DASYA

dasya.itu.dk

RAD

rad.itu.dk

IT UNIVERSITY OF COPENHAGEN

www.itu.dk

toward flexible & accessible I/O options for data systems

Pınar Tözün

Associate Professor
IT University of Copenhagen

Oracle
January 22, 2026

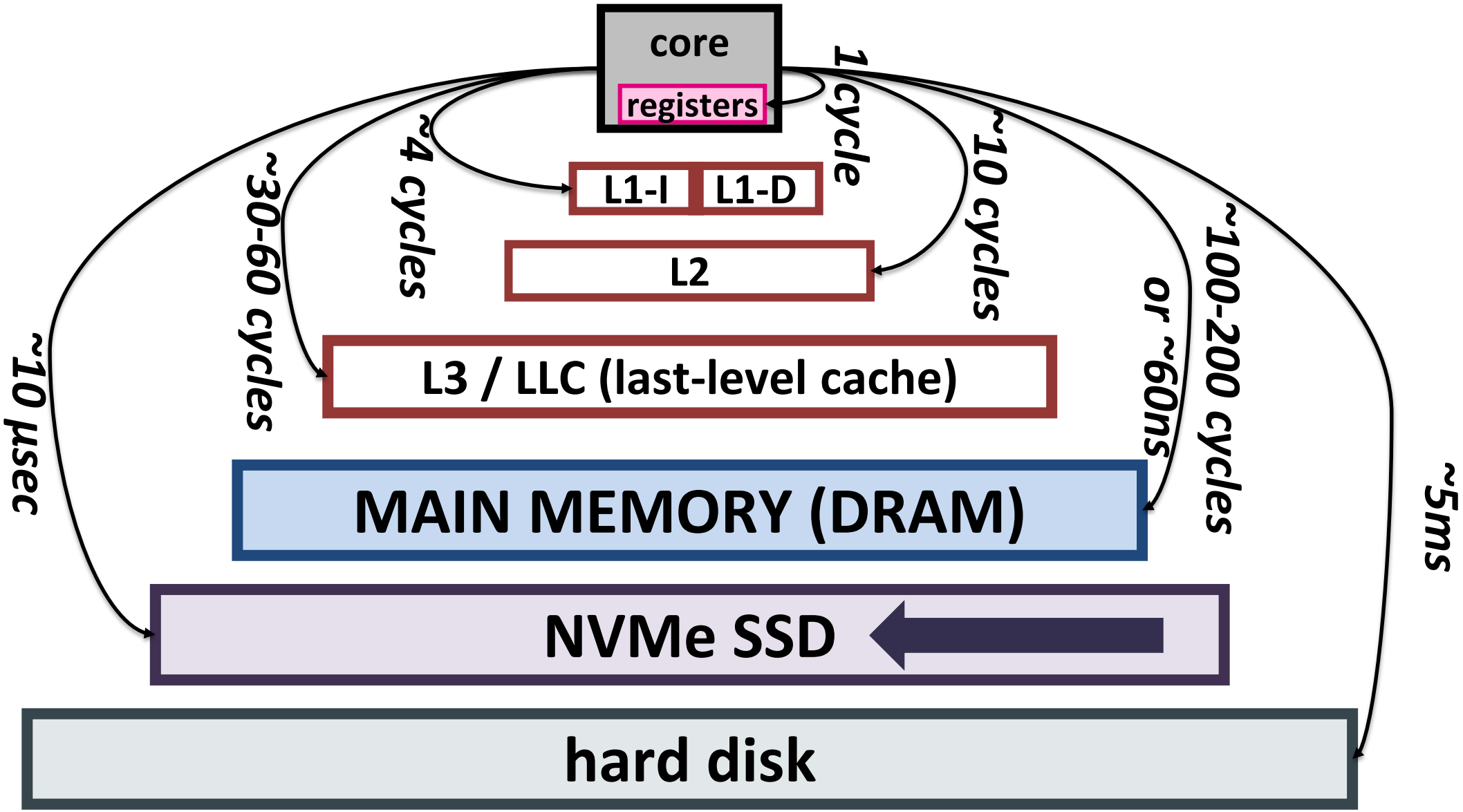
pito@itu.dk
pinartozun.com

Innovationsfonden

INDEPENDENT RESEARCH FUND DENMARK

novo nordisk foundation

storage hierarchy – directly attached



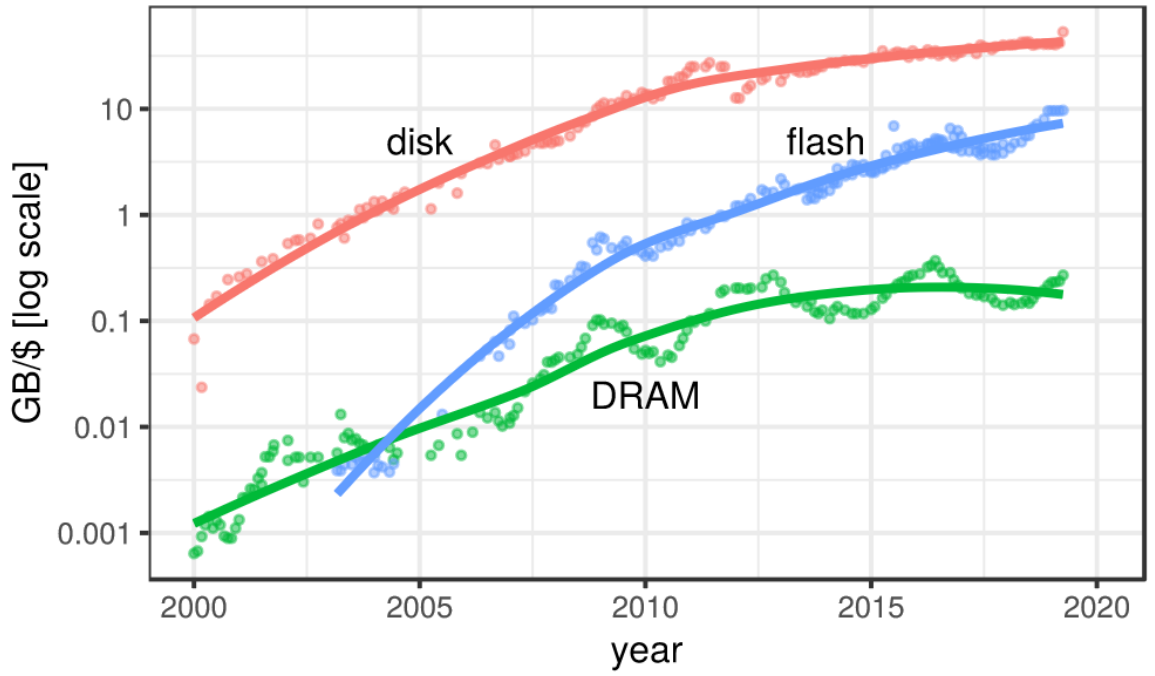
why SSD?

➔ except for SSDs, each layer stayed almost stable the past 15 years in terms of latency

- improvements on SSD internals
- from SAS/SATA to PCIe
- linux IO improvements
  e.g., multiqueue, io-uring

➔ improved price/capacity

source: Haas et al., CIDR 2020



➔ led to several SSD-optimized data systems

- RocksDB, BwTree, LeanStore, Umbra …

shift from pure in-memory-optimized to SSD-optimized data systems!

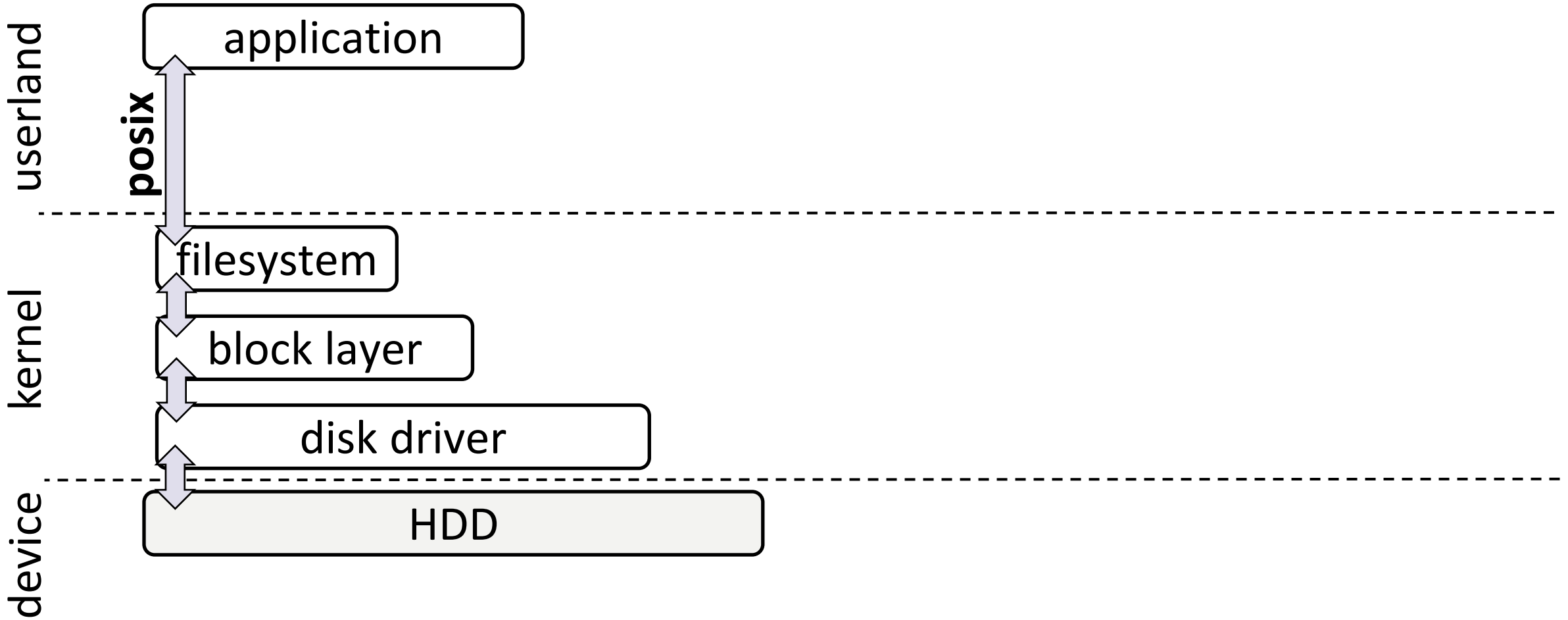
navigating I/O options for data systems

- Flexible I/O for Database Management Systems with xNVMe
  E. Houlborg, A. N. Tietgen, S. A. F. Lund, M. Weisgut, T. Rabl, J. Gonzalez, V. Shah, P. Tözün
  CIDR 2026

- Path to GPU-Initiated I/O for Data-Intensive Systems
  K. B. Torp, S. A. F. Lund, P. Tözün
  DaMoN 2025

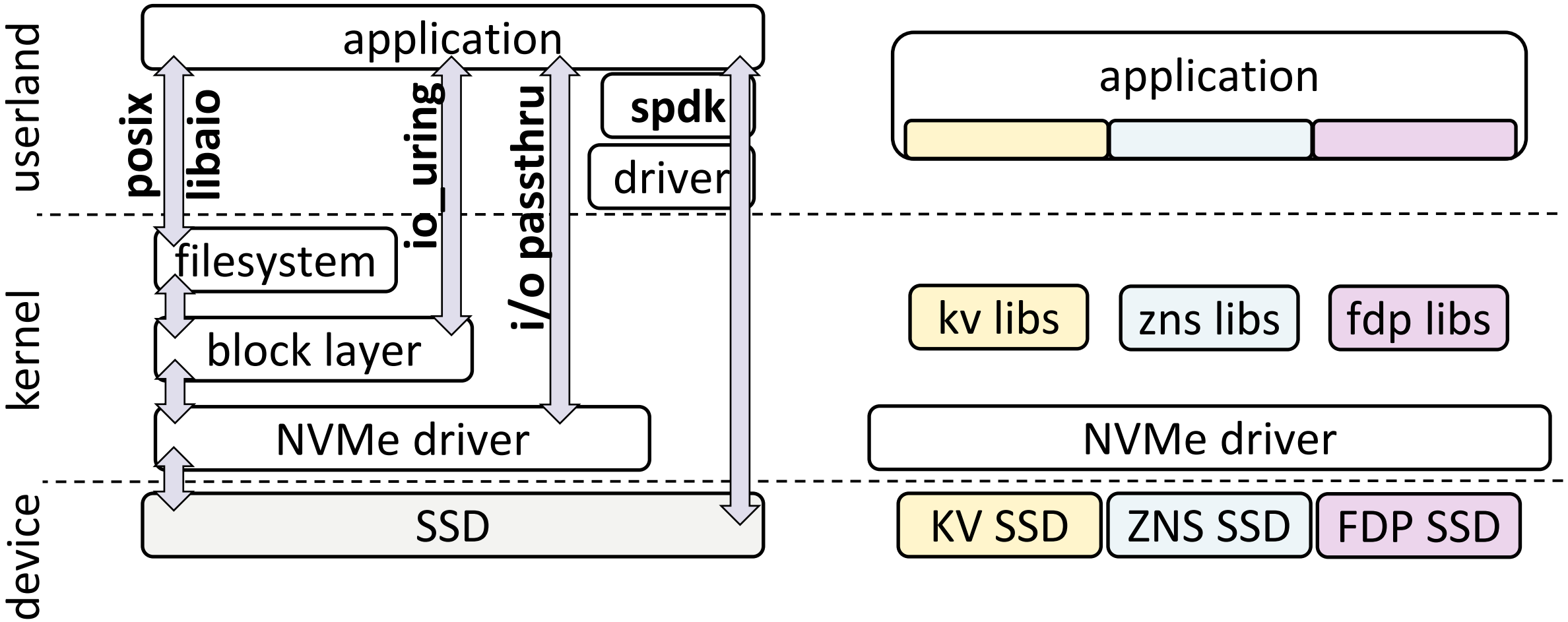
work done in collaboration with

once upon a time …



traditional I/O path was designed for hard disks

today – (non-exhaustive) SSD landscape



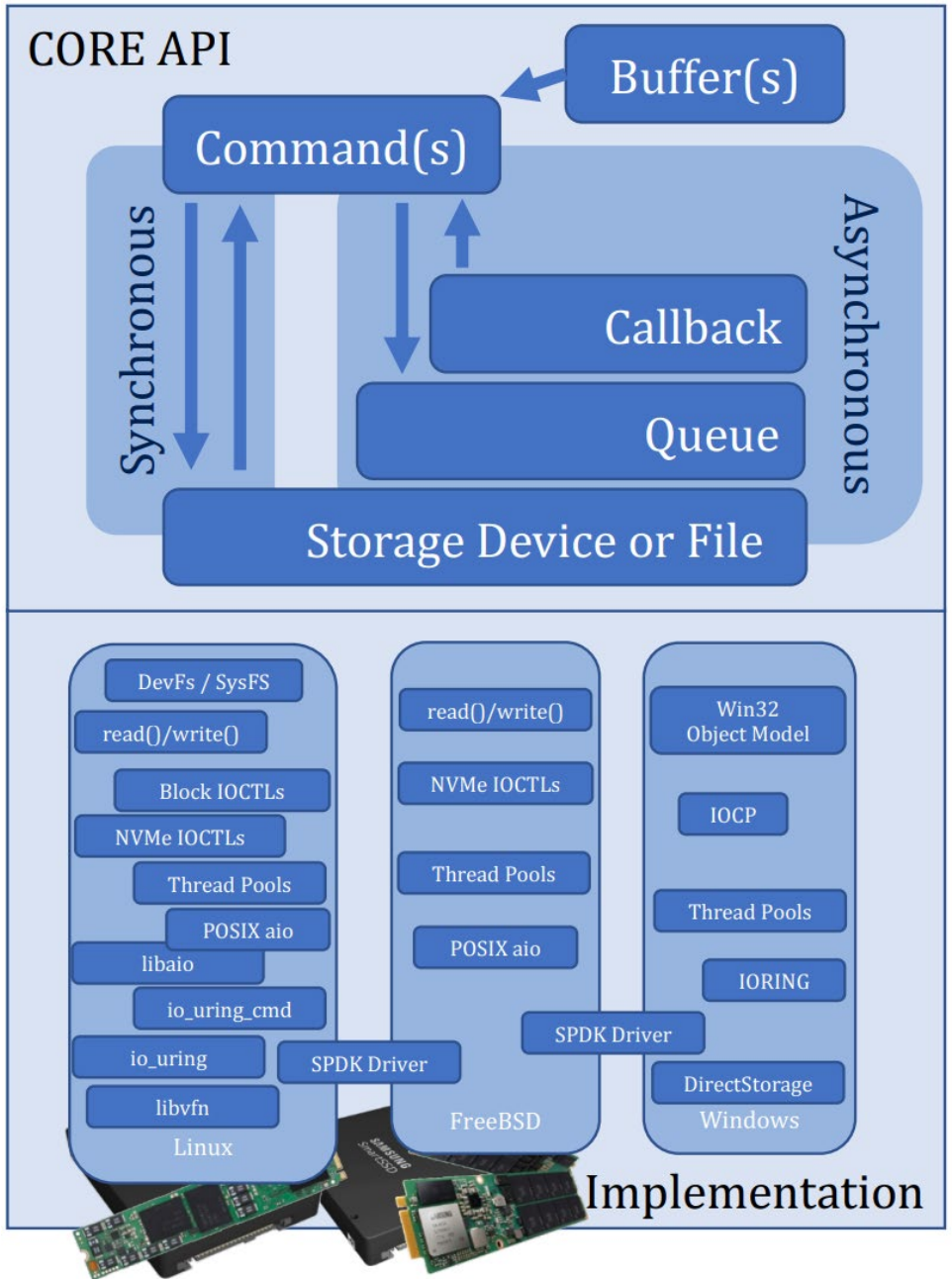
data systems tend to avoid dealing with the landscape

target ➔ “generic” SSDs & synchronous POSIX I/O

a way to approach the landscape

xNVMe

xnvme.io



- C API
  - C++ can directly use
  - bindings for rust & python
- selection of an I/O path …
  - done by ***libxnvme*** based on available paths
  - or specified by the user

I/O Interface Independence with xNVMe

Simon A. F. Lund
Samsung
Copenhagen, Denmark
simon.lund@samsung.com

Philippe Bonnet
IT University of Copenhagen
Copenhagen, Denmark
phbo@itu.dk

Klaus B. A. Jensen
Samsung
Copenhagen, Denmark
k.jensen@samsung.com

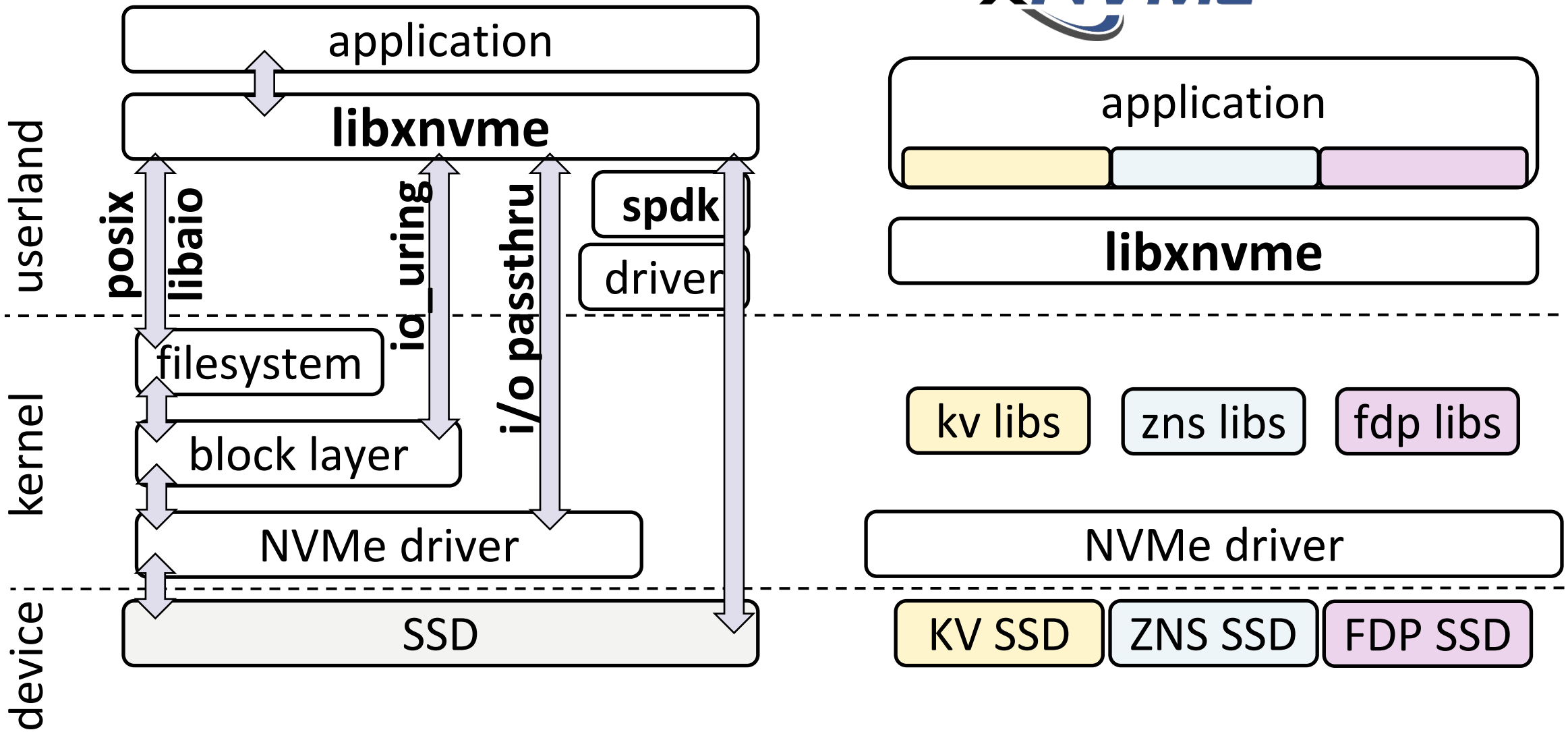
Javier Gonzalez
Samsung
Copenhagen, Denmark
javier.gonz@samsung.com

SYSTOR'22

a way to approach the landscape

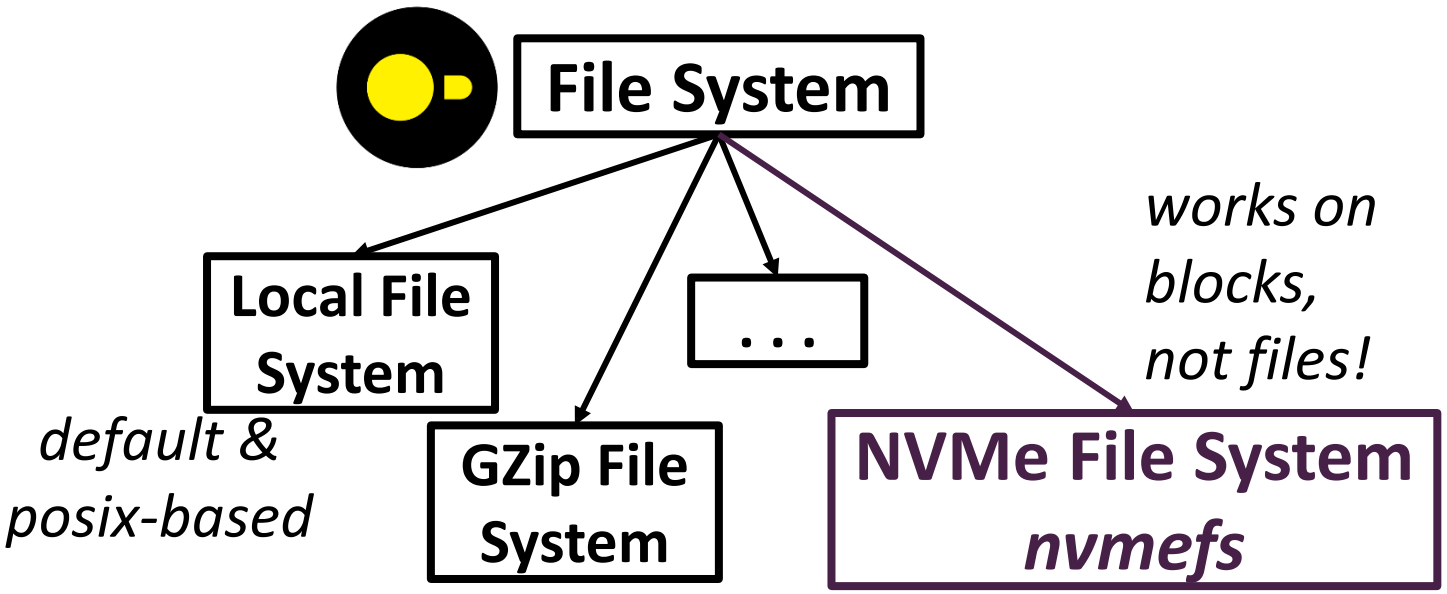
xNVMe

xnvme.io



can xNVMe give database systems more I/O flexibility?

xNVMe integration into DuckDB




```
CREATE PERSISTENT SECRET nvmefs (
  TYPE NVMEFS,
  nvme_device_path '/dev/ng1n1',
  backend          'io_uring_cmd'
);
```

```
ATTACH DATABASE 'nvmefs://example.db'
AS nvme (READ_WRITE);
```

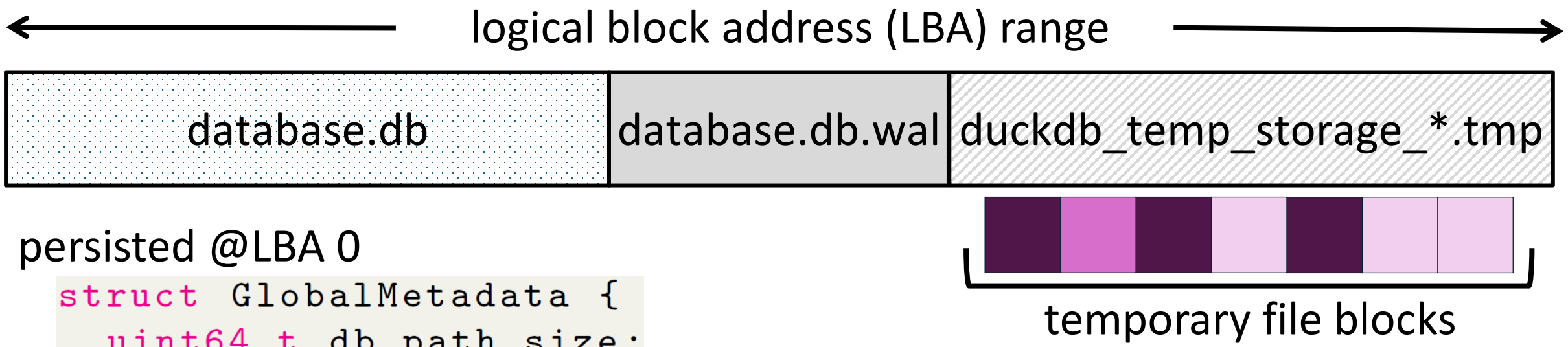
× can get better performance if integrated into DuckDB core

e.g., file system calls are still sync even if *nvmefs* issues async I/O

➔ **our goal is accessible I/O diversity first!**

no impact on DuckDB core & minimal impact on usage!

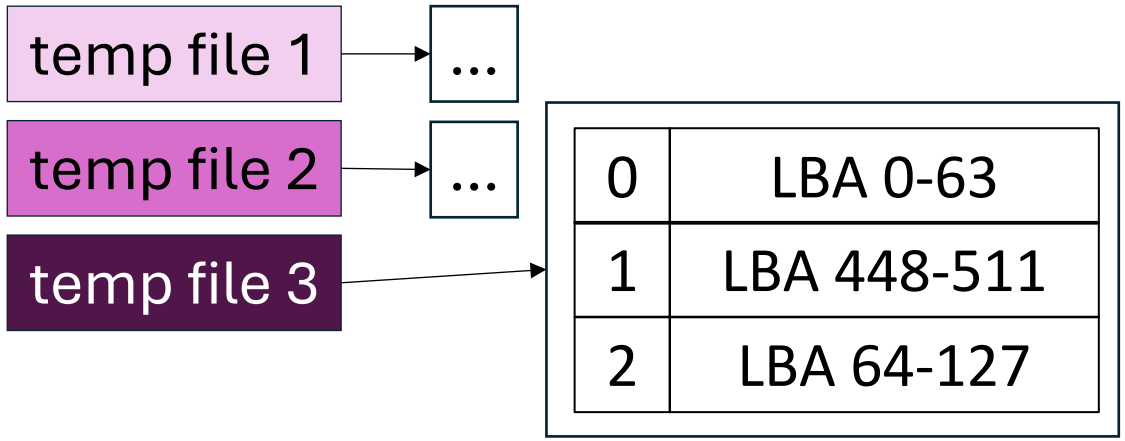
LBA management in *nvmefs*



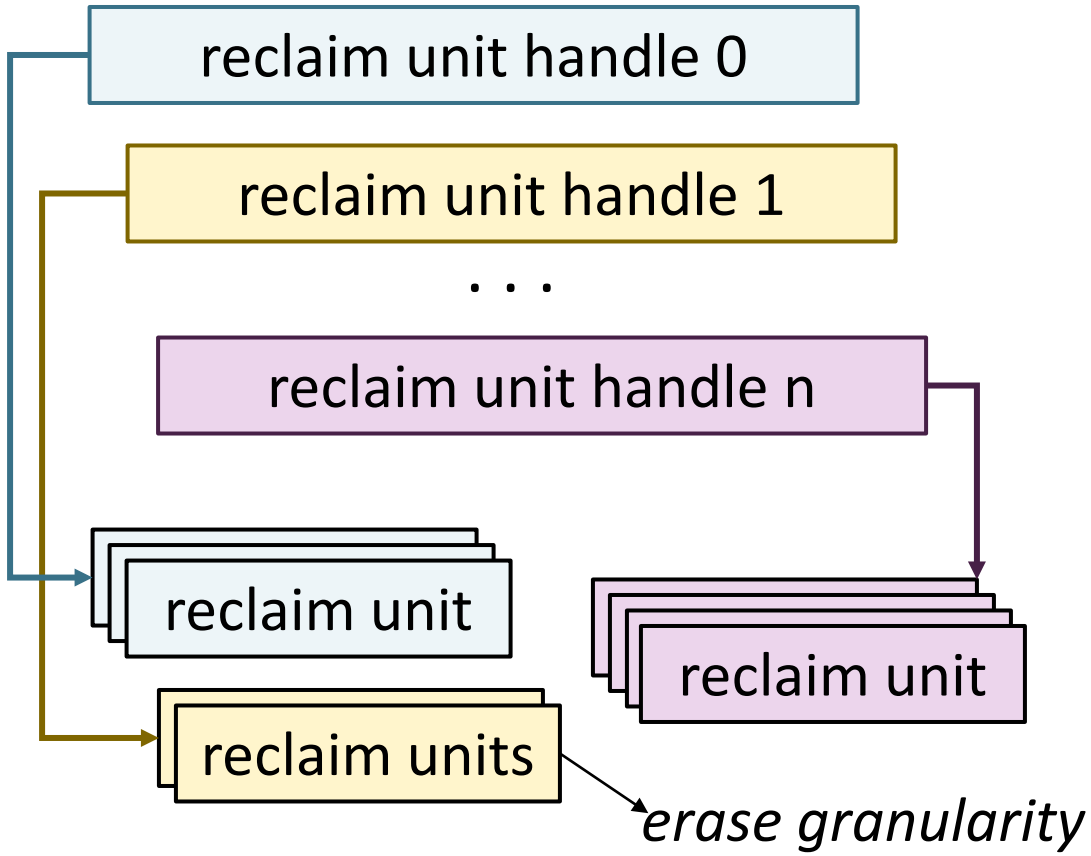
persisted @LBA 0

```
struct GlobalMetadata {
  uint64_t db_path_size;
  char db_path[101];

  uint64_t db_start;
  uint64_t wal_start;
  uint64_t tmp_start;

  uint64_t db_location;
  uint64_t wal_location;
};
```

Temporary File Metadata Manager



FDP (flexible data placement) SSD support



Towards Efficient Flash Caches with Emerging NVMe Flexible Data Placement SSDs

Michael Allison, Arun George, Javier Gonzalez, Dan Helmick, Vikash Kumar, Roshan R Nair, Vivek Shah*
Samsung Electronics

EuroSys'25

nvmefs gives access to FDP SSD commands via xNVMe

preliminary FDP use:

- database data is ***long-lived***
- temporary file blocks are ***short-lived (per query)***
- different reclaim unit handles for database & temporary data

➔ separates their garbage collection

evaluation

- **baseline**
  DuckDB LocalFileSystem – posix
- **DuckDB *nvmefs***
  xnvme – I/O passthru
  xnvme – I/O passthru – FDP
  xnvme – spdk

on

- TPC-H
- aggregation benchmark

Robust External Hash Aggregation in the Solid State Age **ICDE'24**

Laurens Kuiper
CWI, Amsterdam, Netherlands
laurens.kuiper@cwi.nl

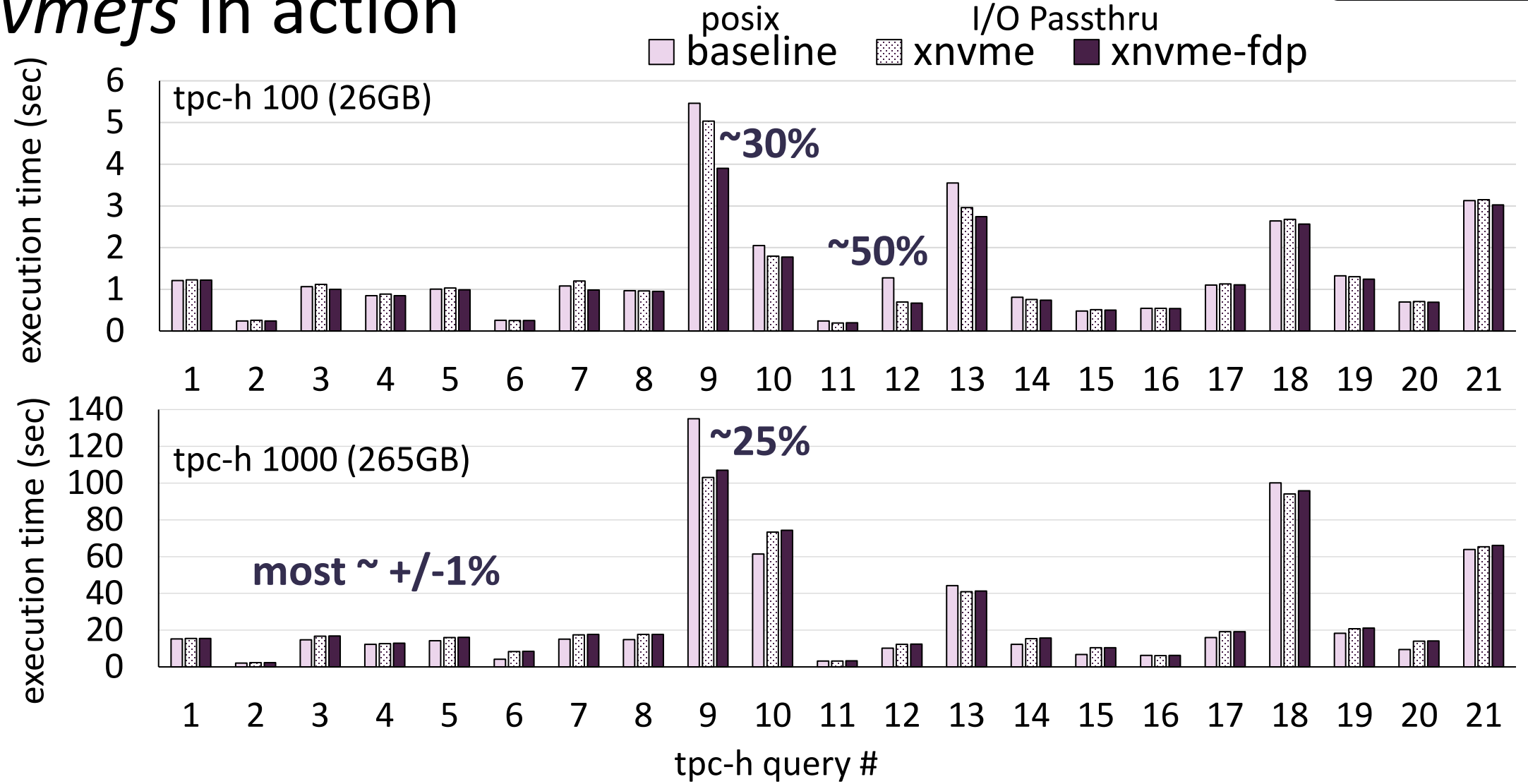
Peter Boncz
CWI, Amsterdam, Netherlands
peter.boncz@cwi.nl

Hannes Mühleisen
CWI, Amsterdam, Netherlands
hannes.muehleisen@cwi.nl

@ Samsung Memory Research Center

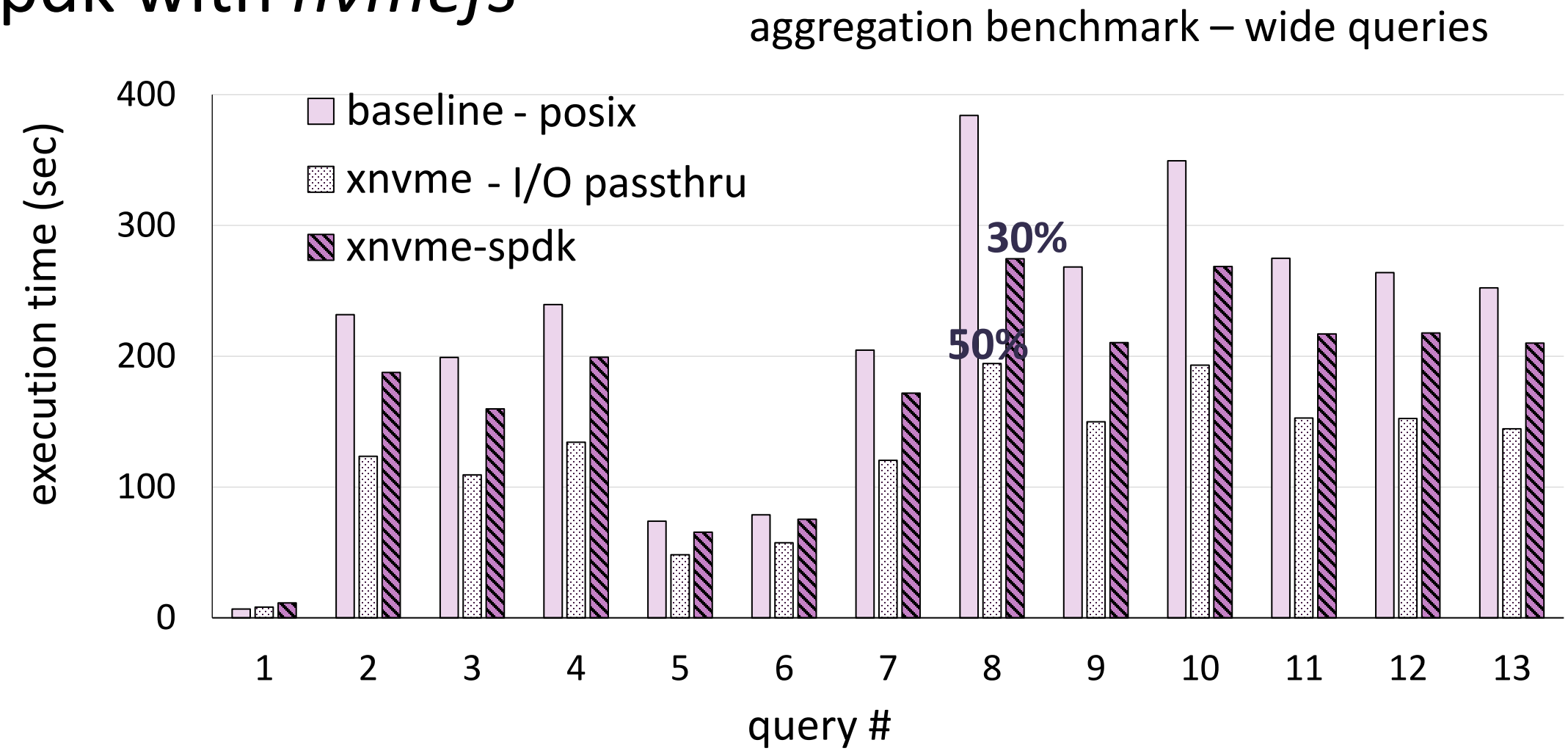
CPU (2X) – Intel® Xeon® Gold 6342	
RAM	500 GB
cores (threads)	24 (48)
clock rate (turbo)	2.8 (3.5) GHz
L1 (D/I) per core	48 / 32 KiB
L2 per core	1.3 MiB
L3 shared	36 MiB
OS	CentOS Stream 9
kernel (linux)	6.13.0
FDP SSD	
capacity	3.76 TB
block size	4 KiB

nvmefs in action



I/O Passthru & FDP benefit I/O-intensive queries sequential access (write amplification ≅ 1)

spdk with *nvmefs*



scaling factor = 32 (5.3GB)

buffer pool = 2GB

#threads = 1

I/O Passthru & SPDK benefit I/O-intensive queries

SPDK support without hurdles, but could be optimized

flexible I/O for data systems

- diversity of the SSD software & hardware landscape is underutilized
- xNVMe gives a unified access to this landscape
- *nvmefs* integrates xNVMe into DuckDB

➔support for io-uring, I/O Passthru, SPDK, flexible data placement …

- steps ahead
  - further & detailed performance analysis
  - improved default SPDK setup
    - better xNVMe buffer management
  - more data systems integrated with xNVMe
  - write- & random-access heavy workloads

nvmefs

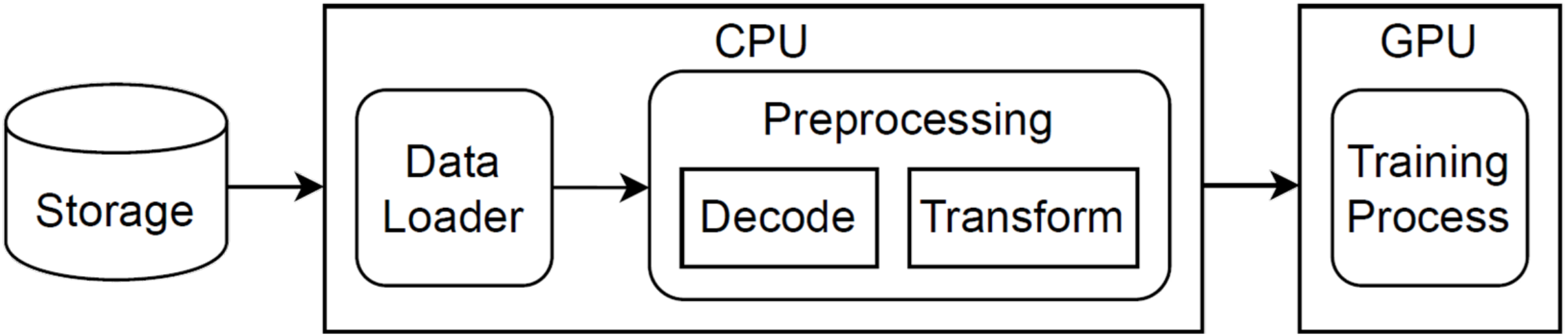
let's make modern SSD landscape more accessible!

navigating I/O options for data systems

- Flexible I/O for Database Management Systems with xNVMe
  E. Houlborg, A. N. Tietgen, S. A. F. Lund, M. Weisgut, T. Rabl, J. Gonzalez, V. Shah, P. Tözün
  CIDR 2026

- Path to GPU-Initiated I/O for Data-Intensive Systems
  K. B. Torp, S. A. F. Lund, P. Tözün
  DaMoN 2025

journey of data in deep learning training



CPU feeds the GPU

- 16-64 CPU cores per GPU (recommended)
- 96 CPU cores per TPU*

➔ otherwise, GPU/TPU may be underutilized

➔ can we do more with fewer CPUs & less of the CPU?

*Audibert et al., "tf.data service: A Case for Disaggregating ML Input Data Processing." ACM SoCC 2023

reducing the CPU needs for deep learning

- data & work sharing

  e.g., CoorDL [PVLDB'21], Joader [NeurIPS'22], tf.data service [SoCC'23], TensorSocket [SIGMOD'26]

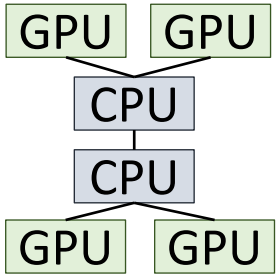
- data pre-processing on the accelerator

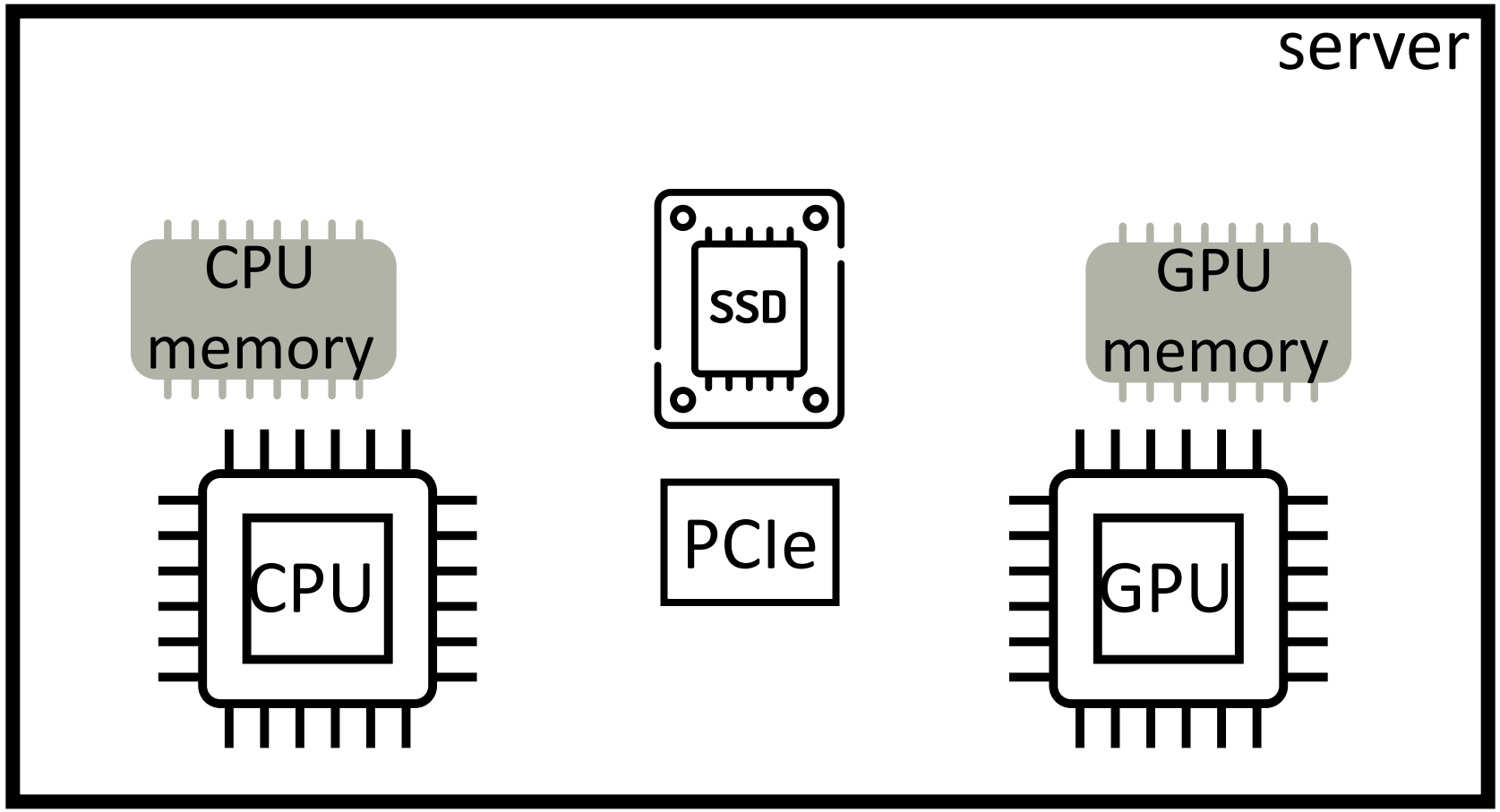
  e.g., DALI [NVIDIA], FusionFlow [PVLDB'24]

- GPU-centric I/O path
  - GPUDirect Storage (GDS)
  - Big Accelerator Memory (BaM)

what are the trade-offs of different options?

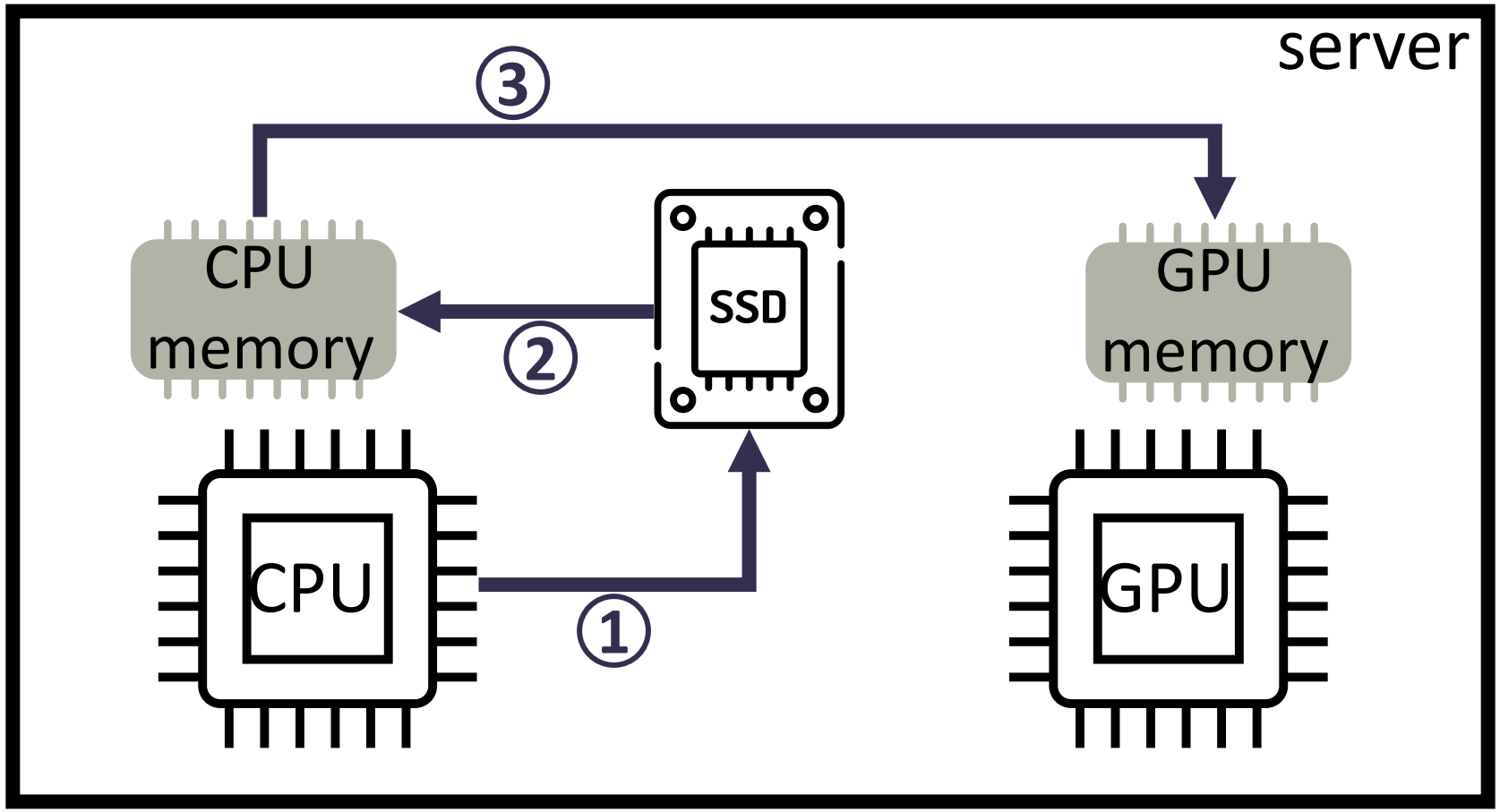
target hardware setup





* PCIe is dropped in the remaining figures for the sake of simplicity in illustrations.

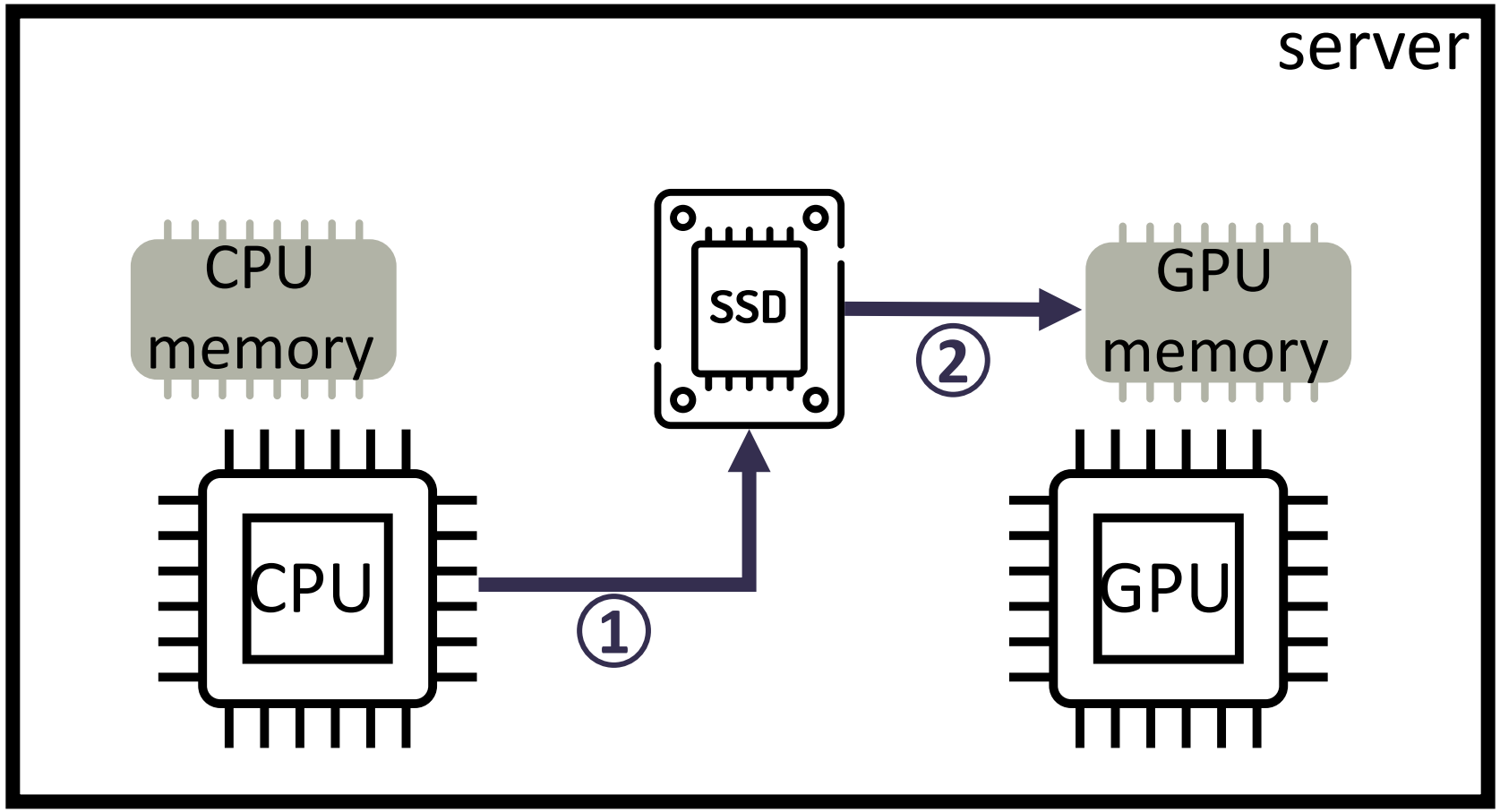
conventional: CPU-centric



- ✓ **ecosystem support**
- × **CPU-bound & overhead from memory copy**

GDS: GPU-centric & CPU-initiated

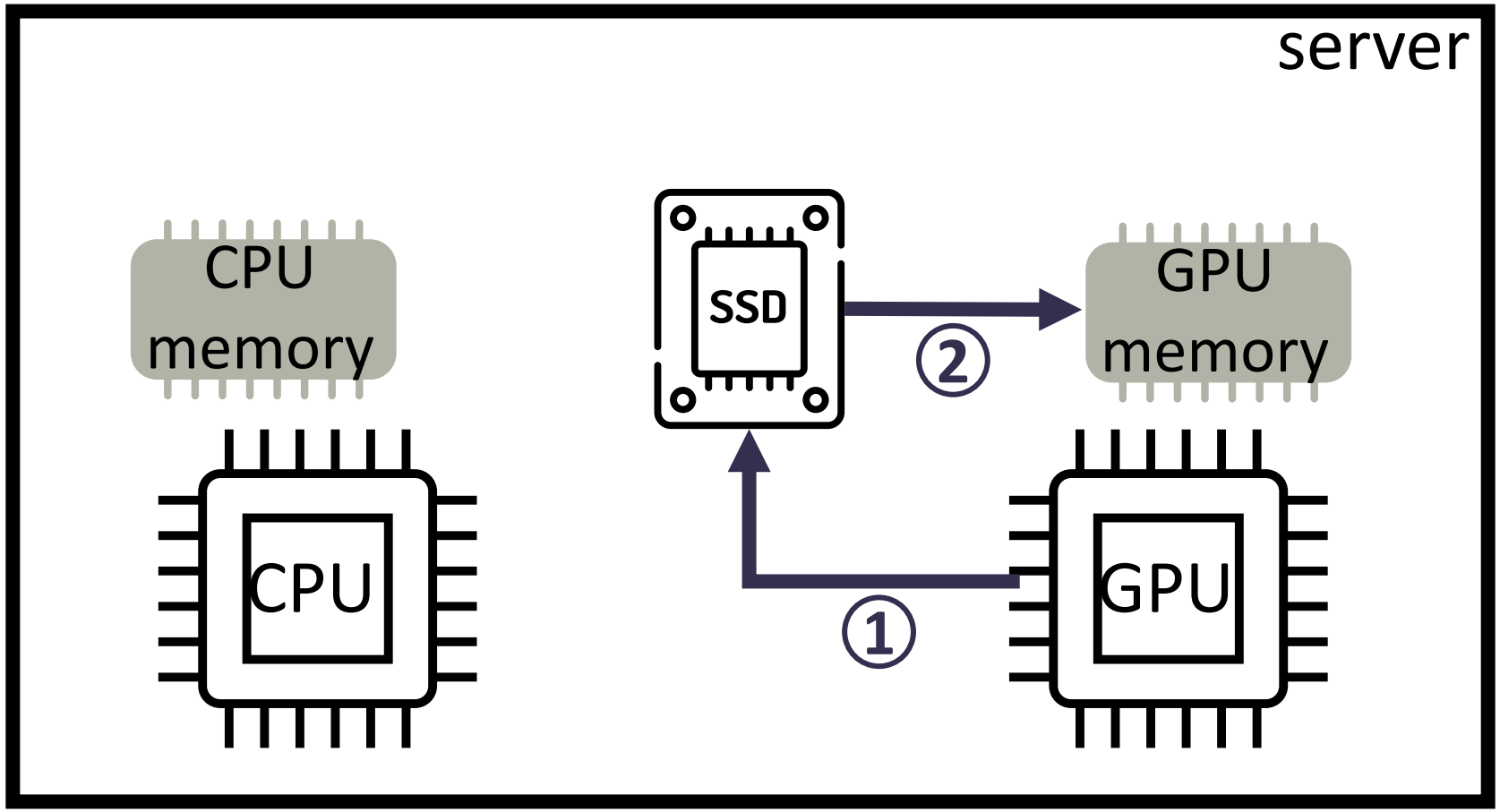
GPU Direct Storage [NVIDIA'19]



- ✓ **eliminates the extra memory copy**
- × **still CPU-bound**

BaM: GPU-centric & GPU-initiated

Big Accelerator Memory [ASPLOS'23]



- ✓ **eliminates the CPU on the path**
- × **ecosystem missing & saturates GPU**

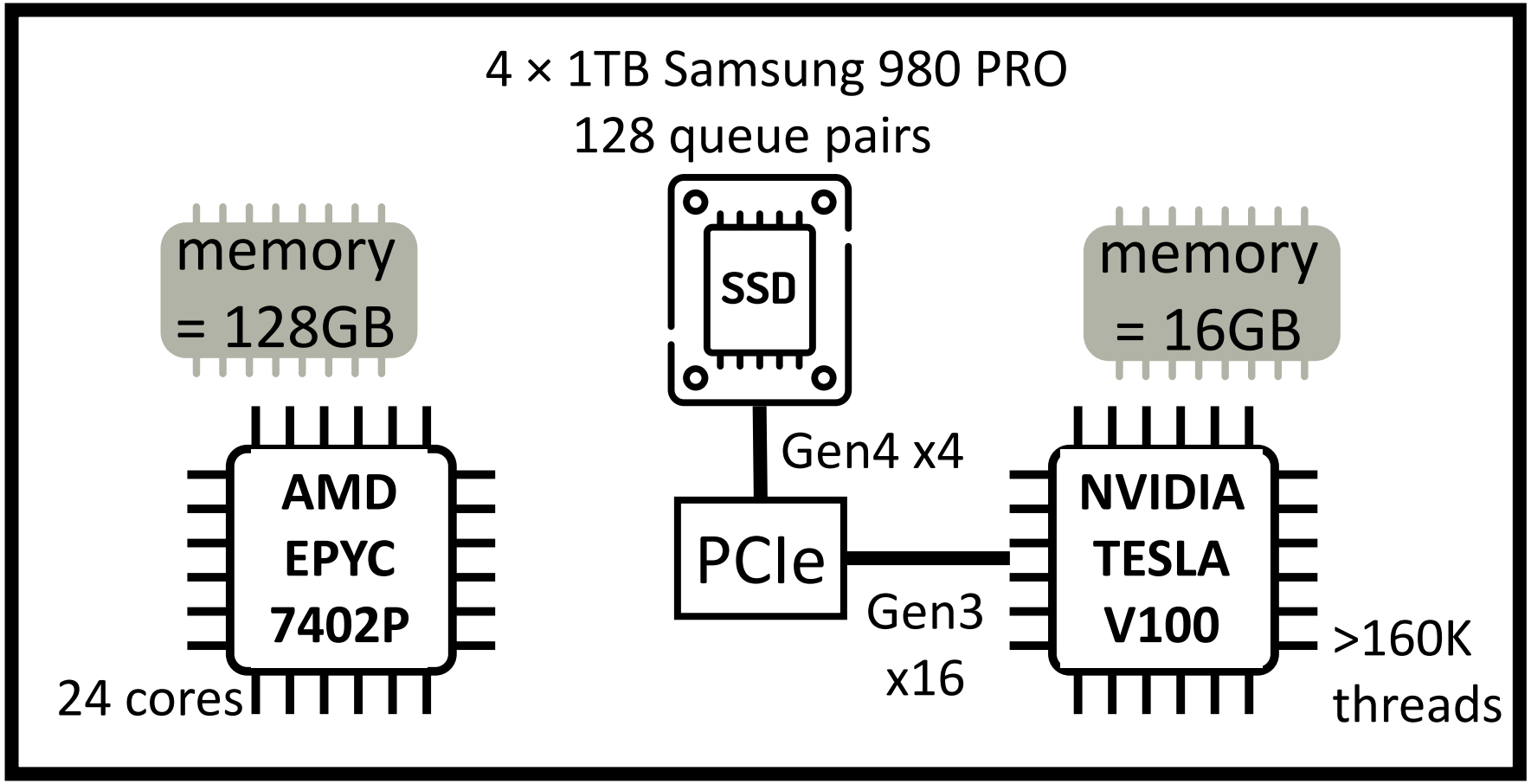
CPU- vs GPU-centric storage access

mechanisms: CPU-centric: SPDK & GPU-centric: GDS, BaM

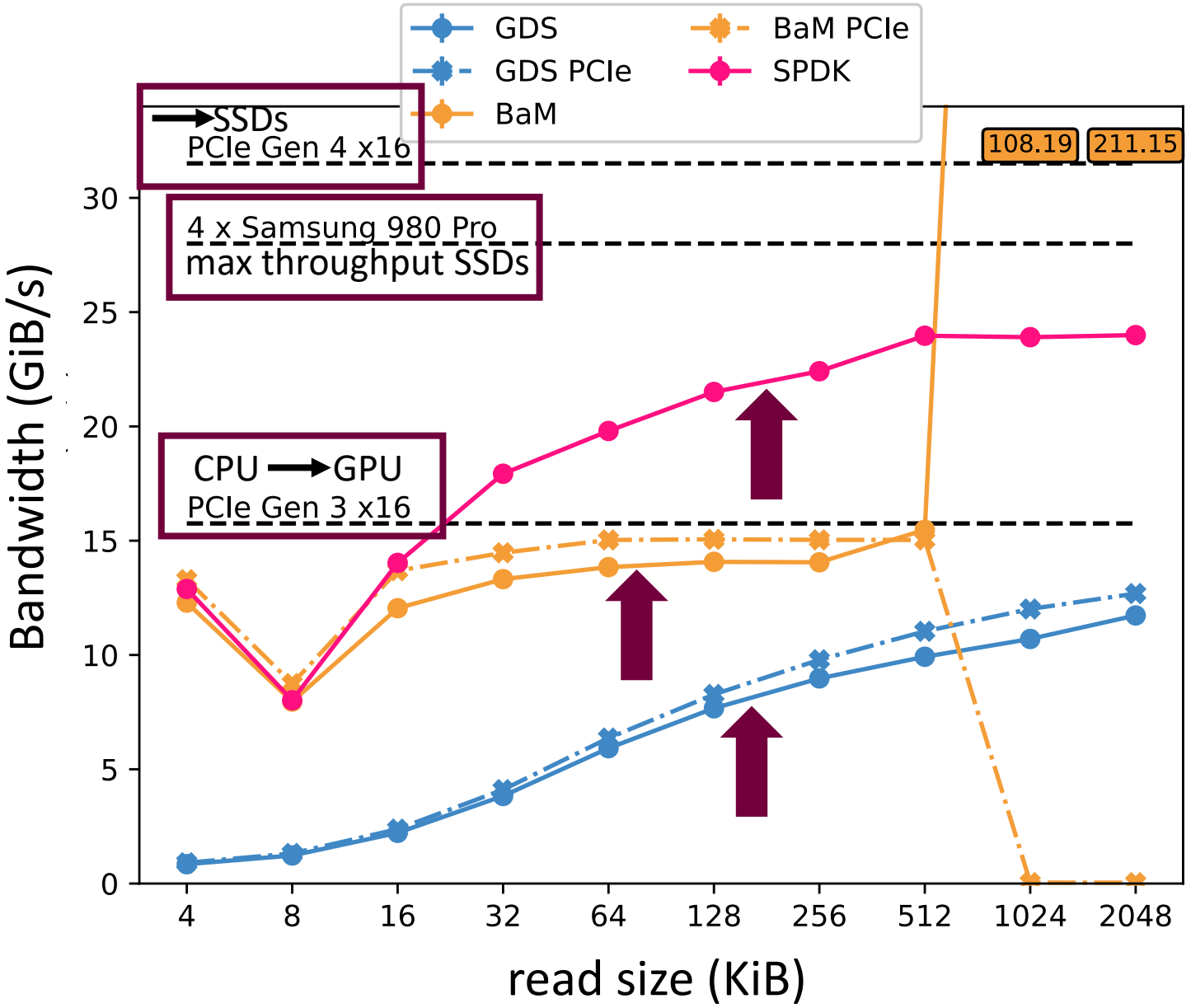
workload: random reads

→ each mechanism has their own tool for benchmarking

hardware



bandwidth utilization – 4 SSDs & PCIe



GDS is CPU-compute heavy.

➔ 16 logical cores utilized

BaM is limited by the PCIe Gen3 link & heavy on the GPU resources.

➔ whole GPU utilized

CPU-centric SPDK is resource-efficient but has a longer path to the GPU.

➔ 2 logical cores utilized

path to GPU-centric storage access

- need to reduce the dependency on CPUs
  for more efficient deep learning pipelines
- GPU-centric data path is a way to do that
  & we have the mechanisms today (e.g., GDS, BaM)
  - GDS has dependency on CPUs still
  - BaM requires a lot of GPU resources

➔ when to use each mechanism while being resource-aware?

➔ how to best integrate them into popular deep learning frameworks or GPU databases for wider-scale use?

navigating I/O options for data systems

- Flexible I/O for Database Management Systems with xNVMe
  E. Houlborg, A. N. Tietgen, S. A. F. Lund, M. Weisgut, T. Rabl, J. Gonzalez, V. Shah, P. Tözün
  CIDR 2026

- Path to GPU-Initiated I/O for Data-Intensive Systems
  K. B. Torp, S. A. F. Lund, P. Tözün
  DaMoN 2025

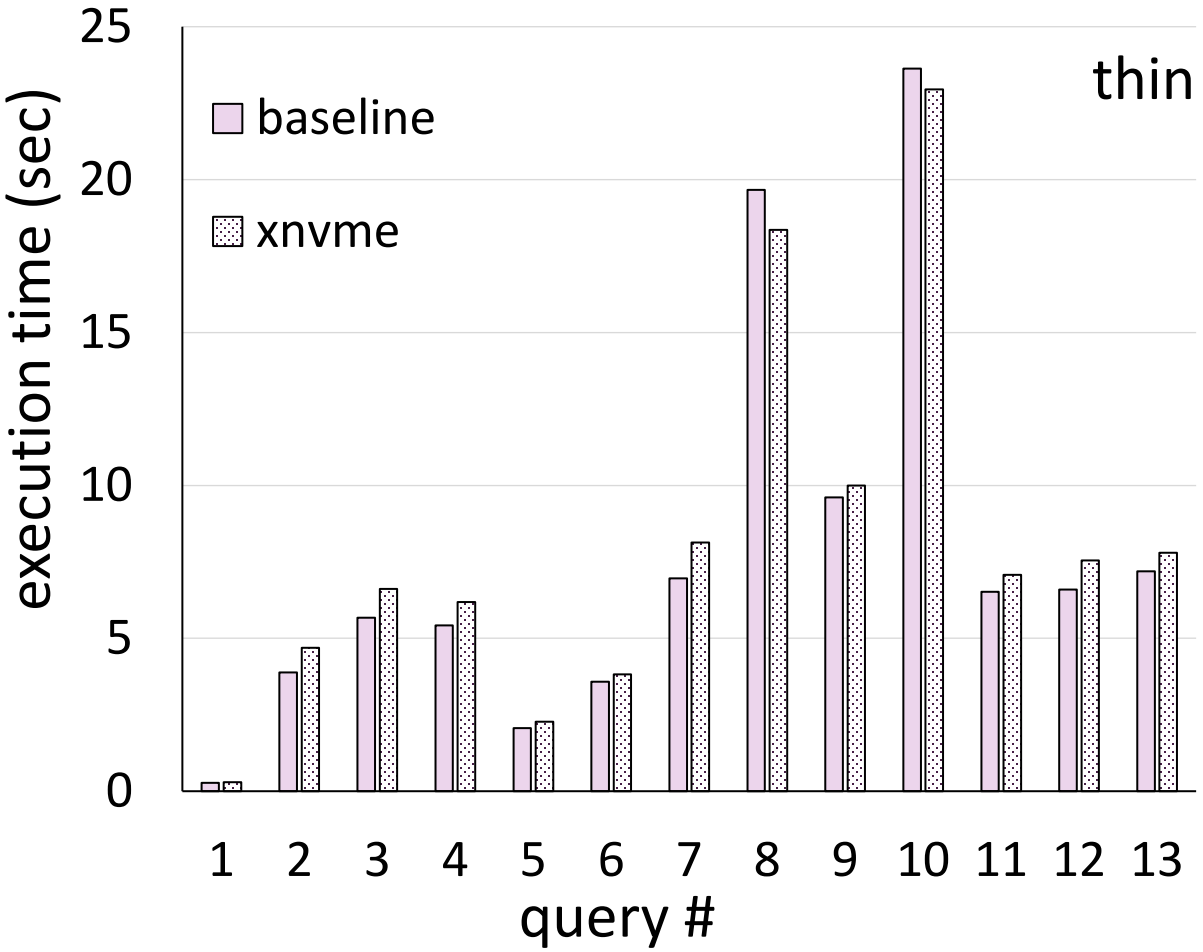
thank you!

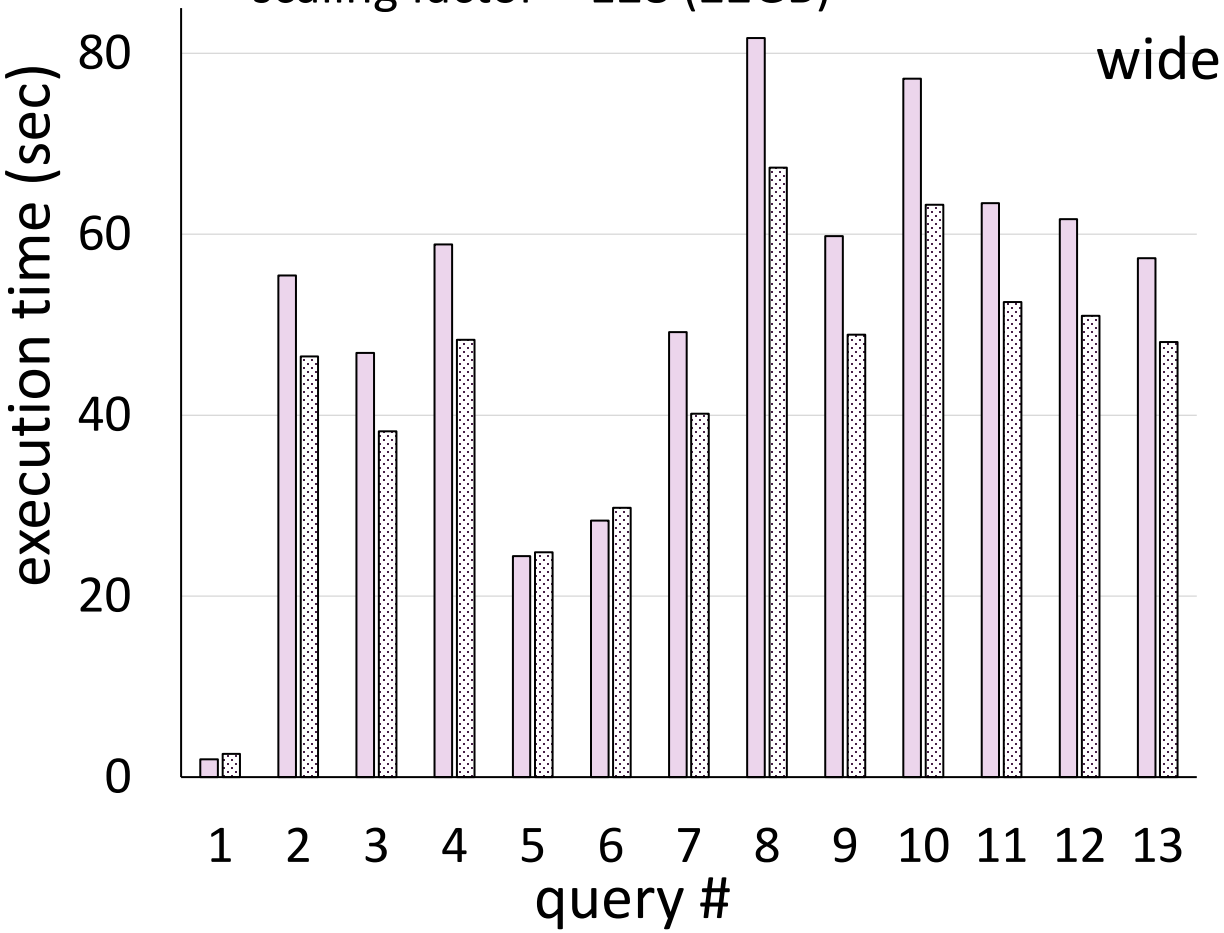
backup

nvmefs in action

aggregation benchmark

#threads = 16
buffer pool = 20GB
scaling factor = 128 (22GB)



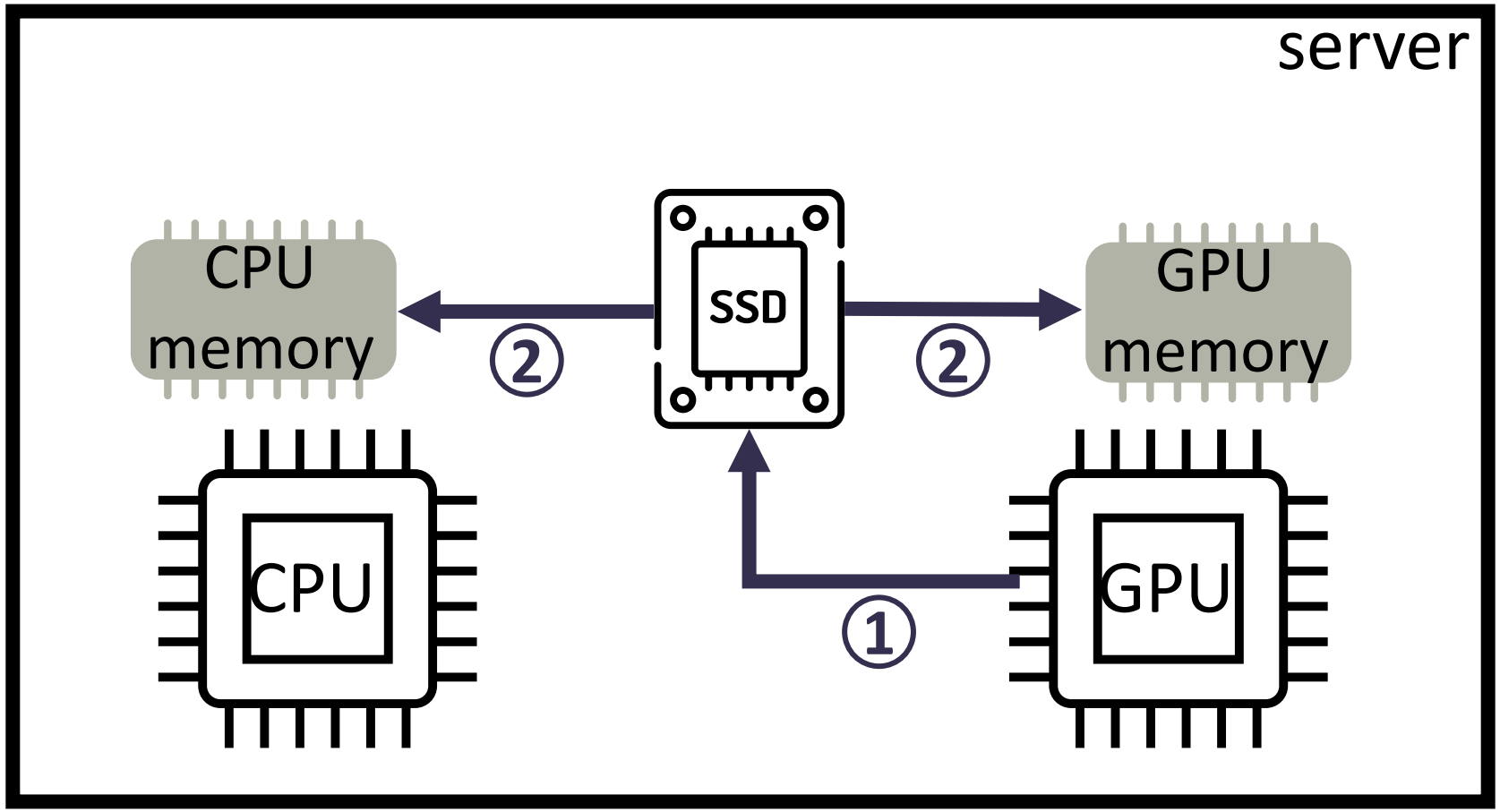


nvmefs helps wide variant (up to 20%)

less I/O-intensive thin variant doesn't benefit

GMT: GPU-centric & CPU- & GPU-initiated

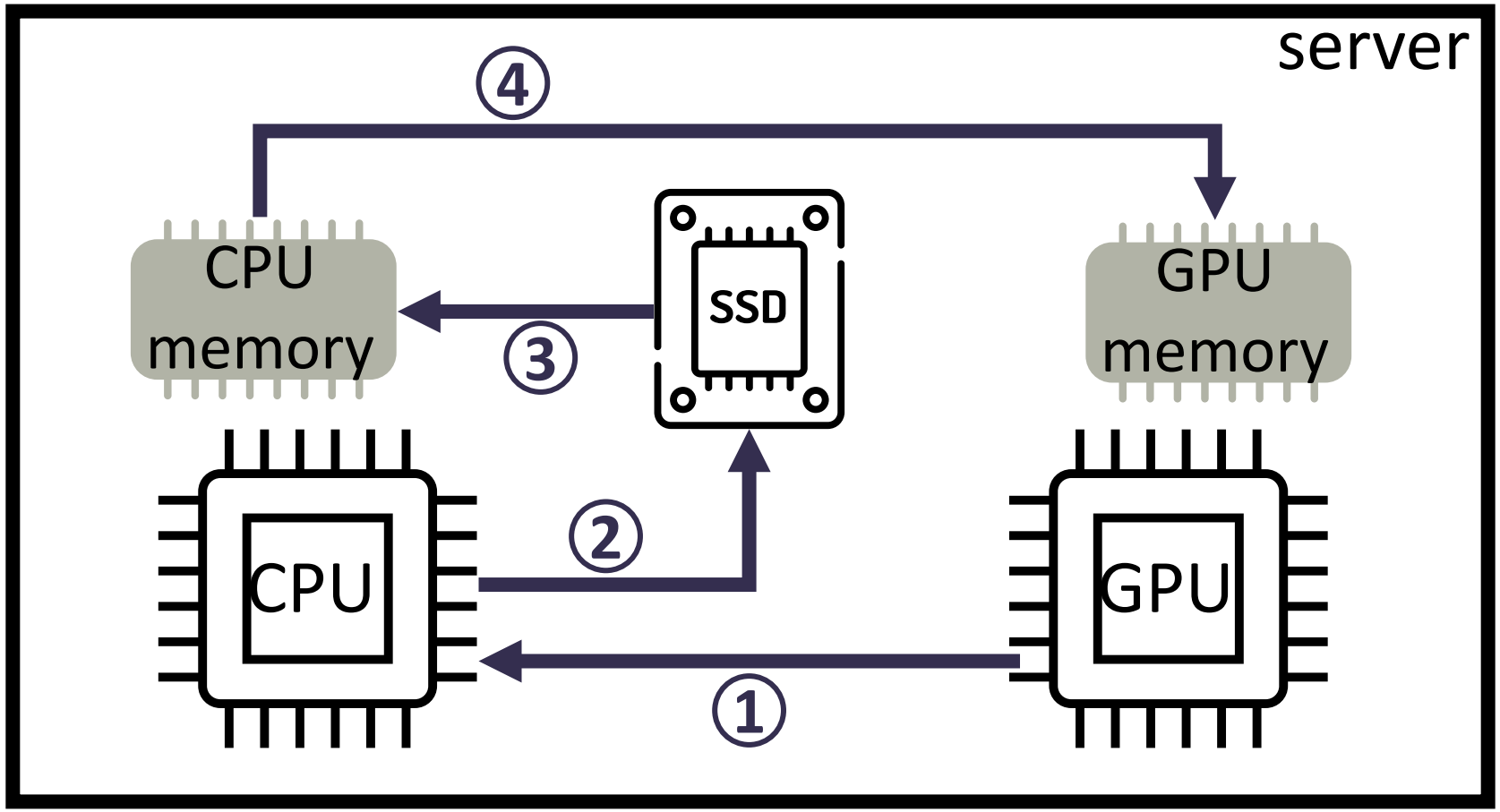
GPU Orchestrated Memory Tiering [ASPLOS'24]



- ✓ **hybrid approach, works well if data reuse is high**
- × **extra CPU/GPU resources used**

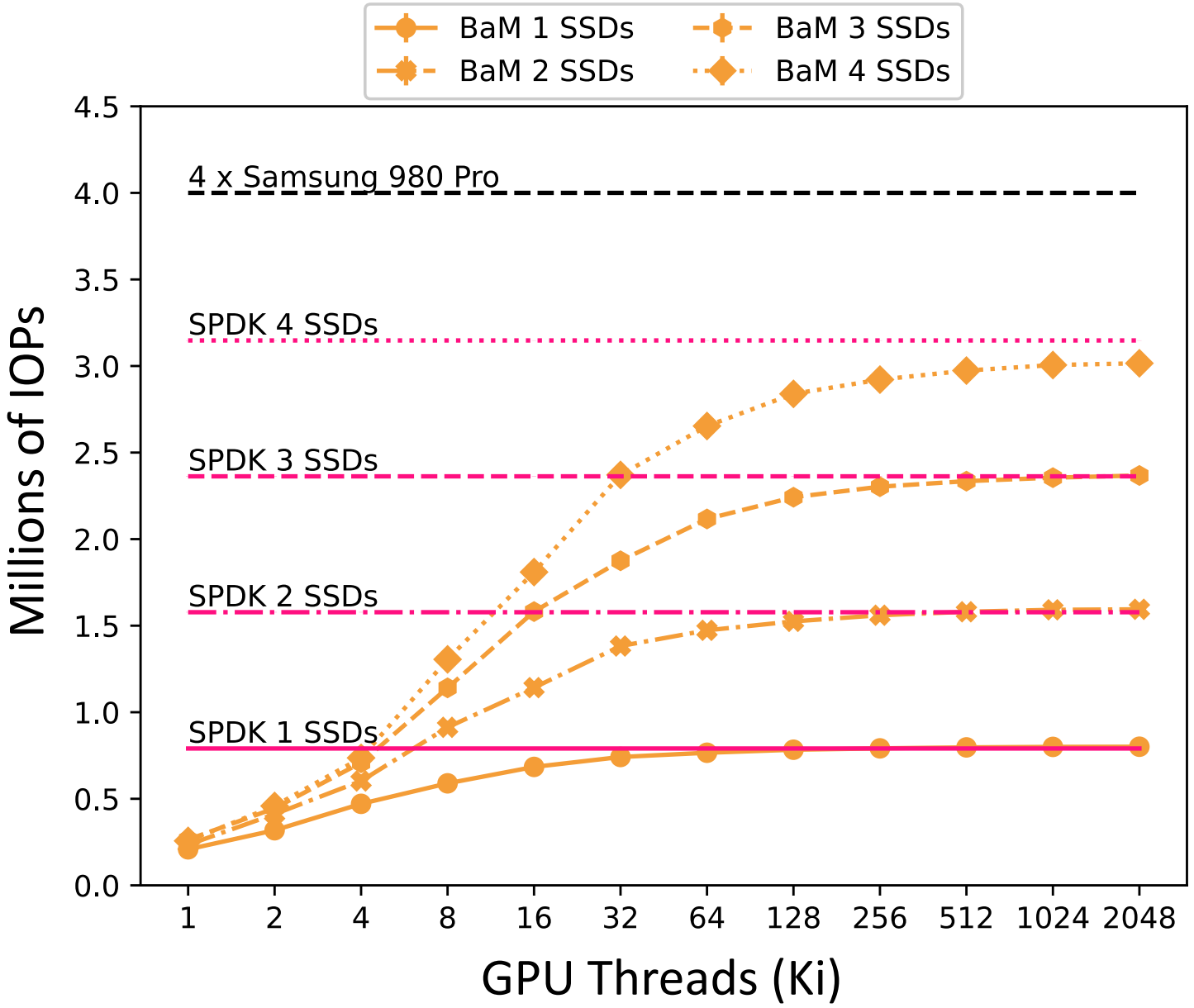
GPUfs & ActivePointers: GPU-centric & GPU-initiated I/O

[ASPLOS'13] [ISCA'16]



- ✓ **improves programmability**
- × **longer I/O path, slight performance impact**

scalability

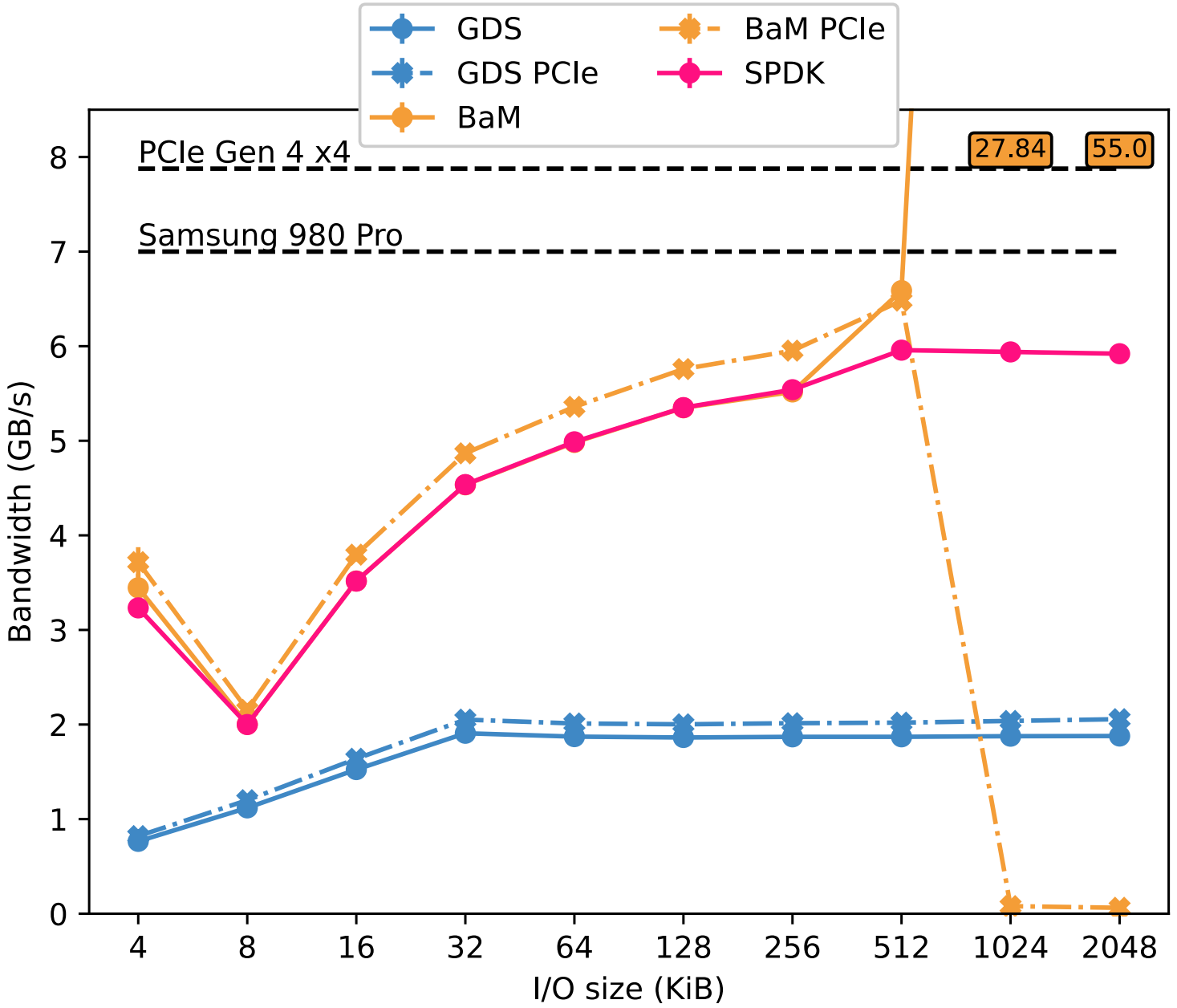


➔ **BaM requires a page allocation in GPU memory for each GPU thread.**

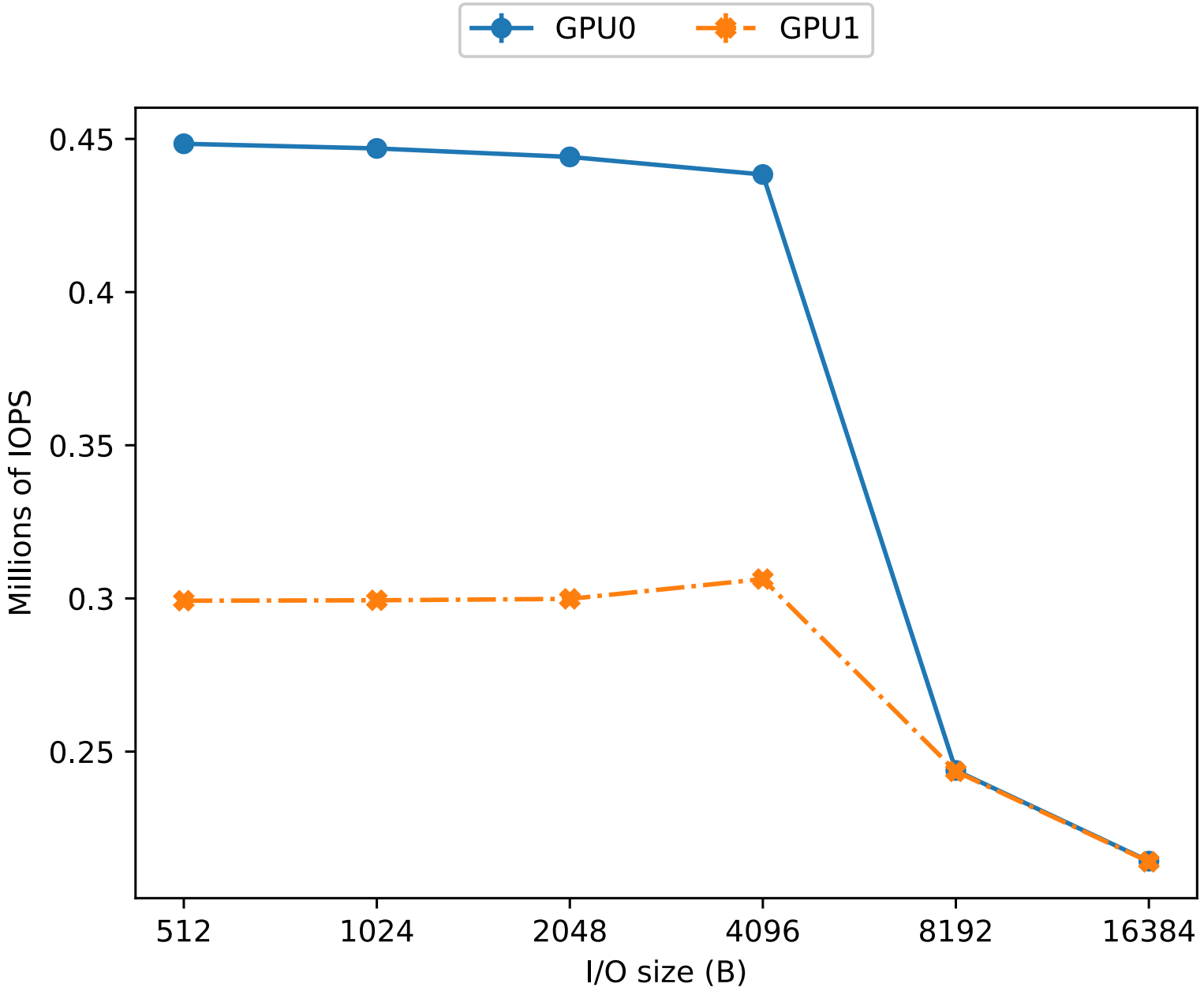
➔ **More threads doing IO, more GPU memory needed.**

➔ **The sub-scalar performance for 4 SSDs is due to being limited by GPU memory.**

bandwidth utilization – 1 SSD



locality


GPU0
GPU1
0.45
0.4
0.35
0.3
0.25
Millions of IOPS
512
1024
2048
4096
8192
16384
I/O size (B)

evaluation: CPU- vs GPU-centric I/O

System	Gigabyte G292-Z20
CPU	AMD EPYC 7402P 24-Core Processor
DRAM	8 X 32GB SK Hynix DDR4 2400MHz
GPUs	2 X NVIDIA Tesla V100-16GB PCIe Gen 3
SSDs	4 X 1TB Samsung 980 PRO w/ Heatsink
OS	Ubuntu 20.04 LTS (Linux 5.8)
NVIDIA	Driver 550, CUDA 12.6
BaM	GitHub ‘master’ branch
GDS	Matching CUDA (12.6)
SPDK	v24.09