Fast Prefix Search in Little Space, with Applications

Djamal Belazzougui
Paolo Boldi
Rasmus Pagh
Sebastiano Vigna

ESA 2010

Talk overview

Talk overview

1. What?
2. Why?
3. What else?
4. How?
5. Then what?

2

1. What

.

1. What

- Standard (RAM) model, word size w.
- Static set S of n strings.
- *Prefix query*: Given a string p, what strings in S have p as a prefix?
  - Report all matching strings.

3

1. What

- Standard (RAM) model, word size w.
- Static set S of n strings.
- *Prefix query*: Given a string p, what strings in S have p as a prefix?
  - Report all matching strings.
  - Index: Assume strings stored sorted.

ranks of

3

1. What

- Standard (RAM) model, word size w.
- Static set S of n strings, w bits each.
- *Prefix query*: Given a string p, what strings in S have p as a prefix?
  - Report all matching strings. *ranks of*
  - Index: Assume strings stored sorted.

3

2. Why?

4

2. Why?

ALGO Liverp*

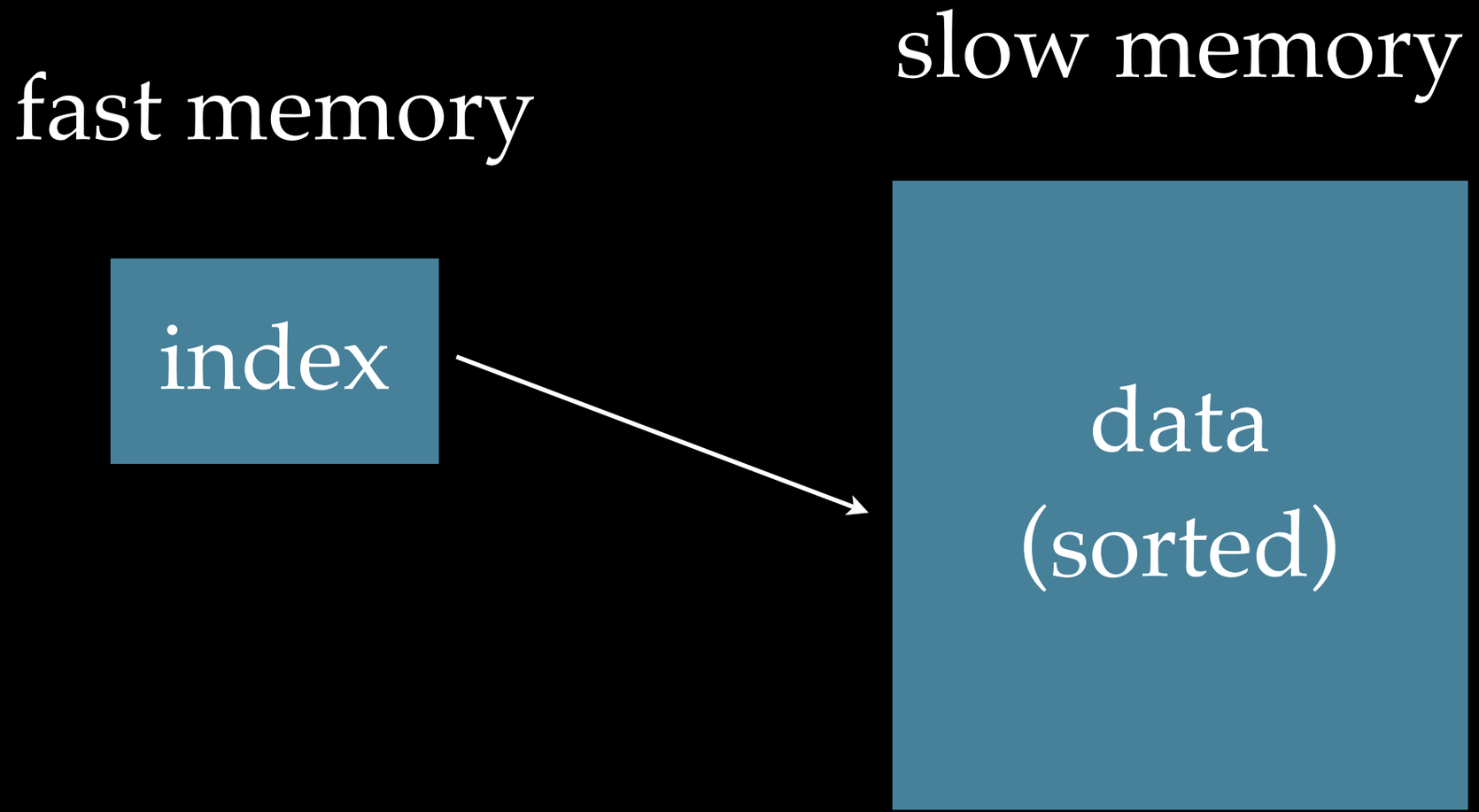
Google Search | I'm Feeling Lucky

4

2. Why?

- OLAP in a nutshell:
  - Dimensions D = Set<rooted tree>.
  - FactTable F = List<node from each D, number>.
  - *Query*: Given subtrees of D, sum up the numbers in F where all nodes are contained in the subtrees.

2. Why?



6

3. What else?

- Special case of *range* query
  - **return** $\mathtt{rank}_S([a;b])$
- Generalizes *point* query
  - **return** $\mathtt{rank}_S(\{x\})$
- No easier than *existence* queries
  - **return** $S \cap [a;b] \neq \emptyset$

7

Results on query time

(space O(nw) bits)

range

8

Results on query time

(space O(nw) bits)

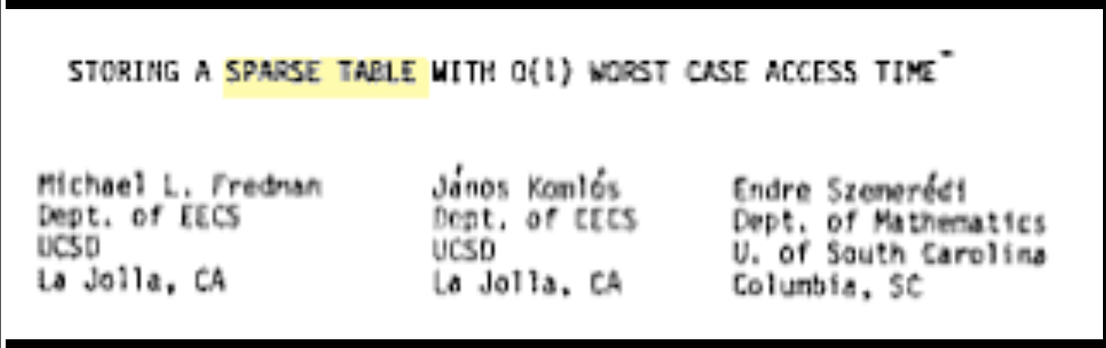
	existence	rank
point		
range		

8

Results on query time

(space O(nw) bits)

	existence	rank
point	O(1) [FKS, FOCS '82]	
range		



STORING A SPARSE TABLE WITH O(1) WORST CASE ACCESS TIME*

Michael L. Fredman
Dept. of EECS
UCSD
La Jolla, CA

János Komlós
Dept. of EECS
UCSD
La Jolla, CA

Endre Szemerédi
Dept. of Mathematics
U. of South Carolina
Columbia, SC

8

Results on query time

(space O(nw) bits)

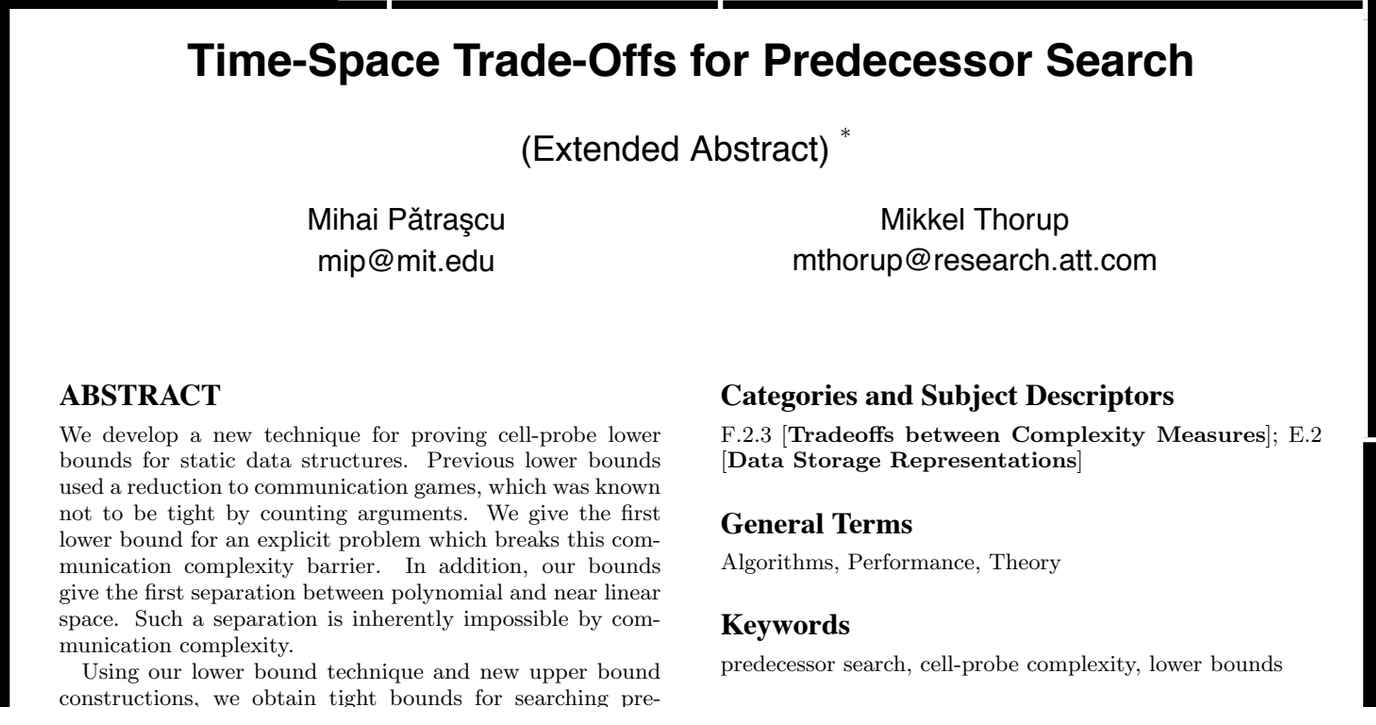
Design and Implementation of an Efficient Priority Queue*

P. VAN EMDE BOAS, R. KAAS, and E. ZIJLSTRA

Mathematical Centre, 2e Boerhaavestraat 49, Amsterdam,
and Mathematical Institute, University of Amsterdam,
Roeterstraat 15, Amsterdam, The Netherlands

ABSTRACT

We present a data structure, based upon a hierarchically decomposed tree, which enables us to manipulate on-line a priority queue whose priorities are selected from the interval 1, . . . , n with a worst case processing time of $\mathcal{O}$ (log log n) per instruction. The structure can be used to obtain a mergeable heap whose time requirements are about as good. Full details are explained based upon an implementation of the structure in a PASCAL program contained in the paper.



Time-Space Trade-Offs for Predecessor Search

(Extended Abstract) *

Mihai Pătraşcu
mip@mit.edu

Mikkel Thorup
mthorup@research.att.com

ABSTRACT

We develop a new technique for proving cell-probe lower bounds for static data structures. Previous lower bounds used a reduction to communication games, which was known not to be tight by counting arguments. We give the first lower bound for an explicit problem which breaks this communication complexity barrier. In addition, our bounds give the first separation between polynomial and near linear space. Such a separation is inherently impossible by communication complexity.

Using our lower bound technique and new upper bound constructions, we obtain tight bounds for searching pre-

Categories and Subject Descriptors

F.2.3 [**Tradeoffs between Complexity Measures**]; E.2 [**Data Storage Representations**]

General Terms

Algorithms, Performance, Theory

Keywords

predecessor search, cell-probe complexity, lower bounds

rank
O(log w) [vEB, FOCS '75]
Ω(log w) [PT, STOC '06]

8

Results on query time

(space O(nw) bits)

	existence	rank
point	O(1) [FKS, FOCS '82]	O(log w) [vEB, FOCS '75]
range	O(1) [ABR, STOC '01]	

Optimal Static Range Reporting in One Dimension

Stephen Alstrup* — The IT University of Copenhagen — stephen@it-c.dk

Gerth Stølting Brodal†‡ — BRICS, Dept. of Computer Science, University of Aarhus — gerth@brics.dk

Theis Rauhe* — The IT University of Copenhagen — theis@it-c.dk

ABSTRACT

We consider static one dimensional range searching problems. These problems are to build static data structures for an integer set $S \subseteq U$, where $U = \{0, 1, \ldots, 2^w - 1\}$, which support various queries for integer intervals of U. For the query of reporting all integers in S contained within a query interval, we present an optimal data structure with linear space cost and with query time linear in the number of integers reported. This result holds in the unit cost RAM model with word size w and a standard instruction set. We also present a linear space data structure for approximate range counting. A range counting query for an interval returns the number of integers in S contained within the interval. For any constant $\varepsilon > 0$, our range counting data structure returns in constant time an approximate answer which is

FindAny(a, b), $a, b \in U$: Report any element in $S \cap [a, b]$ or $\perp$ if there is no such element.

Report(a, b), $a, b \in U$: Report all elements in $S \cap [a, b]$.

Count$_\varepsilon(a, b)$, $a, b \in U, \varepsilon \geq 0$: Return an integer k such that $|S \cap [a, b]| \leq k \leq (1 + \varepsilon)|S \cap [a, b]|$.

We let n denote the size of S and let $u = 2^w$ denote the size of universe U. Our main result is a static data structure with space cost $O(n)$ that supports the query FindAny in constant time. As a corollary, the data structure allows Report in time $O(k)$, where k is the number of elements to be reported.

Furthermore, we give linear space structures for the approximate range counting problem. We present a data structure that uses space $O(n)$ and supports Count$_\varepsilon$ in constant

Results on query time

(space O(nw) bits)

	existence	rank
point	O(1) [FKS, FOCS '82]	O(log w) [vEB, FOCS '75]
range	O(1) [ABR, STOC '01]	Ω(log w) [PT, STOC '06]

8

Weak queries

- Guarantee output only on *some* inputs
  - Rank of prefixes of strings in S, *in O(1) time* [ABR '01].

Optimal Static Range Reporting in One Dimension

Stephen Alstrup*
The IT University of Copenhagen
stephen@it-c.dk

Gerth Stølting Brodal†‡
BRICS
Dept. of Computer Science
University of Aarhus
gerth@brics.dk

Theis Rauhe*
The IT University of Copenhagen
theis@it-c.dk

ABSTRACT

We consider static one dimensional range searching problems. These problems are to build static data structures for an integer set $S \subseteq U$, where $U = \{0, 1, \ldots, 2^w - 1\}$, which support various queries for integer intervals of U. For the query of reporting all integers in S contained within a query interval, we present an optimal data structure with linear space cost and with query time linear in the number of integers reported. This result holds in the unit cost RAM model with word size w and a standard instruction set. We also present a linear space data structure for approximate range

FindAny(a, b), $a, b \in U$: Report any element in $S \cap [a, b]$ or $\perp$ if there is no such element.

Report(a, b), $a, b \in U$: Report all elements in $S \cap [a, b]$.

Count$_\varepsilon(a, b)$, $a, b \in U, \varepsilon \geq 0$: Return an integer k such that $|S \cap [a, b]| \leq k \leq (1 + \varepsilon)|S \cap [a, b]|$.

We let n denote the size of S and let $u = 2^w$ denote the size of universe U. Our main result is a static data structure with space cost $O(n)$ that supports the query FindAny in constant time. As a corollary, the data structure allows Report in time $O(k)$, where k is the number of elements to

Weak queries

- Guarantee output only on *some* inputs
  - Rank of prefixes of strings in S, *in O(1) time* [ABR '01].
  - Represent a function with domain S, *without storing S* [SS '89], [CKRT, '04].

The Bloomier Filter: An Efficient Data Structure for Static Support Lookup Tables *

Bernard Chazelle† Joe Kilian‡ Ronitt Rubinfeld‡ Ayellet Tal§

"Oh boy, here is another David Nelson"
Ticket Agent, Los Angeles Airport
(Source: BBC News)

Abstract

We introduce the *Bloomier filter*. a data structure for

the problem was due to name-matching technology used by airlines."

This story illustrates a common problem that arises when one tries to balance false negatives and false positives: if one is unwilling to accept any false negatives whatsoever, one often pays with a high false positive

Weak queries

- Guarantee output only on *some* inputs
  - Rank of prefixes of strings in S, *in O(1) time* [ABR '01].
  - Represent a function with domain S, *without storing S* [SS '89], [CKRT, '04].
  - Rank of any string in S, using O(n log log w) bits of space [BBPV '09].

Monotone Minimal Perfect Hashing:
Searching a Sorted Table with $O(1)$ Accesses

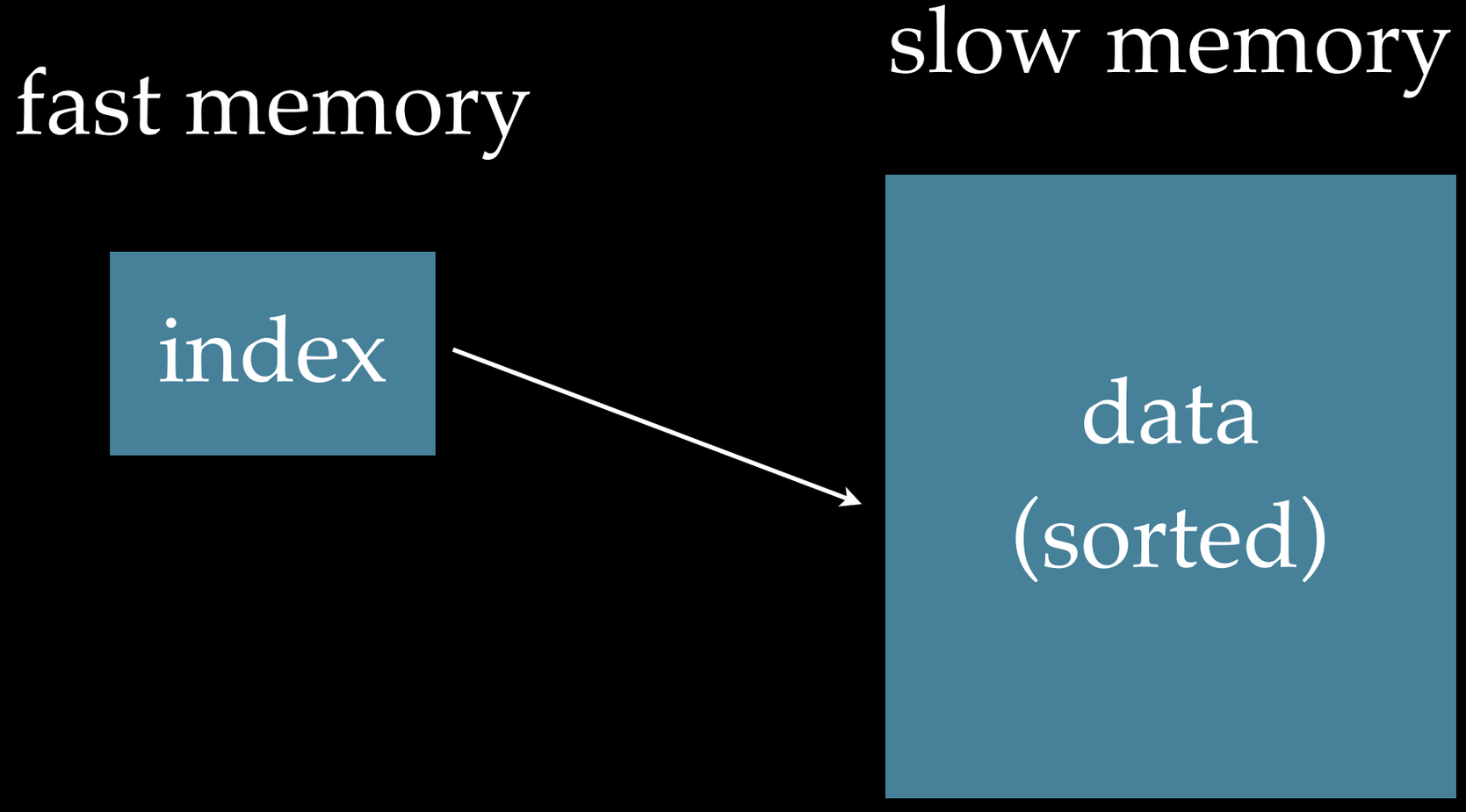
Djamal Belazzougui* Paolo Boldi† Rasmus Pagh‡ Sebastiano Vigna†

Weak queries

- Guarantee output only on *some* inputs
  - Rank of prefixes of strings in S, *in O(1) time* [ABR '01].
  - Represent a function with domain S, *without storing S* [SS '89], [CKRT, '04].
  - Rank of any string in S, using O(n log log w) bits of space [BBPV '09].

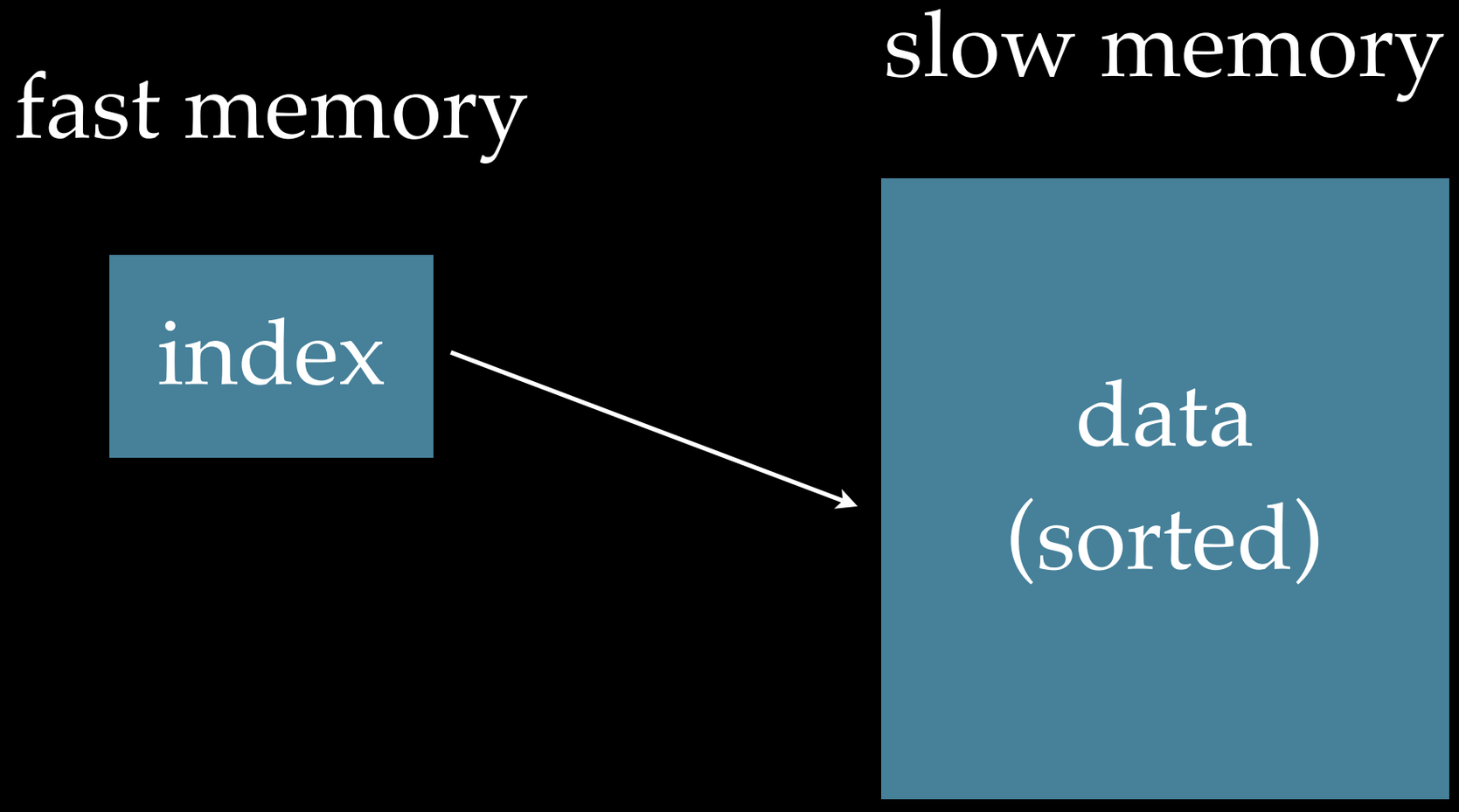
9

1. What (we show)



1. What (we show)

✤ Weak prefix search (on prefixes that exist):
Possible using space O(n log w) bits.



1. What (we show)

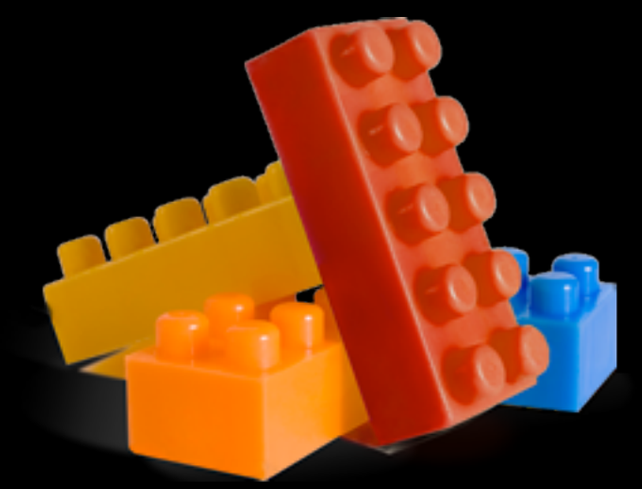
- Weak prefix search (on prefixes that exist): Possible using space O(n log w) bits.
- Ω(n log w) bits required (worst-case).
- Space/query time trade-off: Time O(t) with space $O(nw^{1/t} \log w)$.

1. What (we show)

- Weak prefix search (on prefixes that exist): Possible using space O(n log w) bits.
- Ω(n log w) bits required (worst-case).
- Space/query time trade-off: Time O(t) with space $O(nw^{1/t} \log w)$.
- (Paper generalizes to average length, cache-oblivious model, larger alphabets, "compression",...)

4. How?

- ✤ Building blocks:
  - Monotone minimal perfect hashing
  - Storing a function
  - Fat binary search

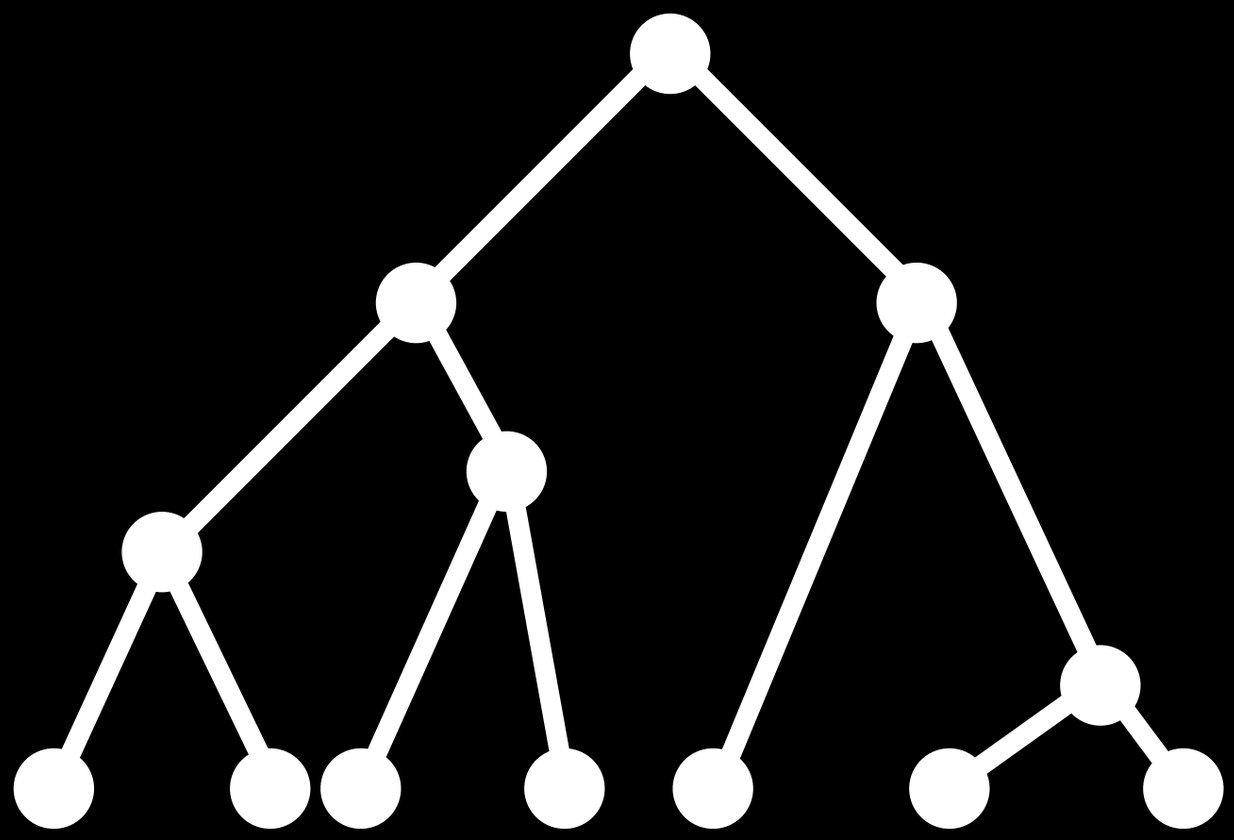


Monotone MPH

- Store a function f where for each $x \in S$ $f(x)=\texttt{rank}_S(x)$.
- O(n log w) bits, time O(1) [BBPV '09].

13

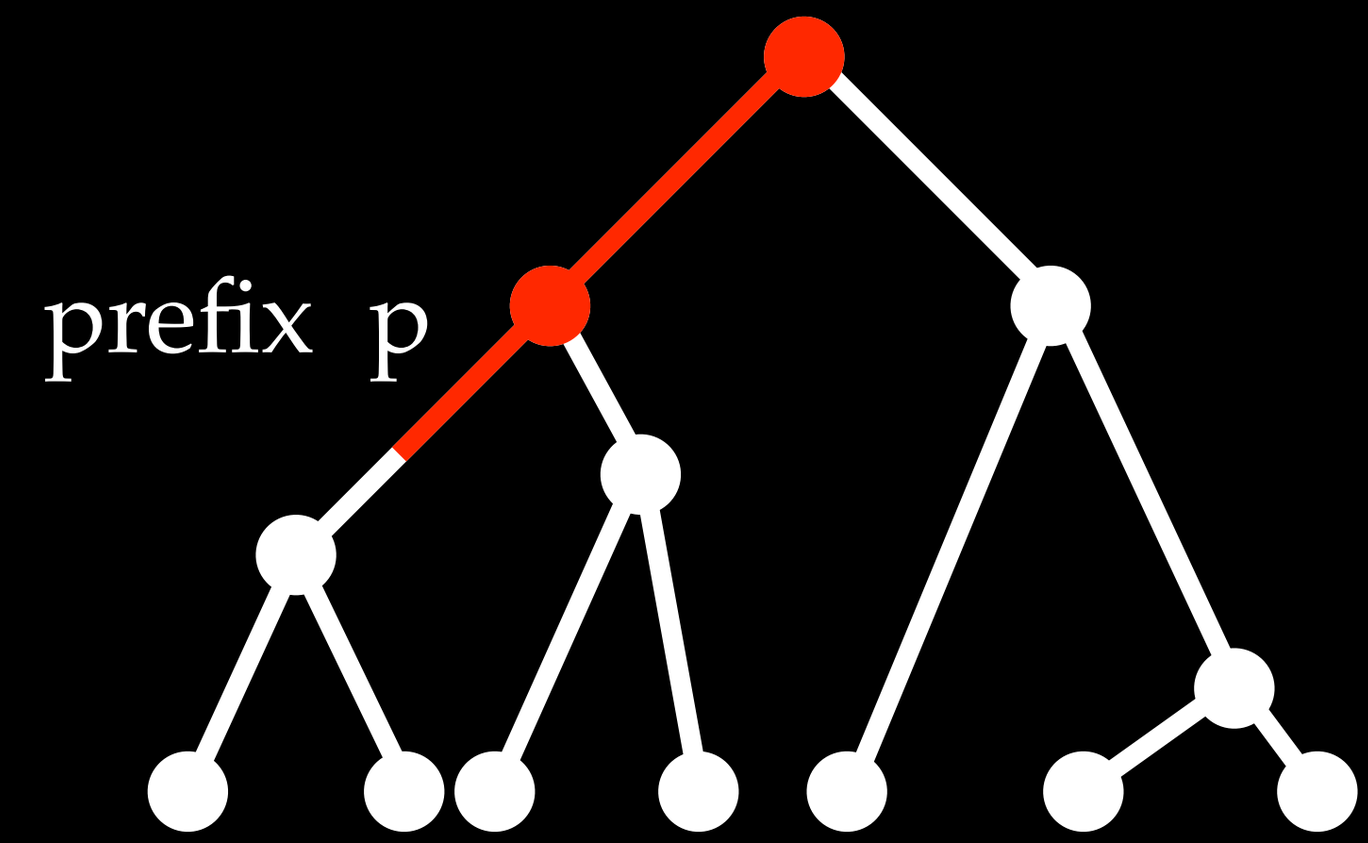
Strategy (simplified)



trie of strings in S

14

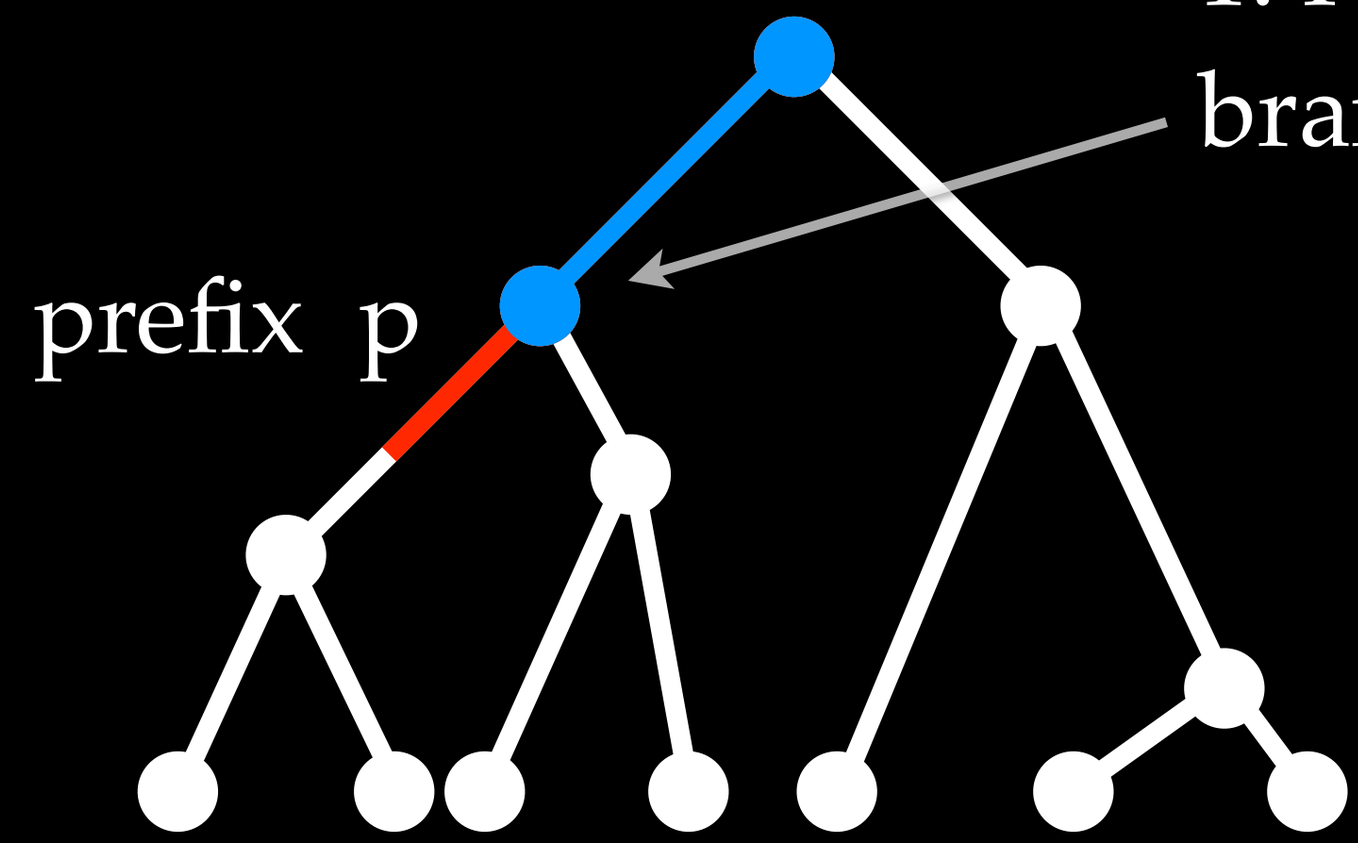
Strategy (simplified)



trie of strings in S

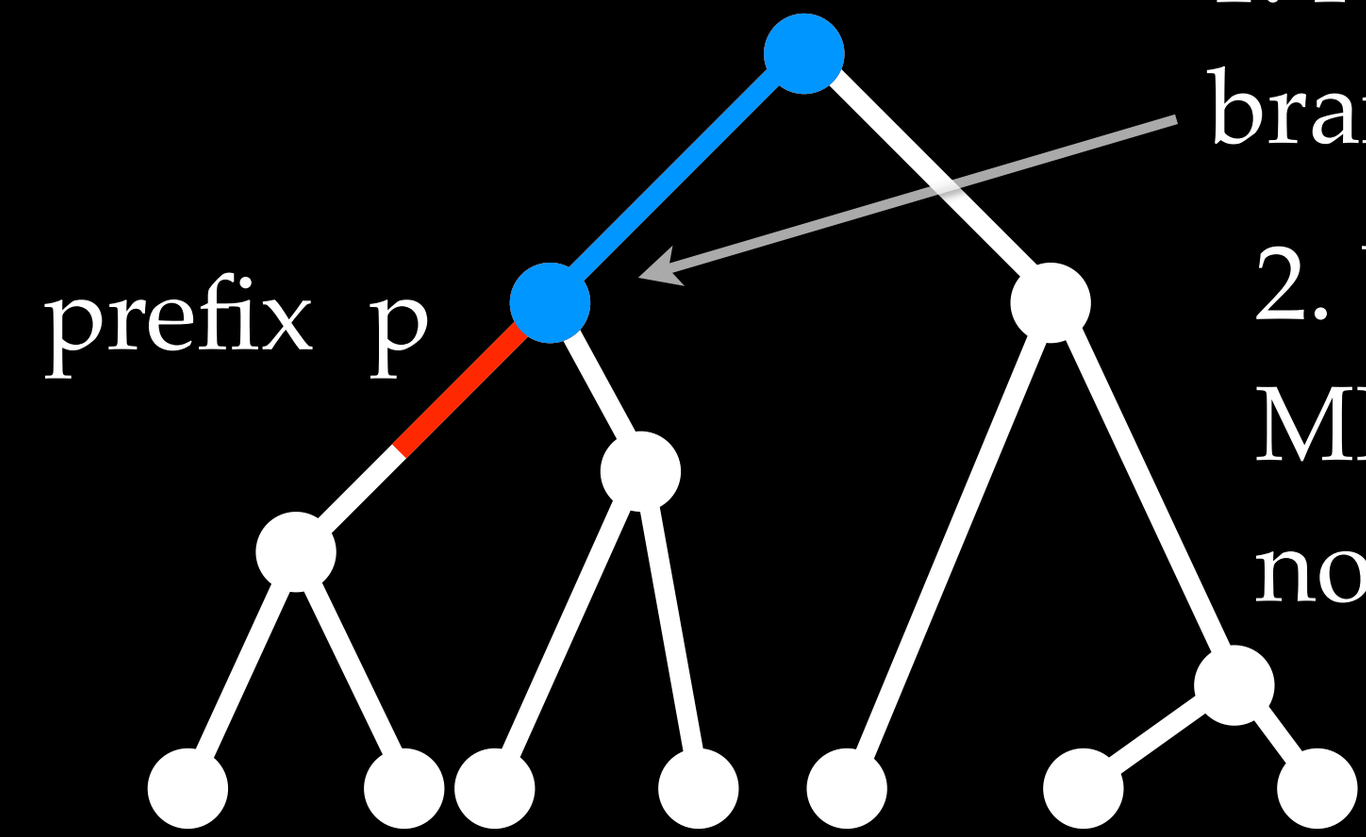
Strategy (simplified)

1. Find nearest branching node



trie of strings in S

Strategy (simplified)



1. Find nearest branching node

2. Use monotone MPH to map this node to rank.

O(n log w) bits

14

Storing a function

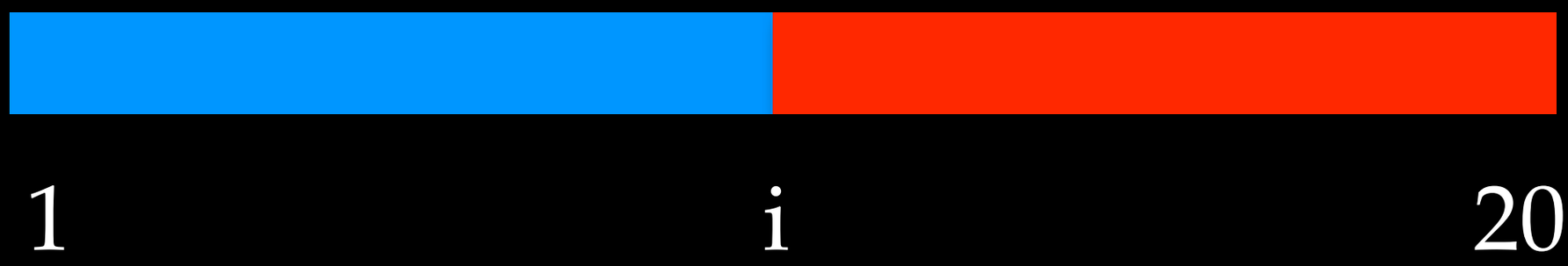
✤ Store a function f: $S \to \{0,1\}^r$.

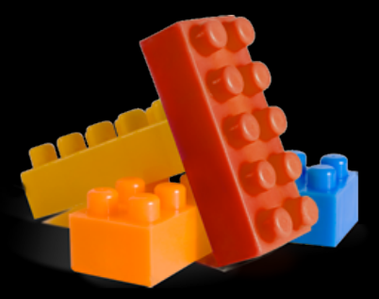
✤ O(nr) bits, time O(1)

[SS ‘89], [MWHC ‘96], [CKRT ‘04].

Fat binary search

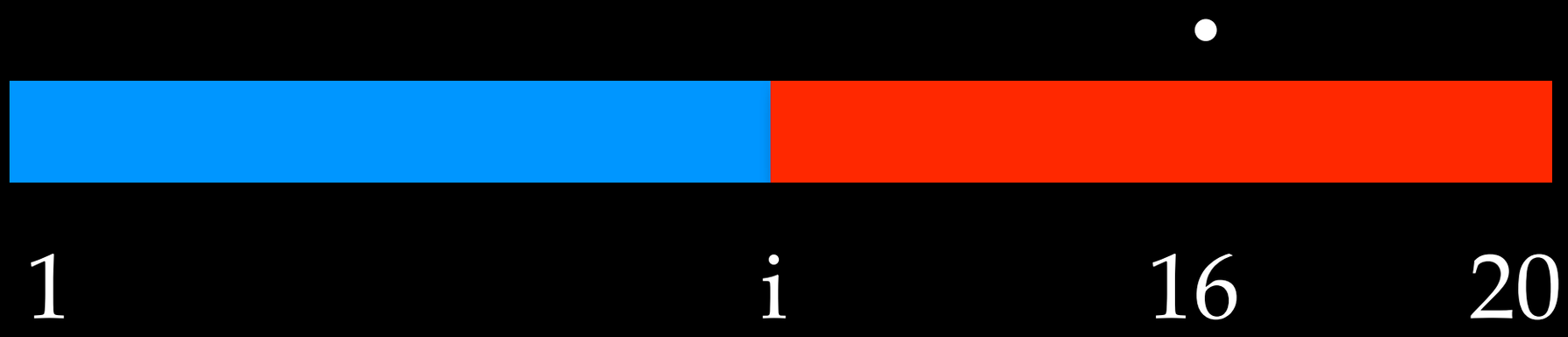
- Choose “middle point” to always have as many trailing 0s as possible.
- log w possible points on search that leads to i, for any starting interval.

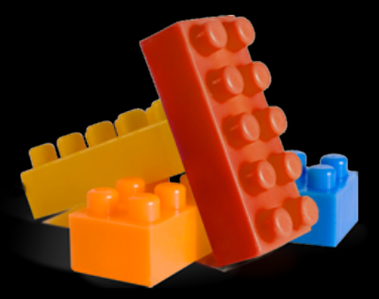




Fat binary search

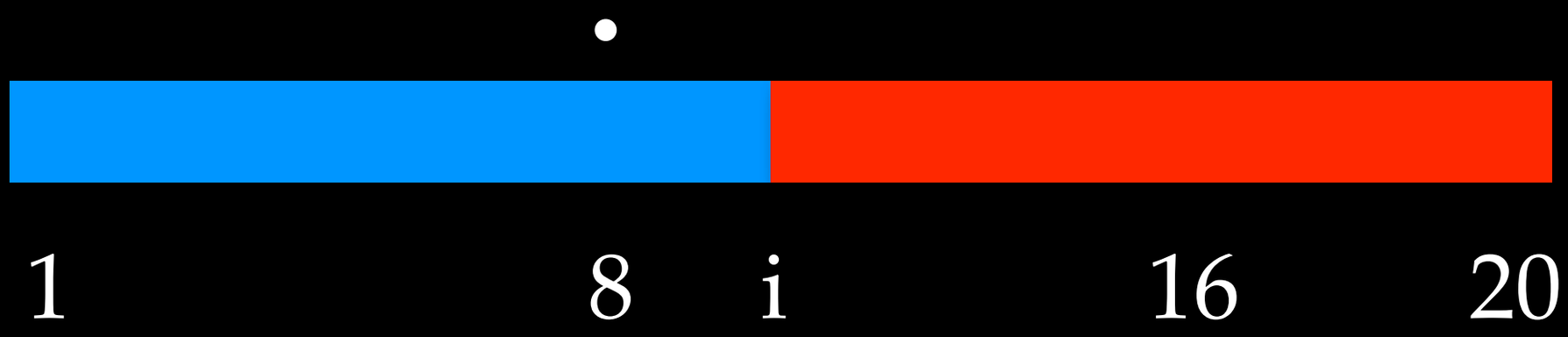
- Choose “middle point” to always have as many trailing 0s as possible.
- log w possible points on search that leads to i, for any starting interval.

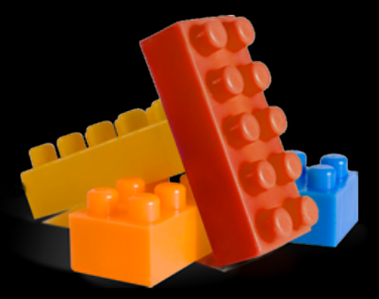




Fat binary search

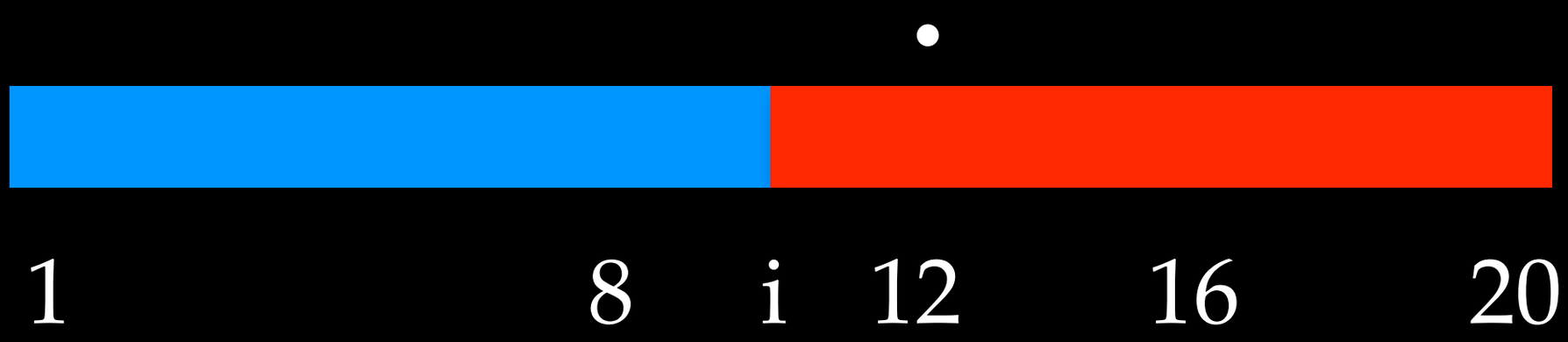
- Choose “middle point” to always have as many trailing 0s as possible.
- log w possible points on search that leads to i, for any starting interval.

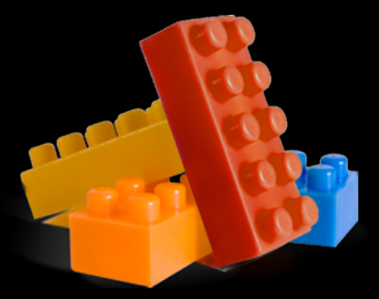




Fat binary search

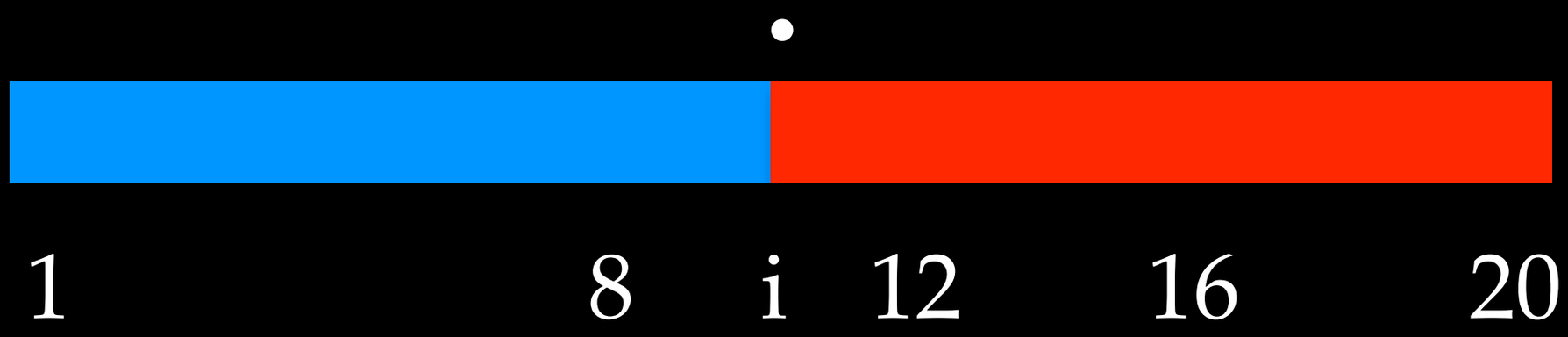
- Choose “middle point” to always have as many trailing 0s as possible.
- log w possible points on search that leads to i, for any starting interval.



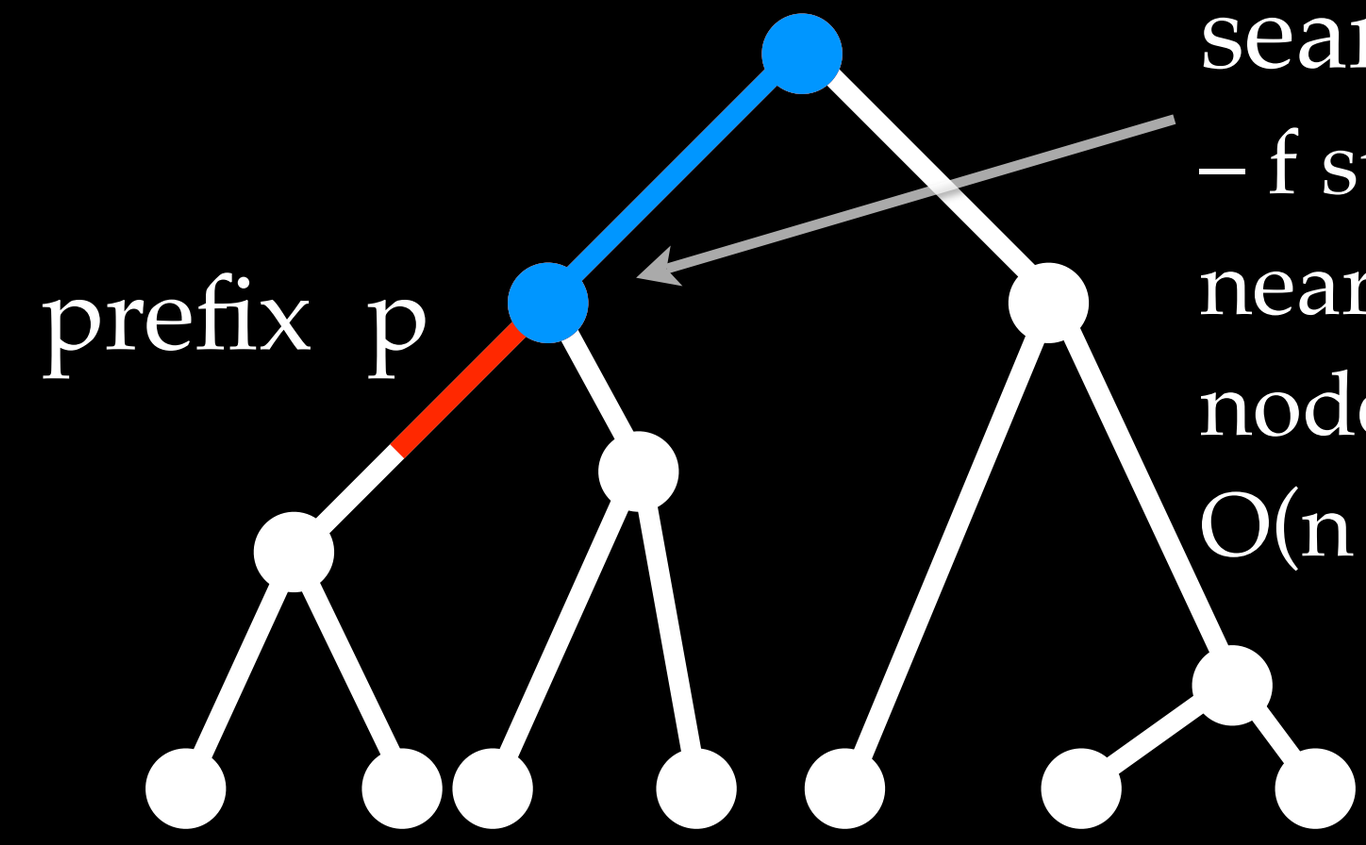


Fat binary search

- Choose “middle point” to always have as many trailing 0s as possible.
- log w possible points on search that leads to i, for any starting interval.

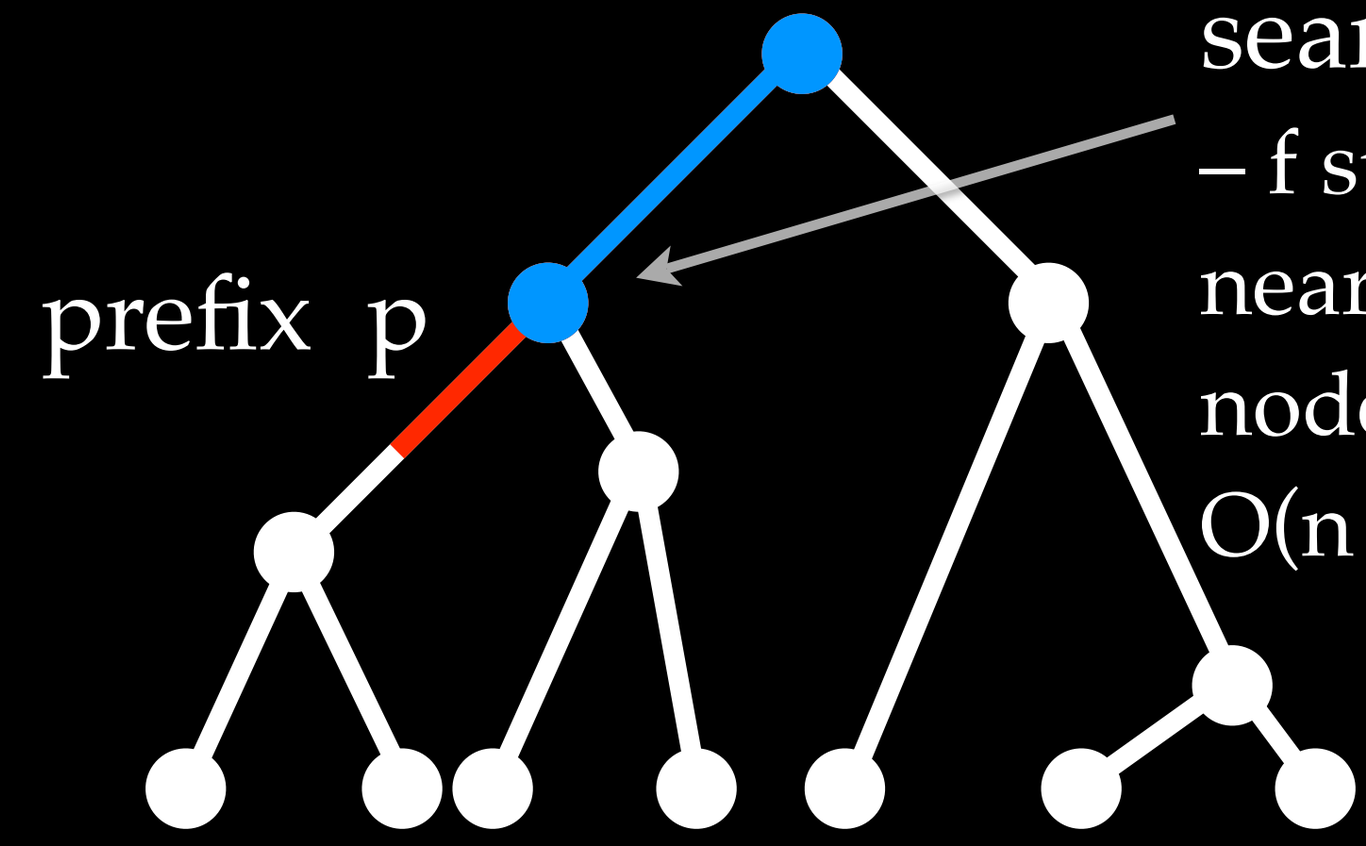


Strategy (simplified)



1. Fat binary search for length
– f stores depth of the nearest branching node for each prefix, $O(n \log^2 w)$ bits.

Strategy (simplified)



1. Fat binary search for length
– f stores depth of the nearest branching node for each prefix, O(n log w) bits.

Lower bound

- For n=2 consider the set H of strings with Hamming weight 1.
  - For distinct a,b∈H, at least one query distinguishes {0,a} and {0,b}.

00000100
00010000

18

Lower bound

- For n=2 consider the set H of strings with Hamming weight 1.
  - For distinct a,b∈H, at least one query distinguishes {0,a} and {0,b}.
  - Need |H|=w distinct data structures, i.e., at least log w bits.

00000100
00010000

18

Lower bound

- For n=2 consider the set H of strings with Hamming weight 1.
  - For distinct a,b∈H, at least one query distinguishes {0,a} and {0,b}.
  - Need |H|=w distinct data structures, i.e., at least log w bits.
- Generalization to n>2 straightforward.

00000100
00010000

18

5. Then what?

- Implications:
  - Prefix search with minimal number of accesses to slow memory.
  - Weak prefix counting + prefix minimum *without* accessing slow memory.
  - Range search with at most 2 extra accesses to slow memory.

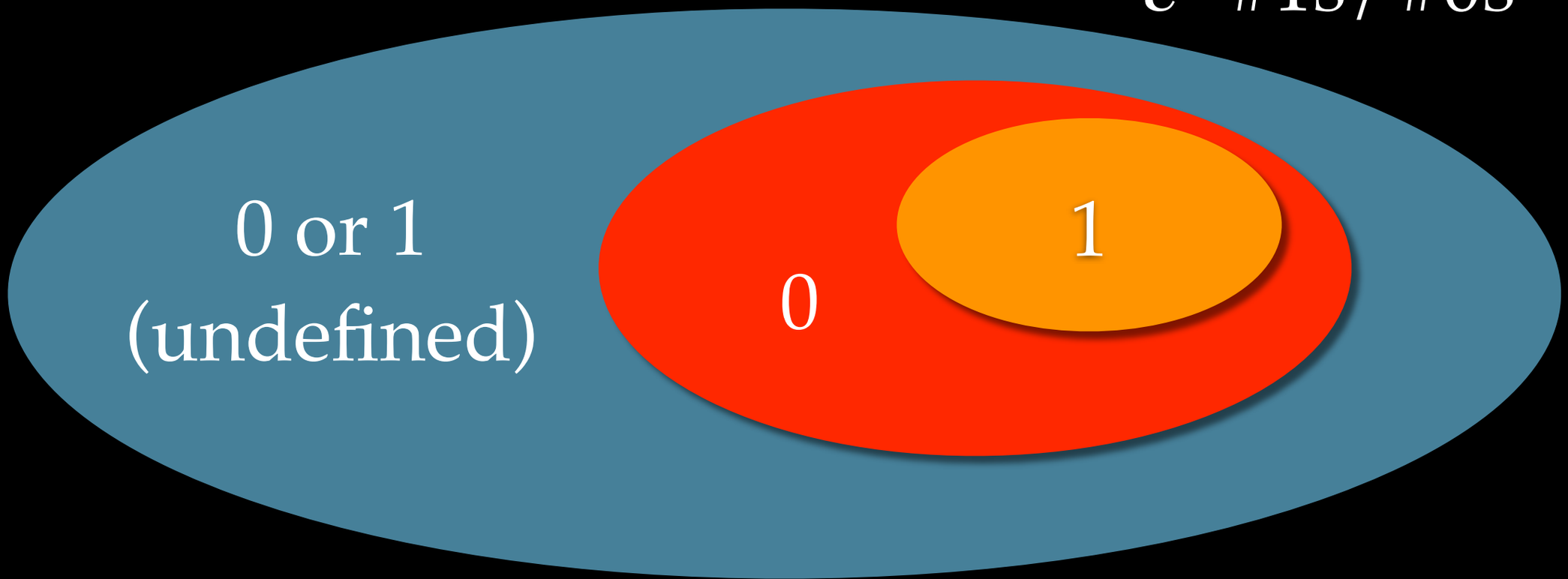
19

Open problems

- Tight space bound for monotone perfect hashing? Ω(n), O(n log log w) bits
- Is our time-space trade-off necessary? O(1) query time and O(n log w) space?
- Can *relative membership* be solved succinctly?

20

Relative membership



O(n log(1 / ε)) bits, O(1) time

[BBPV ‘09]

21