August 21, 2022

CXL Memory Challenges

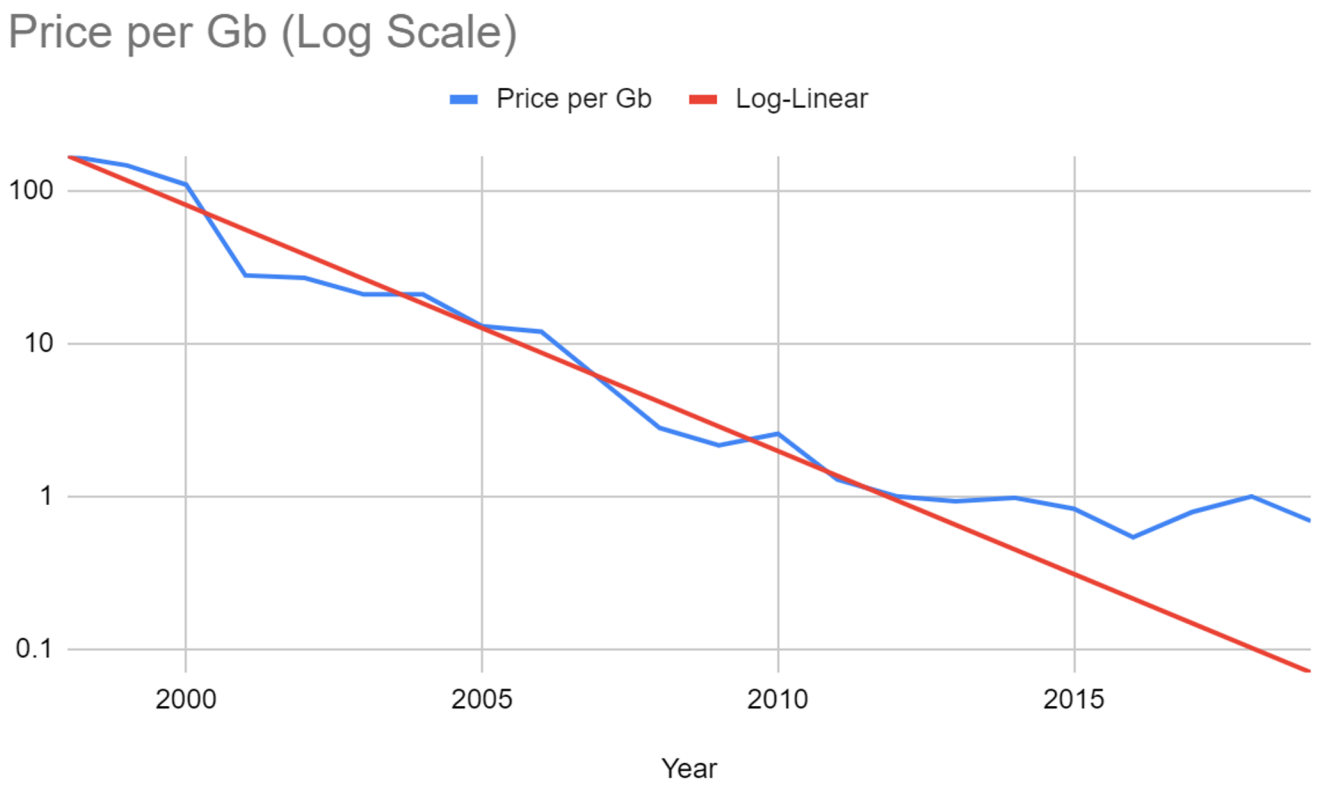
Prakash Chauhan, Meta & Mahesh Wagh, AMD

Agenda

- Memory trends and challenges
  - Cost, bandwidth, capacity
- CXL enabled solutions
  - heterogeneous, tiered, cost and performance optimized
- Evolution of CXL attached memory
  - Simple expanders -> interleaved expanders -> pooled memory -> FAM
  - AI/ML specific topologies
- CXL attached memory challenges
  - Tiered memory performance – making it transparent
  - Deployment challenges - RAS, telemetry, servicing, security

Server Memory – Challenging Trends

- Memory an increasing fraction of system sost
  - Memory Price (cost/bit) flat due to scaling challenges
- Increasing core counts and new workloads driving memory demand
  - Increased Capacity
  - Increased Bandwidth



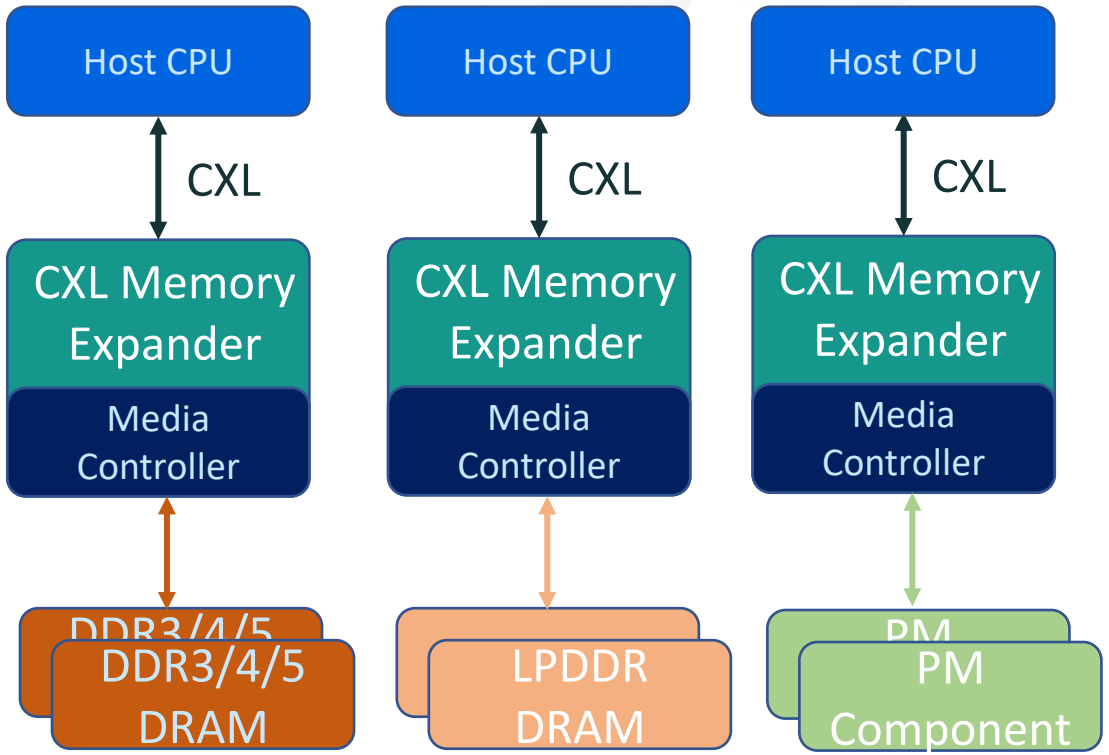
Data Source: De Dios & Associates

System Level Challenges

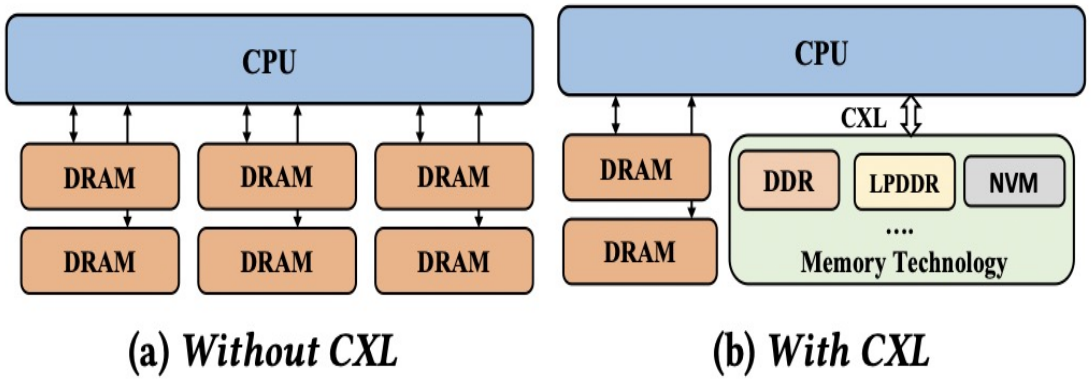
- Adding DDR channels to CPU for bandwidth and capacity
  - Large CPU sockets
    - Cost, Reliability
  - PCB layer count
    - Additional layer per channel
  - Board form-factor
    - Difficulty fitting in standard widths
- Increasing data rates for bandwidth
  - PCB technology
    - Back-drill, SMT connectors, blind vias
  - Equalization circuits
    - Complexity, cost added to both ends
  - 1DPC
    - Capacity/Granularity Issues

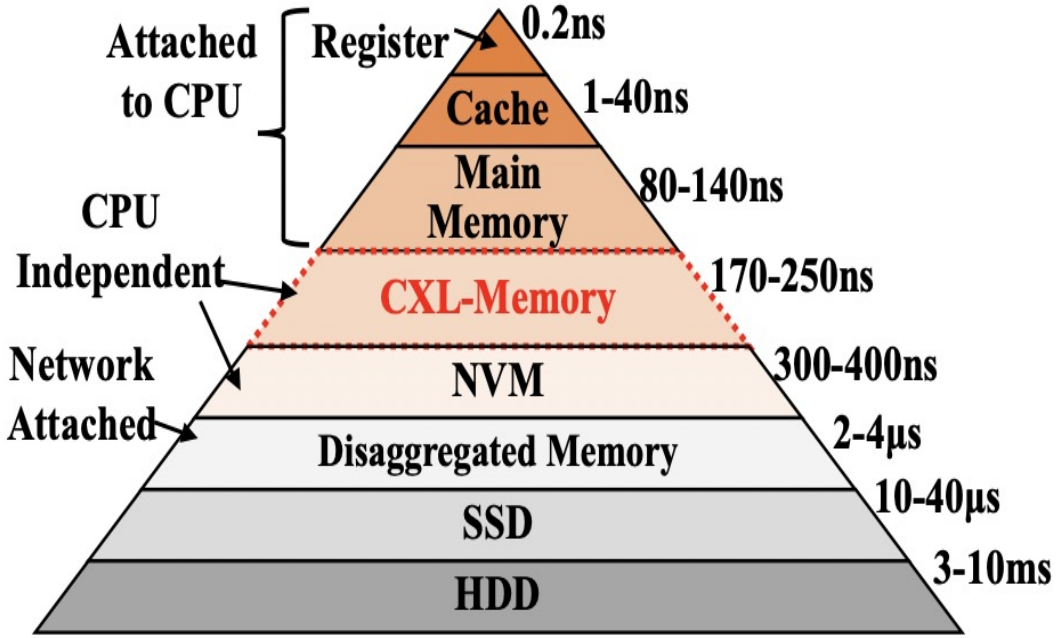
CXL: a media-agnostic memory interface

- A common, standard interface for many types of memory
- Enables system flexibility to make use of different media characteristics
  - persistence, latency, BW, endurance, etc
- Enables usage of heterogeneous memory tiers
- Differential signalling

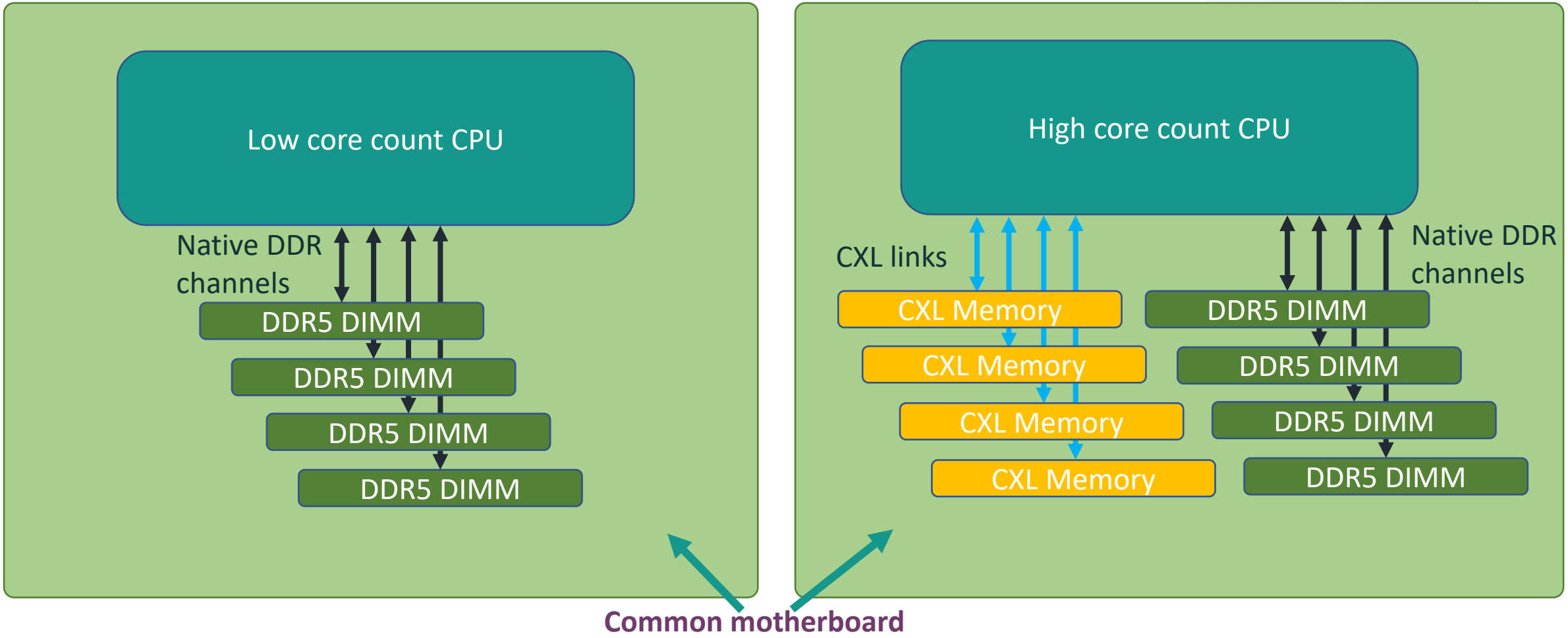


Tiers of happiness



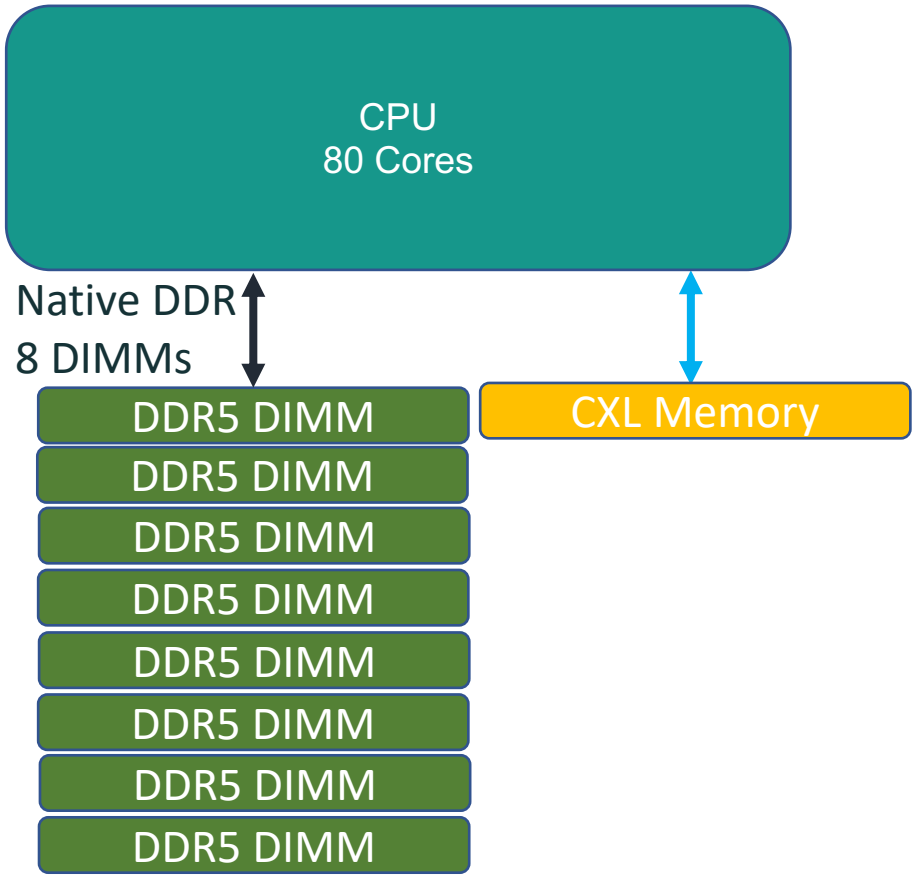


CXL enables System design flexibility



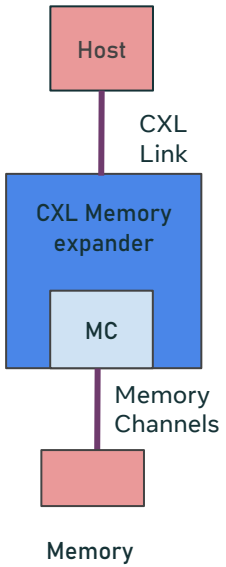
Enables flexibility to add a variety of memory without impacting natively attached DIMMs
CXL Memory can be optimized independently for system cost, capacity, power, bandwidth

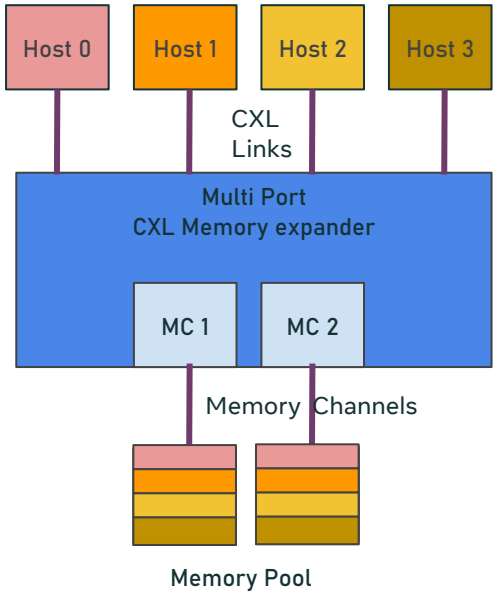
CXL Cost savings example

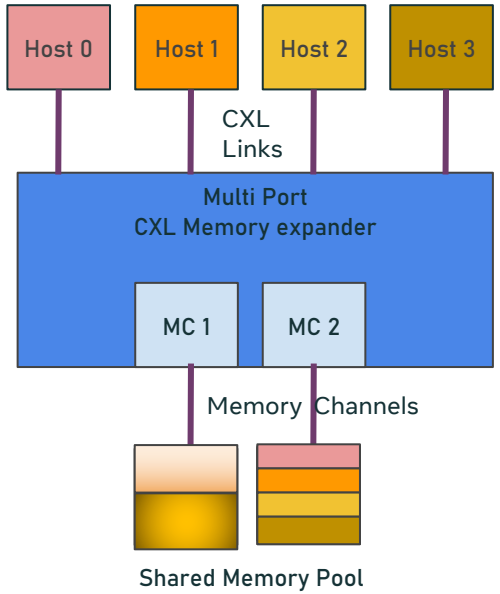


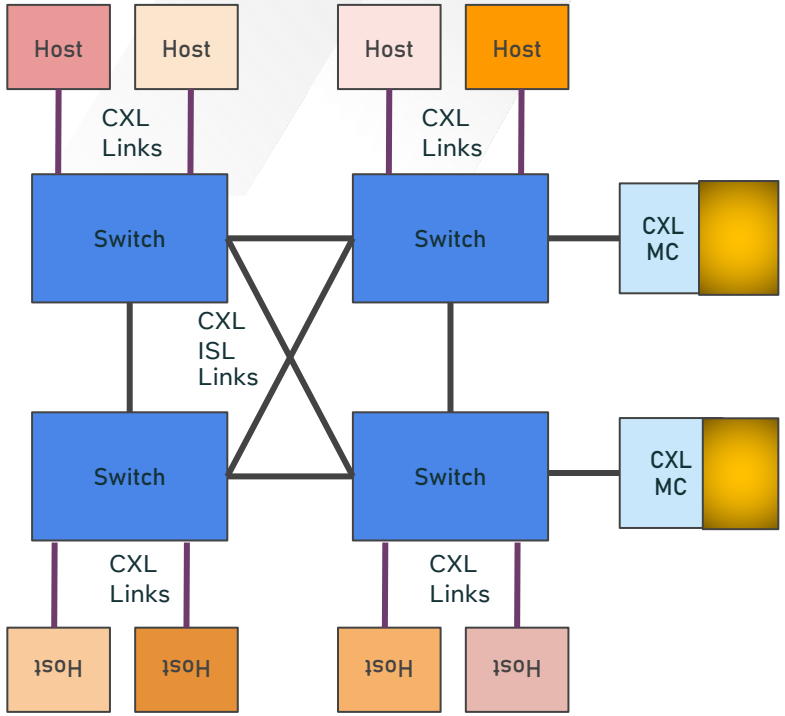
- Assuming 2GB per core – Need 160GB of memory
- 16GB DIMMs x8 = 128GB
- 32GB DIMMs x8 = 256GB
- 32GB DIMMs x5 – Lost BW and Perf
- 16GB DIMMs x8 + 32GB CXL memory
  - Right size with added Bandwidth

CXL Evolution: Flexible Fungible memory









Direct attached

Add Capacity
Add Bandwidth
Slower-cheaper tier

Pooled Memory

Amortize CXL infra cost
Flexible allocation

Shared Memory

Deduplication
Host2host communication
large datasets

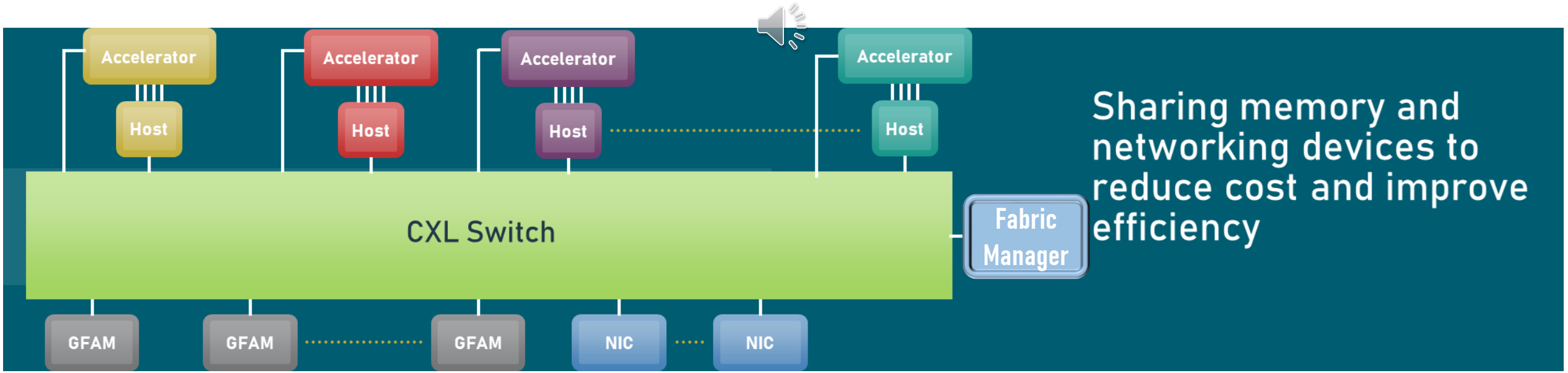
Fabric Memory

Scaling to huge datasets

AI Memory Topologies

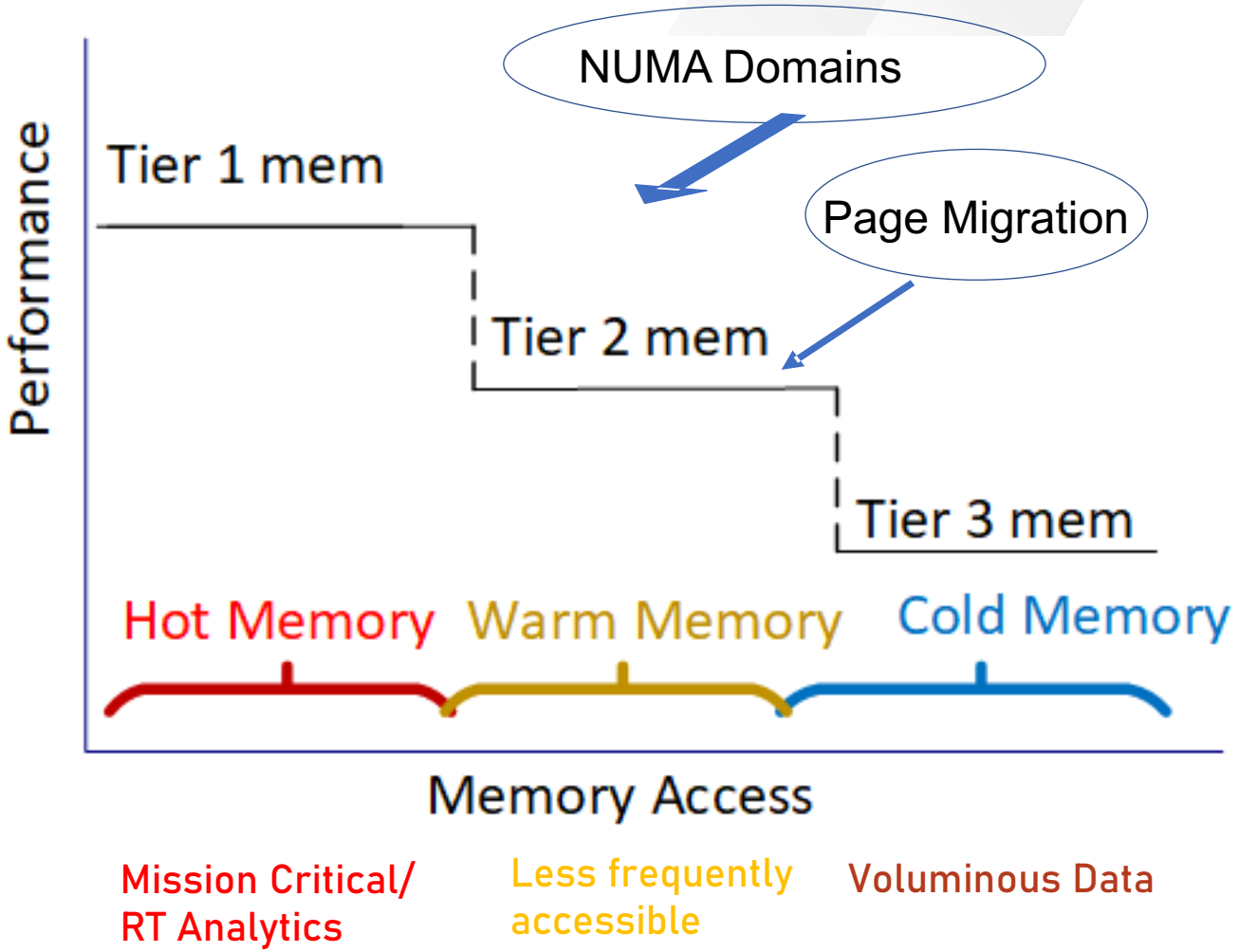
- AI/ML/HPC/HPDA workloads consume significant amount of power

- Optimize compute to happen near the data
- Enable near and in-memory compute

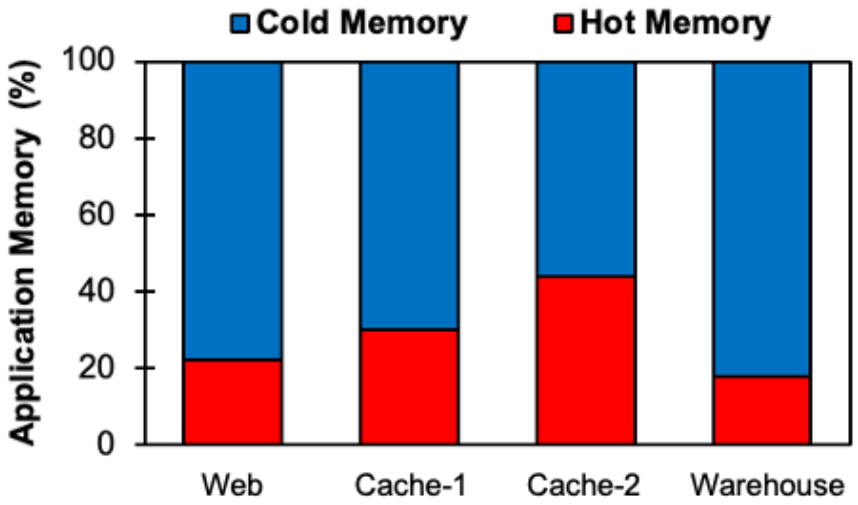


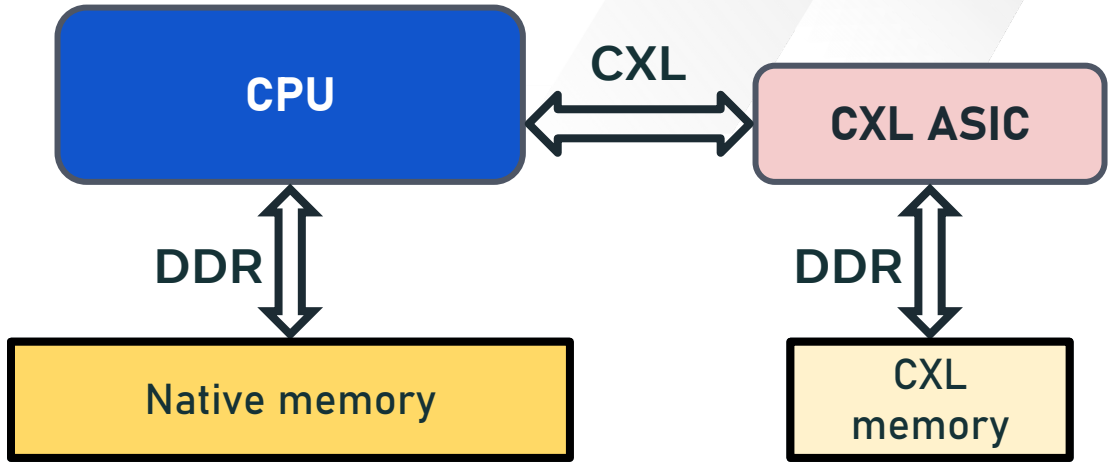
Tiered Memory

- Capacity and bandwidth expansion
- Numa Domains
- Roundtrip latency to CXL Numa ≤ socket-to-socket latency
- Exposed to the HV, Guest OS, Apps
- Based on ACPI objects CDAT– SRAT, SLIT, HMAT
- Affinity/NUMA info describes characteristics
- QoS required for balanced performance
- OS-assisted optimization of the memory subsystem
- Several software and hardware techniques under development for managing hot/cold pages



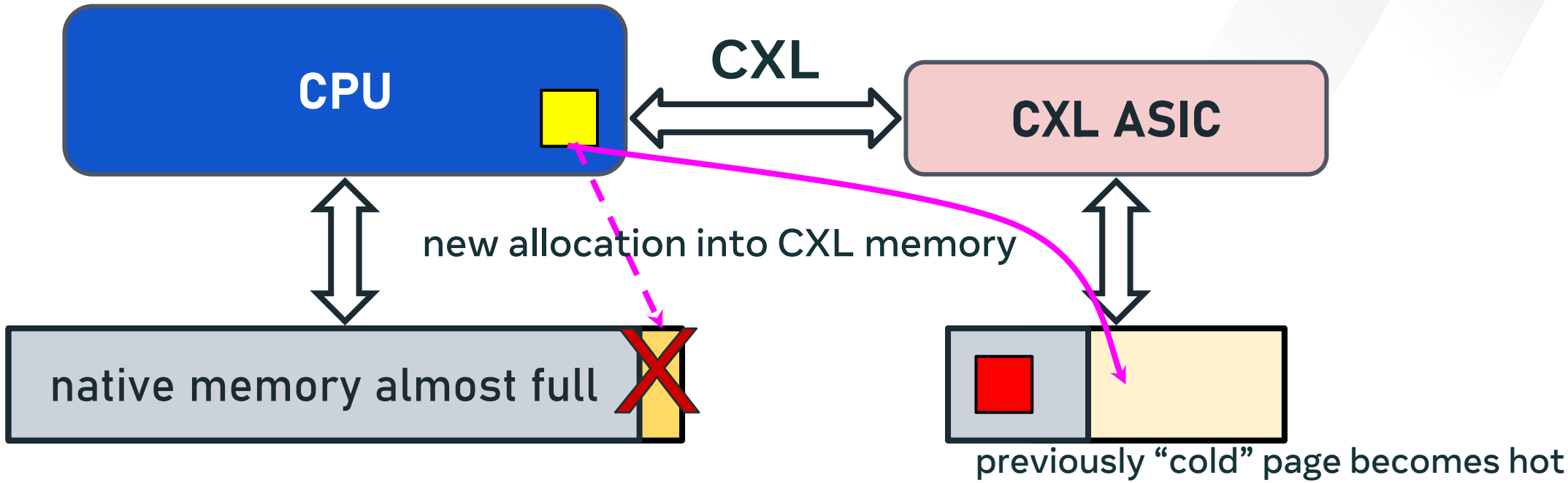
Transparent Tiered memory





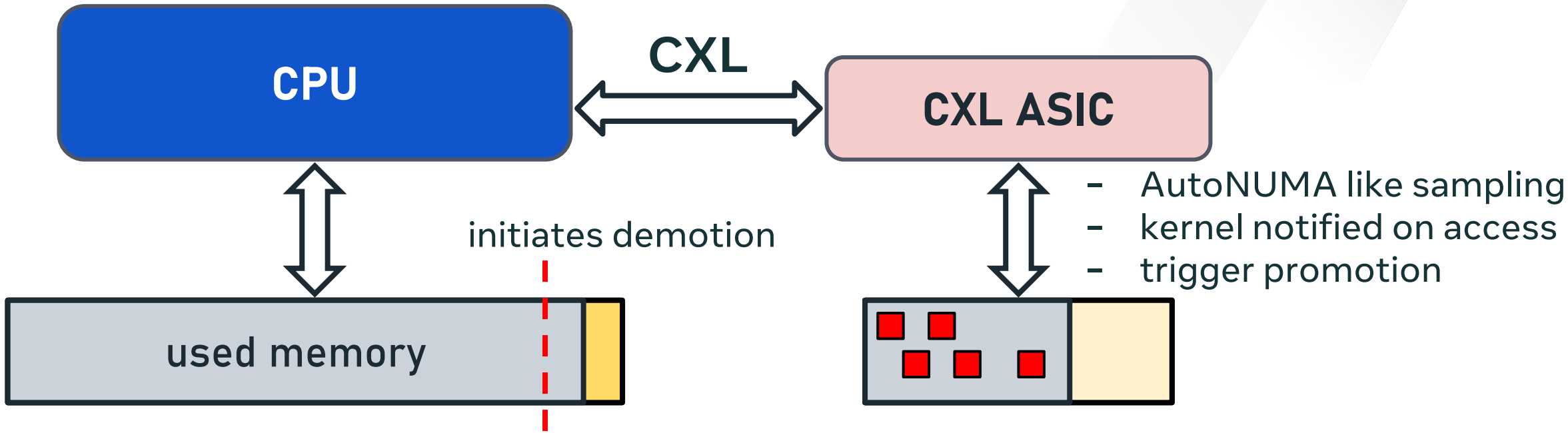
- CXL latency good but nearly 2x native
- Need smart page placement to keep performance intact
  - “hot” memory pages in native memory
  - “cold” memory pages in CXL memory

Tiering – issues with existing software



- With current NUMA schemes hot pages can get stuck in CXL memory
  - Hurts application performance

Example: Tiering with TPP



- Cold page detection using existing kernel mechanism
- Demotion - moving cold pages to CXL memory proactively to maintain headroom in native memory
- Promotion - moving hot pages to native memory
  - Additional optimization to avoid ping-ponging between hot/cold

Deployment Challenges

RAS – CXL ERROR HANDLING

Error Type	Error Reporting	Error Handling
Poison	Reported synchronously with the poisoned data and traverses to the requestor	Consistent with handling poison from direct attached memory. Host Implementation mechanisms (Ex. Machine Check)
Viral	Reported via CXL Link	Consistent with handling fatal error from direct attached memory (Host Implementation specific mechanisms)
CXL Protocol Errors	Reported via PCIe AER. AER "Internal Error" (UIE/CIE) indicates information is also logged in CXL RAS capability structure.	[New] PCIe RCEC collects the error information and signals the FW/OS. Handling is analogous to PCIe Root Port handling.
CXL component events	Reported via CXL Device/Component Command Interface (mailbox, etc.) using CXL-defined record formats.	[New] Device may signal FW using VDM or OS using MSI/MSI-X. FW/OS issues mailbox commands to device to gather records.

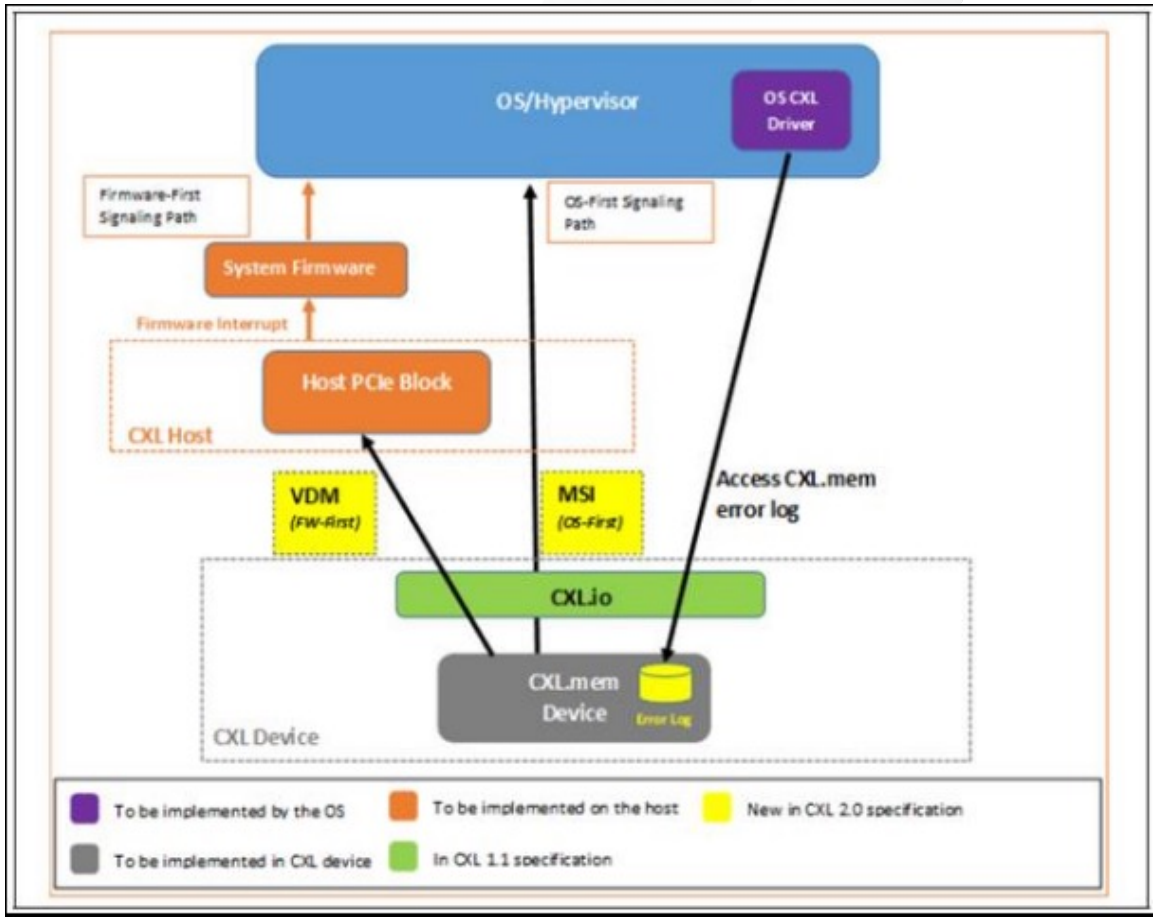
CXL Protocol Errors and Component events requires new software development.

RAS – CXL PROTOCOL ERROR HANDLING

- FW FIRST
  - Platform firmware follows the AER path to decode the errors.
  - Platform firmware collects CXL RAS capability information as part AER handling.
  - Platform firmware will use the UEFI “CXL Protocol Error Section” CPER format to represent data.
  - Platform firmware may signal OS through existing ACPI methods.
- OS FIRST
  - OS must handle AER events through existing methods.
  - If AER status indicates an “Internal Error” (UIE/CIE), and the source is a RCEC, then the OS must inspect the AER and CXL RAS structures of all associated DPs.
  - If AER status indicates an “Internal Error” (UIE/CIE), and the source is a EndPoint, then the OS must inspect the CXL RAS structure of the associated UP.

RAS – CXL COMPONENT EVENT HANDLING

- Boot-time tasks
  - OS must request and be granted “Memory Error reporting” control through _OSC.
  - OS may load a “CXL Memory Device” driver to manage the Device/Component Command Interface.
  - OS may load a “CXL Event” service driver. ▪
    - Leverage “CXL Memory Device” driver to interact with mailbox, etc.
    - Enable polling timers and MSI/MSI-X interrupts for Events
- Run-time tasks
  - Event driver handlers will execute mailbox commands to get and clear records.
  - Event driver will parse records and report to user.
  - Event driver may call other OS facilities to act on error information



OS CXL DRIVER DEVELOPMENT REQUIRED

QoS Telemetry

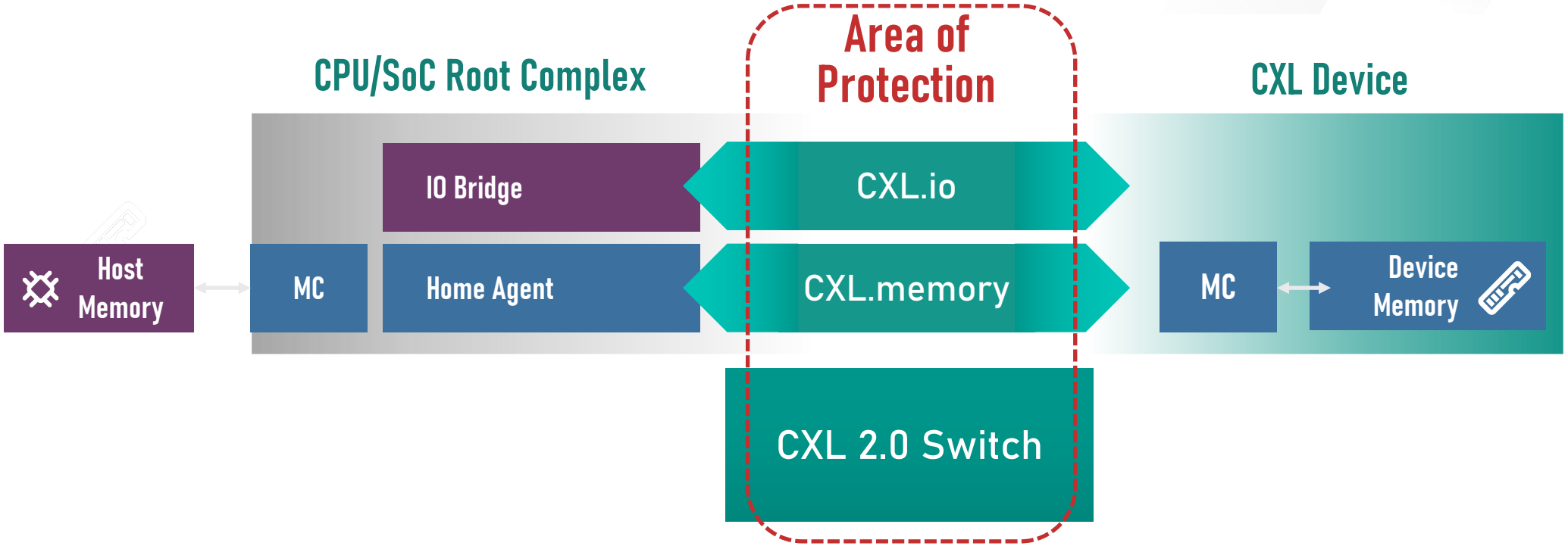
- Access throttling (rate metering) mechanisms required for balanced performance across Memory Tiers
  - Mechanisms are Host implementation specific
- QoS Telemetry mechanisms enable CXL devices to provide timely device load notifications in responses.
  - Immediate & On-going
  - Covers Device Internal Loading, Egress Port Backpressure, and Temporary throughput reduction
- Enables host to optimize rate metering mechanisms
- Multiple QoS Classes supported, covers MLD
- QoS ever important as use models expand

Lifecycle Management

- CXL Devices encounter several events/telemetry that need to be managed
  - Firmware events
  - RAS events
  - Thermal events
  - Device Health
- Run-Time management of CXL Devices requires a mechanism
  - Notify events
  - Read Event Logs
  - Commands to manage/repair Device
  - Read info for Diagnostics
  - Get Device Capabilities information
- In-band & Out-of-band mechanisms to meet manageability requirements
- Ecosystem Software Development required for Lifecycle Management

Security

CXL 2.0 provides Integrity and Data Encryption of traffic across all entities (Root Complex, Switch, Device)

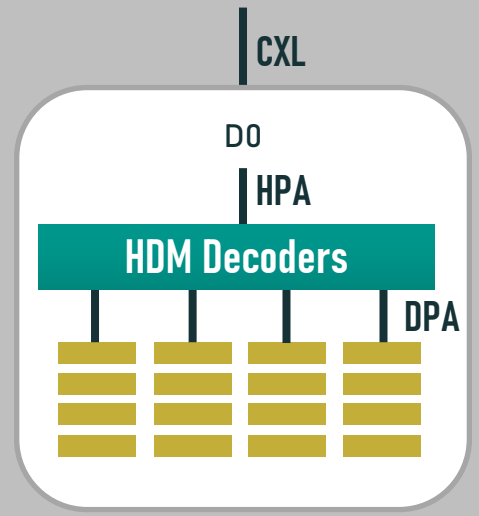


- Link IDE provides link level encryption and integrity – defined.
- Proposals under development to meet emerging security and confidential compute requirements

CXL MEMORY DEVICE TYPES AND POOLING

CXL SLD Memory Device

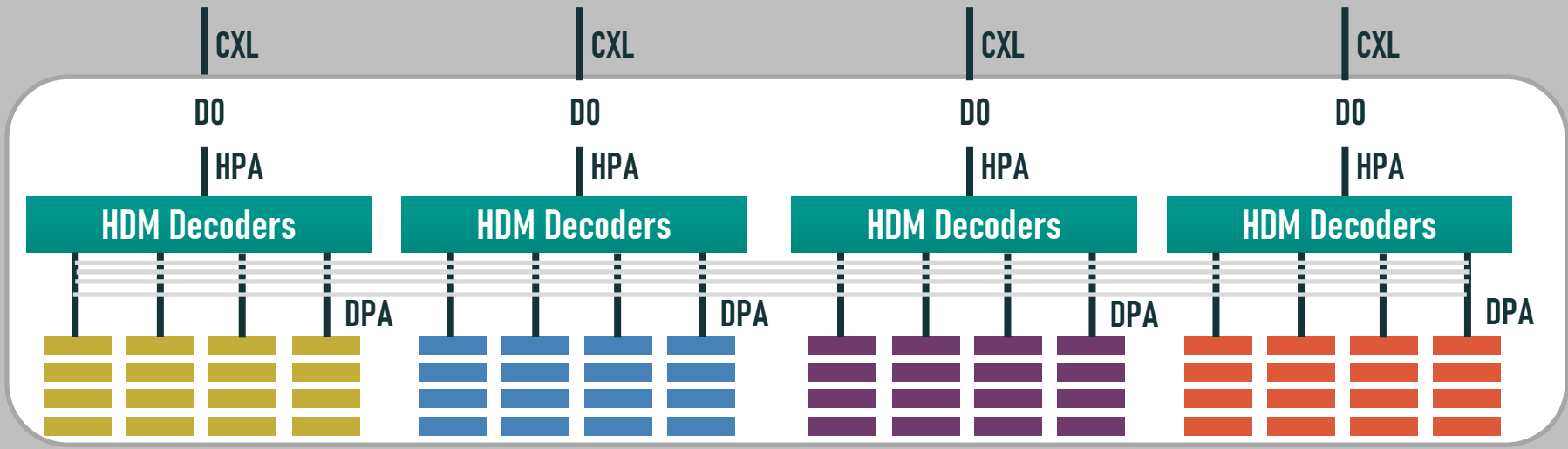
- Support one or more PCIe Endpoint Functions
  - Type 0 header in PCI Configuration space
- Primary function (device number 0, function number 0) must carry one instance of CXL DVSEC ID 0 with Revision 1 or greater.
- Non-CXL Function Map DVSEC to advertise Non-CXL functions
- Must support operating in CXL 1.1 mode
  - PCIe Endpoint → RCIEP
- Type 3 device Component Register Block includes HDM Decoder registers
- Connected to a Single Virtual Hierarchy



HPA – Host Physical Address
DPA – Device Physical Address

CXL Multi-Ported Memory Device

Pooled Memory Device



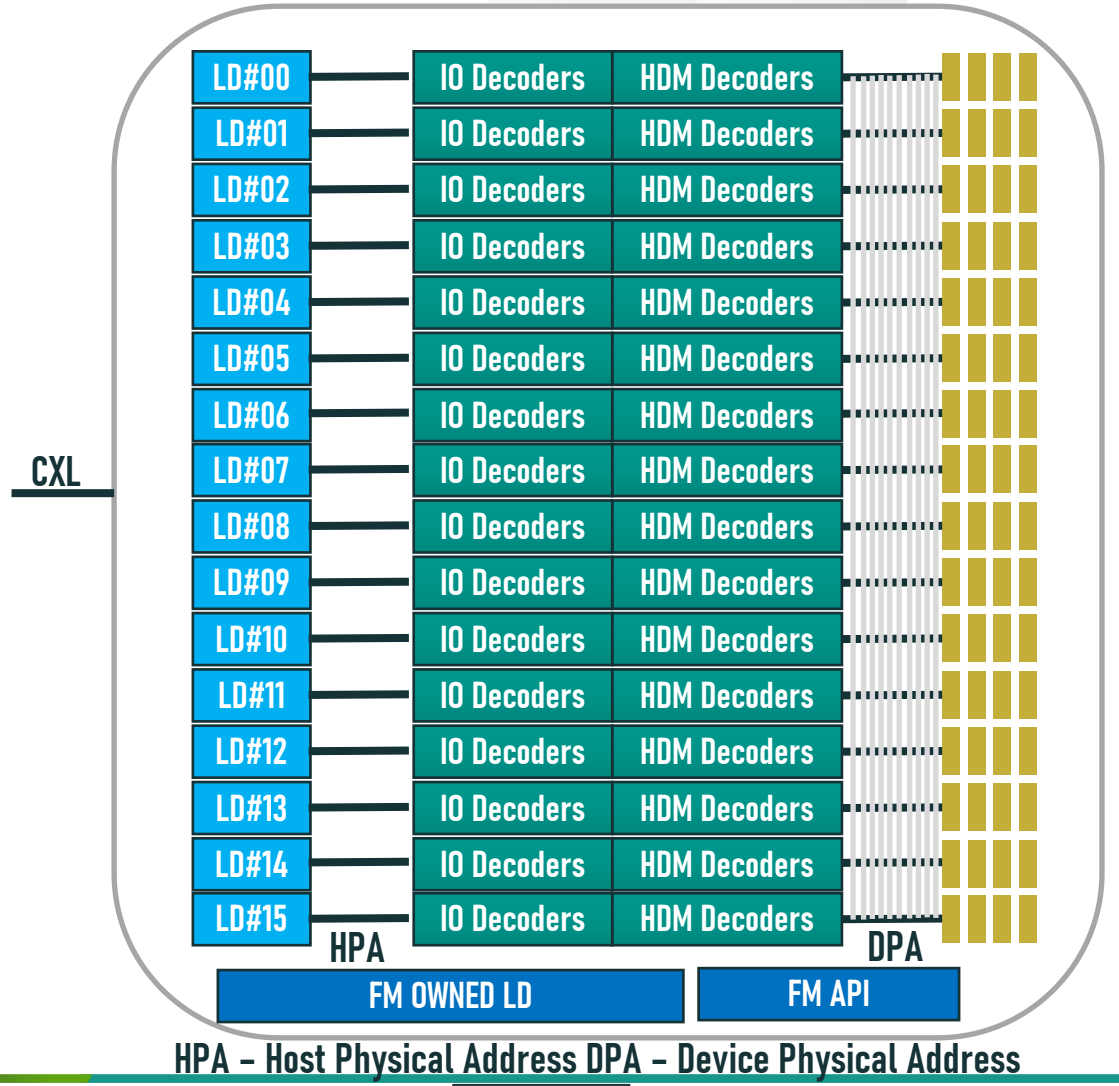
HPA – Host Physical Address
DPA – Device Physical Address

- Represents an SLD behind each CXL Port
- Device Vendor-Specific mechanisms to configure resources per SLD
- Example – 4 Ported CXL Memory Device with equal allocation of pooled resources across ports

CXL Multi-Logical Memory Device

Pooled Memory Device

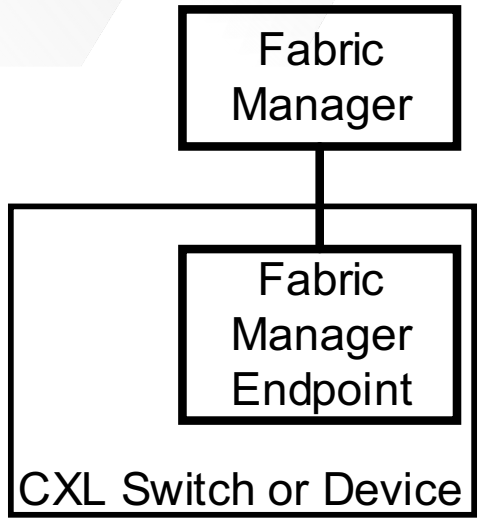
- A Pooled Type 3 device can partition its resources into Logical Devices (LD)
  - Up to 16 LDs (Type 3 only) AND
  - One Fabric Manager (FM) owned LD
- Each LD
  - Appears as Type 3 SLD device
  - Identified by LD-ID
  - FM binds each LD to a Virtual Hierarchy
- FM Owned LD
  - Accessible by FM only by using LD-ID of 0xFFFFh
  - Manage Link and Device
  - Memory resources are not assigned to LD owned LD
  - Error messages generated by LD are routed to FM
  - Does not participate in GPF Flows
- MLD Link
  - MLD Link Discovery & Link Operation configured via Alternate Protocol Negotiation



HPA – Host Physical Address DPA – Device Physical Address

Fabric Manager Endpoint and API

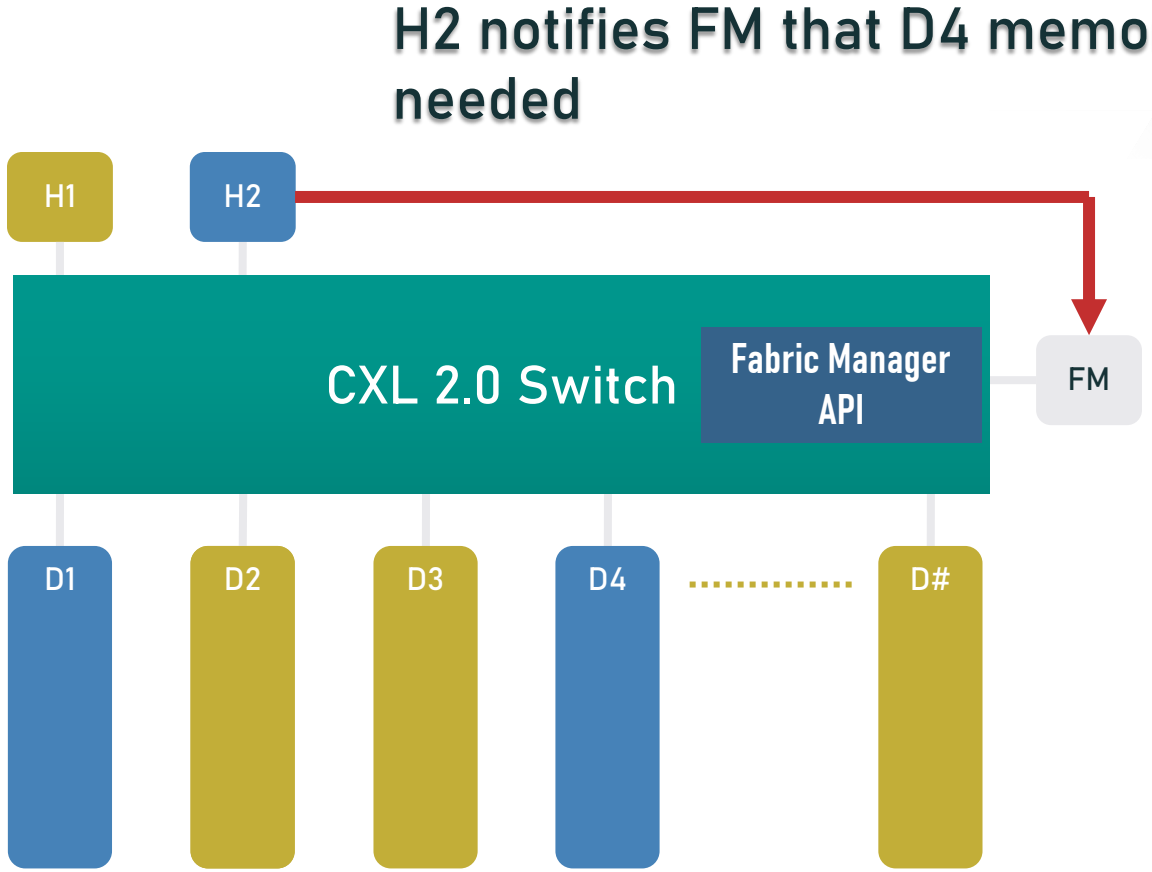
- Fabric Manager is a control entity that manages the CXL 2.0 Switch and the Memory Controller
  - FM can be an external BMC, a Host, or Firmware internal to the Switch
- FM Endpoint is a required feature for any switch that supports MLD ports or that supports dynamic SLD port binding
- FM API is the standardized interface for the FM to communicate with devices
- FM API uses an MCTP interface between Fabric Manager and devices
  - MCTP physical interface is switch vendor specific but could be PCIe, CXL.io VDM, SMBus, Ethernet, UART, USB, internal, …
- In general there are no real-time response requirements for the Fabric Manager so it needn't be performant



Fabric Manager

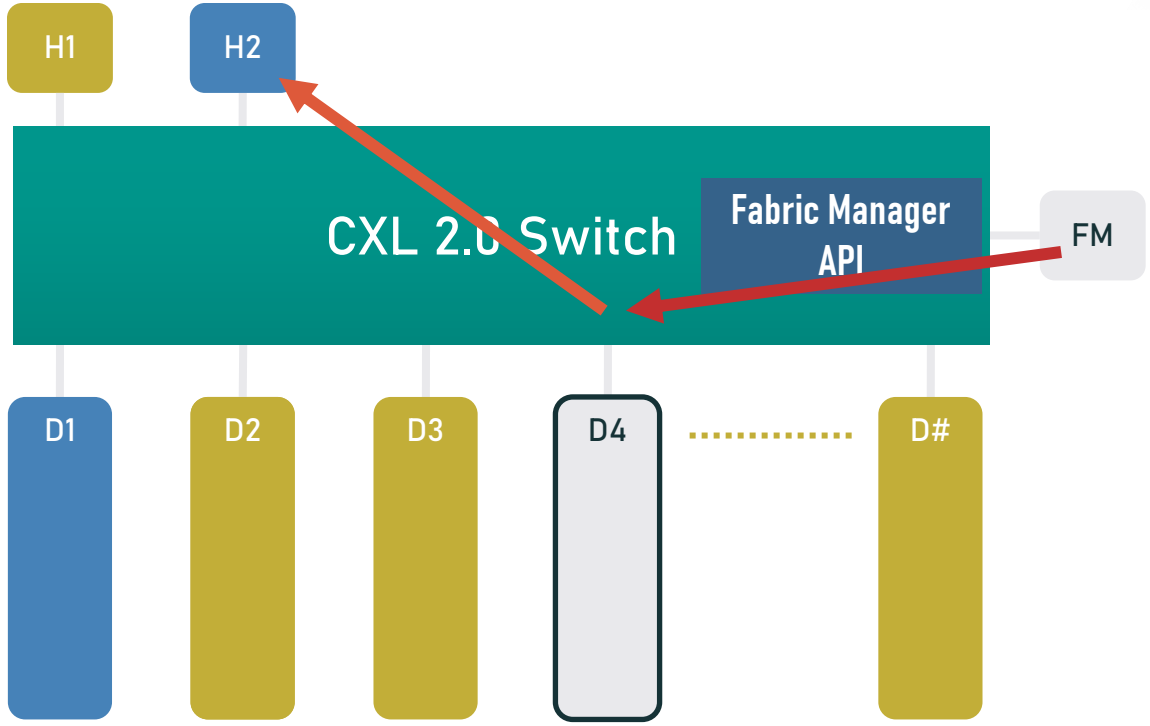
- Fabric Manager plays a critical role in CXL for systems supporting Memory Pooling
- The Fabric Manager enables dynamic system changes supporting Memory disaggregation
- Some examples:
  - Managing all devices that support traffic from multiple Hosts including:
    - Downstream ports connected to MLD ports
    - FM-owned Logical Device within an MLD component
    - Unbinding and rebinding of Logical Devices within an MLD between Hosts
  - Unbinding and rebinding of an SLD
  - Re-allocation of memory within an MLD
  - Re-allocation of memory within a multi-port SLD

Memory Pooling with Single Logical Devices



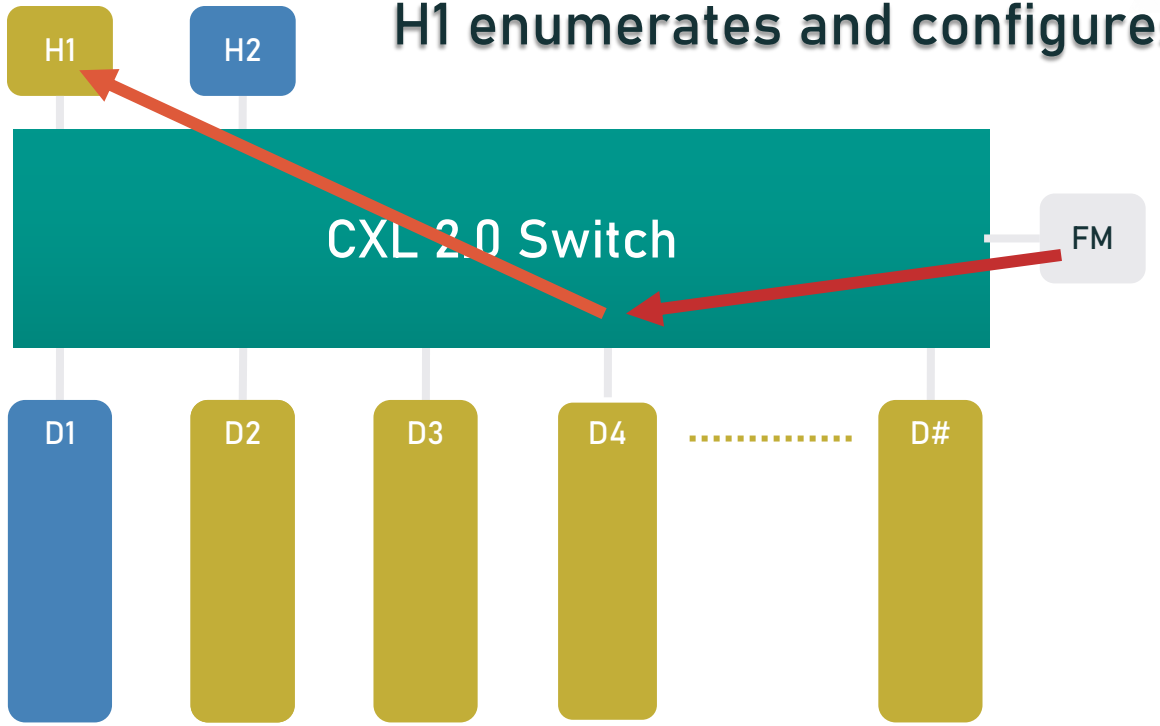
Memory Pooling with Single Logical Devices

FM tells switch to UNBIND D4
Switch notifies H2 of the managed hot remove

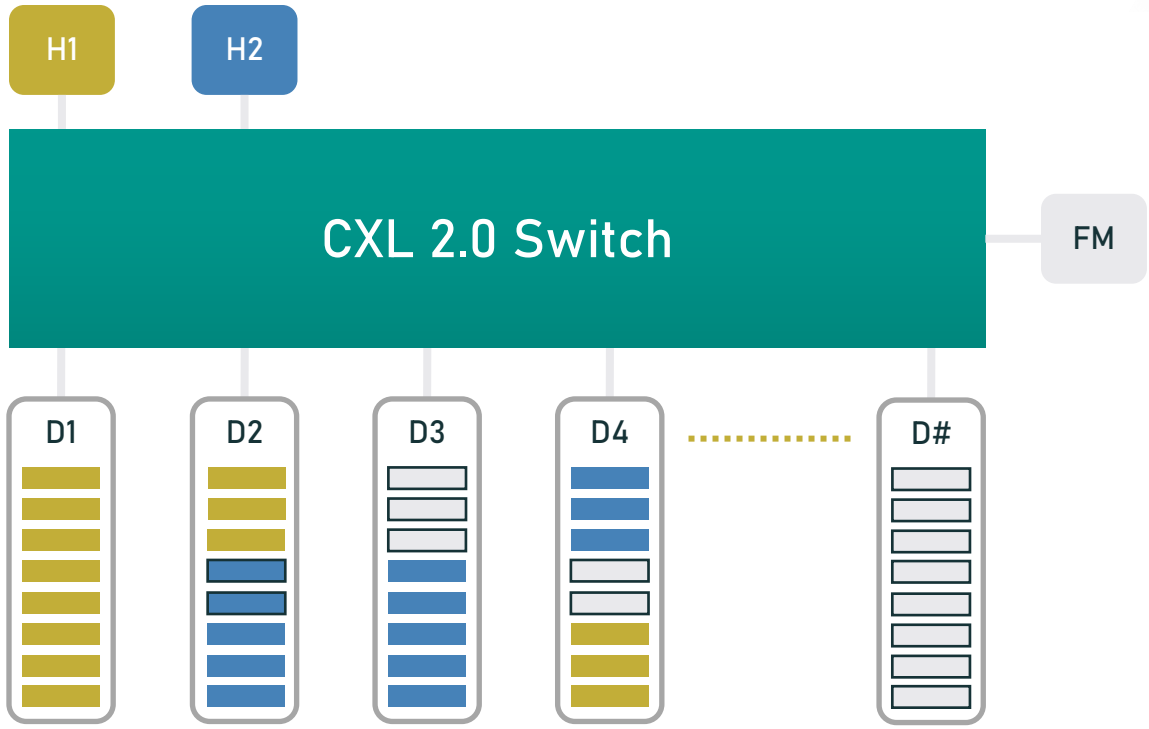


Memory Pooling with Single Logical Devices

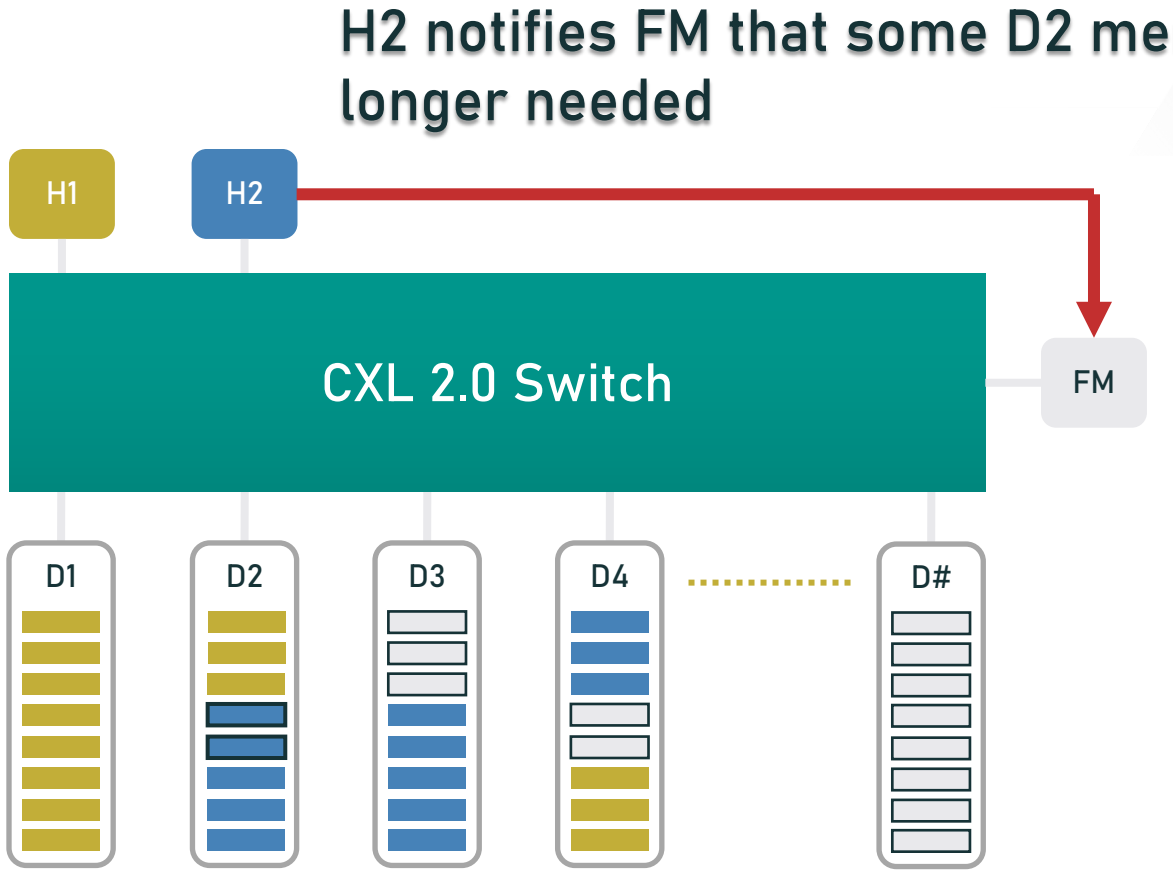
FM tells switch to BIND D4 to H1
Switch notifies H1 using managed hot add
H1 enumerates and configures accesses to D4



Memory Pooling with Multi-Logical

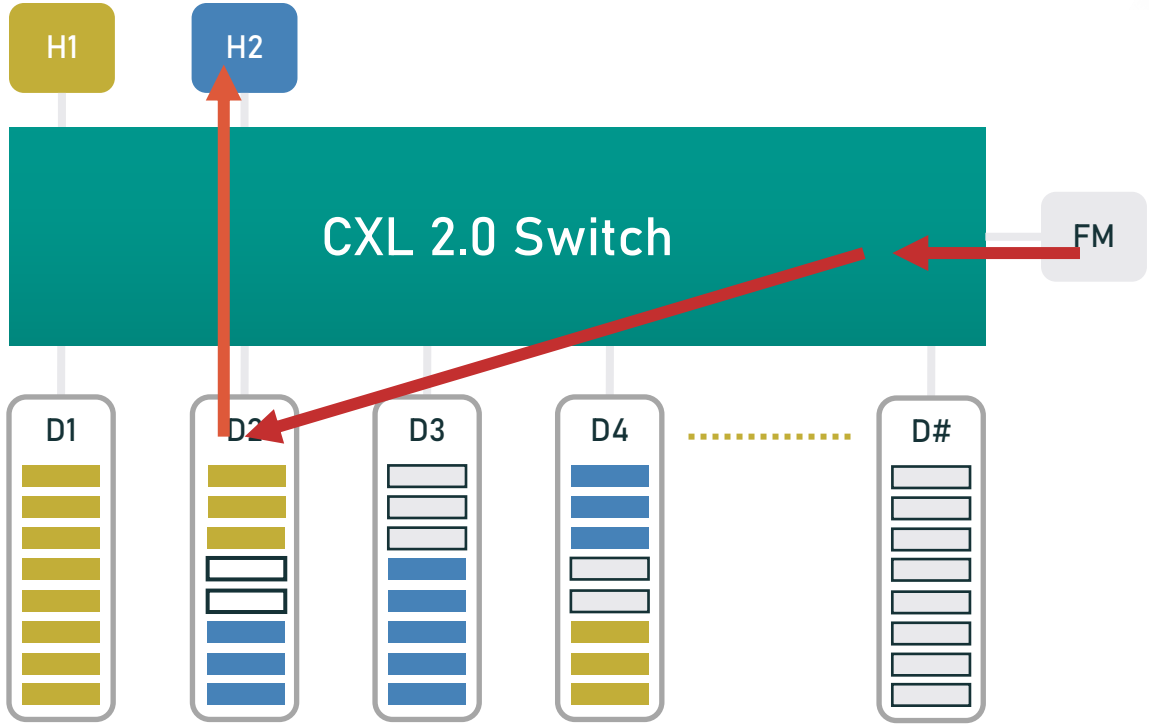


Memory Pooling with Multi-Logical Devices



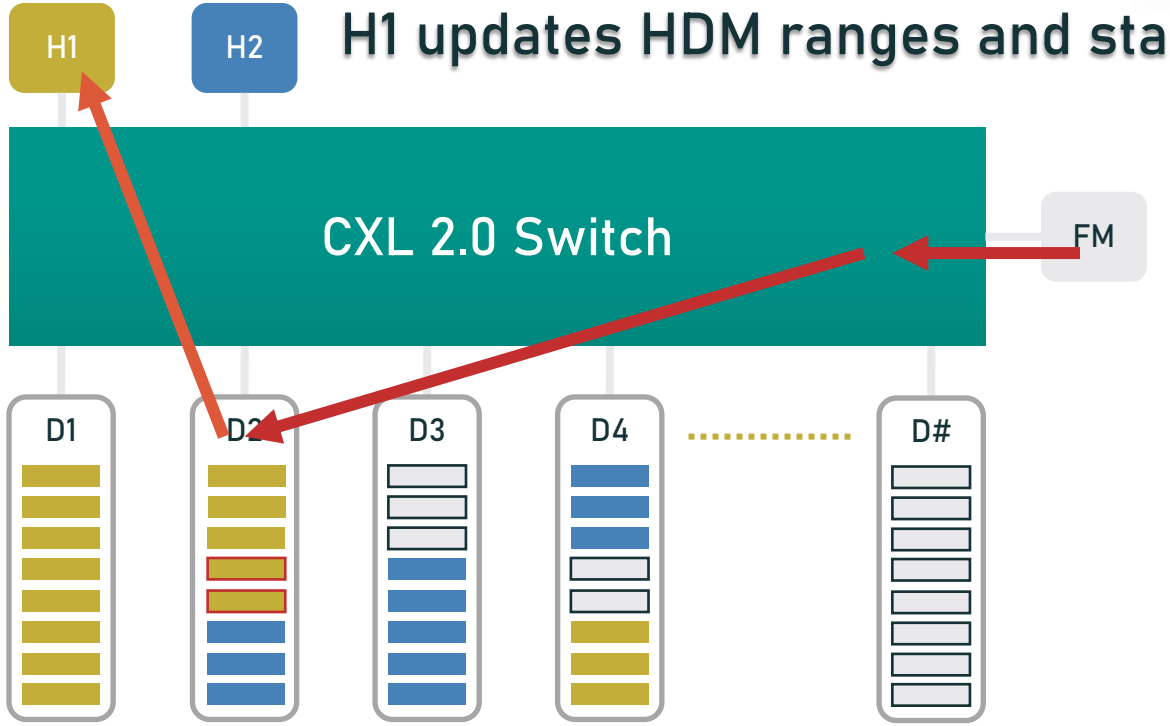
Memory Pooling with Multi-Logical

FM tells D2 to de-allocate some blue memory
D2 notifies H2



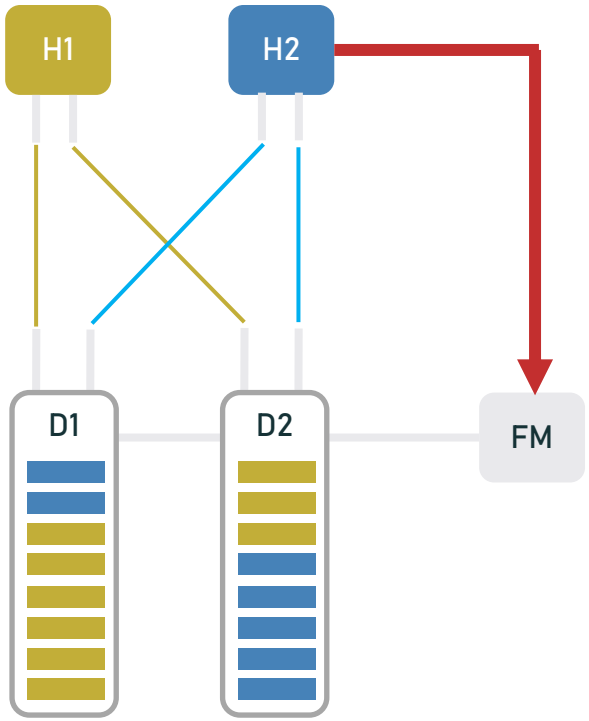
Memory Pooling with Multi-Logical

FM tells D2 to allocate some yellow memory
D2 notifies H1
H1 updates HDM ranges and starts using memory



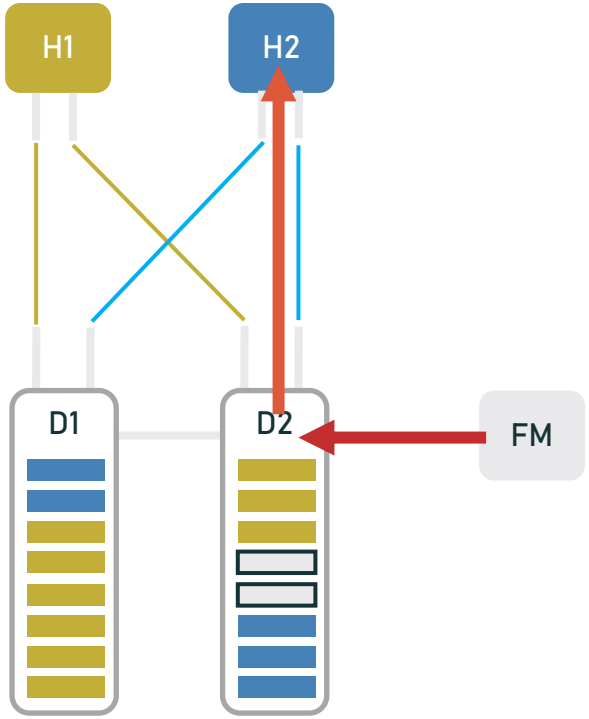
Memory Pooling without a Switch

H2 notifies FM that some D2 memory is no longer needed



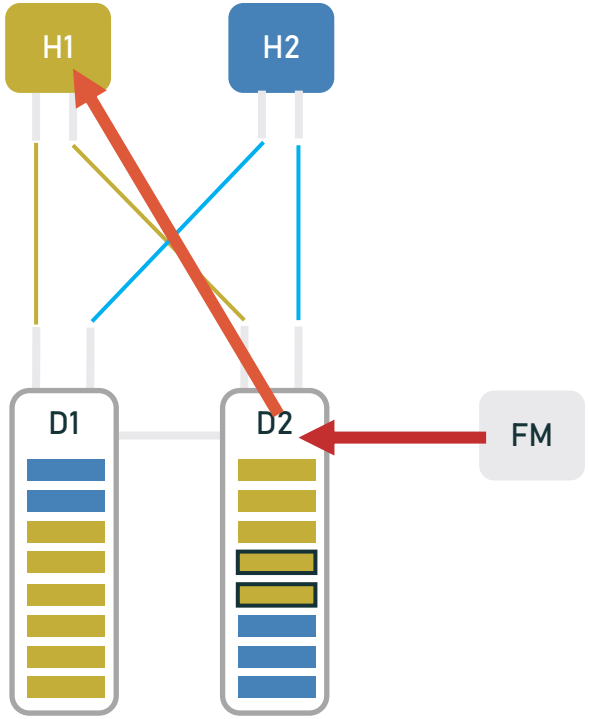
Memory Pooling without a Switch

FM tells D2 to de-allocate some blue memory
D2 notifies H2



Memory Pooling without a Switch

FM tells D2 to allocate some yellow memory
D2 notifies H1, H1 updates HDM ranges



Memory Pooling with CXL

- Other FM API features beyond BIND, UNBIND, and SET LD allocations:
  - Switch discovery including capacity, capabilities, and connected devices
  - Event notification such as switch link events and Advanced Error Reporting for FM owned resources
  - Manage MLD QoS parameters

- Benefits of Memory Pooling
  - Effective utilization of memory resources within a system
  - Dynamic Allocation/deallocation of memory resources
  - Total Cost Of Ownership (TCO) savings

Summary

- CXL attached memory use models continue to grow, from direct attached to realizable memory centric computing
- CXL consortium continues to address the challenges to support CXL attached memory
- Large opportunities lie ahead for the ecosystem

- Get involved!