SDC 20

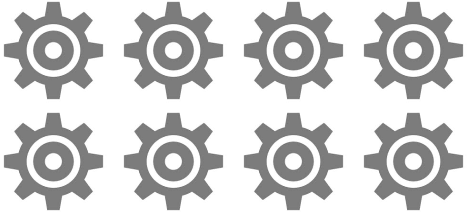
BY Developers FOR Developers

Storage Developer Conference
September 22-23, 2020

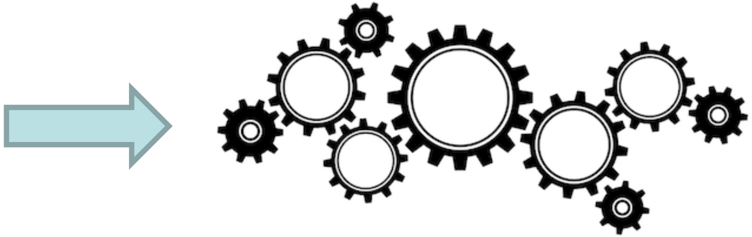
The True Value of Storage Drives with Built-in Transparent Compression: Far Beyond Lower Storage Cost

Tong Zhang
ScaleFlux Inc.
San Jose, CA

The Rise of Computational Storage



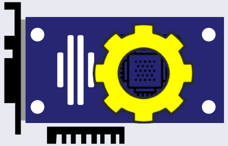
Homogeneous Computing



Heterogenous Computing

Compute

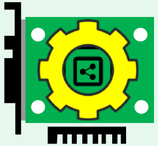
FPGA/GPU/TPU



End of Moore's Law

Networking

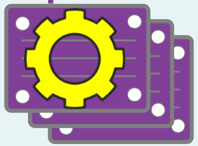
SmartNICs



10 → 100-400Gb/s

Storage

Computational



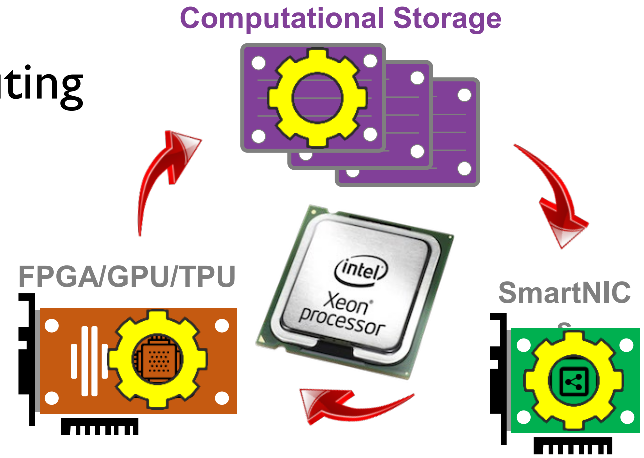
Fast & Big Data Growth

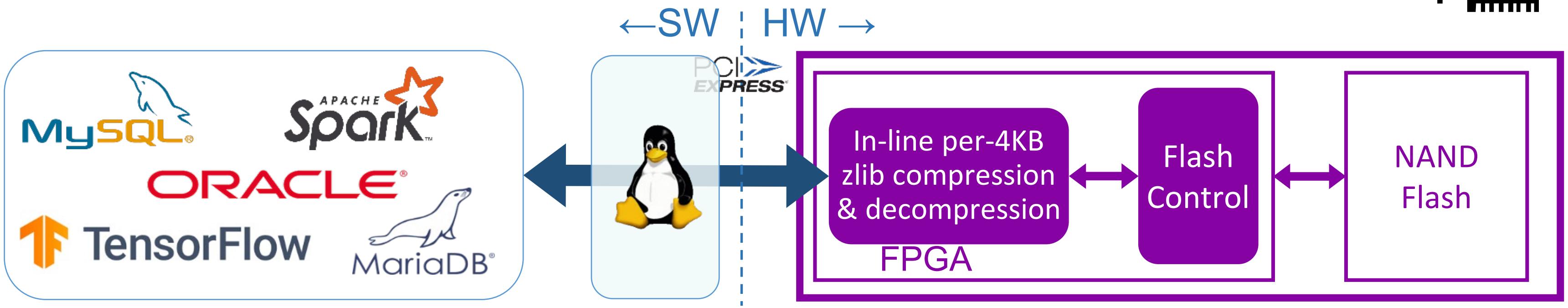
Domain Specific Compute

Computational Storage: A Very Simple Idea

- ❑ End of Moore's Law ➔ heterogeneous computing

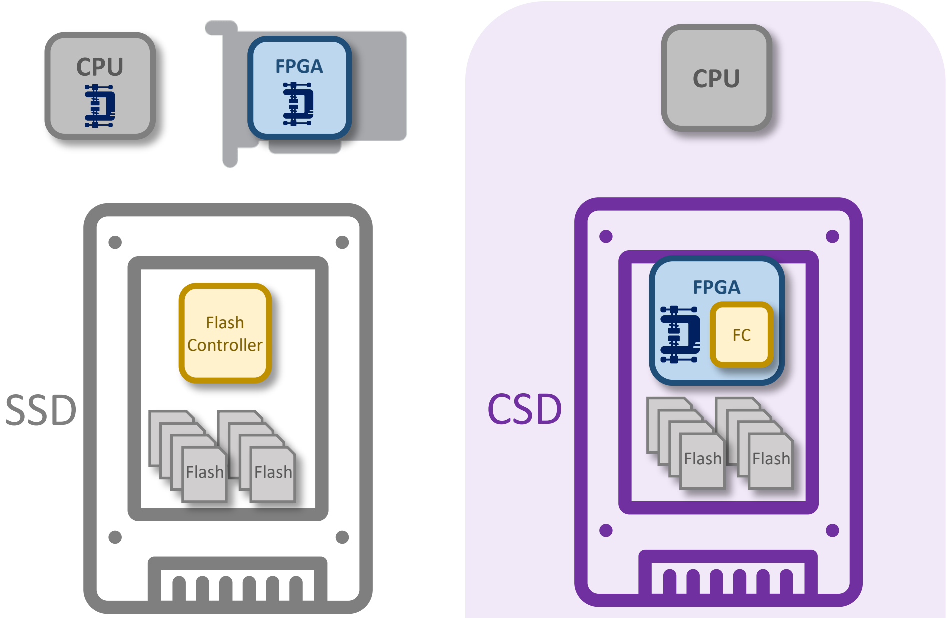
Low-hanging fruits





Computational Storage Drive (CSD) with Data Path Transparent Compression

ScaleFlux Computational Storage Drive: CSD 2000

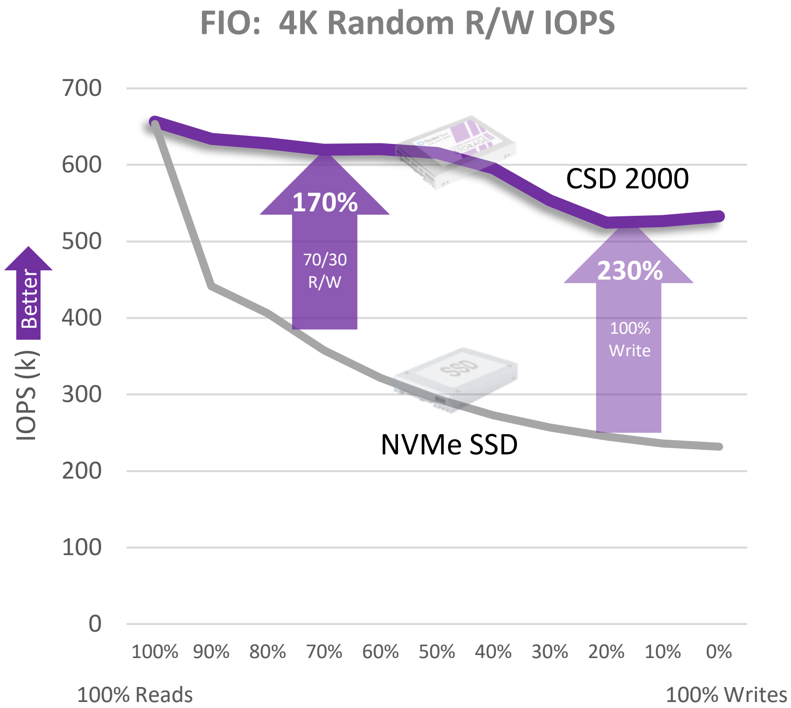


Multiple, discrete components for Compute and SSD Functions

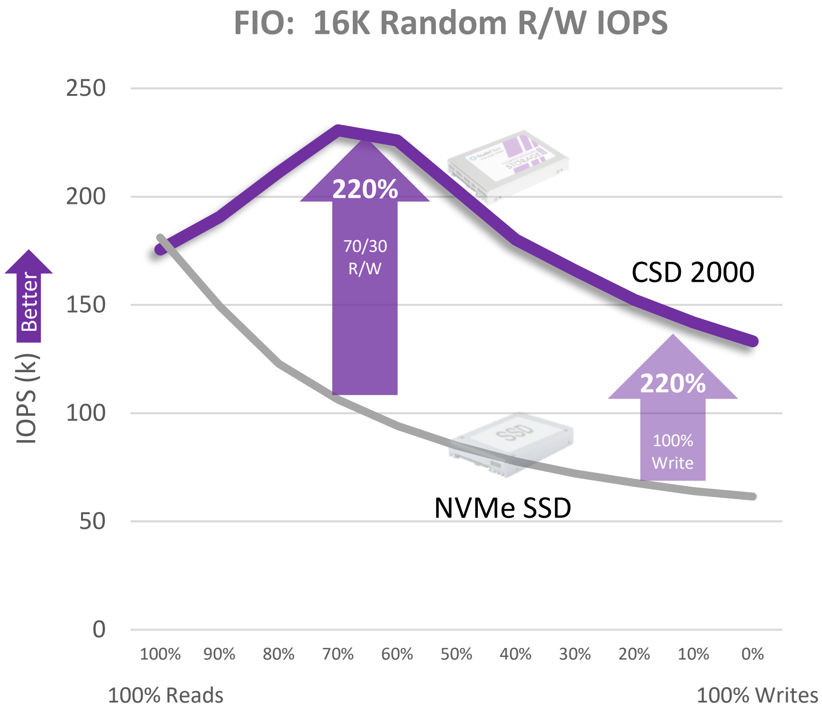
Single FPGA combines Compute and SSD Functions

- ✓ Complete, validated solution
  - ✓ Pre-Programmed FPGA
  - ✓ Hardware
  - ✓ Software
  - ✓ Firmware
- ✓ No FPGA knowledge or coding
- ✓ Field upgradeable
- ✓ Standard U.2 & AIC form factors

CSD 2000: Data Path Transparent Compression

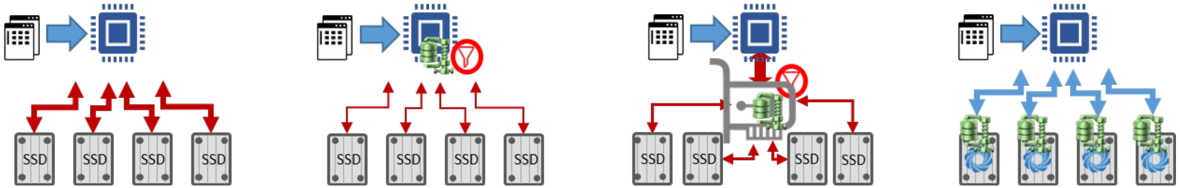


2.5:1 Compressible Data, 8 jobs, 32 QD, steady state after preconditioning



2.5:1 Compressible Data, 8 jobs, 32 QD, steady state after preconditioning

Comparing Compression Options

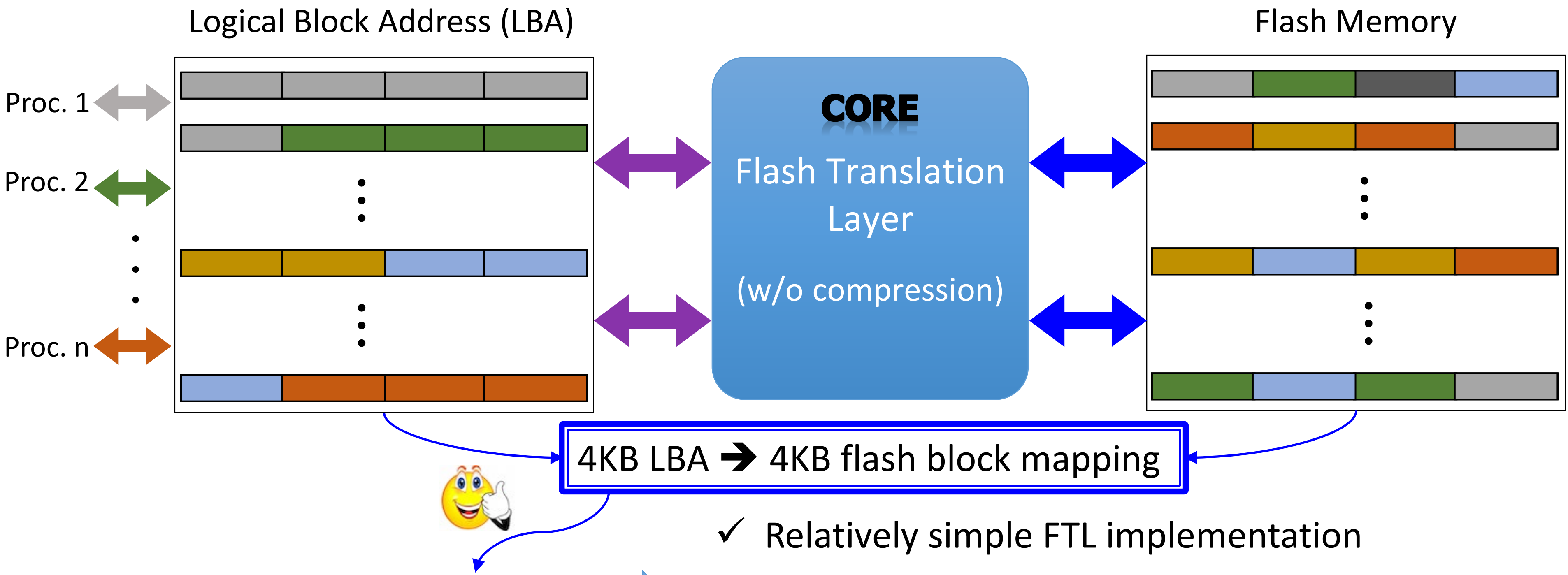


	No Compression	Host-Based	Offload Card	CSD 2000
No CPU Overhead	✓	✗	✓	✓
Reduced $/User GB	✗	✓	✓	✓
Performance scales with capacity	✓	✗	✗	✓
Transparent App Integration	-	✗	✗	✓
Zero App Latency	✓	✗	✗	✓
No incremental power usage	✓	✗	✗	✓
No incremental physical footprint	✓	✓	✗	✓

Scalable CSD-based compression reduces Cost/GB without choking the CPU

CSD 2000

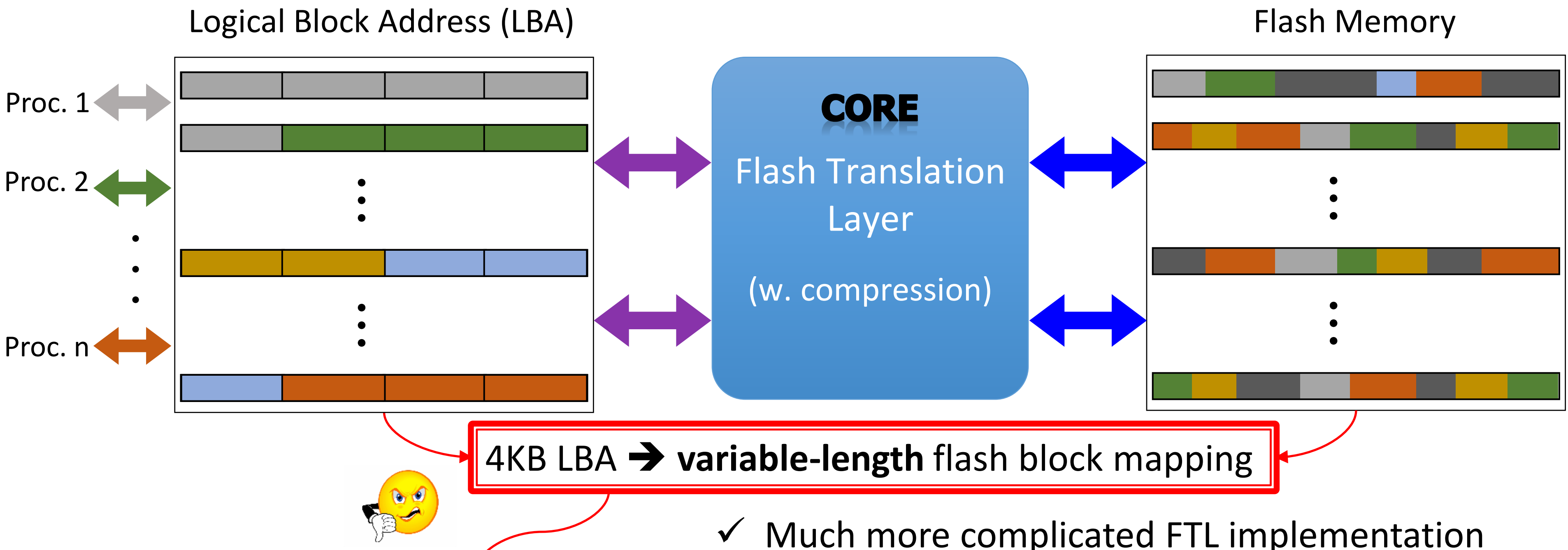
In-Storage Transparent Compression: Why is It Hard to Build?



Regularity & uniformity →

- ✓ Relatively simple FTL implementation
- ✓ Relatively easy to achieve high speed
- ✓ Relatively easy to ensure storage stability

In-Storage Transparent Compression: Why is It Hard to Build?



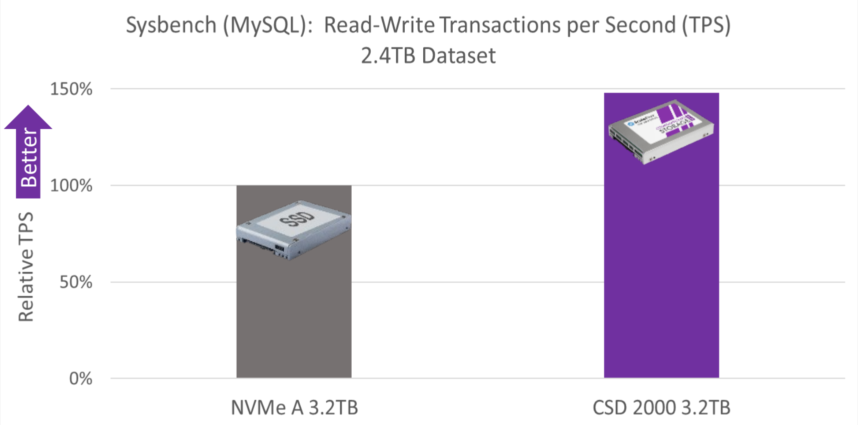
Irregularity & randomness →

- ✓ Much more complicated FTL implementation
- ✓ Much harder to achieve high speed
- ✓ Much harder to ensure storage stability

CSD 2000: Highest OLTP TPS, Lowest $/User GB

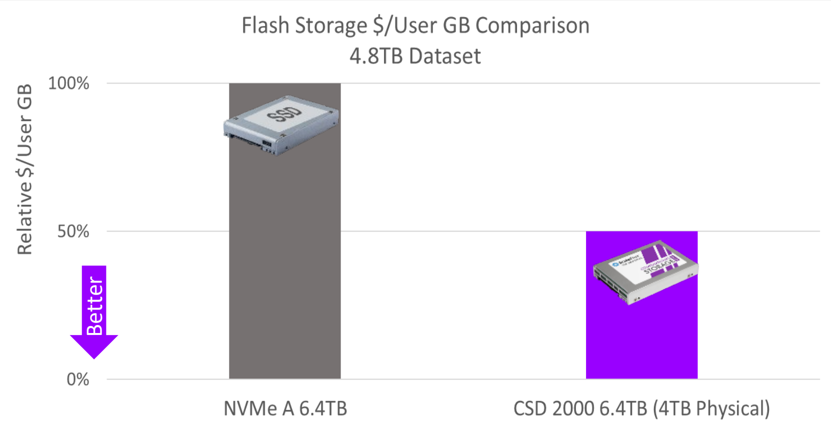
- Sysbench (MySQL 5.7.25, InnoDB)
- 50M records, 64 Threads
- 1hr Test run
- Intel(R) Xeon(R) CPU E5-2667 v4 @ 3.20GHz, 256GB DRAM

Performance: 150% TPS



2.4TB Dataset Physical Flash consumed on NVMe A; 0.9TB on CSD 2000

Cost: 50% Less $/User GB



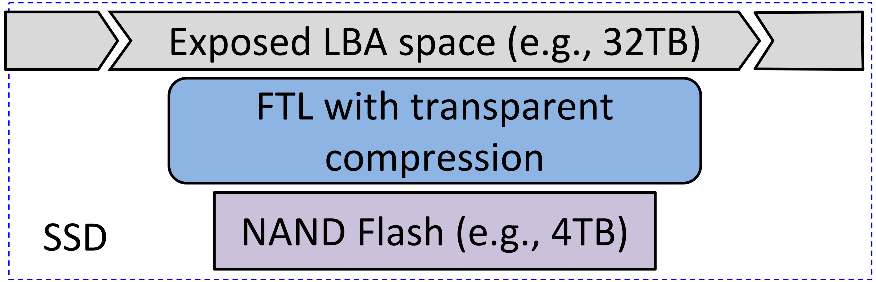
4.8TB Dataset Physical Flash consumed on NVMe A; 1.6TB on CSD 2000
CSD 2000 delivers 30% higher Read-Write TPS in this cost comparison

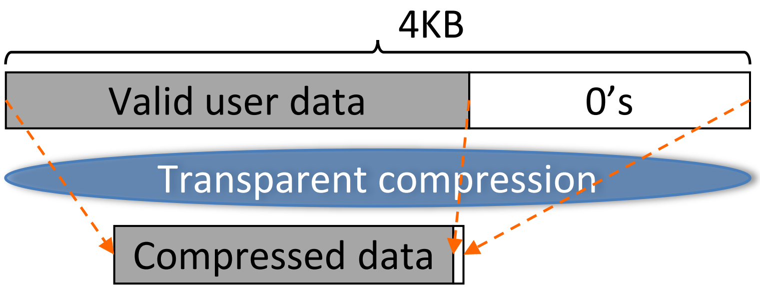
Flexible Drive Capacity Enables the Best Performance ↔ Cost

Open a Door for System Innovation

Logical storage space utilization efficiency

Physical storage space utilization efficiency



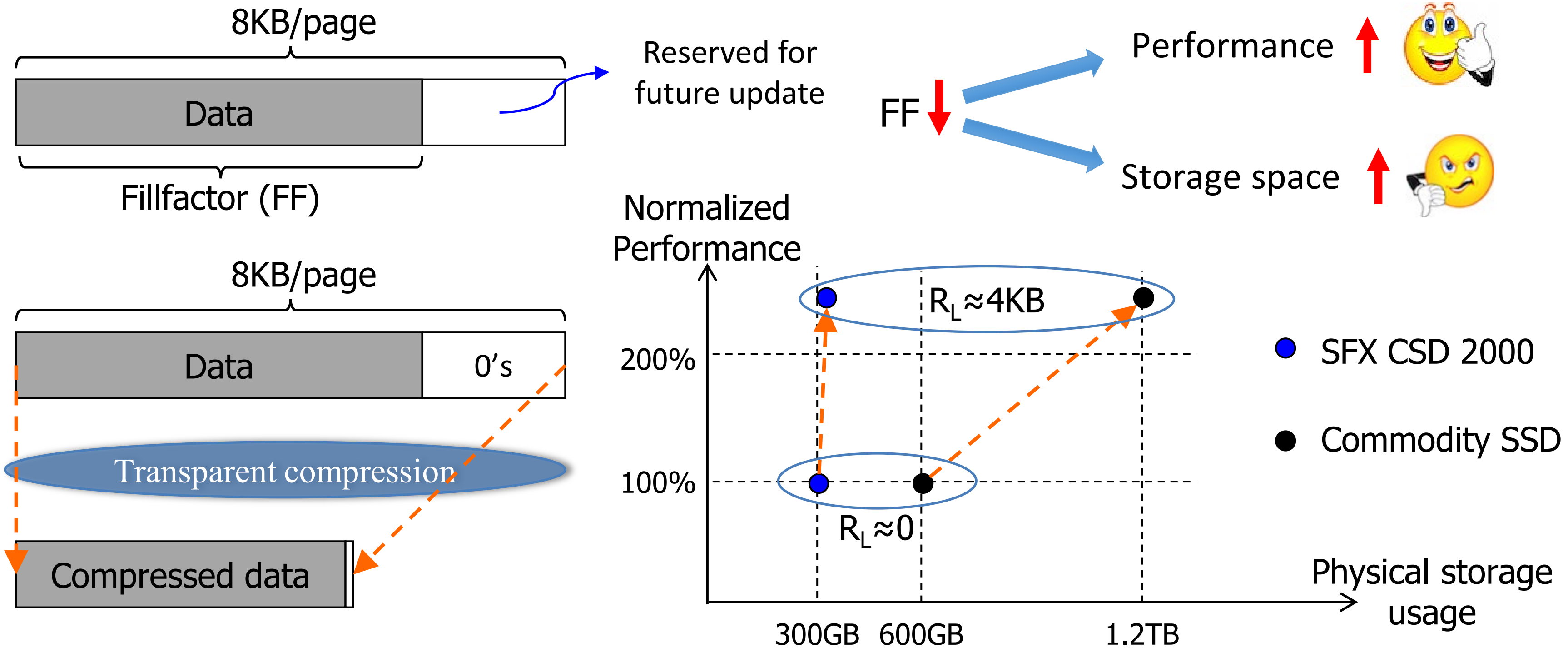


Unnecessary to use all the LBAs

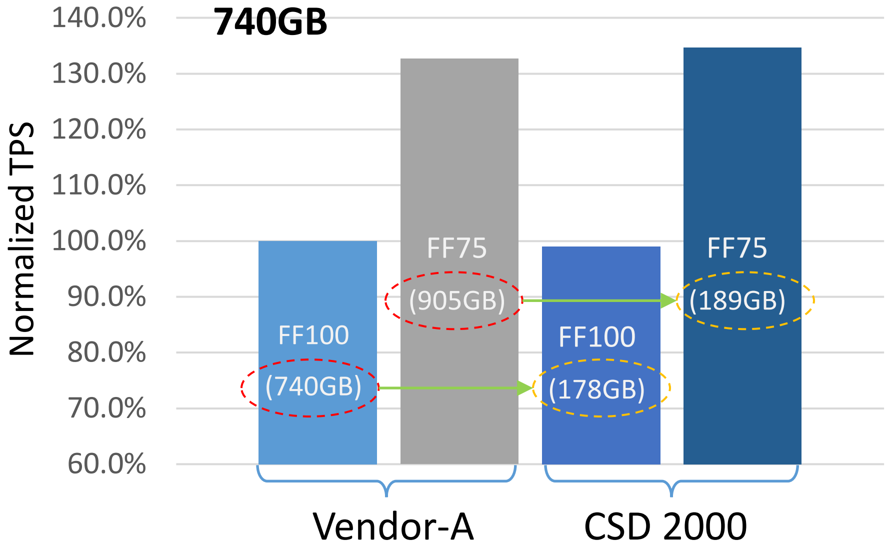
Unnecessary to fill each 4KB sector with user data

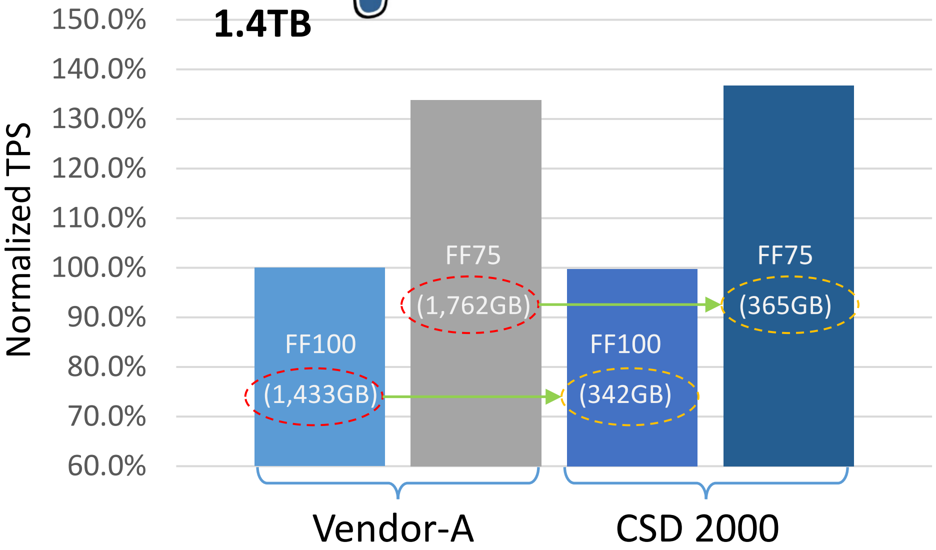
OS/Applications can ***purposely waste*** logical storage space to gain benefits

Case Study I: PostgreSQL



Case Study 1: PostgreSQL



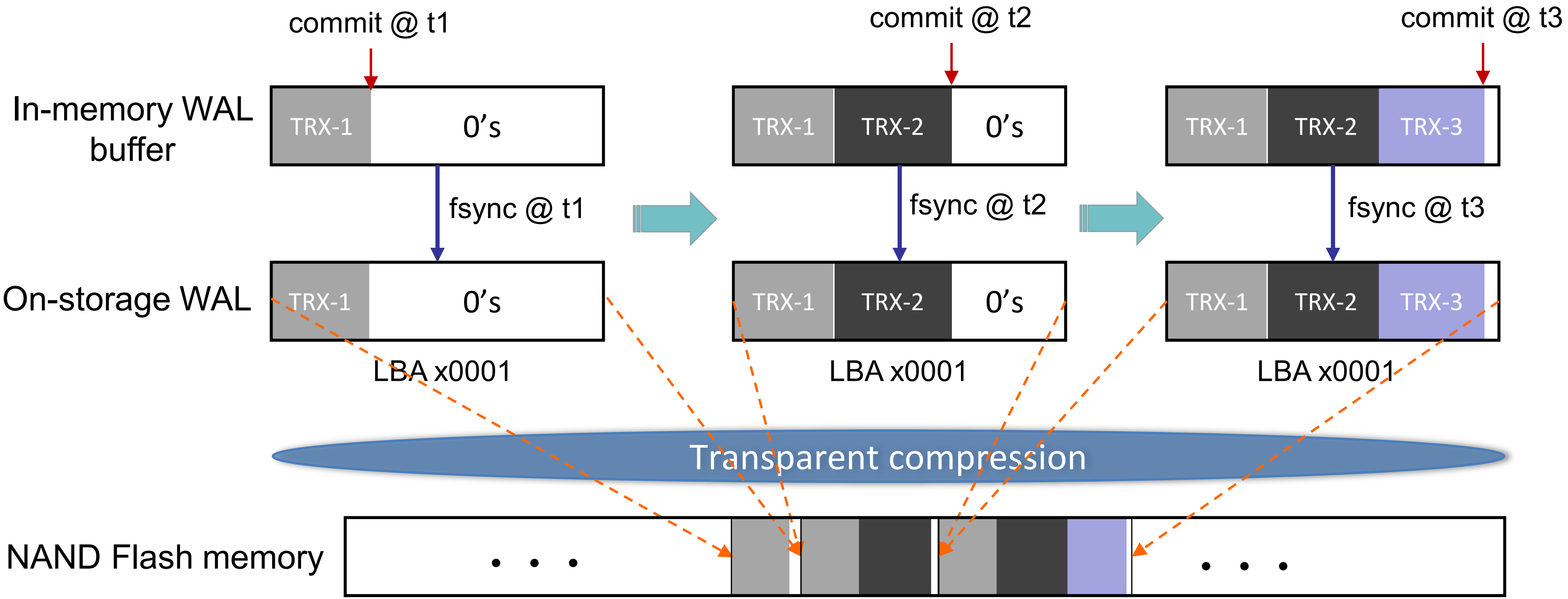


Fillfactor	Drive	Logical size (GB)	Physical size (GB)	Comp Ratio
100	Vendor-A	740	740	1.00
	CSD 2000		**178**	4.12
75	Vendor-A	905	905	1.00
	CSD 2000		**189**	4.75

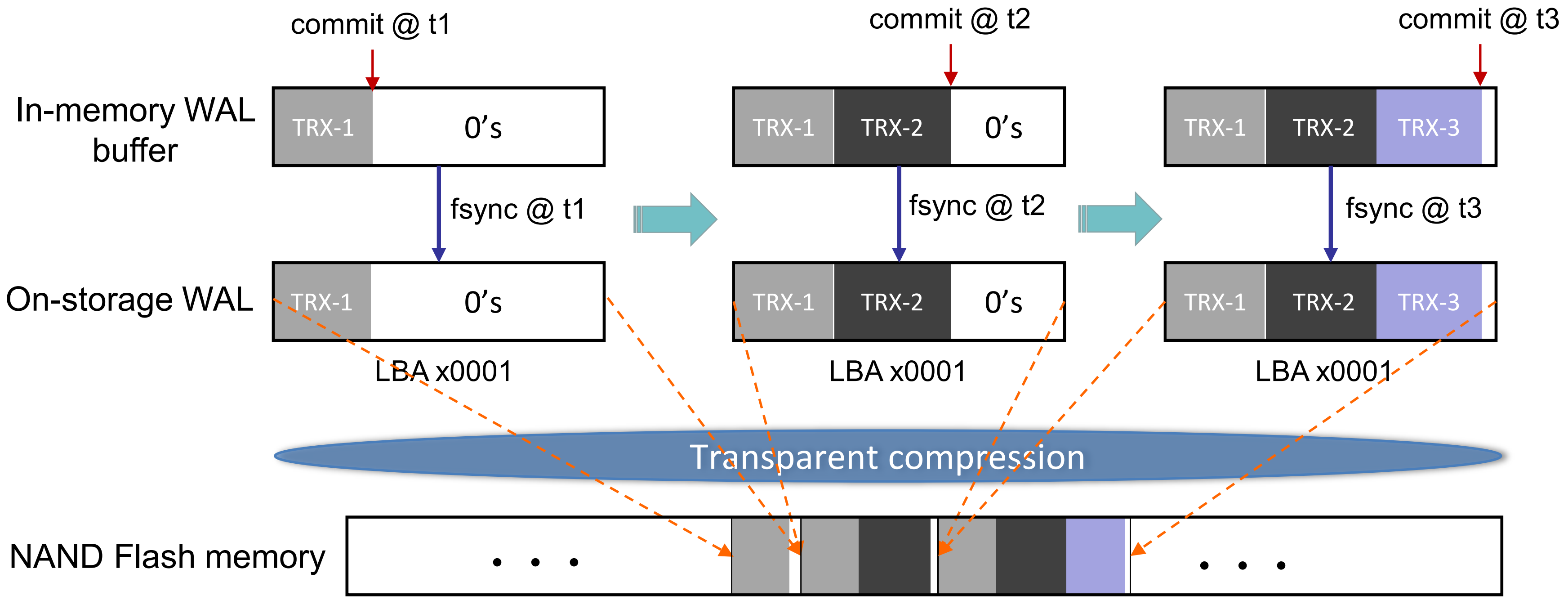
Fillfactor	Drive	Logical size (GB)	Physical size (GB)	Comp Ratio
100	Vendor-A	1,433	1,433	1.00
	CSD 2000		**342**	4.19
75	Vendor-A	1,762	1,762	1.00
	CSD 2000		**365**	4.82

Case Study 2: Sparse Write-Ahead Logging

- ❑ Write-ahead logging (WAL)
  - ➢ Universally used by data management systems to achieve *atomicity* and *durability*



Case Study 2: Sparse Write-Ahead Logging

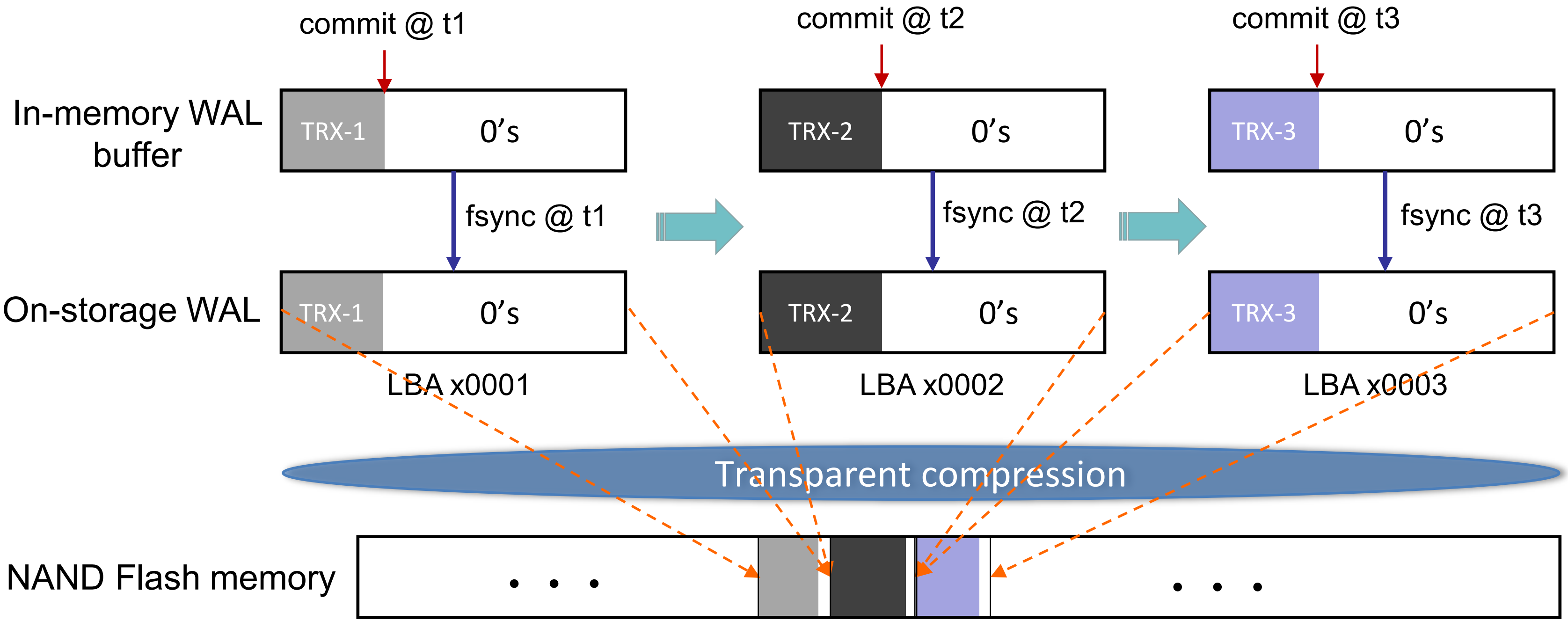


Write amplification → More interference with other IOs ☹

Write amplification → Shorter NAND flash memory lifetime ☹

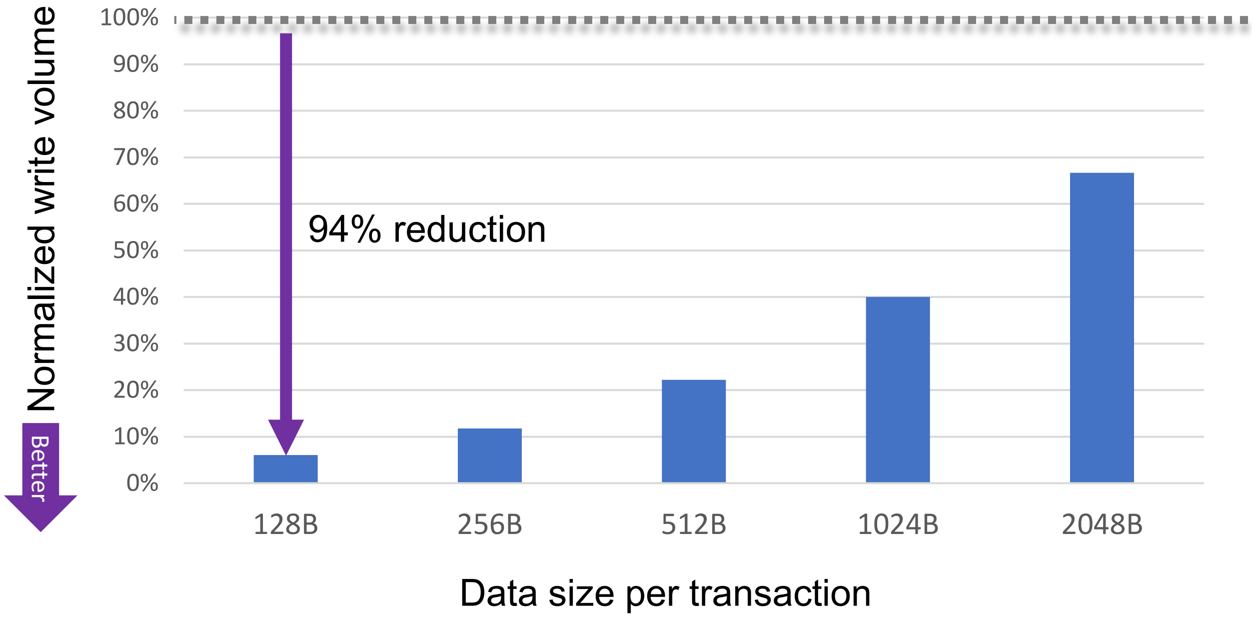
Case Study 2: Sparse Write-Ahead Logging

- Sparse WAL: Allocate a new 4KB sector per transaction commit
  - ✓ Waste logical storage space ➔ reduce WAL-induced write amplification



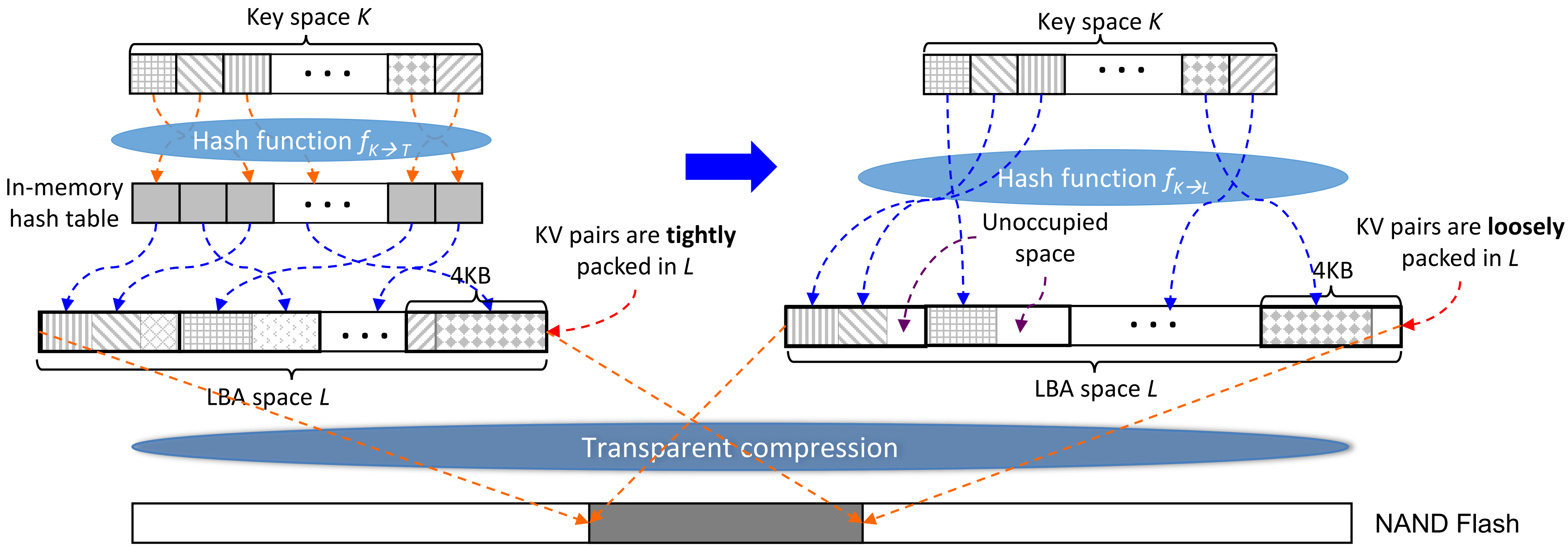
Case Study 2: Sparse Write-Ahead Logging

- Sparse WAL: Allocate a new 4KB sector per transaction commit
  - ✓ Waste logical storage space ➔ reduce WAL-induced write amplification



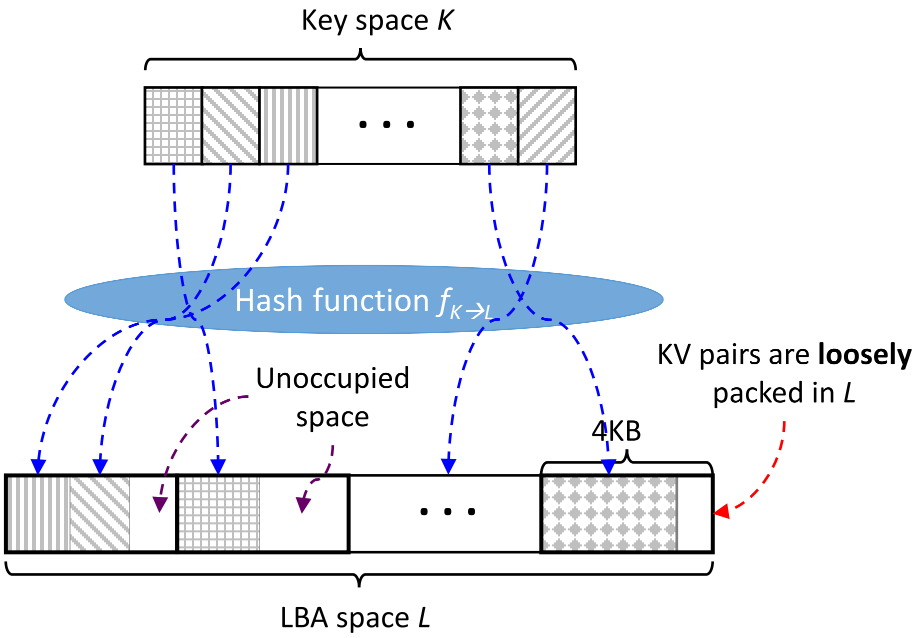
Case Study 3: Table-less Hash-based KV Store

- Very simple idea
  - Hash *key space* directly onto *logical storage space* ➔ eliminate the in-memory hash table
  - Transparent compression eliminates the "unoccupied space" from physical storage space



Case Study 3: Table-less Hash-based KV Store

- ❑ Eliminate in-memory hash table
  - ✓ Very small memory footprint
  - ✓ High operational parallelism
  - ✓ Short data access data path
  - ✓ Very simple code base



- ❑ Under-utilize logical storage space
  - ✓ Obviate frequent background operations (e.g., GC and compaction)

⇨ High performance, low memory cost, and low CPU usage

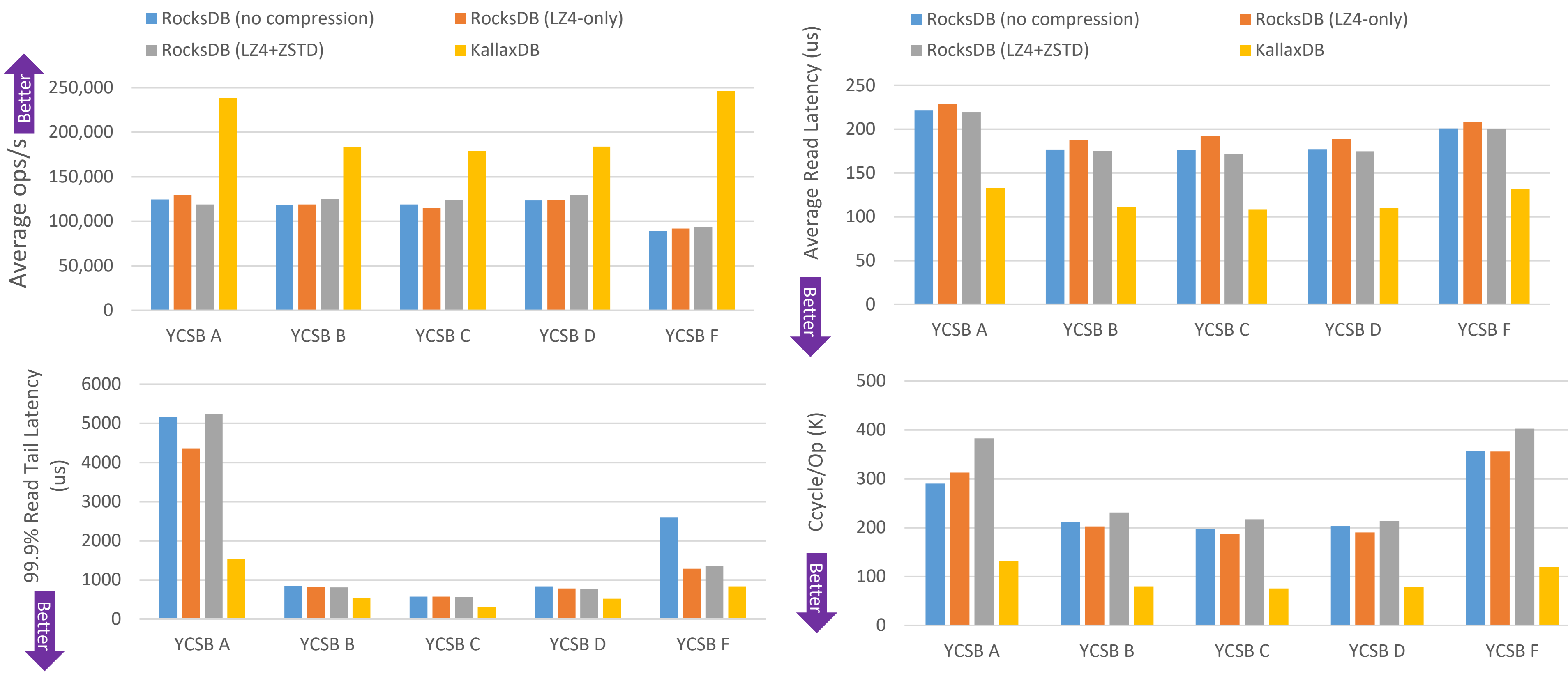
Case Study 3: Table-less Hash-based KV Store

- Experimental Setup
  - 24-core 2.6GHz Intel CPU, 32GB DDR4 DRAM, and a 3.2TB SFX CSD2000
  - RocksDB 6.10 (12 compaction threads and 4 flush threads)
  - 400-byte KV pair size, 1 billion KVs → 400GB raw data
  - Memory usage: RocksDB (5GB), KallaxDB (600MB)

YCSB A	50% reads, 50% updates
YCSB B	95% reads, 5% updates
YCSB C	100% reads
YCSB D	95% reads, 5% inserts
YCSB F	50% reads, 50% read-modify-writes

	Storage Usage
RocksDB (no compression)	428GB
RocksDB (LZ4-only)	235GB
RocksDB (LZ4+ZSTD)	201GB
KallaxDB	216GB

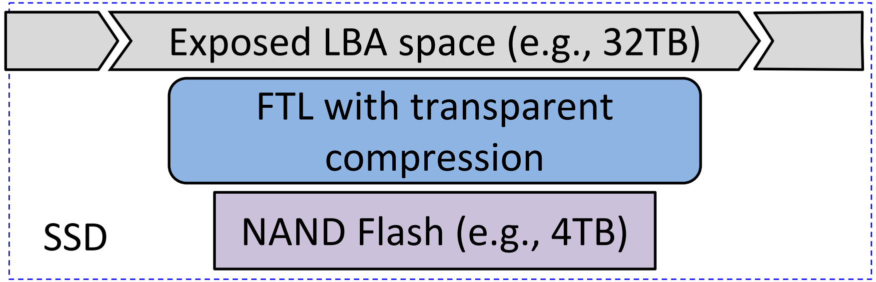
Case Study 3: Experimental Results (24 clients)

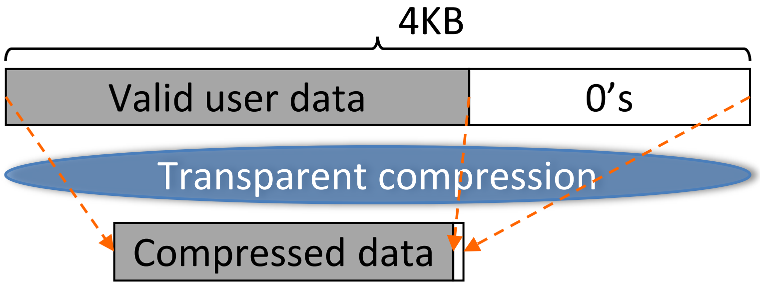


Open a Door for System Innovation

Logical storage space utilization efficiency

Physical storage space utilization efficiency



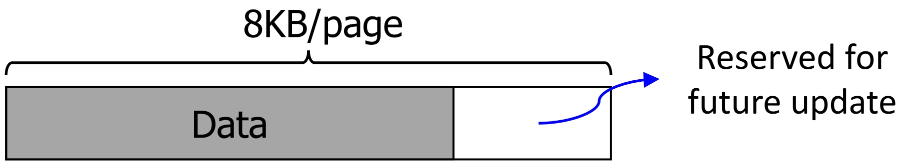


Unnecessary to use all the LBAs

Unnecessary to fill each 4KB sector with user data

OS/Applications can ***purposely waste*** logical storage space to gain benefits

Open a Door for System Innovation



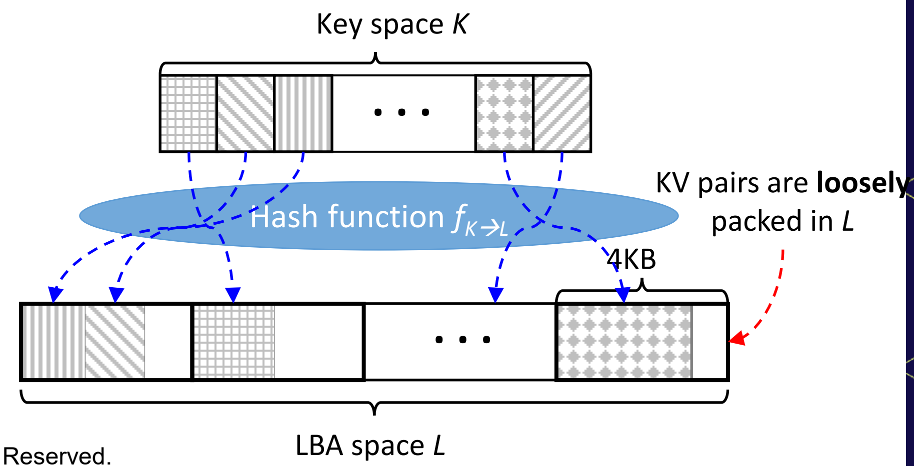
Reserve more space for future update to improve performance @ zero storage overhead

Sparse WAL ⇨ Reduce WAL-induced write amplification @ zero storage overhead

Table-less hash-based KV store

⇩

High performance, low memory/CPU usage @ zero storage overhead



Thank You

www.scaleflux.com

info@scaleflux.com

tong.zhang@scaleflux.com