Bitvectors with runs and the successor/predecessor problem

Data
Compression
Conference

Adrián Gómez-Brandón
adrian.gbrandon@udc.es

Wednesday, 25th March 2020, Snowbird, Utah

This work has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 690941

UNIVERSIDADE DA CORUÑA

citic

Outline

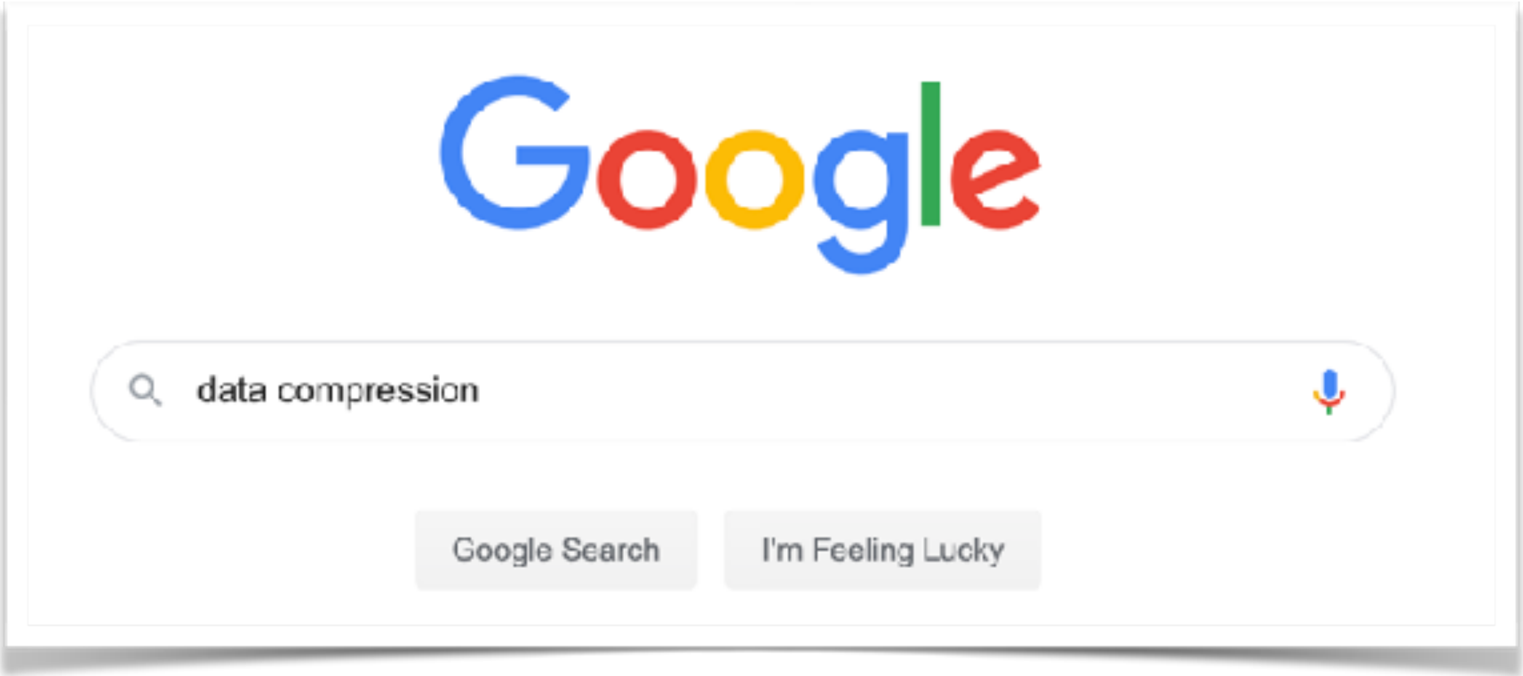
- ☐ **Introduction**
- ☐ Background
- ☐ zombit-vector
- ☐ Experimental evaluation
- ☐ Conclusions
- ☐ Future work

Introduction

- **Search engines:** looking for “*data compression*”

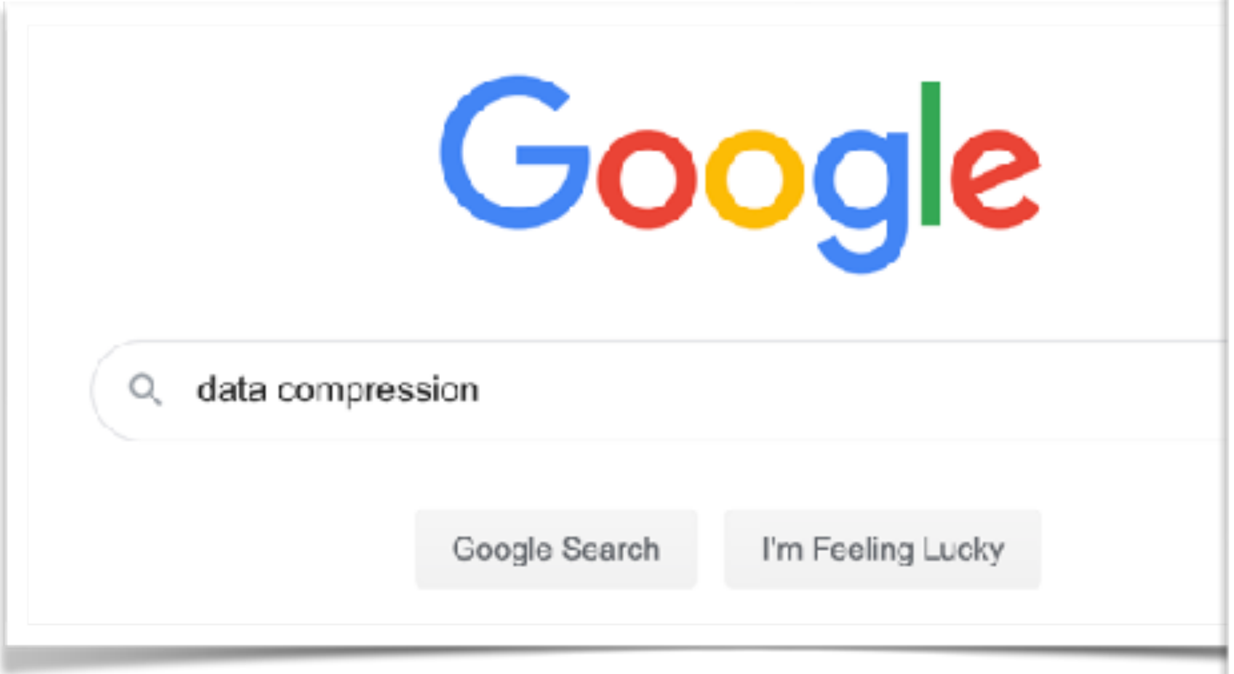
Introduction

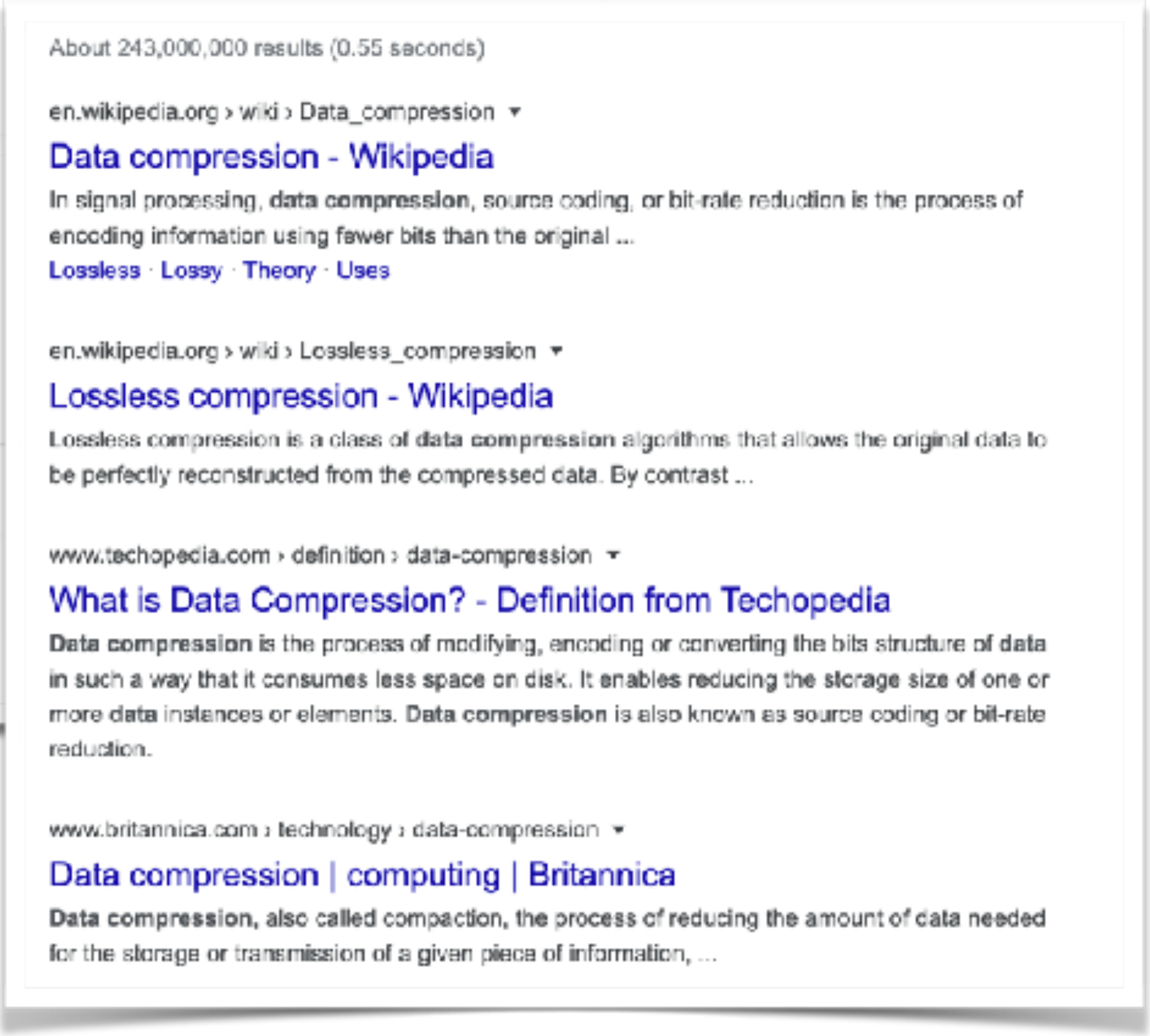
- **Search engines:** looking for "*data compression*"



Introduction

- **Search engines:** looking for “*data compression*”





About 243,000,000 results (0.55 seconds)

en.wikipedia.org › wiki › Data_compression

Data compression - Wikipedia

In signal processing, **data compression**, source coding, or bit-rate reduction is the process of encoding information using fewer bits than the original ...

Lossless · Lossy · Theory · Uses

en.wikipedia.org › wiki › Lossless_compression

Lossless compression - Wikipedia

Lossless compression is a class of **data compression** algorithms that allows the original data to be perfectly reconstructed from the compressed data. By contrast ...

www.techopedia.com › definition › data-compression

What is Data Compression? - Definition from Techopedia

Data compression is the process of modifying, encoding or converting the bits structure of data in such a way that it consumes less space on disk. It enables reducing the storage size of one or more data instances or elements. **Data compression** is also known as source coding or bit-rate reduction.

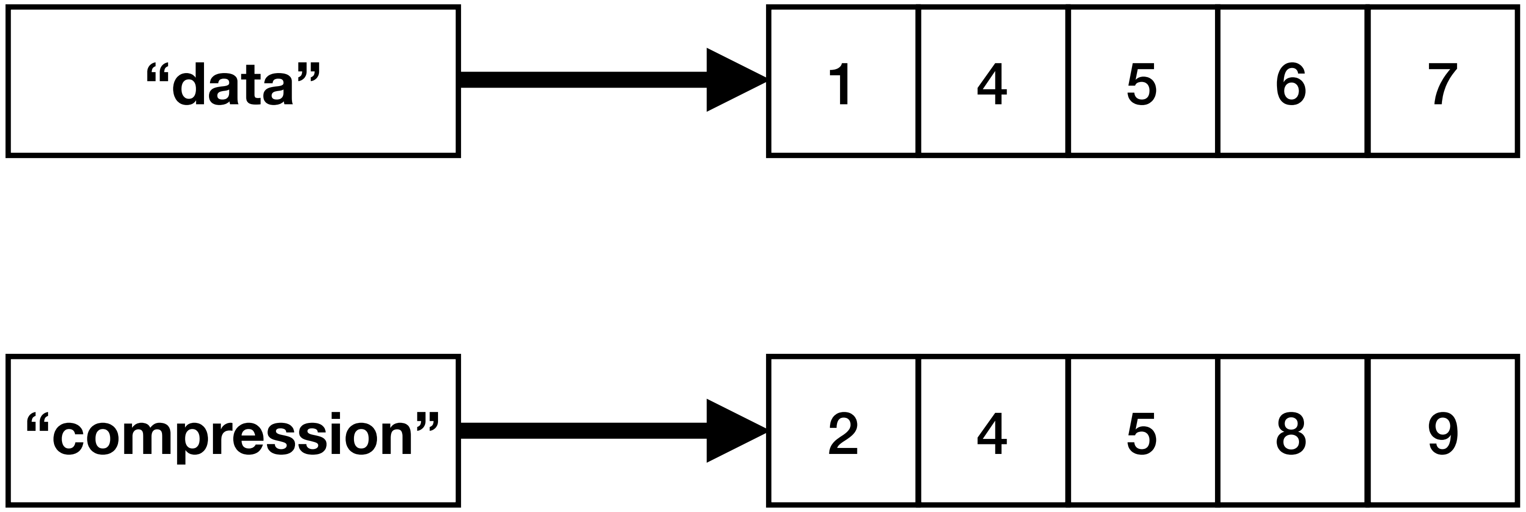
www.britannica.com › technology › data-compression

Data compression | computing | Britannica

Data compression, also called compaction, the process of reducing the amount of data needed for the storage or transmission of a given piece of information, ...

Introduction

- **Inverted lists:** each list corresponds with a term and stores sorted the document identifiers where that term appears.



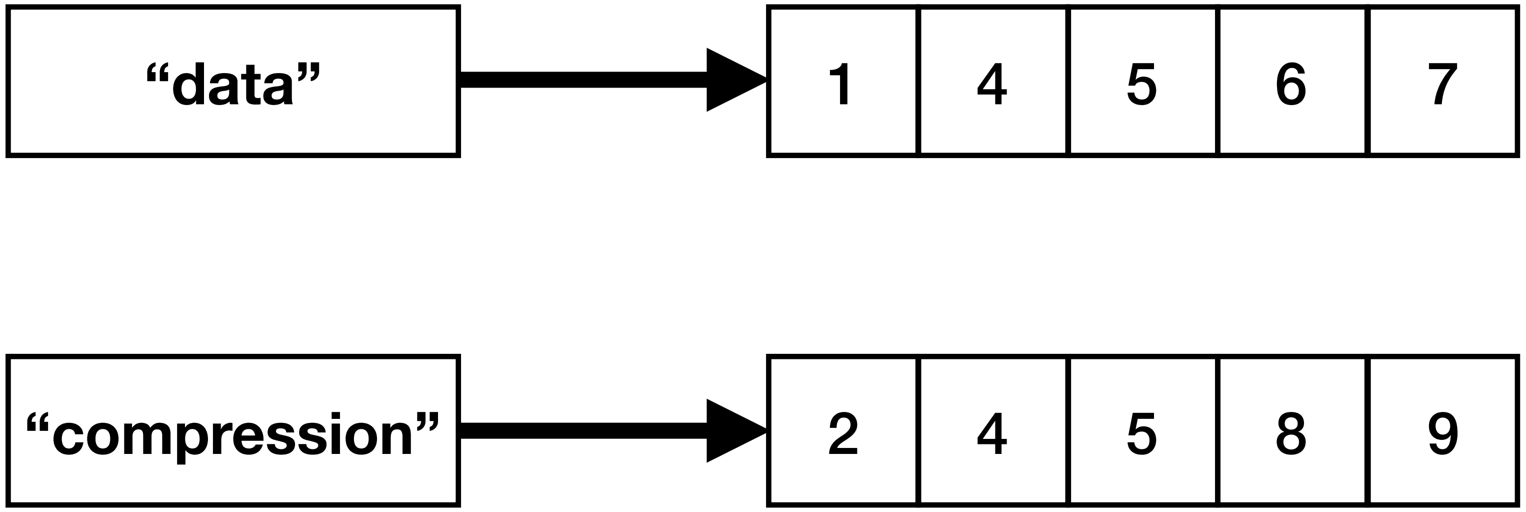
“data” → 1 4 5 6 7

“compression” → 2 4 5 8 9

Introduction

- **Inverted lists:** each list corresponds with a term and stores sorted the document identifiers where that term appears.

- Search with more than one term: **intersection of lists**



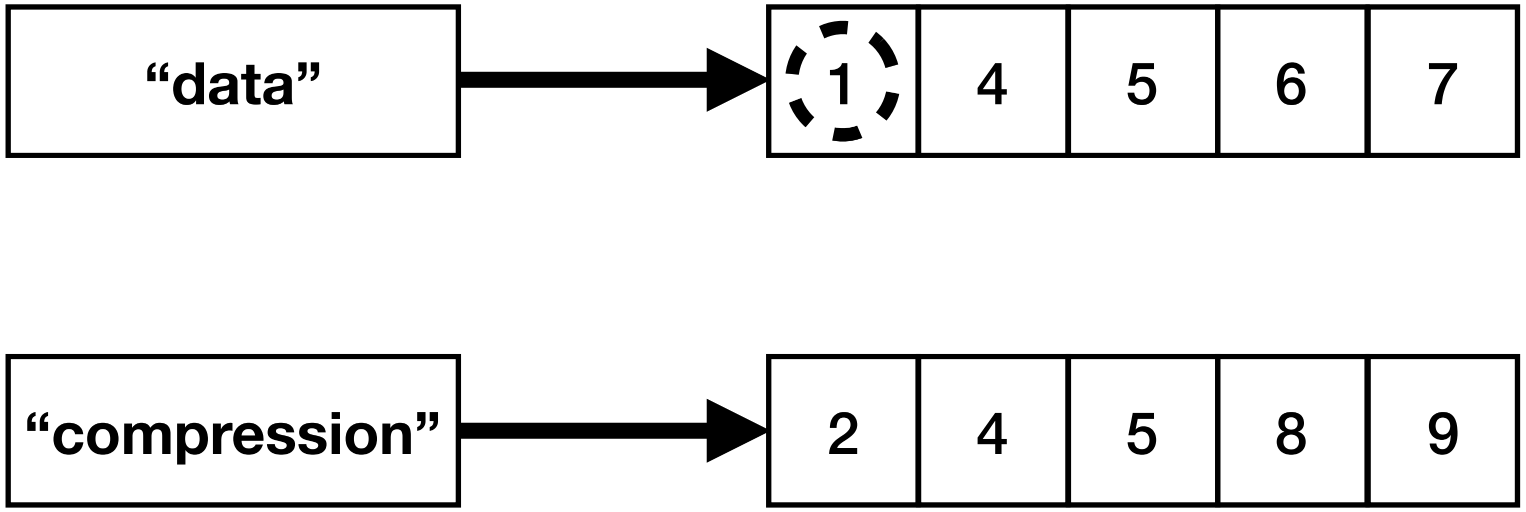
“data” → 1 4 5 6 7

“compression” → 2 4 5 8 9

Introduction

- **Inverted lists:** each list corresponds with a term and stores sorted the document identifiers where that term appears.

- Search with more than one term: **intersection of lists**

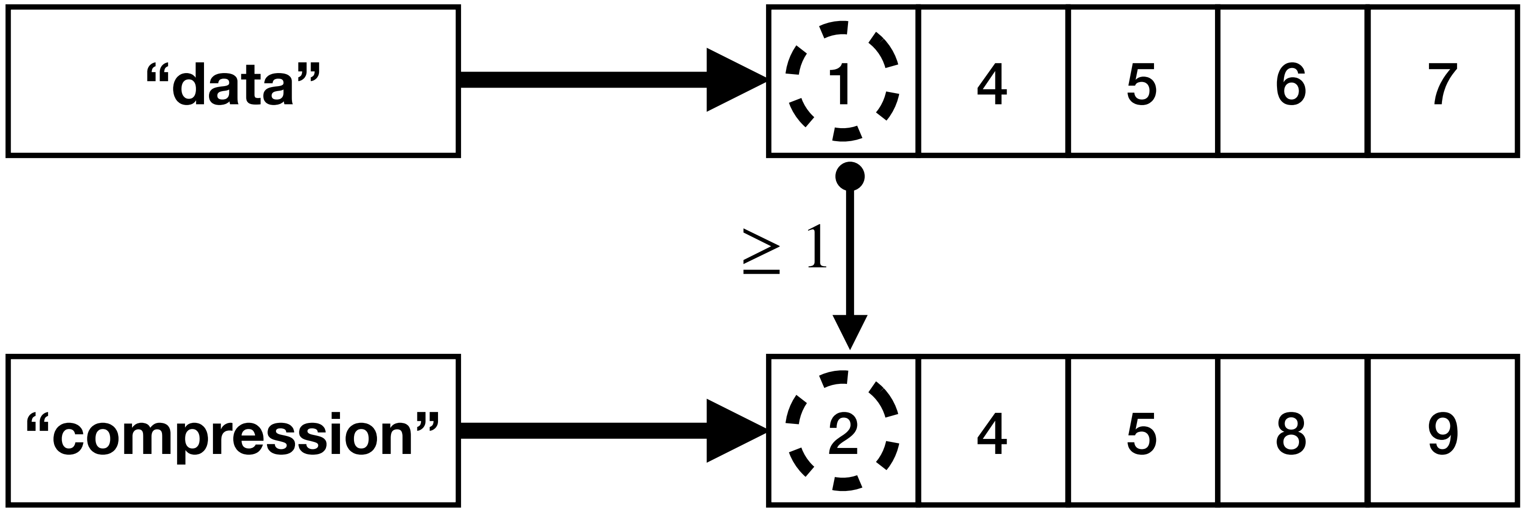


“data” → 1 | 4 | 5 | 6 | 7

“compression” → 2 | 4 | 5 | 8 | 9

Introduction

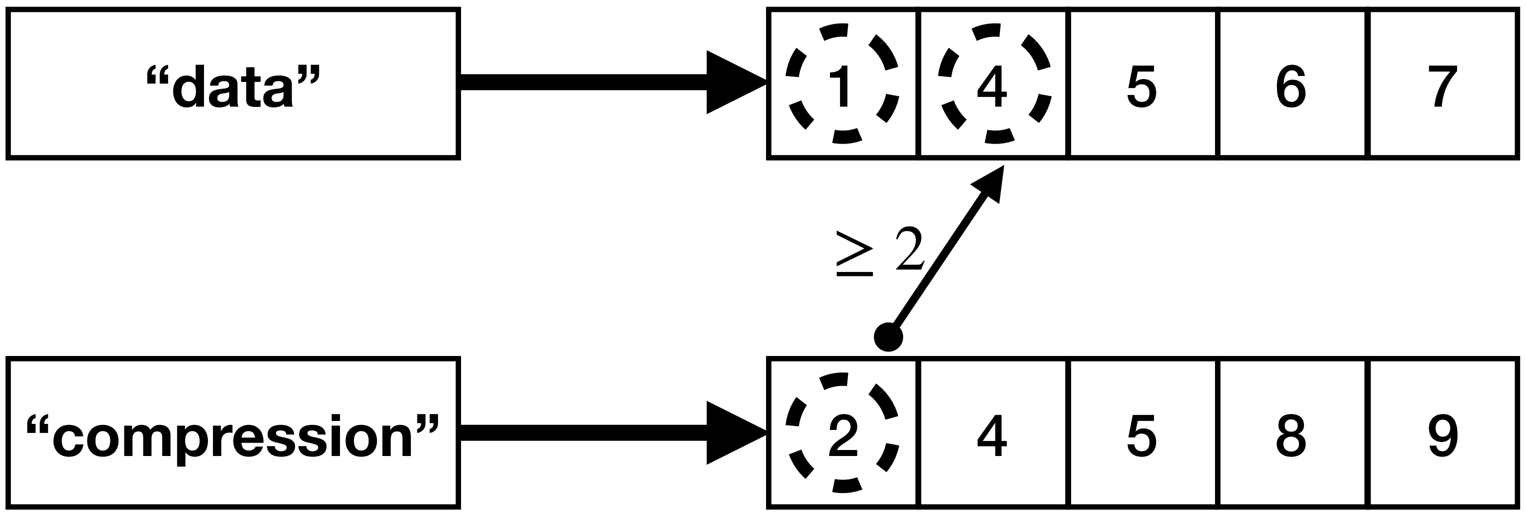
- **Inverted lists:** each list corresponds with a term and stores sorted the document identifiers where that term appears.
- Search with more than one term: **intersection of lists**



Introduction

- **Inverted lists:** each list corresponds with a term and stores sorted the document identifiers where that term appears.

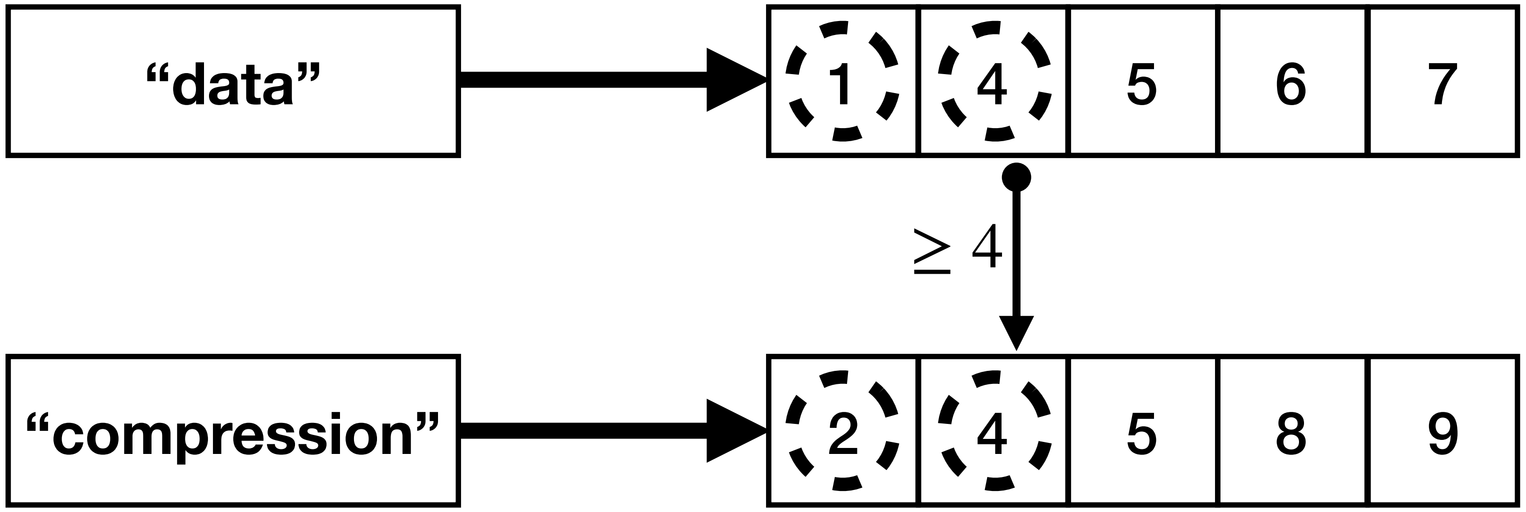
- Search with more than one term: **intersection of lists**



Introduction

- **Inverted lists:** each list corresponds with a term and stores sorted the document identifiers where that term appears.

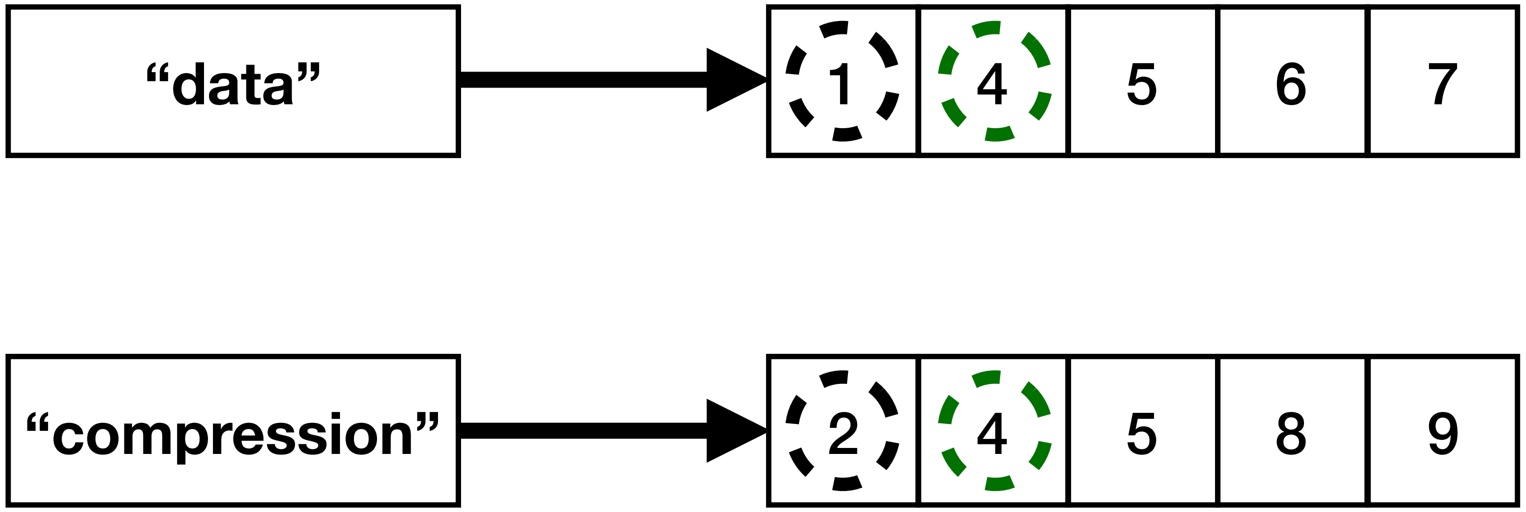
- Search with more than one term: **intersection of lists**



Introduction

- **Inverted lists:** each list corresponds with a term and stores sorted the document identifiers where that term appears.

- Search with more than one term: **intersection of lists**

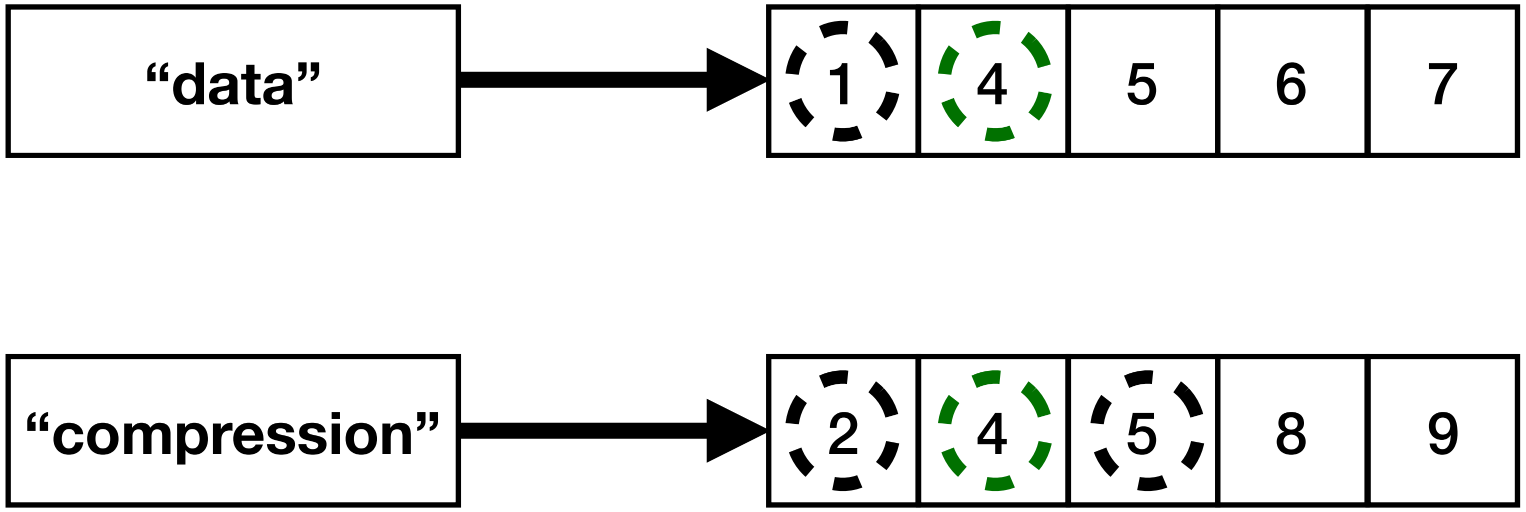


“data” → 1 4 5 6 7

“compression” → 2 4 5 8 9

Introduction

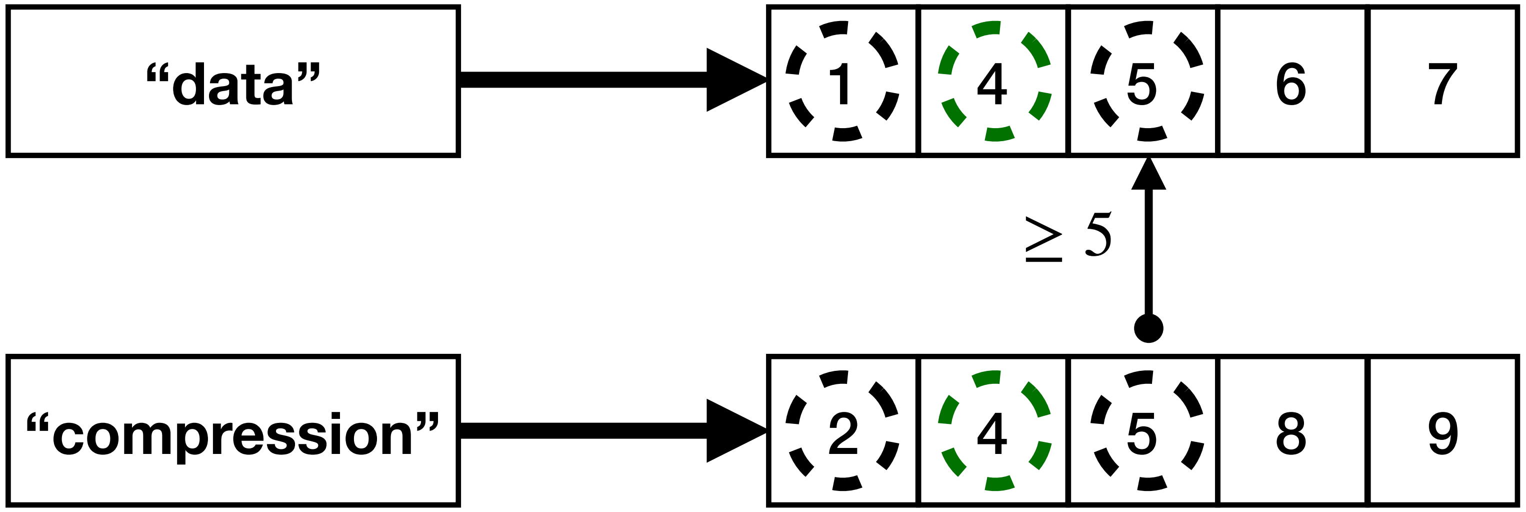
- **Inverted lists:** each list corresponds with a term and stores sorted the document identifiers where that term appears.
- Search with more than one term: **intersection of lists**



Introduction

- **Inverted lists:** each list corresponds with a term and stores sorted the document identifiers where that term appears.

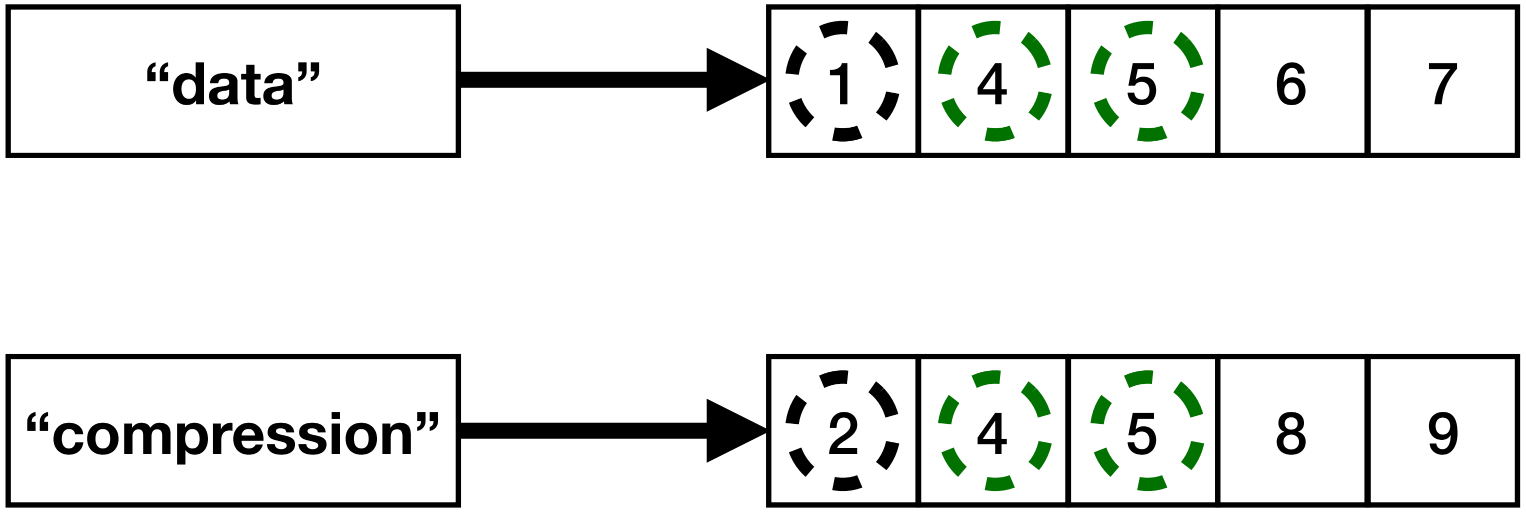
- Search with more than one term: **intersection of lists**



Introduction

- **Inverted lists:** each list corresponds with a term and stores sorted the document identifiers where that term appears.

- Search with more than one term: **intersection of lists**



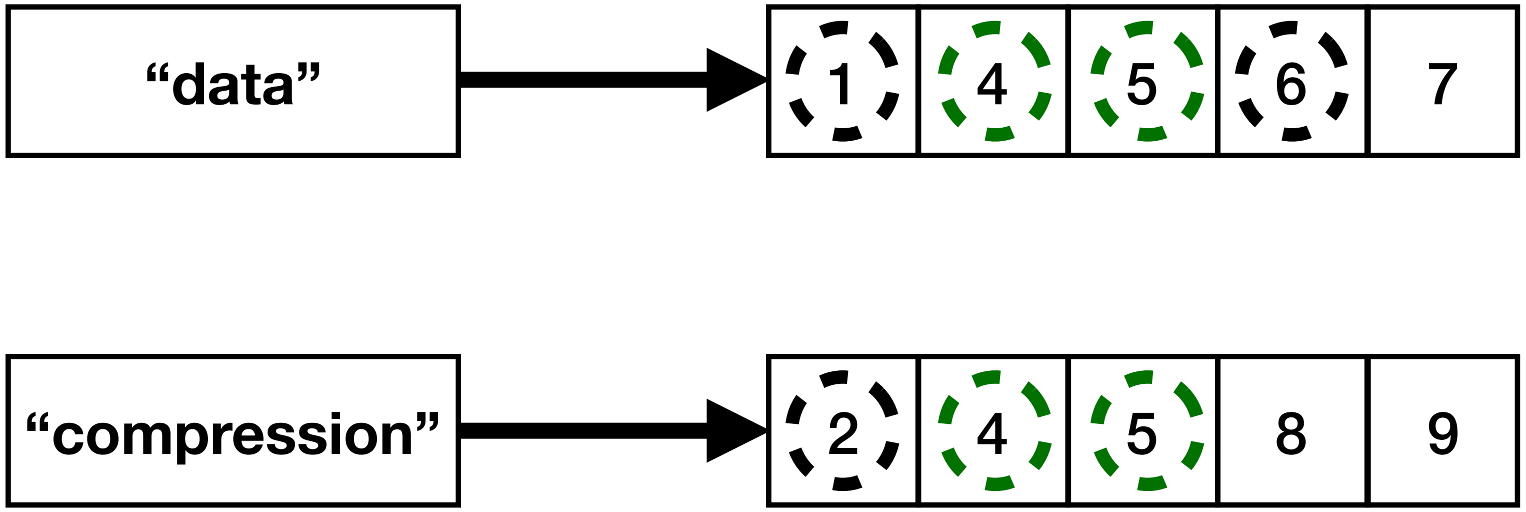
"data" → 1 4 5 6 7

"compression" → 2 4 5 8 9

Introduction

- **Inverted lists:** each list corresponds with a term and stores sorted the document identifiers where that term appears.

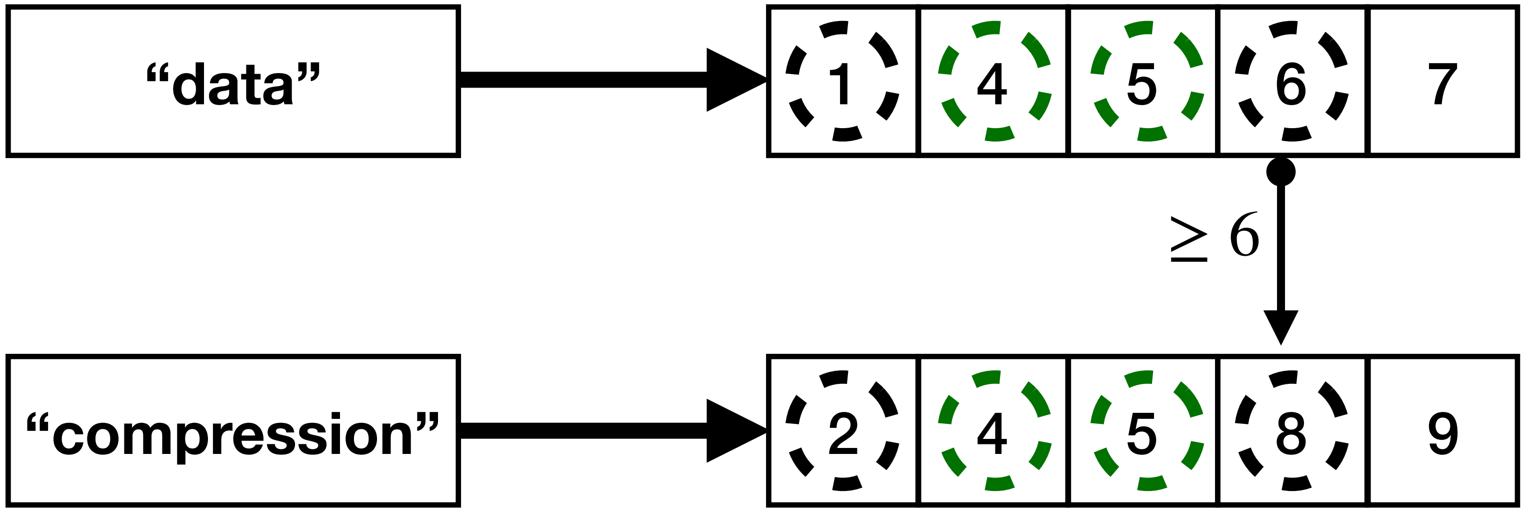
- Search with more than one term: **intersection of lists**



Introduction

- **Inverted lists:** each list corresponds with a term and stores sorted the document identifiers where that term appears.

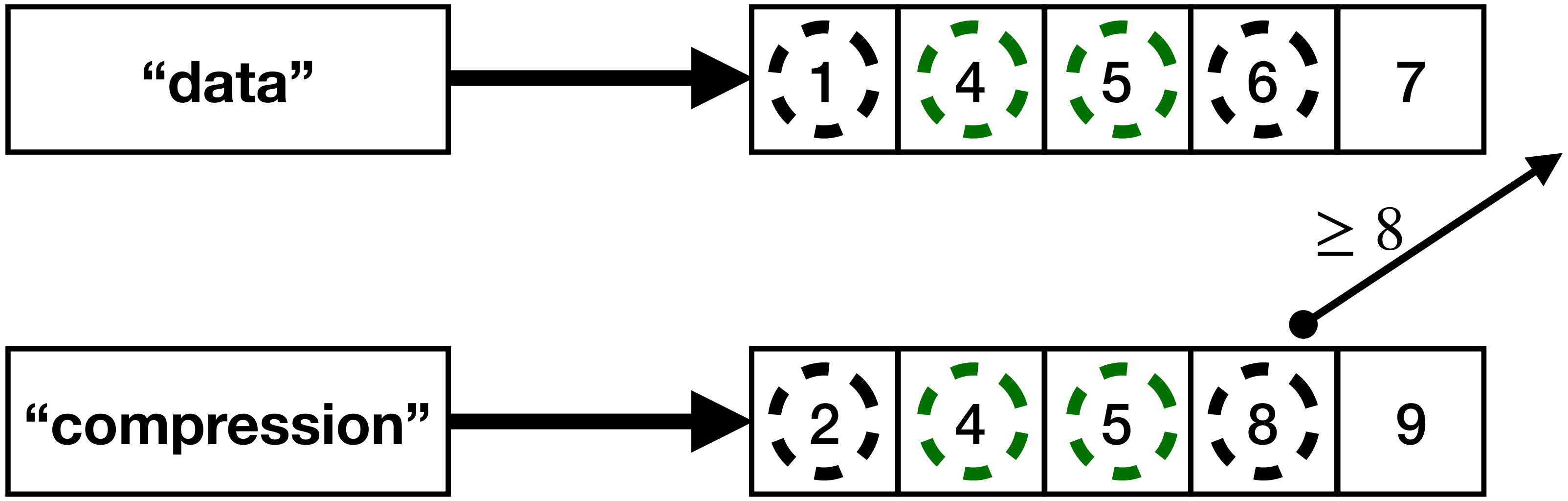
- Search with more than one term: **intersection of lists**



Introduction

- **Inverted lists:** each list corresponds with a term and stores sorted the document identifiers where that term appears.

- Search with more than one term: **intersection of lists**



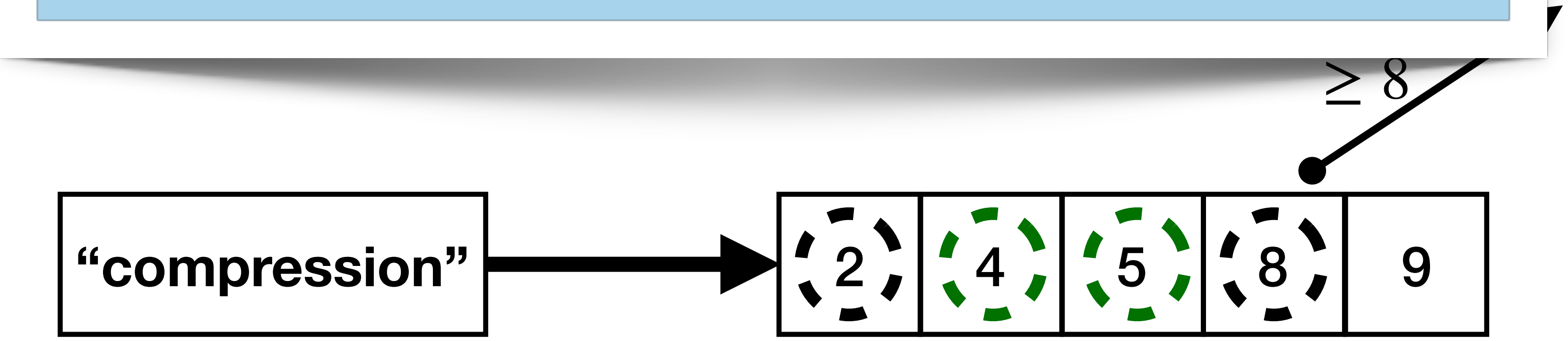
Introduction

- **Inverted lists:** each list corresponds with a term and stores sorted the document identifiers where that term appears.

- Searc **of lists**

An efficient mechanism for finding an equal or higher value in the other list is necessary

Successor and predecessor problem



Introduction

- **Successor and predecessor**

$$S = \{x_1 < x_2 < \ldots < x_m\}$$

$succ(x) = x_i$ *minimum value* $x_i \geq x$ *of* S

$pred(x) = x_i$ *maximum value* $x_i \leq x$ *of* S

Introduction

- **Successor and predecessor**

$$S = \{x_1 < x_2 < \ldots < x_m\}$$

$$succ(x) = x_i \quad minimum\ value\ x_i \geq x\ of\ S$$

$$pred(x) = x_i \quad maximum\ value\ x_i \leq x\ of\ S$$

2	4	5	8	9

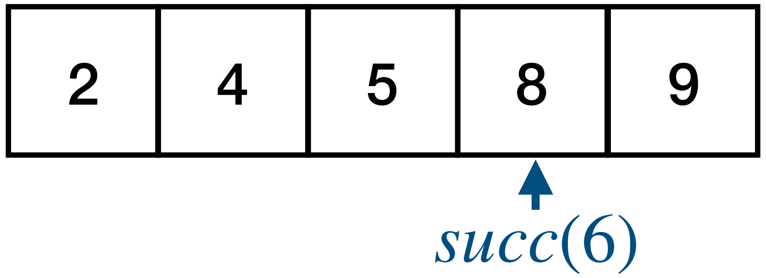
Introduction

- **Successor and predecessor**

$$S = \{x_1 < x_2 < \ldots < x_m\}$$

$succ(x) = x_i$ *minimum value* $x_i \geq x$ *of* S

$pred(x) = x_i$ *maximum value* $x_i \leq x$ *of* S



2	4	5	8	9

$succ(6)$

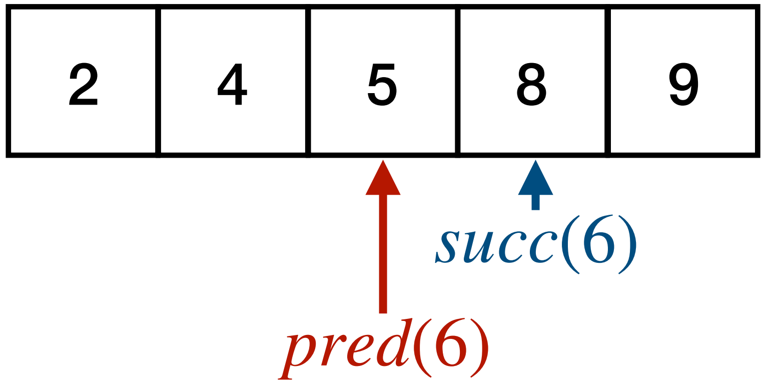
Introduction

- **Successor and predecessor**

$$S = \{x_1 < x_2 < \ldots < x_m\}$$

$succ(x) = x_i$ *minimum value* $x_i \geq x$ *of* S

$pred(x) = x_i$ *maximum value* $x_i \leq x$ *of* S



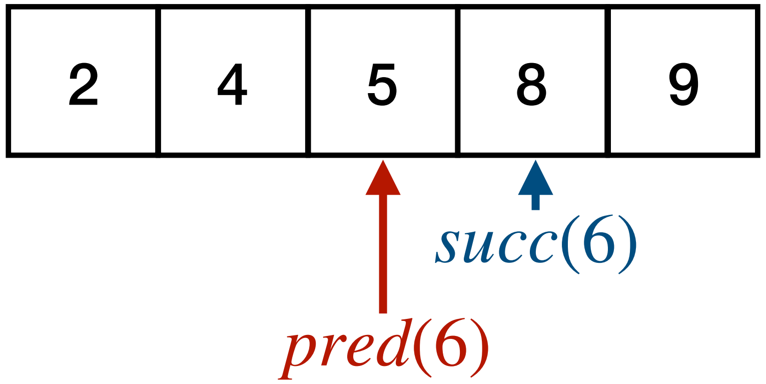
Introduction

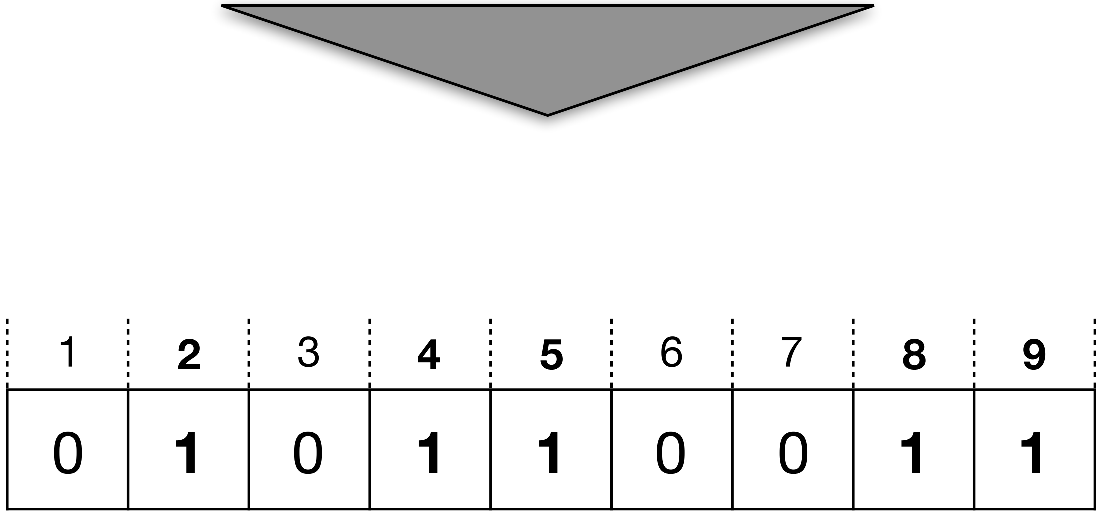
- **Successor and predecessor**

$$S = \{x_1 < x_2 < \ldots < x_m\}$$

$succ(x) = x_i$ *minimum value* $x_i \geq x$ *of* S

$pred(x) = x_i$ *maximum value* $x_i \leq x$ *of* S





1	**2**	3	**4**	**5**	6	7	**8**	**9**
0	**1**	0	**1**	**1**	0	0	**1**	**1**

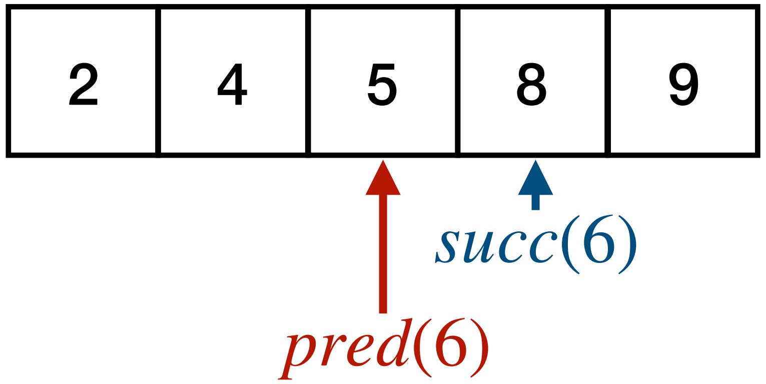
Introduction

- **Successor and predecessor**

$$S = \{x_1 < x_2 < \ldots < x_m\}$$

$succ(x) = x_i$ *minimum value* $x_i \geq x$ *of* S

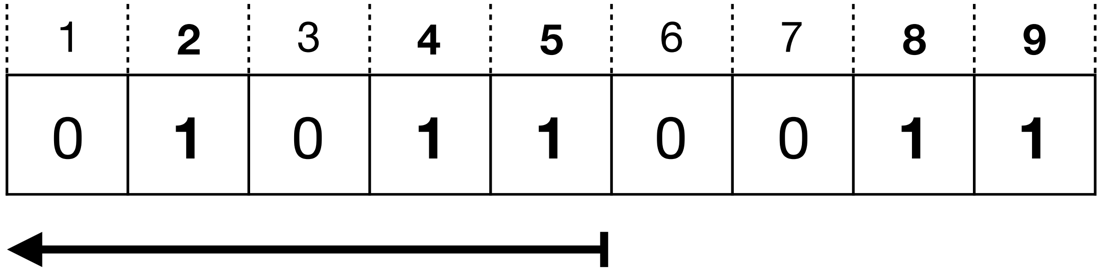
$pred(x) = x_i$ *maximum value* $x_i \leq x$ *of* S



2	4	5	8	9

$succ(6)$

$pred(6)$



1	**2**	3	**4**	**5**	6	7	**8**	**9**
0	**1**	0	**1**	**1**	0	0	**1**	**1**

$$rank_1(6-1) = 3$$

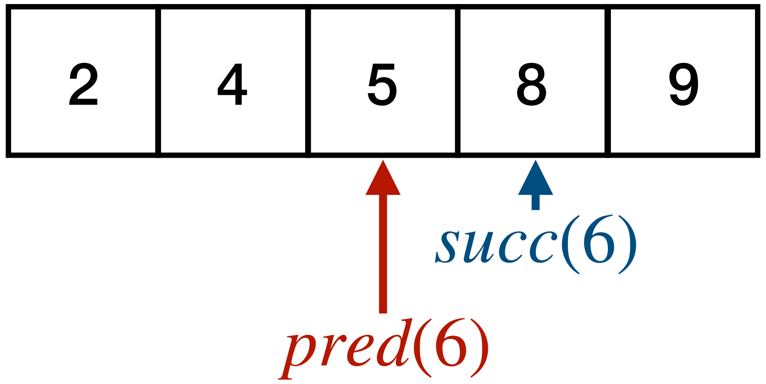
Introduction

- **Successor and predecessor**

$$S = \{x_1 < x_2 < \ldots < x_m\}$$

$succ(x) = x_i$ *minimum value* $x_i \geq x$ *of* S

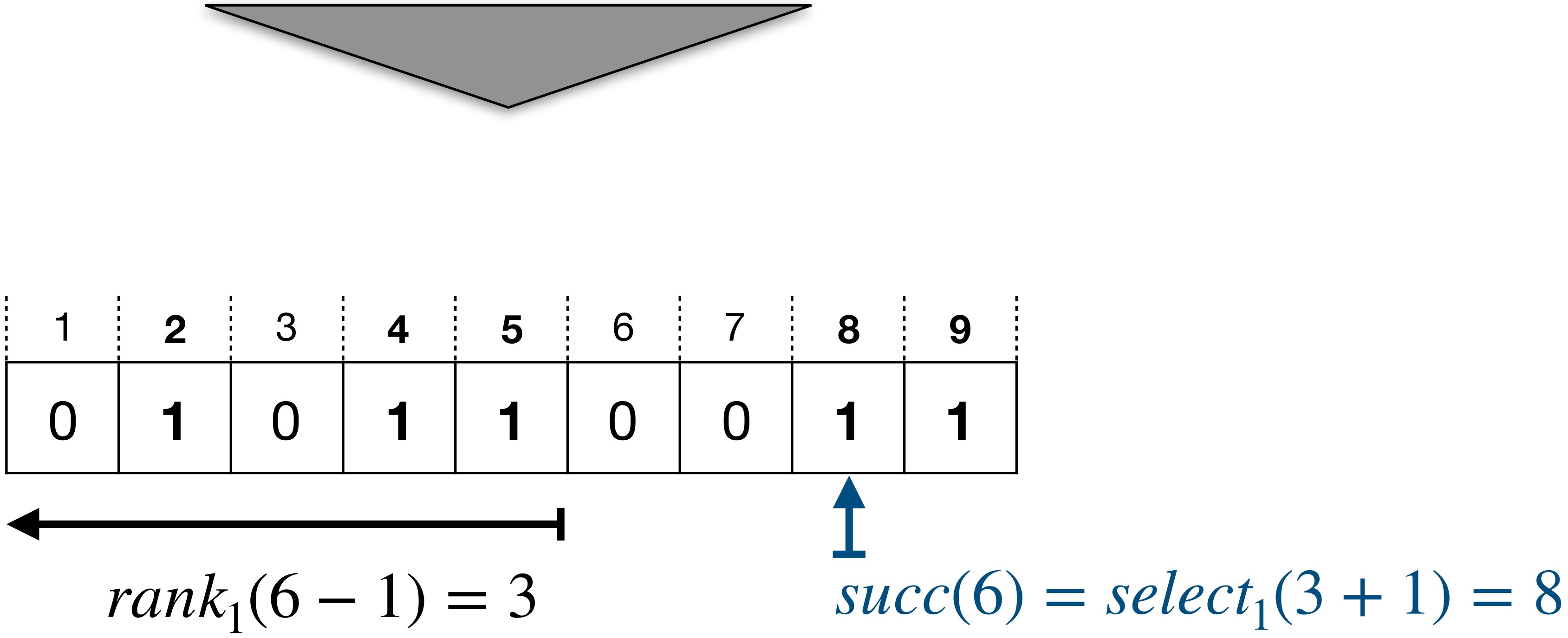
$pred(x) = x_i$ *maximum value* $x_i \leq x$ *of* S



2	4	5	8	9

$succ(6)$

$pred(6)$



1	**2**	3	**4**	**5**	6	7	**8**	**9**
0	**1**	0	**1**	**1**	0	0	**1**	**1**

$rank_1(6 - 1) = 3$

$succ(6) = select_1(3 + 1) = 8$

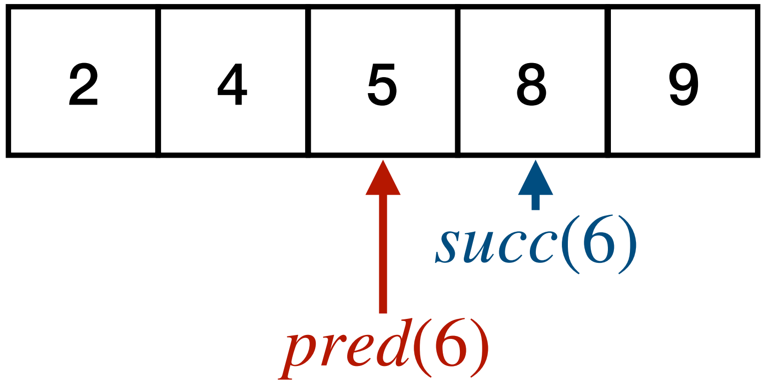
Introduction

- **Successor and predecessor**

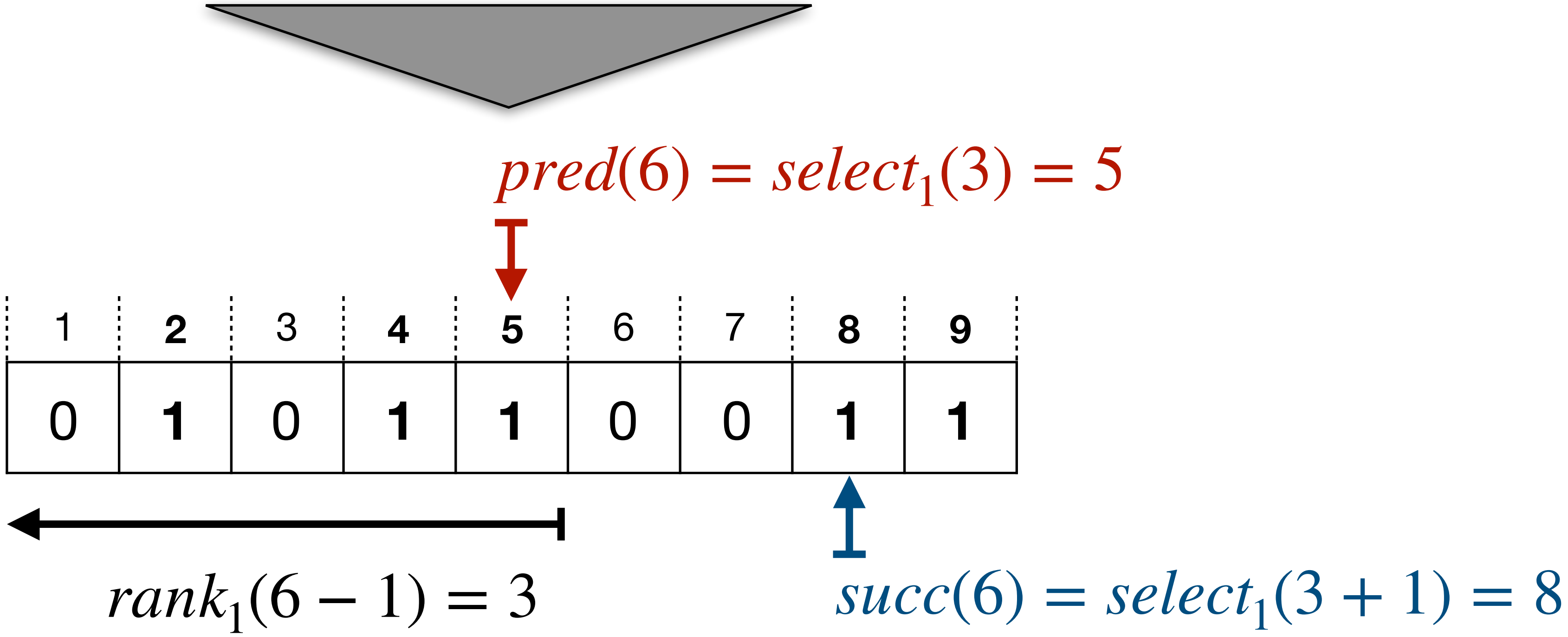
$$S = \{x_1 < x_2 < \ldots < x_m\}$$

$succ(x) = x_i$ minimum value $x_i \geq x$ of S

$pred(x) = x_i$ maximum value $x_i \leq x$ of S



2	4	5	8	9



$pred(6) = select_1(3) = 5$

1	**2**	3	**4**	**5**	6	7	**8**	**9**
0	**1**	0	**1**	**1**	0	0	**1**	**1**

$rank_1(6-1) = 3$

$succ(6) = select_1(3+1) = 8$

Introduction

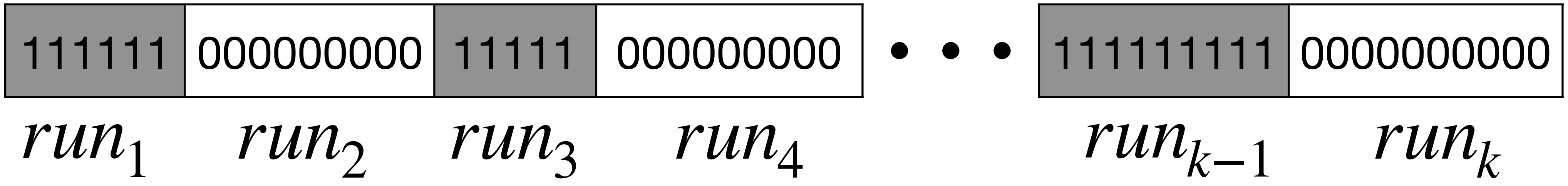
- Successor and predecessor on bitvectors **with *k* runs**

Introduction

- Successor and predecessor on bitvectors **with *k* runs**
  - A run is a sequence of consecutive bits with the same value

Introduction

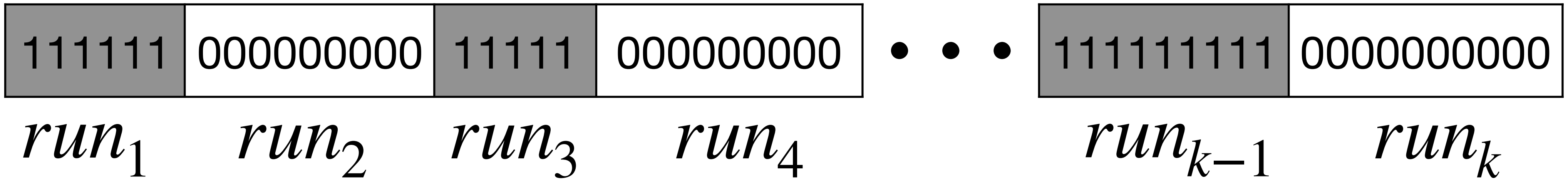
- Successor and predecessor on bitvectors **with *k* runs**
  - A run is a sequence of consecutive bits with the same value



111111	000000000	11111	000000000	• • •	111111111	0000000000
run_1	run_2	run_3	run_4		run_{k-1}	run_k

Introduction

- Successor and predecessor on bitvectors **with *k* runs**
  - A run is a sequence of consecutive bits with the same value



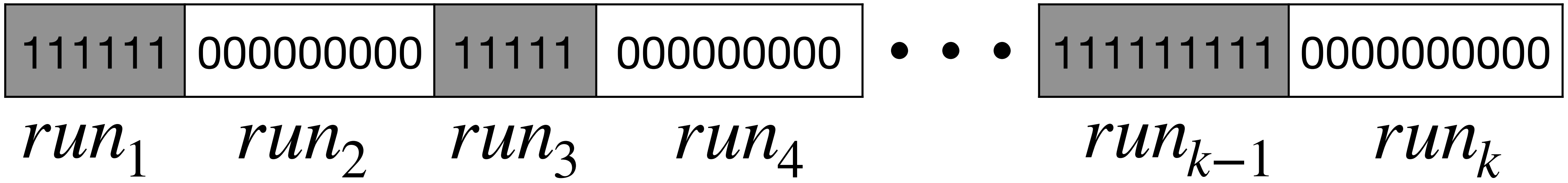
111111 000000000 11111 000000000 • • • 111111111 0000000000

run_1 run_2 run_3 run_4 run_{k-1} run_k

  - Inverted lists where the documents are sorted by URL (e.g. the different pages from a real-state company could produce a run of ones on the inverted list for the word “house”)

Introduction

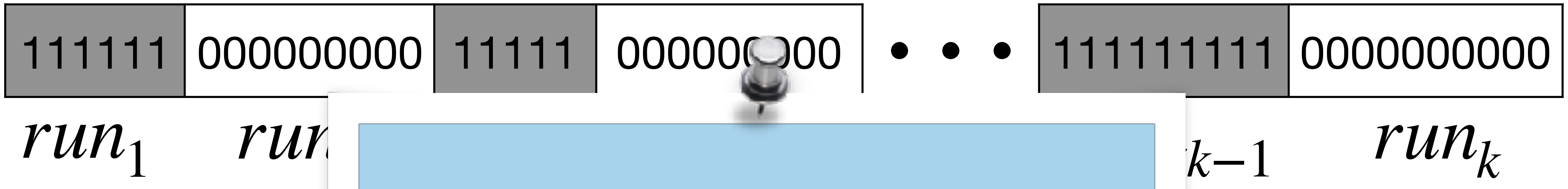
- Successor and predecessor on bitvectors **with *k* runs**
  - A run is a sequence of consecutive bits with the same value



  - Inverted lists where the documents are sorted by URL (e.g. the different pages from a real-state company could produce a run of ones on the inverted list for the word “house”)
  - Representations of moving objects (e.g. tracking the timestamps where a vessel is moving)

Introduction

- Successor and predecessor on bitvectors **with *k* runs**
  - A run is a sequence of consecutive bits with the same value



Successor and predecessor in *O(1)* time and exploiting the *k* runs

  - Inverted lists … by URL (e.g. the different pag… uld produce a run of ones on the inverted list for the word “house”)
  - Representations of moving objects (e.g. tracking the timestamps where a vessel is moving)

Outline

- ☑ Introduction
- ☐ **Background**
- ☐ zombit-vector
- ☐ Experimental evaluation
- ☐ Conclusions
- ☐ Future work

Background

- **Plain bitvector:** requires $n + o(n)$ space and takes $O(1)$ time for solving *succ* and *pred*. [plain]

Background

- **Plain bitvector:** requires $n + o(n)$ space and takes $O(1)$ time for solving *succ* and *pred*. [plain]
- **Zero-order bitvector:** compresses close to the zero order entropy $(nH_0 + o(n))$. [rrr]

Background

- **Plain bitvector:** requires $n + o(n)$ space and takes $O(1)$ time for solving *succ* and *pred*. [plain]

- **Zero-order bitvector:** compresses close to the zero order entropy ($nH_0 + o(n)$). [rrr]

- **Sparse bitvector:** avoids the dependency of $o(n)$ for those bitvectors, whose number of ones m are much smaller than n. [sd-array, rec-rank]

Background

- **Plain bitvector:** requires $n + o(n)$ space and takes $O(1)$ time for solving *succ* and *pred*. [plain]

- **Zero-order bitvector:** compresses close to the zero order entropy $(nH_0 + o(n))$. [rrr]

- **Sparse bitvector:** avoids the dependency of $o(n)$ for those bitvectors, whose number of ones m are much smaller than n. [sd-array, rec-rank]

- **Bitvector with runs:** its compression exploits the runs by transforming it into two sparse bitvectors. [oz-vector]

Background

- **Plain bitvector:** requires $n + o(n)$ space and takes $O(1)$ time for solving *succ* and *pred*. [plain]

- **Zero-order bitvector:** compresses close to the zero order entropy $(nH_0 + o(n))$. [rrr]

- **Sparse bitvector:** avoids the dependency of o(n) for those bitvectors, whose number of ones *m* are much smaller than *n*. [sd-array, rec-rank]

- **Bitvector with runs:** its compression exploits the runs by transforming it into two sparse bitvectors. [oz-vector]

- **Hybrid bitvector:** splits the bitvector into different parts and choose the best technique (*sparse*, *runs*, *plain*) for each individual division. [hybrid]

Background

Bitvector	**Time**	**Space**
plain	$O(1)$	$n + o(n)$
rrr	$O(1)$	$nH_0 + o(n)$
rec-rank	$O(\log \frac{n}{m})$	$\log \frac{n}{m} + m + o(n)$
sd-array	$O(\log \frac{n}{m})$	$m \log \frac{n}{m} + O(m)$
oz-vector	$O(\log k)$	$k \log \frac{2n}{k} + O(k)$
hybrid	$O(\log n)$	$min(k, m) \lceil \log b \rceil + o(n)$

Background

Bitvector	**Time**	**Space**
plain	$O(1)$	$n + o(n)$
rrr	$O(1)$	$nH_0 + o(n)$
rec-rank	$O(\log \frac{n}{m})$	$\log \frac{n}{m} + m + o(n)$
sd-array	$O(\log \frac{n}{m})$	$m \log \frac{n}{m} + O(m)$
oz-vector	$O(\log k)$	$k \log \frac{2n}{k} + O(k)$
hybrid	$O(\log n)$	$min(k, m)\lceil \log b \rceil + o(n)$

Outline

- ☑ Introduction
- ☑ Background
- ☐ **zombit-vector**
- ☐ Experimental evaluation
- ☐ Conclusions
- ☐ Future work

zombit-vector

- Compressing bitvectors exploiting its **runs** and solving successor and predecessor operations in **O(1) time**

zombit-vector

- Compressing bitvectors exploiting its **runs** and solving successor and predecessor operations in **O(1) time**

- **Partitions X_i of β size** and classification of them in three sets:

zombit-vector

- Compressing bitvectors exploiting its **runs** and solving successor and predecessor operations in **O(1) time**

- **Partitions X_i of β size** and classification of them in three sets:
    - $\mathbb{Z}$: uniform blocks full of zeroes

zombit-vector

- Compressing bitvectors exploiting its **runs** and solving successor and predecessor operations in **O(1) time**

- **Partitions X_i of β size** and classification of them in three sets:
    - $\mathbb{Z}$: uniform blocks full of zeroes
    - $\mathbb{O}$: uniform blocks full of ones

zombit-vector

- Compressing bitvectors exploiting its **runs** and solving successor and predecessor operations in **O(1) time**

- **Partitions X_i of β size** and classification of them in three sets:
  - $\mathbb{Z}$: uniform blocks full of zeroes
  - $\mathbb{O}$: uniform blocks full of ones
  - $\mathbb{M}$: mixed blocks, which contain both bits

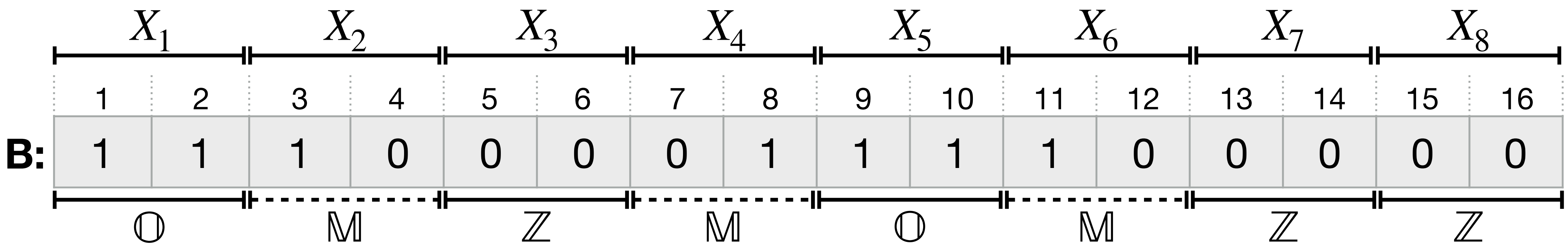
zombit-vector

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
B:	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	0

zombit-vector

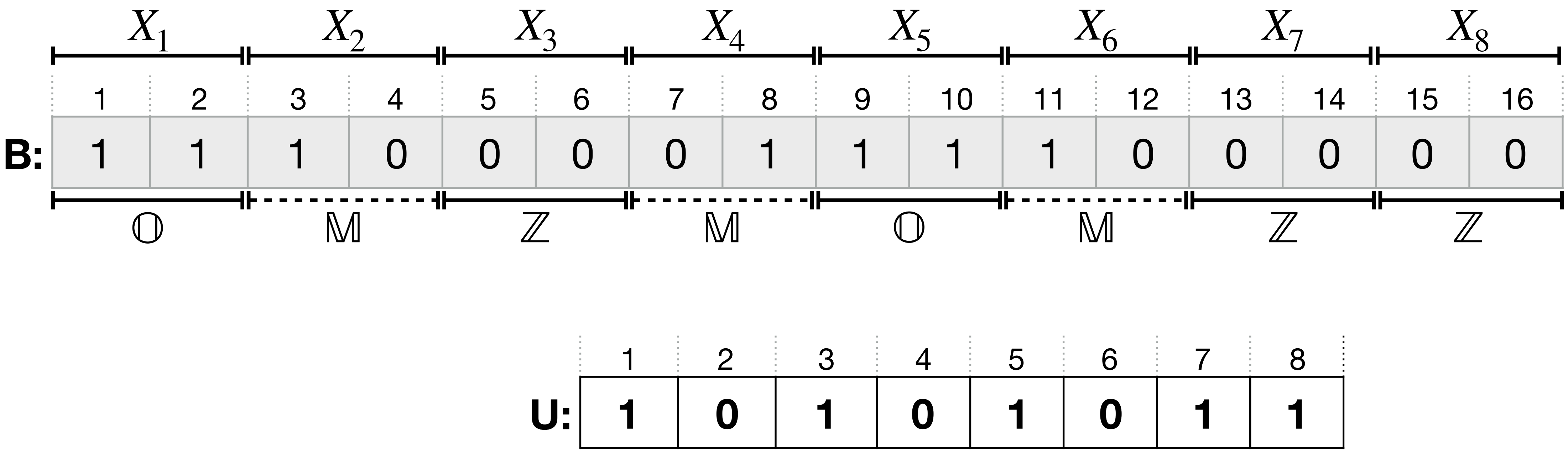
	X_1		X_2		X_3		X_4		X_5		X_6		X_7		X_8	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
B:	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	0

zombit-vector



	X_1		X_2		X_3		X_4		X_5		X_6		X_7		X_8	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
B:	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	0
	$\mathbb{O}$		$\mathbb{M}$		$\mathbb{Z}$		$\mathbb{M}$		$\mathbb{O}$		$\mathbb{M}$		$\mathbb{Z}$		$\mathbb{Z}$	

zombit-vector

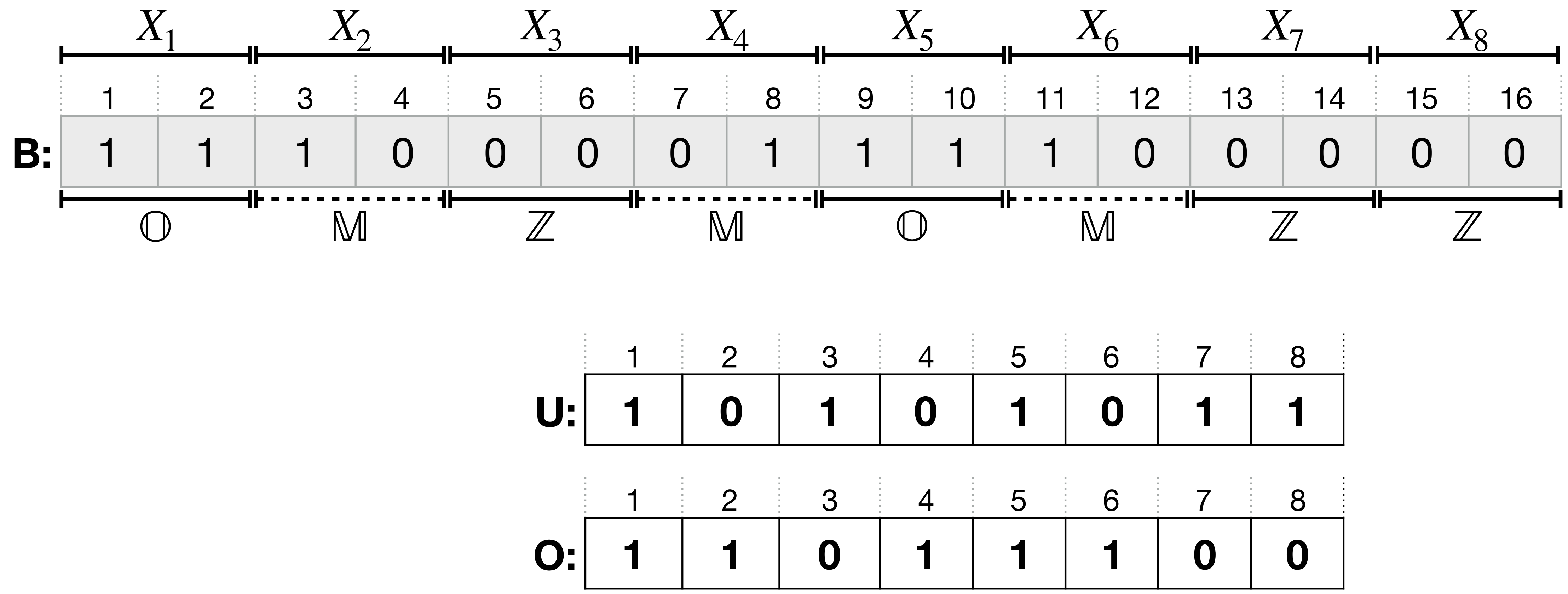


X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8
𝕆	𝕄	ℤ	𝕄	𝕆	𝕄	ℤ	ℤ

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
B:	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	0

	1	2	3	4	5	6	7	8
U:	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**1**

zombit-vector

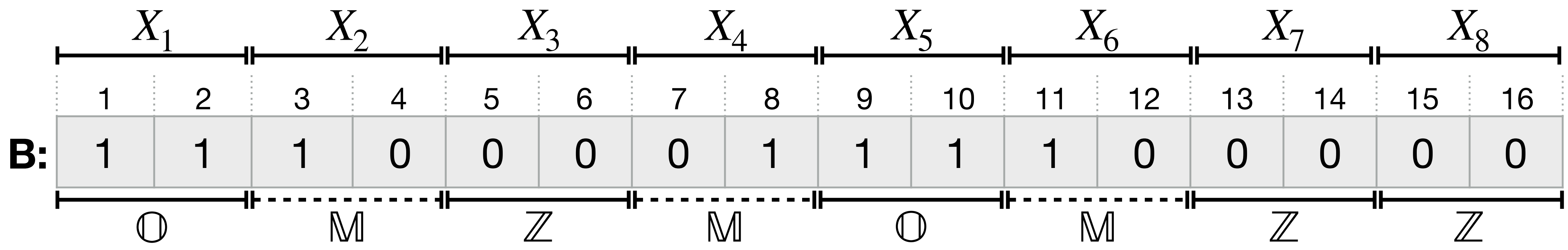


	X_1		X_2		X_3		X_4		X_5		X_6		X_7		X_8	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
B:	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	0
	$\mathbb{O}$		$\mathbb{M}$		$\mathbb{Z}$		$\mathbb{M}$		$\mathbb{O}$		$\mathbb{M}$		$\mathbb{Z}$		$\mathbb{Z}$	

	1	2	3	4	5	6	7	8
U:	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**1**

	1	2	3	4	5	6	7	8
O:	**1**	**1**	**0**	**1**	**1**	**1**	**0**	**0**

zombit-vector



	X_1		X_2		X_3		X_4		X_5		X_6		X_7		X_8	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
B:	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	0
	$\mathbb{O}$		$\mathbb{M}$		$\mathbb{Z}$		$\mathbb{M}$		$\mathbb{O}$		$\mathbb{M}$		$\mathbb{Z}$		$\mathbb{Z}$	

	1	2	3	4	5	6	7	8
U:	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**1**

	1	2	3	4	5	6	7	8
O:	**1**	**1**	**0**	**1**	**1**	**1**	**0**	**0**

	1	2	3	4	5	6
M:	**1**	**0**	**0**	**1**	**1**	**0**

zombit-vector

	X_1		X_2		X_3		X_4		X_5		X_6		X_7		X_8	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
B:	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	0
	$\mathbb{O}$		$\mathbb{M}$		$\mathbb{Z}$		$\mathbb{M}$		$\mathbb{O}$		$\mathbb{M}$		$\mathbb{Z}$		$\mathbb{Z}$	

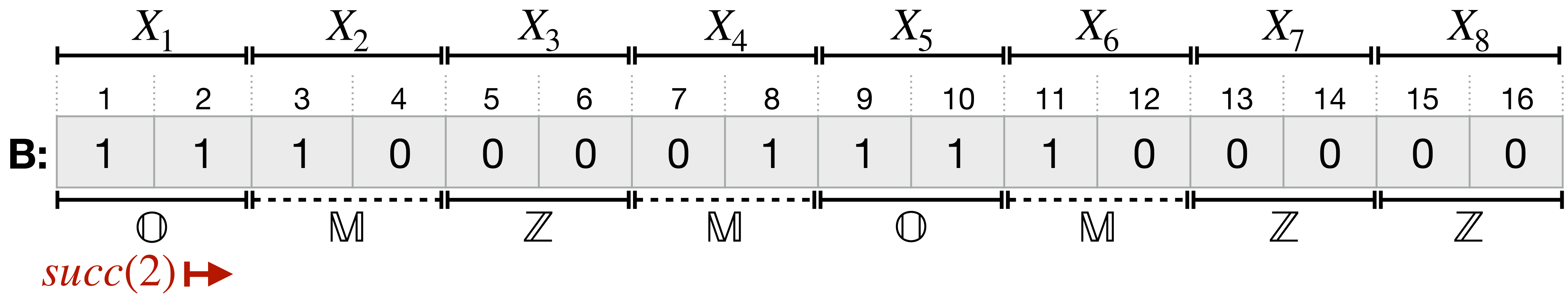
zombit-vector

	1	2	3	4	5	6	7	8
U:	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**1**

	1	2	3	4	5	6	7	8
O:	**1**	**1**	**0**	**1**	**1**	**1**	**0**	**0**

	1	2	3	4	5	6
M:	**1**	**0**	**0**	**1**	**1**	**0**

zombit-vector



	X_1		X_2		X_3		X_4		X_5		X_6		X_7		X_8	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
B:	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	0
	$\mathbb{O}$		$\mathbb{M}$		$\mathbb{Z}$		$\mathbb{M}$		$\mathbb{O}$		$\mathbb{M}$		$\mathbb{Z}$		$\mathbb{Z}$	

$succ(2) \mapsto$

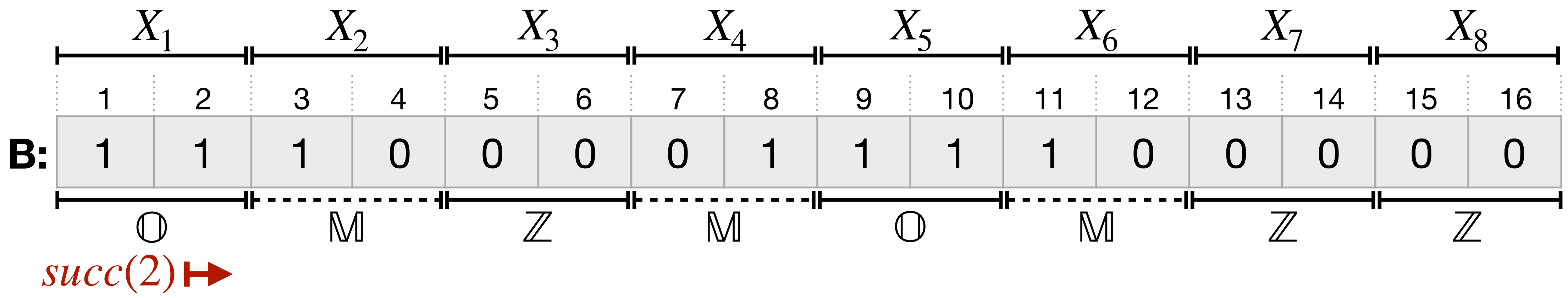
zombit-vector

	1	2	3	4	5	6	7	8
U:	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**1**

	1	2	3	4	5	6	7	8
O:	**1**	**1**	**0**	**1**	**1**	**1**	**0**	**0**

	1	2	3	4	5	6
M:	**1**	**0**	**0**	**1**	**1**	**0**

zombit-vector



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
B:	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	0

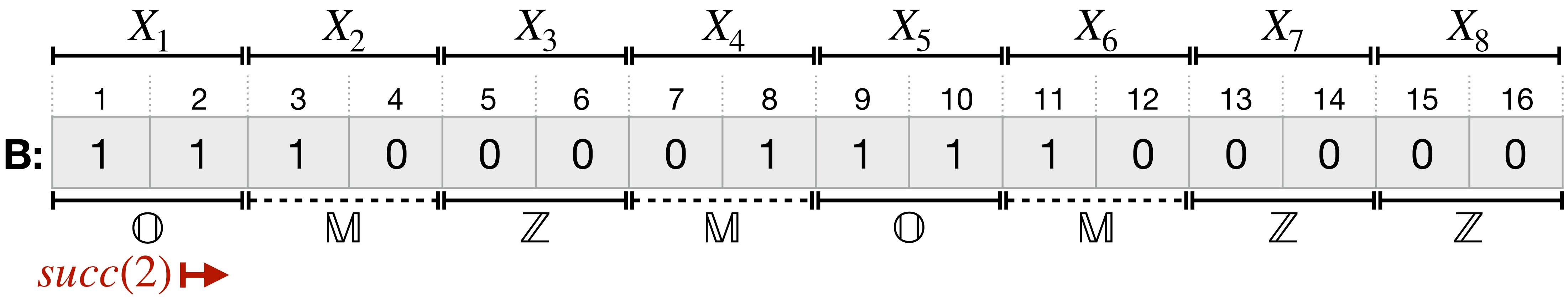
zombit-vector

	1	2	3	4	5	6	7	8
U:	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**1**

	1	2	3	4	5	6	7	8
O:	**1**	**1**	**0**	**1**	**1**	**1**	**0**	**0**

	1	2	3	4	5	6
M:	**1**	**0**	**0**	**1**	**1**	**0**

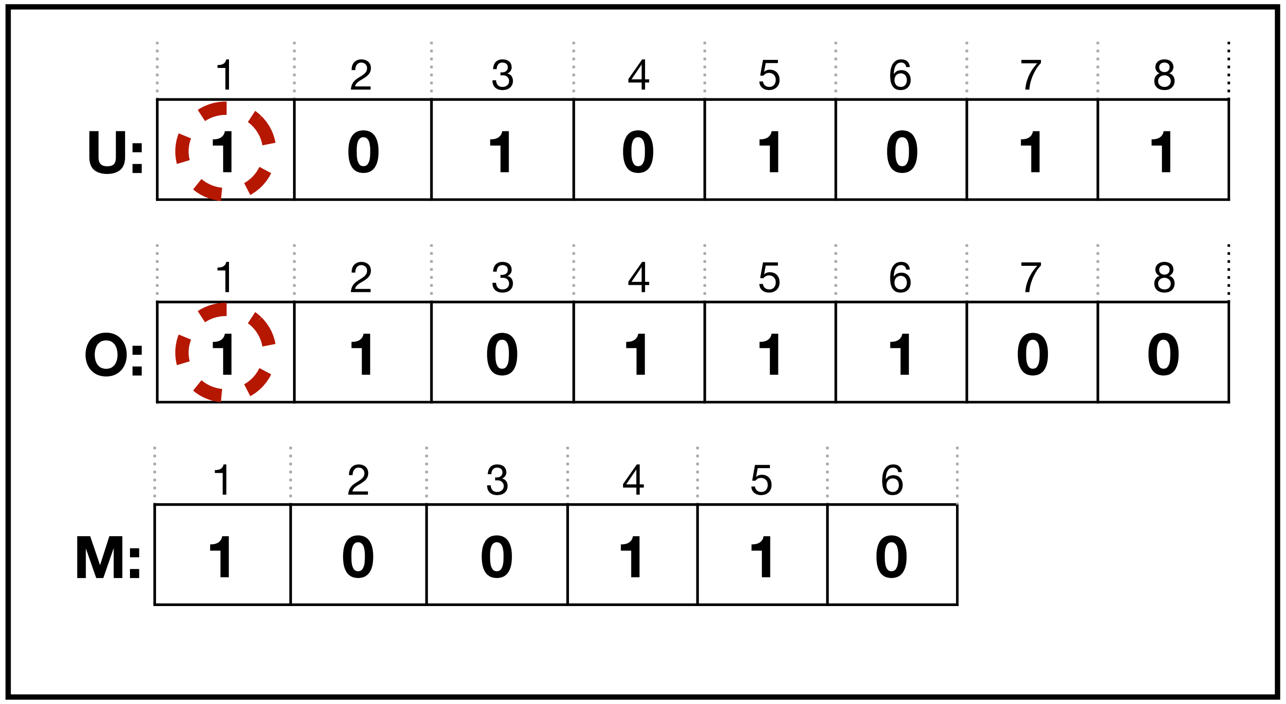
zombit-vector



	X_1		X_2		X_3		X_4		X_5		X_6		X_7		X_8	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
B:	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	0
	$\mathbb{O}$		$\mathbb{M}$		$\mathbb{Z}$		$\mathbb{M}$		$\mathbb{O}$		$\mathbb{M}$		$\mathbb{Z}$		$\mathbb{Z}$	

$succ(2) \mapsto$

zombit-vector

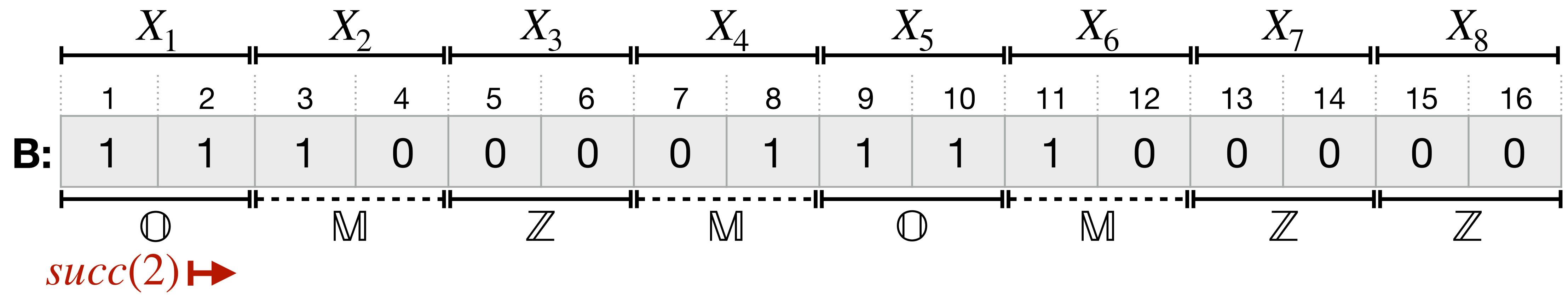


	1	2	3	4	5	6	7	8
U:	1	0	1	0	1	0	1	1

	1	2	3	4	5	6	7	8
O:	1	1	0	1	1	1	0	0

	1	2	3	4	5	6
M:	1	0	0	1	1	0

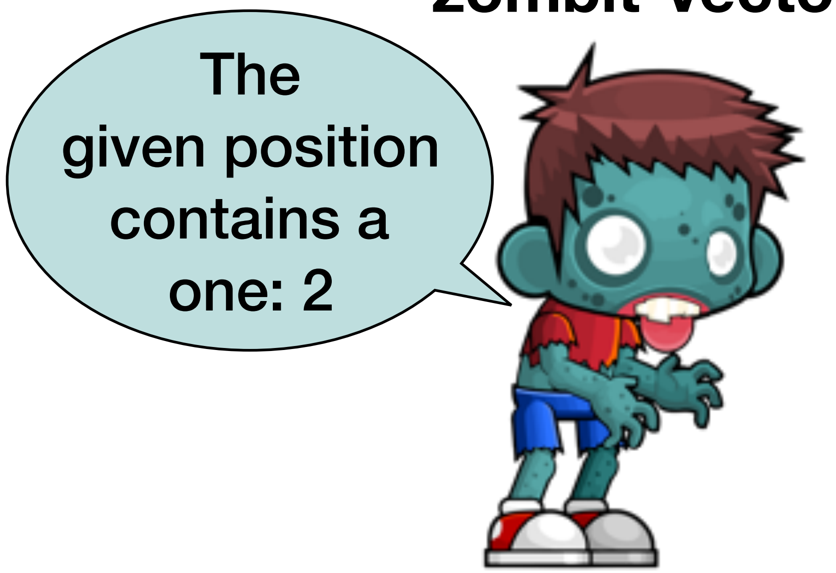
zombit-vector



	X_1		X_2		X_3		X_4		X_5		X_6		X_7		X_8	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
B:	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	0
	$\mathbb{O}$		$\mathbb{M}$		$\mathbb{Z}$		$\mathbb{M}$		$\mathbb{O}$		$\mathbb{M}$		$\mathbb{Z}$		$\mathbb{Z}$	

$succ(2) \mapsto$

zombit-vector

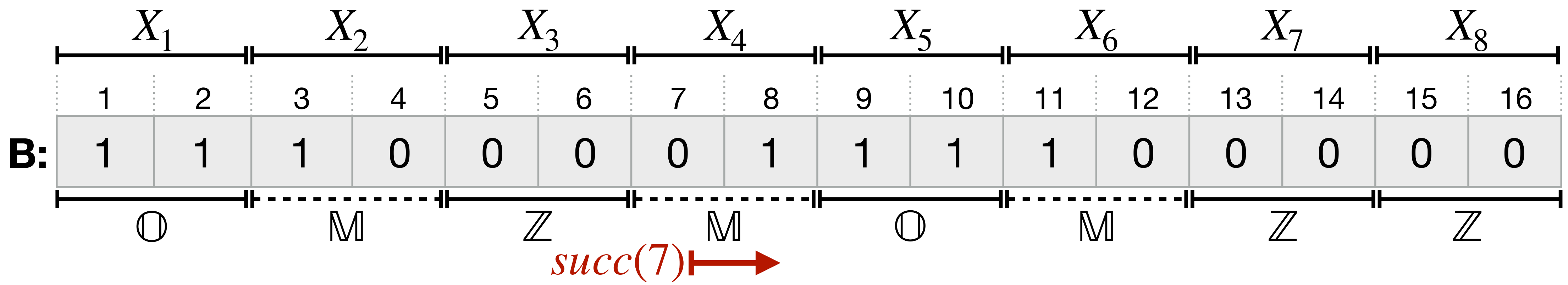


	1	2	3	4	5	6	7	8
U:	1	0	1	0	1	0	1	1

	1	2	3	4	5	6	7	8
O:	1	1	0	1	1	1	0	0

	1	2	3	4	5	6
M:	1	0	0	1	1	0

zombit-vector



	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8
	$\mathbb{O}$	$\mathbb{M}$	$\mathbb{Z}$	$\mathbb{M}$	$\mathbb{O}$	$\mathbb{M}$	$\mathbb{Z}$	$\mathbb{Z}$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
B:	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	0

$succ(7)$

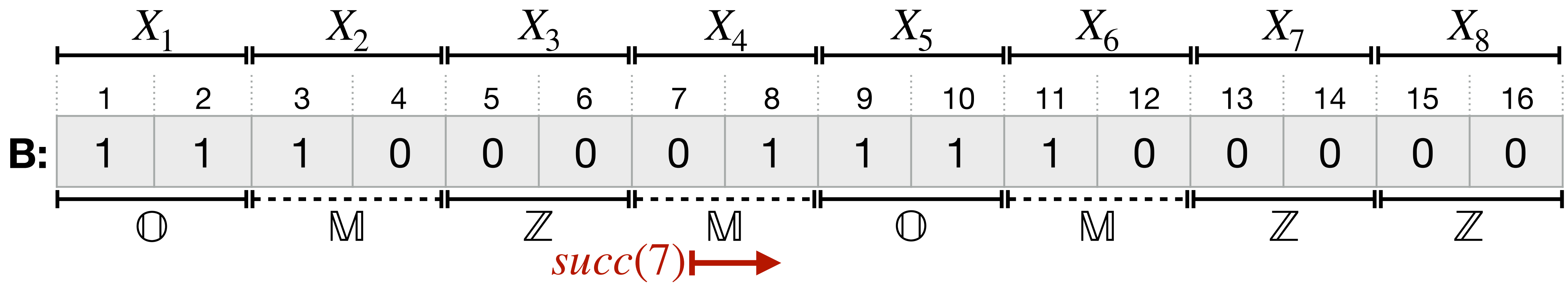
zombit-vector

	1	2	3	4	5	6	7	8
U:	1	0	1	0	1	0	1	1

	1	2	3	4	5	6	7	8
O:	1	1	0	1	1	1	0	0

	1	2	3	4	5	6
M:	1	0	0	1	1	0

zombit-vector



	X_1		X_2		X_3		X_4		X_5		X_6		X_7		X_8	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
B:	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	0
	$\mathbb{O}$		$\mathbb{M}$		$\mathbb{Z}$		$\mathbb{M}$		$\mathbb{O}$		$\mathbb{M}$		$\mathbb{Z}$		$\mathbb{Z}$	

$\mathit{succ}(7) \mapsto$

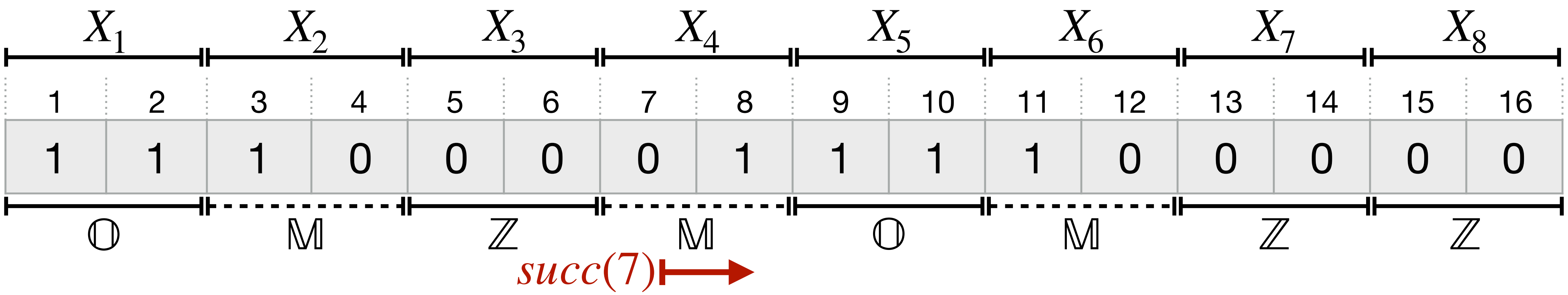
zombit-vector

	1	2	3	4	5	6	7	8
U:	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**1**

	1	2	3	4	5	6	7	8
O:	**1**	**1**	**0**	**1**	**1**	**1**	**0**	**0**

	1	2	3	4	5	6
M:	**1**	**0**	**0**	**1**	**1**	**0**

zombit-vector

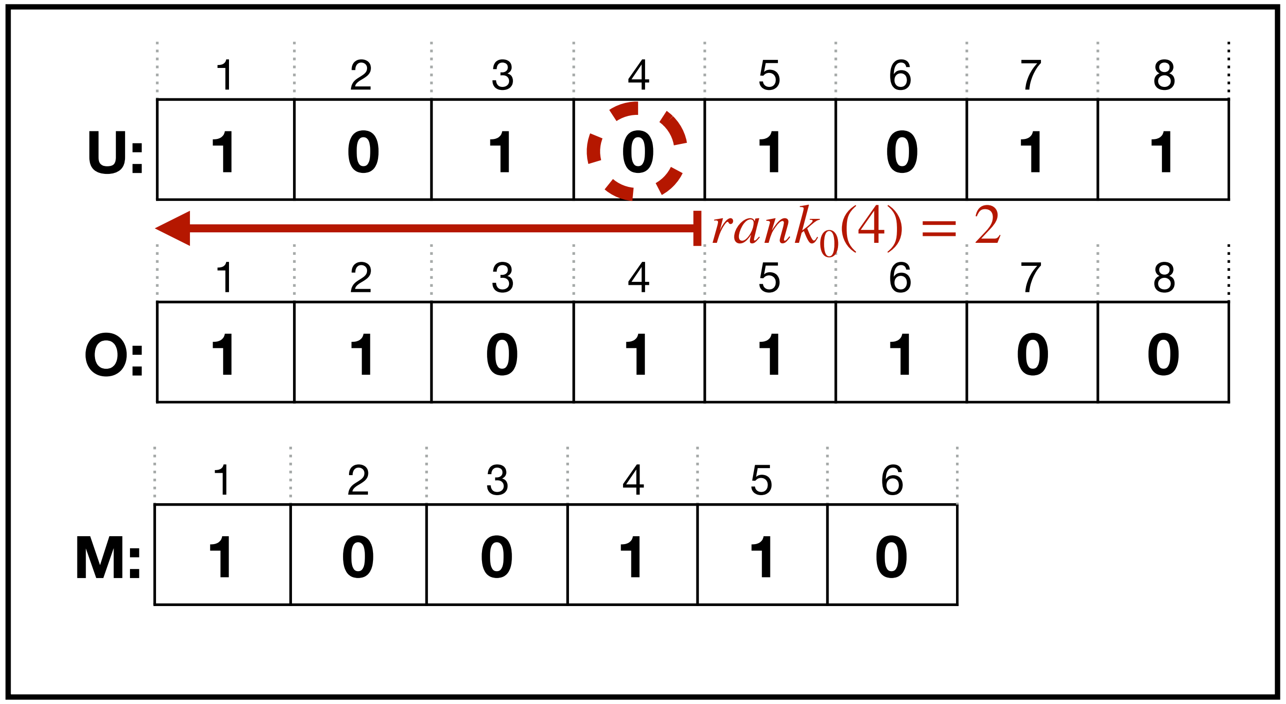


	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8
	$\mathbb{O}$	$\mathbb{M}$	$\mathbb{Z}$	$\mathbb{M}$	$\mathbb{O}$	$\mathbb{M}$	$\mathbb{Z}$	$\mathbb{Z}$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
B:	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	0

$succ(7) \mapsto$

zombit-vector



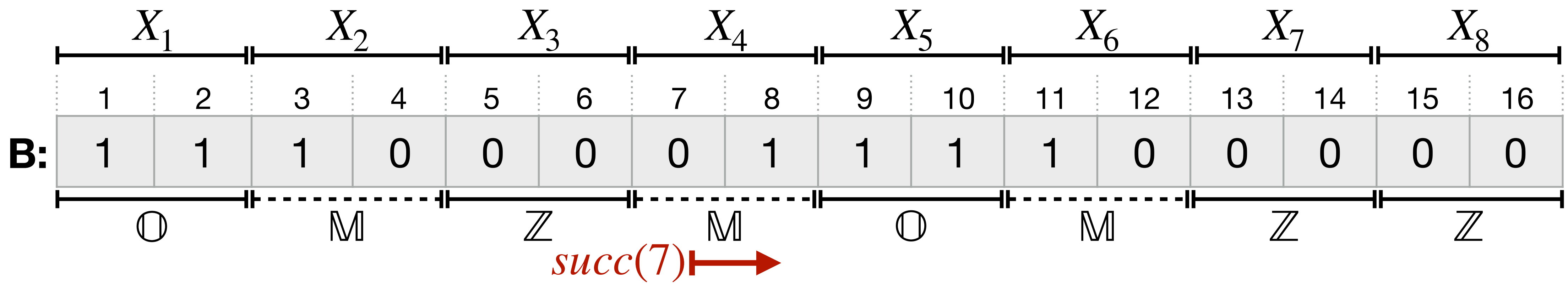
	1	2	3	4	5	6	7	8
U:	1	0	1	0	1	0	1	1

$rank_0(4) = 2$

	1	2	3	4	5	6	7	8
O:	1	1	0	1	1	1	0	0

	1	2	3	4	5	6
M:	1	0	0	1	1	0

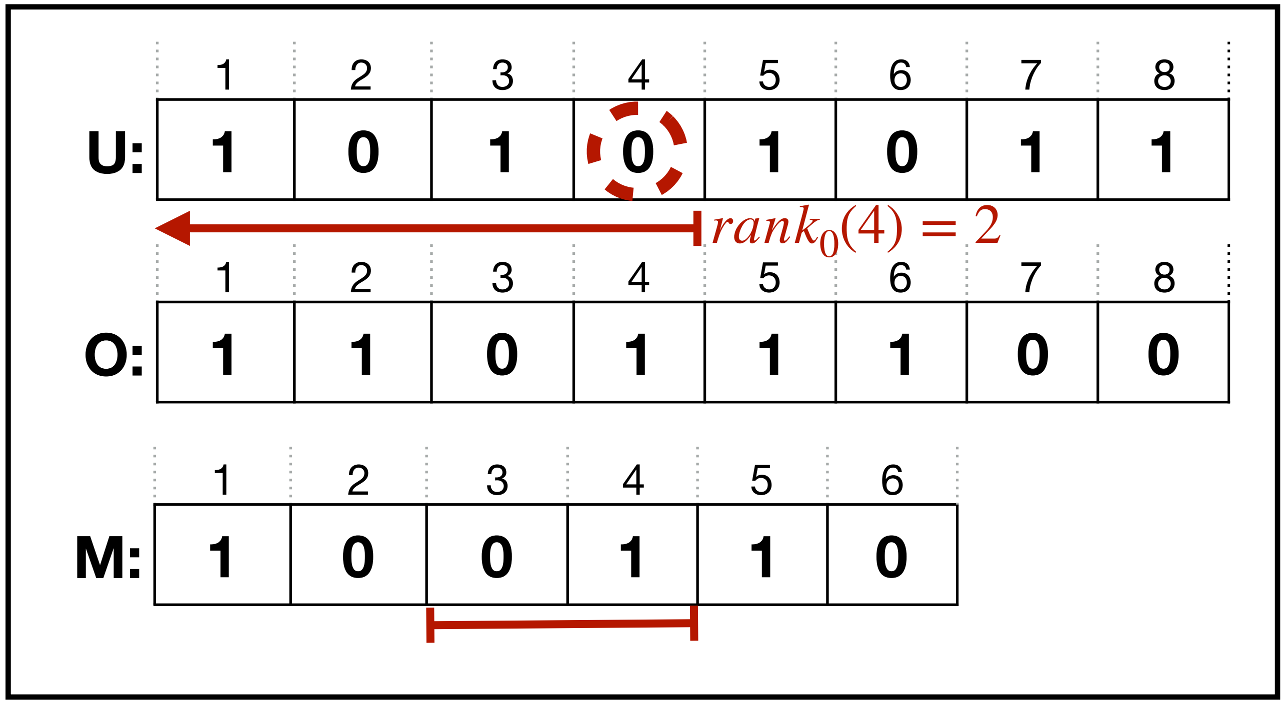
zombit-vector



	X_1		X_2		X_3		X_4		X_5		X_6		X_7		X_8	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
B:	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	0
	$\mathbb{O}$		$\mathbb{M}$		$\mathbb{Z}$		$\mathbb{M}$		$\mathbb{O}$		$\mathbb{M}$		$\mathbb{Z}$		$\mathbb{Z}$	

$succ(7)$

zombit-vector



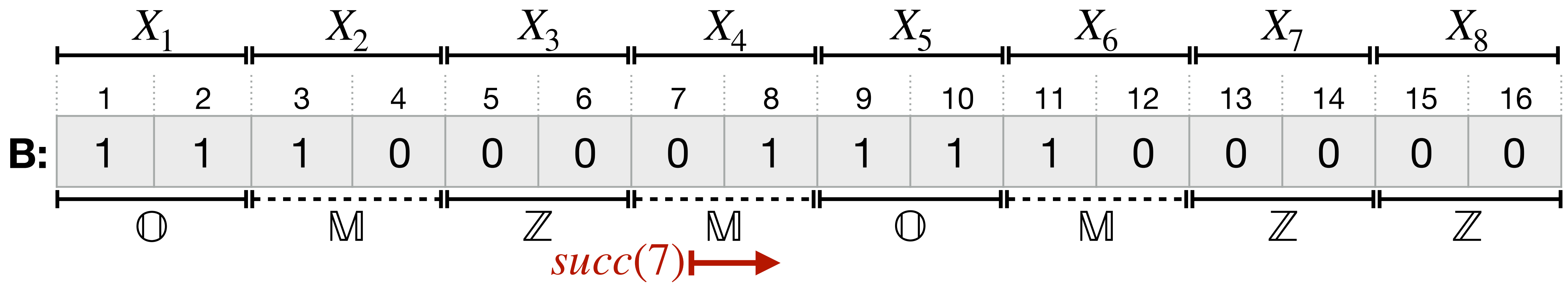
	1	2	3	4	5	6	7	8
U:	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**1**

$rank_0(4) = 2$

	1	2	3	4	5	6	7	8
O:	**1**	**1**	**0**	**1**	**1**	**1**	**0**	**0**

	1	2	3	4	5	6
M:	**1**	**0**	**0**	**1**	**1**	**0**

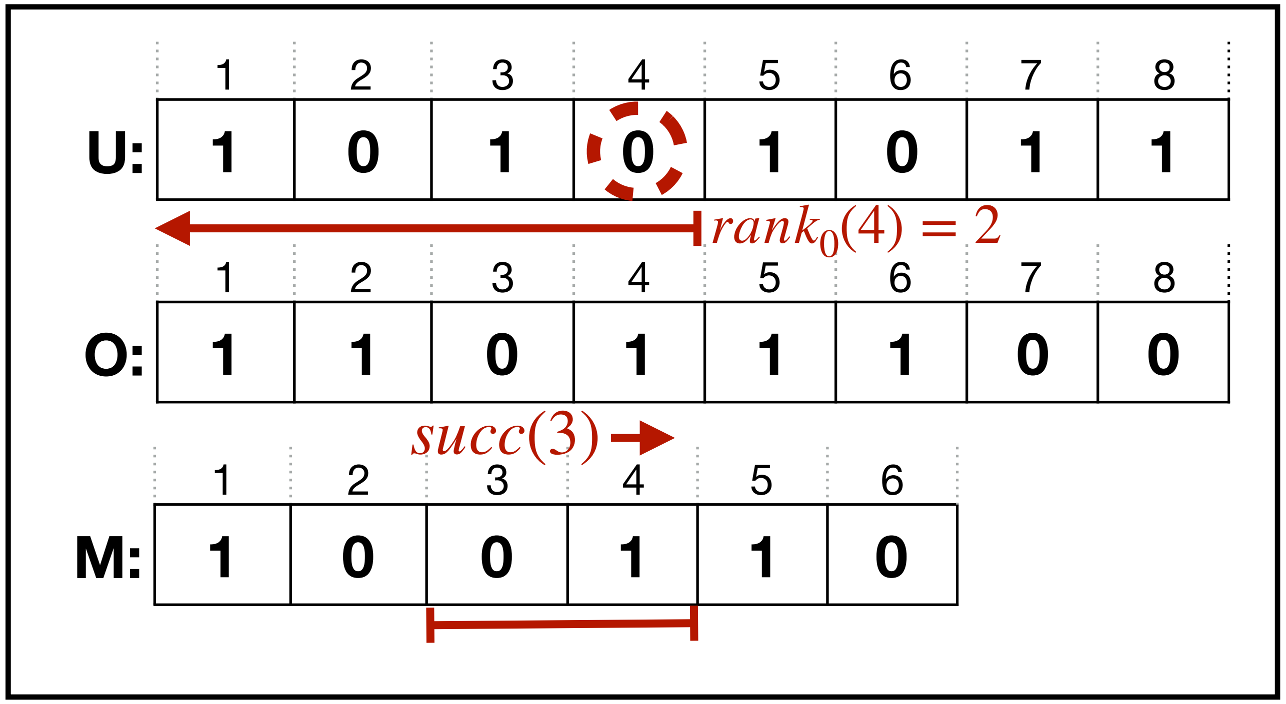
zombit-vector



	X_1		X_2		X_3		X_4		X_5		X_6		X_7		X_8	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
B:	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	0
	$\mathbb{O}$		$\mathbb{M}$		$\mathbb{Z}$		$\mathbb{M}$		$\mathbb{O}$		$\mathbb{M}$		$\mathbb{Z}$		$\mathbb{Z}$	

$succ(7)$

zombit-vector



	1	2	3	4	5	6	7	8
U:	1	0	1	0	1	0	1	1

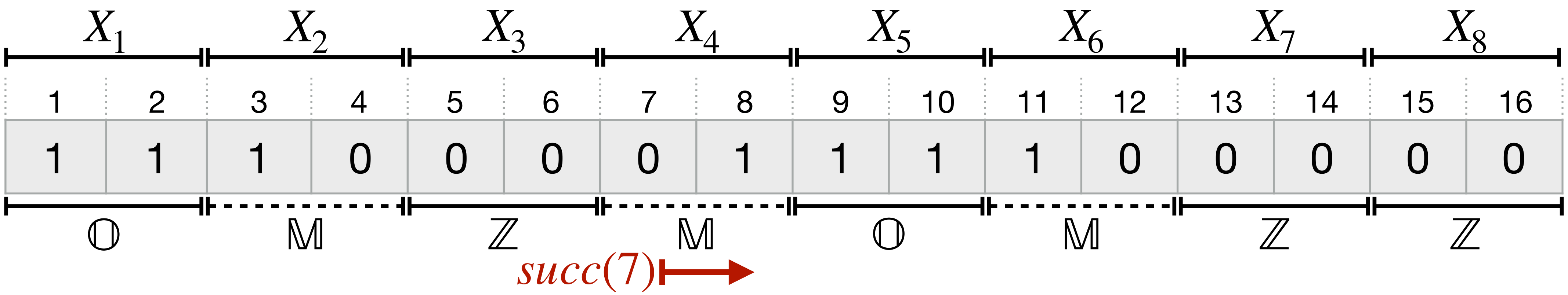
$rank_0(4) = 2$

	1	2	3	4	5	6	7	8
O:	1	1	0	1	1	1	0	0

$succ(3)$

	1	2	3	4	5	6
M:	1	0	0	1	1	0

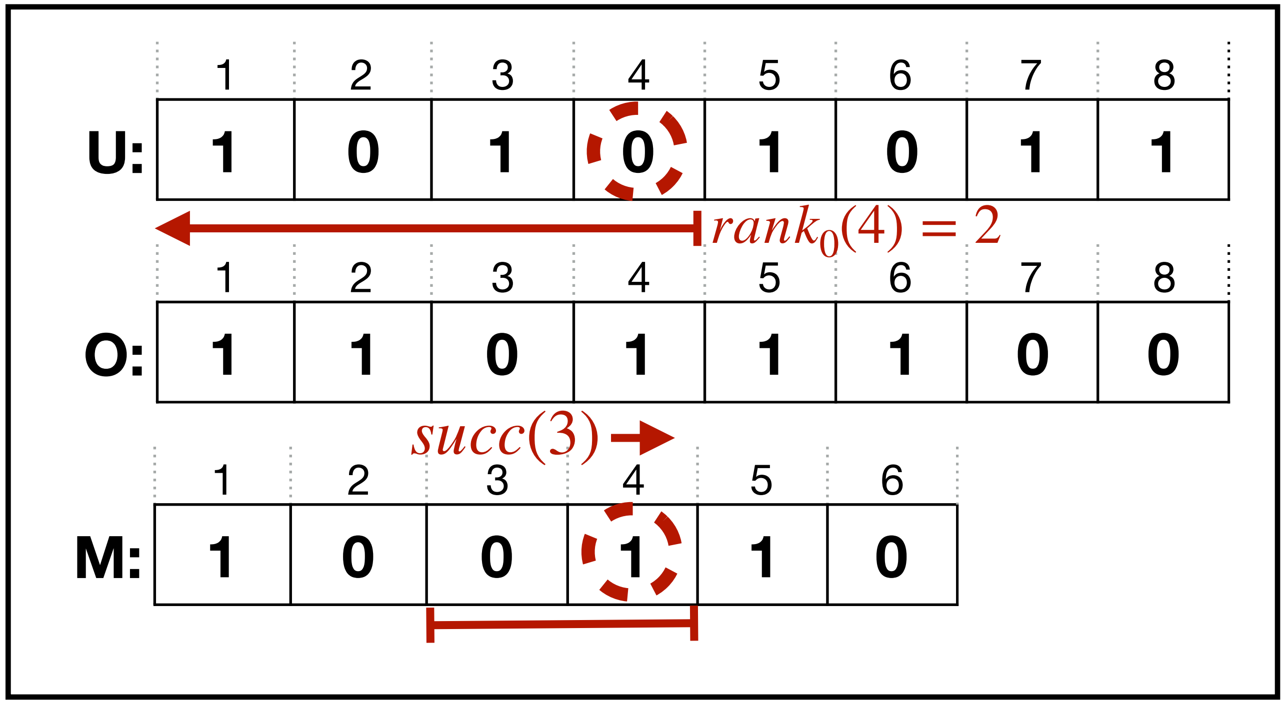
zombit-vector



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
B:	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	0

$succ(7) \mapsto$

zombit-vector



	1	2	3	4	5	6	7	8
U:	1	0	1	0	1	0	1	1

$rank_0(4) = 2$

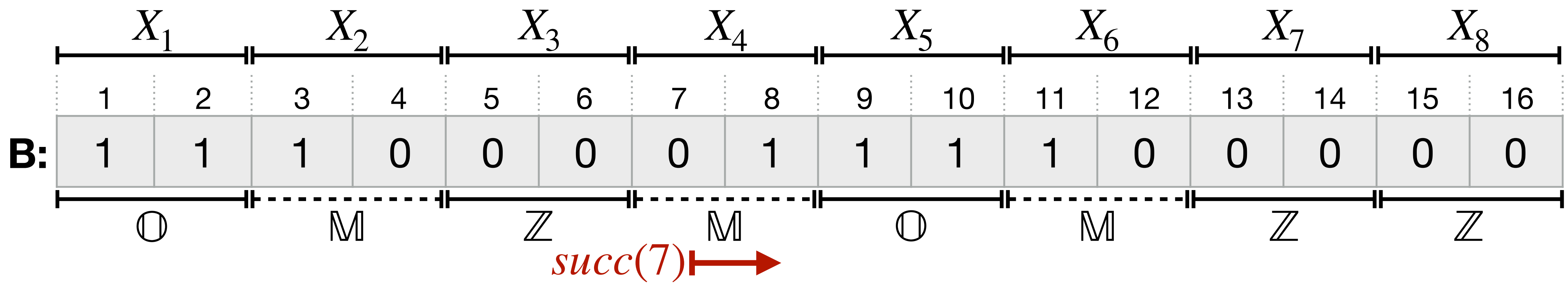
	1	2	3	4	5	6	7	8
O:	1	1	0	1	1	1	0	0

$succ(3) \rightarrow$

	1	2	3	4	5	6
M:	1	0	0	1	1	0

Data
Compression
Conference

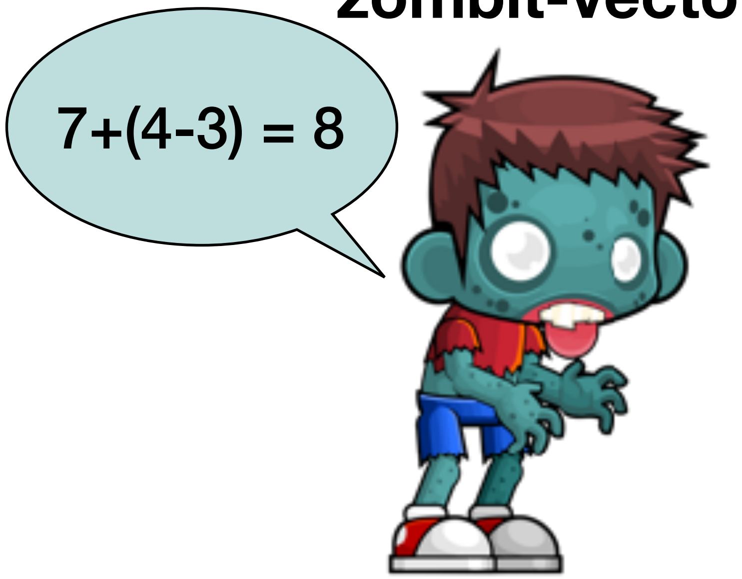
zombit-vector

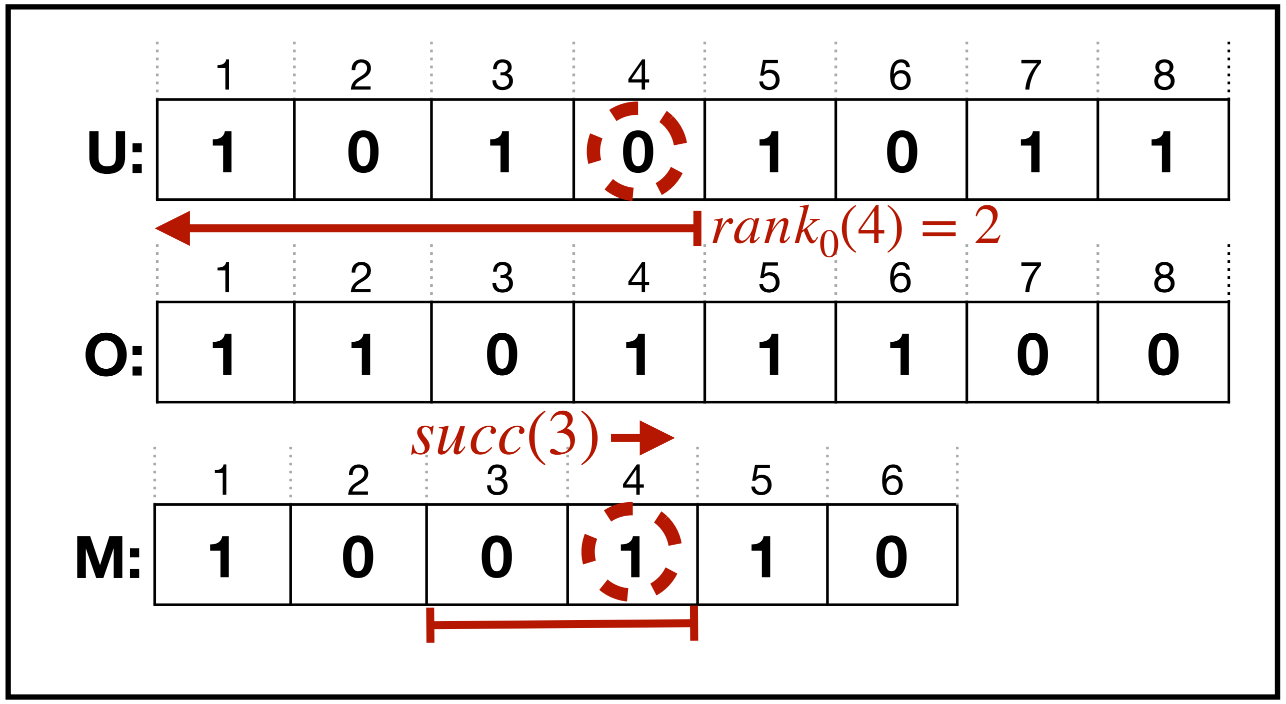


	X_1		X_2		X_3		X_4		X_5		X_6		X_7		X_8	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
B:	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	0
	$\mathbb{O}$		$\mathbb{M}$		$\mathbb{Z}$		$\mathbb{M}$		$\mathbb{O}$		$\mathbb{M}$		$\mathbb{Z}$		$\mathbb{Z}$	

$succ(7)$

zombit-vector





	1	2	3	4	5	6	7	8
U:	1	0	1	0	1	0	1	1

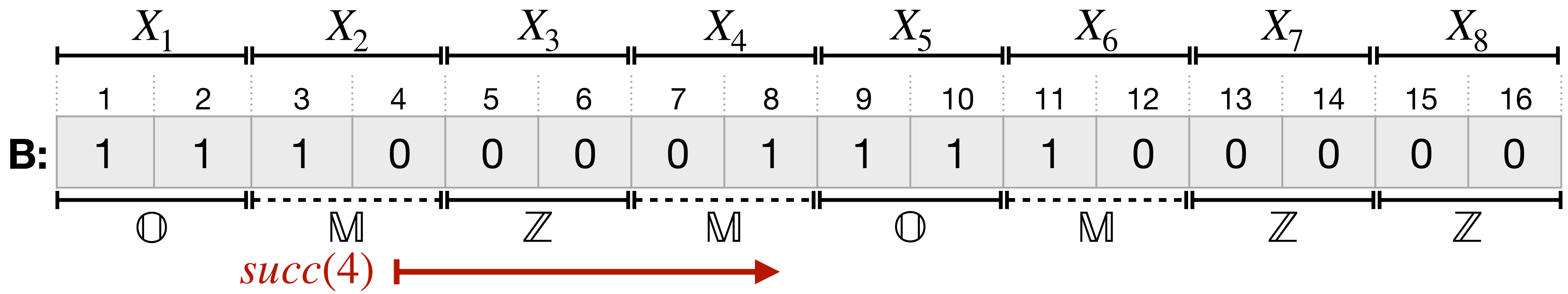
$rank_0(4) = 2$

	1	2	3	4	5	6	7	8
O:	1	1	0	1	1	1	0	0

$succ(3)$

	1	2	3	4	5	6
M:	1	0	0	1	1	0

zombit-vector



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
B:	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	0

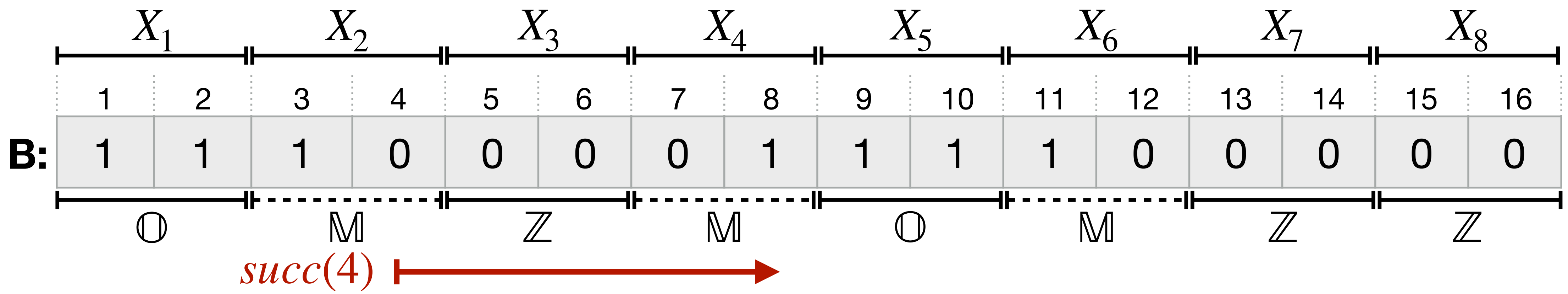
zombit-vector

	1	2	3	4	5	6	7	8
U:	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**1**

	1	2	3	4	5	6	7	8
O:	**1**	**1**	**0**	**1**	**1**	**1**	**0**	**0**

	1	2	3	4	5	6
M:	**1**	**0**	**0**	**1**	**1**	**0**

zombit-vector

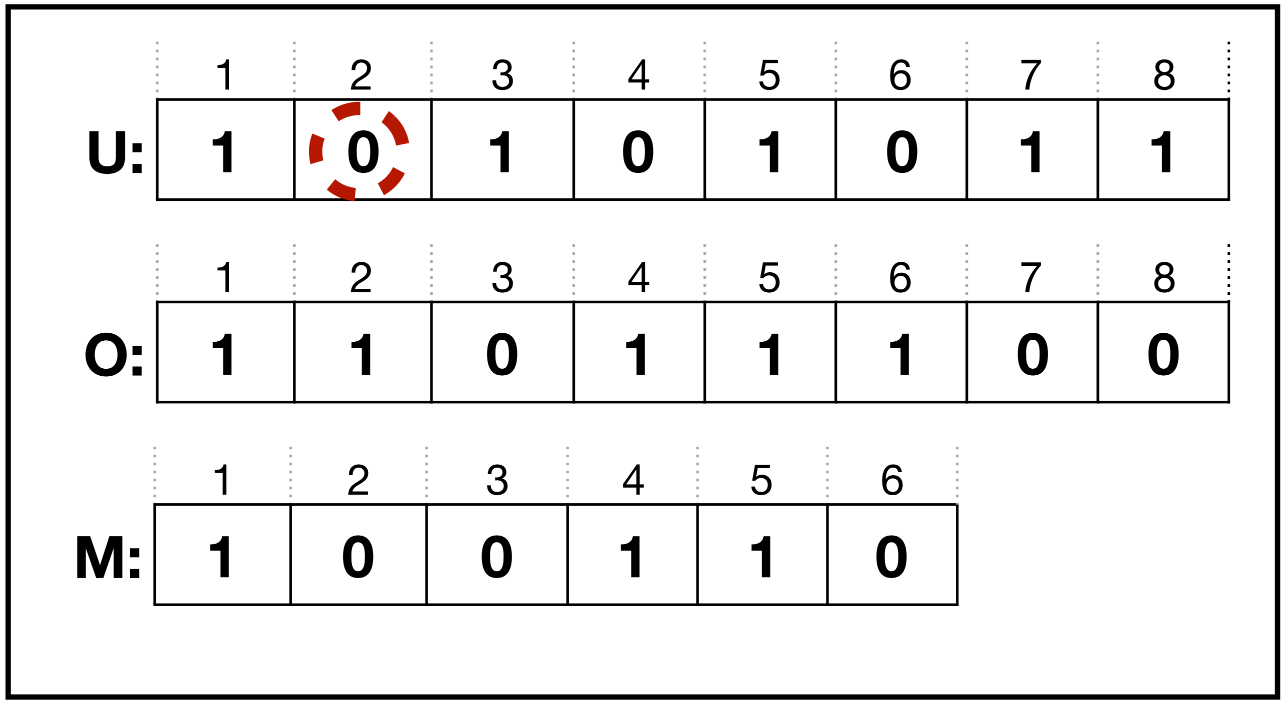


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
B:	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	0

X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8
$\mathbb{O}$	$\mathbb{M}$	$\mathbb{Z}$	$\mathbb{M}$	$\mathbb{O}$	$\mathbb{M}$	$\mathbb{Z}$	$\mathbb{Z}$

$succ(4)$

zombit-vector

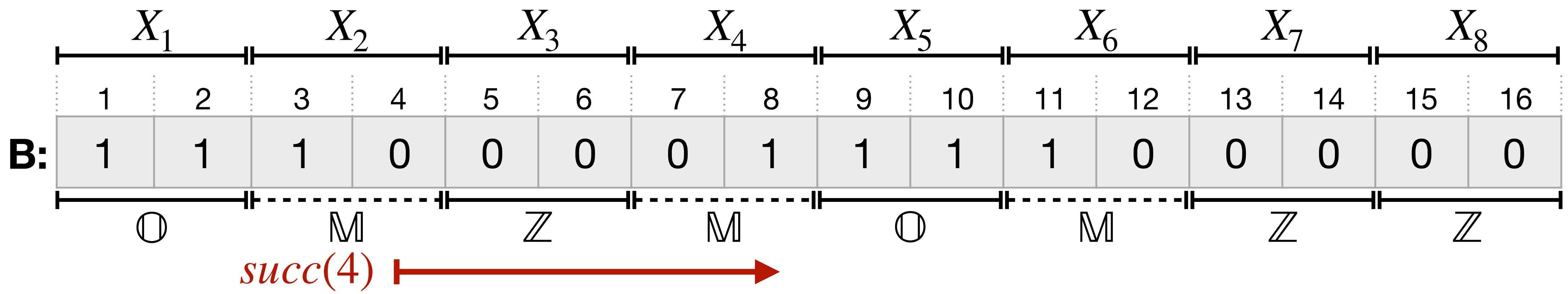


	1	2	3	4	5	6	7	8
U:	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**1**

	1	2	3	4	5	6	7	8
O:	**1**	**1**	**0**	**1**	**1**	**1**	**0**	**0**

	1	2	3	4	5	6
M:	**1**	**0**	**0**	**1**	**1**	**0**

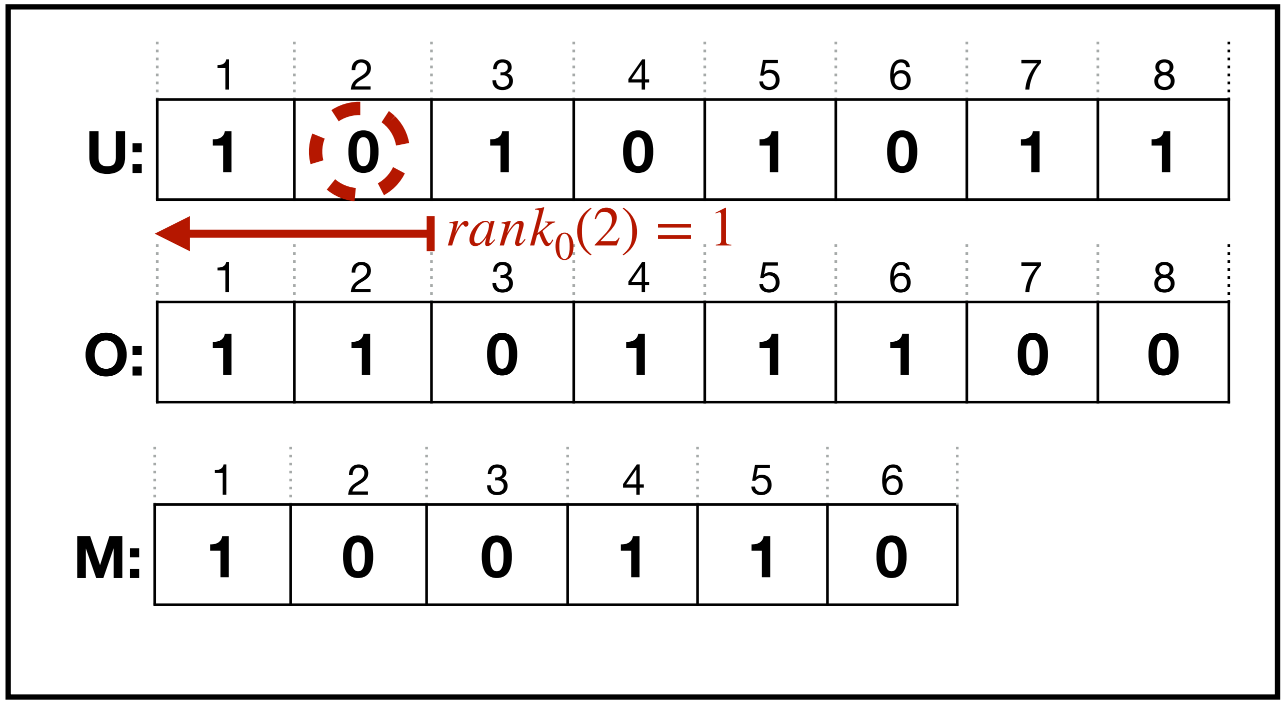
zombit-vector



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
B:	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	0

$succ(4)$

zombit-vector



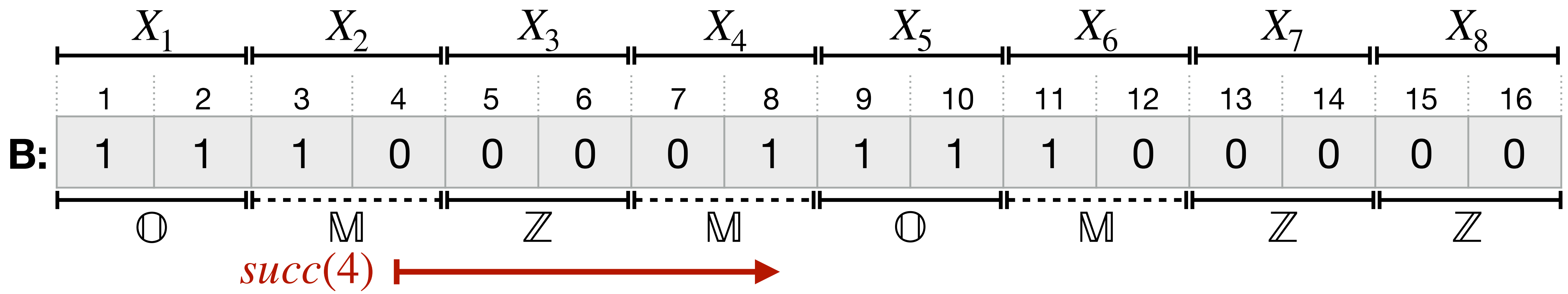
	1	2	3	4	5	6	7	8
U:	1	0	1	0	1	0	1	1

$rank_0(2) = 1$

	1	2	3	4	5	6	7	8
O:	1	1	0	1	1	1	0	0

	1	2	3	4	5	6
M:	1	0	0	1	1	0

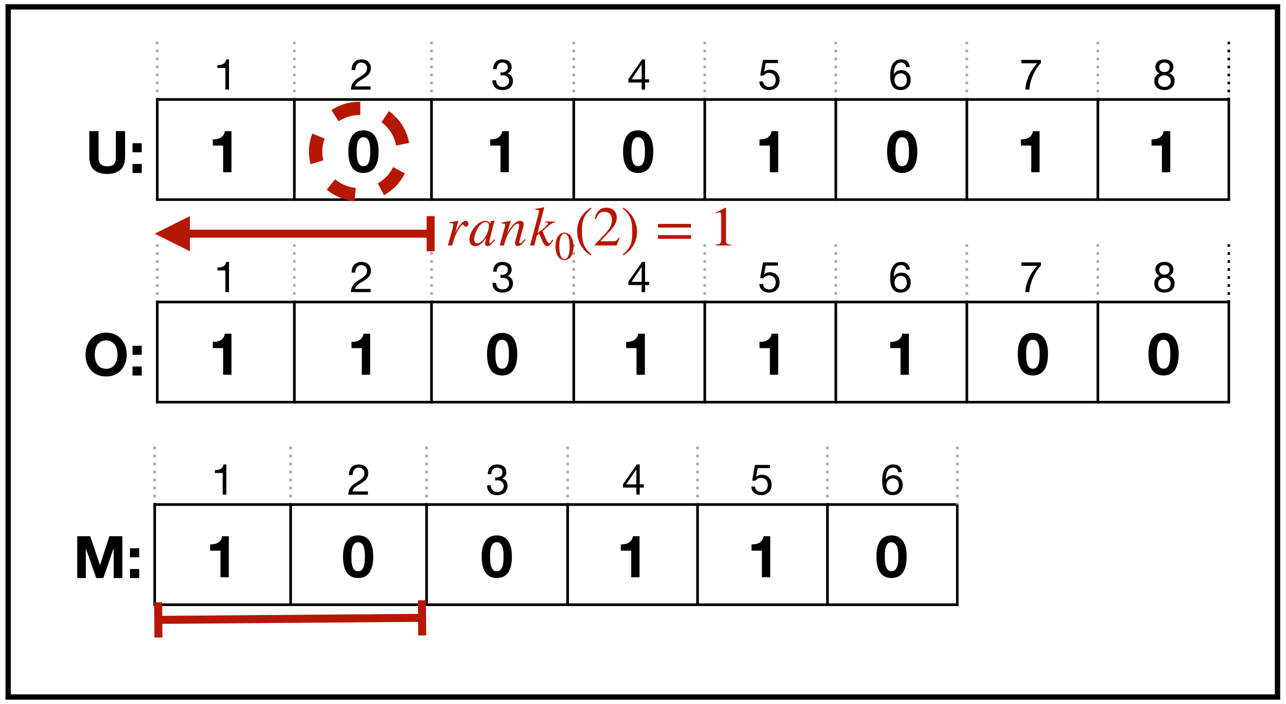
zombit-vector



	X_1		X_2		X_3		X_4		X_5		X_6		X_7		X_8	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
B:	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	0
	$\mathbb{O}$		$\mathbb{M}$		$\mathbb{Z}$		$\mathbb{M}$		$\mathbb{O}$		$\mathbb{M}$		$\mathbb{Z}$		$\mathbb{Z}$	

$succ(4)$

zombit-vector



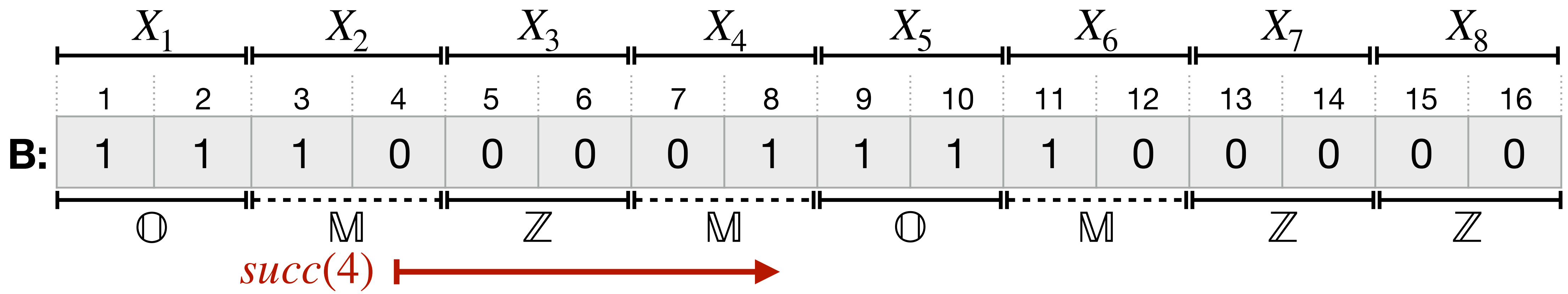
	1	2	3	4	5	6	7	8
U:	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**1**

$rank_0(2) = 1$

	1	2	3	4	5	6	7	8
O:	**1**	**1**	**0**	**1**	**1**	**1**	**0**	**0**

	1	2	3	4	5	6
M:	**1**	**0**	**0**	**1**	**1**	**0**

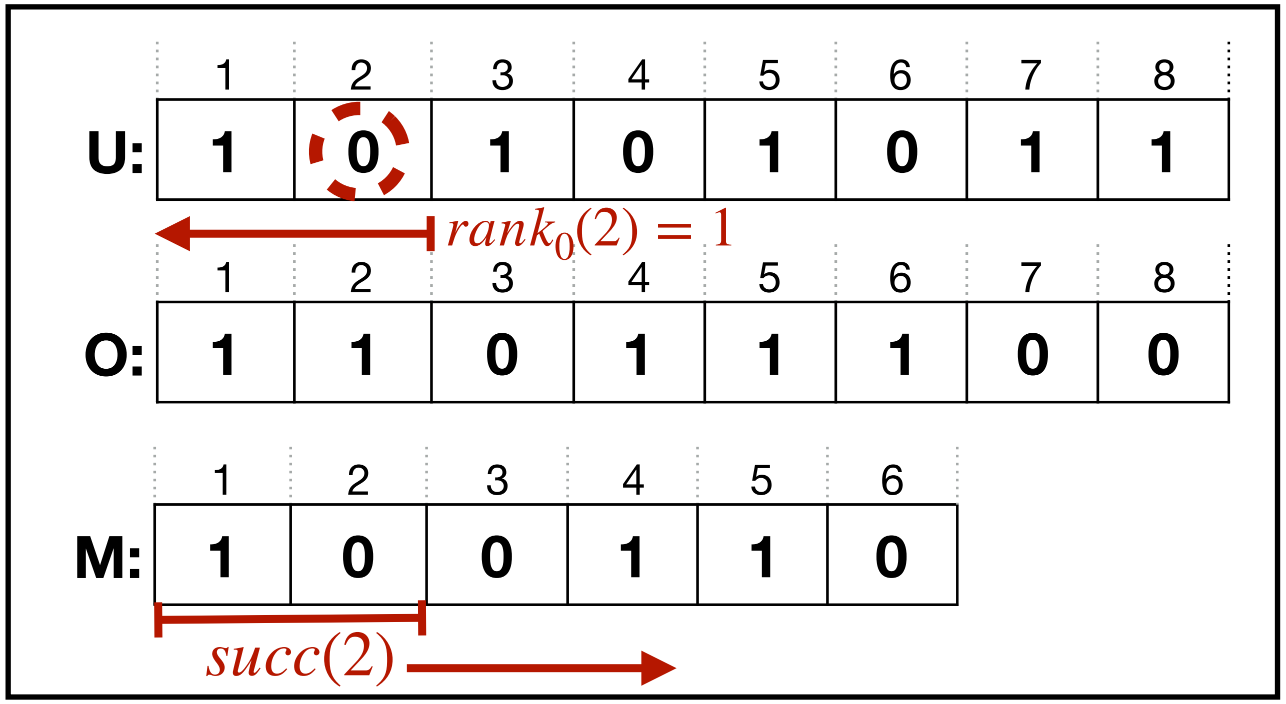
zombit-vector



	X_1		X_2		X_3		X_4		X_5		X_6		X_7		X_8	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
B:	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	0
	$\mathbb{O}$		$\mathbb{M}$		$\mathbb{Z}$		$\mathbb{M}$		$\mathbb{O}$		$\mathbb{M}$		$\mathbb{Z}$		$\mathbb{Z}$	

$succ(4)$

zombit-vector



	1	2	3	4	5	6	7	8
U:	1	0	1	0	1	0	1	1

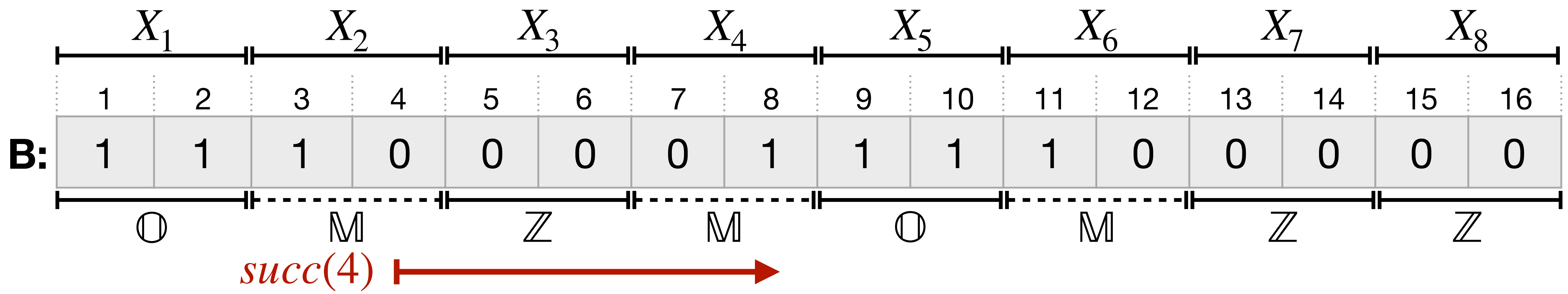
$rank_0(2) = 1$

	1	2	3	4	5	6	7	8
O:	1	1	0	1	1	1	0	0

	1	2	3	4	5	6
M:	1	0	0	1	1	0

$succ(2)$

zombit-vector

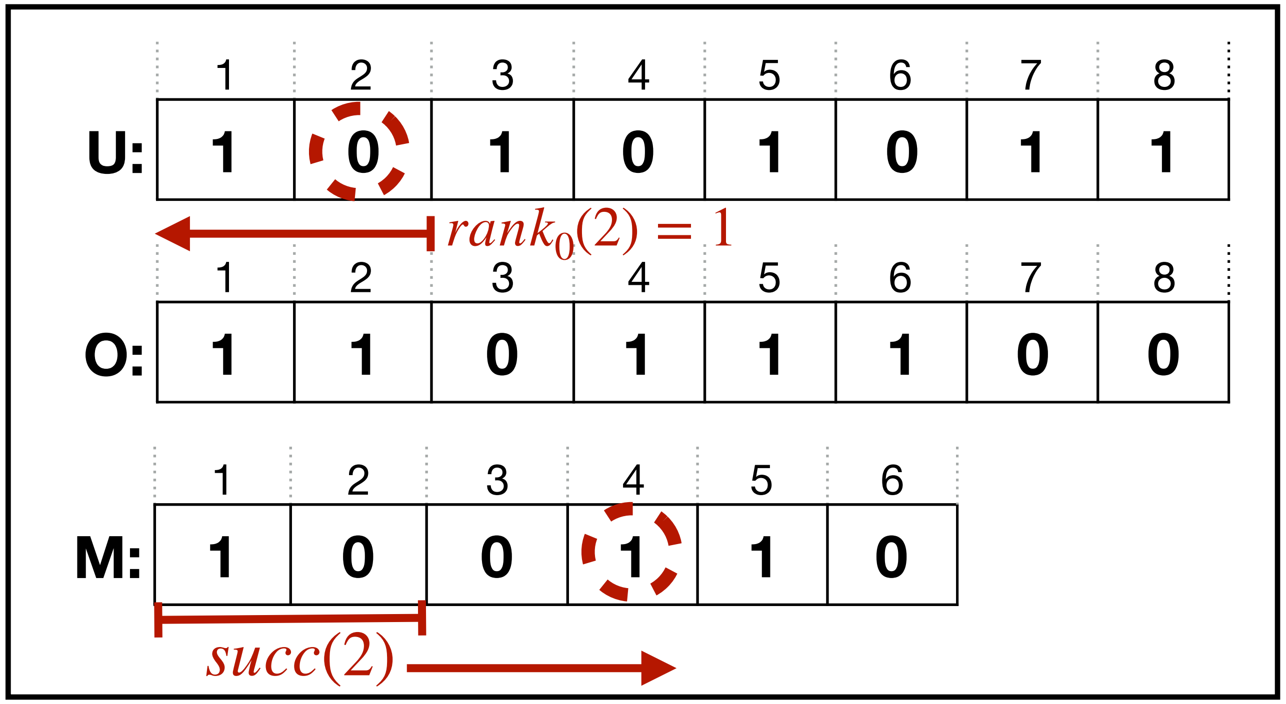


X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8
1 2	3 4	5 6	7 8	9 10	11 12	13 14	15 16
1 1	1 0	0 0	0 1	1 1	1 0	0 0	0 0
𝕆	𝕄	ℤ	𝕄	𝕆	𝕄	ℤ	ℤ

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
B:	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	0

$succ(4)$

zombit-vector



	1	2	3	4	5	6	7	8
U:	1	0	1	0	1	0	1	1

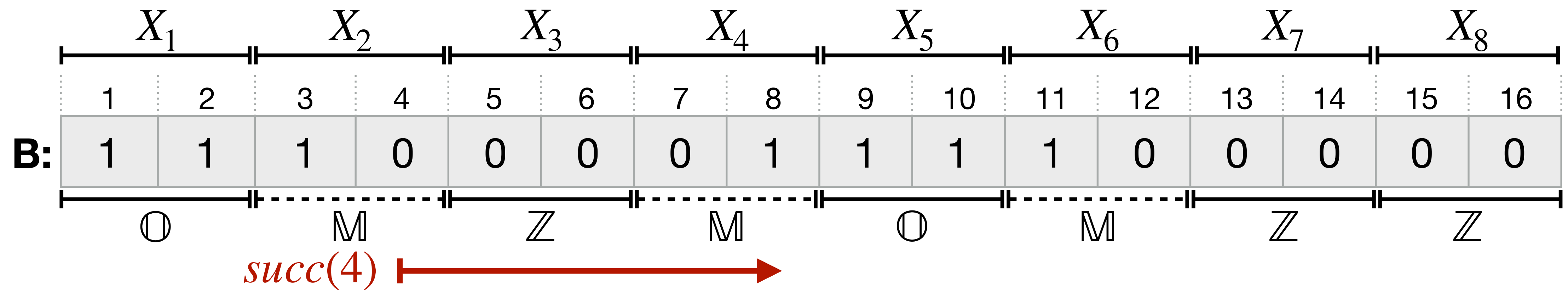
$rank_0(2) = 1$

	1	2	3	4	5	6	7	8
O:	1	1	0	1	1	1	0	0

	1	2	3	4	5	6
M:	1	0	0	1	1	0

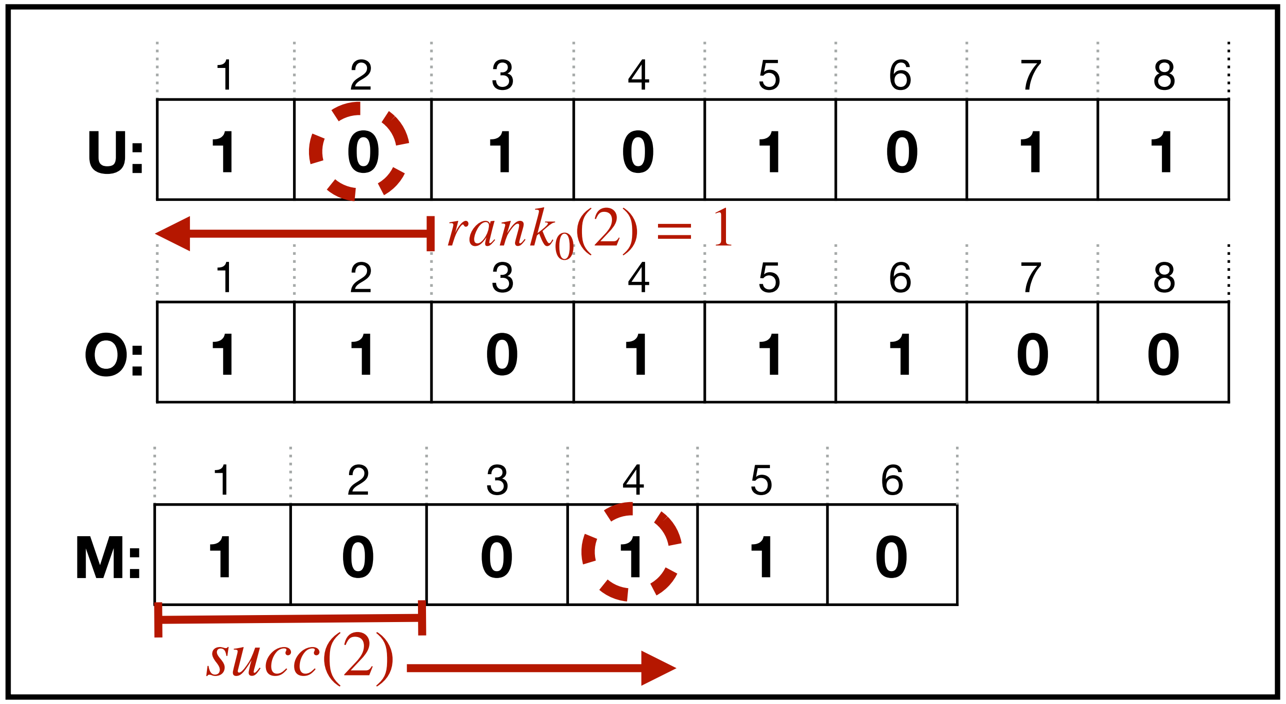
$succ(2)$

zombit-vector



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
B:	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	0

zombit-vector

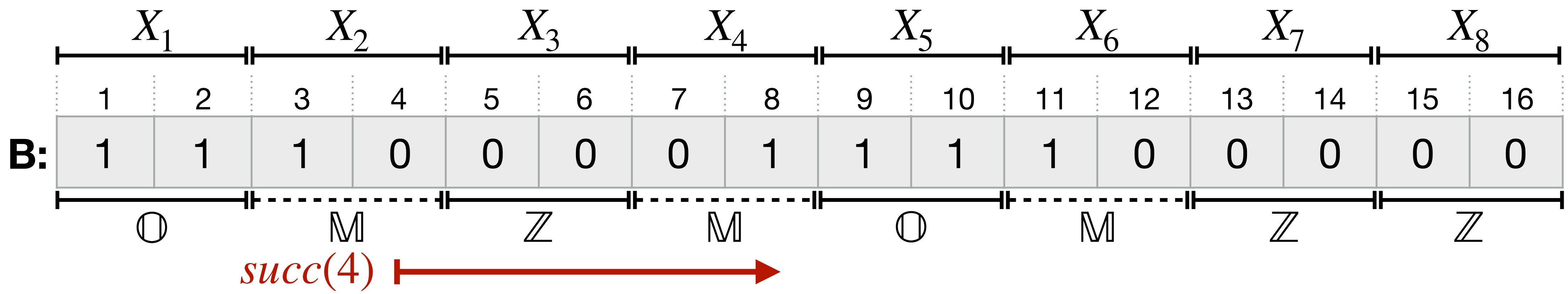


	1	2	3	4	5	6	7	8
U:	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**1**

	1	2	3	4	5	6	7	8
O:	**1**	**1**	**0**	**1**	**1**	**1**	**0**	**0**

	1	2	3	4	5	6
M:	**1**	**0**	**0**	**1**	**1**	**0**

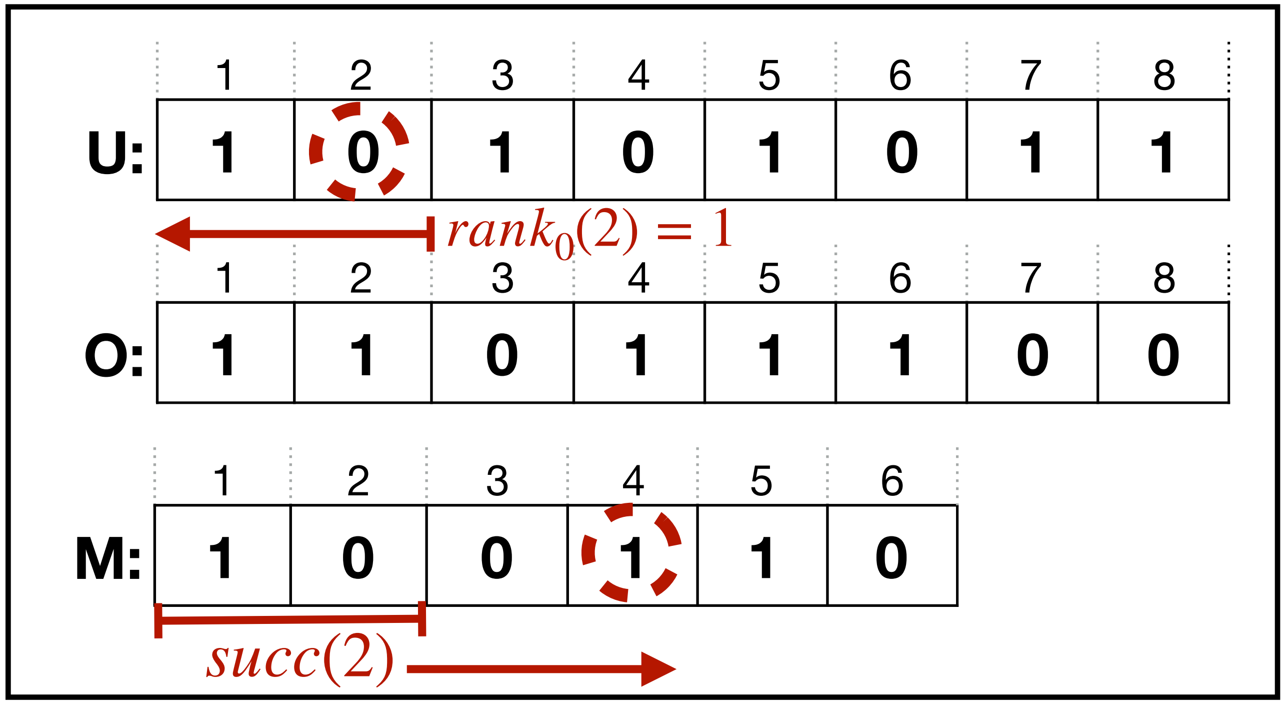
zombit-vector



	X_1		X_2		X_3		X_4		X_5		X_6		X_7		X_8	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
B:	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	0
	$\mathbb{O}$		$\mathbb{M}$		$\mathbb{Z}$		$\mathbb{M}$		$\mathbb{O}$		$\mathbb{M}$		$\mathbb{Z}$		$\mathbb{Z}$	

$succ(4)$

zombit-vector



	1	2	3	4	5	6	7	8
U:	1	0	1	0	1	0	1	1

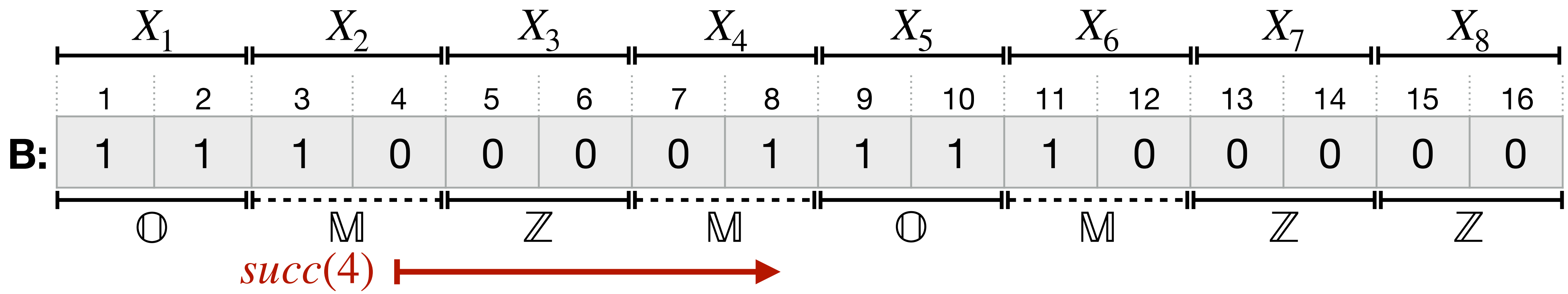
$rank_0(2) = 1$

	1	2	3	4	5	6	7	8
O:	1	1	0	1	1	1	0	0

	1	2	3	4	5	6
M:	1	0	0	1	1	0

$succ(2)$

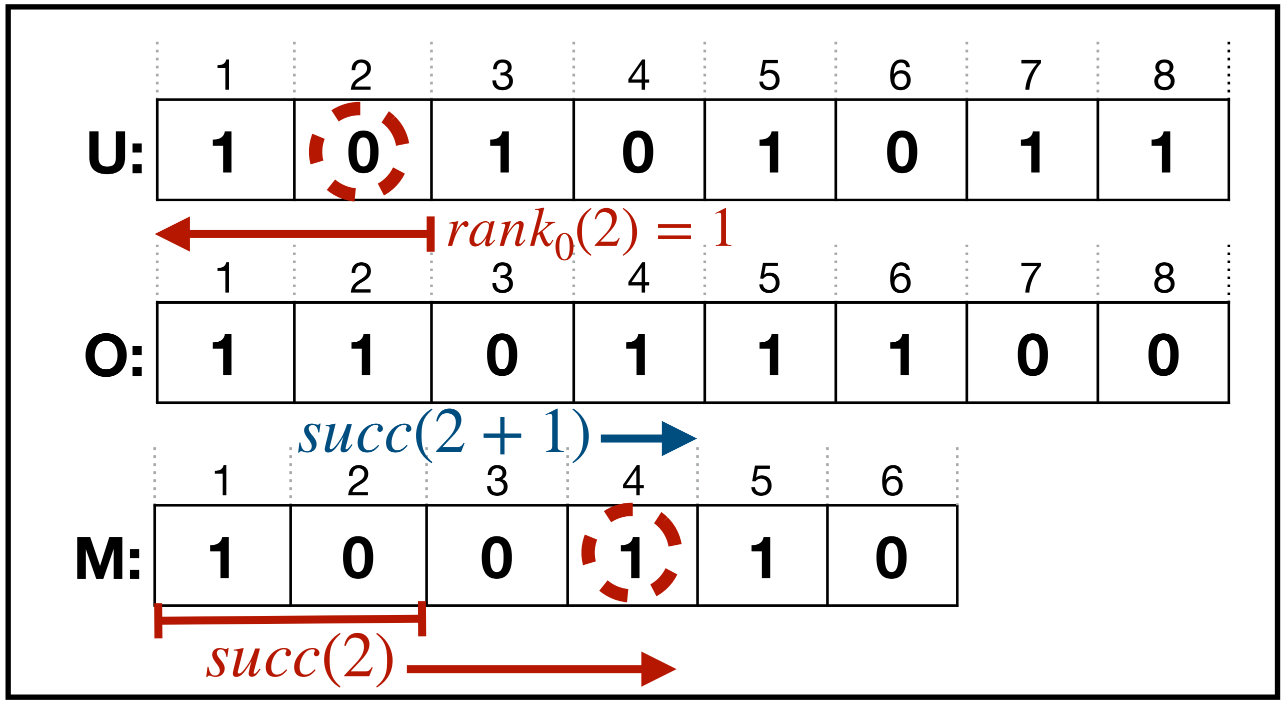
zombit-vector



	X_1		X_2		X_3		X_4		X_5		X_6		X_7		X_8	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
B:	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	0
	$\mathbb{O}$		$\mathbb{M}$		$\mathbb{Z}$		$\mathbb{M}$		$\mathbb{O}$		$\mathbb{M}$		$\mathbb{Z}$		$\mathbb{Z}$	

$succ(4)$

zombit-vector



	1	2	3	4	5	6	7	8
U:	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**1**

$rank_0(2) = 1$

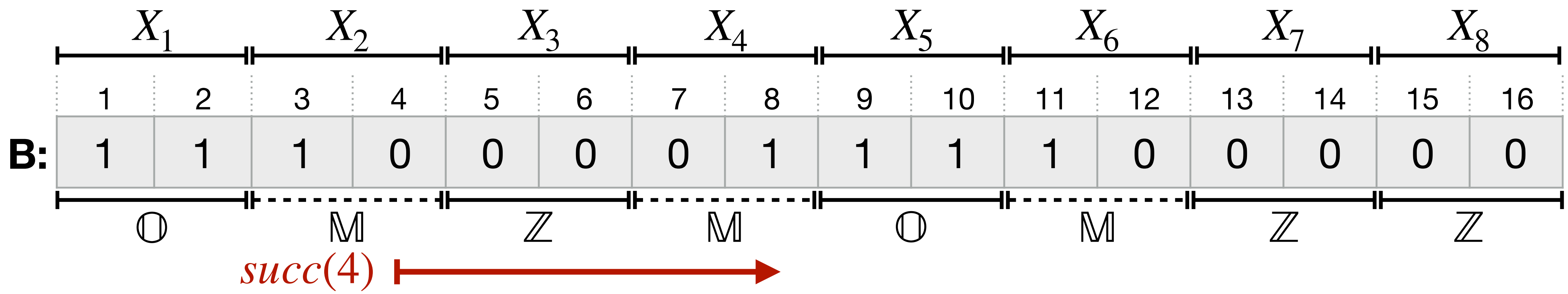
	1	2	3	4	5	6	7	8
O:	**1**	**1**	**0**	**1**	**1**	**1**	**0**	**0**

$succ(2 + 1)$

	1	2	3	4	5	6
M:	**1**	**0**	**0**	**1**	**1**	**0**

$succ(2)$

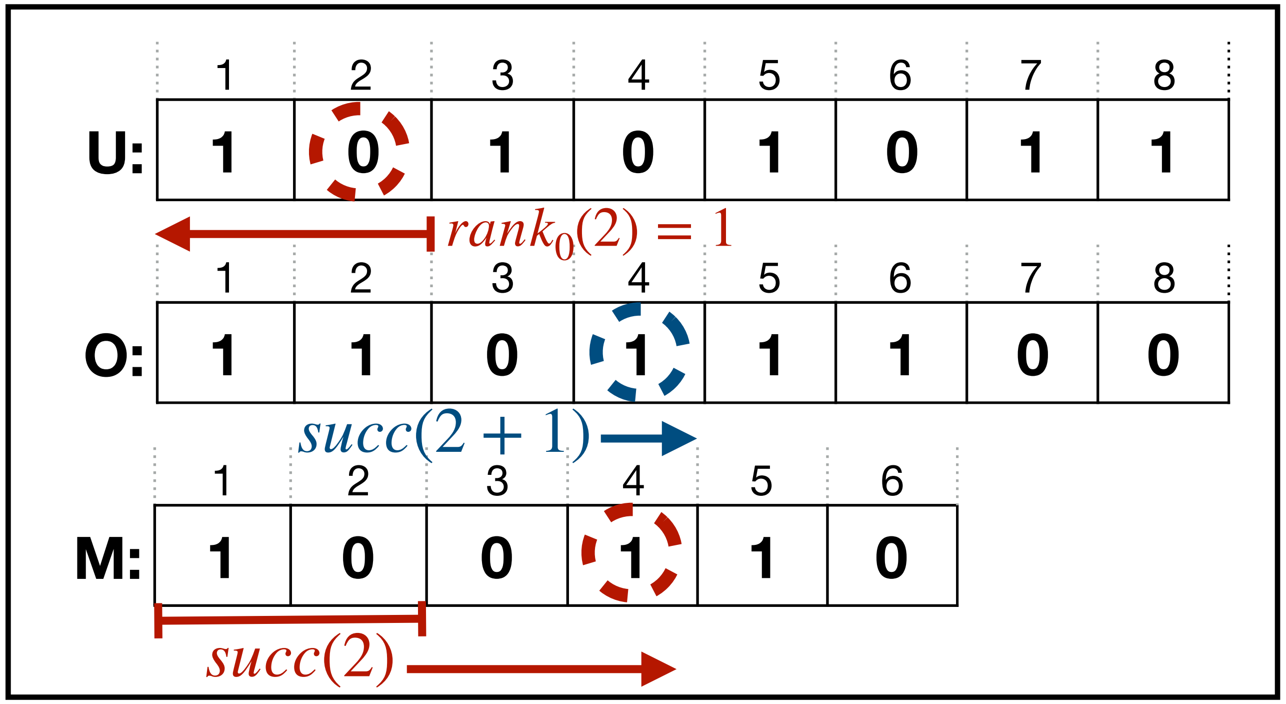
zombit-vector



	X_1		X_2		X_3		X_4		X_5		X_6		X_7		X_8	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
B:	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	0
	$\mathbb{O}$		$\mathbb{M}$		$\mathbb{Z}$		$\mathbb{M}$		$\mathbb{O}$		$\mathbb{M}$		$\mathbb{Z}$		$\mathbb{Z}$	

$succ(4)$

zombit-vector



	1	2	3	4	5	6	7	8
U:	1	0	1	0	1	0	1	1

$rank_0(2) = 1$

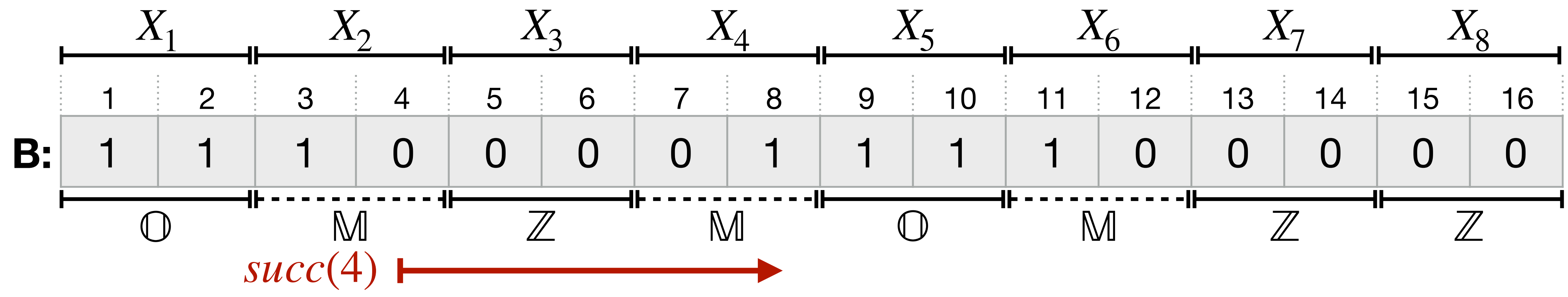
	1	2	3	4	5	6	7	8
O:	1	1	0	1	1	1	0	0

$succ(2 + 1)$

	1	2	3	4	5	6
M:	1	0	0	1	1	0

$succ(2)$

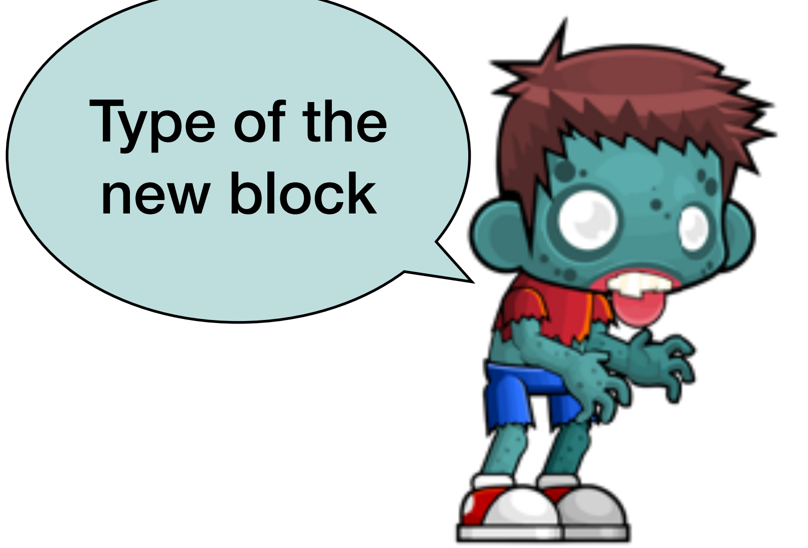
zombit-vector

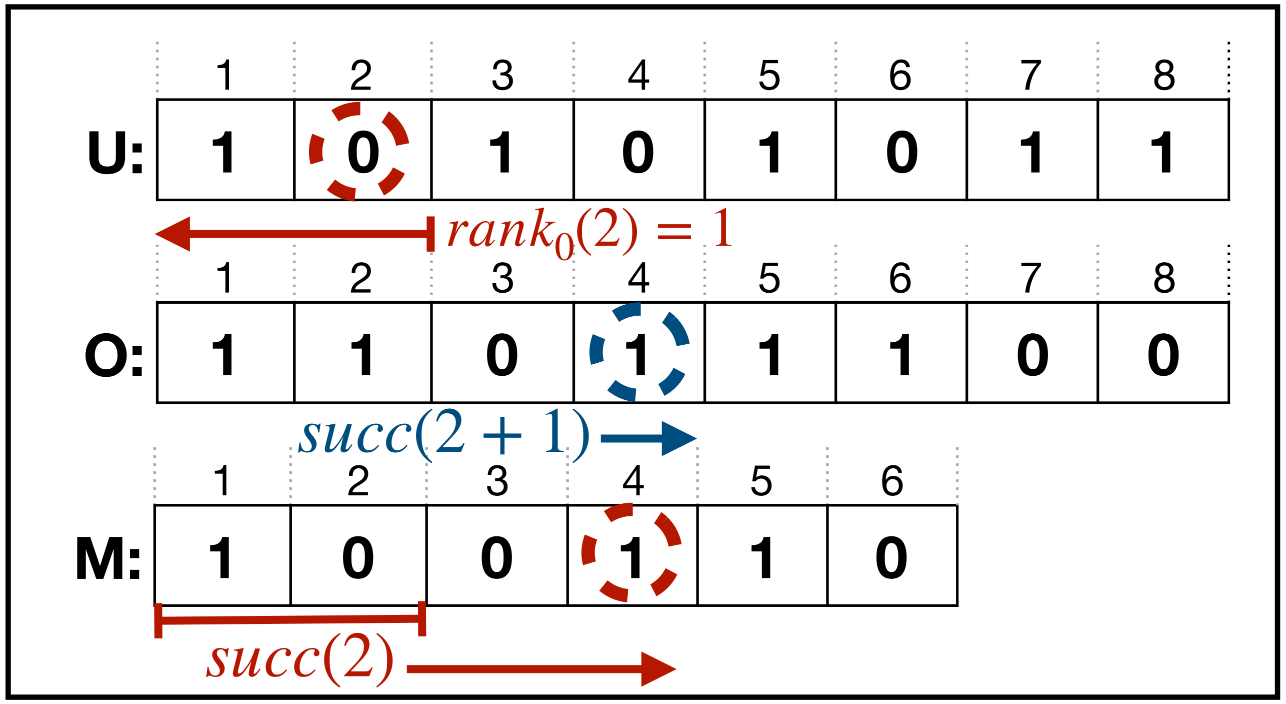


	X_1		X_2		X_3		X_4		X_5		X_6		X_7		X_8	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
B:	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	0
	$\mathbb{O}$		$\mathbb{M}$		$\mathbb{Z}$		$\mathbb{M}$		$\mathbb{O}$		$\mathbb{M}$		$\mathbb{Z}$		$\mathbb{Z}$	

$succ(4)$

zombit-vector





	1	2	3	4	5	6	7	8
U:	1	0	1	0	1	0	1	1

$rank_0(2) = 1$

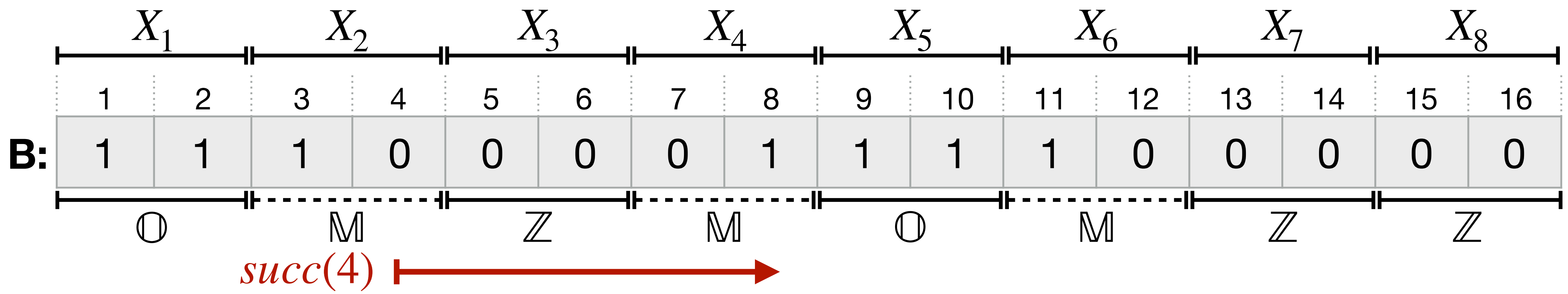
	1	2	3	4	5	6	7	8
O:	1	1	0	1	1	1	0	0

$succ(2+1)$

	1	2	3	4	5	6
M:	1	0	0	1	1	0

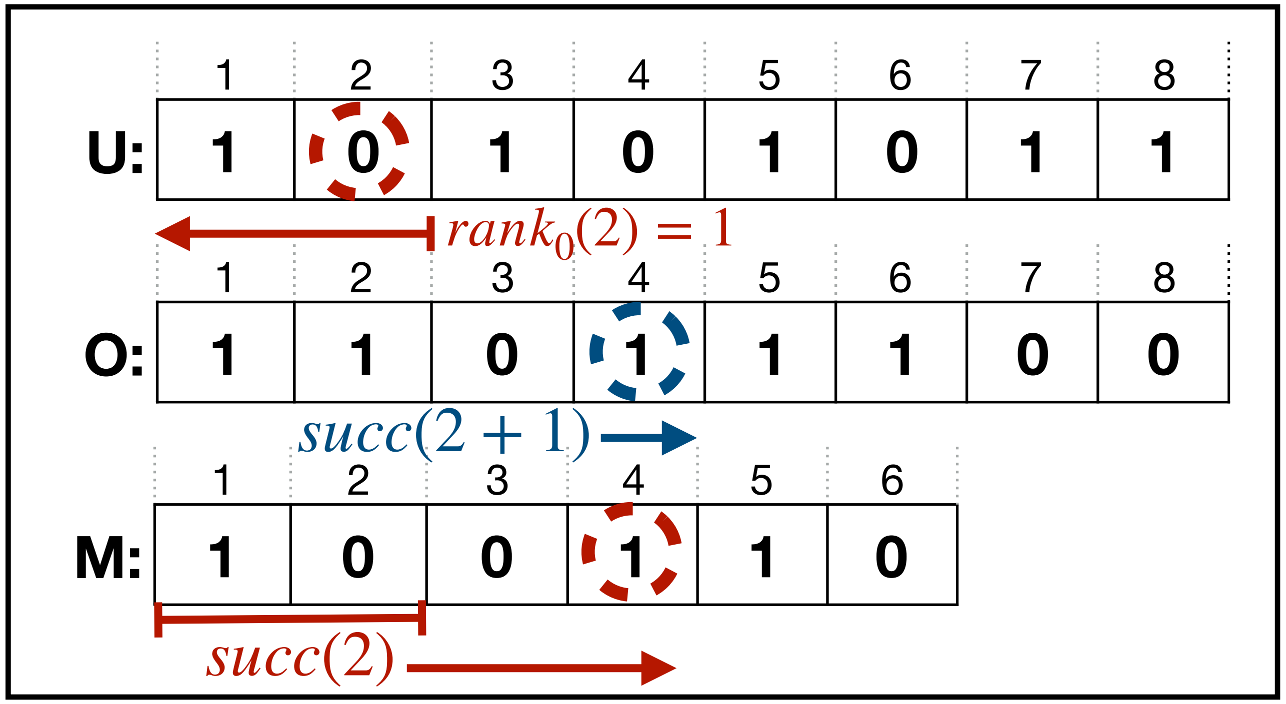
$succ(2)$

zombit-vector



	X_1		X_2		X_3		X_4		X_5		X_6		X_7		X_8	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
B:	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	0
	$\mathbb{O}$		$\mathbb{M}$		$\mathbb{Z}$		$\mathbb{M}$		$\mathbb{O}$		$\mathbb{M}$		$\mathbb{Z}$		$\mathbb{Z}$	

zombit-vector

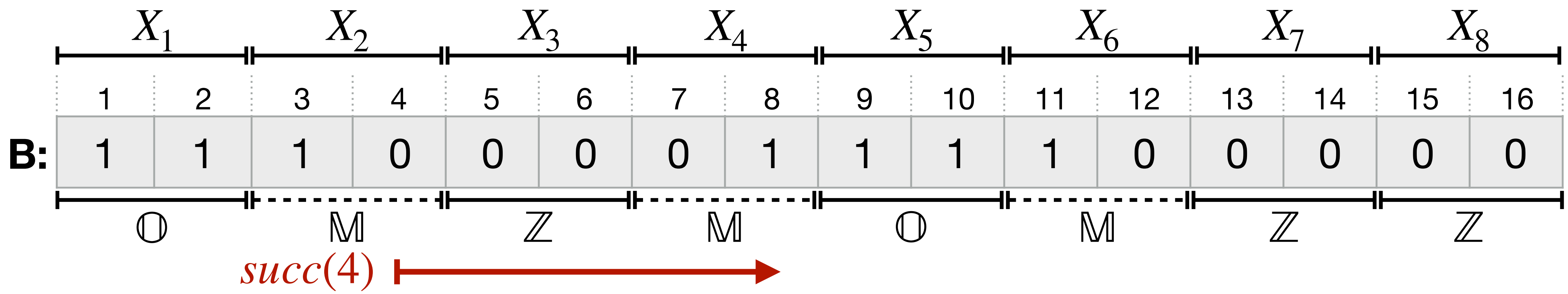


	1	2	3	4	5	6	7	8
U:	1	0	1	0	1	0	1	1

	1	2	3	4	5	6	7	8
O:	1	1	0	1	1	1	0	0

	1	2	3	4	5	6
M:	1	0	0	1	1	0

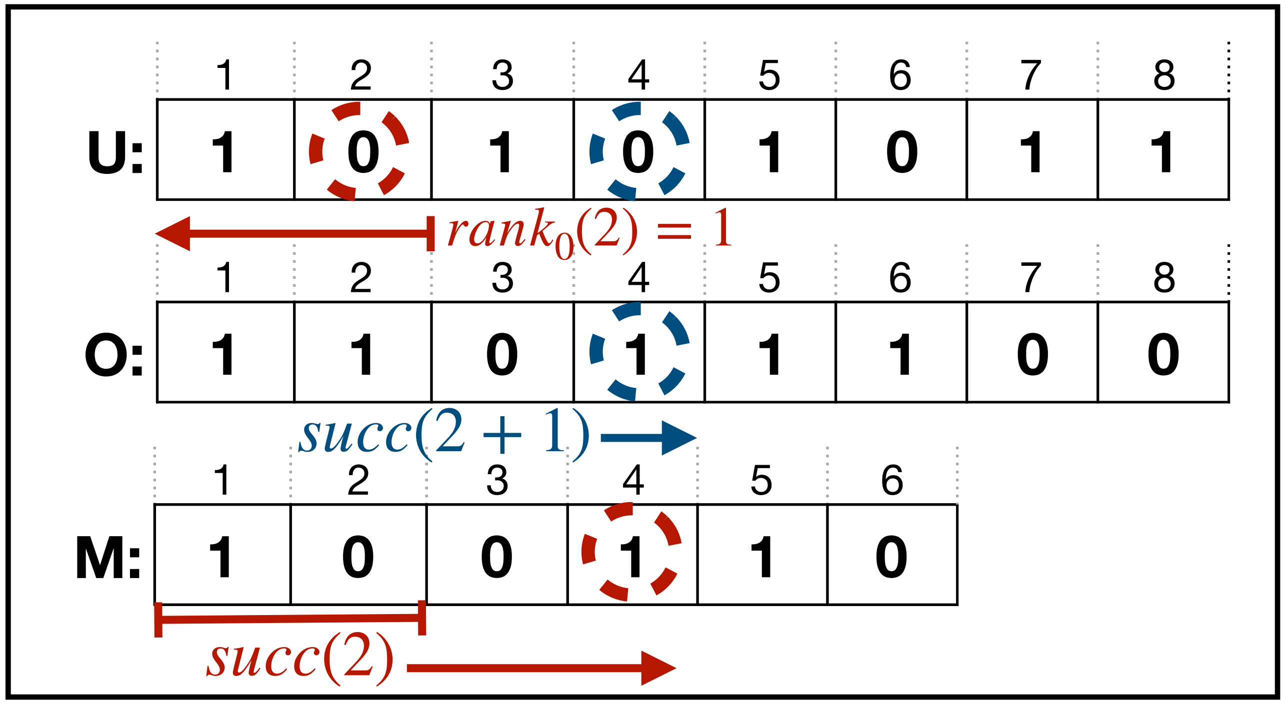
zombit-vector



	X_1		X_2		X_3		X_4		X_5		X_6		X_7		X_8	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
B:	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	0
	$\mathbb{O}$		$\mathbb{M}$		$\mathbb{Z}$		$\mathbb{M}$		$\mathbb{O}$		$\mathbb{M}$		$\mathbb{Z}$		$\mathbb{Z}$	

$succ(4)$

zombit-vector



	1	2	3	4	5	6	7	8
U:	1	0	1	0	1	0	1	1

$rank_0(2) = 1$

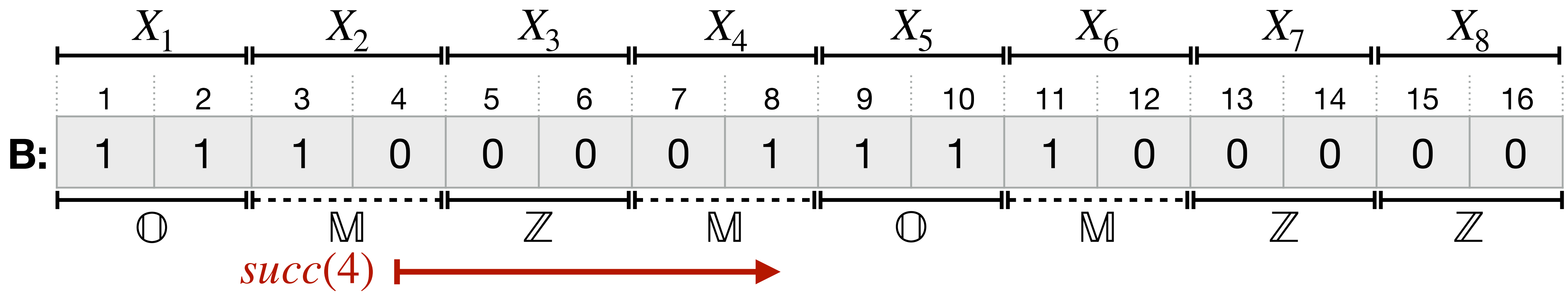
	1	2	3	4	5	6	7	8
O:	1	1	0	1	1	1	0	0

$succ(2 + 1)$

	1	2	3	4	5	6
M:	1	0	0	1	1	0

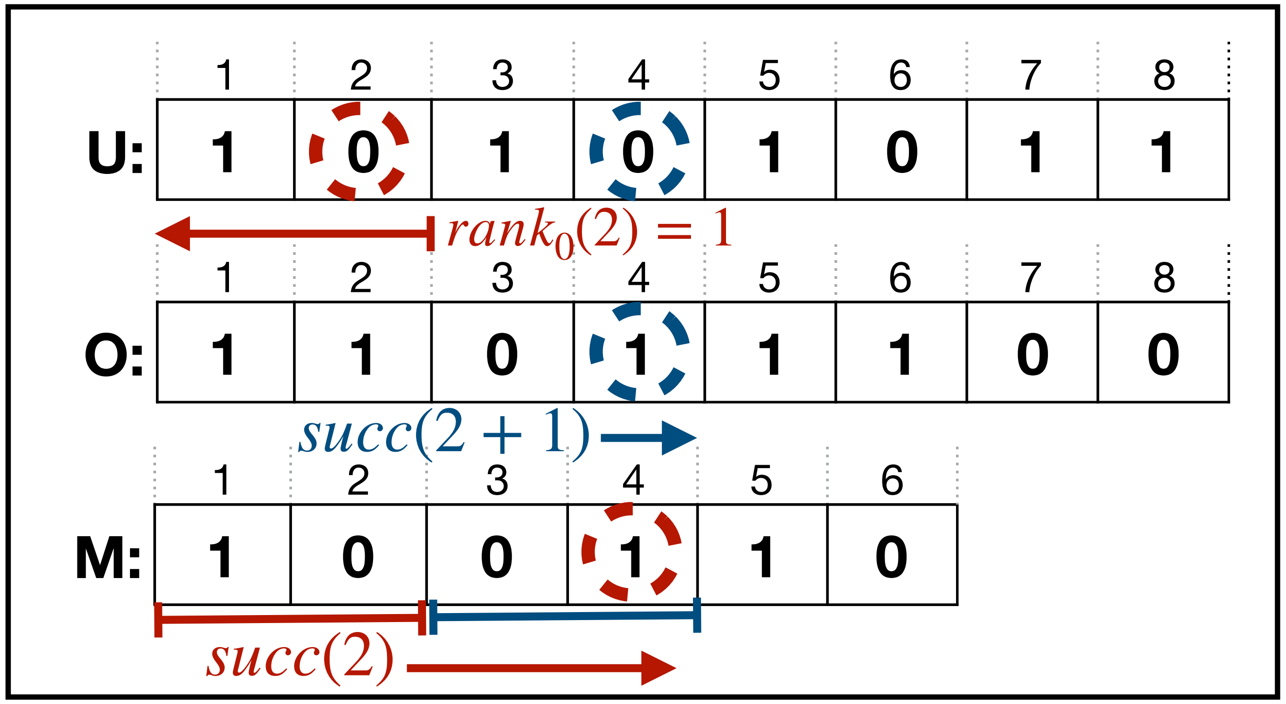
$succ(2)$

zombit-vector



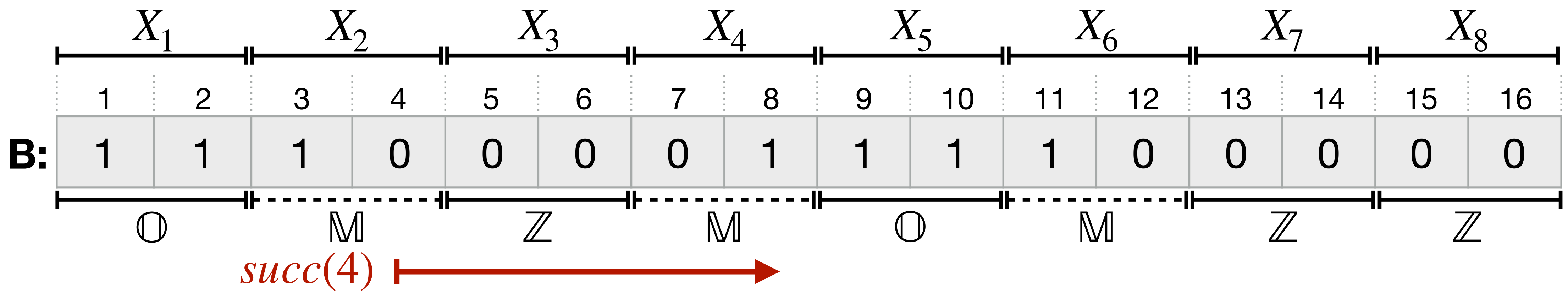
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
B:	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	0

zombit-vector



	1	2	3	4	5	6	7	8
U:	1	0	1	0	1	0	1	1
O:	1	1	0	1	1	1	0	0
M:	1	0	0	1	1	0		

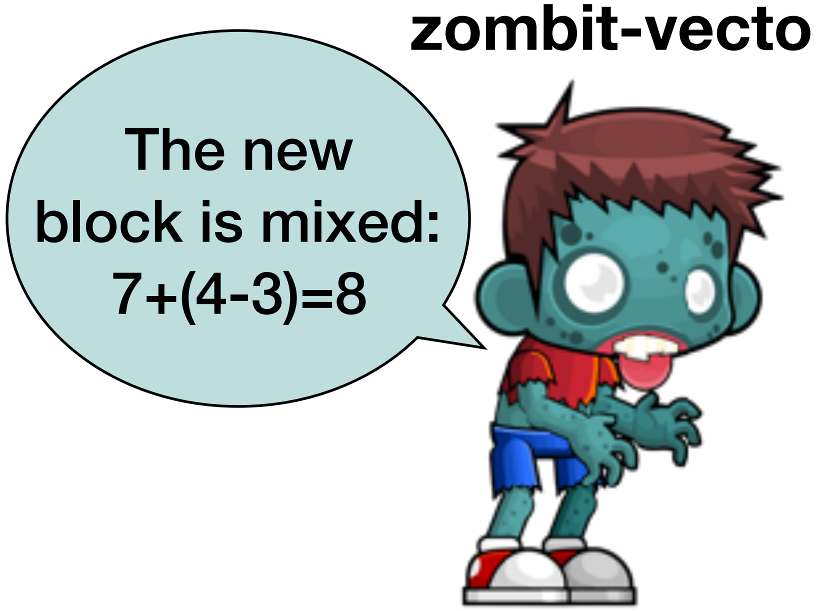
zombit-vector

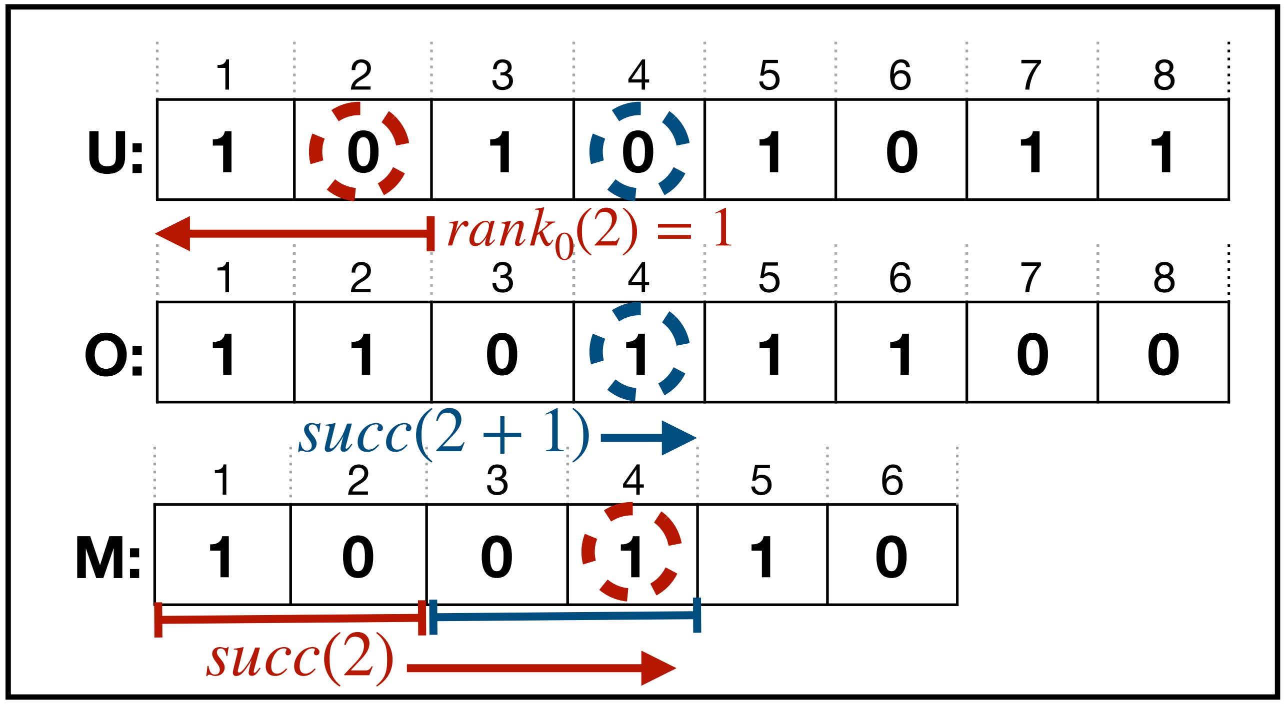


	X_1		X_2		X_3		X_4		X_5		X_6		X_7		X_8	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
B:	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	0
	$\mathbb{O}$		$\mathbb{M}$		$\mathbb{Z}$		$\mathbb{M}$		$\mathbb{O}$		$\mathbb{M}$		$\mathbb{Z}$		$\mathbb{Z}$	

$succ(4)$ ⟶

zombit-vector





	1	2	3	4	5	6	7	8
U:	1	0	1	0	1	0	1	1

$rank_0(2) = 1$

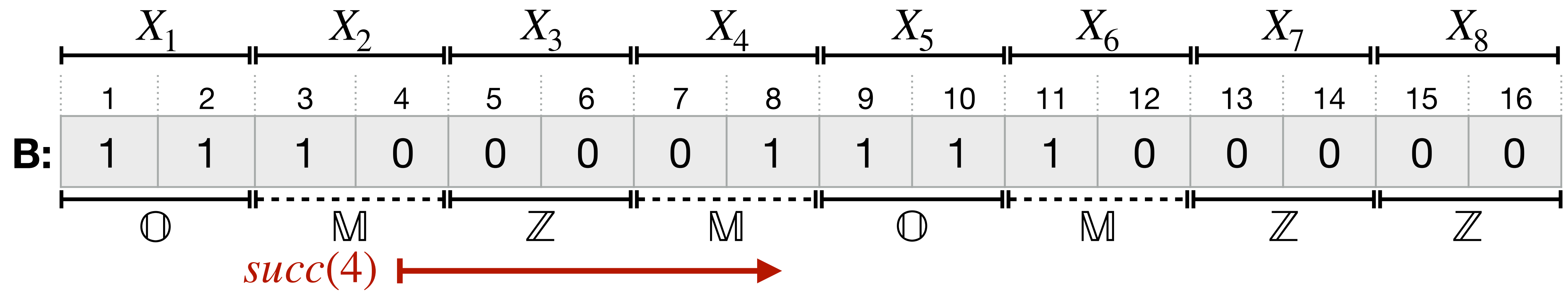
	1	2	3	4	5	6	7	8
O:	1	1	0	1	1	1	0	0

$succ(2+1)$ ⟶

	1	2	3	4	5	6
M:	1	0	0	1	1	0

$succ(2)$ ⟶

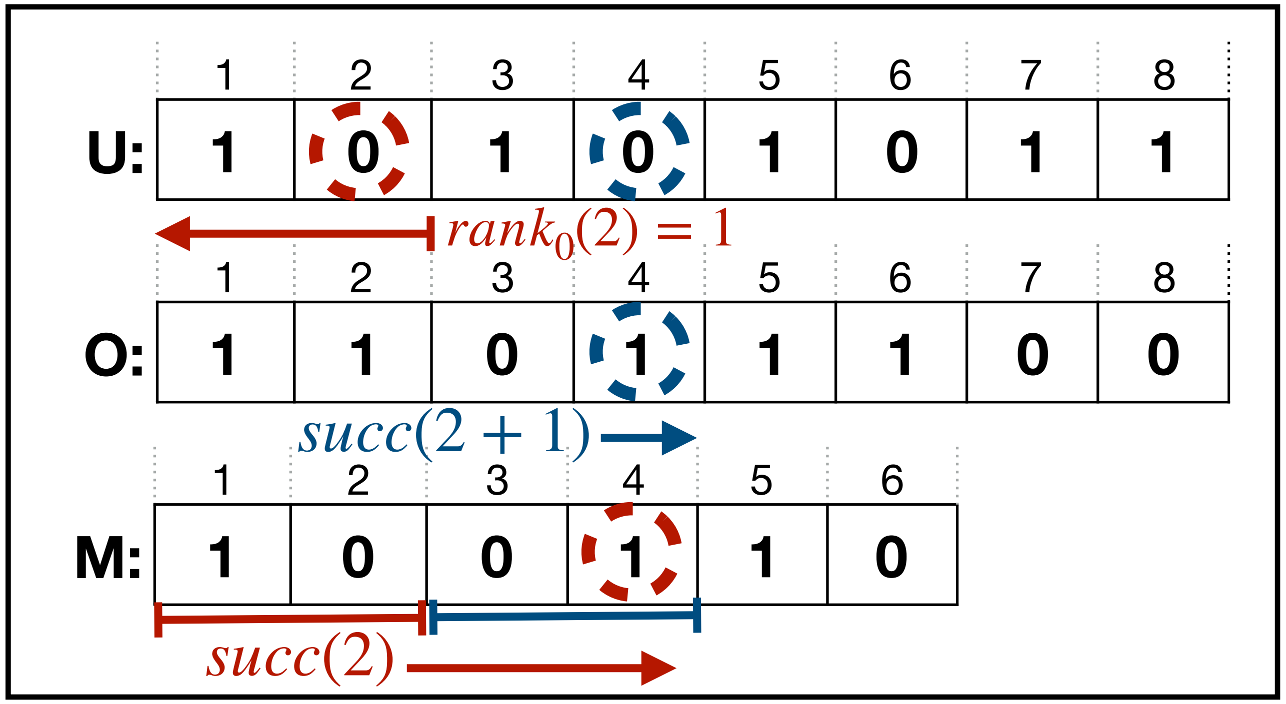
zombit-vector



	X_1		X_2		X_3		X_4		X_5		X_6		X_7		X_8	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
B:	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	0
	$\mathbb{O}$		$\mathbb{M}$		$\mathbb{Z}$		$\mathbb{M}$		$\mathbb{O}$		$\mathbb{M}$		$\mathbb{Z}$		$\mathbb{Z}$	

$succ(4)$

zombit-vector



	1	2	3	4	5	6	7	8
U:	1	0	1	0	1	0	1	1

$rank_0(2) = 1$

	1	2	3	4	5	6	7	8
O:	1	1	0	1	1	1	0	0

$succ(2 + 1)$

	1	2	3	4	5	6
M:	1	0	0	1	1	0

$succ(2)$

zombit-vector

- The optimal value of β is **sqrt(n/k)**, hence the total space requires **O(sqrt(kn))** bits.

zombit-vector

- The optimal value of β is **sqrt(n/k)**, hence the total space requires **O(sqrt(kn))** bits.

- *zombit* can solve **rank** and **access** operations in **O(1)** time and using **O(sqrt(kn))** bits of space.

zombit-vector

- The optimal value of β is **sqrt(n/k)**, hence the total space requires **O(sqrt(kn))** bits.

- *zombit* can solve **rank** and **access** operations in **O(1)** time and using **O(sqrt(kn))** bits of space.

- We can apply recursively this technique over the bitmap M, **zombit-rec.**

zombit-vector

- The optimal value of β is **sqrt(n/k)**, hence the total space requires **O(sqrt(kn))** bits.

- *zombit* can solve **rank** and **access** operations in **O(1)** time and using **O(sqrt(kn))** bits of space.

- We can apply recursively this technique over the bitmap M, **zombit-rec.**

- *zombit-rec* converges to **O(k)** bits and solves access, rank, succ and pred in **O(log(n/k))** time.

zombit-vector

Bitvector	Time	Space
plain	$O(1)$	$n + o(n)$
rrr	$O(1)$	$nH_0 + o(n)$
rec-rank	$O(\log \frac{n}{m})$	$\log \frac{n}{m} + m + o(n)$
sd-array	$O(\log \frac{n}{m})$	$m \log \frac{n}{m} + O(m)$
oz-vector	$O(\log k)$	$k \log \frac{2n}{k} + O(k)$
hybrid	$O(\log n)$	$min(k, m)\lceil \log b \rceil + o(n)$
zombit	$\mathbf{O(1)}$	$\mathbf{O(\sqrt{kn})}$
zombit-rec	$\mathbf{O(\log \frac{n}{k})}$	$\mathbf{O(k)}$

Data
Compression
Conference

Outline

- ☑ Introduction
- ☑ Background
- ☑ zombit-vector
- ☐ **Experimental evaluation**
- ☐ Conclusions
- ☐ Future work

Experimental evaluation

- We evaluate the behavior of *zombit* and *zombit-rec* in three queries: **successor**, **access**, and **rank**.

Experimental evaluation

- We evaluate the behavior of *zombit* and *zombit-rec* in three queries: **successor**, **access**, and **rank.**

- Those queries are compared against the previous presented bitvectors: **plain**, **rrr**, **rec-rank**, **sd-array**, **oz-vector**, and **hybrid**.

Experimental evaluation

- We evaluate the behavior of *zombit* and *zombit-rec* in three queries: **successor**, **access**, and **rank.**

- Those queries are compared against the previous presented bitvectors: **plain**, **rrr**, **rec-rank**, **sd-array**, **oz-vector**, and **hybrid**.

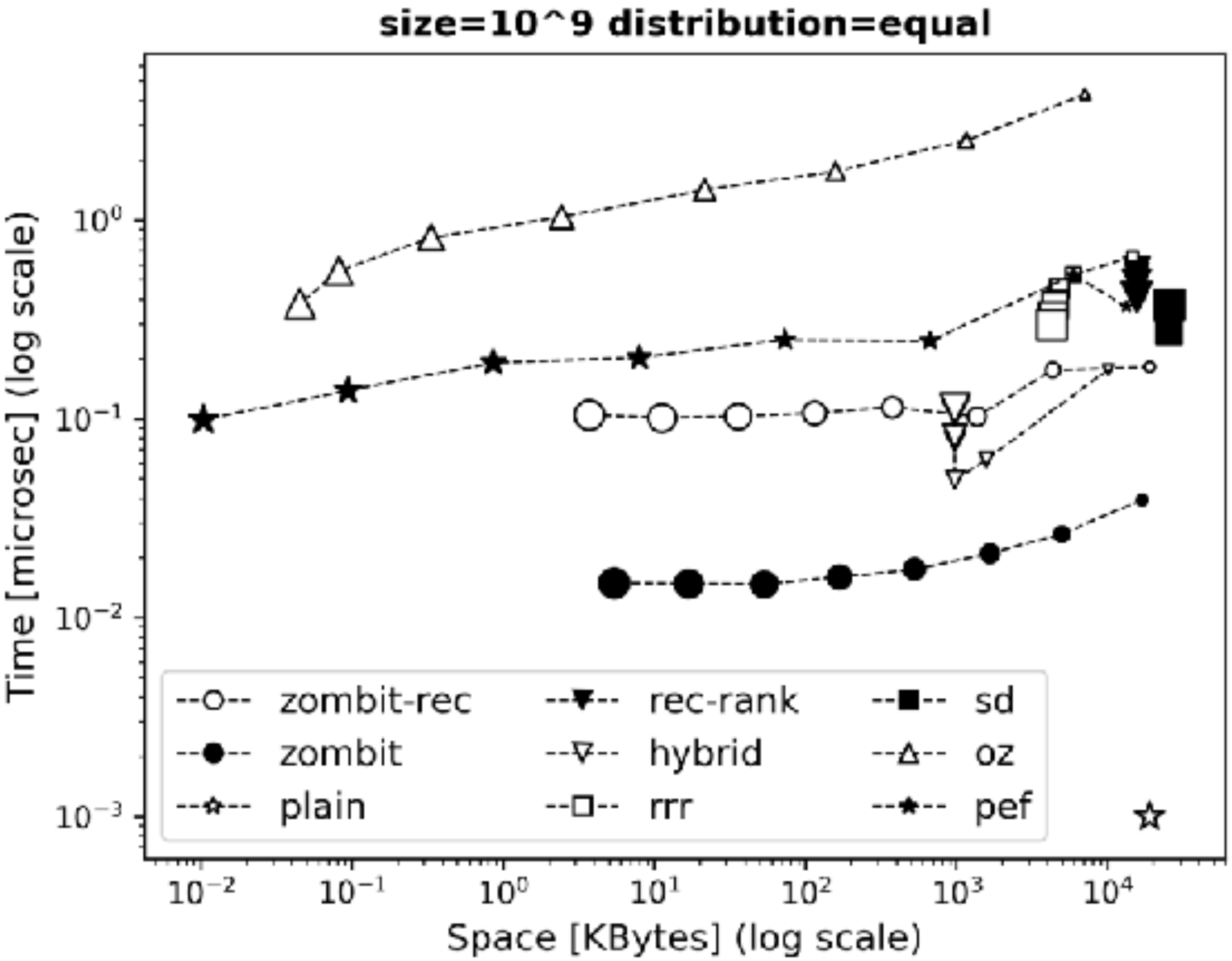
- We compare it with a large used technique on intersection of lists: **Partitioned Elias-Fano**

Experimental evaluation

- We evaluate the behavior of *zombit* and *zombit-rec* in three queries: **successor**, **access**, and **rank.**
- Those queries are compared against the previous presented bitvectors: **plain**, **rrr**, **rec-rank**, **sd-array**, **oz-vector**, and **hybrid**.
- We compare it with a large used technique on intersection of lists: **Partitioned Elias-Fano**
- Synthetic bitvectors with different sizes **10^9**, **10^8** and **10^7** with different **lengths of runs**.

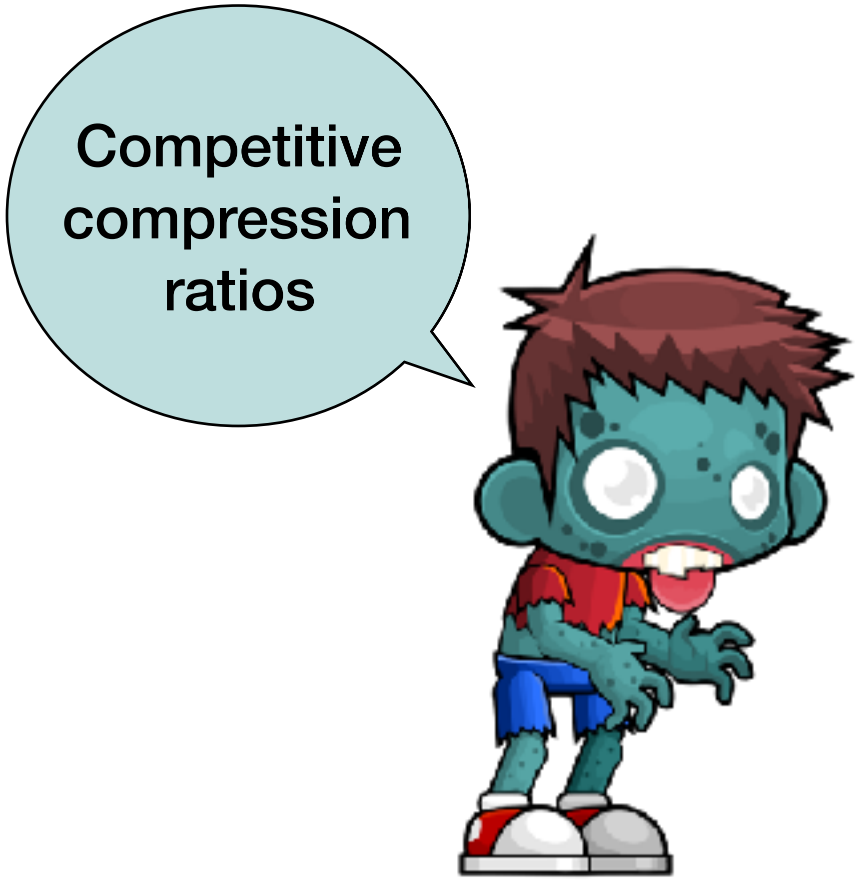
Experimental evaluation

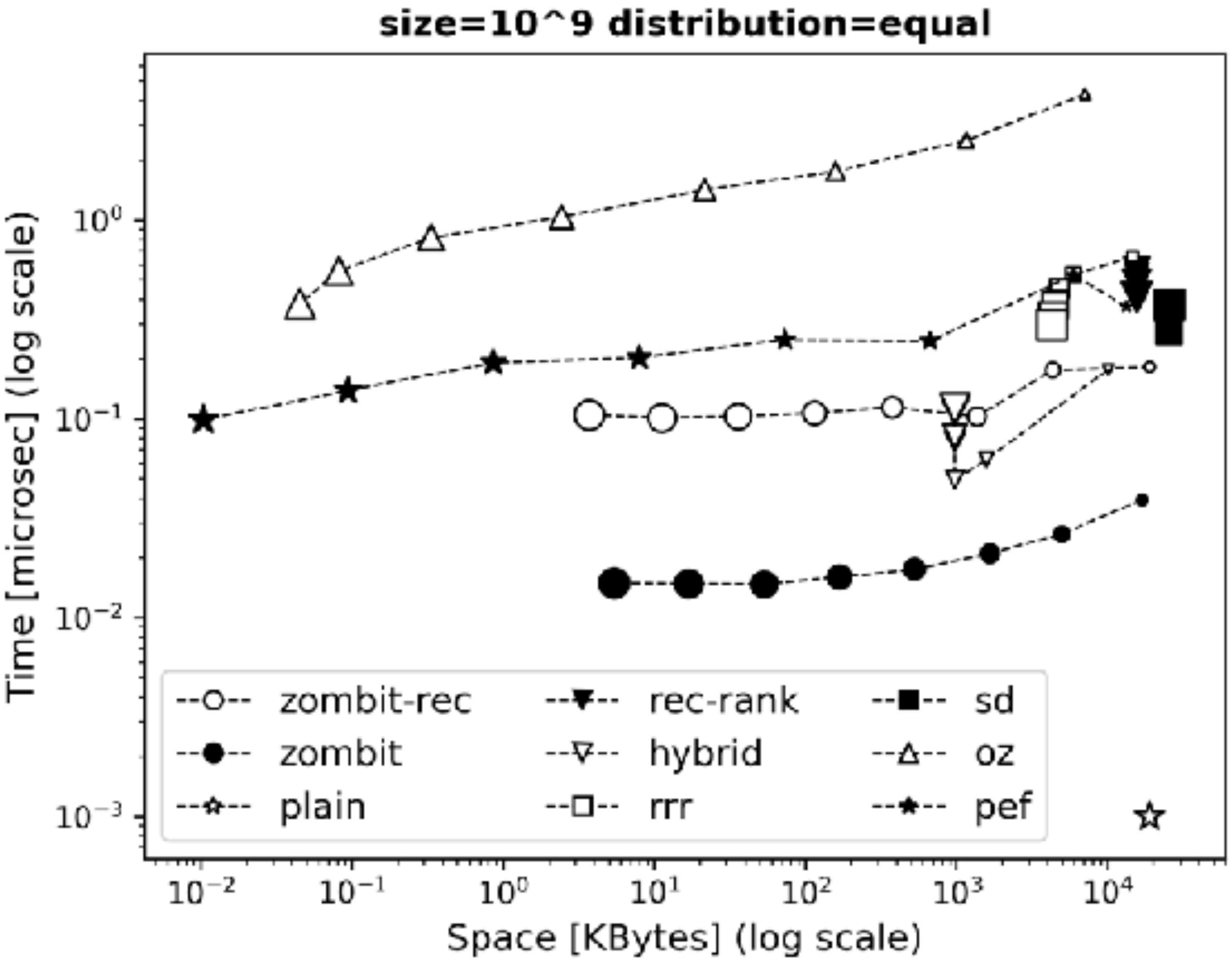
- **Successor operation**



Experimental evaluation

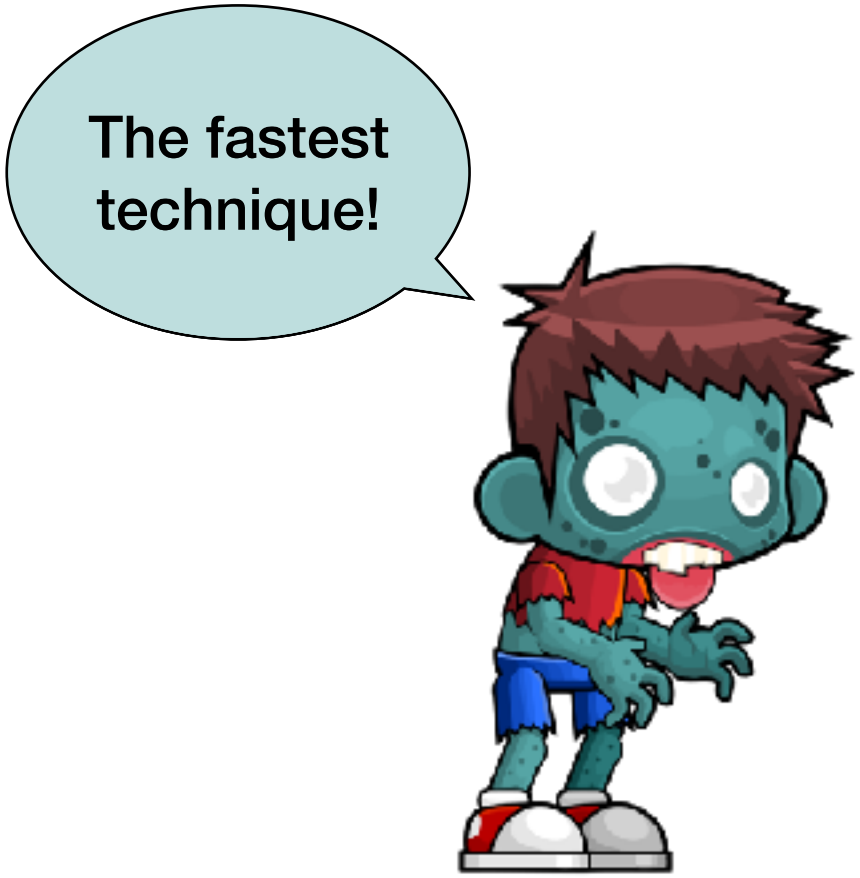
- **Successor operation**

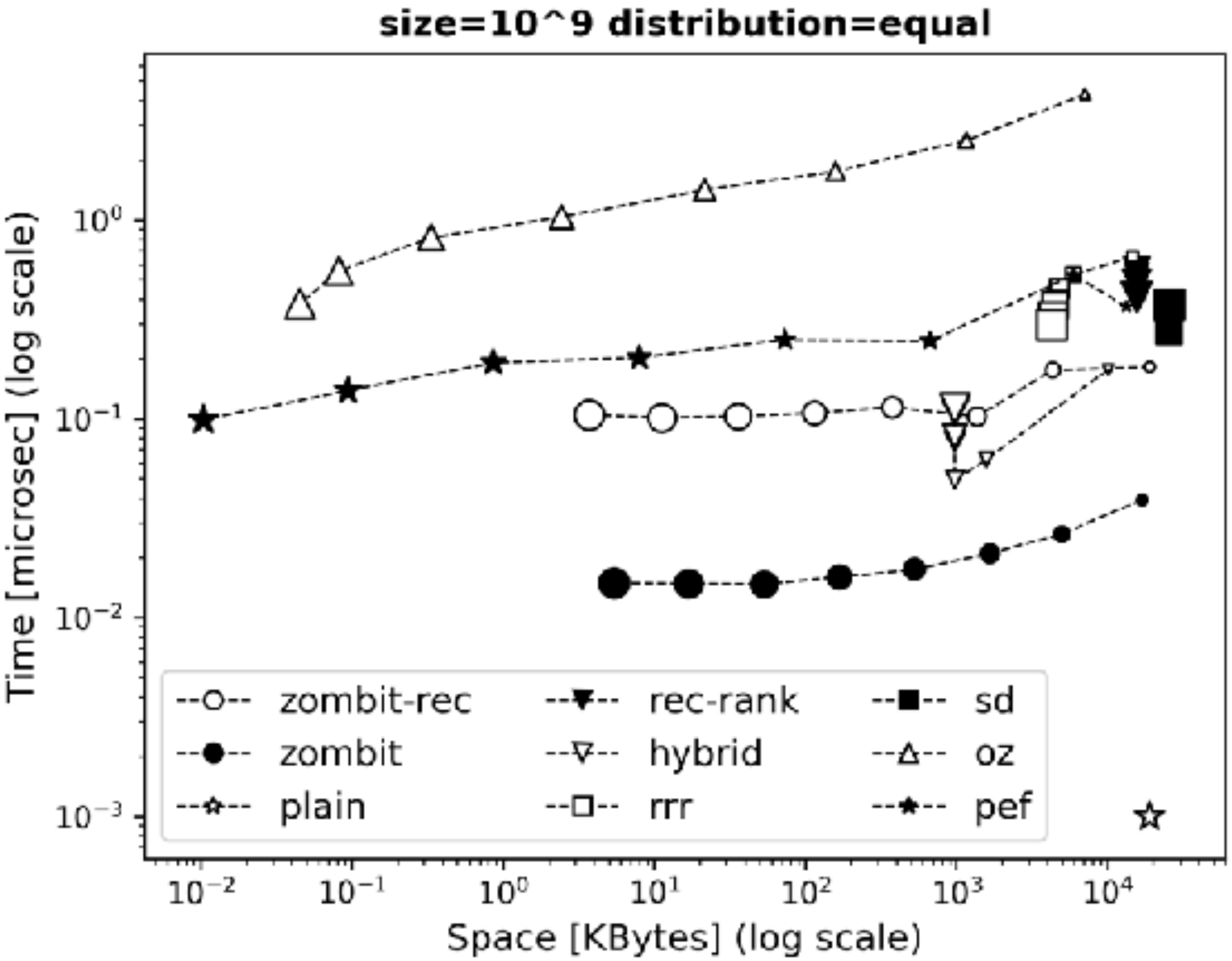

Competitive compression ratios



Experimental evaluation

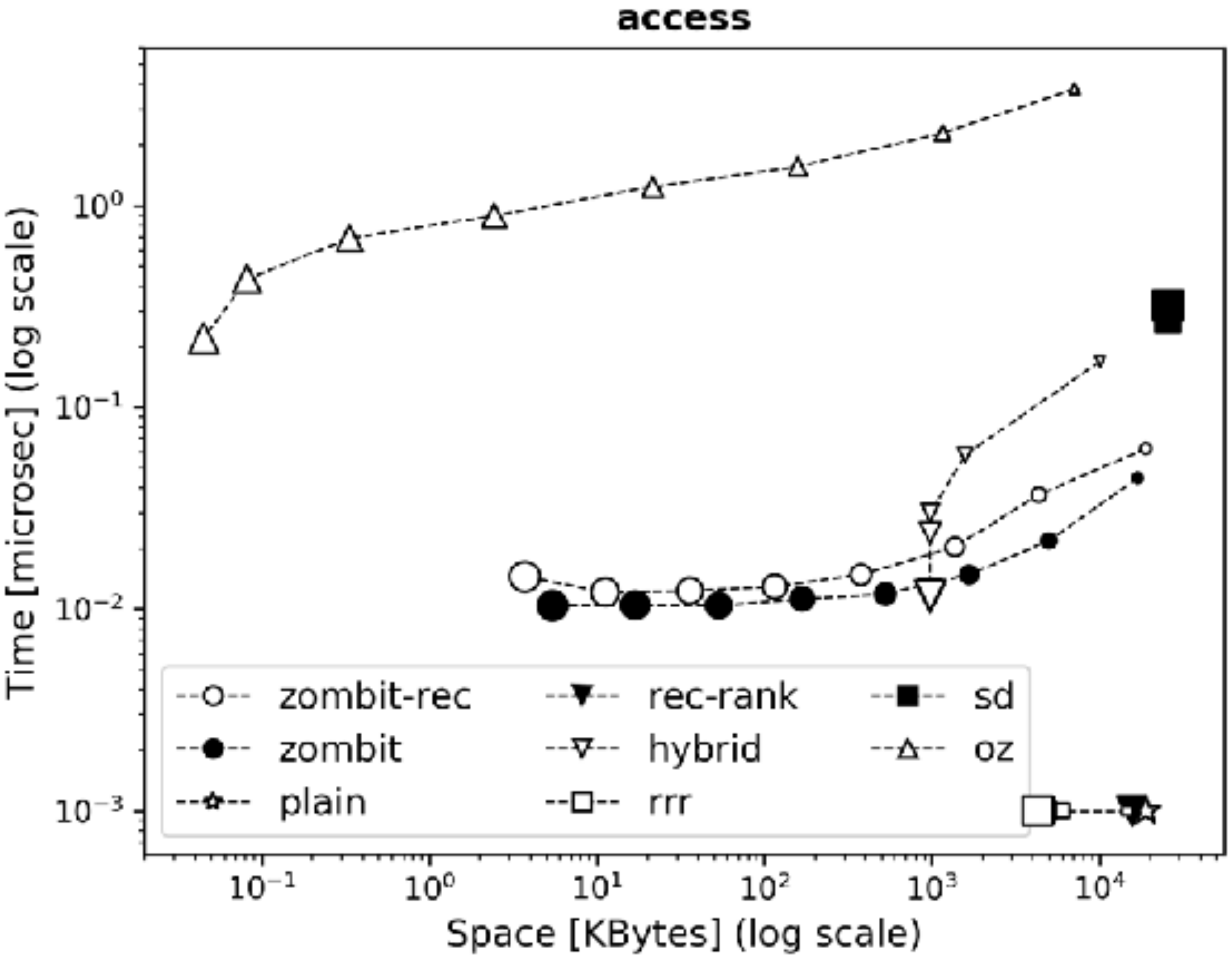
- **Successor operation**





Experimental evaluation

- **Access operation**

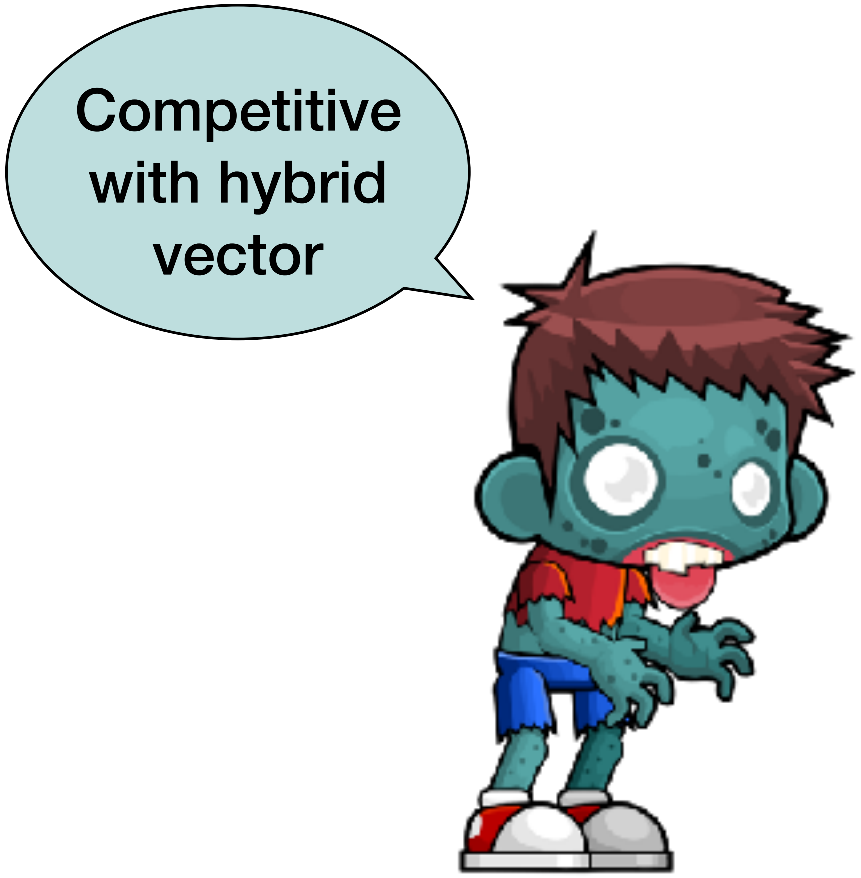


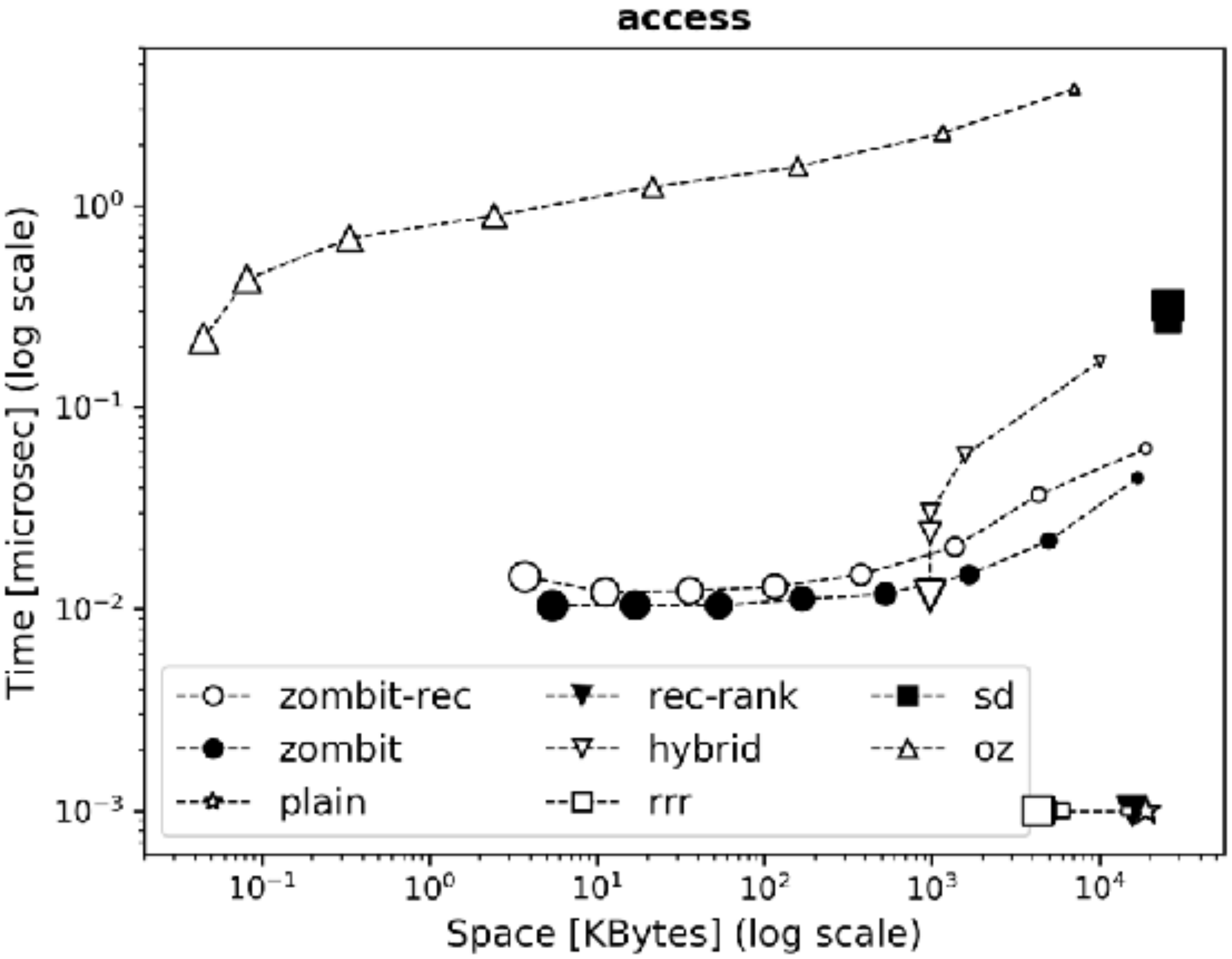
Data Compression Conference

Wednesday, 25th March 2020, Snowbird, Utah

Experimental evaluation

- **Access operation**

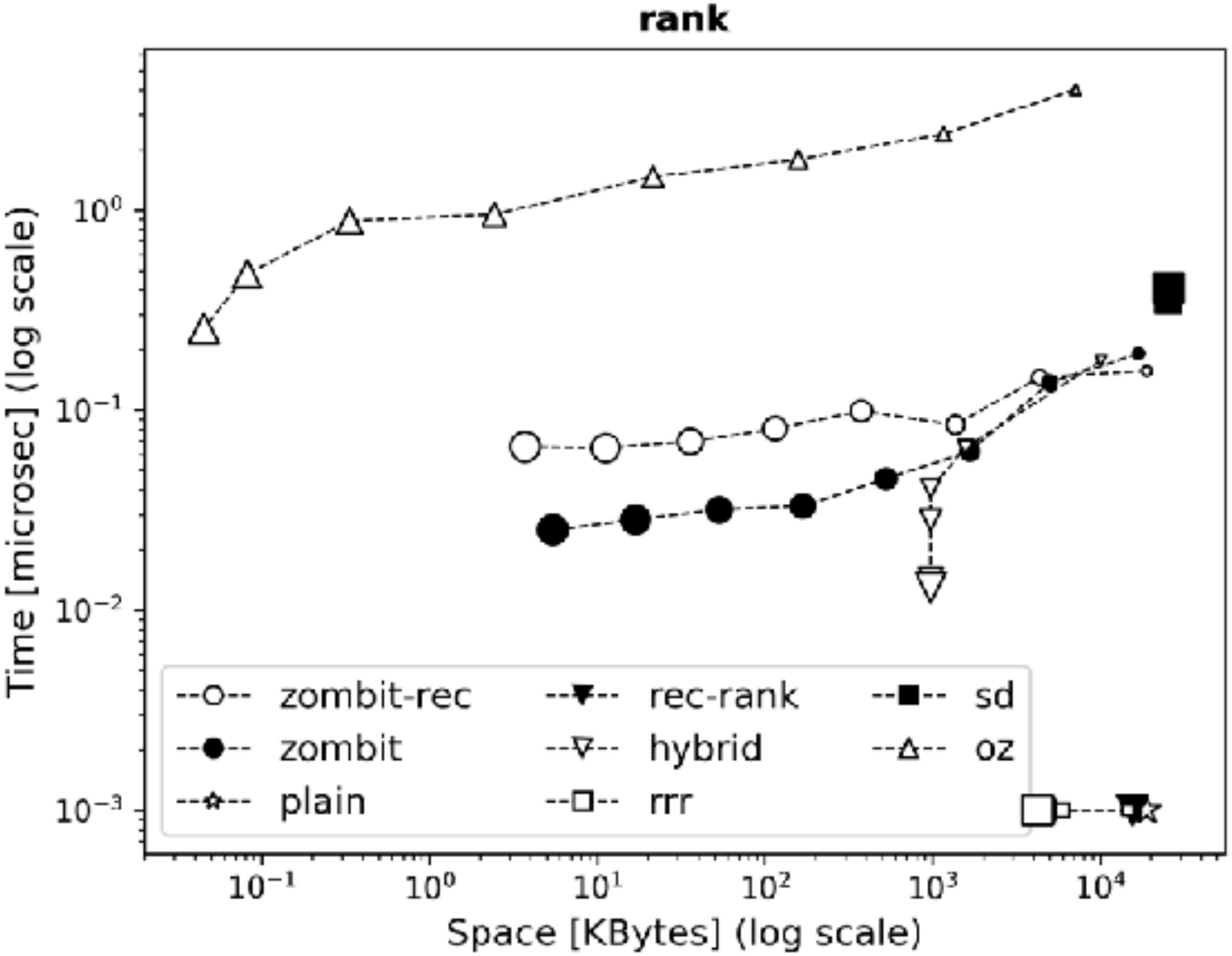




Wednesday, 25th March 2020, Snowbird, Utah

Experimental evaluation

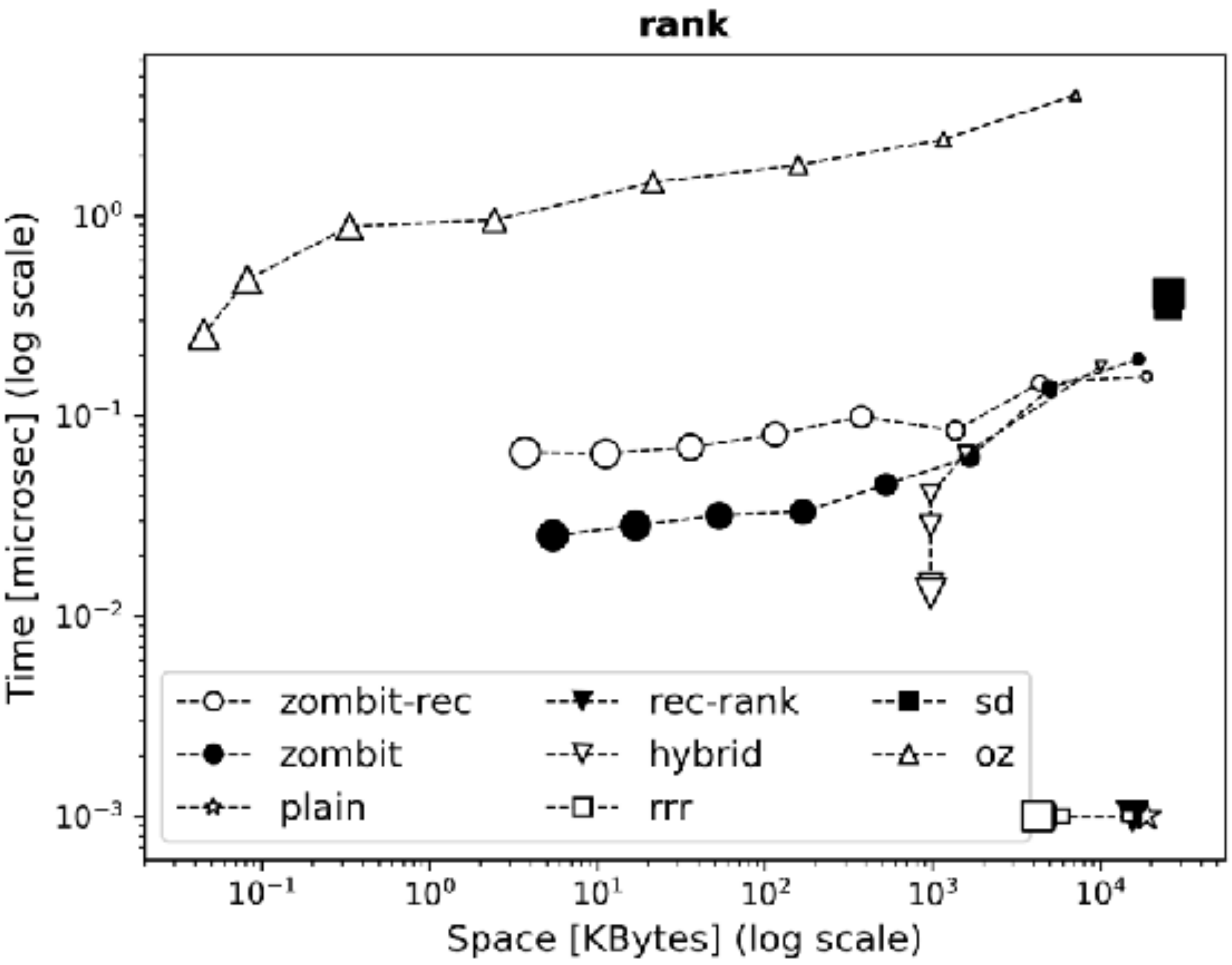
- **Rank operation**



Experimental evaluation

- **Rank operation**

Slower than hybrid vector



Outline

- ☑ Introduction
- ☑ Background
- ☑ zombit-vector
- ☑ Experimental evaluation
- ☐ **Conclusions**
- ☐ Future work

Conclusions

- In theory our proposal can solve successor queries in **O(1)** time, by using **O(sqrt(kn))** bits space.

Conclusions

- In theory our proposal can solve successor queries in **O(1)** time, by using **O(sqrt(kn))** bits space.

- In practice, our experimental evaluation shows:

Conclusions

- In theory our proposal can solve successor queries in **O(1)** time, by using **O(sqrt(kn))** bits space.

- In practice, our experimental evaluation shows:
    - **Good compression ratios** in runs larger than 100.

Conclusions

- In theory our proposal can solve successor queries in **O(1)** time, by using **O(sqrt(kn))** bits space.
- In practice, our experimental evaluation shows:
  - **Good compression ratios** in runs larger than 100.
  - The best time performance in successor queries.

Conclusions

- In theory our proposal can solve successor queries in **O(1)** time, by using **O(sqrt(kn))** bits space.
- In practice, our experimental evaluation shows:
  - **Good compression ratios** in runs larger than 100.
  - The best time performance in successor queries.

3-12

times faster

Outline

- ☑ Introduction
- ☑ Background
- ☑ zombit-vector
- ☑ Experimental evaluation
- ☑ Conclusions
- ☐ **Future work**

Future work

- Experimental evaluation with **real data.**

Future work

- Experimental evaluation with **real data.**
- Improving the compression ratios of bitvectors with **short runs.**

Future work

- Experimental evaluation with **real data.**
- Improving the compression ratios of bitvectors with **short runs.**
- Solving **select** operation with *o(n)* bits of extra-space.

Questions

Bitvectors with runs and the successor/predecessor problem

Data
Compression
Conference

Adrián Gómez-Brandón
adrian.gbrandon@udc.es

Wednesday, 25th March 2020, Snowbird, Utah

This work has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 690941

UNIVERSIDADE DA CORUÑA

citic