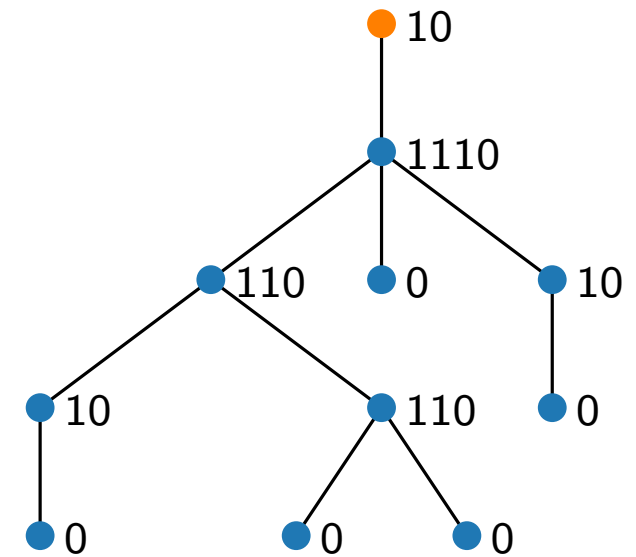
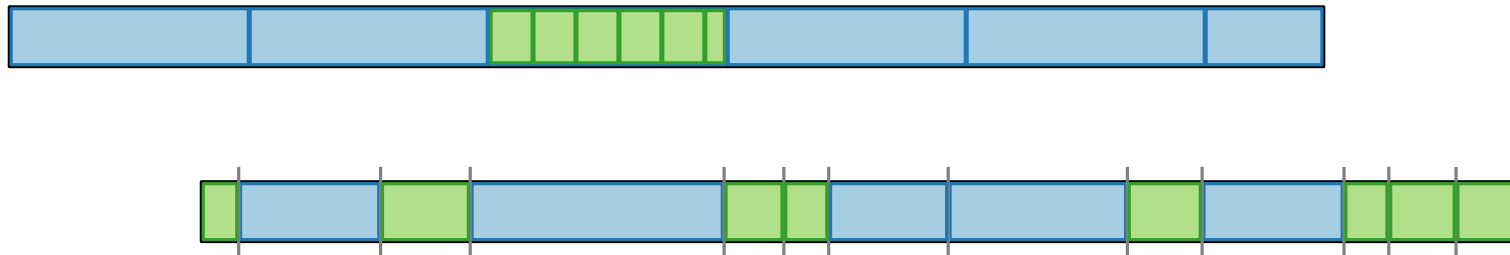


# Advanced Algorithms

## Succinct Data Structures

### Indexable Dictionaries and Trees

Johannes Zink · WS23/24



# Data Structures – Informal Definition

# Data Structures – Informal Definition

A **data structure** is a concept to

- **store**,
- **organize**, and
- **manage** data.

# Data Structures – Informal Definition

A **data structure** is a concept to

- **store**,
- **organize**, and
- **manage** data.

As such, it is a collection of

- **data values**,
- their **relations**, and
- the **operations** that can be applied to the data.

# Data Structures – Informal Definition

A **data structure** is a concept to

- **store**,
- **organize**, and
- **manage** data.

As such, it is a collection of

- **data values**,
- their **relations**, and
- the **operations** that can be applied to the data.

## Remarks.

- We look at data structures as a designer/implementer (and not necessarily as a user).
- To define a data structure and to implement it are two different tasks.

# Data Structures – Informal Definition

A **data structure** is a concept to

- **store**,
- **organize**, and
- **manage** data.



As such, it is a collection of

- **data values**,
- their **relations**, and
- the **operations** that can be applied to the data.

- What do we represent?
- How much space is required?
- Dynamic or static?
- Which operations are defined?
- How fast are they?

## Remarks.

- We look at data structures as a designer/implementer (and not necessarily as a user).
- To define a data structure and to implement it are two different tasks.

# 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.

# 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?

# Examples for **Implicit** Data Structures

# Examples for **Implicit** Data Structures

- **arrays** to represent lists
  - but why not linked lists?

# Examples for **Implicit** Data Structures

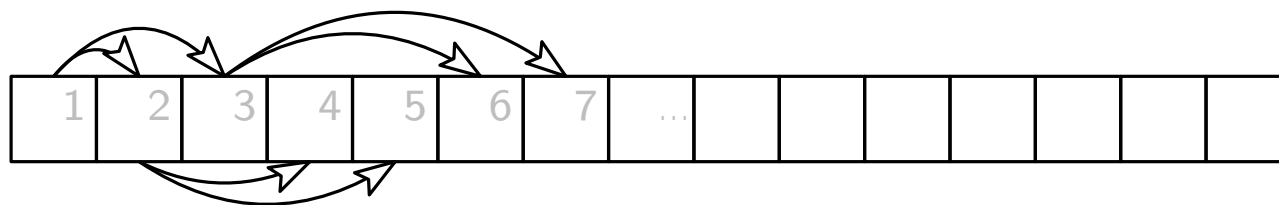
- **arrays** to represent lists
  - but why not linked lists?
- **1-dim arrays** to represent multi-dimensional arrays

# Examples for **Implicit** Data Structures

- **arrays** to represent lists
  - but why not linked lists?
- **1-dim arrays** to represent multi-dimensional arrays
- **sorted arrays** to represent sorted lists
  - but why not binary search trees?

# Examples for **Implicit** Data Structures

- **arrays** to represent lists
  - but why not linked lists?
- **1-dim arrays** to represent multi-dimensional arrays
- **sorted arrays** to represent sorted lists
  - but why not binary search trees?
- **arrays** to represent complete binary trees and heaps



$\text{leftChild}(i) =$

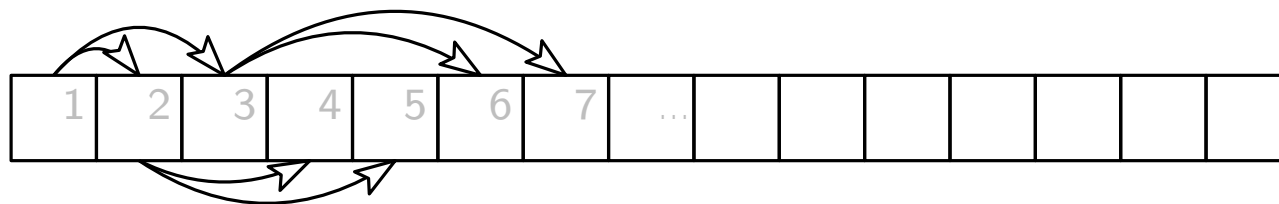
$\text{rightChild}(i) =$

$\text{parent}(i) =$



# Examples for **Implicit** Data Structures

- **arrays** to represent lists
  - but why not linked lists?
- **1-dim arrays** to represent multi-dimensional arrays
- **sorted arrays** to represent sorted lists
  - but why not binary search trees?
- **arrays** to represent complete binary trees and heaps



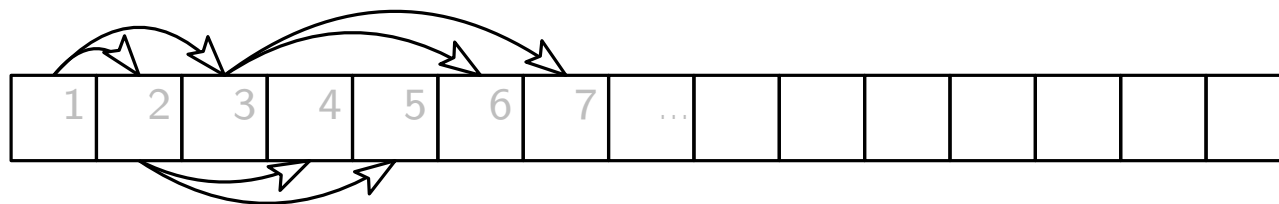
$$\text{leftChild}(i) = 2i$$

$$\text{rightChild}(i) =$$

$$\text{parent}(i) =$$

# Examples for **Implicit** Data Structures

- **arrays** to represent lists
  - but why not linked lists?
- **1-dim arrays** to represent multi-dimensional arrays
- **sorted arrays** to represent sorted lists
  - but why not binary search trees?
- **arrays** to represent complete binary trees and heaps



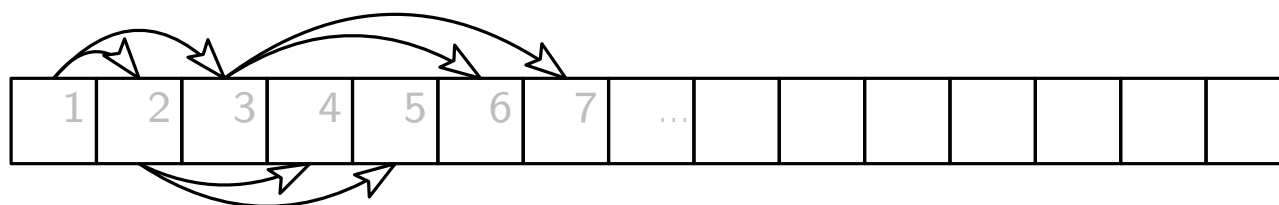
$$\text{leftChild}(i) = 2i$$

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

$$\text{parent}(i) =$$

# Examples for **Implicit** Data Structures

- **arrays** to represent lists
  - but why not linked lists?
- **1-dim arrays** to represent multi-dimensional arrays
- **sorted arrays** to represent sorted lists
  - but why not binary search trees?
- **arrays** to represent complete binary trees and heaps



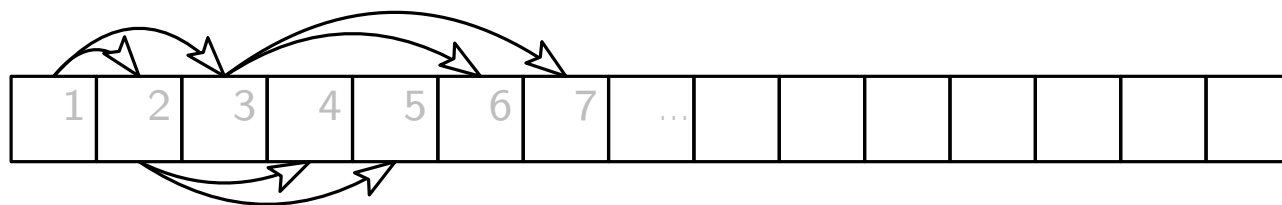
$$\text{leftChild}(i) = 2i$$

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

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

# Examples for **Implicit** Data Structures

- **arrays** to represent lists
  - but why not linked lists?
- **1-dim arrays** to represent multi-dimensional arrays
- **sorted arrays** to represent sorted lists
  - but why not binary search trees?
- **arrays** to represent complete binary trees and heaps



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

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

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

And unbalanced  
trees?

# Succinct Indexable Dictionary

Represent a subset  $S \subseteq \{1, 2, \dots, n\}$  and support the following operations in  $O(1)$  time:

- $\text{member}(i)$  returns if  $i \in S$
- $\text{rank}(i)$  = number of elements in  $S$  that are less or equal to  $i$
- $\text{select}(j)$  =  $j$ -th element in  $S$
- $\text{predecessor}(i)$
- $\text{successor}(i)$

# Succinct Indexable Dictionary

Represent a subset  $S \subseteq \{1, 2, \dots, n\}$  and support the following operations in  $O(1)$  time:

- $\text{member}(i)$  returns if  $i \in S$
- $\text{rank}(i)$  = number of elements in  $S$  that are less or equal to  $i$
- $\text{select}(j)$  =  $j$ -th element in  $S$
- $\text{predecessor}(i)$
- $\text{successor}(i)$

How many different subsets of  $\{1, 2, \dots, n\}$  are there?

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

# Succinct Indexable Dictionary

Represent a subset  $S \subseteq \{1, 2, \dots, n\}$  and support the following operations in  $O(1)$  time:

- $\text{member}(i)$  returns if  $i \in S$
- $\text{rank}(i)$  = number of elements in  $S$  that are less or equal to  $i$
- $\text{select}(j)$  =  $j$ -th element in  $S$
- $\text{predecessor}(i)$
- $\text{successor}(i)$

How many different subsets of  $\{1, 2, \dots, n\}$  are there?  $2^n$

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

# Succinct Indexable Dictionary

Represent a subset  $S \subseteq \{1, 2, \dots, n\}$  and support the following operations in  $O(1)$  time:

- $\text{member}(i)$  returns if  $i \in S$
- $\text{rank}(i)$  = number of elements in  $S$  that are less or equal to  $i$
- $\text{select}(j)$  =  $j$ -th element in  $S$
- $\text{predecessor}(i)$
- $\text{successor}(i)$

How many different subsets of  $\{1, 2, \dots, n\}$  are there?  $2^n$

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

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

our logarithms are all to basis 2, i.e.,  $\log_2$



# 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

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

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

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

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

$\text{member}(i)$  can trivially be answered in  $O(1)$  time  
(assuming that we can access any entry in constant time)

# 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 data structures to answer in  $O(1)$  time

- $\text{rank}(i) = \#$  1s at or before position  $i$  number of
- $\text{select}(j) =$  position of  $j$ -th 1 bit

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

	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

$\text{member}(i)$  can trivially be answered in  $O(1)$  time  
(assuming that we can access any entry in constant time)

# 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 data structures to answer in  $O(1)$  time

- $\text{rank}(i) = \#$  1s at or before position  $i$  number of
- $\text{select}(j) =$  position of  $j$ -th 1 bit

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

$\text{select}(5) =$

	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

$\text{member}(i)$  can trivially be answered in  $O(1)$  time  
(assuming that we can access any entry in constant time)

# 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 data structures to answer in  $O(1)$  time

- $\text{rank}(i) = \#$  1s at or before position  $i$  number of
- $\text{select}(j) =$  position of  $j$ -th 1 bit

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

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

	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

$\text{member}(i)$  can trivially be answered in  $O(1)$  time  
(assuming that we can access any entry in constant time)

# 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 data structures to answer in  $O(1)$  time

- $\text{rank}(i) = \#$  1s at or before position  $i$  number of
- $\text{select}(j) =$  position of  $j$ -th 1 bit

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

	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

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

$$\text{rank}(9) =$$

$\text{member}(i)$  can trivially be answered in  $O(1)$  time  
(assuming that we can access any entry in constant time)

# 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 data structures to answer in  $O(1)$  time

- $\text{rank}(i) = \#$  1s at or before position  $i$  number of
- $\text{select}(j) =$  position of  $j$ -th 1 bit

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

	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

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

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

$\text{member}(i)$  can trivially be answered in  $O(1)$  time  
(assuming that we can access any entry in constant time)



# 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 data structures to answer in  $O(1)$  time

- $\text{rank}(i) = \#$  1s at or before position  $i$  number of
- $\text{select}(j) =$  position of  $j$ -th 1 bit

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

	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

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

$$\text{rank}(9) = 5 = \text{rank}(12)$$

$\text{member}(i)$  can trivially be answered in  $O(1)$  time  
(assuming that we can access any entry in constant time)

# 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 data structures to answer in  $O(1)$  time

- $\text{rank}(i) = \#$  1s at or before position  $i$  number of
- $\text{select}(j) =$  position of  $j$ -th 1 bit

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

	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

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

$$\text{rank}(9) = 5 = \text{rank}(12)$$

$$\text{rank}(15) =$$

$\text{member}(i)$  can trivially be answered in  $O(1)$  time  
(assuming that we can access any entry in constant time)

# 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 data structures to answer in  $O(1)$  time

- $\text{rank}(i) = \#$  1s at or before position  $i$  number of
- $\text{select}(j) =$  position of  $j$ -th 1 bit

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

	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

$\text{member}(i)$  can trivially be answered in  $O(1)$  time  
(assuming that we can access any entry in constant time)

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

$$\text{rank}(9) = 5 = \text{rank}(12)$$

$$\text{rank}(15) = 6$$

# 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 data structures to answer in  $O(1)$  time

- $\text{rank}(i) = \#$  1s at or before position  $i$  number of  $\Rightarrow$  Exercise: Use these methods to answer  $\text{predecessor}(i)$  and  $\text{successor}(i)$  in  $O(1)$  time.
- $\text{select}(j) =$  position of  $j$ -th 1 bit

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

	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

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

$$\text{rank}(9) = 5 = \text{rank}(12)$$

$$\text{rank}(15) = 6$$

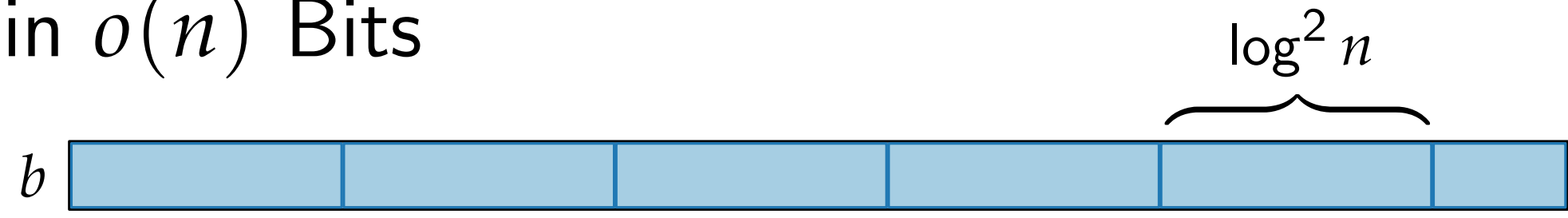
$\text{member}(i)$  can trivially be answered in  $O(1)$  time  
(assuming that we can access any entry in constant time)

# Rank in $o(n)$ Bits

 $b$ 

# Rank in $o(n)$ Bits

$$\log^2 n = (\log n)^2$$



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

# Rank in $o(n)$ Bits

$$\log^2 n = (\log n)^2$$



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

# Rank in $o(n)$ Bits

$$\log^2 n = (\log n)^2$$



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

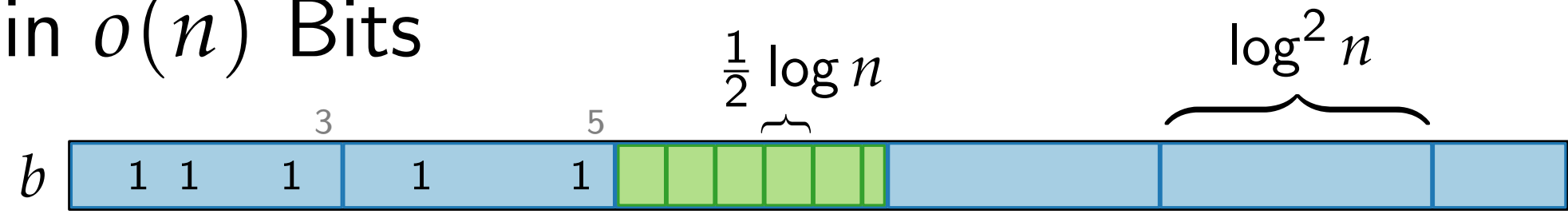
and store cumulative rank: each needs  $\leq \log n$  bits

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



# Rank in $o(n)$ Bits

$$\log^2 n = (\log n)^2$$



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

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

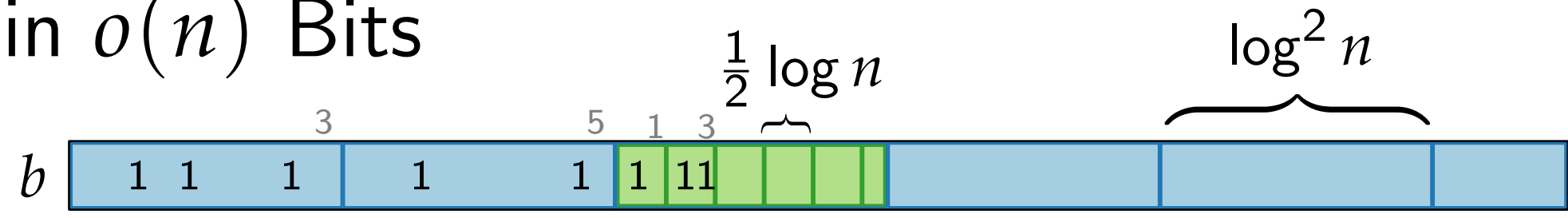
$$\Rightarrow O\left(\frac{n}{\log^2 n} \log n\right) = O\left(\frac{n}{\log n}\right) \subseteq o(n) \text{ bits}$$

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

and store cumulative rank within **chunk**:

# Rank in $o(n)$ Bits

$$\log^2 n = (\log n)^2$$



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

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

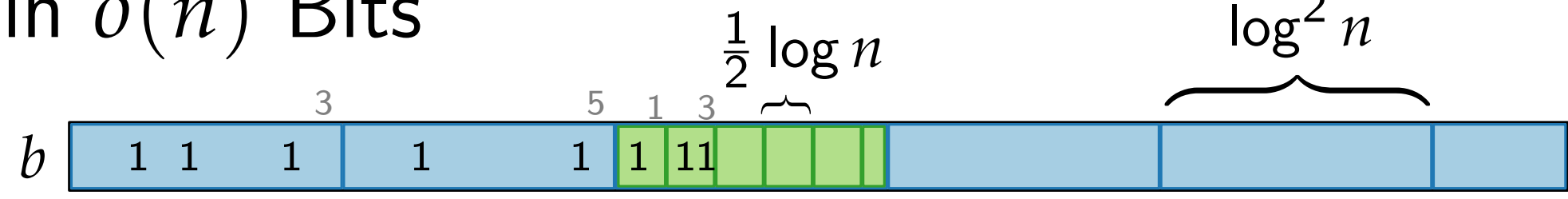
$$\Rightarrow O\left(\frac{n}{\log^2 n} \log n\right) = O\left(\frac{n}{\log n}\right) \subseteq o(n) \text{ bits}$$

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

and store cumulative rank within **chunk**:

# Rank in $o(n)$ Bits

$$\log^2 n = (\log n)^2$$



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

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

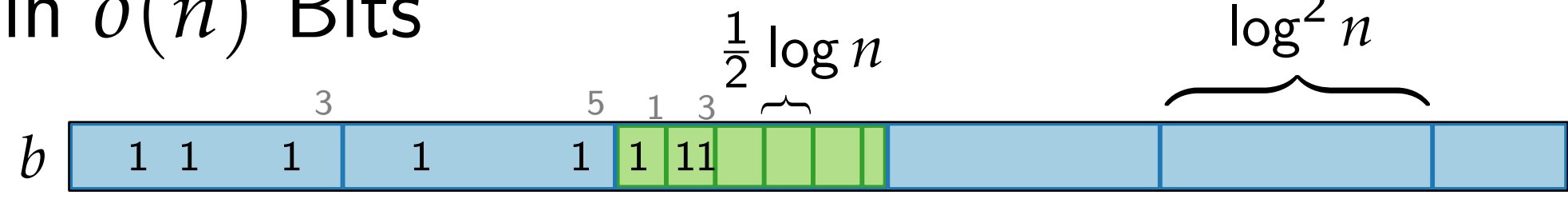
$$\Rightarrow O\left(\frac{n}{\log^2 n} \log n\right) = O\left(\frac{n}{\log n}\right) \subseteq o(n) \text{ bits}$$

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

and store cumulative rank within **chunk**: each needs  $\leq \log \log^2 n = 2 \log \log n$  bits

# Rank in $o(n)$ Bits

$$\log^2 n = (\log n)^2$$



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

and store cumulative rank: each needs  $\leq \log n$  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**

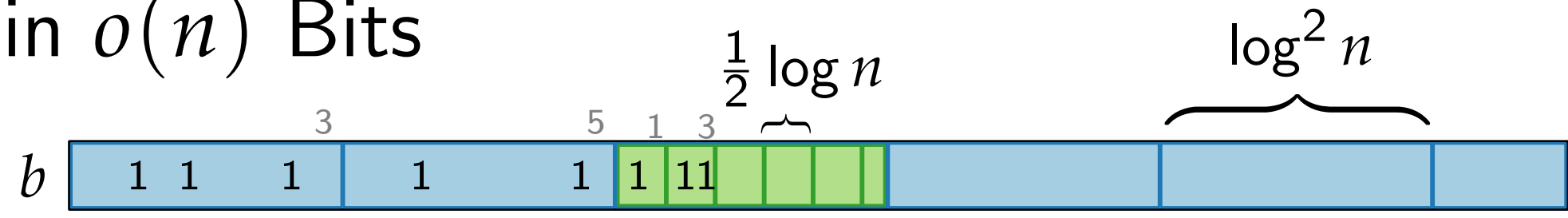
and store cumulative rank within **chunk**: each needs  $\leq \log \log^2 n = 2 \log \log n$  bits

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

# subchunks      rel. rank

# Rank in $o(n)$ Bits

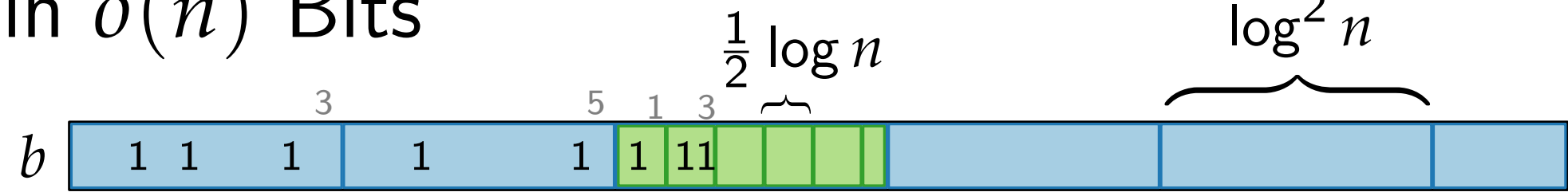
$$\log^2 n = (\log n)^2$$



1. Split into  $(\log^2 n)$ -bit **chunks**  
 and store cumulative rank: each needs  $\leq \log n$  bits  
 $\Rightarrow O\left(\frac{n}{\log^2 n} \log n\right) = O\left(\frac{n}{\log n}\right) \subseteq o(n)$  bits
  
2. Split **chunks** into  $(\frac{1}{2} \log n)$ -bit **subchunks**  
 and store cumulative rank within **chunk**: each needs  $\leq \log \log^2 n = 2 \log \log n$  bits  
 $\Rightarrow O\left(\frac{n}{\log n} \log \log n\right) \subseteq o(n)$  bits
  
3. Use **lookup table** for bitstrings of length  $(\frac{1}{2} \log n)$ :

# Rank in $o(n)$ Bits

$$\log^2 n = (\log n)^2$$



**Example:**  $n = 64 \Rightarrow \frac{1}{2} \log n = 3$

1. Split and
2. Split and
3. Use **lookup table** for bitstrings of length  $(\frac{1}{2} \log n)$ :

$s \leq \log n$  bits

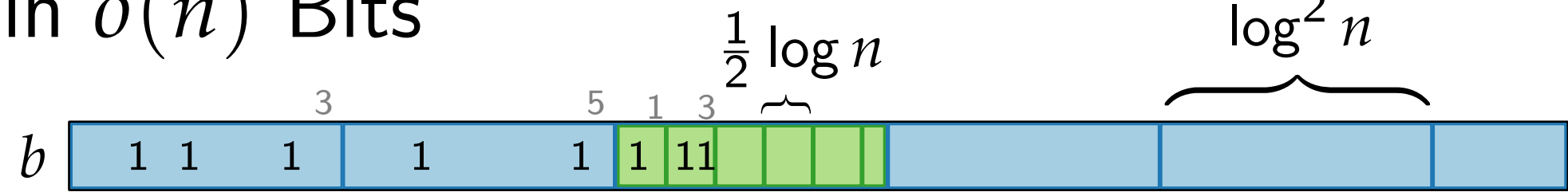
$$O\left(\frac{n}{\log^2 n} \log n\right) = O\left(\frac{n}{\log n}\right) \subseteq o(n) \text{ bits}$$

**ks**

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

# Rank in $o(n)$ Bits

$$\log^2 n = (\log n)^2$$



1. Split  
and

**Example:**  $n = 64 \Rightarrow \frac{1}{2} \log n = 3$

position $\rightarrow$	1	2	3
000	0	0	0
001	0	0	1
010	0	1	1
011	0	1	2
100	1	1	1
101	1	1	2
110	1	2	2
111	1	2	3

2. Split  
and

s  $\leq \log n$  bits

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

ks

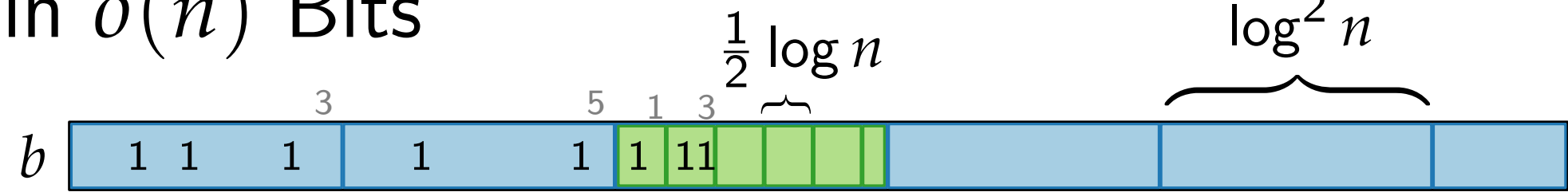
each needs  $\leq \log \log^2 n = 2 \log \log n$  bits

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

3. Use **lookup table** for bitstrings of length  $(\frac{1}{2} \log n)$ :

# Rank in $o(n)$ Bits

$$\log^2 n = (\log n)^2$$



1. Split and
2. Split and
3. Use **lookup table** for bitstrings of length  $(\frac{1}{2} \log n)$ :

**Example:**  $n = 64 \Rightarrow \frac{1}{2} \log n = 3$

position $\rightarrow$	1	2	3
000	0	0	0
001	0	0	1
010	0	1	1
011	0	1	2
100	1	1	1
101	1	1	2
110	1	2	2
111	1	2	3

s  $\leq \log n$  bits

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

ks

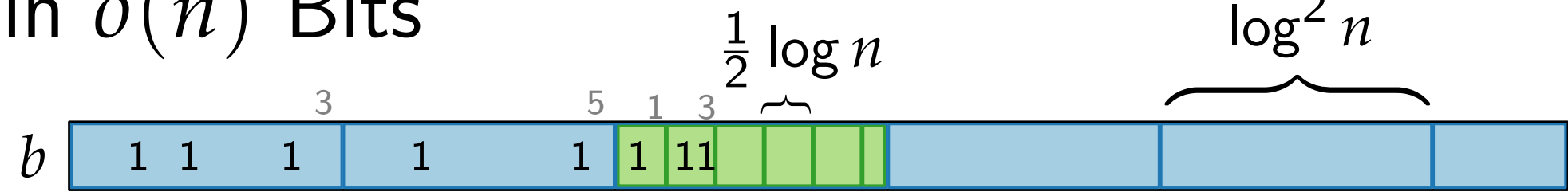
each needs  $\leq \log \log^2 n = 2 \log \log n$  bits  
 $\Rightarrow O\left(\frac{n}{\log n} \log \log n\right) \subseteq o(n)$  bits

3. Use **lookup table** for bitstrings of length  $(\frac{1}{2} \log n)$ :  $2^{\frac{1}{2} \log n} = \sqrt{n}$  distinct bitstrings



# Rank in $o(n)$ Bits

$$\log^2 n = (\log n)^2$$



- 1. Split
- and
- 2. Split
- and

**Example:**  $n = 64 \Rightarrow \frac{1}{2} \log n = 3$

position $\rightarrow$	1	2	3
000	0	0	0
001	0	0	1
010	0	1	1
011	0	1	2
100	1	1	1
101	1	1	2
110	1	2	2
111	1	2	3

s  $\leq \log n$  bits

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

ks

each needs  $\leq \log \log^2 n = 2 \log \log n$  bits

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

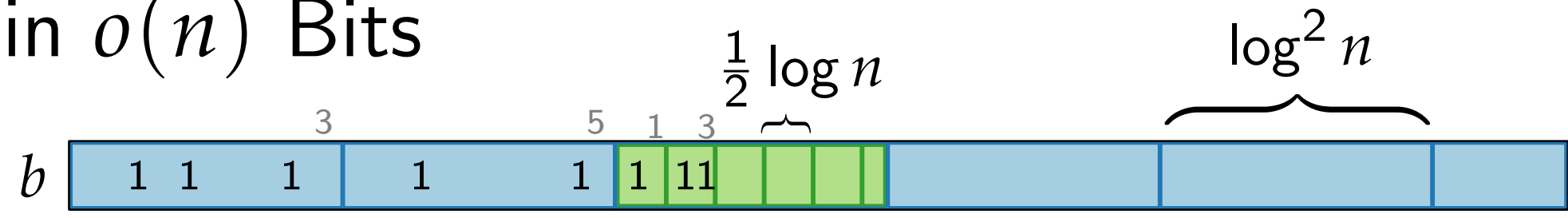
3. Use **lookup table** for bitstrings of length  $\left(\frac{1}{2} \log n\right)$ :  $2^{\frac{1}{2} \log n} = \sqrt{n}$  distinct bitstrings

$$\Rightarrow O\left(\underbrace{\sqrt{n}}_{\# \text{ rows}} \underbrace{\log n}_{\# \text{ columns}} \underbrace{\log \log n}_{\text{rel. rank}}\right) \subseteq o(n) \text{ bits}$$

# rows # columns rel. rank

# Rank in $o(n)$ Bits

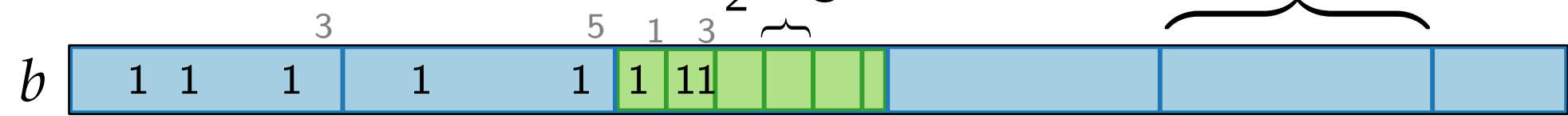
$$\log^2 n = (\log n)^2$$



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

# Rank in $o(n)$ Bits + $O(1)$ Time

$$\log^2 n = (\log n)^2$$

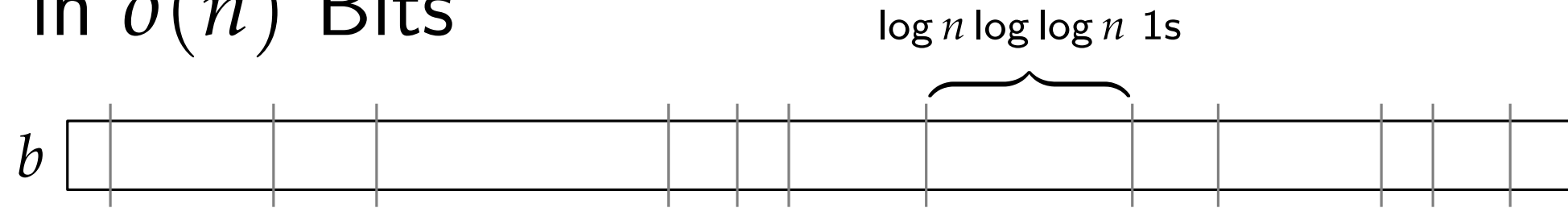


1. Split into  $(\log^2 n)$ -bit **chunks**  
 and store cumulative rank: each needs  $\leq \log n$  bits  
 $\Rightarrow O\left(\frac{n}{\log^2 n} \log n\right) = O\left(\frac{n}{\log n}\right) \subseteq o(n)$  bits
2. Split **chunks** into  $(\frac{1}{2} \log n)$ -bit **subchunks**  
 and store cumulative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

Select in  $o(n)$  Bits

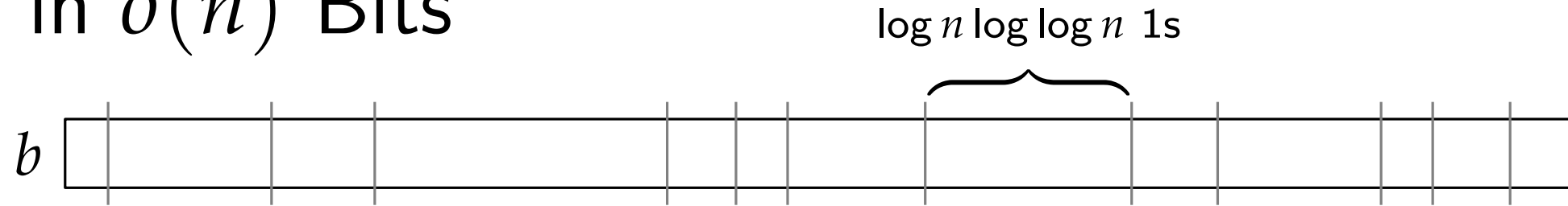
$b$

# 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

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

# Select in $o(n)$ Bits

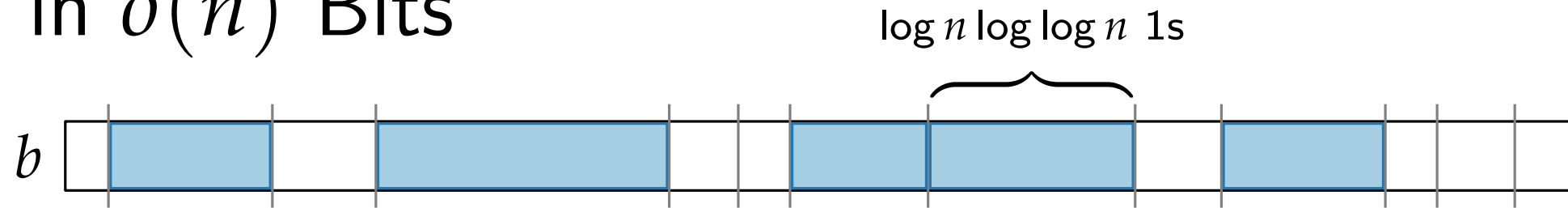


1. Store indices of every  $(\log n \log \log n)$ -th 1 bit in array

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

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

# Select in $o(n)$ Bits



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

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

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

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

then store indices of 1 bits in group in array

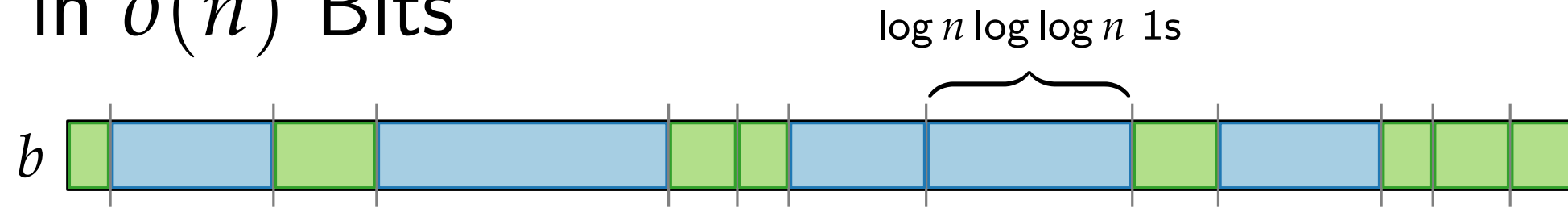
$$\Rightarrow O\left(\underbrace{\frac{n}{(\log n \log \log n)^2}}_{\# \text{ groups}} \underbrace{(\log n \log \log n)}_{\# \text{ 1 bits}} \underbrace{\log n}_{\text{index}}\right) \subseteq O\left(\frac{n}{\log \log n}\right) \text{ bits}$$





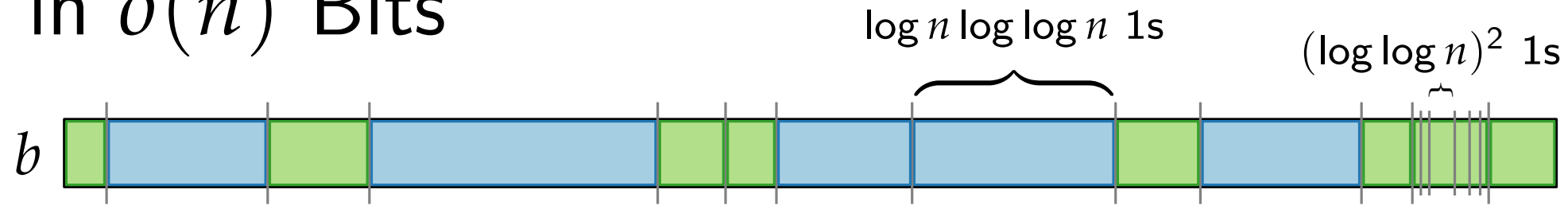


# Select in $o(n)$ Bits



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

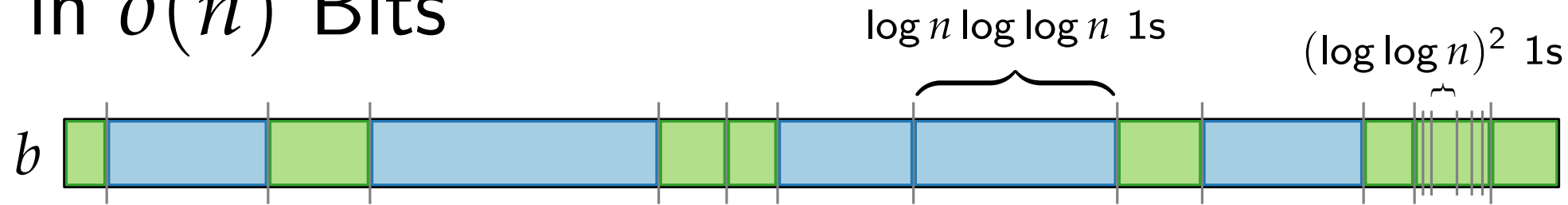
# Select in $o(n)$ Bits



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

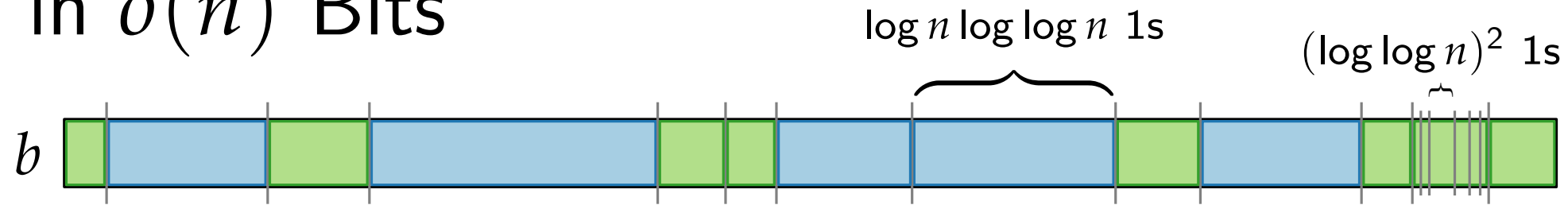


3. Repeat 1. and 2. on reduced bitstrings ( $r < (\log n \log \log n)^2$ ):

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

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

# Select in $o(n)$ Bits



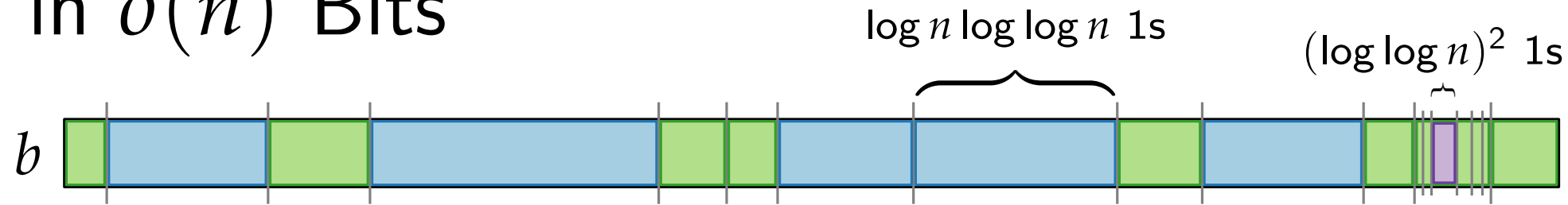
3. Repeat 1. and 2. on reduced bitstrings ( $r < (\log n \log \log n)^2$ ):

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  $(\log \log n)^2$  1 bits of length  $r'$  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

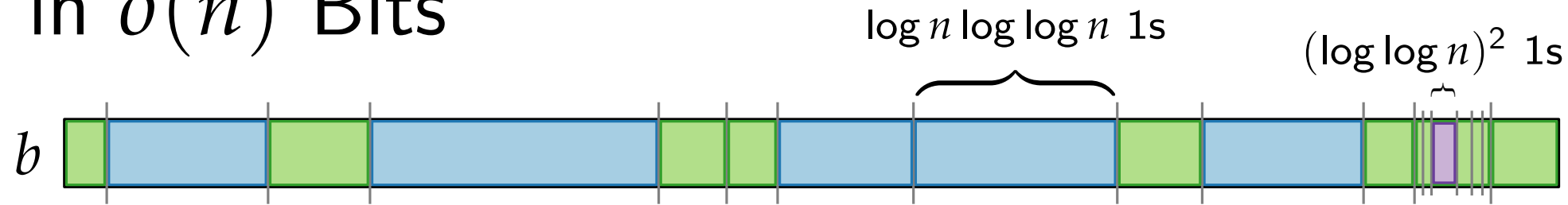
$$\Rightarrow O\left(\frac{n}{(\log \log n)^2} \log \log n\right) = O\left(\frac{n}{\log \log n}\right) \text{ bits}$$

2' Within group of  $(\log \log n)^2$  1 bits of length  $r'$  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  $(\log \log n)^2$  1 bits of length  $r'$  bits:

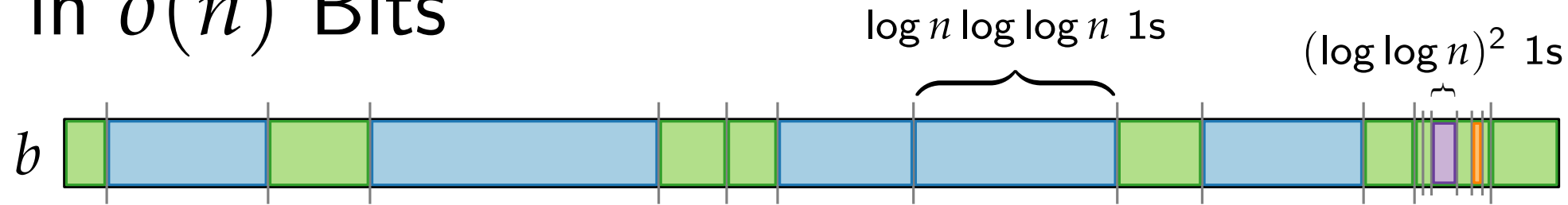
if  $r' \geq (\log \log n)^4$

then store relative indices of 1 bits in subgroup in array

$$\Rightarrow O\left(\underbrace{\frac{n}{(\log \log n)^4}}_{\# \text{ subgroups}} \underbrace{(\log \log n)^2}_{\# \text{ 1 bits}} \underbrace{\log \log n}_{\text{rel. index}}\right) = O\left(\frac{n}{\log \log n}\right) \text{ bits}$$



# 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  $(\log \log n)^2$  1 bits of length  $r'$  bits:

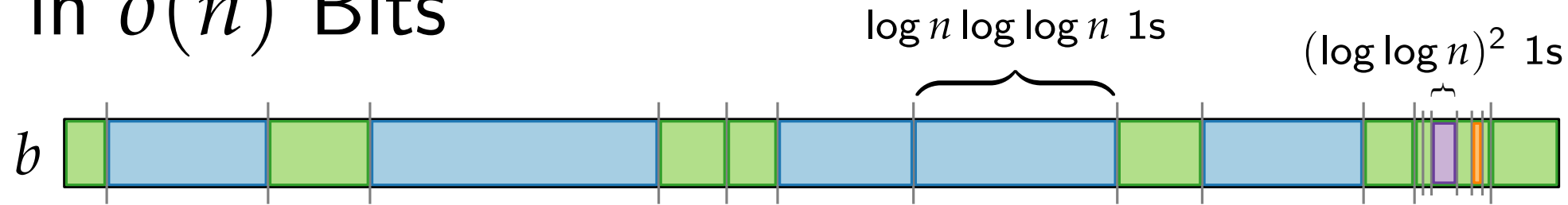
if  $r' \geq (\log \log n)^4$

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 problem is 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  $(\log \log n)^2$  1 bits of length  $r'$  bits:

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

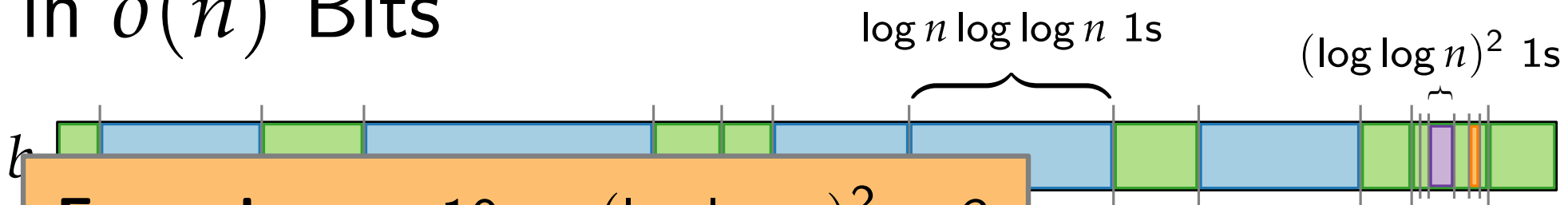
then store relative indices of 1 bits in subgroup in array

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

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

4. Use **lookup table** for bitstrings of length  $r' \leq (\log \log n)^4$ :

# Select in $o(n)$ Bits



3. Repeat  
1' Store  
2' With  
if  $r'$   
then

**Example:**  $n = 10 \Rightarrow (\log \log n)^2 \approx 3$   
 $\Rightarrow r' < (\log \log n)^4 \approx 9$

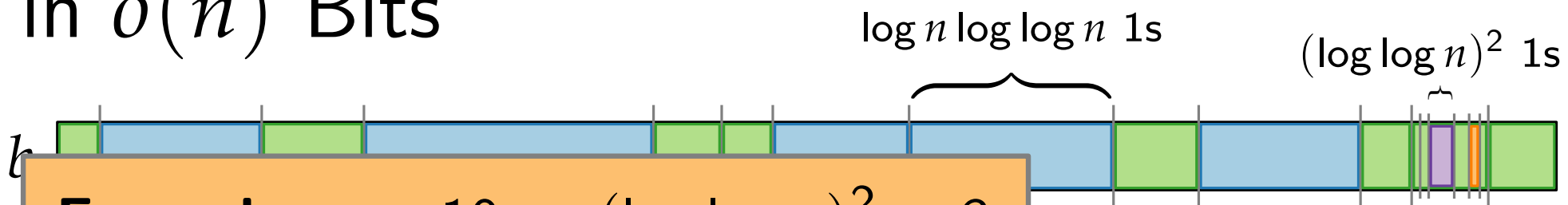
	select $\rightarrow$	1	2	3
bitstring	00000111	6	7	8
	00001011	5	7	8
	00001101	5	6	8
	⋮			
	11001000	1	2	5
	11010000	1	2	4
	11100000	1	2	3

$(n \log \log n)^2$ :  
 1 bit in array  
 $(n)^2 \log \log n) = O(\frac{n}{\log \log n})$  bits  
 $r'$  bits:  
 up in array  
 $(n)^2 \log \log n) = O(\frac{n}{\log \log n})$  bits

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

4. Use **lookup table** for bitstrings of length  $r' \leq (\log \log n)^4$ :

# Select in $o(n)$ Bits



**Example:**  $n = 10 \Rightarrow (\log \log n)^2 \approx 3$   
 $\Rightarrow r' < (\log \log n)^4 \approx 9$

	select $\rightarrow$	1	2	3
00000111	6	7	8	
00001011	5	7	8	
00001101	5	6	8	
⋮				
11001000	1	2	5	
11010000	1	2	4	
11100000	1	2	3	

- 3. Repeat
- 1' Store
- 2' With
- if  $r'$
- then

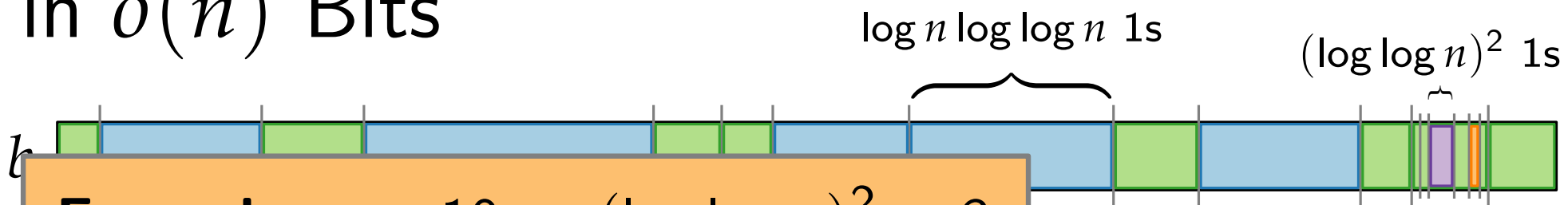
$(n \log \log n)^2$ :  
 1 bit in array  
 $(n)^2 \log \log n) = O(\frac{n}{\log \log n})$  bits  
 $r'$  bits:  
 up in array  
 $(n)^2 \log \log n) = O(\frac{n}{\log \log n})$  bits

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

- 4. Use **lookup table** for bitstrings of length  $r' \leq (\log \log n)^4$ :

$\underbrace{2^{(\log \log n)^4}}_{\# \text{ rows}} \in O(2^{\frac{1}{2} \log n}) = O(\sqrt{n}); \underbrace{(\log \log n)^2}_{\# \text{ columns}} \in O(\log n)$

# Select in $o(n)$ Bits



**Example:**  $n = 10 \Rightarrow (\log \log n)^2 \approx 3$   
 $\Rightarrow r' < (\log \log n)^4 \approx 9$

	select $\rightarrow$	1	2	3
00000111	6	7	8	
00001011	5	7	8	
00001101	5	6	8	
⋮				
11001000	1	2	5	
11010000	1	2	4	
11100000	1	2	3	

- 3. Repeat
- 1' Store
- 2' With
- if  $r'$
- then

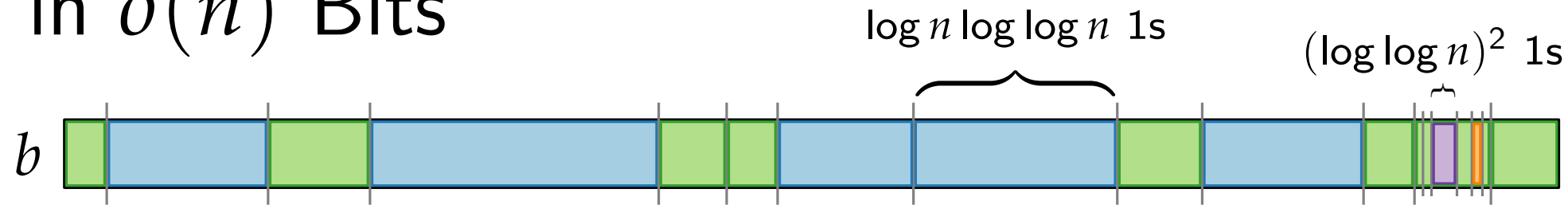
$(n \log \log n)^2$ :  
 1 bit in array  
 $(n)^2 \log \log n) = O(\frac{n}{\log \log n})$  bits  
 $r'$  bits:  
 up in array  
 $(n)^2 \log \log n) = O(\frac{n}{\log \log n})$  bits

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

- 4. Use **lookup table** for bitstrings of length  $r' \leq (\log \log n)^4$ :

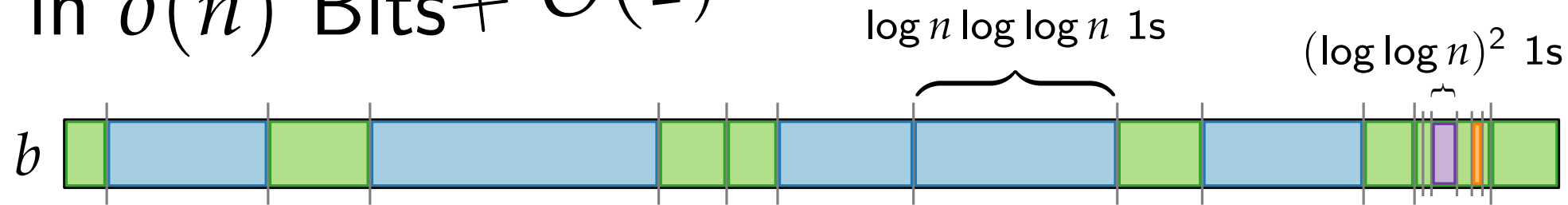
$$\underbrace{2^{(\log \log n)^4}}_{\# \text{ rows}} \in O(2^{\frac{1}{2} \log n}) = O(\sqrt{n}); \quad \underbrace{(\log \log n)^2}_{\# \text{ columns}} \in O(\log n) \Rightarrow \underbrace{O(\sqrt{n})}_{\# \text{ rows}} \underbrace{\log n}_{\# \text{ columns}} \underbrace{\log \log n}_{\text{rel. index}} = o(n) \text{ bits}$$

# Select in $o(n)$ Bits



4.  $\text{select}(j) = \text{select } J\text{-th group where } J = \lfloor j / (\log n \log \log n) \rfloor$ 
  - + directly select  $(j - J)$ -th 1 bit  
or select  $J'$ -th subgroup where  $J' = \lfloor (j - J) / (\log \log n)^2 \rfloor$
  - + directly select  $(j - J - J')$ -th 1 bit  
or select it in the lookup table

Select in  $o(n)$  Bits +  $O(1)$  Time



4.  $\text{select}(j) = \text{select } J\text{-th group where } J = \lfloor j / (\log n \log \log n) \rfloor$

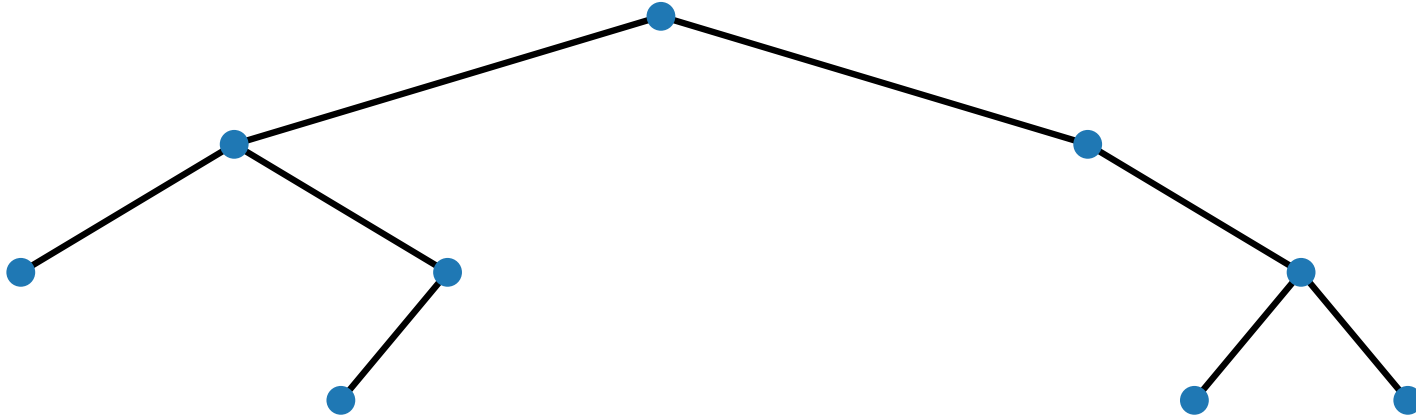
+ directly select  $(j - J)$ -th 1 bit

or select  $J'$ -th subgroup where  $J' = \lfloor (j - J) / (\log \log n)^2 \rfloor$

+ directly select  $(j - J - J')$ -th 1 bit

or select it in the lookup table

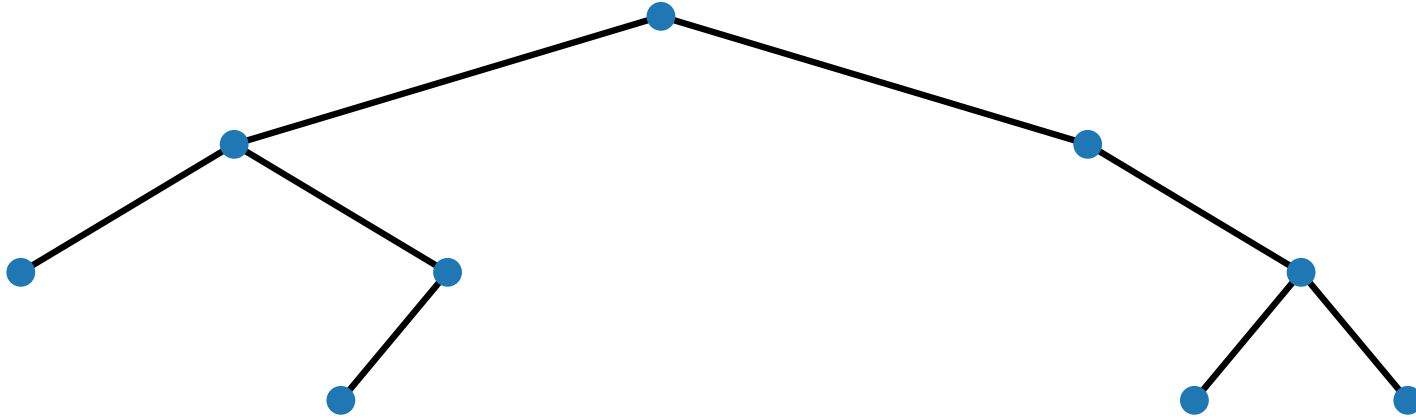
# Succinct Representation of Binary Trees



Number of binary trees on  $n$  vertices:



# Succinct Representation of Binary Trees

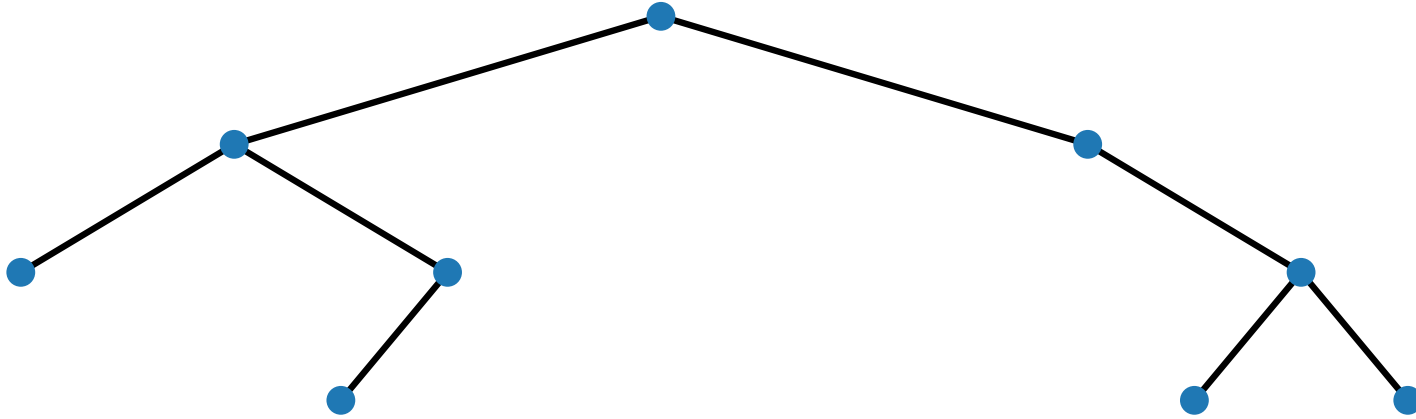


Number of binary trees on  $n$  vertices:

$n = 0$ : "empty tree"

1 possibility

# Succinct Representation of Binary Trees



Number of binary trees on  $n$  vertices:

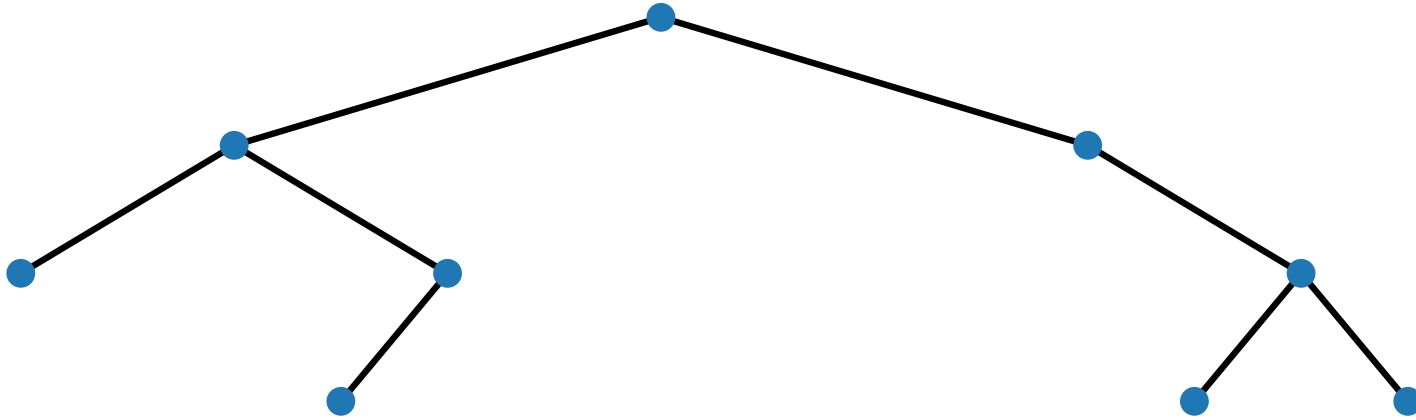
$n = 0$ : "empty tree"

1 possibility

$n = 1$ : ●

1 possibility

# Succinct Representation of Binary Trees



Number of binary trees on  $n$  vertices:

$n = 0$ : "empty tree"

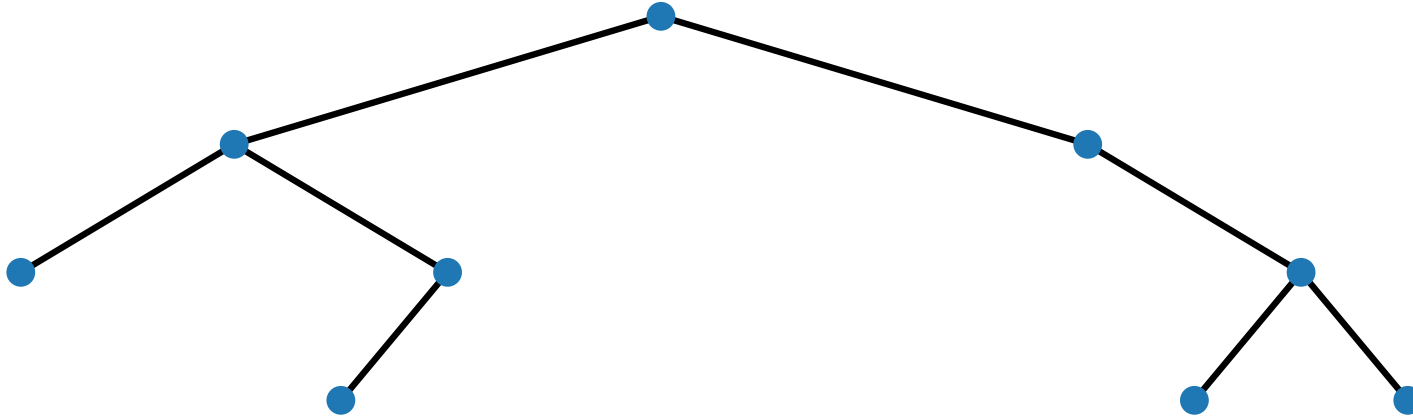
1 possibility

$n = 2$ : ● start with root

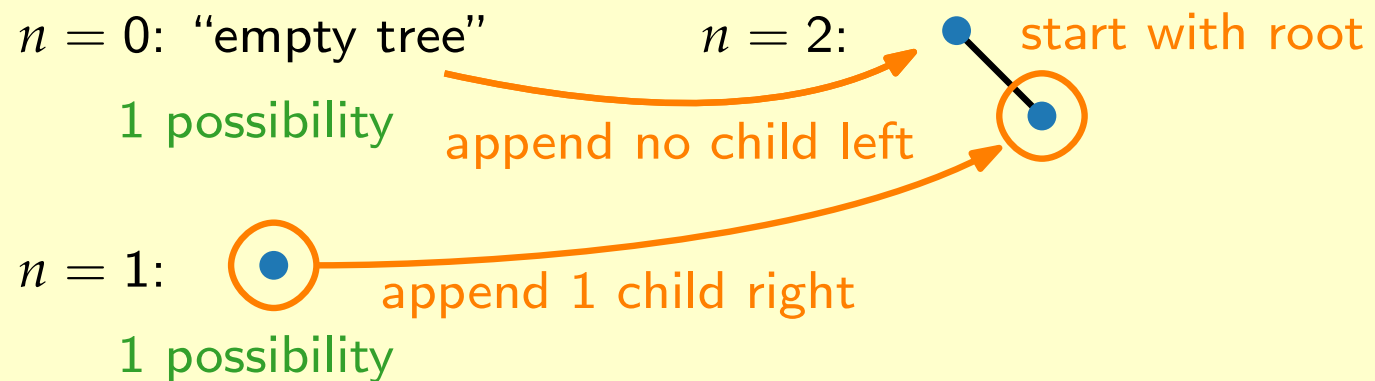
$n = 1$ : ●

1 possibility

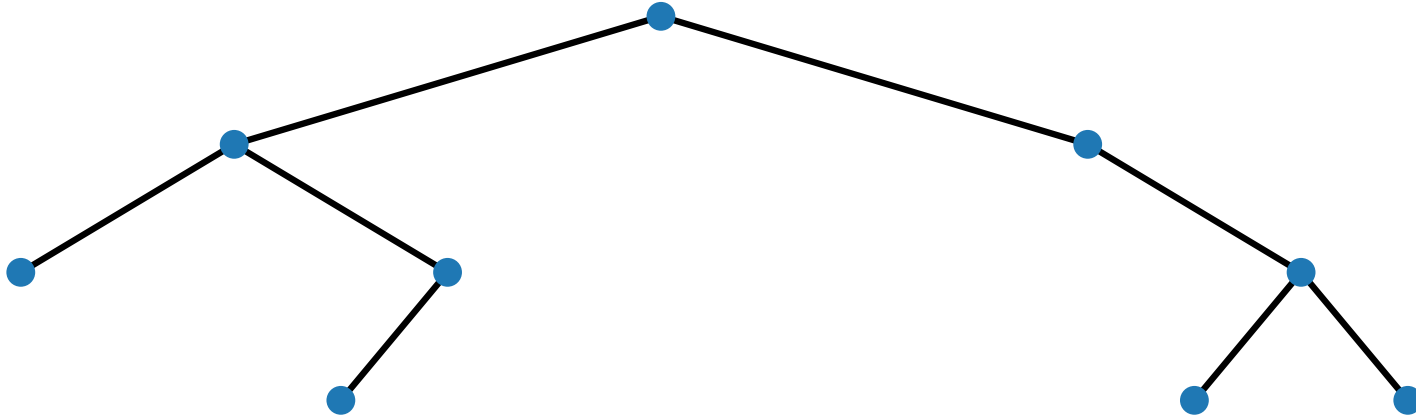
# Succinct Representation of Binary Trees



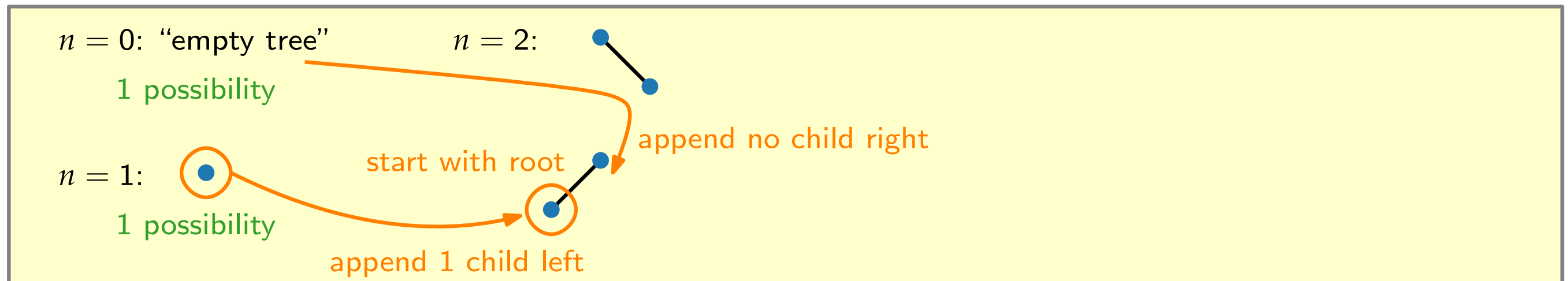
Number of binary trees on  $n$  vertices:



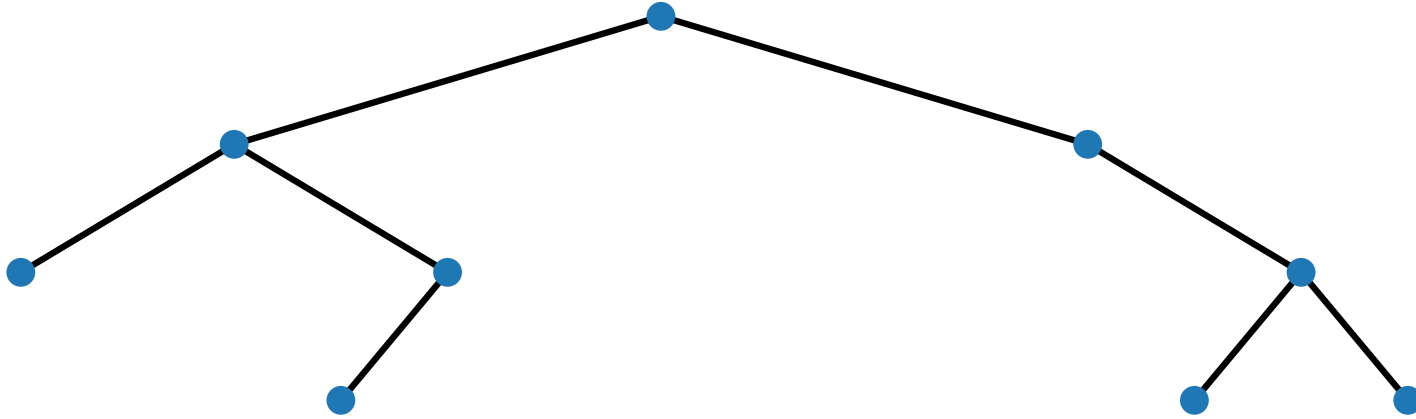
# Succinct Representation of Binary Trees



Number of binary trees on  $n$  vertices:




# Succinct Representation of Binary Trees



Number of binary trees on  $n$  vertices:

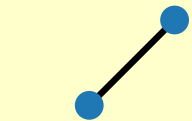
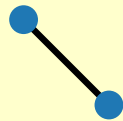
$n = 0$ : "empty tree"

1 possibility

$n = 1$ : 

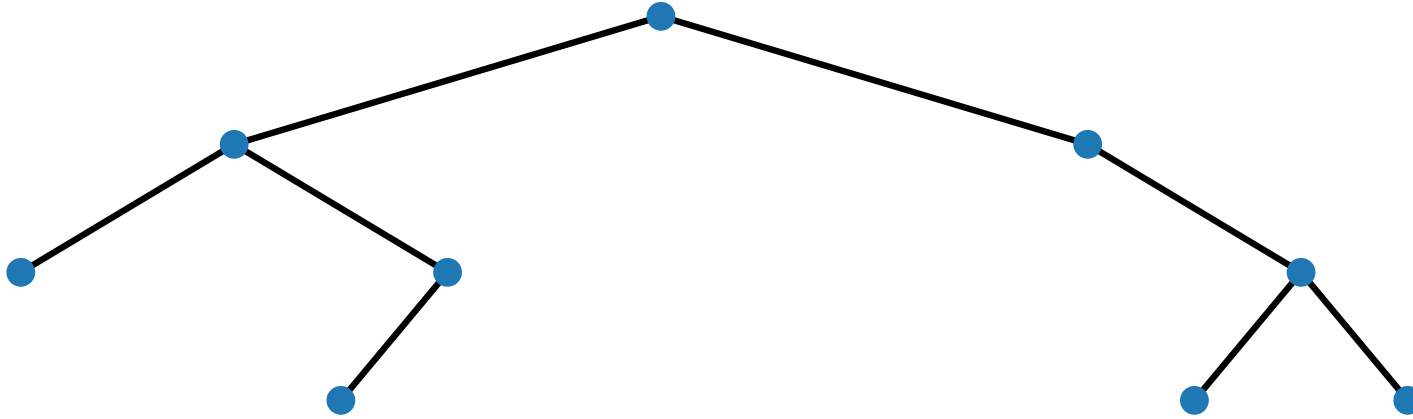
1 possibility

$n = 2$ :



2 possibilities

# Succinct Representation of Binary Trees

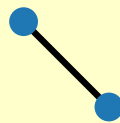


Number of binary trees on  $n$  vertices:

$n = 0$ : "empty tree"

1 possibility

$n = 2$ :



$n = 3$ :

● start with root

$n = 1$ :

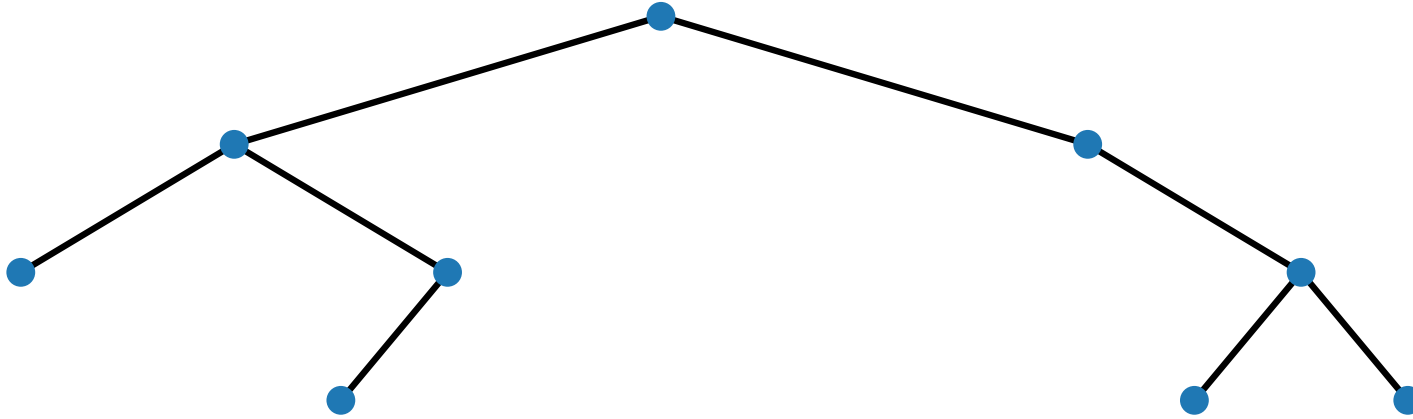


1 possibility



2 possibilities

# Succinct Representation of Binary Trees



Number of binary trees on  $n$  vertices:

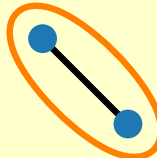
$n = 0$ : "empty tree"

1 possibility

$n = 1$ : •

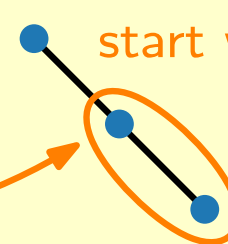
1 possibility

$n = 2$ :



2 possibilities

$n = 3$ :



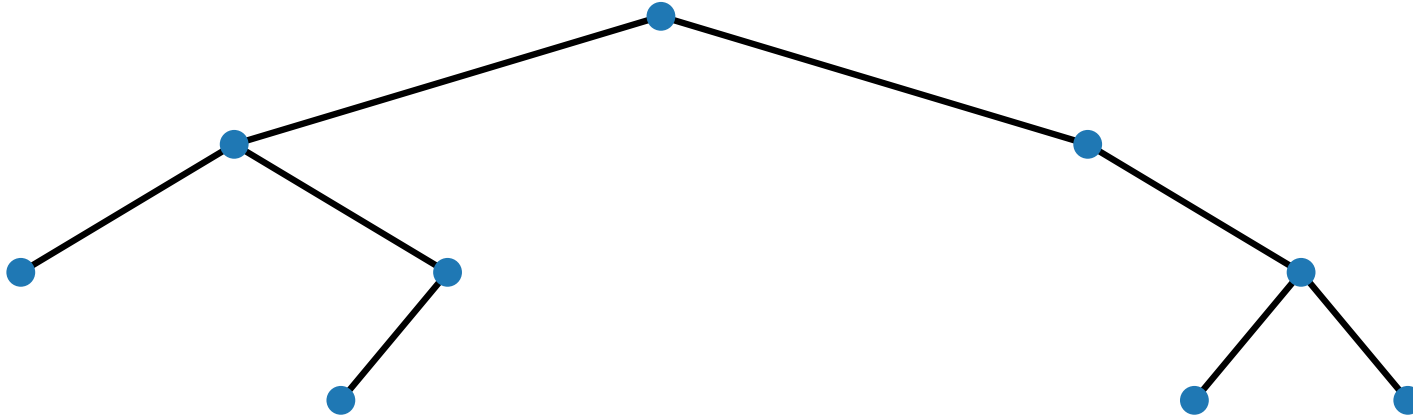
append 2 children right

append no child left

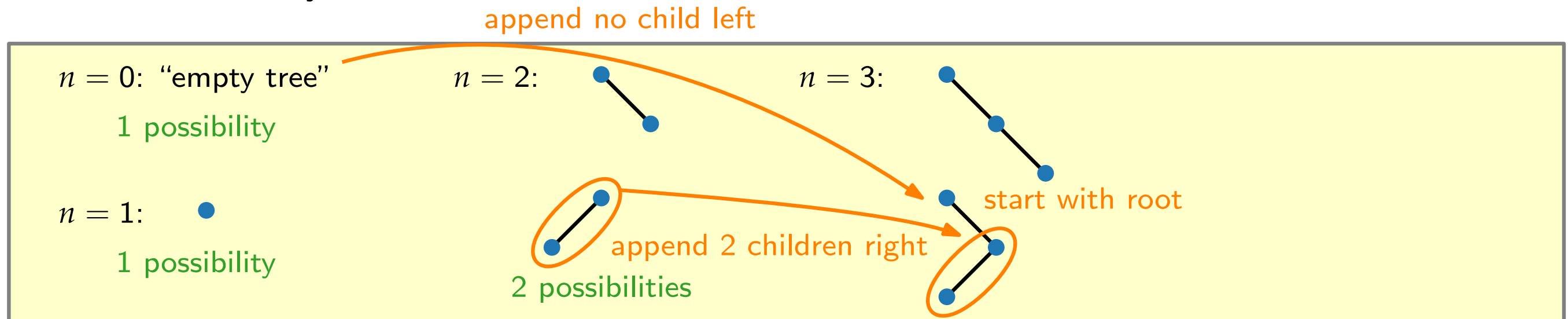
start with root



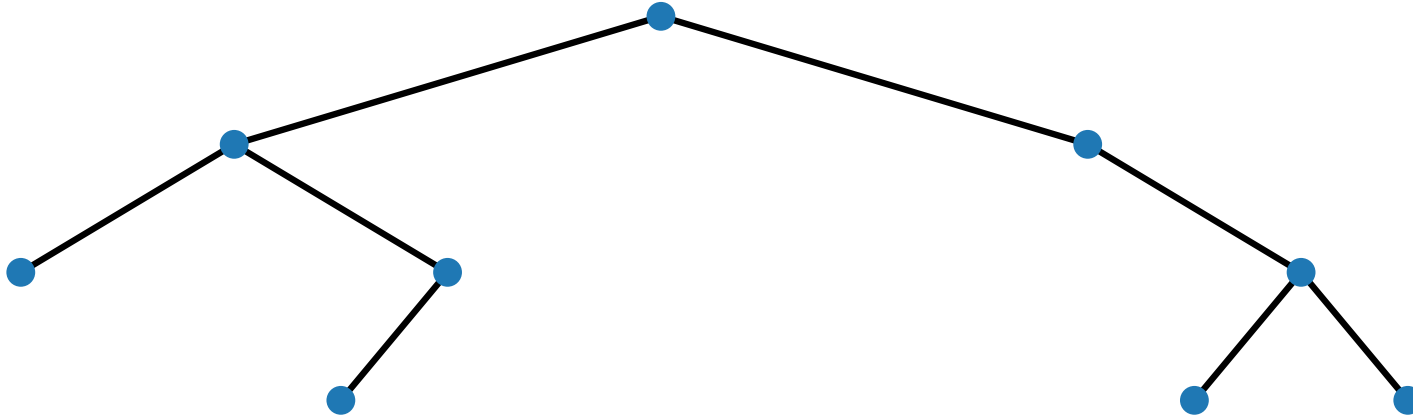
# Succinct Representation of Binary Trees



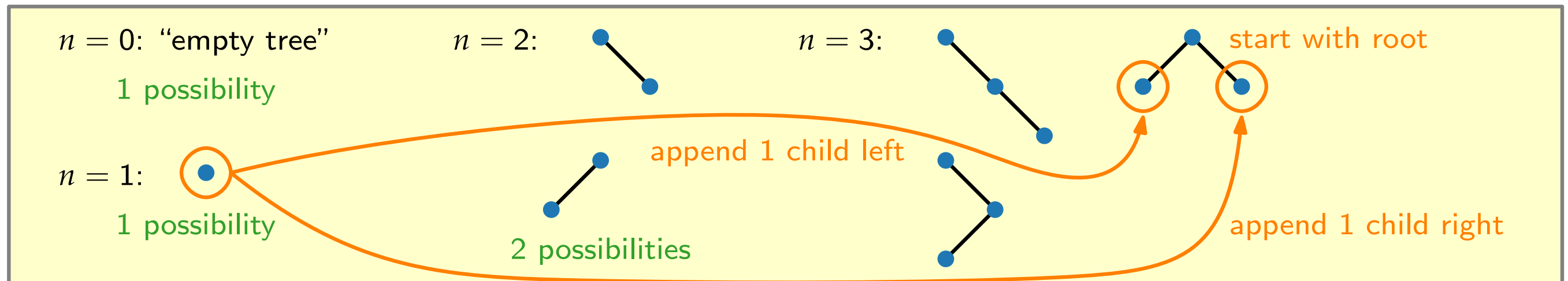
Number of binary trees on  $n$  vertices:



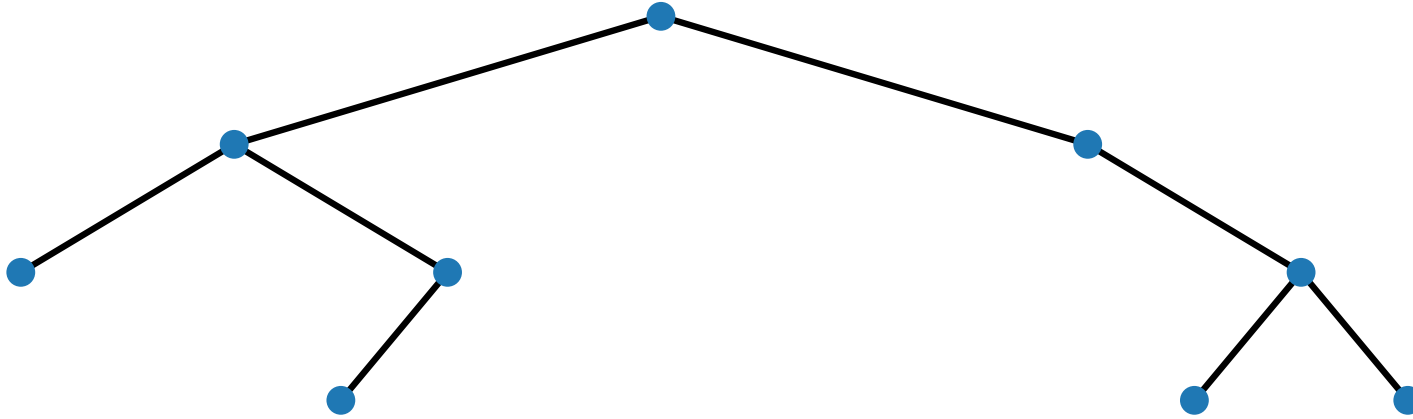
# Succinct Representation of Binary Trees



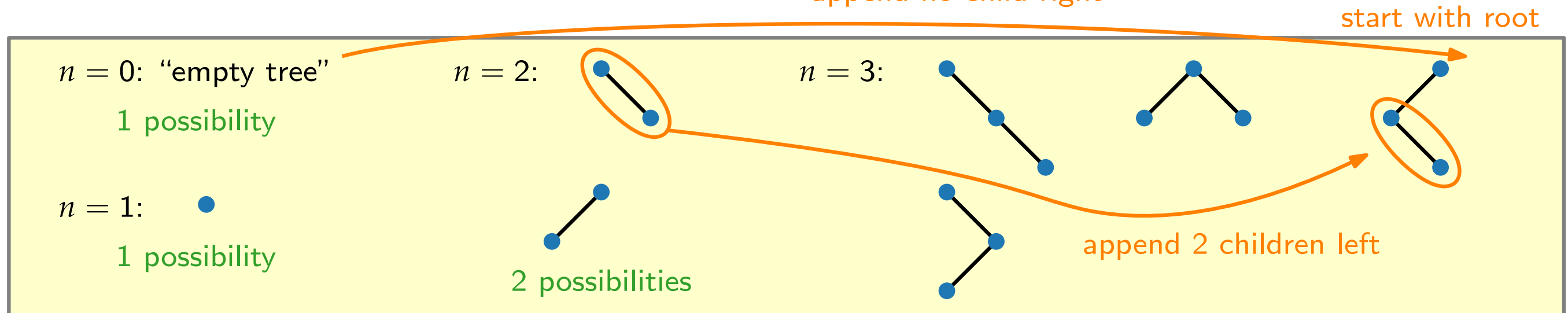
Number of binary trees on  $n$  vertices:



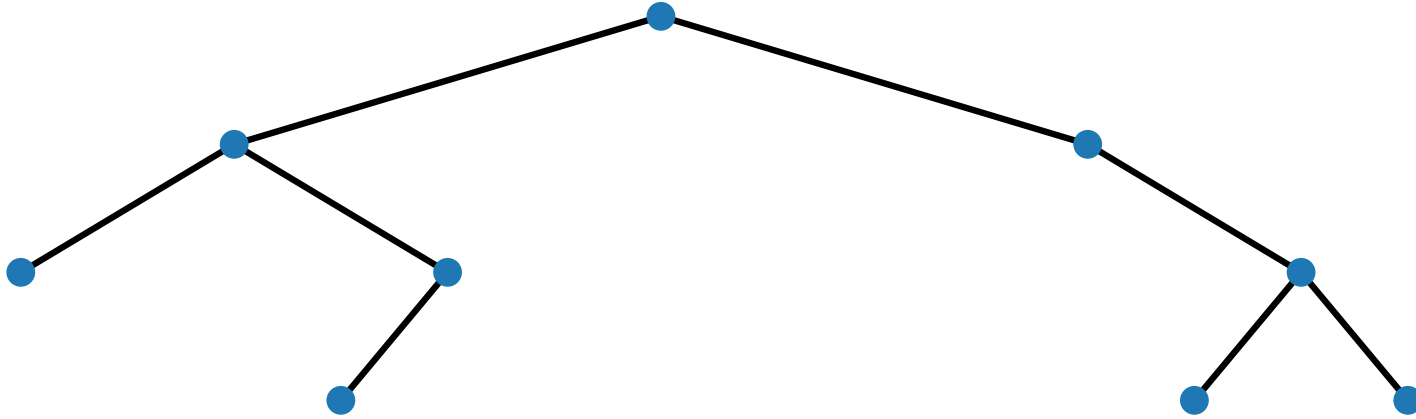
# Succinct Representation of Binary Trees



Number of binary trees on  $n$  vertices:



# Succinct Representation of Binary Trees



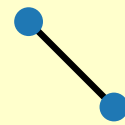
Number of binary trees on  $n$  vertices:

append no child right

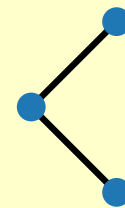
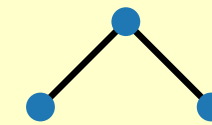
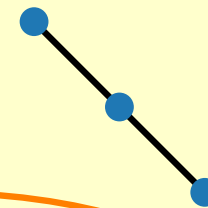
$n = 0$ : "empty tree"

1 possibility

$n = 2$ :



$n = 3$ :

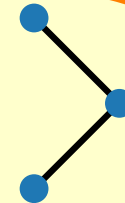


$n = 1$ :



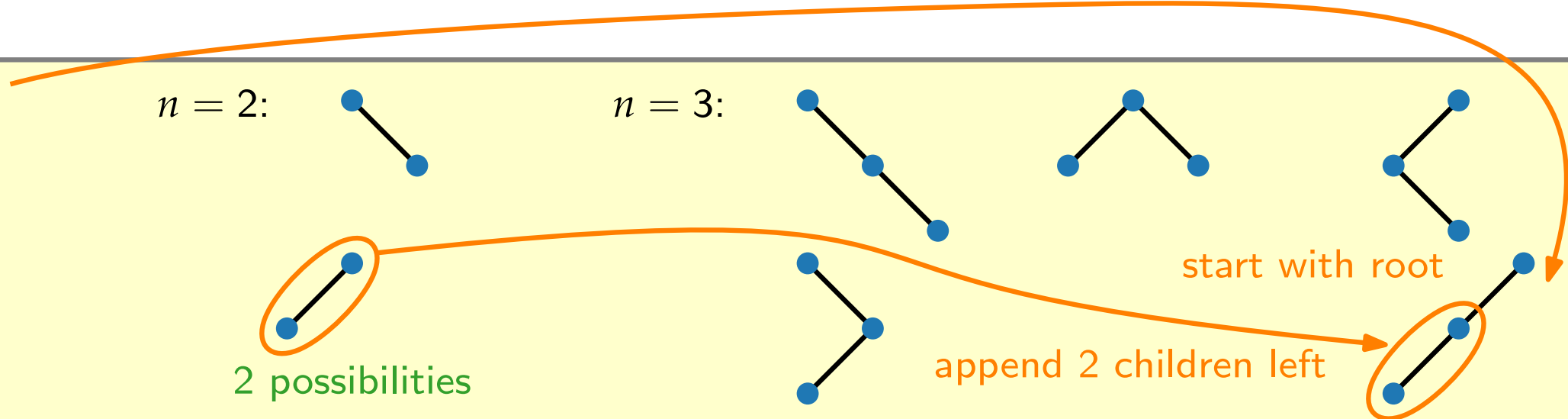
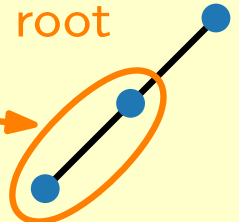
1 possibility

2 possibilities

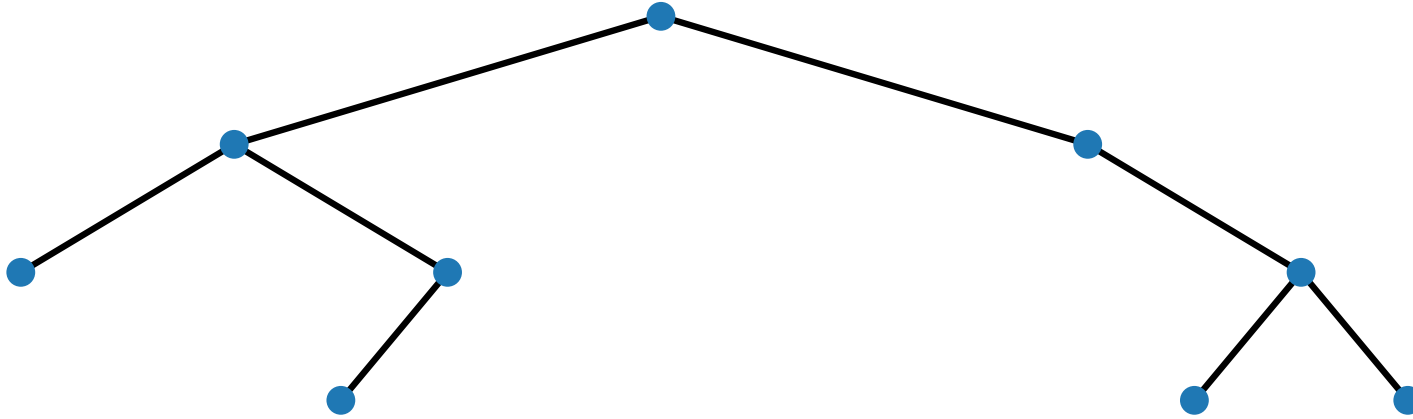


start with root

append 2 children left



# Succinct Representation of Binary Trees

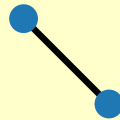


Number of binary trees on  $n$  vertices:

$n = 0$ : "empty tree"

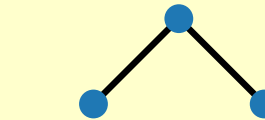
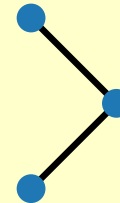
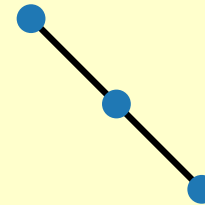
1 possibility

$n = 2$ :

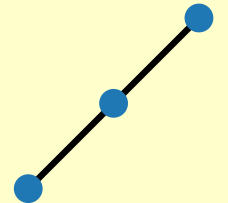
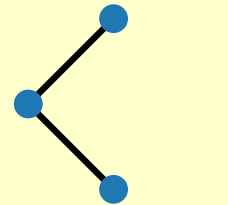


2 possibilities

$n = 3$ :



5 possibilities

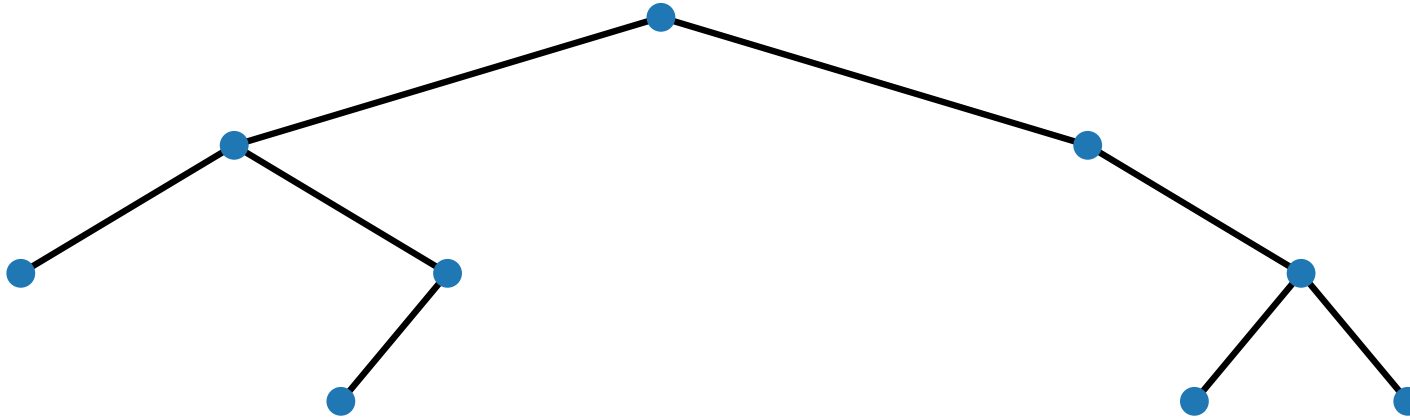


$n = 1$ :



1 possibility

# Succinct Representation of Binary Trees



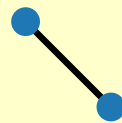
$C_n$  is the  $n$ -th Catalan number and  $C_0 = 1$

Number of binary trees on  $n$  vertices:  $C_n = \sum_{i=0}^{n-1} C_i \cdot C_{n-1-i} = \frac{(2n)!}{(n+1)!n!}$

$n = 0$ : "empty tree"

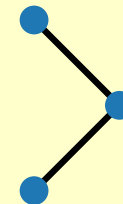
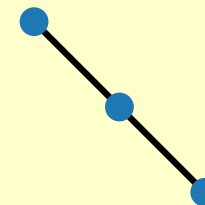
1 possibility

$n = 2$ :

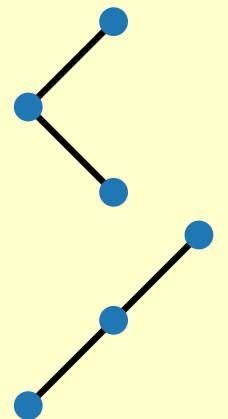


2 possibilities

$n = 3$ :



5 possibilities

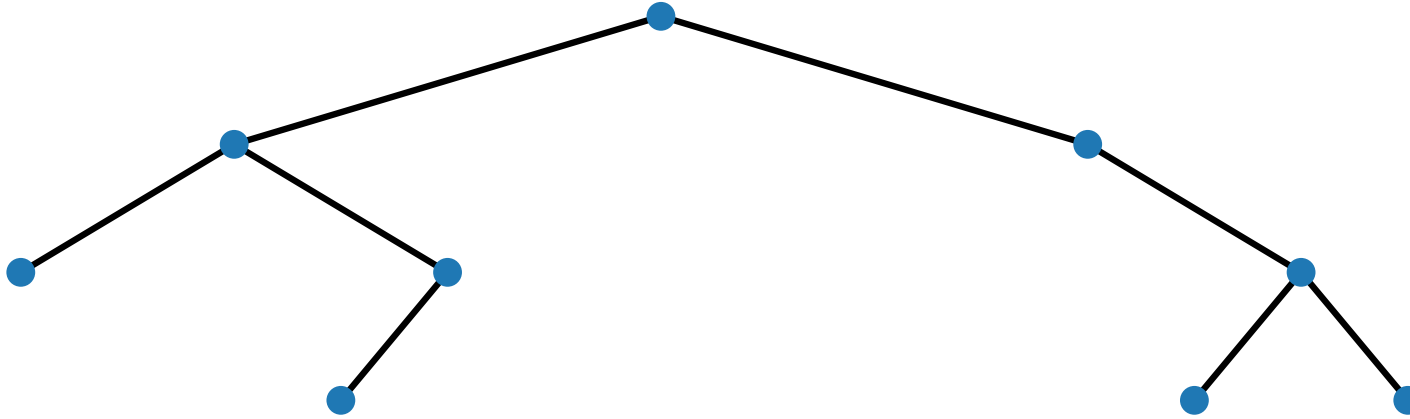


$n = 1$ :



1 possibility

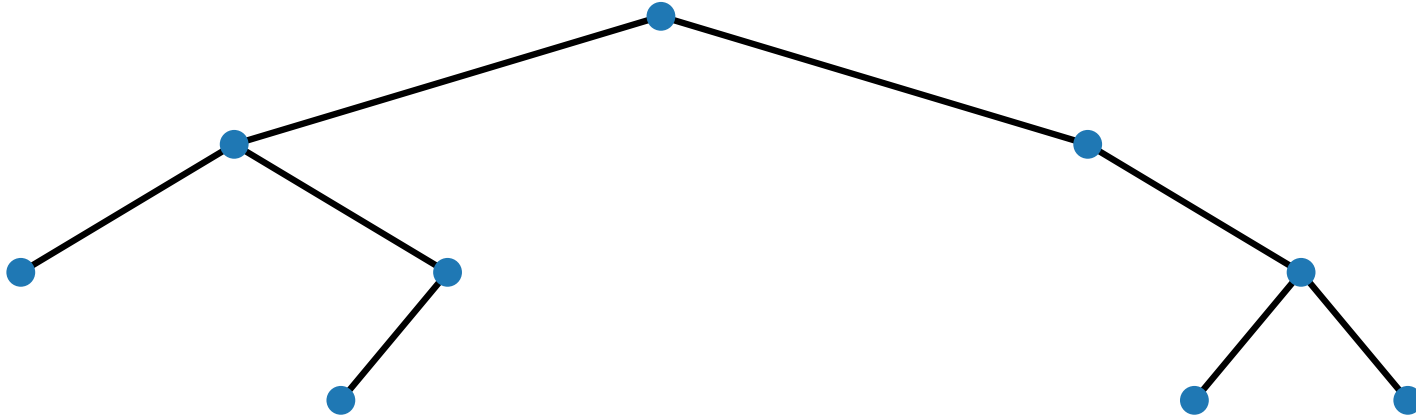
# Succinct Representation of Binary Trees



Number of binary trees on  $n$  vertices:  $C_n = \sum_{i=0}^{n-1} C_i \cdot C_{n-1-i} = \frac{(2n)!}{(n+1)!n!}$

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

# Succinct Representation of Binary Trees



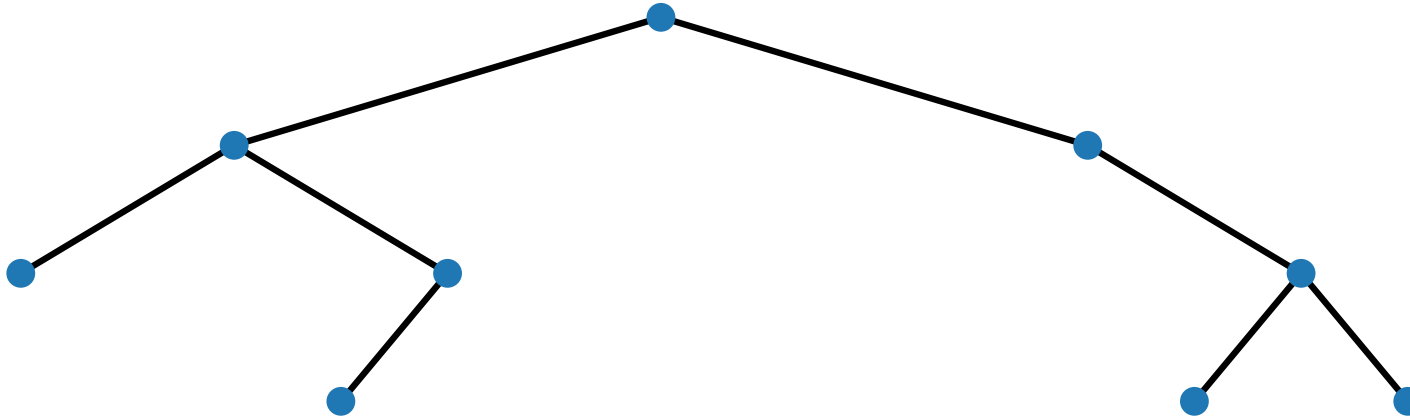
Number of binary trees on  $n$  vertices:  $C_n = \sum_{i=0}^{n-1} C_i \cdot C_{n-1-i} = \frac{(2n)!}{(n+1)!n!}$

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

$\Rightarrow$  We can use  $2n + o(n)$  bits to represent binary trees.



# Succinct Representation of Binary Trees



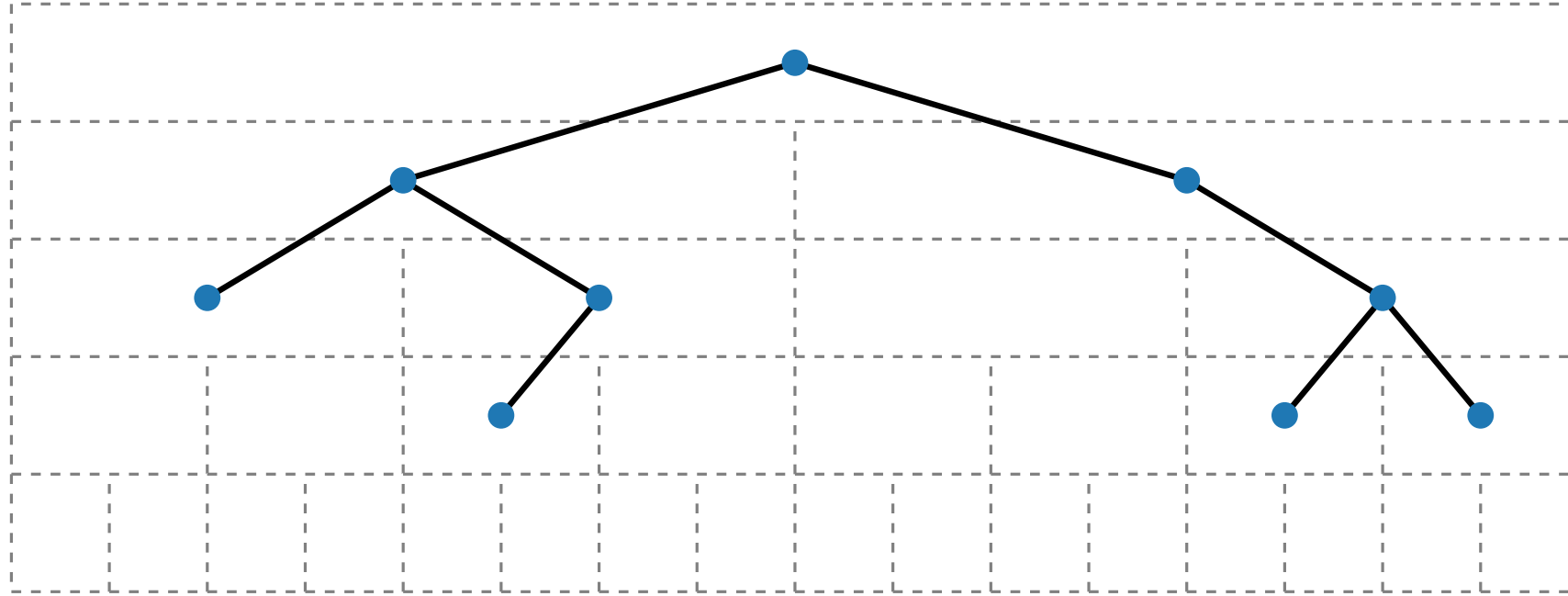
Number of binary trees on  $n$  vertices:  $C_n = \sum_{i=0}^{n-1} C_i \cdot C_{n-1-i} = \frac{(2n)!}{(n+1)!n!}$

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

$\Rightarrow$  We can use  $2n + o(n)$  bits to represent binary trees.

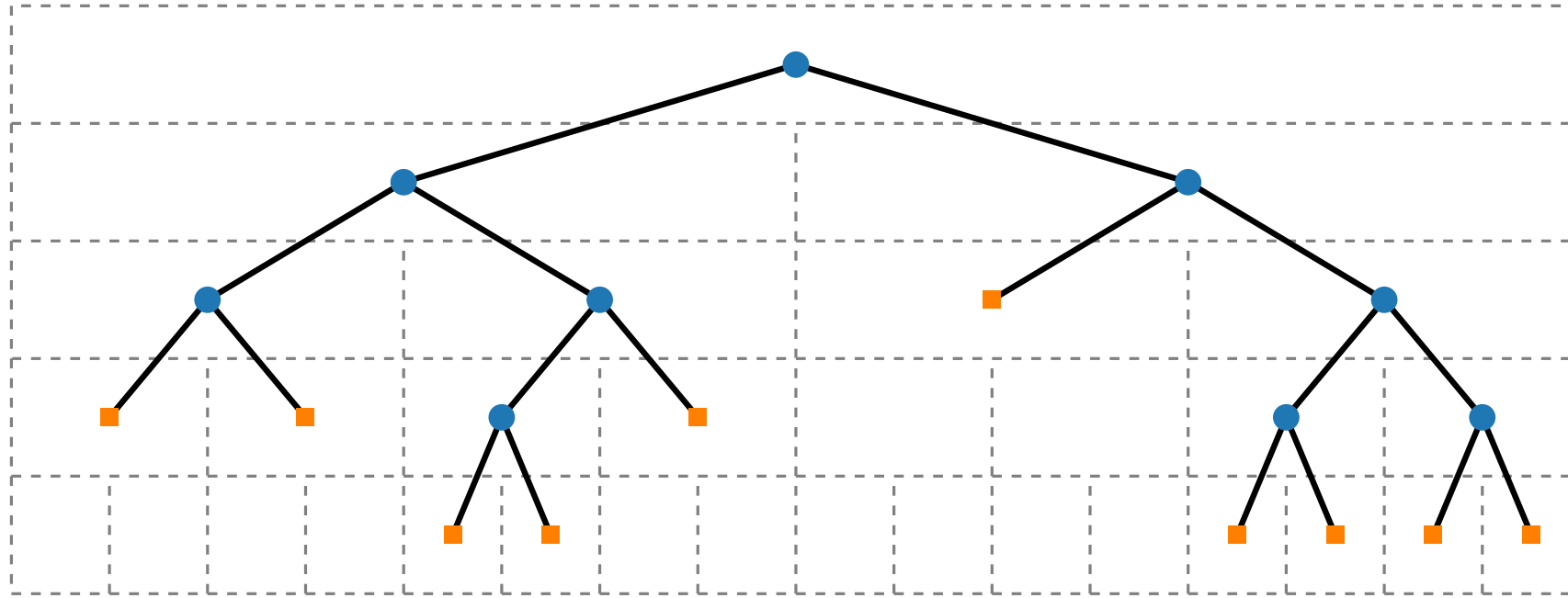
**Difficulty** is when a binary tree is not full.

# Succinct Representation of Binary Trees



Idea.

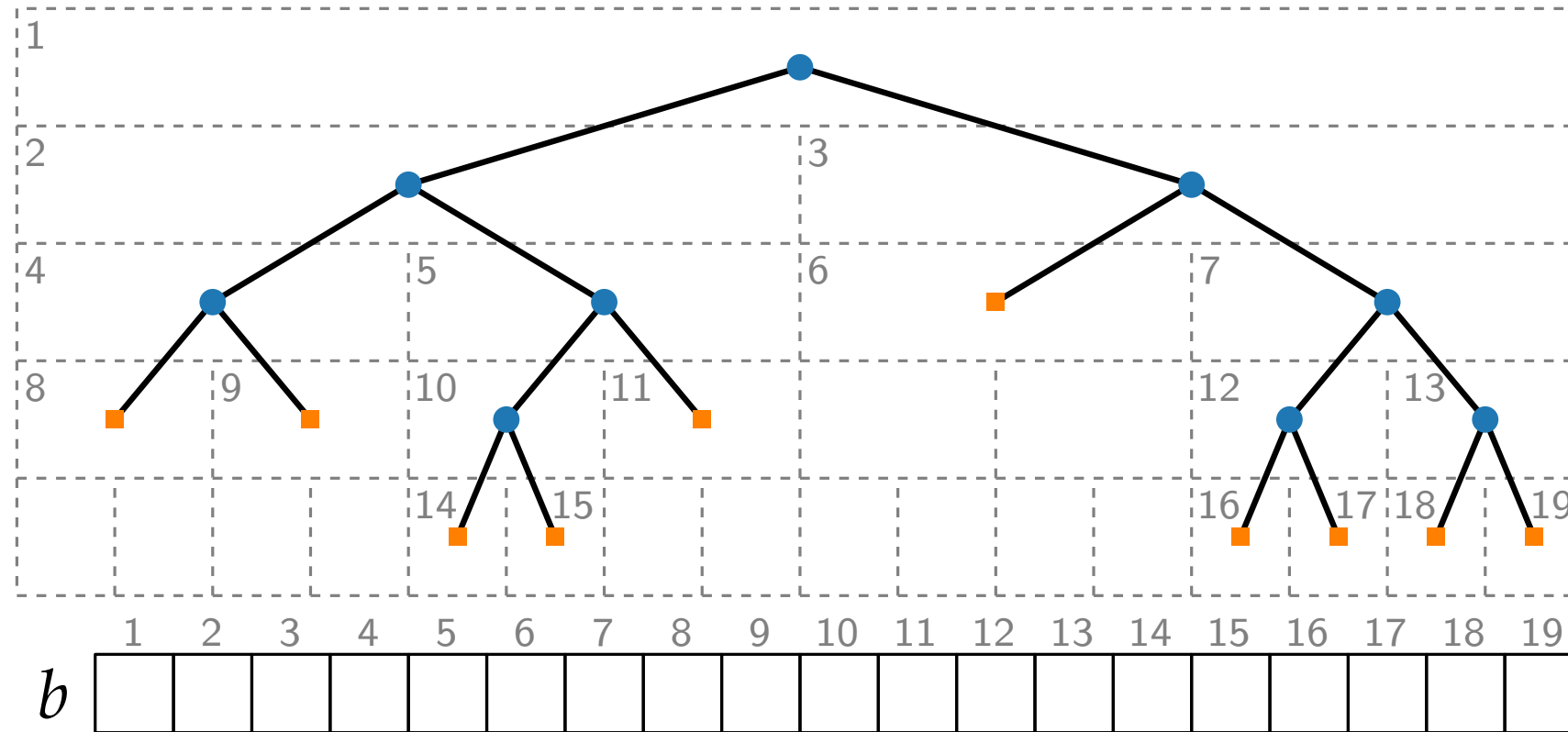
# Succinct Representation of Binary Trees



## Idea.

- Add **external** nodes to have out-degree 2 or 0 at every node

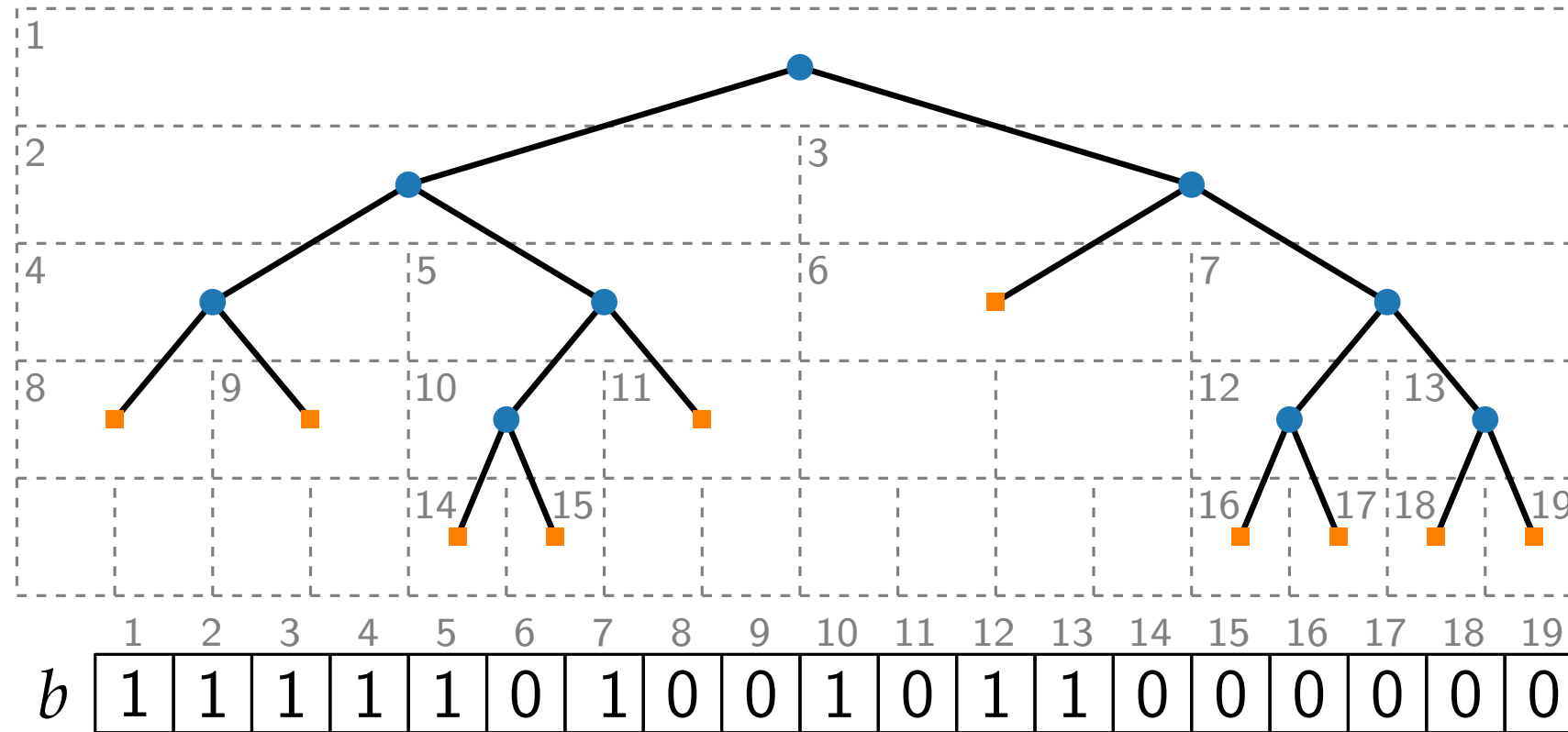
# Succinct Representation of Binary Trees



## Idea.

- Add **external** nodes to have out-degree 2 or 0 at every node

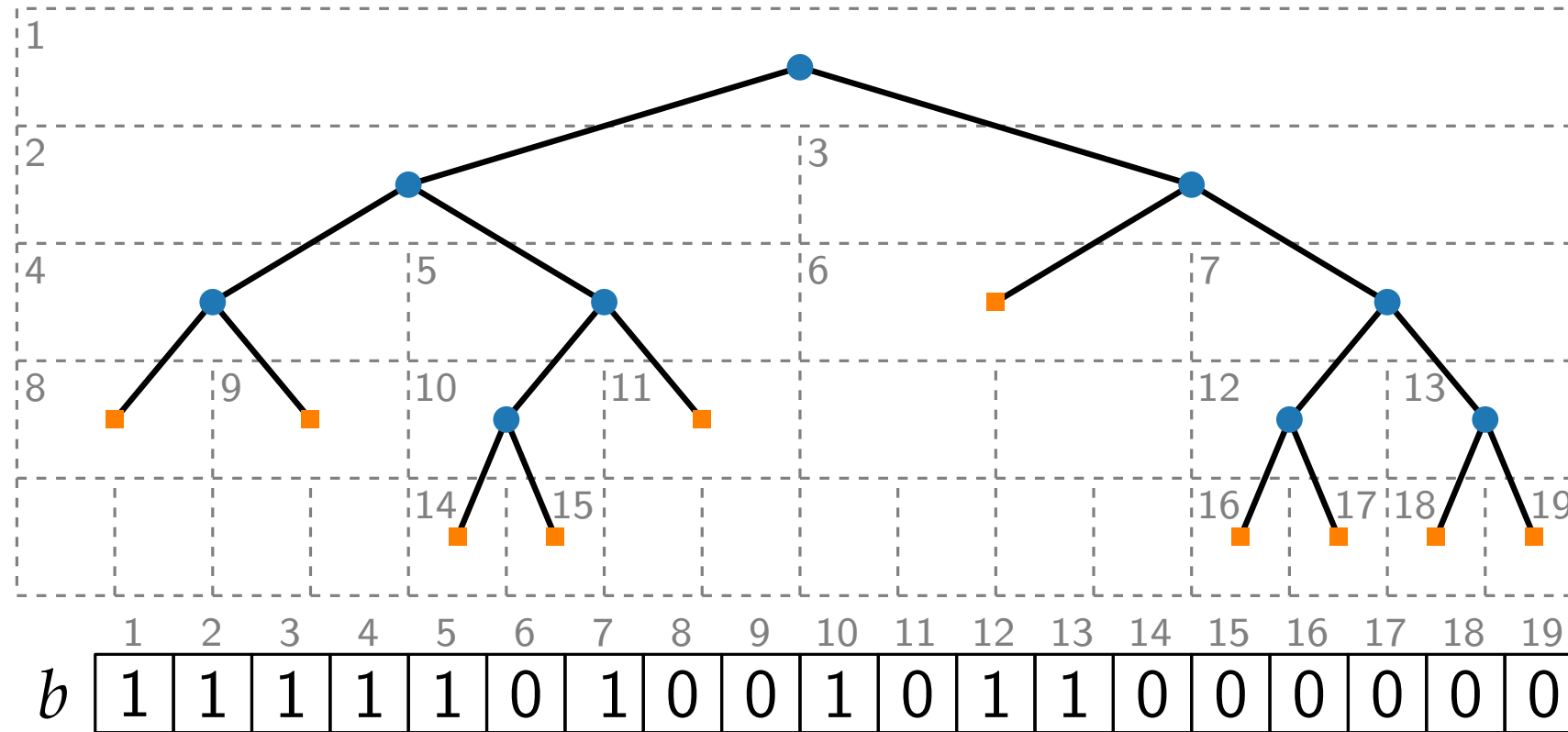
# Succinct Representation of Binary Trees



## Idea.

- Add **external** nodes to have out-degree 2 or 0 at every node
- Read **internal** nodes as 1
- Read **external** nodes as 0

# Succinct Representation of Binary Trees



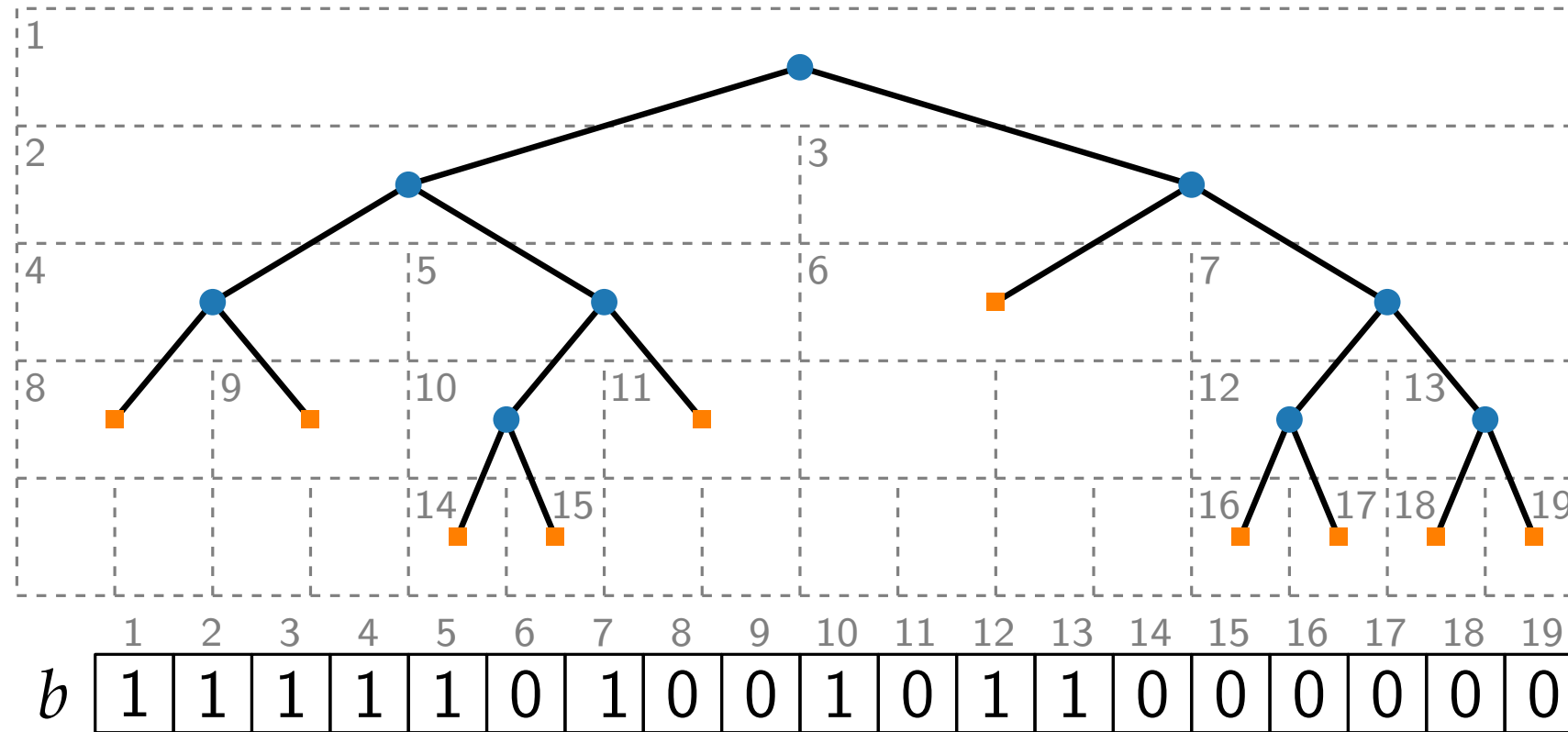
## Size.

- $2n + 1$  bits for  $b$
- $o(n)$  for rank and select

## Idea.

- Add **external** nodes to have out-degree 2 or 0 at every node
- Read **internal** nodes as 1
- Read **external** nodes as 0

# Succinct Representation of Binary Trees



## Size.

- $2n + 1$  bits for  $b$
- $o(n)$  for rank and select

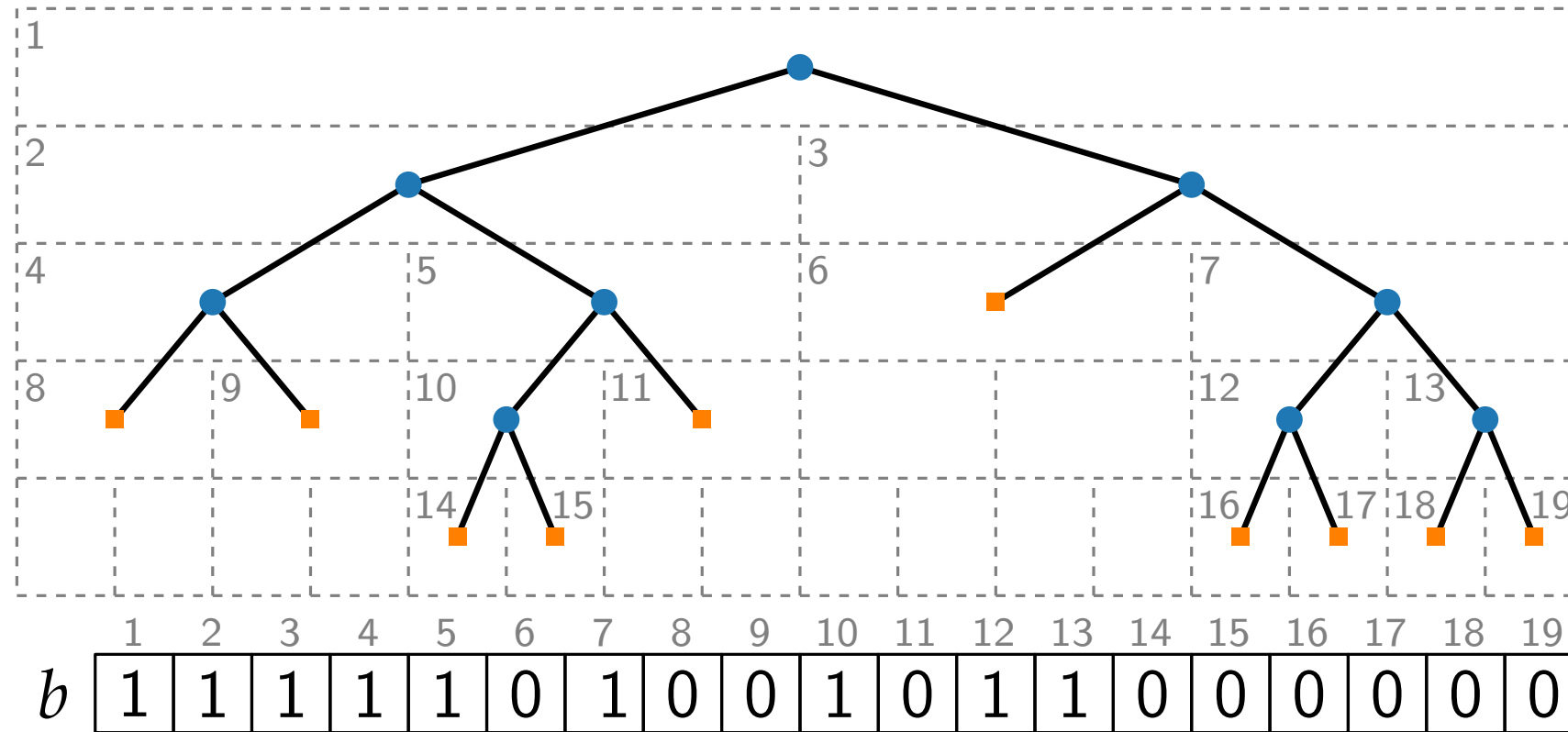
## Idea.

- Add **external** nodes to have out-degree 2 or 0 at every node
- Read **internal** nodes as 1
- Read **external** nodes as 0

## Operations.

- $\text{parent}(i) = ?$
- $\text{leftChild}(i) = ?$
- $\text{rightChild}(i) = ?$

# Succinct Representation of Binary Trees



## Size.

- $2n + 1$  bits for  $b$
- $o(n)$  for rank and select

## Idea.

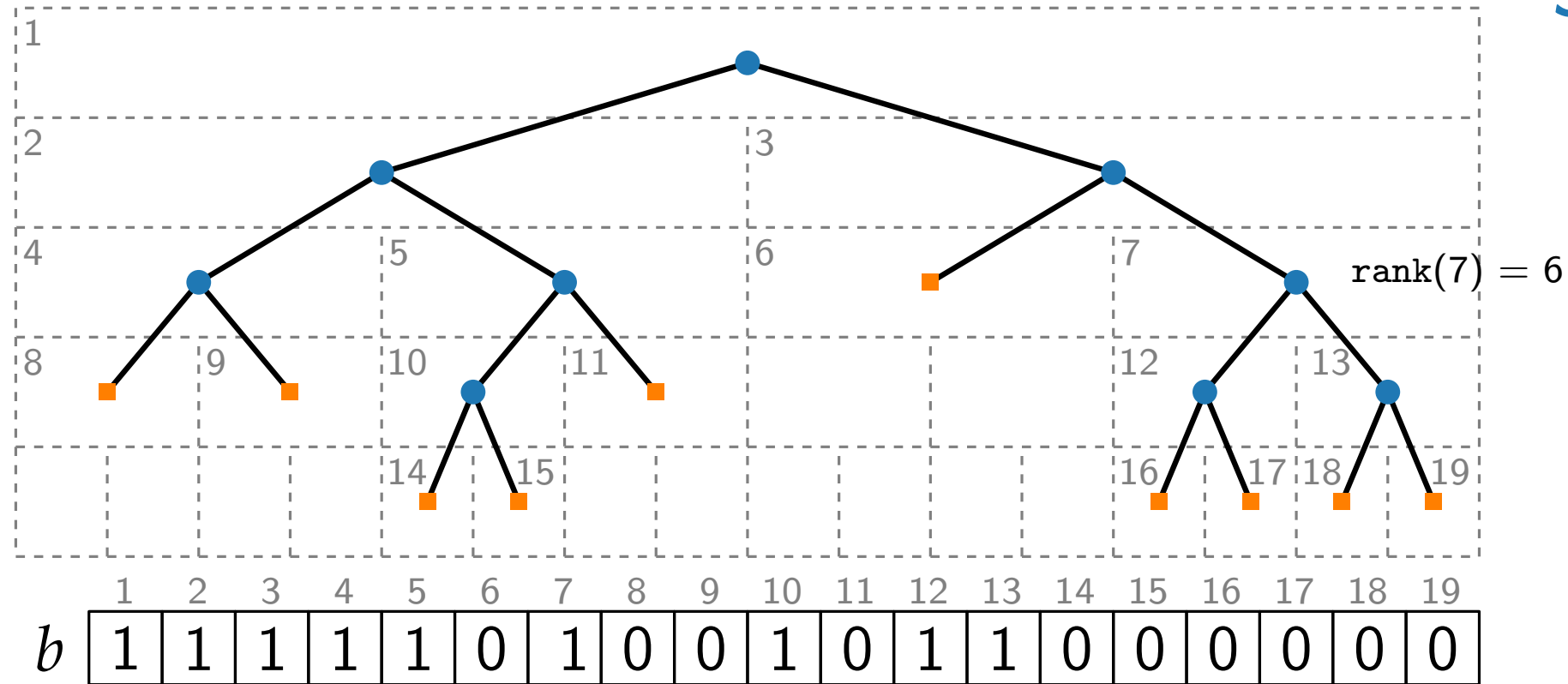
- Add **external** nodes to have out-degree 2 or 0 at every node
- Read **internal** nodes as 1
- Read **external** nodes as 0
- Use rank and select

## Operations.

- $\text{parent}(i) = ?$
- $\text{leftChild}(i) = ?$
- $\text{rightChild}(i) = ?$



# Succinct Representation of Binary Trees



## Size.

- $2n + 1$  bits for  $b$
- $o(n)$  for rank and select

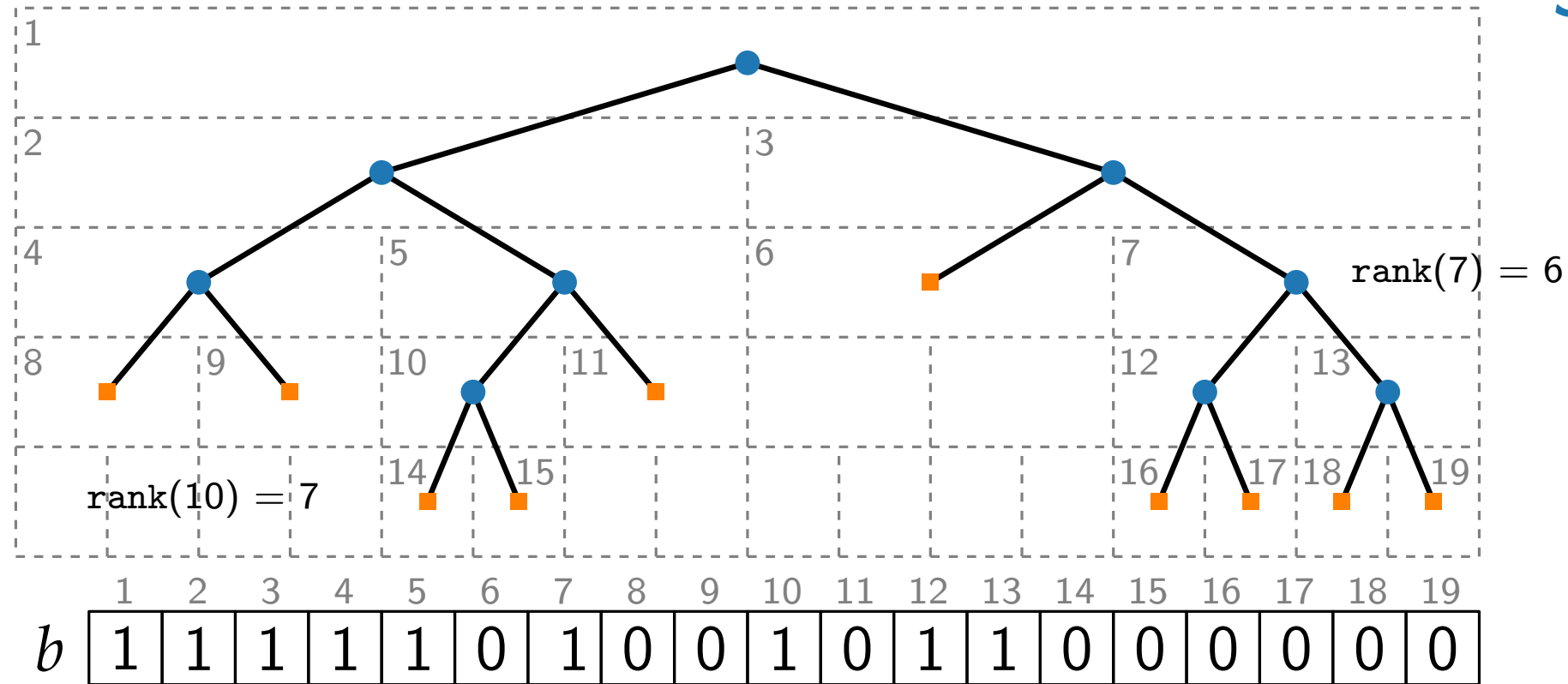
## Idea.

- Add **external** nodes to have out-degree 2 or 0 at every node
- Read **internal** nodes as 1
- Read **external** nodes as 0
- Use rank and select

## Operations.

- $\text{parent}(i) = ?$
- $\text{leftChild}(i) = ?$
- $\text{rightChild}(i) = ?$

# Succinct Representation of Binary Trees



## Size.

- $2n + 1$  bits for  $b$
- $o(n)$  for rank and select

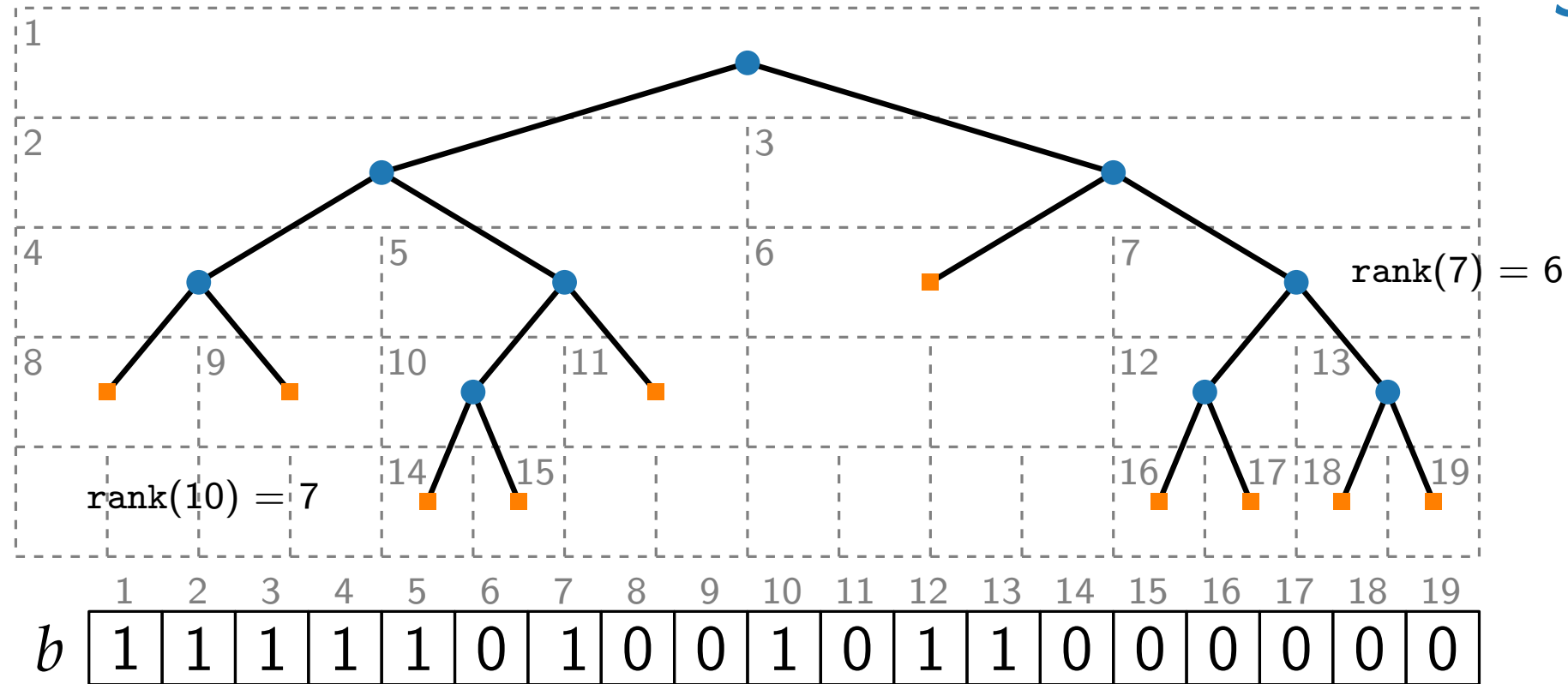
## Idea.

- Add **external** nodes to have out-degree 2 or 0 at every node
- Read **internal** nodes as 1
- Read **external** nodes as 0
- Use rank and select

## Operations.

- $\text{parent}(i) = ?$
- $\text{leftChild}(i) = ?$
- $\text{rightChild}(i) = ?$

# Succinct Representation of Binary Trees



## Size.

- $2n + 1$  bits for  $b$
- $o(n)$  for rank and select

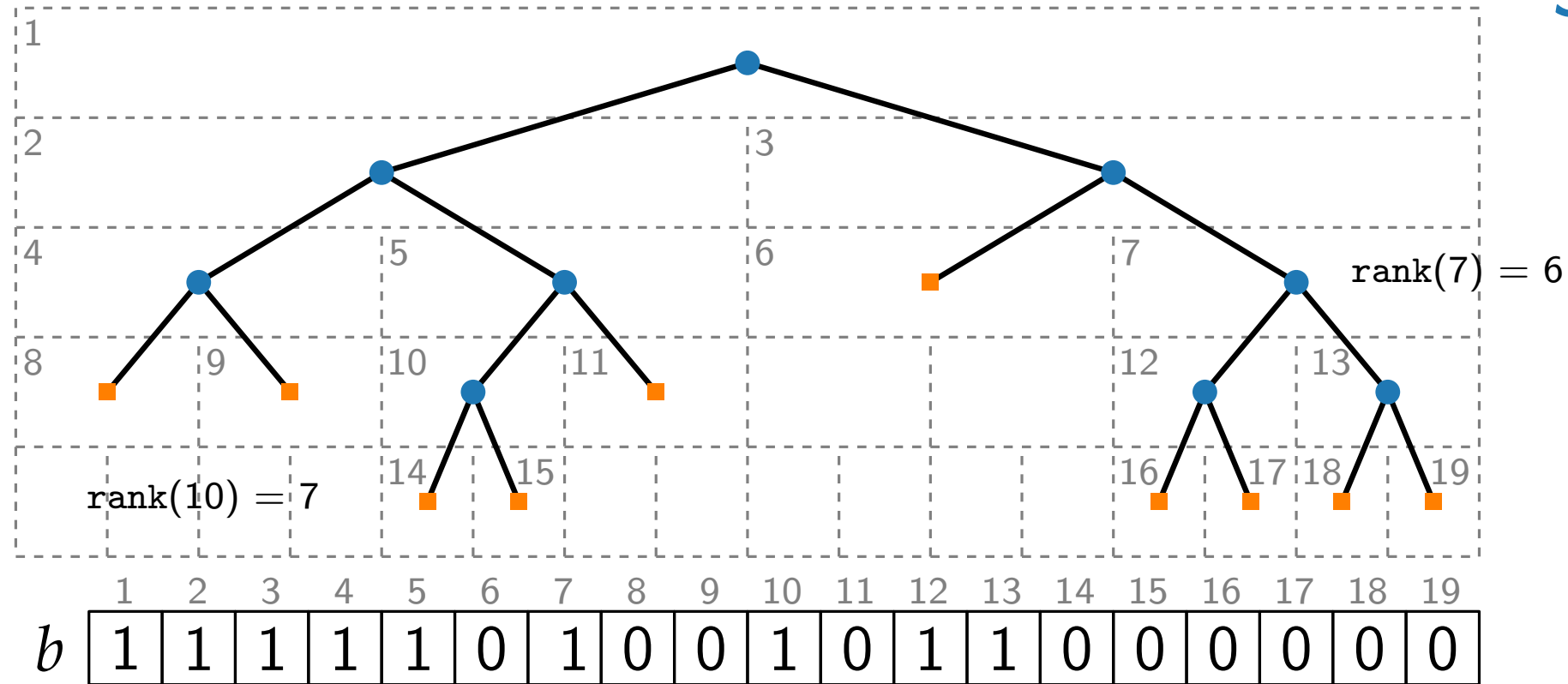
## Idea.

- Add **external** nodes to have out-degree 2 or 0 at every node
- Read **internal** nodes as 1
- Read **external** nodes as 0
- Use rank and select

## Operations.

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

# Succinct Representation of Binary Trees



## Size.

- $2n + 1$  bits for  $b$
- $o(n)$  for rank and select

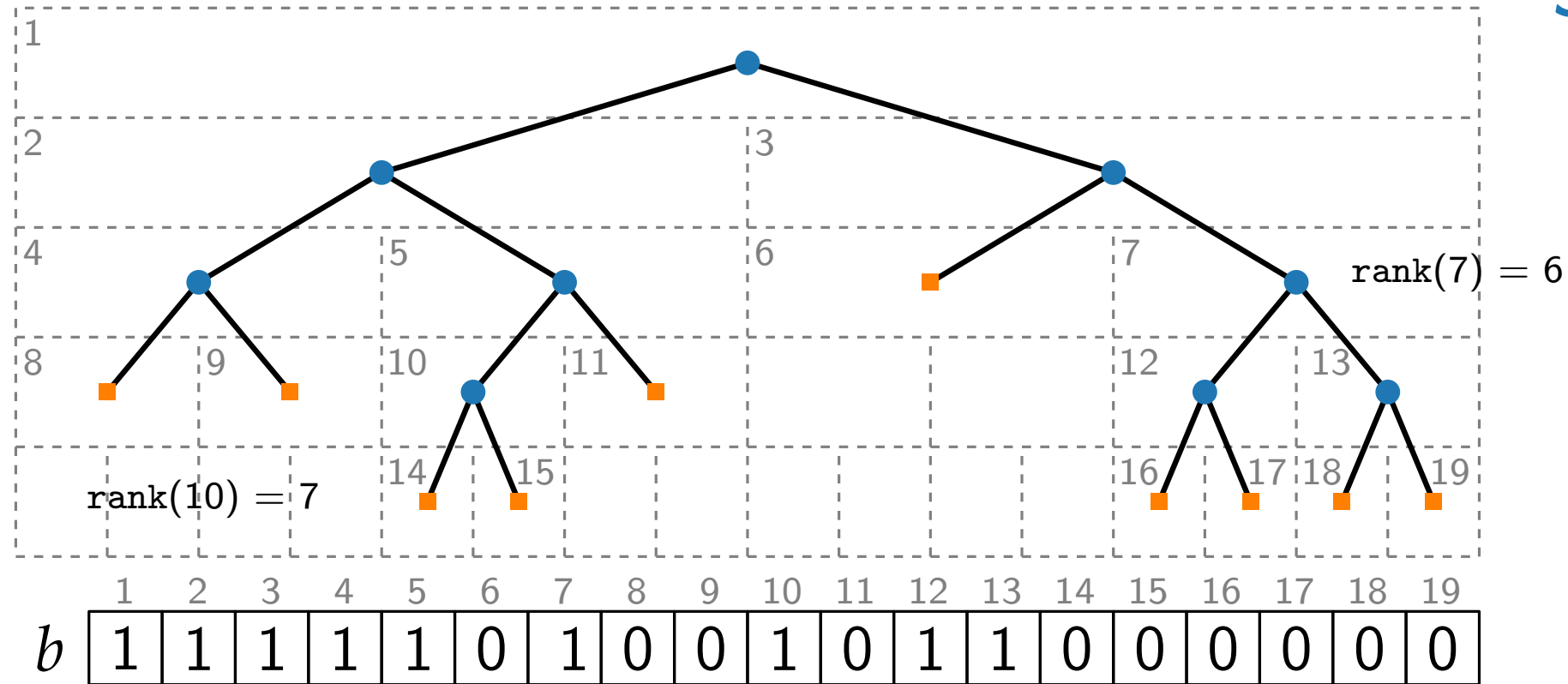
## Idea.

- Add **external** nodes to have out-degree 2 or 0 at every node
- Read **internal** nodes as 1
- Read **external** nodes as 0
- Use rank and select

## Operations.

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

# Succinct Representation of Binary Trees



## Size.

- $2n + 1$  bits for  $b$
- $o(n)$  for rank and select

## Idea.

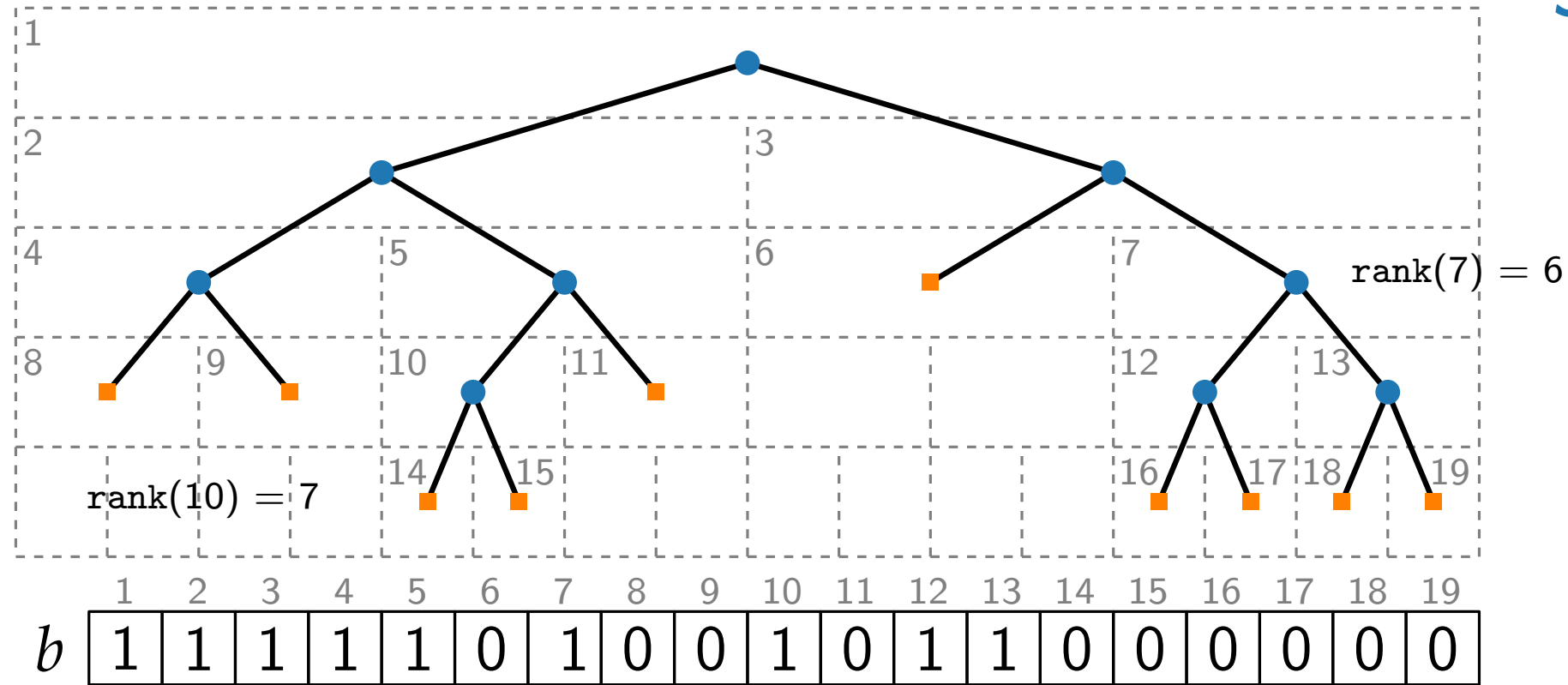
- Add **external** nodes to have out-degree 2 or 0 at every node
- Read **internal** nodes as 1
- Read **external** nodes as 0
- Use rank and select

## Operations.

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

*Proof is exercise.*

# Succinct Representation of Binary Trees



## Size.

- $2n + 1$  bits for  $b$
- $o(n)$  for rank and select

## Idea.

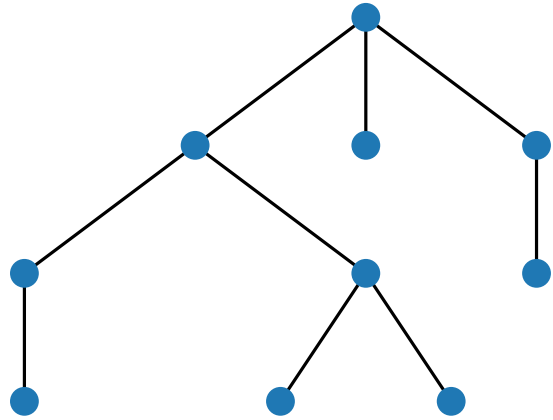
- Add **external** nodes to have out-degree 2 or 0 at every node
- Read **internal** nodes as 1
- Read **external** nodes as 0
- Use rank and select

## Operations.

- $\text{parent}(i) = \text{select}(\lfloor \frac{i}{2} \rfloor)$
- $\text{leftChild}(i) = 2 \text{rank}(i)$
- $\text{rightChild}(i) = 2 \text{rank}(i) + 1$
- $\text{rank}(i)$  is index for array storing actual values

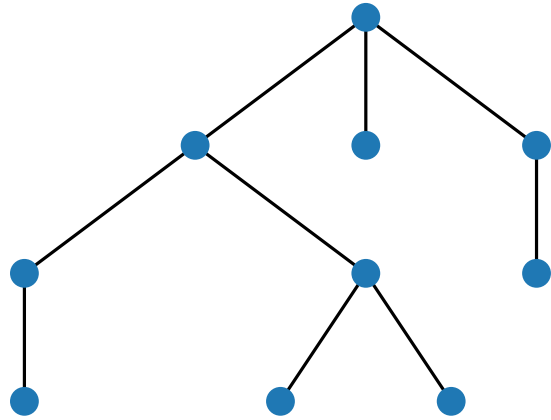
*Proof is exercise.*

# Succinct Representation of Trees - LOUDS



# Succinct Representation of Trees - LOUDS

[Level Order Unary Degree Sequence]

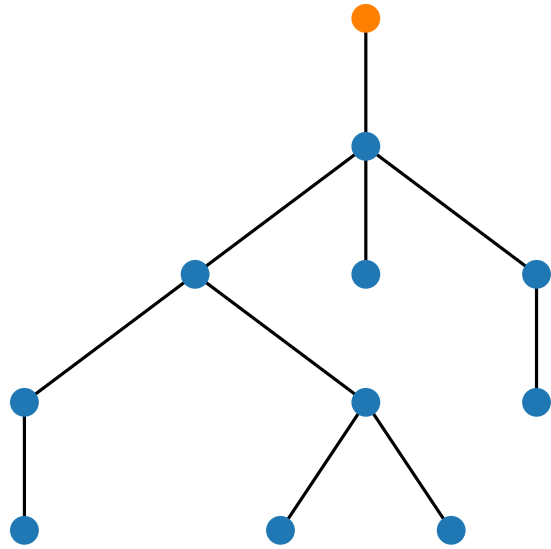




# Succinct Representation of Trees - LOUDS

[Level Order Unary Degree Sequence]

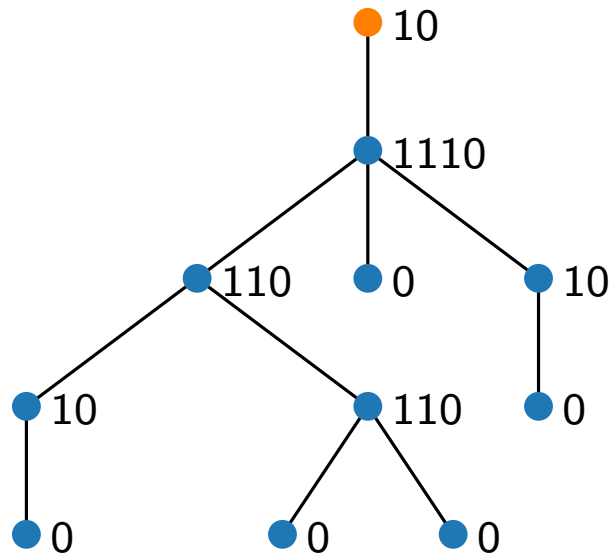
- add extra root with out-degree 1



# Succinct Representation of Trees - LOUDS

[Level Order Unary Degree Sequence]

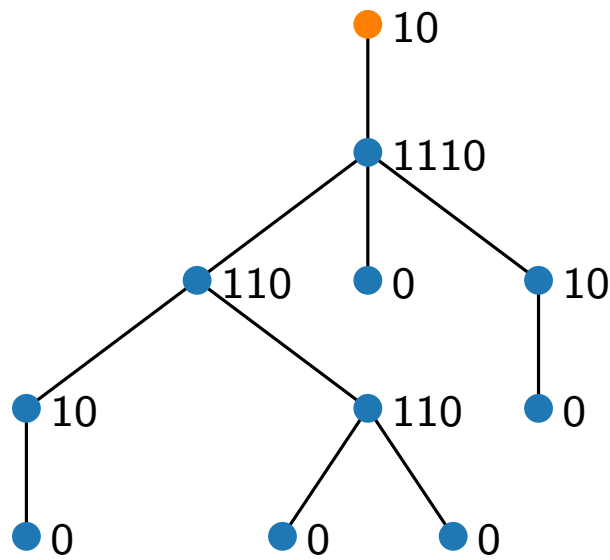
- add extra root with out-degree 1
- unary encoding of out-degree terminated by a 0



# Succinct Representation of Trees - LOUDS

[Level Order Unary Degree Sequence]

- add extra root with out-degree 1
- unary encoding of out-degree terminated by a 0
- gives LOUDS sequence

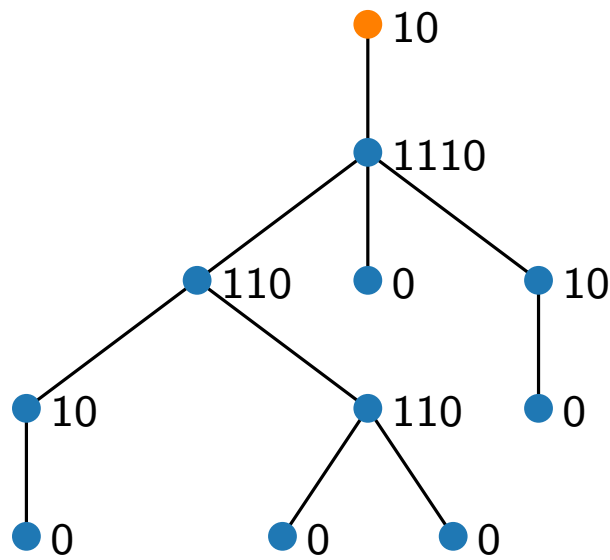


1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	0	1	1	1	0	1	1	0	0	1	0	1	0	1	1	0	0	0	0	0

# Succinct Representation of Trees - LOUDS

[Level Order Unary Degree Sequence]

- add extra root with out-degree 1
- unary encoding of out-degree terminated by a 0
- 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

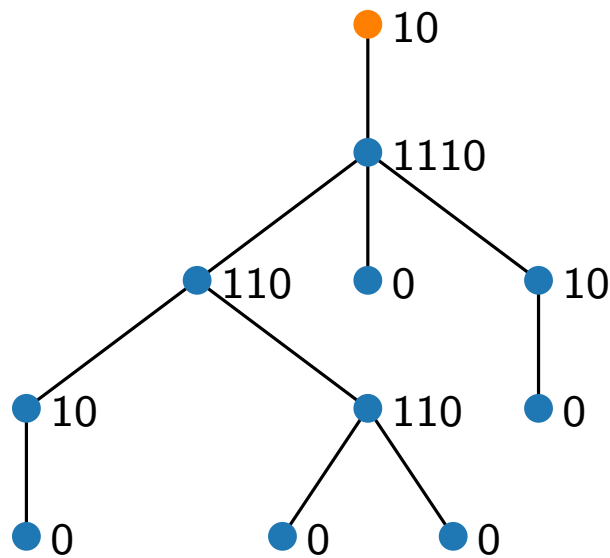
## Size.

- each vertex (except root) is represented twice, namely with a 1 and with a 0
- $o(n)$  bits for rank and select

# Succinct Representation of Trees - LOUDS

[Level Order Unary Degree Sequence]

- add extra root with out-degree 1
- unary encoding of out-degree terminated by a 0
- 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

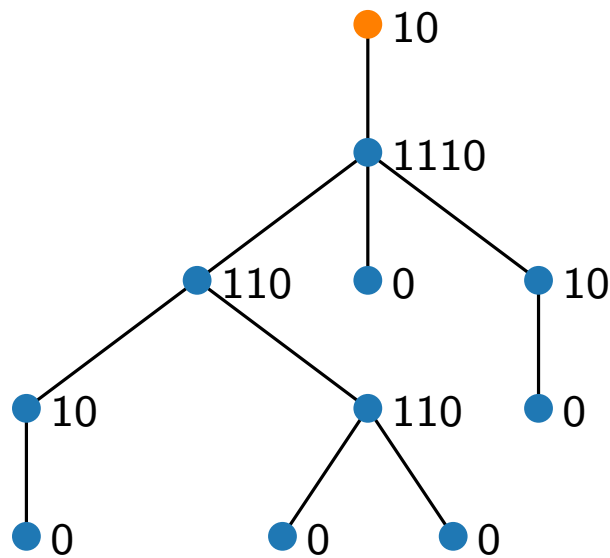
## Size.

- each vertex (except root) is represented twice, namely with a 1 and with a 0  $\Rightarrow 2n + o(n)$  bits
- $o(n)$  bits for rank and select

# Succinct Representation of Trees - LOUDS

[Level Order Unary Degree Sequence]

- add extra root with out-degree 1
- unary encoding of out-degree terminated by a 0
- 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

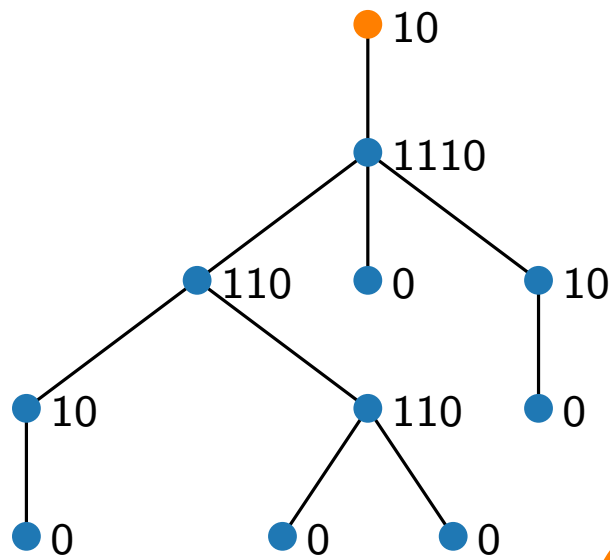
## Operations.

- Let  $i$  be index of 1 in LOUDS sequence. This 1 represents a node (e.g. first 1 represents the root).
- $\text{rank}(i)$  is index for array storing actual values of the nodes.

# Succinct Representation of Trees - LOUDS

[Level Order Unary Degree Sequence]

- add extra root with out-degree 1
- unary encoding of out-degree terminated by a 0
- 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

execute  $\text{select}(j)$  on  
the 0s instead of the 1s

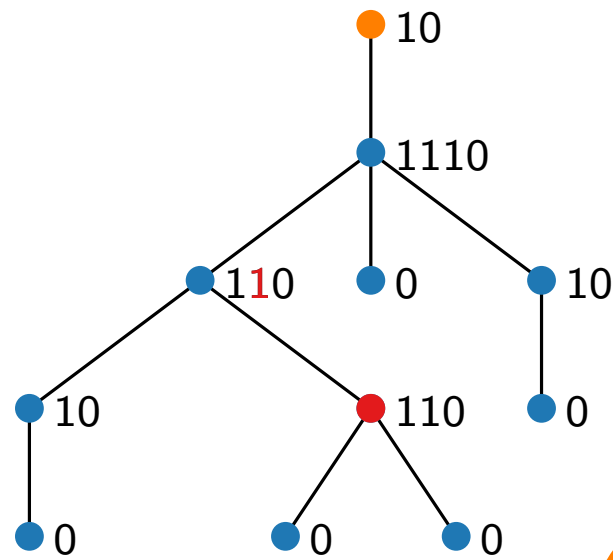
execute  $\text{rank}(i)$  on  
the 1s (as before)

■  $\text{firstChild}(i) = \text{select}_0(\text{rank}_1(i)) + 1$

# Succinct Representation of Trees - LOUDS

[Level Order Unary Degree Sequence]

- add extra root with out-degree 1
- unary encoding of out-degree terminated by a 0
- 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

execute  $\text{select}(j)$  on  
the 0s instead of the 1s

execute  $\text{rank}(i)$  on  
the 1s (as before)

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

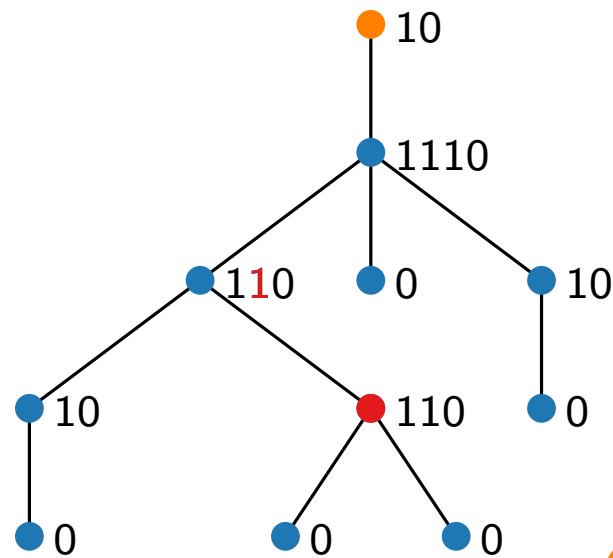
$$\text{firstChild}(8) = \text{select}_0(\text{rank}_1(8)) + 1$$



# Succinct Representation of Trees - LOUDS

[Level Order Unary Degree Sequence]

- add extra root with out-degree 1
- unary encoding of out-degree terminated by a 0
- 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

execute  $\text{select}(j)$  on  
the 0s instead of the 1s

execute  $\text{rank}(i)$  on  
the 1s (as before)

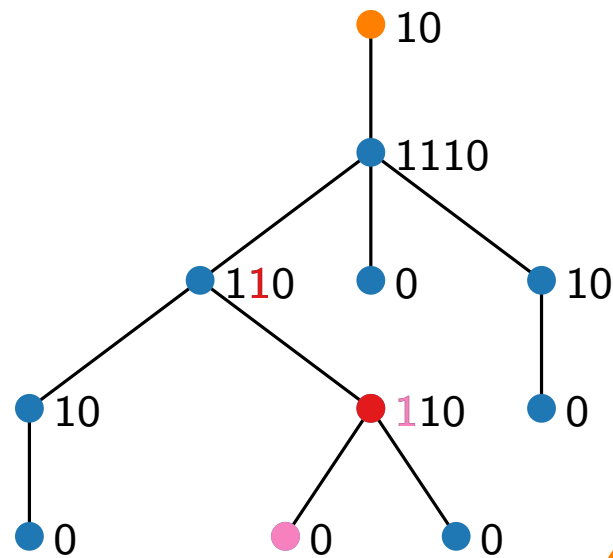
- $\text{firstChild}(i) = \text{select}_0(\text{rank}_1(i)) + 1$

$$\begin{aligned} \text{firstChild}(8) &= \text{select}_0(\text{rank}_1(8)) + 1 \\ &= \text{select}_0(6) + 1 \end{aligned}$$

# Succinct Representation of Trees - LOUDS

[Level Order Unary Degree Sequence]

- add extra root with out-degree 1
- unary encoding of out-degree terminated by a 0
- 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

execute  $\text{select}(j)$  on  
the 0s instead of the 1s

execute  $\text{rank}(i)$  on  
the 1s (as before)

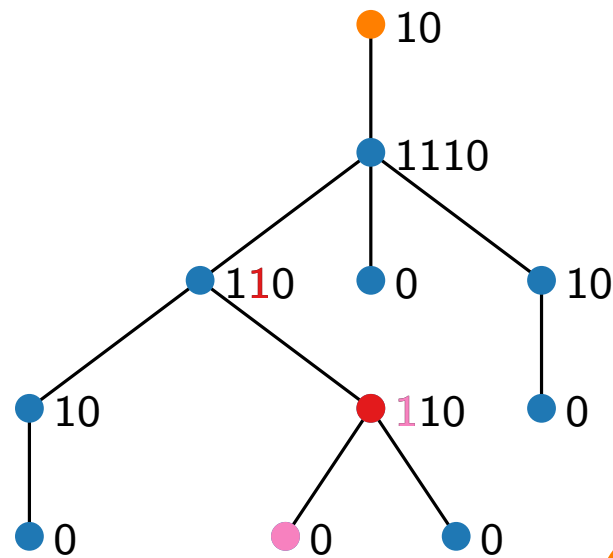
- $\text{firstChild}(i) = \text{select}_0(\text{rank}_1(i)) + 1$

$$\begin{aligned} \text{firstChild}(8) &= \text{select}_0(\text{rank}_1(8)) + 1 \\ &= \text{select}_0(6) + 1 = 14 + 1 = 15 \end{aligned}$$

# Succinct Representation of Trees - LOUDS

[Level Order Unary Degree Sequence]

- add extra root with out-degree 1
- unary encoding of out-degree terminated by a 0
- 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

execute  $\text{select}(j)$  on the 0s instead of the 1s

execute  $\text{rank}(i)$  on the 1s (as before)

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

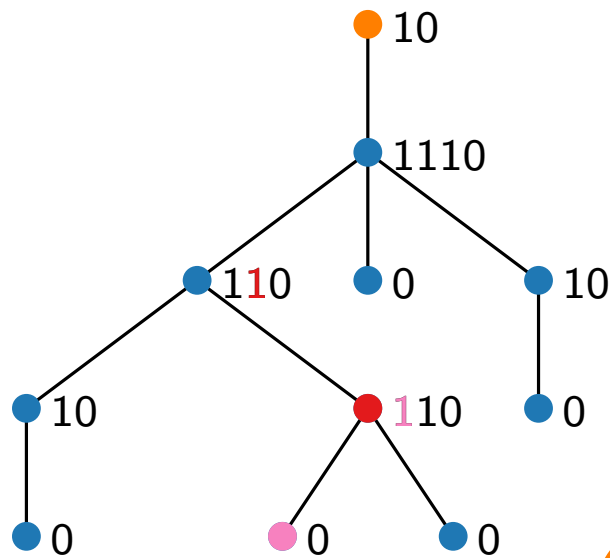
$$\begin{aligned} \text{firstChild}(8) &= \text{select}_0(\text{rank}_1(8)) + 1 \\ &= \text{select}_0(6) + 1 = 14 + 1 = 15 \end{aligned}$$

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

# Succinct Representation of Trees - LOUDS

[Level Order Unary Degree Sequence]

- add extra root with out-degree 1
- unary encoding of out-degree terminated by a 0
- 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

execute  $\text{select}(j)$  on the 0s instead of the 1s

execute  $\text{rank}(i)$  on the 1s (as before)

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

$$\begin{aligned} \text{firstChild}(8) &= \text{select}_0(\text{rank}_1(8)) + 1 \\ &= \text{select}_0(6) + 1 = 14 + 1 = 15 \end{aligned}$$

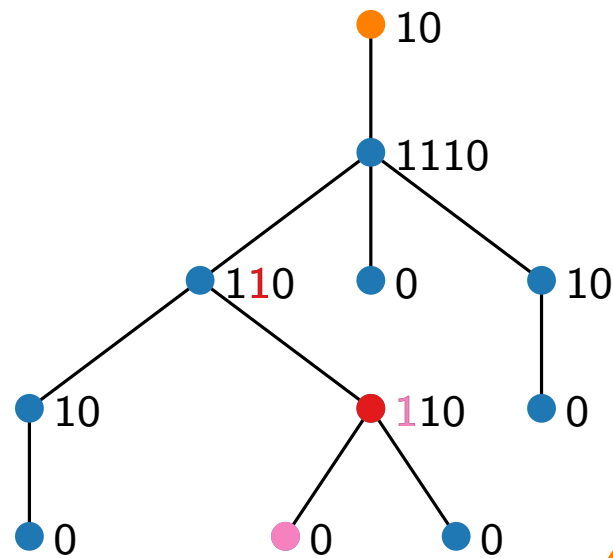
- $\text{nextSibling}(i) = i + 1$

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

# Succinct Representation of Trees - LOUDS

[Level Order Unary Degree Sequence]

- add extra root with out-degree 1
- unary encoding of out-degree terminated by a 0
- 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

execute  $\text{select}(j)$  on the 0s instead of the 1s

execute  $\text{rank}(i)$  on the 1s (as before)

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

$$\begin{aligned} \text{firstChild}(8) &= \text{select}_0(\text{rank}_1(8)) + 1 \\ &= \text{select}_0(6) + 1 = 14 + 1 = 15 \end{aligned}$$

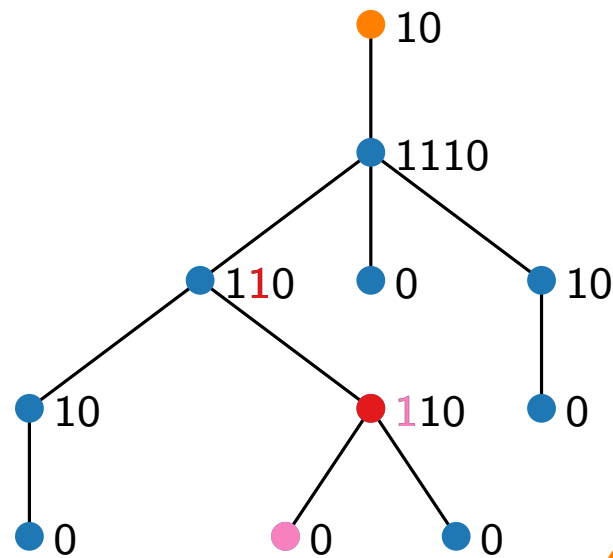
- $\text{nextSibling}(i) = i + 1$

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

# Succinct Representation of Trees - LOUDS

[Level Order Unary Degree Sequence]

- add extra root with out-degree 1
- unary encoding of out-degree terminated by a 0
- 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
1	0	1	1	1	0	1	1	0	0	1	0	1	0	1	1	0	0	0	0	0

execute  $\text{select}(j)$  on the 0s instead of the 1s

execute  $\text{rank}(i)$  on the 1s (as before)

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

- $\text{parent}(i) = \text{select}_1(\text{rank}_0(i))$

$$\begin{aligned} \text{firstChild}(8) &= \text{select}_0(\text{rank}_1(8)) + 1 \\ &= \text{select}_0(6) + 1 = 14 + 1 = 15 \end{aligned}$$

$$\text{parent}(8) = \text{select}_1(\text{rank}_0(8))$$

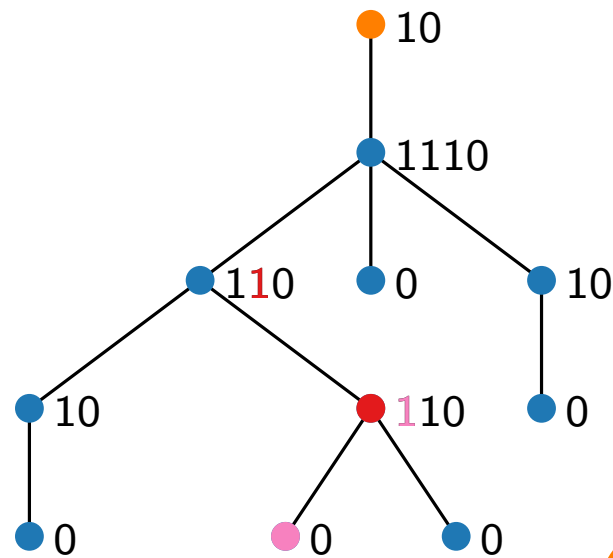
- $\text{nextSibling}(i) = i + 1$

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

# Succinct Representation of Trees - LOUDS

[Level Order Unary Degree Sequence]

- add extra root with out-degree 1
- unary encoding of out-degree terminated by a 0
- 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
1	0	1	1	1	0	1	1	0	0	1	0	1	0	1	1	0	0	0	0	0

execute  $\text{select}(j)$  on the 0s instead of the 1s

execute  $\text{rank}(i)$  on the 1s (as before)

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

$$\begin{aligned} \text{firstChild}(8) &= \text{select}_0(\text{rank}_1(8)) + 1 \\ &= \text{select}_0(6) + 1 = 14 + 1 = 15 \end{aligned}$$

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

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

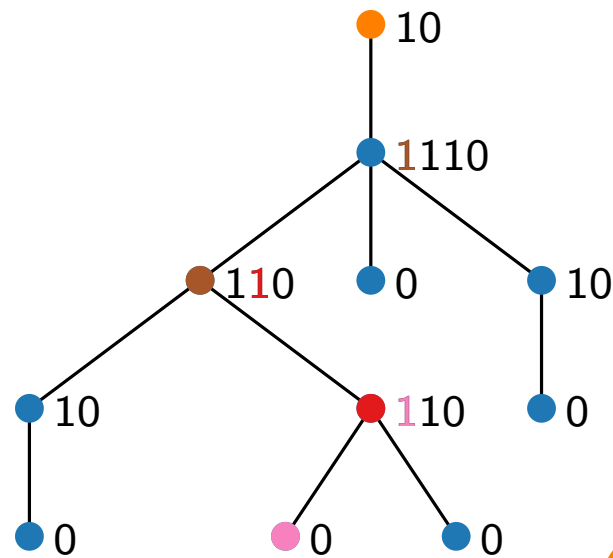
- $\text{parent}(i) = \text{select}_1(\text{rank}_0(i))$

$$\begin{aligned} \text{parent}(8) &= \text{select}_1(\text{rank}_0(8)) \\ &= \text{select}_1(2) \end{aligned}$$

# Succinct Representation of Trees - LOUDS

[Level Order Unary Degree Sequence]

- add extra root with out-degree 1
- unary encoding of out-degree terminated by a 0
- 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
1	0	1	1	1	0	1	1	0	0	1	0	1	0	1	1	0	0	0	0	0

execute  $\text{select}(j)$  on the 0s instead of the 1s

execute  $\text{rank}(i)$  on the 1s (as before)

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

$$\begin{aligned} \text{firstChild}(8) &= \text{select}_0(\text{rank}_1(8)) + 1 \\ &= \text{select}_0(6) + 1 = 14 + 1 = 15 \end{aligned}$$

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

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

- $\text{parent}(i) = \text{select}_1(\text{rank}_0(i))$

$$\begin{aligned} \text{parent}(8) &= \text{select}_1(\text{rank}_0(8)) \\ &= \text{select}_1(2) = 3 \end{aligned}$$



# Discussion

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

# Discussion

- Succinct data structures are
    - space efficient
    - support fast operationsbut
    - are mostly static (dynamic at extra cost),
    - number of operations is limited,
    - complex → harder to implement,
    - the  $o(n)$  and  $O(1)$  term hide constants that might dominate before any asymptotic advantage over the “best” compact data structures becomes apparent.  
→ primarily a theoretical result (also does not consider hardware architecture)
- that means insertions & deletions

# Discussion

- Succinct data structures are
    - space efficient
    - support fast operationsbut
    - are mostly static (dynamic at extra cost),
    - number of operations is limited,
    - complex → harder to implement,
    - the  $o(n)$  and  $O(1)$  term hide constants that might dominate before any asymptotic advantage over the “best” compact data structures becomes apparent.  
→ primarily a theoretical result (also does not consider hardware architecture)
  - rank and select form the basis for many succinct representations (e.g., for specific types of trees or strings).
- that means insertions & deletions

# Discussion

- Succinct data structures are
  - space efficient
  - support fast operationsbut
  - are mostly static (dynamic at extra cost),
  - number of operations is limited,
  - complex  $\rightarrow$  harder to implement,
  - the  $o(n)$  and  $O(1)$  term hide constants that might dominate before any asymptotic advantage over the “best” compact data structures becomes apparent.  
 $\rightarrow$  primarily a theoretical result (also does not consider hardware architecture)
- rank and select form the basis for many succinct representations (e.g., for specific types of trees or strings).
- There are implementations of succinct data structures being used in practice for large data sets in information retrieval, language model representation, bioinformatics, etc.

that means insertions & deletions



# Literature

Main reference:

- Lecture 17 of Advanced Data Structures (MIT, Fall'17) by Erik Demaine
- [Jac '89] "Space efficient Static Trees and Graphs"

Recommendations:

- Lecture 18 of Demaine's course on compact & succinct arrays & trees