

Memory-Centric System Architecture

Pankaj Mehra
Founder



Elephance Memory



Outline:

1. Memory, a pre-virtualization resource
2. Devices don't disaggregate well
3. Introducing fluxy: needs VA, cross-device
4. Baby steps Special Theory of Data Gravity
 - Memory Objects
5. Principles of Memory as a Service
6. Anticipated Benefits
7. Charting a course

For presentation at
Salishan Conference
on Apr 25, 2023
Copyright © Pankaj
Mehra, except noted



Memory is the last remaining wild cat in the otherwise orderly world of data centers

Pankaj Mehra
IEEE Santa Clara Valley Chapter Industry Spotlight talk (2022)
<https://youtu.be/QxbeGBOoQXI>



The pain that is Data Center Server DRAM

DC BoM

The costliest
resource in DC

New Memory?

Decades to introduce.
Build then find out.

Short supply

Bit growth outpaced
by demand, LTAs.

Pre-virtualization

The last such
resource in SDDC

Hard to Sell

20-30% of memory
deployed but cannot be sold

Hard to Use

75% of the apps use <25%
of their provisioned memory



Cloud Markup on DRAM CapEx

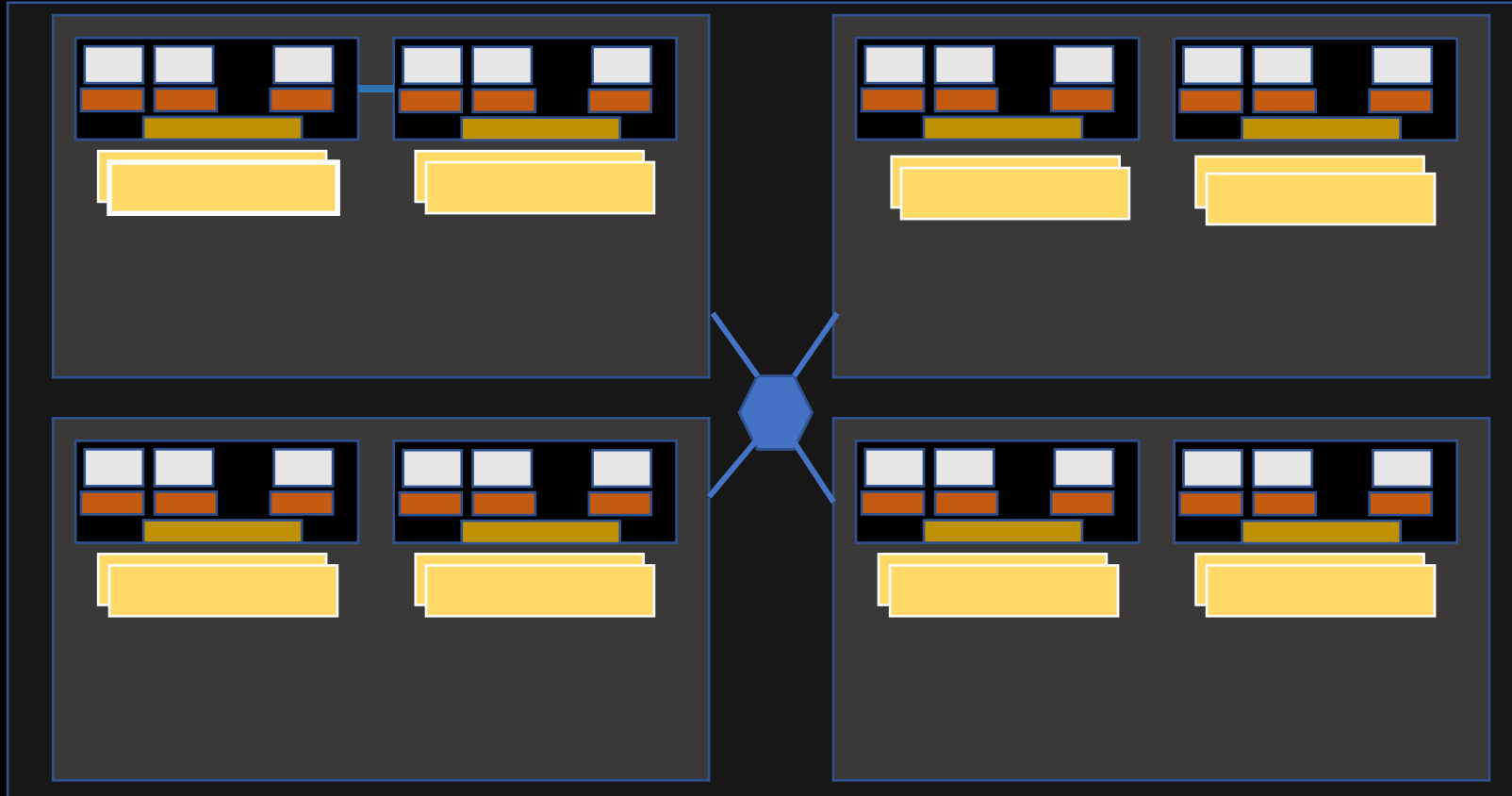
Sizing the Opportunity

AWS Pricing (\$/hr)	AWS Instance Type	DRAM (GB)	Instance Price (\$/yr)	Implied DRAM ARR \$/yr
\$0.09	On-demand C6i.large (compute-optimized)	4	\$ 745.11	
\$0.13	R6i.large (memory optimized)	16	\$ 1,104.52	\$ 29.95
		Other resources identical		Apt comparison between general purpose and memory optimized instance
		2 VCPUs		
\$0.17	x2gd.large (memory optimized)	32	\$ 1,463.92	\$ 22.46
				Apt comparison between two memory optimized instance (x86 vs Arm difference)



DDR: Current Practice in Memory Attach

RDMA: Current Practice in Remote Memory Access





Devices have a “Mother, may I?” relationship with the OS and will not disaggregate well.

Case in point:

- Staff of 5 in a 100K-server data center
- Threats:
Memory leaks? Unzeroed memory after Guest OS crash?
- Opportunities:
CXL 3. Data Gravity. Independently scale compute & mem



Despite rising costs, stranding. Despite ever increasing throughput, low goodput. Complex optimization.

Current Practice: Memory as a Device

Optimize for each Deployment

Vs Fungible Far Memory

- Peak allocation, Peak deployment
- High effort of porting and scaling
- High cost of scaling out for memory
- Inefficient scale-out communication

Mem Sold/Deployed (< 80%)

Mem Utilized/Deployed (< 66%)

GB-sized RPCs

Application agnostic (LD-ST)

Vs Infrastructure-Integrated

- Eagerly fetch
- More data than required
- More often than required

NW and Host Cache Pollution: 90+%

CPU LD-ST stall cycles: 30% (growing)

LD-ST optimizations lost: 97-99%

Location-based addressing

Vs Developer-friendly Managed Memory

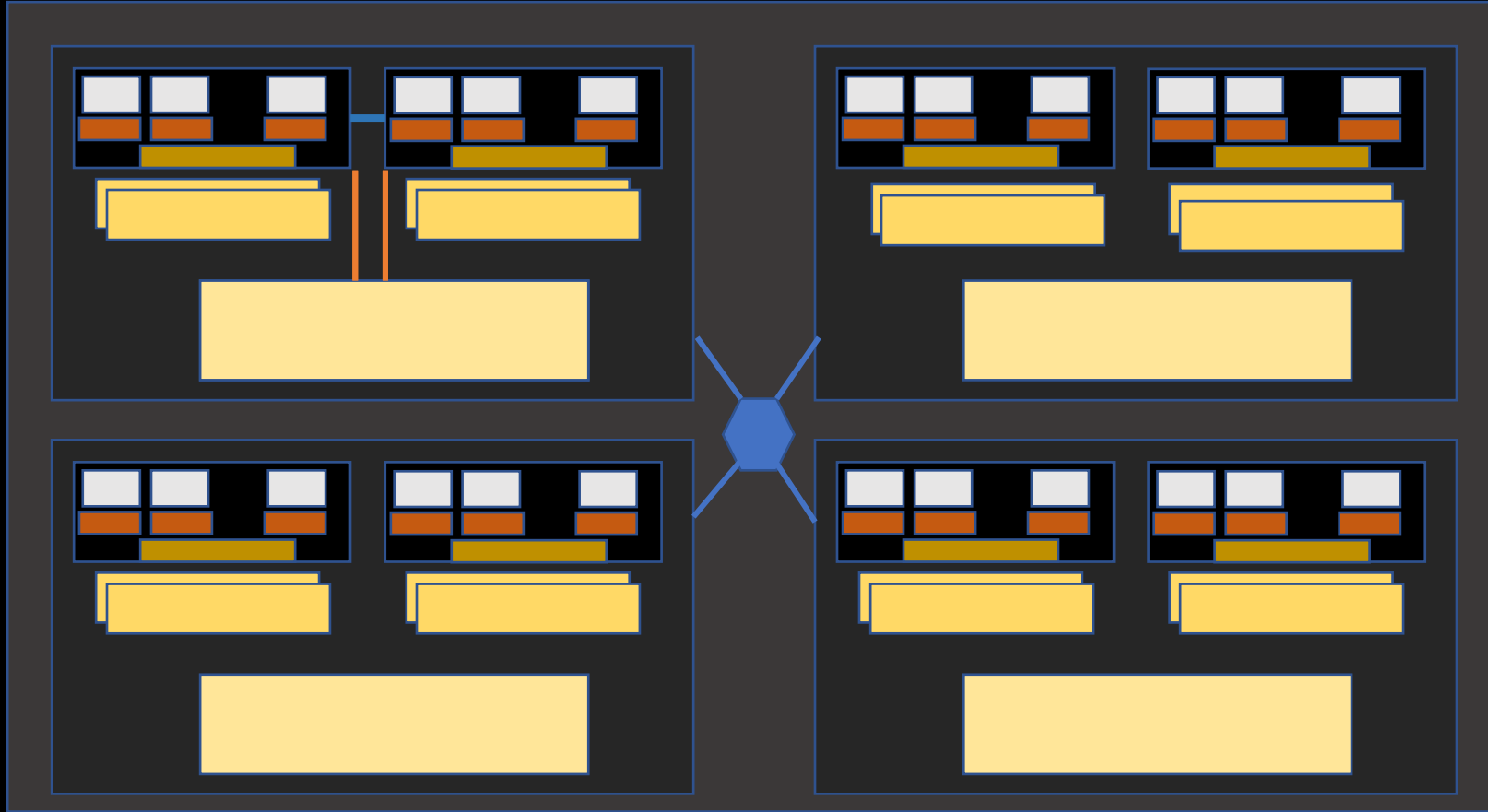
- Complex handling of pointers
- High latency to navigate pointers
- No opportunity to address data through (shipped) functions

Serialization-bound RPC BW: 0.11 GB/s

Pointer-chasing latency 200ns → 1+ μs

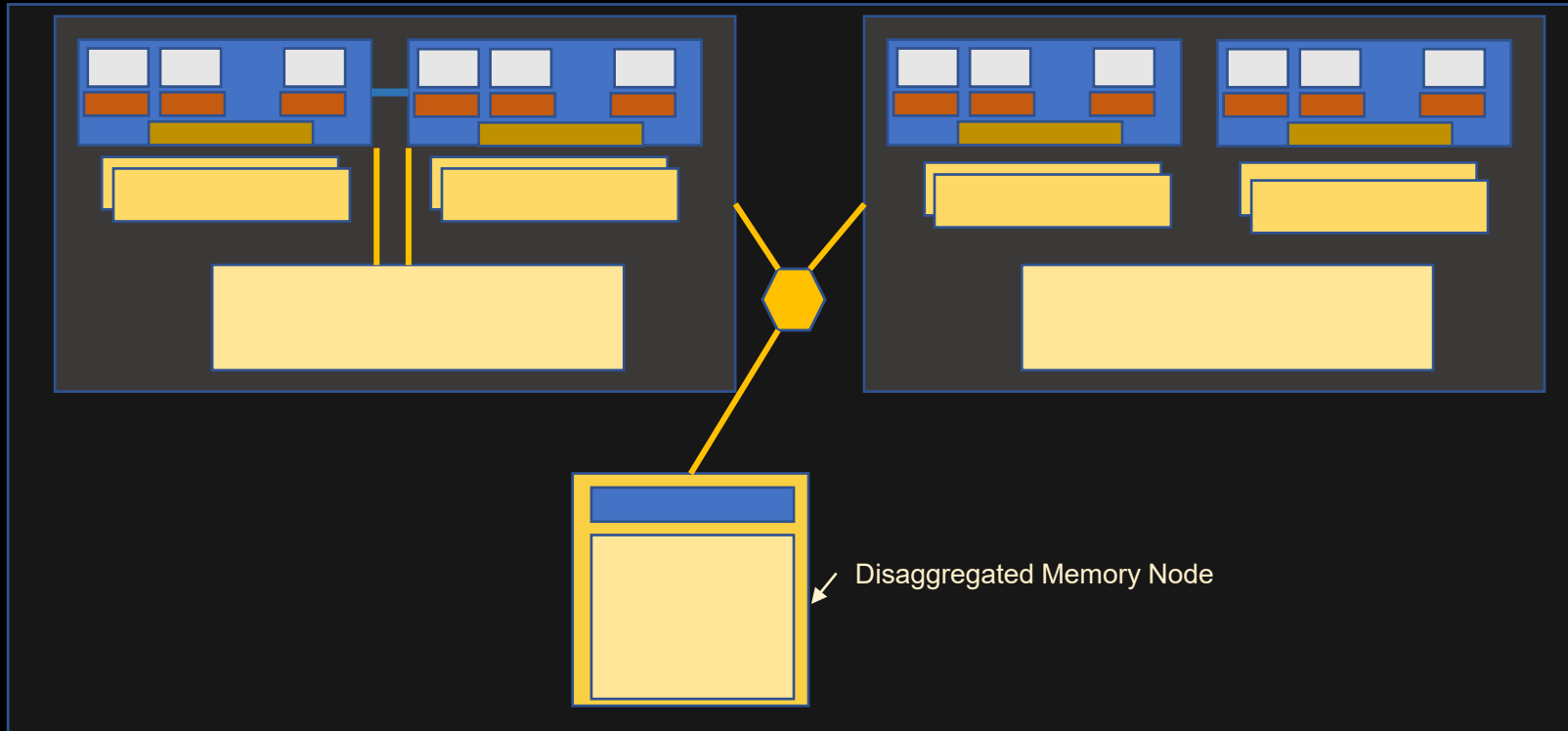
Graph load time/Inference = 70%

CXL 1: Memory Capacity Expansion over PCIe





CXL 3 enables pooling & disaggregation





Disaggregated Memory Nodes (DMNs)

Pros

- Memory deployment that avoids stranded resources
- Cost-effective expansion to petabyte scale memory pools
- Independent scaling of memory and compute capacity, BW
- CXL enables coherent accelerators near fabric memory

Cons

- New devices/APIs/protocols needed due to **higher latency**



SW Implications of Disaggregated Memory

Requirements

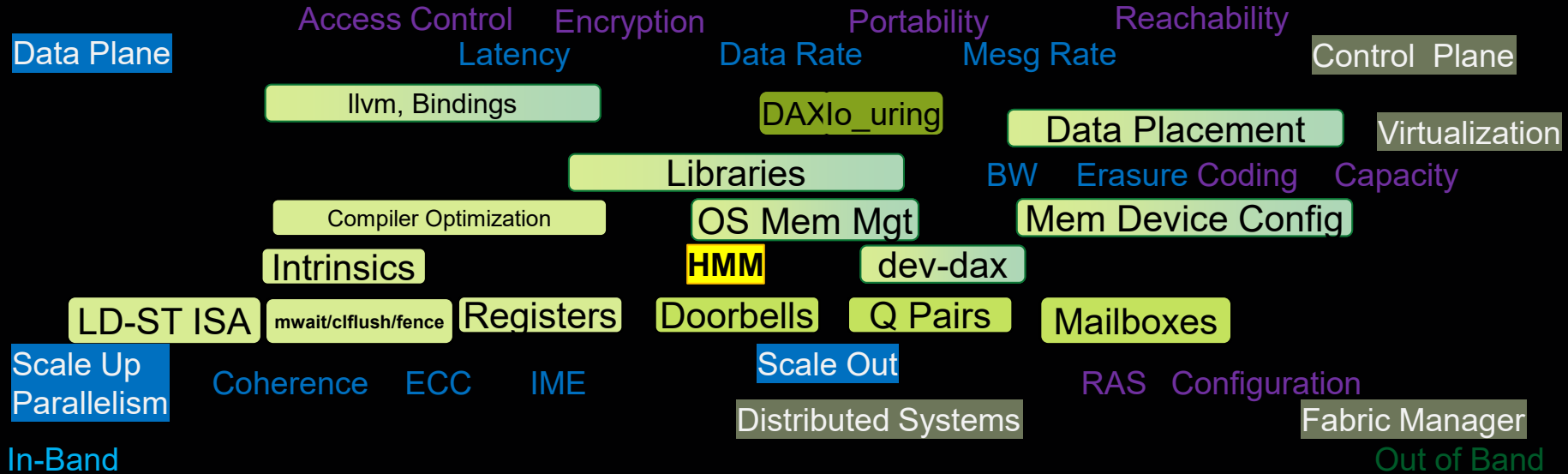
- Data at a Distance
- Accessed from >1 Servers
- Accessed from CPUs, GPUs, ...

Optimizations

- Data Gravity, Function Shipping
- Minimize/eliminate Serialization
- Maps efficiently into existing CPU Memory Management
- Maximize hardware data path

OS Evolution in response to CXL: No Perfect Answers

- Linux and the rest of the software ecosystem is co-evolving with CXL, esp. **HMM**
- Solutions need to span data and control planes, scale-up and scale-out
- Versus pmem, CXL has wider adoption and could see deeper investment in compilers and CPU-native optimizations to complement device-side features





software
performance

Memory as a Service

device-side
Elephance MemOS

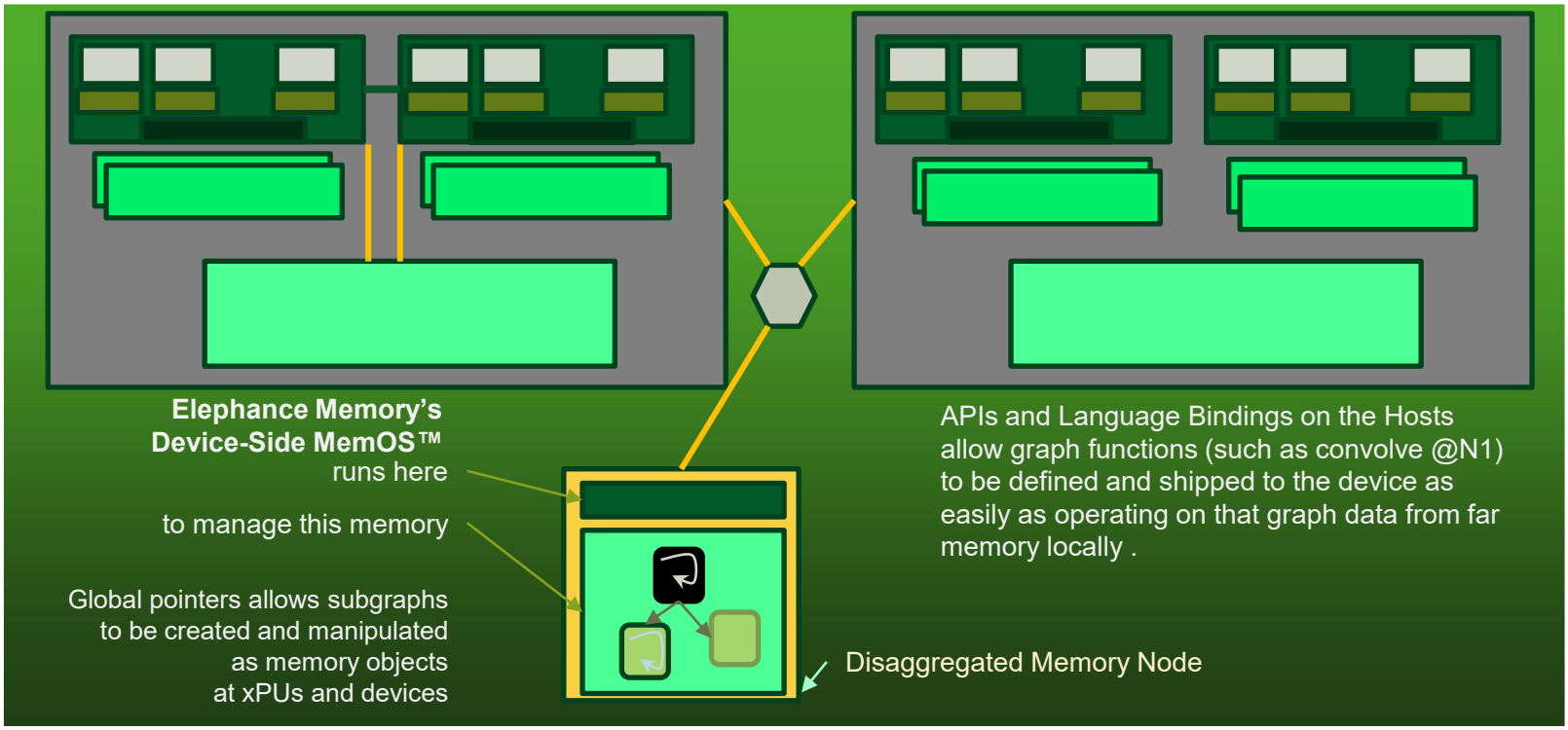
Memory Objects

security
disaggregated memory nodes

Elephance does for Memory
what S3 did for Storage



Elephance MemOS™ An Operating System for Disaggregated Memory Nodes



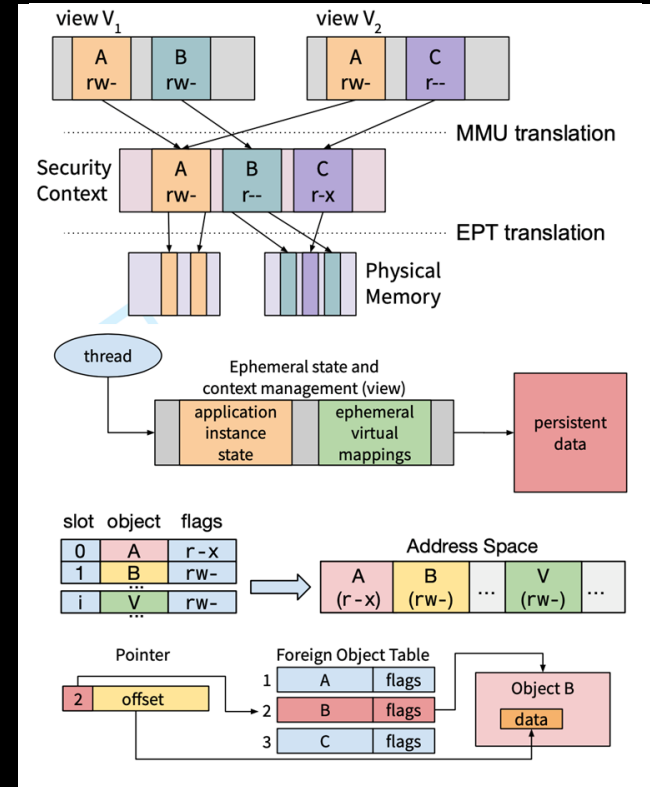
Invariant Pointers for Disaggregated Memory

Invariant under N-S or E-W data movement, meaningful rendezvous with function shipped to device



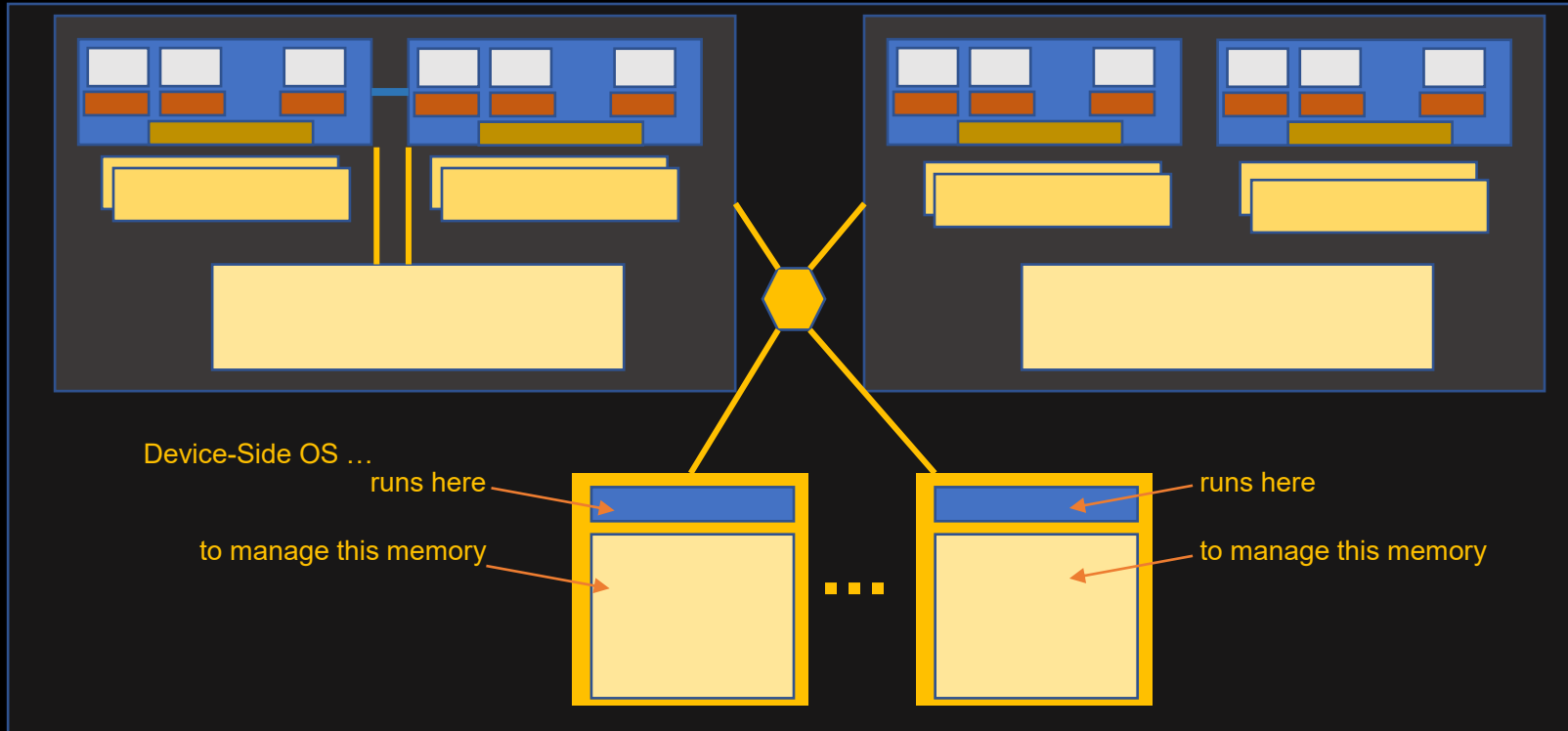
MemOS under the hood

- Translation context disaggregated from process, moved into Memory Objects with 128b ids that are themselves easily embedded inside pointers that traverse very large address spaces
- Efficient representation and fast manipulation of local pointers (intra-object) in 1s nanoseconds
- Scalable persistent representation and fast manipulation of non-local pointers (linked data) using a Foreign Object Table for indirection in every object within 10s nanoseconds



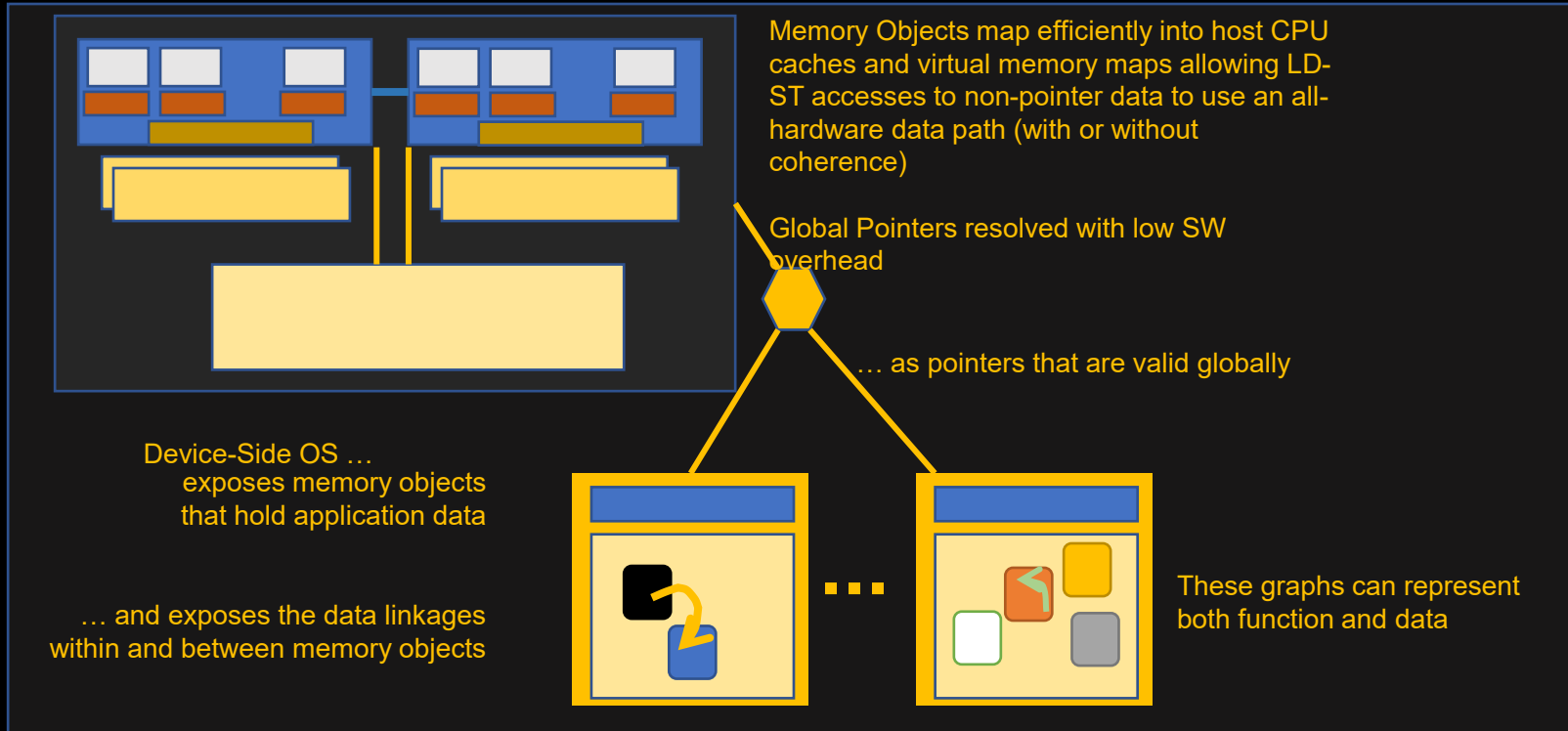


Memory as a Distributed Service from Disaggregated Memory Nodes





Far Memory as a Distributed Service from Disaggregated Memory Node

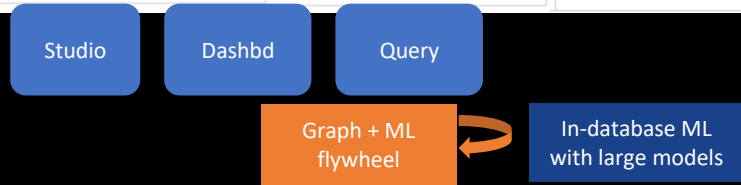


Workloads: Recommenders, GraphDB, and ML



Healthcare, Retail and Banking build Enterprise Knowledge Graphs to power real-time context-mediated experiences

- EKG scales: 10^9 vertices • 10^{10} edges • 10^7 updates/d • 60s CDC • 25K concurrent users • 100ms response
- As little as 0.1% gain in ML model accuracy (e.g. CTR prediction) materially adds to corporate revenue



EKG: Enterprise Knowledge Graph. CTR: Clickthrough Rate. ESG: Env Sustainability Governance. SCM: Supply Chain Mgmt. HLS: Health & Life Science. FSI: Financial Services Industry. CDC: Changed Data Capture.

ISO standard GQL queries
36TB LDBC Benchmark 
FPGA accelerated pointer chasing

An illustrative interface for Disaggregated Memory Nodes!

Graph Storage and Processing Engines:
Data in memory for real-time experience and monitoring

Modern Workloads



Richly linked data structures widely used in modern applications, increasingly so



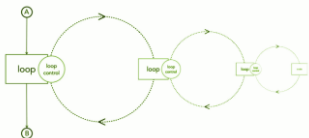
Chasing pointers is very inefficient in general-purpose processors, even worse when memory is farther



Offloading pointer-chasing to near-memory processors brings its own set of challenges such as limited parallelism and virtual-to-physical translation

When graph-structured functions work on graph data

Loopy



In DOE mini-apps,

FP instructions comprise 9-28%
(mean 15.5%)

of all executed instructions

Branchy



Branch and control instructions
(mean: 9.5%)

Fluxy



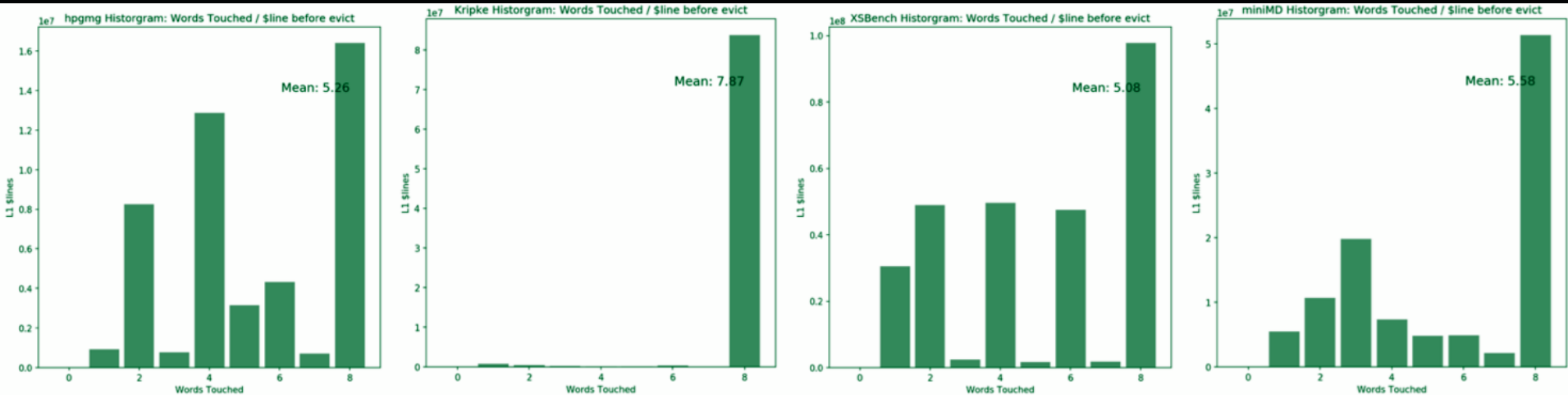
integer instructions (mean 29.5%)
and memory instructions (45%)

majority of integer instructions are
used to calculate memory addresses

Prior DOE Research: ***What HPC programs really “do” is not floating-point, but memory***

Fluxy is no friend of CPUs and GPUs

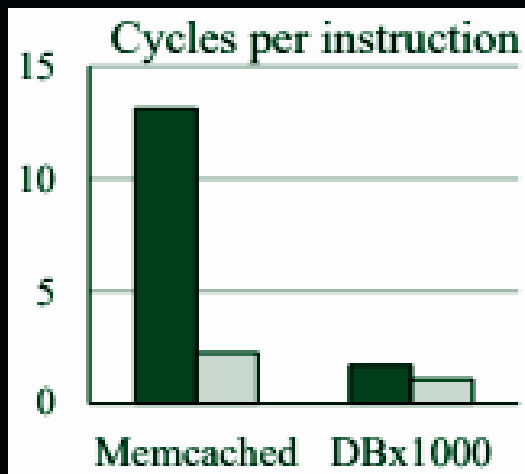
DOE: Caches, our primary architectural mechanism to reduce latency and improve effective bandwidth, are not well utilized. Many (64b) words which are brought in to the L1 cache are never touched before the cacheline is evicted. In three of the four applications, **for every cacheline of eight words that is brought in to the L1, less than 5.6 words (mean) were used before the line was evicted.** Worse results when prefetching is on.



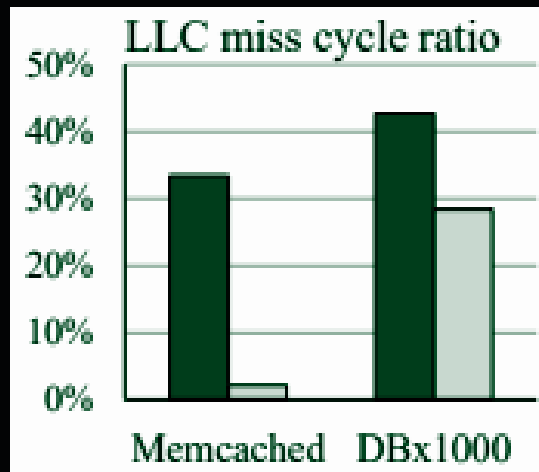
Prior DOE Research: Just as it is reasonable to say that HPC programs really “do” memory, it is also reasonable to say that we do it somewhat poorly.

Pointer-Heavy Code Sections in Analytics Workload

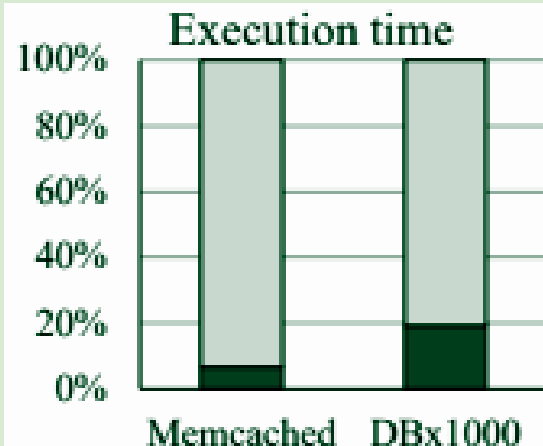
Eat up cycles



Mess up caches



Worth optimizing



■ Pointer Chasing ■ Rest of the Application

Such large-impact easy-to-offload operations represent an uncommon optimization opportunity

https://users.ece.cmu.edu/~omutlu/pub/in-memory-pointer-chasing-accelerator_iccd16.pdf

Here is the pattern



GATHER

COMPUTE

SCATTER



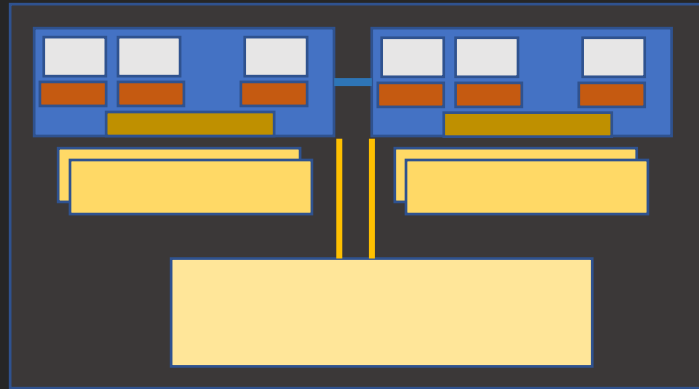
Computational Memory as a Service

Key challenges of offloading Fluxy to Devices:

1. Working with Virtual Addresses
2. Executing offloaded operation across many devices

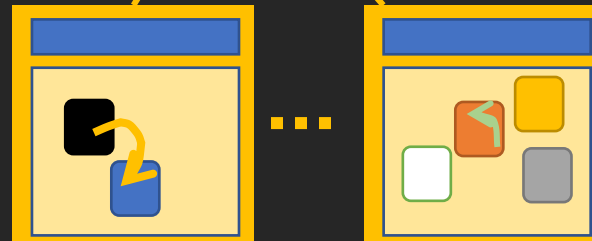


Computational Far Memory as a Service from Disaggregated Memory Node



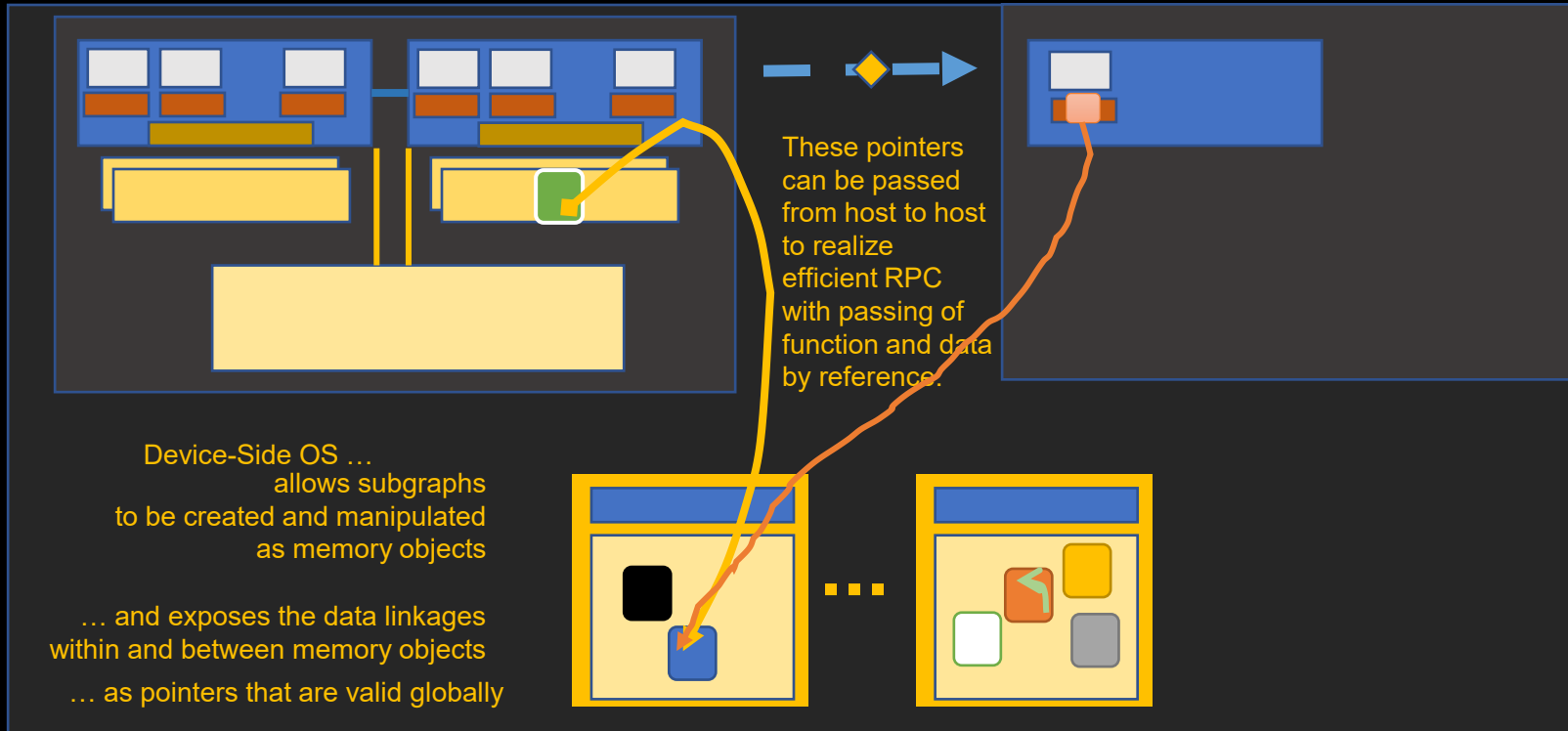
APIs and Language Bindings on the Hosts allow graph functions (such as convolve @N1) to be defined and shipped to the device as easily as operating on that graph data from far memory locally .

Device-Side OS ...
allows subgraphs
to be created and manipulated
as memory objects





Global Memory References as a Service from Disaggregated Memory Node

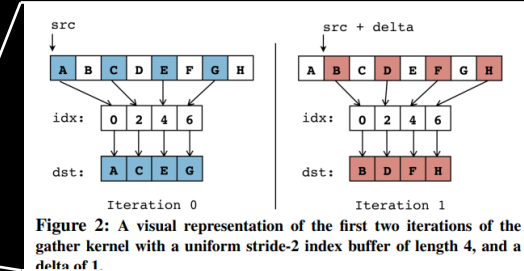




MemOS enables Computational Memory

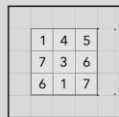
- Low overhead, low coordination global address space via invariant pointers
 - Enables data sharing and removes serialization
 - Increases data and compute placement flexibility
- Invariant pointers aid pushing view of data graph lower in the stack
 - Allows more effective offloading of FLUXY application-level functions
 - Improves infrastructure-level software's ability to optimize and to query plan
- Case in point: **Densification for GPUs**

Rich Data Use Case	Device-side Memory Controller	Elephance MemOS™ Software
SQL/NoSQL	Cores and Acceleration IPs	Stats offload with debug/tracing; Computation graphs
DLRMs, GraphDB, HPC	On-device MMU	Data graphs; pointer chasing ; Flexible resolvers
Shared Data Clustering	.mem MH Data Coherence	Low overhead global addressing; Transactions
μServices and RPC	.cache 1-writer coherence	Pipelining; Serialization avoidance

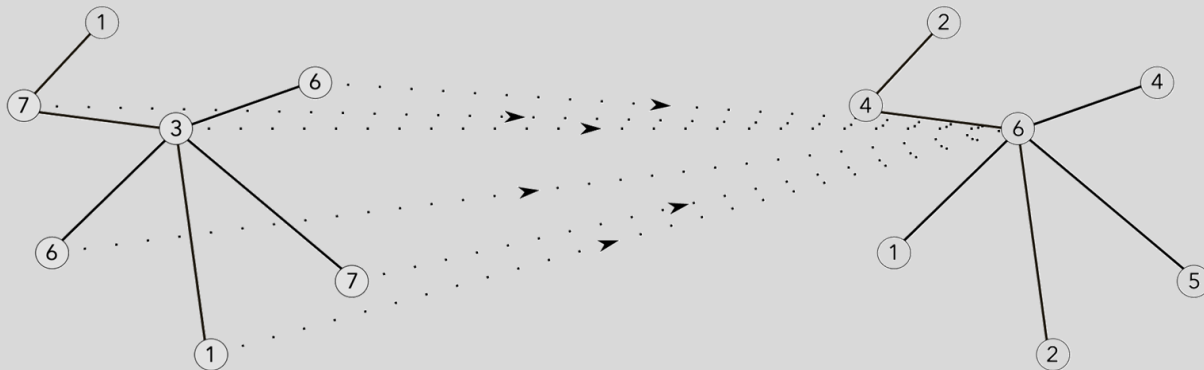
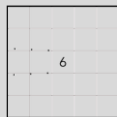




Offloading FLUXY: Local Convolution



Convolution in CNN



Localized Convolution in GNNs



CPU chases pointers,
Retrieves neighbor node,
Retrieves local property,
Calculates filter polynomial

Many RTTs,
Low goodput,
Cache pollution,
NW flooding,
Rule of 3 penalty

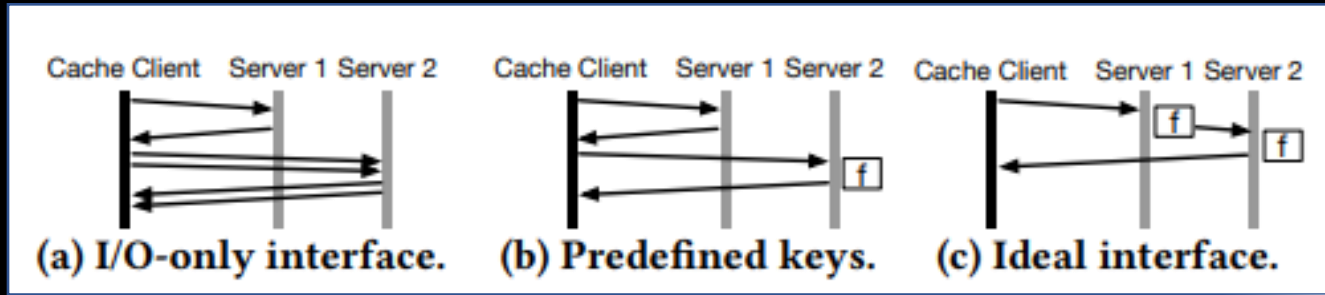


DMN chases pointers,
Retrieves neighbor node,
Retrieves local property,
Calculates filter polynomial
Returns convolved values

Low latency,
High goodput,
Independent
scaling



MemOS offloads Cross-Device/Object Pointer Chasing

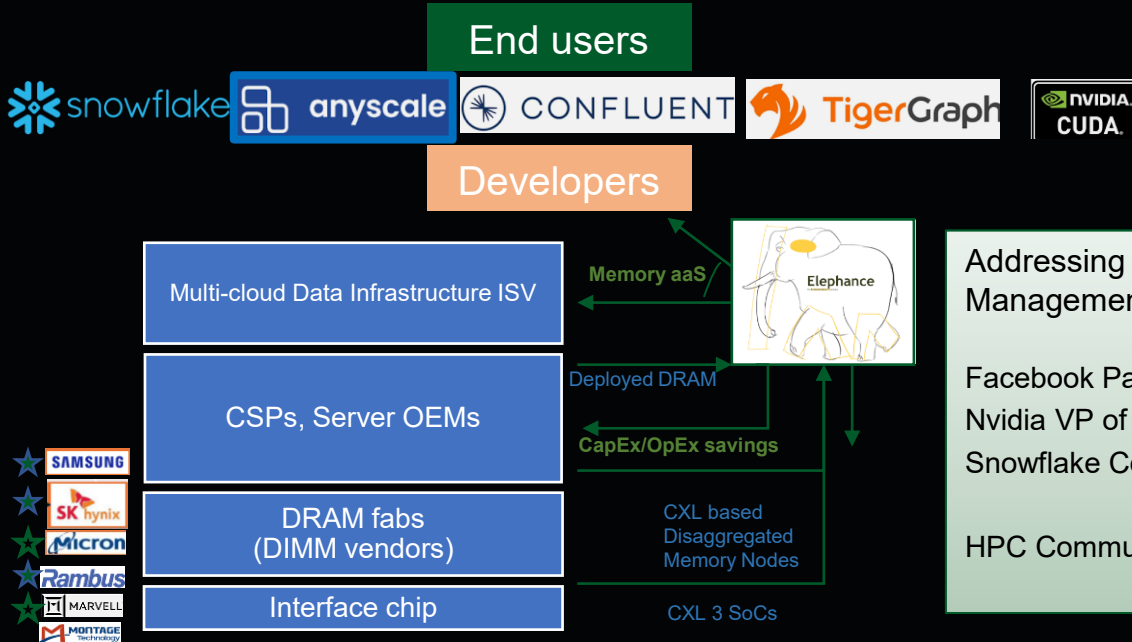


Especially useful for neighborhood operations on graphs that have been sharded for scale across compute nodes such as search-accumulate, PageRank, convolutions

Phil Bernstein, Microsoft Research

Rich Data Use Case	Device-Side Control	
SQL/NoSQL	Cores and Acceleration IPs	
DLRMs, GraphDB, HPC	On-device MMU	Data graphs; pointer chasing ; Flexible resolvers
Shared Data Clustering	.mem MH Data Coherence	Low overhead global addressing; Transactions
μServices and RPC	.cache 1-writer coherence	Pipelining; Serialization avoidance

Deep Ecosystem Engagement. High Initial Customer Interest



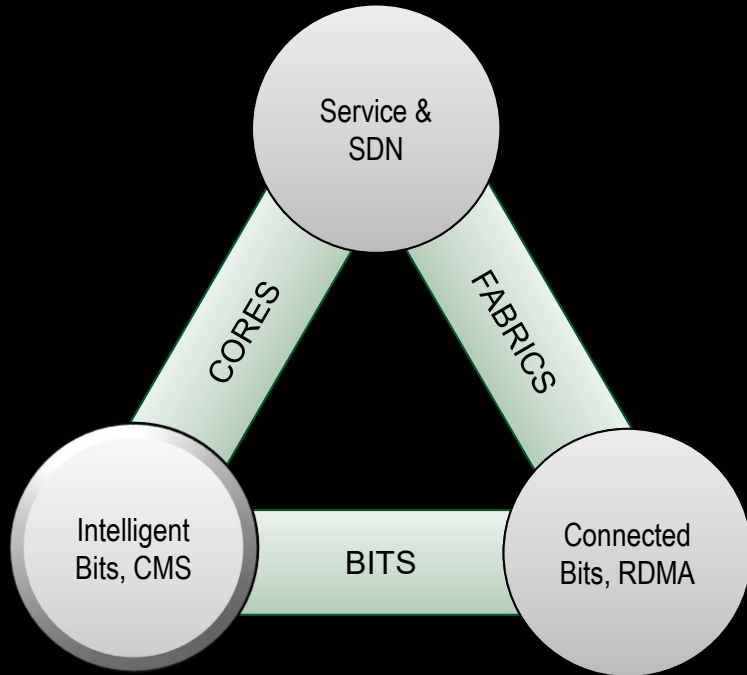
Addressing the **4-27B TAM** market of memory in Data Management servers with a win-win-win value proposition.

Facebook Payments Developer: *"I would use it right now"*
 Nvidia VP of CUDA: *"Your software will make GPUs more efficient"*
 Snowflake Co-founder: *"We will use it when available in AWS"*

HPC Community: Our first broad exposure here

From Devices to Services

Exploiting Integration, HI



Computational Memory. Memory as a Service

Services	Information	Cloud
μ Services	Objects	Protocols
APIs	Metadata	Topologies
Software	Data	Routes
OS	Metabits	End points
Cores	Bits	Fabrics
ALUs	Caches	I/O Ports
Firmware	Data Paths	Switches
control	state	flow

With the memory industry and hyperscalers:

- How much near-memory processing?
- Fixed function vs programmable esp. with VAs
- Data-path: More efficient promotion/demotion
- Control: Better use of finite MMU resources
- Low-overhead hotness tracking

Feature Summary: How Elephance enables Memory aaS

1. High Performance Flexible Remoteable Pointers

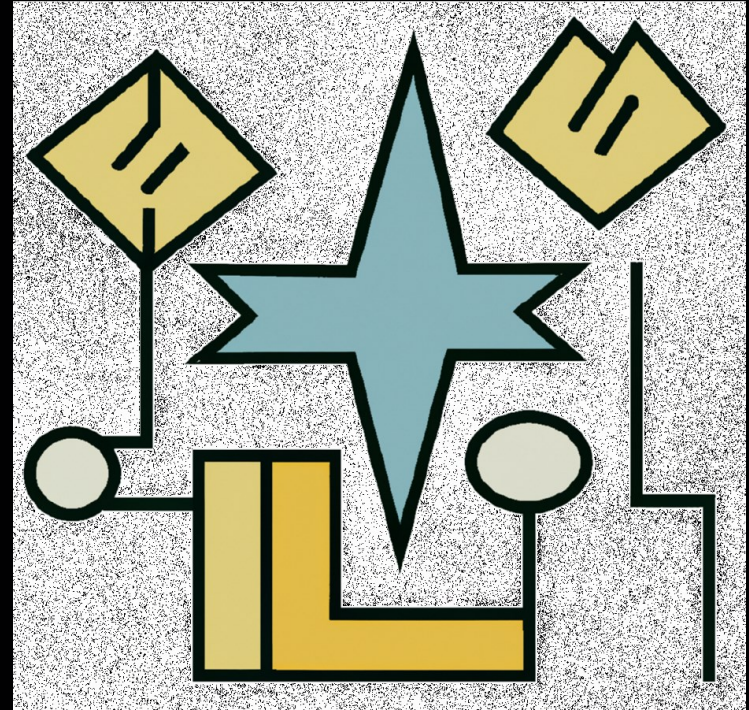
- Native 64b pointers
- Minimize swizzling overheads
- Valid everywhere in space and time

2. Easy to Flex Compute Up and Down

- New compute can just declare and use
- Minimize [de]serialization overheads
- No need to keep compute fired up

3. μ Services point to data in DMN, not copy it around

- Existing microservices benefit right away
- Minimize/Eliminate unnecessary copying, resulting cache pollution, BW abuse
- Minimize [de]serialization overheads



Next weekly Mar 13 1pm in E2-399 and on

Zoom

☰ UC SANTA CRUZ CXL SIG led by Pankaj Mehra

discuss trends, problems, and opportunities arising from CXL-enabled disaggregation in the data-center. Our |
leaders covering CXL-related topics spanning low-level cache and micro-architecture concerns up to sy:
applications and algorithms. Panelists included (click on the panelist's name to see slides where available):

- [Pankaj Mehra](#), *Elephance Memory & UC Santa Cruz (moderating)*
- Suresh Mahanty, *SK hynix*
- KC Ryoo, *Samsung*
- Craig Hampel, *Rambus*
- [Frank Hady](#), *Intel*
- [Nathan Kalyanasundharam](#), *AMD*
- Priya Duraisamy, *Google*
- [Phil Bernstein](#), *Microsoft*
- Daniel Bittman, *UC Santa Cruz*
- Sharad Singhal, *HPE*
- Ike Nassi, *TidalScale & UC Santa Cruz*
- Andrew Quinn, *UC Santa Cruz*

The panel focused on CXL-attached memory as opposed to CXL-attached accelerators. CXL-attached memory
of challenges: It increases DRAM's memory path latency compared to traditional interfaces (e.g., DDR), exp:
the memory hierarchy by exposing a wider array of non-uniform memory accesses, and imposes complex s
responsibilities on devices. In light of these challenges, the panel focused on two broad topics. First, what
bring that incentivize working the use of CXL in spite of these challenges? Second, what techniques can



Thank You

pankaj.mehra@ElephanceMemory.com



“Virtual Memory is one of the pillars of the computing revolution. Its benefits include hardware flexibility, software portability, and overall better security. Many of its fundamental abstractions and implementation approaches are being augmented, extended, or entirely rebuilt in order to ensure that virtual memory remains viable and performant in the years to come.” [1]

Elephance Memory’s new MemOS operating system takes the idea of virtual memory to its logical conclusion and makes possible *Memory as a Service*, unleashing powerful new benefits such as Global Pointers, Independent Scaling of Memory and Compute, RPC by Reference, and Near-Memory Processing.

MemOS does all this while operating at the speed of hardware where memory access/coherence protocols and safe execution are concerned.

