SEA 2021

Engineering Predecessor Data Structures for Dynamic Integer Sets

Patrick Dinklage Johannes Fischer Alexander Herlez

Predecessor Search

- Given a set S of n positive integers from a universe $U = [0, u-1]$
- **Predecessor** of an integer x is the largest $y \leq x$ contained in S

$S := \{ 1, 2, 4, 6, 7, 19, 21 \}$

pred(6) *pred(20)*

- Analogous definition: successor
- Survey: *[Navarro & Rojas-Ledesma, ACM Comput. Surv. 105, 2020]*

Dynamic Predecessor Data Structures

- **Dynamic Problem**: S is subject to indels
- Task: Maintain a data structure to answer predecessor queries
- Classics: van Emde Boas Trees & y-fast tries with query time $O(\lg\lg \boldsymbol{u})$
- Optimal query time achieved by *[Pătraşcu & Thorup, FOCS 2014]*

Related Work & Our Contributions

- Dynamic predecessor benchmarks have been a while ago…
  - *[Nash & Gregg, WEA 2008]* (Burst Tries)

→ We engineer three new data structures based on known ideas
(targeted at *practical* in-memory performance)

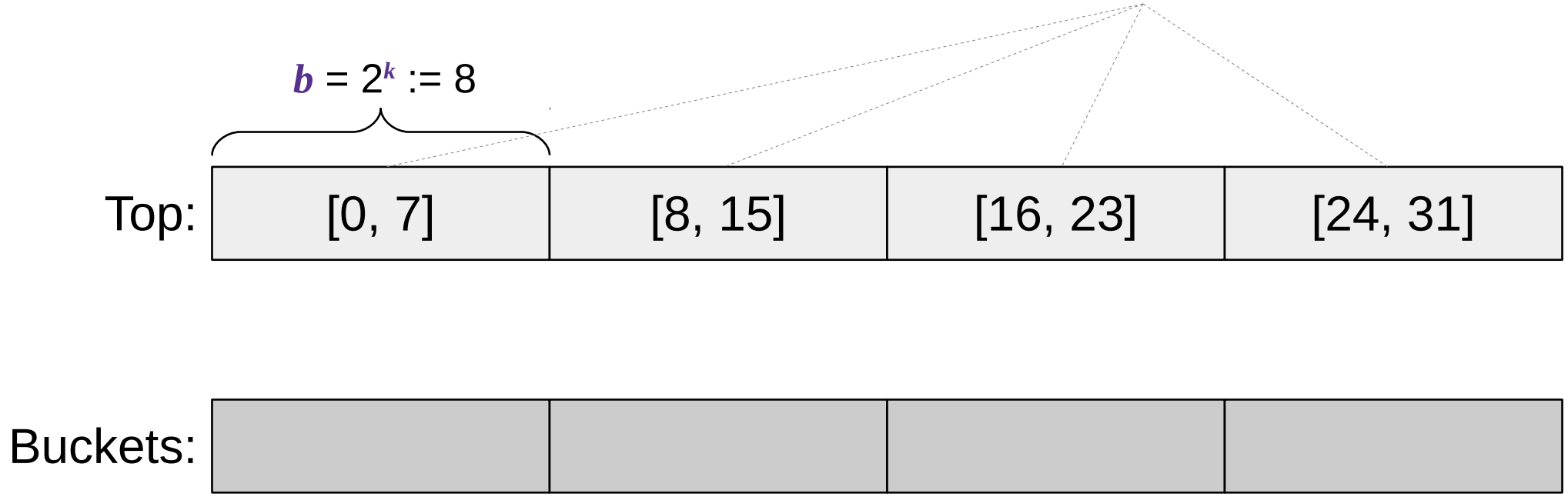
→ We provide an open source implementation

→ **Benchmark for 32-, 40- and 64-bit keys**

↙ **New fastest and most memory efficient data structures**

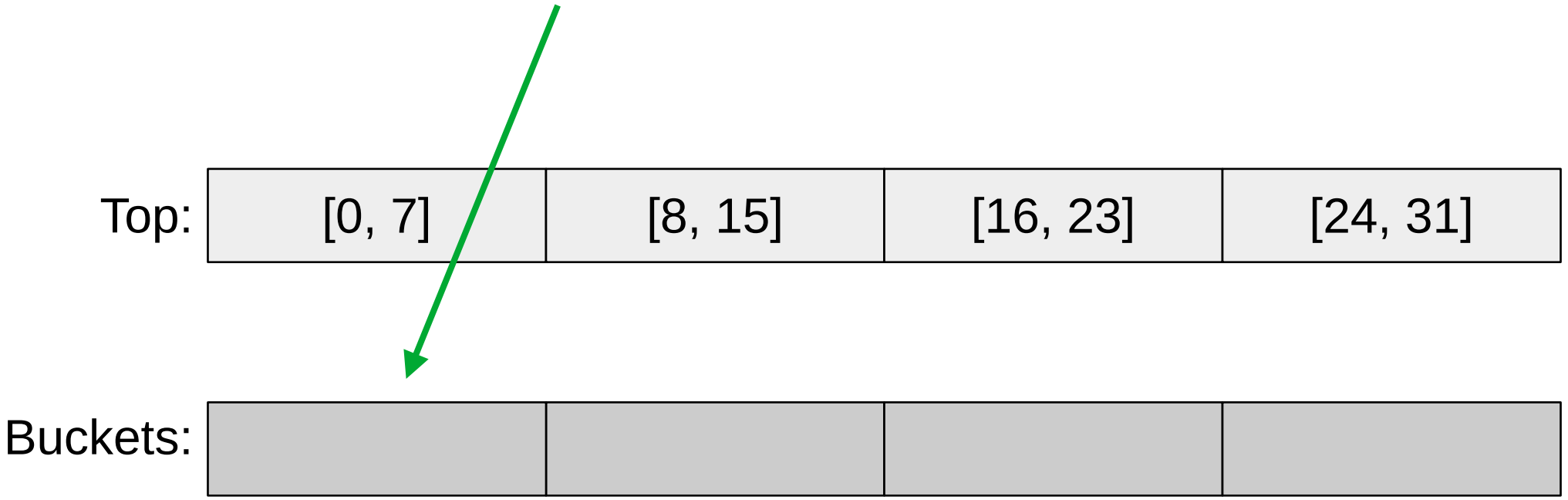
Dynamic Universe Sampling

$S := \{ 4, 19, 1, 21, 7, 2, 6 \}$, $U := [2^5]$



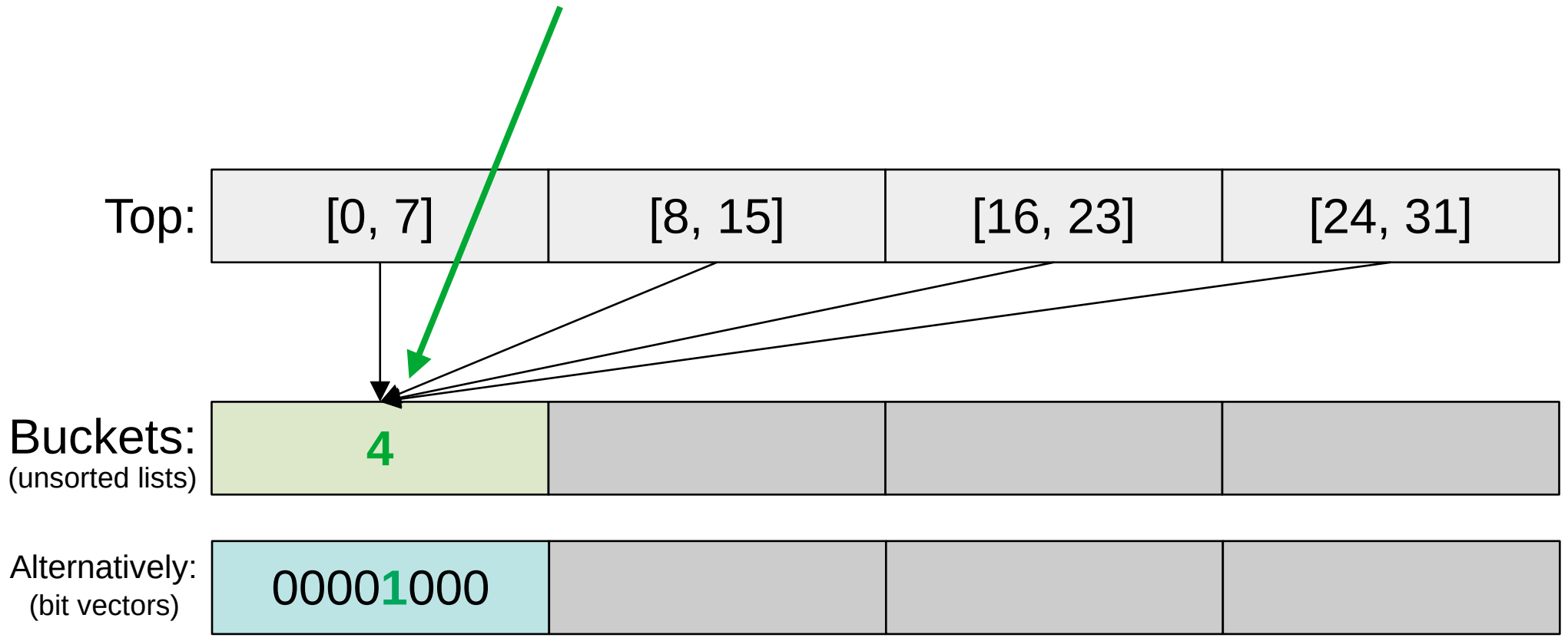
Dynamic Universe Sampling

S := { 4, 19, 1, 21, 7, 2, 6 }, U := $[2^5]$



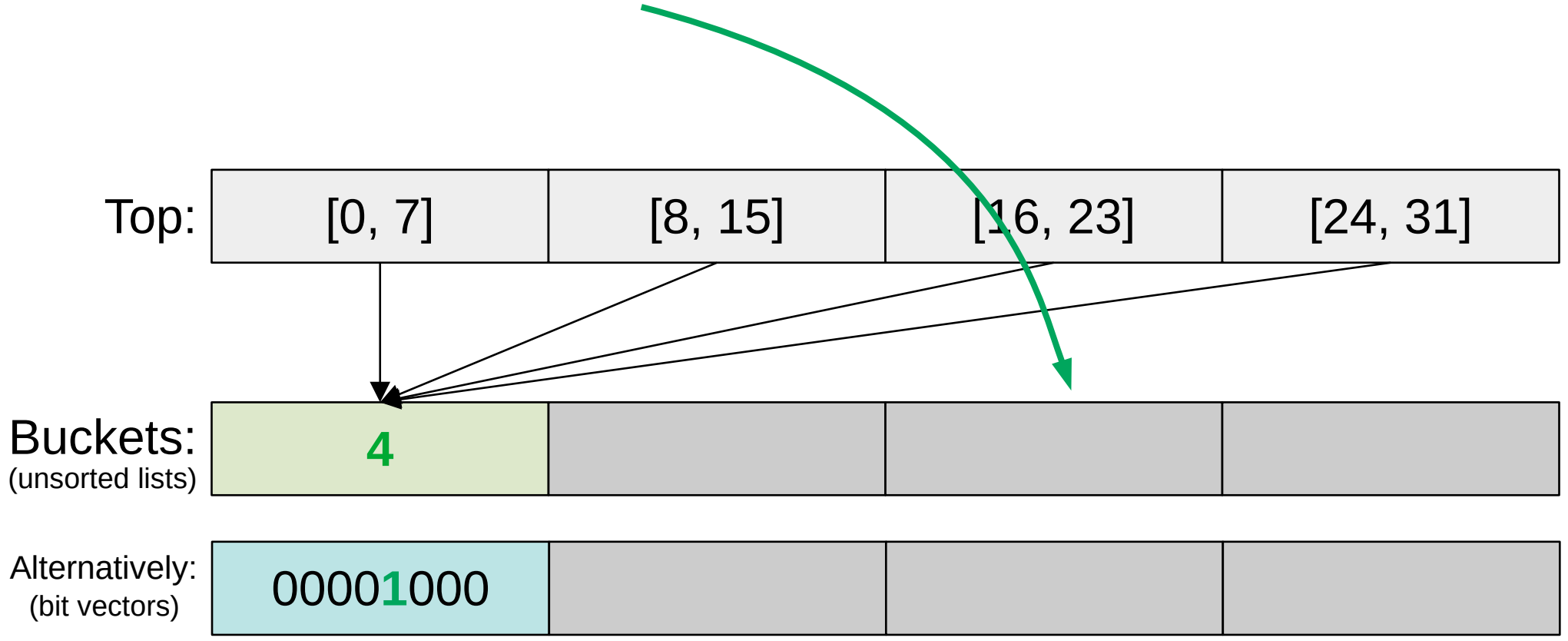
Dynamic Universe Sampling

S := { 4, 19, 1, 21, 7, 2, 6 }, U := $[2^5]$



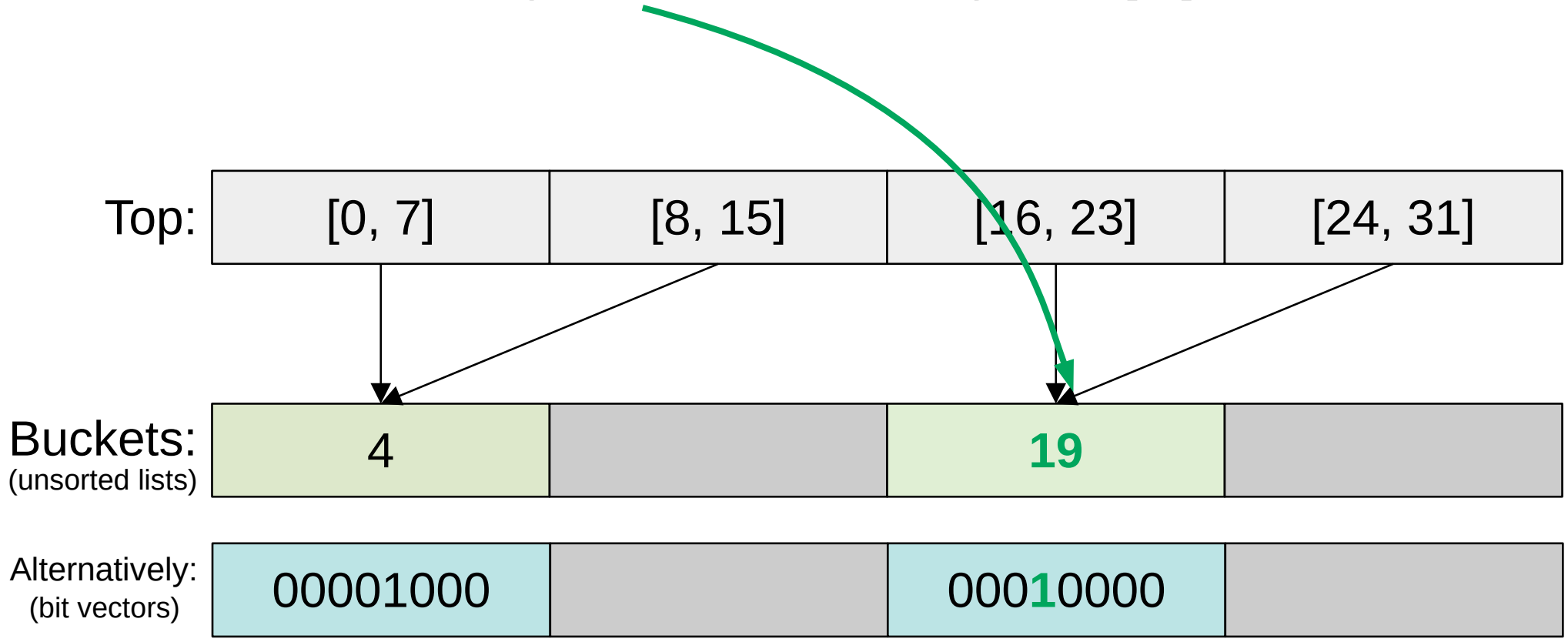
Dynamic Universe Sampling

S := { 4, **19**, 1, 21, 7, 2, 6 }, U := $[2^5]$



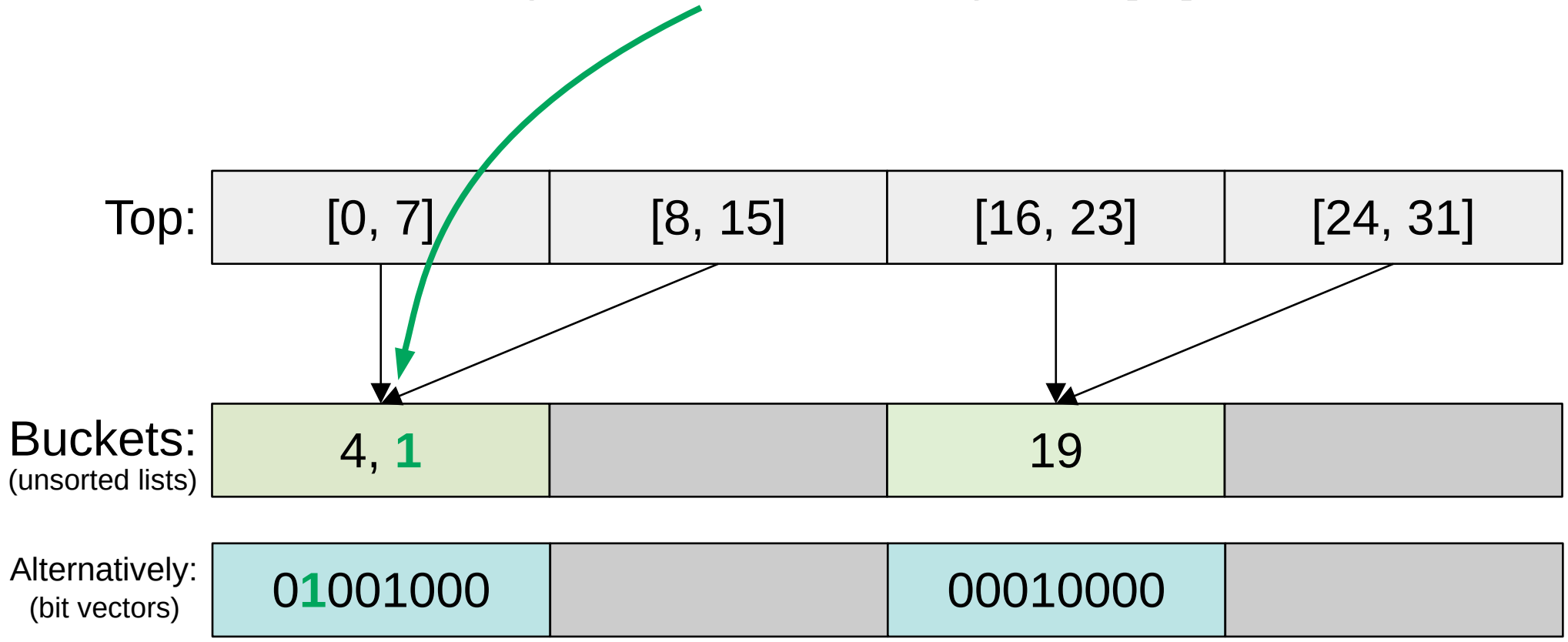
Dynamic Universe Sampling

S := { 4, **19**, 1, 21, 7, 2, 6 }, U := $[2^5]$



Dynamic Universe Sampling

S := { 4, 19, **1**, 21, 7, 2, 6 }, U := $[2^5]$



Dynamic Universe Sampling

- **Top Level**: Predecessor search in universe $[u/b]$

Top Level	Lookup Time	Update Time	Bits
Array	$O(1)$	$O(u/b)$	$(u/b)\lg(u/b)$
Hash Table	$O(u/b)$ exp.	$O(1)$ exp.	$b' \lg(u/b)$

$b' \triangleq$ # of active buckets

Dynamic Universe Sampling

- Top Level: Predecessor search in universe $[u/b]$
- **Bucket Level**: Predecessor search in universe $[b]$

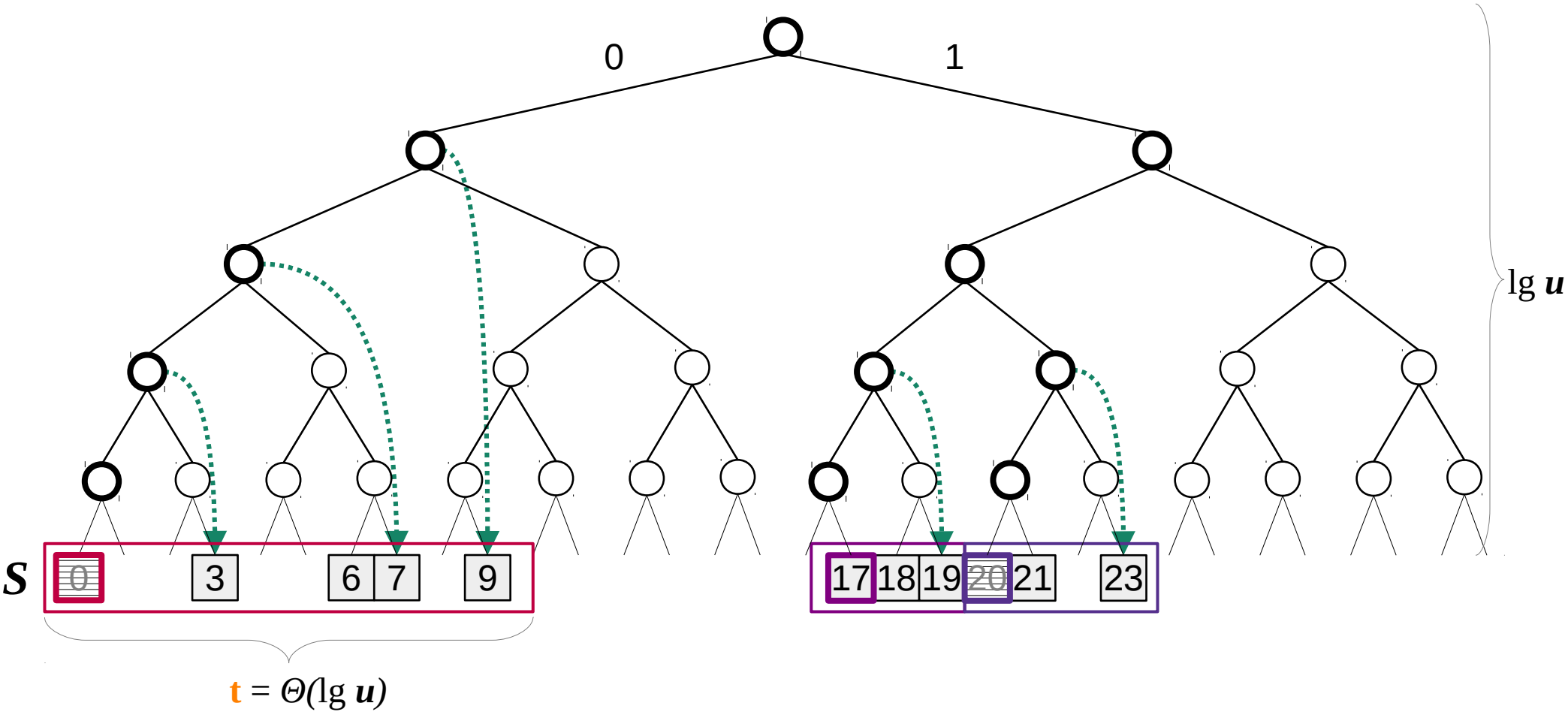
Bucket Level	Bits
Unsorted	$n' \lg b$
Bit Vector	b
Hybrid	$\leq b$

$n' \triangleq$ # of contained items

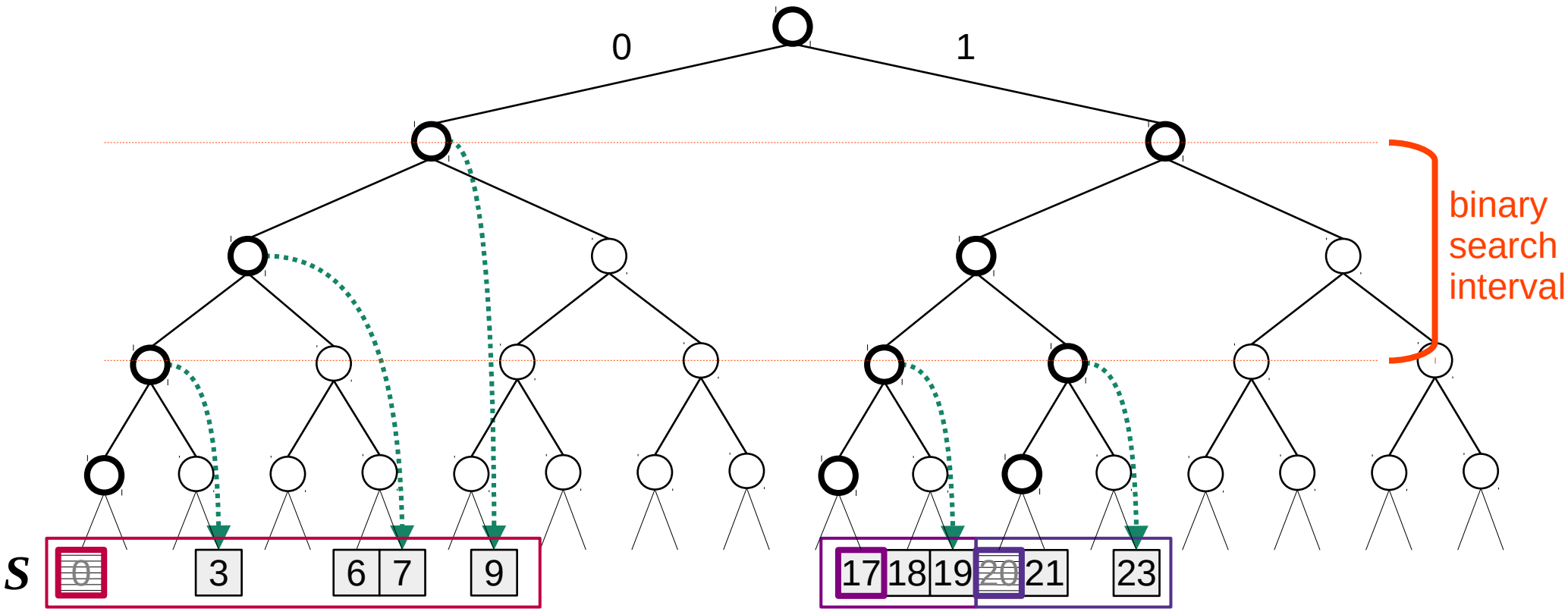
Dynamic Universe Sampling

- Keep b small s.t. buckets fit into few cache lines
- Aimed at small u
- **Best overall setups in our experiments**:
  - b=2^{16} with hybrid buckets (thresholds 2^9 / 2^{10})
  - similar results for array vs. hash table in top level

Y-Fast Tries



Y-Fast Tries



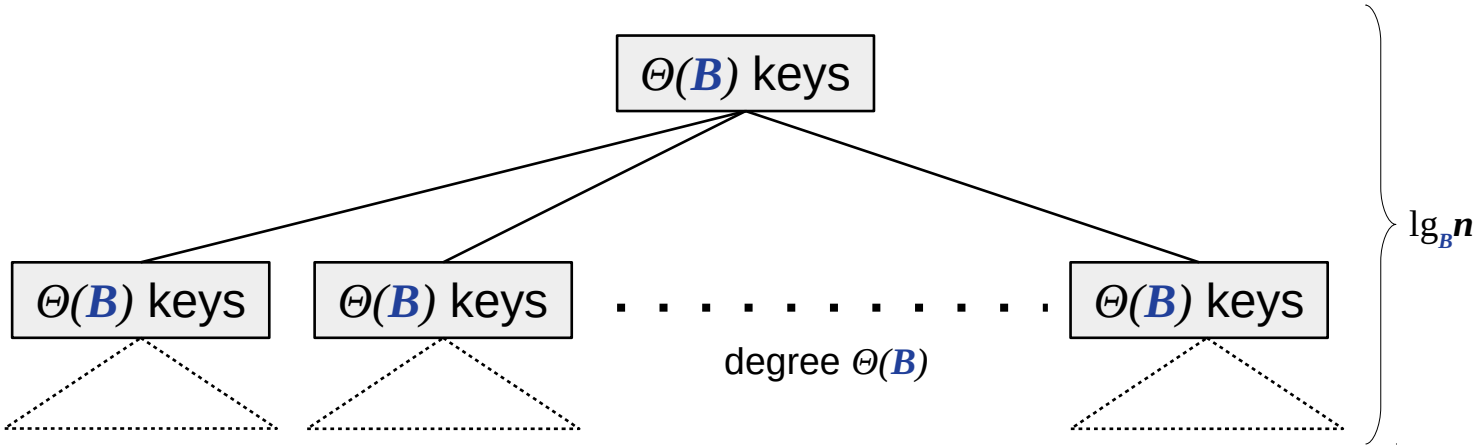
Y-Fast Tries

- **Buckets**: Lists rather than search trees to save memory
  - Sorted lists using binary search
    - Search time $O(\lg\lg \boldsymbol{u})$
    - Order has to be maintained
  - Unsorted lists using linear search
    - Search time $O(\lg \boldsymbol{u})$

Y-Fast Tries

- Keep t small s.t. buckets fit into few cache lines
- **Best overall setups in our experiments**:
  - t = 64 unsorted
  - t = 512 sorted or unsorted

B-Trees & Fusion Trees



Nodes	Pred. Time
Sorted, linear search	$O(B \lg_B n)$
Sorted, binary search	$O(\lg B \lg_B n)$
Fusion	$O(\lg_B n)$

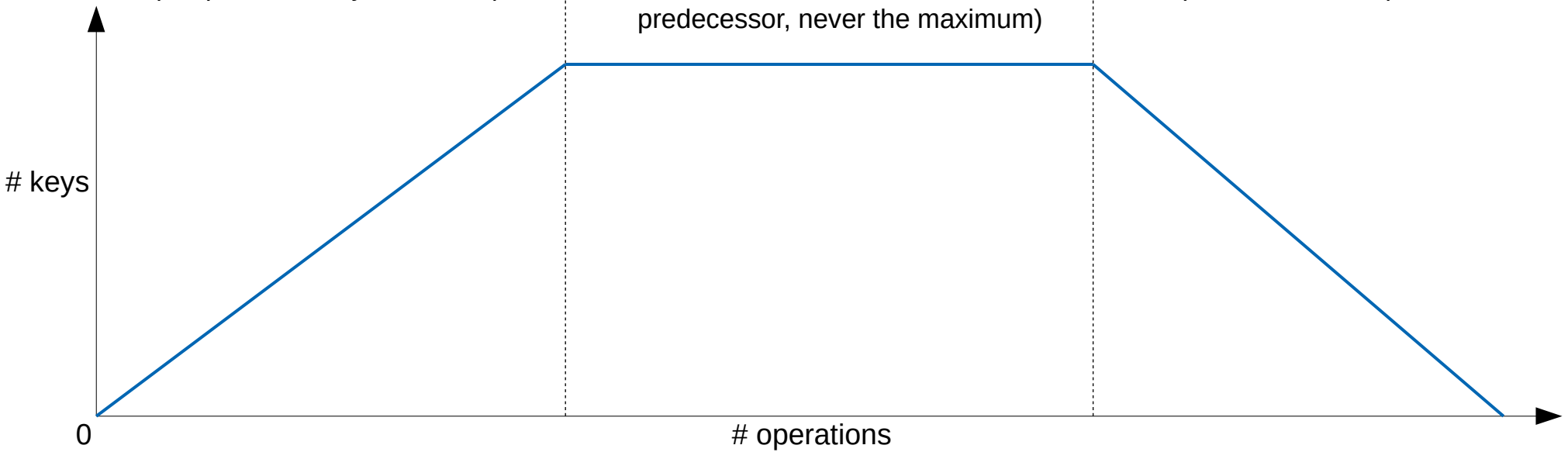
[Pătraşcu & Thorup, FOCS 2014]

B-Trees & Fusion Trees

- Keep B small s.t. nodes fit into few cache lines
- **Best overall setups in our experiments**:
  - B = 64, sorted nodes with linear search
  - B = 128, sorted nodes with binary search
  - (Fusion trees allowed only B = 8 or B = 16 and were slow)

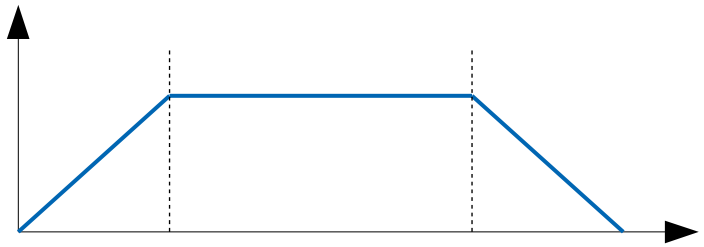
Experimental Setup

- Insert n keys from U
  (unique, uniformly at random)
- Perform 10M queries
  (in input range s.t. there is always a predecessor, never the maximum)
- Delete keys
  (in insertion order)



- Other distributions / operation orders created similar results

Experimental Setup



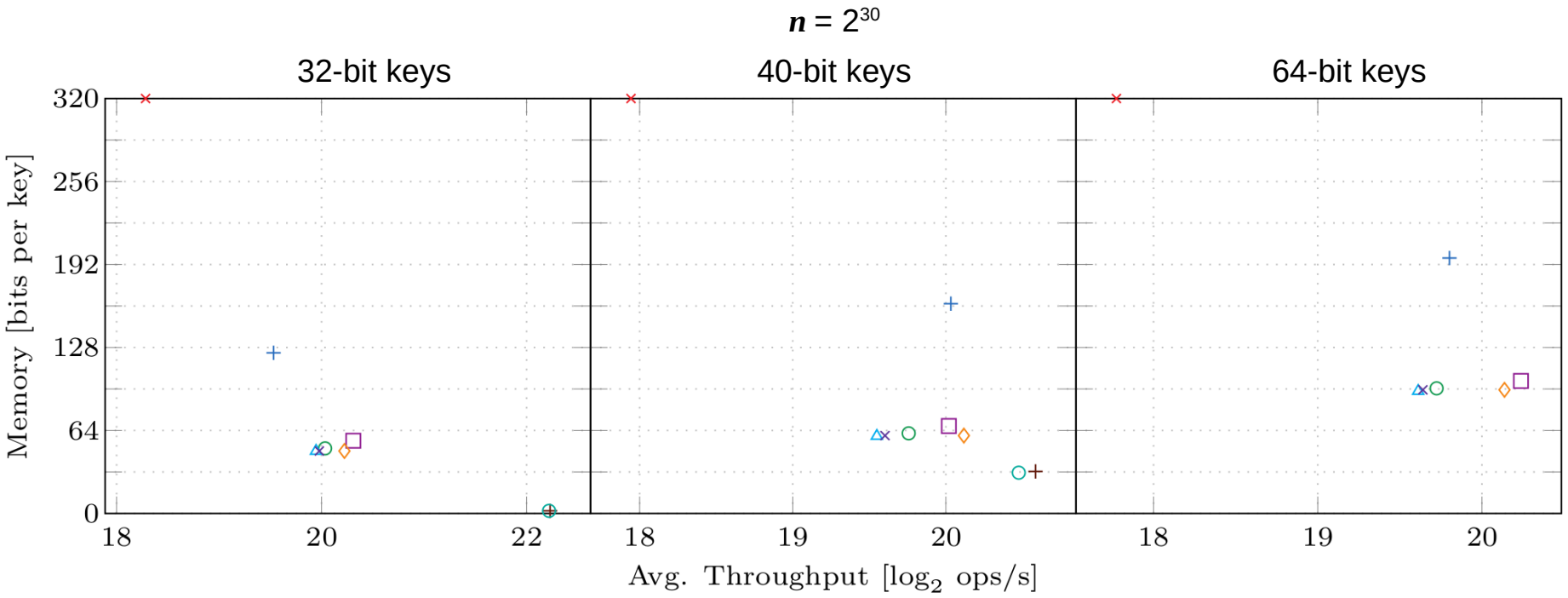
- Measure avg. **throughput** and **memory peak** for 5 seeds
- For comparison: Average of throughputs of phases

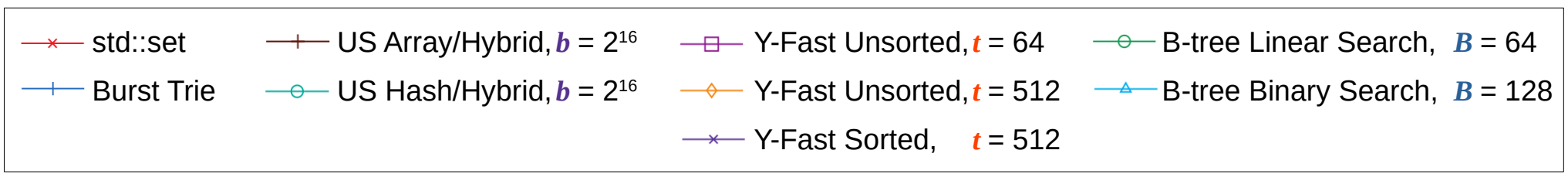
- Experiments run on Linux machine with Intel Xeon E5-4640v4 processor
  - 2.1 GHz with 32kB L1, 256 kB L2
  - 30MB L3 shared, cache line size 64B, 256 GB of memory

Comparison

- Best setups of **our implementations**
- **Burst Trie** of *[Nash & Gregg, WEA 2008]*
  - → Fastest known in practice thus far, *associative*
- STL **std::set** (red-black trees)
- Stratified Trees *[Dementiev et al., ALENEX 2004]*
  - ✗ (exceeded time and memory limits already for small inputs)

Comparison


$n = 2^{30}$
32-bit keys
40-bit keys
64-bit keys
Memory [bits per key]
320
256
192
128
64
0
18
20
22
18
19
20
18
19
20
Avg. Throughput [log2 ops/s]


std::set
Burst Trie
US Array/Hybrid, b = 2^16
US Hash/Hybrid, b = 2^16
Y-Fast Unsorted, t = 64
Y-Fast Unsorted, t = 512
Y-Fast Sorted, t = 512
B-tree Linear Search, B = 64
B-tree Binary Search, B = 128

Conclusions & Further Work

- **New fastest and most memory efficient practical dynamic predecessor data structures**
- Naïve algorithms (e.g., linear scans) outperform sophisticated data structures on small instances
  - Confirms results of other work

- Pure SIMD implementation of Fusion Nodes / Trees?
  - (our implementation: SIMD instructions embedded in a SISD frame)

Thank you for your attention!