

Memory Page Pooling with Auto Refill

Sudarshan Rajagopalan

Engineer, Staff
Qualcomm Innovation Center, Inc.

quic_sudaraja@quicinc.com

1. About me and Team
2. Memory Page Pooling
3. Current Limitations in Generic Memory Pooling
4. Automatic Pool Refilling
5. dmabuf System Heap Auto Pool Refill Implementation
6. Performance and Allocation Latency Numbers
7. Advantages and Limitations

About me

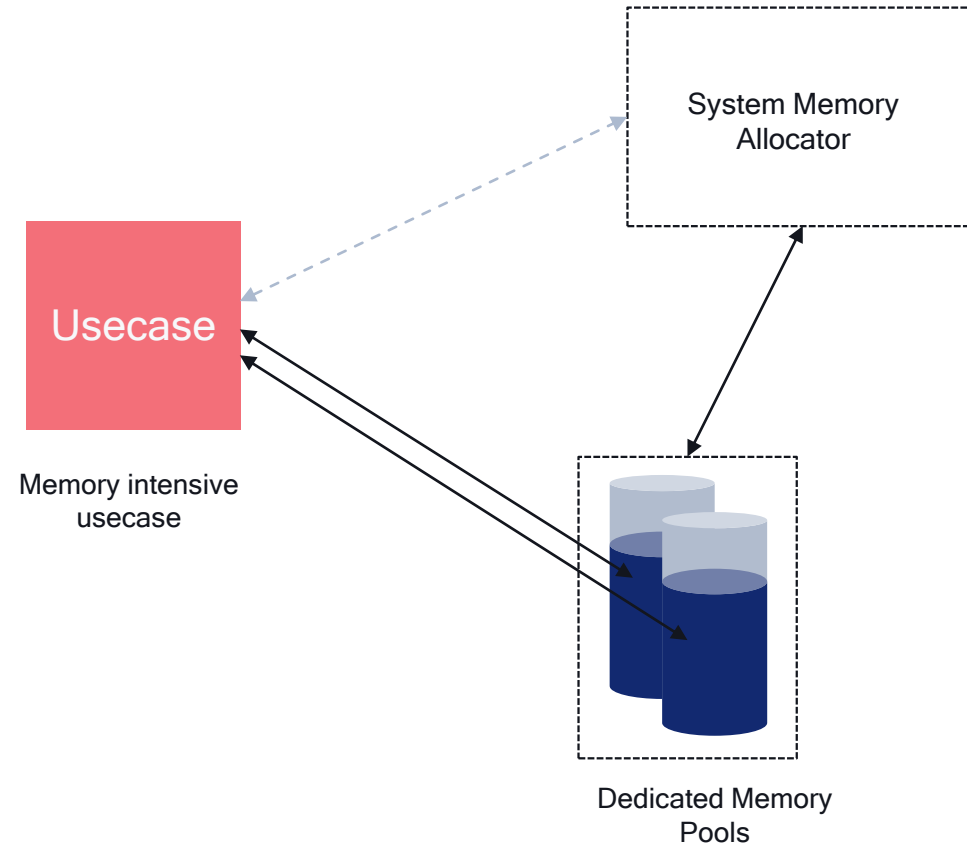
- Linux Kernel Developer in the Linux Kernel team at Qualcomm Innovation Center, Inc. in San Diego, California.
- Interests are in Linux Kernel Memory Management, Embedded Systems and Firmware, System designs and Computer Architecture.



Memory Page Pooling

Memory Pooling

Memory management technique to efficiently allocate and deallocate memory especially in environment where such memory allocations are frequent.



Memory Pooling

Advantages of Memory Pooling in Embedded Systems -

- **Efficient memory management** - availability of pre-allocated memory eliminates overhead of dynamic runtime allocation.
- **Reduced memory fragmentation** - pre-allocated memory of fixed size
- **Increases memory utilization.**
- **Reduces allocation latency** - faster than system memory allocator.
- **Predictable performance** - memory allocation is deterministic and fixed time complexity. Crucial for real-time embedded system.

Can memory pooling be further improved?

Motivation

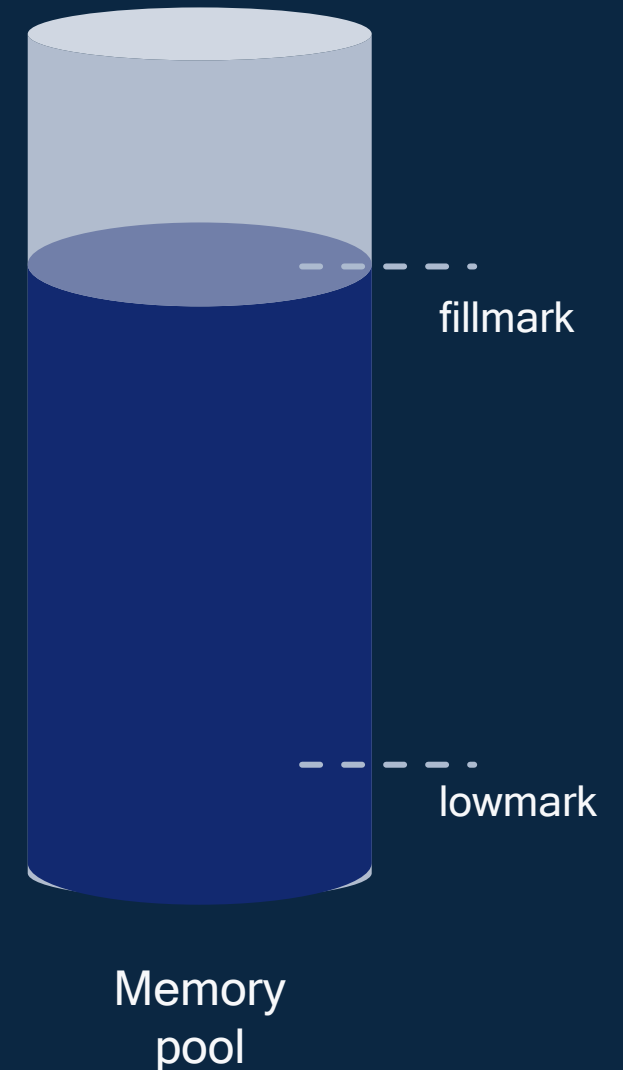
Need for memory pooling

- Memory pooling is great for improving allocation latency as memory is immediately fetched from the pool instead from the system.
- Its possible these pools are empty for longer time thereby impacting latencies.
- Increase the probability of these pools being “filled” in advance so that memory intensive tasks would benefit allocating directly from the pools.
- Once pools are nearly depleted, refill them automatically based on system memory stats upto certain “fillmark”.

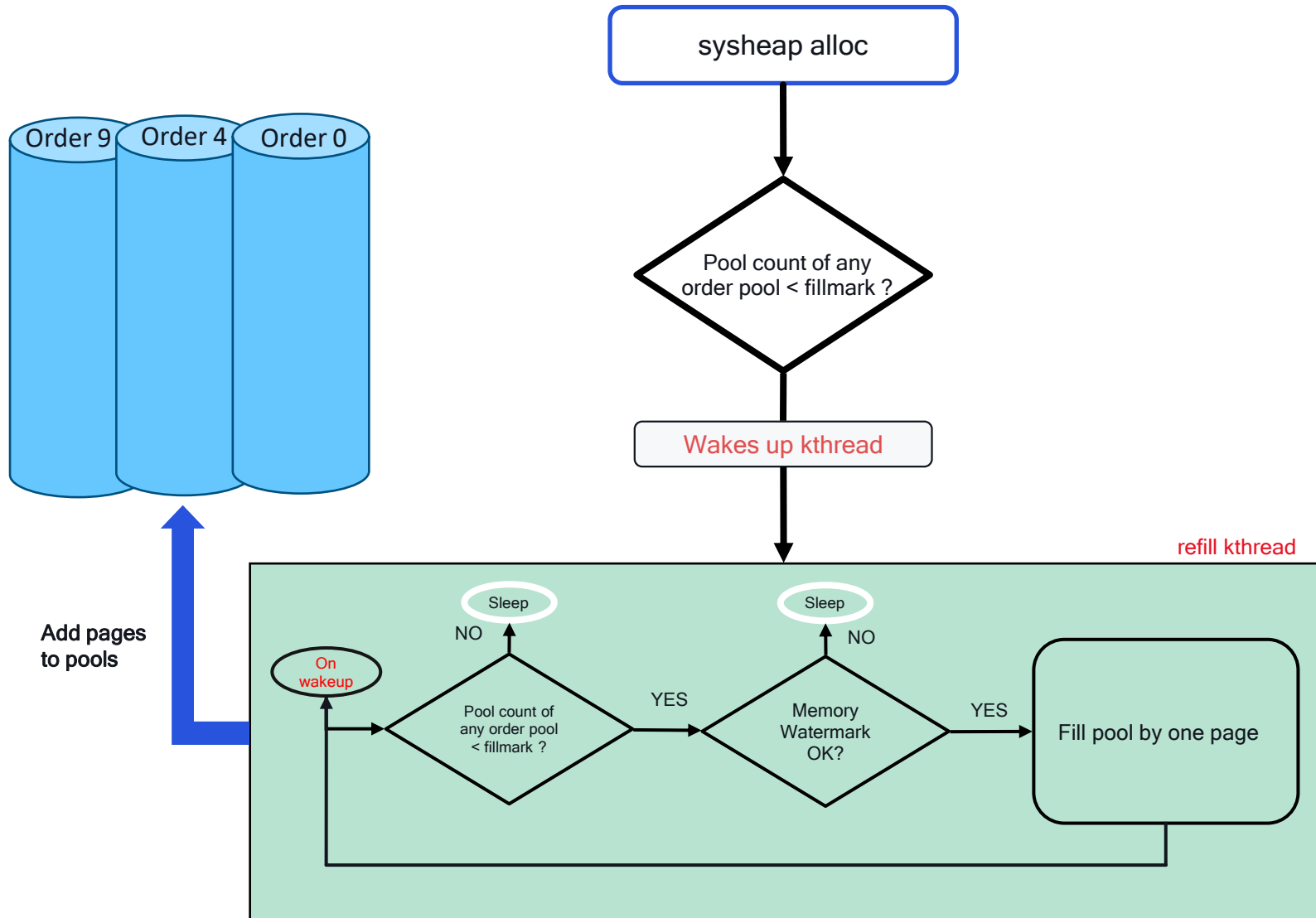
Automatic Refill of Pools

Memory Pool Auto Refill

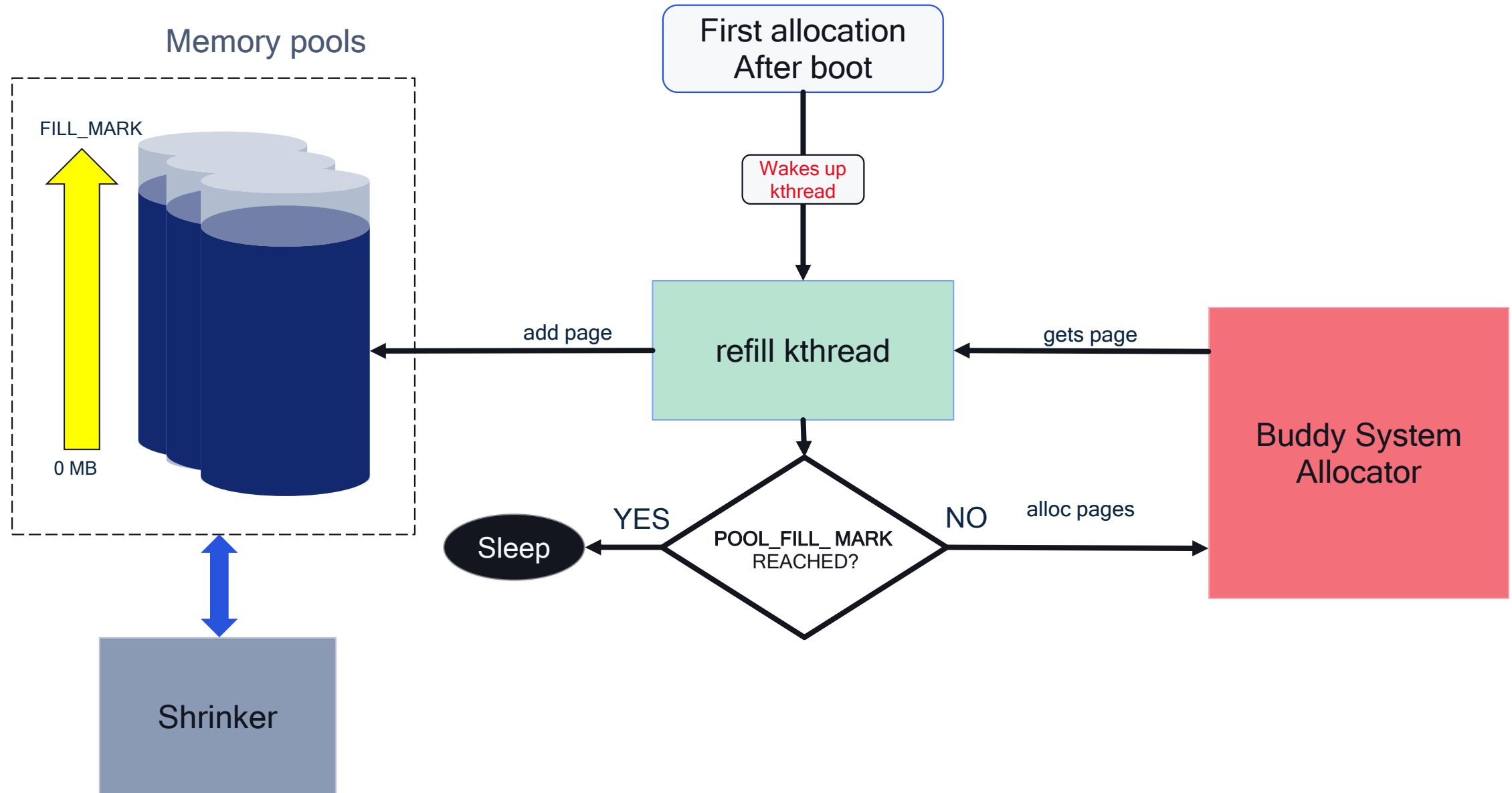
- Refilling of the pools are at the very first memory allocation
- This refilling is done asynchronously via kthreads and fills the pools till “fillmark” of the pool is reached.
- Once pools are filled, memory intensive use cases such as Multimedia clients can directly allocate from the pools instead of buddy system
- The memory in these pools can be quickly reclaimed through memory shrinkers when system is under memory pressure
- On each memory allocation, the pool counts are checked to see if its below “lowmark”
- If pool count < “lowmark, the kernel wakes up the refill kthread and fills the pool asynchronously till “fillmark” of the pool is reached



How it works



How it works



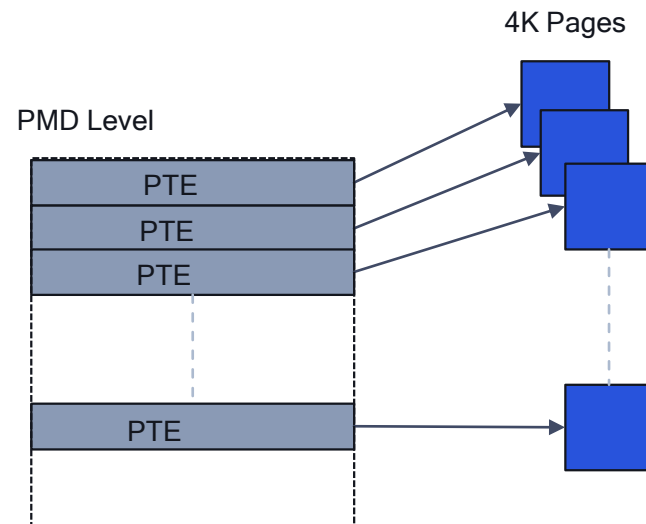
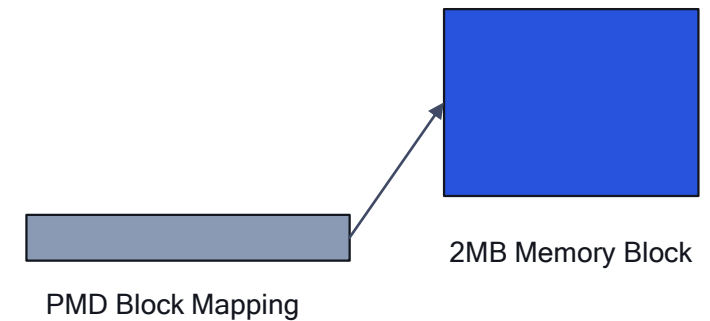
Few Technical Things to
Consider...

Memory Shrinkers in Linux Kernel

- Shrinkers are callback routines used in Linux kernel which are invoked under memory pressure in the system.
- Users (drivers/modules) of memory pools can register to shrinker using `register_shrinker()`.
- Under memory pressure, kernel will invoke all registered shrinker routines via `vmscan`, scanning for extra memory that the drivers can release.
- This makes sure memory in the pools are reclaimed when system is in memory pressure.
- Hence it does not impact any run-time memory regression, but this feature helps in reducing memory allocation latency significantly.

2MB Memory Blocks

- In arm64 platforms, a 2MB memory block can be identified by a single block mapping entry in the MMU.
- These are usually in the PMD level.
- Requires 1 single entry in the TLBs.
- Reduces the MMU Page Table Walk by 1 level.
- Reduces TLB pressure, thereby increases CPU/device throughput.



Lowering Prio / Nice value of refill kthreads

- The refill kthreads have lower priority value (prio=130/Nice val=10) so these would not interfere with normal priority tasks.
- Since refill kthreads would often run frequently, reducing the priority of these tasks makes sure they don't hog CPU thereby increasing power consumption.

Technical Configurations of Auto Refill

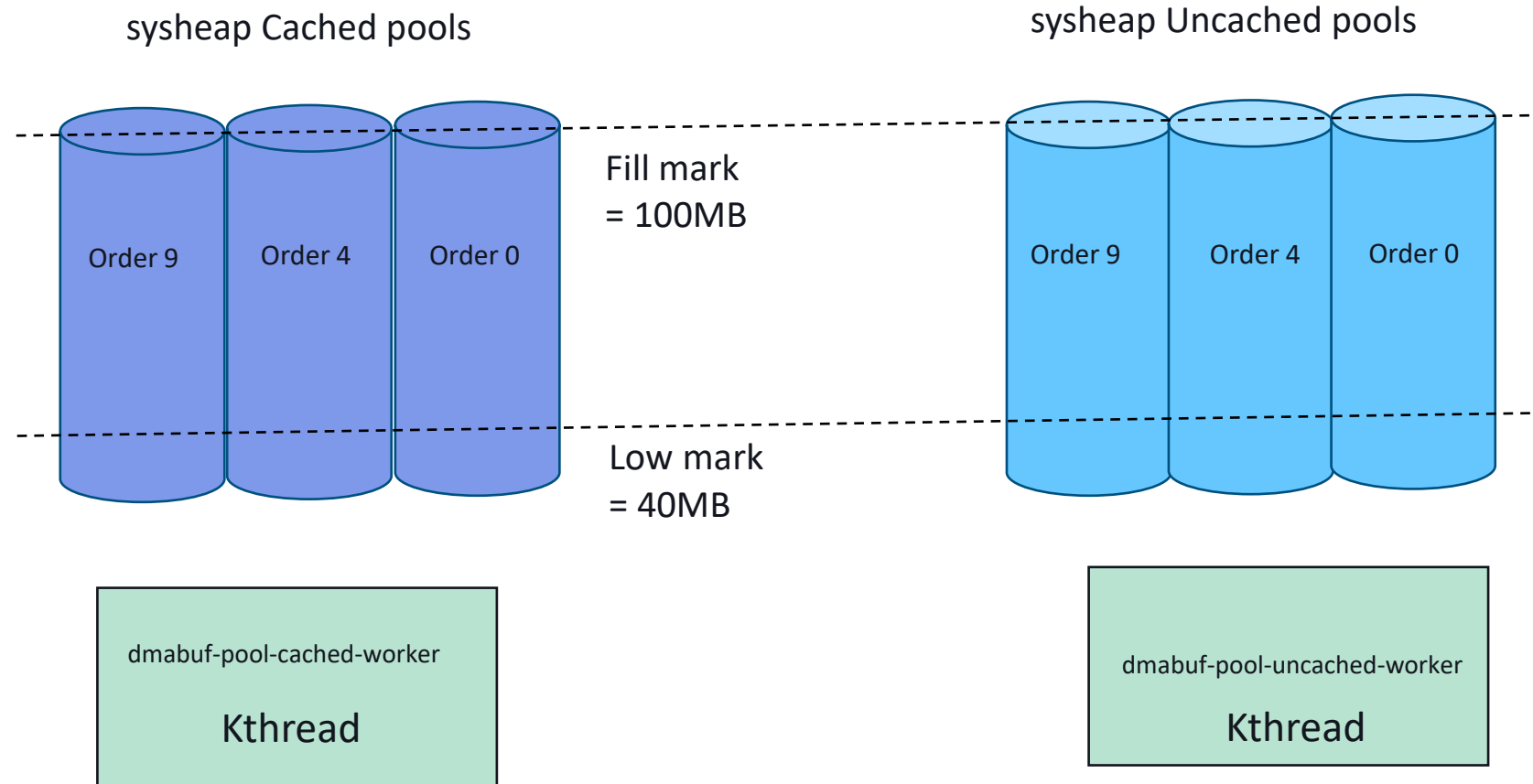
- `POOL_FILL_MARK` - basically size of the pool up to which the refill kthread fills the pool with memory.
- `POOL_LOW_MARK` - configured as percentage of pool size. Example - 40% of `POOL_FILL_MARK`.
- Can be configured uniquely for specific use case.
- Differ pool refilling if system under memory pressure - avoids unwanted repetitive refilling and reclaiming of buddy pages on the pool.

Auto Refill Implemented in dmabuf System Heap

Dynamic page pool System heap

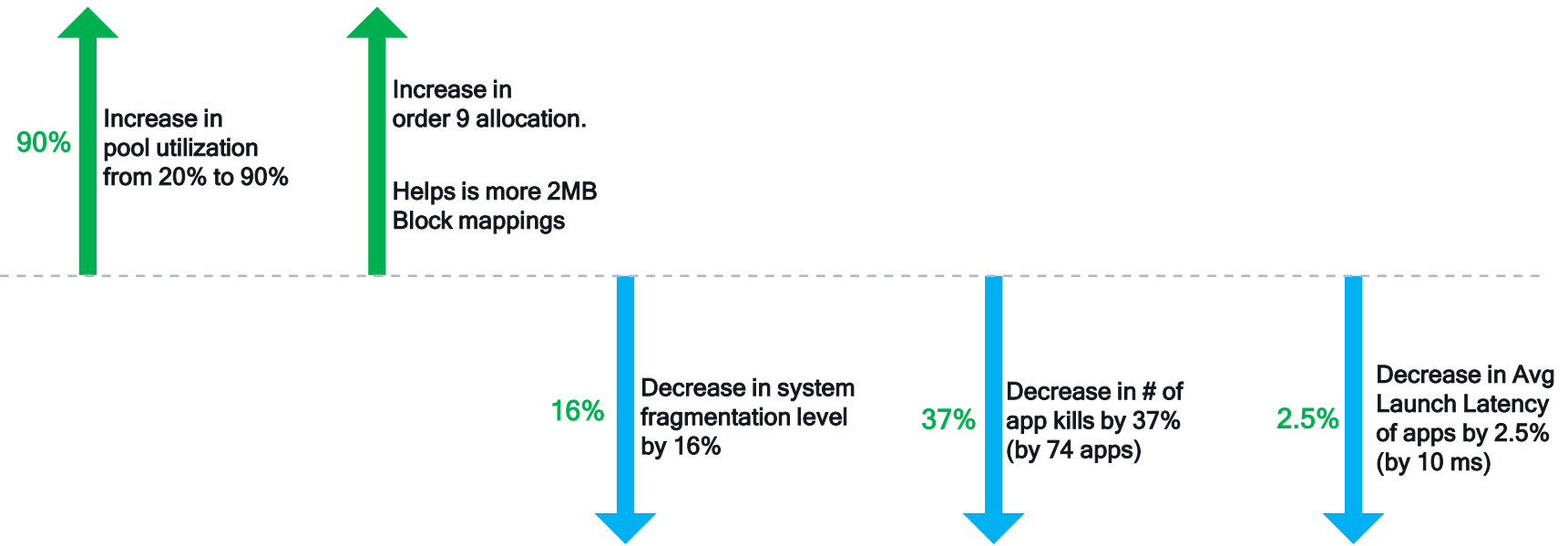
- The auto pool refilling feature is implemented in dmabuf system heap of Qualcomm Innovation Center.
 - https://git.codelinaro.org/clo/la/kernel/msm-5.15/-/blob/kernel.lnx.5.15.r14-rel/drivers/dma-buf/heaps/qcom_dynamic_page_pool.c
 - https://git.codelinaro.org/clo/la/kernel/msm-5.15/-/blob/kernel.lnx.5.15.r14-rel/drivers/dma-buf/heaps/qcom_system_heap.c#L225
- 3 uncached pools - each of page order 9, 4 and 0
- 3 cached pools - each of page order 9, 4 and 0
- Fillmark - 100MB, lowmark - 40MB

Dynamic page pool System heap



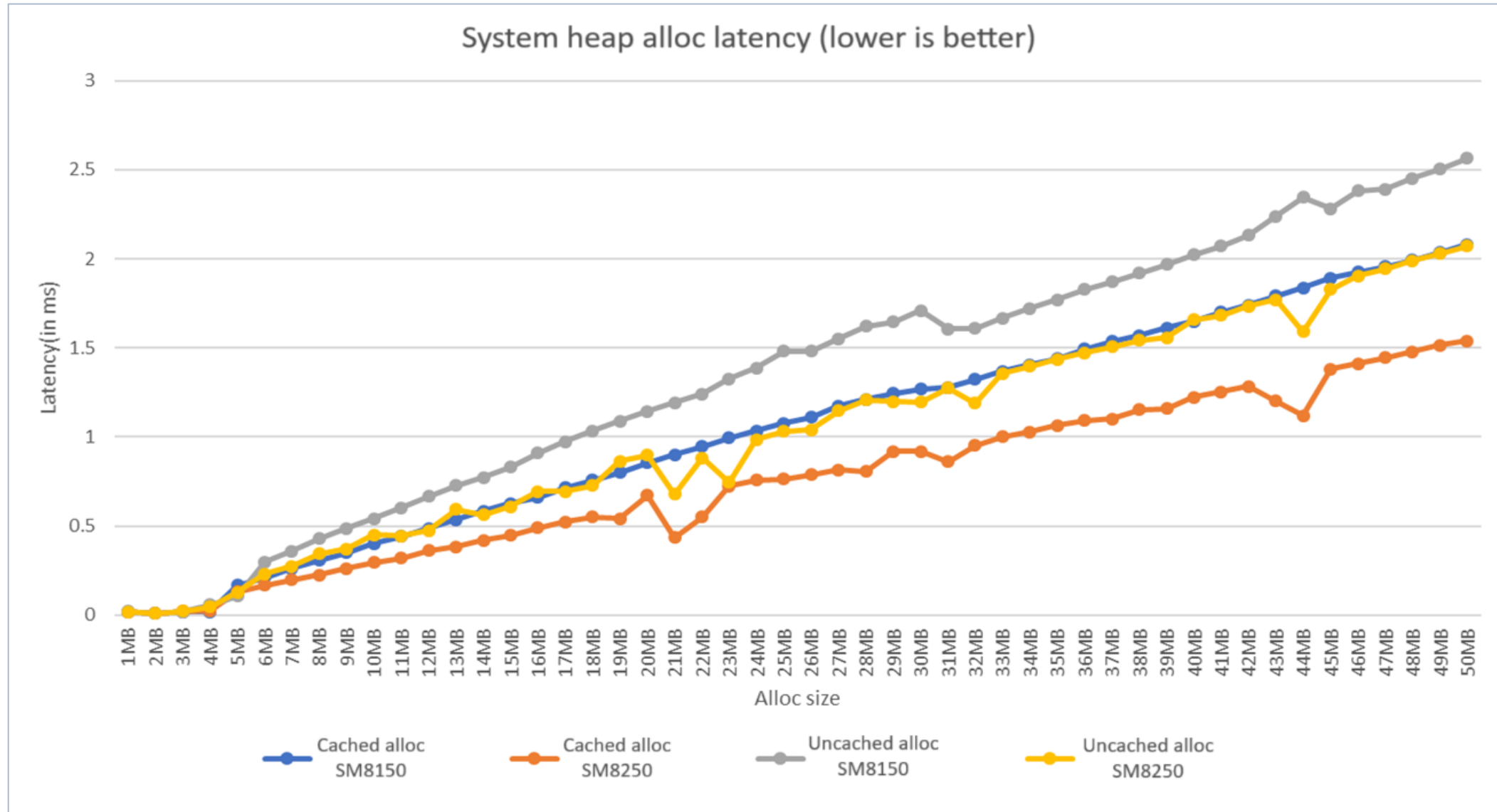
Performance and Allocation Latency

Dynamic page pool System heap



Allocation Latency

- Graph showing improvement in alloc latency in Qualcomm® SM8150 vs Qualcomm® SM8250 with auto refill feature enabled.



Pros and Cons

Pros and Cons

Pros

- Memory pooling is great for improving allocation latency as memory is immediately fetched from the pool instead from the system.
- Its possible these pools are empty for longer time thereby impacting latencies.
- Increase the probability of these pools being “filled” in advance so that memory intensive tasks would benefit allocating directly from the pools.
- Once pools are nearly depleted, refill them automatically based on system memory stats up to certain “fillmark”.

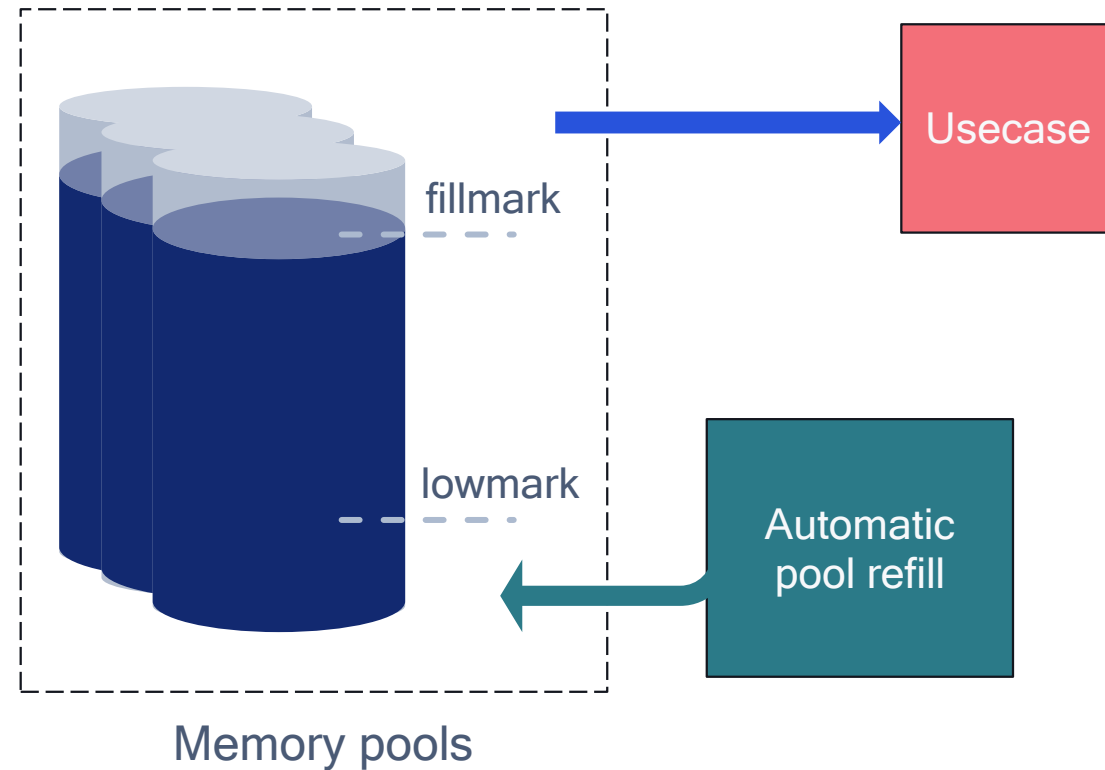
Cons

- “FILL_MARK” amount of memory will be resident in the pools until the shrinkers run. There will be some CPU activity in-terms of reclaiming this memory from the pools.

Uses of memory pool auto refill

- Memory intensive applications such as multimedia - video, camera, graphics etc.
- Real-time embedded systems with predictive memory latency
- Use cases requiring higher order pages (reduces TLB pressure)
- Applications where memory will be frequently be allocated and deallocated.

Just to wrap up...



- Improved alloc latencies
- Reduced fragmentation
- Increased pool utilization
- Reduced TLB pressure
- Lower app kills

All gains, purely by software implementation alone!!

Questions



Thank you



Follow us on: [in](#) [X](#) [@](#) [▶](#) [f](#)

For more information, visit us at:

qualcomm.com & qualcomm.com/blog

Nothing in these materials is an offer to sell any of the components or devices referenced herein.

© Qualcomm Technologies, Inc. and/or its affiliated companies. All Rights Reserved.

Qualcomm is a trademark or registered trademark of Qualcomm Incorporated. Other products and brand names may be trademarks or registered trademarks of their respective owners.

References in this presentation to "Qualcomm" may mean Qualcomm Incorporated, Qualcomm Technologies, Inc., and/or other subsidiaries or business units within the Qualcomm corporate structure, as applicable. Qualcomm Incorporated includes our licensing business, QTL, and the vast majority of our patent portfolio. Qualcomm Technologies, Inc., a subsidiary of Qualcomm Incorporated, operates, along with its subsidiaries, substantially all of our engineering, research and development functions, and substantially all of our products and services businesses, including our QCT semiconductor business.

Snapdragon and Qualcomm branded products are products of Qualcomm Technologies, Inc. and/or its subsidiaries. Qualcomm patented technologies are licensed by Qualcomm Incorporated.