Practical Hardware Transactional vEB Trees

PPoPP'24

Mohammad Khalaji [1] Trevor Brown[1] Khuzaima Daudjee[1] Vitaly Aksenov[2]

[1] 

[2] 

Introduction

Sets and Maps

DS	Look-up	Insert	Delete	Predecessor	Successor
Hash Table	$\ddot{e}(1)$	$\ddot{e}(1)$	$\ddot{e}(1)$	$\ddot{e}(\acute{Y})$	$\ddot{e}(\acute{Y})$
BST	$\ddot{e}(\lg \acute{Y})$	$\ddot{e}(\lg \acute{Y})$	$\ddot{e}(\lg \acute{Y})$	$\ddot{e}(\mathbf{lg}\ \acute{Y})$	$\ddot{e}(\mathbf{lg}\ \acute{Y})$
B-Tree	$\ddot{e}(\log_{@} \acute{Y})$	$\ddot{e}(\log_{@} \acute{Y})$	$\ddot{e}(\log_{@} \acute{Y})$	$\ddot{e}(\mathbf{log}_{@}\ \acute{Y})$	$\ddot{e}(\mathbf{log}_{@}\ \acute{Y})$

Sets and Maps

DS	Look-up	Insert	Delete	Predecessor	Successor
Hash Table	$\ddot{e}(1)$	$\ddot{e}(1)$	$\ddot{e}(1)$	$\ddot{e}(\acute{Y})$	$\ddot{e}(\acute{Y})$
BST	$\ddot{e}(\lg \acute{Y})$	$\ddot{e}(\lg \acute{Y})$	$\ddot{e}(\lg \acute{Y})$	$\ddot{e}(\mathbf{lg}\ \acute{Y})$	$\ddot{e}(\mathbf{lg}\ \acute{Y})$
B-Tree	$\ddot{e}(\log_{@} \acute{Y})$	$\ddot{e}(\log_{@} \acute{Y})$	$\ddot{e}(\log_{@} \acute{Y})$	$\ddot{e}(\mathbf{log}_{@}\ \acute{Y})$	$\ddot{e}(\mathbf{log}_{@}\ \acute{Y})$
vEB Tree	$\ddot{e}(\lg \lg \cdot)$	$\ddot{e}(\lg \lg \cdot)$	$\ddot{e}(\lg \lg \cdot)$	$\ddot{e}(\mathbf{lg}\ \mathbf{lg}\ \cdot)$	$\ddot{e}(\mathbf{lg}\ \mathbf{lg}\ \cdot)$

Sets and Maps

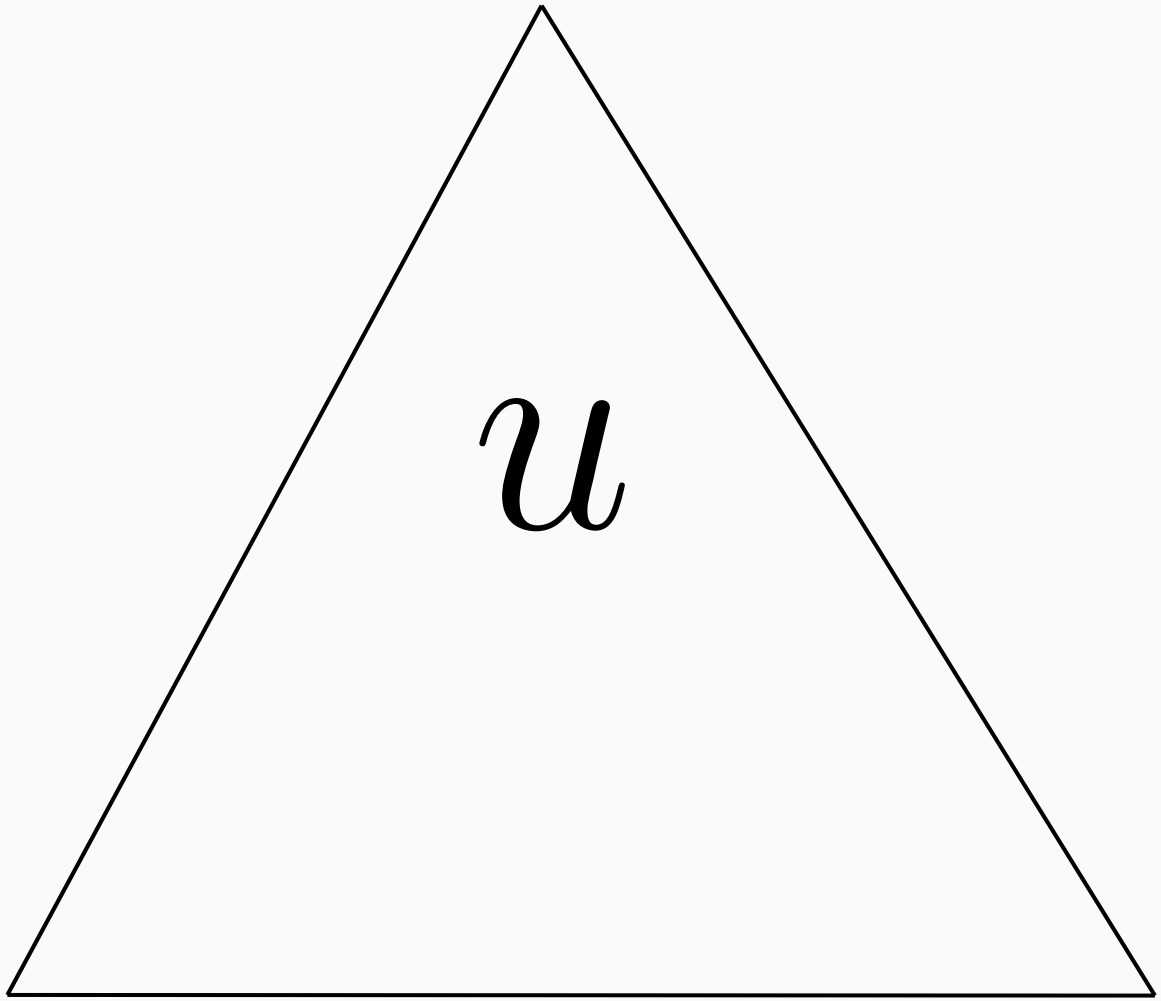
DS	Look-up	Insert	Delete	Predecessor	Successor
Hash Table	$\ddot{e}(1)$	$\ddot{e}(1)$	$\ddot{e}(1)$	$\ddot{e}(\acute{Y})$	$\ddot{e}(\acute{Y})$
BST	$\ddot{e}(\lg \acute{Y})$	$\ddot{e}(\lg \acute{Y})$	$\ddot{e}(\lg \acute{Y})$	$\ddot{e}(\mathbf{lg}\ \acute{Y})$	$\ddot{e}(\mathbf{lg}\ \acute{Y})$
B-Tree	$\ddot{e}(\log_{@} \acute{Y})$	$\ddot{e}(\log_{@} \acute{Y})$	$\ddot{e}(\log_{@} \acute{Y})$	$\ddot{e}(\mathbf{log}_{@}\ \acute{Y})$	$\ddot{e}(\mathbf{log}_{@}\ \acute{Y})$
vEB Tree	$\ddot{e}(\lg \lg \acute{\iota})$	$\ddot{e}(\lg \lg \acute{\iota})$	$\ddot{e}(\lg \lg \acute{\iota})$	$\ddot{e}(\mathbf{lg}\ \mathbf{lg}\ \acute{\iota})$	$\ddot{e}(\mathbf{lg}\ \mathbf{lg}\ \acute{\iota})$

$\acute{\iota}$	$\lg \lg \acute{\iota}$
65K	4
1M	4;3
1B	4;9
1T	5;3

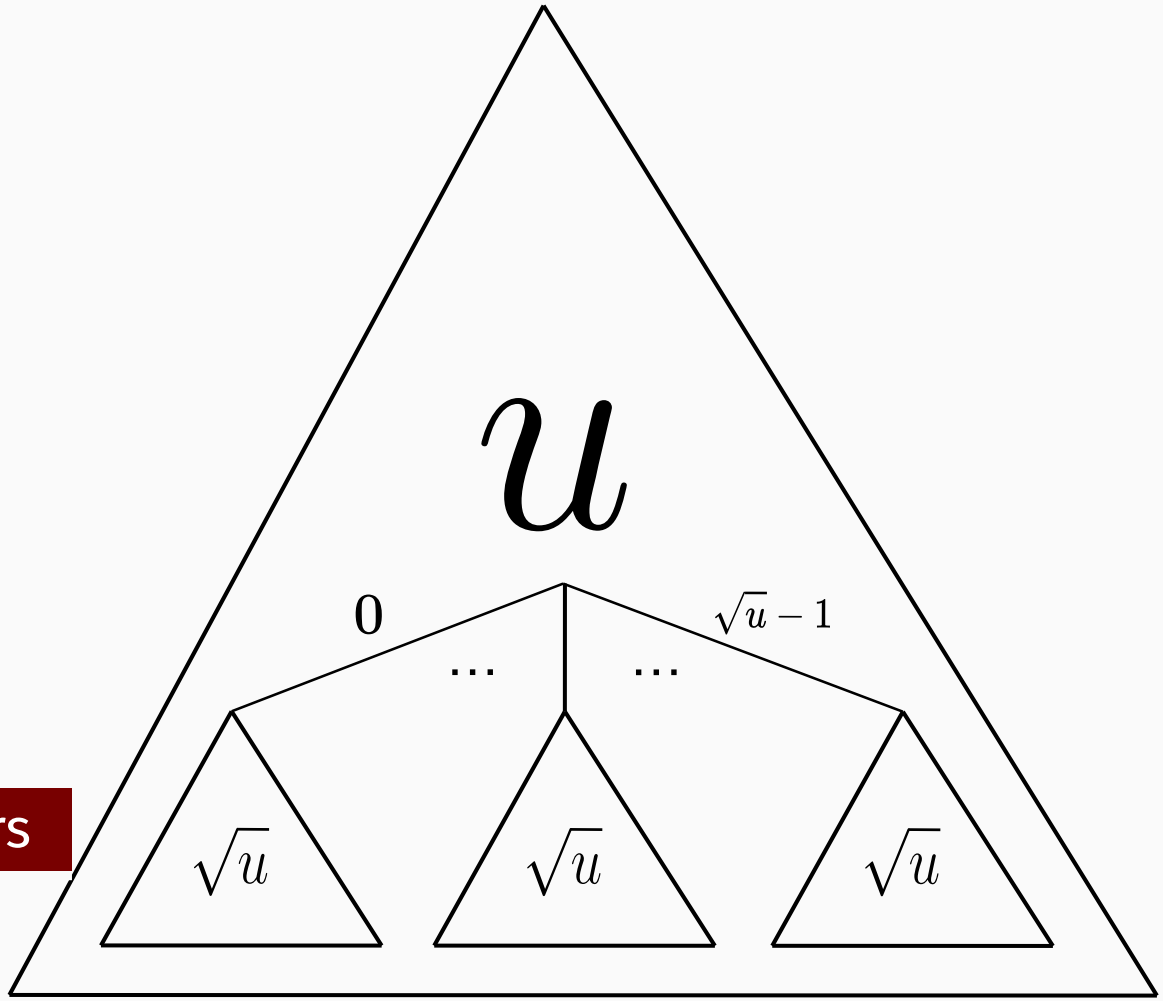
vEB Trees

- Introduced in 1977 by Peter van Emde Boas [4]
- Fixed universe size $\iota = 2^{\lg \iota}$: $f0; : : : ; \iota \quad 1g \rightarrow \lg \iota$ bit keys
- $\ddot{e}(\lg \lg \iota)$ complexity for all operations

vEB Tree Structure

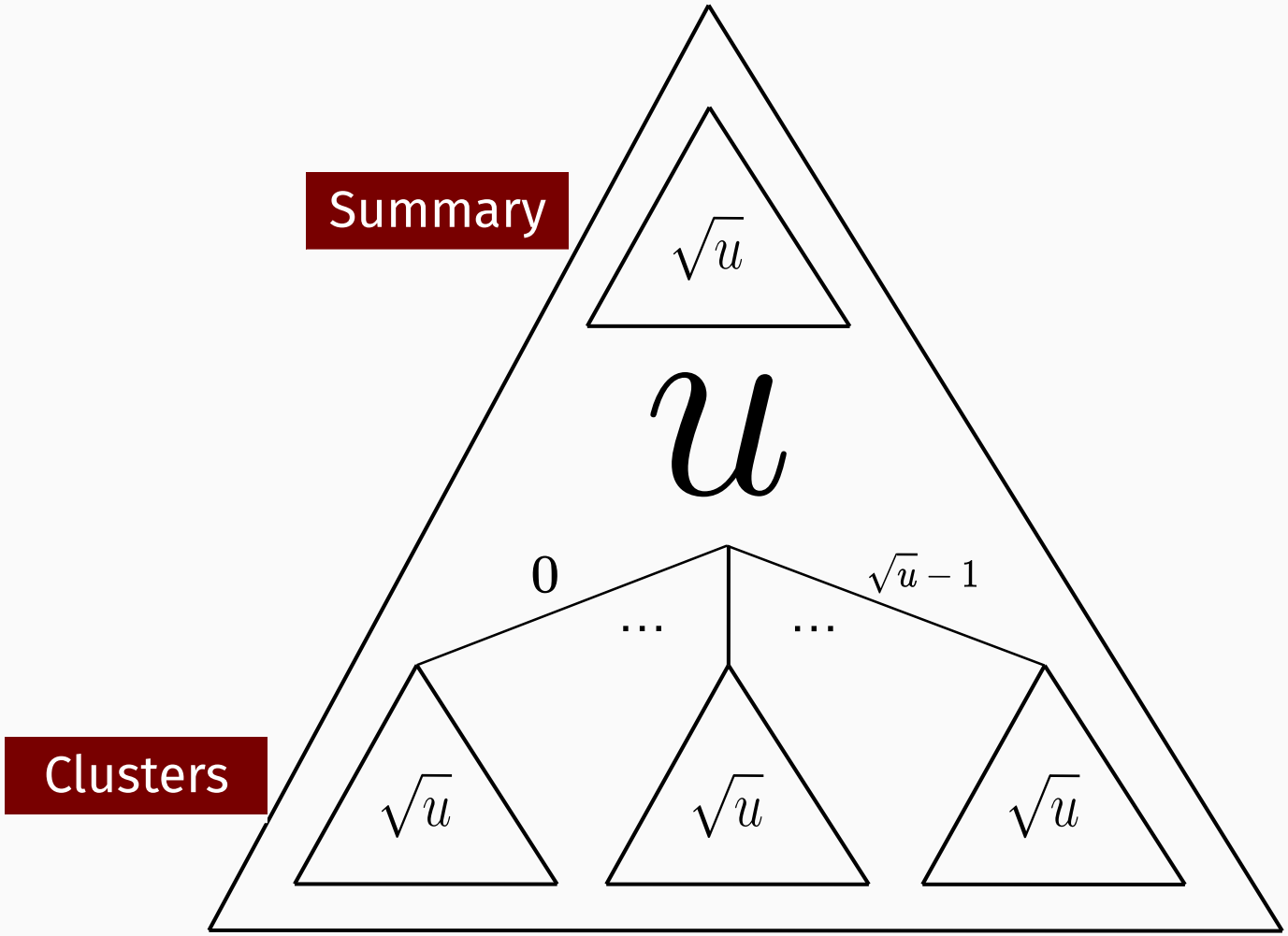


vEB Tree Structure



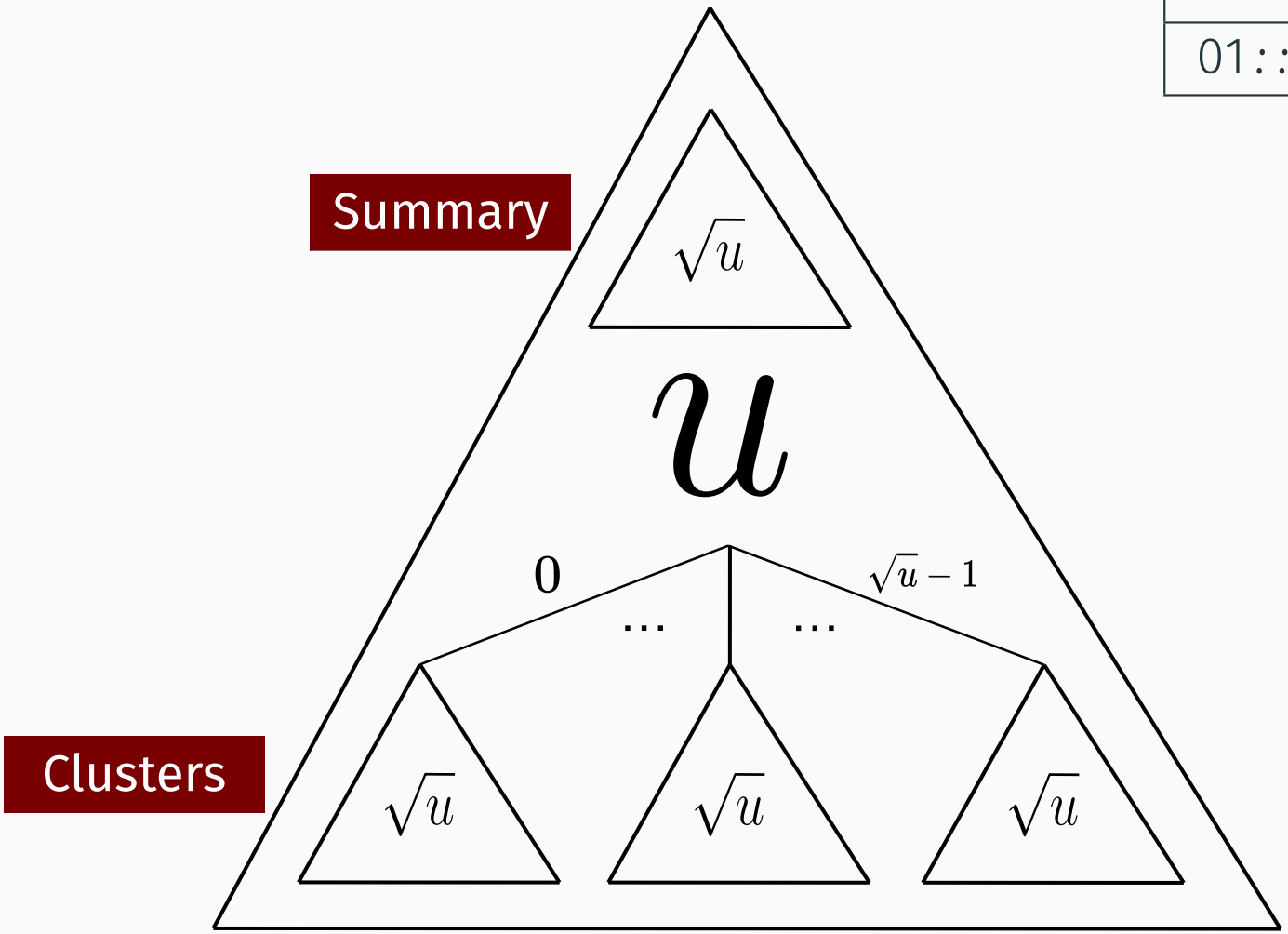
Clusters

vEB Tree Structure

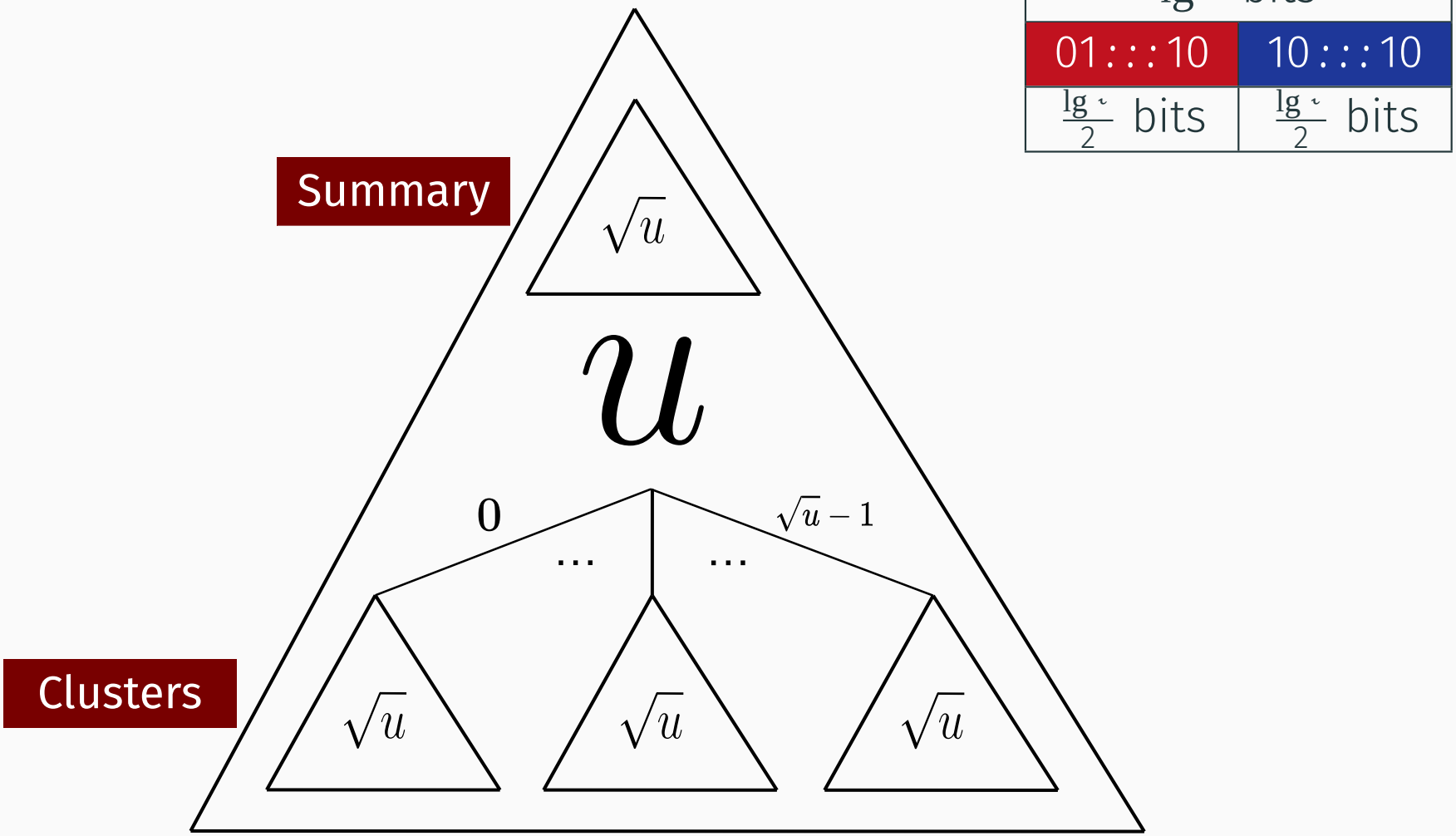


vEB Insert Example

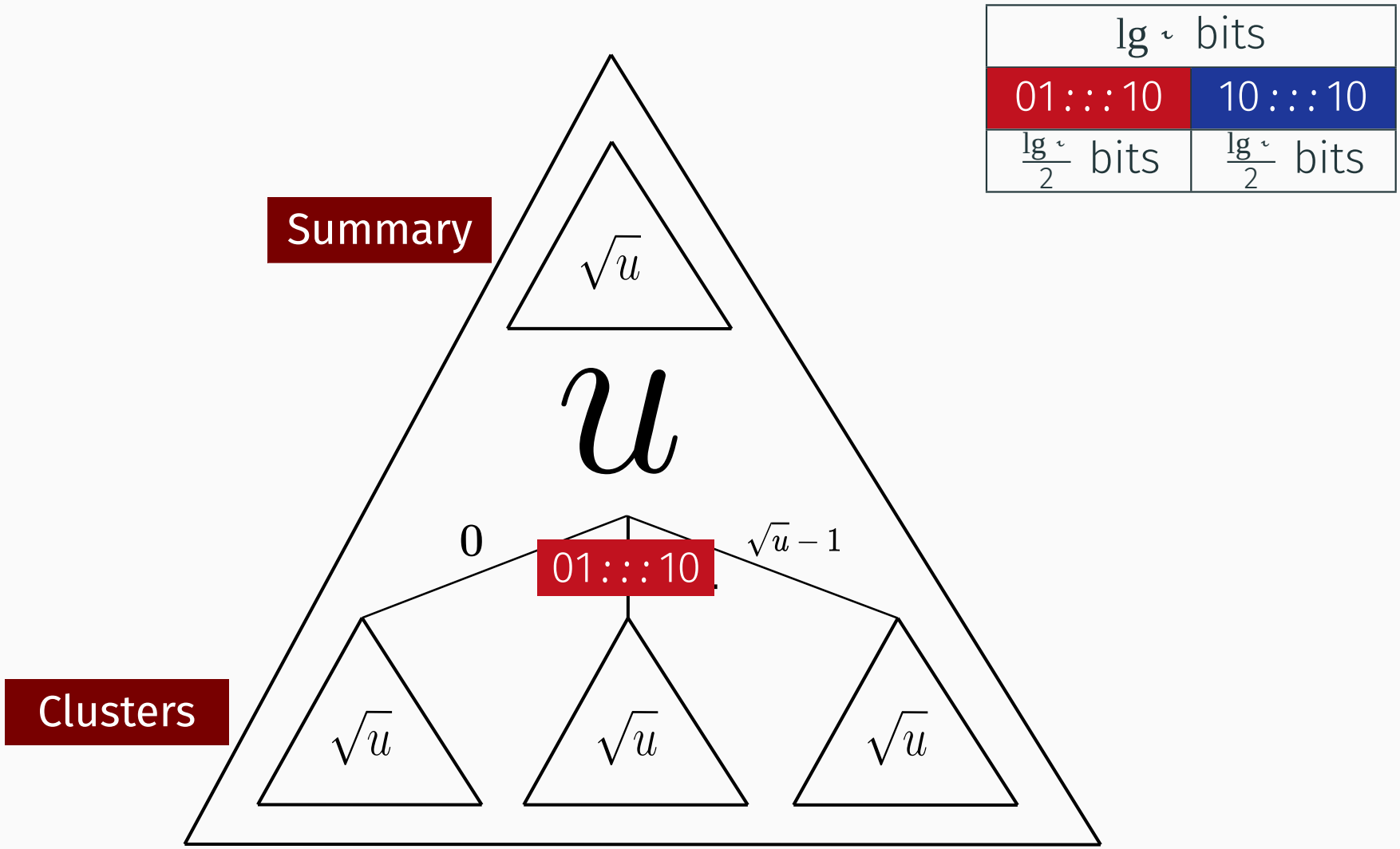
$\lg u$ bits
01 : : : : : : : : : : : : : 10



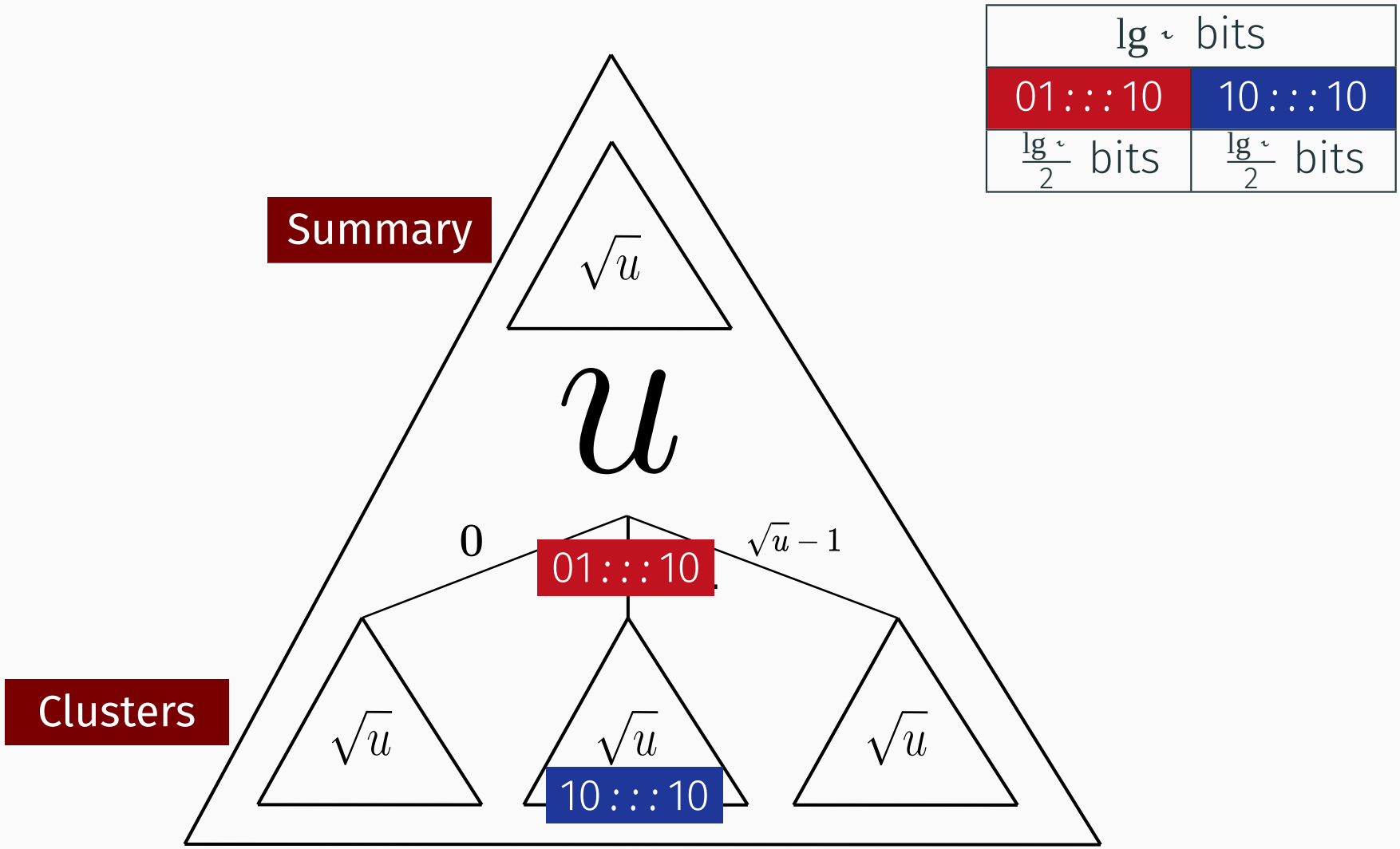
vEB Insert Example



vEB Insert Example

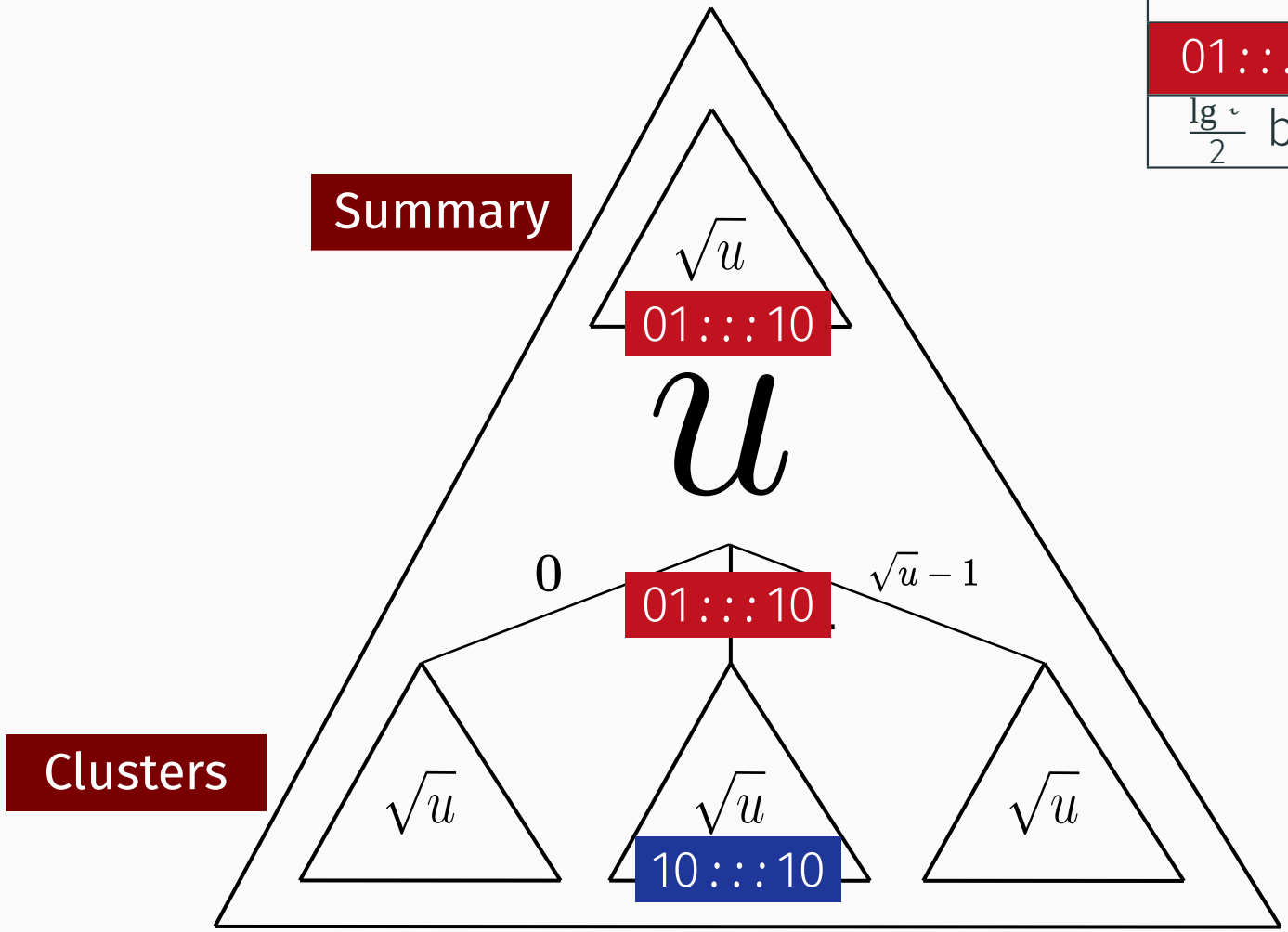


vEB Insert Example

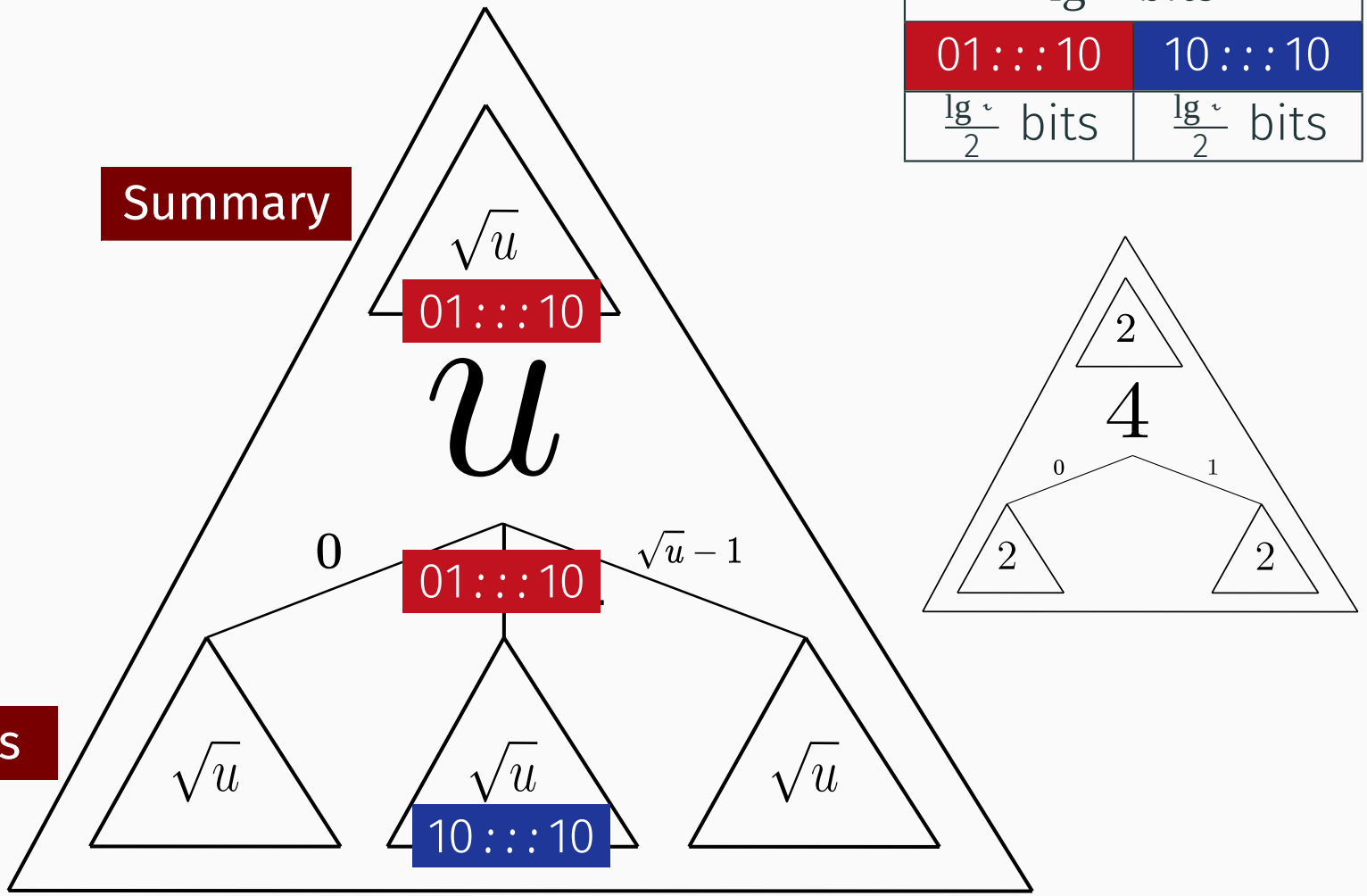


vEB Insert Example

$\lg u$ bits	
01 : : : 10	10 : : : 10
$\frac{\lg u}{2}$ bits	$\frac{\lg u}{2}$ bits

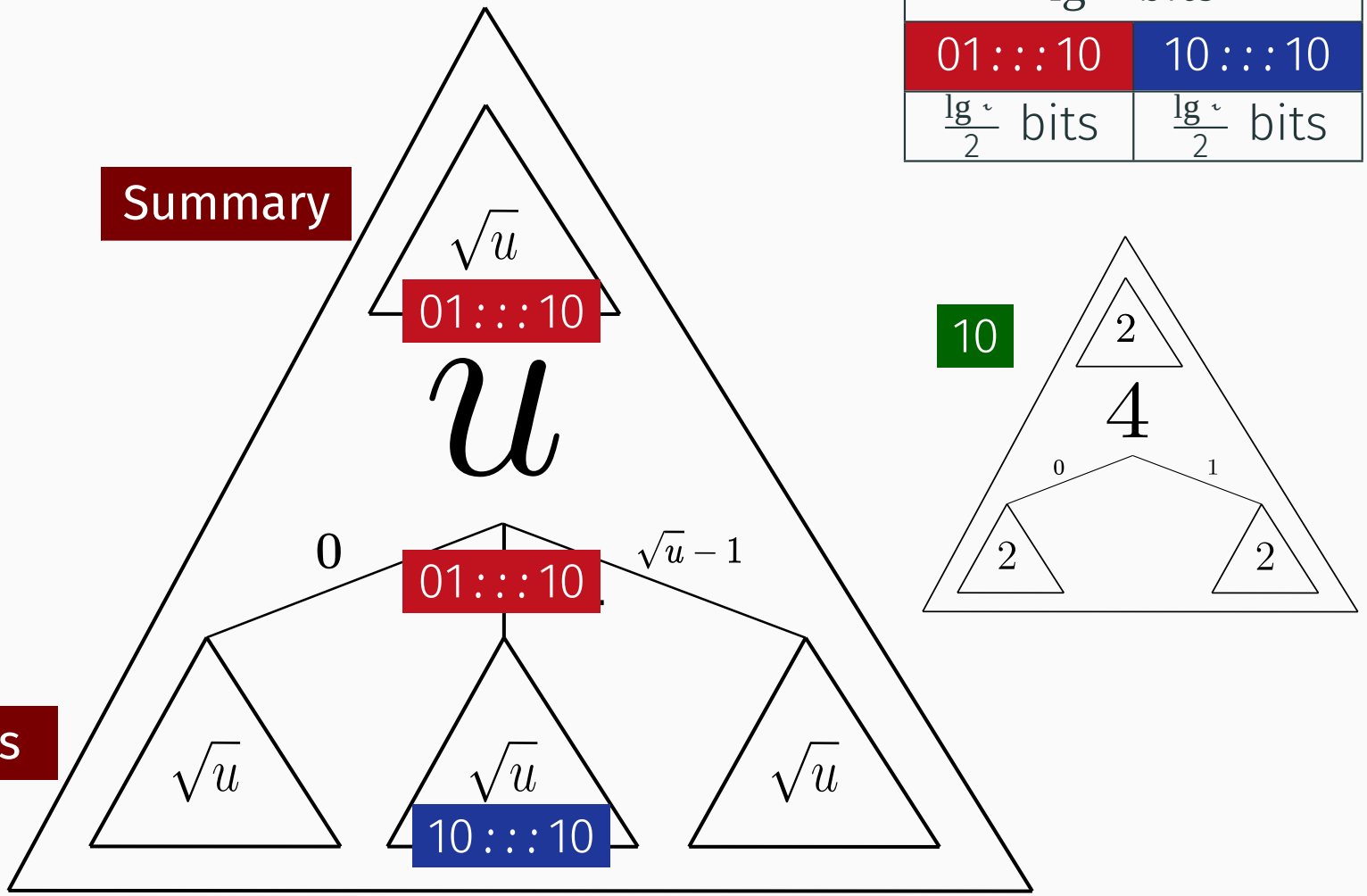


vEB Insert Example



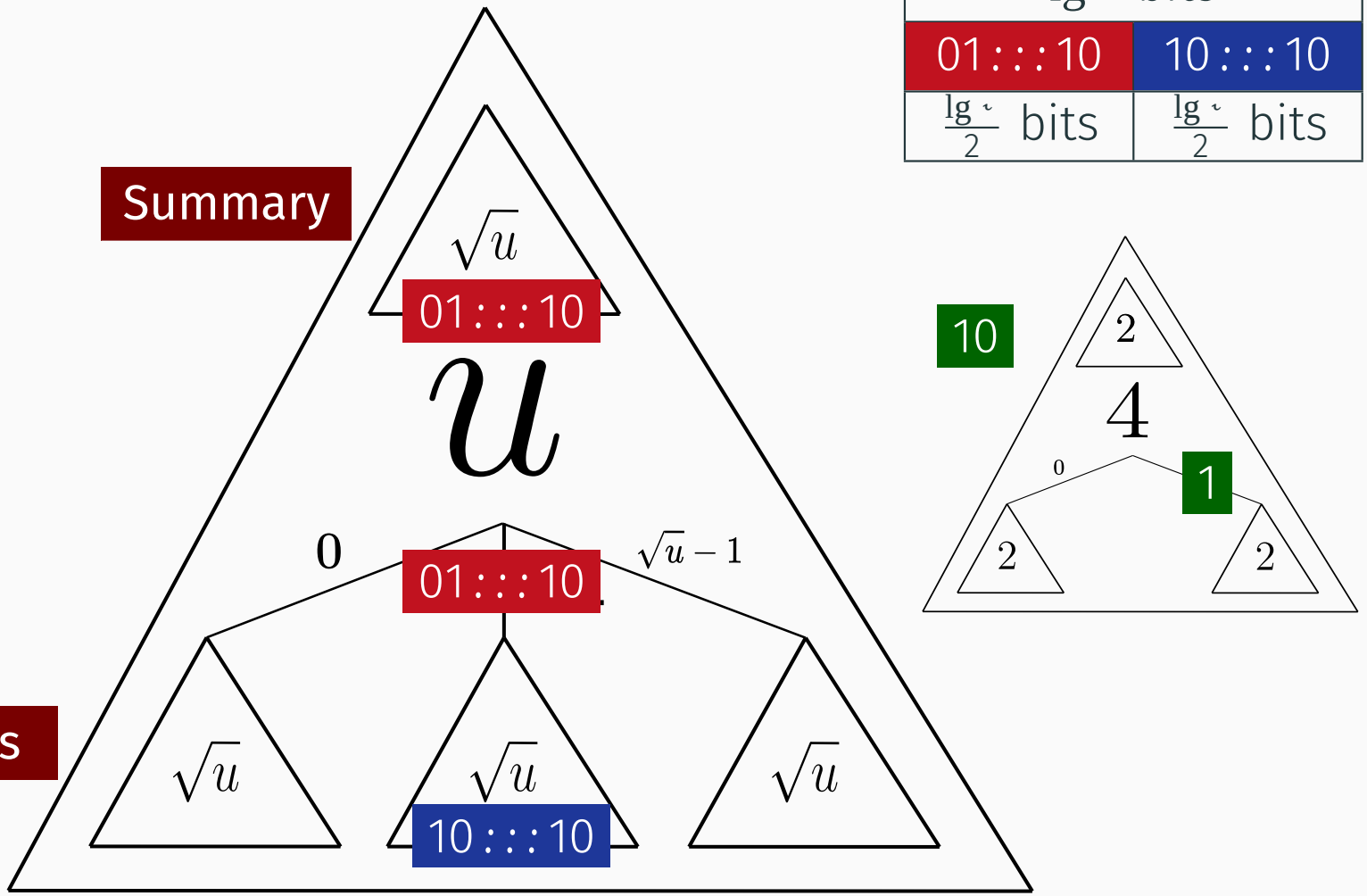
$\lg u$ bits	
01 : : : 10	10 : : : 10
$\frac{\lg u}{2}$ bits	$\frac{\lg u}{2}$ bits

vEB Insert Example

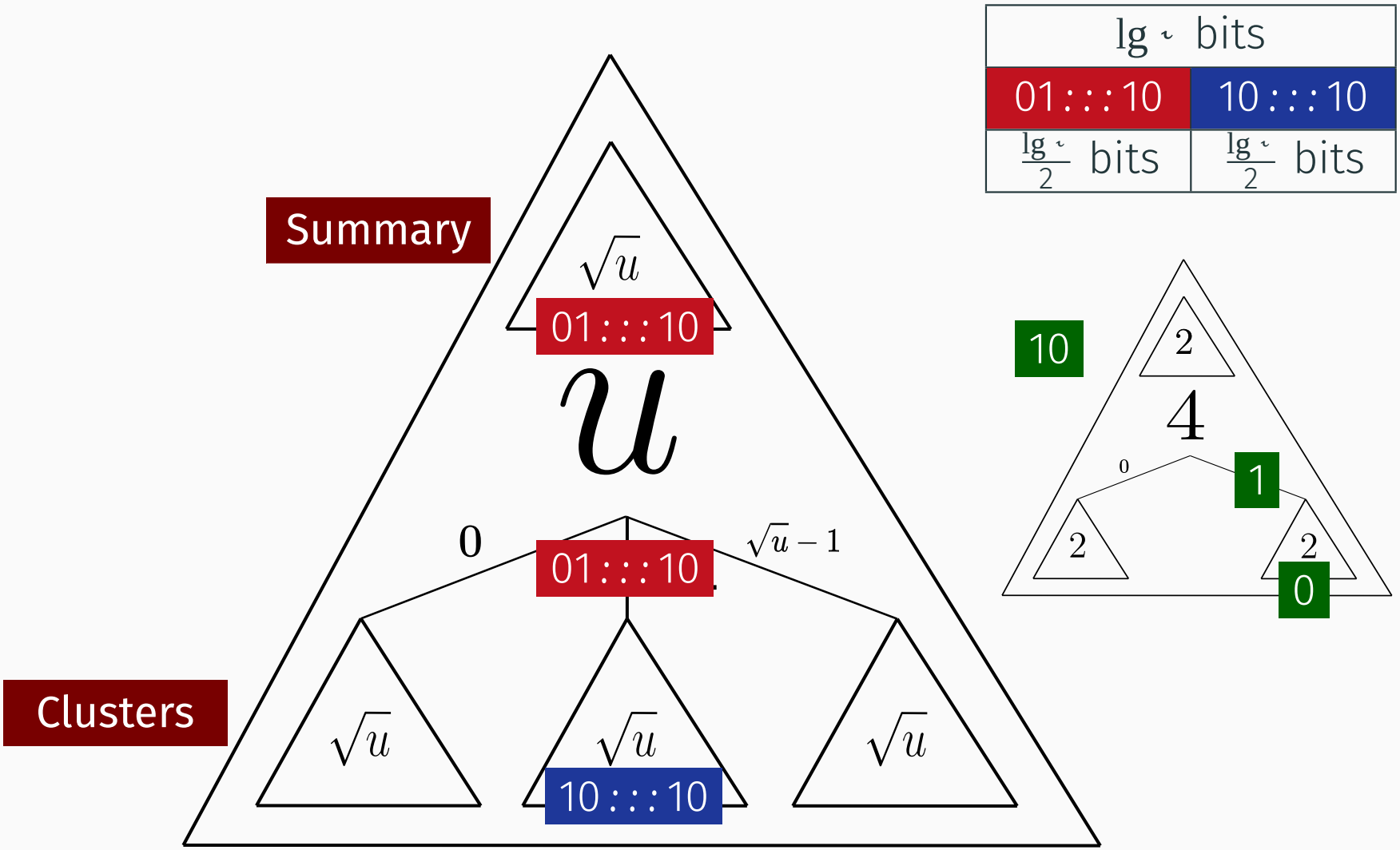


$\lg u$ bits	
01 : : : 10	10 : : : 10
$\frac{\lg u}{2}$ bits	$\frac{\lg u}{2}$ bits

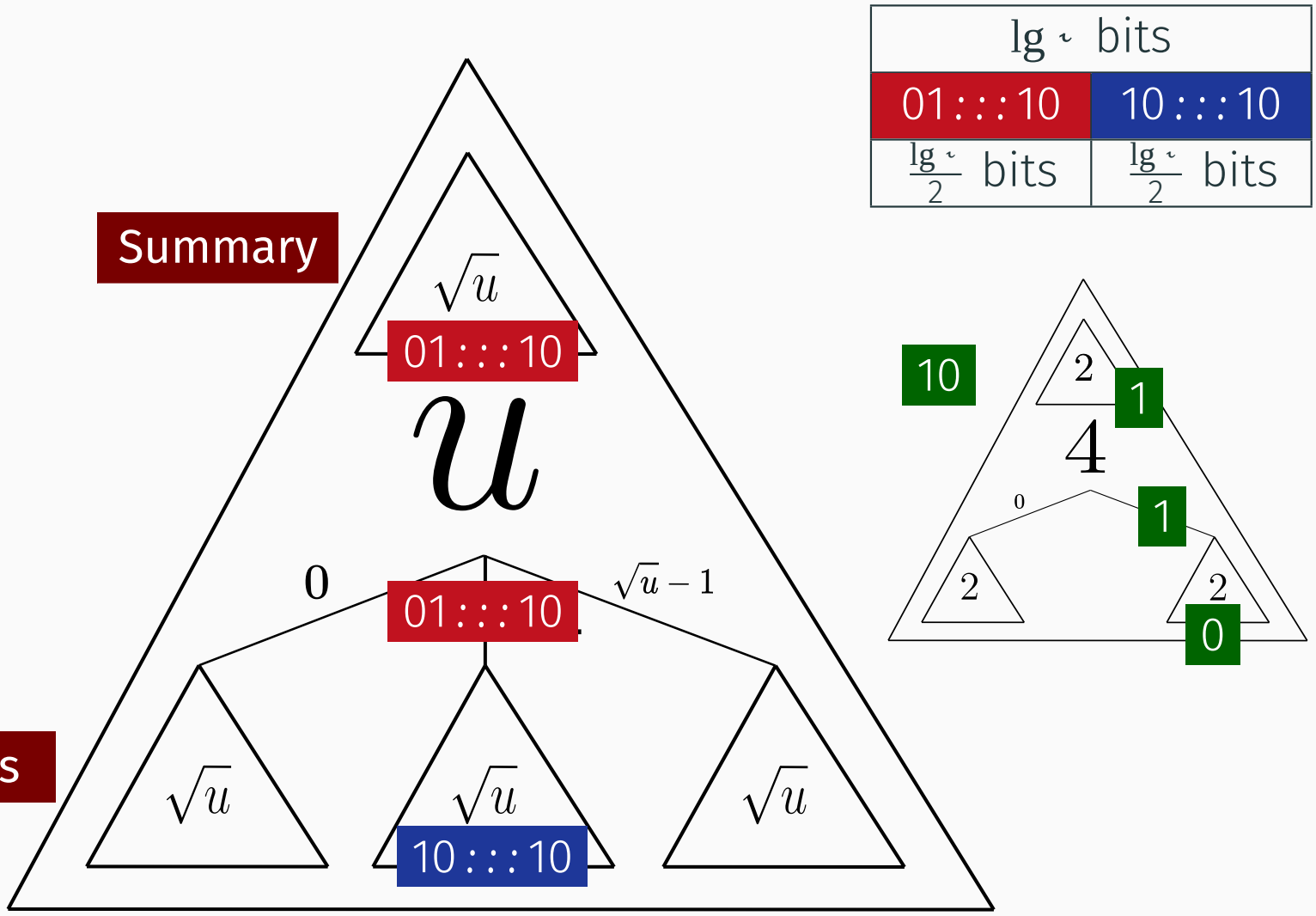
vEB Insert Example



vEB Insert Example



vEB Insert Example



Hardware Transactional Memory (HTM)

- Atomic blocks of code
  - Take effect as a single indivisible unit (Commit)
  - Or rolled back with no effect on shared memory (Abort)
- Hardware level → Efficient

Hardware Transactional Memory (HTM)

- Atomic blocks of code
  - Take effect as a single indivisible unit (Commit)
  - Or rolled back with no effect on shared memory (Abort)
- Hardware level →Efficient

```
hms rs`str < ^wadfhm'( :
```

Hardware Transactional Memory (HTM)

- Atomic blocks of code
  - Take effect as a single indivisible unit (Commit)
  - Or rolled back with no effect on shared memory (Abort)
- Hardware level →Efficient

```
hms rs`str < ^wadfhm'( :
he 'rs`str << ^WADFHM^RS@QSDC( z
     Sq`mr`bshnm`k bncd---
     ^wdmc'( :
|
```

Hardware Transactional Memory (HTM)

- Atomic blocks of code
  - Take effect as a single indivisible unit (Commit)
  - Or rolled back with no effect on shared memory (Abort)
- Hardware level →Efficient

```
hms rs`str < ^wadfhm'( :
he 'rs`str << ^WADFHM^RS@QSDC( z
     Sq`mr`bshnm`k bncd---
     ^wdmc'( :
|
dkrd z
     E`kka`bj bncd---
|
```

Transactional Lock Elision (TLE)

Transactional Lock Elision (TLE)

```
he '^wadfhm'( << ^WADFHM^RS@QSDC( z
```

Fast Path

```
    Sq`mr`bshnm ancx---

    ^wdmc'(:
|
```

Transactional Lock Elision (TLE)

```
he '^wadfhm'( << ^WADFHM^RS@QSDC( z
```

Fast Path

```
    Sq`mr`bshnm ancx---

    ^wdmc'(:
|
dkrd z
    knbj'(:
    Sq`mr`bshnm ancx---
    tmknbj'(:
|
```

Fallback Path

Transactional Lock Elision (TLE)

Fast Path

Fallback Path

```
he '^wadfhm'( << ^WADFHM^RS@QSDC( z
    he knbj hr `kqd`cx gdkc
        `anqs'(:

    Sq`mr`bshnm ancx---

    ^wdmc'(:
|
dkrd z
    knbj'(:
    Sq`mr`bshnm ancx---
    tmknbj'(:
|
```

Transactional Lock Elision (TLE)

Fast Path

Fallback Path

```
he '^wadfhm'( << ^WADFHM^RS@QSDC( z
    he knbj hr `kqd`cx gdkc
        `anqs'(:

    Sq`mr`bshnm ancx--- ƚ S2

    ^wdmc'(:
|
dkrd z
    knbj'(: ƚ S1
    Sq`mr`bshnm ancx---
    tmknbj'(:
|
```

Transactional Lock Elision (TLE)

Fast Path

Fallback Path

```
he '^wadfhm'( << ^WADFHM^RS@QSDC( z
    he knbj hr `kqd`cx gdkc
        `anqs'(:

    Sq`mr`bshnm ancx--- ƀ S2

    ^wdmc'(:
|
dkrd z
    knbj'(:
    Sq`mr`bshnm ancx--- ƀ S1
    tmknbj'(:
|
```

Transactional Lock Elision (TLE)

Fast Path

Fallback Path

```
he '^wadfhm'( << ^WADFHM^RS@QSDC( z
    he knbj hr `kqd`cx gdkc
        `anqs'(:

    Sq`mr`bshnm ancx---

    ^wdmc'(: ƀ S2
|
dkrd z
    knbj'(:
    Sq`mr`bshnm ancx--- ƀ S1
    tmknbj'(:
|
```

Transactional Lock Elision (TLE)

Fast Path

Fallback Path

```
he '^wadfhm'( << ^WADFHM^RS@QSDC( z
    he knbj hr `kqd`cx gdkc
        `anqs'(:

    Sq`mr`bshnm ancx---

    ^wdmc'(:
|
dkrd z ƀ S2
    knbj'(:
    Sq`mr`bshnm ancx--- ƀ S1
    tmknbj'(:
|
```

Methodology

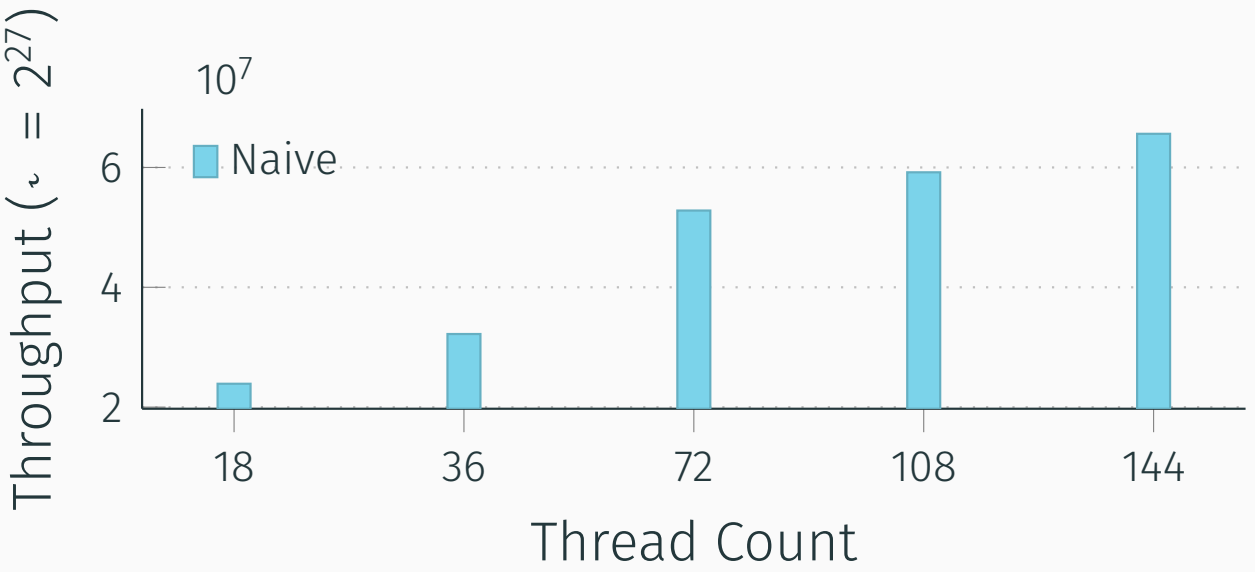
Overview

- Concurrent vEB trees with recursive summaries
- Full support for successor and predecessor queries
- Starting with a naive vEB set implementation
- Use simple building blocks to end up with a performant vEB map

First Step: Naive

Idea

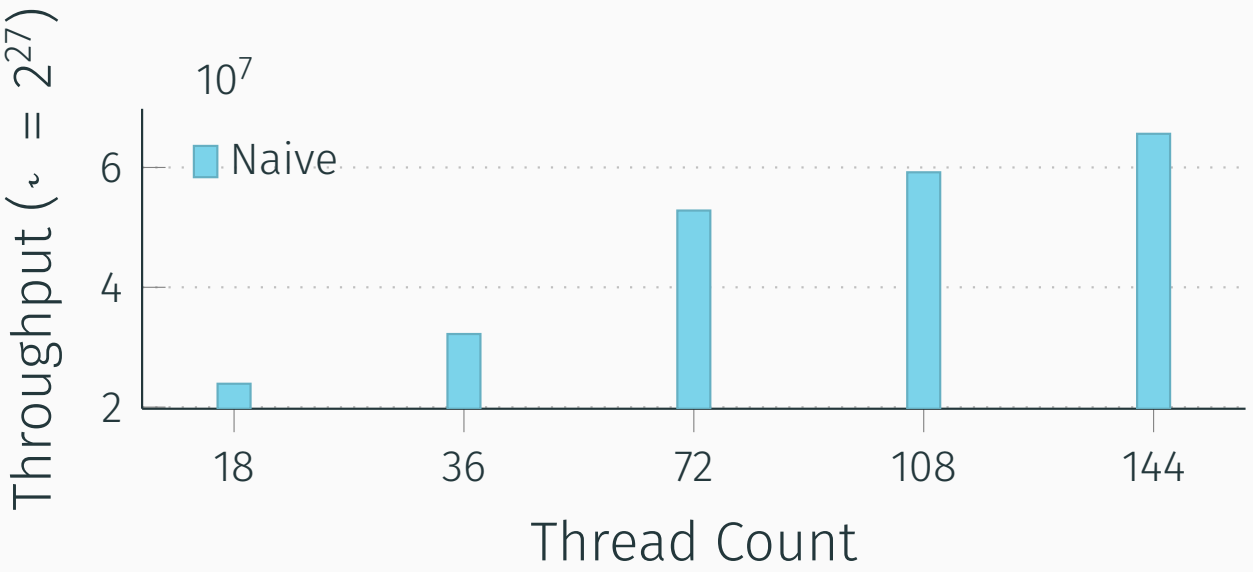
Wrap sequential vEB operations in TLE blocks



First Step: Naive

Idea

Wrap sequential vEB operations in TLE blocks



Problem

Memory usage does not depend on the number of keys!

Improve Memory Footprint: Dynamic

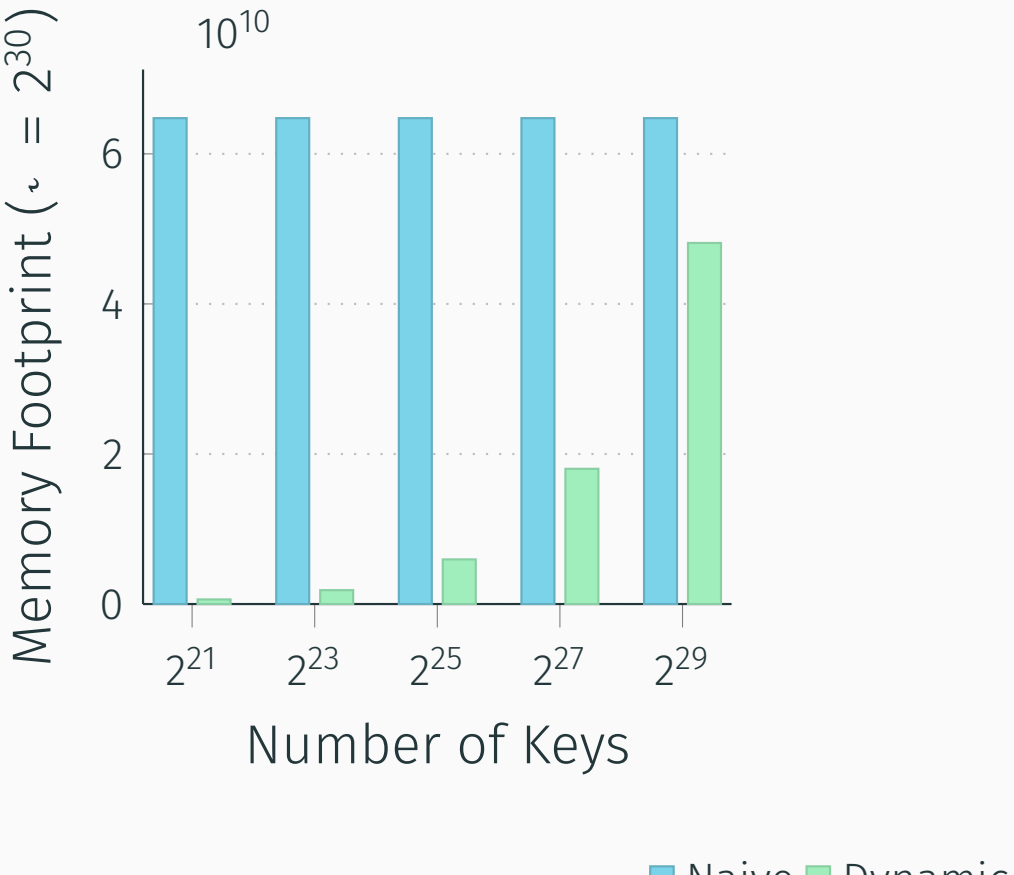
Can we save memory?

In the Naive variation, the entire tree is pre-allocated

Improve Memory Footprint: Dynamic

Can we save memory?

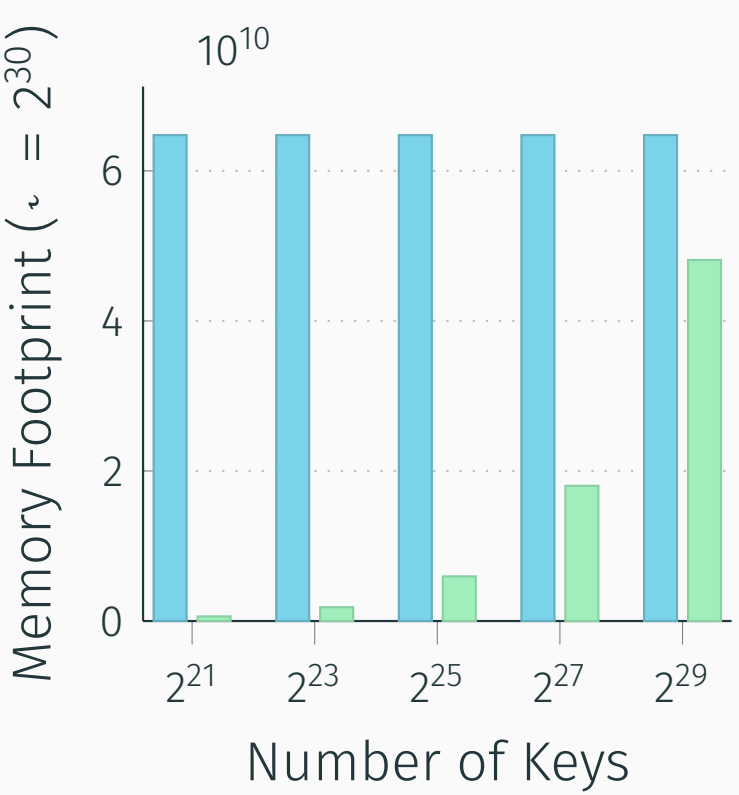
In the Naive variation, the entire tree is pre-allocated

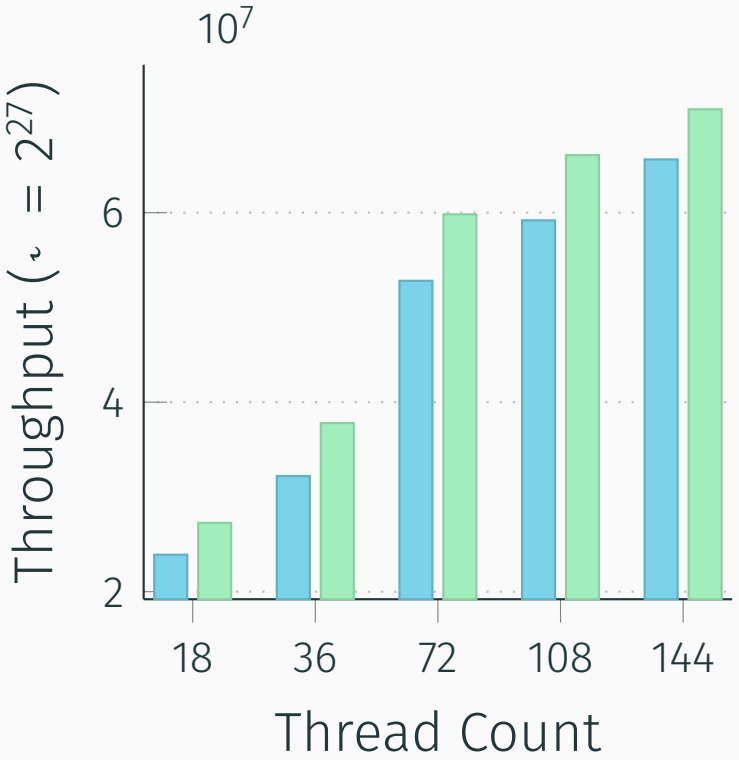


Improve Memory Footprint: Dynamic

Can we save memory?

In the Naive variation, the entire tree is pre-allocated





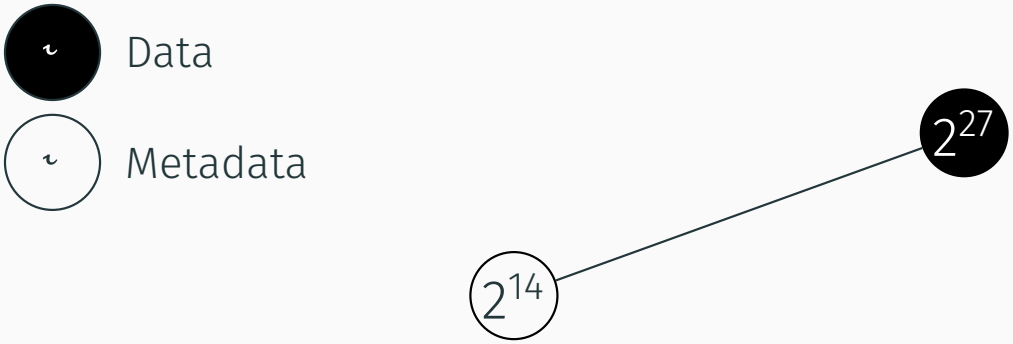
Naive Dynamic

Room For Improvement: Cut-Off

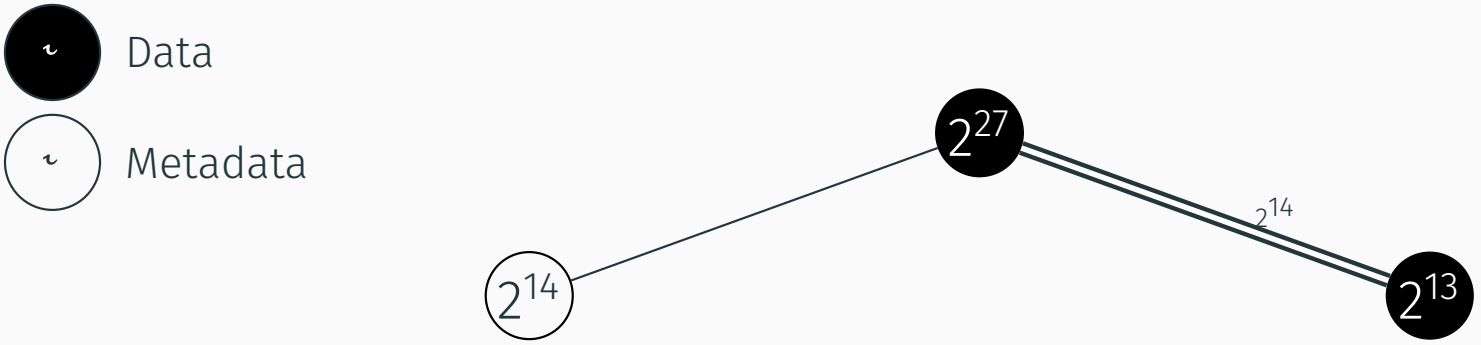
Data

2^{27}

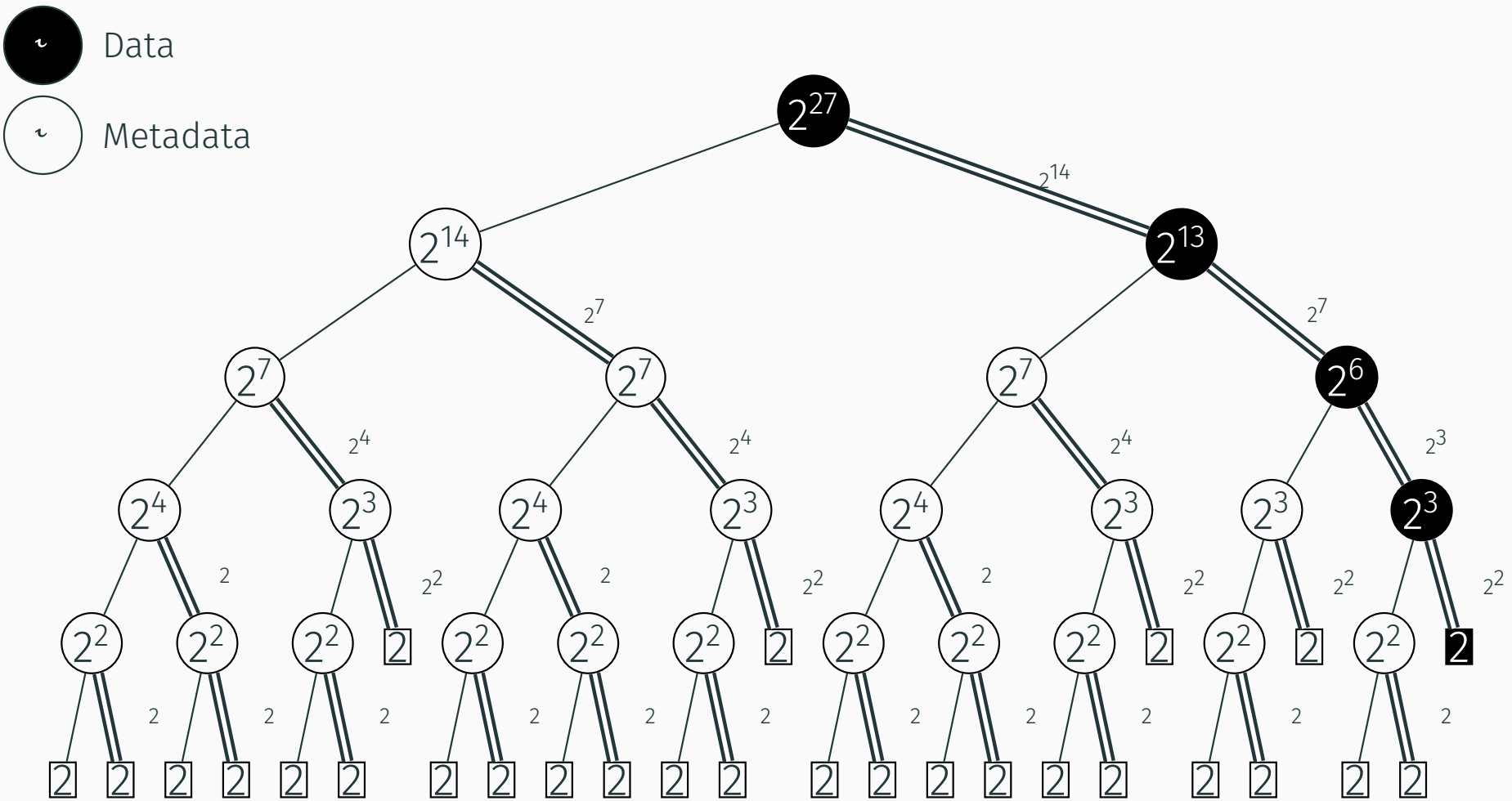
Room For Improvement: Cut-Off



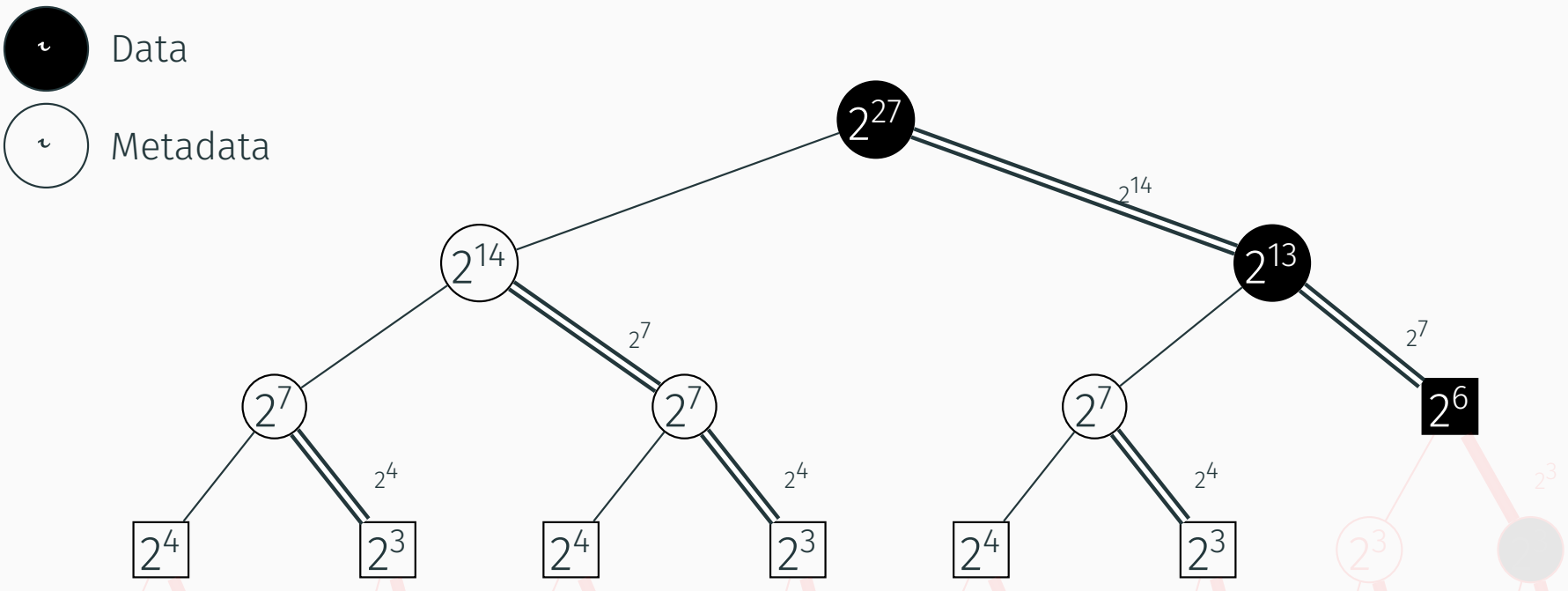
Room For Improvement: Cut-Off



Room For Improvement: Cut-Off



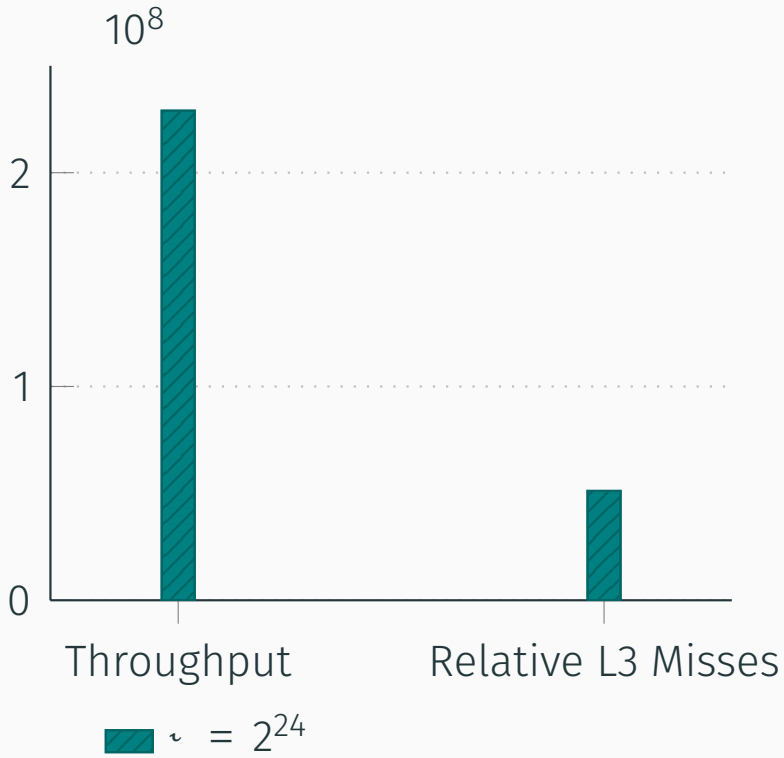
Room For Improvement: Cut-Off

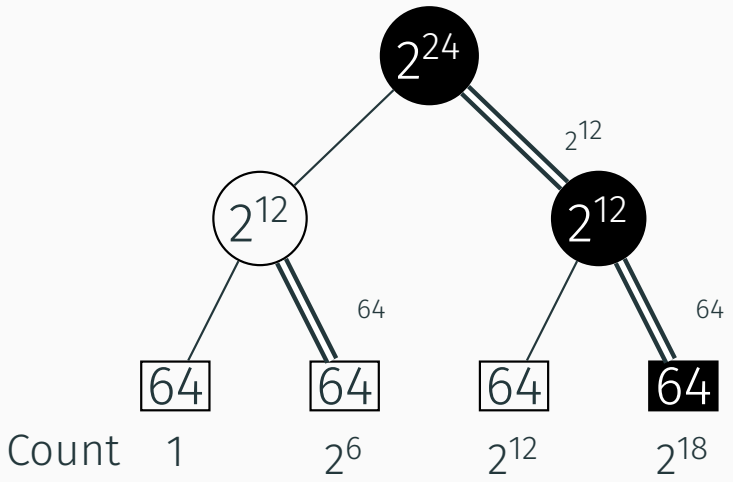


Stop the recursion at ι 64

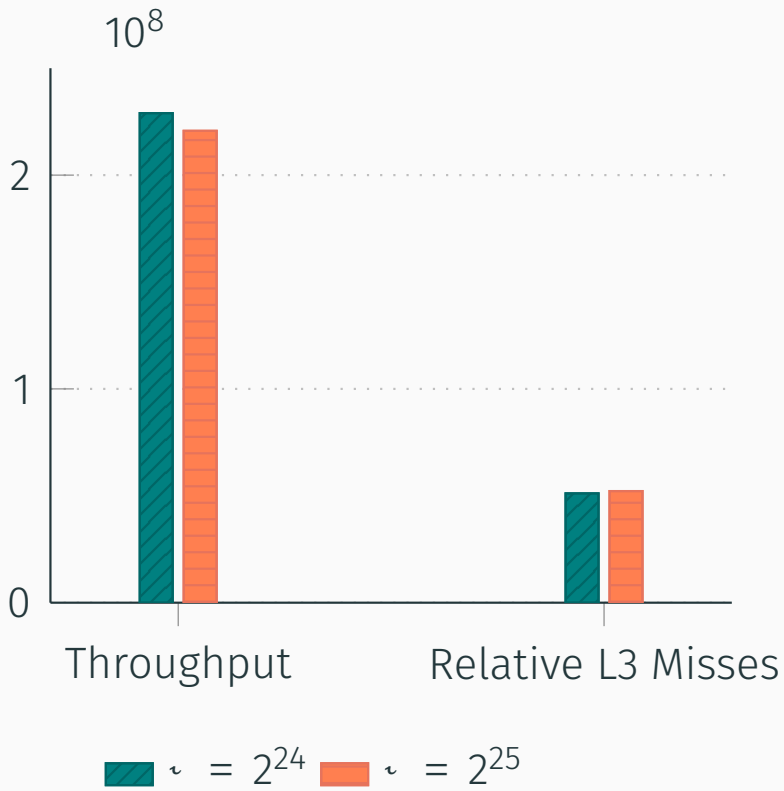
- 98.6% fewer nodes at $\iota = 2^{27}$
- One L3 cache miss vs. 7
- 3 faster
- 75% less memory

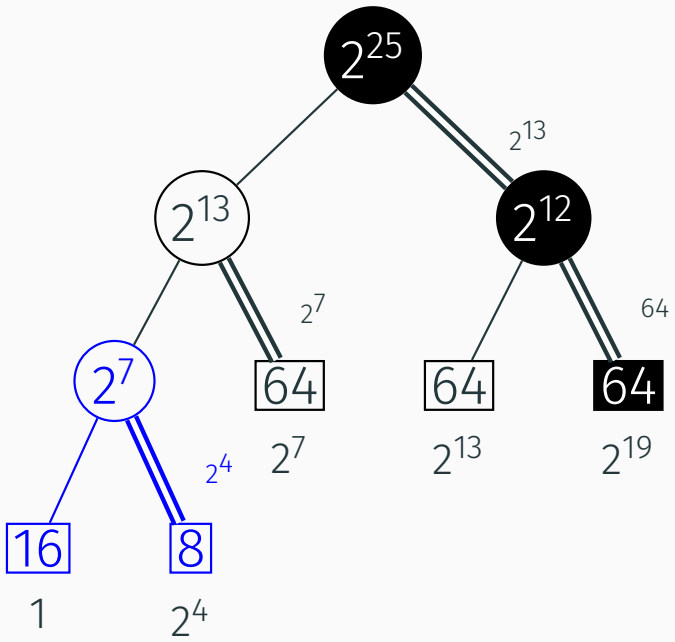
Problem with Cut-Off



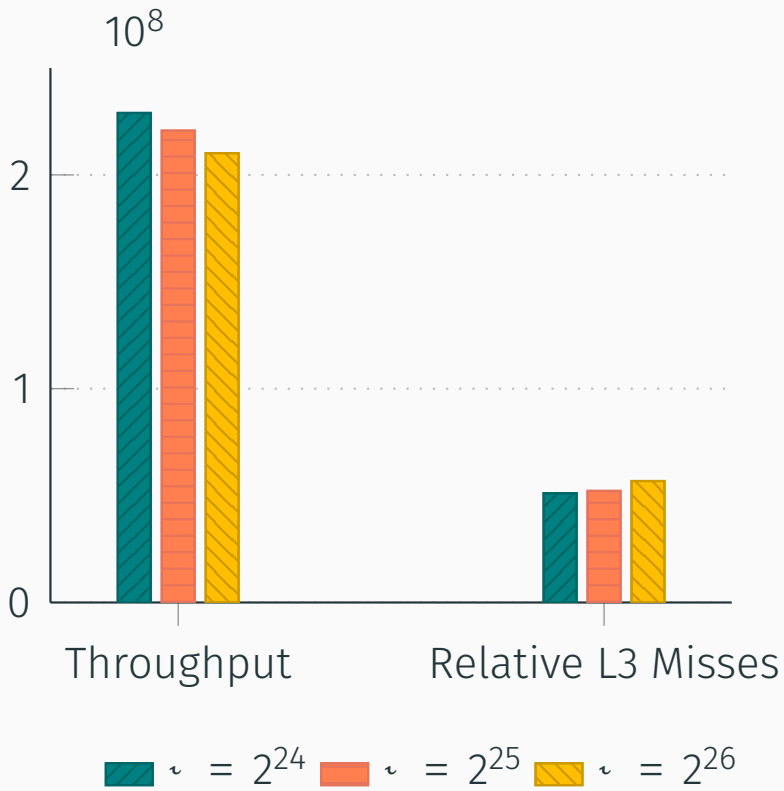


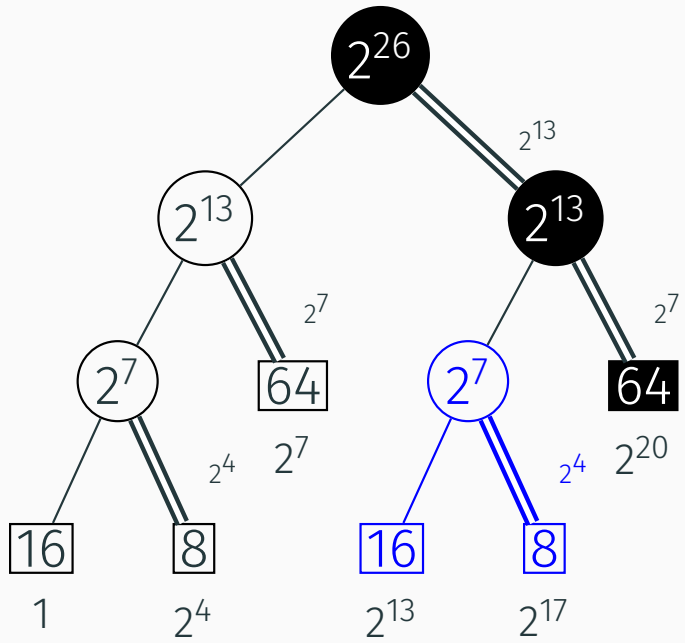
Problem with Cut-Off



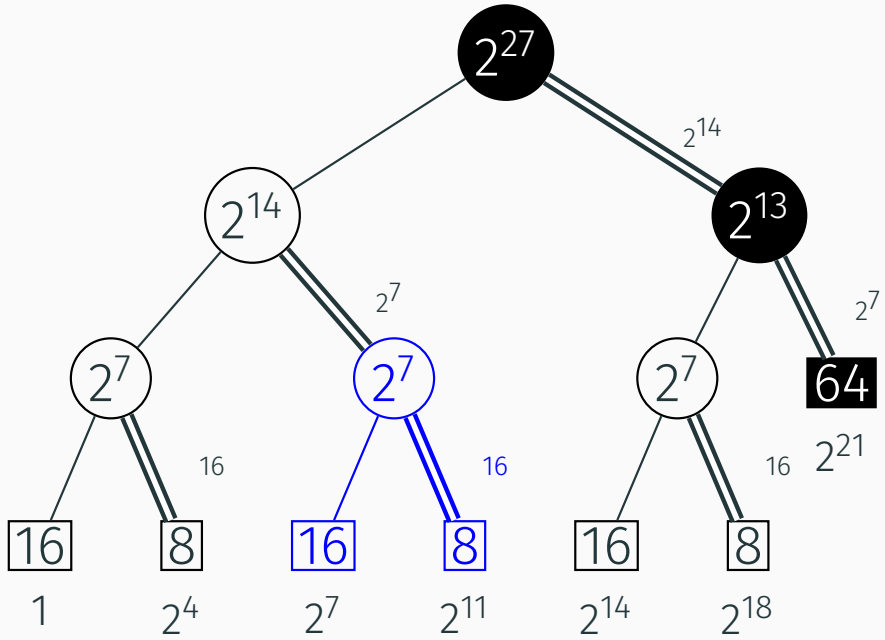


Problem with Cut-Off

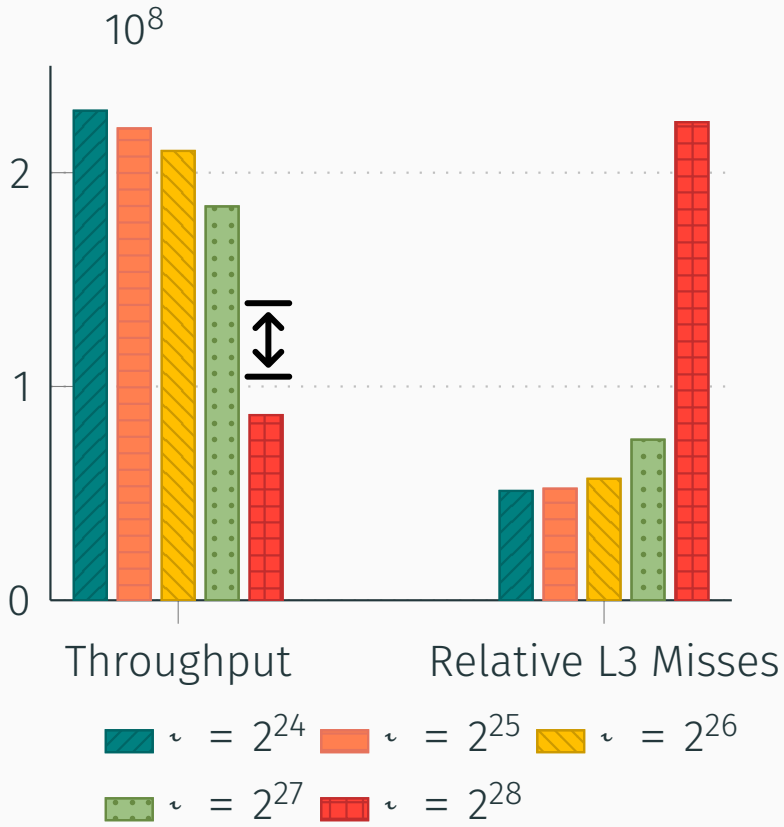




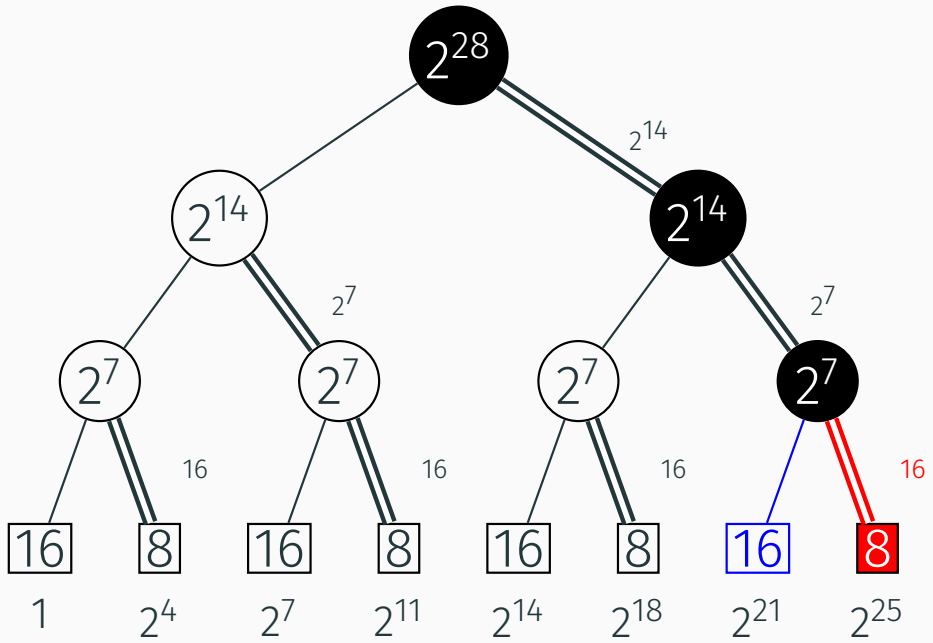
Problem with Cut-Off



Problem with Cut-Off

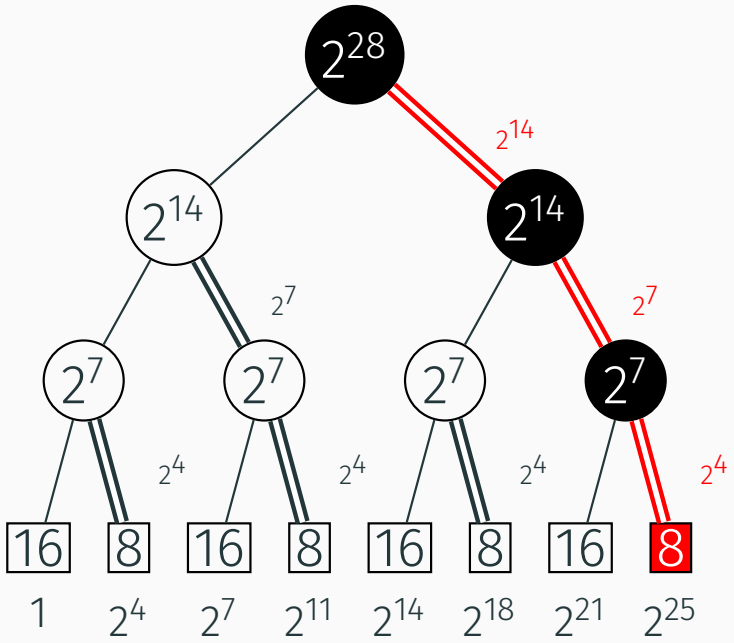


Problem with Cut-Off



Solution: New-Root

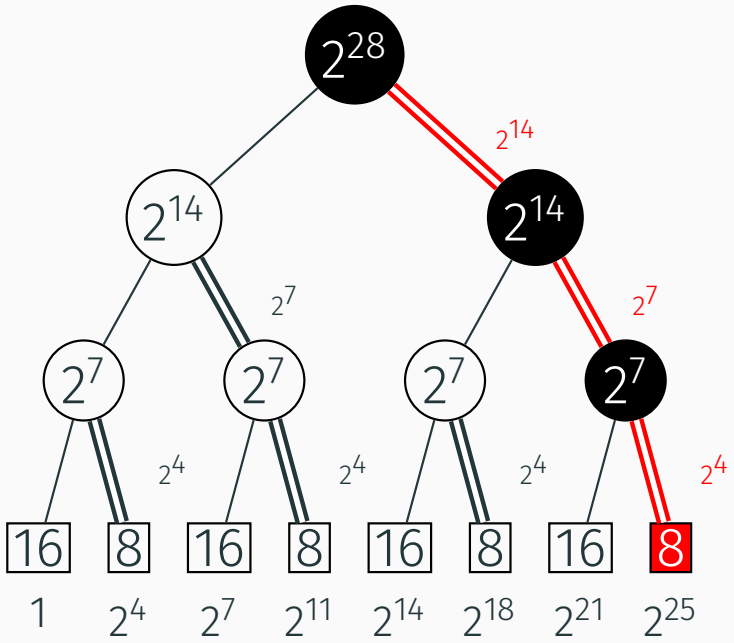
Before



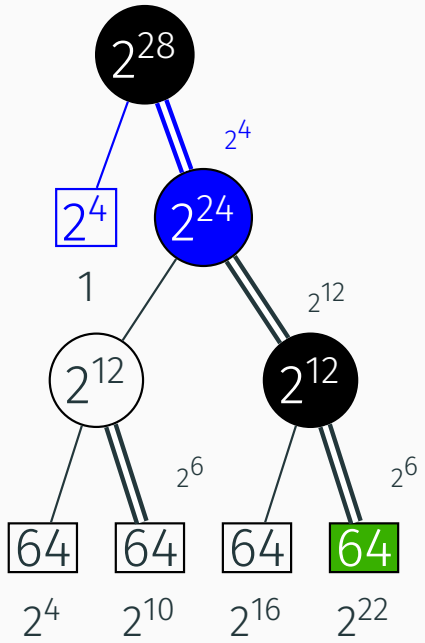
Universe Size	2^{24}	2^{25}	2^{26}	2^{27}	2^{28}	2^{29}
Data Leaf Count	2^{18}	2^{19}	2^{20}	2^{21}	2^{25}	2^{26}
Data Leaf Size	64	64	64	64	8	8

Solution: New-Root

Before

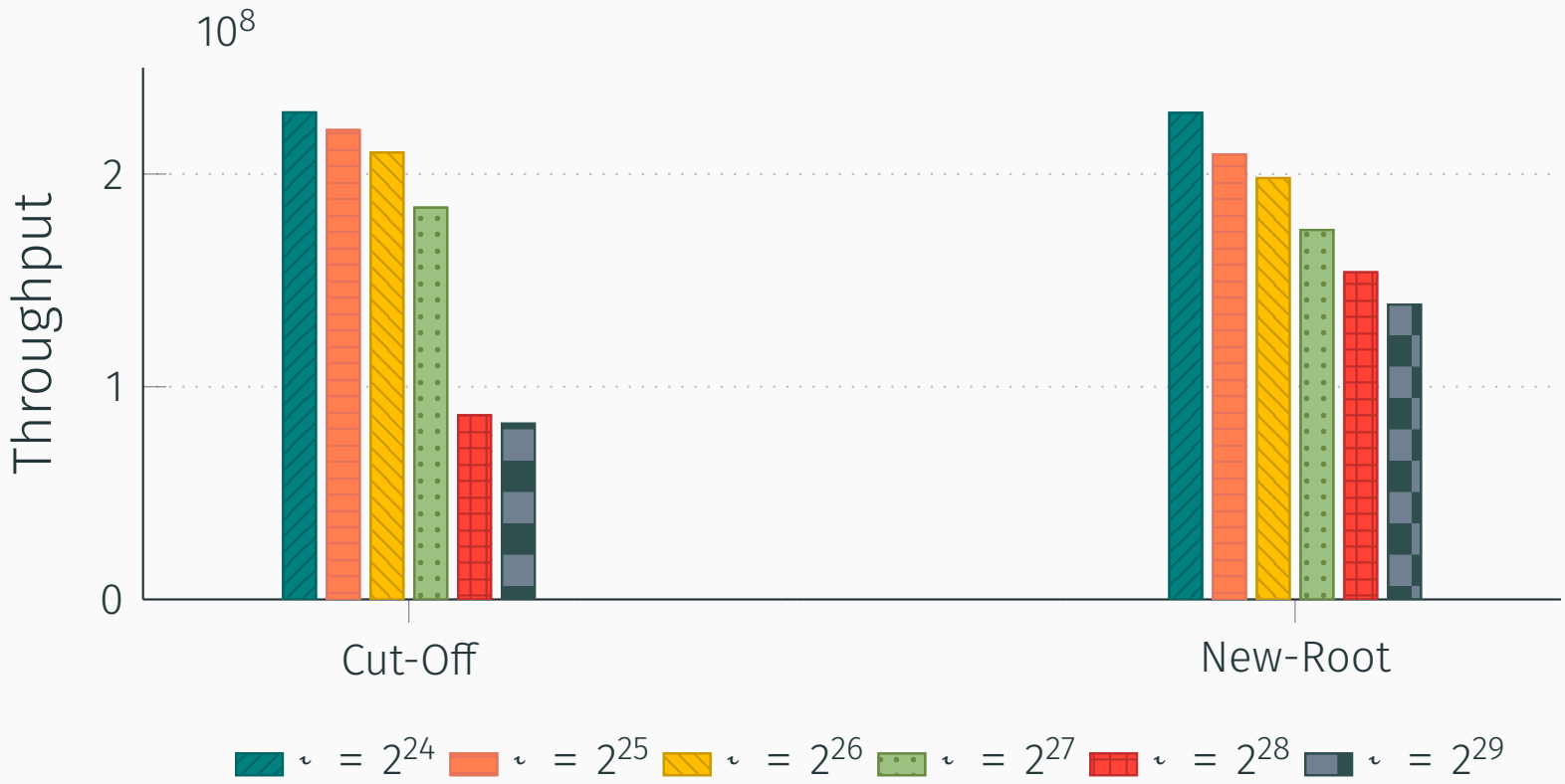


After

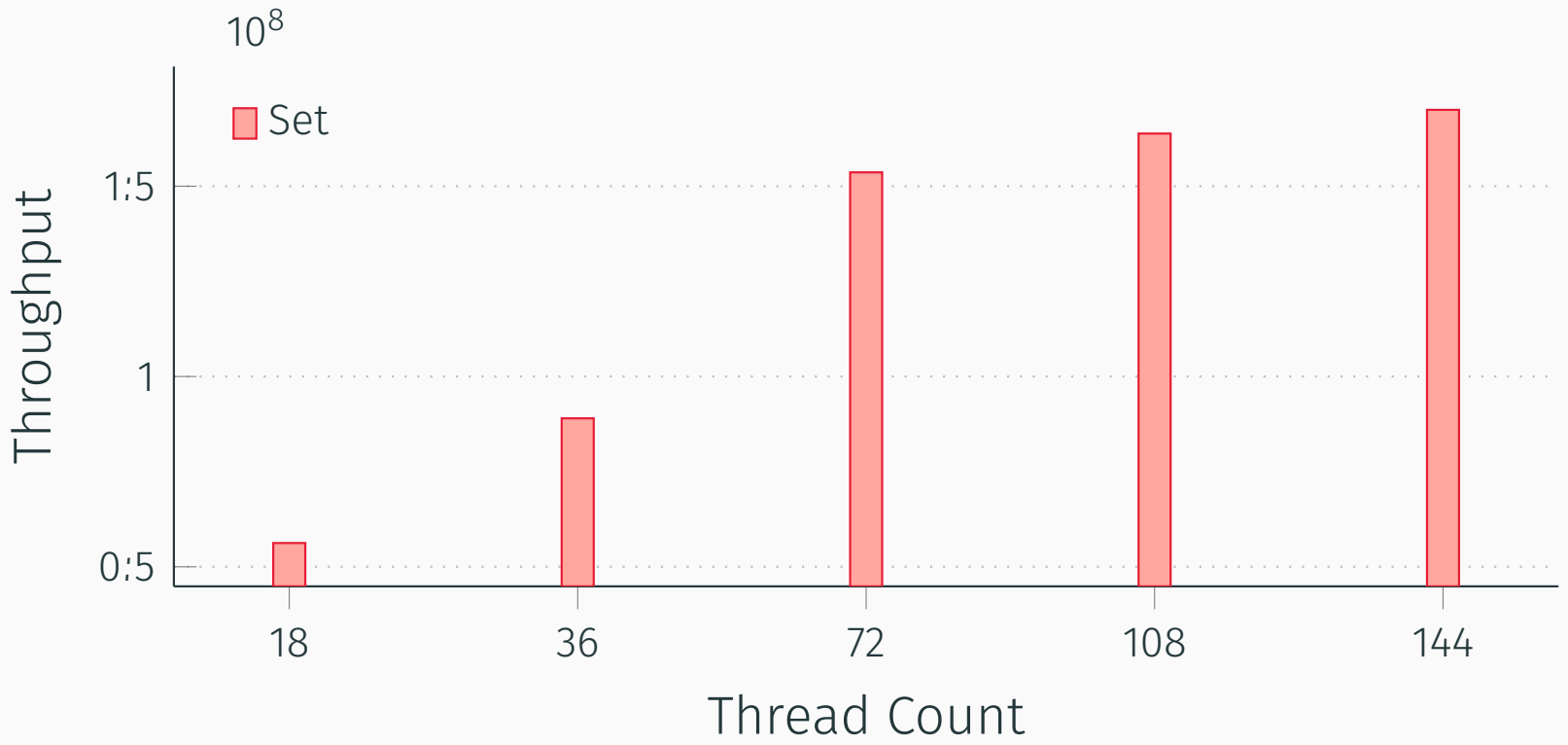


Universe Size	2^{24}	2^{25}	2^{26}	2^{27}	2^{28}	2^{29}
Data Leaf Count	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}
Data Leaf Size	64	64	64	64	64	64

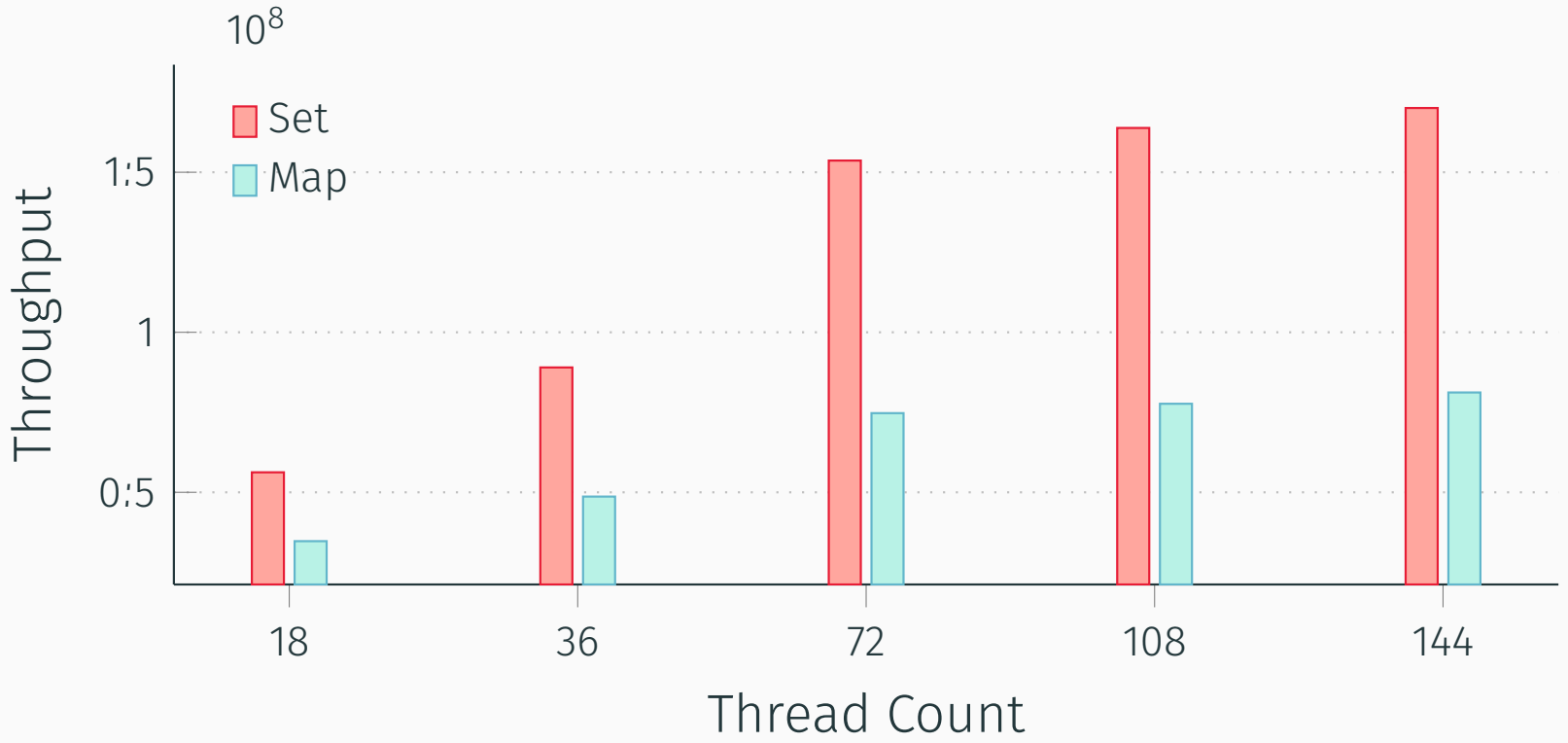
Solution: New-Root



Moving On to Maps

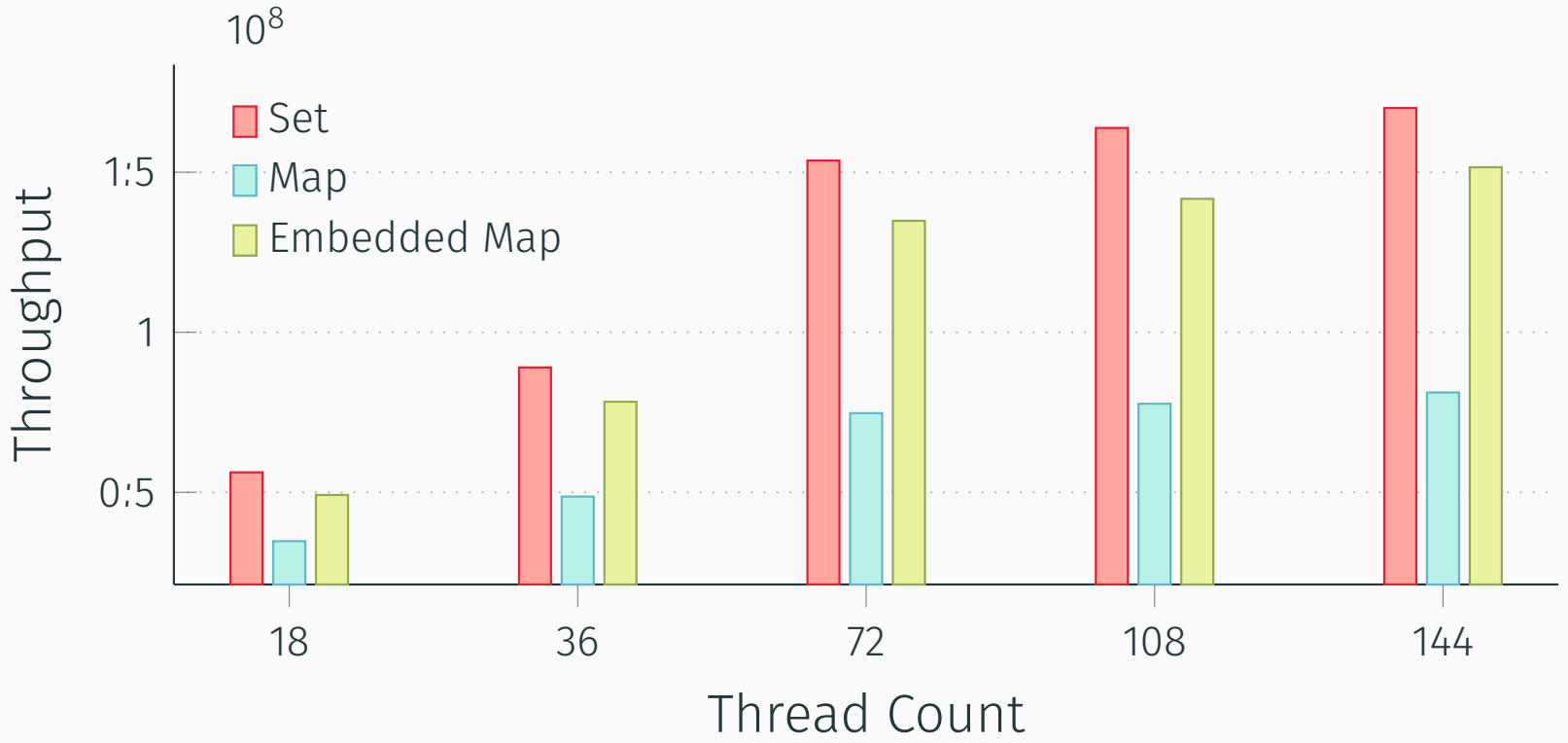


Moving On to Maps

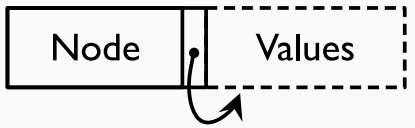


Map

Moving On to Maps



Map **Embedded Map**

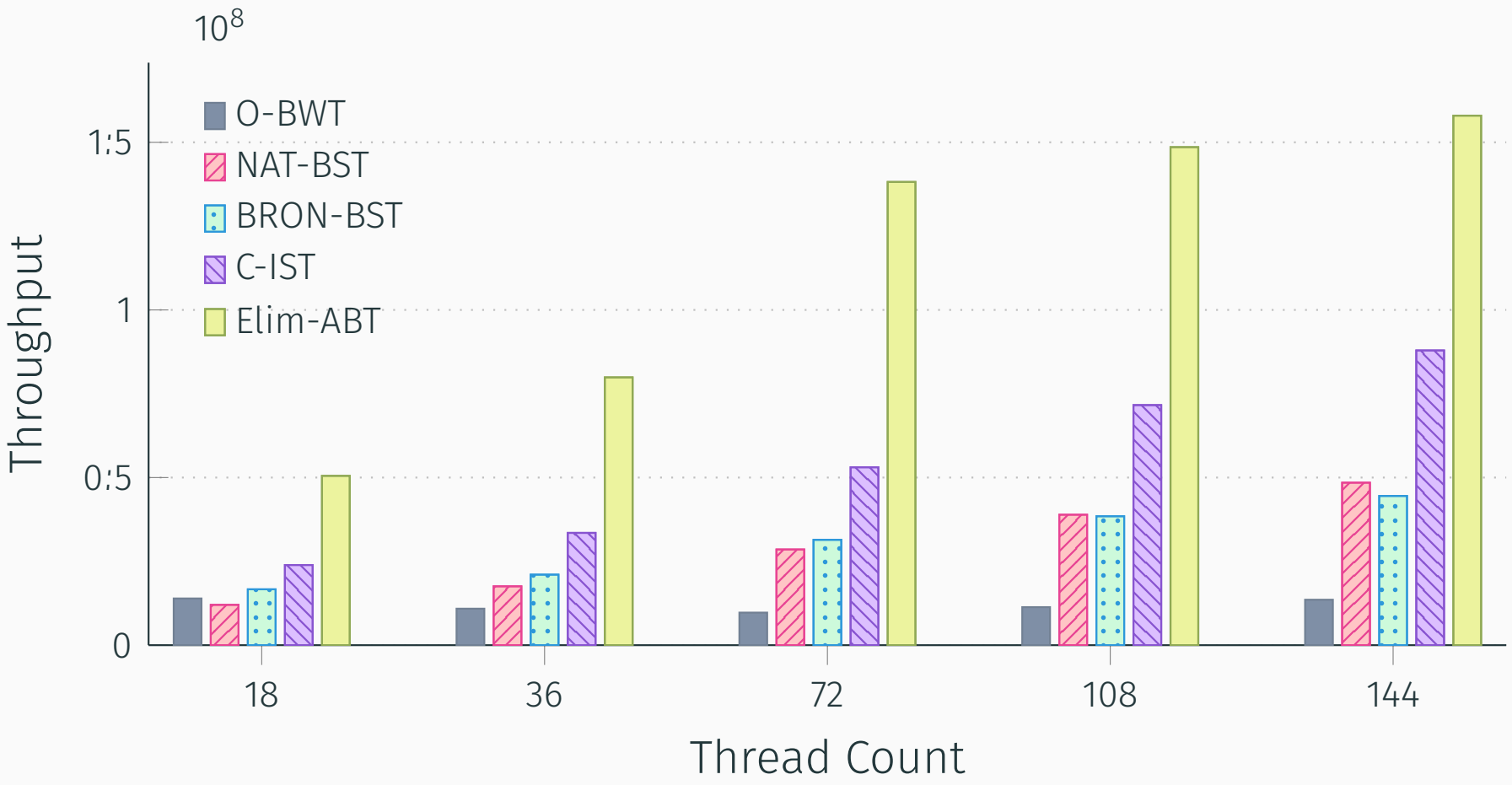


Experimental Evaluation

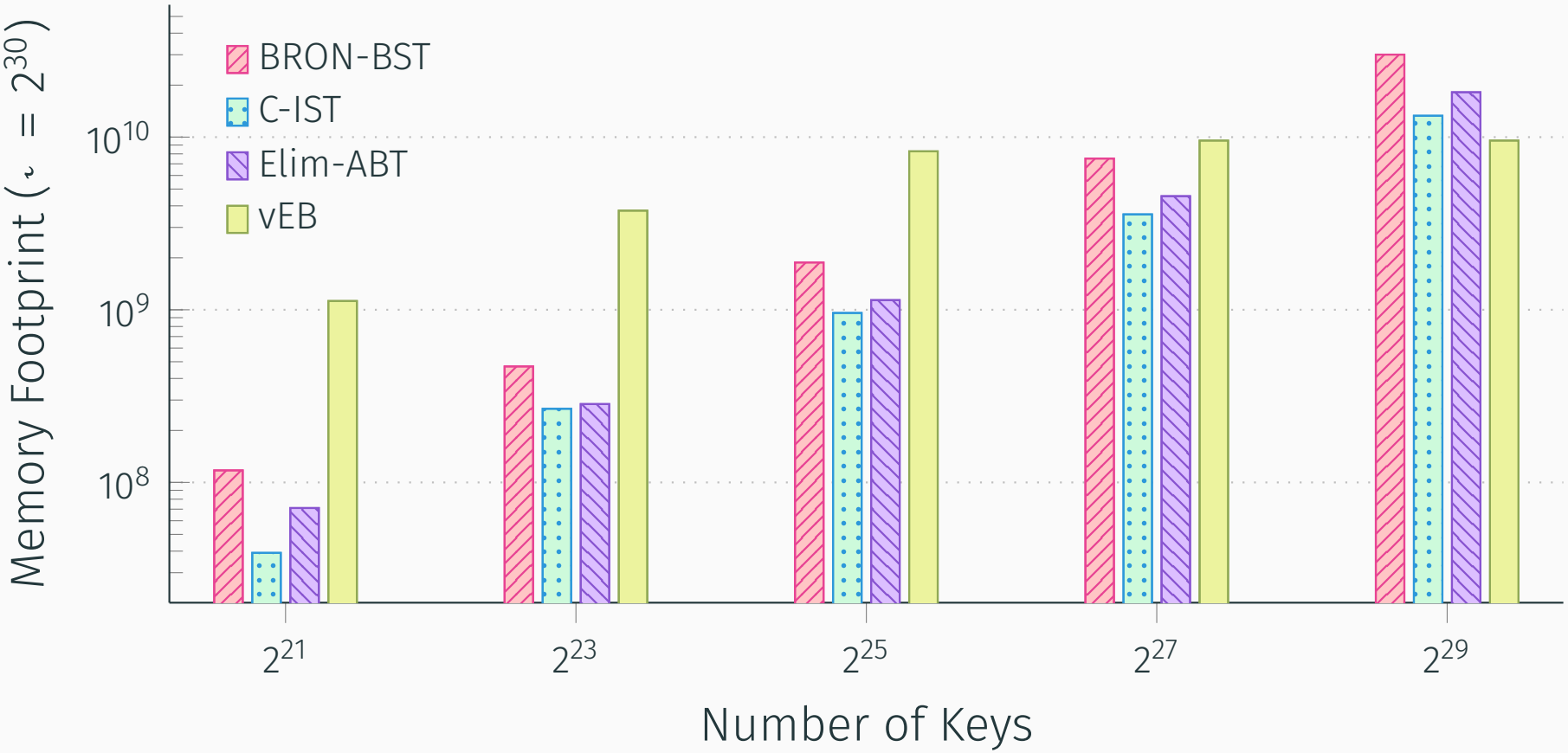
Experimental Setup

- Test harness: SetBench [2]
- 4 sockets 18 cores per socket = 72 cores, 144 hardware threads
- Comparison points:
  - **Elim-ABT:** A concurrent (a,b)-tree [3]
  - **C-IST:** A doubly logarithmic interpolation search tree [2]
  - **BRON-BST:** An OCC BST [1]
  - **O-BWT:** The open-source implementation of BW-Tree [5]

Performance

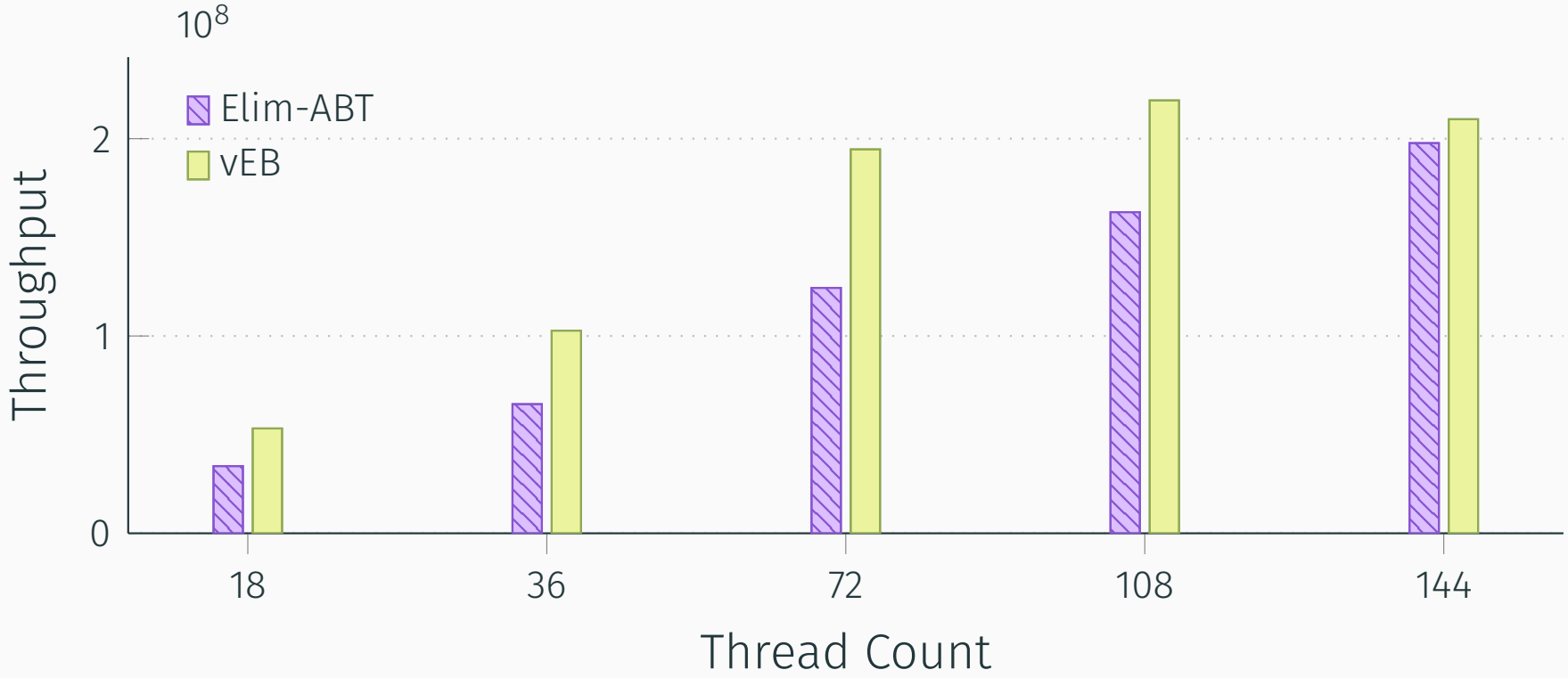


Memory Footprint



Successor Queries

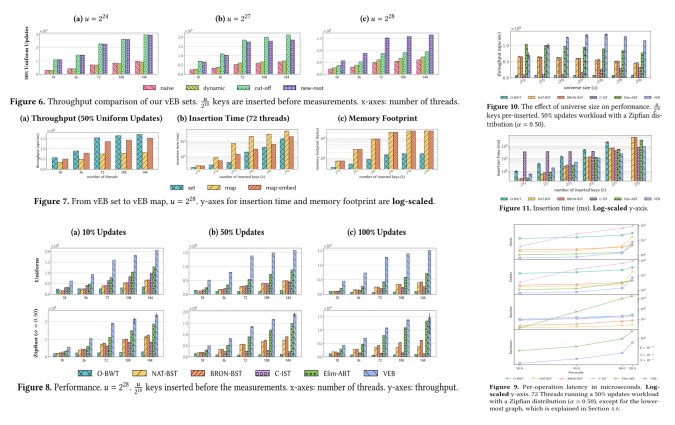
- The successor operation for Elim-ABT is non-linearizable
- Workload: 98% successor queries (Zipfian with = 0:50)



Summary

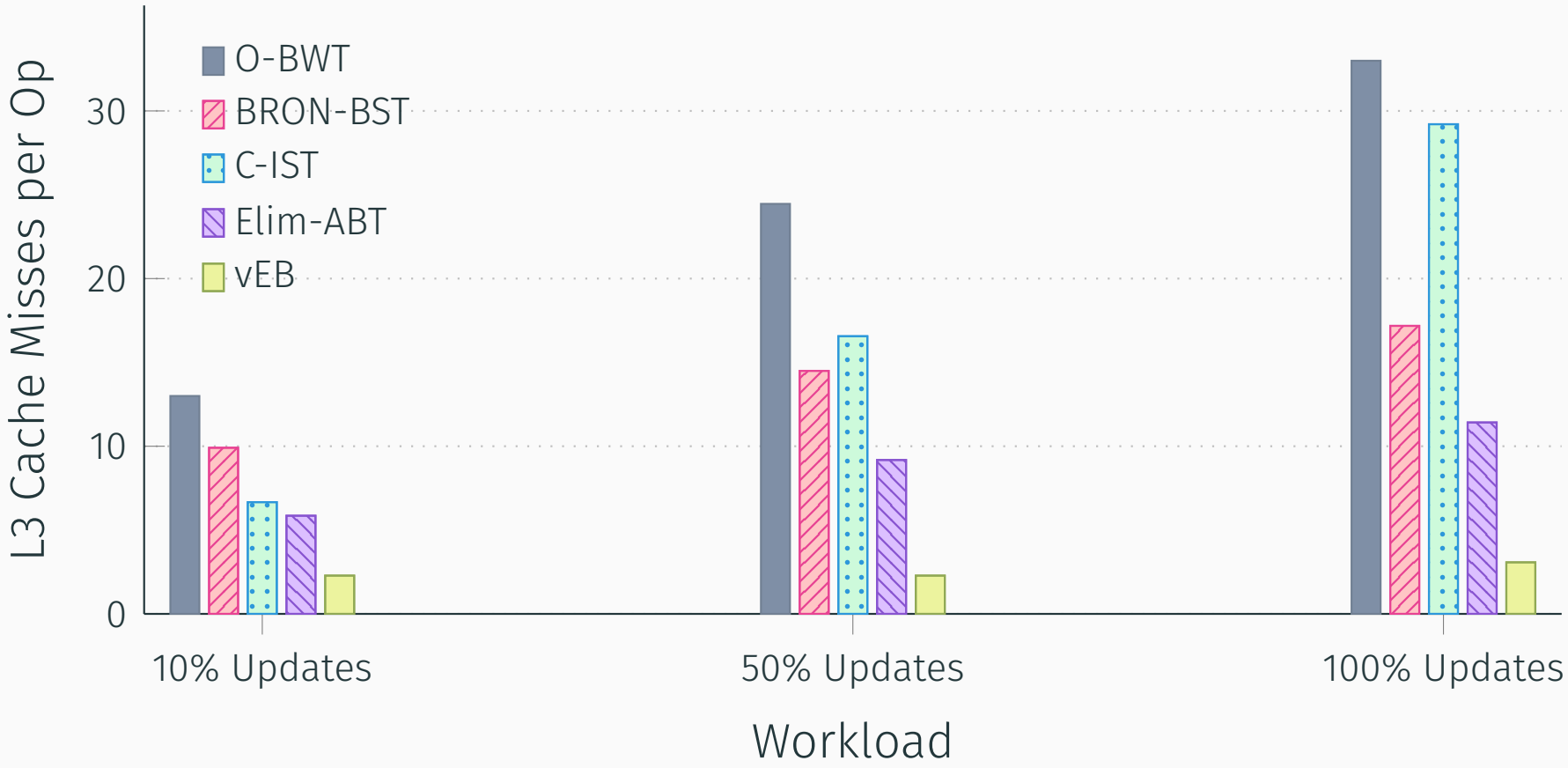
Summary

- Concurrent vEB trees utilizing HTM
- Low-overhead thread synchronization
- On average 5× faster
- More in the paper: **https://bit.ly/htmveb**
  - Zipfian workloads
  - Cache performance
  - Insertion time



Questions?

L3 Cache Misses



[1] N. G. Bronson, J. Casper, H. Chafi, and K. Olukotun.
A practical concurrent binary search tree.
In [illegible], pages 257–268. ACM, 2010.

[2] T. Brown, A. Prokopec, and D. Alistarh.
Non-blocking interpolation search trees with doubly-logarithmic running time.
In [illegible], pages 276–291. ACM, 2020.

[3] A. Srivastava and T. Brown.
Elimination (a, b)-trees with fast, durable updates.
In *[illegible]*, pages 416–430. ACM, 2022.

[4] P. van Emde Boas, R. Kaas, and E. Zijlstra.
Design and implementation of an efficient priority queue.
[illegible], 10:99–127, 1977.

[5] Z. Wang, A. Pavlo, H. Lim, V. Leis, H. Zhang, M. Kaminsky, and D. G. Andersen.
Building a bw-tree takes more than just buzz words.
In *[illegible]*, pages 473–488. ACM, 2018.