Qualcomm

Denver, CO

23 - 25 June 2025

Dynamic VM Memory Resizing Daemon (vmrd)

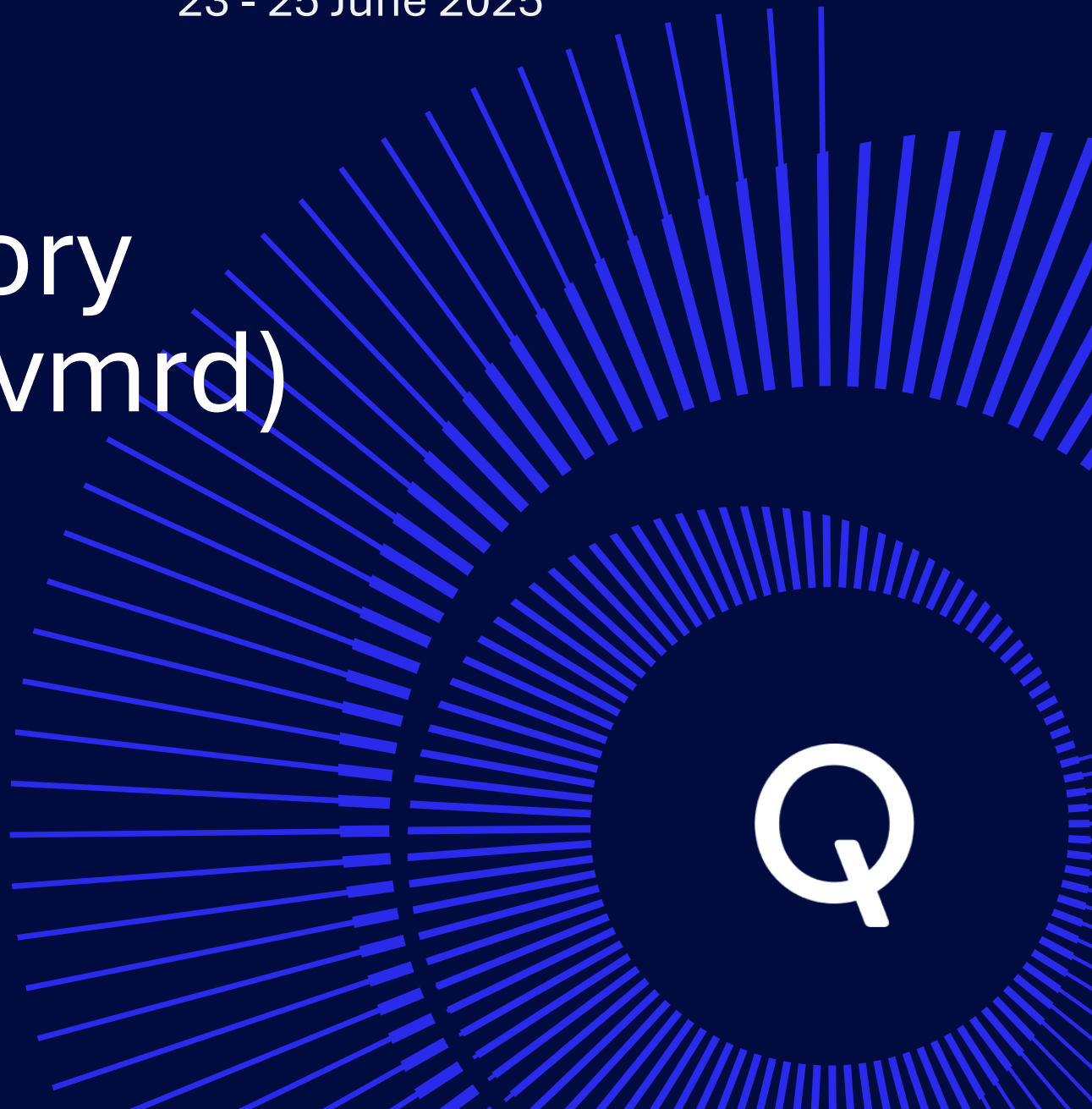
Sudarshan Rajagopalan

Engineer, Staff

Qualcomm Innovation Center, Inc.

quic_sudaraja@quicinc.com

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



Agenda

- About me and the team
- Resizing of Secondary VMs
- Memory Pressure Detection in VMs
- VM Resizing Daemon (vmrd)
- Challenges and Limitations
- Technical Implementation
- Other generic ways to resize VMs

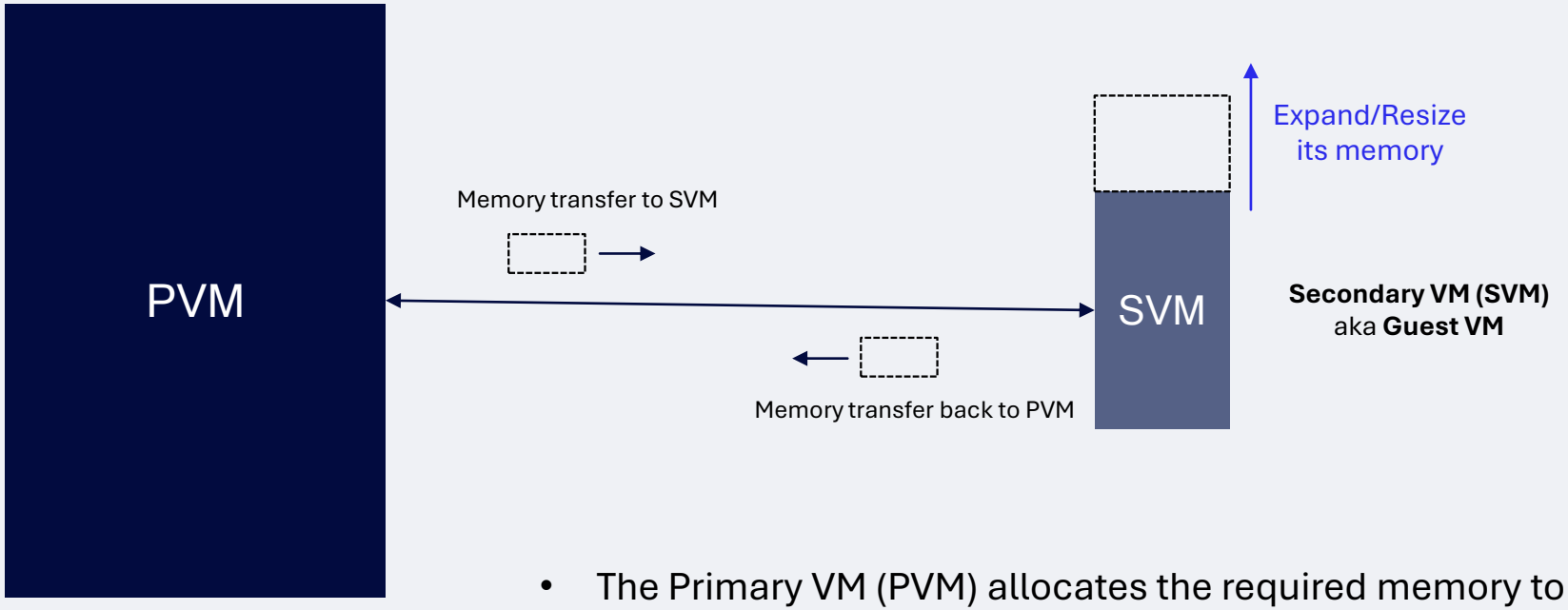
About me

- Linux Kernel Developer in the Linux Kernel team at Qualcomm Innovation Center, Inc. in San Diego, California since 2017.
- Working mainly on the Linux Kernel Memory Management.
- Interests are in Kernel Development, Embedded Systems and Firmware, System designs and Computer Architecture.

LinkedIn: https://www.linkedin.com/in/sudarshanraj/

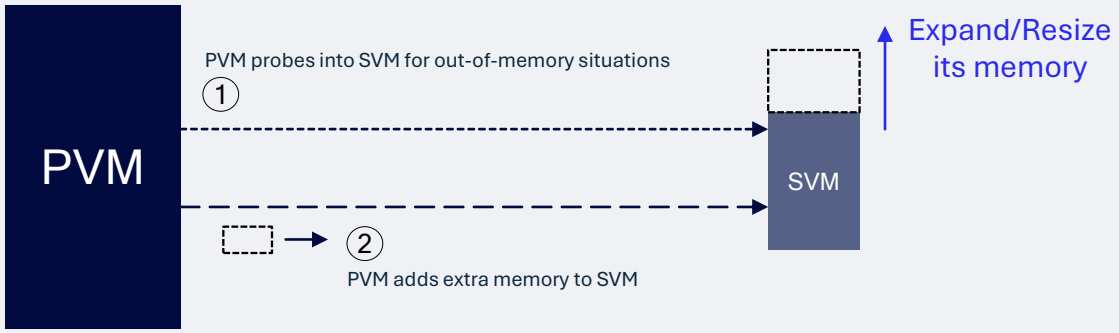
Resizing of Secondary VMs

Resizing of Secondary VM from the Primary VM

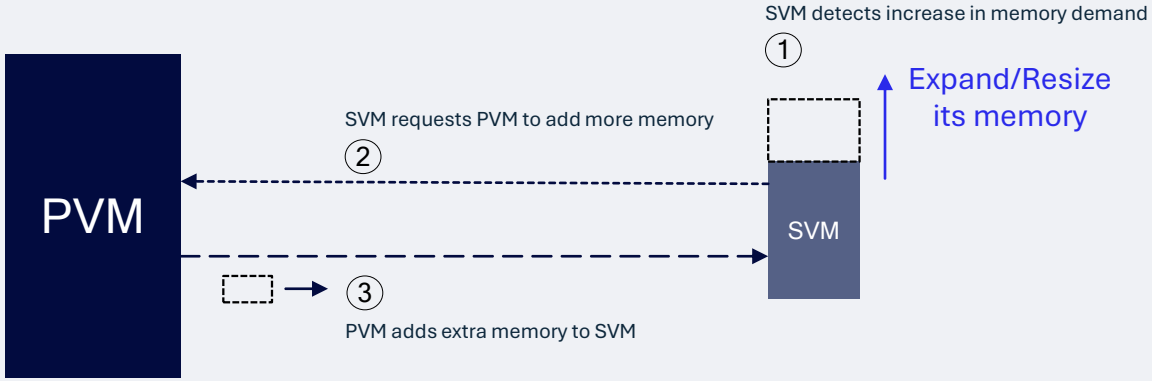


- The Primary VM (PVM) allocates the required memory to transfer to the Secondary VM (SVM).
- The SVM attaches this memory from the PVM into its IPA (intermediate physical address) space.
- The memory headroom in PVM reduces by this memory.
- The SVM can remove and transfer back this memory back to PV.

Resizing of Secondary VM from the Primary VM



- Resize the VM with Admin intervention (Host driven approach)



- VM itself requests for resizing (Guest driven approach)

Resizing VMs based on Memory Demands

Resizing VMs based on Memory Demands

- The Secondary (Guest) VM can monitor its memory pressure/demands in the system, and can request to add/remove memory from the PVM.
- The SVM would need some mechanism to detect memory demands in the system in real time, so that extra memory can be added before a memory intensive task runs out-of-memory.
- Similarly, the SVM needs a mechanism to check extra idle memory after usecases are done, so that the extra memory (which was added before) can be given back to PVM.
- Guest driven approach – the Guest (Secondary) VM determines and drives the memory adding/removing request from the Host (Primary) VM.

Memory Pressure Detection using PSI (Pressure-Stall Information)

PSI (Pressure Stall Information)

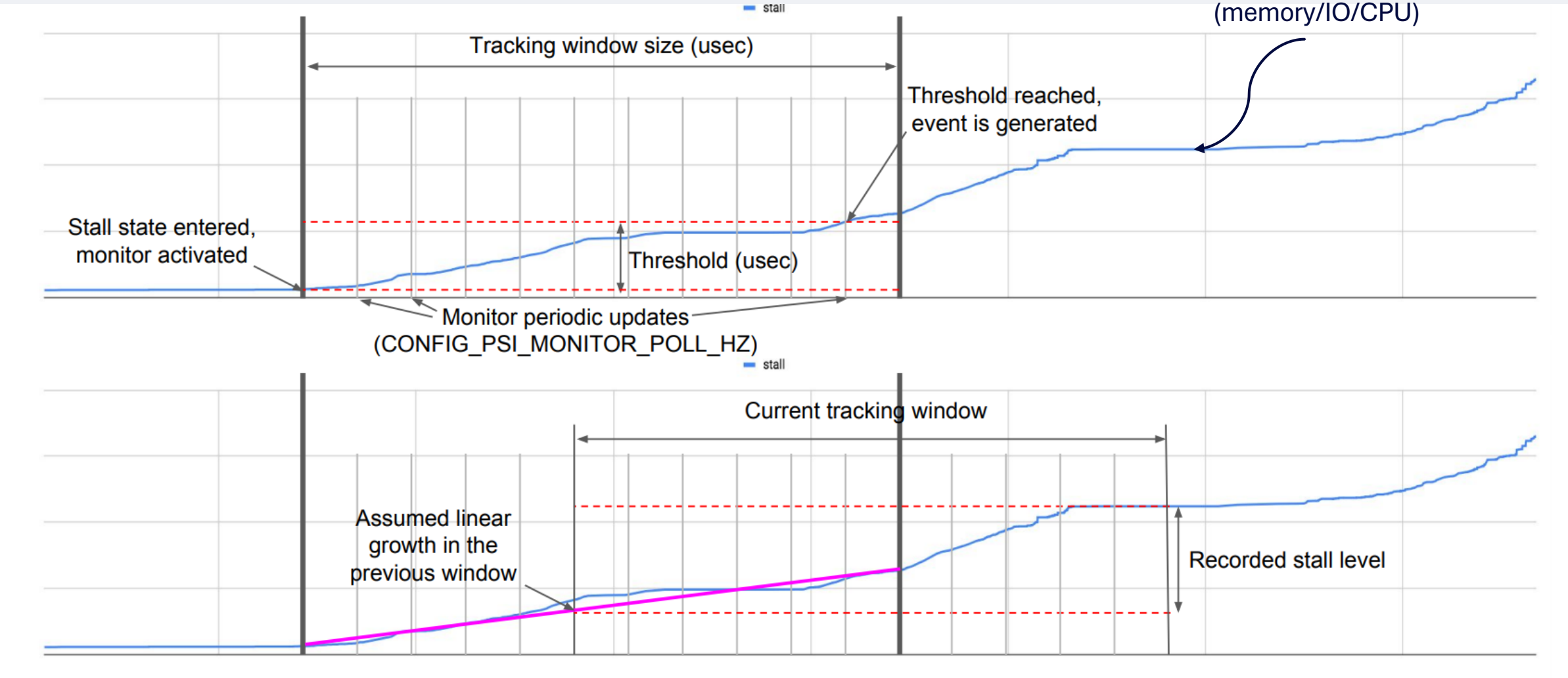
- PSI measures amount of time tasks are delayed as a result of resource contention such as **IO**, **CPU** or **Memory**. It is recorded as total stall time and averages over 10, 60 and 300 secs are provided.
- PSI separately records complete (FULL) and partial (SOME) stall amounts.

```
# cat /proc/pressure/memory
some avg10=70.24 avg60=68.52 avg300=69.91 total=3559632828
full avg10=57.59 avg60=58.06 avg300=60.38 total=3300487258
```

PSI Monitor

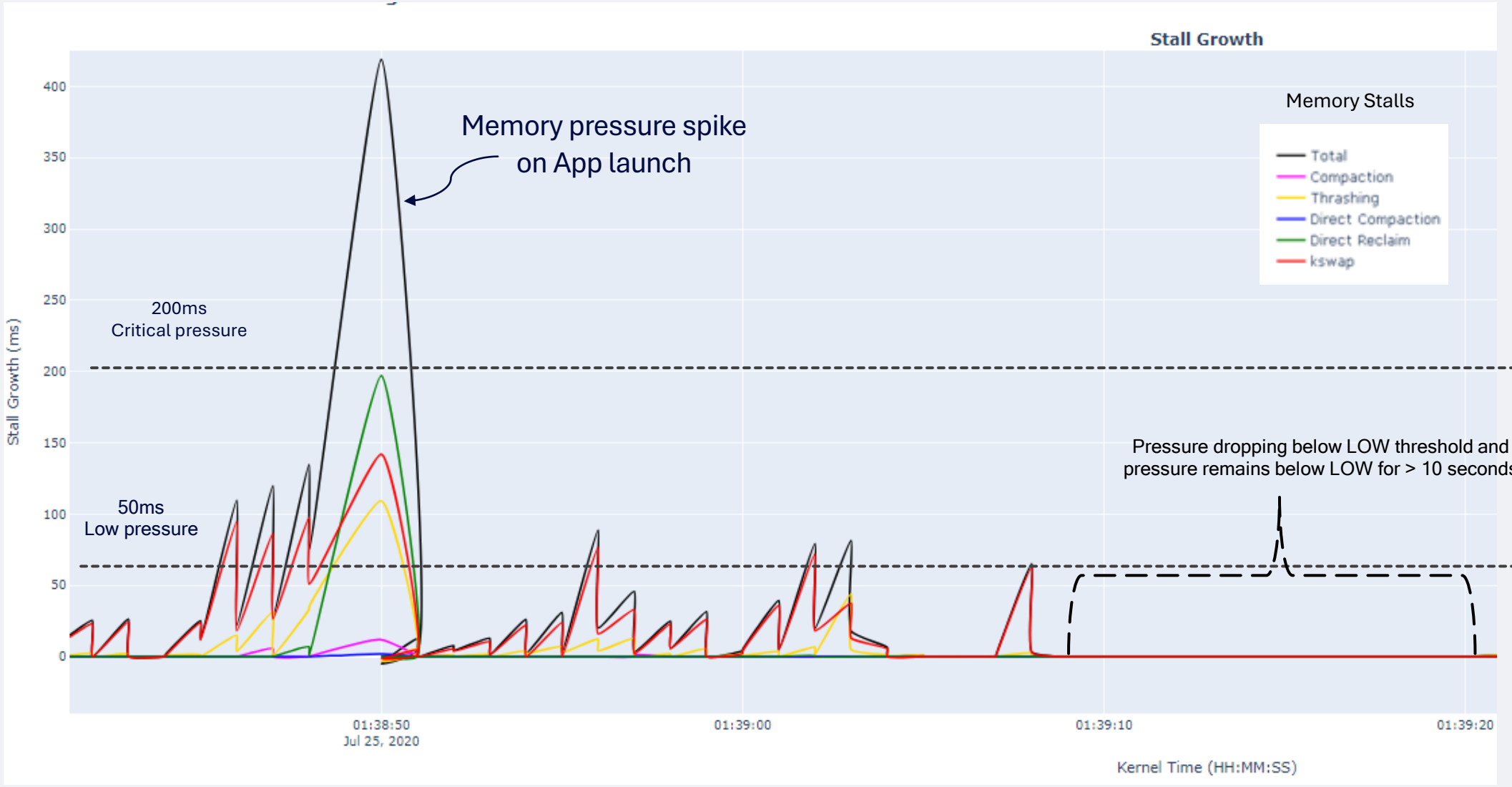
- API similar to vmpressure allowing userspace to wait for events (select/poll/epoll)
- Activates only when monitored metric enters stall state and deactivates upon exit
- Polls stall duration at a frequency configurable via kernel configuration flag
- Provides per-monitor configurable threshold and tracking window size

PSI (Pressure Stall Information)



Source: https://linuxplumbersconf.org/event/2/contributions/221/attachments/45/52/02._PSI_monitor_for_memory_pressure_detection.pdf

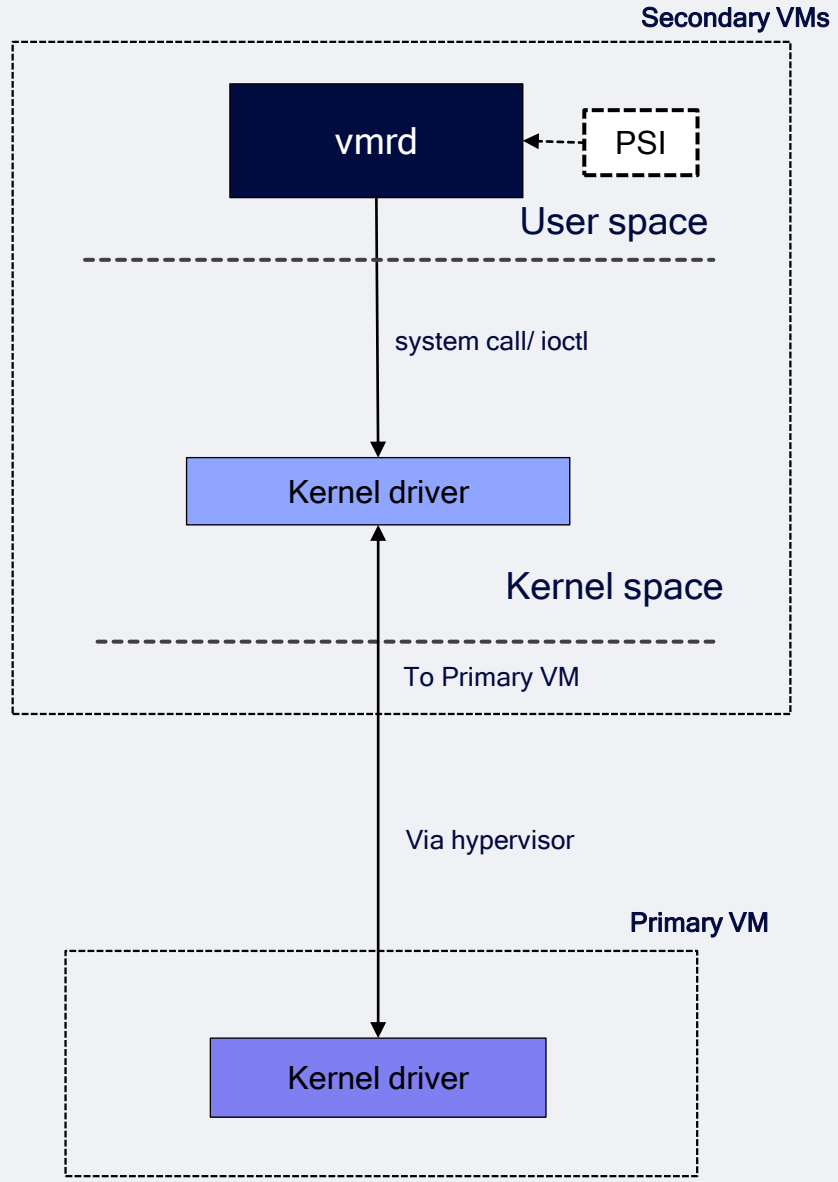
PSI (Pressure Stall Information)



VM Resizing Daemon (vmrd)

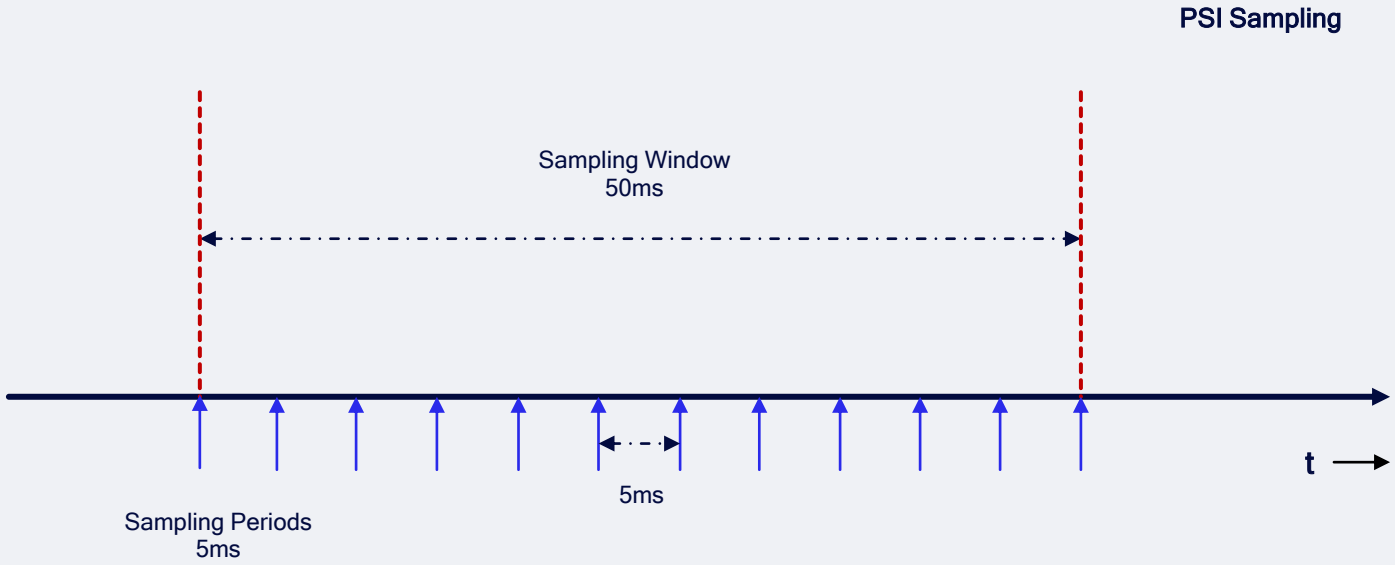
VM Resizing Daemon (vmrd)

- vmrd – a native userspace daemon that registers to Linux kernel's PSI mechanism and listens to memory pressure events.
- This daemon running on guest VM adds memory when demand for memory increases.
- Also tracks for memory pressure going down as the usecase ends and frees memory to release memory back to host proactively.
- Uses Memory Hotplug for adding/removing memory.
- Ability to add/remove even smaller memory chunks (e.g. 4MB) compared to memory blocks of 128MB (default size for arm64 targets with 4K page granule).

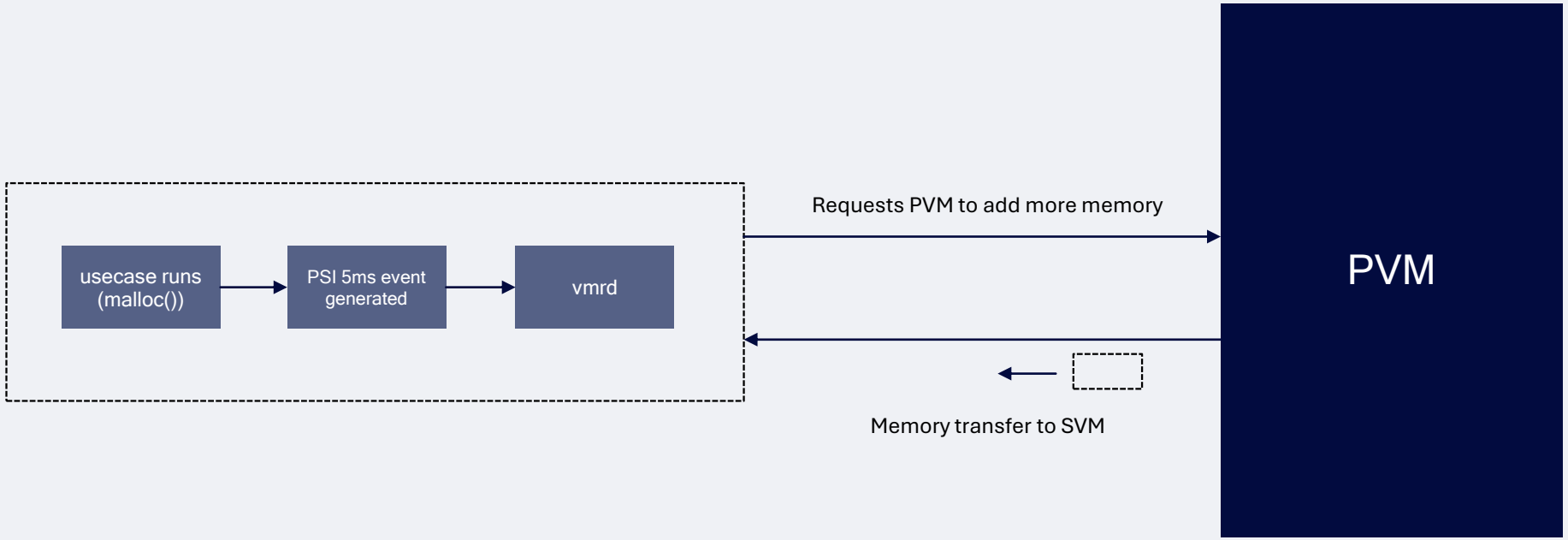


vmrd – detecting increase in memory demand

- Reduce the PSI monitor sampling window to 50ms, so that as small as 5ms increase in memory pressure can be reported to userspace by PSI mechanism.
- When a certain usecase starts in SVM that does memory allocation, the process will stall causing PSI mechanism to generate a memory pressure event to userspace.
- When pressure increases 5ms set threshold, vmrd after checking system memory stats can then make an educated guess that memory requiring usecase has ran and VM system needs memory to be added.

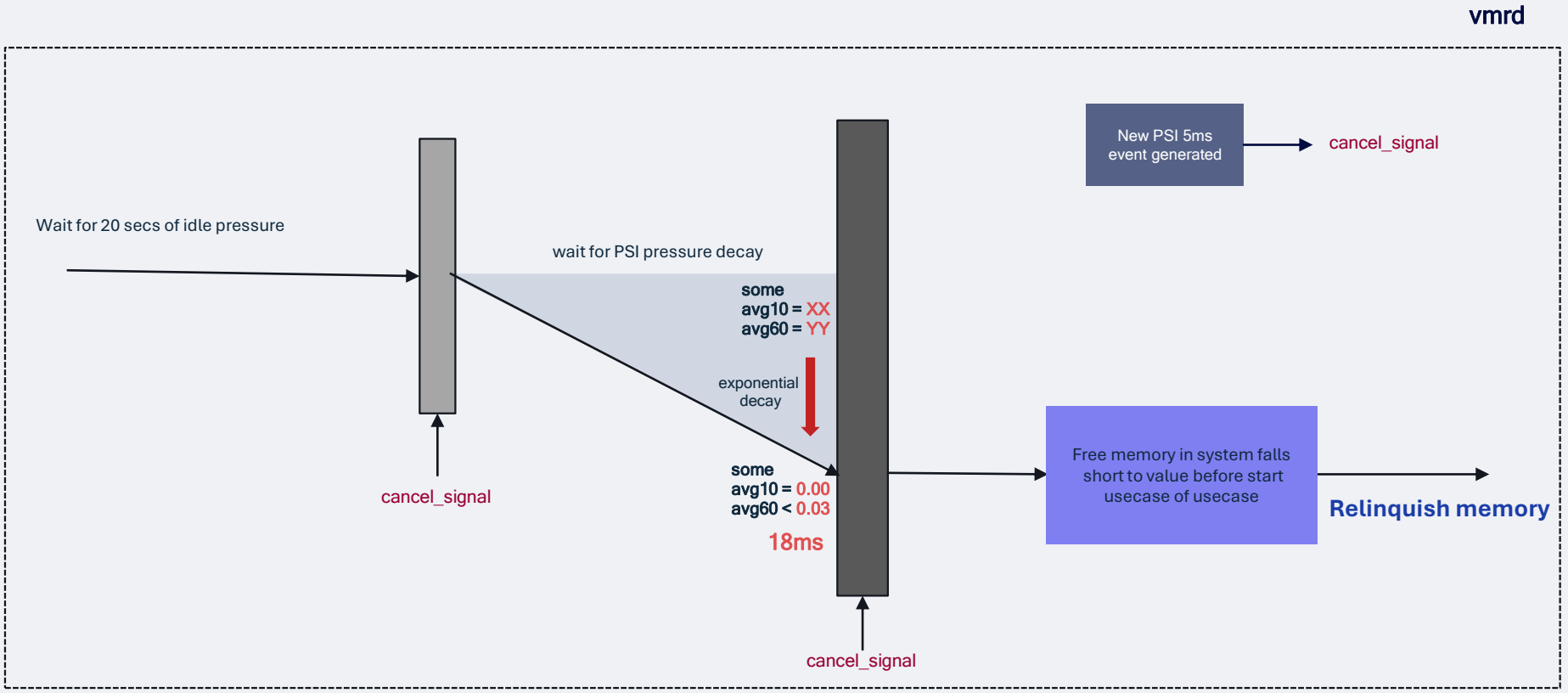


vmrd – detecting increase in memory demand



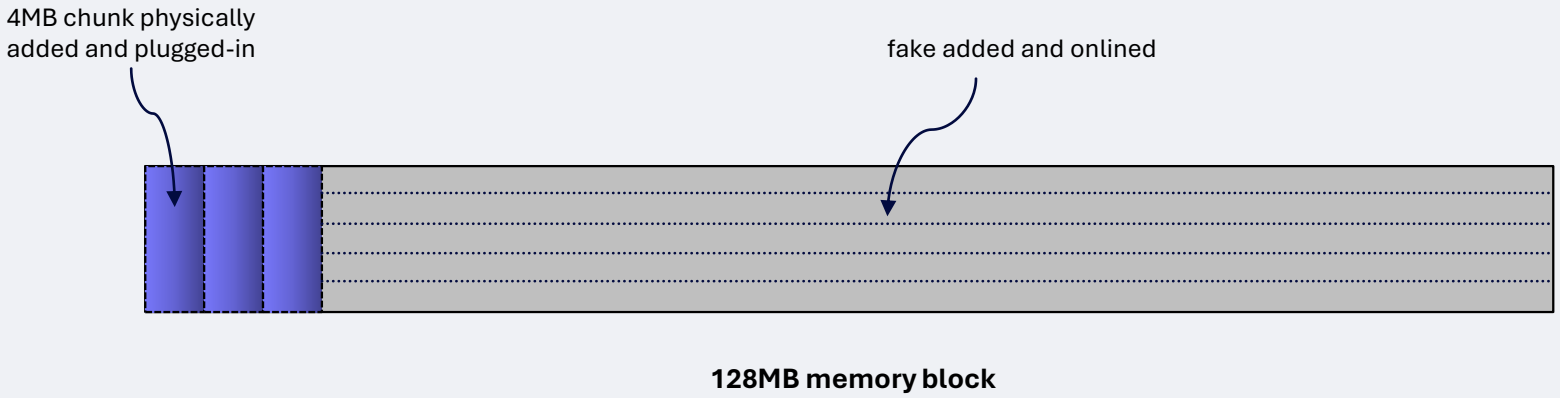
vmrd – relinquish memory after usecase ends

- End of usecase can be detected by PSI pressure going down.
- When PSI pressure goes down, and free memory falls short close to value before start of usecase, this will be an hint as usecase has ended.
- Relinquish memory chunks, one-by-one.

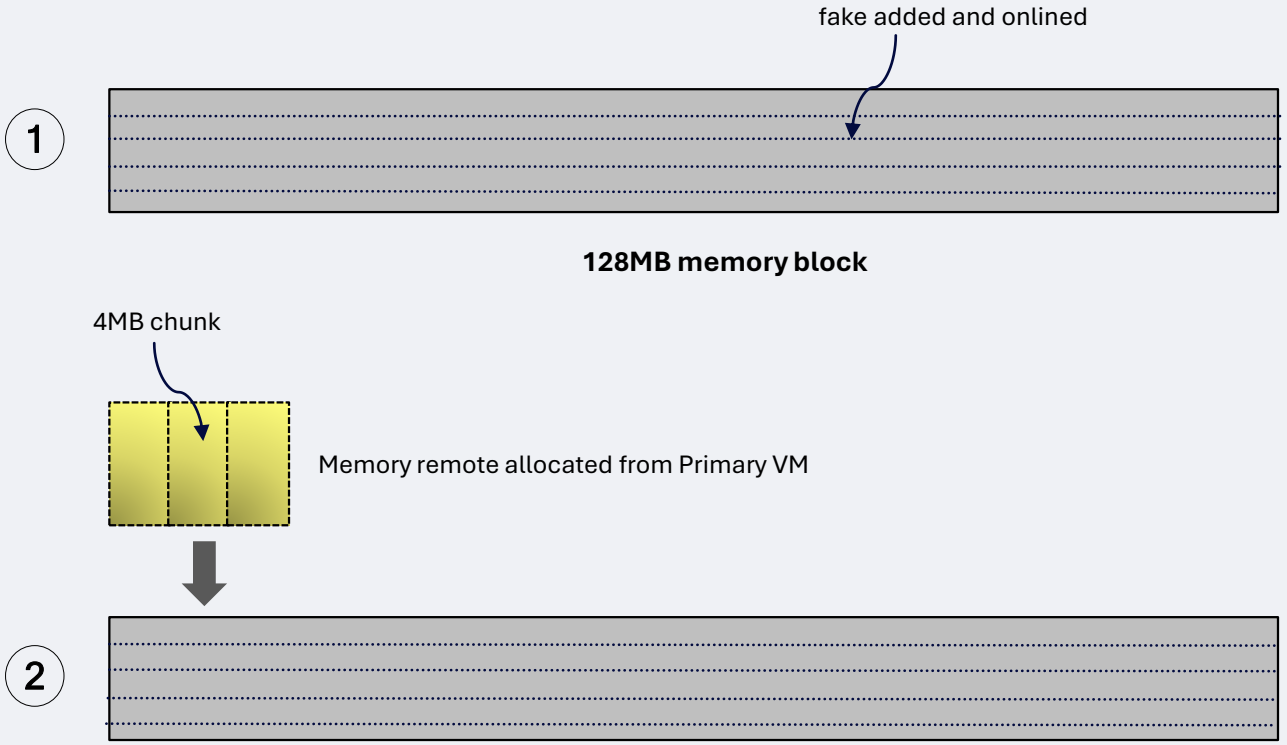


vmrd – splitting block size into multiple smaller chunks

- Memory hotplug only supports adding/removing memblock size (128MB for arm64 targets).
- To allow adding/removing of smaller memory chunks (say 4MB), we need to split the memblock.
- Use memory_on_memmap feature to use part of memblock to host memmap data.



vmrd – splitting block size into multiple smaller chunks



- Plug-in 128MB memory block (virtual add, fake online) using memory hotplug, arch_add_memory().
- Block is marked as “inactive”.
- All pages are marked as **PageOffline()** and **PageDirty()**
- Remote alloc 4MB memory into given IPA address of the block.
- Search for free sub-block, install the 4MB memory chunks.
- Clear PageOffline and Page Dirty bits, invoke free_contig_range() to make memory chunk available to system memory (buddy allocator).

Challenges and Limitations

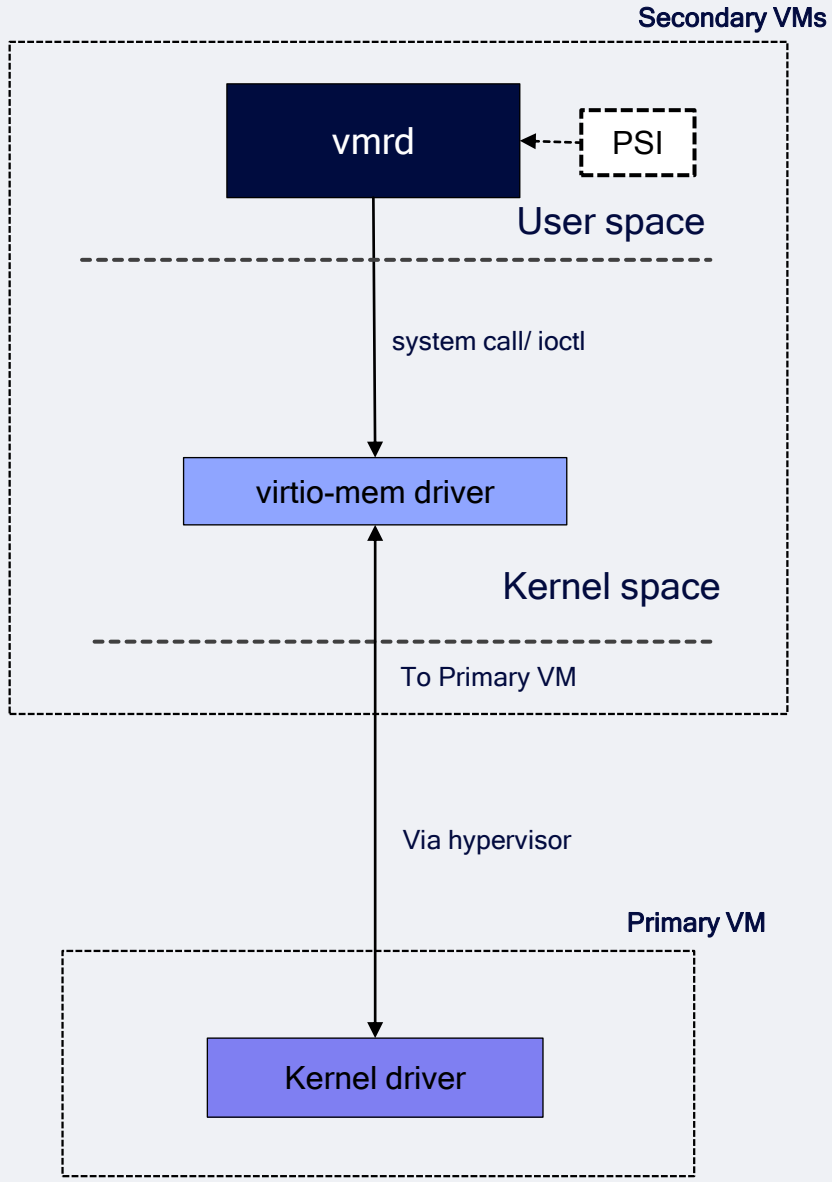
vmrd – challenges and limitations

- Process stall of min 5ms to generate PSI event and userspace (vmrd) taking action.
- Latency incurred in adding the memory chunks – mostly due to **roundtrip delays** between PVM <-> SVM.
- Ability to successfully offline/unplug out a memory block or chunk. A single pinned page can cause offline failure.
- Relinquishing memory is based on estimation.

Technical Implementation

Technical Implementation

- Virtio-mem driver supports the concept of adding sub-blocks within the 128MB memory block using memory hotplug feature.
- Splits the 128MB memory block into multiple virtio-mem blocks (say 4MB), and can be separately added/removed from the system.
- Implementation is leveraged a lot from memory hot-plug feature.
- Qualcomm today uses modified version of virtio-mem driver –
  - added support for memory_on_memmap feature on virtio-mem driver
  - allows userpspace to directly request for adding/removing virtio-mem blocks
- Movable zone is used to add the memory blocks using memory hotplug. The Movable zone in Linux kernel guarantees pages to be isolated and freed, so that memory blocks can be offlined.



Technical Implementation

- Idea of **dynamic resizing of guest VMs** was discussed with Linux upstream community as a RFC (Request for Comment) – https://lore.kernel.org/linux-arm-kernel/1bf30145-22a5-cc46-e583-25053460b105@redhat.com/T/

- **vmrd userspace daemon** – open source code, sent to upstream community – https://lore.kernel.org/linux-mm/de49fa51b794d447acbd097e77b73a3e759f4ad7.1690836010.git.quic_sudaraja@quicinc.com/T/

- Modified **virtio-mem driver** maintained by Qualcomm – https://git.codelinaro.org/clo/la/kernel/qcom/-/blob/kernel.lnx.6.6.r5-rel/drivers/virtio/qti_virtio_mem.c https://git.codelinaro.org/clo/la/kernel/qcom/-/blob/kernel.lnx.6.6.r5-rel/drivers/virtio/virtio_mem.c

Advantages of Dynamic Resizing

Advantages of Dynamic Resizing of VMs

- Takes care of extra memory demand needed naturally in the system
- No explicit calls needed by usecases to add extra memory
- Memory is added only when the system needs it – reduces the memory footprint of idle system.
- System can give away extra free memory back to Primary VM naturally (again without any explic calls)

Some disadvantages –

- Primary VM should make sure the Secondary VMs doesn't over pluggin extra memory than set threshold.
- A rogue application in SVM could result into plugging of all VM memory and lock it. Primary VM would have no way of knowing if SVM genuinely keeping the plugged-in memory.

Other Ways of Resizing VM

Ways of Resizing VM

- Demand Paging method by Google
  - Pages are physically added on demand during page access.
  - This approach is more of “reactive” method where the pages are physically added only during access faults.
  - Memory are released back to PVM page-by-page as in when they are released – ballooning concept.
  - Works on pages rather than memory blocks/chunks.
  - Needs special treatment for memory allocated for DMA access (dma_alloc_coherent).
  - More generic and widely used - https://en.wikipedia.org/wiki/Demand_paging

Ways of Resizing VM

- Demand Paging method by Google
  - Pages are physically added on demand during page access.
  - This approach is more of “reactive” method where the pages are physically added only during access faults.
  - Memory are released back to PVM page-by-page as in when they are released – ballooning concept.
  - Works on pages rather than memory blocks/chunks.
  - Needs special treatment for memory allocated for DMA access (dma_alloc_coherent).
  - More generic and widely used - https://en.wikipedia.org/wiki/Demand_paging

- **Dynamic Resizing using PSI Mechanism by Qualcomm**
  - Memory blocks are added upon increase in memory pressure (demand).
  - This approach is more “proactive” method where memory blocks are physically added before any memory access is made.
  - Memory is released back to PVM block-by-block when system has enough free memory.
  - Takes care of memory allocated for DMA access (memory is always present before any access is made).

Questions?

Thank you

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 and Snapdragon are trademarks or registered trademarks 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.

Follow us on:

For more information, visit us at qualcomm.com & qualcomm.com/blog