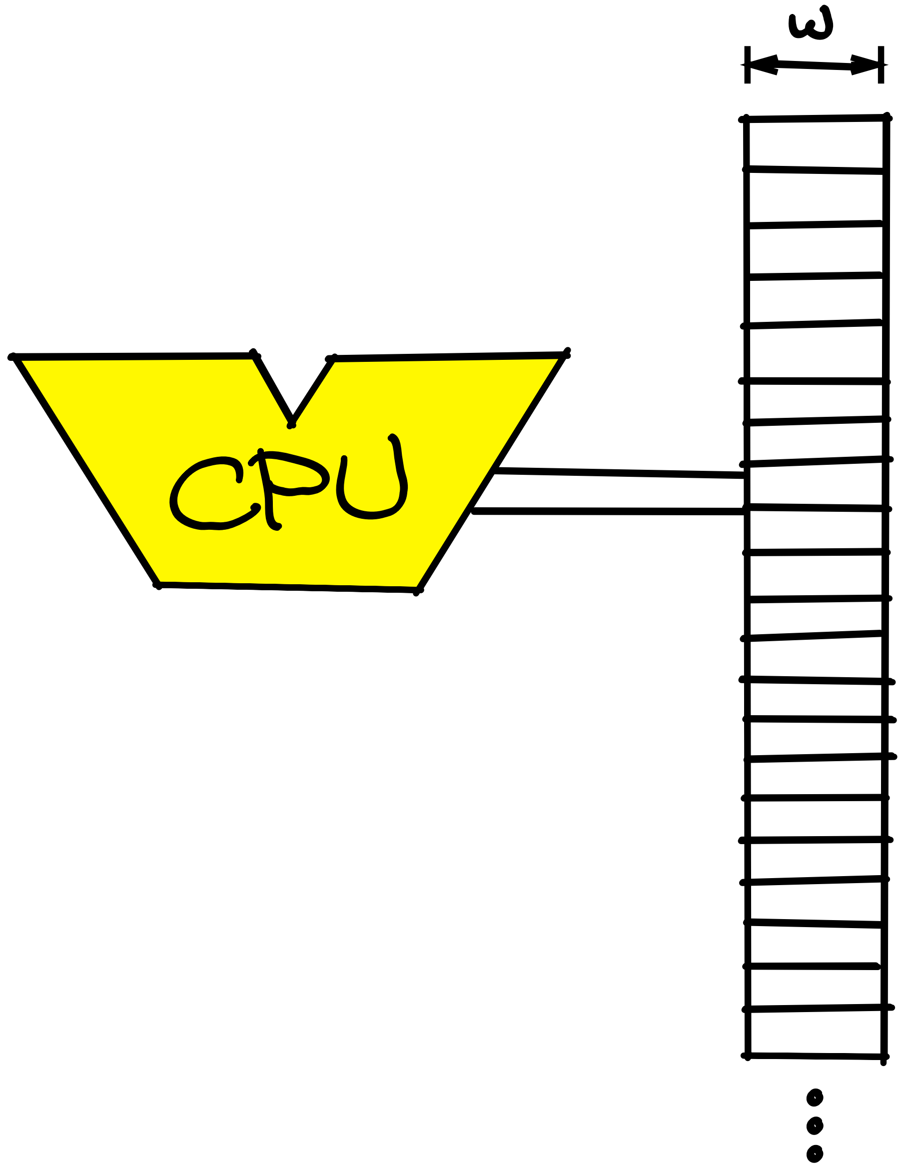
Data Structures on the Ultra-Wide Word RAM

Philip Bille
Inge Li Gørtz
Tord Stordalen

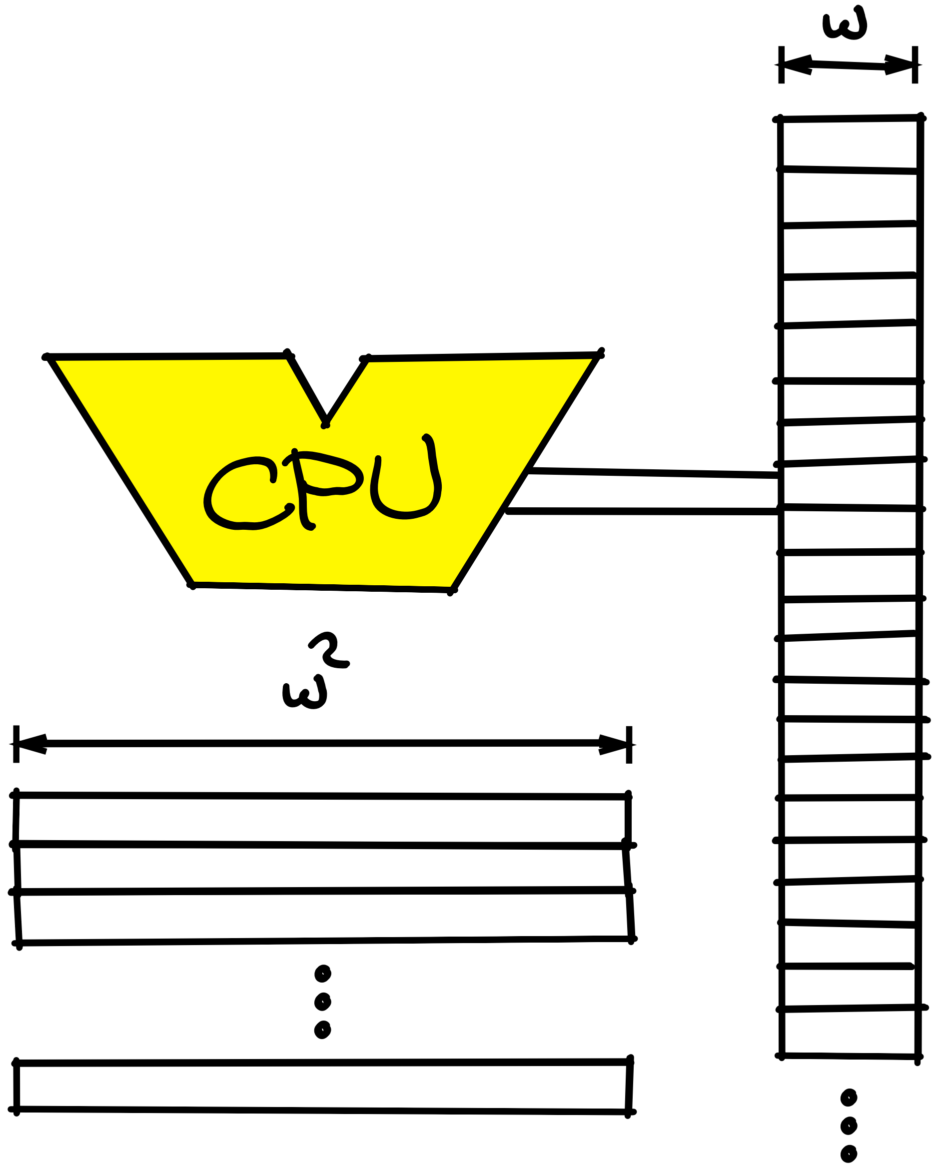
Outline

- UWRAM model.
- Predecessor on UWRAM.
  - Parallel hashing.
  - Dynamic dictionaries with parallel queries.
  - x-fast trie with parallel queries.

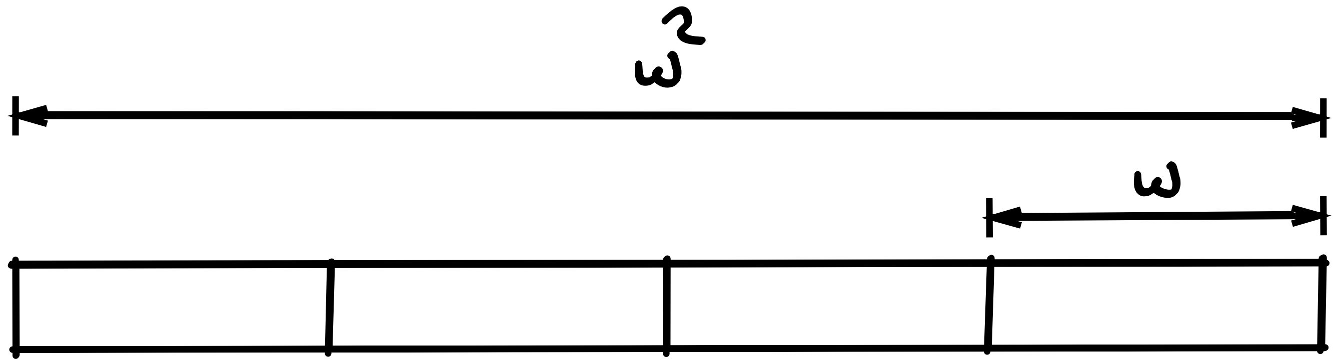
Word RAM



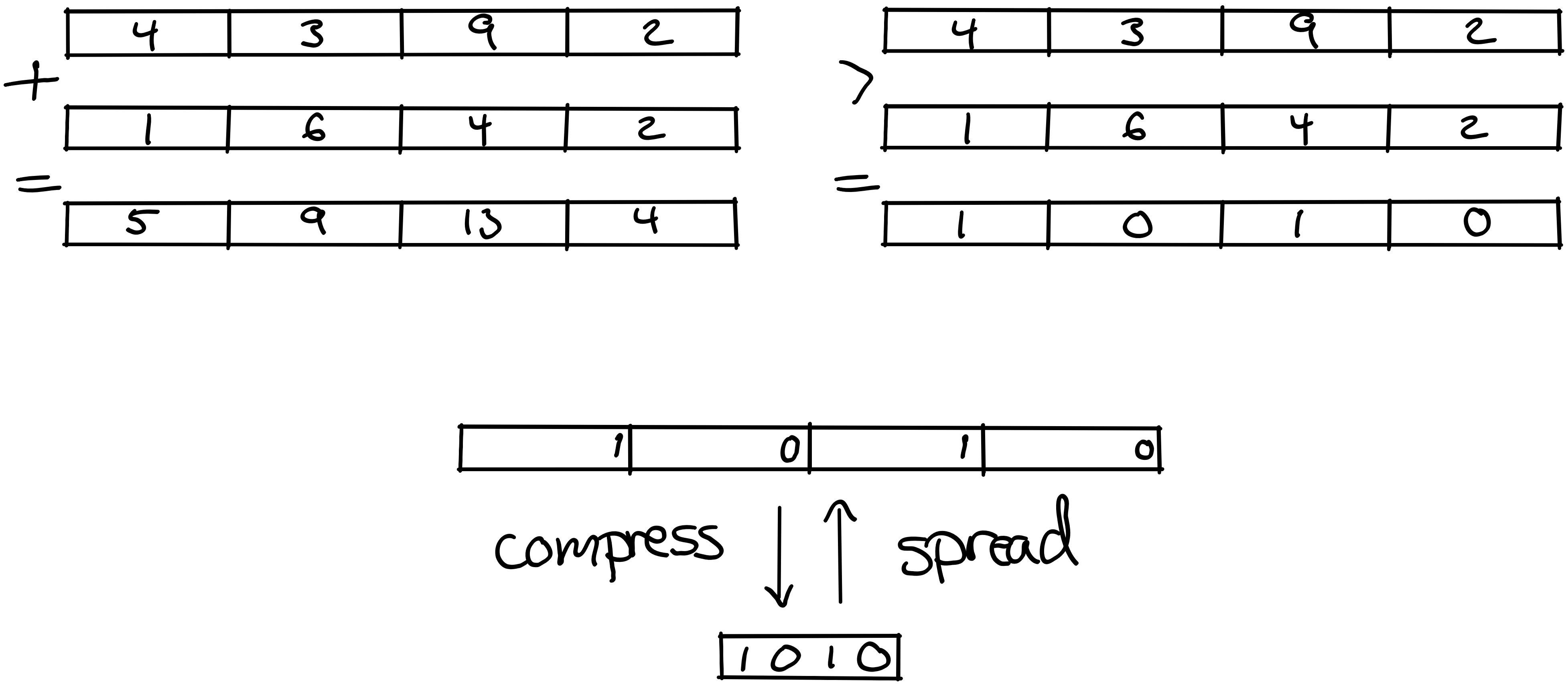
Ultra-Wide Word RAM [FLNS2015]



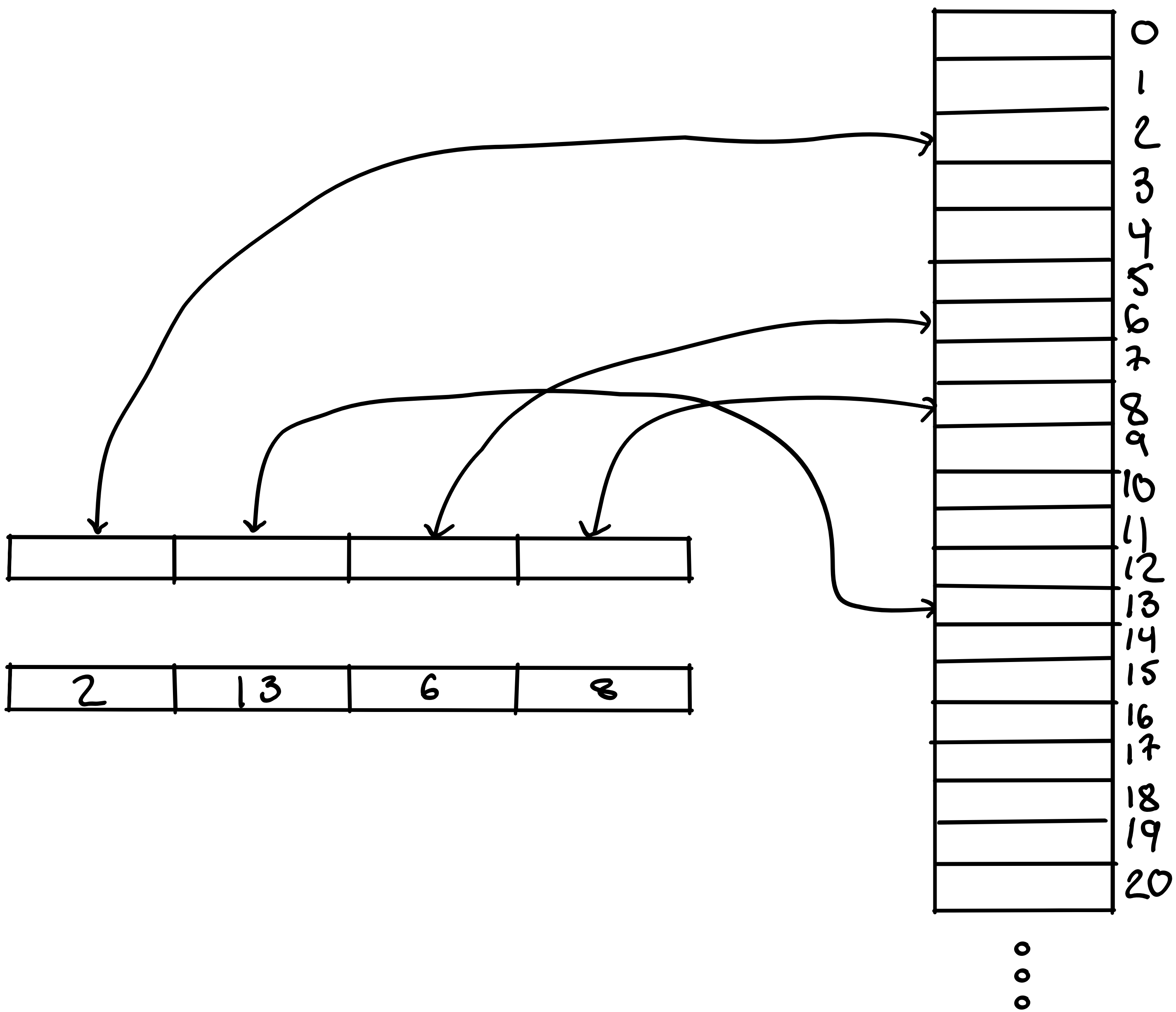
Ultra-Wide Word RAM



Ultra-Wide Word RAM



Ultra-Wide Word RAM



Predecessor

- Predecessor problem. Maintain a set S of n w-bit integers subject to the operations:
  - Predecessor(x): return largest integers in S that is ≤ x.
  - Insert(x): add x to S.
  - Delete(x): remove x from S.

Predecessor

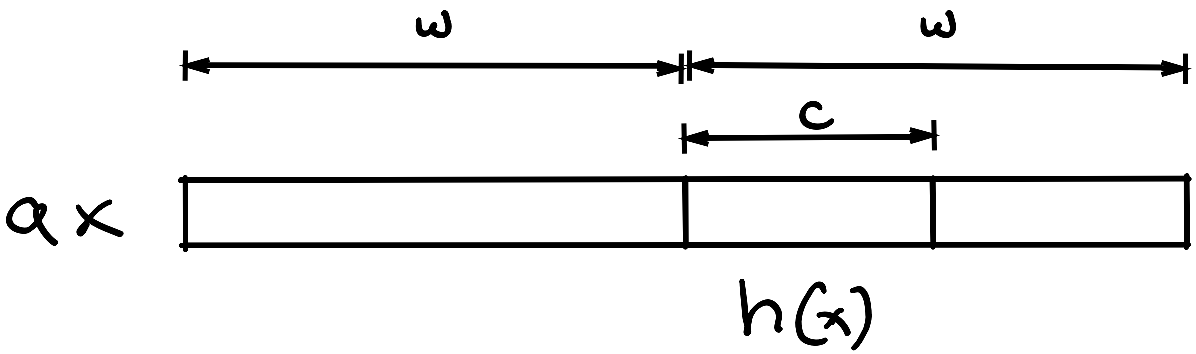
$$\Theta\left(\min\left\{\log_w n, \frac{\log\frac{w}{\log w}}{\log\left(\log\frac{w}{\log w}/\log\frac{\log n}{\log w}\right)}, \log\frac{\log(2^w - n)}{\log w}\right\}\right)$$

Predecessor

time	space	model	ref
$\omega(1)$	$O(n)$	WRAM	[vEB1975, MN1990, Willard1983, FW1990, PT2014, …]
$O(1)$	$O(2^w/w)$	RAMBO	[BKMN2006]
$O(1)$	$O(w2^w)$	UWRAM	[FLNS2015]
$O(1)$	$O(n)$	UWRAM	**new**

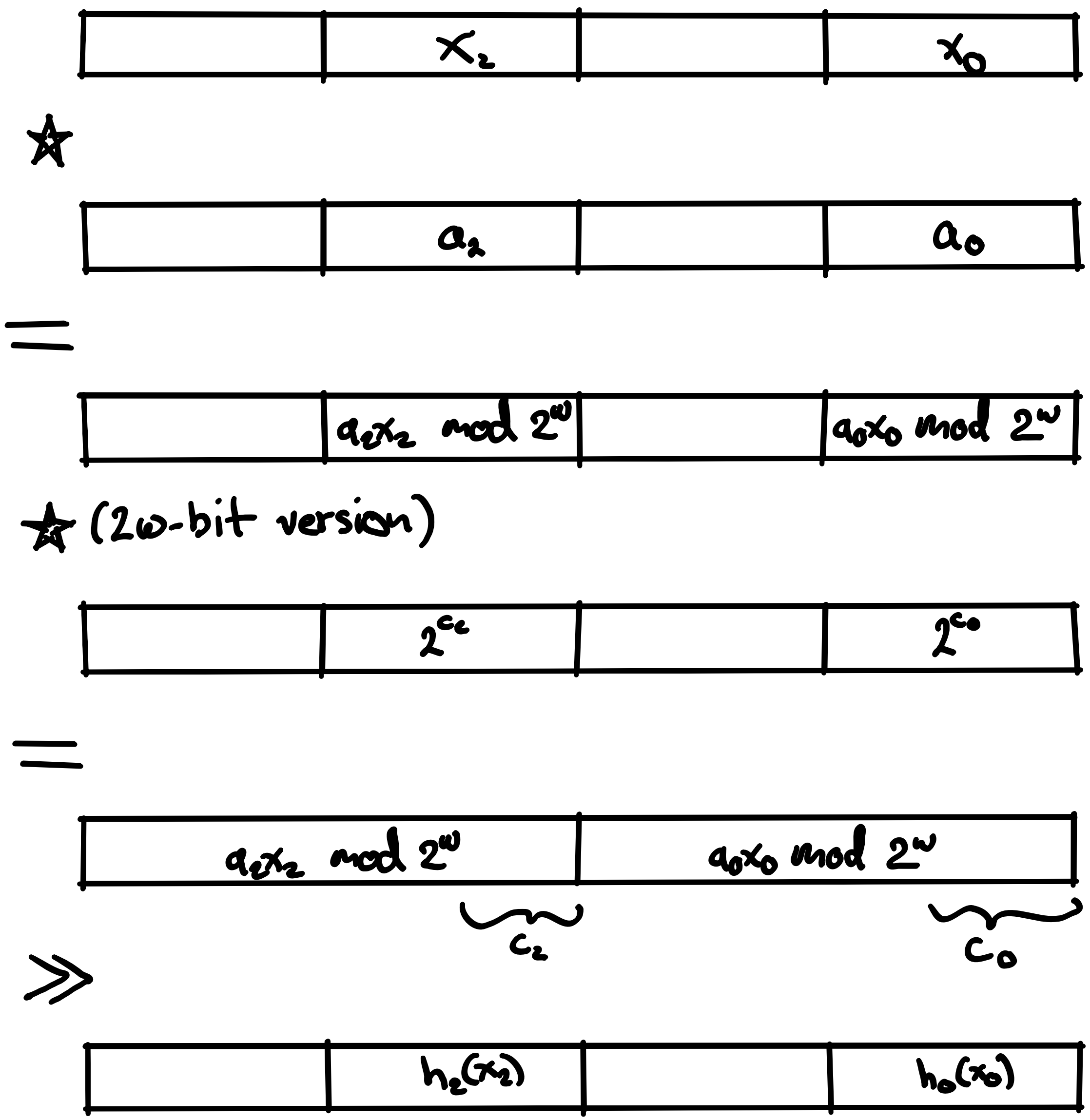
Multiply-Shift Hashing [DM1990]

$$h(x) = (ax \bmod 2^w) \gg (w - c)$$



Multiply Shift Hashing in Parallel

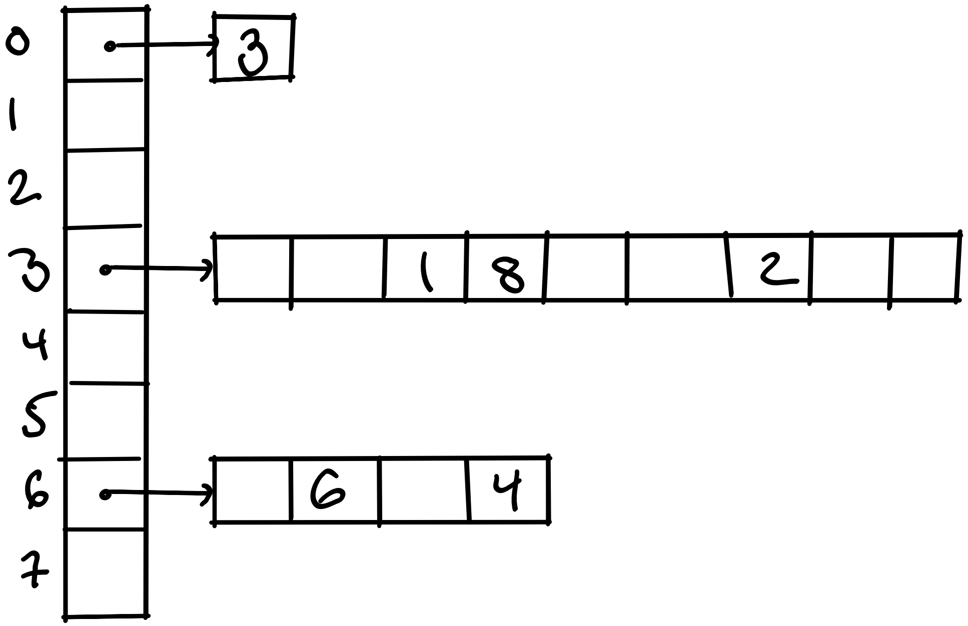
- Given $x_0, \ldots, x_{w-1}$ and $h_0, \ldots, h_{w-1}$, compute $h_0(x_0), \ldots, h_{w-1}(x_{w-1})$ in constant time.



Dictionaries with Parallel Queries

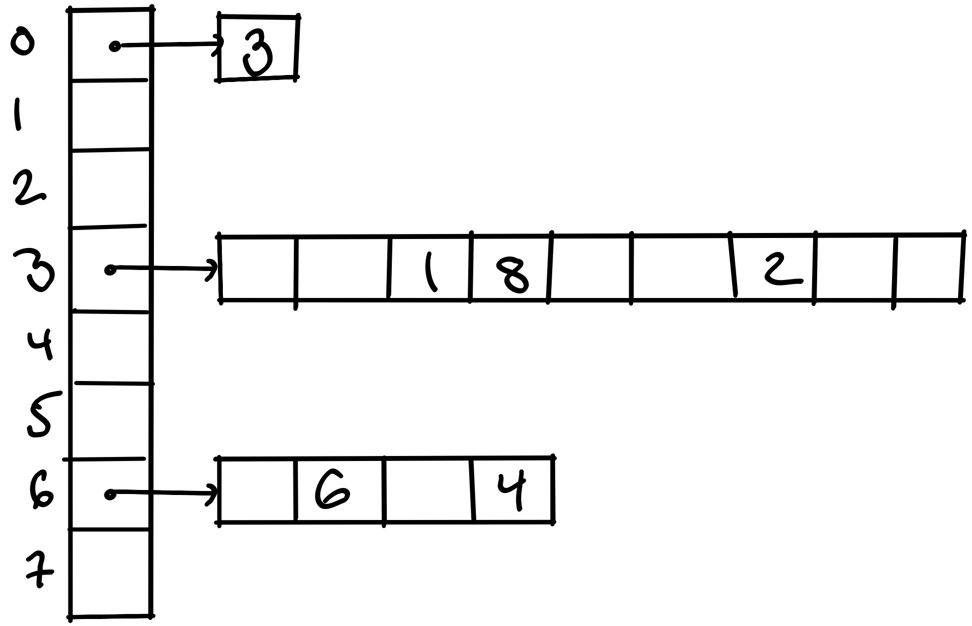
- w-parallel dictionary problem. Maintain a set S of n w-bit integers subject to the operations:
  - Member(x): determine if $x \in S$.
  - pMember($x_0, x_1, \ldots, x_{w-1}$): return $b_0, b_1, \ldots, b_{w-1}$ where $b_i = 1$ iff $x_i \in S$.
  - Insert(x): add x to S.
  - Delete(x): remove x from S.

Dynamic Perfect Hashing [DKMMRT1994]



- Two-level structure.
  - Lookup via universal hash function to find bucket i
  - Lookup in bucket i with universal hash function h_i.
  - Updates via global rebuilding strategy.
  - $\Rightarrow$ O(1) member, O(1) amortized expected insert and delete.

Dynamic Perfect Hashing with Parallel Queries



- Parallel queries.
  - Compute level 1 hash with parallel multiply-shift hashing.
  - Retrieve w buckets + hash function descriptions with scattered reads.
  - Compute level 2 hash with parallel multiply-shift hashing.
  - Retrieve data using scattered reads
  - Verify using component-wise comparison.
  - ⇒ O(1) pMember, O(1) member, O(1) amortized expected insert and delete.

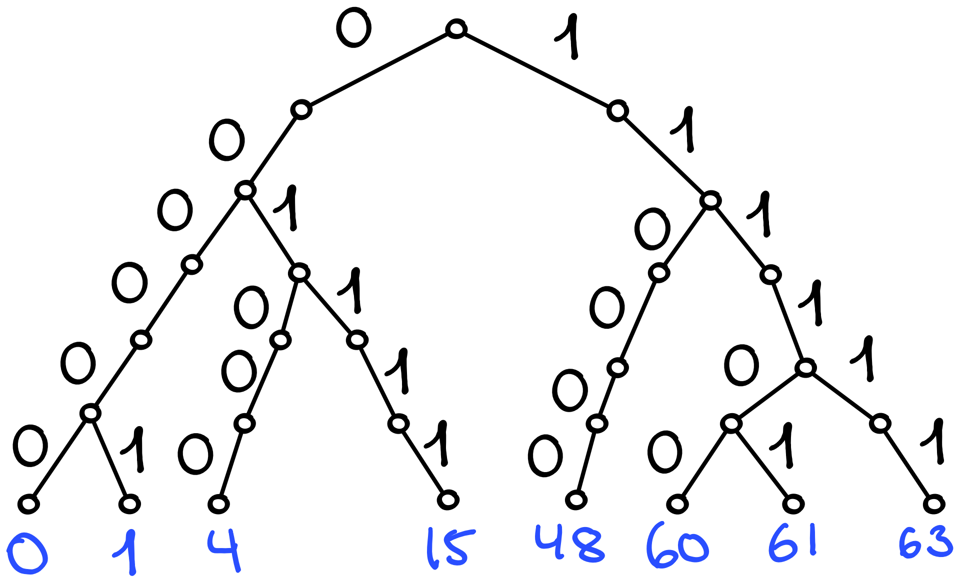
Dictionaries with Parallel Queries

- Dictionary with parallel queries.
  - O(1) time member.
  - O(1) time pMember.
  - O(1) amortized expected time insert and delete.
  - O(n + w) space.

- More tricks.
  - Holds on UWRAM with ultrawords of $w^{1+\varepsilon}$ bits.

x-fast Trie [Willard1983]

$D = \{0, 1, 00, 000, 001, \ldots\}$



- x-fast trie.
  - Trie on S.
  - Dictionary D on all prefixes of S in trie.

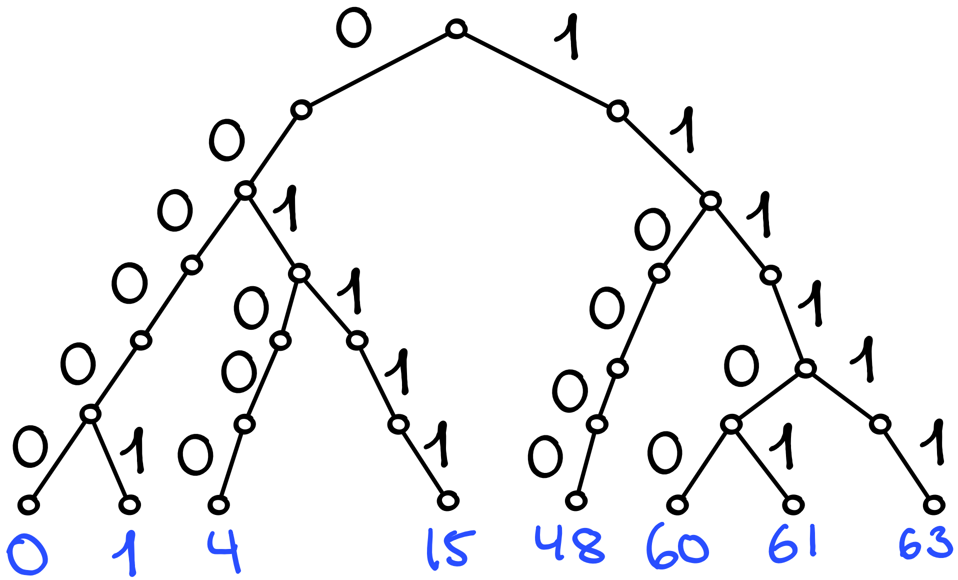
x-fast Trie [Willard1983]

x = 001100

001 ∈ D ✓

0011 ∈ D ✓

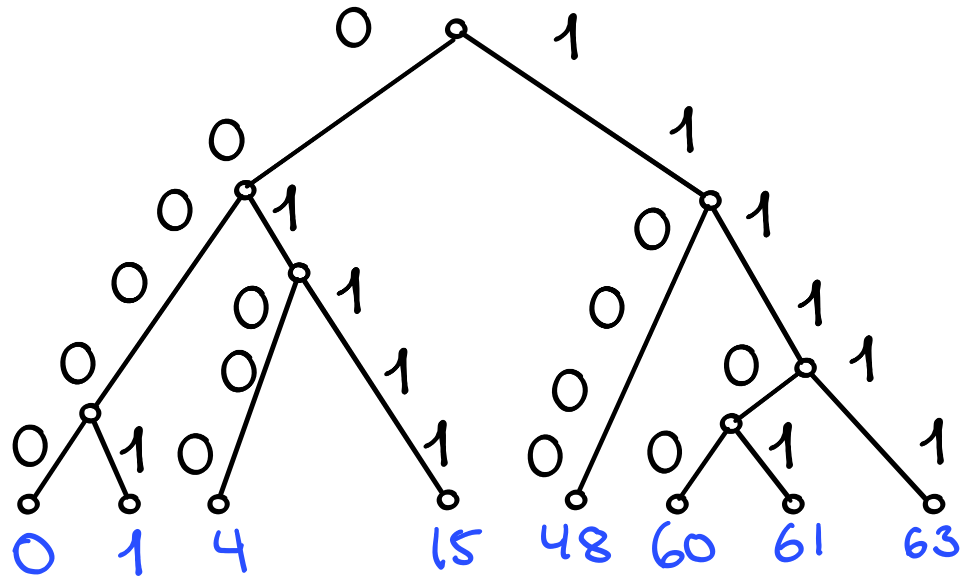
00110 ∈ D ✗



- Predecessor(x):
  - binary search on prefixes of x to find longest common prefix ⇒ identify predecessor.
- Insert and delete by updating dictionary.
- With dynamic perfect hashing ⇒ O(log w) predecessor, O(w) amortized expected insert and delete, O(nw) space.

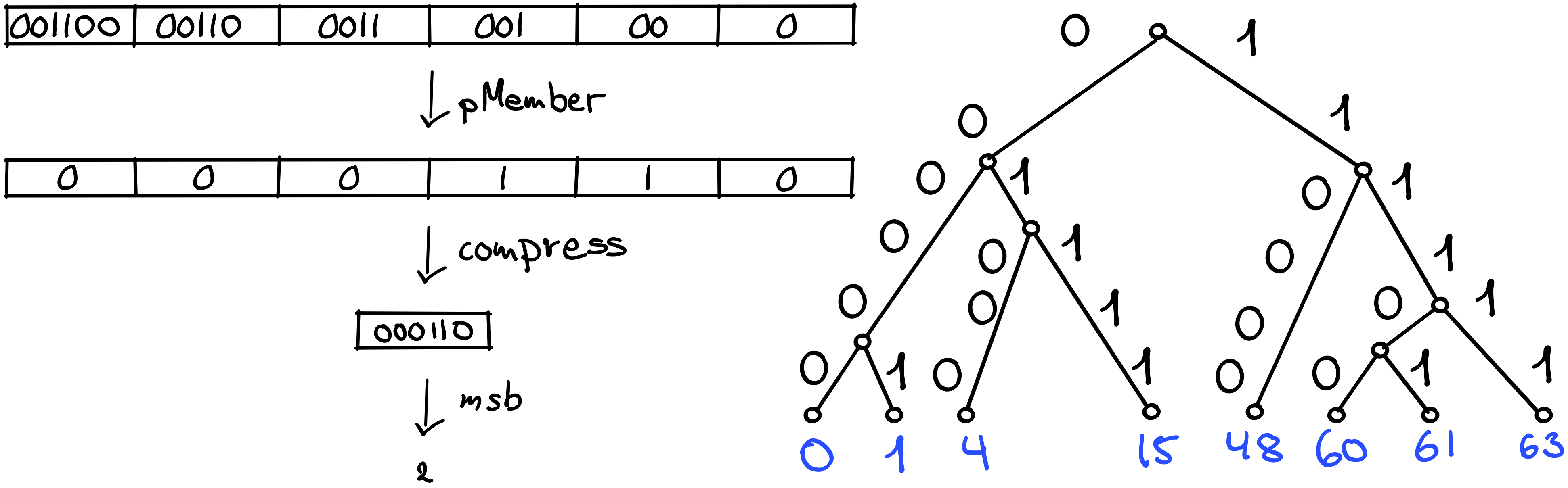
xtra-fast Trie

$D = \{00, 11, 00000, \dots\}$



- xtra-fast trie.
  - Compressed trie on S.
  - Dictionary D on prefixes of branching nodes in trie.

xtra-fast Trie



- Predecessor(x):
  - Generate all prefixes of x in ultraword with scattered read + masking.
  - Parallel member on dictionary.
  - Compress.
  - Find MSB ⇒ length of match ⇒ predecessor.
- Insert and delete change constant number of edges and nodes in trie
- ⇒ O(1) predecessor, O(1) amortized expected insert and delete, O(n + w) space

Predecessor

- Predecessor data structure.
  - O(1) time predecessor.
  - O(1) amortized expected time insert and delete.
  - O(n + w) space.

- More tricks.
  - Holds on UWRAM with ultrawords of $w^{1+\varepsilon}$ bits.
  - No + w term in space.