Octopus: Enhancing CXL Memory Pods via Sparse Topology

Yuhong Zhong Fiodar Kazhamiaka Pantea Zardoshti
Shuwei Teng Rodrigo Fonseca Mark D. Hill
Daniel S. Berger

Columbia University *Microsoft* *University of Washington* *University of Wisconsin–Madison*

Executive Summary: CXL as “Pod Area” Network

Background:

- Memory is expensive (~50% of CapEx) but underutilized due to stranding
- CXL enables a “pod” of servers to reach a common **memory pool**
- CXL also enables intra-pod **communication at memory-like latency**
- However, existing CXL pods are either **expensive** (using CXL switches) or **small** (using multi-port CXL devices, each with ≤ 4 ports)

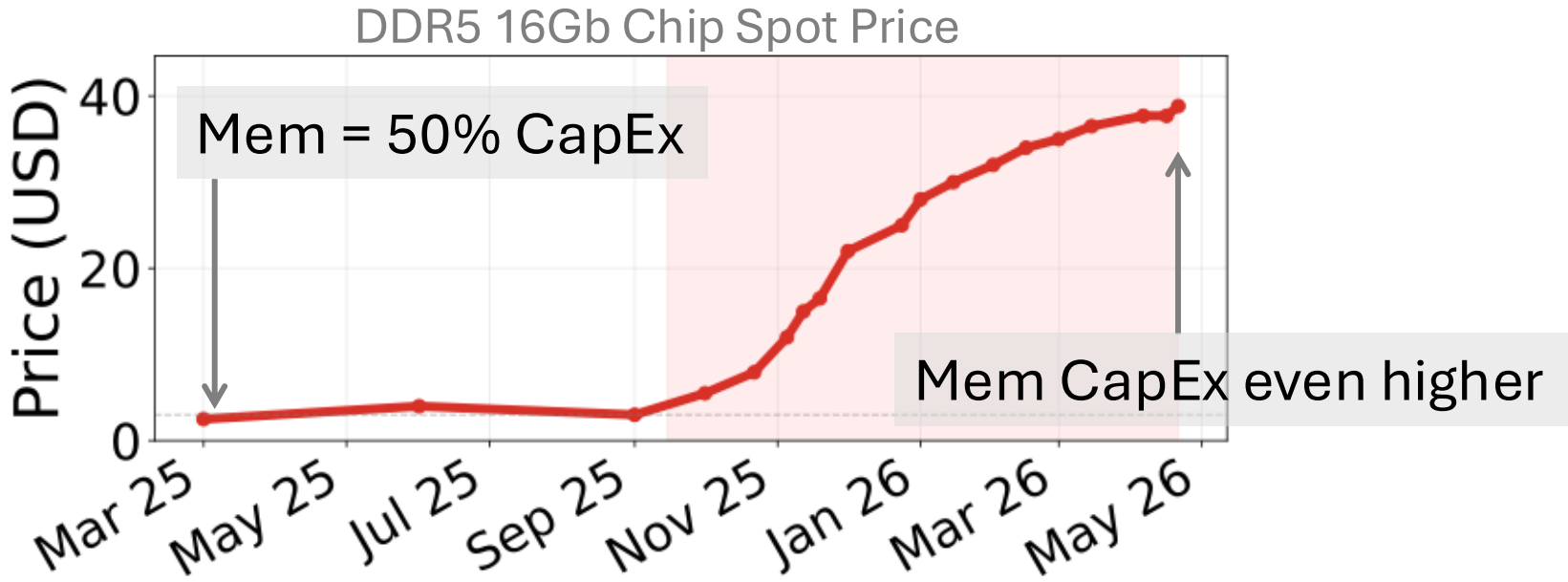
Contributions:

- Octopus: a low-cost, large-scale CXL pod which **sparsely** connects servers with low-port CXL devices, without using costly switches
  - To balance conflicting demands of memory pooling and intra-pod communication, Octopus partitions a pod into “**islands**”
- Results: 16% memory saving, < 1 μs intra-pod communication

Memory Accounts for Up to 50% Server CapEx

In 2023, memory accounts for **50% of server CapEx**

Recent demand surge increases DRAM prices by 10×



Memory capacity is expensive and limited: **utilization** is critical!

Li el al., Pond: CXL-Based Memory Pooling Systems for Cloud Platforms, ASPLOS 2023
Zhong et al., Oasis: Pooling PCIe Devices Over CXL to Boost Utilization, SOSP 2025
DDR5 16Gb Chip Spot Price from TrendForce DRAMeXchange

CXL to Rescue Memory Efficiency via Pooling

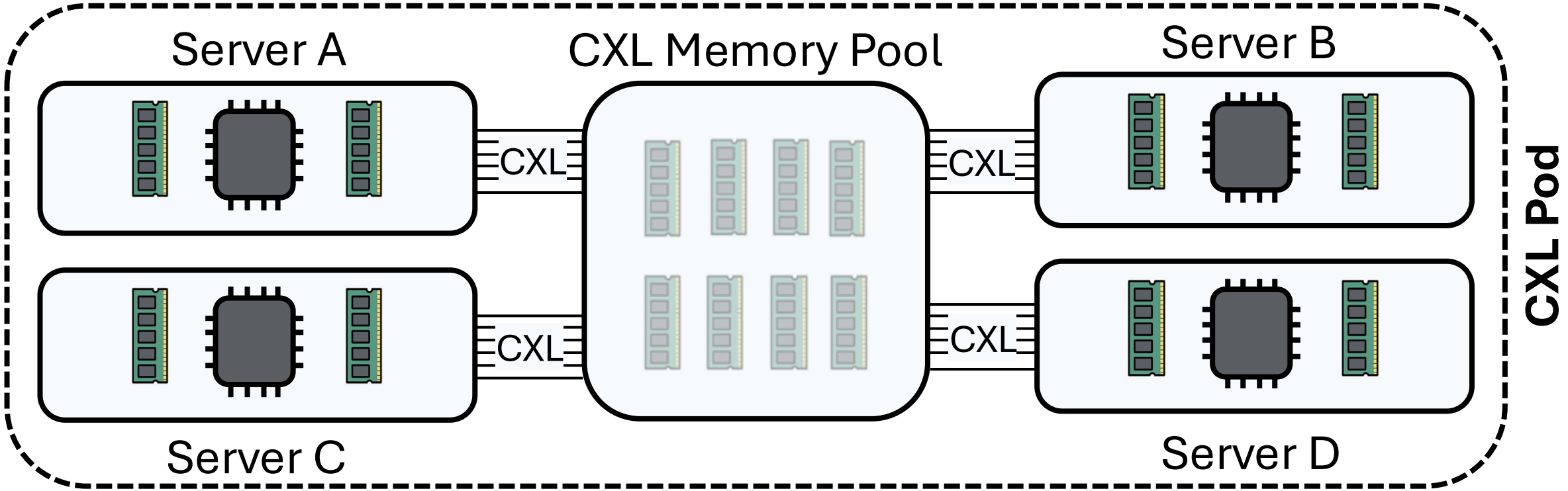
Memory capacity is overprovisioned for **peak demand per-server**

- Idle memory cannot be used by other servers, being “stranded”

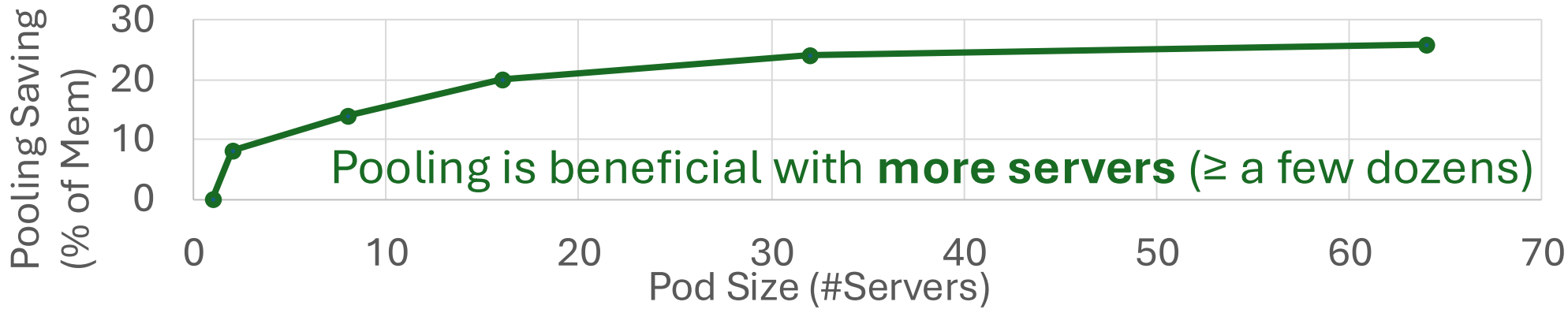
CXL to Rescue Memory Efficiency via Pooling

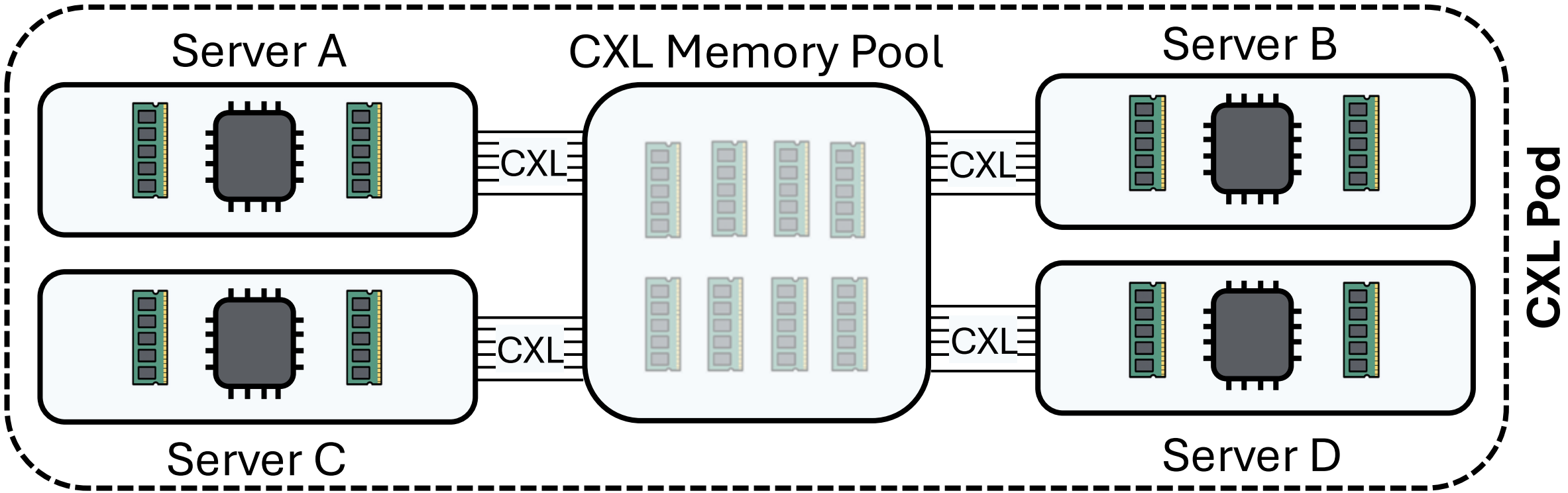
CXL memory pools **multiplex** memory demand across servers

- CXL memory allocated *on demand*, less stranded capacity
- Provision capacity for *aggregate peak*, not peak × N



CXL to Rescue Memory Efficiency via Pooling





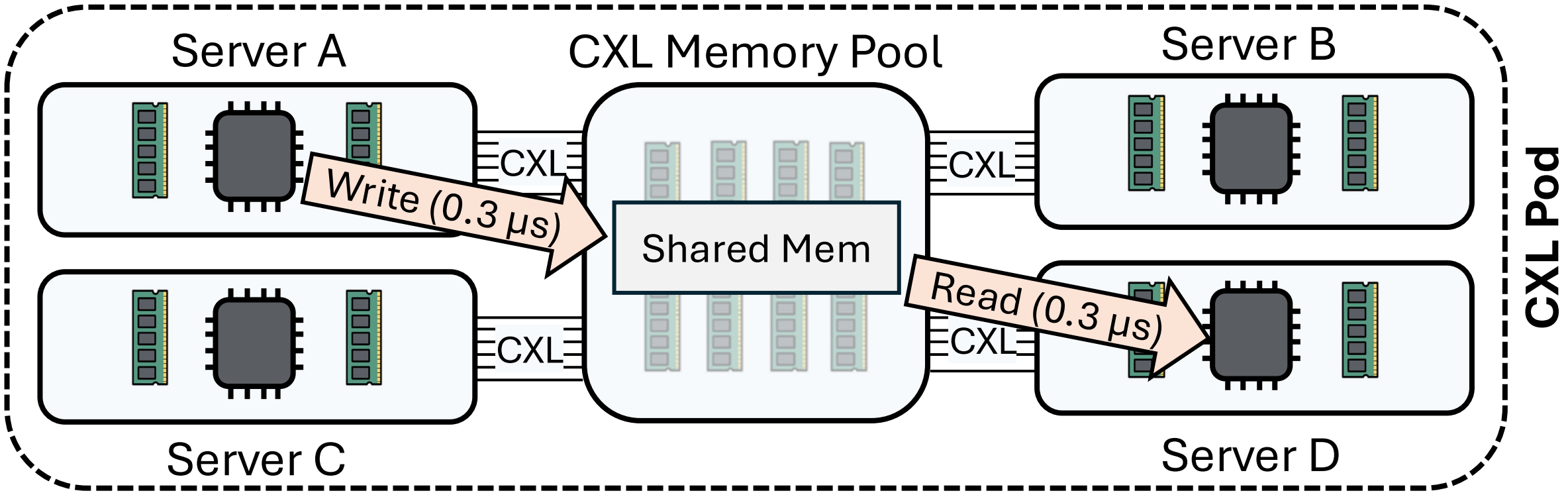
Li et al., Pond: CXL-Based Memory Pooling Systems for Cloud Platforms, ASPLOS 2023

Wang et al., cMPI: Using CXL Memory Sharing for MPI One-Sided and Two-Sided Inter-Node Communications, SC 2025
Ahn et al., An Examination of CXL Memory Use Cases for In-Memory Database Management Systems Using SAP HANA, VLDB 2024
Suetterlein et al., Synchronization for CXL Based Memory, MEMSYS 2024
Huang et al., Tigon: A Distributed Database for a CXL Pod, OSDI 2025

CXL Enables Low-Latency Communication

For communication, CXL (0.6 µs) is much faster than RDMA (~25 µs)

- Especially useful for distributed systems (e.g., consensus protocols, collective communication, and scale-out databases)



Design CXL Pods to Enable Pooling and Comm.

Key CXL use cases: **Memory pooling** and **low-latency communication**

For memory pooling:

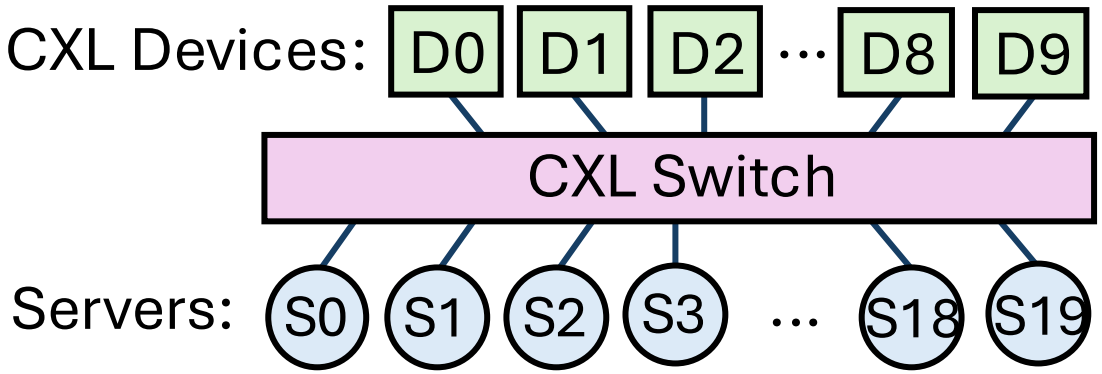
- (#1) **Large CXL pods** to pool across many servers (a few dozens)
- (#2) **Low CXL latency** to pool memory for more workloads
  - Lower latency → more workloads can tolerate CXL → more memory to pool

For commication at memory-like latency:

- (#3) **Common CXL devices** between servers to use shared CXL mem

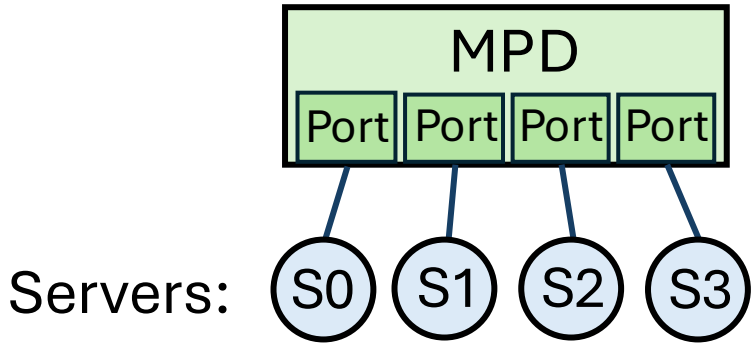
Challenge: Building CXL Pods With Today's CXL Devices

Option 1: **CXL Switches**



- ✗ High latency (490-600 ns)
  - Less memory to pool (35% of total mem)
- ✗ Expensive to deploy ($5000+)
- ✓ High fanout (20 servers)
  - Larger CXL pods to pool mem effectively

Option 2: **Multi-Ported Devices (MPD)**



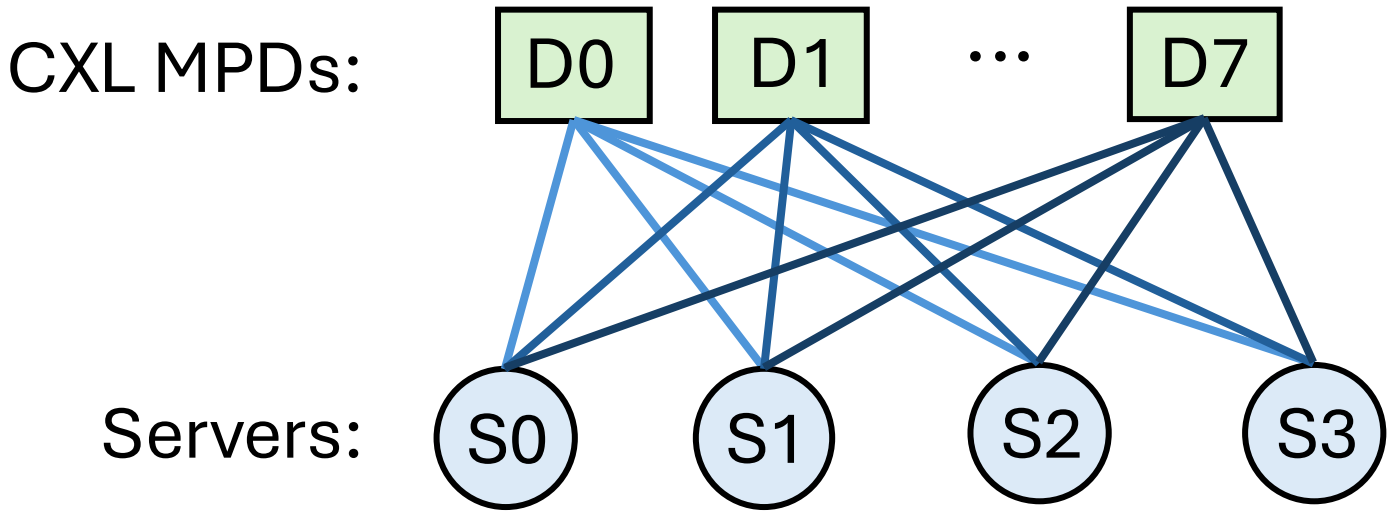
- ✓ Low latency (260-300 ns)
  - Pool up to 65% of total memory
- ✓ Cheap to deploy (no switch)
- ⚠ Low fanout (2-4 servers)
  - Smaller pods have limited pooling saving

Yang et al., Beluga: A CXL-Based Memory Architecture for Scalable and Efficient LLM KVCache Management, SIGMOD 2026
Li et al., Pond: CXL-Based Memory Pooling Systems for Cloud Platforms, ASPLOS 2023

Prior Work: Fully-Connected Servers and MPDs

Prior work: **fully-connected** CXL pods (every MPD connects to every server)

- CXL pod size = MPD port count (2-4 ports)



Each MPD connects to the **same** set of servers

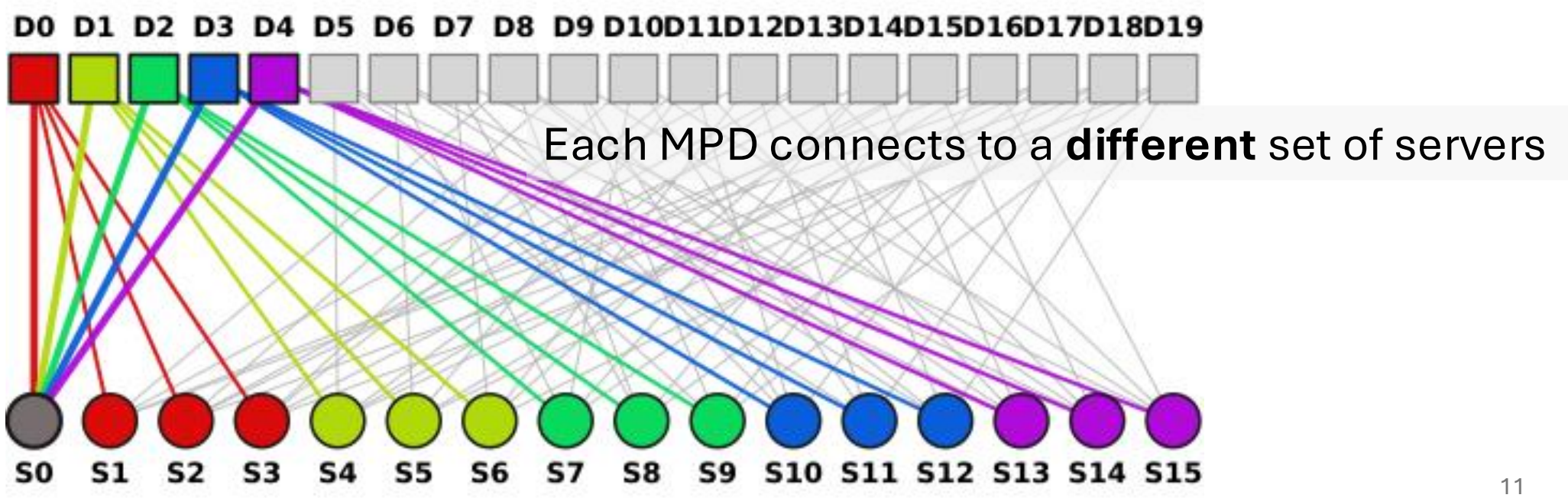
Li et al., Pond: CXL-Based Memory Pooling Systems for Cloud Platforms, ASPLOS 2023

Octopus: ***Sparsely*** Connect Servers and MPDs

Prior work: **fully-connected** CXL pods (every MPD connects to every server)

- CXL pod size = MPD port count (2-4 ports)

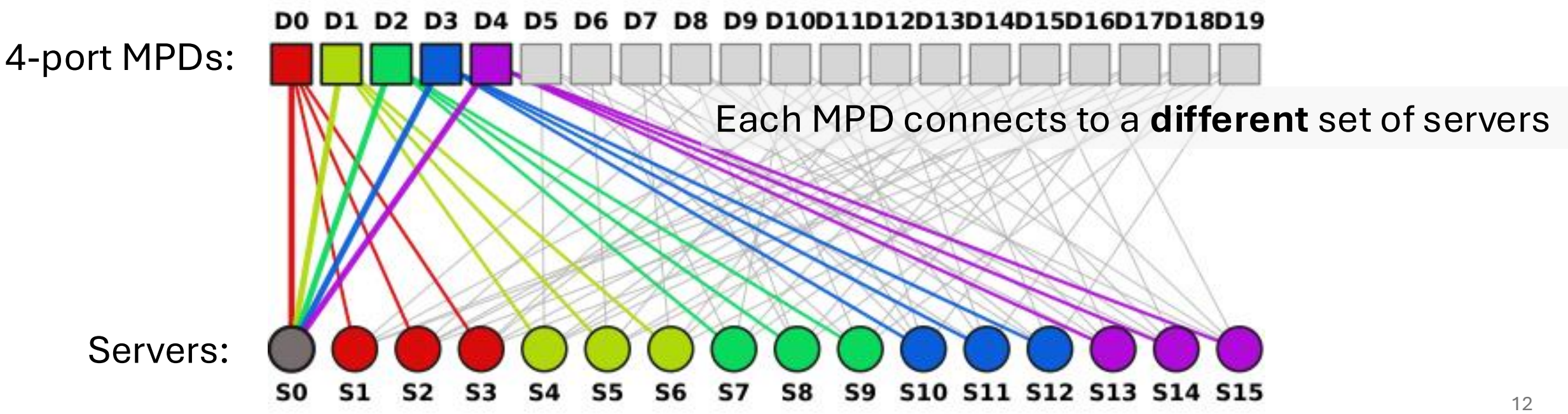
Key design: **sparsely** connect servers and MPDs to build **larger** CXL pods



Li et al., Pond: CXL-Based Memory Pooling Systems for Cloud Platforms, ASPLOS 2023

Octopus: ***Sparsely*** Connect Servers and MPDs

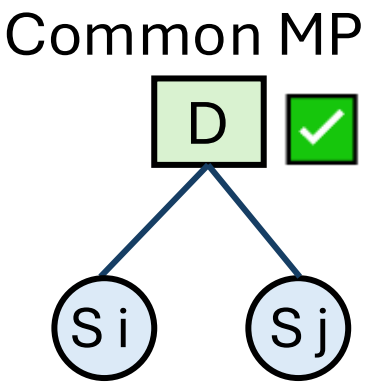
Key question: constructing **a sparse topology** between servers and MPDs to satisfy both memory pooling and low-latency communication



Li et al., Pond: CXL-Based Memory Pooling Systems for Cloud Platforms, ASPLOS 2023

Low-Latency Communication and Memory Pooling Are **Conflicting**

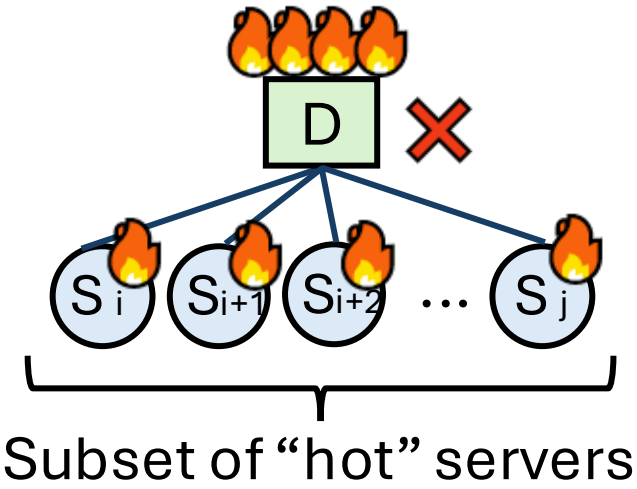
Low-latency communication:



Property: **Pairwise MPD overlap**

Optimal: Balanced Incomplete Block Design (BIBD)

Memory pooling:



Property: **Avoid MPD overlap** to spread load into as many MPDs as possible

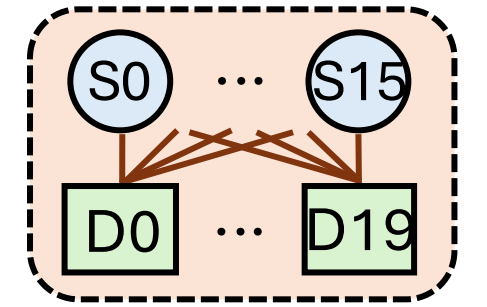
Optimal*: Expander graphs (e.g., random or Ramanujan graphs)

Borokhovich et al., Load-Optimal Local Fast Rerouting for Dense Networks
Singla et al., Jellyfish: Networking Data Centers Randomly

* Asymptotically optimal

Reconcile MPD Overlap With “Islands”

Insight: Low-latency communication is typically needed only at modest scale (e.g., ≤12 servers)

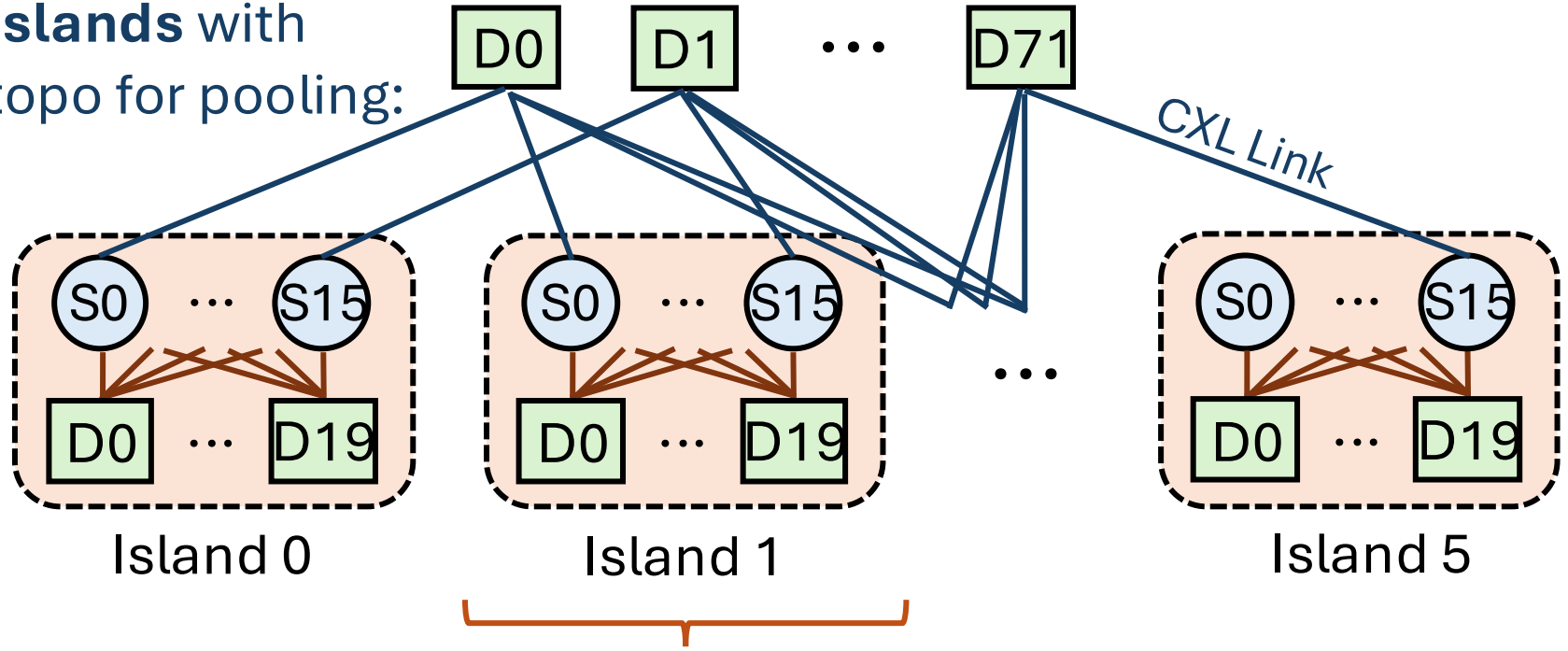


Ensure pairwise MPD overlap
within each island using BIBD

Reconcile MPD Overlap With “Islands”

Insight: Low-latency communication is typically needed only at modest scale (e.g., ≤12 servers)

Interconnect islands with expander-like topo for pooling:



Octopus Reconciles Pooling and Comm.

	Memory Pooling Saving	Communication Latency
Fully-connected (Pod size: 4 servers)	❌ Poor (limited pod size)	⚠️ Low among 4 servers
BIBD (Pod size: 25 servers)	❌ Poor (MPD overlap)	✅ Low among 25 servers
Expander (Pod size: 96 servers)	✅ Optimal	❌ High (multi-hop)
Octopus (Pod size: 96 servers)	✅ **Near-Optimal**	✅ **Low among 16 servers**

Evaluation

- **Can Octopus achieve significant savings by pooling memory?**
- Can Octopus provide memory-like communication latency within a pod?
- How do CXL link failures affect Octopus?

} See the paper

Octopus Reduces Server CapEx by Pooling

	CXL Switch	Fully-connected (MPD)	**Octopus** (MPD)
Pool-able Memory	35%	65%	65%
Saving Within Pooled Memory	44%	8%	25%
Total Memory Saving	**16%**	**5%**	**16%**
CXL Device Cost ($/server)	$3460	$1548	$1548
Server CapEx	**+3.3%**	**+2.5%**	**-3.0%**

Call for Research on CXL Pod Topologies

CXL expansion already deployed in production, pooling next to come

Many research questions in designing CXL pods:

- Can we borrow ideas from datacenter network topologies?
- Can we design sparse topologies for CXL switches?
- How to design CXL pods for different use cases (e.g., ring topologies for collective communications)?
- CXL 3.0 supports multi-level switching, how to design CXL pods?
- For smart/computational CXL devices, how to design CXL pods?

Microsoft Announcement: Azure delivers the first cloud VM with Intel Xeon 6 and CXL
Alibaba: Unlocking the Potential of CXL for Disaggregated Memory in Cloud-Native Databases
“Google has begun deploying CXL technology in its data centers” (from Digital Today)

Executive Summary: CXL as “Pod Area” Network

Background:

- Memory is expensive (~50% of CapEx) but underutilized due to stranding
- CXL enables a “pod” of servers to reach a common **memory pool**
- CXL also enables intra-pod **communication at memory-like latency**
- However, existing CXL pods are either **expensive** (using CXL switches) or **small** (using multi-port CXL devices, each with ≤ 4 ports)

Contributions:

- Octopus: a low-cost, large-scale CXL pod which **sparsely** connects servers with low-port CXL devices, without using costly switches
  - To balance conflicting demands of memory pooling and intra-pod communication, Octopus partitions a pod into “**islands**”
- Results: 16% memory saving, < 1 µs intra-pod communication

Paper