# Computational Geometry

## Real RAM, Word RAM, Point Location

## What is a computer?

Thomas Bläsius

# What Can Your Computer Do?

## Model Of Computation

- RAM (random access machine): memory access in $O(1)$ via an address

# What Can Your Computer Do?

## Model Of Computation

- RAM (random access machine): memory access in $O(1)$ via an address
- real RAM
  - every memory cell holds a real number (of arbitrary size/precision)

# What Can Your Computer Do?

## Model Of Computation

- RAM (random access machine): memory access in $O(1)$ via an address
- real RAM
  - every memory cell holds a real number (of arbitrary size/precision)
  - arithmetic operations $(+, -, \cdot, /)$ in $O(1)$

# What Can Your Computer Do?

## Model Of Computation

- RAM (random access machine): memory access in $O(1)$ via an address
- real RAM
  - every memory cell holds a real number (of arbitrary size/precision)
  - arithmetic operations $(+, -, \cdot, /)$ in $O(1)$
  - rounding to integers is **not** allowed (otherwise, you can do broken things)

# What Can Your Computer Do?

## Model Of Computation

- RAM (random access machine): memory access in $O(1)$ via an address
- real RAM
  - every memory cell holds a real number (of arbitrary size/precision)
  - arithmetic operations $(+, -, \cdot, /)$ in $O(1)$
  - rounding to integers is **not** allowed (otherwise, you can do broken things)
  - common model in computational geometry $\rightarrow$ abstracts away precision issues

# What Can Your Computer Do?

## Model Of Computation

- RAM (random access machine): memory access in $O(1)$ via an address
- real RAM
  - every memory cell holds a real number (of arbitrary size/precision)
  - arithmetic operations $(+, -, \cdot, /)$ in $O(1)$
  - rounding to integers is **not** allowed (otherwise, you can do broken things)
  - common model in computational geometry $\rightarrow$ abstracts away precision issues
  - potential problem: sometimes too powerful (more powerful than your computer)

# What Can Your Computer Do?

## Model Of Computation

- RAM (random access machine): memory access in $O(1)$ via an address
- real RAM
  - every memory cell holds a real number (of arbitrary size/precision)
  - arithmetic operations $(+, -, \cdot, /)$ in $O(1)$
  - rounding to integers is **not** allowed (otherwise, you can do broken things)
  - common model in computational geometry $\rightarrow$ abstracts away precision issues
  - potential problem: sometimes too powerful (more powerful than your computer)
- word RAM
  - every memory cell holds a **word** consisting of $w$ bits
  - $w$ is sufficiently large ($\geq \log n$, but potentially much larger)

# What Can Your Computer Do?

## Model Of Computation

- RAM (random access machine): memory access in $O(1)$ via an address
- real RAM
  - every memory cell holds a real number (of arbitrary size/precision)
  - arithmetic operations $(+, -, \cdot, /)$ in $O(1)$
  - rounding to integers is **not** allowed (otherwise, you can do broken things)
  - common model in computational geometry $\rightarrow$ abstracts away precision issues
  - potential problem: sometimes too powerful (more powerful than your computer)
- word RAM
  - every memory cell holds a **word** consisting of $w$ bits
  - $w$ is sufficiently large ($\geq \log n$, but potentially much larger)
  - arithmetic operations on integers (of size up to $2^w$) in $O(1)$

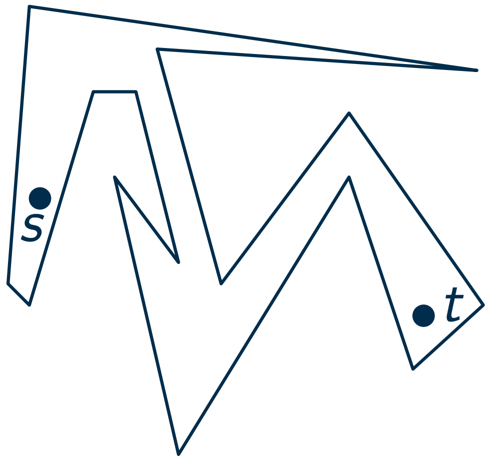
# What Can Your Computer Do?

## Model Of Computation

- RAM (random access machine): memory access in $O(1)$ via an address
- real RAM
  - every memory cell holds a real number (of arbitrary size/precision)
  - arithmetic operations $(+, -, \cdot, /)$ in $O(1)$
  - rounding to integers is **not** allowed (otherwise, you can do broken things)
  - common model in computational geometry $\to$ abstracts away precision issues
  - potential problem: sometimes too powerful (more powerful than your computer)
- word RAM
  - every memory cell holds a **word** consisting of $w$ bits
  - $w$ is sufficiently large ($\geq \log n$, but potentially much larger)
  - arithmetic operations on integers (of size up to $2^w$) in $O(1)$
  - bit-wise logical operations and bit shifts in $O(1)$

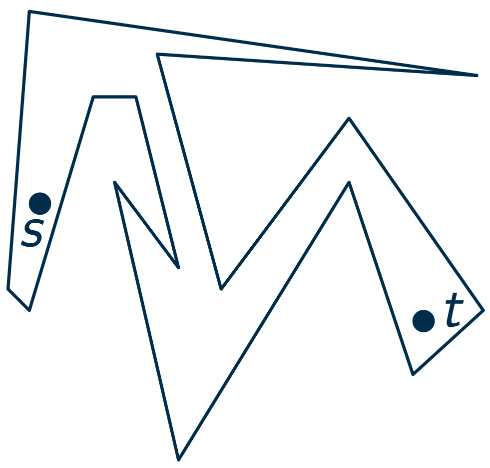
# Minimum Link Path In Polygons

**Problem: Minimum Link Path**
Given a polygon $P$ as well as points $s$ and $t$ in $P$, compute an $st$-path inside $P$ with the minimum number of segments.



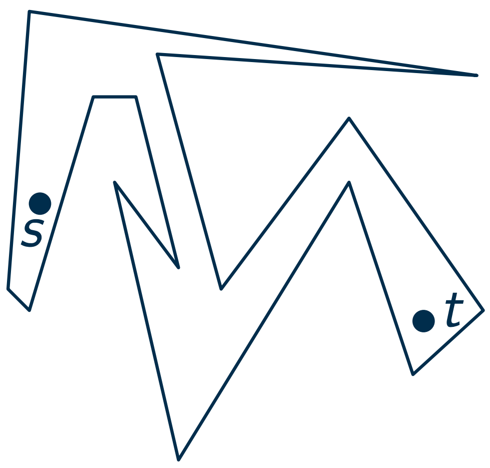
# Minimum Link Path In Polygons

**Problem: Minimum Link Path**
Given a polygon $P$ as well as points $s$ and $t$ in $P$, compute an $st$-path inside $P$ with the minimum number of segments.




**Theorem** **(without proof)**
The minimum link path between two points in a polygon of size $n$ can be computed in $O(n)$ time.

# Minimum Link Path In Polygons




**Problem: Minimum Link Path**
Given a polygon $P$ as well as points $s$ and $t$ in $P$, compute an $st$-path inside $P$ with the minimum number of segments.

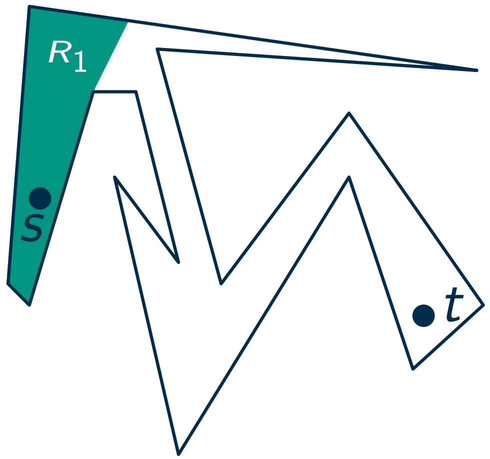
**Theorem** **(without proof)**
The minimum link path between two points in a polygon of size $n$ can be computed in $O(n)$ time.

## General Idea

- $R_i$ = set of points reachable from $s$ with $i$ links
- iteratively compute $R_{i+1}$ from $R_i$

# Minimum Link Path In Polygons

**Problem: Minimum Link Path**
Given a polygon $P$ as well as points $s$ and $t$ in $P$, compute an $st$-path inside $P$ with the minimum number of segments.

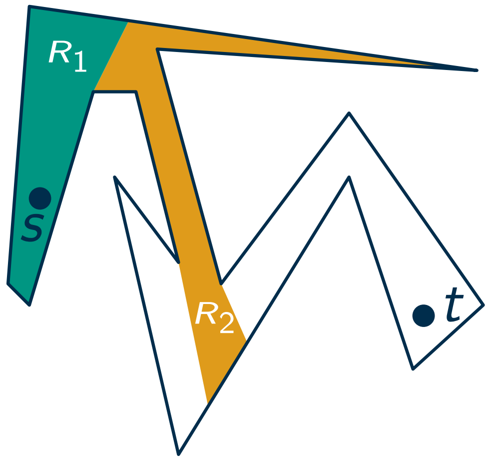



**Theorem** **(without proof)**
The minimum link path between two points in a polygon of size $n$ can be computed in $O(n)$ time.

**General Idea**

- $R_i$ = set of points reachable from $s$ with $i$ links
- iteratively compute $R_{i+1}$ from $R_i$

# Minimum Link Path In Polygons



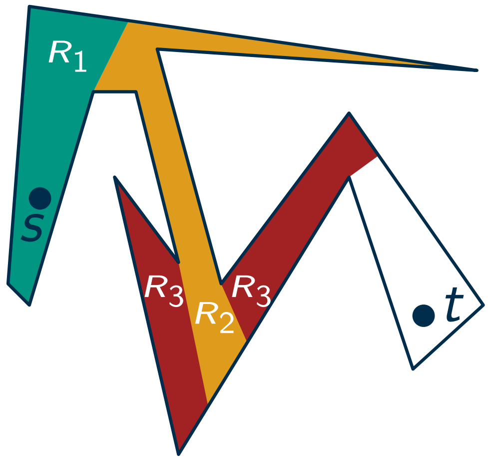

**Problem: Minimum Link Path**
Given a polygon $P$ as well as points $s$ and $t$ in $P$, compute an $st$-path inside $P$ with the minimum number of segments.

**Theorem** **(without proof)**
The minimum link path between two points in a polygon of size $n$ can be computed in $O(n)$ time.

**General Idea**

- $R_i$ = set of points reachable from $s$ with $i$ links
- iteratively compute $R_{i+1}$ from $R_i$

# Minimum Link Path In Polygons



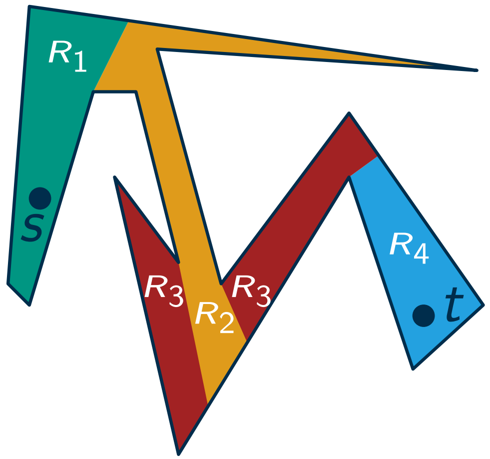

**Problem: Minimum Link Path**
Given a polygon $P$ as well as points $s$ and $t$ in $P$, compute an $st$-path inside $P$ with the minimum number of segments.

**Theorem** **(without proof)**
The minimum link path between two points in a polygon of size $n$ can be computed in $O(n)$ time.

**General Idea**

- $R_i$ = set of points reachable from $s$ with $i$ links
- iteratively compute $R_{i+1}$ from $R_i$

# Minimum Link Path In Polygons



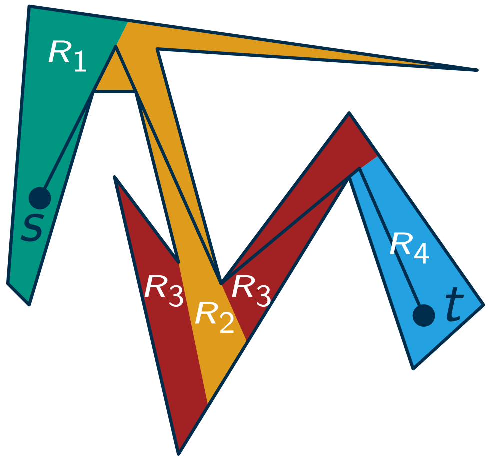

**Problem: Minimum Link Path**
Given a polygon $P$ as well as points $s$ and $t$ in $P$, compute an $st$-path inside $P$ with the minimum number of segments.

**Theorem** **(without proof)**
The minimum link path between two points in a polygon of size $n$ can be computed in $O(n)$ time.

**General Idea**

- $R_i$ = set of points reachable from $s$ with $i$ links
- iteratively compute $R_{i+1}$ from $R_i$

# Minimum Link Path In Polygons



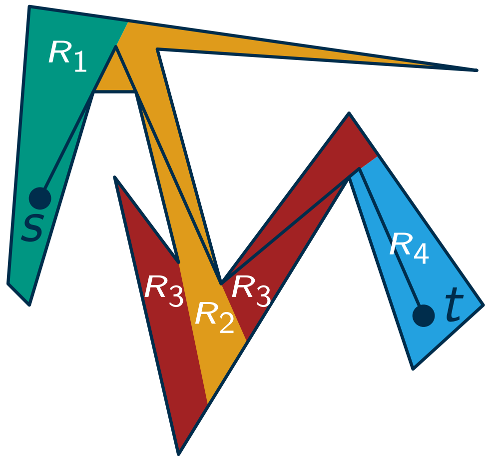

**Problem: Minimum Link Path**
Given a polygon $P$ as well as points $s$ and $t$ in $P$, compute an $st$-path inside $P$ with the minimum number of segments.

**Theorem** **(without proof)**
The minimum link path between two points in a polygon of size $n$ can be computed in $O(n)$ time.

## General Idea

- $R_i$ = set of points reachable from $s$ with $i$ links
- iteratively compute $R_{i+1}$ from $R_i$

# Minimum Link Path In Polygons




**Problem: Minimum Link Path**
Given a polygon $P$ as well as points $s$ and $t$ in $P$, compute an $st$-path inside $P$ with the minimum number of segments.

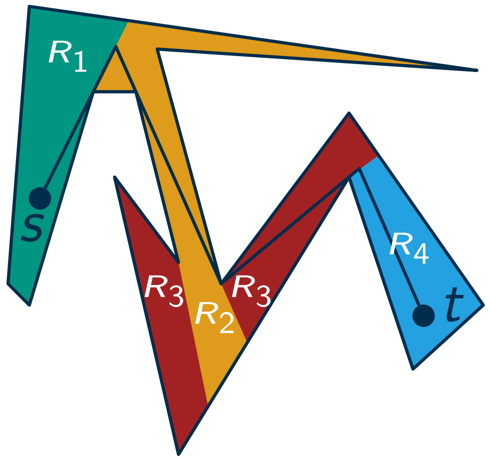
**Theorem** **(without proof)**
The minimum link path between two points in a polygon of size $n$ can be computed in $O(n)$ time.

**Theorem** **(without proof)**
There are instances encodable with $\Theta(n \log n)$ bits such that representing the polygons $R_1, \ldots, R_n$ requires $\Theta(n^2 \log n)$ bits.

## General Idea

- $R_i$ = set of points reachable from $s$ with $i$ links
- iteratively compute $R_{i+1}$ from $R_i$

# Minimum Link Path In Polygons




**Problem: Minimum Link Path**
Given a polygon $P$ as well as points $s$ and $t$ in $P$, compute an $st$-path inside $P$ with the minimum number of segments.

**Theorem** **(without proof)**
The minimum link path between two points in a polygon of size $n$ can be computed in $O(n)$ time.

**Theorem** **(without proof)**
There are instances encodable with $\Theta(n \log n)$ bits such that representing the polygons $R_1, \ldots, R_n$ requires $\Theta(n^2 \log n)$ bits.

## General Idea

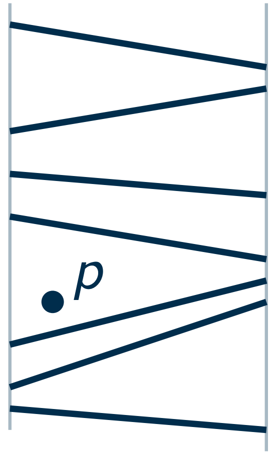
- $R_i$ = set of points reachable from $s$ with $i$ links
- iteratively compute $R_{i+1}$ from $R_i$

## Which Of Them Is True?

- first theorem assumes a real RAM
- an implementations (e.g., with doubles for coordinates) is maybe not robust

# Point-Location In A Vertical Slab (word RAM)

**Problem**
Let $S$ be a set of disjoint segments between two vertical lines. Build a data structure that can answer between which two segments a query point $p$ lies. (with acceptable memory consumption)

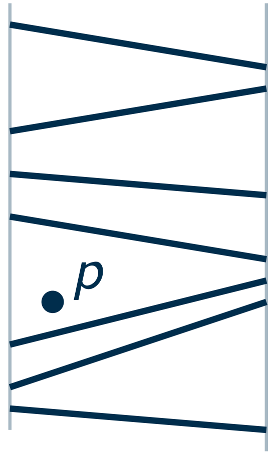


# Point-Location In A Vertical Slab (word RAM)

**Problem**
Let $S$ be a set of disjoint segments between two vertical lines. Build a data structure that can answer between which two segments a query point $p$ lies. (with acceptable memory consumption)

## Lets Start With One Dimension

- predecessor search in a sequence of numbers
- default solution: binary search provides $\Theta(\log n)$ queries

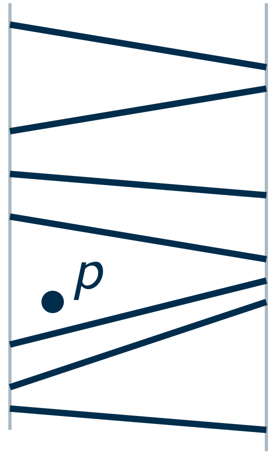


# Point-Location In A Vertical Slab (word RAM)

**Problem**
Let $S$ be a set of disjoint segments between two vertical lines. Build a data structure that can answer between which two segments a query point $p$ lies. (with acceptable memory consumption)

## Lets Start With One Dimension

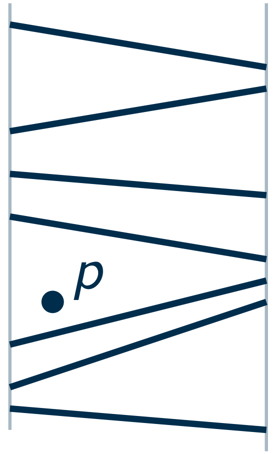
- predecessor search in a sequence of numbers
- default solution: binary search provides $\Theta(\log n)$ queries
- goal: queries in $o(\log n)$



# Point-Location In A Vertical Slab (word RAM)

**Problem**
Let $S$ be a set of disjoint segments between two vertical lines. Build a data structure that can answer between which two segments a query point $p$ lies. (with acceptable memory consumption)

## Lets Start With One Dimension




- predecessor search in a sequence of numbers
- default solution: binary search provides $\Theta(\log n)$ queries
- goal: queries in $o(\log n)$
- use properties of the word RAM
  - numbers are integers
  - numbers lie in the interval $[0, 2^w)$
  - arithmetic operations, bit-wise logical operations, and bit-shifts on words of length $w$ in $O(1)$

# Beating The $\log n$ Lower Bound

**Binary Search Basics**

- walks down a decision tree
- decision for left/right subset: one comparison
- per step: number of integers is halved
- recursion depth: $\log_2(n)$

# Beating The $\log n$ Lower Bound

## Ideas For Improvement

- wider branching: shrink number of possible integers by the factor $b$ in each step

**Binary Search Basics**

- walks down a decision tree
- decision for left/right subset: one comparison
- per step: number of integers is halved
- recursion depth: $\log_2(n)$

# Beating The $\log n$ Lower Bound

## Ideas For Improvement

- wider branching: shrink number of possible integers by the factor $b$ in each step
  - recursion depth: $\log_b(n)$ (sub-logarithmic if $b$ super-constant)

**Binary Search Basics**

- walks down a decision tree
- decision for left/right subset: one comparison
- per step: number of integers is halved
- recursion depth: $\log_2(n)$

# Beating The $\log n$ Lower Bound

## Ideas For Improvement

- wider branching: shrink number of possible integers by the factor $b$ in each step
  - recursion depth: $\log_b(n)$ (sub-logarithmic if $b$ super-constant)
  - problem: decision for the correct subinterval too expensive

**Binary Search Basics**

- walks down a decision tree
- decision for left/right subset: one comparison
- per step: number of integers is halved
- recursion depth: $\log_2(n)$

# Beating The $\log n$ Lower Bound

## Ideas For Improvement

- wider branching: shrink number of possible integers by the factor $b$ in each step
  - recursion depth: $\log_b(n)$ (sub-logarithmic if $b$ super-constant)
  - problem: decision for the correct subinterval too expensive
- shrinking the interval (in the beginning, all numbers lie in $[0, 2^w)$)
  - shrink the interval by the factor $2^h$

**Binary Search Basics**

- walks down a decision tree
- decision for left/right subset: one comparison
- per step: number of integers is halved
- recursion depth: $\log_2(n)$

# Beating The $\log n$ Lower Bound

## Ideas For Improvement

- wider branching: shrink number of possible integers by the factor $b$ in each step
  - recursion depth: $\log_b(n)$ (sub-logarithmic if $b$ super-constant)
  - problem: decision for the correct subinterval too expensive
- shrinking the interval (in the beginning, all numbers lie in $[0, 2^w)$)
  - shrink the interval by the factor $2^h$
  - recursion depth: $\frac{w}{h}$

**Binary Search Basics**

- walks down a decision tree
- decision for left/right subset: one comparison
- per step: number of integers is halved
- recursion depth: $\log_2(n)$

# Beating The $\log n$ Lower Bound

## Ideas For Improvement

- wider branching: shrink number of possible integers by the factor $b$ in each step
  - recursion depth: $\log_b(n)$ (sub-logarithmic if $b$ super-constant)
  - problem: decision for the correct subinterval too expensive
- shrinking the interval (in the beginning, all numbers lie in $[0, 2^w)$)
  - shrink the interval by the factor $2^h$
  - recursion depth: $\frac{w}{h}$
  - problem: deciding whether a subinterval still contains integers

**Binary Search Basics**

- walks down a decision tree
- decision for left/right subset: one comparison
- per step: number of integers is halved
- recursion depth: $\log_2(n)$

# Beating The $\log n$ Lower Bound

## Ideas For Improvement

- wider branching: shrink number of possible integers by the factor $b$ in each step
  - recursion depth: $\log_b(n)$ (sub-logarithmic if $b$ super-constant)
  - problem: decision for the correct subinterval too expensive
- shrinking the interval (in the beginning, all numbers lie in $[0, 2^w)$)
  - shrink the interval by the factor $2^h$
  - recursion depth: $\frac{w}{h}$
  - problem: deciding whether a subinterval still contains integers
- combining both ideas
  - in each step: shrink the number of integers or the interval

**Binary Search Basics**

- walks down a decision tree
- decision for left/right subset: one comparison
- per step: number of integers is halved
- recursion depth: $\log_2(n)$

# Beating The $\log n$ Lower Bound

## Ideas For Improvement

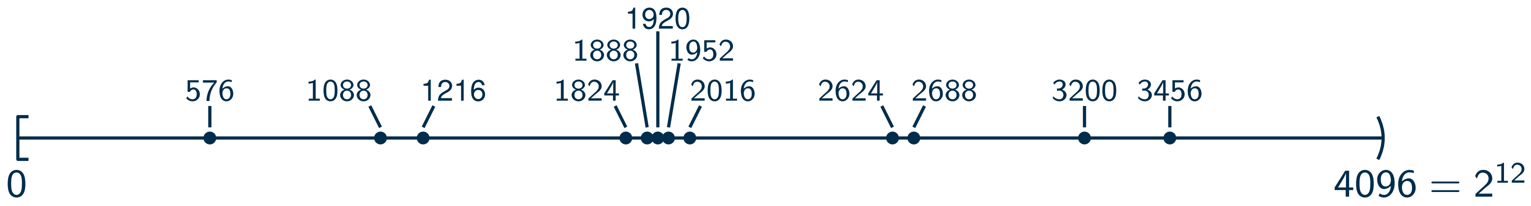
- wider branching: shrink number of possible integers by the factor $b$ in each step
  - recursion depth: $\log_b(n)$ (sub-logarithmic if $b$ super-constant)
  - problem: decision for the correct subinterval too expensive
- shrinking the interval (in the beginning, all numbers lie in $[0, 2^w)$)
  - shrink the interval by the factor $2^h$
  - recursion depth: $\frac{w}{h}$
  - problem: deciding whether a subinterval still contains integers
- combining both ideas
  - in each step: shrink the number of integers or the interval
  - decision for the correct interval: $O(1)$ with bit magic

**Binary Search Basics**

- walks down a decision tree
- decision for left/right subset: one comparison
- per step: number of integers is halved
- recursion depth: $\log_2(n)$

# Subdivision Into Subintervals

word size: $w$
number of integers: $n$
interval: $[0, 2^\ell)$
number of cells: $2^h$
branching width: $b$



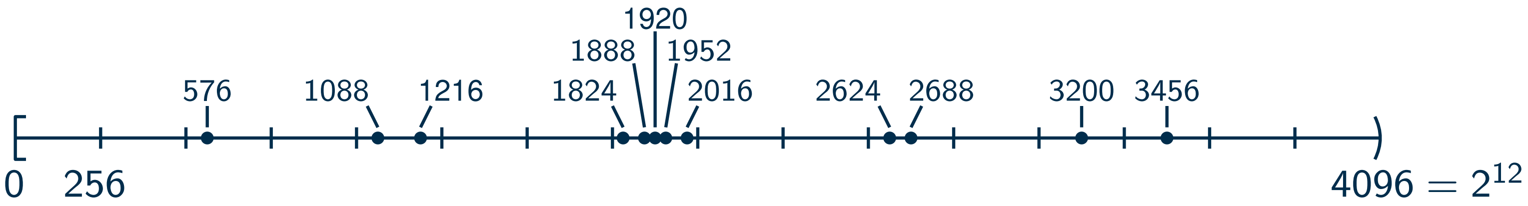

$n = 12$
$\ell = 12$
$h = 4$
$b = 3$

# Subdivision Into Subintervals

word size: $w$
number of integers: $n$
interval: $[0, 2^\ell)$
number of cells: $2^h$
branching width: $b$

## One Subdivision Step

- divide interval of size $2^\ell$ into $2^h$ cells of size $2^{\ell-h}$



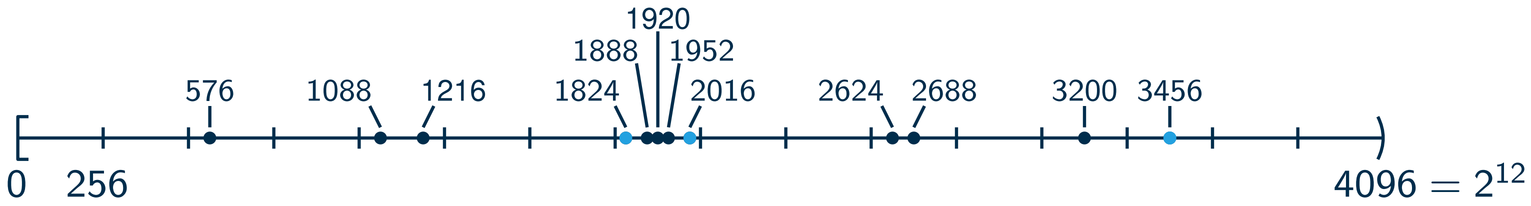

$n = 12$
$\ell = 12$
$h = 4$
$b = 3$

# Subdivision Into Subintervals

word size: $w$
number of integers: $n$
interval: $[0, 2^\ell)$
number of cells: $2^h$
branching width: $b$

## One Subdivision Step

- divide interval of size $2^\ell$ into $2^h$ cells of size $2^{\ell-h}$
- mark each $\frac{n}{b}$th integer



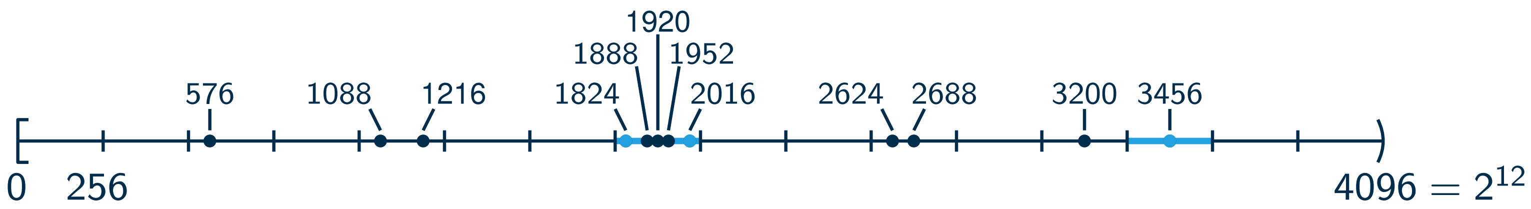

$n = 12$
$\ell = 12$
$h = 4$
$b = 3$

# Subdivision Into Subintervals

word size: $w$
number of integers: $n$
interval: $[0, 2^\ell)$
number of cells: $2^h$
branching width: $b$

## One Subdivision Step

- divide interval of size $2^\ell$ into $2^h$ cells of size $2^{\ell-h}$
- mark each $\frac{n}{b}$th integer
- mark cells that contain marked integers



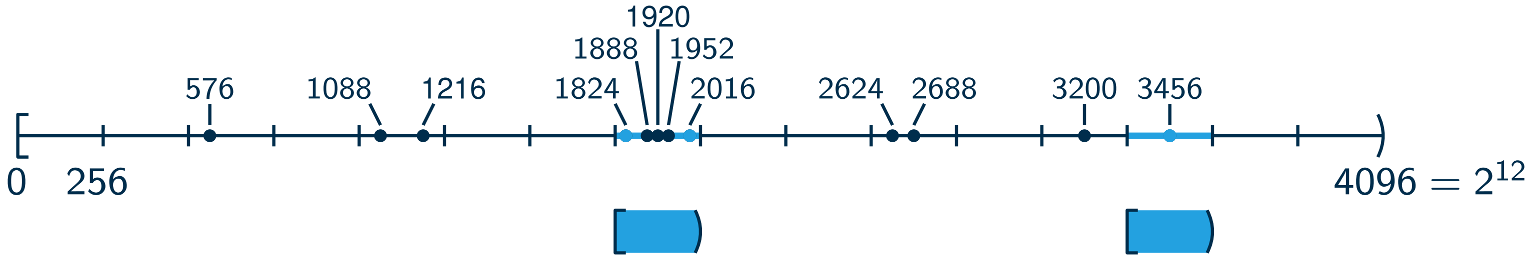

$n = 12$
$\ell = 12$
$h = 4$
$b = 3$

# Subdivision Into Subintervals

word size: $w$
number of integers: $n$
interval: $[0, 2^\ell)$
number of cells: $2^h$
branching width: $b$

## One Subdivision Step

- divide interval of size $2^\ell$ into $2^h$ cells of size $2^{\ell-h}$
- mark each $\frac{n}{b}$th integer
- mark cells that contain marked integers
- define partition into subintervals
  - each marked cell is a subinterval



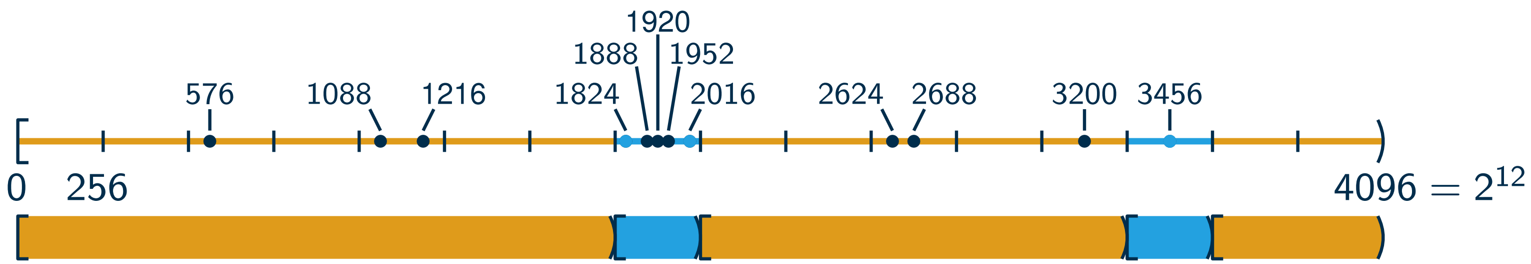

$n = 12$
$\ell = 12$
$h = 4$
$b = 3$

# Subdivision Into Subintervals

word size: $w$
number of integers: $n$
interval: $[0, 2^\ell)$
number of cells: $2^h$
branching width: $b$

## One Subdivision Step

- divide interval of size $2^\ell$ into $2^h$ cells of size $2^{\ell-h}$
- mark each $\frac{n}{b}$th integer
- mark cells that contain marked integers
- define partition into subintervals
  - each marked cell is a subinterval
  - each maximal sequence of unmarked cells is a subinterval



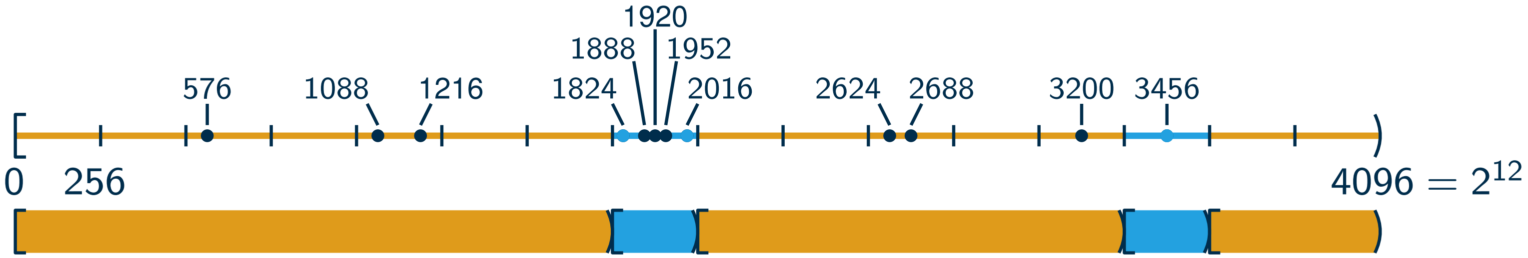

$n = 12$
$\ell = 12$
$h = 4$
$b = 3$

# Subdivision Into Subintervals

word size: $w$
number of integers: $n$
interval: $[0, 2^\ell)$
number of cells: $2^h$
branching width: $b$

## One Subdivision Step

- divide interval of size $2^\ell$ into $2^h$ cells of size $2^{\ell-h}$
- mark each $\frac{n}{b}$th integer
- mark cells that contain marked integers
- define partition into subintervals
  - each marked cell is a subinterval
  - each maximal sequence of unmarked cells is a subinterval




$n = 12$
$\ell = 12$
$h = 4$
$b = 3$

## Properties Of This Subdivision

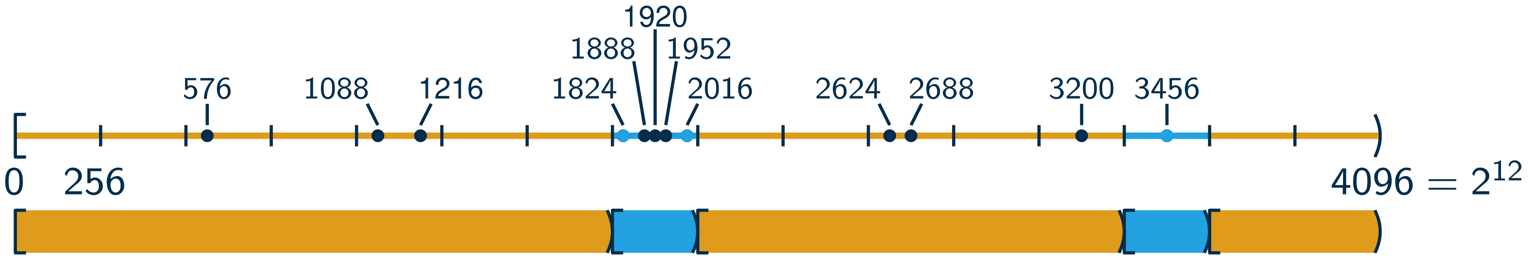
- we have $O(b)$ subintervals

# Subdivision Into Subintervals

word size: $w$
number of integers: $n$
interval: $[0, 2^\ell)$
number of cells: $2^h$
branching width: $b$

## One Subdivision Step

- divide interval of size $2^\ell$ into $2^h$ cells of size $2^{\ell-h}$
- mark each $\frac{n}{b}$th integer
- mark cells that contain marked integers
- define partition into subintervals
  - each marked cell is a subinterval
  - each maximal sequence of unmarked cells is a subinterval




$n = 12$
$\ell = 12$
$h = 4$
$b = 3$

## Properties Of This Subdivision

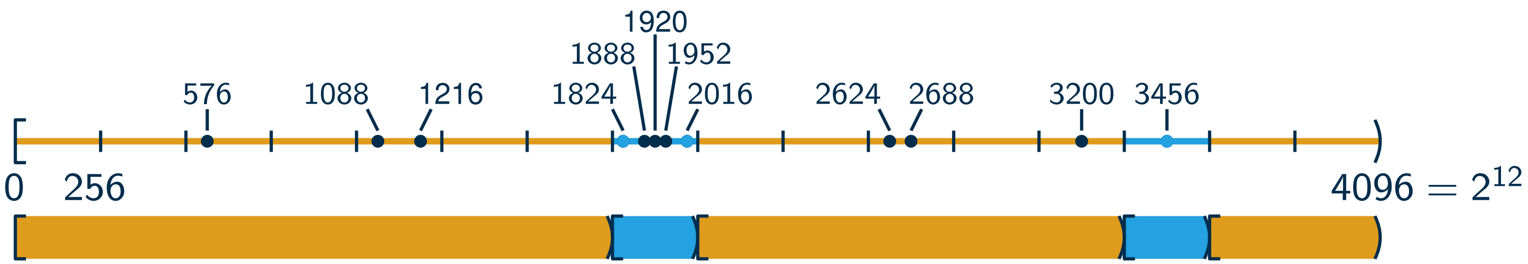
- we have $O(b)$ subintervals
- the boundaries of each subinterval is a multiple of $2^{\ell-h}$

# Subdivision Into Subintervals

word size: $w$
number of integers: $n$
interval: $[0, 2^\ell)$
number of cells: $2^h$
branching width: $b$

## One Subdivision Step

- divide interval of size $2^\ell$ into $2^h$ cells of size $2^{\ell-h}$
- mark each $\frac{n}{b}$th integer
- mark cells that contain marked integers
- define partition into subintervals
  - each marked cell is a subinterval
  - each maximal sequence of unmarked cells is a subinterval




$n = 12$
$\ell = 12$
$h = 4$
$b = 3$

## Properties Of This Subdivision

- we have $O(b)$ subintervals
- the boundaries of each subinterval is a multiple of $2^{\ell-h}$
- each subinterval has length $2^{\ell-h}$ or contains at most $\frac{n}{b}$ integers

# Recursive Subdivision

word size: $w$
number of integers: $n$
interval: $[0, 2^\ell)$
number of cells: $2^h$
branching width: $b$

**Lemma** **(nice $(h, b)$-subdivision)**

Given $n$ integers in $I = [0, 2^\ell)$. $I$ can be subdivided into $O(b)$ subintervals such that:

- each subinterval has length $2^{\ell-h}$ or contains at most $\frac{n}{b}$ integers
- the boundaries of the subintervals are multiples of $2^{\ell-h}$

# Recursive Subdivision

word size: $w$
number of integers: $n$
interval: $[0, 2^\ell)$
number of cells: $2^h$
branching width: $b$

**Lemma** **(nice $(h, b)$-subdivision)**

Given $n$ integers in $I = [0, 2^\ell)$. $I$ can be subdivided into $O(b)$ subintervals such that:

- each subinterval has length $2^{\ell-h}$ or contains at most $\frac{n}{b}$ integers
- the boundaries of the subintervals are multiples of $2^{\ell-h}$

**Recursive Decision Tree**

- one child for each subinterval in the $(h, b)$-subdivision

# Recursive Subdivision

word size: $w$
number of integers: $n$
interval: $[0, 2^\ell)$
number of cells: $2^h$
branching width: $b$

**Lemma** **(nice $(h, b)$-subdivision)**

Given $n$ integers in $I = [0, 2^\ell)$. $I$ can be subdivided into $O(b)$ subintervals such that:

- each subinterval has length $2^{\ell-h}$ or contains at most $\frac{n}{b}$ integers
- the boundaries of the subintervals are multiples of $2^{\ell-h}$

## Recursive Decision Tree

- one child for each subinterval in the $(h, b)$-subdivision

**Note:** In the recursive calls, we may need to shift the subinterval to 0 and increase it to the next power of 2, such that it has the form $[0, 2^\ell)$.

# Recursive Subdivision

word size: $w$
number of integers: $n$
interval: $[0, 2^\ell)$
number of cells: $2^h$
branching width: $b$

**Lemma** **(nice $(h, b)$-subdivision)**

Given $n$ integers in $I = [0, 2^\ell)$. $I$ can be subdivided into $O(b)$ subintervals such that:

- each subinterval has length $2^{\ell-h}$ or contains at most $\frac{n}{b}$ integers
- the boundaries of the subintervals are multiples of $2^{\ell-h}$

## Recursive Decision Tree

- one child for each subinterval in the $(h, b)$-subdivision
- stop recursion if only few integers are left

**Note:** In the recursive calls, we may need to shift the subinterval to 0 and increase it to the next power of 2, such that it has the form $[0, 2^\ell)$.

# Recursive Subdivision

word size: $w$
number of integers: $n$
interval: $[0, 2^\ell)$
number of cells: $2^h$
branching width: $b$

**Lemma** **(nice $(h, b)$-subdivision)**

Given $n$ integers in $I = [0, 2^\ell)$. $I$ can be subdivided into $O(b)$ subintervals such that:

- each subinterval has length $2^{\ell-h}$ or contains at most $\frac{n}{b}$ integers
- the boundaries of the subintervals are multiples of $2^{\ell-h}$

## Recursive Decision Tree

- one child for each subinterval in the $(h, b)$-subdivision
- stop recursion if only few integers are left
- height of the tree: at most $\log_b(n) + \frac{w}{h}$

Why?

**Note:** In the recursive calls, we may need to shift the subinterval to 0 and increase it to the next power of 2, such that it has the form $[0, 2^\ell)$.

# Recursive Subdivision

word size: $w$
number of integers: $n$
interval: $[0, 2^\ell)$
number of cells: $2^h$
branching width: $b$

**Lemma** **(nice $(h, b)$-subdivision)**

Given $n$ integers in $I = [0, 2^\ell)$. $I$ can be subdivided into $O(b)$ subintervals such that:

- each subinterval has length $2^{\ell-h}$ or contains at most $\frac{n}{b}$ integers
- the boundaries of the subintervals are multiples of $2^{\ell-h}$

## Recursive Decision Tree

- one child for each subinterval in the $(h, b)$-subdivision
- stop recursion if only few integers are left
- height of the tree: at most $\log_b(n) + \frac{w}{h}$

Why?

## Searching For The Successor Of An Integer $q$

- find subinterval containing $q$ at most $\log_b(n) + \frac{w}{h}$ times

**Note:** In the recursive calls, we may need to shift the subinterval to 0 and increase it to the next power of 2, such that it has the form $[0, 2^\ell)$.

# Recursive Subdivision

word size: $w$
number of integers: $n$
interval: $[0, 2^\ell)$
number of cells: $2^h$
branching width: $b$

**Lemma** **(nice $(h, b)$-subdivision)**

Given $n$ integers in $I = [0, 2^\ell)$. $I$ can be subdivided into $O(b)$ subintervals such that:

- each subinterval has length $2^{\ell-h}$ or contains at most $\frac{n}{b}$ integers
- the boundaries of the subintervals are multiples of $2^{\ell-h}$

## Recursive Decision Tree

- one child for each subinterval in the $(h, b)$-subdivision
- stop recursion if only few integers are left
- height of the tree: at most $\log_b(n) + \frac{w}{h}$

Why?

**Note:** In the recursive calls, we may need to shift the subinterval to 0 and increase it to the next power of 2, such that it has the form $[0, 2^\ell)$.

## Searching For The Successor Of An Integer $q$

- find subinterval containing $q$ at most $\log_b(n) + \frac{w}{h}$ times
- bit magic: each step runs in $O(1)$ if $\frac{w}{h} \geq \frac{b}{c}$ (for a constant $c$)

# Recursive Subdivision

word size: $w$
number of integers: $n$
interval: $[0, 2^\ell)$
number of cells: $2^h$
branching width: $b$

**Lemma** **(nice $(h, b)$-subdivision)**

Given $n$ integers in $I = [0, 2^\ell)$. $I$ can be subdivided into $O(b)$ subintervals such that:

- each subinterval has length $2^{\ell-h}$ or contains at most $\frac{n}{b}$ integers
- the boundaries of the subintervals are multiples of $2^{\ell-h}$

## Recursive Decision Tree

- one child for each subinterval in the $(h, b)$-subdivision
- stop recursion if only few integers are left
- height of the tree: at most $\log_b(n) + \frac{w}{h}$

Why?

**Note:** In the recursive calls, we may need to shift the subinterval to 0 and increase it to the next power of 2, such that it has the form $[0, 2^\ell)$.

## Searching For The Successor Of An Integer $q$

$$\left.\begin{array}{l}\text{find subinterval containing } q \text{ at most } \log_b(n) + \frac{w}{h} \text{ times}\\ \text{bit magic: each step runs in } O(1) \text{ if } \frac{w}{h} \geq \frac{b}{c} \text{ (for a constant } c\text{)}\end{array}\right\} \quad O(\log_b(n) + b) \text{ for } \frac{w}{h} = \frac{b}{c}$$

# Recursive Subdivision

word size: $w$
number of integers: $n$
interval: $[0, 2^\ell)$
number of cells: $2^h$
branching width: $b$

**Lemma** **(nice $(h, b)$-subdivision)**

Given $n$ integers in $I = [0, 2^\ell)$. $I$ can be subdivided into $O(b)$ subintervals such that:

- each subinterval has length $2^{\ell-h}$ or contains at most $\frac{n}{b}$ integers
- the boundaries of the subintervals are multiples of $2^{\ell-h}$

## Recursive Decision Tree

- one child for each subinterval in the $(h, b)$-subdivision
- stop recursion if only few integers are left
- height of the tree: at most $\log_b(n) + \frac{w}{h}$

Why?

**Note:** In the recursive calls, we may need to shift the subinterval to 0 and increase it to the next power of 2, such that it has the form $[0, 2^\ell)$.

## Searching For The Successor Of An Integer $q$

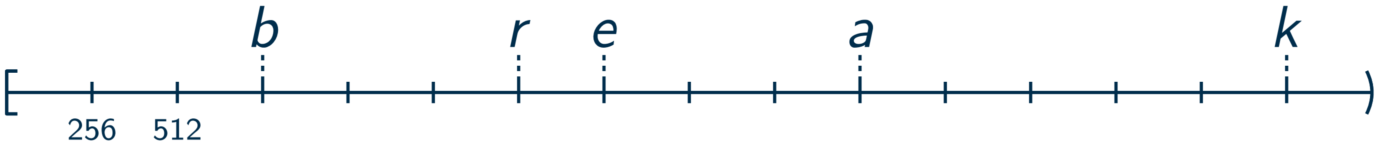
- find subinterval containing $q$ at most $\log_b(n) + \frac{w}{h}$ times
- bit magic: each step runs in $O(1)$ if $\frac{w}{h} \geq \frac{b}{c}$ (for a constant $c$)

$$\left.\right\} \begin{array}{l} O(\log_b(n) + b) \text{ for } \frac{w}{h} = \frac{b}{c} \\ O(\frac{\log n}{\log\log n}) \text{ for } b = \sqrt{\log n} \end{array}$$

# How Many Bits Do You Need?


**How many bits are necessary in total to encode the numbers $b$, $r$, $e$, $a$, $k$?**

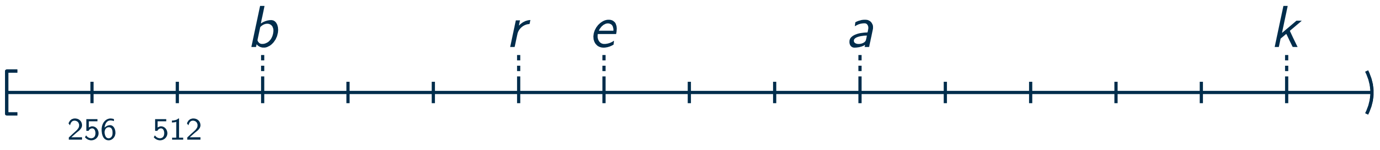
# How Many Bits Do You Need?




**How many bits are necessary in total to encode the numbers $b$, $r$, $e$, $a$, $k$?**

$$
\begin{aligned}
b &= 3 \cdot 256 \\
r &= 6 \cdot 256 \\
e &= 7 \cdot 256 \\
a &= 10 \cdot 256 \\
k &= 15 \cdot 256
\end{aligned}
$$

# How Many Bits Do You Need?




**How many bits are necessary in total to encode the numbers *b*, *r*, *e*, *a*, *k*?**

$$b = 3 \cdot 256 \longrightarrow 001100000000$$
$$r = 6 \cdot 256 \longrightarrow 011000000000$$
$$e = 7 \cdot 256 \longrightarrow 011100000000$$
$$a = 10 \cdot 256 \longrightarrow 101000000000$$
$$k = 15 \cdot 256 \longrightarrow 111100000000$$

# How Many Bits Do You Need?


**How many bits are necessary in total to encode the numbers *b*, *r*, *e*, *a*, *k*?**

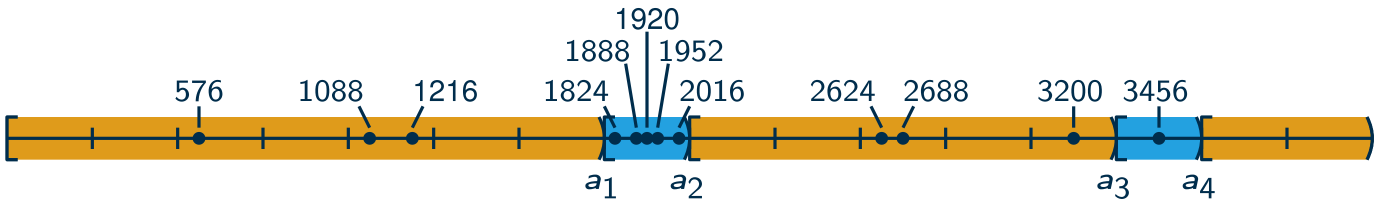
$$\left.\begin{aligned}
b &= 3 \cdot 256 \longrightarrow \boxed{0011}00000000 \\
r &= 6 \cdot 256 \longrightarrow \boxed{0110}00000000 \\
e &= 7 \cdot 256 \longrightarrow \boxed{0111}00000000 \\
a &= 10 \cdot 256 \longrightarrow \boxed{1010}00000000 \\
k &= 15 \cdot 256 \longrightarrow \boxed{1111}00000000
\end{aligned}\right\} \boxed{0011} \mid \boxed{0110} \mid \boxed{0111} \mid \boxed{1010} \mid \boxed{1111}$$

**20 bits**

# Representing The Subintervals

word size: $w$
number of integers: $n$
interval: $[0, 2^\ell)$
number of cells: $2^h$
branching width: $b$

$n = 12$
$\ell = 12$
$h = 4$
$b = 3$

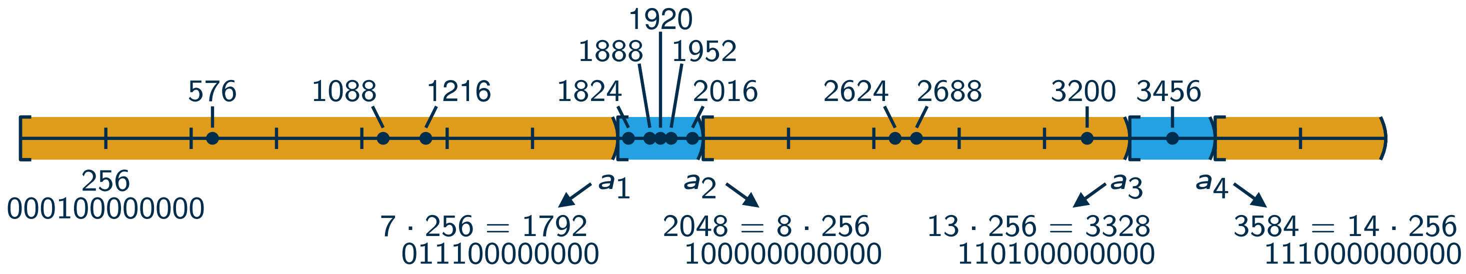



## Representing The $\bar{b}$ Interval Boundaries $a_1, \ldots, a_{\bar{b}}$

# Representing The Subintervals

word size: $w$
number of integers: $n$
interval: $[0, 2^\ell)$
number of cells: $2^h$
branching width: $b$

$n = 12$
$\ell = 12$
$h = 4$
$b = 3$



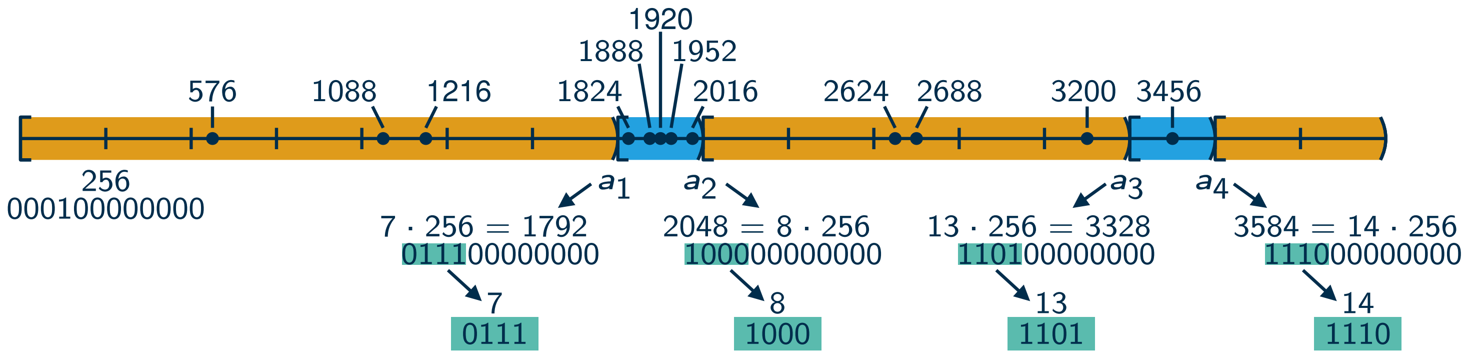

## Representing The $\bar{b}$ Interval Boundaries $a_1, \ldots, a_{\bar{b}}$

- boundaries are multiples of $2^{\ell-h}$

# Representing The Subintervals

word size: $w$
number of integers: $n$
interval: $[0, 2^\ell)$
number of cells: $2^h$
branching width: $b$

$n = 12$
$\ell = 12$
$h = 4$
$b = 3$



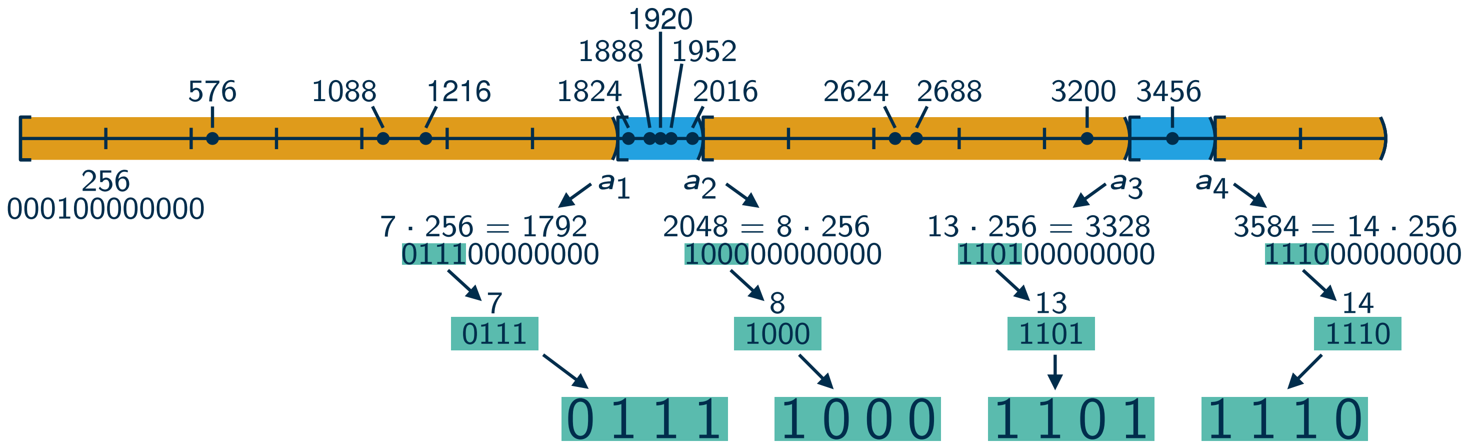

## Representing The $\bar{b}$ Interval Boundaries $a_1, \ldots, a_{\bar{b}}$

- boundaries are multiples of $2^{\ell-h}$
- divide by $2^{\ell-h}$ $\rightarrow$ result is $\leq 2^h$

# Representing The Subintervals

word size: $w$
number of integers: $n$
interval: $[0, 2^\ell)$
number of cells: $2^h$
branching width: $b$

$n = 12$
$\ell = 12$
$h = 4$
$b = 3$



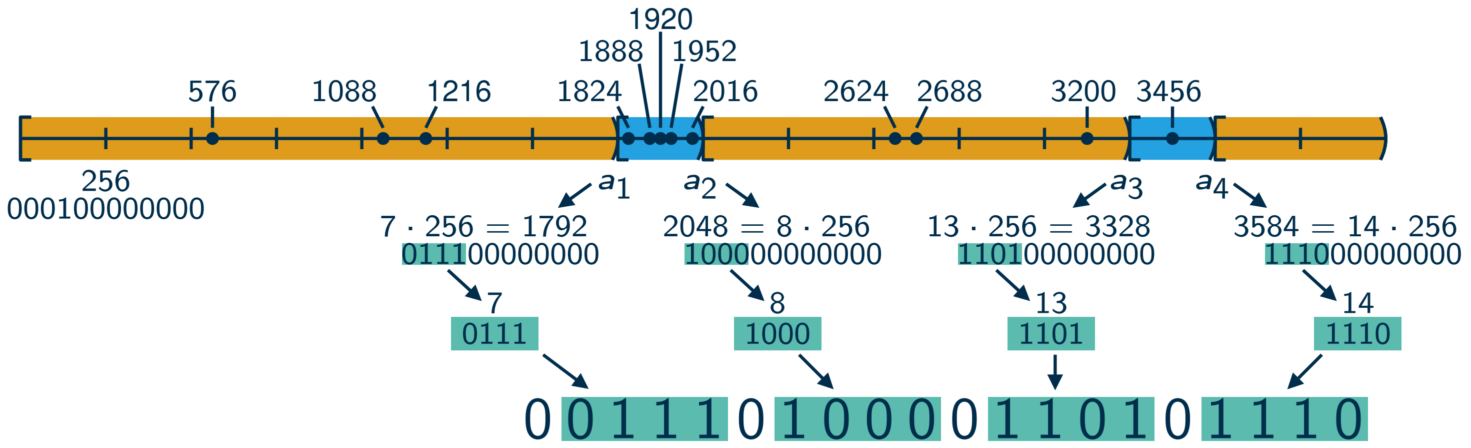

## Representing The $\bar{b}$ Interval Boundaries $a_1, \ldots, a_{\bar{b}}$

- boundaries are multiples of $2^{\ell-h}$
- divide by $2^{\ell-h}$ $\rightarrow$ result is $\leq 2^h$
- concatenate binary representation: $h \cdot \bar{b}$ bits

# Representing The Subintervals

word size: $w$
number of integers: $n$
interval: $[0, 2^\ell)$
number of cells: $2^h$
branching width: $b$

$n = 12$
$\ell = 12$
$h = 4$
$b = 3$



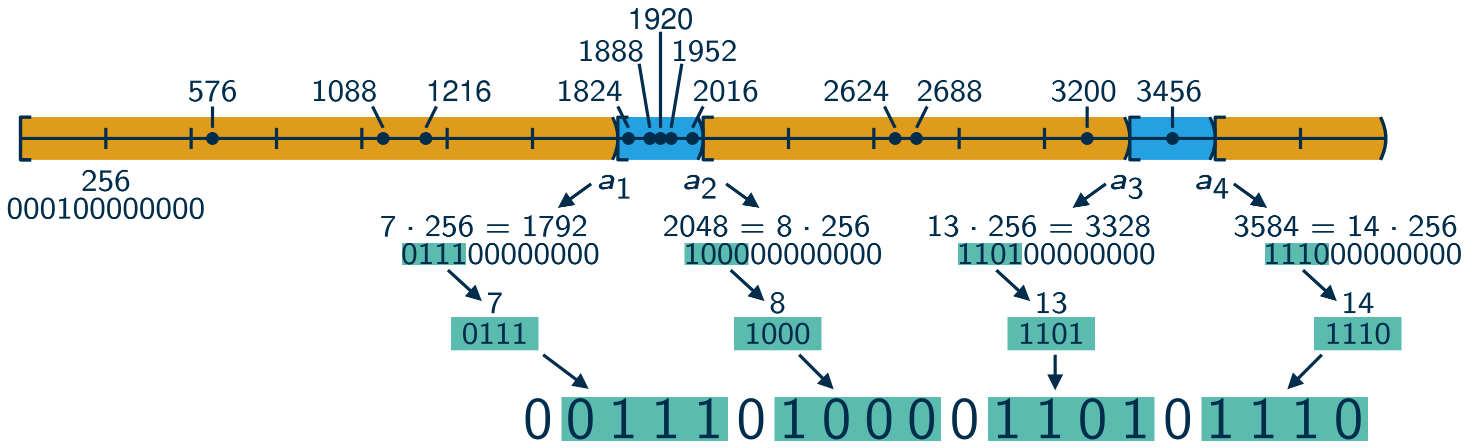

## Representing The $\bar{b}$ Interval Boundaries $a_1, \ldots, a_{\bar{b}}$

- boundaries are multiples of $2^{\ell-h}$
- divide by $2^{\ell-h} \to$ result is $\leq 2^h$
- concatenate binary representation: $h \cdot \bar{b}$ bits
- spend one extra bit per boundary $\to (h+1) \cdot \bar{b}$ bits

# Representing The Subintervals

word size: $w$
number of integers: $n$
interval: $[0, 2^\ell)$
number of cells: $2^h$
branching width: $b$

$n = 12$
$\ell = 12$
$h = 4$
$b = 3$



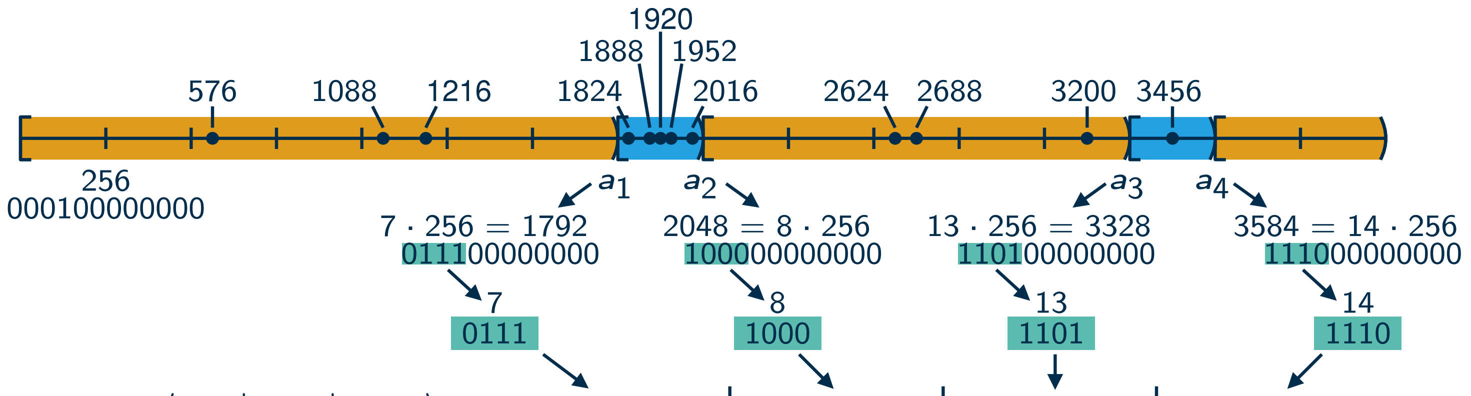

## Representing The $\bar{b}$ Interval Boundaries $a_1, \ldots, a_{\bar{b}}$

- boundaries are multiples of $2^{\ell-h}$
- divide by $2^{\ell-h} \rightarrow$ result is $\leq 2^h$
- concatenate binary representation: $h \cdot \bar{b}$ bits
- spend one extra bit per boundary $\rightarrow (h+1) \cdot \bar{b}$ bits
- assume $(h+1) \cdot \bar{b} \leq w \rightarrow$ requires just one word

(recall: we assumed things to work if $\frac{w}{h} \geq \frac{b}{c}$ for a constant $c$)

# Representing The Subintervals

word size: $w$ — $n = 12$
number of integers: $n$ — $\ell = 12$
interval: $[0, 2^\ell)$ — $h = 4$
number of cells: $2^h$ — $b = 3$
branching width: $b$




$$\langle \tilde{a}_1 \mid \tilde{a}_2 \mid \ldots \rangle = 0\,0111 \,|\, 0\,1000 \,|\, 0\,1101 \,|\, 0\,1110$$

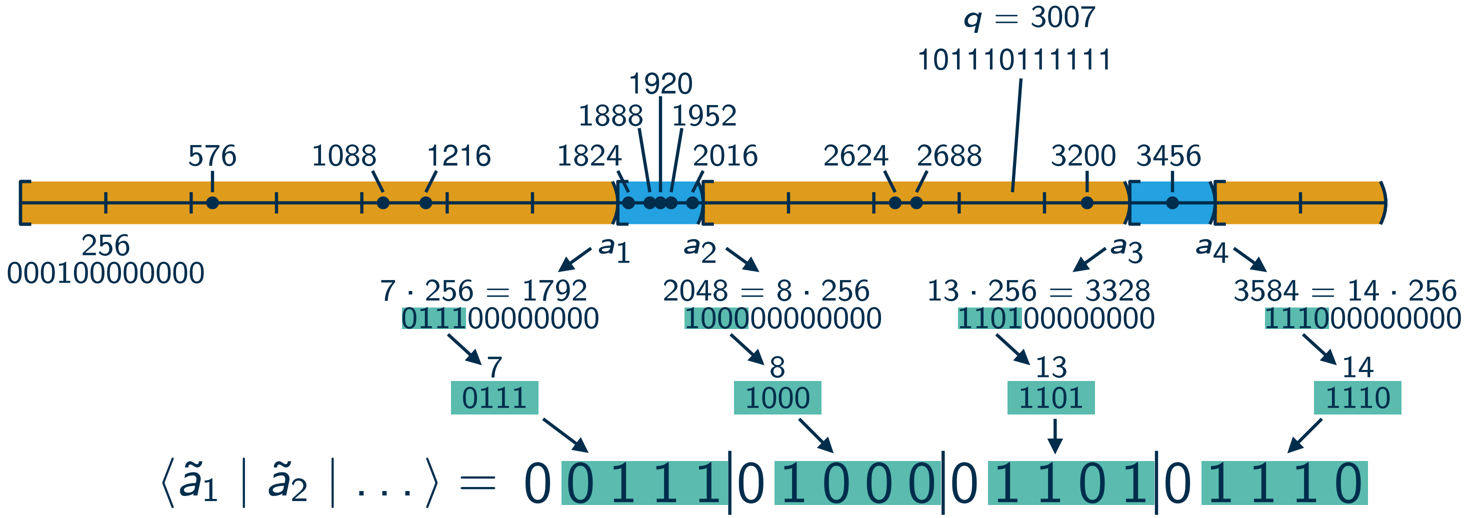
## Representing The $\bar{b}$ Interval Boundaries $a_1, \ldots, a_{\bar{b}}$

- boundaries are multiples of $2^{\ell-h}$
- divide by $2^{\ell-h} \to$ result is $\leq 2^h$
- concatenate binary representation: $h \cdot \bar{b}$ bits
- spend one extra bit per boundary $\to (h+1) \cdot \bar{b}$ bits
- assume $(h+1) \cdot \bar{b} \leq w \to$ requires just one word (recall: we assumed things to work if $\frac{w}{h} \geq \frac{b}{c}$ for a constant $c$)
- store the resulting word $\langle \tilde{a}_1 \mid \tilde{a}_2 \mid \ldots \rangle$ at the corresponding node in the recursion tree

# Finding The Right Subintervall

word size: $w$
number of integers: $n$
interval: $[0, 2^\ell)$
number of cells: $2^h$
branching width: $b$

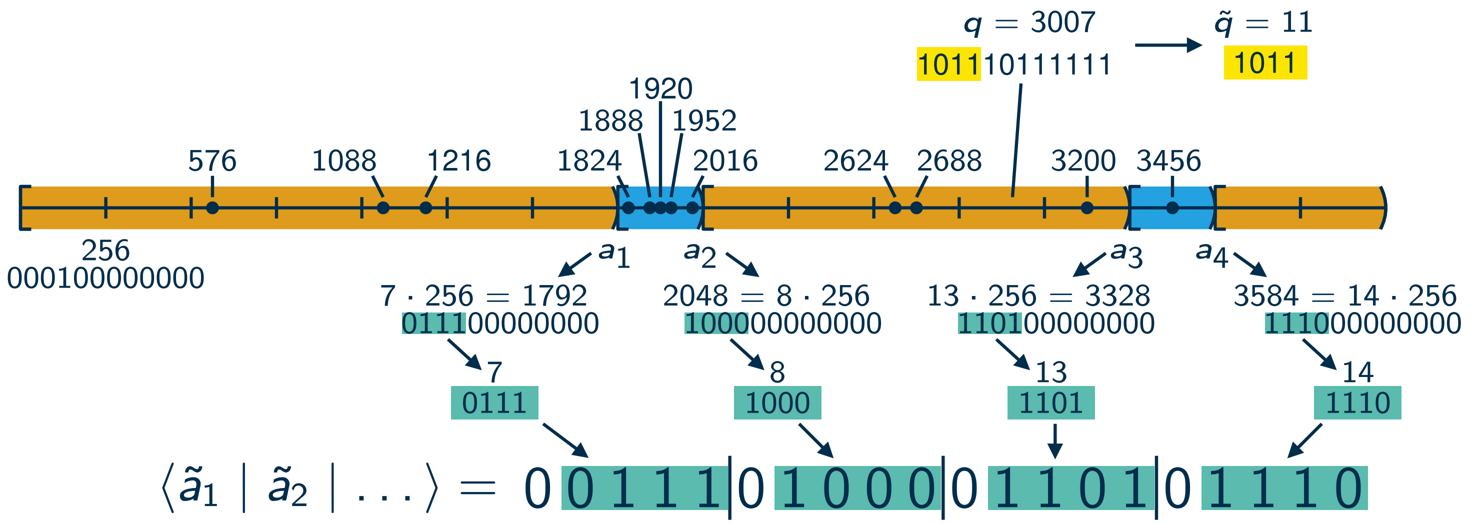
$n = 12$
$\ell = 12$
$h = 4$
$b = 3$




## Query: Which Subinterval Contains $q$?

# Finding The Right Subintervall

| | |
|---|---|
| word size: $w$ | $n = 12$ |
| number of integers: $n$ | $\ell = 12$ |
| interval: $[0, 2^\ell)$ | $h = 4$ |
| number of cells: $2^h$ | $b = 3$ |
| branching width: $b$ | |



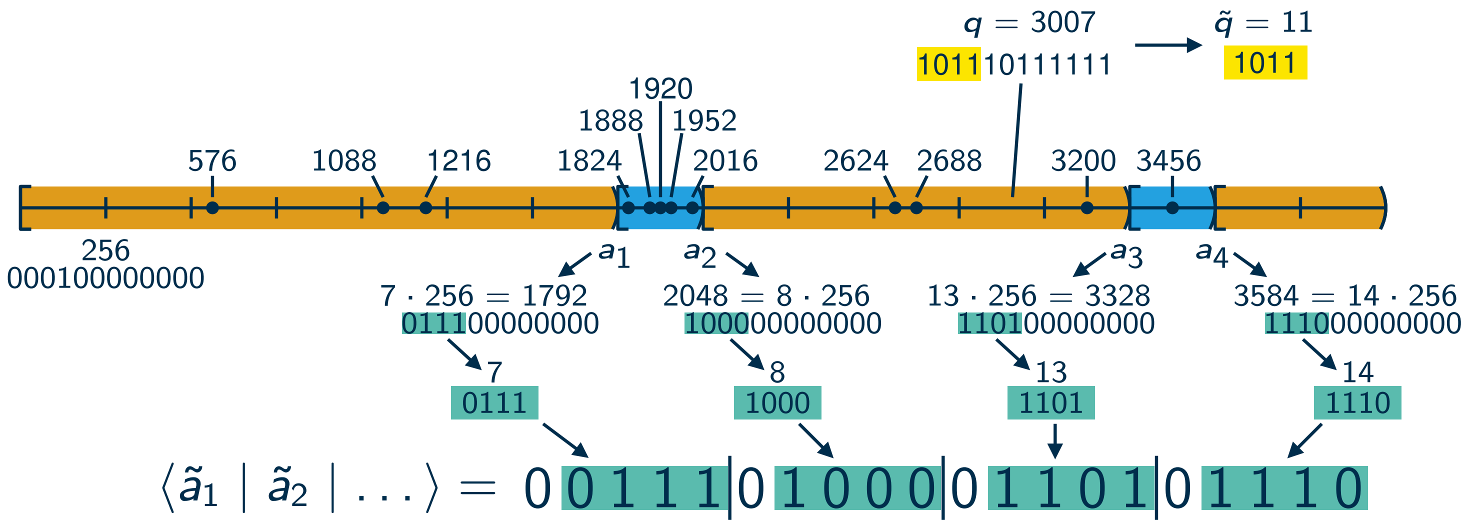

## Query: Which Subinterval Contains $q$?

- find $\tilde{q} = \lfloor q/2^{\ell-h} \rfloor$ in $\{\tilde{a}_1, \tilde{a}_2, \dots\}$ instead (division by $2^{\ell-h}$ preserves order)

# Finding The Right Subintervall

word size: $w$
number of integers: $n$
interval: $[0, 2^\ell)$
number of cells: $2^h$
branching width: $b$

$n = 12$
$\ell = 12$
$h = 4$
$b = 3$




## Query: Which Subinterval Contains $q$?

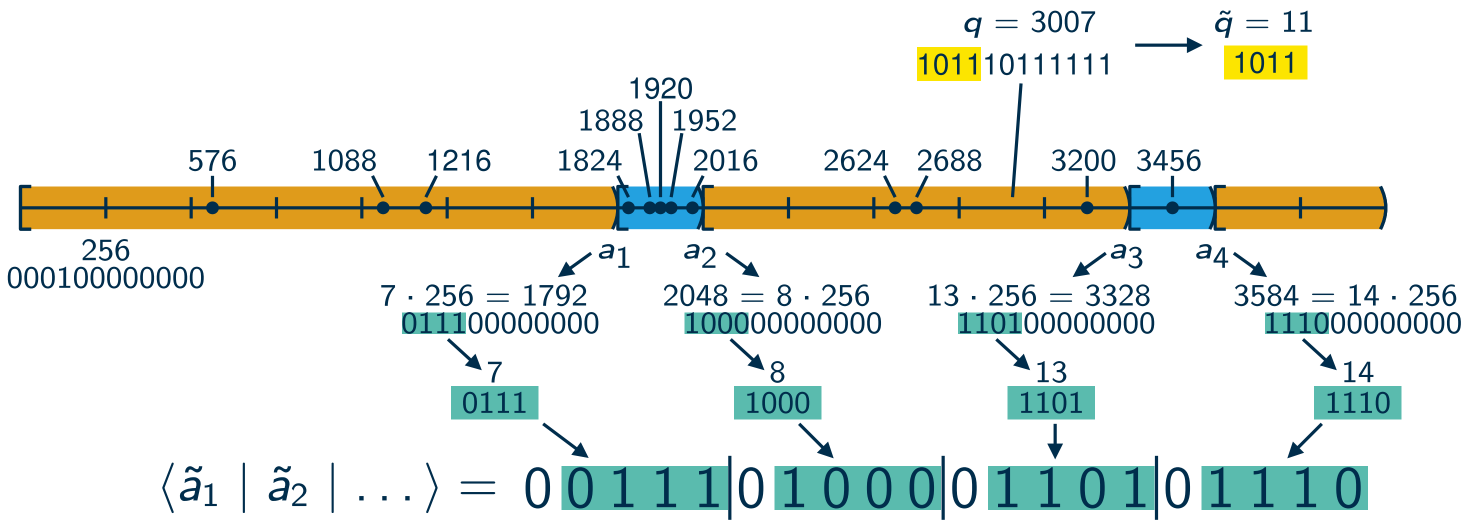
- find $\tilde{q} = \lfloor q / 2^{\ell - h} \rfloor$ in $\{\tilde{a}_1, \tilde{a}_2, \ldots\}$ instead (division by $2^{\ell-h}$ preserves order)
- it holds that: $\tilde{a}_i < \tilde{q} \Leftrightarrow (2^h + \tilde{a}_i - \tilde{q})\,\&\,2^h = 0$

Why?

# Finding The Right Subintervall

word size: $w$
number of integers: $n$
interval: $[0, 2^\ell)$
number of cells: $2^h$
branching width: $b$

$n = 12$
$\ell = 12$
$h = 4$
$b = 3$




## Query: Which Subinterval Contains $q$?

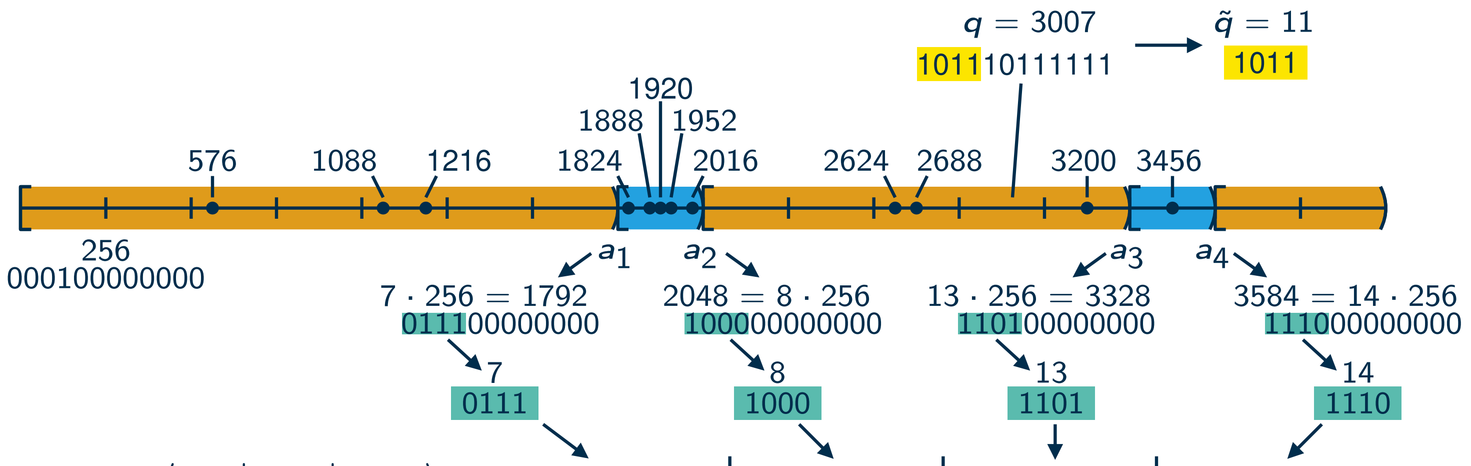
- find $\tilde{q} = \lfloor q/2^{\ell-h} \rfloor$ in $\{\tilde{a}_1, \tilde{a}_2, \ldots\}$ instead (division by $2^{\ell-h}$ preserves order)
- it holds that: $\tilde{a}_i < \tilde{q} \Leftrightarrow (2^h + \tilde{a}_i - \tilde{q})\,\&\,2^h = 0$
- compute: $(\langle 2^h \mid 2^h \mid \ldots \rangle + \langle \tilde{a}_1 \mid \tilde{a}_2 \mid \ldots \rangle - \tilde{q} \cdot \langle 1 \mid 1 \mid \ldots \rangle)\,\&\,\langle 2^h \mid 2^h \mid \ldots \rangle$

Why?

# Finding The Right Subintervall

word size: $w$
number of integers: $n$
interval: $[0, 2^\ell)$
number of cells: $2^h$
branching width: $b$

$n = 12$
$\ell = 12$
$h = 4$
$b = 3$




$$\langle 2^h \mid 2^h \mid \dots \rangle + \langle \tilde{a}_1 \mid \tilde{a}_2 \mid \dots \rangle = 1\,0\,1\,1\,1 \mid 1\,1\,0\,0\,0 \mid 1\,1\,1\,0\,1 \mid 1\,1\,1\,1\,0$$

## Query: Which Subinterval Contains $q$?

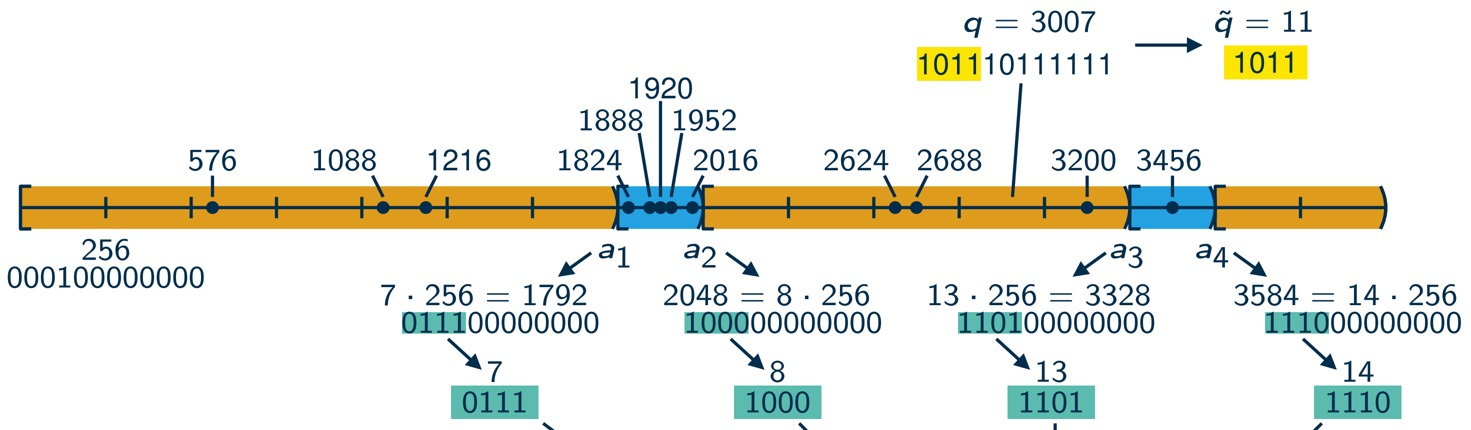
- find $\tilde{q} = \lfloor q / 2^{\ell - h} \rfloor$ in $\{\tilde{a}_1, \tilde{a}_2, \dots\}$ instead (division by $2^{\ell - h}$ preserves order)
- it holds that: $\tilde{a}_i < \tilde{q} \Leftrightarrow (2^h + \tilde{a}_i - \tilde{q})\,\&\,2^h = 0$
- compute: $(\langle 2^h \mid 2^h \mid \dots \rangle + \langle \tilde{a}_1 \mid \tilde{a}_2 \mid \dots \rangle - \tilde{q} \cdot \langle 1 \mid 1 \mid \dots \rangle)\,\&\,\langle 2^h \mid 2^h \mid \dots \rangle$

Why?

# Finding The Right Subintervall

word size: $w$
number of integers: $n$
interval: $[0, 2^\ell)$
number of cells: $2^h$
branching width: $b$

$n = 12$
$\ell = 12$
$h = 4$
$b = 3$




$$\langle 2^h \mid 2^h \mid \ldots \rangle + \langle \tilde{a}_1 \mid \tilde{a}_2 \mid \ldots \rangle = 1\,0\,1\,1\,1 \mid 1\,1\,0\,0\,0 \mid 1\,1\,1\,0\,1 \mid 1\,1\,1\,1\,0$$
$$\tilde{q} \cdot \langle 1 \mid 1 \mid \ldots \rangle = 0\,1\,0\,1\,1 \mid 0\,1\,0\,1\,1 \mid 0\,1\,0\,1\,1 \mid 0\,1\,0\,1\,1$$

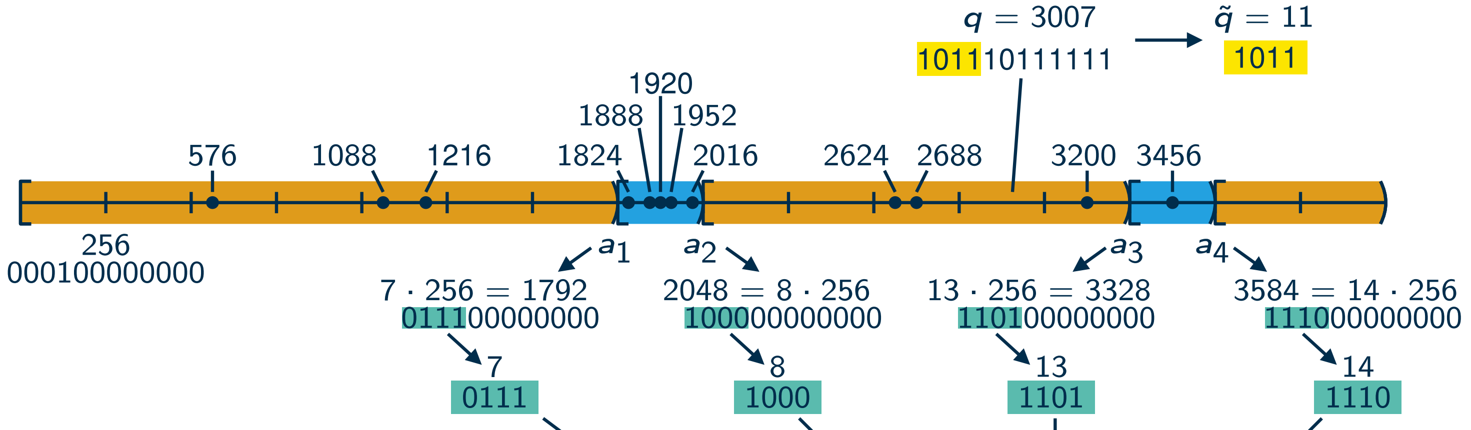
## Query: Which Subinterval Contains $q$?

- find $\tilde{q} = \lfloor q/2^{\ell-h} \rfloor$ in $\{\tilde{a}_1, \tilde{a}_2, \ldots\}$ instead (division by $2^{\ell-h}$ preserves order)
- it holds that: $\tilde{a}_i < \tilde{q} \Leftrightarrow (2^h + \tilde{a}_i - \tilde{q}) \,\&\, 2^h = 0$
- compute: $(\langle 2^h \mid 2^h \mid \ldots \rangle + \langle \tilde{a}_1 \mid \tilde{a}_2 \mid \ldots \rangle - \tilde{q} \cdot \langle 1 \mid 1 \mid \ldots \rangle) \,\&\, \langle 2^h \mid 2^h \mid \ldots \rangle$

Why?

# Finding The Right Subintervall

word size: $w$ — $n = 12$
number of integers: $n$ — $\ell = 12$
interval: $[0, 2^\ell)$ — $h = 4$
number of cells: $2^h$ — $b = 3$
branching width: $b$




$$\langle 2^h \mid 2^h \mid \ldots \rangle + \langle \tilde{a}_1 \mid \tilde{a}_2 \mid \ldots \rangle = 1\,0\,1\,1\,1 \mid 1\,1\,0\,0\,0 \mid 1\,1\,1\,0\,1 \mid 1\,1\,1\,1\,0$$
$$\tilde{q} \cdot \langle 1 \mid 1 \mid \ldots \rangle = 0\,1\,0\,1\,1 \mid 0\,1\,0\,1\,1 \mid 0\,1\,0\,1\,1 \mid 0\,1\,0\,1\,1$$
$$-$$

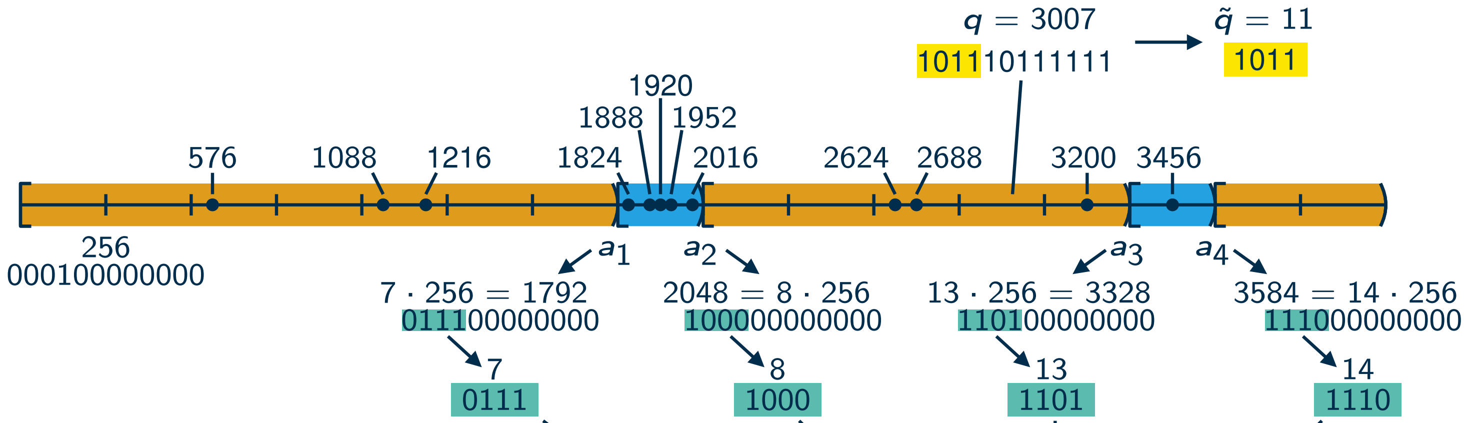
## Query: Which Subinterval Contains $q$?

- find $\tilde{q} = \lfloor q / 2^{\ell - h} \rfloor$ in $\{\tilde{a}_1, \tilde{a}_2, \ldots\}$ instead (division by $2^{\ell-h}$ preserves order)
- it holds that: $\tilde{a}_i < \tilde{q} \Leftrightarrow (2^h + \tilde{a}_i - \tilde{q}) \,\&\, 2^h = 0$
- compute: $(\langle 2^h \mid 2^h \mid \ldots \rangle + \langle \tilde{a}_1 \mid \tilde{a}_2 \mid \ldots \rangle - \tilde{q} \cdot \langle 1 \mid 1 \mid \ldots \rangle) \,\&\, \langle 2^h \mid 2^h \mid \ldots \rangle$

Why?

# Finding The Right Subintervall

word size: $w$ | $n = 12$
number of integers: $n$ | $\ell = 12$
interval: $[0, 2^\ell)$ | $h = 4$
number of cells: $2^h$ | $b = 3$
branching width: $b$




$$
\begin{array}{rl}
\langle 2^h \mid 2^h \mid \ldots \rangle + \langle \tilde{a}_1 \mid \tilde{a}_2 \mid \ldots \rangle = & 1\,0\,1\,1\,1 \mid 1\,1\,0\,0\,0 \mid 1\,1\,1\,0\,1 \mid 1\,1\,1\,1\,0 \\
\tilde{q} \cdot \langle 1 \mid 1 \mid \ldots \rangle = & 0\,1\,0\,1\,1 \mid 0\,1\,0\,1\,1 \mid 0\,1\,0\,1\,1 \mid 0\,1\,0\,1\,1 \\
\hline
- & 0\,1\,1\,0\,0 \mid 0\,1\,1\,0\,1 \mid 1\,0\,0\,1\,0 \mid 1\,0\,0\,1\,1
\end{array}
$$

## Query: Which Subinterval Contains $q$?

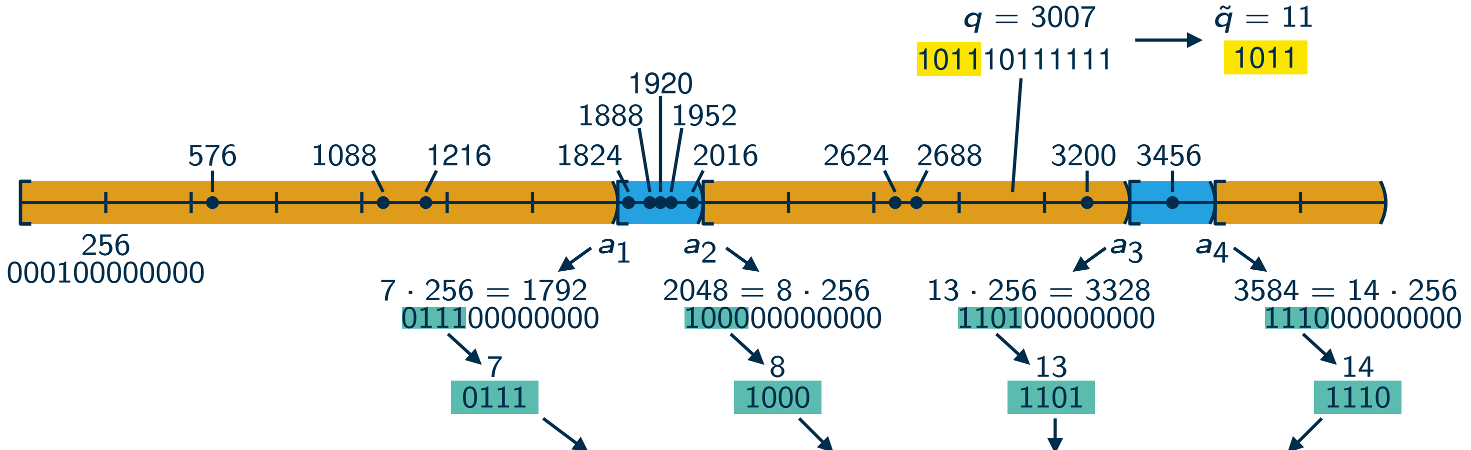
- find $\tilde{q} = \lfloor q / 2^{\ell - h} \rfloor$ in $\{\tilde{a}_1, \tilde{a}_2, \ldots\}$ instead (division by $2^{\ell-h}$ preserves order)
- it holds that: $\tilde{a}_i < \tilde{q} \Leftrightarrow (2^h + \tilde{a}_i - \tilde{q}) \mathbin{\&} 2^h = 0$
- compute: $(\langle 2^h \mid 2^h \mid \ldots \rangle + \langle \tilde{a}_1 \mid \tilde{a}_2 \mid \ldots \rangle - \tilde{q} \cdot \langle 1 \mid 1 \mid \ldots \rangle) \mathbin{\&} \langle 2^h \mid 2^h \mid \ldots \rangle$

Why?

# Finding The Right Subintervall

word size: $w$
number of integers: $n$
interval: $[0, 2^\ell)$
number of cells: $2^h$
branching width: $b$

$n = 12$
$\ell = 12$
$h = 4$
$b = 3$




$$\langle 2^h \mid 2^h \mid \ldots \rangle + \langle \tilde{a}_1 \mid \tilde{a}_2 \mid \ldots \rangle = 1\,0111 \mid 1\,1000 \mid 1\,1101 \mid 1\,1110$$
$$\tilde{q} \cdot \langle 1 \mid 1 \mid \ldots \rangle = 0\,1011 \mid 0\,1011 \mid 0\,1011 \mid 0\,1011$$
$$- \quad 0\,1100 \mid 0\,1101 \mid 1\,0010 \mid 1\,0011$$

## Query: Which Subinterval Contains $q$?

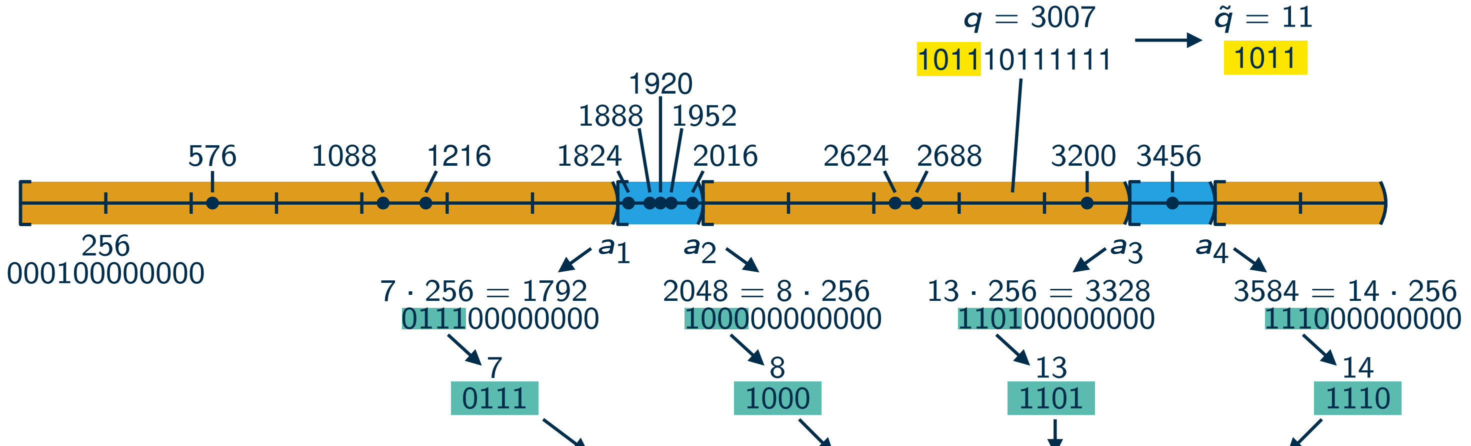
- find $\tilde{q} = \lfloor q / 2^{\ell - h} \rfloor$ in $\{\tilde{a}_1, \tilde{a}_2, \ldots\}$ instead (division by $2^{\ell-h}$ preserves order)
- it holds that: $\tilde{a}_i < \tilde{q} \Leftrightarrow (2^h + \tilde{a}_i - \tilde{q}) \,\&\, 2^h = 0$
- compute: $(\langle 2^h \mid 2^h \mid \ldots \rangle + \langle \tilde{a}_1 \mid \tilde{a}_2 \mid \ldots \rangle - \tilde{q} \cdot \langle 1 \mid 1 \mid \ldots \rangle) \,\&\, \langle 2^h \mid 2^h \mid \ldots \rangle$

Why?

# Finding The Right Subintervall

word size: $w$
number of integers: $n$
interval: $[0, 2^\ell)$
number of cells: $2^h$
branching width: $b$

$n = 12$
$\ell = 12$
$h = 4$
$b = 3$




$$\langle 2^h \mid 2^h \mid \ldots \rangle + \langle \tilde{a}_1 \mid \tilde{a}_2 \mid \ldots \rangle = 1\,0111 \mid 1\,1000 \mid 1\,1101 \mid 1\,1110$$
$$\tilde{q} \cdot \langle 1 \mid 1 \mid \ldots \rangle = 0\,1011 \mid 0\,1011 \mid 0\,1011 \mid 0\,1011$$
$$- \quad 0\,1100 \mid 0\,1101 \mid 1\,0010 \mid 1\,0011$$

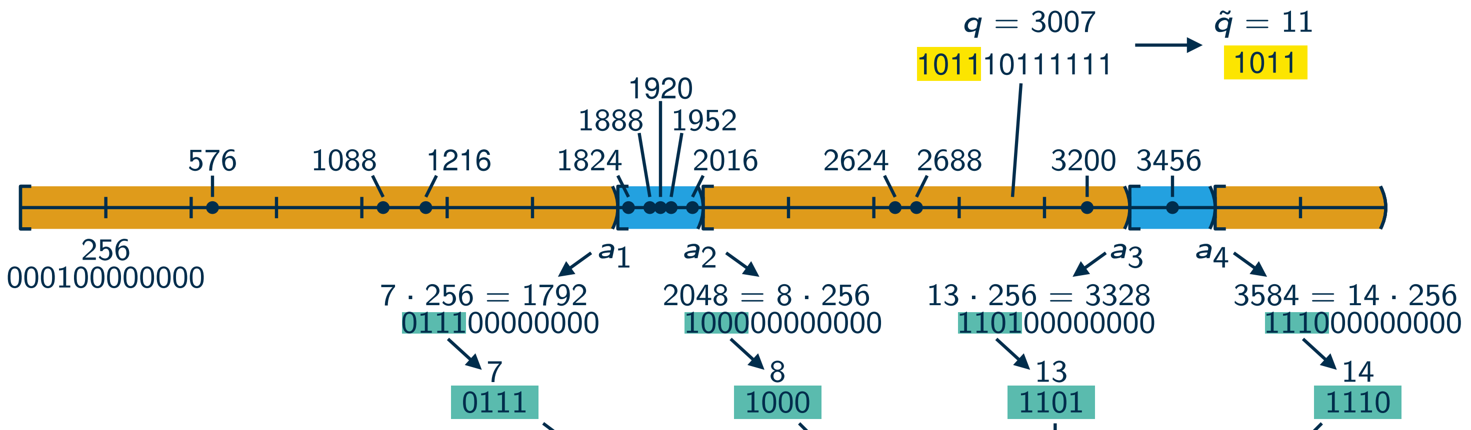
## Query: Which Subinterval Contains $q$?

- find $\tilde{q} = \lfloor q/2^{\ell-h} \rfloor$ in $\{\tilde{a}_1, \tilde{a}_2, \ldots\}$ instead (division by $2^{\ell-h}$ preserves order)
- it holds that: $\tilde{a}_i < \tilde{q} \Leftrightarrow (2^h + \tilde{a}_i - \tilde{q})\,\&\,2^h = 0$
- compute: $(\langle 2^h \mid 2^h \mid \ldots \rangle + \langle \tilde{a}_1 \mid \tilde{a}_2 \mid \ldots \rangle - \tilde{q} \cdot \langle 1 \mid 1 \mid \ldots \rangle)\,\&\,\langle 2^h \mid 2^h \mid \ldots \rangle$
- most of it can be precomputed $\to O(1)$ ($1\times$ subtraction, $1\times$ multiplication, $1\times$ bit-wise &)

Why?

# Finding The Right Subintervall

| | | | |
|---|---|---|---|
| word size: | $w$ | | $n = 12$ |
| number of integers: | $n$ | | $\ell = 12$ |
| interval: | $[0, 2^\ell)$ | | $h = 4$ |
| number of cells: | $2^h$ | | $b = 3$ |
| branching width: | $b$ | | |




$$
\begin{array}{rl}
\langle 2^h \mid 2^h \mid \dots \rangle + \langle \tilde{a}_1 \mid \tilde{a}_2 \mid \dots \rangle = & 1\,0\,1\,1\,1 \mid 1\,1\,0\,0\,0 \mid 1\,1\,1\,0\,1 \mid 1\,1\,1\,1\,0 \\
\tilde{q} \cdot \langle 1 \mid 1 \mid \dots \rangle = & 0\,1\,0\,1\,1 \mid 0\,1\,0\,1\,1 \mid 0\,1\,0\,1\,1 \mid 0\,1\,0\,1\,1 \\
\hline
- & 0\,1\,1\,0\,0 \mid 0\,1\,1\,0\,1 \mid 1\,0\,0\,1\,0 \mid 1\,0\,0\,1\,1
\end{array}
$$

## Query: Which Subinterval Contains $q$?

- find $\tilde{q} = \lfloor q / 2^{\ell - h} \rfloor$ in $\{\tilde{a}_1, \tilde{a}_2, \dots\}$ instead (division by $2^{\ell - h}$ preserves order)
- it holds that: $\tilde{a}_i < \tilde{q} \Leftrightarrow (2^h + \tilde{a}_i - \tilde{q}) \,\&\, 2^h = 0$
- compute: $(\langle 2^h \mid 2^h \mid \dots \rangle + \langle \tilde{a}_1 \mid \tilde{a}_2 \mid \dots \rangle - \tilde{q} \cdot \langle 1 \mid 1 \mid \dots \rangle) \,\&\, \langle 2^h \mid 2^h \mid \dots \rangle$
- most of it can be precomputed $\to O(1)$ ($1\times$ subtraction, $1\times$ multiplication, $1\times$ bit-wise &)
- the most significant 1-bit yields $i$, such that $a_{i-1} < q \leq a_i$ (MS1B computable with $O(1)$ elementary operations)

Why?

# Recap

word size: $w$
number of integers: $n$
interval: $[0, 2^\ell)$
number of cells: $2^h$
branching width: $b$

**Lemma** **(nice $(h, b)$-subdivision)**
Given $n$ integers in $I = [0, 2^\ell)$. $I$ can be subdivided into $O(b)$ subintervals such that:
- each subinterval has length $2^{\ell-h}$ or contains at most $\frac{n}{b}$ integers
- the boundaries of the subintervals are multiples of $2^{\ell-h}$

## Recursive Decision Tree
- one child for each subinterval in the $(h, b)$-subdivision
- stop recursion if only few integers are left
- height of the tree: at most $\log_b(n) + \frac{w}{h}$

## Searching For The Successor Of An Integer $q$
- find subinterval containing $q$ at most $\log_b(n) + \frac{w}{h}$ times
- bit magic: each step runs in $O(1)$ if $\frac{w}{h} \geq \frac{b}{c}$ (for a constant $c$)

$\Big\}$ $O(\log_b(n) + b)$ for $\frac{w}{h} = \frac{b}{c}$
$O(\frac{\log n}{\log \log n})$ for $b = \sqrt{\log n}$

# 1D Search $\to$ 2D Search

## 1D: In Each Node Of The Searchtree

- $O(b)$ boundaries, each a multiple of $2^{\ell-h}$
- subintervals: small ($2^{\ell-h}$) or few integers ($\frac{n}{b}$)

## 1D: Query

- search neighboring boundary
- continue in corresponding subtree

# 1D Search $\rightarrow$ 2D Search

## 1D: In Each Node Of The Searchtree

- $O(b)$ boundaries, each a multiple of $2^{\ell-h}$
- subintervals: small ($2^{\ell-h}$) or few integers ($\frac{n}{b}$)

## 1D: Query

- search neighboring boundary
- continue in corresponding subtree

## Why Does This Help?

sufficient progress in each node

happens $\leq \log_b(n) + \frac{w}{h}$ times

# 1D Search $\rightarrow$ 2D Search

| **1D: In Each Node Of The Searchtree** | **Why Does This Help?** |
|---|---|
| ■ $O(b)$ boundaries, each a multiple of $2^{\ell-h}$ | $h$ bits per boundary $\rightarrow$ fit in just one word |
| ■ subintervals: small ($2^{\ell-h}$) or few integers ($\frac{n}{b}$) | sufficient progress in each node |
| **1D: Query** | |
| ■ search neighboring boundary | works in $O(1)$ |
| ■ continue in corresponding subtree | happens $\leq \log_b(n) + \frac{w}{h}$ times |

# 1D Search → 2D Search

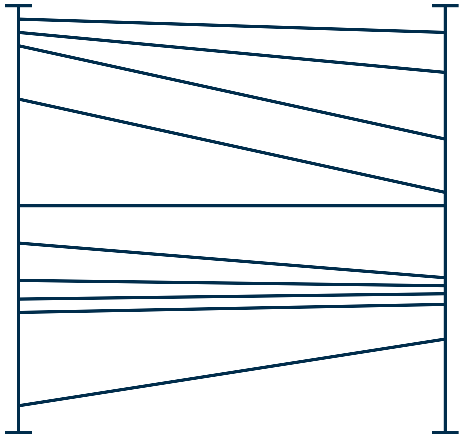
**1D: In Each Node Of The Searchtree** | **Why Does This Help?**

- $O(b)$ boundaries, each a multiple of $2^{\ell-h}$ — $h$ bits per boundary → fit in just one word
- subintervals: small ($2^{\ell-h}$) or few integers ($\frac{n}{b}$) — sufficient progress in each node

**1D: Query**

- search neighboring boundary — works in $O(1)$
- continue in corresponding subtree — happens $\leq \log_b(n) + \frac{w}{h}$ times

**Extension To Point Location In A Slab**



# 1D Search $\rightarrow$ 2D Search

**1D: In Each Node Of The Searchtree** | **Why Does This Help?**

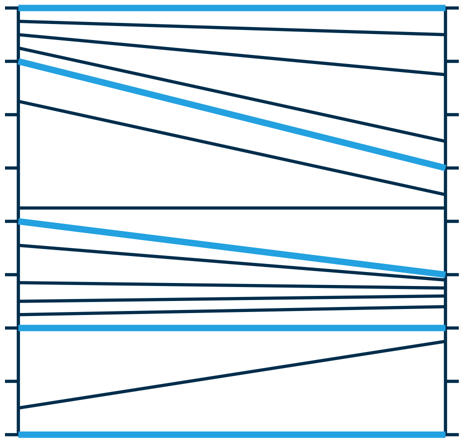
- $O(b)$ boundaries, each a multiple of $2^{\ell-h}$ — $h$ bits per boundary $\rightarrow$ fit in just one word
- subintervals: small ($2^{\ell-h}$) or few integers ($\frac{n}{b}$) — sufficient progress in each node

**1D: Query**

- search neighboring boundary — works in $O(1)$
- continue in corresponding subtree — happens $\leq \log_b(n) + \frac{w}{h}$ times

**Extension To Point Location In A Slab**

- choose boundaries with similar properties



# 1D Search $\rightarrow$ 2D Search

### 1D: In Each Node Of The Searchtree

### Why Does This Help?

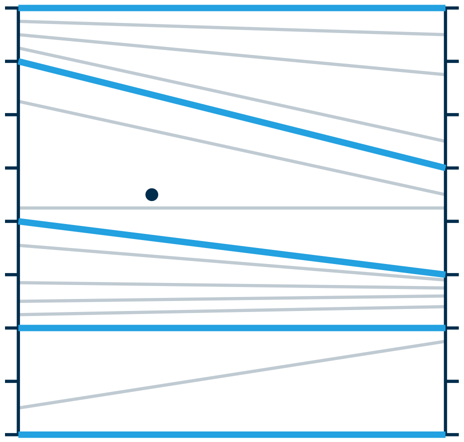
- $O(b)$ boundaries, each a multiple of $2^{\ell-h}$ — $h$ bits per boundary $\rightarrow$ fit in just one word
- subintervals: small $(2^{\ell-h})$ or few integers $(\frac{n}{b})$ — sufficient progress in each node

### 1D: Query

- search neighboring boundary — works in $O(1)$
- continue in corresponding subtree — happens $\leq \log_b(n) + \frac{w}{h}$ times

### Extension To Point Location In A Slab

- choose boundaries with similar properties
- query a point $(x, y)$



# 1D Search $\rightarrow$ 2D Search

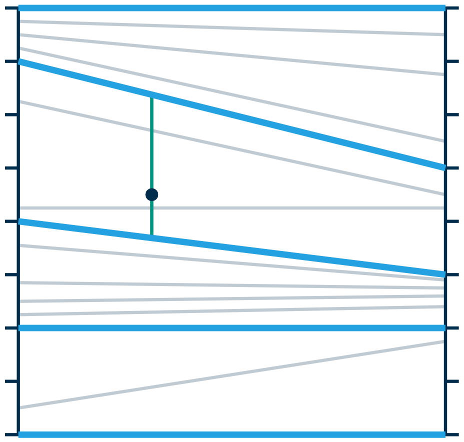
| **1D: In Each Node Of The Searchtree** | **Why Does This Help?** |
|---|---|
| ■ $O(b)$ boundaries, each a multiple of $2^{\ell-h}$ | $h$ bits per boundary $\rightarrow$ fit in just one word |
| ■ subintervals: small ($2^{\ell-h}$) or few integers ($\frac{n}{b}$) | sufficient progress in each node |

**1D: Query**

| | |
|---|---|
| ■ search neighboring boundary | works in $O(1)$ |
| ■ continue in corresponding subtree | happens $\leq \log_b(n) + \frac{w}{h}$ times |

**Extension To Point Location In A Slab**

- choose boundaries with similar properties
- query a point $(x, y)$
  - 1D search with respect to $y$ at the position $x$



# 1D Search $\rightarrow$ 2D Search

**1D: In Each Node Of The Searchtree**

**Why Does This Help?**

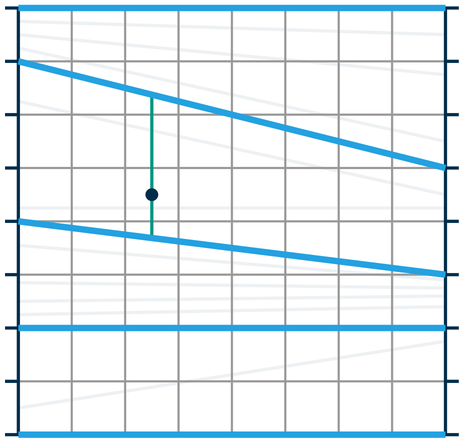
- $O(b)$ boundaries, each a multiple of $2^{\ell-h}$ — $h$ bits per boundary $\rightarrow$ fit in just one word
- subintervals: small ($2^{\ell-h}$) or few integers ($\frac{n}{b}$) — sufficient progress in each node

**1D: Query**

- search neighboring boundary — works in $O(1)$
- continue in corresponding subtree — happens $\leq \log_b(n) + \frac{w}{h}$ times

**Extension To Point Location In A Slab**

- choose boundaries with similar properties
- query a point $(x, y)$
  - 1D search with respect to $y$ at the position $x$
  - round all coordinates $\rightarrow$ $h$ bits per coordinate



# 1D Search $\to$ 2D Search

### 1D: In Each Node Of The Searchtree

- $O(b)$ boundaries, each a multiple of $2^{\ell-h}$
- subintervals: small ($2^{\ell-h}$) or few integers ($\frac{n}{b}$)

### Why Does This Help?

$h$ bits per boundary $\to$ fit in just one word

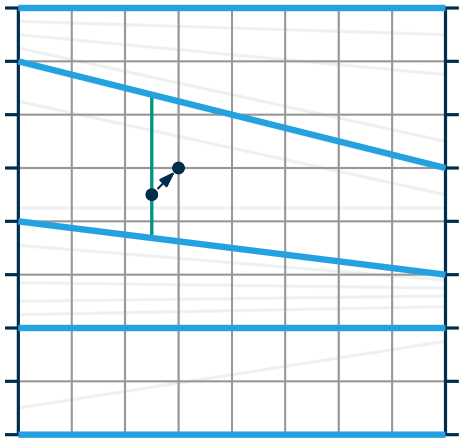
sufficient progress in each node

### 1D: Query

- search neighboring boundary — works in $O(1)$
- continue in corresponding subtree — happens $\leq \log_b(n) + \frac{w}{h}$ times

### Extension To Point Location In A Slab

- choose boundaries with similar properties
- query a point $(x, y)$
  - 1D search with respect to $y$ at the position $x$
  - round all coordinates $\to$ $h$ bits per coordinate



# 1D Search $\rightarrow$ 2D Search

**1D: In Each Node Of The Searchtree** | **Why Does This Help?**

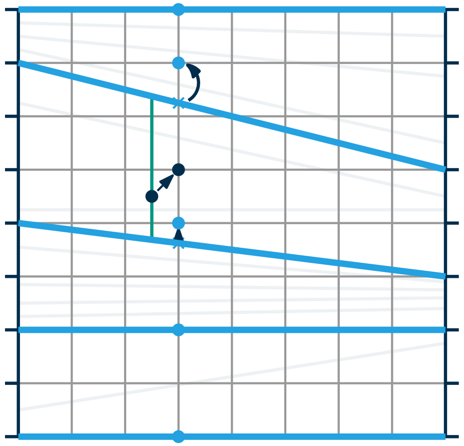
- $O(b)$ boundaries, each a multiple of $2^{\ell-h}$ — $h$ bits per boundary $\rightarrow$ fit in just one word
- subintervals: small ($2^{\ell-h}$) or few integers ($\frac{n}{b}$) — sufficient progress in each node

**1D: Query**

- search neighboring boundary — works in $O(1)$
- continue in corresponding subtree — happens $\leq \log_b(n) + \frac{w}{h}$ times

**Extension To Point Location In A Slab**

- choose boundaries with similar properties
- query a point $(x, y)$
  - 1D search with respect to $y$ at the position $x$
  - round all coordinates $\rightarrow$ $h$ bits per coordinate



# 1D Search $\rightarrow$ 2D Search

### 1D: In Each Node Of The Searchtree

- $O(b)$ boundaries, each a multiple of $2^{\ell-h}$
- subintervals: small ($2^{\ell-h}$) or few integers ($\frac{n}{b}$)

### Why Does This Help?

$h$ bits per boundary $\rightarrow$ fit in just one word

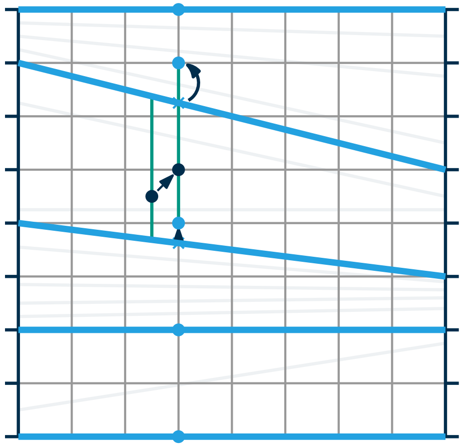
sufficient progress in each node

### 1D: Query

- search neighboring boundary — works in $O(1)$
- continue in corresponding subtree — happens $\leq \log_b(n) + \frac{w}{h}$ times

### Extension To Point Location In A Slab

- choose boundaries with similar properties
- query a point $(x, y)$
  - 1D search with respect to $y$ at the position $x$
  - round all coordinates $\rightarrow$ $h$ bits per coordinate



# 1D Search $\to$ 2D Search

## 1D: In Each Node Of The Searchtree

- $O(b)$ boundaries, each a multiple of $2^{\ell-h}$
- subintervals: small ($2^{\ell-h}$) or few integers ($\frac{n}{b}$)

## Why Does This Help?

$h$ bits per boundary $\to$ fit in just one word

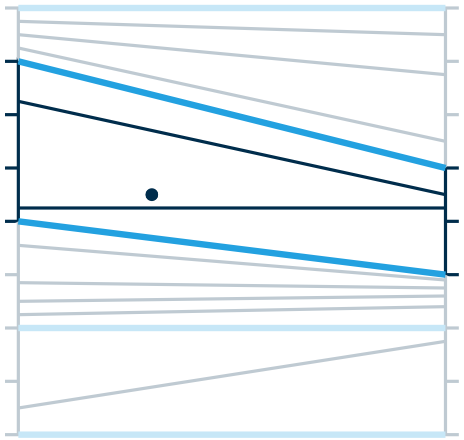
sufficient progress in each node

## 1D: Query

- search neighboring boundary — works in $O(1)$
- continue in corresponding subtree — happens $\leq \log_b(n) + \frac{w}{h}$ times

## Extension To Point Location In A Slab

- choose boundaries with similar properties
- query a point $(x, y)$
  - 1D search with respect to $y$ at the position $x$
  - round all coordinates $\to$ $h$ bits per coordinate
- recursive call between boundaries



# 1D Search $\to$ 2D Search

**1D: In Each Node Of The Searchtree**

**Why Does This Help?**

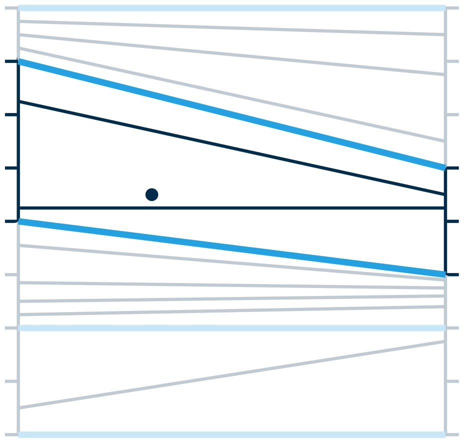
- $O(b)$ boundaries, each a multiple of $2^{\ell - h}$ — $h$ bits per boundary $\to$ fit in just one word
- subintervals: small ($2^{\ell - h}$) or few integers ($\frac{n}{b}$) — sufficient progress in each node

**1D: Query**

- search neighboring boundary — works in $O(1)$
- continue in corresponding subtree — happens $\leq \log_b(n) + \frac{w}{h}$ times

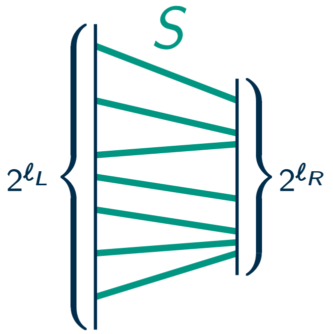
**Extension To Point Location In A Slab**

- choose boundaries with similar properties
- query a point $(x, y)$
  - 1D search with respect to $y$ at the position $x$
  - round all coordinates $\to$ $h$ bits per coordinate
- recursive call between boundaries



**Problem:** What if boundaries intersect the segments?

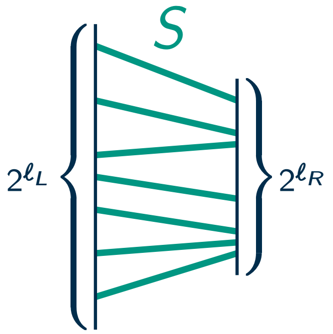
# Nice Subdivision




**Input**

- two vertical line segments that have length $2^{\ell_L}$ and $2^{\ell_R}$
- $n$ disjoint segments $S$ between the two vertical segments

# Nice Subdivision



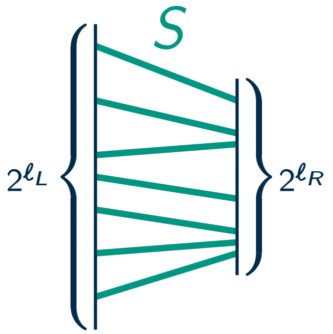

**Lemma** **(nice $(h, b)$-subdivision)**
There are $O(b)$ boundary segments $s_0, s_1, \dots \in S$ such that

- between $s_i$ and $s_{i+1}$ there are $\leq \frac{n}{b}$ segments from $S$ or $y_L(s_{i+1}) - y_L(s_i) < 2^{\ell_L - h}$ or $y_R(s_{i+1}) - y_R(s_i) < 2^{\ell_R - h}$
- $\tilde{s}_0 \prec s_0 \prec \tilde{s}_2 \prec s_2 \prec \cdots$ for "rounded" boundaries $\tilde{s}_i$

**Input**

- two vertical line segments that have length $2^{\ell_L}$ and $2^{\ell_R}$
- $n$ disjoint segments $S$ between the two vertical segments

# Nice Subdivision




**Lemma** **(nice $(h, b)$-subdivision)**
There are $O(b)$ boundary segments $s_0, s_1, \dots \in S$ such that

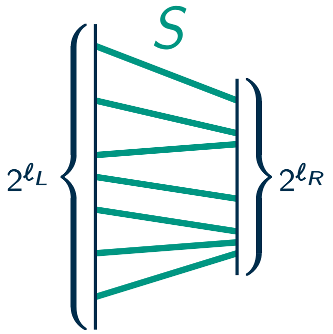
- between $s_i$ and $s_{i+1}$ there are $\leq \frac{n}{b}$ segments from $S$ or $y_L(s_{i+1}) - y_L(s_i) < 2^{\ell_L - h}$ or $y_R(s_{i+1}) - y_R(s_i) < 2^{\ell_R - h}$
- $\tilde{s}_0 \prec s_0 \prec \tilde{s}_2 \prec s_2 \prec \cdots$ for "rounded" boundaries $\tilde{s}_i$

**Proof:** exercise

**Input**

- two vertical line segments that have length $2^{\ell_L}$ and $2^{\ell_R}$
- $n$ disjoint segments $S$ between the two vertical segments

# Nice Subdivision




**Lemma** **(nice $(h, b)$-subdivision)**
There are $O(b)$ boundary segments $s_0, s_1, \dots \in S$ such that

- between $s_i$ and $s_{i+1}$ there are $\leq \frac{n}{b}$ segments from $S$ or $y_L(s_{i+1}) - y_L(s_i) < 2^{\ell_L - h}$ or $y_R(s_{i+1}) - y_R(s_i) < 2^{\ell_R - h}$
- $\tilde{s}_0 \prec s_0 \prec \tilde{s}_2 \prec s_2 \prec \cdots$ for "rounded" boundaries $\tilde{s}_i$

**Proof:** exercise

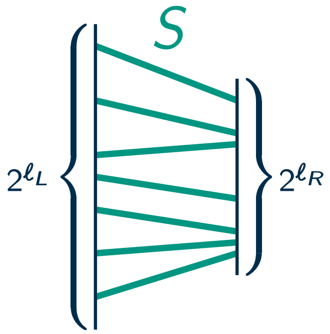
**Input**

- two vertical line segments that have length $2^{\ell_L}$ and $2^{\ell_R}$
- $n$ disjoint segments $S$ between the two vertical segments

**Notes On Running The Query For A Point $q = (x, y)$**

- left/right endpoints of all $\tilde{s}_i$ can be packed into just one word

# Nice Subdivision



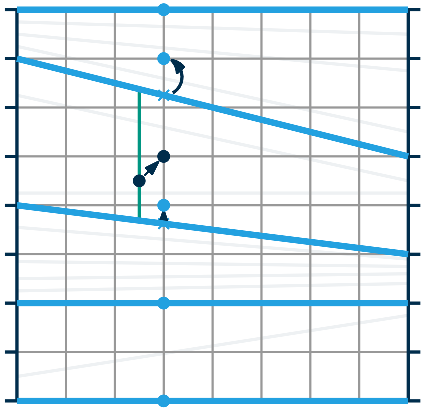

**Lemma** **(nice $(h, b)$-subdivision)**
There are $O(b)$ boundary segments $s_0, s_1, \dots \in S$ such that

- between $s_i$ and $s_{i+1}$ there are $\leq \frac{n}{b}$ segments from $S$ or $y_L(s_{i+1}) - y_L(s_i) < 2^{\ell_L - h}$ or $y_R(s_{i+1}) - y_R(s_i) < 2^{\ell_R - h}$
- $\tilde{s}_0 \prec s_0 \prec \tilde{s}_2 \prec s_2 \prec \cdots$ for "rounded" boundaries $\tilde{s}_i$

**Proof:** exercise

**Input**

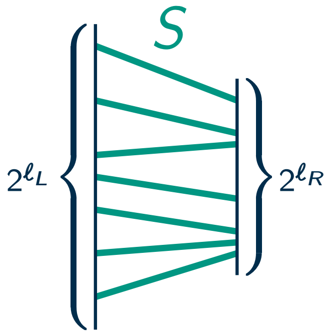
- two vertical line segments that have length $2^{\ell_L}$ and $2^{\ell_R}$
- $n$ disjoint segments $S$ between the two vertical segments



**Notes On Running The Query For A Point $q = (x, y)$**

- left/right endpoints of all $\tilde{s}_i$ can be packed into just one word
- the (rounded) intersections of all $\tilde{s}_i$ with a vertical line at $\tilde{x}$ are computable with $O(1)$ operations

# Nice Subdivision




**Lemma** **(nice $(h, b)$-subdivision)**

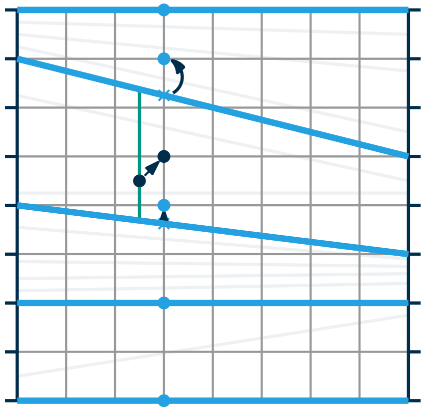
There are $O(b)$ boundary segments $s_0, s_1, \dots \in S$ such that

- between $s_i$ and $s_{i+1}$ there are $\leq \frac{n}{b}$ segments from $S$ or $y_L(s_{i+1}) - y_L(s_i) < 2^{\ell_L - h}$ or $y_R(s_{i+1}) - y_R(s_i) < 2^{\ell_R - h}$
- $\tilde{s}_0 \prec s_0 \prec \tilde{s}_2 \prec s_2 \prec \cdots$ for "rounded" boundaries $\tilde{s}_i$

**Proof:** exercise

**Input**

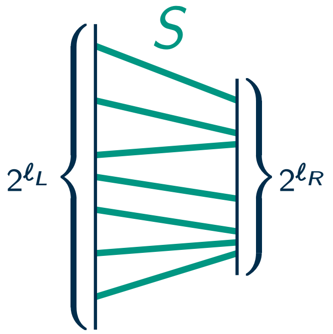
- two vertical line segments that have length $2^{\ell_L}$ and $2^{\ell_R}$
- $n$ disjoint segments $S$ between the two vertical segments



## Notes On Running The Query For A Point $q = (x, y)$

- left/right endpoints of all $\tilde{s}_i$ can be packed into just one word
- the (rounded) intersections of all $\tilde{s}_i$ with a vertical line at $\tilde{x}$ are computable with $O(1)$ operations
- searching on the rounded intersections for $\tilde{y}$ as in the 1D case

# Nice Subdivision



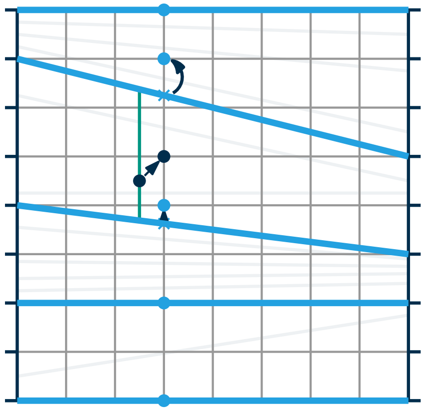

**Lemma** **(nice $(h, b)$-subdivision)**
There are $O(b)$ boundary segments $s_0, s_1, \dots \in S$ such that

- between $s_i$ and $s_{i+1}$ there are $\leq \frac{n}{b}$ segments from $S$ or $y_L(s_{i+1}) - y_L(s_i) < 2^{\ell_L - h}$ or $y_R(s_{i+1}) - y_R(s_i) < 2^{\ell_R - h}$
- $\tilde{s}_0 \prec s_0 \prec \tilde{s}_2 \prec s_2 \prec \cdots$ for "rounded" boundaries $\tilde{s}_i$

**Proof:** exercise

**Input**

- two vertical line segments that have length $2^{\ell_L}$ and $2^{\ell_R}$
- $n$ disjoint segments $S$ between the two vertical segments



## Notes On Running The Query For A Point $q = (x, y)$

- left/right endpoints of all $\tilde{s}_i$ can be packed into just one word
- the (rounded) intersections of all $\tilde{s}_i$ with a vertical line at $\tilde{x}$ are computable with $O(1)$ operations
- searching on the rounded intersections for $\tilde{y}$ as in the 1D case
- un-rounding: position of $\tilde{q}$ in the $\tilde{s}_i \rightarrow$ position of $q$ in the $s_i$ (only $O(1)$ additional comparisons)

# Wrap-Up

## Seen Today

- models of computation: word RAM and real RAM

# Wrap-Up

## Seen Today

- models of computation: word RAM and real RAM
- real RAM often useful for computational geometry (but high precision sometimes unrealistic)

# Wrap-Up

## Seen Today

- models of computation: word RAM and real RAM
- real RAM often useful for computational geometry (but high precision sometimes unrealistic)
- bounded precision of the word RAM can be useful:
  $\rightarrow$ search and point location in vertical slab in $O(\frac{\log n}{\log \log n})$

# Wrap-Up

## Seen Today

- models of computation: word RAM and real RAM
- real RAM often useful for computational geometry (but high precision sometimes unrealistic)
- bounded precision of the word RAM can be useful:
  $\rightarrow$ search and point location in vertical slab in $O(\frac{\log n}{\log \log n})$

## What Else Is There?

- general point location with $o(\log n)$ queries in the word RAM

# Wrap-Up

### Seen Today

- models of computation: word RAM and real RAM
- real RAM often useful for computational geometry (but high precision sometimes unrealistic)
- bounded precision of the word RAM can be useful:
  $\rightarrow$ search and point location in vertical slab in $O(\frac{\log n}{\log \log n})$

### What Else Is There?

- general point location with $o(\log n)$ queries in the word RAM
- problems with $o(n \log n)$ solutions on the word RAM:
  - 3D convex hull

# Wrap-Up

## Seen Today

- models of computation: word RAM and real RAM
- real RAM often useful for computational geometry (but high precision sometimes unrealistic)
- bounded precision of the word RAM can be useful:
  $\rightarrow$ search and point location in vertical slab in $O(\frac{\log n}{\log \log n})$

## What Else Is There?

- general point location with $o(\log n)$ queries in the word RAM
- problems with $o(n \log n)$ solutions on the word RAM:
  - 3D convex hull
  - Voronoi diagram

# Wrap-Up

## Seen Today

- models of computation: word RAM and real RAM
- real RAM often useful for computational geometry (but high precision sometimes unrealistic)
- bounded precision of the word RAM can be useful:
  $\rightarrow$ search and point location in vertical slab in $O(\frac{\log n}{\log \log n})$

## What Else Is There?

- general point location with $o(\log n)$ queries in the word RAM
- problems with $o(n \log n)$ solutions on the word RAM:
  - 3D convex hull
  - Voronoi diagram
  - Euclidean MST

# Wrap-Up

## Seen Today

- models of computation: word RAM and real RAM
- real RAM often useful for computational geometry (but high precision sometimes unrealistic)
- bounded precision of the word RAM can be useful:
  $\rightarrow$ search and point location in vertical slab in $O(\frac{\log n}{\log \log n})$

## What Else Is There?

- general point location with $o(\log n)$ queries in the word RAM
- problems with $o(n \log n)$ solutions on the word RAM:
  - 3D convex hull
  - Voronoi diagram
  - Euclidean MST
  - triangulation of polygons

# Wrap-Up

## Seen Today

- models of computation: word RAM and real RAM
- real RAM often useful for computational geometry (but high precision sometimes unrealistic)
- bounded precision of the word RAM can be useful:
  $\rightarrow$ search and point location in vertical slab in $O(\frac{\log n}{\log \log n})$

## What Else Is There?

- general point location with $o(\log n)$ queries in the word RAM
- problems with $o(n \log n)$ solutions on the word RAM:
  - 3D convex hull
  - Voronoi diagram
  - Euclidean MST
  - triangulation of polygons
  - line segment intersection

# Literature

- **Transdichotomous Results in Computational Geometry, I: Point Location in Sublogarithmic Time** (2009)
  Timothy Chan, Mihai Pătraşcu
  https://doi.org/10.1137/07068669X

- **Transdichotomous Results in Computational Geometry, II: Offline Search** (2010)
  Timothy Chan, Mihai Pătraşcu
  https://arxiv.org/abs/1010.1948

- **On the bit complexity of minimum link paths: Superquadratic algorithms for problem solvable in linear time** (1999)
  Simon Kahana, Jack Snoeyink
  https://doi.org/10.1016/S0925-7721(98)00041-8