



STORAGE DEVELOPER CONFERENCE

SNIA ■ SANTA CLARA, 2016

Enabling Remote Access to Persistent Memory on an IO Subsystem using NVM Express and RDMA

Stephen Bates, PhD

Microsemi

Otherwise Known As



STORAGE DEVELOPER CONFERENCE

SNIA ■ SANTA CLARA, 2016

iopmem: pmem for MMIO

Stephen Bates, PhD

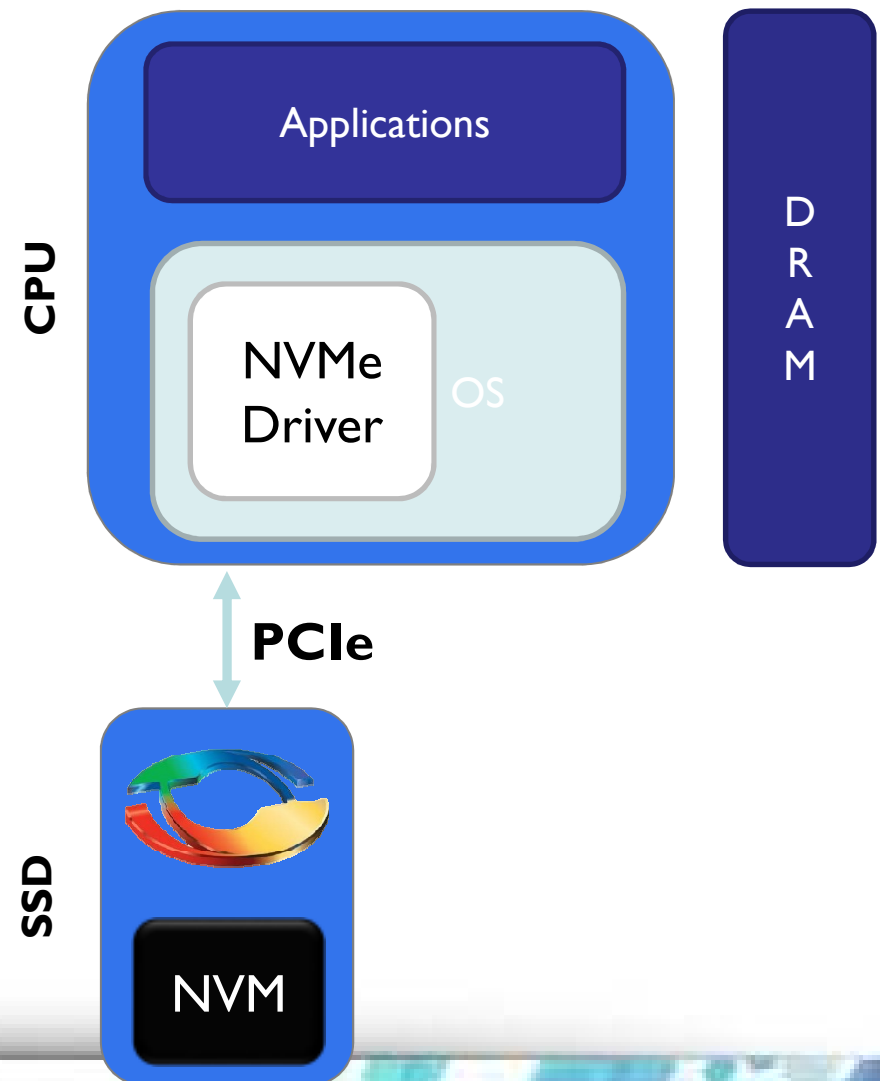
Microsemi

Motivation

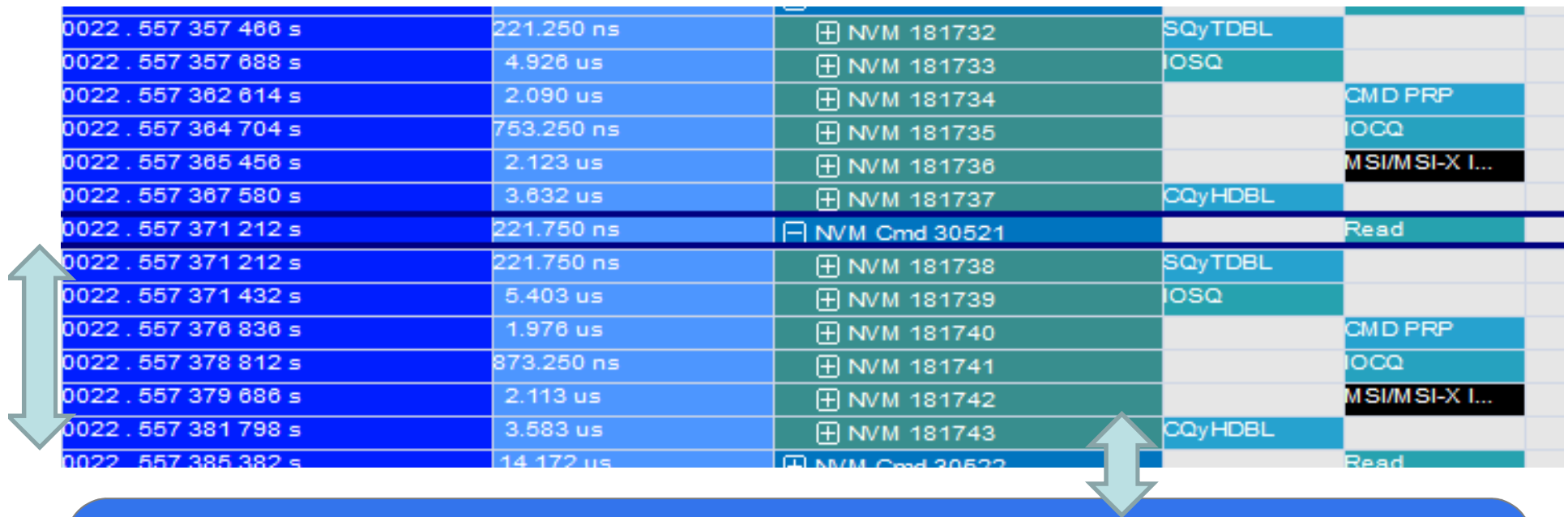
- ❑ PCIe IO devices are getting faster.
- ❑ Almost always these PCIe devices have either a high performance DMA engine, a number of exposed PCIe BARs or both.
- ❑ Until this work, (almost) any high-performance transfer of information between two PCIe devices has required the use of a buffer in system memory.
- ❑ With this patch to Linux the bandwidth between CPU cores and system memory is not compromised when high-throughput transfers occurs between PCIe devices.

PCIe NVM Express 101

- ❑ NVMe defines a PCIe memory regions and the fields within that region.
- ❑ The CPU can mmap/ioremap those regions and then request IO.
- ❑ Admin commands used at boot/probe time to configure queues and doorbells.



How NVMe Express Works

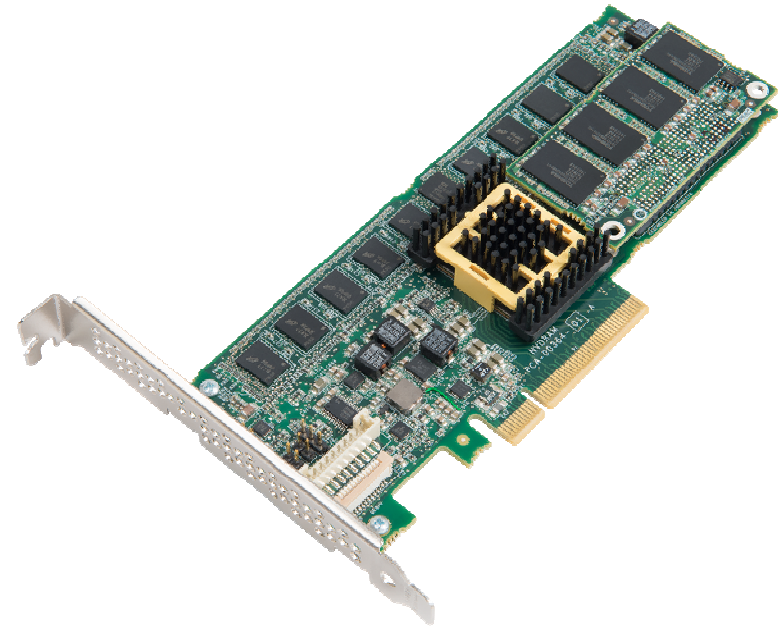


0022.557357488 s	221.250 ns	⊕ NVM 181732	SQyTDBL		
0022.557357688 s	4.926 us	⊕ NVM 181733	IOSQ		
0022.557362614 s	2.090 us	⊕ NVM 181734		CMD PRP	
0022.557364704 s	753.250 ns	⊕ NVM 181735		IOCQ	
0022.557365456 s	2.123 us	⊕ NVM 181736		MSI/MSI-X I...	
0022.557367580 s	3.632 us	⊕ NVM 181737	CQyHDBL		
0022.557371212 s	221.750 ns	⊖ NVM Cmd 30521		Read	
0022.557371212 s	221.750 ns	⊕ NVM 181738	SQyTDBL		
0022.557371432 s	5.403 us	⊕ NVM 181739	IOSQ		
0022.557376836 s	1.976 us	⊕ NVM 181740		CMD PRP	
0022.557378812 s	873.250 ns	⊕ NVM 181741		IOCQ	
0022.557379686 s	2.113 us	⊕ NVM 181742		MSI/MSI-X I...	
0022.557381798 s	3.583 us	⊕ NVM 181743	CQyHDBL		
0022.557385382 s	14.172 us	⊖ NVM Cmd 30522		Read	

The anatomy of a single NVMe Read command.
~10us total time for QD=1 NVMe Read

The Hardware

- ❑ NVMe compliant DRAM based SSD.
- ❑ Presents to host as a NVMe device (see next slide).
- ❑ Also can present some or all of the DRAM as a PCIe BAR (aka NVMe CMB).



**Shipping in
volume today!**

The Hardware

```
batesste@cgy1-donard:~/qemu-minimal$ sudo lspci -vvv -s 01:00.0
```

01:00.0 Non-Volatile memory controller: PMC-Sierra Inc. Device f117 (rev 06) (prog-if 02 [NVM Express])

Subsystem: PMC-Sierra Inc. Device f117

Control: I/O+ Mem+ BusMaster+ SpecCycle- MemWINV- VGASnoop- ParErr- Stepping- SERR- FastB2B- DisINTx+

Status: Cap+ 66MHz- UDF- FastB2B- ParErr- DEVSEL=fast >TAbort- <TAbort- <MAbort- >SERR- <PERR- INTx-

Latency: 0, Cache Line Size: 64 bytes

Interrupt: pin A routed to IRQ 16

Region 0: Memory at dfa00000 (64-bit, non-prefetchable) [size=16K]

Region 4: Memory at 381f80000000 (64-bit, prefetchable) [size=1G]

Standard NVMe BAR,
standard driver maps this.

1GB CMB we can use as
we see fit.

NVMe Controller Memory Buffers

- ❑ NVMe specification defines how to advertise a CMB.
- ❑ A CMB can be used for a variety of things.
- ❑ In this case we use it as a staging area for write/read data.
- ❑ NVMe standard is looking at adding persistence to CMBs.

3.1.11 Offset 38h: CMBLOC – Controller Memory Buffer Location

This optional register defines the location of the Controller Memory Buffer (refer to section 4.7). If CMBSZ is 0, this register is reserved.

Bit	Type	Reset	Description
31:12	RO	Impl Spec	Offset (OFST): Indicates the offset of the Controller Memory Buffer in multiples of the Size Unit specified in CMBSZ. This value shall be 4KB aligned.
11:3	RO	0h	Reserved
2:0	RO	Impl Spec	Base Indicator Register (BIR): Indicates the Base Address Register (BAR) that contains the Controller Memory Buffer. For a 64-bit BAR, the BAR for the lower 32-bits of the address is specified. Values 0h, 2h, 3h, 4h, and 5h are valid.

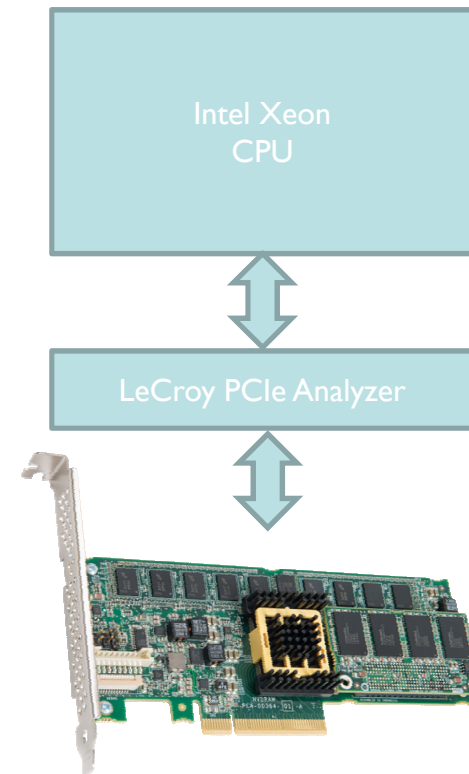
3.1.12 Offset 3Ch: CMBSZ – Controller Memory Buffer Size

This optional register defines the size of the Controller Memory Buffer (refer to section 4.7). If the controller does not support the Controller Memory Buffer feature then this register shall be cleared to 0h.

Bit	Type	Reset	Description																		
31:12	RO	Impl Spec	Size (SZ): Indicates the size of the Controller Memory Buffer available for use by the host. The size is in multiples of the Size Unit. If the Offset + Size exceeds the length of the indicated BAR, the size available to the host is limited by the length of the BAR.																		
11:8	RO	Impl Spec	Size Units (SZU): Indicates the granularity of the Size field.																		
			<table><tr><th>Value</th><th>Granularity</th></tr><tr><td>0h</td><td>4 KB</td></tr><tr><td>1h</td><td>64 KB</td></tr><tr><td>2h</td><td>1 MB</td></tr><tr><td>3h</td><td>16 MB</td></tr><tr><td>4h</td><td>256 MB</td></tr><tr><td>5h</td><td>4 GB</td></tr><tr><td>6h</td><td>64 GB</td></tr><tr><td>7h – Fh</td><td>Reserved</td></tr></table>	Value	Granularity	0h	4 KB	1h	64 KB	2h	1 MB	3h	16 MB	4h	256 MB	5h	4 GB	6h	64 GB	7h – Fh	Reserved
			Value	Granularity																	
			0h	4 KB																	
			1h	64 KB																	
			2h	1 MB																	
			3h	16 MB																	
			4h	256 MB																	
			5h	4 GB																	
6h	64 GB																				
7h – Fh	Reserved																				
7:5	RO	0h	Reserved																		
4	RO	Impl Spec	Write Data Support (WDS): If this bit is set to '1', then the controller supports data and metadata in the Controller Memory Buffer for commands that transfer data from the host to the controller (e.g., Write). If this bit is cleared to '0', then all data and metadata for commands that transfer data from the host to the controller shall be transferred from host memory.																		

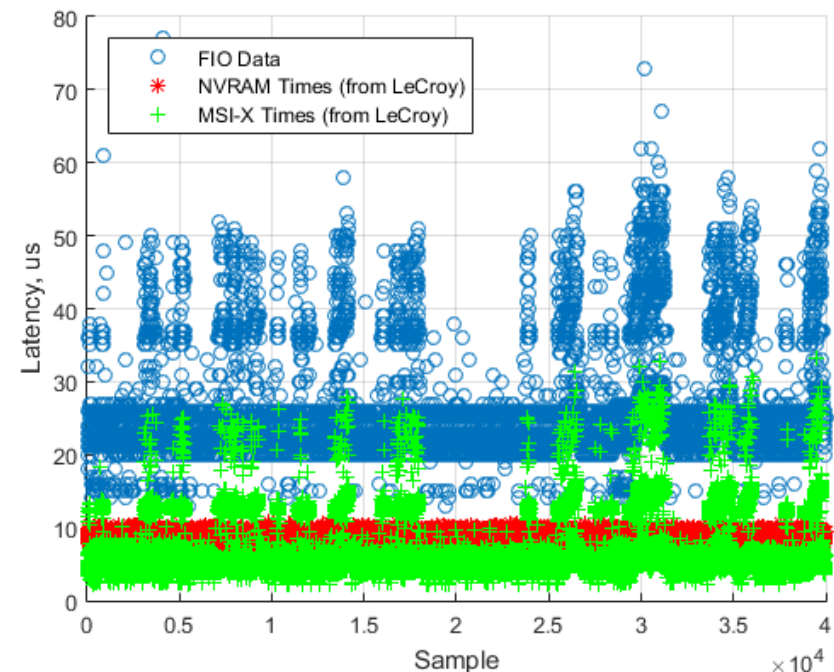
NVMe Latency Analysis

- ❑ When latency is below 20us it becomes interesting to understand what is contributing to that latency.
- ❑ What part is due to the SSD and what part is due to the OS and CPU architecture.
- ❑ We did some measurements to find out.



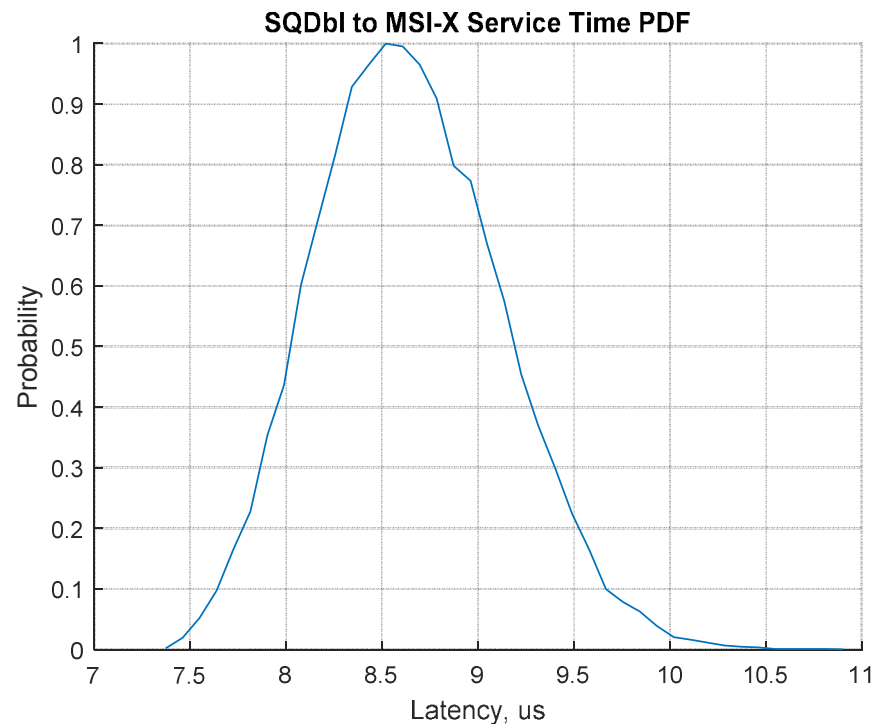
Latency Analysis

- ❑ The IO time contribution from the NVRAM is very consistently about 8-9us (see next slide).
- ❑ The IO time measured at the application is ~10us more. This is mostly software overhead.
- ❑ The application overhead is also more “spikey”. Why is that?



Latency Analysis

- ❑ This is the SSD IO time as a distribution.
- ❑ Very tight distribution with a mean of under 9us.
- ❑ This SSD is very consistent.



Linux Kernel Background

- ❑ There have been some recent additions to the kernel in preparation of new memory types on the memory channel:
 - ❑ ZONE_DEVICE– The ability to associate a range of memory addresses (PFNs) with a specific driver and not allocate them to system memory.
 - ❑ PMEM – A ZONE_DEVICE device driver that takes a memory region and exposes it to the rest of the OS as a DAX enabled block device.
 - ❑ DAX – A framework that allows a memory addressable block device to bypass the page-cache. Allows supporting filesystems (like ext4) to be placed on these block devices.
 - ❑ STRUCT PAGE SUPPORT – PMEM devices can (optionally) include struct page backing so they can be used for things like DMA.

How pmem.c works

- ❑ User reserves a memory range using a kernel boot option (e.g. memmap=2G!14G).
- ❑ The pmem driver binds to this reserved memory region and presents it to user-space as a /dev/pmem0 block device.
- ❑ The user can put filesystems on the block device and then run IO to/from it.

PMEM Performance

	QD=1, T=1	QD=128, T=1	QD=128, T=8
PMEM	3.0us (1050MB/s)	520us (1050MB/s)	510us (7500MB/s)
NVMe (DRAM)	16.0us (228MB/s)	765us (663MB/s)	959us (3980MB/s)
NVMe (NAND)	88us (42MB/s)	794us (64MB/s)	2554 (1544MB/s)

Gory Details: Linux cgy1-donard 4.6.0+3-00004-ga573b70 #118 SMP Fri Jun 3 15:21:30 MDT 2016 x86_64 GNU/Linux. 8 cores on two sockets, Intel(R) Xeon(R) CPU E5-2609 0 @ 2.40GHz, memmap=2G!14G, fio-2.2.10-17-g217b0, 512B random read, direct IO, libaio engine, DDR3 4GB DIMMs @ 8+1 Bytes @ 1066MHz.

15

iopmem [1]

- ❑ Our proposal is an 88 line change to the kernel that enables struct page support for ZONE_DEVICE memory that resides in IO memory space.
- ❑ To provide an example usage of our change we provide an example driver (iopmem.c) that can be bound to any PCIe BAR in the system.
- ❑ Note we are NOT proposing iopmem.c be added to the kernel (though it could be). It would be assumed that PCIe device vendors would update their drivers (e.g. nvme) using our example if they wished to avail of the services provided by the change.

[1] <https://github.com/sbates130272/linux-donard/tree/iopmem-rfc>

iopmem.c: Example Driver [1]

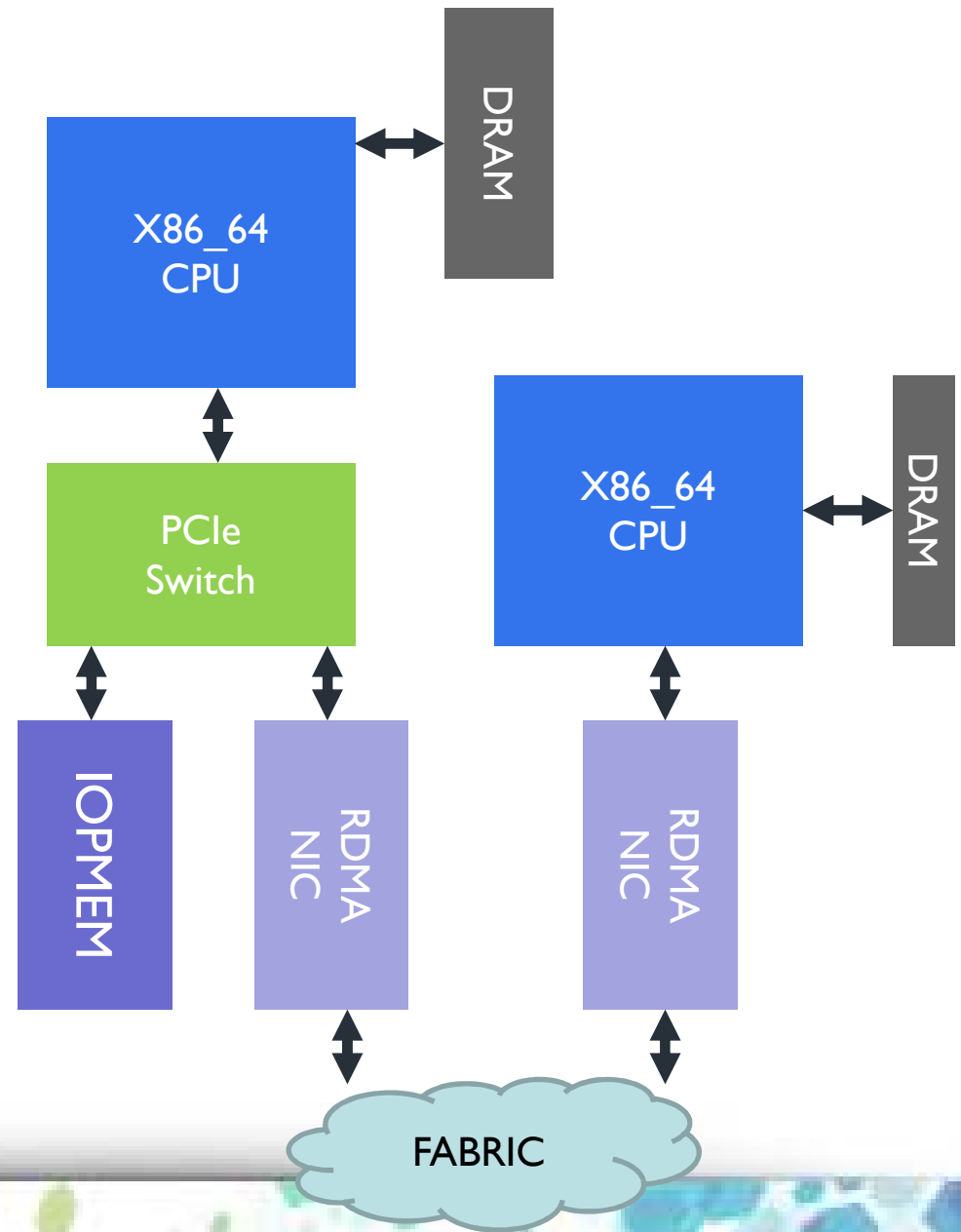
- ❑ A self-contained PCIe driver. Note that this driver binds to the device so it WILL replace any other driver for that device (e.g. NVMe).
- ❑ Has a module parameter to identify which of the BARs you want to utilize. For now, the entire BAR is used.
- ❑ The driver registers a DAX enabled block device and presents this to the OS. The size of the block device is the size of the BAR.
- ❑ The driver also creates a character device and this can be mmap()ed. This would be for advanced users who do their file layout in user-space.

[1] <https://github.com/sbates130272/iopmem>

RDMA <-> IOPMEM

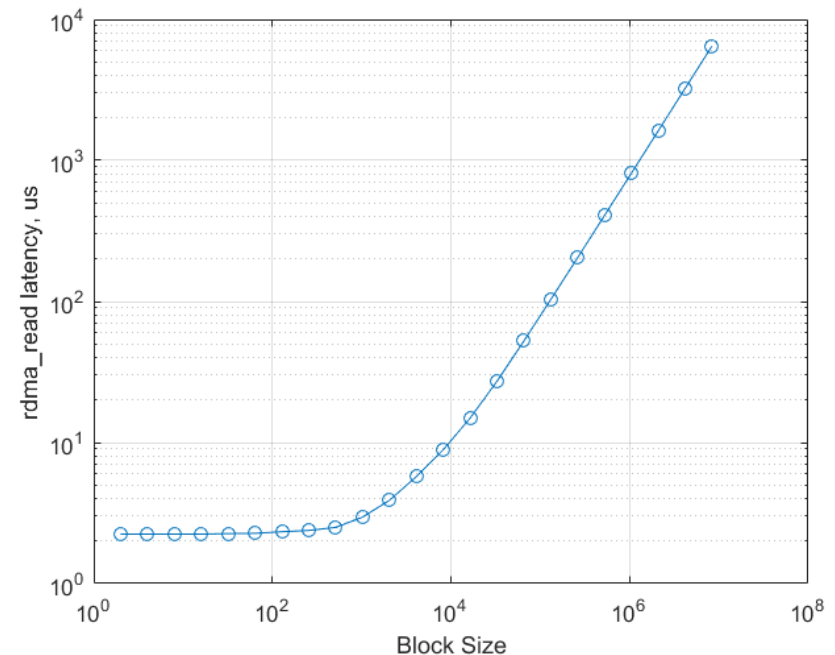
- ❑ The DMA engines in the RDMA device can target the BAR on the IOPMEM device.
- ❑ Can either use the mmap() on the IOPEM direct OR mmap() files on a DAX mounted filesystem as the RDMA Memory Regions.
- ❑ Our system achieved 4GB/s IOPMEM writes and 1.2GB/s IOPMEM reads (limits of our IOPMEM hardware)

Tested using IB, iWARP and RoCE!



RDMA <-> IOPMEM

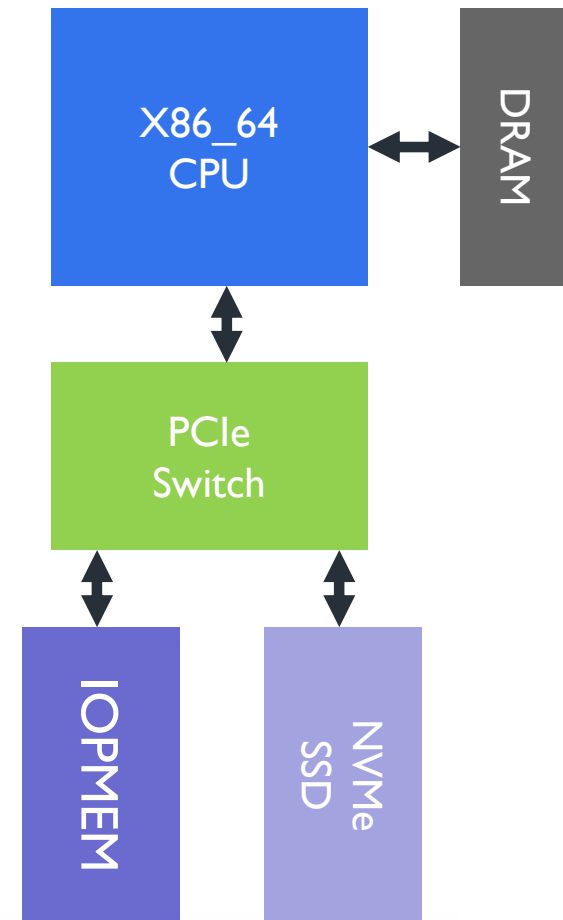
- ❑ Sub 3us read times for small access sizes.
- ❑ CX4, RoCE and no RDMA switch.
- ❑ Latency rises for larger accesses. Note these are unaligned and byte addressable.
- ❑ Results for 4KB are around 6us. This is 6us total, not incremental!



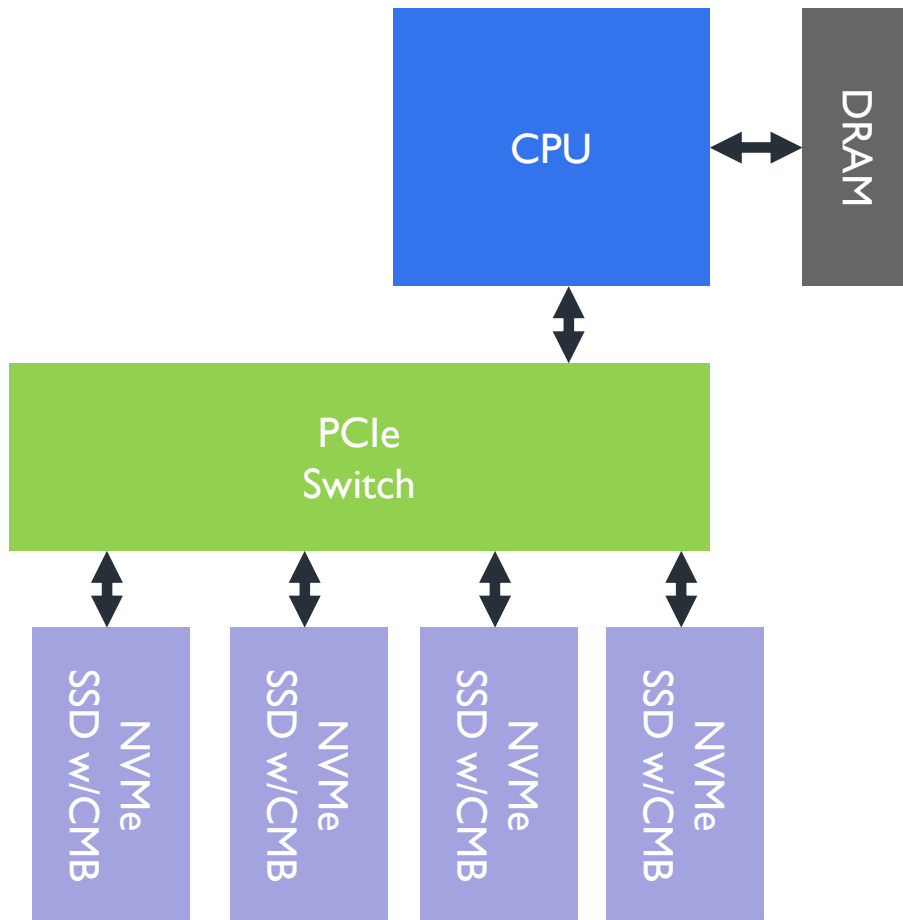
This is, in essence remote access to a NVMe Persistent CMB. i.e. PMEM over Fabrics!

NVMe <-> IOPMEM

- ❑ The DMA engines on the NVMe device can target the BAR on the IOPMEM device.
- ❑ Can either use the mmap() on the IOPMEM direct OR mmap() files on a DAX mounted filesystem.
- ❑ Our system achieved 2GB/s IOPMEM writes and 1GB/s IOPMEM reads (limits of the NVMe hardware).



Use Case 1



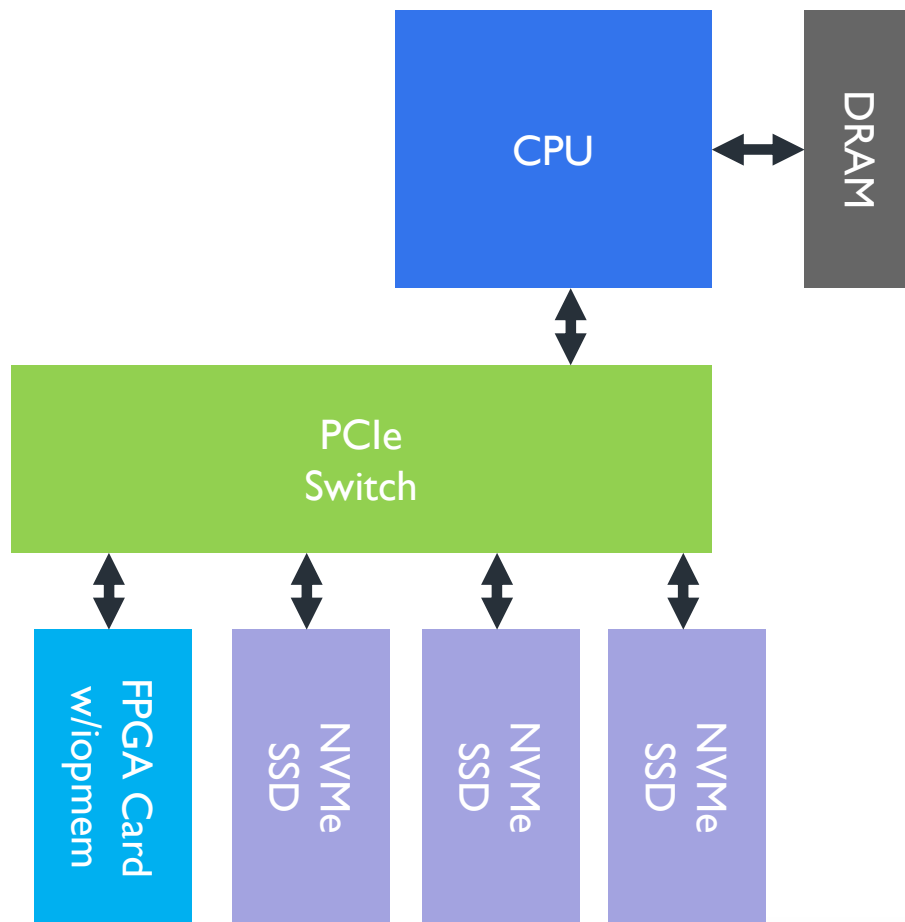
Background data copying between NVMe devices.

Can use extra processing on the NVMe SSDs to calculate RAID parity, perform dedupe etc.

Host OS still in control (it is issuing all the submission queue entries) but now all DMA traffic is below the PCIe switch.

CPU<->DRAM bandwidth is not compromised by DMA traffic. Frees up memory space and bandwidth for other tasks.

Use Case 2



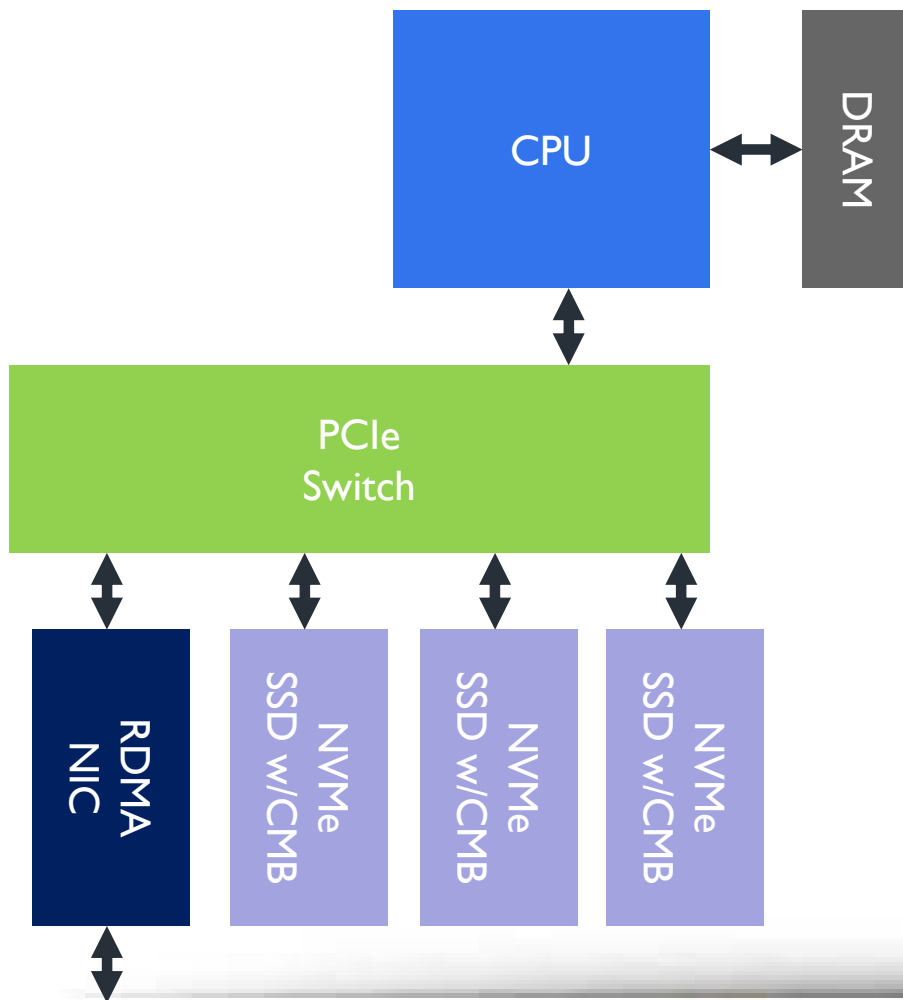
Use FPGA with exposed BAR as iopmem device.

Now it can receive data from standard NVMe SSDs and perform RAID, encryption, dedupe, data search etc.

The FPGA device could even present to the OS as a NVMe device with a CMB! A vendor specific command set could define the acceleration functions.

Again all DMA traffic sits below PCIe switch, offloading the CPU.

Use Case 3

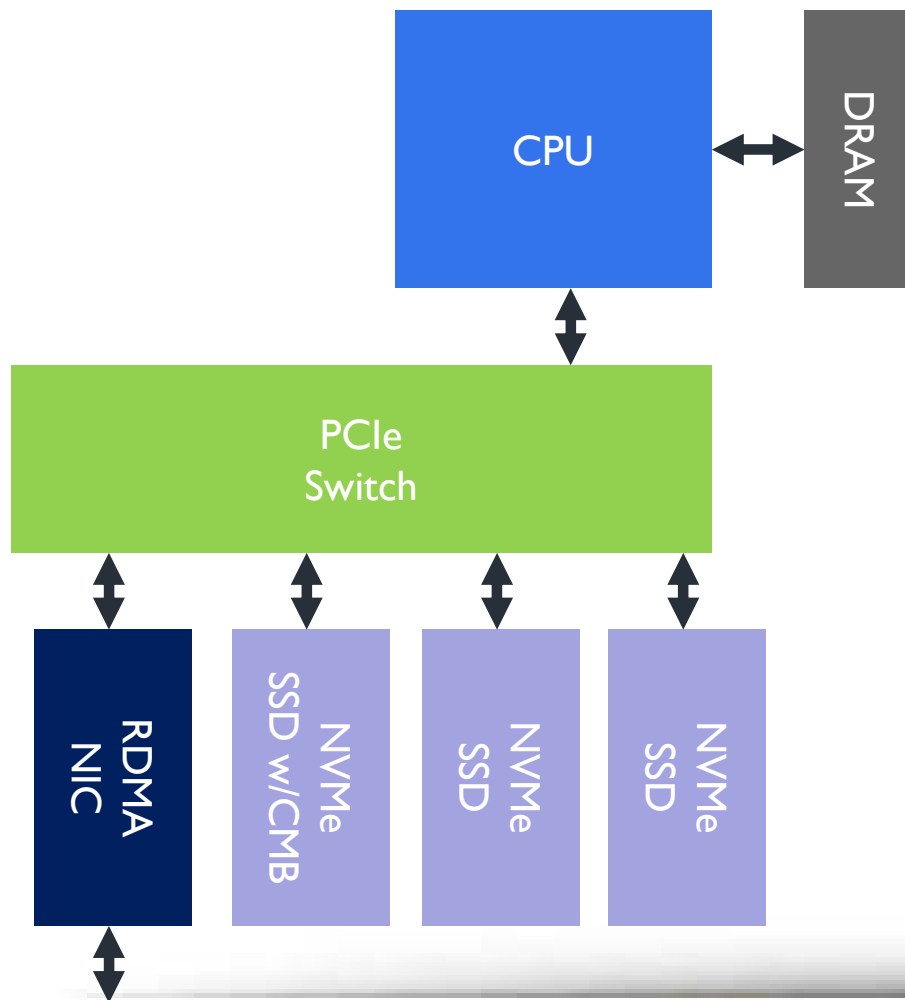


RDMA NIC can push data direct to NVMe CMBs.

Offloads CPU memory subsystem and reduces NVMf latency.

Could be used to improve performance in a NVMe over Fabrics target system.

Use Case 4

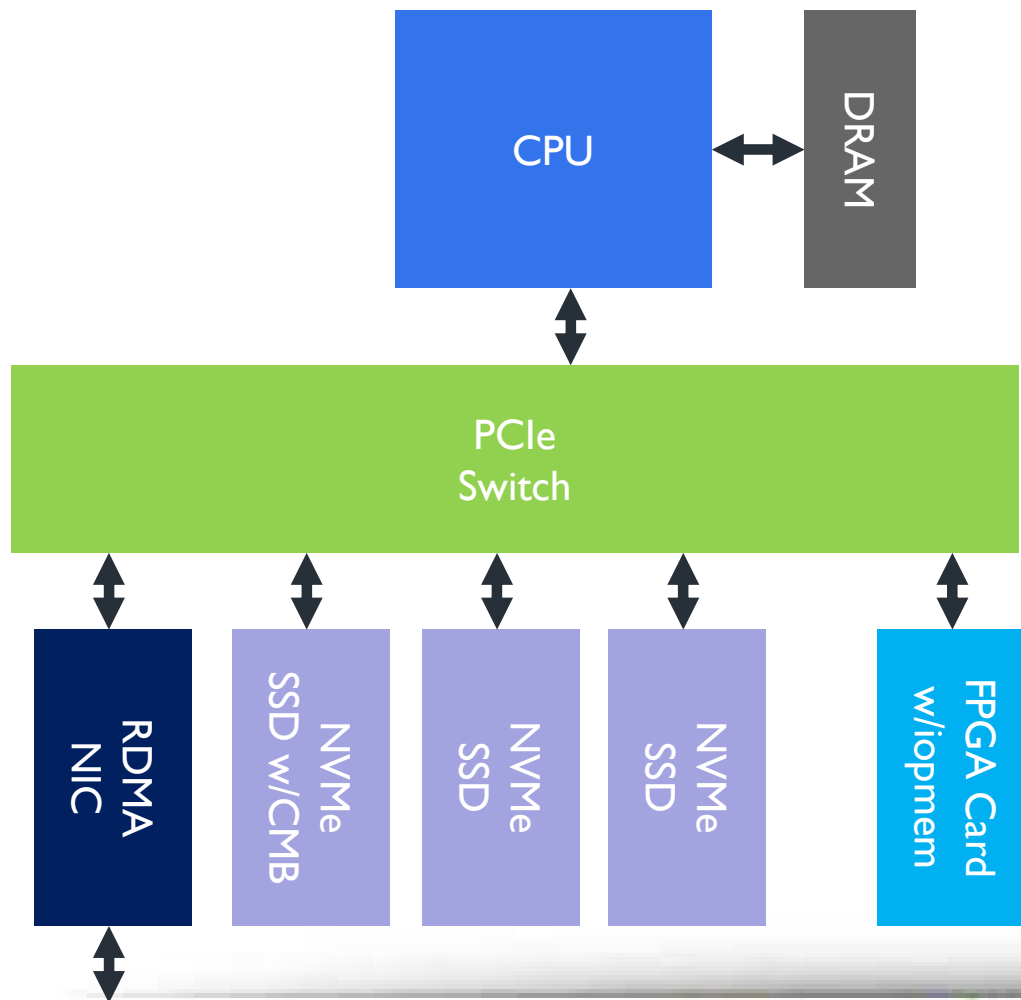


RDMA NIC can push data direct to one NVMe w/CMBs. This SSD works as a write-back cache.

Data is then lazily copied out of the NVMe SSD w/CMB onto standard NVMe SSDs.

Avoids the need for all SSDs to be CMB enabled (cost reduction).

Use Case 5



Like use case 4 but add the FPGA card into the mix to offload storage services from the CPU.

You get the idea. Many other permutations possible and we ain't even added any GPUs yet ;-).

Conclusions

- ❑ The Linux kernel has been adapting in preparation of new NVMs and memory attached NVM. We can leverage this work.
- ❑ We extend existing work to the PCIe IO memory subsystem and enable IO memory as a DMA target.
- ❑ Our example driver exposes IOMEM as both a DAX enabled block device and a mmap()'able region.
- ❑ We show good performance between PCIe devices and note that datapath avoids CPU when a PCIe switch is used.
- ❑ There are many interesting use cases when PCIe DMA traffic can be offloaded from the CPU!

Simple Math

NVMe
+ RDMA

AWESOME

PMEM
+ RDMA

AWESOME²

