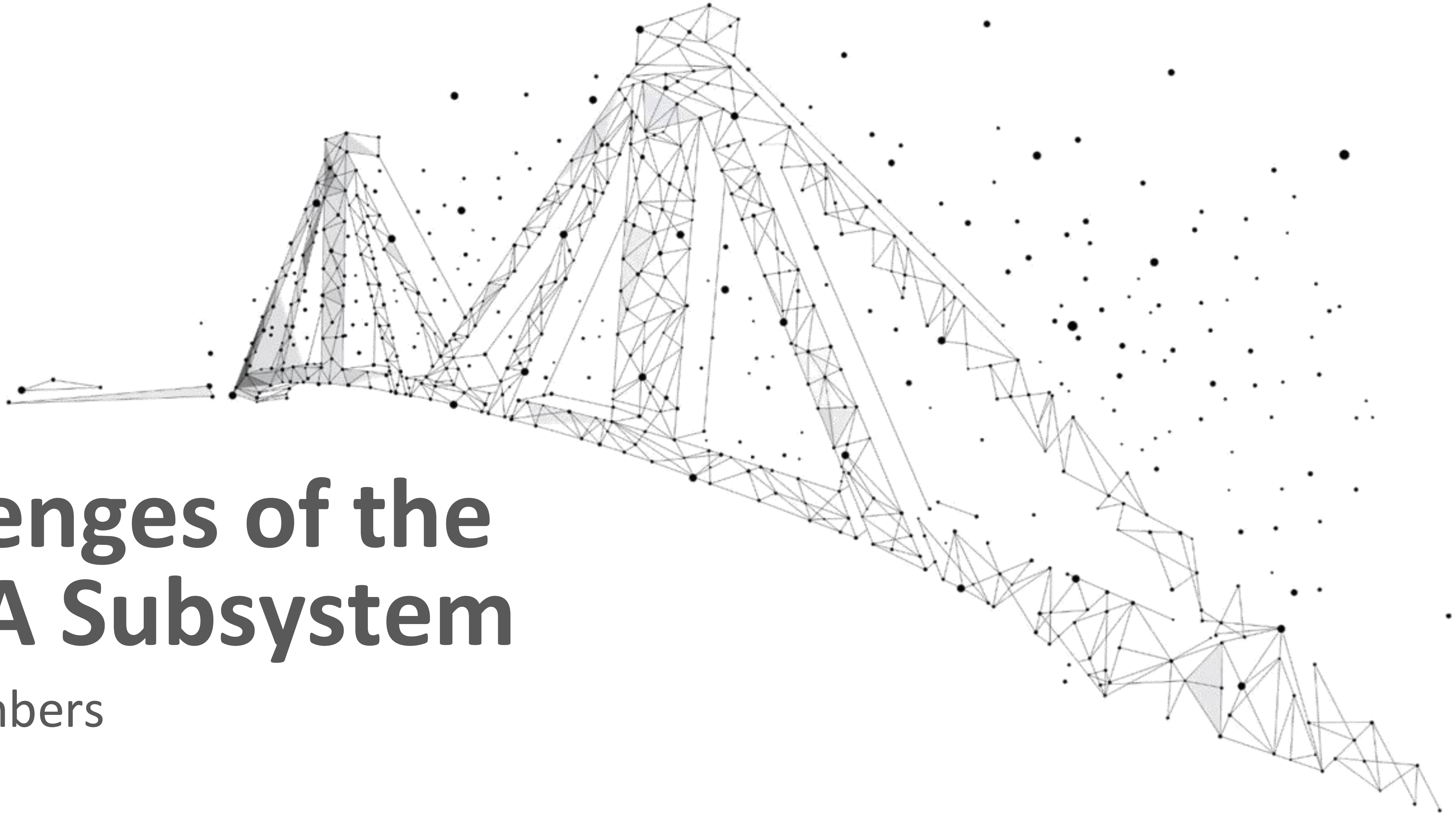


Challenges of the RDMA Subsystem

Linux Plumbers

Sept 2019

What is RDMA

- In the Linux kernel: `drivers/infiniband`, ~540 files, ~400kloc.
  Historically a top 10 subsystem by patch volume

- Evolved from VIA in the 1990's
  What we now think of as RDMA became standardized as InfiniBand around 2000 (RDMA over IB)
- Core IB 'verbs' software interface was largely integrated into IETF's iWarp (RDMA over TCP)
- IB itself evolved into RoCE (RDMA over Converged Ethernet)

- Other technologies started to use '`drivers/infiniband`' for limited forms of RDMA
  Sometimes called 'fabrics' now
- Today we see all sorts of stateful offload networking schemes living in RDMA

Elevator Pitch

- Networking technology
- OSI layer 5, like TCP
- Hardware offload
- Zero Copy data transfer
- High network bandwidth

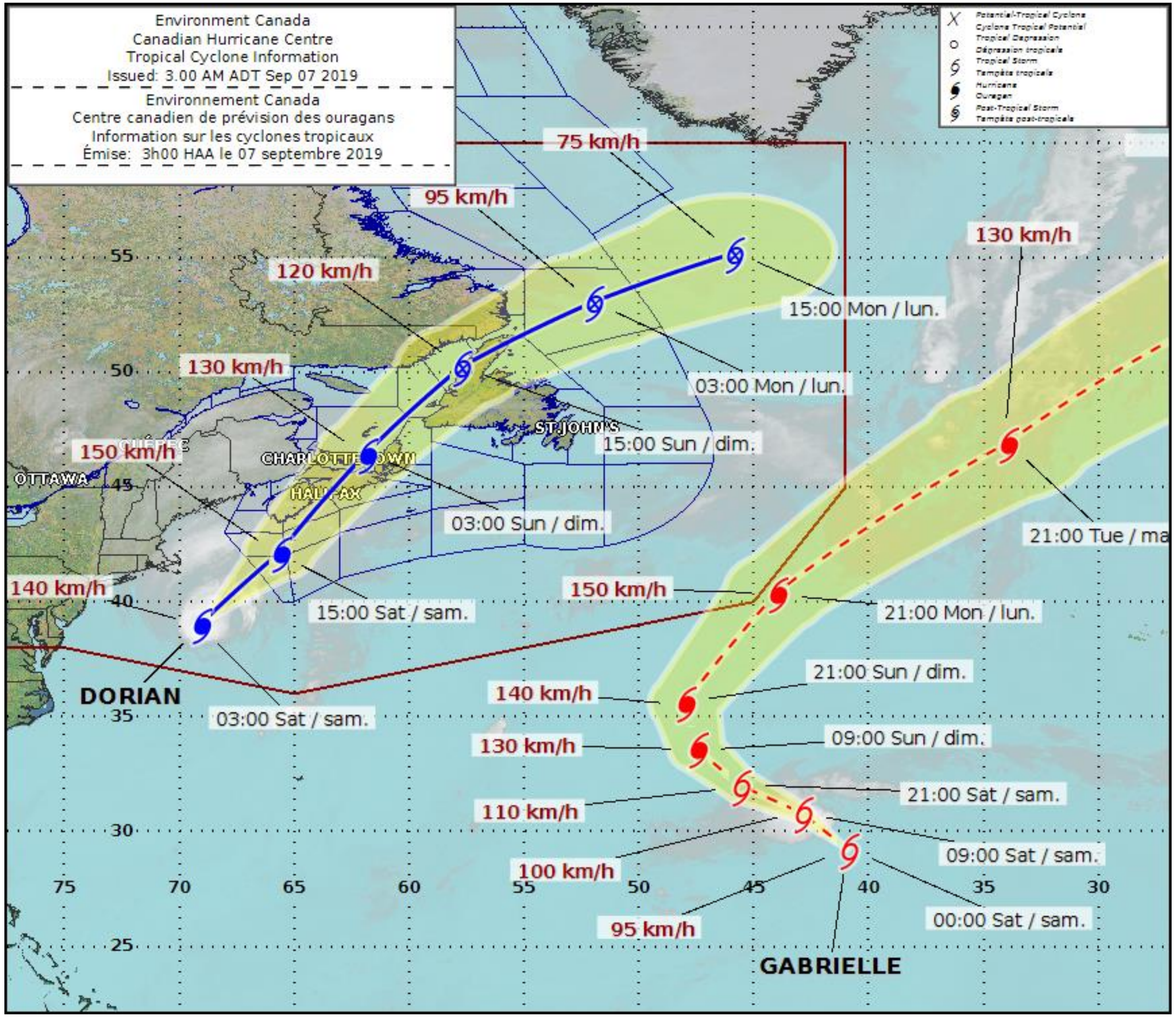
Why does it exist?

RDMA technologies have consistently given a 5x-10x bump in network performance compared to classical networking

- Bandwidth:
  - RDMA offload excels at high bandwidth single stream applications with low CPU usage
  - At inception, 10Gbit/s IB greatly outpaced classic TCP
    Today a RDMA card can drive 200G, single stream, single CPU
- Latency:
  - Kernel bypass makes the small message latency much lower than TCP
  - Gap is increasing as kernel syscalls become more expensive due to security
- Scale:
  - Many RDMA technologies have a non-Ethernet link layer that can scale upwards of 10,000's of nodes on a single network
  - Elaborate networks give very good node-to-node performance, not just node-to-router

Use Case Examples

- High Performance Technical Computing (HPC) aka ‘super computing’
  - Large scale physics/chemistry/weather simulation
  - Top machine today is Summit: 2.5 million PPC cores drawing 10MW running Linux using RDMA
  - Make the world a better place type of stuff
- Enterprise Clustering
  - Most commercial databases cluster and scale using RDMA
  - Some VM environments cluster with RDMA for migration and storage
  - Storage connect via iSCSI over RDMA or NVMe over RDMA (aka NVMeoF)
- Hyperscale
  - Distributed storage backplane
  - Application RPC
  - HPC-like applications (ie AI training, cloud HPC, etc)


Environment Canada
Canadian Hurricane Centre
Tropical Cyclone Information
Issued: 3.00 AM ADT Sep 07 2019
Environnement Canada
Centre canadien de prévision des ouragans
Information sur les cyclones tropicaux
Émise: 3h00 HAA le 07 septembre 2019
Potential-Tropical Cyclone
Cyclone Tropical Potential
Tropical Depression
Dépression tropicale
Tropical Storm
Tempête tropicale
Hurricane
Ouragan
Post-Tropical Storm
Tempête post-tropicale
75 km/h
95 km/h
120 km/h
130 km/h
150 km/h
140 km/h
15:00 Mon / lun.
03:00 Mon / lun.
15:00 Sun / dim.
03:00 Sun / dim.
15:00 Sat / sam.
03:00 Sat / sam.
DORIAN
ST.JOHN'S
CHARLOTTETOWN
HALIFAX
QUEBEC
OTTAWA
130 km/h
21:00 Tue / ma
150 km/h
21:00 Mon / lun.
21:00 Sun / dim.
140 km/h
130 km/h
09:00 Sun / dim.
21:00 Sat / sam.
110 km/h
09:00 Sat / sam.
100 km/h
00:00 Sat / sam.
95 km/h
GABRIELLE
55
50
45
40
35
30
25
75
70
65
60
55
50
45
40
35
30

Broad Elements

- In-kernel APIs are pretty normal
  - ‘Command Queue’ and ‘Completion Queue’ style HW programming model
  - Fairly normal use of DMA API in some cases
  - ‘Double action’ on DMA mapping in others
- Lots of in-kernel users now
  - Storage: SCSI over RDMA (srp), iSCSI over RDMA (iser), NVMe over RDMA (nvme/rdma)
  - Filesystem: NFS RPC over RDMA (xprtrdma), CIFS/SMB over RDMA, 9p over RDMA (9p/trans_rdma)
  - Networking: IP over InfiniBand (ipoib), RDS over RDMA, SMC
- User API is exciting
  - uAPI is very big
  - Direct DMA to/from user memory
  - ‘interrupt’ delivery to user space
  - HW enforced user space security
  - Supports a split driver model where part of the driver lives in user space, part in the kernel
  - Significant user-space performance considerations

Not so Unique Anymore

- GPU/DRM has a similar split user/kernel driver concept
- ‘AI’ related drivers are heading the same directions
- General ideas for ‘accelerator’ frameworks: warp drive, gen-z, etc
- vfio has a lot of the same problems with user-space DMA
- Industry wide push toward PASID based DMA
  - ie frictionless support for user space DMA

Kernel probably needs more tools to support this style of user API

Challenge: What is RDMA

- What should be in the subsystem?
  - Is a request /response DMA ring enough?
- Any kind of stateful networking?
  - Even if it doesn't do direct data placement?
  - Even if parts are not offloaded?
- Non-networking accelerators?

- When has a driver gone 'too far'?
  - Are char devs in a driver OK?
  - Delegate more security decisions to the device?
  - Do not support in-kernel verbs?

What is RDMA

The endless extensions

- ‘standard’ verbs is 20 years old now, little has changed, but the industry has moved
  - Computational offloads!
  - Invent new wire protocols!
  - Invent new objects!
  - Expose micro-optimizations as API!
  - Support Ethernet packets, all the offloads, all the optimizations!

- ‘standard verbs’ is the minimum necessary to be useful for kernel users
- ‘device specific verbs’ are really required to meet the needs of large scale users

- How to balance innovation, API and standards?

Challenge: User/Kernel ABI

- Broad and multi-faceted:
  - Four char devices (`/dev/infiniband/*`)
  - Growing netlink interface (NETLINK_RDMA)
  - Wide & wild usage of sysfs
  - ~7000 lines of `include/uapi/` headers
  - >150 unique kernel ‘syscalls’
  - 397 uapi structs

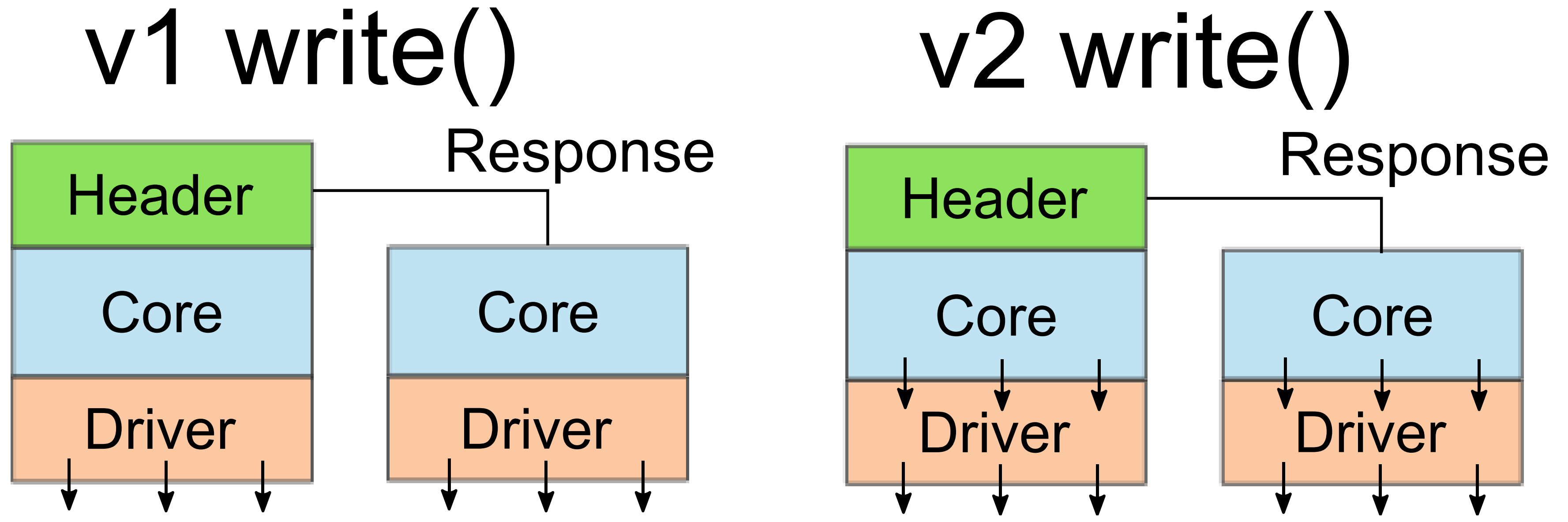
User/Kernel ABI

- 15 years of being upstream and keeping compatibility has taken its toll
  - `include/uapi/` is not complete! Still lots of ‘stealth’ uapi floating around in other headers
  - Until recently kernel headers were not even being used by user-space
  - For a while chunks of the user-space was largely abandoned
  - New multiplexor uABIs introduced, but old stuff wasn’t migrated
  - Multiple ABIs in the kernel and user-space, for compatibility
  - Driver inherently supplies part of the user ABI
    Constant battle to prevent drivers from doing bad things here
  - Many things would not be acceptable if proposed today
- A lot to keep track of and get right, limited participation

User/Kernel ABI

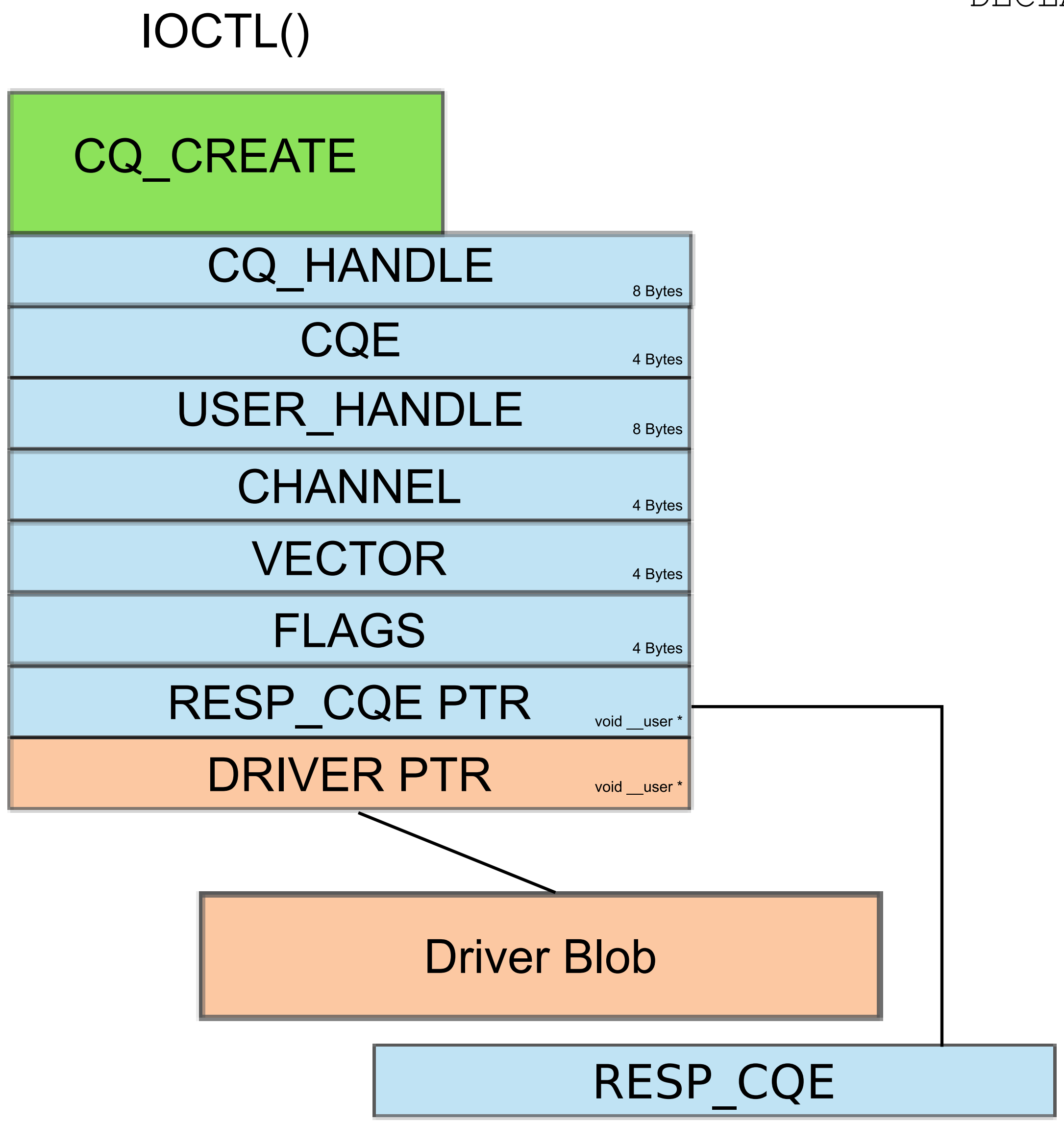
ABI Generations

- The ABI was designed to have a multiplexor. In 2005 someone said “don’t use IOCTL, it is bad”
  So write() was used.
  The scheme allowed the command struct to be variable sized, but only the driver part could be grown
- In 2013 the multiplexor header was redone to have sizes for both core & driver.
  But only ‘new’ commands used this header
- In 2016 someone noticed that using write was an insane security hole
- In 2017 RDMA gained a netlink inspired ioctl() interface, but only ‘new new’ commands used it
- In late 2018 IOCTL finally was updated to execute any command and became the preferred interface
  Still a lot of technical debt in the drivers around handling their part of the uABI



ioctl ABI

- Somewhat abstract and general, may be useful in other places



```
DECLARE_UVERBS_NAMED_METHOD(
        UVERBS_METHOD_CQ_CREATE,
        UVERBS_ATTR_IDR(UVERBS_ATTR_CREATE_CQ_HANDLE,
                        UVERBS_OBJECT_CQ,
                        UVERBS_ACCESS_NEW,
                        UA_MANDATORY),
        UVERBS_ATTR_PTR_IN(UVERBS_ATTR_CREATE_CQ_CQE,
                           UVERBS_ATTR_TYPE(u32),
                           UA_MANDATORY),
        UVERBS_ATTR_PTR_IN(UVERBS_ATTR_CREATE_CQ_USER_HANDLE,
                           UVERBS_ATTR_TYPE(u64),
                           UA_MANDATORY),
        UVERBS_ATTR_FD(UVERBS_ATTR_CREATE_CQ_COMP_CHANNEL,
                       UVERBS_OBJECT_COMP_CHANNEL,
                       UVERBS_ACCESS_READ,
                       UA_OPTIONAL),
        UVERBS_ATTR_PTR_IN(UVERBS_ATTR_CREATE_CQ_COMP_VECTOR,
                           UVERBS_ATTR_TYPE(u32),
                           UA_MANDATORY),
        UVERBS_ATTR_FLAGS_IN(UVERBS_ATTR_CREATE_CQ_FLAGS,
                             enum ib_uverbs_ex_create_cq_flags),
        UVERBS_ATTR_PTR_OUT(UVERBS_ATTR_CREATE_CQ_RESP_CQE,
                            UVERBS_ATTR_TYPE(u32),
                            UA_MANDATORY),
        UVERBS_ATTR_UHW());
```

ioctl ABI

- Similar to netlink, arguments are described by a Tag and a Length:
  - IOCTL uses a flex array of fixed size tag descriptions
  - Data larger than the fixed size is transferred by pointer using copy_to/from_user
  - Runs synchronously during ioctl() vs ‘like a network packet’ of netlink
- Has a funky domain-specific-language made with the C preprocessor to declare the syntax of the methods and tags
  - Core code validates a lot of input. Much harder for drivers to make mistakes
  - DSL is elaborated at runtime to build the validation database
  - Language describes more than we can validate to help humans understand the ABI
- Includes a way for the driver to have its own tag number space on every command. Drivers can extend commands with their own stuff in a much more controlled and safe way

ABI: Toolbox vs midlayer

- View the RDMA core subsystem’s uAPI more as a ‘toolbox’ and less as a ‘midlayer’
- A lot of uAPIs are just the core marshaling syscall arguments and calling the driver, not adding much value.
  - Some drivers then go an re-marshal the data again and send it on to the device

- Experimenting with drivers providing user interfaces directly in their device-specific form RDMA core is just some general toolbox to help this along

- Much less kernel side code
- More trust in the driver and device

- Challenging the notion of what a driver is

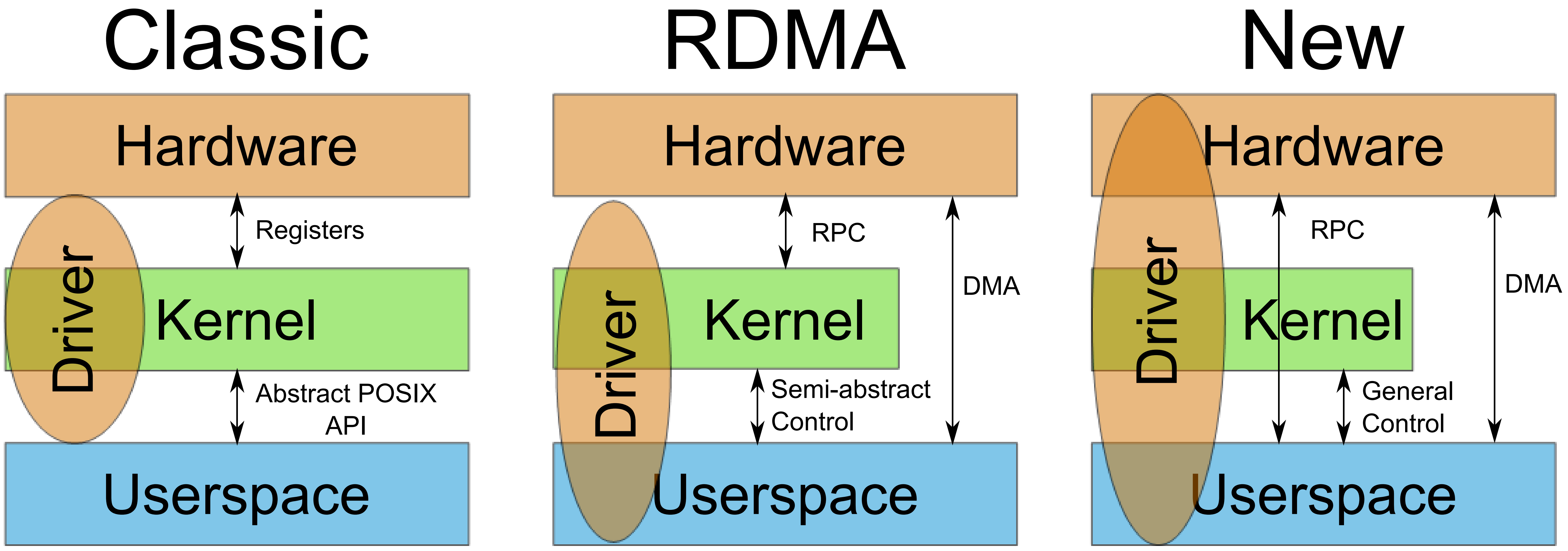
Challenge: Security

RDMA inherently does DMA from user-space controlled memory!
Triggers DMA without a kernel system call.
Must trust the device, so we have a HW security model too

- Devices must implement a baseline security model:
  - Awareness of a security group – no accesses across groups permitted!
  - Multiple levels of privilege, at least kernel level and user space level
  - Privileged IOMMU to contain and virtualize DMA to user-space.
    IOMMU must be linked to the security group

- Ethernet derived devices additionally:
  - Must respect kernel concepts like CAP_NET_RAW/CAP_NET_BROADCAST/etc
  - Changing the list of allowed source and destination addresses is privileged
  - User-space must not control L2/L3 headers

What’s in a driver anyhow?



Partial BAR mmap'd to user-space
DMA commands from user buffers

Kernel forwards RPCs from userspace
Mediates general things like dma_map

Security

- The HW involved security model exists for performance
  - Similar for other HW security models like CPU privilege, ASID, IOMMU, etc
  - Unusual for a PCI-E device to provide this kind of model
- Published standards guide the implementation
  - Do all devices implement the standard correctly?
  - Vetting a ‘driver’ is hard, lots of important information is secret
  - Even if it wasn’t secret the driver and HW spec are vast
  - Who would review it?

Containers

- HW involved security means it can’t be extended. HW does as the standard says

- Container namespaces are a new security model that doesn’t have a direct mapping to standards
  - Containers have restricted access to global device resources
  - HW can’t fully enforce this

- Current solution dedicates a PCI function to each container
  - A PCI function corresponds to a VM and provides a matching security boundary for net namespaces
  - Future will see ‘light weight’ PCI functions per container

- Some stress migrating into a net namespace world
  - net namespaces were introduced without updating RDMA
  - RDMA customers are now used to seeing all RDMA devices in all containers –have to preserve for compatibility

Challenge: DMA from user-space

- RDMA does DMA from user-space on arbitrary user controlled Virtual Addresses:
  Page Cache pages, DAX, anonymous memory, huge pages, PCI BAR mmaps (future)
- Support for this is not great in the core kernel:
  - Writing to file backed pages can trigger crashes
  - DAX can’t be used with long term pins (ie RDMA)
  - Scattered DMA addresses have a big negative impact on performance
    Hard to get physical memory more contiguous than huge pages in user-space
  - Minimal support for RDMA to ZONE_DEVICE pages

- No ‘DMA API’ in user space for manual cache coherence
  - Not clear this could even exist generally, lots of complicated corner cases
  - Today user space relies on ‘DMA Coherent’ architectures
  - But the kernel can’t enforce this, so it is up to the user not to use it wrong
- Existing long standing programming libraries in user space ignore DMA incoherence
  - Even if we provided all the cache flushing APIs nobody would care enough to use it
  - Zero industry interest in this topic

DMA between other devices

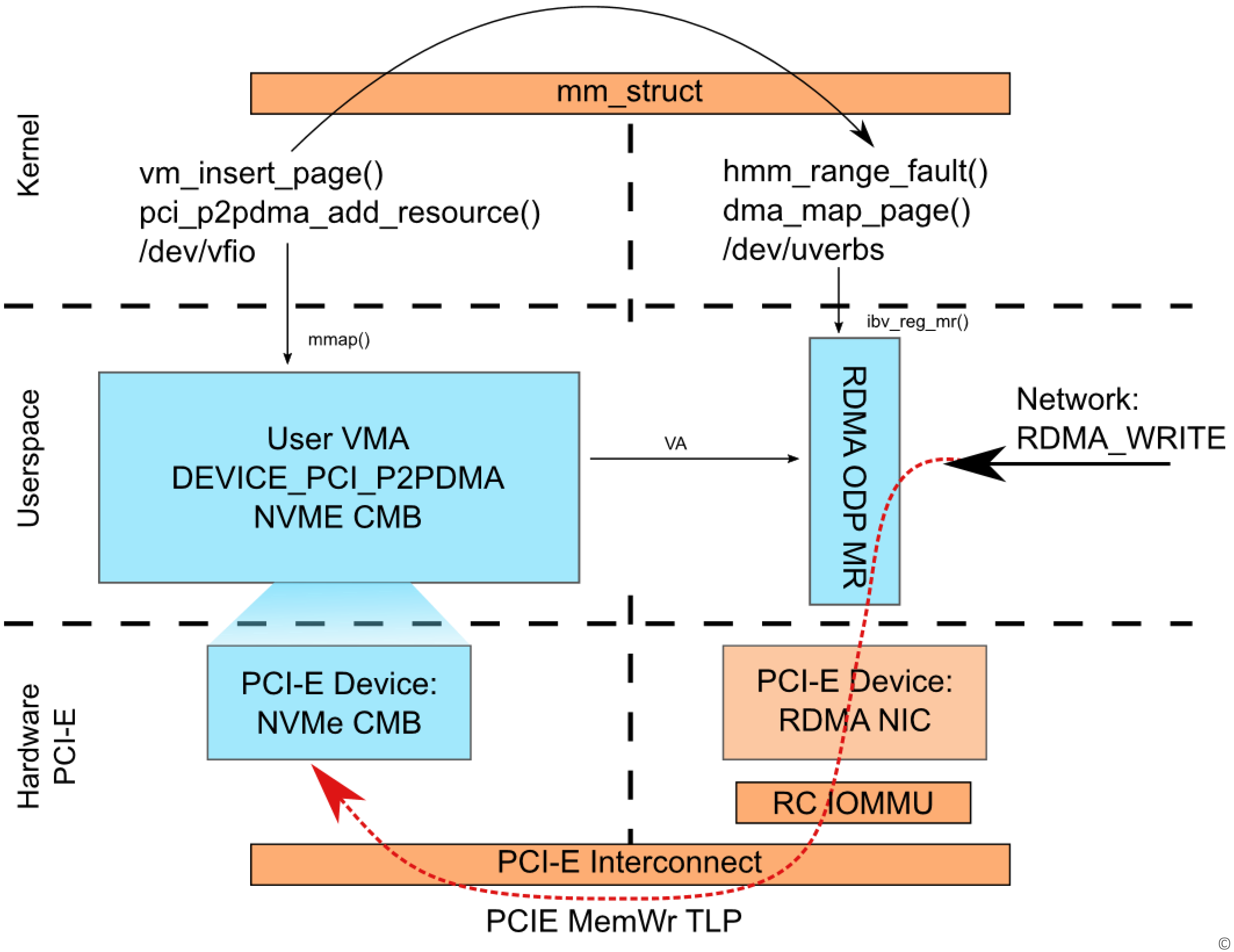
Modern systems tend to be constrained in memory bandwidth

- RDMA is the big bandwidth pipe into the system
- The CPU is not the only device with memory
  - GPU Compute and Computational Storage
  - Storage

- Desire to transfer data directly from network to non-CPU devices

- PCI Peer to Peer transfer

- Want to move toward the idea of a ‘DMA Fault’ for pages
  - Means the page is being touched for DMA, vs touched by the CPU
  - Trigger a PCI Peer to Peer DMA transfer
  - Do not force the data to move into CPU memory if it is DEVICE_PRIVATE
  - Work with BAR backed pages mmap’d to user space: DMABUF, VFIO


Kernel
Userspace
Hardware PCI-E
mm_struct
vm_insert_page()
pci_p2pdma_add_resource()
/dev/vfio
hmm_range_fault()
dma_map_page()
/dev/uverbs
mmap()
ibv_reg_mr()
User VMA
DEVICE_PCI_P2PDMA
NVME CMB
VA
RDMA ODP MR
Network:
RDMA_WRITE
PCI-E Device:
NVMe CMB
PCI-E Device:
RDMA NIC
RC IOMMU
PCI-E Interconnect
PCIE MemWr TLP

DMA from user-space

Intersection with HMM and GPU

- RDMA has an mode called On Demand Paging, to avoid long term page pins
  - Required to mix RDMA with DAX
  - Allows overcommit, swap, COW, etc
- Implementation of similar ideas as CONFIG_HMM_MIRROR used by GPU drivers
  - RDMA's implementation is many years old now
  - Working to harmonize
- GPU drivers and RDMA have differences in implementation on how DMA is managed
  - No dma_fence() in RDMA
  - User space triggered DMA
- DMA directly to process VAs. ie PCI-E PASID, POWER9 CAPI, ARM SVMM
  - Still emerging, not clear if we have the right kernel APIs for this

Challenge: Overlap with netdev

It seems all networks need to eventually carry IP traffic, and all protocols eventually need to be carried over IP

Even though RDMA is networking distinct from netdev, there is a lot of interaction and requirements between netdev

Challenge: Overlap with netdev

- Netdev deliberately excludes stateful networking offloads from their stack
  - A full TCP offload would skip everything related to netfilter, BPF, routing, etc.
  - RDMA always runs on some stateful offload

- When the underlying physical connection is ethernet, RDMA mirrors netdev state into the HW offloads:
  - Syncs IP addresses, neighbor discovery, routing, bonding, and so on
  - Takes packets before they reach netdev, so ignores netfilter, BPF, statistics, etc
  - Access only through RDMA APIs

- Things get complicated when RDMA is using the same L3 headers, ie UDP or TCP
  - netdev just looses a port, traffic is captured by the RDMA HW
  - Really ugly port reservation scheme for iWarp in user-space

- Two netdev drivers in RDMA: ipoib and OPA vnic
  - For performance both delegate their entire data plane to the driver

DPDK and RDMA

- RDMA always had the idea of a ‘raw’ datagram QP
- In some cases you could get simple access to ethernet frames on RoCE devices, do ‘DPDK’ like things

- Eventually a DPDK PMD was created for this
- Now the RDMA stack, not VFIO, underlies a major DPDK PMD
  - Makes sense, DPDK is exactly in the line of high performance kernel bypass users RDMA was invented for

- Lots and lots of stress from this:
  - DPDK wants every wild offload and optimization you can imagine
  - How to accommodate incredibly device specific things within a common ABI?

- RDMA is growing more overlap with VFIO, particularly vfio-mdev
  - Security guarantees come from the inherent security model of RDMA
  - Relies on the RDMA device to handle ‘DMA isolated to one process’
    Same as any other RDMA user

Mellanox TECHNOLOGIES

Mellanox TECHNOLOGIES

YEAR ANNIVERSARY