MARVELL

Enabling Memory Tiering in ASTRA-sim with Extended Chakra Traces

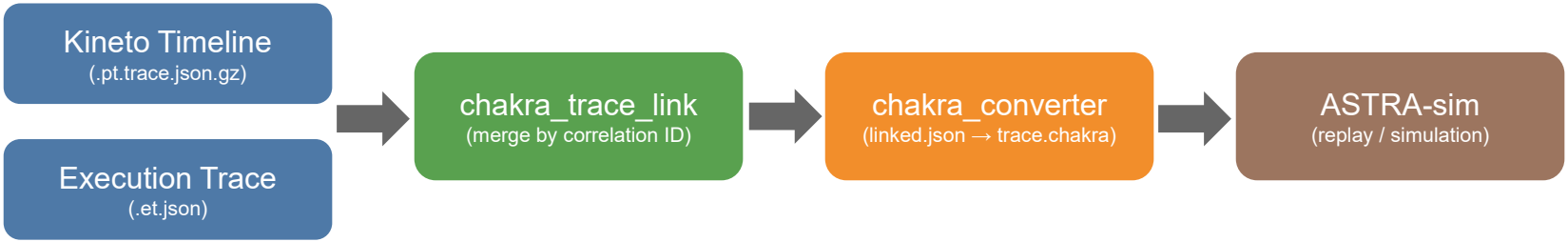
AstraSim Tutorial - ISCA 2026 Conference

Ulf Hanebutte, Nikhil Bernard John Stephen, Shuting Du*
Marvell Technology

June 27, 2026

*Purdue Graduate Student

Background: Chakra Execution Trace Pipeline

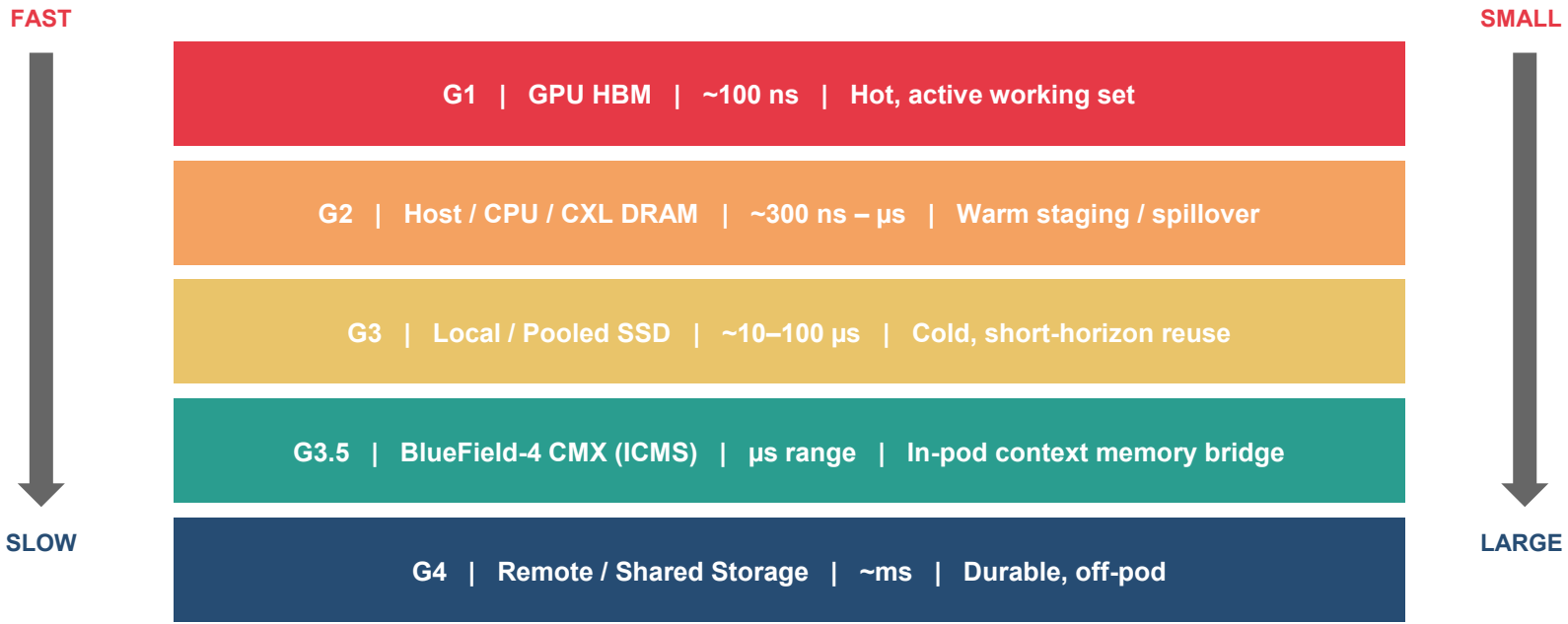


Chakra defines 3 node types:

COMPUTE **MEMORY** **COMMS**

This proposal: extend MEMORY with rich attributes (chakra.mem.* namespace)
— zero mandatory schema changes, fully backward-compatible

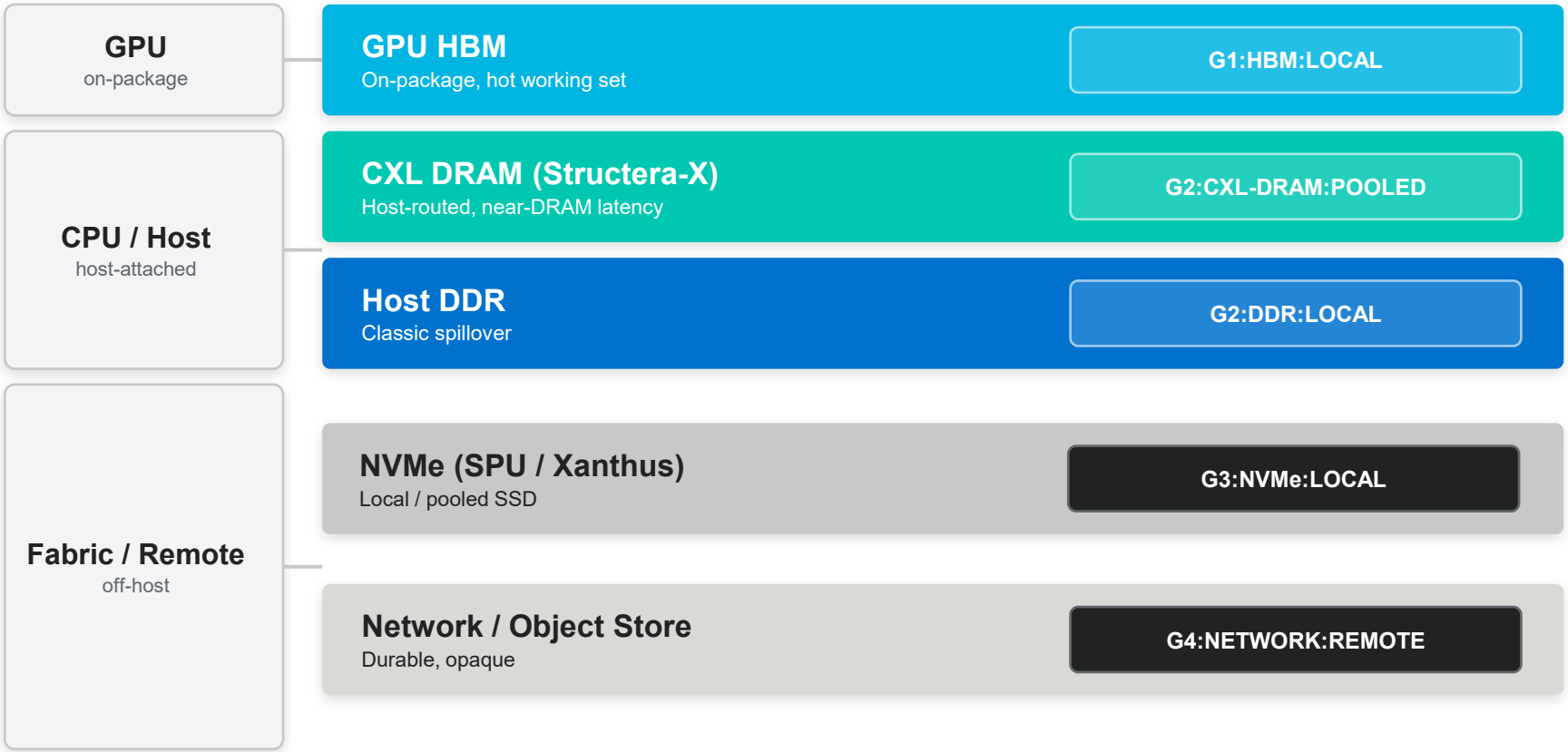
Memory Tier Hierarchy (NVIDIA G-Tiers)



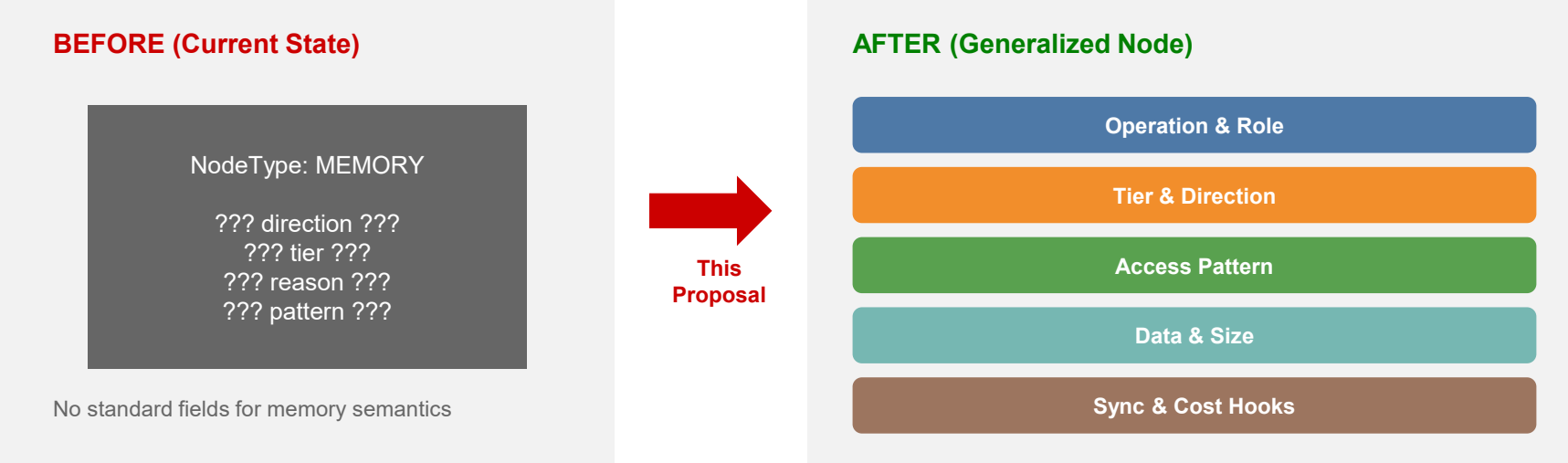
Source: NVIDIA Dynamo KVBM / BlueField-4 CMX blog (developer.nvidia.com)

Our extension: tier encoding = <tier_class>:<tier_medium>:<tier_role>
e.g., G2:CXL-DRAM:POOLED — makes CXL explicit, never hidden behind a bare letter

Memory Tier Hierarchy

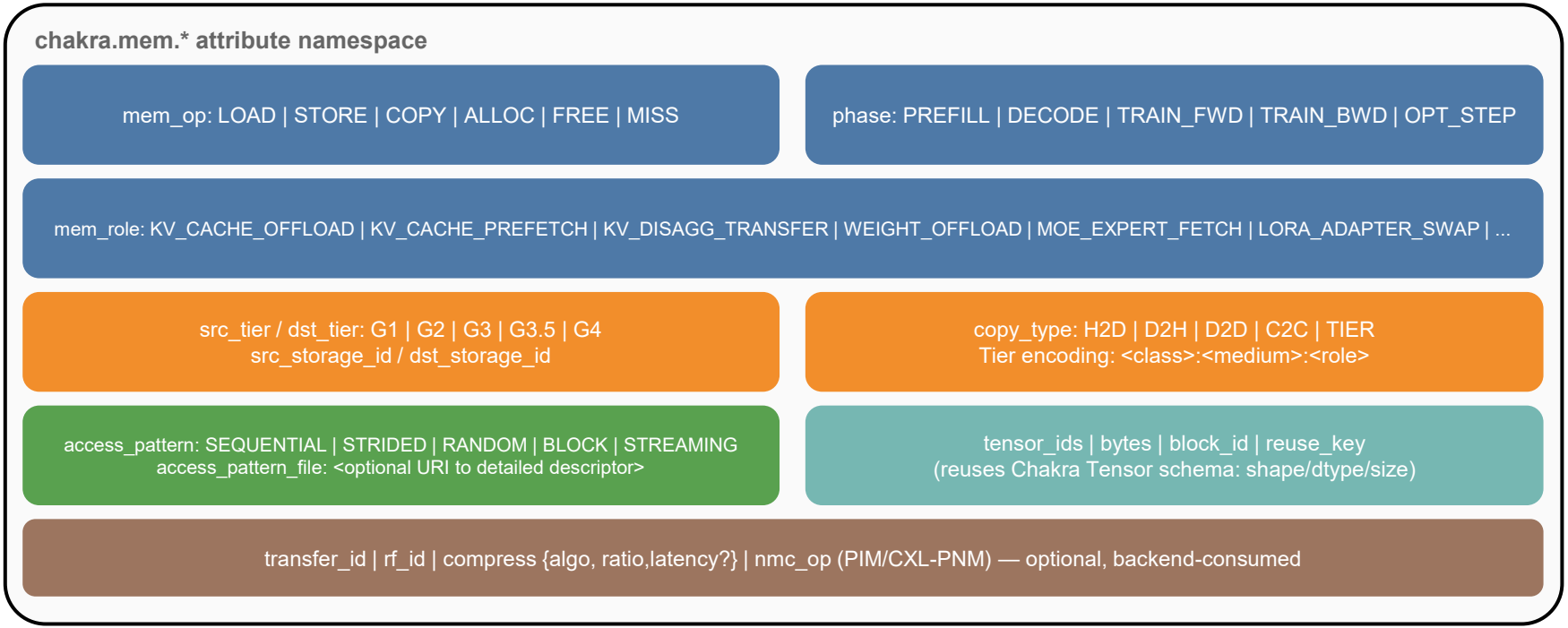


Motivation: Why Generalize the Memory Node?



Design principle: Use existing NodeType.MEMORY + chakra.mem.* attribute namespace
→ Zero mandatory schema changes | Fully backward-compatible | Upstreamable to MLCommons

The Generalized Memory Node — Structure



Field Detail: mem_op & mem_role

mem_op (required)

LOAD	Bring data from remote tier into working tier
STORE	Write/evict from working tier to another tier
COPY	Tier-to-tier movement (e.g., P/D KV transfer)
ALLOC	Reserve capacity in a tier
FREE	Release capacity in a tier
MISS	Object not resident; triggers dependent LOAD

mem_role (required) — covers data type + reason

KV Cache	KV_CACHE_OFFLOAD | KV_CACHE_PREFETCH | KV_CACHE_DISAGG_TRANSFER
Weights	WEIGHT_OFFLOAD | WEIGHT_PREFETCH | MOE_EXPERT_FETCH
Adapters	LORA_ADAPTER_SWAP
Activations	ACTIVATION_CHECKPOINT | ACTIVATION_OFFLOAD
Training	GRADIENT_OFFLOAD | OPTIMIZER_OFFLOAD
Other	EMBEDDING_LOOKUP | CAPACITY_EVICTION | OTHER

Tier Encoding & Direction of Movement

Storage.device encoding format:

Direction: src_tier → dst_tier (explicit on every movement node)

Physical Tier Mapping:

Physical Tier	Encoded Label	Note
GPU HBM	G1:HBM:LOCAL	On-package, hot working set
CXL DRAM (Structera-X)	G2:CXL-DRAM:POOLED	Host-routed, near-DRAM latency
Host DDR	G2:DDR:LOCAL	Classic spillover
NVMe (SPU/Xanthus)	G3:NVMe:LOCAL	Local/pooled SSD
Network / Object Store	G4:NETWORK:REMOTE	Durable, opaque

Memory Access Patterns

SEQUENTIAL	STRIDED	RANDOM	BLOCK	STREAMING
Linear, contiguous addresses	Fixed stride between accesses	Non-predictable access locations	Contiguous chunks (KV blocks)	Continuous flow, no reuse

access_pattern_file: optional URI to detailed pattern descriptor

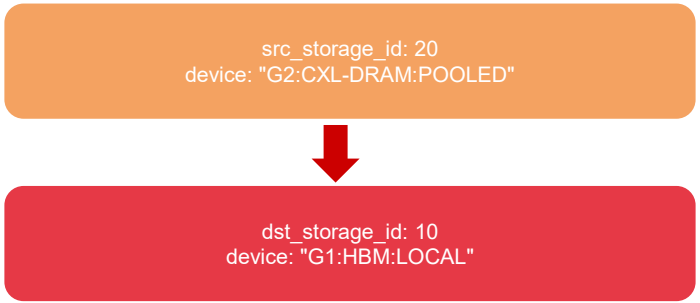
Load/Store Pattern	Packet Pattern	Spatial Pattern	Temporal Pattern
Stride lengths Tile dimensions Burst sizes Reuse distance	Packet sizes Inter-arrival dist. Batching factor Protocol overhead	Cache lines touched Block heatmap Locality metrics Working set size	Access frequency Periodicity Lifetime of blocks Prefetch hints

→ **Enables: contention modeling, prefetch analysis, BW utilization, capacity planning**

Complete Node Example: KV Cache Load (All Fields)

```
Node {
  id: 42
  name: "mem_kv_load_layer12"
  type: MEMORY

  // ── Operation & Role ──
  attr "chakra.mem.mem_op":      LOAD
  attr "chakra.mem.mem_role":    KV_CACHE_PREFETCH
  attr "chakra.mem.phase":       DECODE

  // ── Tier & Direction ──
  attr "chakra.mem.src_tier":    G2
  attr "chakra.mem.dst_tier":    G1
  attr "chakra.mem.src_storage_id": 20
  attr "chakra.mem.dst_storage_id": 10
  attr "chakra.mem.copy_type":   H2D

  // ── Access Pattern ──
  attr "chakra.mem.access_pattern": BLOCK
  attr "chakra.mem.access_pattern_file":
       "patterns/kv_block_512.json"

  // ── Data & Size ──
  attr "chakra.mem.tensor_ids":  [100]
  attr "chakra.mem.bytes":       2097152
  attr "chakra.mem.block_id":    512
  attr "chakra.mem.reuse_key":   "0xABCD..."

  // ── Sync & Cost ──
  attr "chakra.mem.transfer_id": 7
  attr "chakra.mem.rf_id":       101
  attr "chakra.mem.compress":    {algo:LZ4,ratio:0.6}

  // ── Standard Chakra fields ──
  ctrl_deps: [41]  // depends on MISS node
  duration_micros: 85
}
```

How src/dst tier works:

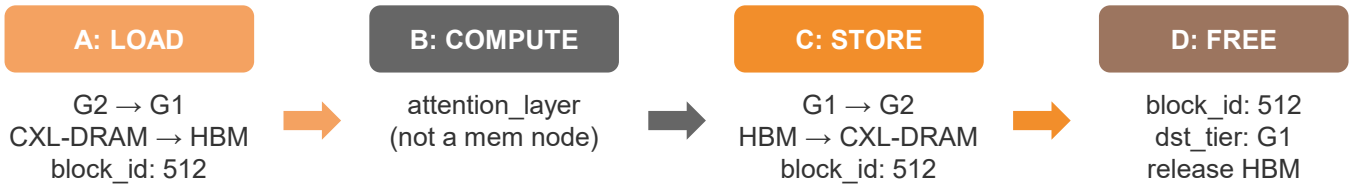


Per operation type:

LOAD:	src=remote, dst=local (G1)
STORE:	src=local (G1), dst=remote
COPY:	src=any tier, dst=any tier
ALLOC/FREE:	only dst_tier (the tier affected)
MISS:	only dst_tier (where miss occurred)

src_tier/dst_tier = shorthand
(tools that don't resolve Storage
can still route by tier class)

Worked Example: KV Tier Miss



Context: KV block for layer L lives in CXL DRAM (G2), not resident in HBM (G1)

```
storage 10: { device: "G1:HBM:LOCAL" }
storage 20: { device: "G2:CXL-DRAM:POOLED" }
```

```
tensor 100: { storage_id: 20, dtype: fp16,
              size_bytes: 2097152 }  // in G2
```

→ Exercises: tier-miss → tier-fetch → compute → write-back → free
All on standard Chakra MEMORY node via attributes. Lowerable to both ASTRA-sim 2.0 & 3.0.

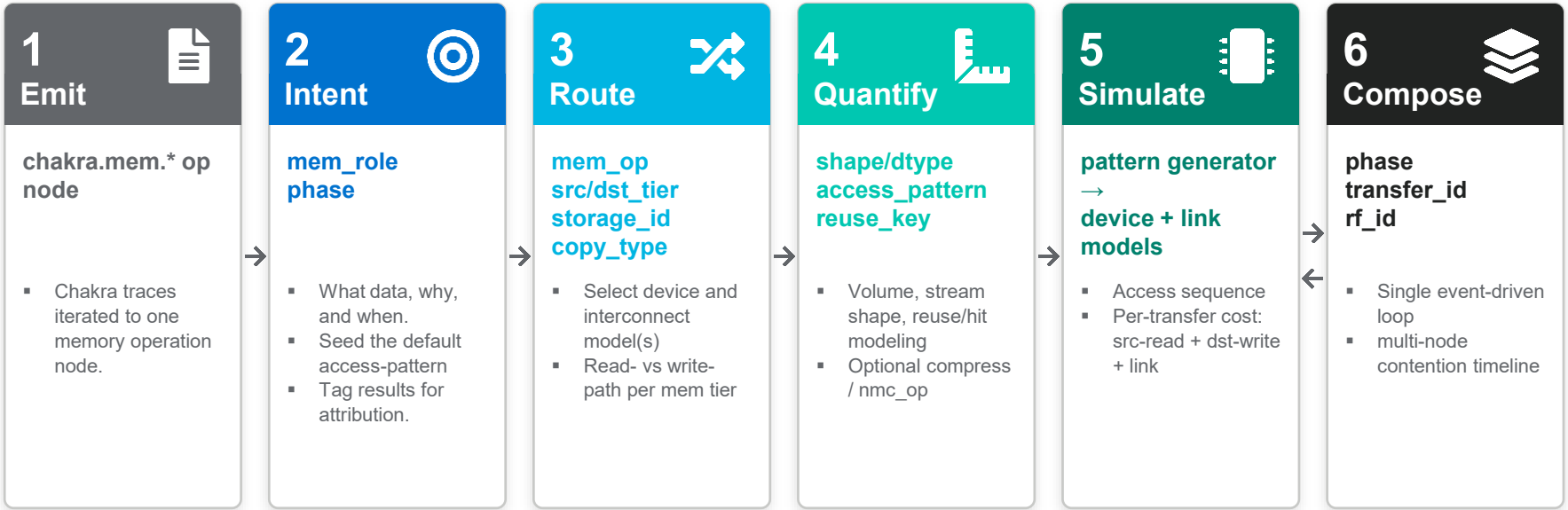
Mapping vLLM Workloads to the Generalized Node

vLLM Operation	mem_op	mem_role	src → dst	copy_type
KV Offload (GPU→CPU store)	STORE	KV_CACHE_OFFLOAD	G1 → G2	D2H
KV Restore (CPU→GPU load)	LOAD	KV_CACHE_PREFETCH	G2 → G1	H2D
Disaggregated KV (Prefill→Decode)	COPY	KV_CACHE_DISAGG_TRANSFER	G1 → G1 (cross-node)	D2D
LoRA Adapter Swap	LOAD	LORA_ADAPTER_SWAP	G2 → G1	H2D
MoE Expert Fetch (weight from host)	LOAD	MOE_EXPERT_FETCH	G2/G3 → G1	H2D
MoE Token Dispatch (all-to-all)	—	— (stays COMMS)	N/A	N/A

Scope discipline: MoE token routing (all-to-all) stays COMMS — only the expert weight fetch is a memory node. This keeps the proposal acceptable to the WG and faithful to the Chakra schema design.

How attributes drive a memory backend

One record per memory event flows through 6 stages — generalized to any memory node



Simulation Flow Example: KV_CACHE_OFFLOAD

Eviction of a KV block from GPU HBM (G1) down to a shared CXL pool (G2) — under 4-GPU contention

1	**Emit**	mem operation node: a KV block leaves HBM under decode-phase capacity pressure
2	**Intent**	phase = after DECODE mem_role = KV_CACHE_OFFLOAD → default access_pattern = BLOCK
3	**Route**	mem_op = STORE src_tier = G1 (HBM), dst_tier = G2 (CXL pool) copy_type = TIER → engage CXL link model
4	**Quantify**	shape / dtype → bytes block_id / reuse_key → identify KV block Pattern generator → BLOCK transfer
5	**Simulate**	Memory backend: G1 read + CXL link + G2 write
6	**Compose**	transfer_id / rf_id → couple fragments and encode dependencies phase → burst timescale / epoch granularity Merge 4 GPUs' streams at the shared pool arbiter → contention timeline

What this example actually needs

- **Required**
  - mem_role (KV_CACHE_OFFLOAD)
  - mem_op (STORE)
  - endpoints (G1 → G2)
- **Defaulted but overridable**
  - access_pattern (BLOCK)
  - copy_type (TIER)
- **Upper-layer context**
  - phase (DECODE)
  - transfer_id
  - rf_id
- **Optional backend features**
  - Compress
  - nmc_op

Takeaways

- **Backward-compatible by design — no new node types**
- **Memory semantics made explicit and tier-aware**
- **Generalized attributes actively drive memory backends**
- **Towards LLM serving and beyond**

MARVELL

Q & A

Essential technology, done right™