XINNOR

Enhancing Lustre for Modern AI with Disaggregated Storage and ML-Tuned Caching Policies

Sergei Platonov,
VP of Product Strategy, Xinnor

Our Motivation

Lustre has powered national research labs for decades. We want to bring it to the enterprise — where AI inference, and especially multi-turn workloads on-premise, demand a storage layer built for extreme throughput and low latency.

Beyond National Labs

Bringing the same parallel-filesystem performance story to enterprise data centers — where the storage problem is no longer simulation, but production AI.

The Inference Era

Overtaking training as the defining AI workload — and with a very different storage profile: many small reads, long context, persistent state.

Multi-Turn, On-Prem

How can Lustre solve the storage bottlenecks of multi-turn inference — KV cache, session state, retrieval — for enterprises running entirely on-premise?

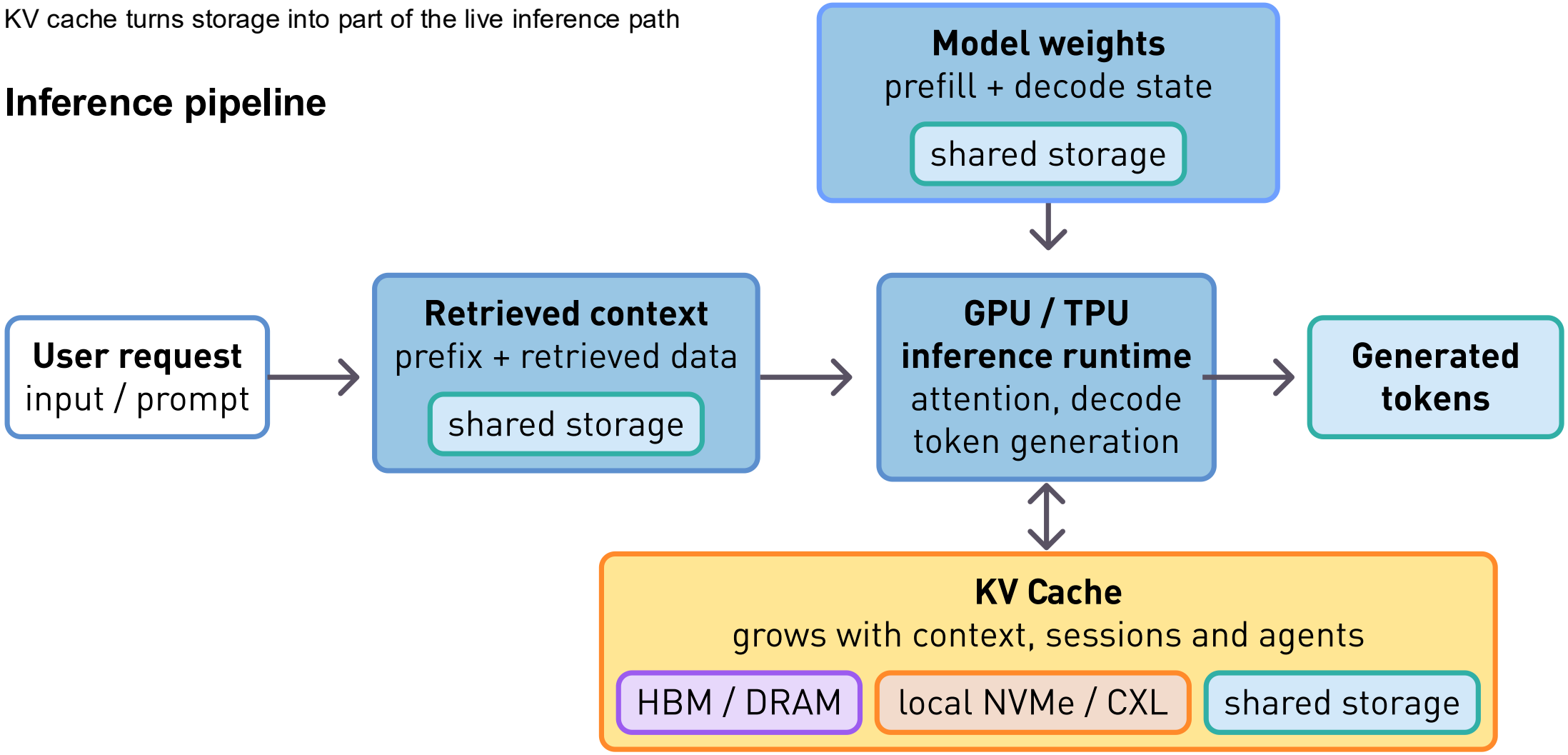
Part 1.

Why has storage for AI inference become such a hot topic?

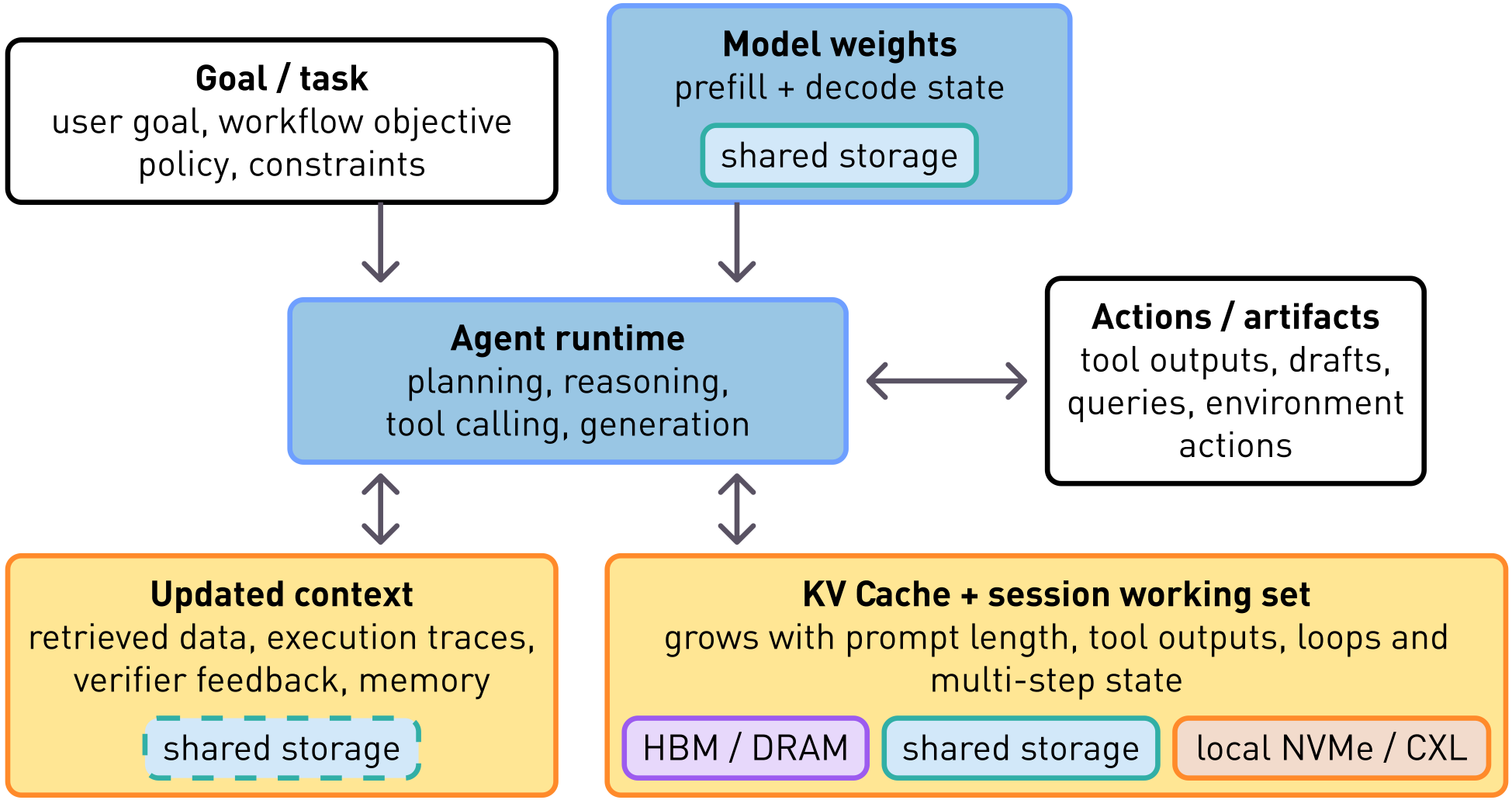
AI model inference

KV cache turns storage into part of the live inference path

Inference pipeline



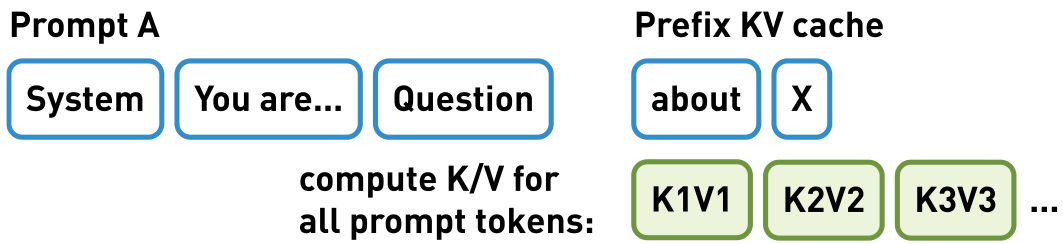
Agentic inference. Agent loop



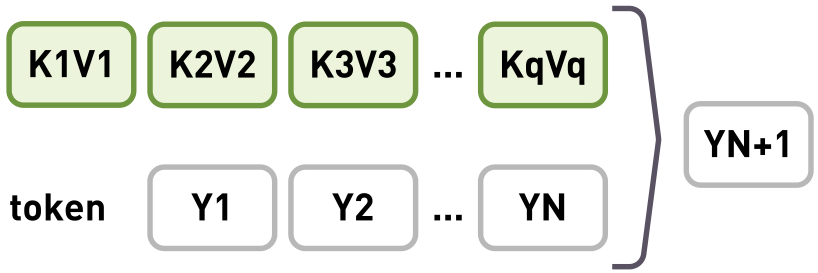
Prefill, decode, and how prefix cache works

1. prefill + decode without prefix

Prefill. Process all prompt tokens in parallel and write K/V into cache.



Decode. Generate one token at a time; each new token attends to cached prefix K/V and previously generated tokens.



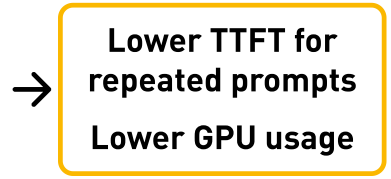
2. prefix reuse

Stored prefix cache from Request A

reuse K/V for the shared prefix

Request B. Same prefix, different suffix.

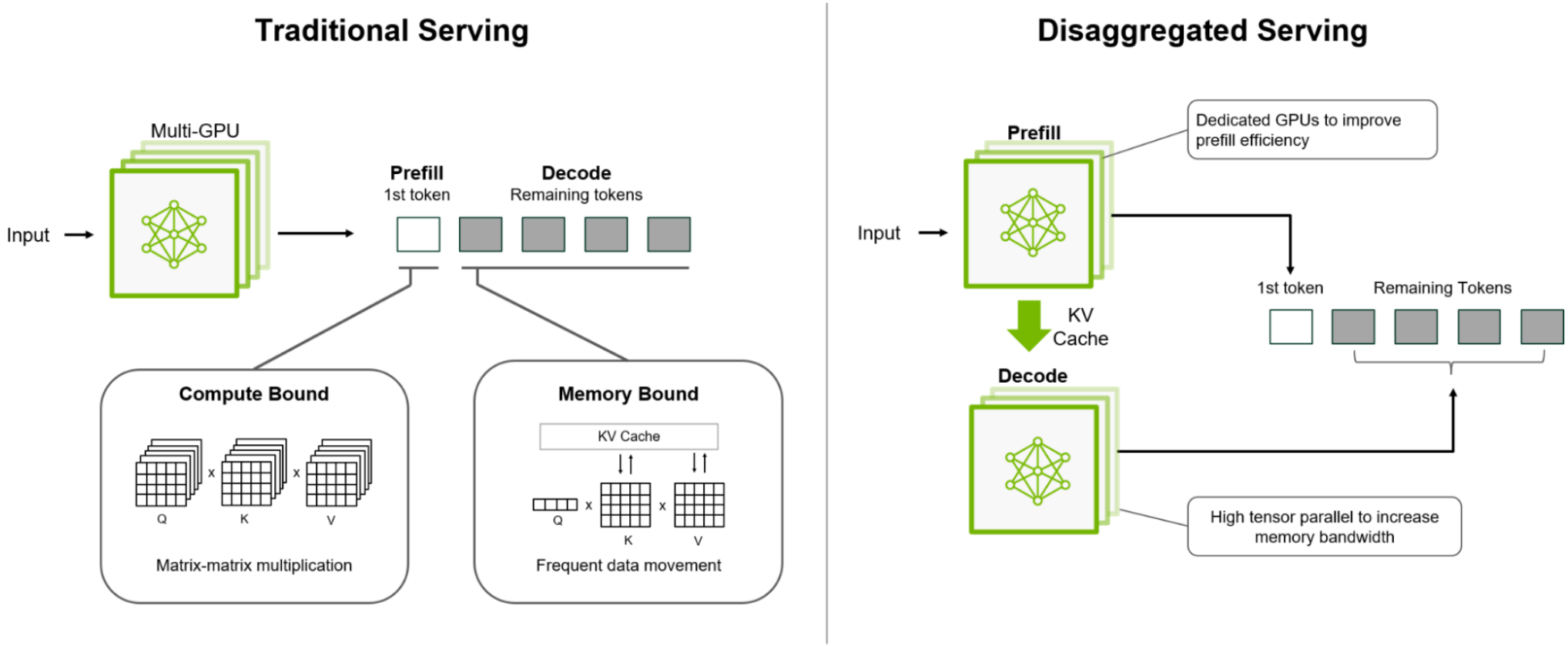
shared prefix



Skip prefill for shared prefix. Compute K/V only for new suffix tokens, then decode normally from the reused prefix.

Prefill computes keys and values (K/V) for prompt tokens once. Decode generates tokens one by one using cached K/V.
Prefix cache reuses K/V from a shared prompt prefix across requests.

Serving Architectures for LLM Inference



Source: NVIDIA Technical Blog. NVIDIA Dynamo, A Low-Latency Distributed Inference Framework for Scaling Reasoning AI Models

Inference storage is *not one workload.*

1. Data plane. Model loading & staging

Large sequential checkpoint/weight reads; sensitive to startup latency and reload events;

requires high throughput;

multiple models can be used in agentic AI environment

2. Context plane. Retrieved knowledge / RAG

Many concurrent (4–48 KB) reads against a shared corpus;

benefits from locality-aware caching and reuse of retrieval-related state

3. Control plane. Session artifacts & traces

Append-heavy, metadata-heavy operational data;

requires durable shared storage

4. Context memory. KV cache / context memory

Hot reusable state;

sensitive to tail latency and TTFT;

requires explicit multi-tier hierarchy and offload beyond GPU memory

1. Data Plane. Model loading & staging

Large sequential weight reads. Sensitive to startup latency and reload events. Requires high throughput.

I/O fingerprint

- **Pattern:** Large sequential reads, multi-stage: storage → ~~DRAM~~ → HBM
- **Size:** 10s–100s of GB per shard; full model TB-class
- **Mix:** Read-dominant on cold start;

Key references

- ServerlessLLM
  `Fu et al. · OSDI '24`
  Tiered NVMe → DRAM → HBM loader, locality-aware scheduling, loading-optimized checkpoint format. 3.6–8.2× faster load; 10–200× lower cold-start latency on Azure traces.
- fastsafetensors
  `Yoshimura et al. · IBM Research, IEEE CLOUD '25`
  Aggregated tensor copy direct to device memory; parallel I/O, P2P DMA, GPU offloading, GDS support. 4.8–7.5× speedup on Llama-7/13/70B and Falcon-40B.
- Additional research: DataStates-LLM, HPDC '24 (best paper); MegaScale, NSDI '24 (ByteDance); Gemini, SOSP '23.

Best tier:

Shared throughput storage — PFS or scale-out NAS, RDMA-capable

LUSTRE ON THIS WORKLOAD ●●● Strong fit

Lustre's home turf. OST striping + LNet over IB/RoCE matches the GPU fabric. Production-proven on Frontier, Aurora, Leonardo.

2. Context Plane. Retrieved knowledge / RAG

Many concurrent reads against a shared corpus. Benefits from locality-aware caching.

I/O fingerprint

- **Pattern:** Many small random reads on the index; large sequential on cached doc-KV
- **Size:** 4–48 KB per ANN access; MB-scale per cached chunk-KV
- **Mix:** >95% read once index built; high intra-query fan-out

Key references

- RAGCache
  `Jin et al. · arXiv 2404.12457 / ACM TOCS`
  Power-law over retrieved docs; multi-level KV-tensor cache (HBM → DRAM) as knowledge tree with prefix-aware eviction. Up to 4× lower TTFT, 2.1× higher throughput vs vLLM + Faiss.
- Cache-Craft
  `Agarwal et al. · SIGMOD '25 (Adobe Research)`
  Manages chunk-KVs at non-prefix positions: identifies reusable chunks, recomputes ~20–30% to restore cross-attention, tiers across GPU/CPU/SSD. 51% less compute, 1.6× throughput.
- Additional research: DiskANN, NeurIPS '19 (Microsoft); SPANN, NeurIPS ‘21; Starling, SIGMOD '24

Best tier:

Shared namespace + locality-aware caching, IOPS-optimized

LUSTRE ON THIS WORKLOAD ●●○ **Mixed fit**

Data-on-MDT changes the calculus. DoM tuned to access size — small ANN reads bypass OSS, land on NVMe-backed MDT in one RPC.

3. Control-plane. Session artifacts & traces

Append-heavy, metadata-heavy operational data. Requires durable shared storage, efficient ingest, retention/queryability, and an observability-oriented pipeline.

I/O fingerprint

- **Pattern:** Sustained record append from many concurrent workers
- **Size:** Needed to be investigated further
- **Mix:** Write-dominant ingest; bimodal read

Key references

- AgentTrace
  `AlSayyad et al. · arXiv 2602.10133 (UC Berkeley)`
  Structured logging for LLM-agent observability. Captures three trace surfaces — operational, cognitive, contextual — at runtime. Defines the schema and ingest pattern the storage tier must absorb.
- Hindsight
  `Zhang et al. · USENIX NSDI '23`
  Retroactive sampling for distributed tracing — local buffer until problem detected ("dash-cam" model). Nanosecond-level overhead at ingest, GB/s per node. Resolves head- vs tail-sampling tradeoff.

Best tier:

Persistent shared storage

LUSTRE ON THIS WORKLOAD

Lack of modern research — to investigate!

4. KV Cache / Context Memory

Sensitive to tail latency and TTFT. Requires an explicit multi-tier hierarchy and offload beyond GPU memory.

I/O fingerprint

- **Pattern:** Multi-tier: HBM → DRAM → NVMe → remote shared storage
- **Size:** 64KiB – several dozens MiBs
- **Mix:** Mixed R/W; sequential append on prefill, large bloc read on decode
- Performance: **2-200GiBps** on decode phase

Key references

- KV-NVMe I/O study
  `Ren, Doekemeijer et al. · CHEOPS '25`
  First block-level trace of KV offload on NVMe. 128 KiB dominates; FlexGen sees 2.0 GiB/s read vs 11 MiB/s write — 186× asymmetry.
- CachedAttention / AttentionStore
  `Gao et al. · USENIX ATC '24`
  Layer-wise KV prefetch (HBM, DRAM, NVMe) hidden behind compute. 7.8× prefill, 87% TTFT reduction, ~70% lower inference cost.
- Mooncake
  `Qin et al. · FAST '25 best paper`
  Production KV-disaggregated store behind Kimi (>100B tokens/day). RDMA: 87 GB/s on 4×200G, 190 GB/s on 8×400G; 2.36× hit rate.

Best tier:

Explicit multi-tier design with RDMA fabric and layer-pipelined I/O

LUSTRE ON THIS WORKLOAD ●●○ **Warm tier**

Lustre demonstrates solid performance for some workloads we will discuss further

An I/O Characterizing Study of KV Cache Offload to NVMe SSD

The first — and still the only — block-level trace of what KV offload actually looks like on NVMe. Not what systems designers imagined; what fio and iostat saw.

I/O SIZE

128 KiB dominates

Both reads and writes concentrate at 128 KiB.

Storage sees medium coalesced I/O

BANDWIDTH

2.0 GiB/s read · 11 MiB/s write

FlexGen KV offload on an NVMeVirt SSD (5.3 GiB/s single-thread, 16.9 GiB/s four-thread peak).

A 186× read/write imbalance

TIER TOPOLOGY

HBM->A single SSD

SSD not saturated.

Workload: OPT-family models, FP16 · L4 GPU, 32-core CPU, 128 GiB DRAM · NVMeVirt SSD emulator (5.3 → 16.9 GiB/s peak)

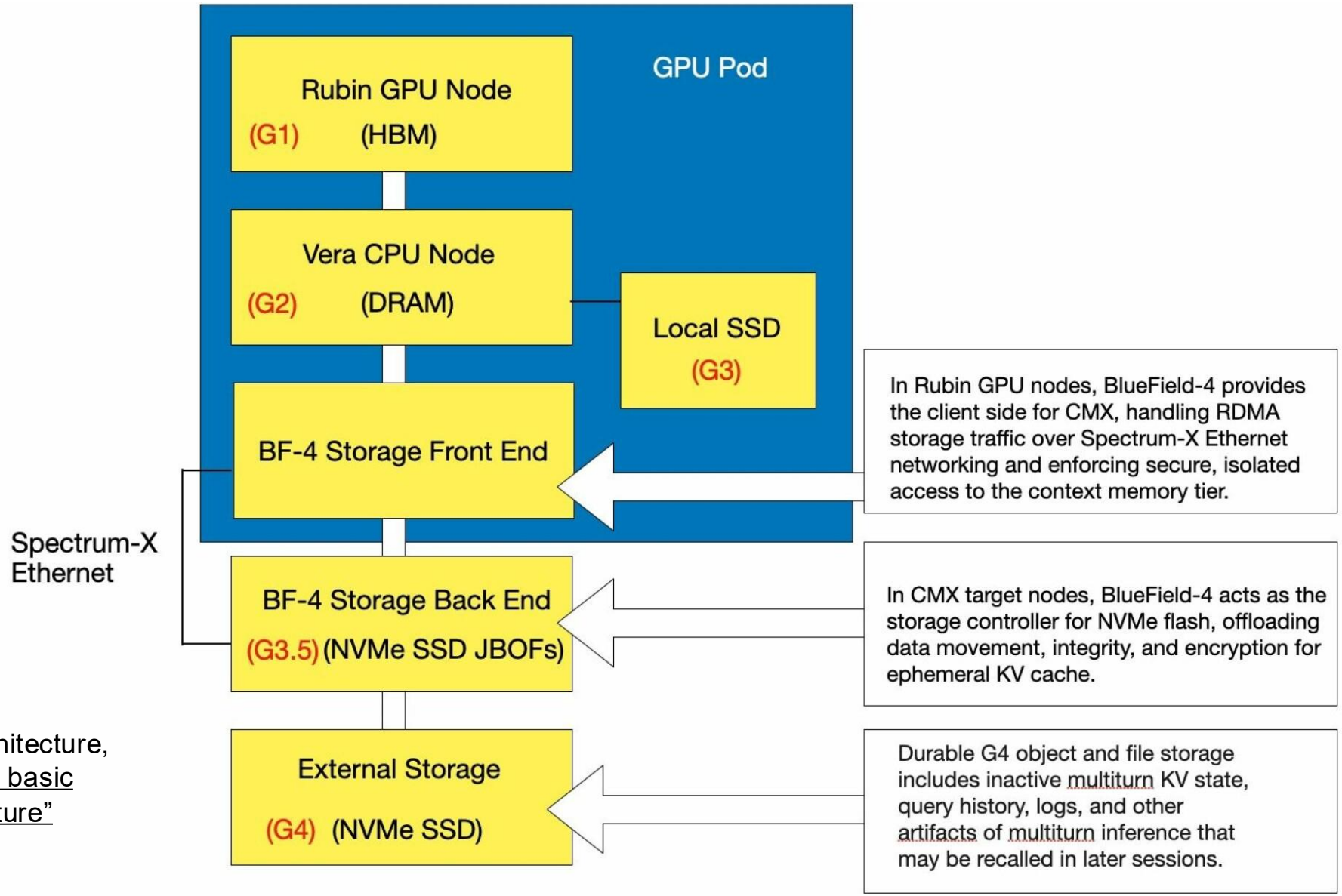
Only paper that publishes the actual SSD block trace:

`Ren, Doekemeijer, et al. · IBM + VU Amsterdam · CHEOPS '25`

Part 2.

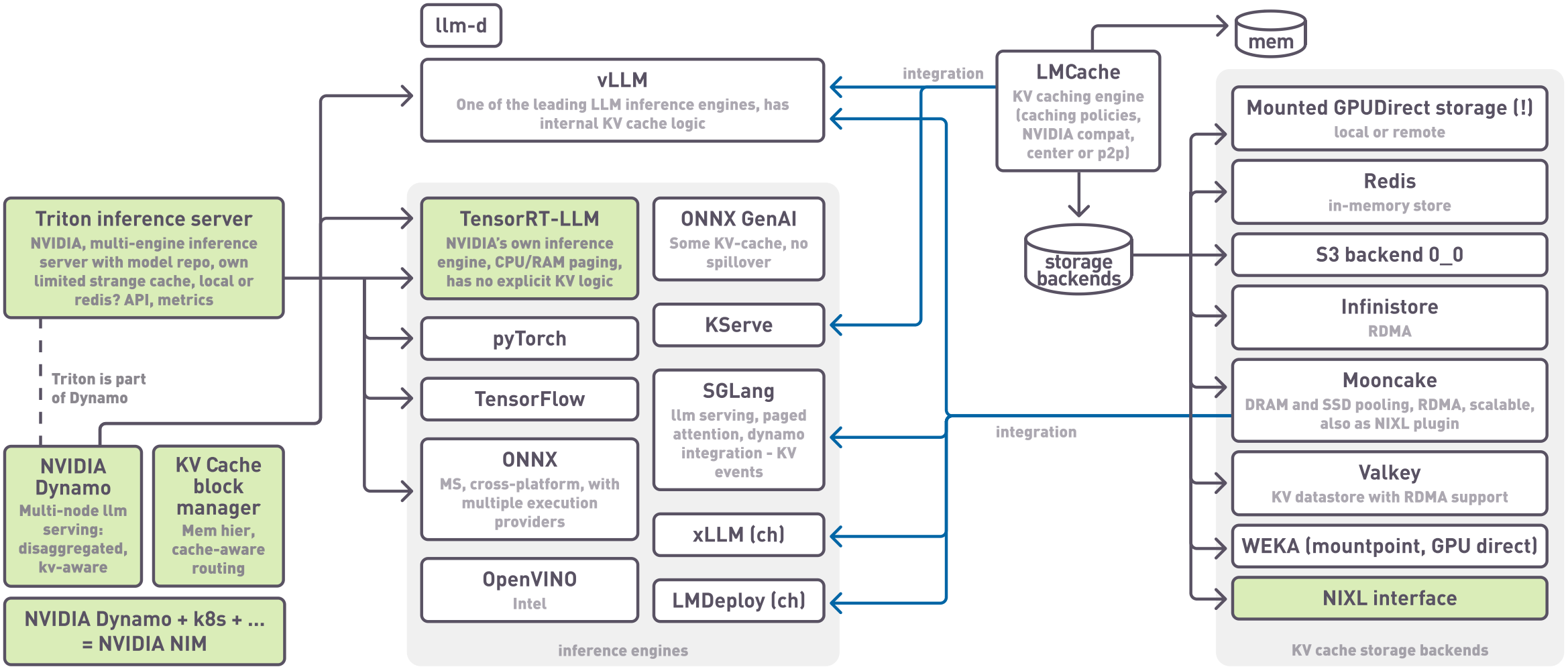
A new storage layer for AI inference

Inference hardware architecture

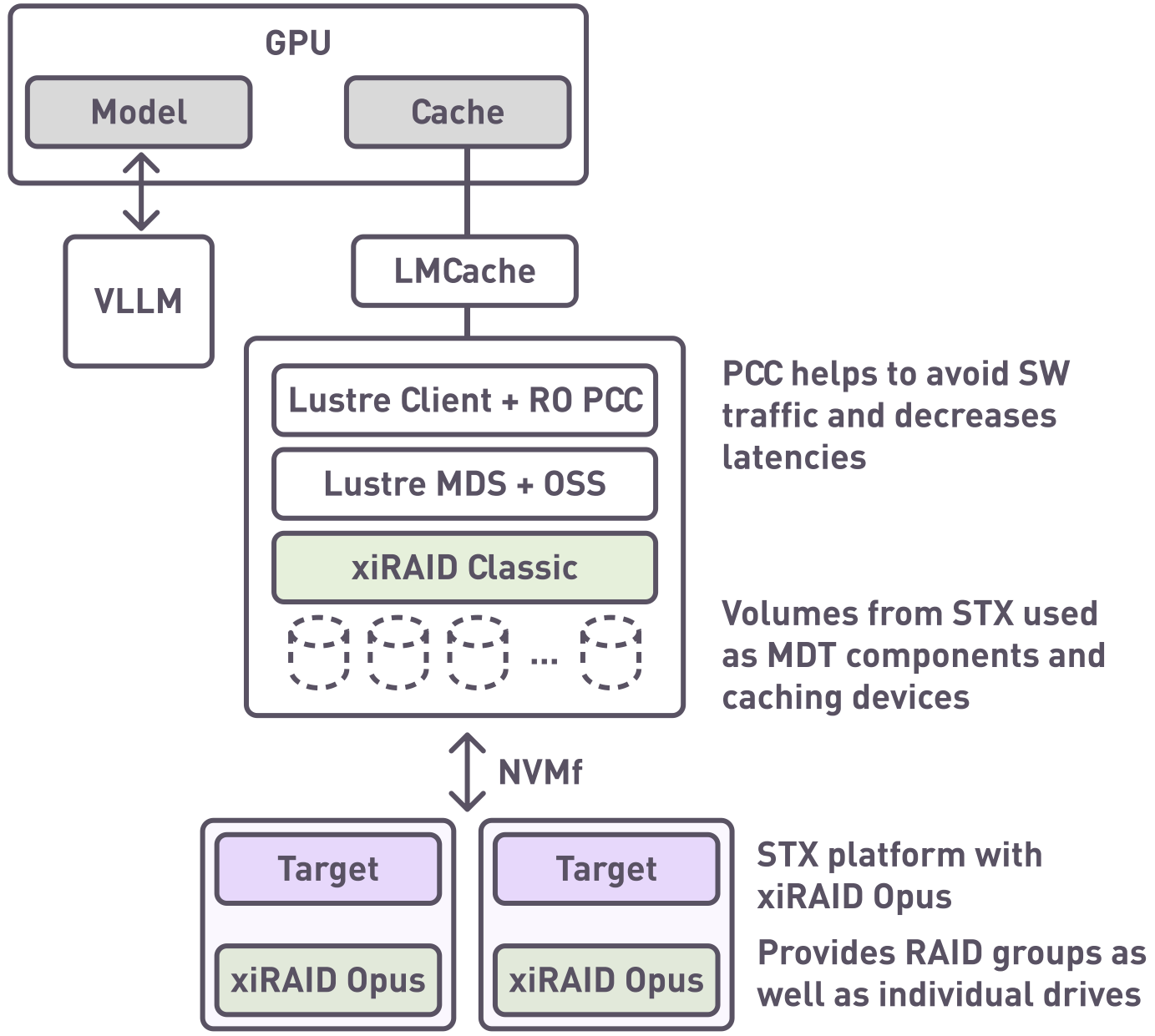


Source: NVIDIA CMX / Dynamo architecture, as shown in Blocks & Files, "Nvidia's basic context memory extension infrastructure"

Inference software architecture

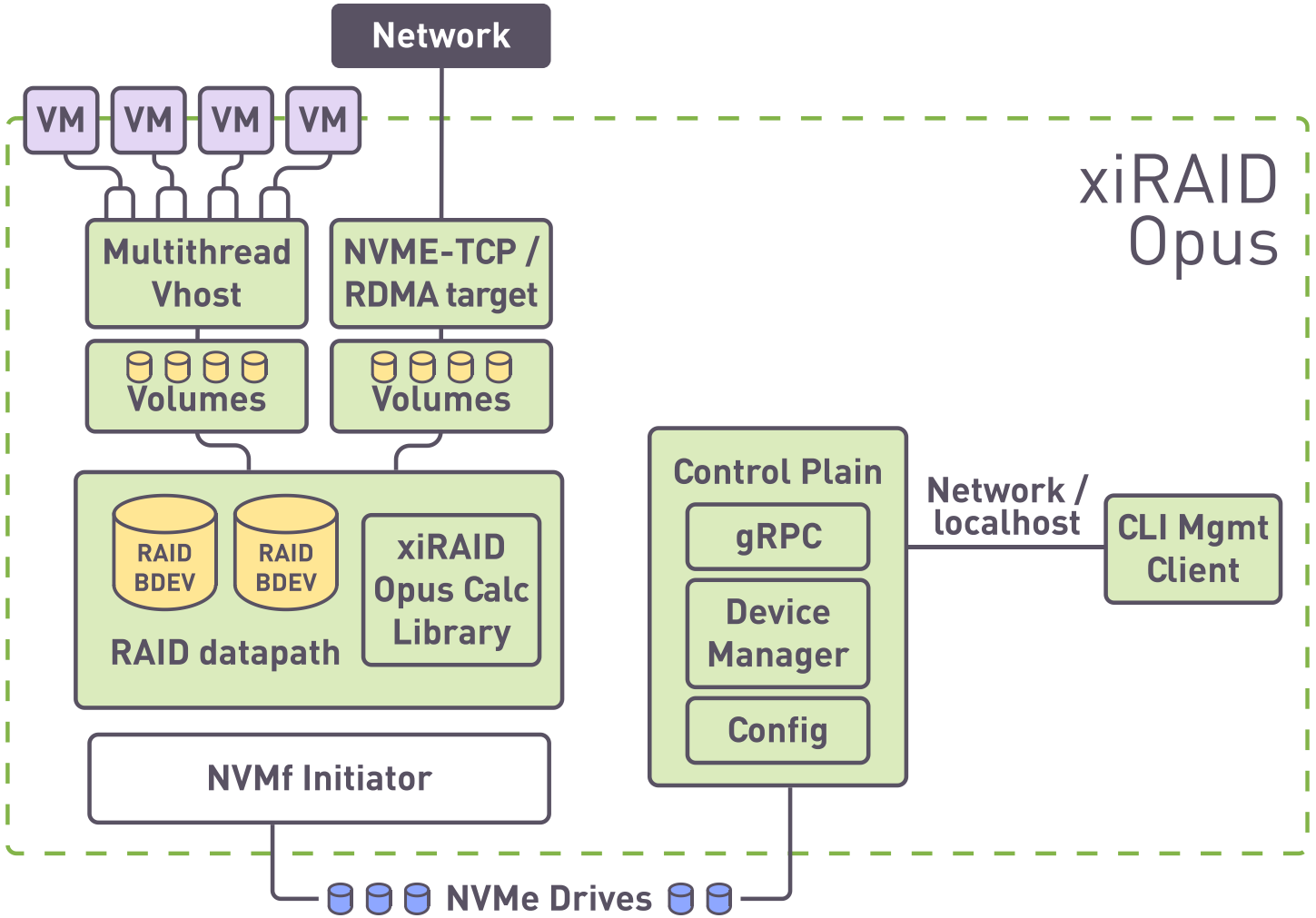


What implementation looks like with the current Lustre server and client

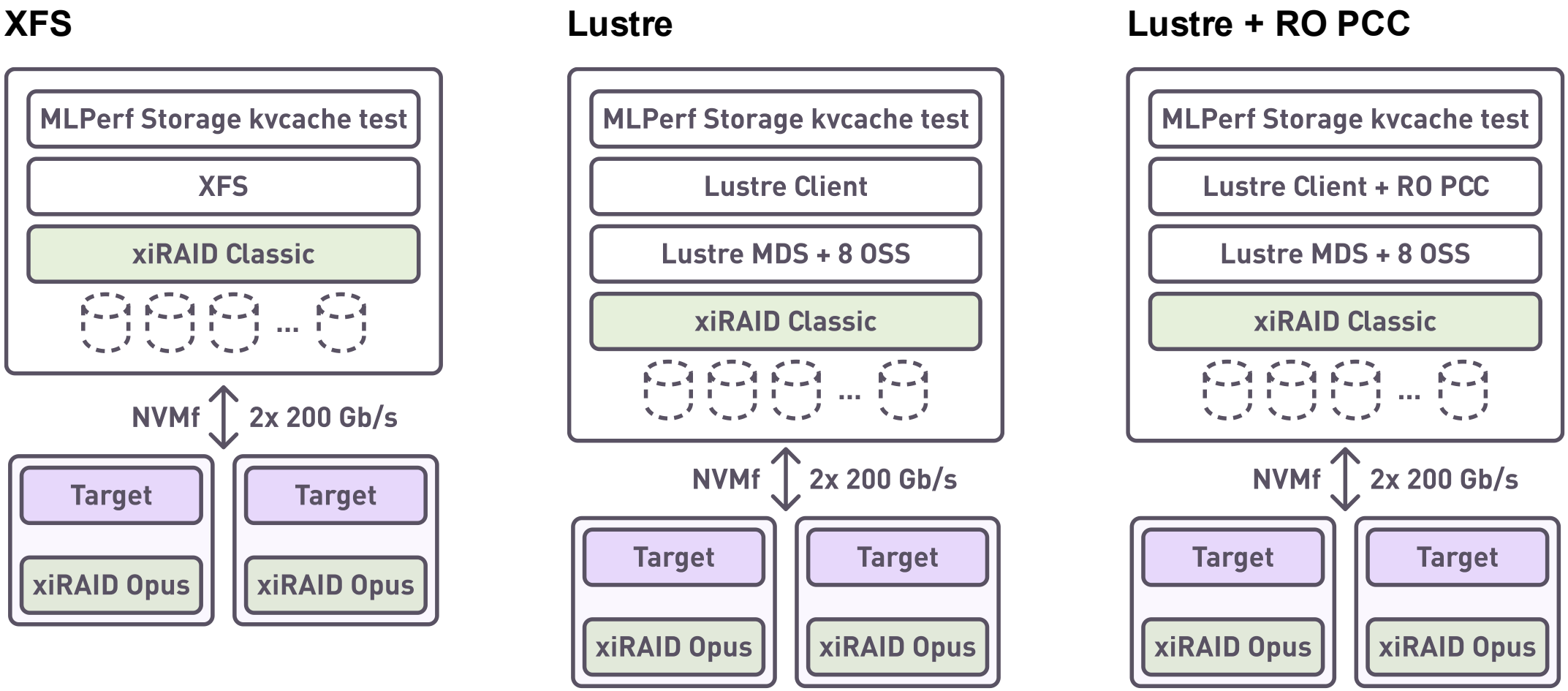


xiRAID Opus Architecture

- xiRAID Opus is based on SPDK libraries to operate in User Space (data path and drive connection)
- xiRAID Opus uses polled-mode instead of interrupts (Kernel mode) to check when data is ready, reducing CPU cycles and maximizing performance
- CPU utilization: xiRAID Opus fully occupies one or more CPU cores to achieve max performance
- Unique vectorized RAID Checksum calculation:
  - AVX2 on x86
  - Vector registers on ARM



MLPerf testing (1)



MLPerf testing (2)

Other test dimensions:

- **Model (KV arch):** `llama3.1-70b-instruct` dense GQA; `gpt-oss-120b` MoE; `deepseek-v3 - MLA`
- **Workload shape:** `synthetic-chatbot`; `synthetic-coding`; `synthetic-document`; `sharegpt`; `burstgpt`

  https://github.com/mlcommons/storage/blob/main/kv_cache_benchmark/README.md

  - Scenario 1 — caches ON, capacity-mode autoscaler (min=10 max=500)
  - Scenario 2— caches OFF --cpu-mem-gb 0 --disable-multi-turn --disable-prefix-caching
- **Phase:** prefill-only; decode-only; "cold" decode in the PCC variant/warm decode in the PCC variant

MLPerf testing (3)

Fixed benchmark parameters:

- **Duration:** 180s per experiment (XFS/Lustre);
- `--max-concurrent-allocs 32`
  - `--storage-capacity-gb 200`
- **Trials:** 1 per experiment

Experiments counts per variant:

- **XFS:** 55 executed experiments
  5× 70B/s1/decode skipped
- **Lustre:** 40 executed experiments
  all llama3.1-70b experiments skipped — DIO kernel panic
- **Lustre + RO-PCC:** 48 executed experiments
  70B excluded (same panic); burstgpt excluded trace-rate-bound

How we run each test

Invocation

```
python3 kv-cache.py \
  --config config_<workload>.yaml \
  --model <70B | 120B | DSv3> \
  --cache-dir /mnt/<fs>/kvcache-sweep/... \
  --output result.json \

  --generation-mode none \
  --gpu-mem-gb 0 \
  --cpu-mem-gb <4 | 0>          # 4=s1  0=s2
  --max-concurrent-allocs 32 \
  --enable-autoscaling --autoscaler-mode capacity \
  --storage-capacity-gb 200 \
  --duration 180  --seed 42 \

  <--prefill-only | --decode-only> \
  [--disable-multi-turn ...]  # s2
  [--dataset-path ShareGPT.json ...]  # sharegpt
  [--use-burst-trace ...]  # burstgpt
```

One **kv-cache.py** invocation per experiment:
model × workload × scenario × phase

Same harness for every filesystem under test.

Limitations

PRE-RELEASE SOFTWARE

MLPerf Storage 3.0 still under development

These runs are against an unreleased SW. Bug-fix patches and tuning are still landing — the same workload may behave differently in the GA release

PENDING PATCH

~15% headroom from a not-yet-merged patch

A patch we applied gave ~15% better throughput in our testing, but it has not yet been accepted upstream.

INTERNAL TESTING

Results not externally submitted

Numbers are from in-house benchmarking on a single testbed; they have not been submitted to any external organization or independently reviewed. Reproducibility on a different cluster is not yet established.

SCOPE

Only the most valuable experiments shown

The full sweep is 143 cells across 3 filesystems × 5 workloads × 2 scenarios × 2 phases. This deck highlights the cells that carry the most signal; the long tail of routine results is omitted to keep the story tight.

I/O Profile on XFS

s2 prefill (writes only — reads are near-zero)

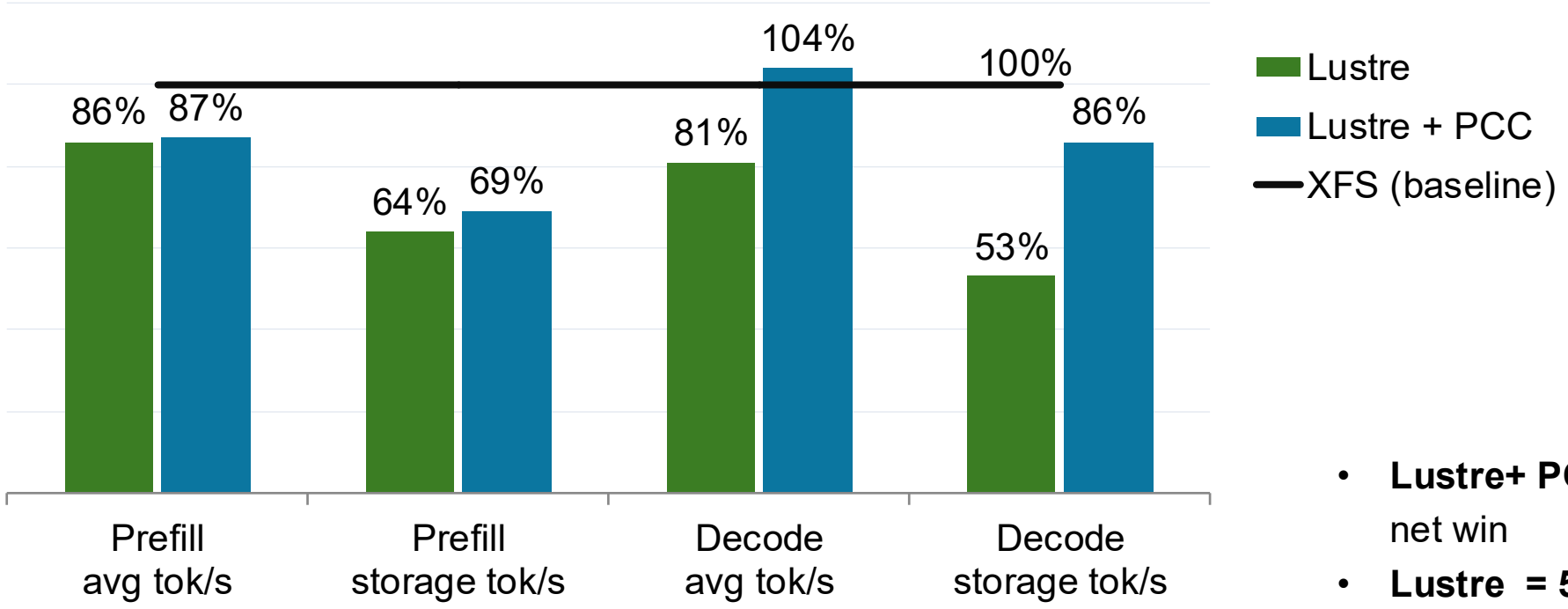
Model	Workload	W avg MB	W peak MB	W avg GB/s	W peak GB/s	QD avg	QD peak
deepseek-v3	synthetic-chatbot	144	1,007	9.4	13.7	3	32
gpt-oss-120b	synthetic-chatbot	147	1,014	9.1	15.9	4	48
llama3.1-70b	**synthetic-chatbot**	**328**	**1,049**	**9.5**	**25.2**	**3**	**28**
deepseek-v3	synthetic-document	140	1,012	9.3	14	4	60
gpt-oss-120b	synthetic-document	145	1,014	9.3	14.1	5	65
llama3.1-70b	**synthetic-document**	**290**	**1,049**	**8.7**	**21.7**	**2**	**17**

s2 decode (reads dominate, writes are fsync flushes)

Model	Workload	R avg MB	R peak MB	W avg MB	W peak MB	R pk GB/s	W pk GB/s	QD peak
deepseek-v3	synthetic-chatbot	11	23	66	141	13.2	13.3	75
gpt-oss-120b	synthetic-chatbot	12	23	69	147	14	12.2	44
llama3.1-70b	**synthetic-chatbot**	**18**	**23**	**200**	**655**	**27**	**27.6**	**24**
deepseek-v3	synthetic-document	11	23	66	141	14.2	12.4	59
gpt-oss-120b	synthetic-document	12	23	69	147	13.8	11.5	58
llama3.1-70b	**synthetic-document**	**18**	**23**	**212**	**655**	**26.6**	**36.5**	**32**

Throughput vs XFS Baseline (averaged by all models and workloads)

Lustre trails XFS by ~15–20% on end-to-end throughput. Average across all tests; XFS = 100%.



- **Lustre+ PCC decode = 104% XFS** — net win
- **Lustre = 53–64%** — net-bound
- **Lustre + PCC** — cache warms fast

Throughput vs XFS Baseline (details by 2 models and all workloads)

Prefill

Workload	Scen	Lustre	PCC
gpt-oss-120b MoE 120B			
sharegpt	s1	82%	93%
sharegpt	s2	–	–
synthetic-chatbot	s1	94%	96%
synthetic-chatbot	s2	86%	83%
synthetic-coding	s1	85%	80%
synthetic-coding	s2	87%	85%
synthetic-document	s1	92%	92%
synthetic-document	s2	88%	91%
deepseek-v3 MLA			
sharegpt	s1	93%	112%
sharegpt	s2	–	–
synthetic-chatbot	s1	95%	102%
synthetic-chatbot	s2	66%	61%
synthetic-coding	s1	101%	97%
synthetic-coding	s2	72%	70%
synthetic-document	s1	95%	92%
synthetic-document	s2	67%	69%

Decode

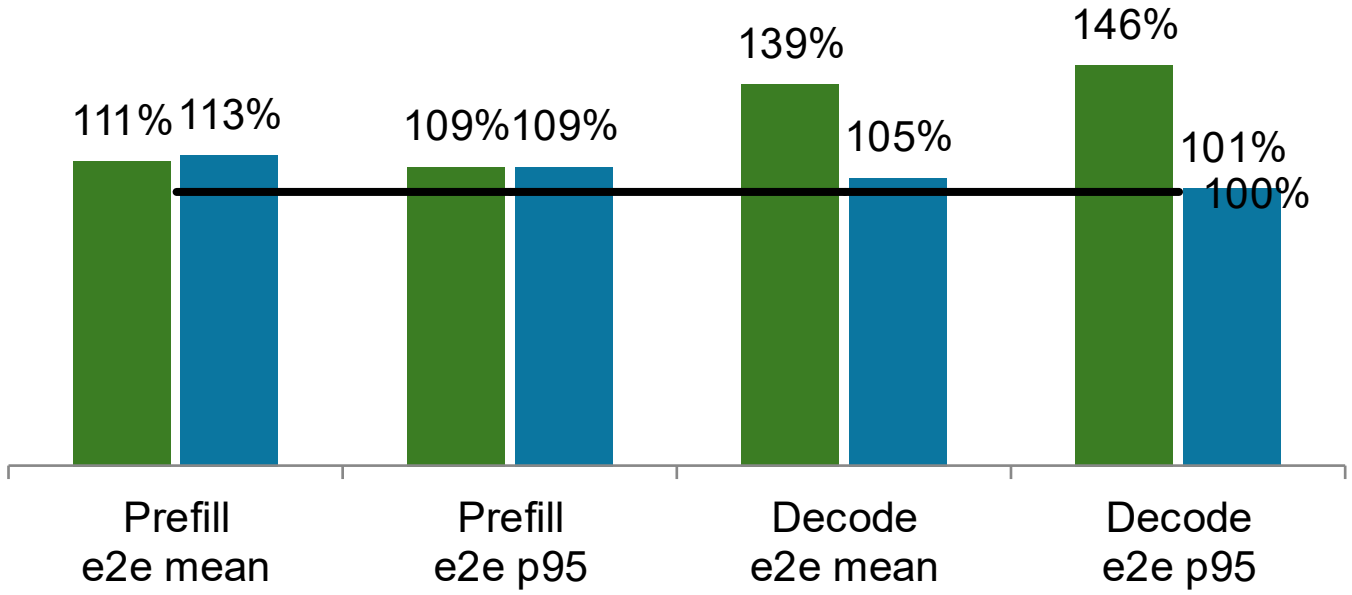
Workload	Scen	Lustre	PCC d1	PCC d2
gpt-oss-120b MoE 120B				
sharegpt	s1	87%	122%	109%
sharegpt	s2	72%	101%	95%
synthetic-chatbot	s1	93%	98%	94%
synthetic-chatbot	s2	80%	108%	107%
synthetic-coding	s1	95%	103%	96%
synthetic-coding	s2	81%	109%	105%
synthetic-document	s1	95%	111%	116%
synthetic-document	s2	78%	105%	102%
deepseek-v3 MLA				
sharegpt	s1	71%	100%	92%
sharegpt	s2	73%	96%	98%
synthetic-chatbot	s1	80%	90%	88%
synthetic-chatbot	s2	57%	107%	108%
synthetic-coding	s1	84%	86%	96%
synthetic-coding	s2	85%	114%	114%
synthetic-document	s1	85%	105%	108%
synthetic-document	s2	79%	105%	100%

cell shading:

- < 50%
- 50–69%
- 70–84%
- 85–99%
- 100–109%
- ≥ 110%

Latency vs XFS Baseline (averaged by all models and workloads)

Lower is better. Lustre + PCC slashes decode tail from 146% → 101% and mean from 139% → 105%, both within ~5% of XFS. Prefill unchanged.



- **Lustre+ PCC decode p95: 146% → 101%** — tail win
- **Lustre decode mean = 139%** — 1.4× XFS
- **Prefill ≈ 110% on both** — no regression

Eviction Policy Study — Setup

Lustre 2.17.52 ships PCC with a fixed eviction path.

We replayed real KV-cache traces through 7 cache policies in an offline simulator to find out the best eviction policy.

1. Capture

Real workloads on Lustre + PCC

models	**llama3.1-8b · gpt-oss-120b · deepseek-v3**
phases	**phase1 (no PCC) · phase2 (PCC RO auto-attach)**
tools	**strace -y · bpftrace pcc_* · llite stats**
duration	**20 min per (model, phase)**

30 traces **8.0M** events **281GB** WS

2. Replay

Seven policies, one offline simulator

belady	oracle MIN — upper bound
lru	byte-LRU baseline
2q	Kin/Kout — promote on 2nd hit
arc	adaptive recency / frequency
clock_pro	hot/cold/ghost CLOCK
lfu_aging	frequency + counter halving
tinylfu_slru	CMS admission door + SLRU

6 cache fractions **1,260** runs

3. Score

Three metrics per (trace × policy × frac)

hit_ratio = hits / total_ops

How often a request is served from cache. Higher = lower latency.

amplification = pop_bytes / hit_bytes

Bytes pulled from backend per byte served. Lower = less wasted I/O.

wasted_ratio = admits never reused

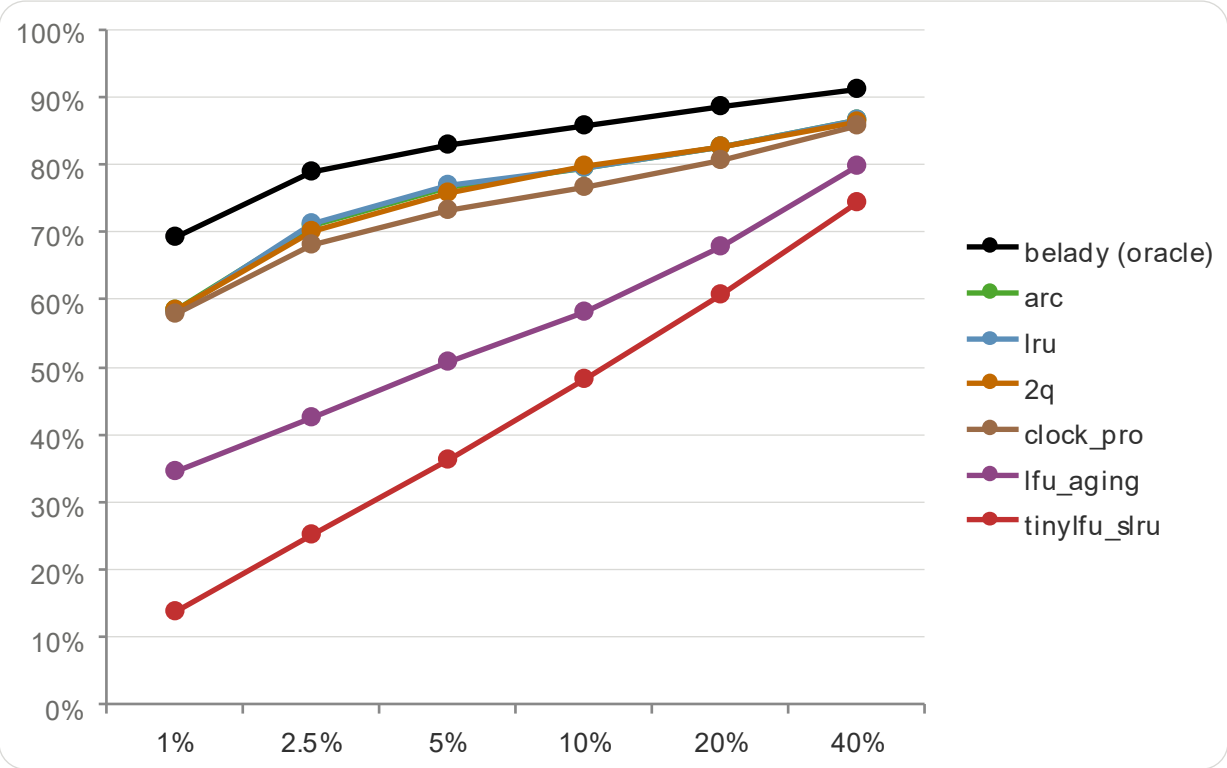
Did this admit even matter? Lower = fewer pointless inserts.

Pareto: hit_ratio ↔ amplification *reveals the regime split*

Headline Policy Comparison

1,260 simulator runs (30 traces × 7 policies × 6 cache fractions). Means across the full grid. **ARC, LRU and 2Q tie within 1 pt of each other and within 7 pts of the Belady oracle.** TinyLFU+SLRU is the outlier — much lower hit ratio but **10× lower amplification.**

Hit ratio across cache fractions



Mean across all 30 traces × 6 fracs

Policy	hit_ratio	amp	gap to Belady
belady	0.827	0.254	—
arc	0.758	0.430	0.07
lru	0.756	0.430	0.07
2q	0.755	0.434	0.07
clock_pro	0.737	0.510	0.09
lfu_aging	0.556	1.106	0.27
tinylfu_slru	0.430	0.035	0.40

TOP TIER ARC ≈ LRU ≈ 2Q — within 7 pts of Belady

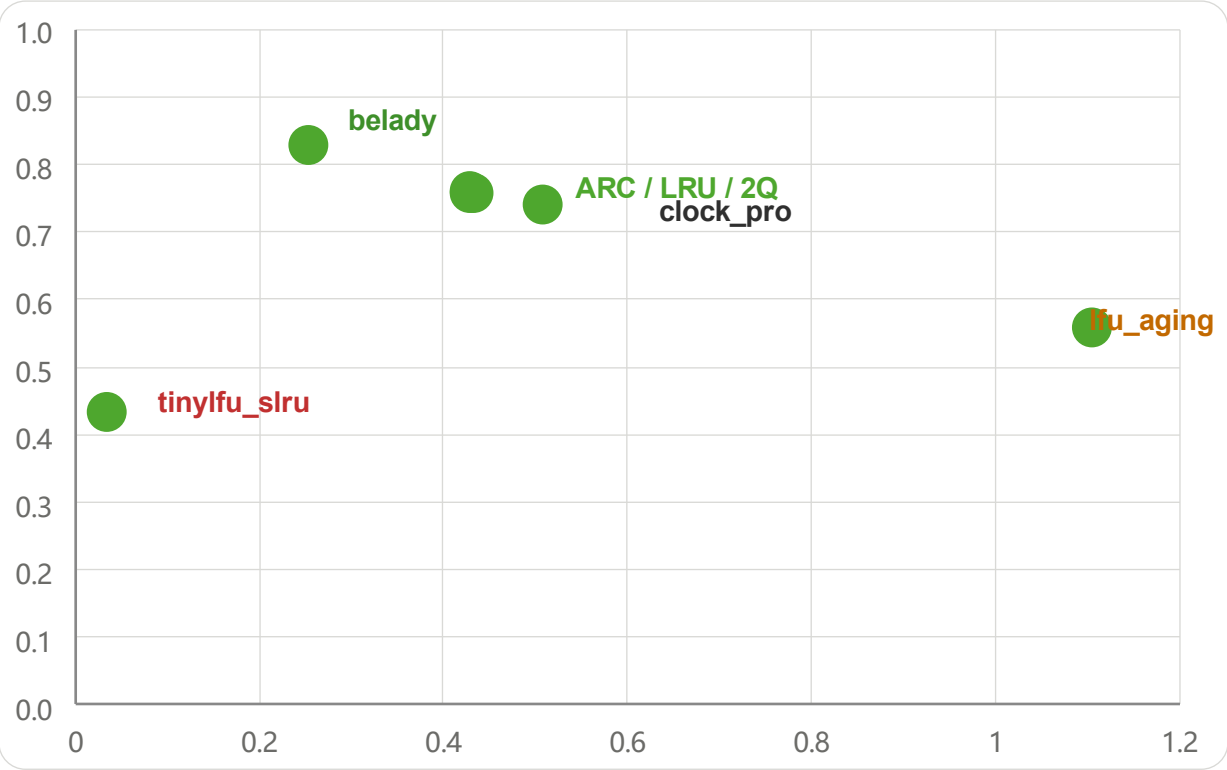
SECOND TIER CLOCK-Pro 0.737 · small gap below 5%

OUTLIERS LFU-aging 0.556 · TinyLFU 0.430

Two Regimes — Hit Ratio vs Amplification

Plotting hit_ratio against amplification reveals two clean operating bands. **ARC / LRU / 2Q form a tight Pareto front for latency-bound serving.** TinyLFU sits alone in the bandwidth-bound band — sacrifice 33 pts of hit ratio, get 10× lower backend pressure.

Mean hit_ratio vs mean amplification



Recommendation depends on bottleneck

HIT-RATIO BOUND

Pick ARC / LRU / 2Q

When request latency dominates and storage bandwidth is plentiful. The three are within `1 pt` of each other — choice is dominated by **implementation cost, not numbers**. LRU wins by simplicity.

BANDWIDTH BOUND

Pick TinyLFU + SLRU

When backend bandwidth is the bottleneck and 30–40 pts of hit ratio can be sacrificed. CMS admission door produces `amp = 0.035` (10× better than LRU). The right pick when storage can't keep up.

Tuning Doesn't Change the Tier

13 parameter variants × 30 traces × 6 fracs = **2,340 runs** sweeping the underperformers' knobs. **Neither TinyLFU nor LFU-aging moves out of its band.**

TinyLFU + SLRU

variant	prob_frac	cms_width	hit_ratio
tinylfu_p10	0.10	16,384	0.423
tinylfu_p20	**0.20**	16,384	**0.430**
tinylfu_p20_w65k	0.20	65,536	0.438
tinylfu_p50	0.50	16,384	0.428
tinylfu_p80	0.80	16,384	0.426

Spread: **0.423 → 0.438** · **1.5 pt** across all five variants

VERDICT

Only useful tweak: **set cms_width = 65,536**. +0.8 pt mean hit ratio at near-zero memory cost. Otherwise prob_frac is a no-op.

LFU-aging

variant	age_every	age_shift	hit_ratio
lfu_e1k_s1	1,000	1	0.544
lfu_e5k_s1	5,000	1	0.553
lfu_e10k_s1	**10,000**	**1**	**0.556**
lfu_e10k_s2	10,000	2	0.556
lfu_e10k_s3	10,000	3	0.556
lfu_e50k_s1	50,000	1	0.561

Spread: **0.544 → 0.561** · **1.7 pt** · *age_shift = no effect*

VERDICT

No setting closes the gap to LRU.

Three-Stage Testing Roadmap

Goal. Validate Lustre and RO-PCC across progressively more complex scenarios — from synthetic performance optimization to real-world AI agent workflows.

1. Vertical Scaling & Testbed Optimization

Increase the performance capacity of the test environment before moving to larger configurations.

Focus areas

→ Drive the 400 Gb storage network to full utilization (at least selected tests) keeping latency bellow TH

> **Key question:** *How does RO-PCC behave when the storage network is no longer underutilized?*

2. Scale-Out Configuration

Expand the environment by adding more storage resources, OSS nodes, and Lustre-integrated clients.

Expected effects

→ More parallelism via native Lustre load distribution

> **Key question:** *Does RO-PCC improve performance in a horizontally scaled Lustre configuration?*

3. Real Workload Validation

After meaningful results from stages 1–2, move from synthetic to a real value-test scenario.

Planned setup

→ Run an agentic closed-loop workflow on a complex software project. Beyond KV cache, store: source code · docs · etc

> **Key question:** *Which Lustre settings improve real agentic workload behavior*