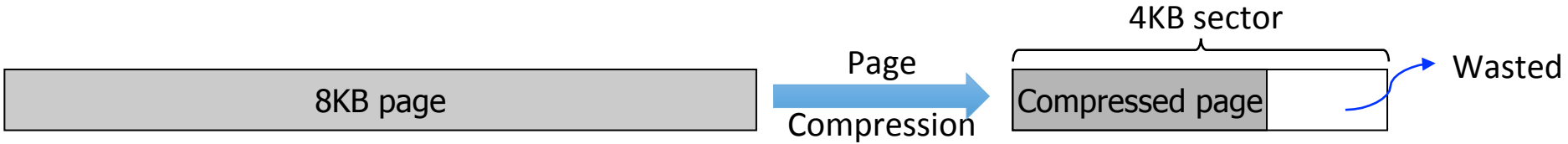
Bring Compression to Postgres at Zero Cost of Performance

Page Compression: A Missing Feature in Postgres

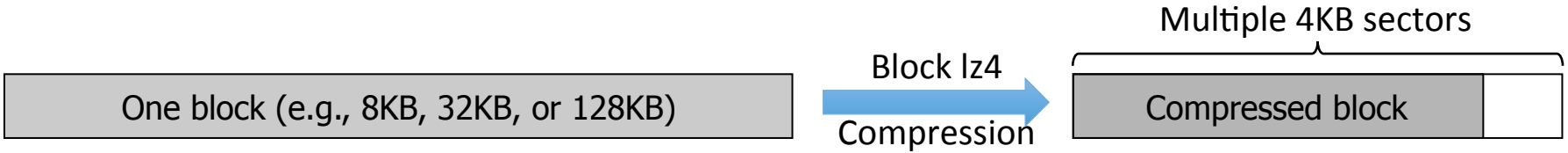
- Data compression: Boring but very useful (who does not like saving cost?)
- Why is page compression missing in Postgres?
  - Of course, CPU cost and hence performance penalty
  - Hole punching support of filesystems



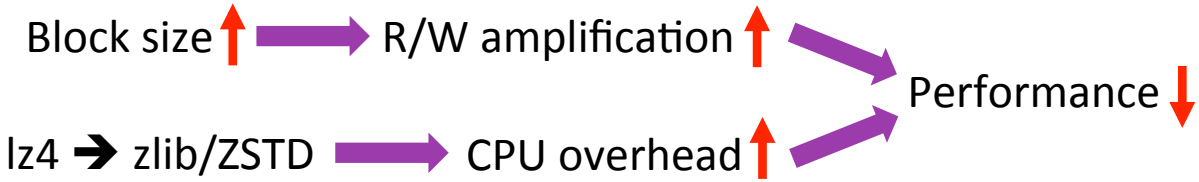
Storage cost saving < 50% → Hardly justify the efforts/penalty

Transparent Compression

Filesystem w/ transparent compression (ZFS, Btrfs, …)



Improve compression ratio & reduce storage space waste



Transparent Compression

Journaling filesystem (ext4, XFS, …)

Block layer w/ transparent compression (RedHat VDO, …)

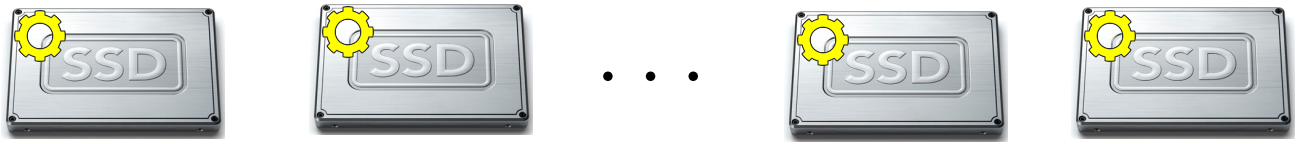
Improve compression ratio & reduce storage space waste

lz4 → zlib/ZSTD → CPU overhead ↑ → Performance ↓

Transparent Compression

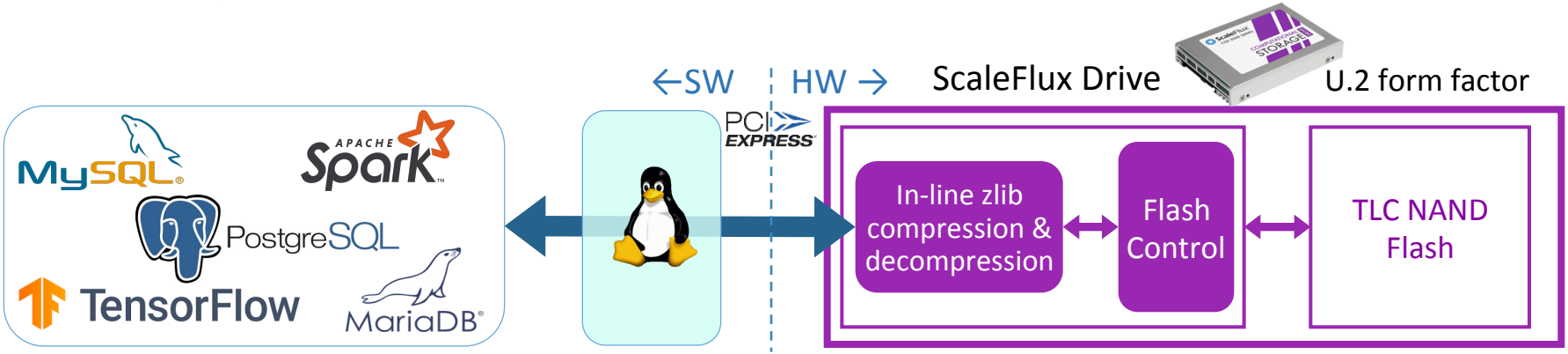
Journaling filesystem (ext4, XFS, …)

Normal block layer

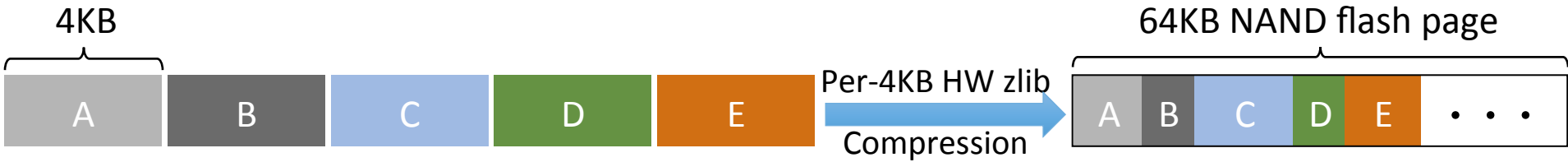


In-storage data path compression

In-Storage Data Path Compression

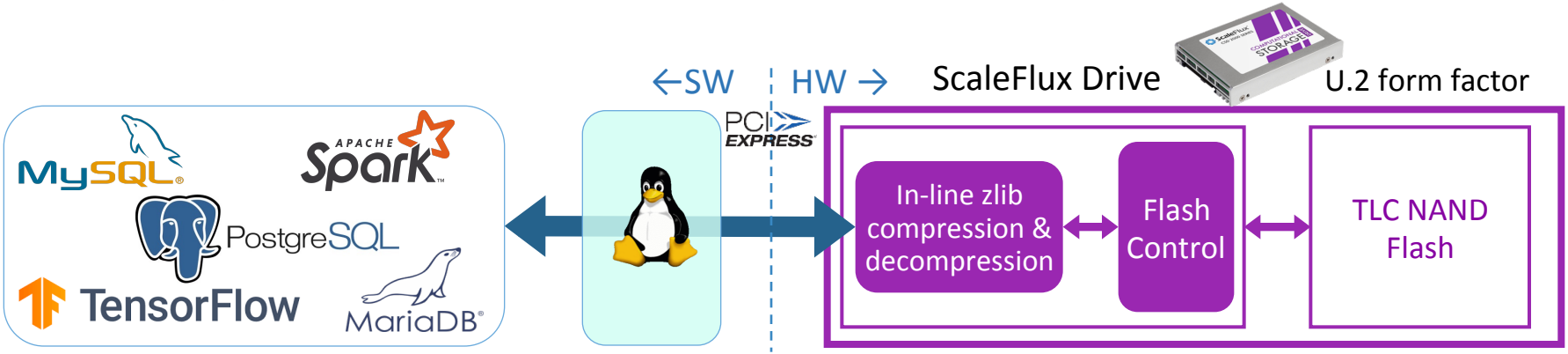


- Data path per-4KB zlib compression (2.4GB/s) & decompression (4GB/s)

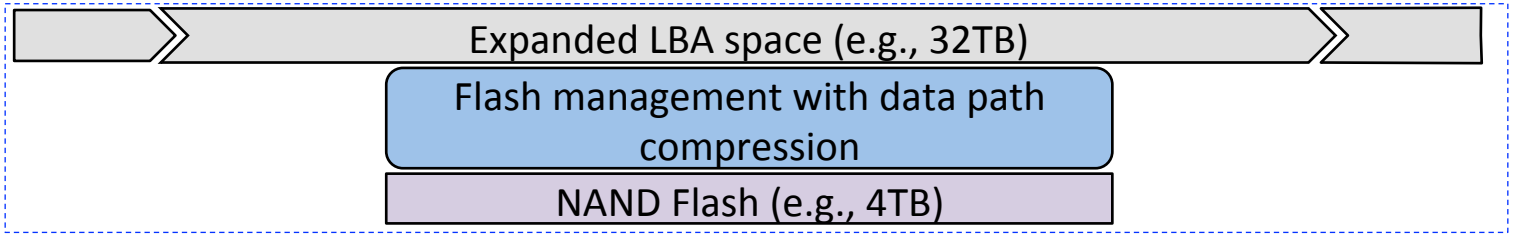


zlib compression → High compression ratio Tight packing → Zero space waste

In-Storage Data Path Compression



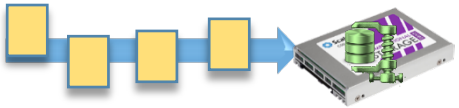
- ❑ Completely transparent to OS and user applications
- ❑ Expose an expanded LBA space to materialize cost saving



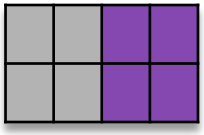
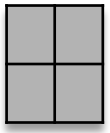
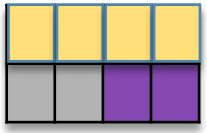
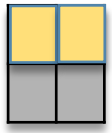
Managing Logical & Physical Capacity

1. Initial installation & formatting
   - User selects 6.4TB Logical Capacity

2. User sends 3.2TB of User Data
   - Data is 2:1 compressible



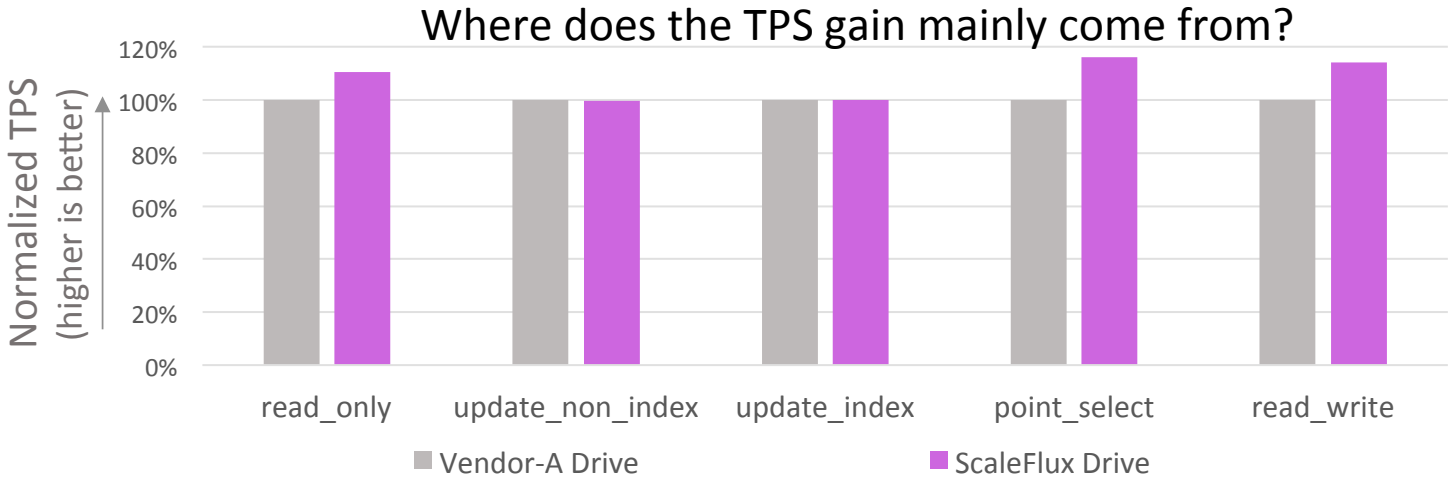
3. Compresses & writes the data
   - Data uses 1.6TB of physical capacity

	File System View (Logical)	Drive View (Physical)
		
	6.4TB Capacity	3.2TB Physical Capacity
	3.2TB User Data	1.6TB Used Physical Capacity
		
	3.2TB Free Capacity	1.6TB Free Physical Capacity

Transparent & Dynamic Free Space Management

Straightforward Usage

- ❑ Xeon E5-2667 v4 32-core @ 3.2GHz, 256GB DRAM
- ❑ CentOS 7.5.1804, Postgres 10.10, Sysbench 1.1.0 (64 threads)
- ❑ 3.2TB vendor-A NVMe drive vs. 3.2TB ScaleFlux drive
- ❑ In-storage compression: **2TB** Postgres dataset ➔ **776GB** (60% reduction)



Smaller storage footprint

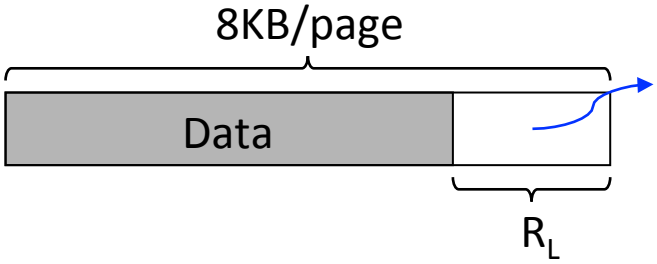
↓

Less flash memory access conflict

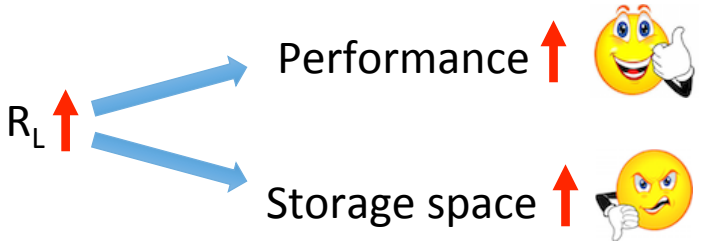
↓

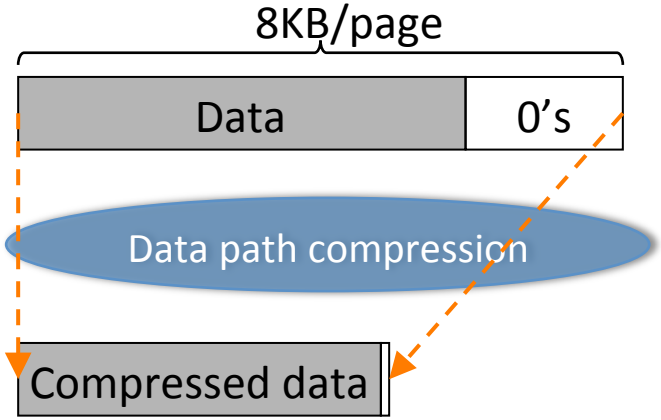
Higher page read throughput

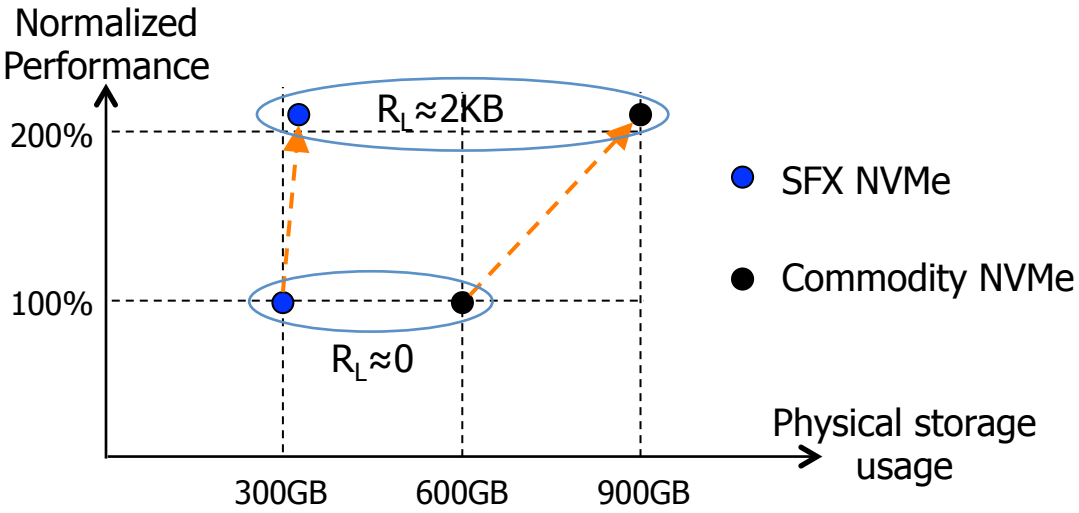
One Step Further



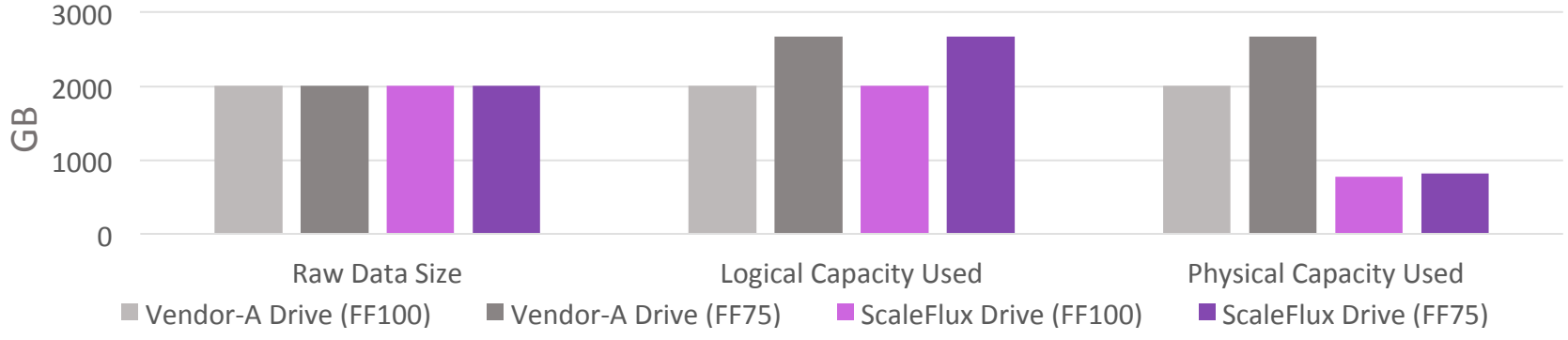
Reserved for future update (fillfactor)

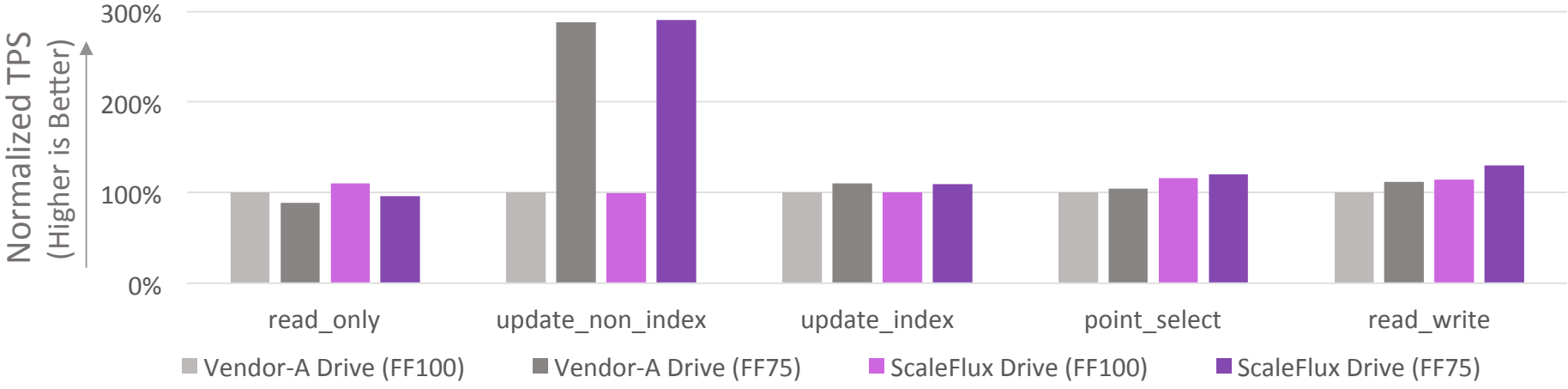






Reduce Fillfactor at Zero Storage Cost



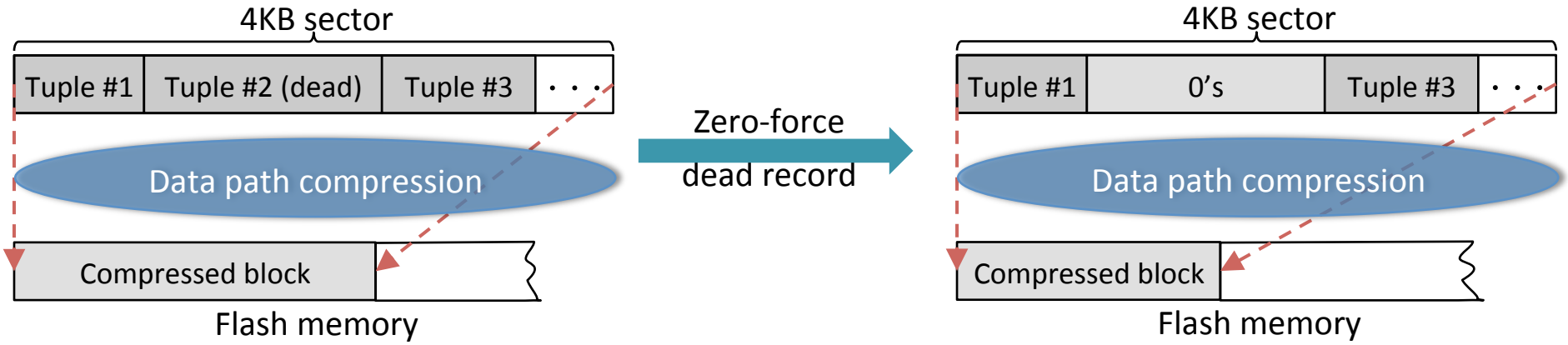


More Opportunities

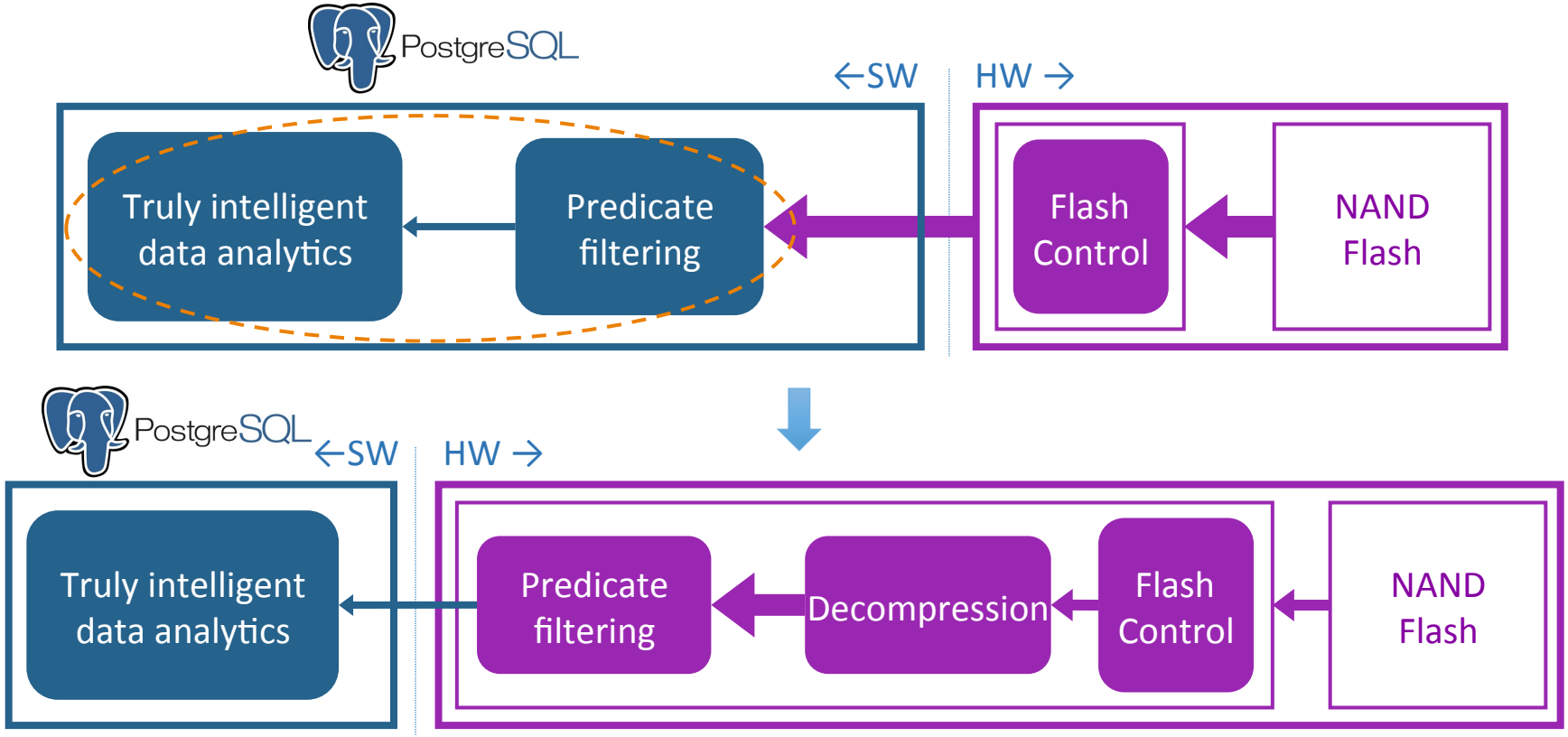
- ❑ The well-known **bloat** problem of Postgres
- ❑ Maintains UNDO within each table ➔ updating a tuple leaves a **dead tuple** in table

Updates ↑ → # of dead tuples ↑ → Storage waste ↑ → Vacuum → Performance ↓

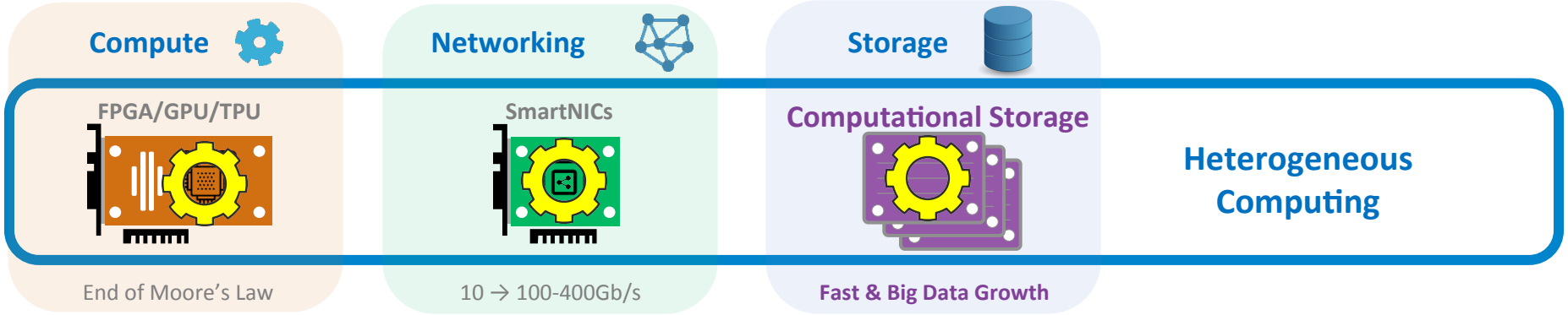
Zero-force dead tuples to reduce the vacuum activities



More Opportunities



Rise of Computational Storage



Tremendous Value-Add to Applications & Infrastructure

- Scale performance with increasing workload capacities
- Optimize total infrastructure footprint
- Highly adaptable to evolving modern applications

Summary

- **In-storage data path compression: The perfect match with Postgres**
- **Volume deployment today**
  - The most advanced Computational Storage Drive
  - Immediate impact of compute AND storage I/O acceleration

Questions?

www.scaleflux.com

tong.zhang@scaleflux.com

ScaleFlux™

Computational Storage for Data-Driven Applications

Thank You

97 East Brokaw Road, Suite 260
San Jose, CA 95112
www.scaleflux.com #compute2data