Support for `VM_BIND` and sparse textures in Freedreno

Connor Abbott, Valve Corporation
Rob Clark, Qualcomm Technologies Inc.

Memory Management History

- "Memory management": how to allocate, free, and assign addresses to GPU memory
  - Buffer Objects (BOs) in the kernel and Mesa, `VkDeviceMemory` in the Vulkan API
  - BO Addresses are often called "iova"s (I/O Virtual Address)
  - Can also map BOs in userspace to share memory with the GPU
- Main complication/antagonist: buffer eviction
  - Need to evict unused GPU memory to swap when memory is tight
  - Critical for user experience on shipping Chromebooks!

Memory Management History: Softpin

- Userspace (turnip) has an address space per DRM FD
- Userspace allocates a GEM BO and then queries its address via `GET_IOVA`
- Kernel in charge of address space allocation
- Job submission requires a *submit list*
  - List of BOs used by the submission
  - Anything not used can be evicted
  - Submit commands/IBs reference the BO via its index in the submit list
  - Jobs take a reference on the BOs in the submit list
- But, various Vulkan features mean turnip cannot know which BOs are used!
  - Especially `VK_KHR_buffer_device_address`
  - So... put everything in the submit list!
  - Kernel iterates and locks *every* BO on *every* submit

Memory Management History: SET_IOVA

- Various usecases for userspace control of the address space:
  - Replayability in `VK_KHR_buffer_device_address`
  - Virtualization via VirtIO
- So: `SET_IOVA` ioctl
- Call it instead of `GET_IOVA` immediately after BO creation
- Simple to implement, right?!?
  - No :(

Memory Management History: Necromancy

- Deleting a BO does not actually delete it
  - The kernel takes a reference to the BO for each submit that uses it
  - The BO is not freed until each already-submitted job finishes
- There is no way to remove the BO from its iova without deleting it
- When we delete a BO, it may become a *zombie*:
  - Vulkan user thinks it is dead
  - But the kernel doesn't trust it and keeps it alive!

Memory Management History: Necromancy

- Maintain a list of zombie BOs
- When allocating a new BO:
    - Try allocating without a zombie
    - Check if any zombie BO can be freed
    - Finally, stall waiting for zombies to be freed
- Massive complexity!

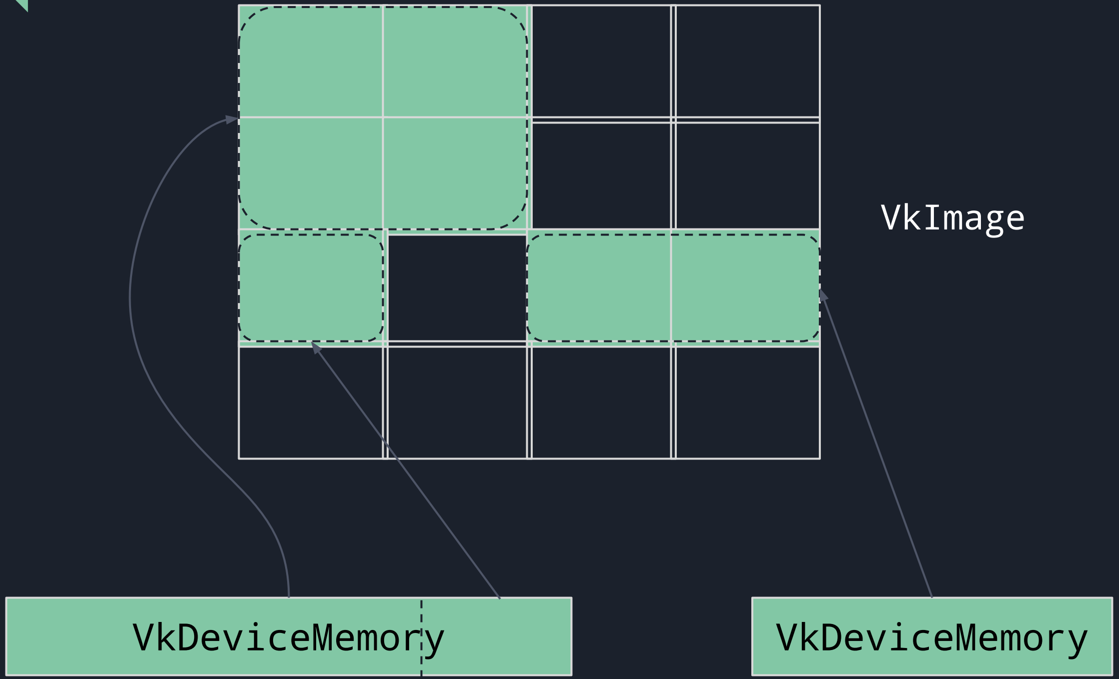
There must be a better way...

In the meantime… sparse textures!

Sparse Textures

- Some games want to use absolutely massive textures
- Larger than what would fit into memory
- Swap parts in and out of memory
- Take advantage of *image tiling*

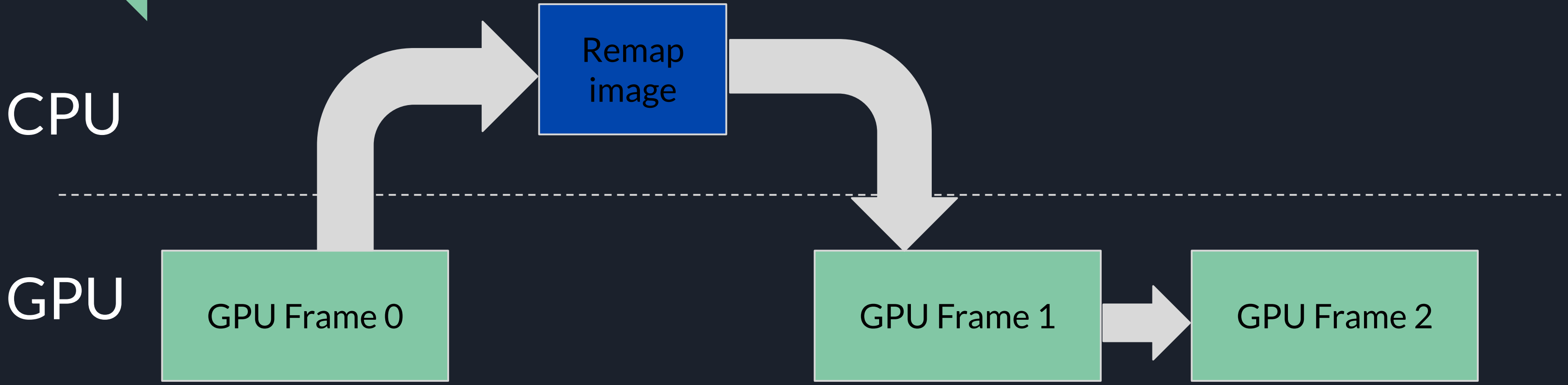
Sparse Textures



Sparse Textures

- Mipchains are supported
    - Miplevels smaller than the tile size are part of the *miptail*
    - Miptail must be allocated together
- What do unmapped tiles return?
    - With `sparseResidencyStrict`: must return 0

Sparse Textures



- Earlier jobs must not see memory mapped afterwards
- Later jobs must see the mapped memory
- Mapping must be done *on the device timeline!*

Sparse Textures

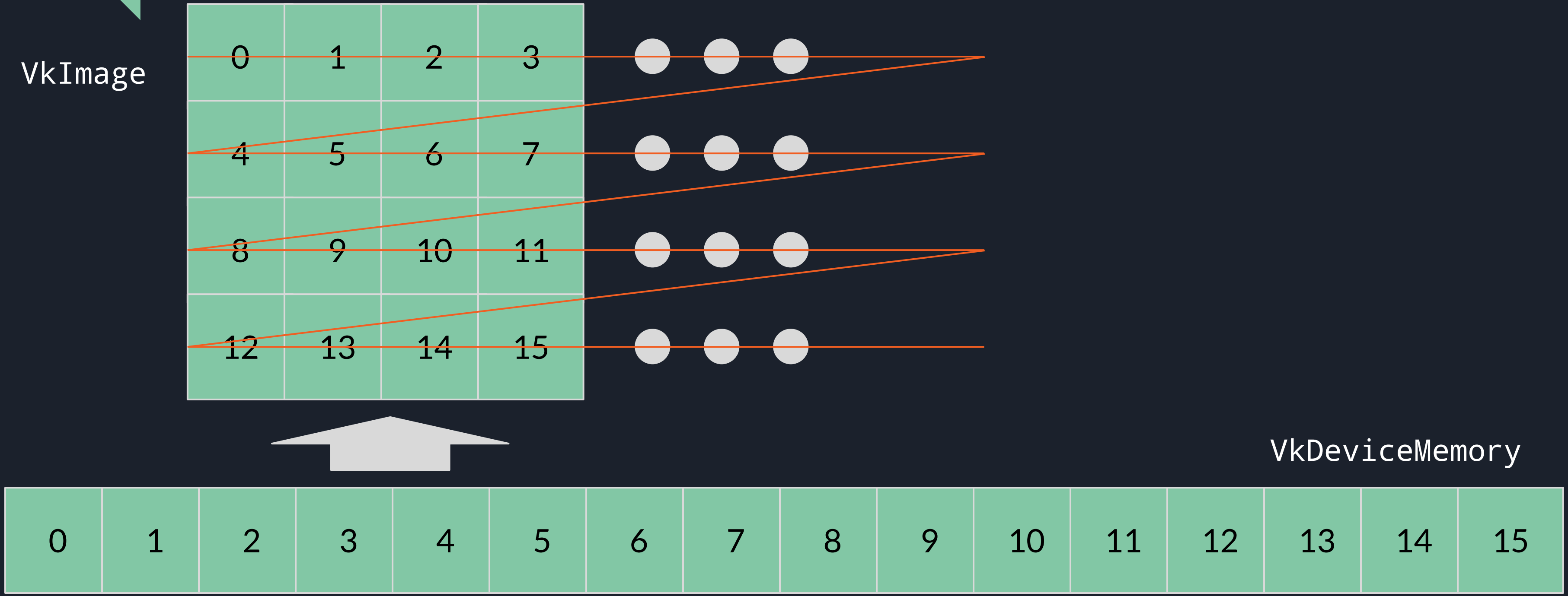
- New queue submission command: `vkQueueBindSparse()`
- Normally executed on the CPU
- However, on the critical path between submits
  - Taking too long stalls the GPU!
- Maps and unmaps image tiles
- Also supports *sparse buffers*
- The only way to map and unmap sparse images/buffers

Enter… VM_BIND!

Sparse Textures on Adreno

- Standard tile size: 64K bytes
  - Size in pixels defined by Vulkan & D3D specs
- Largest native tile is the *macrotile*: 4K bytes
- How to implement 64K standard sparse tiles?
- Fake it till you make it!

Sparse Textures on Adreno



Sparse Textures on Adreno

- Have to deal with *bank swizzling*
  - Swaps the macrotile order within a row
  - Deals with DDR bank access conflicts
- Have to deal with partial tiles