Lecture 4:
vEB Trees (ctd.)
Fusion Trees

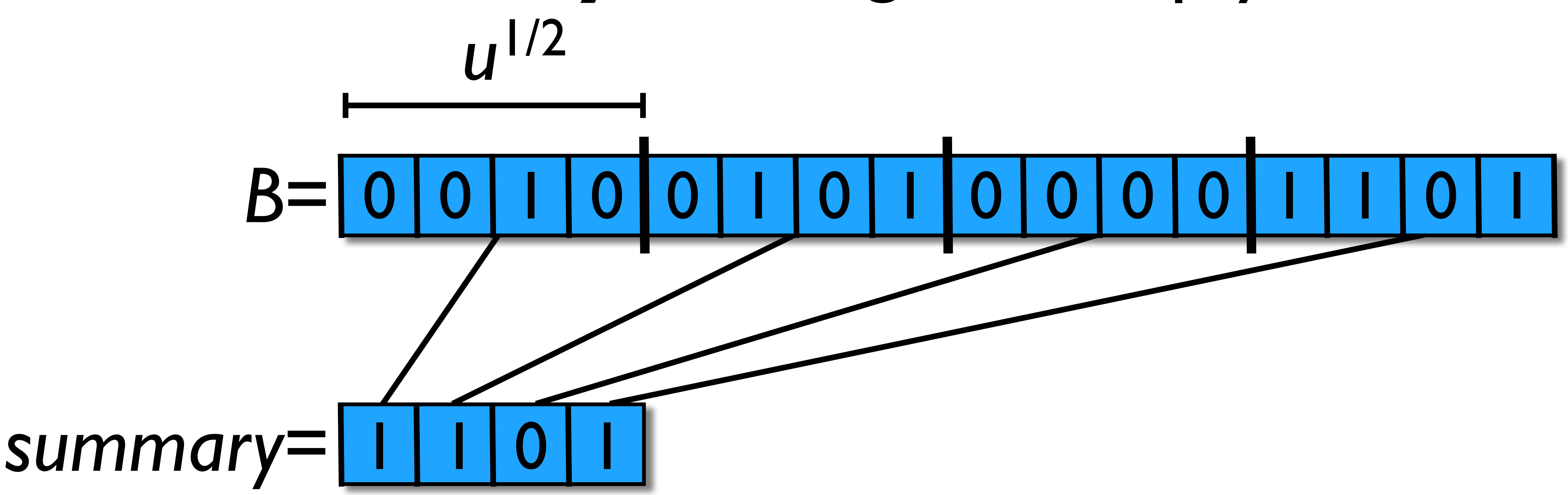
Johannes Fischer

van Emde Boas Trees

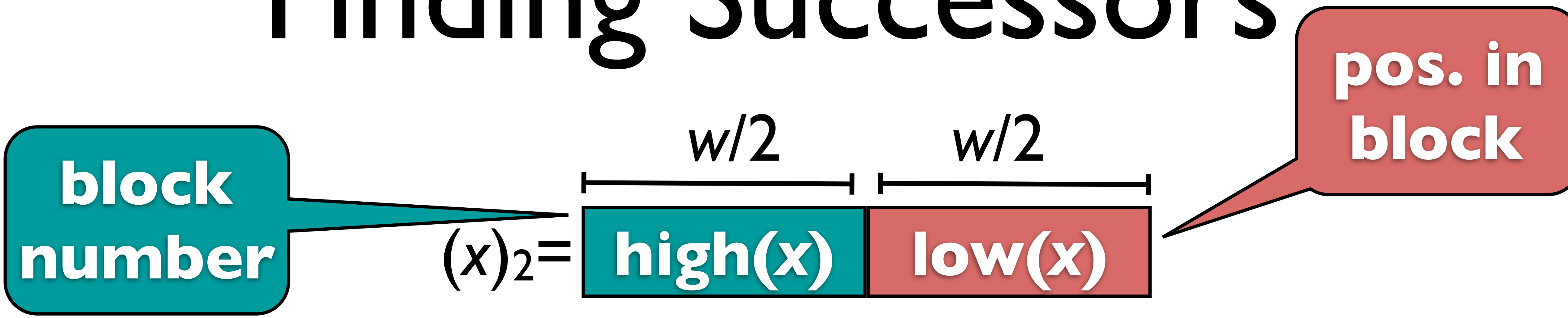
vEB trees	**dynamic**	
pred/succ	$O(\lg w)$ w.c.	$O(\lg w)$ w.c.
insert/delete	$O(\lg w)$ w.c.	$O(\lg w)$ exp. & amort.
space	$O(u)$ w.c.	$O(n)$ w.c.

Idea

- **bit vector** B marking members of S
- $u^{1/2}$ blocks $B_0, B_1, \ldots$ of length $u^{1/2}$ ($= 2^{w/2}$)
  - block(x)= $\lfloor x/u^{1/2} \rfloor$ = upper $w/2$ bits of $(x)_2$
- **summary** marking non-empty blocks



Finding Successors



function succ(B, x):

  inblock-succ ← succ($B_{\text{high}(x)}$, low(x))

  if (*inblock-succ* ≠ ⊥)

    return *inblock-succ* + (high(x) × $B.u^{1/2}$)

  else

    succ-block ← succ(B.summary, high(x))

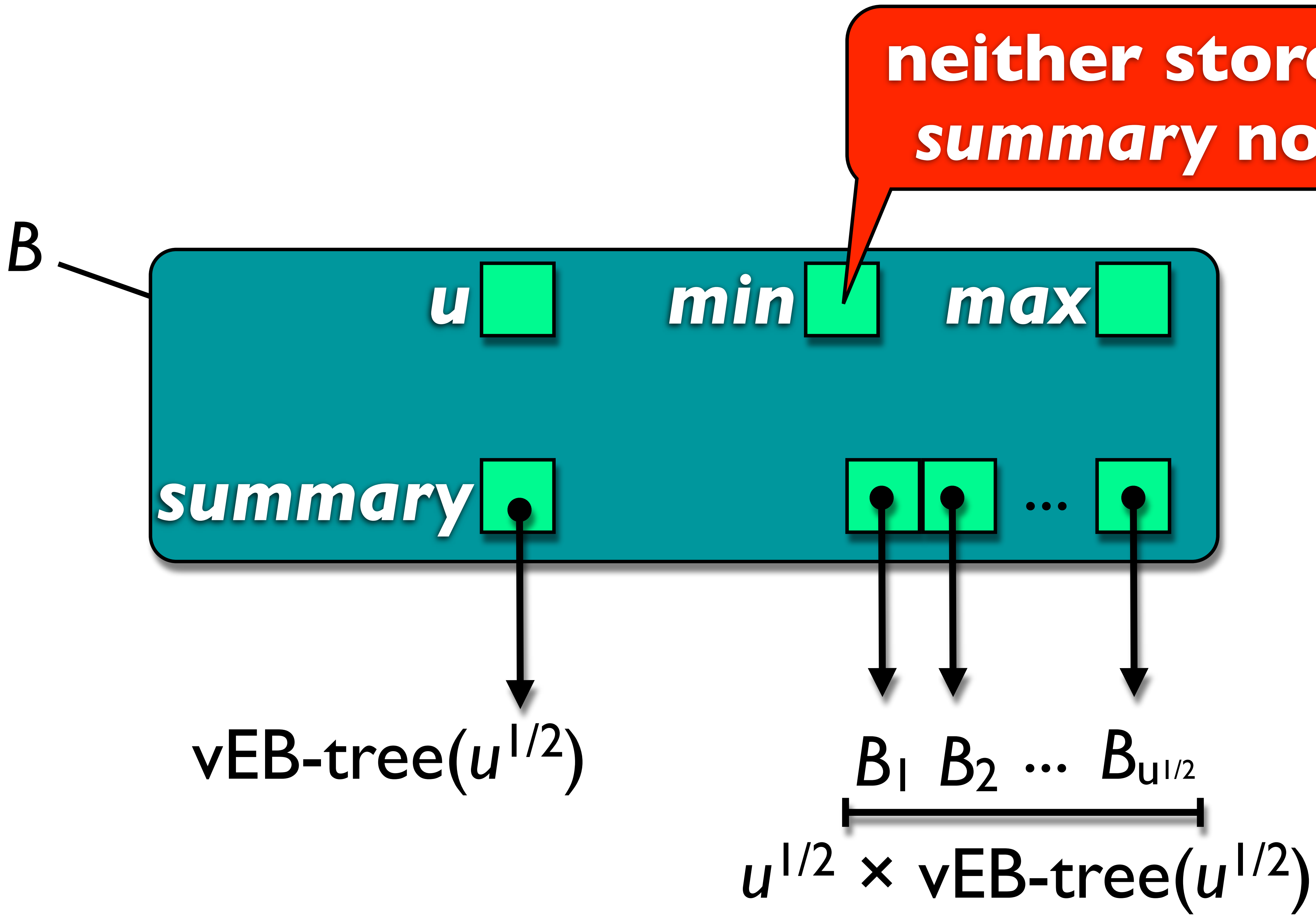
    if (*succ-block* = ⊥) **return** ⊥

    return min($B_{\textit{succ-block}}$) + (*succ-block* × $B.u^{1/2}$)

Running Time

- base case if $B.u=2$
- $T(u) = 2T(u^{1/2}) + O(1)$
  $= \Theta(\lg u)$
- Too **slow**!
- Modify for only **one** recursive call
  - $T'(u) = T'(u^{1/2}) + O(1)$
    $= \Theta(\lg\lg u)$
- **Idea**: storing also **max** saves 1 recursion

vEB Tree Node



Successor Revisited

```
function succ(B, x):
    if (min(B) ≠ ⊥ and x < min(B)) return min(B)
    else
        block-max ← max(B_high(x))
        if (block-max ≠ ⊥ and low(x) < block-max)
            inblock-succ ← succ(B_high(x), low(x))
            return inblock-succ + (high(x) × B.u^1/2)
        else
            succ-block ← succ(B.summary, high(x))
            if (succ-block = ⊥) return ⊥
            return min(B_succ-block) + (succ-block×B.u^1/2)
```

Insertions

function insert(B, x):
    if (min(B) = ⊥) min(B) ← max(B) ← x
    else
        if (x < min(B)) swap x with min(B)
        if (min($B_{high(x)}$) = ⊥)
            insert(B.summary, high(x))
        insert($B_{high(x)}$, low(x))
        if (max(B) < x) max(B) ← x

Space

recursive structures — pointers

- $S(u) = \underbrace{(1+u^{1/2}) \times S(u^{1/2})}_{\text{recursive structures}} + \underbrace{\Theta(u^{1/2})}_{\text{pointers}}$
  $= \Theta(u)$
- space $\Theta(n)$:
  - store only non-empty blocks recursively
  - use hash tables!
  - summary only if ≥ 1 non-empty block

Fusion Trees

- S static, U = [0,u] = [0,2^w-1]
  - predecessor/successor O(lg n / lg w) time
- M.L. Fredman, D. E. Willard [J. Comput. Syst. Sci. 1993]

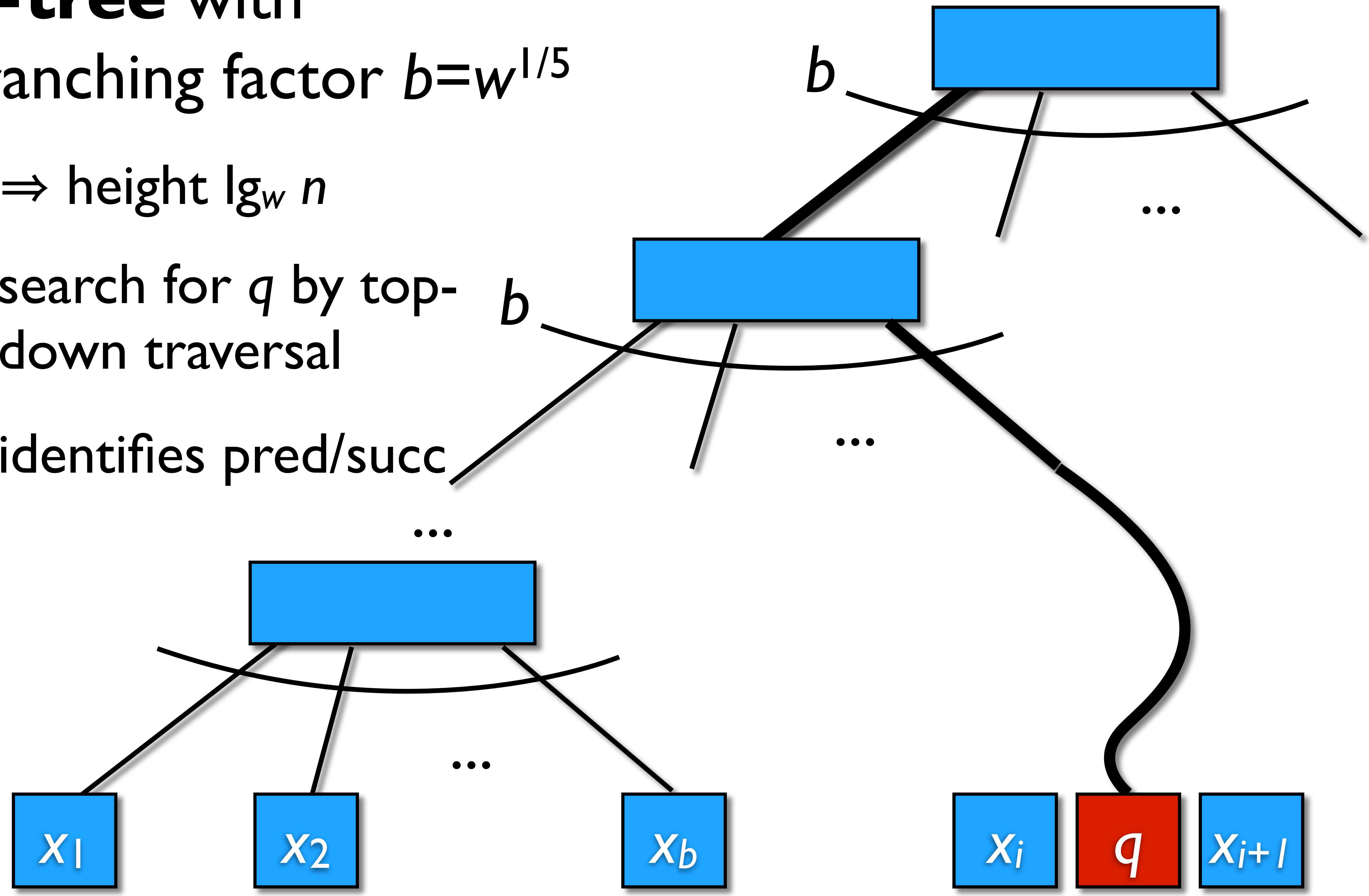
fusion tree	
pred/succ	O(lg n / lg w) w.c.
construction	$O(n)$ w.c. + SORT(n,w)
space	$O(n)$ w.c.

Comparison

- **fusion trees:**
  - O(lg n / lg w) time good if n small w.r.t. w
- **van Emde Boas trees:**
  - O(lg w) time good if w small w.r.t. n
- **equal** when $w = 2^{\sqrt{\lg n}}$
  - $\lg^{1/2} n$ time **independent** of w
  - better than "traditional" search trees

Idea

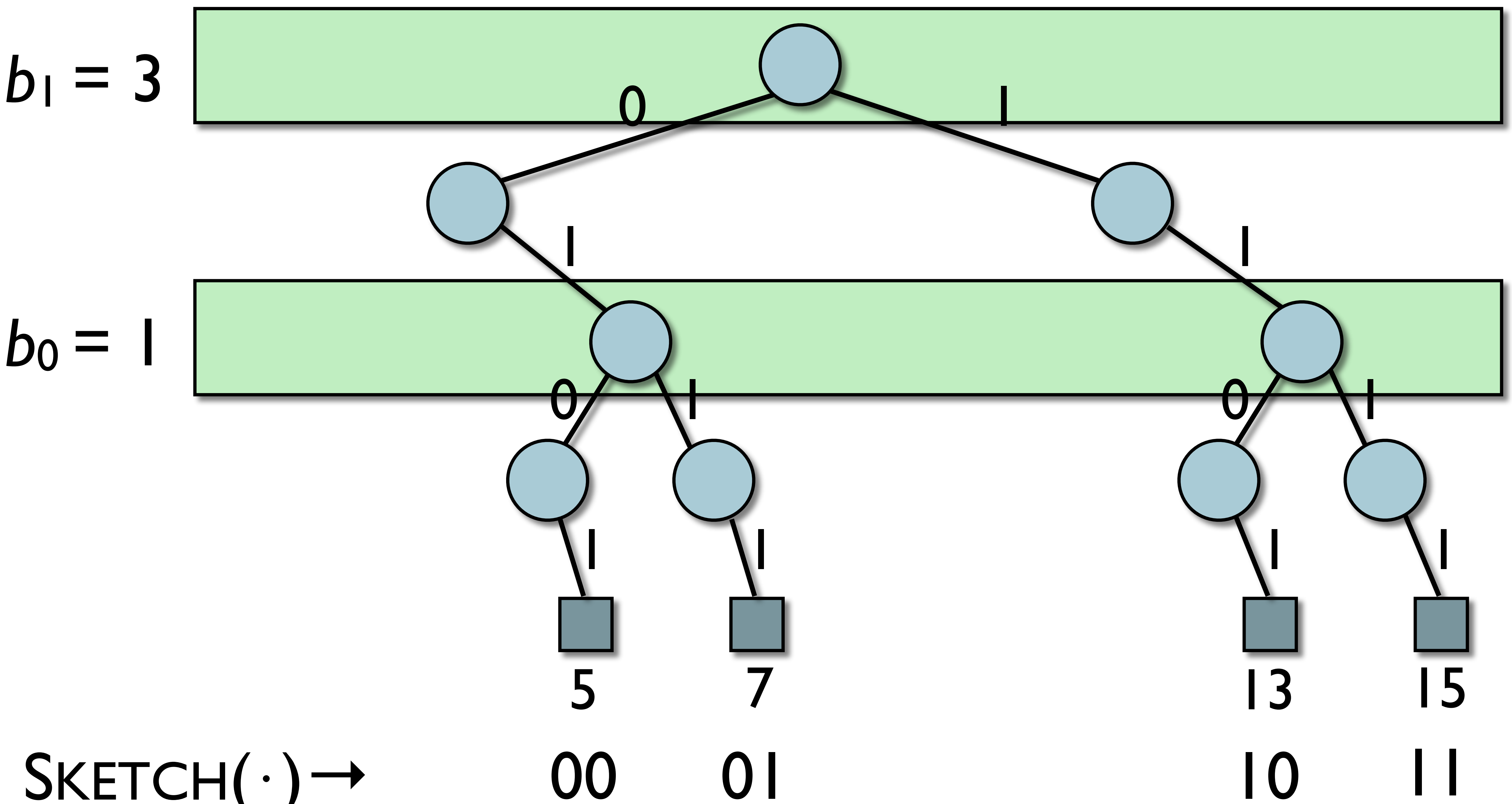
- **B-tree** with branching factor $b = w^{1/5}$
  - ⇒ height $\lg_w n$
  - search for q by top-down traversal
  - identifies pred/succ



Node Sketches

- spend only **O(1) time** per B-tree node
  - $O(w^{6/5})$ bits cannot be read
  - assume $w \leq$ word size
- need to **compress** keys
  - possible since w large (w.r.t. #keys/node = $w^{1/5}$)
- **important bits** $b_0 < \ldots < b_{r-1}$ for $x_0 < \ldots < x_{b-1}$: levels with **branching** nodes in trie
- **sketch** of x_i = restricted to important bits

Example



Observe

- $r \leq b = O(w^{1/5})$
  $\Rightarrow$ all b sketches can be **fused** into
  one *w*-word: $b \times r \leq w^{2/5}$
- SKETCH **preserves order**:
  SKETCH(x_i) < SKETCH(x_j) $\Leftrightarrow$ $x_i < x_j$
- problem: does **not** hold for query *q*:
  SKETCH(x_i) < SKETCH(q) $\leq$ SKETCH(x_{i+1})
  $\not\Rightarrow$ $x_i < q \leq x_{i+1}$

Example