Improved Dynamic Geodesic Nearest Neighbor Searching in a Simple Polygon

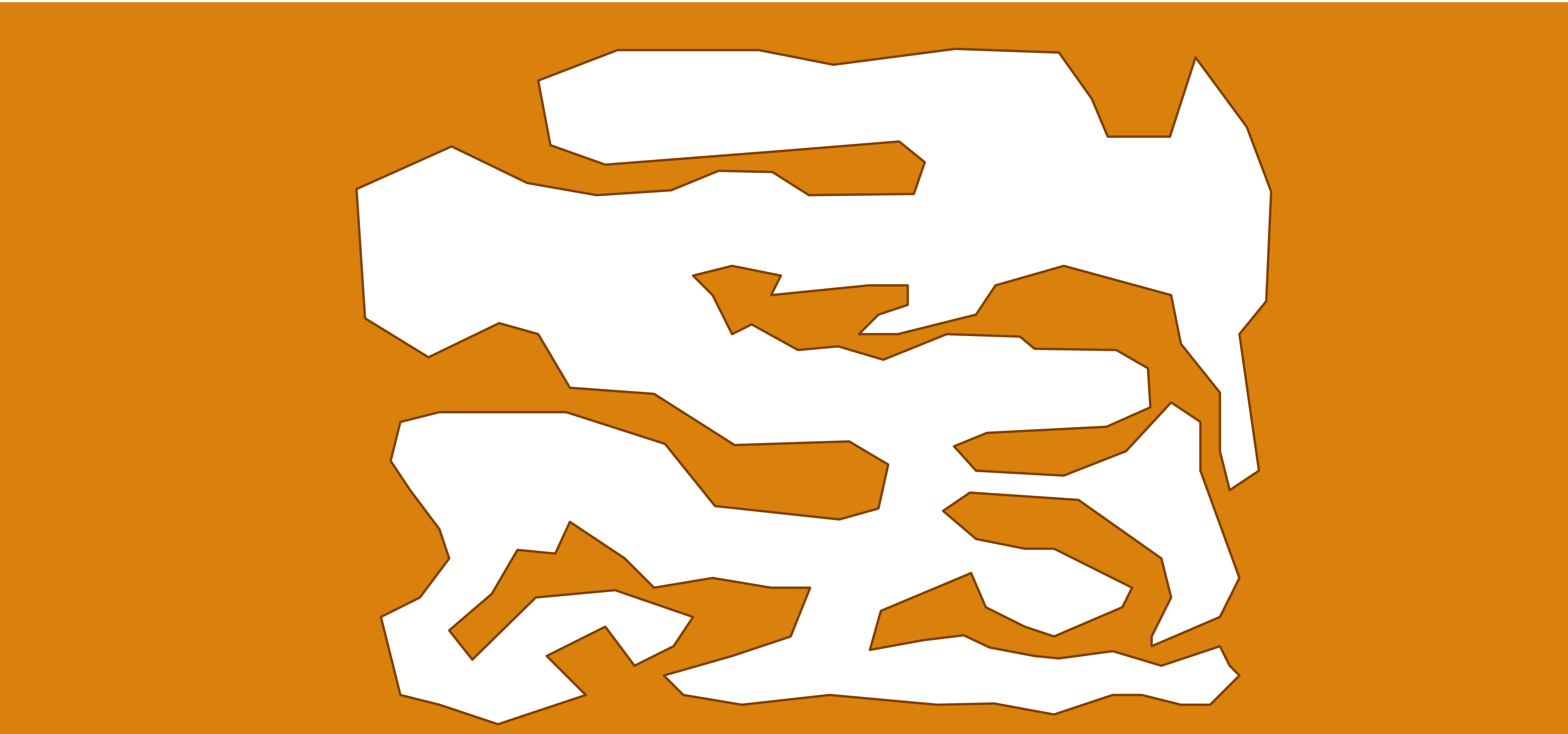
Lars Arge

Pankaj Agarwal

Frank Staals

The Problem

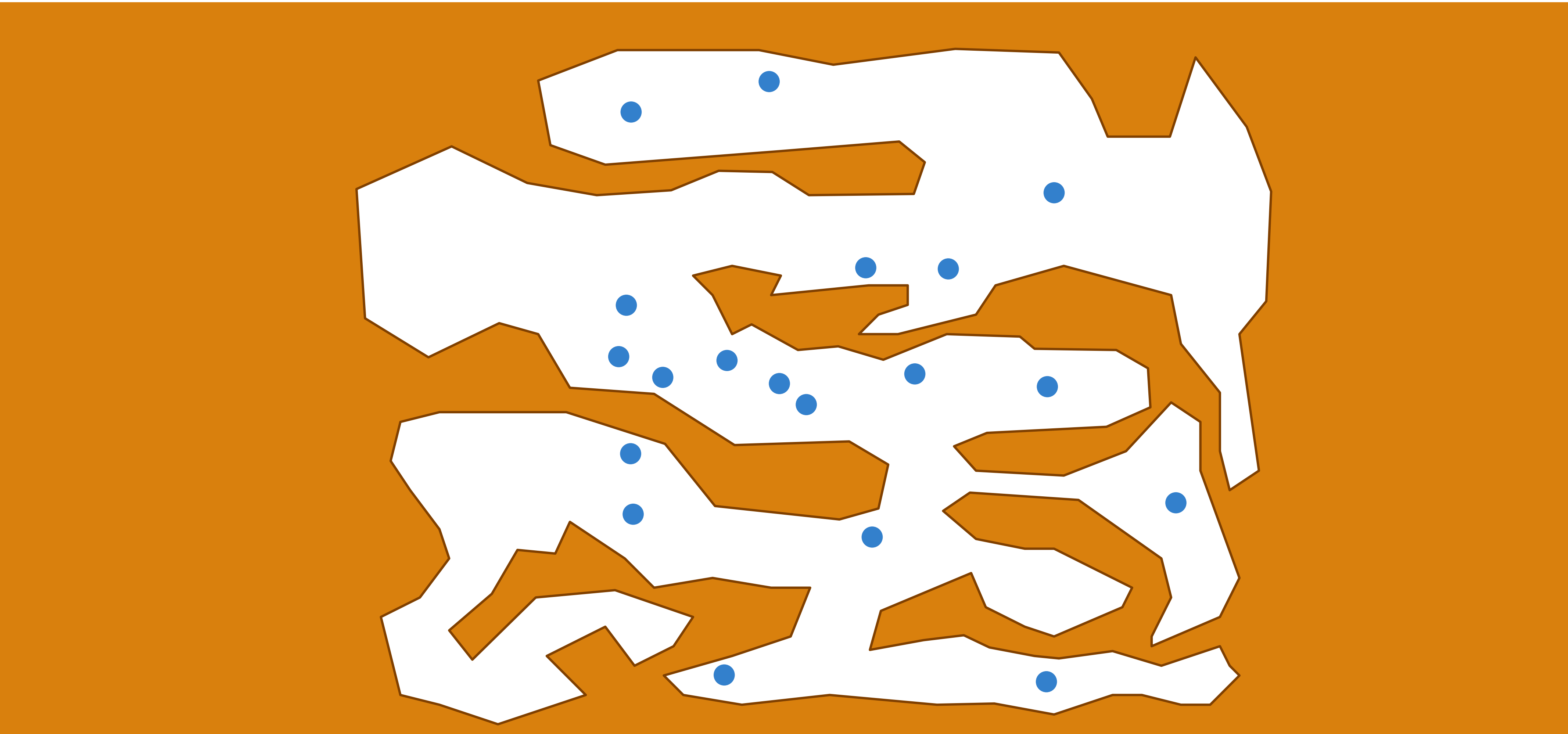
Given: P: simple polygon



The Problem

Given: P: simple polygon, S: dynamic set of point sites inside P

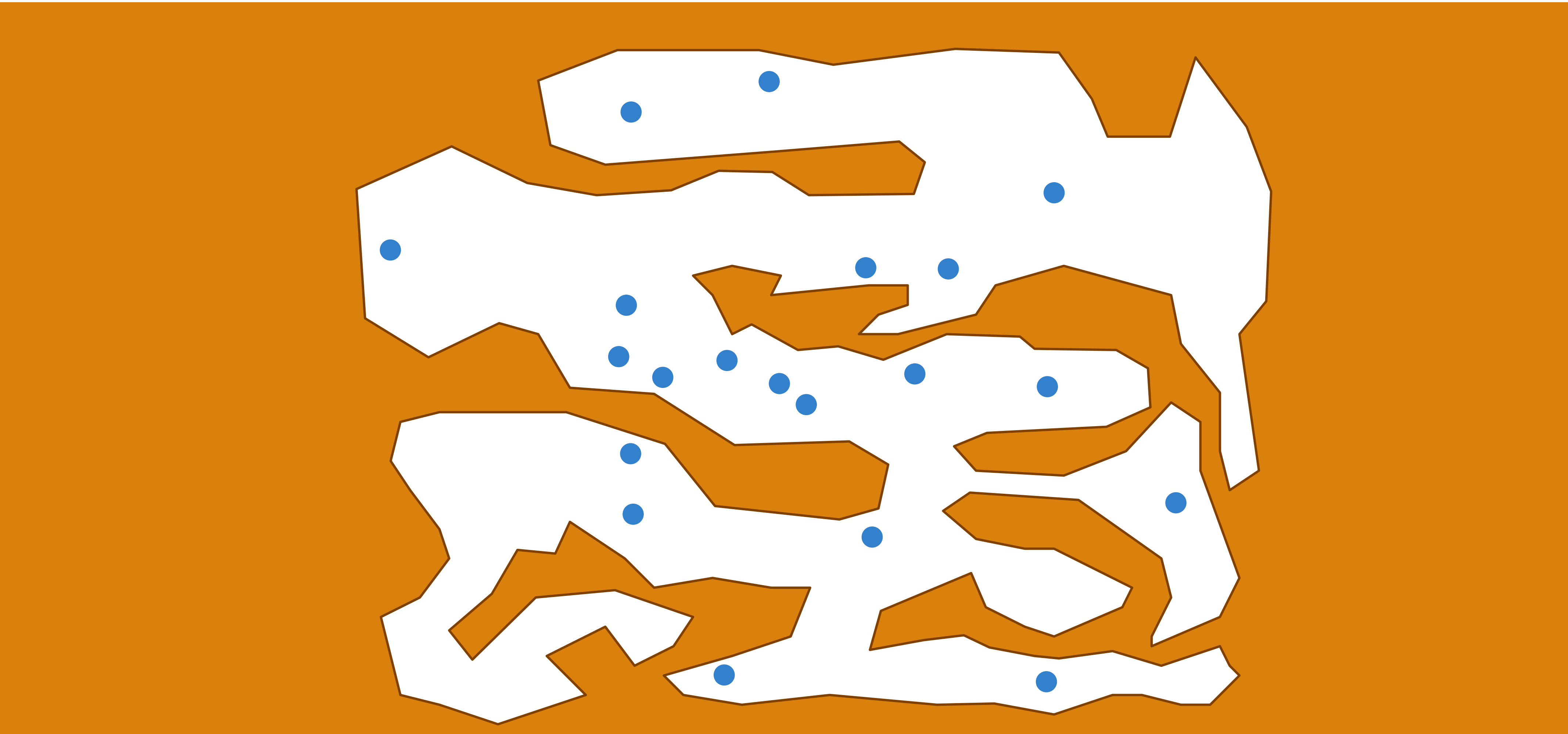
Problem: store S s.t. we can



The Problem

Given: P: simple polygon, S: dynamic set of point sites inside P

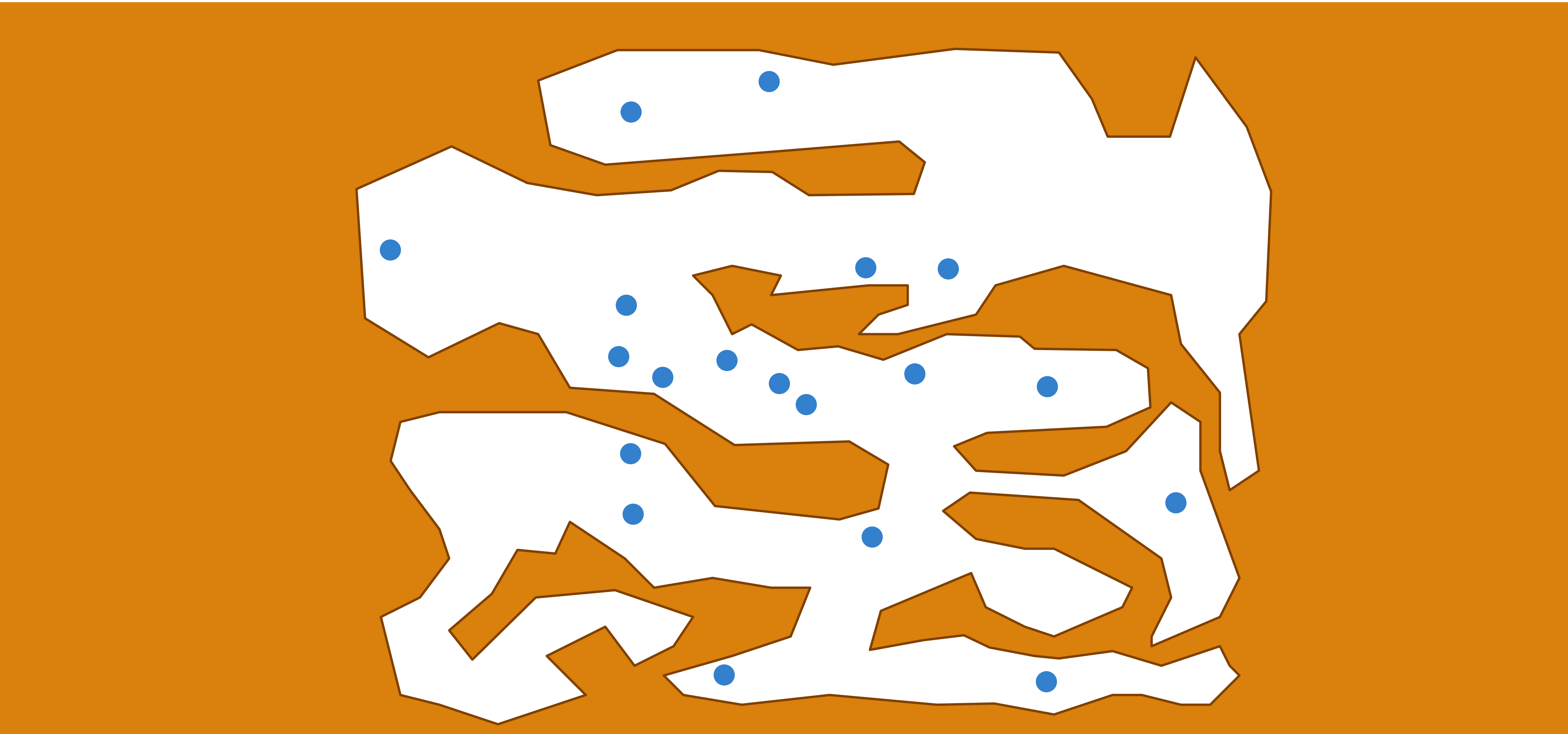
Problem: store S s.t. we can
insert a site,



The Problem

Given: P: simple polygon, S: dynamic set of point sites inside P

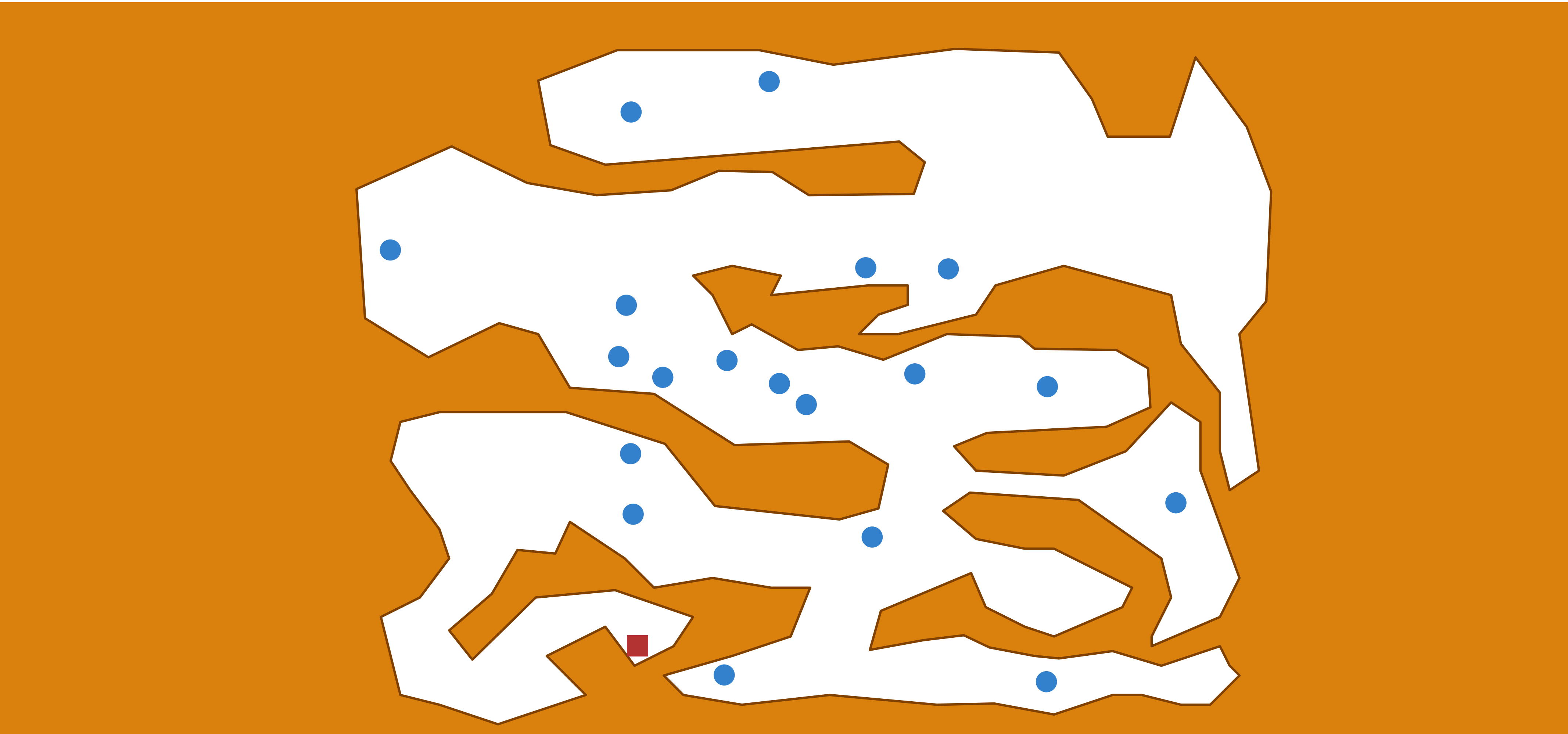
Problem: store S s.t. we can
insert a site,
delete a site,



The Problem

Given: P: simple polygon, S: dynamic set of point sites inside P

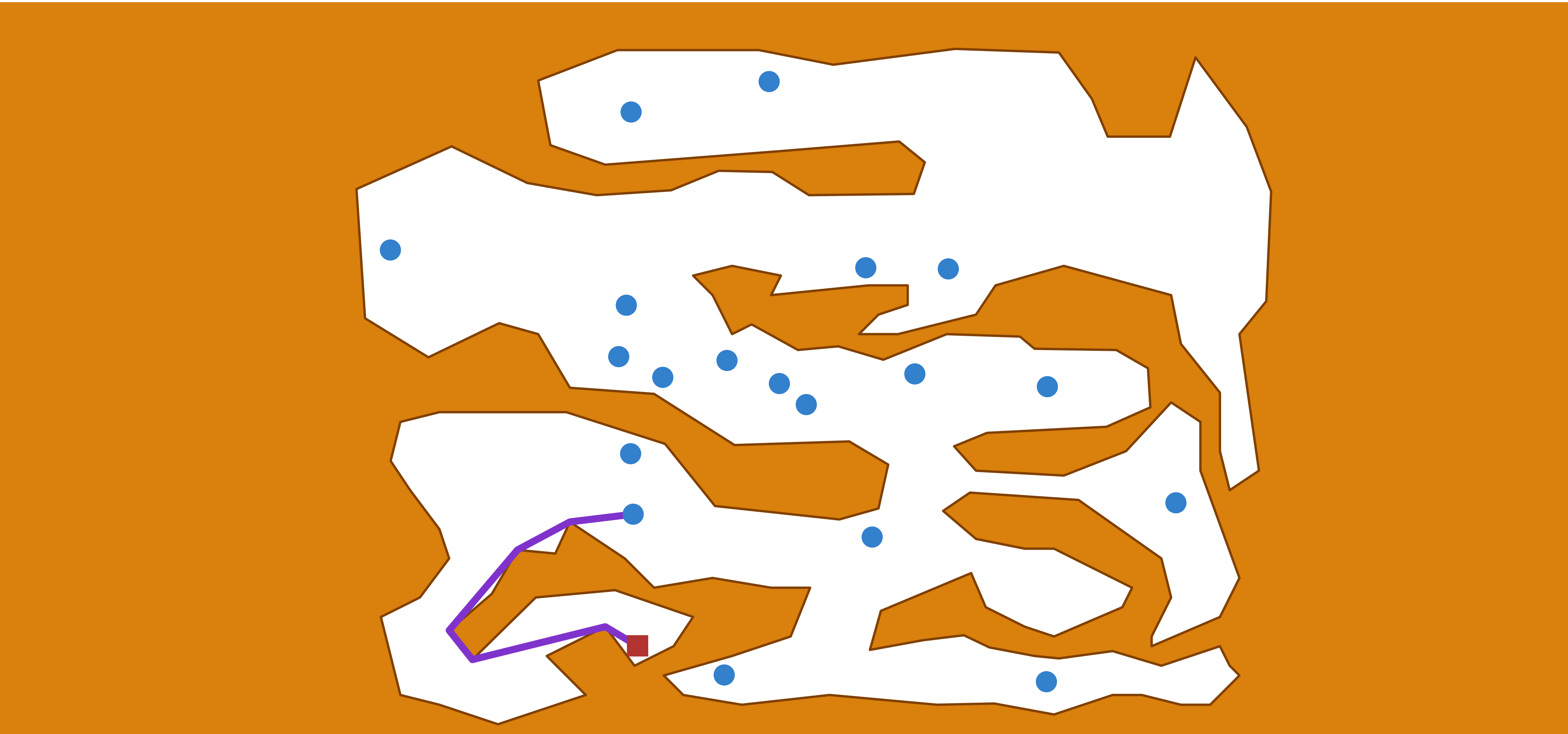
Problem: store S s.t. we can
insert a site,
delete a site,
find the site $s \in S$ closest to a query point q



The Problem

Given: P: simple polygon, S: dynamic set of point sites inside P

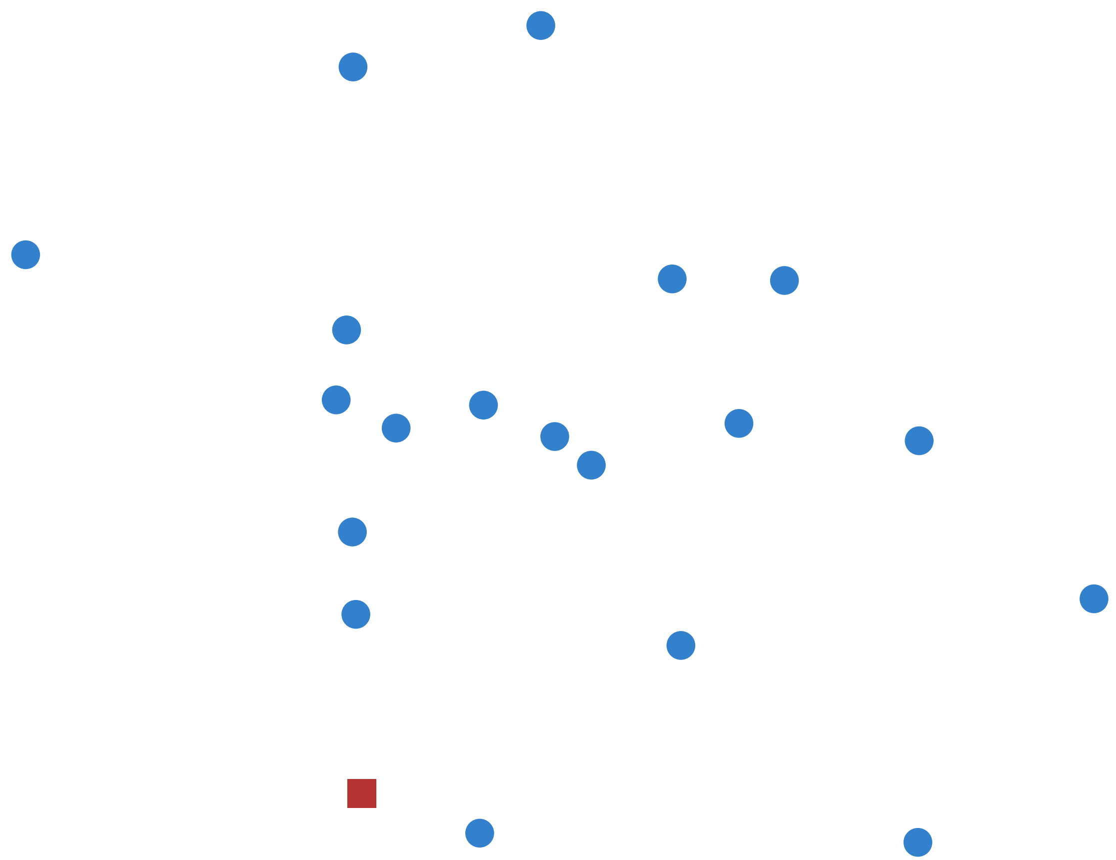
Problem: store S s.t. we can
insert a site,
delete a site,
find the site $s \in S$ closest to a query point q
we measure the **geodesic** distance between s and q



Euclidean Dynamic NN Searching

Given: S: dynamic set of n point sites in $\mathbb{R}^2$

Problem: store S s.t. we can
insert a site,
delete a site,
find the site $s \in S$ closest to a query point q



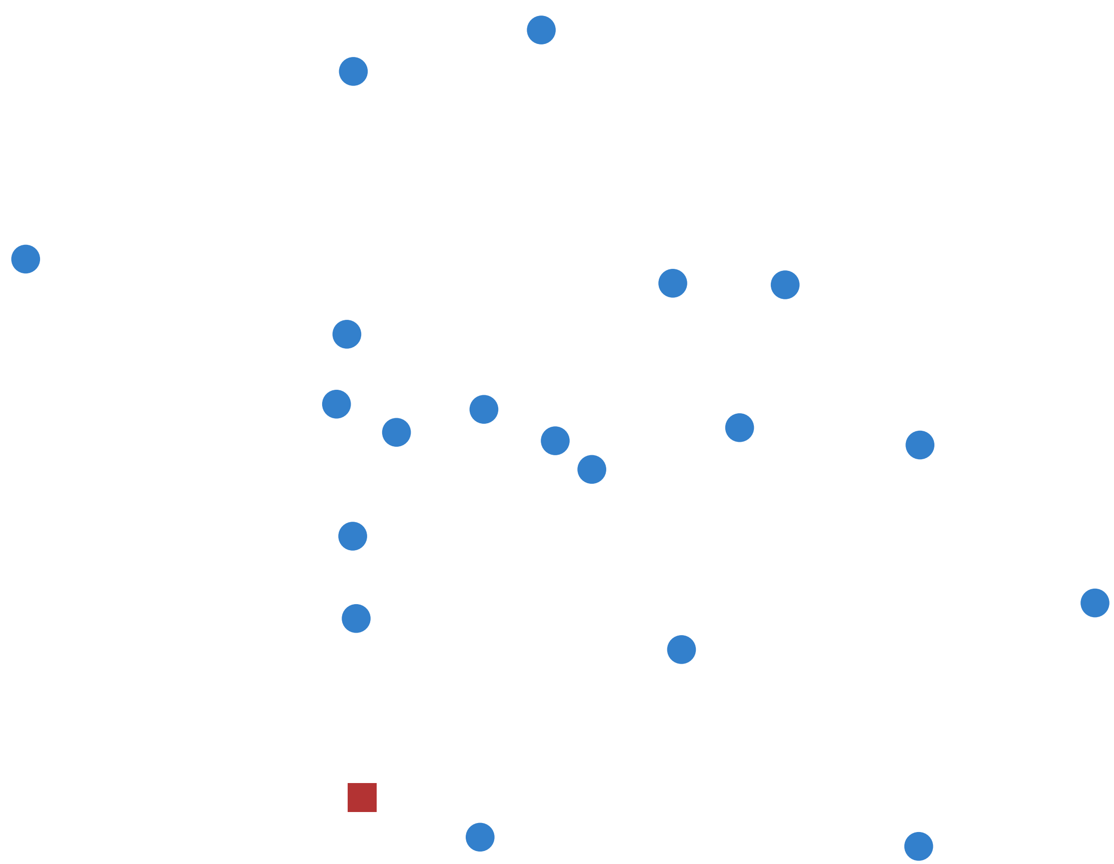
Euclidean Dynamic NN Searching

Given: S: dynamic set of n point sites in $\mathbb{R}^2$

Problem: store S s.t. we can
insert a site,
delete a site,
find the site $s \in S$ closest to a query point q

$$\left.\begin{matrix} O(\log^3 n) \\ O(\log^5 n) \end{matrix}\right\} \text{amortized}$$
$$O(\log^2 n)$$

[Chan, JACM '10], [Kaplan et al., SODA '17]



What's the Problem?

Obs: Nearest Neighbor Searching is a **decomposable search problem**

split S into $O(\sqrt{n})$ groups of size $O(\sqrt{n})$,

build	Voronoi diagram for each
query	$O(\sqrt{n} \log n)$
update	$O(\sqrt{n} \log n)$

n = max #sites in S
m = complexity P

What's the Problem?

Obs: Nearest Neighbor Searching is a **decomposable search problem**

split S into $O(\sqrt{n})$ groups of size $O(\sqrt{n})$,
build geodesic Voronoi diagram for each

query	$O(\sqrt{n}\log(n+m))$
update	$O((\sqrt{n}+m)\log(n+m))$

Problem: geodesic Voronoi Diagram has size $\Theta(n+m)$

n = max #sites in S
m = complexity P

What's the Problem?

Obs: Nearest Neighbor Searching is a **decomposable search problem**

split S into $O(\sqrt{n})$ groups of size $O(\sqrt{n})$,
build geodesic Voronoi diagram for each

query $O(\sqrt{n} \log(n+m))$

update $O(\sqrt{n} \log n \log^2 m)$ [Oh & Ahn, SoCG '17]

Problem: geodesic Voronoi Diagram has size $\Theta(n+m)$

n = max #sites in S
m = complexity P

Results

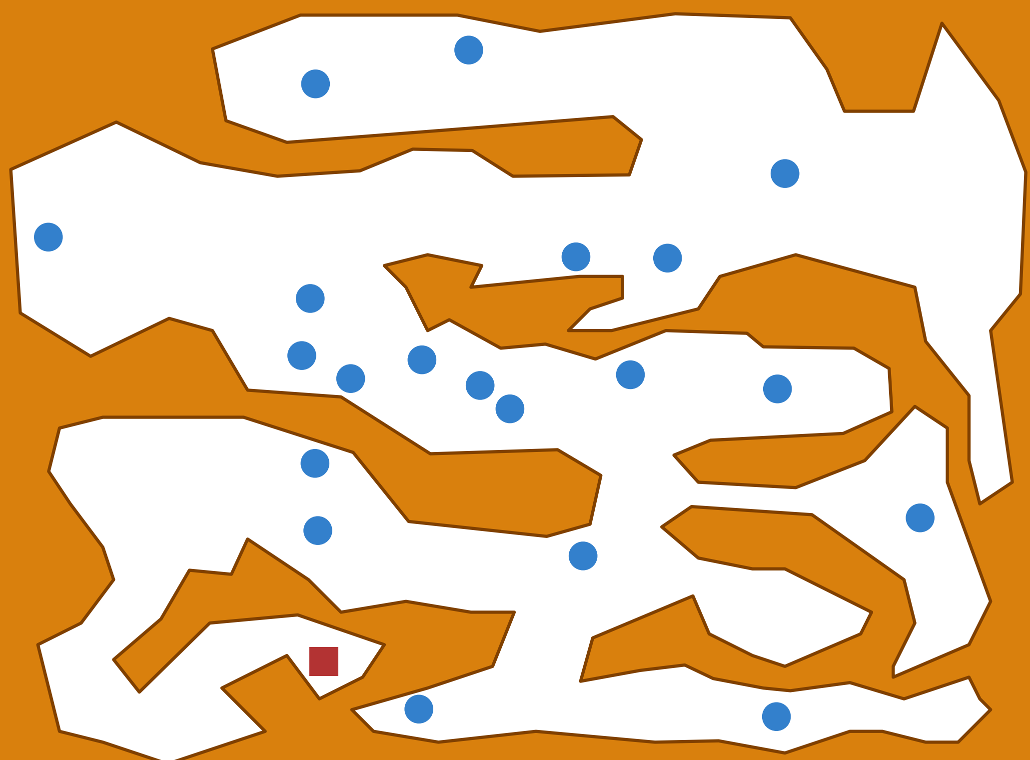
Given: P: simple polygon, S: dynamic set of point sites inside P

The data structure supports:

insert	$O(\log^7(n+m))$	expected, amortized
delete	$O(\log^9(n+m))$	expected, amortized
query	$O(\log^4(n+m))$	

n = max #sites in S
m = complexity P

expected space: $O(m + n\log^3 n \log m)$



Results

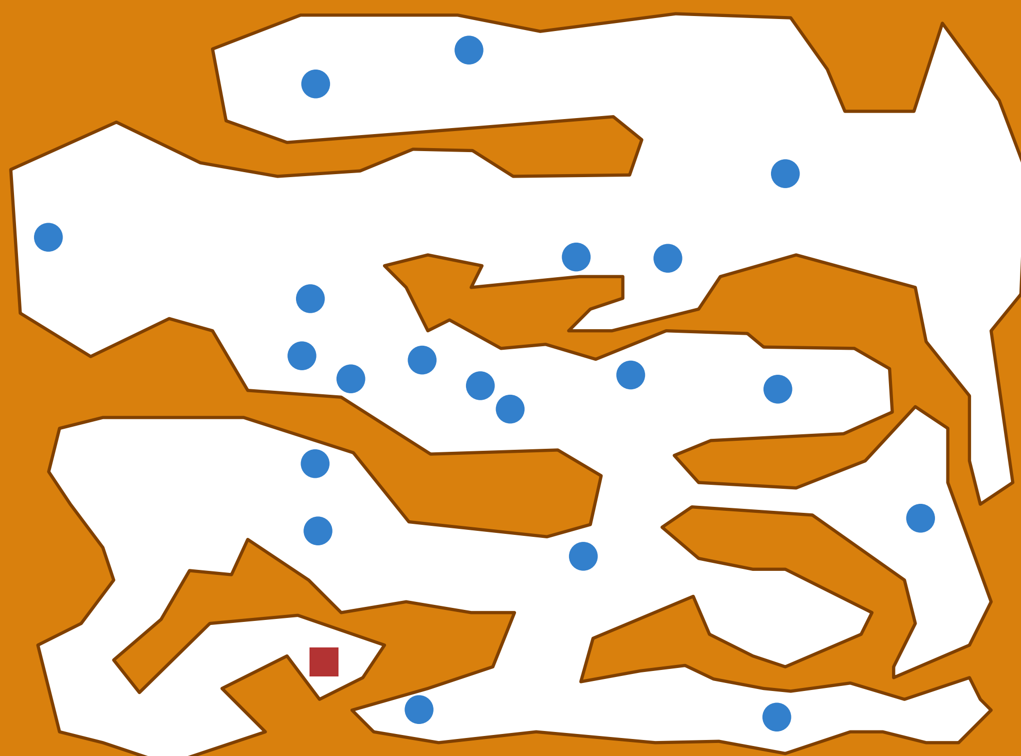
Given: P: simple polygon, S: dynamic set of point sites inside P
sequence of updates

The data structure supports:

insert	$O(\log^4(n+m))$	amortized
delete	$O(\log^4(n+m))$	amortized
query	$O(\log^4(n+m))$	

space: $O(m + n \log n \log m)$

n = max #sites in S
m = complexity P



Results

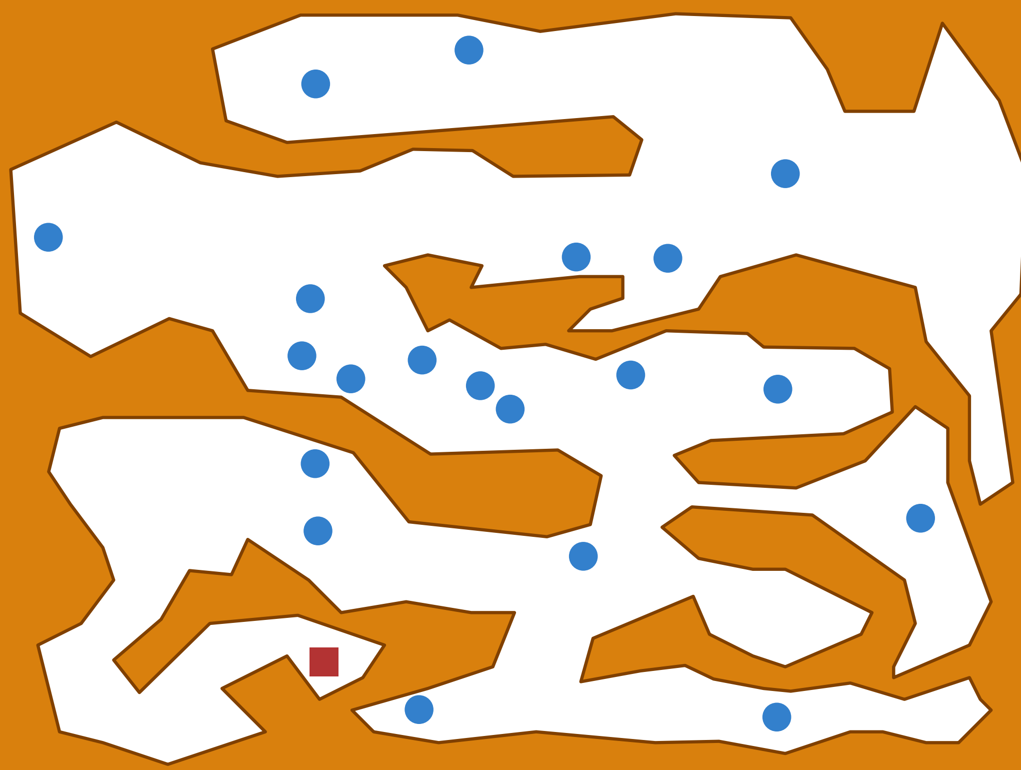
Given: P: simple polygon, S: dynamic set of point sites inside P

The data structure supports:

insert	$O(\log^4(n+m))$	amortized
query	$O(\log^4(n+m))$	

space: $O(m + n \log m)$

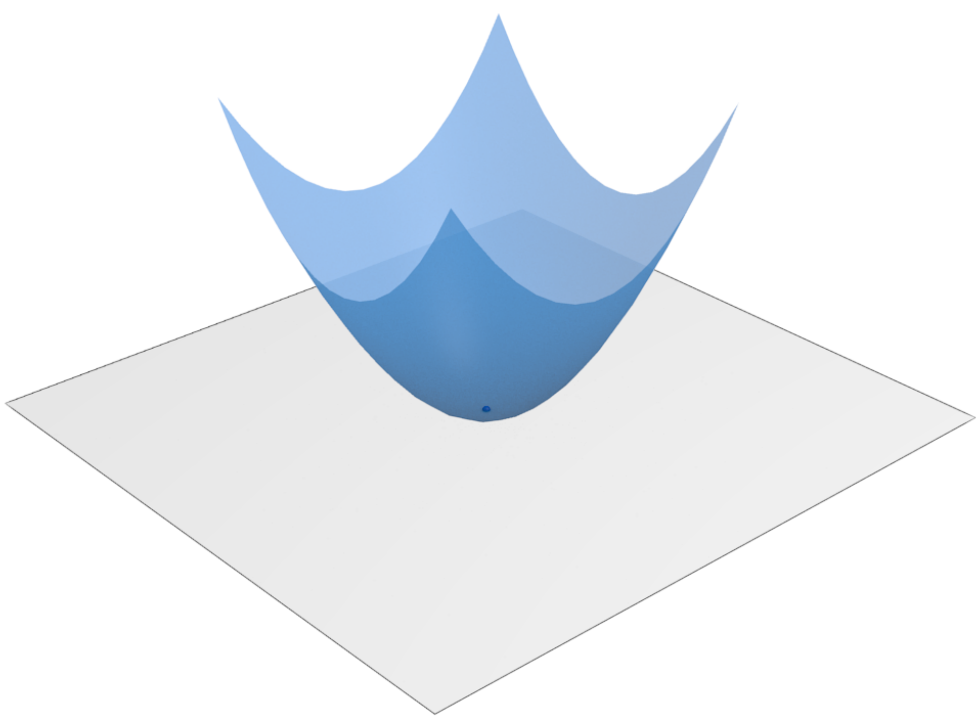
n = max #sites in S
m = complexity P



How?

Strategy in the Kaplan et al. approach:

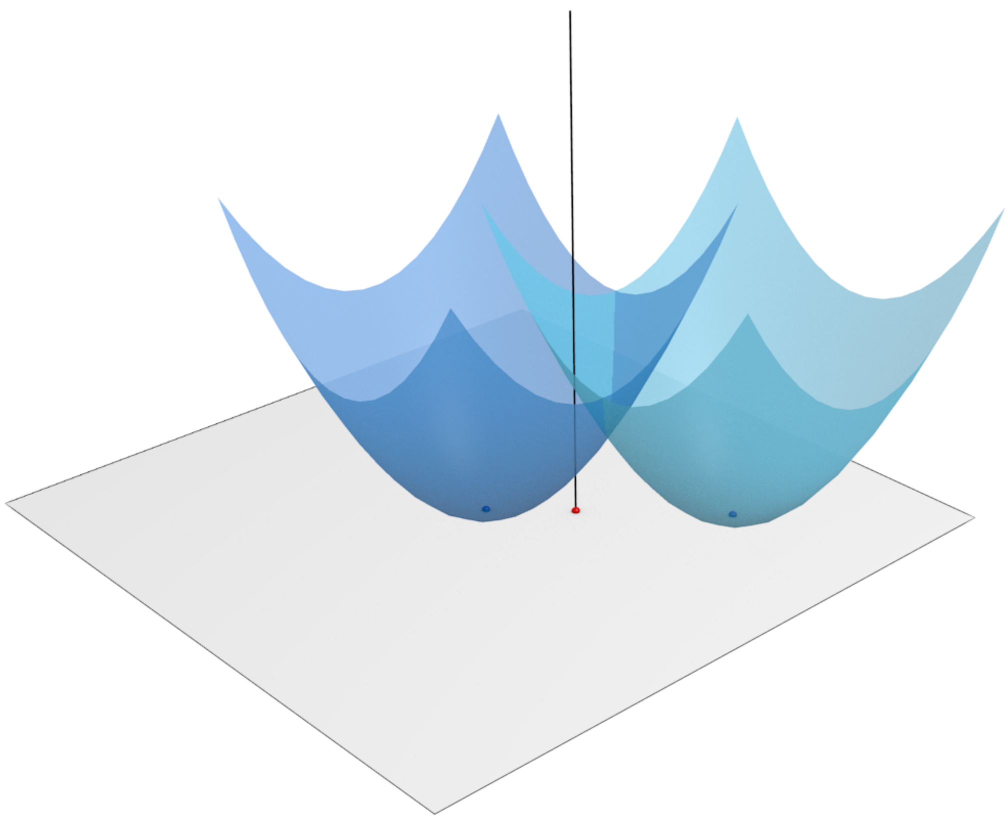
1. Consider lower envelope of distance functions F



How?

Strategy in the Kaplan et al. approach:

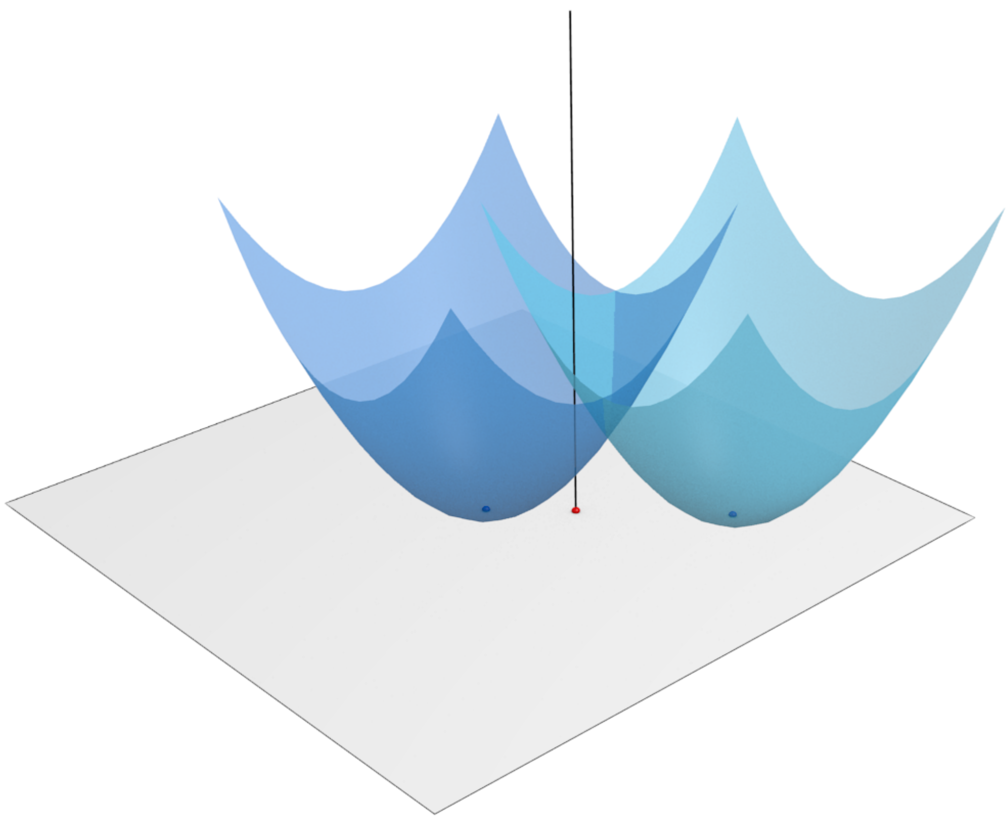
1. Consider lower envelope of distance functions F



How?

Strategy in the Kaplan et al. approach:

1. Consider lower envelope of distance functions F
2. Design algorithm $\mathcal{A}$ to compute a k-shallow cutting $\Lambda_k(F)$



How?

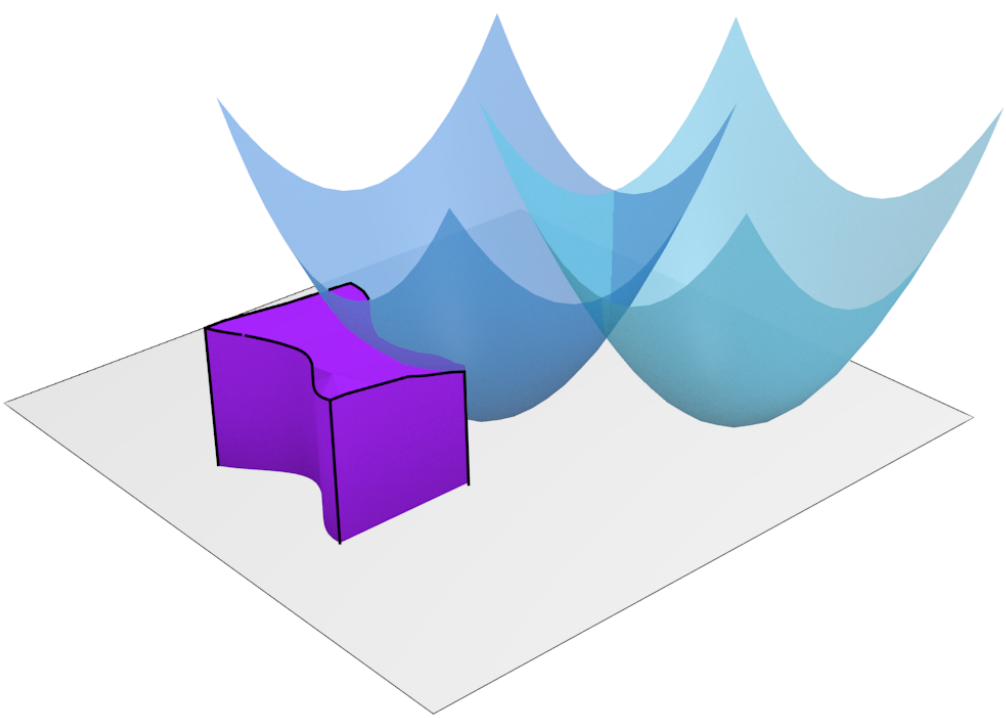
Strategy in the Kaplan et al. approach:

1. Consider lower envelope of distance functions F
2. Design algorithm $\mathcal{A}$ to compute a k-shallow cutting $\Lambda_k(F)$

$\Lambda_k(F)$ = collection of few disjoint **prisms**

each prism has $O(1)$ complexity

intersects $O(k)$ functions



How?

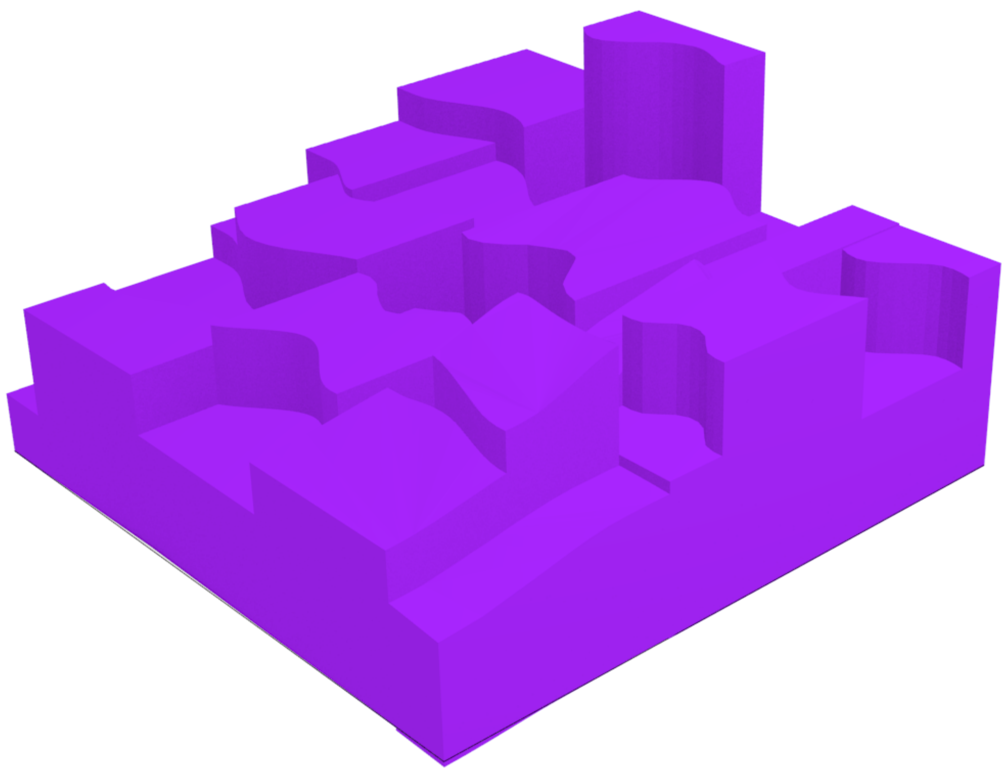
Strategy in the Kaplan et al. approach:

1. Consider lower envelope of distance functions F
2. Design algorithm $\mathcal{A}$ to compute a k-shallow cutting $\Lambda_k(F)$

$\Lambda_k(F)$ = collection of few disjoint **prisms**

each prism has $O(1)$ complexity

intersects $O(k)$ functions



How?

Strategy in the Kaplan et al. approach:

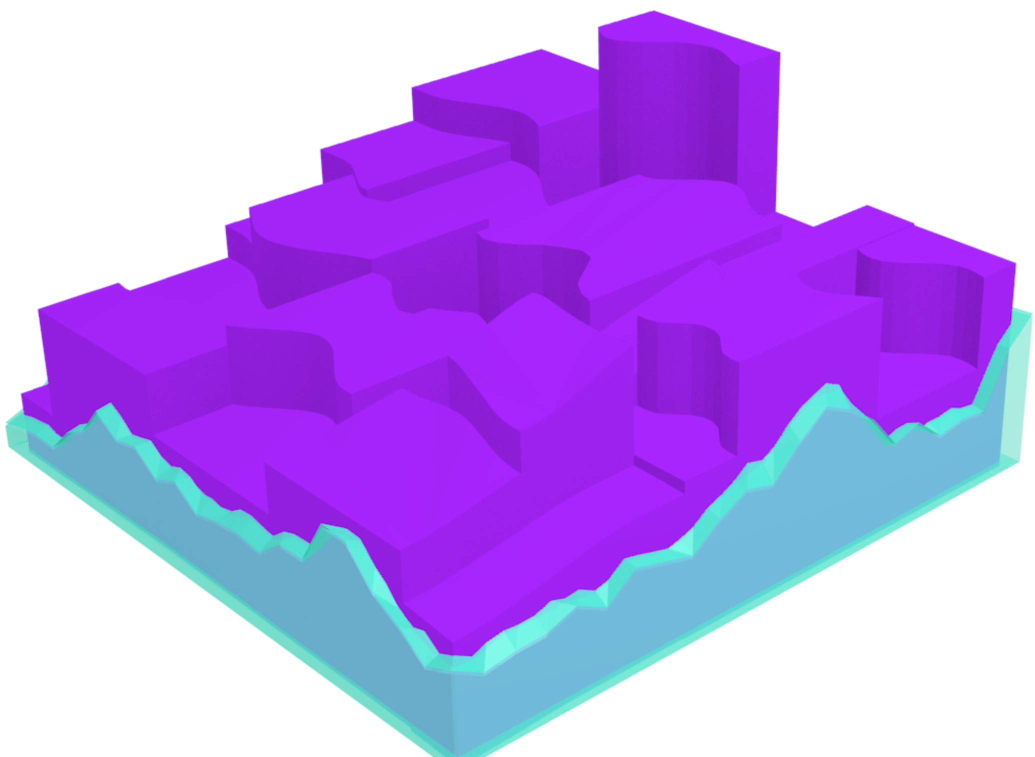
1. Consider lower envelope of distance functions F
2. Design algorithm $\mathcal{A}$ to compute a k-shallow cutting $\Lambda_k(F)$

$\Lambda_k(F)$ = collection of few disjoint **prisms**

each prism has $O(1)$ complexity

intersects $O(k)$ functions

that together cover the $\leq k$-level $L_{\leq k}(F)$



How?

Strategy in the Kaplan et al. approach:

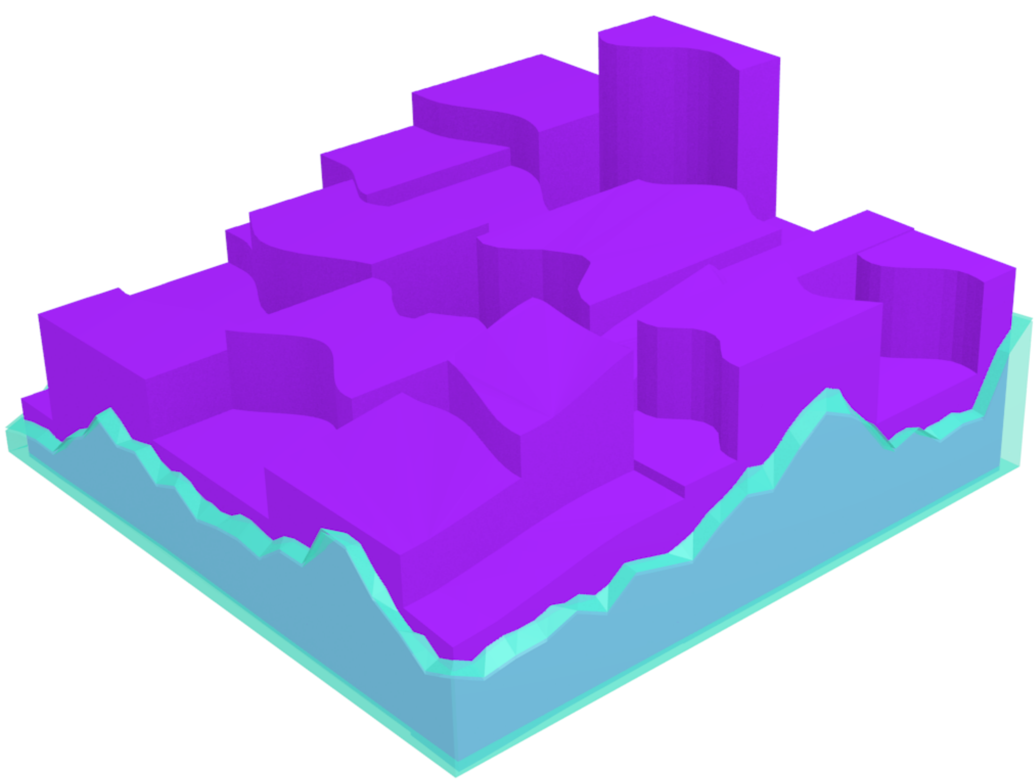
1. Consider lower envelope of distance functions F
2. Design algorithm $\mathcal{A}$ to compute a k-shallow cutting $\Lambda_k(F)$

$\Lambda_k(F)$ = collection of few ($O(n/k)\log^c n$) disjoint **prisms**

each prism has $O(1)$ complexity

intersects $O(k)$ functions

that together cover the $\leq k$-level $L_{\leq k}(F)$



How?

Strategy in the Kaplan et al. approach:

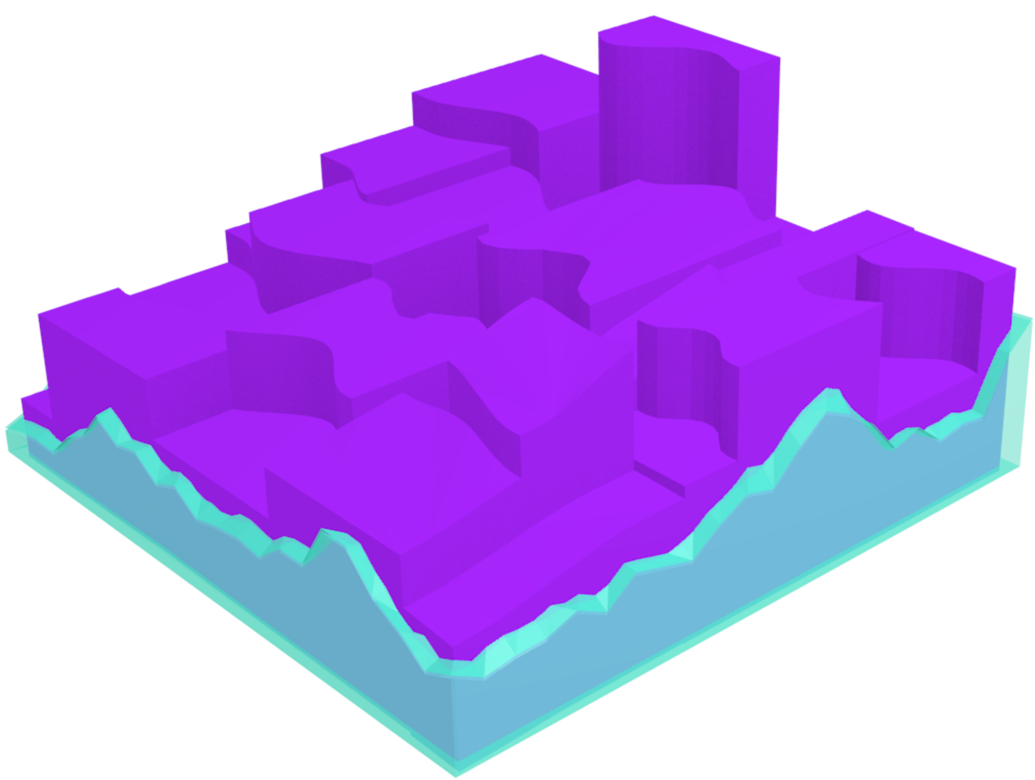
1. Consider lower envelope of distance functions F
2. Design algorithm $\mathcal{A}$ to compute a k-shallow cutting $\Lambda_k(F)$

   $\Lambda_k(F)$ = collection of few ($O(n/k)\log^c n$) disjoint **prisms**

   each prism has $O(1)$ complexity

   intersects $O(k)$ functions

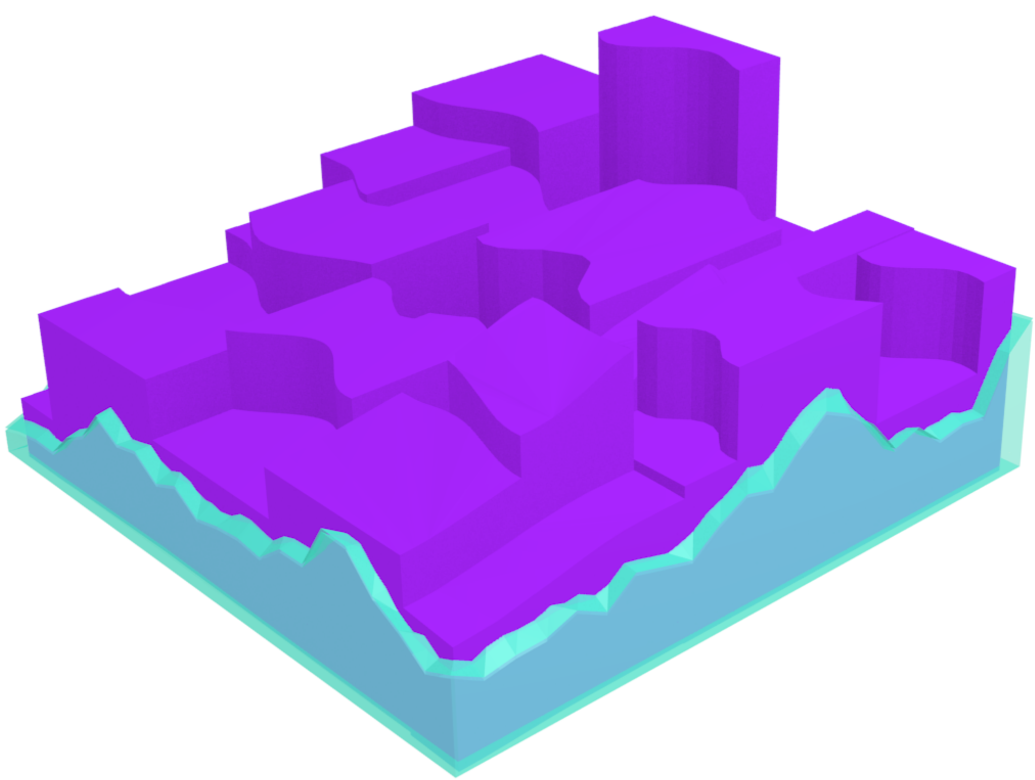
   that together cover the $\leq k$-level $L_{\leq k}(F)$
3. Use $\mathcal{A}$ to construct a data structure.



How?

1. Consider lower envelope of distance functions F
2. Design algorithm $\mathcal{A}$ to compute a k-shallow cutting $\Lambda_k(F)$

   $\Lambda_k(F)$ = collection of few ($O(n/k)\log^c n$) disjoint **prisms**
   each prism has $O(1)$ complexity
   intersects $O(k)$ functions
   that together cover the $\leq k$-level $L_{\leq k}(F)$
3. Use $\mathcal{A}$ to construct a data structure.



How?

1. Consider lower envelope of distance functions F
2. Design algorithm $\mathcal{A}$ to compute a k-shallow cutting $\Lambda_k(F)$

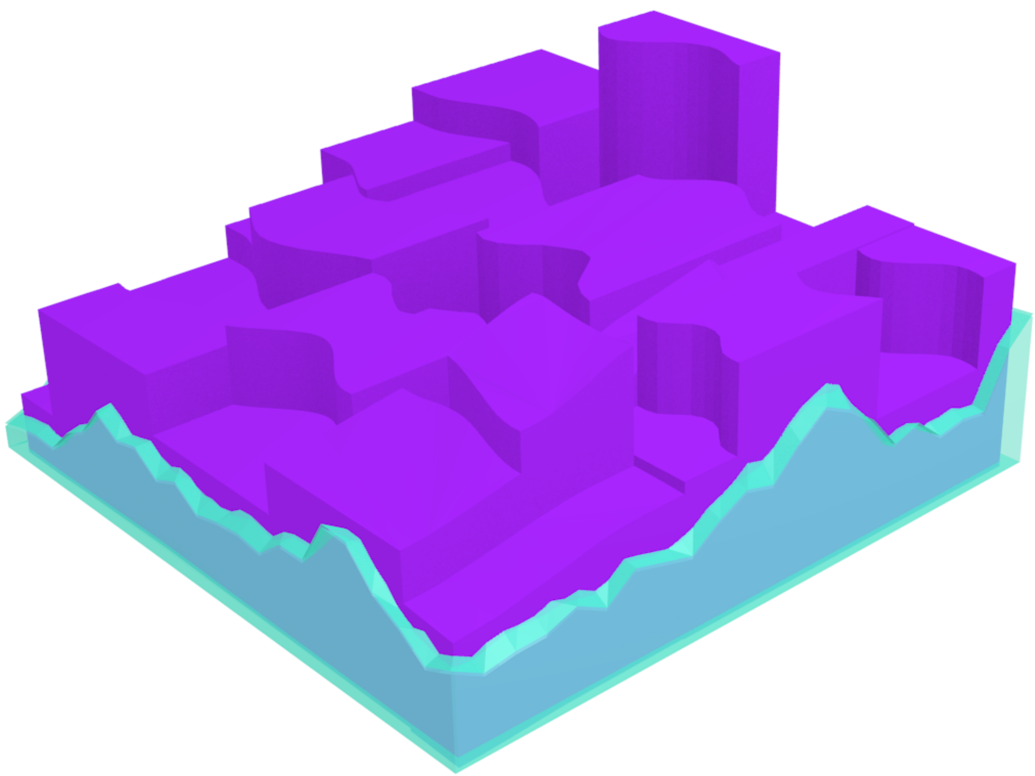
   $\Lambda_k(F)$ = collection of few ($O(n/k)\log^c n$) disjoint **prisms**

   each prism ~~has $O(1)$ complexity~~

   intersects $O(k)$ functions

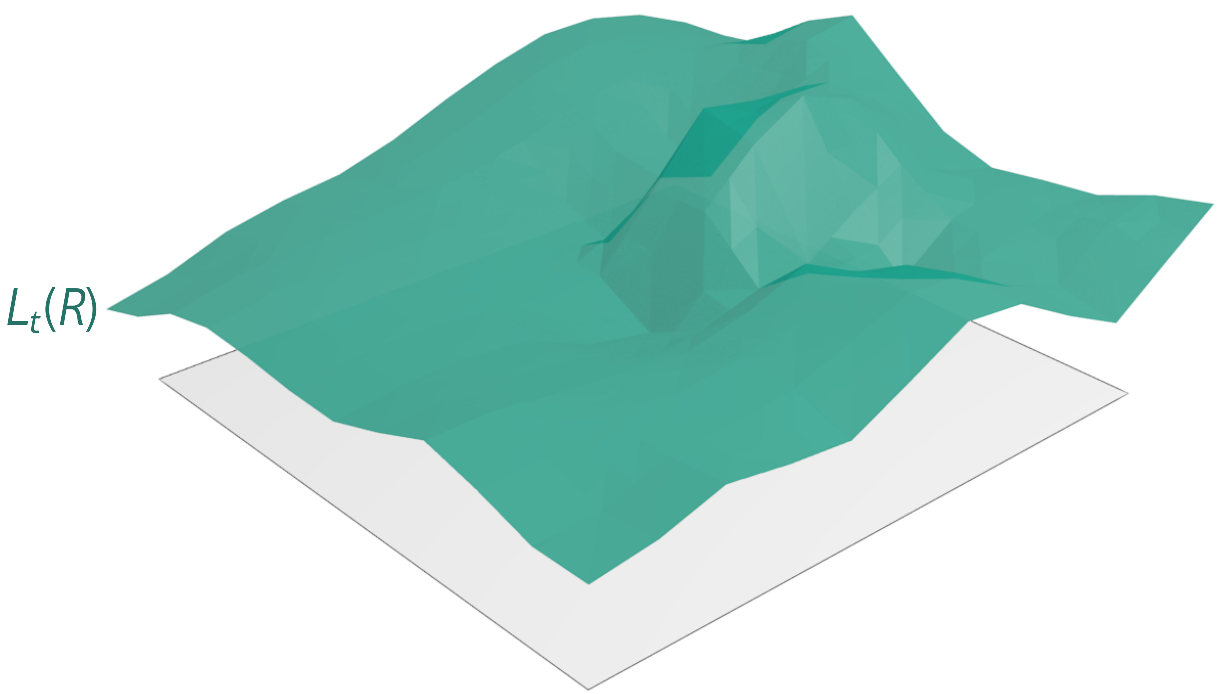
   that together cover the $\leq k$-level $L_{\leq k}(F)$

3. Use $\mathcal{A}$ to construct a data structure.



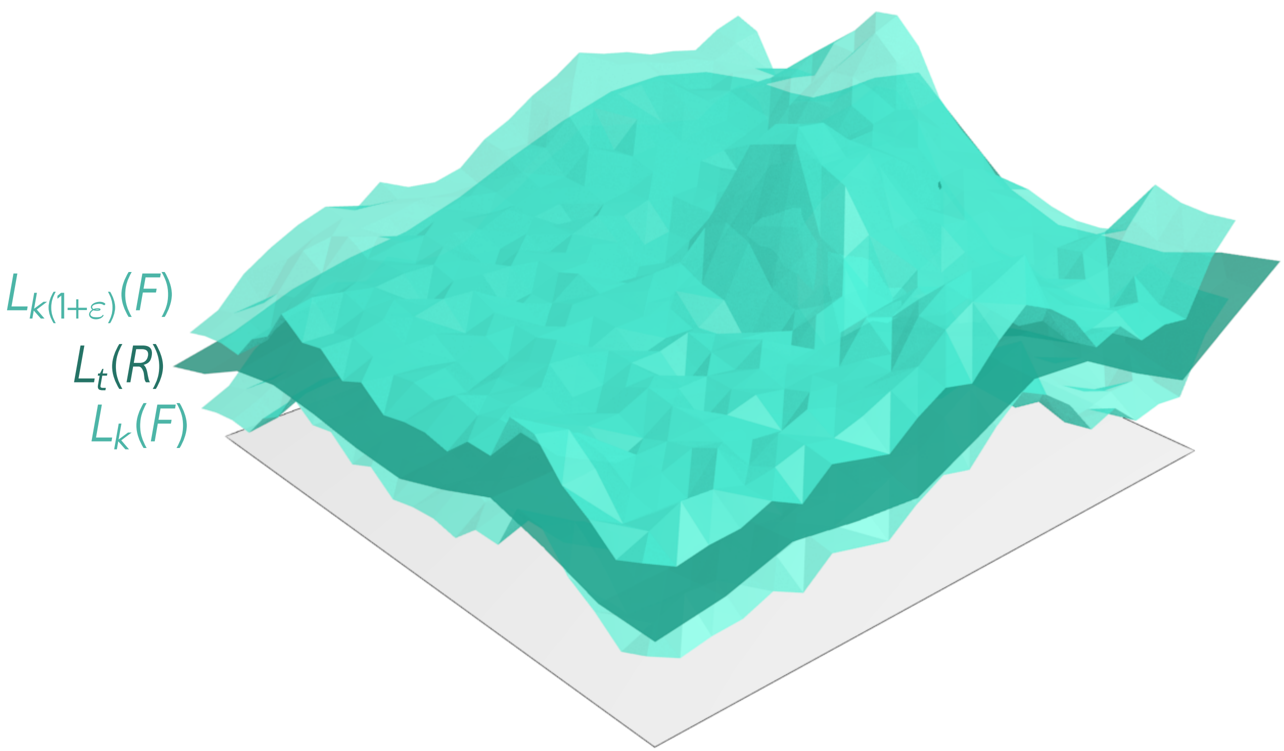
Constructing a Shallow Cutting

1. Take a random sample $R \subseteq F$
2. Construct the t-level $L_t(R)$



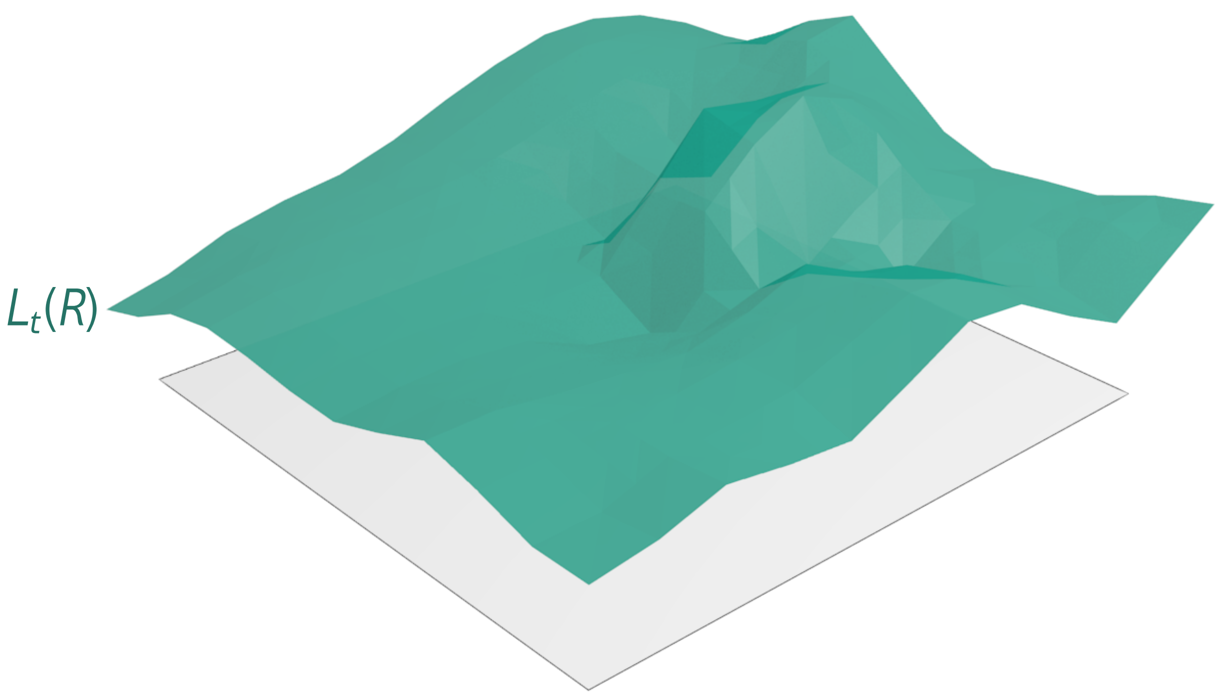
Constructing a Shallow Cutting

1. Take a random sample $R \subseteq F$
2. Construct the t-level $L_t(R)$
3. Prove that $L_t(R)$ lies between $L_k(F)$ and $L_{k(1+\varepsilon)}(F)$ and that it has low complexity



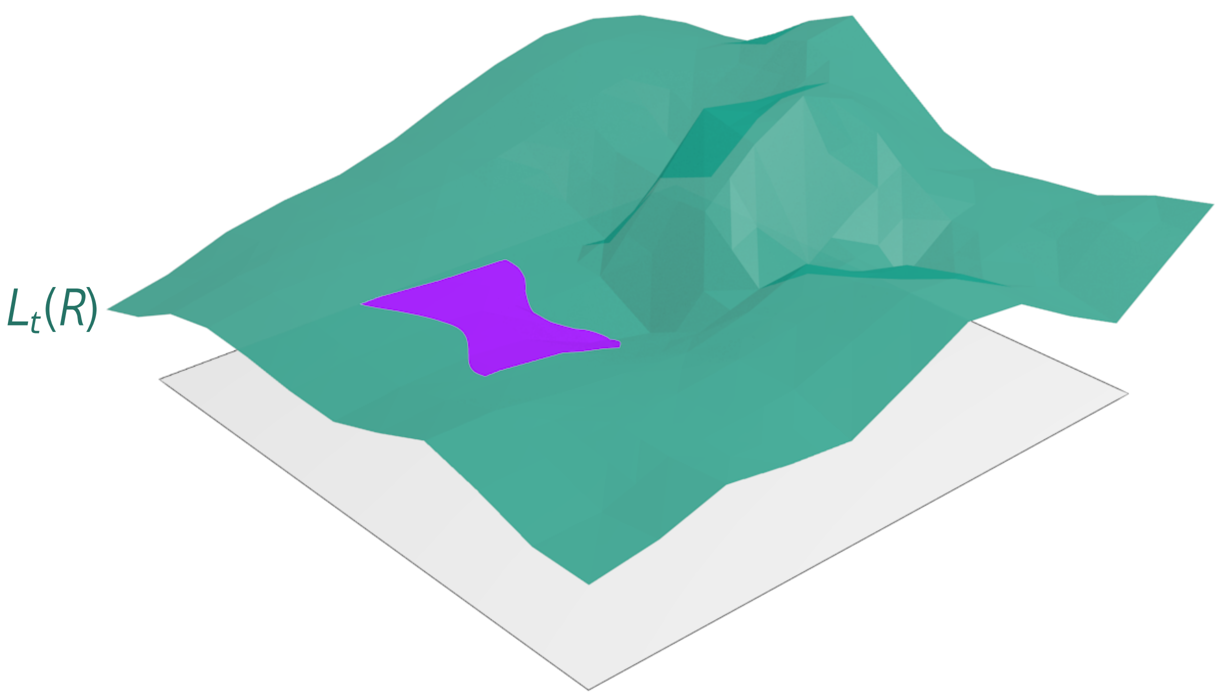
Constructing a Shallow Cutting

1. Take a random sample $R \subseteq F$
2. Construct the t-level $L_t(R)$
3. Prove that $L_t(R)$ lies between $L_k(F)$ and $L_{k(1+\varepsilon)}(F)$
   and that it has low complexity
4. Turn $L_t(R)$ into a k-shallow cutting $\Lambda_k(F)$



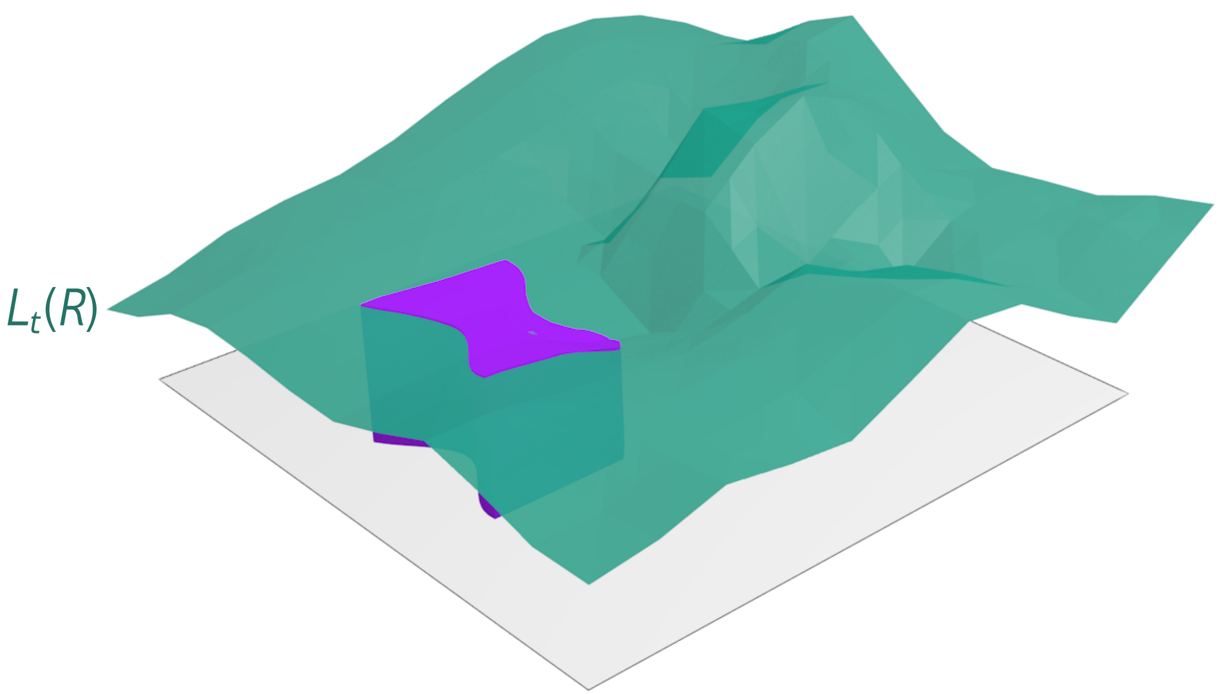
Constructing a Shallow Cutting

1. Take a random sample $R \subseteq F$
2. Construct the t-level $L_t(R)$
3. Prove that $L_t(R)$ lies between $L_k(F)$ and $L_{k(1+\varepsilon)}(F)$ and that it has low complexity
4. Turn $L_t(R)$ into a k-shallow cutting $\Lambda_k(F)$



Constructing a Shallow Cutting

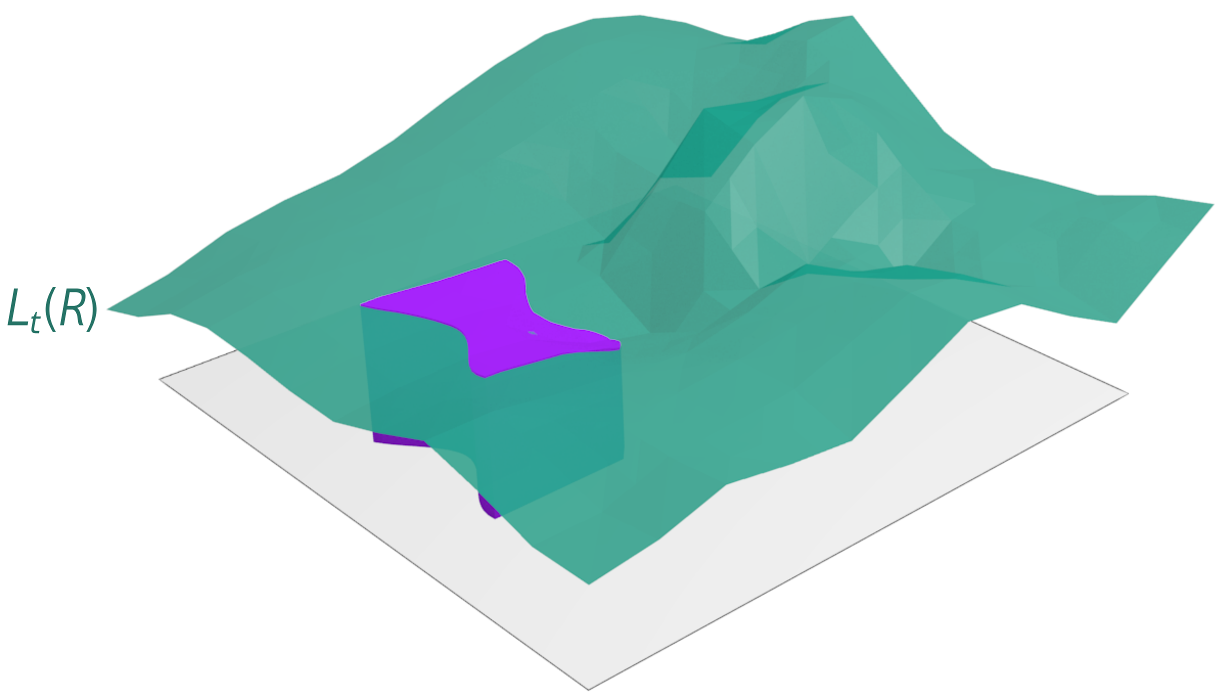
1. Take a random sample $R \subseteq F$
2. Construct the t-level $L_t(R)$
3. Prove that $L_t(R)$ lies between $L_k(F)$ and $L_{k(1+\varepsilon)}(F)$
   and that it has low complexity
4. Turn $L_t(R)$ into a k-shallow cutting $\Lambda_k(F)$



Constructing a Shallow Cutting

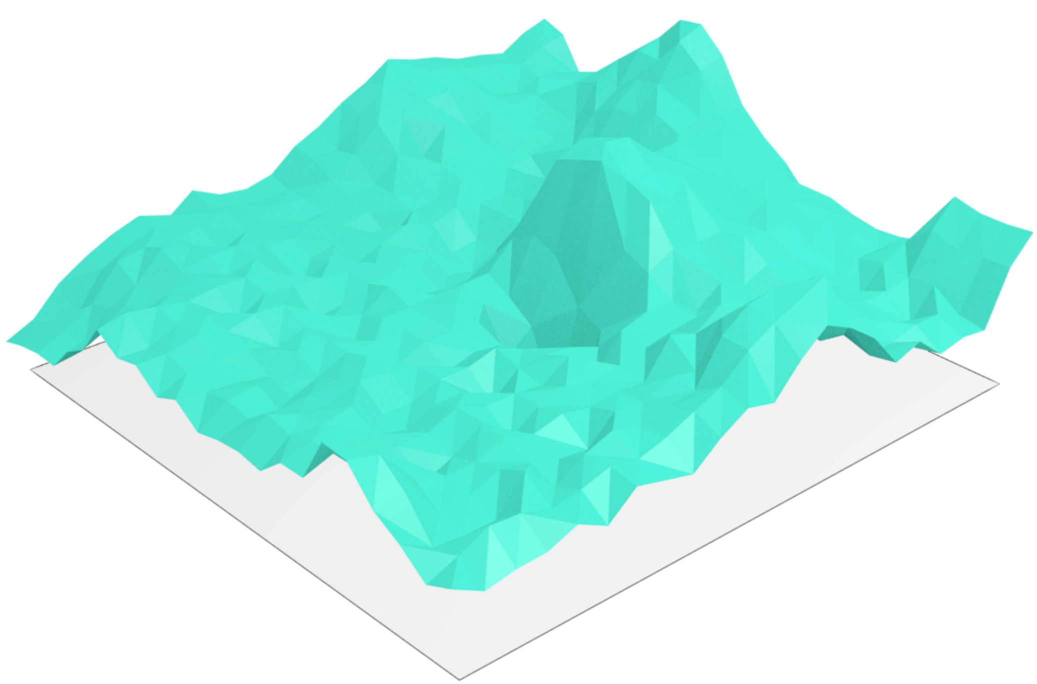
1. Take a random sample $R \subseteq F$
2. Construct the t-level $L_t(R)$
3. Prove that $L_t(R)$ lies between $L_k(F)$ and $L_{k(1+\varepsilon)}(F)$

   and that it has low complexity
4. Turn $L_t(R)$ into a k-shallow cutting $\Lambda_k(F)$
5. For each $\nabla \in \Lambda_k(F)$, compute **conflict list** F_∇



Constructing a Shallow Cutting

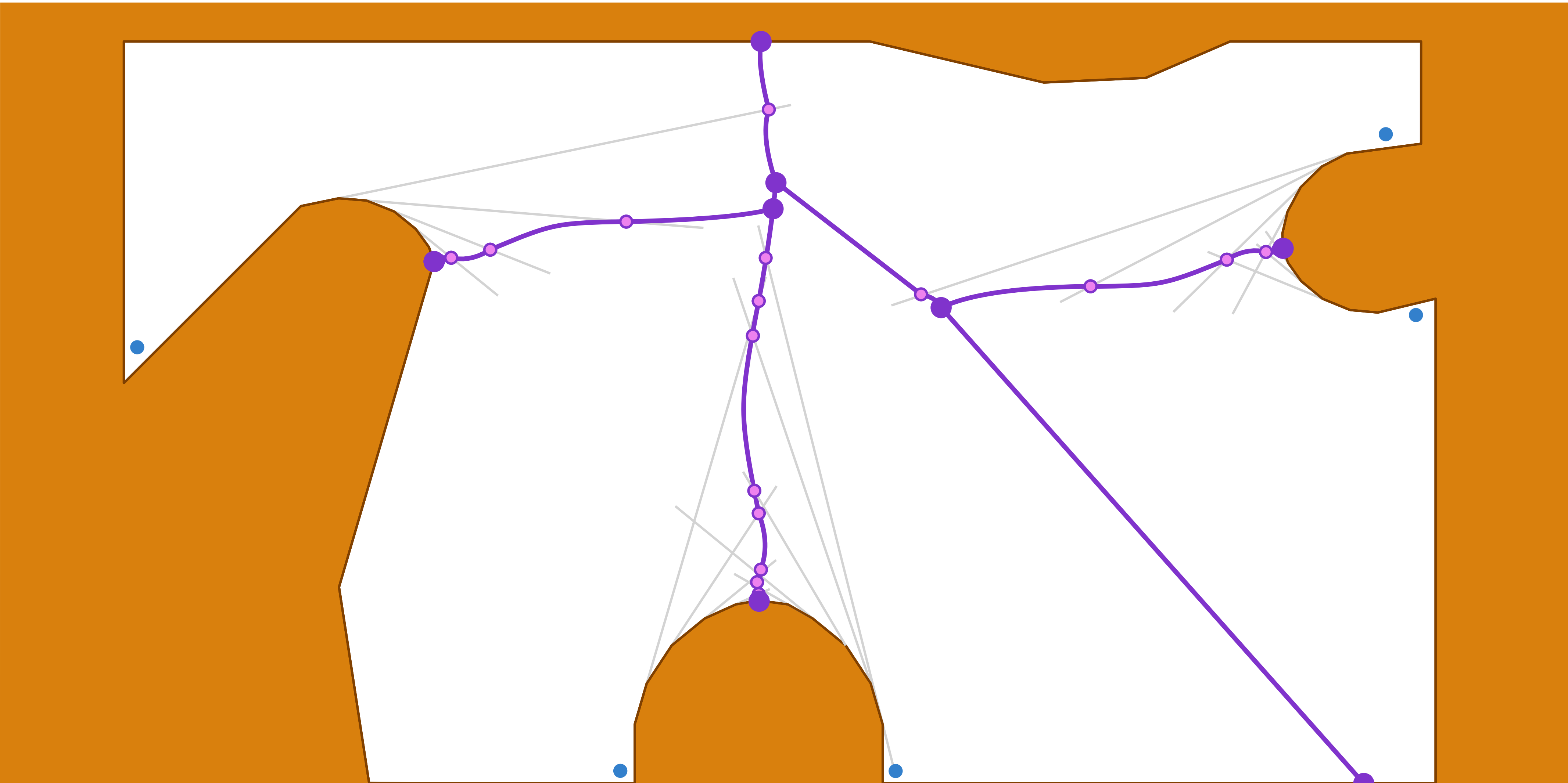
Obs. $L_k(F) \approx k^{\text{th}}$ order Voronoi Diagram $\mathcal{V}_k(S)$



Representing a Level

Thm. Geodesic $\mathcal{V}_k(S)$ has $O(kn)$ vertices of degree 1 or 3
$O(km)$ vertices of degree 2

[Liu et al., SODA '13]

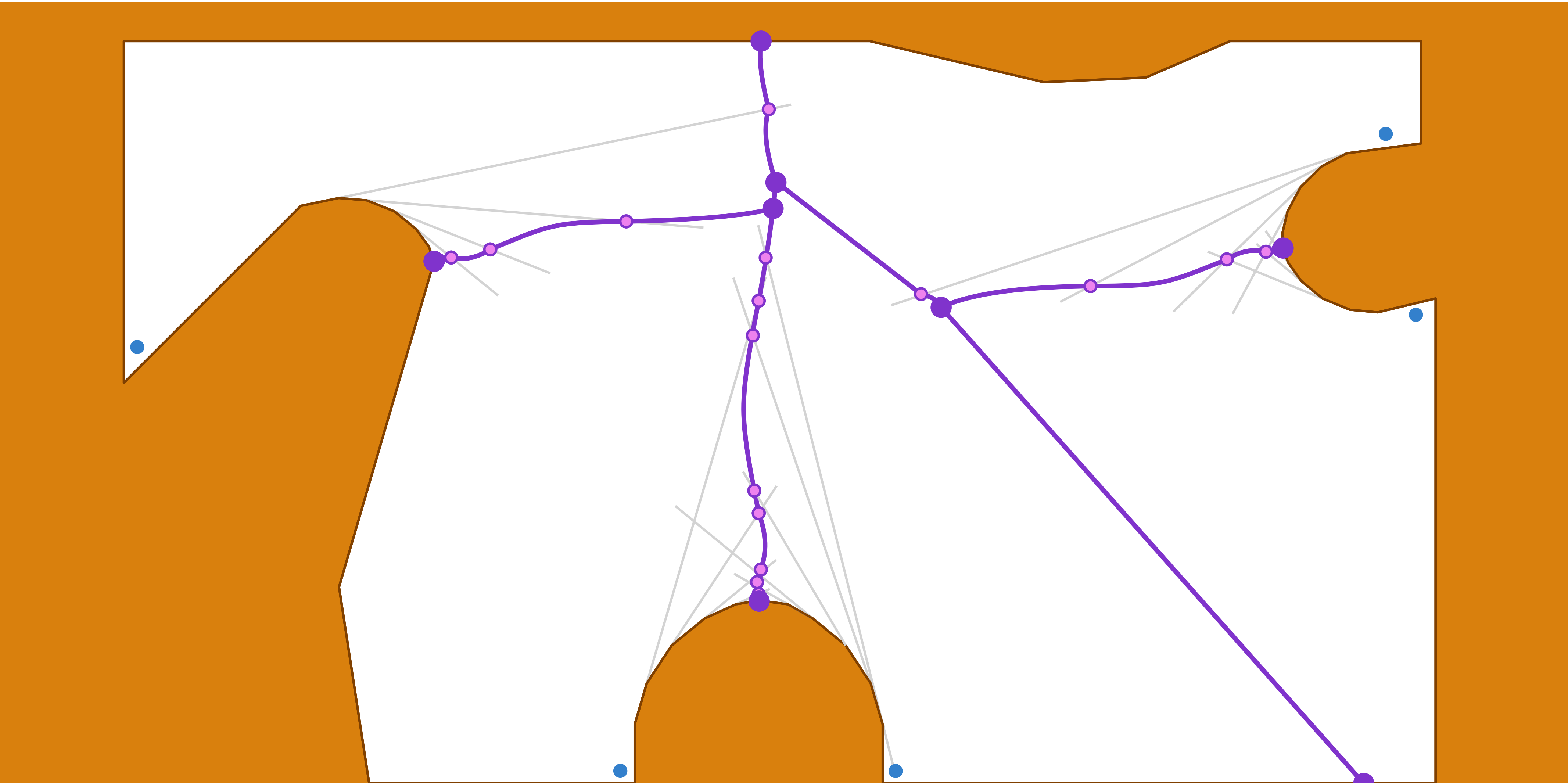


Representing a Level

Thm. Geodesic $\mathcal{V}_k(S)$ has $O(kn)$ vertices of degree 1 or 3
$O(km)$ vertices of degree 2

[Liu et al., SODA '13]

Main Idea: **Store locations of only degree 1 or 3 vertices + topology $\mathcal{V}_k(S)$**

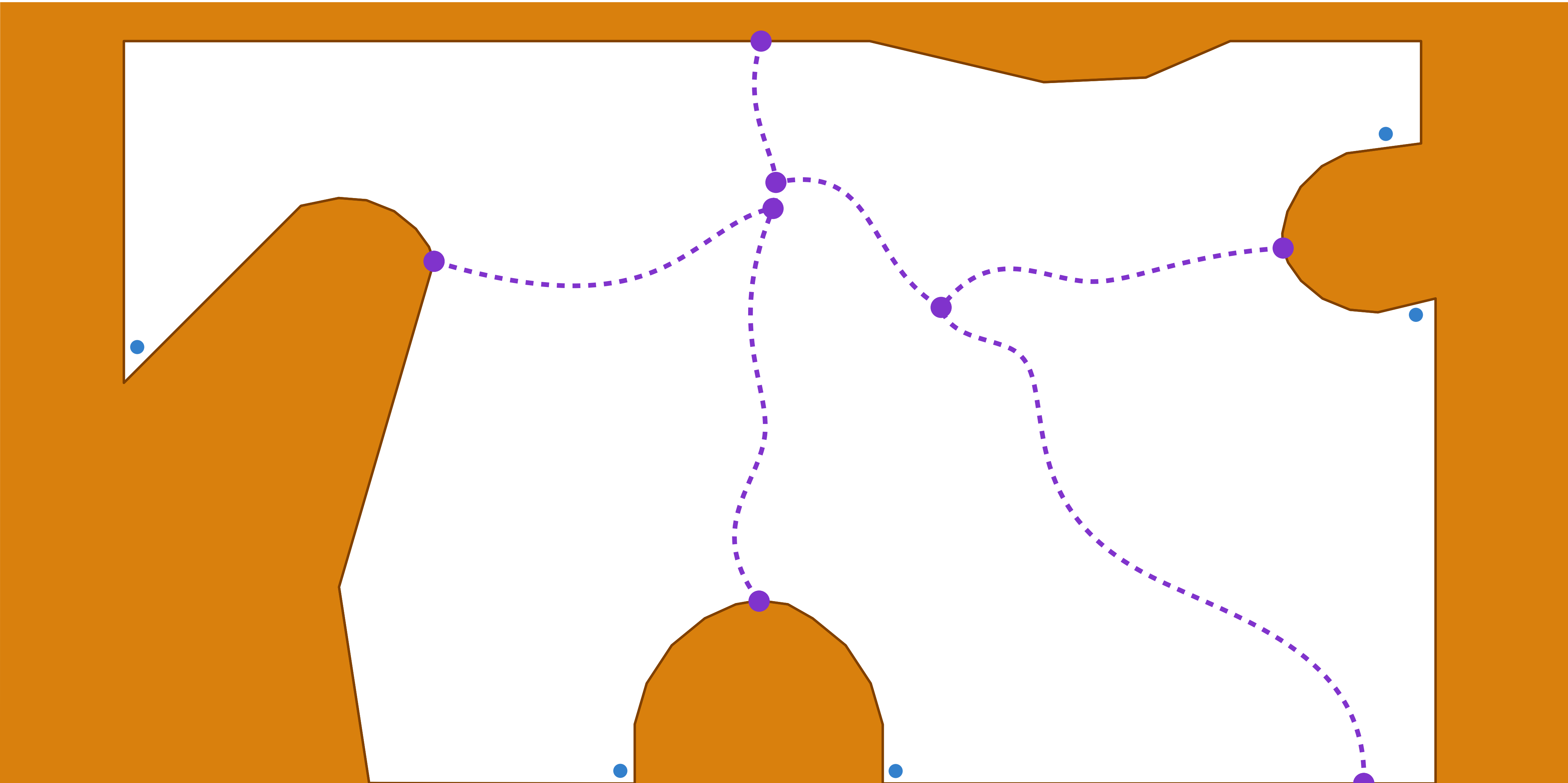


Representing a Level

Thm. Geodesic $\mathcal{V}_k(S)$ has $O(kn)$ vertices of degree 1 or 3
$O(km)$ vertices of degree 2

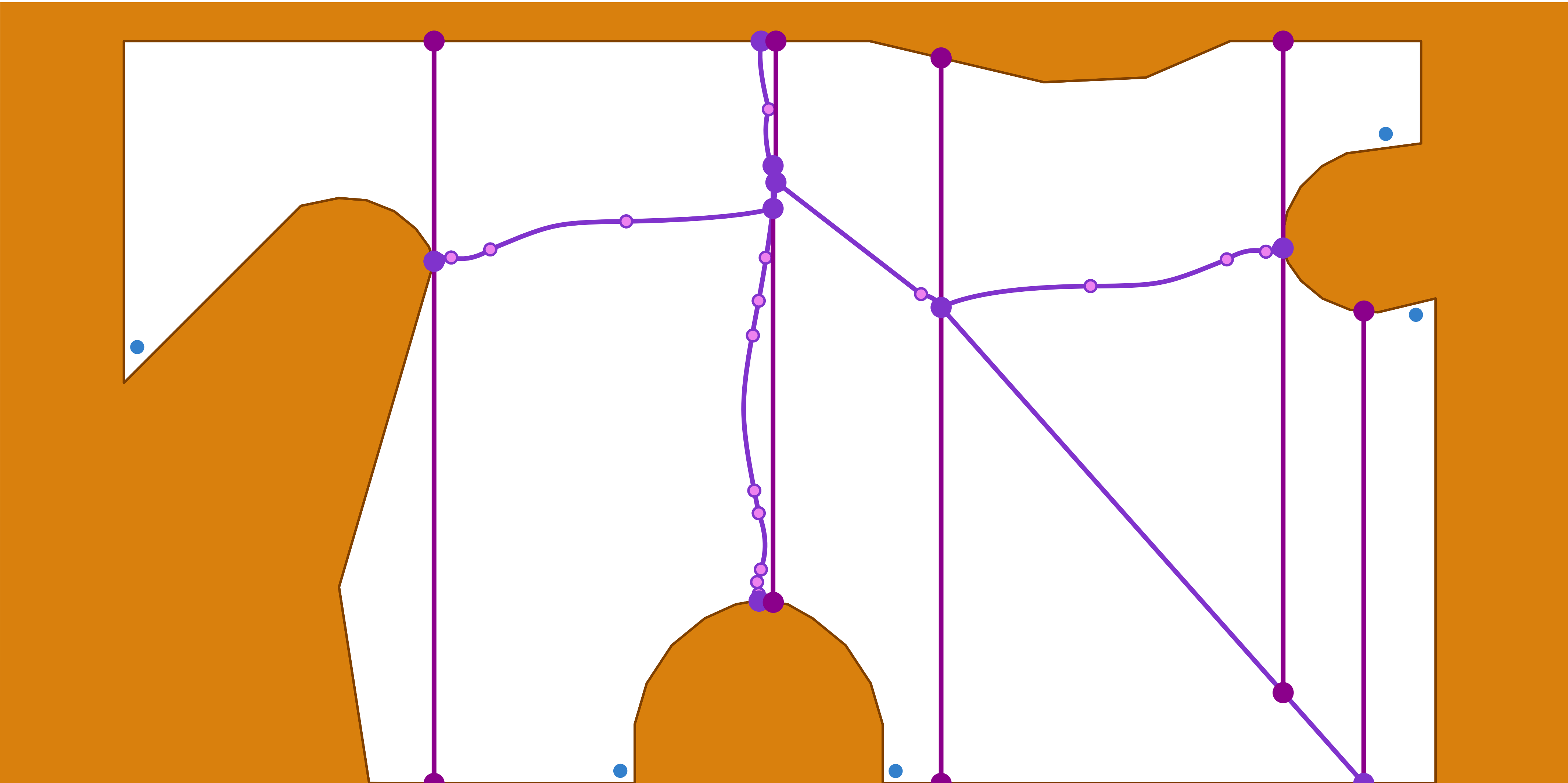
[Liu et al., SODA '13]

Main Idea: **Store locations of only degree 1 or 3 vertices + topology $\mathcal{V}_k(S)$**



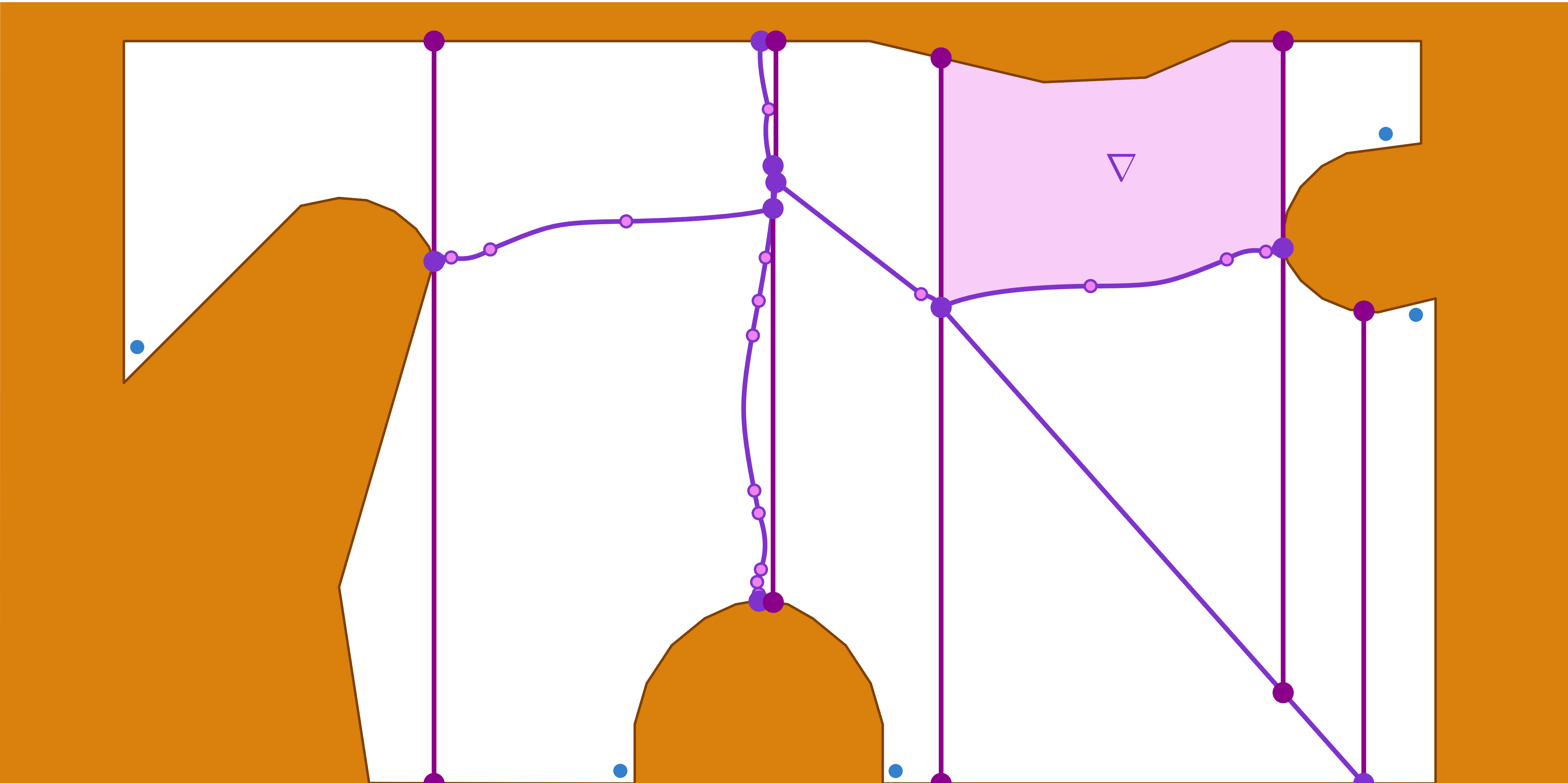
Representing a Level

Thm. $\mathcal{V}_k(S)$ can be represented using $O(kn)$ space, and s.t. all regions are **pseudo trapezoids**



Representing a Level

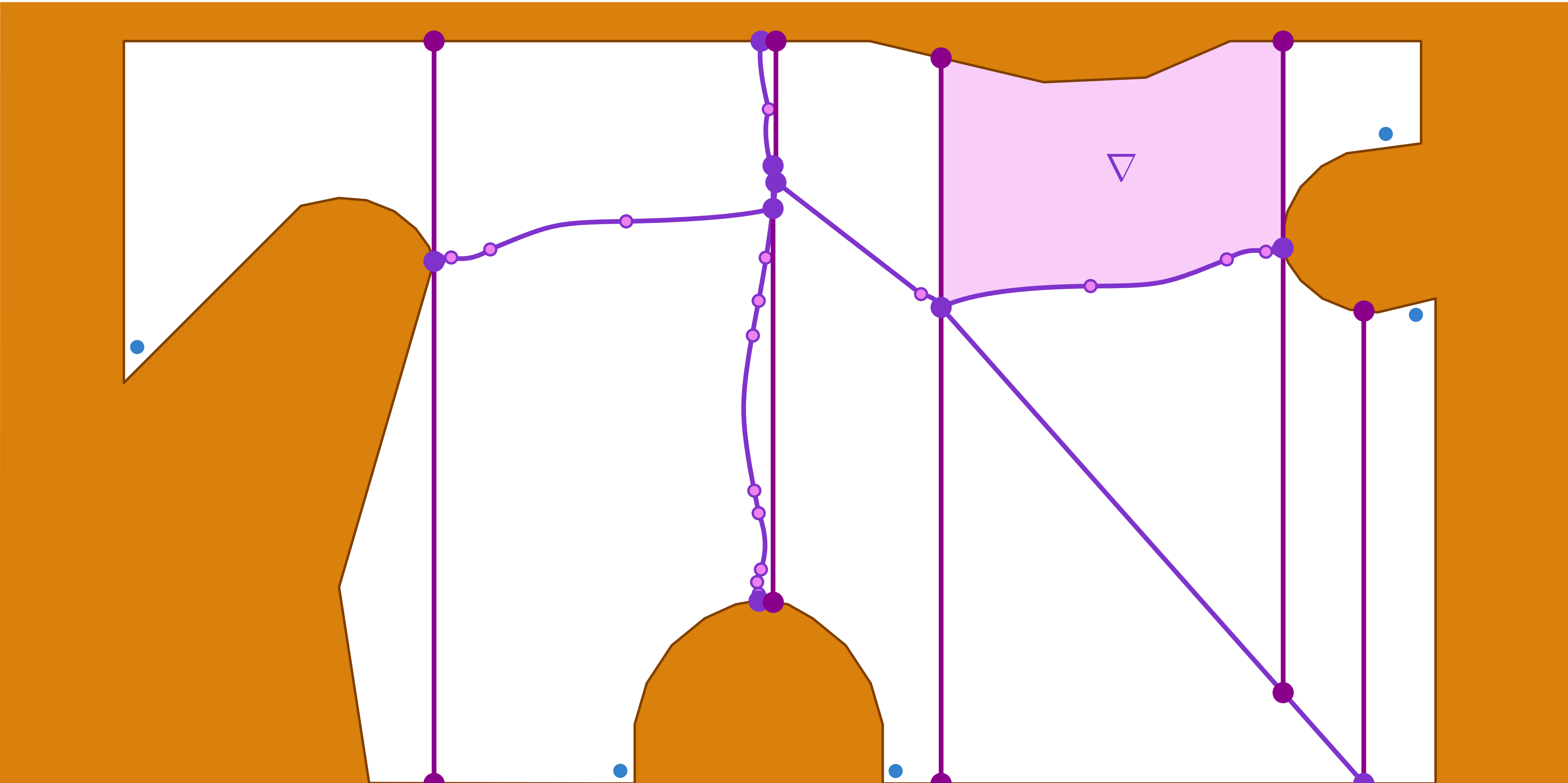
Thm. $\mathcal{V}_k(S)$ can be represented using $O(kn)$ space, and s.t. all regions are **pseudo trapezoids**



Representing a Level

Thm. $\mathcal{V}_k(S)$ can be represented using $O(kn)$ space, and s.t. all regions are **pseudo trapezoids**

$\mathcal{V}_k(S)$ can be computed in $O(k^2 n \log^3(n+m))$ time

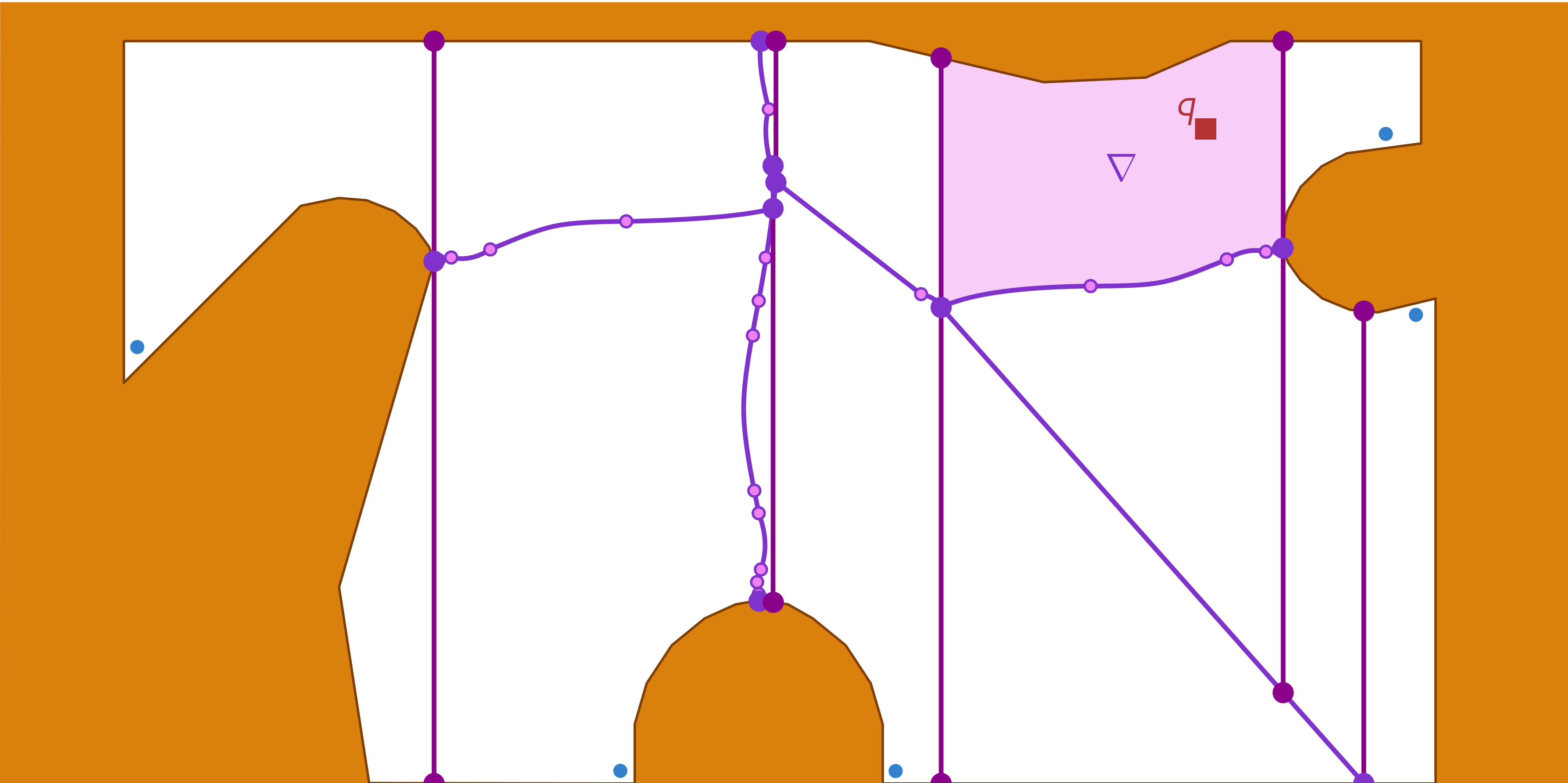


Representing a Level

Thm. $\mathcal{V}_k(S)$ can be represented using $O(kn)$ space, and s.t. all regions are **pseudo trapezoids**

$\mathcal{V}_k(S)$ can be computed in $O(k^2 n \log^3(n+m))$ time

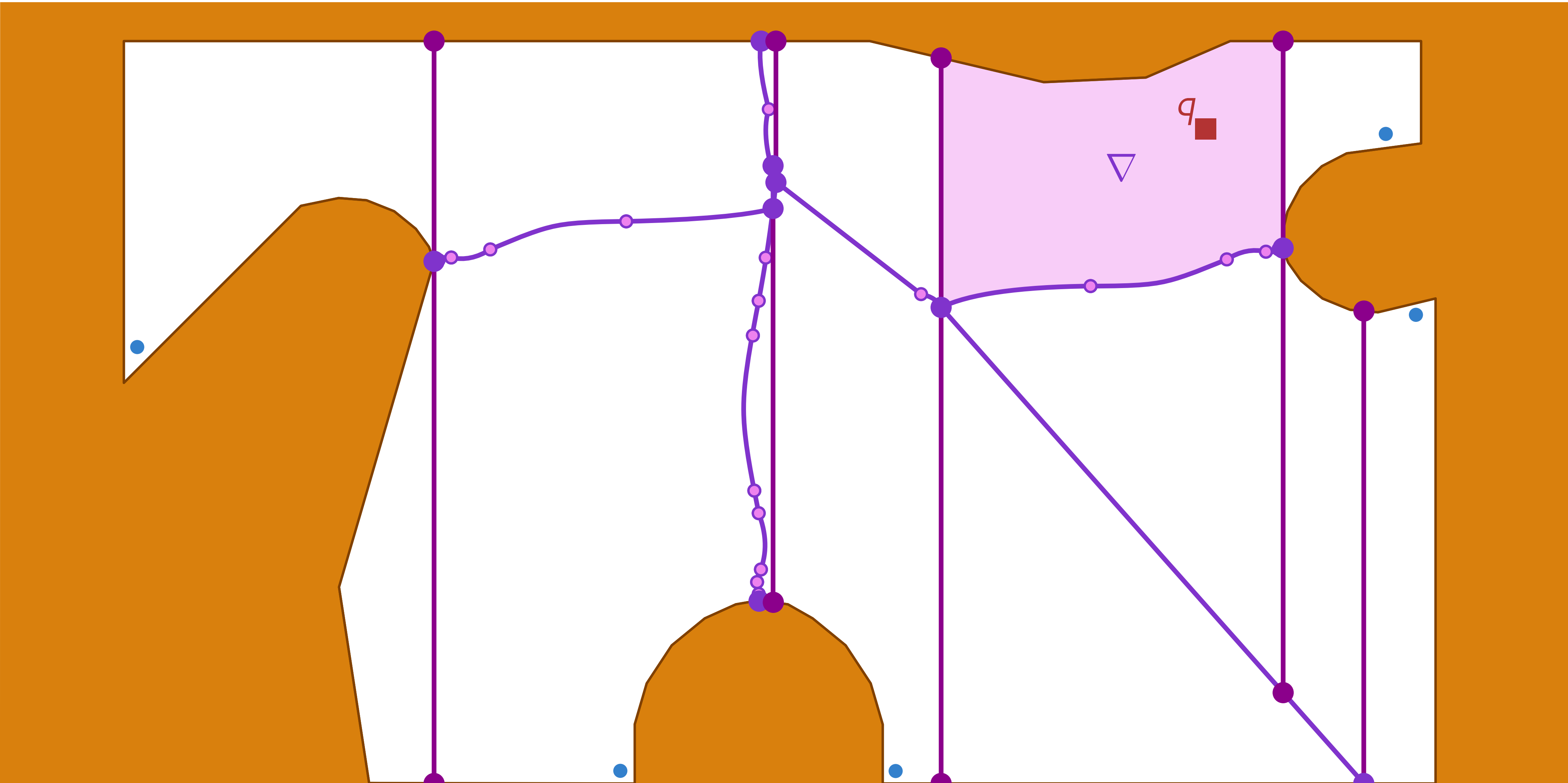
finding the region $\nabla \ni q$ takes $O(\log(n+m))$ time.



Representing a Level

Thm. $\mathrm{L_k}(\mathbf{F})$ can be represented using $O(kn)$ space

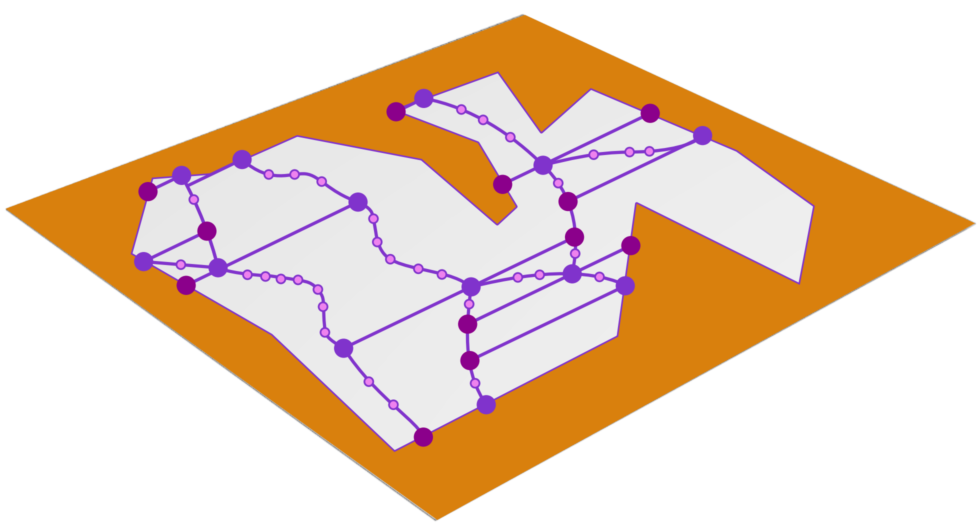
$\mathrm{L_k}(\mathbf{F})$ can be computed in $O(k^2 n \log^3(n+m))$ time
finding the region $\nabla \ni q$ takes $O(\log(n+m))$ time.



Representing a Level

Thm. $\mathbf{L_k}(\mathbf{F})$ can be represented using $O(kn)$ space

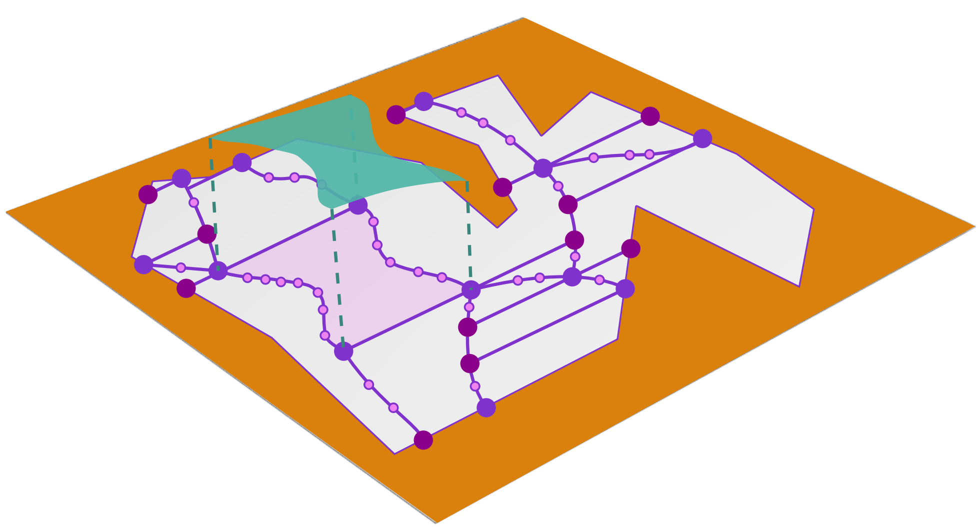
$\mathbf{L_k}(\mathbf{F})$ can be computed in $O(k^2 n \log^3(n+m))$ time
finding the region $\nabla \ni q$ takes $O(\log(n+m))$ time.



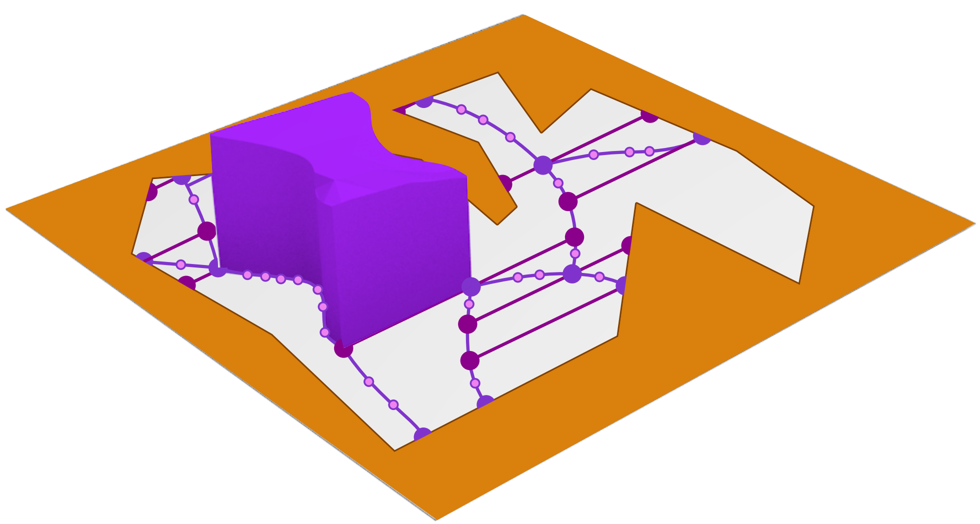
Representing a Level

Thm. $\mathbf{L_k}(\mathbf{F})$ can be represented using $O(kn)$ space

$\mathbf{L_k}(\mathbf{F})$ can be computed in $O(k^2 n \log^3(n+m))$ time
finding the region $\nabla \ni q$ takes $O(\log(n+m))$ time.

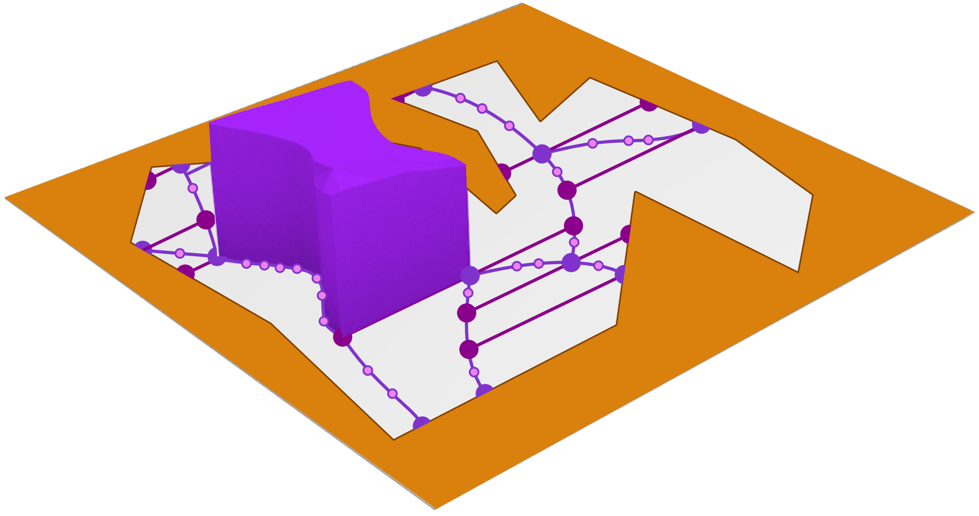


Representing a Shallow Cutting



Representing a Shallow Cutting

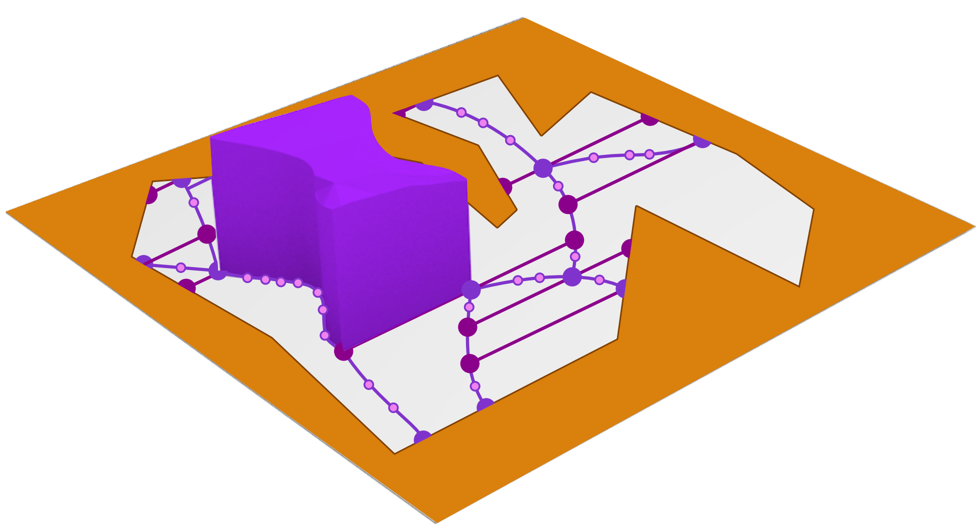
Problem 1: How to prove that a **pseudo prism** ▽ intersects $O(k)$ functions?



Representing a Shallow Cutting

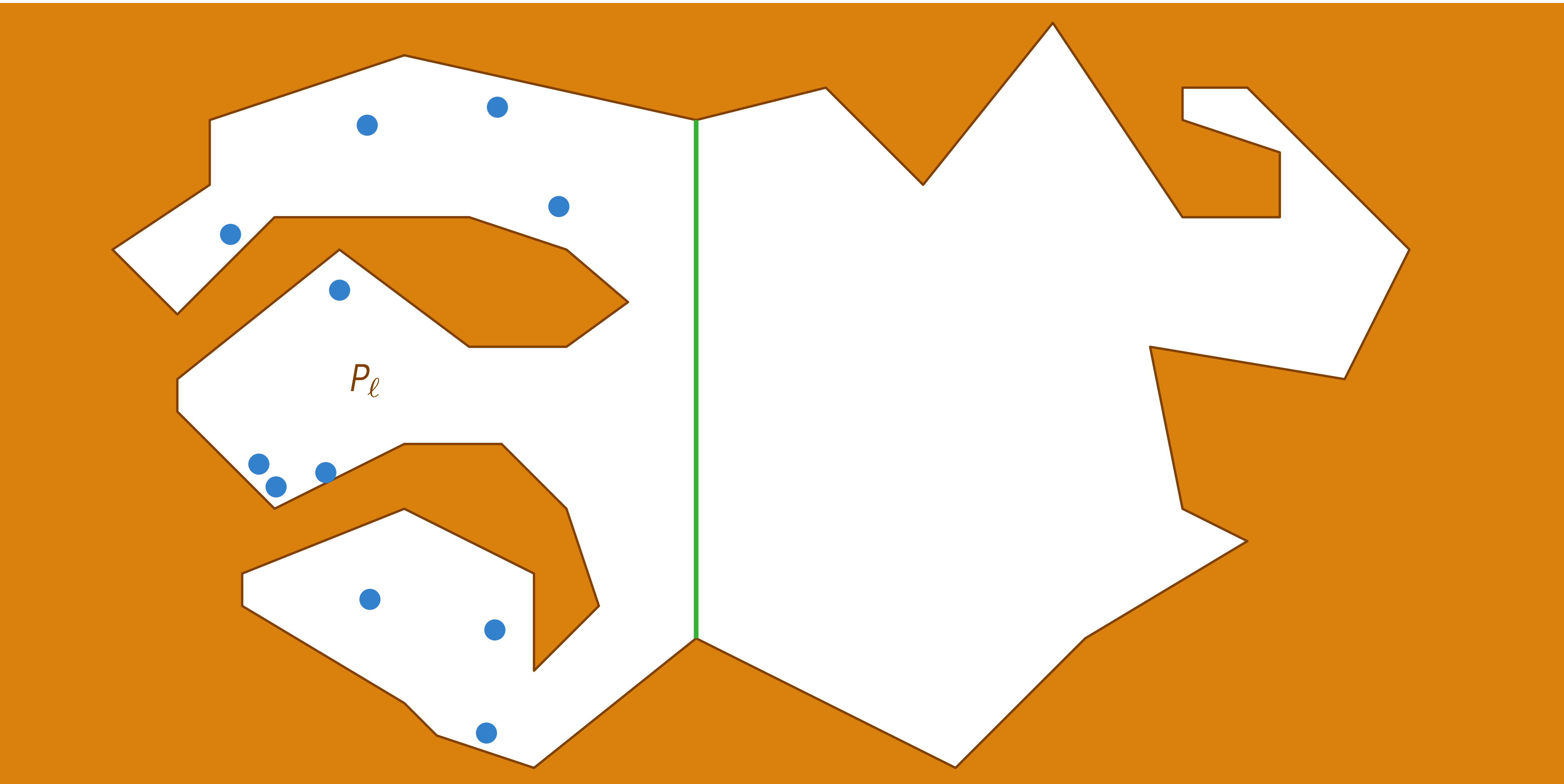
Problem 1: How to prove that a **pseudo prism** $\triangledown$ intersects $O(k)$ functions?

Problem 2: How to compute $F_\triangledown$?



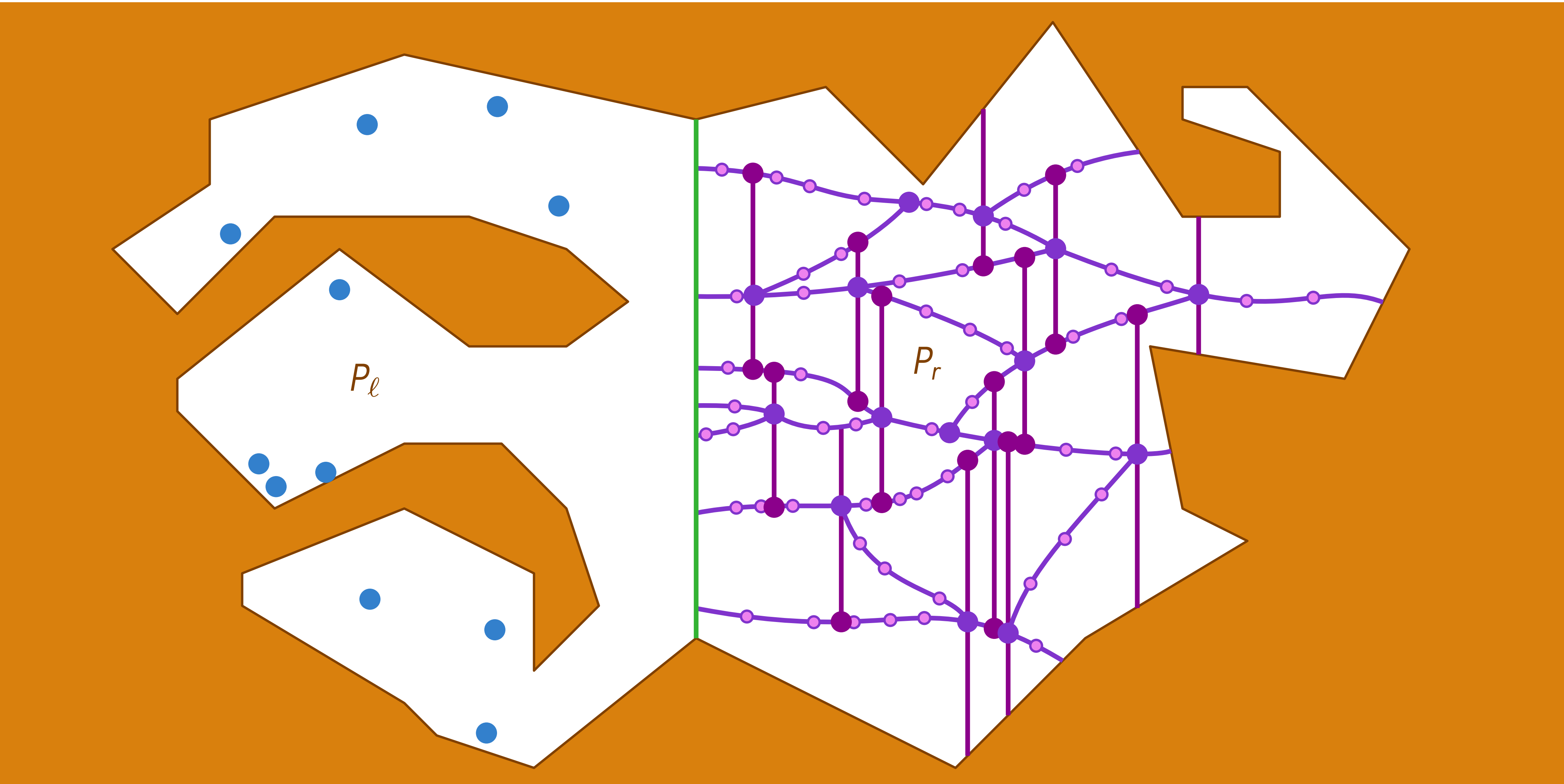
Splitting P

Main idea: Restrict S to P_ℓ and domain to P_r.



Splitting P

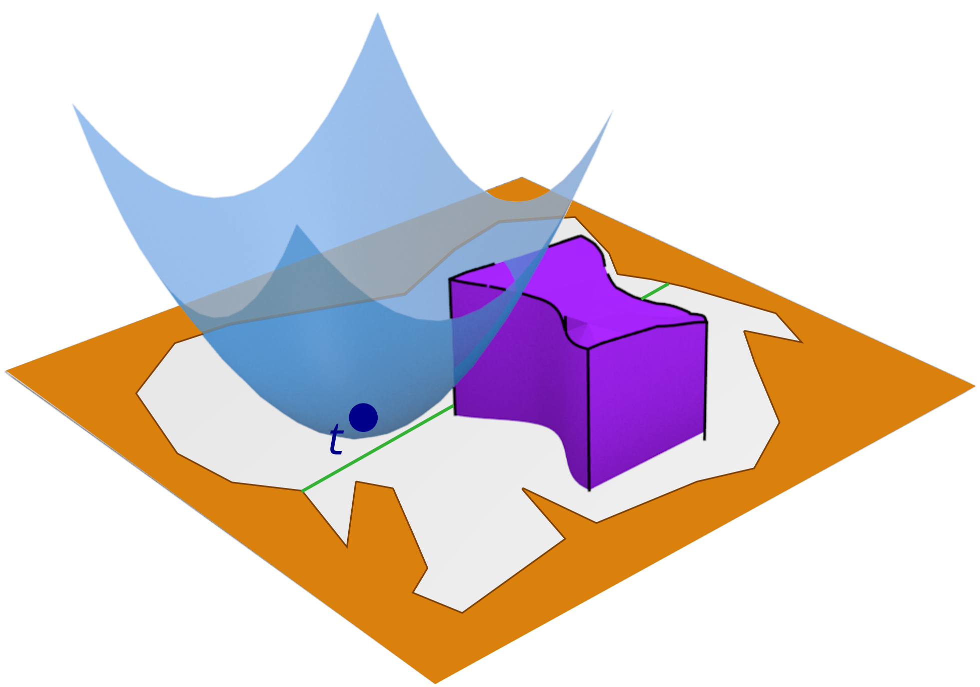
Main idea: Restrict S to P_ℓ and domain to P_r.



Splitting P

Main idea: Restrict S to P_ℓ and domain to P_r.

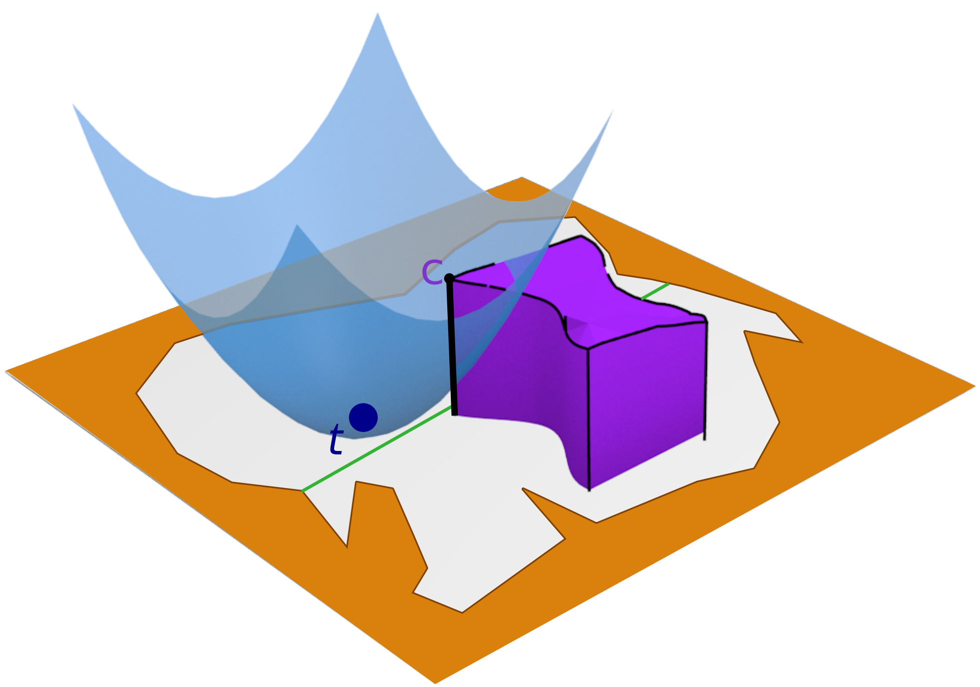
Lemma. t **conflicts** with $\nabla \iff t$ conflicts with a corner of ∇



Splitting P

Main idea: Restrict S to P_ℓ and domain to P_r.

Lemma. t **conflicts** with $\nabla \iff t$ conflicts with a corner of ∇

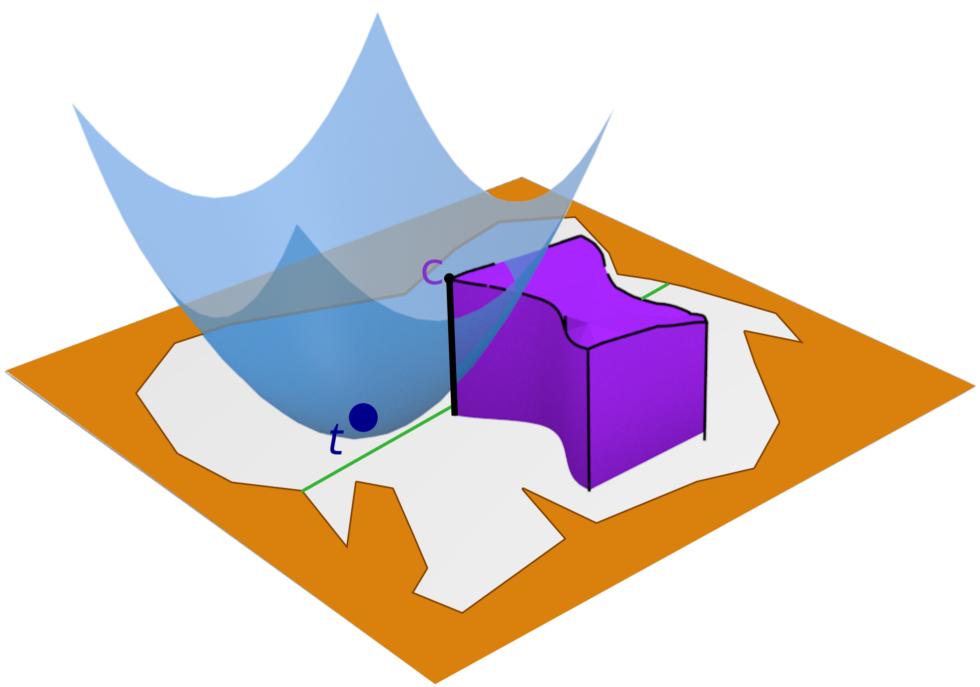


Splitting P

Main idea: Restrict S to P_ℓ and domain to P_r.

Lemma. t **conflicts** with ∇ $\iff$ t conflicts with a corner of ∇

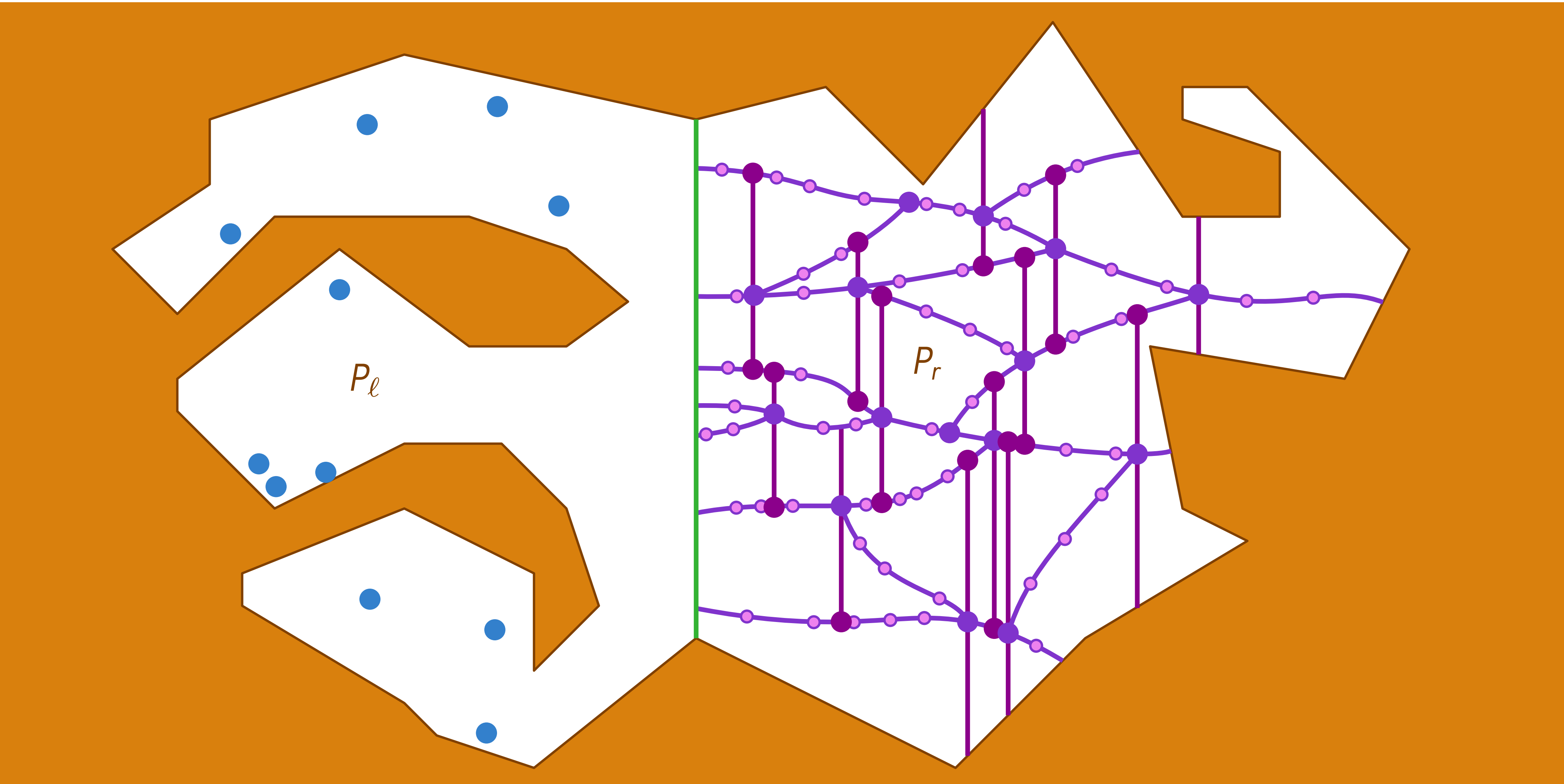
$\implies |F_\nabla| = O(k)$



Splitting P

Main idea: Restrict S to P_ℓ and domain to P_r.

Thm. $\Lambda(F)$ is a k-shallow cutting of size $O((n/k)\log^2 n)$

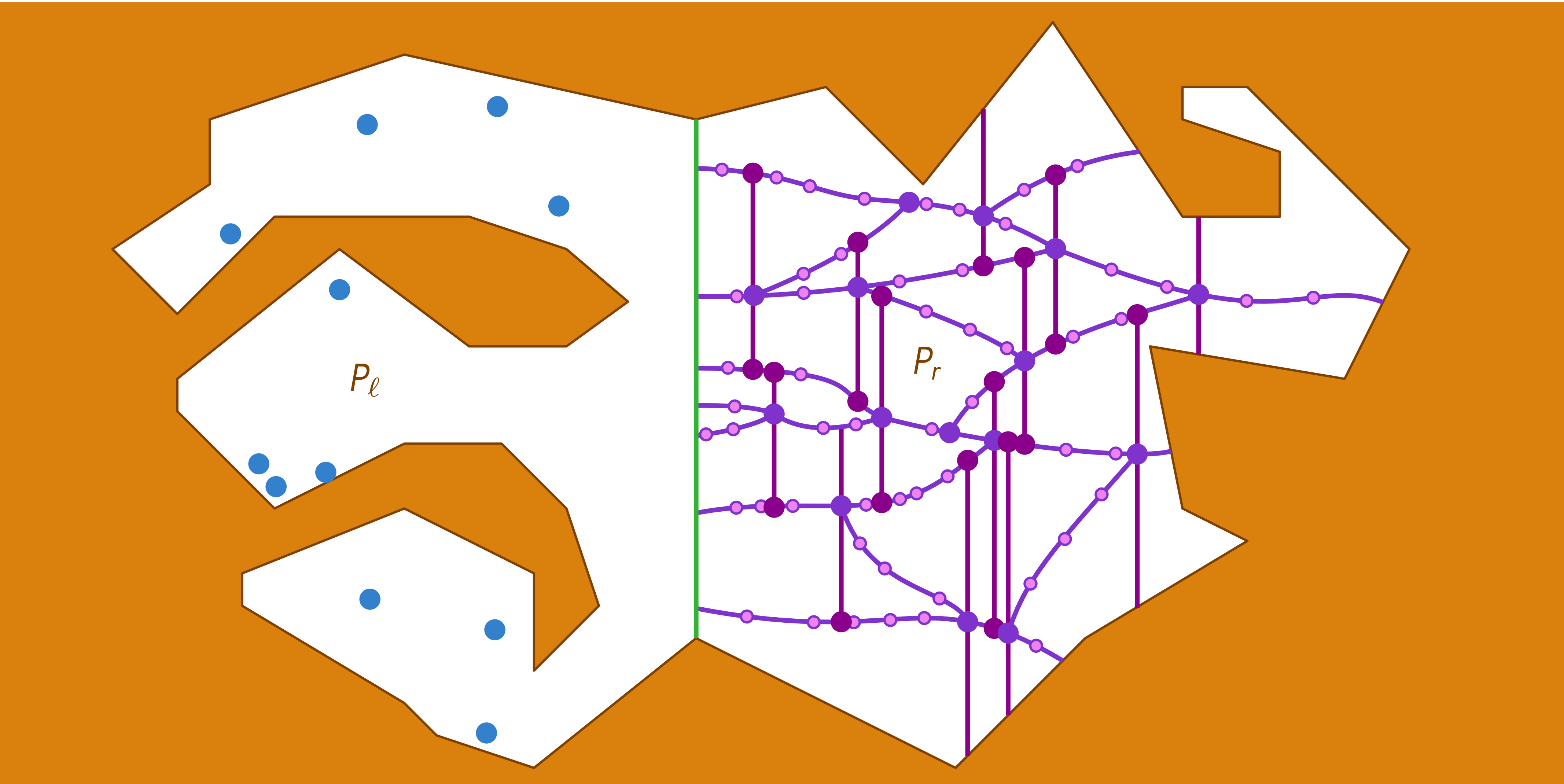


Splitting P

Main idea: Restrict S to P_ℓ and domain to P_r.

Thm. $\Lambda(F)$ is a k-shallow cutting of size $O((n/k)\log^2 n)$

$\Lambda(F)$ can be computed in $O((n/k)\log^5(n+m) + n\log^4(n+m))$ time



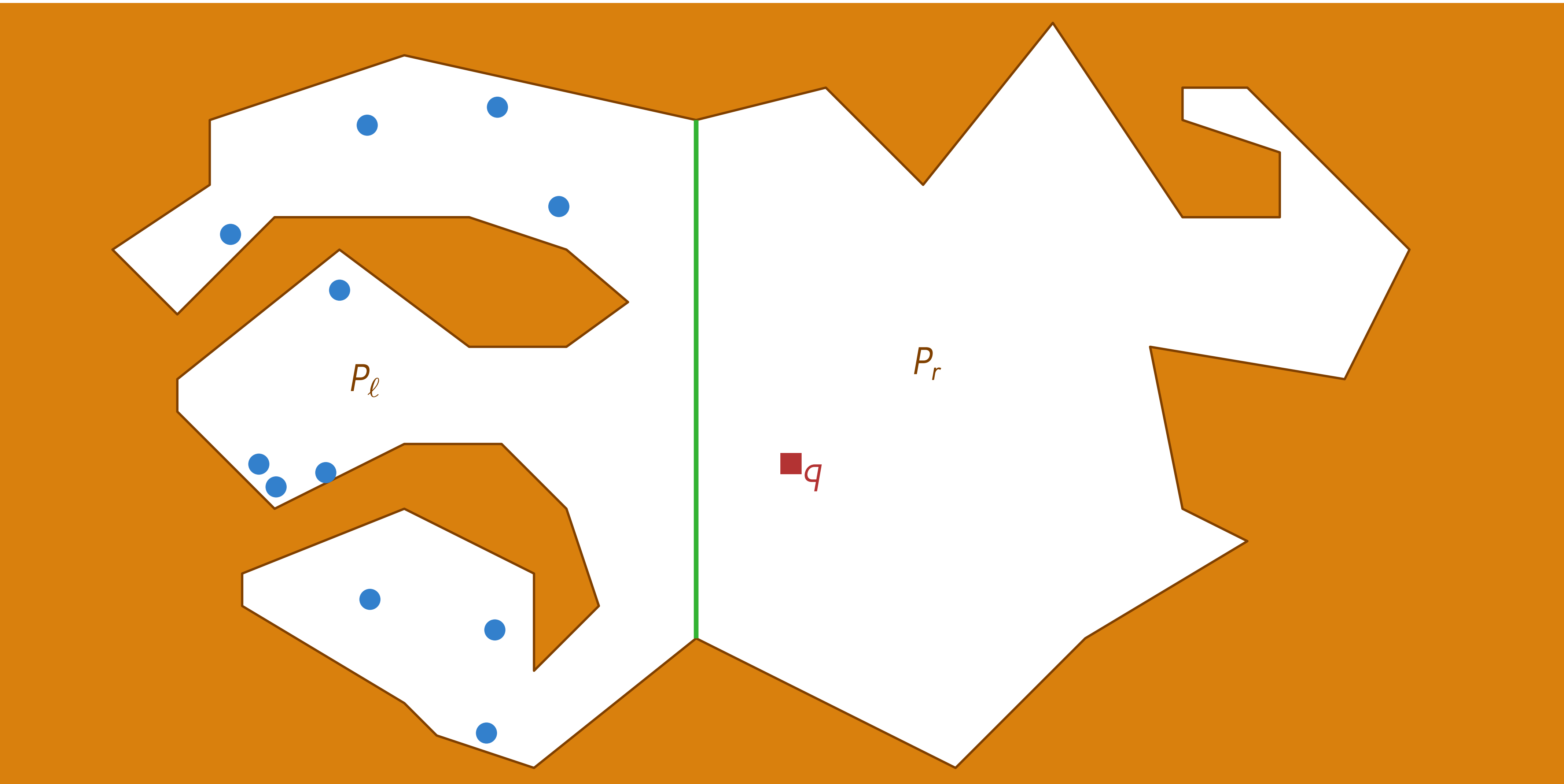
Splitting P

Main idea: Restrict S to P_ℓ and domain to P_r.

Lemma. $\exists$ dynamic DS to maintain $S \cap P_\ell$ s.t.

NN-queries with $q \in P_r$: $O(\log^3(n+m))$

updates: $O(\log^8(n+m))$



Results

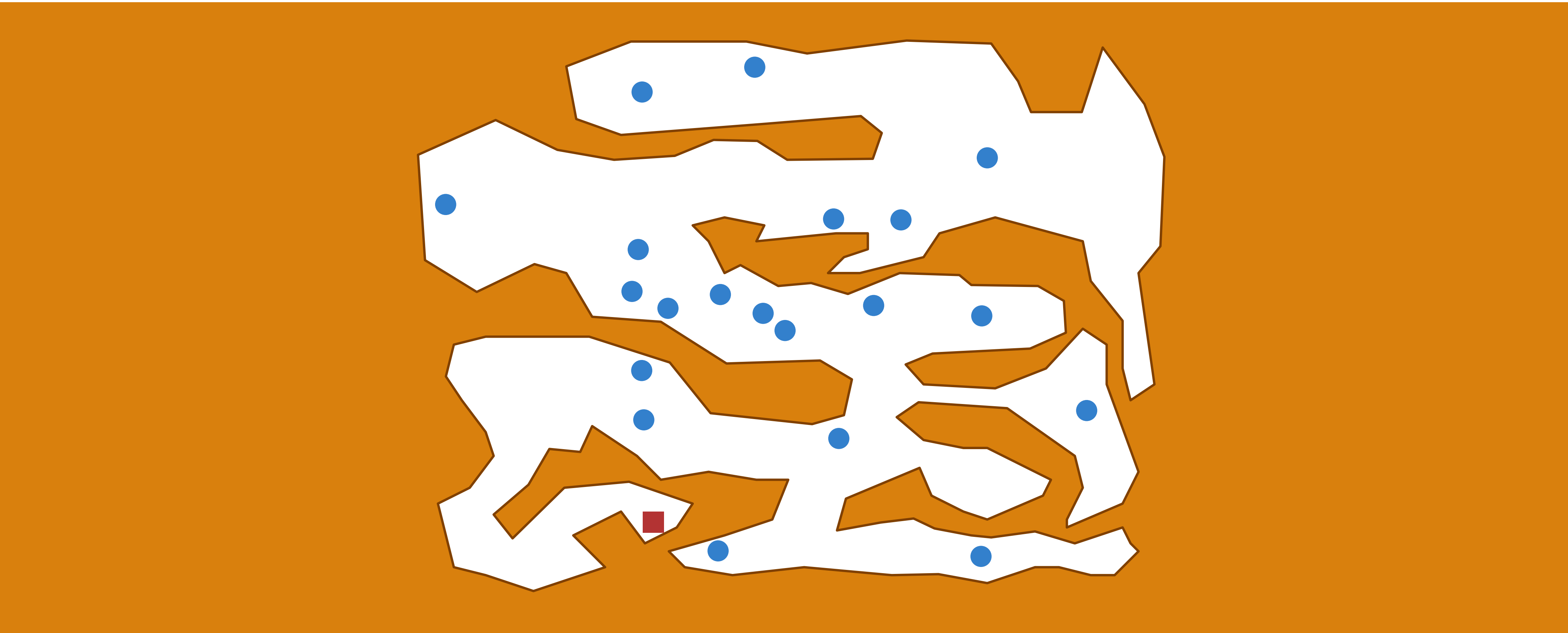
Given: P: simple polygon, S: dynamic set of point sites inside P

The data structure supports:

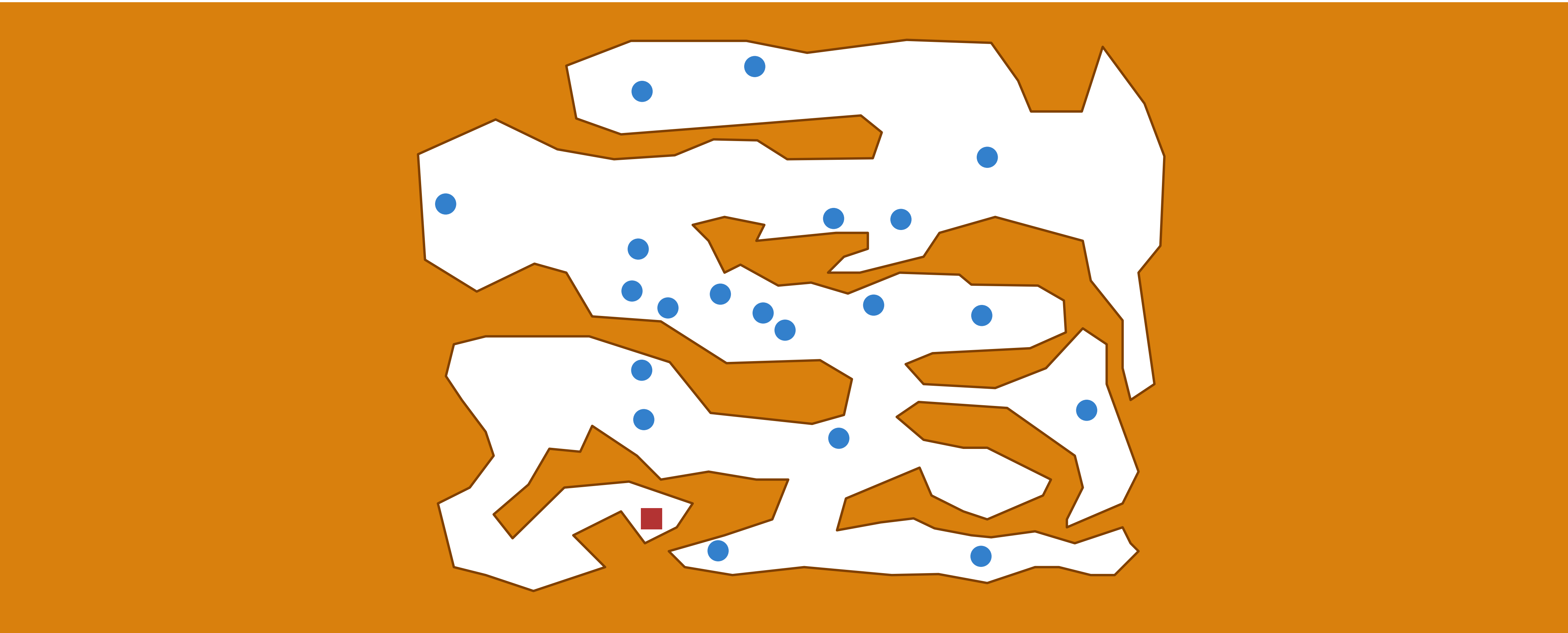
insert	$O(\log^7(n+m))$	expected, amortized
delete	$O(\log^9(n+m))$	expected, amortized
query	$O(\log^4(n+m))$	

expected space: $O(m + n\log^3 n \log m)$

n = max #sites in S
m = complexity P

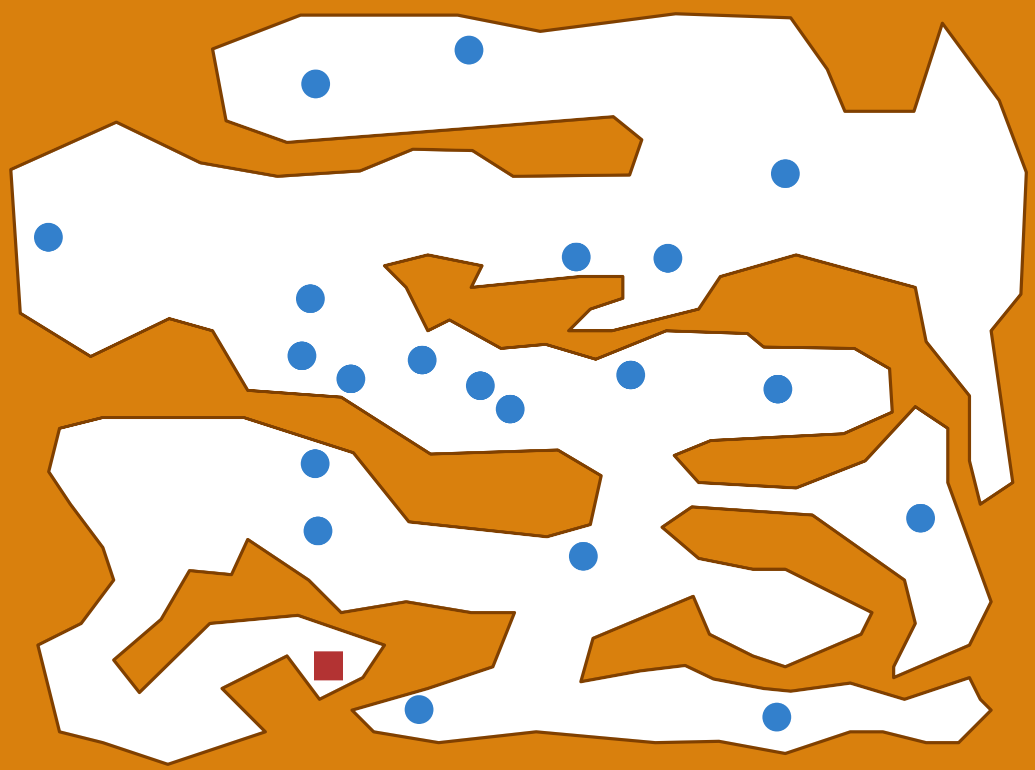


Done?



Done?

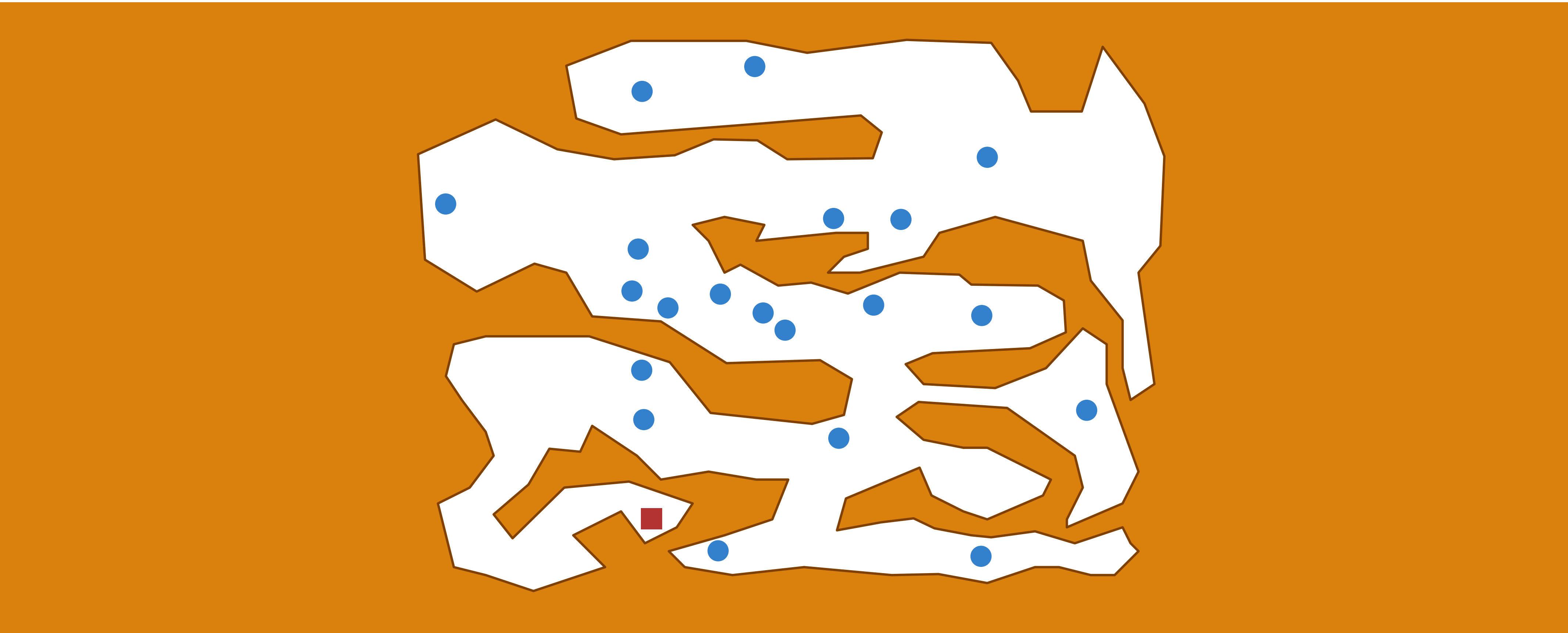
Question: can we shave some logs?



Done?

Question: can we shave some logs?

Question: Is there a $\Lambda_k(F)$ on the full polygon P?

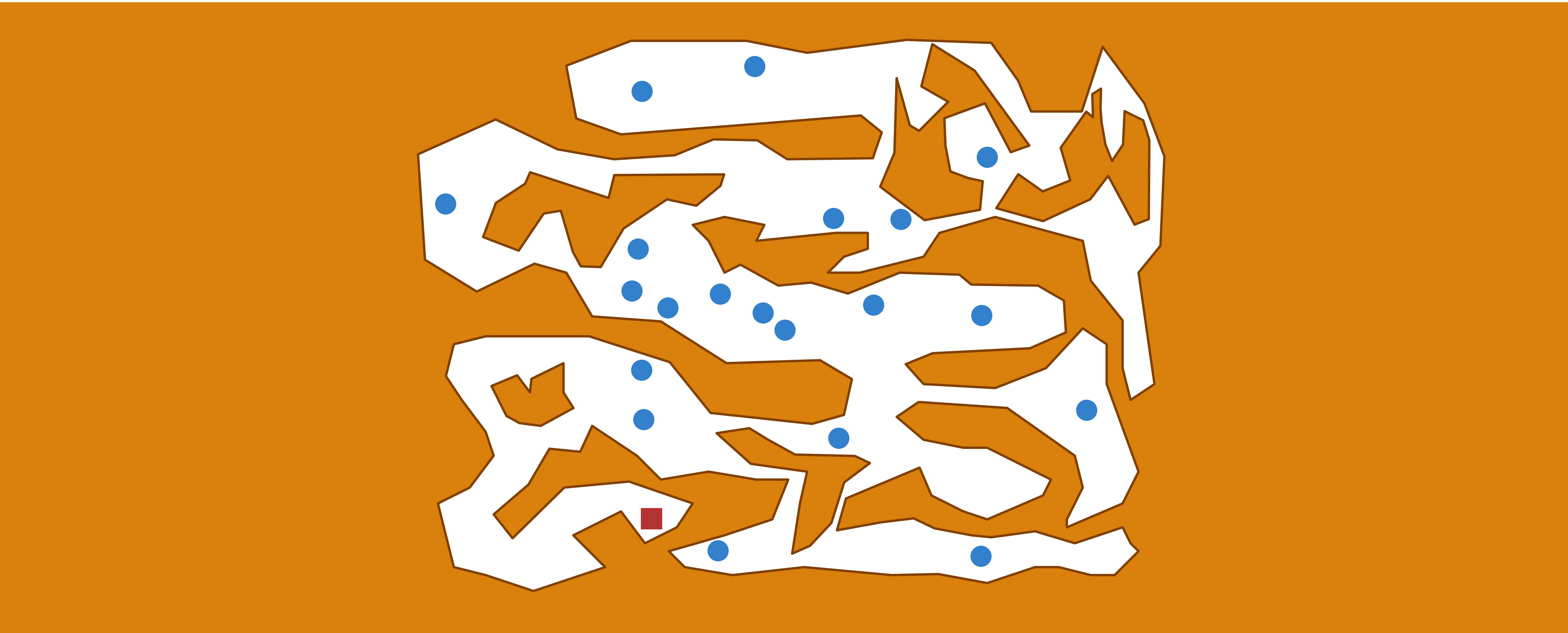


Done?

Question: can we shave some logs?

Question: Is there a $\Lambda_k(F)$ on the full polygon P?

Question: How about polygons with holes?

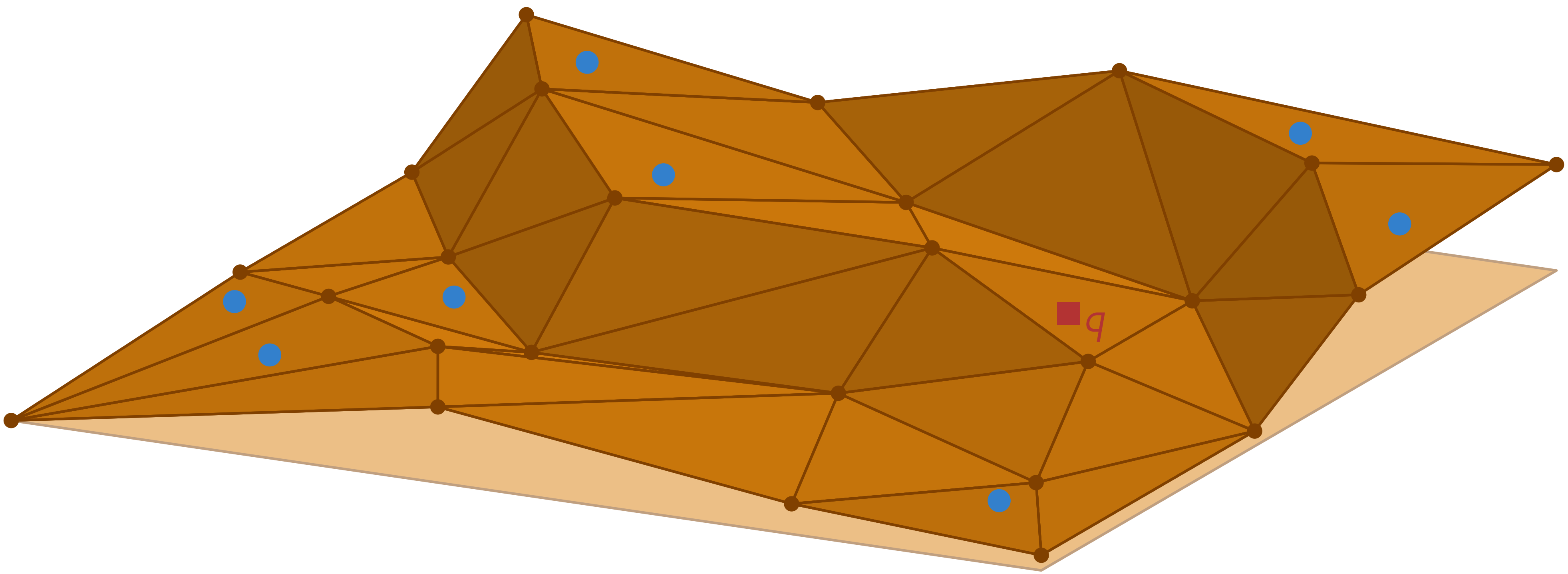


Done?

Question: can we shave some logs?

Question: Is there a $\Lambda_k(F)$ on the full polygon P?

Question: How about polygons with holes? or terrains?



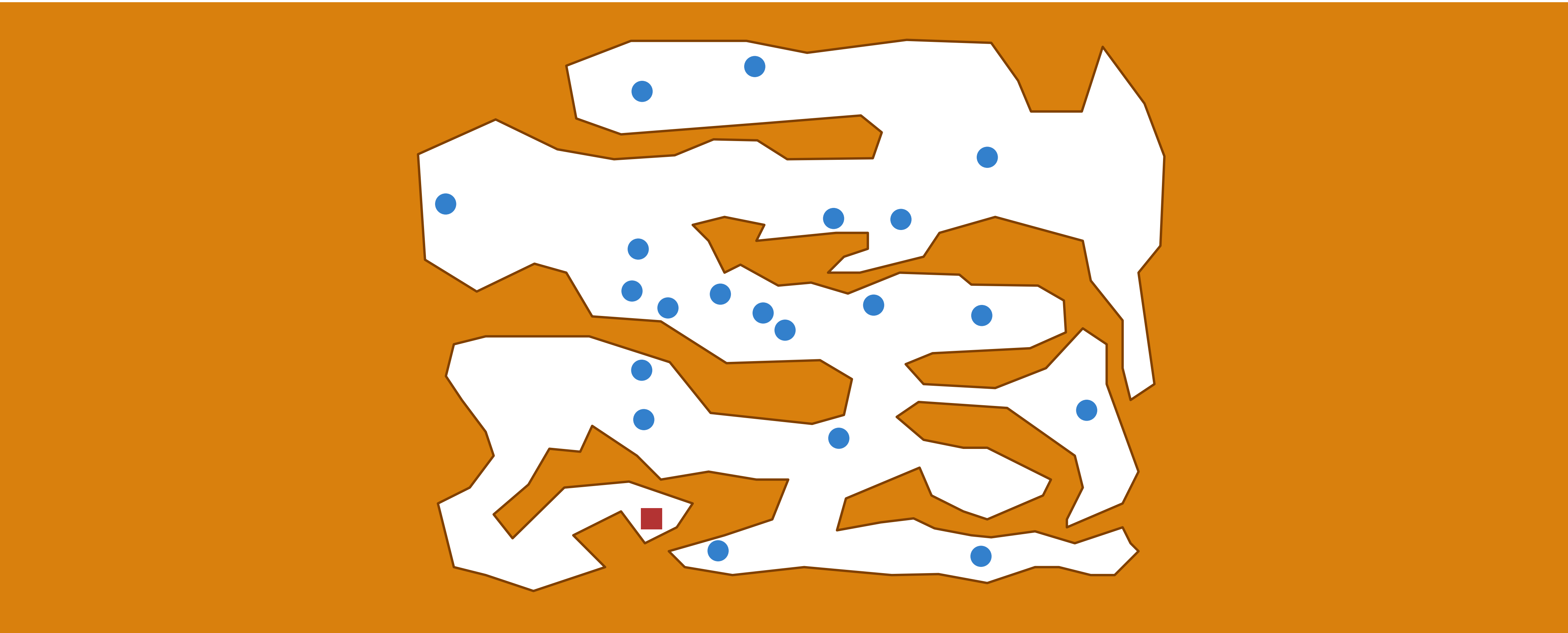
Done?

Question: can we shave some logs?

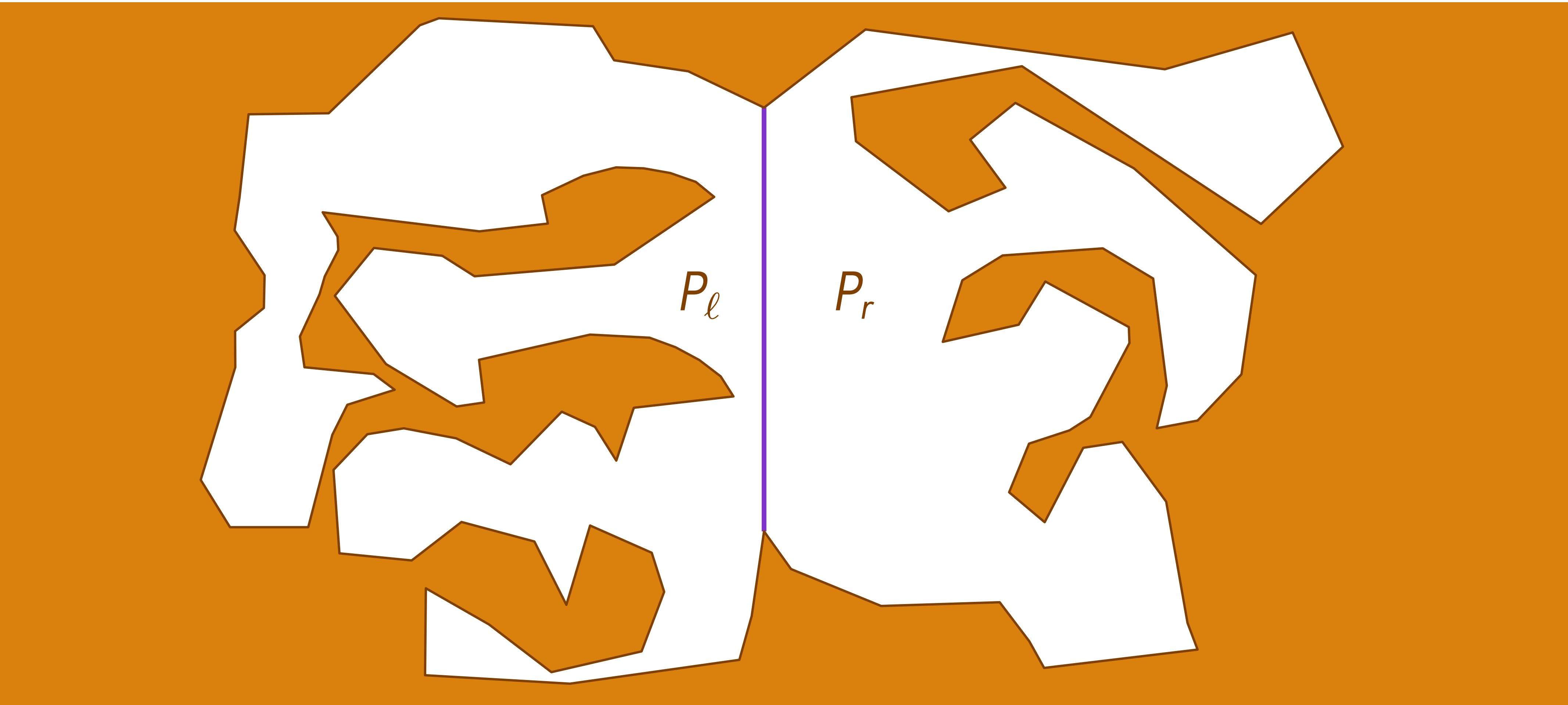
Question: Is there a $\Lambda_k(F)$ on the full polygon P?

Question: How about polygons with holes? or terrains?

Thank You!

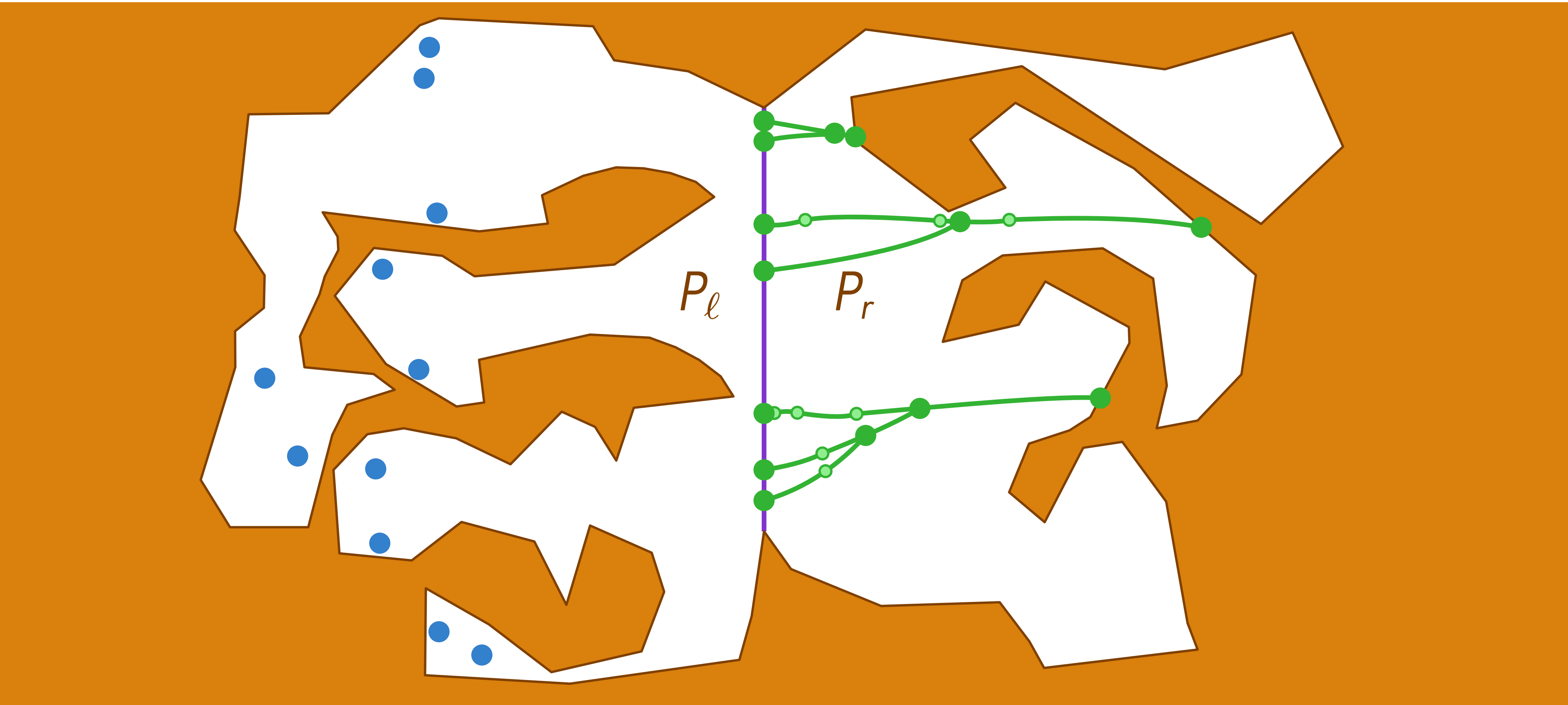


The Offline Data Structure



The Offline Data Structure

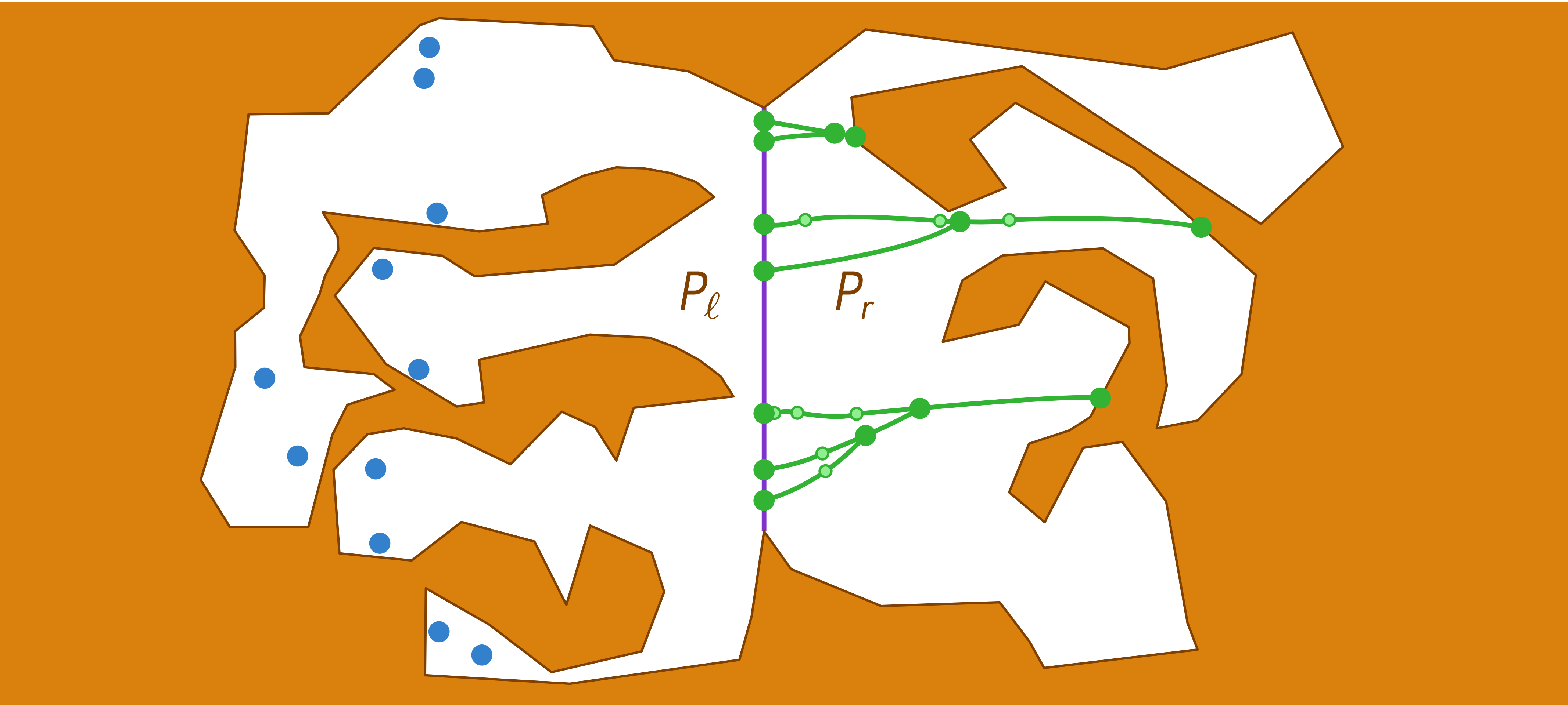
Obs. Let S_ℓ be k sites in P_ℓ.
The geodesic VD $VD(S_\ell)$ in P_r is a forest with $O(k)$ degree 1 and 3 vertices



The Offline Data Structure

Obs. Let S_ℓ be k sites in P_ℓ.
The geodesic VD $VD(S_\ell)$ in P_r is a forest with $O(k)$ degree 1 and 3 vertices

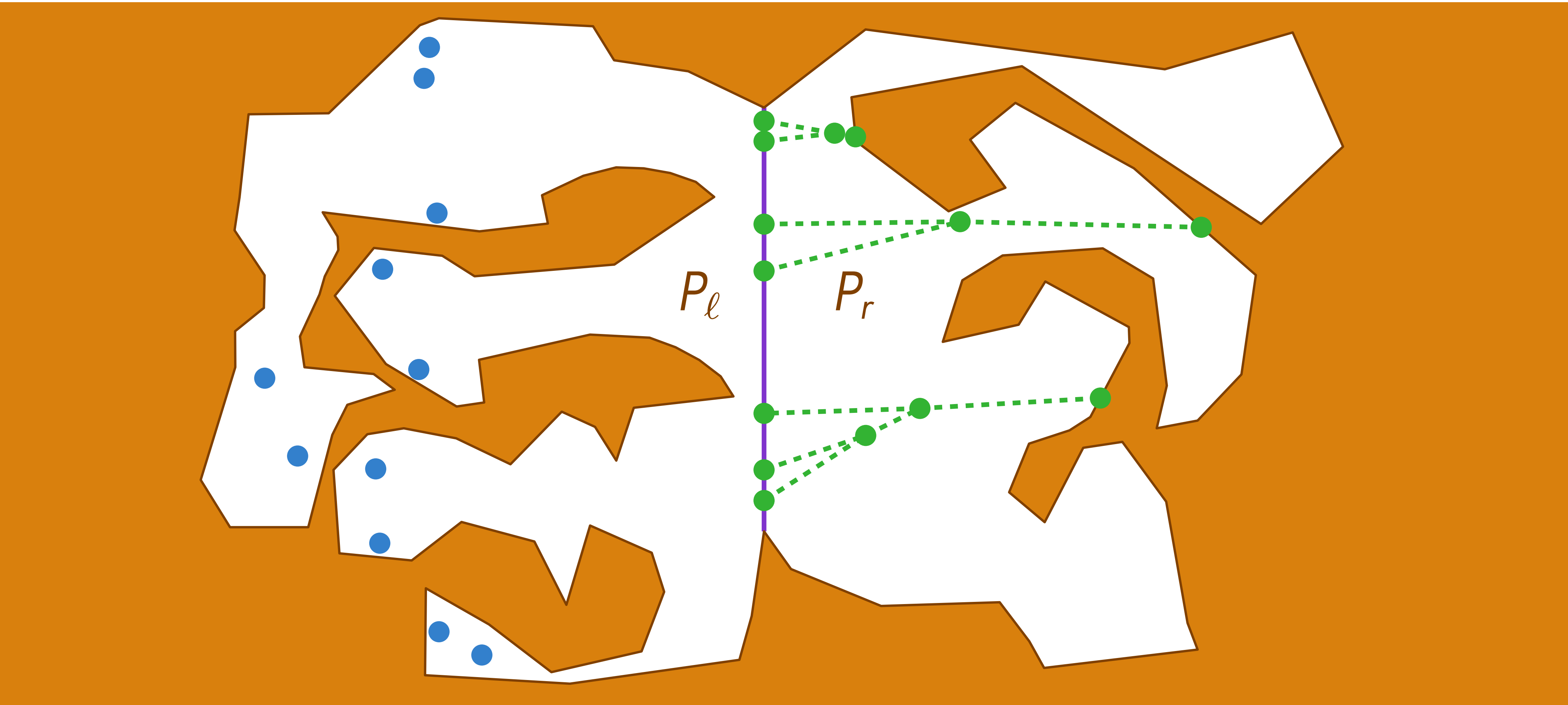
Main idea: Compute the locations of only those vertices and the topology of $VD(S)$



The Offline Data Structure

Obs. Let S_ℓ be k sites in P_ℓ.
The geodesic VD $VD(S_\ell)$ in P_r is a forest with $O(k)$ degree 1 and 3 vertices

Main idea: Compute the locations of only those vertices and the topology of $VD(S)$

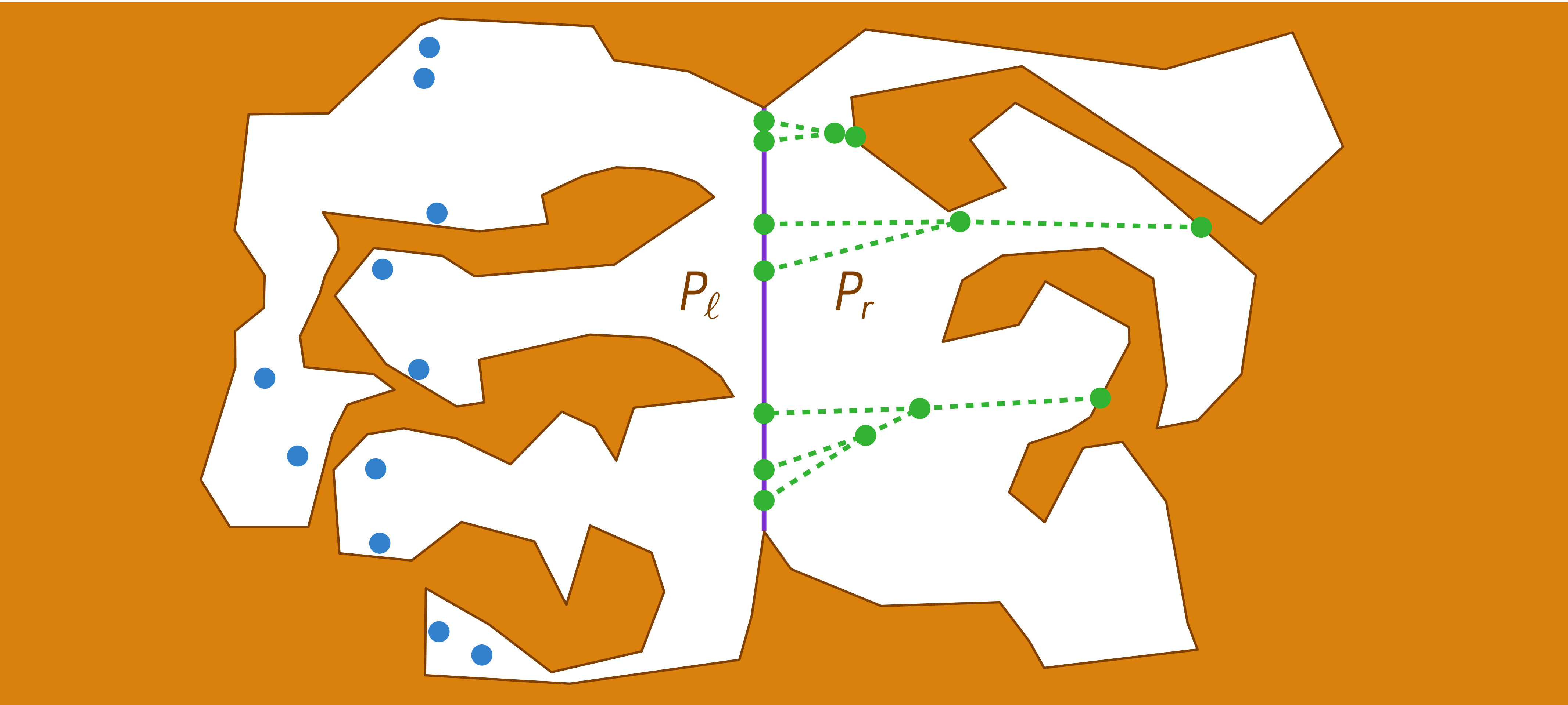


The Offline Data Structure

Obs. Let S_ℓ be k sites in P_ℓ.
The geodesic VD $VD(S_\ell)$ in P_r is a forest with $O(k)$ degree 1 and 3 vertices

Main idea: Compute the locations of only those vertices and the topology of $VD(S)$

$\Longrightarrow$ takes $O(k \log^2 m)$ time



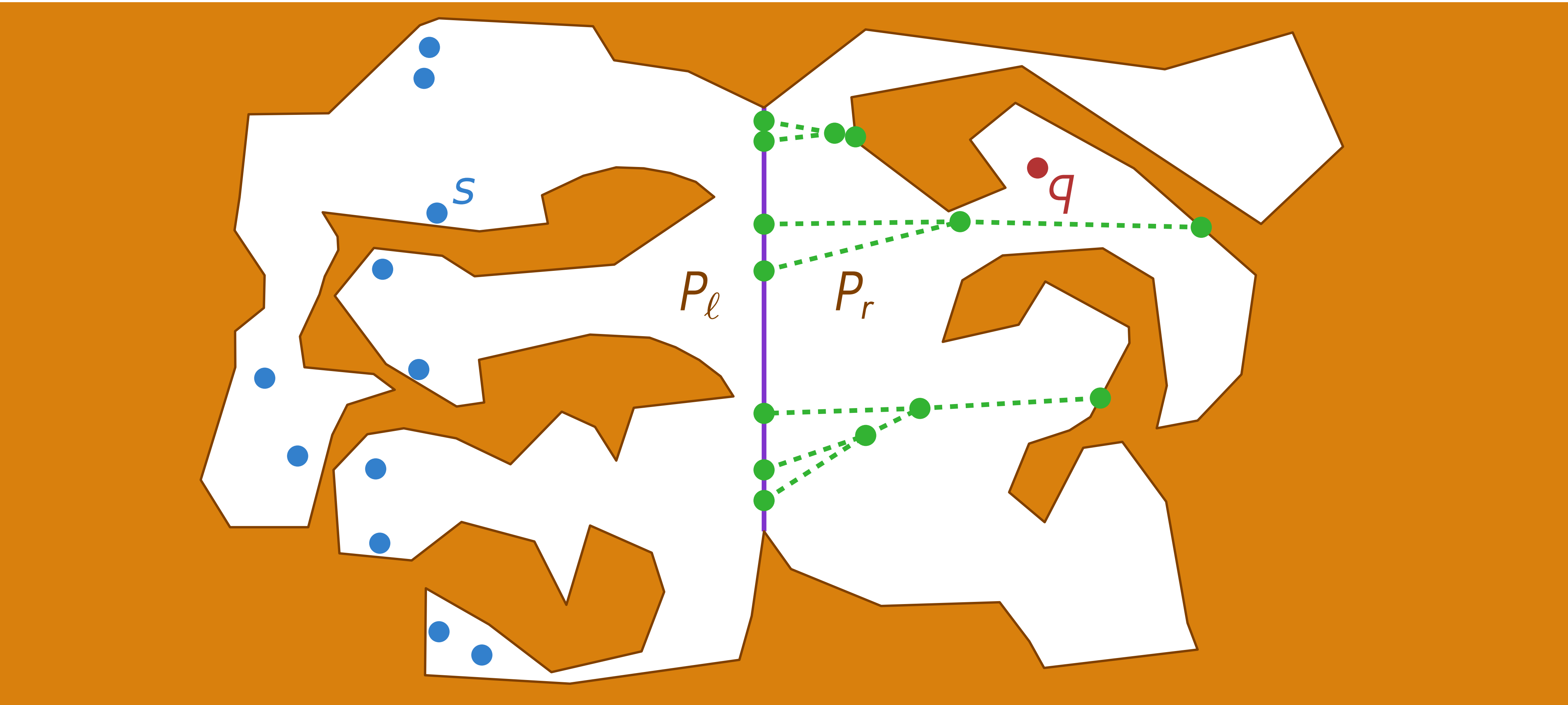
The Offline Data Structure

Obs. Let S_ℓ be k sites in P_ℓ.
The geodesic VD $VD(S_\ell)$ in P_r is a forest with $O(k)$ degree 1 and 3 vertices

Main idea: Compute the locations of only those vertices and the topology of $VD(S)$

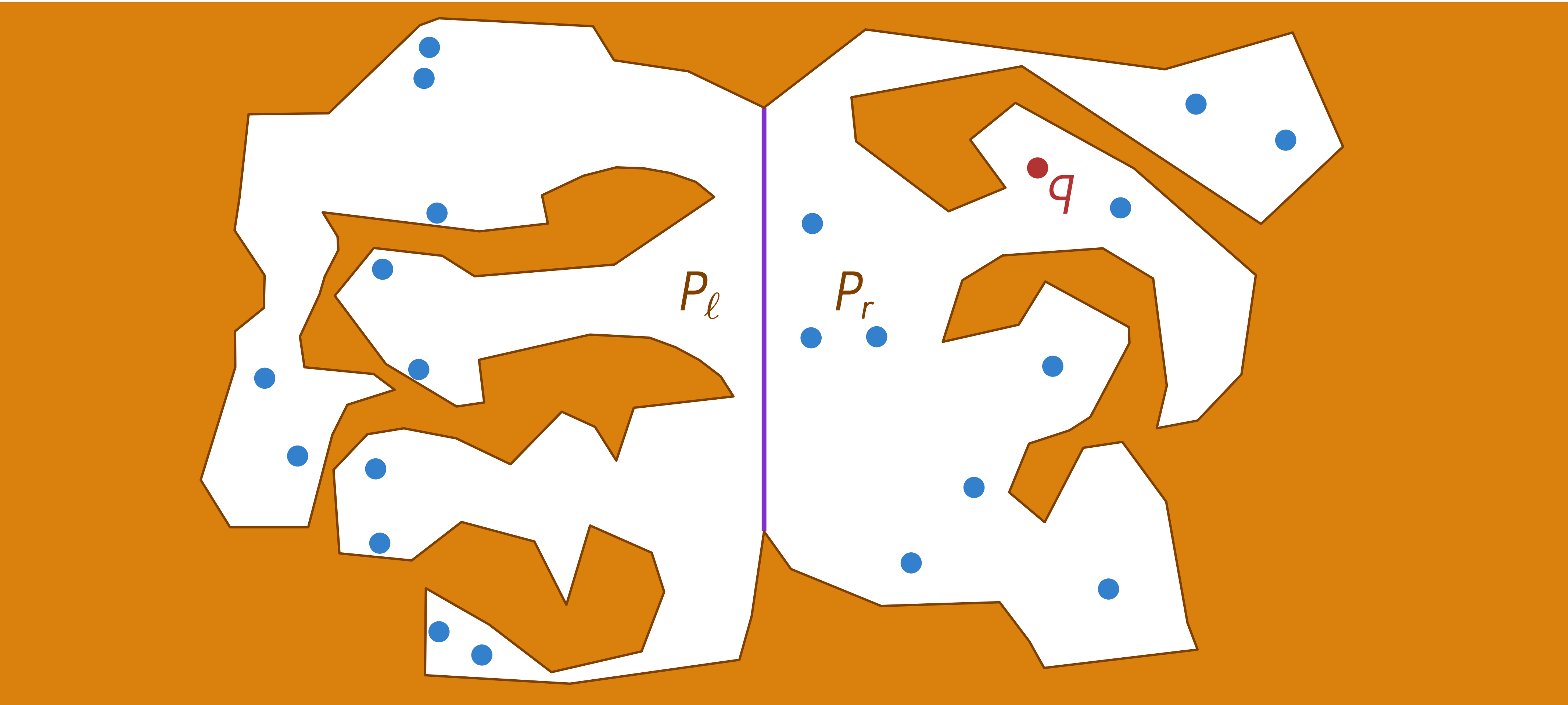
$\Longrightarrow$ takes $O(k \log^2 m)$ time

we can find $s \in S_\ell$ closest to $q \in P_r$ in $O(\log k \log m)$ time.



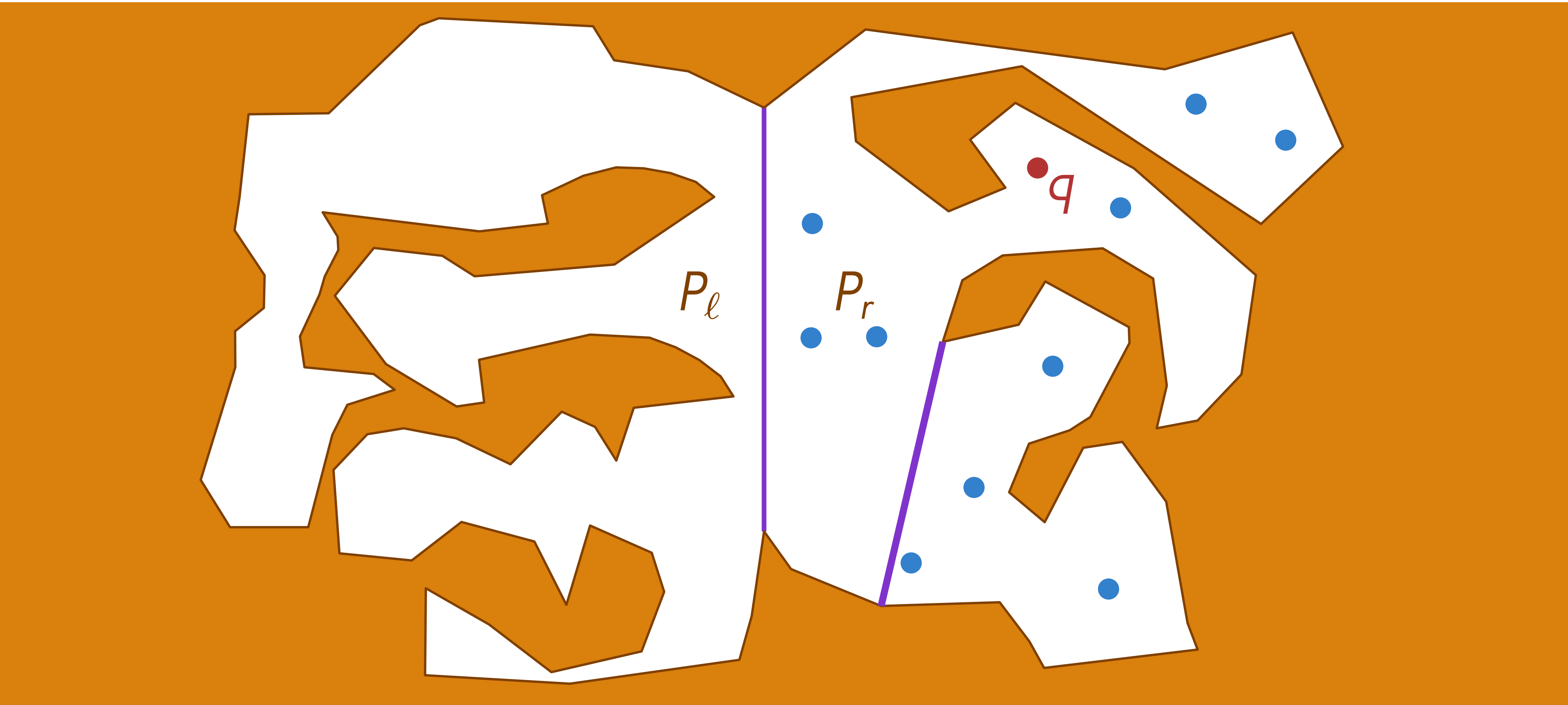
The Offline Data Structure

What about $S_r = S \cap P_r$?



The Offline Data Structure

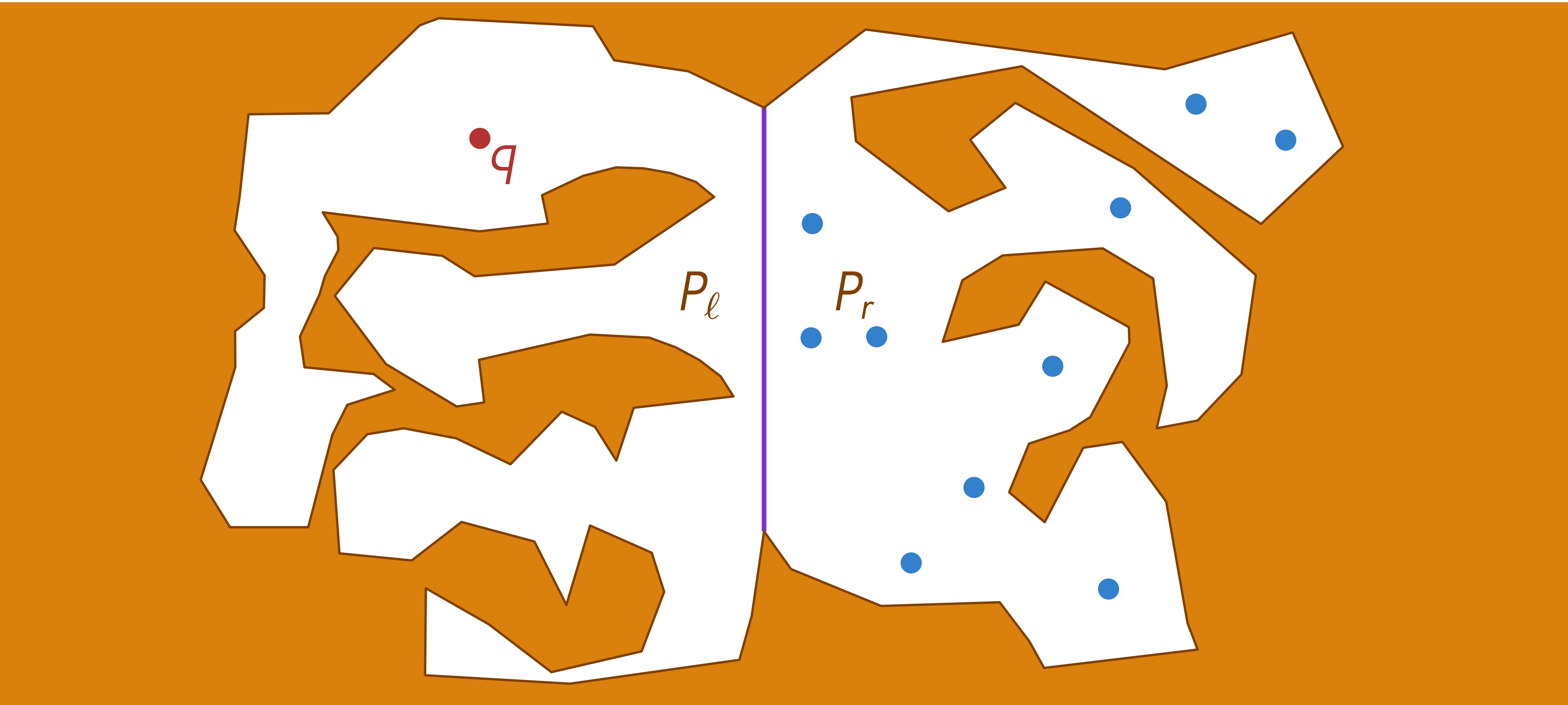
What about $S_r = S \cap P_r$? Recursively partition P_r



The Offline Data Structure

What about $S_r = S \cap P_r$? Recursively partition P_r

What if $q \in P_\ell$?

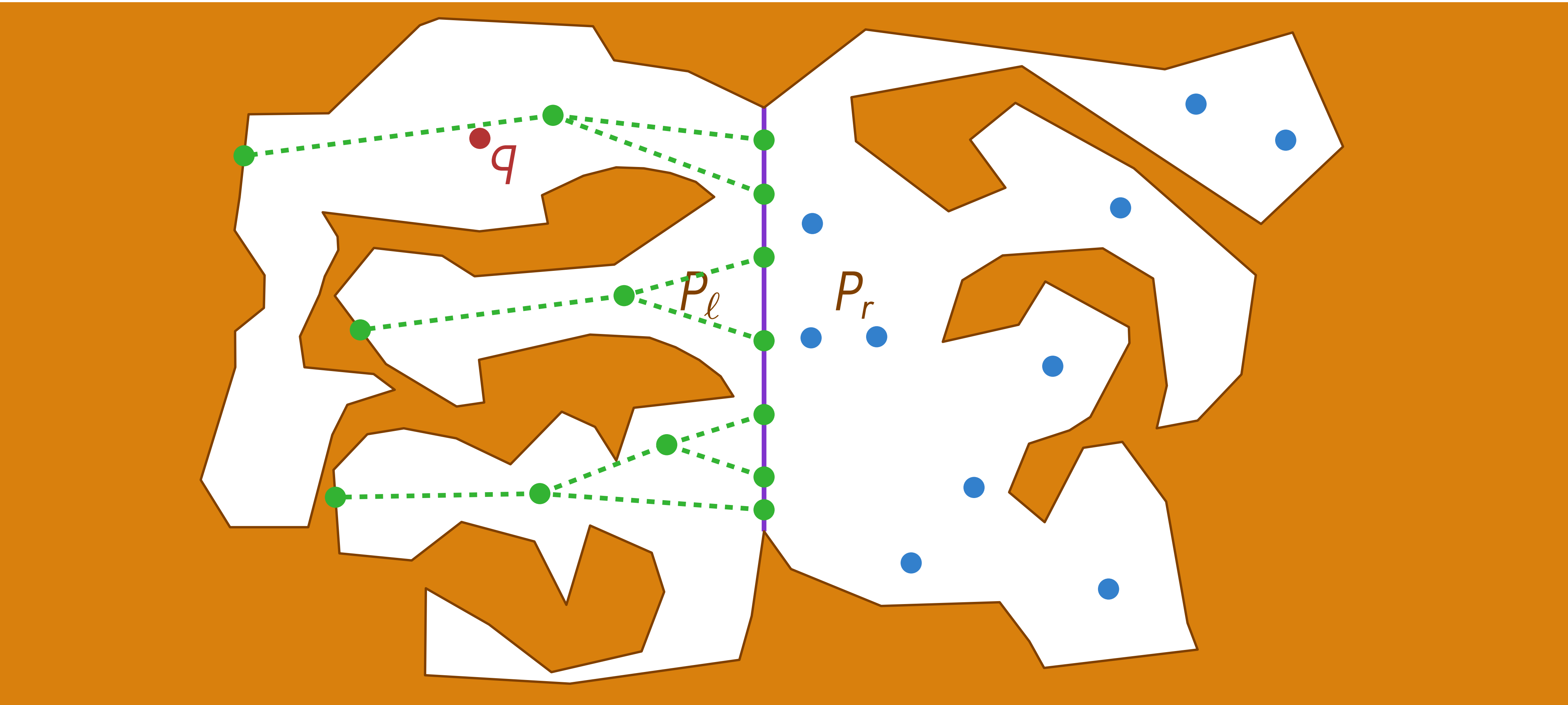


The Offline Data Structure

What about $S_r = S \cap P_r$? Recursively partition P_r

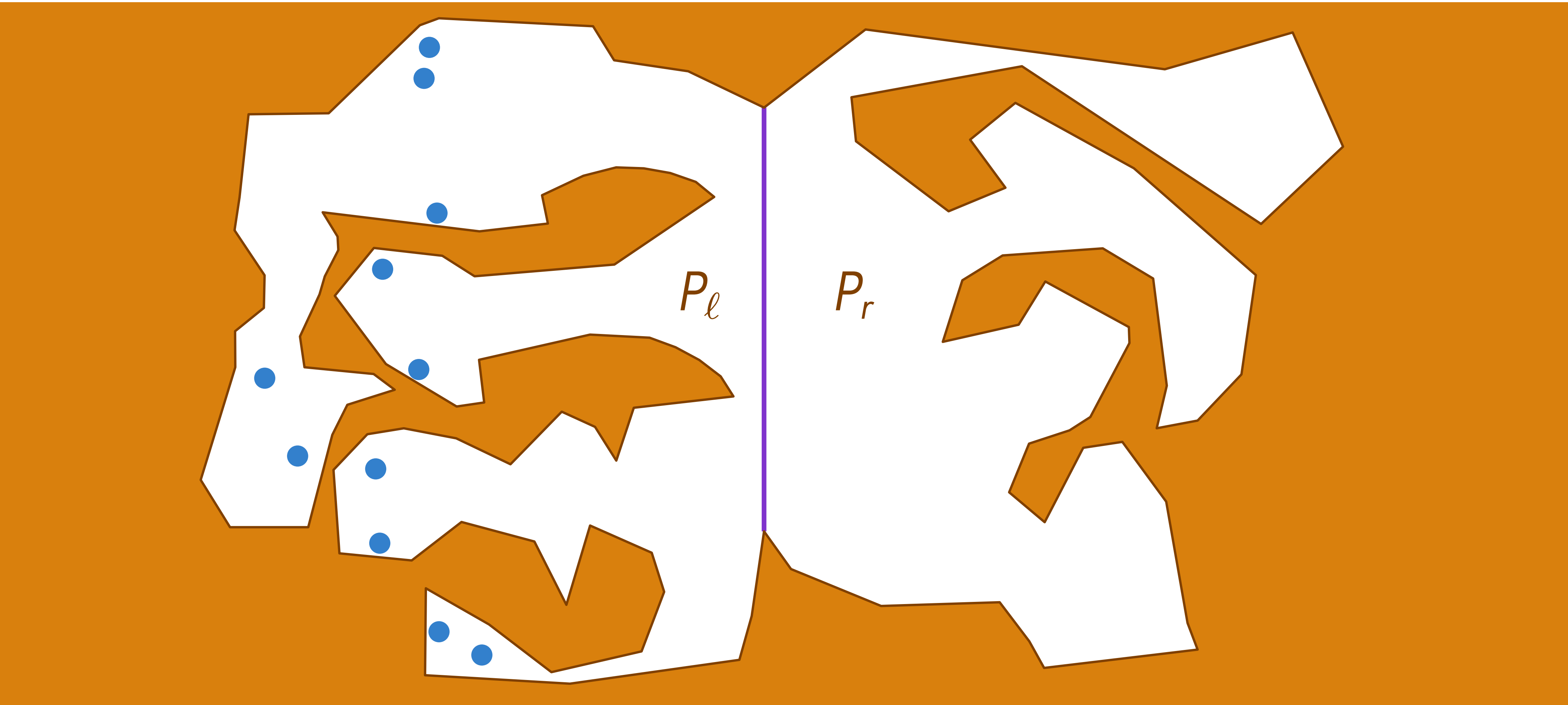
What if $q \in P_\ell$? Symmetric:

- Use $VD(S_r)$ to find closest site in S_r
- Recursively partition P_ℓ to find closest site in S_ℓ



The Offline Data Structure

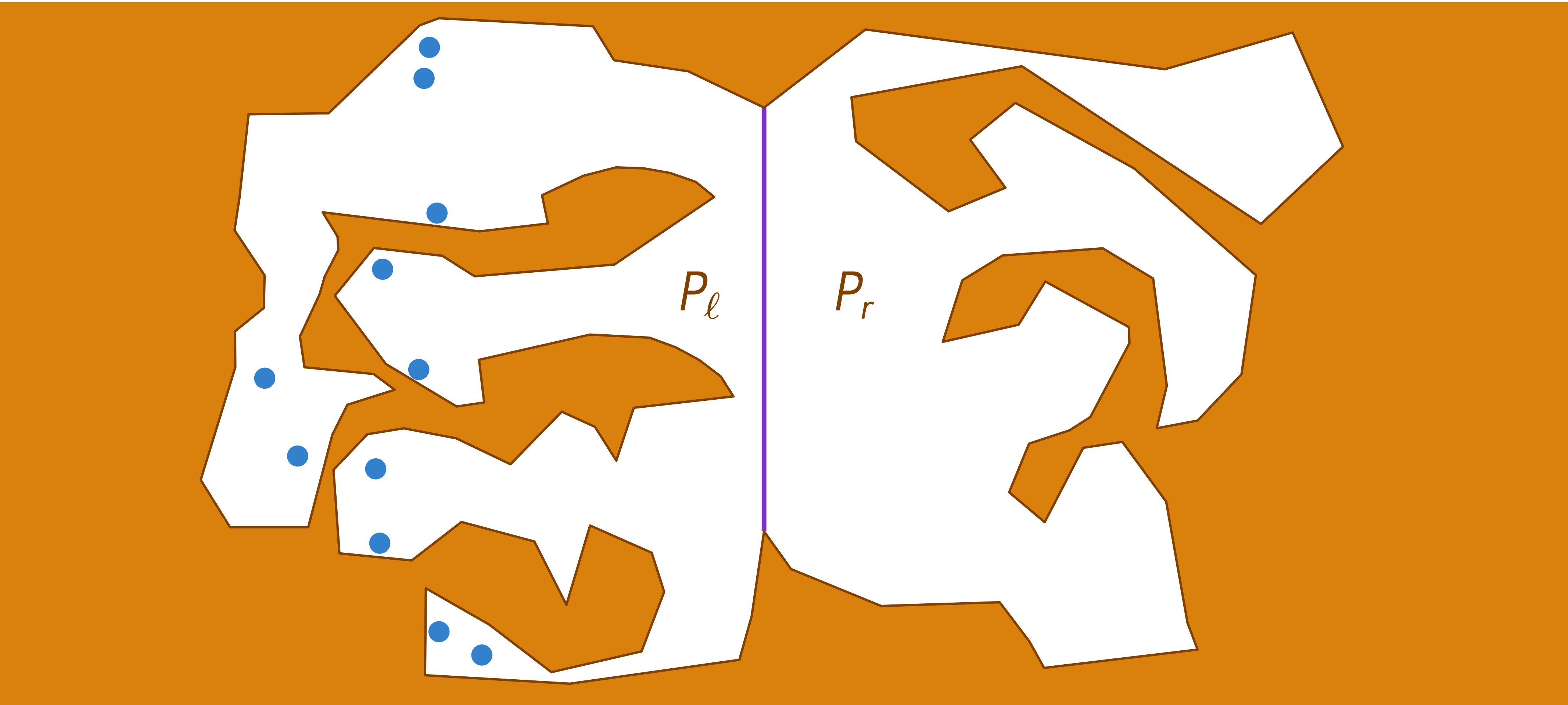
How to deal with updates?



The Offline Data Structure

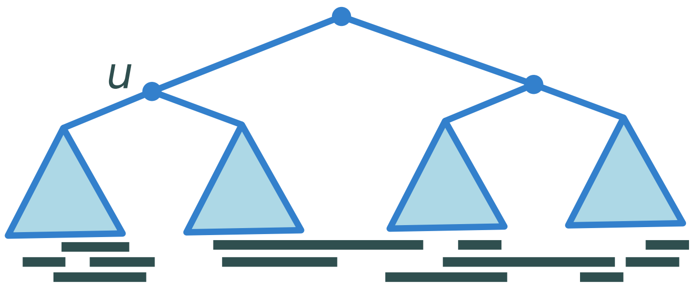
How to deal with updates?

Map each $s \in S_\ell$ to a time interval $I_s = [t_{insert\ s}, t_{delete\ s}]$



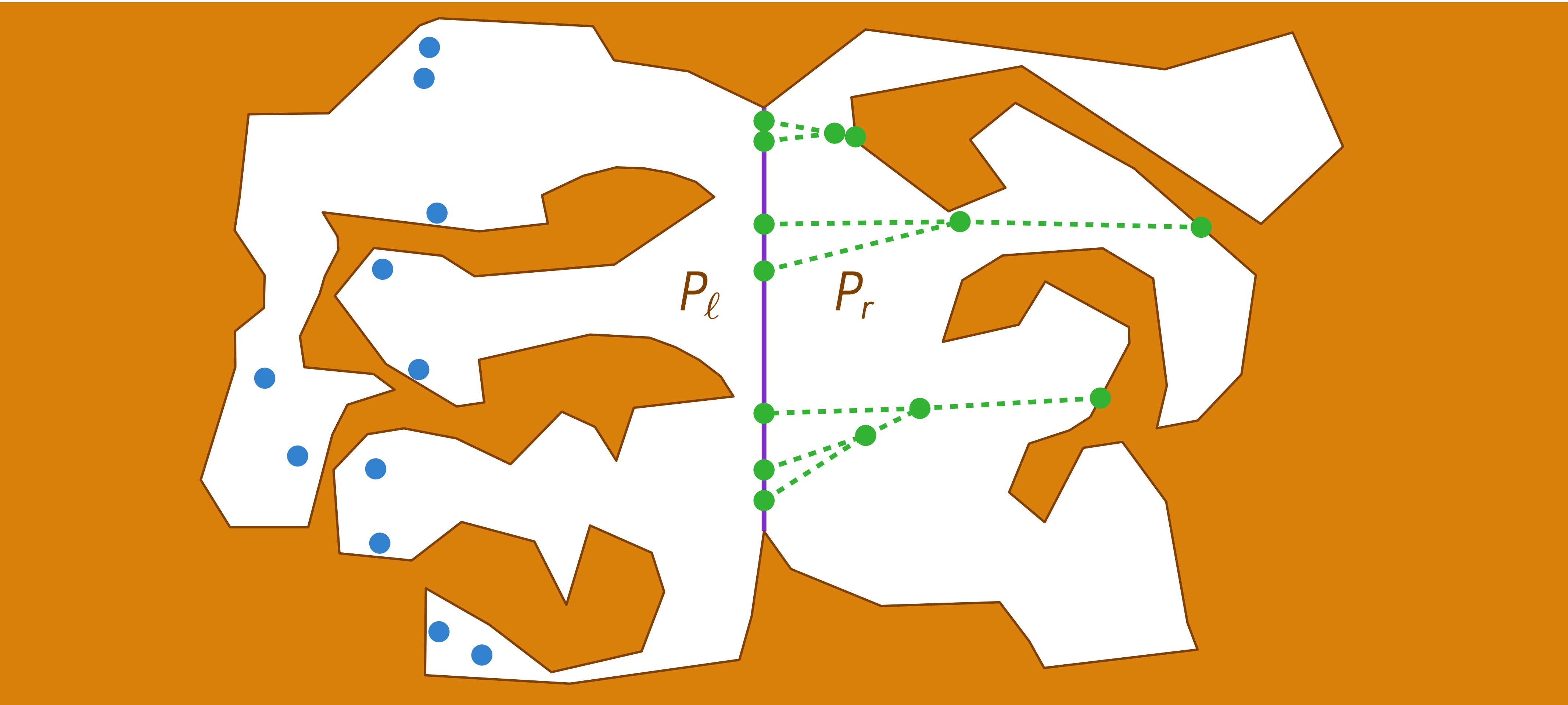
The Offline Data Structure

How to deal with updates?



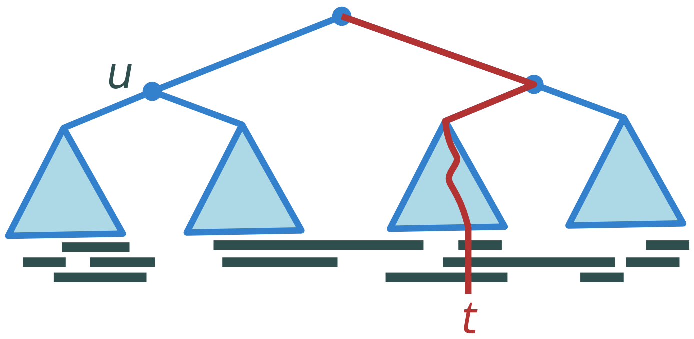
Map each $s \in S_\ell$ to a time interval $I_s = [t_{insert\ s}, t_{delete\ s}]$

Build $VD(S_u)$ for each u



The Offline Data Structure

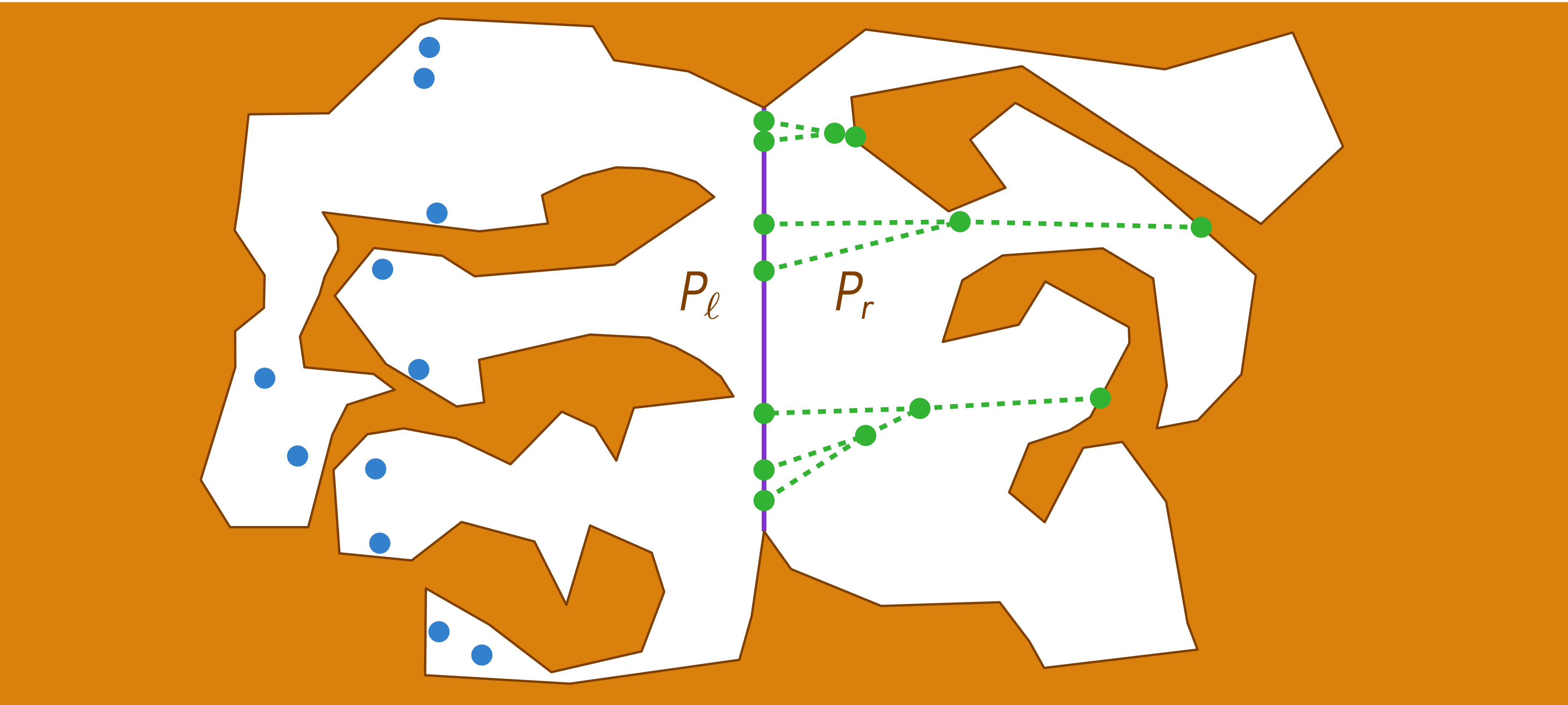
How to deal with updates?



Map each $s \in S_\ell$ to a time interval $I_s = [t_{insert\ s}, t_{delete\ s}]$

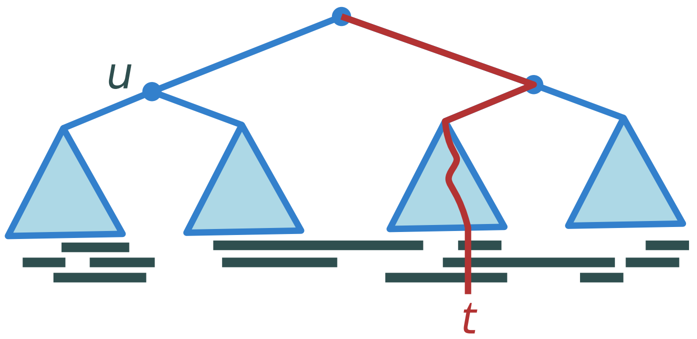
Build $VD(S_u)$ for each u

At query time t, select $O(\log n)$ nodes u, query each $VD(S_u)$.



The Offline Data Structure

How to deal with updates?



Map each $s \in S_\ell$ to a time interval $I_s = [t_{insert\ s}, t_{delete\ s}]$

Build $VD(S_u)$ for each u

At query time t, select $O(\log n)$ nodes u, query each $VD(S_u)$.

$\implies O(\log^4(n+m))$ time amortized updates and queries