Elias-Fano Encoding

Succinct representation of monotone integer sequences with search operations

Giulio Ermanno Pibiri
giulio.pibiri@di.unipi.it
Computer Science Department
University of Pisa

21/06/2016

Problem

Consider a sequence S[0,n) of n *positive* and *monotonically increasing integers*, i.e., S[i-1] ≤ S[i] for 1 ≤ i ≤ n-1, possibly repeated.

How to represent it as a *bit vector* in which each original integer is *self-delimited*, using as few as possible bits?

Problem

Consider a sequence S[0,n) of n *positive* and *monotonically increasing integers*, i.e., S[i-1] ≤ S[i] for 1 ≤ i ≤ n-1, possibly repeated.

How to represent it as a *bit vector* in which each original integer is *self-delimited*, using as few as possible bits?

Huge research corpora describing different space/time trade-offs.

- Elias gamma/delta [Salomon-2007]
- Variable Byte [Salomon-2007]
- Varint-G8IU [Stepanov et al.-2011]
- Simple-9/16 [Anh and Moffat 2005-2010]
- PForDelta (PFD) [Zukowski et al.-2006]
- OptPFD [Yan et al.-2009]
- Binary Interpolative Coding [Moffat and Stuiver-2000]

Inverted Indexes

1

Given a *textual collection* D, each document can be seen as a (multi-)set of terms. The set of terms occurring in D is the *lexicon* T.

For each term t in T we store in a list L_t the identifiers of the documents in which t appears.

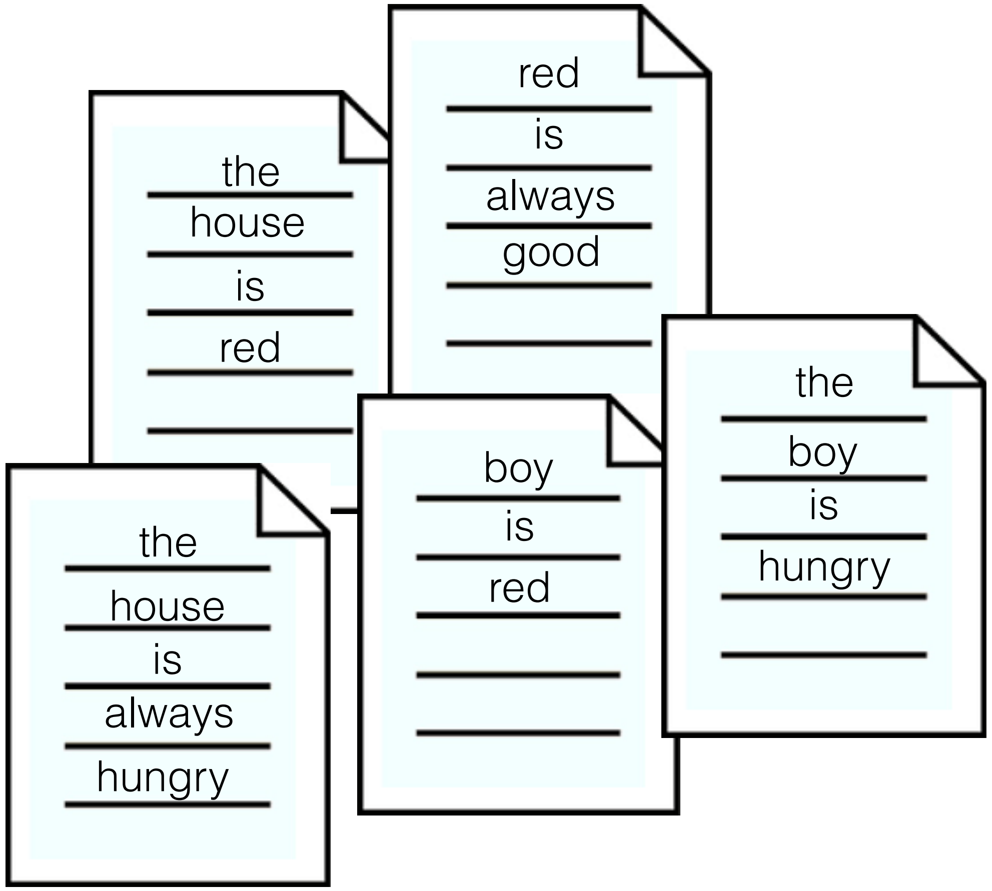
The collection of all inverted lists $\{L_{t_1},\ldots,L_{t_T}\}$ is the inverted index.

Inverted Indexes

Given a *textual collection* D, each document can be seen as a (multi-)set of terms. The set of terms occurring in D is the *lexicon* T.

For each term t in T we store in a list L_t the identifiers of the documents in which t appears.

The collection of all inverted lists $\{L_{t_1},\ldots,L_{t_T}\}$ is the inverted index.

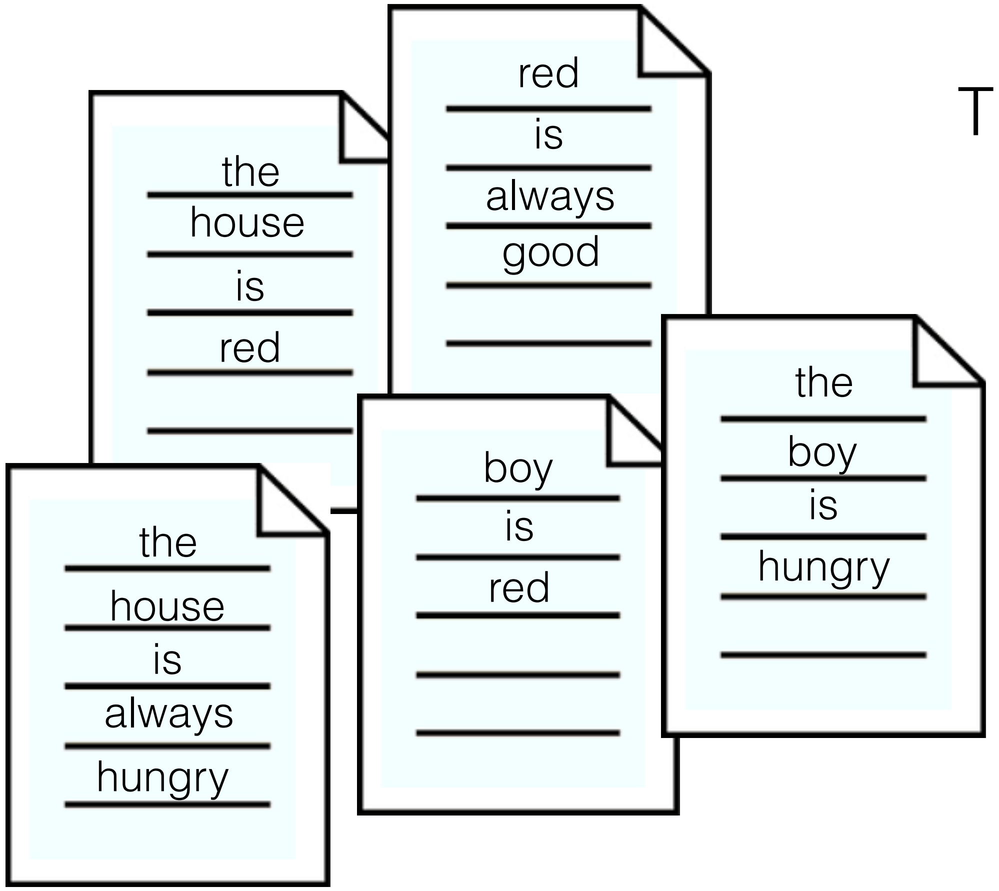


Inverted Indexes

Given a *textual collection* D, each document can be seen as a (multi-)set of terms. The set of terms occurring in D is the *lexicon* T.

For each term t in T we store in a list L_t the identifiers of the documents in which t appears.

The collection of all inverted lists $\{L_{t_1},\dots,L_{t_T}\}$ is the inverted index.



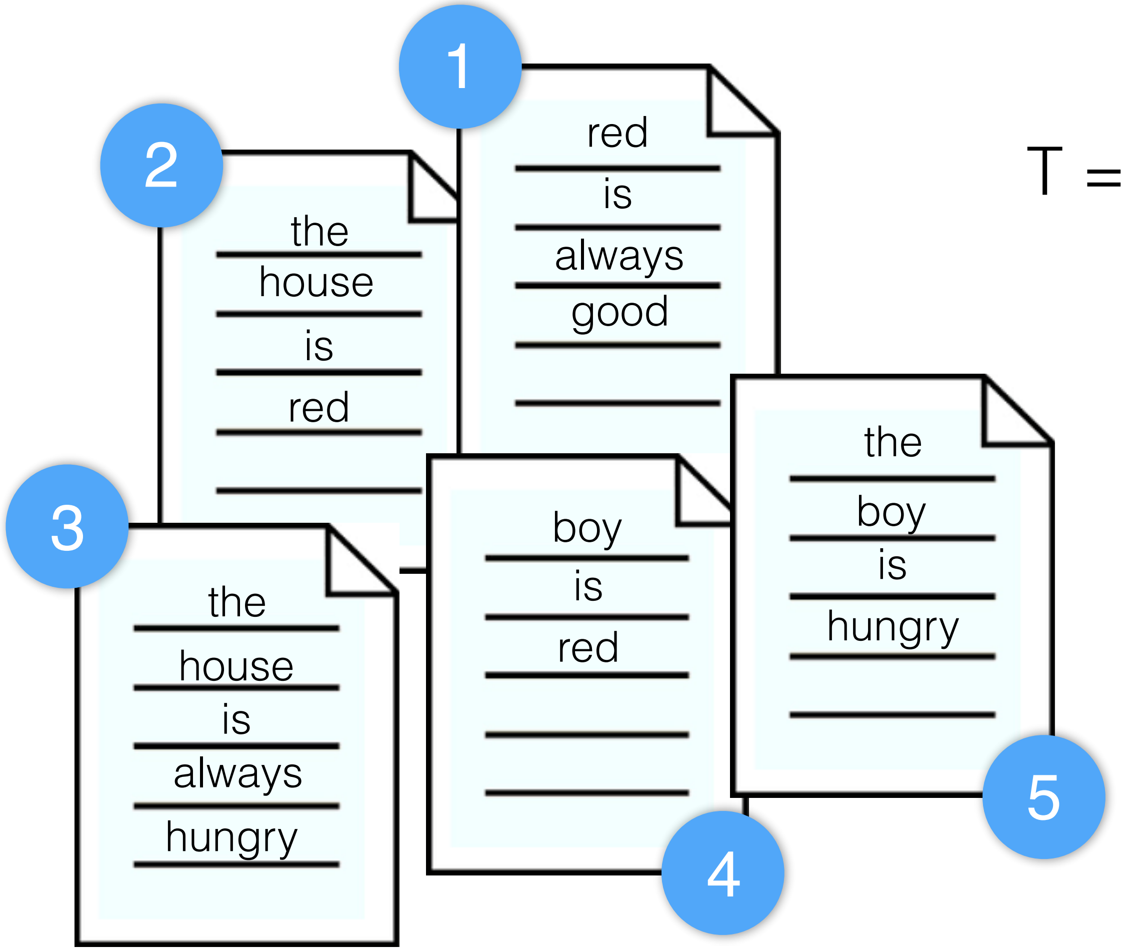
$$T = \{\overset{t_1}{\text{always}}, \overset{t_2}{\text{boy}}, \overset{t_3}{\text{good}}, \overset{t_4}{\text{house}}, \overset{t_5}{\text{hungry}}, \overset{t_6}{\text{is}}, \overset{t_7}{\text{red}}, \overset{t_8}{\text{the}}\}$$

Inverted Indexes

Given a *textual collection* D, each document can be seen as a (multi-)set of terms. The set of terms occurring in D is the *lexicon* T.

For each term t in T we store in a list L_t the identifiers of the documents in which t appears.

The collection of all inverted lists $\{L_{t_1},\dots,L_{t_T}\}$ is the inverted index.



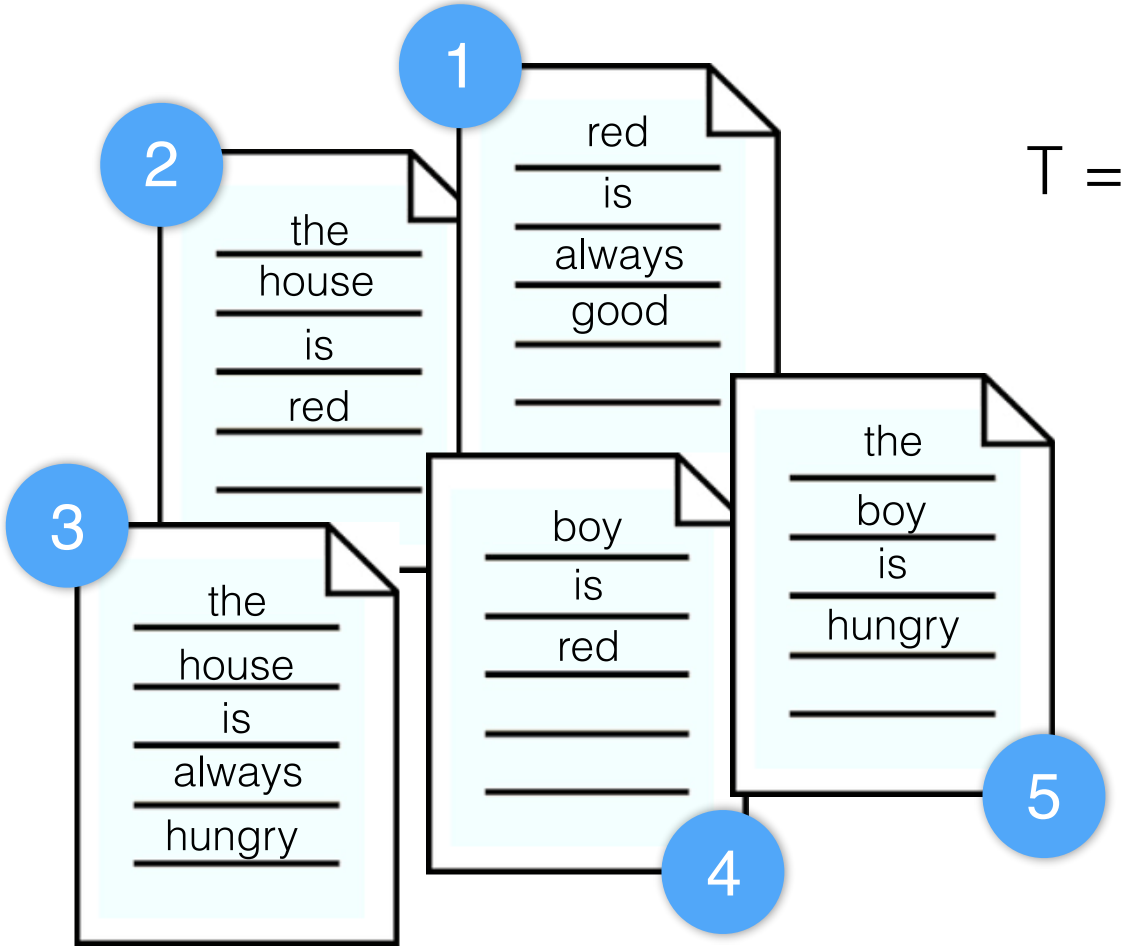
$$T = \{\overset{t_1}{\text{always}}, \overset{t_2}{\text{boy}}, \overset{t_3}{\text{good}}, \overset{t_4}{\text{house}}, \overset{t_5}{\text{hungry}}, \overset{t_6}{\text{is}}, \overset{t_7}{\text{red}}, \overset{t_8}{\text{the}}\}$$

Inverted Indexes

Given a *textual collection* D, each document can be seen as a (multi-)set of terms. The set of terms occurring in D is the *lexicon* T.

For each term t in T we store in a list L_t the identifiers of the documents in which t appears.

The collection of all inverted lists $\{L_{t_1},\ldots,L_{t_T}\}$ is the inverted index.



$$T = \{\overset{t_1}{\text{always}}, \overset{t_2}{\text{boy}}, \overset{t_3}{\text{good}}, \overset{t_4}{\text{house}}, \overset{t_5}{\text{hungry}}, \overset{t_6}{\text{is}}, \overset{t_7}{\text{red}}, \overset{t_8}{\text{the}}\}$$

→

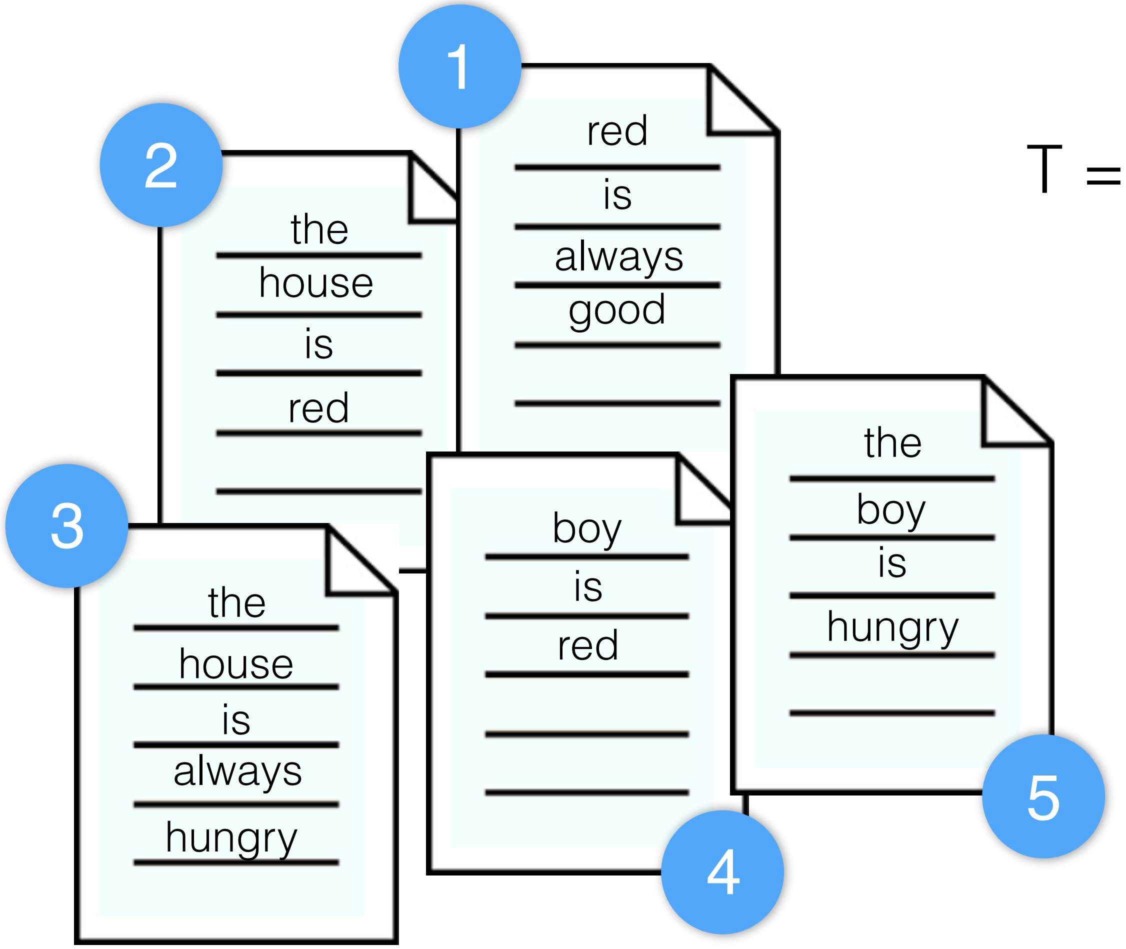
L_{t_1}=[1, 3]
L_{t_2}=[4, 5]
L_{t_3}=[1]
L_{t_4}=[2, 3]
L_{t_5}=[3, 5]
L_{t_6}=[1, 2, 3, 4, 5]
L_{t_7}=[1, 2, 4]
L_{t_8}=[2, 3, 5]

Inverted Indexes

Inverted Indexes owe their popularity to the *efficient resolution of queries*, such as: “return me all documents in which terms $\{t_1,\ldots,t_k\}$ occur”.

Inverted Indexes

Inverted Indexes owe their popularity to the *efficient resolution of queries*, such as: “return me all documents in which terms $\{t_1,\ldots,t_k\}$ occur”.



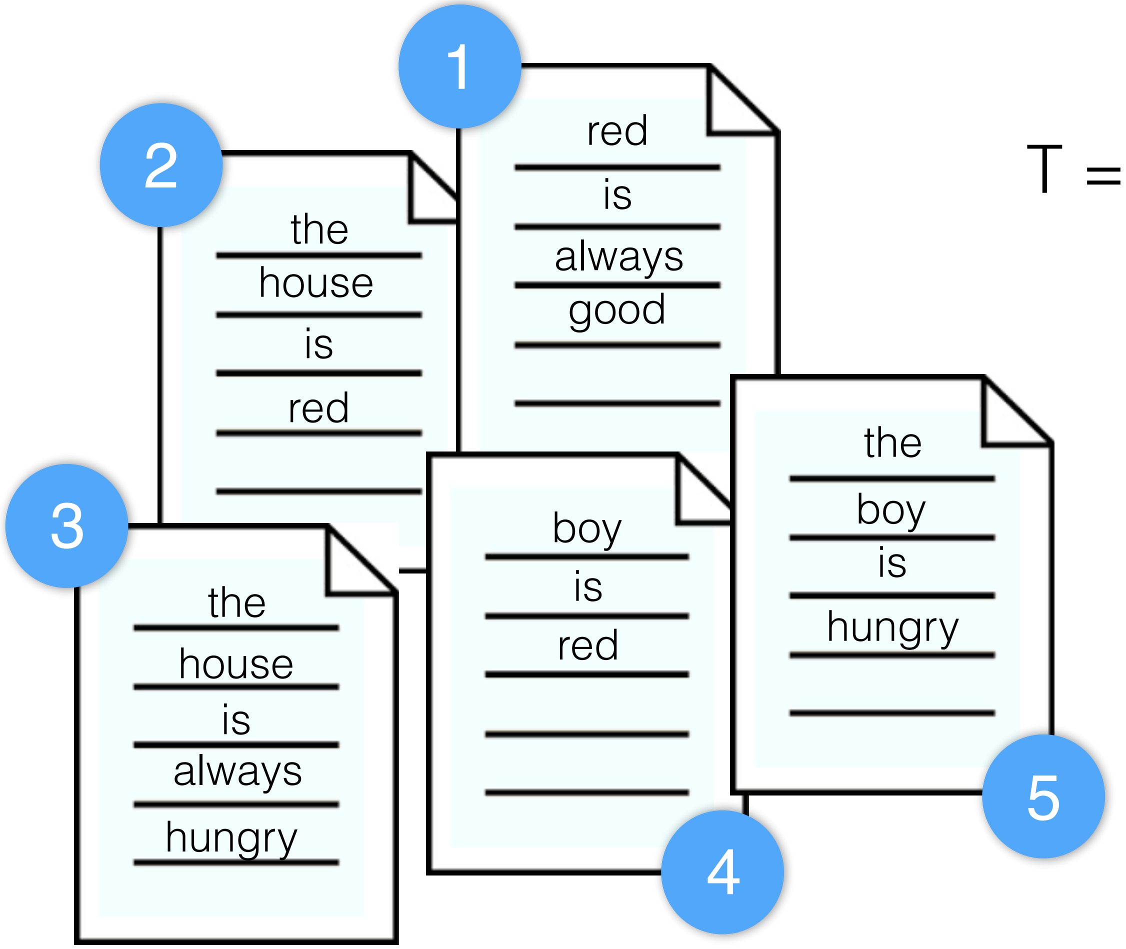
$$\begin{array}{cccccccc} t_1 & t_2 & t_3 & t_4 & t_5 & t_6 & t_7 & t_8 \end{array}$$

$T = \{\text{always, boy, good, house, hungry, is, red, the}\}$

$L_{t_1}=[1, 3]$
$L_{t_2}=[4, 5]$
$L_{t_3}=[1]$
$L_{t_4}=[2, 3]$
$L_{t_5}=[3, 5]$
$L_{t_6}=[1, 2, 3, 4, 5]$
$L_{t_7}=[1, 2, 4]$
$L_{t_8}=[2, 3, 5]$

Inverted Indexes

Inverted Indexes owe their popularity to the *efficient resolution of queries*, such as: “return me all documents in which terms $\{t_1,\ldots,t_k\}$ occur”.



1
red
is
always
good

2
the
house
is
red

3
the
house
is
always
hungry

4
boy
is
red

5
the
boy
is
hungry

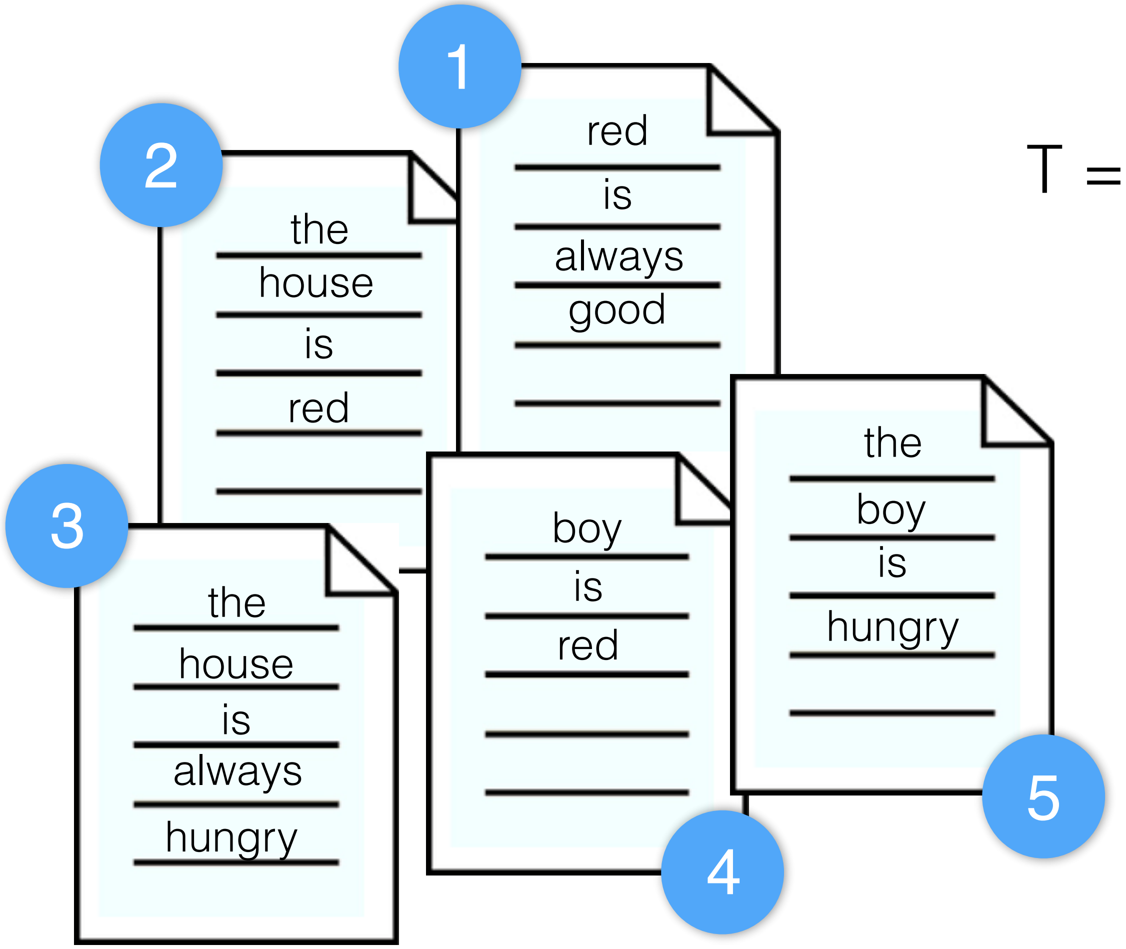
$$T = \{\overset{t_1}{\text{always}}, \overset{t_2}{\text{boy}}, \overset{t_3}{\text{good}}, \overset{t_4}{\text{house}}, \overset{t_5}{\text{hungry}}, \overset{t_6}{\text{is}}, \overset{t_7}{\text{red}}, \overset{t_8}{\text{the}}\}$$

$L_{t_1}=[1, 3]$
$L_{t_2}=[4, 5]$
$L_{t_3}=[1]$
$L_{t_4}=[2, 3]$
$L_{t_5}=[3, 5]$
$L_{t_6}=[1, 2, 3, 4, 5]$
$L_{t_7}=[1, 2, 4]$
$L_{t_8}=[2, 3, 5]$

$q = \{\text{boy, is, the}\}$

Inverted Indexes

Inverted Indexes owe their popularity to the *efficient resolution of queries*, such as: “return me all documents in which terms {$t_1,\ldots,t_k$} occur”.



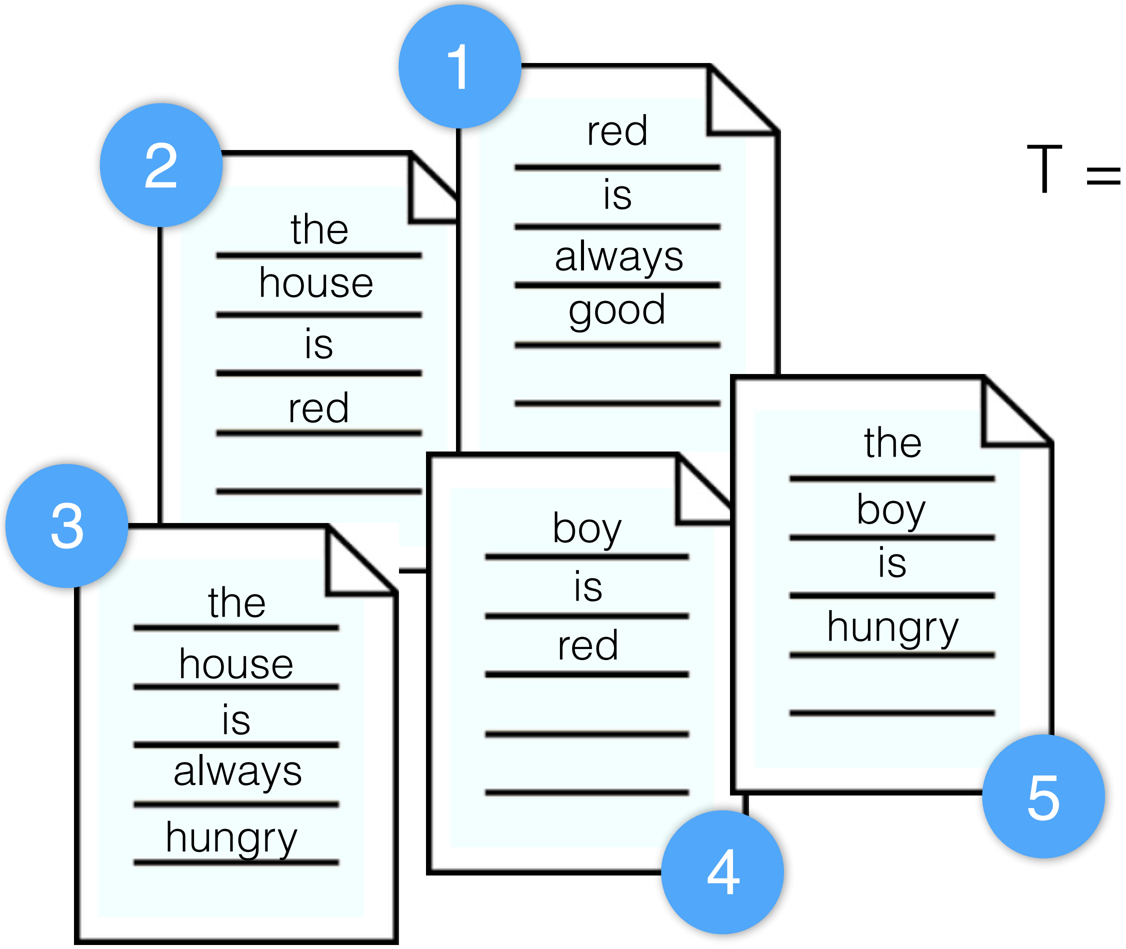
$$T = \{\overset{t_1}{\text{always}}, \overset{t_2}{\text{boy}}, \overset{t_3}{\text{good}}, \overset{t_4}{\text{house}}, \overset{t_5}{\text{hungry}}, \overset{t_6}{\text{is}}, \overset{t_7}{\text{red}}, \overset{t_8}{\text{the}}\}$$

$L_{t_1}=[1, 3]$
$L_{t_2}=[4, 5]$
$L_{t_3}=[1]$
$L_{t_4}=[2, 3]$
$L_{t_5}=[3, 5]$
$L_{t_6}=[1, 2, 3, 4, 5]$
$L_{t_7}=[1, 2, 4]$
$L_{t_8}=[2, 3, 5]$

q = {boy, is, the}

Inverted Indexes

Inverted Indexes owe their popularity to the *efficient resolution of queries*, such as: “return me all documents in which terms $\{t_1,\ldots,t_k\}$ occur”.



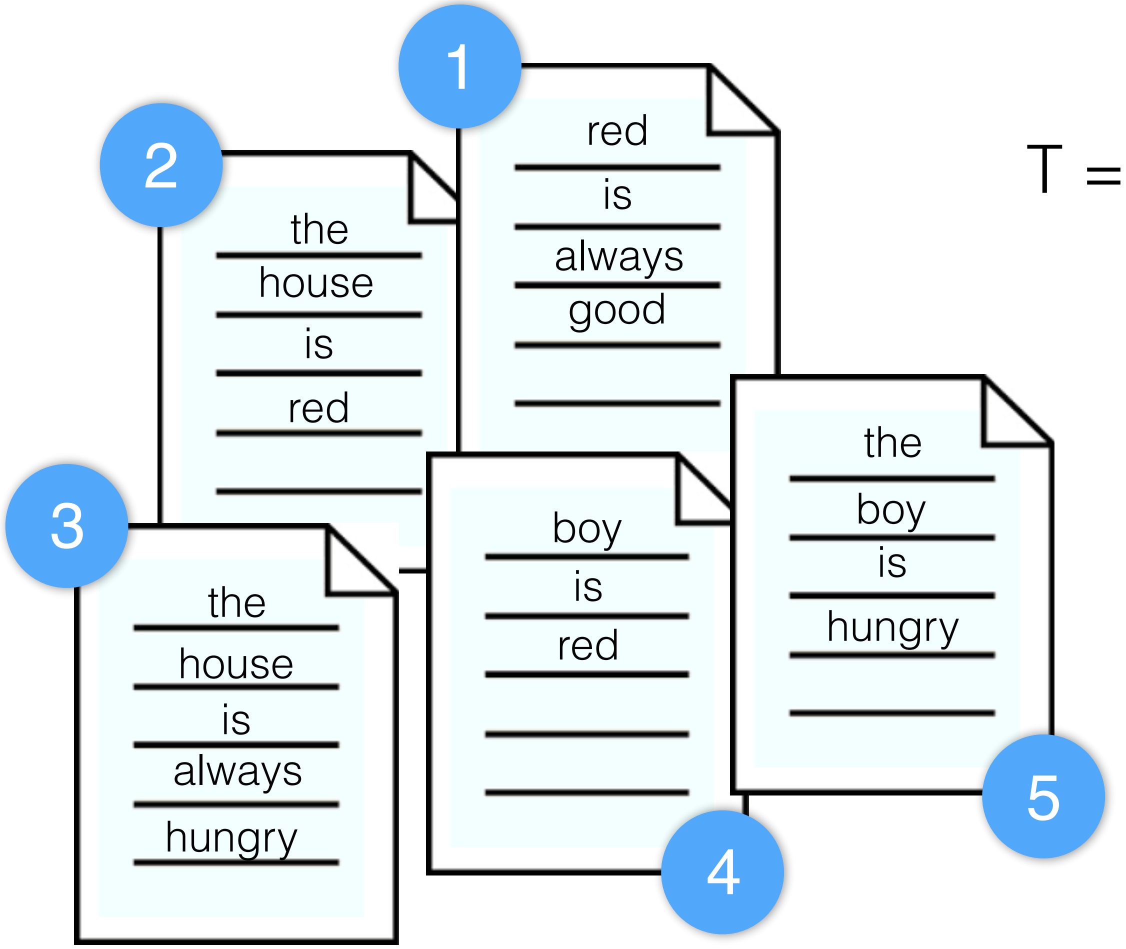
$$T = \{\overset{t_1}{\text{always}}, \overset{t_2}{\text{boy}}, \overset{t_3}{\text{good}}, \overset{t_4}{\text{house}}, \overset{t_5}{\text{hungry}}, \overset{t_6}{\text{is}}, \overset{t_7}{\text{red}}, \overset{t_8}{\text{the}}\}$$

$L_{t_1}=[1, 3]$
$L_{t_2}=[4, 5]$
$L_{t_3}=[1]$
$L_{t_4}=[2, 3]$
$L_{t_5}=[3, 5]$
$L_{t_6}=[1, 2, 3, 4, 5]$
$L_{t_7}=[1, 2, 4]$
$L_{t_8}=[2, 3, 5]$

$q = \{\text{boy, is, the}\}$

Inverted Indexes

Inverted Indexes owe their popularity to the *efficient resolution of queries*, such as: “return me all documents in which terms $\{t_1,\ldots,t_k\}$ occur”.



$$T = \{\overset{t_1}{always}, \overset{t_2}{boy}, \overset{t_3}{good}, \overset{t_4}{house}, \overset{t_5}{hungry}, \overset{t_6}{is}, \overset{t_7}{red}, \overset{t_8}{the}\}$$

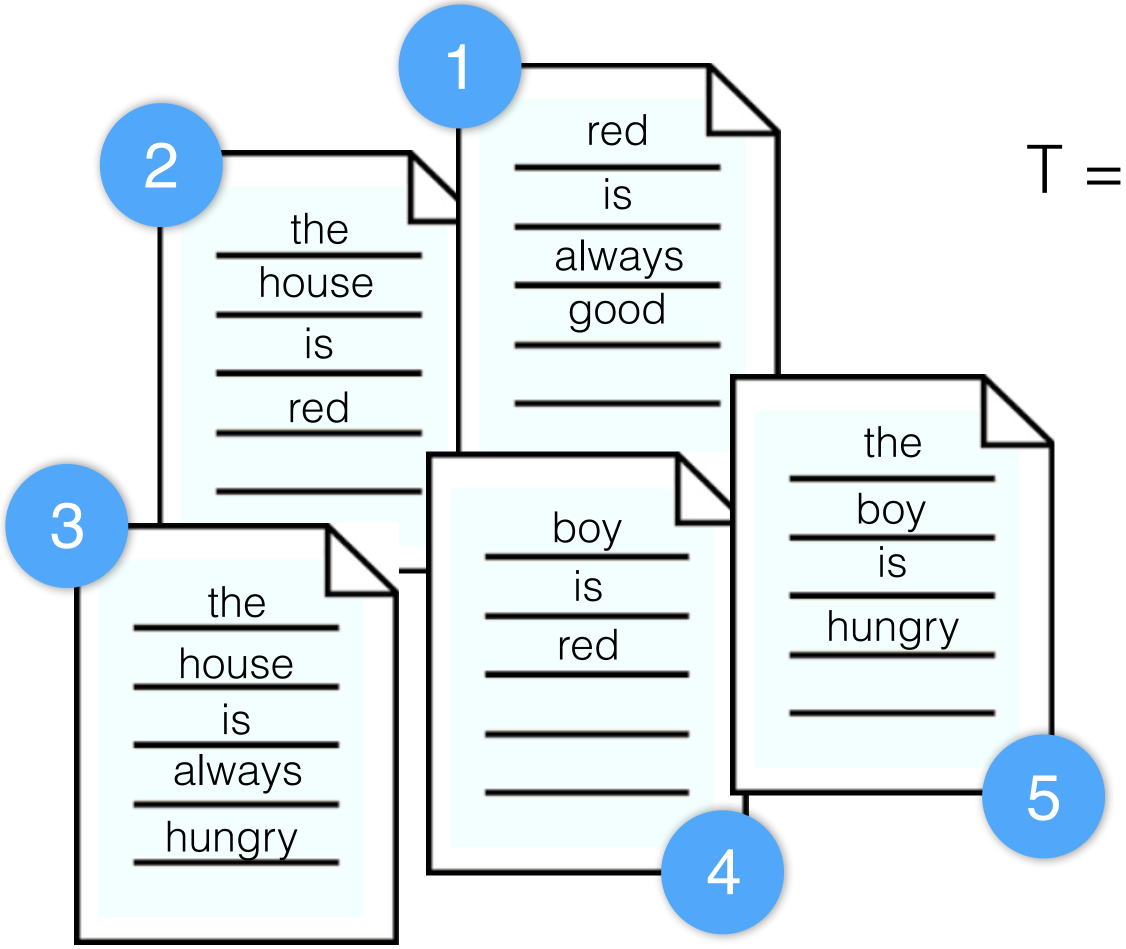
$L_{t_1}=[1, 3]$
$L_{t_2}=[4, 5]$
$L_{t_3}=[1]$
$L_{t_4}=[2, 3]$
$L_{t_5}=[3, 5]$
$L_{t_6}=[1, 2, 3, 4, 5]$
$L_{t_7}=[1, 2, 4]$
$L_{t_8}=[2, 3, 5]$

$q = \{boy, is, the\}$

$q = \{good, hungry\}$

Inverted Indexes

Inverted Indexes owe their popularity to the *efficient resolution of queries*, such as: “return me all documents in which terms $\{t_1,\ldots,t_k\}$ occur”.



$$T = \{\overset{t_1}{\text{always}}, \overset{t_2}{\text{boy}}, \overset{t_3}{\text{good}}, \overset{t_4}{\text{house}}, \overset{t_5}{\text{hungry}}, \overset{t_6}{\text{is}}, \overset{t_7}{\text{red}}, \overset{t_8}{\text{the}}\}$$

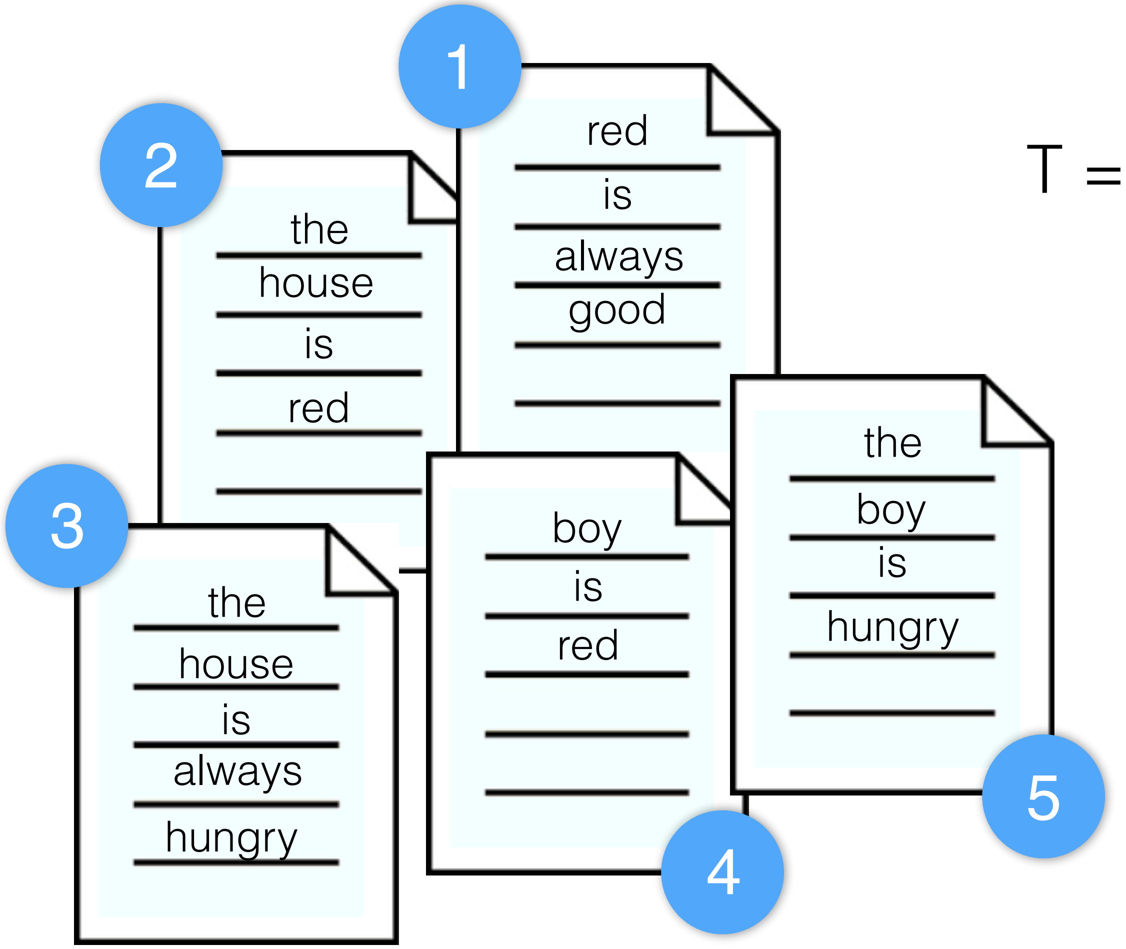
$L_{t_1}=[1, 3]$
$L_{t_2}=[4, 5]$
$L_{t_3}=[1]$
$L_{t_4}=[2, 3]$
$L_{t_5}=[3, 5]$
$L_{t_6}=[1, 2, 3, 4, 5]$
$L_{t_7}=[1, 2, 4]$
$L_{t_8}=[2, 3, 5]$

$q = \{\text{boy, is, the}\}$

$q = \{\text{good, hungry}\}$

Inverted Indexes

Inverted Indexes owe their popularity to the *efficient resolution of queries*, such as: “return me all documents in which terms $\{t_1,\ldots,t_k\}$ occur”.



$$T = \{\overset{t_1}{\text{always}}, \overset{t_2}{\text{boy}}, \overset{t_3}{\text{good}}, \overset{t_4}{\text{house}}, \overset{t_5}{\text{hungry}}, \overset{t_6}{\text{is}}, \overset{t_7}{\text{red}}, \overset{t_8}{\text{the}}\}$$

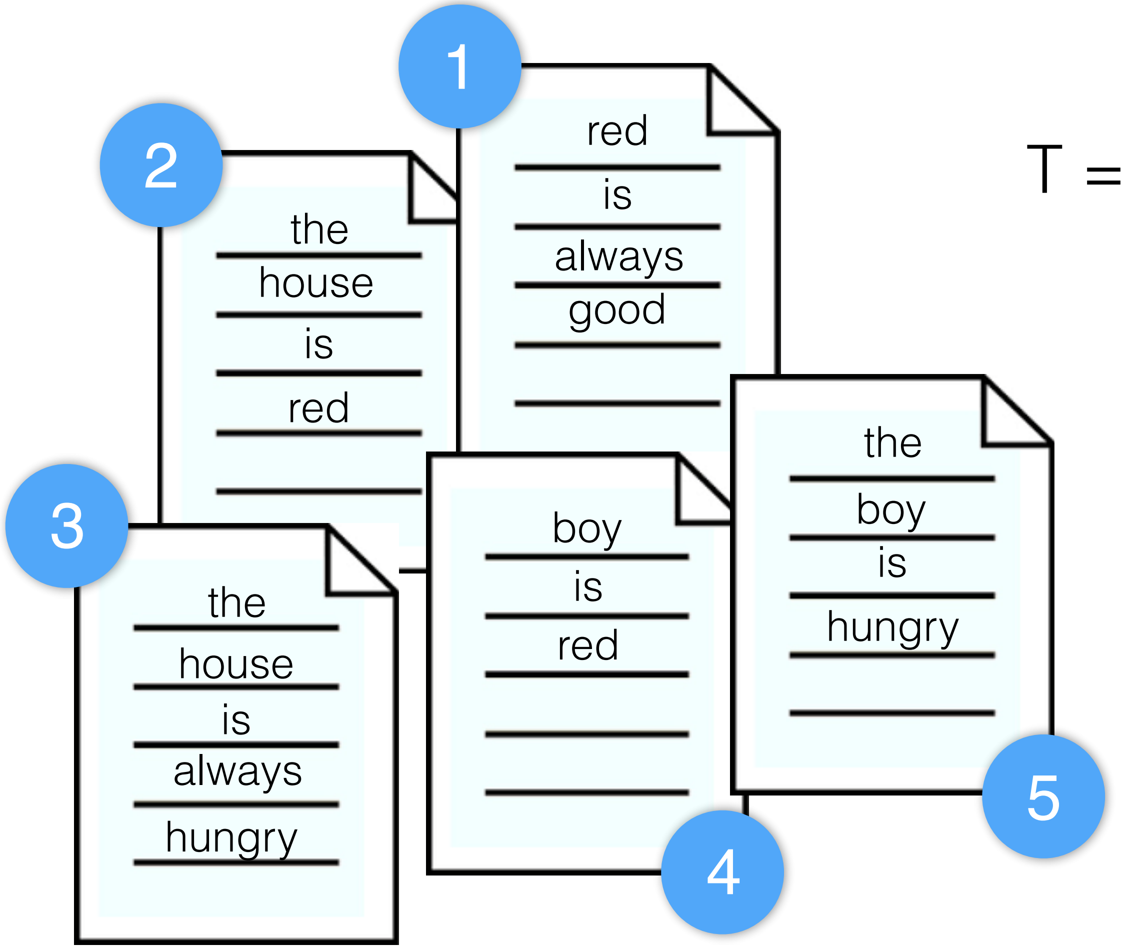
$L_{t_1}=[1, 3]$
$L_{t_2}=[4, 5]$
$L_{t_3}=[1]$
$L_{t_4}=[2, 3]$
$L_{t_5}=[3, 5]$
$L_{t_6}=[1, 2, 3, 4, 5]$
$L_{t_7}=[1, 2, 4]$
$L_{t_8}=[2, 3, 5]$

$q = \{\text{boy, is, the}\}$

$q = \{\text{good, hungry}\}$

Inverted Indexes

Inverted Indexes owe their popularity to the *efficient resolution of queries*, such as: “return me all documents in which terms $\{t_1,\ldots,t_k\}$ occur”.



$$T = \{\overset{t_1}{\text{always}}, \overset{t_2}{\text{boy}}, \overset{t_3}{\text{good}}, \overset{t_4}{\text{house}}, \overset{t_5}{\text{hungry}}, \overset{t_6}{\text{is}}, \overset{t_7}{\text{red}}, \overset{t_8}{\text{the}}\}$$

$L_{t_1}=[1, 3]$
$L_{t_2}=[4, 5]$
$L_{t_3}=[1]$
$L_{t_4}=[2, 3]$
$L_{t_5}=[3, 5]$
$L_{t_6}=[1, 2, 3, 4, 5]$
$L_{t_7}=[1, 2, 4]$
$L_{t_8}=[2, 3, 5]$

$q = \{\text{boy}, \text{is}, \text{the}\}$

$q = \{\text{good}, \text{hungry}\}$

inverted lists intersection

Genesis - 1970s

Peter Elias
[1923 - 2001]

Robert Fano
[1917 -]

Robert Fano. *On the number of bits required to implement an associative memory*. Memorandum 61, Computer Structures Group, MIT (1971).

Peter Elias. *Efficient Storage and Retrieval by Content and Address of Static Files*. Journal of the ACM (JACM) 21, 2, 246–260 (1974).

Genesis - 1970s

Peter Elias
[1923 - 2001]

Robert Fano
[1917 -]

Robert Fano. *On the number of bits required to implement an associative memory*. Memorandum 61, Computer Structures Group, MIT (1971).

Peter Elias. *Efficient Storage and Retrieval by Content and Address of Static Files*. Journal of the ACM (JACM) 21, 2, 246–260 (1974).

40 years later!

Sebastiano Vigna. *Quasi-succinct indices*.

In Proceedings of the 6-th ACM International Conference on Web Search and Data Mining (WSDM), 83-92 (2013).

Elias-Fano solution

3	1
4	2
7	3
13	4
14	5
15	6
21	7
43	8

Elias-Fano solution

	3	1
	4	2
	7	3
	13	4
	14	5
	15	6
	21	7
u =	43	8

Elias-Fano solution

0 0 0 0 1 1	3	1
0 0 0 1 0 0	4	2
0 0 0 1 1 1	7	3
0 0 1 1 0 1	13	4
0 0 1 1 1 0	14	5
0 0 1 1 1 1	15	6
0 1 0 1 0 1	21	7
1 0 1 0 1 1	u = 43	8

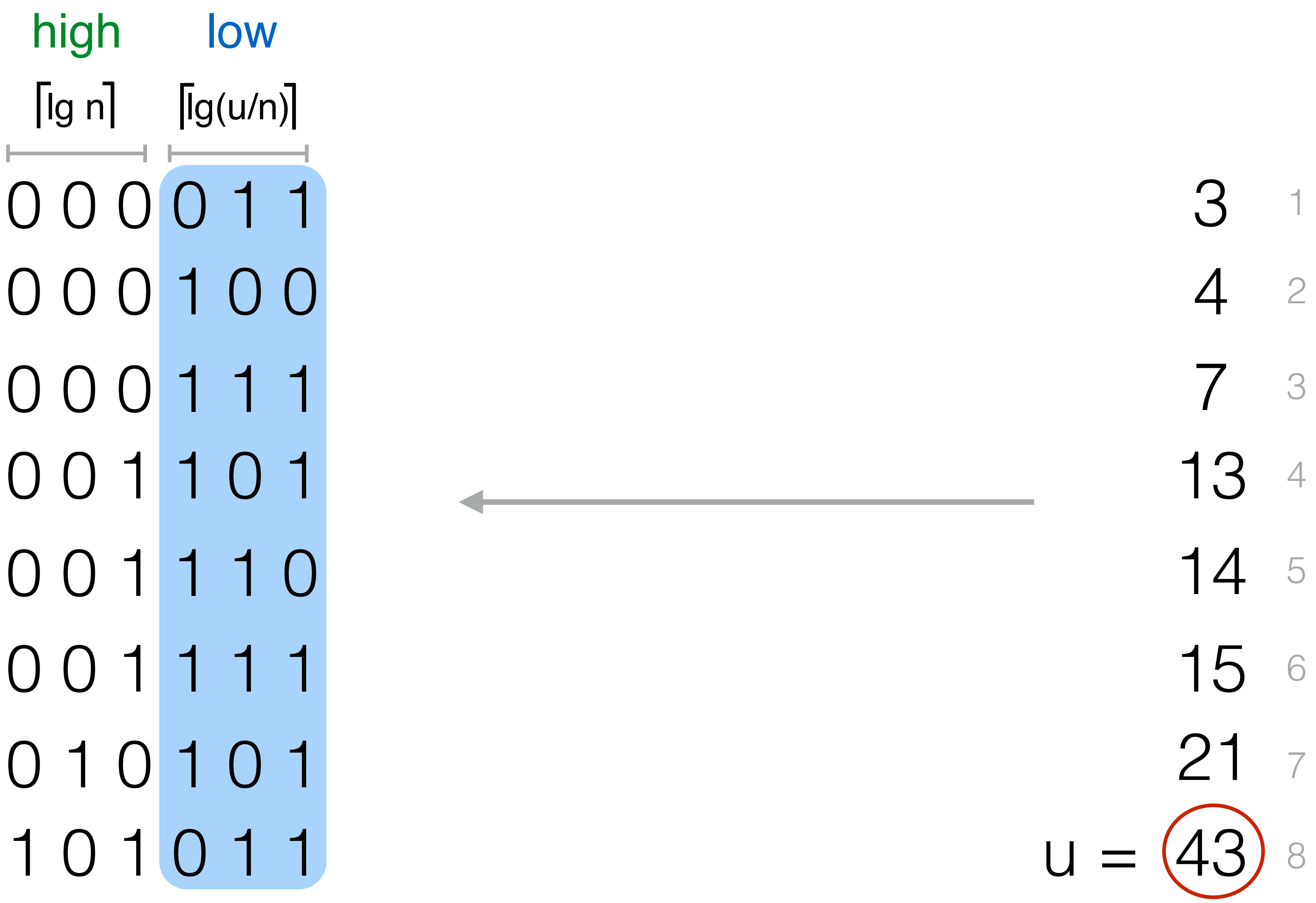
Elias-Fano solution

high $\lceil \lg n \rceil$	low $\lceil \lg(u/n) \rceil$		
0 0 0	0 1 1	3	1
0 0 0	1 0 0	4	2
0 0 0	1 1 1	7	3
0 0 1	1 0 1	13	4
0 0 1	1 1 0	14	5
0 0 1	1 1 1	15	6
0 1 0	1 0 1	21	7
1 0 1	0 1 1	u = 43	8

Elias-Fano solution

high $\lceil \lg n \rceil$	low $\lceil \lg(u/n) \rceil$		
0 0 0	0 1 1	3	1
0 0 0	1 0 0	4	2
0 0 0	1 1 1	7	3
0 0 1	1 0 1	13	4
0 0 1	1 1 0	14	5
0 0 1	1 1 1	15	6
0 1 0	1 0 1	21	7
1 0 1	0 1 1	u = 43	8

Elias-Fano solution



high ⌈lg n⌉	low ⌈lg(u/n)⌉		
0 0 0	0 1 1	3	1
0 0 0	1 0 0	4	2
0 0 0	1 1 1	7	3
0 0 1	1 0 1	13	4
0 0 1	1 1 0	14	5
0 0 1	1 1 1	15	6
0 1 0	1 0 1	21	7
1 0 1	0 1 1	u = 43	8

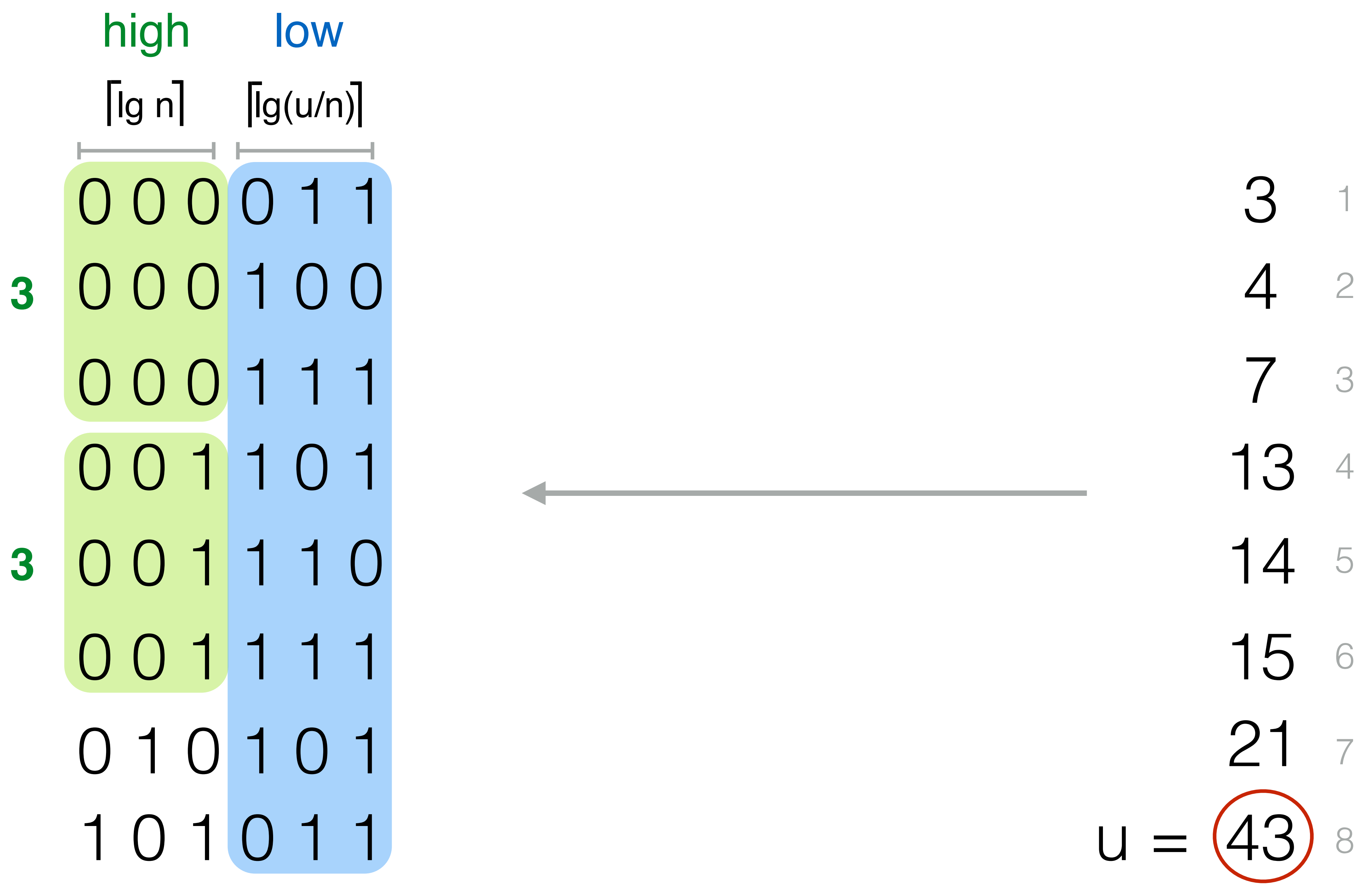
L = 011100111101110111101011

Elias-Fano solution

	high $\lceil \lg n \rceil$	low $\lceil \lg(u/n) \rceil$		
	0 0 0	0 1 1	3	1
3	0 0 0	1 0 0	4	2
	0 0 0	1 1 1	7	3
	0 0 1	1 0 1	13	4
	0 0 1	1 1 0	14	5
	0 0 1	1 1 1	15	6
	0 1 0	1 0 1	21	7
	1 0 1	0 1 1	u = 43	8

L = 011100111101110111101011

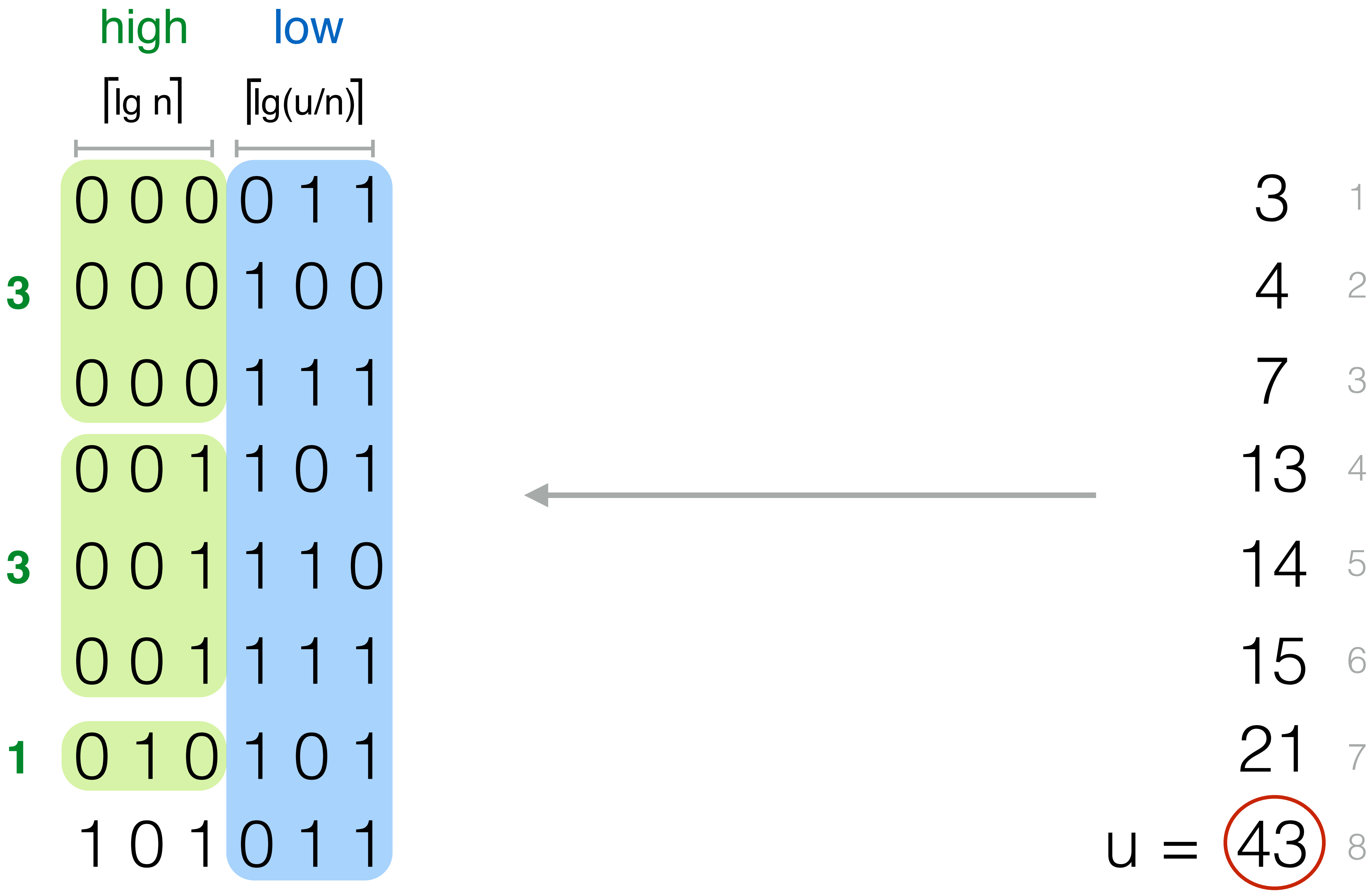
Elias-Fano solution



	high	low		
	⌈lg n⌉	⌈lg(u/n)⌉		
	0 0 0	0 1 1	3	1
3	0 0 0	1 0 0	4	2
	0 0 0	1 1 1	7	3
	0 0 1	1 0 1	13	4
3	0 0 1	1 1 0	14	5
	0 0 1	1 1 1	15	6
	0 1 0	1 0 1	21	7
	1 0 1	0 1 1	u = 43	8

L = 011100111101110111101011

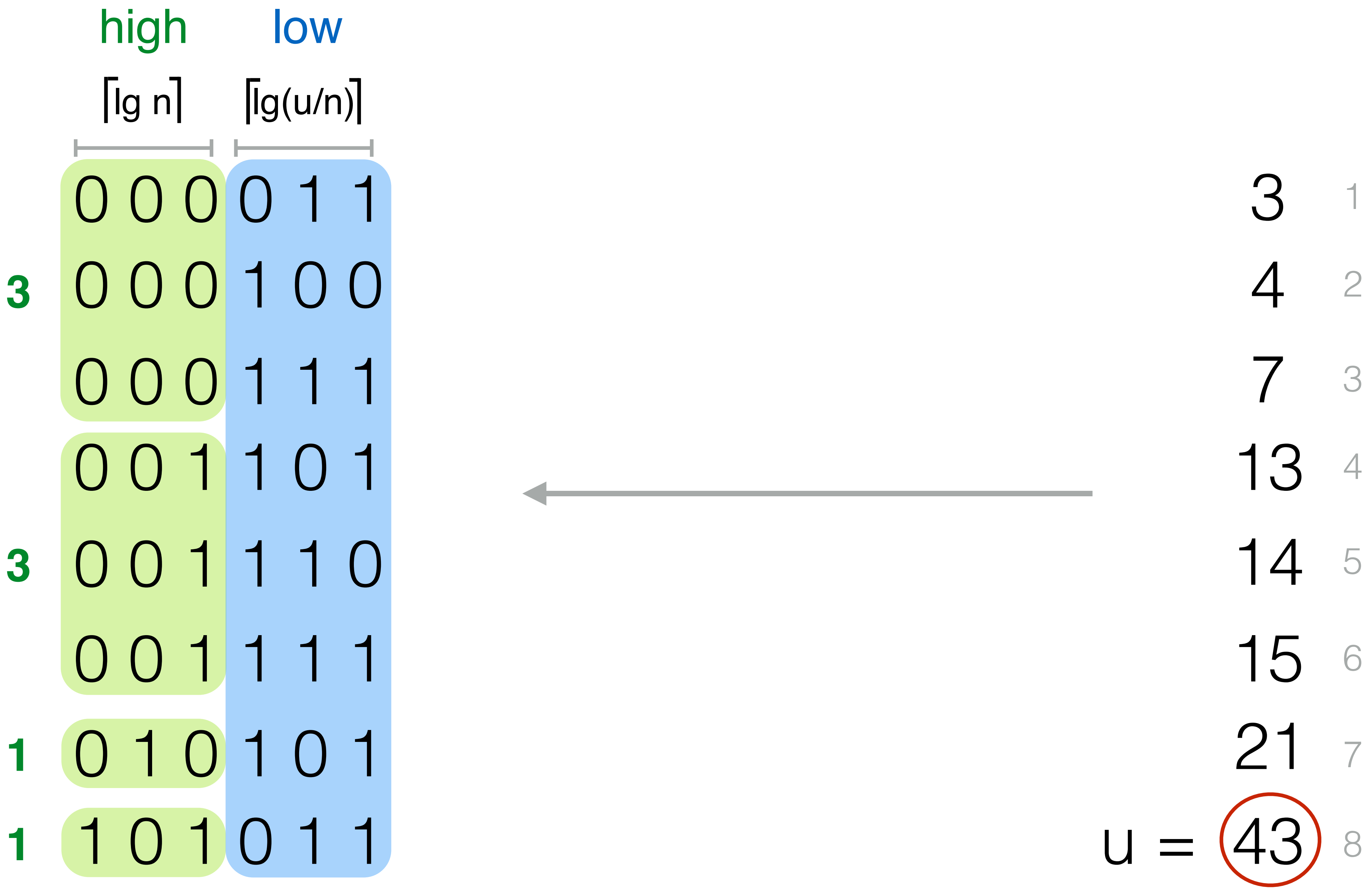
Elias-Fano solution



	high ⌈lg n⌉	low ⌈lg(u/n)⌉		
	0 0 0	0 1 1	3	1
3	0 0 0	1 0 0	4	2
	0 0 0	1 1 1	7	3
	0 0 1	1 0 1	13	4
3	0 0 1	1 1 0	14	5
	0 0 1	1 1 1	15	6
1	0 1 0	1 0 1	21	7
	1 0 1	0 1 1	u = 43	8

L = 011100111101110111101011

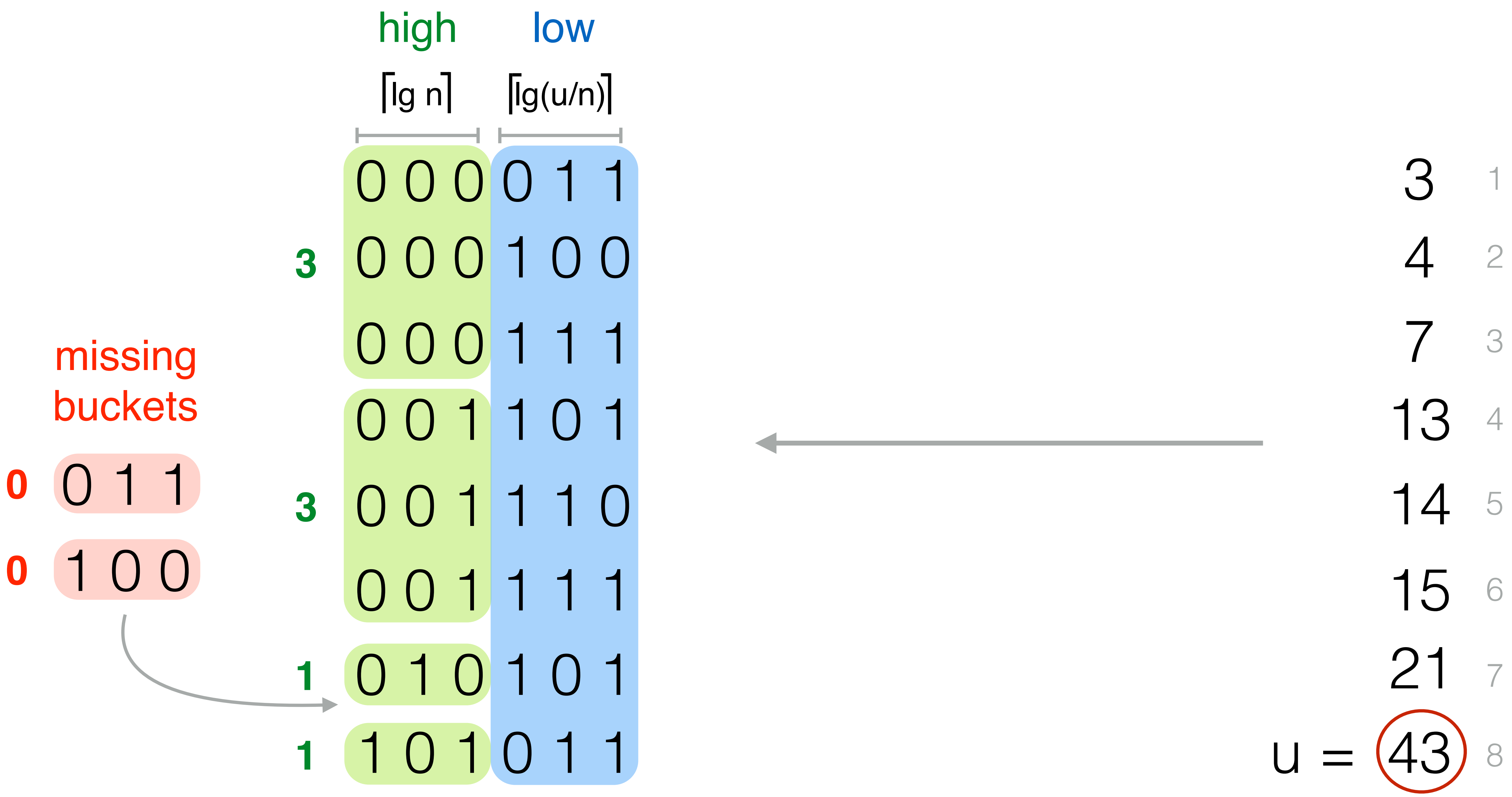
Elias-Fano solution



	high ⌈lg n⌉	low ⌈lg(u/n)⌉		
	0 0 0	0 1 1	3	1
3	0 0 0	1 0 0	4	2
	0 0 0	1 1 1	7	3
	0 0 1	1 0 1	13	4
3	0 0 1	1 1 0	14	5
	0 0 1	1 1 1	15	6
1	0 1 0	1 0 1	21	7
1	1 0 1	0 1 1	u = 43	8

L = 011100111101110111101011

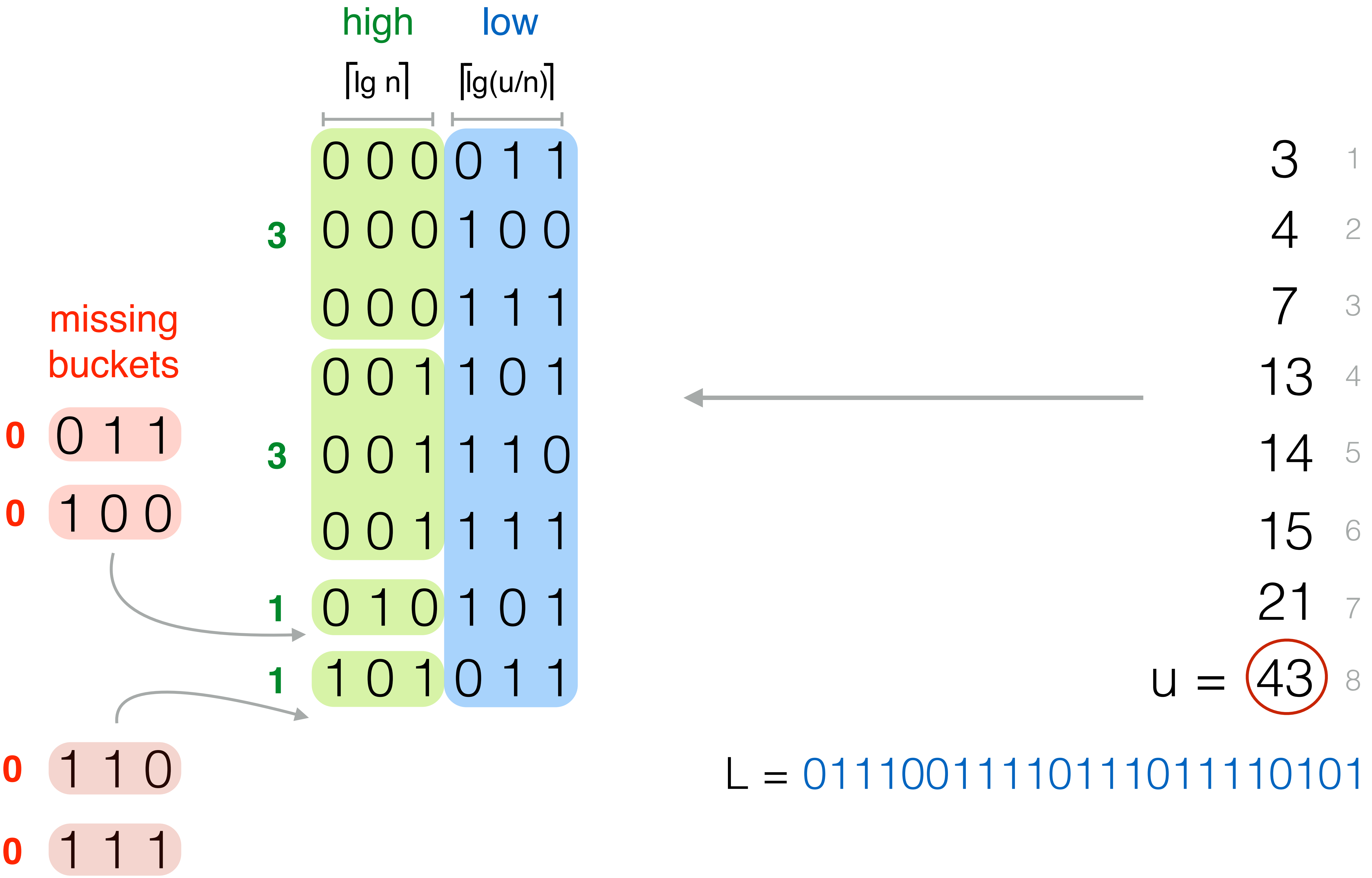
Elias-Fano solution



	high ⌈lg n⌉	low ⌈lg(u/n)⌉		
3	0 0 0	0 1 1	3	1
	0 0 0	1 0 0	4	2
	0 0 0	1 1 1	7	3
3	0 0 1	1 0 1	13	4
	0 0 1	1 1 0	14	5
	0 0 1	1 1 1	15	6
1	0 1 0	1 0 1	21	7
1	1 0 1	0 1 1	u = 43	8

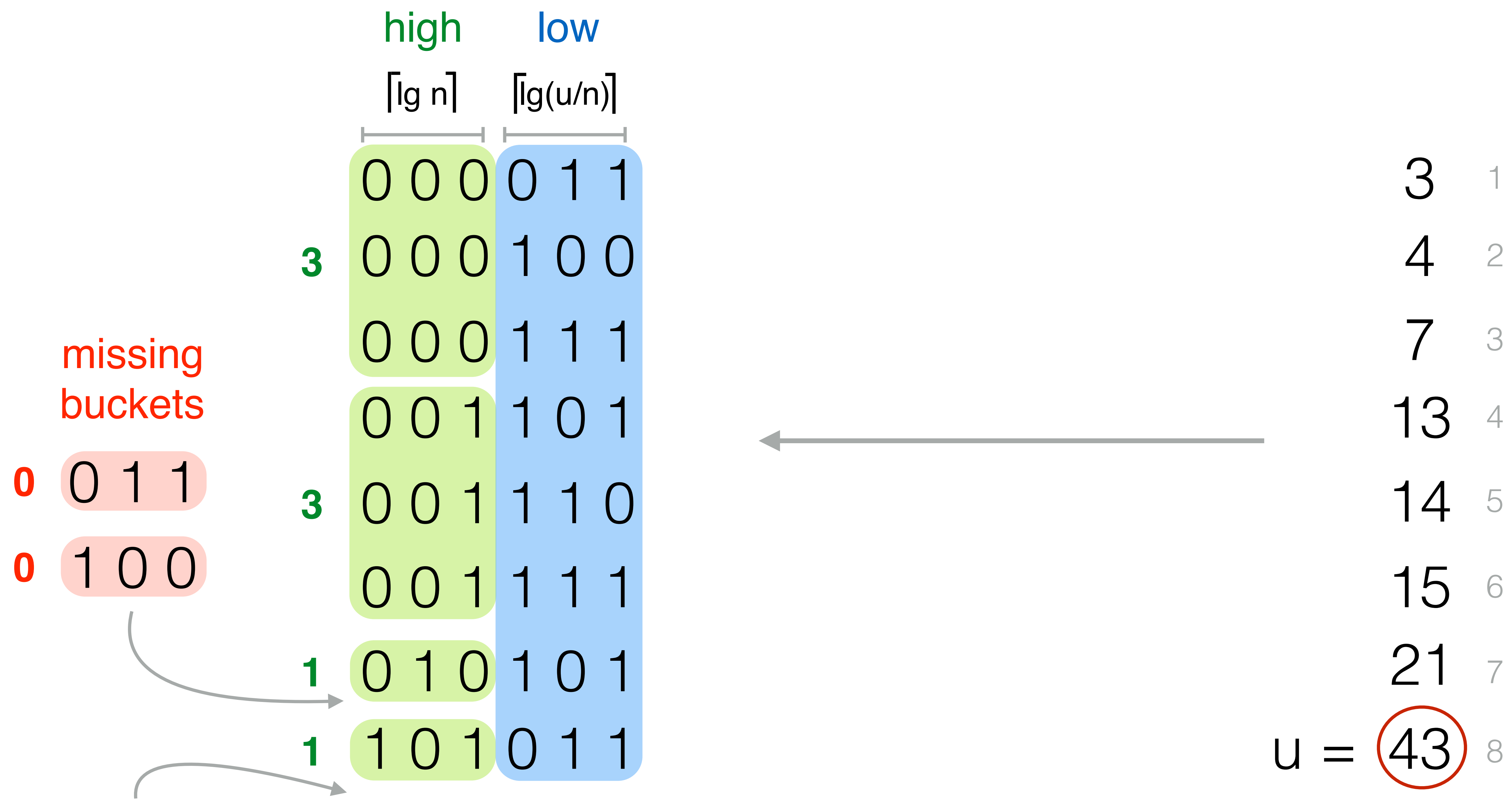
L = 011100111101110111101011

Elias-Fano solution



L = 011100111101110111101011

Elias-Fano solution



	high ⌈lg n⌉	low ⌈lg(u/n)⌉		
	0 0 0	0 1 1	3	1
3	0 0 0	1 0 0	4	2
	0 0 0	1 1 1	7	3
	0 0 1	1 0 1	13	4
3	0 0 1	1 1 0	14	5
	0 0 1	1 1 1	15	6
1	0 1 0	1 0 1	21	7
1	1 0 1	0 1 1	u = 43	8

missing buckets

0 0 1 1

0 1 0 0

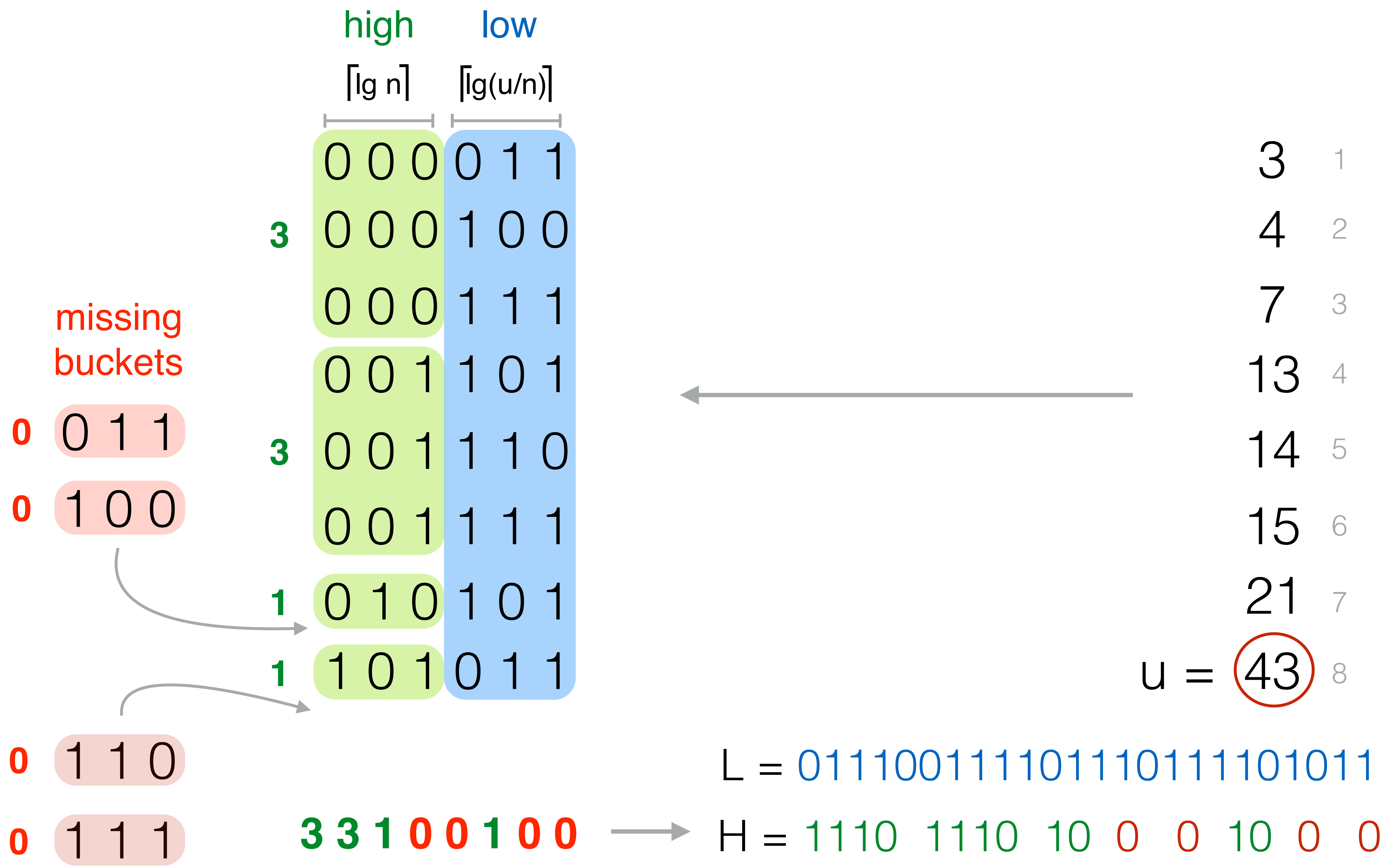
0 1 1 0

0 1 1 1

3 3 1 0 0 1 0 0

L = 011100111101110111101011

Elias-Fano solution



	high $\lceil \lg n \rceil$	low $\lceil \lg(u/n) \rceil$		
	0 0 0	0 1 1	3	1
3	0 0 0	1 0 0	4	2
	0 0 0	1 1 1	7	3
	0 0 1	1 0 1	13	4
3	0 0 1	1 1 0	14	5
	0 0 1	1 1 1	15	6
1	0 1 0	1 0 1	21	7
1	1 0 1	0 1 1	u = 43	8

missing buckets

0 0 1 1

0 1 0 0

0 1 1 0

0 1 1 1

3 3 1 0 0 1 0 0

L = 011100111101110111101011

H = 1110 1110 10 0 0 10 0 0

Properties - Space

$EF(S[0,n)) = ?$

Properties - Space

$$EF(S[0,n)) = ?$$

$\lceil \lg(u/n) \rceil$

L = 011100111101110111101011

H = 1110 1110 10 0   0 10 0   0

Properties - Space

1

$$EF(S[0,n)) = n \left\lceil \lg \frac{u}{n} \right\rceil$$

$$L = \overbrace{011}^{\lceil \lg(u/n) \rceil}100111101110111101011$$

$$H = 1110 \quad 1110 \quad 10 \quad 0 \quad 0 \quad 10 \quad 0 \quad 0$$

Properties - Space

$$\mathrm{EF}(S[0,n)) = n\left\lceil \lg \frac{u}{n} \right\rceil$$

$\lceil \lg(u/n) \rceil$

L = 011100111101110111101011

H = 1110 1110 10 0   0 10 0   0

n ones

Properties - Space

$$EF(S[0,n)) = n \left\lceil \lg \frac{u}{n} \right\rceil$$

$$\overbrace{}^{\lceil \lg(u/n) \rceil}$$

L = 011100111101110111101011

H = 1110 1110 10 0   0 10 0   0

n ones

We store a 0 whenever we change bucket.

Properties - Space

$$EF(S[0,n)) = n \left\lceil \lg \frac{u}{n} \right\rceil$$

$$L = \overbrace{011}^{\lceil \lg(u/n) \rceil}1001111011101111101011$$

$$H = 1110 \;\; 1110 \;\; 10 \;\; 0 \;\; 0 \;\; 10 \;\; 0 \;\; 0$$

We store a 0 whenever we change bucket.

n ones

$2^{\lceil \lg n \rceil}$ zeros

Properties - Space

$$EF(S[0,n)) = n \left\lceil \lg \frac{u}{n} \right\rceil + 2n \text{ bits}$$

$L = \overbrace{011}^{\lceil \lg(u/n) \rceil}100111101110111101011$

$H = 1110 \;\; 1110 \;\; 10 \;\; 0 \;\; 0 \;\; 10 \;\; 0 \;\; 0$

We store a 0 whenever we change bucket.

n ones

$2^{\lceil \lg n \rceil}$ zeros

Properties - Space

$$EF(S[0,n)) = n \left\lceil \lg \frac{u}{n} \right\rceil + 2n \ \text{ bits}$$

$\overbrace{}^{\lceil \lg(u/n) \rceil}$

L = 011100111101110111101011

H = 1110 1110 10 0 0 10 0 0

n ones

$2^{\lceil \lg n \rceil}$ zeros

Properties - Space

$$EF(S[0,n)) = n \left\lceil \lg \frac{u}{n} \right\rceil + 2n \text{ bits}$$

$\lceil \lg(u/n) \rceil$

L = 0111001111011101111101011

H = 1110 1110 10 0   0 10 0   0

n ones

Properties - Space

1

$$EF(S[0,n)) = n \left\lceil \lg \frac{u}{n} \right\rceil + 2n \text{ bits}$$

$\lceil \lg(u/n) \rceil$

L = 011100111101110111101011

H = 1110 1110 10 0   0 10 0   0

Properties - Space

1

$$\mathrm{EF}(S[0,n)) = n \left\lceil \lg \frac{u}{n} \right\rceil + 2n \quad \text{bits}$$

Properties - Space

$$\text{EF}(S[0,n)) = n \left\lceil \lg \frac{u}{n} \right\rceil + 2n \ \text{ bits}$$

Is it good or not?

Properties - Space

$$EF(S[0,n)) = n \left\lceil \lg \frac{u}{n} \right\rceil + 2n \ \text{ bits}$$

Is it good or not?

Information Theoretic Lower Bound

The minimum number of bits needed to describe a set $\mathcal{X}$ is

$$\left\lceil \lg |\mathcal{X}| \right\rceil \text{ bits.}$$

Properties - Space

$$\mathrm{EF}(S[0,n)) = n\left\lceil \lg \frac{u}{n} \right\rceil + 2n \text{ bits}$$

Is it good or not?

Information Theoretic Lower Bound

The minimum number of bits needed to describe a set $\mathcal{X}$ is

$$\left\lceil \lg |\mathcal{X}| \right\rceil \text{ bits.}$$

$\mathcal{X}$ is the set of all monotone sequence of length n drawn from a universe u.

$$|\mathcal{X}|\,?$$

Properties - Space

$$EF(S[0,n)) = n \left\lceil \lg \frac{u}{n} \right\rceil + 2n \quad \text{bits}$$

Is it good or not?

Information Theoretic Lower Bound

The minimum number of bits needed to describe a set $\mathcal{X}$ is

$$\left\lceil \lg |\mathcal{X}| \right\rceil \text{ bits.}$$

$\mathcal{X}$ is the set of all monotone sequence of length n drawn from a universe u.

$|\mathcal{X}|$?

000000000000000000

Properties - Space

$$EF(S[0,n)) = n \left\lceil \lg \frac{u}{n} \right\rceil + 2n \text{ bits}$$

Is it good or not?

Information Theoretic Lower Bound

The minimum number of bits needed to describe a set $\mathcal{X}$ is

$$\left\lceil \lg |\mathcal{X}| \right\rceil \text{ bits.}$$

$\mathcal{X}$ is the set of all monotone sequence of length n drawn from a universe u.

$|\mathcal{X}|$?

000100000000000000

3

Properties - Space

$$EF(S[0,n)) = n \left\lceil \lg \frac{u}{n} \right\rceil + 2n \text{ bits}$$

Is it good or not?

Information Theoretic Lower Bound

The minimum number of bits needed to describe a set $\mathcal{X}$ is

$$\left\lceil \lg |\mathcal{X}| \right\rceil \text{ bits.}$$

$\mathcal{X}$ is the set of all monotone sequence of length n drawn from a universe u.

$|\mathcal{X}|$?

0001001000000000000

3 6

Properties - Space

$$EF(S[0,n)) = n \left\lceil \lg \frac{u}{n} \right\rceil + 2n \ \text{ bits}$$

Is it good or not?

Information Theoretic Lower Bound

The minimum number of bits needed to describe a set $\mathcal{X}$ is

$$\left\lceil \lg |\mathcal{X}| \right\rceil \text{ bits.}$$

$\mathcal{X}$ is the set of all monotone sequence of length n drawn from a universe u.

$|\mathcal{X}|$?

000**1**00**1**000**1**0000000

3 6 10

Properties - Space

$$EF(S[0,n)) = n\left\lceil \lg \frac{u}{n} \right\rceil + 2n \ \text{ bits}$$

Is it good or not?

Information Theoretic Lower Bound

The minimum number of bits needed to describe a set $\mathcal{X}$ is

$$\left\lceil \lg |\mathcal{X}| \right\rceil \text{ bits.}$$

$\mathcal{X}$ is the set of all monotone sequence of length n drawn from a universe u.

$|\mathcal{X}|$?

000100100011000000

3 6 10 11

Properties - Space

1

$$EF(S[0,n)) = n\left\lceil \lg \frac{u}{n} \right\rceil + 2n \;\; \text{bits}$$

Is it good or not?

Information Theoretic Lower Bound

The minimum number of bits needed to describe a set $\mathcal{X}$ is

$$\left\lceil \lg |\mathcal{X}| \right\rceil \text{ bits.}$$

$\mathcal{X}$ is the set of all monotone sequence of length n drawn from a universe u.

$|\mathcal{X}|$?

000**1**00**1**000**11**00000**1**

3 6 1011 17

Properties - Space

$$EF(S[0,n)) = n \left\lceil \lg \frac{u}{n} \right\rceil + 2n \;\; \text{bits}$$

Is it good or not?

Information Theoretic Lower Bound

The minimum number of bits needed to describe a set $\mathcal{X}$ is

$\lceil \lg |\mathcal{X}| \rceil$ bits.

$\mathcal{X}$ is the set of all monotone sequence of length n drawn from a universe u.

$|\mathcal{X}|$?

000**1**00**1**0000**11**00000**1**

3 6 1011 17

With possible repetitions!
(*weak* monotonicity)

Properties - Space

$$\mathrm{EF}(S[0,n)) = n\left\lceil \lg \frac{u}{n} \right\rceil + 2n \ \text{ bits}$$

Is it good or not?

Information Theoretic Lower Bound

The minimum number of bits needed to describe a set $\mathcal{X}$ is $\left\lceil \lg |\mathcal{X}| \right\rceil$ bits.

$\mathcal{X}$ is the set of all monotone sequence of length n drawn from a universe u.

$$|\mathcal{X}| = \binom{u+n}{n}$$

000**1**00**1**000**11**00000**1**

3 6 10 11 17

With possible repetitions!
(*weak* monotonicity)

Properties - Space

1

$$\text{EF}(S[0,n)) = n\left\lceil \lg \frac{u}{n} \right\rceil + 2n \text{ bits}$$

Is it good or not?

Information Theoretic Lower Bound

The minimum number of bits needed to describe a set $\mathcal{X}$ is

$$\left\lceil \lg |\mathcal{X}| \right\rceil \text{ bits.}$$

$\mathcal{X}$ is the set of all monotone sequence of length n drawn from a universe u.

$$|\mathcal{X}| = \binom{u+n}{n}$$

$$\left\lceil \lg \binom{u+n}{n} \right\rceil \approx n \lg \frac{u+n}{n}$$

0001001000011000001

3 6 1011 17

With possible repetitions!
(*weak* monotonicity)

Properties - Space

$$\text{EF}(S[0,n)) = \boxed{n \left\lceil \lg \frac{u}{n} \right\rceil + 2n} \text{ bits}$$

Is it good or not?

Information Theoretic Lower Bound

The minimum number of bits needed to describe a set $\mathcal{X}$ is $\lceil \lg |\mathcal{X}| \rceil$ bits.

$\mathcal{X}$ is the set of all monotone sequence of length n drawn from a universe u.

$$|\mathcal{X}| = \binom{u+n}{n}$$

$$\left\lceil \lg \binom{u+n}{n} \right\rceil \approx \boxed{n \lg \frac{u+n}{n}}$$

000**1**00**1**0000**11**000000**1**

3 6 10 11 17

With possible repetitions!
(*weak* monotonicity)

Properties - Space

$$\mathrm{EF}(S[0,n)) = \boxed{n \left\lceil \lg \frac{u}{n} \right\rceil + 2n} \text{ bits}$$

Is it good or not?

(less than half a bit away [Elias-1974])

Information Theoretic Lower Bound

The minimum number of bits needed to describe a set $\mathcal{X}$ is

$$\lceil \lg |\mathcal{X}| \rceil \text{ bits.}$$

$\mathcal{X}$ is the set of all monotone sequence of length n drawn from a universe u.

$$|\mathcal{X}| = \binom{u+n}{n}$$

$$\left\lceil \lg \binom{u+n}{n} \right\rceil \approx \boxed{n \lg \frac{u+n}{n}}$$

000100100011000001

3 6 10 11 17

With possible repetitions!
(*weak* monotonicity)

Properties - Operations

Properties - Operations

`access` to each S[i] in O(1) worst-case

Properties - Operations

`access` to each S[i] in O(1) worst-case

`predecessor(x)` = $\max\{S[i] \mid S[i] < \mathtt{x}\}$

`successor(x)` = $\min\{S[i] \mid S[i] \geq \mathtt{x}\}$

queries in $O\left(\lg \frac{u}{n}\right)$ worst-case

Properties - Operations 2

access to each S[i] in O(1) worst-case

predecessor(x) = $\max\{S[i] \mid S[i] < x\}$

successor(x) = $\min\{S[i] \mid S[i] \geq x\}$

queries in $O\left(\lg \frac{u}{n}\right)$ worst-case

Properties - Operations

`access` to each S[i] in O(1) worst-case

`predecessor(x)` = $\max\{S[i] \mid S[i] < x\}$

`successor(x)` = $\min\{S[i] \mid S[i] \geq x\}$

queries in $O\left(\lg \frac{u}{n}\right)$ worst-case

but…

Properties - Operations 2

access to each S[i] in O(1) worst-case

$$\texttt{predecessor(x)} = \max\{S[i] \mid S[i] < \mathbf{x}\}$$

$$\texttt{successor(x)} = \min\{S[i] \mid S[i] \geq \mathbf{x}\}$$

queries in $O\left(\lg \frac{u}{n}\right)$ worst-case

but…

they need o(n) bits more space in order to support *fast* rank/select primitives on bitvector H

Properties - Operations

`access` to each S[i] in O(1) worst-case

`predecessor(x)` = $\max\{S[i] \mid S[i] < x\}$

`successor(x)` = $\min\{S[i] \mid S[i] \geq x\}$

queries in $O\left(\lg \frac{u}{n}\right)$ worst-case

but…

they need o(n) bits more space in order to support *fast* **`rank/select`** primitives on bitvector H

Succinct rank/select

Definition

Given a bitvector B of n bits:

$\texttt{rank}_{0/1}(\texttt{i})$ = # of 0/1 in [0,$\texttt{i}$)

$\texttt{select}_{0/1}(\texttt{i})$ = position of $\texttt{i}$-th 0/1

Succinct `rank/select`

Definition

Given a bitvector B of n bits:

`rank`$_{0/1}$`(i)` = # of 0/1 in [0,`i`)

`select`$_{0/1}$`(i)` = position of `i`-th 0/1

Examples

B = 101011010101111010110101

Succinct `rank/select`

Definition

Given a bitvector B of n bits:

`rank`$_{0/1}$`(i)` = # of 0/1 in [0,`i`)

`select`$_{0/1}$`(i)` = position of `i`-th 0/1

Examples

B = 101011010101111010110101

`rank`$_{0}$`(5) = 2`

Succinct rank/select

Definition

Given a bitvector B of n bits:

$\mathtt{rank}_{0/1}(\mathtt{i})$ = # of 0/1 in $[0,\mathtt{i})$

$\mathtt{select}_{0/1}(\mathtt{i})$ = position of $\mathtt{i}$-th 0/1

Examples

B = 101011010101111010110101

$\mathtt{rank}_0(5) = 2$

$\mathtt{rank}_1(7) = 4$

Succinct $\texttt{rank/select}$

Definition

Given a bitvector B of n bits:

$\texttt{rank}_{0/1}(\texttt{i})$ = # of 0/1 in [0,$\texttt{i}$)

$\texttt{select}_{0/1}(\texttt{i})$ = position of $\texttt{i}$-th 0/1

Examples

B = 101011010101111010110101

$\texttt{rank}_{0}(5) = 2$ $\texttt{select}_{0}(5) = 10$

$\texttt{rank}_{1}(7) = 4$

Succinct rank/select

Definition

Given a bitvector B of n bits:

$\texttt{rank}_{0/1}(\texttt{i})$ = # of 0/1 in [0,$\texttt{i}$)

$\texttt{select}_{0/1}(\texttt{i})$ = position of $\texttt{i}$-th 0/1

Examples

B = 101011010101111010110101

$\texttt{rank}_{0}(5) = 2$ $\texttt{select}_{0}(5) = 10$

$\texttt{rank}_{1}(7) = 4$ $\texttt{select}_{1}(7) = 11$

Succinct rank/select

Definition

Given a bitvector B of n bits:

$\mathrm{rank}_{0/1}(i)$ = # of 0/1 in [0,i)

$\mathrm{select}_{0/1}(i)$ = position of i-th 0/1

Examples

B = 1010110101011110101101010

$\mathrm{rank}_{0}(5) = 2$ $\mathrm{select}_{0}(5) = 10$

$\mathrm{rank}_{1}(7) = 4$ $\mathrm{select}_{1}(7) = 11$

Relations

$$\mathrm{rank}_{1/0}(\mathrm{select}_{0/1}(i)) = \mathrm{select}_{0/1}(i) - i$$

$$\mathrm{rank}_{0/1}(\mathrm{select}_{0/1}(i)) = i\text{-}1$$

$$\mathrm{rank}_{0/1}(i) + \mathrm{rank}_{1/0}(i) = i$$

Succinct rank/select

O(1)-solutions with o(n) bits

rank	(multi)-layered index + precomputed table	[Jacobson-1989]
select	three-level directory tree	[Clark-1996]

Succinct rank/select

O(1)-solutions with o(n) bits

rank (multi)-layered index + precomputed table [Jacobson-1989]

2^{30} bits → ~**67**% more bits!

select three-level directory tree [Clark-1996]

Succinct rank/select 2

O(1)-solutions with o(n) bits

rank (multi)-layered index + precomputed table [Jacobson-1989]

2^{30} bits → ~**67**% more bits!

select three-level directory tree [Clark-1996]

2^{30} bits → ~**60**% more bits!

Succinct rank/select 2

O(1)-solutions with o(n) bits

rank (multi)-layered index + precomputed table [Jacobson-1989]

2^{30} bits → ~**67**% more bits!

select three-level directory tree [Clark-1996]

2^{30} bits → ~**60**% more bits!

Nowadays *practical* solutions are based on [Vigna-2008, Zhou *et al.*-2013]:

- broadword programming
- interleaving
- Intel hardware **popcnt** instruction:

`Long().bitCount(x)` in Java

`__builtin_popcountl(x)` in C/C++

Succinct `rank/select`

O(1)-solutions with o(n) bits

`rank` (multi)-layered index + precomputed table [Jacobson-1989]

2^{30} bits → ~**67**% more bits!

`select` three-level directory tree [Clark-1996]

2^{30} bits → ~**60**% more bits!

Nowadays *practical* solutions are based on [Vigna-2008, Zhou *et al.*-2013]:

- broadword programming
- interleaving
- Intel hardware **`popcnt`** instruction:

`Long().bitCount(x)` in Java

`__builtin_popcountl(x)` in C/C++

`rank` → ~**3**% more bits

`select` → ~**0.39**% more bits

with practical constant-time `selection`

access example

S = [3, 4, 7, 13, 14, 15, 21, 43]

1 2 3 4 5 6 7 8

access example

S = [3, 4, 7, 13, 14, 15, 21, 43]

1 2 3 4 5 6 7 8

access(4) = S[4] = ?

access example

S = [3, 4, 7, 13, 14, 15, 21, 43]

1 2 3 4 5 6 7 8

access(4) = S[4] = ?

H = 1110111010001000

L = $\underbrace{011}_{k = \lceil \lg(u/n) \rceil}$100111101110111101011

access example

S = [3, 4, 7, 13, 14, 15, 21, 43]

1 2 3 4 5 6 7 8

access(4) = S[4] = ?

Recall: we store a 0 whenever we change bucket.

H = 1110111010001000

$$L = \underbrace{011}_{k = \lceil \lg(u/n) \rceil}100111101110111101011$$

access example

S = [3, 4, 7, 13, 14, 15, 21, 43]

1 2 3 4 5 6 7 8

access(4) = S[4] = ?

Recall: we store a 0 whenever we change bucket.

H = 1110111010001000

L = 011100111101110111101011

$k = \lceil \lg(u/n) \rceil$

access example

S = [3, 4, 7, 13, 14, 15, 21, 43]

1 2 3 4 5 6 7 8

access(4) = S[4] = ?

Recall: we store a 0 whenever we change bucket.

H = 1110111010001000

L = 011100111101110111101011

$k = \lceil \lg(u/n) \rceil$

access(i) = $\text{select}_1(i)$

access example

S = [3, 4, 7, 13, 14, 15, 21, 43]

1 2 3 4 5 6 7 8

access(4) = S[4] = ?

Recall: we store a 0 whenever we change bucket.

H = 1110111010001000

$$L = \underbrace{011}_{k = \lceil \lg(u/n) \rceil}100111101110111101011$$

$\text{access}(i) = \text{rank}_0(\text{select}_1(i))$

access example

S = [3, 4, 7, 13, 14, 15, 21, 43]

1 2 3 4 5 6 7 8

access(4) = S[4] = 001000

Recall: we store a 0 whenever we change bucket.

H = 1110111010001000

L = 011100111101110111101011

$k = \lceil \lg(u/n) \rceil$

$\mathrm{access}(i) = \mathrm{rank}_0(\mathrm{select}_1(i))$

access example

S = [3, 4, 7, 13, 14, 15, 21, 43]

1 2 3 4 5 6 7 8

access(4) = S[4] = 001000

Recall: we store a 0 whenever we change bucket.

H = 11101110100001000

L = 011100111101110111101011

$k = \lceil \lg(u/n) \rceil$

$$\text{access}(i) = \text{rank}_0(\text{select}_1(i)) = \text{select}_1(i) - i$$

`access` example

S = [3, 4, 7, 13, 14, 15, 21, 43]

1 2 3 4 5 6 7 8

`access(4)` = S[4] = 001000

Recall: we store a 0 whenever we change bucket.

H = 1110111010001000

L = $\underbrace{011}_{k = \lceil \lg(u/n) \rceil}$100111101110111101011

`access(i) = select`$_1$`(i) - i`

access example

S = [3, 4, 7, 13, 14, 15, 21, 43]

1 2 3 4 5 6 7 8

access(4) = S[4] = 001 101

Recall: we store a 0 whenever we change bucket.

H = 1110111010001000

L = 011100111101110111101011

$k = \lceil \lg(u/n) \rceil$

$$\mathtt{access}(i) = \mathtt{select}_1(i) - i \ll k \mid L[(i-1)k, ik)$$

access example

S = [3, 4, 7, 13, 14, 15, 21, 43]

1 2 3 4 5 6 7 8

access(4) = S[4] = 001 101

Recall: we store a 0 whenever we change bucket.

H = 1110111010001000

$$L = \underbrace{011}_{k = \lceil \lg(u/n) \rceil}100111101110111101011$$

$$\mathtt{access}(i) = \mathtt{select}_1(i) - i \ll k \mid L[(i-1)k, ik)$$

access example

S = [3, 4, 7, 13, 14, 15, 21, 43]

1 2 3 4 5 6 7 8

access(4) = S[4] = 001 101

access(7) = S[7] = ?

Recall: we store a 0 whenever we change bucket.

H = 1110111010001000

L = $\underbrace{011}_{k = \lceil \lg(u/n) \rceil}$100111101110111101011

$\mathtt{access(i)} = \mathtt{select}_1\mathtt{(i)} - \mathtt{i} \ll \mathtt{k} \mid \mathtt{L[(i-1)k,ik)}$

access example

S = [3, 4, 7, 13, 14, 15, 21, 43]

1 2 3 4 5 6 7 8

access(4) = S[4] = 001 101

access(7) = S[7] = ?

Recall: we store a 0 whenever we change bucket.

H = 11101110100001000

L = 011100111101110111101011

$k = \lceil \lg(u/n) \rceil$

$\mathtt{access}(i) = \mathtt{select}_1(i) - i \ll k \mid L[(i-1)k, ik)$

access example

S = [3, 4, 7, 13, 14, 15, 21, 43]

1 2 3 4 5 6 7 8

access(4) = S[4] = 001 101

access(7) = S[7] = 010000

Recall: we store a 0 whenever we change bucket.

H = 1110111010001000

L = 011100111101110111101011

$k = \lceil \lg(u/n) \rceil$

access(i) = select_1(i) - i << k | L[(i-1)k,ik)

`access` example

S = [3, 4, 7, 13, 14, 15, 21, 43]

1 2 3 4 5 6 7 8

`access(4)` = S[4] = 001 101

`access(7)` = S[7] = 010 101

Recall: we store a 0 whenever we change bucket.

H = 1110111010001000

L = 011100111101110111101011

k = $\lceil \lg(u/n) \rceil$

```
access(i) = select₁(i) - i << k | L[(i-1)k,ik)
```

access example

S = [3, 4, 7, 13, 14, 15, 21, 43]

1 2 3 4 5 6 7 8

access(4) = S[4] = 001 101

access(7) = S[7] = 010 101

Recall: we store a 0 whenever we change bucket.

H = 1110111010001000

L = 011100111101110111101011

k = ⌈lg(u/n)⌉

Complexity: O(1)

access(i) = select₁(i) - i << k | L[(i-1)k,ik)

successor **example**

S = [3, 4, 7, 13, 14, 15, 21, 43]

1 2 3 4 5 6 7 8

H = 1110111010001000

L = 011100111101110111101011

successor **example**

S = [3, 4, 7, 13, 14, 15, 21, 43]

1 2 3 4 5 6 7 8

successor(12) = ?

H = 1110111010001000

L = 011100111101110111101011

`successor` example

S = [3, 4, 7, 13, 14, 15, 21, 43]

1 2 3 4 5 6 7 8

`successor(12)` = ?

001100

H = 1110111010001000

L = 011100111101110111101011

successor **example**

S = [3, 4, 7, 13, 14, 15, 21, 43]

1 2 3 4 5 6 7 8

successor(12) = ?

$h_{12} = \boxed{001}100$

H = 1110111010001000

L = 011100111101110111101011

successor example

S = [3, 4, 7, 13, 14, 15, 21, 43]

1 2 3 4 5 6 7 8

successor(12) = ?

$h_{12} = \boxed{001}100$

$p_1 = \text{select}_0(h_x) - h_x$

$p_2 = \text{select}_0(h_x+1) - h_x - 1$

H = 1110111010001000

L = 011100111101110111101011

successor example

S = [3, 4, 7, 13, 14, 15, 21, 43]

1 2 3 4 5 6 7 8

successor(12) = ?

$h_{12} = \boxed{001}100$

$p_1 = \mathrm{select}_0(h_x) - h_x$

$p_2 = \mathrm{select}_0(h_x+1) - h_x - 1$

H = ~~1110~~111010001000

L = 011100111101110111101011

successor **example**

S = [3, 4, 7, 13, 14, 15, 21, 43]

1 2 3 4 5 6 7 8

successor(12) = ?

$h_{12} = \boxed{001}100$

$p_1 = \mathtt{select}_0(h_x) - h_x$

$p_2 = \mathtt{select}_0(h_x+1) - h_x - 1$

H = ~~1110~~1110~~10001000~~

L = 011100111101110111101011

successor **example**

S = [3, 4, 7, 13, 14, 15, 21, 43]

1 2 3 4 5 6 7 8

successor(12) = ?

h_{12} = $\boxed{001}$100

p_1 = select$_0$(h_x)-h_x

p_2 = select$_0$(h_x+1)-h_x-1

H = ~~1110~~1110~~10001000~~

L = ~~011100111~~101110111~~101011~~

successor **example**

S = [3, 4, 7, 13, 14, 15, 21, 43]

1 2 3 4 5 6 7 8

successor(12) = ?

$h_{12} = \boxed{001}100$

p_1 = select$_0$(h_x)-h_x

p_2 = select$_0$(h_x+1)-h_x-1

H = ~~1110~~1110~~10001000~~

L = ~~011100111~~101110111~~101011~~

p_1 p_2

→

binary search
in [p_1,p_2)

successor **example**

$S = [3, 4, 7, 13, 14, 15, 21, 43]$

1 2 3 4 5 6 7 8

$\texttt{successor}(12) = 13$

$h_{12} = \boxed{001}100$

$p_1 = \texttt{select}_0(h_x) - h_x$

$p_2 = \texttt{select}_0(h_x + 1) - h_x - 1$

H = ~~1110~~1110~~10001000~~

L = ~~011100111~~101110111~~101011~~

p_1 p_2

→ *binary search* in $[p_1, p_2)$

successor **example**

S = [3, 4, 7, 13, 14, 15, 21, 43]

1 2 3 4 5 6 7 8

successor(12) = 13

$h_{12} = \boxed{001}100$

$p_1 = \mathrm{select}_0(h_x) - h_x$

$p_2 = \mathrm{select}_0(h_x+1) - h_x - 1$

H = ~~1110~~1110~~10001000~~

L = ~~011100111~~101110111~~101011~~

p_1 p_2

→

binary search
in $[p_1, p_2)$

Complexity: $O\left(\lg \frac{u}{n}\right)$

Performance

4 Intel i7-4790K cores (8 threads) clocked at 4Ghz, with 32 GB RAM, running Linux 4.2.0, 64 bits

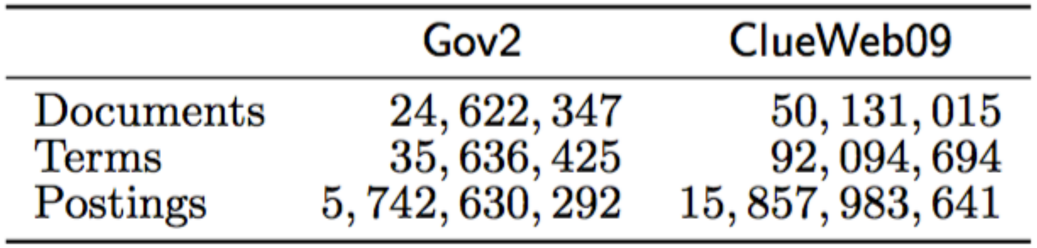
C++11, compiled with `gcc` 5.3.0 with the highest optimisation setting

n	u	`access`	`successor`	iterated `successor`	iterator
~2.4x10^{6}	~1.76x10^{9}	27.6 ns	0.24 μs	7.61 ns	2.34 ns
~10.5x10^{6}	~7.83x10^{9}	41.4 ns	0.29 μs	7.61 ns	2.36 ns

n	uncompressed sequence bytes	Elias-Fano bytes	compression ratio
~2.4x10^{6}	18,787,288	3,530,704	532%
~10.5x10^{6}	83,565,504	15,704,680	532%

Performance

Datasets



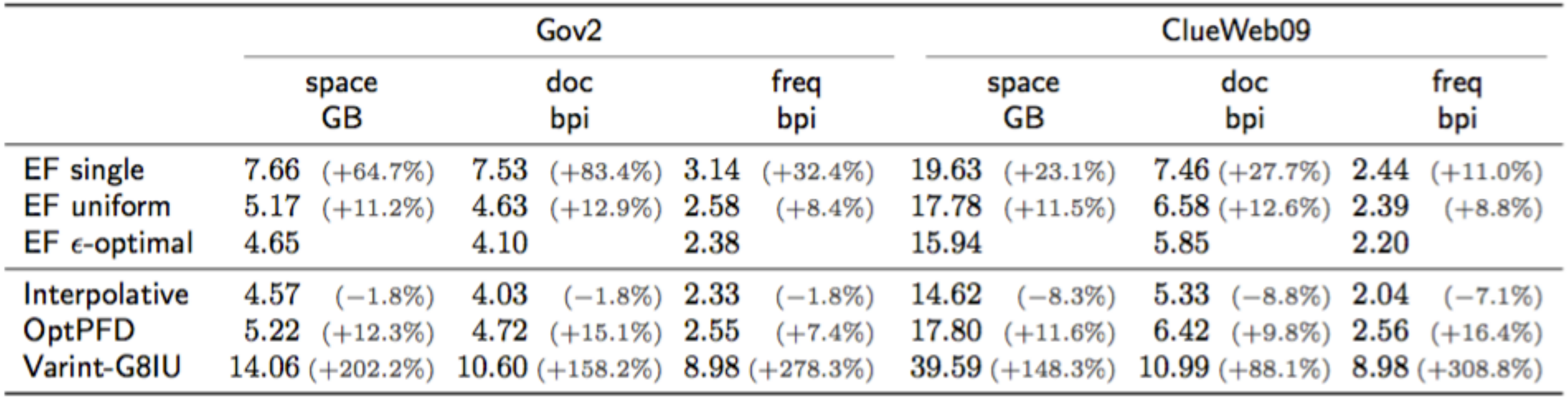
	Gov2	ClueWeb09
Documents	24, 622, 347	50, 131, 015
Terms	35, 636, 425	92, 094, 694
Postings	5, 742, 630, 292	15, 857, 983, 641

24 Intel Xeon E5-2697 Ivy Bridge cores (48 threads) clocked at 2.70Ghz, with 64 GB RAM, running Linux 3.12.7, 64 bits

C++11, compiled with `gcc` 4.9 with the highest optimisation setting

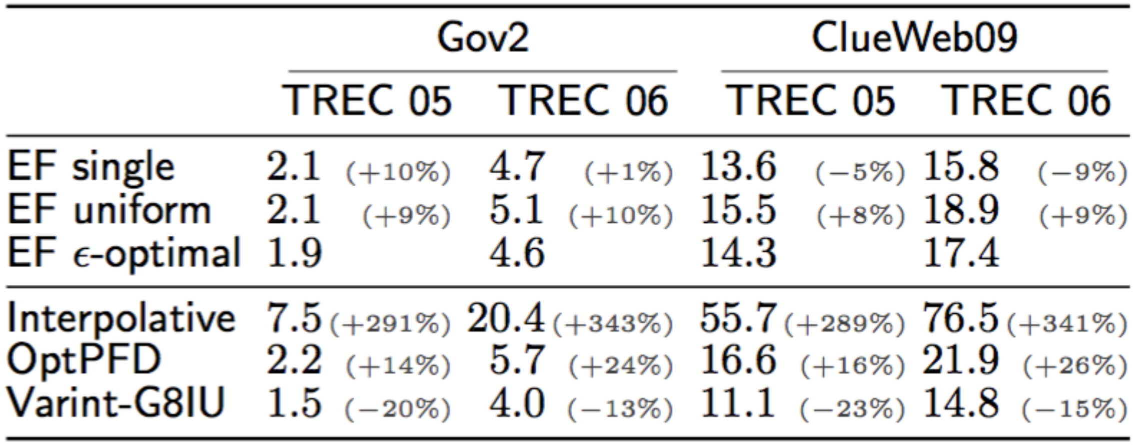
Numbers from [Ottaviano and Venturini-2014].

Space



	Gov2			ClueWeb09		
	space GB	doc bpi	freq bpi	space GB	doc bpi	freq bpi
EF single	7.66 (+64.7%)	7.53 (+83.4%)	3.14 (+32.4%)	19.63 (+23.1%)	7.46 (+27.7%)	2.44 (+11.0%)
EF uniform	5.17 (+11.2%)	4.63 (+12.9%)	2.58 (+8.4%)	17.78 (+11.5%)	6.58 (+12.6%)	2.39 (+8.8%)
EF ϵ-optimal	4.65	4.10	2.38	15.94	5.85	2.20
Interpolative	4.57 (−1.8%)	4.03 (−1.8%)	2.33 (−1.8%)	14.62 (−8.3%)	5.33 (−8.8%)	2.04 (−7.1%)
OptPFD	5.22 (+12.3%)	4.72 (+15.1%)	2.55 (+7.4%)	17.80 (+11.6%)	6.42 (+9.8%)	2.56 (+16.4%)
Varint-G8IU	14.06 (+202.2%)	10.60 (+158.2%)	8.98 (+278.3%)	39.59 (+148.3%)	10.99 (+88.1%)	8.98 (+308.8%)

AND queries (timings are in milliseconds)



	Gov2		ClueWeb09	
	TREC 05	TREC 06	TREC 05	TREC 06
EF single	2.1 (+10%)	4.7 (+1%)	13.6 (−5%)	15.8 (−9%)
EF uniform	2.1 (+9%)	5.1 (+10%)	15.5 (+8%)	18.9 (+9%)
EF ϵ-optimal	1.9	4.6	14.3	17.4
Interpolative	7.5 (+291%)	20.4 (+343%)	55.7 (+289%)	76.5 (+341%)
OptPFD	2.2 (+14%)	5.7 (+24%)	16.6 (+16%)	21.9 (+26%)
Varint-G8IU	1.5 (−20%)	4.0 (−13%)	11.1 (−23%)	14.8 (−15%)

Killer applications

1. Inverted Indexes

Sebastiano Vigna. *Quasi-succinct indices*. In Proceedings of the 6-th ACM International Conference on Web Search and Data Mining (WSDM), 83-92 (2013).

Giuseppe Ottaviano, Rossano Venturini. *Partitioned Elias-Fano Indexes*. In Proceedings of the 37-th ACM International Conference on Research and Development in Information Retrieval (SIGIR), 273-282 (2014).

Killer applications

1. Inverted Indexes

Sebastiano Vigna. *Quasi-succinct indices*. In Proceedings of the 6-th ACM International Conference on Web Search and Data Mining (WSDM), 83-92 (2013).

Giuseppe Ottaviano, Rossano Venturini. *Partitioned Elias-Fano Indexes*. In Proceedings of the 37-th ACM International Conference on Research and Development in Information Retrieval (SIGIR), 273-282 (2014).

2. Social Networks

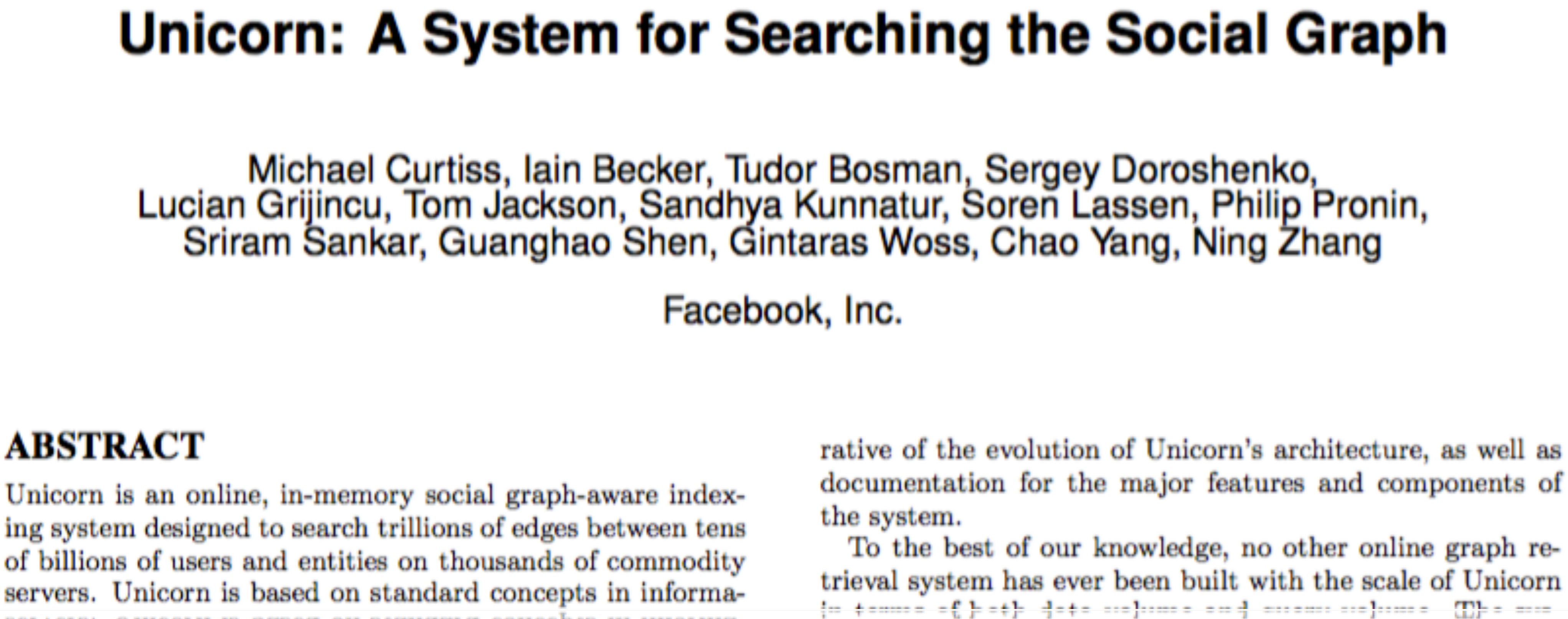
Killer applications

1. Inverted Indexes

Sebastiano Vigna. *Quasi-succinct indices*. In Proceedings of the 6-th ACM International Conference on Web Search and Data Mining (WSDM), 83-92 (2013).

Giuseppe Ottaviano, Rossano Venturini. *Partitioned Elias-Fano Indexes*. In Proceedings of the 37-th ACM International Conference on Research and Development in Information Retrieval (SIGIR), 273-282 (2014).

2. Social Networks



Unicorn: A System for Searching the Social Graph

Michael Curtiss, Iain Becker, Tudor Bosman, Sergey Doroshenko,
Lucian Grijincu, Tom Jackson, Sandhya Kunnatur, Soren Lassen, Philip Pronin,
Sriram Sankar, Guanghao Shen, Gintaras Woss, Chao Yang, Ning Zhang

Facebook, Inc.

ABSTRACT

Unicorn is an online, in-memory social graph-aware indexing system designed to search trillions of edges between tens of billions of users and entities on thousands of commodity servers. Unicorn is based on standard concepts in informa-

rative of the evolution of Unicorn's architecture, as well as documentation for the major features and components of the system.

To the best of our knowledge, no other online graph retrieval system has ever been built with the scale of Unicorn

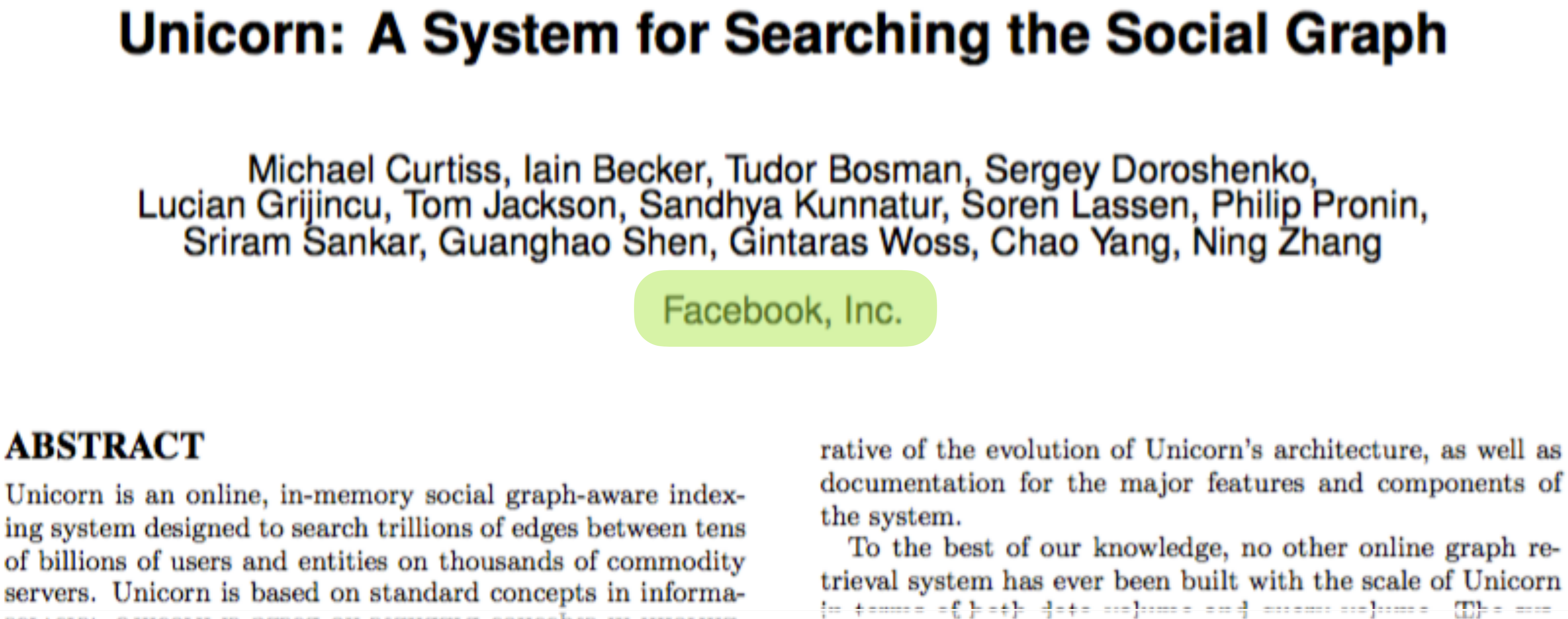
Killer applications

1. Inverted Indexes

Sebastiano Vigna. *Quasi-succinct indices*. In Proceedings of the 6-th ACM International Conference on Web Search and Data Mining (WSDM), 83-92 (2013).

Giuseppe Ottaviano, Rossano Venturini. *Partitioned Elias-Fano Indexes*. In Proceedings of the 37-th ACM International Conference on Research and Development in Information Retrieval (SIGIR), 273-282 (2014).

2. Social Networks



Unicorn: A System for Searching the Social Graph

Michael Curtiss, Iain Becker, Tudor Bosman, Sergey Doroshenko,
Lucian Grijincu, Tom Jackson, Sandhya Kunnatur, Soren Lassen, Philip Pronin,
Sriram Sankar, Guanghao Shen, Gintaras Woss, Chao Yang, Ning Zhang

Facebook, Inc.

ABSTRACT

Unicorn is an online, in-memory social graph-aware indexing system designed to search trillions of edges between tens of billions of users and entities on thousands of commodity servers. Unicorn is based on standard concepts in informa-

rative of the evolution of Unicorn's architecture, as well as documentation for the major features and components of the system.

To the best of our knowledge, no other online graph retrieval system has ever been built with the scale of Unicorn

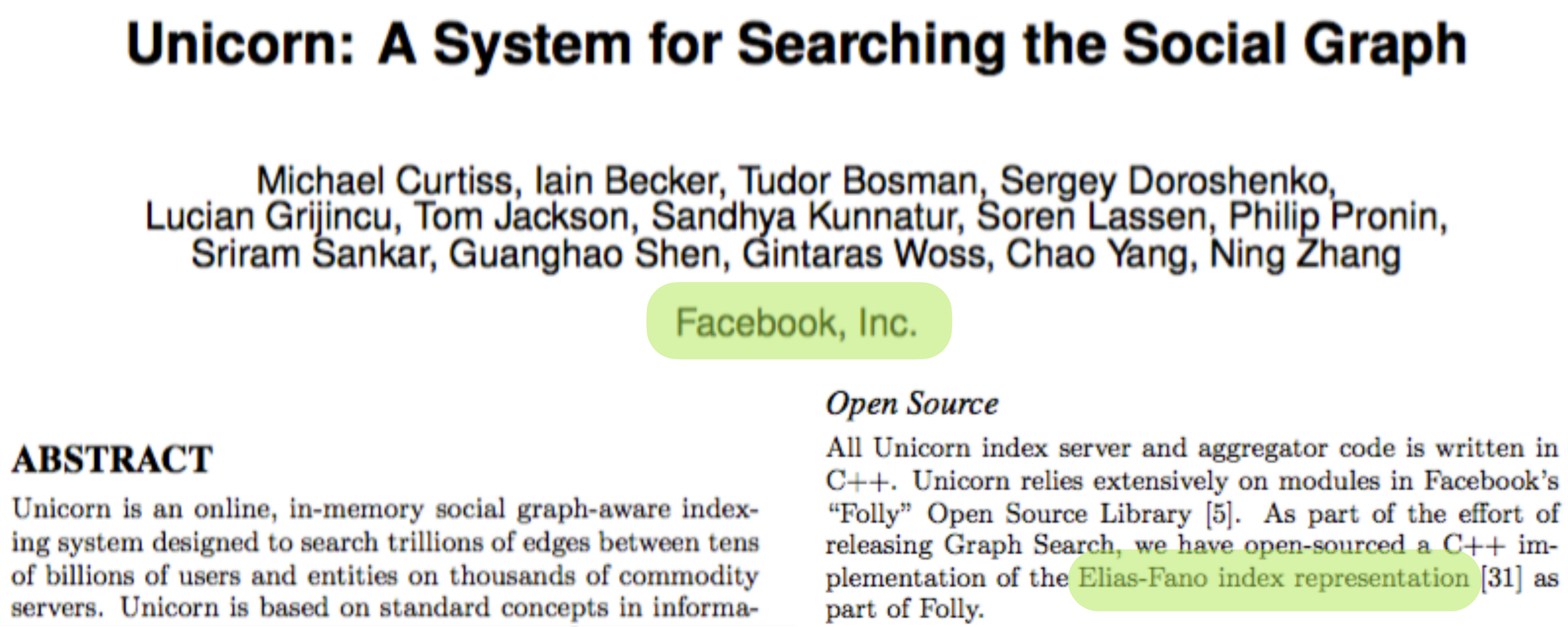
Killer applications

1. Inverted Indexes

Sebastiano Vigna. *Quasi-succinct indices*. In Proceedings of the 6-th ACM International Conference on Web Search and Data Mining (WSDM), 83-92 (2013).

Giuseppe Ottaviano, Rossano Venturini. *Partitioned Elias-Fano Indexes*. In Proceedings of the 37-th ACM International Conference on Research and Development in Information Retrieval (SIGIR), 273-282 (2014).

2. Social Networks



Unicorn: A System for Searching the Social Graph

Michael Curtiss, Iain Becker, Tudor Bosman, Sergey Doroshenko,
Lucian Grijincu, Tom Jackson, Sandhya Kunnatur, Soren Lassen, Philip Pronin,
Sriram Sankar, Guanghao Shen, Gintaras Woss, Chao Yang, Ning Zhang

Facebook, Inc.

ABSTRACT

Unicorn is an online, in-memory social graph-aware indexing system designed to search trillions of edges between tens of billions of users and entities on thousands of commodity servers. Unicorn is based on standard concepts in informa-

Open Source

All Unicorn index server and aggregator code is written in C++. Unicorn relies extensively on modules in Facebook's "Folly" Open Source Library [5]. As part of the effort of releasing Graph Search, we have open-sourced a C++ implementation of the Elias-Fano index representation [31] as part of Folly.

Available Implementations

Library	Author(s)	Link	Language
folly	Facebook, Inc.	https://github.com/facebook/folly	C++
sdsl	Simon Gog	https://github.com/simongog/sdsl-lite	C++
ds2i	Giuseppe Ottaviano Rossano Venturini Nicola Tonellotto	https://github.com/ot/ds2i	C++
Sux	Sebastiano Vigna	http://sux.di.unimi.it	Java/C++

Summary

Elias-Fano encodes *monotone integer sequences* in *space close to the information theoretic minimum*, while allowing *powerful search operations*, namely `predecessor/successor` queries and random `access`.

Successfully applied to crucial problems, such as *inverted indexes* and *social graphs* representation.

Several *optimized* software implementations are available.

References

[Fano-1971] Robert Mario Fano. *On the number of bits required to implement an associative memory*. Memorandum 61, Computer Structures Group, MIT (1971).

[Elias-1974] Peter Elias. *Efficient Storage and Retrieval by Content and Address of Static Files*. Journal of the ACM (JACM) 21, 2, 246–260 (1974).

[Jacobson-1989] Guy Jacobson. *Succinct Static Data Structures*. Ph.D. Thesis, Carnegie Mellon University (1989).

[Clark-1996] David Clark. *Compact Pat Trees*. Ph.D. Thesis, University of Waterloo (1996).

[Moffat and Stuiver-2000] Alistair Moffat and Lang Stuiver. *Binary Interpolative Coding for Effective Index Compression.* Information Retrieval Journal 3, 1, 25–47 (2000).

[Anh and Moffat-2005] Vo Ngoc Anh and Alistair Moffat. *Inverted Index Compression Using Word-Aligned Binary Codes.* Information Retrieval Journal 8, 1, 151–166 (2005).

[Salomon-2007] David Salomon. *Variable-length Codes for Data Compression.* Springer (2007).

[Vigna-2008] Sebastiano Vigna. *Broadword implementation of rank/select queries*. In Workshop in Experimental Algorithms (WEA), 154-168 (2008).

References

2

[Yan *et al.*-2009] Hao Yan, Shuai Ding, and Torsten Suel. *Inverted index compression and query processing with optimized document ordering.* In Proceedings of the 18th International Conference on World Wide Web (WWW). 401–410 (2009).

[Anh and Moffat-2010] Vo Ngoc Anh and Alistair Moffat. *Index compression using 64-bit words.* In Software: Practice and Experience 40, 2, 131–147 (2010).

[Zukowski *et al.*-2010] Marcin Zukowski, Sandor Hèman, Niels Nes, and Peter Boncz. *Super-Scalar RAM-CPU Cache Compression.* In Proceedings of the 22nd International Conference on Data Engineering (ICDE). 59–70 (2006).

[Stepanov *et al.*-2011] Alexander Stepanov, Anil Gangolli, Daniel Rose, Ryan Ernst, and Paramjit Oberoi. *SIMD-based decoding of posting lists.* In Proceedings of the 20th ACM International Conference on Information and Knowledge Management (CIKM). 317–326 (2011).

[Zhou *et al.*-2013] Dong Zhou, David Andersen, Michael Kaminsky. *Space-Efficient, High-Performance Rank and Select Structures on Uncompressed Bit Sequences*. In Proceedings of the 12-nd International Symposium on Experimental Algorithms (SEA), 151-163 (2013).

[Vigna-2013] Sebastiano Vigna. *Quasi-succinct indices*. In Proceedings of the 6-th ACM International Conference on Web Search and Data Mining (WSDM), 83-92 (2013).

[Curtiss *et al.*-2013] Michael Curtiss *et al. Unicorn: A System for Searching the Social Graph*. In Proceedings of the Very Large Database Endowment (PVLDB), 1150-1161 (2013).

[Ottaviano and Venturini-2014] Giuseppe Ottaviano, Rossano Venturini. *Partitioned Elias-Fano Indexes*. In Proceedings of the 37-th ACM International Conference on Research and Development in Information Retrieval (SIGIR), 273-282 (2014).

Thanks for your attention, time, patience!

Any questions?