KubeCon CloudNativeCon North America 2026

#KubeCon #CloudNativeCon

CoHDI: Bridging Kubernetes and Disaggregated Memory for Memory-Aware LLM Inference

Naoki Oguchi, Fsas Technologies
- Hyde Sugiyama, RedHat

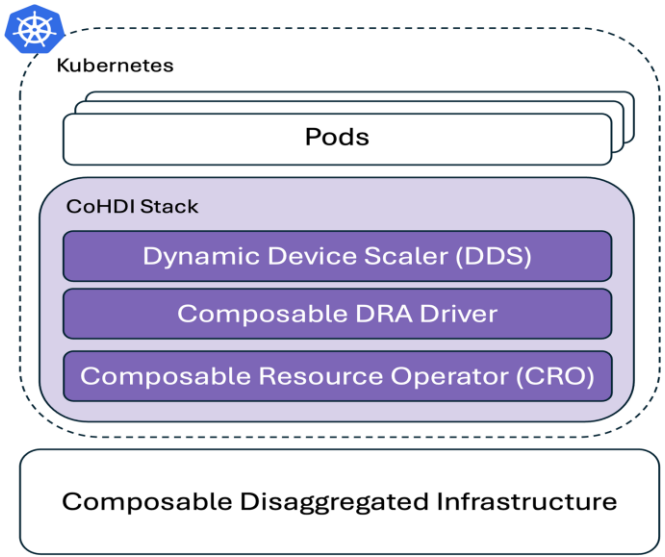


What Is CoHDI?

KubeCon CloudNativeCon North America 2026

CoHDI(**Co**mposable **H**ardware in **D**isaggregated **I**nfrastructure) enables Kubernetes to dynamically compose disaggregated hardware resources

- Selected as a CNCF Sandbox project (Dec 2025)
- Part of the CNCF ecosystem (including Kubernetes)
- Open collaboration across industry
- Addressing emerging challenges in computing infrastructure

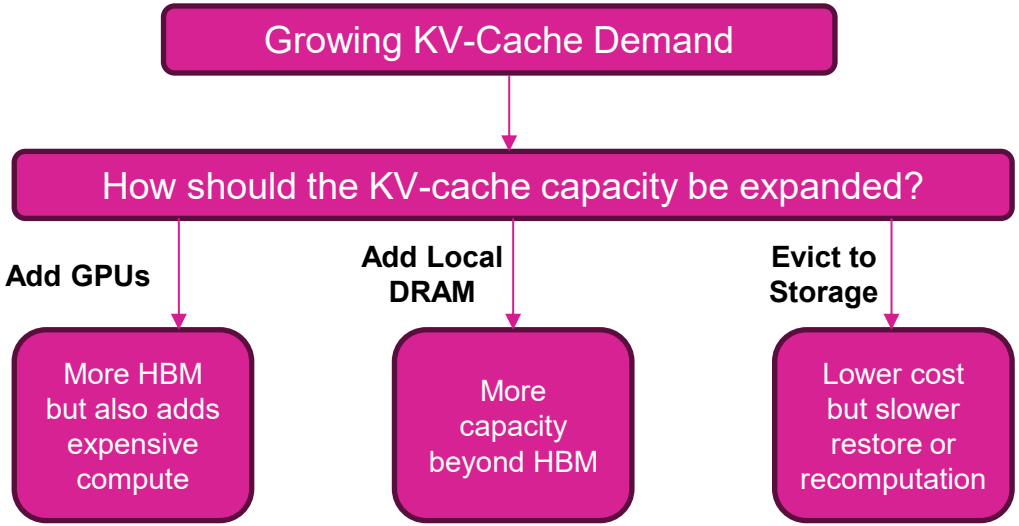


- **Composable Resource Operator (CRO):**
  - ✓ Handles the physical attachment and detachment of pooled hardware resources.
- **Composable DRA Driver:**
  - ✓ Gives Kubernetes a live, accurate view of composable infrastructure capacity
  - ✓ Abstracts low-level hardware details
- **Dynamic Device Scaler:**
  - ✓ Tracks pending requests for resources and triggers the required hardware scaling actions.

CoHDI

Growing KV-Cache Working Sets Exceed GPU Memory

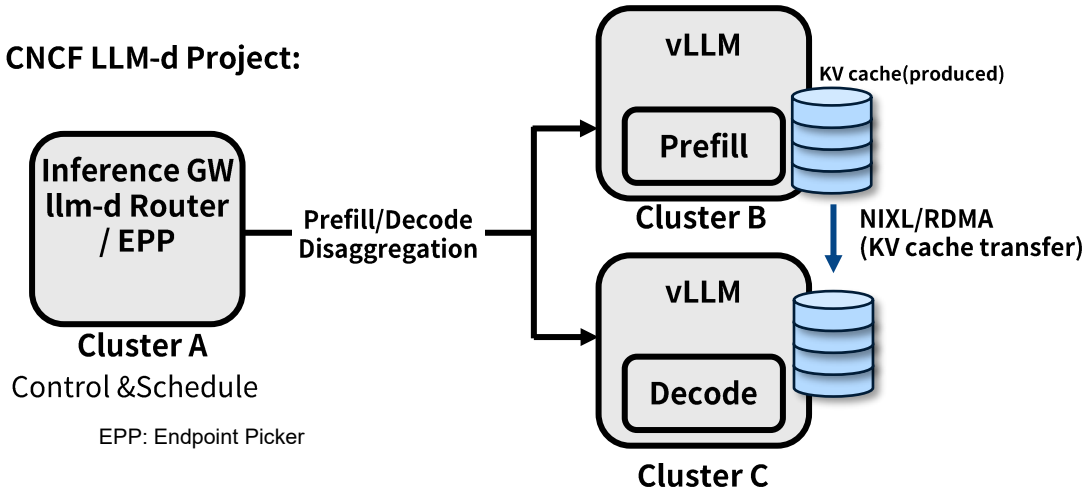
- Agentic and long-context workloads create large reusable KV-cache working sets that cannot always fit in GPU HBM.

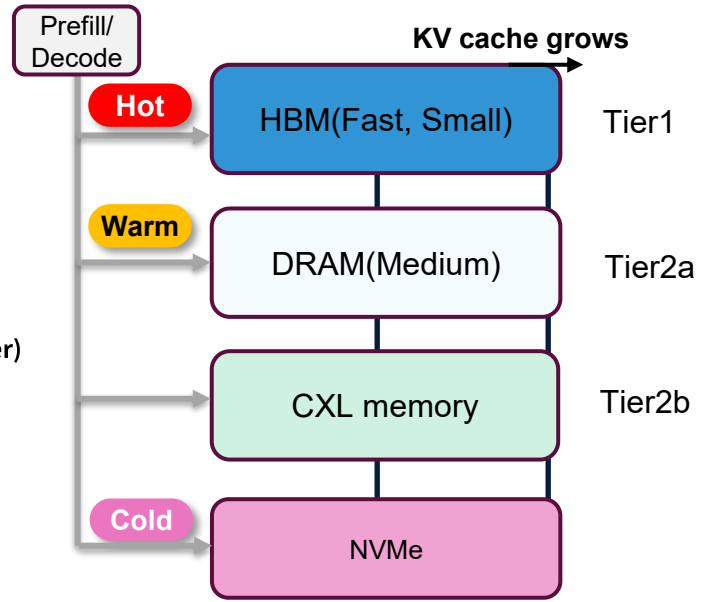


KV-cache capacity needs to scale independently from GPU compute.

Tiered KV Caching Extends Capacity Beyond GPU HBM

- llm-d separates Prefill and Decode and enables KV cache to extend beyond HBM through multiple memory tiers
  - Prefill → GPU Cluster B (compute-intensive)
  - Decode → GPU Cluster C (latency-optimized)
  - KV Cache → HBM / DRAM / CXL Memory

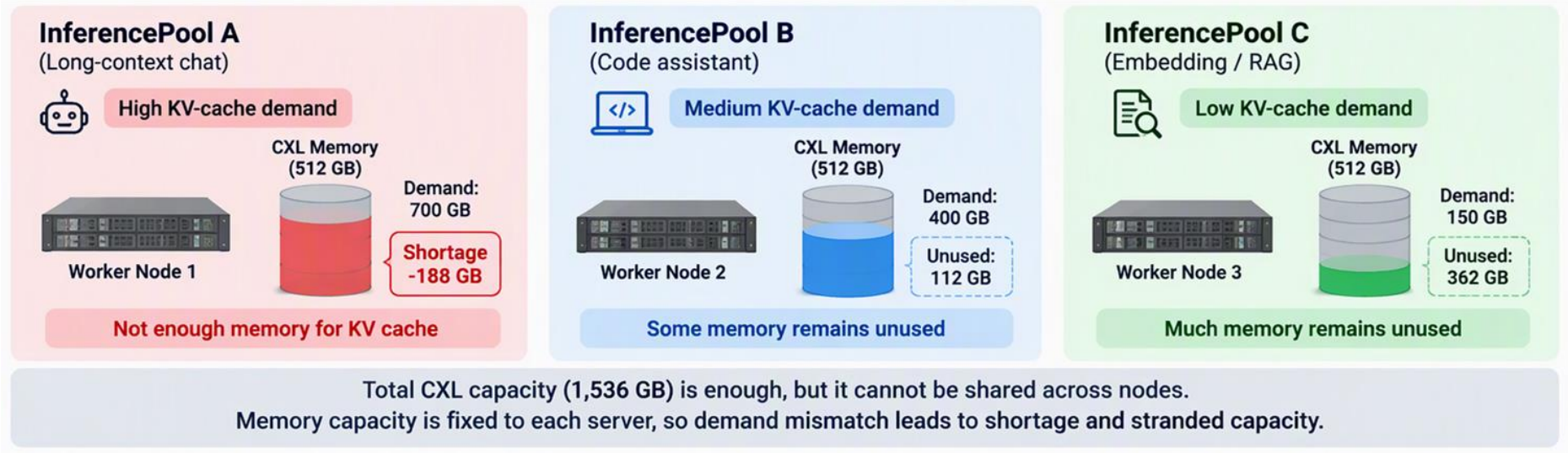




CXL memory can serve as a large KV-cache capacity tier

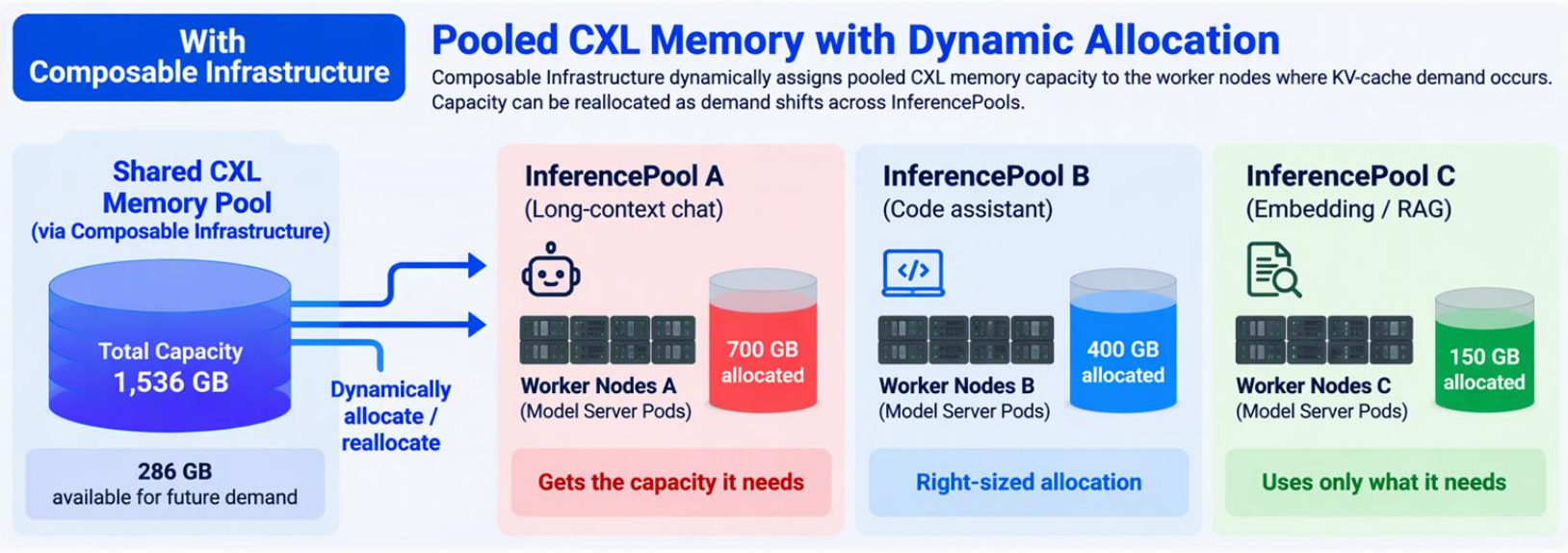
Fixed Memory Capacity Cannot Follow Dynamic KV-Cache Demand

- KV-cache demand changes with concurrency, context length, workload lifecycle, and cache reuse patterns
- However, local DRAM and direct-attached CXL memory remain fixed to individual servers



Dynamic CXL Memory for Shared LLM Inference Platforms

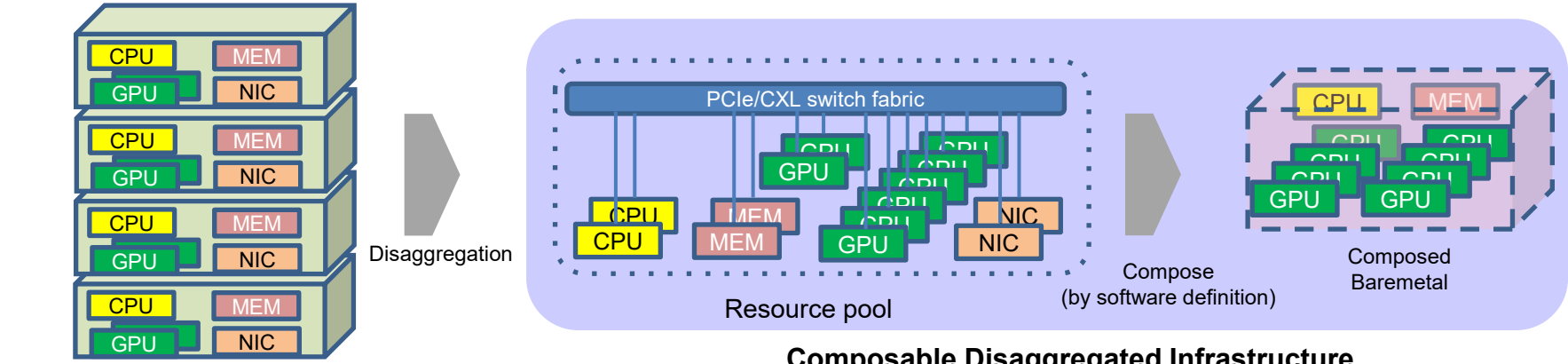
- Organizations often operate shared LLM inference platforms with multiple InferencePools for different AI services.
- However, KV-cache demand shifts across pools and worker nodes, while direct-attached memory capacity remains fixed.



CDI turns fixed CXL memory into elastic capacity that follows KV-cache demand

Why Composable Infrastructure Enables This

- **Composable infrastructure breaks the fixed server model and enables on-demand, AI-optimized systems**
  - Disaggregate hardware into resource pools (CPU, GPU, MEM, NIC)
  - Dynamically compose “baremetal” systems per workload
  - Connected via PCIe/CXL fabric



Traditional cloud Infrastructure

Composable Disaggregated Infrastructure
Dynamically match GPU, memory, and bandwidth precisely to AI workload demands

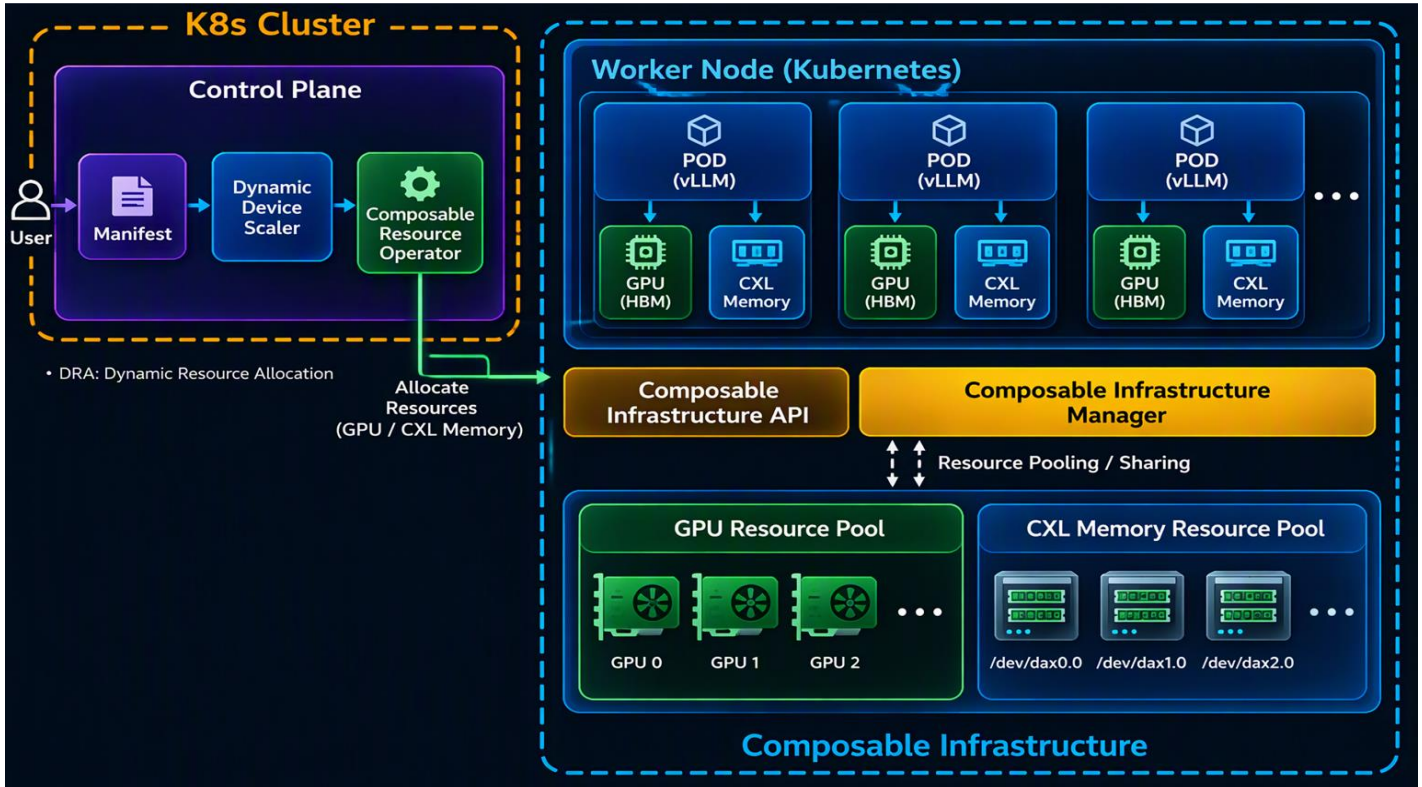
- **Exposed and managed as Kubernetes resources**

CoHDI: Realizing composable infrastructure with Kubernetes

KubeCon CloudNativeCon North America 2026

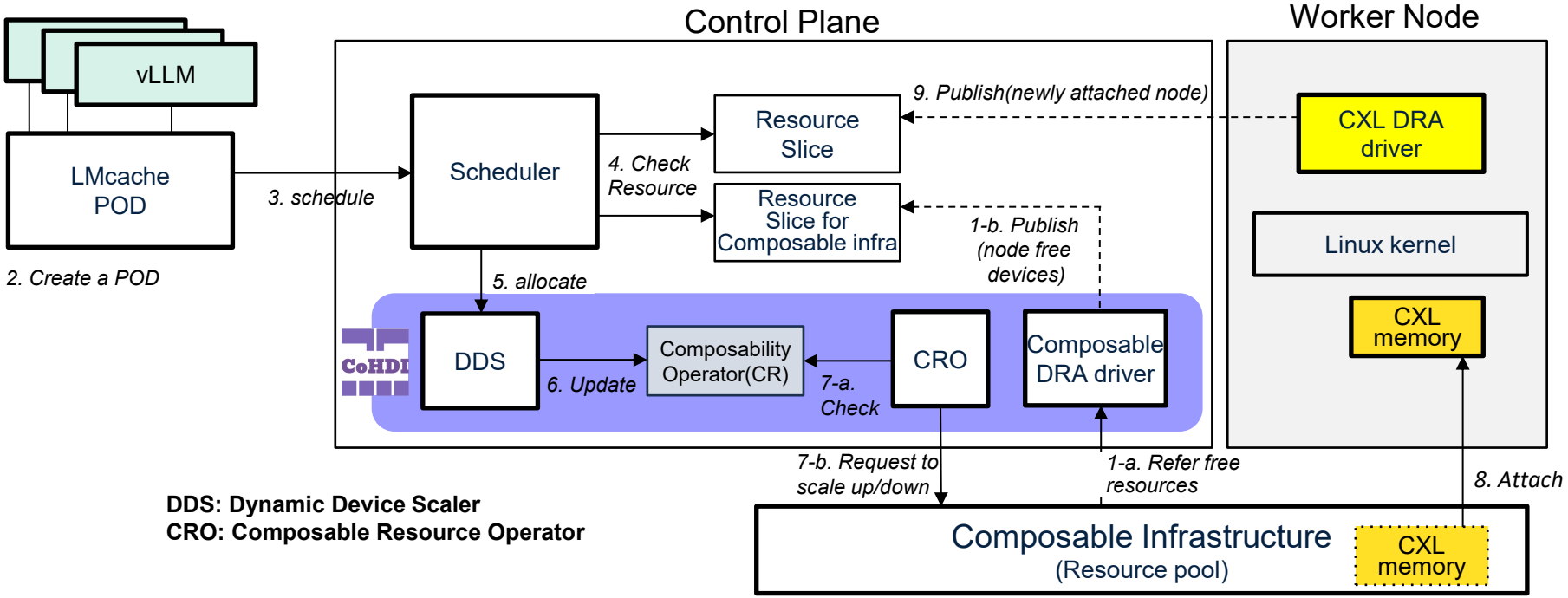
CoHDI extends Kubernetes from workload orchestration to infrastructure orchestration

1. Pod scheduling initiates composition
2. Dynamic Device Scaler determines required resources
3. Composable Resource Operator requests resources
4. Resources are attached dynamically



Dynamic CXL Memory Allocation for LLM-d with CoHDI

KubeCon CloudNativeCon North America 2026



CoHDI translates Kubernetes workload demand into dynamic CXL memory composition

Tiered KV Cache Management with DRAM and CXL Device-DAX

KubeCon CloudNativeCon North America 2026

1 Allocate the CXL Device-DAX resource

The CXL DRA driver assigns a /dev/dax device to the LMCache instance.

2 Provision two cache tiers

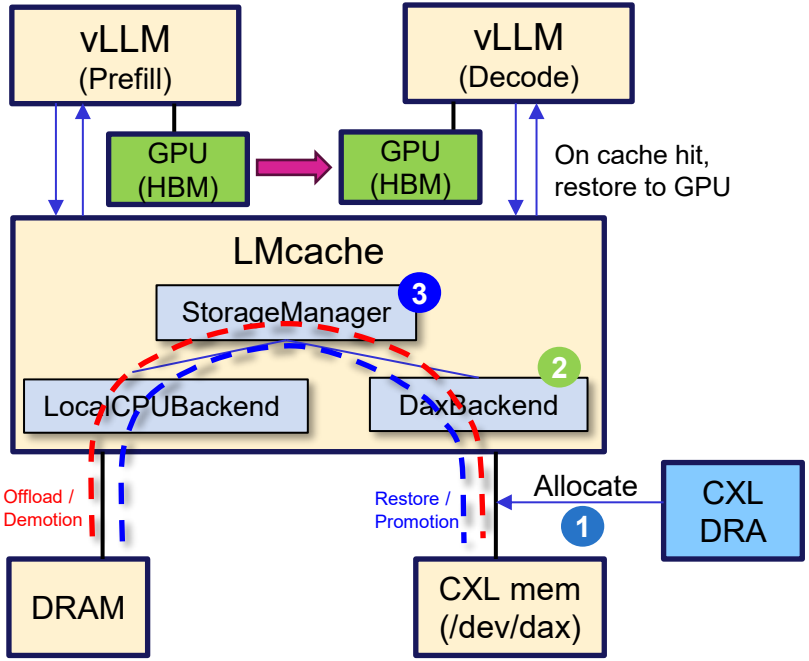
DRAM Pinned / page-locked host memory managed by LocalCPUBackend.

CXL DaxBackend maps /dev/dax with mmap(MAP_SHARED) as a fixed-size KV arena.

3 Move KV-cache chunks by policy

StorageManager coordinates chunk-level movement:

Demotion DRAM → CXL
Promotion CXL → DRAM
Eviction when capacity is exhausted(e.g. 80%)



Metrics changes

Demoting KVcache chunks after exceeding 80% of DRAM

```
UTC: 2026-10-07T11:33:38.160402+00:00
lmcache_mp_l1_usage_ratio 0.7119140625
lmcache_mp_l1_eviction_loop_triggered_total 44.0
lmcache_mp_l2_usage_bytes{l2_name="dax"} 3.3558626304e+010
l2_evicted_objects: not exposed
lmcache_mp_l2_prefetch_load_completed_chunks_total 105.0

UTC: 2026-10-07T11:33:40.294328+00:00
lmcache_mp_l1_usage_ratio 0.84375
lmcache_mp_l1_eviction_loop_triggered_total 44.0
lmcache_mp_l2_usage_bytes{l2_name="dax"} 3.4124857344e+010
l2_evicted_objects: not exposed
lmcache_mp_l2_prefetch_load_completed_chunks_total 105.0

UTC: 2026-10-07T11:33:42.307912+00:00
lmcache_mp_l1_usage_ratio 0.6767578125
lmcache_mp_l1_eviction_loop_triggered_total 45.0
lmcache_mp_l2_usage_bytes{l2_name="dax"} 3.4124857344e+010
l2_evicted_objects: not exposed
lmcache_mp_l2_prefetch_load_completed_chunks_total 105.0

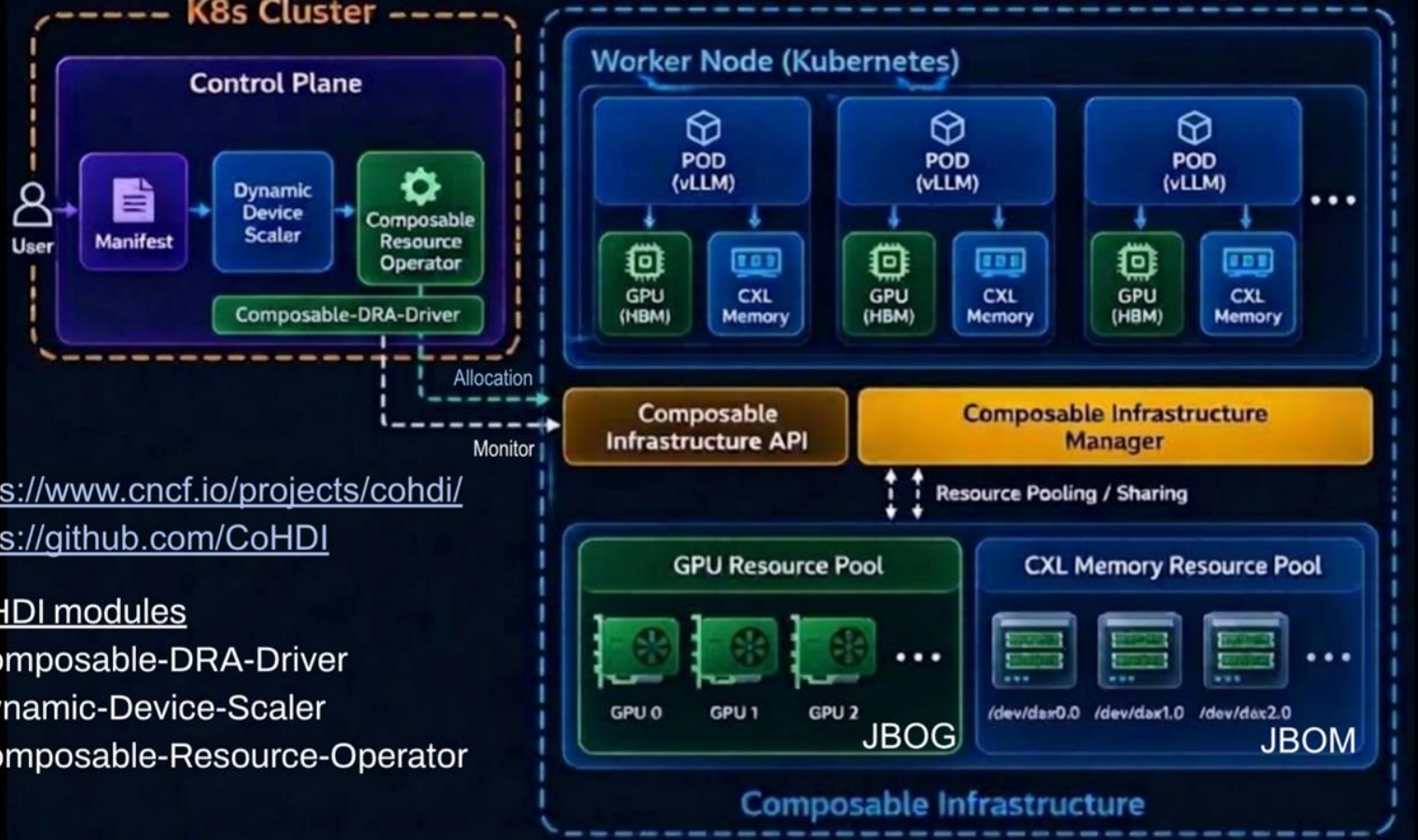
UTC: 2026-10-07T11:33:44.729552+00:00
lmcache_mp_l1_usage_ratio 0.7470703125
lmcache_mp_l1_eviction_loop_triggered_total 45.0
lmcache_mp_l2_usage_bytes{l2_name="dax"} 3.4426847232e+010
l2_evicted_objects: not exposed
lmcache_mp_l2_prefetch_load_completed_chunks_total 105.0
```

User Prompt	P1	P2	P3	P4		P1 (again)
L1 (DRAM) usage:	71%	84%	67%	74%	…	67%
Number of L1 evictions:	44	44	**45**	45	…	49
L2 (DAX) usage:	33 GB	34 GB	34 GB	34GB	…	36 GB
Number of L2 evictions :	None	None	None	None	…	None
Blocks read from L2:	105	105	105	105	…	**135**

Re-submitting a previous prompt promotes the KV cache from CXL memory

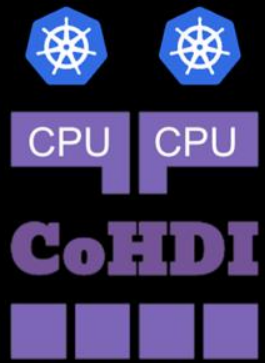
```
UTC: 2026-10-07T11:34:17.086364+00:00
lmcache_mp_l1_usage_ratio 0.6767578125
lmcache_mp_l1_eviction_loop_triggered_total 49.0
lmcache_mp_l2_usage_bytes{l2_name="dax"} 3.6956012544e+010
l2_evicted_objects: not exposed
lmcache_mp_l2_prefetch_load_completed_chunks_total 135.0
```

CoHDI: Composable Hardware in Disaggregated Infrastructure



https://www.cncf.io/projects/cohdi/
https://github.com/CoHDI

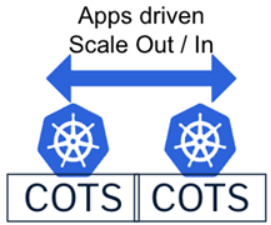
CoHDI modules
- Composable-DRA-Driver
- Dynamic-Device-Scaler
- Composable-Resource-Operator



GPU, RNIC, CXL memory, Others

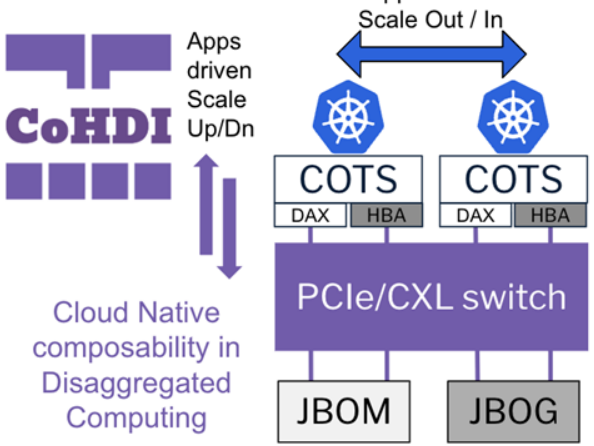
Transitioning from COTS to Next-Generation Computing Systems for AI Computing Continuum with scale-across - IOWN PoC

(Phase 1)



(Phase 2)

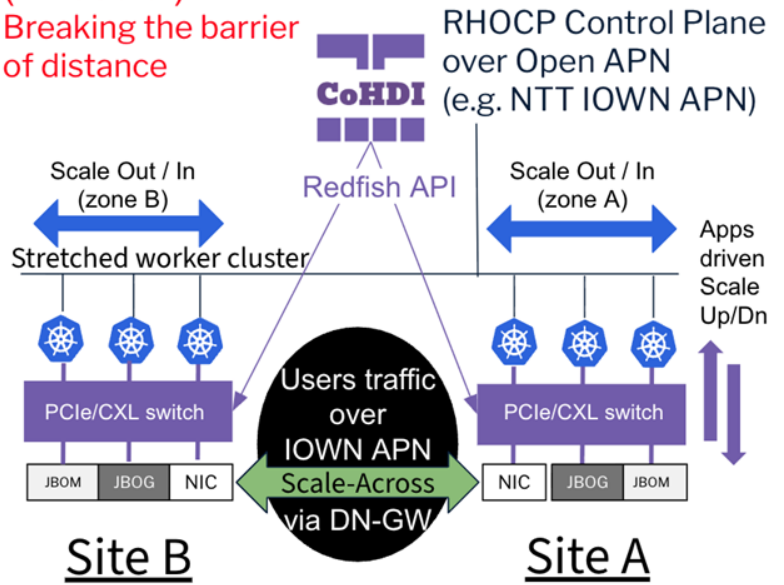
Breaking through the limitations of managing fixed COTS hardware resources



HBA: Host Bus Adapter card for GPU/NIC
JBOG: Just a Bunch of GPUs
DAX: CXL Cable card for CXL memory in JBOM
JBOM: Just a Bunch of Memory (CXL Memory)

(Phase 3)

Breaking the barrier of distance



e.g. PD Disaggregation w/ GPU Multi-Rail RoCEv2 over IOWN APN via DN-GW for Distributed LLM inference

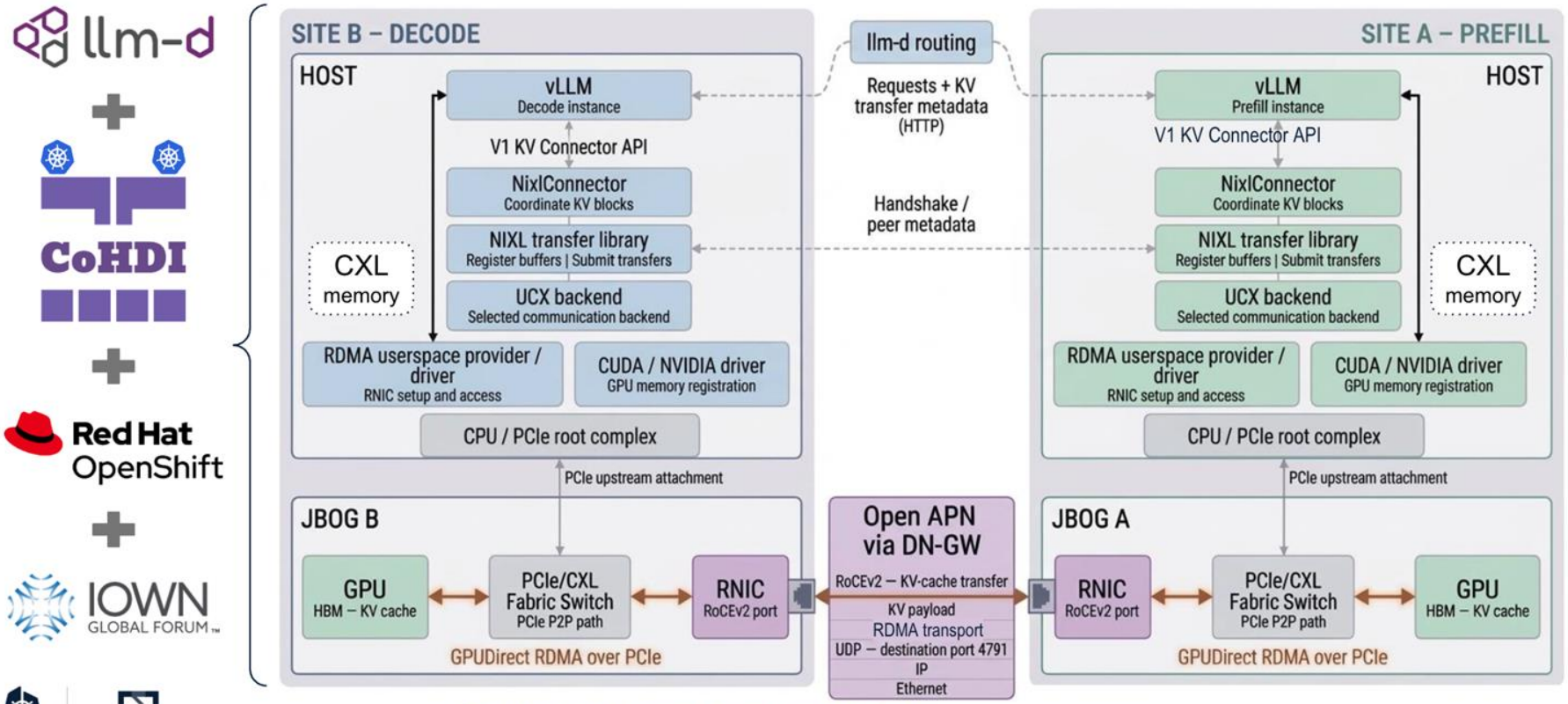
KubeCon CloudNativeCon North America 2026

https://github.com/CoHDI

CoHDI catalog https://catalog.redhat.com/en/search?q=CoHDI

Cross-Site KV-Cache Transfer over Open APN(optical network)

Preserve local GPU-RNIC affinity. Transfer KV between sites over Open APN.



GPU Multi-Rail supported by DN-GW
for AI-WAN(Open All-Photonics Network)

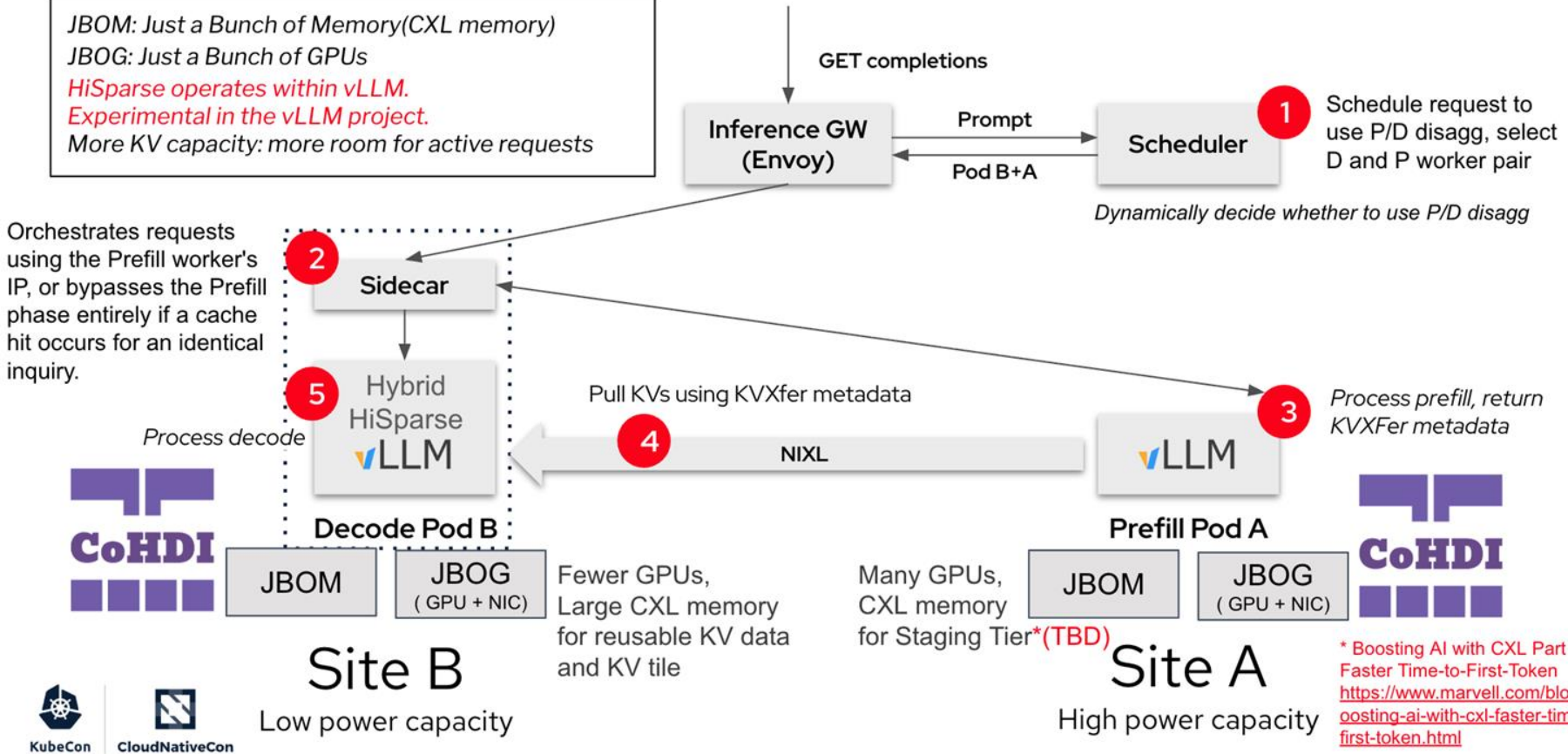
Visit IOWN Global Forum booth #275 !

KubeCon CloudNativeCon North America 2026

PD Disaggregation with Composable Disaggregated Infrastructure

llm-d llm-d composes vLLM and Gateway to enable P/D disaggregation

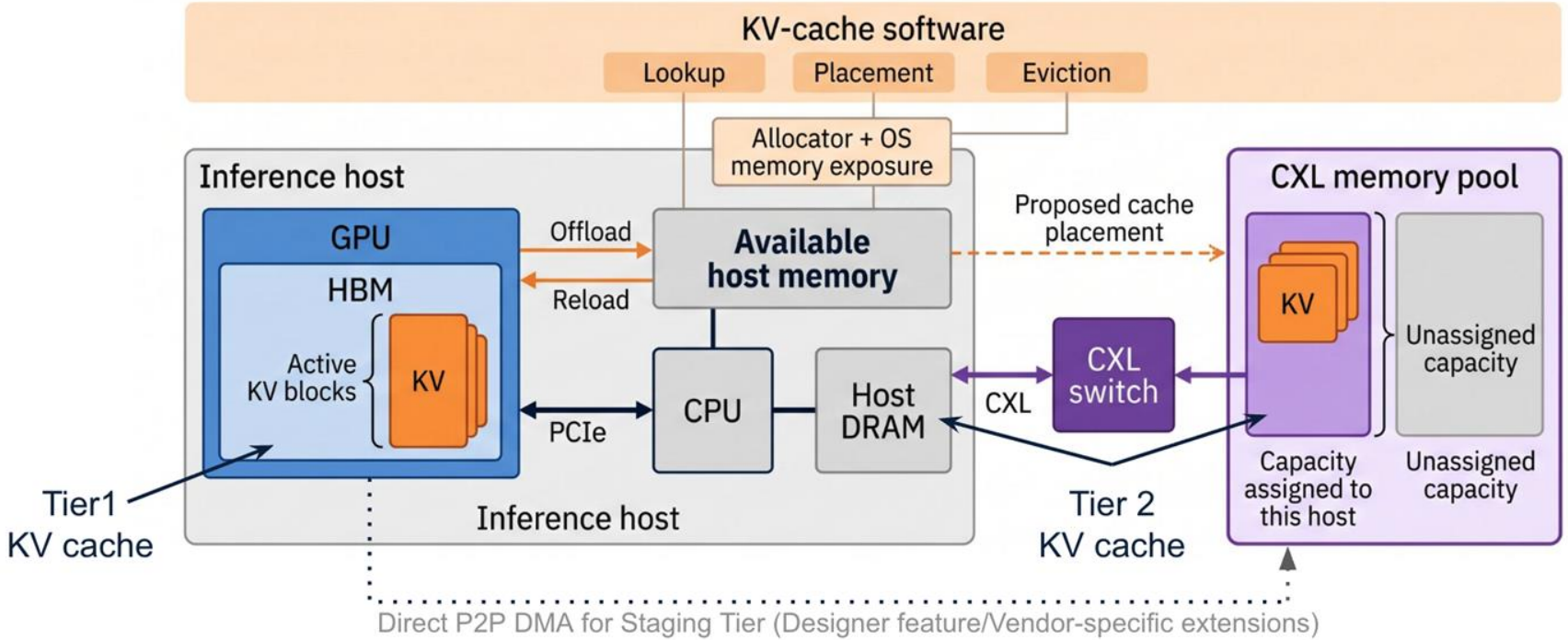
JBOM: Just a Bunch of Memory(CXL memory)
JBOG: Just a Bunch of GPUs
HiSparse operates within vLLM.
Experimental in the vLLM project.
More KV capacity: more room for active requests



* Boosting AI with CXL Part III: Faster Time-to-First-Token https://www.marvell.com/blogs/boosting-ai-with-cxl-faster-time-to-first-token.html

KV Cache Beyond GPU Memory: The CXL Opportunity

Additional memory capacity for retaining and reusing KV blocks.



Memory pooling assigns capacity. Cache software manages reusable KV blocks.

Takeaway: Reuse is valuable when fetching cached KV costs less than recomputing it.

Key Takeaways

KubeCon CloudNativeCon North America 2026

Dynamic composition for CXL memory by CoHDI

Adding CXL memory as KV cache

Overcoming hardware resource imbalance:

Overcoming the constraints of fixed COTS resources with CoHDI.

Unlocking synergies with LLM-d PD Disaggregation & HiSparse:

- Since HiSparse in vLLM operates natively on Decode instances (while Prefill handles initial heavy-compute chunking and prefix caching), Decode nodes become purely memory-bound workers.
- Backed by CoHDI-orchestrated JBOM CXL memory pools, these Decode nodes can maintain high concurrency across million-token contexts—eliminating OOM crashes and prohibitive preemption penalties.

Breaking through the distance barrier:

Achieving location-agnostic compute via the All-Photonics Network (e.g., IOWN APN).

IOWN Global Forum booth Location: #275

https://www.cncf.io/projects/cohdi/

KubeCon CloudNativeCon
North America 2026

Q & A

