Linux Plumbers Conference

Vienna, Austria | September 18-20, 2024

mTHP and SWAP allocator

Chris Li (Google) and Kairui Song(Tencent)

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

Help needed: SSD internal and write amplification factor

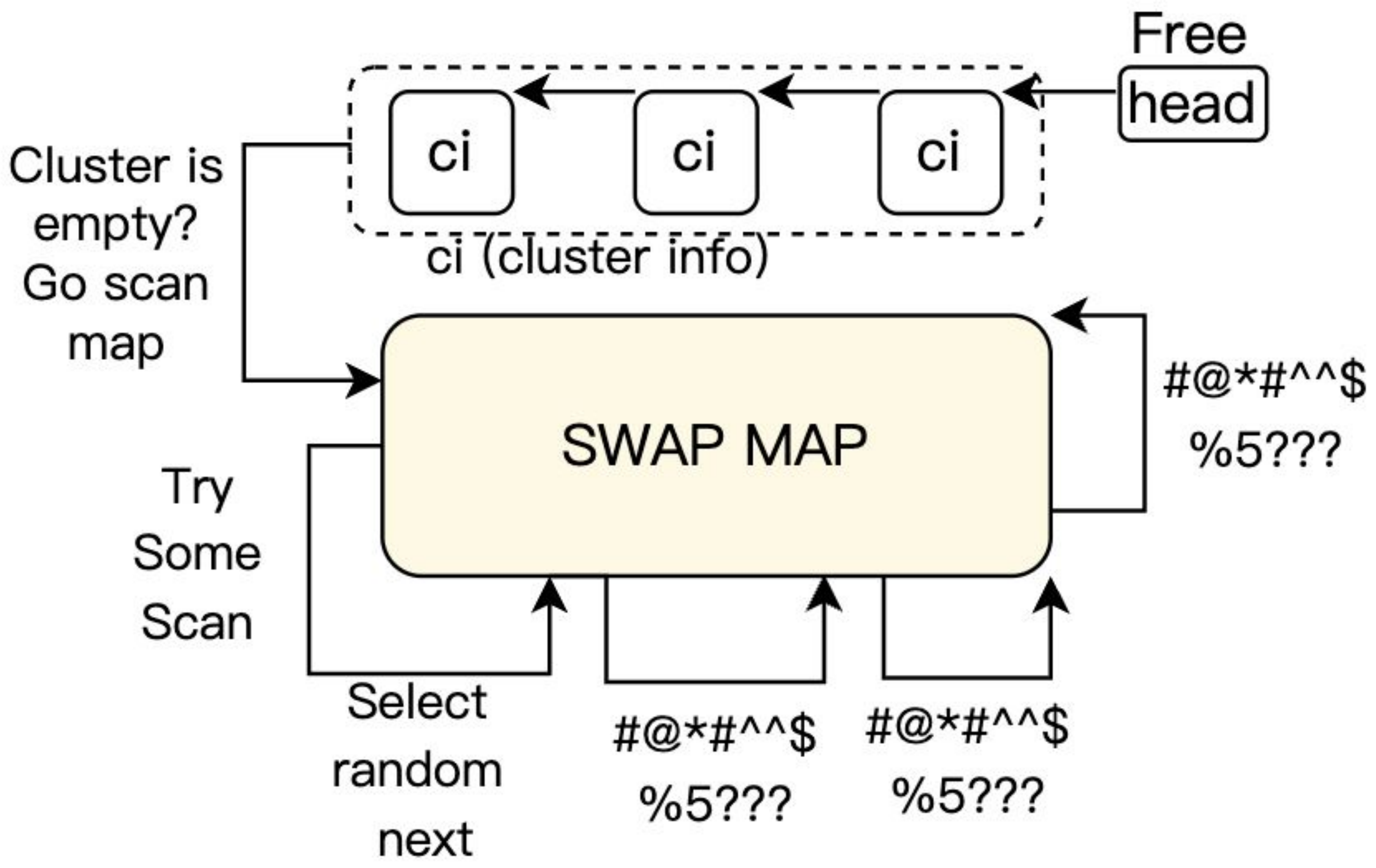
- Current SSD swap allocator might cause high write amplification factor.
- I want to provide options for SSD swap to reduce the write implications.
- It depends on vendor specific SSD algorithms.
- Please reach out to me if you know internal of how SSD garbage collects old erase blocks.

Swap Background

- Swap entries are just like memory pages, are system resources:
  - Has limited resource
  - Subject to page order and fragmentation.
- Swap entry allocation is a bin packing problem.
- New trend and challenge in swap usage:
  - Compression based: ZRAM Android, zswap data center
  - 0 or PMD order -> mTHP more order in between.
  - Nobody care about HDD swap?

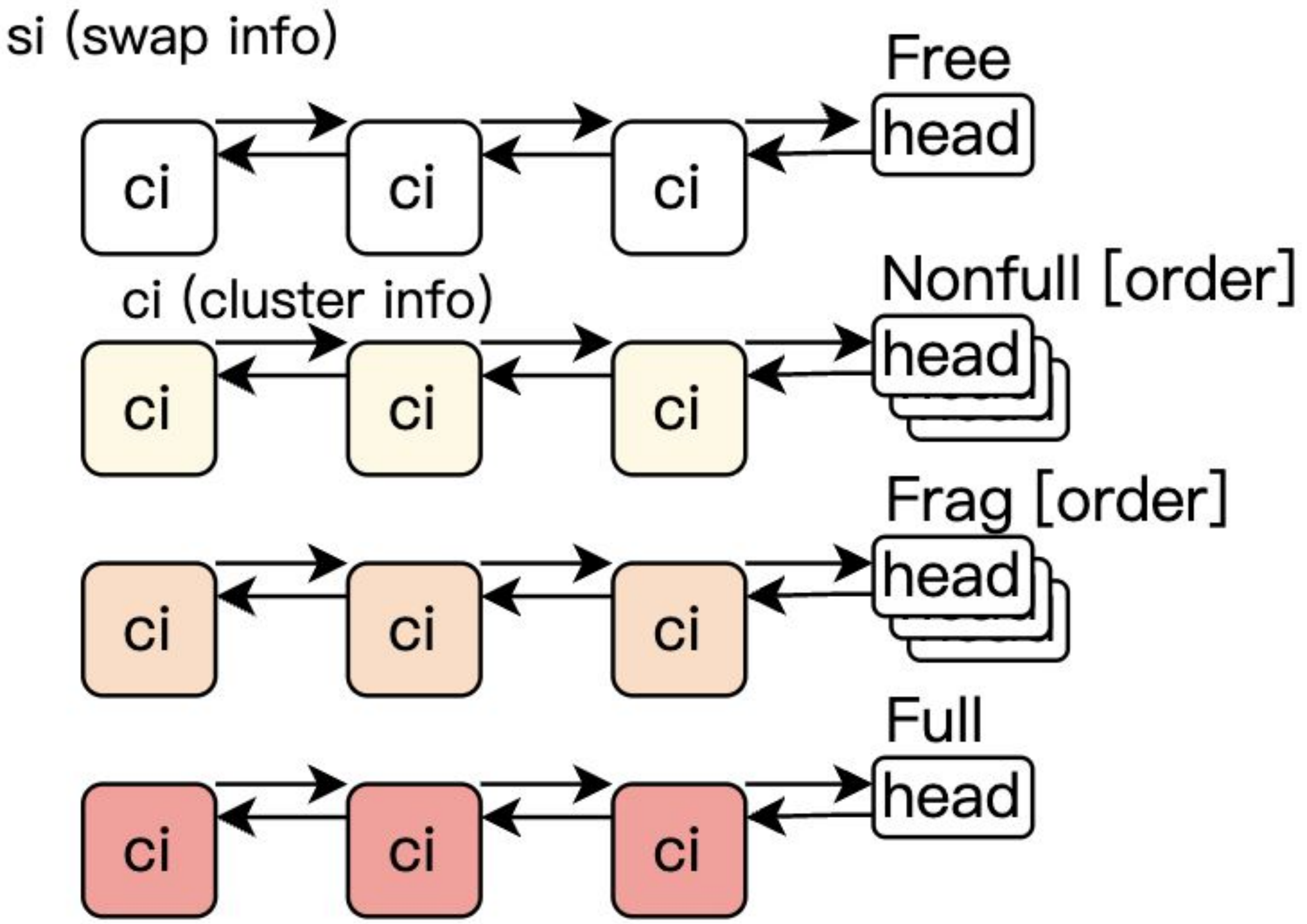
Current Swap Allocator

- scan_swap_map_try_ssd_cluster
  - Limited single link list for empty cluster only
  - PMD size only
  - Only find the cluster, does not allocate from it.
- scan_swap_map_slots()
  - Pre allocated per swap entry swap_map array.
  - Actual allocation
- Complex relationship between cluster allocation and swap_map scan.
  - Try, fallback and retry.
  - Allocation conflict on free cluster list head.
  - Cluster allocation only works if there is empty cluster.
  - Random select cluster position otherwise.
  - Complex execution flow.
- Complex execution flow.
- Order > 0 allocation failure rate is very high after exhaust free cluster list.



New Cluster Order based Swap Allocator

- "Do or do not. There is no try."
- Not longer best effort, cluster allocation is a complete allocator.
  - Switch to double linked list for cluster.
  - Handle not empty cluster.
  - No cluster allocation conflict any more.
  - Find and allocate the swap entry together.
- Cluster are organized by cluster lists.
  - Each order has list, fit mTHP usage case better.
  - Many cluster list.
    - Free cluster list.
    - Nonfull/partial allocated list (per order).
    - fragmented list per order (per order).
    - Full list.
- Get rid of swap_map array scan(). Always ways allocate swap entry from cluster list.
- Easier to find the last few free swap entries.

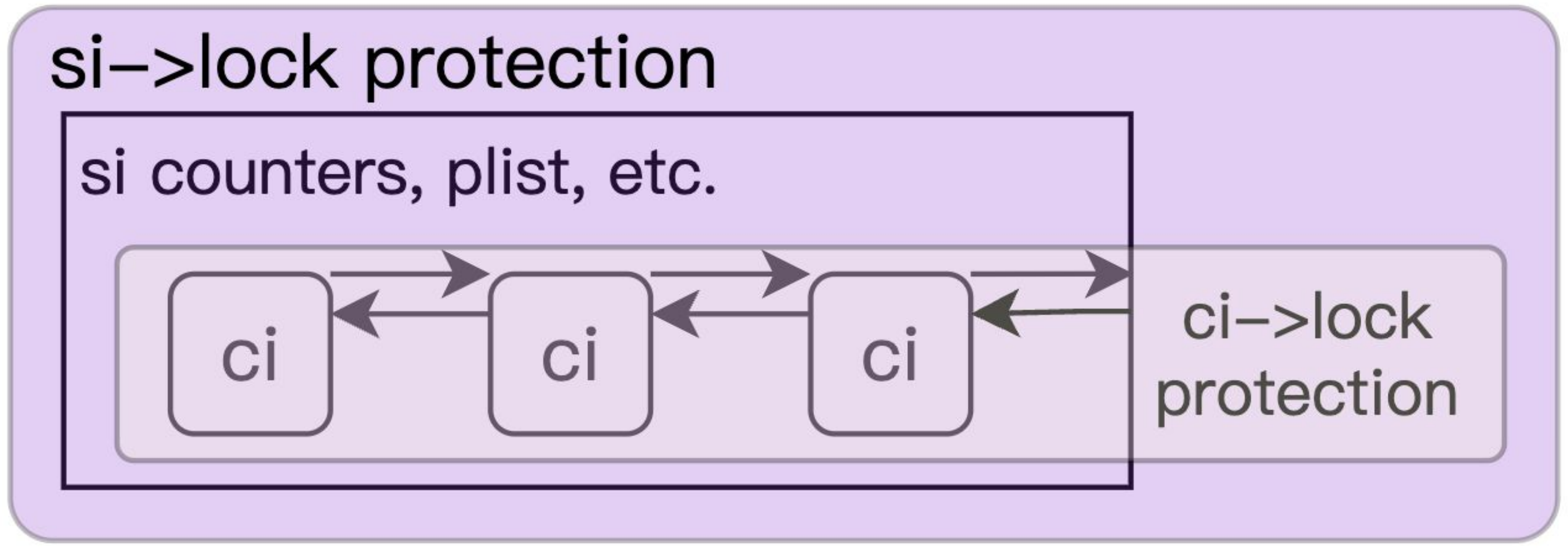


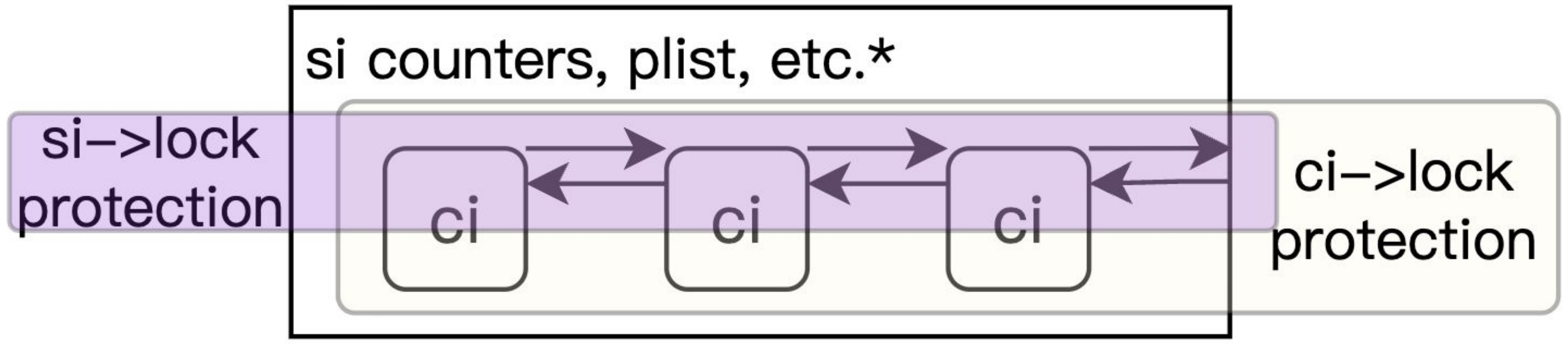
Reducing the swap device lock contention

- There are two locks:
  - si->lock (swap info - device lock, big lock)
  - ci->lock (cluster info - cluster lock, per 2M swap cluster)
- Current swap allocator take these two locks together on allocation or freeing (*mostly):

  First si->lock (per swap device **big lock**, **contention**!).

  Then ci->lock (per 2M cluster).
- Cluster based operation provides the chance and motivation (performance and feasibility) to finally get rid of si->lock contention (as much as possible).
- New swap allocator want to use ci->lock as much as possible:
  - Reduce si->lock critical section, decouple list unrelated data,
  - Reverse the dependency of si and ci lock.
    - **si->lock only protect the cluster lists**.

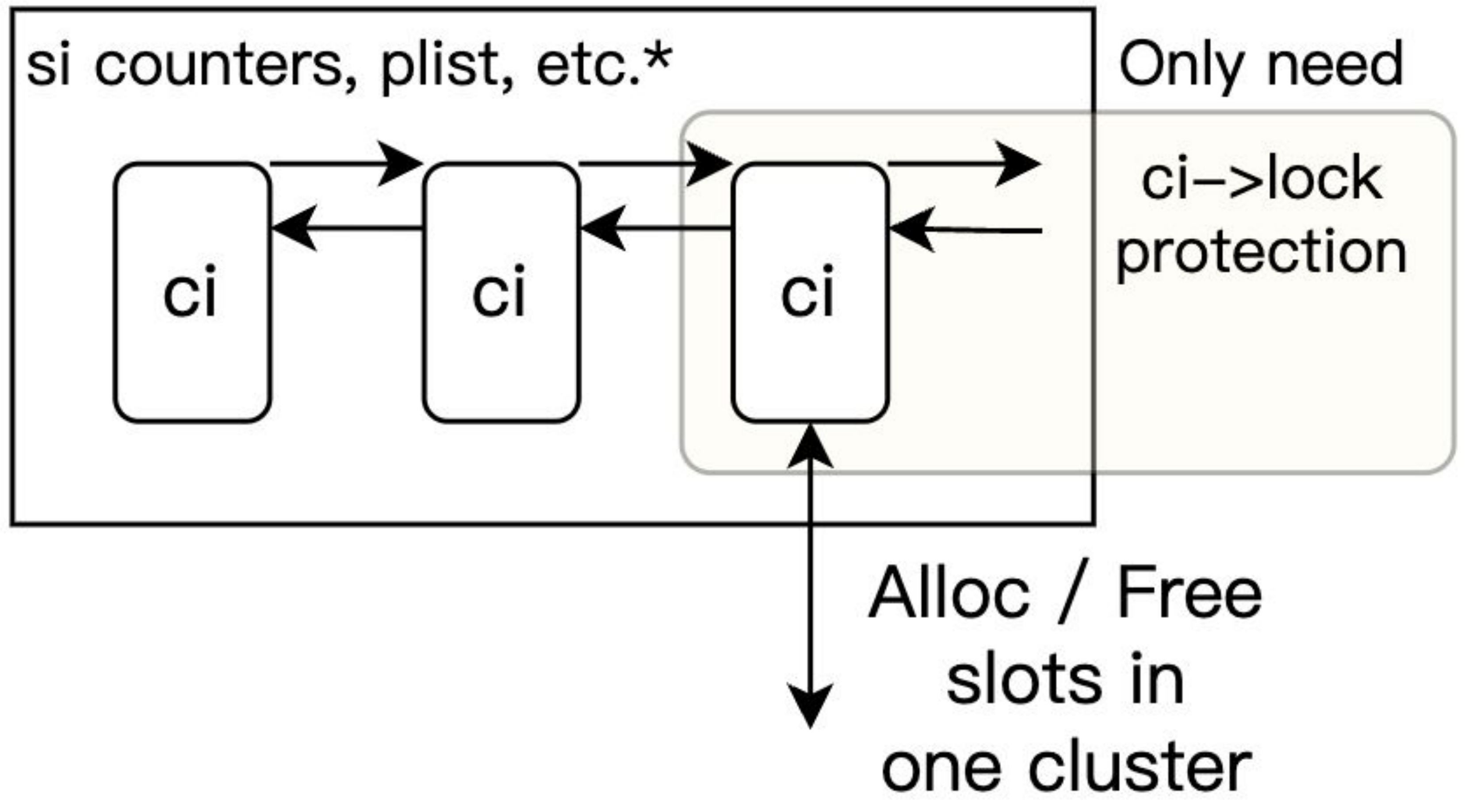


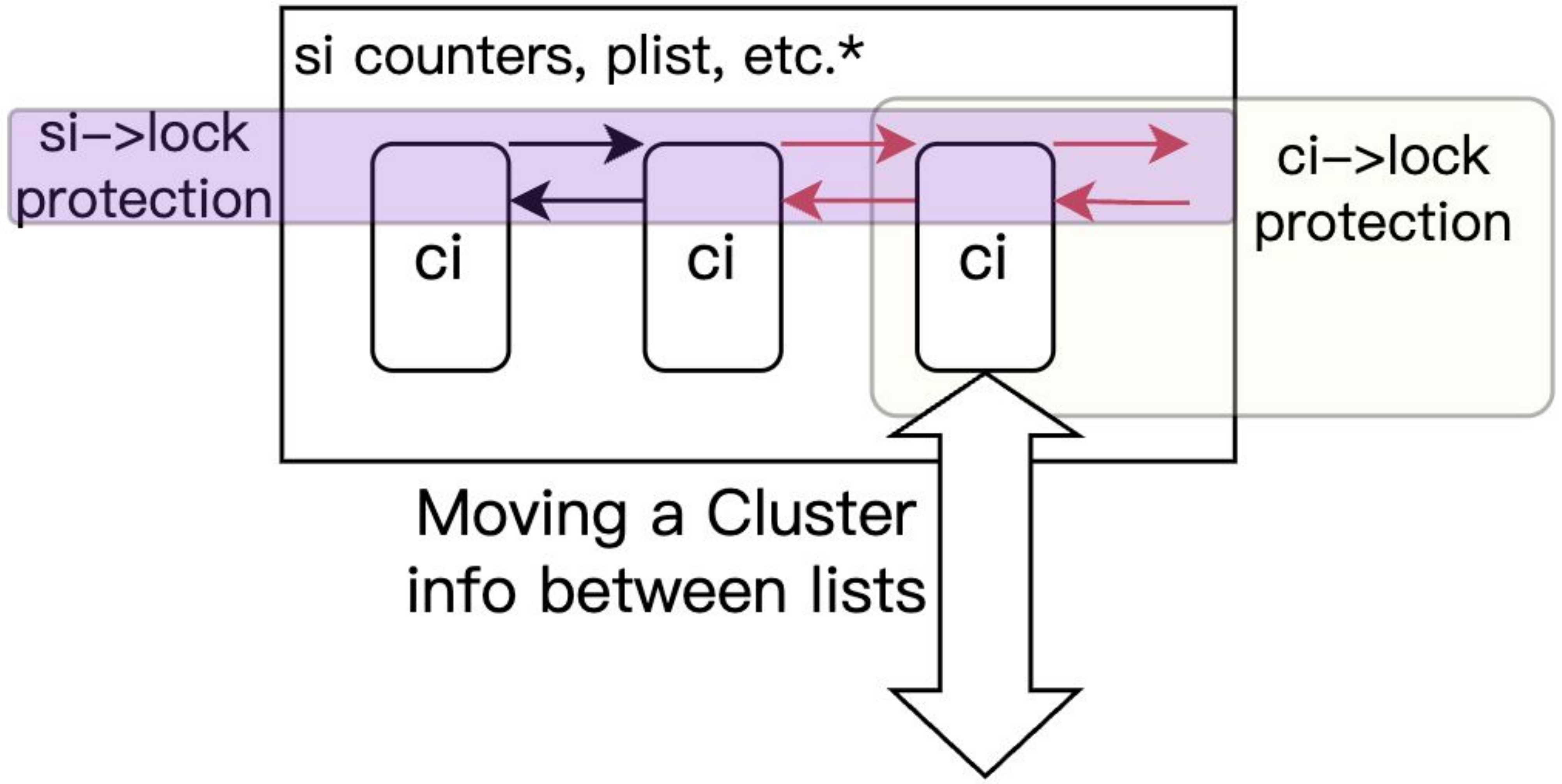


*turn into lockless or use different locks

Reducing the swap device lock contention

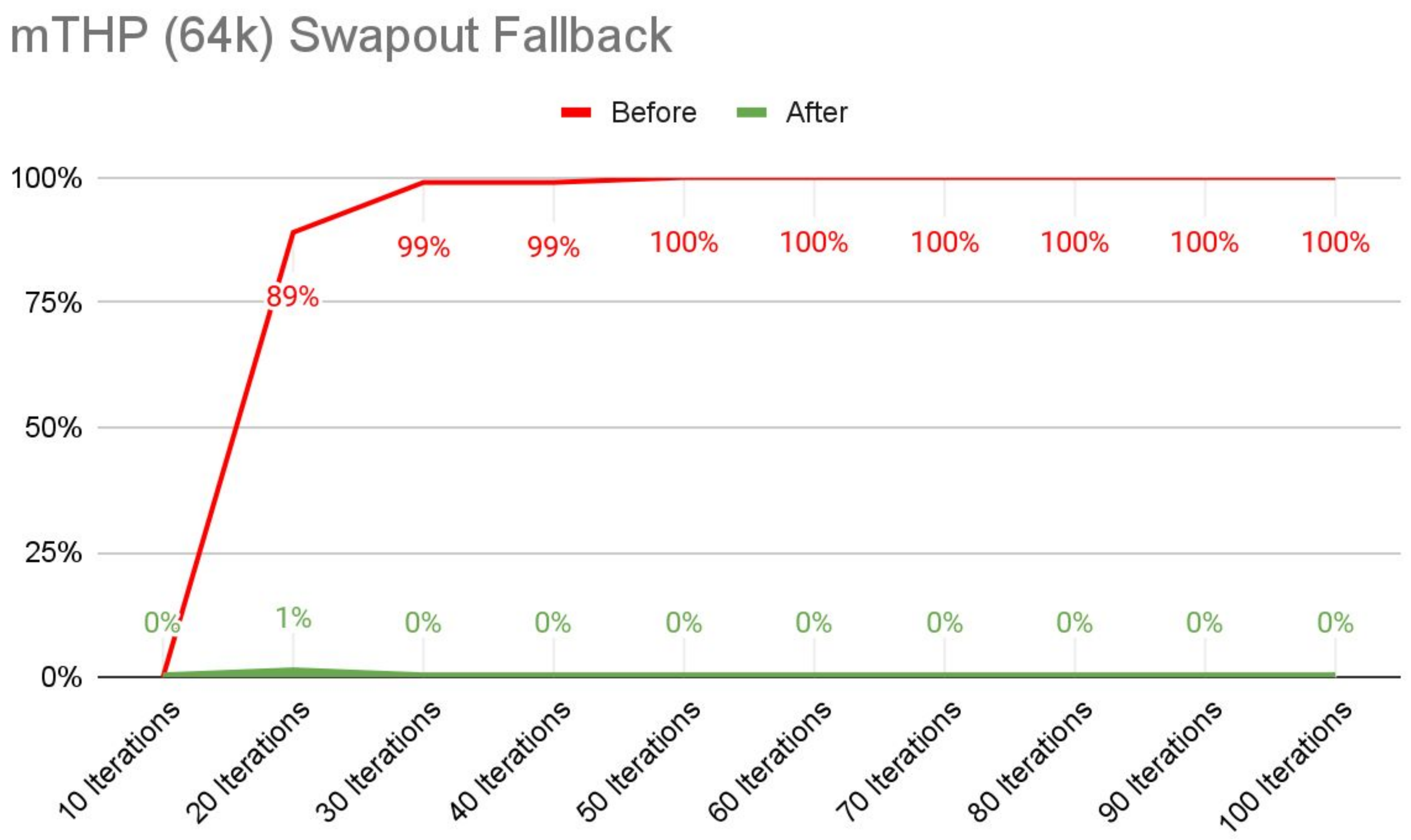
- New allocator only need to take si->lock when touching cluster **lists**, operation inside one cluster only need ci->lock.
  - Operation in one cluster is the common case now.
- On **freeing** swap entry:
  - full list -> nonfull list -> free list
  - Most of the time stay in the nonfull list (512 entries), no list movement needed.
  - Can avoid si->lock on most freeing operations.
  - Get rid of swap slot cache on the free path? Yes.
- On **allocating** swap entry**:**
  - Just keep using same cluster as much as possible.
    - Which is already true.
  - Up to 512 entries serve as a local cache
  - Don’t need to touch the si->lock as long as the cluster is not drained.
  - Get rid of swap slot cache on the allocation path? Maybe…





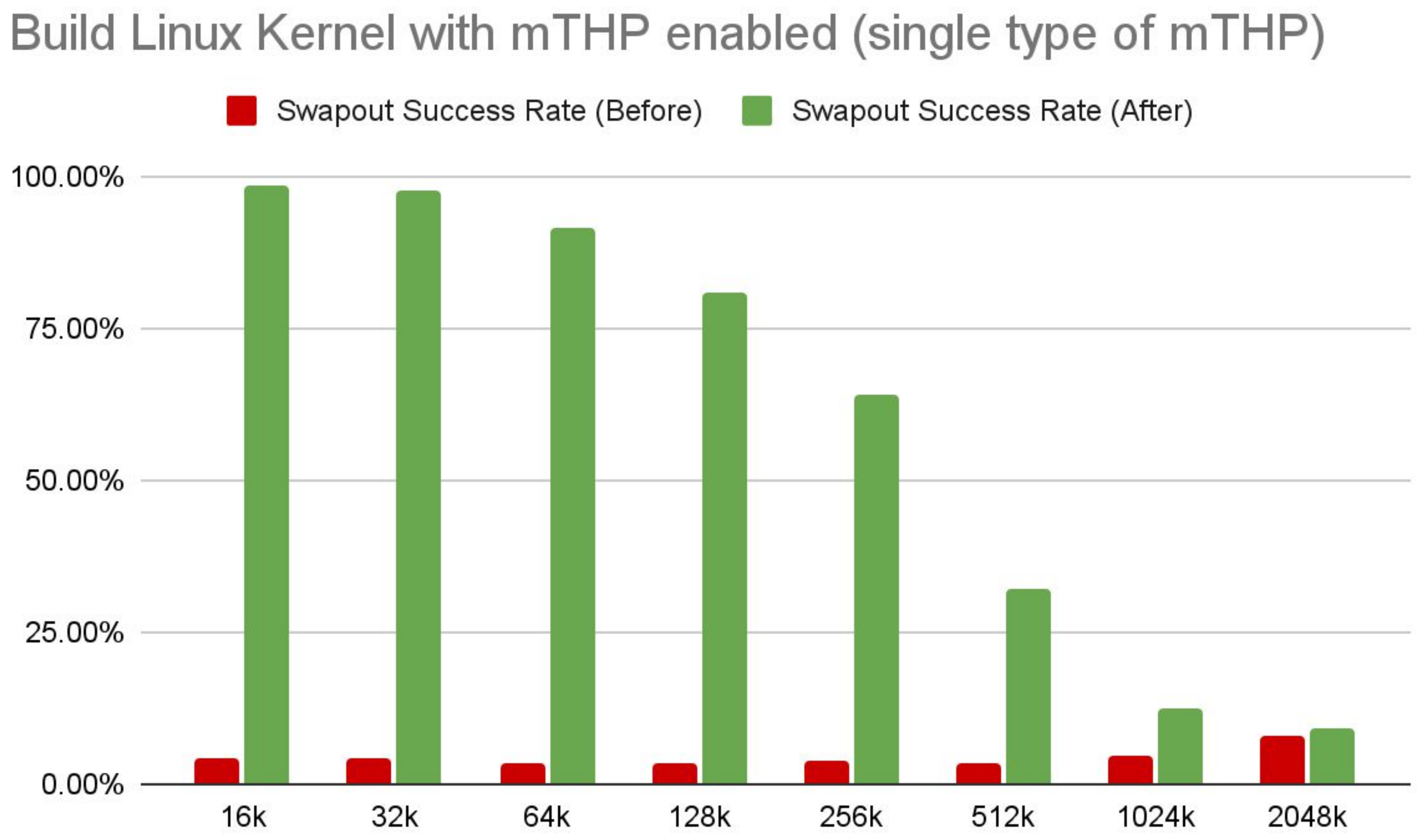
Test and numbers

- Intree mTHP swap bench
- **tools/mm/thp_swap_allocator_test.c**
- Synthesis for simulating certain app booting loop on Android / PC.
- ~99% failure rate to ~0% for -a (aligned swap out)



Test and numbers

- mTHP allocation test with build-linux-kernel.
- **Fallback** drop from **~99%** to **~1% - ~90%** (<10% for 64K), high success rate for mTHP.
- SWAP fragmented overtime, even slight fragmentation will make larger order allocation struggle.
- Could be improved by:
  - Reserving part of SWAP as mTHP only, avoid 0 order from polluting these clusters.
  - Non-continuous swapout.



Test and numbers

- Performance
- 1~5% workload performance gain with first step, even without mTHP (mm-stable now).
- ~30% - ~40% workload performance gain with WIP patch, even without mTHP:
- Scaling up build linux kernel test:

With make -j96:

- **make -j96 in 1G memcg** (Before):

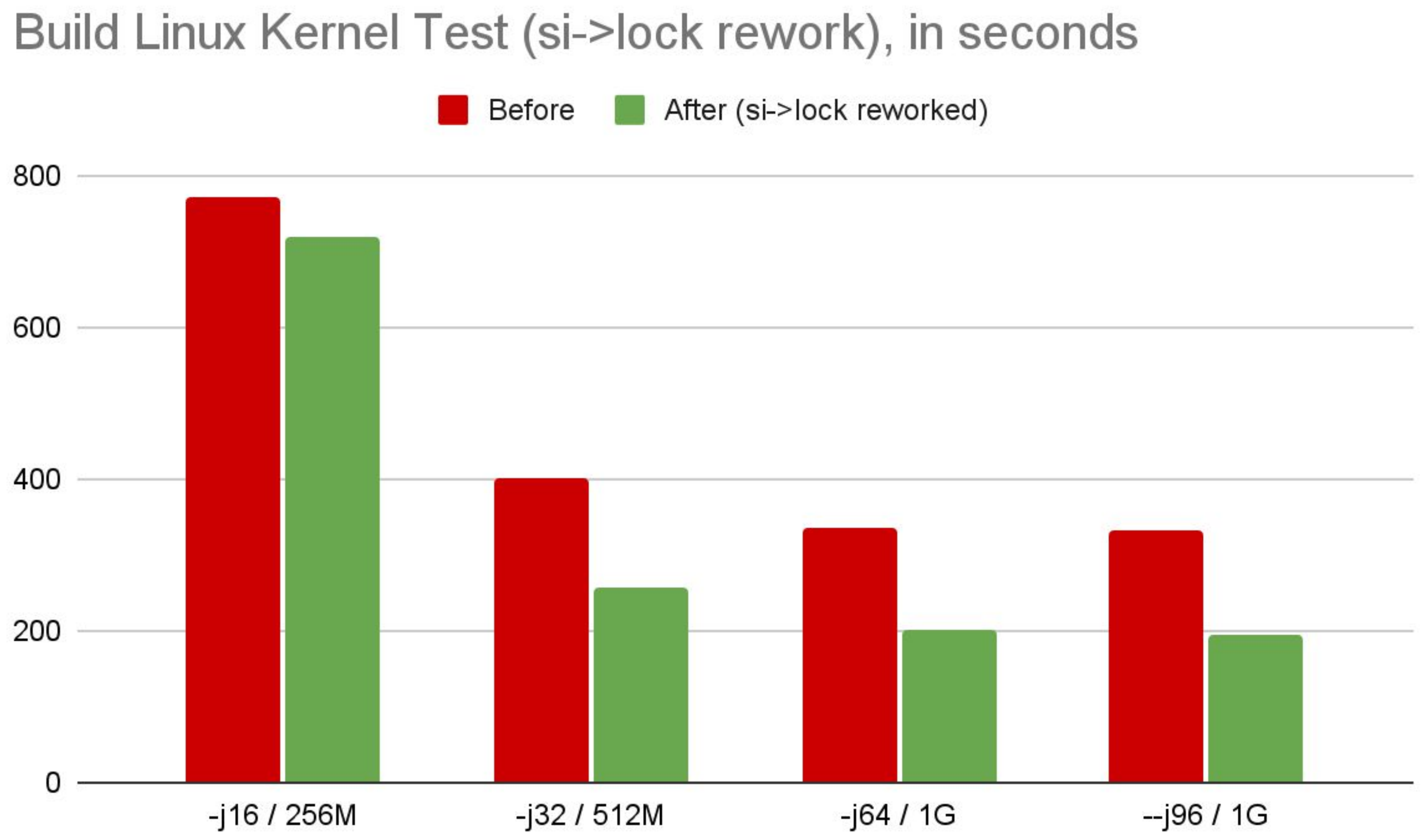
2506.66user **14856.77system 5:02.95elapsed**

- Perf lock contention:
  - `perf lock contention -ab sleep 5`
  - **Total Wait time on si->lock (1.6m in 5s)**

- **make -j96 in 1G memcg** (After): **~35% faster**

2637.94user **9384.29system 3:38.35elapsed**

- Perf lock contention:
  - `perf lock contention -ab sleep 5`
  - **Total Wait time on si->lock (<5s in 5s)**



Questions?

- Chris Li <chrisl@kernel.org> , Kairui Song <kasong@tencent.com>

Linux Plumbers Conference

Vienna, Austria | September 18-20, 2024

Appendix

Test and numbers

- More impressive data on Android with 64K mTHP and optimized ZRAM
- Reference to another topic “**mTHP swap-out and swap-in**”

Test and numbers

- mTHP allocation test for memtier / build-linux-kernel
- **Fallback** drop from **~99%** to **~1% - ~90%** (<10% for 64K), high success rate for mTHP.
- SWAP heavily fragmented overtime, could be covered by non-continuous swapout