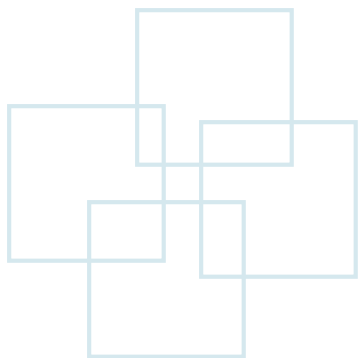


Rethinking Self-balancing Binary Search Tree over Phase Change Memory with Write Asymmetry



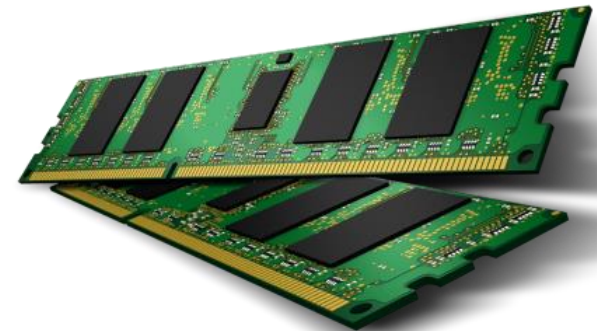
ASP-DAC 2018

Outline

- Introduction
- Background and Motivation
- Write-Asymmetry-Aware Self-Balance Tree
 - Basic Concepts
 - Analysis of Tree Rotations
 - Depth-First-Alternating Traversal
 - Address Conflict Manager
- Evaluation
 - Experimental Setup
 - Experimental Results
- Conclusion

The Needs of Huge Memory/Storage

- Big data and data mining applications require huge storage
 - Store huge amount of data
 - In-memory computing
- DRAM and NAND Flash are hitting the wall of its transistor scaling*
 - Density limitation
 - High power consumption
 - Leakage power
 - Operation power

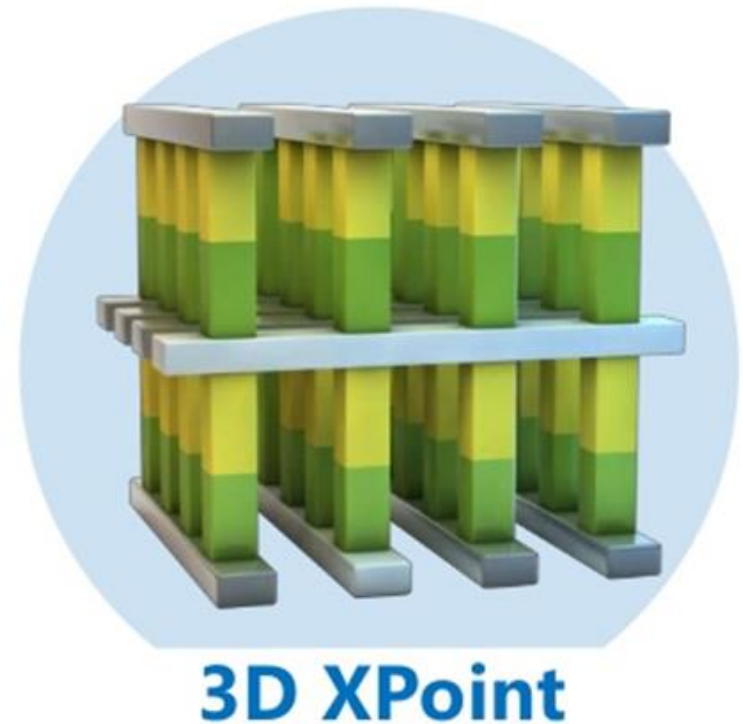


* : B. C. Lee, E. Ipek, O. Mutlu, and D. Burger. Architecting phase change memory as a scalable DRAM alternative. In *The International Symposium on Computer Architecture (ISCA)*, 2009.

H. Zheng and Z. Zhu. Power and performance trade-offs in contemporary DRAM system designs for multicore processors. *IEEE Transactions on Computers*, 59(8), 2010.

Non-volatile Memory

- Phase-change memory and 3D Xpoint
 - High density
 - Low leakage power
 - Non-volatility
- Features of 3D Xpoint
 - 1000X faster than NAND
 - 1000X more endurance of NAND
 - 10X higher density than DRAM

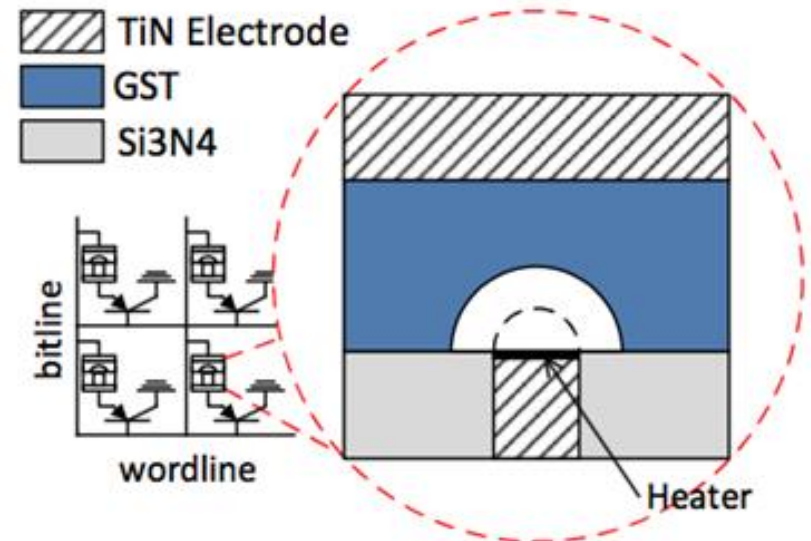


Outline

- Introduction
- Background and Motivation
- Write-Asymmetry-Aware Self-Balance Tree
 - Basic Concepts
 - Analysis of Tree Rotations
 - Depth-First-Alternating Traversal
 - Address Conflict Manager
- Evaluation
 - Experimental Setup
 - Experimental Results
- Conclusion

Phase Change Memory (PCM)

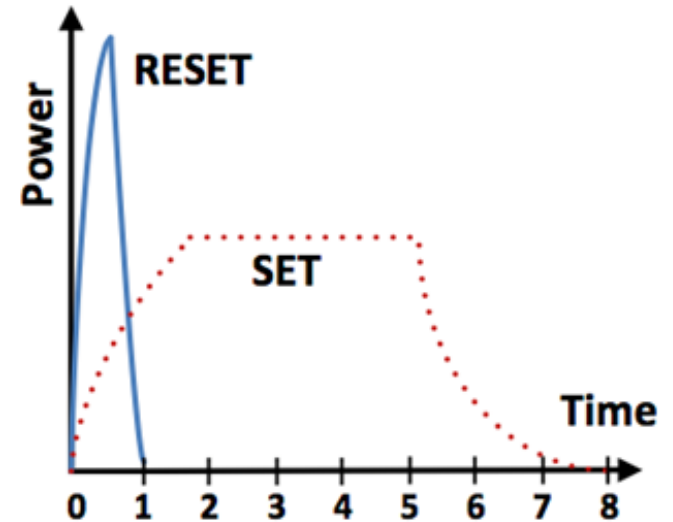
- Physical structure and mechanisms
 - Two phases :
 - The **amorphous** phase with high resistance
 - The **crystalline** phase with lower resistance
- Advantages
 - High density
 - Non-volatility
 - Low power consumption
 - Outstanding I/O performance
 - Byte addressability



Issues of Using PCM

- Write Asymmetry

- Reset
 - High instant power with short time
- Set
 - Low power with long time



- Write Latency

- Write Endurance

- Suggested Solution?

- Reducing #write bits

Types & Attributes	DRAM	PCM
Non-volatility	No	Yes
Bit alterability	Yes	Yes
Retention time	~ 60 ms	> 10 years
Density	20 – 32 nm	< 20 nm
Write endurance	$> 10^{15}$ cycles	$10^6 - 10^8$ cycles
Write latency	20 – 50 ns	150 ns
Read latency	50 ns	50 ns

Relative Works

- Data-Comparison Write (DCW)
 - Read the **old (stored)** data.
 - Do comparison with the new data.
 - Skip any bit write if it is not needed.
- Coset Coding
 - Provide a one-to-many mapping for each data word to a (co)set of vectors.
 - Choose the **vector with the minimum overhead** for each write.

Motivation

- Big/massive data applications demand extremely **large main memory space** for better performance.
 - PCM has **low leakage power** and **high density** which make it a promising candidate to replace DRAM.
 - **Write endurance** and **latency** are critical for using PCM.
 - Existing studies improve the write mechanism to handle **the given** write patterns on PCM.
- ***Why don't we fundamentally generate more suitable write patterns for PCM***
- ***By improving address allocation for data structures***

Outline

- Introduction
- Background and Motivation
- Write-Asymmetry-Aware Self-Balance Tree
 - Basic Concepts
 - Analysis of Tree Rotations
 - Depth-First-Alternating Traversal
 - Address Conflict Manager
- Evaluation
 - Experimental Setup
 - Experimental Results
- Conclusion

AVL Tree

- The properties of an AVL tree (for n data nodes)
 - The average and worst-case space utilization: $O(n)$
 - The average and worst-case time complexity of tree search, insertion, update and deletion: $O(\log n)$
- The expense for having the above properties
 - **Tree Rotations**
 - Conducting a rotation if the height difference of the left and right sub-trees is more than one level.
 - **Multiple Update Writes**
 - For each rotation, there are multiple update writes of pointers.
 - It could exacerbate the endurance and latency issues on PCM.

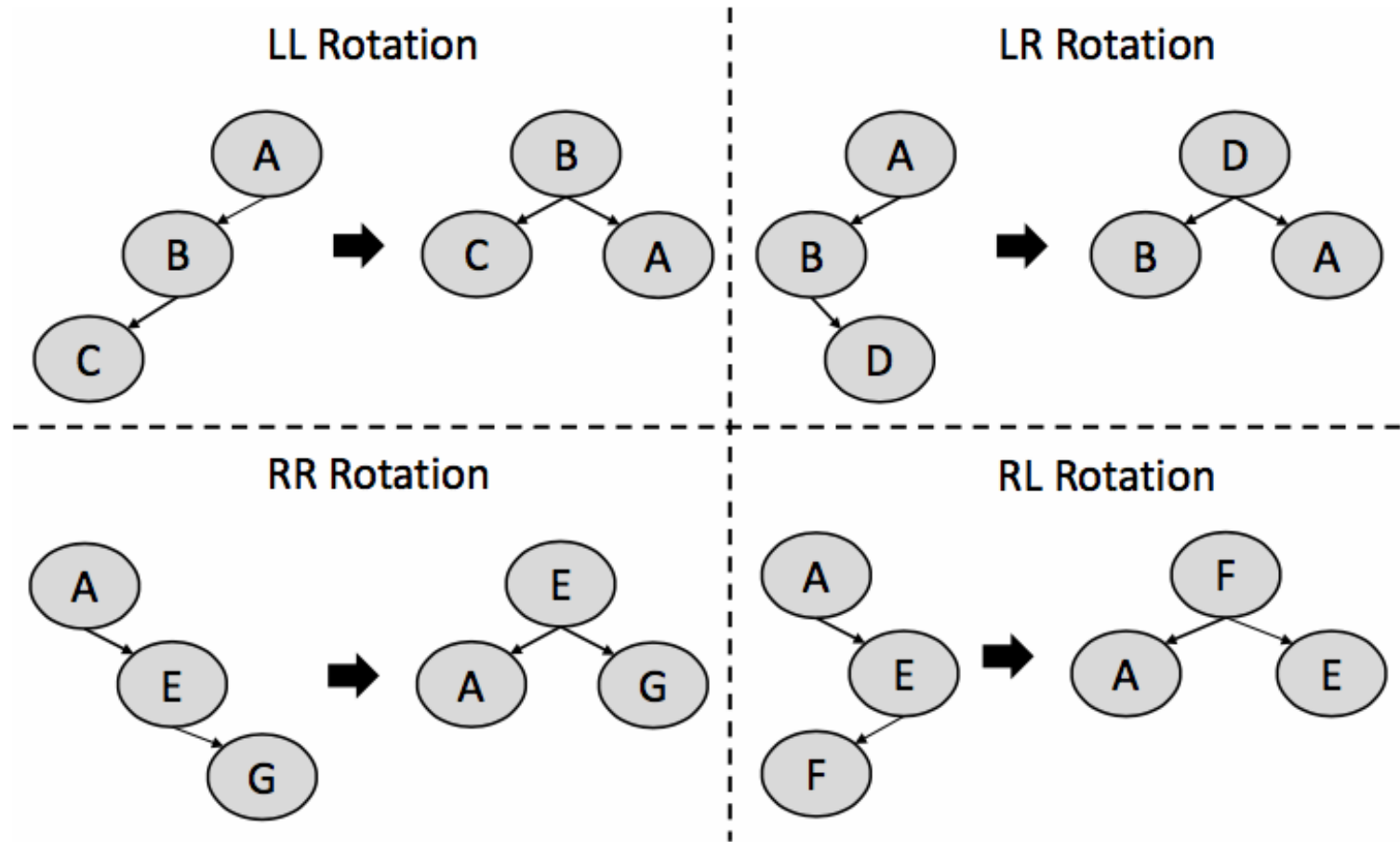
An Overview of Our Design

- **Tree rotation analysis** is conducted to better understand the relation among nodes.
- Our ***DFAT algorithm*** is developed to find the node relation path with the consideration of possible tree rotations.
 - The **Gray code** technique is leveraged to minimize the distance of two given address values in the address sequence that we use to map to our node relation path.
 - An **address conflict manager** is proposed to resolve possible address conflicts caused by rotations.

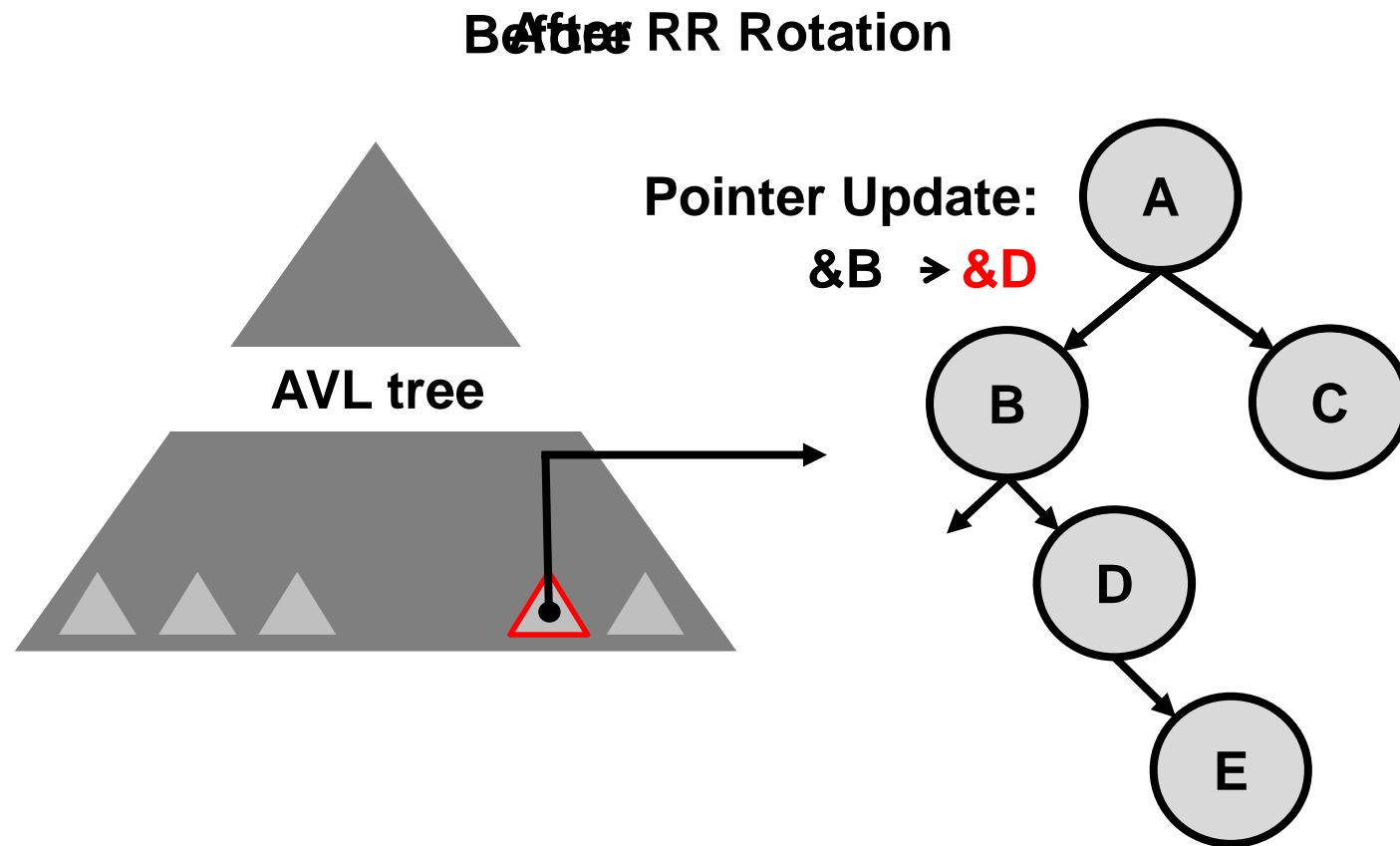
Outline

- Introduction
- Background and Motivation
- Write-Asymmetry-Aware Self-Balance Tree
 - Basic Concepts
 - Analysis of Tree Rotations
 - Depth-First-Alternating Traversal
 - Address Conflict Manager
- Evaluation
 - Experimental Setup
 - Experimental Results
- Conclusion

Four Types of AVL Tree Rotations

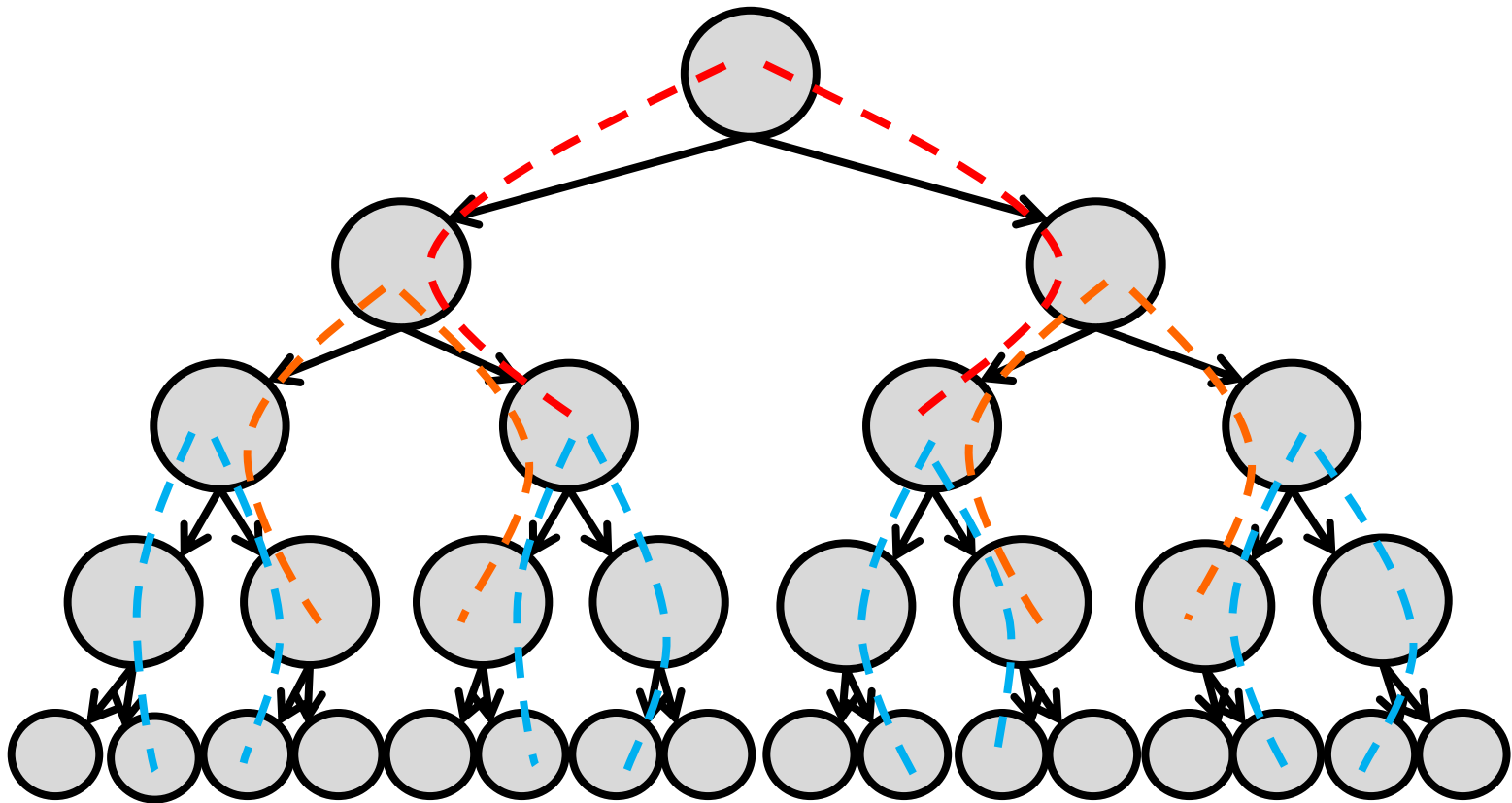


Relation among Nodes in an RR Rotation



Our Idea: Assign nodes B & D with close addresses!

Relation Paths of Tree Nodes

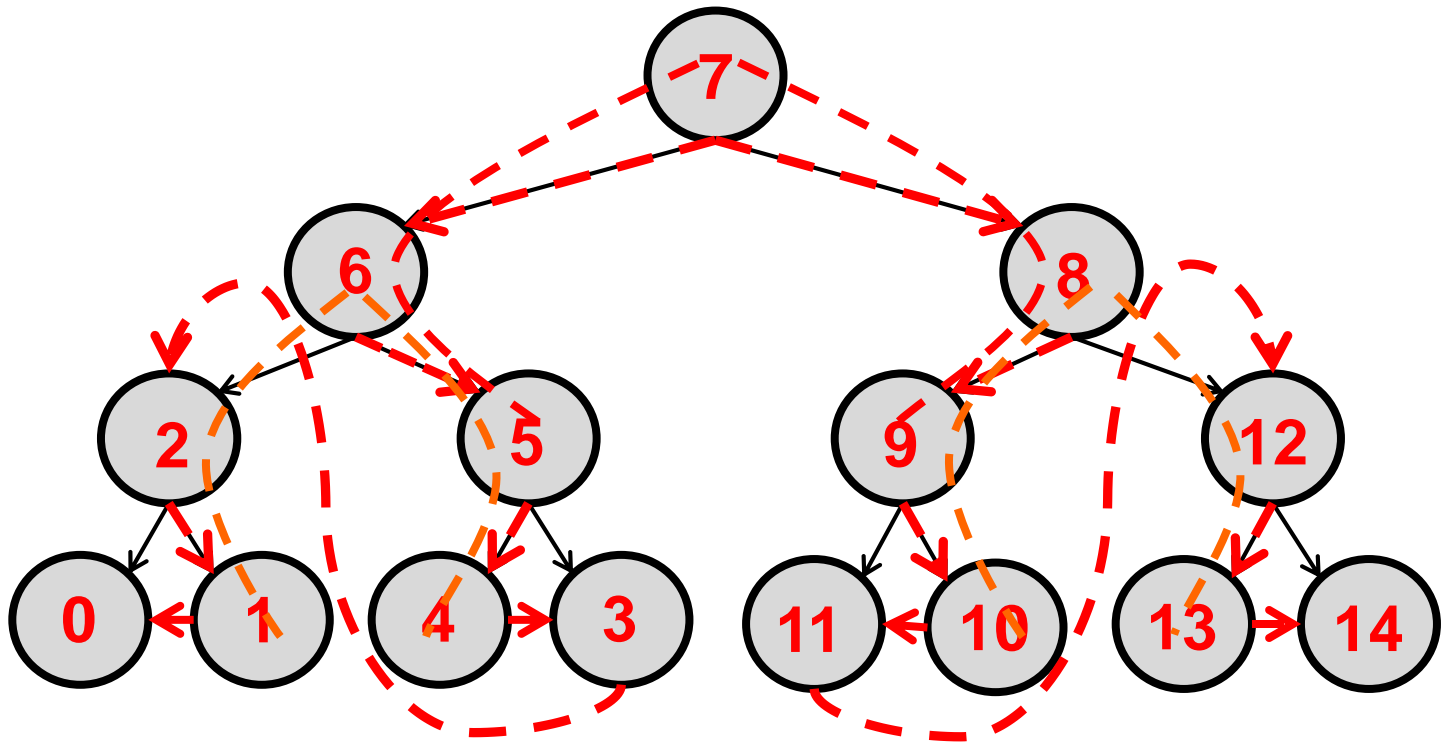


Outline

- Introduction
- Background and Motivation
- Write-Asymmetry-Aware Self-Balance Tree
 - Basic Concepts
 - Analysis of Tree Rotations
 - Depth-First-Alternating Traversal
 - Address Conflict Manager
- Evaluation
 - Experimental Setup
 - Experimental Results
- Conclusion

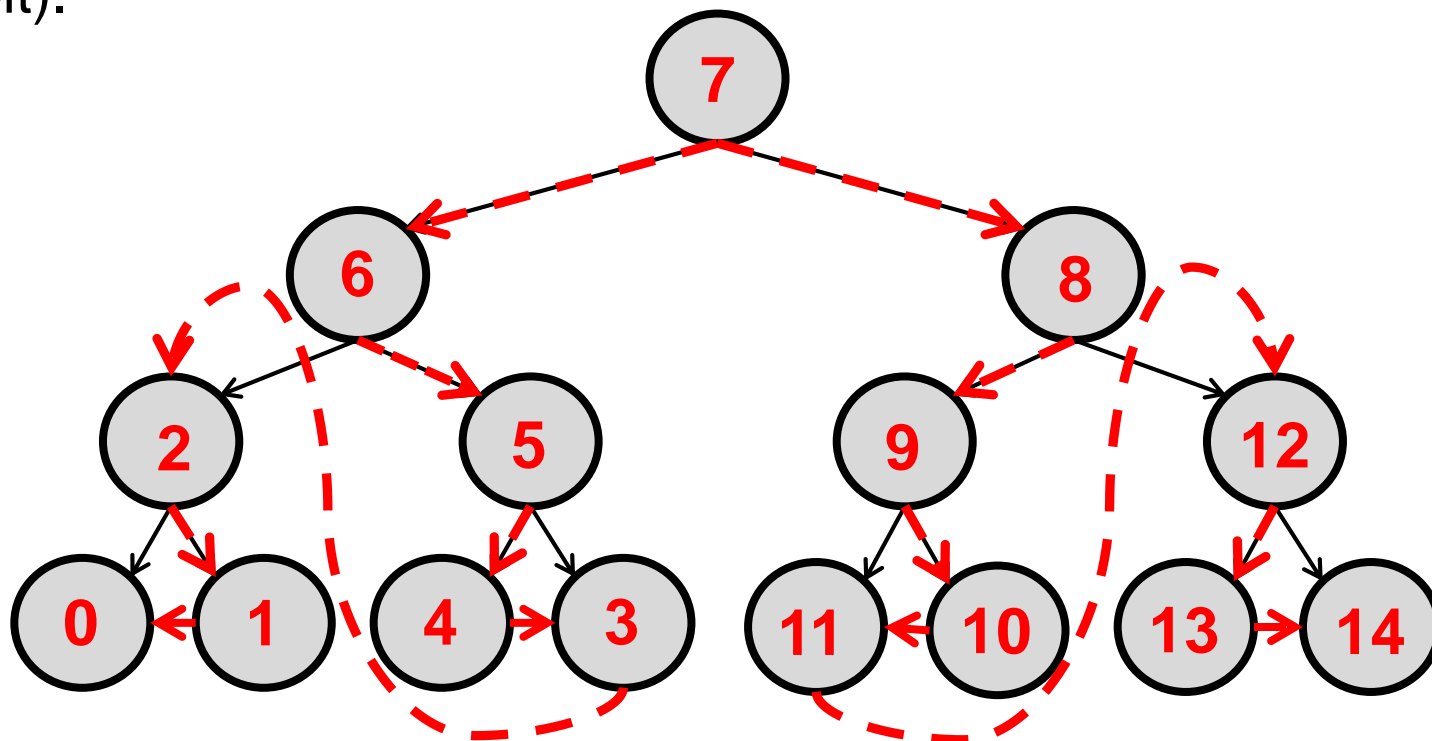
Depth-First-Alternating Traversal (DFAT)

- A systematic approach for indexing all nodes, where nodes having **stronger relations** will be assigned **closer indexes**.



Leveraging Gray Code on DFAT

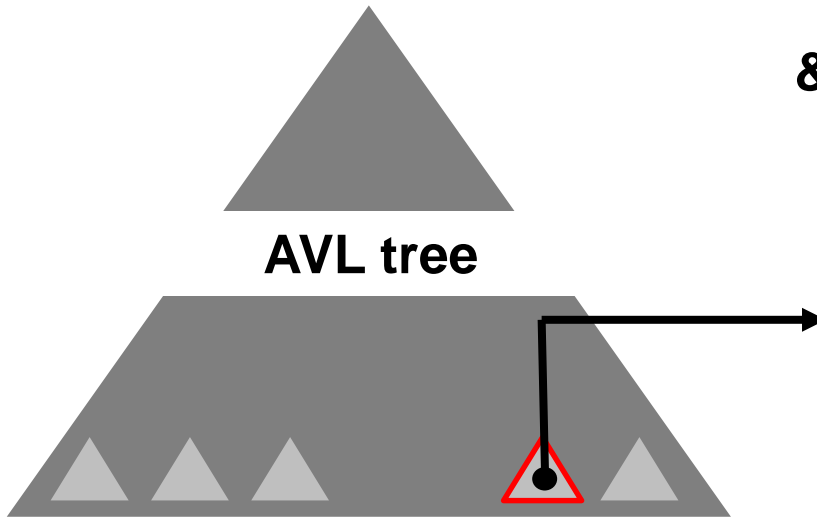
- **Gray code:** An ordering of the binary numeral system such that **two successive values** have the **shortest distance** (differ in only one bit).



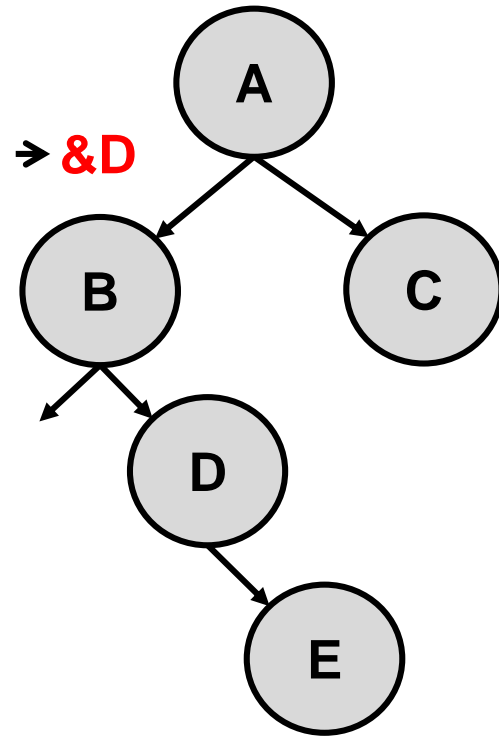
Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Gray Code	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110

A Running Example Of Our Solution

Before RR Rotation



&B → &D



Key value	Binary code address	Gray code address
A	0111111111111101	0100000000000011
C	0111111111111110	0100000000000001
B	<u>0111111111111111</u>	<u>0100000000000000</u>
D	1000000000000000	1100000000000000
E	1000000000000001	1100000000000001

Outline

- Introduction
- Background and Motivation
- Write-Asymmetry-Aware Self-Balance Tree
 - Basic Concepts
 - Analysis of Tree Rotations
 - Depth-First-Alternating Traversal
 - Address Conflict Manager
- Evaluation
 - Experimental Setup
 - Experimental Results
- Conclusion

Node Address Collisions

- The problem of address collisions
 - The address of a to-be-inserted node might be used because there are rotations.
- Our address conflict manager
 - Build a **redundant queue** for keeping some addresses.
 - Put free and can-not-be-used (by **DFAT**) addresses into the redundant queue after a tree rotation.
 - **Select one free address** from the redundant queue when there is an **address collision** during an insertion.

Outline

- Introduction
- Background and Motivation
- Write-Asymmetry-Aware Self-Balance Tree
 - Basic Concepts
 - Analysis of Tree Rotations
 - Depth-First-Alternating Traversal
 - Address Conflict Manager
- Evaluation
 - Experimental Setup
 - Experimental Results
- Conclusion

Experimental Setup (1/2)

- The experiment was conducted in a simulator we made to evaluate the proposed solution with **different numbers of data items** and **different sizes of address space**.
- The random permutation of the data array is used as the input data, and we sequentially insert the data items into an AVL tree.
- For the input size of N nodes, we reserve memory space for an AVL tree of $L + 2$ levels, where $L = \lceil \log_2(N + 1) \rceil$.

Experimental Setup (2/2)

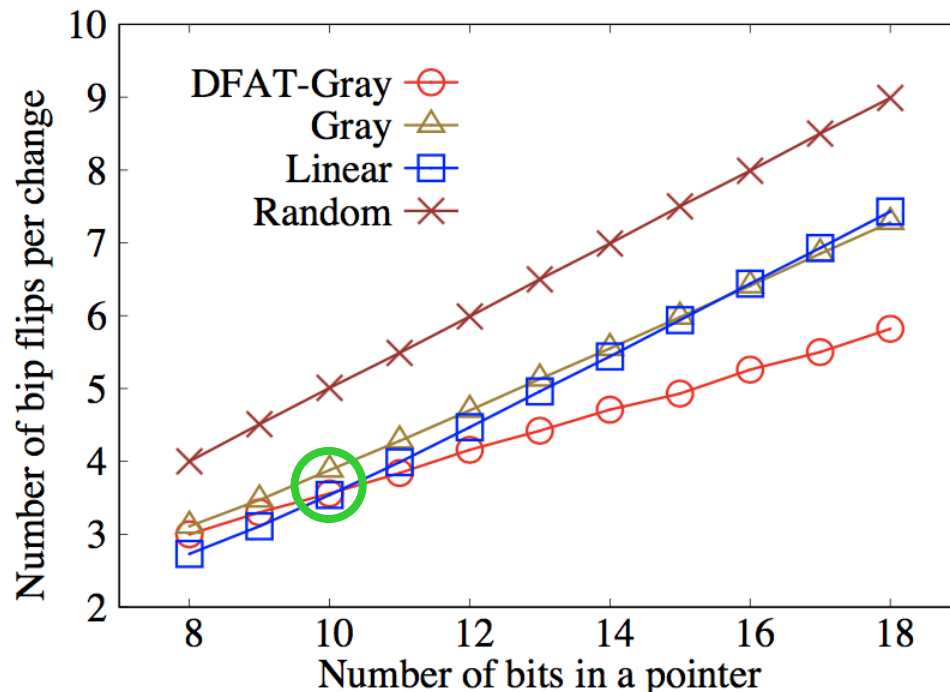
- We assign an address to each to-be-inserted node by the following solutions
 - *Random*: randomly selects an available address.
 - *Linear*: sequentially selects an available address, and the address value starts at 0.
 - *Gray*: uses the original tree indexes and leverages the Gray code technique to assign an address for each node.
 - *DFAT-Gray*: is our solution which indexes all nodes by the DFAT algorithm and leverages the Gray code technique to assign an address for each node.

Outline

- Introduction
- Background and Motivation
- Write-Asymmetry-Aware Self-Balance Tree
 - Basic Concepts
 - Analysis of Tree Rotations
 - Depth-First-Alternating Traversal
 - Address Conflict Manager
- Evaluation
 - Experimental Setup
 - Experimental Results
- Conclusion

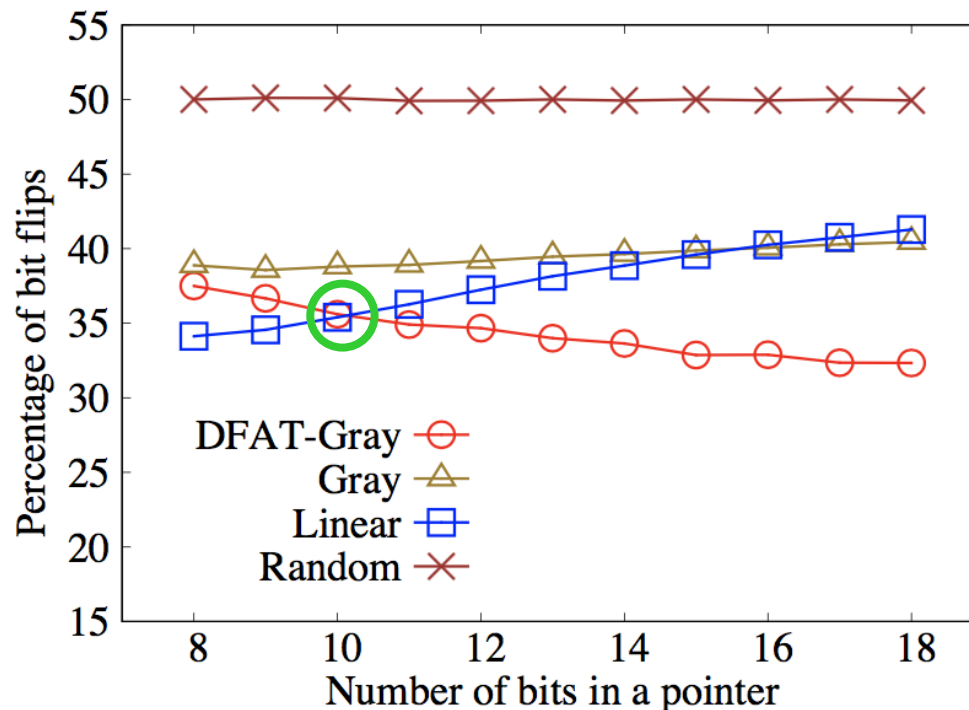
Result of Solutions with Different Data Sizes (1/2)

- *Linear* : is the best solution when the memory space $\leq 2^{10}$.
- *DFAT-Gray* : outperforms the other solutions when the memory space $> 2^{10}$.



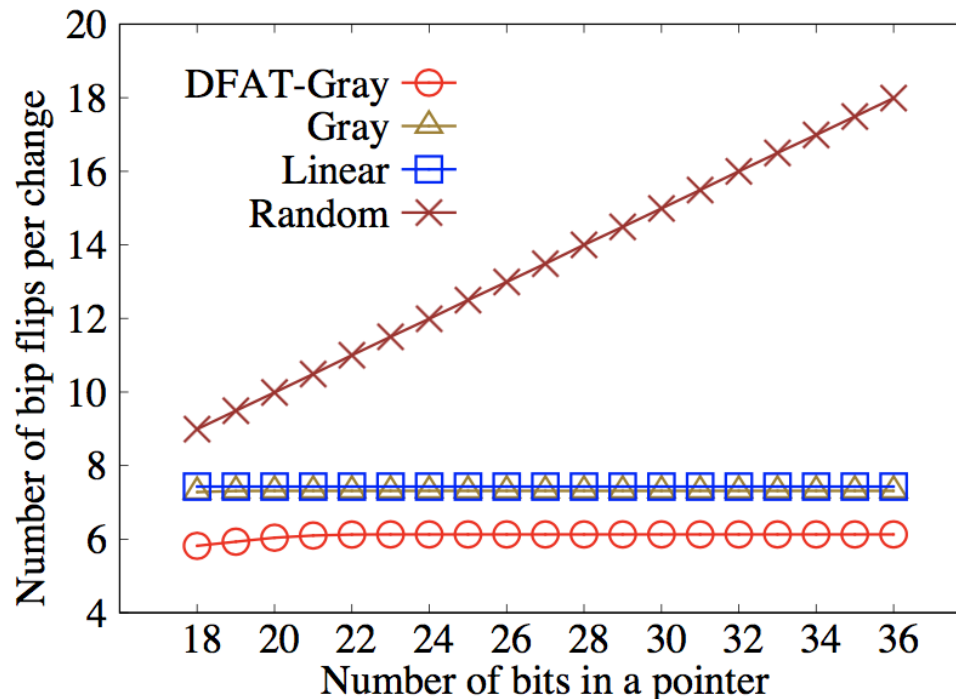
Result of Solutions with Different Data Sizes (2/2)

- The percentage of bit flips = $\frac{\text{number of bit bit flips per write}}{\text{number of bits in a pointer}}$
- When the number of bits in a pointer increases
 - The bit-flip ratios of *Linear* and *Gray* increase.
 - The bit-flip ratio of *DFAT-Gray* decreases noticeably.



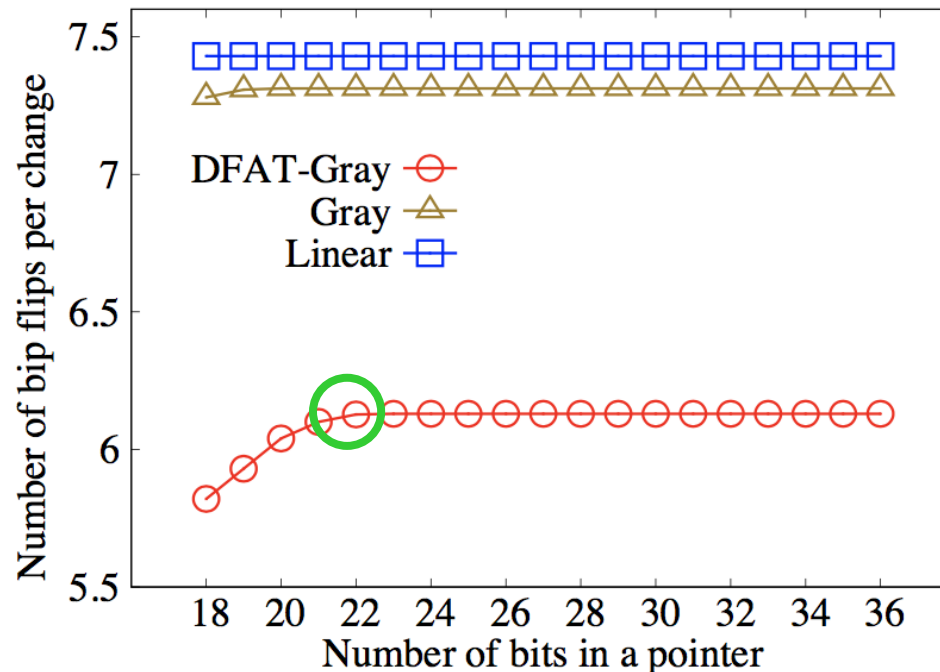
Result of Simulation for Huge Memory (1/2)

- We consider $2^{18}-1 < \text{memory space} < 2^{36}-1$, but the data size is fixed at 2^{16} for fast simulation.
- *Random* significantly increases the number of bit flips per change along with the memory space getting larger.



Result of Simulation for Huge Memory (2/2)

- *Gray* and *Linear*: have a stable performance with the fixed input data size
- *DFAT-Gray*: increases its bit flips ratio slightly when the memory space increases from 2^{18} to 2^{22} , and then gets saturated.



Outline

- Introduction
- Background and Motivation
- Write-Asymmetry-Aware Self-Balance Tree
 - Basic Concepts
 - Analysis of Tree Rotations
 - Depth-First-Alternating Traversal
 - Address Conflict Manager
- Evaluation
 - Experimental Setup
 - Experimental Results
- Conclusion

Conclusion

- We redesign the memory allocation scheme of a self-balancing binary search tree for PCM.
- *DFAT-Gray* on AVL trees can reduce more than **15%** of bit flips when the size of the input data is more than $2^{15} - 1$ nodes.
- Further extending our concepts to other relative data structures is our ongoing studies.