Google

DEcoupled Fragmentation-Resistant Allocation Groups (DEFRAG)

AKA Page Allocator v2

Goals

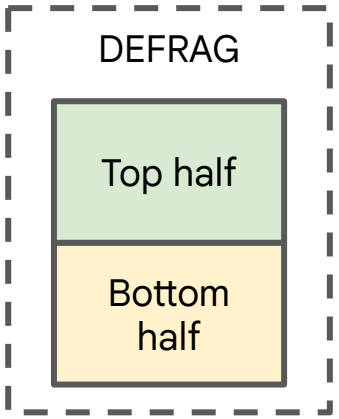
- Profitability
  - Break the zero-sum game
  - Shift the cost back to who incurs it
- Economic models
  - Overbooking
  - PAYG

The cost

- Physical contiguousness
  - Reduces h/w overhead, e.g., TLB misses
  - Reduces s/w overhead, e.g., metadata
- Mobility
  - Reversible v.s. irreversible fragmentation
  - A grouping policy favoring mobile allocations
- Reclaimability
  - To compact, or to reclaim, that is the question
  - A better frame of reference to answer that

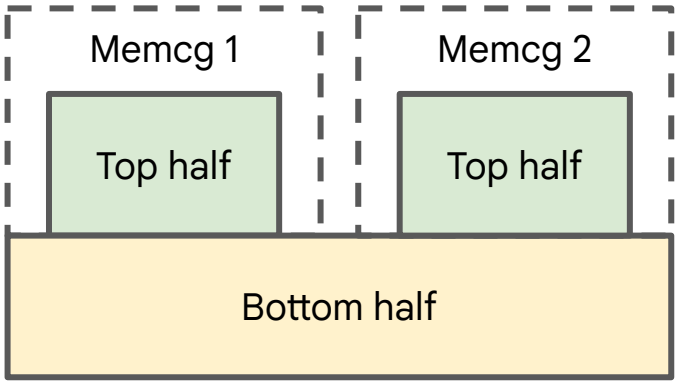
Top and bottom halves

- Bottom half
  - Manages 2MB blocks
  - Treats contiguousness as a resource
- Top half
  - Manages base pages
  - Maintains API compatibility with the current page allocator



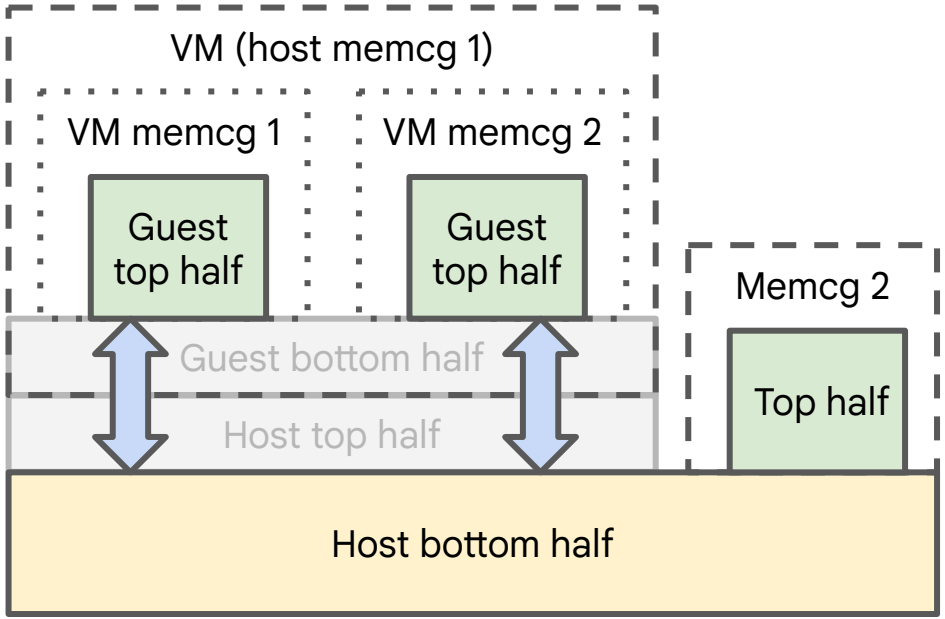
Memcgs

- Blocks are charged to memcgs
  - In addition to page usage
  - Enforces fragmentation isolation
- Compaction becomes per memcg
  - Linked list based, not PFN based
  - Targets the culprit
- Migration between v1 memcgs
  - Requires page migration



VMs

- Can share a single pool of blocks
  - Communicate through hypercalls
  - Return free blocks to the host, hence PAYG
  - Blocks zeroed only by the host
  - No `struct page []` in the host



Blocks

- Grouping policy
  - Differentiates “good/bad” allocations
  - E.g., mobile allocations use immobile blocks & immobile allocations pay for migration
- Runtime behavior awareness
  - Hotness (coldness) and lifetime
  - Coordination between compaction & reclaim

Metadata

- Per block metadata
  - Allocated at boot time
  - A fraction of the size of `struct page []`
    - Short term: ⅛ (similar to HVO)
    - Long term: 1% (breaks arithmetics on `*page`)
  - Sufficient for huge pages (THP and HugeTLB)
- Per page metadata
  - Allocated on split
  - Charged to the splitter
  - Freed when the block becomes empty

State of the art

- Hardware acceleration and fault tolerance
  - DMA zeroing
  - Hwpoison
- Physical address space engineering
  - PGHO interoperable
  - NUMA/tiering aware
- Separation of mechanism and policy
  - BPF interoperable
  - Rowhammer/cache coloring aware