



The Non-Volatile Memory Verbs Provider (NVP): Using the OFED Framework to access solid state storage

Bernard Metzler¹, Animesh Trivedi¹,
Lars Schneidenbach², Michele Franceschini²,
Patrick Stuedi¹, Blake Fitch²

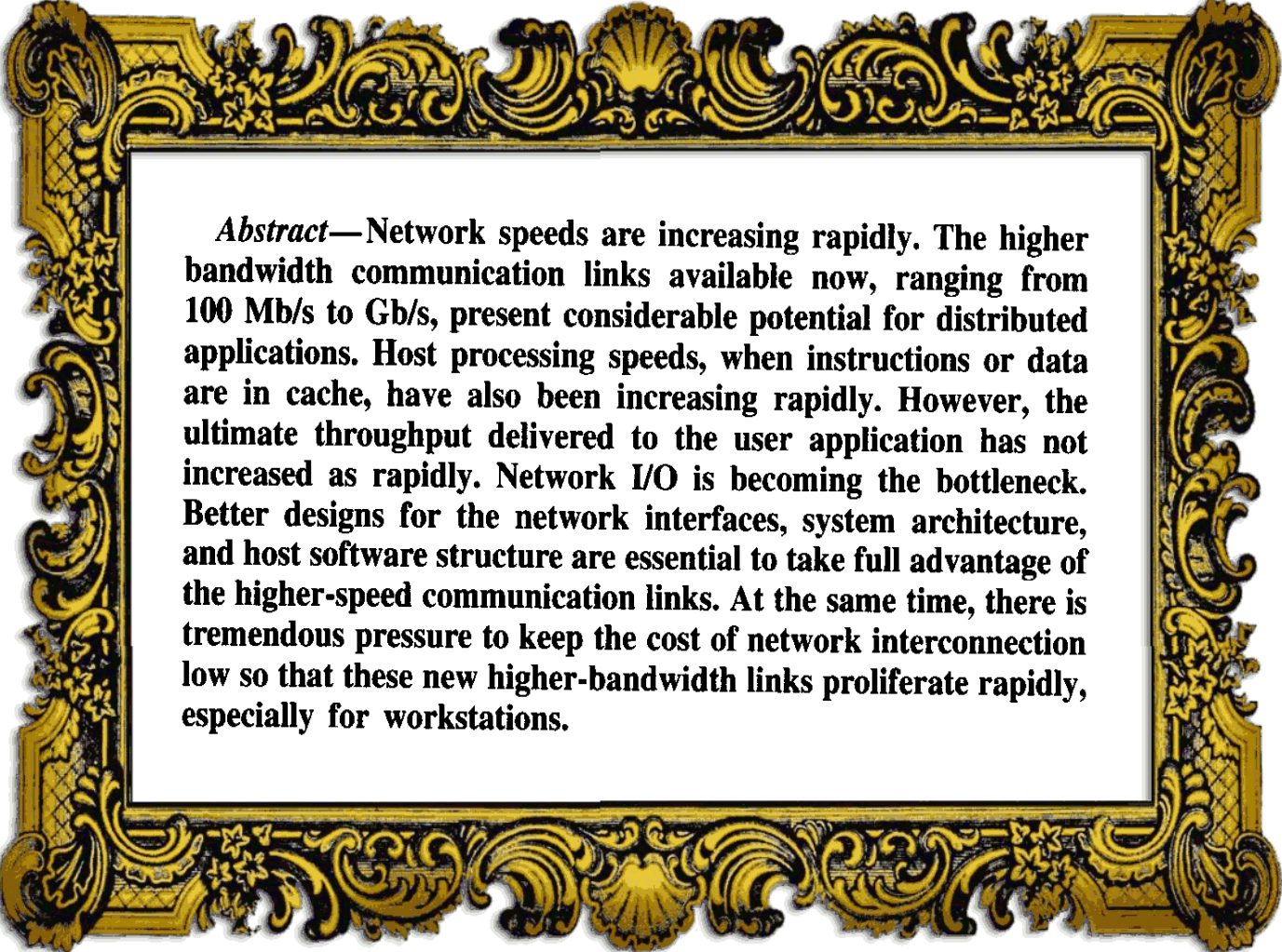
¹IBM Zurich Research, ²IBM T.J. Watson Research Center

Questions

Can we learn from the past of networking stack design to improve storage stack design?

Are there even synergies between those two stacks?

A Look back into 1993



Abstract—Network speeds are increasing rapidly. The higher bandwidth communication links available now, ranging from 100 Mb/s to Gb/s, present considerable potential for distributed applications. Host processing speeds, when instructions or data are in cache, have also been increasing rapidly. However, the ultimate throughput delivered to the user application has not increased as rapidly. Network I/O is becoming the bottleneck. Better designs for the network interfaces, system architecture, and host software structure are essential to take full advantage of the higher-speed communication links. At the same time, there is tremendous pressure to keep the cost of network interconnection low so that these new higher-bandwidth links proliferate rapidly, especially for workstations.

Present, Past and Future

	CPU Speed	Network BW	Storage BW
1980 – 2010	1000x	3000x	50x
2010 – now	1 – 1.5x	4 – 10x	10 – 100x

- Until mid-2000s, storage was slow
 - Milliseconds access time @ some MB/s, 100s of IOP's, low density, stationary + power hungry
- It has grown faster ever since
 - Now ~100 μ s access time @ 100s of MB/s, millions of IOP's, high density, mobile
- What will the future see?
 - 50ns read, 1 μ s write (PCM), ...

Learning from the Past

Yesterdays Network architecture

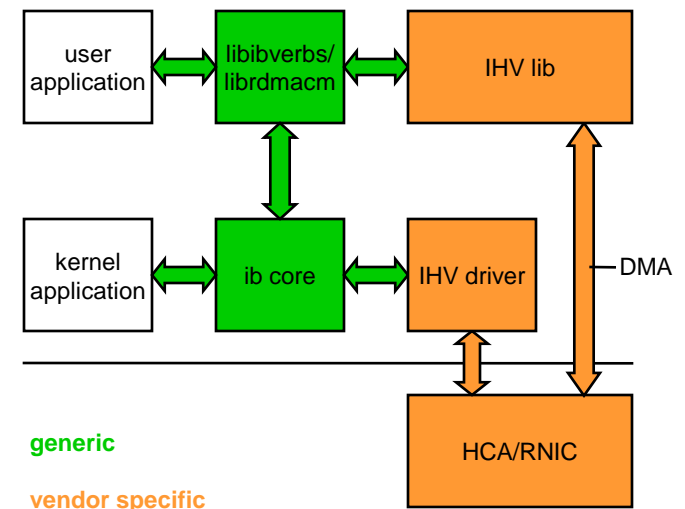
- Bottleneck for then new technologies like IB or GbE
 - Centralized, CPU orchestrated
 - CPU intense communication buffer copy
 - Unstructured byte stream data
 - Socket (file w/o seek) abstraction for access

Today's legacy IO-architecture

- Bottleneck for emerging storage technologies like NVM
 - Centralized, CPU orchestrated
 - CPU intense I/O data movement
 - Block abstraction for data
 - File abstraction for access

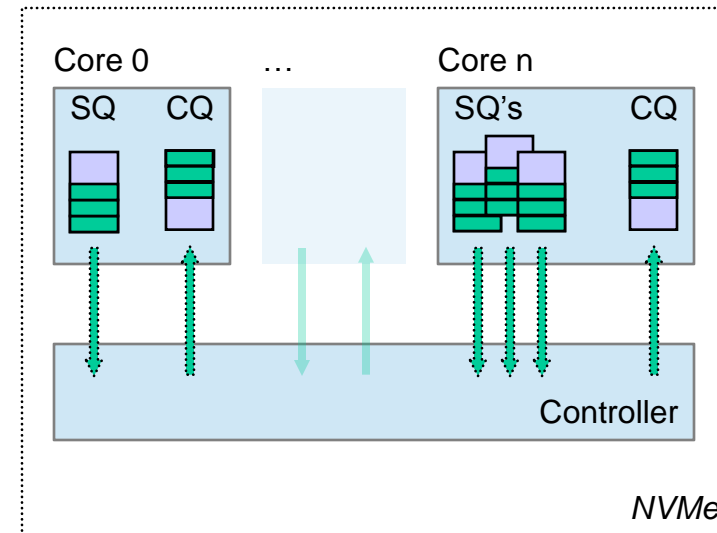
Modern Network Stacks

- ✓ Efficient host interface
 - ✓ Direct and save mapping of HW control to application
 - ✓ Application private communication channel
- ✓ Operating system bypass
 - ✓ Separation of control and data
 - ✓ Communication resource pre-allocation
 - ✓ Zero copy data movement
- ✓ Asynchronous data/completion path
- ✓ Synchronous completion support
- ✓ Rich communication semantics
 - ✓ send/receive
 - ✓ read/write/atomics
 - ✓ scatter/gather support



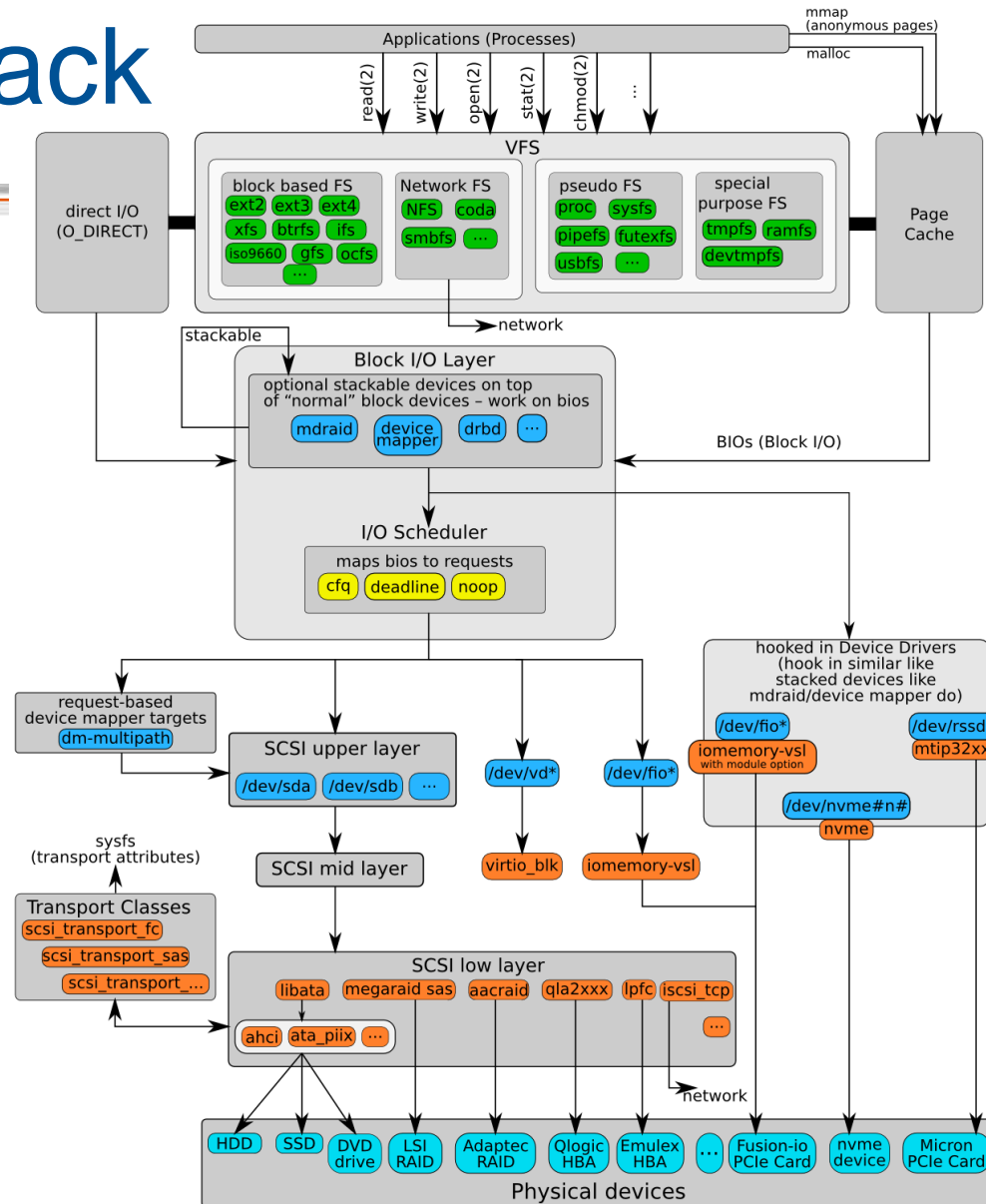
How NVM Integration evolves

- NVM as ‘fast hard disk’
 - Least intrusive/most economical
 - Keeps block access
- Host controller interfaces such as
 - AHCI (HBA), SCSIe, or
 - NVMe for PCI Express attached SSDs
 - partitioning w/multiple name spaces,
 - parallel IO, SG support,
 - up to 64K I/O Command Queues, 64K command depth, Completion Queues...,
 - interrupt mapping, interrupt coalescing, CQ polling, doorbell
- Will it be appropriate for next generation NVM technology?
 - NAND → PCM: Write 500us → 1us, Read 25us → 50ns
 - New architectures: Moneta-D, Gordon, NVHeap, FusionIO, ...



Linux I/O-Stack

- File based I/O
- Centralized Block I/O Layer
- Page Cache for efficiency
- SSD mimics NVM as HDD
 - least intrusive
 - performance limitations
- Several new NVM drivers
 - hook into BIO Layer
 - thus give block access
 - bypass legacy I/O scheduling
 - bypass SCSI layer
 - no OS bypass on fast path
 - e.g.: NVMe, Fusion-IO, Micron, ...



Efficient NW and Novel I/O Stacks



	RDMA Network	Moneta-D	Gordon	NVHeap	FusionIO
Efficient HW Access	x	x	x	x	x
OS Bypass	x	x	n/a	n/a	proprietary
Zero Copy	x	-	n/a	x	-
Async. I/O	x	x	x	n/a	x
Sync. Completion	x	-	-	n/a	-
Rich I/O Semantics	x	-	-	transaction	proprietary

Proposal: Unified I/O Stack

- Shortcut storage stack evolution: re-use NW stack
 - User mapped I/O queues/channels
 - Access abstraction: byte addressable access space
 - I/O request opcodes: Read/Write/Send/Recv/Atomics
- Higher level storage systems as first level citizens
 - File-systems, databases, object stores, ...
 - Translation of objects to I/O device address range
 - File, database column, key/value item, ...
 - Enables direct data transfer between device and object
- Unified
 - I/O semantics (work requests, event notification, polling, ...)
 - I/O memory management
 - Device management, device capability discovery, ...

Byte addressable Storage?

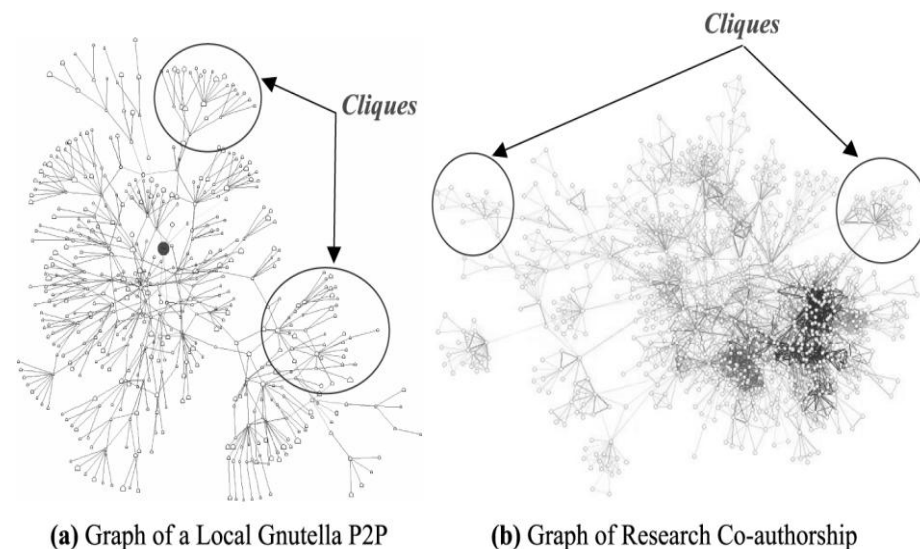
- Lowest level of access abstraction
 - System I/O view: [PA, len]: NVM access above FTL
 - Application view: [VA, len]: most concrete object representation
 - [VA, len] to be mapped to [key, offset, len] for access

- Advantages

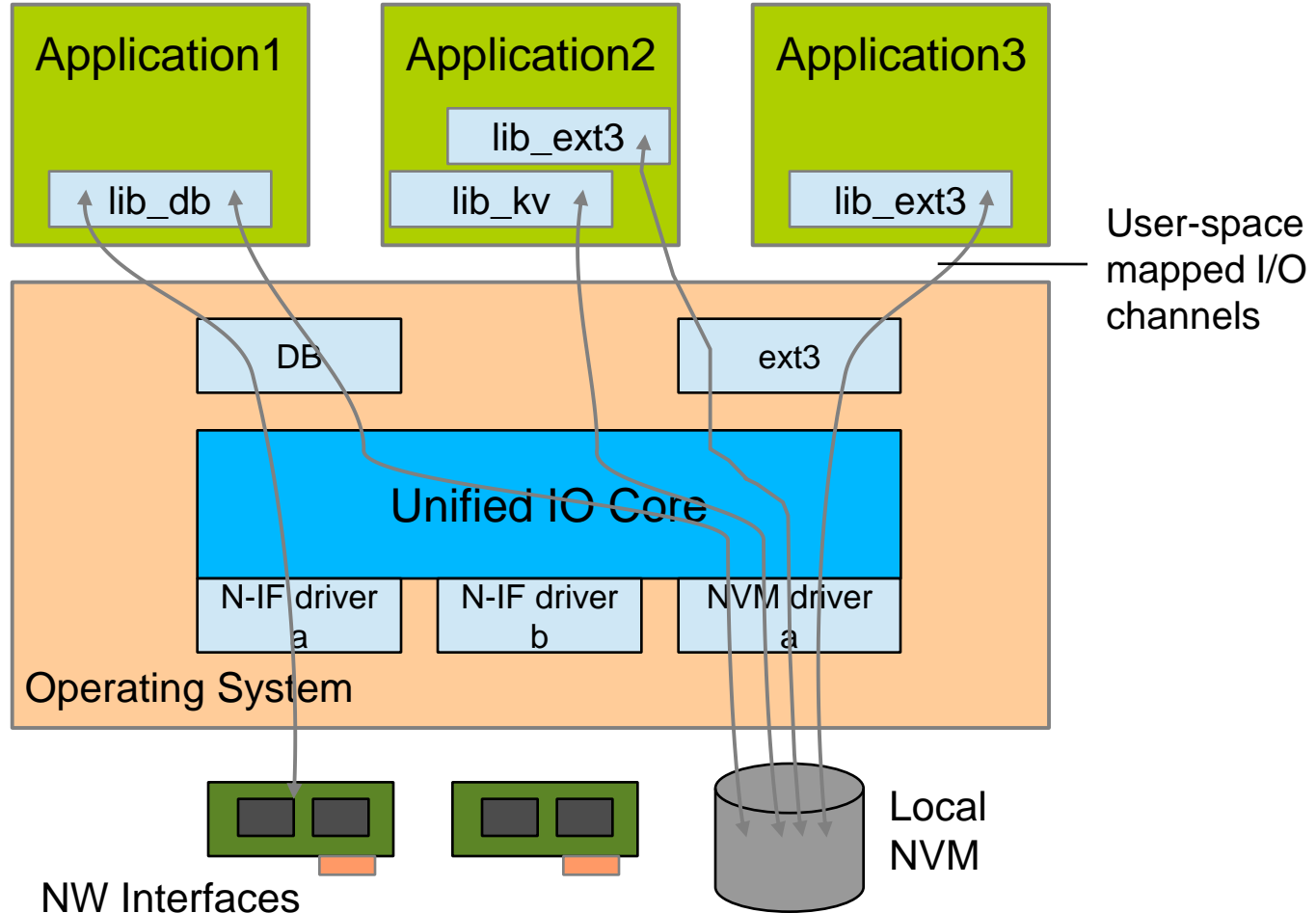
- Efficient data I/O
- Direct object reference
- Higher levels of abstraction if needed

- Examples:

- Byte addressable object store
- Traversing nodes of terabyte graph, random pointer chasing
- Terabyte Sorting



A Unified Stack



What about OFED?

You guessed it

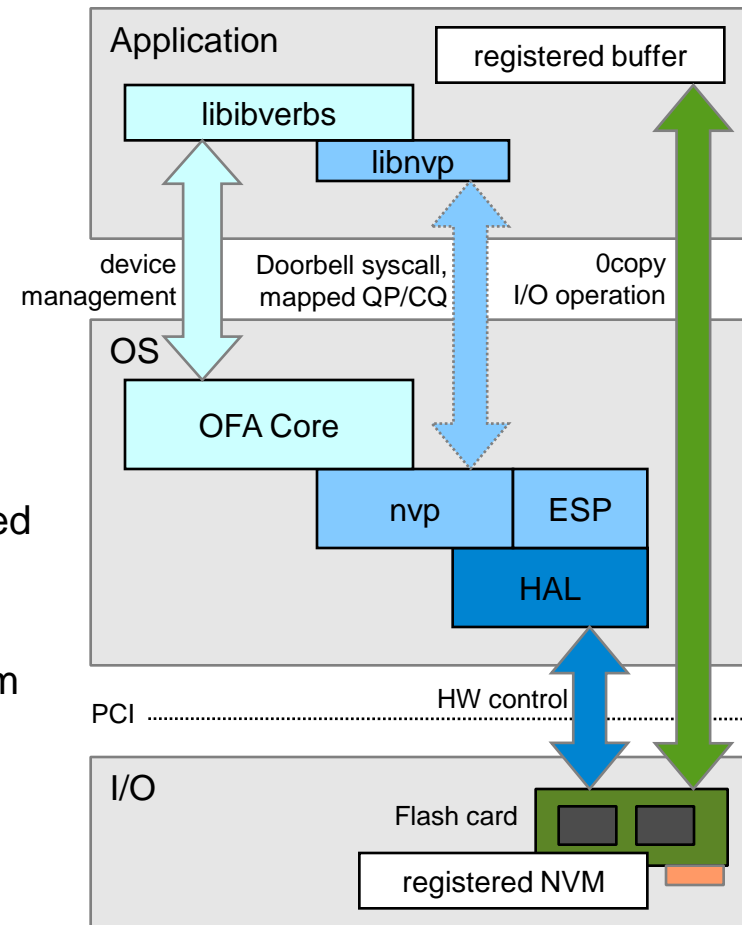
Unified IO \geq OpenFabrics

OFED: An (almost) perfect software environment to access high-performance storage

- Let the NVM device appear as an OFED RDMA verbs provider
- Communicate with a 'local storage peer'
- Plug and Play: RDMA Read/Write to 'local peer':
 - Zero copy NVM access
 - Byte addressable
 - Application private Queue Pairs, asynchronous operations, ...

NVP: A Prototype Implementation

- Prototype
 - PCI attached flash adapter
 - 'nvp' NVM OFED verbs provider with local peer
- NVP application operation:
 - Open nvp OFED device, create PD
 - Register local target buffers (`ibv_reg_mr()`)
 - Create QP and move it to RTS
 - Post Receive's + Send's executing RPC to 'embedded storage peer' (ESP) to learn partition parameters, register I/O memory and associated RTag's
 - Post READ/WRITE to read/write IO memory into/from local registered buffer
 - Mapped kernel QP/CQ, proprietary DB syscall

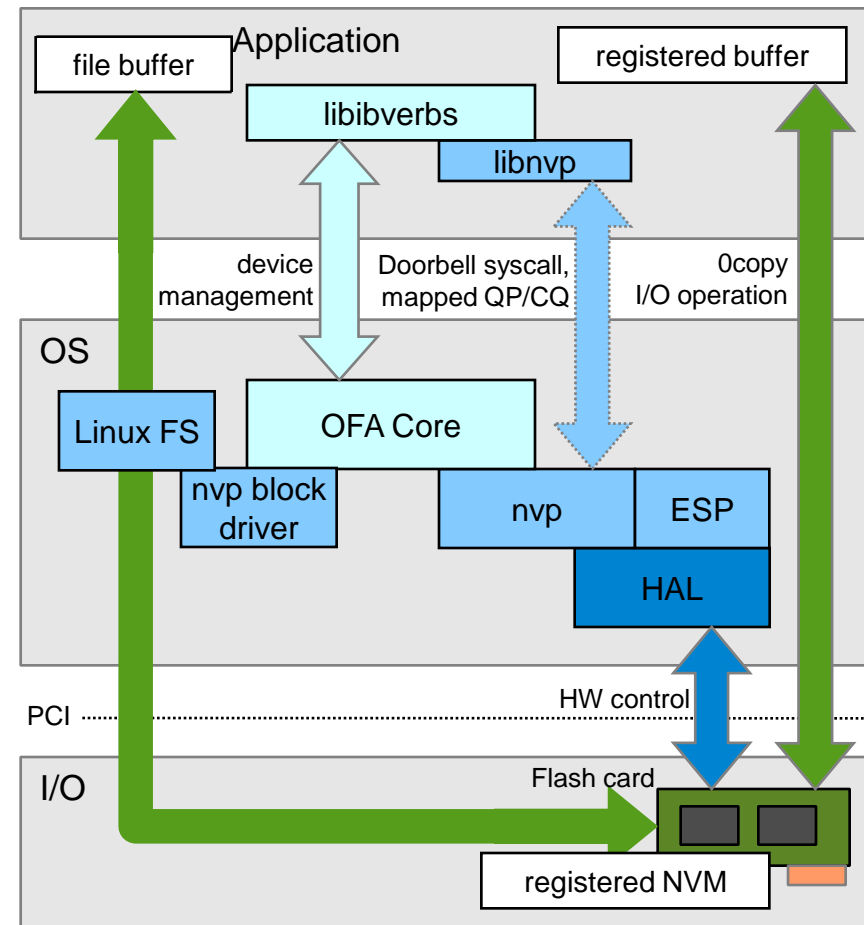


NVP: Very preliminary Results

First simple tests:

- μ -bench: low-level access inside driver, full page (8k) transfer, physical addressing (upper bound of what's possible from host side of PCIe)
- FIO: single process, standard linux raw block device, full page access
- NVP: single process, single QP, random read test, full page access (* write measured in different env. only)

	Read [MiB/s]	Write [MiB/s]	Read [kIOPS]
μ -bench	2450 (8k)	1820 (8k)	610
FIO	550 (4k) 1303 (8k)	181 (4k) 264 (8k)	140
NVP	2070 (4k) 2410 (8k)	n/a *1450 (8k)	500



Further Thoughts

Unified IO = OpenFabrics + **x**

x Memory registration model

x Currently based on RPC mechanism carried in Send/Receive work requests

x other ideas include:

- `mr = ibv_reg_mr(pd *, void *mem_id, length, IBV_ACCESS_XXX | IBV_ACCESS_IOMEM)`

x `mem_id` retrieved off-band from provider (via dedicated QP?)

x Overloads VA parameter, which would be NULL for IO memory

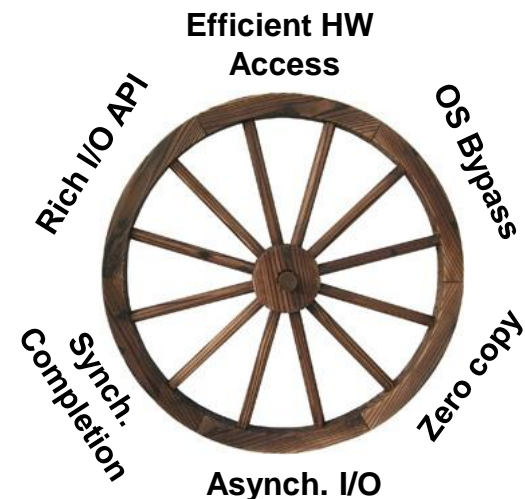
x Would also make send/receive RDMA model useable for data (is that needed?)

x What else?

x All perfect?

Conclusion

- Storage stacks are on a familiar crossroad
 - I/O device performance catches up with CPU speed
 - That's what happened to NW performance before
- Do not re-invent the performance wheel
- Reuse matured
 - Network Stack abstractions,
 - frameworks, and
 - APIs
- ✓ Integrate with OFED





Thank You



OPENFABRICS
ALLIANCE