# Emerging Memory Technology on CXL™

WSOS 2023

**Andy Rudoff, Intel Labs**

# Emerging Memory Technology on CXL™

WSOS 2023

**Andy Rudoff, Intel Labs, CXL Consortium Member**

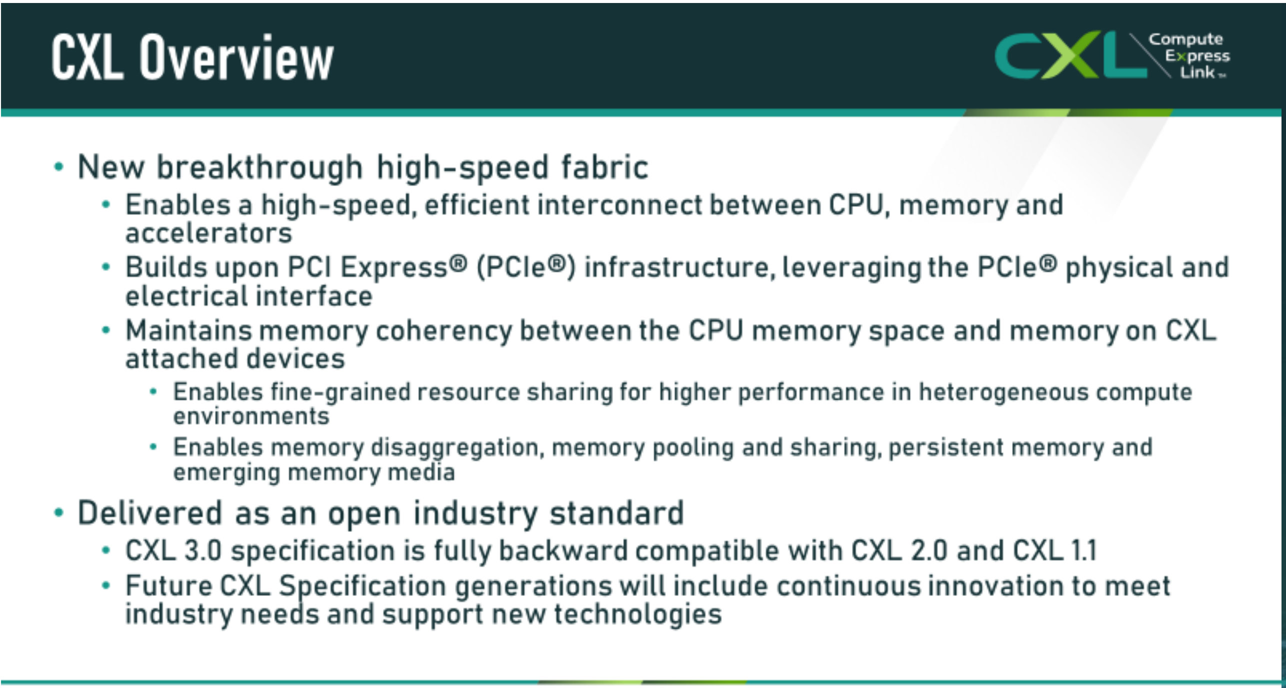
**Other Than Media**

# Emerging Memory Technology on CXL™

WSOS 2023

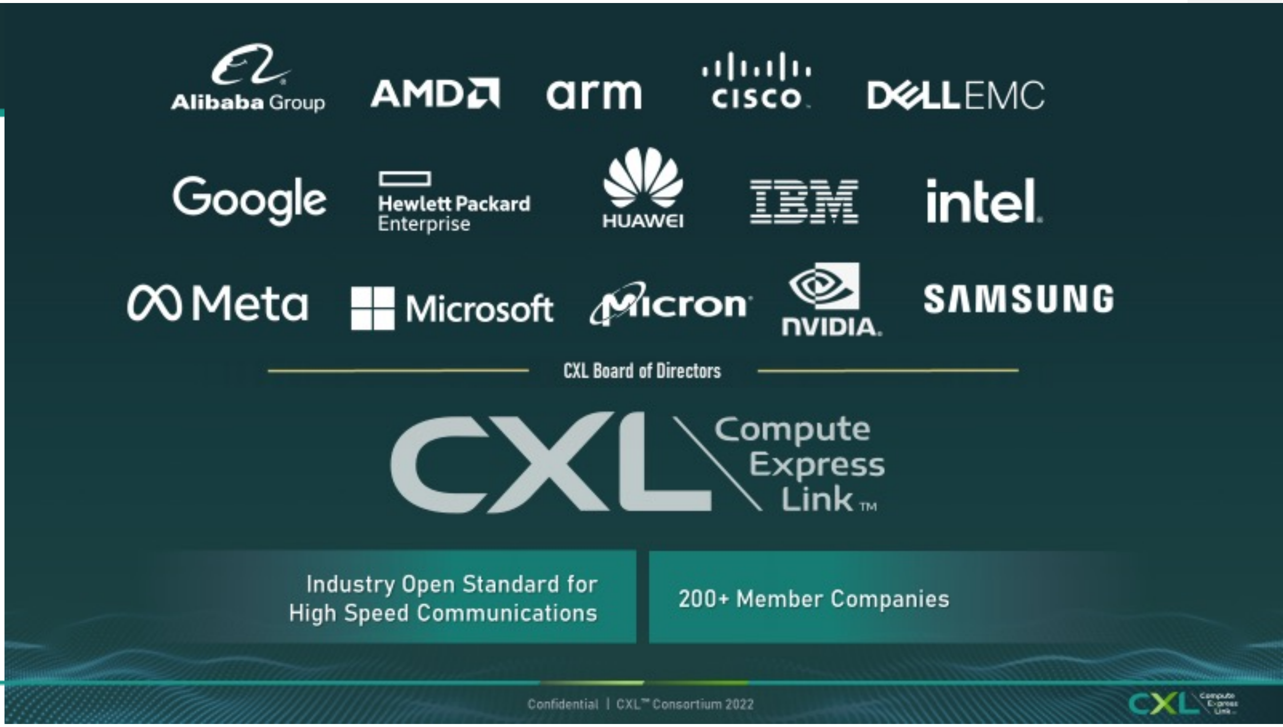
**Andy Rudoff, Intel Labs, CXL Consortium Member, Product-Oriented Software Architect**

# Compute Express Link (CXL)



## CXL Overview

- New breakthrough high-speed fabric
  - Enables a high-speed, efficient interconnect between CPU, memory and accelerators
  - Builds upon PCI Express® (PCIe®) infrastructure, leveraging the PCIe® physical and electrical interface
  - Maintains memory coherency between the CPU memory space and memory on CXL attached devices
    - Enables fine-grained resource sharing for higher performance in heterogeneous compute environments
    - Enables memory disaggregation, memory pooling and sharing, persistent memory and emerging memory media
- Delivered as an open industry standard
  - CXL 3.0 specification is fully backward compatible with CXL 2.0 and CXL 1.1
  - Future CXL Specification generations will include continuous innovation to meet industry needs and support new technologies

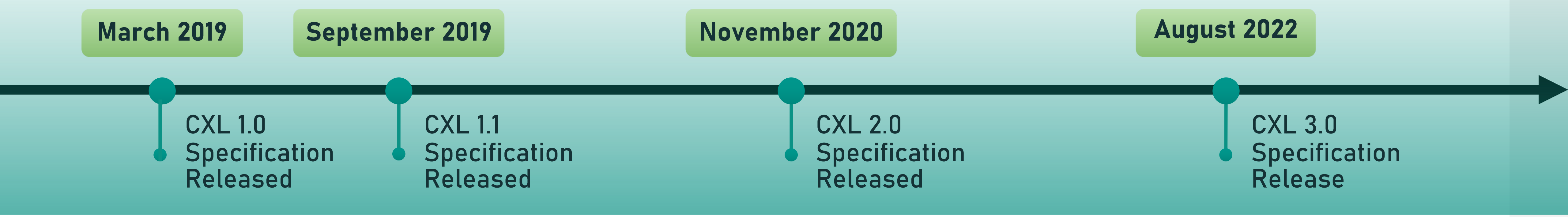


CXL Board of Directors

Industry Open Standard for High Speed Communications

200+ Member Companies

Confidential | CXL™ Consortium 2022



March 2019 — CXL 1.0 Specification Released

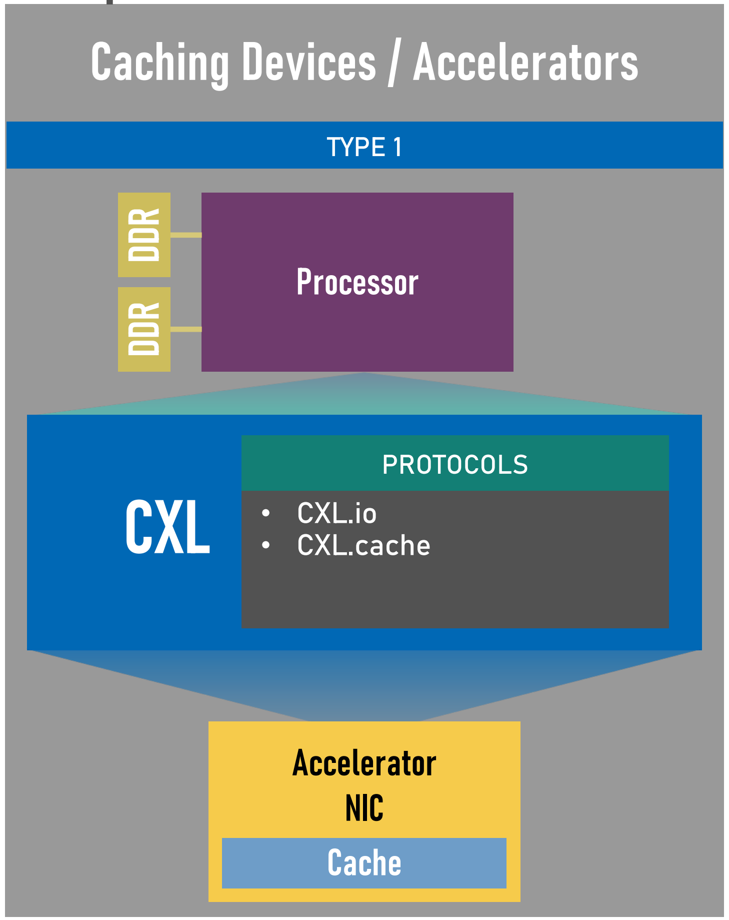
September 2019 — CXL 1.1 Specification Released

November 2020 — CXL 2.0 Specification Released

August 2022 — CXL 3.0 Specification Release

# Representative CXL Usages

## Caching Devices / Accelerators



TYPE 1

DDR

DDR

Processor

CXL

PROTOCOLS

- CXL.io
- CXL.cache

Accelerator

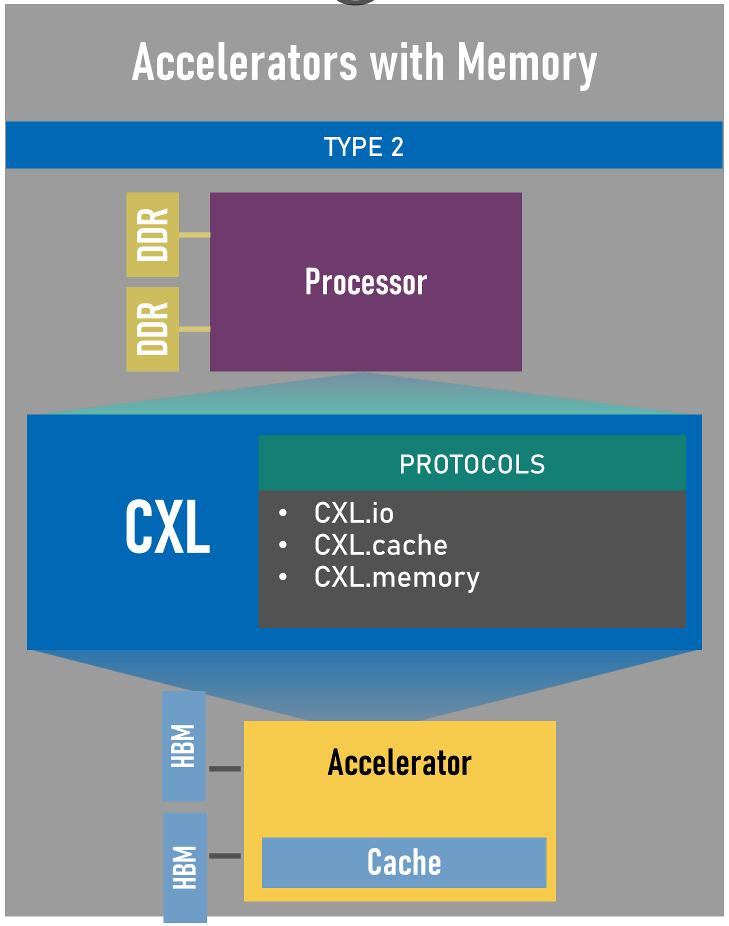
NIC

Cache

USAGES

- PGAS NIC
- NIC atomics

## Accelerators with Memory



TYPE 2

DDR

DDR

Processor

CXL

PROTOCOLS

- CXL.io
- CXL.cache
- CXL.memory

HBM

HBM

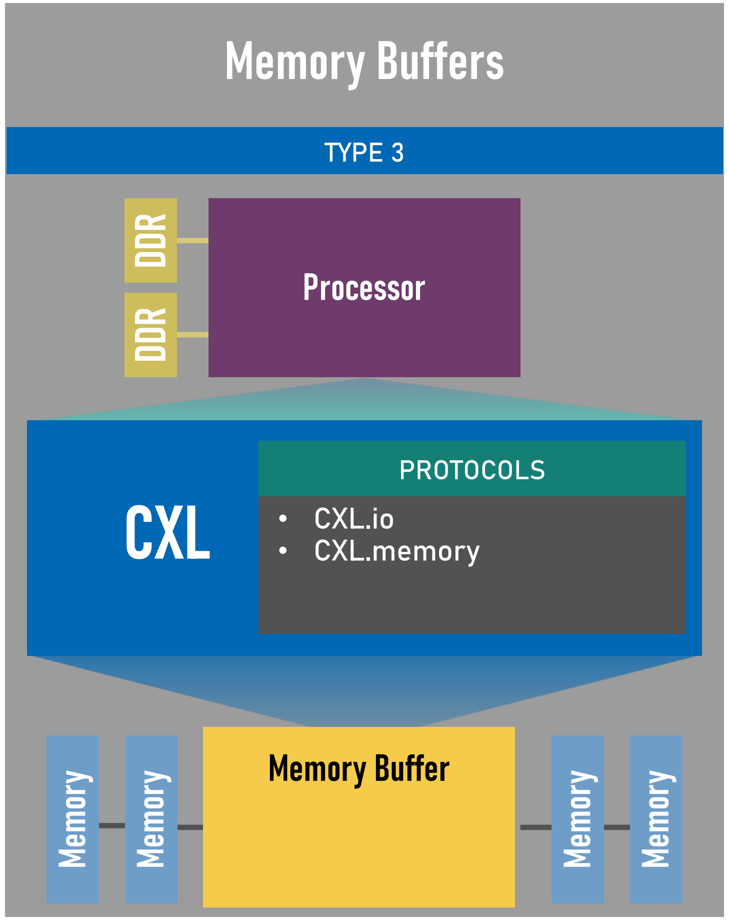
Accelerator

Cache

USAGES

- GPU
- FPGA
- Dense computation

## Memory Buffers



TYPE 3

DDR

DDR

Processor

CXL

PROTOCOLS

- CXL.io
- CXL.memory

Memory

Memory

Memory Buffer

Memory

Memory

USAGES

- Memory BW expansion
- Memory capacity expansion
- Persistent memory

# CXL 3.0 Specification

## Industry trends

- Use cases driving need for higher bandwidth include: high performance accelerators, system memory, SmartNIC and leading edge networking
- CPU efficiency is declining due to reduced memory capacity and bandwidth per core
- Efficient peer-to-peer resource sharing across multiple domains
- Memory bottlenecks due to CPU pin and thermal constraints

## CXL 3.0 introduces…

- Fabric capabilities
  - Multi-headed and fabric attached devices
  - Enhance fabric management
  - Composable disaggregated infrastructure
- Improved capability for better scalability and resource utilization
  - Enhanced memory pooling
  - Multi-level switching
  - New enhanced coherency capabilities
  - Improved software capabilities
- Double the bandwidth
- Zero added latency over CXL 2.0
- Full backward compatibility with CXL 2.0, CXL 1.1, and CXL 1.0

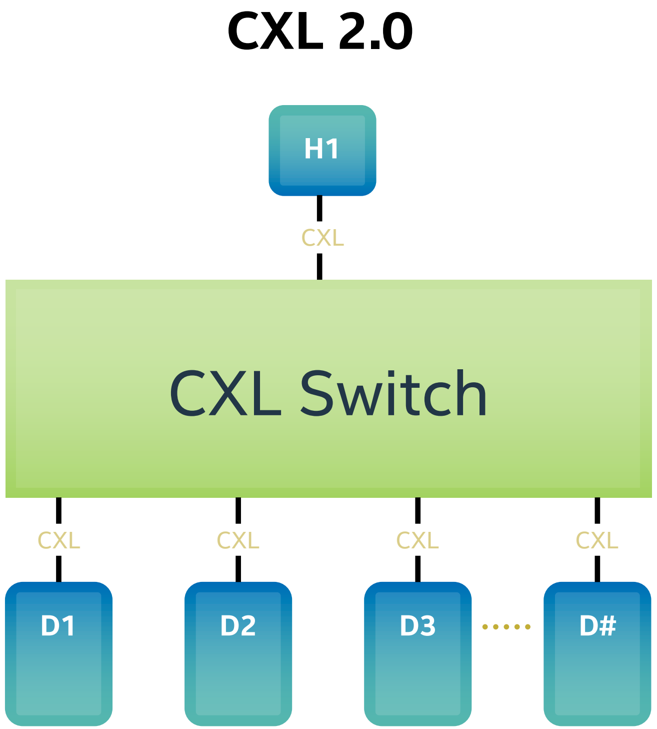
# CXL 3.0 Spec Feature Summary

| Features | CXL 1.0 / 1.1 | CXL 2.0 | CXL 3.0 |
|---|---|---|---|
| Release date | 2019 | 2020 | August 2022 |
| Max link rate | 32GTs | 32GTs | 64GTs |
| Flit 68 byte (up to 32 GTs) | ✓ | ✓ | ✓ |
| Flit 256 byte (up to 64 GTs) |  |  | ✓ |
| Type 1, Type 2 and Type 3 Devices | ✓ | ✓ | ✓ |
| Memory Pooling w/ MLDs |  | ✓ | ✓ |
| Global Persistent Flush |  | ✓ | ✓ |
| CXL IDE |  | ✓ | ✓ |
| Switching (Single-level) |  | ✓ | ✓ |
| Switching (Multi-level) |  |  | ✓ |
| Direct memory access for peer-to-peer |  |  | ✓ |
| Enhanced coherency (256 byte flit) |  |  | ✓ |
| Memory sharing (256 byte flit) |  |  | ✓ |
| Multiple Type 1/Type 2 devices per root port |  |  | ✓ |
| Fabric capabilities (256 byte flit) |  |  | ✓ |

Not supported

✓ Supported

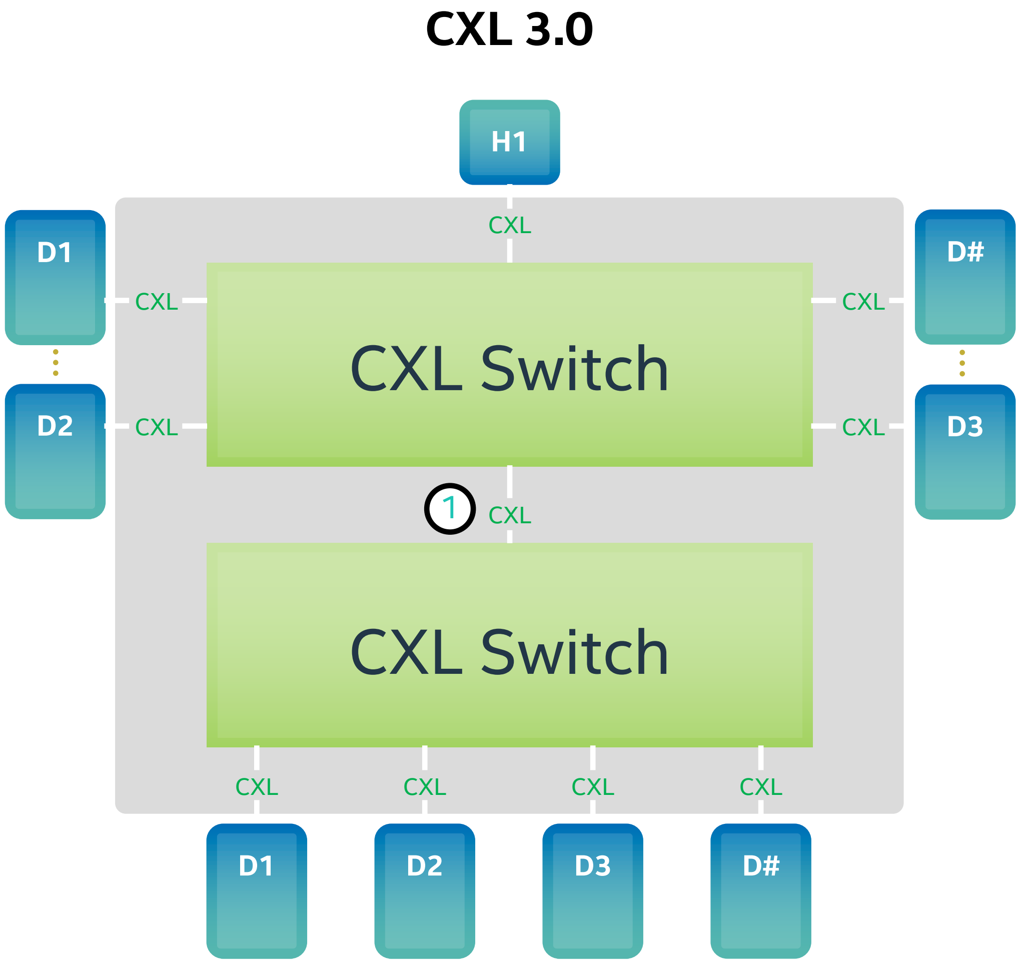
# RECAP: CXL 2.0 Feature Summary
## Switch Capability

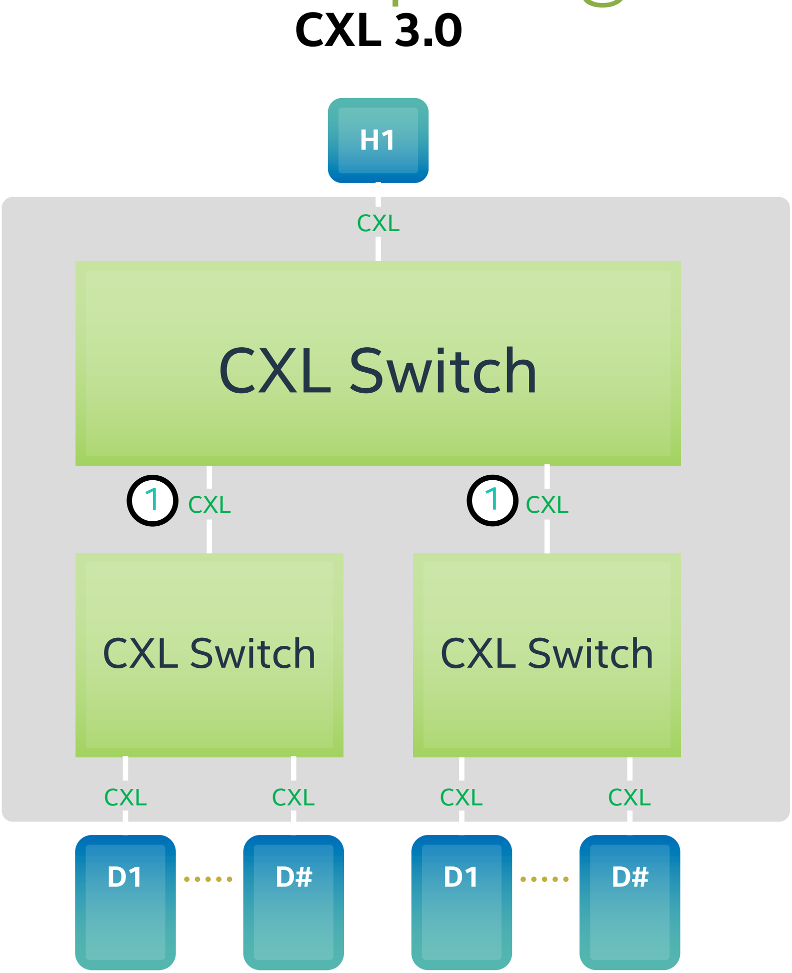


- Supports single-level switching
- Enables memory expansion and resource allocation

# CXL 3.0: Switch Cascade/Fanout

## Supporting vast array of switch topologies

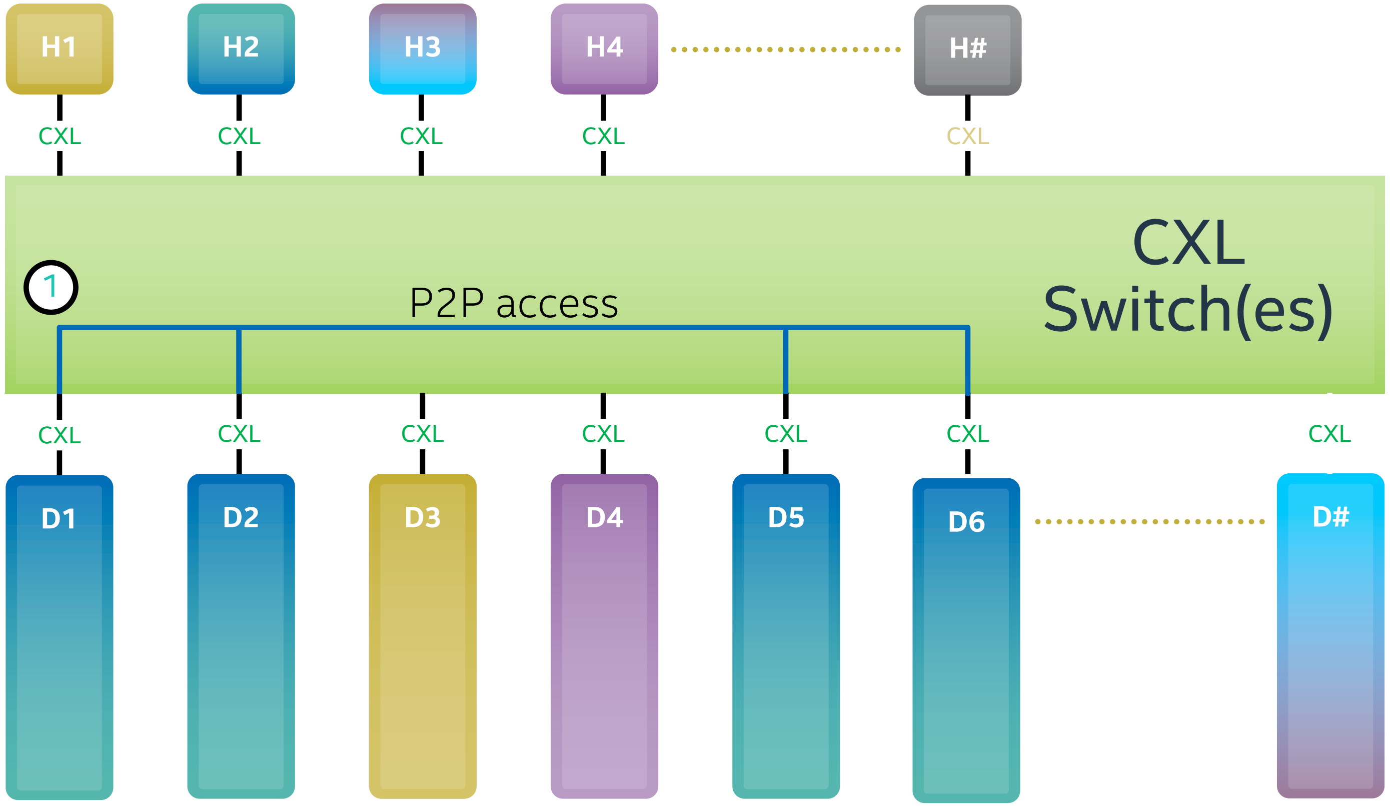




1 Multiple switch levels (aka cascade)

- Supports fanout of all device types

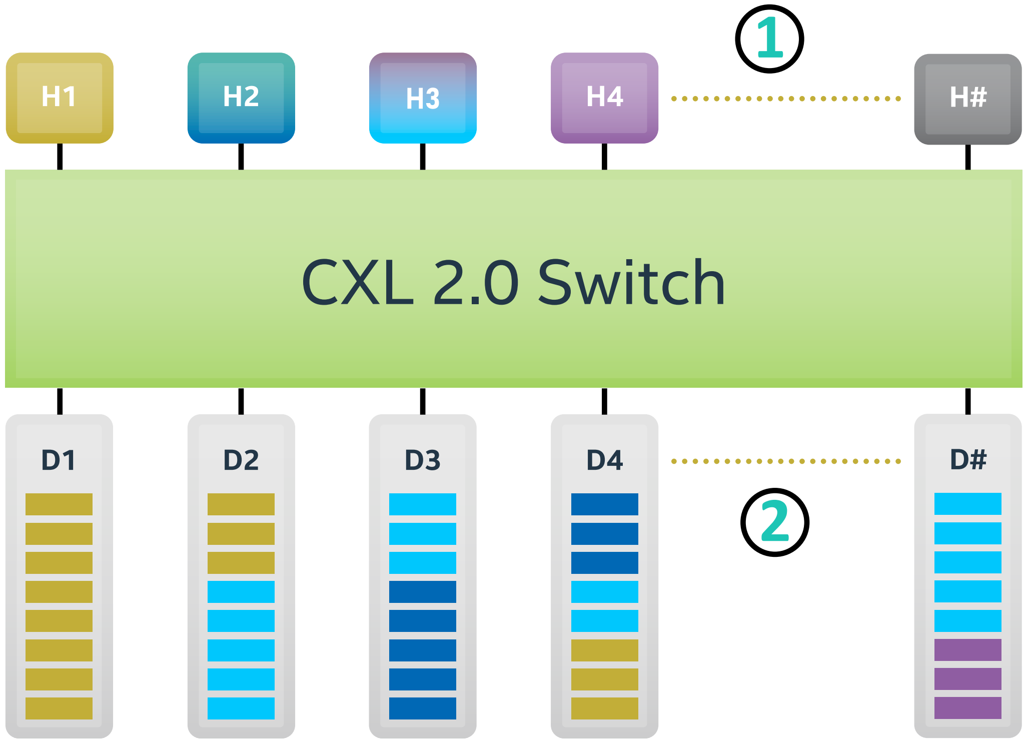
# CXL 3.0: Device to Device Comms



1 CXL 3.0 enables **peer-to-peer communication (P2P)** within a virtual hierarchy of devices

- Virtual hierarchies are associations of devices that maintains a coherency domain

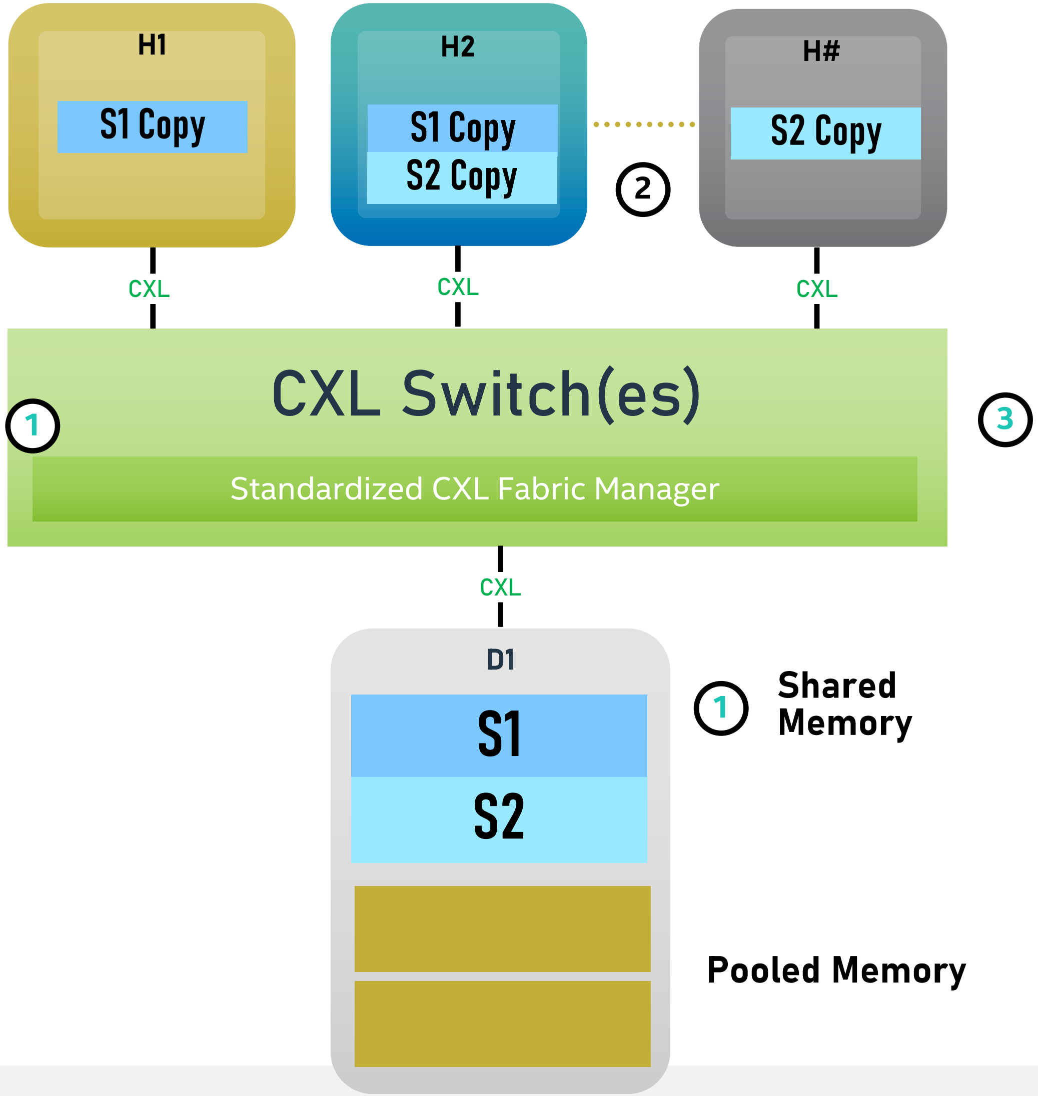
# RECAP: CXL 2.0 Feature Summary
## Memory Pooling



1. Device memory can be allocated across multiple hosts.

2. Multi Logical Devices allow for finer grain memory allocation

# CXL 3.0 Coherent Memory Sharing

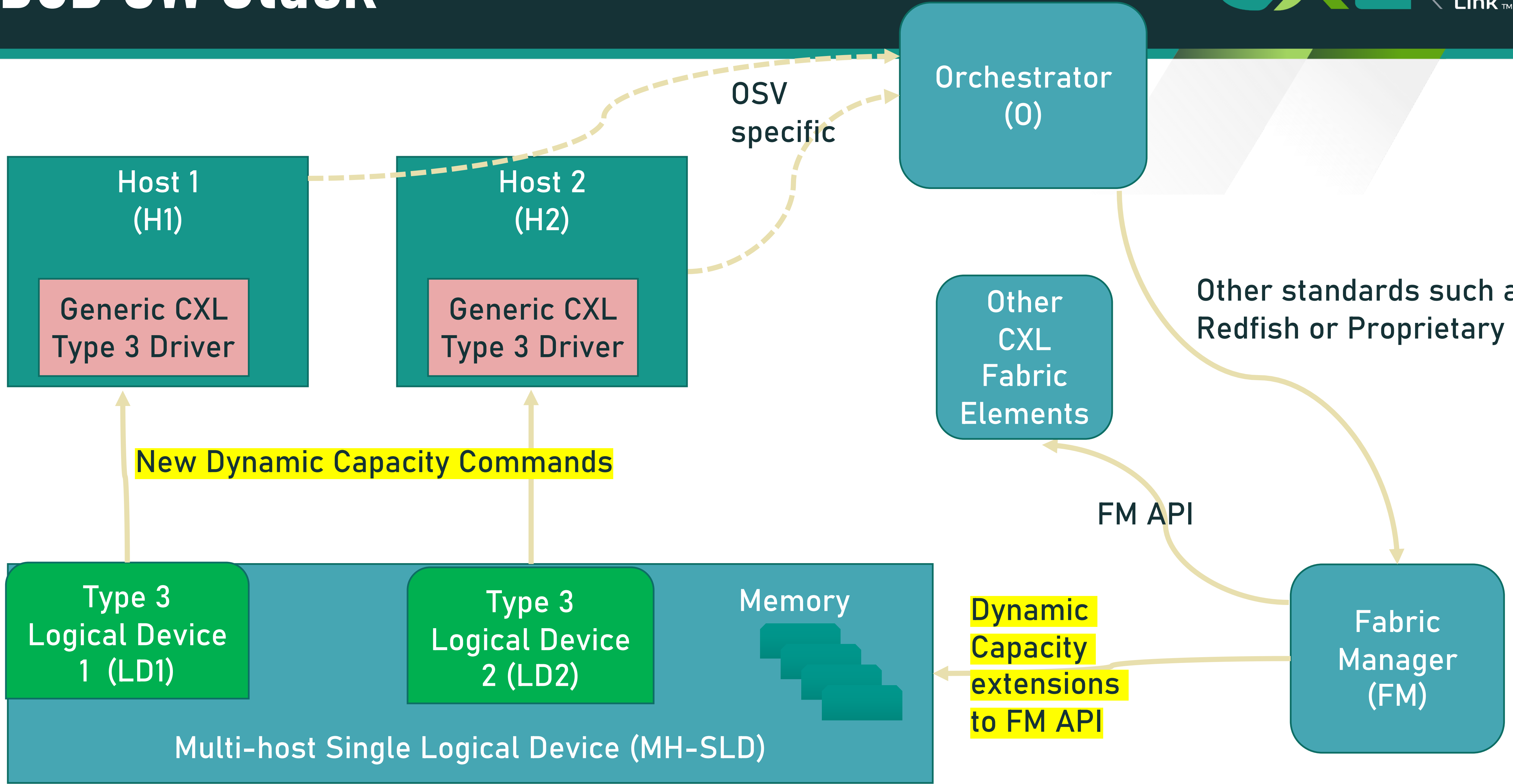


1. Device memory can be shared by all hosts to increase data flow efficiency and improve memory utilization

2. Host can have a coherent copy of the shared region or portions of shared region in host cache

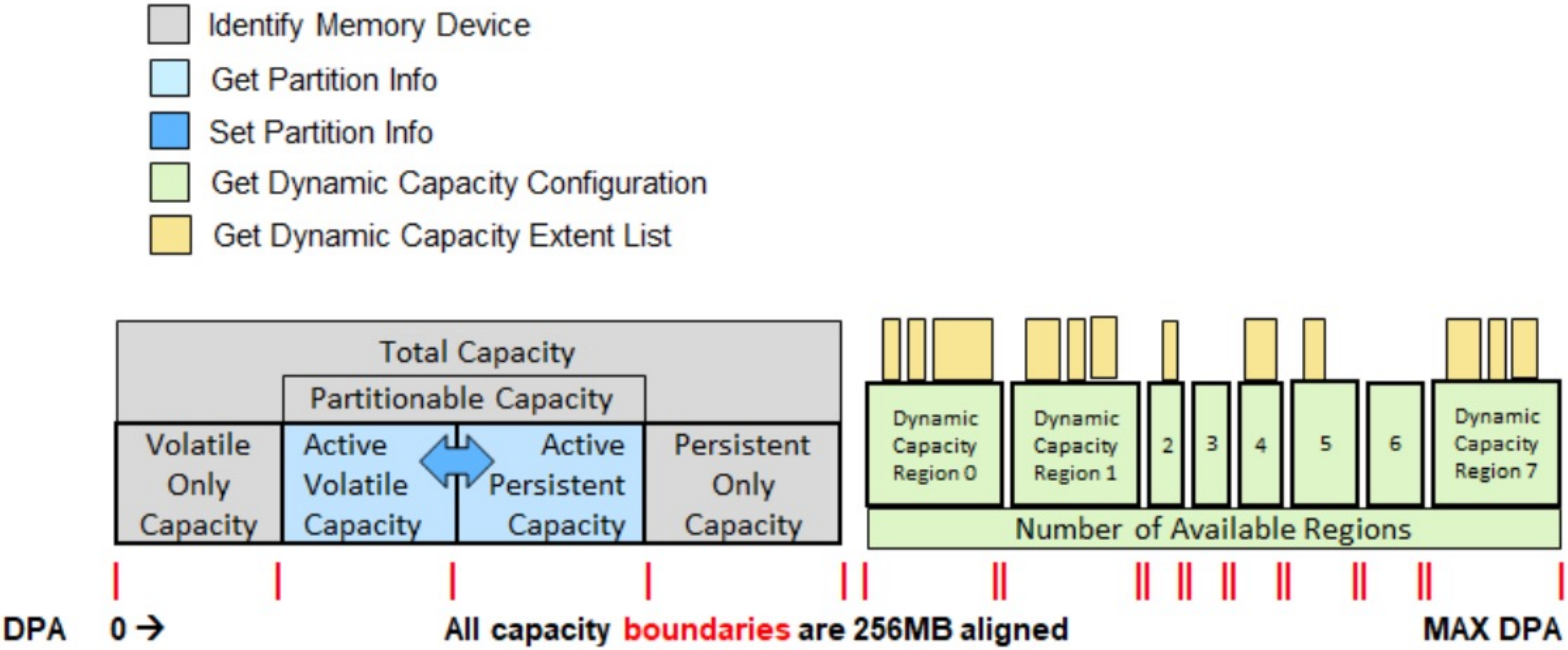
3. CXL 3.0 defined mechanisms to enforce hardware cache coherency between copies

# DCD SW Stack


Orchestrator (O)
OSV specific
Host 1 (H1)
Generic CXL Type 3 Driver
Host 2 (H2)
Generic CXL Type 3 Driver
Other CXL Fabric Elements
Other standards such as Redfish or Proprietary
New Dynamic Capacity Commands
FM API
Type 3 Logical Device 1 (LD1)
Type 3 Logical Device 2 (LD2)
Memory
Dynamic Capacity extensions to FM API
Fabric Manager (FM)
Multi-host Single Logical Device (MH-SLD)

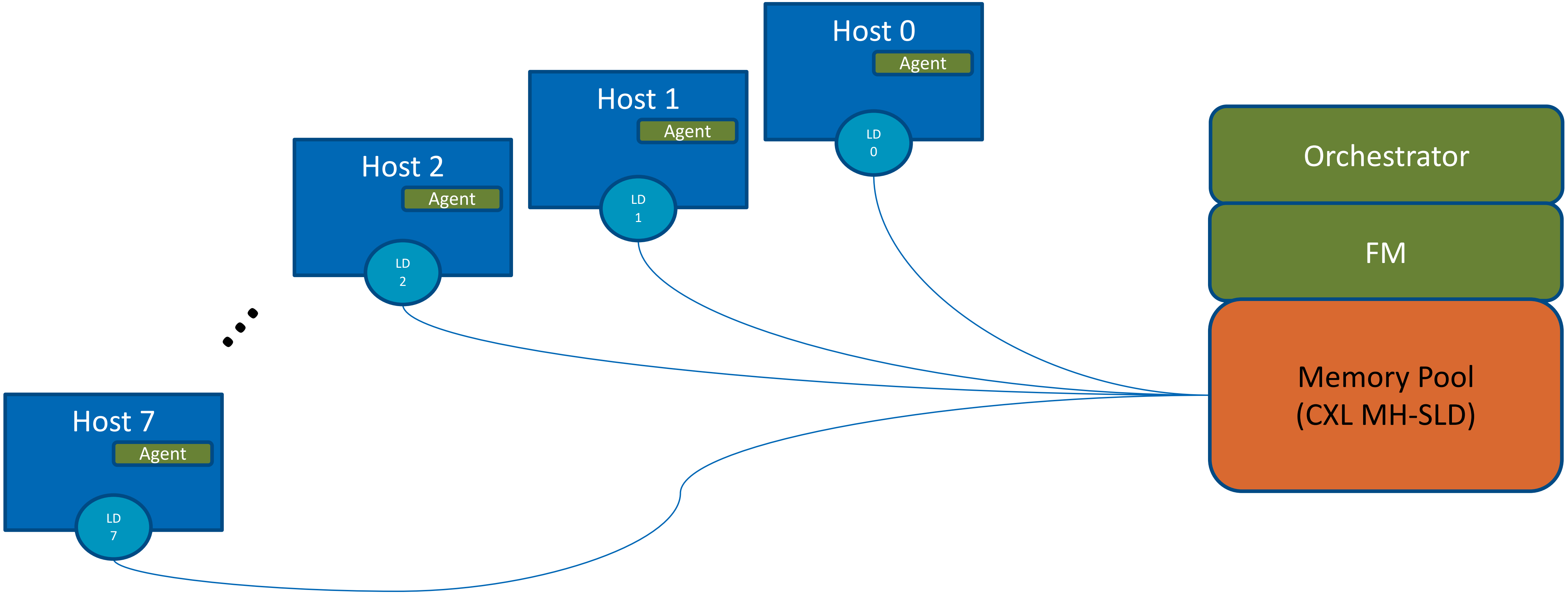
# Dynamic Capacity Device (DCD)

- Defined in CXL 3.0 Specification

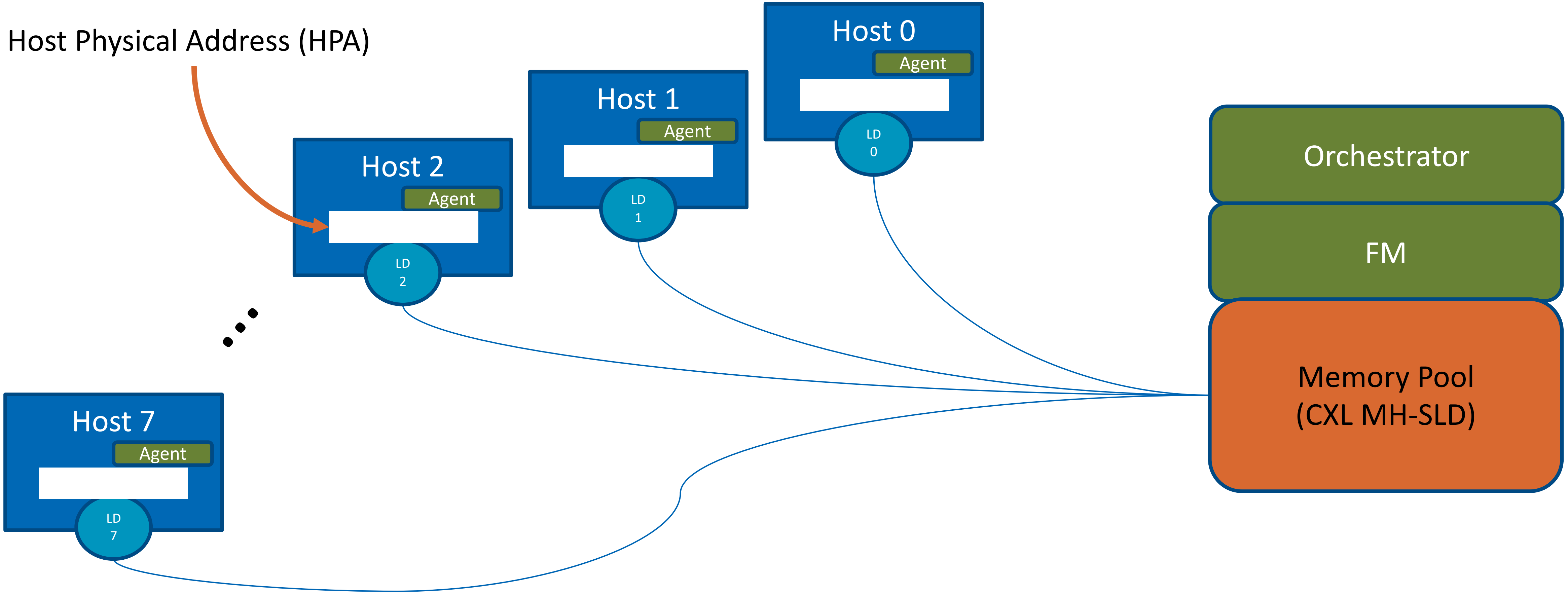


Identify Memory Device
Get Partition Info
Set Partition Info
Get Dynamic Capacity Configuration
Get Dynamic Capacity Extent List
Total Capacity
Partitionable Capacity
Volatile Only Capacity
Active Volatile Capacity
Active Persistent Capacity
Persistent Only Capacity
Dynamic Capacity Region 0
Dynamic Capacity Region 1
2
3
4
5
6
Dynamic Capacity Region 7
Number of Available Regions
DPA
0 →
All capacity boundaries are 256MB aligned
MAX DPA

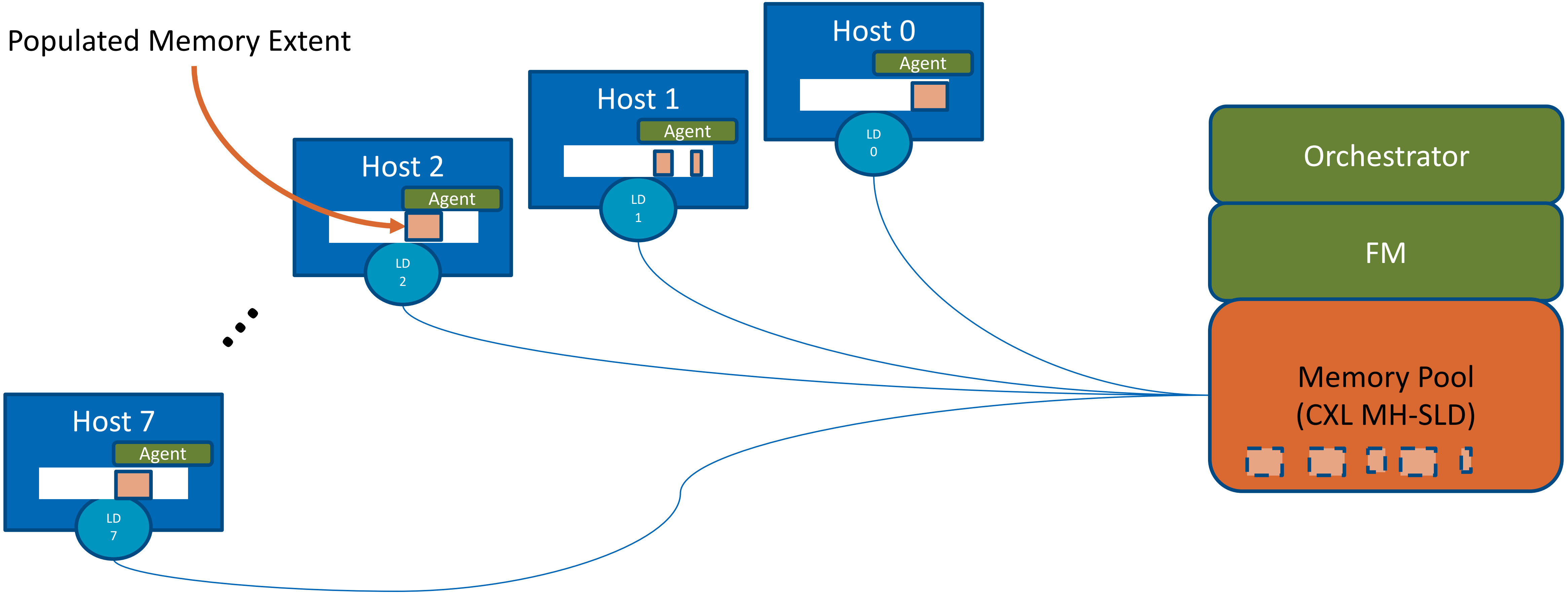
# Example: Memory Pool



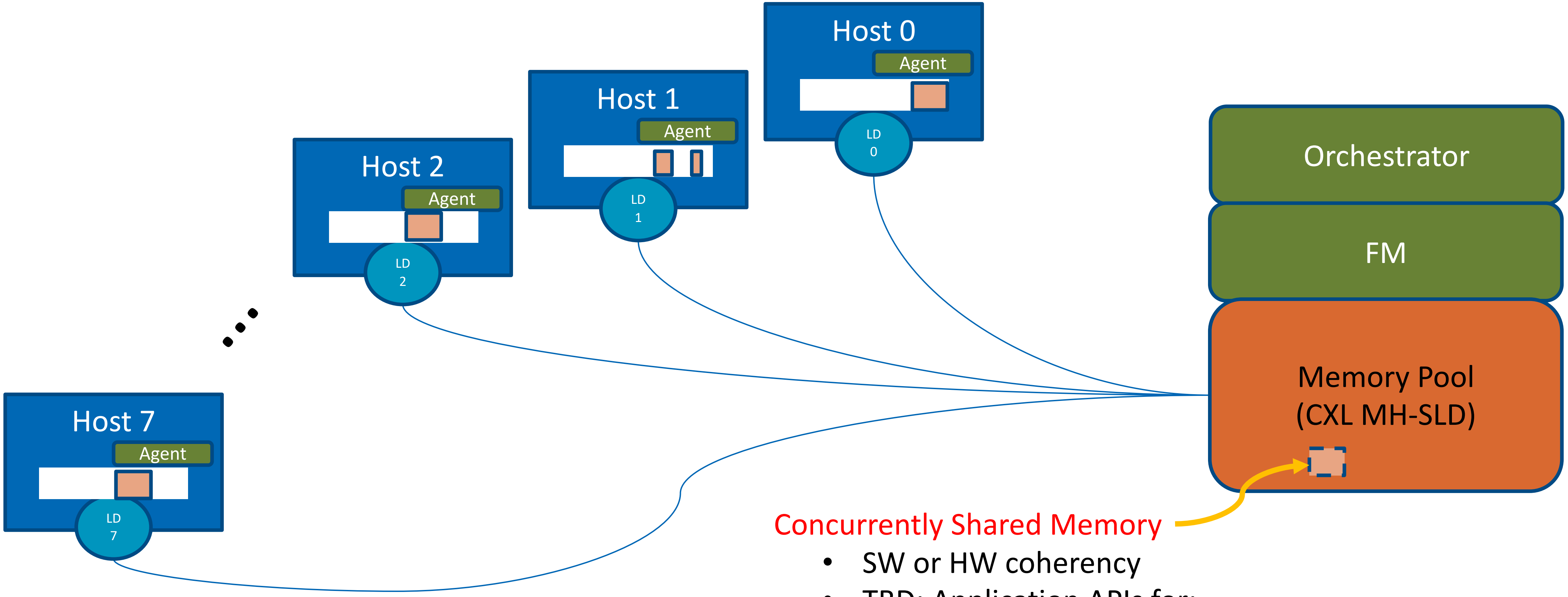
# Example: Initial HDM Decoder Programming



# Example: Add Memory



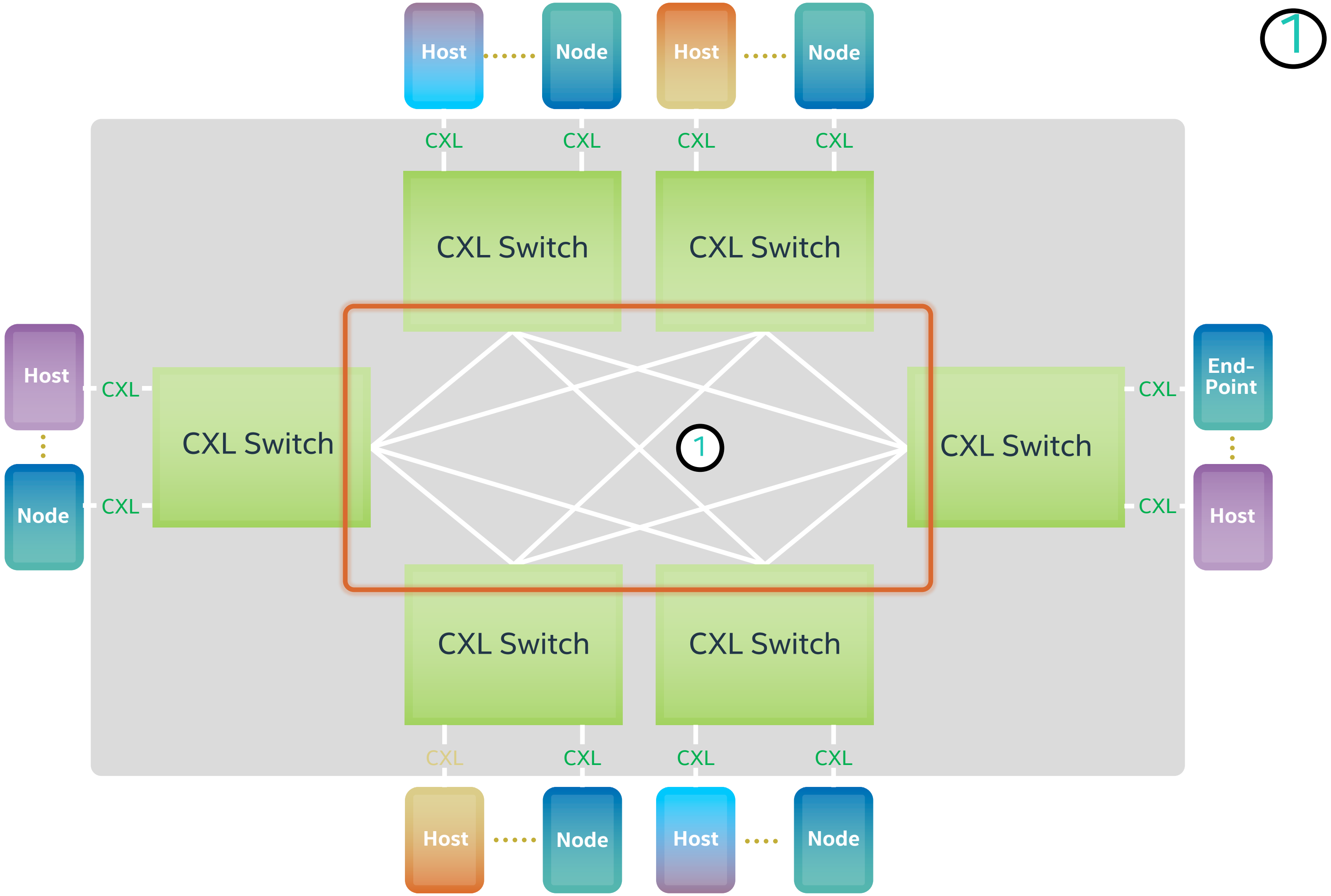
# Example: Shared Memory



Concurrently Shared Memory

- SW or HW coherency
- TBD: Application APIs for:
  - Allocation
  - Cross-host coordination

# CXL 3.0: Fabrics Example



1 Nodes can be any combination:

- Hosts
- Type 1 – Device with cache
- Type 2 – Device with cache and memory
- Type 3 – Device with memory

# The Memory Area Network (MAN)

## Modeled after the Storage Area Network (SAN)

### SAN

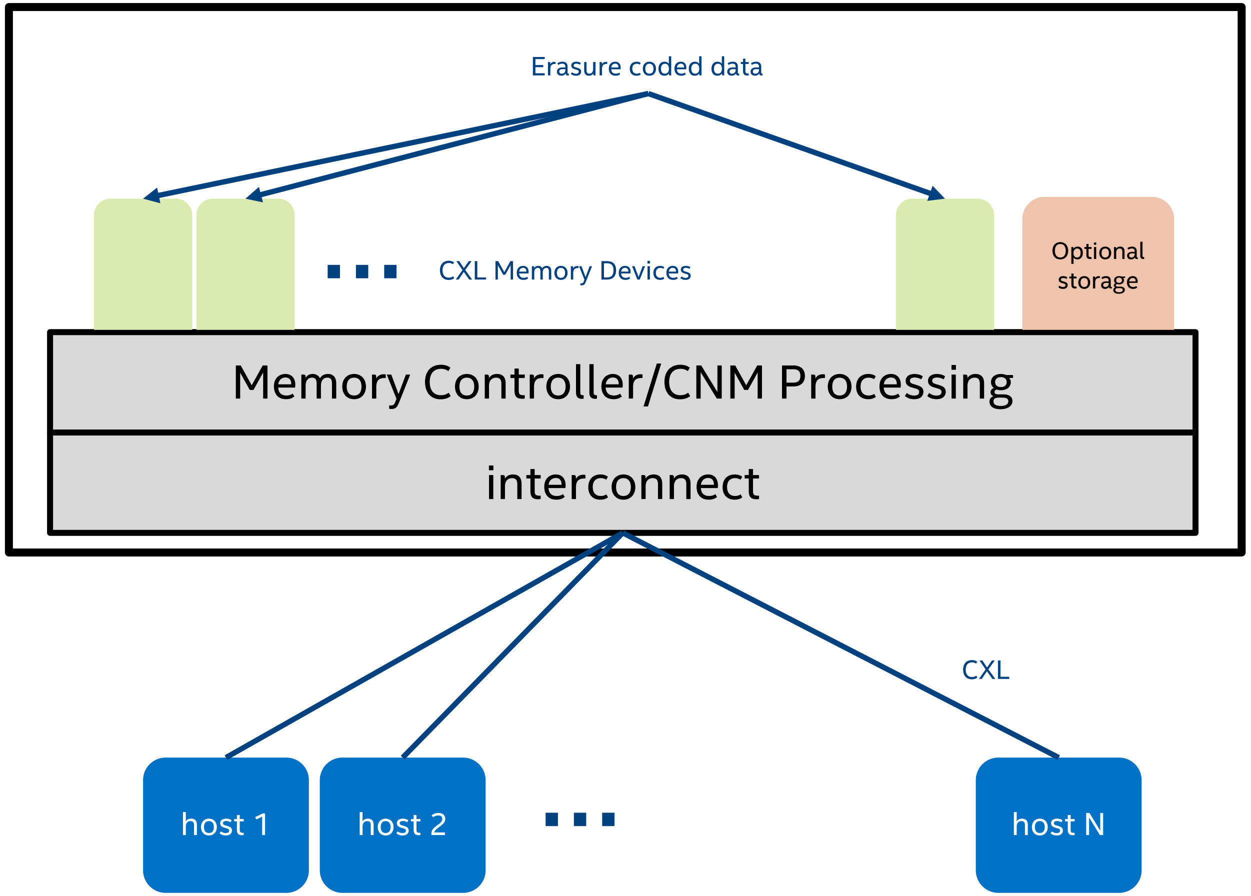
- Applications can use it like direct-connect **storage**
- Features added transparently:
  - Replication (i.e., RAID)
  - Management
  - Pooling/sharing between nodes
- Advanced features:
  - Processing in storage

### MAN

- Applications can use it like direct-connect **memory**
- Features added transparently:
  - Replication
  - Management
  - Pooling/sharing between nodes
- Advanced features:
  - Processing in memory

# The Memory Area Network

High RAS shared memory:
- Dynamic Capacity Device (DCD)
- Chip Kill
- DIMM Kill (CXL device kill)
- No SPOF
- PMem programming model



CNM Operations:
- Vector operations
- Shared access coordination
- PMem operations
  - transactions
  - allocations

Possible topologies:
- CXL interconnect
- Potentially other fabrics
- Replication:
  - RAID-style in pool
  - Between pools

Software:
- OneAPI
  - i.e., GPU APIs
- Shared Memory FS
- Shared PMDK libraries

# The Vision: Build on CXL Memory Pooling/Sharing

- **Memory Appliance Features**, similar to what SAN did for storage
  - Like transparent replication, higher RAS, advanced management

- Provide **Memory Tiering** to mitigate the latency of “far” CXL memory
  - IHVs can provide tiering features to add value to their products

- Provide the **PMem programming model**
  - Implementation could use <u>either</u> persistent or volatile media

- Build **Compute Near Memory** features into the pooled memory
  - Can share CNM logic and memory among hosts – no “stranded” resources

# Works, Needs Work, Really Needs Work

## Works

- App transparent NUMA
  - Kernel handles this
  - Most common case

## Needs Work

- Hot/Cold page telemetry
- NUMA APIs for Applications
  - Existing libraries are a good start
    - libnuma, libmemkind
  - Need easy abstraction for HMAT info

## Really Needs Work

- Make existing sharing APIs work
  - Not too difficult
    - OpenMP, OpenFabrics, existing PGAS work
- APIs to better leverage CXL sharing
  - Maintaining consistency
    - PMem work can be leveraged
  - Full load/store sharing
    - Like two local threads, but across hosts
    - Need easy abstraction for allocation/coordination