Fast and Compact Set Intersection through Recursive Universe Partitioning

Giulio Ermanno Pibiri

giulio.ermanno.pibiri@isti.cnr.it
http://pages.di.unipi.it/pibiri

ISTITUTO DI SCIENZA E TECNOLOGIE
DELL'INFORMAZIONE "A. FAEDO"

IEEE Data Compression Conference (DCC)

March 2021

The Compressed Set Intersection Problem

Design a **compressed** representation for a sorted integer sequence S[0..n) whose values are drawn from a universe of size u, so that **intersecting** two such sequences is done efficiently.

Other queries of interest:
- Union
- random Access
- Contains
- Predecessor/Successor

Applications

Inverted indexes

Databases

E-Commerce

Graph compression

Semantic data

Geospatial data

Compressed Representations

Large research corpora describing **different space/time trade-offs**.

- Golomb
- Elias' Gamma and Delta
- Elias-Fano
- Variable-Byte
- Binary Interpolative Coding
- Simple
- PForDelta
- QMX
- Quasi-Succinct
- Partitioned Elias-Fano
- SIMD-BP
- Clustered Elias-Fano
- Optimal Variable-Byte
- ANS-based
- DINT

~1960

↓

present day

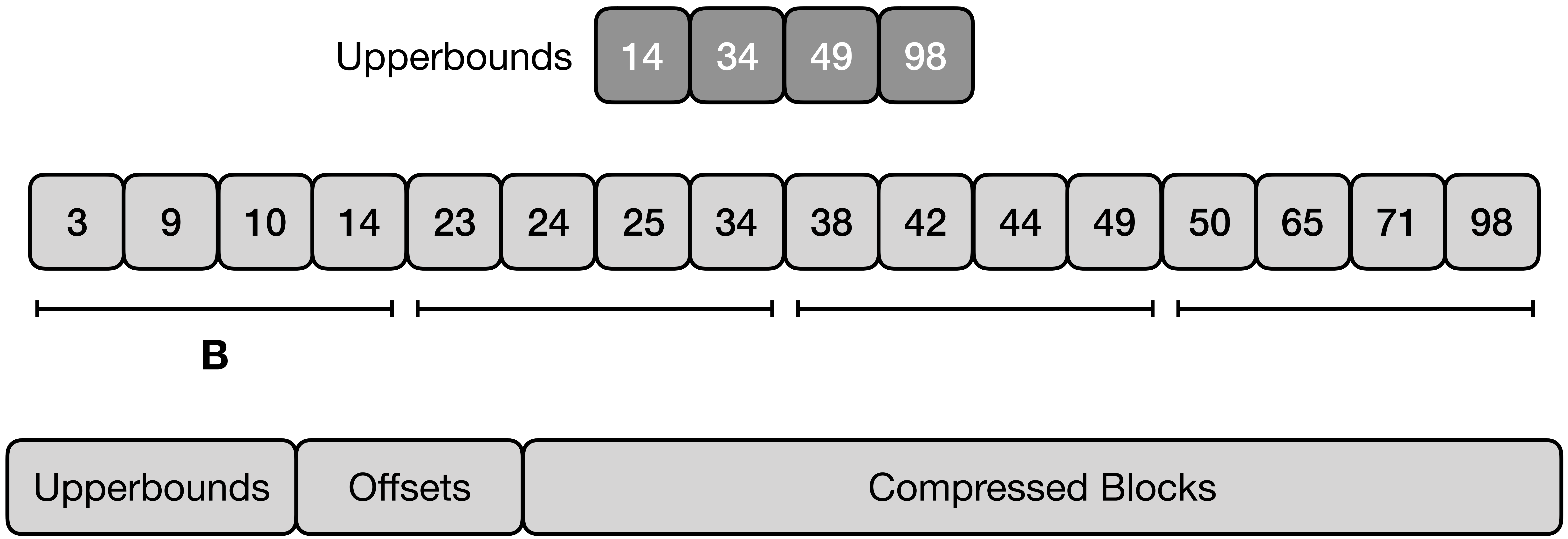
See a recent survey paper "Techniques for Inverted Index Compression", ACM CSUR 2020, by G. E. P. and Rossano Venturini

Partitioning by Cardinality

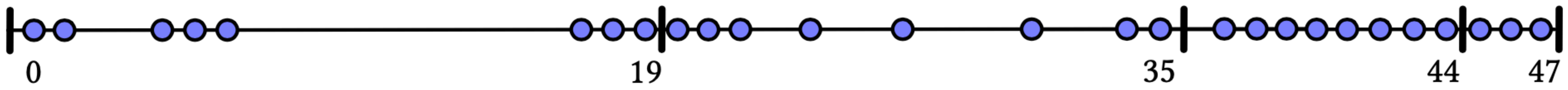
The problem is that the operations of interest
are not natively supported:
we can just **decode sequentially**.

Partitioning by Cardinality

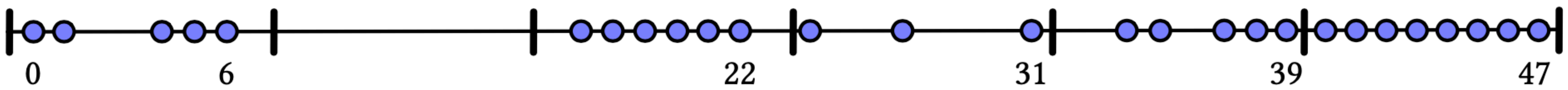
The problem is that the operations of interest
are not natively supported:
we can just **decode sequentially**.



Partitioning by Universe

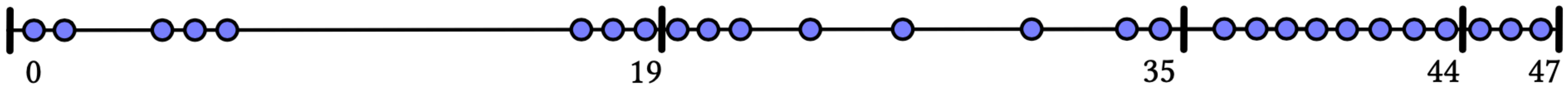


(a) cardinality partitioning – CP

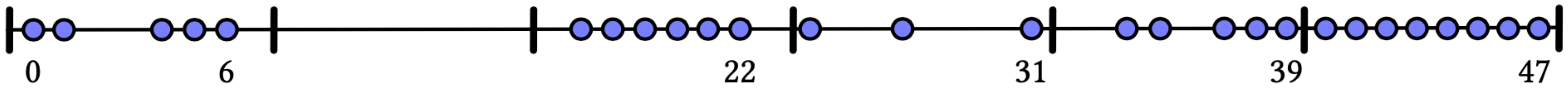


(b) universe partitioning – UP

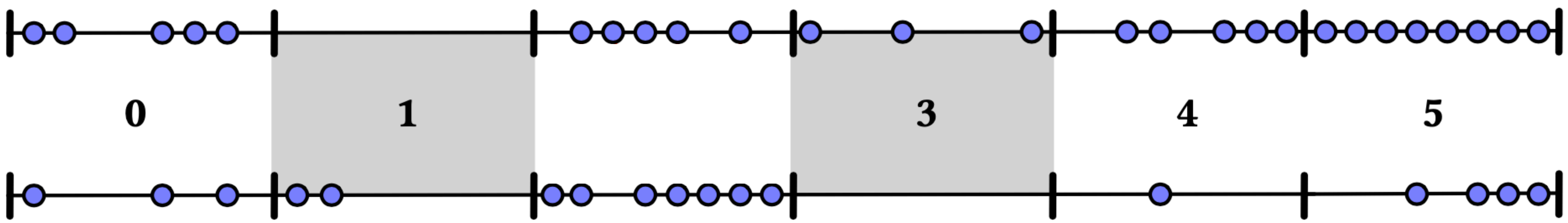
Partitioning by Universe



(a) cardinality partitioning – CP



(b) universe partitioning – UP



Intersection(lists):

Intersect only the non-empty slices in common between the lists.

Bitmaps

Good old data structure for storing **dense** sets:
x-th bit is set if integer x is in the set.

Bitmaps

Good old data structure for storing **dense** sets:
x-th bit is set if integer x is in the set.

S = {0,1,5,7,8,10,11,14,18,21,22,28,29,30}

↕

1	1	0	0	0	1	0	1	1	0	1	1	0	0	1	0	0	0	1	0	0	1	1	0	0	0	0	0	1	1	1	0
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

Bitmaps

Good old data structure for storing **dense** sets:
x-th bit is set if integer x is in the set.

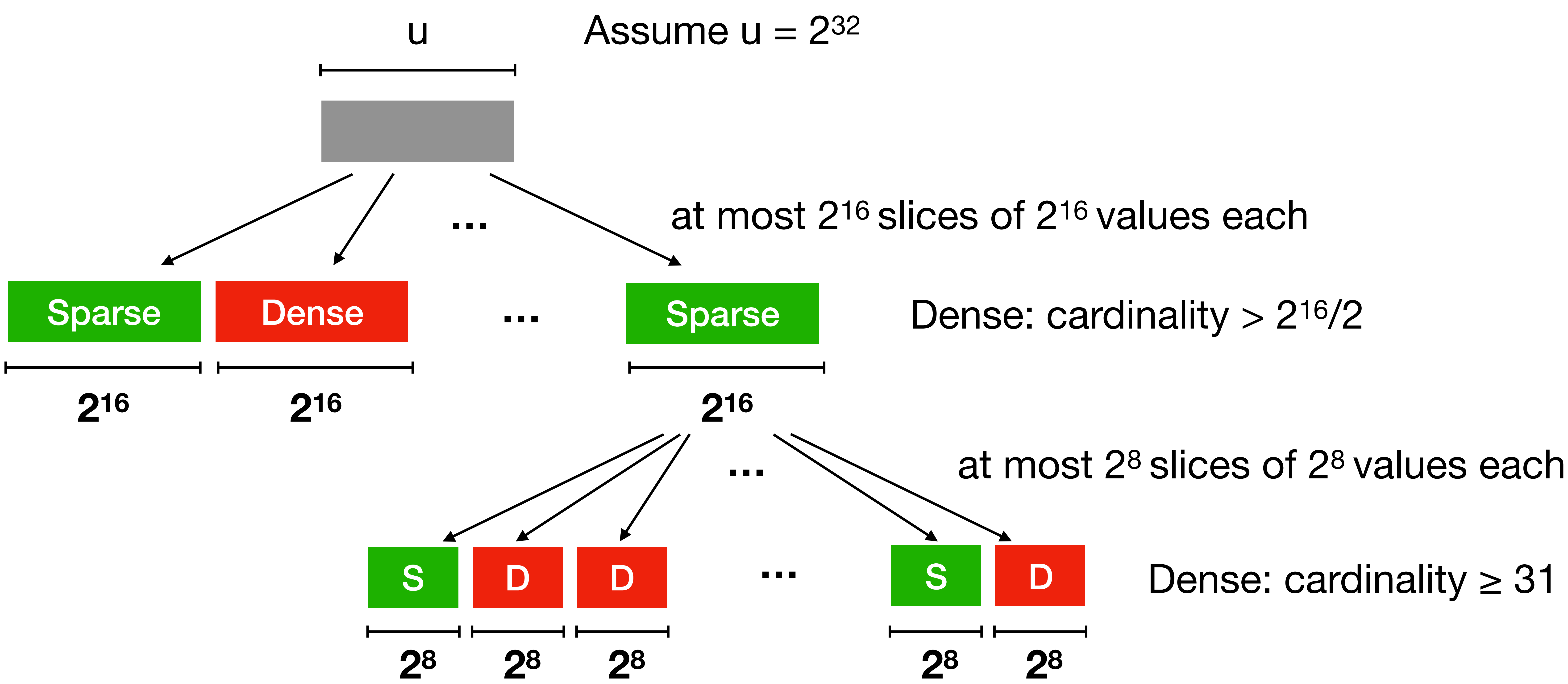
S = {0,1,5,7,8,10,11,14,18,21,22,28,29,30}

↕

1	1	0	0	0	1	0	1	1	0	1	1	0	0	1	0	0	0	1	0	0	1	1	0	0	0	0	0	1	1	1	0
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

Intersection: bitwise AND
Union: bitwise OR
Contains: testing a bit
Successor/Predecessor: __builtin_ctzll
Select: pdep + __builtin_popcnt
Max: __builtin_clzll
Min: __builtin_ctzll
Decode: __builtin_ctzll
Insertion: setting a bit
Deletion: clearing a bit

Recursive Universe Partitioning (or Slicing)



Dense slices are represented with **bitmaps** of 2^{16} or 2^8 bits.

Sparse slices are represented with **sorted-arrays** of 8-bit integers.

Intersection

Intersection between lists has to intersect only the non-empty slices in common between the lists:

- **Dense vs. Dense** (Bitmap vs. Bitmap):
Bitwise AND operations + (usually) automatic vectorization

- **Dense vs. Sparse** (Bitmap vs. Array):
Given the array A, check if bit A[i] is set in the bitmap.

- **Sparse vs. Sparse** (Array vs. Array):
Vectorized processing using _mm_cmpestrm and _mm_shuffle_epi8 **SIMD** instructions.

Experiments — Setting and Code

Machine

Intel i9-9900 CPU @3.6GHz, 64 GiB RAM, Linux 5

Compiler

gcc 9.2.1 with all optimizations enabled: -march=native and -O3

C++ code at **https://github.com/jermp/s_indexes**

Experiments — Methods and Datasets

Method	Partitioned by	SIMD	Alignment	Description
VByte	cardinality	yes	byte	fixed-size partitions of 128
Opt-VByte	cardinality	yes	bit	variable-size partitions
BIC	cardinality	no	bit	fixed-size partitions of 128
δ	cardinality	no	bit	fixed-size partitions of 128
Rice	cardinality	no	bit	fixed-size partitions of 128
PEF	cardinality	no	bit	variable-size partitions
DINT	cardinality	no	16-bit word	fixed-size partitions of 128
Opt-PFor	cardinality	no	32-bit word	fixed-size partitions of 128
Simple16	cardinality	no	64-bit word	fixed-size partitions of 128
QMX	cardinality	yes	128-bit word	fixed-size partitions of 128
Roaring	universe	yes	byte	single-span
Slicing	universe	yes	byte	multi-span

(a) basic statistics

	Gov2	ClueWeb09	CCNews
Lists	39,177	96,722	76,474
Universe	24,622,347	50,131,015	43,530,315
Integers	5,322,883,266	14,858,833,259	19,691,599,096
Entropy of the gaps	3.02	4.46	5.44
$\lceil \log_2 \rceil$ of the gaps	1.35	2.28	2.99

(b) TREC 2005/06 queries

	Gov2	ClueWeb09	CCNews
Queries	34,327	42,613	22,769
2 terms	32.2%	33.6%	37.5%
3 terms	26.8%	26.5%	27.3%
4 terms	18.2%	17.7%	16.8%
5+ terms	22.8%	22.2%	18.4%

Experiments — Compression Effectiveness and Decoding

Method	Gov2			ClueWeb09			CCNews		
	GiB	bits/int	ns/int	GiB	bits/int	ns/int	GiB	bits/int	ns/int
VByte	5.46	8.81	0.96	15.92	9.20	1.09	21.29	9.29	1.03
Opt-VByte	2.41	3.89	0.73	9.89	5.72	0.92	14.73	6.42	0.72
BIC	1.82	2.94	5.06	7.66	4.43	6.31	12.02	5.24	6.97
δ	2.32	3.74	3.56	8.95	5.17	3.72	14.58	6.36	3.85
Rice	2.53	4.08	2.92	9.18	5.31	3.25	13.34	5.82	3.32
PEF	1.93	3.12	0.76	8.63	4.99	1.10	12.50	5.45	1.31
DINT	2.19	3.53	1.13	9.26	5.35	1.56	14.76	6.44	1.65
Opt-PFor	2.25	3.63	1.38	9.45	5.46	1.79	13.92	6.07	1.53
Simple16	2.59	4.19	1.53	10.13	5.85	1.87	14.68	6.41	1.89
QMX	3.17	5.12	0.80	12.60	7.29	0.87	16.96	7.40	0.84
Roaring	4.11	6.63	0.50	16.92	9.78	0.71	21.75	9.49	0.61
Slicing	2.67	4.31	0.53	12.21	7.06	0.68	17.83	7.78	0.69

CP-based methods, such as BIC and PEF, are best for space usage.
Slicing (UP-based) stands in trade-off position.

UP-based methods, are as fast as the fastest (vectorized) CP-based methods.

Experiments — Intersections

Method	Gov2					ClueWeb09					CCNews				
	2	3	4	5+	avg.	2	3	4	5+	avg.	2	3	4	5+	avg.
VByte	2.2	2.8	2.7	3.3	2.8	10.2	12.1	13.7	13.9	12.5	14.0	22.4	19.7	21.9	19.5
Opt-VByte	2.8	3.1	2.8	3.2	3.0	12.2	13.3	14.0	13.6	13.3	16.0	23.2	19.6	20.3	19.8
BIC	6.8	9.7	10.4	13.2	10.0	31.7	44.2	51.5	53.8	45.3	45.6	79.7	76.9	88.8	72.8
δ	4.6	6.3	6.5	8.2	6.4	20.9	28.3	33.5	34.5	29.3	28.6	50.9	48.0	55.6	45.8
Rice	4.1	5.6	5.8	7.3	5.7	19.2	25.7	30.2	31.1	26.6	26.5	46.5	43.5	50.1	41.6
PEF	2.5	3.1	2.8	3.2	2.9	12.3	13.5	14.4	13.8	13.5	17.2	24.6	21.0	21.9	21.2
DINT	2.5	3.3	3.3	4.1	3.3	11.9	14.6	16.5	17.1	15.0	16.9	27.3	24.6	28.1	24.2
Opt-PFor	2.6	3.5	3.5	4.3	3.5	12.8	15.9	18.0	18.3	16.3	16.6	27.2	24.3	27.1	23.8
Simple16	2.8	3.7	3.7	4.6	3.7	12.8	16.3	18.4	18.9	16.6	17.6	28.8	26.3	29.5	25.5
QMX	2.0	2.6	2.5	3.0	2.5	9.6	11.5	13.0	13.1	11.8	13.3	21.5	18.8	20.8	18.6
Roaring	0.3	0.5	0.7	0.8	0.6	1.5	2.5	3.1	4.3	2.9	1.1	2.0	2.6	4.1	2.5
Slicing	0.3	1.0	1.2	1.6	1.0	1.5	4.5	5.4	6.7	4.5	1.8	4.3	5.1	6.0	4.3

UP-based methods outperform CP-based methods.

Future Work

- Investigate the use of **more succinct encodings** to represent the sparse regions, without hurting efficiency.
- Support **ranked** retrieval instead of boolean by means of a scoring function, such as BM25.
- Support for **insertions/deletions**.

Thank you for the attention!