Beyond Block I/O: Rethinking Traditional Storage Primitives

Xiangyong Ouyang[*] [†], David Nellans [†], Robert Wipfel [†],
David Flynn [†], D. K. Panda[*]

[*] **The Ohio State University** [†]**Fusion-io**

Agenda

- Introduction and Motivation
  - Solid State Storage (SSS) Characteristics
  - Duplicated efforts at SSS and upper layers
- Atomic-Write Primitive within FTL
- Leverage Atomic-Write in DBMS
  - Example with MySQL
- Experimental Results
- Conclusion and Future Work

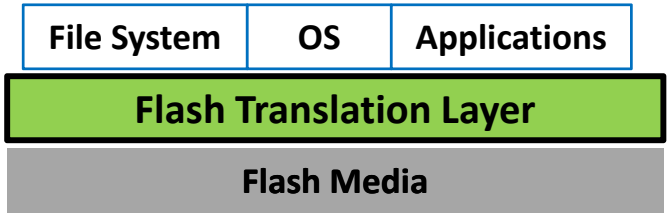
Evolution of Storage Devices

- Interface to persistent storage remains unchanged for decades
  - seek, read, write
  - Fits well with mechanical hard disks

- Solid State Storage (SSS)
  - ✓ Merits
    - Fast random access, high throughput
    - Low power consumption
    - Shock resistance, small form factor
  - Expose the same disk-based block I/O interface
  - Challenges…

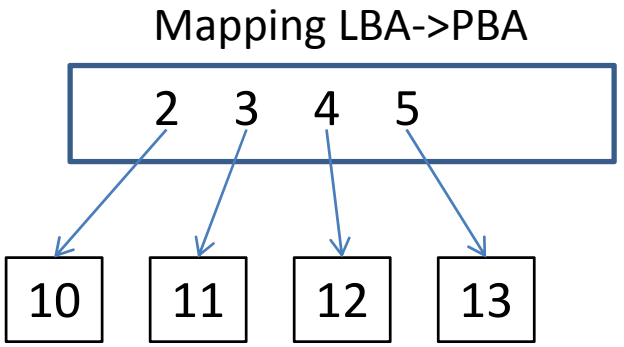
NAND-flash Based Solid State Storage (SSS)

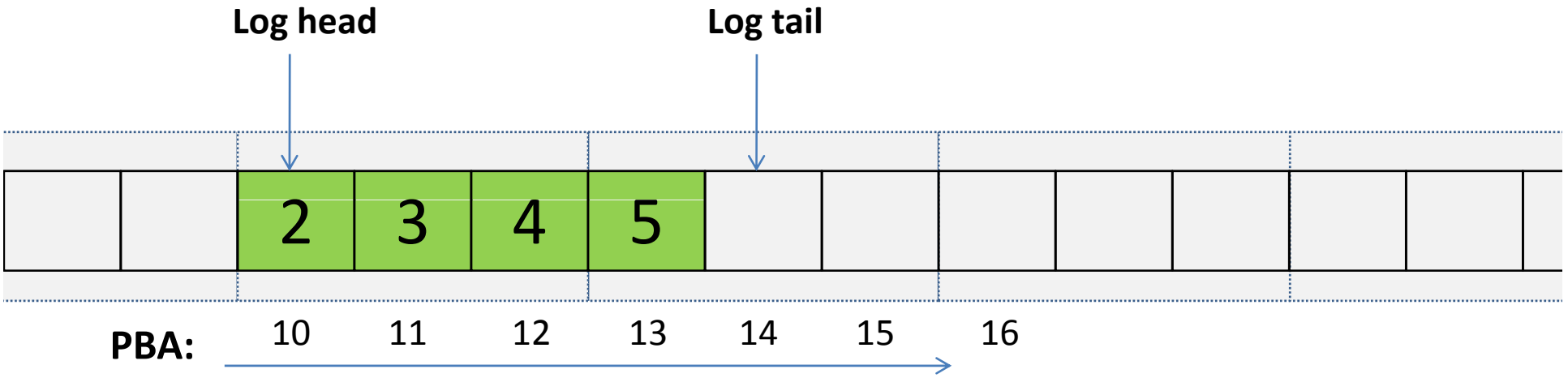
- Pitfalls
  - ✘ Asymmetric read/write latency
    - Cannot overwrite before erasure
    - Erasure at large unit (64-256 pages), very slow (1+ ms)
  - ✘ Flash Wear-out: limited write durability
    - SLC: 30K erase/program cycles, MLC: 3K erase/program cycles



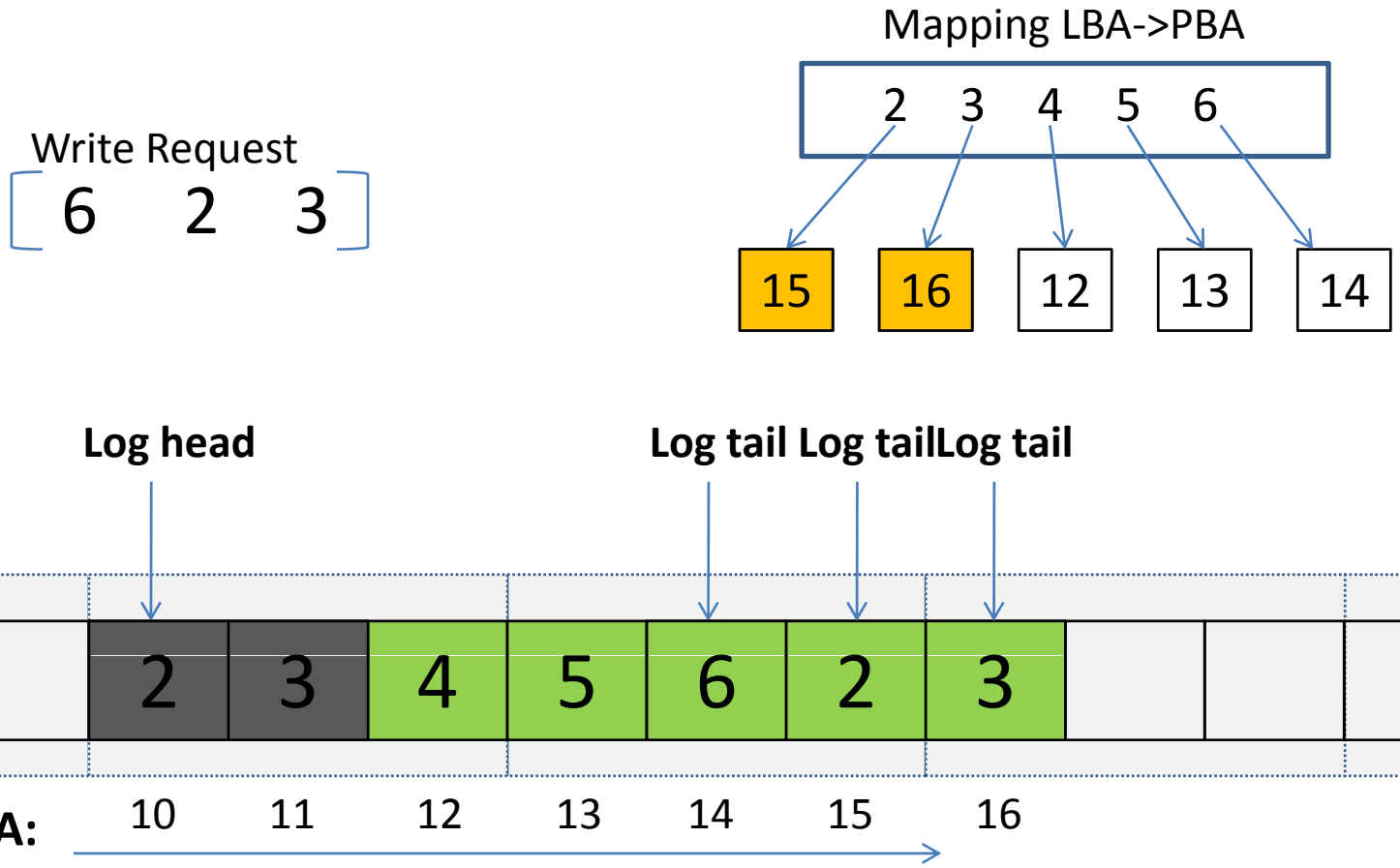
- Flash Translation Layer (FTL)
  - Input: **Logical Block Address (LBA)**
  - Output: **Physical Block Address (PBA)**

Log-Structured FTL





Log-Structured FTL



Log-FTL Advantages

- ✓ *Avoid in-place update (Block Remapping)*
- ✓ *Even wear-leveling*

Duplicated Efforts at Upper Layers and FTL

- ***Multi-Version*** at Upper Layer
  - DBMS (Transactional Log)
  - File-systems (Metadata journaling, Copy-on-Write)
  - To achieve **Write Atomicity**
    - ACID: Atomicity, Consistency, Isolation, durability

- ***Block-Remapping*** at FTL
  - Avoid **in-place update** in critical path

- **Common Thread: Multi-versions of same data**
- **Why duplicate this effort ?**

- Proposed approach:
  - **Offload Write-Atomicity guarantee to FTL**
  - **Provide Atomic-Write primitive to upper layers**

Agenda

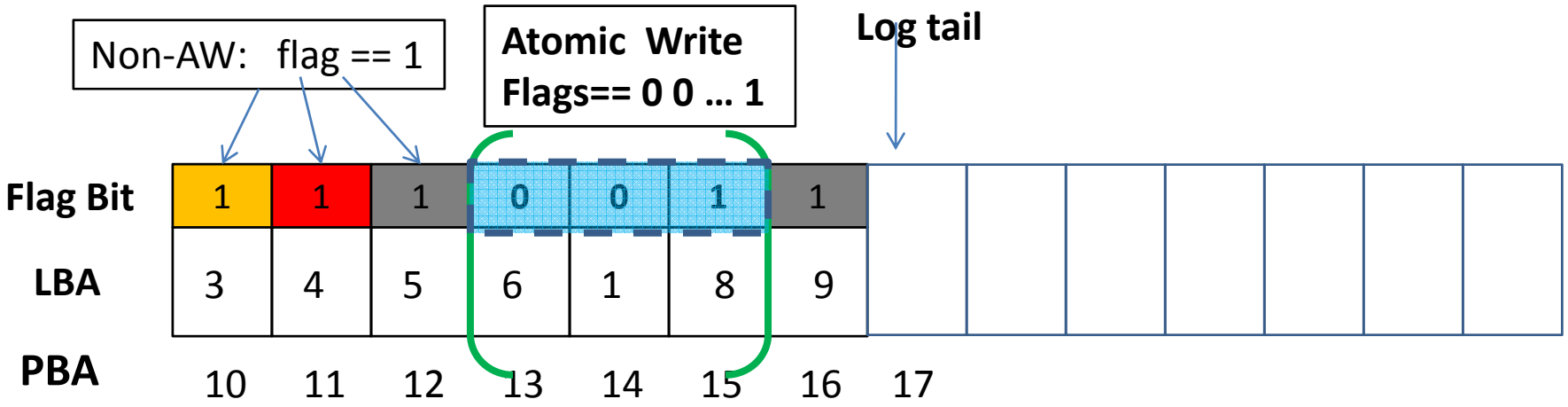
- Introduction and Motivation
- Atomic-Write Primitive at FTL
- Leverage Atomic-Write in DBMS
- Experimental Results
- Conclusion and Future Work

Atomic-Write: a New Block I/O Primitive

- Offload the Write-Atomicity guarantee into FTL
- Combines multi-block writes into a logical group (contiguous , non-contiguous)
- Commit the group as an atomic unit, if the compound operation succeeds
- Rollback the whole group is any individual fails

Atomic-Write (1): Flag Bit in Block Header

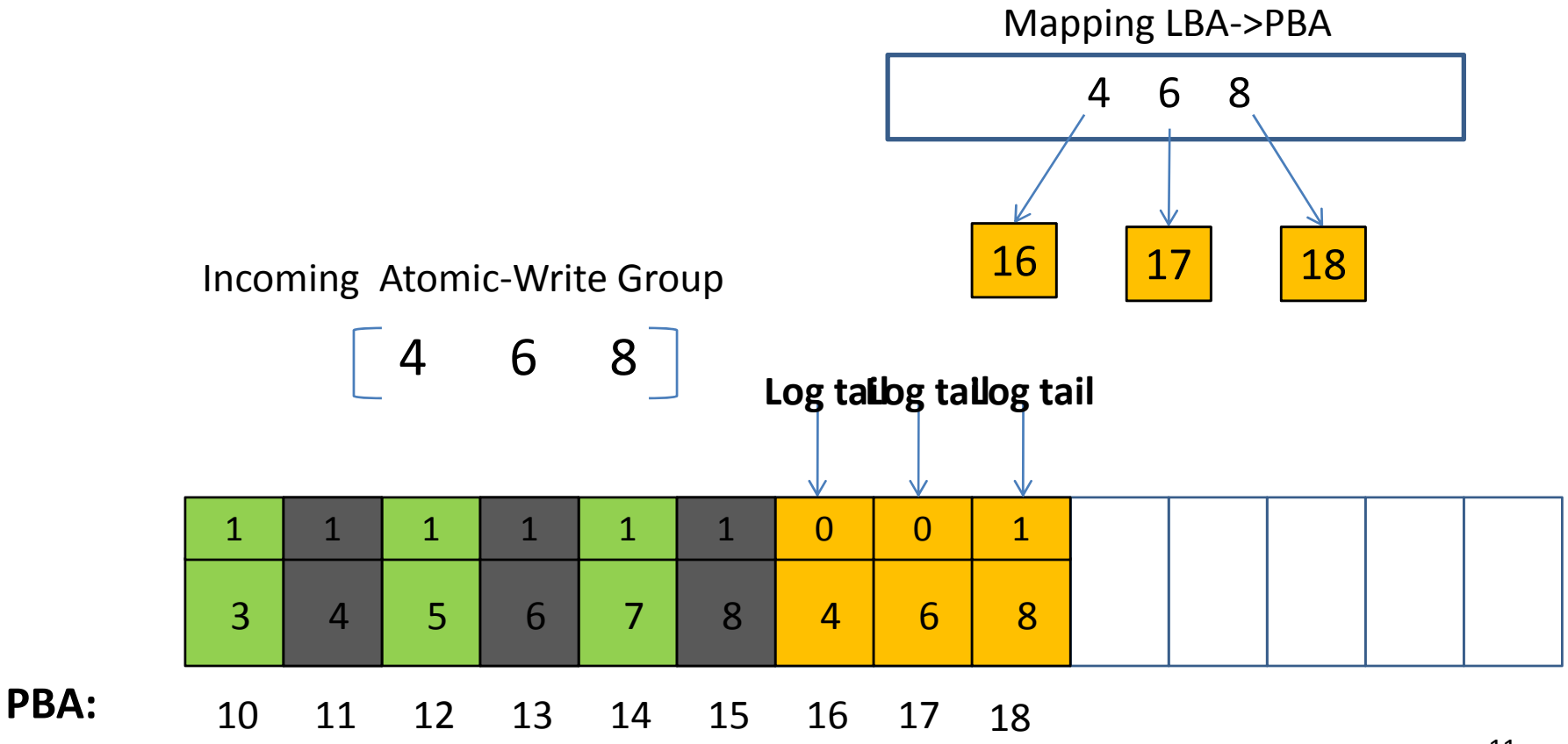
- One Flag Bit per block header
  - ➢ Identify blocks belonging to the same atomic-group



- Don't allow Non-AW to interleave with Atomic-Write

Atomic-Write (2): Deferred Mapping Table Update

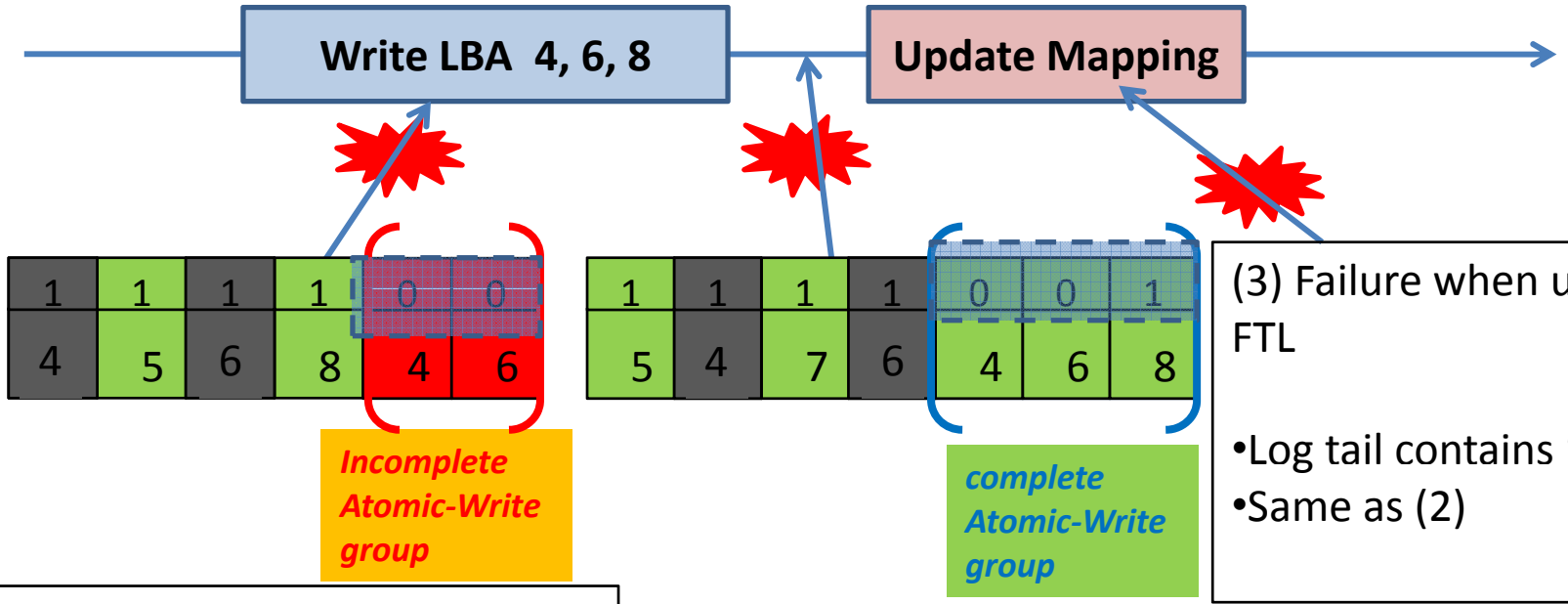
- Defer mapping table update
  - ➢Not expose partial state to readers



Atomic-Write (3): Failure Recovery

Atomic-Write Group

[4 6 8]



(1) Failure during writing:

•Scan backwards, discard blocks with "0" flag bits
•Rollback the partial blocks to previous version

(2) Failure after writing

•Scan the log from beginning, rebuild the FTL mapping

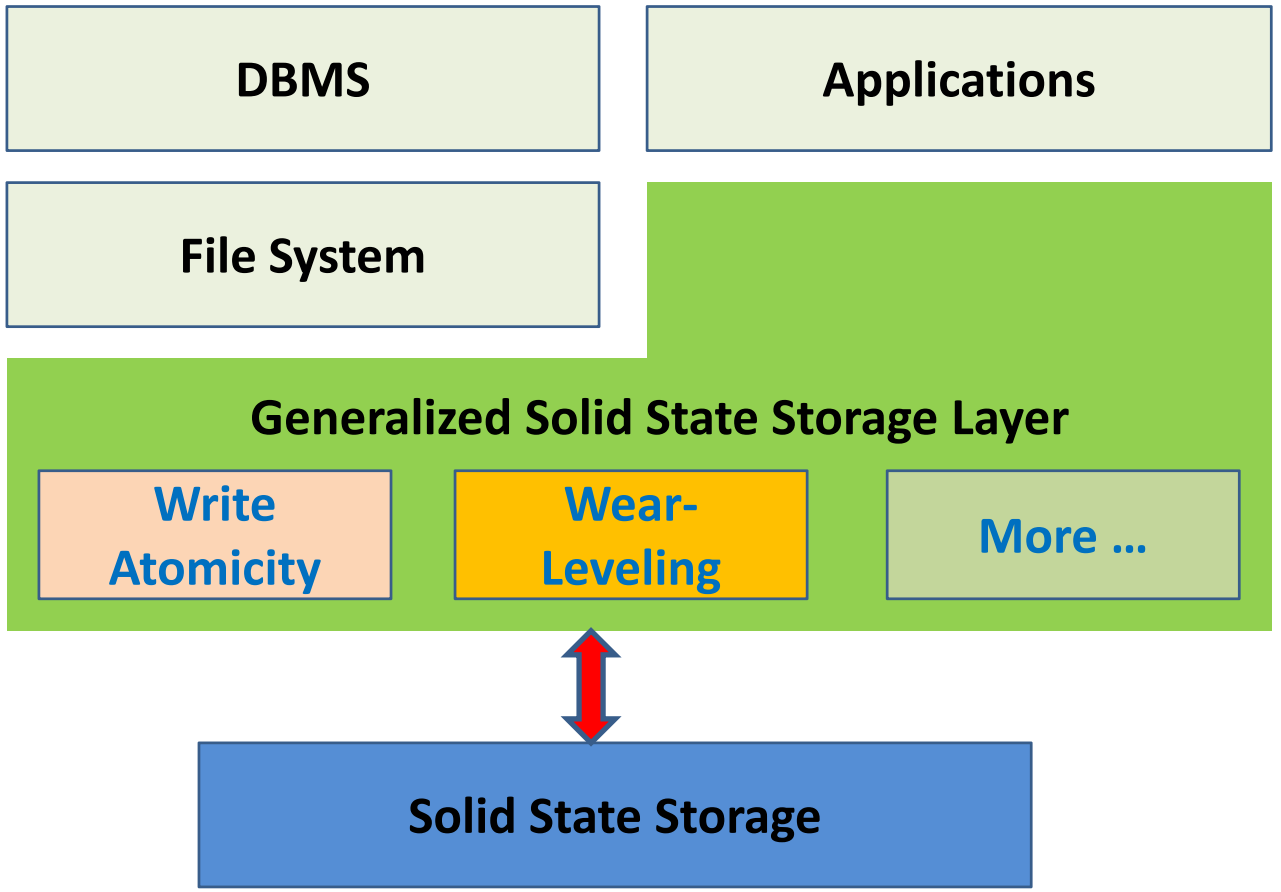
(3) Failure when updating FTL

•Log tail contains "1" flag bit
•Same as (2)

Agenda

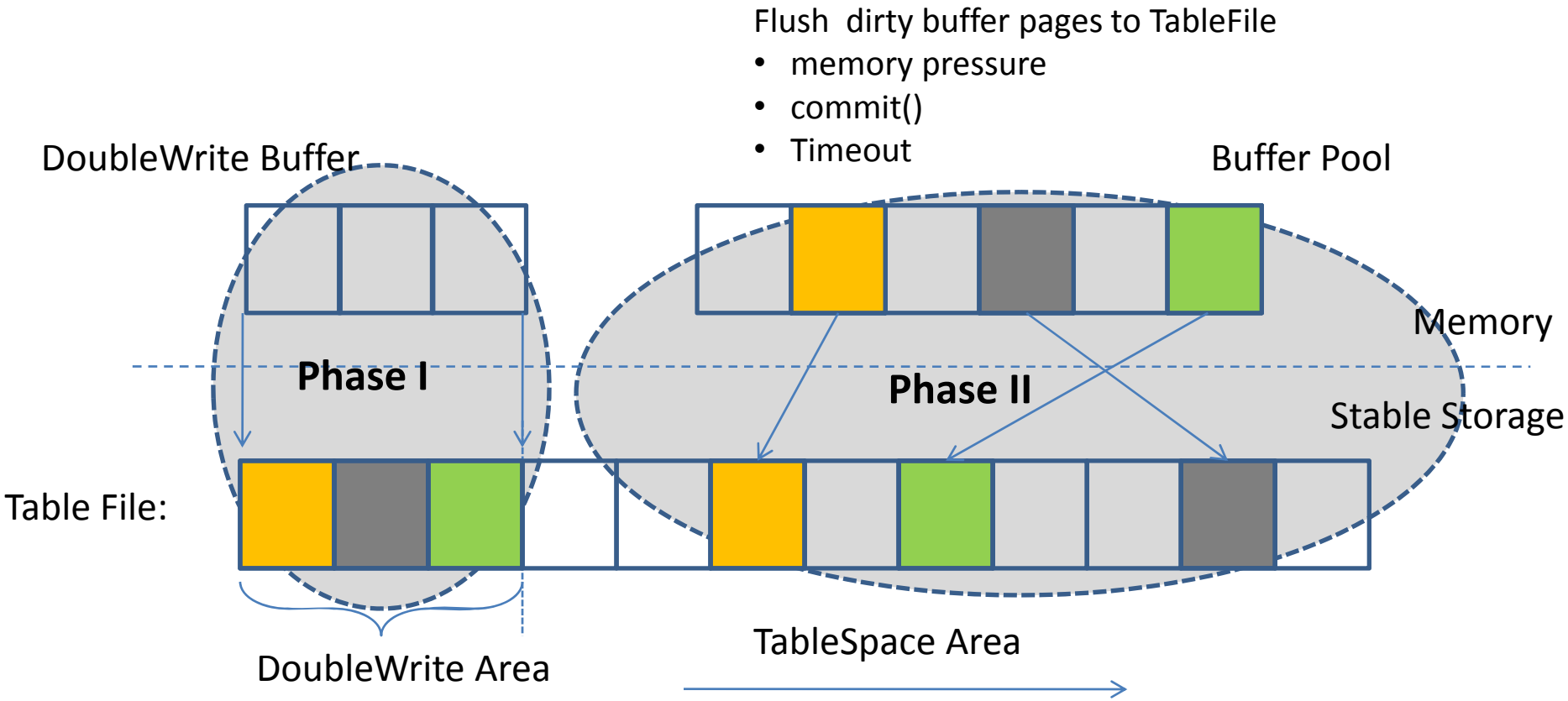
- Introduction and Motivation
- Atomic-Write Primitive at FTL
- Leverage Atomic-Write in DBMS
  - Example with MySQL
- Experimental Results
- Conclusion and Future Work

Proposed Storage Stack



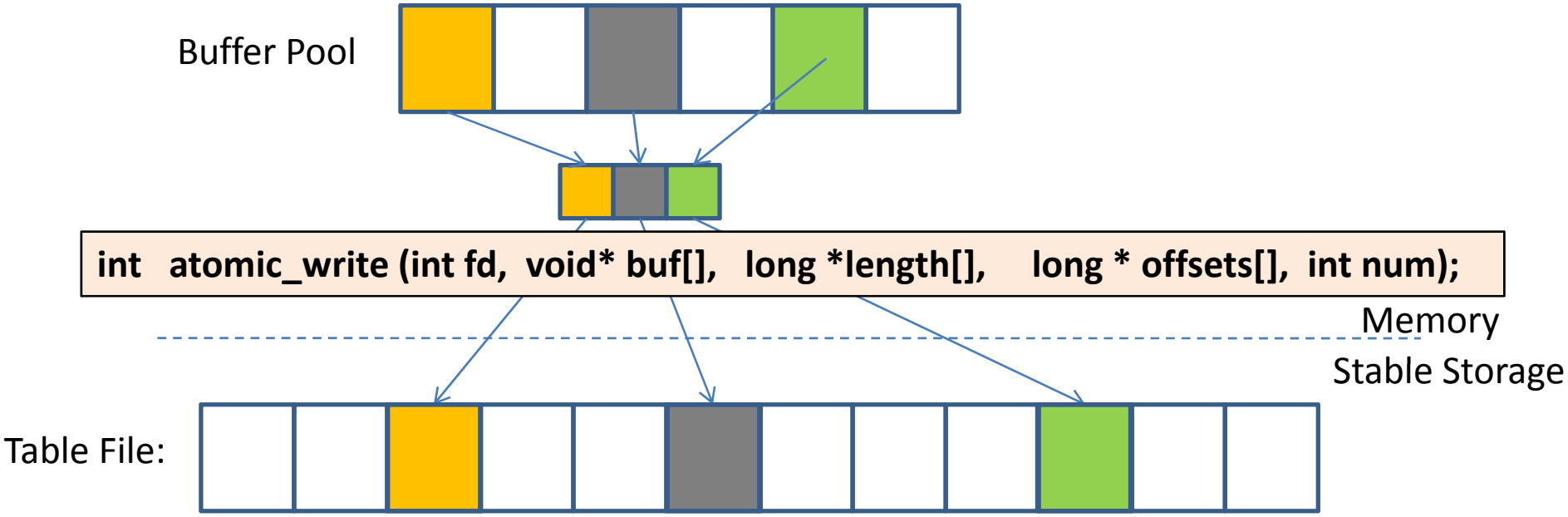
➔**Example: Leverage Atomic-Write in DBMS (MySQL)**

DoubleWrite with MySQL InnoDB Storage Engine



- **Every data page is written twice !**
  - **Impact the performance**
  - ***Double amount of writes to Flash media halve device's lifespan***

MySQL InnoDB: Atomic-Write



- ✓**Reduce the data written by half**
  Double the effective wear-out life
- ✓**Simplify the upper layer design**
- ✓**Better performance**
- ✓**Guarantee the same level of data integrity as DoubleWrite**

Agenda

- Introduction and Motivation
- Atomic-Write Primitive at FTL
- Leverage Atomic-Write in DBMS
- Experimental Results
- Conclusion and Future Work

Experiment Setup

- Fusion-io 320GB MLC NAND-flash based device
- Atomic-Write implemented in a research branch of v2.1 Fusion-io driver
- MySQL 5.1.49 InnoDB (extended with Atomic-Write)
  - 2 machines connected with 1 GigE
  - Both Trans. log and table-file stored on solid state

Processor	Xeon X3210 @ 2.13GHz
DRAM	8GB DDR2 667MHz, 4X2GB
Boot Device	250GB SATA-II 3.0Gb/s
DB Storage Device	Fusion-io ioDrive 320GB PCIe 1.0 4x Lanes
OS	Ubuntu 9.10 , Linux Kernel 2.6.33

Micro Benchmark

- Different Write Mechanisms:
  - ***Synchronous***: write() + fsync()
  - ***Asynchronous***: libaio
  - ***Atomic-Write***
- Different write patterns:
  - Sequential
  - Strided
  - Random
- Buffer strategies
  - Buffered_IO: OS page cache
  - Direct_IO: bypasses OS page cache

I/O Microbenchmark: Latency

Write Latency *(Lower is Better)*
(64 blocks, 512B each)

Write Pattern	Buffering	Latency (us)		
		Write Strategy		
		Sync	Async	A-Write
Random	Buffered	4042	1112	NA
	DirectIO	**3542**	**851**	**671**
Strided	Buffered	4006	1146	NA
	DirectIO	**3447**	**857**	**669**
Sequential	Buffered	3955	330	NA
	DirectIO	**3402**	**898**	**685**

- Atomic-Write : all blocks in one compound write
- Synchronous Write: write ( ) + fsync( )
- Asynchronous Write: Linux libaio

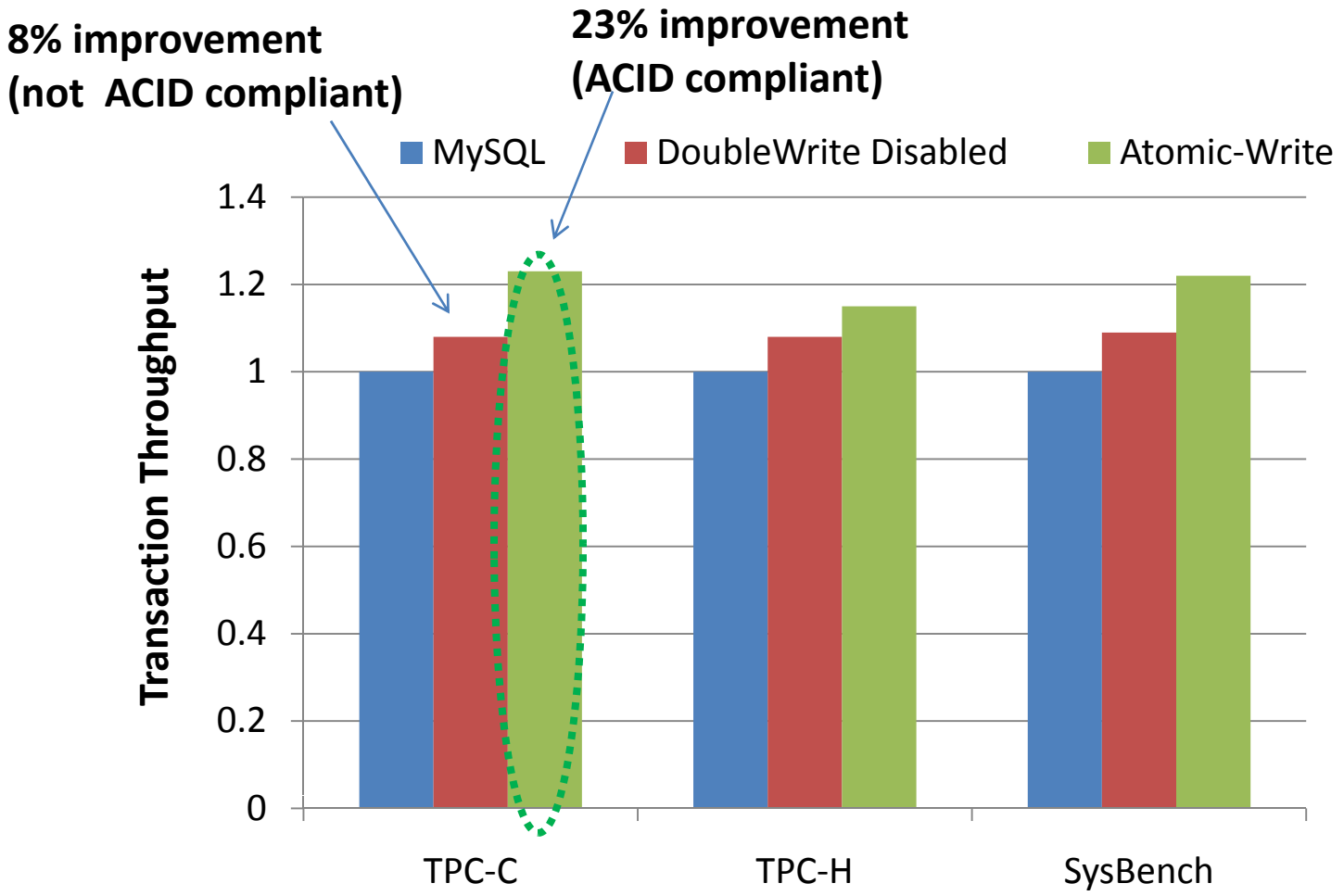
I/O Microbenchmark: Bandwidth

Write Bandwidth *(Higher is Better)*
(64 blocks, 16KB each)

		Bandwidth (MB/s)		
Write Pattern	Buffering	Write Strategies		
		Sync	Async	A-Write
Random	Buffered	302	301	NA
	DirectIO	**212**	**505**	**513**
Strided	Buffered	306	300	NA
	DirectIO	**217**	**503**	**513**
Sequential	Buffered	308	304	NA
	DirectIO	**213**	**507**	**514**

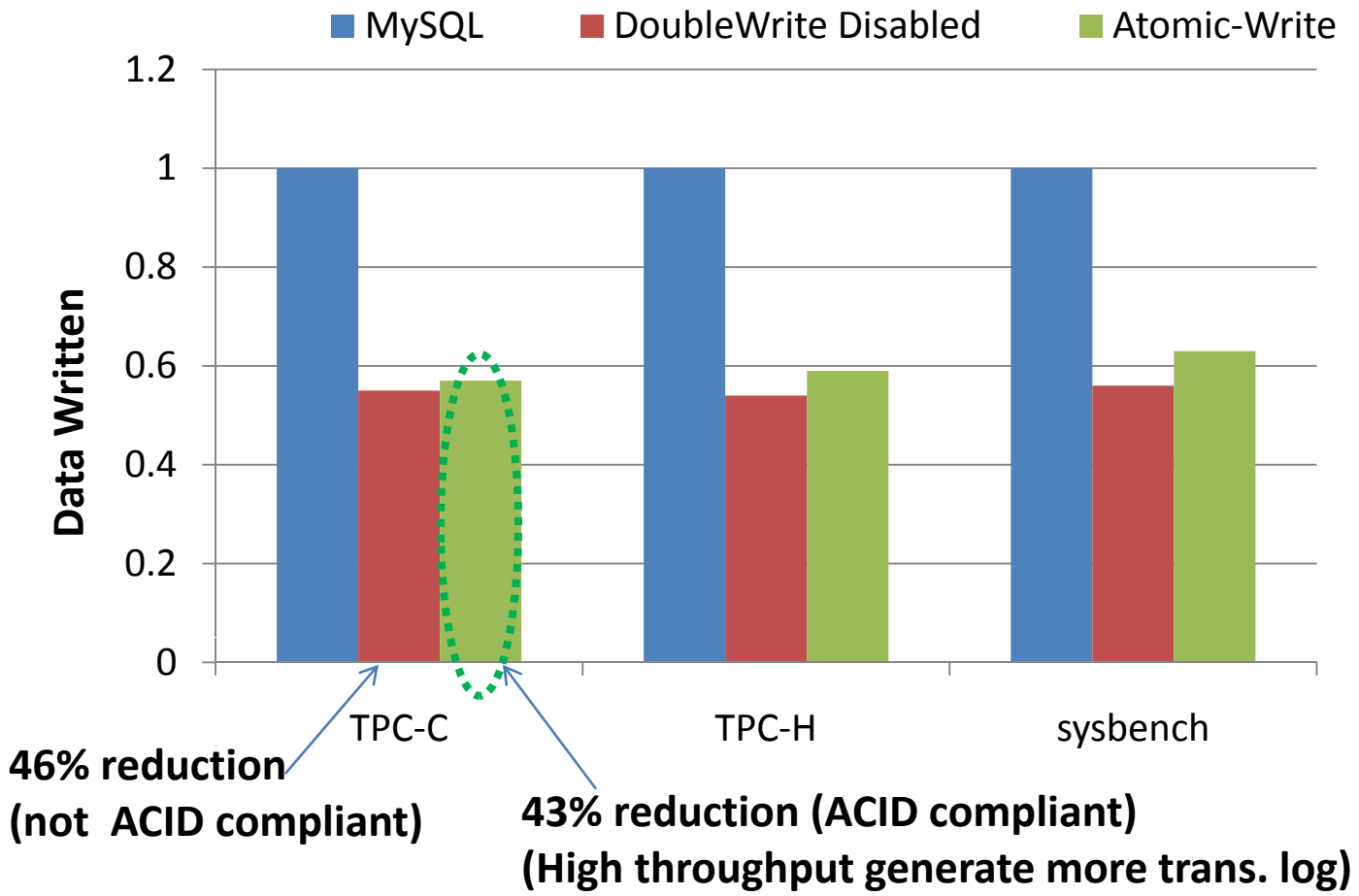
- Atomic-Write : all blocks in one compound write
- Synchronous Write: write ( ) + fsync( )
- Asynchronous Write: Linux libaio

Transaction Throughput



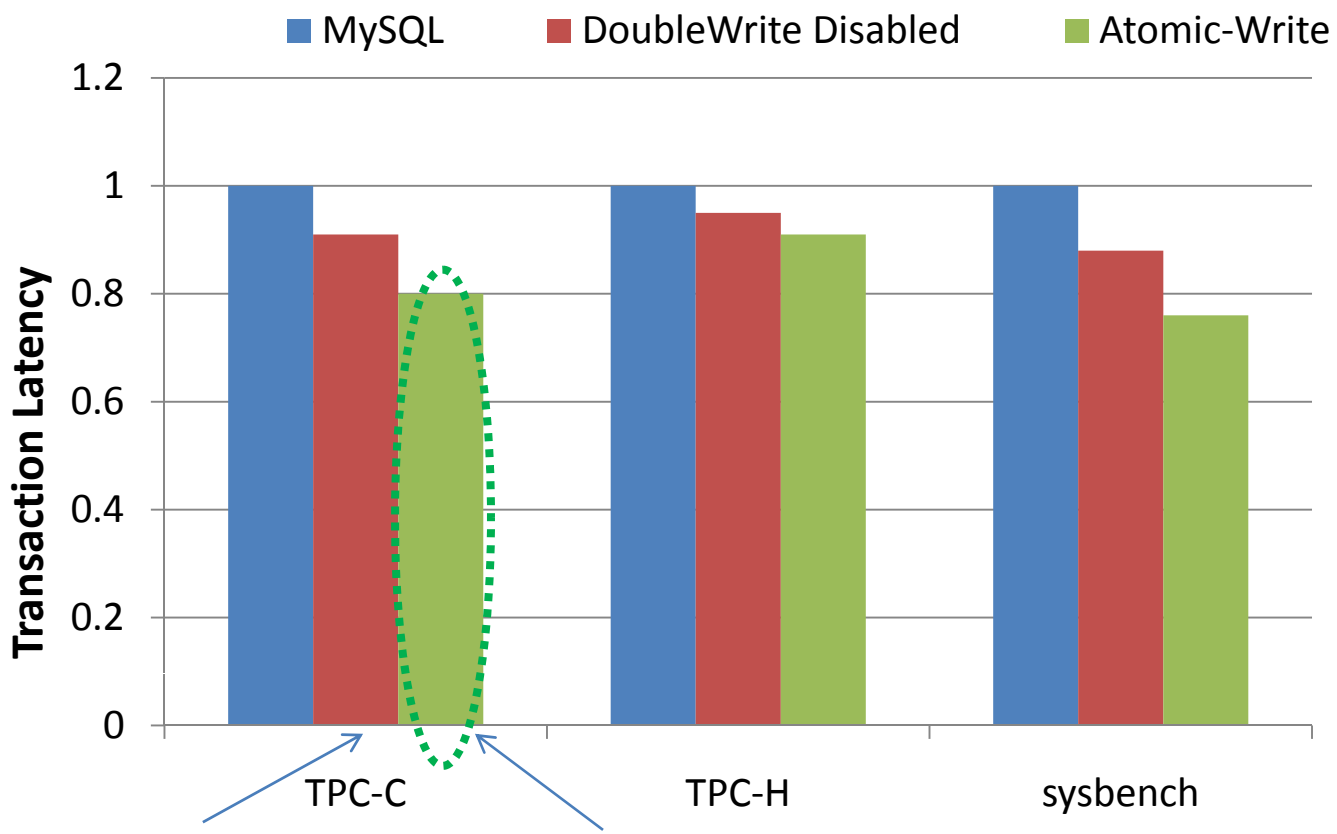
- Buffer Pool : Database = 1 : 10
- DB workload: TPC-C (DBT2) , TPC-H (DBT3) , SysBench

Data Written to SSS



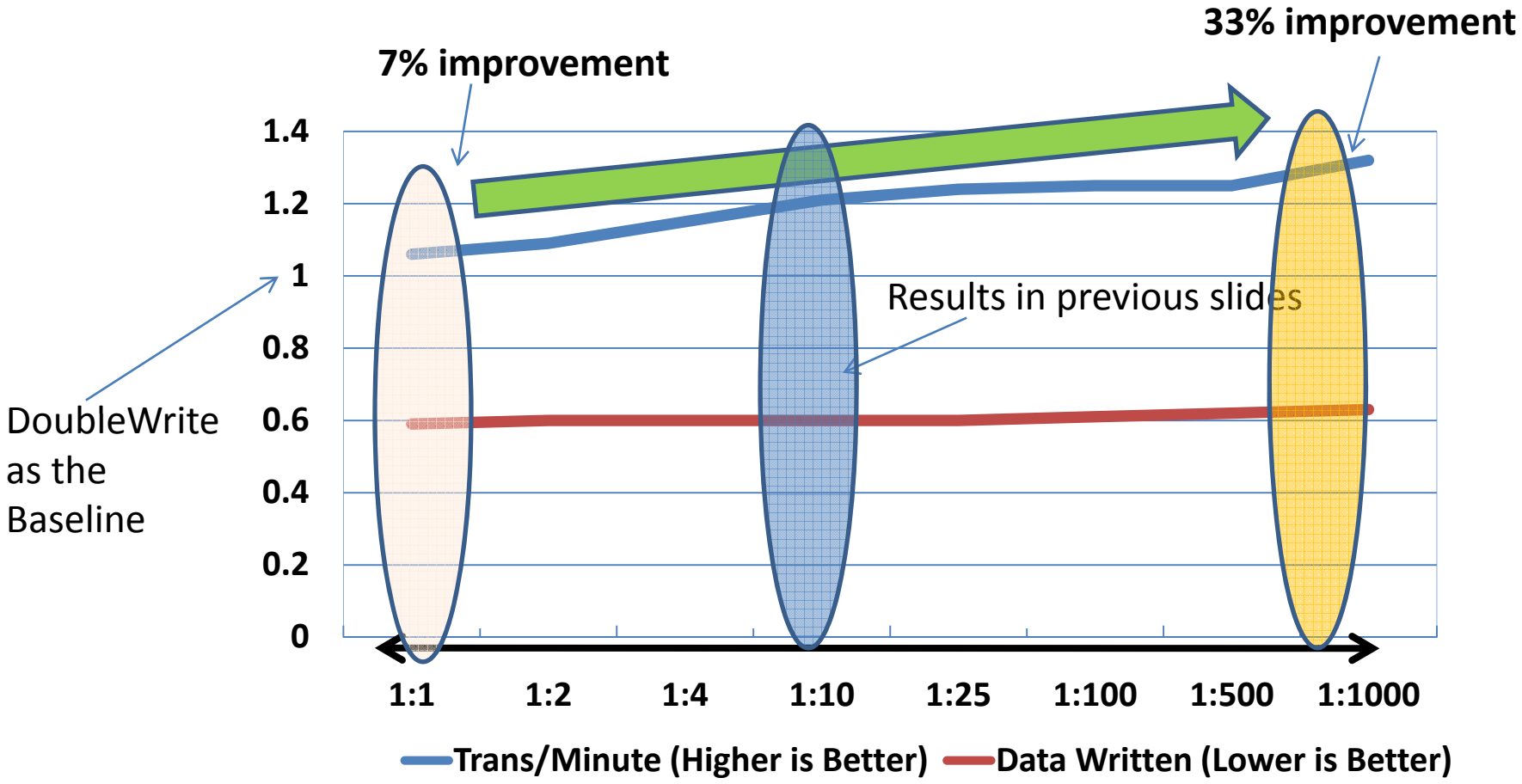
- Buffer Pool : Database = 1 : 10
- DB workload: TPC-C (DBT2) , TPC-H (DBT3) , SysBench

Transaction Latency



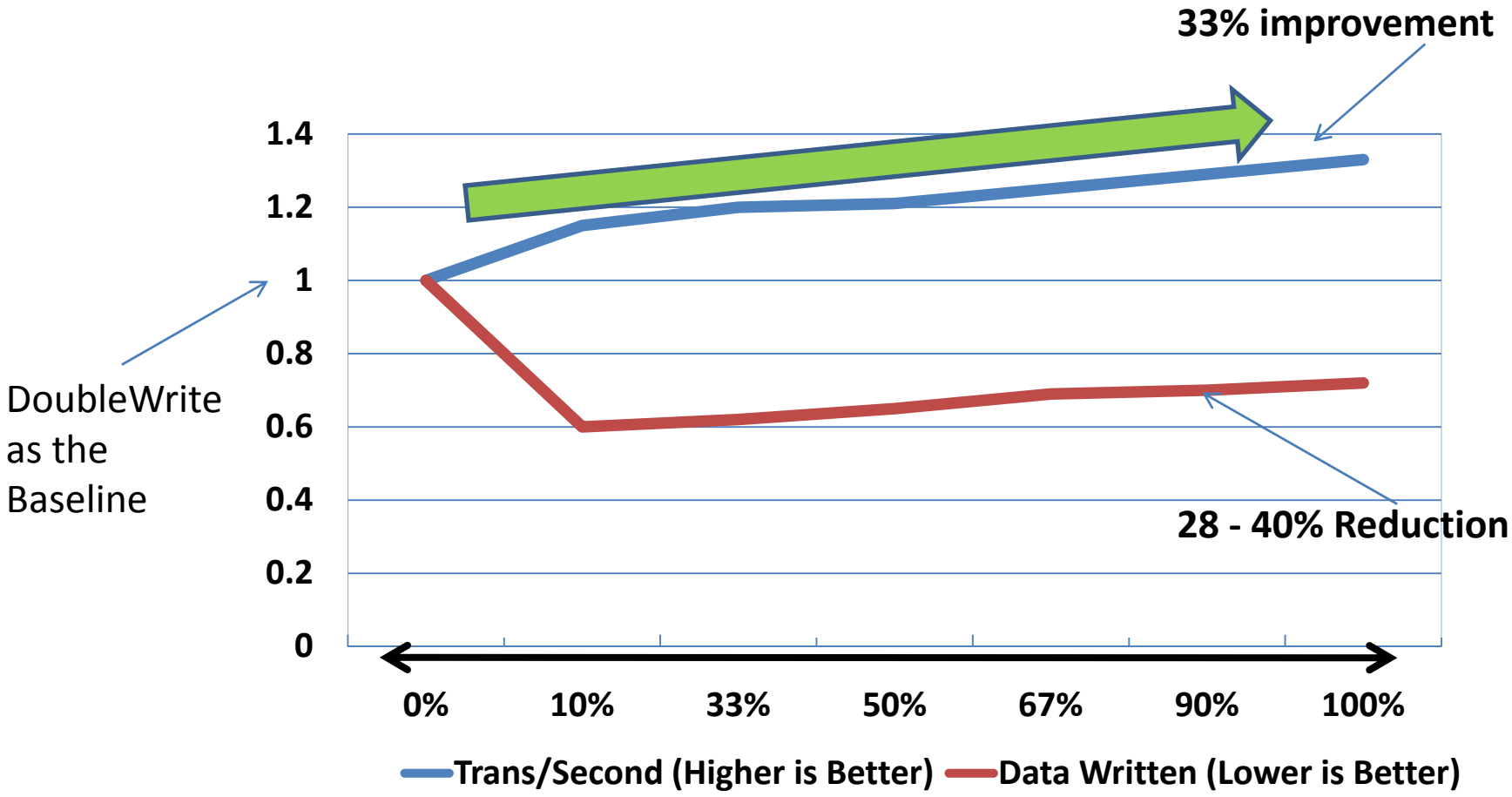
- Buffer Pool : Database = 1 : 10
- DB workload: TPC-C (DBT2) , TPC-H (DBT3) , SysBench

DB-buffer-pool size : DB on-disk size



- DB workload: TPC-C (DBT2)
- Vary Buffer Pool : Database size
- Atomic-Write vs. DoubleWrite

DB Records Update Ratio



- DB workload: SysBench
- Vary Update ratio in total workload
- Atomic-Write vs. DoubleWrite

Conclusions

- Solid State Storage opens opportunities for higher order primitives in storage interfaces
- Atomic-Write: allows multi-block write operations to be completed as an atomic unit
- Benefit upper layers with ACID requirements
  - OS, Filesystem, DBMS, applications
  - Reduced complexity
  - Improved performance
  - Improved device durability

Future Work

- Work with Linux kernel maintainers to integrate atomic-write in a non-proprietary way
- To support multiple outstanding atomic-write groups
  - Full transactional support
- Explore other higher order I/O primitives

Thank You!