Linux Plumbers Conference

Vienna, Austria | September 18-20, 2024

Memory Allocation Profiling deployment results and future improvements

Suren Baghdasaryan <surenb@google.com>
Kent Overstreet <kent.overstreet@linux.dev>
Sourav Panda <souravpanda@google.com>
Pasha Tatashin <tatashin@google.com>

LINUX
PLUMBERS
CONFERENCE Vienna, Austria / Sept. 18-20, 2024

Overview

Memory Allocation Profiling

Every allocation is accounted and reported via /proc/allocinfo

Enabled using CONFIG_MEM_ALLOC_PROFILING

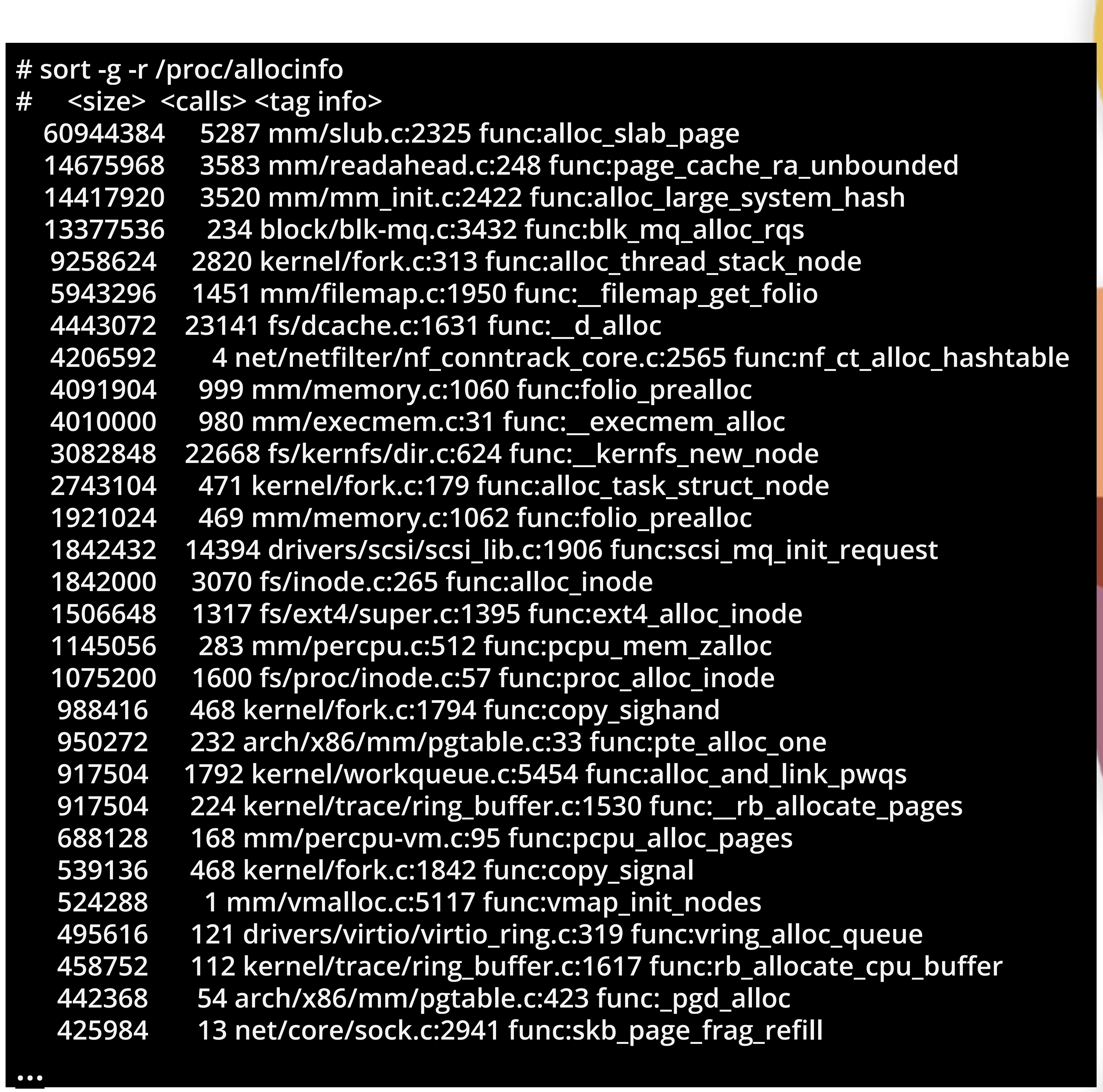
Low overhead to deploy in production

Current state

Merged in 6.10

Deployed at Google as a pilot program (over 1000 servers)

Preparing to be included in the next Android kernel release (based on 6.12)



```
# sort -g -r /proc/allocinfo
#     <size>  <calls> <tag info>
   60944384    5287 mm/slub.c:2325 func:alloc_slab_page
   14675968    3583 mm/readahead.c:248 func:page_cache_ra_unbounded
   14417920    3520 mm/mm_init.c:2422 func:alloc_large_system_hash
   13377536     234 block/blk-mq.c:3432 func:blk_mq_alloc_rqs
    9258624    2820 kernel/fork.c:313 func:alloc_thread_stack_node
    5943296    1451 mm/filemap.c:1950 func:__filemap_get_folio
    4443072   23141 fs/dcache.c:1631 func:__d_alloc
    4206592       4 net/netfilter/nf_conntrack_core.c:2565 func:nf_ct_alloc_hashtable
    4091904     999 mm/memory.c:1060 func:folio_prealloc
    4010000     980 mm/execmem.c:31 func:__execmem_alloc
    3082848   22668 fs/kernfs/dir.c:624 func:__kernfs_new_node
    2743104     471 kernel/fork.c:179 func:alloc_task_struct_node
    1921024     469 mm/memory.c:1062 func:folio_prealloc
    1842432   14394 drivers/scsi/scsi_lib.c:1906 func:scsi_mq_init_request
    1842000    3070 fs/inode.c:265 func:alloc_inode
    1506648    1317 fs/ext4/super.c:1395 func:ext4_alloc_inode
    1145056     283 mm/percpu.c:512 func:pcpu_mem_zalloc
    1075200    1600 fs/proc/inode.c:57 func:proc_alloc_inode
     988416     468 kernel/fork.c:1794 func:copy_sighand
     950272     232 arch/x86/mm/pgtable.c:33 func:pte_alloc_one
     917504    1792 kernel/workqueue.c:5454 func:alloc_and_link_pwqs
     917504     224 kernel/trace/ring_buffer.c:1530 func:__rb_allocate_pages
     688128     168 mm/percpu-vm.c:95 func:pcpu_alloc_pages
     539136     468 kernel/fork.c:1842 func:copy_signal
     524288       1 mm/vmalloc.c:5117 func:vmap_init_nodes
     495616     121 drivers/virtio/virtio_ring.c:319 func:vring_alloc_queue
     458752     112 kernel/trace/ring_buffer.c:1617 func:rb_allocate_cpu_buffer
     442368      54 arch/x86/mm/pgtable.c:423 func:_pgd_alloc
     425984      13 net/core/sock.c:2941 func:skb_page_frag_refill
...
```

Deployment

Goal

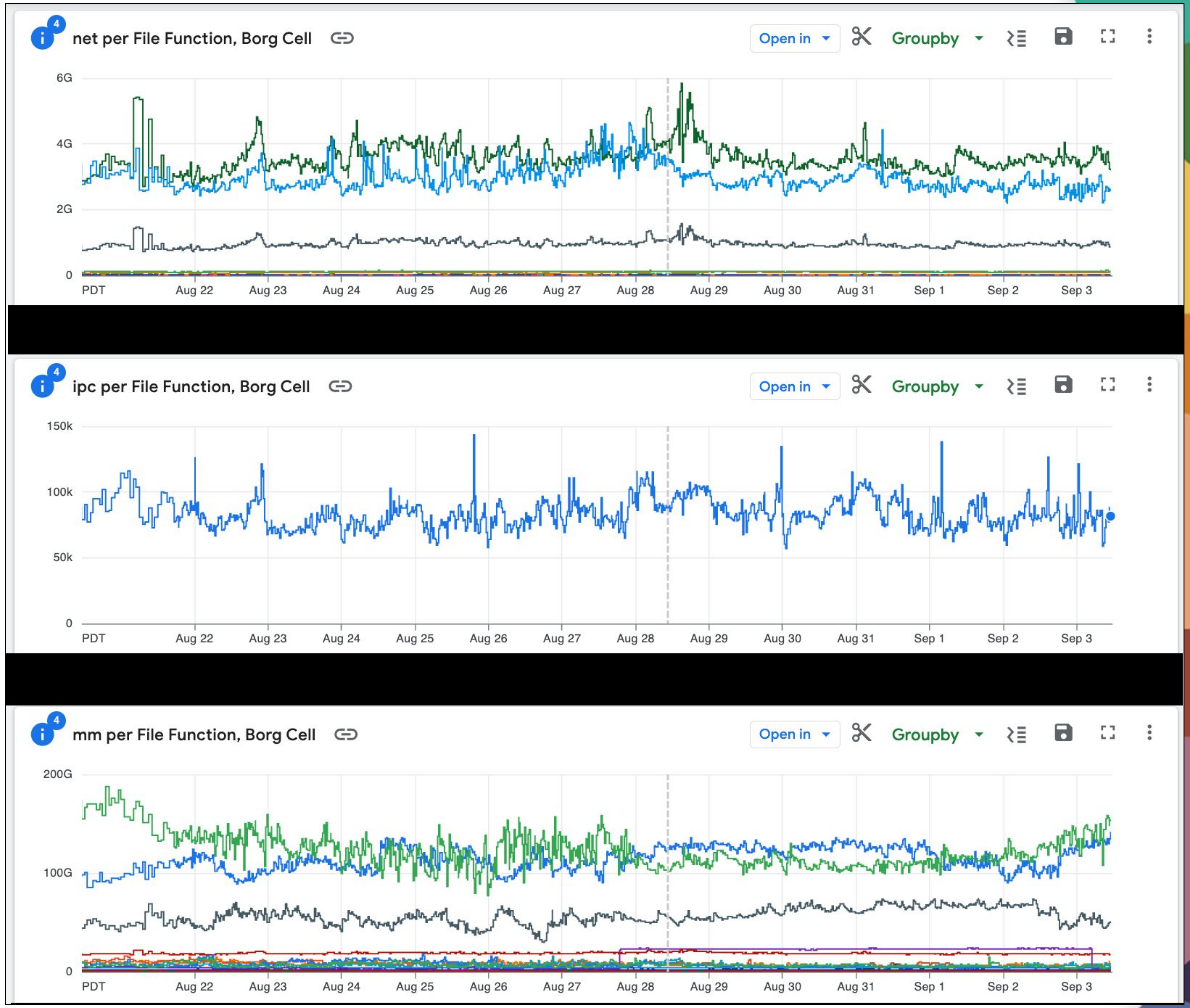
Reducing wasteful memory allocations

Pilot

- Standalone testbed machines
- 1251 production machines

Methodology

- allocinfo gathered and stored into a central DB
- per-subsystems dashboards for visualization and tracking
- analysed using expressive queries over ~0.4 PB of data



Allocations made on a single machine, categorized by subsystem (net, mm, ...)

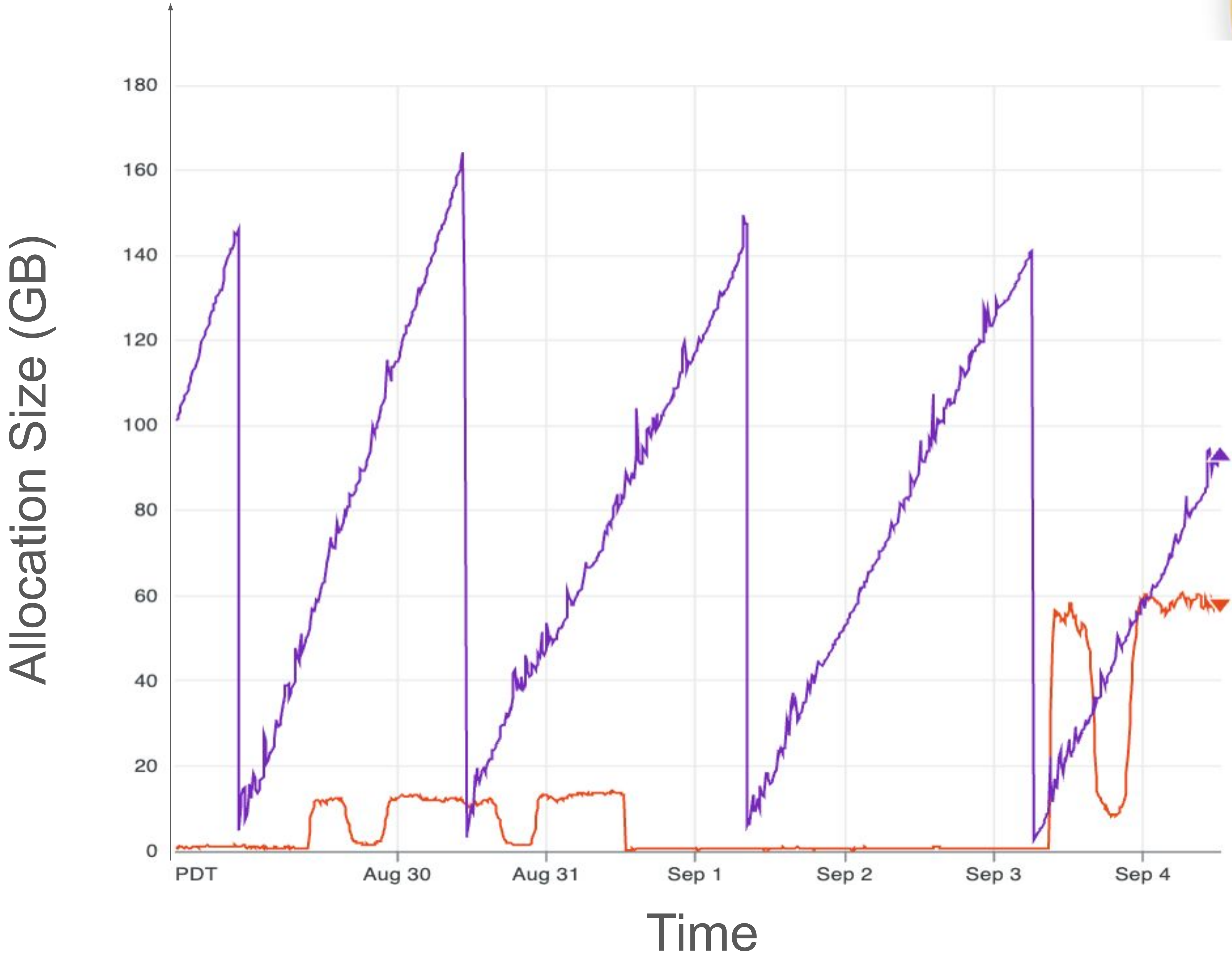
Discoveries: Rx buffer allocations by the networking driver

Identified

- Heaviest allocations
- Per-site changes in allocated memory

Next Step

- Can it be reduced or charged correctly?
- Does it correlate to SLO violations (packet latency or packet drops)?



Rx buffer allocations by the networking driver (top two machines)

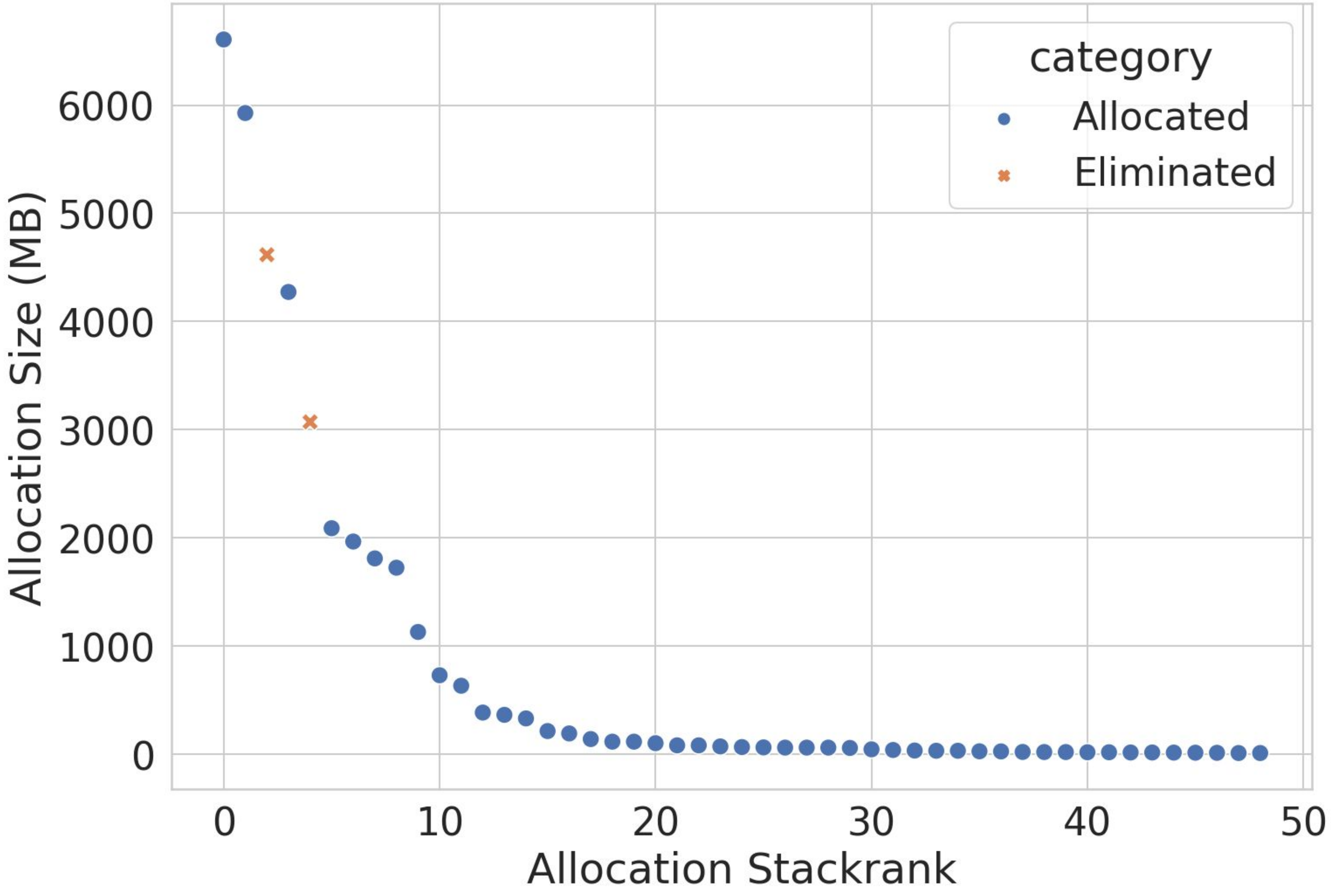
Discoveries: memory overhead on an idle cloud machine

Identified

- Large allocations after boot (exceeding 10 MB)
- Allocations that can be reduced

Results

- Over 7GB of memory savings per machine
- Identified 2GB of more potential savings
- Less kernel overhead, more memory for guest VMs



Post-boot single machine per-site allocations exceeding 10 MB

Needs identified

Overhead

Memory overhead from page extensions is ~2GB per machine (~0.3% of the average total memory)

Pilot machines spend 41% more cycles in __alloc_pages_nodemask than control machines

Lack of context

Who is invoking xas_alloc()?

shmem_add_to_page_cache	:	90% allocations
add_to_swap_cache	:	4% allocations
other	:	6% allocations

Useful for analyzing potential allocation issues

Ongoing and future improvements

- Option to use page flags instead of page extensions is discussed on LKML [1]
- Internal POC for distinguishing GFP_ACCOUNT allocations for more accurate accounting
- Internal POC for collecting average lifespan for every memory allocation.
- Planning to add context capture support showcased in the original RFC [2]
- Working on userspace tooling for allocinfo capture and analyzes

[1] https://lore.kernel.org/all/20240902044128.664075-1-surenb@google.com/
[2] https://lore.kernel.org/all/20230501165450.15352-35-surenb@google.com/

Thank you!