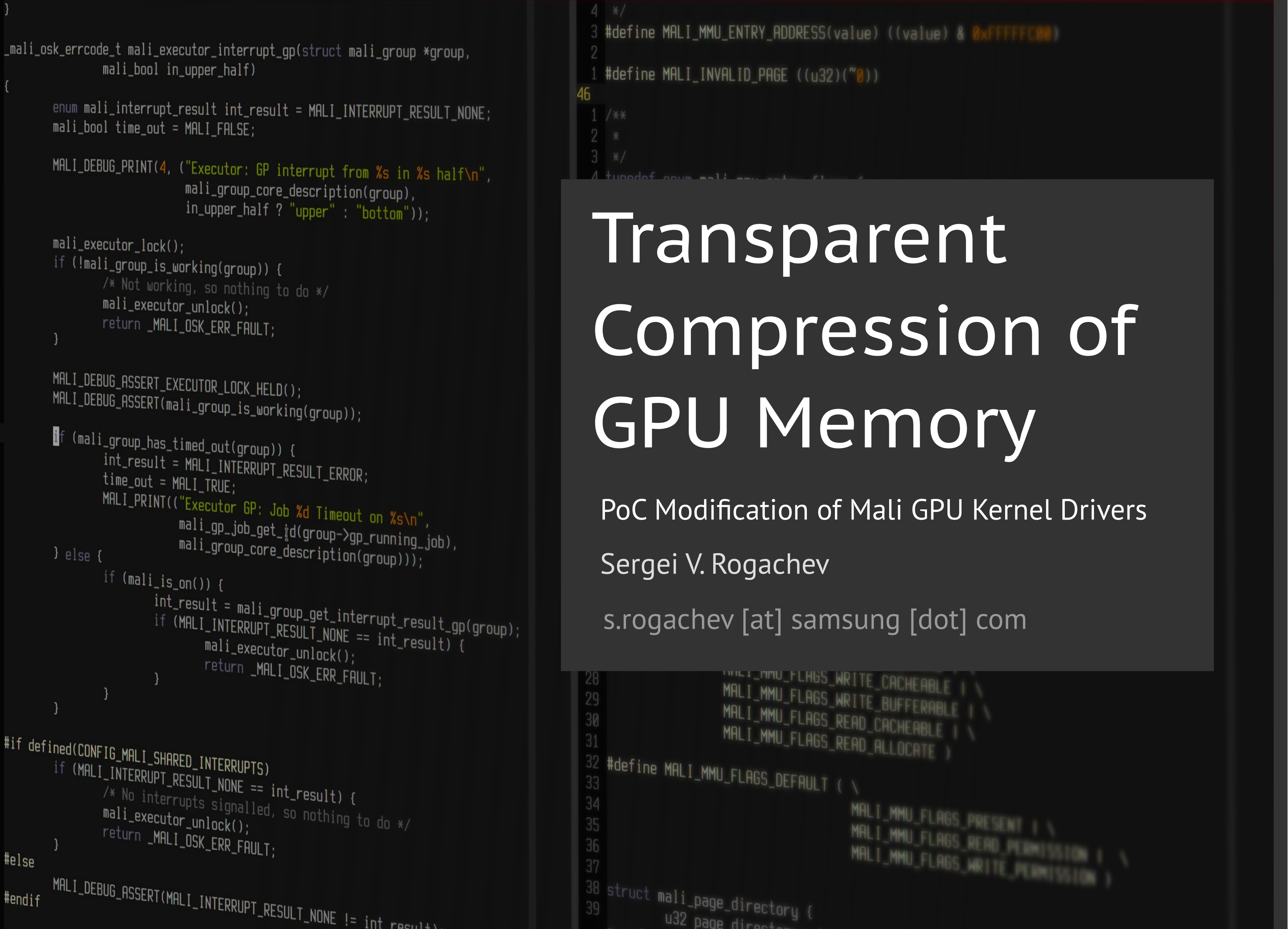


Transparent Compression of GPU Memory

PoC Modification of Mali GPU Kernel Drivers

Sergei V. Rogachev

s.rogachev [at] samsung [dot] com

Licensed under the terms of the Creative Commons license CC BY-NC-SA

Background

- Today almost every interactive device is equipped with graphics processing unit a.k.a. **GPU**:
  - Smart phones;
  - Tablet PCs;
  - Smart watches;
  - Smart TVs;
  - Set top boxes;

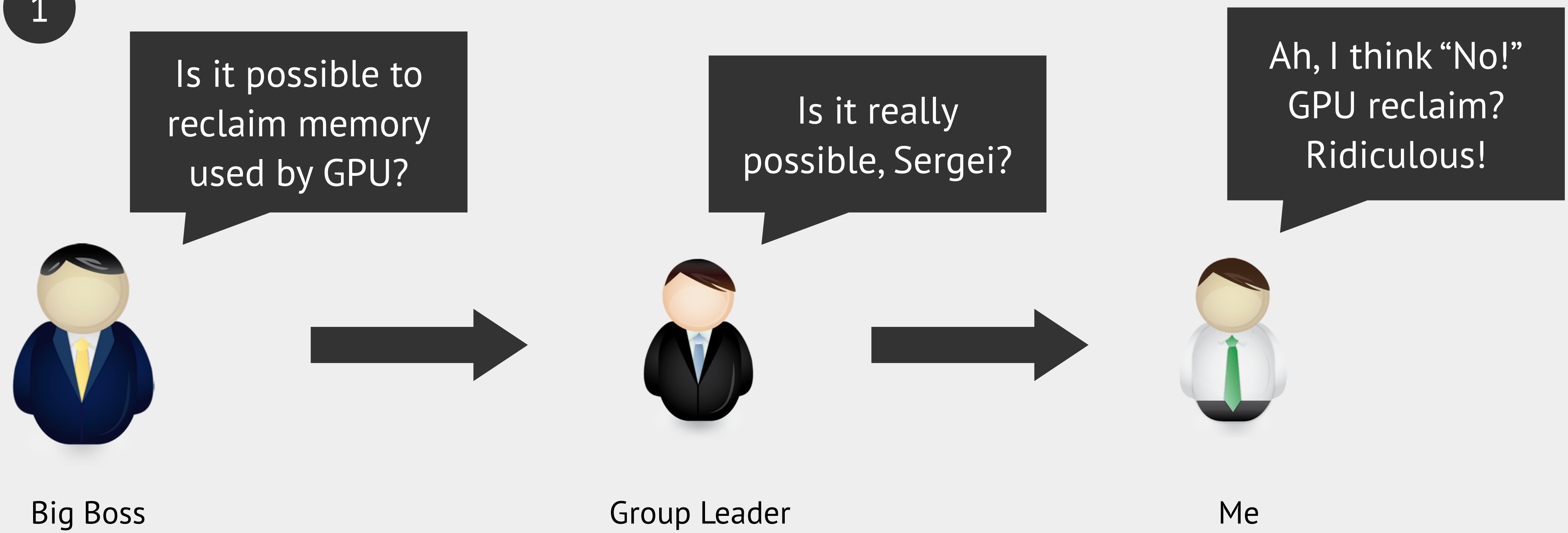
  … and even refrigerators…

Problem Proposition

- Mobile GPUs utilized in the mentioned devices **don't have dedicated memory** and use the systems' one.
- A GPU driver allocates memory for results of rendering, shaders and graphical primitives: textures, color buffers, tiler's buffers, etc.
- Thus, the **GPU driver consumes memory** that could be used by an operating system and user applications!

Preliminary Work

1

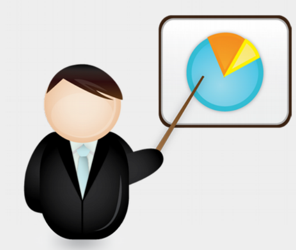


The idea of GPU memory compression/swapping looked strange and non-easily implementable. Honestly, we knew almost nothing about GPU drivers and graphical stack. We saw an example of a fail in the sphere of GPU memory paging:
Carmack, J. GPU data paging (2010)

Summer 2015

Preliminary Work

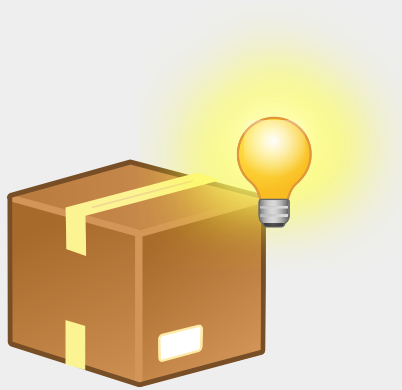
Almost at the same time a number of scientists from Korea participate in the **EMSOFT** conference with their implementation of a graphical buffers compression technique.
Kwon, S., Kim, S.-H., Kim, J.-S., and Jeong, J. Managing gpu buffers for caching more apps in mobile systems. Proceeding EMSOFT '15, 207–216 (2015)



Autumn 2015

Preliminary Work

3



A prototype
GPU SWAP

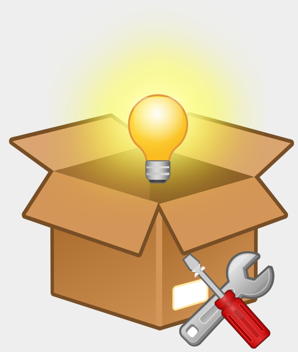
GPU SWAP – modification of a kernel GPU driver (Mali Midgard) implementing swapping of least recently used graphical memory.

Main characteristics:

- Reuses the swap facilities of the Linux kernel (overhead due to usage of block layer, necessity to modify the core kernel code);
- Implements own per-page LRU policy;
- Manually manages CPU and GPU mappings to pages of graphical memory;
- Uses ZRAM as a swap backend.

Current State

4



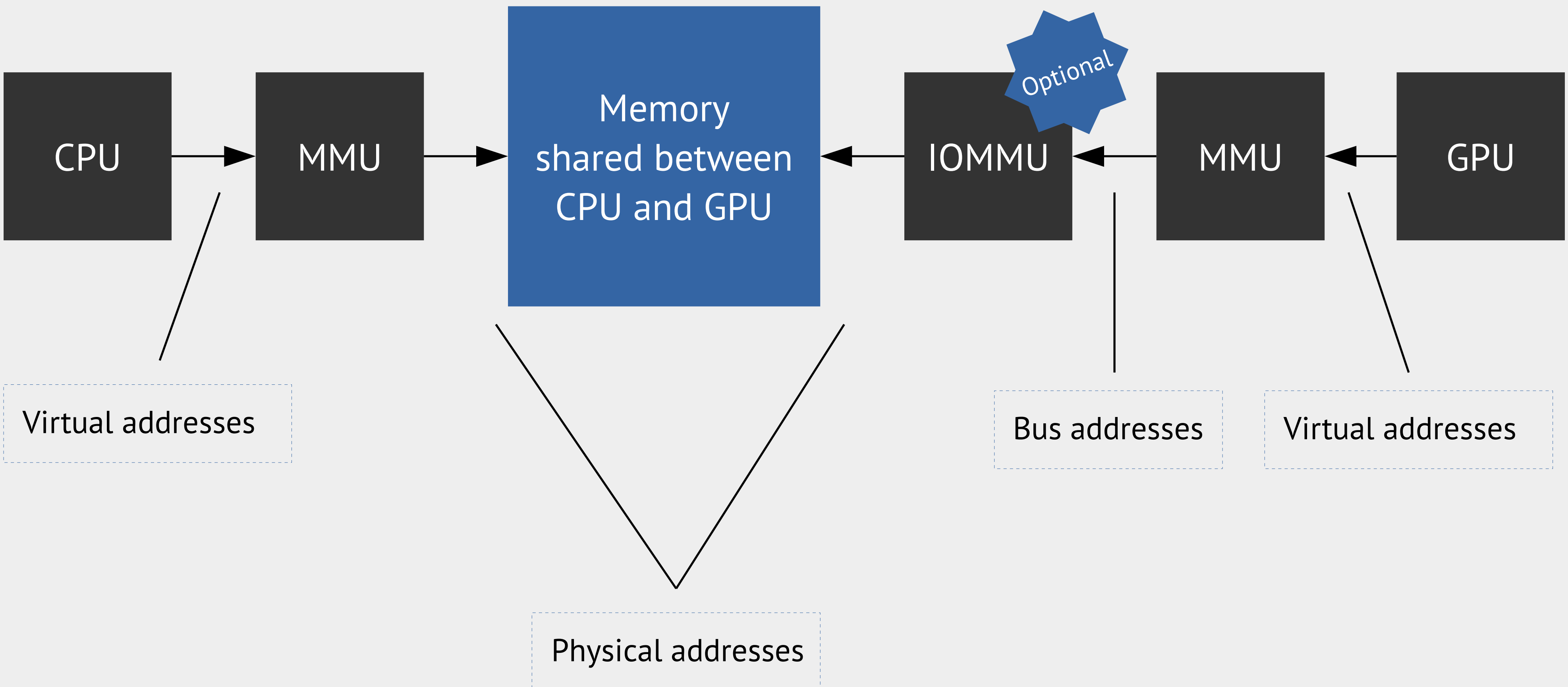
Transparent GPU memory compression

Transparent GPU memory compression – a PoC solution for Midgard and Utgard GPU kernel drivers for compression of temporary unused GPU memory.

Main characteristics:

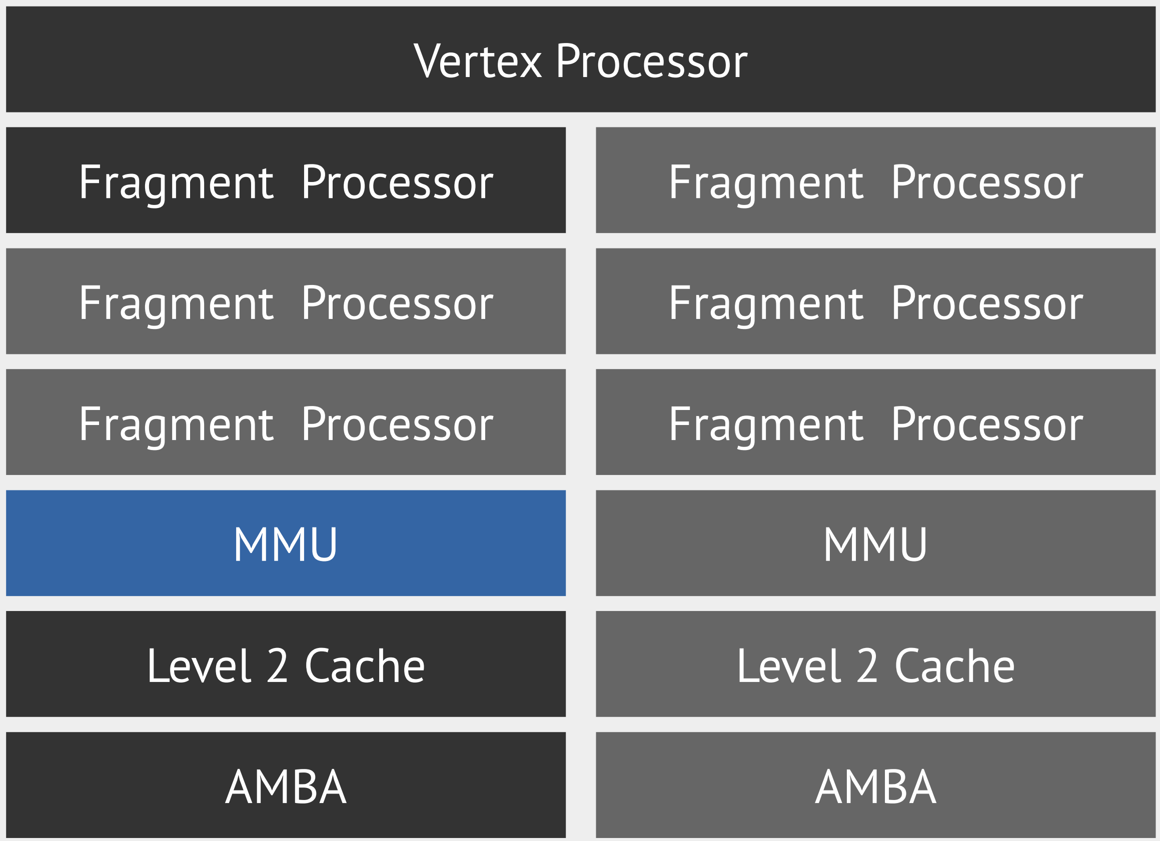
- Doesn't use the swapping facilities of the Linux kernel (smaller overhead, no core kernel modifications);
- Implements generic layer **GMC** with in-memory compressed storage based on **ZPOOL** and **CRYPTO COMP API**;
- Tries to keep modifications to the Mali kernel drivers code as little as possible.

Hardware Overview



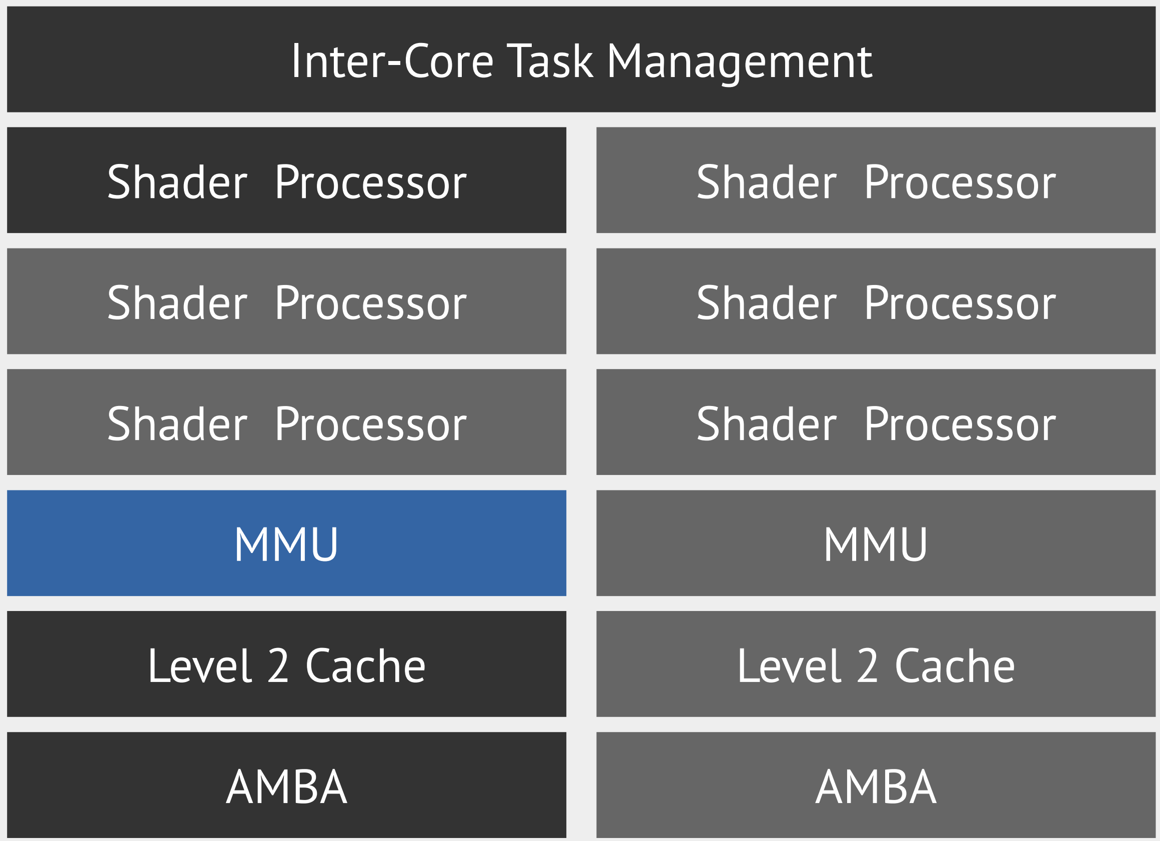
Hardware Overview

Utgard



Hardware Overview

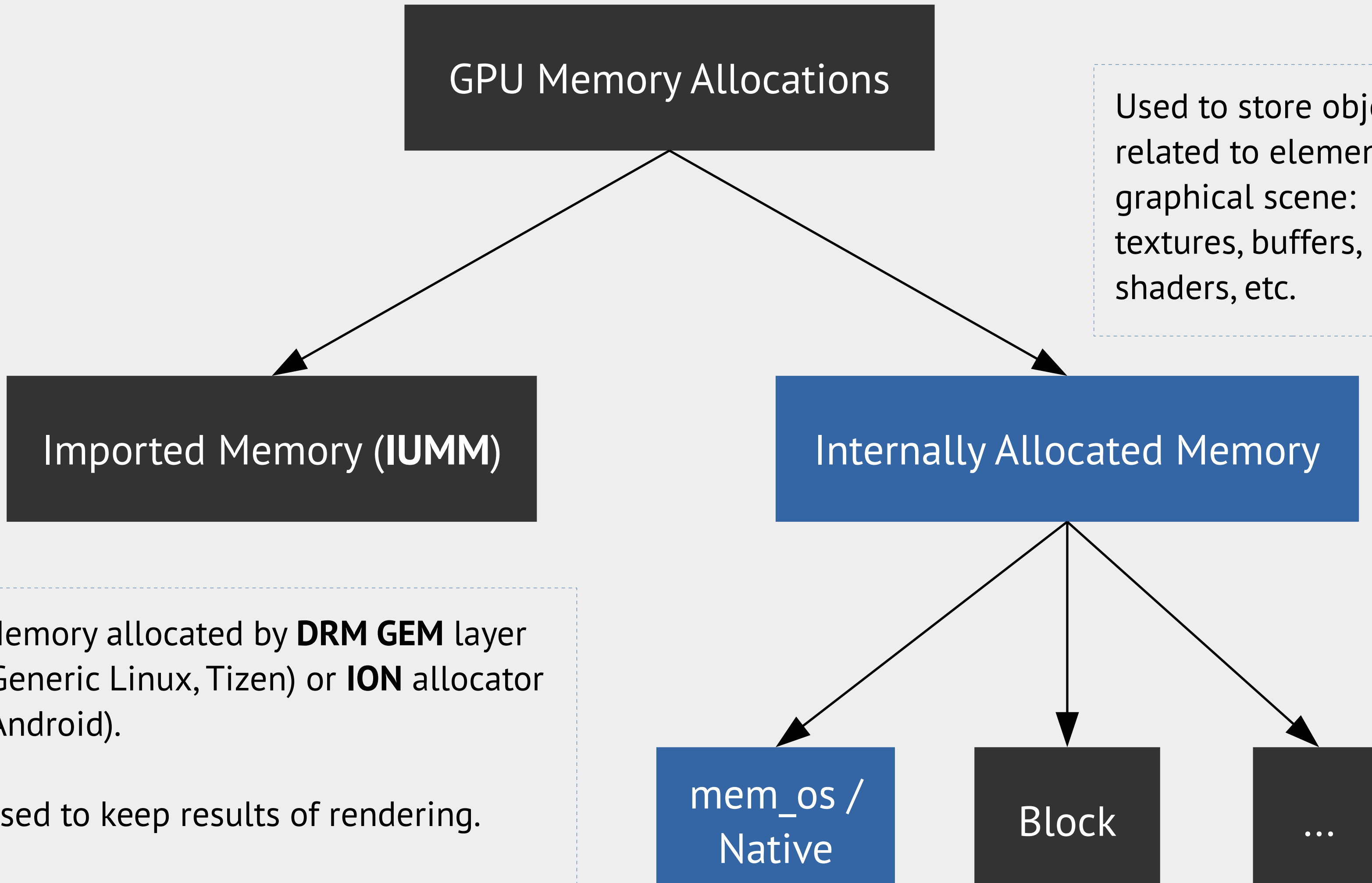
Midgard



Mali Driver: Memory Management

- GPU drivers are typically split to two parts: kernel driver (**mali.ko**) and user mode library (**libmali.so**).
- The user mode library initiates a session by opening the device file.
- The user mode library requests the kernel driver to allocate memory via IOCTL. The memory is mapped to GPU via GPU MMU.
- The memory is mapped to user space via mmap() system call.

Types of GPU Memory



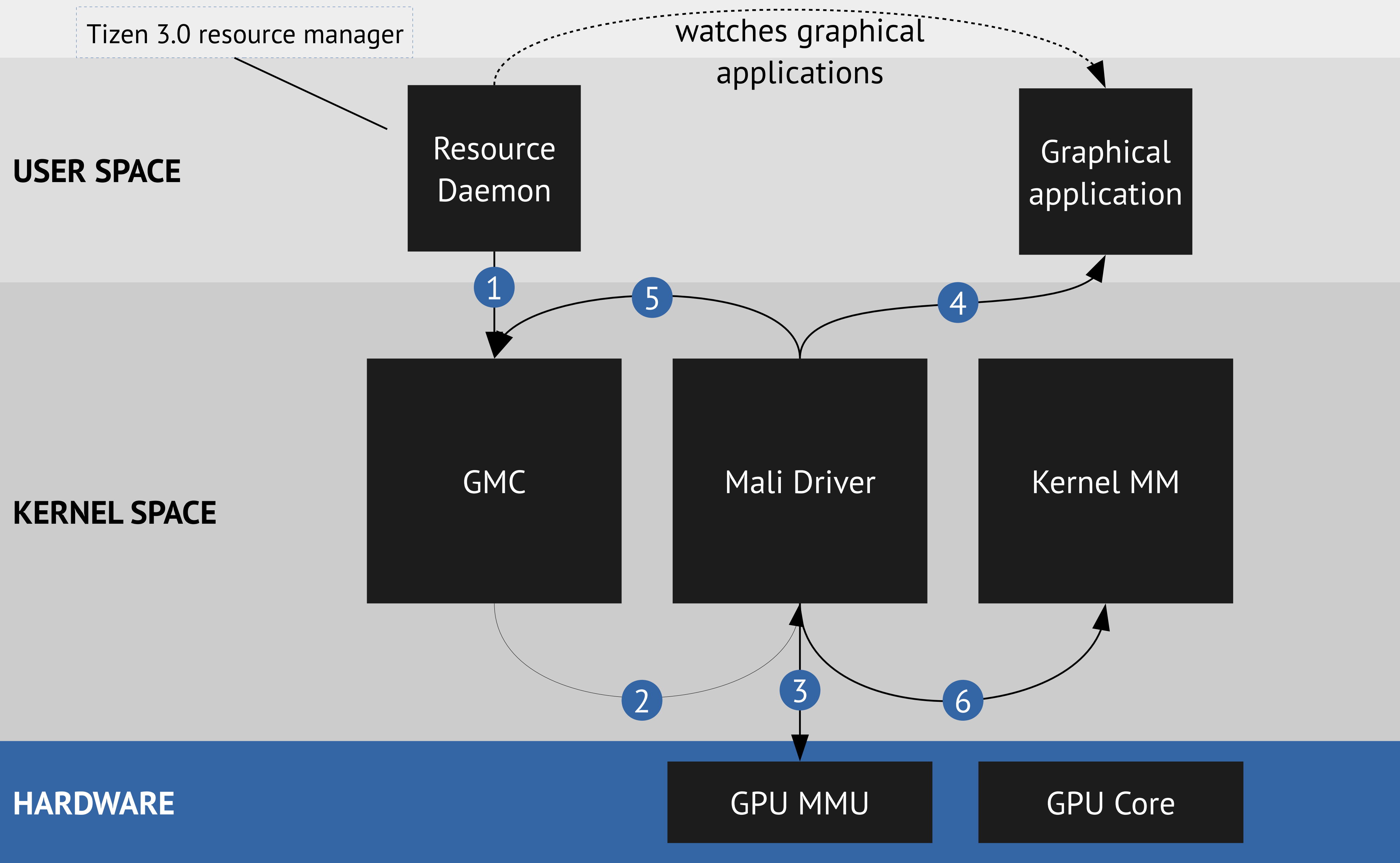
Used to store objects related to elements of graphical scene: textures, buffers, shaders, etc.

Memory allocated by **DRM GEM** layer (Generic Linux, Tizen) or **ION** allocator (Android).

Used to keep results of rendering.

Cleverly managed at the level of modern composite managers. Nothing to do here.

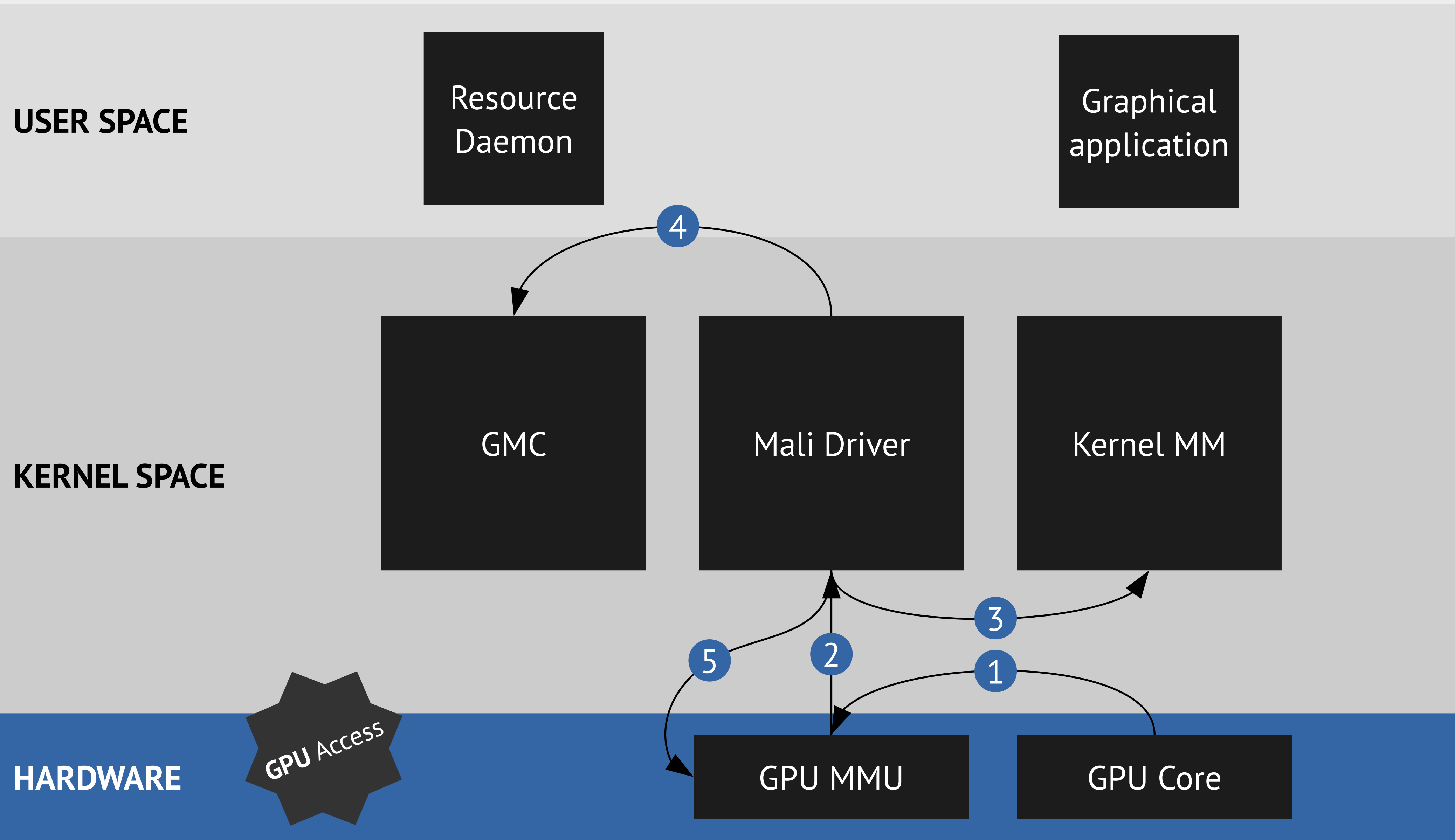
Operation Sequence: Compression



Operation Sequence: Compression

- Resource Daemon watches applications and notifies **GMC** layer when some application goes to background. ❶
- **GMC** translates the request to the GPU driver. ❷
- GPU driver performs unmapping of graphical memory pages of the process from GPU and CPU perspectives of view. ❸ ❹
- GPU driver passes pages to **GMC storage** to store them in compressed form. ❺
- GPU driver frees the pages via Linux MM API. ❻

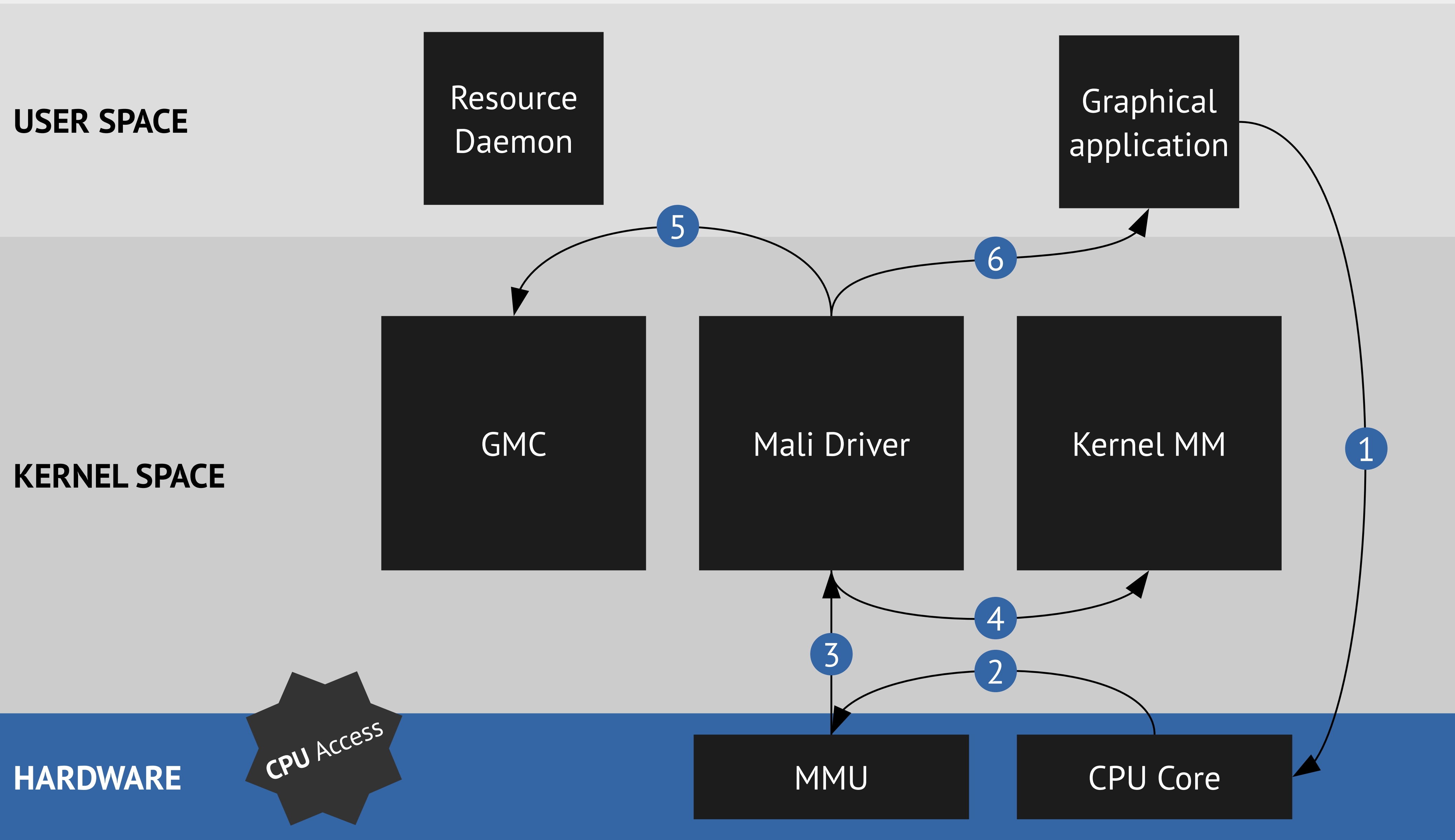
Operation Sequence: Decompression



Operation Sequence: Decompression

- Some **GPU job** accesses a memory address corresponding to a compressed page frame. 1
- **GPU MMU** signals the CPU about a page fault exception via interrupt, interrupt handler of the **GPU driver** is executed. 2
- The **GPU driver** tries to allocate a page frame using Linux MM API. 3
- The **GPU driver** requests the **GMC storage** to decompress page's data. 4
- The **GPU driver** maps the page to the corresponding GPU virtual address. 5

Operation Sequence: Decompression



Operation Sequence: Decompression

- Some **graphical application** is scheduled on a CPU core. ①
- Being executed on **CPU** the code accesses some memory address. ②
- **MMU** signals the **CPU** about a page fault exception (data abort), interrupt handler of the **GPU driver** is executed. ③
- The **GPU driver** tries to allocate a page frame using Linux MM API. ④
- The **GPU driver** requests the **GMC storage** to decompress page's data. ⑤
- The **GPU driver** maps the page to the corresponding **CPU** virtual address in the address space of the faulted process. ⑥

Internals

- **GMC (new)**
  - Infrastructure;
  - Storage;
  - Interface (debugfs).
- **GPU Driver (modified)**
  - Native / mem_os allocator;
  - GPU page fault handling;
  - CPU page fault handling.

Results

Characteristic	Value
Compression ratio (with zeroed pages)	6 – 9
Fair compression ratio	2.5 – 3
Saved memory	5 – 10%
LOC (generic)	~ 600
LOC (driver specific)	~ 500

Current Problems

- **GPU drivers are different internally**: often different versions of the same driver revision have notable differences in internals.
- GPU drivers are **designed** initially **without** any **reclaim facility** in mind.
- It is **difficult to develop** a more-or-less generic and **abstract layer/subsystem** for implementation of GPU memory compression in different GPU drivers.
- The solution is too far from Linux MM default mechanisms.

Current Problems

- The solution cannot be upstreamed easily:
  - Generic code looks **similar to ZRAM**, **ZSWAP**, **ZCACHE**, etc. The community is not interested in such code.
  - Driver specific part can be contributed to ARM Mali community through the support team. The process is difficult.
  - GPU compression/swapping/reclaim is not interesting for community in general.

References

- Kwon, S., Kim, S.-H., Kim, J.-S., and Jeong, J. Managing gpu buffers for caching more apps in mobile systems. Proceeding EMSOFT '15, 207–216 (2015)
- Carmack, J. GPU data paging, http://media.armadilloaerospace.com/misc/gpuDataPaging.htm (2010)
- Dominé, S. Using texture compression in OpenGL, NVIDIA Corporation (2000)
- Dae, I. DRM Driver Development for Embedded Systems, Embedded Linux Conference (2011)
- Ziv, J. and Lempel. A. A universal algorithm for data compression, IEEE trans. on information theory, IT-23, No 3, 337–343 (1977)
- Tanenbaum, A. Operating Systems Design and Implementation, Third Edition (Prentice Hall, 2006)
- Gorman, M. Understanding the Linux Virtual Memory Manager (Prentice Hall PTR, 2004)
- Corbet, J. Transcendent memory. Linux Weekly News (2009)
- Magenheimer, D. In-Kernel Memory Compression. Linux Weekly News (2013)
- Prodduturi. R. Effective Handling of Low Memory Scenarios in Android, Indian Institute of Technology (2013)

Acknowledgement

- **Dmitry Safonov** – developed the first GPU SWAP prototype.
- **Alexander Yaschenko** – develops the current version of transparent GPU memory compression on Midgard.
- **Krzysztof Kozlowski** – developed the GEM memory compression prototype.

Thank you!

Questions?