The hard work behind large physical memory allocations in the kernel

Vlastimil Babka
SUSE Labs
vbabka@suse.cz

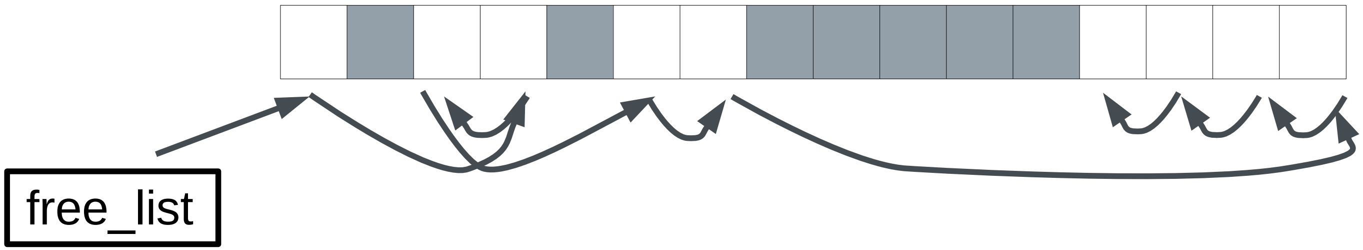
We adapt. You succeed.

Physical Memory Allocator

- Physical memory is divided into several zones
  - 1+ zone per NUMA node
- Binary **buddy allocator** for pages in each zone
  - Free *base pages* (e.g. 4KB) coalesced to groups of power-of-2 pages (naturally aligned), put on *free lists*
  - Exponent = page *order*; 0 for 4KB → 10 for 4MB pages
  - Good performance, finds page of requested order instantly

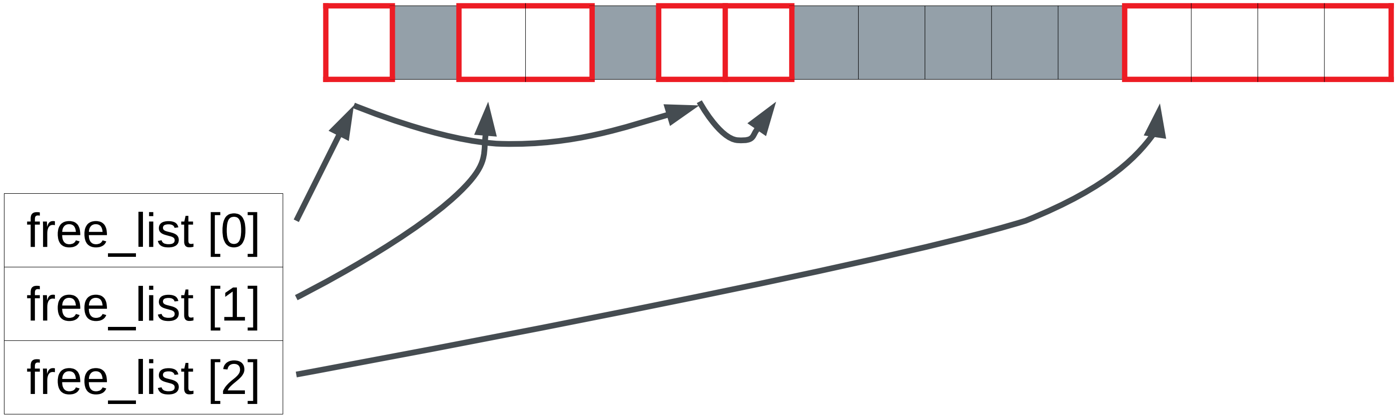
Physical Memory Allocator

- Physical memory is divided into several zones
  - 1+ zone per NUMA node
- Binary **buddy allocator** for pages in each zone
  - Free *base pages* (e.g. 4KB) coalesced to groups of power-of-2 pages (naturally aligned), put on *free lists*
  - Exponent = page *order*; 0 for 4KB → 10 for 4MB pages
  - Good performance, finds page of requested order instantly



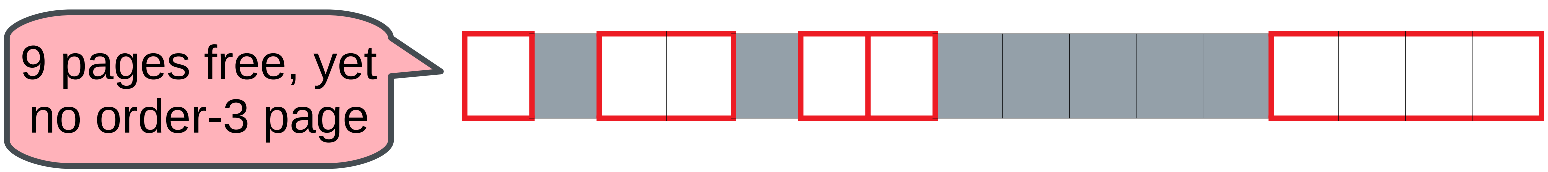
Physical Memory Allocator

- Physical memory is divided into several zones
  - 1+ zone per NUMA node
- Binary **buddy allocator** for pages in each zone
  - Free *base pages* (e.g. 4KB) coalesced to groups of power-of-2 pages (naturally aligned), put on *free lists*
  - Exponent = page *order*; 0 for 4KB → 10 for 4MB pages
  - Good performance, finds page of requested order instantly



Physical Memory Allocator

- Physical memory is divided into several zones
  - 1+ zone per NUMA node
- Binary **buddy allocator** for pages in each zone
  - Free *base pages* (e.g. 4KB) coalesced to groups of power-of-2 pages (naturally aligned), put on *free lists*
  - Exponent = page *order*; 0 for 4KB → 10 for 4MB pages
  - Good performance, finds page of requested order instantly
- Problem: allocations of order > 0 may fail due to *(external) memory fragmentation*
  - There is enough free memory, but not contiguous



Why We Need High-order Allocations?

- Huge pages for userspace (both hugetlbfs and THP)
  - 2MB is order-9; 1GB is order-18 (but max order is 10...)
- Other physically contiguous area of memory
  - Buffers for hardware that requires it (no scatter/gather)
  - Potentially page cache (64KB?)
- Virtually contiguous area of memory
  - Kernel stacks until recently (order-2 on x86), now vmalloc
  - SLUB caches (max 32KB by default) for performance reasons
    - Fallback to smaller sizes when possible – generally advisable
  - vmalloc is a generic alternative, but not for free
    - Limited area (on 32bit), need to allocate and setup page tables…
    - Somewhat discouraged, but now a kvmalloc() helper exists

Example: Failed High-order Allocation

[874475.784075] **chrome: page allocation failure: order:4, mode:0xc0d0**
[874475.784079] CPU: 4 PID: 18907 Comm: chrome Not tainted 3.16.1-gentoo #1
[874475.784081] Hardware name: Dell Inc. OptiPlex 980 /0D441T, BIOS A15 01/09/2014
[874475.784318] Node 0 DMA free:15888kB min:84kB low:104kB high:124kB active_anon:0kB inactive_anon:0kB active_file:0kB inactive_file:0kB unevictable:0kB isolated(anon):0kB isolated(file):0kB present:15988kB managed:15904kB mlocked:0kB dirty:0kB writeback:0kB mapped:0kB shmem:0kB slab_reclaimable:0kB slab_unreclaimable:16kB kernel_stack:0kB pagetables:0kB unstable:0kB bounce:0kB free_cma:0kB writeback_tmp:0kB pages_scanned:0 all_unreclaimable? Yes
[874475.784322] lowmem_reserve[]: 0 3418 11929 11929
[874475.784325] **Node 0 DMA32 free:157036kB** min:19340kB low:24172kB high:29008kB active_anon:1444992kB inactive_anon:480776kB active_file:538856kB inactive_file:513452kB unevictable:0kB isolated(anon):0kB isolated(file):0kB present:3578684kB managed:3504680kB mlocked:0kB dirty:1304kB writeback:0kB mapped:157908kB shmem:85752kB slab_reclaimable:278324kB slab_unreclaimable:20852kB kernel_stack:4688kB pagetables:28472kB unstable:0kB bounce:0kB free_cma:0kB writeback_tmp:0kB pages_scanned:0 all_unreclaimable? no
[874475.784329] lowmem_reserve[]: 0 0 8510 8510

- [874475.784332] **Node 0 Normal free:100168kB** min:48152kB low:60188kB high:72228kB active_anon:4518020kB inactive_anon:746232kB active_file:1271196kB inactive_file:1261912kB unevictable:96kB isolated(anon):0kB isolated(file):0kB present:8912896kB managed:8714728kB mlocked:96kB dirty:5224kB writeback:0kB mapped:327904kB shmem:143496kB slab_reclaimable:502940kB slab_unreclaimable:52156kB kernel_stack:11264kB pagetables:70644kB unstable:0kB bounce:0kB free_cma:0kB writeback_tmp:0kB pages_scanned:0 all_unreclaimable? no

[874475.784338] Node 0 DMA: 0*4kB 0*8kB 1*16kB (U) 2*32kB (U) **1*64kB (U) 1*128kB (U) 1*256kB (U) 0*512kB 1*1024kB (U) 1*2048kB (R) 3*4096kB (M)** = 15888kB
[874475.784348] Node 0 DMA32: 31890*4kB (UEM) 3571*8kB (UEM) 31*16kB (UEM) 16*32kB (UMR) **6*64kB (UEMR) 1*128kB (R) 0*256kB 0*512kB 1*1024kB (R) 0*2048kB 0*4096kB** = 158672kB
[874475.784358] Node 0 Normal: 22272*4kB (UEM) 726*8kB (UEM) 75*16kB (UEM) 24*32kB (UEM) **1*64kB (M) 0*128kB 0*256kB 0*512kB 0*1024kB 0*2048kB 1*4096kB (R)** = 101024kB
[874475.784378] **[drm:radeon_cs_ioctl] *ERROR* Failed to parse relocation -12!**

Enabling High-Order Allocations

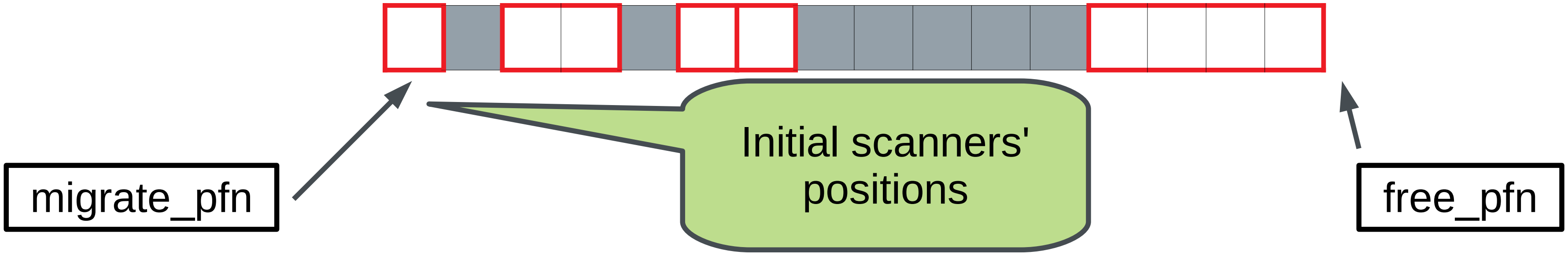
- Prevent memory fragmentation?
  - Buddy allocator design helps by splitting the smallest pages
  - Works only until memory becomes full (which is desirable)
- Reclaim contiguous areas?
  - LRU based reclaim → pages of similar last usage time (*age*) not guaranteed to be near each other physically
  - “Lumpy reclaim” did exist, but it violated the LRU aging
- Defragment memory by moving pages around?
  - *Memory compaction* can do that within each zone
  - Relies on *page migration* functionality

Memory Compaction Overview

- Execution alternates between two page (pfn) scanners
- Migration scanner looks for migration source pages
  - Starts at beginning (first page) of a zone, moves towards end
  - Isolates movable pages from their LRU lists
- Free scanner looks for migration target pages
  - Starts at the end of zone, moves towards beginning
  - Isolates free pages from buddy allocator (splits as needed)

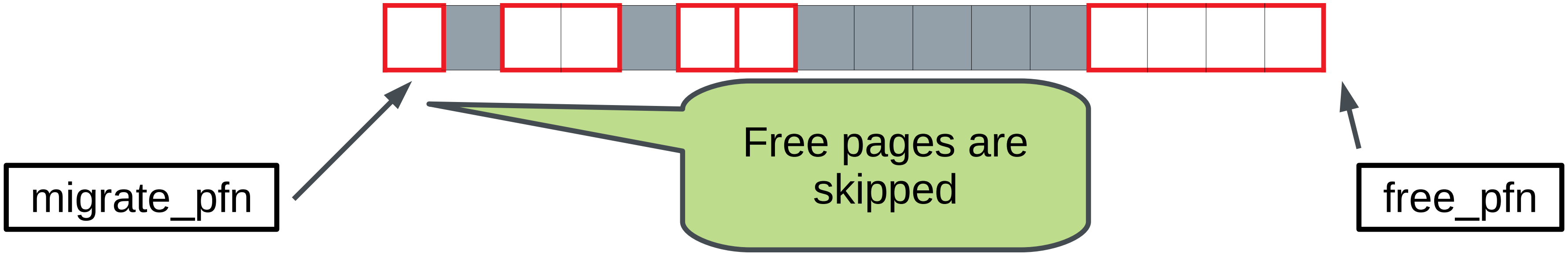
Memory Compaction Overview

- Execution alternates between two page (pfn) scanners
- Migration scanner looks for migration source pages
  - Starts at beginning (first page) of a zone, moves towards end
  - Isolates movable pages from their LRU lists
- Free scanner looks for migration target pages
  - Starts at the end of zone, moves towards beginning
  - Isolates free pages from buddy allocator (splits as needed)



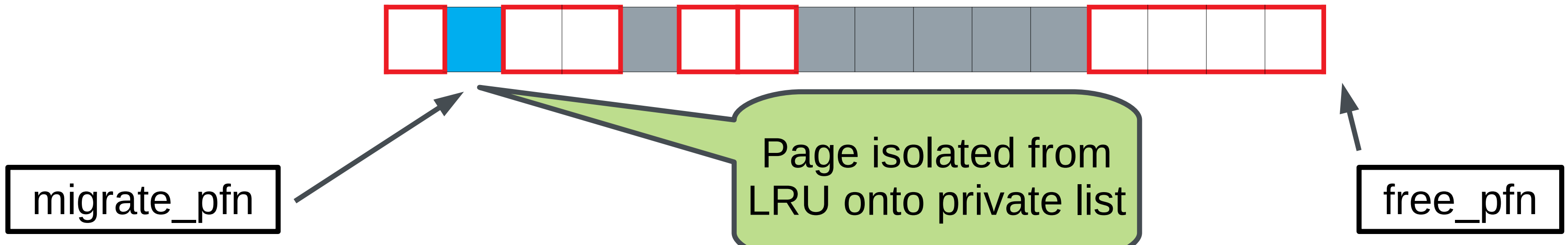
Memory Compaction Overview

- Execution alternates between two page (pfn) scanners
- Migration scanner looks for migration source pages
  - Starts at beginning (first page) of a zone, moves towards end
  - Isolates movable pages from their LRU lists
- Free scanner looks for migration target pages
  - Starts at the end of zone, moves towards beginning
  - Isolates free pages from buddy allocator (splits as needed)



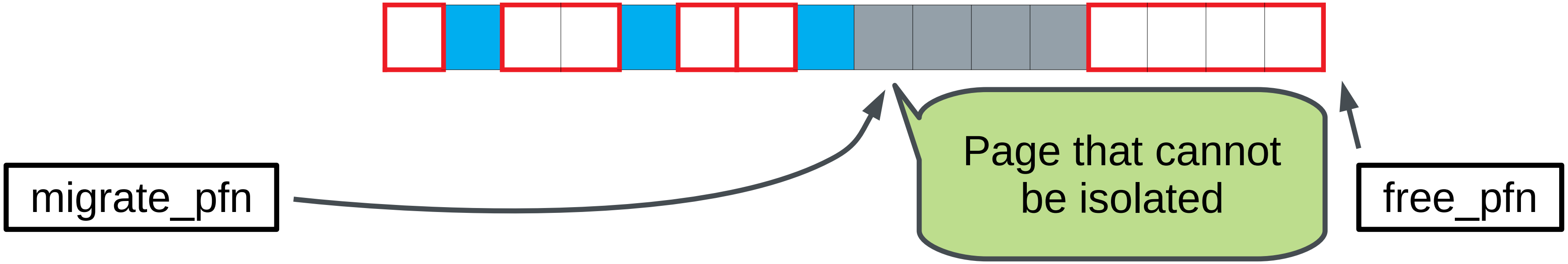
Memory Compaction Overview

- Execution alternates between two page (pfn) scanners
- Migration scanner looks for migration source pages
  - Starts at beginning (first page) of a zone, moves towards end
  - Isolates movable pages from their LRU lists
- Free scanner looks for migration target pages
  - Starts at the end of zone, moves towards beginning
  - Isolates free pages from buddy allocator (splits as needed)



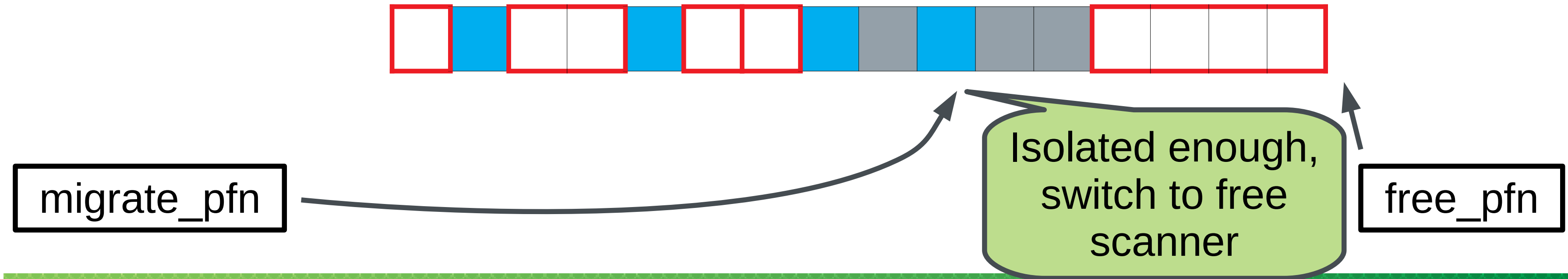
Memory Compaction Overview

- Execution alternates between two page (pfn) scanners
- Migration scanner looks for migration source pages
  - Starts at beginning (first page) of a zone, moves towards end
  - Isolates movable pages from their LRU lists
- Free scanner looks for migration target pages
  - Starts at the end of zone, moves towards beginning
  - Isolates free pages from buddy allocator (splits as needed)



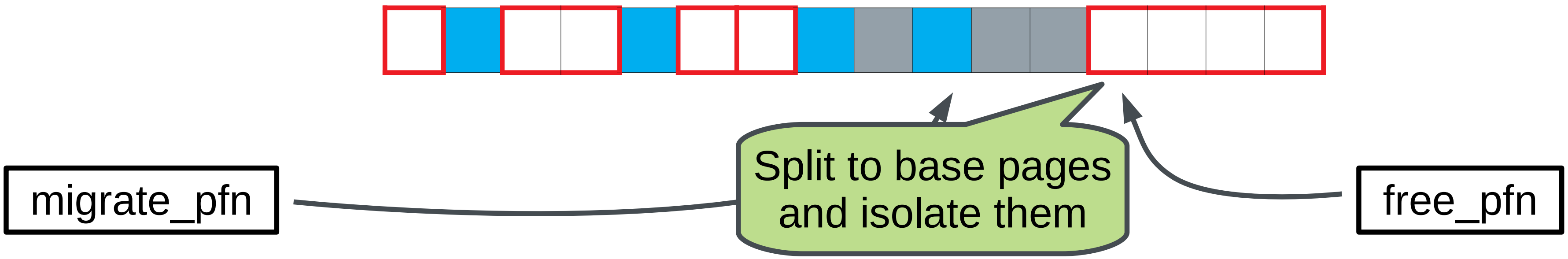
Memory Compaction Overview

- Execution alternates between two page (pfn) scanners
- Migration scanner looks for migration source pages
  - Starts at beginning (first page) of a zone, moves towards end
  - Isolates movable pages from their LRU lists
- Free scanner looks for migration target pages
  - Starts at the end of zone, moves towards beginning
  - Isolates free pages from buddy allocator (splits as needed)



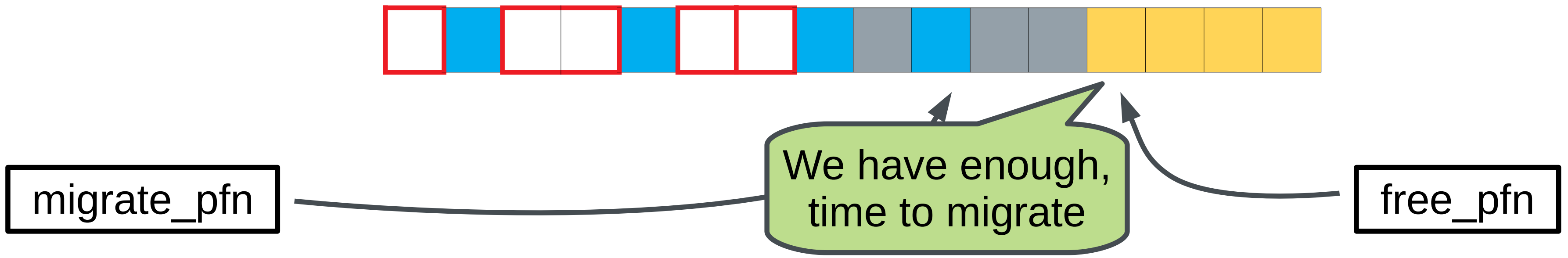
Memory Compaction Overview

- Execution alternates between two page (pfn) scanners
- Migration scanner looks for migration source pages
  - Starts at beginning (first page) of a zone, moves towards end
  - Isolates movable pages from their LRU lists
- Free scanner looks for migration target pages
  - Starts at the end of zone, moves towards beginning
  - Isolates free pages from buddy allocator (splits as needed)



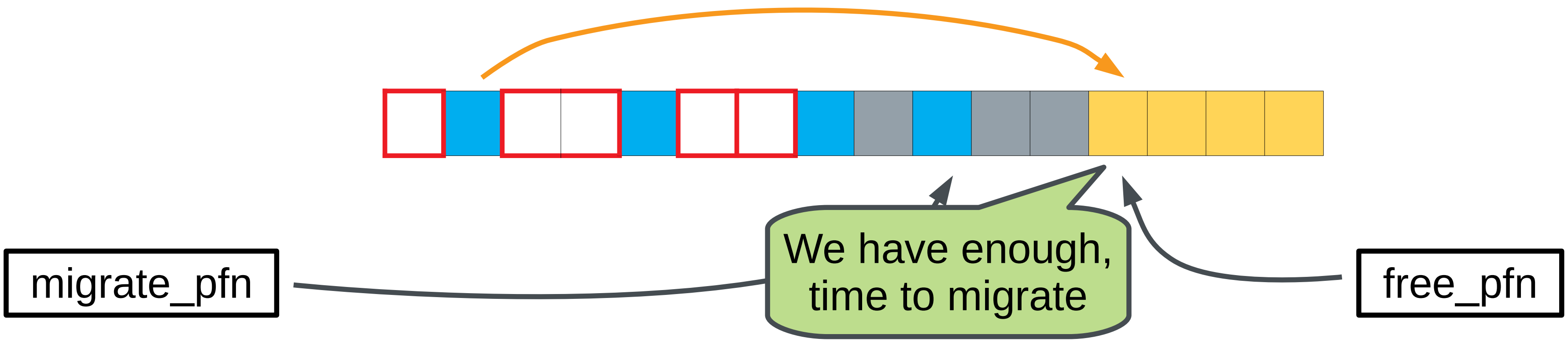
Memory Compaction Overview

- Execution alternates between two page (pfn) scanners
- Migration scanner looks for migration source pages
  - Starts at beginning (first page) of a zone, moves towards end
  - Isolates movable pages from their LRU lists
- Free scanner looks for migration target pages
  - Starts at the end of zone, moves towards beginning
  - Isolates free pages from buddy allocator (splits as needed)



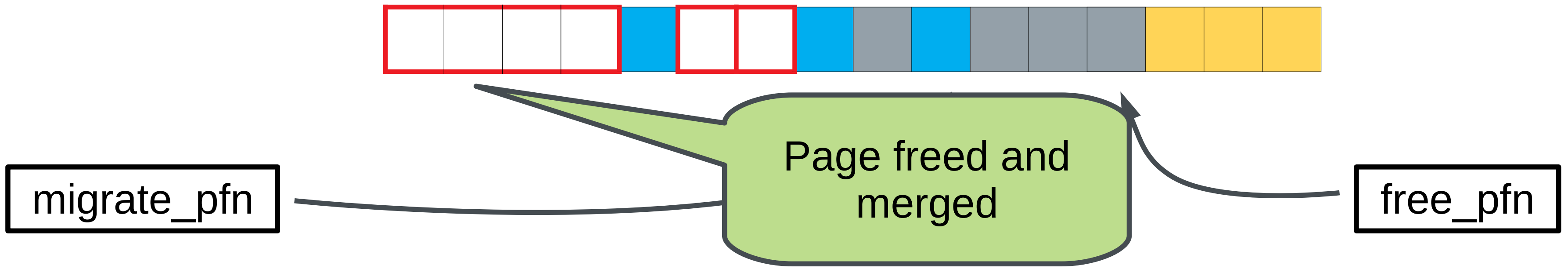
Memory Compaction Overview

- Execution alternates between two page (pfn) scanners
- Migration scanner looks for migration source pages
  - Starts at beginning (first page) of a zone, moves towards end
  - Isolates movable pages from their LRU lists
- Free scanner looks for migration target pages
  - Starts at the end of zone, moves towards beginning
  - Isolates free pages from buddy allocator (splits as needed)



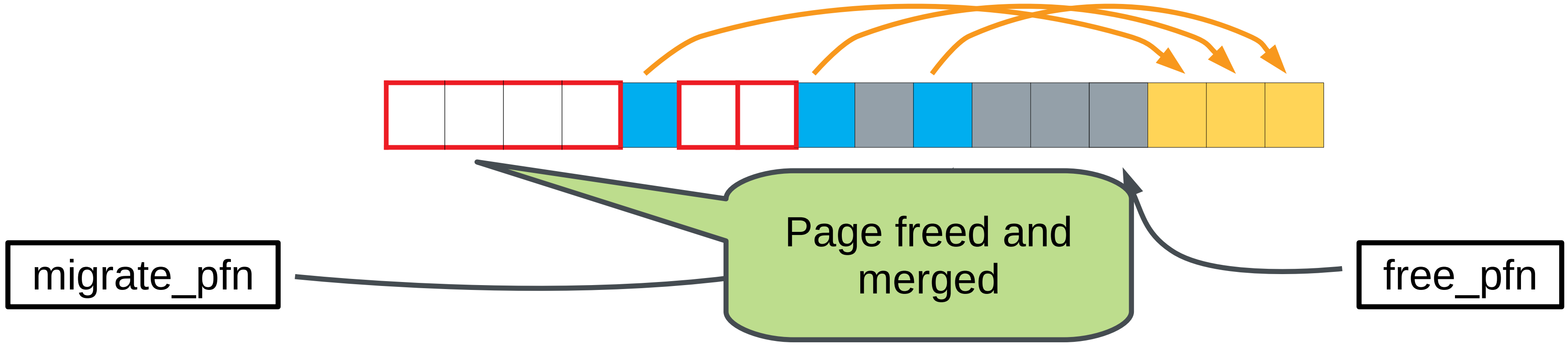
Memory Compaction Overview

- Execution alternates between two page (pfn) scanners
- Migration scanner looks for migration source pages
  - Starts at beginning (first page) of a zone, moves towards end
  - Isolates movable pages from their LRU lists
- Free scanner looks for migration target pages
  - Starts at the end of zone, moves towards beginning
  - Isolates free pages from buddy allocator (splits as needed)



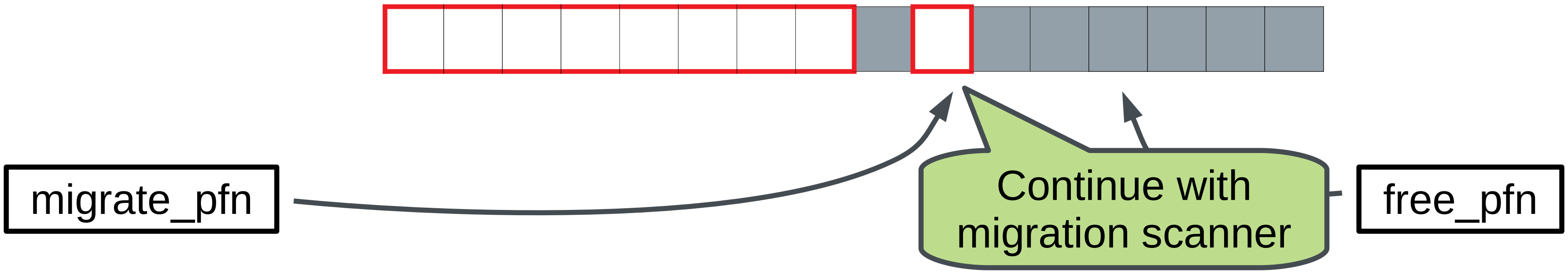
Memory Compaction Overview

- Execution alternates between two page (pfn) scanners
- Migration scanner looks for migration source pages
  - Starts at beginning (first page) of a zone, moves towards end
  - Isolates movable pages from their LRU lists
- Free scanner looks for migration target pages
  - Starts at the end of zone, moves towards beginning
  - Isolates free pages from buddy allocator (splits as needed)



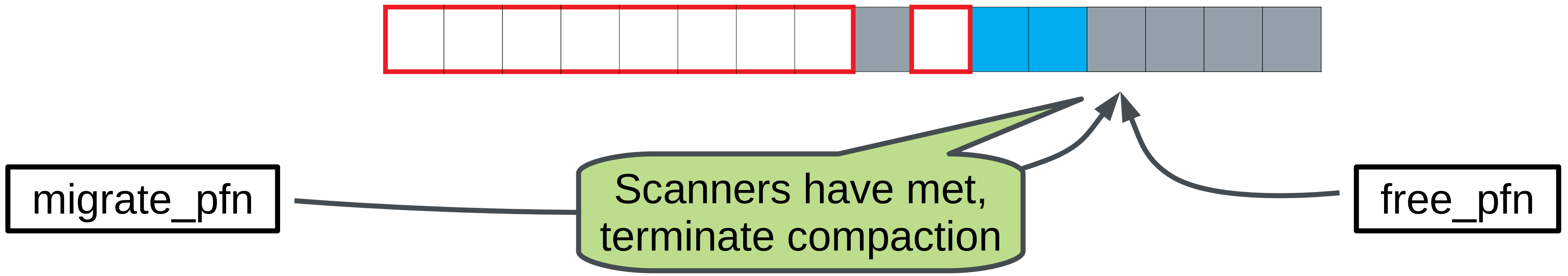
Memory Compaction Overview

- Execution alternates between two page (pfn) scanners
- Migration scanner looks for migration source pages
  - Starts at beginning (first page) of a zone, moves towards end
  - Isolates movable pages from their LRU lists
- Free scanner looks for migration target pages
  - Starts at the end of zone, moves towards beginning
  - Isolates free pages from buddy allocator (splits as needed)



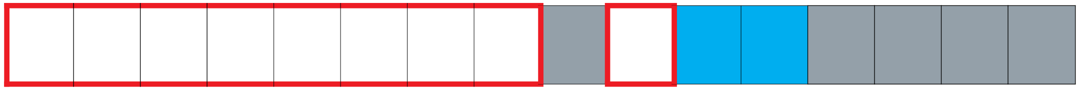
Memory Compaction Overview

- Execution alternates between two page (pfn) scanners
- Migration scanner looks for migration source pages
  - Starts at beginning (first page) of a zone, moves towards end
  - Isolates movable pages from their LRU lists
- Free scanner looks for migration target pages
  - Starts at the end of zone, moves towards beginning
  - Isolates free pages from buddy allocator (splits as needed)



Memory Compaction Overview

- Execution alternates between two page (pfn) scanners
- Migration scanner looks for migration source pages
  - Starts at beginning (first page) of a zone, moves towards end
  - Isolates movable pages from their LRU lists
- Free scanner looks for migration target pages
  - Starts at the end of zone, moves towards beginning
  - Isolates free pages from buddy allocator (splits as needed)
- Stops when scanner positions cross each other
  - Or, when free page of requested order has been created
  - Or due to lock contention, exhausted timeslice, fatal signal...



Memory Compaction Limitations

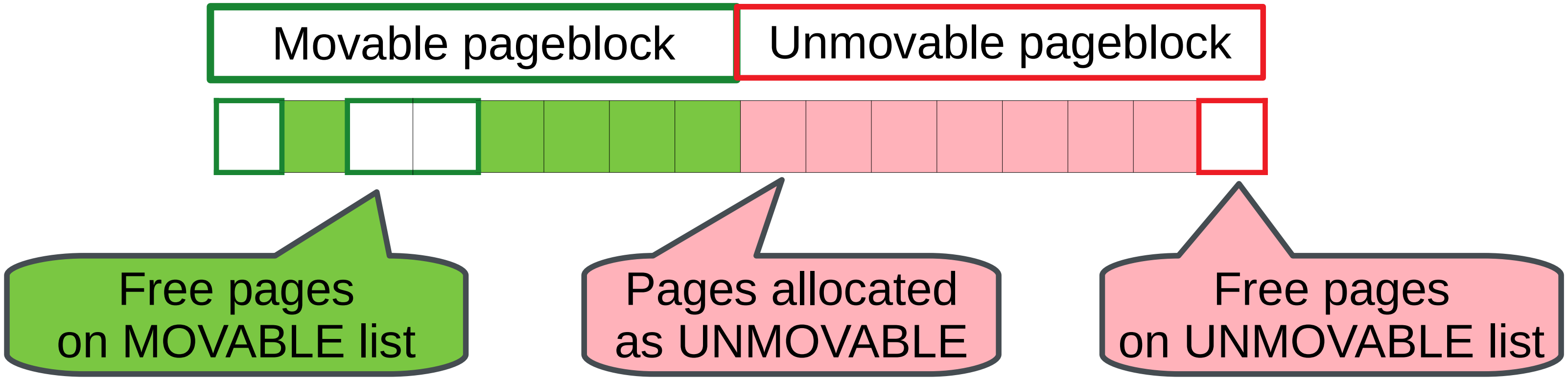
- Only a subset of pages can be isolated and migrated
  - Pages on LRU lists (user-space mapped, either anonymous or page cache)
  - Pages marked with PageMovable “flag”
    - Currently just zsmalloc (used by zram and zswap) and virtio balloon pages
  - No other page references (pins) except from mappings, only clean pages on some filesystems…
- A single non-migratable page in an order-9 block can prevent allocating a whole huge page there, resulting in permanent fragmentation
- Solution: keep such pages close together
  - *Page grouping by mobility*

Grouping by Mobility Overview

- Zones divided to pageblocks (order-9 = 2MB on x86)
  - Each marked as MOVABLE, UNMOVABLE or RECLAIMABLE *migratetype* (there are few more for other purposes)
- Separate buddy free lists for each migratetype
- Allocations declare (via GFP flags) intended type
  - Tries to be satisfied first from matching pageblock type
  - Fallback to other type when matching pageblocks full

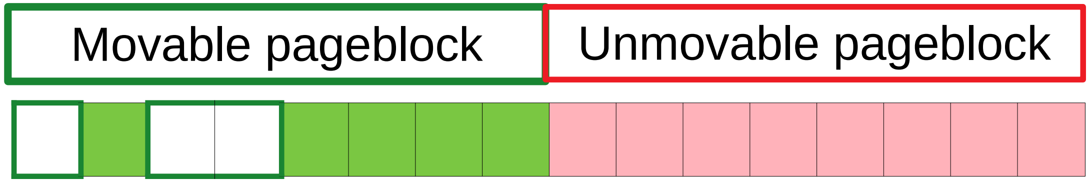
Grouping by Mobility Overview

- Zones divided to pageblocks (order-9 = 2MB on x86)
  - Each marked as MOVABLE, UNMOVABLE or RECLAIMABLE *migratetype* (there are few more for other purposes)
- Separate buddy free lists for each migratetype
- Allocations declare (via GFP flags) intended type
  - Tries to be satisfied first from matching pageblock type
  - Fallback to other type when matching pageblocks full



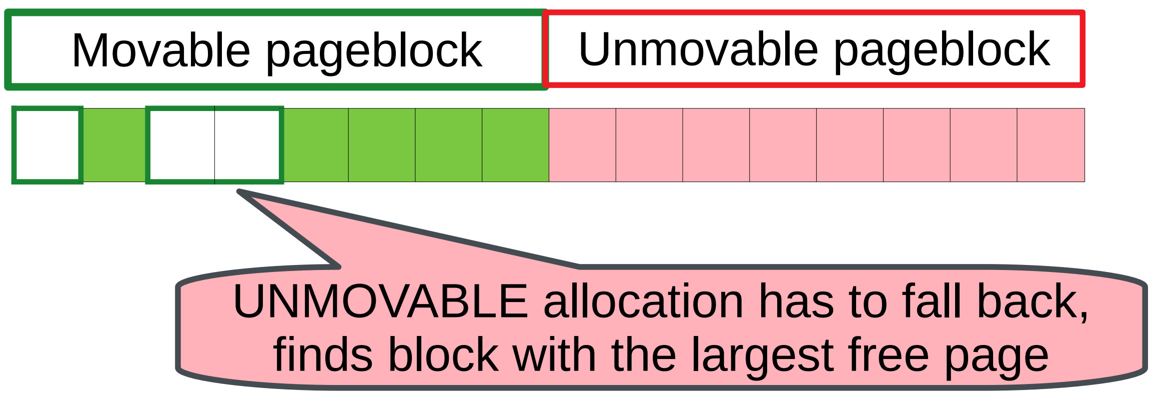
Grouping by Mobility Overview

- Zones divided to pageblocks (order-9 = 2MB on x86)
  - Each marked as MOVABLE, UNMOVABLE or RECLAIMABLE *migratetype* (there are few more for other purposes)
- Separate buddy free lists for each migratetype
- Allocations declare (via GFP flags) intended type
  - Tries to be satisfied first from matching pageblock type
  - Fallback to other type when matching pageblocks full



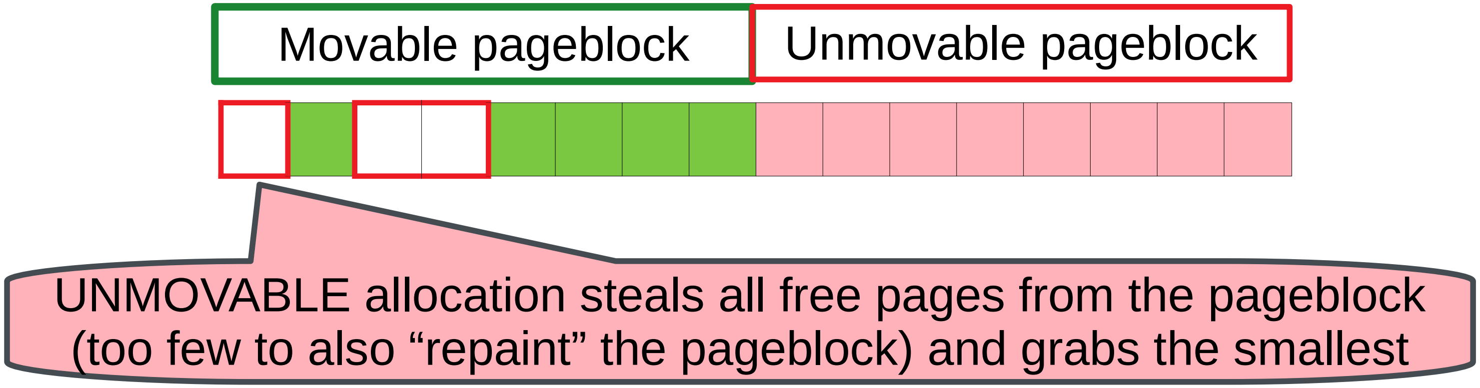
Grouping by Mobility Overview

- Zones divided to pageblocks (order-9 = 2MB on x86)
  - Each marked as MOVABLE, UNMOVABLE or RECLAIMABLE *migratetype* (there are few more for other purposes)
- Separate buddy free lists for each migratetype
- Allocations declare (via GFP flags) intended type
  - Tries to be satisfied first from matching pageblock type
  - Fallback to other type when matching pageblocks full



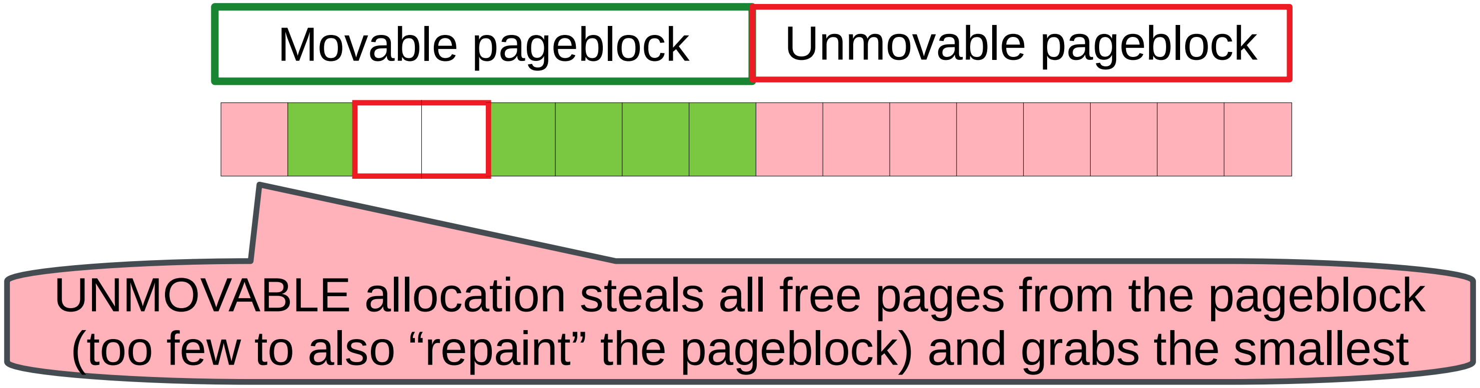
Grouping by Mobility Overview

- Zones divided to pageblocks (order-9 = 2MB on x86)
  - Each marked as MOVABLE, UNMOVABLE or RECLAIMABLE *migratetype* (there are few more for other purposes)
- Separate buddy free lists for each migratetype
- Allocations declare (via GFP flags) intended type
  - Tries to be satisfied first from matching pageblock type
  - Fallback to other type when matching pageblocks full



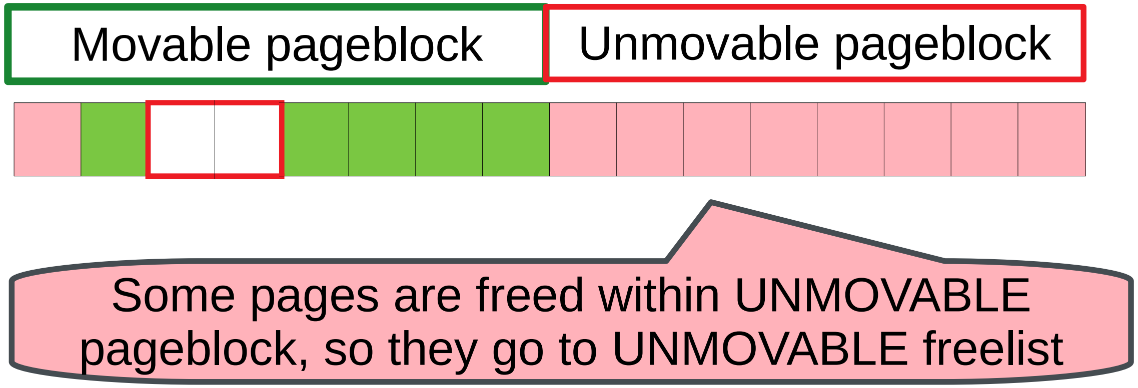
Grouping by Mobility Overview

- Zones divided to pageblocks (order-9 = 2MB on x86)
  - Each marked as MOVABLE, UNMOVABLE or RECLAIMABLE *migratetype* (there are few more for other purposes)
- Separate buddy free lists for each migratetype
- Allocations declare (via GFP flags) intended type
  - Tries to be satisfied first from matching pageblock type
  - Fallback to other type when matching pageblocks full



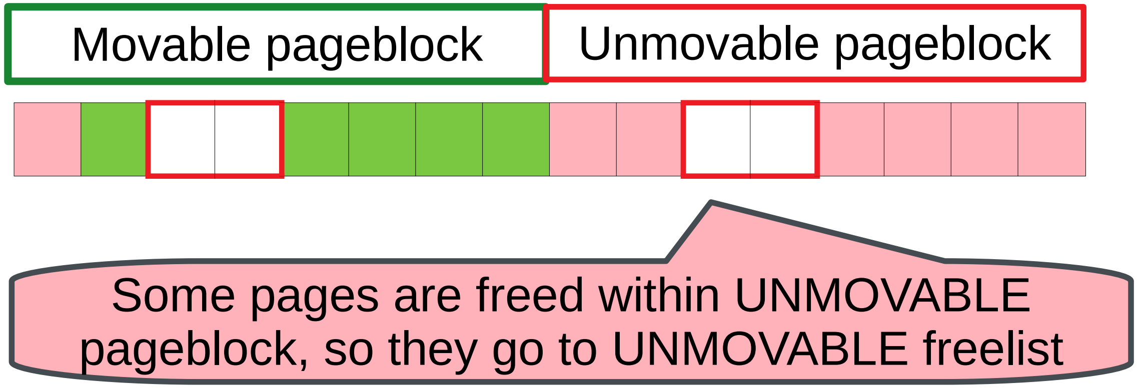
Grouping by Mobility Overview

- Zones divided to pageblocks (order-9 = 2MB on x86)
  - Each marked as MOVABLE, UNMOVABLE or RECLAIMABLE *migratetype* (there are few more for other purposes)
- Separate buddy free lists for each migratetype
- Allocations declare (via GFP flags) intended type
  - Tries to be satisfied first from matching pageblock type
  - Fallback to other type when matching pageblocks full



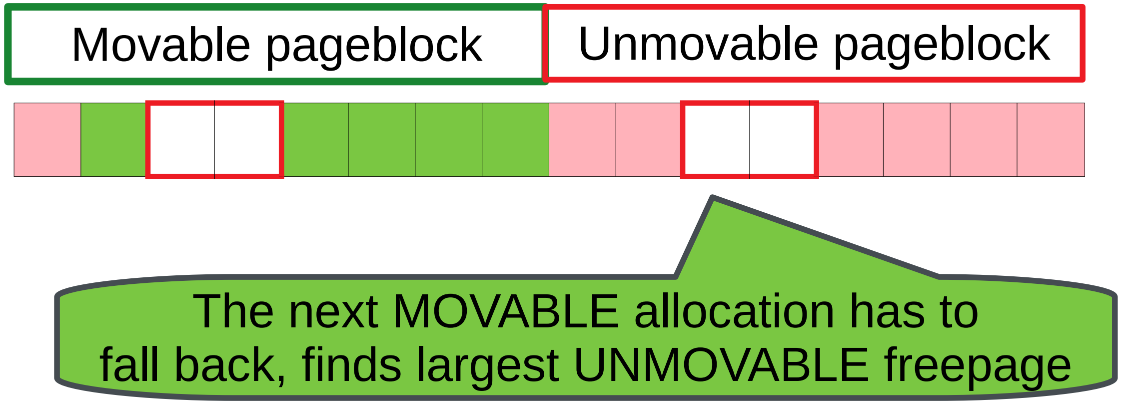
Grouping by Mobility Overview

- Zones divided to pageblocks (order-9 = 2MB on x86)
  - Each marked as MOVABLE, UNMOVABLE or RECLAIMABLE *migratetype* (there are few more for other purposes)
- Separate buddy free lists for each migratetype
- Allocations declare (via GFP flags) intended type
  - Tries to be satisfied first from matching pageblock type
  - Fallback to other type when matching pageblocks full



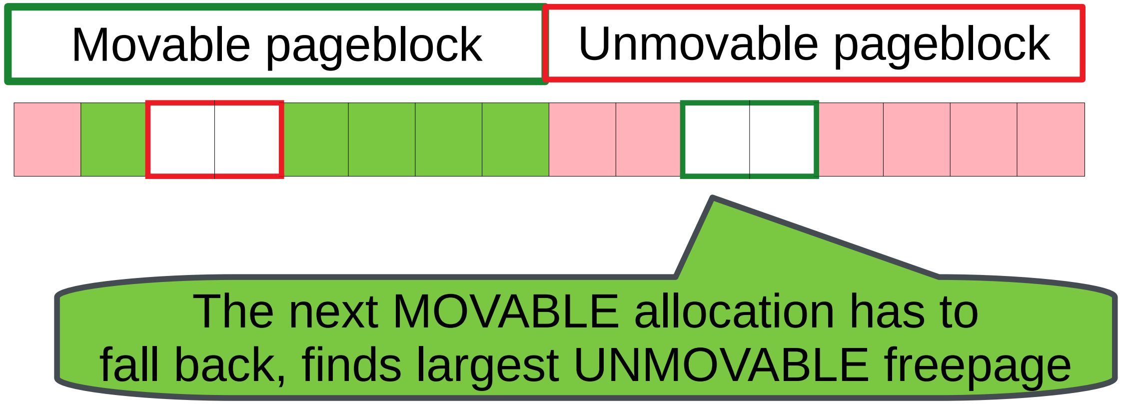
Grouping by Mobility Overview

- Zones divided to pageblocks (order-9 = 2MB on x86)
  - Each marked as MOVABLE, UNMOVABLE or RECLAIMABLE *migratetype* (there are few more for other purposes)
- Separate buddy free lists for each migratetype
- Allocations declare (via GFP flags) intended type
  - Tries to be satisfied first from matching pageblock type
  - Fallback to other type when matching pageblocks full



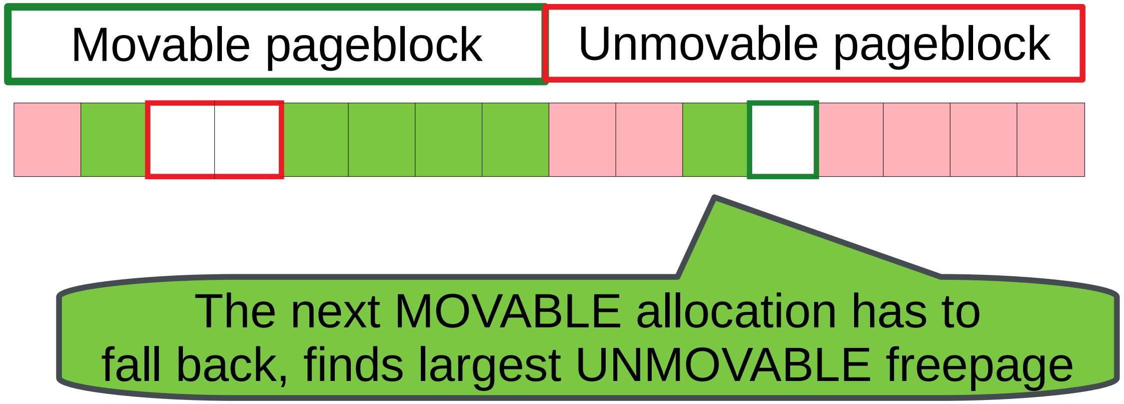
Grouping by Mobility Overview

- Zones divided to pageblocks (order-9 = 2MB on x86)
  - Each marked as MOVABLE, UNMOVABLE or RECLAIMABLE *migratetype* (there are few more for other purposes)
- Separate buddy free lists for each migratetype
- Allocations declare (via GFP flags) intended type
  - Tries to be satisfied first from matching pageblock type
  - Fallback to other type when matching pageblocks full



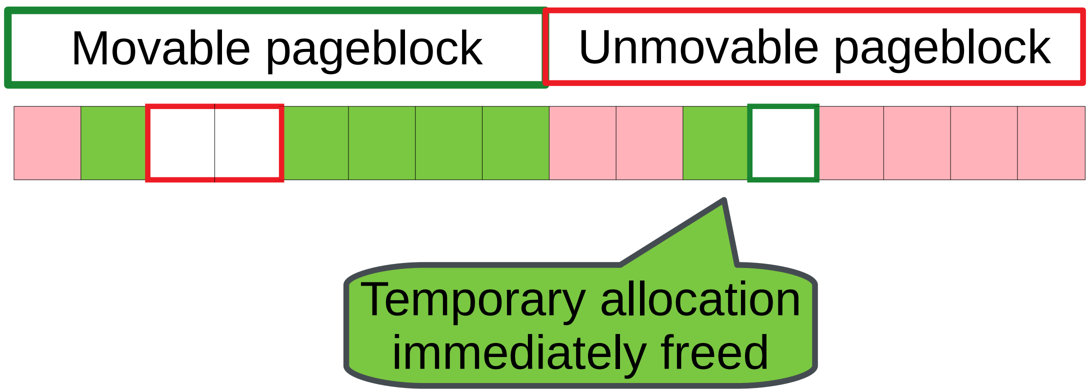
Grouping by Mobility Overview

- Zones divided to pageblocks (order-9 = 2MB on x86)
  - Each marked as MOVABLE, UNMOVABLE or RECLAIMABLE *migratetype* (there are few more for other purposes)
- Separate buddy free lists for each migratetype
- Allocations declare (via GFP flags) intended type
  - Tries to be satisfied first from matching pageblock type
  - Fallback to other type when matching pageblocks full



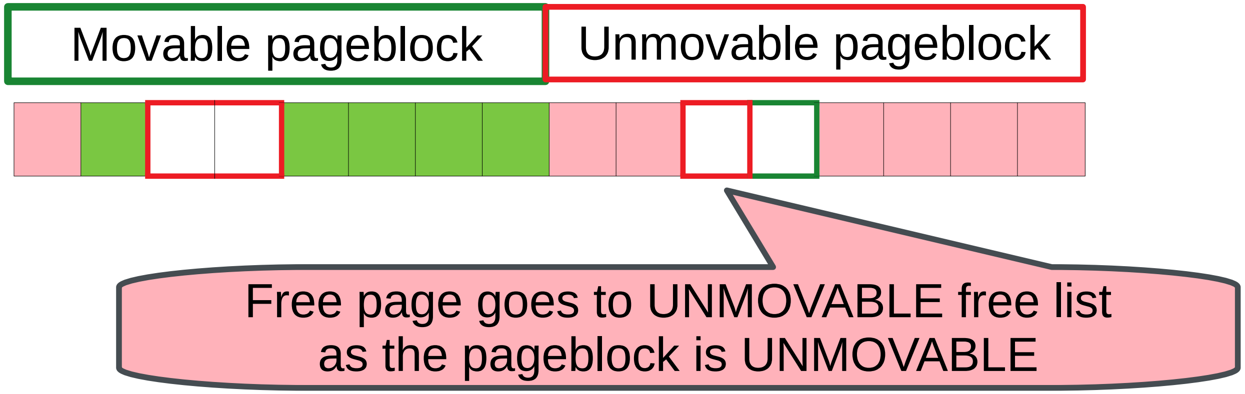
Grouping by Mobility Overview

- Zones divided to pageblocks (order-9 = 2MB on x86)
  - Each marked as MOVABLE, UNMOVABLE or RECLAIMABLE *migratetype* (there are few more for other purposes)
- Separate buddy free lists for each migratetype
- Allocations declare (via GFP flags) intended type
  - Tries to be satisfied first from matching pageblock type
  - Fallback to other type when matching pageblocks full



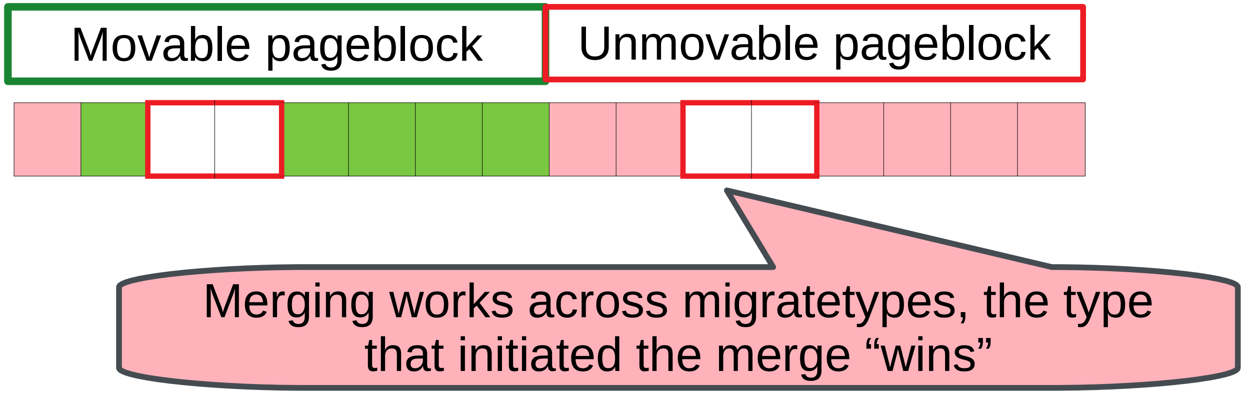
Grouping by Mobility Overview

- Zones divided to pageblocks (order-9 = 2MB on x86)
  - Each marked as MOVABLE, UNMOVABLE or RECLAIMABLE *migratetype* (there are few more for other purposes)
- Separate buddy free lists for each migratetype
- Allocations declare (via GFP flags) intended type
  - Tries to be satisfied first from matching pageblock type
  - Fallback to other type when matching pageblocks full



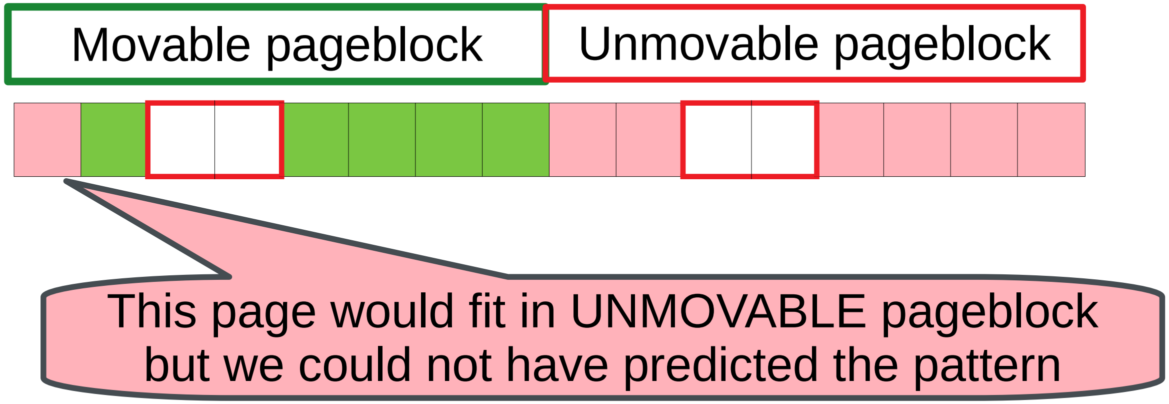
Grouping by Mobility Overview

- Zones divided to pageblocks (order-9 = 2MB on x86)
  - Each marked as MOVABLE, UNMOVABLE or RECLAIMABLE *migratetype* (there are few more for other purposes)
- Separate buddy free lists for each migratetype
- Allocations declare (via GFP flags) intended type
  - Tries to be satisfied first from matching pageblock type
  - Fallback to other type when matching pageblocks full



Grouping by Mobility Overview

- Zones divided to pageblocks (order-9 = 2MB on x86)
  - Each marked as MOVABLE, UNMOVABLE or RECLAIMABLE *migratetype* (there are few more for other purposes)
- Separate buddy free lists for each migratetype
- Allocations declare (via GFP flags) intended type
  - Tries to be satisfied first from matching pageblock type
  - Fallback to other type when matching pageblocks full



Mobility Grouping Fallback Heuristics

- Perfection generally impossible without knowing future
  - Also the effort has to be reasonable wrt allocation latency
- Find+steal the largest free page of other migratetype
  - Approximates finding a pageblock with the most free pages
  - Each migratetype has fallback types ordered by preference
- Can we steal all free pages from the pageblock?
  - UNMOVABLE and RECLAIMABLE allocations always can.
  - MOVABLE: the initially found page has to be order >= 4
- Steal X free pages, count Y pages of compatible type
  - If $X + Y \geq 256$ (half of pageblock), change pageblock type
- Allocate one of the stolen pages, splitting the smallest

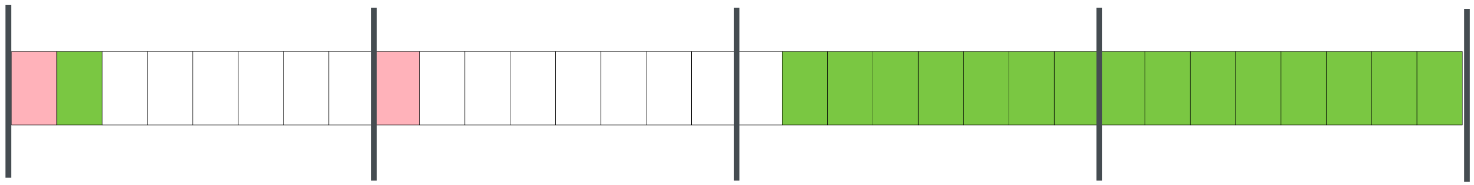
Open Issues of Compaction and Mobility Grouping

Open Issues: Compaction overhead

- Direct compaction to satisfy a high-order allocation increases its latency
- Sometimes reported to be unacceptable
  - Especially for THP page faults, some users disable THP
    - Defaults have changed not to reclaim+compact directly for THP faults
- Defer more work to kcompactd?
  - Woken up after kswapd reclaims up to high watermark
  - Currently makes just one page or highest requested order available
  - Count all requests since last wakeup?
  - Extreme: all pages freed by kswapd consolidated to form free pageblocks

Open Issues: Insufficient Scanning

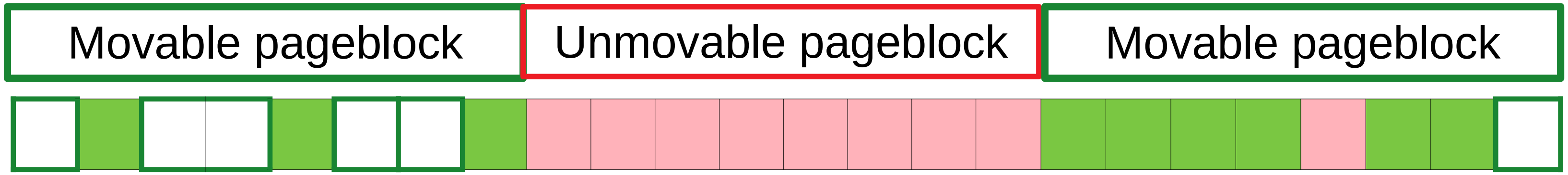
- When memory full, only first half of zone is scanned
  - But no success there due to scattered unmovable pages
  - Second half full, scanners meet roughly in the middle



- Compaction cannot help in this case, what to do?
  - Change starting points from beginning/end of zone?
  - Move both scanners in the same direction?
  - Replace free scanner with direct allocation from freelist?
    - Free scanner can scan 30x pages compared to migration scanner
- Danger of migrating the same pages back and forth
  - Or several parallel compactions undoing each other’s work

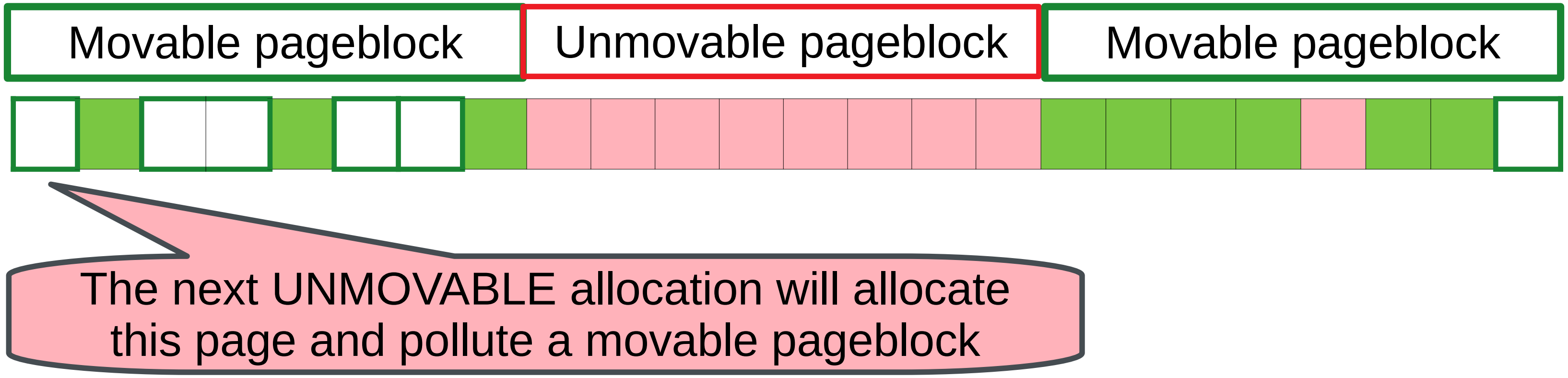
Open Issues: Mobility Grouping

- Problem: unmovable allocation falling back to movable pageblock when memory is nearly full
  - It might pollute another “pure” pageblock containing only movable or free pages, instead of an already polluted one



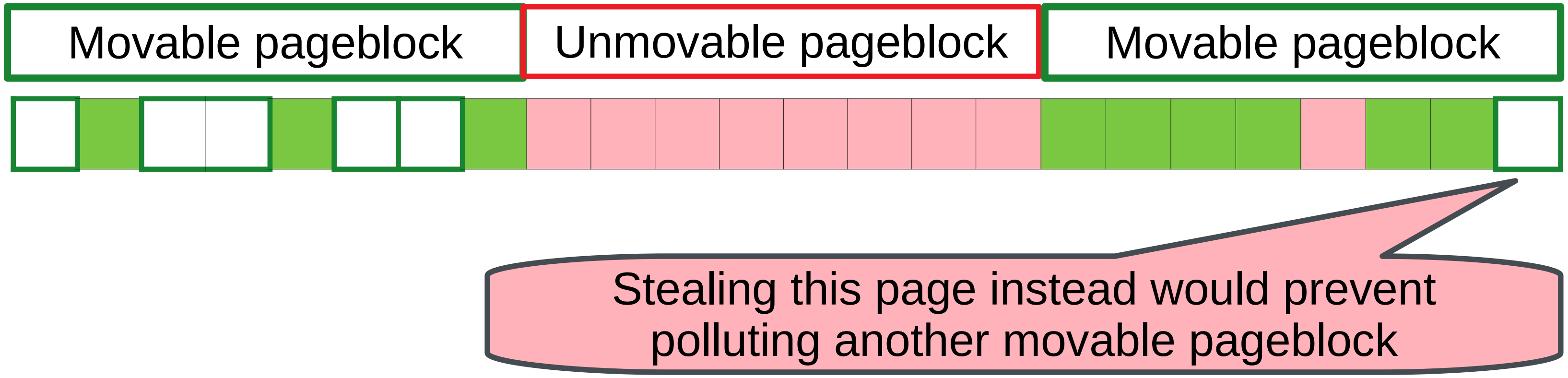
Open Issues: Mobility Grouping

- Problem: unmovable allocation falling back to movable pageblock when memory is nearly full
  - It might pollute another “pure” pageblock containing only movable or free pages, instead of an already polluted one

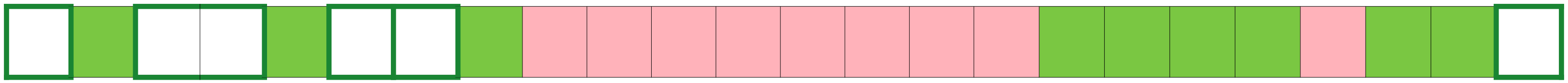


Open Issues: Mobility Grouping

- Problem: unmovable allocation falling back to movable pageblock when memory is nearly full
  - It might pollute another “pure” pageblock containing only movable or free pages, instead of an already polluted one



Open Issues: Mobility Grouping



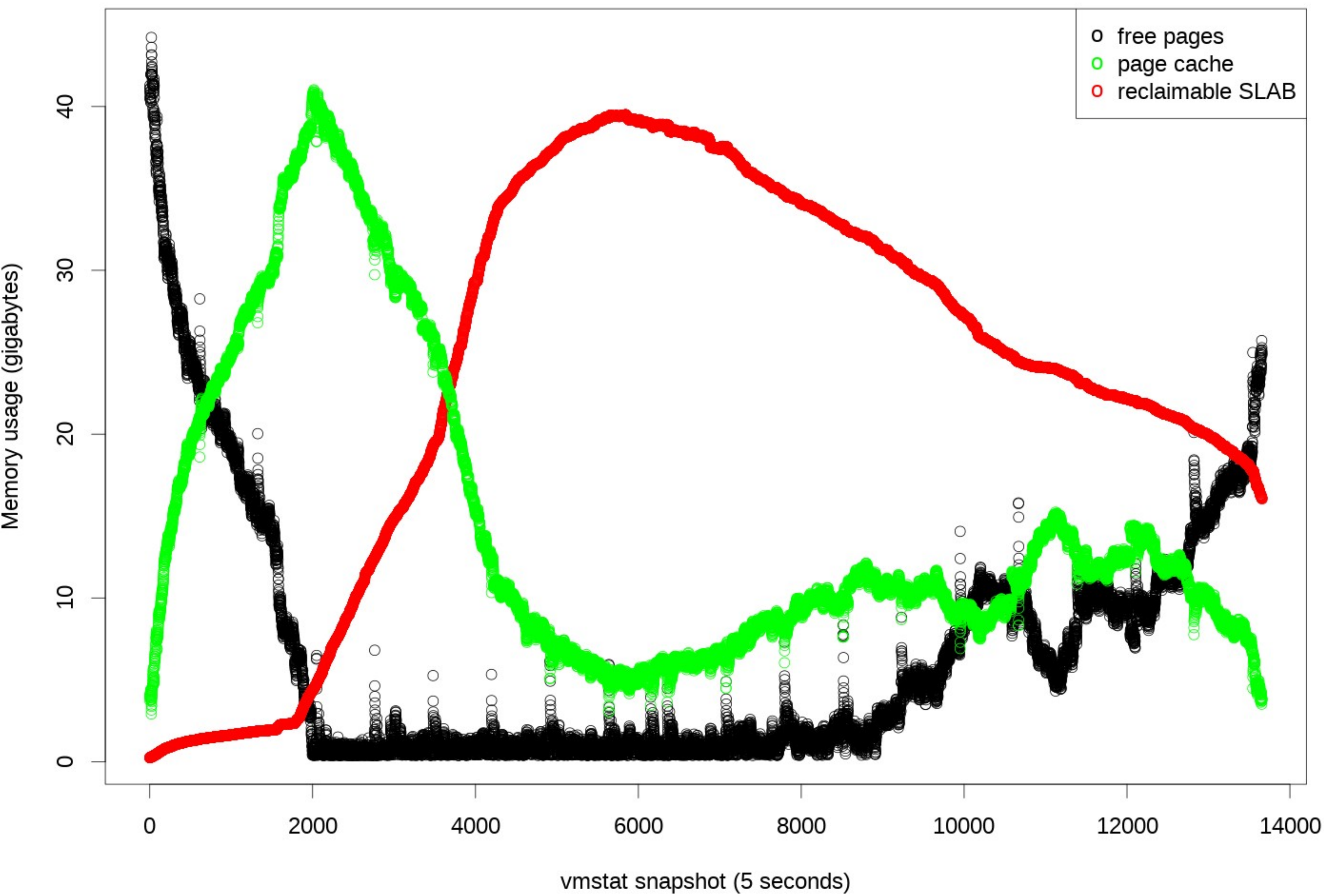
- Migrate movable pages away from the fallback pageblock to accommodate more unmovable pages?
  - Compaction may not reach the pageblock soon enough
    - Or not at all, for pageblocks in second half of the zone
  - Solution: targeted pageblock compaction?
    - Proposed several times (e.g. via kcompactd), not finalized
- New migratetype MIGRATE_MIXED to always prefer polluted blocks over clean ones during fallback?
  - RFC patch in Feb 2017; Panwar et al. ASPLOS'18 paper
  - How to recognize pageblocks that are no longer polluted, to convert them back? Possible during compaction scanning.

Open Issues: Mobility Grouping

- In general, it’s desirable to have fewer fallback events
  - Fewer opportunities to pollute MOVABLE pageblock with UNMOVABLE allocation fallback
  - Fewer opportunities to steal pages from UNMOVABLE pageblocks for MOVABLE allocations fallback
    - Fewer free pages in UMOVABLE pageblocks means further fallbacks
- Recent (last week) series from Mel Gorman
  - **Define a test case** – based on fio and THP allocations
    - Mix of page cache (movable) and slab (unmovable) allocations
  - Try a different zone (same NUMA node) first, before fallback
  - Reclaim more memory (via kswapd) when fallback occurs
  - Stall severely fragmenting allocations to let kswapd progress
  - Result: ~95% less fragmenting events; more THP success

Limits of Mobility Grouping

- Some workloads can defeat even perfect grouping
  - Occupy lots of memory with unmovable pages (slab objects)
  - Free them in “random” (or LRU) order
    - All objects (e.g. 21 dentries) in a page need to be reclaimed to free it
    - All 512 pages in pageblock need to be reclaimed to allow THP allocation
- Not just a theoretical concern
  - A user in linux-mm fighting this, and consequences, for months
  - Tracked down to overnight maintenance via find/du filling 40 GB (of 64) with reclaimable slab (dentries, inodes)
    - Slowly being reclaimed afterwards, but high fragmentation remains
    - Excessive reclaim of page cache as a (non-regular) consequence, not yet clear why, suspected corner case in reclaim/compaction interaction
    - Explicit echo 2 > /proc/sys/vm/drop_caches “fixes” the issue



free pages
page cache
reclaimable SLAB
Memory usage (gigabytes)
0
10
20
30
40
0
2000
4000
6000
8000
10000
12000
14000
vmstat snapshot (5 seconds)

Possible Solutions?

- Make more classes of pages movable
  - Candidates: vmalloc pages, page tables, where concurrent access could be trapped and delayed to allow their migration
- Make certain slab objects movable?
  - Very complex, needs tracking all pointers to the objects
  - RFC posted in Dec 2017 for XArray (by Christopher Lameter)
- Targeted reclaim of slab objects?
  - Easier, but same cons as lumpy reclaim of page cache
- Tweak reclaim speed / prevent unchecked growth of slab caches
  - Some recent efforts for negative dentries (Waiman Long)
  - Might help in this particular case, but not in general?

Questions?

Thank you.