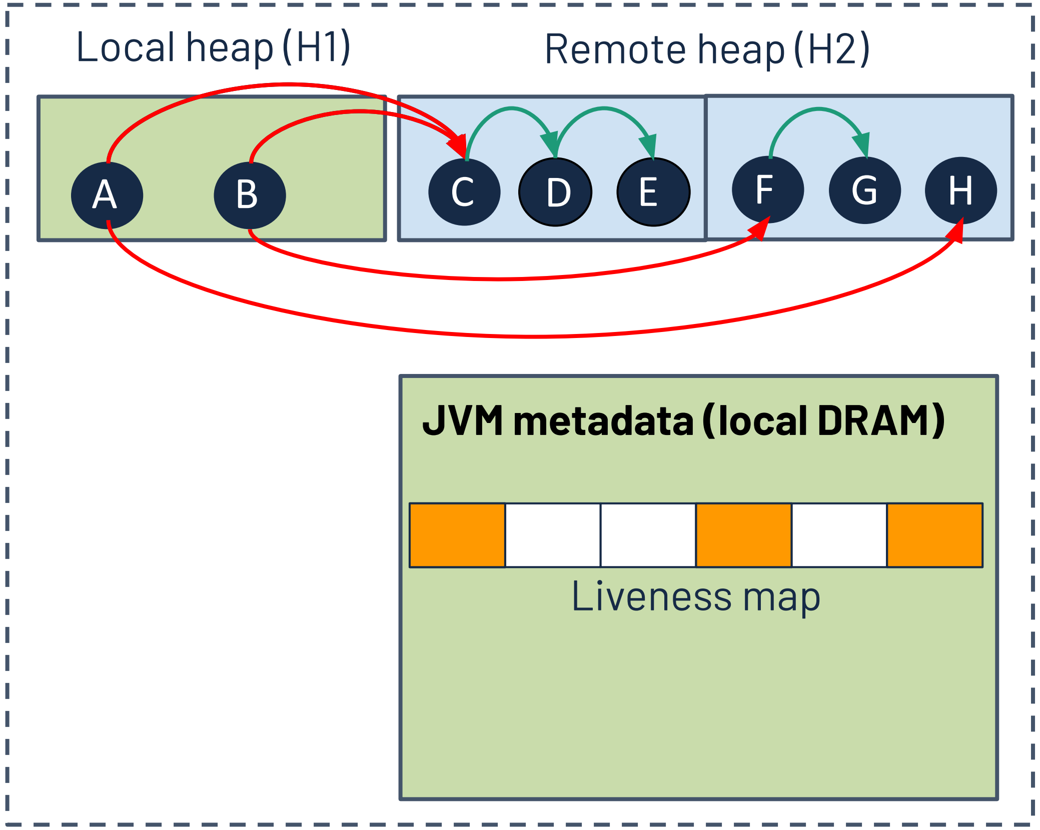
Finding dead objects without scanning the remote heap

- Estimating live objects for each H2 region
  - Track forward references (H1 → H2)
  - Detect changes in references inside regions
- For forward references we use spatial information
  - Simple reference counting → misleading results



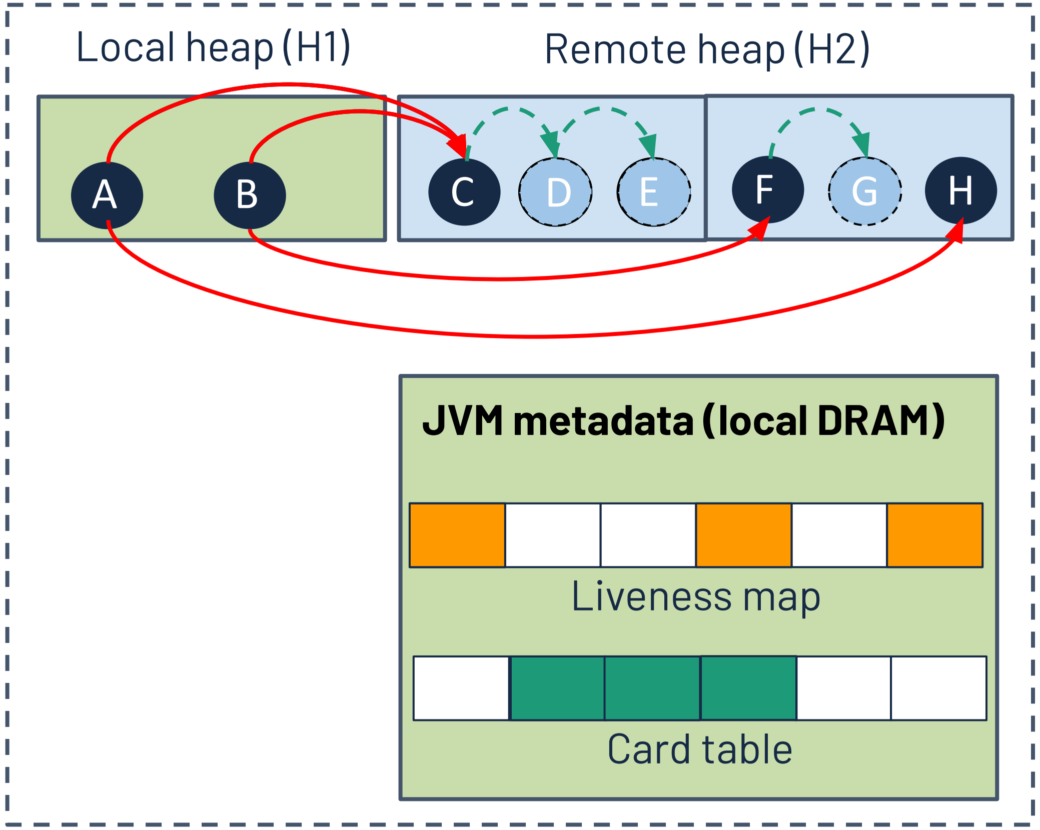
Finding dead objects without scanning the remote heap

- Estimating live objects for each H2 region
  - Track forward references (H1 → H2)
  - Detect changes in references inside regions
- For forward references we use spatial information
  - Simple reference counting → misleading results
  - Use a liveness map (1 byte per 4 KB H2 segment)
  - Dirty the byte that correspond to the fwd ref.



Finding dead objects without scanning the remote heap

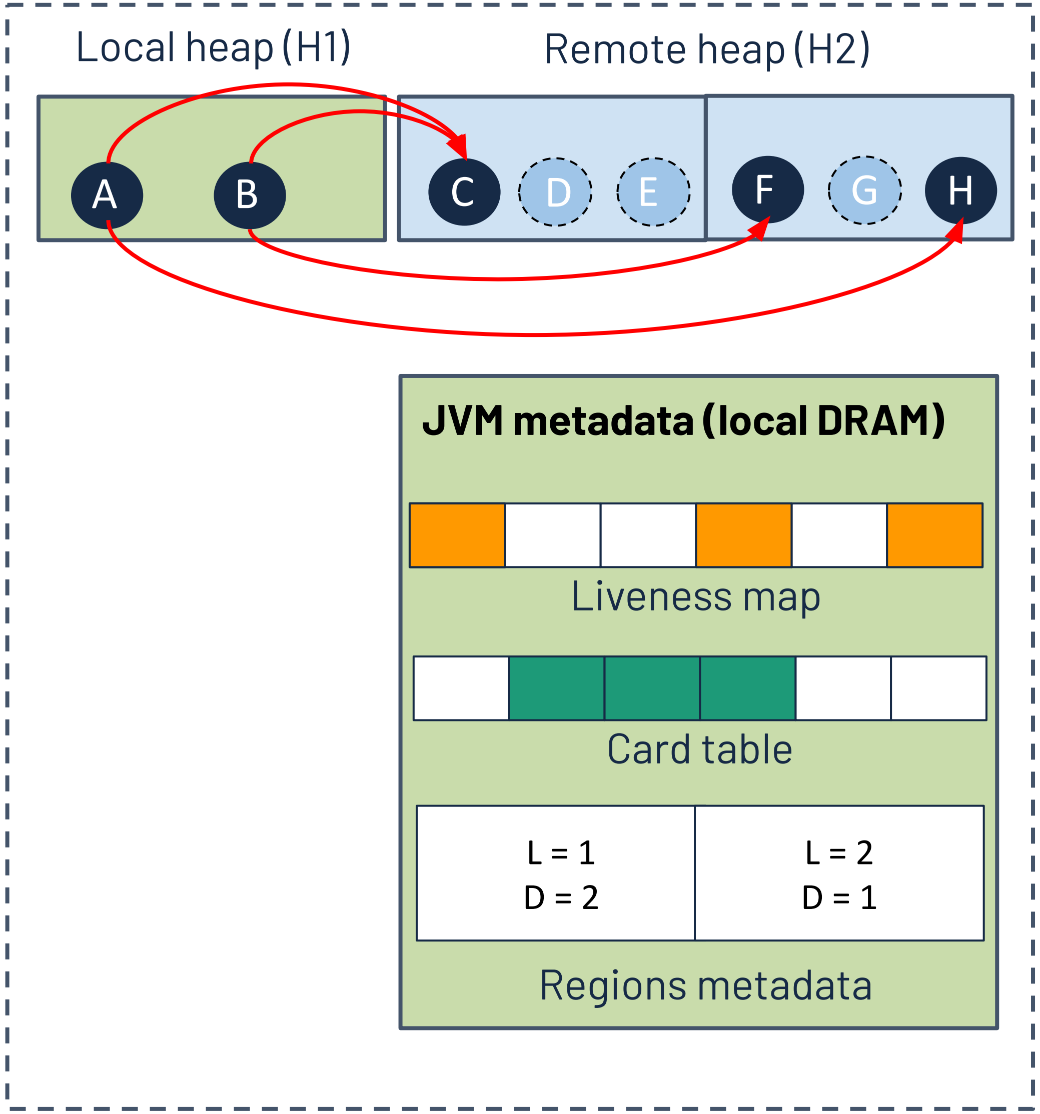
- Estimating live objects for each H2 region
  - Track forward references (H1 → H2)
  - Detect changes in references inside regions
- For forward references we use spatial information
  - Simple reference counting → misleading results
  - Use a liveness map (1 byte per 4 KB H2 segment)
  - Dirty the byte that correspond to the fwd ref.
- For inter-region references we track the updates
  - Use a card table (one byte per 4 KB H2 segment)
  - Record mutator threads updates
  - Count the number of dirty cards → reveal reference changes



Reclaim dead objects in the remote heap without compactions

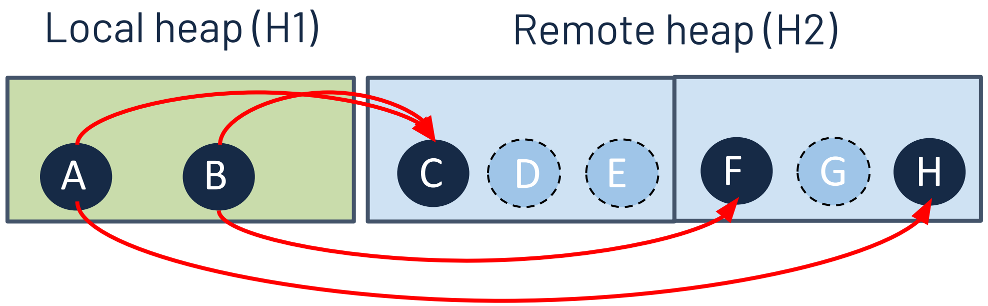
- Score each H2 region using
  - L: # dirty bytes in liveness map / # region obj
  - D: # dirty cards per region

- We define a threshold (U):
  - Score(Ri) < U: Ri is queued for transfer

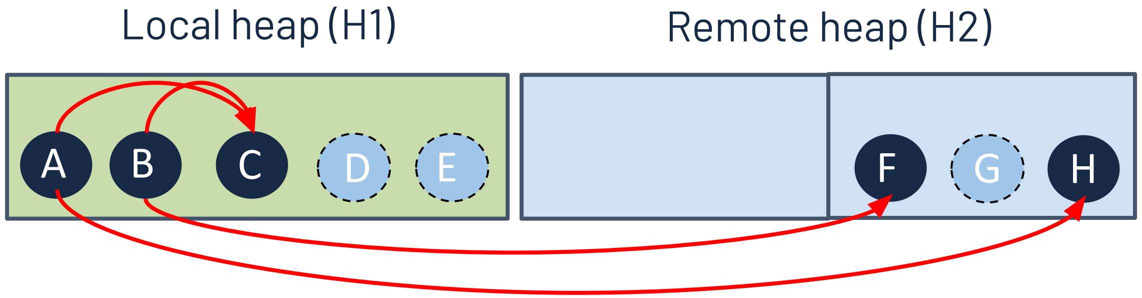


Reclaim dead objects in the remote heap without compactions

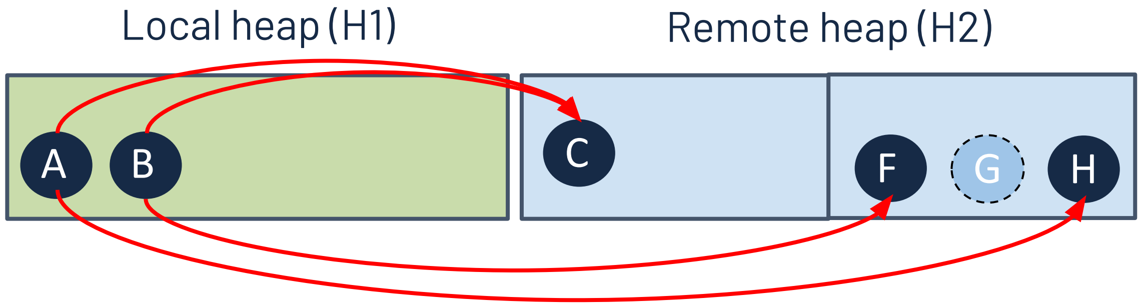
- Score each H2 region using
  - L: # dirty bytes in liveness map / # region obj
  - D: # dirty cards per region



- We define a threshold (U):
  - Score(Ri) < U: Ri is queued for transfer



- To avoid in-place compactions
  - Transfer regions from H2 to H1
  - Next GC cycle: GC reclaims the dead objects
  - GC transfers live H2 objects back to H2



- For H2 regions full of garbage we just zero their metadata

Maintaining cross-region and cross-heap references

- Naïve fix: move the region's entire transitive closure
  - Too expensive when dependencies are widespread

- SmartSweep's placement strategy
  - Put the transitive closure of an object into a separate region
  - ≈ 70 % of H2 regions are referenced by less than 2 other regions
  - Limited connectivity → **cheaper to patch in place**

- **Future work:** Maintain a cross-region remember sets for each region
  - Updated during GC and via JIT post-write barriers

Outline

- Motivation
- Preliminary design
  - Finding dead objects without scanning H2
  - Reclaim space in H2
  - Maintaining cross-region references
- Preliminary evaluation
- Conclusions & Future work

Preliminary implementation

- We implement SmartSweep on top of TeraHeap (ASPLOS '23) in OpenJDK 17
  - TeraHeap is the state-of-the-art dual heap architecture
    - Organize objects into regions and reclaim regions when all objects are dead
    - Suitable for large and low-cost capacity storage devices
    - **Impractical for limited remote memory**, resulting in **OOM errors**

- Our prototype moves to H2 only primitive and leaf objects
  - Only forward references (H1 $\rightarrow$ H2)
  - Primitive and leaf objects occupy > 70% of the Java heap in Spark and Neo4j-GDS
  - Confirms that our prototype captures the dominant memory behavior

- SmartSweep accesses remote memory via NVMe-over-fabric (NVMe-oF) via MMIO

Preliminary evaluation

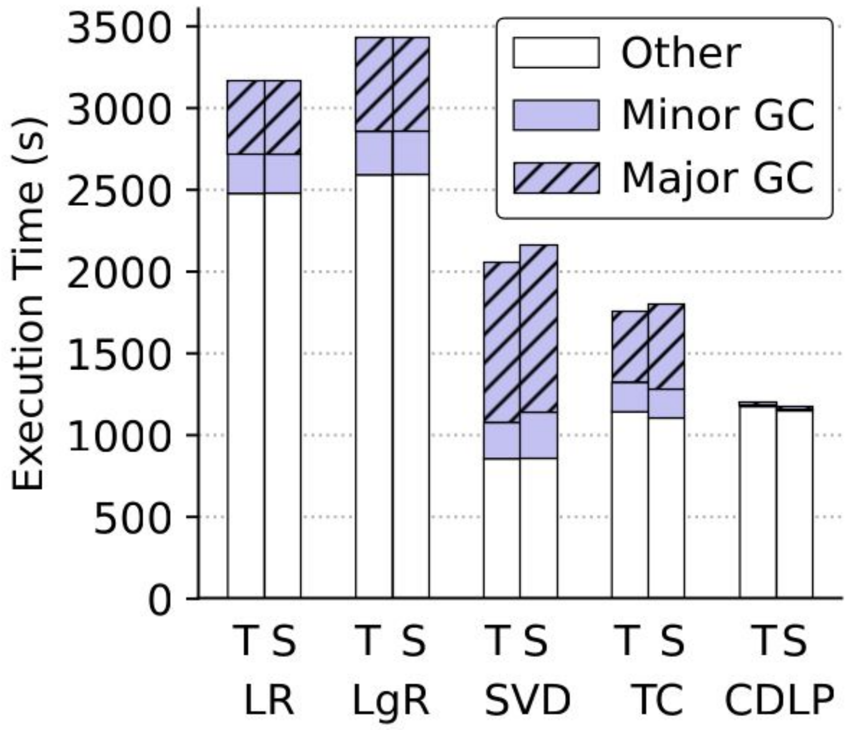
- Experimental platform:
  - 4 dual-socket servers with Intel Xeon E5-2630 v3 CPUs (2.4 GHz, 8 cores / 16 threads each → 32 threads total per node)
  - 256 GB DDR4 DRAM per server
  - Ubuntu 24.02 with Linux kernel 5.14

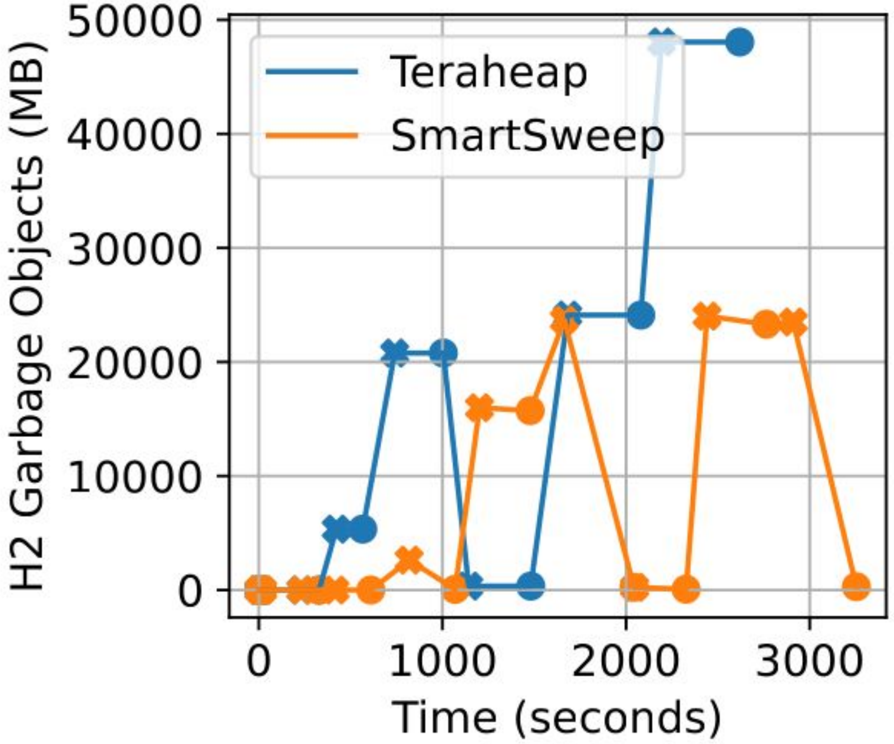
- Configuration:
  - 1 server runs the application; 3 servers act as remote memory (NVMe-oF)
  - Spark 3.3.0 with 1 executor and 8 mutator threads
  - Neo4j-GDS with 4 mutator threads (community edition limit)
  - 8 GC threads in all configurations

SmartSweep reduces wasted space without hurting performance

- SmartSweep achieves comparable performance with TeraHeap

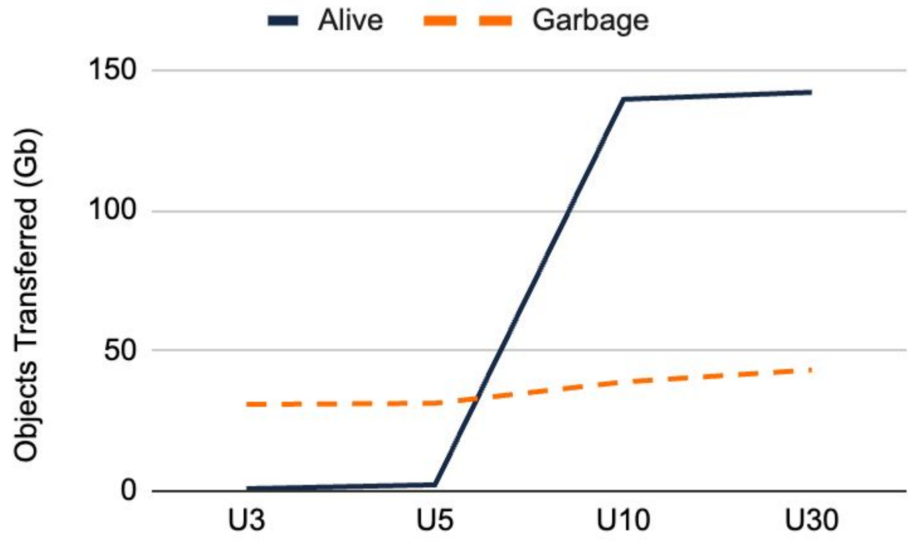
- SmartSweep reduces space waste by 50%

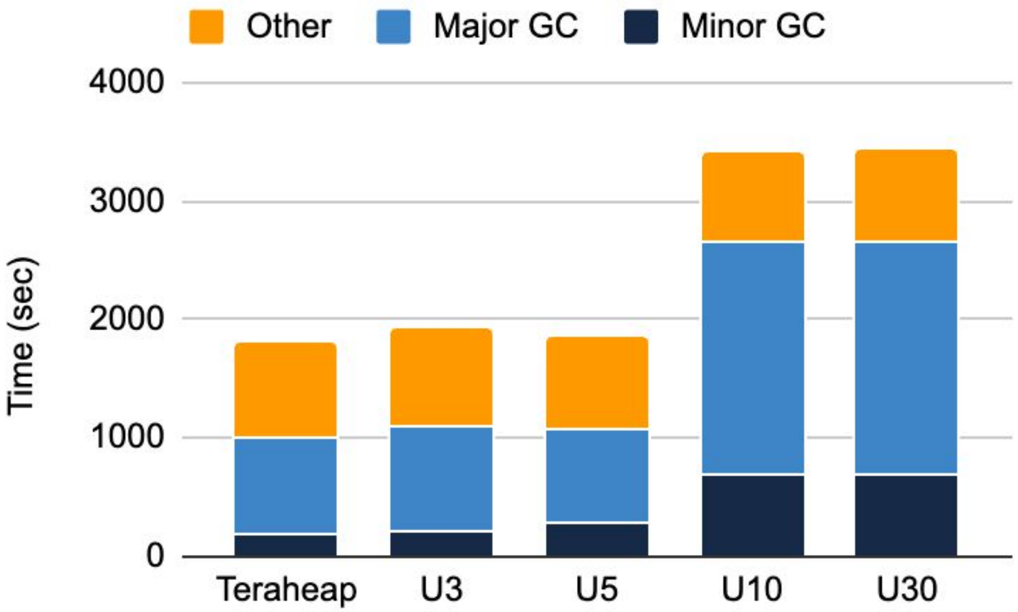




Transfer threshold affects GC overhead

- Low U →less reclamation, lower cost
- High U → aggressive reclamation, higher GC cost
  - Transfer to H1 more live H2 objects
  - Increase memory pressure in H1
- The threshold value directly affects GC overhead
- So, we need a dynamic threshold selection





Conclusions & Future work

- Dual-heap architecture can deal with large GC cost in remote managed heaps

- But, they need to reclaim space in remote memory promptly

- We propose SmartSweep
  - Estimate the amount of garbage objects per region
  - Move regions with large amount of garbage from H2 to H1

- SmartSweep cuts wasted space by half and stays just as fast as TeraHeap

- **For future work**
  - Support for dynamic transfer threshold & and reclaim objects with references in H2
  - Evaluation with CXL and compressed-DRAM

Thank you!
Questions?

Iacovos G. Kolokasis
kolokasis@ics.forth.gr
jackkolokasis.com

We thankfully acknowledge the support of the European Commision projects AERO (GA No 10048318), EUPEX (GA No 101033975) through the European High-Performance Computing Joint Undertaking (JU), and VMware's University Research Fund.