Lecture 4: Fusion Trees

Johannes Fischer

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
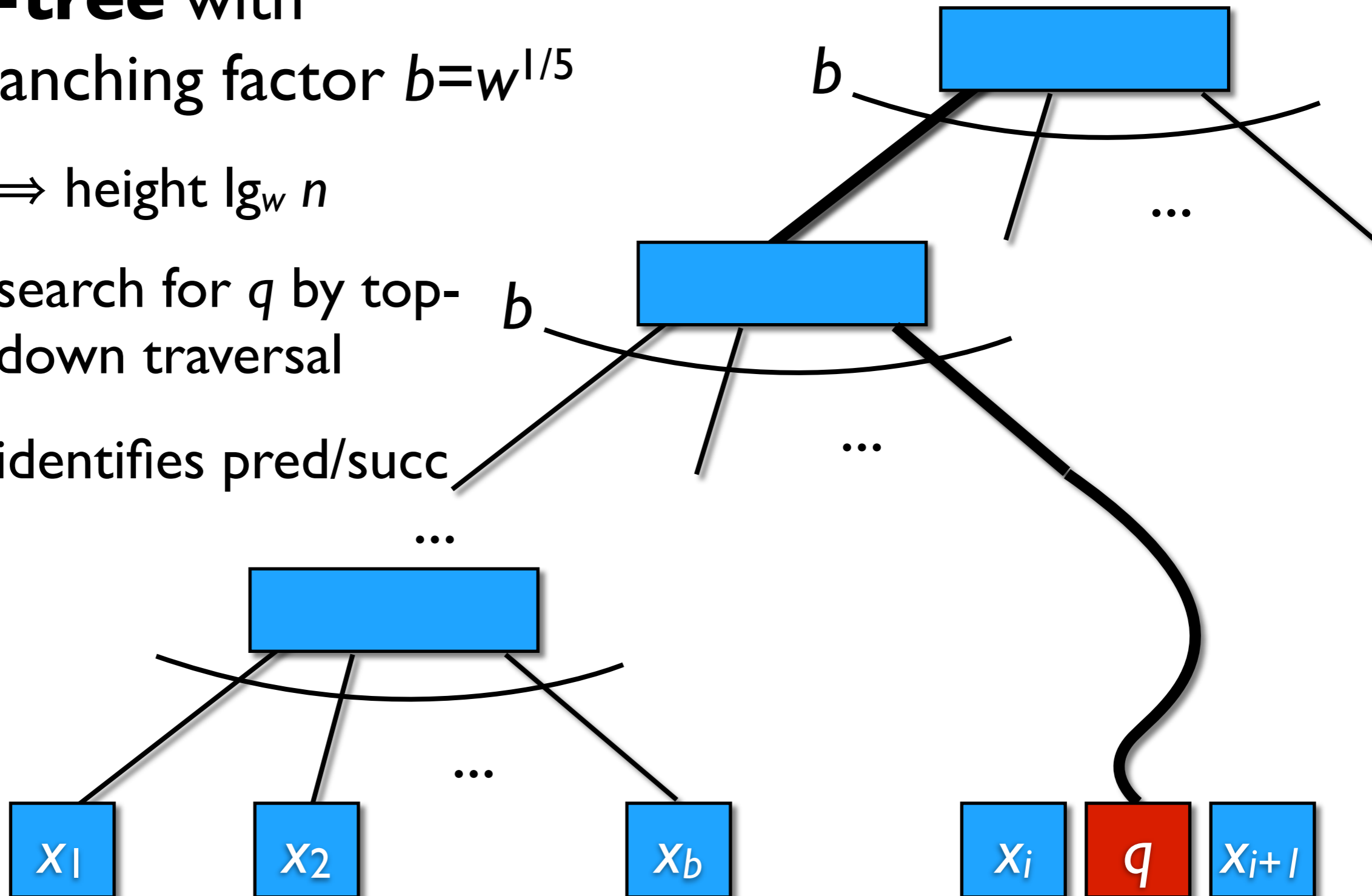
- **equal** when $w = 2^{\sqrt{\lg n}}$

  ▶ $\lg^{1/2} n$ time **independent** of w

  ▶ better than "traditional" search trees

Idea

- **B-tree** with branching factor $b=w^{1/5}$

  ▶ $\Rightarrow$ height $\lg_w n$

  ▶ search for q by top-down traversal

  ▶ identifies pred/succ

b

b

...

...

...

...

x_1 x_2 x_b x_i q x_{i+1}

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

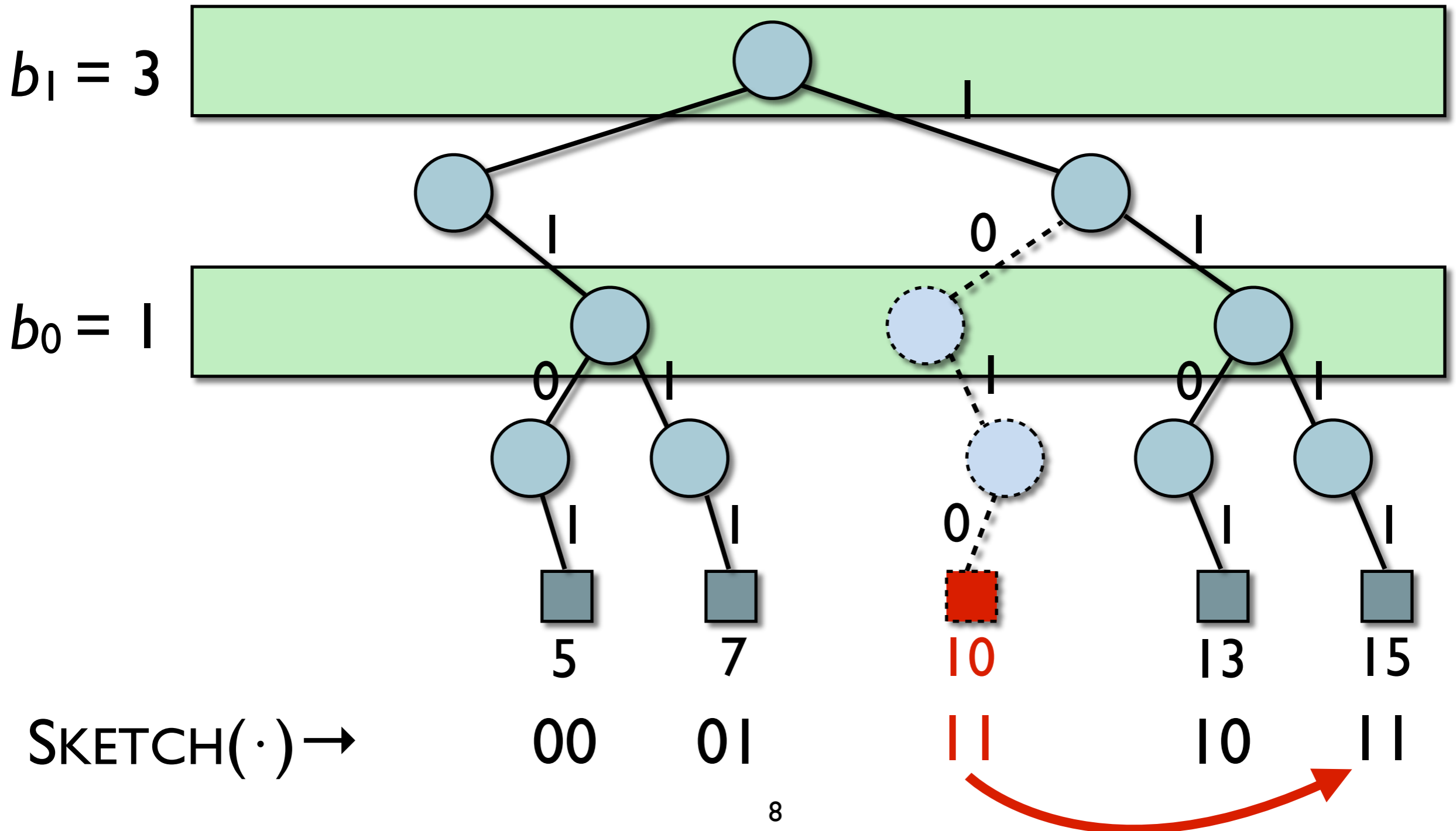- **sketch** of x_i = restricted to important bits

Example

$b_1 = 3$

$b_0 = 1$

SKETCH($\cdot$) $\rightarrow$ 00   01   10   11

6
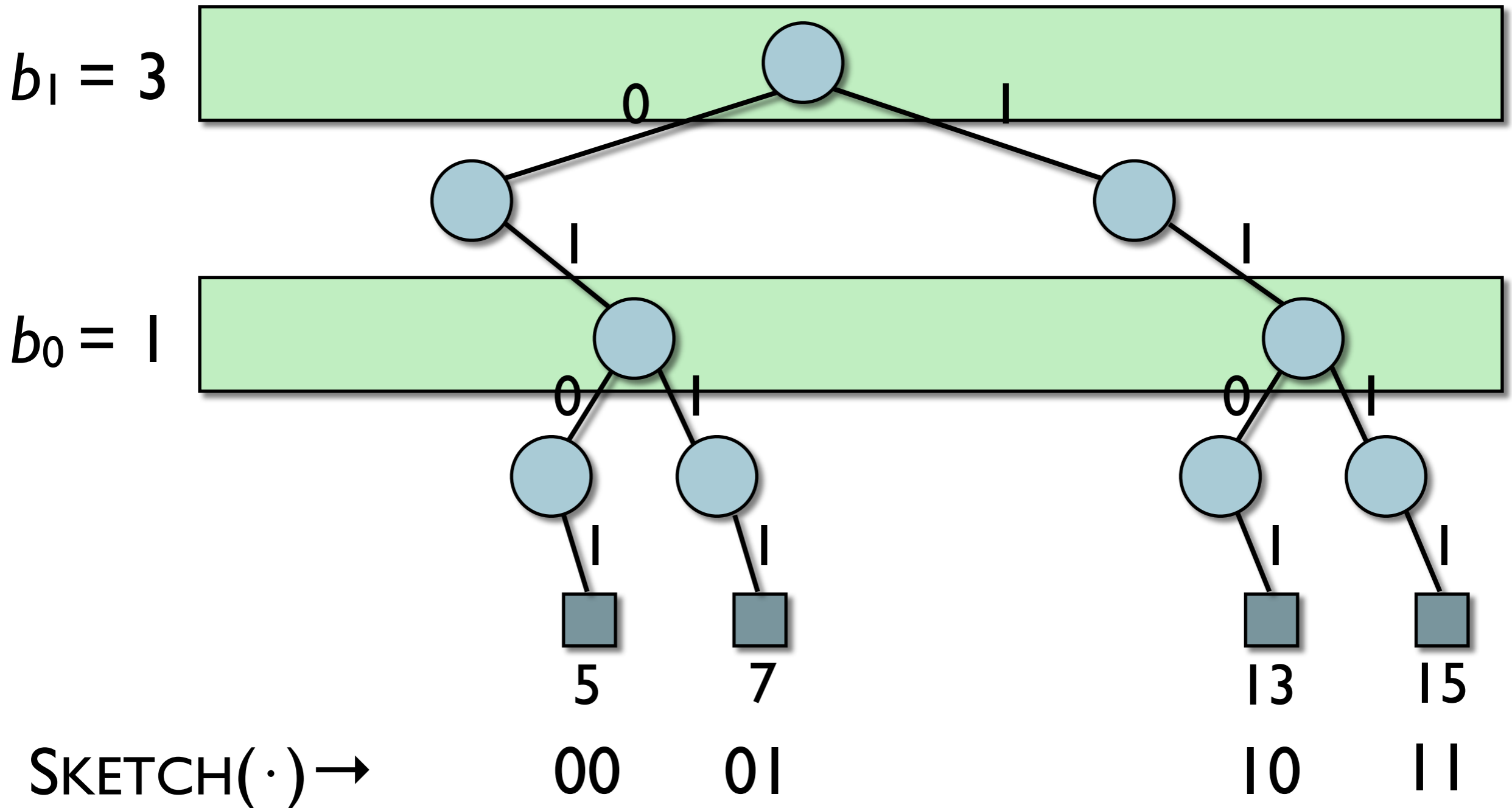
Observe

- $r \leq b = O(w^{1/5})$
  $\Rightarrow$ all b sketches can be **fused** into

  one _w_-word: $b \times r \leq w^{2/5}$

- SKETCH **preserves order**:
  $$\text{SKETCH}(x_i) < \text{SKETCH}(x_j) \Leftrightarrow x_i < x_j$$

- problem: does **not** hold for query q:
  $\text{SKETCH}(x_i) < \text{SKETCH}(q) \leq \text{SKETCH}(x_{i+1})$
  $\not\Rightarrow x_i < q \leq x_{i+1}$

Example

$b_1 = 3$

$b_0 = 1$

SKETCH($\cdot$) →
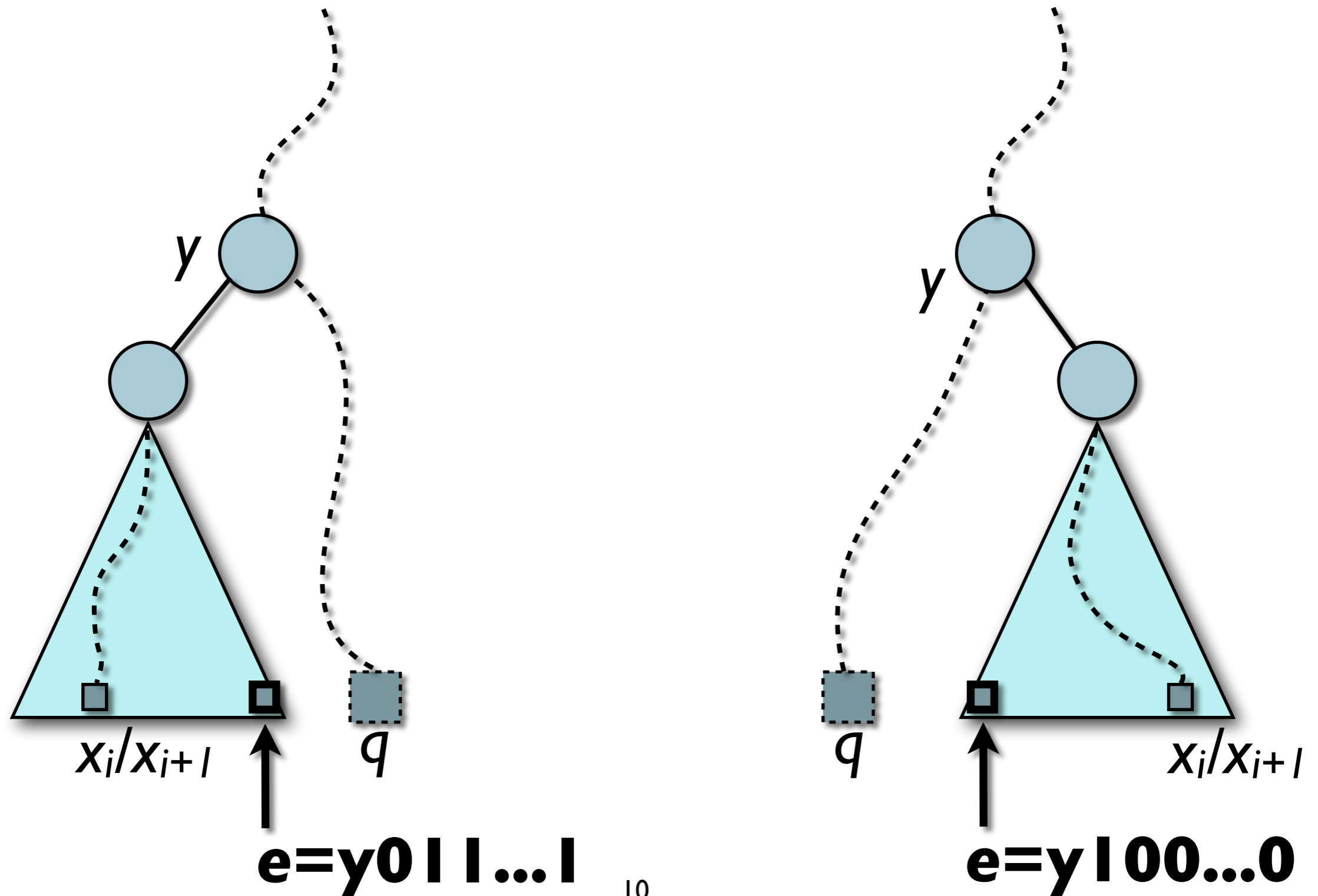
5   7   10   13   15

00   01   11   10   11

Right Insertion Point

- $\text{SKETCH}(x_i) \leq \text{SKETCH}(q) \leq \text{SKETCH}(x_{i+1})$

- $y \leftarrow \max\{\text{LCP}(q,x_i), \text{LCP}(q,x_{i+1})\}$

  ▸ point where q deviates from x_i/x_{i+1}

  ▸ **most significant bit** of q XOR x_i/x_{i+1}

- subtree of y must contain pred **or** succ

  ▸ repeat computation in that subtree

Right Insertion Point

x_i/x_{i+1} q q x_i/x_{i+1}

e=y011...1   **e=y100...0**

Summary

- at every node of the B-tree with keys $x_0 < \ldots < x_{b-1}$ and important bits $b_0 < \ldots < b_{r-1}$

  ▶ compute $\textsc{Sketch}(q)$ in $O(1)$ time

  ▶ find i such that $\textsc{Sketch}(x_i) \leq \textsc{Sketch}(q) \leq \textsc{Sketch}(x_{i+1})$

  TODO 1

  ▶ $y \leftarrow \max\{\textsc{MSB}(q \text{ XOR } x_i), \textsc{MSB}(q \text{ XOR } x_{i+1})\}$

  TODO 2

  ▶ $e \leftarrow q_{w-1}\ldots q_{y-1}011\ldots1$ *or* $q_{w-1}\ldots q_{y-1}100\ldots0$

  ▶ compute $\textsc{Sketch}(e)$ in $O(1)$ time

  TODO 3

  ▶ find i' such that $\textsc{Sketch}(x_{i'}) \leq \textsc{Sketch}(e) \leq \textsc{Sketch}(x_{i'+1})$

  ▶ then $x_{i'}$ or $x_{i'+1}$ is predecessor or successor of q

TODO 1:
MSB in $O(1)$ time

- possible, but **complicated** if not supported by CPU:

$$t_1 \leftarrow h \,\&\, (x \mid ((x \mid h) - l)), \quad \text{where } h = 2^{g-1}l \text{ and } l = (2^n - 1)/(2^g - 1);$$

$$y \leftarrow (((a \cdot t_1) \bmod 2^n) \gg (n - g)) \cdot l, \quad \text{where } a = (2^{n-g} - 1)/(2^{g-1} - 1);$$

$$t_2 \leftarrow h \,\&\, (y \mid ((y \mid h) - b)), \quad \text{where } b = (2^{n+g} - 1)/(2^{g+1} - 1);$$

$$m \leftarrow (t_2 \ll 1) - (t_2 \gg (g - 1)), \; m \leftarrow m \oplus (m \gg g);$$

$$z \leftarrow (((l \cdot (x \,\&\, m)) \bmod 2^n) \gg (n - g)) \cdot l;$$

$$t_3 \leftarrow h \,\&\, (z \mid ((z \mid h) - b));$$

$$\lambda \leftarrow ((l \cdot ((t_2 \gg (2g - \lg g - 1)) + (t_3 \gg (2g - 1)))) \bmod 2^n) \gg (n - g).$$

© D.E. Knuth: *The Art of Computer Programming*, Vol. 4A. Addison-Wesley, 2011.

- note the use of 5 **multiplications**

MSB in $O(\lg w)$ time

- idea:

  ▶ binary search over x

  ▶ with ANDs determine if search interval empty

$x=$ | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 |

$mask=$ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

$x\ AND\ mask=$ | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | $\neq 0 \Rightarrow$ continue in left half

MSB in $O(\lg w)$ time

- idea:

  ▸ binary search over x

  ▸ with ANDs determine if search interval empty

$x=$ | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 |

$mask=$ | 1 | 1 | 1 | 1 |

$x\ AND\ mask=$ | 0 | 0 | 0 | 0 | $=0 \Rightarrow$ continue in right half

MSB in $O(\lg w)$ time

- idea:

  ▸ binary search over x

  ▸ with ANDs determine if search interval empty

$x=$ | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 |

$mask=$ | 1 | 1 |

$x\ AND\ mask=$ | 0 | 1 | $\neq 0 \Rightarrow$ continue in left half

etc.

MSB in $O(\lg w)$ time

- neat variation of this idea:

function MSB(x):
    $\lambda \leftarrow 0$
    for $k=\lg(w)\text{-}1$ **downto** 0
        $z \leftarrow x \gg 2^k$
        if ($z \neq 0$)
            $\lambda \leftarrow \lambda + 2^k$
        $x \leftarrow z$
    return λ

- possible: **table lookup** for small blocks

Implementation

```
unsigned int x;   // 32-bit value to find the log2 of
const unsigned int b[] = {0x2, 0xC, 0xF0, 0xFF00,
                          0xFFFF0000};
const unsigned int S[] = {1, 2, 4, 8, 16};
int i;

register unsigned int l = 0; // result will go here
for (i = 4; i >= 0; i--) // unroll for speed...
{
  if (x & b[i])
  {
    x >>= S[i];
    l |= S[i];
  }
}
```
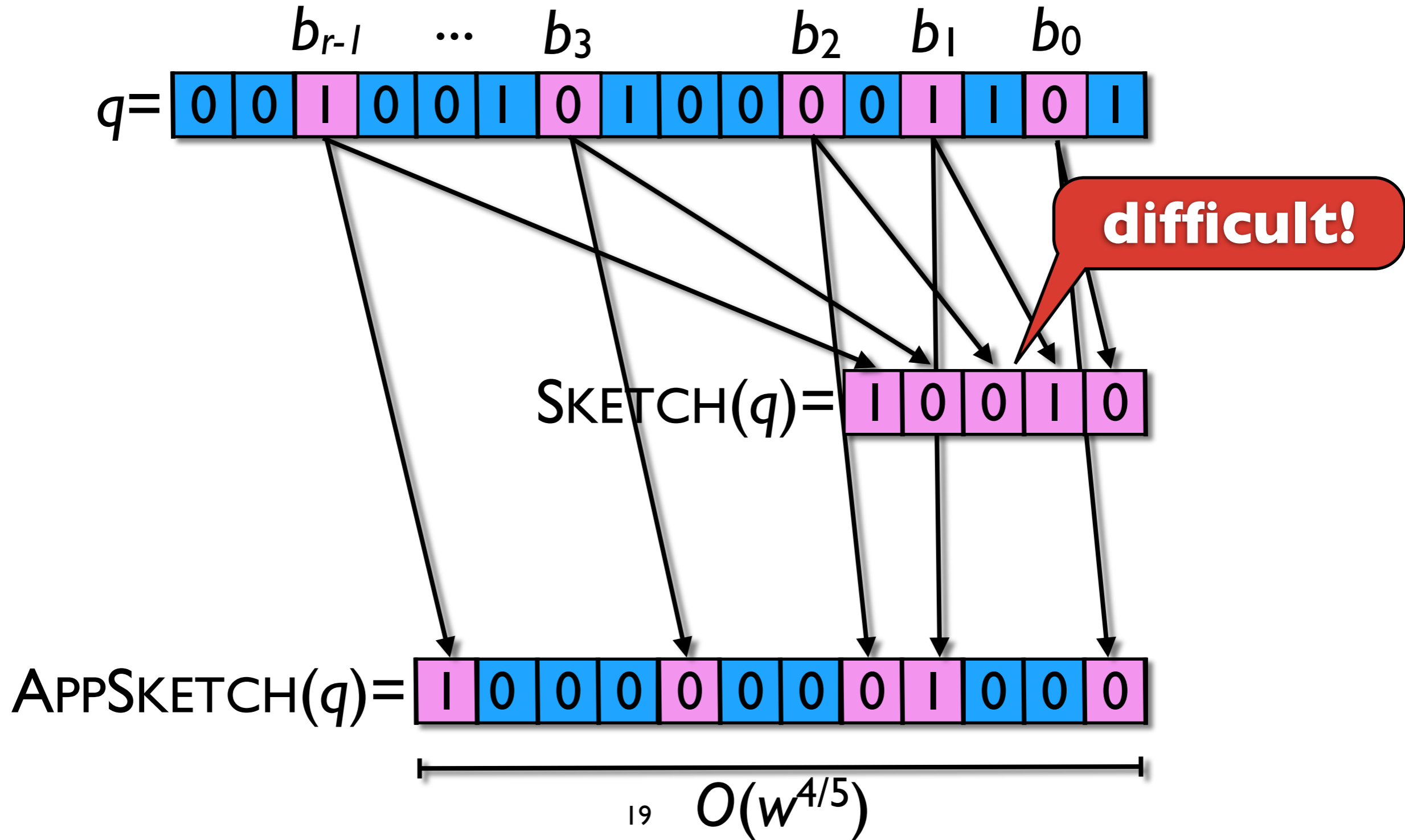
Implementation with Table Lookup

```
unsigned int l = 0;              // will be lg(x)
register unsigned int t, tt; // temporaries
if (tt = x >> 16)
    l = (t = x >> 24) ? 24 + LogTable256[t]
                      : 16 + LogTable256[tt & 0xFF];
else
    l = (t = x >> 8) ? 8 + LogTable256[t] : LogTable256[x];
return l;
```
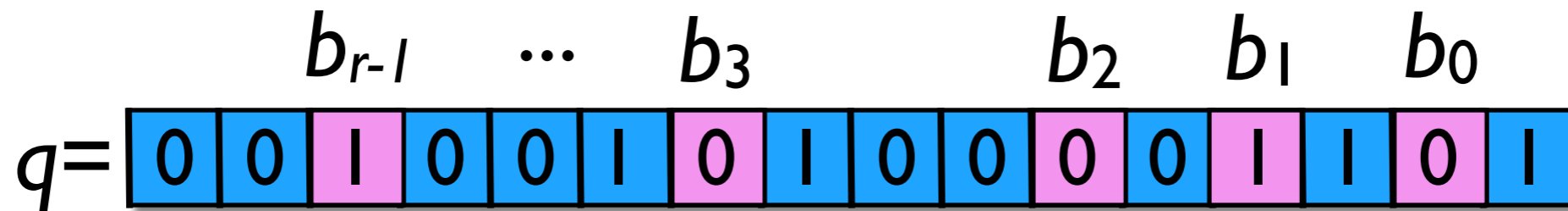
```
const char LogTable256[256] =
    {
        0,0,1,1,2,2,2,2,3,3,3,3,3,3,3,3,
        4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,
        5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,
        5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,
        6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,
        6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,
        6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,
        6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,
        7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,
        7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,
        7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,
        7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,
        7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,
        7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,
        7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,
        7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7
    };
```
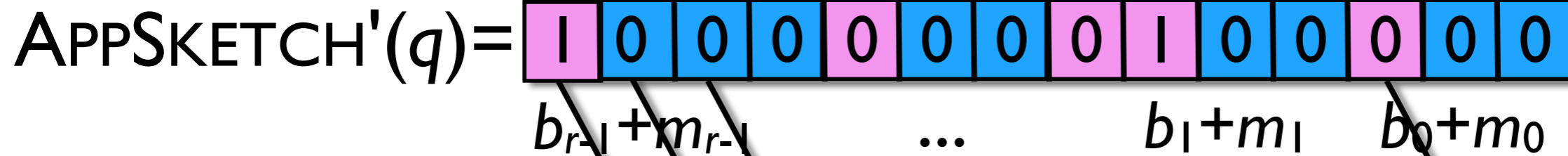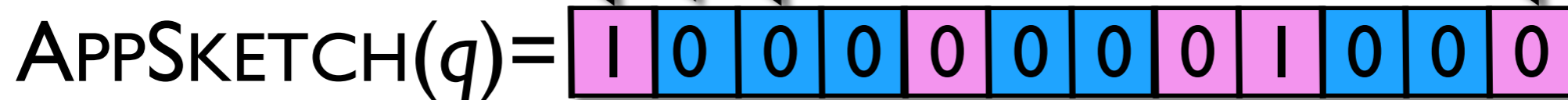
TODO 2:
Computing SKETCH(*q*)

b_{r-1} ... b_3 b_2 b_1 b_0

q= | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 |

difficult!

SKETCH(*q*)= | 1 | 0 | 0 | 1 | 0 |

APPSKETCH(*q*)= | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |

$O(w^{4/5})$

APPSKETCH(*q*)

APPSKETCH'(q)

Requirements on m_i's

- $\sum_{i,j} q_{b_i} 2^{b_i + m_j}$ should contain bits b_i of q at positions $b_i + m_j \Rightarrow$ **no overflow**

  a) $b_i + m_j = b_k + m_l \Leftrightarrow i{=}k$ and $j{=}l$

  b) $b_0 + m_0 < b_1 + m_1 < ... < b_{r-1} + m_{r-1}$

  c) $(b_{r-1} + m_{r-1}) - (b_0 + m_0) = O(w^{4/5})$

Existence of m_i's

- strong (a): distinctness **modulo r^3**:
  $b_i + m'_j \equiv b_k + m'_l \pmod{r^3} \Leftrightarrow i{=}k$ and $j{=}l$

  ▶ proof by induction:

    - for m'_j **avoid** $m'_l + (b_k{-}b_i) \pmod{r^3}$
    - for all $0 \leq l < j, 0 \leq k,i < r \Rightarrow jr^2 < r^3$
    - r^3 possible choices $\Rightarrow m'_j$ exists

- $\Rightarrow$ can make $b_i + m_i$ lie in different blocks of length r^3: $m_j = m'_j + (\lfloor (w{-}b_j{+}jr^3)/r^3 \rfloor \cdot r^3)$

Existence of m_i's

- Requirements satisfied:

  a) $b_i + m_j$ distinct modulo r^3 $\Rightarrow$ distinct overall

  b) $b_i + m_i$ lie in consecutive r-blocks
  $\Rightarrow b_0 + m_0 < b_1 + m_1 < ... < b_{r-1} + m_{r-1}$

  c) $b_0 + m_0 = w$
  $b_i + m_i \leq b_{i-1} + m_{i-1} + r^3$
  $r < w$
  $\Rightarrow (b_{r-1} + m_{r-1}) - (b_0 + m_0) \leq w + r^4 = O(w^{4/5})$

TODO 3: Finding q's Insertion Point

- B-tree node v with $x_0 < ... < x_{b-1}$: find i s.th. $\text{APPSKETCH}(x_i) \leq \text{APPSKETCH}(q) \leq \text{APPSKETCH}(x_{i+1})$

- $b = O(w^{1/5})$ sketches of size $O(w^{4/5}) \Rightarrow$

  b sketches **fused**=packed in **O(1) words**

- separate by 1-bits:

$$\text{APPSKETCH}(v) = \boxed{1 \;|\; \text{APPSKETCH}(x_0) \;|\; 1 \;|\; \text{APPSKETCH}(x_1) \;|\; 1 \;|\; \cdots \;|\; 1 \;|\; \text{APPSKETCH}(x_{b-1})}$$

Parallel Comparison

1 iff $x_i \geq q$

APPSKETCH(q) = $\boxed{0 \qquad\qquad \dots \qquad\qquad 0\ 0\ 0\ 0\ 0\ \text{APPSKETCH}(q)}$

C = $\boxed{0\ \dots\ 0\ |\ 0\ \dots\ 0\ |\quad \dots\quad |\ 0\ \dots\ 0\ |}$

APPSKETCH(v)= $\boxed{|\ \text{APPSKETCH}(x_0)\ |\ \text{APPSKETCH}(x_1)\ |\quad \dots\quad |\ \text{APPSKETCH}(x_{b-1})}$

APPSKETCH(q)*C= $\boxed{0\ \text{APPSKETCH}(q)\ 0\ \text{APPSKETCH}(q)\ 0\quad \dots\quad 0\ \text{APPSKETCH}(q)}$ = D

APPSKETCH(v)-D= $\boxed{{}^{0}_{1}\ ?\ \dots\ ?\ {}^{0}_{1}\ ?\ \dots\ ?\ {}^{0}_{1}\quad \dots\quad {}^{0}_{1}\ ?\ \dots\ ?}$ = Δ

E= $\boxed{|\ 0\ \dots\ 0\ |\ 0\ \dots\ 0\ |\quad \dots\quad |\ 0\ \dots\ 0}$

Δ AND E = $\boxed{{}^{0}_{1}\ 0\ \dots\ 0\ {}^{0}_{1}\ 0\ \dots\ 0\ {}^{0}_{1}\quad \dots\quad {}^{0}_{1}\ 0\ \dots\ 0}$

x_i's **ordered** $\Rightarrow {}^{0}_{1}$ pattern will be 00...011..1 $\Rightarrow$ find **MSB**

Summary

- multiple keys per B-tree node in $O(1)$ time

- **broadword computing**:

  ▶ subtraction: parallel comparison

  ▶ AND: masking

  ▶ multiplication:

    - create copies

    - bit permutation

- $O(\lg n / \lg w)$: **beat information theoretic barrier** of $O(n \lg n)$