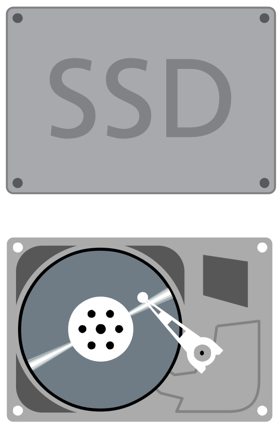
NVM Software Interfaces
New Directions

Nisha Talagala

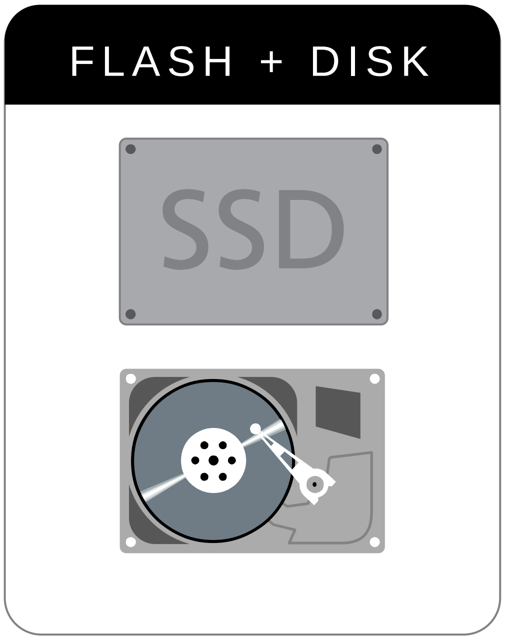
Evolution of Flash Adoption

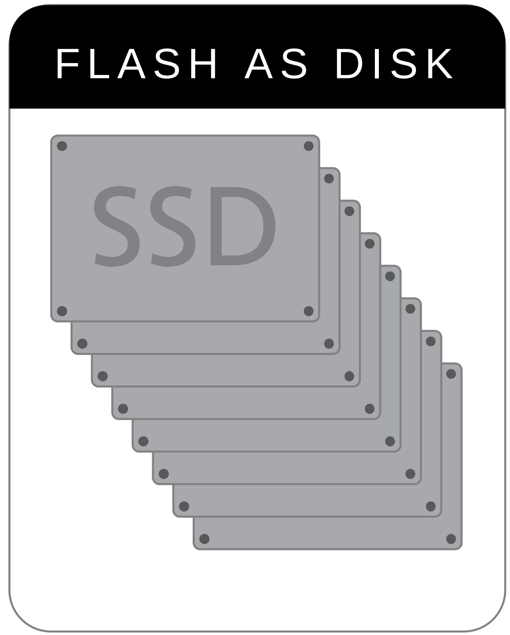
FUSION-IO

FLASH + DISK



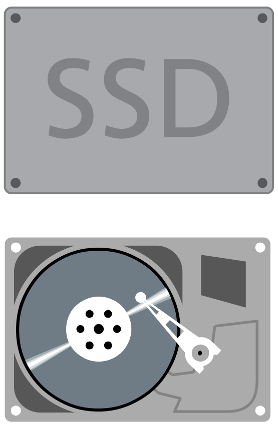
Evolution of Flash Adoption



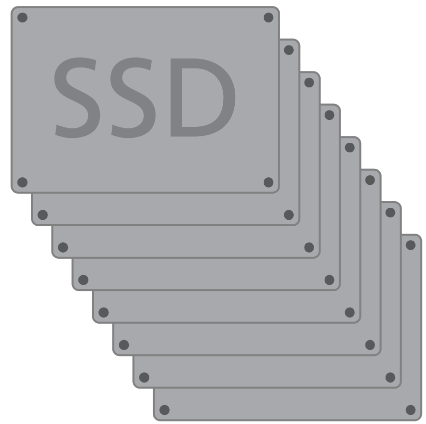


Evolution of Flash Adoption

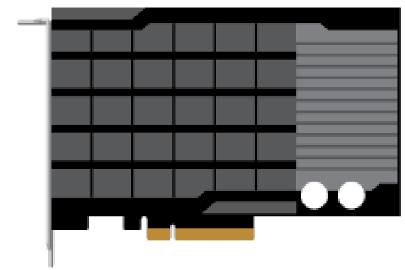
FLASH + DISK



FLASH AS DISK



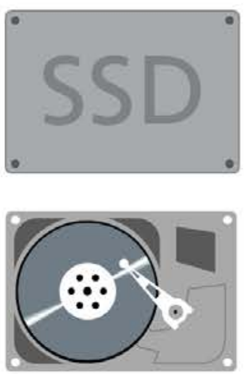
FLASH AS MEMORY



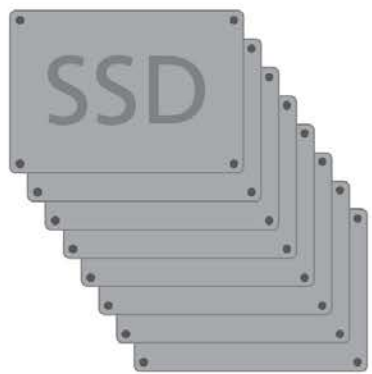
Evolution of Flash Adoption

FUSION-IO

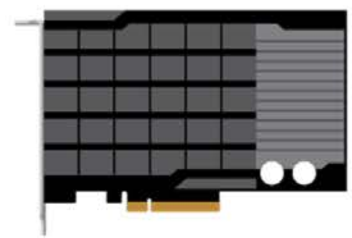
FLASH + DISK



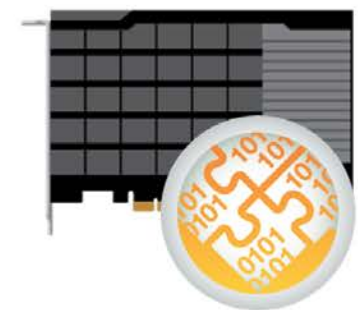
FLASH AS DISK



FLASH AS MEMORY



NATIVE APIs



Conventional I/O Access

FUSION-iO

APPLICATION

Application source code

Conventional I/O access

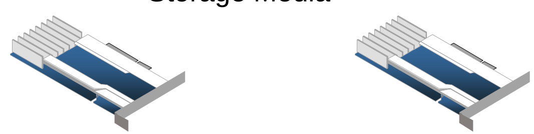
Simple Block | **Network File** | **Simple Block**

Proprietary Storage OS

Storage Media

Native Flash Translation Layer

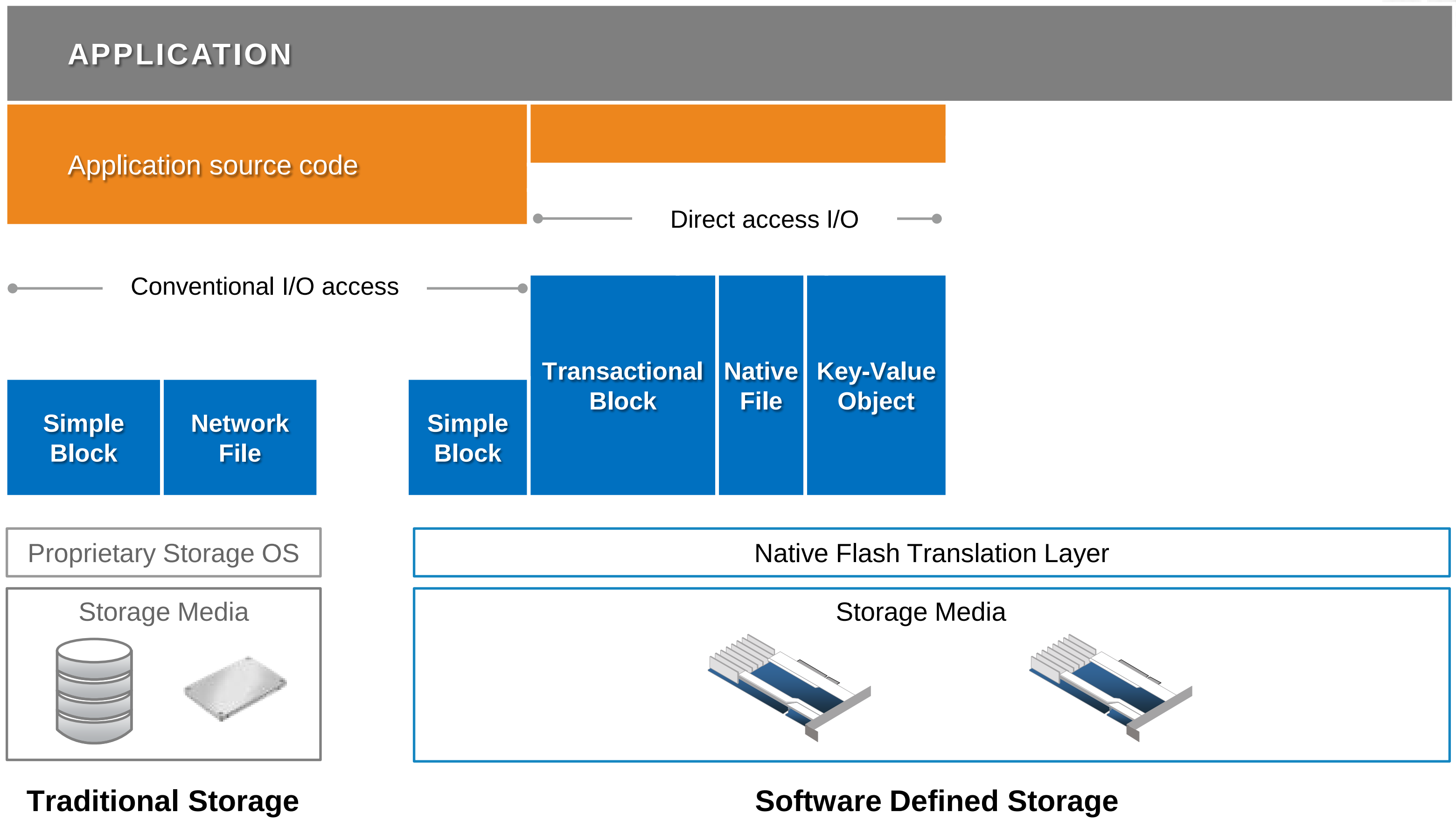
Storage Media



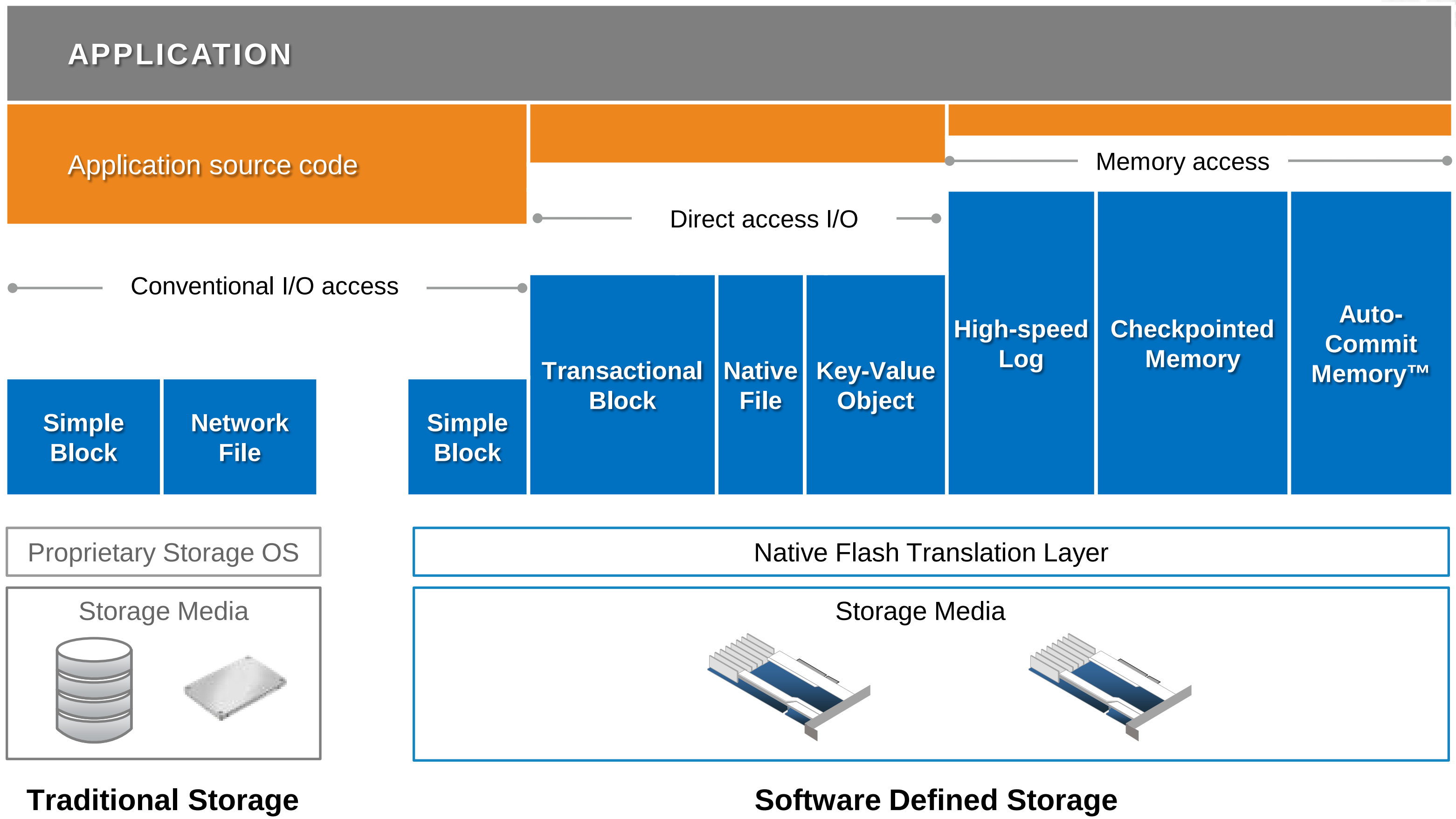
Traditional Storage

Software Defined Storage

Direct-Access I/O through Native Interfaces



Memory-access through Native Interfaces



New Primitives for a New Type of Media

Tape	Open, read, write, rewind, close.
Disk	Open, read, write, seek, close.
SSD	Open, read, write, seek, close.
ioMemory NVM	Open, read, write, seek, close. *Plus, new primitives to exploit characteristics of non-volatile memory* Basic write + atomic write, conditional write. Basic write + TTL expiry for auto-deletion. Basic mmap + crash-safety, versioning.

ATOMIC I/O Primitives: Sample Uses and Benefits

FUSION-iO

Databases

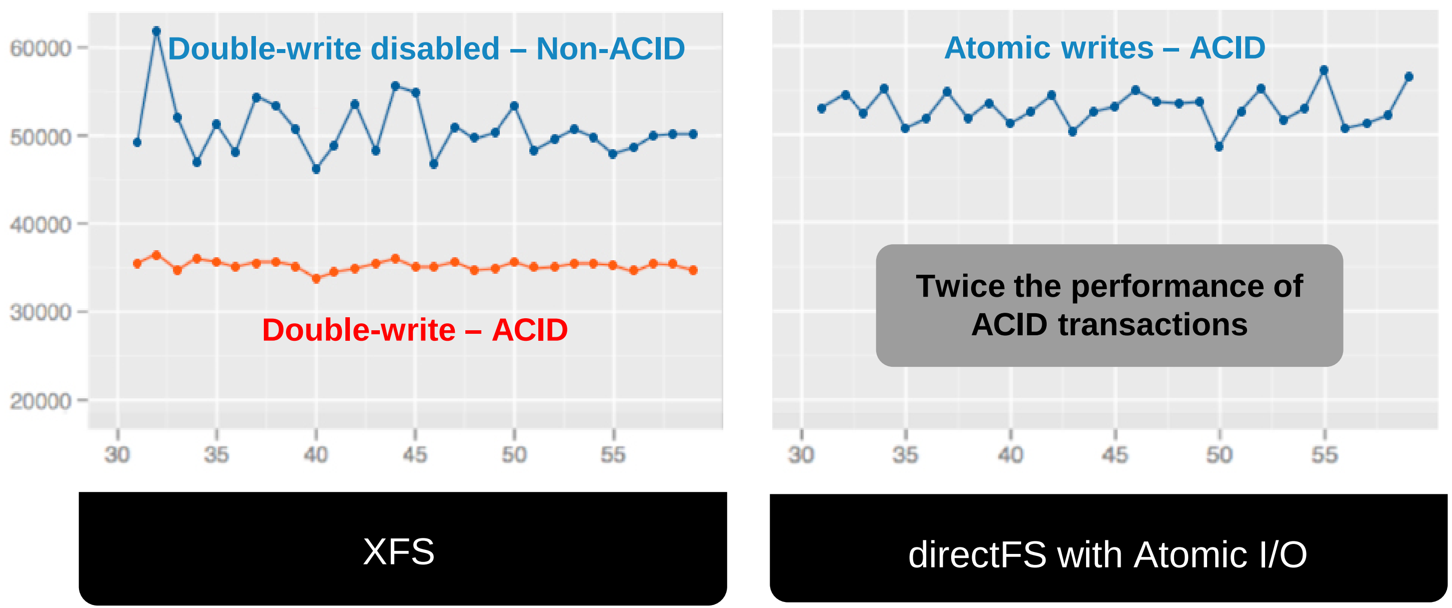
Transactional Atomicity:
Replace various workarounds implemented in database code to provide write atomicity (double-buffered writes, etc.)

Filesystems

File Update Atomicity:
Replace various workarounds implemented in filesystem code to provide file/directory update atomicity (journaling, etc.)

- 99% performance of raw writes
  Smarter media now natively understands atomic updates, with no additional metadata overhead.
- 2x longer flash media life Atomic Writes increase the life of flash media up to 2x due to reduction in write-ahead-logging and double-write buffering.
- 50% less code in key modules
  Atomic operations dramatically reduce application logic, such as journaling, built as work-arounds.

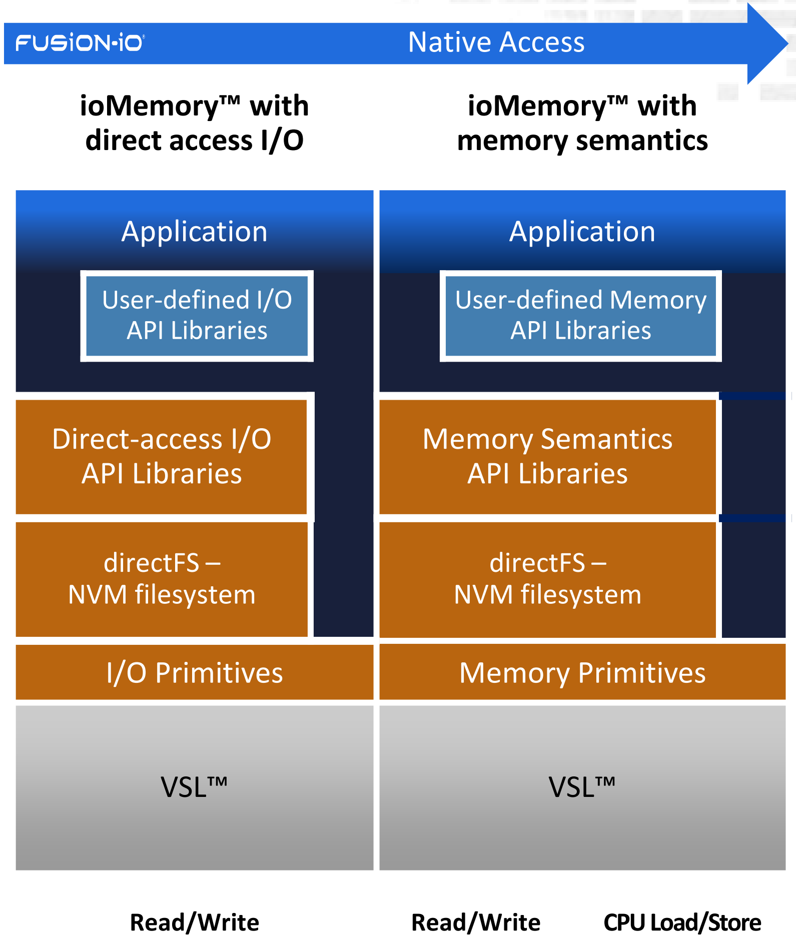
MySQL with Atomic Writes



Native File System Access

FUSION-IO

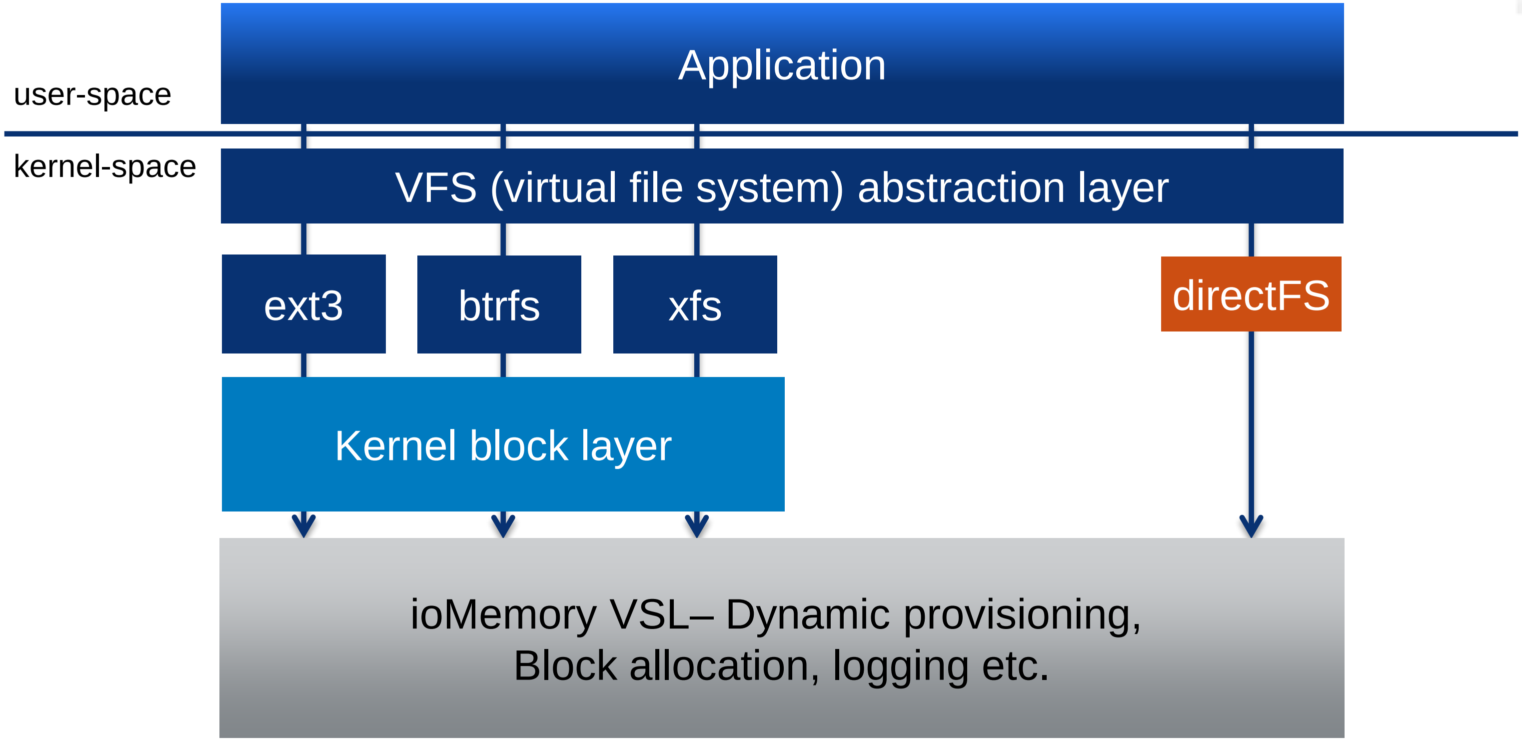
- Leverages native flash capabilities for file system acceleration
- File services layer
- Consumes and uses native primitives
- Exports primitives for use by applications
- Applications can run on either devices or filesystems



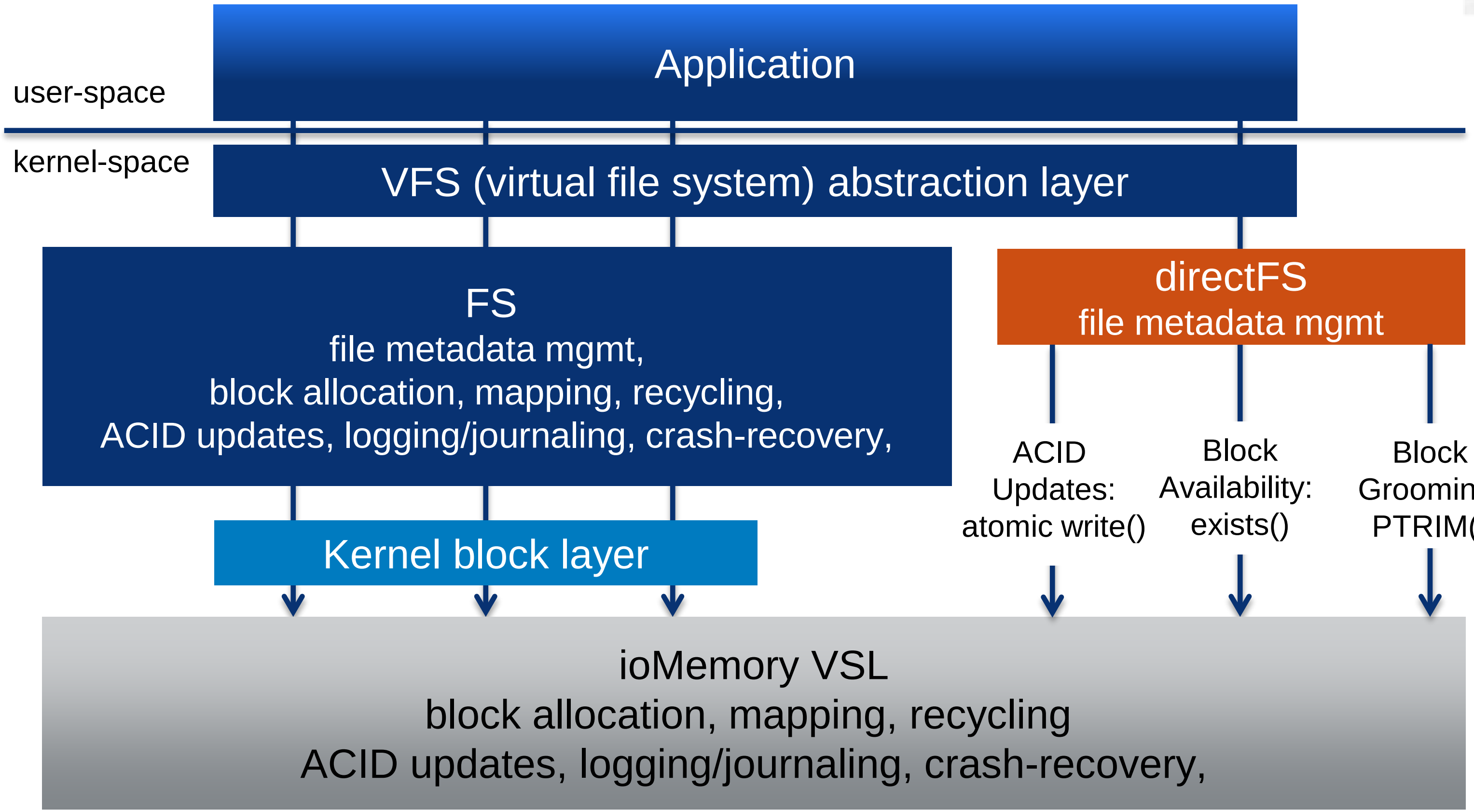
directFS

- Appears as any other file system in Linux
- Applications can use directFS file system unmodified with performance benefits
- Focuses only on file namespace
- Employs virtualized flash storage layer's logic for:
  - Large virtualized addressed space
  - Direct flash access
  - Crash recovery mechanisms
- Exports Primitives through file namespace
- Applications can use primitives through directFS or directly to device

directFS: Native File Name Space for NVM

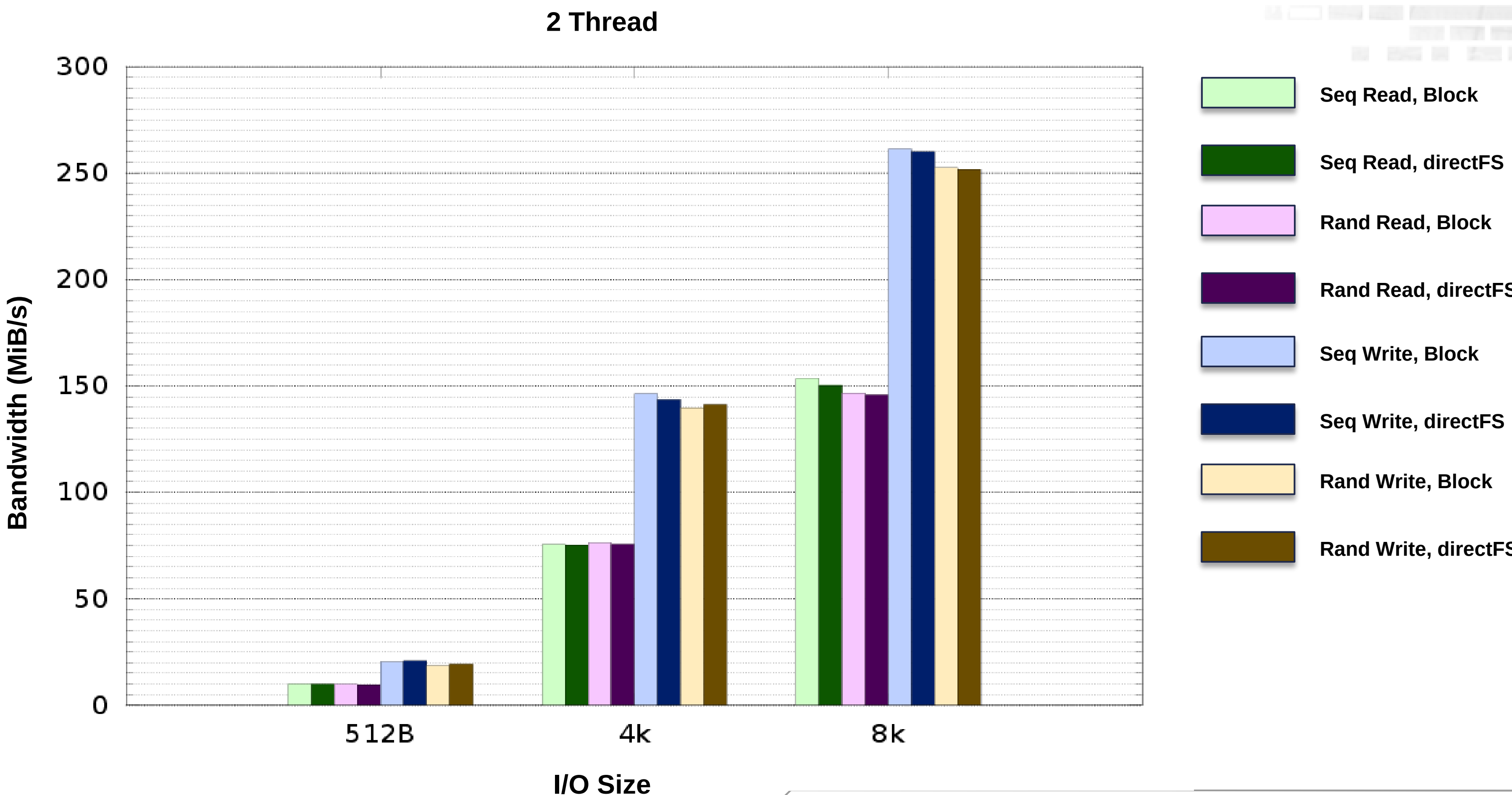


directFS – Eliminating duplicate logic

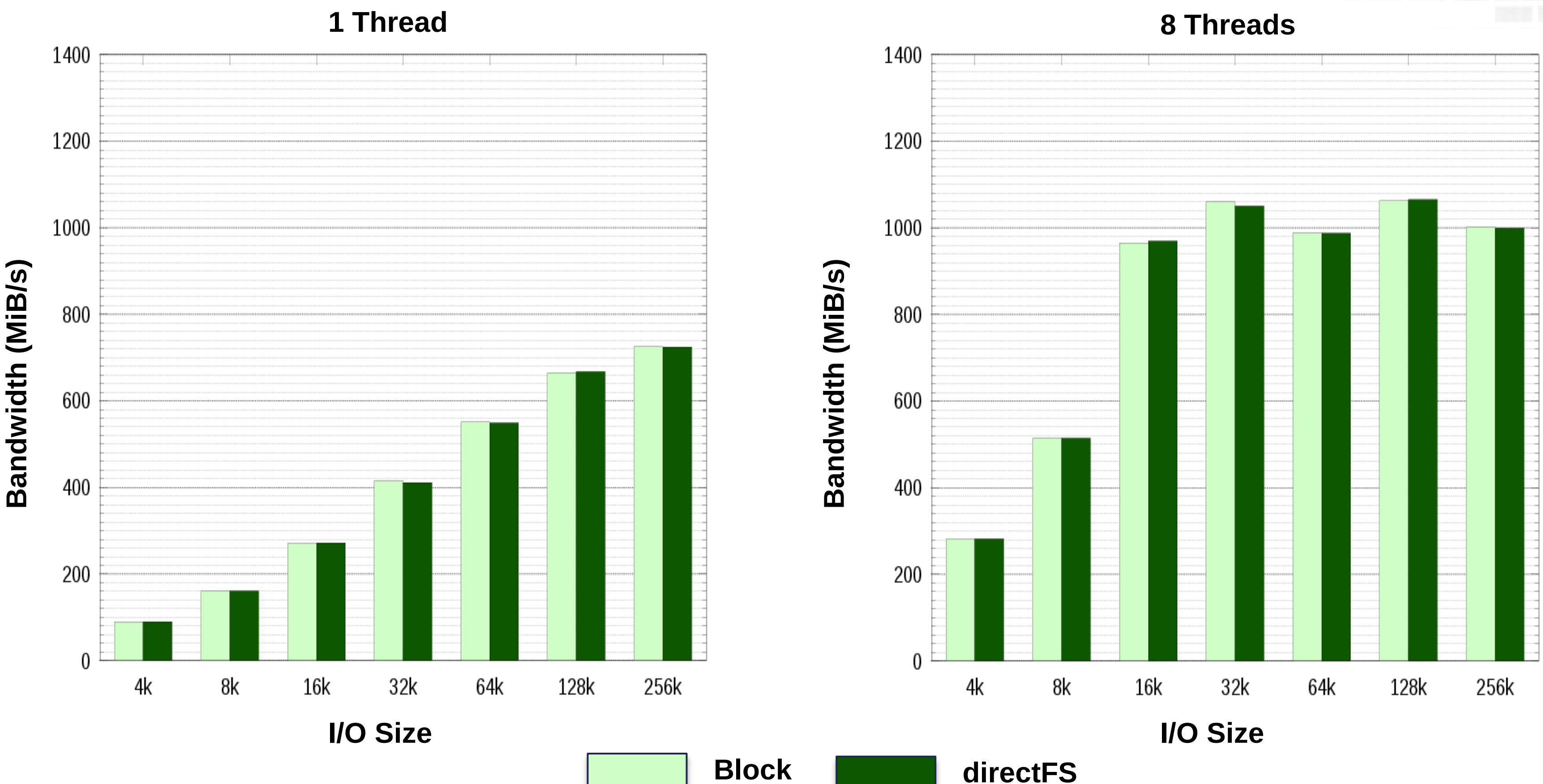


Performance PARITY BETWEEN BLOCK AND DIRECTFS: micro-benchmarks

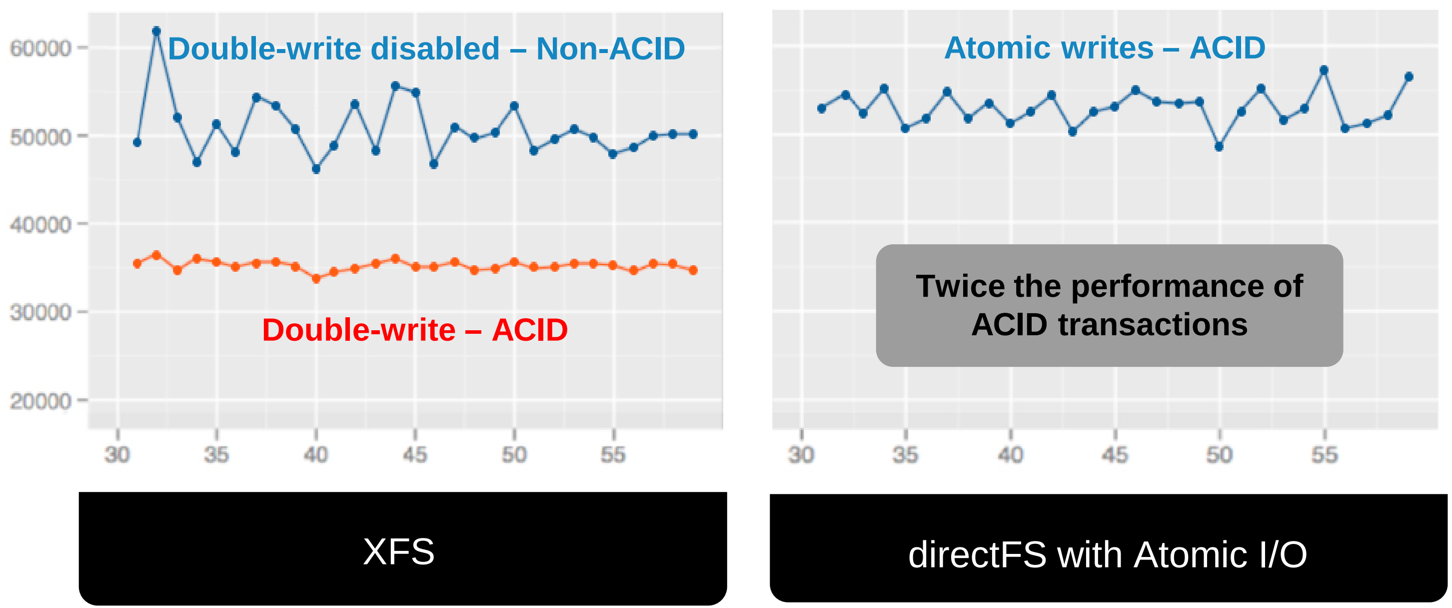
FUSION-iO



Performance PARITY BETWEEN BLOCK AND DIRECTFS: atomic writes



MySQL on directFS with Atomic Writes



Key-value store API Library: Sample Uses and Benefits

FUSION-IO

NoSQL Applications

Reduce overprovisioning due to lack of coordination between two-layers of garbage collection (application-layer and flash-layer). Some top NoSQL applications recommend over-provisioning by 3x due to this.

Reduce application I/Os through batched put and get operations.

Increase performance by eliminating packing and unpacking blocks, defragmentation, and duplicate metadata at app layer.

- **95% performance of raw device**
  Smarter media now natively understands a key-value I/O interface with lock-free updates, crash recovery, and no additional metadata overhead.

- **Up to 3x capacity increase**
  Dramatically reduces over-provisioning with coordinated garbage collection and automated key expiry.

- **3x throughput on same SSD**
  Early benchmarks comparing against memcached with BerkeleyDB persistence show up to 3x improvement.

Key-Value Store API library Benchmarks

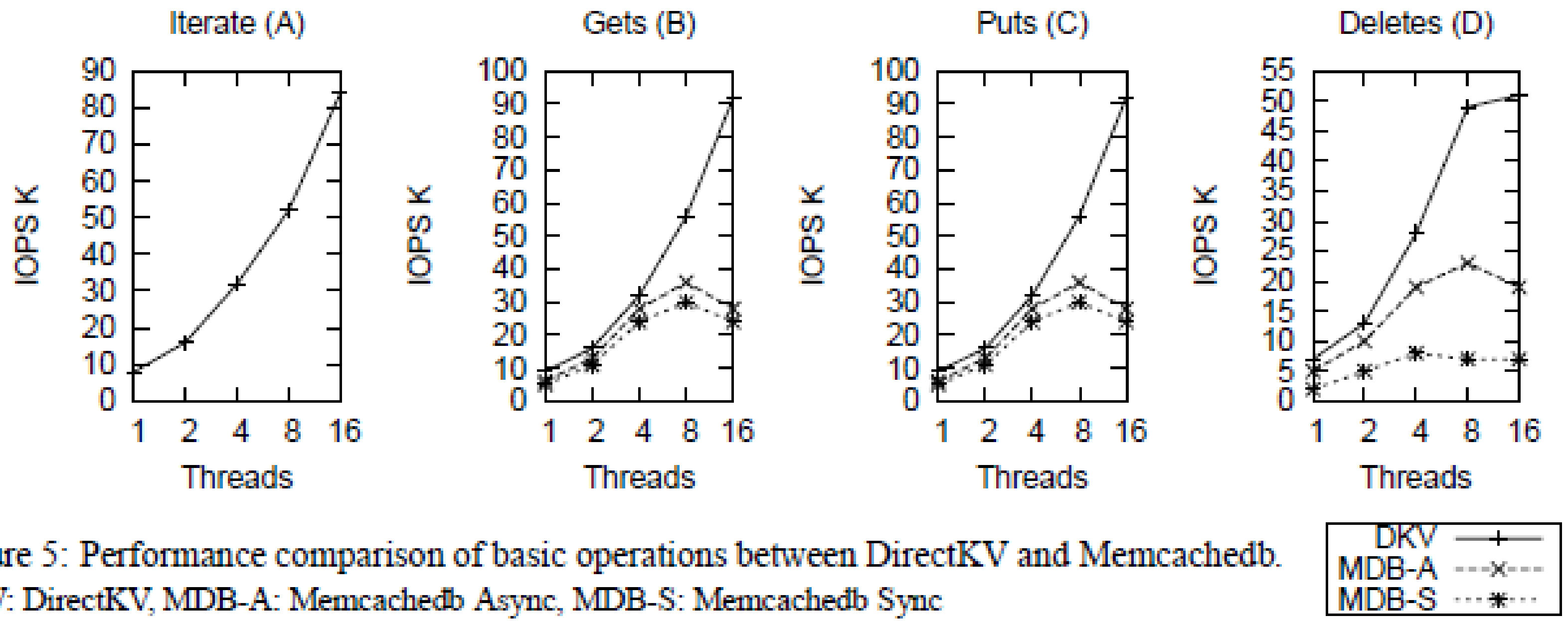


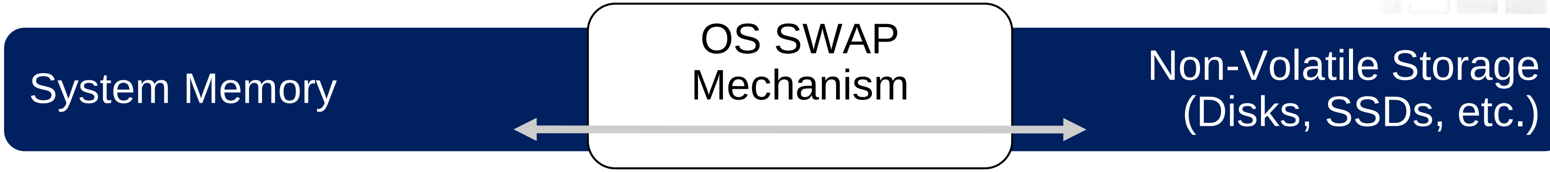
Figure 5: Performance comparison of basic operations between DirectKV and Memcachedb.
DKV: DirectKV, MDB-A: Memcachedb Async, MDB-S: Memcachedb Sync

Range of memory-Access Semantics

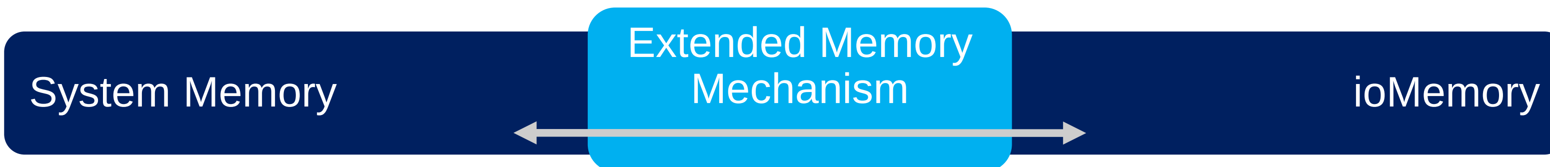
Extended Memory	**Volatile**	Transparently extends DRAM onto flash, extending application virtual memory
Checkpointed Memory	Volatile with non-volatile checkpoints	Region of application virtual memory with ability to preserve snapshots to flash namespace
Auto-Commit Memory™	**Non-volatile**	Region of application memory automatically persisted to non-volatile memory and recoverable post-system failure

OS Swap vs. Extended Memory

FUSION-IO



Originally designed as a last resort to prevent OOM (out-of-memory) failures
Never tuned for high-performance demand-paging
Never tuned for multi-threaded apps
Poor performance, ex. < 30 MB/sec throughput



No application code changes required
Designed to migrate hot pages to DRAM and cold pages to ioMemory
Tuned to run natively on flash (leverages native characteristics)
Tuned for multi-threaded apps
10-15x throughput improvement over standard OS Swap

Comparing I/O and Memory Access Semantics

FUSION-iO

I/O	I/O semantics examples: • Open file descriptor – open(), read(), write(), seek(), close() • (New) Write multiple data blocks atomically, nvm_vectored_write() • (New) Open key-value store – nvm_kv_open(), kv_put(), kv_get(), kv_batch_*()
Memory Access **(Volatile)**	Volatile memory semantics example: • Allocate virtual memory, e.g. malloc() • memcpy/pointer dereference writes (or reads) to memory address • (Improved) Page-faulting transparently loads data from NVM into memory
Memory Access **(Non-Volatile)**	Non-volatile memory semantics example: • (New) Allocate and map Auto-Commit Memory™ (ACM) virtual memory pages • memcpy/pointer dereference writes (or reads) to memory address • (New) Call checkpoint() to create application-consistent ACM page snapshots • (New) After system failure, remap ACM snapshot pages to recover memory state • (New) De-stage completed ACM pages to NVM namespace • (New) Remap and access ACM pages from NVM namespace at any time

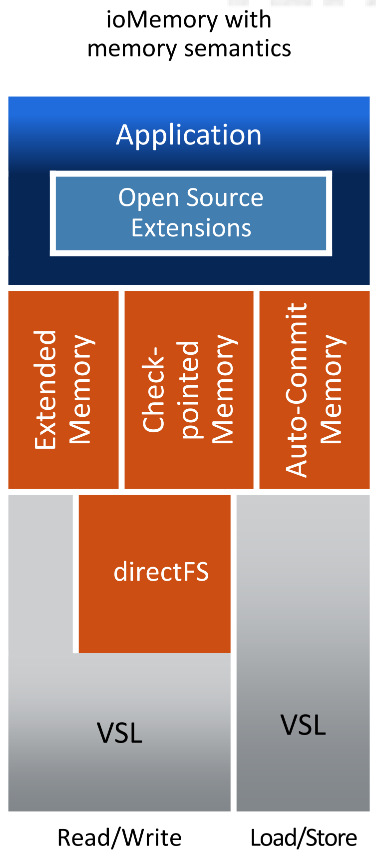
Application Use of Memory-Access Semantics

1. Application source uses memory programming semantics, such as malloc(), free(), pointer ops, etc.

2. Stack can exhibit different properties ranging from purely volatile (DRAM extension), to intermediate points of persistence (checkpoints), to fine grained persistence (ACM)

3. Underlying technology can be block oriented or support direct CPU load/store operations

4. Integrates with existing storage namespaces



Open Interfaces and Open Source

- Primitives: Open Interface
- directFS: Open Source
- API Libraries: Open Source, Open Interface
- Application modifications: Open Source
- INCITS SCSI (T10) active standards proposals:
  - SBC-4 SPC-5 Atomic-Write
    http://www.t10.org/cgi-bin/ac.pl?t=d&f=11-229r6.pdf
  - SBC-4 SPC-5 Scattered writes, optionally atomic
    http://www.t10.org/cgi-bin/ac.pl?t=d&f=12-086r3.pdf
  - SBC-4 SPC-5 Gathered reads, optionally atomic
    http://www.t10.org/cgi-bin/ac.pl?t=d&f=12-087r3.pdf
- SNIA NVM-Programming TWG active member

QUESTIONS?

fusionio.com | REDEFINE WHAT'S POSSIBLE

THANK YOU

fusionio.com | REDEFINE WHAT’S POSSIBLE