HotStorage'22

Hello Bytes, Bye Blocks: PCIe Storage Meets Compute Express Link for Memory Expansion

Myoungsoo Jung

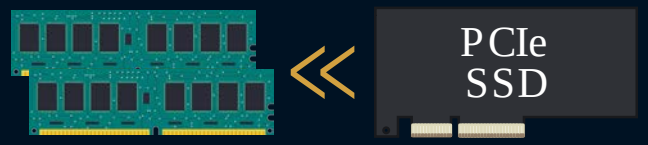
Computer Architecture and MEmory systems Laboratory

KAIST CAMEL

High-Level Summary

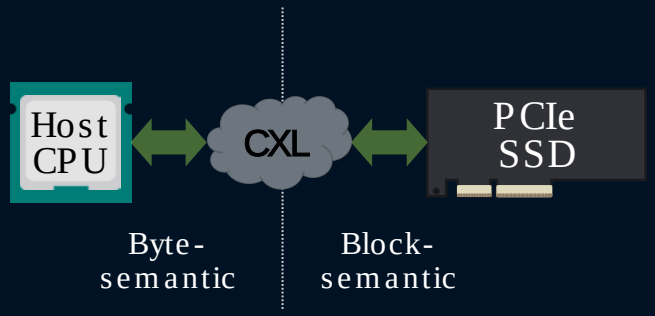
There is a need for storage devices as working memory

Can achieve ~50x larger memory

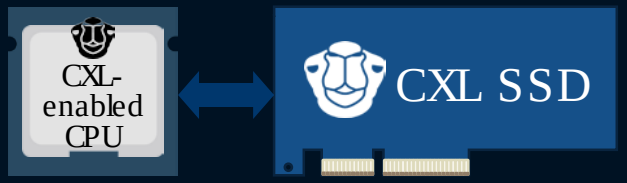


※ capacity comparison: when DRAM chip and NAND flash chip have the same price (excluding controller price)

Emerging CXL can make it possible (+ cache-coherent)



We propose a "storage-integrated memory expander"



- Hardware prototyping
- User guides for better use

1. Long-Standing Dream

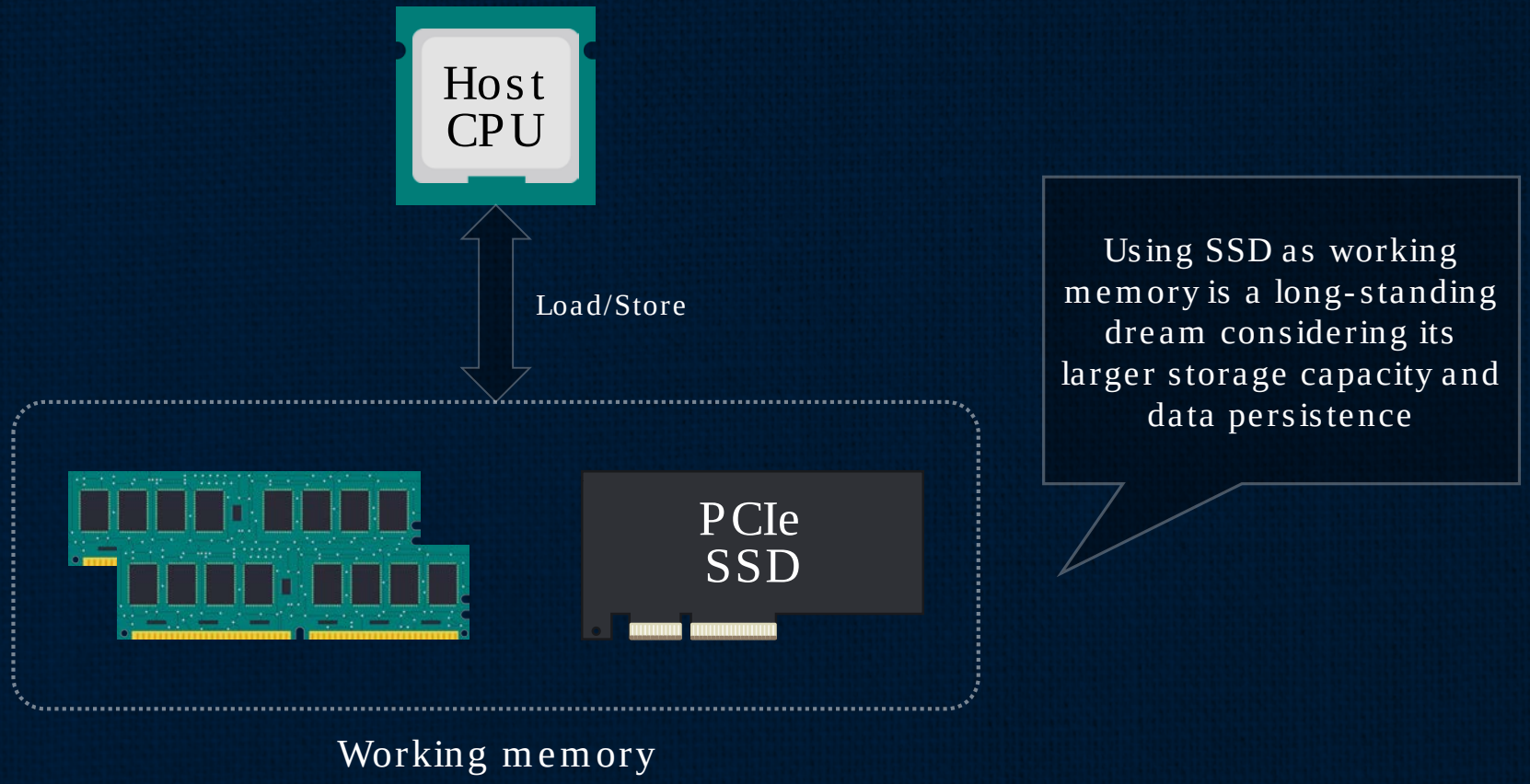
2. Why CXL?

3. Storage-Integrated Memory Expander

4. User Guide #1 - Pooling

5. User Guide #2 -Storage-Aware Annotation

PCIe Storage as Working Memory



Benefits: Larger Memory

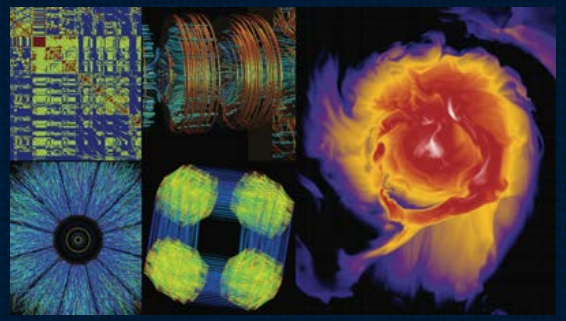
Chip-to-Chip comparison

The capacity of the NAND flash chip is 16x larger than the DRAM chip. In parallel, flash offers excellent persistence capability

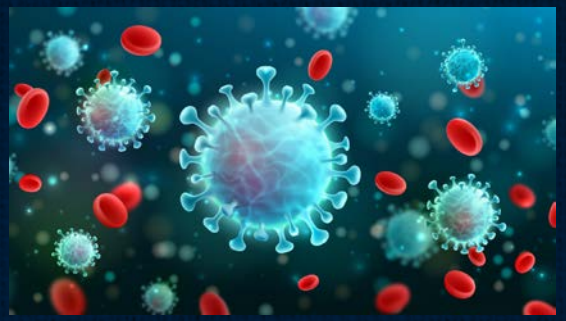
Capacity comparison at the same price

Furthermore, we can buy several NAND flash chips with a single DRAM chip price.

Benefits: Larger Memory



Scientific analysis



Health care

Recommendation system

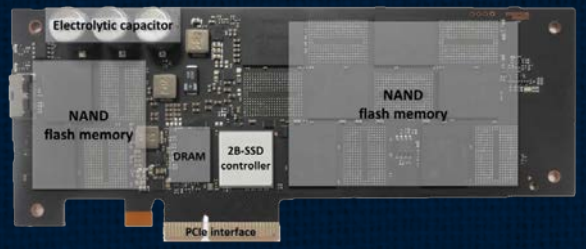
ML-based automotive system

Achieving larger memory (with data persistence) can **open a new door** for big data applications (more exploration and more analysis).

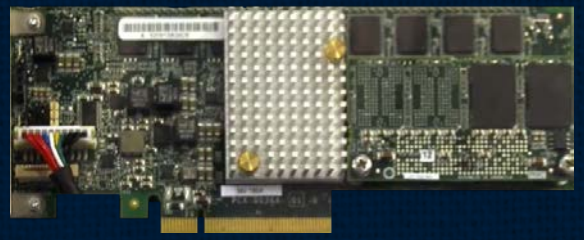
Conventional Attempts

Industry prototype

SAMSUNG : 2BSSD



Microsemi : NVRAM cards



There were several attempts to use SSD as working memory by supporting byte-addressability.

NVMe standard

nvm EXPRESS : PMR

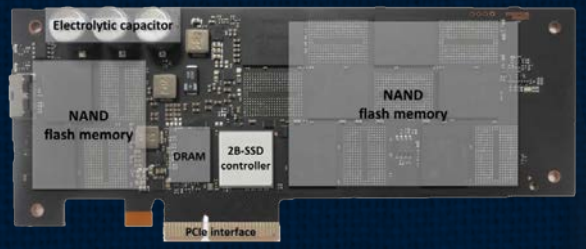
8.14 Persistent Memory Region

The Persistent Memory Region (PMR) is an optional region of general purpose PCI Express read/write persistent memory that may be used for a variety of purposes. The controller indicates support for the PMR by setting CAP.PMRS (refer to section 3.1.3.1) to '1' and indicates whether the controller supports command data and metadata transfers to or from the PMR by setting support flags in the PMRCAP property. When command data and metadata transfers to or from PMR are supported, all data and metadata associated with a particular command shall be either entirely located in the Persistent Memory Region or outside the Persistent Memory Region.

The PMR's PCI Express address range is used for external memory read and write requests to the PMR. The PCI Express address range and size of the PMR is defined by the PCI Base Address Register (BAR) indicated by PMRCAP.BIR. The PMR consumes the entire address region exposed by the BAR and supports all the required features of the PCI Express programming model (i.e., it in no way restricts what is otherwise permitted by PCI Express).

Commonality of Conventional Attempts

SAMSUNG : 2BSSD



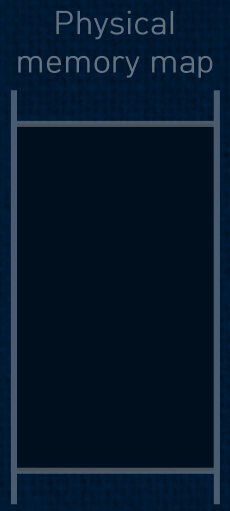
Microsemi : NVRAM cards

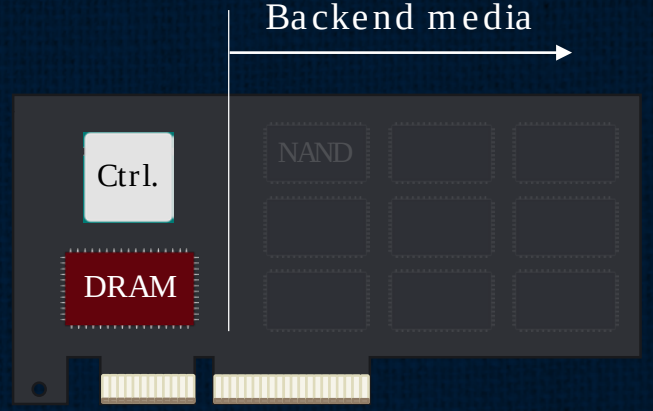
nvm EXPRESS : PMR

8.14 Persistent Memory Region

The Persistent Memory Region (PMR) is an optional region of general purpose PCI Express read/write persistent memory that may be used for a variety of purposes. The controller indicates support for the PMR by setting CAP.PMRS (refer to section 3.1.3.1) to '1' and indicates whether the controller supports command data and metadata transfers to or from the PMR by setting support flags in the PMRCAP property. When command data and metadata transfers to or from PMR are supported, all data and metadata associated with a particular command shall be either entirely located in the Persistent Memory Region or outside the Persistent Memory Region.

The PMR's PCI Express address range is used for external memory read and write requests to the PMR. The PCI Express address range and size of the PMR is defined by the PCI Base Address Register (BAR) indicated by PMRCAP.BIR. The PMR consumes the entire address region exposed by the BAR and supports all the required features of the PCI Express programming model (i.e., it in no way restricts what is otherwise permitted by PCI Express).

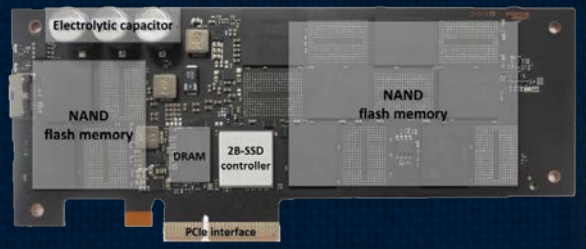




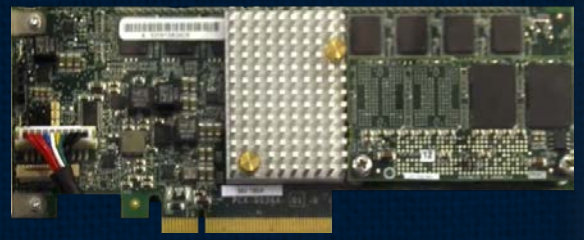
The conventional attempts expose SSD's internal DRAM to gap the byte and block I/O granularity and hide the long latency of SSD backend block media as much as possible.

Commonality of Conventional Attempts

SAMSUNG : 2BSSD



Microsemi : NVRAM cards

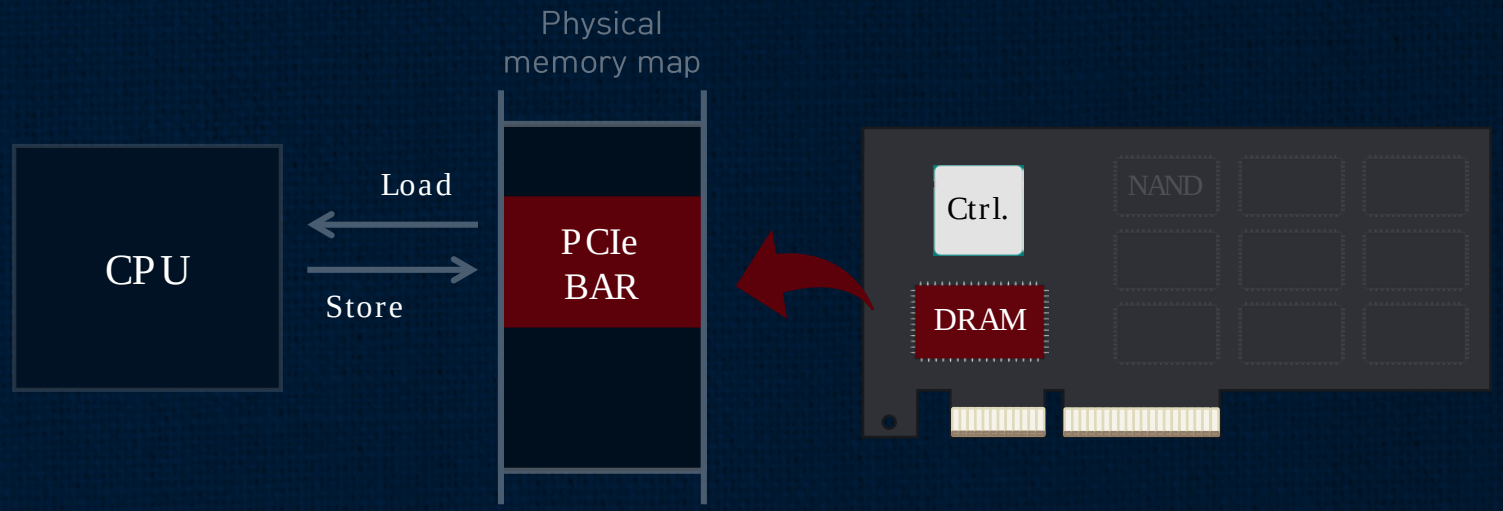


nvm EXPRESS : PMR

8.14 Persistent Memory Region

The Persistent Memory Region (PMR) is an optional region of general purpose PCI Express read/write persistent memory that may be used for a variety of purposes. The controller indicates support for the PMR by setting CAP.PMRS (refer to section 3.1.3.1) to '1' and indicates whether the controller supports command data and metadata transfers to or from the PMR by setting support flags in the PMRCAP property. When command data and metadata transfers to or from PMR are supported, all data and metadata associated with a particular command shall be either entirely located in the Persistent Memory Region or outside the Persistent Memory Region.

The PMR's PCI Express address range is used for external memory read and write requests to the PMR. The PCI Express address range and size of the PMR is defined by the PCI Base Address Register (BAR) indicated by PMRCAP.BIR. The PMR consumes the entire address region exposed by the BAR and supports all the required features of the PCI Express programming model (i.e., it in no way restricts what is otherwise permitted by PCI Express).



Thus, CPU can access SSD with load/store instructions (e.g., byte-addressability via MMIO).

SSD's internal DRAM space is mapped to PCIe configuration space (e.g., BARs).

1. Long-Standing Dream

2. Why CXL? (Not Conventional Attempts, PCIe BAR?)

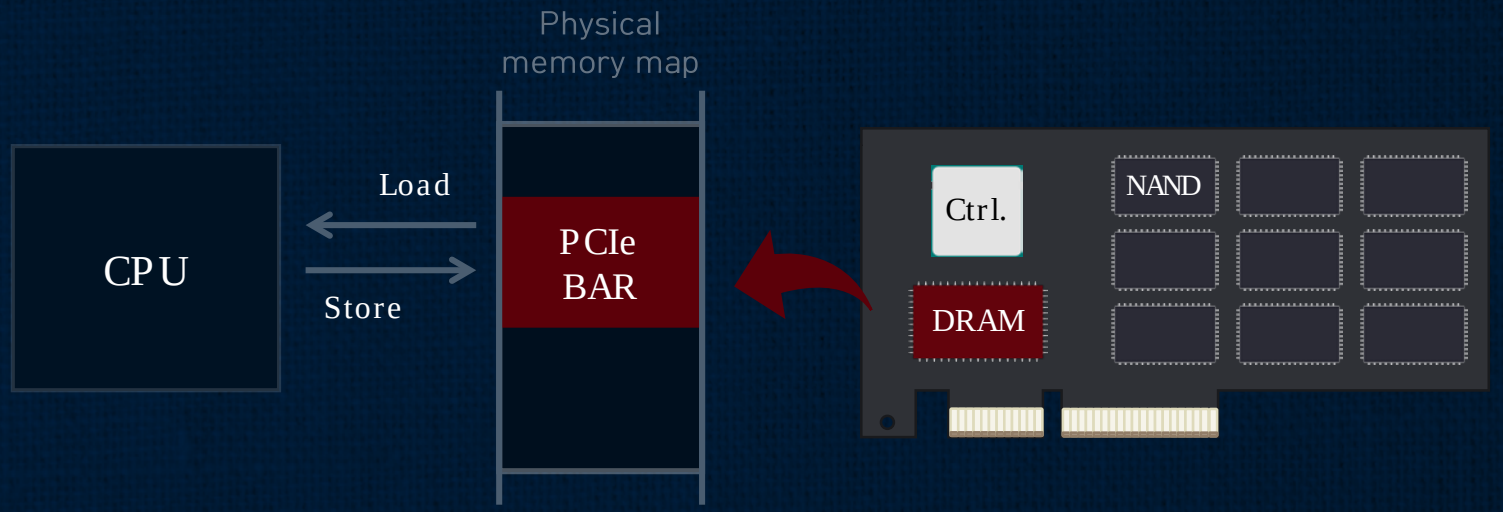
3. Storage-Integrated Memory Expander

4. User Guide #1 - Pooling

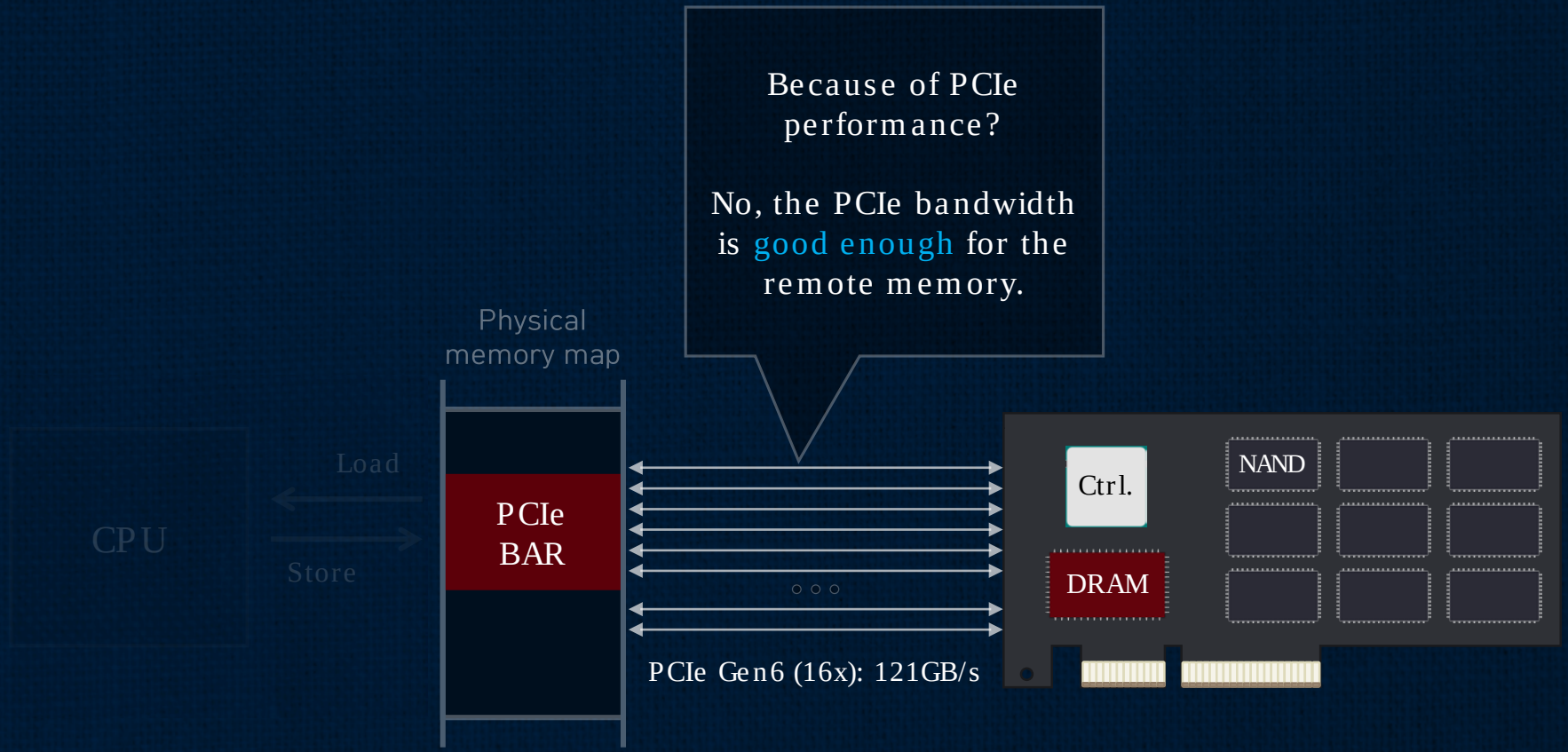
5. User Guide #2 -Storage-Aware Annotation

Limitations of Conventional Attempts

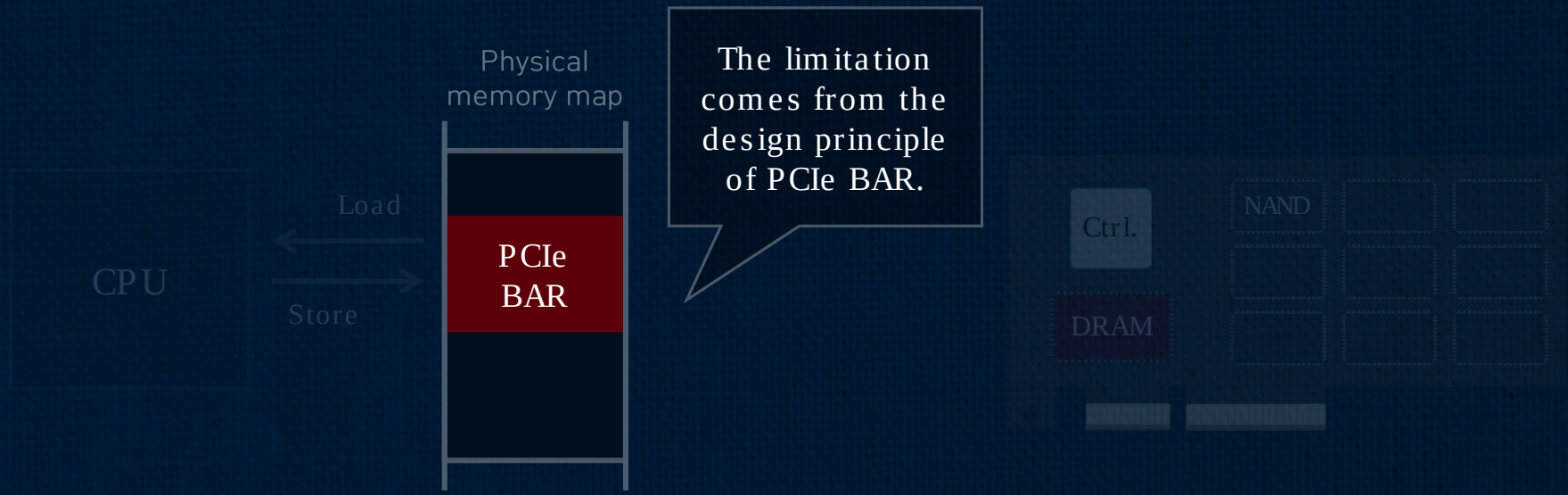
Memory expansion using PCIe BAR is impractical due to its limitation.



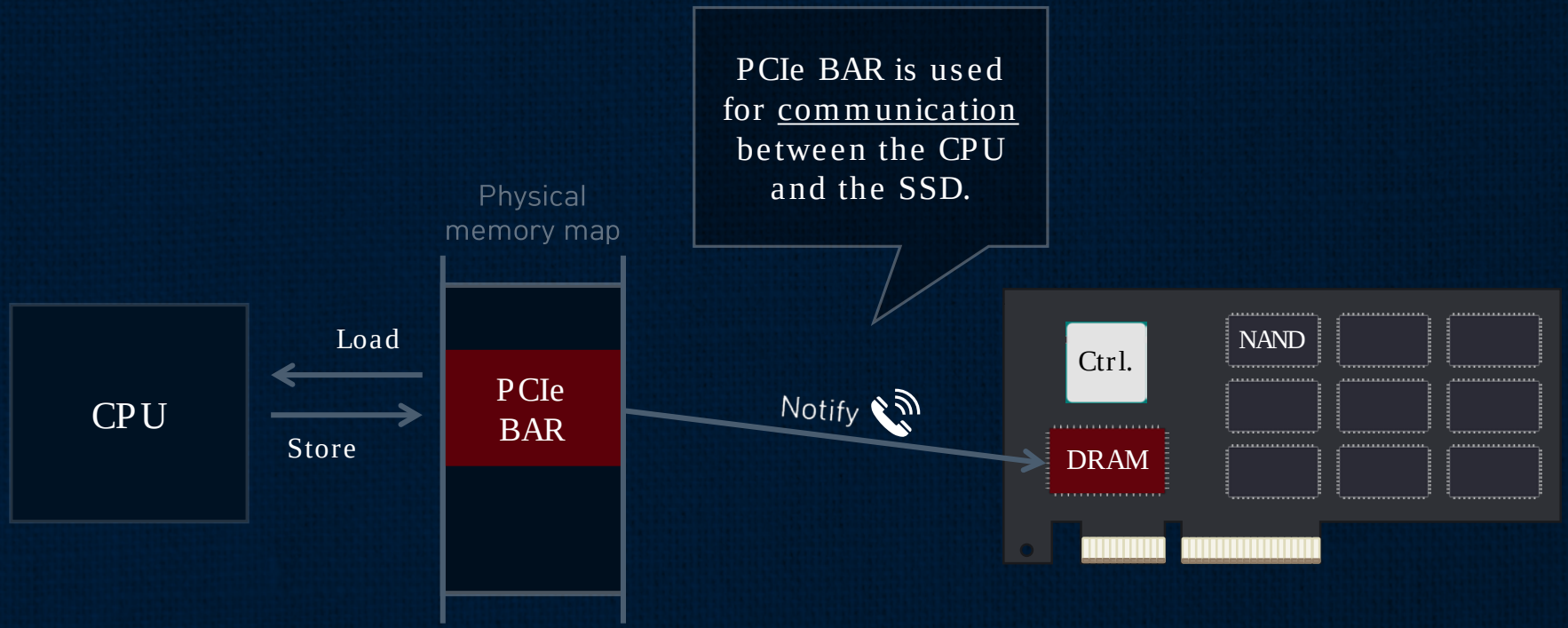
Good Enough PCIe Bandwidth



But, Non-Cacheable Access



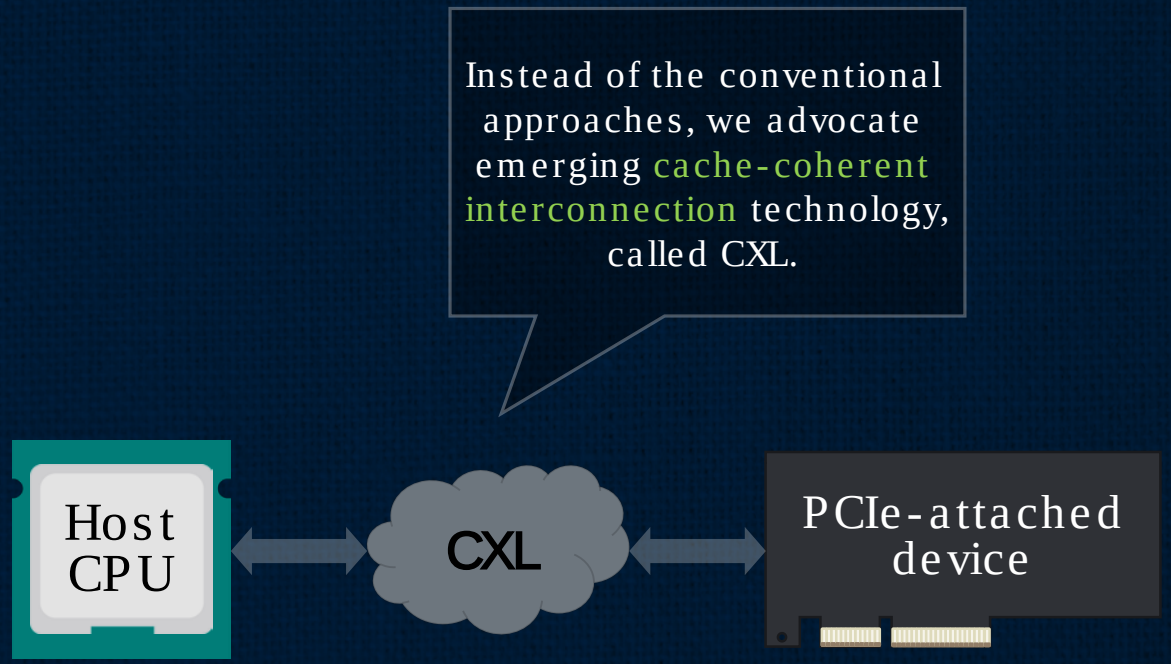
But, Non-Cacheable Access



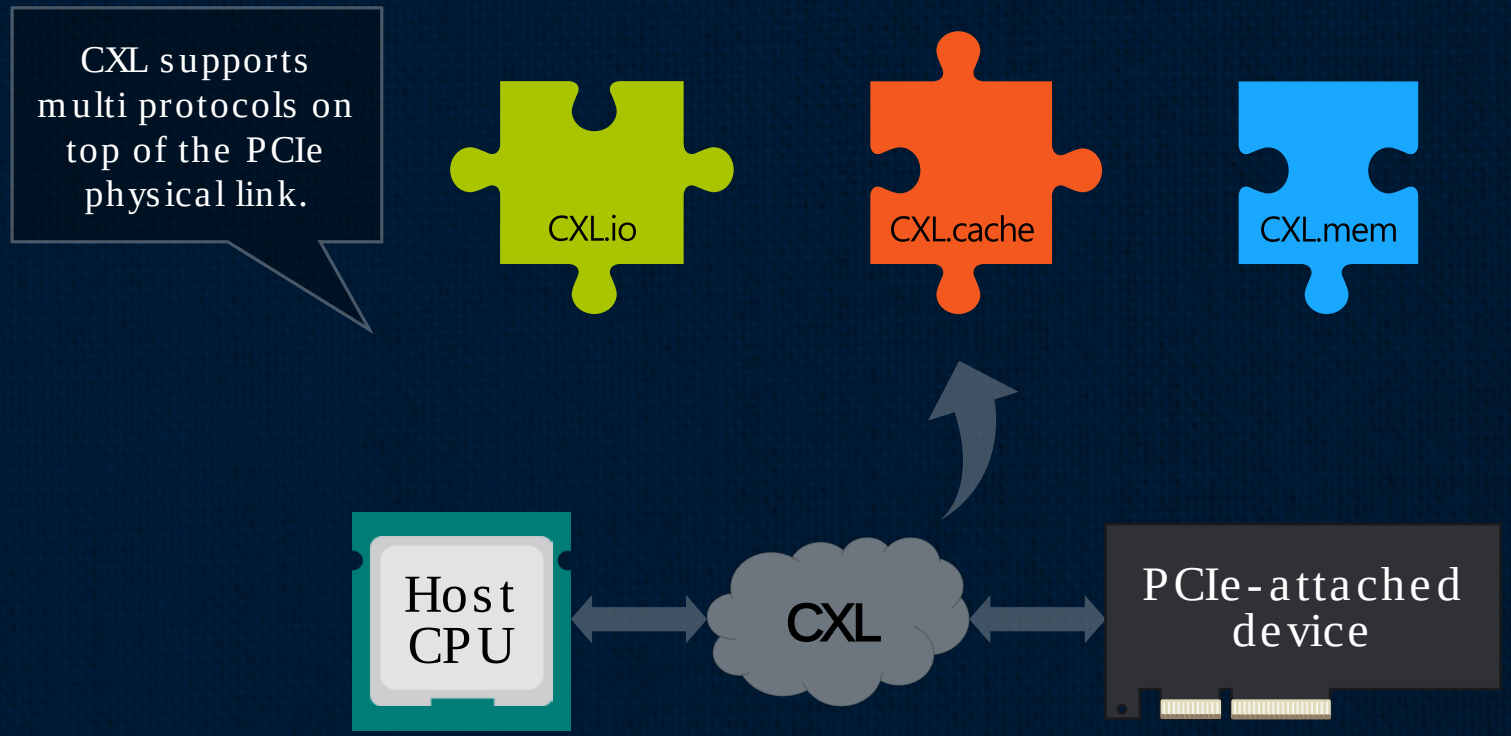
But, Non-Cacheable Access

Thus, Intel and AMD do not allow CPU-side caching for PCIe-related memory requests to prevent the PCIe devices' malfunction.

Advocation: CXL for Memory Expansion



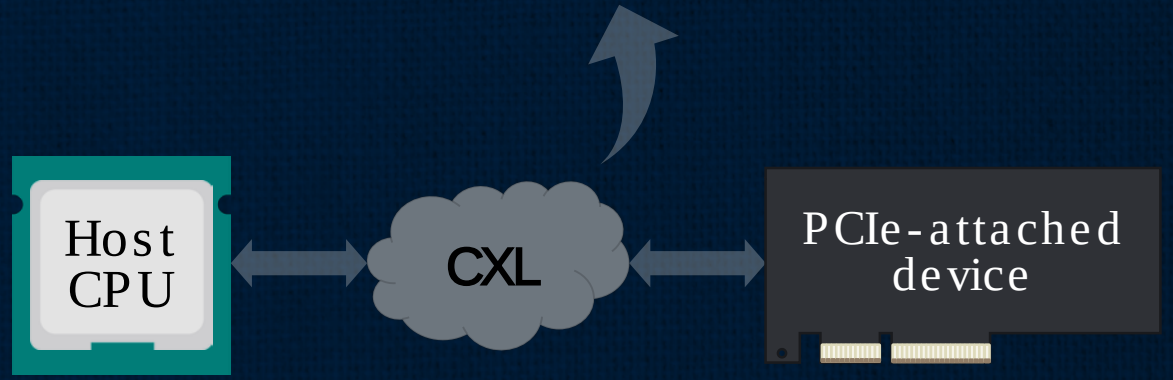
CXL: Multi Protocols



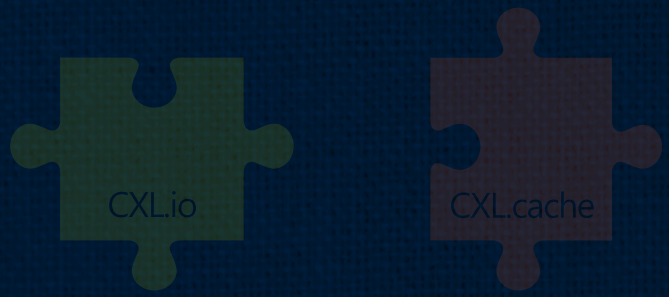
CXL: Multi Protocols

The differences come from how they consider the PCIe-attached device.

Consider PCIe-attached device as:	I/O device	Cache memory	Working memory
Role of protocol:	• CPU-device communication	• Memory requests (device → host) • Coherent requests (host ↔ device)	• Memory requests (host → device)

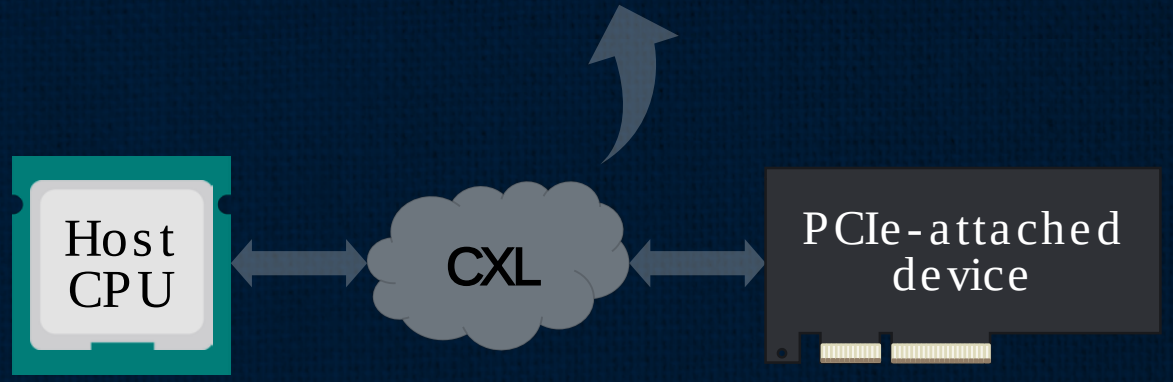


CXL: Multi Protocols

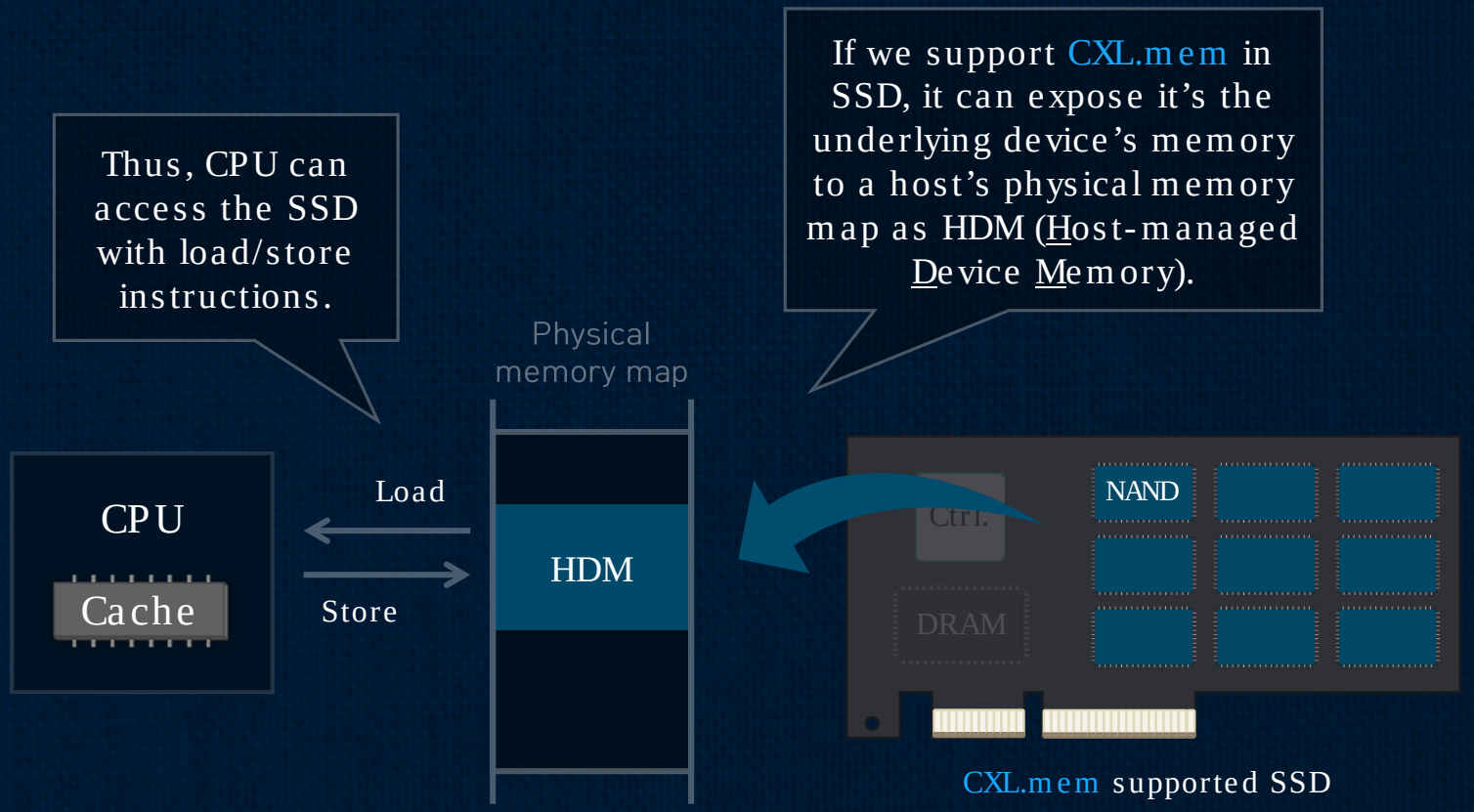


Consider PCIe-attached device as:	I/O device	Cache memory	Working memory
Role of protocol:	• CPU-device communication	• Memory requests (device → host) • Coherent requests (host ↔ device)	• Memory requests (host → device)

Since CXL.mem is what the memory expander mainly requires, let's check how this protocol works.

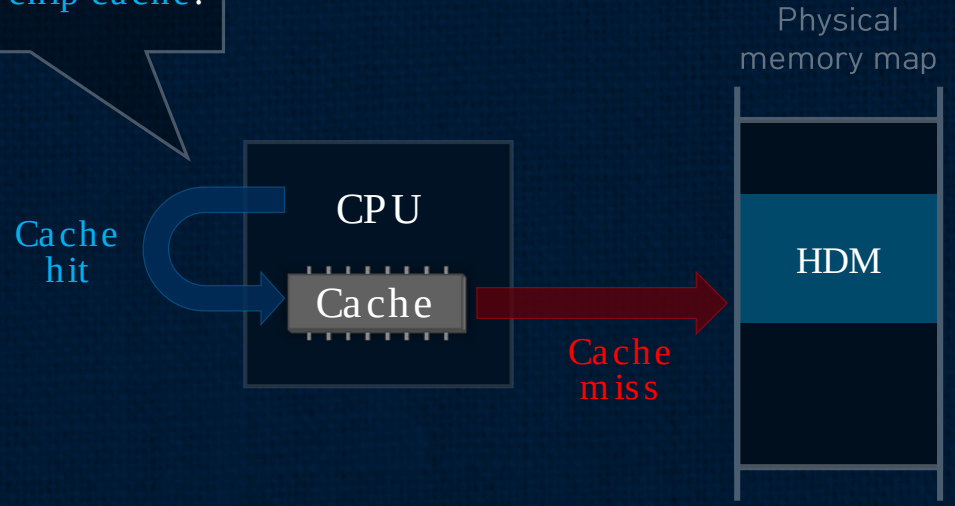


Cacheable Access

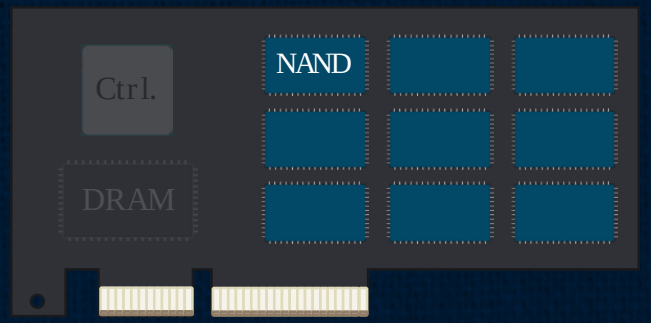


Cacheable Access

Since the host-managed device memory is cacheable, memory access can be hit in the on-chip cache.

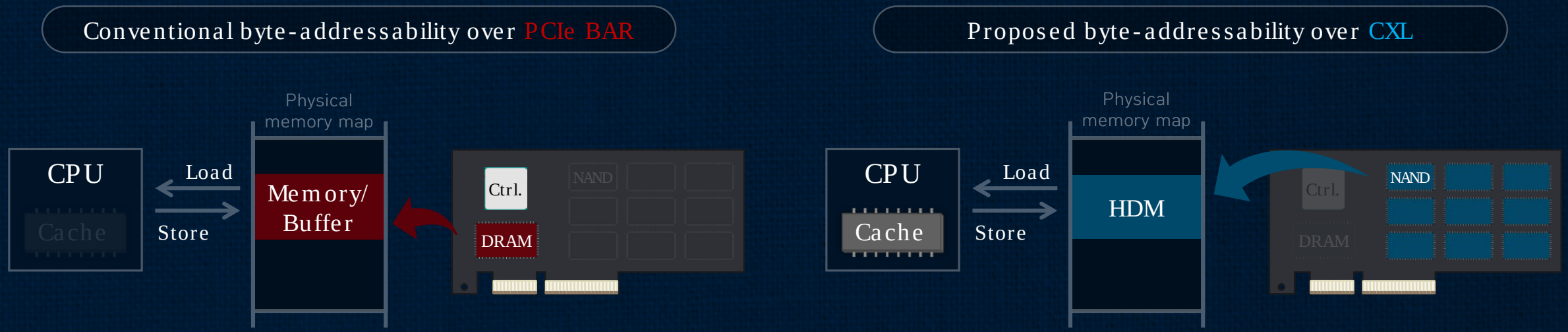


Note that HDM is accessed via CXL interconnection only when cache misses.



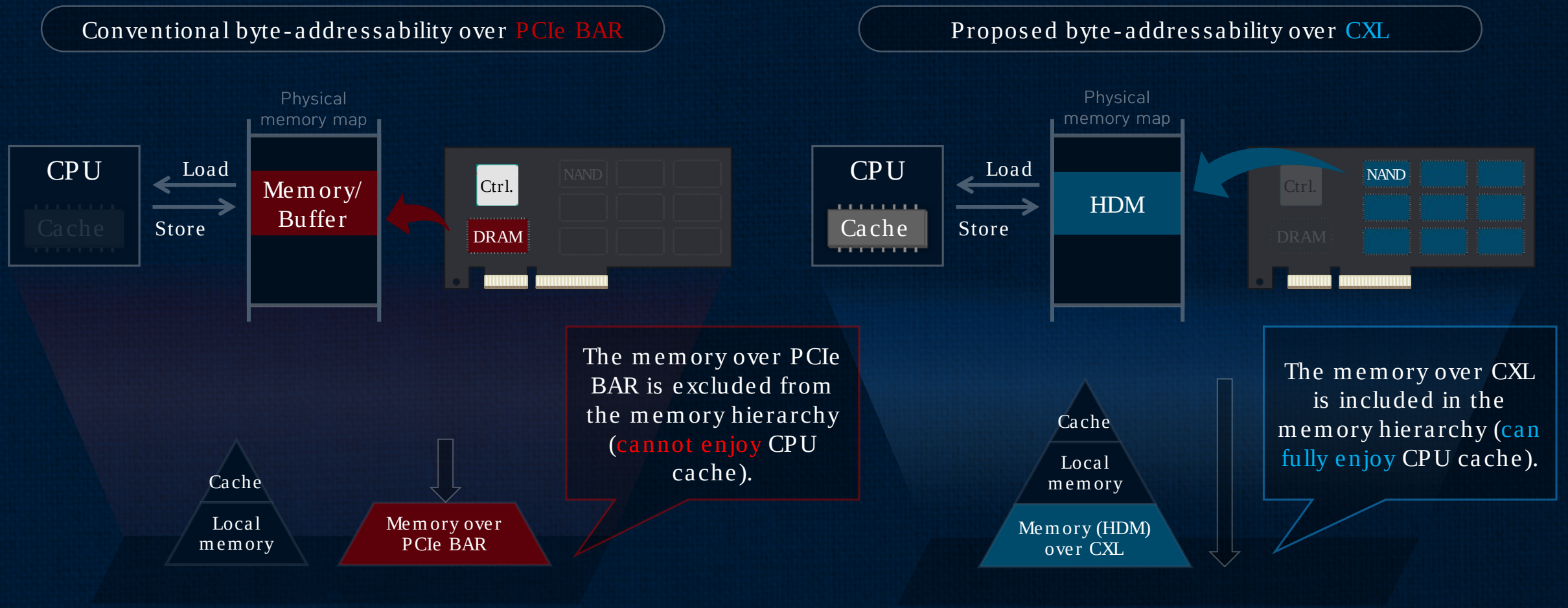
CXL.mem supported SSD

Side-by-Side Comparison



Now, let's compare the conventional and proposed byte-addressability side by side.

Side-by-Side Comparison



1. Long-Standing Dream

2. Why CXL?

3. Storage-Integrated Memory Expander

4. User Guide #1 - Pooling

5. User Guide #2 -Storage-Aware Annotation

Design #1: Device Type Consideration

In order to enable storage-integrated memory expander over CXL, we have to decide CXL device type first.

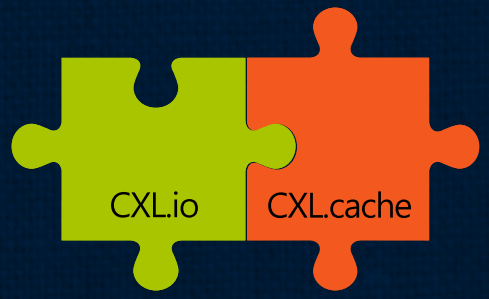
CXL Device: Mix & Match CXL Protocols

We can compose CXL devices by **mixing and matching** CXL protocols.

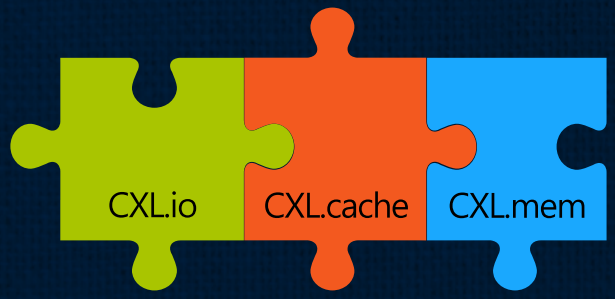
CXL Device: Mix & Match CXL Protocols

There are three types of CXL devices.

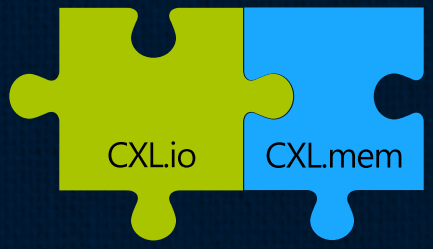
Type 1 device
(CXL.io + CXL.cache)



Type 2 device
(CXL.io + CXL.cache + CXL.mem)

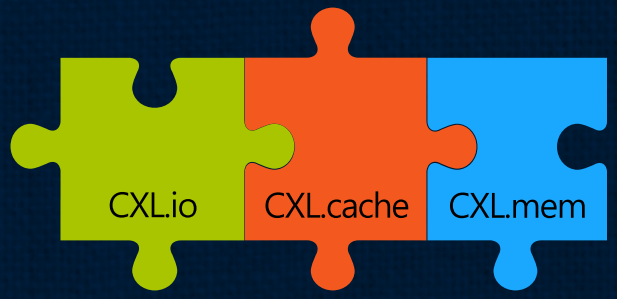


Type 3 device
(CXL.io + CXL.mem)

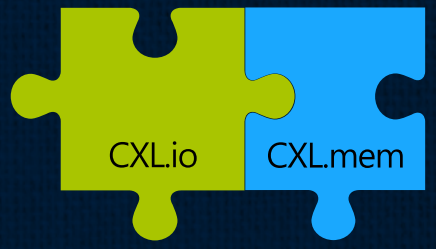


Best-Fit: Type 3 CXL Device

Type 2 device
(CXL.io + CXL.cache + CXL.mem)

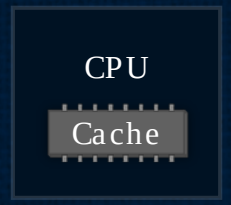


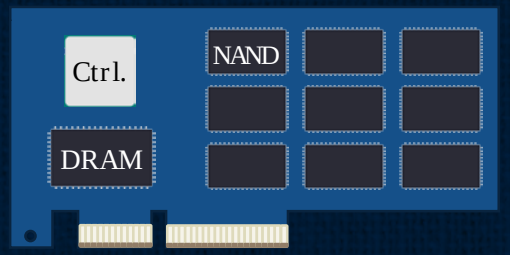
Type 3 device
(CXL.io + CXL.mem)



Between two CXL.mem-supported devices, the type 3 device is the best fit for storage-integrated memory expansion – Why?

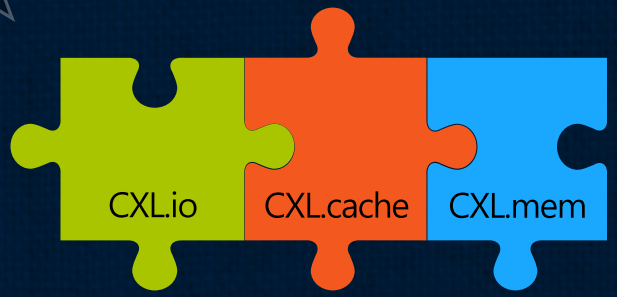
Limits of Type 2 CXL Device





Let's suppose that the CXL SSD is type 2.

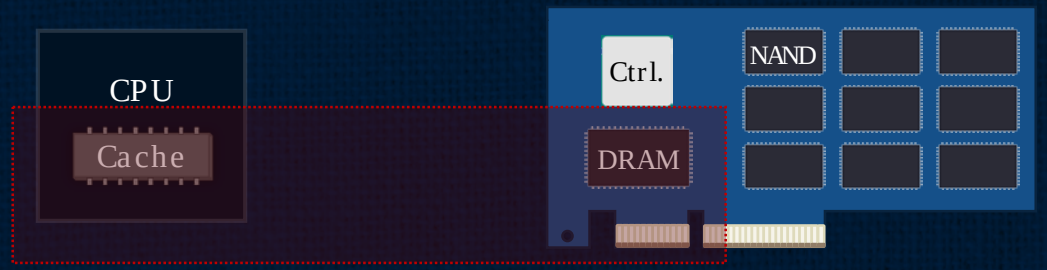
Type 2 device
(CXL.io + CXL.cache + CXL.mem)



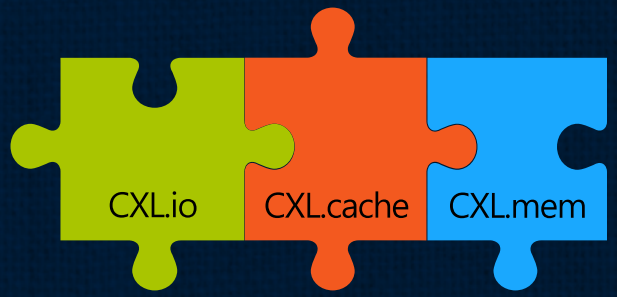
Type 3 device
(CXL.io + CXL.mem)

Limits of Type 2 CXL Device

Then, the address spaces that the CPU and SSD manage should be coherent.

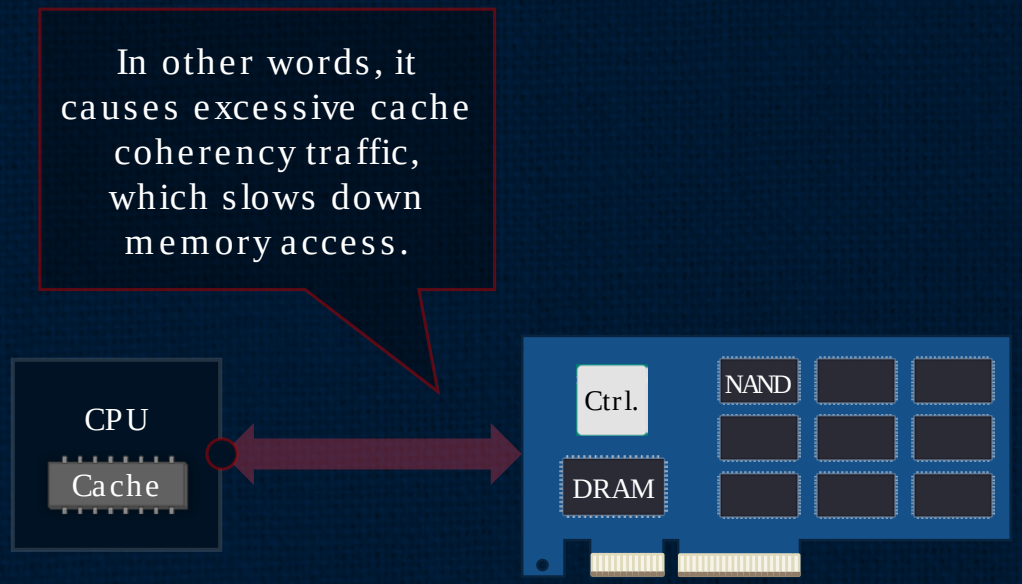


Type 2 device
(CXL.io + CXL.cache + CXL.mem)

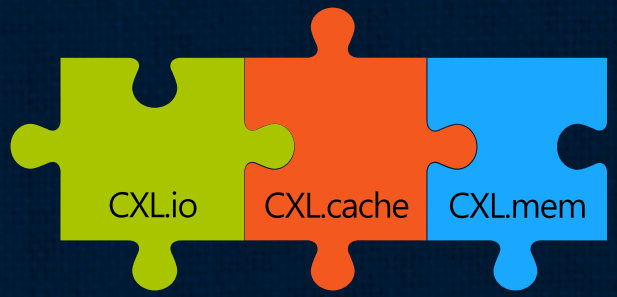


Type 3 device
(CXL.io + CXL.mem)

Limits of Type 2 CXL Device

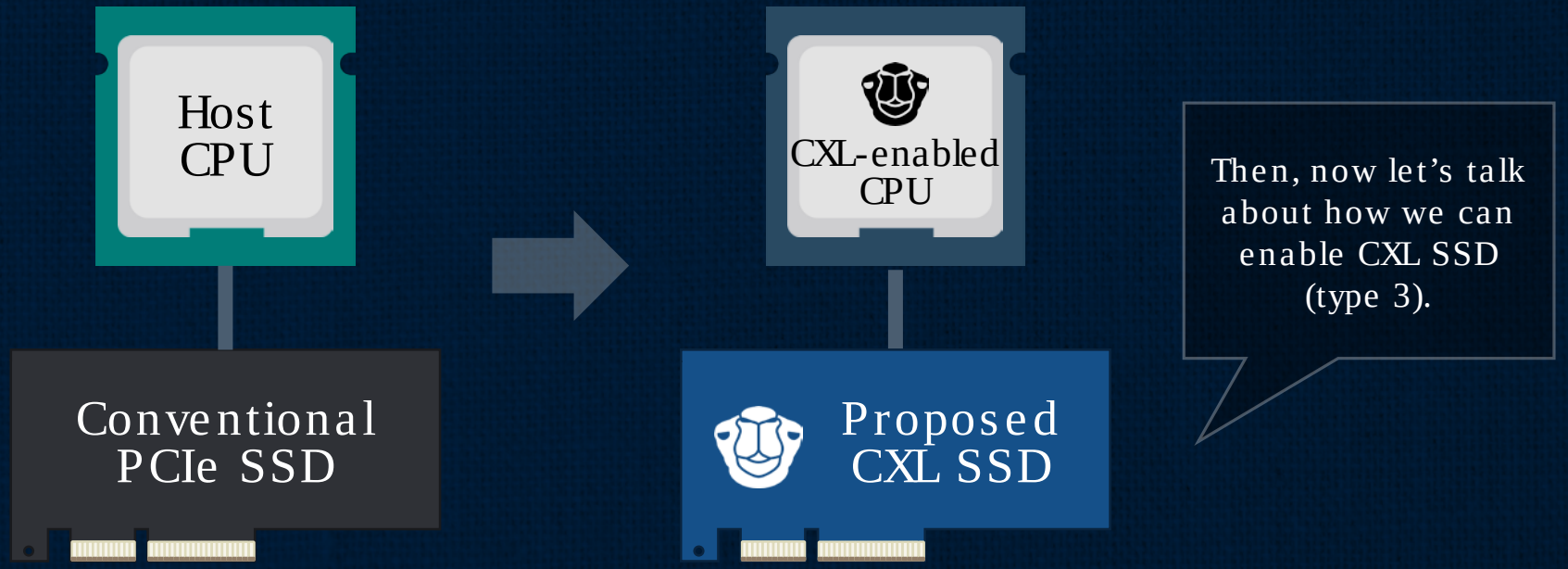


Type 2 device
(CXL.io + CXL.cache + CXL.mem)

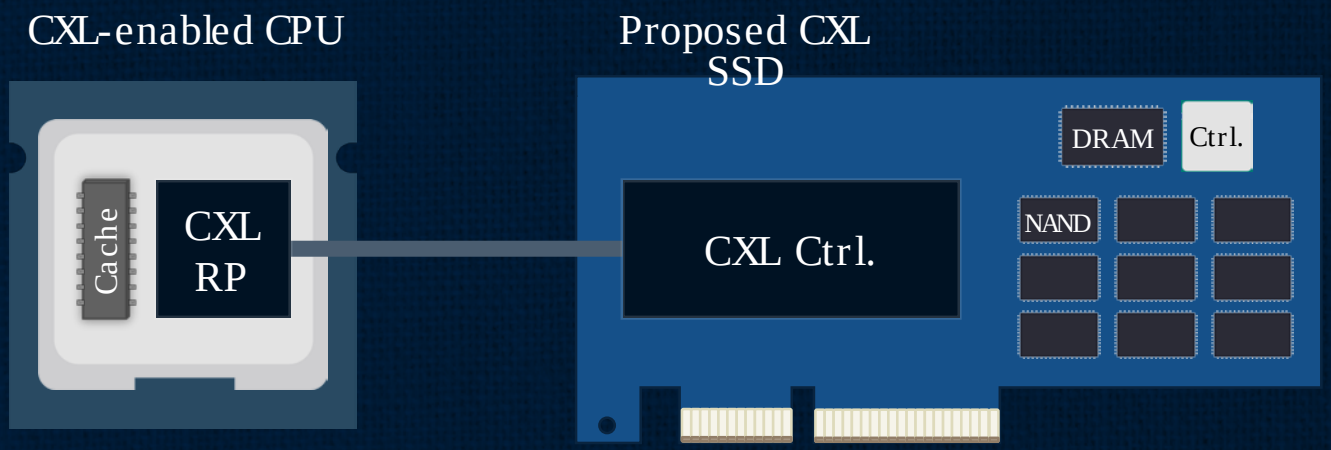


Type 3 device
(CXL.io + CXL.mem)

Design #2: Enable CXL SSD (Type 3)

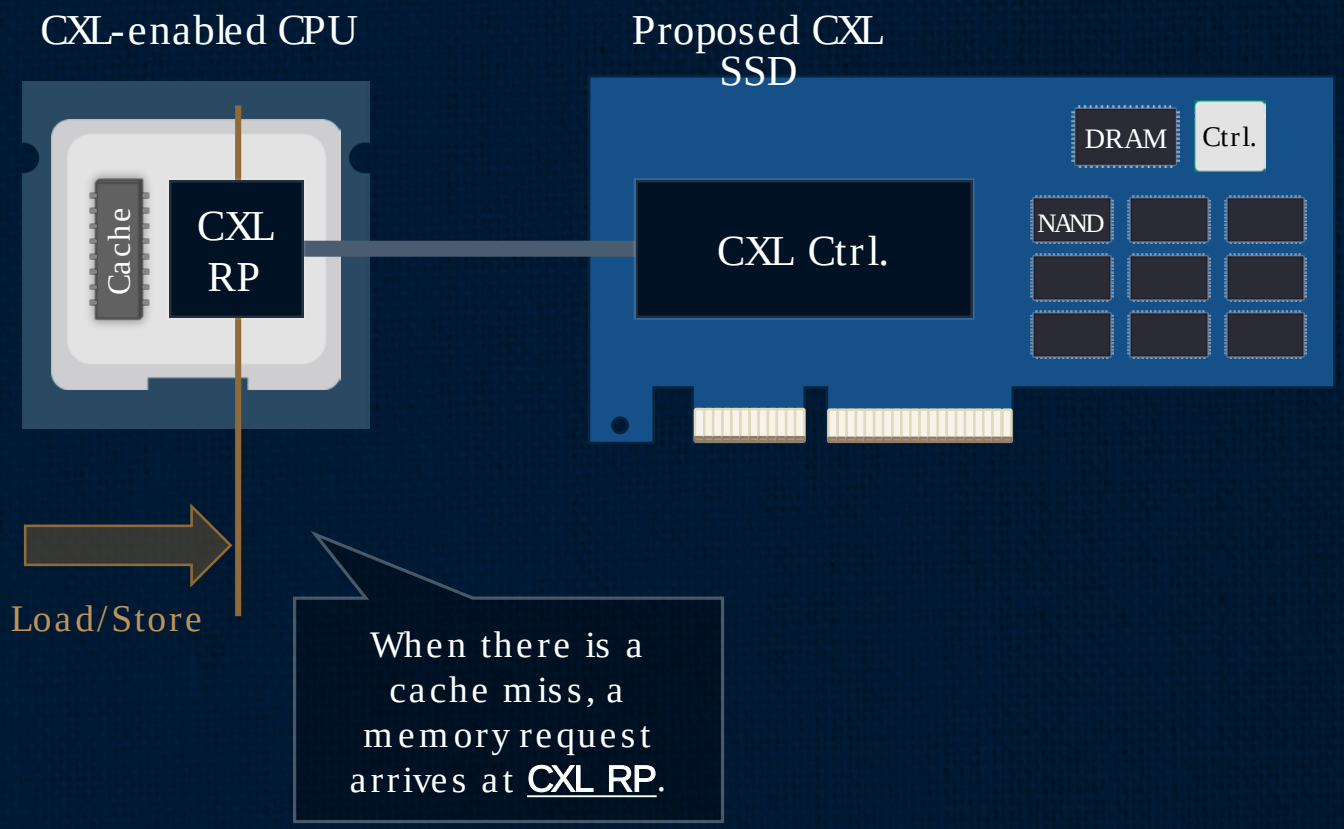


At A Glance: CXL SSD & CXL-enabled CPU

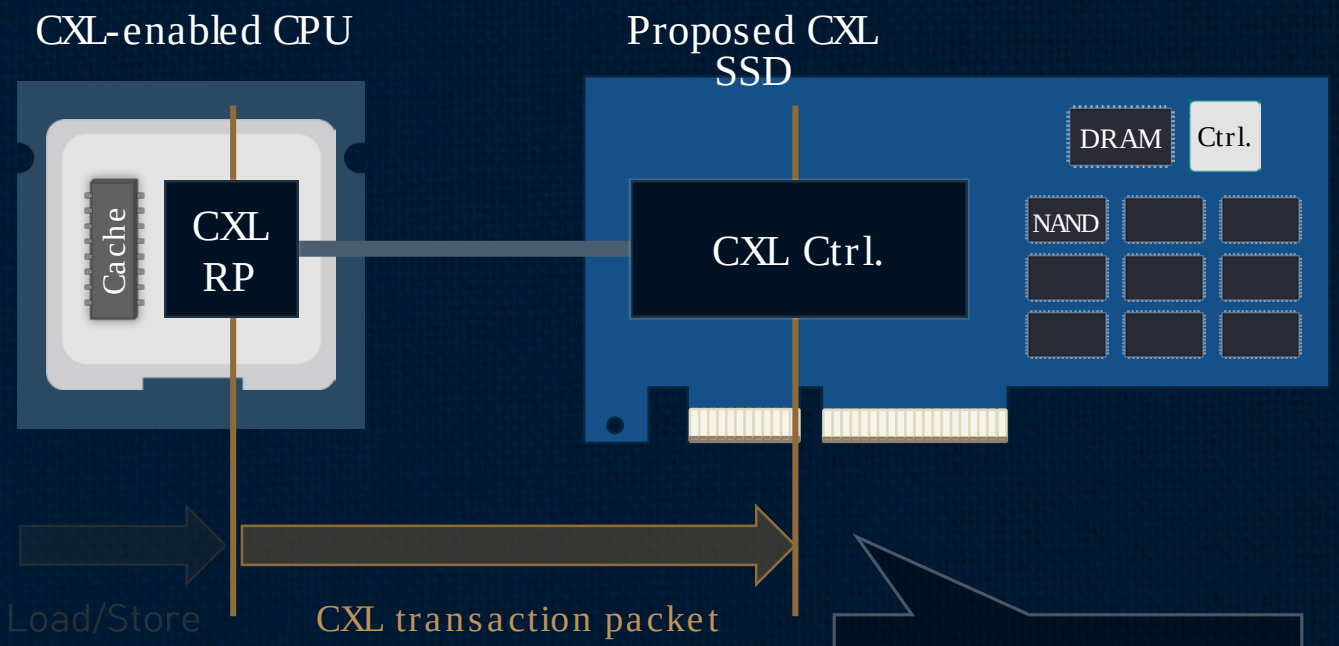


You can see at a glance what CXL SSD and CXL-enabled CPU look like with this drawing.

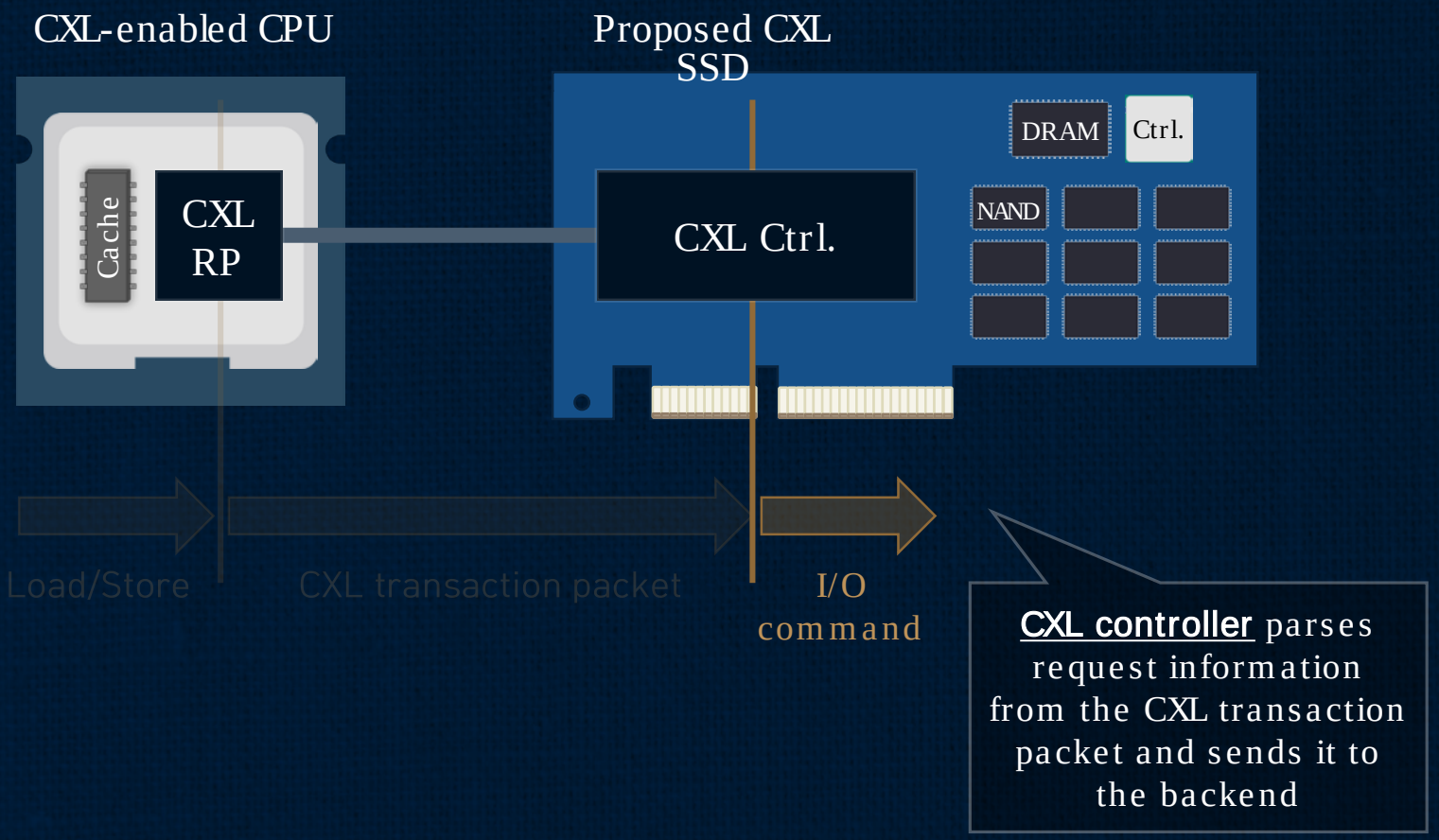
At A Glance: CXL SSD & CXL-enabled CPU



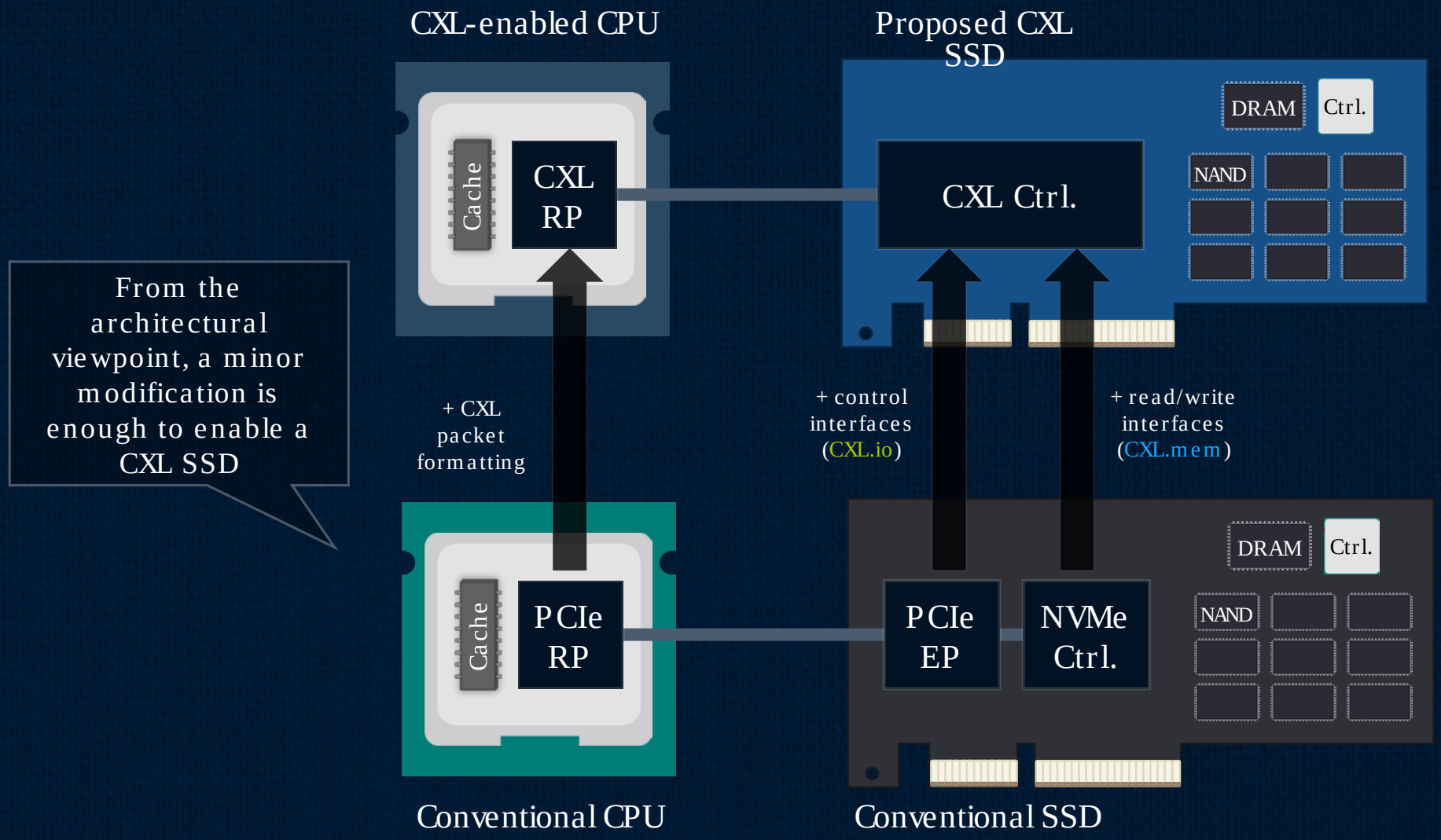
At A Glance: CXL SSD & CXL-enabled CPU



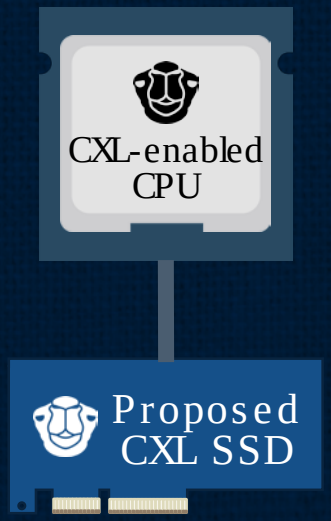
At A Glance: CXL SSD & CXL-enabled CPU



Simple Modification is Enough



Performance Projection



- RISC-V 64 bit O3
- 128 KB L1 cache
- 4MB L2 cache

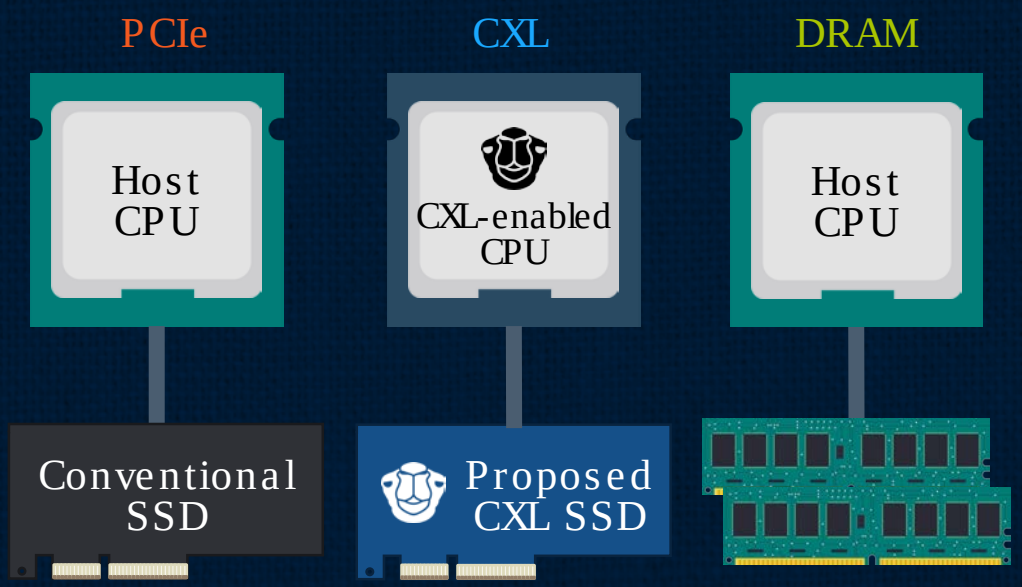
- Z-NAND emulation
- 32GB capacity
- OpenExpress-based

※Separated customized FPGA board (16nm)

We speculate how much effect a CXL SSD has on system performance using our CXL hardware prototype.

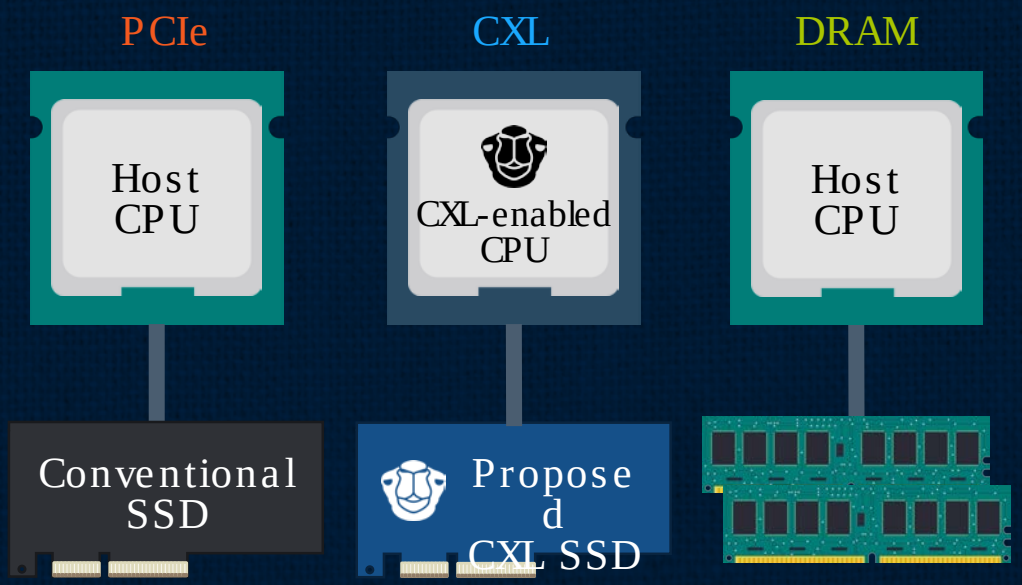
RISC-V OpenExpress

Experimental Group

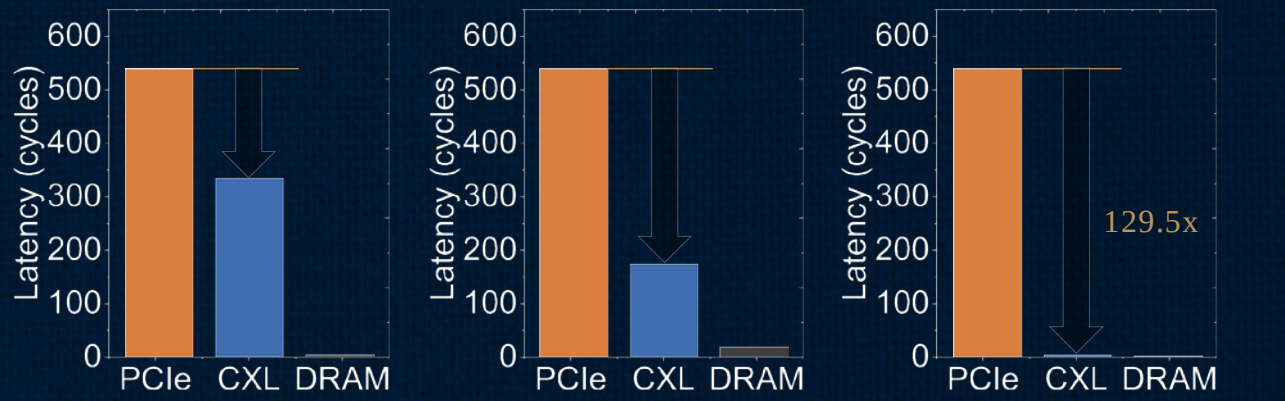


In addition, we evaluate a local DRAM-only system (DRAM) and PCIe BAR-based memory expander (PCIe).

Result Analysis



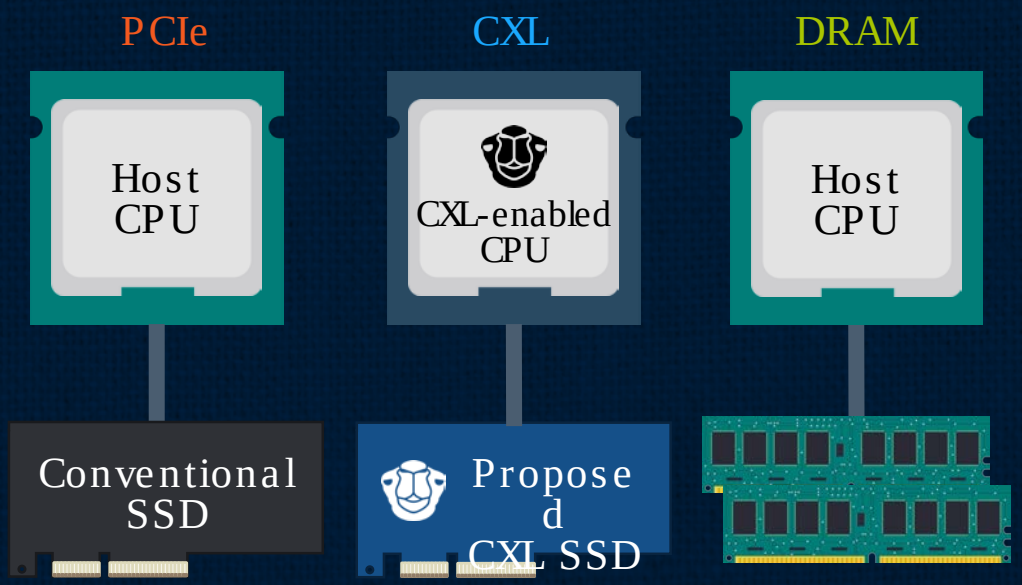
When we compare CXL with PCIe, CXL performance is much better, thanks to cache hits.



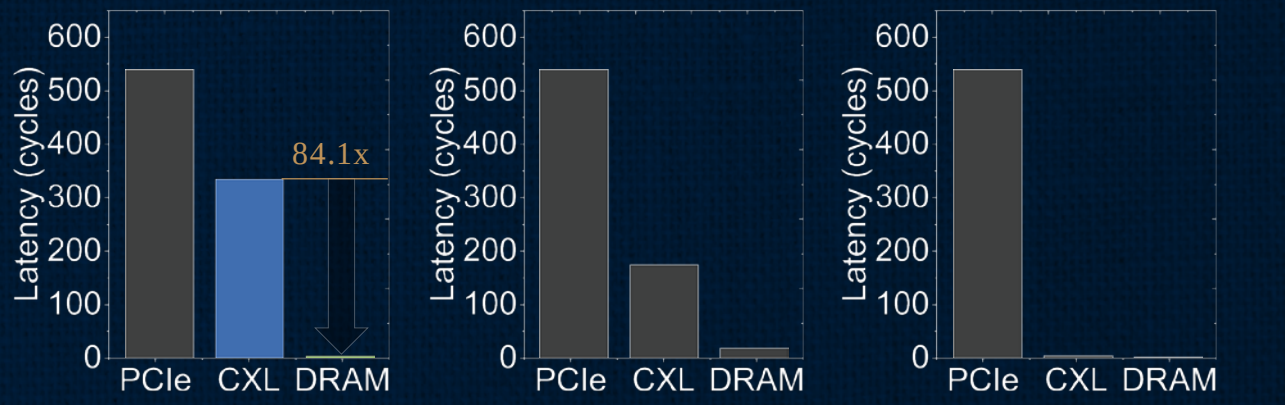
α=1 Average α α=0.001

Lowest locality Highest locality

Result Analysis



When we compare CXL with DRAM, CXL is 84.1x slower than DRAM if it cannot enjoy cache hit.

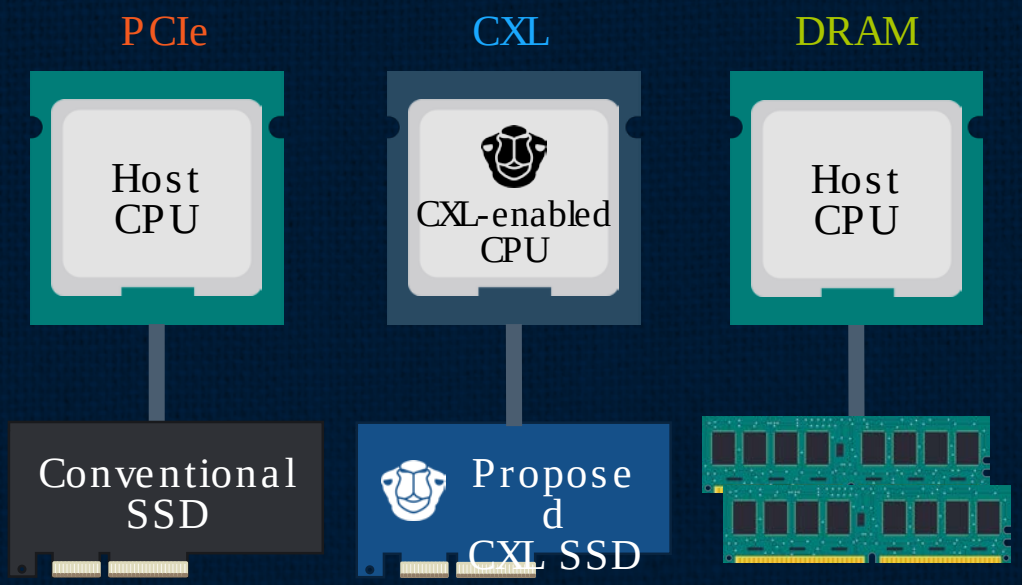


α=1 Average α α=0.001

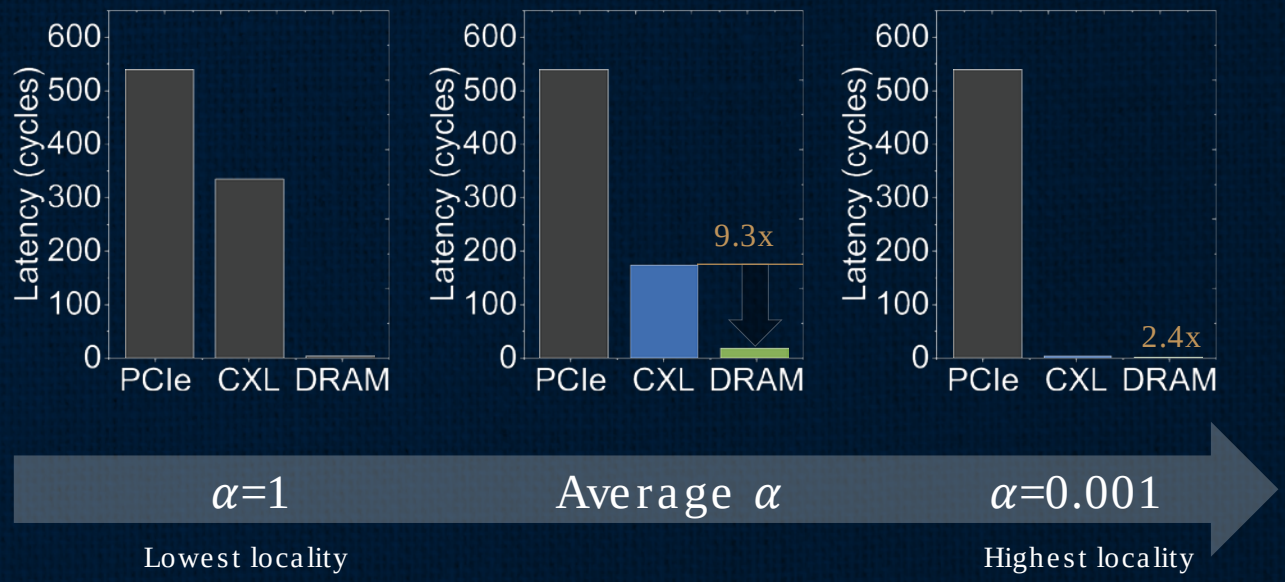
Lowest locality

Highest locality

Result Analysis



However, it could be okay as the lowest locality is rare in real applications; CXL can achieve performance similar to DRAM on average or high locality.



1. Long-Standing Dream

2. Why CXL?

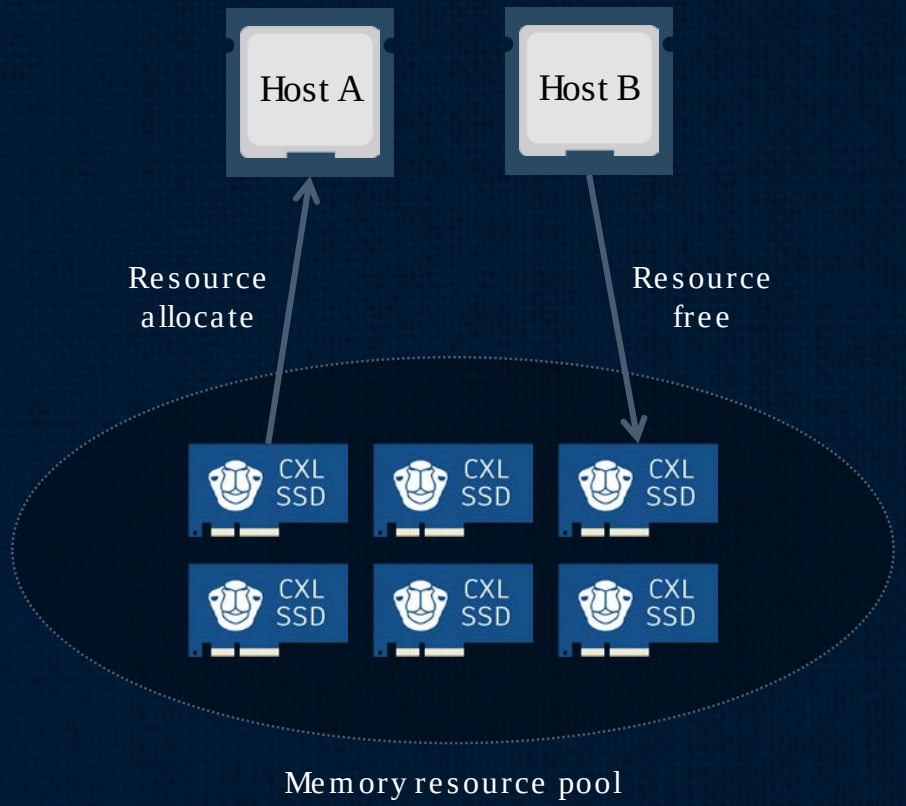
3. Storage-Integrated Memory Expander

4. User Guide #1 – Pooling

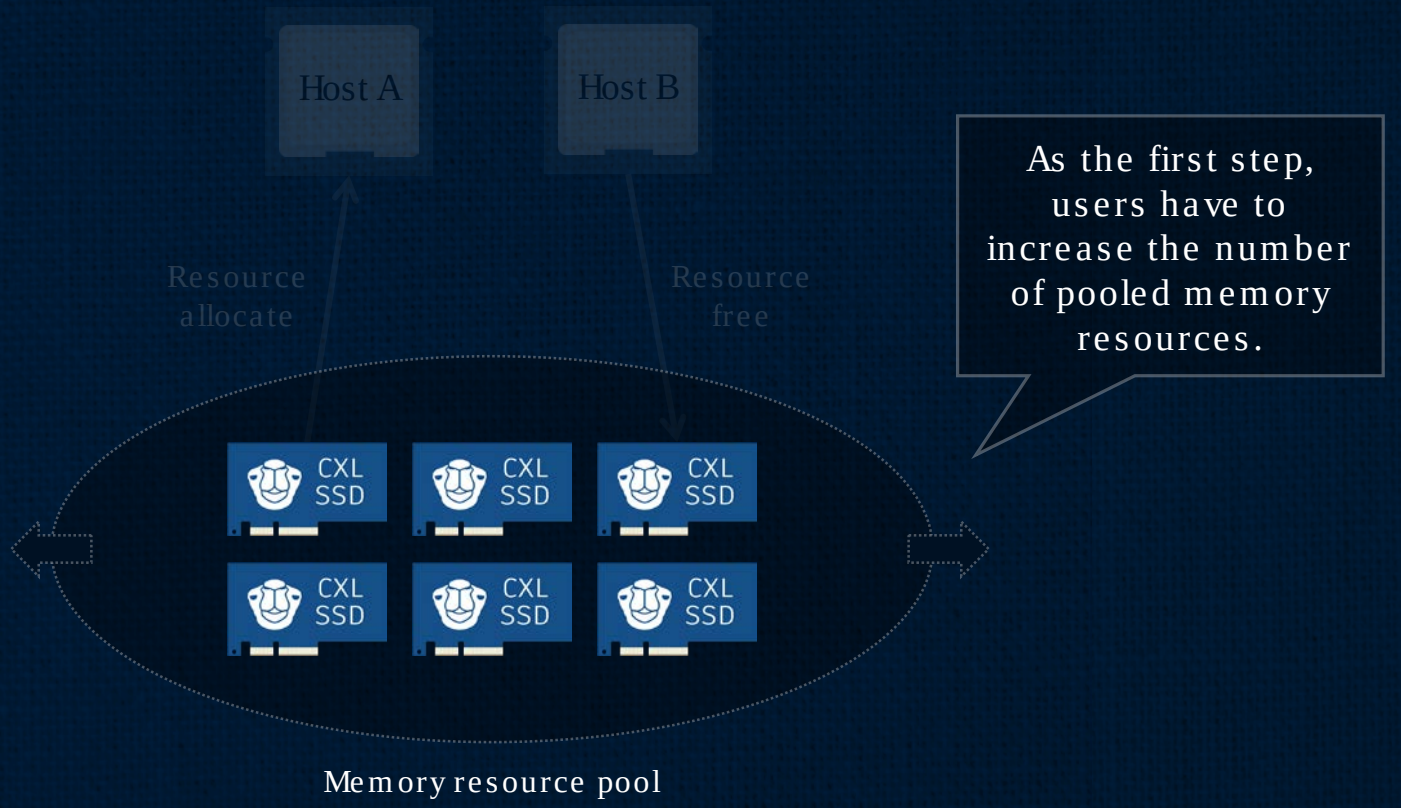
5. User Guide #2 -Storage-Aware Annotation

Needs: Memory Pooling

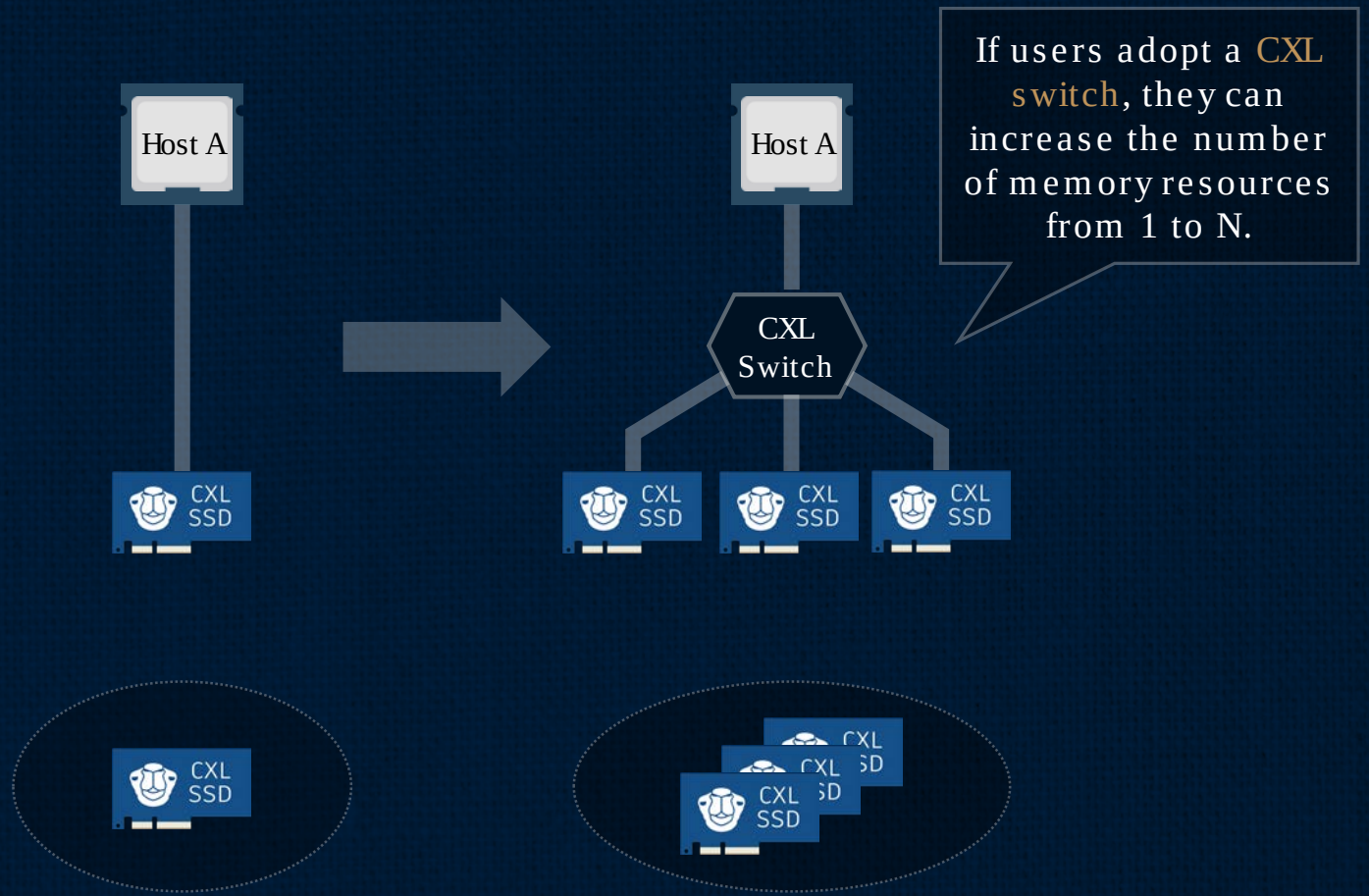
Users might want to maintain a memory pool that allows allocate or free the memory resources.



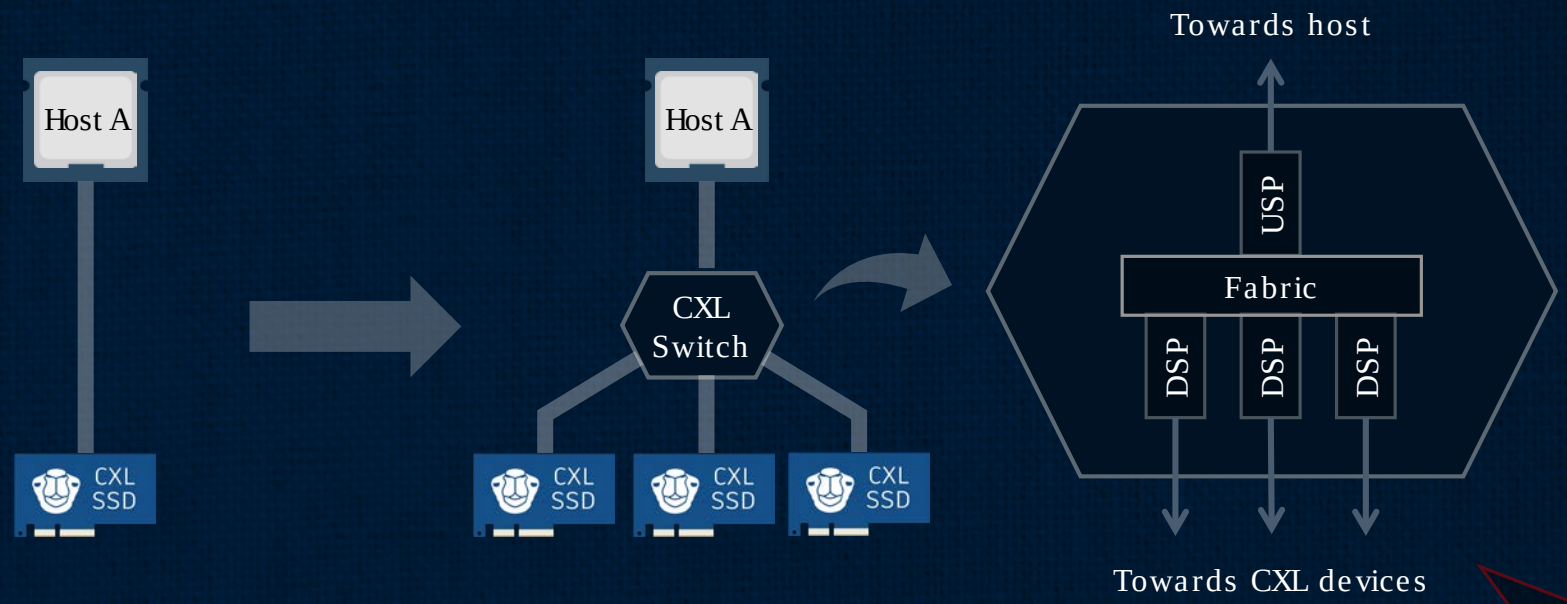
Guide #1: Resource Expansion



#1-1: CXL Switch



#1-1: CXL Switch



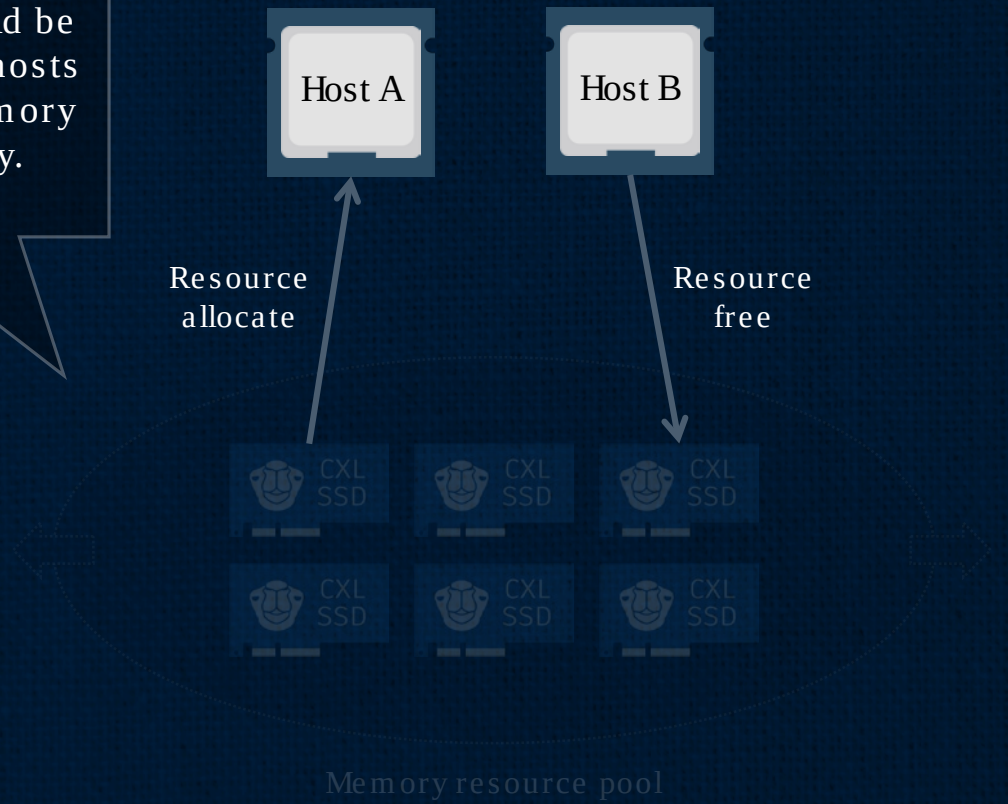
memory resource = 1

memory resource = N

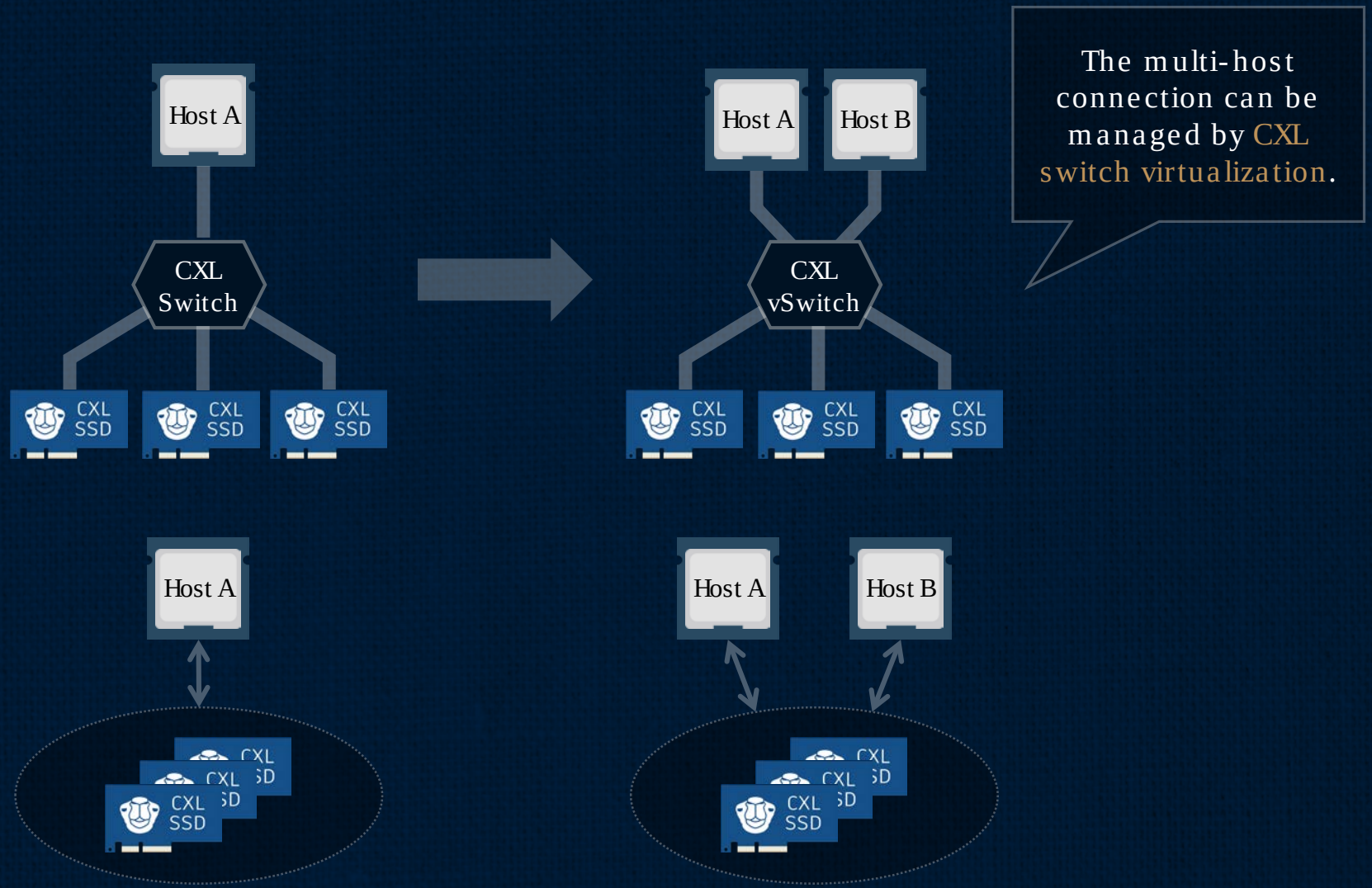
The switch has a fabric manager that can connect USP(s) and DSP(s) based on what the system demands

Guide #2: Resource Pooling

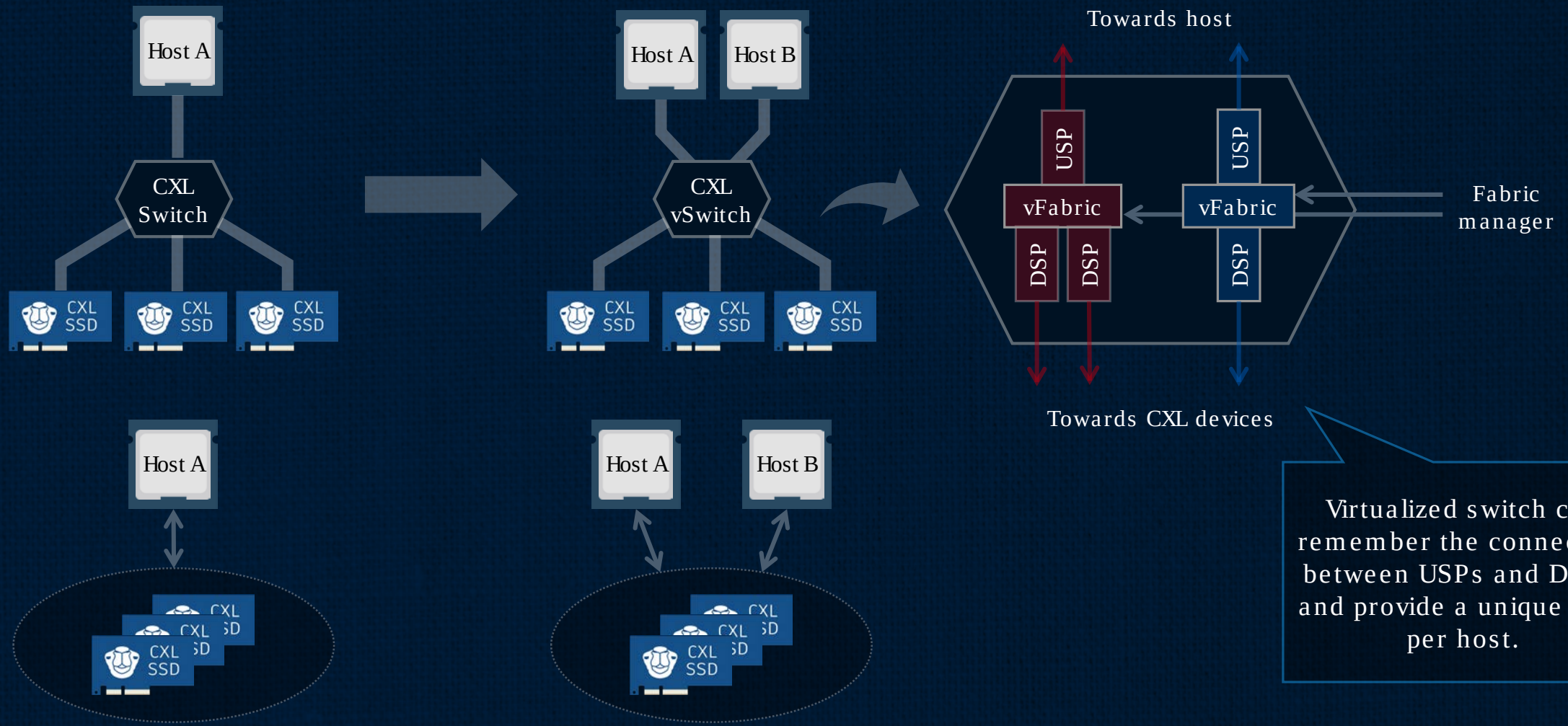
As the second step, resource pooling should be supported by multiple hosts when the multiple memory expanders are ready.



#2-1: Switch Virtualization

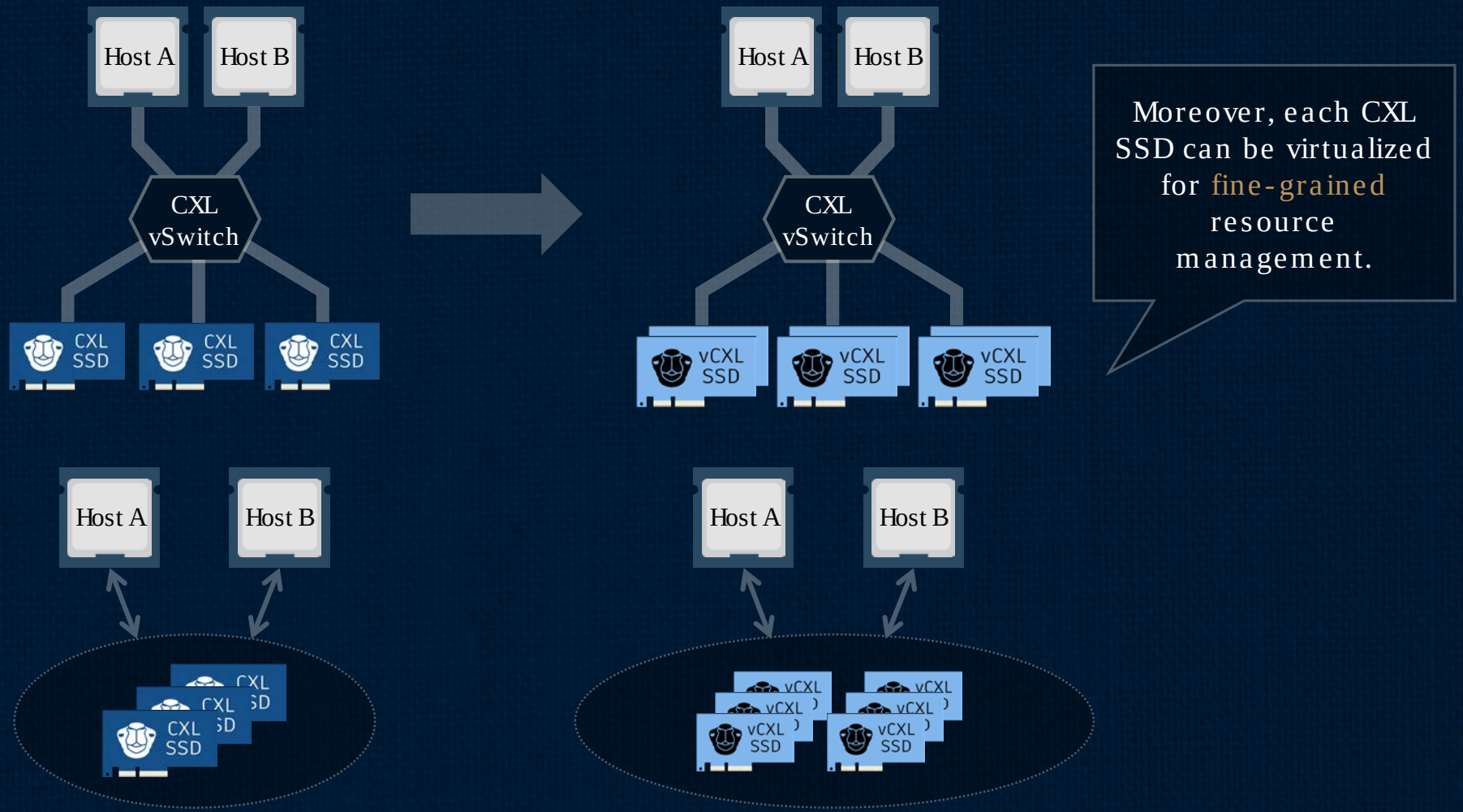


#2-1: Switch Virtualization

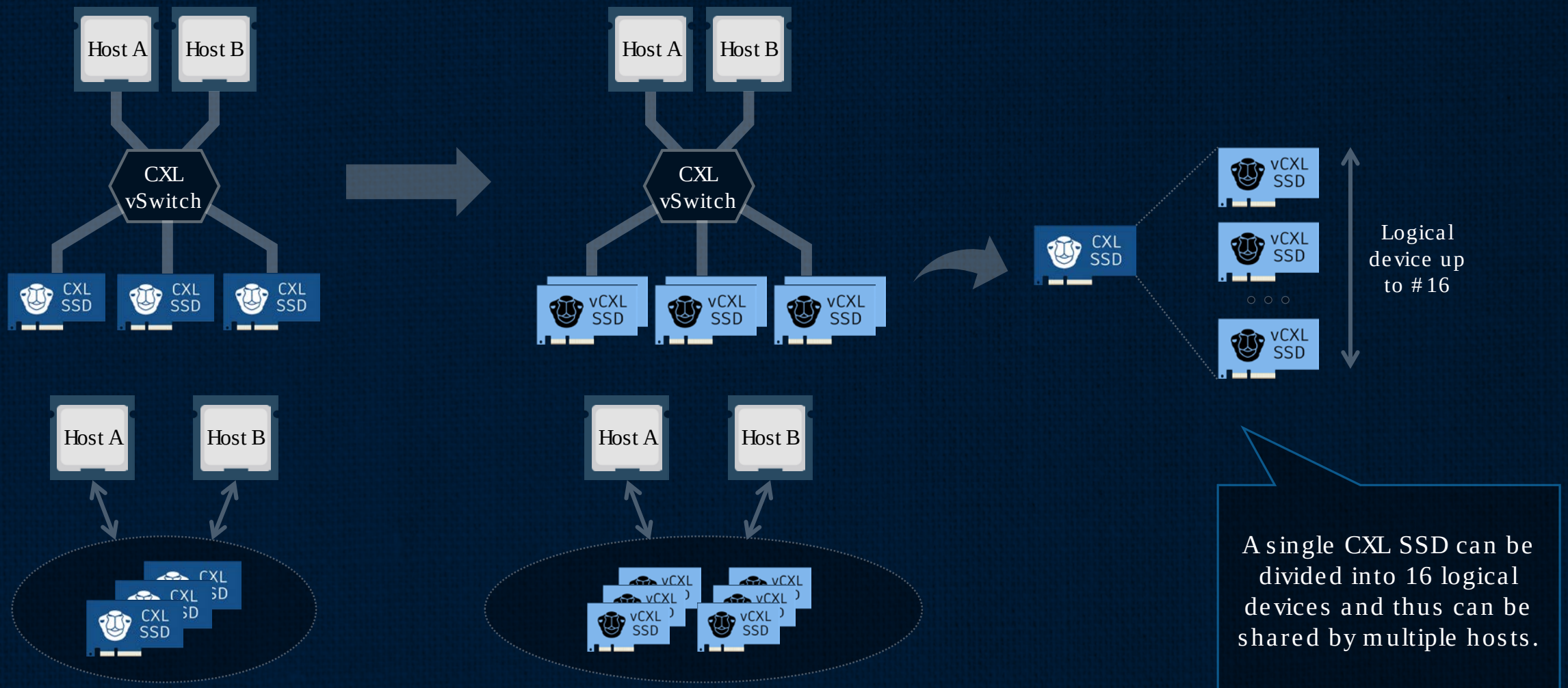


Virtualized switch can remember the connection between USPs and DSPs, and provide a unique path per host.

#2-2: Device Virtualization



#2-2: Device Virtualization



1. Long-Standing Dream

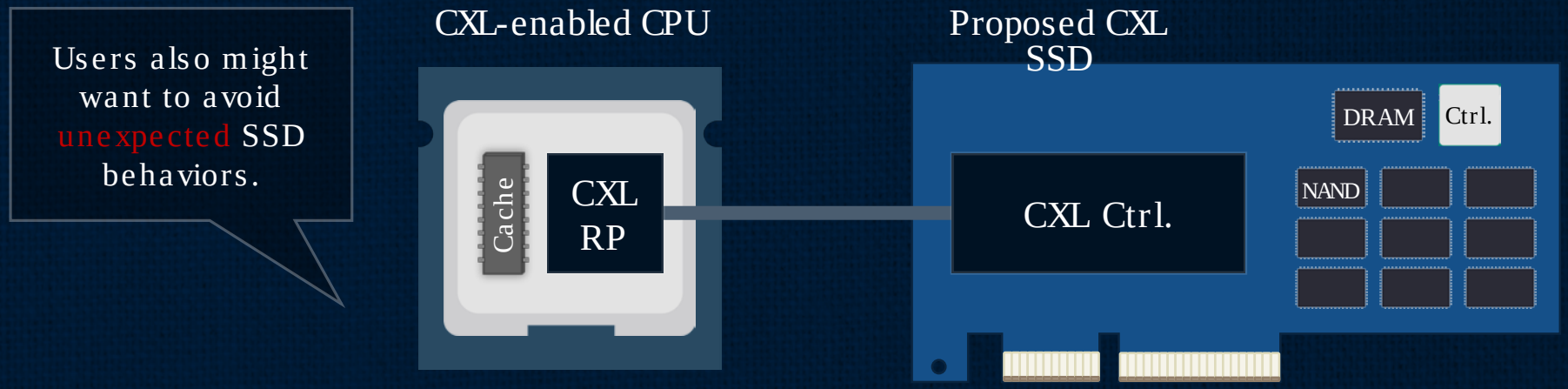
2. Why CXL?

3. Storage-Integrated Memory Expander

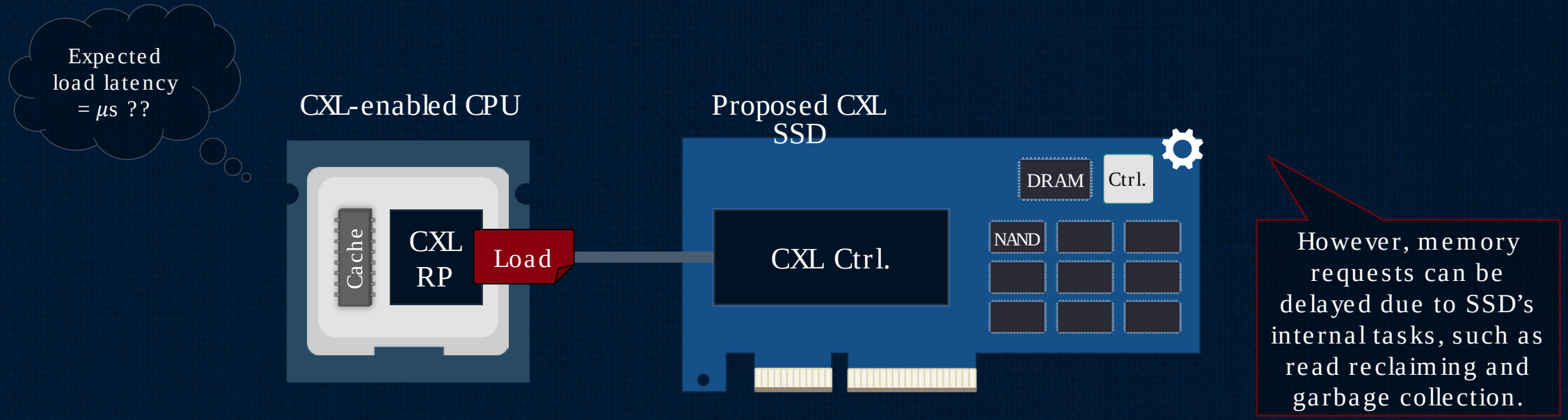
4. User Guide #1 – Pooling

5. User Guide #2 – Storage-Aware Annotation

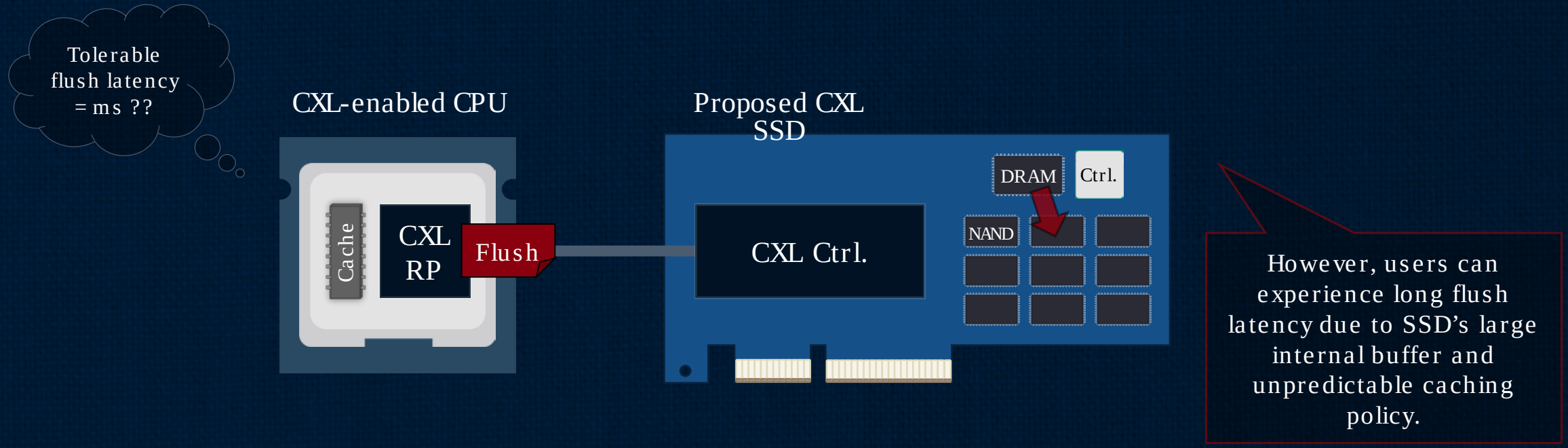
Needs: Latency & Persistence Control



Unexpected Case #1: Internal Tasks

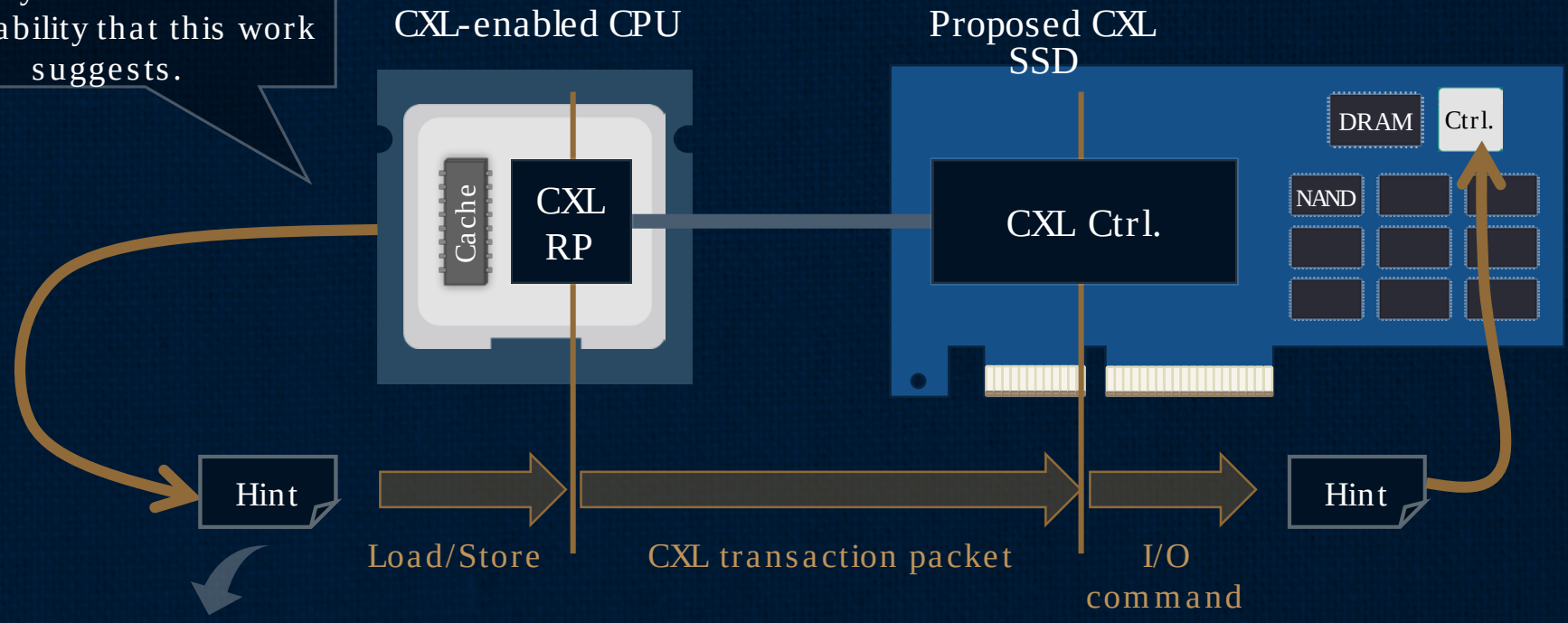


Unexpected Case #2: Internal Buffer



Annotations for Hint

Thus, we suggest an annotation method that allows users to send a hint to an SSD, such as latency determinism or bufferability that this work suggests.



Then SSD controller behaves appropriately based on which hint is transferred.

Determinism	**DT**: deterministic	**ND**: Non-deterministic
Bufferability	**BF**: Bufferable	**NB**: Non-bufferable

Conclusion

We study how CXL can be applied to PCIe storage to fill the semantic gap between bytes and blocks by exploring:

1) Success in converting the block semantics to byte semantics over CXL
2) Speculate the performance with a CXL hardware prototype
3) Explore new opportunities to use the memory expander efficiently

Thank You

Contact: Myoungsoo Jung (mj@camelab.org)

KAIST CAMEL