Efficient Use-after-Free Prevention with Opportunistic Page-Level Sweeping

Chanyoung Park, Hyungon Moon

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

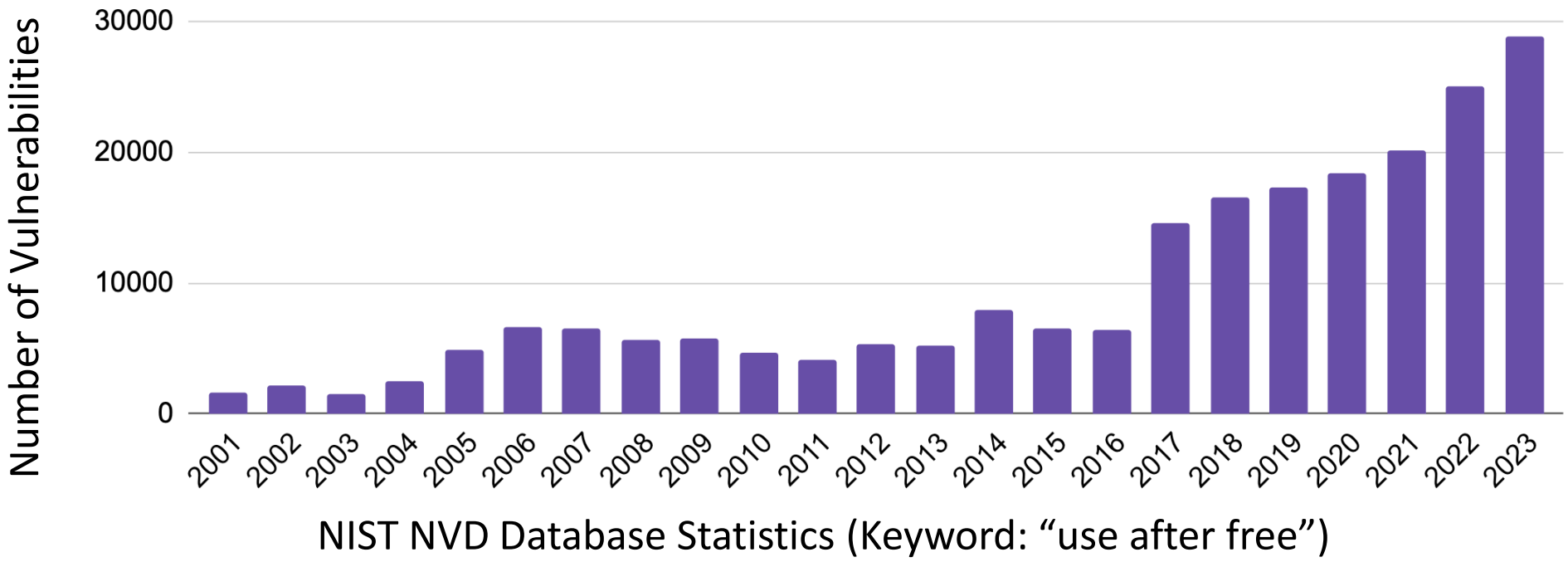
NDSS SYMPOSIUM/2024

Presented by Internet Society

#NDSSSymposium2024

Use-after-Free is still Prevalent

- An increasing number of use-after-free vulnerabilities are reported every year.

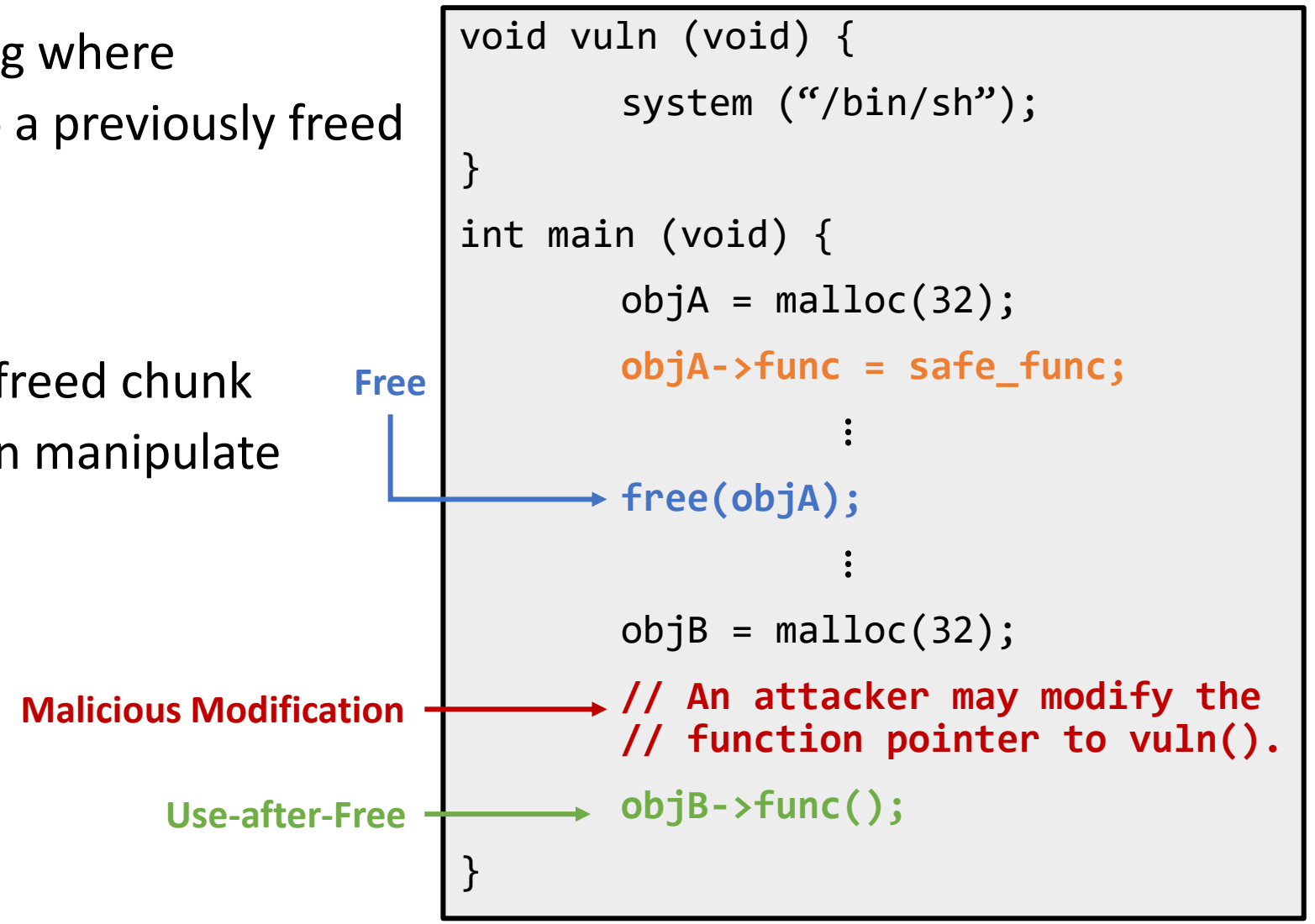


NIST NVD Database Statistics (Keyword: “use after free”)

Use-after Free: Example

- Use-after-Free (UaF) is a bug where a program uses a pointer to a previously freed heap chunk.

- An attacker controlling the freed chunk (i.e., reuse of the chunk) can manipulate the program's behavior.



Existing Approaches

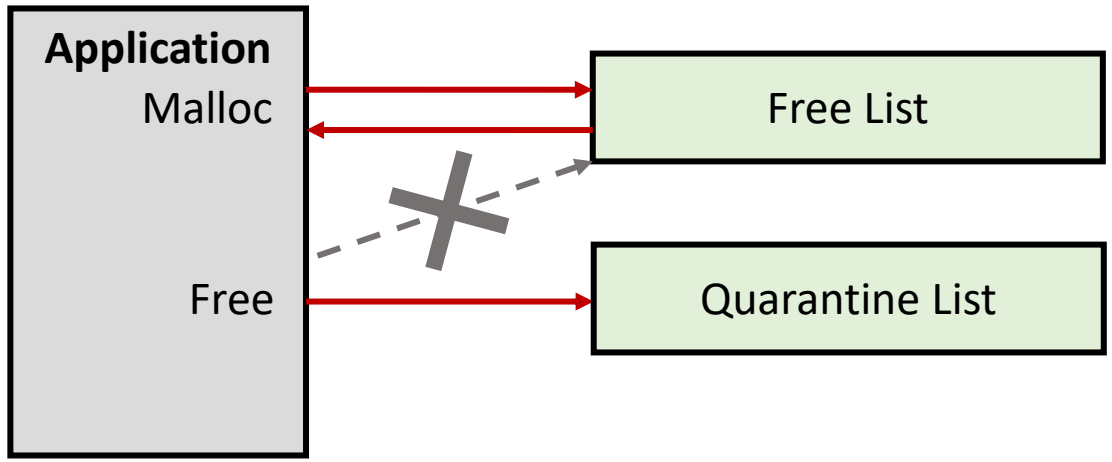
- Garbage collector-like (MarkUs, MineSweeper)
  - ➢ Reuse delayed freed chunks after Mark-Sweep to know the dangling pointer's existence.
- One-time Allocation (FFmalloc)
  - ➢ Only use the allocated region at once.

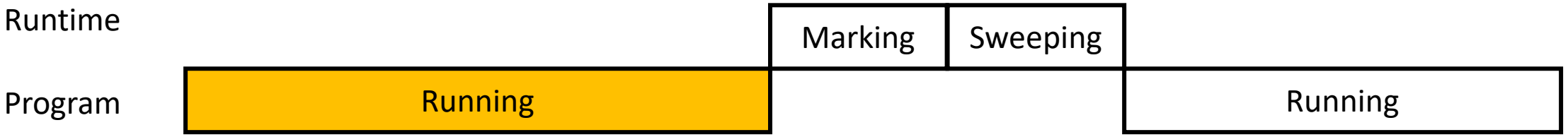
Binary-only
No Recompilation
No Custom Hardware

- Reference counting (CRCount) **Recompilation**
- Pointer nullification (DangNULL) **Recompilation**
- Access validation (ViK, PACMem) **Recompilation or Custom Hardware**

Garbage Collector-like: Allocation/Free

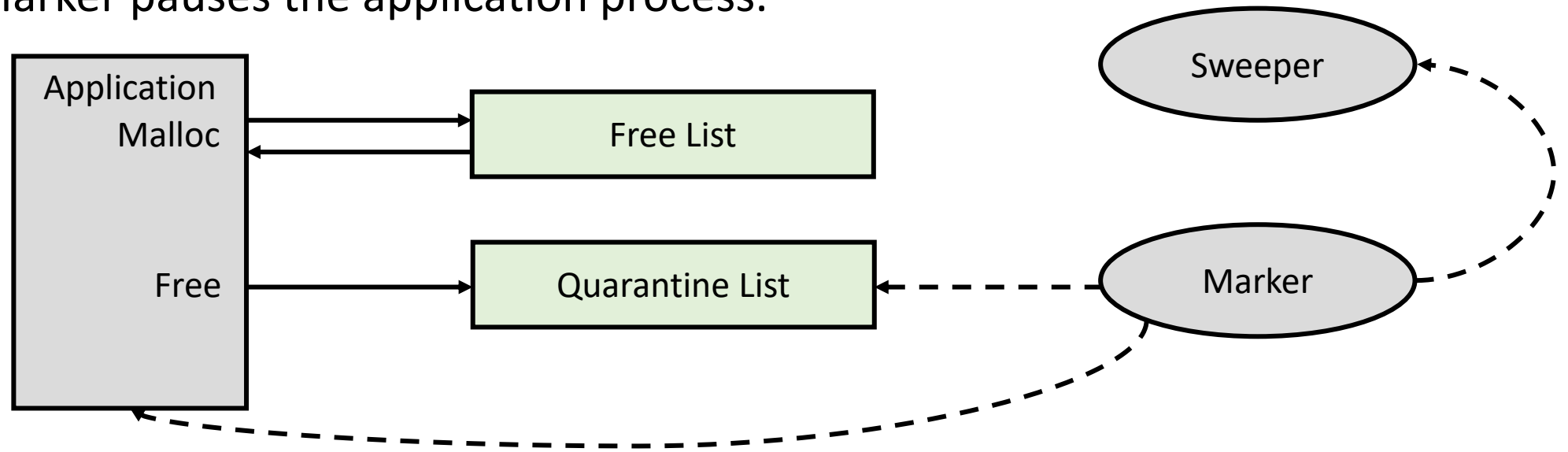
- Garbage Collector-like approaches delay deallocations.

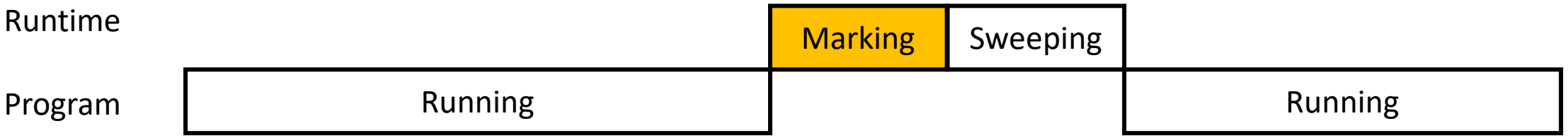




Garbage Collector-like: Marking

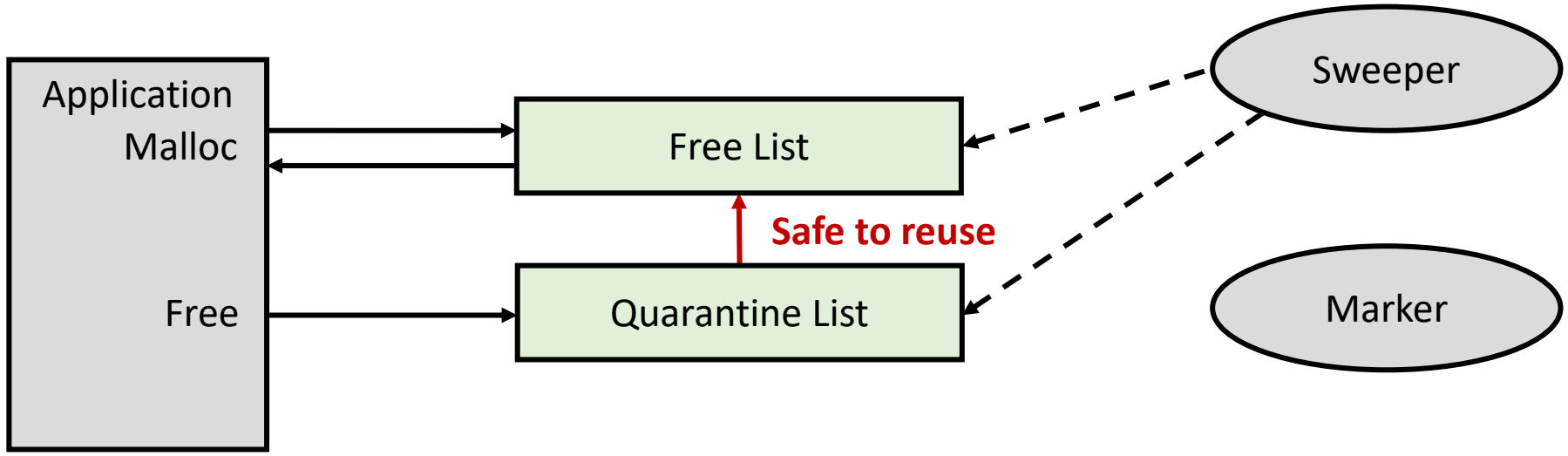
- Marker determines if each chunk can be safely reused or not by memory scanning.
- Marker pauses the application process.

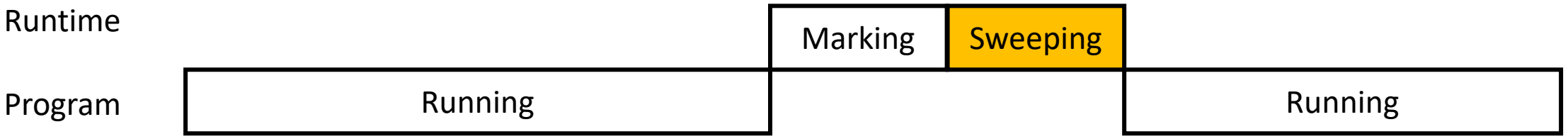




Garbage Collector-like: Sweeping

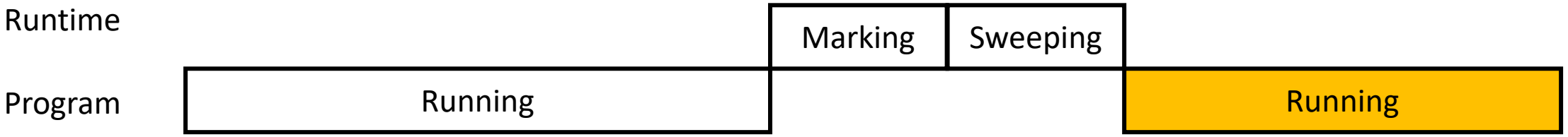
- Sweeper traverses quarantine list and inserts safe objects to the free list.





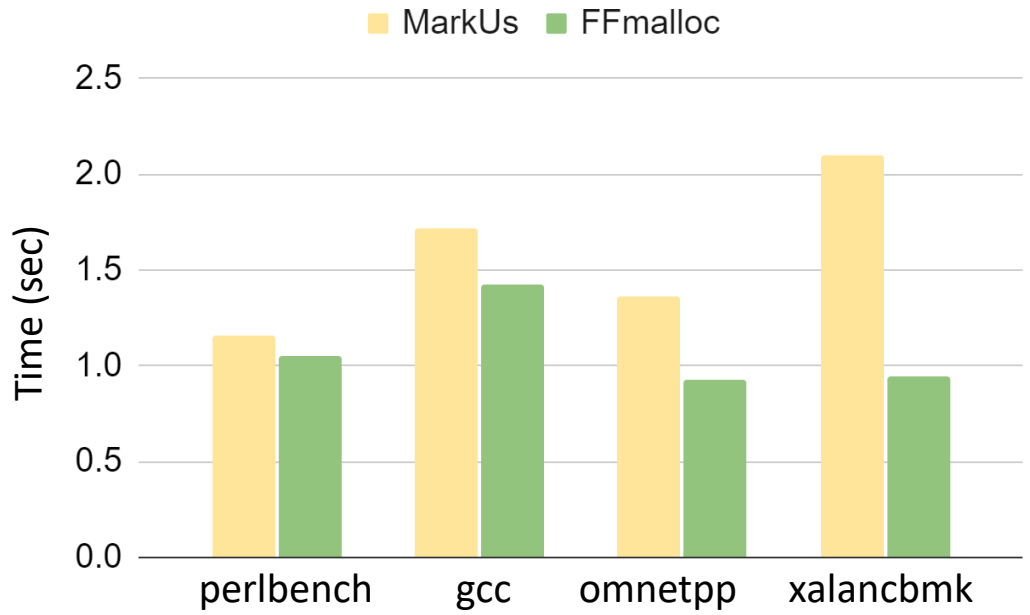
Garbage Collector-like: Reuse

- All chunks in the free list are guaranteed to be safe to reuse.



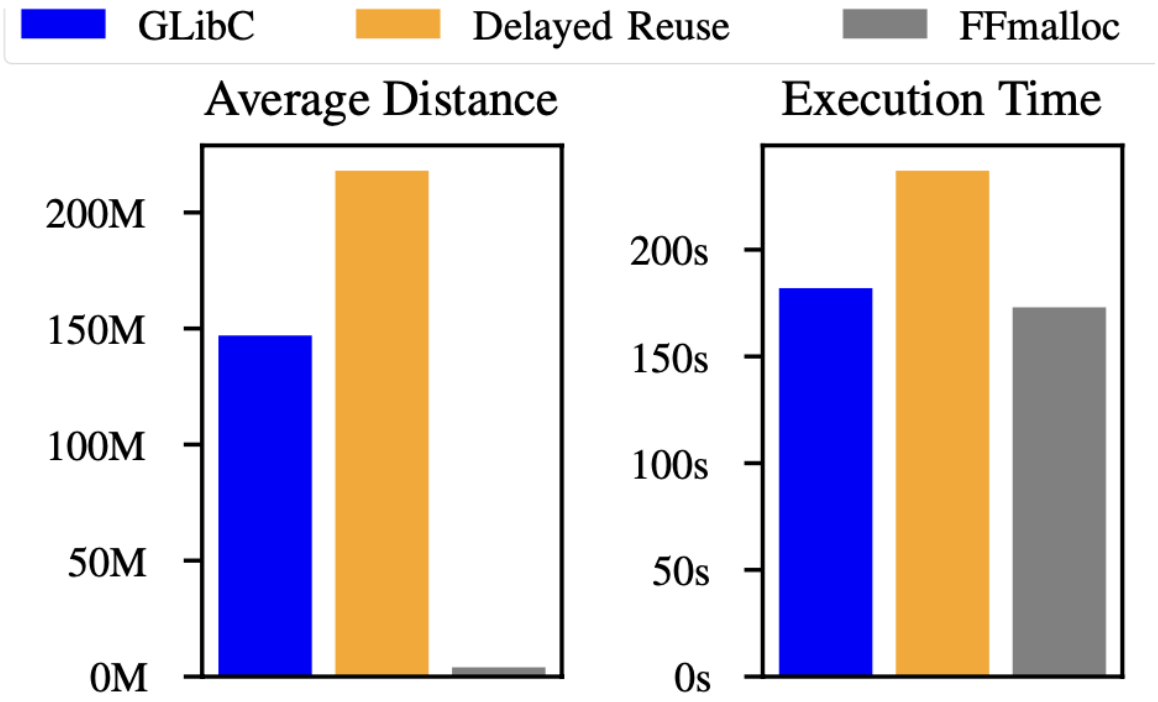
Observations

- Garbage collector-like approaches suffer from significant overhead on the execution time for allocation-intensive benchmarks.
- One-time allocator (OTA) does not … why?



Delayed Reuse Lowers Spatial Locality and Performance

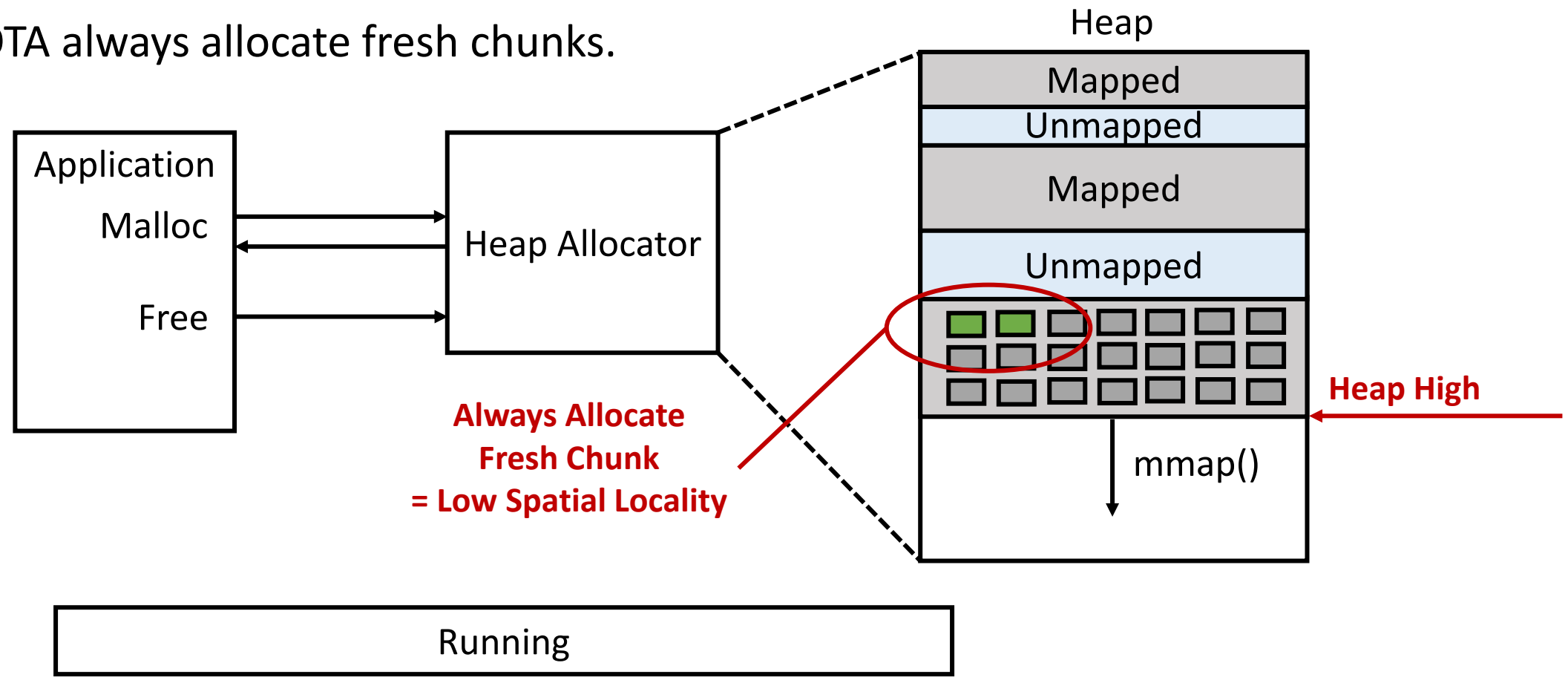
- **Delayed Reuse**: simply delays deallocations and reallocate them (no safety check).
- Just delaying the reuse lowers spatial locality of temporally local allocations.
- OTA does not harm the spatial locality.



Measurements while running SPEC CPU 2006 **xalancbmk**

One-time Allocation Details

- FFmalloc does not reuse virtual address space.
- OTA always allocate fresh chunks.

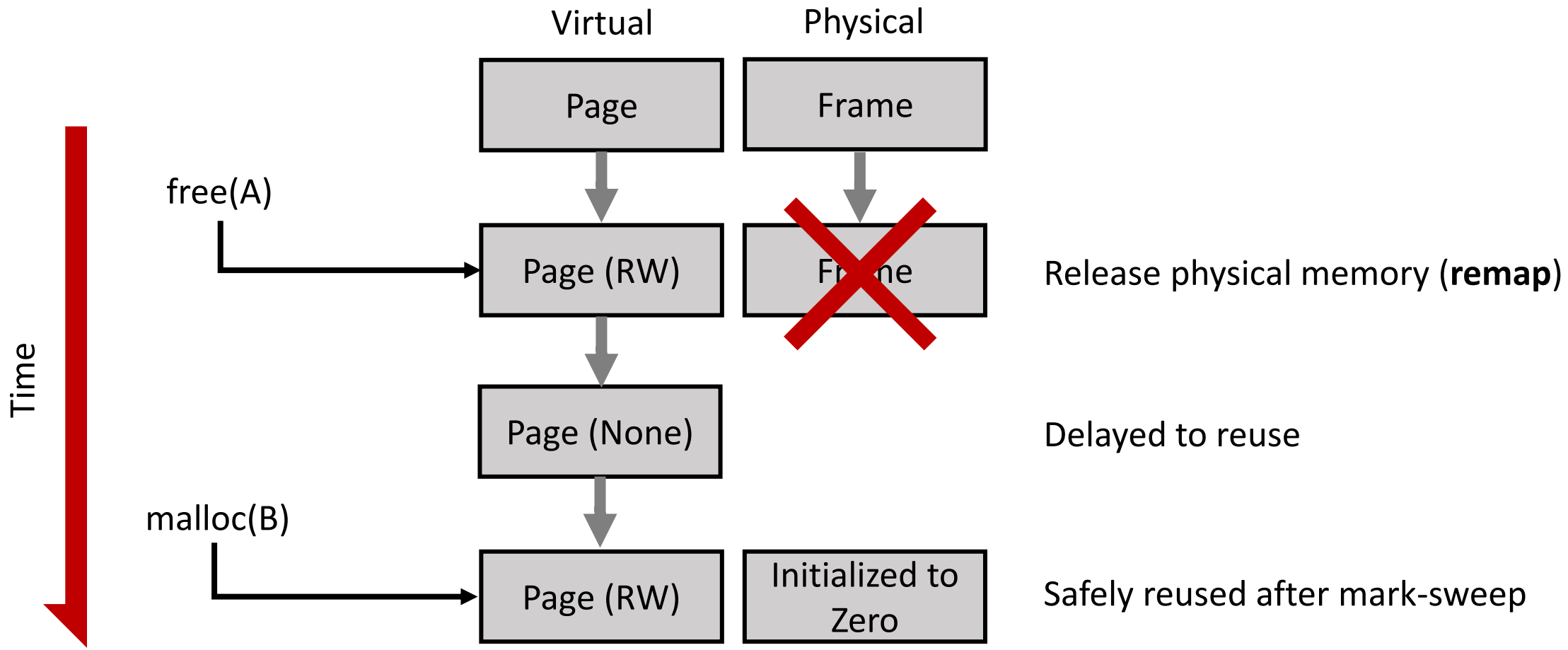


Our Approach

- Garbage Collector-like
  - ➢ **Significant overhead for allocation-intensive benchmarks**
- One-time Allocation
  - ➢ **Do not support indefinite applications**

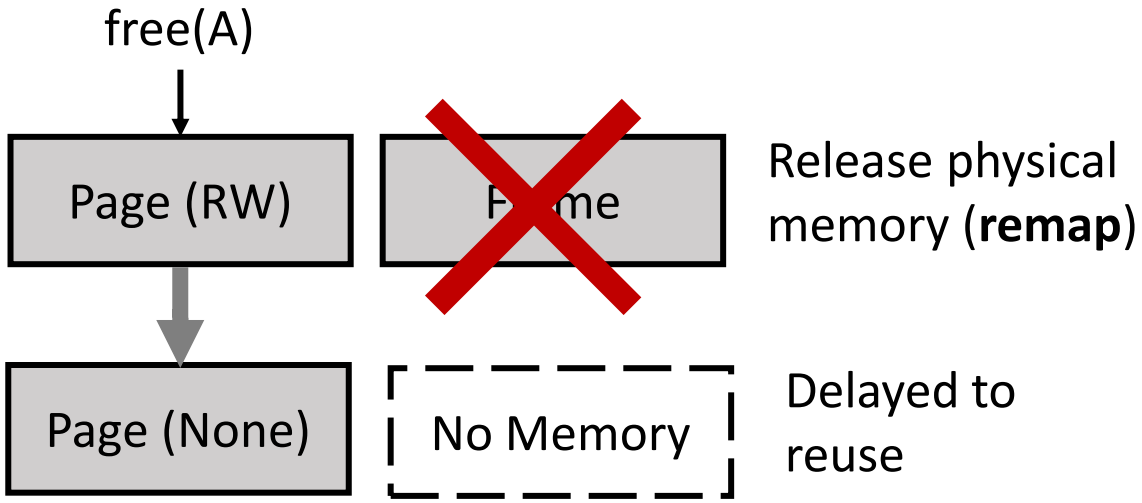
- HushVac
  - ➢ **Mark-Sweep allocator having allocation strategies of FFmalloc**

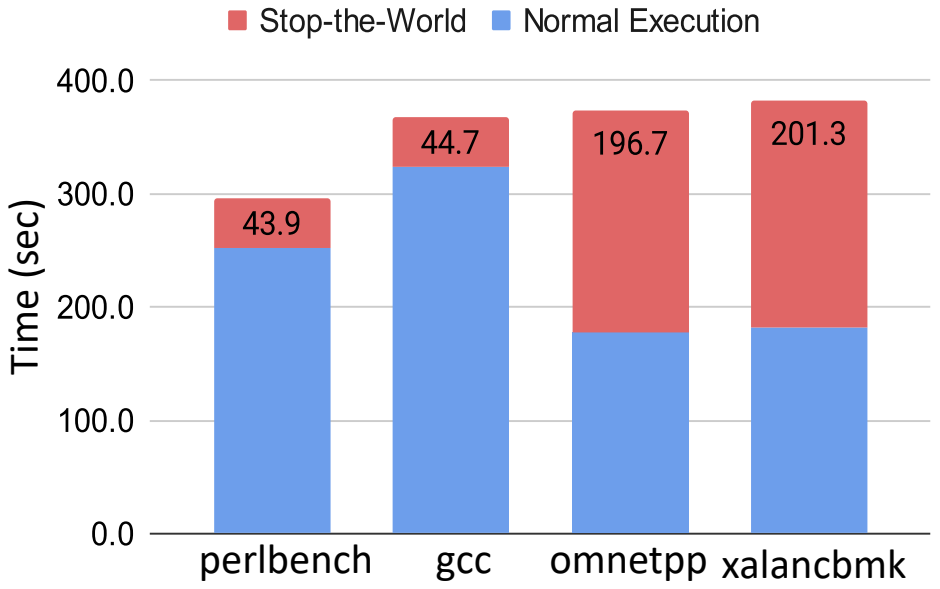
Virtual Address Space Reuse



Reuse Virtual Pages Opportunistically

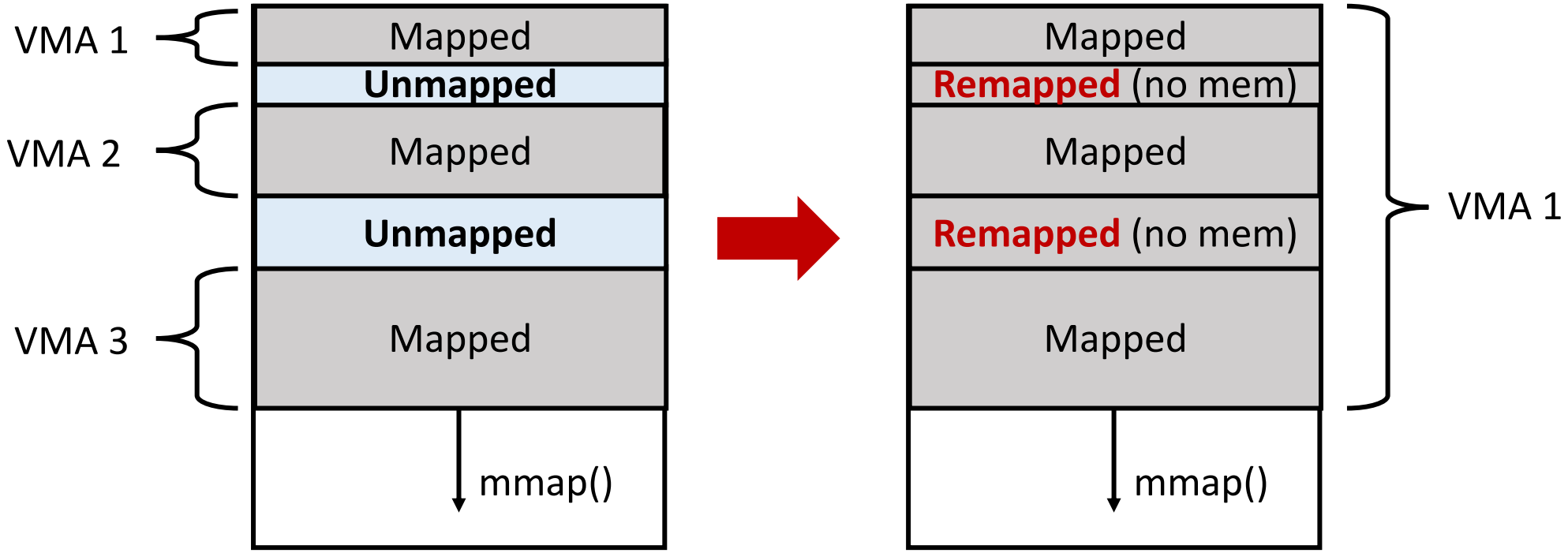
- Ok to do so because:
  - Long quarantine list does not imply the waste of physical memory.
  - Delaying the reuse of virtual pages does not lower spatial locality.
- Desired to do so because stop-the-world time becomes the increased exec time.





OK to Reuse at Page Level Rather than as a Batch

- HushVac detaches physical pages without splitting VMA unlike FFmalloc.
- Remapped virtual address space enables no concerns about VMA fragmentations.

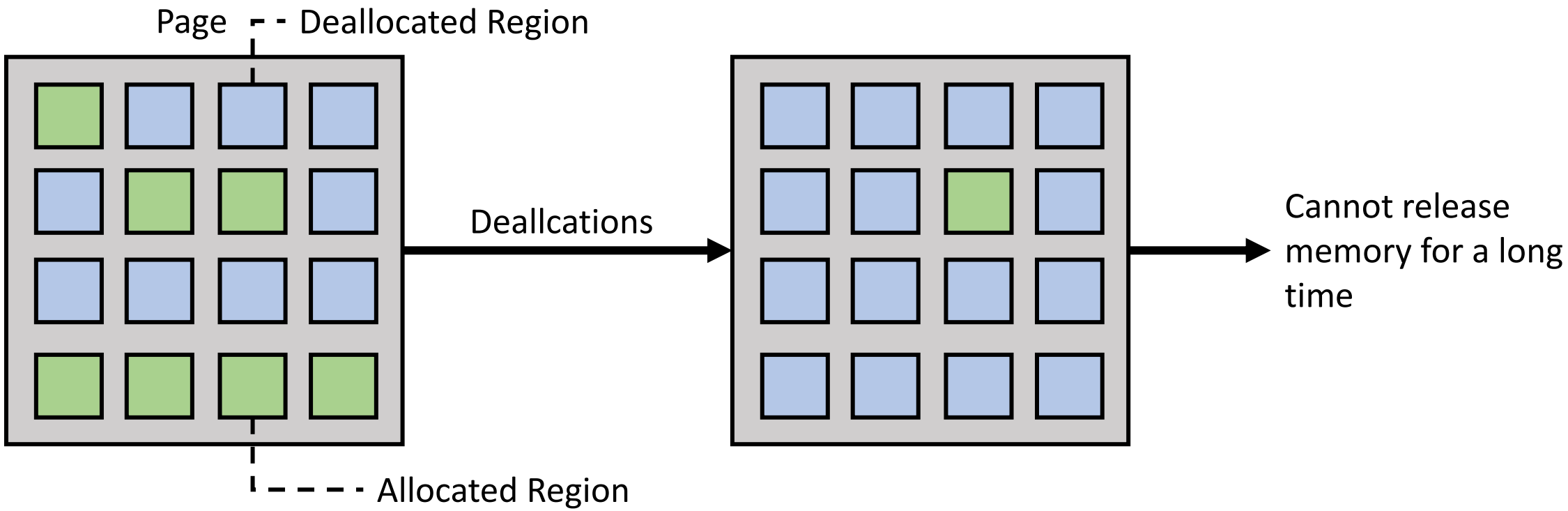


Additional Design Choices

- **Subpage reuse**
- Two staged marking
  - Concurrent marking to reduce stop-the-world cost.
- Comprehensive scanning
  - Scanning entire memory more.

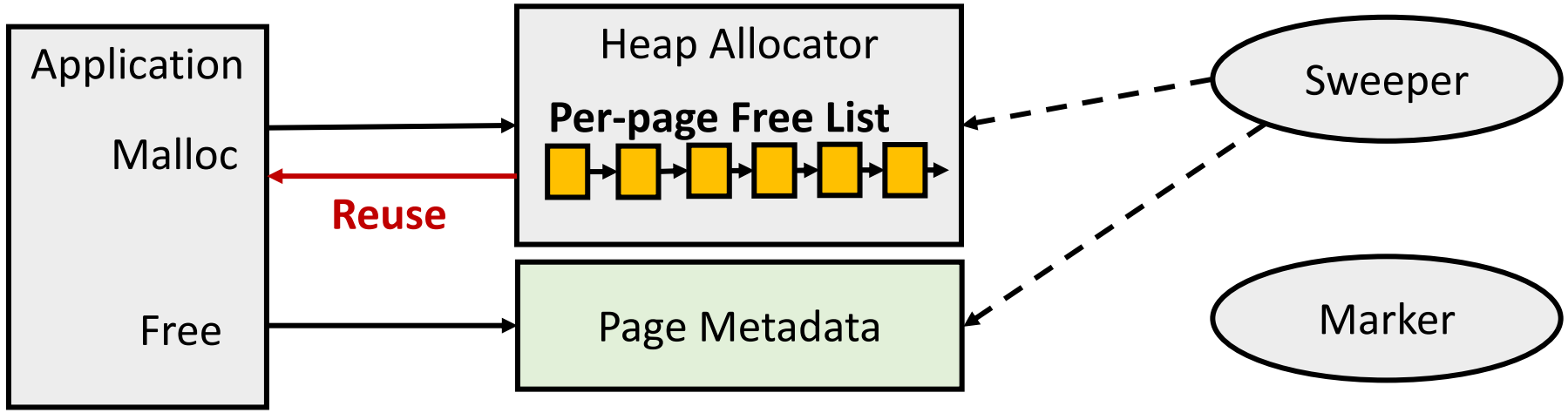
Internal Fragmentation of FFmalloc

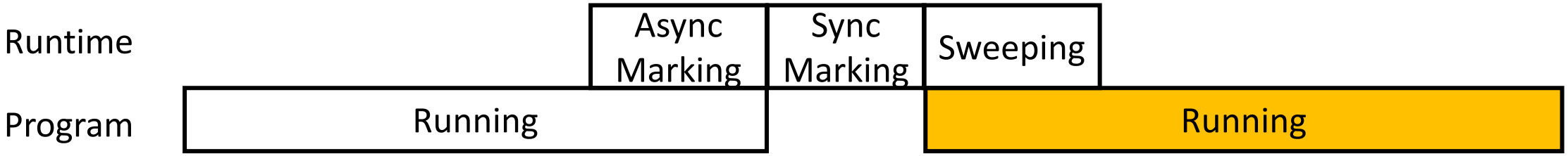
- Long-lived objects prevent a page from being released.



Subpage Reuse for Mitigating Internal Fragmentation

- HushVac maintains a per-page free list inspired by mimalloc.
- HushVac consumes the free list as much as possible.

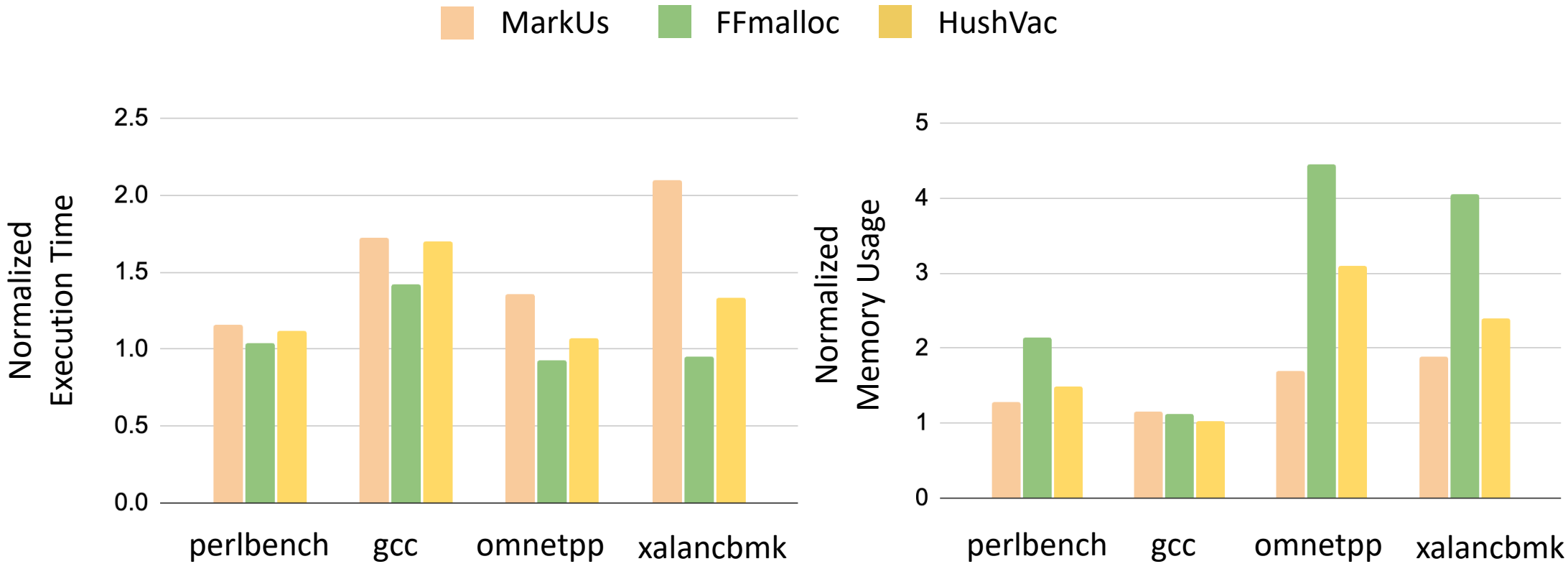




Experimental Setup

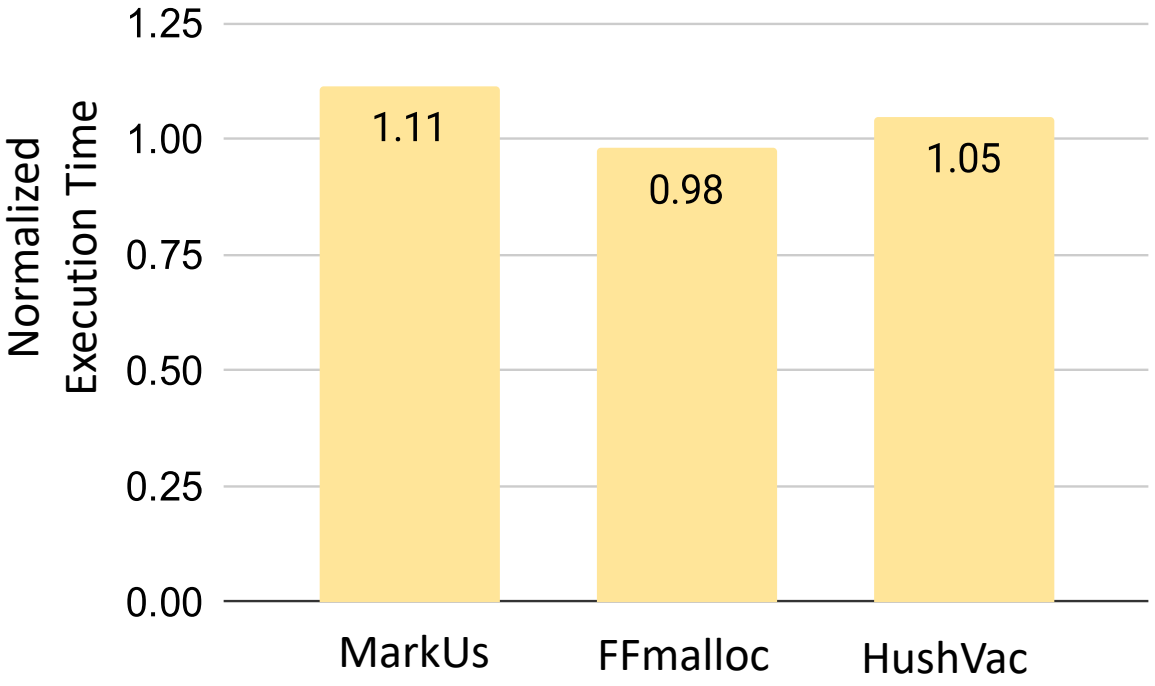
- Ubuntu 18.04 with Linux 5.4.0-150-generic
- AMD Ryzen 5 2600
- 32GB Main Memory
- HushVac runs one mark-sweep thread and 10 marker threads
- The baseline is glibc

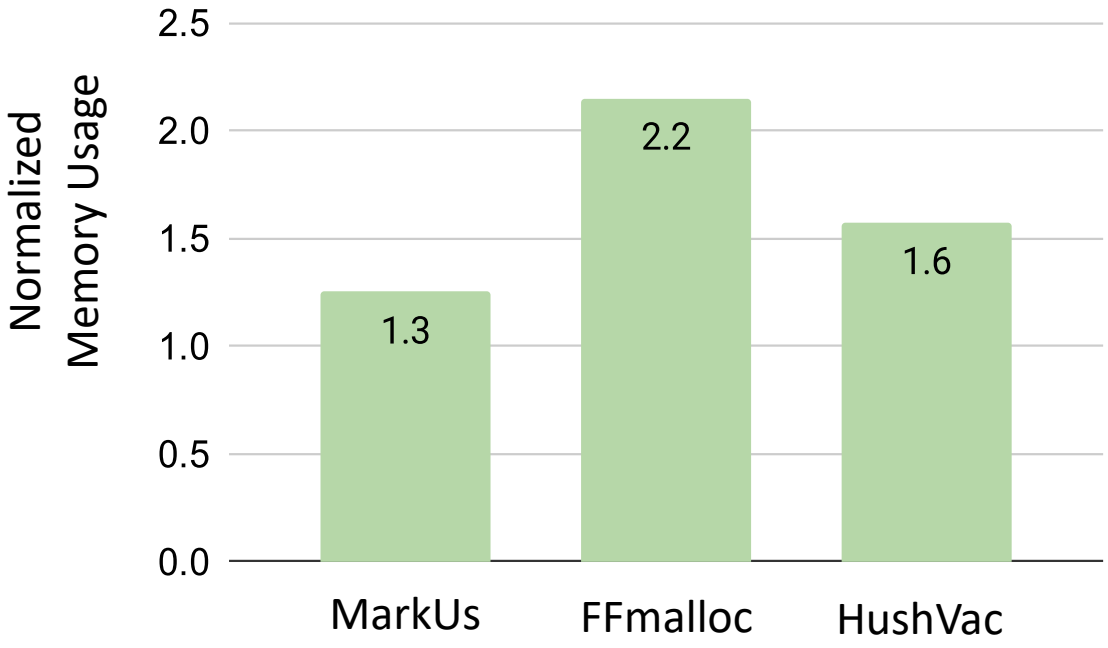
Performance in Allocation-intensive Benchmarks



HushVac is faster than Markus and has lower memory usage than FFmalloc.

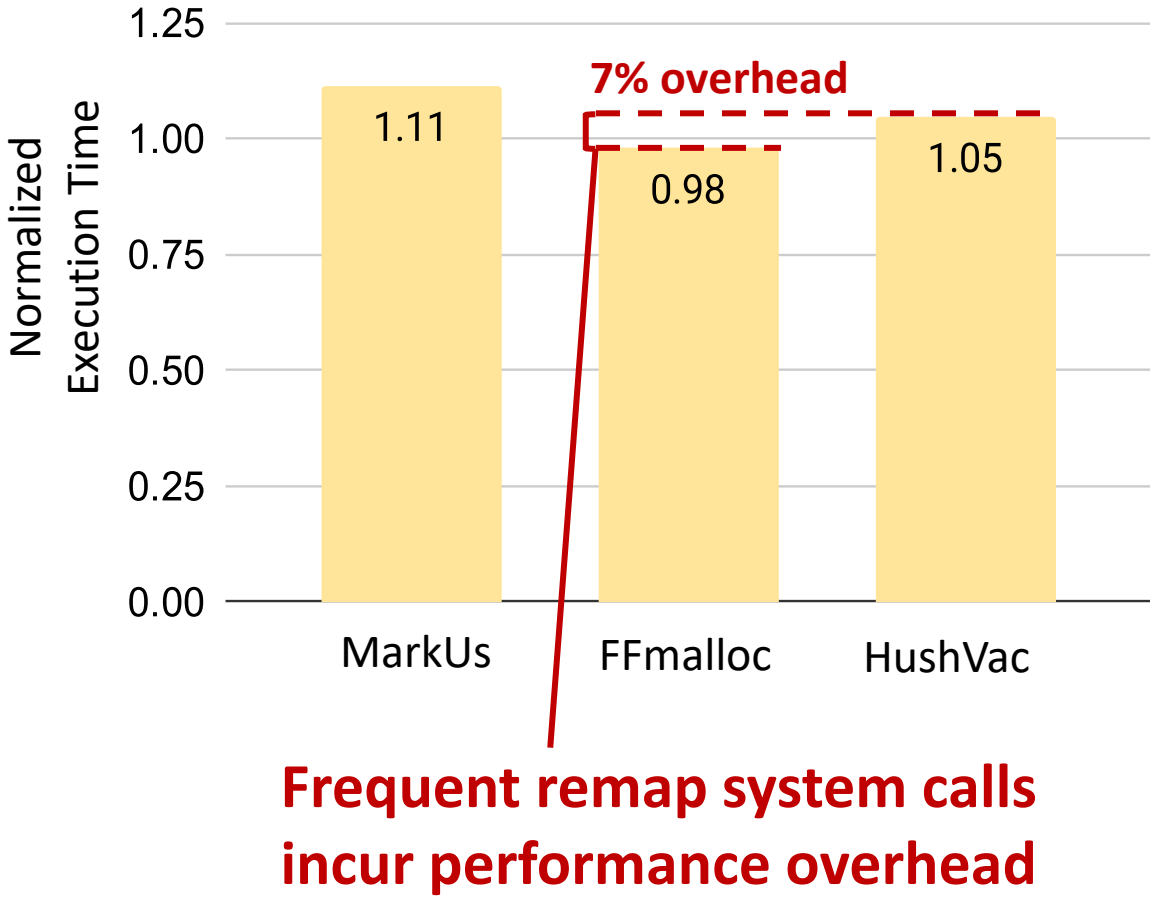
Average Performance on SPEC 2006





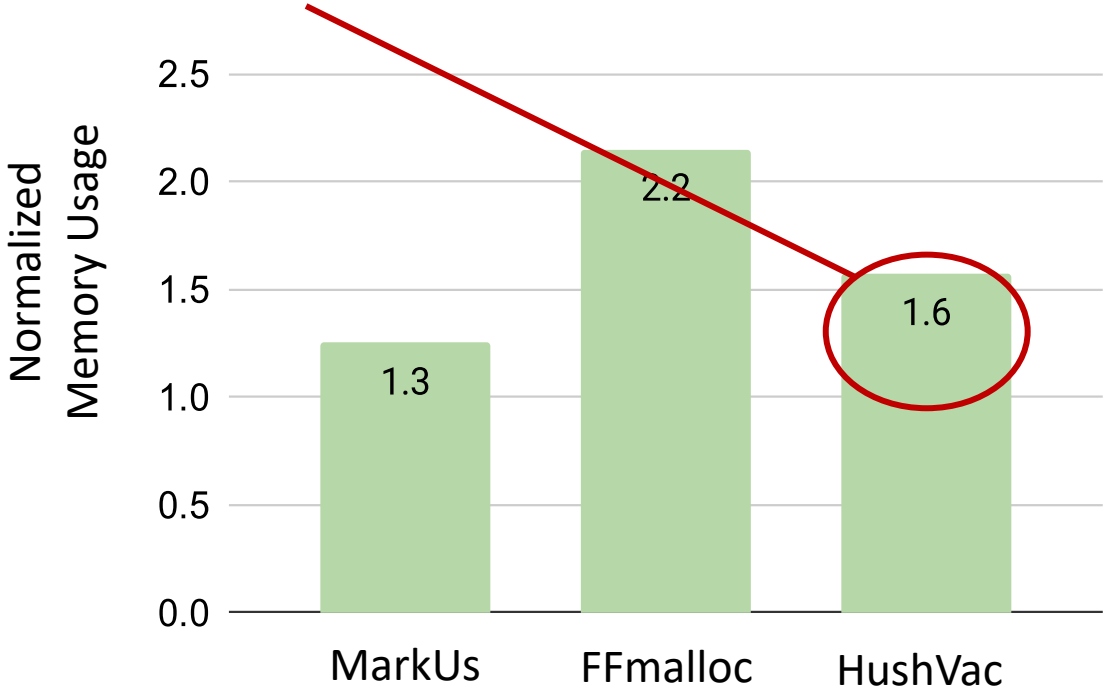
HushVac is faster than Markus and has lower memory usage than FFmalloc.

Limitations



Frequent remap system calls incur performance overhead

Internal fragmentation as FFmalloc
It is impossible to fully safely reuse every page



Conclusion

- The root cause of overhead in garbage collector-like approaches.
  - The spatial locality of temporally local allocations affects the performance.
  - Simply delaying the reuse of freed chunks reduces spatial locality.

- Giving preference to top chunks, as in the OTA, results in higher spatial locality.

- Combining the strengths of the two with several design choices leads to HushVac:
  - Allocation that is aware of spatial locality for both fresh and previously freed chunks.
  - Reduced performance overhead compared to the garbage collector-like approach.
  - Decreased memory overhead and additional ability to reuse chunks compared to OTA.