Bitvector Rank, Select, Access

Foundations of succinct data structures

Bitvector introduction & rank slides adapted directly from https://langmead-lab.org/teaching-materials/#indexing

Bitvectors

Does this bitvector have a "meaning?"

What if its name was is_prime? 😉

How might we query it?

E.g. next-highest-prime

E.g. designing a 2-universal hash,
we want smallest prime (leftmost 1)
greater than some number

0
1
1
0
1
0
1
0
0
0
1
0
1
⋮

Bitvectors

What if the bitvector represented membership in some set?

Why might want to "navigate" it?

Say we are counting 1s to estimate cardinality

Might want to "jump" between 1s, ask how spaced out they are (k^{th} minimum value)

0
0
0
1
0
0
0
1
0
0
0
0
1
0
0
0
0
0
0
0

⋮

Bitvectors

Might represent words in a document

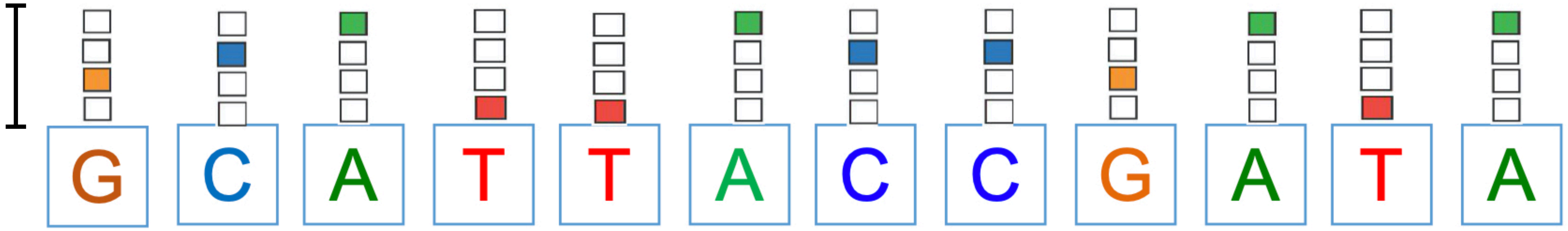
0	abinoam
1	abiogenesis
0	abiological
0	abiosis
0	abiotic
0	abiotically
0	abiotrophy
0	abirritate
0	abishag
0	abit
0	abitibi
0	abiu
1	abject

⋮

Bitvectors

Could be a "one-hot" encoding of string

4 parallel bitvectors



Navigating bitvectors = navigating the occurrences of characters in the string

https://www.encodeproject.org/documents/5284a1d9-7eb6-44d9-8979-b90408332b12/@@download/attachment/Anshul_slides.pdf

Bitvectors

How do we navigate / query bitvectors?

Proposal: "RSA" (Rank, Select, Access)

0
1
1
0
1
0
1
0
0
0
1
0
1

Bitvectors

$B\,.\,\text{access}(i) = B[i]$

Conceptually trivial, but harder if we compress B

(more later)

Indexing starts at 0

B	
0	
1	
1	← $B\,.\,\text{access}(2) = 1$
0	
1	
0	
1	
0	← $B\,.\,\text{access}(7) = 0$
0	
0	
1	
0	
1	

Bitvectors

$$B\,.\,\mathrm{rank}_1(i) = \sum_{j=0}^{i-1} B[j]$$

$$B\,.\,\mathrm{rank}_0(i) = i - B\,.\,\mathrm{rank}_1(i)$$

Rank counts up to *but not including* offset i

B
0
1
1
0
1
0
1
0
0
0
1
0
1

$B\,.\,\mathrm{rank}_1(2) = 1$

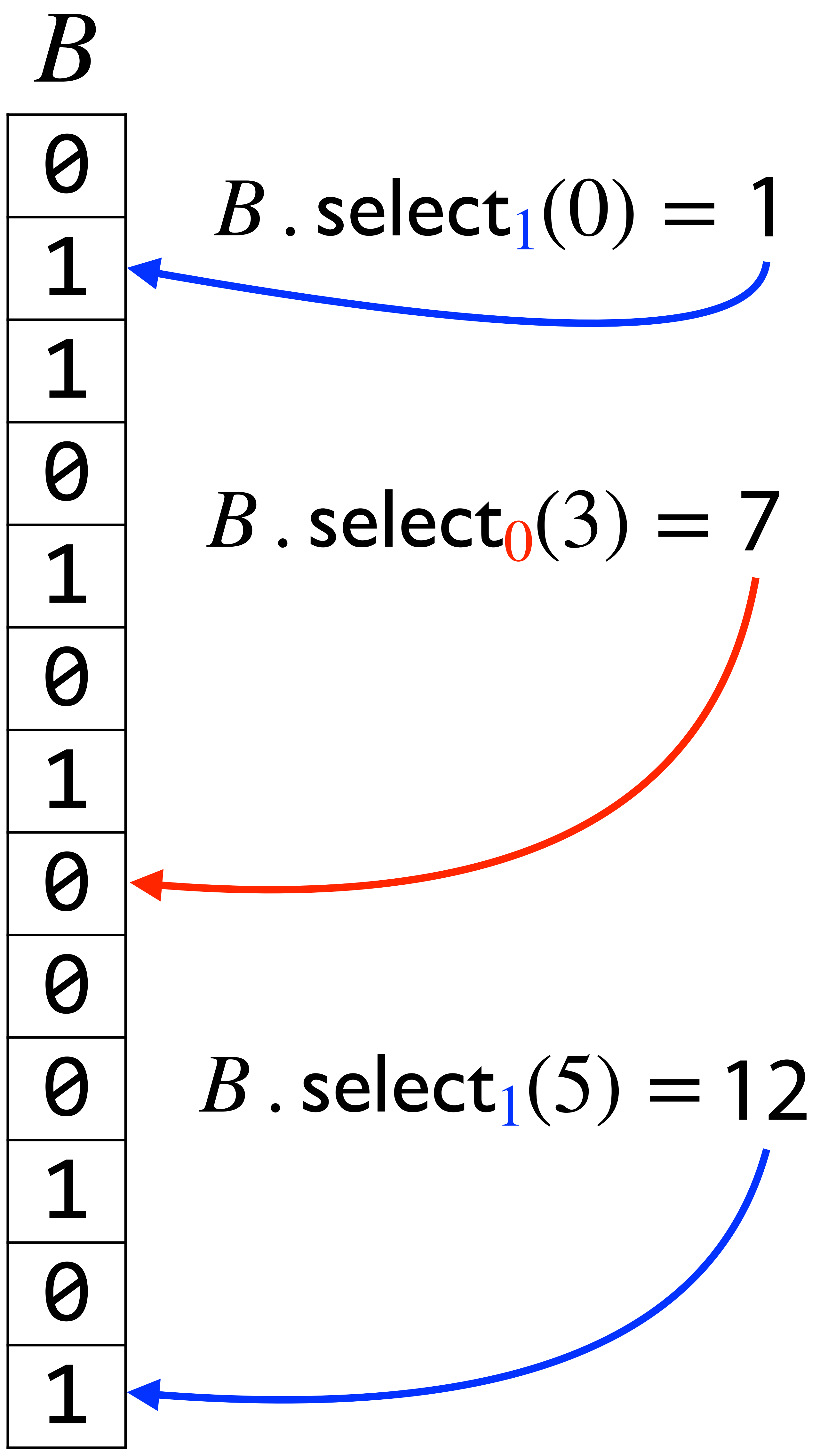
$B\,.\,\mathrm{rank}_0(5) = 2$

$B\,.\,\mathrm{rank}_1(8) = 4$

Bitvectors

$$B\,.\,\mathsf{select}_1(i) = \max\{\, j \mid B\,.\,\mathsf{rank}_1(j) = i \,\}$$

$$B\,.\,\mathsf{select}_0(i) = \max\{\, j \mid B\,.\,\mathsf{rank}_0(j) = i \,\}$$



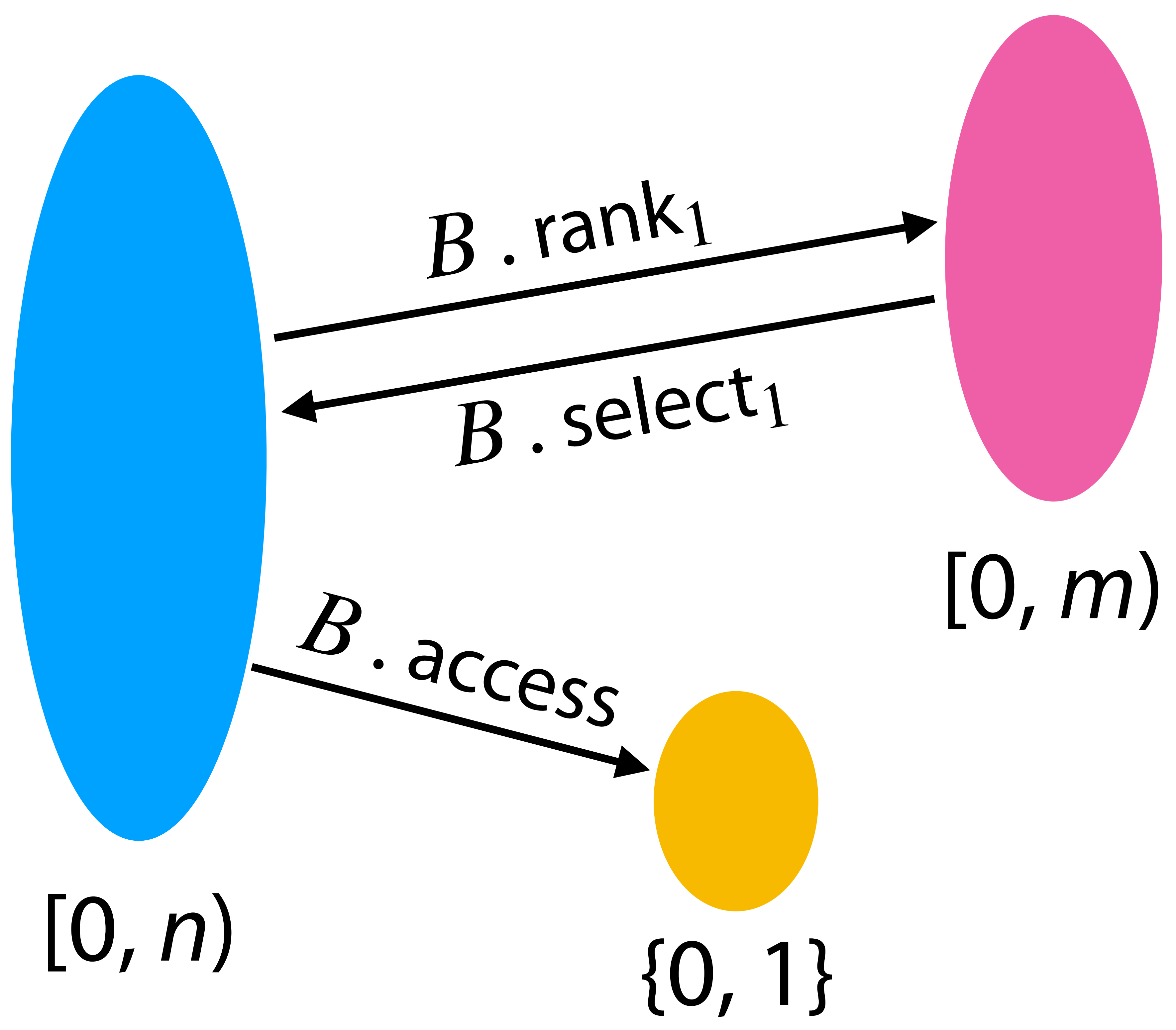
Bitvectors

$B\,.\,\text{access}(\,.\,.\,.\,)$

$B\,.\,\text{rank}(\,.\,.\,.\,)$

$B\,.\,\text{select}(\,.\,.\,.\,)$

Let $|B| = n$ and let m equal the number of set bits



Bitvectors

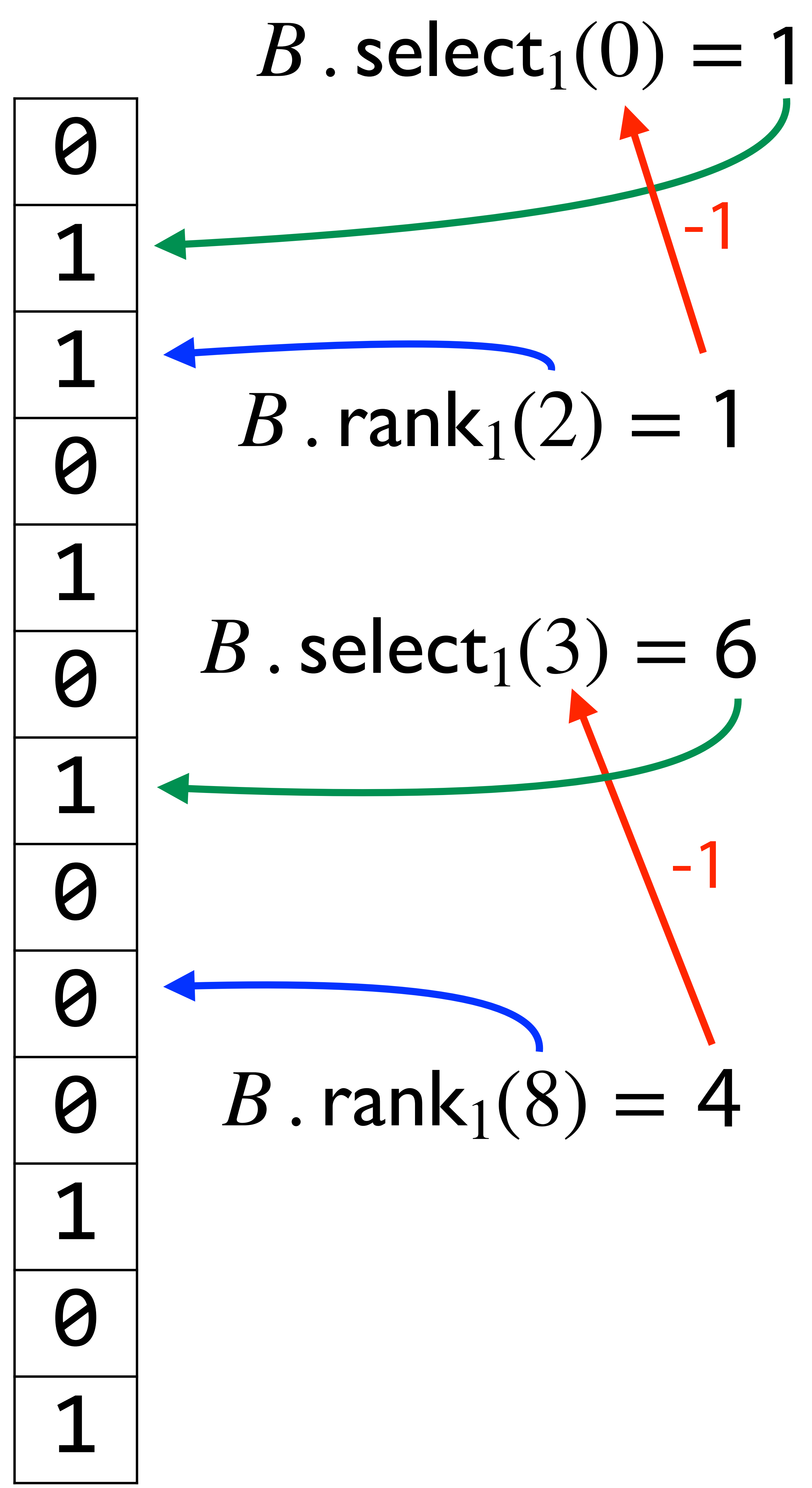
What does this do?

$B.\text{select}_1(B.\text{rank}_1(i) - 1)$

$$B.\text{rank}_1(i) = \sum_{j=0}^{i-1} B[j]$$

$$B.\text{select}_1(i) = \max\{\, j \mid B.\text{rank}_1(j) = i \,\}$$

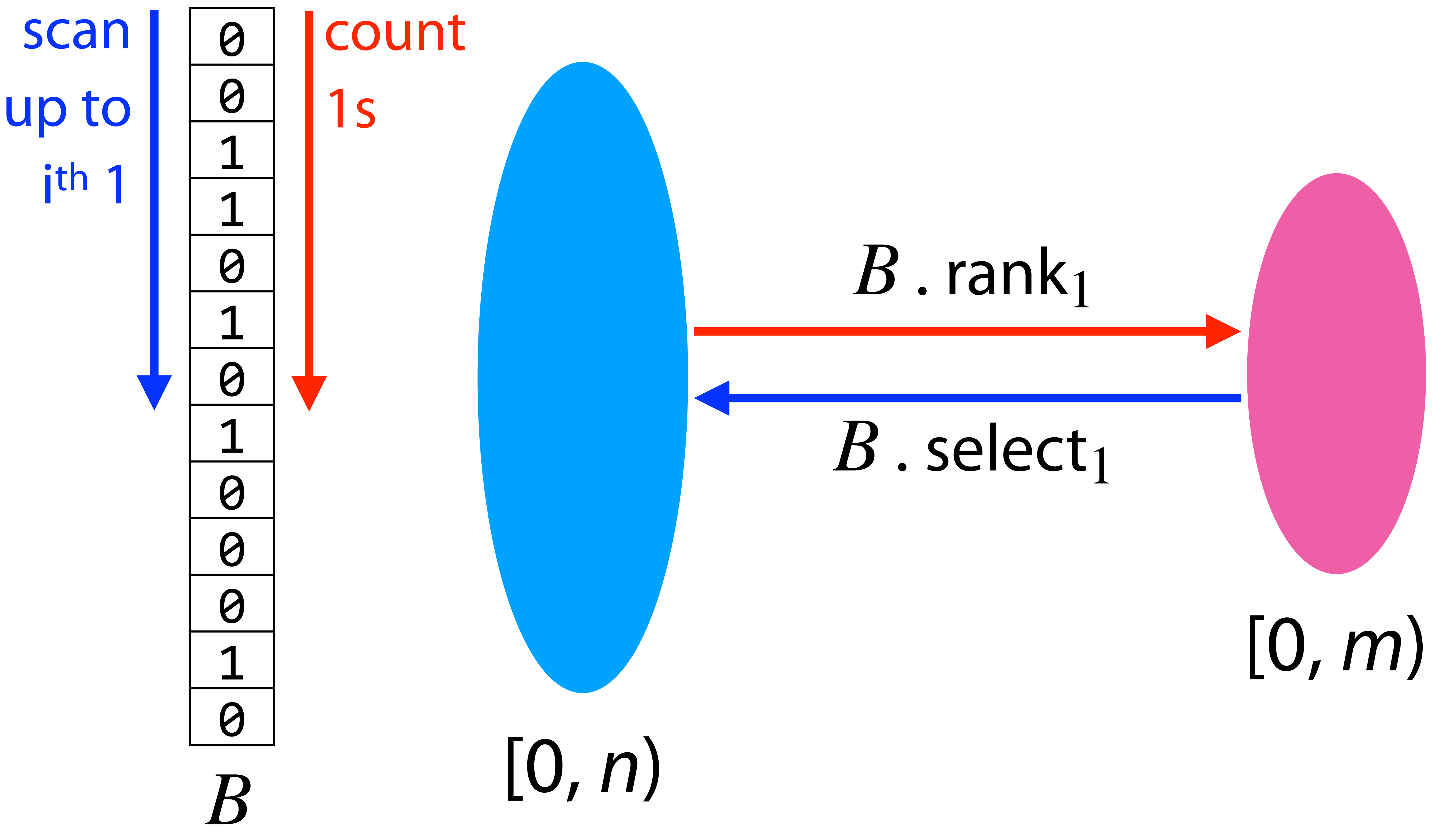
Gives offset of next-earliest set bit -- *predecessor*



Bitvectors

How to implement $B\,.\,\mathrm{rank}_1$ & $B\,.\,\mathrm{select}_1$?

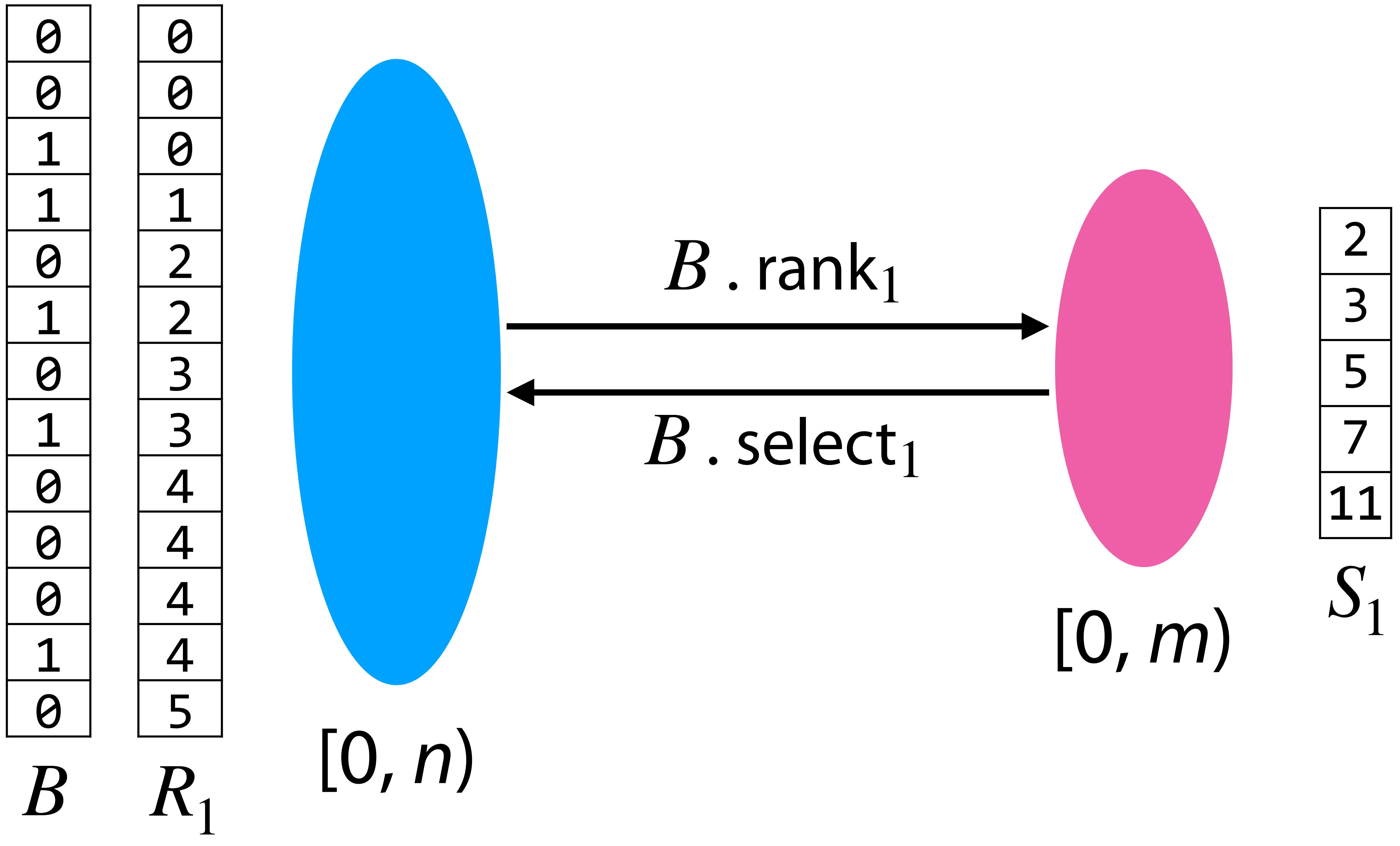
Idea 0: linear scans over B



Can we be more efficient?

Bitvectors

Idea 1: Pre-calculate all answers



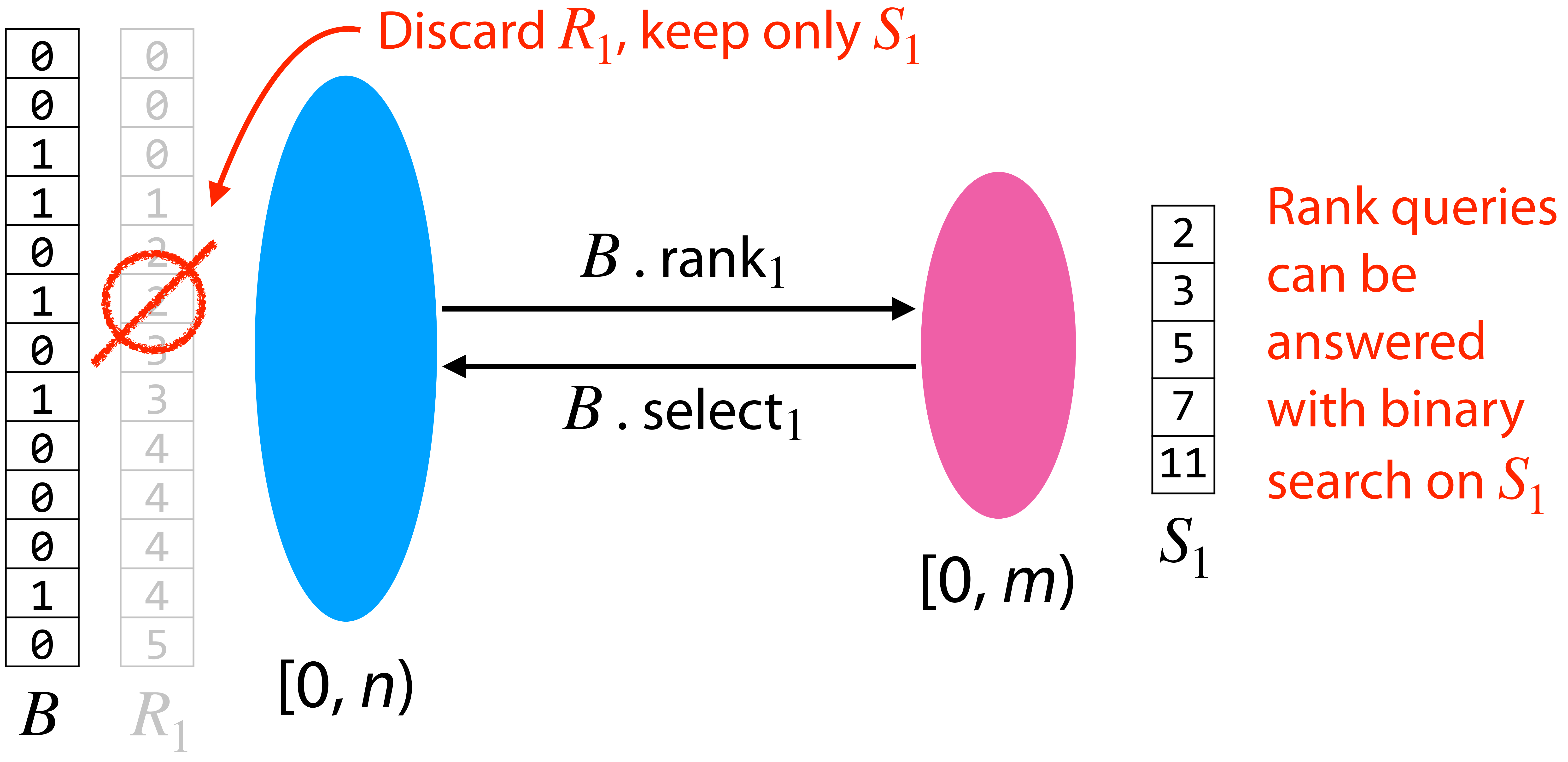
Bitvectors

Idea 1: Pre-calculate all answers

	Time	Space (bits)	Note
B . access	$O(1)$	n	Lookup
B . select$_1$	$O(1)$	$O(m \log n)$	Pre-calculate S_1
B . rank$_1$	$O(1)$	$O(n \log m)$	Pre-calculate R_1

Bitvectors

Idea 2: Pre-calculate all answers for $B\,.\,\mathrm{select}_1$



$O(m \log n)$ bits. $B\,.\,\mathrm{rank}_1$ is $O(\log m)$ time.

Bitvectors

Idea 2: Pre-calculate all answers for $B.\text{select}_1$

	Time	Space (bits)	Note
$B.\text{access}$	$O(1)$	n	Lookup
$B.\text{select}_1$	$O(1)$	$O(m \log n)$	Pre-calculate S_1
$B.\text{rank}_1$	$O(\log m)$	$O(m \log n)$	Binary search on S_1

Bitvectors

Coming soon:

	Time	Space (bits)	Note
B . access	$O(1)$	n	Lookup
B . select$_1$	$O(1)$	$\check{o}(n)$	?🧚‍♂️🦄🧚‍♀️?
B . rank$_1$	$O(1)$	$\check{o}(n)$	?🧚‍♂️🦄🧚‍♀️?