Optimizing Systems for Byte-Addressable NVM by Reducing Bit Flipping

Daniel Bittman Peter Alvaro Darrell D. E. Long Ethan L. Miller

Center for Research In Storage Systems
University of California, Santa Cruz

FAST '19
2019-02-26

Byte-addressable Non-volatile Memory

Intel Launches Optane DIMMs Up To 512GB: Apache Pass Is Here!

by Ian Cutress & Billy Tallis on May 30, 2018 2:15 PM EST

BNVM is coming, and with it, new optimization targets

Byte-addressable Non-volatile Memory

Intel Launches Optane DIMMs Up To 512GB: Apache Pass Is Here!

by Ian Cutress & Billy Tallis on May 30, 2018 2:15 PM EST

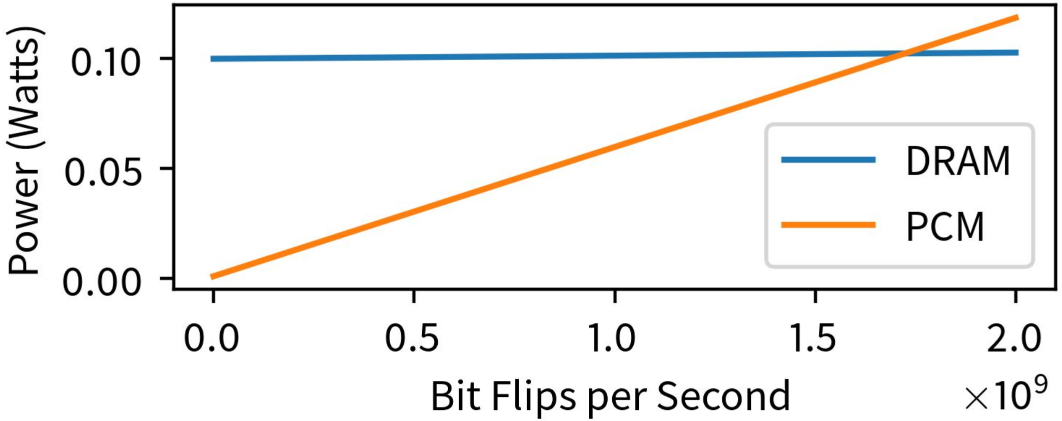
It's not *just* writes...

0	1	0	1	0	0	0	1

0	1	**1**	**0**	0	**1**	0	1

...it's the bits *flipped* by those writes

BNVM power usage



PCM power consumption scales with bits flipped

DRAM has refresh cost that PCM does not have

Can we take advantage of this?

Software vs. hardware?

Can we take advantage of this?

Software vs. hardware?

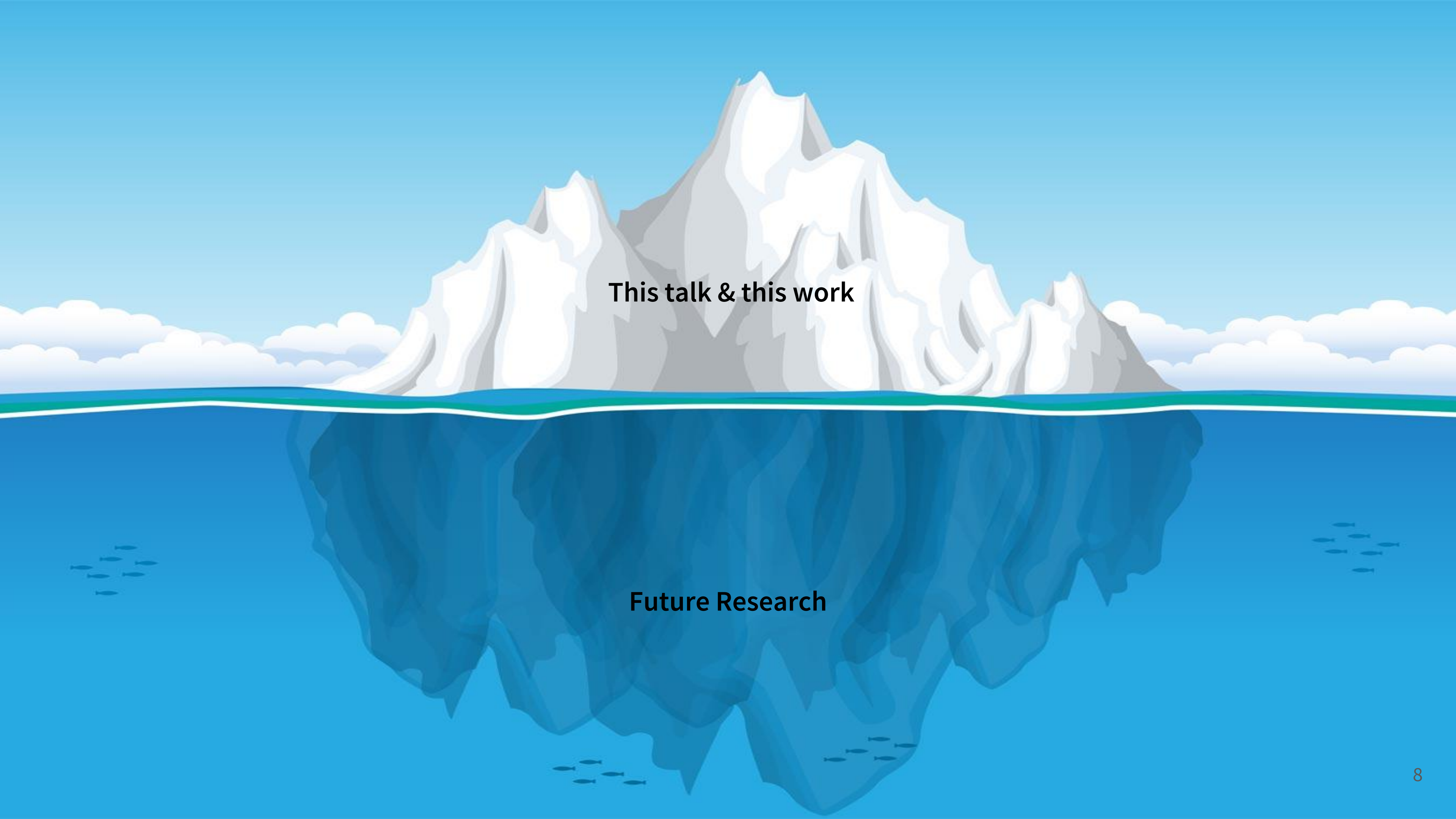
How hard is it to reason about bit flips?

Can we take advantage of this?

Software vs. hardware?

How hard is it to reason about bit flips?

How do we *design* data structures to reduce bit flips?

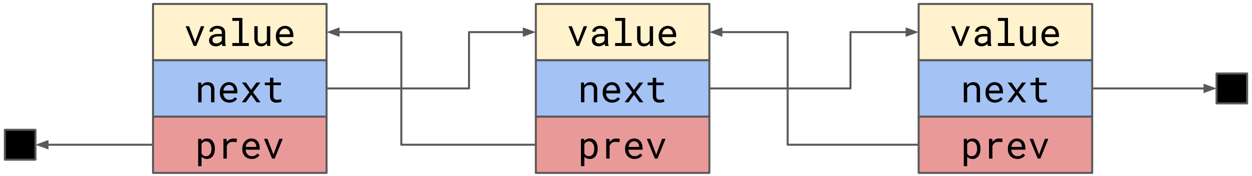

This talk & this work
Future Research

Reducing Bit Flips in Software

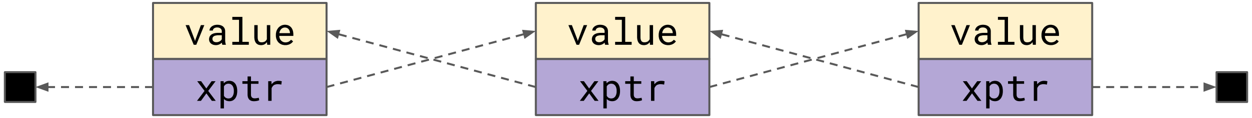
- XOR linked lists
- Red black trees
- Hash tables

XOR linked lists

Traditional doubly linked list



XOR linked list



xptr = next ⊕ prev

Pointers!

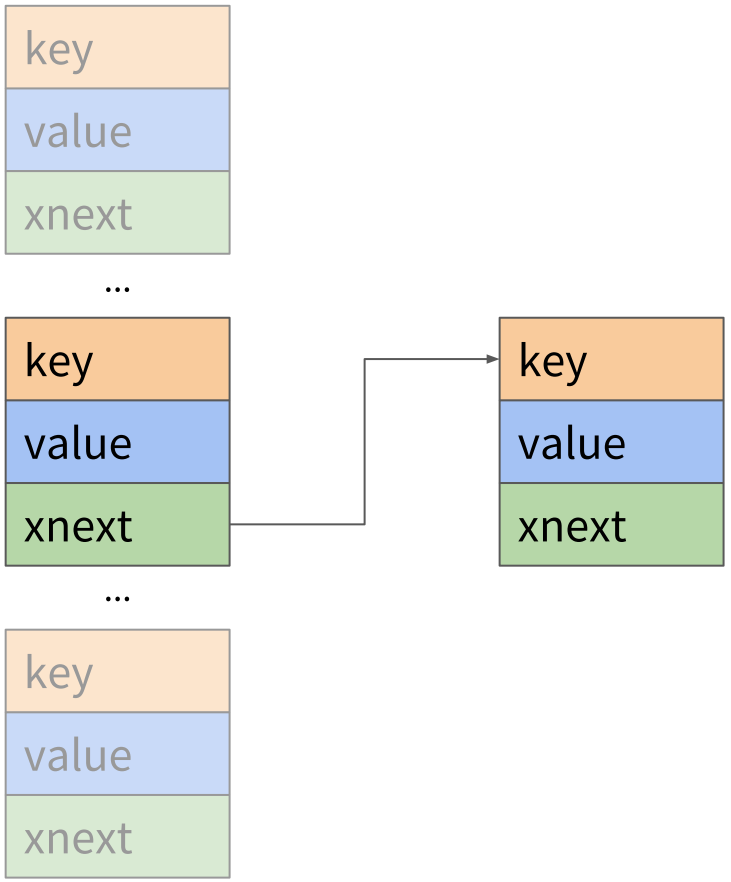
<u>Some actual pointers</u>

A = 0x000055b7bda8f260

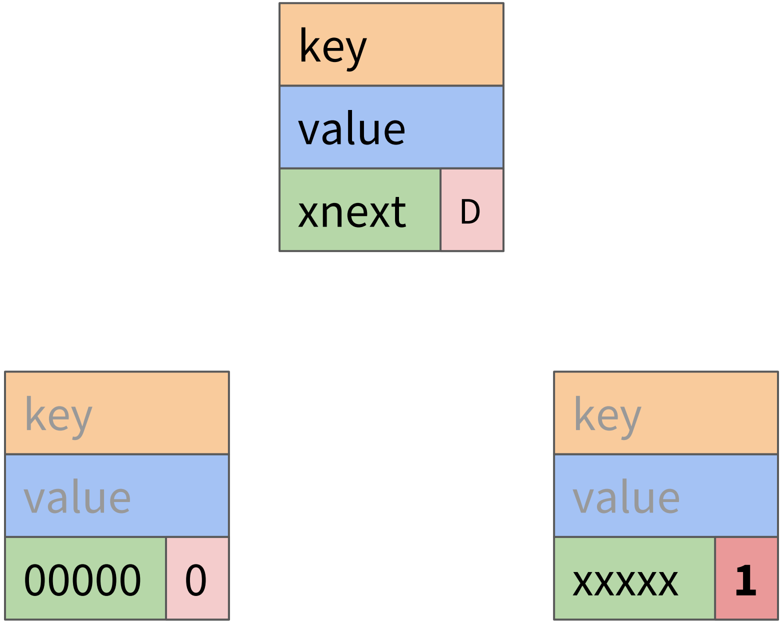
B = 0x000055b7bda8f6a0

$A \oplus B$ = 0x4C0 = 0b10011000000

Using XOR in hash tables



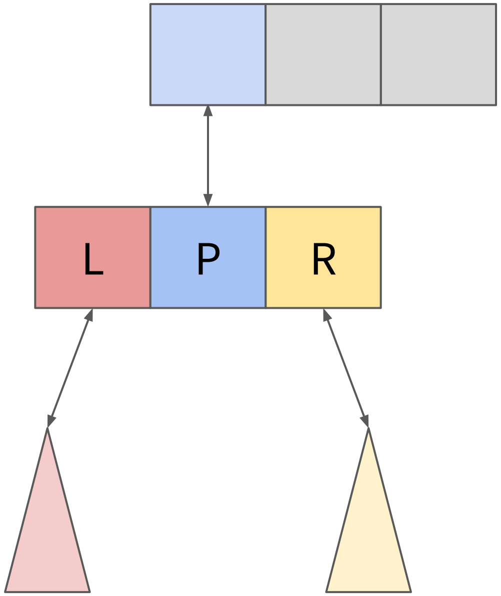
Using XOR in hash tables



Both indicate “entry is empty”

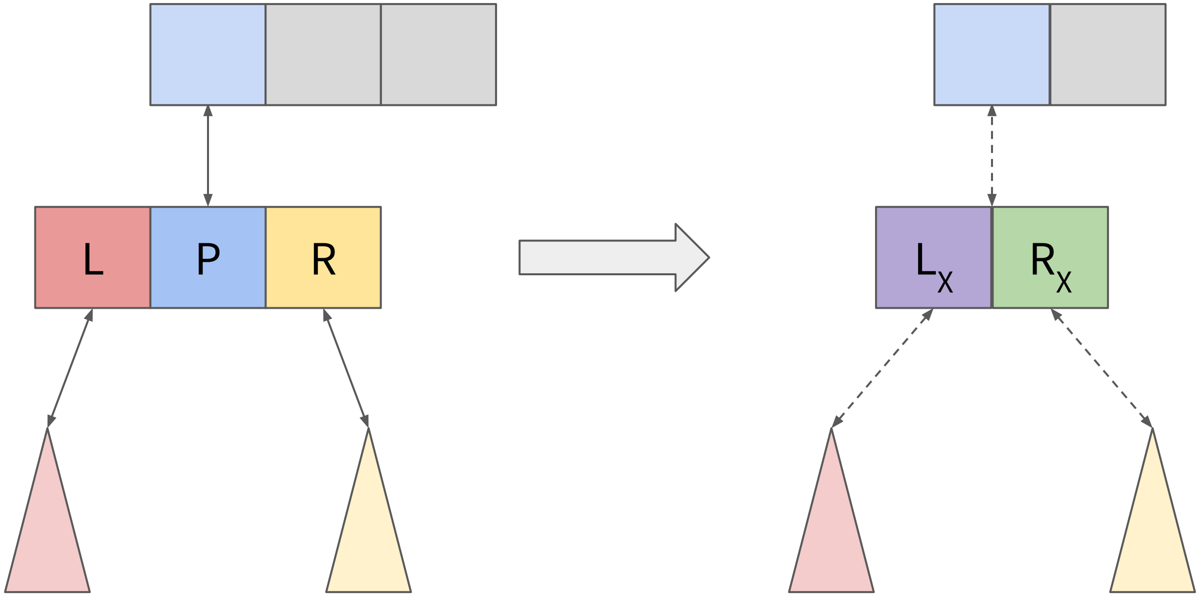
Saves bitflips during overwrites

From XOR linked lists to Red Black Trees



Standard 3-pointer red-black tree design

From XOR linked lists to Red Black Trees



$R_X = R \oplus P$

$L_X = L \oplus P$

Now 2-pointer, and XOR pointers

Evaluation

- Determine bit flip characteristics
- Measure performance impact

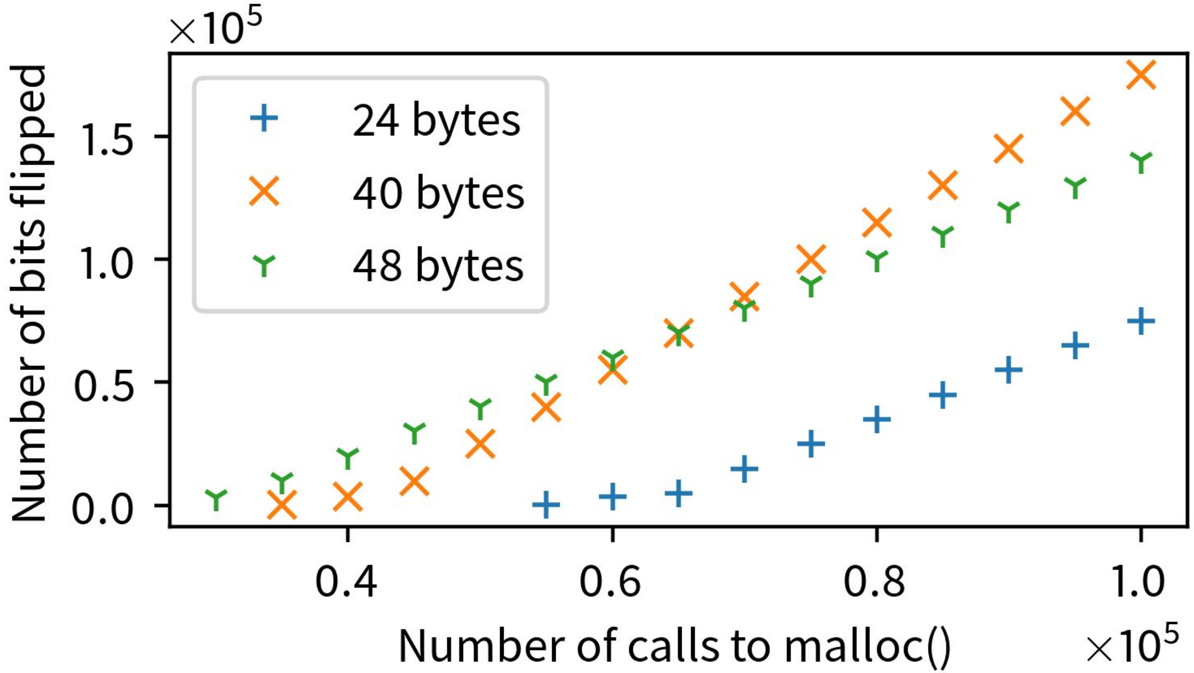
Experimental framework

CENTER FOR RESEARCH IN STORAGE SYSTEMS

Full system simulator, patched to support bitflip counting at the memory controller

Test different data structures, with different cache parameters, over a varying number of operations

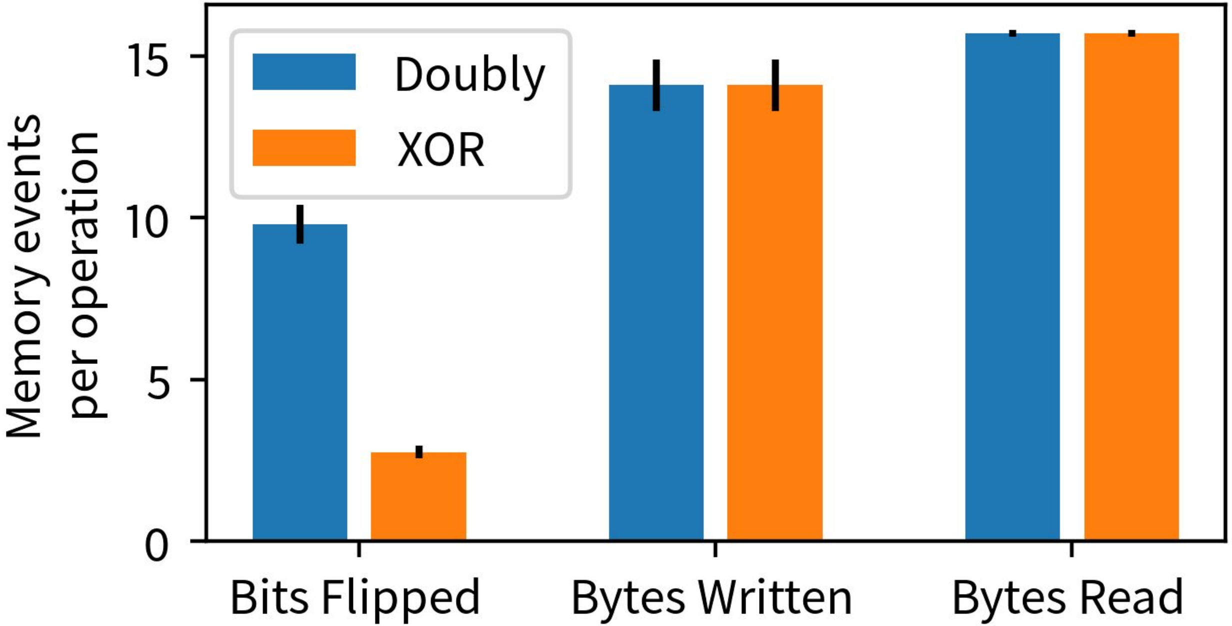
Bit flips: calling malloc()



Trends often become linear

Cross-over point between 40 and 48 bytes

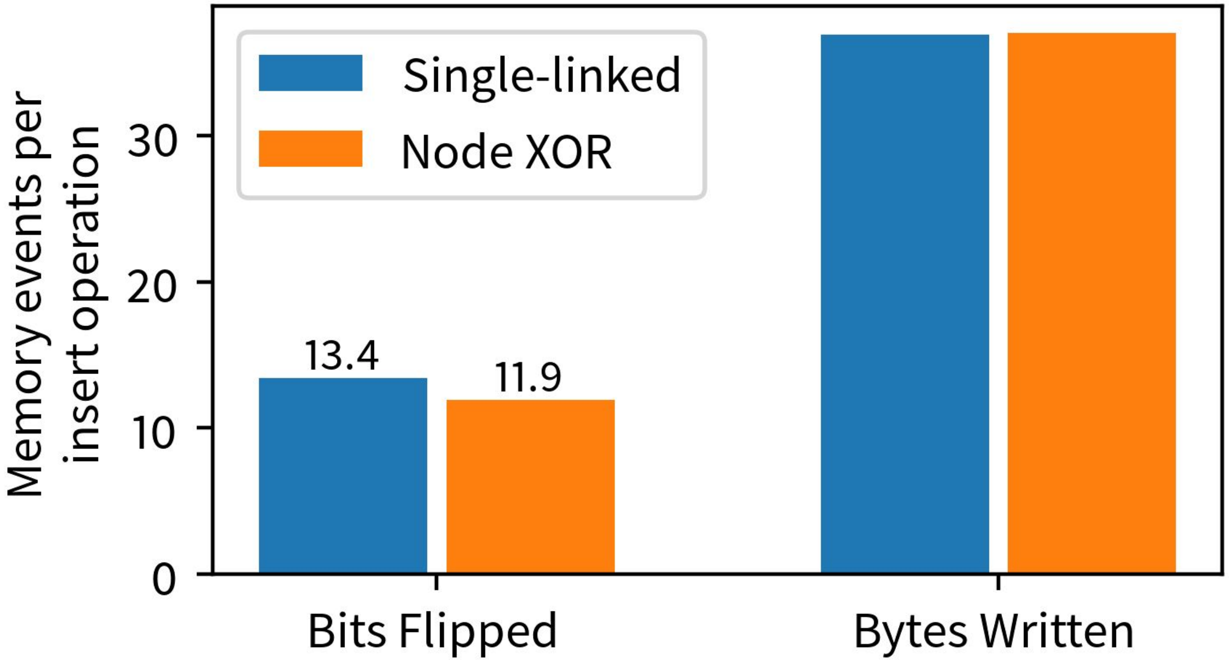
Bit flips: XOR Linked Lists



XOR linked lists reduce bit flips dramatically

better

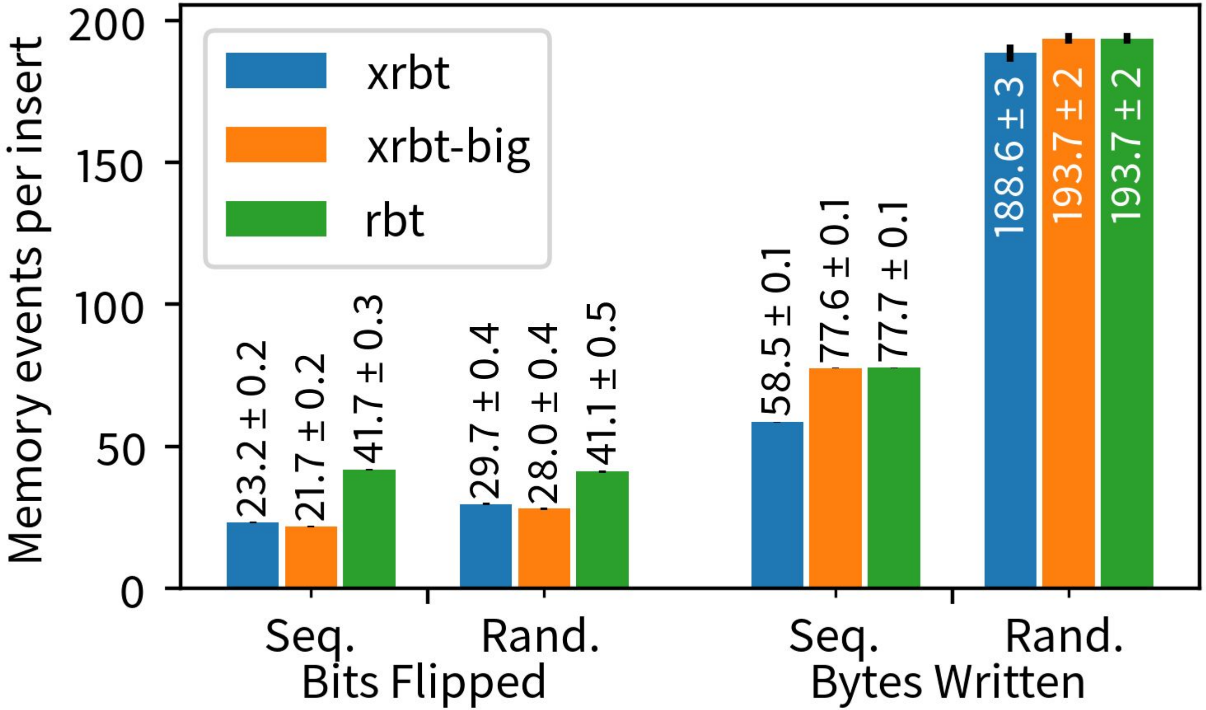
Bit flips: Hash table



Hash table already had few flips to save: chains should be short

better

Bit flips: Red-black Trees

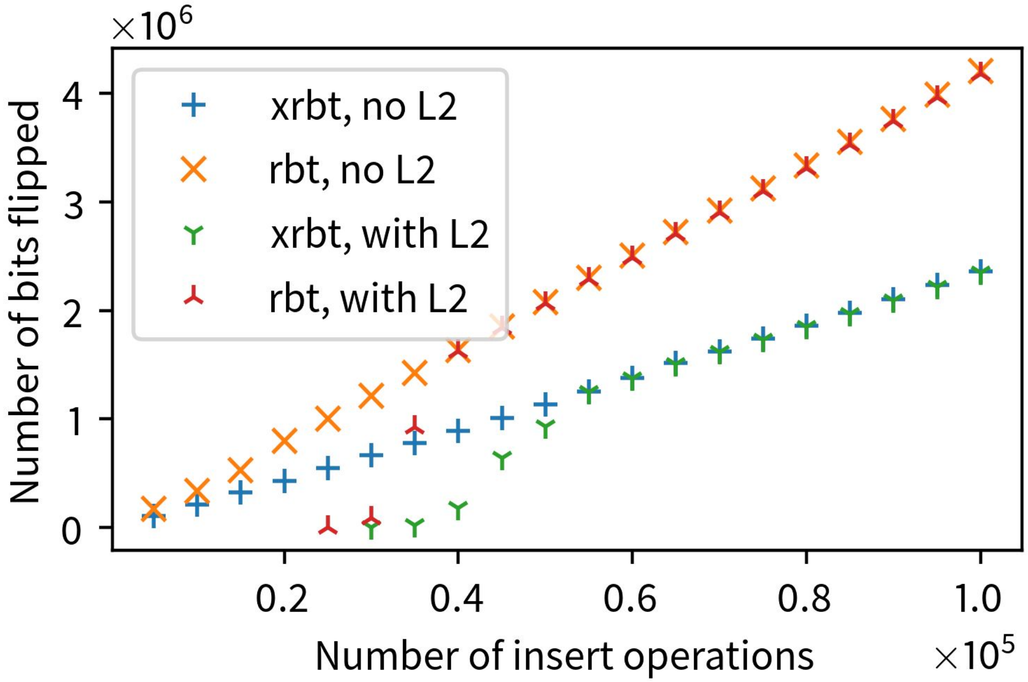


Write/bitflip inversion!

Significant savings

better

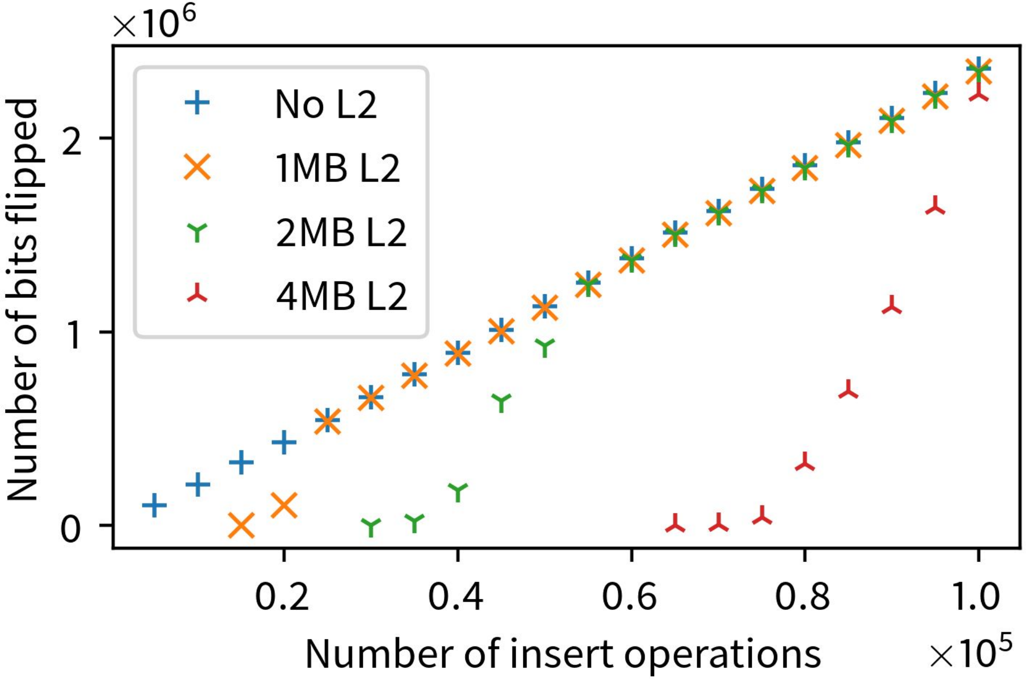
Bit flips: L2 Cache Behavior



L2 cache has ultimately little effect!

better

Bit flips: L2 Cache Behavior

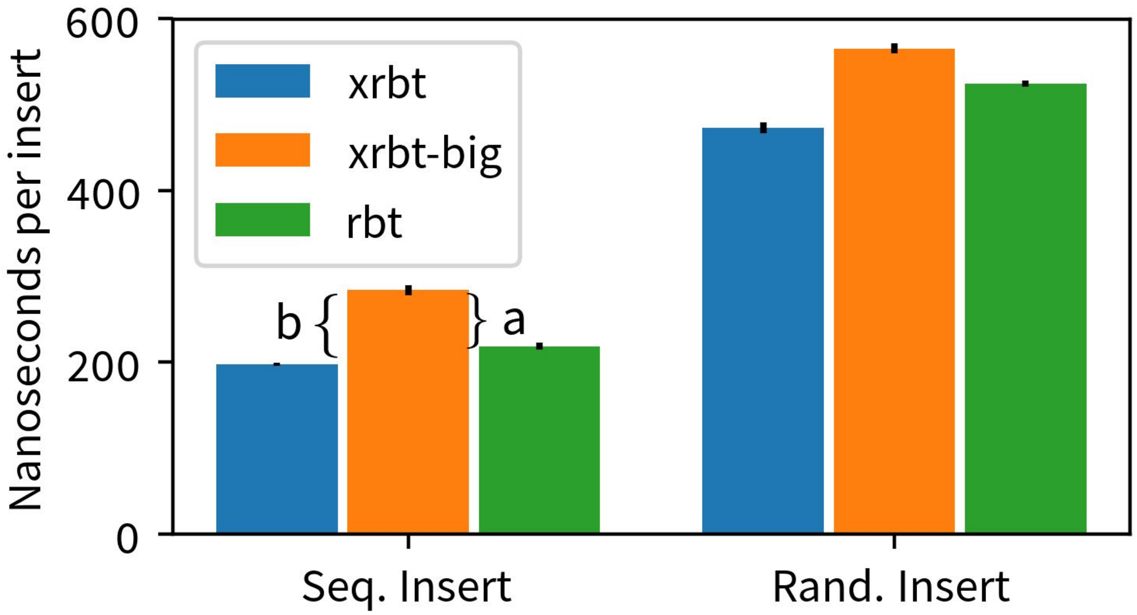


L2 cache has ultimately little effect!

...even when increasing in size

better

Performance: RBT insert

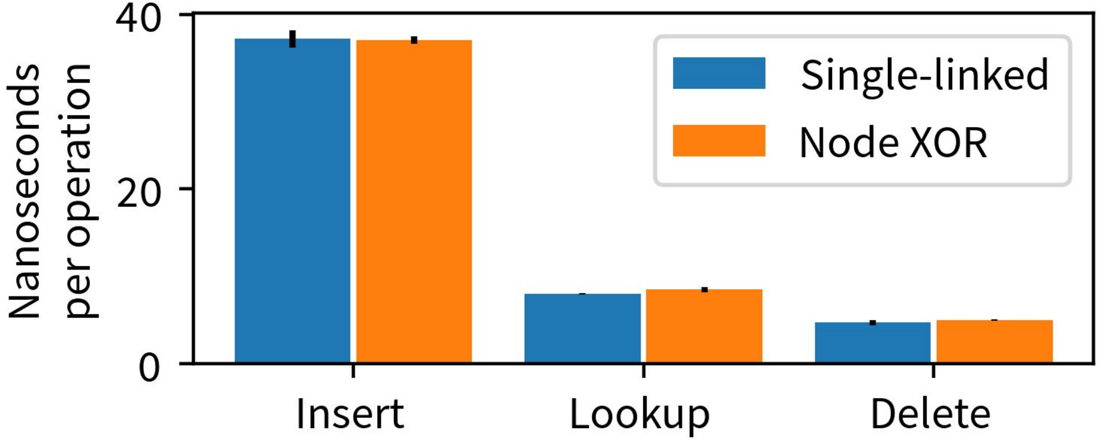


Performance is not significantly affected!

better

a) Performance cost of XORs
b) Performance benefit of smaller node size

Performance: hash table



Almost no effect on performance

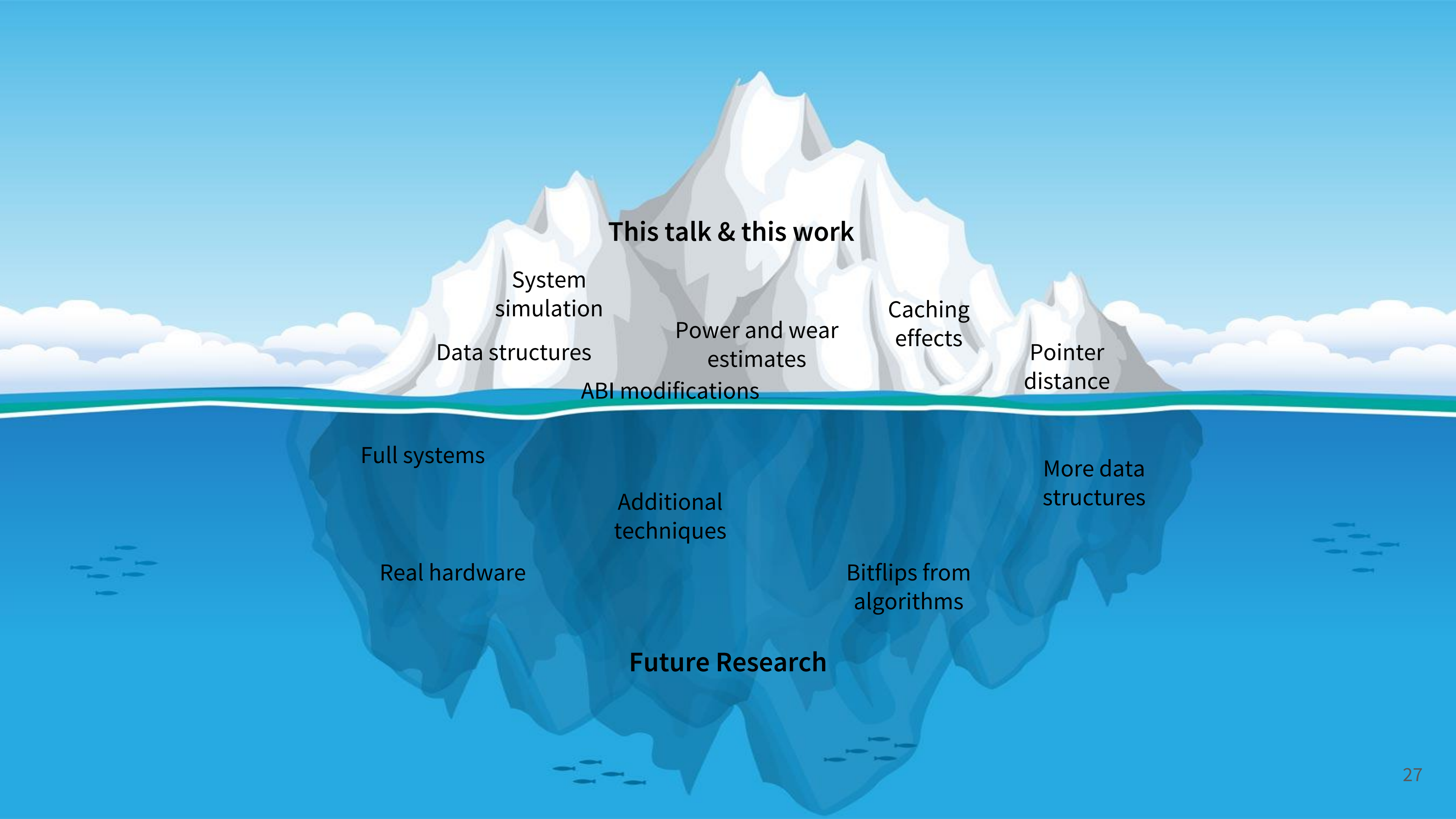
better

Conclusions

Savings are significant with little performance impact.

We *can* design around bit flips, and we *should.*

Bit flip/write inversion


This talk & this work
System simulation
Data structures
Power and wear estimates
Caching effects
Pointer distance
ABI modifications
Full systems
More data structures
Additional techniques
Real hardware
Bitflips from algorithms
Future Research

Center for
Research in
Storage
Systems

Thank You! Questions?

Daniel Bittman
@danielbittman
dbittman@ucsc.edu

Peter Alvaro
palvaro@ucsc.edu

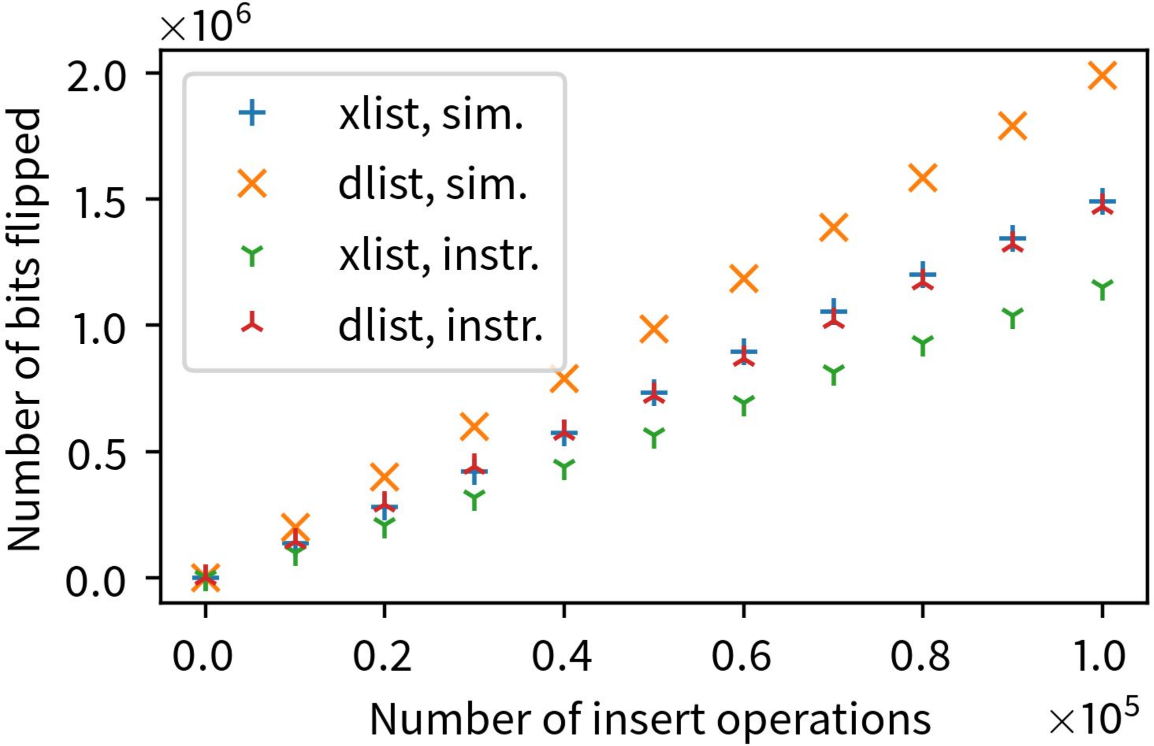
Darrell Long
darrell@ucsc.edu

Ethan Miller
elm@ucsc.edu

`https://gitlab.soe.ucsc.edu/gitlab/crss/opensource-bitflipping-fast19`

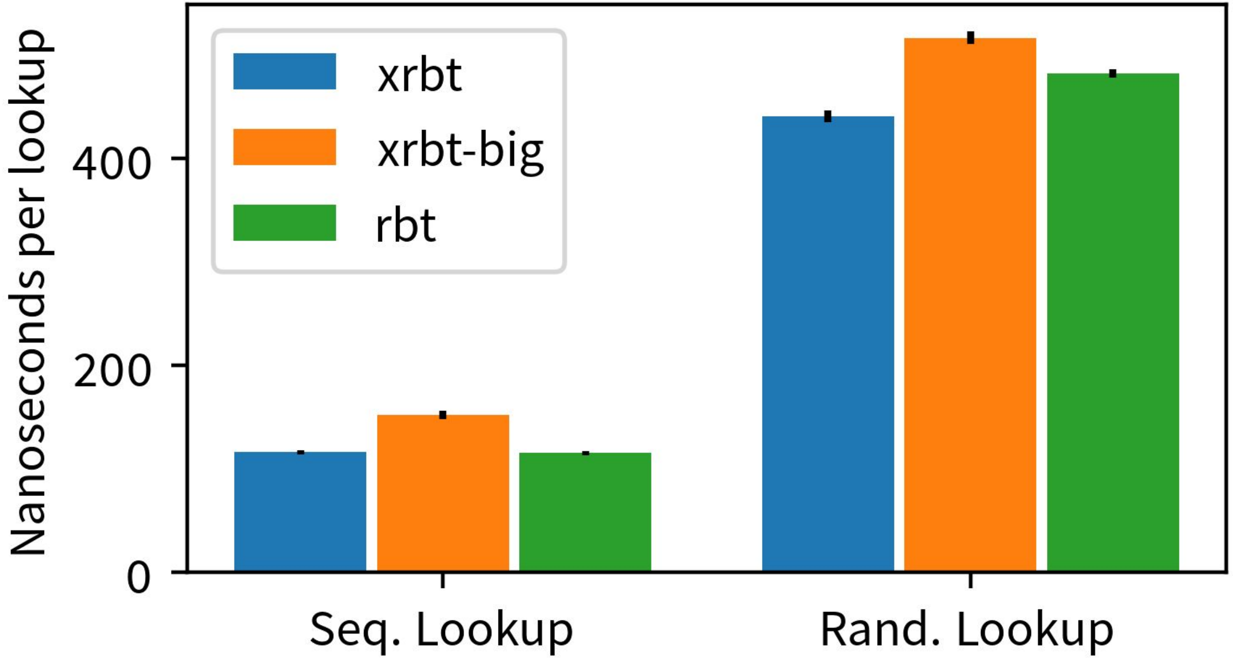
Backup slides

Bit flips: instrumentation



better

Performance: RBT lookup

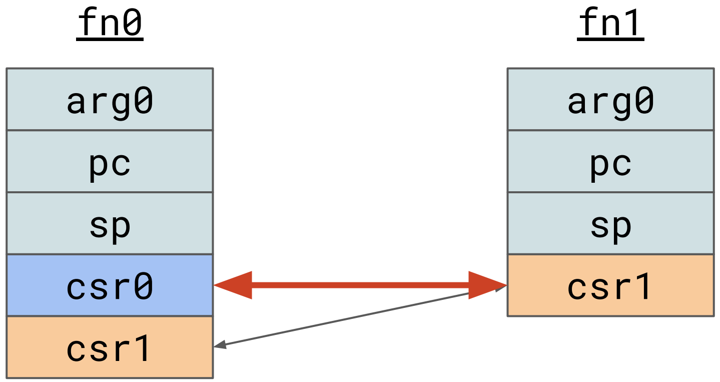


better

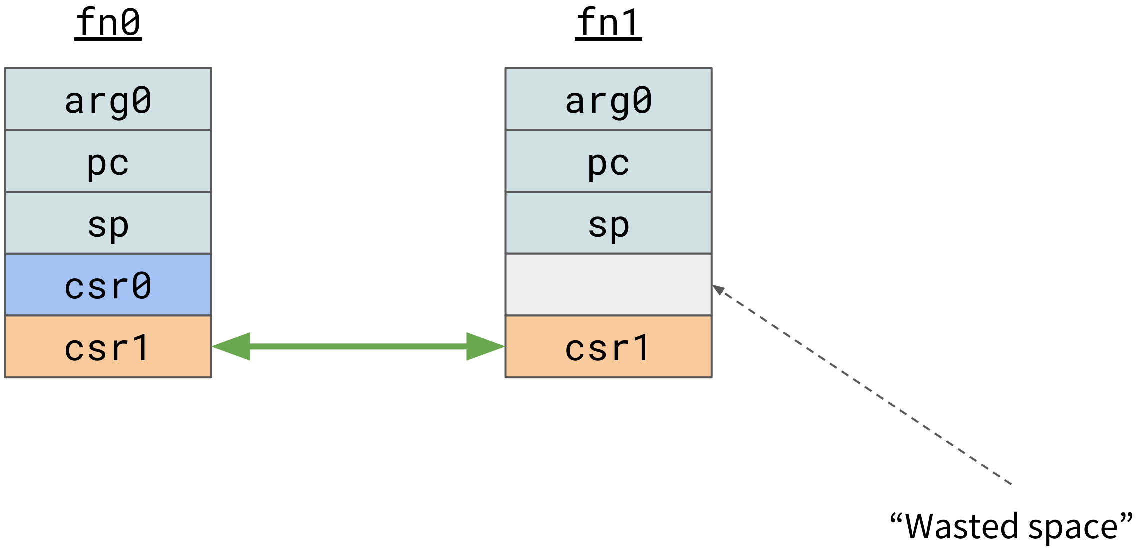
Performance: XOR Linked List

Operation	XOR Linked List	Doubly Linked List
Insert (ns)	45 +/- 1	45 +/- 1
Pop (ns)	27 +/- 1	28 +/- 1
Traverse (ns/node)	2.6 +/ 0.1	2.2 +/- 0.1

Stack frames



Stack frames



Bit flips: stack frames