NVMe Over CXL

How CXL Lets Us Do

Controller Memory Buffers the Right Way

Bill Gervasi, Principal Systems Architect

Wolley Inc.

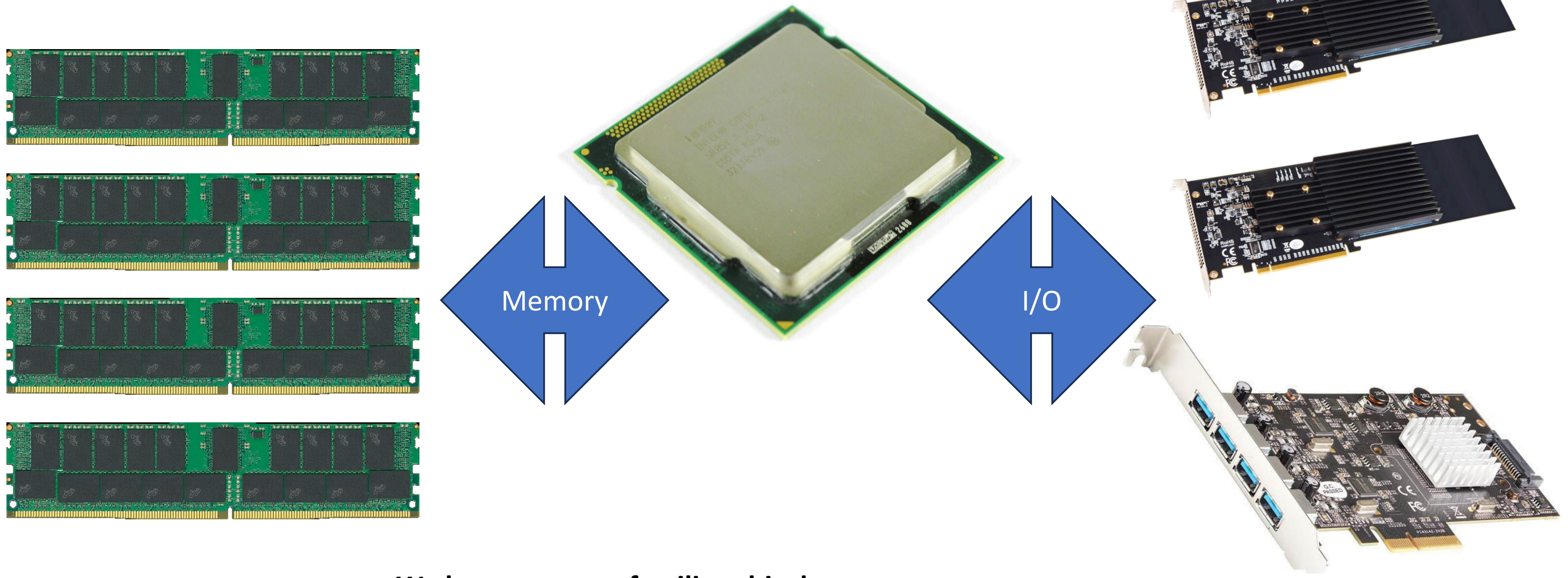
bilge@wolleytech.com

the **Future** *of* **Memory** *and* **Storage**

1. **NVMe Over CXL: How CXL Lets Us Do Controller Memory Buffers the Right Way**

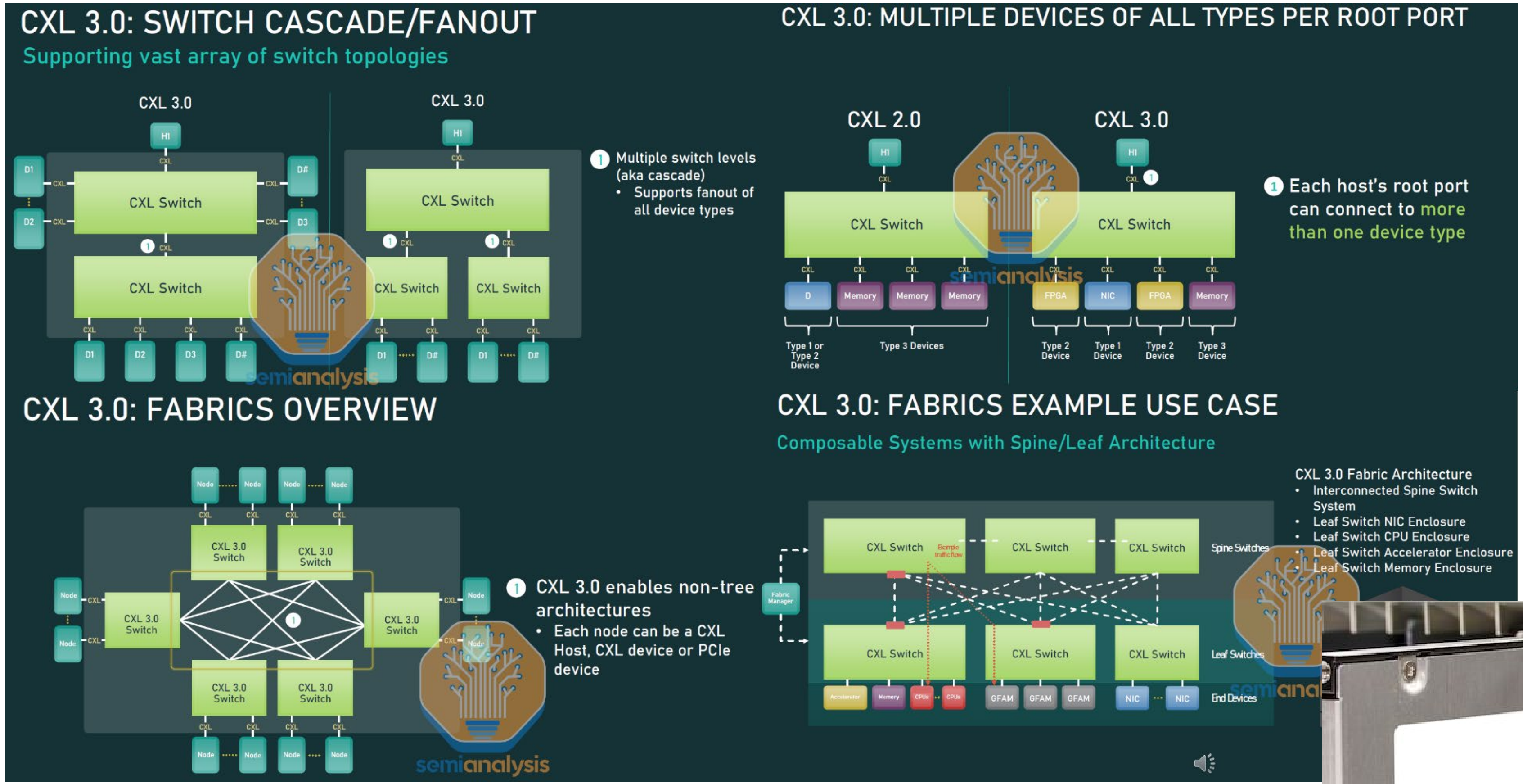
   NVMe has supported controller memory buffers since version 1.2 of the specification, however CMB performance advantages were limited by the PCIe bus itself which does not support a lightweight memory protocol. CXL fixes this fundamental limitation of CMBs by allowing efficient memory accesses with the CXL.mem protocol over that same PCIe physical interface while the CXL.io protocol supports all the legacy functionality of NVMe without requiring applications to be rewritten. Race conditions in resource allocation are resolved by having storage and memory on the same device. Advantages of SSDs using NVMe Over CXL are detailed and compared to memory semantic SSDs. The merging of storage and memory has another side benefit: DRAM persistence ala NVDIMM-N.

What millennium are we in?



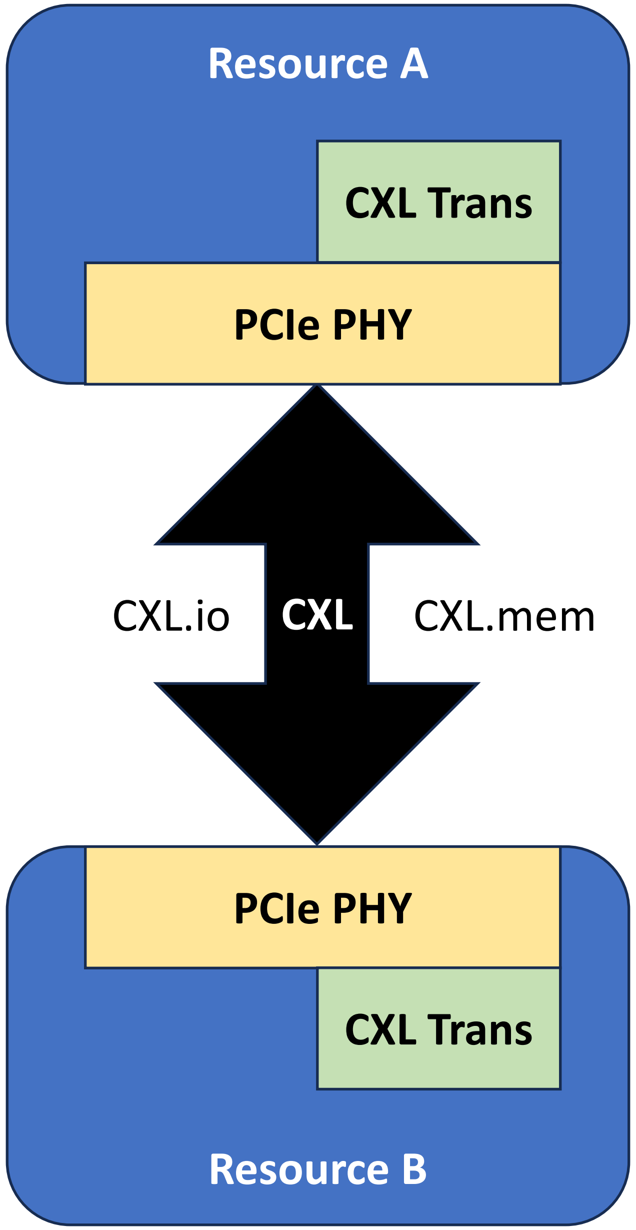
We have no way of nailing this down

This has been our basic system architecture for thousands of years


CXL 3.0: SWITCH CASCADE/FANOUT
Supporting vast array of switch topologies
CXL 3.0
CXL 3.0
CXL Switch
Multiple switch levels (aka cascade)
• Supports fanout of all device types
CXL 3.0: MULTIPLE DEVICES OF ALL TYPES PER ROOT PORT
CXL 2.0
CXL 3.0
Memory
FPGA
NIC
Type 1 or Type 2 Device
Type 3 Devices
Type 2 Device
Type 1 Device
Type 3 Device
Each host's root port can connect to more than one device type
CXL 3.0: FABRICS OVERVIEW
CXL 3.0 Switch
Node
CXL 3.0 enables non-tree architectures
• Each node can be a CXL Host, CXL device or PCIe device
CXL 3.0: FABRICS EXAMPLE USE CASE
Composable Systems with Spine/Leaf Architecture
CXL 3.0 Fabric Architecture
• Interconnected Spine Switch System
• Leaf Switch NIC Enclosure
• Leaf Switch CPU Enclosure
• Leaf Switch Accelerator Enclosure
• Leaf Switch Memory Enclosure
Fabric Manager
Spine Switches
Leaf Switches
End Devices
GFAM
semianalysis

One way to consider CXL expansion is the concept of racks with specific functions

Along comes CXL as a new fabric-based model for considering connecting processing, memory, storage, and communications



CXL is more than just another I/O bus

CXL allows the blending of processing, memory, storage, and I/O over a consistent protocol

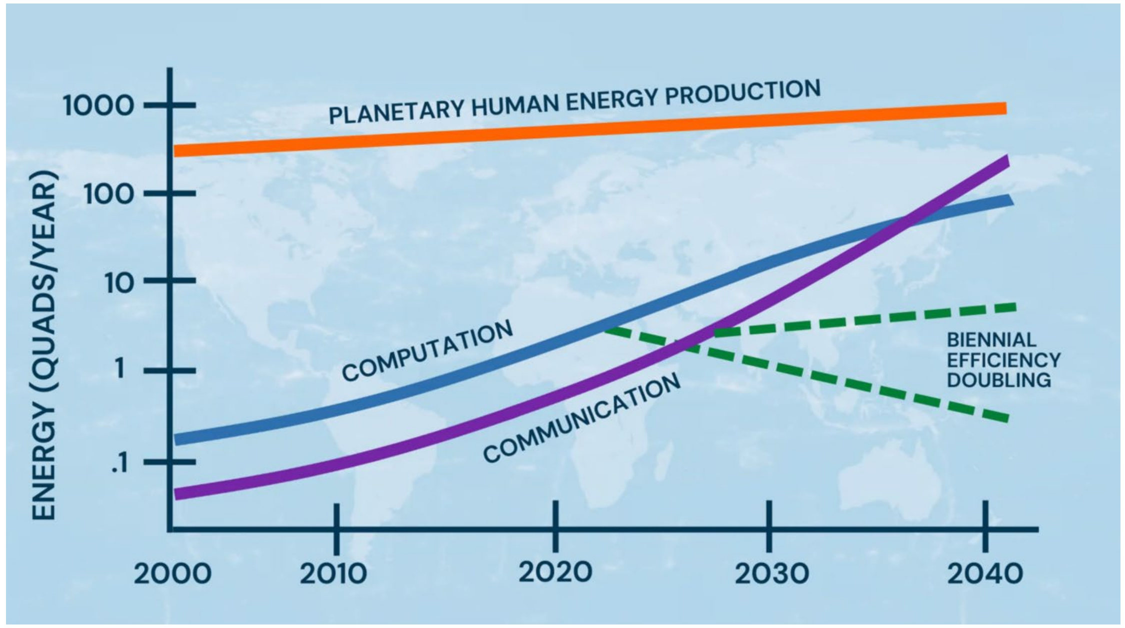
This enables virtualizing resources in interesting new ways

Yes, note that I drop CXL.cache from this diagram… CXL type 2 devices may disappear in an NVLink/UALink world…

This also implies that I believe CXL is complementary with xxLink

EES2 ENERGY EFFICIENCY SCALING

Data centers are facing a crisis in power consumption increases that threatens all electronics

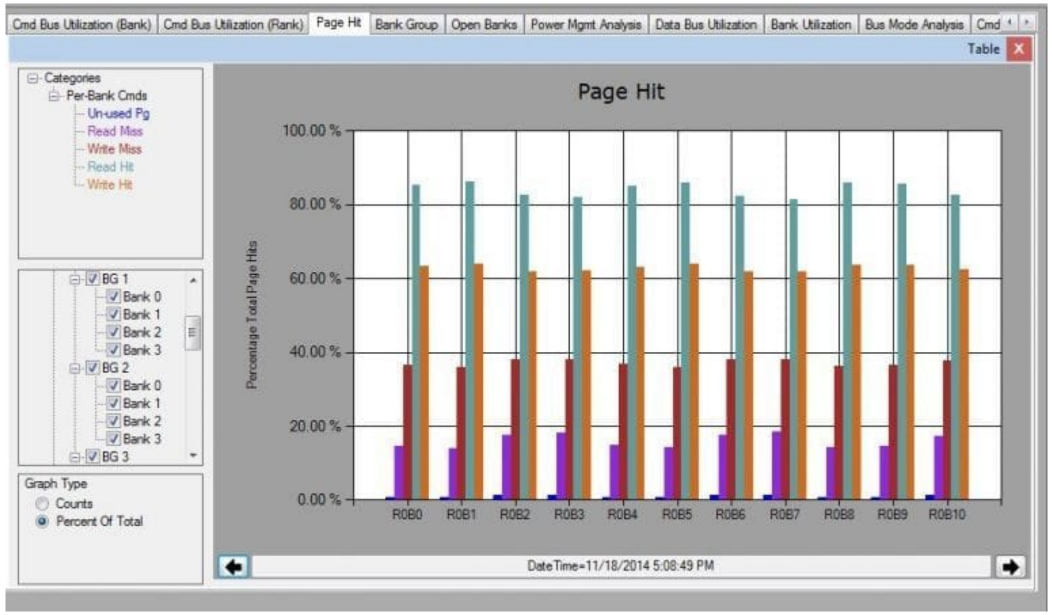


Operation	Energy per bit
Wireless data	10 – 30µJ
Internet: access	40 – 80nJ
Internet: routing	20nJ
Internet: optical WDM links	3nJ
Reading DRAM	5pJ
Communicating off chip	1 – 20 pJ
Data link multiplexing and timing circuits	~ 2 pJ
Communicating across chip	600 fJ
Floating point operation	100fJ
Energy in DRAM cell	10fJ
Switching CMOS gate	~50aJ – 3fJ
1 electron at 1V, or 1 photon @1eV	0.16aJ (160zJ)

While data centers are good at moving data around

They are terrible at actually performing work on that data

Efficiency < 0.00004%



NVMe Over CXL™ (NVMe-oC)

is a method for leaving data where it is and minimizing the traffic:

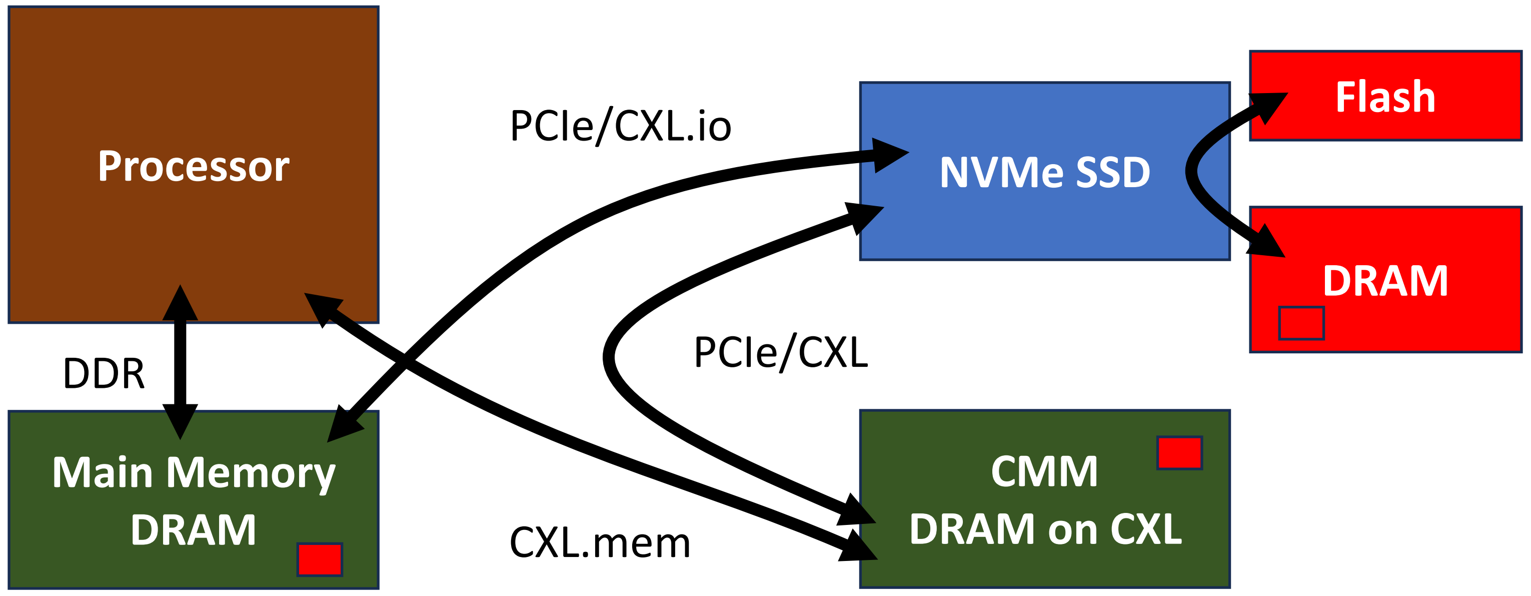
- Over the fabric
- In and out of multiple memories
- Letting the Host decide where dat should be

Leave the data where it is!

Don't chug-a-lug: Sip only what you need

Save the planet

NVMe SSD operation today uses legacy PCIe bus operations



Current systems separate storage and memory

Data is copied between storage and memory as needed in 4KB blocks

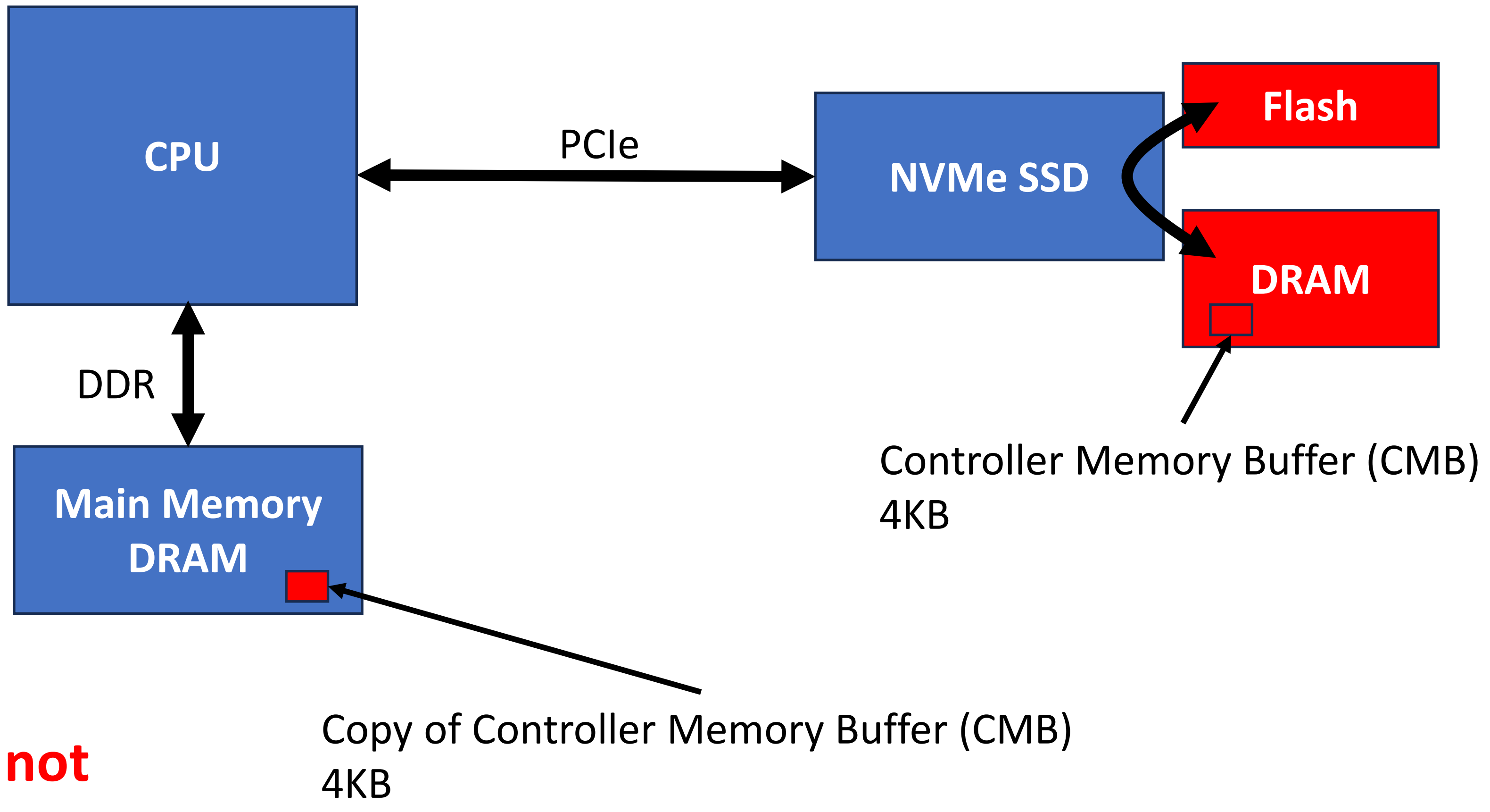
Each NAND read may result in many DRAM writes

- **SSD cache**
- **Remote memory**
- **Processor-local DRAM**

Why 4KB? Largely to compensate for PCIe overhead

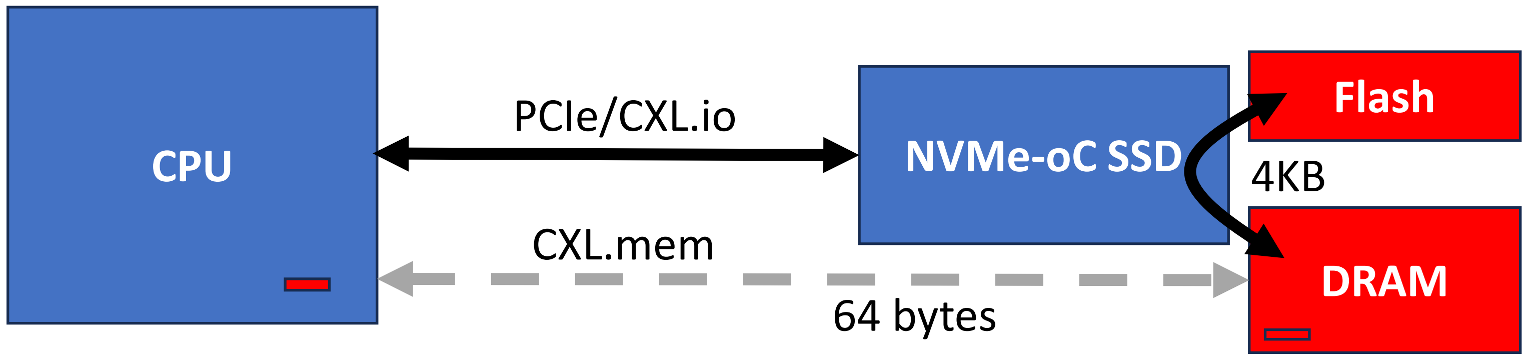
Wait, didn't NVMe V1.2 add the controller memory buffer (CMB) to solve this?

DMA between NAND and DRAM did not require the fabric


CPU
PCIe
NVMe SSD
Flash
DRAM
DDR
Main Memory DRAM
Controller Memory Buffer (CMB) 4KB
Copy of Controller Memory Buffer (CMB) 4KB

No, NVMe CMB does not solve the problem… the PCIe overhead still forces large data transfers

The NVMe Over CXL Solution: Only grab the FLITs you need



NVMe commands over PCIe (CXL.io) just as before to move 4KB between NAND and device local DRAM

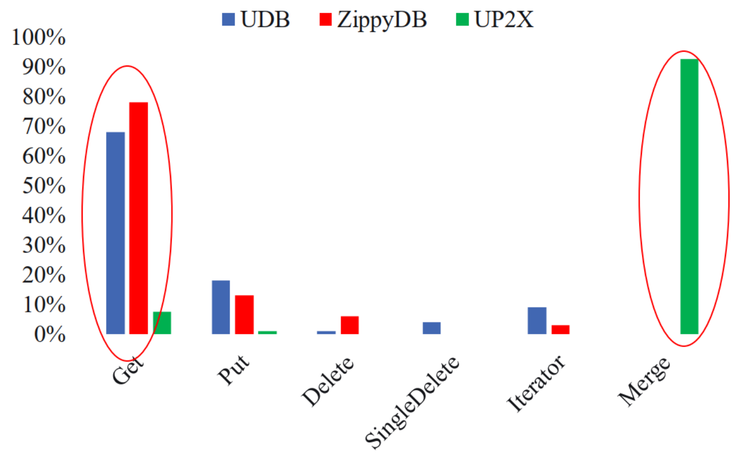
The CMB is allocated in CXL host-directed memory (HDM) address space

Processor grabs only the FLITs needed using CXL.mem

The rest of the BMC data remains where it is

Why HDM HMB in the DMZ, PDQ?*

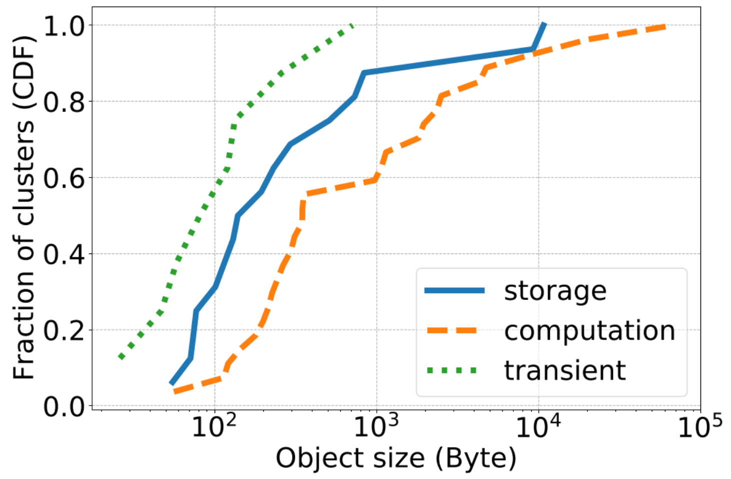
Facebook RocksDB



The average key size (AVG-K), the standard deviation of key size (SD-K), the average value size (AVG-V), and the standard deviation of value size (SD-V) of UDB, ZippyDB, and UP2X (in bytes)

	AVG-K	SD-K	AVG-V	SD-V
UDB	27.1	2.6	126.7	22.1
ZippyDB	47.9	3.7	42.9	26.1
UP2X	10.45	1.4	46.8	11.6

X (Twitter) Twemcache



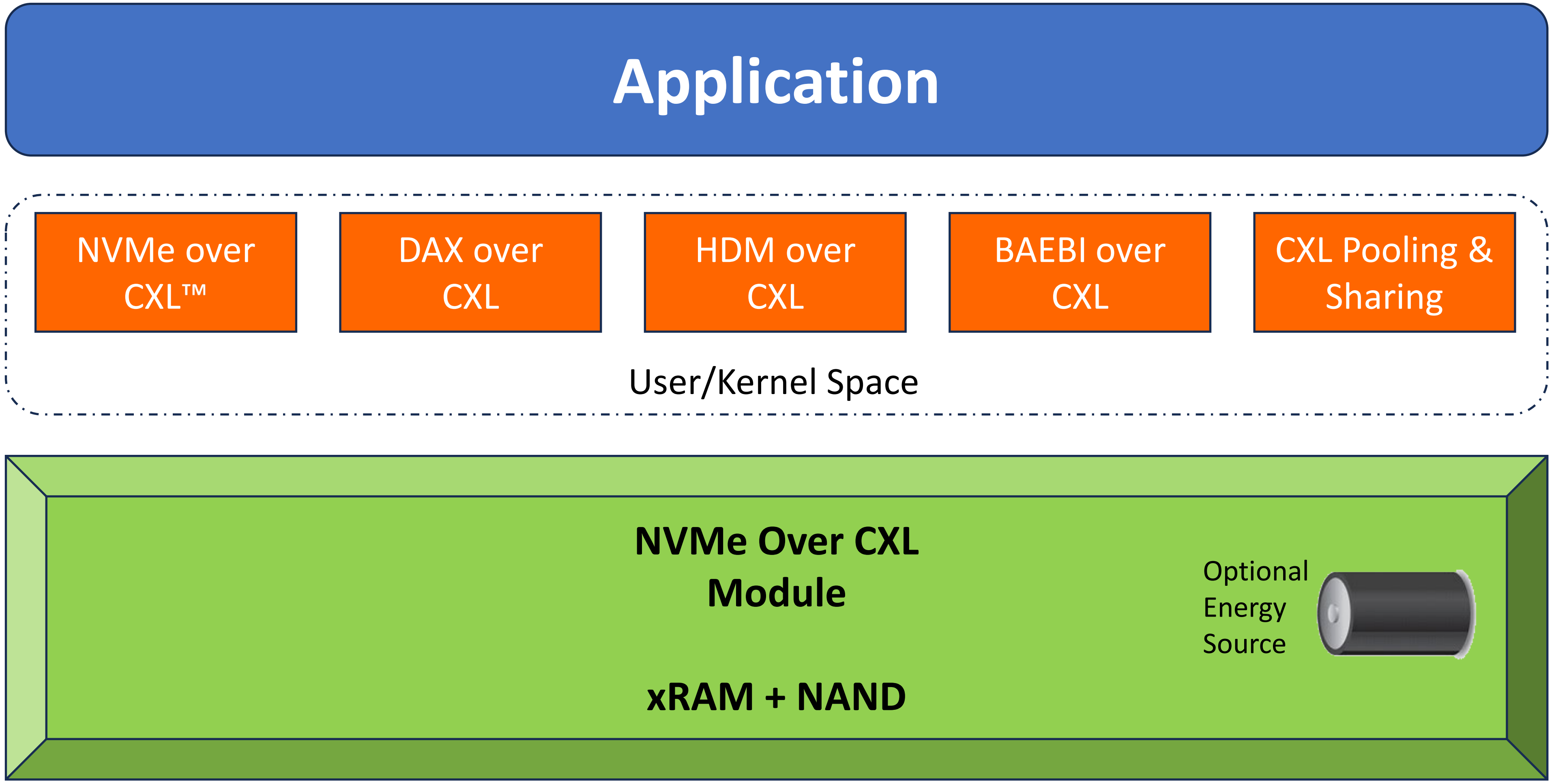
* Translation: why use the DRAM on the device as CXL memory?

Majority of data accesses are between 50 and 300 bytes with median ~100 bytes (key values, objects)

With NVMe-oC, PCIe traffic reduction > 97%

Cao, Zhichao, et al. "Characterizing, Modeling, and Benchmarking RocksDB Key-Value Workloads at Facebook" 18th USENIX Conference on File and Storage Technologies (FAST 20). 2020.

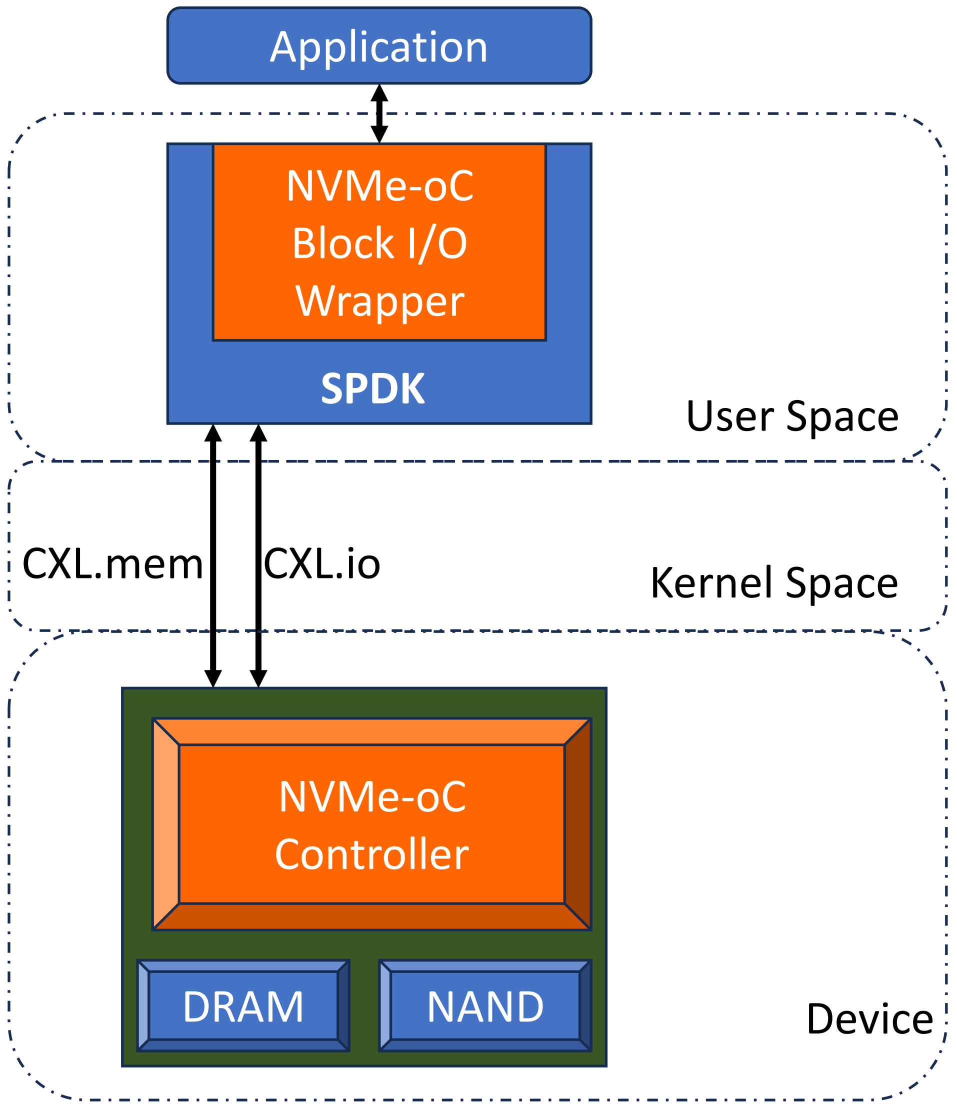
Yang, Juncheng, Yao Yue, and K. V. Rashmi. "A Large-scale Analysis of Hundreds of In-memory Key-value Cache Clusters at Twitter" ACM Transactions on Storage (TOS) 17.3 (2021): 1-35.



Use existing software APIs where possible

Release drivers open source on GitHub

NVMe Block Mode with Direct CMB



NVMe commands are issued like always

Controller memory buffer (CMB) and command/completion queue addresses directed to CXL device HDM

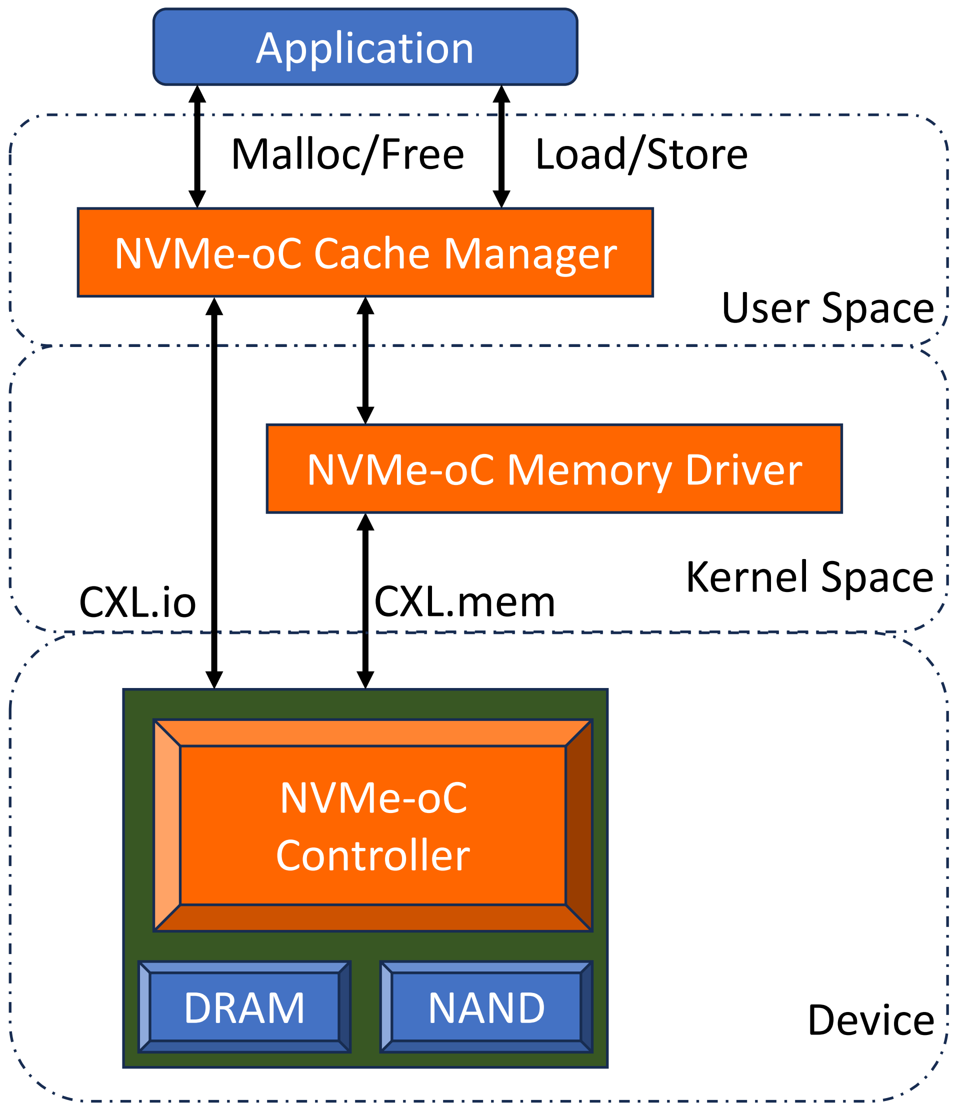
NVMe-oC controller

- Accepts NVMe commands over CXL.io
- Uses local HDM for operation queues
- Uses local HDM for block data transfers to and from local NAND
- Issues completion interrupts over CXL.io

Host CPU access HDM over CXL.mem

- CMB data buffers
- Command and completion queues

Host-Managed Memory Mode

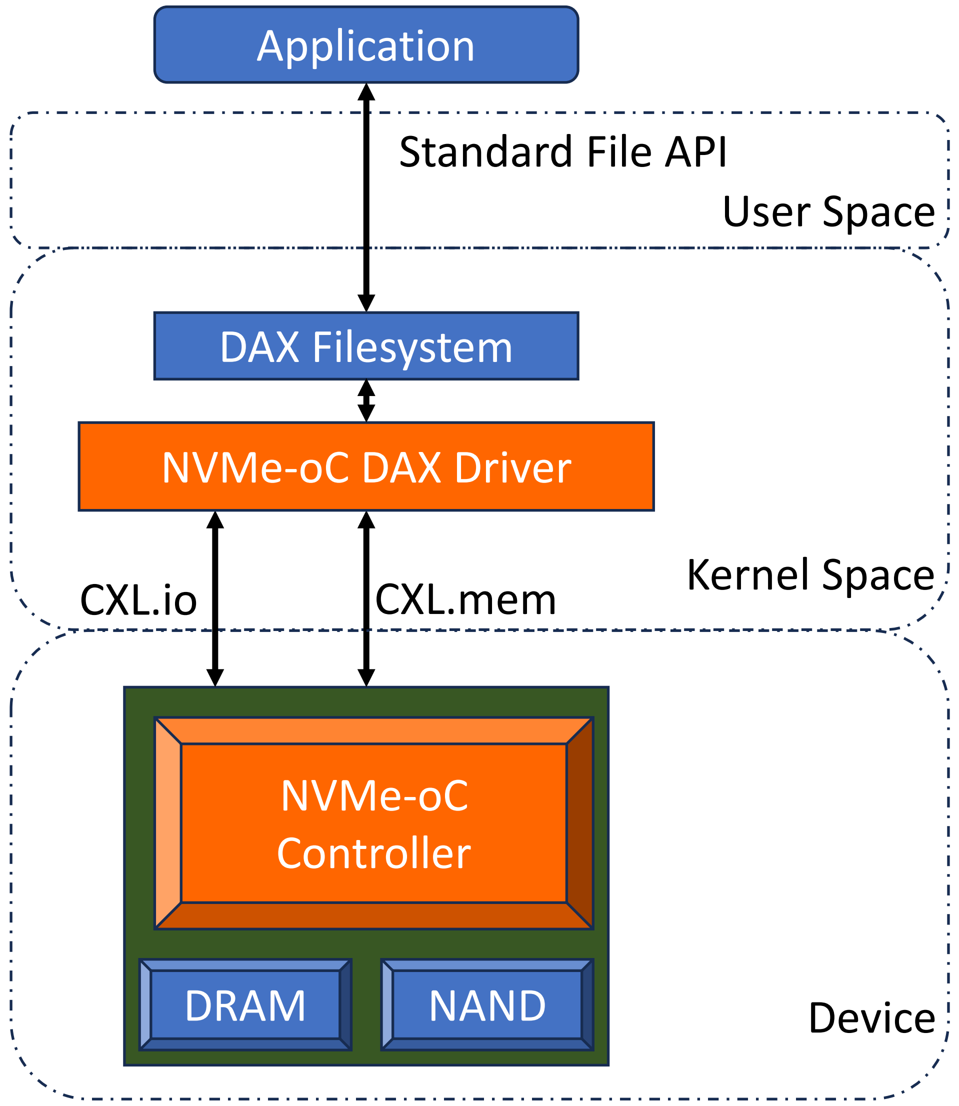


Memory allocation and freeing performed on Host, commands sent to NVMe-oC controller over CXL.io

Load/store operations from application access device HDM over CXL.mem

Traps allow swapping of blocks between device HDM and NAND

DAX File Mode

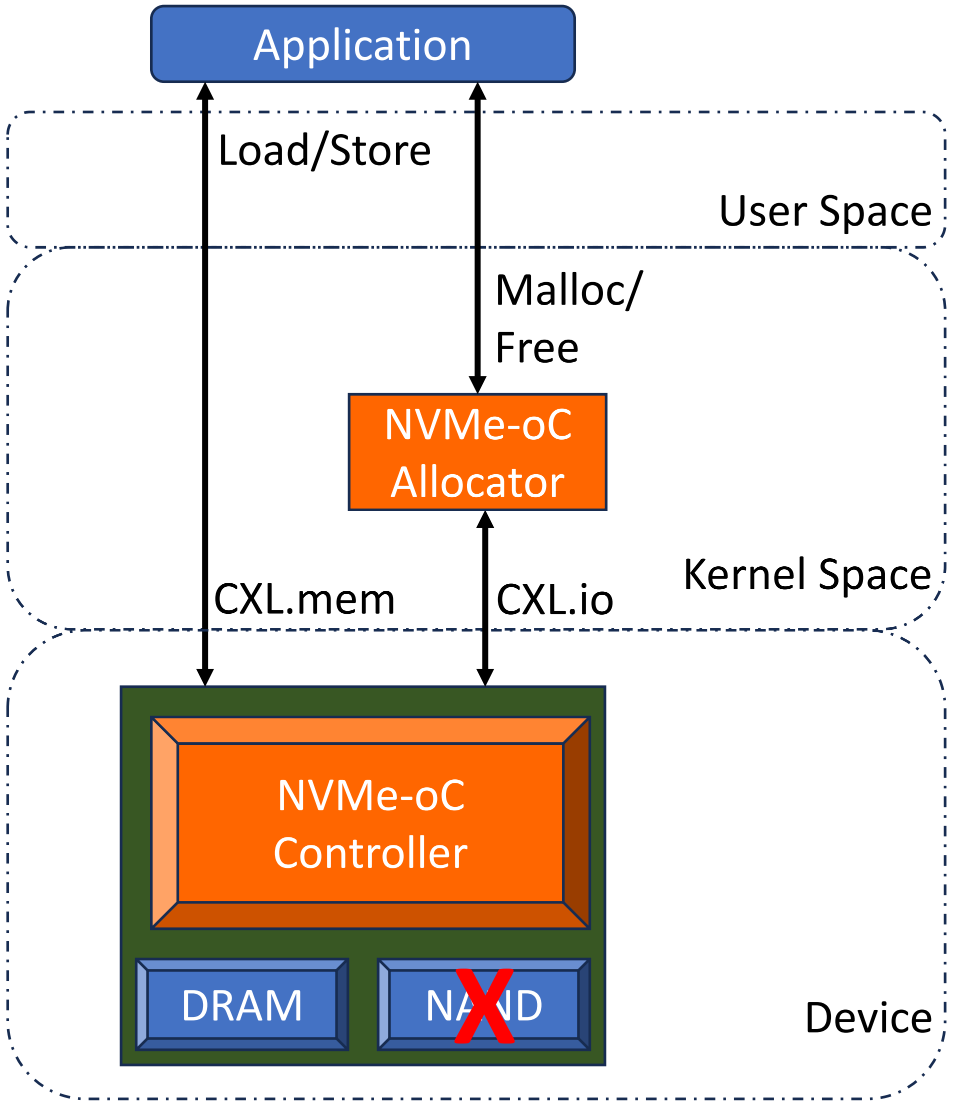


DAX filesystems provide programmer interface to HDM address space

NVMe-oC DAX driver manipulates hit and misses, directing transfers between device HDM and NAND over CXL.io

Host accesses device HDM using CXL.mem

Volatile Host Directed Memory Mode



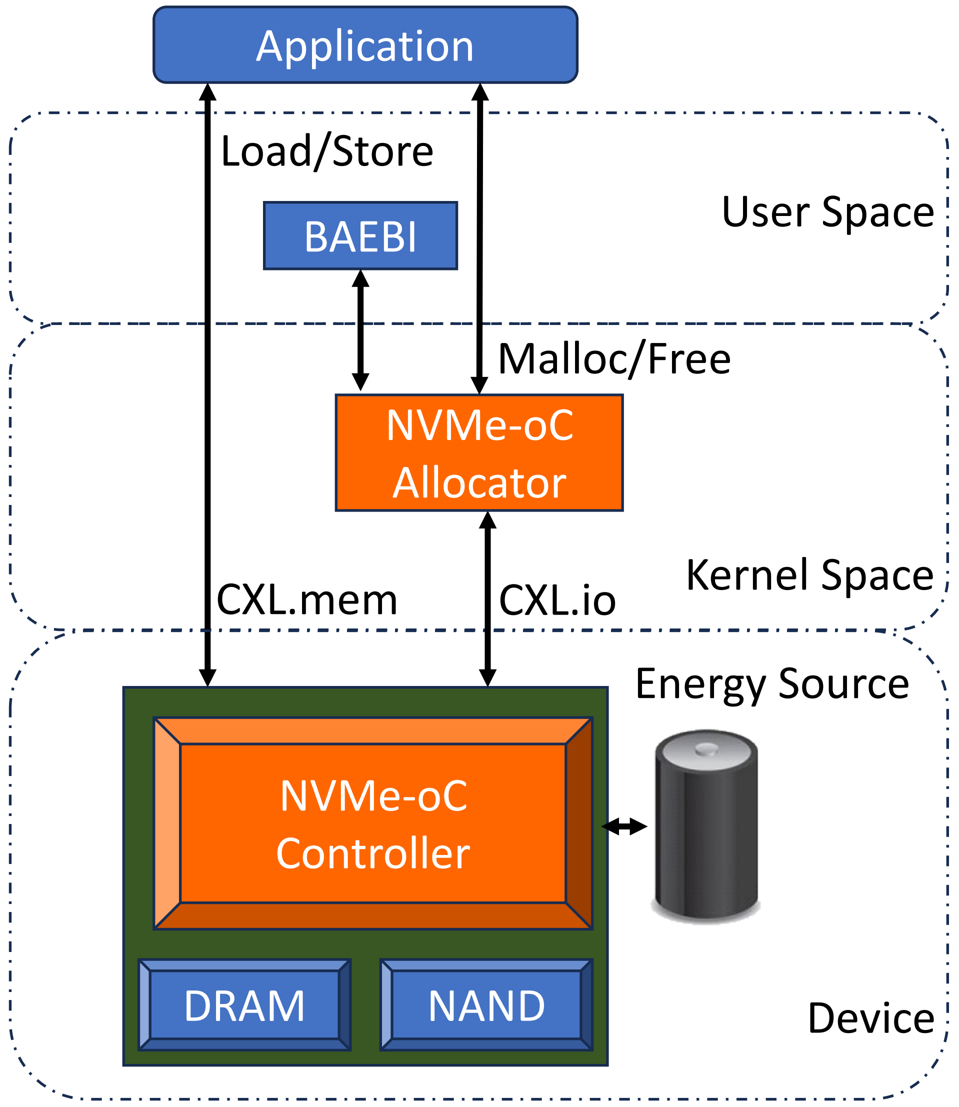
NAND not used in Volatile HDM mode

Device DRAM appears as standard HDM to the Host

NVMe-oC Allocator controls sharing and pooling over CXL.io to the NVMe-oC controller

Data lost on power failure

Non-Volatile Host Directed Memory Mode



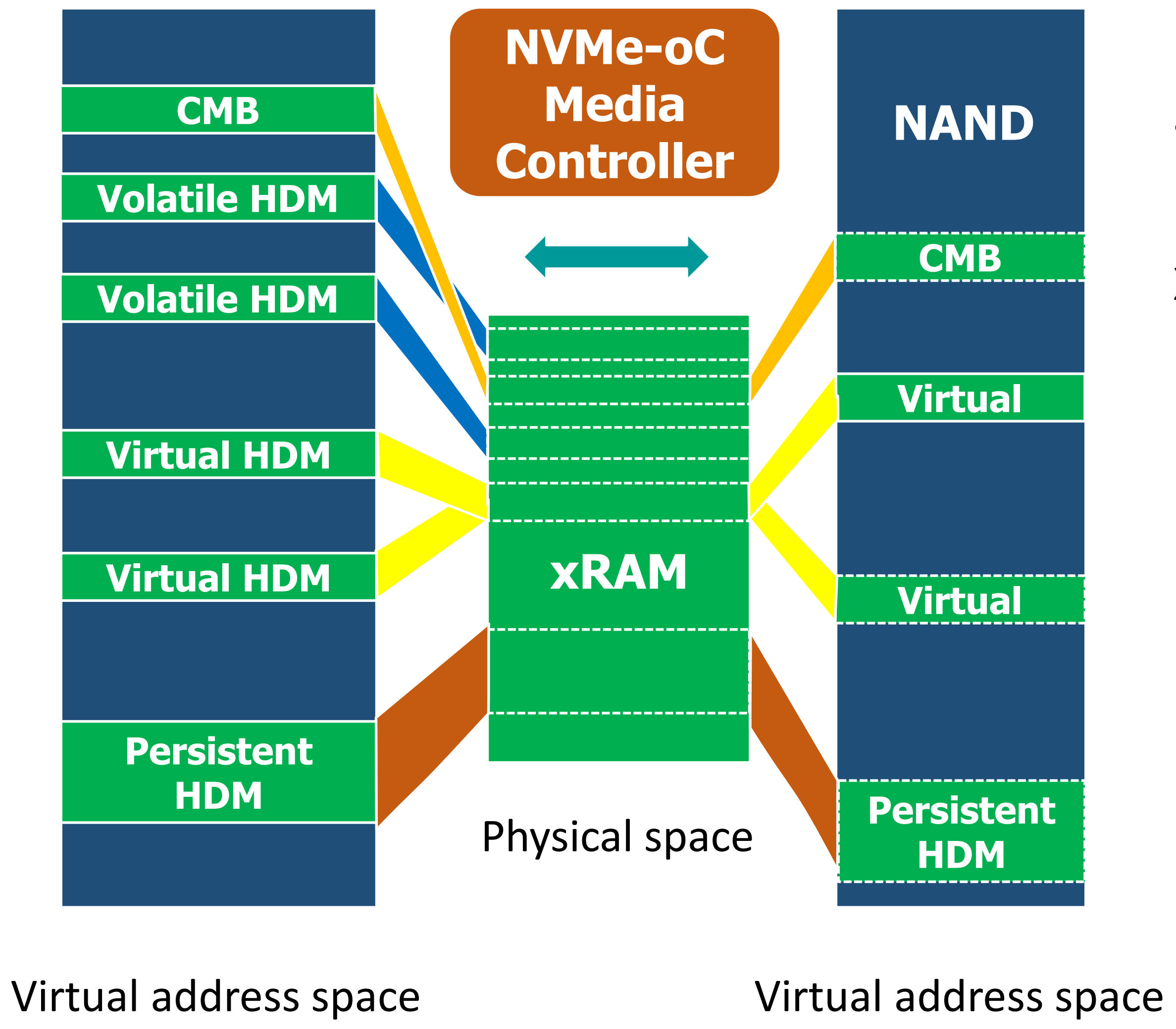
Operates as persistent memory in HDM or DAX modes

BAEBI driver controls

- Backup on power fail
- Restore when power back on
- Validation of memory image
- Security

Data not lost on power failure

Can also be implemented with non-volatile DRAM replacement without energy source



NVMe-oC operates in all access modes simultaneously

xRAM always accessed as HDM

CMB, DAX, HDM all allowed

NAND to xRAM transfer schemes driven by host using NVMe commands

Persistence regions can be partial

Summary

LEAVE DATA WHERE IT IS

NVMe is highly inefficient today

NVMe Over CXL is a virtualizing NAND+DRAM subsystem

CXL allows blending memory and storage

Uses multiple standard APIs

Open source drivers users can tweak

LEAVE DATA WHERE IT IS!!!

Non-volatile memory or energy for persistence

Multiple APIs, modes available simultaneously

Thank you for your time

Any more questions?

Bill Gervasi, Principal Systems Architect

Wolley Inc.

bilge@wolleytech.com