Lazy-Adaptive Tree: An Optimized Index Structure for Flash Devices

Devesh Agrawal

University of Massachusetts, Amherst, MA, USA

Collaborators: Deepak Ganesan, Yanlei Diao, Ramesh Sitaraman and Shashi Singh

Flash Based Databases

"Disk is Dead, Flash is Disk"

Jim Gray, 2006

Advantages Over Disk

- Fast Random Reads
- Energy Efficient
- Robust
- Small Size

64GB SSD

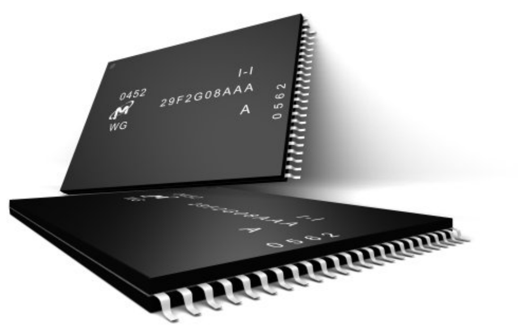
Indexing Over Flash

- Crucial for Efficient Retrieval
- Challenging on Flash

1GB NAND chip for Mote

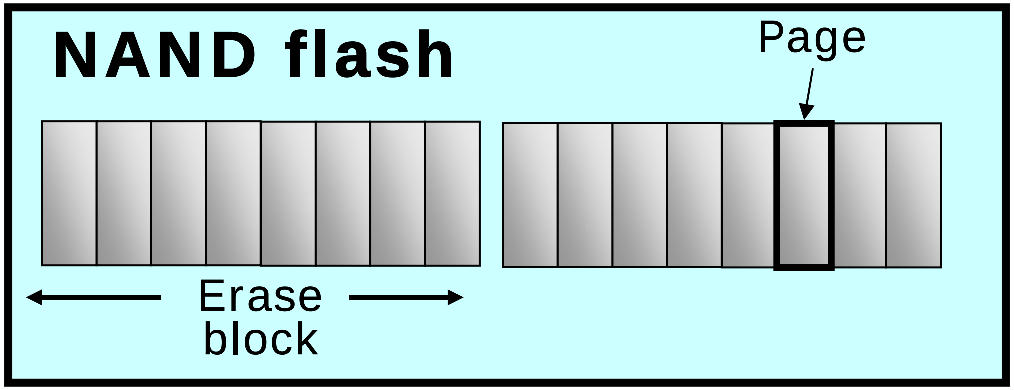
Flash Characteristics

Packaged Flash Disks



Raw NAND chips

Flash Hierarchy

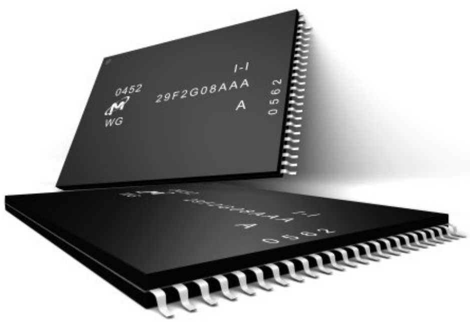


Flash Constraints

- Erase before rewrite
- Can’t delete individual pages

Flash Characteristics

Packaged Flash Disks



Raw NAND chips

Flash Hierarchy

Memory

Modify *single* page in-memory

Load *entire* block Into Memory

Write *whole* block back

Flash

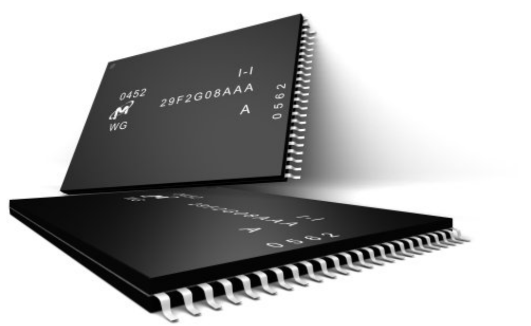
Erase block on flash

Existing Solutions

- ***In-Place* Updates**
- *Out-of-Place* Updates *(FTL)*

Flash Characteristics

Packaged Flash Disks

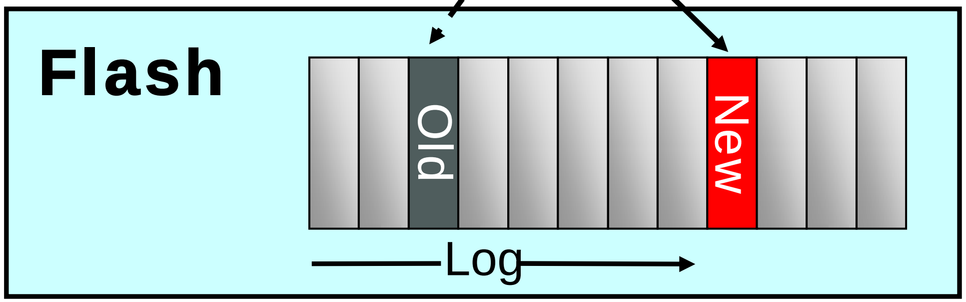


Raw NAND chips

Flash Hierarchy

FTL

Logical	Physical
100	120
101	250



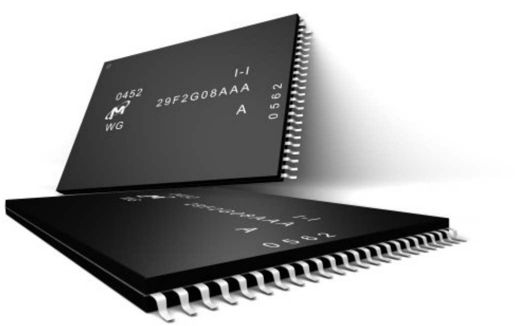
Existing Solutions

- *In-Place* Updates
- ***Out-of-Place* Updates (*FTL*)**

Page updates expensive

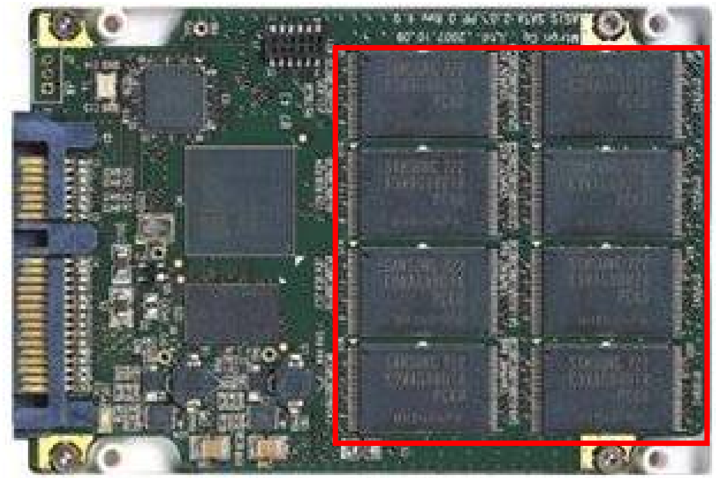
Flash Characteristics

Packaged Flash Disks



Raw NAND chips

Flash Hierarchy



Cut-Out of an SSD

- Controller provides disk interface
- Fast random reads & sequential I/O

Expensive Random Writes

[Birrell et al: SIGOPS 07]

Flash makes Indexing Hard

Flash Characteristics

Sequential Reads	✓
Random Reads	✓
Sequential Writes	✓
Random Writes	✗

Cheap Lookups (Sequential Reads, Random Reads, Sequential Writes)

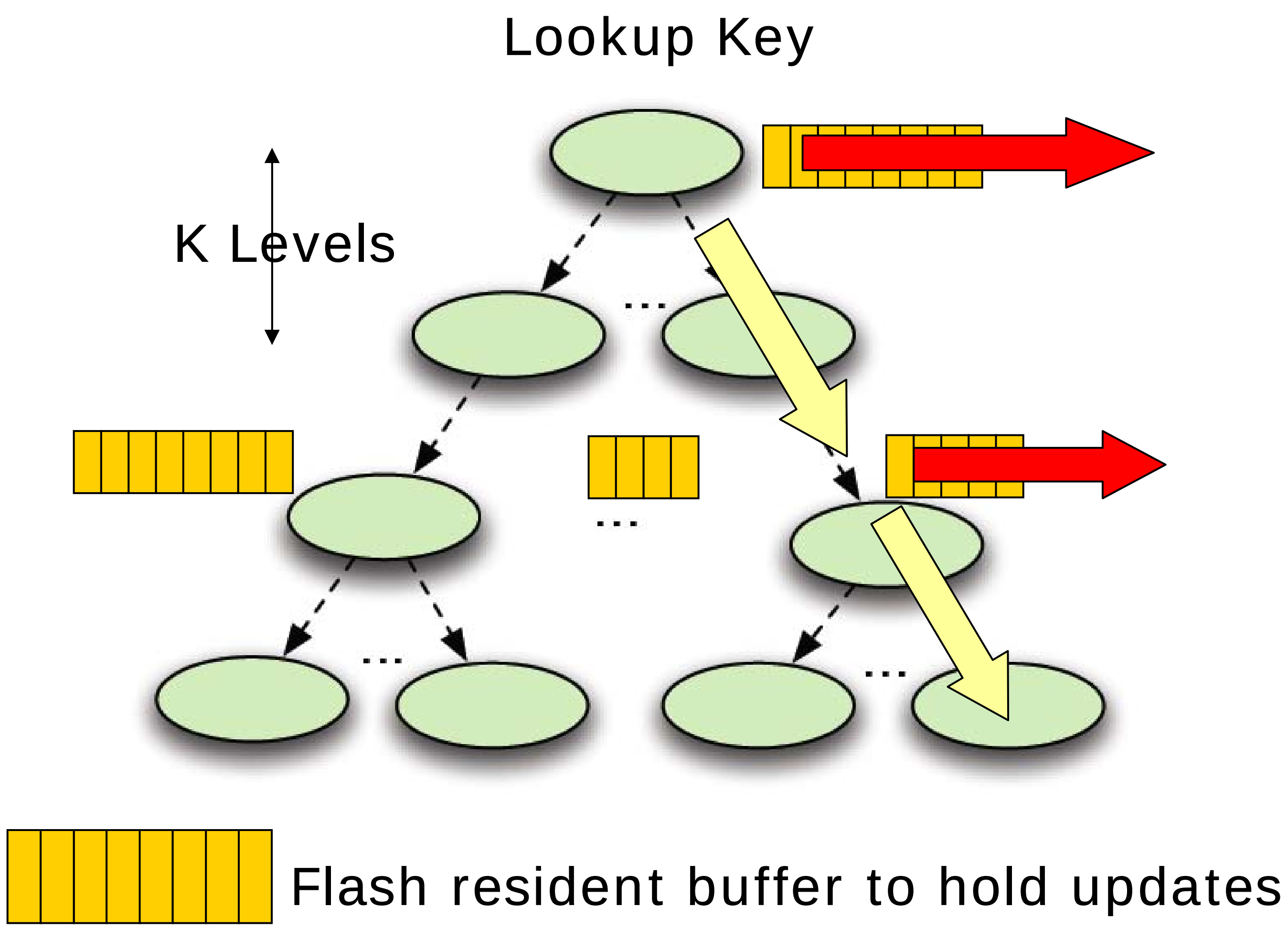
Expensive Updates (Random Writes)

***Goal:* Design a flash friendly index**

- Minimize random I/Os to improve update performance
- Perform efficiently for lookups

Lazy Adaptive Tree (LA-Tree)

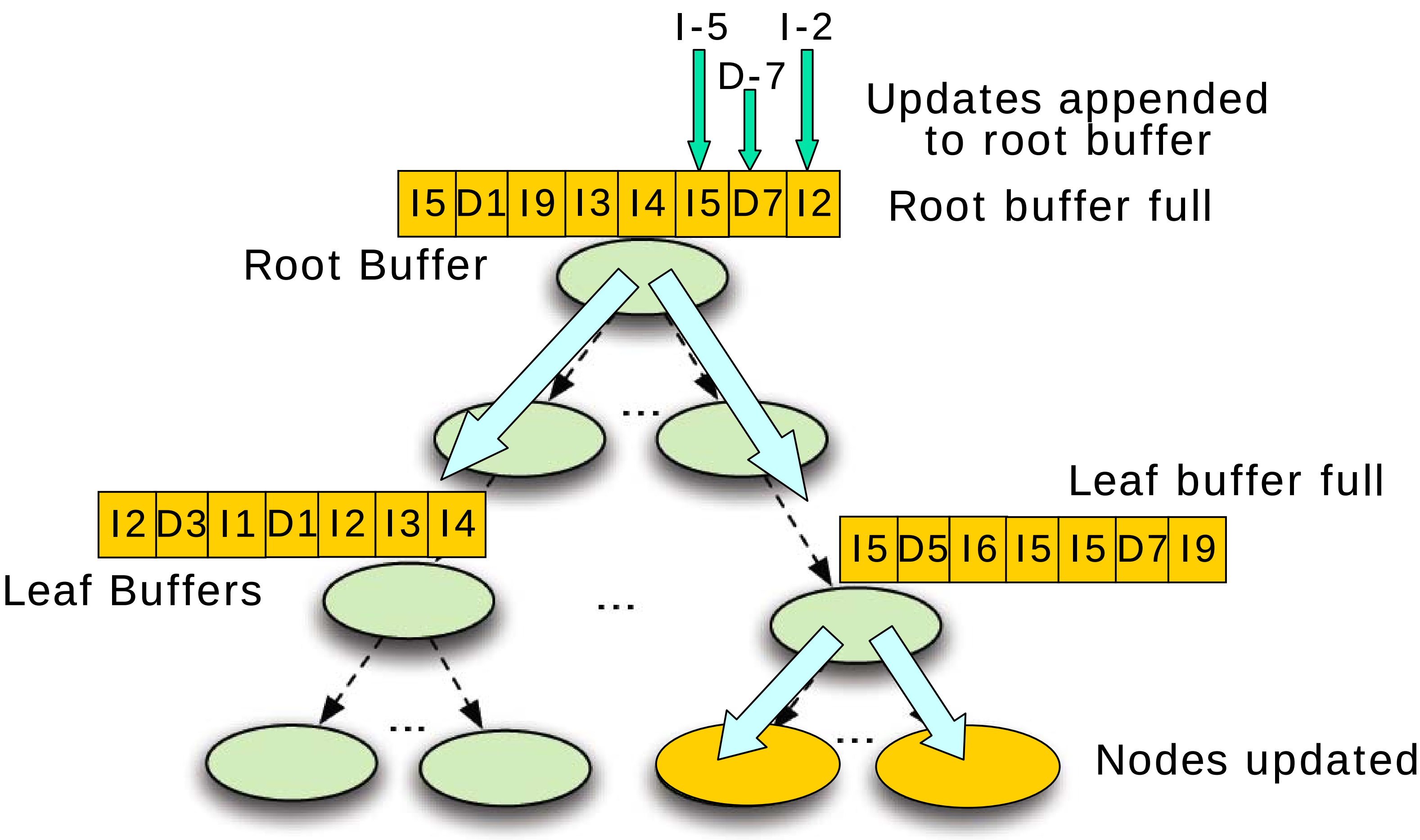
B+ Tree augmented with flash resident buffers to hold updates



Outline

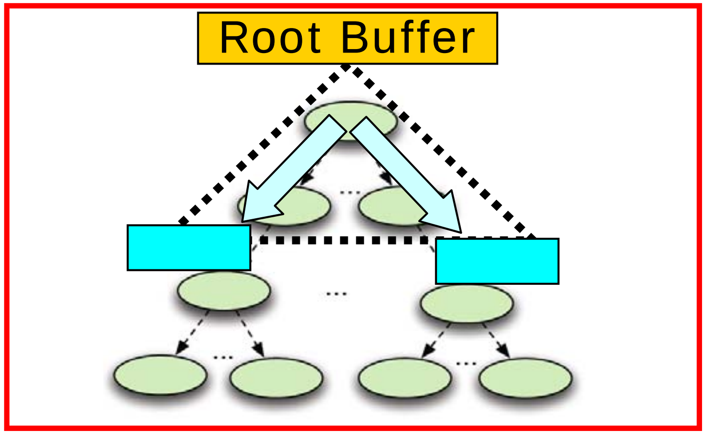
- Introduction
- LA-Tree Overview
- LA-Tree Design
  - Lazy Updates
  - Adaptive Buffer Size Control
- LA-Tree flash-optimized implementation
- Performance Evaluation
- Related Work
- Conclusion

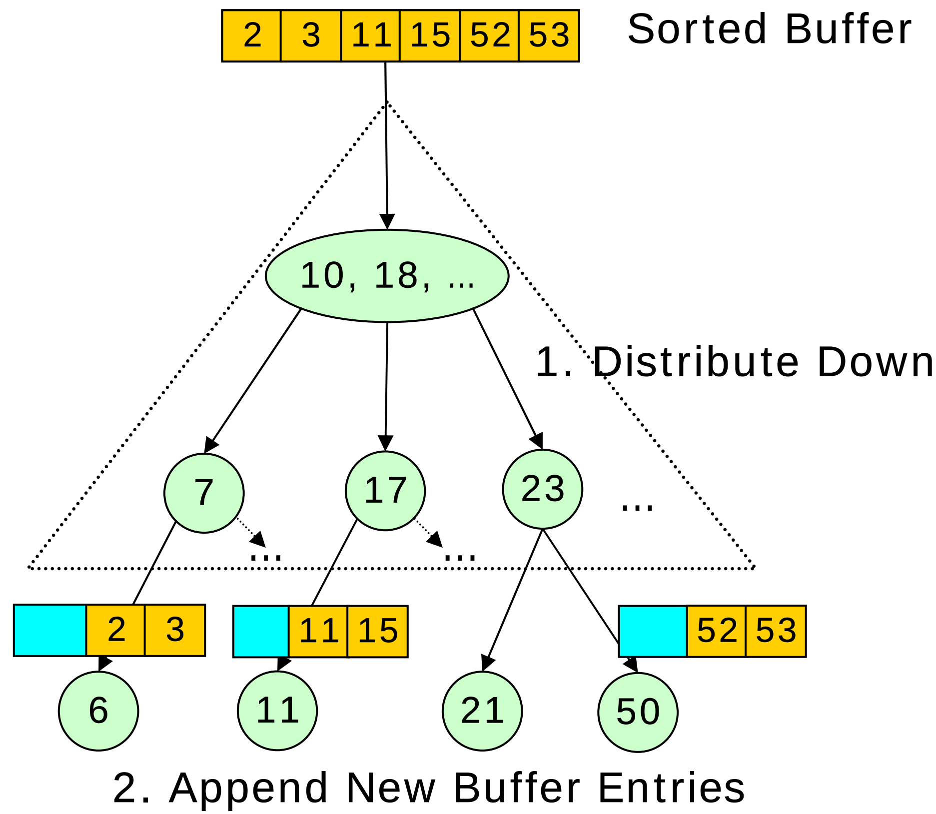
Idea 1: Lazy Updates



Buffer Empties push down updates in a batch

Lazy Updates: Non Leaf Buffer Empty

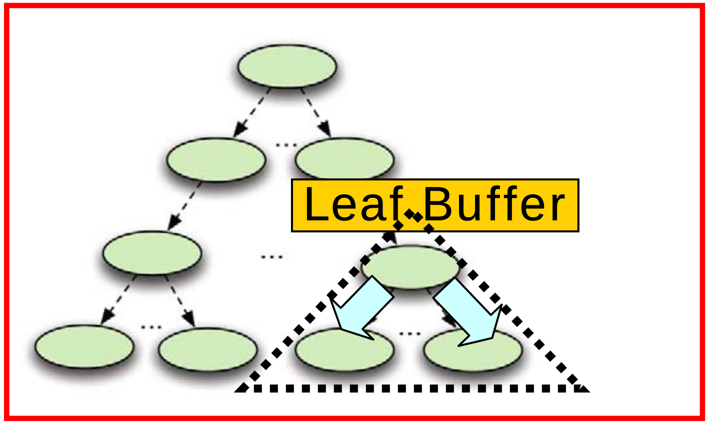


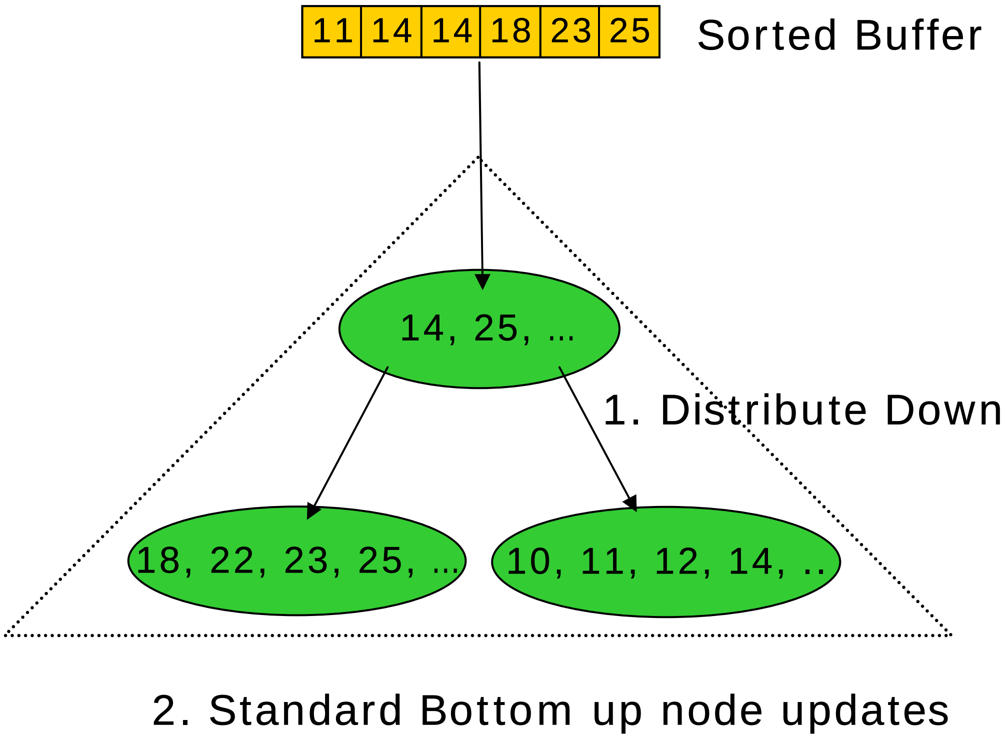


Advantages

- Amortizes node reads
- Buffers cheap to maintain

Lazy Updates: Leaf Buffer Empty

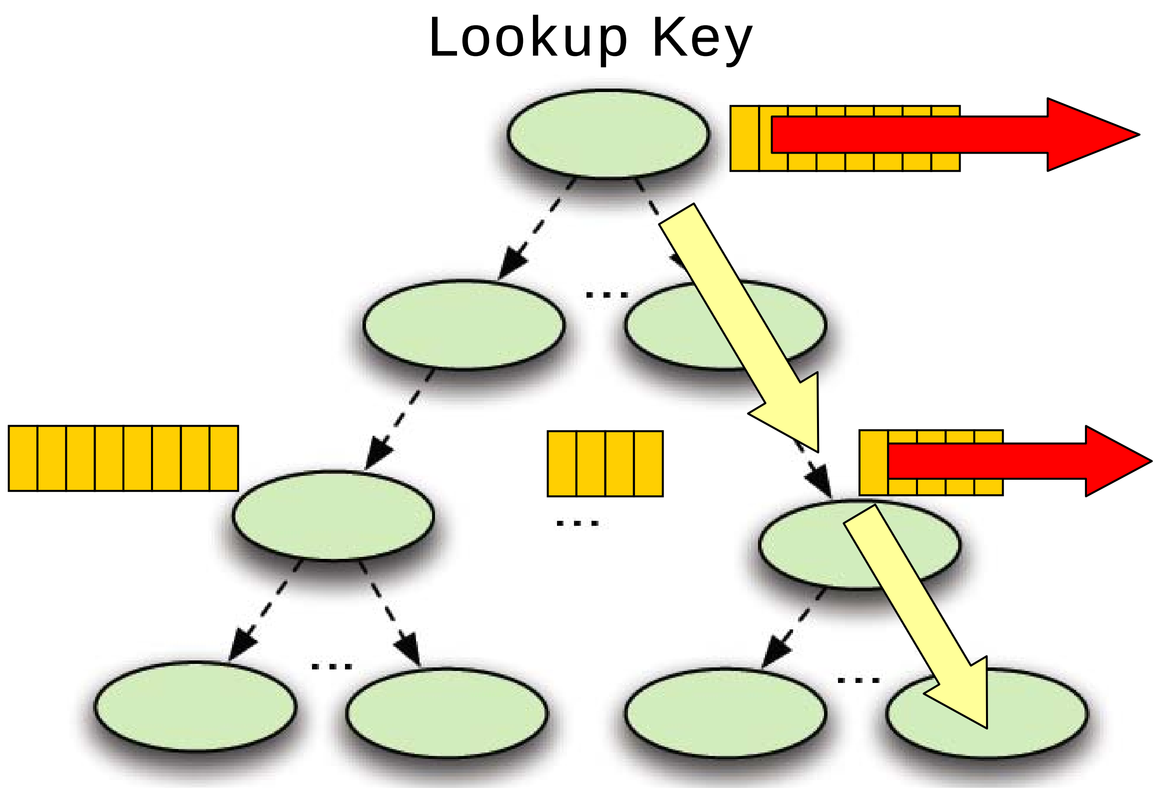




Advantages

- Amortizes node writes

Problem with Lazy Updates



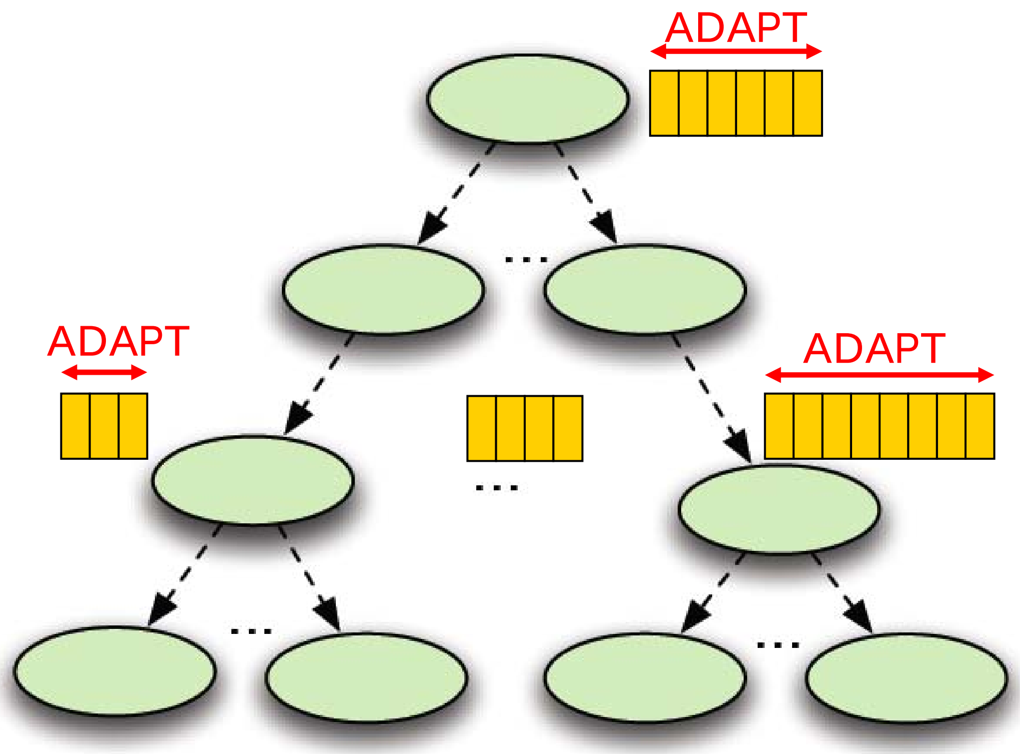
Critical to adapt buffer size to workload

- Lookup Dominated → Small Buffer
- Update Dominated → Large Buffer

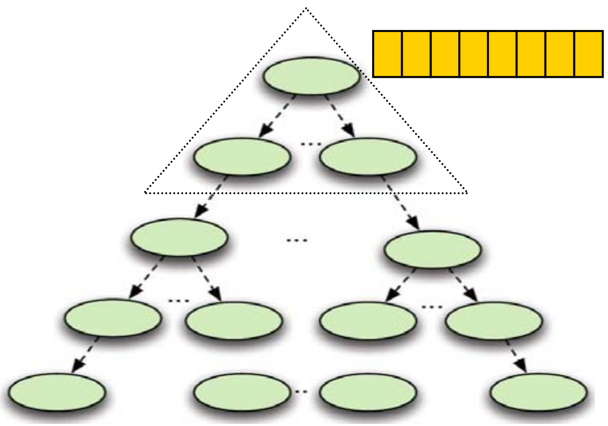
Idea 2: Buffer Size Control Algorithm

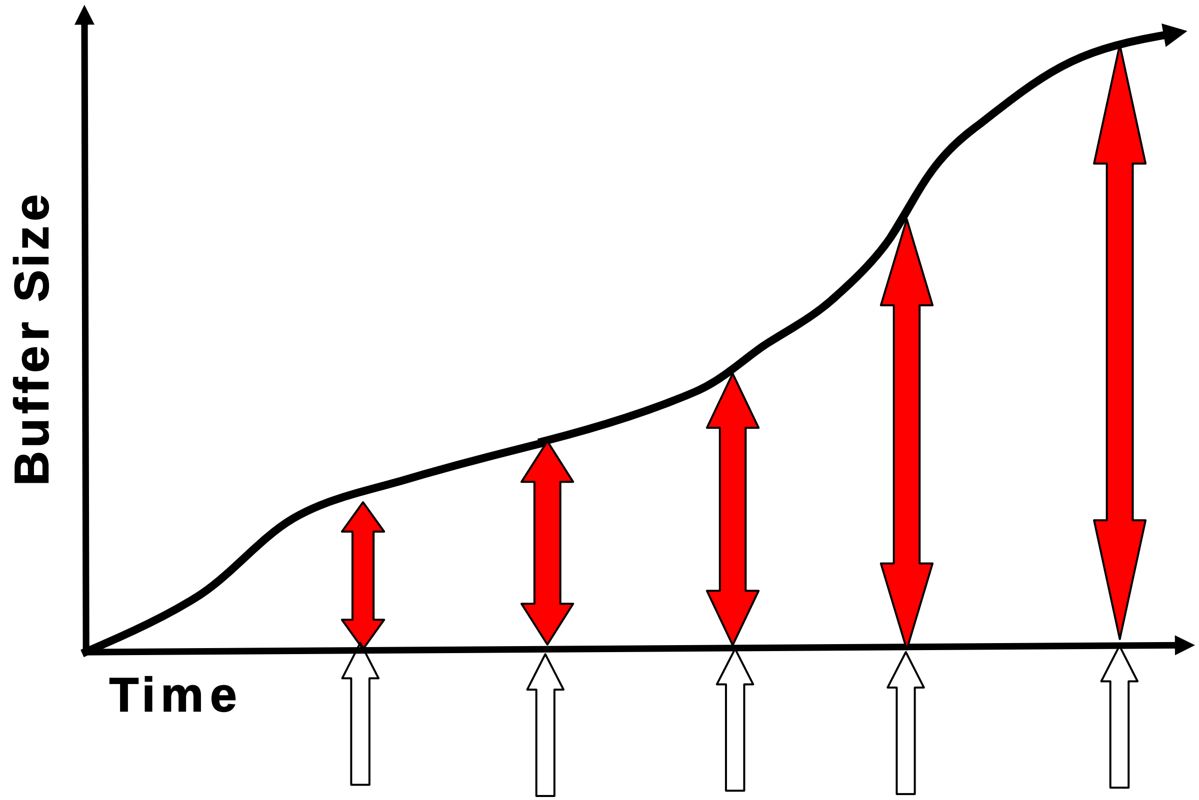
Features

- *Online Algorithm:* Makes no assumption about workload
- Adapts each buffer independently



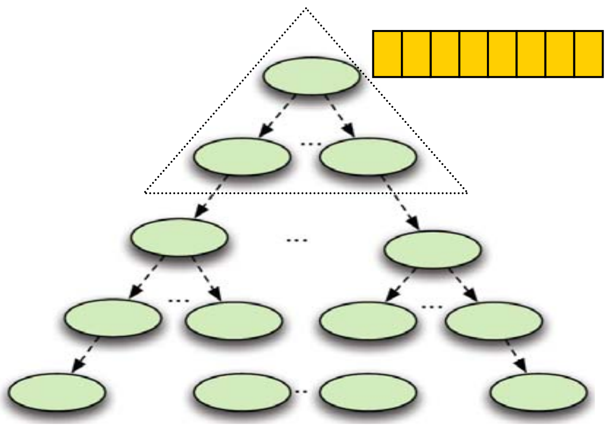
ADAPT : Intuition

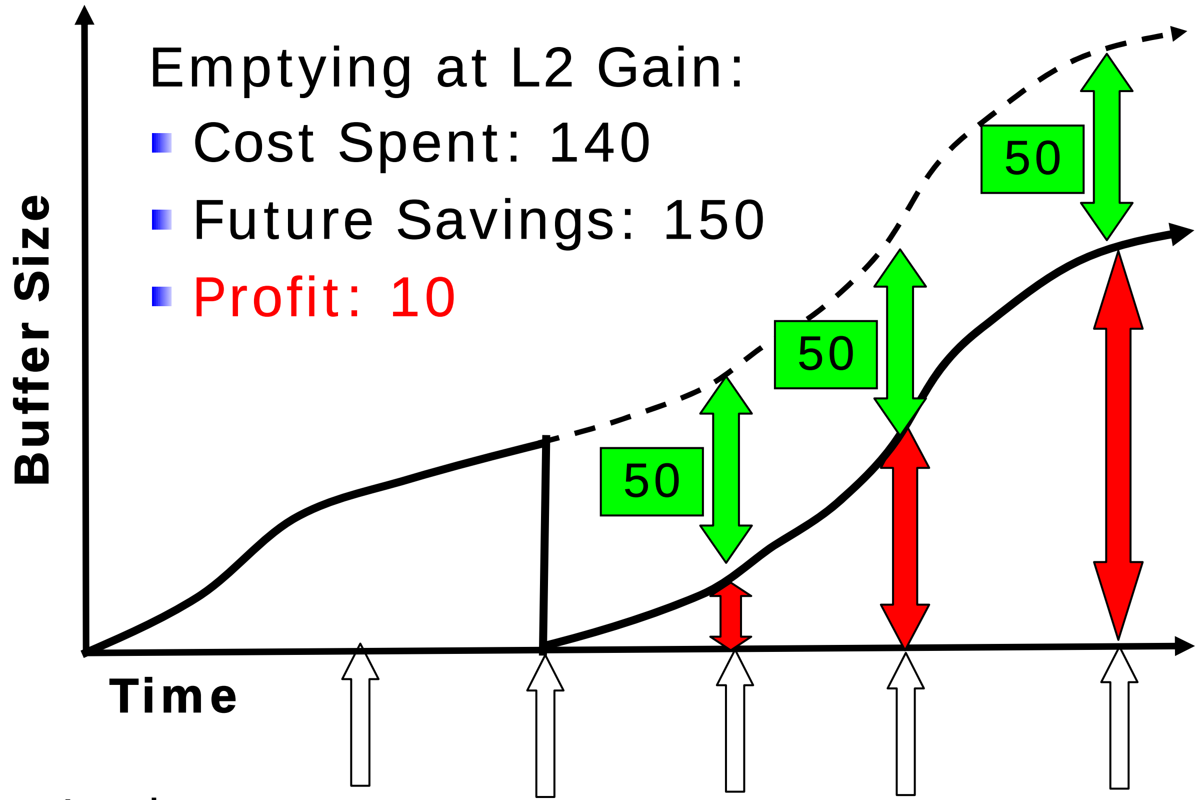




Lookups:	L1	L2	L3	L4	L5
Scan Cost:	15	50	80	110	150
Empty Cost:	70	140	200	260	340

ADAPT : Intuition



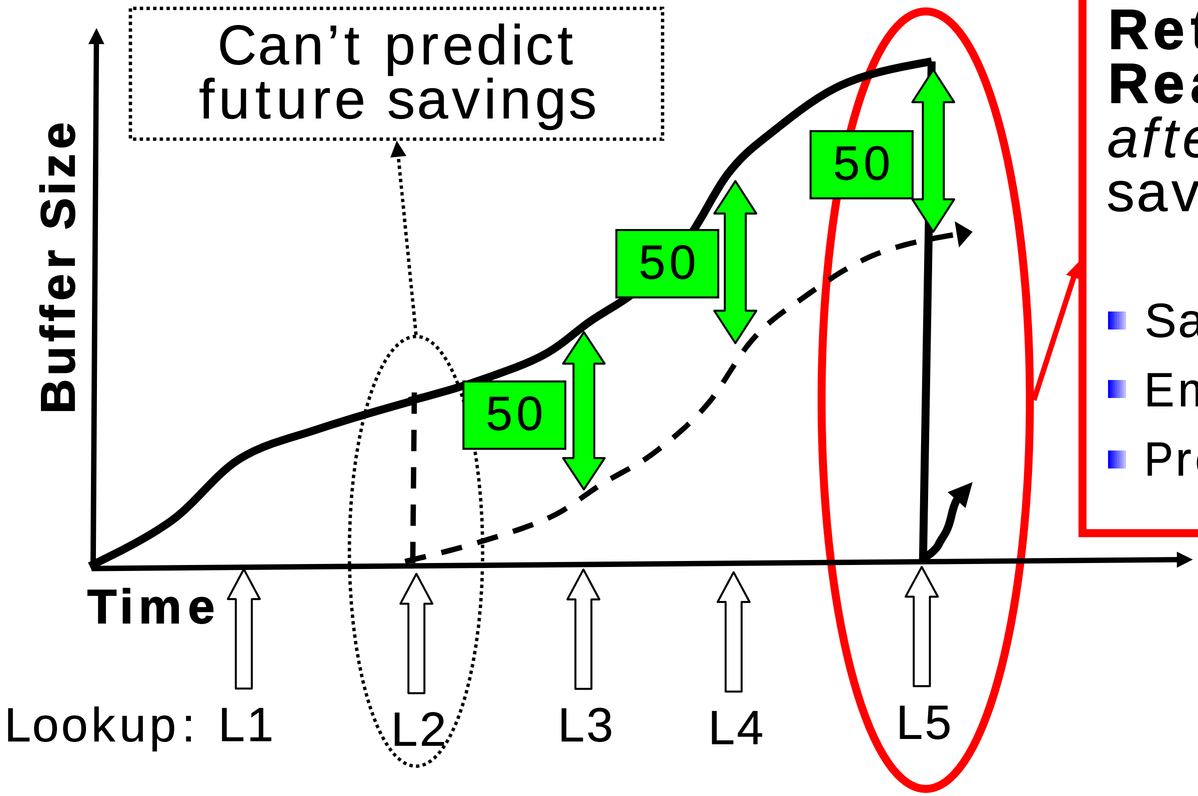


Emptying at L2 Gain:

- Cost Spent: 140
- Future Savings: 150
- Profit: 10

Lookups:	L1	L2	L3	L4	L5
Empty Cost:	70	140	200	260	340
Profit of emptying:	-10	10	-40	-150	-340

ADAPT : Retrospective Reasoning



Retrospective Reasoning: Empty here *after* observing L3-L5 savings

- Savings in Hindsight: 150
- Empty Cost at L2: 140
- Profit in Hindsight : 10

Optimal Online Algorithm

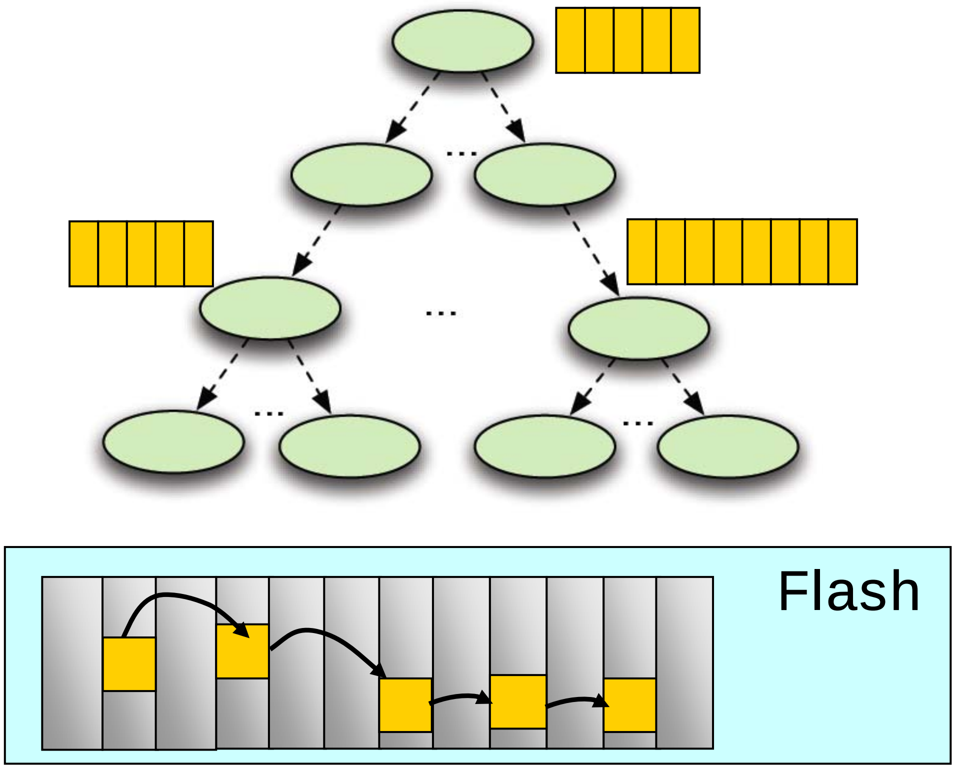
- Proved 2-competitive
- Best possible competitive ratio

Outline

- Introduction
- LA-Tree Overview
- LA-Tree Design
  - Lazy Updates
  - Adaptive Buffer Size Control
- LA-Tree flash-optimized implementation
- Performance Evaluation
- Related Work
- Conclusion

Implementation Challenges

- Reduce Buffer Fragmentation
- Reduce Garbage Collection overhead

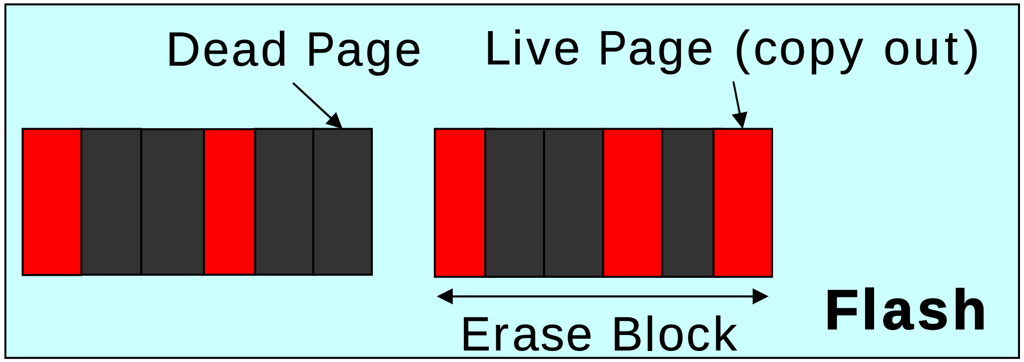


Buffer maintained as linked list on flash

Fragmentation increases buffer read cost

Implementation Challenges

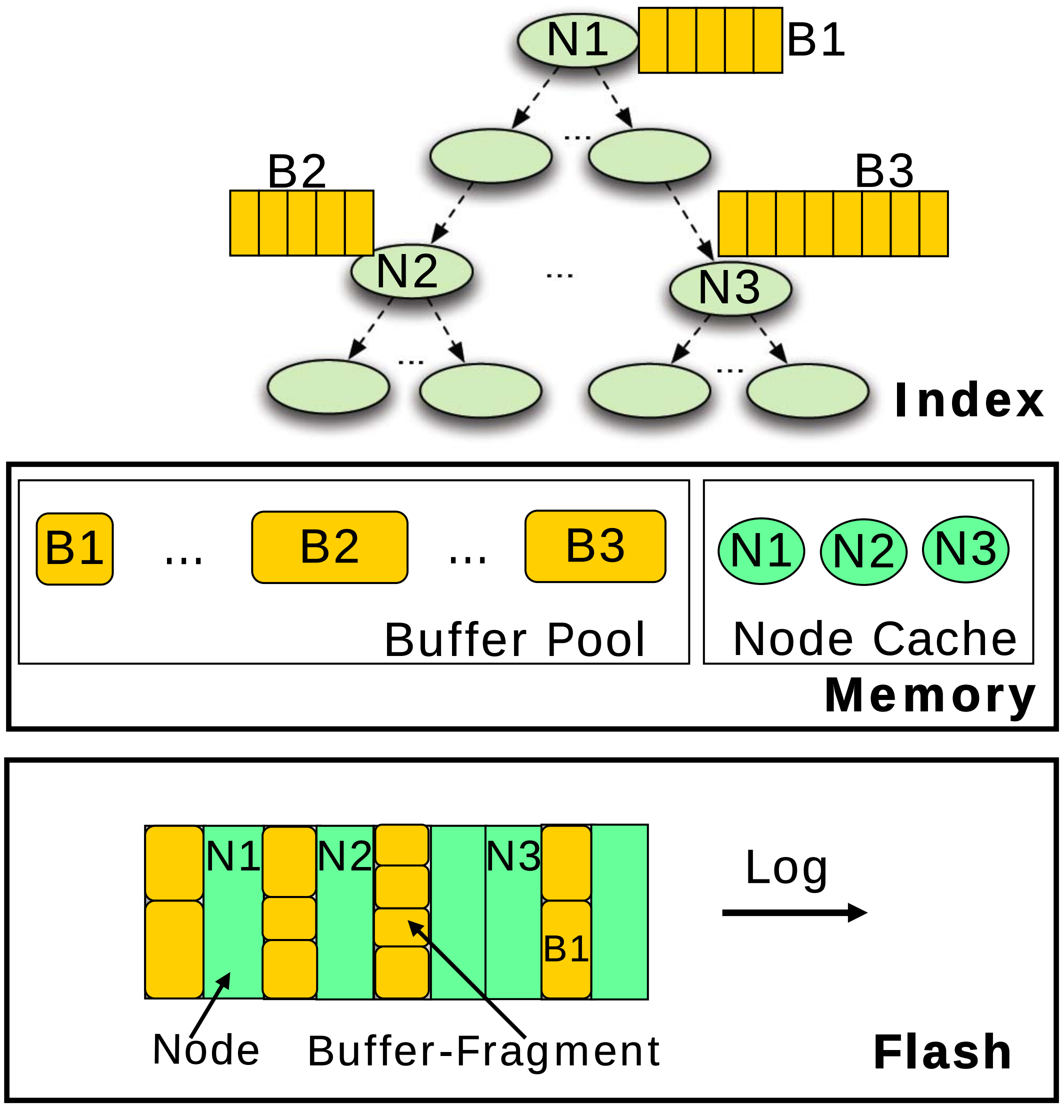
- Reduce Buffer Fragmentation
- Reduce Garbage Collection overhead



Flash increases GC costs

LA-Tree System Architecture

- Memory Manager
  - Write Coalescing Buffer Pool
  - LRU node cache
  - Reduces Buffer Fragmentation
- Flash Manager
  - Flash written as a log
  - Empty buffers before GC
  - Reduces GC overhead



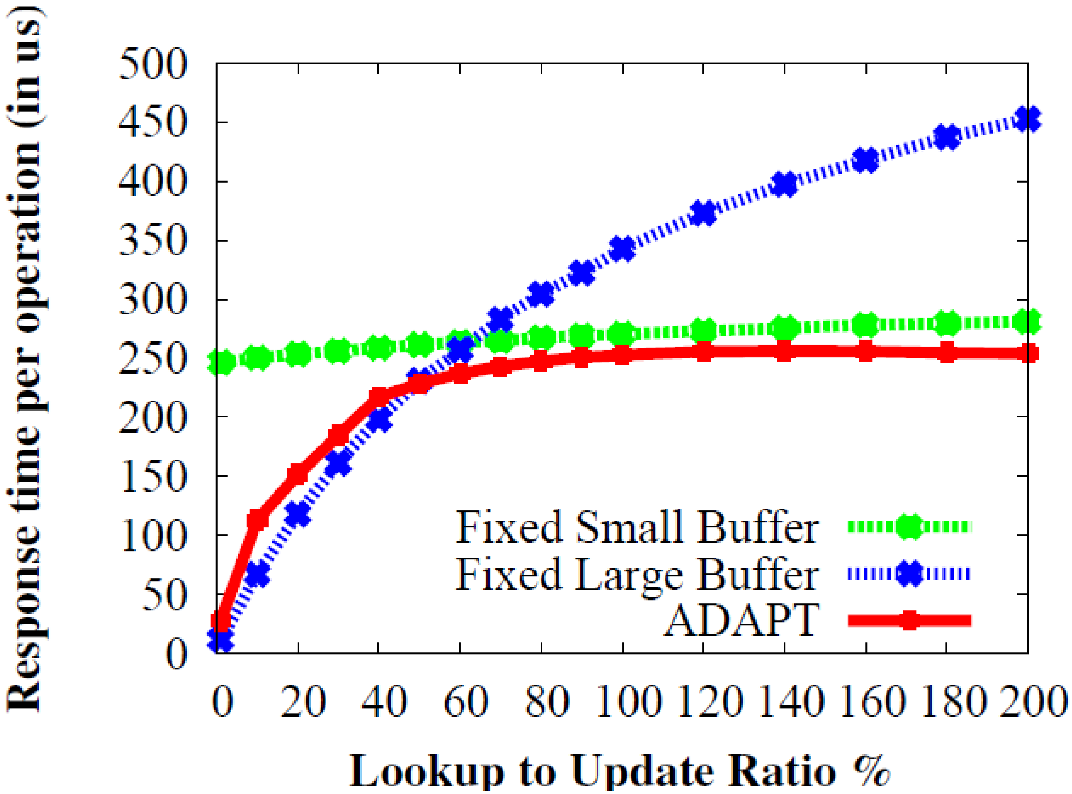
Outline

- Introduction
- LA-Tree Overview
- LA-Tree Design
  - Lazy Updates
  - Adaptive Buffer Size Control
- LA-Tree flash-optimized implementation
- Performance Evaluation
- Related Work
- Conclusion

Evaluation Setup

- Flash devices
  - *Toshiba* TC58DVG02A1FT00 *1Gb NAND flash*
  - *MTRON PRO 7000 16GB SATA SSD*
- Data sets
  - *Synthetic*: Uniformly random key distributions
  - *Scientific DB*: meteorological radar data traces
  - *Transactional*: TPC-C Index Trace

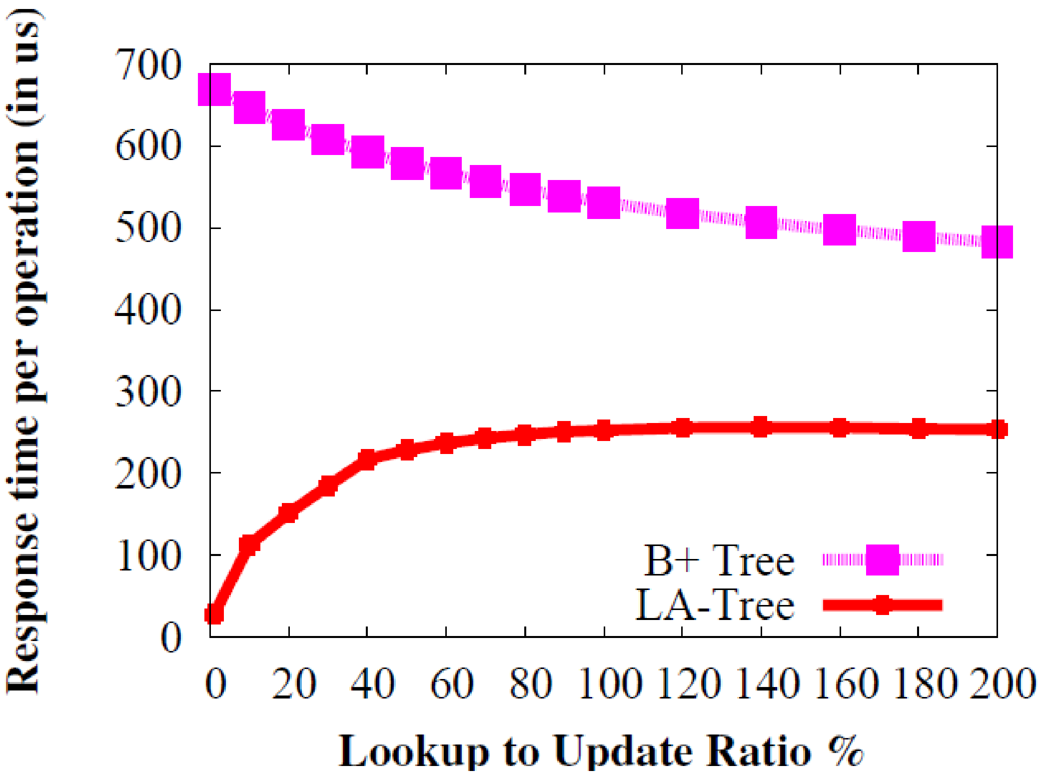
MicroBenchmarks: ADAPT evaluation



- Raw NAND flash
- 128KB RAM
- Synthetic workload with 1M updates
- 33% updates are deletes

- Fixed buffer sizes do not scale with workload
- ADAPT matches any workload

MicroBenchmarks: versus B+ Tree



- Raw NAND flash
- 128KB RAM
- Synthetic workload with 1M updates
- 33% updates are deletes

(B+ tree implemented as LA-Tree with buffering turned off)

LA-Tree outperforms B+ tree across all workloads

Versus Existing Schemes (NAND)

	Synthetic		Sci-DB	TPC-C	
LTU	10%	200%	0.01%	8% (Customer)	197% (Order)
FlashDB	1270	844	1385	550	111
BFTL	1322	2073	1478	862	69
IPL	2489	1702	2205	999	115
LA-tree	113	254	163	119	31
Gain	**11.2 x**	**3.3 x**	**8.5 x**	**4.6 x**	**2.2 x**

Response Time Per Operation in micro-seconds

- High gains over spectrum of workloads
- Gains are higher for update-heavy workloads

[FlashDB: IPSN 07], [IPL: SIGMOD 07], [BFTL: TECS 07]

Versus Existing Schemes (SSD)

	Synthetic		Sci-DB	TPC-C	
LTU	10%	200%	0.01%	8% (Customer)	197% (Order)
FlashDB	1487	1083	642	1877	64
BFTL	1463	1192	1175	2661	108
IPL	2791	2289	6705	5312	544
LA-tree	315	351	12	216	9
Gain	**4.6 x**	**3 x**	**52 x**	**5.5 x**	**6.6 x**

Response Time Per Operation in micro-seconds

LA-Tree significantly reduces random writes

[FlashDB: IPSN 07], [IPL: SIGMOD 07], [BFTL: TECS 07]

Summary of Other Results in Paper

- Garbage Collection:
  - GC overhead comparable to the best
- Memory Manager:
  - Gains scale with memory
- Other Flash Devices
  - 37% to 5x gains over SD cards
  - 8x – 23x gains over large block raw NAND flashes

Outline

- Introduction
- LA-Tree Overview
- LA-Tree Design
  - Lazy Updates
  - Adaptive Buffer Size Control
- LA-Tree flash-optimized implementation
- Performance Evaluation
- Related Work
- Conclusion

Related Work

- **Flash optimized B+ Tree implementations**
  [FlashDB: IPSN 07], [IPL: SIGMOD 07], [BFTL: TECS 07]
  - Optimize individual node layout
  - Layout improves node updates at expense of node reads
- **Buffer Trees** [Arge: Algorithmica 03]
  - Lookups not supported
  - Not optimized for flash
- **FD-Trees** for SSDs [Li: ICDE 09]
  - Avoids random writes by repeated sequential writes

Summary and Ongoing Work

LA-Tree is a new flash optimized index

- **Idea 1:** Lazy Updates
- **Idea 2:** Adaptive buffer size control
- Efficient implementation over flash and memory constraints
- Large gains over many workloads and flash types

Ongoing Work

- Transaction support
- Extend to *GiST* family of Indexes
- Extend to other storage devices (HDD in particular)

Thank You

- Questions ?