Bitvector Rank and Select in succinct space

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

Jacobson's rank

(Note: the following assumes we are doing B . rank_1 queries, but B . rank_0 queries are also doable with same methods.)

Basic ideas:

When a bitvector is sparse enough, we can simply store answers for all 1-bits

When a bitvector is short enough, we can store all answers for all possible vectors and queries

Guy Joseph Jacobson. Succinct static data structures. PhD thesis, Carnegie Mellon University, 1988.

Jacobson's rank

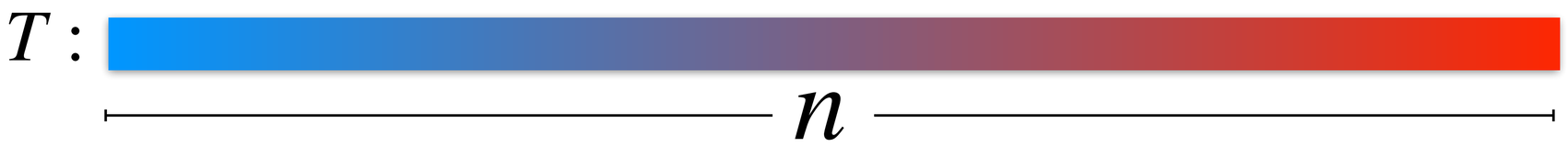
Jacobson's rank

Split the string into chunks of length $\log_2^2 n$

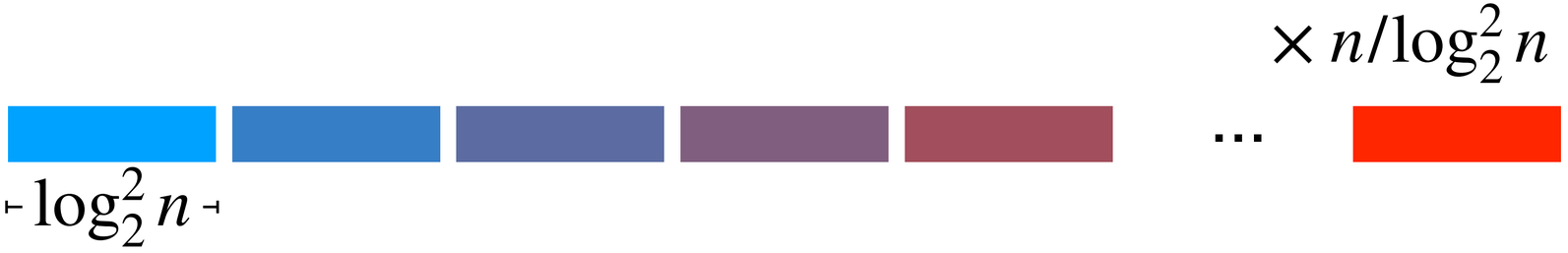
Jacobson's rank

Split the string into chunks of length $\log_2^2 n$

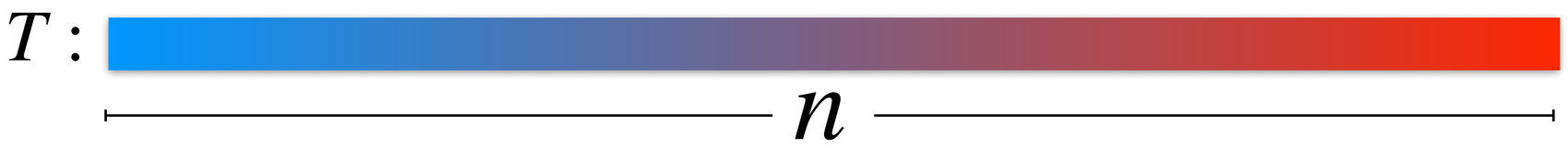
Jacobson's rank



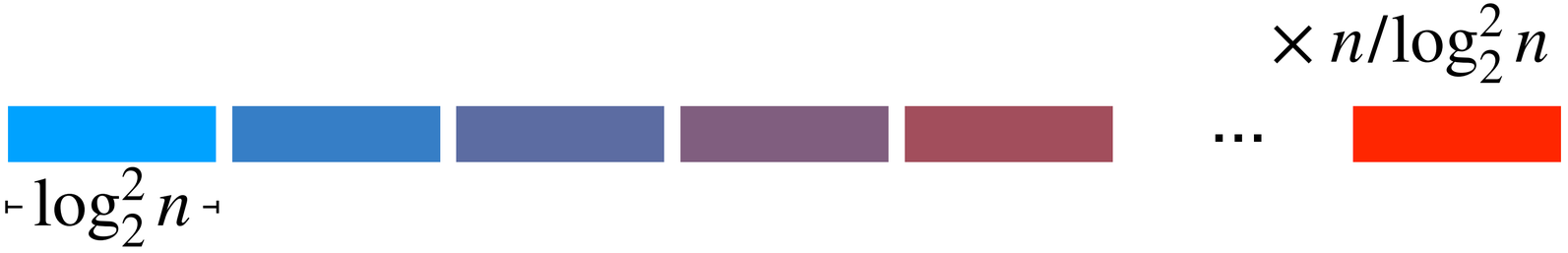
Split the string into chunks of length $\log_2^2 n$



Jacobson's rank

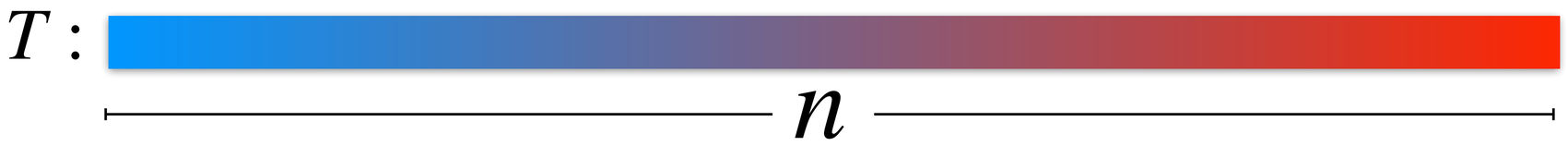


Split the string into chunks of length $\log_2^2 n$

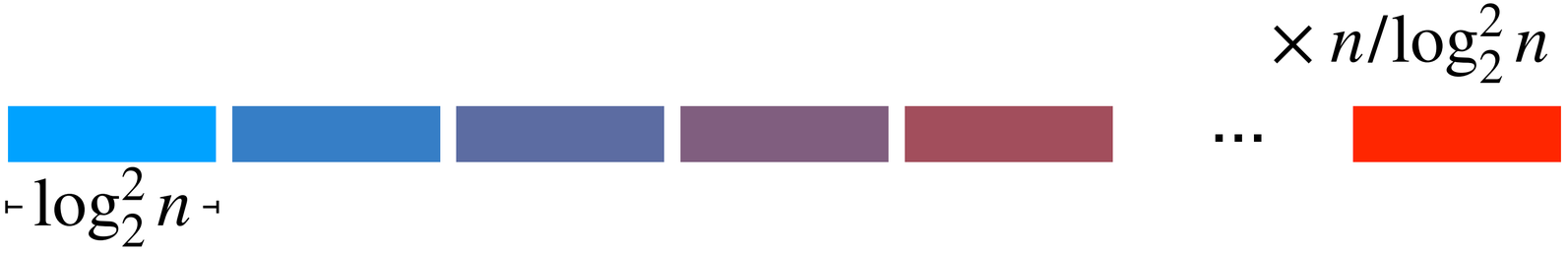


$$\log_2^2 n \equiv \left(\log_2 n\right)^2$$

Jacobson's rank



Split the string into chunks of length $\log_2^2 n$



$$\log_2^2 n \equiv \left(\log_2 n\right)^2$$

I'll omit base-2 from logs from now on

Jacobson's rank

Jacobson's rank

Store pre-calculated cumulative rank up to each chunk

Jacobson's rank

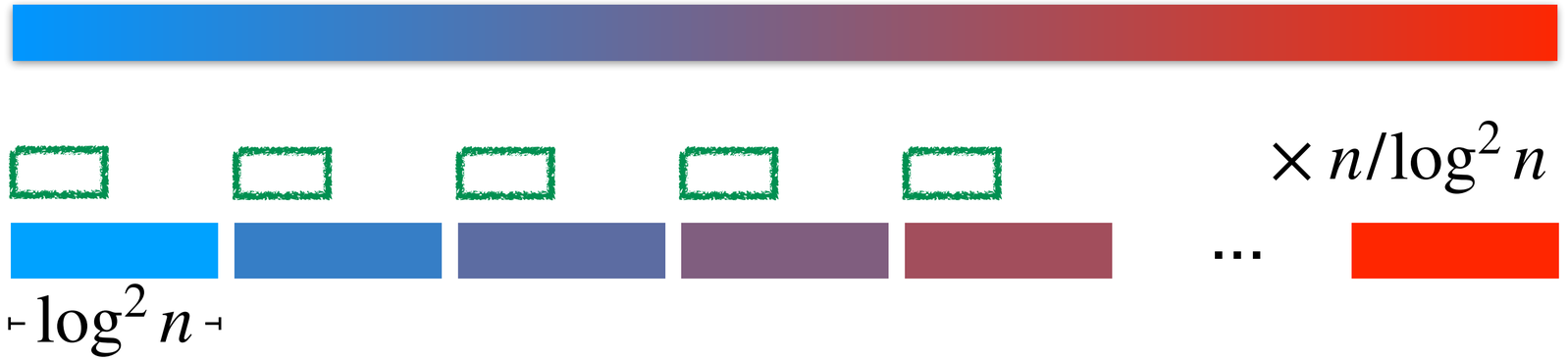
Store pre-calculated cumulative rank up to each chunk

Jacobson's rank

Store pre-calculated cumulative rank up to each chunk

$$O\left(\log n \cdot n/\log^2 n\right) = O\left(n/\log n\right) = \check{o}(n)$$

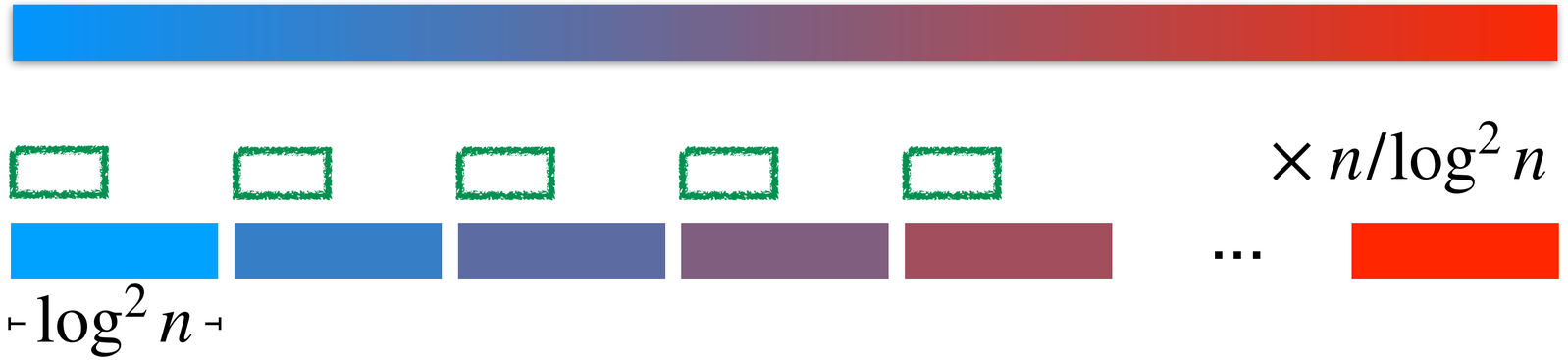
Jacobson's rank



Store pre-calculated cumulative rank up to each chunk

$$O\left(\underbrace{\log n}_{\text{bits to store cum. rank}} \cdot n/\log^2 n\right) = O\left(n/\log n\right) = \check{o}(n)$$

Jacobson's rank



Store pre-calculated cumulative rank up to each chunk

$$O\left(\underbrace{\log n}_{\text{bits to store cum. rank}} \cdot \underbrace{n/\log^2 n}_{\text{\# chunks}}\right) = O\left(n/\log n\right) = \check{o}(n)$$

Jacobson's rank

So far, extra space is $o(n)$

Jacobson's rank

So far, extra space is
$o(n)$

Finding a rank can be decomposed:

Jacobson's rank

So far, extra space is
$o(n)$

Finding a rank can be decomposed:

(a) find what chunk it's in (division)

Jacobson's rank

So far, extra space is
$\check{o}(n)$

Finding a rank can be decomposed:

(a) find what chunk it's in (division)

(b) look up cumulative rank

Jacobson's rank

So far, extra space is
$\check{o}(n)$

Finding a rank can be decomposed:

(a) find what chunk it's in (division)

(b) look up cumulative rank

(c) find (relative) rank *within* chunk

Jacobson's rank

So far, extra space is
$\check{o}(n)$

Finding a rank can be decomposed:

(a) find what chunk it's in (division)

(b) look up cumulative rank

(c) find (relative) rank *within* chunk

(d) add (b) + (c)

Jacobson's rank

So far, extra space is
$\check{o}(n)$

Finding a rank can be decomposed:

(a) find what chunk it's in (division)

(b) look up cumulative rank

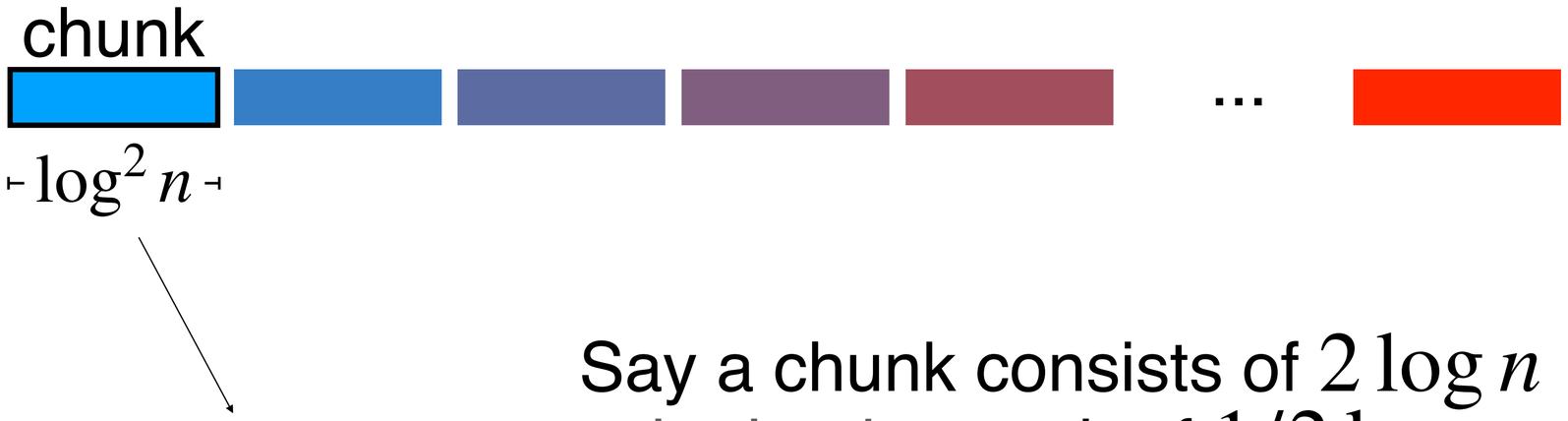
(c) find (relative) rank *within* chunk

(d) add (b) + (c)

TODO

Jacobson's rank

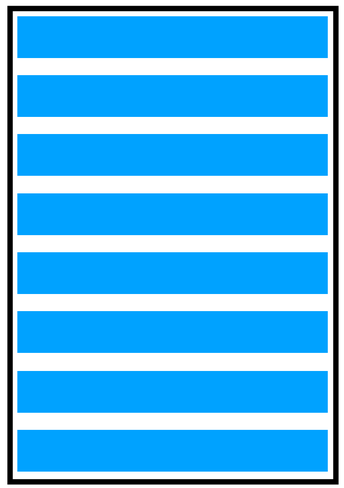
Jacobson's rank



Say a chunk consists of $2 \log n$ sub-chunks, each of $1/2 \log n$ bits

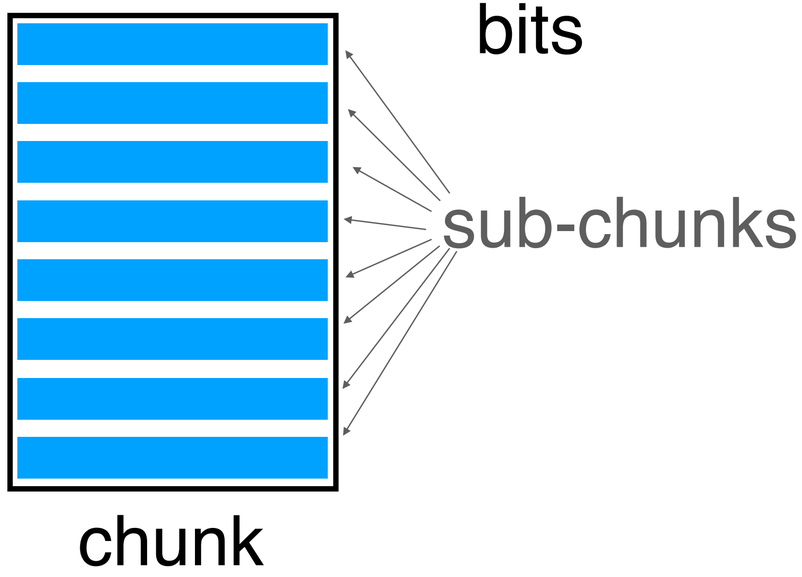
Jacobson's rank

Say a chunk consists of $2 \log n$ sub-chunks, each of $1/2 \log n$ bits

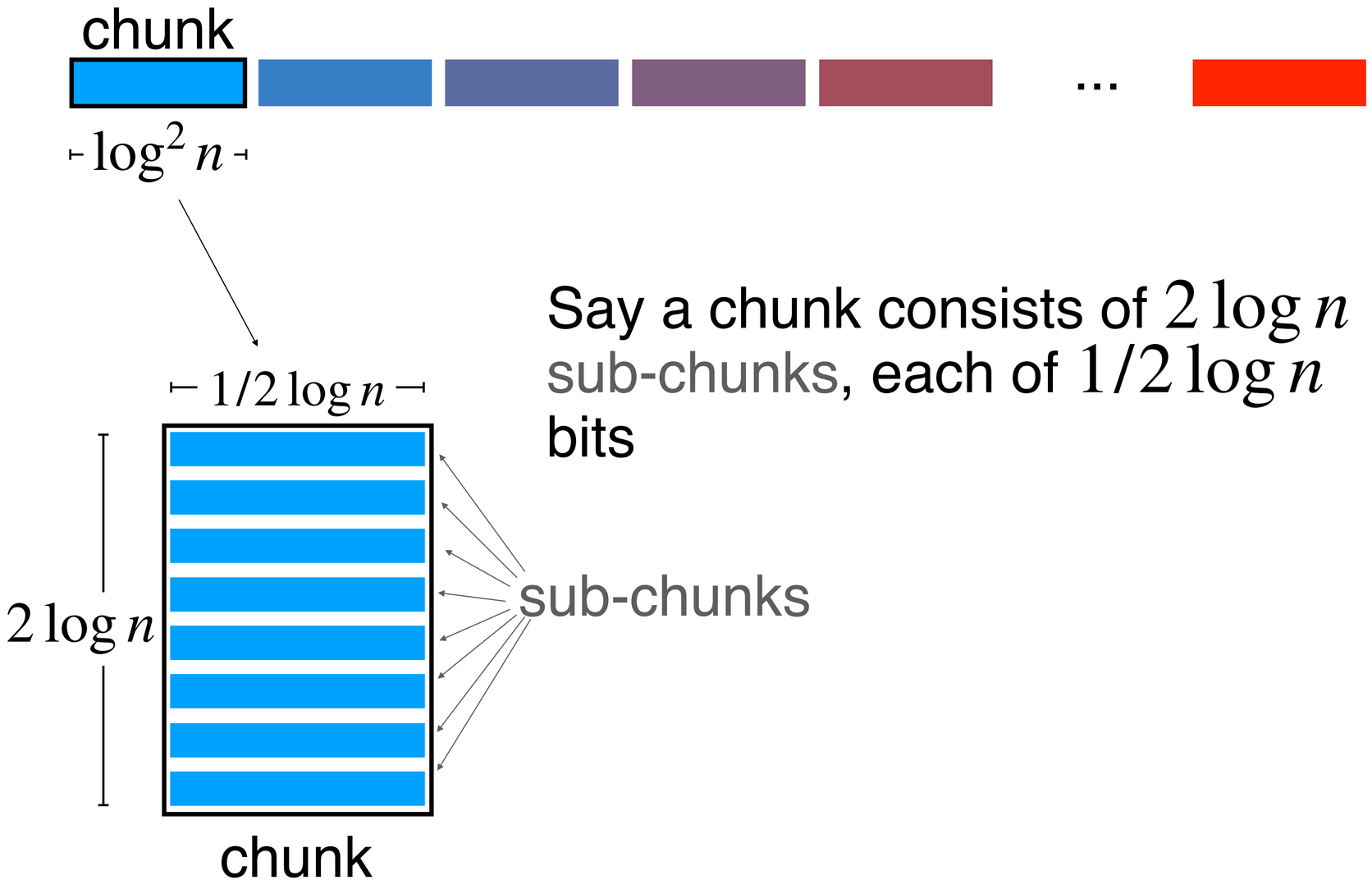


Jacobson's rank

Say a chunk consists of $2 \log n$ sub-chunks, each of $1/2 \log n$ bits

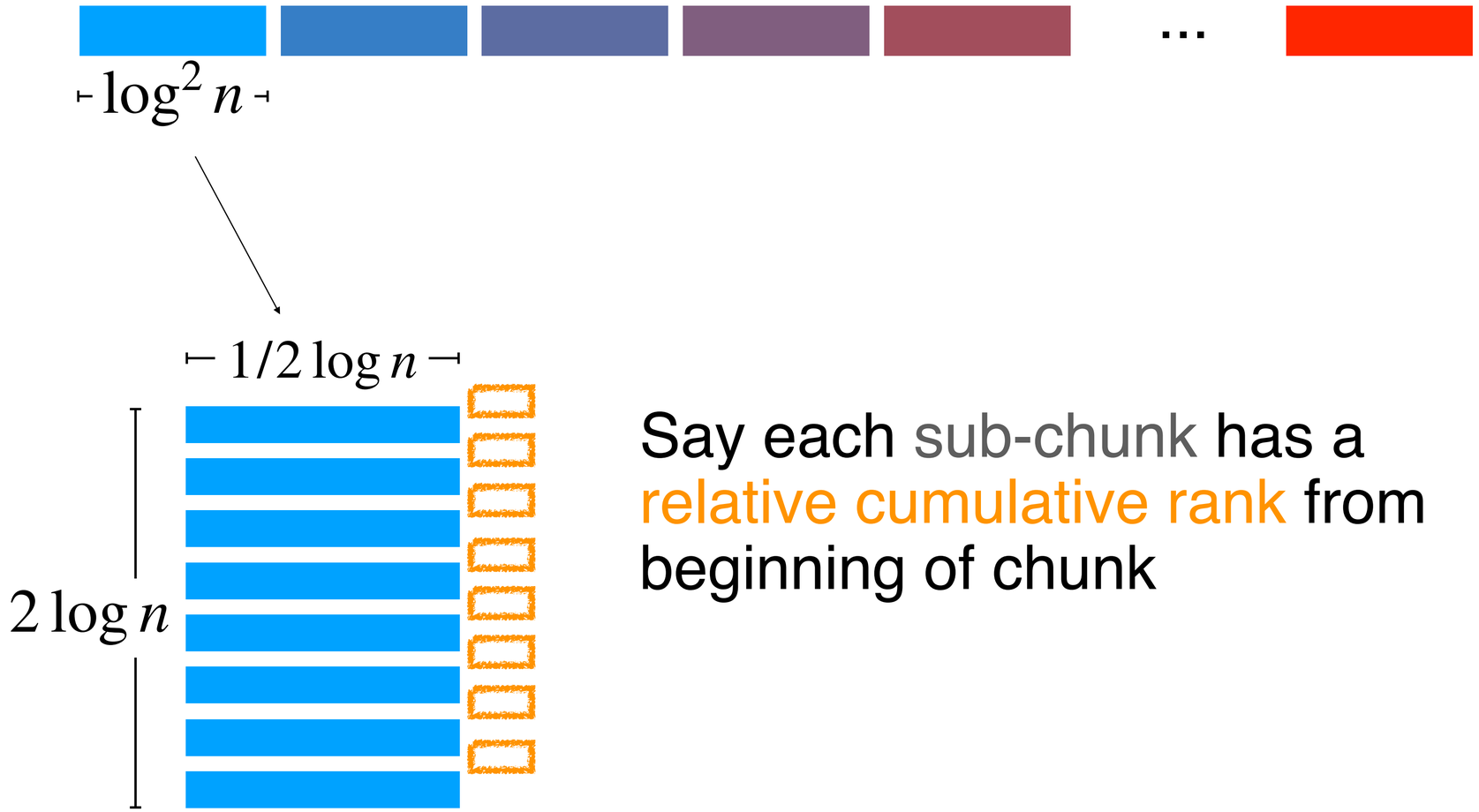


Jacobson's rank



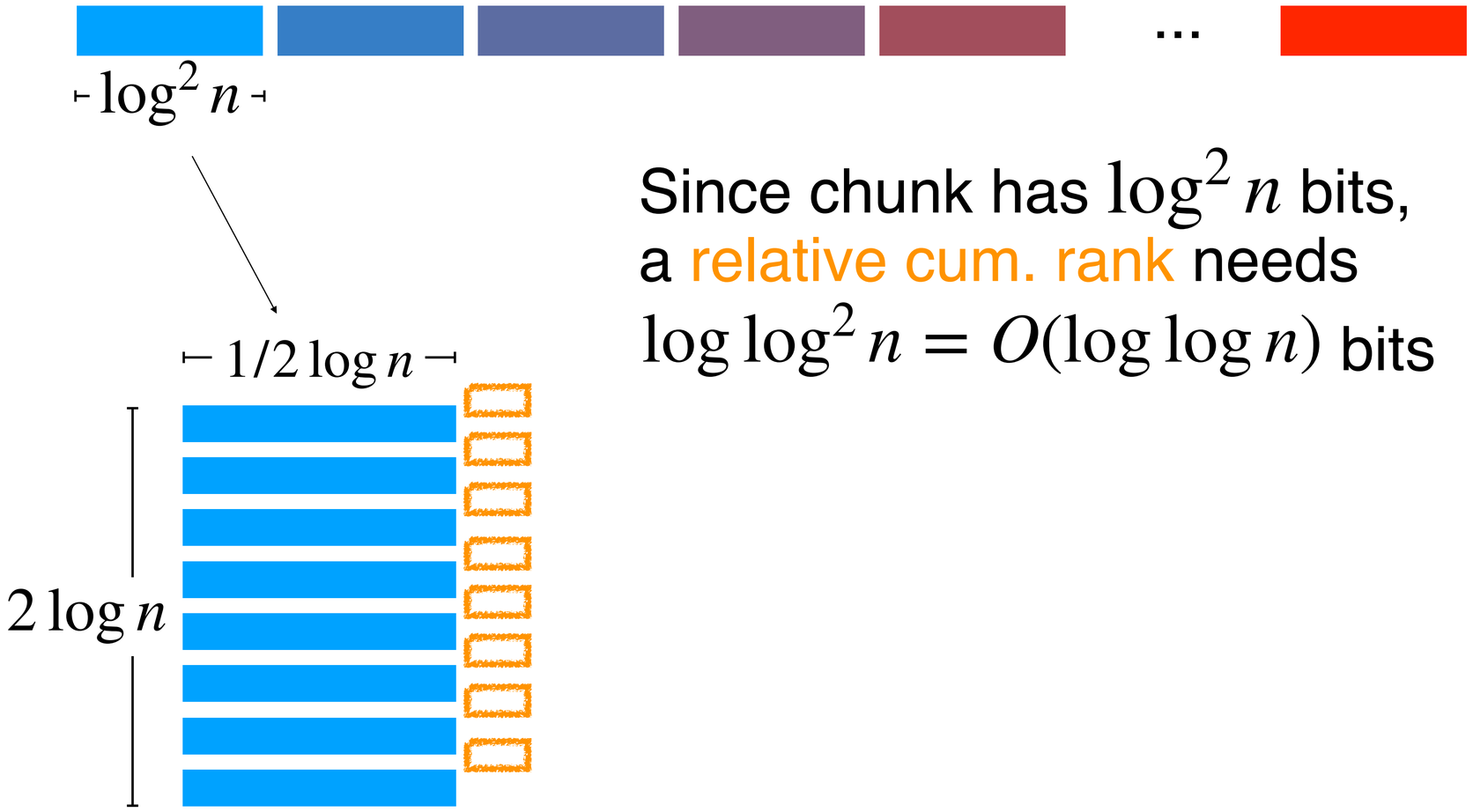
Say a chunk consists of $2 \log n$ sub-chunks, each of $1/2 \log n$ bits

Jacobson's rank



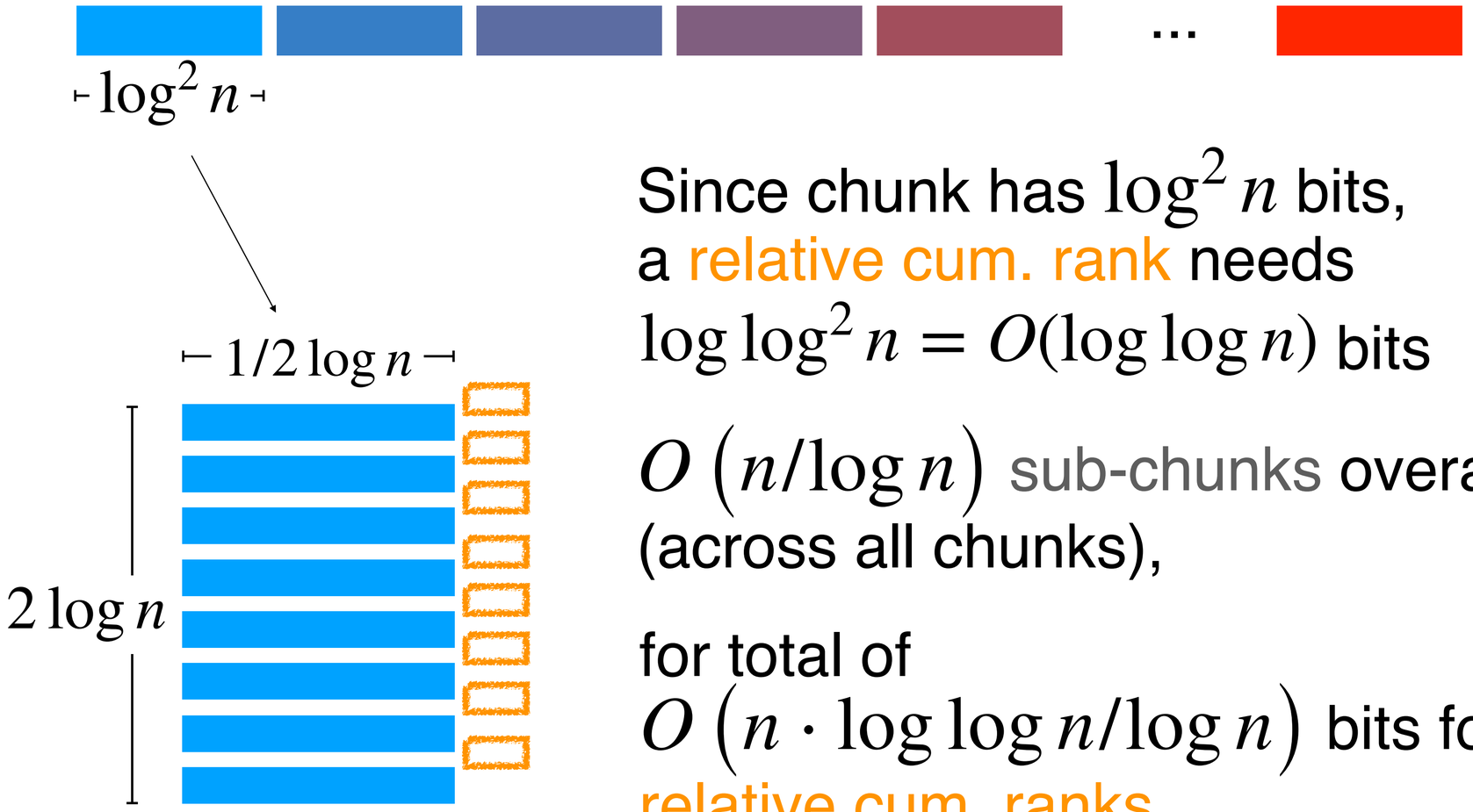
Say each sub-chunk has a relative cumulative rank from beginning of chunk

Jacobson's rank



Since chunk has $\log^2 n$ bits,
a relative cum. rank needs
$\log \log^2 n = O(\log \log n)$ bits

Jacobson's rank

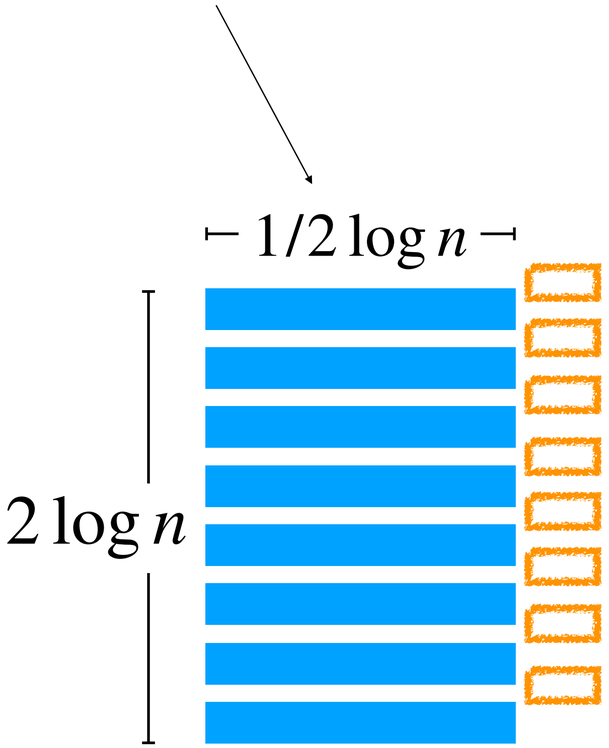


Since chunk has $\log^2 n$ bits,
a relative cum. rank needs
$\log \log^2 n = O(\log \log n)$ bits

$O\left(n/\log n\right)$ sub-chunks overall
(across all chunks),

for total of
$O\left(n \cdot \log \log n/\log n\right)$ bits for
relative cum. ranks

Jacobson's rank



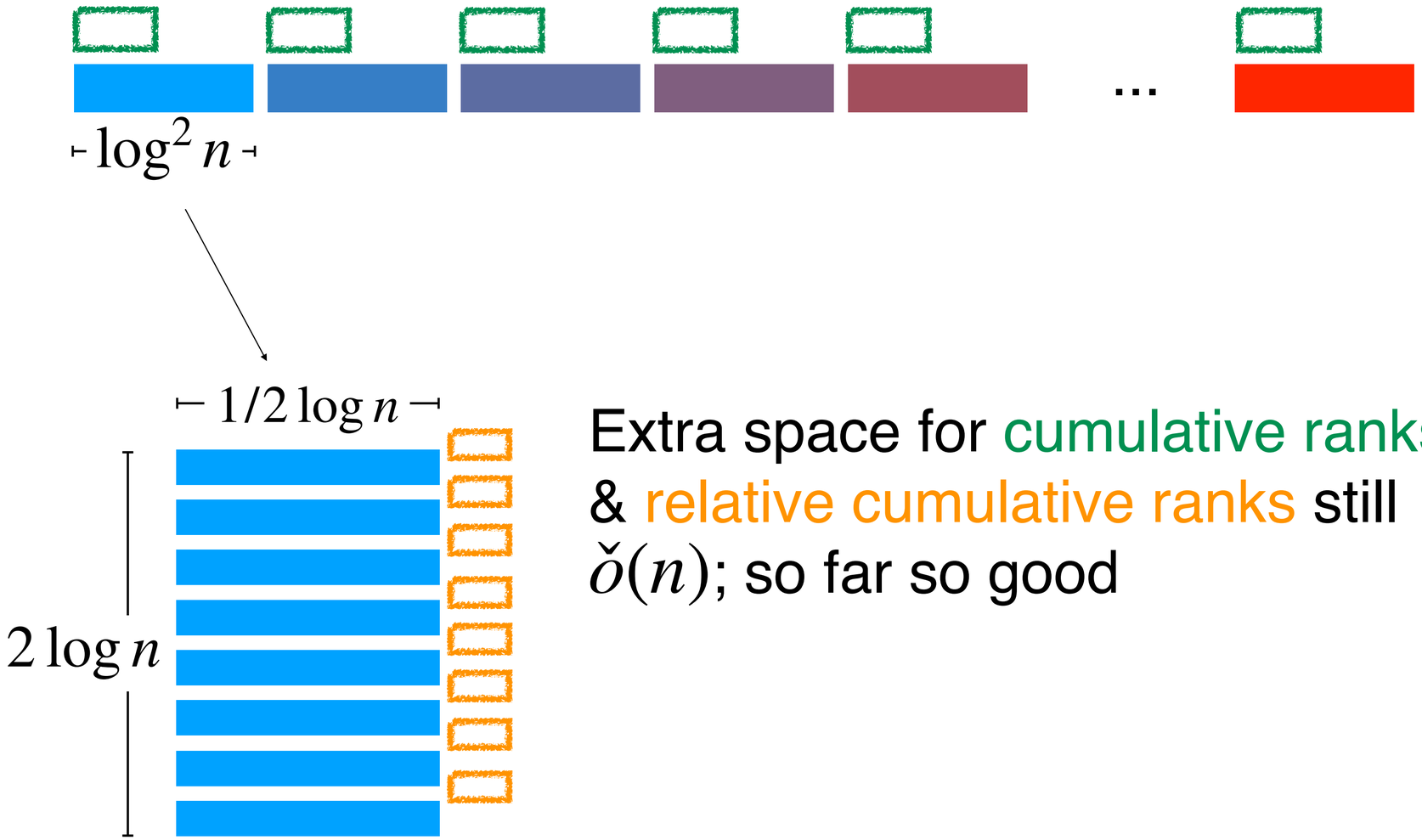
Since chunk has $\log^2 n$ bits,
a relative cum. rank needs
$\log \log^2 n = O(\log \log n)$ bits

$O\left(n/\log n\right)$ sub-chunks overall
(across all chunks),

for total of
$O\left(n \cdot \log \log n/\log n\right)$ bits for
relative cum. ranks

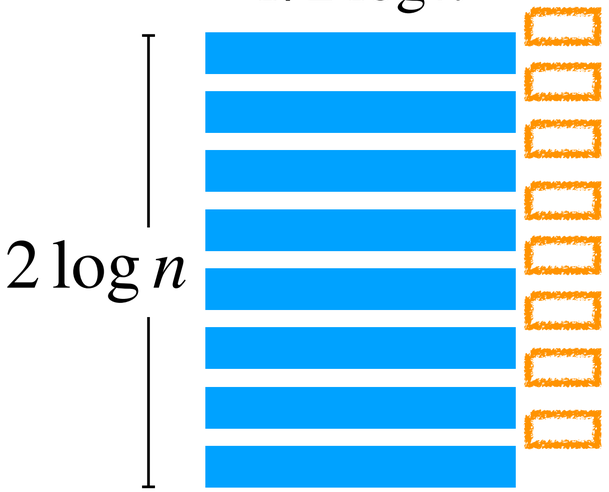
$O\left(n \cdot \log \log n/\log n\right) = \check{o}(n)$

Jacobson's rank



Extra space for cumulative ranks & relative cumulative ranks still $\check{o}(n)$; so far so good

Jacobson's rank



Finding a rank:

(a) find what chunk it's in (division)

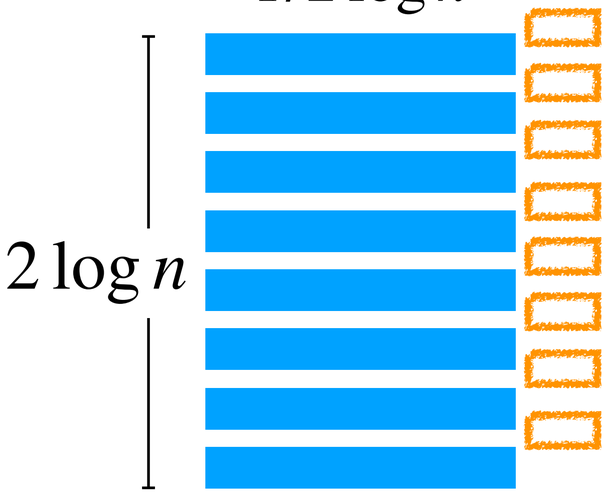
(b) look up cumulative rank

(c) find rank *within* chunk

- **(c.i)** find what sub-chunk it's in
- **(c.ii)** look up relative cum. rank
- **(c.iii)** find rank *within* sub-chunk

(d) add (b) + (c.ii) + (c.iii)

Jacobson's rank



Finding a rank:

(a) find what chunk it's in (division)

(b) look up cumulative rank

(c) find rank *within* chunk

- **(c.i)** find what sub-chunk it's in
- **(c.ii)** look up relative cum. rank
- **(c.iii)** find rank *within* sub-chunk

(d) add (b) + (c.ii) + (c.iii)

TODO

Jacobson's rank

Finding rank *within a sub-chunk*:
two ways of thinking

Jacobson's rank

Finding rank *within a sub-chunk*: two ways of thinking

Way 1: $1/2 \log n$ is ~ a machine word; use instructions like "population count" to find rank in $O(1)$ time

Jacobson's rank

Finding rank *within a sub-chunk*: two ways of thinking

Way 1: $1/2 \log n$ is ~ a machine word; use instructions like "population count" to find rank in $O(1)$ time

Way 2: Lookup table

(Next slide)

Jacobson's rank

Say we naively store answers to all rank queries for all length-x bitvectors. How many bits required?

$$2^x$$

possible bitvectors

Jacobson's rank

Say we naively store answers to all rank queries for all length-x bitvectors. How many bits required?

$$2^x \cdot x$$

possible bitvectors

possible offsets

Jacobson's rank

Say we naively store answers to all rank queries for all length-x bitvectors. How many bits required?

$$2^x \cdot x \cdot \log x$$

possible bitvectors

possible offsets

answer

Jacobson's rank

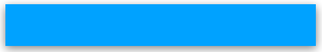
Say we naively store answers to all rank queries for all length-x bitvectors. How many bits required?

$$2^x \cdot x \cdot \log x$$

possible bitvectors · possible offsets · answer

sub-chunk

$\vdash 1/2 \log n \dashv$



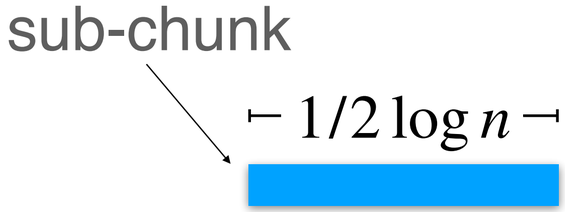
Let $x = 1/2 \log n$

Jacobson's rank

Say we naively store answers to all rank queries for all length-x bitvectors. How many bits required?

$$2^x \cdot x \cdot \log x$$

possible bitvectors · possible offsets · answer



Let $x = 1/2 \log n$

$$2^{1/2 \log n} \cdot 1/2 \log n \cdot \log 1/2 \log n = O\left(\sqrt{n} \log n \log \log n\right)$$

$$= \check{o}(n)$$

Storing an "all query" lookup table

- Blocks have size b = $\log_2 n/2$
  - There are 2^b such blocks possible
  - In each block there are b possible rank queries
  - Each answer (relative to the block) is in the range 1..b

R_p

Type	0	1	rank(0)	rank(1)
0	0	0	0	0
1	0	1	0	1
2	1	0	1	1
3	1	1	1	2

- Therefore size of R_p, the in-block data structure is
  - $2^b * b * \log b = n^{1/2} * \log n * \log\log n/2$ bits = o(n) bits

credit to Simon J. Puglisi, University of Helsinki

Jacobson's rank

Finding a rank:

(a) find what chunk it's in (division)

(b) look up checkpoint

(c) find rank *within* chunk

- **(c.i)** find what sub-chunk it's in
- **(c.ii)** look up relative checkpoint
- **(c.iii)** find rank *within* sub-chunk

(d) add (b) + (c.ii) + (c.iii)

Jacobson's rank

Finding a rank:

(a) find what chunk it's in (division)

(b) look up checkpoint

(c) find rank *within* chunk

- **(c.i)** find what sub-chunk it's in
- **(c.ii)** look up relative checkpoint
- **(c.iii)** find rank *within* sub-chunk

(d) add (b) + (c.ii) + (c.iii)

$O(1)$

Bitvectors

	Time	Space (bits)	Note
B . access	$O(1)$	n	Lookup
B . select_1	$O(1)$	$\check{o}(n)$	
B . rank_1	$O(1)$	$\check{o}(n)$	✅ Jacobson

Bitvectors

	Time	Space (bits)	Note
B . access	$O(1)$	n	Lookup
B . select_1	$O(1)$	$\check{o}(n)$	? 🧚 🦄 🧚 ?
B . rank_1	$O(1)$	$\check{o}(n)$	✅ Jacobson

More details on rank

- Rank index takes O(nloglogn/logn) = o(n) bits so we use n + o(n) overall and can answer queries in O(1) time

- While it is sublinear, we'd still like the o(n) term to be small
  - Best is by Patrastcu: $O(n/\log^k n)$ bits, O(k) time queries

- Dynamic solutions exist
  - Queries no longer constant: O(logn/loglogn) time (Raman et al.)

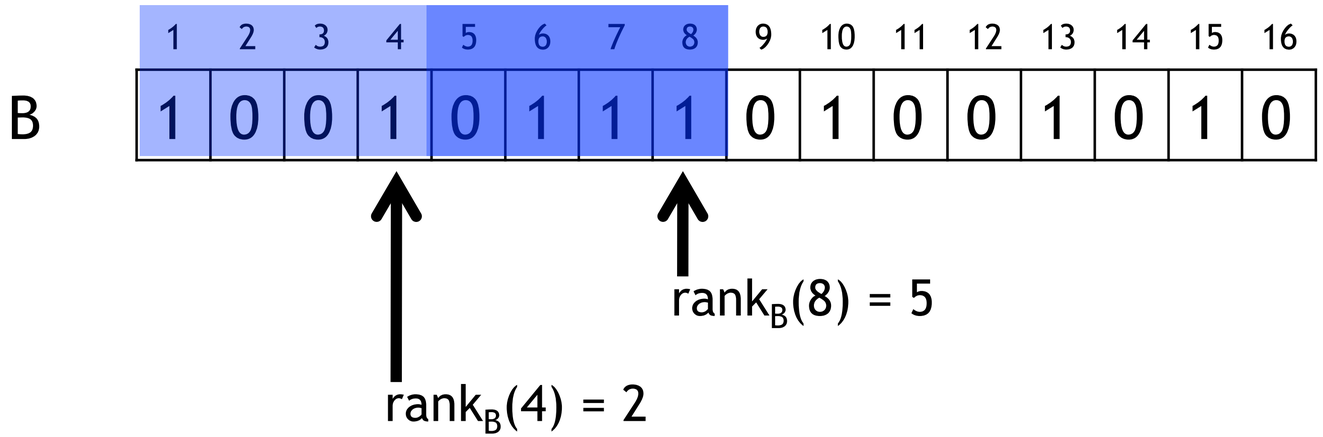
credit to Simon J. Puglisi, University of Helsinki

Select in O(lg n) via rank

- We can use our solution to rank to get a (fairly) efficient solution to select(i), with this observation:

- If rank(n/2) > i, then the i^{th} 1-bit is in B[1..n/2]
  - Otherwise it is in B[n/2+1..n]

$select_B(3)$



Other solutions for rank & select

- Applying this idea recursively to arrive at select(i)
  - $O(\log_2 n)$ time, o(n) space
- O(1) time, o(n) space solutions for select also exist
  - Slightly more complicated than O(1) rank
  - (Munro and Clark)
- Similar variations as we discussed with rank (trading space for query time) are also possible

Information Systems 73 (2018) 25–34

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/is

Rank and select: Another lesson learned

Szymon Grabowski*, Marcin Raniszewski

Lodz University of Technology, Institute of Applied Computer Science, Al. Politechniki 11, Łódź 90–924, Poland

credit to Simon J. Puglisi, University of Helsinki

Clark's select

Clark's select

Unlike rank:

Clark's select

Unlike rank:

Chunks are defined by # 1s, not # bits

Clark's select

Unlike rank:

Chunks are defined by # 1s, not # bits

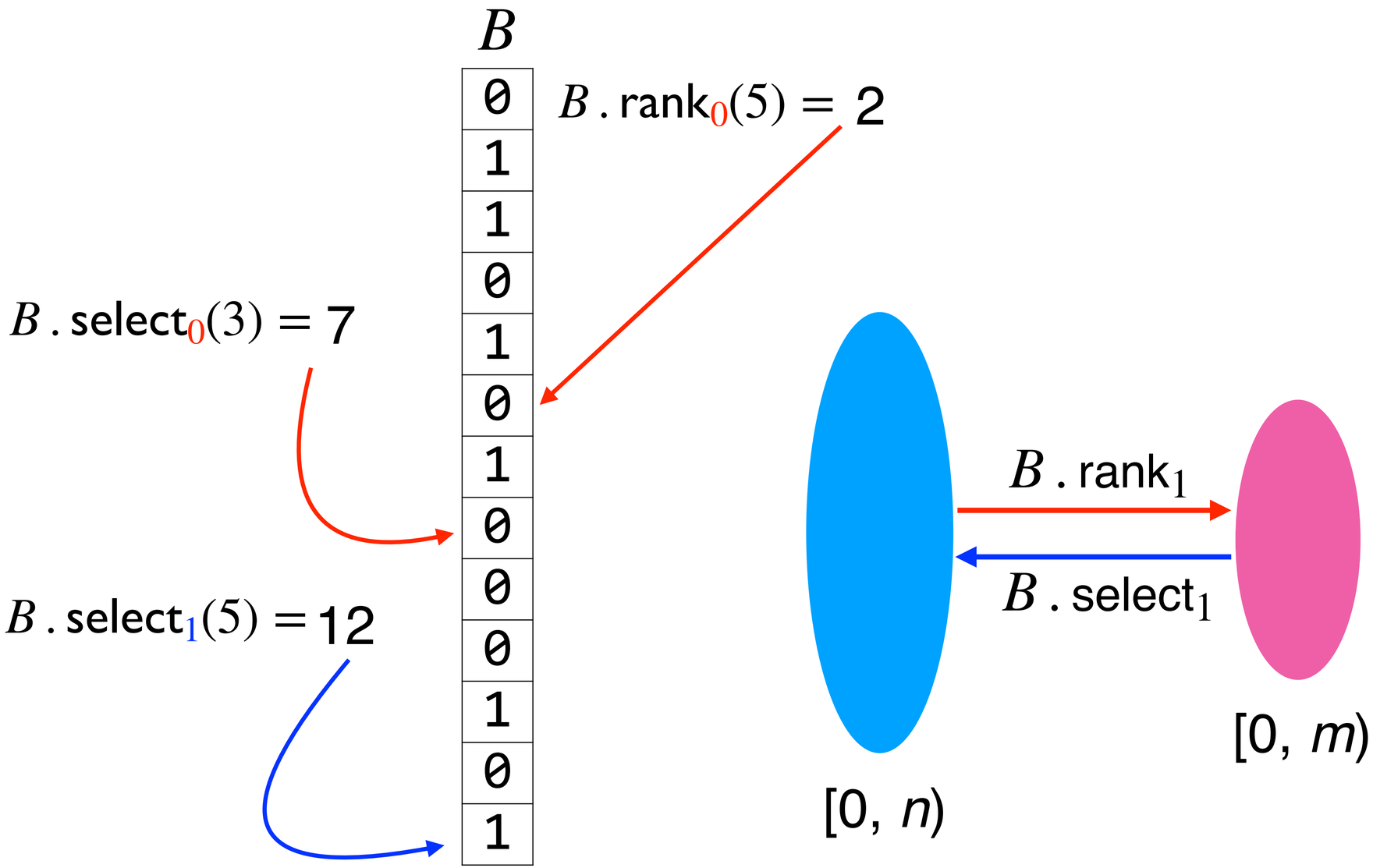
Two layers of “sparsity”

Clark's select

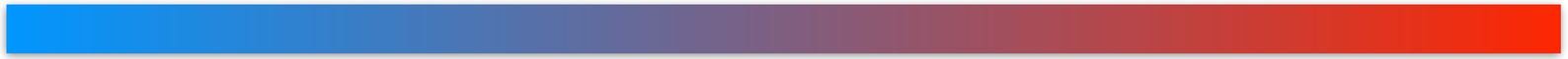
Unlike rank:

- Chunks are defined by # 1s, not # bits
- Two layers of “sparsity”
- Answer is an offset into bitvector

Bitvectors


B
0
1
1
0
1
0
1
0
0
0
1
0
1
B . rank$_0(5) = 2$
B . select$_0(3) = 7$
B . select$_1(5) = 12$
B . rank$_1$
B . select$_1$
[0, n)
[0, m)

Clark's select



Split into $\log^2 n$-weight chunks

Sparse ($\geq \log^4 n$-length)

Lookup table for each 1-bit ✅

Dense ($< \log^4 n$-length)

Split into $\sqrt{\log n}$-weight sub-chunks

Sparse ($\geq 1/2 \log n$-length)

Lookup table for each 1-bit ✅

Dense ($< 1/2 \log n$-length)

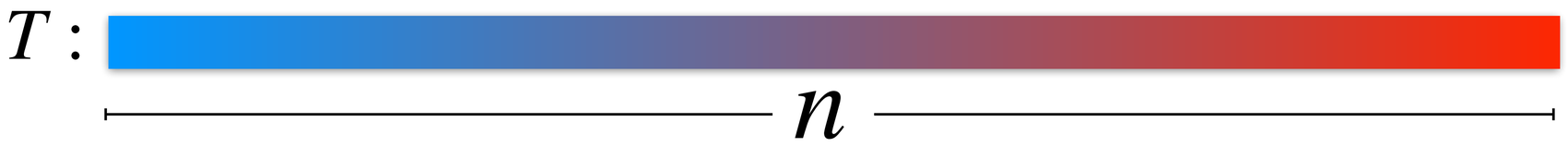
Lookup table for all possible sub-chunks ✅

Clark's select

Split the string into chunks each containing

$\log^2 n$ 1-bits

Clark's select



Split the string into chunks each containing

$\log^2 n$ 1-bits

Clark's select

Split the string into chunks each containing

$\log^2 n$ 1-bits

Larger chunks are sparse; 1’s spread out

Clark's select

Split the string into chunks each containing

$\log^2 n$ 1-bits

Larger chunks are sparse; 1's spread out

Shorter chunks are dense; 1's packed together

Clark's select

Each chunk contains $\log^2 n$ 1-bits

Worst case:
every bit set

Clark's select

Each chunk contains $\log^2 n$ 1-bits

We store offset of each chunk start

Worst case:
every bit set

Clark's select

Each chunk contains $\log^2 n$ 1-bits

We store offset of each chunk start

Worst case:
every bit set

Clark's select

Each chunk contains $\log^2 n$ 1-bits

We store offset of each chunk start

This takes:

Worst case:
every bit set

Clark's select

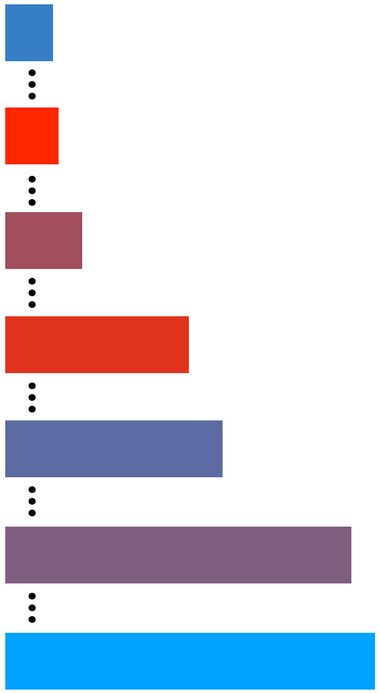
Each chunk contains $\log^2 n$ 1-bits

We store offset of each chunk start

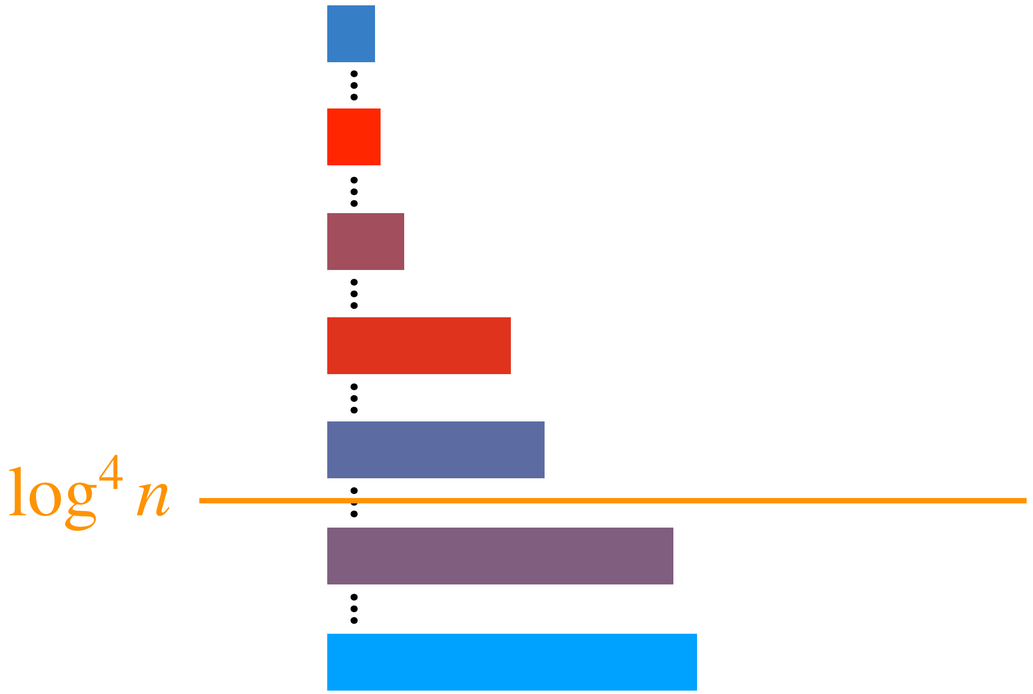
This takes:

$$O\left(\underbrace{\frac{n}{\log^2 n}}_{\text{Worst case: every bit set}} \log n\right) = O\left(\frac{n}{\log n}\right) = \check{o}(n) \text{ bits}$$

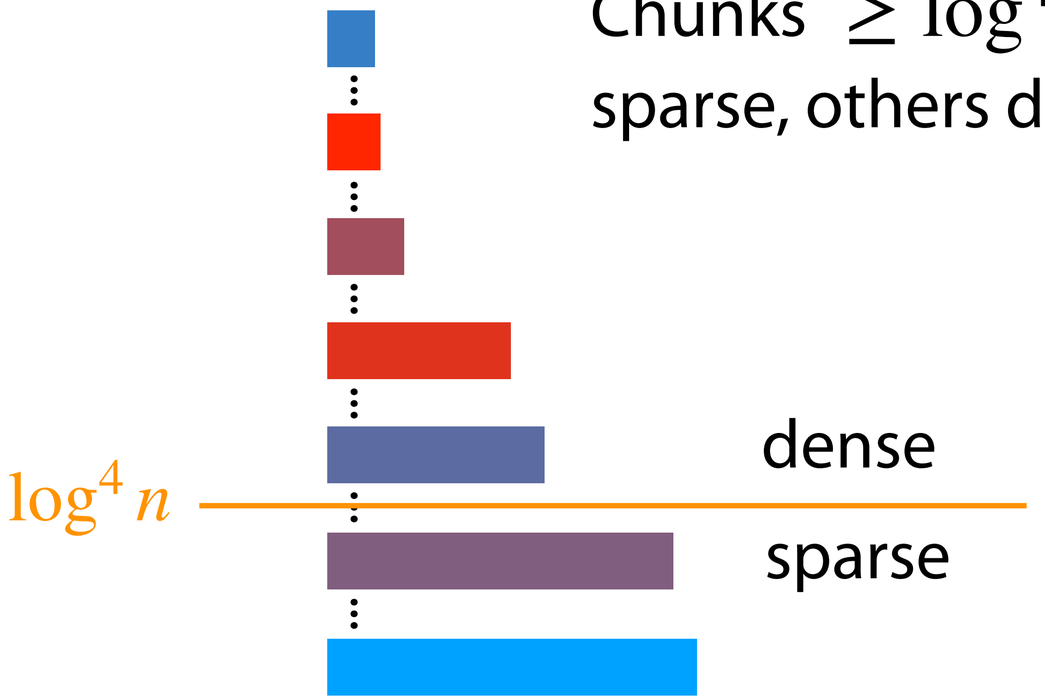
Clark's select



Clark's select

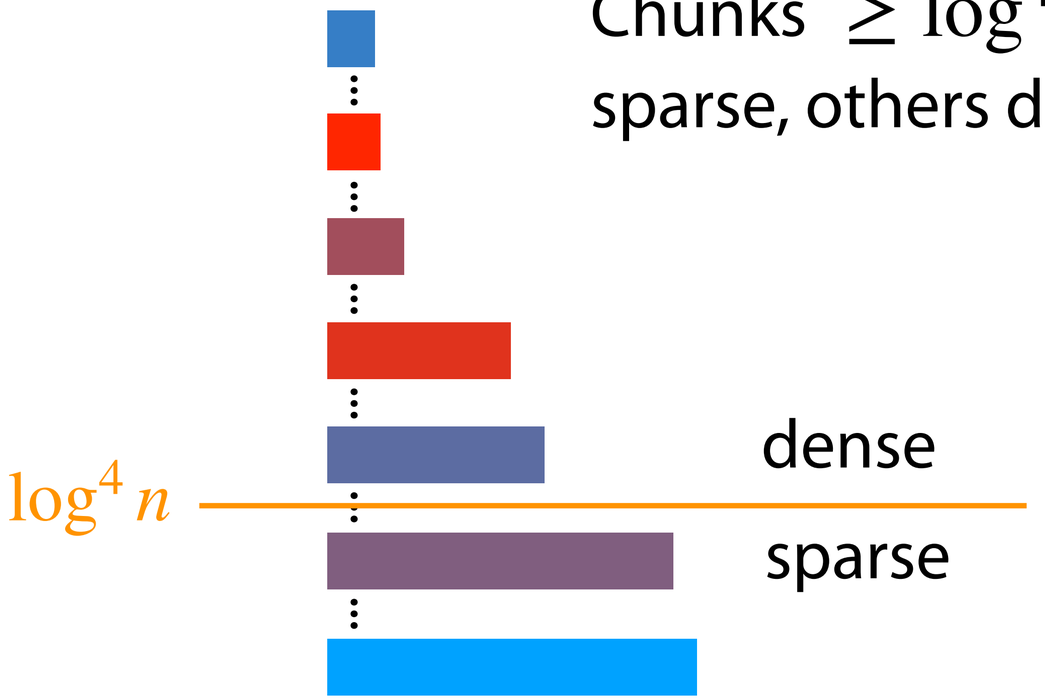


Clark's select



Chunks $\geq \log^4 n$ bits in length are sparse, others dense

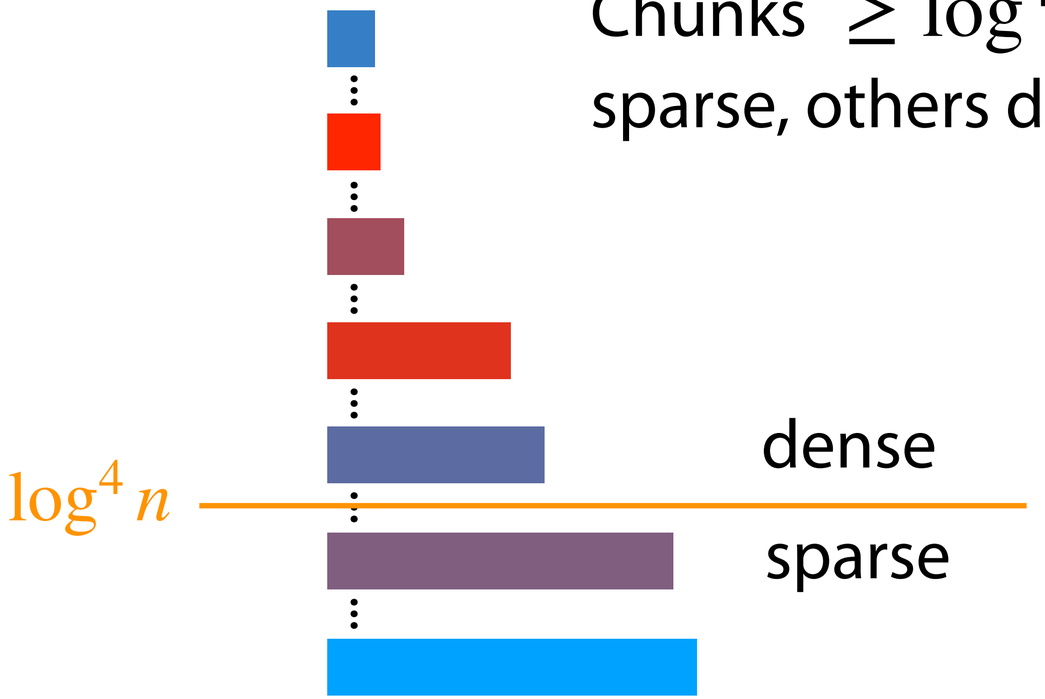
Clark's select



Chunks $\geq \log^4 n$ bits in length are sparse, others dense

$\log^4 n$ is square of the # of set bits per chunk, $\log^2 n$

Clark's select



Chunks $\geq \log^4 n$ bits in length are sparse, others dense

$\log^4 n$ is square of the # of set bits per chunk, $\log^2 n$

("sparse" roughly means "less than $\sqrt{\#\text{ bits}}$ are set")

Clark's select: sparse case

Pre-calculate B . select_1 for 1-bits in sparse chunks

Clark's select: sparse case

Pre-calculate B . select_1 for 1-bits in sparse chunks

$$O\left(\frac{n}{\log^4 n} \cdot \log n \cdot \log^2 n\right)$$

Clark's select: sparse case

Pre-calculate B . select_1 for 1-bits in sparse chunks

$$O\left(\underbrace{\frac{n}{\log^4 n}}_{\text{Max \# sparse chunks}} \cdot \log n \cdot \log^2 n\right)$$

Clark's select: sparse case

Pre-calculate B . select_1 for 1-bits in sparse chunks

$$O\left(\underbrace{\frac{n}{\log^4 n}}_{\text{Max \# sparse chunks}} \cdot \underbrace{\log n}_{\text{\# bits to store 1 answer}} \cdot \log^2 n\right)$$

Clark's select: sparse case

Pre-calculate B . select_1 for 1-bits in sparse chunks

$$O\left(\underbrace{\frac{n}{\log^4 n}}_{\text{Max \# sparse chunks}} \cdot \underbrace{\log n}_{\substack{\text{\# bits to} \\ \text{store 1} \\ \text{answer}}} \cdot \underbrace{\log^2 n}_{\substack{\text{\# answers} \\ \text{per chunk}}}\right)$$

Clark's select: sparse case

Pre-calculate B . select_1 for 1-bits in sparse chunks

$$O\left(\underbrace{\frac{n}{\log^4 n}}_{\text{Max \# sparse chunks}} \cdot \underbrace{\log n}_{\substack{\text{\# bits to} \\ \text{store 1} \\ \text{answer}}} \cdot \underbrace{\log^2 n}_{\substack{\text{\# answers} \\ \text{per chunk}}}\right)$$

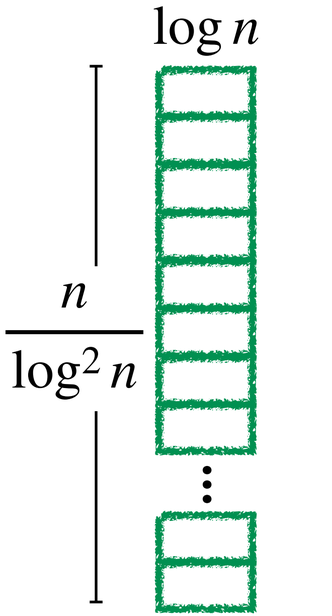
$$= O\left(\frac{n}{\log n}\right) = \check{o}(n)$$

Clark's select: sparse case

Pre-calculate B . select_1 for 1-bits in sparse chunks

$$O\left(\underbrace{\frac{n}{\log^4 n}}_{\text{Max \# sparse chunks}} \cdot \underbrace{\log n}_{\text{\# bits to store 1 answer}} \cdot \underbrace{\log^2 n}_{\text{\# answers per chunk}}\right)$$

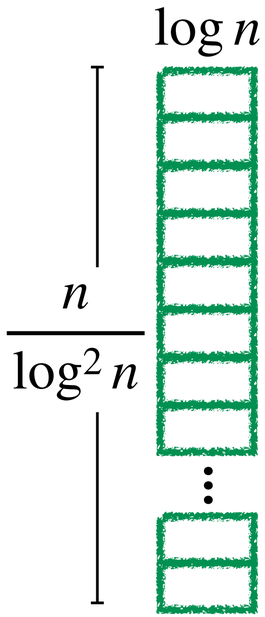
$$= O\left(\frac{n}{\log n}\right) = \check{o}(n)$$



Offsets for 1-bits in sparse chunks

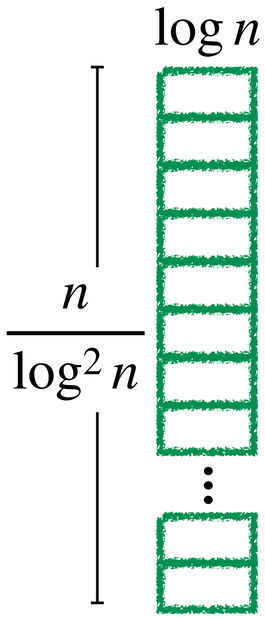
Clark's select

Offsets for chunks



Offsets for 1-bits
in sparse chunks

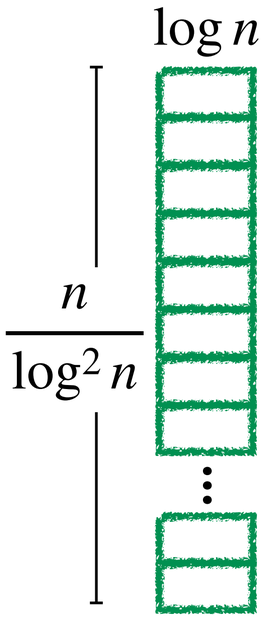
Clark's select



Offsets for 1-bits
in sparse chunks

So far, strategy for select is:

Clark's select

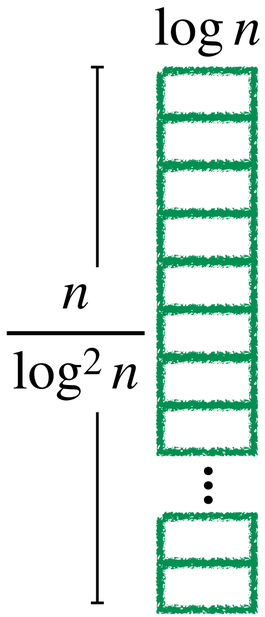


Offsets for 1-bits
in sparse chunks

So far, strategy for select is:

(a) find what chunk it's in (division)

Clark's select



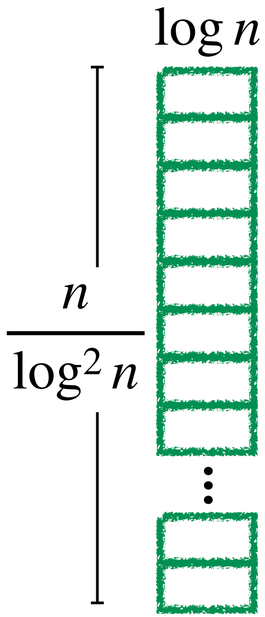
Offsets for 1-bits
in sparse chunks

So far, strategy for select is:

(a) find what chunk it's in (division)

(b) if chunk is sparse ($\geq \log^4 n$ bits)

Clark's select



Offsets for 1-bits
in sparse chunks

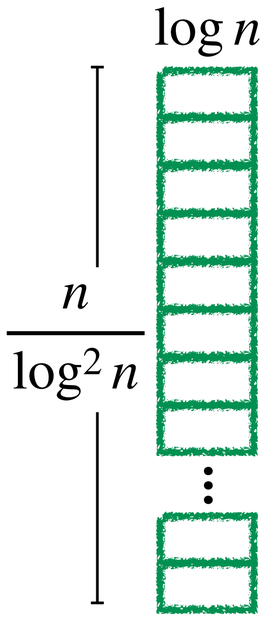
So far, strategy for select is:

(a) find what chunk it's in (division)

(b) if chunk is sparse ($\geq \log^4 n$ bits)

(b.i) look up in sparse offset table

Clark's select



Offsets for 1-bits
in sparse chunks

So far, strategy for select is:

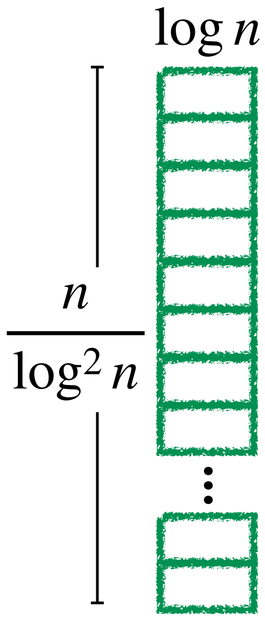
(a) find what chunk it's in (division)

(b) if chunk is sparse ($\geq \log^4 n$ bits)

(b.i) look up in sparse offset table

(c) if chunk is dense ($< \log^4 n$ bits)

Clark's select



Offsets for 1-bits
in sparse chunks

So far, strategy for select is:

(a) find what chunk it's in (division)

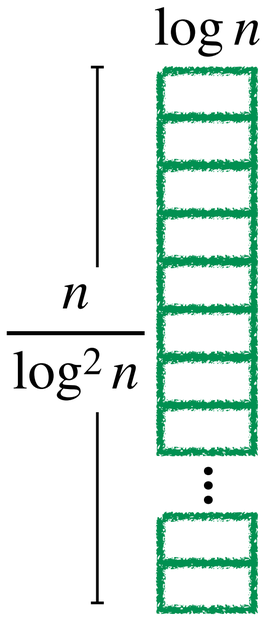
(b) if chunk is sparse ($\geq \log^4 n$ bits)

(b.i) look up in sparse offset table

(c) if chunk is dense ($< \log^4 n$ bits)

TODO

Clark's select



Offsets for 1-bits
in sparse chunks

So far, strategy for select is:

(a) find what chunk it's in (division)

(b) if chunk is sparse ($\geq \log^4 n$ bits)

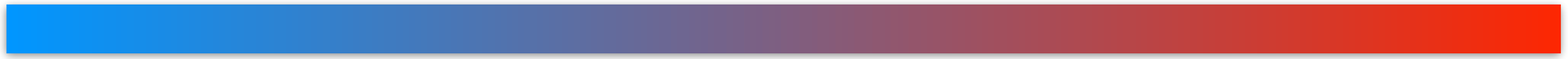
(b.i) look up in sparse offset table

(c) if chunk is dense ($< \log^4 n$ bits)

TODO

So far, space is $\check{o}(n)$

Clark's select



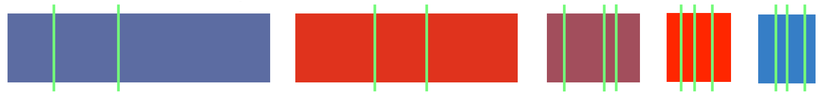
Split into $\log^2 n$-weight chunks

Sparse ($\geq \log^4 n$-length)

Lookup table for each 1-bit ✅

Dense ($< \log^4 n$-length)

Split into $\sqrt{\log n}$-weight sub-chunks



Sparse ($\geq 1/2 \log n$-length)

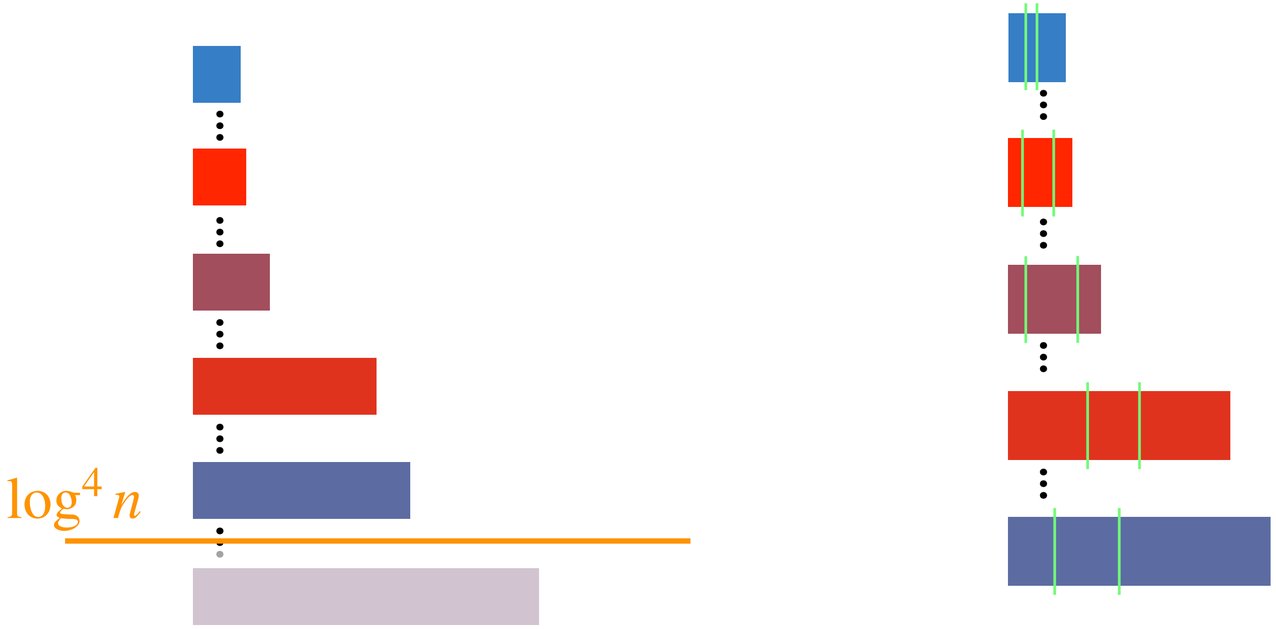
Lookup table for each 1-bit ✅

Dense ($< 1/2 \log n$-length)

Lookup table for all possible sub-chunks ✅

Clark's select: dense case

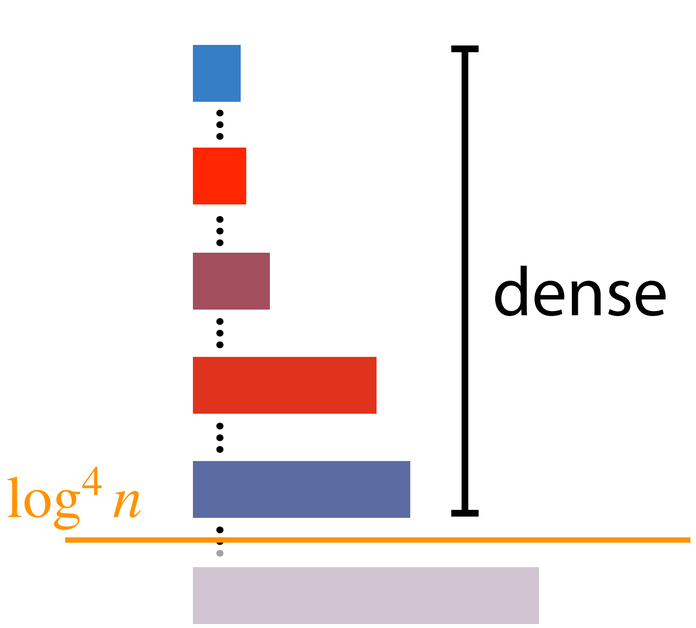
Dense chunks are shorter than $\log^4 n$ bits; further subdivide these to sub-chunks of $\sqrt{\log n}$ 1-bits each

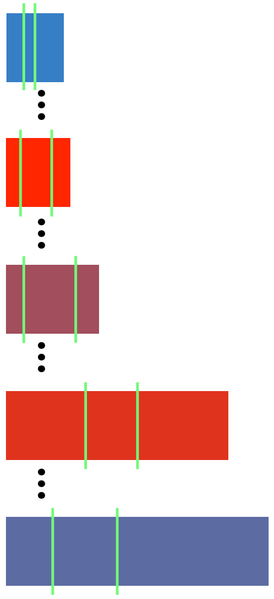


Chunks with $\log^2 n$ 1s

Clark's select: dense case

Dense chunks are shorter than $\log^4 n$ bits; further subdivide these to sub-chunks of $\sqrt{\log n}$ 1-bits each

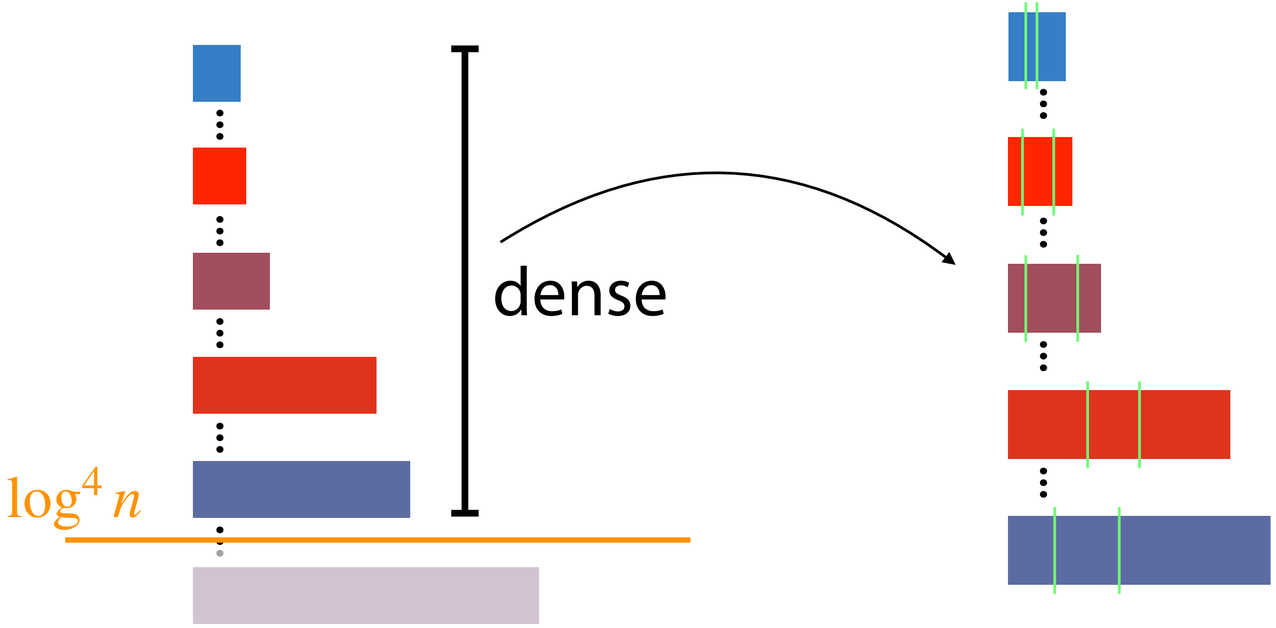




Chunks with $\log^2 n$ 1s

Clark's select: dense case

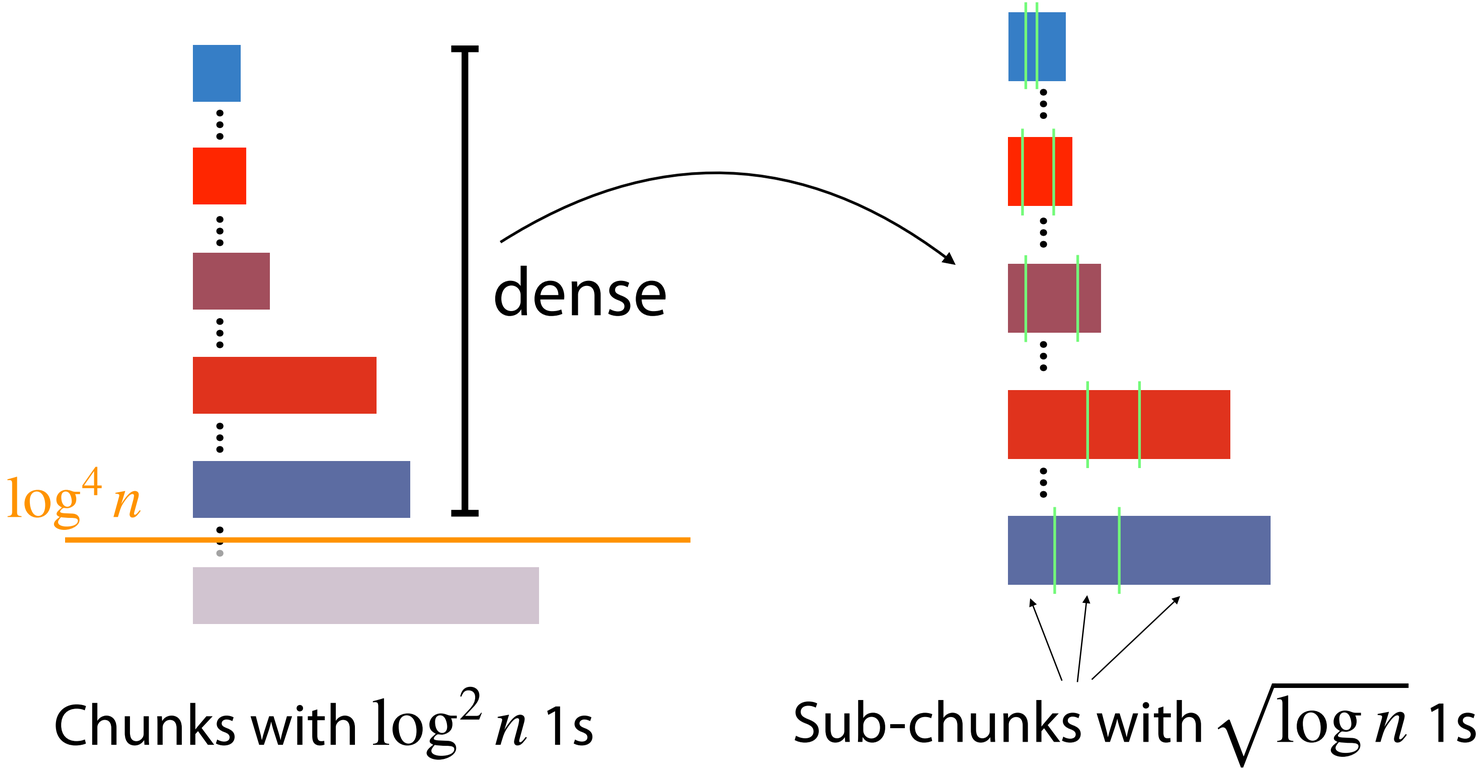
Dense chunks are shorter than $\log^4 n$ bits; further subdivide these to sub-chunks of $\sqrt{\log n}$ 1-bits each



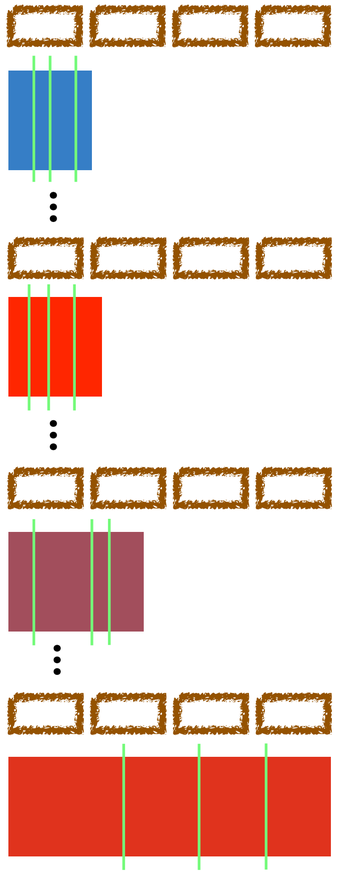
Chunks with $\log^2 n$ 1s

Clark's select: dense case

Dense chunks are shorter than $\log^4 n$ bits; further subdivide these to sub-chunks of $\sqrt{\log n}$ 1-bits each



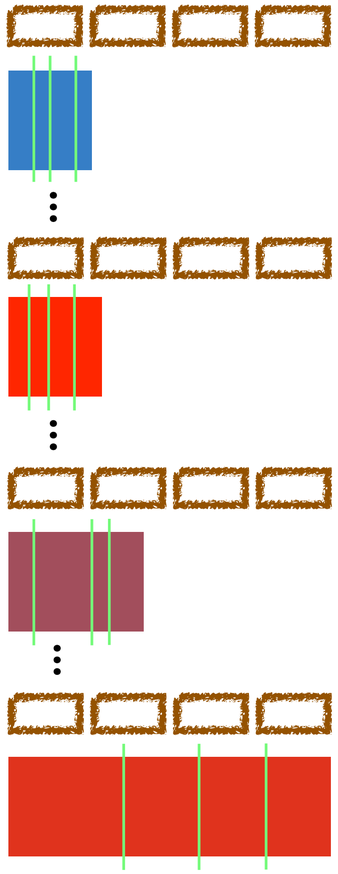
Clark's select: dense case



Store relative offset per sub-chunk

Containing chunk has $< \log^4 n$ bits,
so relative offset fits in
$O(\log \log^4 n) = O(\log \log n)$ bits

Clark's select: dense case

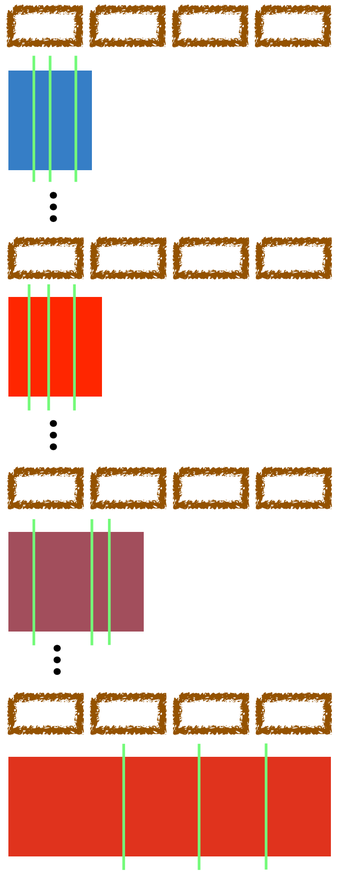


Store relative offset per sub-chunk

There are $\leq n/\sqrt{\log n}$ sub-chunks

Containing chunk has $< \log^4 n$ bits,
so relative offset fits in
$O(\log \log^4 n) = O(\log \log n)$ bits

Clark's select: dense case



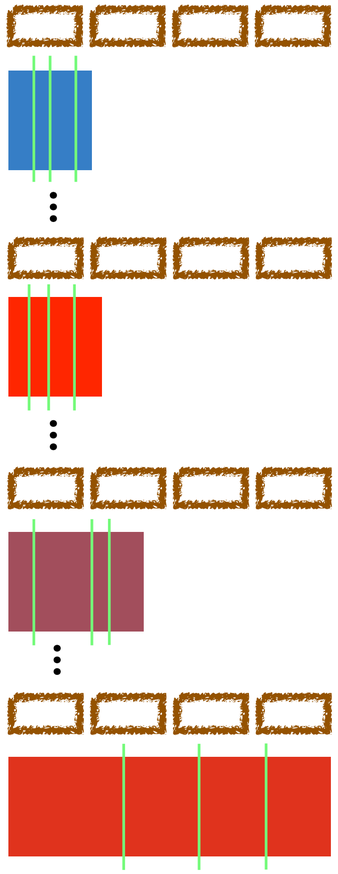
Store relative offset per sub-chunk

There are $\leq n/\sqrt{\log n}$ sub-chunks

Containing chunk has $< \log^4 n$ bits,
so relative offset fits in
$O(\log \log^4 n) = O(\log \log n)$ bits

Overall:

Clark's select: dense case



Store relative offset per sub-chunk

There are $\leq n/\sqrt{\log n}$ sub-chunks

Containing chunk has $< \log^4 n$ bits,
so relative offset fits in
$O(\log \log^4 n) = O(\log \log n)$ bits

Overall: $O\left(\frac{n \log \log n}{\sqrt{\log n}}\right) = \check{o}(n)$

Clark's select

So far, strategy for select is:

Clark's select

So far, strategy for select is:

(a) find what chunk it's in (division)

(b) if chunk is sparse

(c) if chunk is dense

Clark's select

So far, strategy for select is:

(a) find what chunk it's in (division)

(b) if chunk is sparse

(b.i) look up in sparse offset table

(c) if chunk is dense

Clark's select

So far, strategy for select is:

(a) find what chunk it's in (division)

(b) if chunk is sparse

(b.i) look up in sparse offset table

(c) if chunk is dense

(c.i) look up chunk's offset

Clark's select

So far, strategy for select is:

(a) find what chunk it's in (division)

(b) if chunk is sparse

(b.i) look up in sparse offset table

(c) if chunk is dense

(c.i) look up chunk's offset

(c.ii) find what sub-chunk it's in (division by $\sqrt{\log n}$)

Clark's select

So far, strategy for select is:

(a) find what chunk it's in (division)

(b) if chunk is sparse

(b.i) look up in sparse offset table

(c) if chunk is dense

(c.i) look up chunk's offset

(c.ii) find what sub-chunk it's in (division by $\sqrt{\log n}$)

(c.iii) look up sub-chunk's relative offset

Clark's select

So far, strategy for select is:

(a) find what chunk it's in (division)

(b) if chunk is sparse

(b.i) look up in sparse offset table

(c) if chunk is dense

(c.i) look up chunk's offset

(c.ii) find what sub-chunk it's in (division by $\sqrt{\log n}$)

(c.iii) look up sub-chunk's relative offset

TODO: need to look within sub-chunks

Clark's select: dense/sparse case

Sub-chunks with $\geq 1/2 \log n$ bits are sparse; we simply store relative offsets for every 1-bit

Clark's select: dense/sparse case

Sub-chunks with $\geq 1/2 \log n$ bits are sparse; we simply store relative offsets for every 1-bit

Overall:

Clark's select: dense/sparse case

Sub-chunks with $\geq 1/2 \log n$ bits are sparse; we simply store relative offsets for every 1-bit

Overall: $O\left(\frac{n}{1/2 \log n} \quad \log \log n \quad \sqrt{\log n}\right)$

Clark's select: dense/sparse case

Sub-chunks with $\geq 1/2 \log n$ bits are sparse; we simply store relative offsets for every 1-bit

Overall: $O\left(\underbrace{\frac{n}{1/2 \log n}}_{\text{Max \# sparse sub-chunks}} \log \log n \sqrt{\log n} \right)$

Clark's select: dense/sparse case

Sub-chunks with $\geq 1/2 \log n$ bits are sparse; we simply store relative offsets for every 1-bit

Overall: $O\left(\underbrace{\frac{n}{1/2 \log n}}_{\text{Max \# sparse sub-chunks}} \quad \underbrace{\log \log n}_{\text{\# bits to store 1 answer (rel. to chunk)}} \quad \sqrt{\log n} \right)$

Clark's select: dense/sparse case

Sub-chunks with $\geq 1/2 \log n$ bits are sparse; we simply store relative offsets for every 1-bit

Overall: $$O\left(\underbrace{\frac{n}{1/2 \log n}}_{\text{Max \# sparse sub-chunks}} \underbrace{\log \log n}_{\text{\# bits to store 1 answer (rel. to chunk)}} \underbrace{\sqrt{\log n}}_{\text{\# 1-bits per chunk}}\right)$$

Clark's select: dense/sparse case

Sub-chunks with $\geq 1/2 \log n$ bits are sparse; we simply store relative offsets for every 1-bit

Overall: $$O\left(\underbrace{\frac{n}{1/2 \log n}}_{\text{Max \# sparse sub-chunks}} \underbrace{\log \log n}_{\text{\# bits to store 1 answer (rel. to chunk)}} \underbrace{\sqrt{\log n}}_{\text{\# 1-bits per chunk}}\right)$$

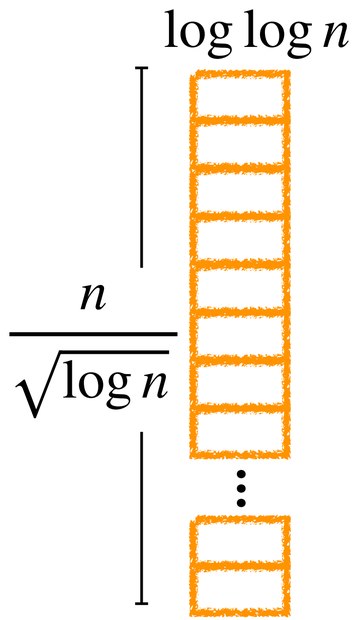
$$= O\left(\frac{n\sqrt{\log n} \log \log n}{\log n}\right) = O\left(\frac{n \log \log n}{\sqrt{\log n}}\right) = \check{o}(n)$$

Clark's select: dense/sparse case

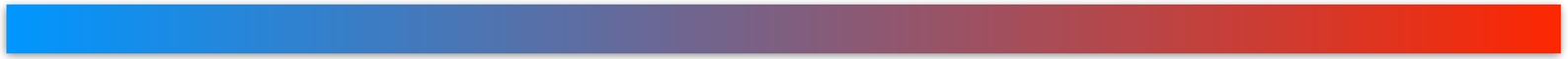
Sub-chunks with $\geq 1/2 \log n$ bits are sparse; we simply store relative offsets for every 1-bit

Overall: $$O\left(\underbrace{\frac{n}{1/2 \log n}}_{\text{Max \# sparse sub-chunks}} \underbrace{\log \log n}_{\text{\# bits to store 1 answer (rel. to chunk)}} \underbrace{\sqrt{\log n}}_{\text{\# 1-bits per chunk}}\right)$$

$$= O\left(\frac{n\sqrt{\log n} \log \log n}{\log n}\right) = O\left(\frac{n \log \log n}{\sqrt{\log n}}\right) = \check{o}(n)$$



Clark's select



Split into $\log^2 n$-weight chunks

Sparse ($\geq \log^4 n$-length)

Lookup table for each 1-bit ✅

Dense ($< \log^4 n$-length)

Split into $\sqrt{\log n}$-weight sub-chunks

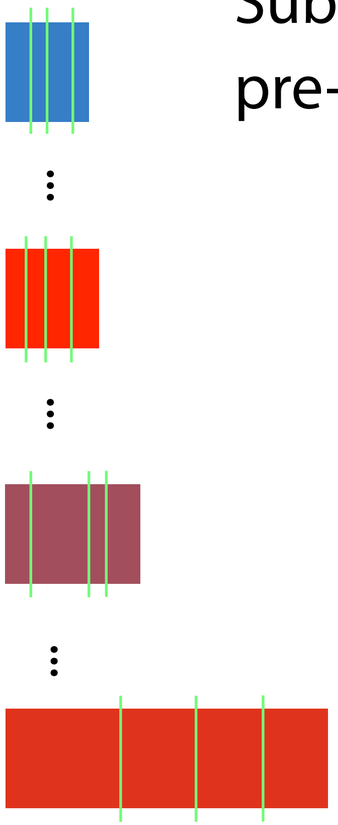
Sparse ($\geq 1/2 \log n$-length)

Lookup table for each 1-bit ✅

Dense ($< 1/2 \log n$-length)

Lookup table for all possible sub-chunks ✅

Clark's select: dense/dense case



Sub-chunks $< 1/2 \log n$ bits are dense;
pre-calculate answers for all such chunks, like rank:

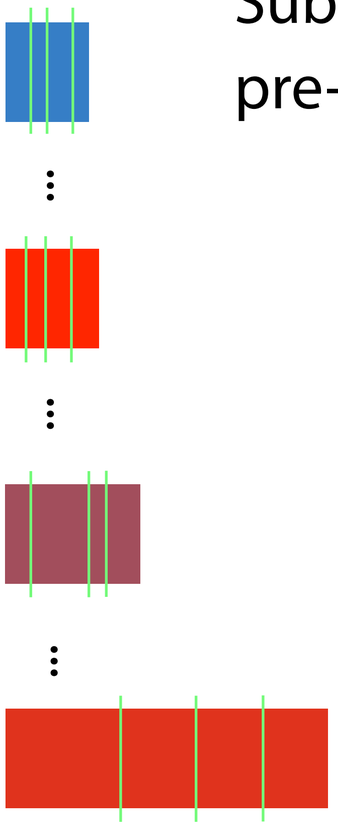
$$2^{1/2 \log n} \cdot \sqrt{\log n} \cdot \log \log n$$

possible bitvectors · possible 1-bits · answer

$$= O\left(\sqrt{n \log n} \log \log n\right)$$

$$= \check{o}(n)$$

Clark's select: dense/dense case



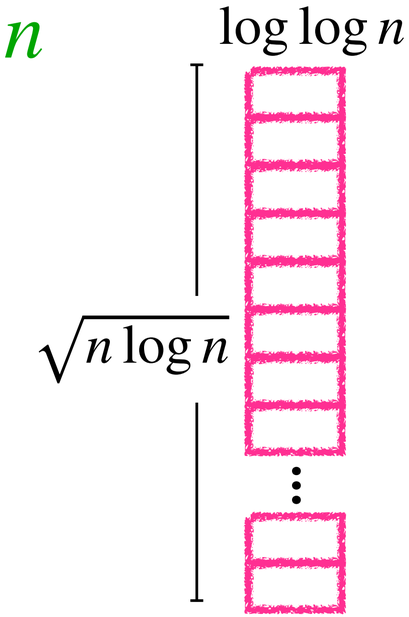
Sub-chunks $< 1/2 \log n$ bits are dense;
pre-calculate answers for all such chunks, like rank:

$$2^{1/2 \log n} \cdot \sqrt{\log n} \cdot \log \log n$$

possible bitvectors · possible 1-bits · answer

$$= O\left(\sqrt{n \log n} \log \log n\right)$$

$$= \check{o}(n)$$



Clark's select

(a) find what chunk it's in (division by $\log^2 n$)

(b) if chunk is sparse ($\geq \log^4 n$ bits)

(c) if chunk is dense ($< \log^4 n$ bits)

Clark's select

(a) find what chunk it's in (division by $\log^2 n$)

(b) if chunk is sparse ($\geq \log^4 n$ bits)

(b.i) look up answer in sparse offset table

(c) if chunk is dense ($< \log^4 n$ bits)

Clark's select

(a) find what chunk it's in (division by $\log^2 n$)

(b) if chunk is sparse ($\geq \log^4 n$ bits)

(b.i) look up answer in sparse offset table

(c) if chunk is dense ($< \log^4 n$ bits)

(c.i) look up chunk's offset

(c.ii) find what sub-chunk it's in (divide by $\sqrt{\log n}$)

(c.iii) look up sub-chunk's relative offset

(c.iv) if sub-chunk is sparse ($\geq 1/2 \log n$ bits)

(c.v) if sub-chunk is dense

Clark's select

(a) find what chunk it's in (division by $\log^2 n$)

(b) if chunk is sparse ($\geq \log^4 n$ bits)

(b.i) look up answer in sparse offset table

(c) if chunk is dense ($< \log^4 n$ bits)

(c.i) look up chunk's offset

(c.ii) find what sub-chunk it's in (divide by $\sqrt{\log n}$)

(c.iii) look up sub-chunk's relative offset

(c.iv) if sub-chunk is sparse ($\geq 1/2 \log n$ bits)

(c.iv.A) look up answer in sparse 1-bit table

(c.iv.B) return (c.i) + (c.iii) + (c.iv.A)

(c.v) if sub-chunk is dense

Clark's select

(a) find what chunk it's in (division by $\log^2 n$)

(b) if chunk is sparse ($\geq \log^4 n$ bits)

(b.i) look up answer in sparse offset table

(c) if chunk is dense ($< \log^4 n$ bits)

(c.i) look up chunk's offset

(c.ii) find what sub-chunk it's in (divide by $\sqrt{\log n}$)

(c.iii) look up sub-chunk's relative offset

(c.iv) if sub-chunk is sparse ($\geq 1/2 \log n$ bits)

(c.iv.A) look up answer in sparse 1-bit table

(c.iv.B) return (c.i) + (c.iii) + (c.iv.A)

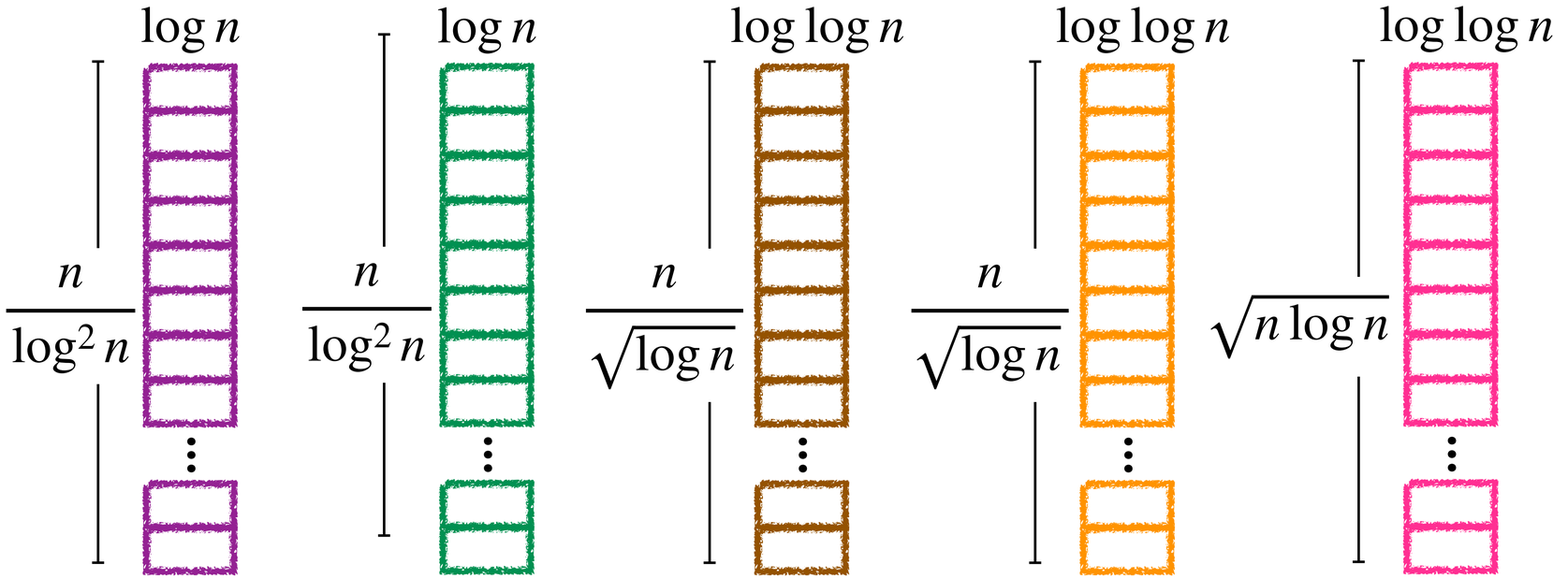
(c.v) if sub-chunk is dense

(c.v.A) look up answer in all possible dense/dense table

(c.v.B) return (c.i) + (c.iii) + (c.v.A)

Clark's select

Overall, space is $\check{o}(n)$



Sparse chunk offsets

Offsets for 1-bits in sparse chunks

Dense sub-chunk relative offsets

Answers for 1-bits in dense/ sparse sub-chunks

Answers for all possible dense/dense sub-chunks

Bitvectors

	Time	Space (bits)	Note
B . access	$O(1)$	n	Lookup
B . select$_1$	$O(1)$	$\check{o}(n)$	✅ Clark
B . rank$_1$	$O(1)$	$\check{o}(n)$	✅ Jacobson