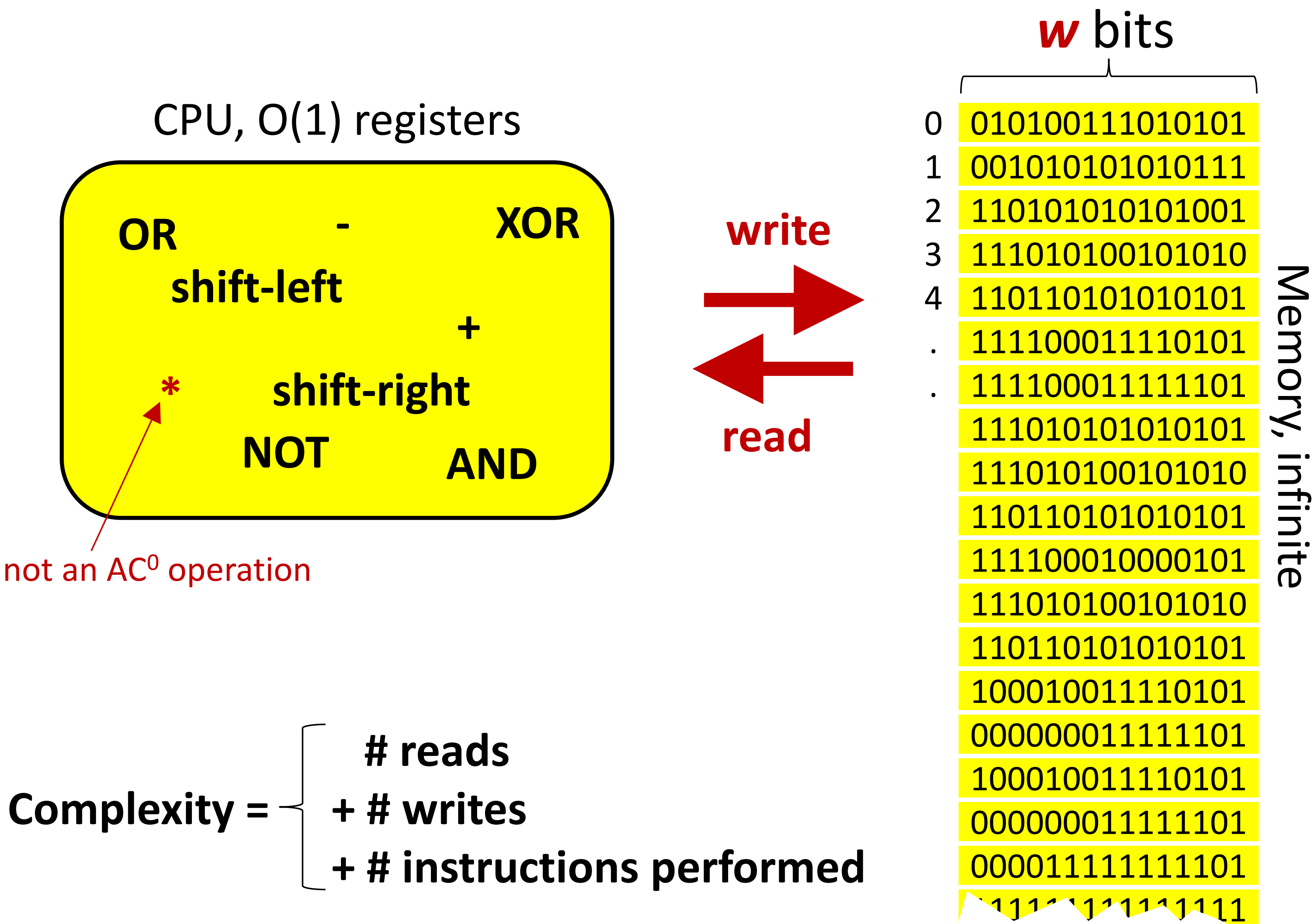
msb(x) in O(1) steps using 5 multiplications

[M.L. Fredman, D.E. Willard, *Surpassing the information-theoretic bound with fusion trees*, Journal of Computer and System Sciences 47 (3): 424–436, 1993]

$$
\begin{aligned}
& t_1 \leftarrow h \,\&\, (x \mid ((x \mid h) - l)), \quad \text{where } h = 2^{g-1} l \text{ and } l = (2^n - 1)/(2^g - 1); \\
& y \leftarrow (((a \cdot t_1) \bmod 2^n) \gg (n - g)) \cdot l, \quad \text{where } a = (2^{n-g} - 1)/(2^{g-1} - 1); \\
& t_2 \leftarrow h \,\&\, (y \mid ((y \mid h) - b)), \quad \text{where } b = (2^{n+g} - 1)/(2^{g+1} - 1); \\
& m \leftarrow (t_2 \ll 1) - (t_2 \gg (g - 1)),\ m \leftarrow m \oplus (m \gg g); \\
& z \leftarrow (((l \cdot (x \,\&\, m)) \bmod 2^n) \gg (n - g)) \cdot l; \\
& t_3 \leftarrow h \,\&\, (z \mid ((z \mid h) - b)); \\
& \lambda \leftarrow ((l \cdot ((t_2 \gg (2g - \lg g - 1)) + (t_3 \gg (2g - 1)))) \bmod 2^n) \gg (n - g).
\end{aligned}
$$

Word size $n = g \cdot g$, g a power of 2

RAM model (Random Access Machine)



Complexity = # reads + # writes + # instructions performed

Radix Sort

$w/\log n$ x COUNTING-SORT
$= O(n \cdot w/\log n)$

GOAL: Design algorithms with complexity independent of w (**trans-dichotomous**)

[M.L. Fredman, D.E. Willard, *Surpassing the information-theoretic bound with fusion trees*, Journal of Computer and System Sciences 47 (3): 424–436, 1993]

```
RADIX-SORT(A, d)
for i = 1 to d
        use a stable sort to sort array A on digit i

COUNTING-SORT(A, B, k)
let C[0..k] be a new array
for i = 0 to k
        C[i] = 0
for j = 1 to A.length
        C[A[j]] = C[A[j]] + 1
// C[i] now contains the number of elements equal to i.
for i = 1 to k
        C[i] = C[i] + C[i − 1]
// C[i] now contains the number of elements less than or equal to i.
for j = A.length downto 1
        B[C[A[j]]] = A[j]
        C[A[j]] = C[A[j]] − 1
```

[Cormen et al. 2009]

w bits

1	010100111010101
2	001010101010111
3	110101010101001
4	111010100101010
.	110110101010101
.	111100011110101
	111100011111101
	111010101010101
	111010100101010
	110110101010101
	111100010000101
	111010100101010
	110110101010101
n	100010011110101
	000000000000000
	000000000000000
	000000000000000
	000000000000000
	000000000000000

Sorting

Comparison	$O(n \cdot \log n)$
Radix-Sort	$O(n \cdot w / \log n)$
[T96]	$O(n \cdot \log\log n)$
[HT02]	$O(n \cdot \sqrt{\log\log n})$ exp.
[AHNR95]	$O(n)$ exp., $w \geq \log^{2+\varepsilon} n$
[BBN13]	$O(n)$ exp., $w \geq \log^2 n \cdot \log\log n$

[M. Thorup, *On RAM Priority Queues*. ACM-SIAM Symposium on Discrete Algorithms, 59-67, 1996]
[Y. Han, M. Thorup, *Integer Sorting in* $O(n\sqrt{\log\log n})$ *Expected Time and Linear Space*, IEEE Foundations of Computer Science, 135-144, 2002]
[A. Andersson, T. Hagerup, S. Nilsson, R. Raman: *Sorting in linear time?* ACM Symposium on Theory of Computing, 427-436, 1995]
[D. Belazzougu, G. S. Brodal, J. A. S. Nielsen, *Expected Linear Time Sorting for Word Size* $\Omega(\log^2 n \cdot \log\log n)$, manuscript 2013]

Priority queues (Insert/DeleteMin)

Comparison	$O(\log n)$
[T96]	$O(\log\log n)$
[HT02,T07]	$O(\sqrt{\log\log n})$ exp.

[M. Thorup, *On RAM Priority Queues*. ACM-SIAM Symposium on Discrete Algorithms, 59-67, 1996]
[Y. Han, M. Thorup, *Integer Sorting in* $O(n\sqrt{\log\log n})$ *Expected Time and Linear Space*, IEEE Foundations of Computer Science, 135-144, 2002]
[Mikkel Thorup, *Equivalence between priority queues and sorting*, J. ACM 54(6), 2007]

Dynamic predecessor searching (*w* dependent)

[vKZ77]	$O(\log w)$
[BF02]	$O(\log w/\log\log w)$ (static, space $n^{O(1)}$)
	$O(\log w/\log\log w \cdot \log\log n)$ (dynamic)

[P. van Emde Boas, R. Kaas, and E. Zijlstra, *Design and Implementation of an Efficient Priority Queue*, Mathematical Systems Theory 10, 99-127, 1977]
[P. Beame, F.E. Fich, *Optimal Bounds for the Predecessor Problem and Related Problems*. J. Comput. Syst. Sci. 65(1): 38-72, 2002]
[M. Patrascu, M. Thorup, *Time-space trade-offs for predecessor search*, ACM Symposium on Theory of Computing, 232-240, 2006]

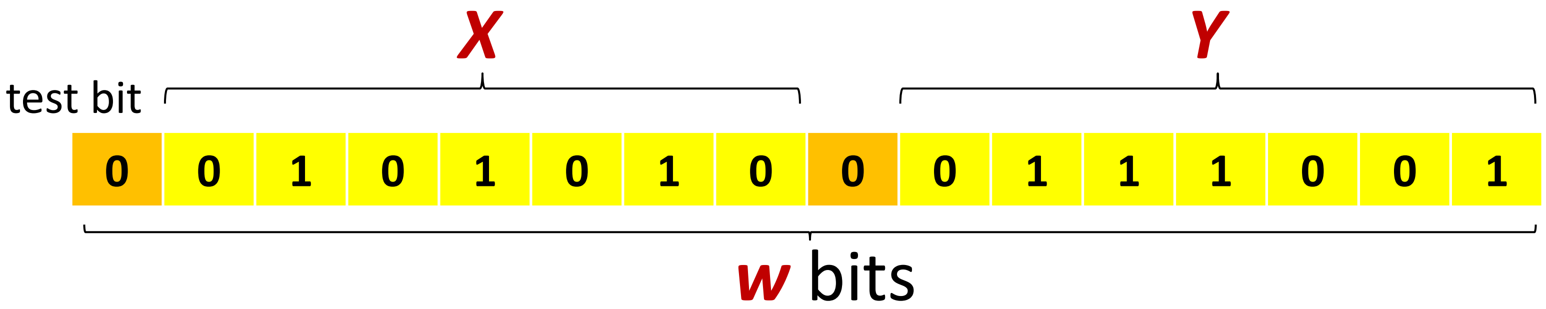
Dynamic predecessor searching (*w* independent)

Comparison	$O(\log n)$
[FW93]	$O(\log n/\log\log n)$
[AT07]	$O(\sqrt{\log n/\log\log n})$

[M.L. Fredman, D.E. Willard, *Surpassing the information-theoretic bound with fusion trees*, Journal of Computer and System Sciences 47 (3): 424–436, 1993]
[A. Andersson, M. Thorup, *Dynamic ordered sets with exponential search trees*. J. ACM 54(3): 13, 2007]

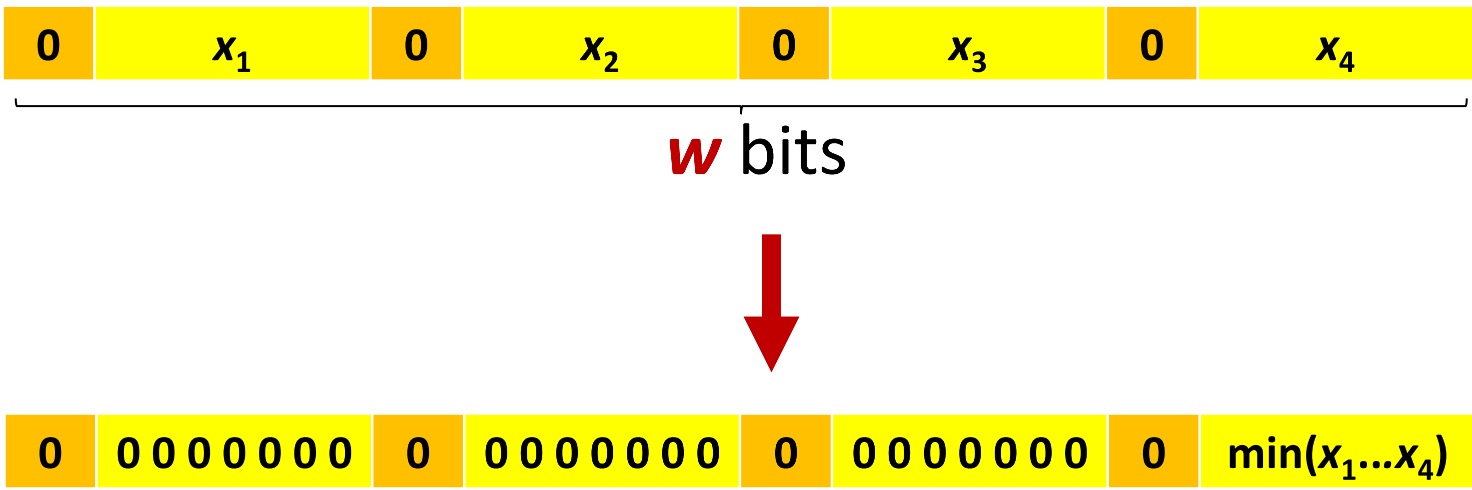
Sorting two elements in one word...

...without comparisons



Finding minimum of *k* elements in one word...

...without comparisons



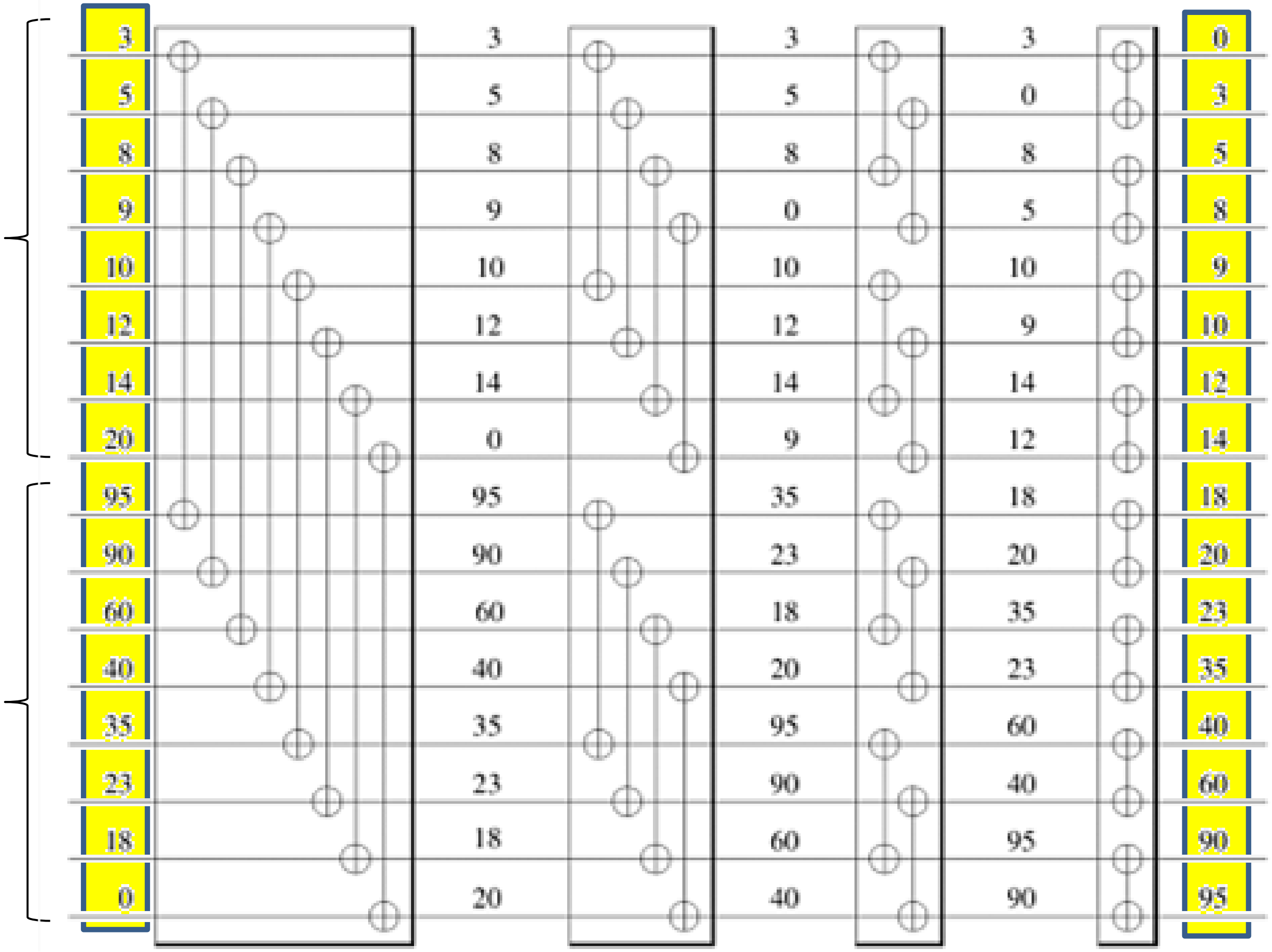
- Searching a sorted set...

Batcher’s bitonic merger

[K.E. Batcher, *Sorting Networks and Their Applications,* AFIPS Spring Joint Computing Conference 1968: 307-314]
[S. Albers, T. Hagerup, *Improved Parallel Integer Sorting without Concurrent Writing*, ACM-SIAM symposium on Discrete algorithms, 463-472, 1992] ← word implementation, O(log #elements) operations

increasing sequence

decreasing sequence

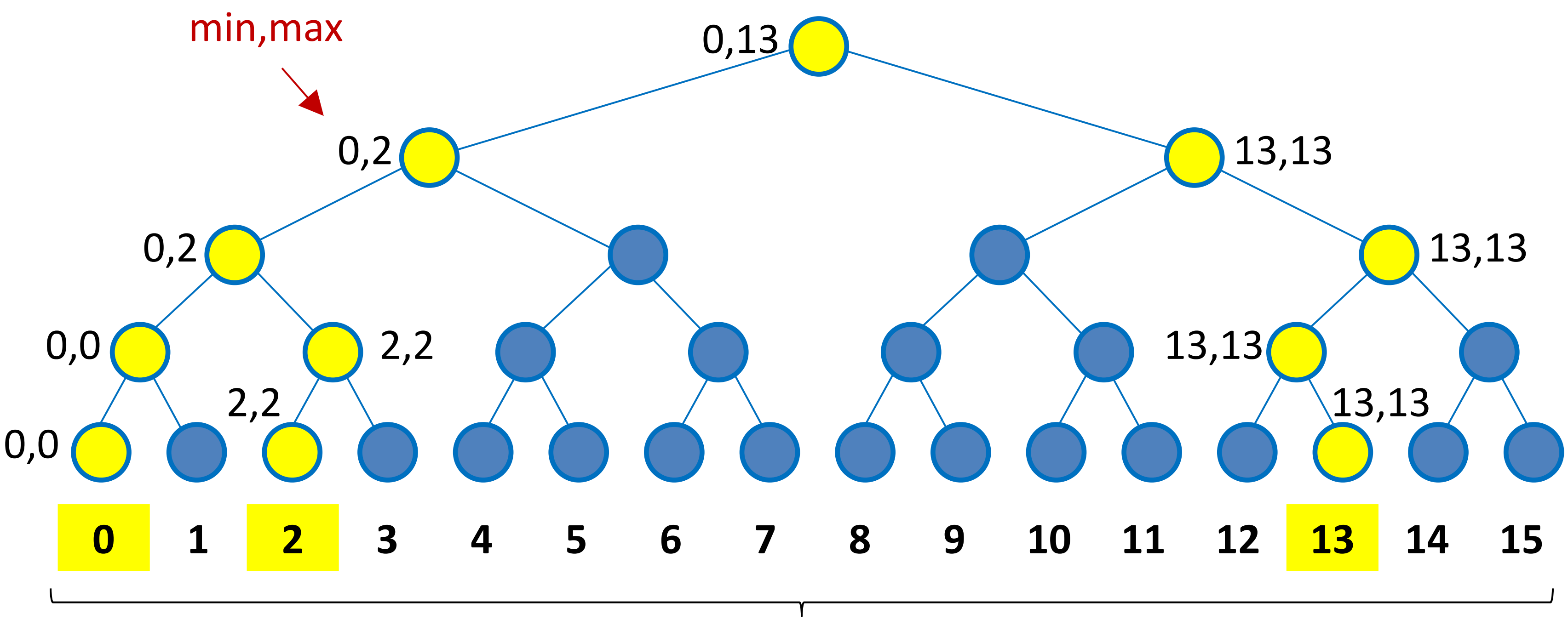


Round 1 Round 2 Round 3 Round 4

Remark: Sorting networks recently revived interest for GPU sorting

van Emde Boas (the idea in the static case)

[P. van Emde Boas, R. Kaas, and E. Zijlstra, *Design and Implementation of an Efficient Priority Queue*, Mathematical Systems Theory 10, 99-127, 1977]



Universe $U \leq 2^w$

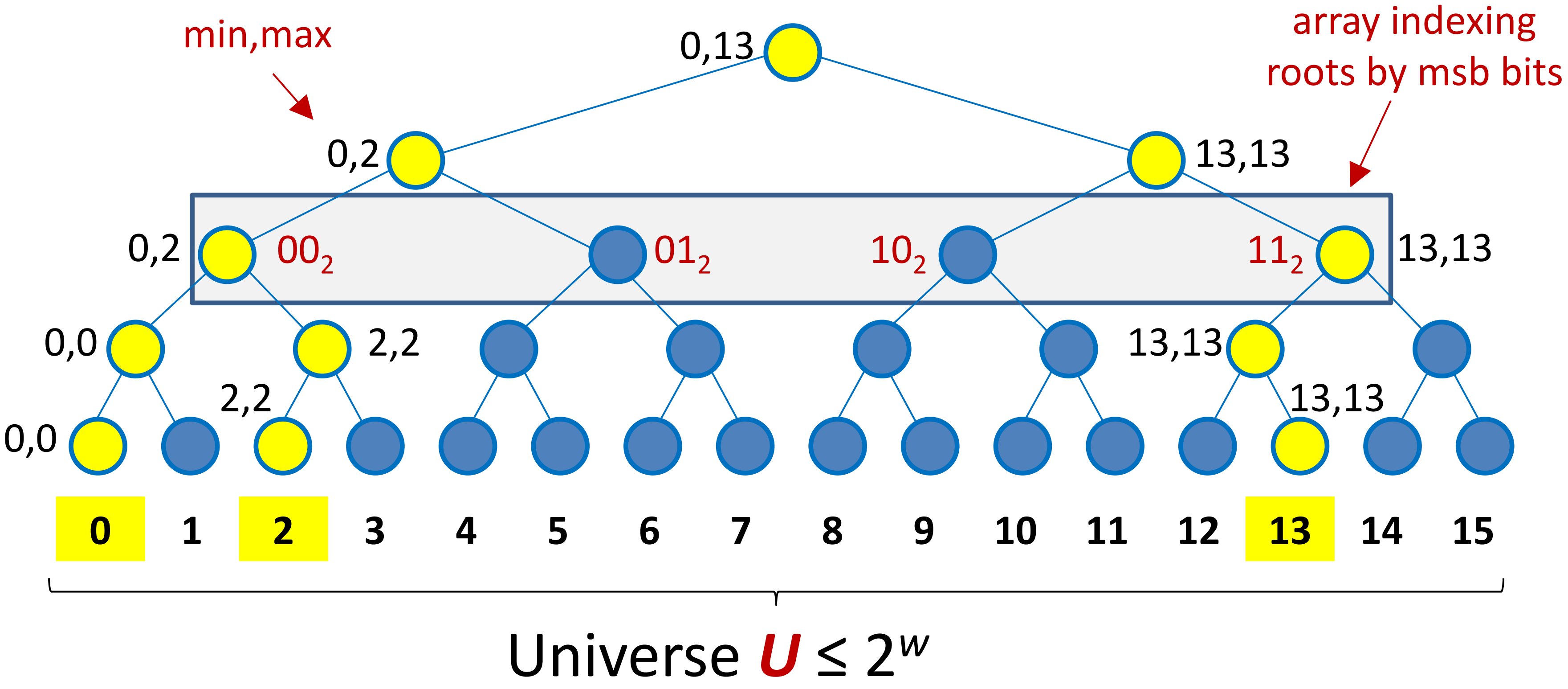
Predecessor search = find nearest yellow ancestor

= binary search on path O(loglog U)

Space O(U) ☹

van Emde Boas (addressing)

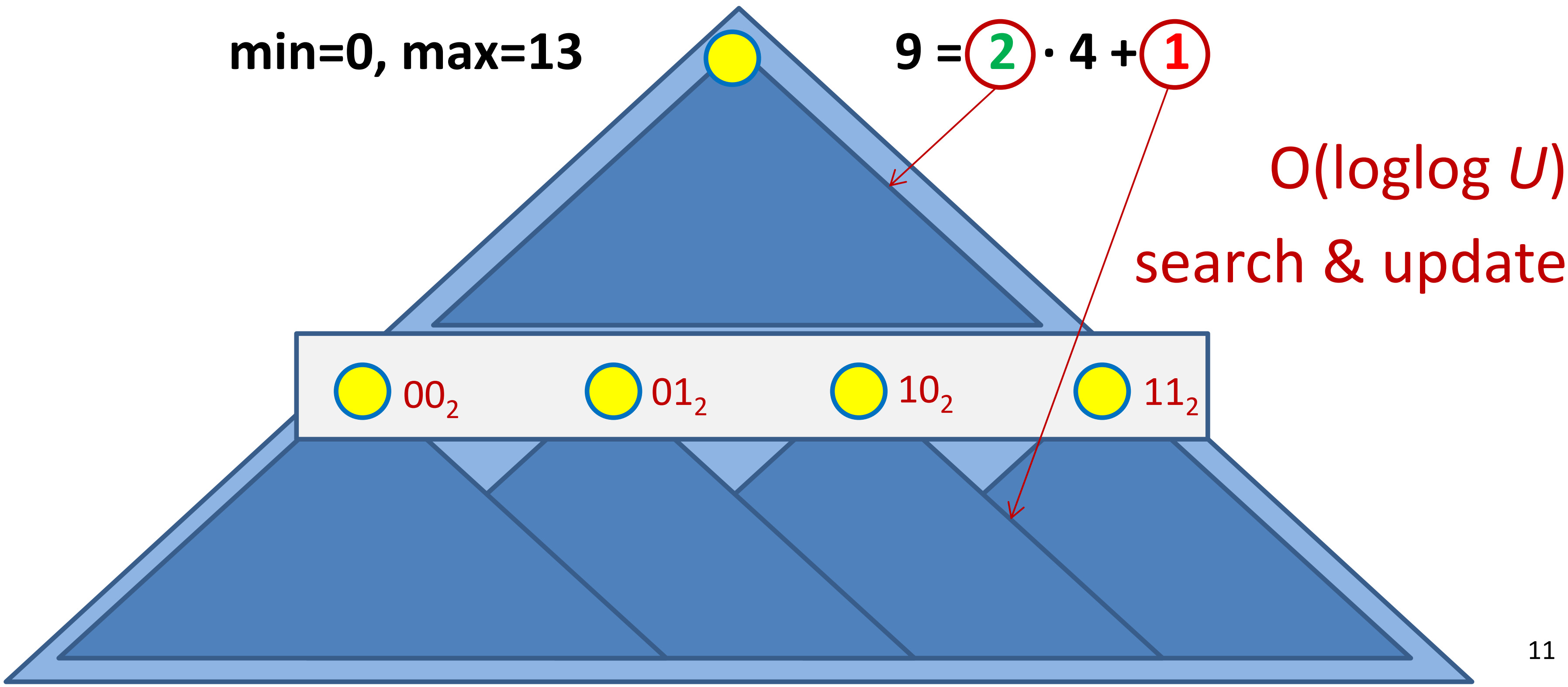
[P. van Emde Boas, R. Kaas, and E. Zijlstra, *Design and Implementation of an Efficient Priority Queue*, Mathematical Systems Theory 10, 99-127, 1977]



van Emde Boas (dynamic)

[P. van Emde Boas, R. Kaas, and E. Zijlstra, *Design and Implementation of an Efficient Priority Queue*, Mathematical Systems Theory 10, 99-127, 1977]

- 1 recursive top-structure and $\sqrt{U}$ bottom structures of the most and least significant log U/2 bits
- Keep min & max outside structure $\Rightarrow$ 1 recursive call



van Emde Boas (pseudo code)

[P. van Emde Boas, R. Kaas, and E. Zijlstra, *Design and Implementation of an Efficient Priority Queue*, Mathematical Systems Theory 10, 99-127, 1977]

```
succ(i)
  { i = a√n + b }
  if i > max then return +∞
  if i ≤ min then return min
  if size ≤ 2 then return max
  if bottom[a]. size > 0 and bottom[a].max ≥ b then
    return a√n + bottom[a].succ(b)
  else if top.max ≤ a then return max
  c := top.succ(a + 1)
  return c√n + bottom[c].min
```

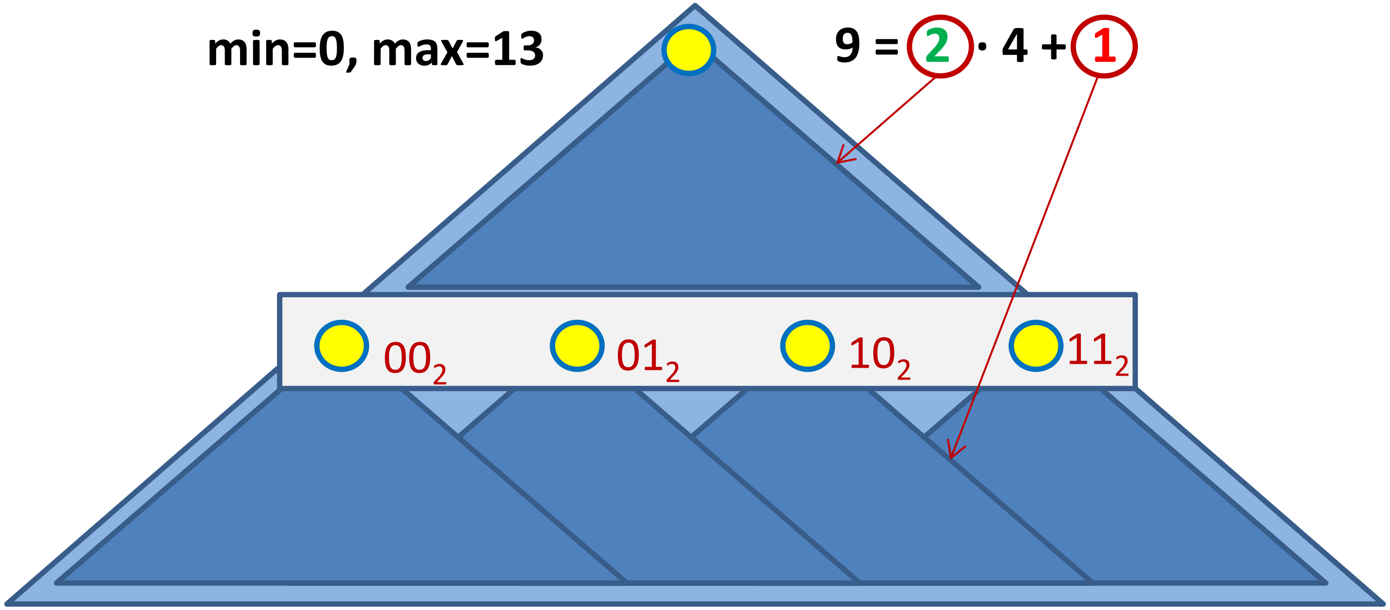
```
insert(i)
  if size = 0 then max := min := i
  if size = 1 then
    if i < min then min := i else max := i
  if size ≥ 2 then
    if i < min then swap(i, min)
    if i > max then swap(i, max)
    { i = a√n + b }
    if bottom[a].size = 0 then top.insert(a)
    bottom[a].insert(b)
  size := size + 1
```

```
delete(i)
  if size = 2 then
    if i = max then max := min else min := max
  if size > 2 then
    if i = min then i := min := top.min · √n + bottom[top.min].min
    else if i = max then i := max := top.max · √n + bottom[top.max].max
    { i = a√n + b }
    bottom[a].delete(b)
    if bottom[a].size = 0 then top.delete(a)
  size := size – 1
```

$O(\log\log U)$

van Emde Boas (linear space)

[P. van Emde Boas, R. Kaas, and E. Zijlstra, *Design and Implementation of an Efficient Priority Queue*, Mathematical Systems Theory 10, 99-127, 1977]
[Dan E. Willard, *Log-logarithmic worst-case range queries are possible in space* Θ(*N*), Information Processing Letters 17(2): 81–84, 1983]



- Buckets = lists of size O(loglog *U*), store only bucket minimum in vEB
- (Dynamic perfect) Hashing to store all O(*n*) non-zero nodes of vEB

→ O(*n*) space, O(loglog *U*) search

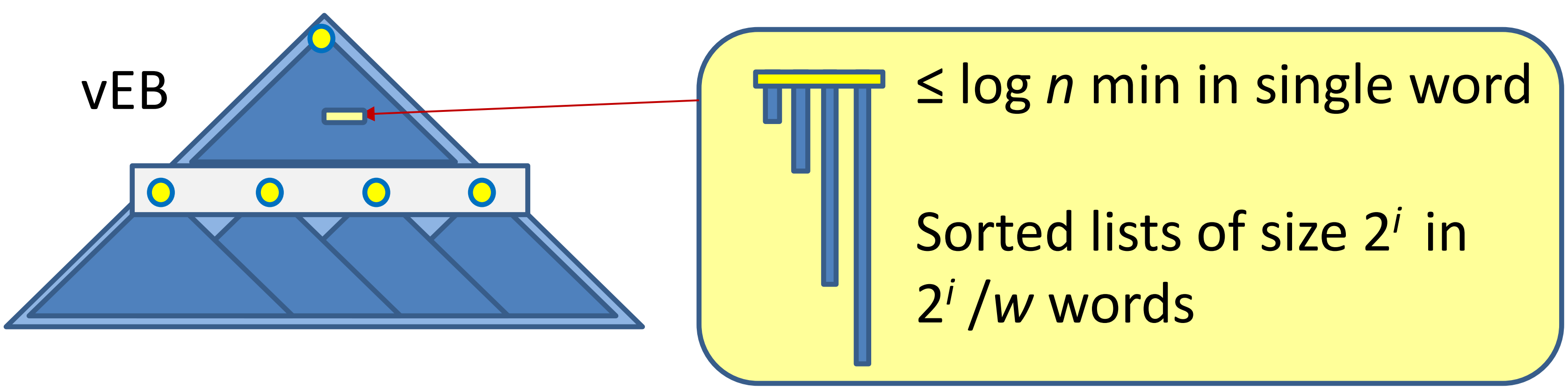
O(n·loglog n) Sorting

[M. Thorup, *On RAM Priority Queues*. ACM-SIAM Symposium on Discrete Algorithms, 59-67, 1996]

- loglog n recursive levels of vEB
  $\Rightarrow$ bottom of recursion **log *U* / log *n* bit** elements
- subproblems of ***k*** elements stored in k/log n words
  $\Rightarrow$ mergesort $O(\underbrace{k \cdot \log k}_{\text{merge-sort}} \cdot \underbrace{\log\log n}_{\text{merging 2 words}} / \underbrace{\log n}_{\text{\#elements per word}})$

O(loglog n) priority queue

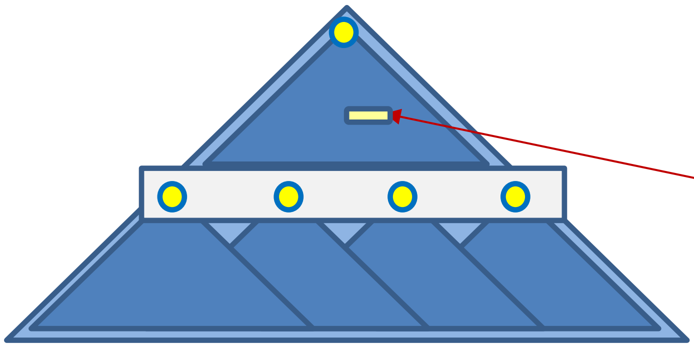
[M. Thorup, *On RAM Priority Queues*. ACM-SIAM Symposium on Discrete Algorithms, 59-67, 1996]



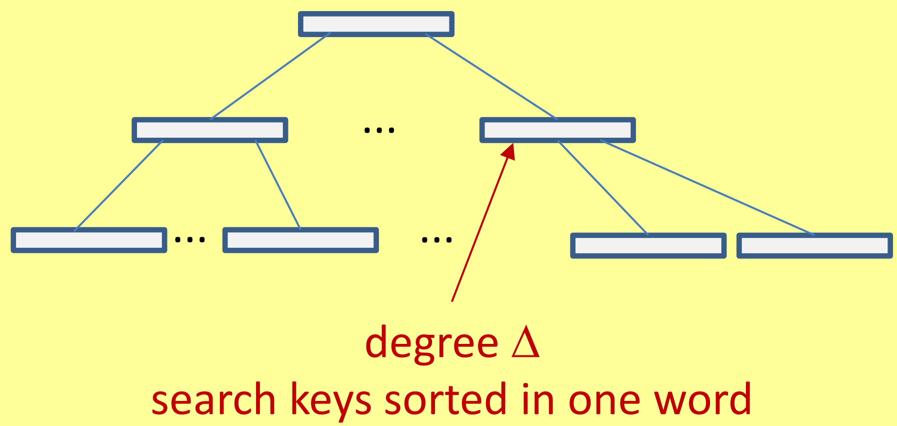
O($\sqrt{\log n}$) Dynamic predecessor searching

[Arne Andersson, *Sublogarithmic Searching Without Multiplications*. IEEE Foundations of Computer Science, 655-663, 1995]
[Arne Andersson, Mikkel Thorup, *Dynamic Ordered Sets with Exponential Search Trees,* J. ACM 54(3): 13, 2007]

vEB - $\sqrt{\log n}$ recursive levels



- $w / 2^{\sqrt{\log n}}$ bit elements
- packed B-tree of degree $\Delta = 2^{\sqrt{\log n}}$ and height $\log n / \log \Delta = \sqrt{\log n}$

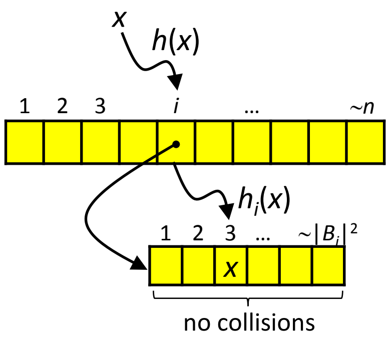


- O(1) time navigation at node

Sorting in O(*n*) time ?

Dynamic perfect hashing

[Michael L. Fredman, János Komlós, Endre Szemerédi, *Storing a Sparse Table with* O(1) *Worst Case Access Time*, J. ACM 31(3): 538-544, 1984]
[Martin Dietzfelbinger, Anna R. Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Heide, Hans Rohnert, Robert Endre Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, SIAM J. Computing 23(4): 738-761, 1994]



- Prime $p \geq U$
- $H = \{ h_k \mid 0<k<p \wedge h_k(x) = k \cdot x \bmod p \}$
- $\Pr[\, h(x)=h(y) \,] = 1/\text{table-size}$ – pr. $\Omega(1)$ no collision in bucket
- $E[\, \sum_i |B_i|^2 \,] = O(n^2/\text{table-size})$ – pr. $\Omega(1)$ total bucket space $O(n)$

- 2-level hashing of set S of size n
- Random hash functions from H: $h, h_1, h_2, \ldots$ (mod table size)
- Bucket $B_i = \{ x \in S \mid h(x) = i \}$
- Rehash:
  - whole table if $\sum_i |B_i|^2 \geq c \cdot n \rightarrow$ new table size n – all hash functions new
  - bucket if collision $\rightarrow$ new bucket size $|B_i|^2$ – one new h_i
- Search O(1) worst-case & updates O(1) expected amortized