Jacobson's Rank

Ben Langmead

JOHNS HOPKINS
WHITING SCHOOL of ENGINEERING

Department of Computer Science

Jacobson's rank

(Note: the following assumes we are doing $B.\text{rank}_1$ queries, but $B.\text{rank}_0$ 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.

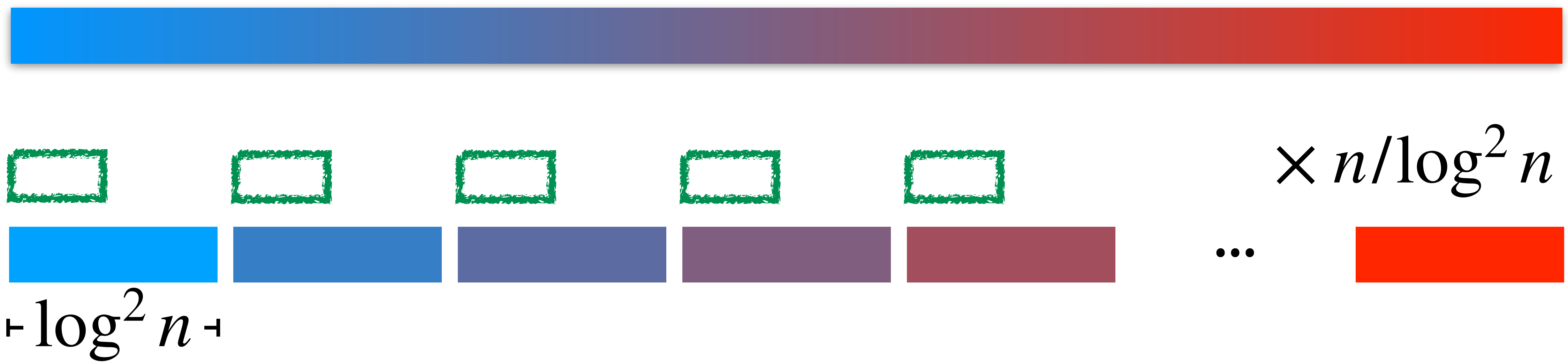
Jacobson's rank

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

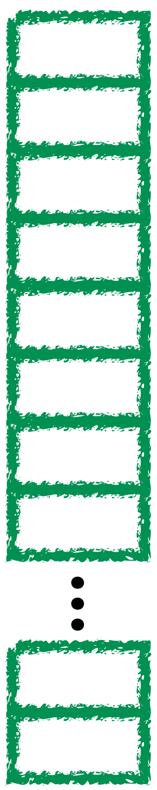
Jacobson's rank



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 $\check{o}(n)$

Finding a rank can be decomposed:

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

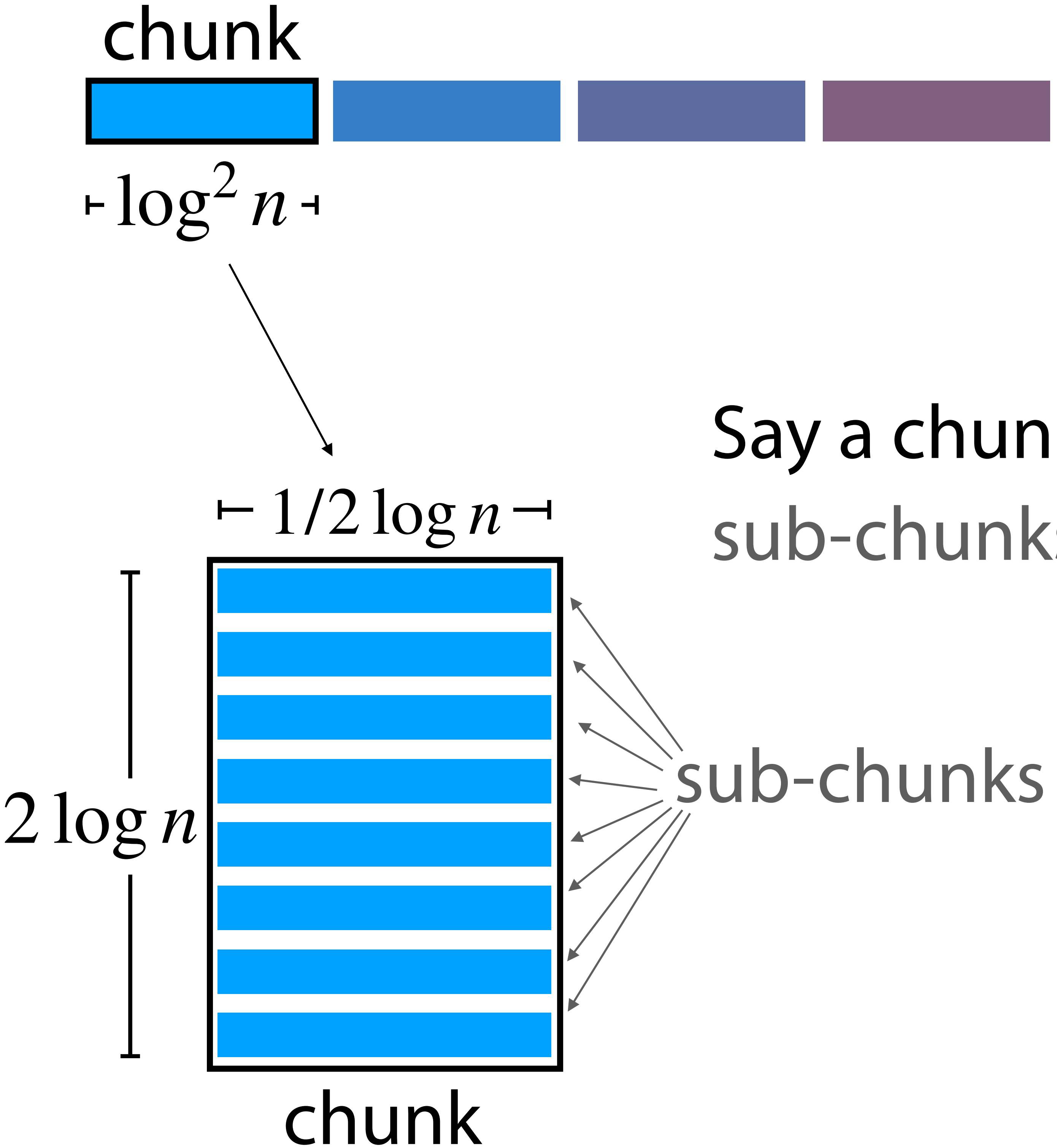
(b) look up cumulative rank

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

(d) add (b) + (c)

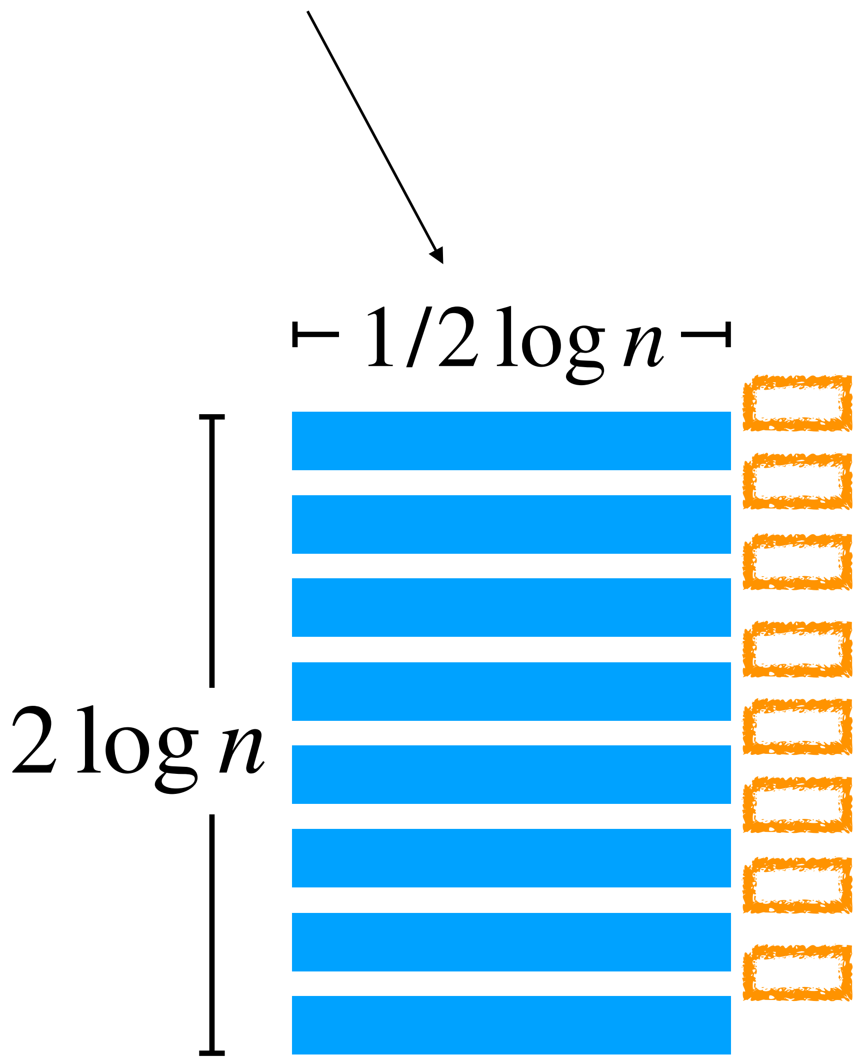
TODO

Jacobson's rank



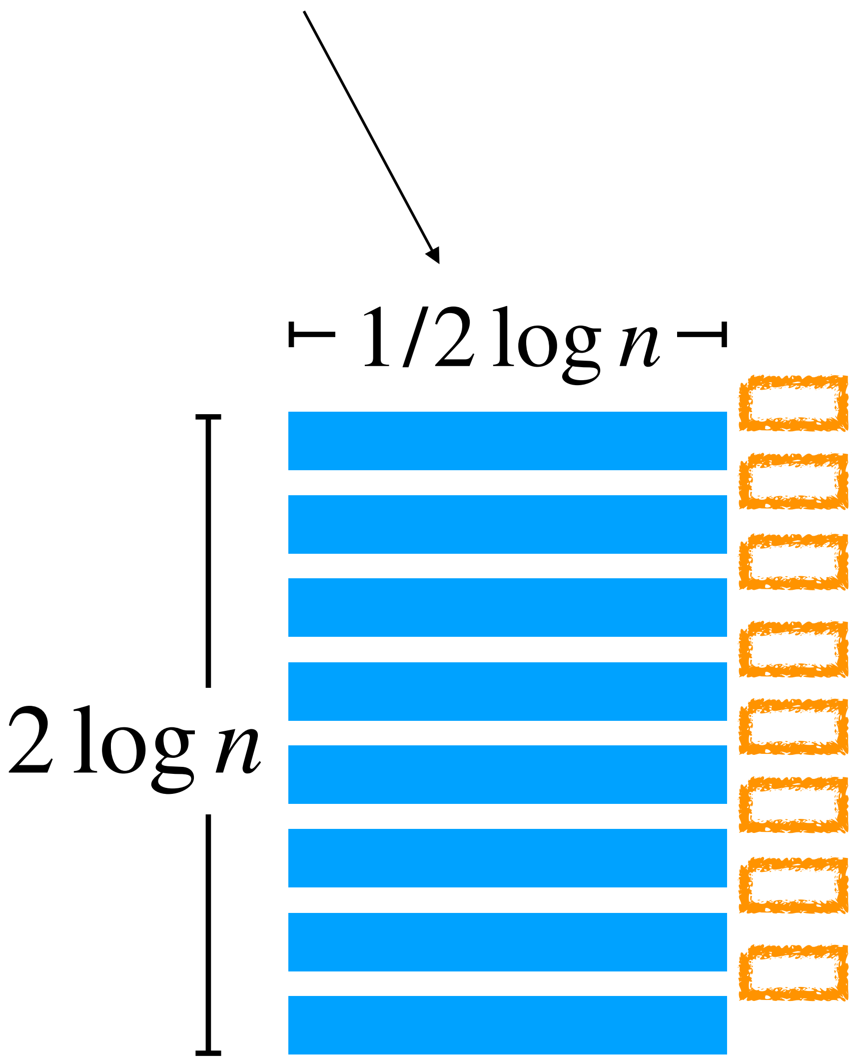
Say a chunk consists of $2 \log n$ sub-chunks, each of $1/2 \log n$ bits

Jacobson's rank



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

Jacobson's rank

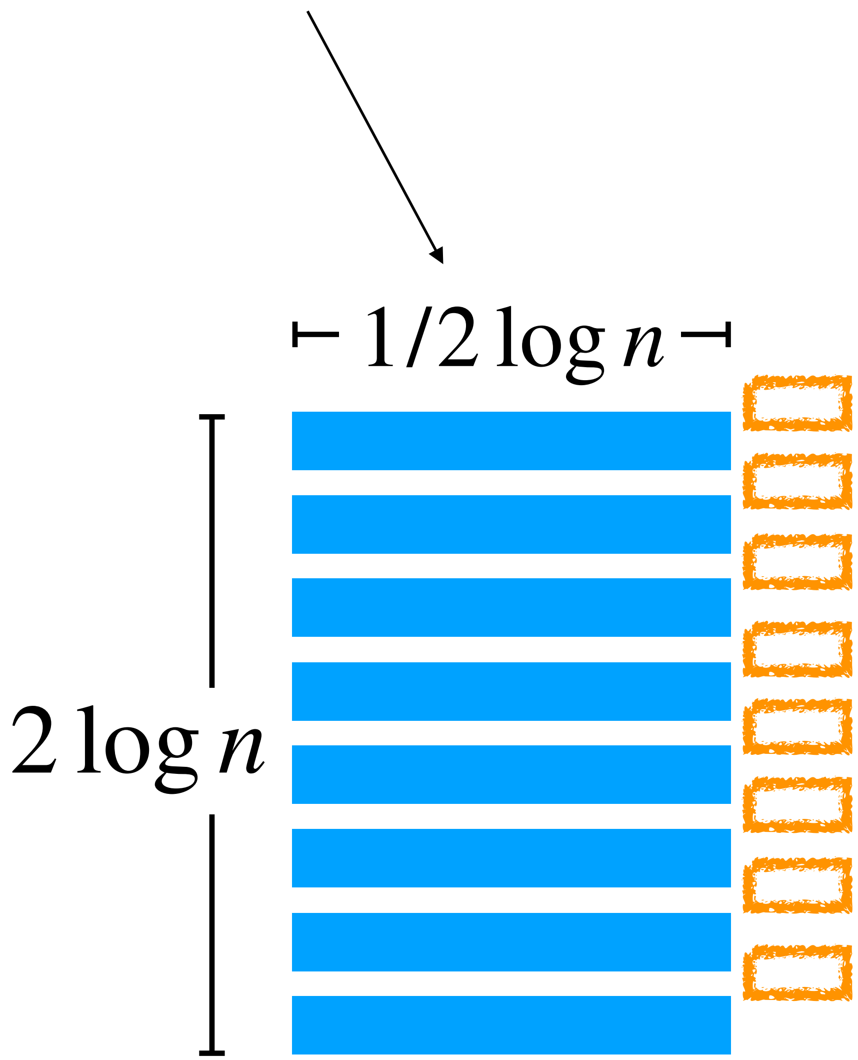


Since chunk has $\log^2 n$ bits, a relative cum. rank needs $\log \log^2 n = O(\log \log n)$ bits

$O\left(n/\log n\right)$ sub-chunks overall (across all chunks),
for total of $O\left(n \cdot \log \log n/\log n\right)$ bits for relative cum. ranks

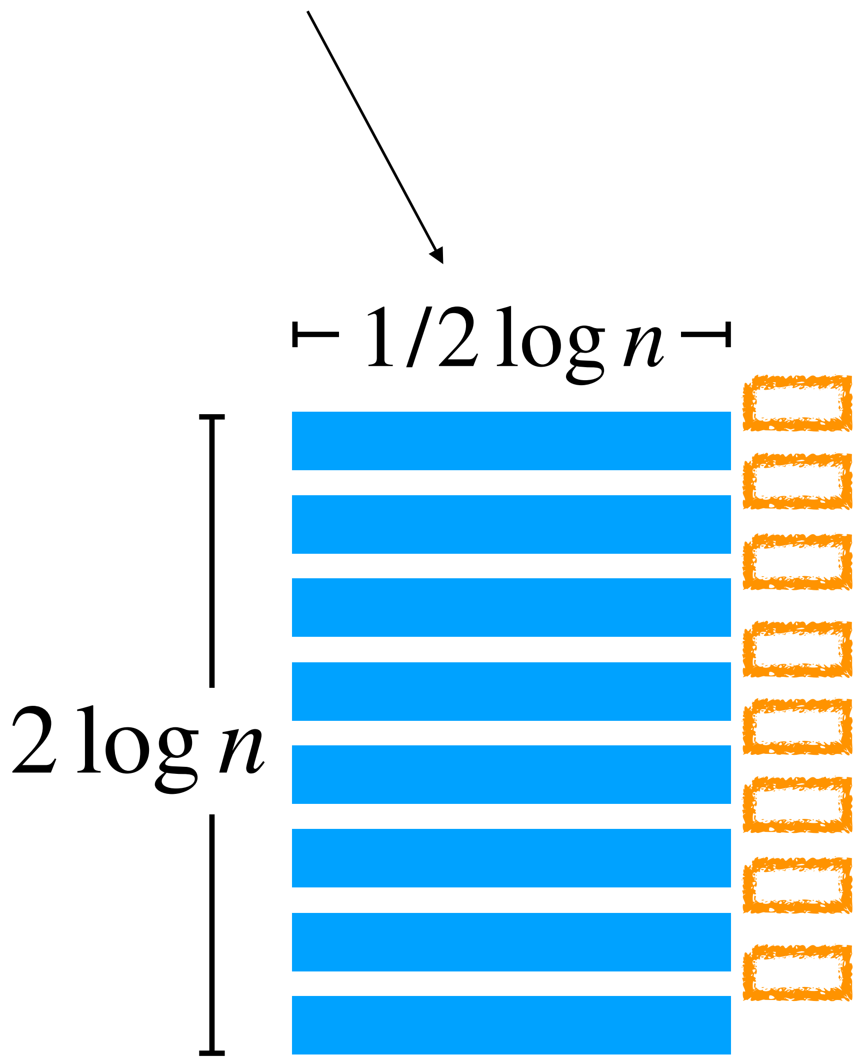
$$O\left(n \cdot \log \log n/\log n\right) = \check{o}(n)$$

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

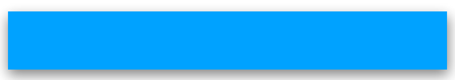
> **(c.i)** find what sub-chunk it's in
>
> **(c.ii)** look up relative cum. rank
>
> **(c.iii)** find rank *within* sub-chunk

TODO

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

Jacobson's rank

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



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?

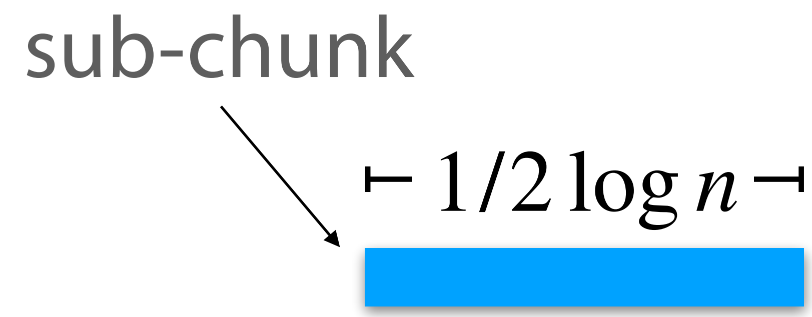
$$\underbrace{2^x}_{\text{possible bitvectors}} \cdot \underbrace{x}_{\text{possible offsets}} \cdot \underbrace{\log x}_{\text{answer}}$$

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



Let $x = 1/2 \log n$

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

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

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