FACEBOOK

# NetGPU

Jonathan Lemon
bsd@fb.com

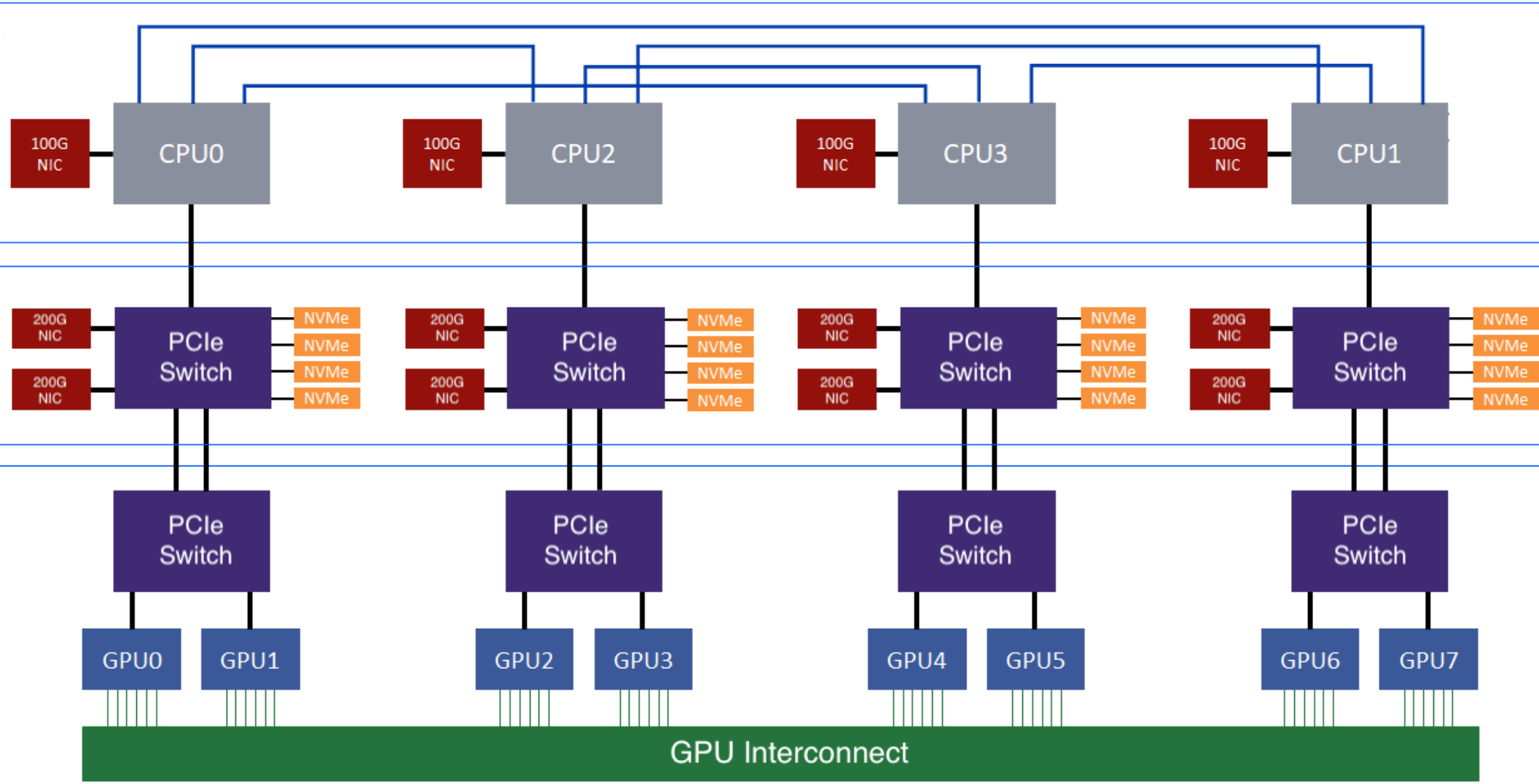
FACEBOOK

# Agenda

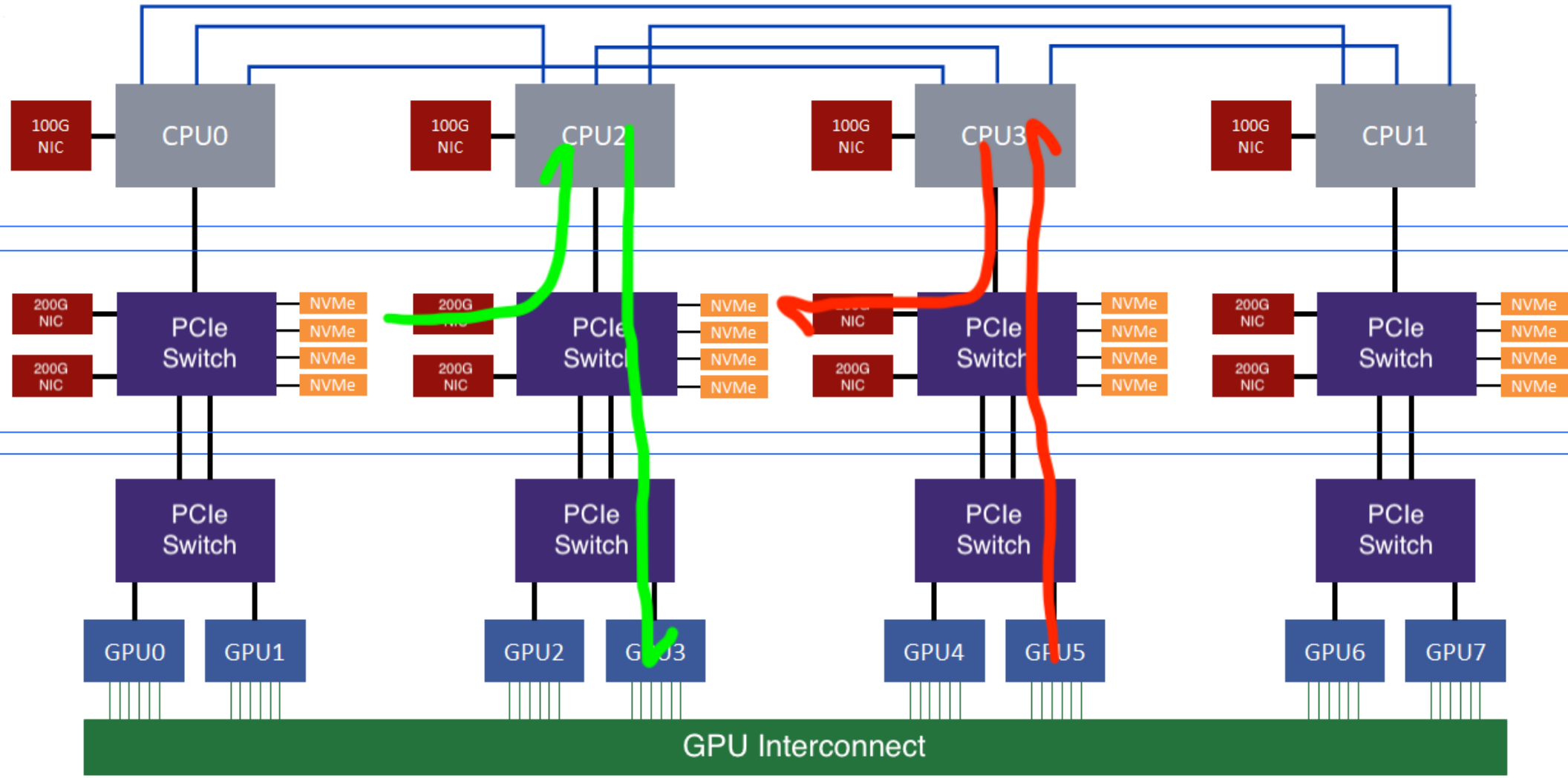
1. Motivation behind NetGPU
2. Targeted Architecture and Applications
3. Operation
4. API design
5. VM providers
6. Conclusion

## Motivation

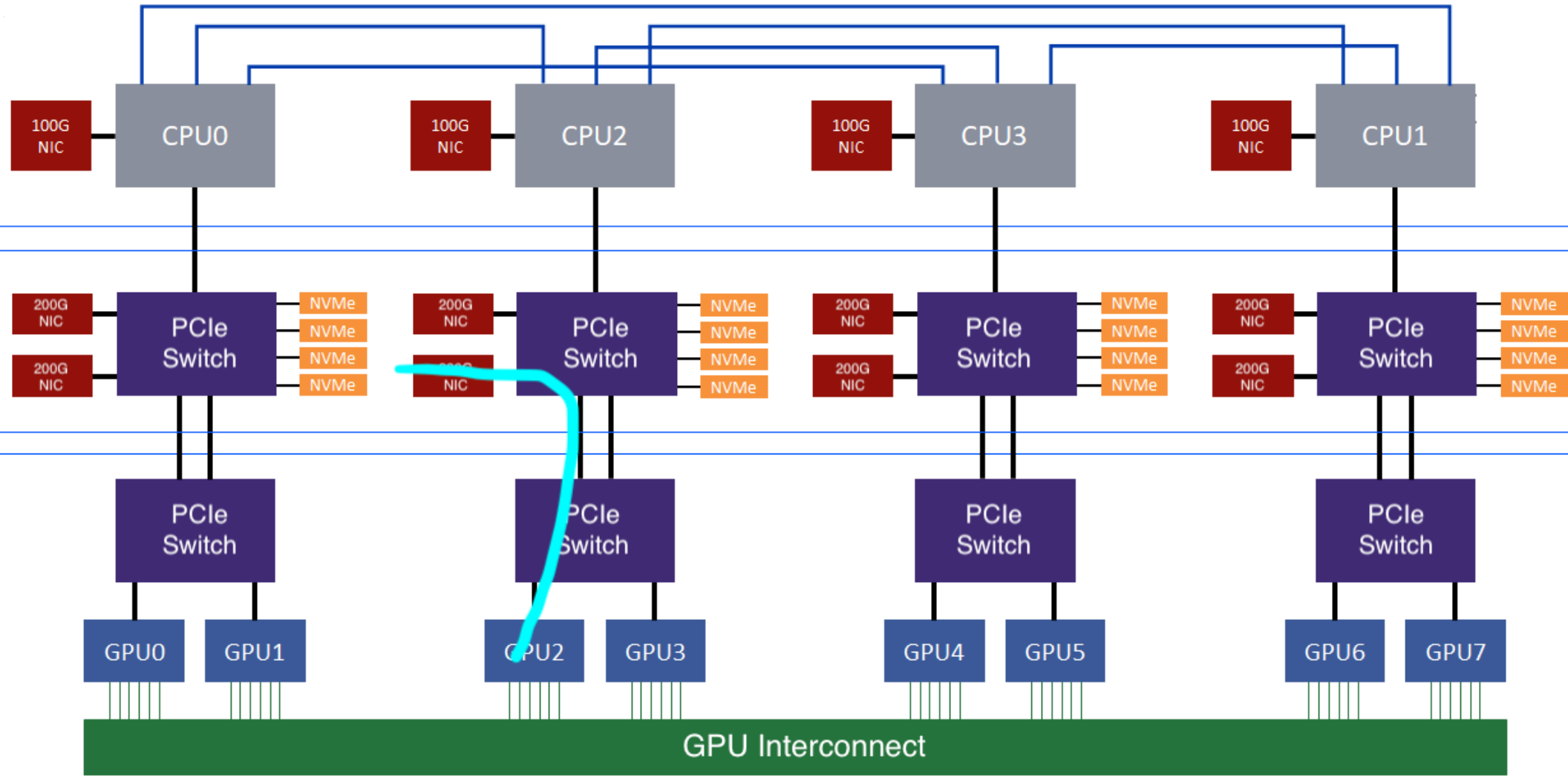
Scale up ML clusters for larger datasets.  A proposed design under consideration has 200Gbps NICs and GPUs attached to a PCIe switch. There are several of these nodes, and the link to the host CPU can't handle the dataset traffic.


100G NIC
CPU0
100G NIC
CPU2
100G NIC
CPU3
100G NIC
CPU1
200G NIC
200G NIC
PCIe Switch
NVMe
NVMe
NVMe
NVMe
200G NIC
200G NIC
PCIe Switch
NVMe
NVMe
NVMe
NVMe
200G NIC
200G NIC
PCIe Switch
NVMe
NVMe
NVMe
NVMe
200G NIC
200G NIC
PCIe Switch
NVMe
NVMe
NVMe
NVMe
PCIe Switch
PCIe Switch
PCIe Switch
PCIe Switch
GPU0
GPU1
GPU2
GPU3
GPU4
GPU5
GPU6
GPU7
GPU Interconnect

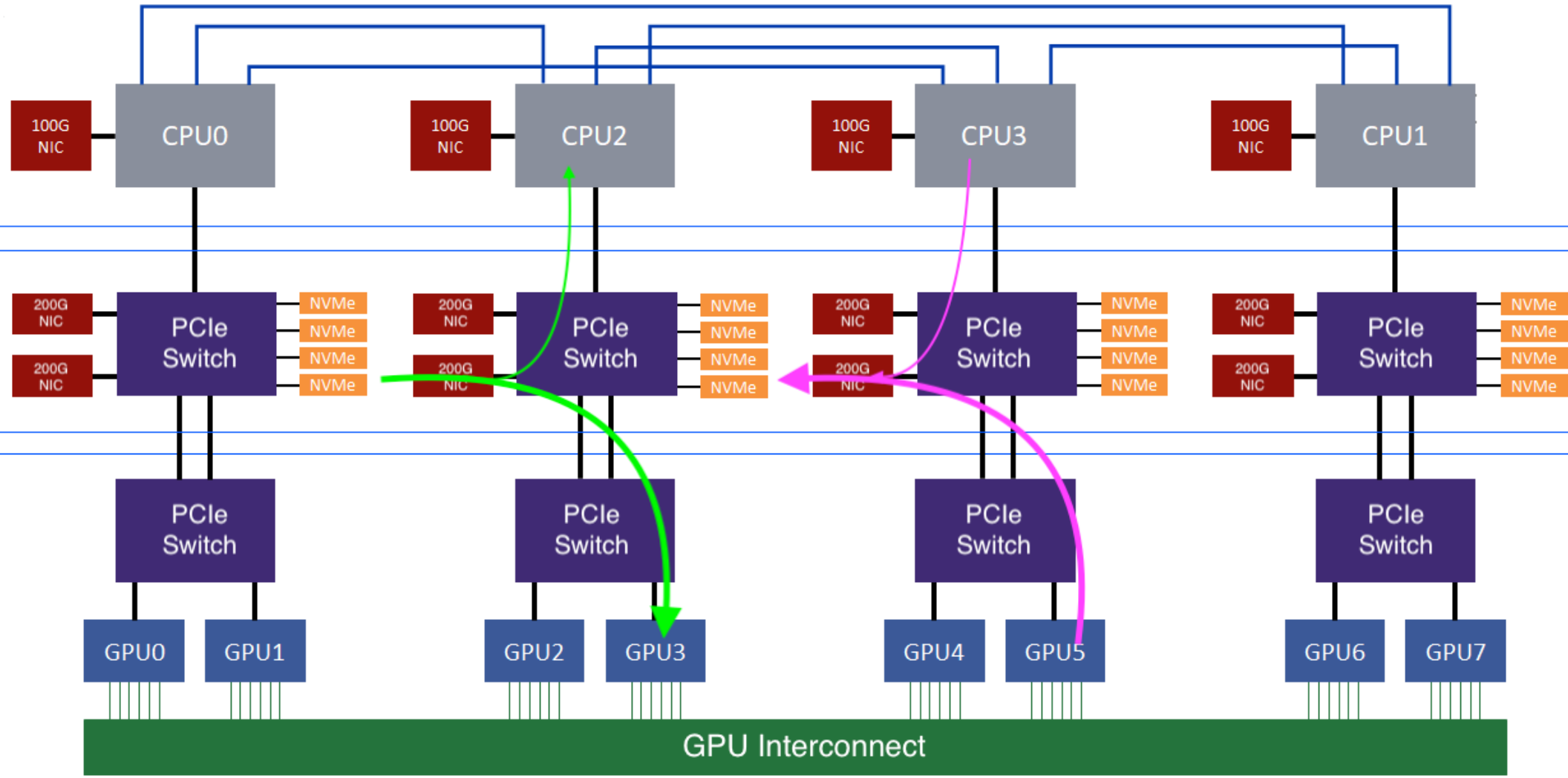
## Normal datapath



# RDMA or RoCE datapath



# NetGPU datapath



## Objectives

- Why not ...
  - RDMA?  RoCE?  Current zerocopy?  AF_XDP?  io_uring?
- Utilize Kernel protocol stack, keep control in app/kernel.
  - Leverage existing code: TCP, UDP, GRO/GSO, kernel routing
- Latency is very important to us
  - Pre-register memory, pre-map DMA addresses.
- Zero-copy transfers, reduce PCI bus usage.
- Allow bpf inspection of headers

## Target Applications

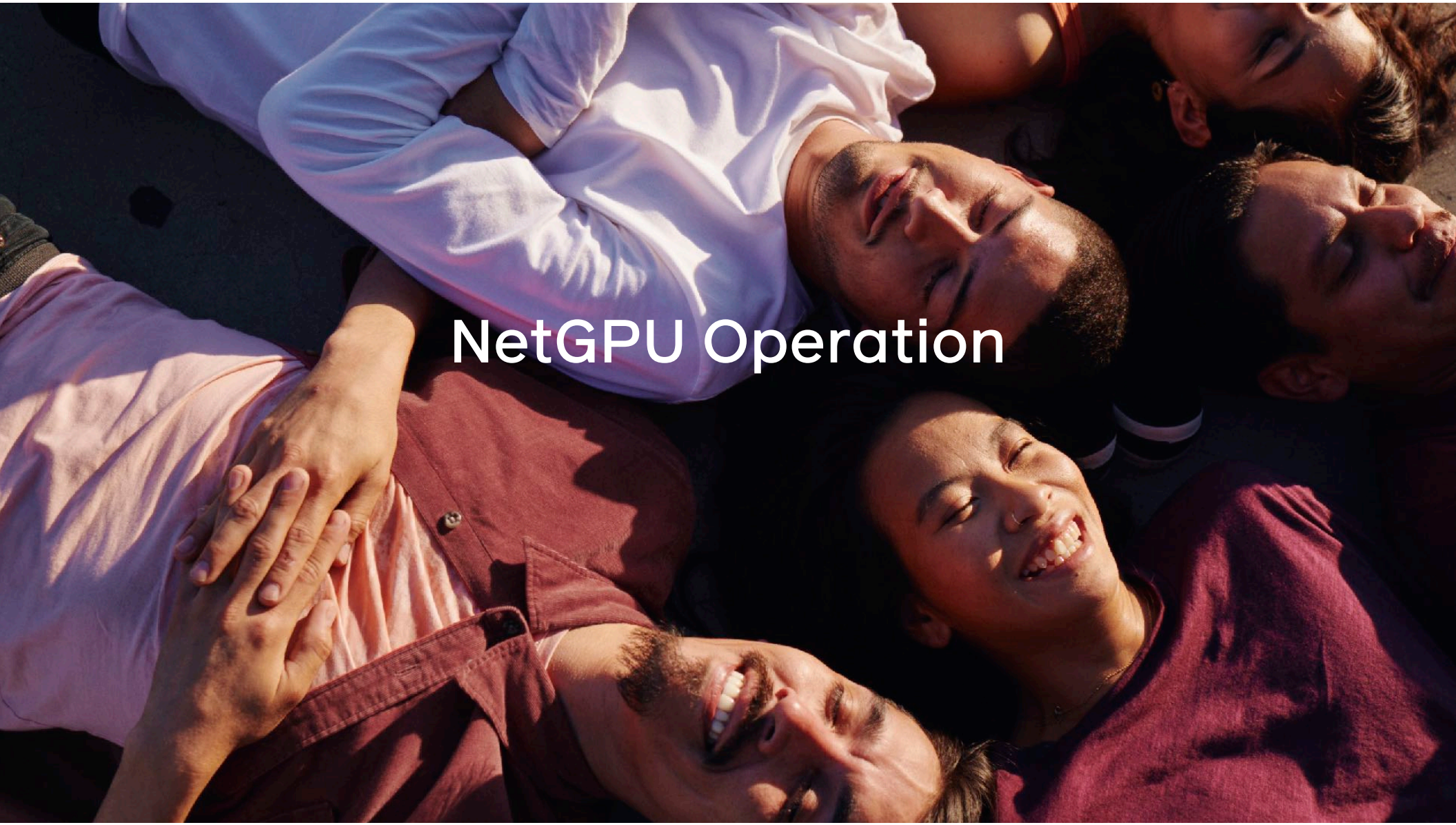
- Machine Learning on GPU
  - ALLREDUCE
    - Datapath on GPU uses one TX socket, one RX socket, in a ring.
  - ALLGATHER
    - Datapath follows star layout (1:N)

- Storage (NVMe)
- Unicast same data to multiple endpoints.

## Machine Learning Specifics
## .... and constraints

- ML applications use own packages.
  - E.g.: pyTorch
- Built on top of vendor's GPU libraries
  - AMD hip, NVIDIA cuda
- FB hipify tool provides source level rewrites from cuda → hip
- Clang extension compiles .hip code into AMD or NVIDIA kernels.

- Should be able to interoperate with GPU library code.


NetGPU Operation

# Packet TX path

- va_handle is provided to sendmsg()
  - GPU: gpu_vaddr, from hipMalloc() or cudaMalloc()
  - host_memory: user_address
  - io_uring: region_id:offset
- iov_iter_get_pages() maps va_handle into pages.
  - This needs to access the registered memory mapping.
- Retrieved pages are attached to the skb,
  - skb is sent down stack, which does not touch the data pages.

- driver calls dma_map() function to get dma addresses for the pages.
- DMA path between NIC and GPU is already established, need to retrieve the mapping.

# Packet RX path

- Packet arrives at NIC
- Header splitting places header on host page, and data on GPU page
- skb is constructed:
  - Header is copied into skb linear region
  - skb frag points to gpu page data.
- skb is sent up the stack, and deposited onto the sk receive queue.
- sk_data_ready places (va_handle, len, ?) entries on the skq RX queue.

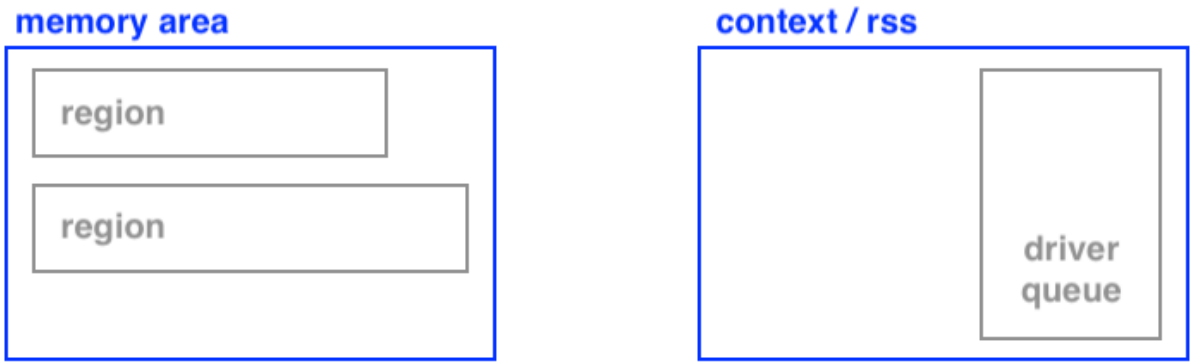
# TX Notifications

- Application needs to know when kernel is done with buffer pages.
- Existing zerocopy code collects completions in socket error queue
  - Application reads error queue to obtain completions.

- Instead of polling(reading) completions, request explicit notification.
  - sendmsg( ... cmsg= {SO_NOTIFY, { fd, user_data } } )

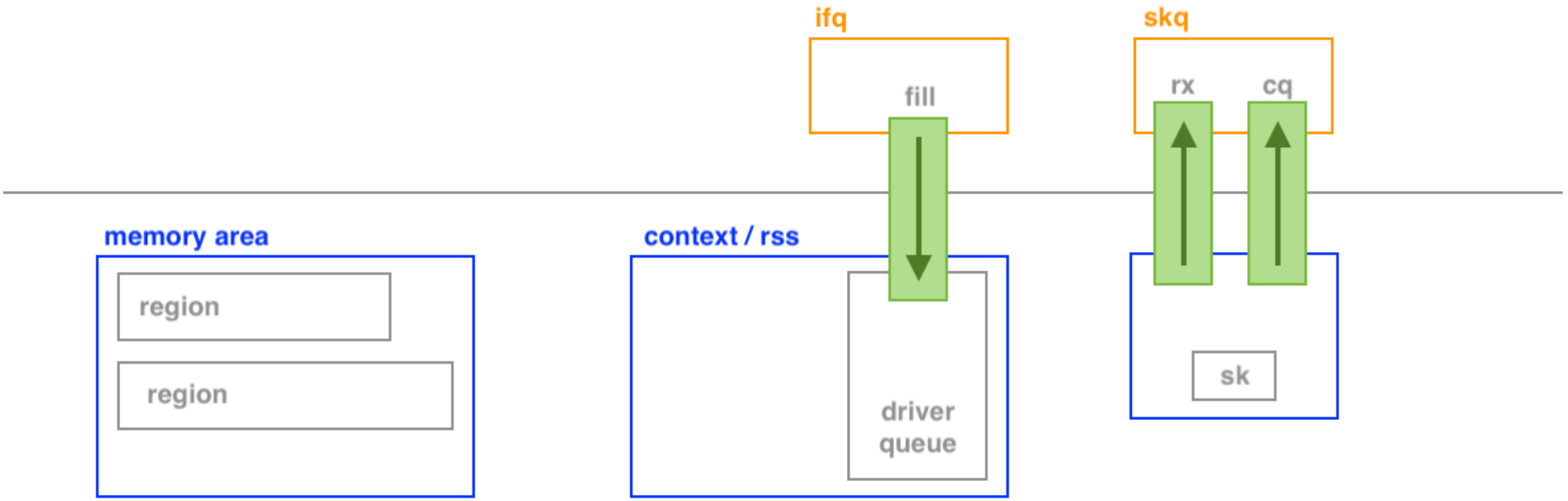
- Post notification when all skbs up to the notify point are available for reuse.
  - Ordered notifications.
  - fd of -1 == use pre-registered notification channel  (netgpu skq.cq or io_uring CQ)

- Under evaluation




Memory area contains registered memory, provides va_handle → page lookup

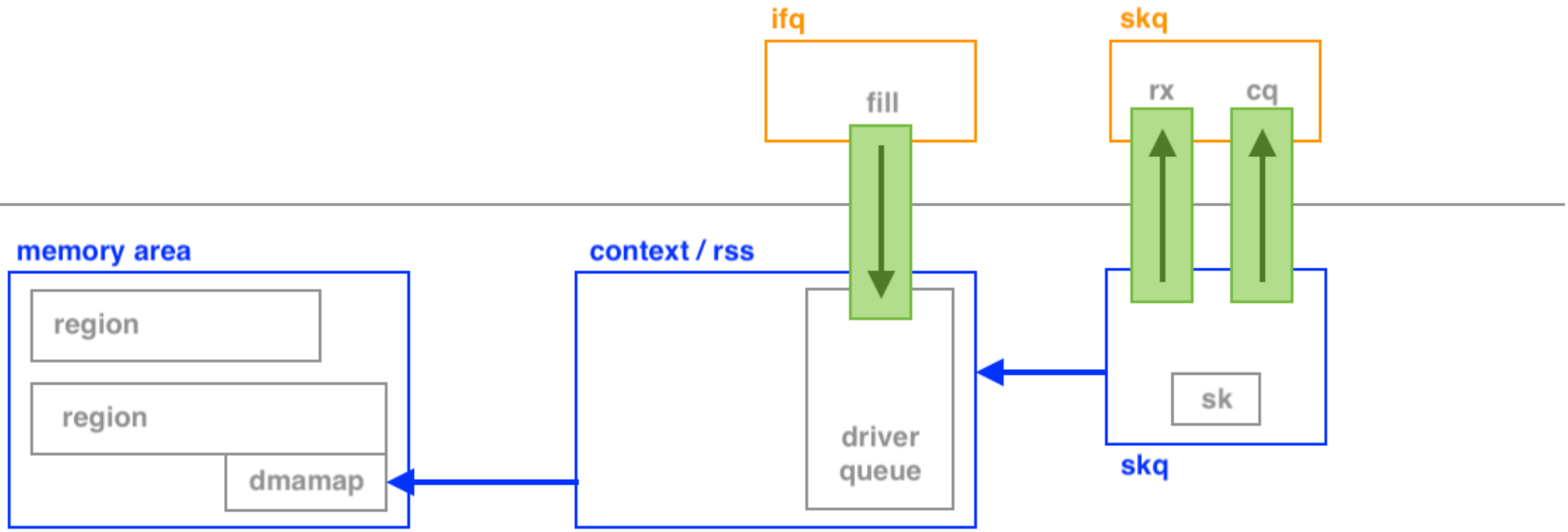
Device context (aka RSS) may contain multiple IFQs




Like AF_XDP, ifq provides buffers to driver ring via a fill queue.

RX data packets are placed in the skq.rx shared ring.

TX completion notifications are delivered to skq.cq, or io_uring cq.




skq receives data from any ifq in the context.

dmamaps are specific each device, but may be shared between multiple contexts on a device.

region page mappings are shared across all contexts.


API Design

## NetGPU API

- API shown here are userspace wrappers for prototyping.
- Kernel interface is just 5 ioctls.
- Subject to change - e.g.: fold some things into io_uring

# Memarea

```
int netgpu_open_memarea(struct netgpu_mem **memp, int provider_fd);
int netgpu_add_memarea(struct netgpu_mem *mem, void *va, size_t size);
```

- memarea holds registered memory/buffers.
- add() returns id of registered region.
- would be nice if this could converge with io_uring registered buffers.
- for host memory, va is process memory.
- for GPU memory, va is addr returned from hipMalloc() or cudaMalloc().

# Device context

```
int netgpu_open_ctx(struct netgpu_ctx **ctxp, const char *ifname);
int netgpu_attach_region(struct netgpu_ctx *ctx, struct netgpu_mem *mem, int idx);
```

- Opens context on device.
- Attaches identified memory region.
- Sets up the DMA path between the memory provider and NIC.

- Context is intended as an RSS context
  - Add more parameters to open() to handle specific cases.

# Interface queue

```
int netgpu_open_ifq(struct netgpu_ifq **ifqp, struct netgpu_ctx *ctx,
                    int queue_id, int fill_entries);
bool netgpu_recycle_buffer(struct netgpu_ifq *ifq, void *ptr);
```

- For RX (not required for TX)
- Allocates a RX queue from the device.  queue_id can be -1 for "any queue".
- Can add multiple queues to a context.
- Relies on external traffic steering.
  - Could extend open() with steering rules.

- recycle_buffer places ptr in fill queue for driver to use.  (bulk operations not shown)
  - fill queue is a shared memory ring.

# Socket queue
## ... for RX

```
int netgpu_attach_socket(struct netgpu_skq **skqp, struct netgpu_ctx *ctx,
                         int fd, int nentries);
int netgpu_get_rx_batch(struct netgpu_skq *skq,
                        struct iovec *iov[], int count);
int netgpu_get_cq_batch(struct netgpu_skq *skq,
                        uint64_t *notify[], int count);
```

- creates skq structure for socket.
  - read() should fail - can't mix normal and zerocopy sockets.
- get() accessors just read the shared queue.
- epoll() support: EPOLLIN and EPOLLRDBAND indicate data available in shared ring for RX or CQ.
- would like to converge CQ with io_uring implementation.


Memory providers

## Memory providers

- Current memory providers:
  - Host (CPU), NVIDIA GPU, AMD GPU (in progress)

- Provides va_handle for allocated memory regions.
- Sets up page mappings, pins pages.
- Set up DMA path between NIC and provider.
- Provides DMA mappings for pages to the NIC.

```
struct netgpu_ops {

        struct netgpu_region *
                (*add_region)(struct netgpu_mem *, const struct iovec *);
        void    (*free_region)(struct netgpu_mem *, struct netgpu_region *);


        struct netgpu_dmamap *
                (*map_region)(struct netgpu_region *, struct device *);
        void    (*unmap_region)(struct netgpu_dmamap *);


        dma_addr_t
                (*get_dma)(struct netgpu_dmamap *map, unsigned long addr);
        int     (*get_page)(struct netgpu_dmamap *map, unsigned long addr,
                            struct page **page, dma_addr_t *dma);
        int     (*get_pages)(struct netgpu_region *r, struct page **pages,
                             unsigned long addr, int count);
};
```

```
struct netgpu_ops {

        struct netgpu_region *
                (*add_region)(struct netgpu_mem *, const struct iovec *);
        void    (*free_region)(struct netgpu_mem *, struct netgpu_region *);


        struct netgpu_dmamap *
                (*map_region)(struct netgpu_region *, struct device *);
        void    (*unmap_region)(struct netgpu_dmamap *);


        dma_addr_t
                (*get_dma)(struct netgpu_dmamap *map, unsigned long addr);
        int     (*get_page)(struct netgpu_dmamap *map, unsigned long addr,
                            struct page **page, dma_addr_t *dma);
        int     (*get_pages)(struct netgpu_region *r, struct page **pages,
                             unsigned long addr, int count);
};
```

```
struct dma_buf_ops {

        int (*attach)(struct dma_buf *, struct dma_buf_attachment *);
        void (*detach)(struct dma_buf *, struct dma_buf_attachment *);


        int (*pin)(struct dma_buf_attachment *attach);
        void (*unpin)(struct dma_buf_attachment *attach);


        struct sg_table * (*map_dma_buf)(struct dma_buf_attachment *,
                                         enum dma_data_direction);
        void (*unmap_dma_buf)(struct dma_buf_attachment *,
                              struct sg_table *,
                              enum dma_data_direction);


        void *(*vmap)(struct dma_buf *);
        void (*vunmap)(struct dma_buf *, void *vaddr);
};
```

```
struct netgpu_ops {

        struct netgpu_region *
                (*add_region)(struct netgpu_mem *, const struct iovec *);
        void    (*free_region)(struct netgpu_mem *, struct netgpu_region *);


        struct netgpu_dmamap *
                (*map_region)(struct netgpu_region *, struct device *);
        void    (*unmap_region)(struct netgpu_dmamap *);


        dma_addr_t
                (*get_dma)(struct netgpu_dmamap *map, unsigned long addr);
        int     (*get_page)(struct netgpu_dmamap *map, unsigned long addr,
                            struct page **page, dma_addr_t *dma);
        int     (*get_pages)(struct netgpu_region *r, struct page **pages,
                             unsigned long addr, int count);
};
```

```
   add_region():
       dmabuf = dma_buf_get(fd);
       allocate(dmabuf, va, size);

   map_region():
       attach(dmabuf, nic_device_attachment);
       pin(nic_device_attachment);
       sgt = map_dma_buf(nic_device_attachment, DMA_BIDIRECTIONAL);

       drm_prime_sg_to_page_addr_arrays(
                        struct sg_table *sgt, struct page **pages,
                        dma_addr_t *addrs, int max_pages);
```

```
struct dma_buf_ops {

        int (*attach)(struct dma_buf *, struct dma_buf_attachment *);
        void (*detach)(struct dma_buf *, struct dma_buf_attachment *);


        int (*pin)(struct dma_buf_attachment *attach);
        void (*unpin)(struct dma_buf_attachment *attach);


        struct sg_table * (*map_dma_buf)(struct dma_buf_attachment *,
                                         enum dma_data_direction);
        void (*unmap_dma_buf)(struct dma_buf_attachment *,
                              struct sg_table *,
                              enum dma_data_direction);


        void *(*vmap)(struct dma_buf *);
        void (*vunmap)(struct dma_buf *, void *vaddr);
};
```

## Pain Points

- Need fallback path for access to unmapped pages.
- External flow steering rules for ifq.
- RX requires L4 header splitting (or workarounds).
- dma_buf not made available by AMD ROCt.
- When skb is freed to system, must not release any GPU pages.
  - Annotate skb instead of page?

# Thank you!

FACEBOOK

## AMD specific
### ... trying to get at the dma_buf object.

- ROCk (kernel driver) provides bufobj handles to ROCt
- ROCt - userspace thunk driver which interfaces between application and kernel driver.
- ROCt maintains address → bufobj translation in userspace rbtree, returning the VA address to the application.
- Application calls sendmsg(), kernel cannot translate address → bufobj.
- Need to extend ROCt, in order to register addresses/bufobj with netgpu.