Beyond Block I/O

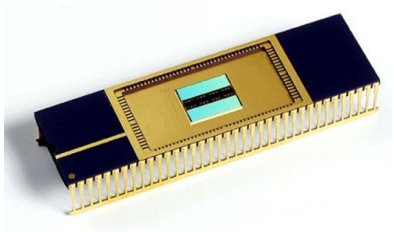
Exposing Native FTL Capabilities

David Nellans

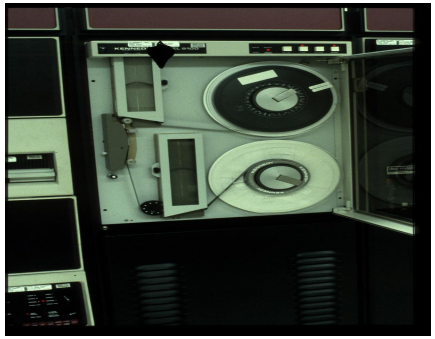
dnellans@fusionio.com

What is NVM?

Storage?



Memory?



Volatile?

Interfaces Define Classifications

Storage

Block oriented
Persistent namespace
Physically addressed

Memory

Word oriented
Volatile namespace
Virtually addressed

Random Access

Data Persistence

Interfaces Define Classifications

Storage

Block oriented
Persistent namespace
Physically addressed

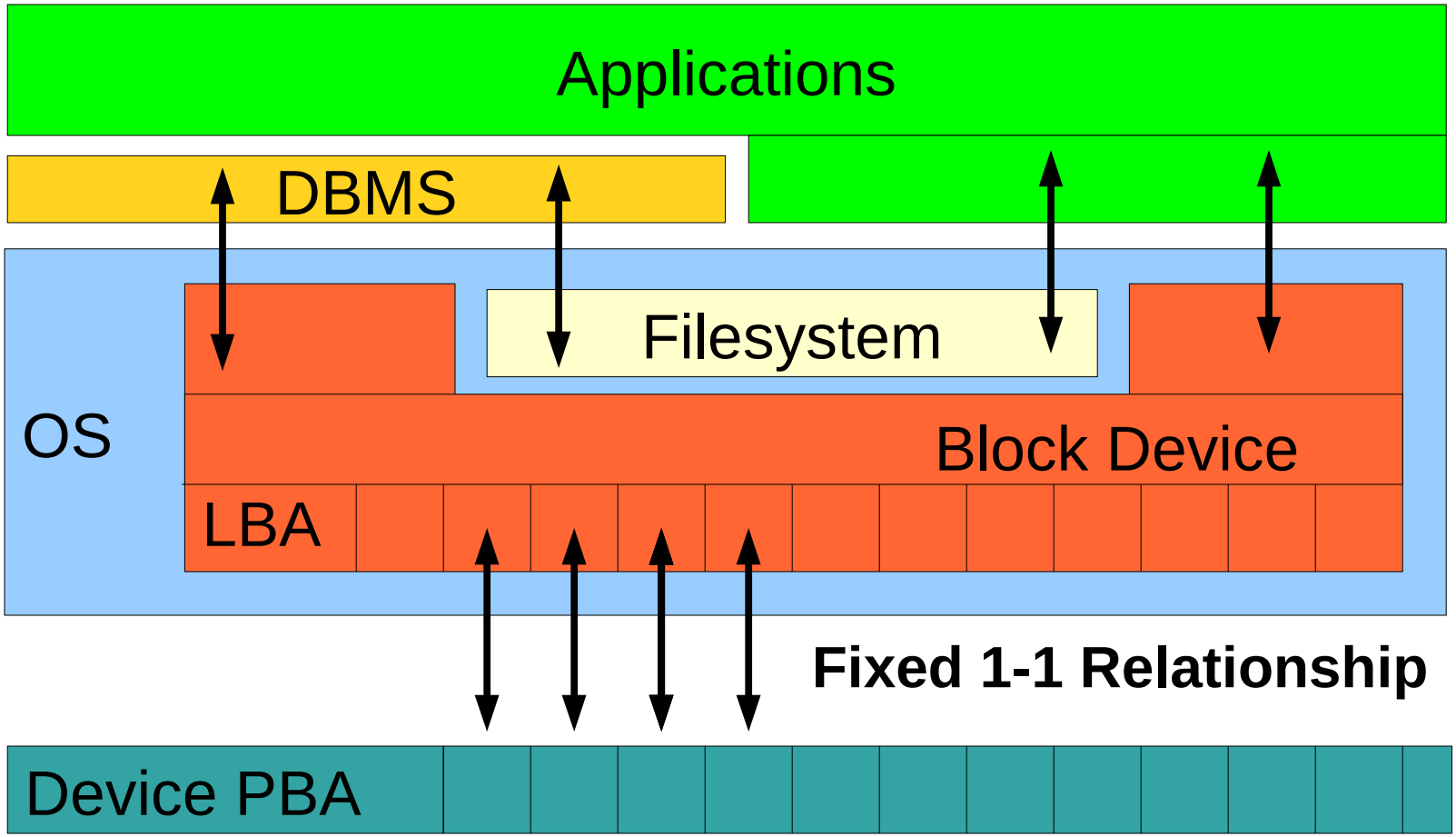
Memory

Word oriented
Volatile namespace
Virtually addressed

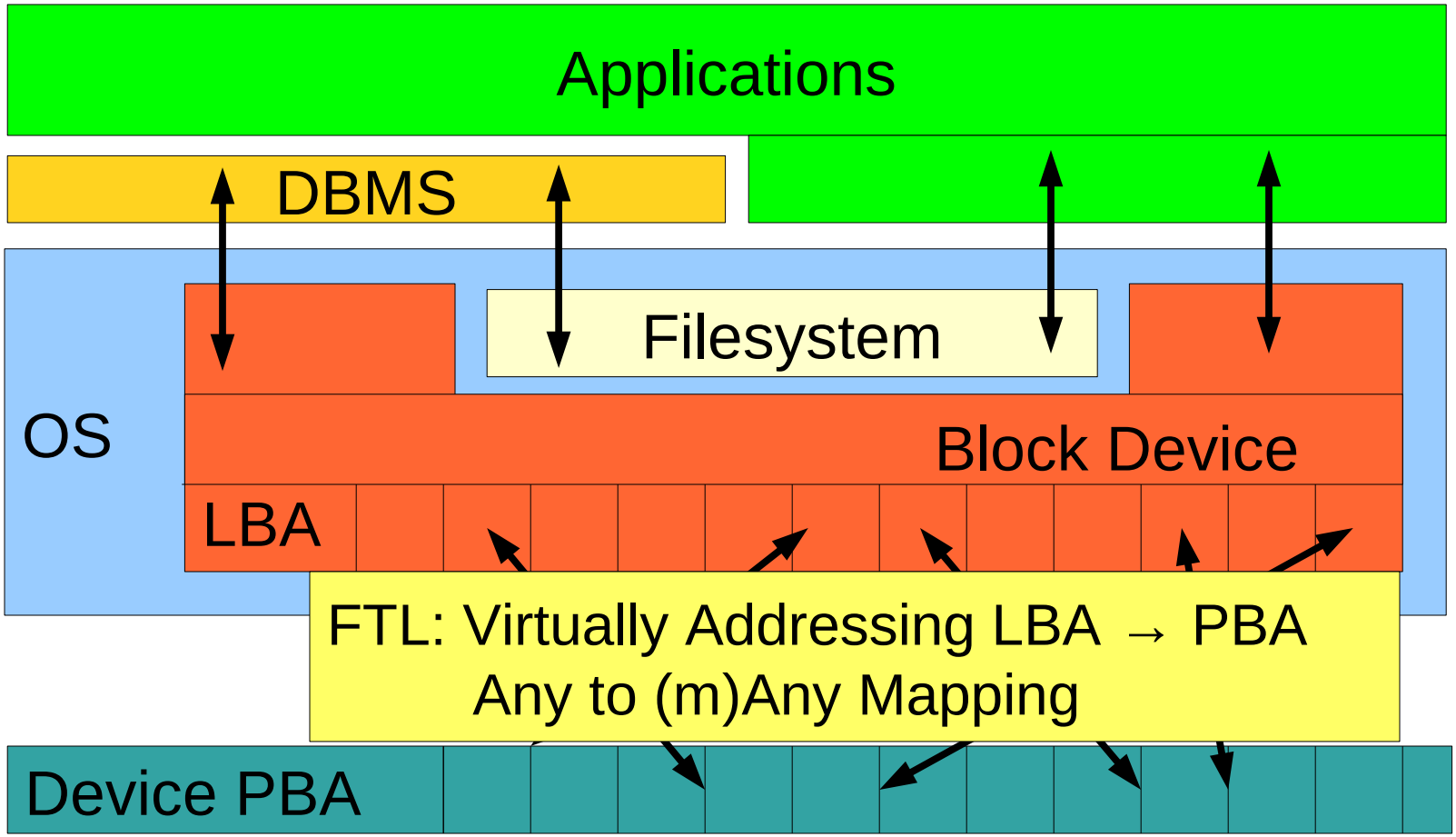
Random Access

Data Persistence

Traditional Storage Stack



Storage Stack Using FTL



Why Virtually Addressed Storage?

Industry hides virtual addressing behind FTL

- Maintain block storage interface
- Hides limited cell endurance issues
- Hides asymmetric performance characteristics

Creates multiple redundant mapping layers

Application:	ALB → FLB
Filesystem:	FLB → LBA
FTL:	LBA → PBA

Why not collapse Tracking?

Virtually Addressed Storage Primitives

trim(LBA)

- Hint to FTL to unmap LBA->PBA
- Improves wear leveling
- Improves write performance

- Analogous to free() in virtual memory system?

Virtually Addressed Storage Primitive

trim(LBA)

• No data DMA required	Addresses only
• Write after Trim (WaT)	Well specified behavior
• Read after Trim (RaT)	Return zeros? Return old data? Return new data? Powercut situations?

New VA Storage Primitive

persistent_trim(LBA)

• No data DMA required	Addresses only
• Write after Trim (WaT)	Well specified behavior
• Read after Trim (RaT)	Must return zero! Must survive powercut! Requires metadata write Slow?

Proposed VA Storage Primitives

read(LBA)	Vallocation on first access
write(LBA)	Vallocation on first access
trim(LBA)	Hint for VA block deallocation
persistent_ trim(LBA)	Directive for VA block mapping
exists(LBA)	Query state of allocation

read/write() are allocating operations
ptrim() is deallocating operation
exists() is querying operation

Proposed VA Storage Primitives

read(LBA) – write(LBA) – trim (LBA)

persistent_ trim(LBA)	Directive for VA block mapping
exists(LBA)	Query state of allocation
atomic_write(LBA's)	Atomically write multiple LBAs

Virtual addressing inherently allows copy-on-write
FTL can support atomic vectored writes natively

More VA Storage Primitives

read(LBA) – write(LBA) – trim (LBA)

persistent_ trim(LBA)	Directive for VA block mapping
exists(LBA)	Query state of allocation
atomic_write(L, B, A)	Atomically write multiple LBAs
nameless_write(data)	Return optimal LBA range

Applications/FS may not care what LBA is
Return the LBA range optimal for the device

More VA Storage Primitives

read(LBA) – write(LBA) – trim (LBA)

persistent_ trim(LBA)	Directive for VA block mapping
exists(LBA)	Query state of allocation
atomic_write(LBA's)	Atomically write multiple LBAs
nameless_write(data)	Return optimal LBA range

Use-cases for these new primitives?

Atomic Write*

DBMS implicitly write data twice to maintain correctness

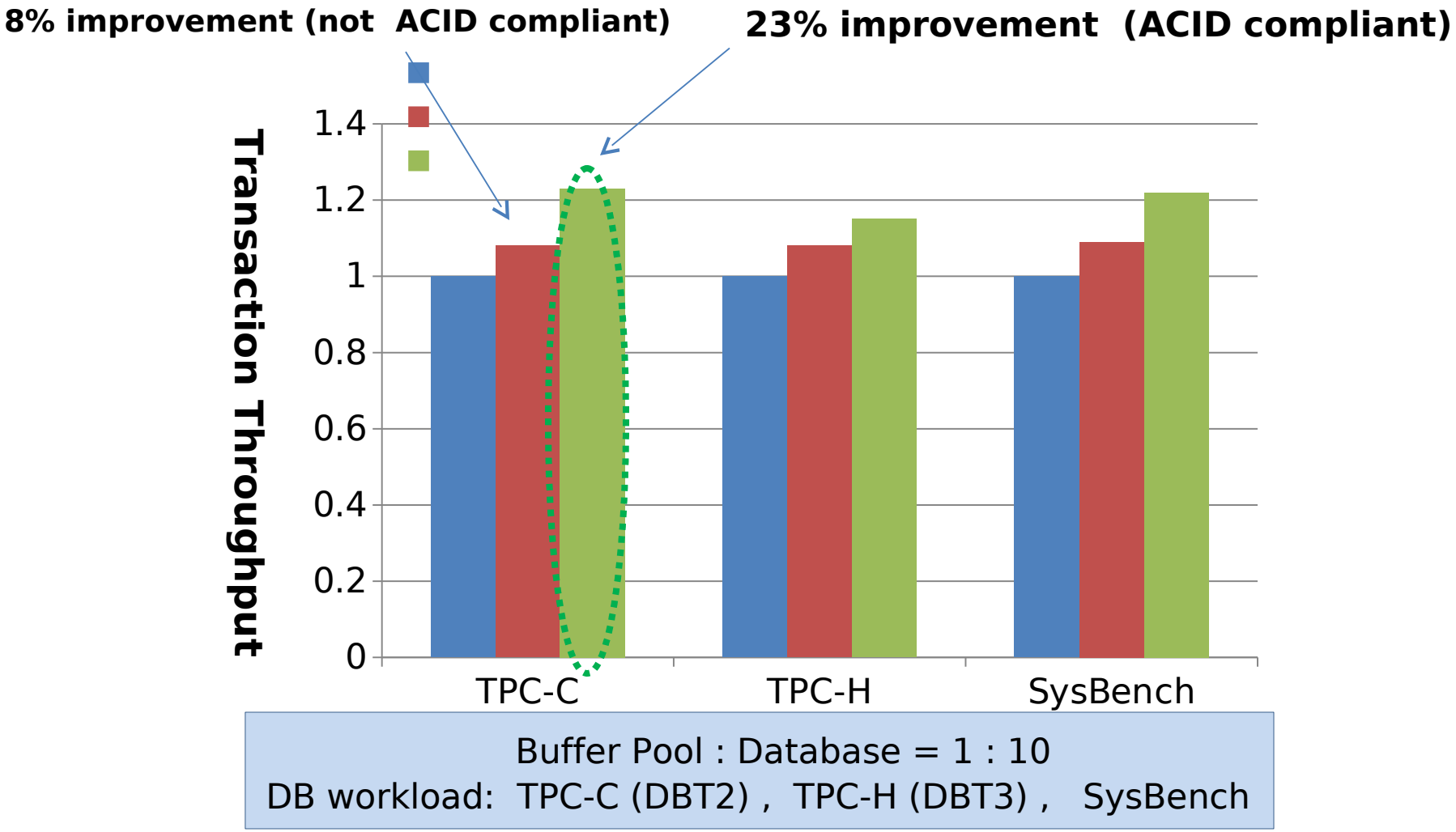
Results in 2x data written overhead to flash.

Often requires synchronous operations to disk.

Atomic-write allows databases to over-write in place but maintain ACID compliance.

*Ouyang and Nellans - HPCA 2011

Atomic Write* Performance



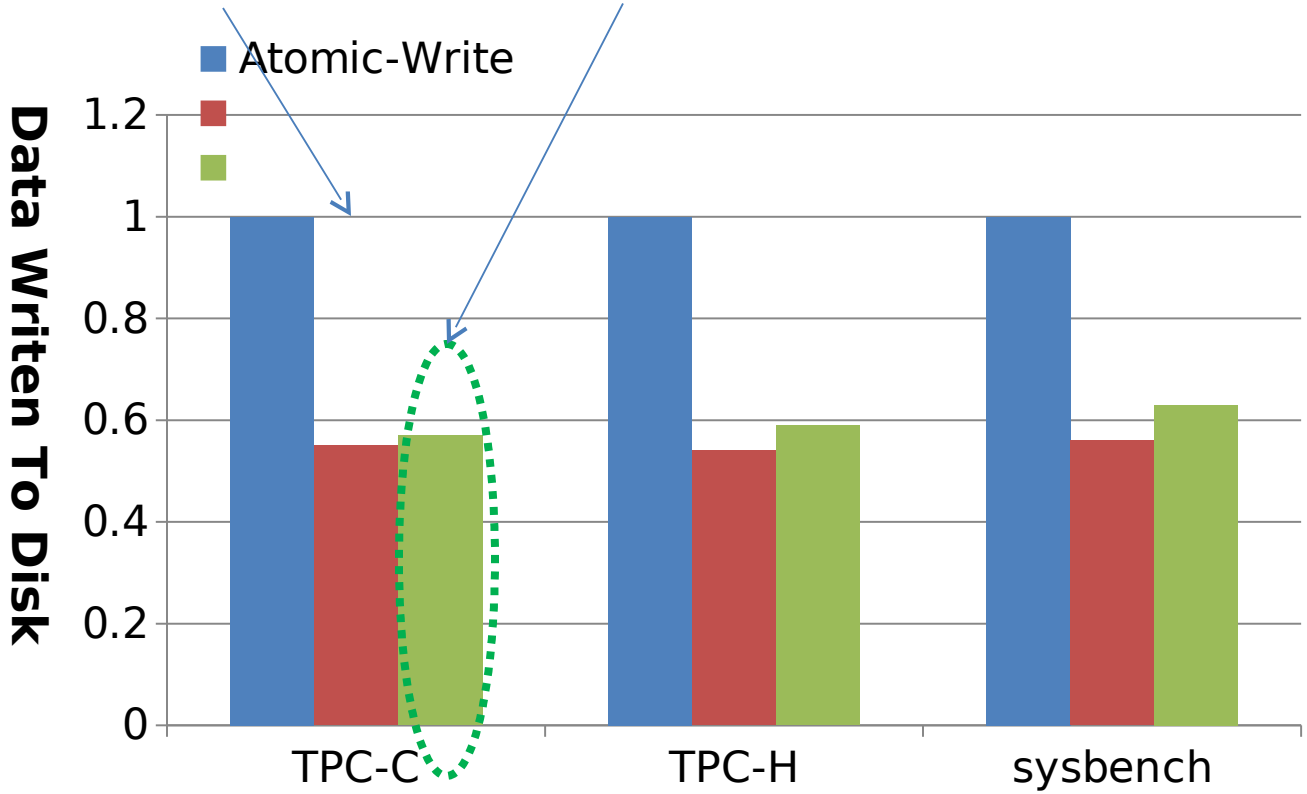
Buffer Pool : Database = 1 : 10
DB workload: TPC-C (DBT2) , TPC-H (DBT3) , SysBench

*Ouyang and Nellans - HPCA 2011

Atomic Write* Data Written

46% improvement (not ACID compliant)

43% reduction (ACID compliant)

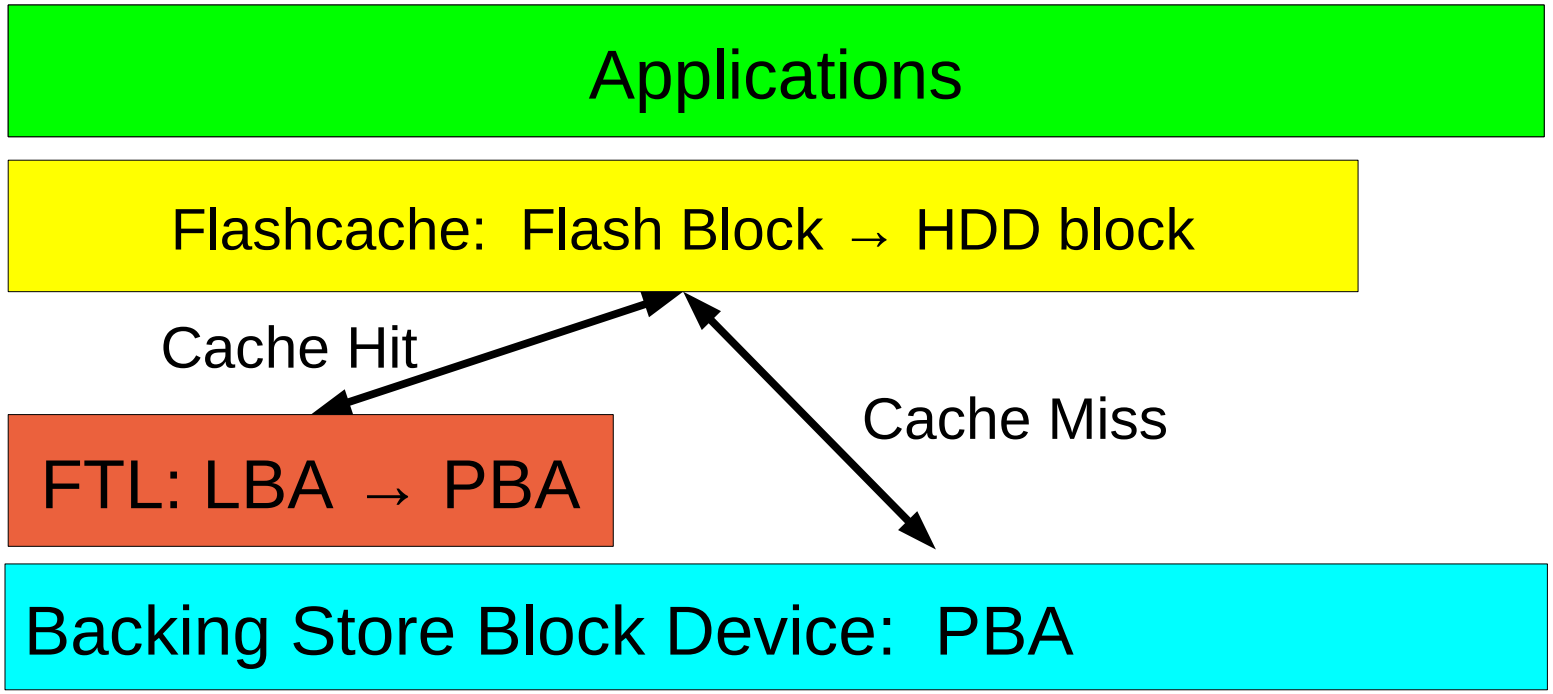


Buffer Pool : Database = 1 : 10
DB workload: TPC-C (DBT2) , TPC-H (DBT3) , SysBench

*Ouyang and Nellans - HPCA 2011

ptrim()* and exists()*

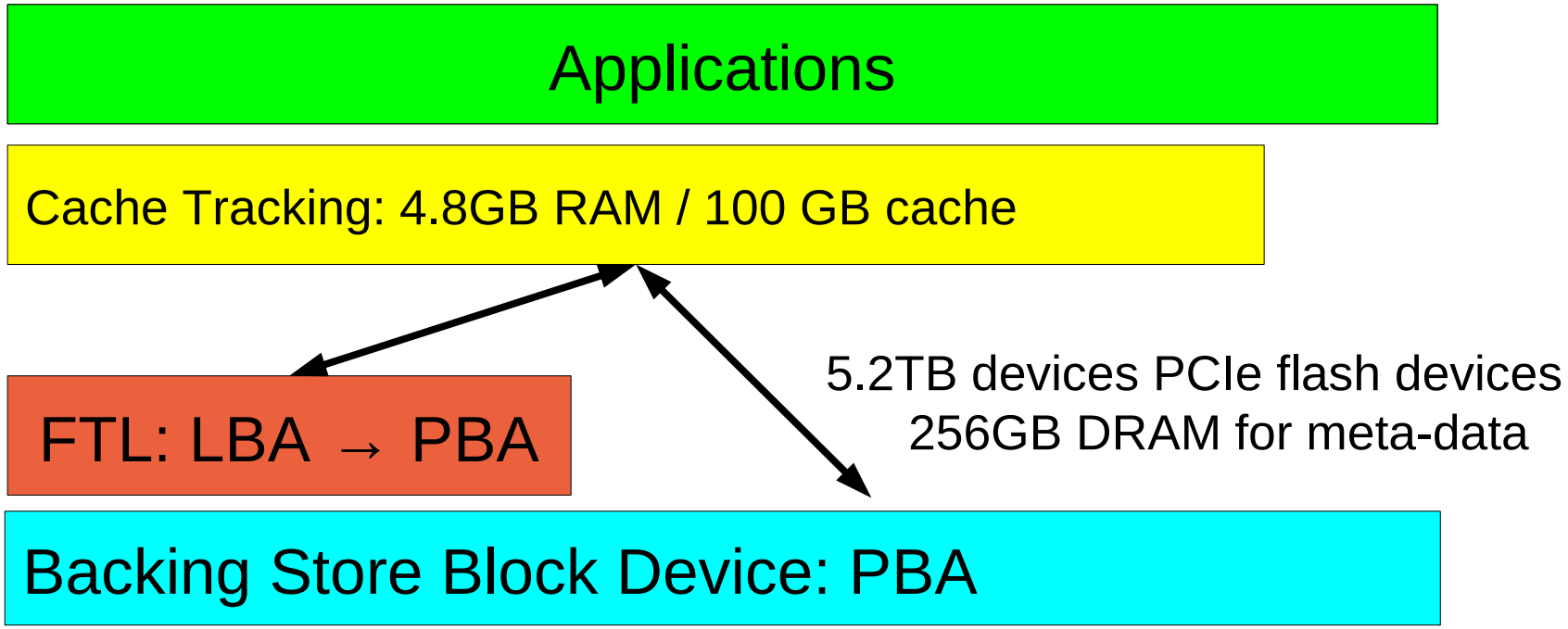
Reduce application mapping layers.
Example: Block caching onto flash from disk.



*Nellans and Zappe - NVMW 2011

ptrim()* and exists()*

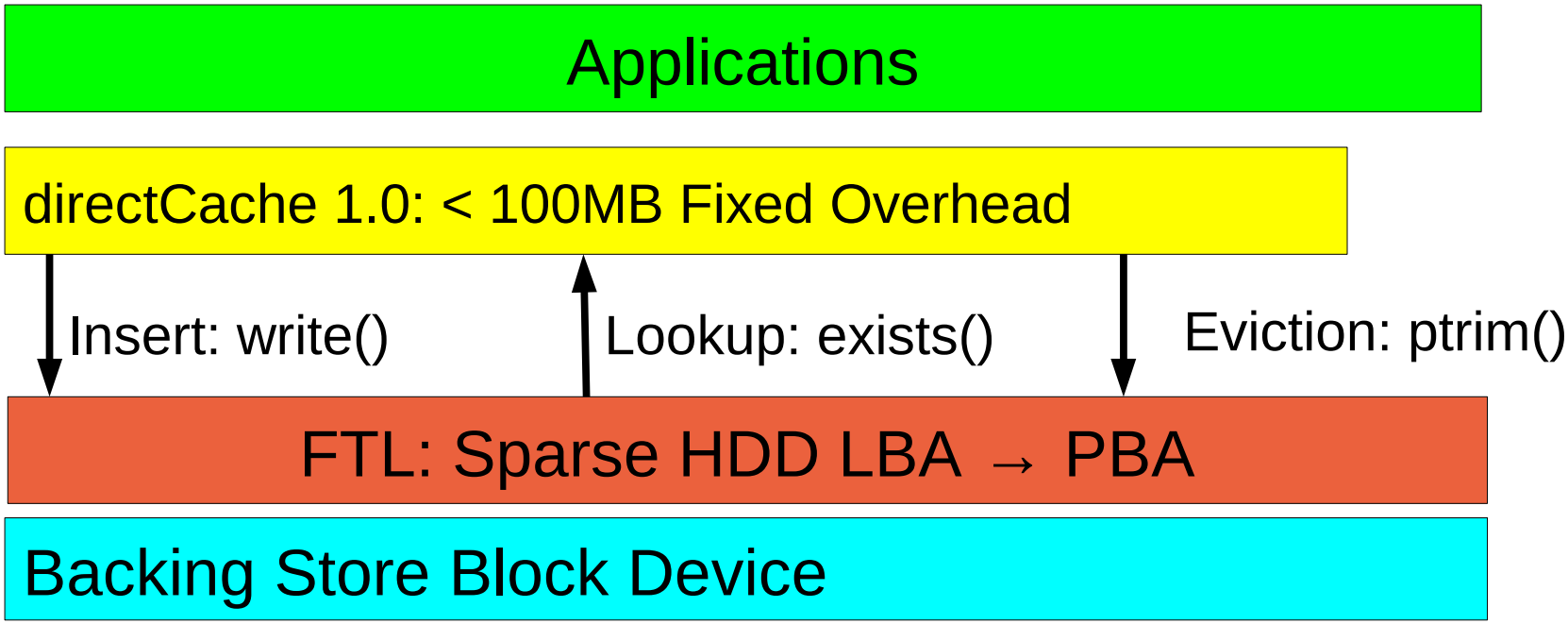
Reduce application mapping layers.
Example: Block caching onto flash from disk.



*Nellans and Zappe - NVMW 2011

ptrim()* and exists()*

Reduce application mapping layers.
Example: Block caching onto flash from disk.



*Nellans and Zappe - NVMW 2011

Virtually Address Storage Uses

Reduce application mapping layers.

Eliminate CoW for transactional systems.

Optimize flash device performance and wear-out.

Step towards semantics for Storage Class Memory?

Conclusions

Storage is now virtually addressed, embrace it!

Backwards compatible with block interface

Work up the stack

- Applications have worked around block storage
- Provide new primitives to applications
- Leverage inherent properties of FTL for efficiency

Questions and Comments

Thank you!

David Nellans

dnellans@fusionio.com