Flexible I/O for Database Management Systems with xNVMe

Emil Houlborg
Andreas N. Tietgen
Pınar Tözün

Simon A. F. Lund
Javier González
Vivek Shah

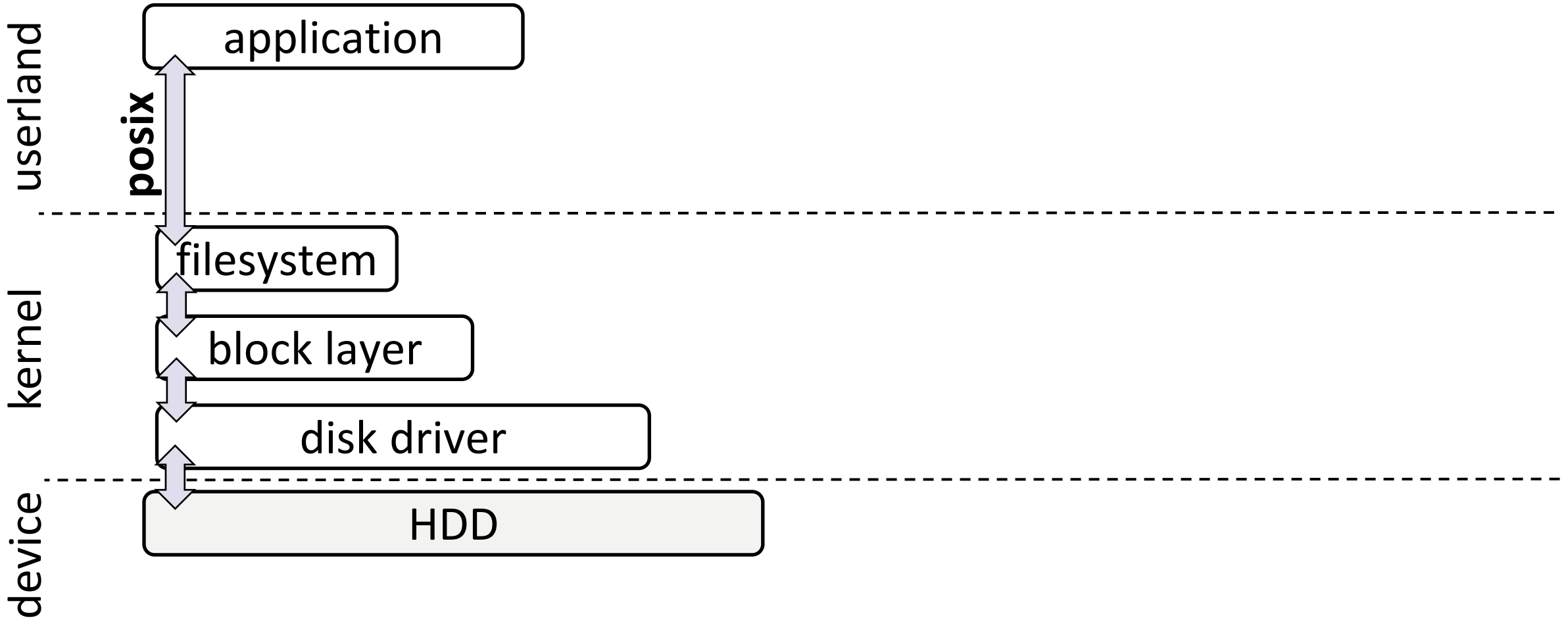
Marcel Weisgut
Tilmann Rabl

RAD
IT UNIVERSITY OF COPENHAGEN

SAMSUNG

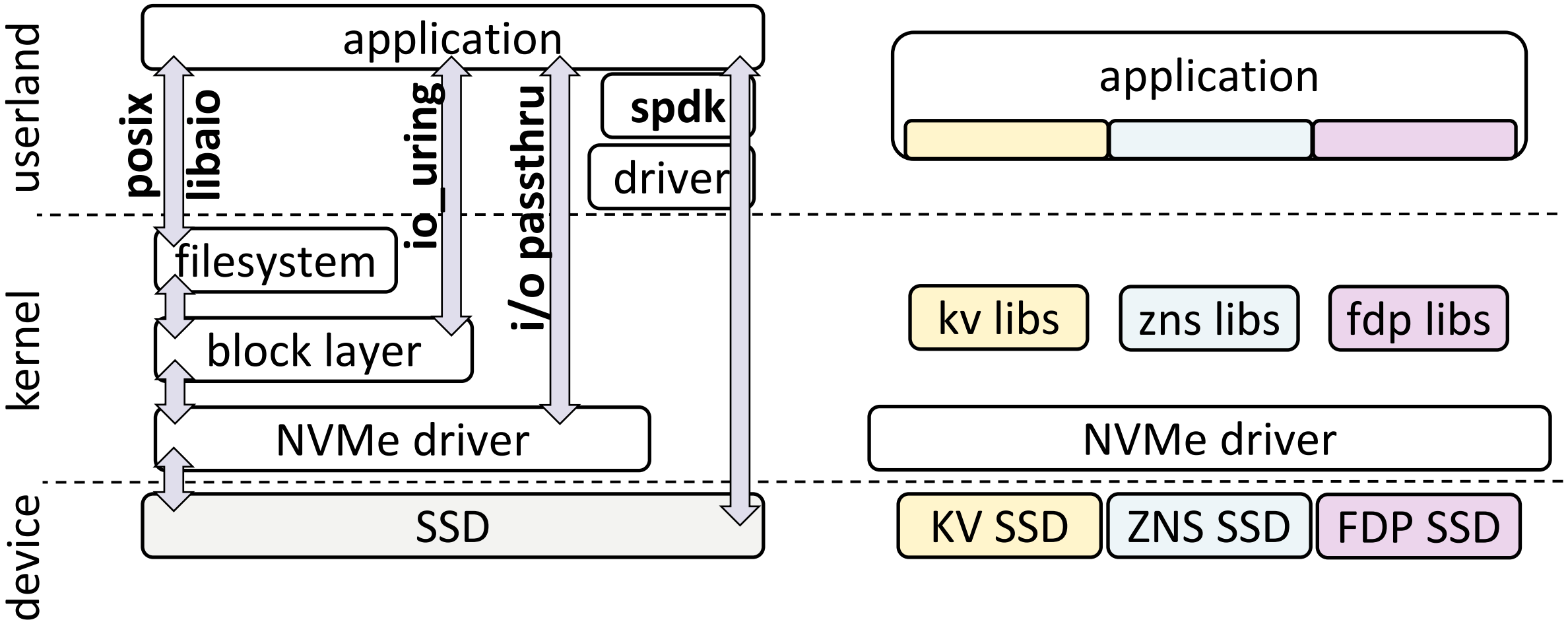
DES
HPI

once upon a time …



traditional I/O path was designed for hard disks

today – (non-exhaustive) SSD landscape



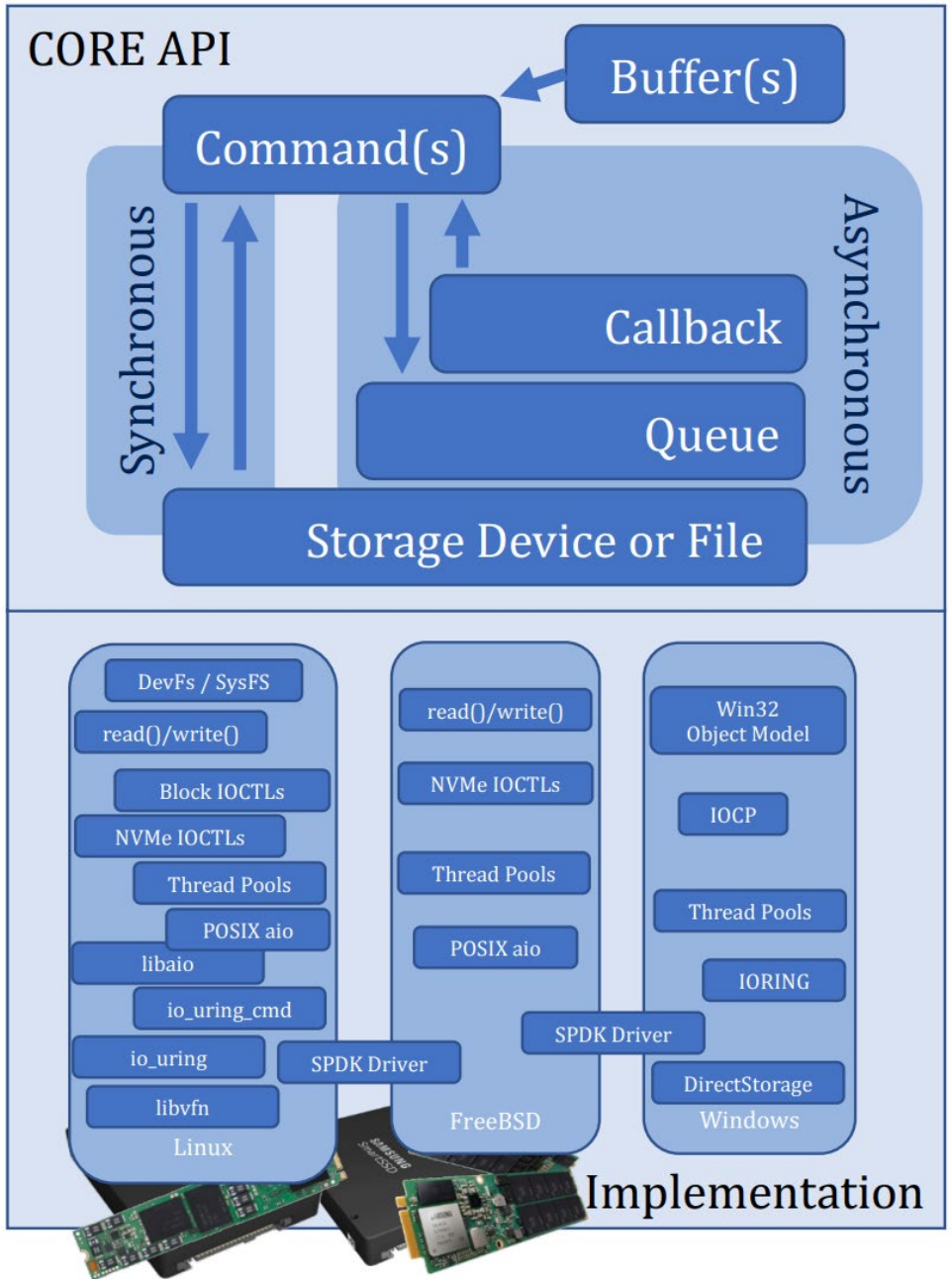
data systems tend to avoid dealing with the landscape

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

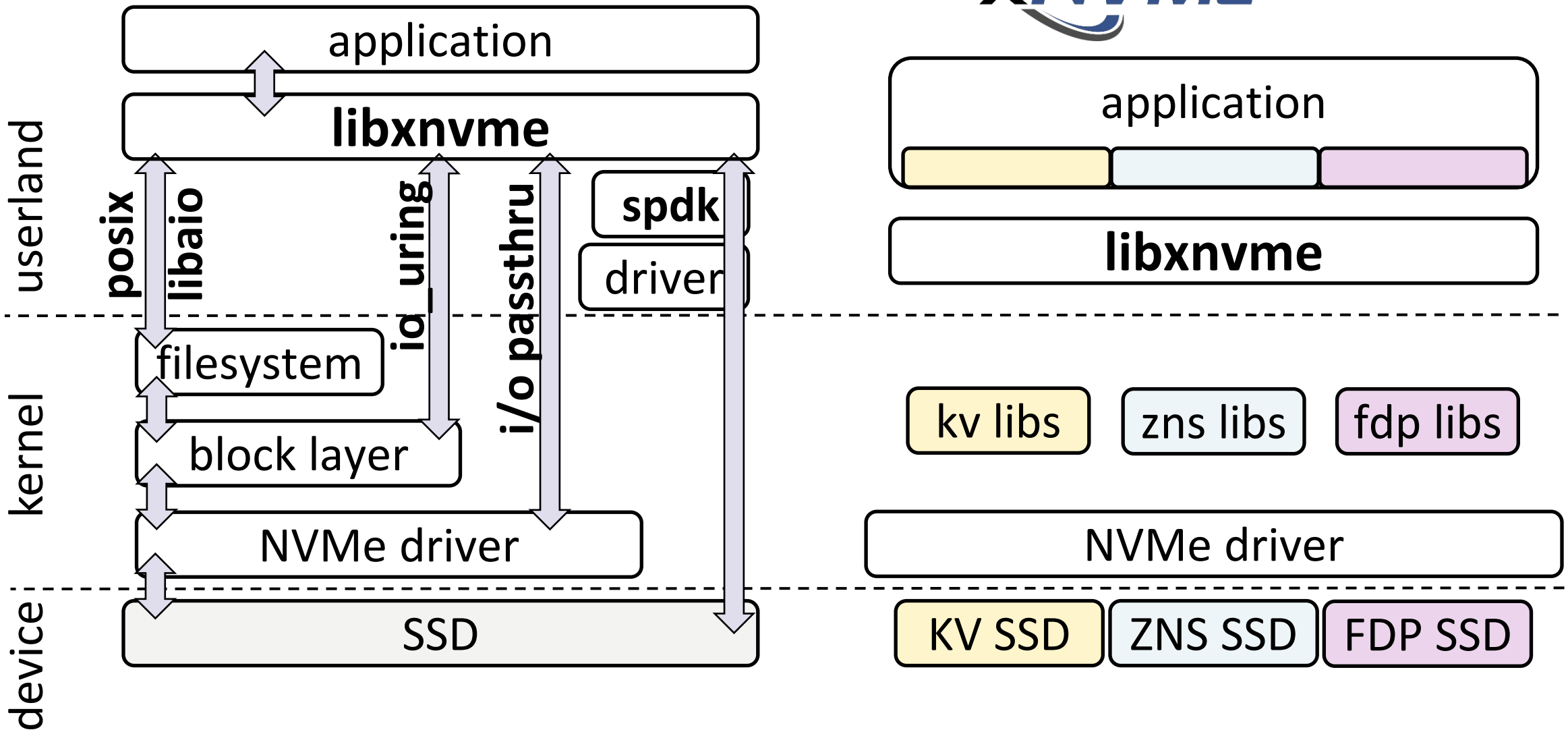
Javier Gonzalez
Samsung
Copenhagen, Denmark
javier.gonz@samsung.com

SYSTOR'22

a way to approach the landscape

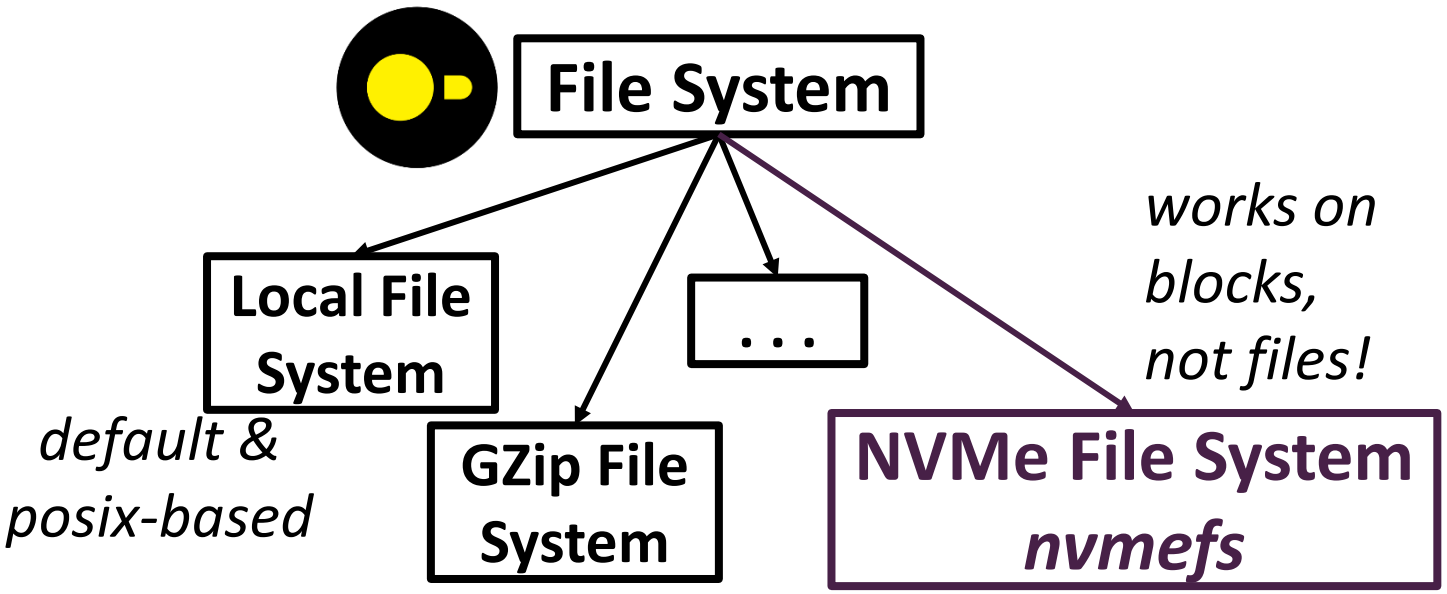
xNVMe

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);
```

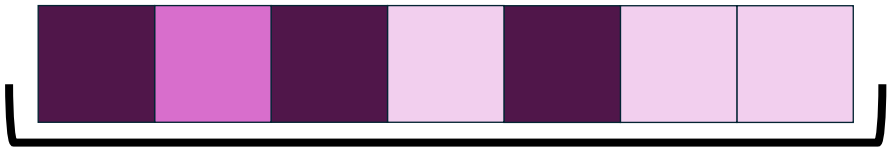
× can get better performance if integrated into DuckDB core

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*

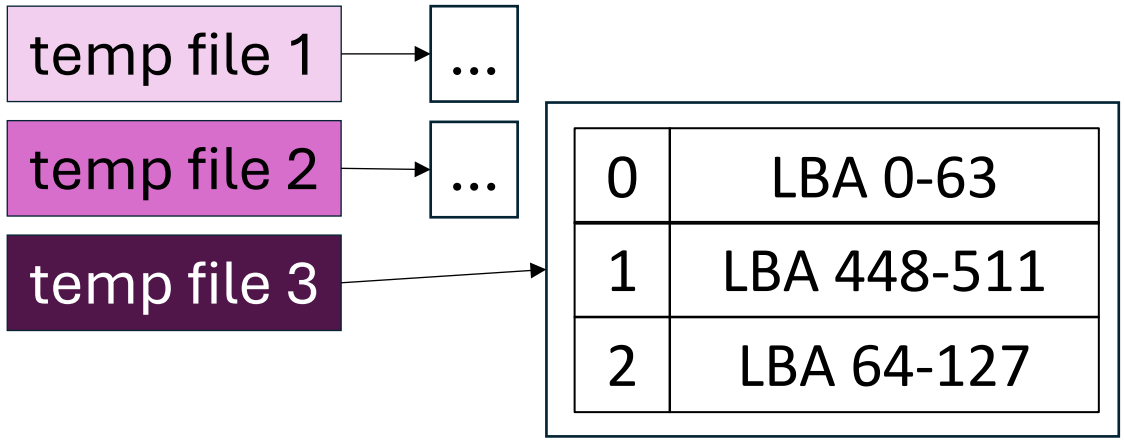


temporary file blocks

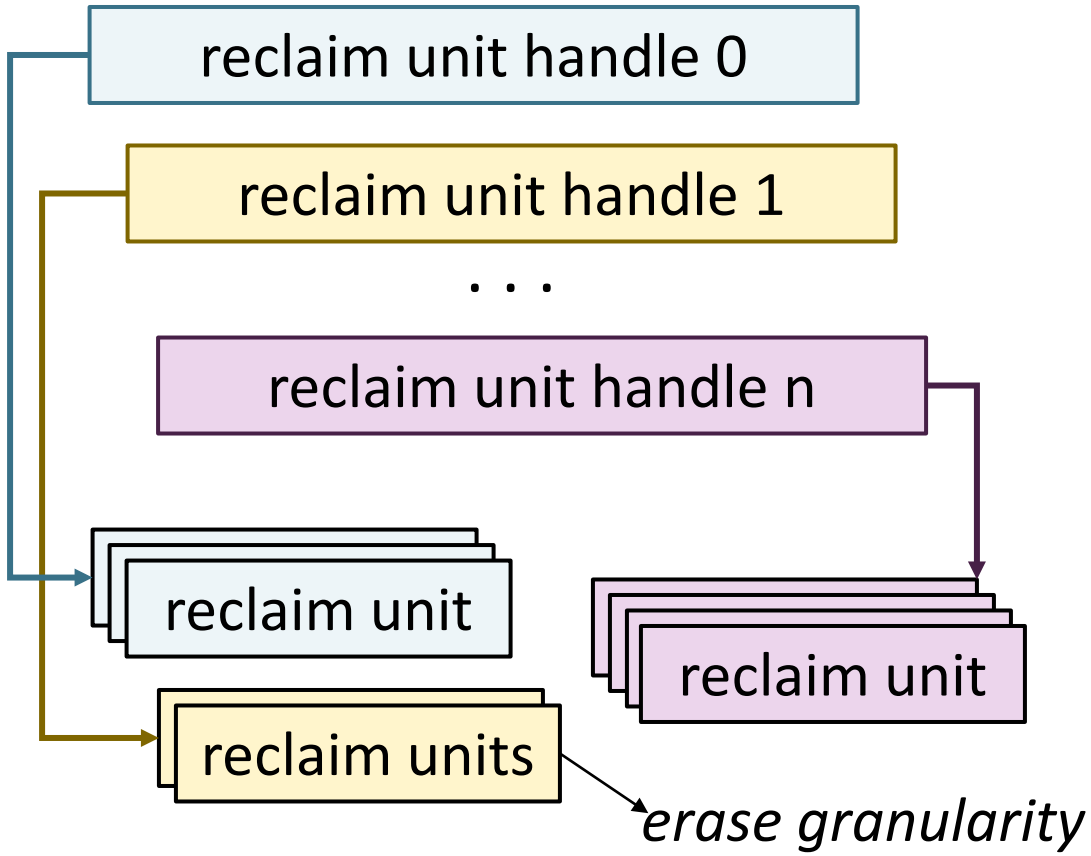
persisted @LBA 0

```
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

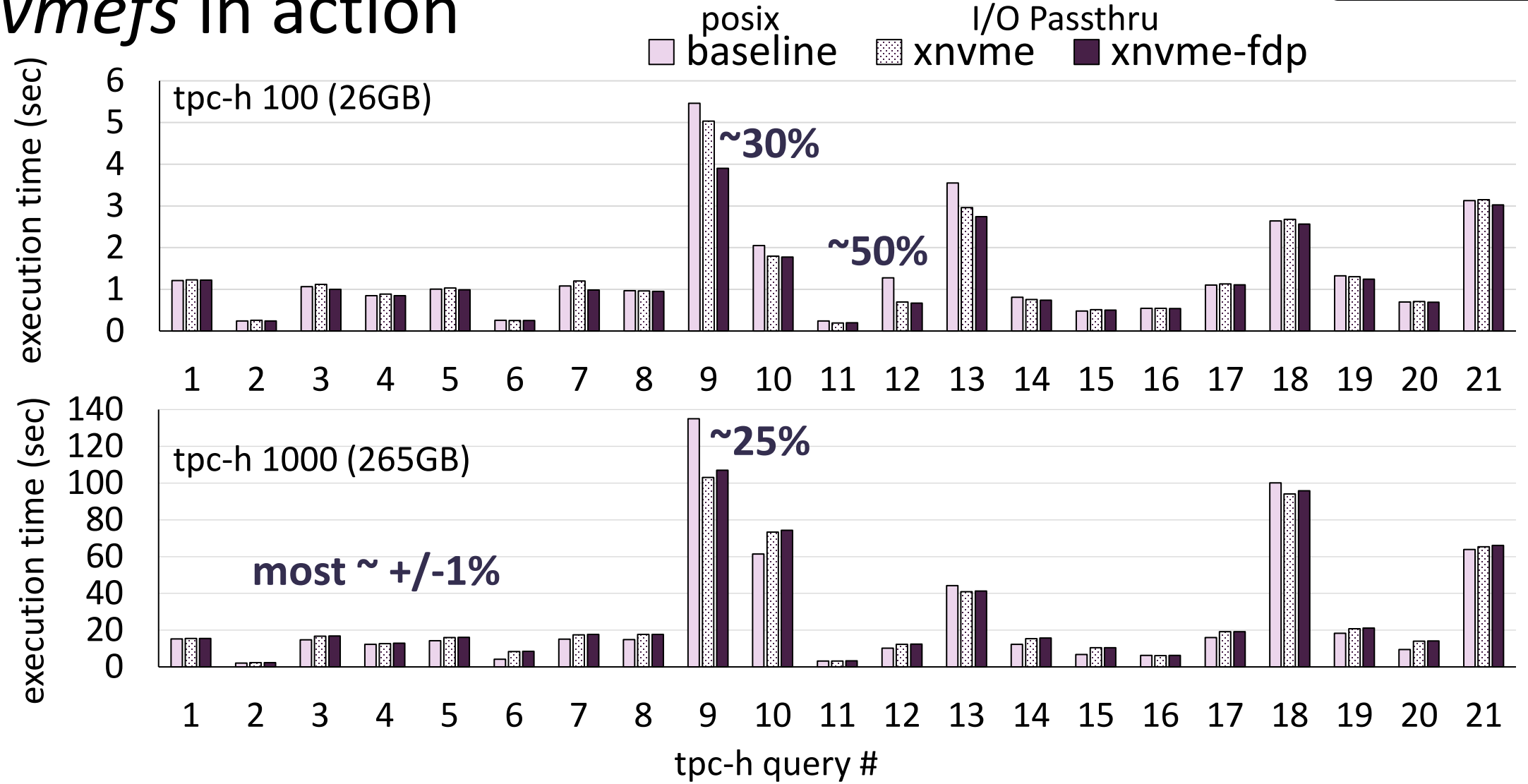
Peter Boncz
CWI, Amsterdam, Netherlands
peter.boncz@cwi.nl

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

@ Samsung Memory Research Center

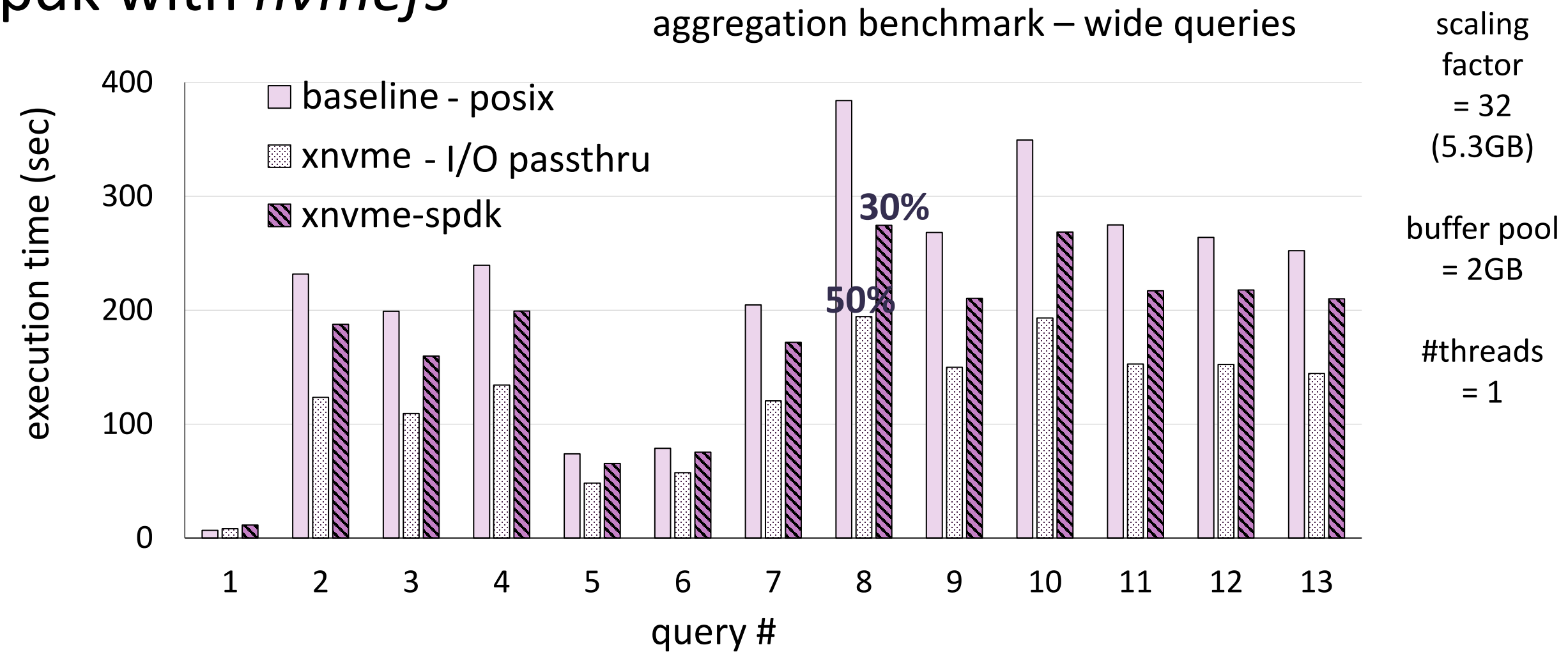
CPU (2X) – Intel® Xeon® Gold 6342	
RAM	500 GB
cores (threads)	24 (48)
clock rate (turbo)	2.8 (3.5) GHz
L1 (D/I) per core	48 / 32 KiB
L2 per core	1.3 MiB
L3 shared	36 MiB
OS	CentOS Stream 9
kernel (linux)	6.13.0
FDP SSD	
capacity	3.76 TB
block size	4 KiB

nvmefs in action



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

spdk with *nvmefs*



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

SPDK support without hurdles, but could be optimized

conclusion

thank you!

- 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

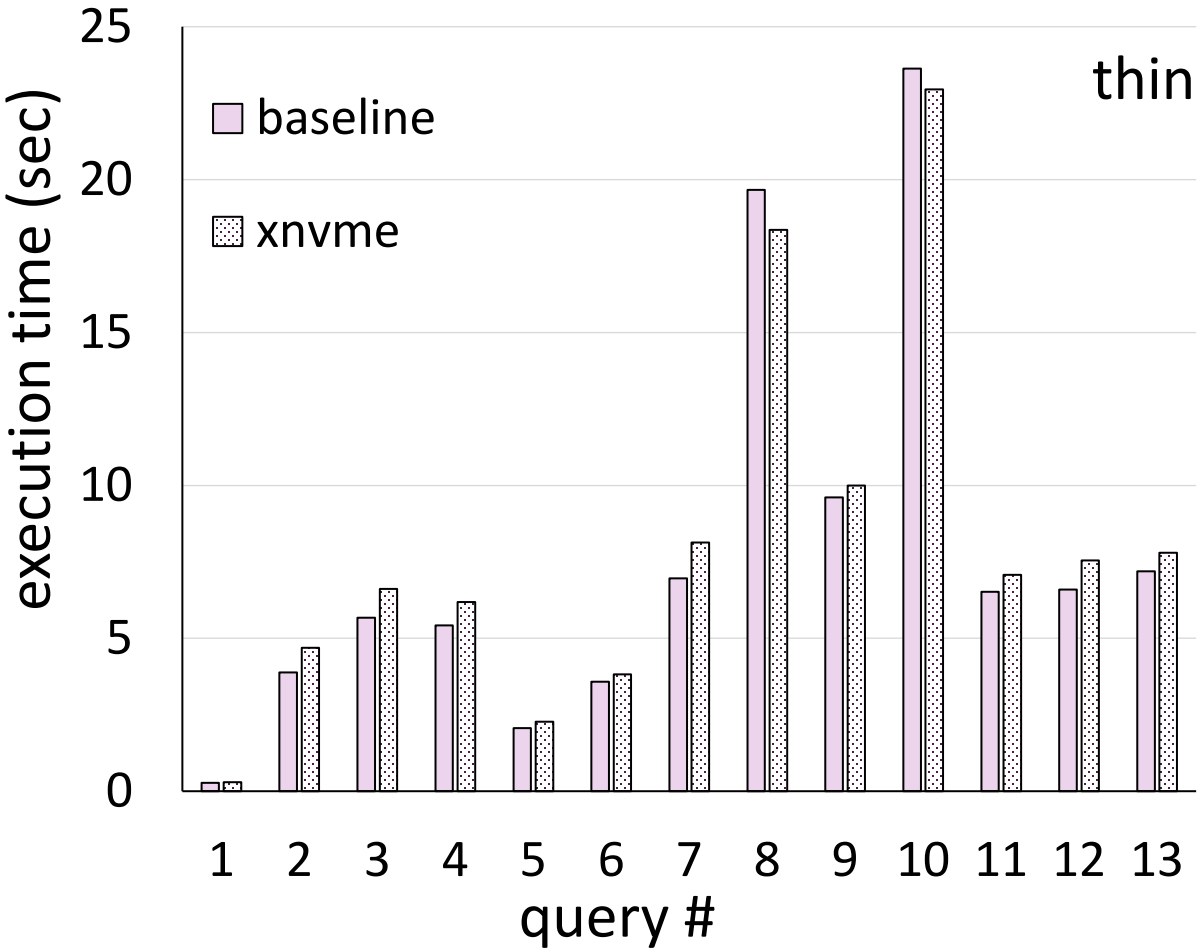
let’s make modern SSD landscape more accessible!

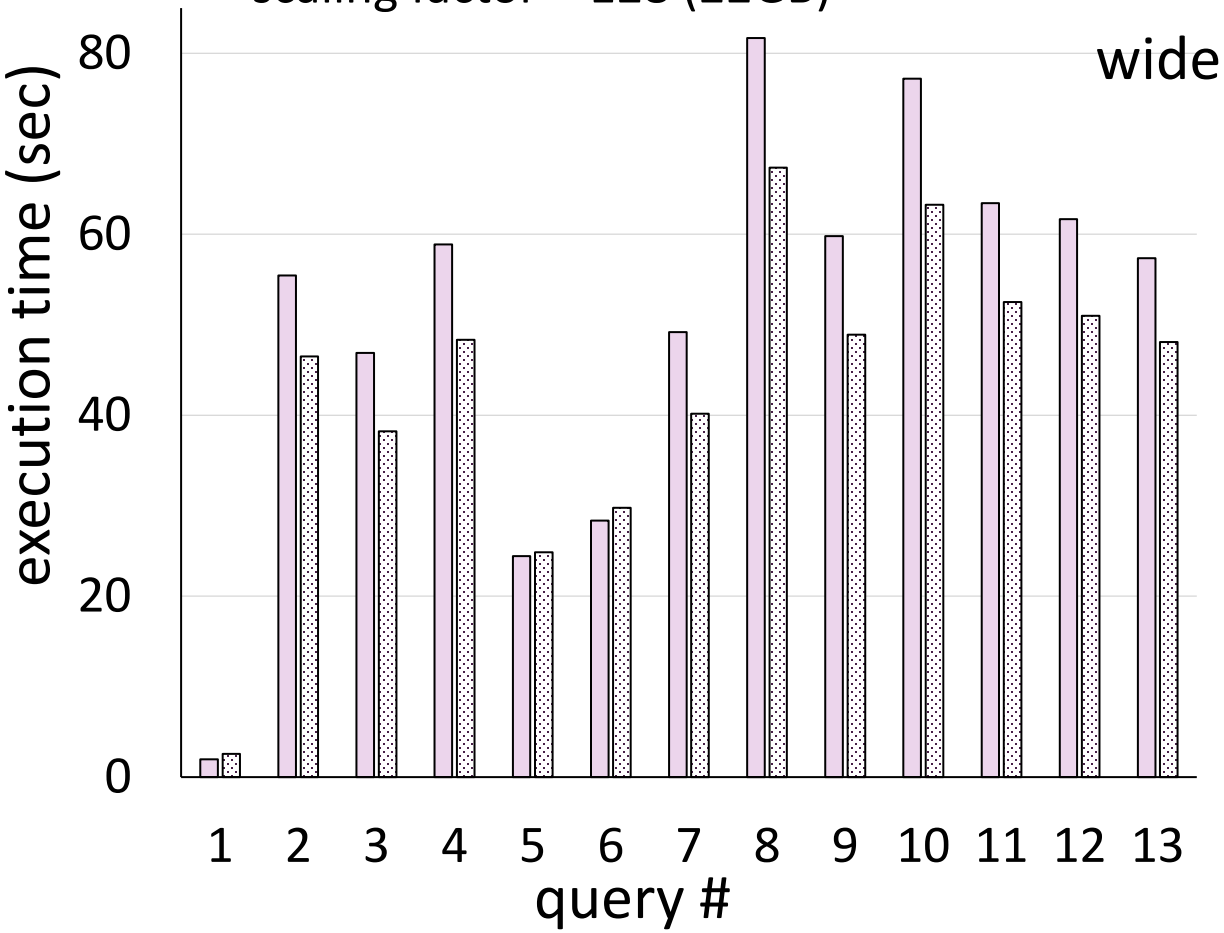
backup

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