DMA-BUF Heaps

Linux user-space device buffer allocation and its uses

Embedded Linux Conference North America 2020

Andrew Davis < afd@ti.com / andrew.davis@linaro.org >

Texas Instruments

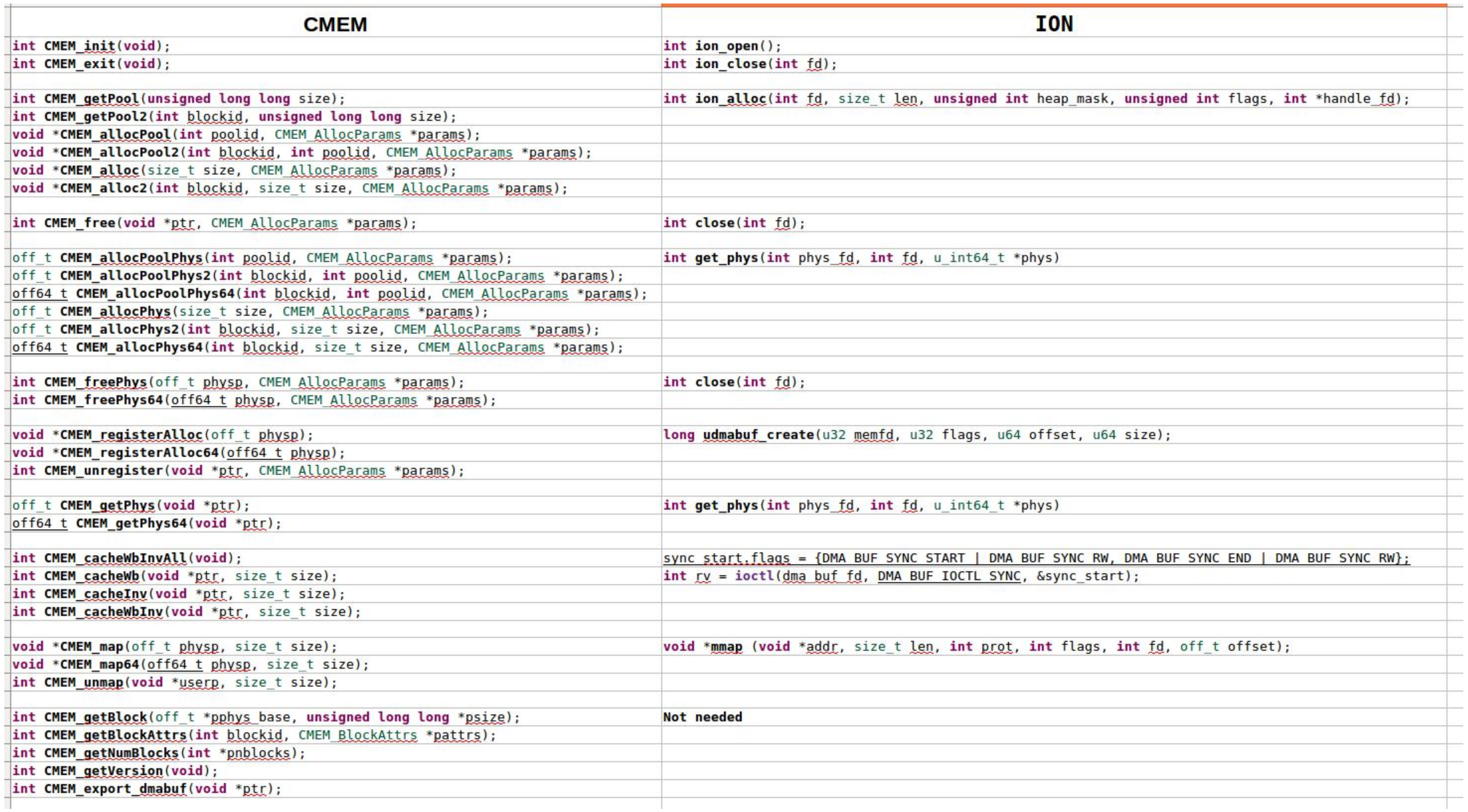
What are we trying to solve

- Frameworks that operate on memory buffers developed independently
  - DRM, V4L2, RemoteProc, TEE
- DMA-BUFs created as a way for these devices to trade buffers between frameworks as directed by userspace
- Memory areas themselves often not a part of the device or framework that exports them
- Might even be common CMA or normal system memory we want to work with, but we still need to pick a exporting device framework
- Non-standard memory would have to be shoehorned into one of the above DMA-BUF exporting frameworks
- The exporting framework is responsible for ensuring all buffer attaching devices can interoperate
- Several central device memory allocators exist, can we unify their use-cases under one API?

CMEM

- Texas Instruments specific allocations, we are not alone, NVIDIA had "NVMAP", Qualcomm had "PMEM"
- Memory areas described by module parameters or with DT
  - modprobe cmemk phys_start=0xC2200000 phys_end=0xC3200000 pools=1x5250000,3x1048576,3x829440,1x256000,4x131072
  - phandles to reserved memory nodes
- Everything and the kitchen sink API
  - Allocate from pools and blocks of memory based on ID
  - Explicit cache operations
- Expanded over the years, v2 APIs, 64bit support glued on
- Exposed physical addresses to userspace

CMEM to ION



CMEM	ION
`int CMEM_init(void);`	`int ion_open();`
`int CMEM_exit(void);`	`int ion_close(int fd);`
`int CMEM_getPool(unsigned long long size);`	`int ion_alloc(int fd, size_t len, unsigned int heap_mask, unsigned int flags, int *handle_fd);`
`int CMEM_getPool2(int blockid, unsigned long long size);`	
`void *CMEM_allocPool(int poolid, CMEM_AllocParams *params);`	
`void *CMEM_allocPool2(int blockid, int poolid, CMEM_AllocParams *params);`	
`void *CMEM_alloc(size_t size, CMEM_AllocParams *params);`	
`void *CMEM_alloc2(int blockid, size_t size, CMEM_AllocParams *params);`	
`int CMEM_free(void *ptr, CMEM_AllocParams *params);`	`int close(int fd);`
`off_t CMEM_allocPoolPhys(int poolid, CMEM_AllocParams *params);`	`int get_phys(int phys_fd, int fd, u_int64_t *phys)`
`off_t CMEM_allocPoolPhys2(int blockid, int poolid, CMEM_AllocParams *params);`	
`off64_t CMEM_allocPoolPhys64(int blockid, int poolid, CMEM_AllocParams *params);`	
`off_t CMEM_allocPhys(size_t size, CMEM_AllocParams *params);`	
`off_t CMEM_allocPhys2(int blockid, size_t size, CMEM_AllocParams *params);`	
`off64_t CMEM_allocPhys64(int blockid, size_t size, CMEM_AllocParams *params);`	
`int CMEM_freePhys(off_t physp, CMEM_AllocParams *params);`	`int close(int fd);`
`int CMEM_freePhys64(off64_t physp, CMEM_AllocParams *params);`	
`void *CMEM_registerAlloc(off_t physp);`	`long udmabuf_create(u32 memfd, u32 flags, u64 offset, u64 size);`
`void *CMEM_registerAlloc64(off64_t physp);`	
`int CMEM_unregister(void *ptr, CMEM_AllocParams *params);`	
`off_t CMEM_getPhys(void *ptr);`	`int get_phys(int phys_fd, int fd, u_int64_t *phys)`
`off64_t CMEM_getPhys64(void *ptr);`	
`int CMEM_cacheWbInvAll(void);`	`sync_start.flags = {DMA_BUF_SYNC_START | DMA_BUF_SYNC_RW, DMA_BUF_SYNC_END | DMA_BUF_SYNC_RW};`
`int CMEM_cacheWb(void *ptr, size_t size);`	`int rv = ioctl(dma_buf_fd, DMA_BUF_IOCTL_SYNC, &sync_start);`
`int CMEM_cacheInv(void *ptr, size_t size);`	
`int CMEM_cacheWbInv(void *ptr, size_t size);`	
`void *CMEM_map(off_t physp, size_t size);`	`void *mmap (void *addr, size_t len, int prot, int flags, int fd, off_t offset);`
`void *CMEM_map64(off64_t physp, size_t size);`	
`int CMEM_unmap(void *userp, size_t size);`	
`int CMEM_getBlock(off_t *pphys_base, unsigned long long *psize);`	**Not needed**
`int CMEM_getBlockAttrs(int blockid, CMEM_BlockAttrs *pattrs);`	
`int CMEM_getNumBlocks(int *pnblocks);`	
`int CMEM_getVersion(void);`	
`int CMEM_export_dmabuf(void *ptr);`	

ION History

- Introduced in Android ICS (4.0) 2011, in Linux drivers staging area by 2013
- Generic buffer sharing for multiple device pipelines
  - Trying to replace per SoC/GPU custom buffer sharing implementations
  - Address the issue of fragmented memory management interfaces across different Android devices.
- Motivated the development of DMA-BUFs upstream and became an early user
- Core API is basis for DMA-BUF heaps, in a way ION is now destaged (with new branding)

DMA-BUF

- A common mechanism to share memory buffers across different devices
- ION converted to produce DMA-BUF handles
- CMEM converted to produce handles converted to CMEM handles
- Userspace applications moving to use DMA-BUFs
  - OpenCL: cl_arm_import_memory_dma_buf
  - EGL: EGL_EXT_image_dma_buf_import
  - V4L2: V4L2_MEMORY_FD
  - DRM: DRM_IOCTL_PRIME
- As long as someone is giving out DMA-BUF buffers we can use them in more and more places
- But who should be given them out?

ION Issues

- Lots of legacy, was not clear which “ION” was in use
  - Compatibility was needed for the pre-4.12 ION
- As more functionally gets handled by DMA-BUF the less was needed in core ION
- ION handles replaced with DMA-BUF handles, usage tracking simplified
- Even though ION was in Staging, changes to the upstream API were met with (justified) resistance to prevent confusion and breaks of existing applications/userspace.
- Flags can change the allocation, some ION heaps supported this others did not, no guarantee what you are getting what what you asked for. (SRAM with cache flags still would return uncached memory cache ops could then break coherency).
- All one device file, limited permissions/security
- Trying to do too much in one interface (constraint solving, cache handling, etc)
- Pseudo-constraint solving interface
  - Bitfield of heap types you’d accept, and ION will allocate from one of those
  - Tried to handle all the cache management so vendors couldn't get it wrong
- Vendor extensions often modified that shared code, making them incompatible and impossible to upstream
- Fair amount of DMA-API abuse (would work on ARM, but no promises elsewhere)

DMA-BUF Heaps

- A framework for memory allocators to behave like devices that export buffers as DMA-BUFs without being part of a use-case specific framework.
- Focused on one feature from ION: Userland dma-buf allocation interface for various “types” of memory
  - Each heap driver is its own dmabuf exporter, so no need to modify core code for custom heap
  - Each heap driver gets its own chardev, no multiplexing heaps through one interface
- A vendor heap can have system level understanding of attaching devices
- Deal with system level coherency and cache management in the heap code, not the core
- Handle coordinating backing storage based on attached devices as DMA-BUF in one place

DMM/Tiler

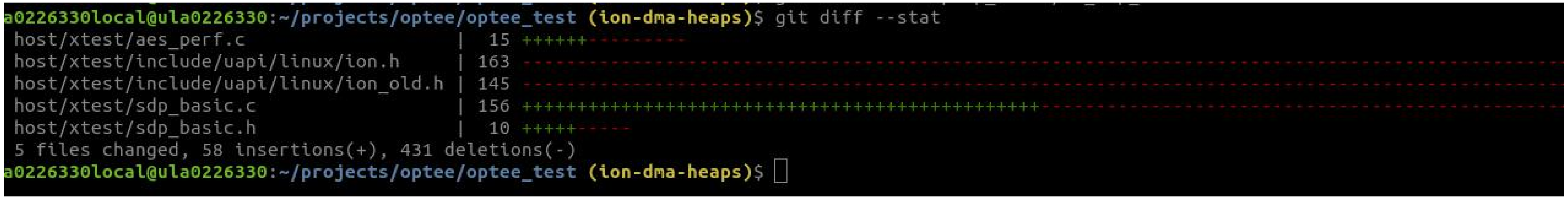
- IP available on several OMAP class devices, sits out by the external memory interface
- Allowed for hardware rotations of images in memory, but only when the image was written into memory through a special memory “window”
- Allowed for simple system-wide IOMMU/GART
- Linux support implemented through DRM framework
  - omap_bo_new_tiled()
- To use this memory outside of DRM one would allocate buffers and export
  - omap_bo_dmabuf()
- DRM framework had to be used to access this device even when the buffers were not used by the DRM framework
- If exported as a DMA-BUF Heap device then OMAP specific buffer handling could be removed from libdrm

SRAM

- SRAM locations distinct from DRAM in address range.
- SRAM usually outside of normal kernel page mappings, located somewhere in device memory area.
- Not cached, it's not needed; on TI K3 SRAM and L3 Cache are build out of the same on-chip memory.
- Existing userspace exporting, /sysfs/xx/7000000.sram, uses read()/write() fops, no mmap()
- Device drivers can
  - of_gen_pool_get()
  - gen_pool_alloc()
  - gen_pool_virt_to_phys()
- Exporting SRAM regions as DMA-BUF Heaps handles all the above
- Open questions on how this works with the existing userspace/driver exporting, should they be exclusive?
- Patch posted: https://lore.kernel.org/lkml/20200424222740.16259-1-afd@ti.com/

Secure buffers

- Firewalls may need to be defined statically at boot-time
  - These would be memory carveouts in DT (no-map;)
- Dynamic firewalls can change properties based on the set of devices using them
  - Pages removed from Linux allocators if needed
  - Lock/unlock firewall read/write at DMA-BUF attach/map
  - Deny at attach time to prevent firewall exceptions
- OP-TEE xtest suite converted to DMA-BUF Heaps from ION (patches pending)



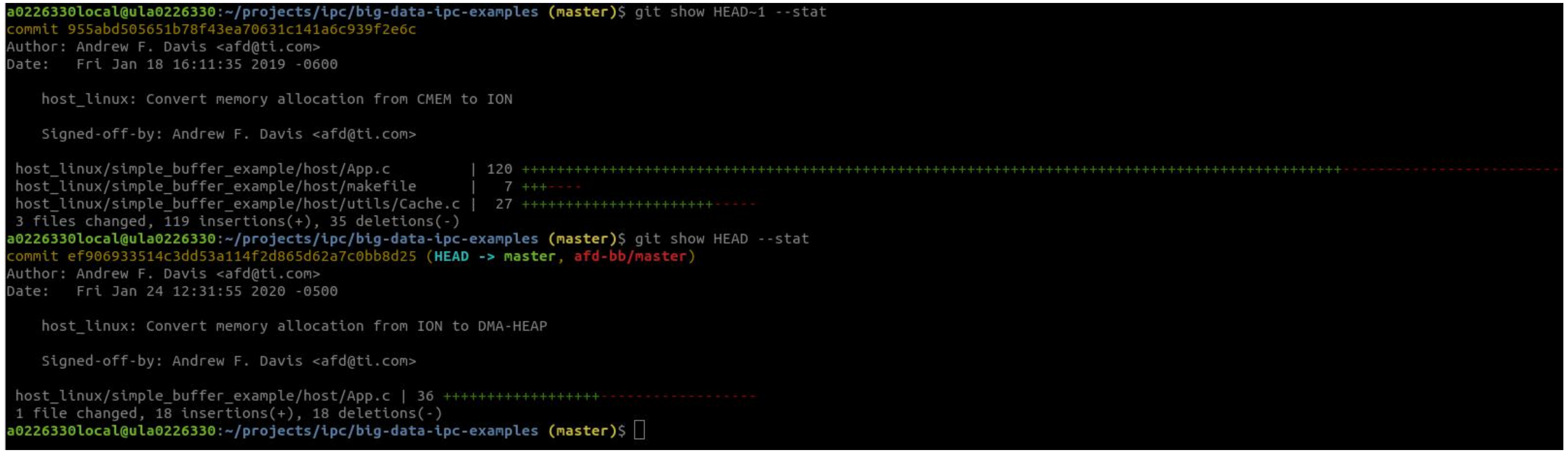
```
a0226330local@ula0226330:~/projects/optee/optee_test (ion-dma-heaps)$ git diff --stat
 host/xtest/aes_perf.c                      |  15 ++++++---------
 host/xtest/include/uapi/linux/ion.h        | 163 ----------------------------------------------------------------------------------------------
 host/xtest/include/uapi/linux/ion_old.h    | 145 ----------------------------------------------------------------------------------------------
 host/xtest/sdp_basic.c                     | 156 +++++++++++++++++++++++++++++++++++++++++++++++-----------------------------------------------
 host/xtest/sdp_basic.h                     |  10 +++++-----
 5 files changed, 58 insertions(+), 431 deletions(-)
a0226330local@ula0226330:~/projects/optee/optee_test (ion-dma-heaps)$ 
```

TI OpenCL

- Allows for zero-copy buffer sharing
  - void *__malloc_ddr(size_t size)
  - void *__malloc_msmc(size_t size)
  - clCreateBuffer with CL_MEM_USE_HOST_PTR
  - Backend was all CMEM
- Now we can avoid these by allowing userspace to perform the allocation and passing the DMA-BUF into the runtime using standard API extensions (cl_arm_import_memory_dma_buf)
- Now uses DMA-BUF Heaps on 64bit platforms, all platform support in works
- Cache operation with clCreateSubBuffer still an issue
  - DMA-BUF allows operations on the whole buffer
  - Sub-buffers need to be copied out to another DMA-BUF for fine-grain synchronization

Remoteproc subsystem physical addresses

- TI Keystone2 systems require large physically contiguous buffers to load remote processor firmwares (+2GB)
- Buffers passed to the remote processor as part of remote function calls
  - Applications should only need to pass in a DMA-BUF handle (and optional offset)
  - These need to arrive at the rproc as a physical address
  - Linux needs to do the conversion and setup any IOMMUs
  - Supported in TI evil vendor tree, still a gap upstream
- Up to the exporter to determine when a buffer can be used by a core
- TI buffer exchanging example code converted to DMA-BUF Heaps



```
a0226330local@ula0226330:~/projects/ipc/big-data-ipc-examples (master)$ git show HEAD~1 --stat
commit 955abd505651b78f43ea70631c141a6c939f2e6c
Author: Andrew F. Davis <afd@ti.com>
Date:   Fri Jan 18 16:11:35 2019 -0600

    host_linux: Convert memory allocation from CMEM to ION

    Signed-off-by: Andrew F. Davis <afd@ti.com>

 host_linux/simple_buffer_example/host/App.c         | 120 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++-------------------------
 host_linux/simple_buffer_example/host/makefile      |   7 +++----
 host_linux/simple_buffer_example/host/utils/Cache.c |  27 ++++++++++++++++++++++-----
 3 files changed, 119 insertions(+), 35 deletions(-)
a0226330local@ula0226330:~/projects/ipc/big-data-ipc-examples (master)$ git show HEAD --stat
commit ef906933514c3dd53a114f2d865d62a7c0bb8d25 (HEAD -> master, afd-bb/master)
Author: Andrew F. Davis <afd@ti.com>
Date:   Fri Jan 24 12:31:55 2020 -0500

    host_linux: Convert memory allocation from ION to DMA-HEAP

    Signed-off-by: Andrew F. Davis <afd@ti.com>

 host_linux/simple_buffer_example/host/App.c | 36 ++++++++++++++++++------------------------
 1 file changed, 18 insertions(+), 18 deletions(-)
a0226330local@ula0226330:~/projects/ipc/big-data-ipc-examples (master)$ 
```

Android Gralloc

- Buffer backing source determined by GRALLOC_USAGE_* bitmask
- Handles constraint solving based on system and devices set at allocation time
- In our case almost always uses DMA-BUF CMA Heap
- Rather trivial as old backing was ION

Cache Management

- Some software uses consumer/producer patterns causing cleanup and unmap to happen at each stage of buffer life
- Might not be possible to know if buffer will be reused
- Causes lots of often unneeded cache management operations
- Should this be handled by the exporter or can we make this generic in the DMA-BUF or DMA-API layer?
- Great write-up available here:
  - https://lwn.net/Articles/822052/
  - https://lwn.net/Articles/822521/

Whats next

- Heaps accessed as DevFS device files
  - We are currently hard-coding file name
  - Generic query for buffer properties (cached, physically contiguous, secure, etc.)
  - System configuration file (fstab, interfaces)
  - Gralloc like userspace API, specify usage at allocation time and let it pick the right backing
- Remaining legacy allocators and ION users
  - Some bits of our gstreamer solution
  - Binary blob GPU drivers
  - Tutorials and reference material needs updating

Missing bits

- Are we ready for `# rm -r linux/drivers/staging/android/ion/` ?
- Deferred buffer freeing
  - Buffers can be schedules to be freed later, saves some time if instantly re-allocated
  - Can schedule zeroing pages out until system is idle
- DebugFS stats
  - Are DMA-BUF stats enough?
  - Should we report available heap space left?
- Your heaps!

References and Acknowledgments

- The Android ION memory allocator (https://lwn.net/Articles/480055/)
- Introduce DMA buffer sharing mechanism (https://lwn.net/Articles/473668/)
- DMA Buffer Sharing Framework: An Introduction (https://elinux.org/images/a/a8/DMA_Buffer_Sharing-_An_Introduction.pdf)
- DMA-BUF Heaps (destaging ION) (https://lwn.net/Articles/806219/)
- DMA-BUF Developments (https://static.linaro.org/connect/san19/presentations/san19-415.pdf)

- John Stultz <john.stultz@linaro.org>
- Sumit Semwal <sumit.semwal@linaro.org>
- Laura Abbott <labbott@redhat.com>