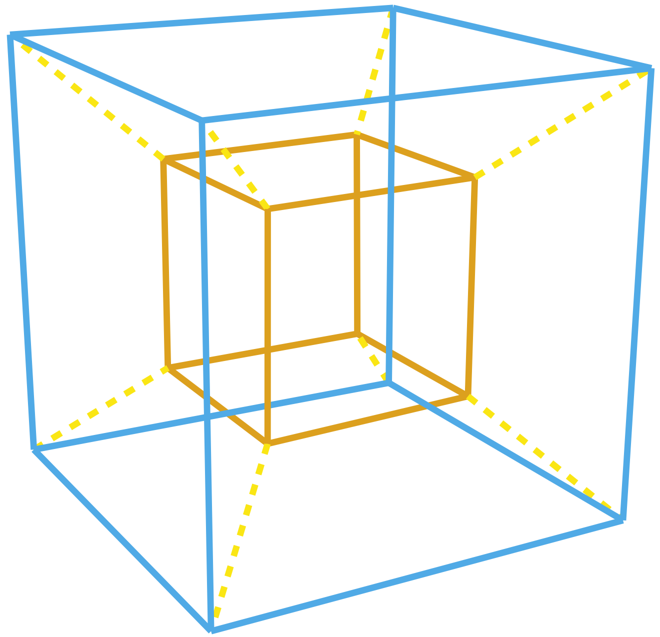
iNGEGNERIA
CIVILE, INFORMATICA E DELLE TECNOLOGIE AERONAUTICHE
ROMA TRE
UNIVERSITÀ DEGLI STUDI

Fractional Cascading: Reporting the intersections between a polygonal path and a query line



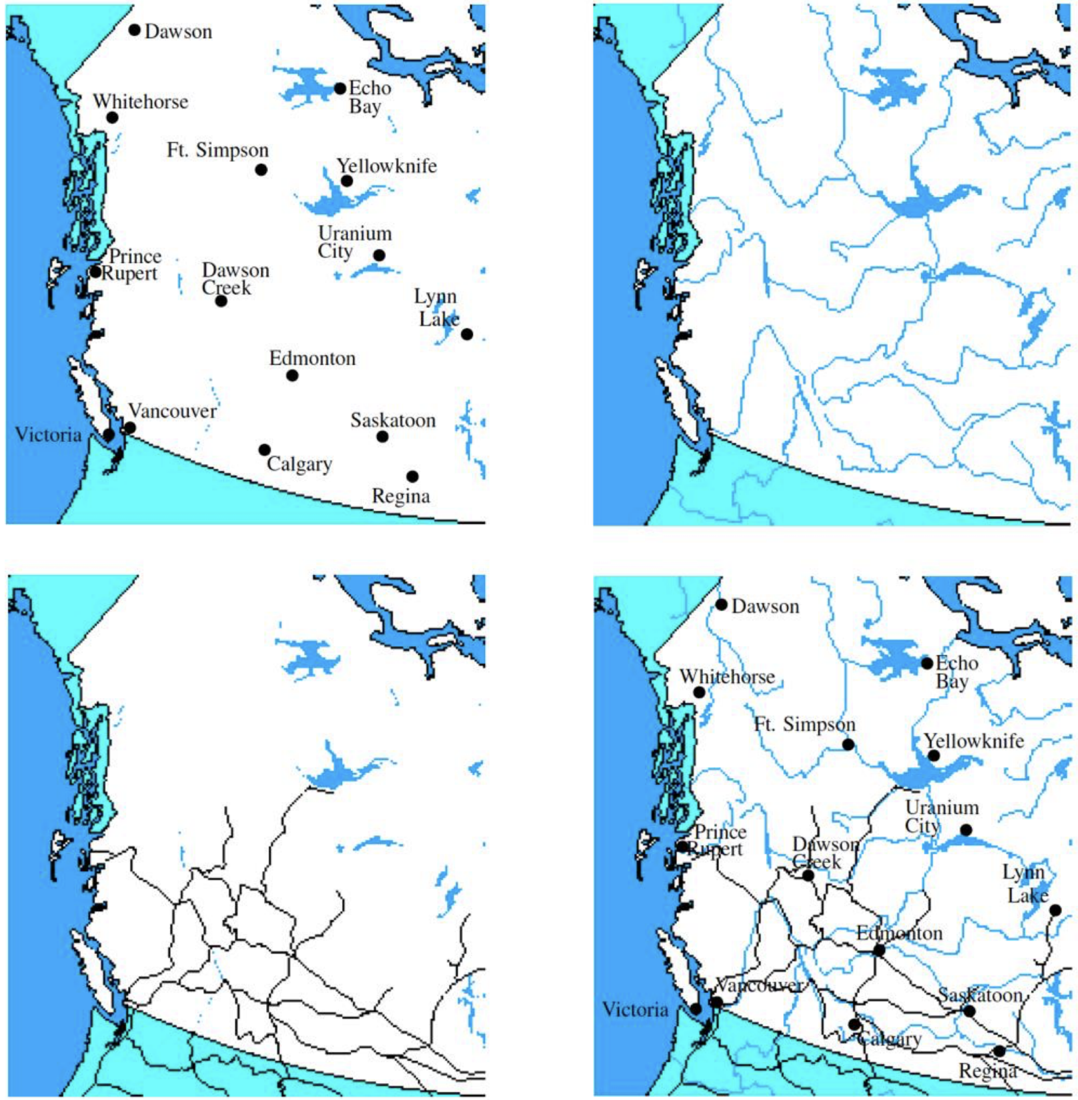
Giordano Da Lozzo | last update: 2025-10-13

Master Course: Algorithms for Big Data

geometric intersections

Important problem in Computational Geometry

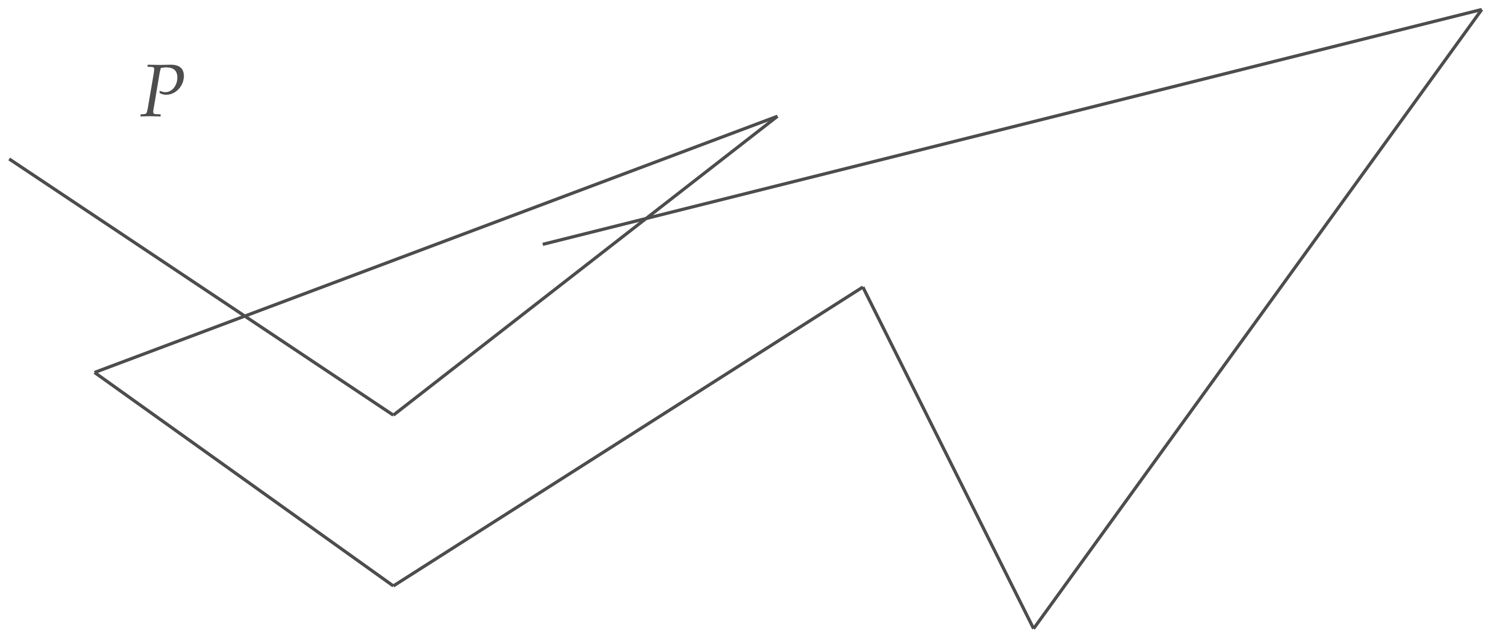
- **Constructive solid modeling:** complex shapes are constructed by applying **set operations** (intersection, union, ..) to simple shapes
- **Robotics and motion planning:** collision detection and avoidance
- **Geographic information systems:** Overlay two subdivisions (e.g., road network and river network)
- **Computer graphics:** determine the intersections of a ray with objects (*ray shooting*)



cities, rivers, railroads, and their overlay in western Canada

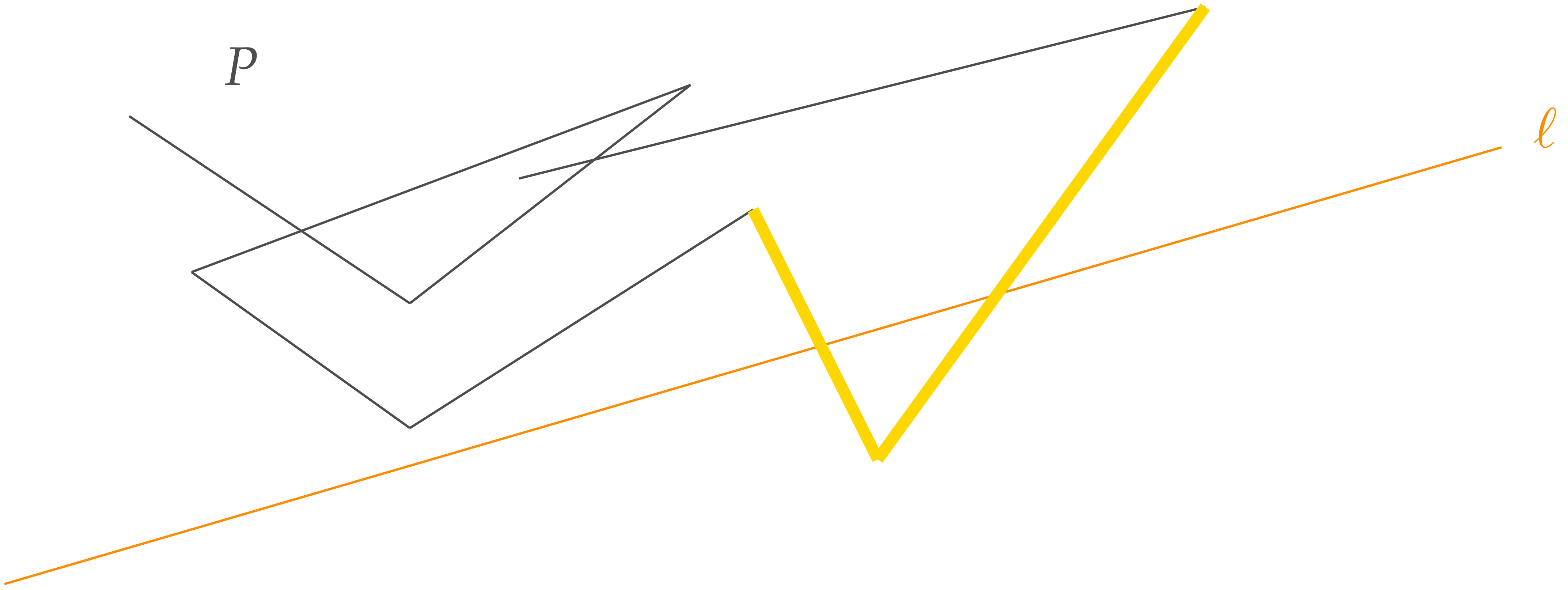
problem definition

Task: given a **polygonal path** P, we wish to **preprocess it** into a data structure so that, given **any query line** ℓ, we can quickly **report all the intersections of** P **and** ℓ**.**



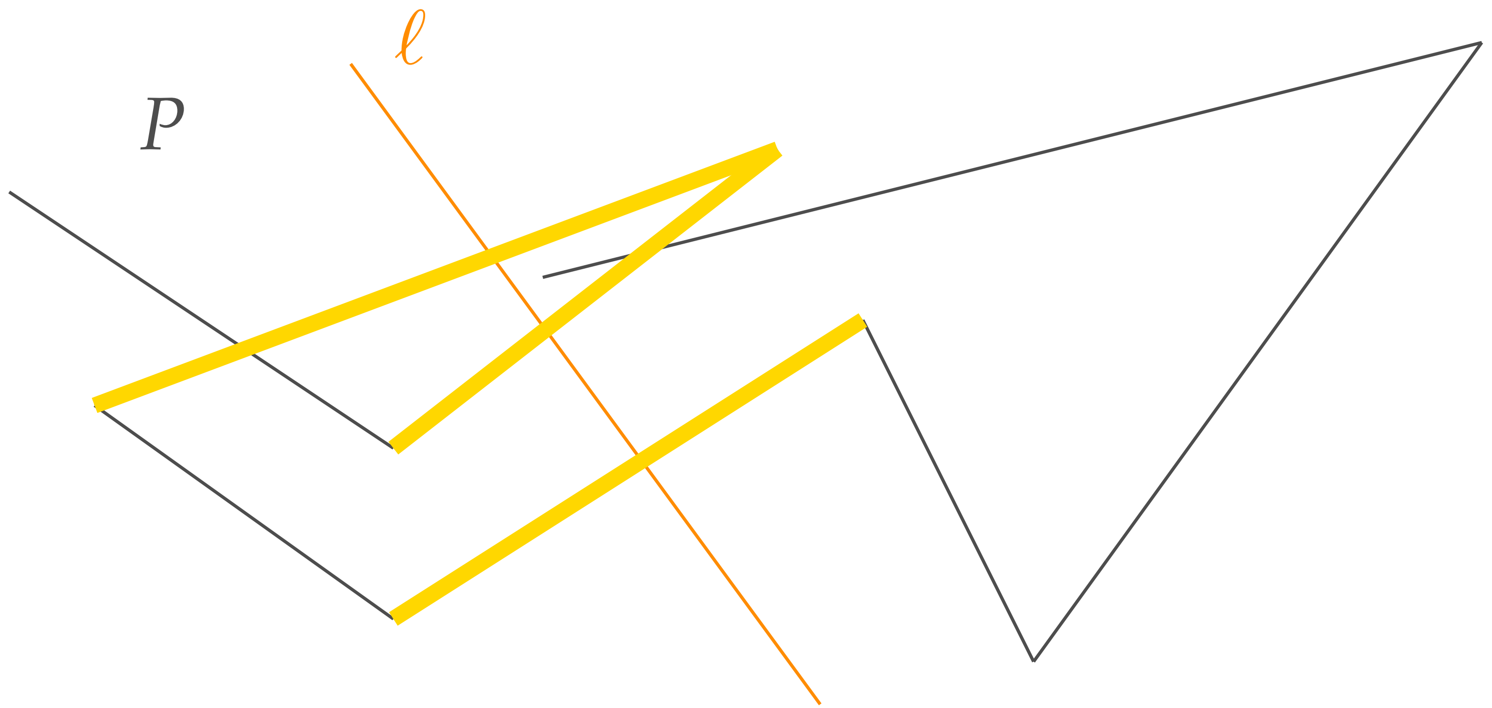
problem definition

Task: given a **polygonal path** P, we wish to **preprocess it** into a data structure so that, given **any query line** ℓ, we can quickly **report all the intersections of** P **and** ℓ**.**



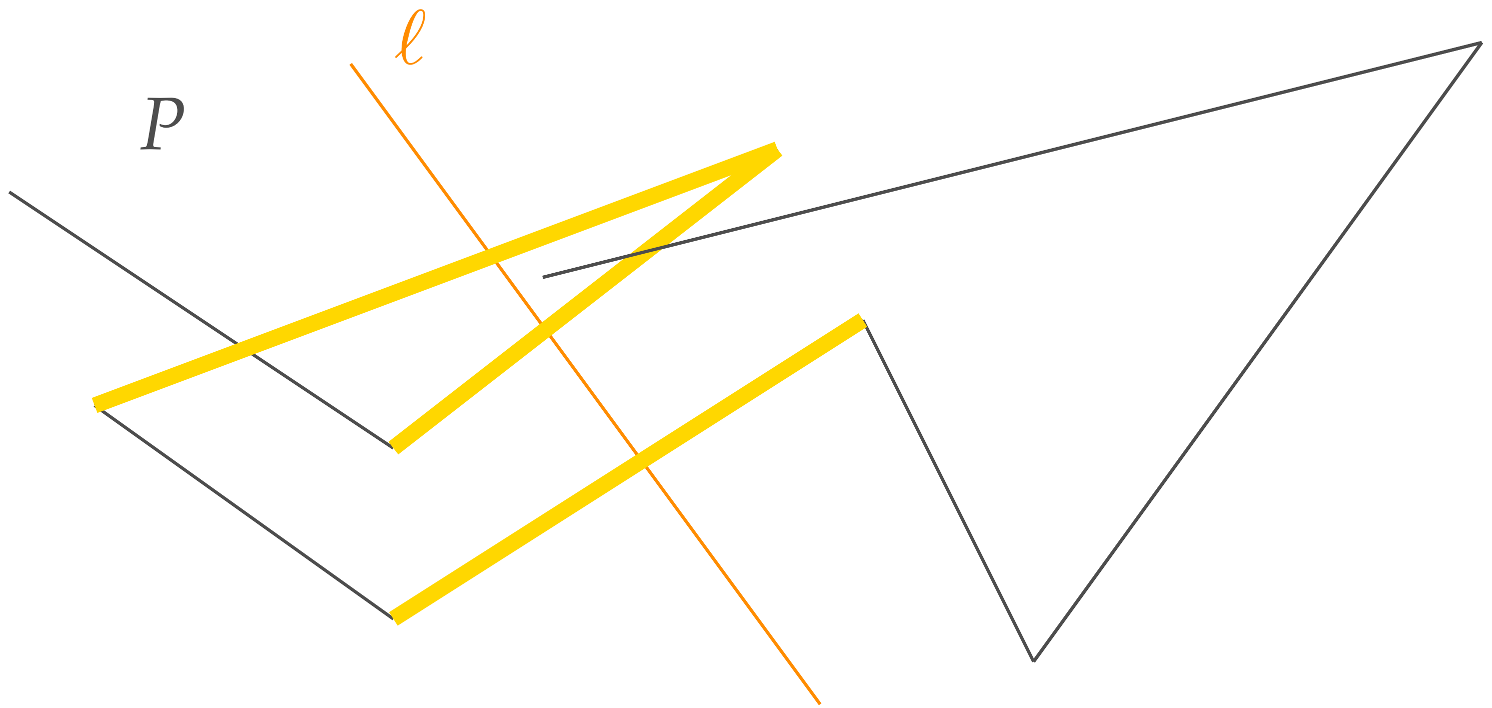
problem definition

Task: given a **polygonal path** P, we wish to **preprocess it** into a data structure so that, given **any query line** ℓ, we can quickly **report all the intersections of** P **and** ℓ.



problem definition

Task: given a **polygonal path** P, we wish to **preprocess it** into a data structure so that, given **any query line** ℓ, we can quickly **report all the intersections of** P **and** ℓ.



The obvious method

check each side of P for intersection with ℓ: This method requires $O(n)$ **storage and query time**, if n is the length of P.

our goal

Task: given a **polygonal path** P, we wish to **preprocess it** into a data structure so that, given **any query line** ℓ, we can quickly **report all the intersections of P and ℓ.**

our goal

Task: given a **polygonal path** P, we wish to **preprocess it** into a data structure so that, given **any query line** ℓ, we can quickly **report all the intersections of P and ℓ.**

$\Downarrow$

we aim at $O(f(n) + h)$, where $f(n) = o(n)$ and h is the number of intersections reported.

our goal

Task: given a **polygonal path** P, we wish to **preprocess it** into a data structure so that, given **any query line** ℓ, we can quickly **report all the intersections of** P **and** ℓ**.**

$\Downarrow$

we aim at $O(f(n)+h)$, where $f(n)=o(n)$ and h is the number of intersections reported.

$\Downarrow$

using fractional cascading, we are able to develop a technique that achieves $O((h+1)\log\lceil n/(h+1)\rceil)$ query time

our goal

Task: given a **polygonal path** P, we wish to **preprocess it** into a data structure so that, given **any query line** ℓ, we can **quickly report all the intersections of P and ℓ.**

⇓

we aim at $O(f(n)+h)$, where $f(n)=o(n)$ and h is the number of intersections reported.

⇓

using fractional cascading, we are able to develop a technique that achieves $O((h+1)\log\lceil n/(h+1)\rceil)$ query time

when h is a **small constant**

query time is $O(\log n)$, which is optimal!

our goal

Task: given a **polygonal path** P, we wish to **preprocess it** into a data structure so that, given **any query line** ℓ, we can **quickly report all the intersections of** P **and** ℓ**.**

$\Downarrow$

we aim at $O(f(n)+h)$, where $f(n)=o(n)$ and h is the number of intersections reported.

$\Downarrow$

using fractional cascading, we are able to develop a technique that achieves $O((h+1)\log\lceil n/(h+1)\rceil)$ query time

when h is a **small constant**

query time is $O(\log n)$, which is optimal!

when $h \in \Omega(n)$

query time is $O(h)$, which is optimal!

our goal

Task: given a **polygonal path** P, we wish to **preprocess it** into a data structure so that, given **any query line** ℓ, we can **quickly report all the intersections of** P **and** ℓ**.**

$\Downarrow$

we aim at $O(f(n)+h)$, where $f(n)=o(n)$ and h is the number of intersections reported.

$\Downarrow$

using fractional cascading, we are able to develop a technique that achieves $O((h+1)\log\lceil n/(h+1)\rceil)$ query time

when h is a **small constant**

query time is $O(\log n)$, which is optimal!

$\Downarrow$

for intermediate values of h

the discovery of each intersection incurs the cost of a binary search

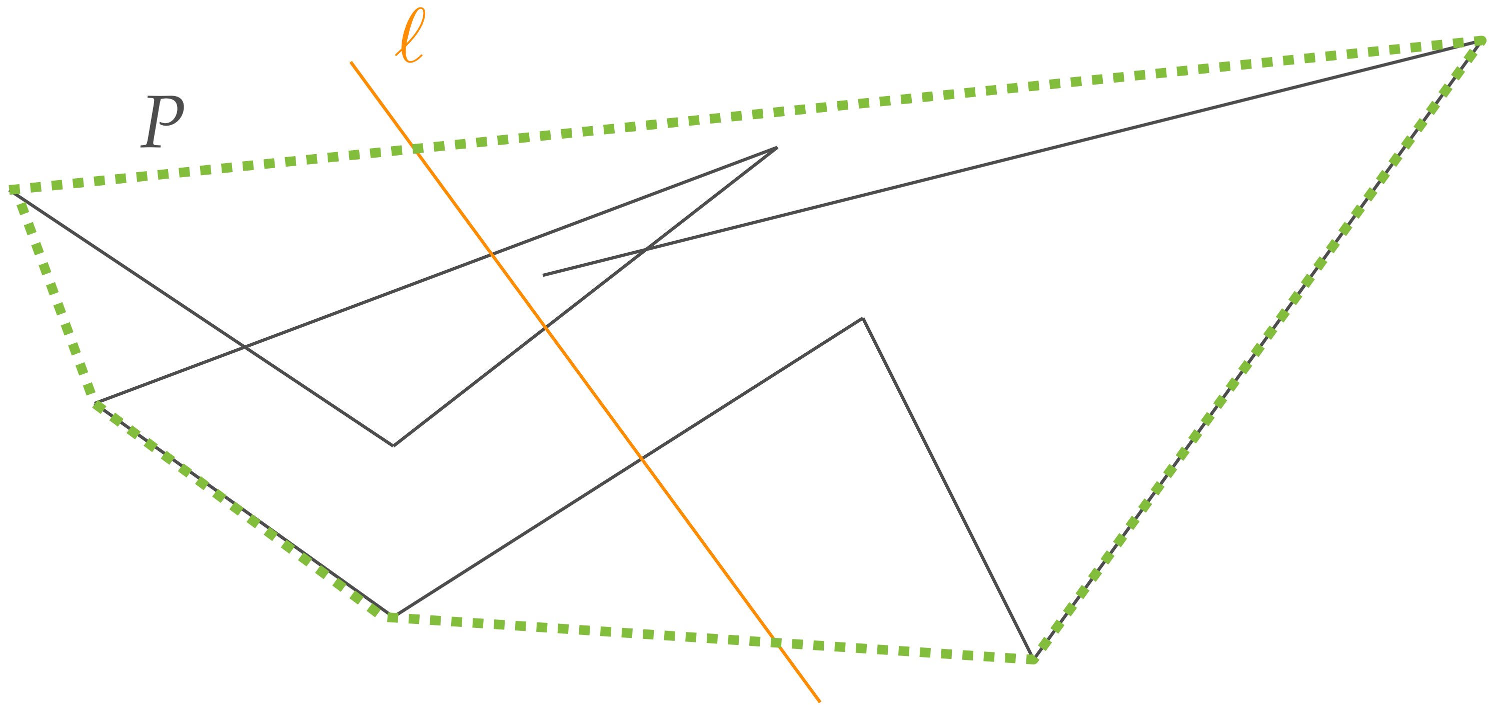
when $h \in \Omega(n)$

query time is $O(h)$, which is optimal!

key observation

Lemma 1 [Condition for intersection]

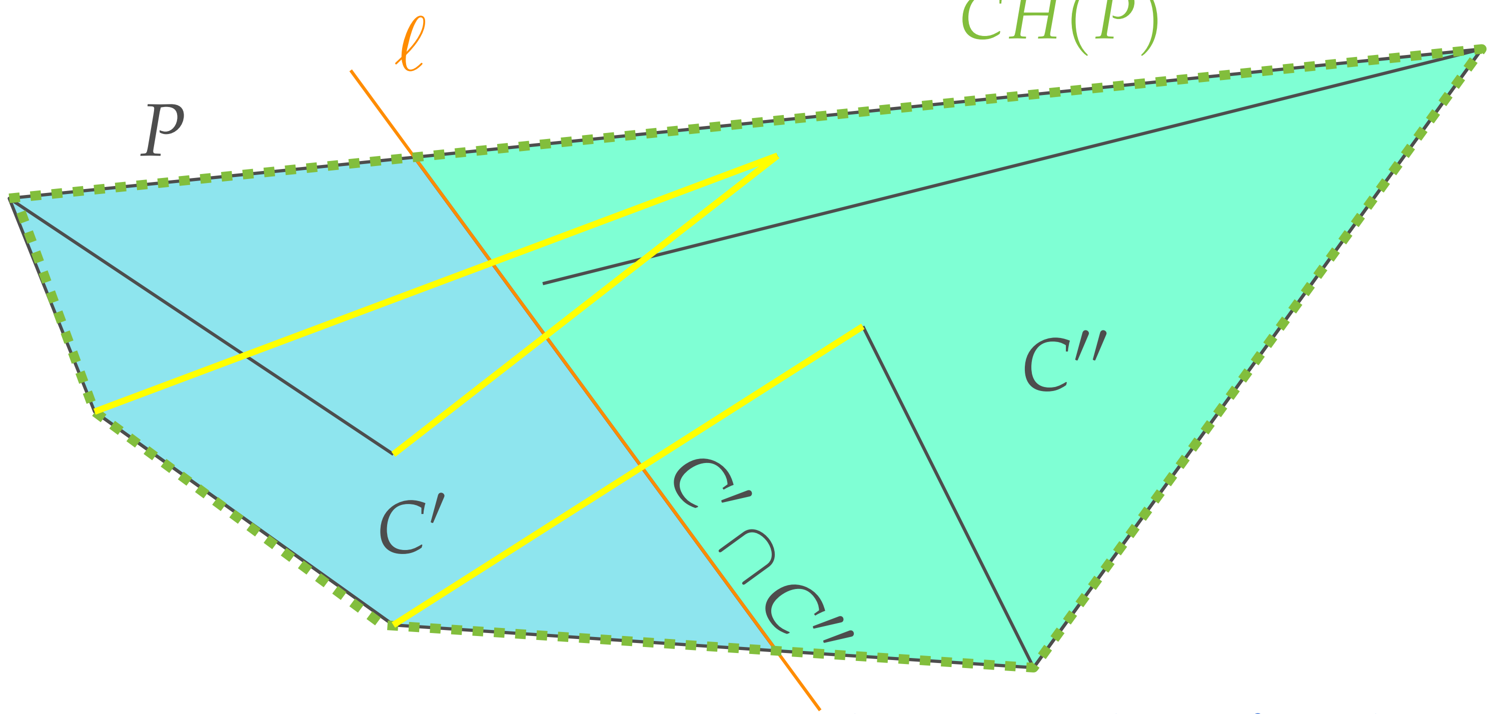
A straight-line ℓ intersects a polygonal line path P if and only if ℓ intersects the convex hull $CH(P)$ of P.



key observation

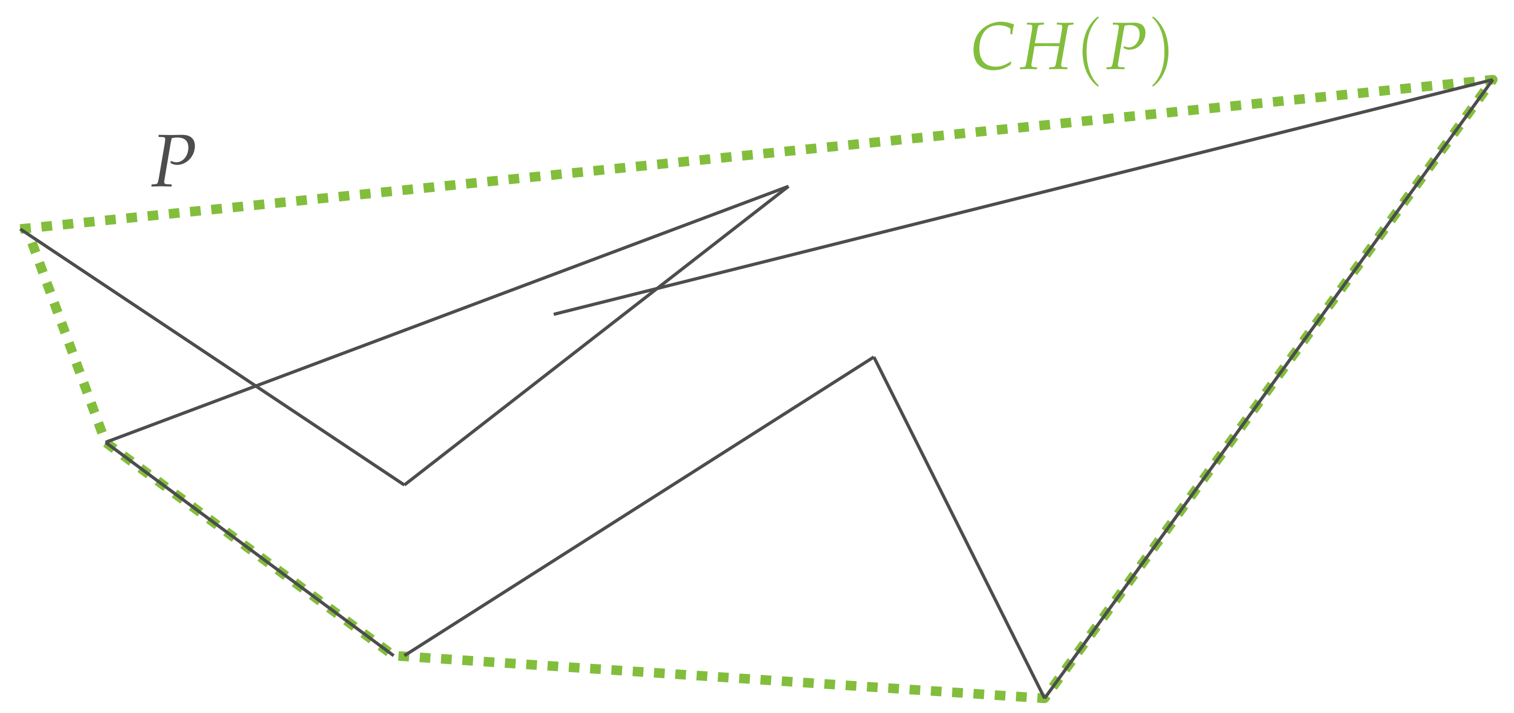
Lemma 1 [Condition for intersection]

A straight-line ℓ intersects a polygonal line path P if and only if ℓ intersects the convex hull $CH(P)$ of P.



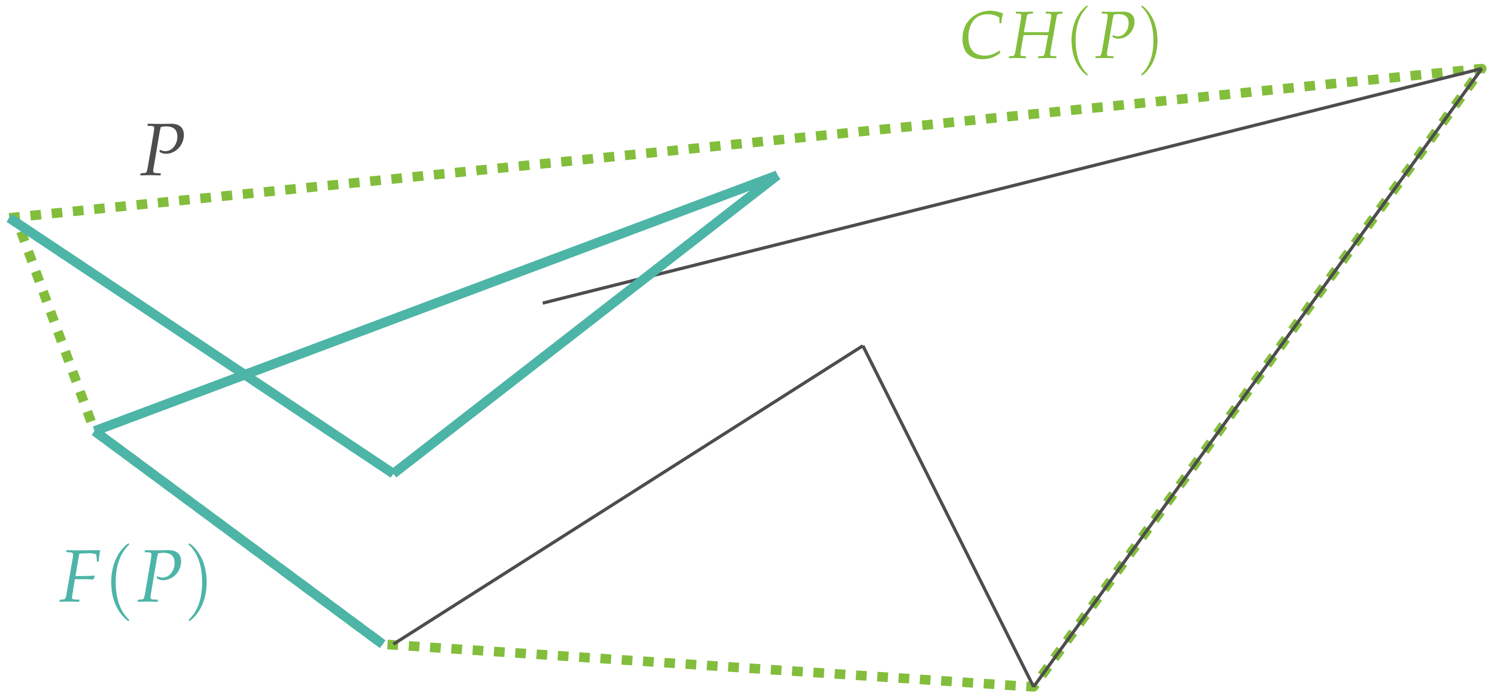
Proof.
Necessity is obvious. **For the sufficiency**, observe that ℓ splits $CH(P)$ into **two convex polygons** C' **and** C''. Since P is connected, the segment $C' \cap C''$ shared by C' and C'' must be traversed by (or incident to) an edge e of P, and thus e crosses ℓ. $\square$

techniques's idea



techniques's idea

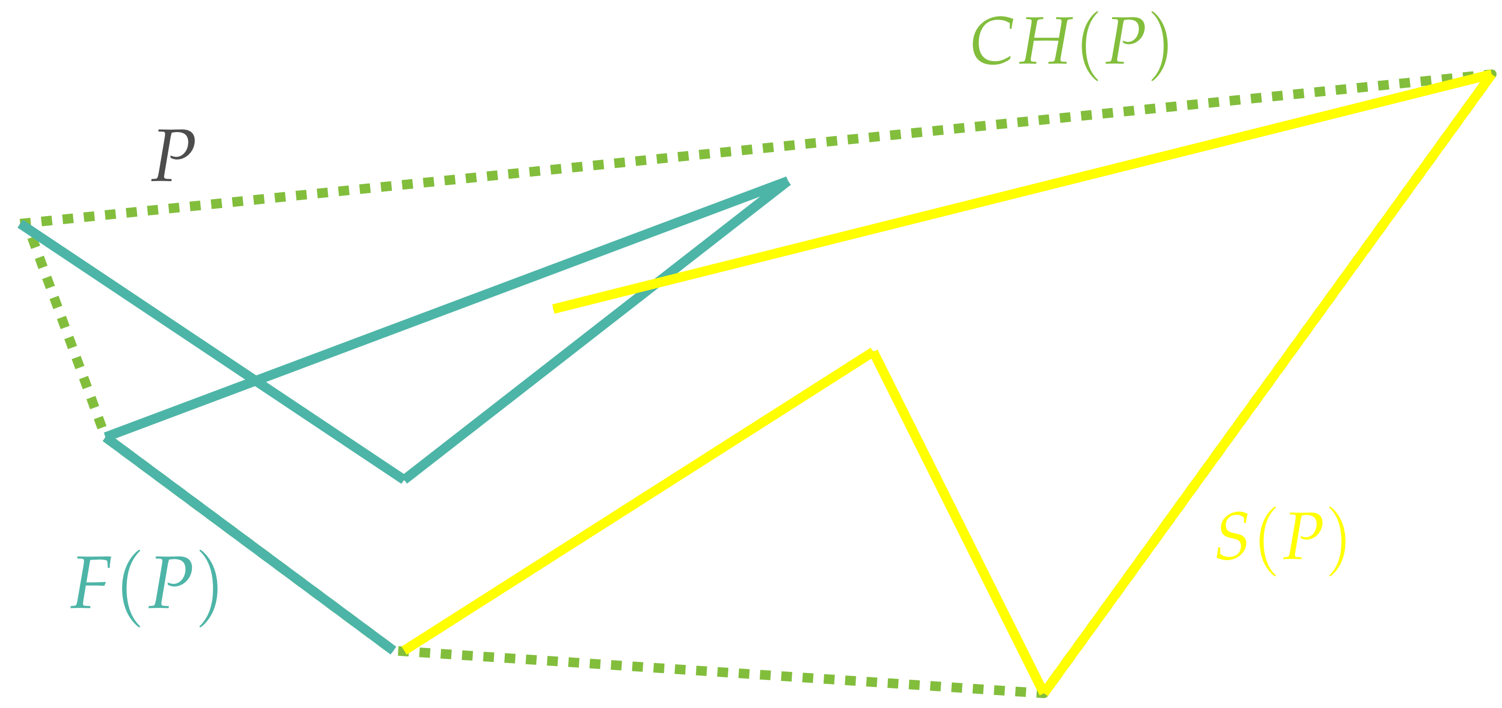
$F(P) :=$ the **first half** of P, i.e., the subpath of P consisting of the first $\lfloor n/2 \rfloor$ edges



techniques's idea

$F(P)$:= the **first half** of P, i.e., the subpath of P consisting of the first $\lfloor n/2 \rfloor$ edges

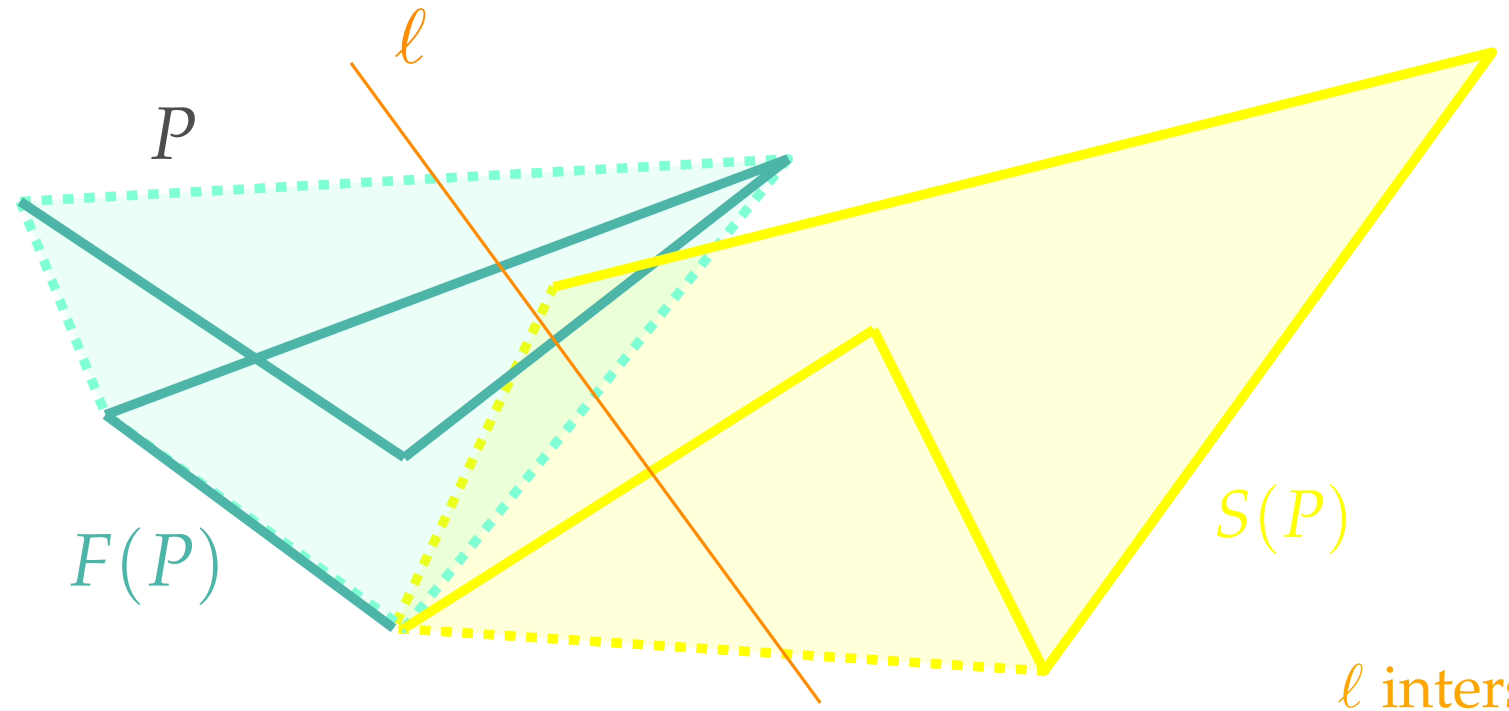
$S(P)$:= the **secod half** of P, i.e., the subpath of P consisting of the last $\lceil n/2 \rceil$ edges



techniques's idea

$F(P)$:= the **first half** of P, i.e., the subpath of P consisting of the first $\lfloor n/2 \rfloor$ edges

$S(P)$:= the **secod half** of P, i.e., the subpath of P consisting of the last $\lceil n/2 \rceil$ edges



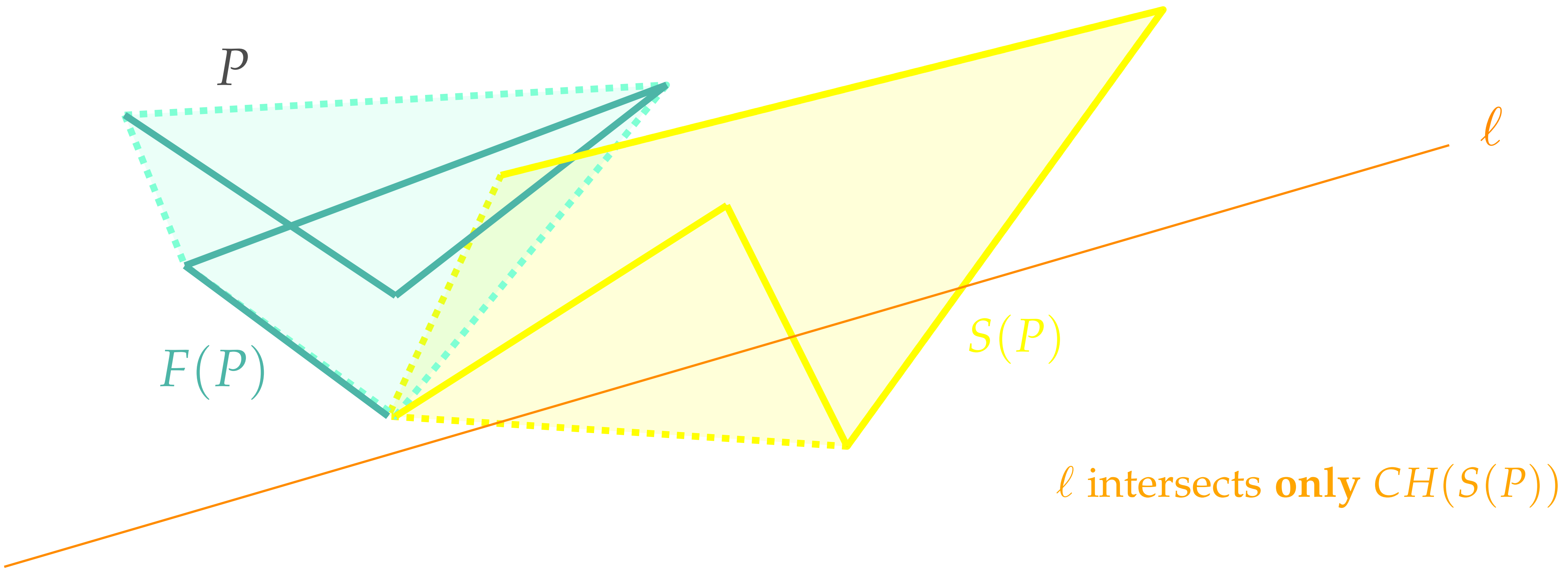
ℓ intersects **both** $CH(F(P))$ and $CH(S(P))$

Observation. Even if ℓ intersects $CH(P)$ of P, it may intersect the convex hull of only one of $F(P)$ and $S(P)$

techniques's idea

$F(P)$:= the **first half** of P, i.e., the subpath of P consisting of the first $\lfloor n/2 \rfloor$ edges

$S(P)$:= the **secod half** of P, i.e., the subpath of P consisting of the last $\lceil n/2 \rceil$ edges



Observation. Even if ℓ intersects $CH(P)$ of P, it may intersect the convex hull of only one of $F(P)$ and $S(P)$

Algorithm to report all the intersections of P and ℓ

- the algorithm implements a recursive strategy based on Lemma 1

```
INTERSECT(P, ℓ):
  if |P| = 1 (single edge e) then
      if ℓ intersects e then   //ℓ ∩ P can be computed directly in O(1) time
          return (e)
      else
          return (∅)   //exit!
  else
      if ℓ does not intersects CH(P) then   //Lemma 1
          return (∅)   //exit!
      else
          F ← INTERSECT(F(P),ℓ)
          S ← INTERSECT(S(P),ℓ)
          return (F ∪ S)
```

Algorithm to report all the intersections of P and ℓ

- the algorithm implements a recursive strategy based on Lemma 1

```
INTERSECT(P, ℓ):
  if |P| = 1 (single edge e) then
      if ℓ intersects e then   //ℓ ∩ P can be computed directly in O(1) time
          return (e)
      else
          return (∅)   //exit!
  else
      if ℓ does not intersects CH(P) then   //Lemma 1
          return (∅)   //exit!
      else
          F ← INTERSECT(F(P),ℓ)
          S ← INTERSECT(S(P),ℓ)
          return (F ∪ S)
```

This test is (in general) expensive:
it costs $O(\log m)$ time to test whether a line intersects a convex polygon of m sides

running time analysis

To decide whether to descend into a subtree, we test for intersections between the convex hull stored in its root and ℓ

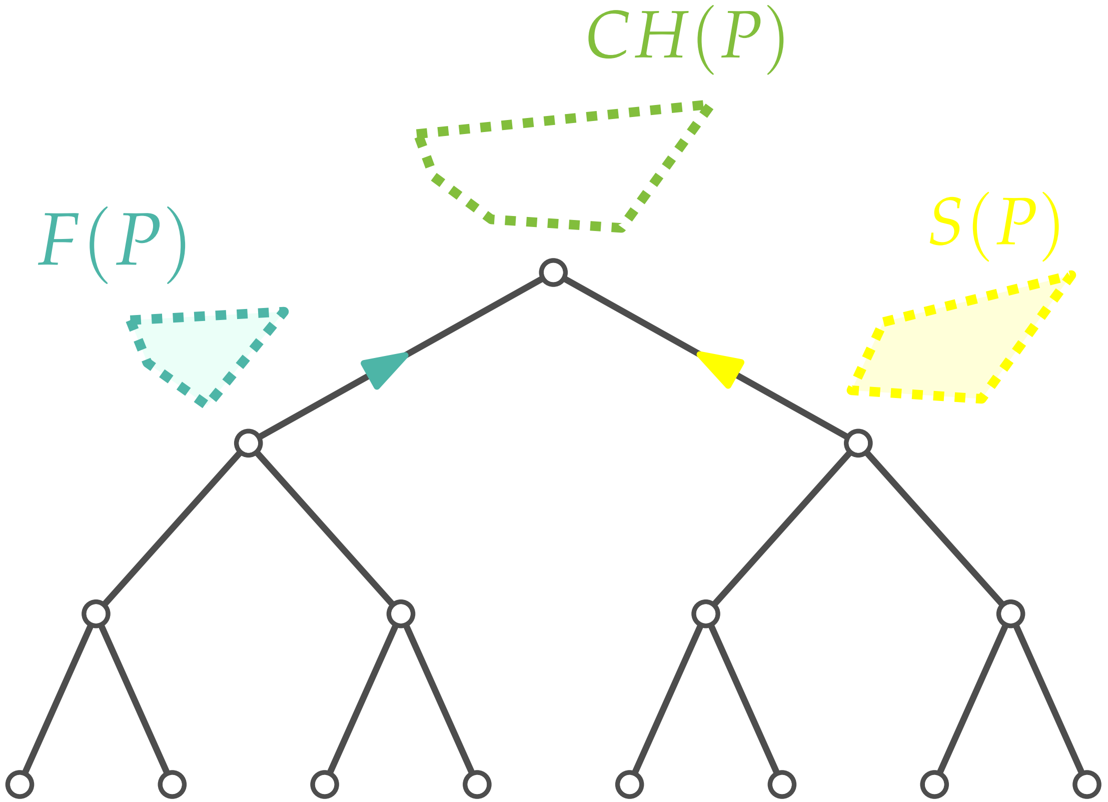
Even if we were to report **only one intersection** (i.e., we track only one path down to the intersected edge), **the total cost of all these tests would be:**

$$\Omega(\sum_i \log \frac{n}{2^i}) = \Omega(\log^2 n)$$

This is already too expensive!!
Here is where **fractional cascading comes in.**

preprocessing

Since **we are allowed to preprocess** P,
we can precompute and store in a binary tree
all the convex hulls we may need in $O(n \log n)$
time, where $n = |P|$

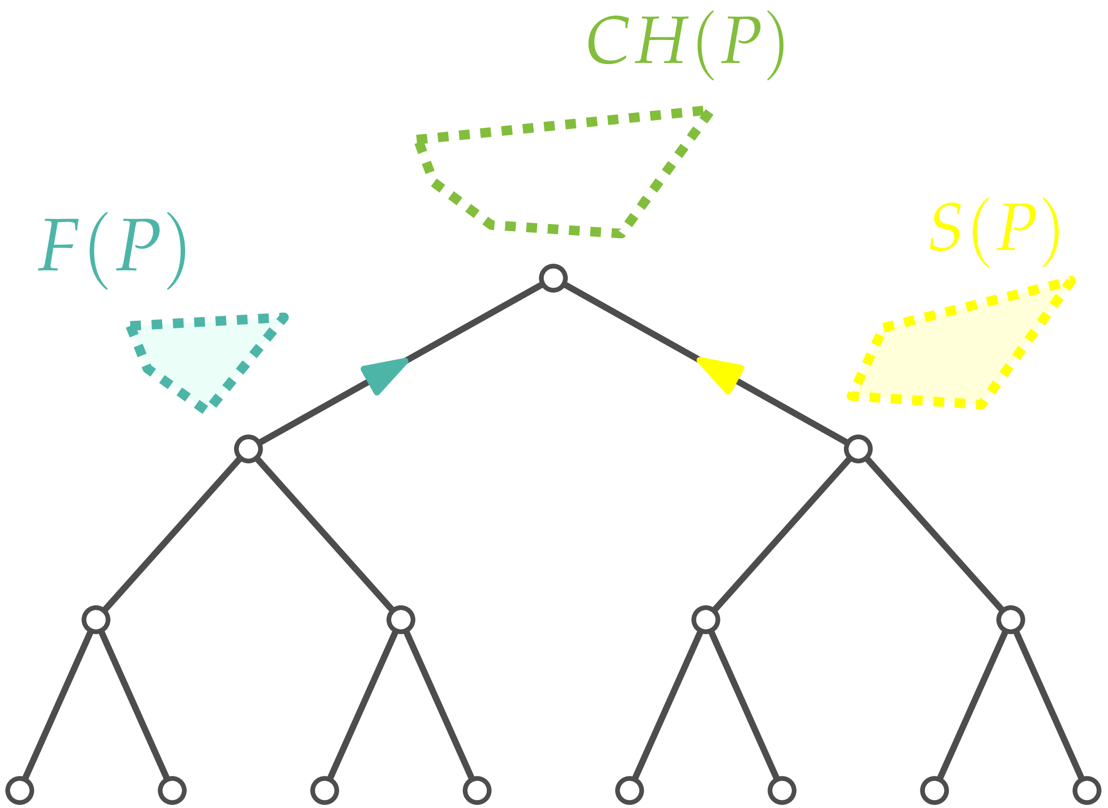


preprocessing

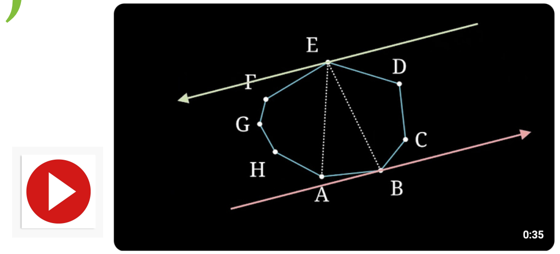
Since **we are allowed to preprocess** P, we can precompute and store in a binary tree **all the convex hulls** we may need in $O(n \log n)$ time, where $n = |P|$

This can be done by a recursion similar to the previous one, where:

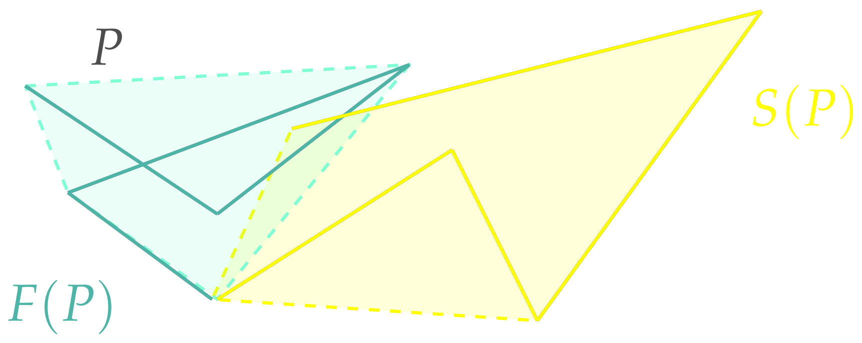
- first, we recursively compute $CH(F(P))$ and $CH(S(P))$
- then, we obtain $CH(P)$ from $CH(F(P))$ and $CH(S(P))$ (using any linear-time algorithms for computing **the convex hull of (the union of) two convex polygons [PH])**

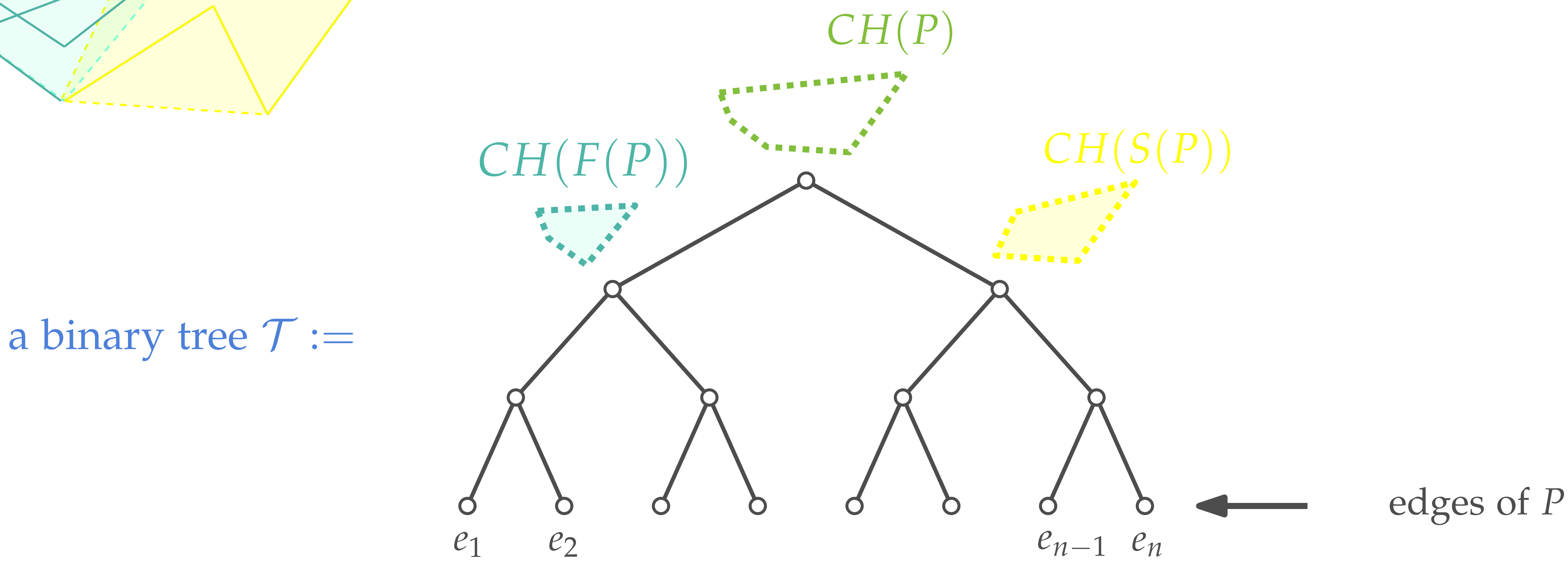


rotating calipers
(*calibri rotanti*)



data structure

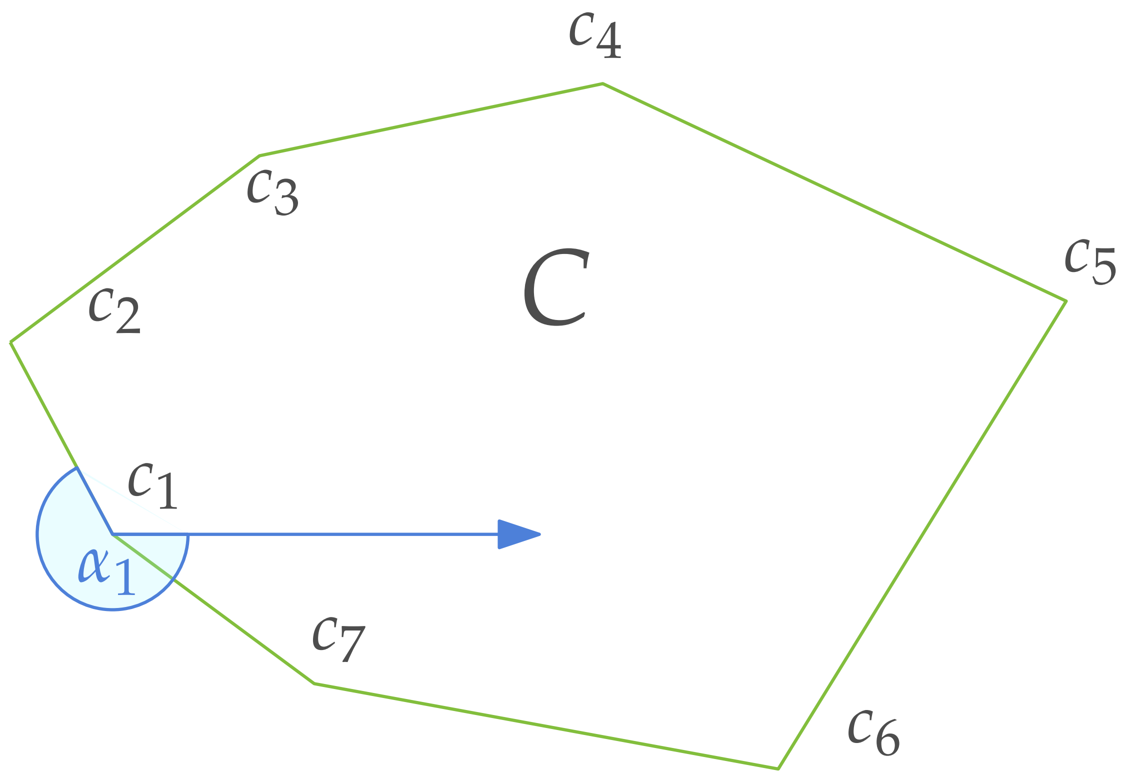




- **the n leaves are the edges of our path** P (which coincide with their own convex hulls and from left to right occur in the same order as in P)
- **the interior nodes correspond to subpaths of P and store the respective convex hull**

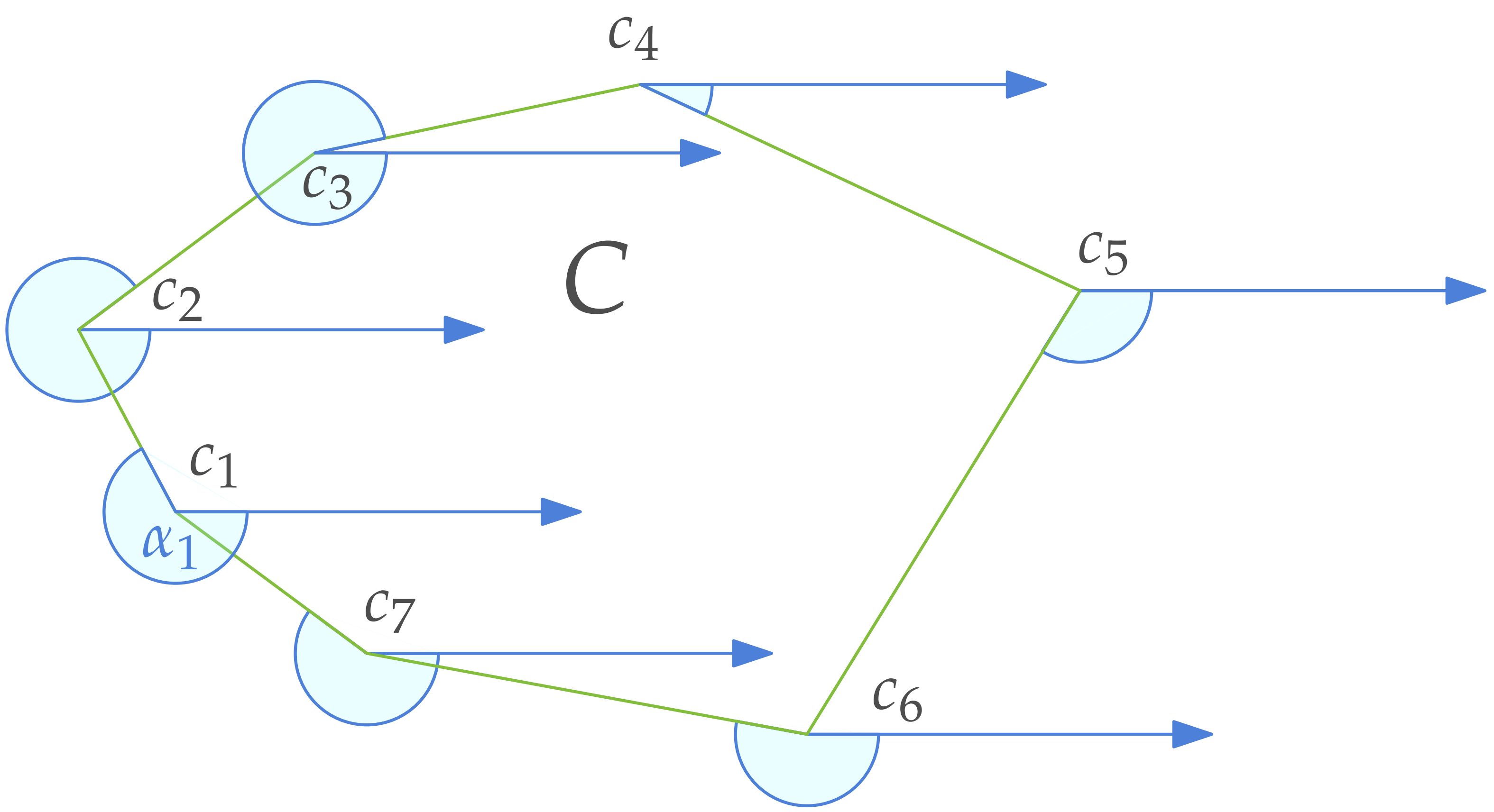
testing intersection bw line and convex polygon

Def. Let C be a convex polygon. The **slope of the segment** $\overline{c_i c_{i+1}}$ of C is the angle $\alpha_i \in [0, 2\pi)$ bw the horizontal ray originating at c_i and direct rightward and the segment $\overline{c_i c_{i+1}}$.



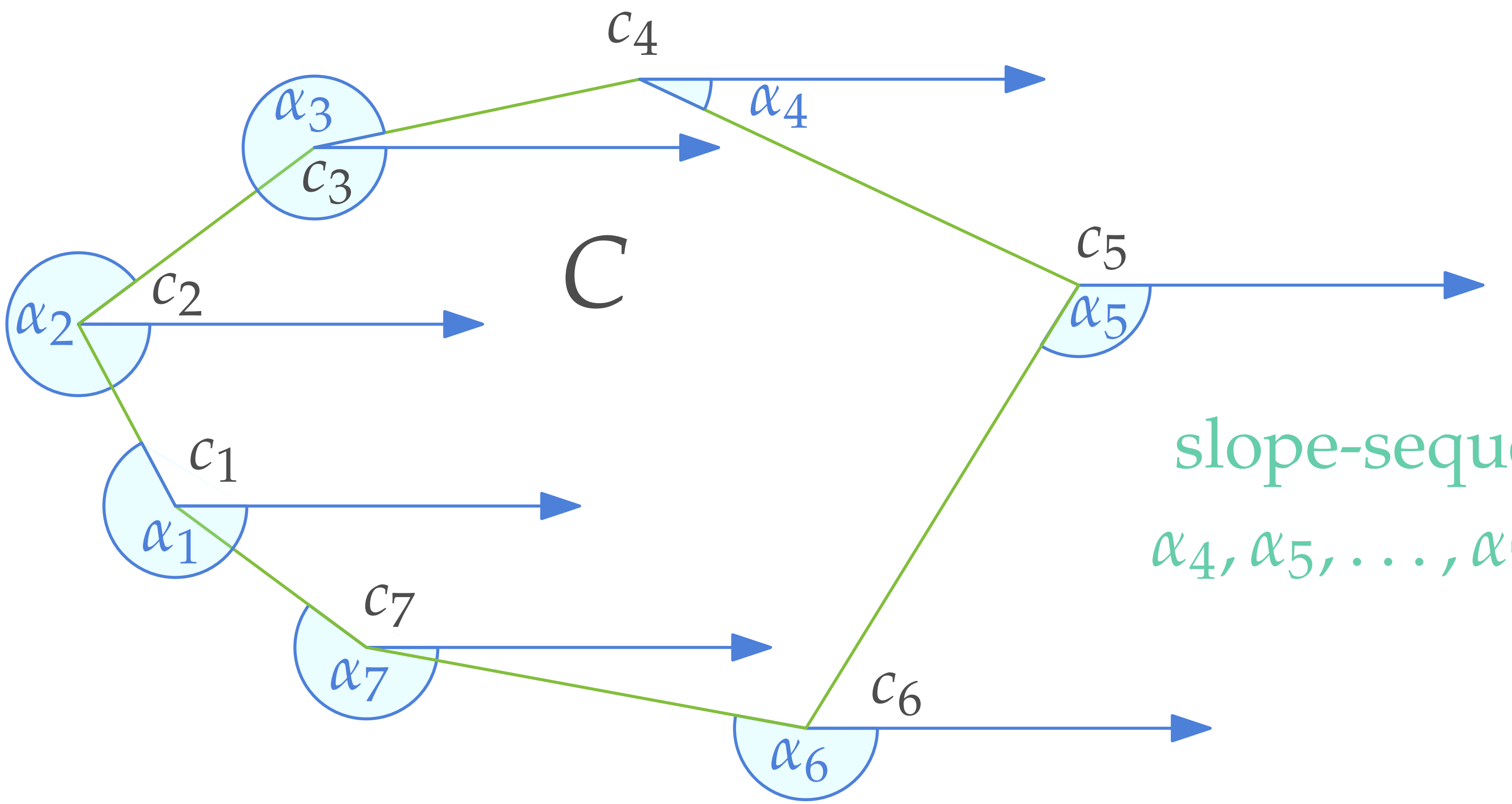
testing intersection bw line and convex polygon

Def. Let C be a convex polygon. The **slope of the segment** $\overline{c_i c_{i+1}}$ of C is the angle $\alpha_i \in [0, 2\pi)$ bw the horizontal ray originating at c_i and direct rightward and the segment $\overline{c_i c_{i+1}}$.



testing intersection bw line and convex polygon

Def. Let C be a convex polygon. The **slope of the segment** $\overline{c_i c_{i+1}}$ of C is the angle $\alpha_i \in [0, 2\pi)$ bw the horizontal ray originating at c_i and direct rightward and the segment $\overline{c_i c_{i+1}}$.

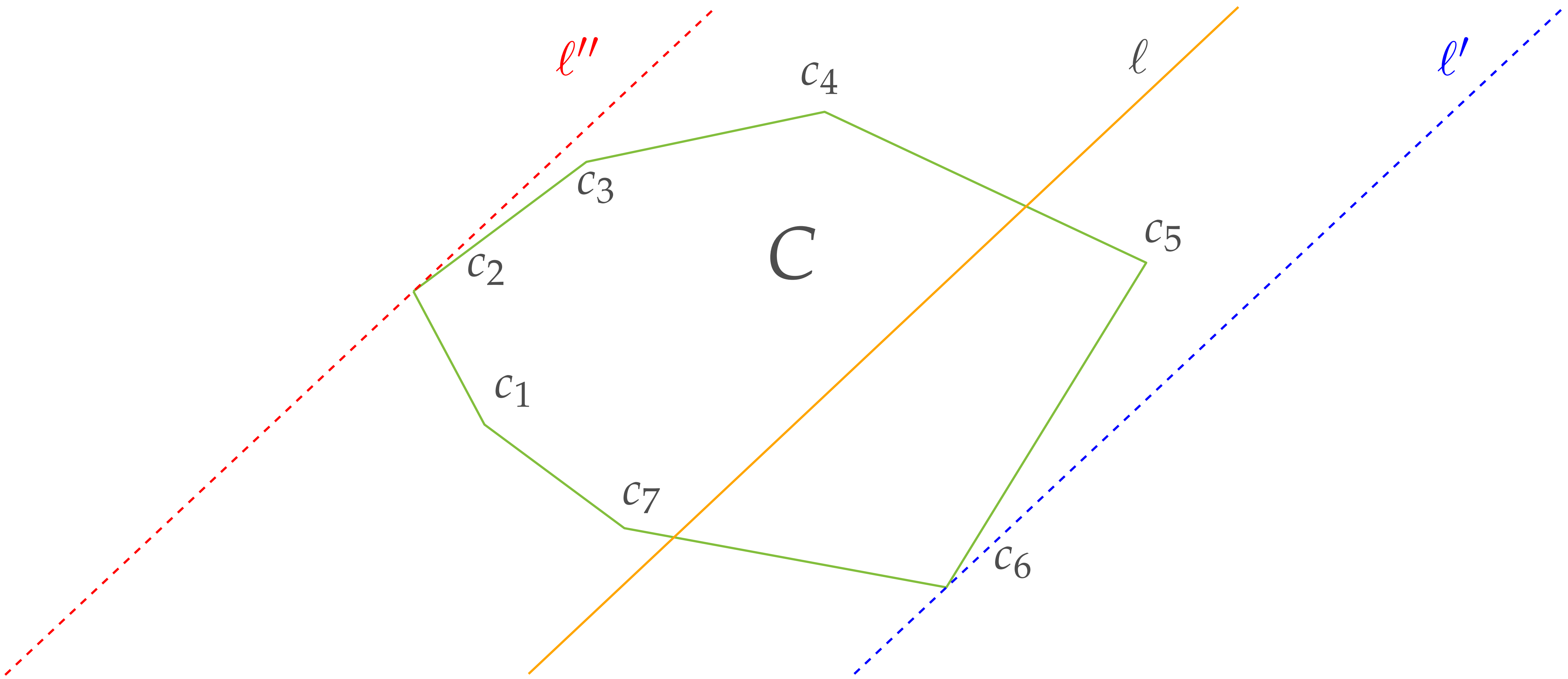


slope-sequence of C :

$\alpha_4, \alpha_5, \ldots, \alpha_7, \alpha_1, \ldots, \alpha_3$

- Since C is convex, there exists a circular permutation of the edges of C such that **the sequence of slopes is nondecreasing**.
- This sequence is **unique**, and is called **the slope-sequence of** C.

testing intersection bw line and convex polygon

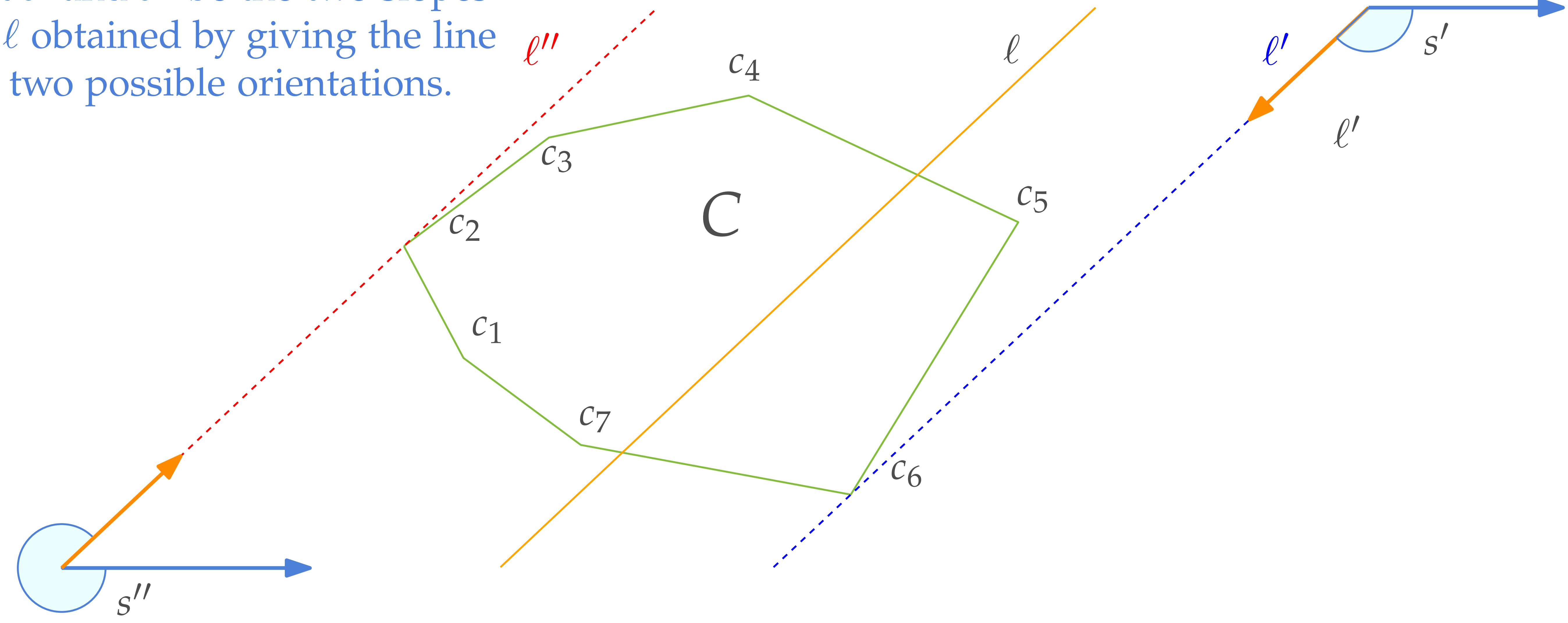


Observation:

ℓ crosses C iff it lies between the two tangents at C parallel to ℓ

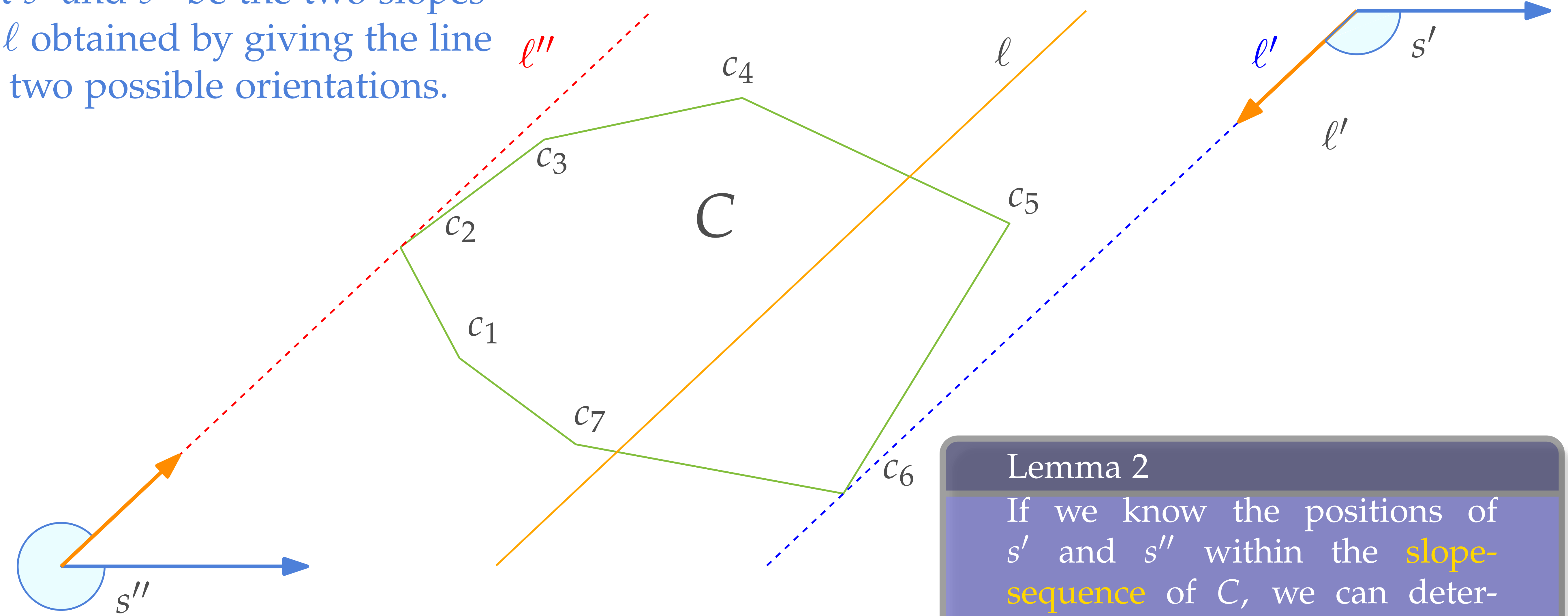
testing intersection bw line and convex polygon

Let s' and s'' be the two slopes of ℓ obtained by giving the line its two possible orientations.



testing intersection bw line and convex polygon

Let s' and s'' be the two slopes of ℓ obtained by giving the line its two possible orientations.

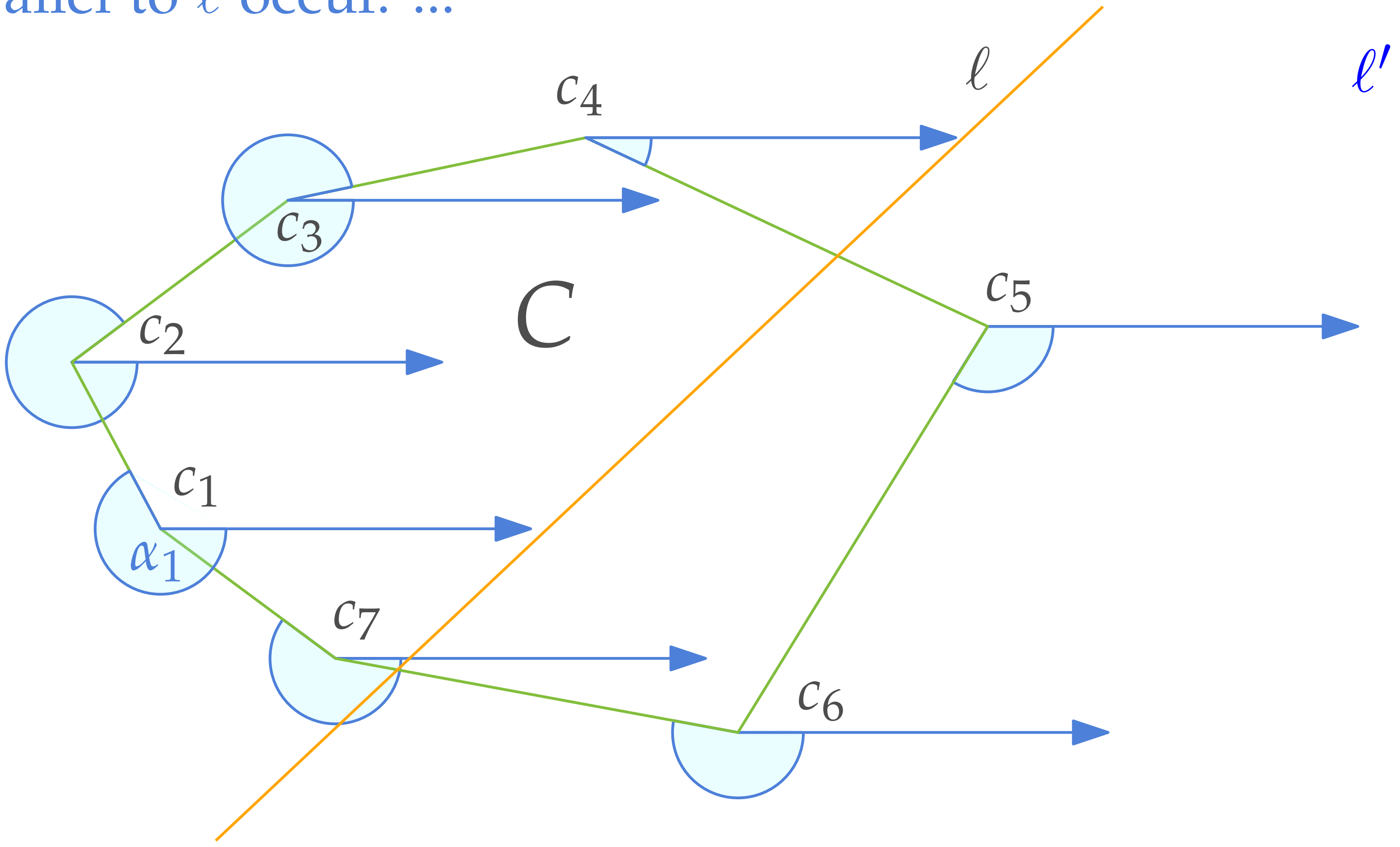


Lemma 2

If we know the positions of s' and s'' within the slope-sequence of C, we can determine whether C and ℓ intersect in $O(1)$ **time.**

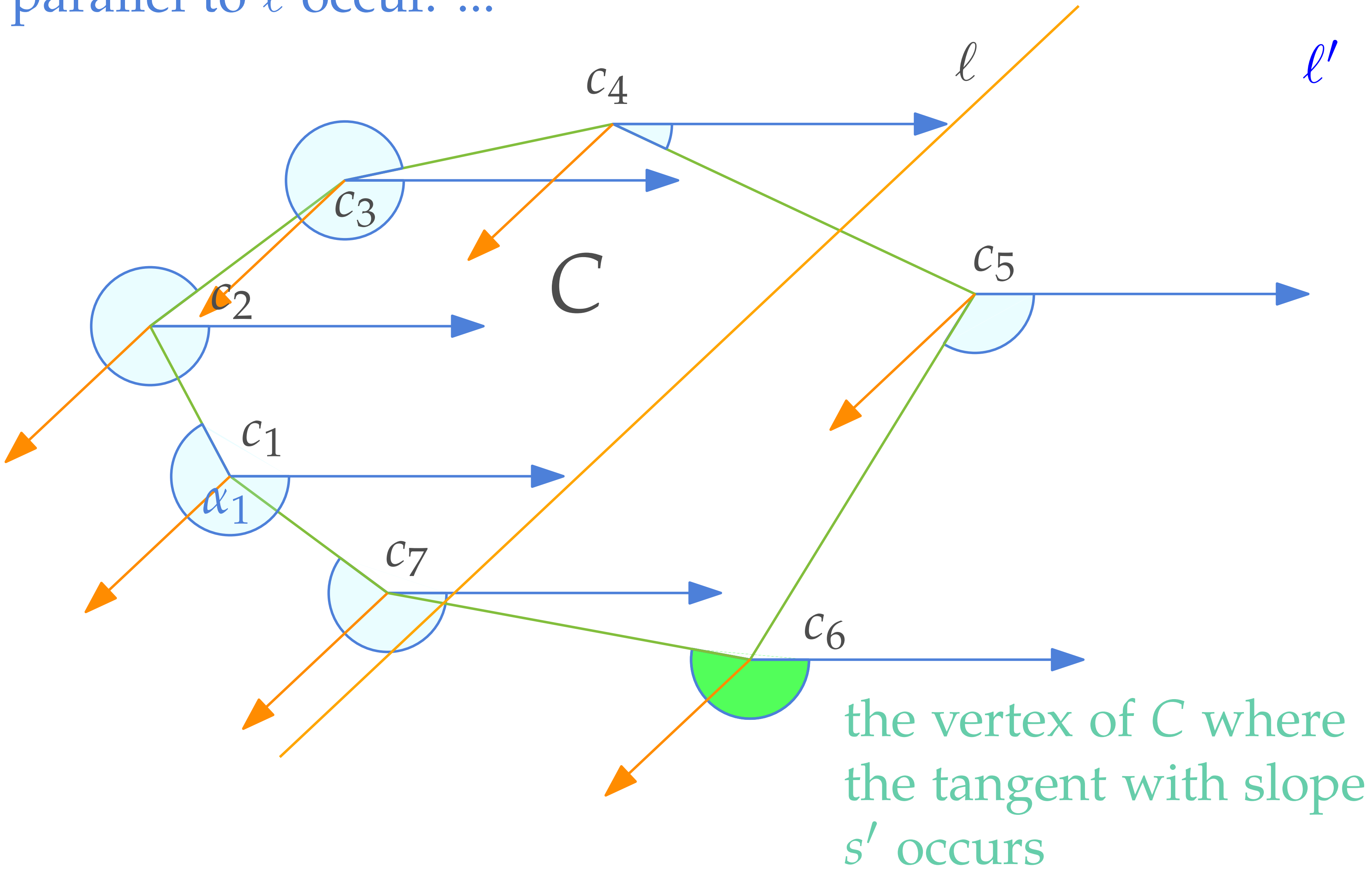
testing intersection bw line and convex polygon

Proof of Lemma 2: The positions of s' and s'' in the slope-sequence tell us the vertices of C where the tangents parallel to ℓ occur. ...



testing intersection bw line and convex polygon

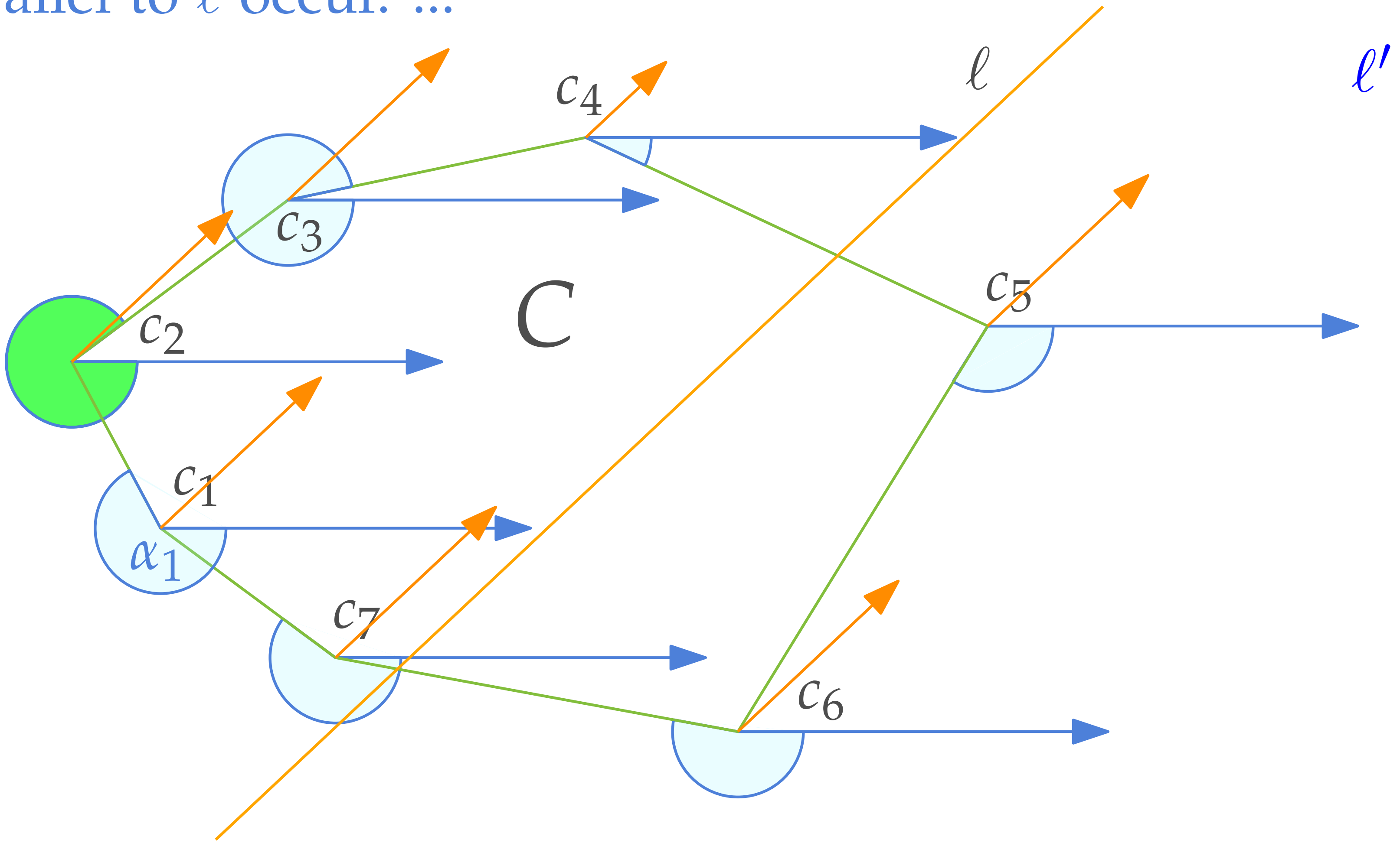
Proof of Lemma 2: The positions of s' and s'' in the slope-sequence tell us the vertices of C where the tangents parallel to ℓ occur. ...



testing intersection bw line and convex polygon

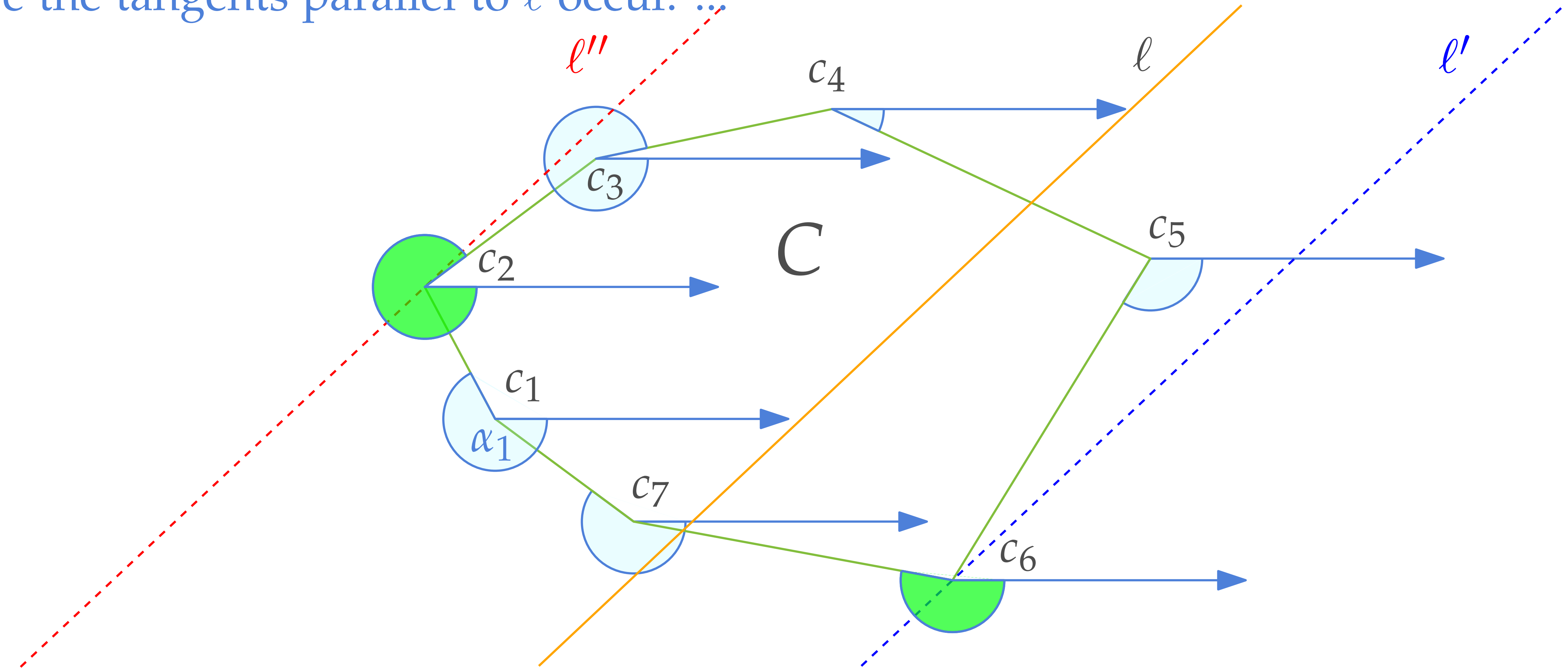
Proof of Lemma 2: The positions of s' and s'' in the slope-sequence tell us the vertices of C where the tangents parallel to ℓ occur. ...

the vertex of C where the tangent with slope s'' occurs



testing intersection bw line and convex polygon

Proof of Lemma 2: The positions of s' and s'' in the slope-sequence tell us the vertices of C where the tangents parallel to ℓ occur. ...



... If we have such vertices, we have ℓ' and ℓ'' (given that we also have s' and s''). Also, by Observation 1, $C \cap \ell \neq \emptyset$ iff ℓ lies between ℓ' and ℓ'', which can be tested in $O(1)$ time. □

GET READY FOR FRACTIONAL CASCADING

A Data Structure for Reporting All Intersections

previously: fractional cascading

Task: Given sets $B_r \subseteq \cdots \subseteq B_2 \subseteq B_1 \subseteq A \subset \mathbb{R}$ stored in **sorted order** in **arrays** $A[1 \ldots n], B_1[1 \ldots m_1], B_2[1 \ldots m_2], \ldots, B_r[1 \ldots m_r]$, perform a **1D range query** $[x : x']$ on the **multiset** $U = A \cup \left(\bigcup_1^r B_i\right)$ in $O(h + r + \log n)$ time (where $h = |[x : x'] \cap U|$)

A											
3	10	19	23	30	37	59	62	70	80	100	105

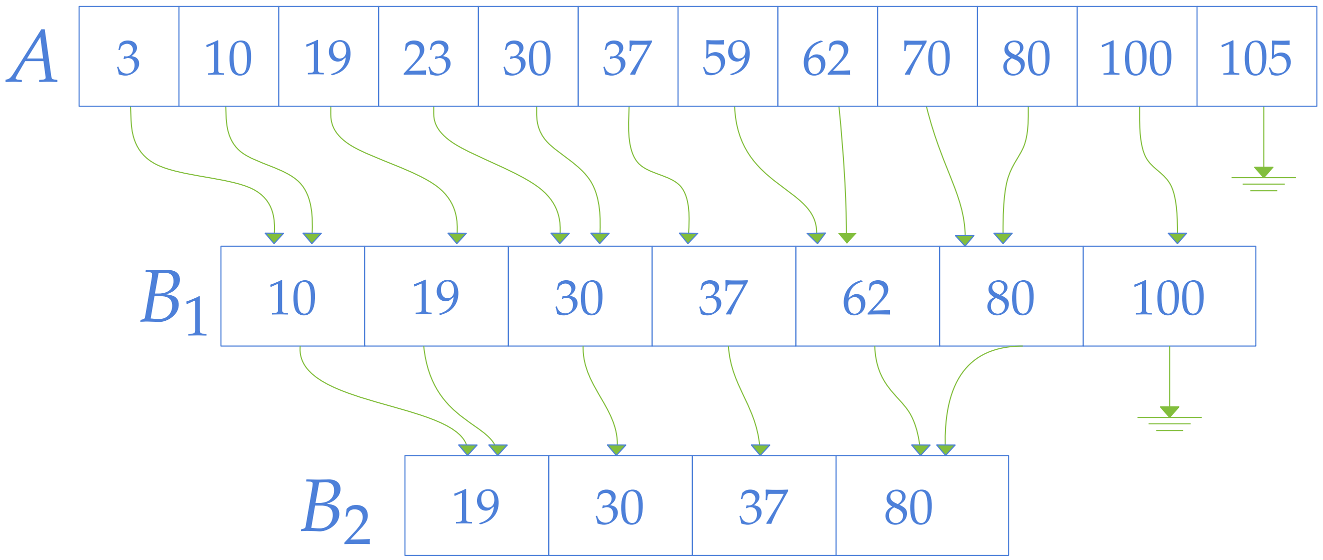
B_1						
10	19	30	37	62	80	100

B_2			
19	30	37	80

previously: fractional cascading

Task: Given sets $B_r \subseteq \cdots \subseteq B_2 \subseteq B_1 \subseteq A \subset \mathbb{R}$ stored in **sorted order** in **arrays** $A[1 \dots n], B_1[1 \dots m_1], B_2[1 \dots m_2], \dots, B_r[1 \dots m_r]$, perform a **1D range query** $[x : x']$ on the **multiset** $U = A \cup \left(\bigcup_1^r B_i\right)$ in $O(h + r + \log n)$ time (where $h = |[x : x'] \cap U|$)

key tool: we allow $n \log m_1 + \sum_{i=1}^{r-1} m_i \log m_{i+1}$ bits extra space



link $a \in A$ with smallest $b \geq a$ in B_1

link $b \in B_i$ with smallest $\hat{b} \geq b$ in B_{i+1}

previously: fractional cascading

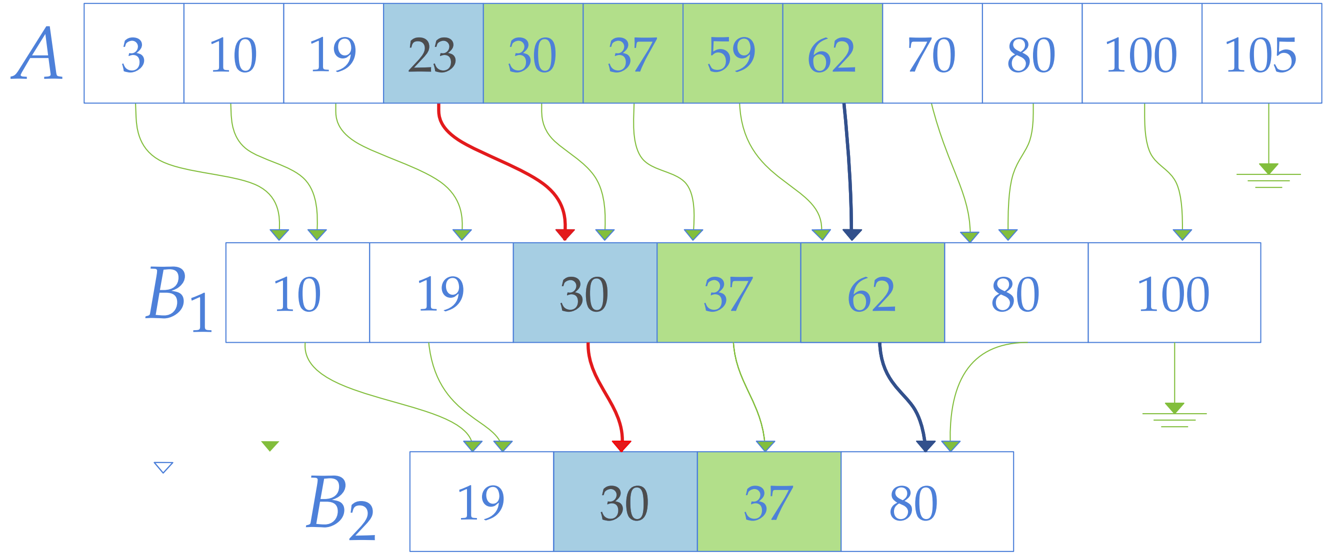
Task: Given sets $B_r \subseteq \cdots \subseteq B_2 \subseteq B_1 \subseteq A \subset \mathbb{R}$ stored in **sorted order** in **arrays** $A[1\ldots n], B_1[1\ldots m_1], B_2[1\ldots m_2], \ldots, B_r[1\ldots m_r]$, perform a **1D range query** $[x : x']$ on the **multiset** $U = A \cup \left(\bigcup_1^r B_i\right)$ in $O(h + r + \log n)$ time (where $h = |[x : x'] \cap U|$)

key tool: we allow $n \log m_1 + \sum_{i=1}^{r-1} m_i \log m_{i+1}$ bits extra space

Strategy:

1. perform binary search on A in $O(\log n)$ time to find the smallest element $x_a = 23$ in $A \cap [20 : 65]$
2. scan A from x_a and report $A \cap [20 : 65]$
3. follow pointer from $x_a \in A$ to $x_b = 30 \in B_1$
4. scan B_1 from x_b and report $B_1 \cap [20 : 65]$
5. repeat 3 and 4, for $B_2, \ldots, B_{r-1}$

note: $x_b :=$ min element in $B_1 \cap [20 : 65]$



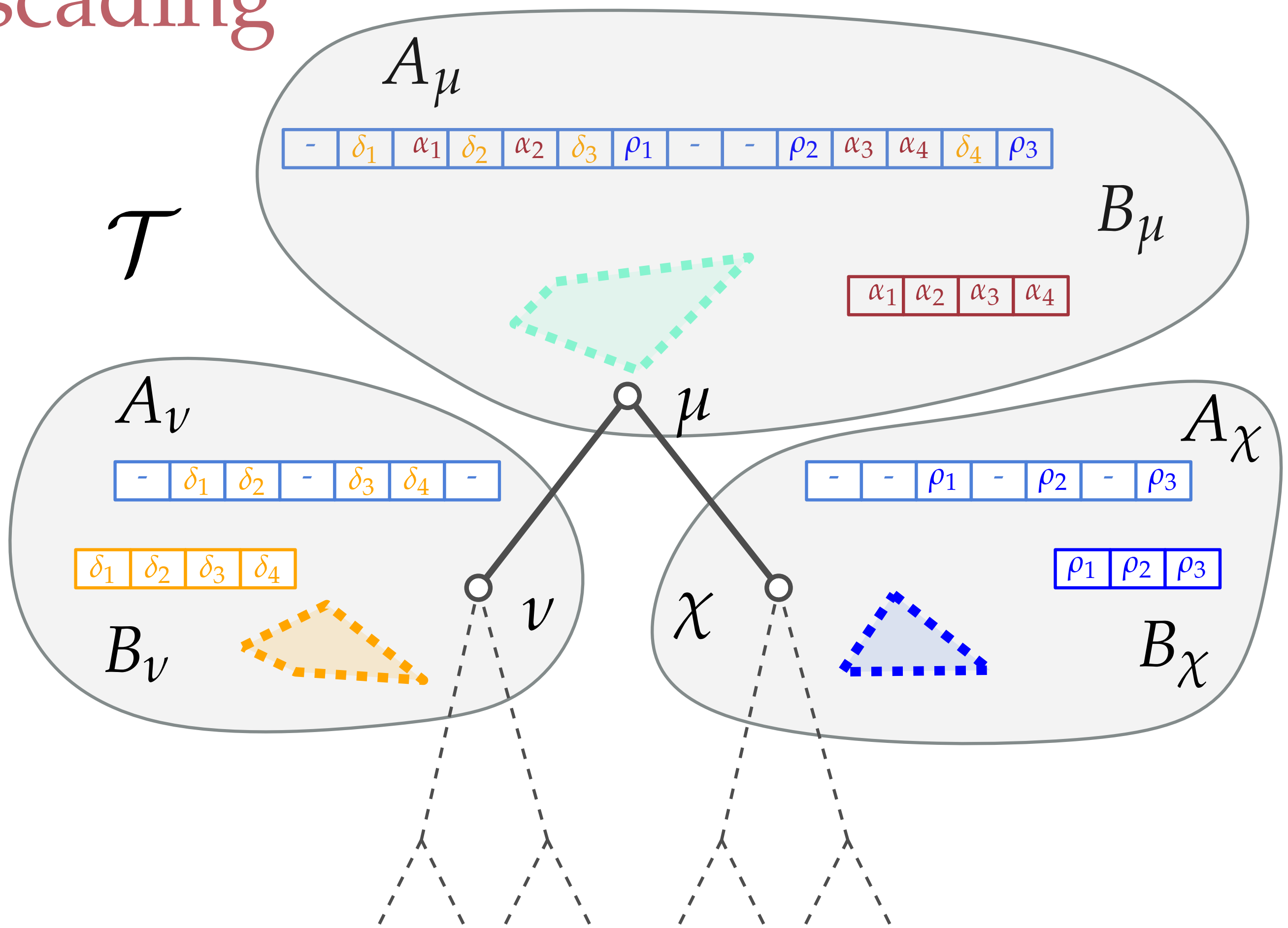
link $a \in A$ with smallest $b \geq a$ in B_1

link $b \in B_i$ with smallest $\hat{b} \geq b$ in B_{i+1}

example: query with range $[20 : 65]$

fractional cascading

The Data Structure:

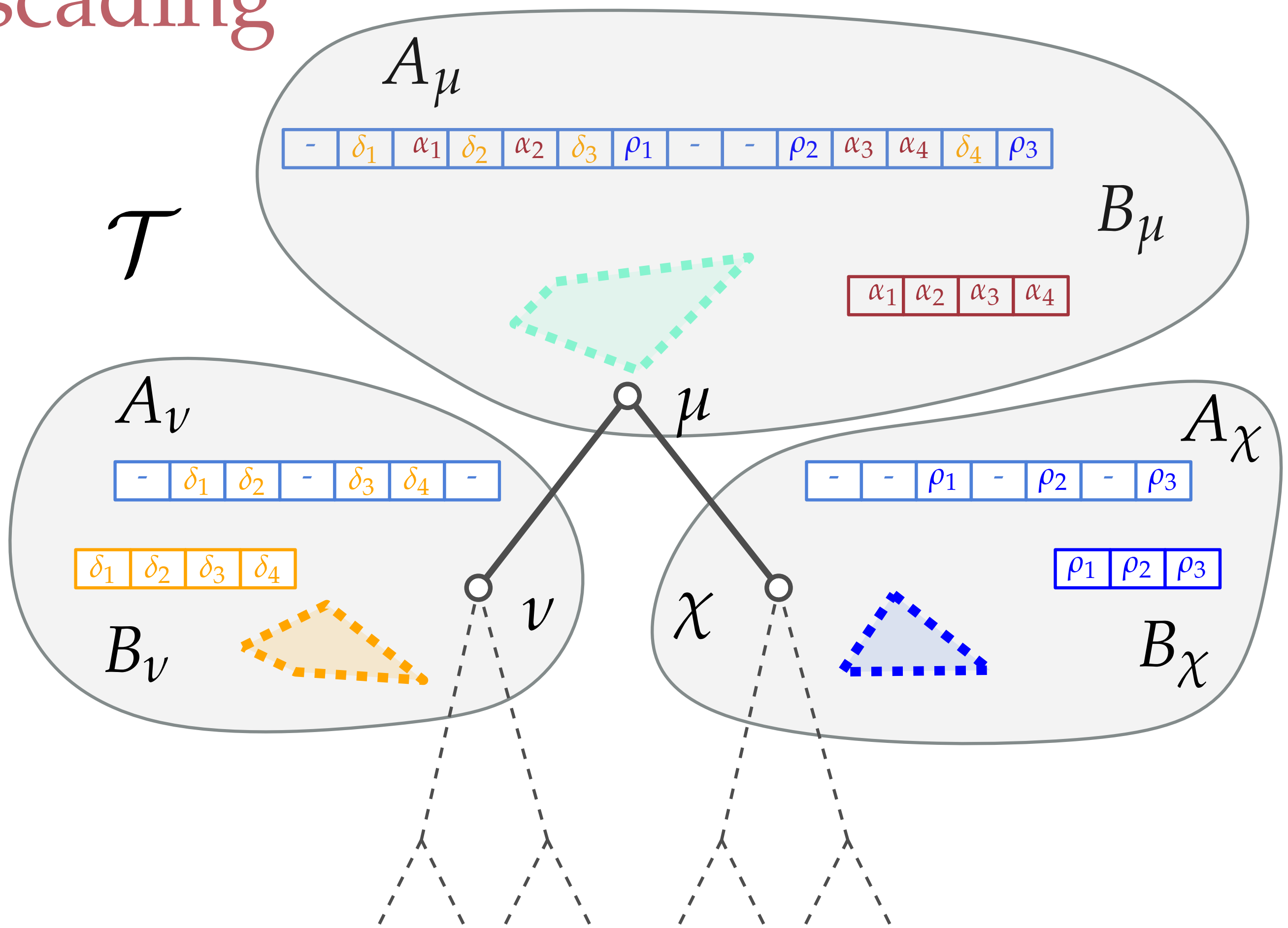


fractional cascading

THE DATA STRUCTURE:

Each **node** μ of $\mathcal{T}$, associated with the **convex hull** C_μ of a polygonal path, is equipped with:

- a **sorted array** A_μ of **all the slopes** occurring in the convex hulls associated with the nodes of the subtree rooted at μ
- a **sorted array** B_μ containing **the slope-sequence of** C_μ



fractional cascading

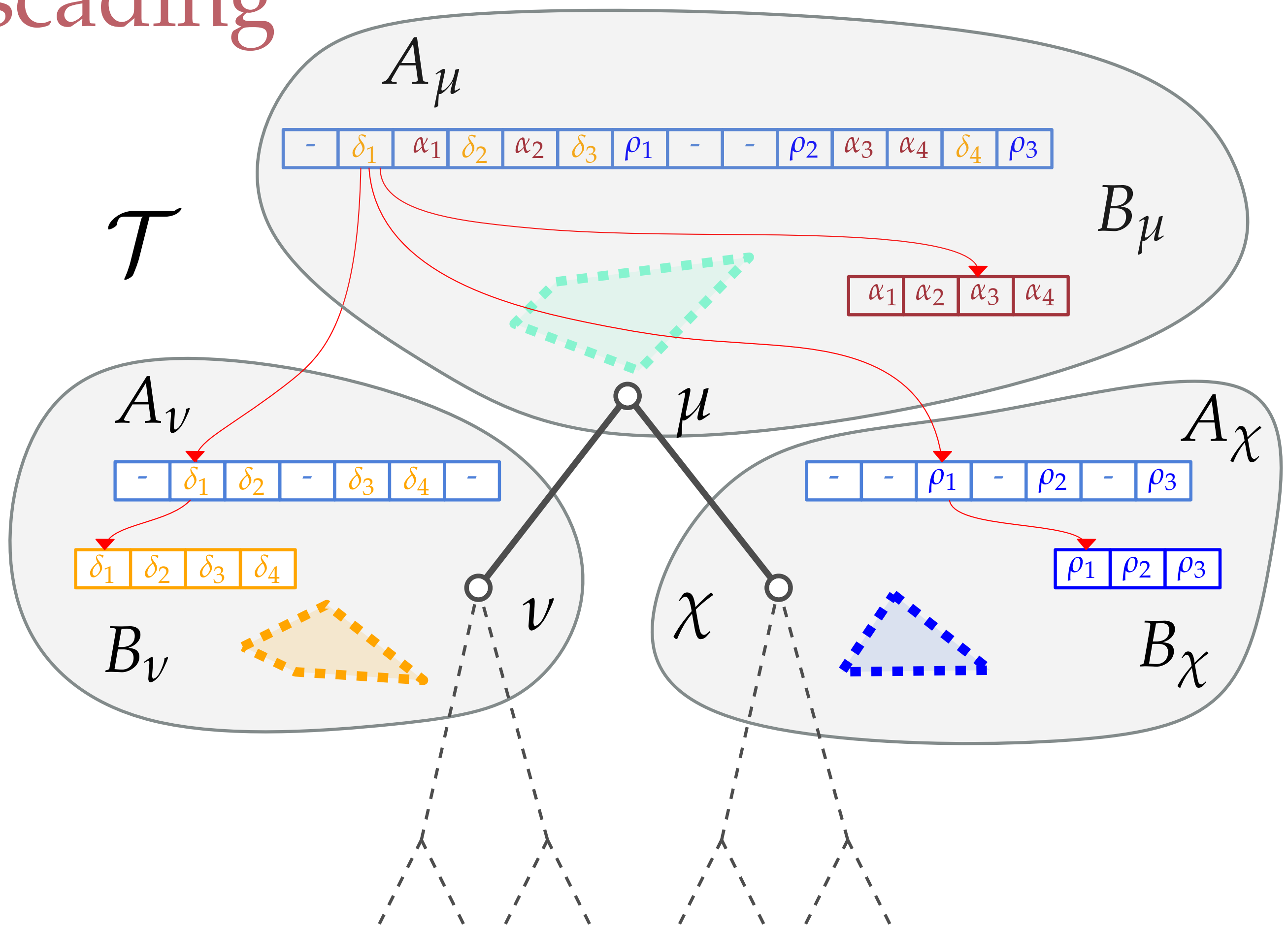
The Data Structure:

Each **node** μ of $\mathcal{T}$, associated with the **convex hull** C_μ of a polygonal path, is equipped with:

- a **sorted array** A_μ of **all the slopes** occurring in the convex hulls associated with the nodes of the subtree rooted at μ
- a **sorted array** B_μ containing **the slope-sequence of** C_μ

Consider a node μ with children ν and χ.
Each entry x **of** A_μ has **a pointer to**:

- the smallest entry $\alpha_i \geq x$ of B_μ
- the smallest entry $\delta_i \geq x$ of A_ν
- the smallest entry $\rho_i \geq x$ of A_χ

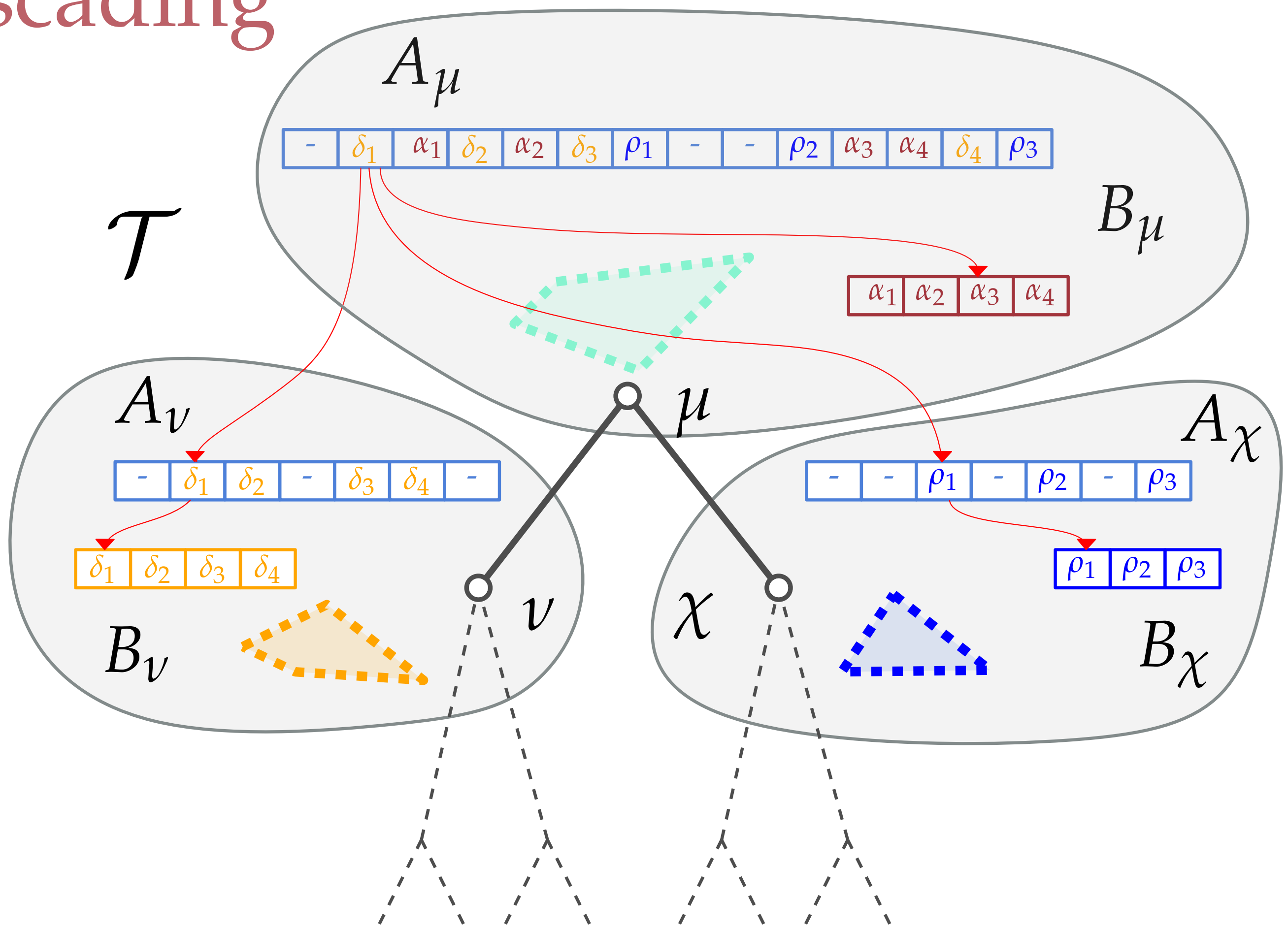


Space: still only $O(n \log n)$

fractional cascading

QUERY TIME:

- The **data structure** allows us to implement all **the (convex polygon, line) intersection tests** except for the one at the root, in $O(1)$ time per test.
  - ◇ By Lemma 2, to decide whether to descend into the subtree rooted at μ, we just look up the slopes s' and s'' of ℓ in B_μ.
- There is an $O(\log n)$ cost **at the root** of $\mathcal{T}$ to get the whole process started.

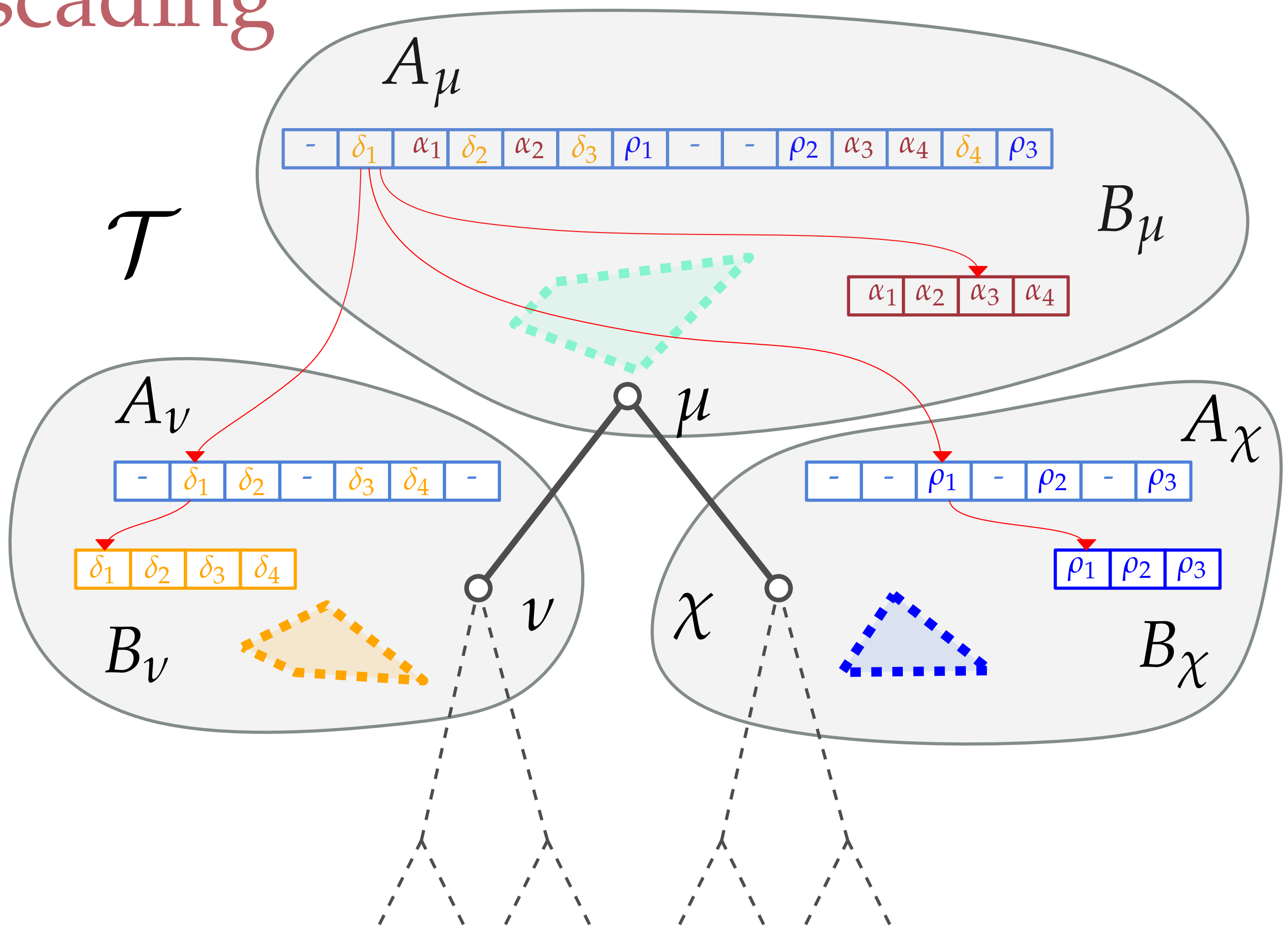


Query time:
$O(\log n + \text{size of subtree of } \mathcal{T} \text{ actually visited})$

fractional cascading

QUERY TIME:

- The **data structure** allows us to implement all **the (convex polygon, line) intersection tests** except for the one at the root, in $O(1)$ time per test.
  - By Lemma 2, to decide whether to descend into the subtree rooted at μ, we just look up the slopes s' and s'' of ℓ in B_μ.
- There is an $O(\log n)$ cost **at the root** of $\mathcal{T}$ to get the whole process started.



Query time:

$O(\log n + \underbrace{\text{size of subtree of } \mathcal{T} \text{ actually visited}})$

proof of the query time

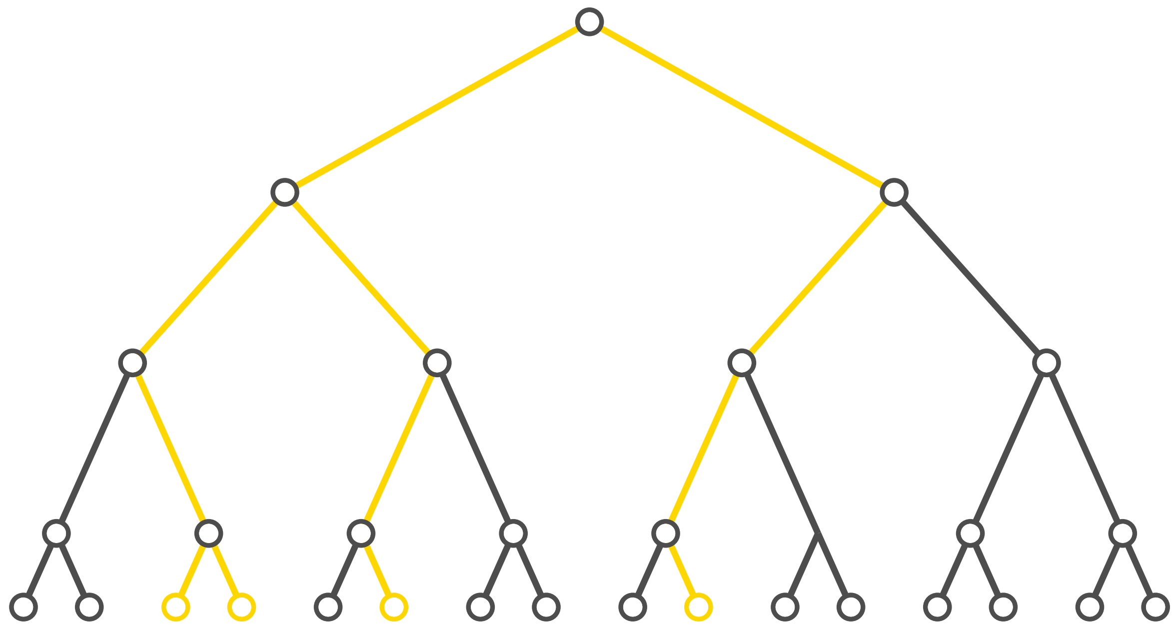
Our claimed query time bound of $O((h+1)\log[n/(h+1)])$ follows from the next lemma.

Lemma 3

Let $\mathcal{T}$ be a perfectly balanced tree on n leaves and consider any subtree $\mathcal{S}$ of $\mathcal{T}$ with h leaves chosen among the leaves of $\mathcal{T}$. Then,

$$|\mathcal{S}| \leq h(\lceil \log n \rceil - \lfloor \log h \rfloor) + 2h - 1.$$

Proof:



proof of the query time

Our claimed query time bound of $O((h+1)\log[n/(h+1)])$ follows from the next lemma.

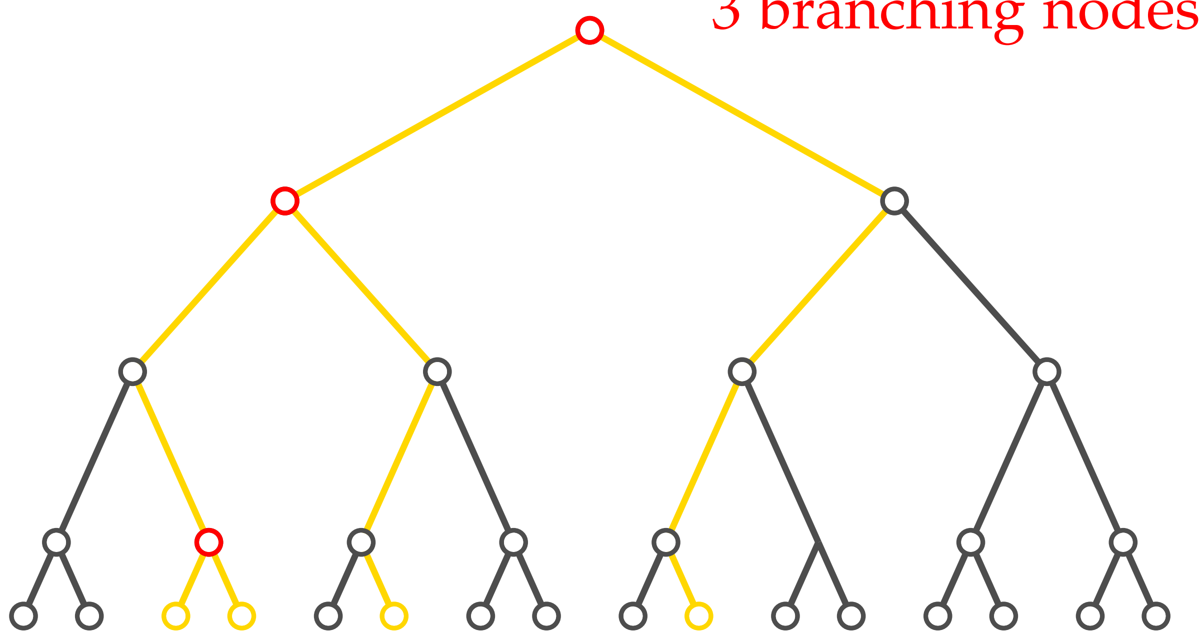
Lemma 3

Let $\mathcal{T}$ be a perfectly balanced tree on n leaves and consider any subtree $\mathcal{S}$ of $\mathcal{T}$ with h leaves chosen among the leaves of $\mathcal{T}$. Then,

$$|\mathcal{S}| \leq h(\lceil \log n \rceil - \lfloor \log h \rfloor) + 2h - 1.$$

Proof:

- In $\mathcal{S}$ there are h leaves and $h-1$ branching nodes (outdegree 2).



proof of the query time

Our claimed query time bound of $O((h+1)\log[n/(h+1)])$ follows from the next lemma.

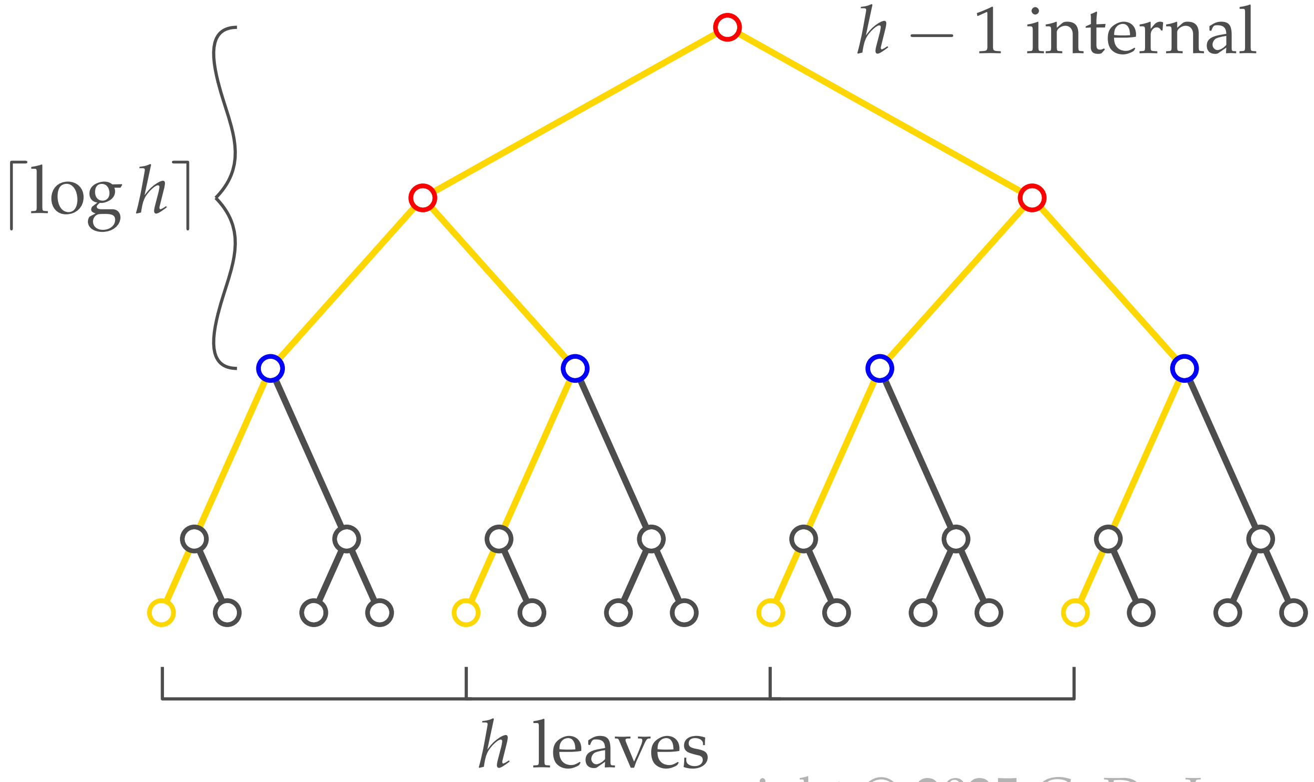
Lemma 3

Let $\mathcal{T}$ be a perfectly balanced tree on n leaves and consider any subtree $\mathcal{S}$ of $\mathcal{T}$ with h leaves chosen among the leaves of $\mathcal{T}$. Then,

$$|\mathcal{S}| \leq h(\lceil \log n \rceil - \lfloor \log h \rfloor) + 2h - 1.$$

Proof:

- In $\mathcal{S}$ there are h leaves and $h-1$ branching nodes (outdegree 2).
- $|\mathcal{S}|$ is maximized when all the **branching nodes occur as high in $\mathcal{T}$ as possible**.



proof of the query time

Our claimed query time bound of $O((h+1)\log[n/(h+1)])$ follows from the next lemma.

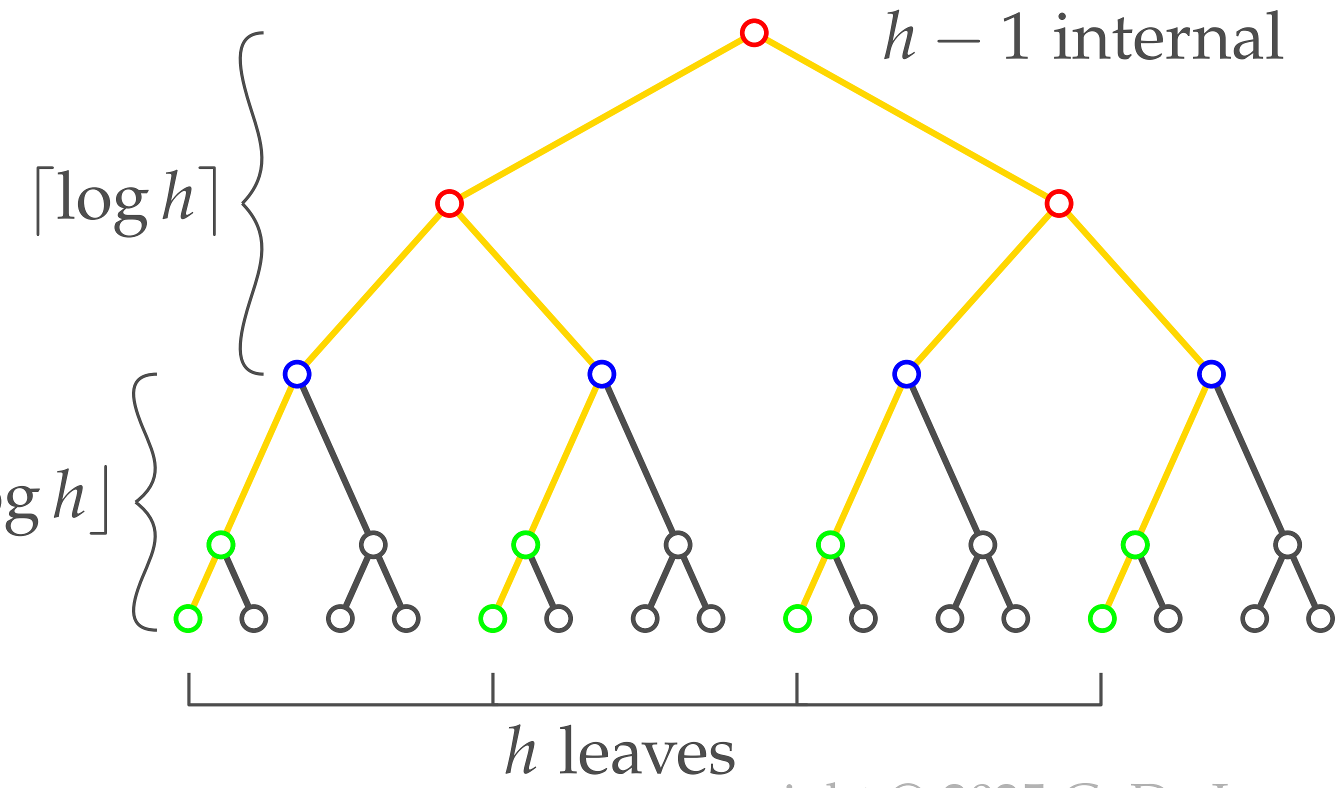
Lemma 3

Let $\mathcal{T}$ be a perfectly balanced tree on n leaves and consider any subtree $\mathcal{S}$ of $\mathcal{T}$ with h leaves chosen among the leaves of $\mathcal{T}$. Then,

$$|\mathcal{S}| \leq h(\lceil \log n \rceil - \lfloor \log h \rfloor) + 2h - 1.$$

Proof:

- In $\mathcal{S}$ there are h leaves and $h-1$ branching nodes (outdegree 2).
- $|\mathcal{S}|$ is maximized when all the **branching nodes occur as high in $\mathcal{T}$ as possible**.
- Then the number of remaining **nonbranching nodes** in $\mathcal{S}$ is at most $h(\lceil \log n \rceil - \lfloor \log h \rfloor)$. $\square$



final result

Theorem

Given a polygonai path P of length n, it is possible in time $O(n \log n)$ to build a data structure of size $O(n \log n)$, so that given any line ℓ, if ℓ intersects P in h **edges**, then these edges can be found and reported in time $O((h+1) \log[n/(h+1)])$.

final result

Theorem

Given a polygonai path P of length n, it is possible in time $O(n \log n)$ to build a data structure of size $O(n \log n)$, so that given any line ℓ, if ℓ intersects P in h **edges**, then these edges can be reported in time $O((h+1)\log[n/(h+1)])$.

Query time:

$O(\log n + \underline{\text{size of subtree of } \mathcal{T} \text{ actually visited}})$

$$\leq h(\lceil \log n \rceil - \lfloor \log h \rfloor) + 2h - 1$$

$$= h(\lceil \log n \rceil - \lfloor \log h \rfloor + 2) - 1$$

$$< h(\lceil \log n \rceil - \lfloor \log h \rfloor + \log 4)$$

$$\in O(h(\lceil \log \tfrac{n}{h+1} \rceil)) \qquad \square$$

references

[dBCvKO08] Mark de Berg, Otfried Cheong, Marc J. van Kreveld, Mark H. Overmars: Computational geometry: algorithms and applications, 3rd Edition. Springer 2008, ISBN 9783540779735, pp. I-XII, 1-386

[K20] Philipp Kindermann. Youtube channel on Computational Geometry. https://www.youtube.com/channel/UCuAzKw_VngkAsQh7ummYq0A

[CG86a] Bernard Chazelle, Leonidas J. Guibas: Fractional Cascading: I. A Data Structuring Technique. Algorithmica 1(2): 133-162 (1986)

[CG86b] Bernard Chazelle, Leonidas J. Guibas: Fractional Cascading: II. Applications. Algorithmica 1(2): 163-191 (1986)

[PH] Pirzadeh, Hormoz: Computational geometry with the rotating calipers. PhD Thesis (1999). https://escholarship.mcgill.ca/concern/theses/fx719p46g