Accelerate Everything

Enabling Peer-to-Peer Traffic with NoLoad® NVMe Computational Storage and P2PDMA and Examining Real World Use Cases

Andrew Maier
Strategic Account Lead/Firmware Team Lead

September 16, 2024

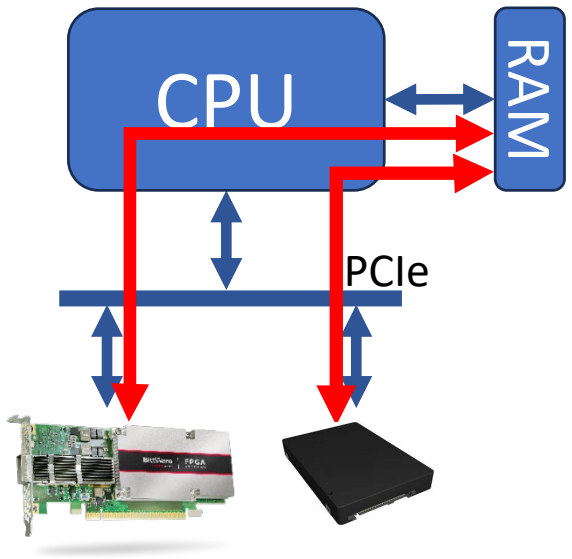
- Introduction to P2PDMA
  - Discuss recent framework updates
- Userspace Interface Example
- Userspace Interface Performance Testing
- Potential Use Cases
  - SSD <-> SSD
  - NIC <-> SSD
- Summarize and discuss next steps

Why P2PDMA?

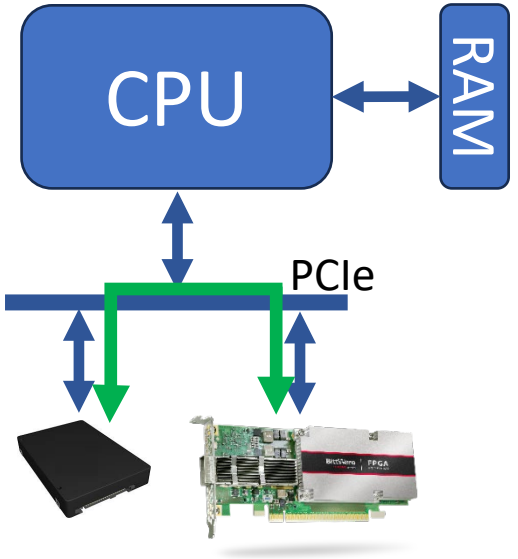
- PCIe devices are getting more plentiful and faster
- PCIe devices have increasing more and more memory (GPUs, SSDs, etc)
  - More and more memory is becoming HBM
- Some DMA transfers can be made more efficient by eliminating the use of host DRAM
  - Results in more efficient DRAM usage and, in some cases, lower latency for applications
- Supported on most modern CPUs (Intel/AMD/ARM)
- Upstream equivalent to NVIDIA's GPUDirect

Introduce Peer-to-Peer Transfers (P2PDMA)

Traditional DMA Between PCIe Devices



Traditional DMA Between PCIe Devices with P2PDMA



Features

- High performance interface
- Open-source framework
- Eliminates RAM Usage

What is Peer-to-Peer (P2P)?

- P2PDMA is an open source upstream framework for registering/using p2p memory
- Simple API for registering resources
  - Any PCIe BAR (or partial BAR) can be mapped as a P2PDMA region

```
/**
 * pci_p2pdma_add_resource - add memory for use as p2p memory
 * @pdev: the device to add the memory to
 * @bar: PCI BAR to add
 * @size: size of the memory to add, may be zero to use the whole BAR
 * @offset: offset into the PCI BAR
 *
 * The memory will be given ZONE_DEVICE struct pages so that it may
 * be used with any DMA request.
 */
int pci_p2pdma_add_resource(struct pci_dev *pdev, int bar, size_t size,
                            u64 offset)
{
```

- P2PDMA regions are available to both Kernelspace and Userspace applications
- P2PDMA supported in NVMe

NVMe Controller Memory Buffer (CMB)

- Controller Memory Buffer (CMB) is a BAR exposed by an NVMe device in the standard
- Introduced in NVMe v1.2 (2014)

> **3.1.4.11 Offset 38h: CMBLOC – Controller Memory Buffer Location**
>
> This optional property defines the location of the Controller Memory Buffer (refer to section 8.2.1). If the controller does not support the Controller Memory Buffer (CAP.CMBS), this property is reserved. If the controller supports the Controller Memory Buffer and CMBMSC.CRE is cleared to '0', this property shall be cleared to 0h.

- Developed to support both control and data flow through the NVMe CMB
  - Supported in the upstream linux driver
- All NVMe CMB devices are registered as P2PDMA accessible memory
  - Accessible by both Kernelspace and Userspace
- NVMe 2.0 introduced more features for supporting CMB (CMBMSC for Virtual Machine Support)

Figure 52: Offset 50h: CMBMSC – Controller Memory Buffer Memory Space Control

Bits	Type	Reset	Description
63:12	RW	0h	**Controller Base Address (CBA):** This field specifies the 52 most significant bits of the 64-bit base address for the Controller Memory Buffer's controller address range. The Controller Memory Buffer's controller base address and its size determine its controller address range. The specified address shall be valid only under the following conditions: a) no part of the Controller Memory Buffer's controller address range is greater than $2^{64} - 1$; and b) if the Persistent Memory Region's controller memory space is enabled, then the Controller Memory Buffer's controller address range does not overlap the Persistent Memory Region's controller address range.

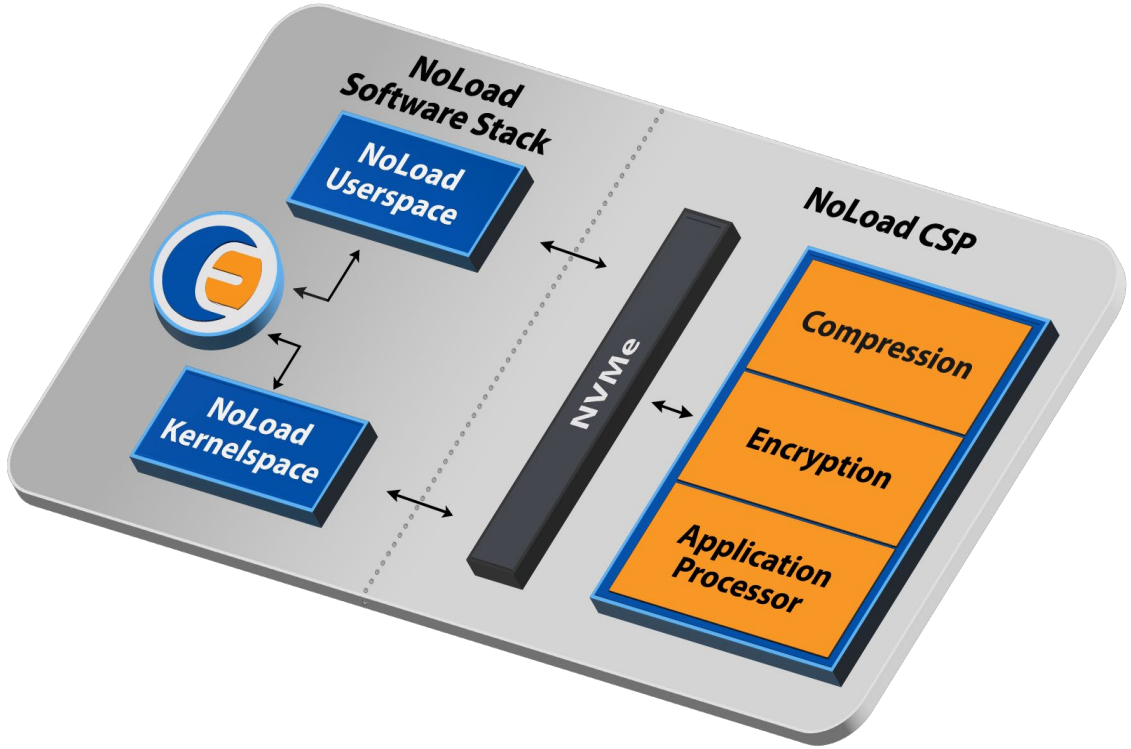
NoLoad® Computational Storage Processor

Computational Storage Processor (CSP)

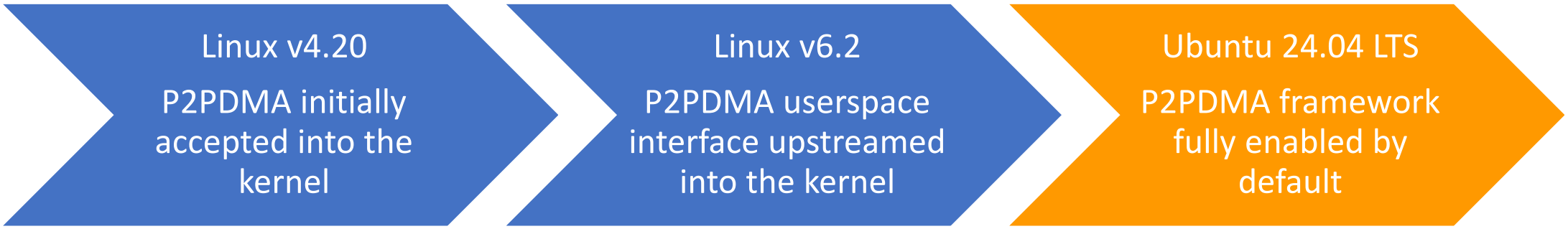
- Purpose built for accelerator of storage and compute workloads
- Does not directly have any storage

NoLoad Platform

- NoLoad NVMe Front End
- Flexible size NVMe CMB (all NoLoad images have at least 512MiB)
- P2PDMA enabled by default (with NVMe CMB)
    - Accelerators can be added into the P2PDMA path easily and using upstream tools/drivers



P2PDMA Recent Milestones



P2PDMA Integration – Kernelspace Applications

- Kernelspace applications have fine grained access to memory types
- Full API for registering, monitoring, and using P2PDMA memory
  - *pci_p2pdma_add_resource* – register a P2PDMA region
  - *pci_p2pdma_distance_many* – determine the distance from a P2PDMA provider
  - *pci_has_p2pmem* – check if PCIe device has any P2PDMA memory published
  - *pci_p2pmem_find_many* – Find a P2PDMA published device
  - *pci_alloc_p2pmem* – allocate P2PDMA memory
  - *pci_free_p2pmem* – free P2PDMA memory
  - *pci_p2pmem_virt_to_bus* – Get the bus address of a P2PDMA virtual address
  - *pci_p2pmem_alloc_sgl* – allocate SGL of P2PDMA memory
  - *pci_p2pmem_free_sgl* – free SGL P2PDMA memory
  - *pci_p2pmem_publish* – publish P2PDMA memory for other devices to use with pci_p2pdma_find
  - *pci_p2pdma_enable_store* – parse and store the P2PDMA enabled configfs/sysfs attribute
  - *pci_p2pdma_enable_show* – show the P2PDMA enabled configfs/sysfs attribute

P2PDMA Integration – Userspace Applications

- Linux community finally agreed on an interface
  - After many failed iterations
- sysfs/configfs interface for accessing P2PDMA registered memory (/sys/device/…./p2pmem)

```
nike_/sys/class/nvme/nvme10/device/p2pmem $ pwd -P
/sys/devices/pci0000:5d/0000:5d:00.0/0000:5e:00.0/0000:5f:07.0/0000:63:00.0/p2pmem
nike_/sys/class/nvme/nvme10/device/p2pmem $ ls
allocate  available  published  size
nike_/sys/class/nvme/nvme10/device/p2pmem $ cat size
536870912
nike_/sys/class/nvme/nvme10/device/p2pmem $ cat available
536870912
nike_/sys/class/nvme/nvme10/device/p2pmem $
```

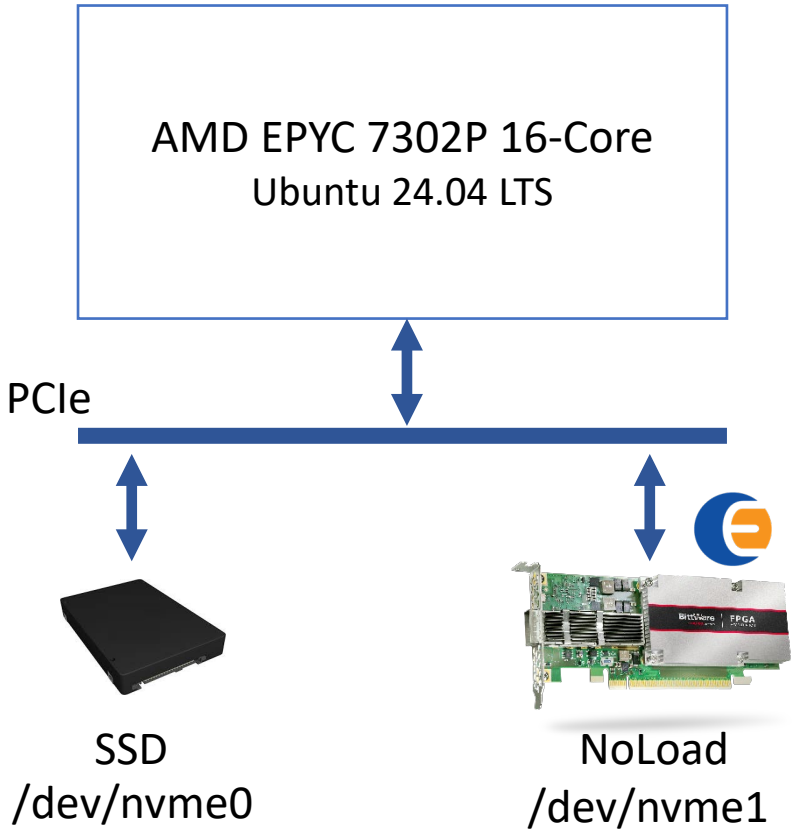
allocate: File to mmap for allocating against the P2PDMA memory

available: (print) Shows the current available P2PDMA memory (in bytes)

published: (print) Shows the amount of published P2PDMA memory(in bytes)

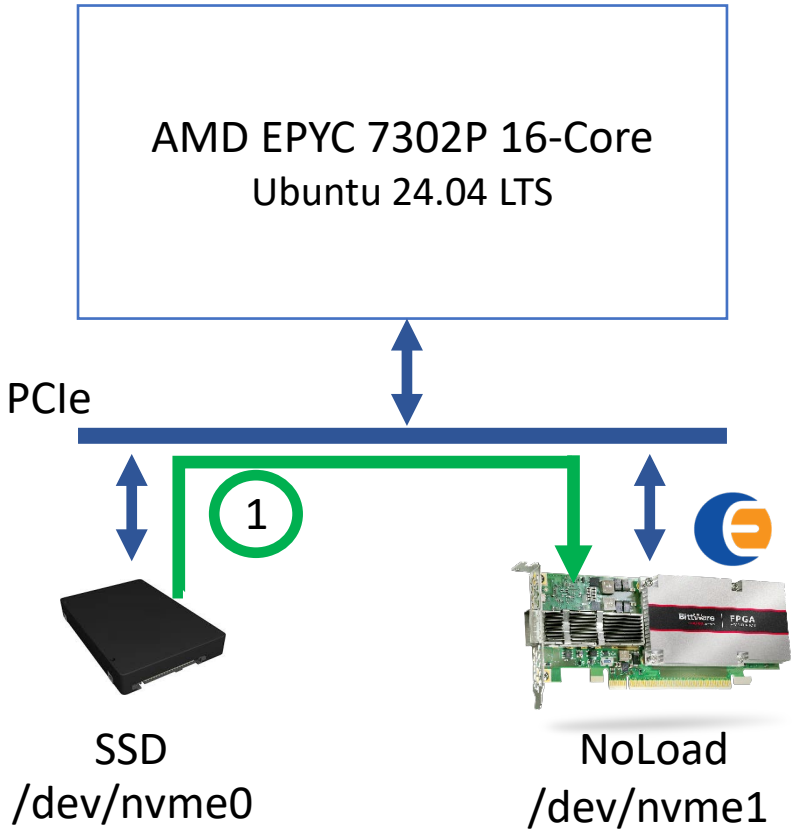
size: (print) Shows the total amount of P2PDMA memory (in bytes)

P2PDMA Integration – Userspace Example



- Consider a system with an SSD (/dev/nvme0) and a NoLoad (/dev/nvme1) and the following example data path

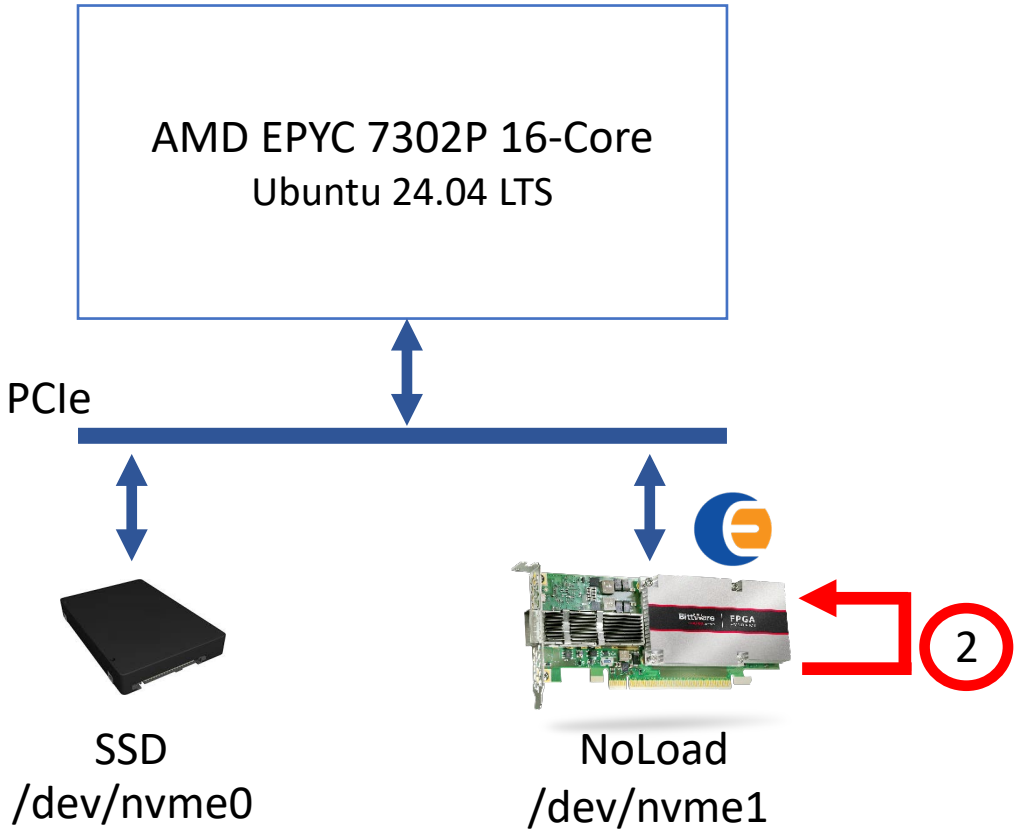
P2PDMA Integration – Userspace Example



- Consider a system with an SSD (/dev/nvme0) and a NoLoad (/dev/nvme1) and the following example data path

1. Data transfer from the SSD to the NoLoad CMB
   - Aka a read from the SSD to the NoLoad CMB buffer

P2PDMA Integration – Userspace Example



- Consider a system with an SSD (/dev/nvme0) and a NoLoad (/dev/nvme1) and the following example data path

1. Data transfer from the SSD to the NoLoad CMB
   - Aka a read from the SSD to the NoLoad CMB buffer
2. Trigger internal data transfer from CMB to accelerator (i.e compression)

P2PDMA Integration – Userspace Example

- P2PDMA sysfs allocate file access is simple
  - Simply put the fd of the opened file into a mmap call

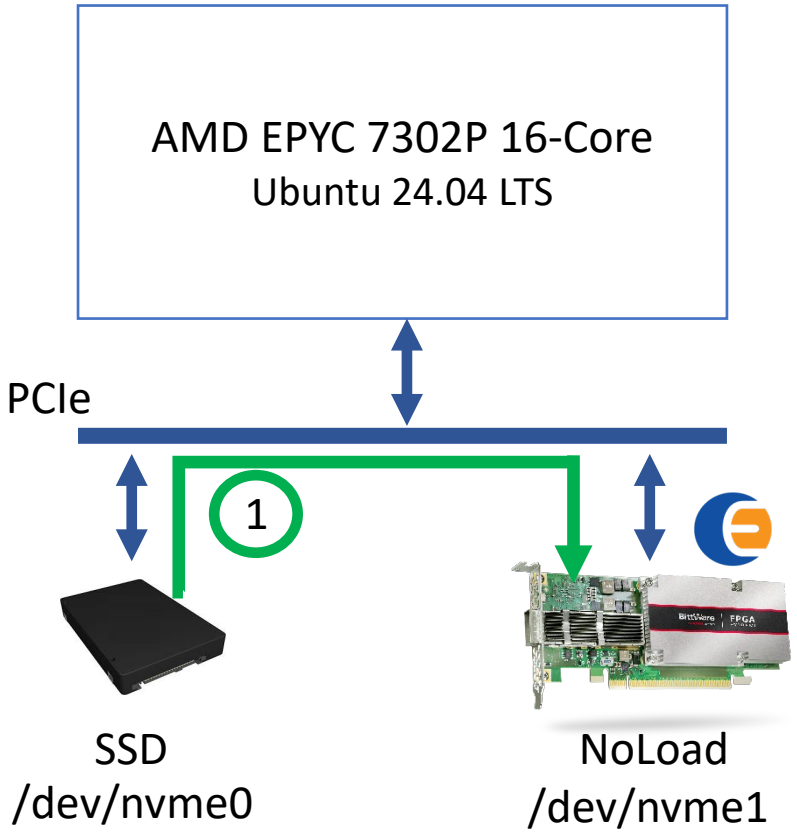
```
// Open a file descriptor to the allocate file
int fd = open("/sys/class/nvme/nvme1/device/p2pmem/allocate", O_RDWR | O_BINARY);

// mmap X number of bytes against the file descriptor
void *data = mmap(NULL, 4096, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
```

- The *data* variable in the above snippet now (upon success) will contain a pointer to the P2PDMA memory that can be used in DMA operations
  - For example, this can be used as the source/destination buffer for NVMe read calls (or IOCTLs)

```
rc = read(ssd_fd, data, 4096);
```

```
// C IOCTL structure for an NVMe command in linux/nvme_ioctl.h
struct nvme_user_io cmd = {};
cmd.opcode = NVME_CMD_WRITE;
cmd.nblocks = (uint16_t)(4096/512) - 1;
cmb.addr = (uintptr_t)data;

rc = ioctl(noload_fd, NVME_IOCTL_SUBMIT_IO, cmd);
```

P2PDMA Integration – Userspace Example

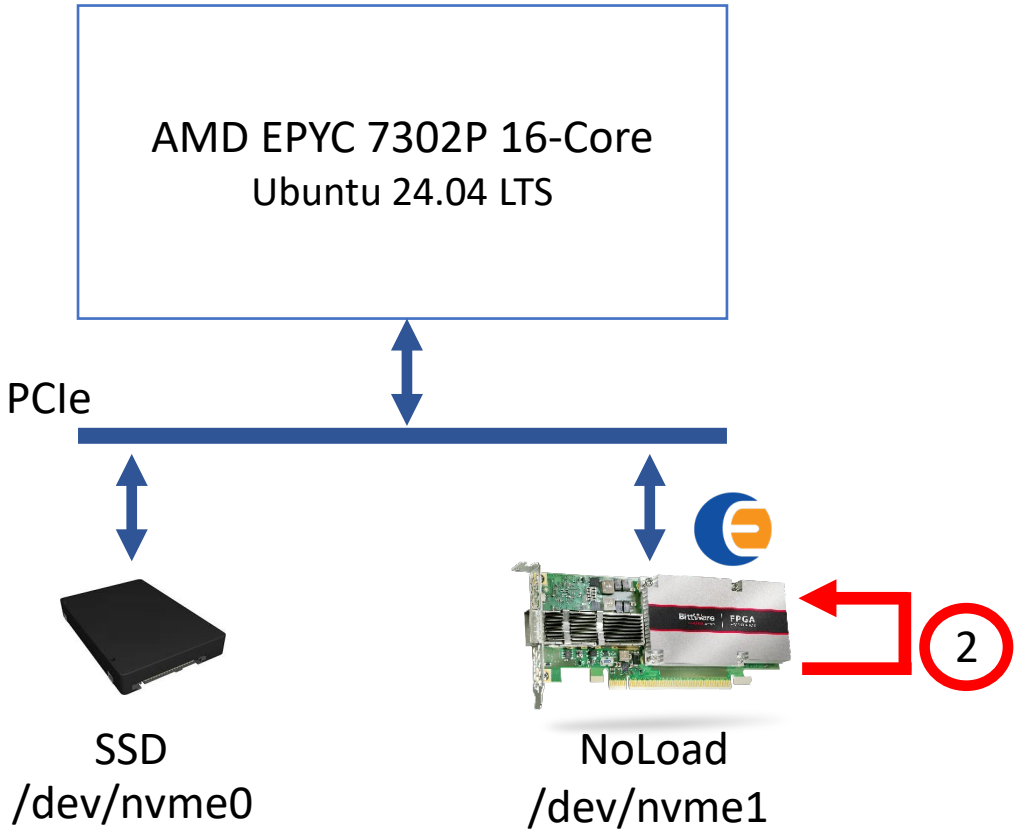


- Consider a system with an SSD (/dev/nvme0) and a NoLoad (/dev/nvme1) and the following example data path

1. Data transfer from the SSD to the NoLoad CMB
   - Aka a read from the SSD to the NoLoad CMB buffer

```
rc = read(ssd_fd, data, 4096);
```

P2PDMA Integration – Userspace Example

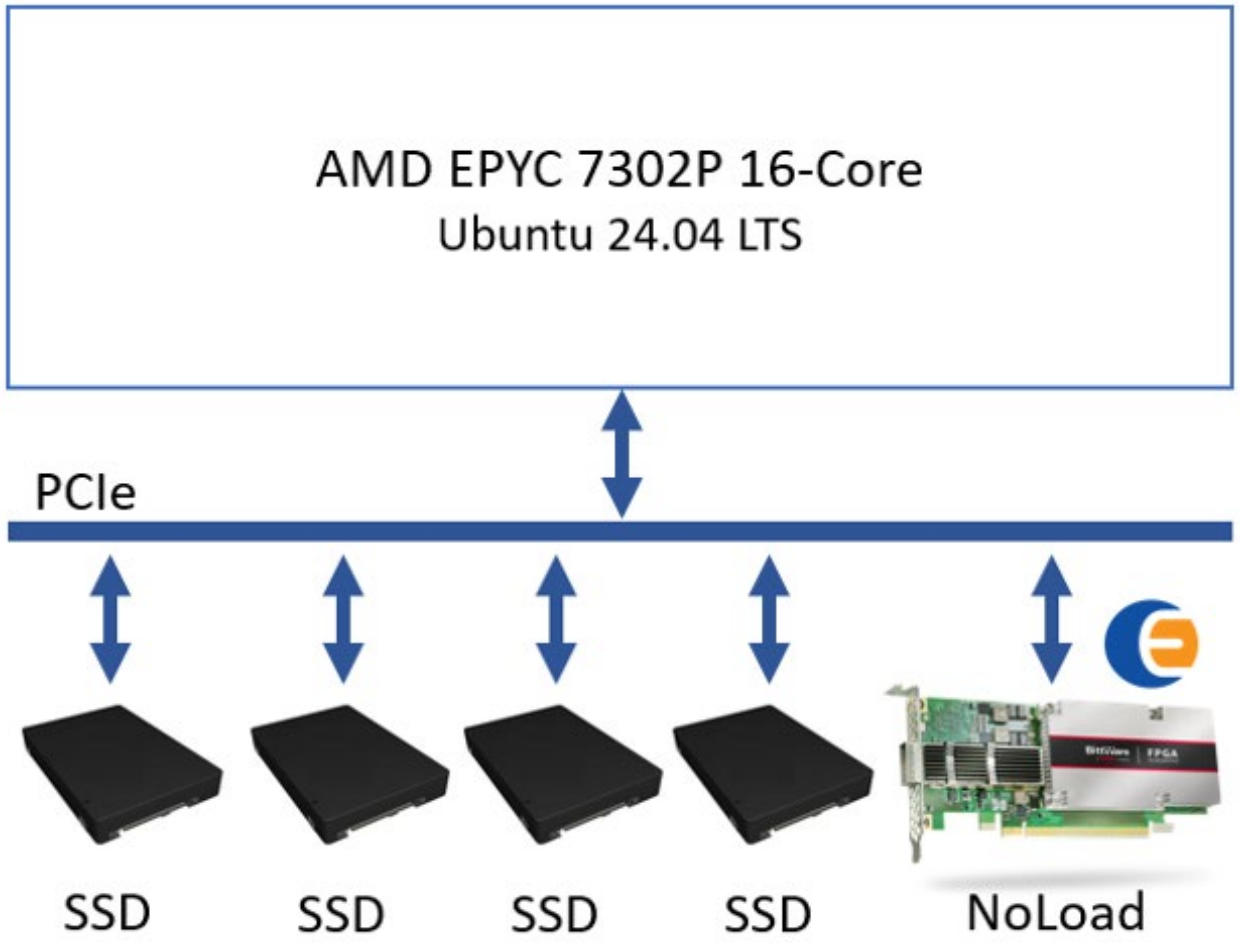


- Consider a system with an SSD (/dev/nvme0) and a NoLoad (/dev/nvme1) and the following example data path

1. Data transfer from the SSD to the NoLoad CMB
   - Aka a read from the SSD to the NoLoad CMB buffer
2. Trigger internal data transfer from CMB to accelerator (i.e compression)

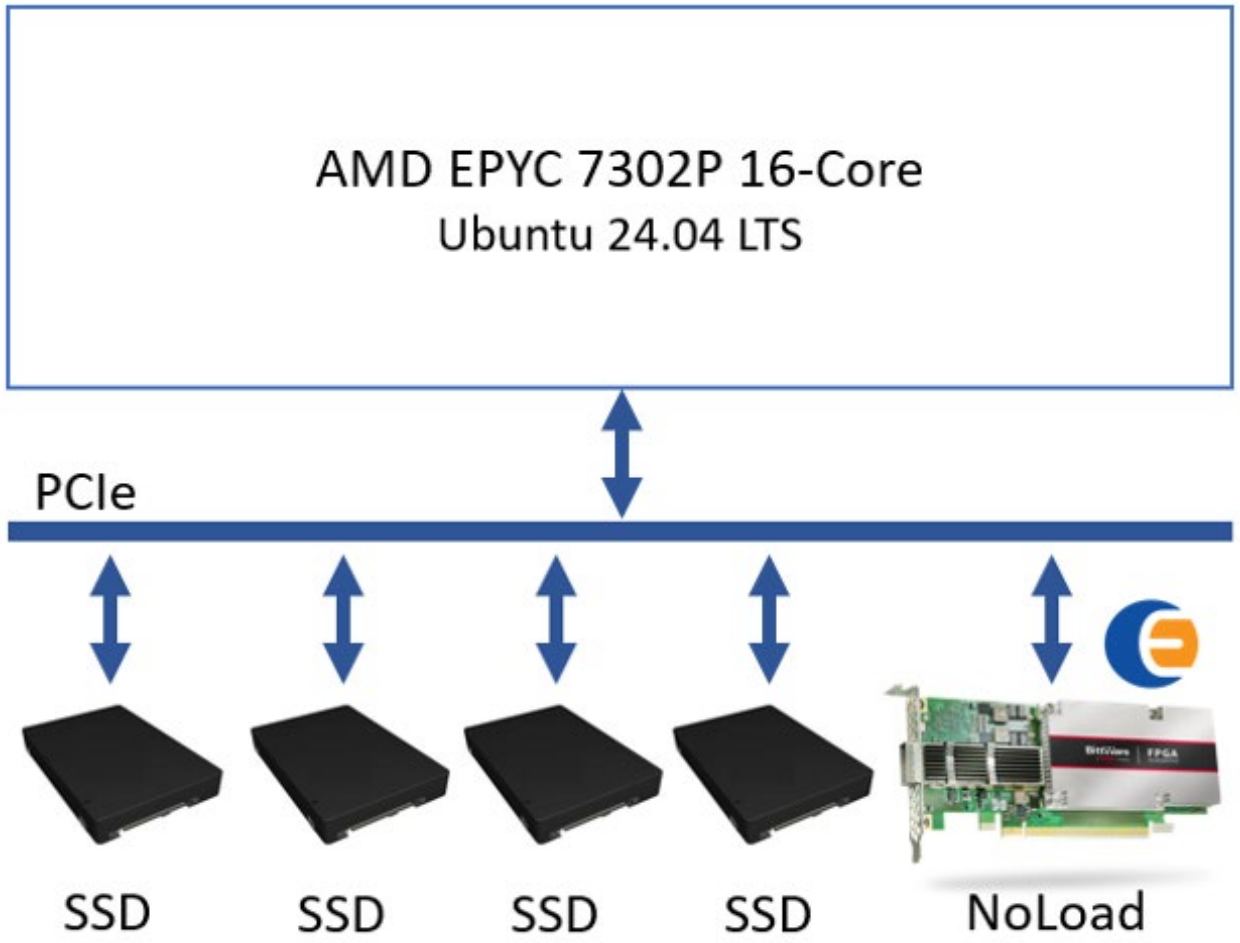
```c
// C IOCTL structure for an NVMe command in linux/nvme_ioctl.h
struct nvme_user_io cmd = {};
cmd.opcode = NVME_CMD_WRITE;
cmd.nblocks = (uint16_t)(4096/512) - 1;
cmb.addr = (uintptr_t)data;

rc = ioctl(noload_fd, NVME_IOCTL_SUBMIT_IO, cmd);
```

Testing the Framework – SSD <-> SSD



- Consider a test to use the new Userspace interface

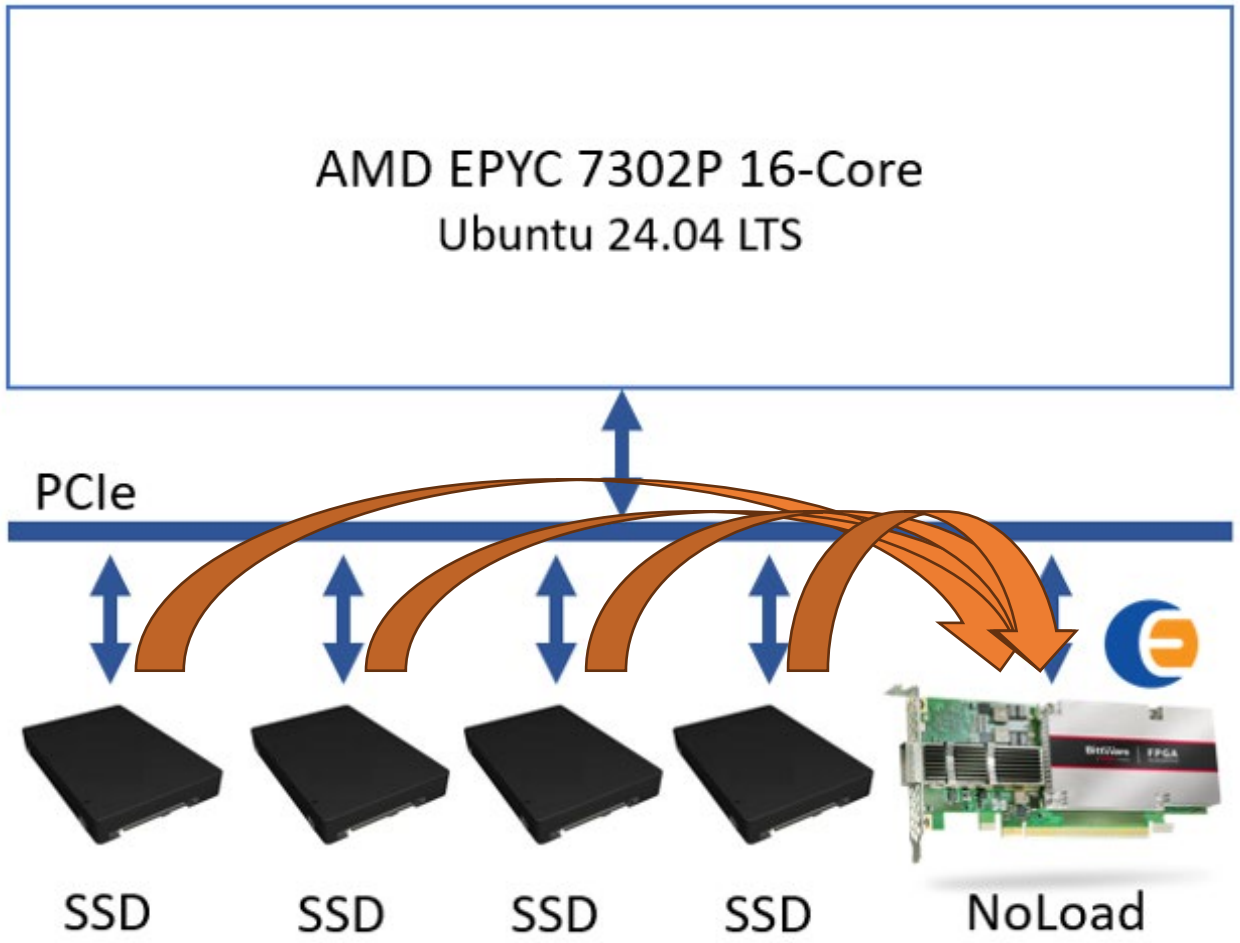
Testing the Framework – SSD <-> SSD



- Consider a test to use the new Userspace interface
- Generate traffic using the DMA engines of the SSDs targeting the NoLoad CMB using P2PDMA
- FIO will generate the traffic
  - We will send *Read* commands to the SSDs with the output data buffer being the NoLoad CMB (via P2PDMA)
  - To enable P2PDMA userspace with FIO we simply change the *iomem* variable as shown below:

```
odin ~/FIO $ cat p2p-read.fio
[global]
runtime=1h
time_based=1
group_reporting=1
ioengine=libaio
bs=1M
iomem=mmapshared:/sys/class/nvme/nvme2/device/p2pmem/allocate
iodepth=16
direct=1
zero_buffers=1

[p2pmem-read]
rw=read
numjobs=16
```

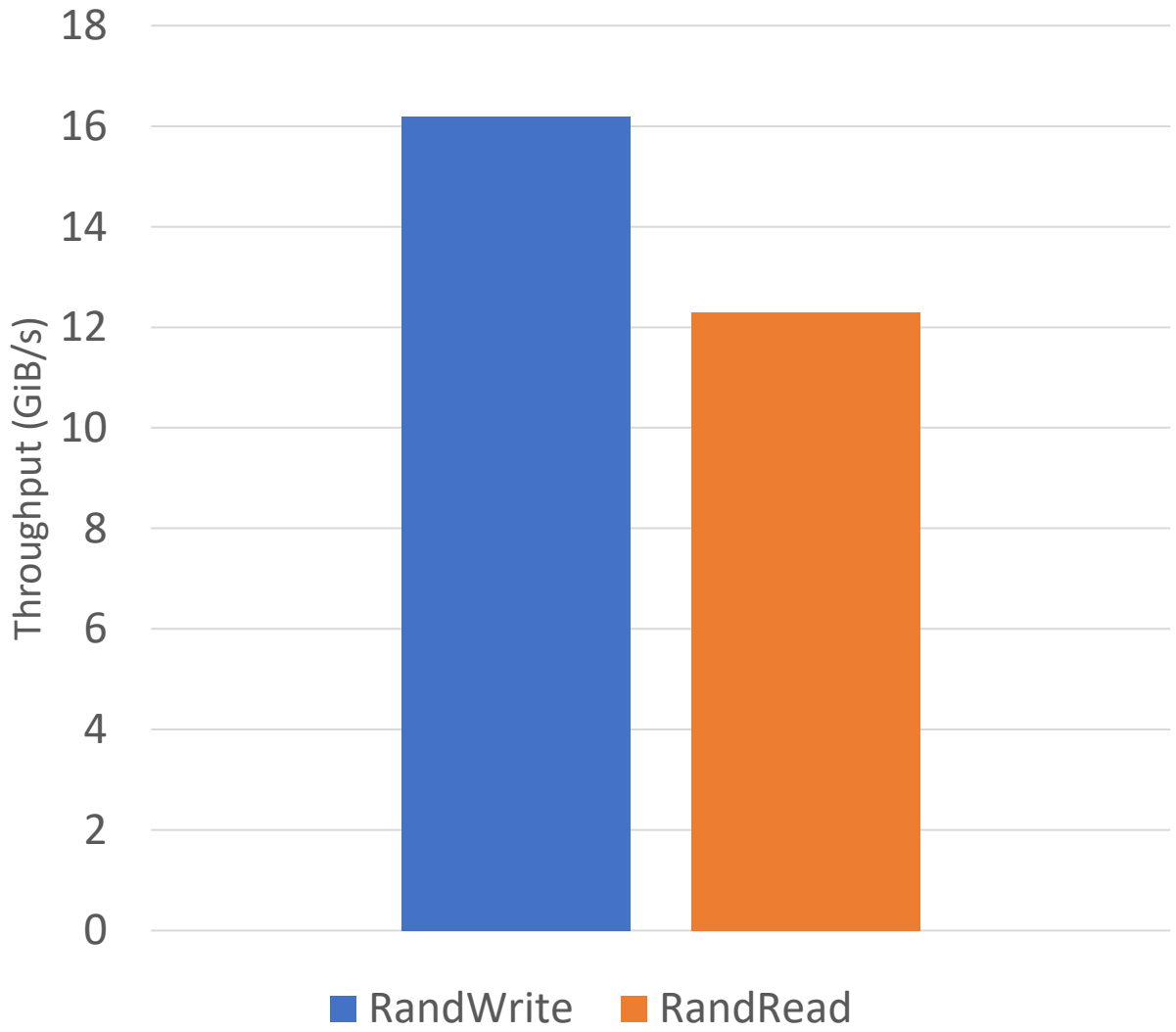
Testing the Framework – SSD <-> SSD



- Consider a test to use the new Userspace interface
- Generate traffic using the DMA engines of the SSDs targeting the NoLoad CMB using P2PDMA
- FIO will generate the traffic
  - We will send *Read* commands to the SSDs with the output data buffer being the NoLoad CMB (via P2PDMA)
  - To enable P2PDMA userspace with FIO we simply change the *iomem* variable as shown below:

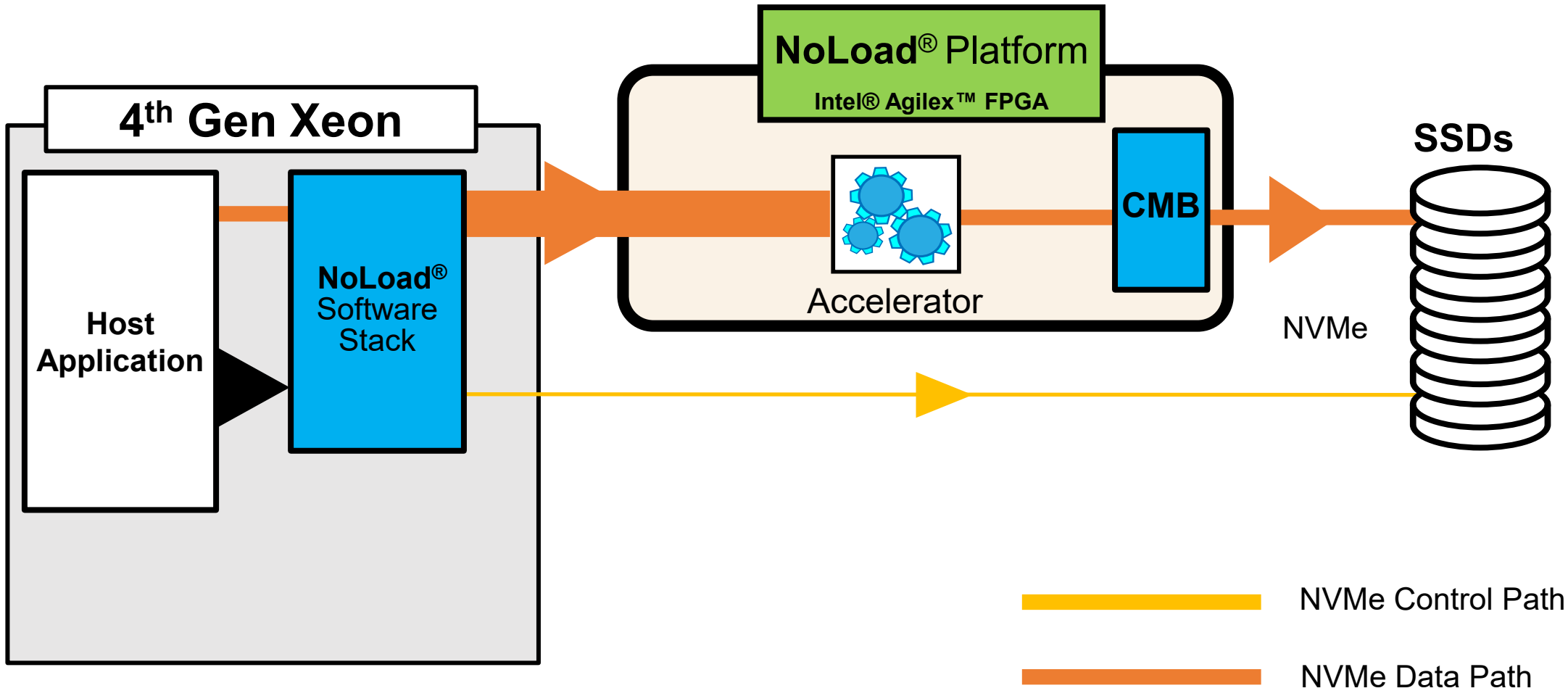
```
odin ~/FIO $ cat p2p-read.fio
[global]
runtime=1h
time_based=1
group_reporting=1
ioengine=libaio
bs=1M
iomem=mmapshared:/sys/class/nvme/nvme2/device/p2pmem/allocate
iodepth=16
direct=1
zero_buffers=1

[p2pmem-read]
rw=read
numjobs=16
```

Testing the Framework – SSD <-> SSD Results

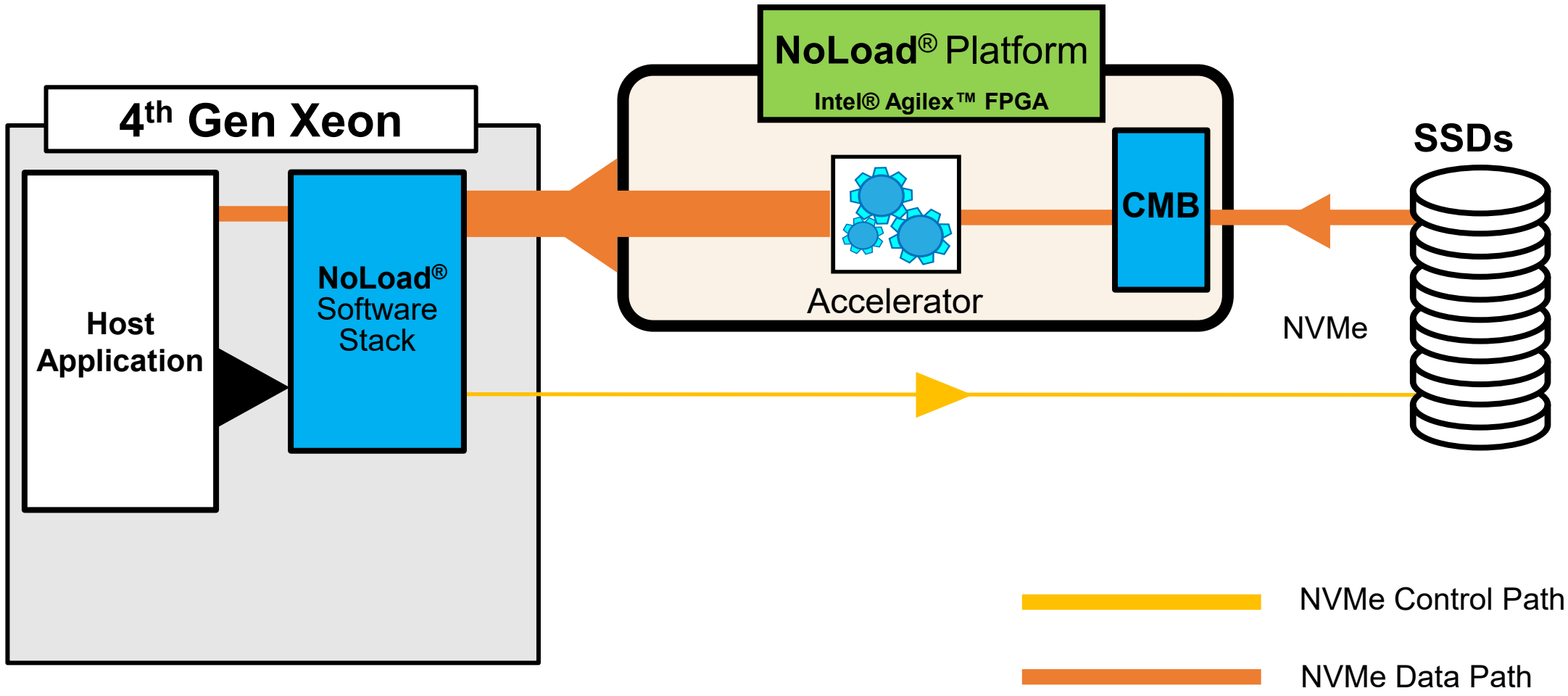


- P2PDMA was able to saturate the NoLoad PCIe bus (gen4x8)
- No host DRAM usage
- Verified using built-in NoLoad counters
- Performance was identical using P2PDMA or using host DRAM

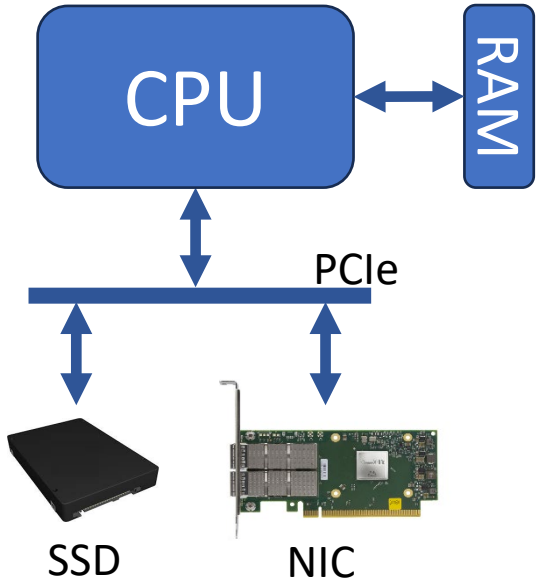
P2PDMA Use Case – SSD <-> Accelerator Compression



P2PDMA Use Case – SSD <-> Accelerator Decompression

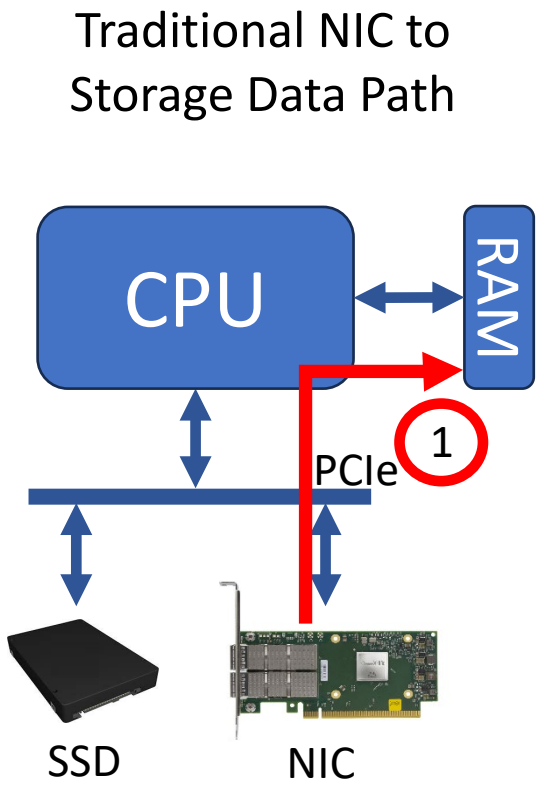


Traditional NIC to Storage Data Path



Traditional NIC Data Capture Path

- Consider the Traditional NIC data capture with a (many) SSDs and a NIC on the PCIe bus

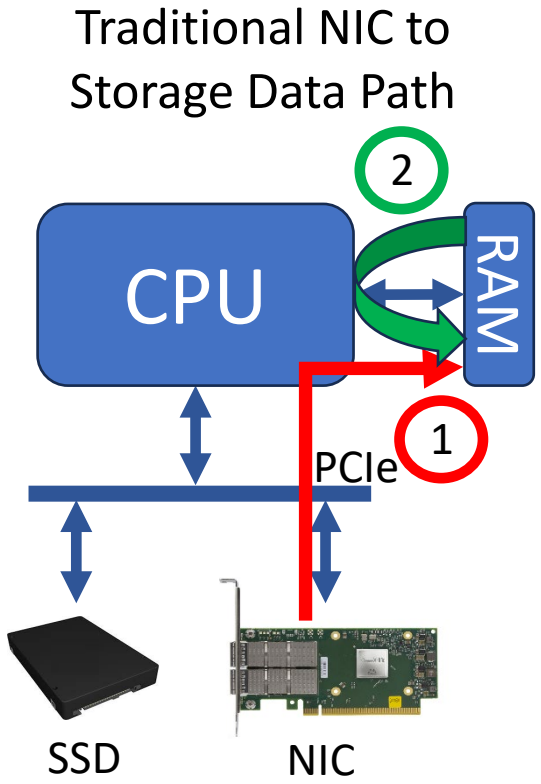


Traditional NIC Data Capture Path

- Consider the Traditional NIC data capture with a (many) SSDs and a NIC on the PCIe bus

1. Data comes in on the NIC and is sent via DMA to the host DRAM where it is released to the application.

P2PDMA Usecase NIC Data Capture

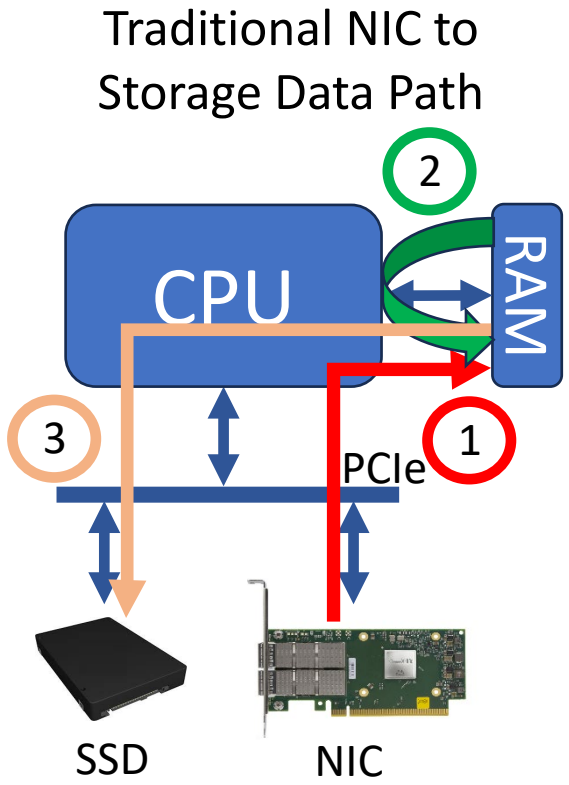


Traditional NIC Data Capture Path

- Consider the Traditional NIC data capture with a (many) SSDs and a NIC on the PCIe bus

1. Data comes in on the NIC and is sent via DMA to the host DRAM where it is released to the application.
2. The application performs some process on the data (i.e. compression)

P2PDMA Usecase NIC Data Capture



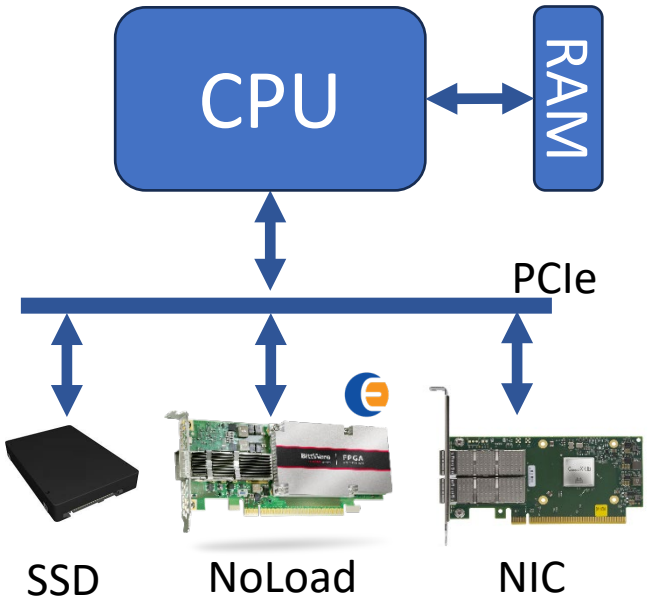
Traditional NIC Data Capture Path

- Consider the Traditional NIC data capture with a (many) SSDs and a NIC on the PCIe bus

1. Data comes in on the NIC and is sent via DMA to the host DRAM where it is released to the application.
2. The application performs some process on the data (i.e. compression)
3. The application then writes the data to the storage SSD

P2PDMA Usecase NIC Data Capture

NIC to Storage Data Path
with NoLoad and P2PDMA

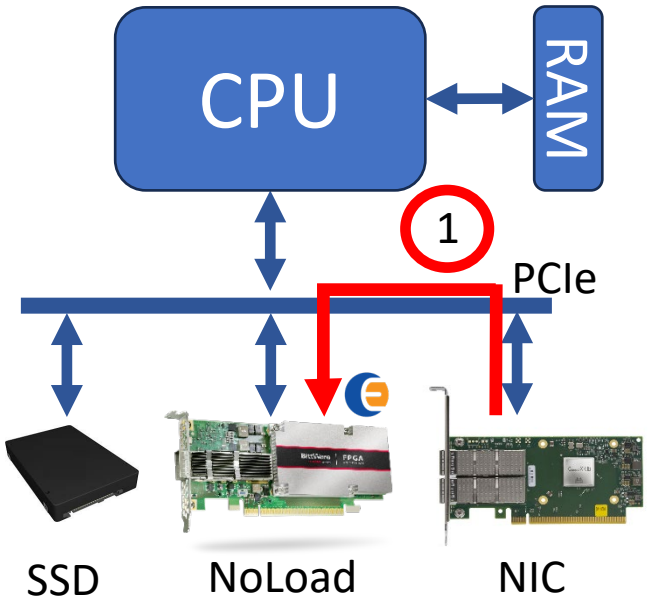


NIC Data Capture Path w/ NoLoad P2P

- Consider a setup with SSDs, a NoLoad CSP Accelerator with compression, and a high speed NIC

P2PDMA Usecase NIC Data Capture

NIC to Storage Data Path
with NoLoad and P2PDMA



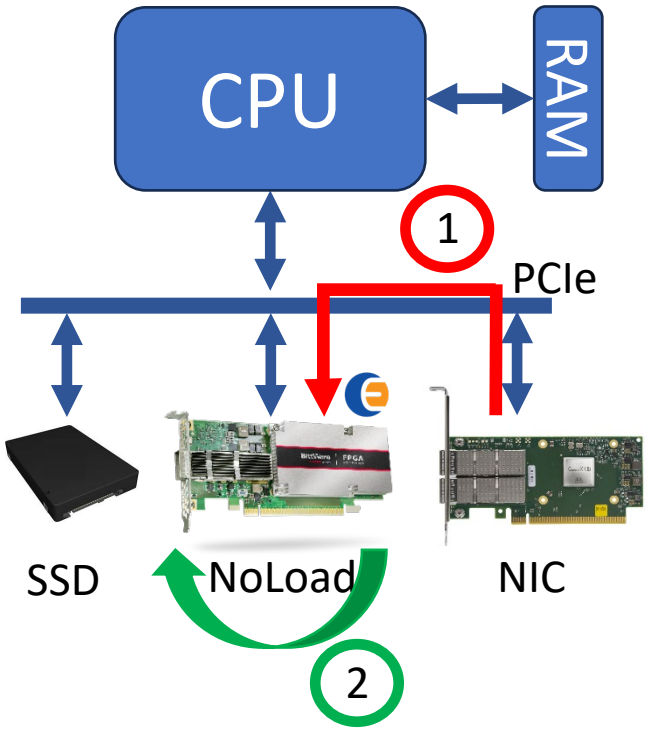
NIC Data Capture Path w/ NoLoad P2P

- Consider a setup with SSDs, a NoLoad CSP Accelerator with compression, and a high speed NIC

1. Data is sent via DMA to the CMB on NoLoad from the NIC (via P2PDMA)

P2PDMA Usecase NIC Data Capture

NIC to Storage Data Path
with NoLoad and P2PDMA



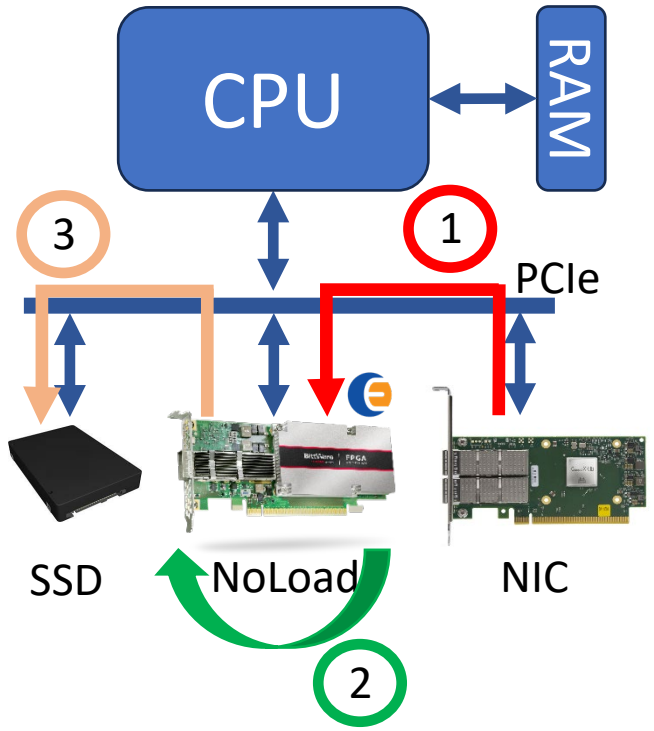
NIC Data Capture Path w/ NoLoad P2P

- Consider a setup with SSDs, a NoLoad CSP Accelerator with compression, and a high speed NIC

1. Data is sent via DMA to the CMB on NoLoad from the NIC (via P2PDMA)
2. NoLoad compresses the data at line rate (100 Gbps ingest)

P2PDMA Usecase NIC Data Capture

NIC to Storage Data Path
with NoLoad and P2PDMA



NIC Data Capture Path w/ NoLoad P2P

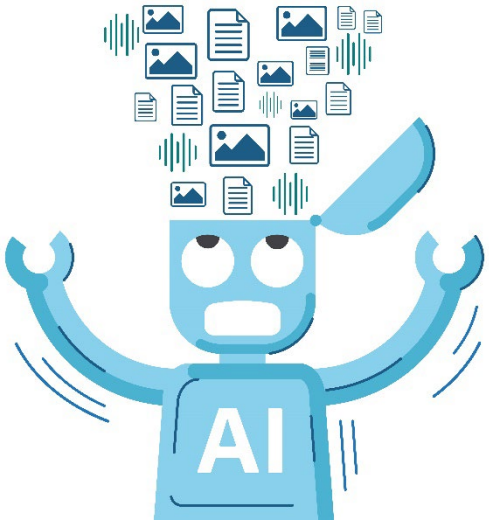
- Consider a setup with SSDs, a NoLoad CSP Accelerator with compression, and a high speed NIC

1. Data is sent via DMA to the CMB on NoLoad from the NIC (via P2PDMA)
2. NoLoad compresses the data at line rate (100 Gbps ingest)
3. Data is sent via DMA to the SSD storage

Benefits

- Access to inline acceleration (Compression)
- Eliminates any DRAM bottlenecks
- Reduces CPU load by offloading compression
- Lower latency to the SSD (due to dedicated compression hardware)

Other Possible Applications



- AI applications
  - Feeding a training engine or lowering the latency of inference
- Data analytics/analysis
  - Hardware offload (or network offload) of analytics
- Data capture/collection
  - Direct NIC to storage connection
  - Opensource equivalent to GPUDirect-to-Storage
- Any connection where a transfer between two PCIe devices exists

Summary

P2PDMA Status

- P2PDMA is an open-source upstream framework for peer-to-peer transactions
- Userspace interface has been upstreamed into Linux v6.2
- As of Ubuntu 24.04 LTS, P2PDMA is now enabled by default on all deployments (along with the userspace interface)

P2PDMA Testing

- Showed full bandwidth P2PDMA traffic
- P2PDMA can be tested with minimal application changes
  - Using FIO or even perftests like ib_read_bw*

Use Cases

- P2PDMA is an excellent framework when considering the data capture use case when paired with a computational storage accelerator such as the NoLoad compression
  - Lower overall latency is achieved with hardware acceleration
  - Improved CPU efficiency is observed

*Currently being upstreamed

Next Steps

- Complete support into upstream RDMA framework
- Continue pushing companies for p2pdma integration (i.e. NICs, GPUs, Accelerator Cards, etc)
  - Upstream support in drivers
- Continue to integrate test infrastructure into modern CPUs and motherboards

For More Information

- Check out our latest P2PDMA blog posts for more information