

The shortening of the way..

RDMA enhancements for Data and AI

Kyle Bader
Chief Architect, Data and AI
Ceph @ IBM Storage

KV cache is now a storage problem

DeepSeek's production KV cache tier, as described in DeepInfer

~100 PiB

of SSD dedicated to KV cache

3 / 7 days

TTL for agent / chat traffic

14.4 TB

of cache per GPU (logical)

10+2

erasure coding, no replication: it's a cache

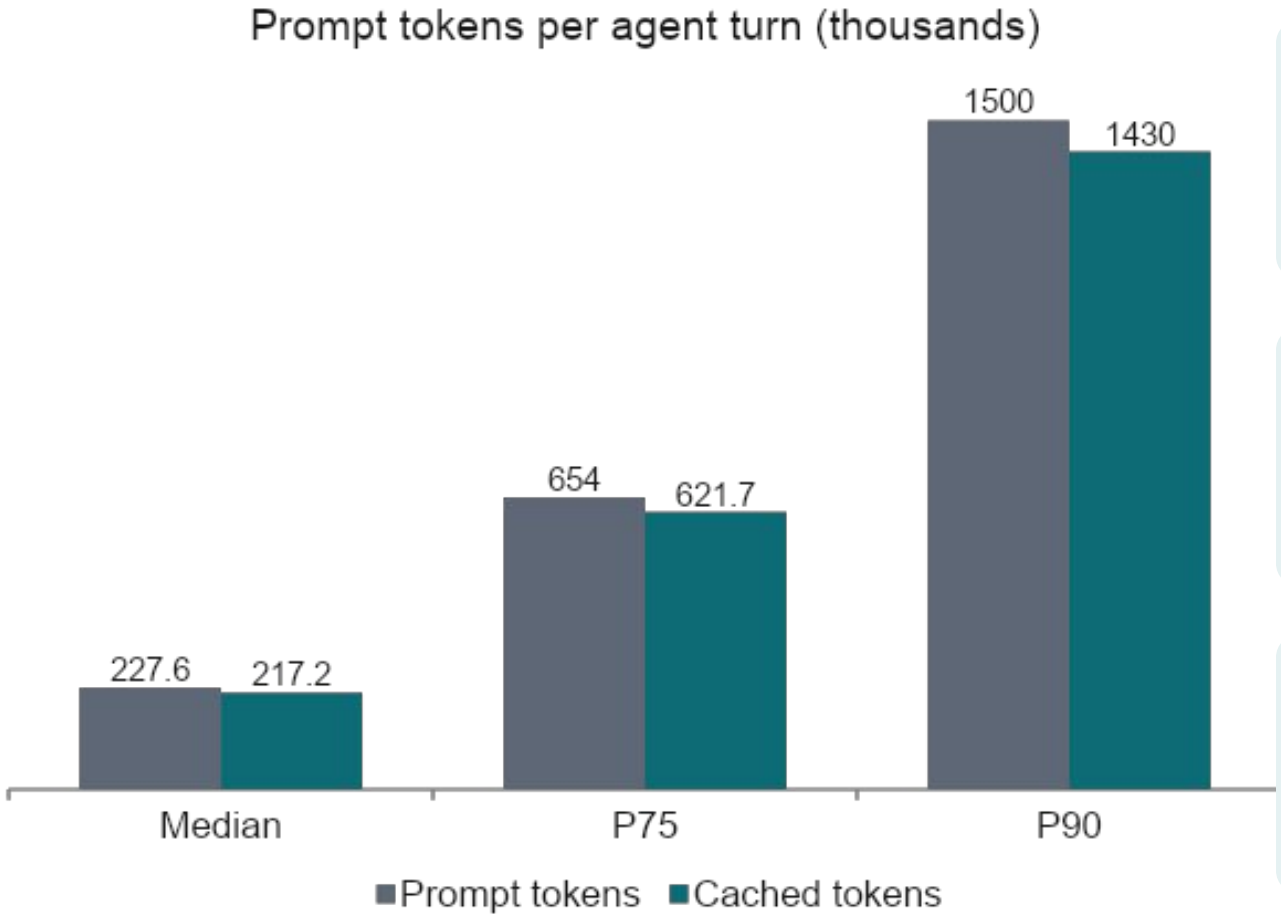
Per storage node: 20 × 30.73 TB SSD, ≥40 GB/s read, 5 GB/s write

Prefill on ordinary 8-GPU hosts, decode on 64–128-GPU supernodes, NVIDIA and Ascend

Dropping chat from 7 to 3 days costs only ~0.5–1% of hit rate

Source: github.com/deepseek-ai/DeepInfer, doc/inference_deployment_qa.md

Agent turns: long, and mostly cached



68K

median prompt per LLM call

95%

of a median turn's prompt is already cached (217K of 228K)

90% → 55%

cache hit rate within a turn vs across turn boundaries

Source: Liu et al., "Agentic Coding in the Wild: Characterizing GitHub Copilot at Production Scale," arXiv:2608.00101 (Table 4, §5)

Locality-oblivious, like Hadoop on S3

Then: analytics

- HDFS moved compute to the data
- Networks got fast enough to stop caring
- Object storage won: local reads traded for scheduling flexibility

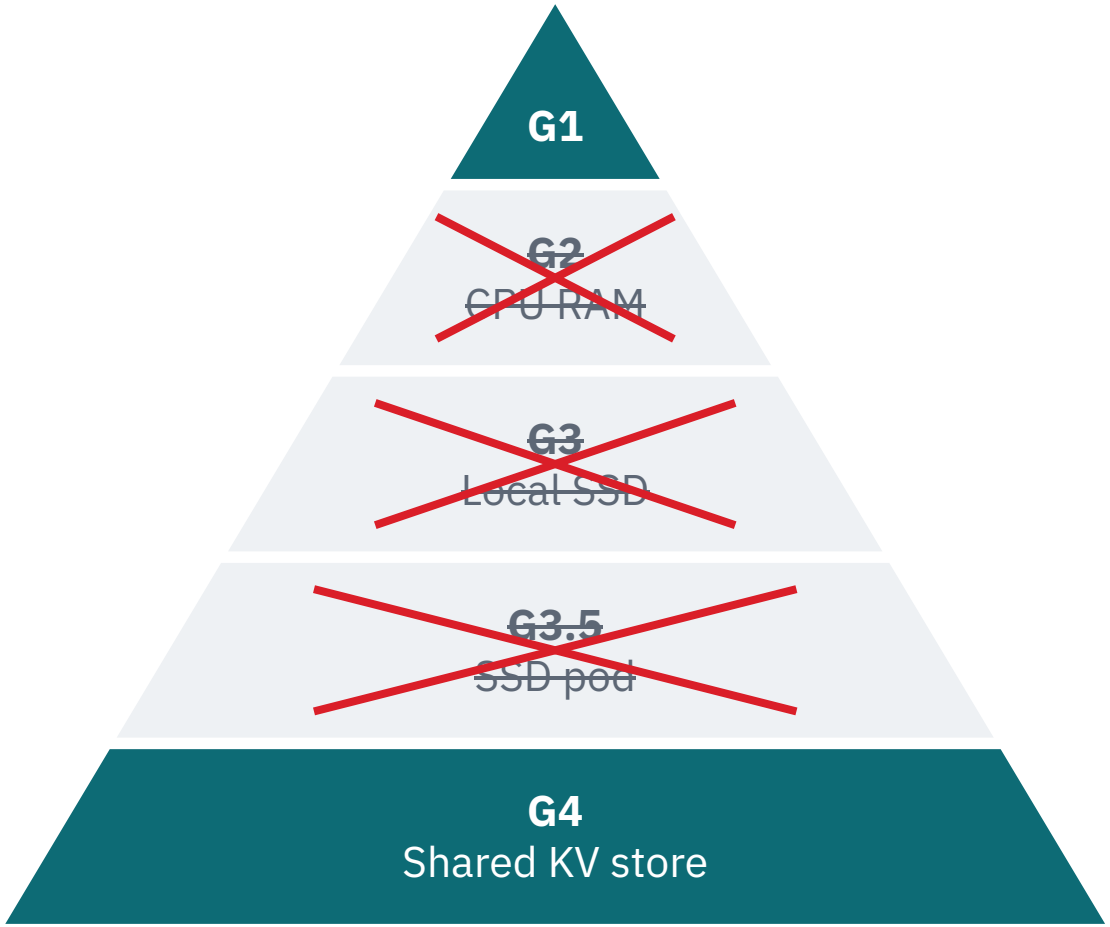
Now: KV cache in DeepInfer

- “Since any engine can hit any prefix, there is no prefix-affinity routing.”
- No DRAM cache tier; even SWA state is pulled over a 400G network
- Prefill and decode can split across datacenters (~400 Gb/s per 8 GPUs)

Storage brings the bytes: what matters is the path from storage to GPU

Source: github.com/deepseek-ai/DeepInfer, doc/inference_deployment_qa.md

No local cache, so GPUs are fungible



GPU HBM
Paged KV for the requests running now

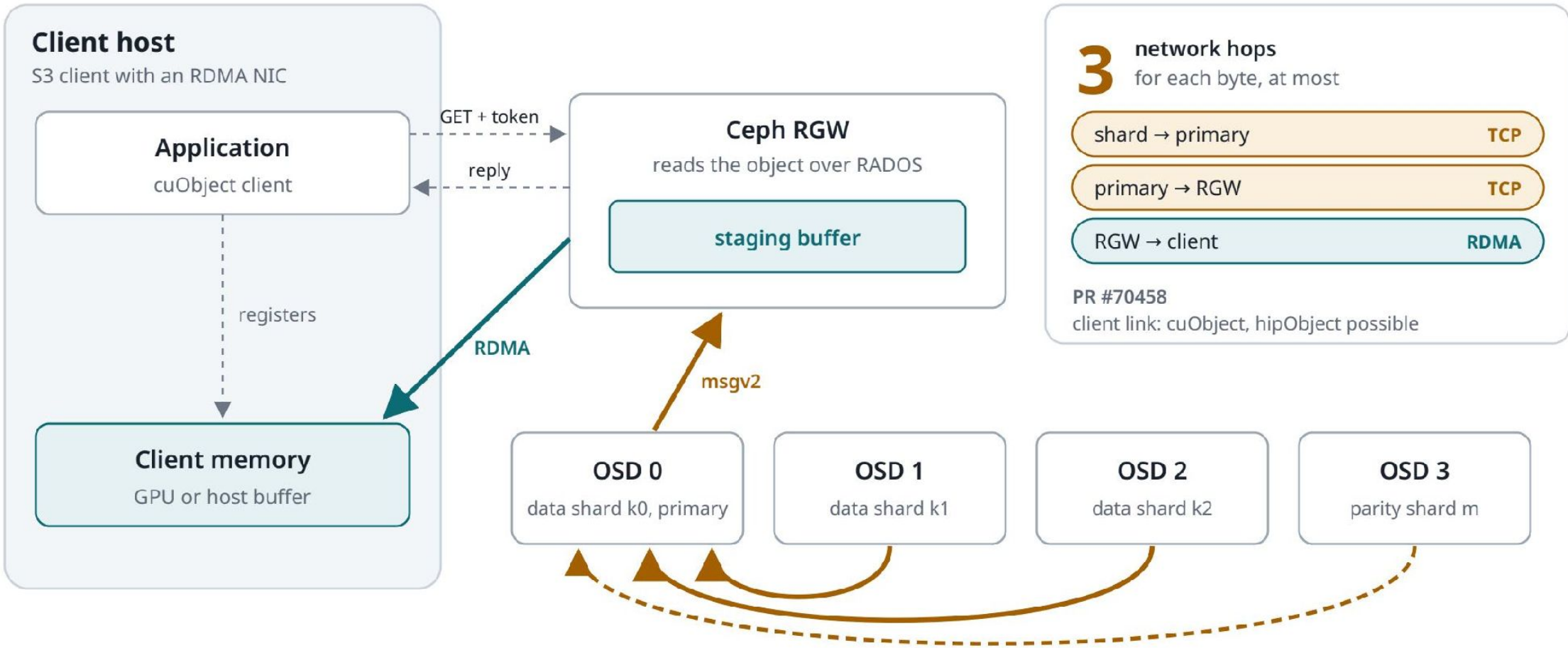
Not a cache tier
SWA state only; DeepInfer reads still stage here, S3 RDMA skips it

No local SSD cache
Any engine can serve any prefix, so no host holds state worth protecting

No pod-pinned flash
One store for every engine, no prefix-affinity routing

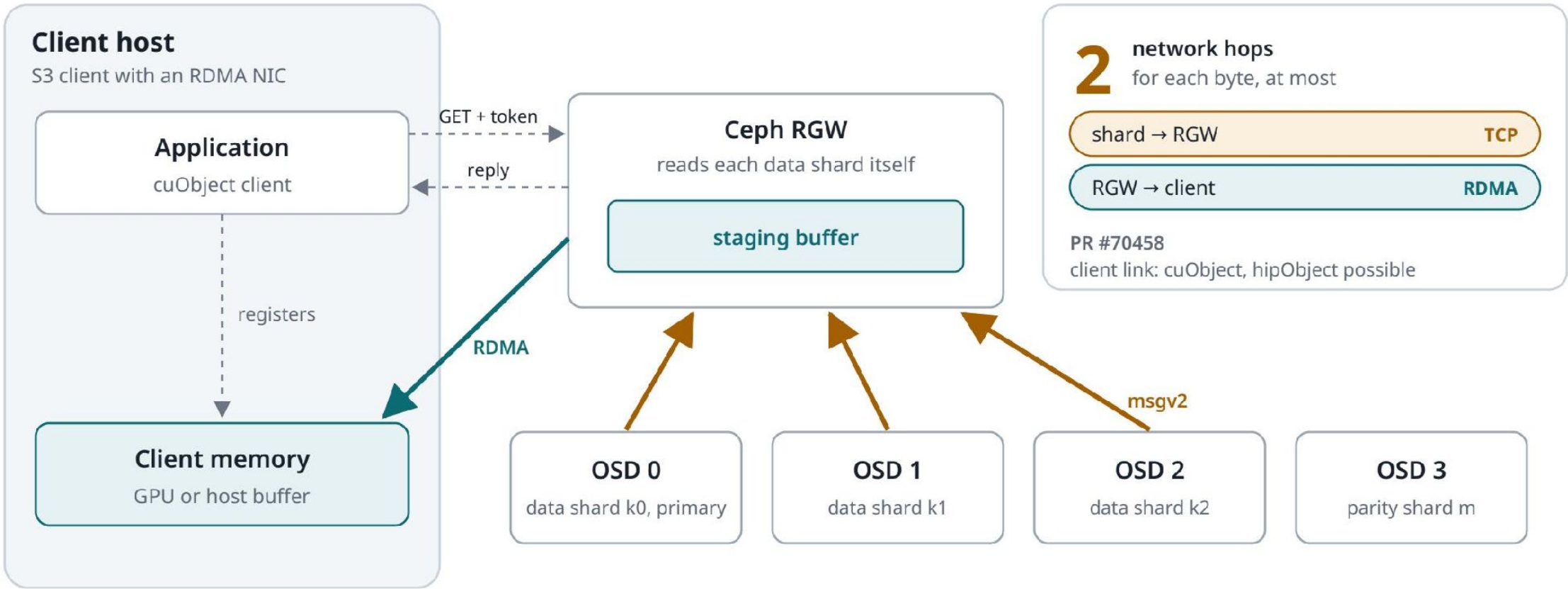
3FS in DeepInfer, or Ceph S3 over RDMA
~100 PiB, 10+2 EC, 3–7 day TTL

Tiers: NVIDIA Dynamo KV cache offloading (G1–G4), after TrendForce, June 2026. DeepSeek details: DeepInfer, doc/inference_deployment_qa.md



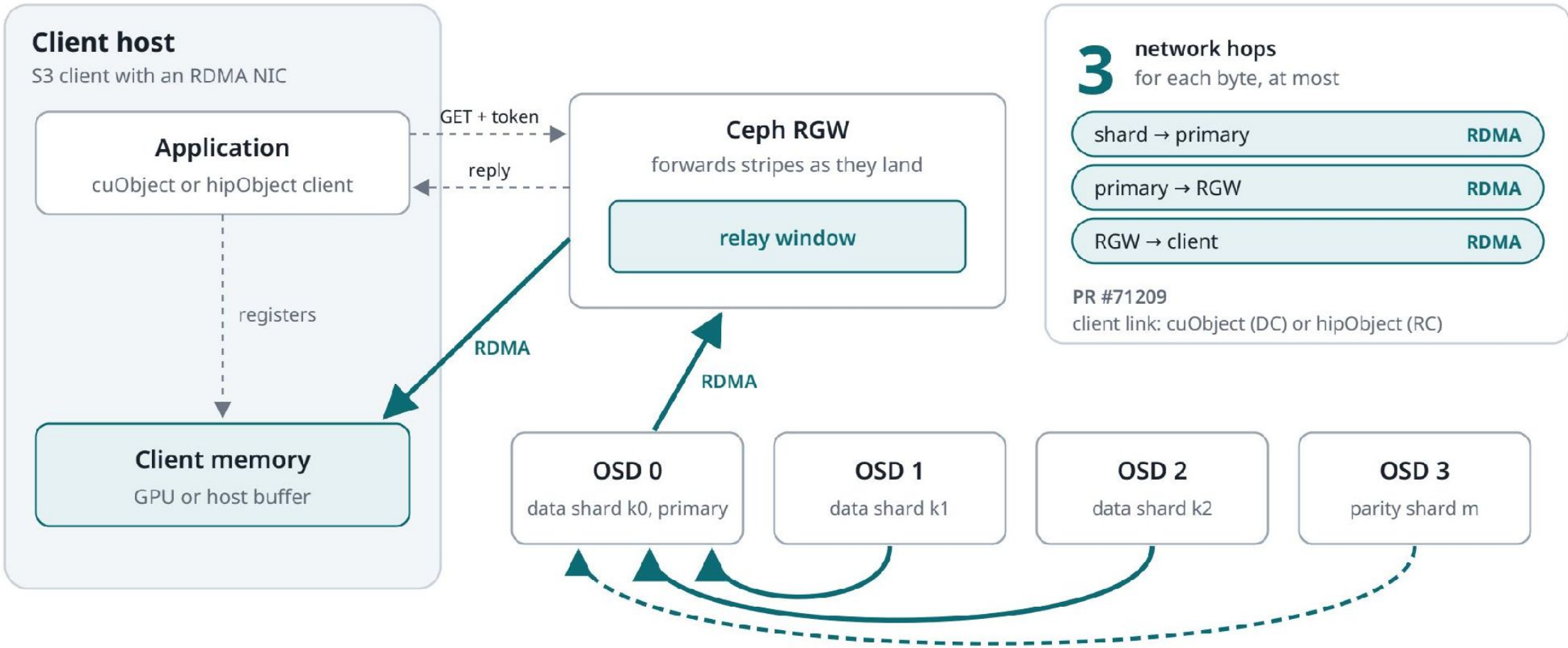
The primary gathers k1 and k2 over the messenger. When a data shard is down, the primary also reads parity and decodes.

data over the messenger (msgv2, TCP) · data over RDMA · only when a data shard is down · request or reply



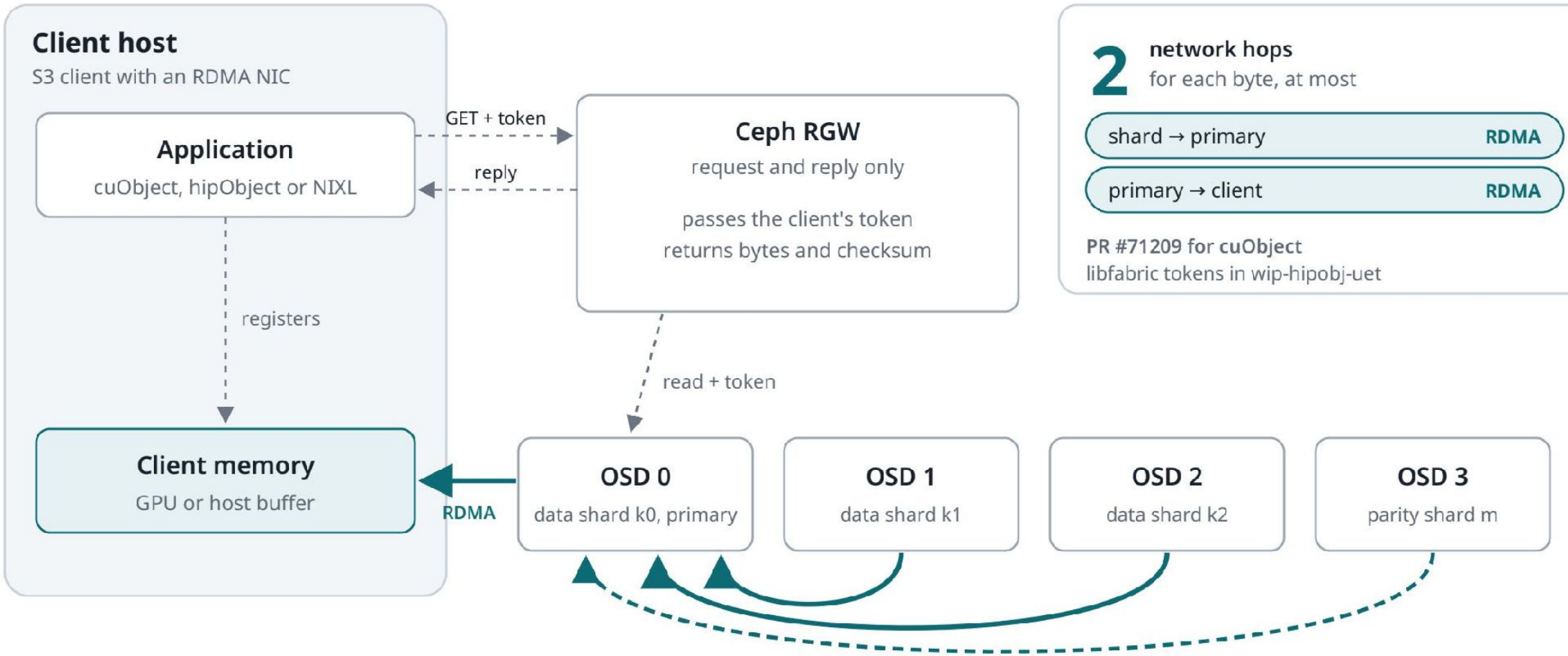
RGW reads each data shard from its own OSD, so no gather runs at the primary.
A shard that cannot read sends the read back to the primary.

data over the messenger (msgv2, TCP) data over RDMA - - - - request or reply



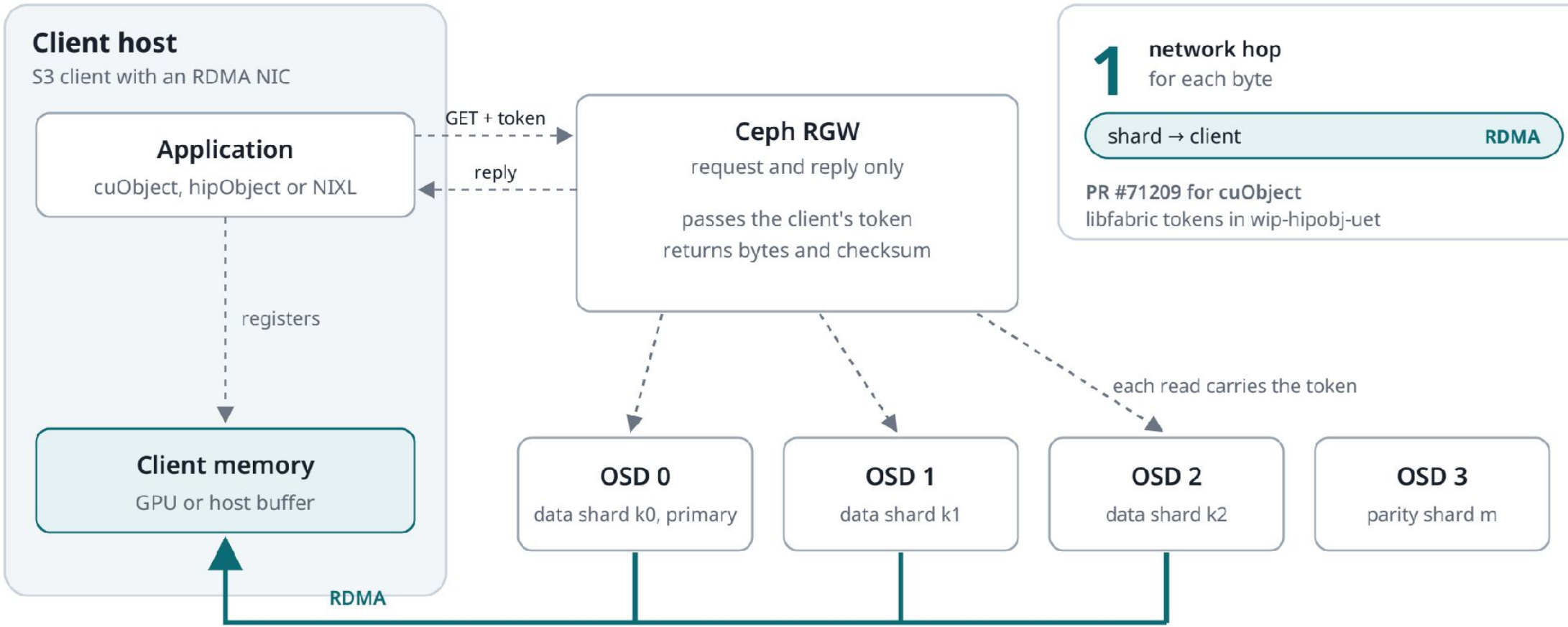
The shards write into the primary over RDMA (osd_oob_gather). When a data shard is down, the primary also reads parity and decodes.

data over the messenger (msgv2, TCP) — data over RDMA — only when a data shard is down - - - - request or reply



The primary gathers k1 and k2 over RDMA and writes each stripe into client memory.
When a data shard is down, the primary also reads parity and decodes.

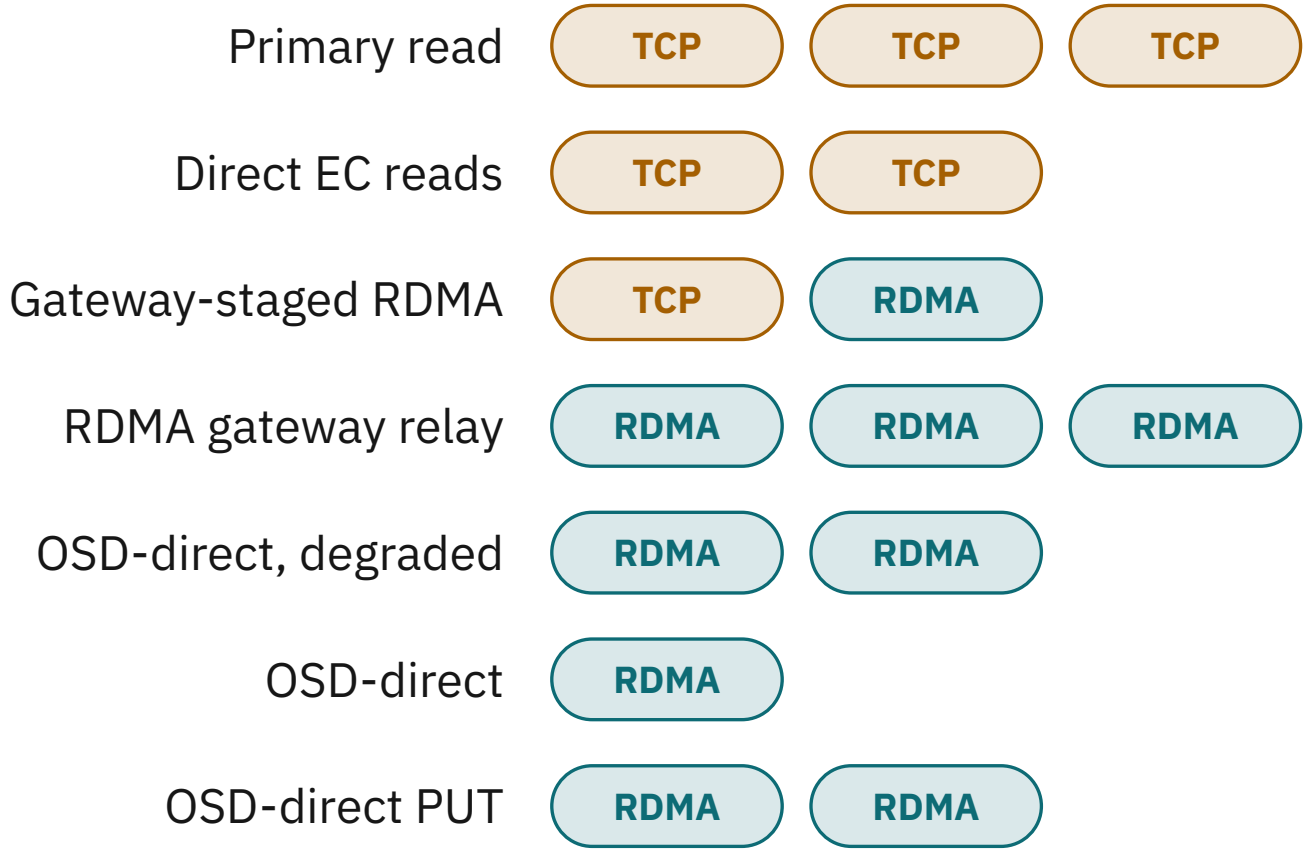
data over the messenger (msgv2, TCP) — data over RDMA — only when a data shard is down - - - - request or reply



Each data shard OSD writes its chunks to their place in client memory.
A shard that cannot read sends the read back to the primary.

data over the messenger (msgv2, TCP) — data over RDMA - - - - request or reply

Network hops for each byte



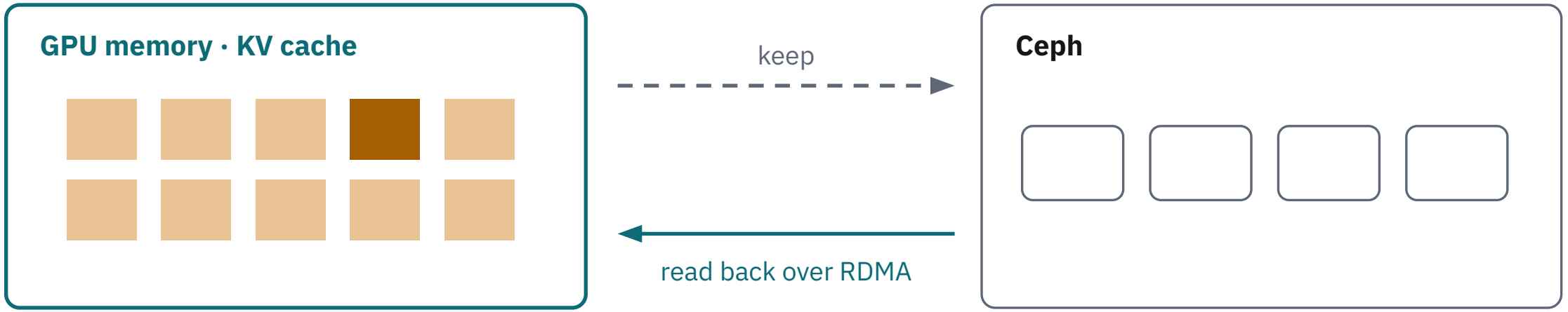
Degraded

The primary reads parity and decodes; the healthy shards still write direct

PUT

The primary reads client memory over RDMA, encodes, and pushes the shards

ECC for context memory

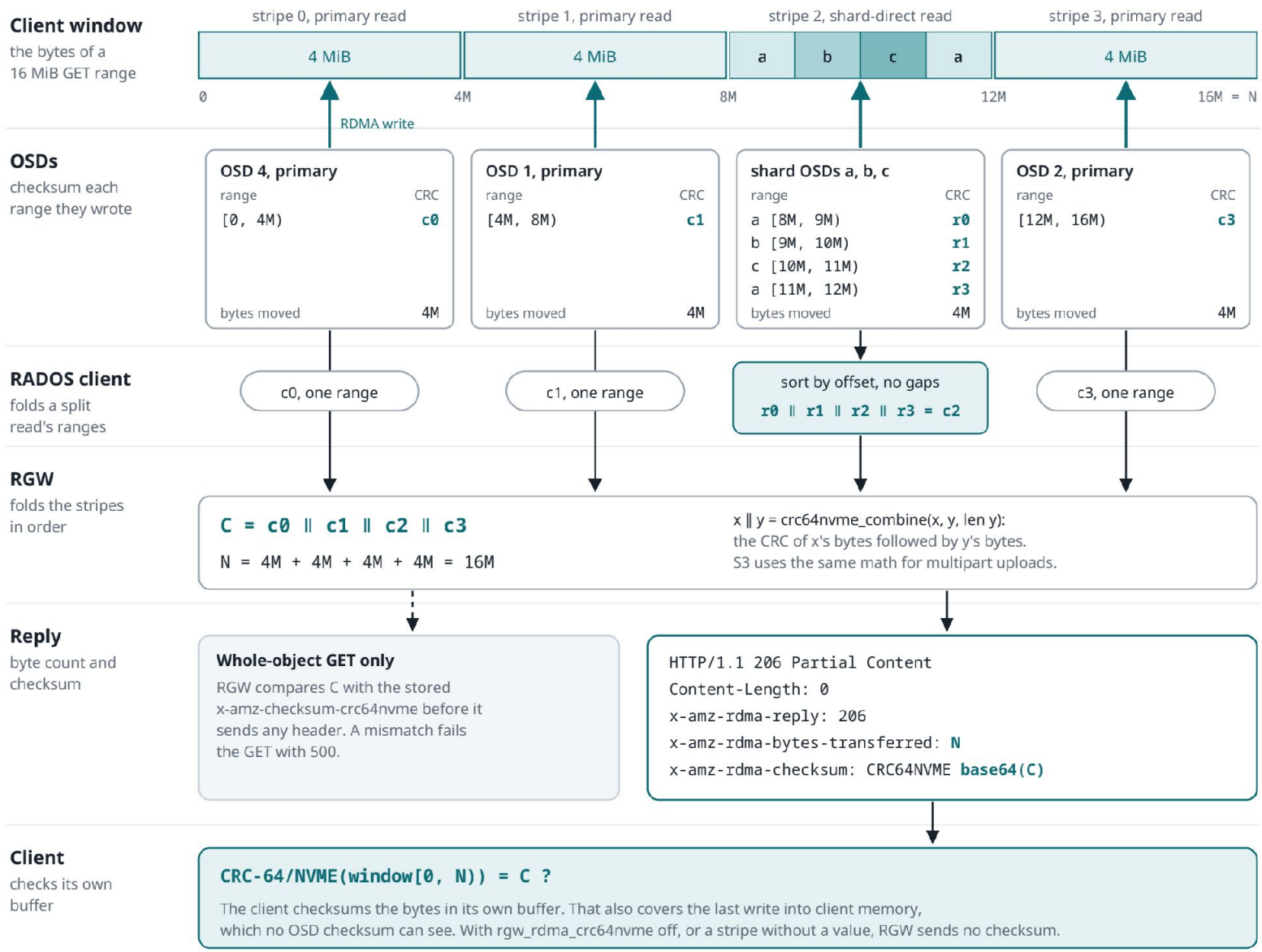


The answer is 47

No error, no crash: a flipped bit in KV cache just changes the output

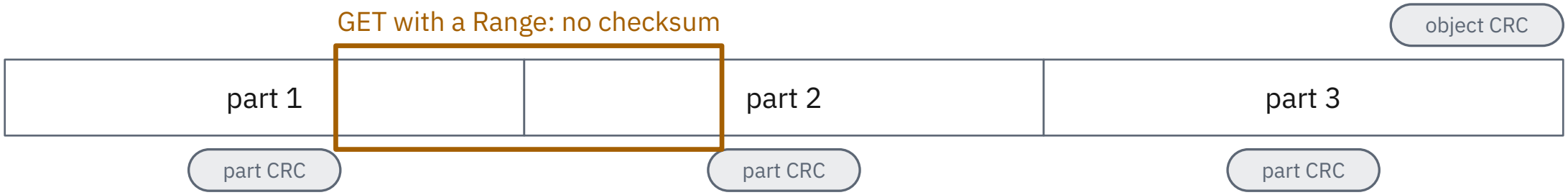
disk · BlueStore	OSD · each range	gateway · fold	client · own buffer
✓ CRC-32C	**✓ CRC-64/NVME**	**✓ math**	**✓ compare**

Erasure coding repairs the disk; the checksums cover the whole way into GPU memory

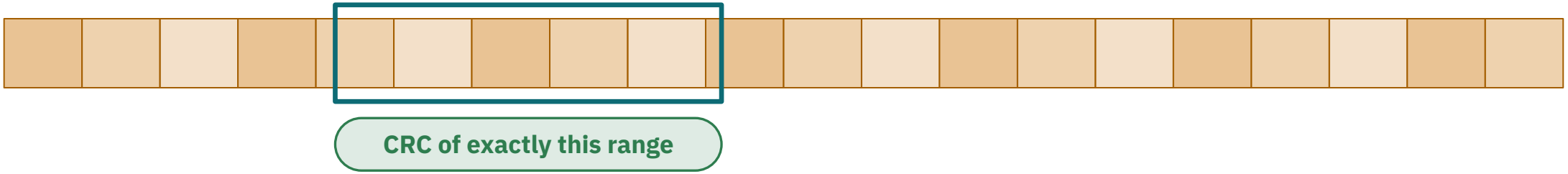

Client window
the bytes of a 16 MiB GET range
stripe 0, primary read
stripe 1, primary read
stripe 2, shard-direct read
stripe 3, primary read
4 MiB
4 MiB
a
b
c
a
4 MiB
0
4M
8M
12M
16M = N
RDMA write
OSDs
checksum each range they wrote
OSD 4, primary
range
CRC
[0, 4M)
c0
bytes moved
4M
OSD 1, primary
range
CRC
[4M, 8M)
c1
bytes moved
4M
shard OSDs a, b, c
range
CRC
a [8M, 9M)
r0
b [9M, 10M)
r1
c [10M, 11M)
r2
a [11M, 12M)
r3
bytes moved
4M
OSD 2, primary
range
CRC
[12M, 16M)
c3
bytes moved
4M
RADOS client
folds a split read's ranges
c0, one range
c1, one range
sort by offset, no gaps
r0 ‖ r1 ‖ r2 ‖ r3 = c2
c3, one range
RGW
folds the stripes in order
C = c0 ‖ c1 ‖ c2 ‖ c3
N = 4M + 4M + 4M + 4M = 16M
x ‖ y = crc64nvme_combine(x, y, len y):
the CRC of x's bytes followed by y's bytes.
S3 uses the same math for multipart uploads.
Reply
byte count and checksum
Whole-object GET only
RGW compares C with the stored x-amz-checksum-crc64nvme before it sends any header. A mismatch fails the GET with 500.
HTTP/1.1 206 Partial Content
Content-Length: 0
x-amz-rdma-reply: 206
x-amz-rdma-bytes-transferred: N
x-amz-rdma-checksum: CRC64NVME base64(C)
Client
checks its own buffer
CRC-64/NVME(window[0, N)) = C ?
The client checksums the bytes in its own buffer. That also covers the last write into client memory,
which no OSD checksum can see. With rgw_rdma_crc64nvme off, or a stripe without a value, RGW sends no checksum.

A checksum for any byte range

Amazon S3

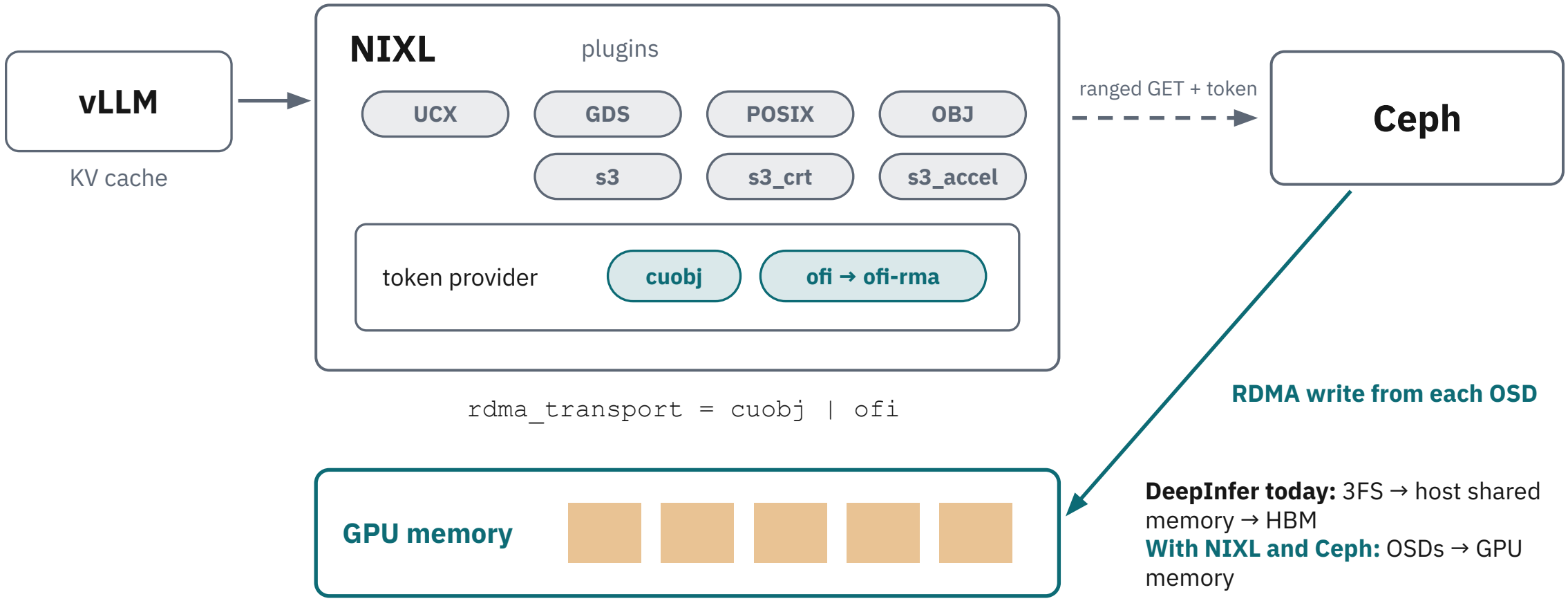


Ceph, S3 over RDMA



A KV block or a Parquet column chunk lands in GPU memory with its own checksum

vLLM to Ceph through NIXL



DeepInfer today: 3FS → host shared memory → HBM
With NIXL and Ceph: OSDs → GPU memory

Test Environment

Single client and cluster host, each with

- 1x AMD Genoa Epyc 9454 48C/96T
- 4x 128GB DDR5 4800 MT/s
- 1x NVIDIA ConnectX-7 dual-port 200GbE

Cluster host media

- 30 GB/s NVMe

Ceph

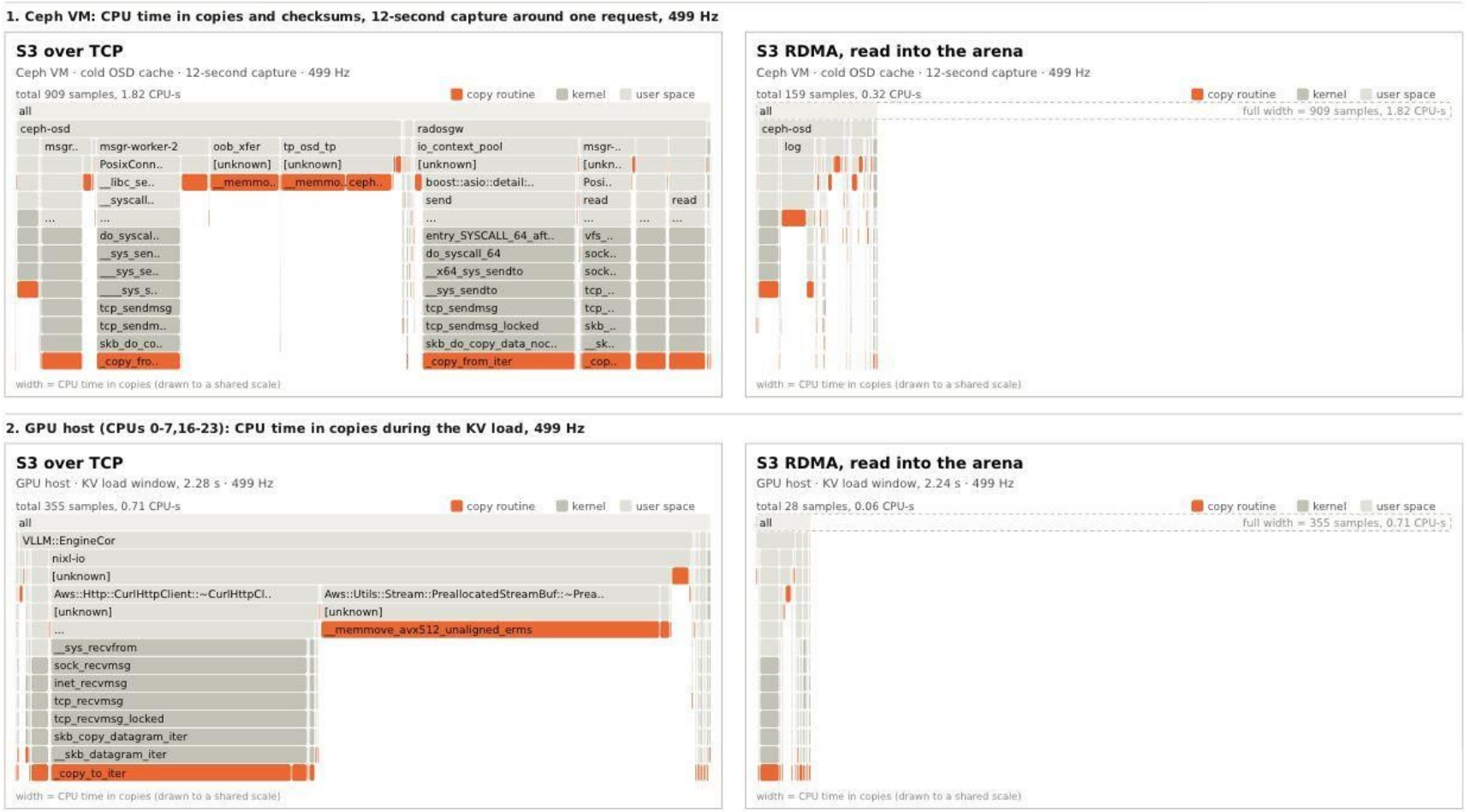
- RHEL 10.2
- DOCA mlx5 drivers
- libcuobject client/server
- vstart cluster (wip-rgw-cuobj-osd)
  - buckets.data: 6+2 with 512 pgs + ec optimizations
  - buckets.index: 3x with 128 pgs
  - buckets.non-ec: 3x with 128 pgs

Headline Results



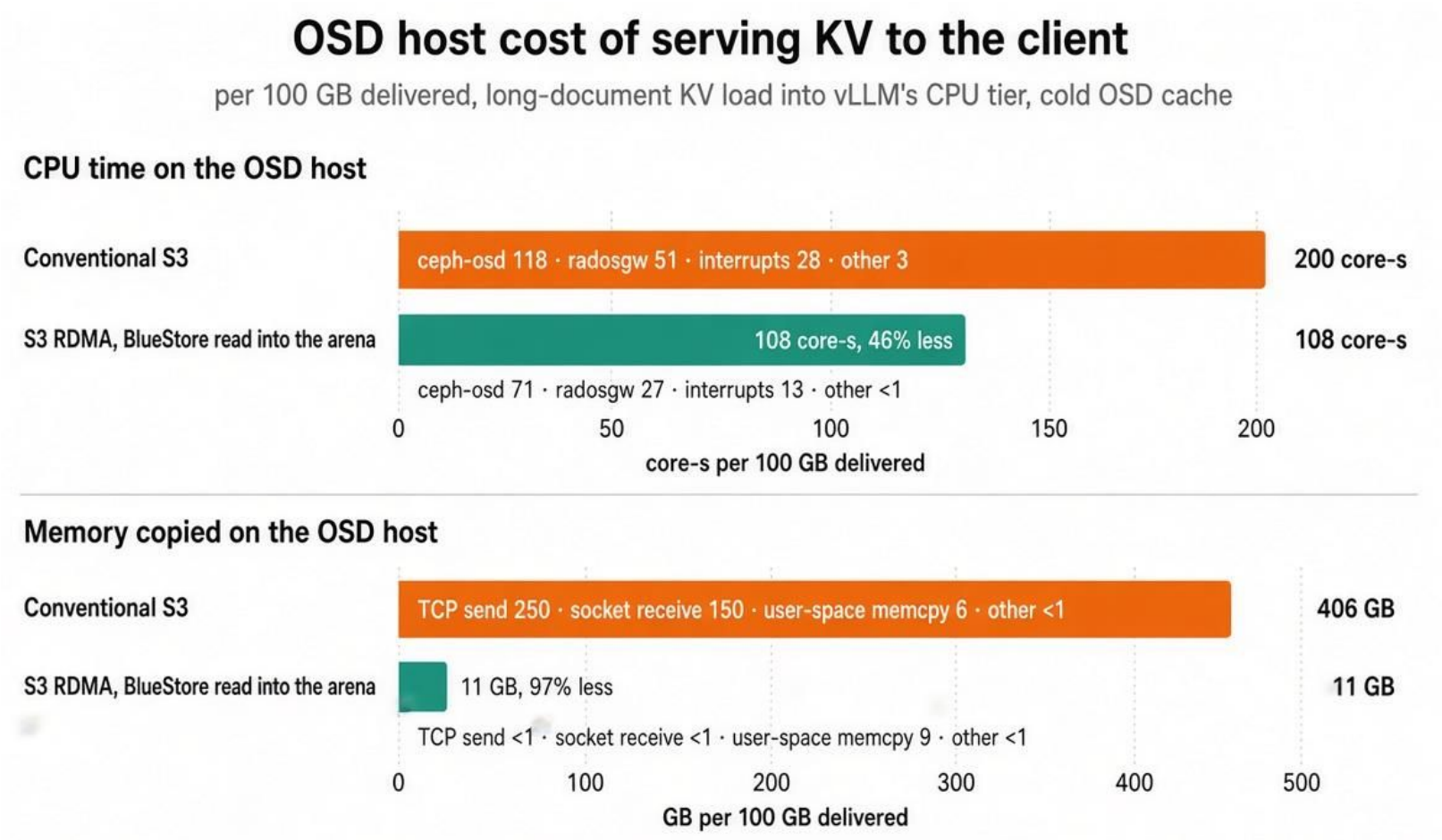
Hit throughput reaches 47.3 GB/s, 95% of the 2 × 200GbE rNIC limit

Ceph memcpy reduction



Measured locally on an AMD Strix Halo system with a single consumer NVMe SSD, Ceph in a VM, not the cluster on the Test Environment slide

100GB of KV cache blocks



Measured locally on an AMD Strix Halo system with a single consumer NVMe SSD, not the cluster on the Test Environment slide

Where the copies went

The copy budget for a 16 MiB object, memcpy ≥ 4 KiB counted in the OSDs

Gateway, OSD-direct GET and PUT	**0 bytes**
Replicated read, default	**1 copy**
Replicated read into the arena	**0 copies**
EC balanced read into the arena	**0 copies**
EC gather on the primary	**1 checked copy of each pushed shard**
Pulled write	**1 copy more than a plain write**

MB copied in the OSDs, 16 MiB write, 3 replicas

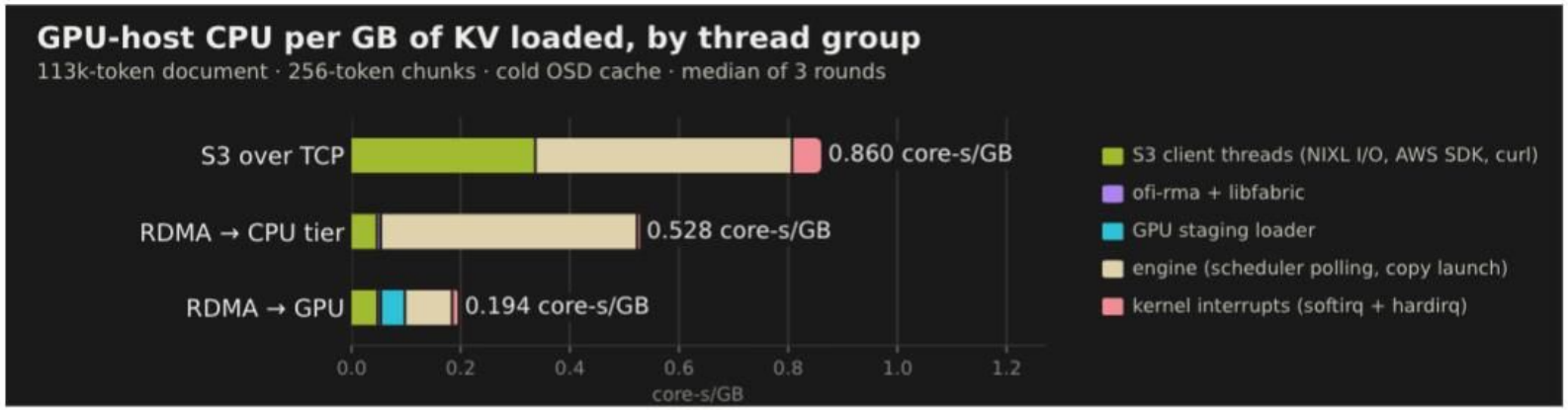


In the gateway

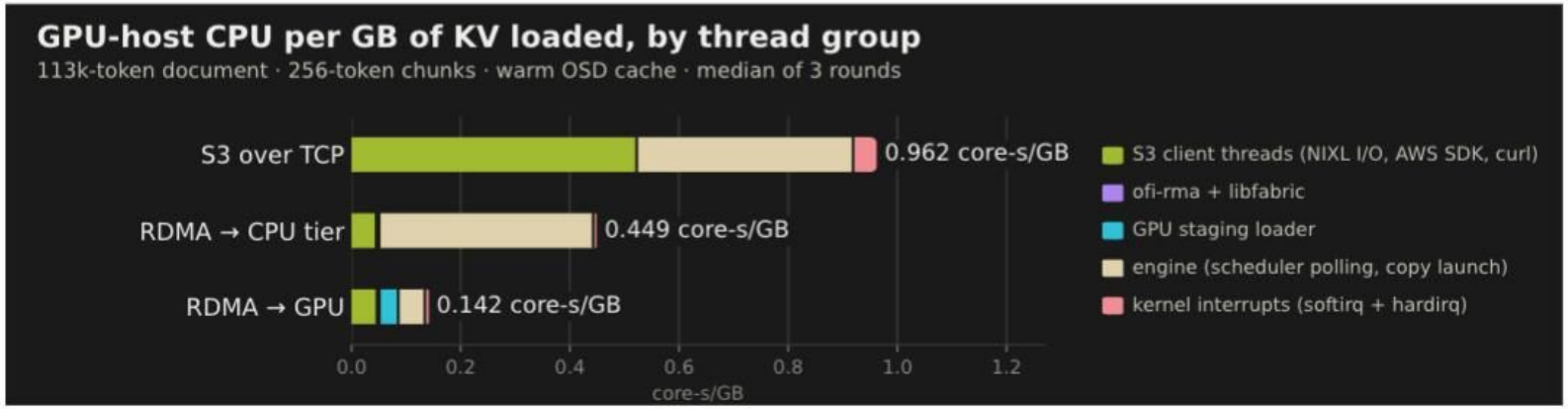
0 bytes copied, hashed or forwarded

GPU host efficiency

Cold OSD cache



Warm OSD cache



Measured locally on an AMD Strix Halo system with a single consumer NVMe SSD, Ceph in a VM, not the cluster on the Test Environment slide

References

This work

Feature branch
github.com/mmgaggle/ceph/tree/wip-s3-oob-umbrella

Split for review
wip-s3rdma-p0-fixes … wip-s3rdma-p8-read-into

Design and setup guide
doc/radosgw/s3-rdma.rst on the branch

Pull requests
ceph/ceph #70458 (gateway-staged) · #71209 (OSD-direct)

Client libraries
mmgaggle/ofi-rma · nixl (wip-xobj) · fastsafetensors (cuObject) · hipObject

UET provider
mmgaggle/uet-ref-prov (wip-libfabric-provider)

Cited

DeepSeek DeepInfer
github.com/deepseek-ai/DeepInfer, doc/inference_deployment_qa.md

GitHub Copilot traces
Liu, Qiu, Goiri, Fonseca, Bianchini, Choukse. “Agentic Coding in the Wild: Characterizing GitHub Copilot at Production Scale.” arXiv:2608.00101