Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Lehrstuhl für
INFORMATIK I
Effiziente Algorithmen und
wissensbasierte Systeme

Institut für Informatik

Advanced Algorithms

Winter term 2019/20

Lecture 9. Succinct data structures

(Based on lectures from Simon Gog and from Erik Demaine)

Jonathan Klawitter

Chair for Computer Science I

Succinct data structures

Goal

- use space “close” to information-theoretical minimum
- but still support time-efficient operations

Succinct data structures

Goal

- use space “close” to information-theoretical minimum
- but still support time-efficient operations

Let L be the information-theoretical lower bound to represent a class of objects. Then a data structure which still supports time-efficient operations is called

- implicit, if it takes $L + O(1)$ bits of space;

Succinct data structures

Goal

- use space "close" to information-theoretical minimum
- but still support time-efficient operations

Let L be the information-theoretical lower bound to represent a class of objects. Then a data structure which still supports time-efficient operations is called

- implicit, if it takes $L + O(1)$ bits of space;
- succinct, if it takes $L + o(L)$ bits of space;

Succinct data structures

Goal

- use space "close" to information-theoretical minimum
- but still support time-efficient operations

Let L be the information-theoretical lower bound to represent a class of objects. Then a data structure which still supports time-efficient operations is called

- implicit, if it takes $L + O(1)$ bits of space;
- succinct, if it takes $L + o(L)$ bits of space;
- compact, if it takes $O(L)$ bits of space.

Examples for implicit data structures

Examples for implicit data structures

- array to represent list; but why not linked list?

Examples for implicit data structures

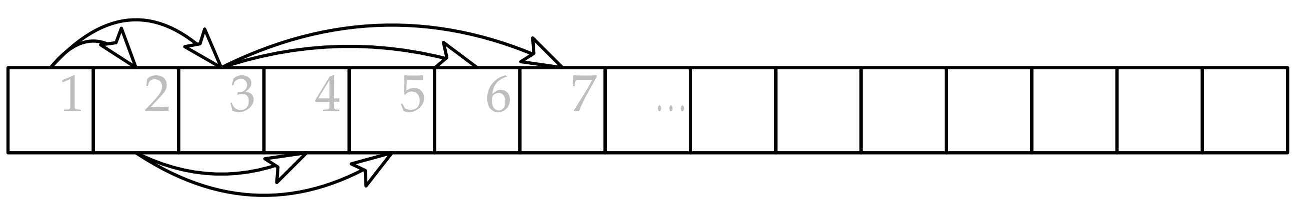
- array to represent list; but why not linked list?
- 1-dim array to represent multi-dimensional array

Examples for implicit data structures

- array to represent list; but why not linked list?
- 1-dim array to represent multi-dimensional array
- sorted array to represent sorted list;
  but why not binary search tree?

Examples for implicit data structures

- array to represent list; but why not linked list?
- 1-dim array to represent multi-dimensional array
- sorted array to represent sorted list;
  but why not binary search tree?
- array to represent complete binary tree or heap



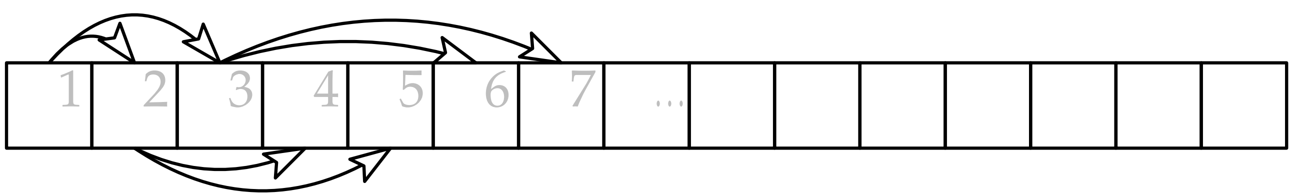
$\texttt{leftChild}(i) =$

$\texttt{rightChild}(i) =$

$\texttt{parent}(i) =$

Examples for implicit data structures

- array to represent list; but why not linked list?
- 1-dim array to represent multi-dimensional array
- sorted array to represent sorted list;
  but why not binary search tree?
- array to represent complete binary tree or heap



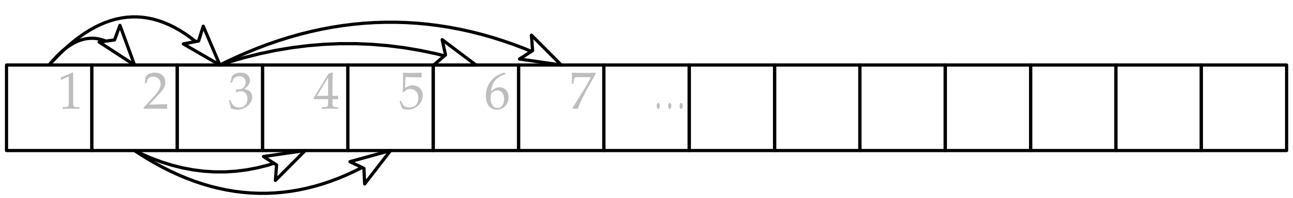
$\texttt{leftChild}(i) = 2i$

$\texttt{rightChild}(i) = 2i + 1$

$\texttt{parent}(i) = \lfloor \frac{i}{2} \rfloor$

Examples for implicit data structures

- array to represent list; but why not linked list?
- 1-dim array to represent multi-dimensional array
- sorted array to represent sorted list;
  but why not binary search tree?
- array to represent complete binary tree or heap



And un-balanced trees?

$\texttt{leftChild}(i) = 2i$

$\texttt{rightChild}(i) = 2i + 1$

$\texttt{parent}(i) = \lfloor \frac{i}{2} \rfloor$

Succinct indexable dictionary

Represent a subset $S \subset [n]$ and support $O(1)$ operations:

- `member`(i) returns if $i \in S$
- `rank`(i) = # 1's at or before position i
- `select`(j) = position of jth 1 bit
- `predecessor` and `successor` can be answered using `rank` and `select`

Succinct indexable dictionary

Represent a subset $S \subset [n]$ and support $O(1)$ operations:

- $\texttt{member}(i)$ returns if $i \in S$
- $\texttt{rank}(i) = $ # 1's at or before position i
- $\texttt{select}(j) = $ position of jth 1 bit
- `predecessor` and `successor` can be answered using `rank` and `select`

How many different subsets of $[n]$ are there?

How many bits of space do we need to distinguish them?

Succinct indexable dictionary

Represent a subset $S \subset [n]$ and support $O(1)$ operations:

- `member`(i) returns if $i \in S$
- `rank`(i) $=$ # 1's at or before position i
- `select`(j) $=$ position of jth 1 bit
- `predecessor` and `successor` can be answered using `rank` and `select`

How many different subsets of $[n]$ are there? 2^n

How many bits of space do we need to distinguish them?

Succinct indexable dictionary

Represent a subset $S \subset [n]$ and support $O(1)$ operations:

- `member`(i) returns if $i \in S$
- `rank`(i) $=$ # 1's at or before position i
- `select`(j) $=$ position of jth 1 bit
- `predecessor` and `successor` can be answered using `rank` and `select`

How many different subsets of $[n]$ are there? 2^n

How many bits of space do we need to distinguish them?

$$\log 2^n = n \text{ bits}$$

Succinct indexable dictionary

Represent S with a bit vector b of length n where

$$b[i] = \begin{cases} 1 & \text{if } i \in S \\ 0 & \text{otherwise} \end{cases}$$

Succinct indexable dictionary

Represent S with a bit vector b of length n where

$$b[i] = \begin{cases} 1 & \text{if } i \in S \\ 0 & \text{otherwise} \end{cases}$$

$$S = \{3, 4, 6, 8, 9, 14\} \text{ where } n = 15$$

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

Succinct indexable dictionary

Represent S with a bit vector b of length n where

$$b[i] = \begin{cases} 1 & \text{if } i \in S \\ 0 & \text{otherwise} \end{cases}$$

plus $o(n)$ space structures to answer in $O(1)$ time

- $\texttt{rank}(i) = \#\ 1$'s at or before position i
- $\texttt{select}(j) =$ position of jth 1 bit

$S = \{3, 4, 6, 8, 9, 14\}$ where $n = 15$

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

Succinct indexable dictionary

Represent S with a bit vector b of length n where

$$b[i] = \begin{cases} 1 & \text{if } i \in S \\ 0 & \text{otherwise} \end{cases}$$

plus $o(n)$ space structures to answer in $O(1)$ time

- $\texttt{rank}(i)$ = # 1's at or before position i
- $\texttt{select}(j)$ = position of jth 1 bit

$S = \{3, 4, 6, 8, 9, 14\}$ where $n = 15$

$\texttt{select}(5) =$

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

Succinct indexable dictionary

Represent S with a bit vector b of length n where

$$b[i] = \begin{cases} 1 & \text{if } i \in S \\ 0 & \text{otherwise} \end{cases}$$

plus $o(n)$ space structures to answer in $O(1)$ time

- $\texttt{rank}(i)$ = # 1's at or before position i
- $\texttt{select}(j)$ = position of jth 1 bit

$S = \{3, 4, 6, 8, 9, 14\}$ where $n = 15$

$\texttt{select}(5) = 9$

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

Succinct indexable dictionary

Represent S with a bit vector b of length n where

$$b[i] = \begin{cases} 1 & \text{if } i \in S \\ 0 & \text{otherwise} \end{cases}$$

plus $o(n)$ space structures to answer in $O(1)$ time

- $\texttt{rank}(i)$ = # 1's at or before position i
- $\texttt{select}(j)$ = position of jth 1 bit

$S = \{3, 4, 6, 8, 9, 14\}$ where $n = 15$

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

$\texttt{select}(5) = 9$

$\texttt{rank}(9) =$

Succinct indexable dictionary

Represent S with a bit vector b of length n where

$$b[i] = \begin{cases} 1 & \text{if } i \in S \\ 0 & \text{otherwise} \end{cases}$$

plus $o(n)$ space structures to answer in $O(1)$ time

- $\texttt{rank}(i)$ = # 1's at or before position i
- $\texttt{select}(j)$ = position of jth 1 bit

$S = \{3, 4, 6, 8, 9, 14\}$ where $n = 15$

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

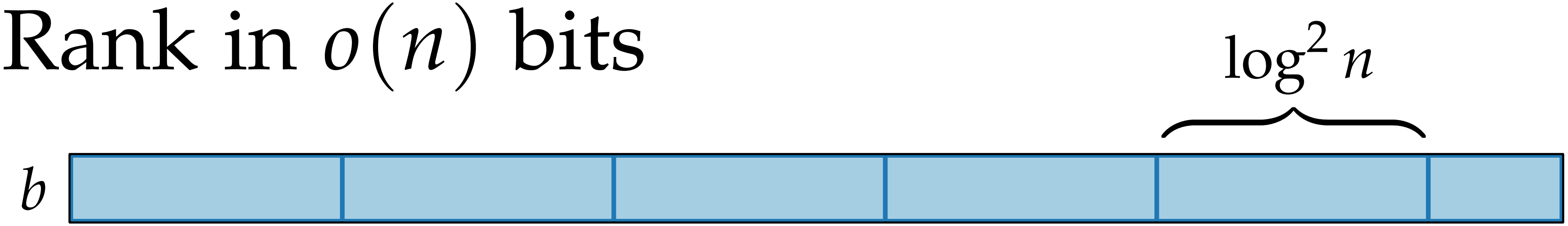
$\texttt{select}(5) = 9$

$\texttt{rank}(9) = 5$

Rank in $o(n)$ bits

b $\boxed{}$

Rank in $o(n)$ bits



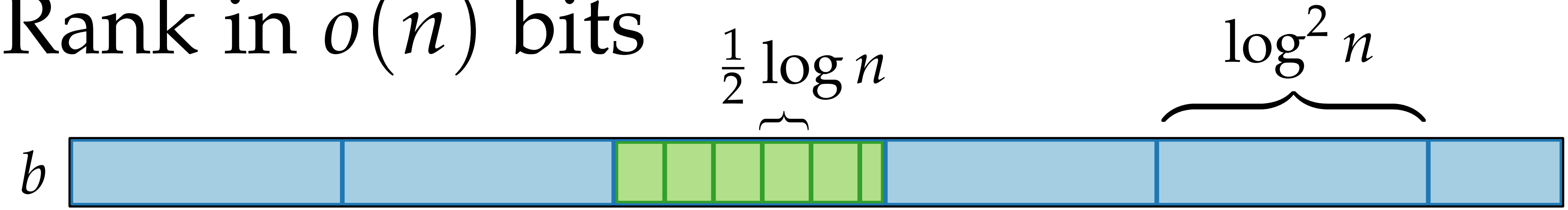
1. Split into $(\log^2 n)$-bit chunks
   and store cumulative rank: each $\log n$ bits

Rank in $o(n)$ bits

1. Split into $(\log^2 n)$-bit chunks
   and store cumulative rank: each $\log n$ bits

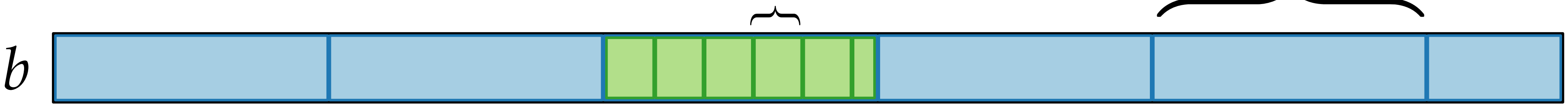
$$\Rightarrow O(\underbrace{\tfrac{n}{\log^2 n}}_{\#\text{ chunks}} \underbrace{\log n}_{\text{rank}}) = O(\tfrac{n}{\log n}) \subseteq o(n) \text{ bits}$$

Rank in $o(n)$ bits



1. Split into $(\log^2 n)$-bit chunks
   and store cumulative rank: each $\log n$ bits
   $$\Rightarrow O\left(\frac{n}{\log^2 n}\log n\right) = O\left(\frac{n}{\log n}\right) \subseteq o(n) \text{ bits}$$
2. Split chunks into $(\frac{1}{2}\log n)$-bit subchunks
   and store cummulative rank within chunk:

Rank in $o(n)$ bits



1. Split into $(\log^2 n)$-bit chunks

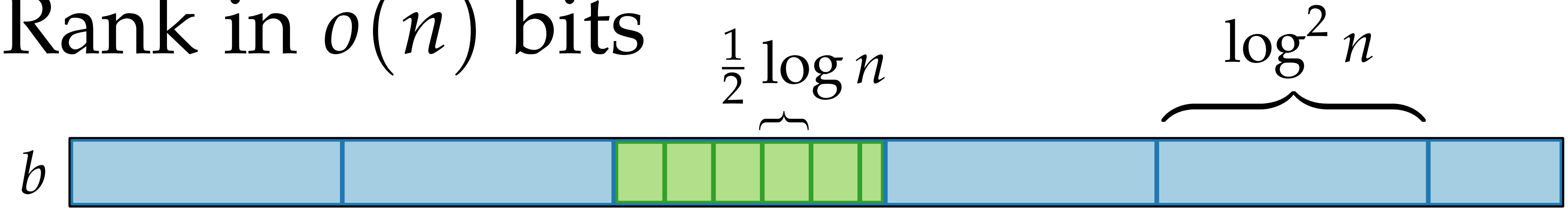
   and store cumulative rank: each $\log n$ bits

   $$\Rightarrow O\left(\frac{n}{\log^2 n}\log n\right) = O\left(\frac{n}{\log n}\right) \subseteq o(n) \text{ bits}$$

2. Split chunks into $(\frac{1}{2}\log n)$-bit subchunks

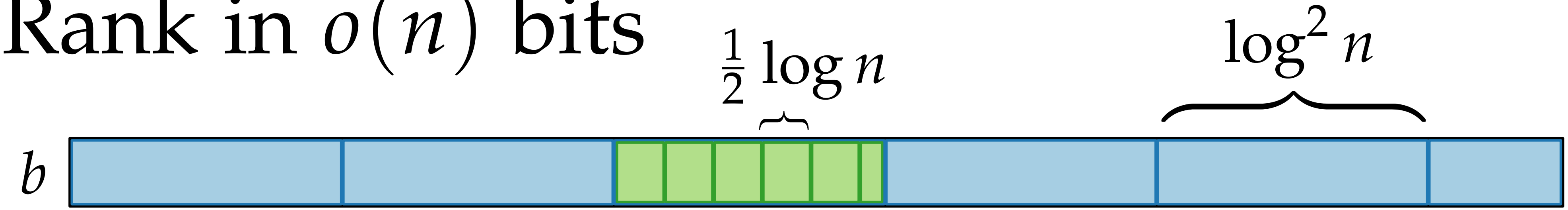
   and store cumulative rank within chunk: $2\log\log n$ bits

Rank in $o(n)$ bits



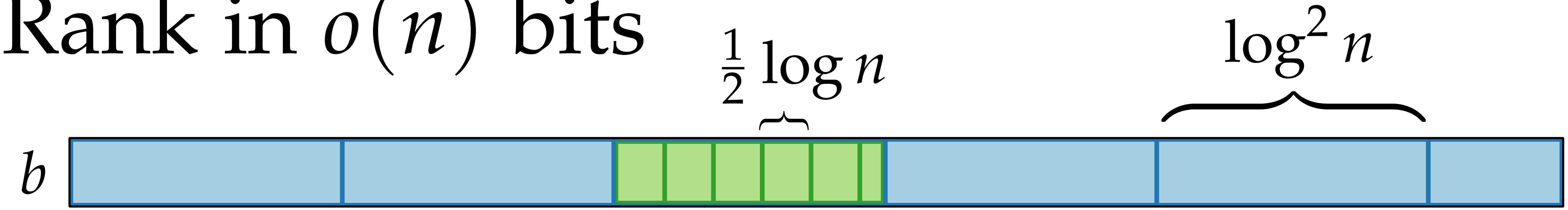
1. Split into $(\log^2 n)$-bit chunks
   and store cumulative rank: each $\log n$ bits
   $$\Rightarrow O\left(\frac{n}{\log^2 n}\log n\right) = O\left(\frac{n}{\log n}\right) \subseteq o(n) \text{ bits}$$
2. Split chunks into $(\frac{1}{2}\log n)$-bit subchunks
   and store cumulative rank within chunk: $2\log\log n$ bits
   $$\Rightarrow O\Big(\underbrace{\frac{n}{\log n}}_{\#\text{ subch.}}\ \underbrace{\log\log n}_{\text{rel. rank}}\Big) \subseteq o(n) \text{ bits}$$

Rank in $o(n)$ bits



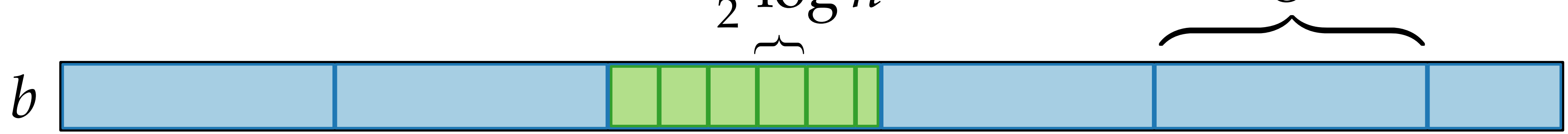
1. Split into $(\log^2 n)$-bit chunks
   and store cumulative rank: each $\log n$ bits
   $$\Rightarrow O\left(\frac{n}{\log^2 n}\log n\right) = O\left(\frac{n}{\log n}\right) \subseteq o(n) \text{ bits}$$
2. Split chunks into $(\frac{1}{2}\log n)$-bit subchunks
   and store cumulative rank within chunk: $2\log\log n$ bits
   $$\Rightarrow O\left(\frac{n}{\log n}\log\log n\right) \subseteq o(n) \text{ bits}$$
3. Use lookup table for bitstrings of length $(\frac{1}{2}\log n)$
   $$\Rightarrow O(\underbrace{\sqrt{n}}_{\text{bitstring}}\ \underbrace{\log n}_{\text{query } i}\ \underbrace{\log\log n}_{\text{answer}}) \subseteq o(n) \text{ bits}$$

Rank in $o(n)$ bits



1. Split into $(\log^2 n)$-bit chunks
   and store cumulative rank: each $\log n$ bits
   $$\Rightarrow O\left(\frac{n}{\log^2 n}\log n\right) = O\left(\frac{n}{\log n}\right) \subseteq o(n) \text{ bits}$$
2. Split chunks into $(\frac{1}{2}\log n)$-bit subchunks
   and store cumulative rank within chunk: $2\log\log n$ bits
   $$\Rightarrow O\left(\frac{n}{\log n}\log\log n\right) \subseteq o(n) \text{ bits}$$
3. Use lookup table for bitstrings of length $(\frac{1}{2}\log n)$
   $$\Rightarrow O(\sqrt{n}\log n \log\log n) \subseteq o(n) \text{ bits}$$
4. `rank` = `rank` of chunk
   + relative `rank` of subchunk within chunk
   + relative `rank` of element within subchunk

Rank in $o(n)$ bits

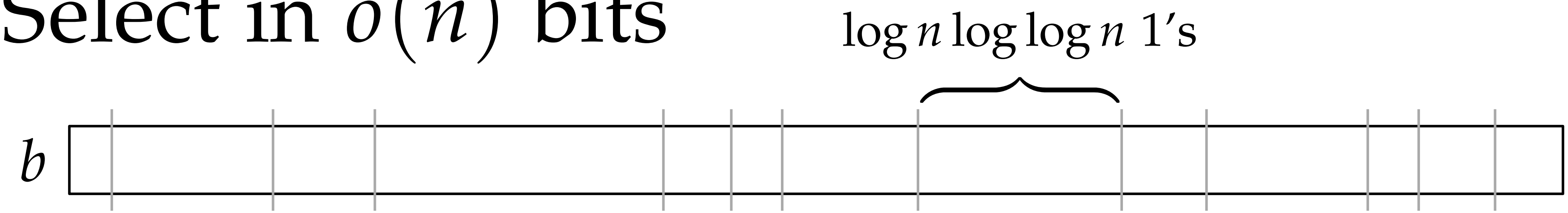


1. Split into $(\log^2 n)$-bit chunks
   and store cumulative rank: each $\log n$ bits
   $$\Rightarrow O\left(\frac{n}{\log^2 n}\log n\right) = O\left(\frac{n}{\log n}\right) \subseteq o(n) \text{ bits}$$
2. Split chunks into $(\frac{1}{2}\log n)$-bit subchunks
   and store cumulative rank within chunk: $2\log\log n$ bits
   $$\Rightarrow O\left(\frac{n}{\log n}\log\log n\right) \subseteq o(n) \text{ bits}$$
3. Use lookup table for bitstrings of length $(\frac{1}{2}\log n)$
   $$\Rightarrow O(\sqrt{n}\log n \log\log n) \subseteq o(n) \text{ bits}$$
4. `rank` = `rank` of chunk $\Rightarrow O(1)$ time
   + relative `rank` of subchunk within chunk
   + relative `rank` of element within subchunk

Select in $o(n)$ bits

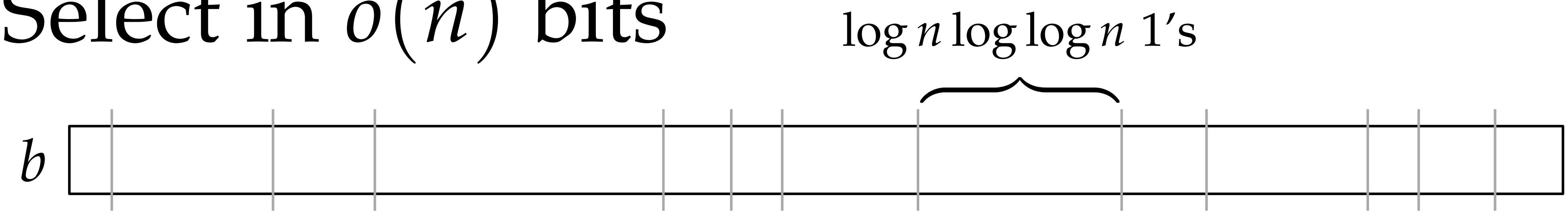
b

Select in $o(n)$ bits



1. Store indices of every $(\log n \log\log n)$th 1 bit in array

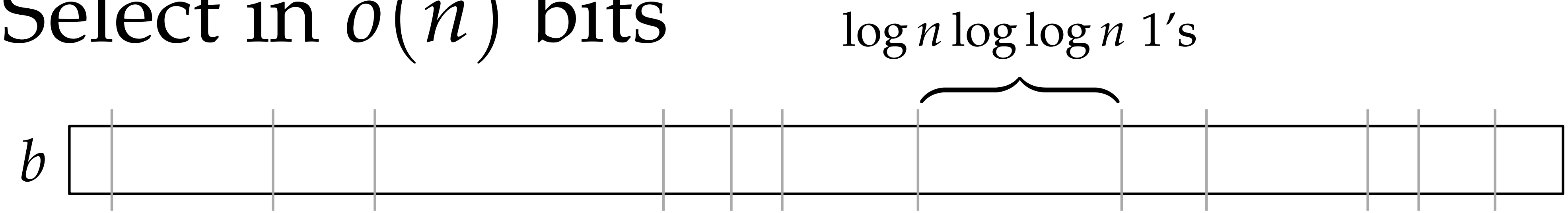
Select in $o(n)$ bits



1. Store indices of every $(\log n \log\log n)$th 1 bit in array

$$\Rightarrow O\Big(\underbrace{\frac{n}{\log n \log\log n}}_{\#\text{ groups}} \underbrace{\log n}_{\text{index}}\Big) = O\Big(\frac{n}{\log\log n}\Big) = o(n) \text{ bits}$$

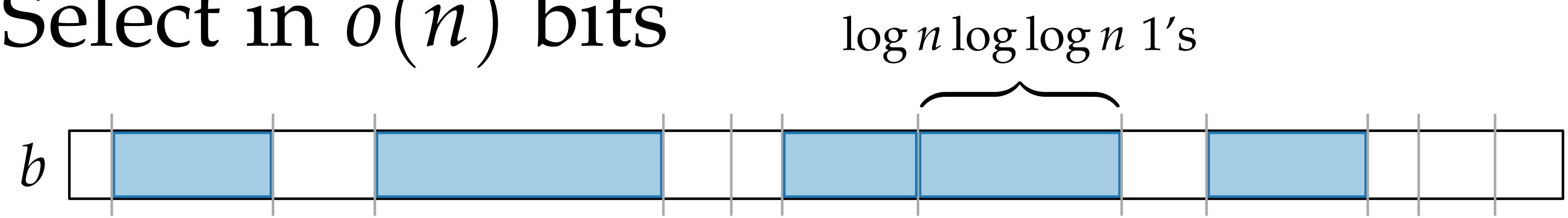
Select in $o(n)$ bits



1. Store indices of every $(\log n \log\log n)$th 1 bit in array
$$\Rightarrow O\left(\frac{n}{\log n \log\log n} \log n\right) = O\left(\frac{n}{\log\log n}\right) = o(n) \text{ bits}$$

2. Within group of $(\log n \log\log n)$ 1 bits, say r bits:

Select in $o(n)$ bits



1. Store indices of every $(\log n \log\log n)$th 1 bit in array

$$\Rightarrow O\left(\frac{n}{\log n \log\log n}\log n\right) = O\left(\frac{n}{\log\log n}\right) = o(n) \text{ bits}$$

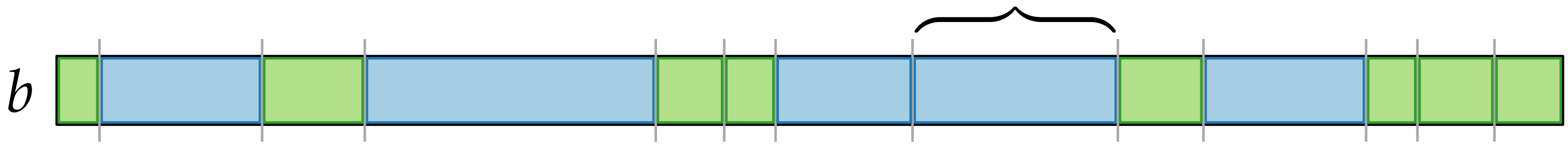
2. Within group of $(\log n \log\log n)$ 1 bits, say r bits:

if $r \geq (\log n \log\log n)^2$

then store indices of 1 bits in group in array

$$\Rightarrow O\left(\underbrace{\frac{n}{(\log n \log\log n)^2}}_{\#\text{ groups}}\underbrace{(\log n \log\log n)}_{\#\text{ 1 bits}}\underbrace{\log n}_{\text{index}}\right) = O\left(\frac{n}{\log\log n}\right)$$

Select in $o(n)$ bits

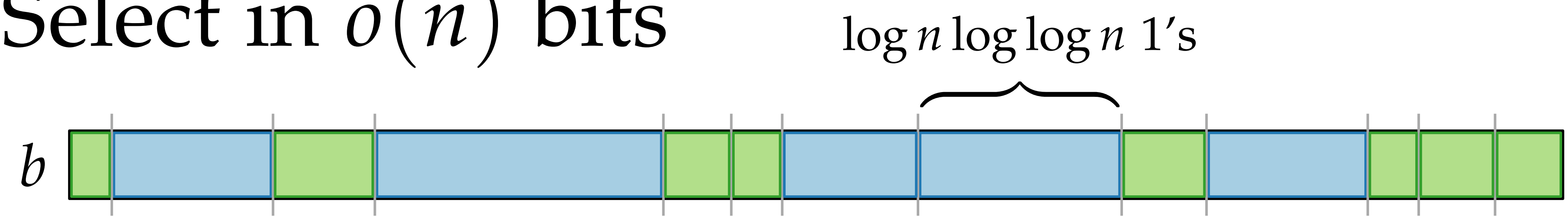


1. Store indices of every $(\log n \log\log n)$th 1 bit in array
$$\Rightarrow O\left(\frac{n}{\log n \log\log n}\log n\right) = O\left(\frac{n}{\log\log n}\right) = o(n) \text{ bits}$$

2. Within group of $(\log n \log\log n)$ 1 bits, say r bits:

   if $r \geq (\log n \log\log n)^2$
   then store indices of 1 bits in group in array
   $$\Rightarrow O\left(\frac{n}{(\log n \log\log n)^2}(\log n \log\log n)\log n\right) = O\left(\frac{n}{\log\log n}\right)$$
   else reduced to bitstrings of length $r < (\log n \log\log n)^2$

Select in $o(n)$ bits



1. Store indices of every $(\log n \log\log n)$th 1 bit in array
$$\Rightarrow O\left(\frac{n}{\log n \log\log n} \log n\right) = O\left(\frac{n}{\log\log n}\right) = o(n) \text{ bits}$$

2. Within group of $(\log n \log\log n)$ 1 bits, say r bits:

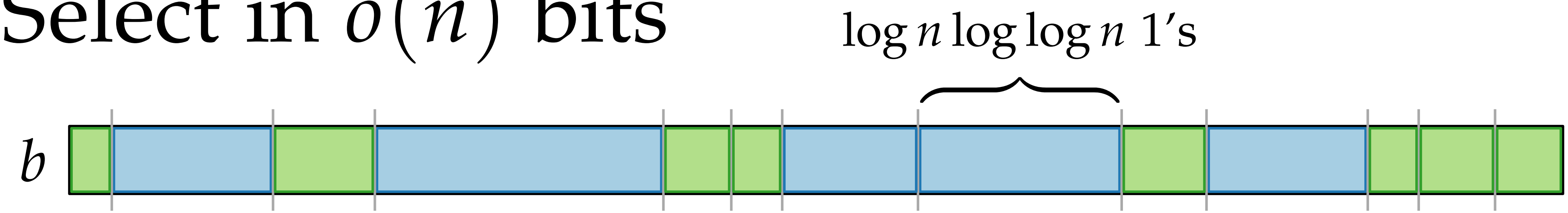
   if $r \geq (\log n \log\log n)^2$

   then store indices of 1 bits in group in array
   $$\Rightarrow O\left(\frac{n}{(\log n \log\log n)^2} (\log n \log\log n) \log n\right) = O\left(\frac{n}{\log\log n}\right)$$

   else reduced to bitstrings of length $r < (\log n \log\log n)^2$

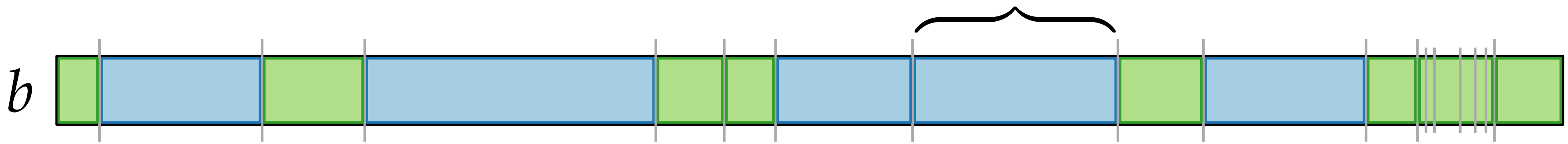
3. Repeat 1. and 2. on reduced bitstrings

Select in $o(n)$ bits



3. Repeat 1. and 2. on reduced bitstrings ($r < (\log n \log\log n)^2$):

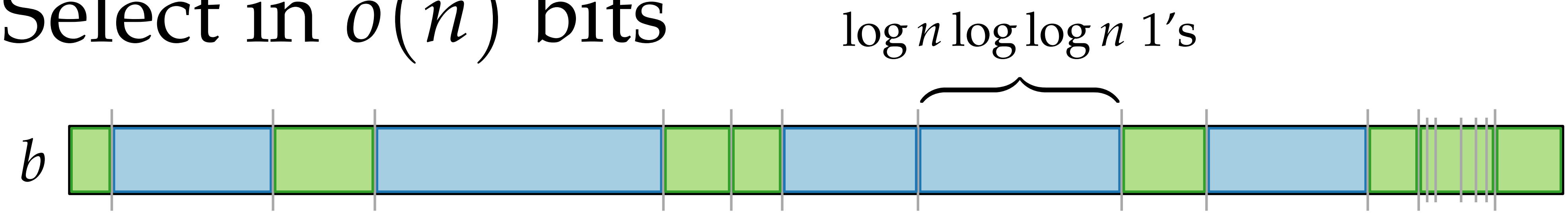
Select in $o(n)$ bits



3. Repeat 1. and 2. on reduced bitstrings ($r < (\log n \log\log n)^2$):

   1′ Store relative indices of every $(\log\log n)^2$th 1 bit in array

Select in $o(n)$ bits

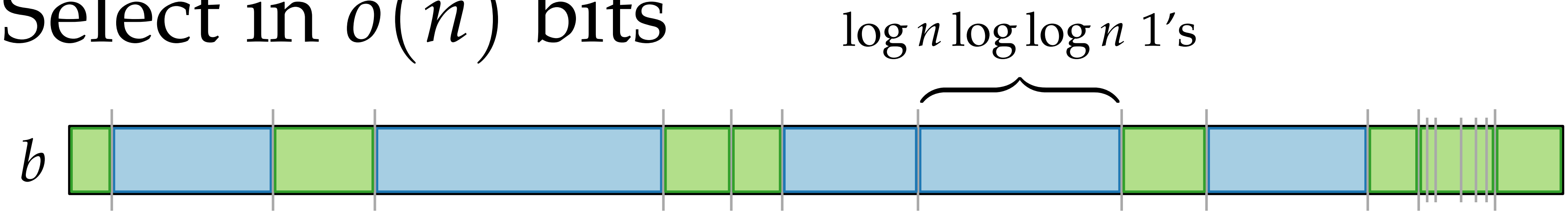


3. Repeat 1. and 2. on reduced bitstrings ($r < (\log n \log\log n)^2$):

   1′ Store relative indices of every $(\log\log n)^2$th 1 bit in array

   $$\Rightarrow O\left(\frac{n}{(\log\log n)^2}\log\log n\right) = O\left(\frac{n}{\log\log n}\right) \text{ bits}$$

Select in $o(n)$ bits



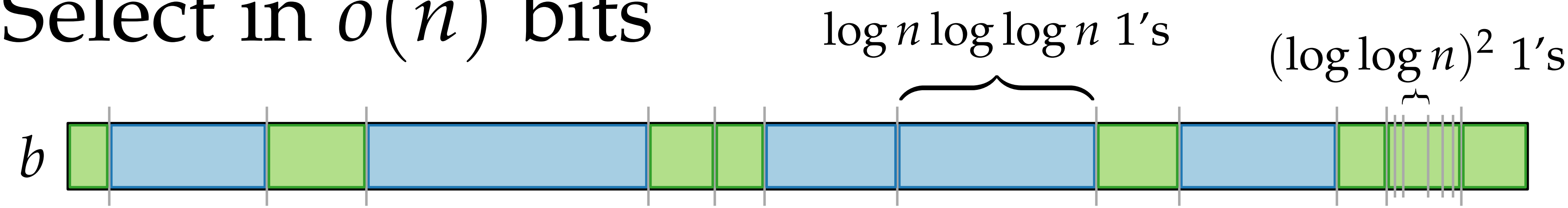
3. Repeat 1. and 2. on reduced bitstrings ($r < (\log n \log\log n)^2$):

   1′ Store relative indices of every $(\log\log n)^2$th 1 bit in array

$$\Rightarrow O\left(\frac{n}{(\log\log n)^2} \log\log n\right) = O\left(\frac{n}{\log\log n}\right) \text{ bits}$$

\# subgroups rel. index

Select in $o(n)$ bits



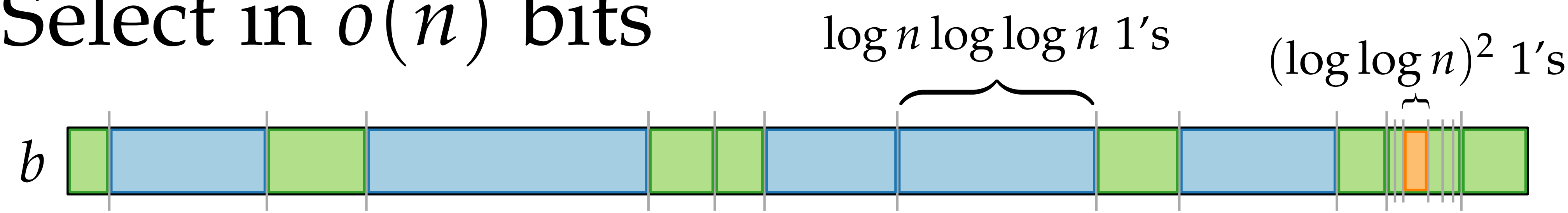
3. Repeat 1. and 2. on reduced bitstrings ($r < (\log n \log\log n)^2$):

1′ Store relative indices of every $(\log\log n)^2$th 1 bit in array

$$\Rightarrow O(\frac{n}{(\log\log n)^2} \log\log n) = O(\frac{n}{\log\log n}) \text{ bits}$$

2′ Within group of $(\log\log n)^2$th 1 bits, say r' bits:

Select in $o(n)$ bits



3. Repeat 1. and 2. on reduced bitstrings ($r < (\log n \log\log n)^2$):

   1′ Store relative indices of every $(\log\log n)^2$th 1 bit in array

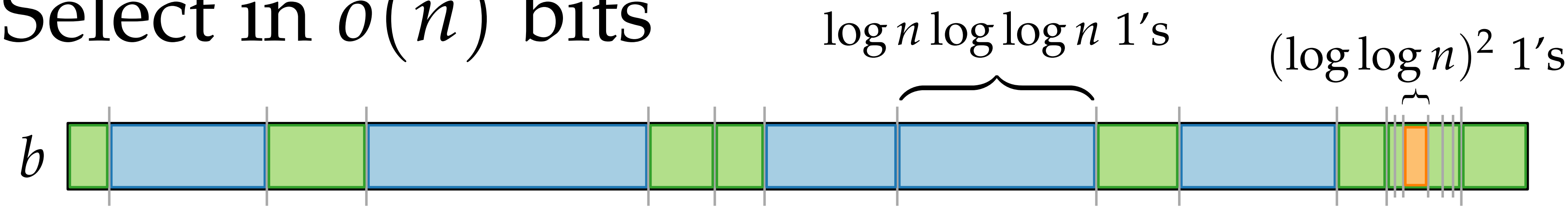
   $$\Rightarrow O\left(\frac{n}{(\log\log n)^2} \log\log n\right) = O\left(\frac{n}{\log\log n}\right) \text{ bits}$$

   2′ Within group of $(\log\log n)^2$th 1 bits, say r' bits:

   if $r' \geq (\log\log n)^4$

   then store relative indices of 1 bits in subgroup in array

Select in $o(n)$ bits



3. Repeat 1. and 2. on reduced bitstrings ($r < (\log n \log\log n)^2$):

   1′ Store relative indices of every $(\log\log n)^2$th 1 bit in array

   $$\Rightarrow O\left(\frac{n}{(\log\log n)^2} \log\log n\right) = O\left(\frac{n}{\log\log n}\right) \text{ bits}$$

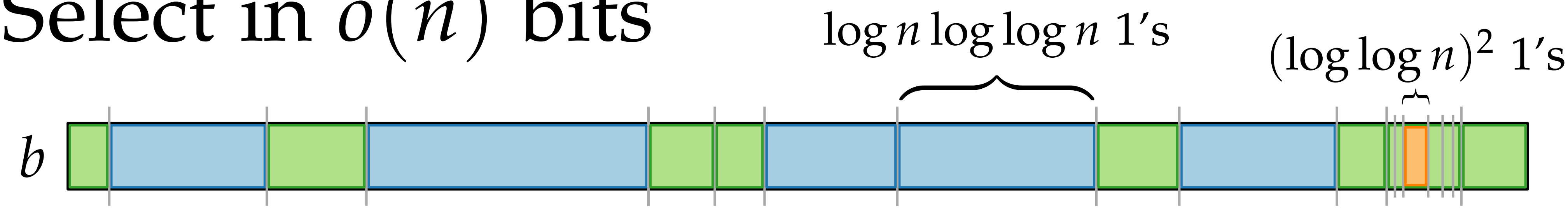
   2′ Within group of $(\log\log n)^2$th 1 bits, say r' bits:

   if $r' \geq (\log\log n)^4$

   then store relative indices of 1 bits in subgroup in array

   $$\Rightarrow O\left(\frac{n}{(\log\log n)^4} (\log\log n)^2 \log\log n\right) = O\left(\frac{n}{\log\log n}\right) \text{ bits}$$

Select in $o(n)$ bits



3. Repeat 1. and 2. on reduced bitstrings ($r < (\log n \log\log n)^2$):

1′ Store relative indices of every $(\log\log n)^2$th 1 bit in array

$$\Rightarrow O\left(\frac{n}{(\log\log n)^2}\log\log n\right) = O\left(\frac{n}{\log\log n}\right) \text{ bits}$$

2′ Within group of $(\log\log n)^2$th 1 bits, say r' bits:

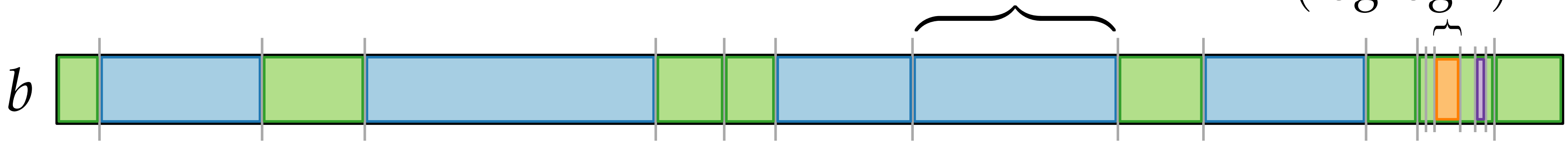
if $r' \geq (\log\log n)^4$

then store relative indices of 1 bits in subgroup in array

$$\Rightarrow O\left(\frac{n}{(\log\log n)^4}(\log\log n)^2 \log\log n\right) = O\left(\frac{n}{\log\log n}\right) \text{ bits}$$

\# subgroups   \# 1 bits   rel. index

Select in $o(n)$ bits



3. Repeat 1. and 2. on reduced bitstrings ($r < (\log n \log\log n)^2$):

1′ Store relative indices of every $(\log\log n)^2$th 1 bit in array

$$\Rightarrow O\left(\frac{n}{(\log\log n)^2}\log\log n\right) = O\left(\frac{n}{\log\log n}\right) \text{ bits}$$

2′ Within group of $(\log\log n)^2$th 1 bits, say r' bits:

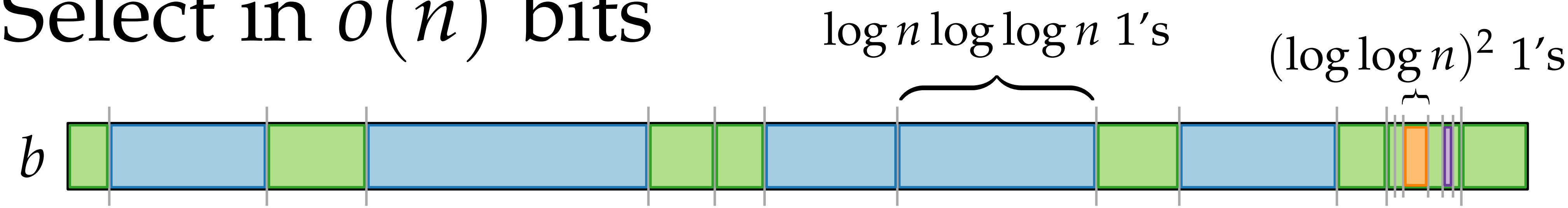
if $r' \geq (\log\log n)^4$

then store relative indices of 1 bits in subgroup in array

$$\Rightarrow O\left(\frac{n}{(\log\log n)^4}(\log\log n)^2\log\log n\right) = O\left(\frac{n}{\log\log n}\right) \text{ bits}$$

else reduced to bitstrings of length $r' < (\log\log n)^4$

Select in $o(n)$ bits



3. Repeat 1. and 2. on reduced bitstrings ($r < (\log n \log\log n)^2$):

   1′ Store relative indices of every $(\log\log n)^2$th 1 bit in array

   $$\Rightarrow O\left(\frac{n}{(\log\log n)^2}\log\log n\right) = O\left(\frac{n}{\log\log n}\right) \text{ bits}$$

   2′ Within group of $(\log\log n)^2$th 1 bits, say r' bits:

   if $r' \geq (\log\log n)^4$

   then store relative indices of 1 bits in subgroup in array

   $$\Rightarrow O\left(\frac{n}{(\log\log n)^4}(\log\log n)^2\log\log n\right) = O\left(\frac{n}{\log\log n}\right) \text{ bits}$$

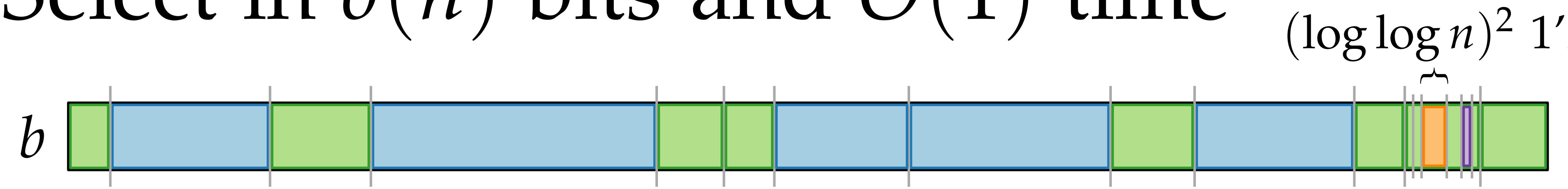
   else reduced to bitstrings of length $r' < (\log\log n)^4$

4. Use lookup table for bitstrings of length $r' \leq \frac{1}{2}\log n$

   $$\Rightarrow O(\sqrt{n}\log n\log\log n) = o(n) \text{ bits}$$

   bitstring query j answer

Select in $o(n)$ bits and $O(1)$ time



3. Repeat 1. and 2. on reduced bitstrings ($r < (\log n \log\log n)^2$):

   1′ Store relative indices of every $(\log\log n)^2$th 1 bit in array

   $\Rightarrow O(\frac{n}{(\log\log n)^2} \log\log n) = O(\frac{n}{\log\log n})$ bits

   2′ Within group of $(\log\log n)^2$th 1 bits, say r' bits:

   if $r' \geq (\log\log n)^4$

   then store relative indices of 1 bits in subgroup in array

   $\Rightarrow O(\frac{n}{(\log\log n)^4} (\log\log n)^2 \log\log n) = O(\frac{n}{\log\log n})$ bits

   else reduced to bitstrings of length $r' < (\log\log n)^4$

4. Use lookup table for bitstrings of length $r' \leq \frac{1}{2}\log n$

   $\Rightarrow O(\underbrace{\sqrt{n}}_{\text{bitstring}}\ \underbrace{\log n}_{\text{query } j}\ \underbrace{\log\log n}_{\text{answer}}) = o(n)$ bits

Succinct representation of binary trees

Number of binary trees on n vertices: $C_n = \frac{1}{n+1}\binom{2n}{n}$

$$\log C_n = 2n + o(n) \text{ (by Stirling's approximation)}$$

Operations we want to support:

`parent(v)`, `leftChild(v)`, `rightChild(v)`

Succinct representation of binary trees

Number of binary trees on n vertices: $C_n = \frac{1}{n+1}\binom{2n}{n}$

$$\log C_n = 2n + o(n) \text{ (by Stirling's approximation)}$$

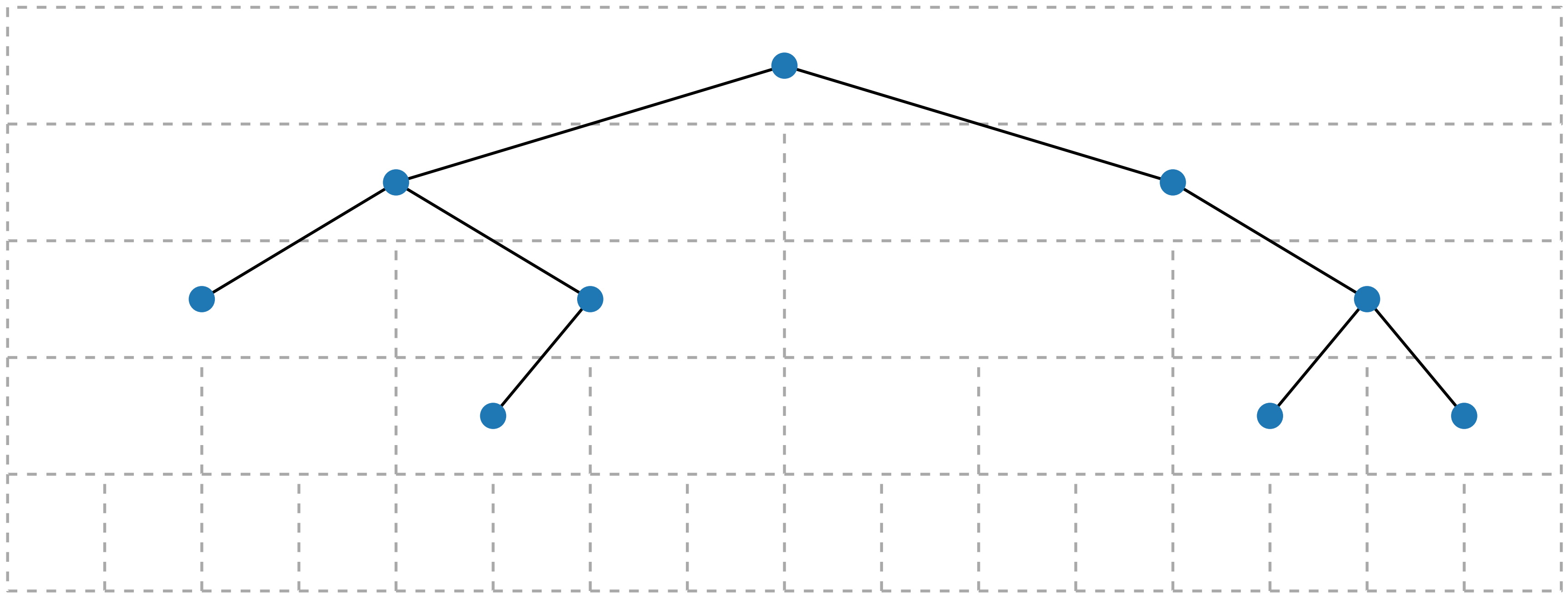
Operations we want to support:

`parent(v)`, `leftChild(v)`, `rightChild(v)`

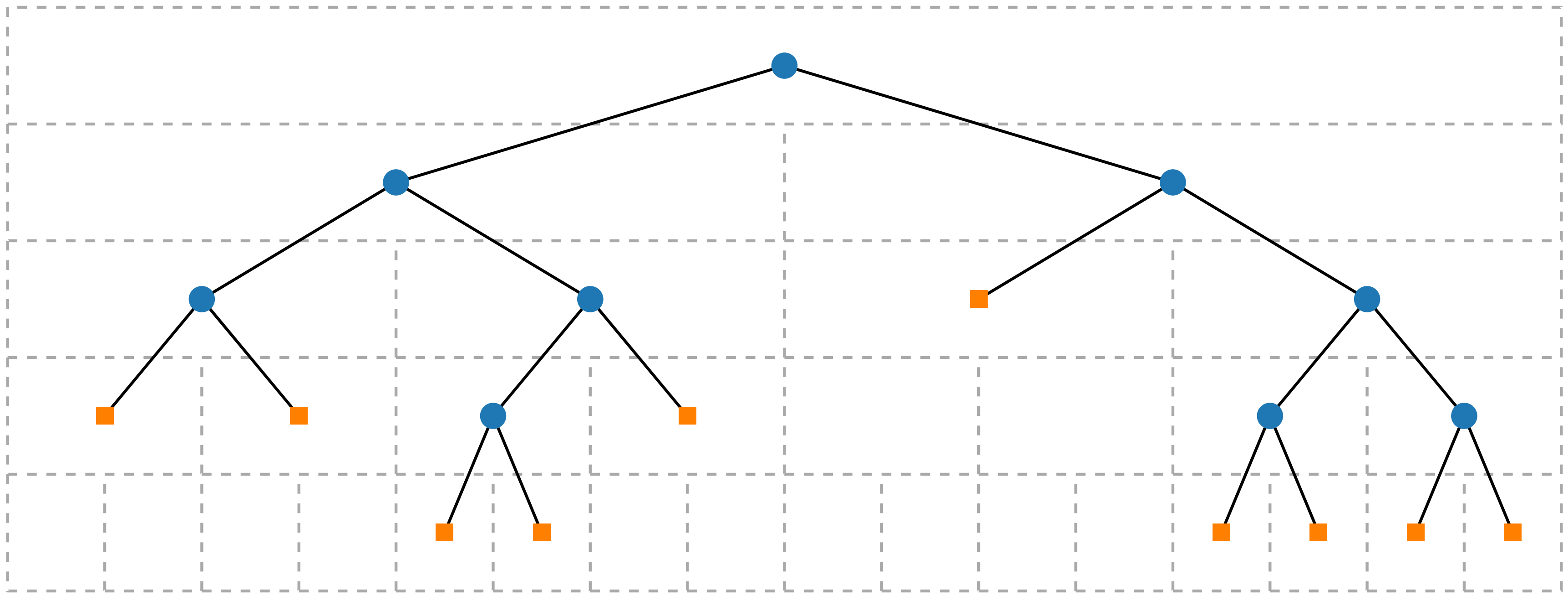
Idea:

- add external nodes
- read internal nodes as 1
- read external nodes as 0
- use `rank` and `select`

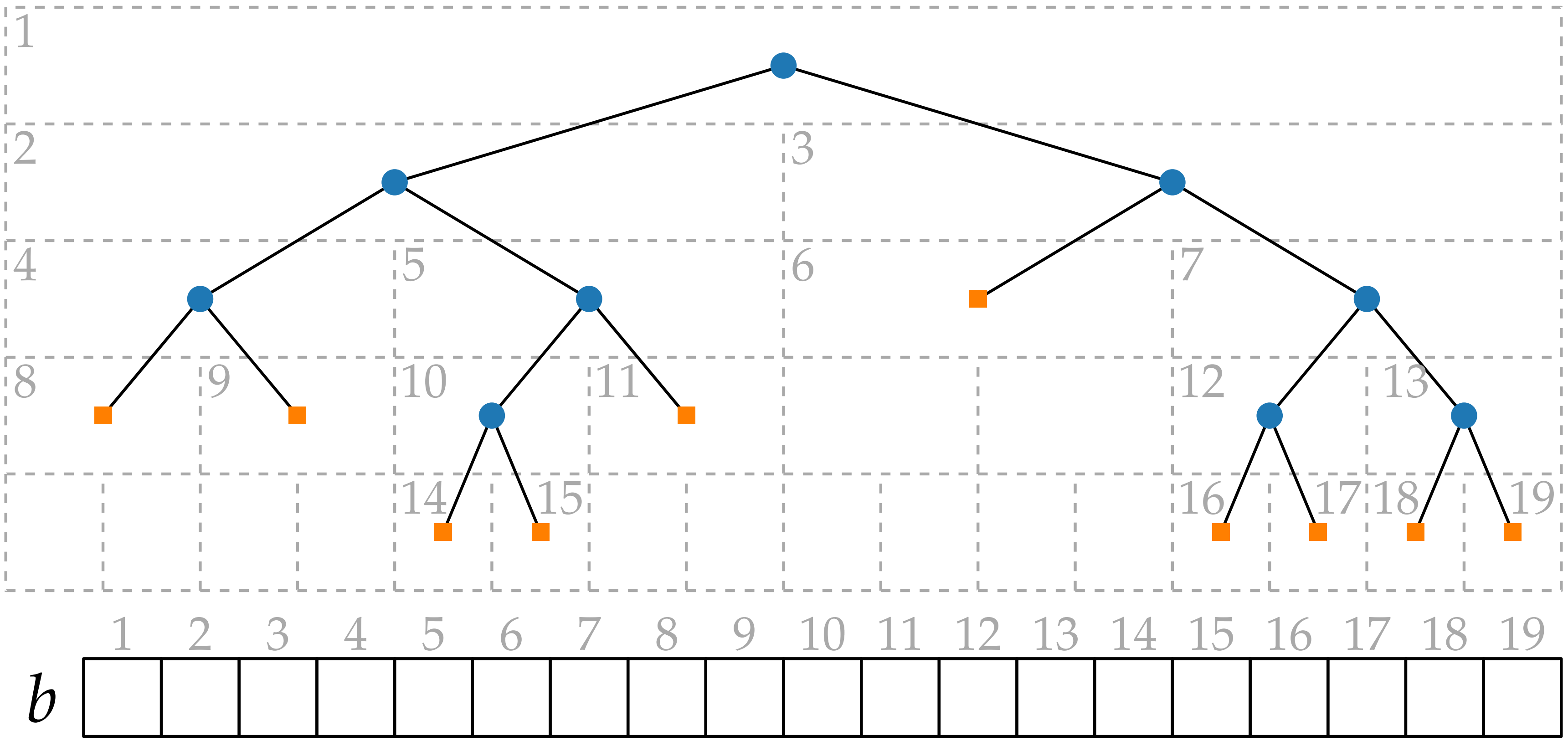
Succinct representation of binary trees



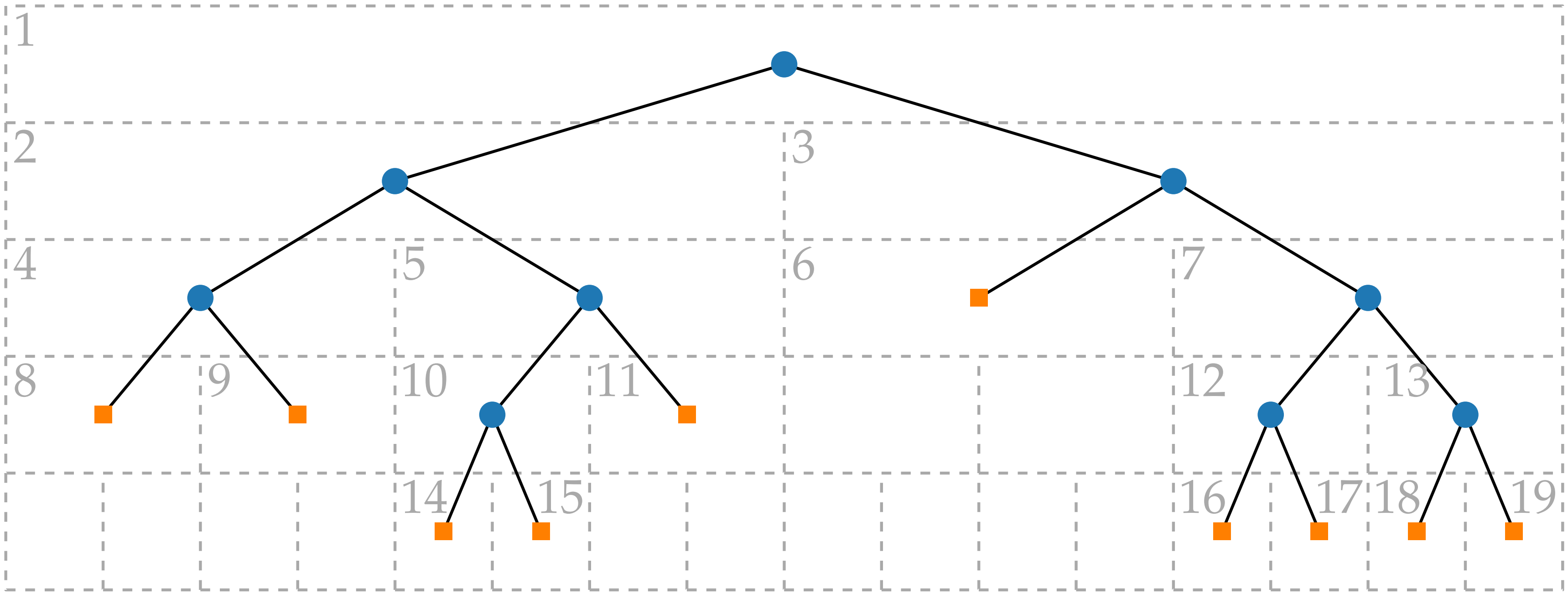
Succinct representation of binary trees



Succinct representation of binary trees

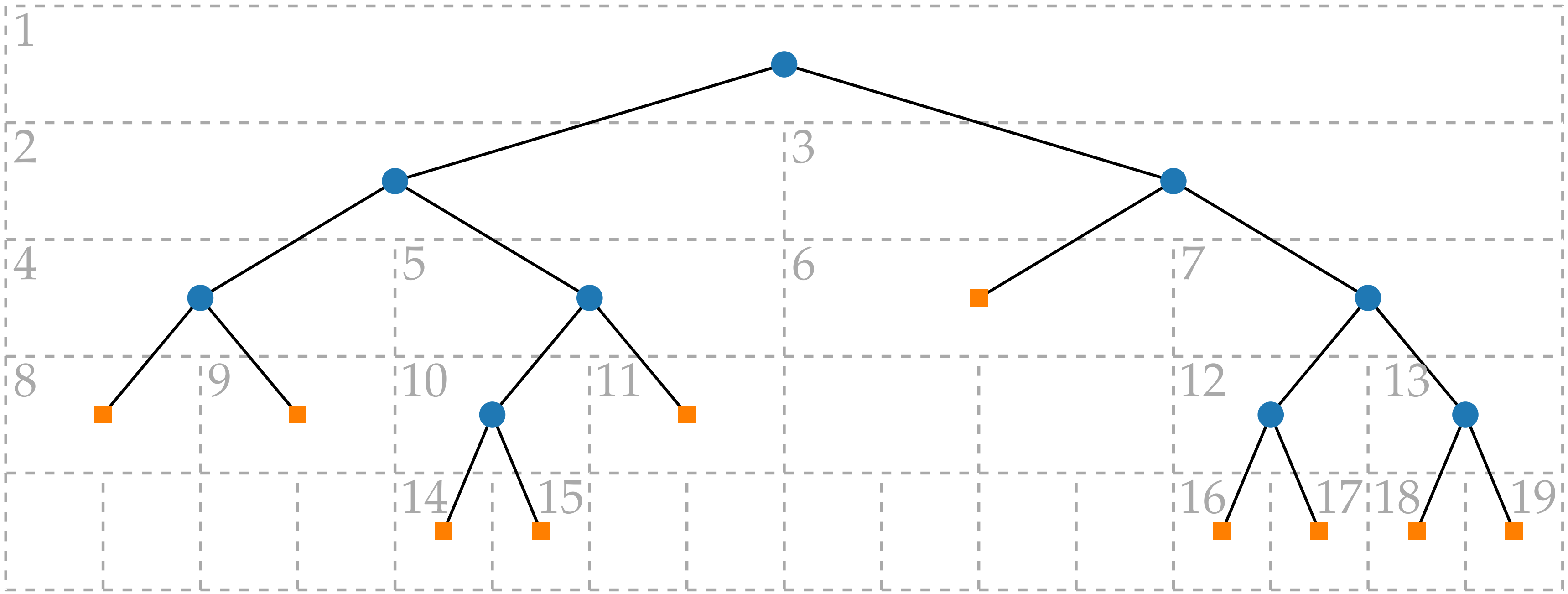


Succinct representation of binary trees



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
b	1	1	1	1	1	0	1	0	0	1	0	1	1	0	0	0	0	0	0

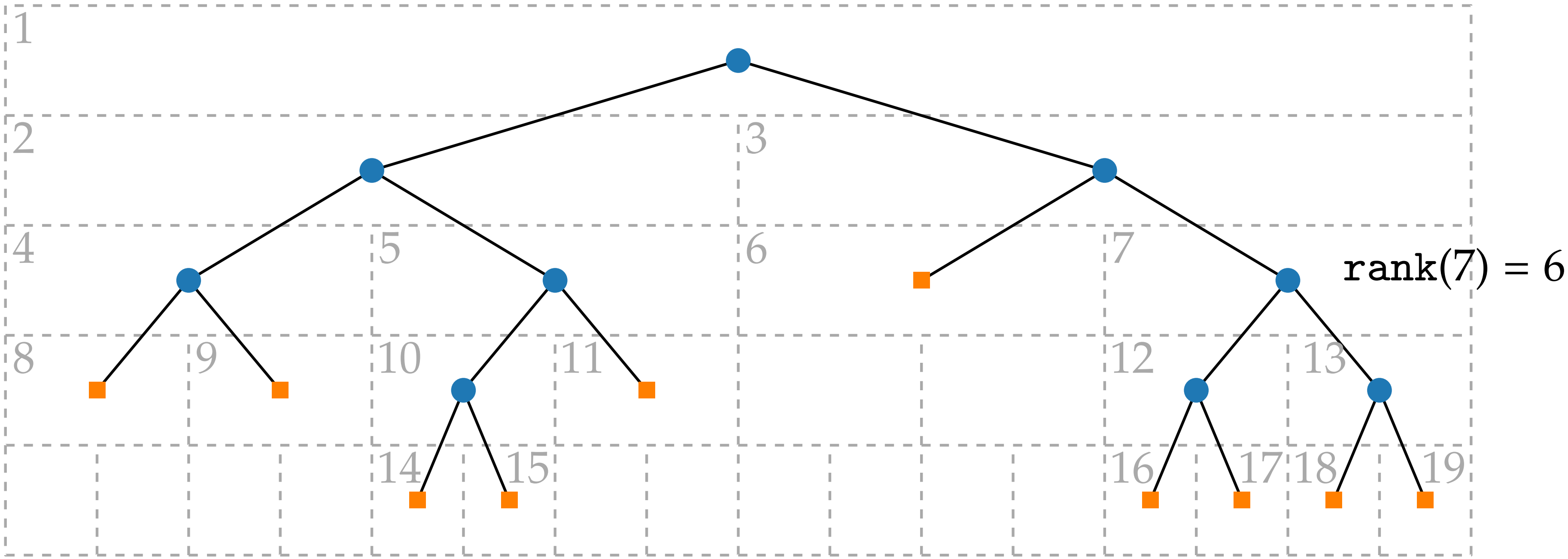
Succinct representation of binary trees



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
b	1	1	1	1	1	0	1	0	0	1	0	1	1	0	0	0	0	0	0

- $\texttt{leftChild}(i) =$
- $\texttt{rightChild}(i) =$

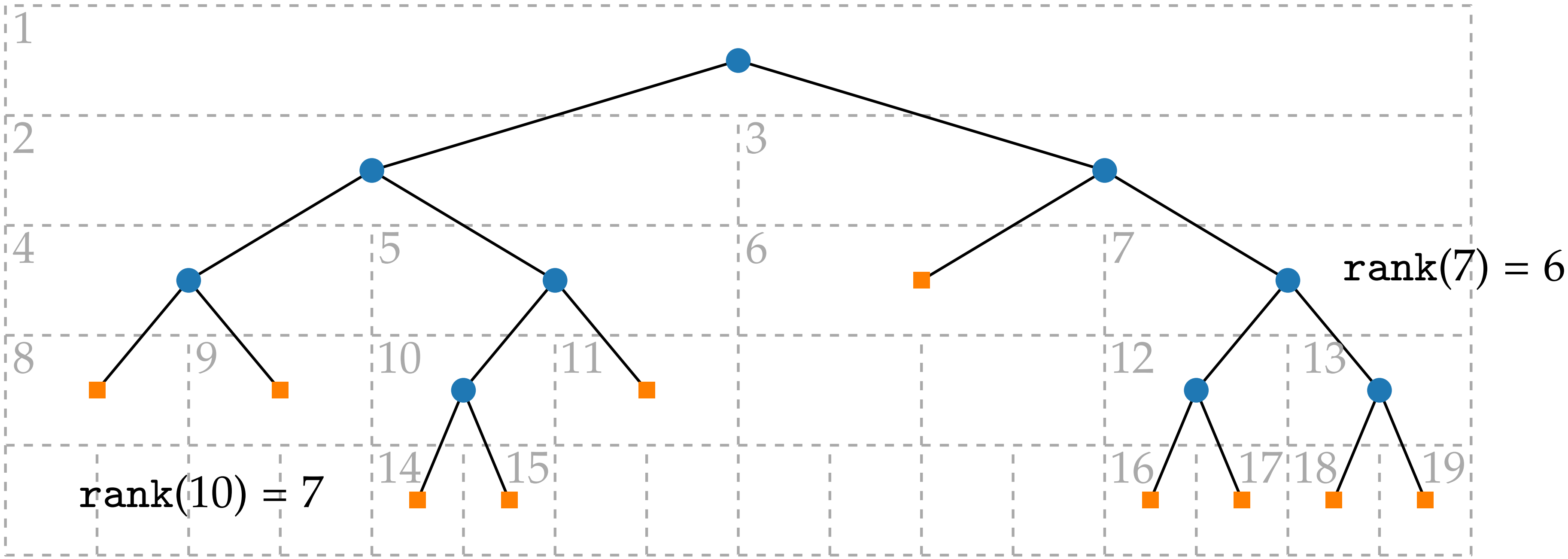
Succinct representation of binary trees



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
b	1	1	1	1	1	0	1	0	0	1	0	1	1	0	0	0	0	0	0

- $\texttt{leftChild}(i) =$
- $\texttt{rightChild}(i) =$

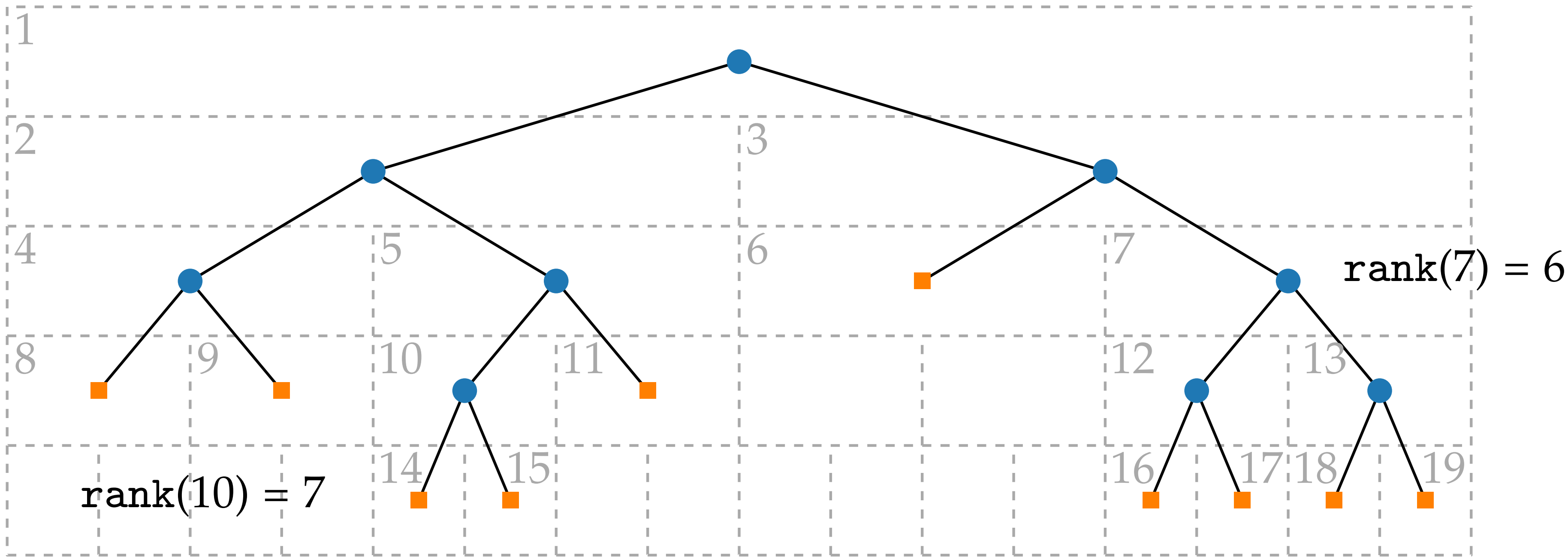
Succinct representation of binary trees



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
b	1	1	1	1	1	0	1	0	0	1	0	1	1	0	0	0	0	0	0

- $\texttt{leftChild}(i) =$
- $\texttt{rightChild}(i) =$

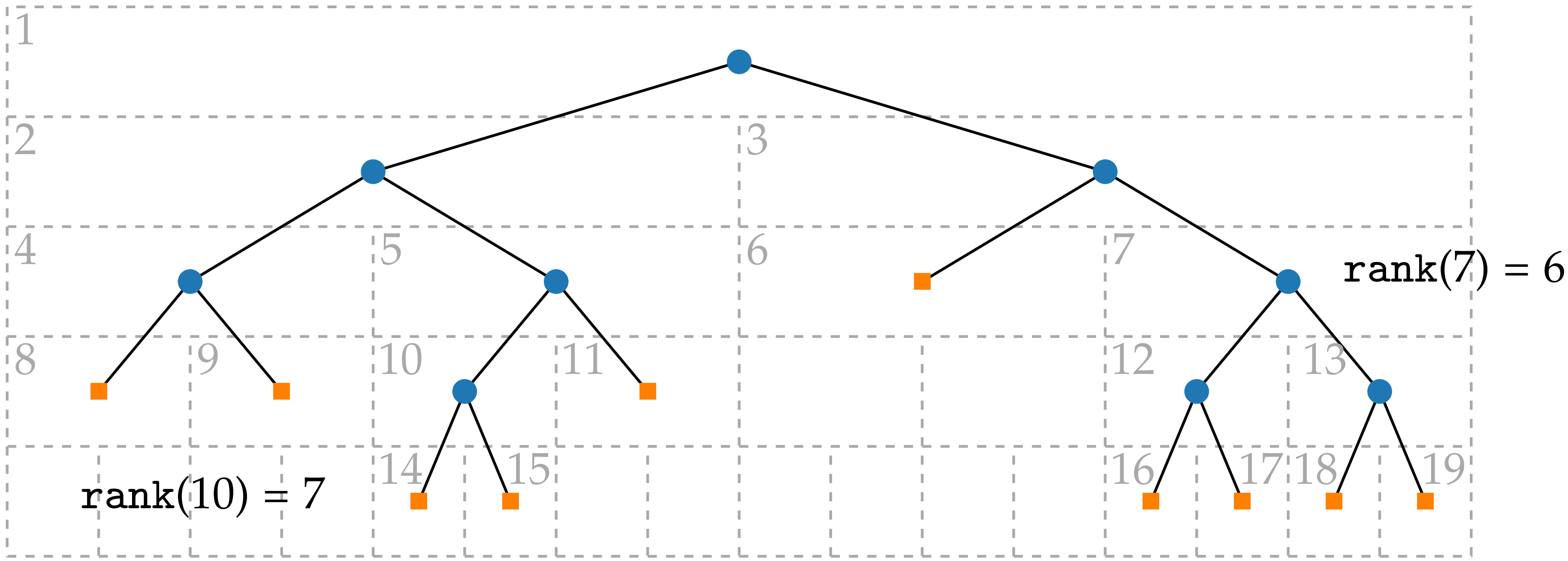
Succinct representation of binary trees



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
b	1	1	1	1	1	0	1	0	0	1	0	1	1	0	0	0	0	0	0

- $\texttt{leftChild}(i) = 2\,\texttt{rank}(i)$
- $\texttt{rightChild}(i) = 2\,\texttt{rank}(i)\ +1$

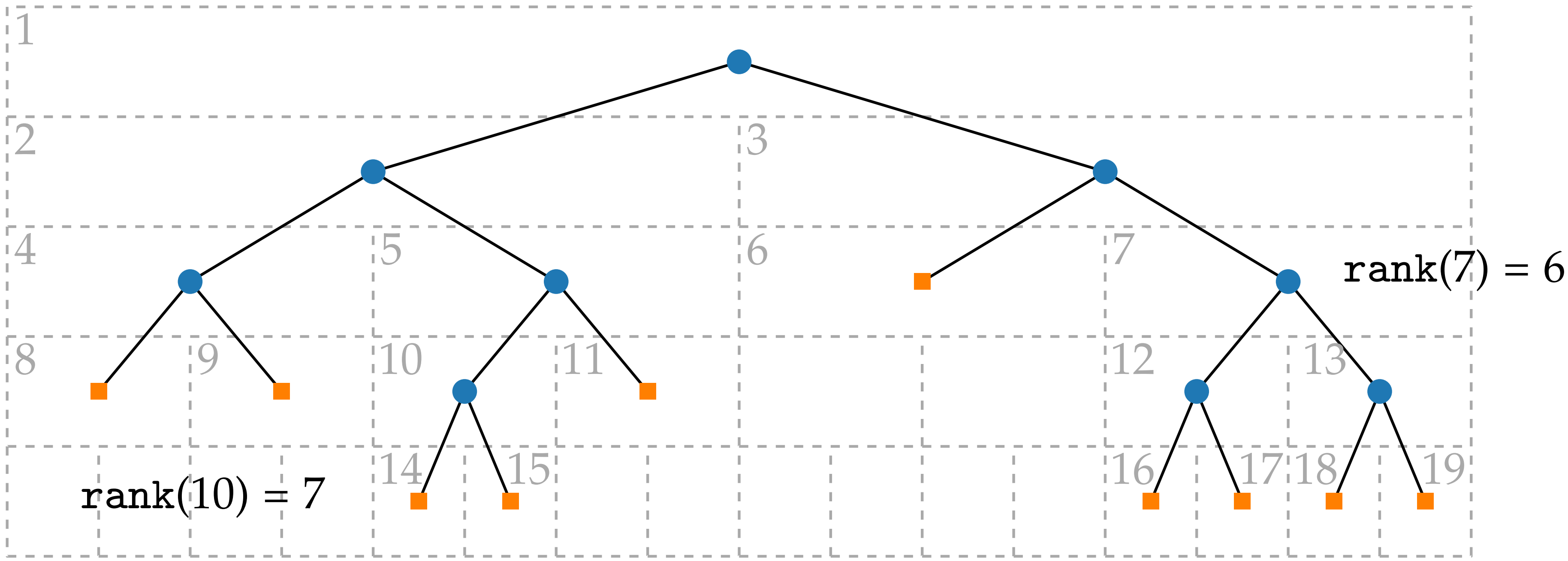
Succinct representation of binary trees



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
b	1	1	1	1	1	0	1	0	0	1	0	1	1	0	0	0	0	0	0

- $\texttt{leftChild}(i) = 2\,\texttt{rank}(i)$
- $\texttt{rightChild}(i) = 2\,\texttt{rank}(i)\ +1$
- $\texttt{parent}(i) =$

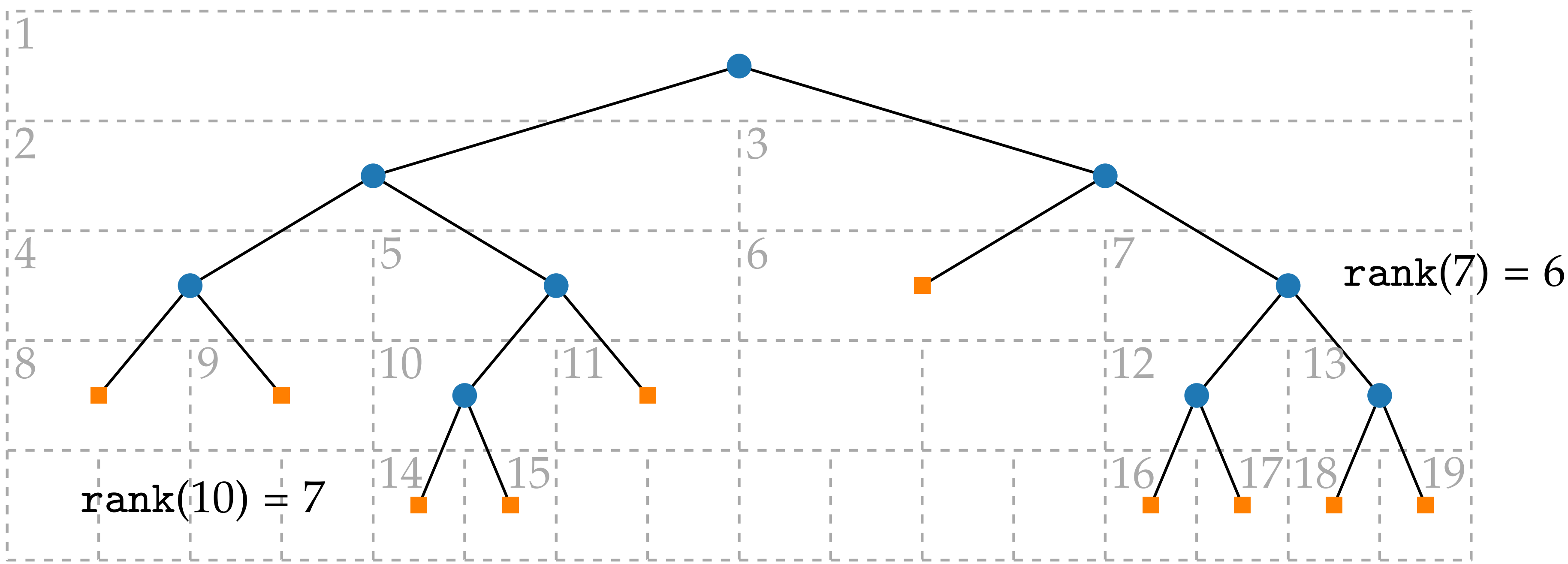
Succinct representation of binary trees



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
b	1	1	1	1	1	0	1	0	0	1	0	1	1	0	0	0	0	0	0

- $\texttt{leftChild}(i) = 2\,\texttt{rank}(i)$
- $\texttt{rightChild}(i) = 2\,\texttt{rank}(i)\ {+1}$
- $\texttt{parent}(i) = \texttt{select}(\lfloor \frac{i}{2} \rfloor)$

Succinct representation of binary trees

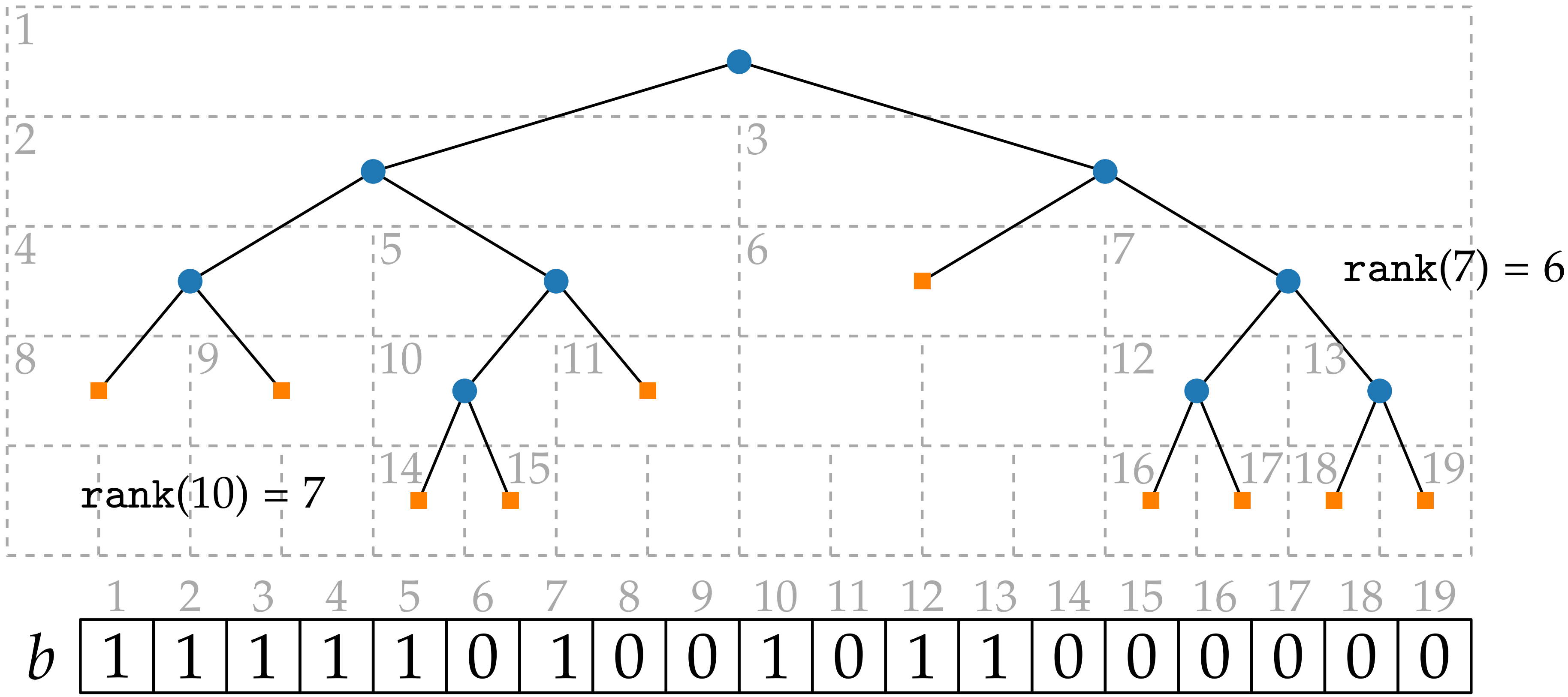


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
b	1	1	1	1	1	0	1	0	0	1	0	1	1	0	0	0	0	0	0

- $\texttt{leftChild}(i) = 2\,\texttt{rank}(i)$
- $\texttt{rightChild}(i) = 2\,\texttt{rank}(i)\ +1$
- $\texttt{parent}(i) = \texttt{select}(\lfloor \frac{i}{2} \rfloor)$

use $\texttt{rank}(i)$ for index in array storing actual values

Succinct representation of binary trees



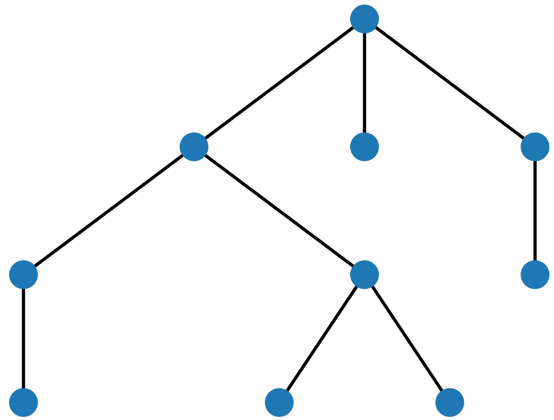
- $\texttt{leftChild}(i) = 2\,\texttt{rank}(i)$
- $\texttt{rightChild}(i) = 2\,\texttt{rank}(i)$ +1
- $\texttt{parent}(i) = \texttt{select}(\lfloor \frac{i}{2} \rfloor)$

use $\texttt{rank}(i)$ for index in array storing actual values

- Size: 2n +1 bits for b, plus $o(n)$ for `rank` and `select`

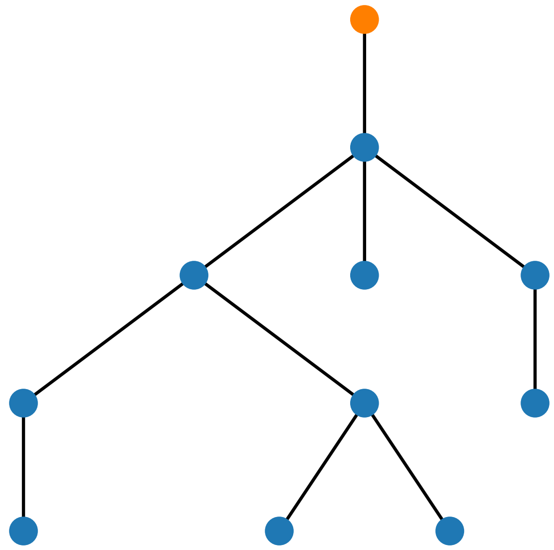
Succinct representation of trees - LOUDS

Level order unary degree sequence

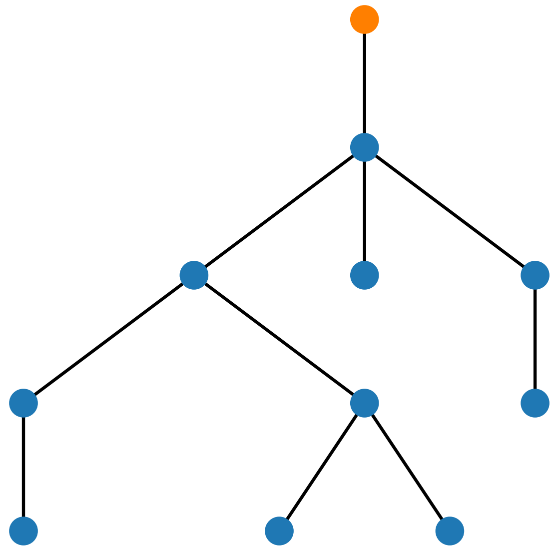


Succinct representation of trees - LOUDS

Level order unary degree sequence



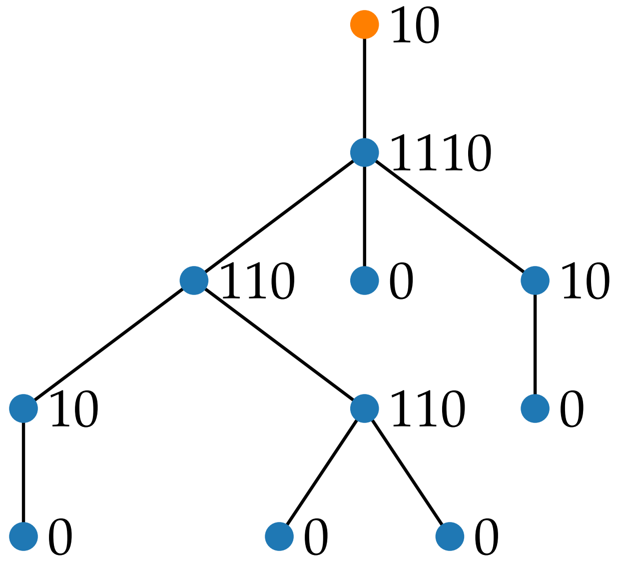
Succinct representation of trees - LOUDS



Level order unary degree sequence

- unary decoding of outdegree

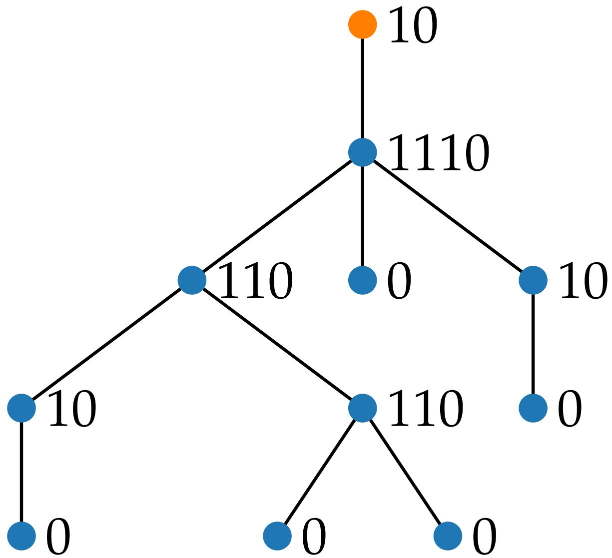
Succinct representation of trees - LOUDS



Level order unary degree sequence

- unary decoding of outdegree

Succinct representation of trees - LOUDS



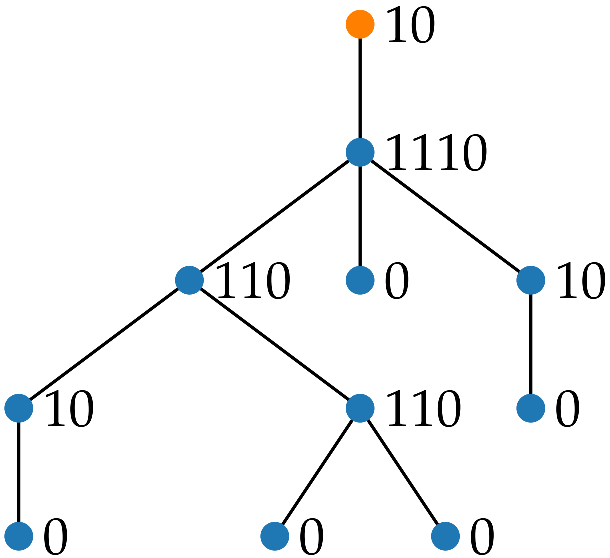
Level order unary degree sequence

- unary decoding of outdegree
- gives LOUDS sequence

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	0	1	1	1	0	1	1	0	0	1	0	1	0	1	1	0	0	0	0	0

- each node represented twice
- use index of its corresponding 1

Succinct representation of trees - LOUDS



Level order unary degree sequence

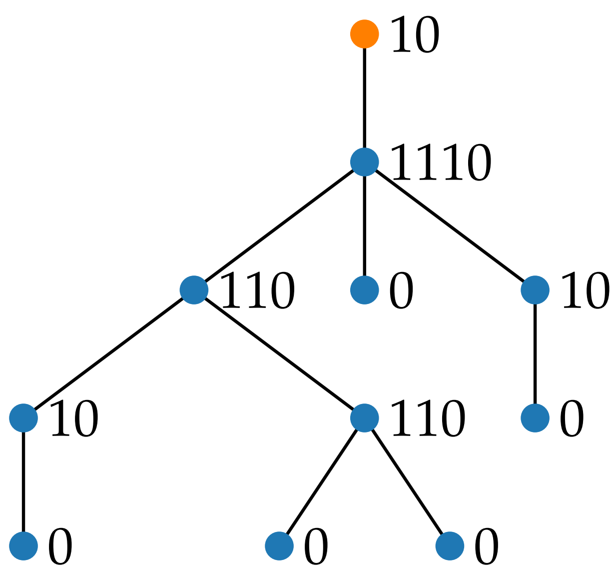
- unary decoding of outdegree
- gives LOUDS sequence

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	0	1	1	1	0	1	1	0	0	1	0	1	0	1	1	0	0	0	0	0

- each node represented twice
- use index of its corresponding 1

$\Rightarrow 2n + o(n)$ bits

Succinct representation of trees - LOUDS



Level order unary degree sequence

- unary decoding of outdegree
- gives LOUDS sequence

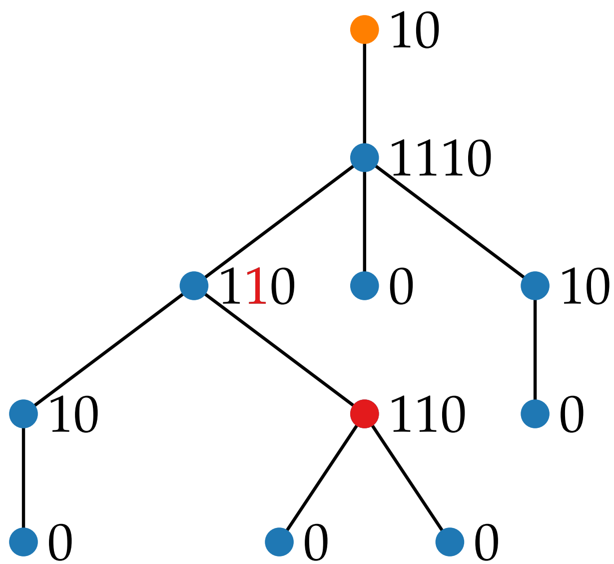
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	0	1	1	1	0	1	1	0	0	1	0	1	0	1	1	0	0	0	0	0

- each node represented twice
- use index of its corresponding 1

$\Rightarrow 2n + o(n)$ bits

- $\texttt{firstChild}(i) = \texttt{select}_0(\texttt{rank}_1(i)) + 1$

Succinct representation of trees - LOUDS



Level order unary degree sequence

- unary decoding of outdegree
- gives LOUDS sequence

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	0	1	1	1	0	1	1	0	0	1	0	1	0	1	1	0	0	0	0	0

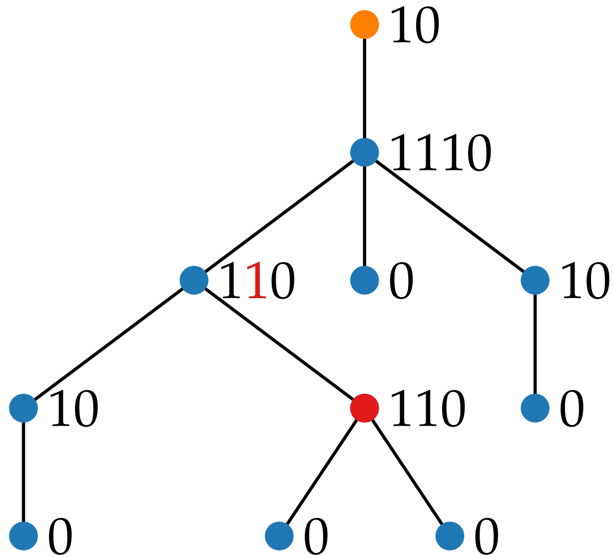
- each node represented twice except root
- use index of its corresponding 1

$\Rightarrow 2n + o(n)$ bits

- $\texttt{firstChild}(i) = \texttt{select}_0(\texttt{rank}_1(i)) + 1$

$\texttt{firstChild}(8) = \texttt{select}_0(\texttt{rank}_1(8)) + 1$

Succinct representation of trees - LOUDS



Level order unary degree sequence

- unary decoding of outdegree
- gives LOUDS sequence

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	0	1	1	1	0	1	1	0	0	1	0	1	0	1	1	0	0	0	0	0

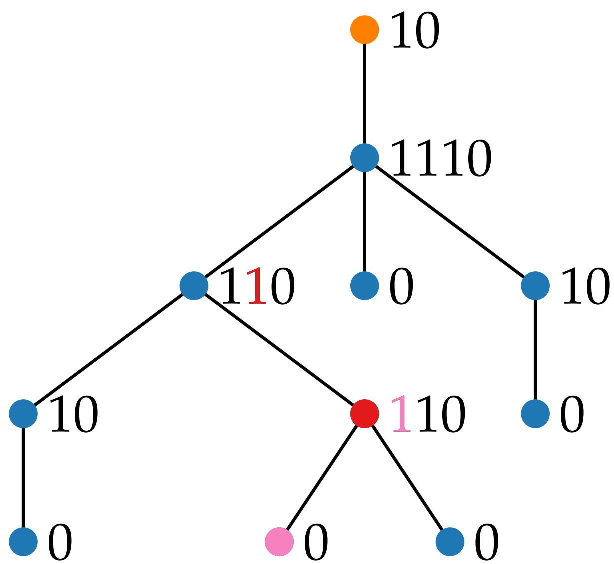
- each node represented twice except root
- use index of its corresponding 1

$\Rightarrow 2n + o(n)$ bits

- $\texttt{firstChild}(i) = \texttt{select}_0(\texttt{rank}_1(i)) + 1$

$\texttt{firstChild}(8) = \texttt{select}_0(\texttt{rank}_1(8)) + 1$
$= \texttt{select}_0(6) + 1$

Succinct representation of trees - LOUDS



Level order unary degree sequence

- unary decoding of outdegree
- gives LOUDS sequence

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	0	1	1	1	0	1	1	0	0	1	0	1	0	1	1	0	0	0	0	0

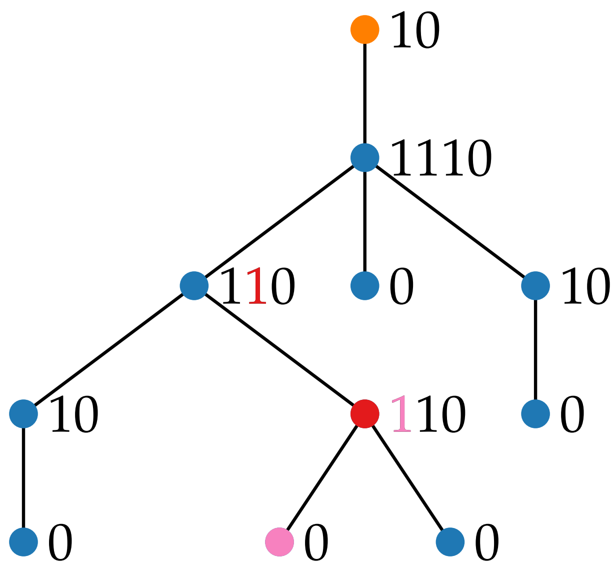
- each node represented twice except root
- use index of its corresponding 1

$\Rightarrow 2n + o(n)$ bits

- $\texttt{firstChild}(i) = \texttt{select}_0(\texttt{rank}_1(i)) + 1$

$\texttt{firstChild}(8) = \texttt{select}_0(\texttt{rank}_1(8)) + 1$
$= \texttt{select}_0(6) + 1 = 10 + 1 = 11$

Succinct representation of trees - LOUDS



Level order unary degree sequence

- unary decoding of outdegree
- gives LOUDS sequence

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	0	1	1	1	0	1	1	0	0	1	0	1	0	1	1	0	0	0	0	0

- each node represented twice (except root)
- use index of its corresponding 1

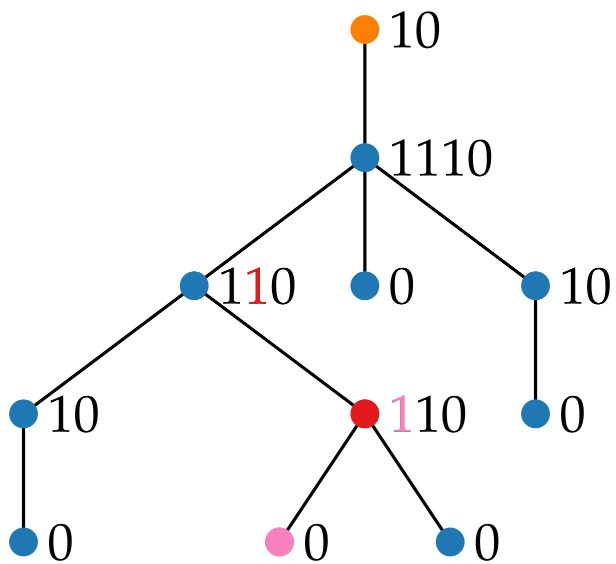
$\Rightarrow 2n + o(n)$ bits

- $\texttt{firstChild}(i) = \texttt{select}_0(\texttt{rank}_1(i)) + 1$

  $\texttt{firstChild}(8) = \texttt{select}_0(\texttt{rank}_1(8)) + 1$
  $= \texttt{select}_0(6) + 1 = 10 + 1 = 11$

- $\texttt{nextSibling}(i) = i + 1$

Succinct representation of trees - LOUDS



Level order unary degree sequence

- unary decoding of outdegree
- gives LOUDS sequence

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	0	1	1	1	0	1	1	0	0	1	0	1	0	1	1	0	0	0	0	0

- each node represented twice except root
- use index of its corresponding 1

$\Rightarrow 2n + o(n)$ bits

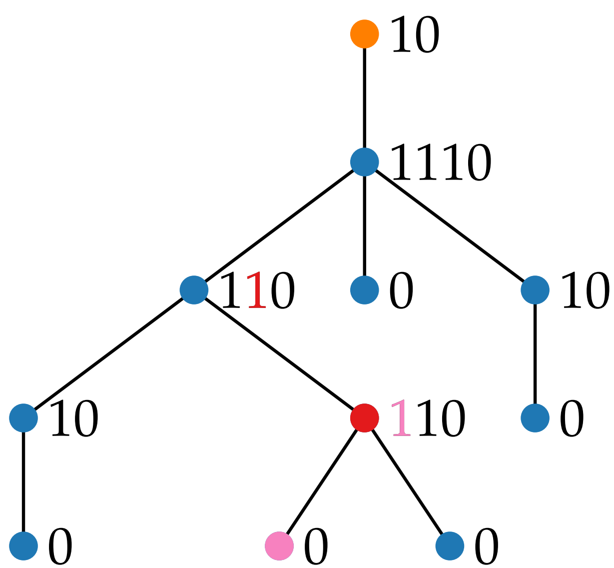
- $\texttt{firstChild}(i) = \texttt{select}_0(\texttt{rank}_1(i)) + 1$

  $\texttt{firstChild}(8) = \texttt{select}_0(\texttt{rank}_1(8)) + 1$
  $= \texttt{select}_0(6) + 1 = 10 + 1 = 11$

- $\texttt{nextSibling}(i) = i + 1$

Exercise: $\texttt{child}(i, j)$ with validity check

Succinct representation of trees - LOUDS



Level order unary degree sequence

- unary decoding of outdegree
- gives LOUDS sequence

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	0	1	1	1	0	1	1	0	0	1	0	1	0	1	1	0	0	0	0	0

- each node represented twice except root
- use index of its corresponding 1

$\Rightarrow 2n + o(n)$ bits

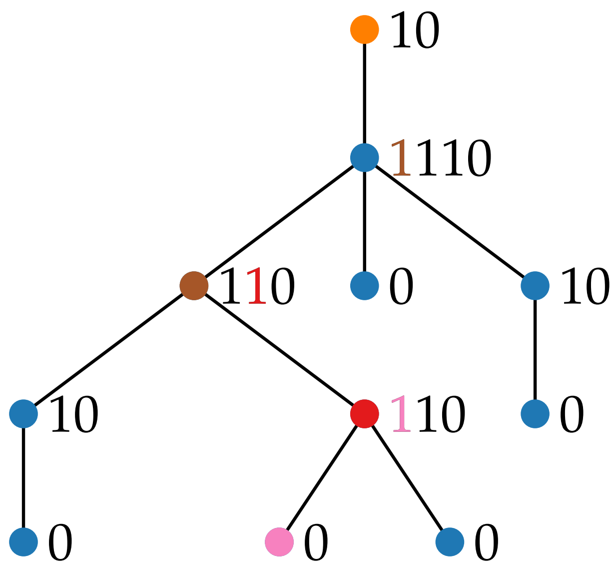
- $\texttt{firstChild}(i) = \texttt{select}_0(\texttt{rank}_1(i)) + 1$

  $\texttt{firstChild}(8) = \texttt{select}_0(\texttt{rank}_1(8)) + 1$
  $= \texttt{select}_0(6) + 1 = 10 + 1 = 11$

Exercise: $\texttt{child}(i, j)$ with validity check

- $\texttt{nextSibling}(i) = i + 1$
- $\texttt{parent}(i) = \texttt{select}_1(\texttt{rank}_0(i))$

Succinct representation of trees - LOUDS



Level order unary degree sequence

- unary decoding of outdegree
- gives LOUDS sequence

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	0	1	1	1	0	1	1	0	0	1	0	1	0	1	1	0	0	0	0	0

- each node represented twice (except root)
- use index of its corresponding 1

$\Rightarrow 2n + o(n)$ bits

- $\texttt{firstChild}(i) = \texttt{select}_0(\texttt{rank}_1(i)) + 1$

  $\texttt{firstChild}(8) = \texttt{select}_0(\texttt{rank}_1(8)) + 1$
  $= \texttt{select}_0(6) + 1 = 10 + 1 = 11$

- $\texttt{nextSibling}(i) = i + 1$
- $\texttt{parent}(i) = \texttt{select}_1(\texttt{rank}_0(i))$

  $\texttt{parent}(8) = \texttt{select}_1(\texttt{rank}_0(8)) = \texttt{select}_1(2) = 3$

Exercise: $\texttt{child}(i, j)$ with validity check

Discussion

- Succinct data structures are
  - space efficient
  - support fast operations

  but
  - are mostly static (dynamic at extra cost),
  - number of operations are limited,
  - complex $\rightarrow$ harder to implement

Discussion

- Succinct data structures are
  - space efficient
  - support fast operations

  but
  - are mostly static (dynamic at extra cost),
  - number of operations are limited,
  - complex $\rightarrow$ harder to implement

- Rank and select form basis for many succinct representations

References

- Lecture 17 of Advanced Data Structures (MIT, Fall’17) by Erik Demaine
- see also Lecture 18 on compact & succinct suffix arrays & trees

References

- Lecture 17 of Advanced Data Structures (MIT, Fall'17) by Erik Demaine
- see also Lecture 18 on compact & succinct suffix arrays & trees

- Guy Jacobson "Space efficient Static Trees and Graphs", FOCS'89
- also contains how to store planar graphs in linear space