Detecting the intersection of two convex shapes by searching on the 2-sphere

Samuel Hornus

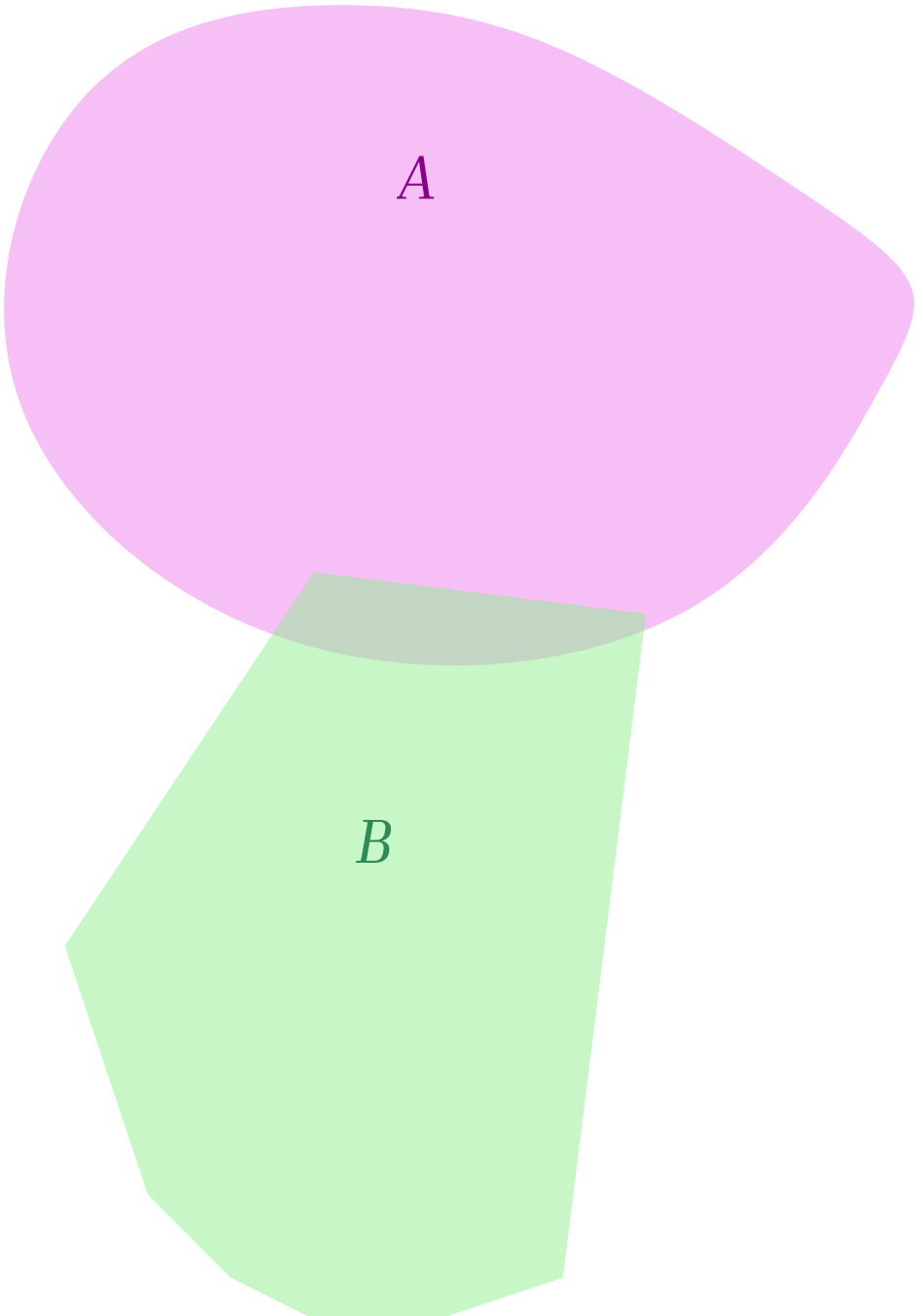
Solid and Physical Modeling conference
June 2017, Berkeley, CA

Loria
Laboratoire lorrain de recherche en informatique et ses applications

Inria

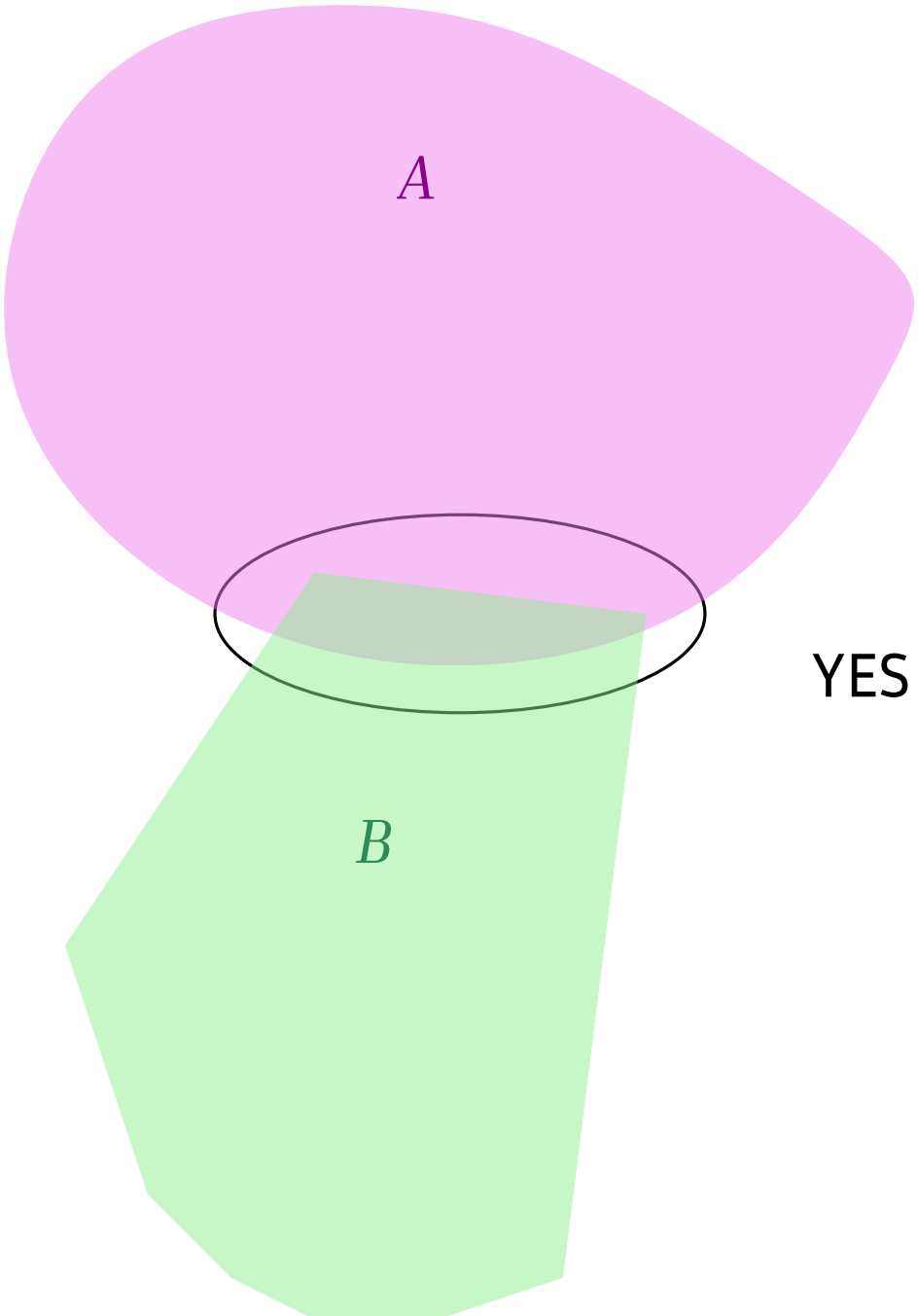
UNIVERSITÉ DE LORRAINE

A binary decision problem



Do A and B intersect?

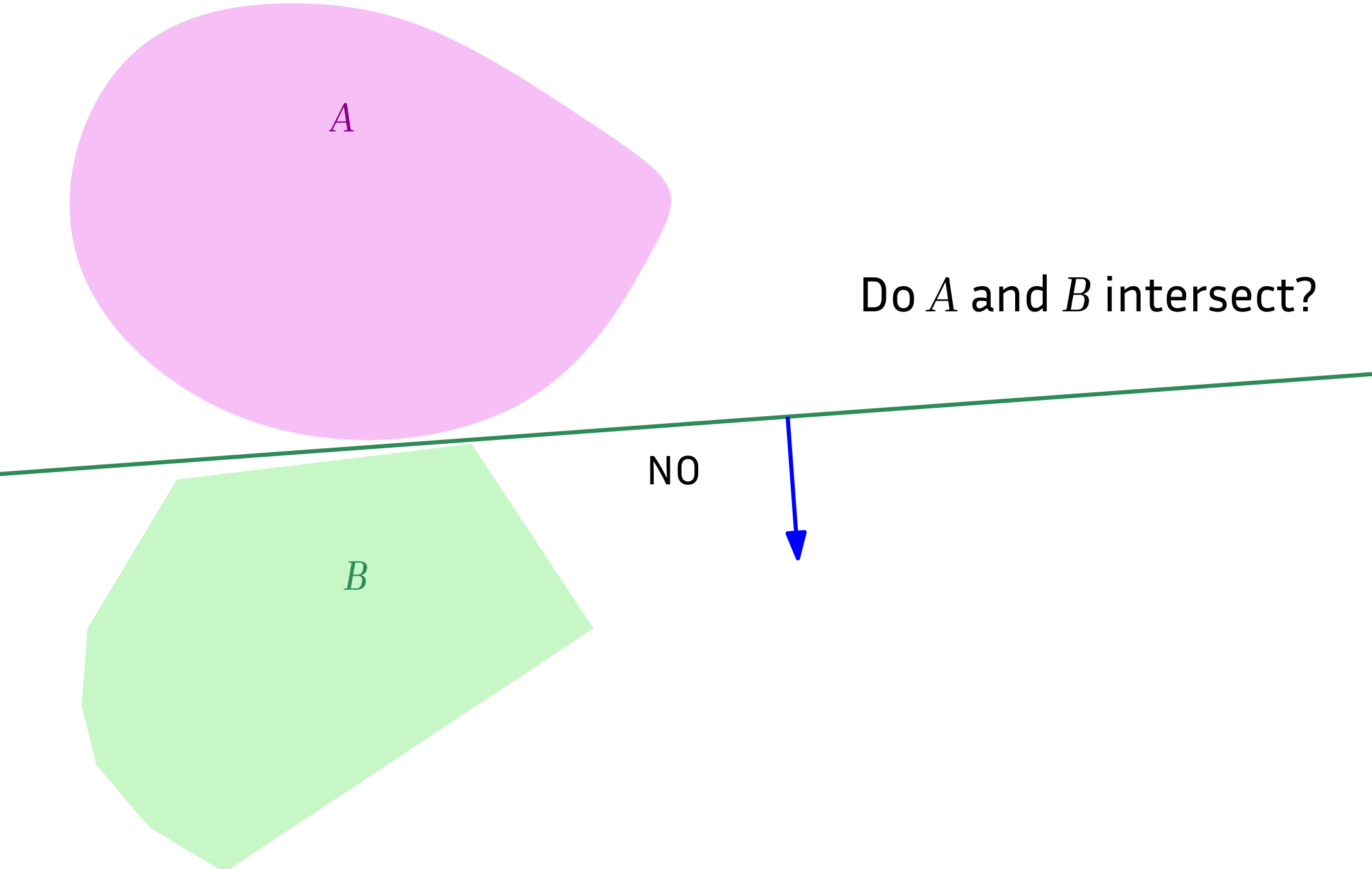
A binary decision problem



YES

Do A and B intersect?

A binary decision problem



Related work

- Spheres: 1 interval test [Folklore]
- Axis-aligned boxes: 3 interval tests [Folklore]
- Oriented boxes:
  Separating Axis Test [Gottschalk, Lin and Manocha 1996]
- Polytope and axis-aligned box:
  preprocessed silhouettes [Greene 1994]
- Ellipsoids [Eberly 2008]
- Zonotopes [Guibas *et al.* 2003]
- With motion [Guibas, Kim, Manocha *et al.*]
- Any polyhedra (GJK) [Gilbert, Johnson and Keerthi 1988]
- Any convexes (GF/GJK) [Gilbert and Foo 1990]
- Improvements to GJK [van der Bergen 1999, Cameron 1997]
- Books [Ericson 2004, Eberly 2008]
- Survey [Jiménez *et al.* 2001]
- Theory for polyhedra [Barba and Langerman 2015]

Our contribution “DSS” (Decision Sphere Search)

DSS is a new technique for testing any 2 convex shapes

Our contribution “DSS” (Decision Sphere Search)

DSS is a new technique for testing any 2 convex shapes

DSS is similar *in spirit* to GF/GJK

- Shape agnostic
- Same mild assumption
- Iterative

Our contribution "DSS" (Decision Sphere Search)

DSS is a new technique for testing any 2 convex shapes

DSS is similar *in spirit* to GF/GJK

- Shape agnostic
- Same mild assumption
- Iterative

DSS has advantages

- In general, a bit faster, fewer iterations
- Numerically more robust
- Easier to understand
- Easier to implement (even exactly)

Plan

Problem reduction & tools

The DSS algorithm

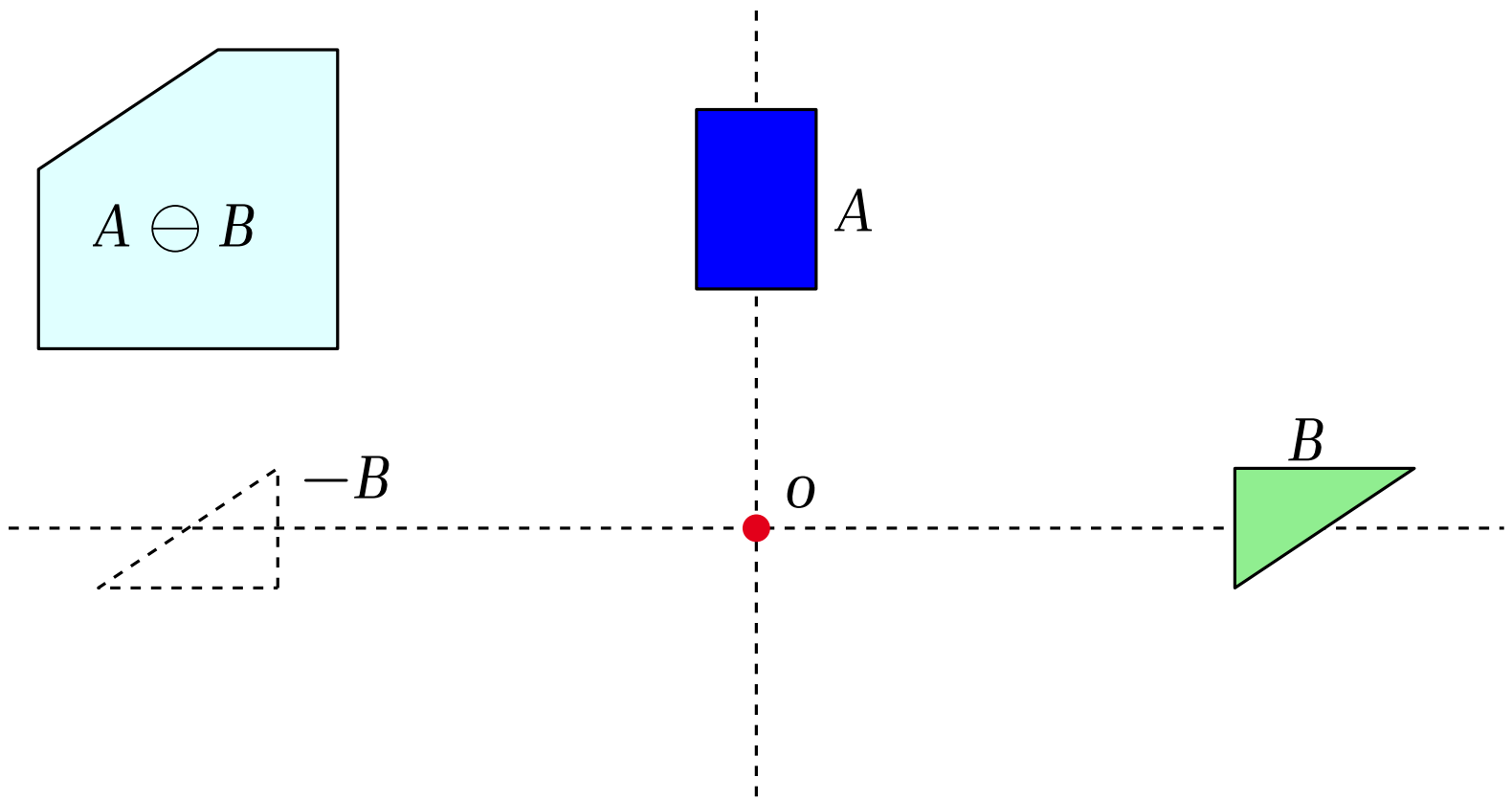
Theoretical comparison of GF/GJK and DSS

Practical comparison

Reduction to a single convex shape

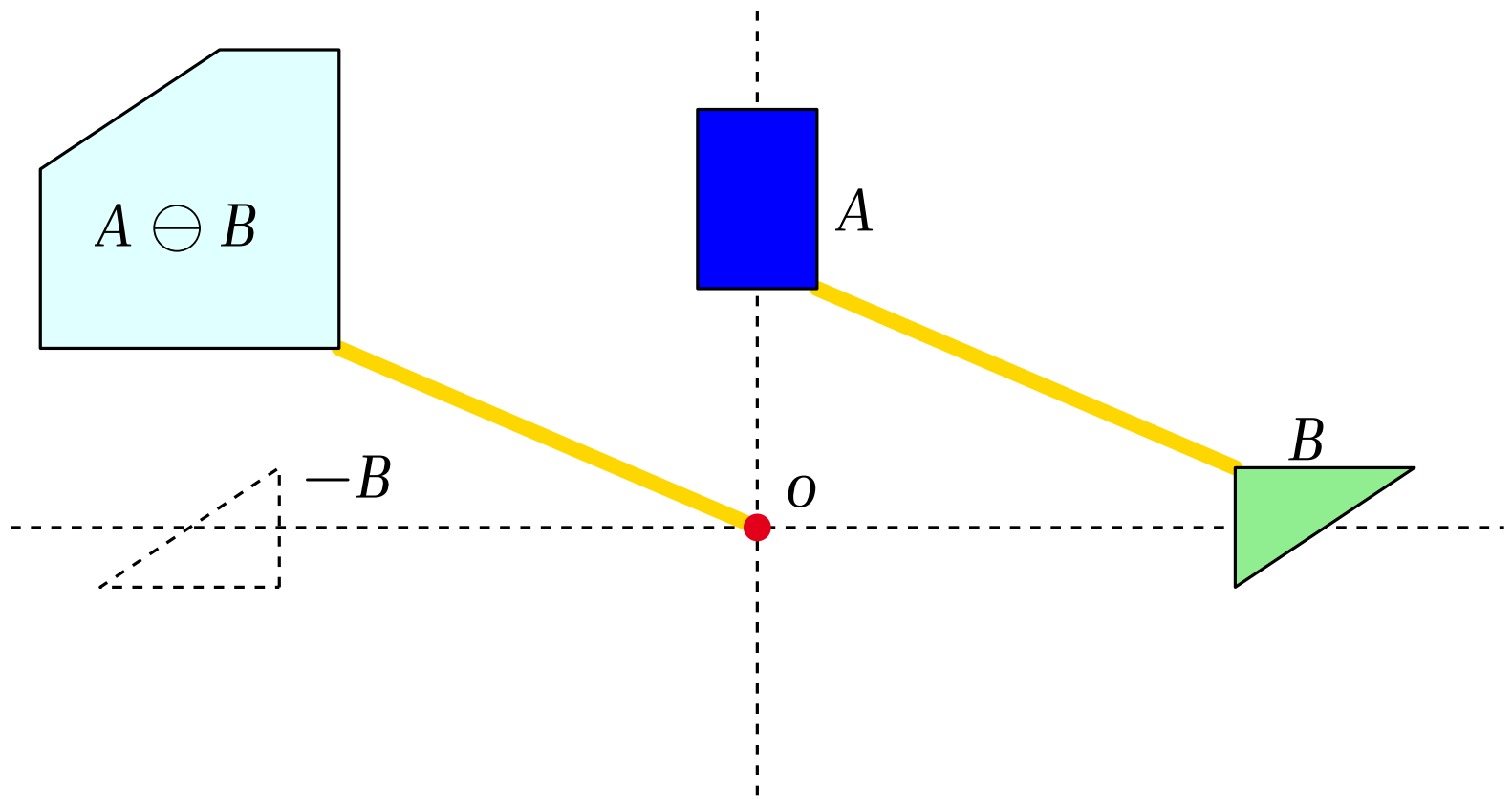
Minkowski difference: $A \ominus B = \{a - b \mid a \in A, b \in B\}$

$= A \oplus (-B)$



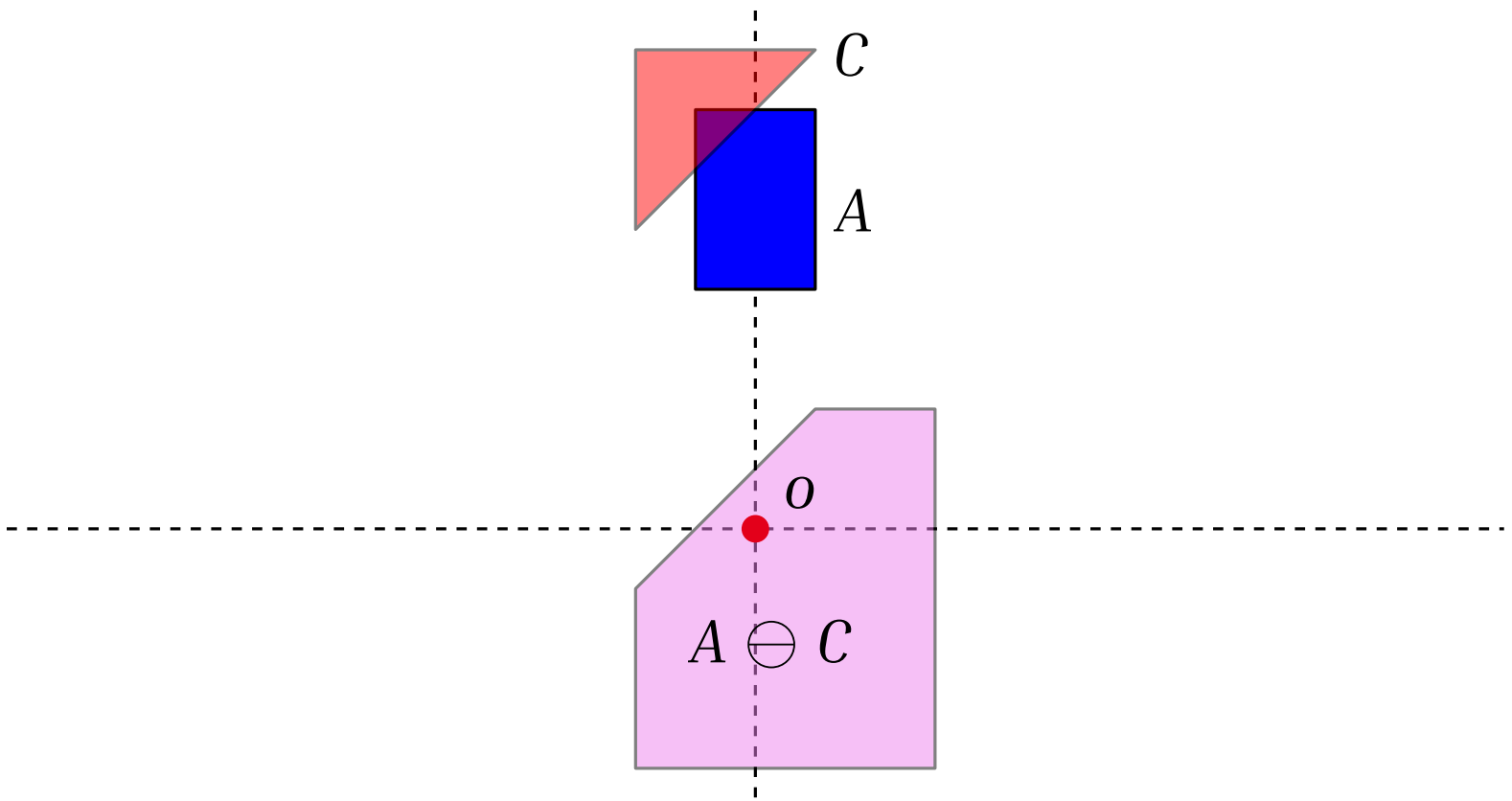
Reduction to a single convex shape

$$d(A,B) = d(o, A \ominus B)$$
$$(A \cap B) = \emptyset \Leftrightarrow o \notin (A \ominus B)$$



Reduction to a single convex shape

$$d(A, C) = d(o, A \ominus C)$$
$$(A \cap C) \neq \emptyset \Leftrightarrow o \in (A \ominus C)$$



Reduction to a single convex shape

$$d(A, C) = d(o, A \ominus C)$$
$$(A \cap C) \neq \emptyset \Leftrightarrow o \in (A \ominus C)$$

Given a convex P,

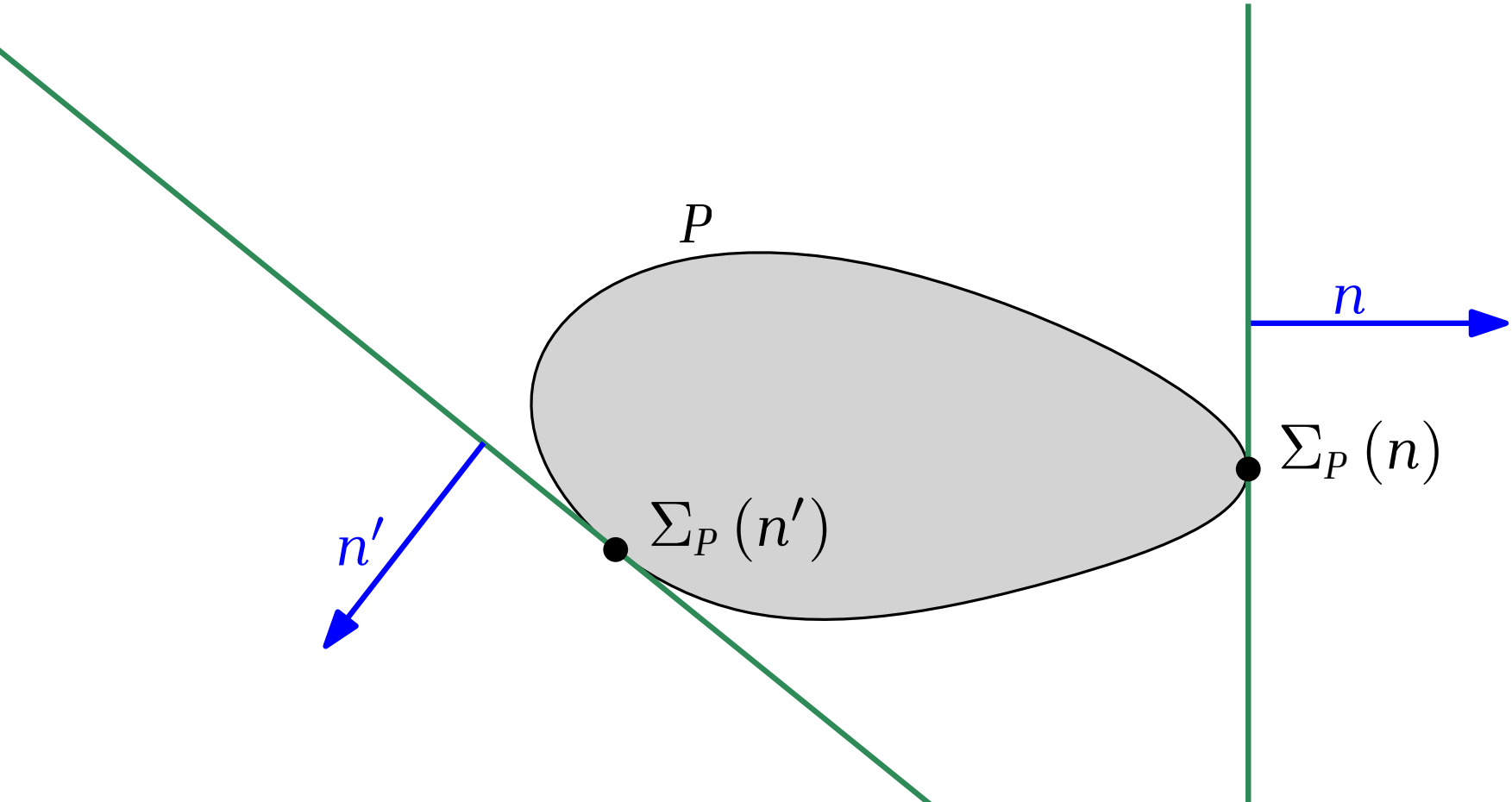
decide if $o \in P$ or not

Tools

$\mathbb{S}^2$ the unit sphere; n a direction in $\mathbb{S}^2$; P a convex shape.

The support function $h_P : \mathbb{S}^2 \to \mathbb{R}, \quad n \mapsto \max_{x \in P} (x \cdot n)$

The extremal function $\Sigma_P : \mathbb{S}^2 \to \mathbb{R}^3, \quad n \mapsto \arg\max_{x \in P} (x \cdot n)$

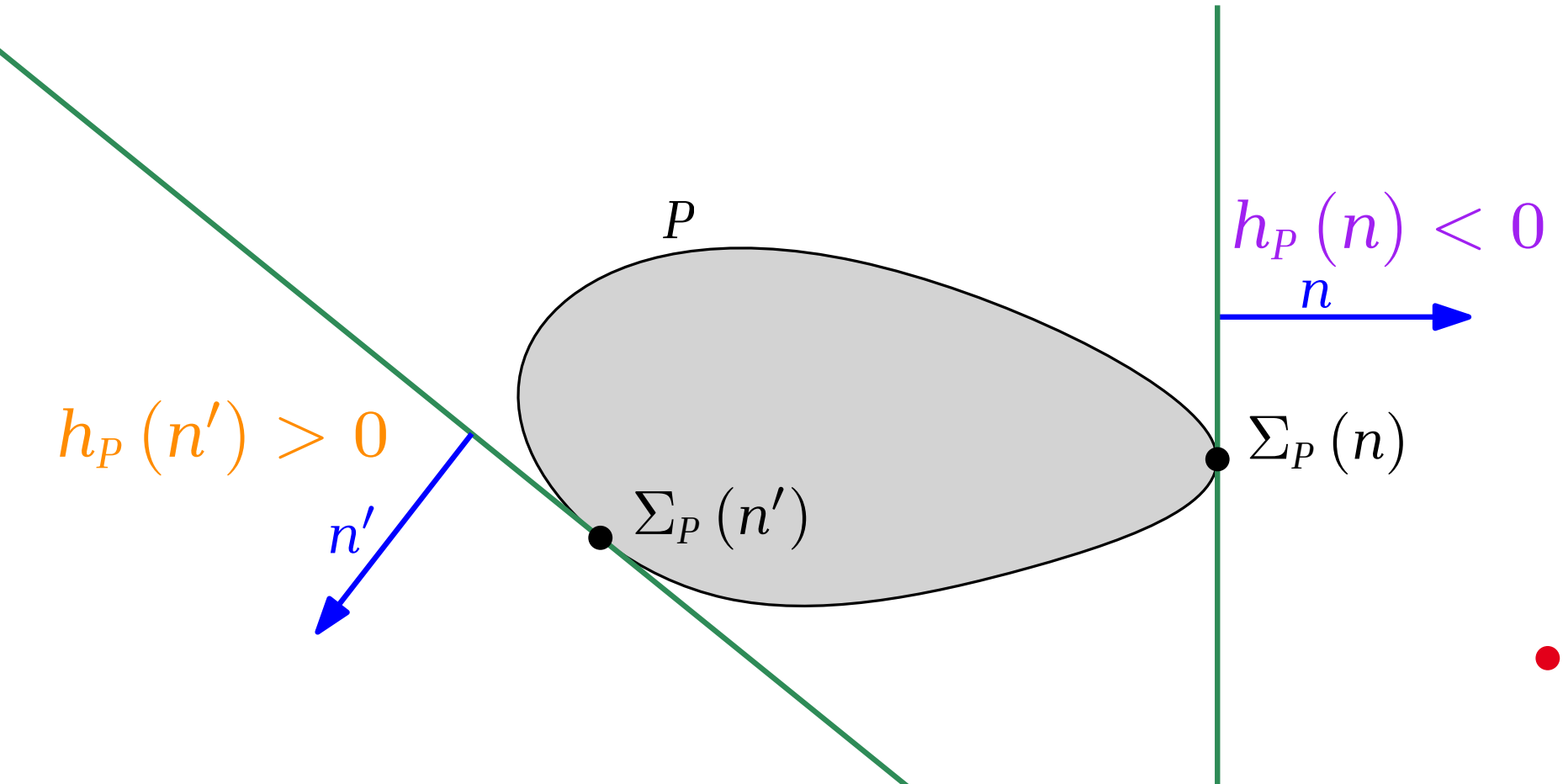


Tools

$\mathbb{S}^2$ the unit sphere; n a direction in $\mathbb{S}^2$; P a convex shape.

The support function $h_P : \mathbb{S}^2 \to \mathbb{R}, \quad n \mapsto \max_{x \in P} (x \cdot n)$

The extremal function $\Sigma_P : \mathbb{S}^2 \to \mathbb{R}^3, \quad n \mapsto \arg\max_{x \in P} (x \cdot n)$



Tools

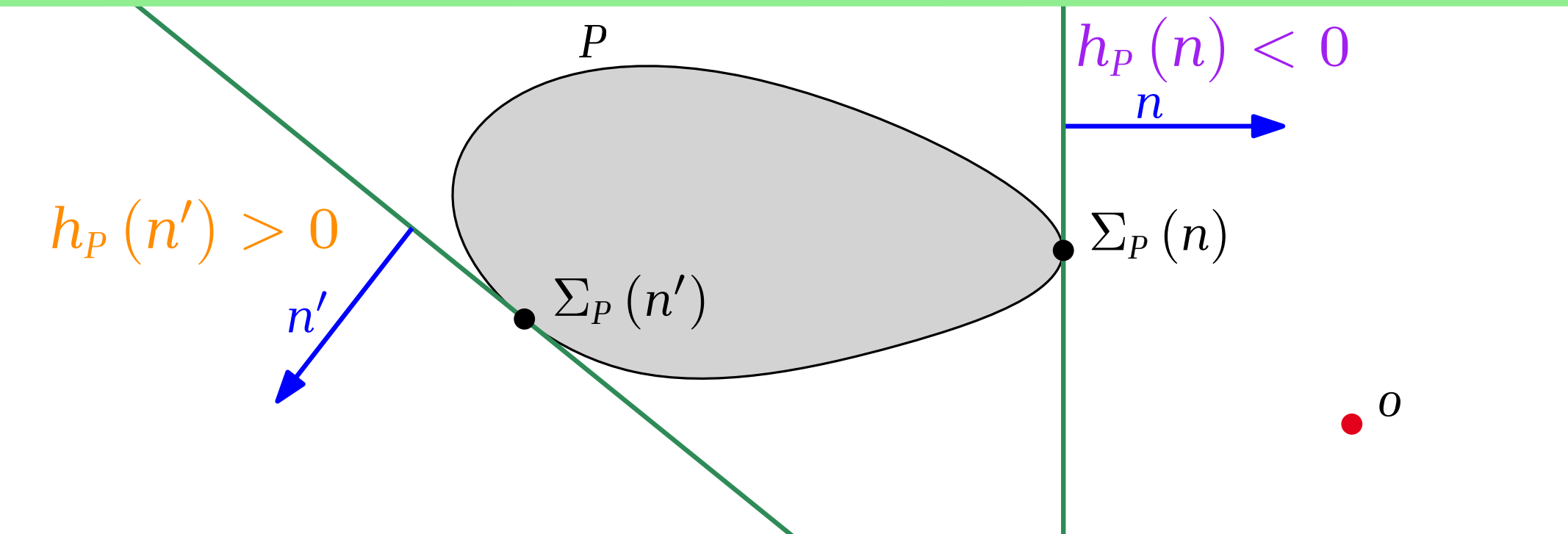
$\mathbb{S}^2$ the unit sphere; n a direction in $\mathbb{S}^2$; P a convex shape.

The support function $h_P : \mathbb{S}^2 \to \mathbb{R}, \quad n \mapsto \max_{x \in P} (x \cdot n)$

The extremal function $\Sigma_P : \mathbb{S}^2 \to \mathbb{R}^3, \quad n \mapsto \arg\max_{x \in P} (x \cdot n)$

$$\Sigma_{A \ominus B}(n) = \Sigma_A(n) - \Sigma_B(-n).$$

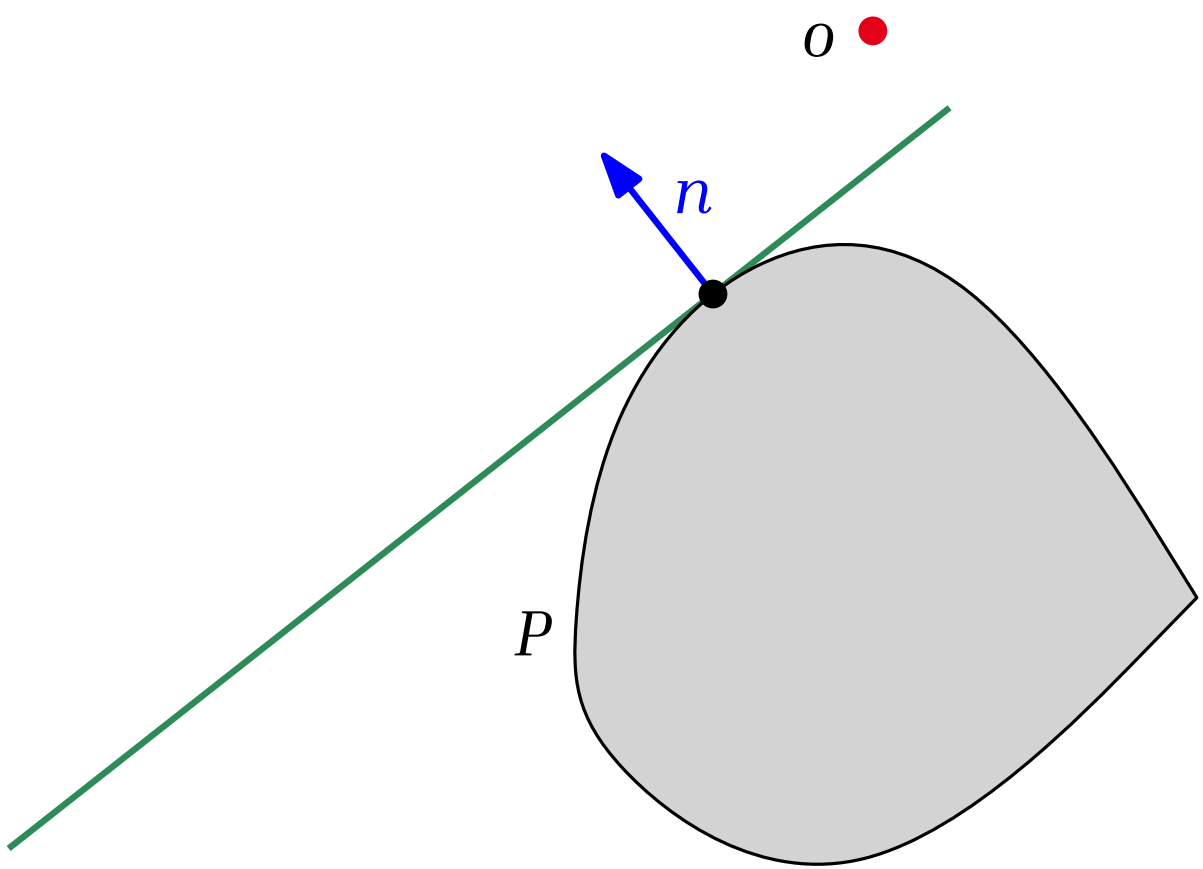
Computable in $O(|A| + |B|)$ time (without building $A \ominus B$!)



DSS idea

$o \notin P \quad \Leftrightarrow \quad \exists$ oriented tangent plane separating o and P

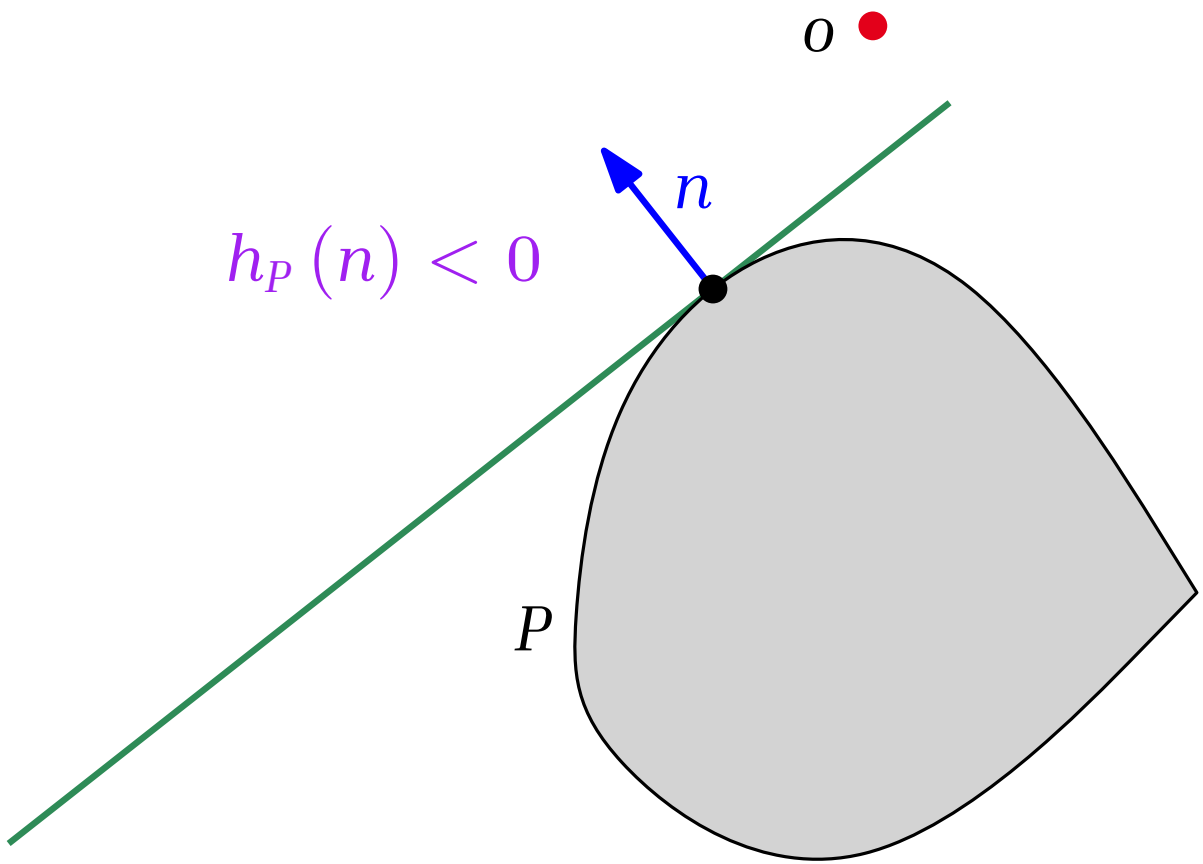
$\Leftrightarrow \quad \exists n \in \mathbb{S}^2, h_P(n) < 0$



DSS idea

$o \notin P \quad \Leftrightarrow \quad \exists$ oriented tangent plane separating o and P

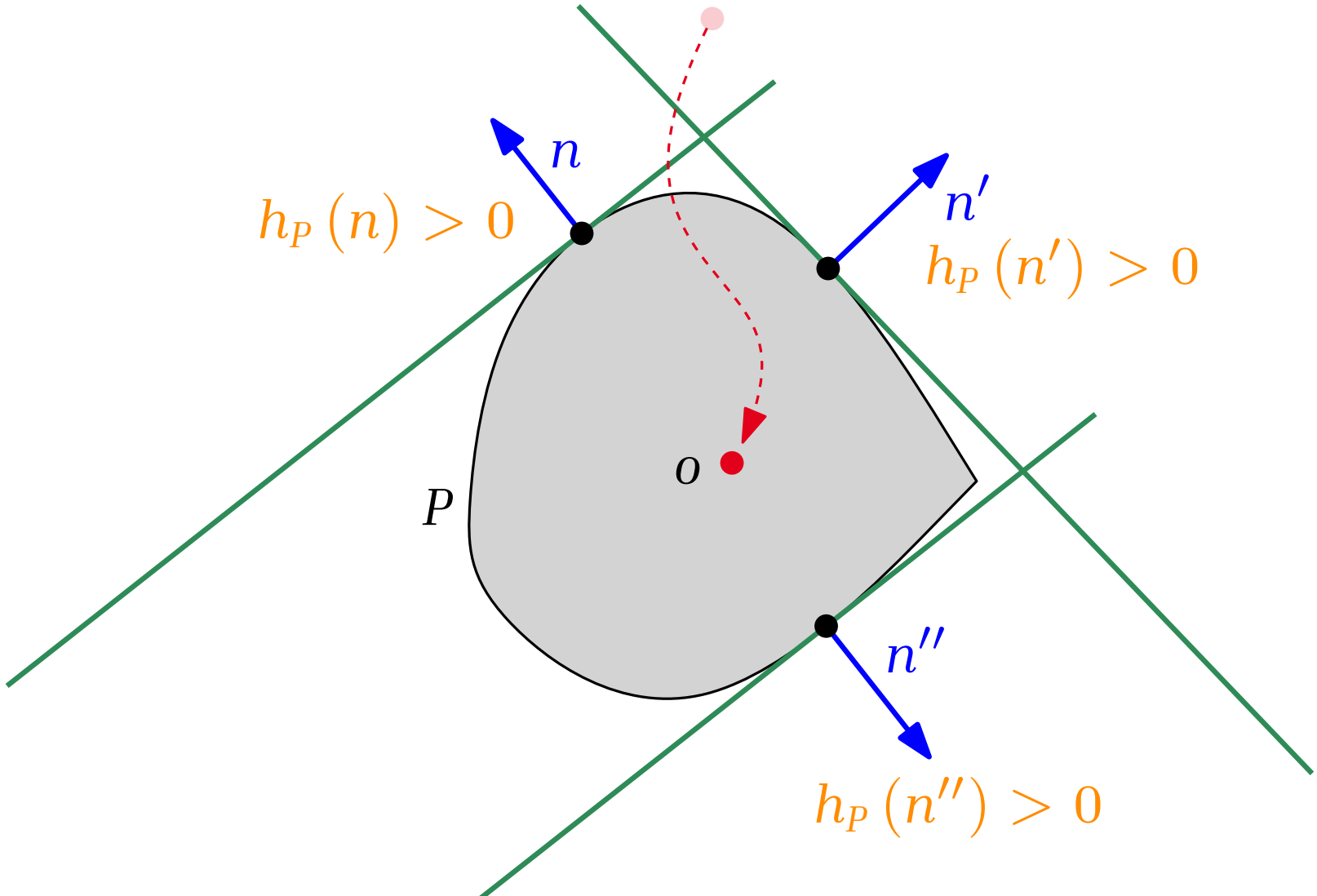
$\Leftrightarrow \quad \exists n \in \mathbb{S}^2, h_P(n) < 0$



DSS idea

$o \notin P \quad \Leftrightarrow \quad \exists$ oriented tangent plane separating o and P

$\Leftrightarrow \quad \exists n \in \mathbb{S}^2, h_P(n) < 0$

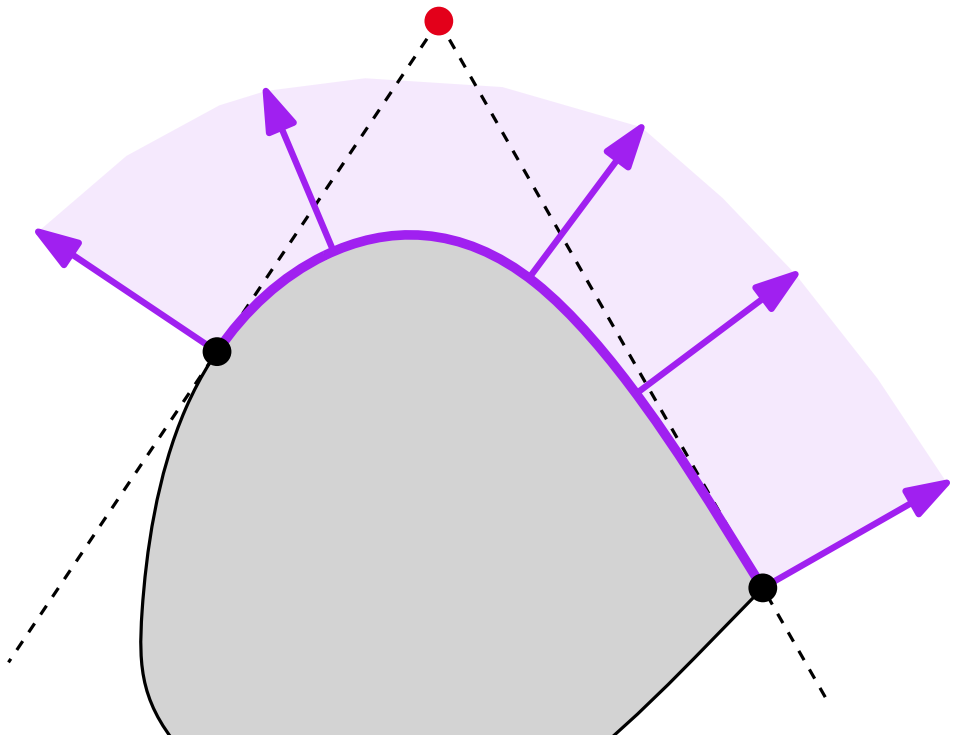


DSS idea

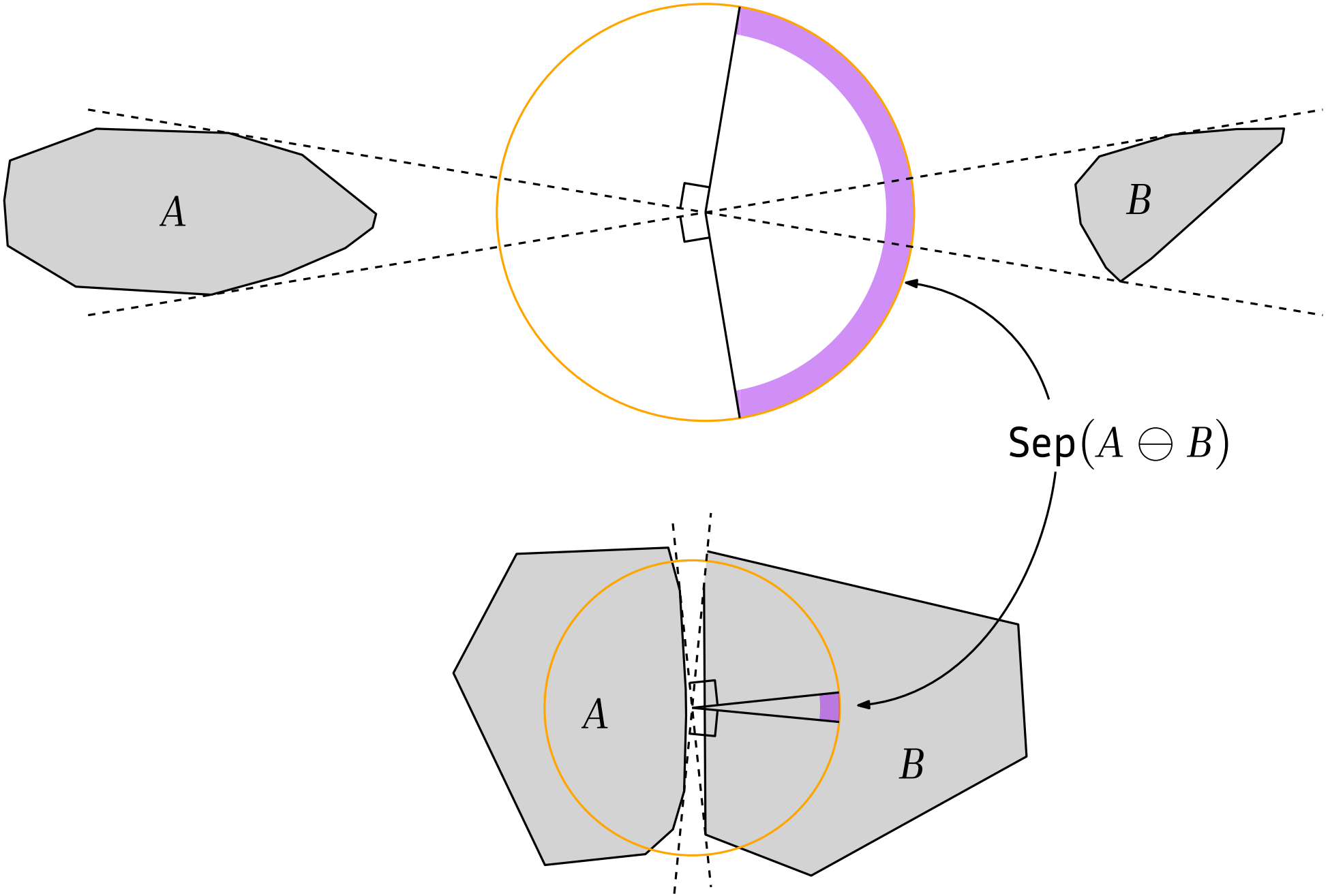
The separating set $\mathrm{Sep}(P) = \{n \in \mathbb{S}^2 \mid h_P(n) < 0\}$:

$$o \notin P \quad \Leftrightarrow \quad \mathrm{Sep}(P) \neq \emptyset$$

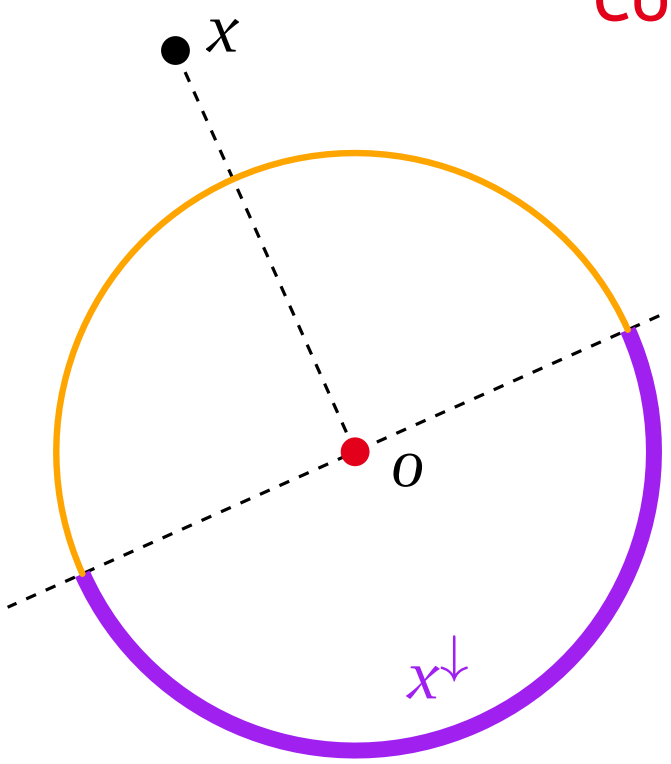
Idea: find a direction $n \in \mathrm{Sep}(P)$
or decide that $\mathrm{Sep}(P)$ is empty.



Examples of separating sets when $P = A \ominus B$

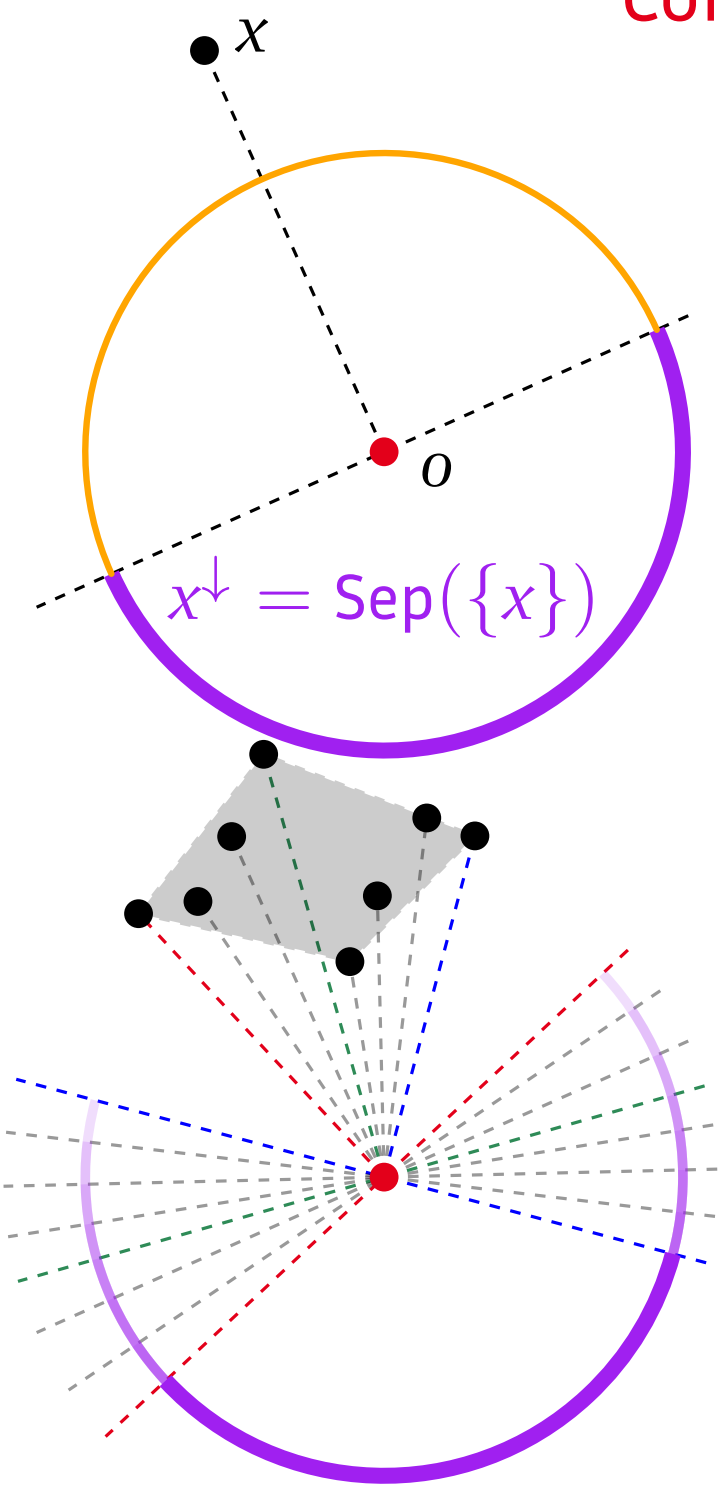


Computing separating sets



Define $x^{\downarrow} = \{n \in \mathbb{S}^2 \mid n \cdot x < 0\}$

Computing separating sets

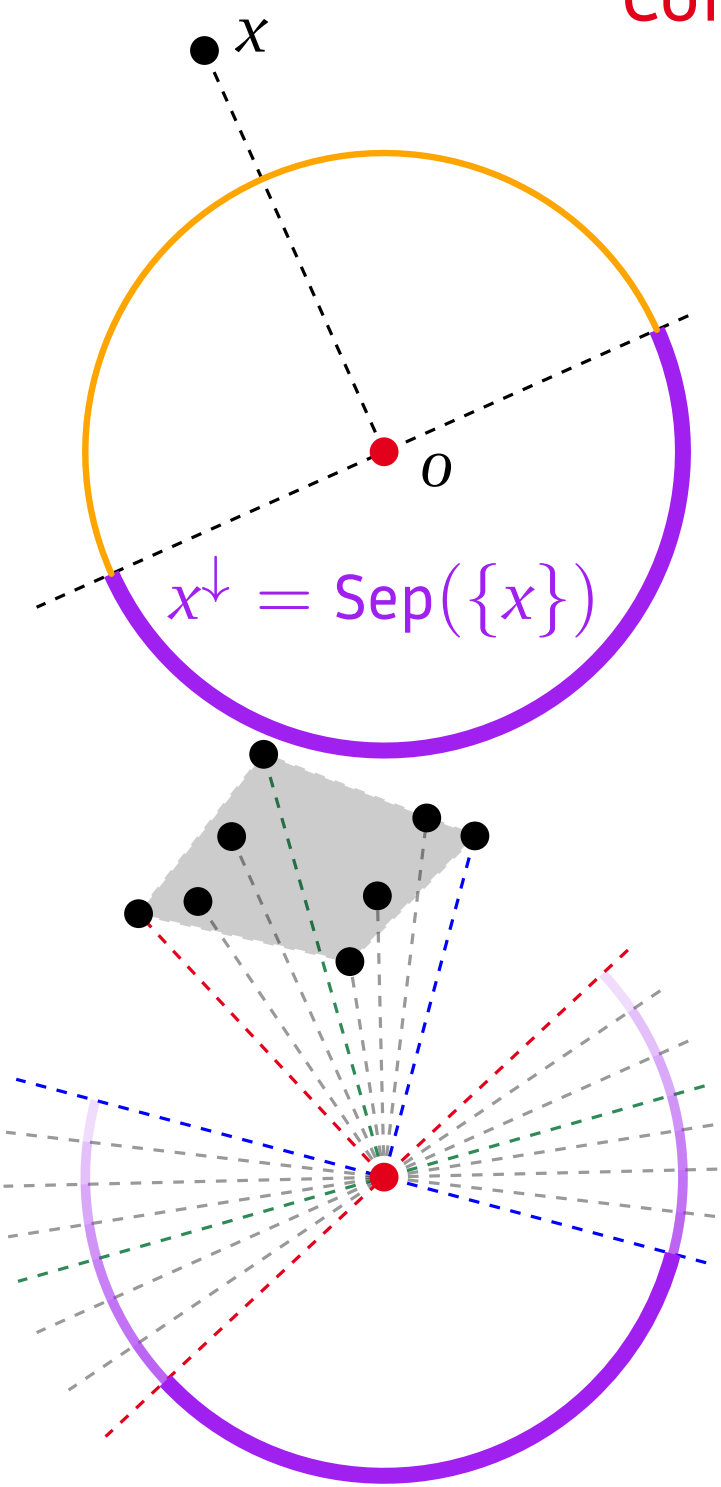


Define $x^{\downarrow} = \{n \in \mathbb{S}^2 \mid n \cdot x < 0\}$

Observations:

1. $\mathsf{Sep}(\{x\}) = x^{\downarrow}$
2. $\mathsf{Sep}(X \cup Y) = \mathsf{Sep}(X) \cap \mathsf{Sep}(Y)$
3. $\mathsf{Sep}(\mathcal{H}(X)) = \mathsf{Sep}(X)$
4. $X \subset Y \Rightarrow \mathsf{Sep}(Y) \subset \mathsf{Sep}(X)$

Computing separating sets



Define $x^{\downarrow} = \{n \in \mathbb{S}^2 \mid n \cdot x < 0\}$

Observations:

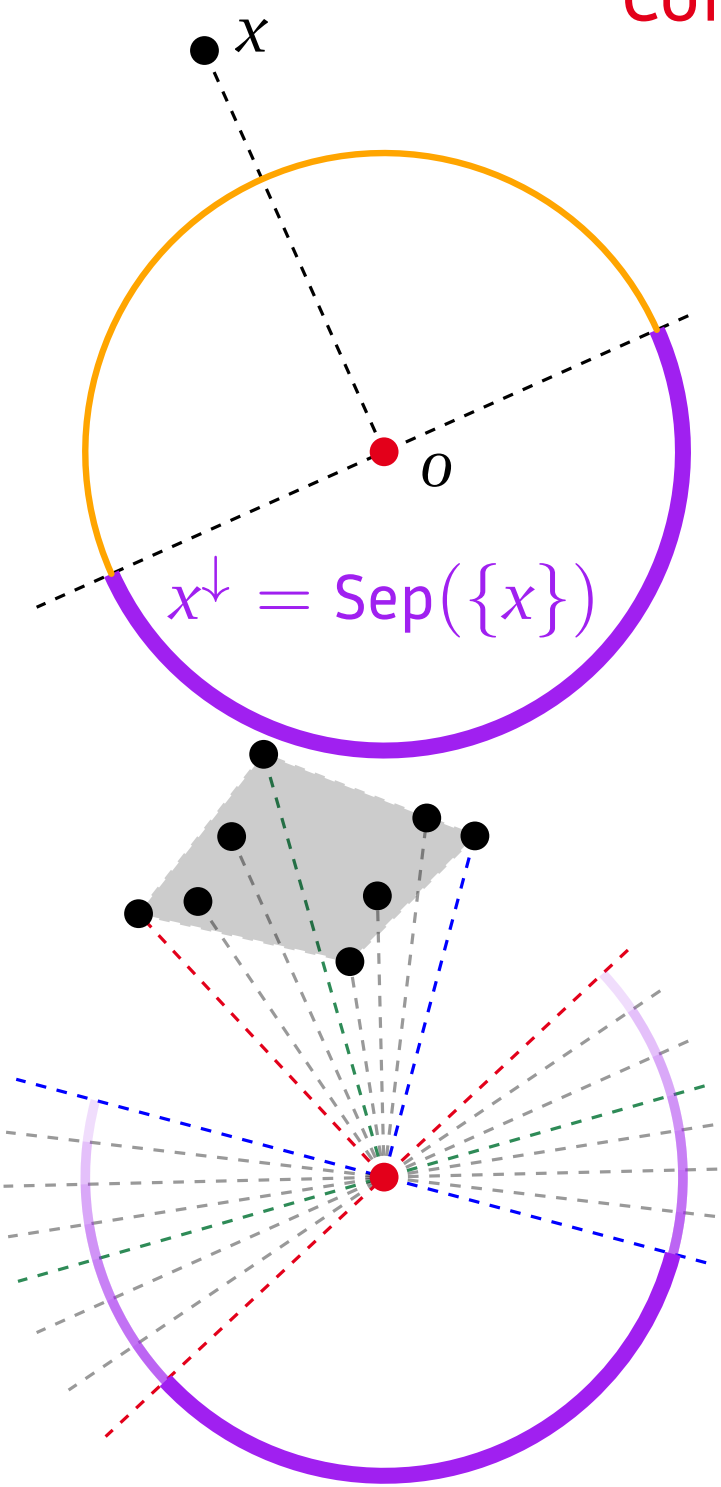
1. $\mathsf{Sep}(\{x\}) = x^{\downarrow}$
2. $\mathsf{Sep}(X \cup Y) = \mathsf{Sep}(X) \cap \mathsf{Sep}(Y)$
3. $\mathsf{Sep}(\mathcal{H}(X)) = \mathsf{Sep}(X)$
4. $X \subset Y \Rightarrow \mathsf{Sep}(Y) \subset \mathsf{Sep}(X)$

Lemma. $\mathsf{Sep}(P) = \bigcap_{x \in P} x^{\downarrow}$.

If $P = \mathcal{H}\left(V_k = \{v_0, v_1, \ldots, v_k\}\right)$, then

$$\mathsf{Sep}(\mathcal{H}(V_k)) = \bigcap_{i=0}^{k} v_i^{\downarrow}$$

Computing separating sets



Define $x^{\downarrow} = \{n \in \mathbb{S}^2 \mid n \cdot x < 0\}$

Observations:

1. $\mathrm{Sep}(\{x\}) = x^{\downarrow}$
2. $\mathrm{Sep}(X \cup Y) = \mathrm{Sep}(X) \cap \mathrm{Sep}(Y)$
3. $\mathrm{Sep}(\mathcal{H}(X)) = \mathrm{Sep}(X)$
4. $X \subset Y \Rightarrow \mathrm{Sep}(Y) \subset \mathrm{Sep}(X)$

Lemma. $\mathrm{Sep}(P) = \bigcap_{x \in P} x^{\downarrow}$.

If $P = \mathcal{H}\left(V_k = \{v_0, v_1, \ldots, v_k\}\right)$, then

$$\mathrm{Sep}(\mathcal{H}(V_k)) = \bigcap_{i=0}^{k} v_i^{\downarrow}$$

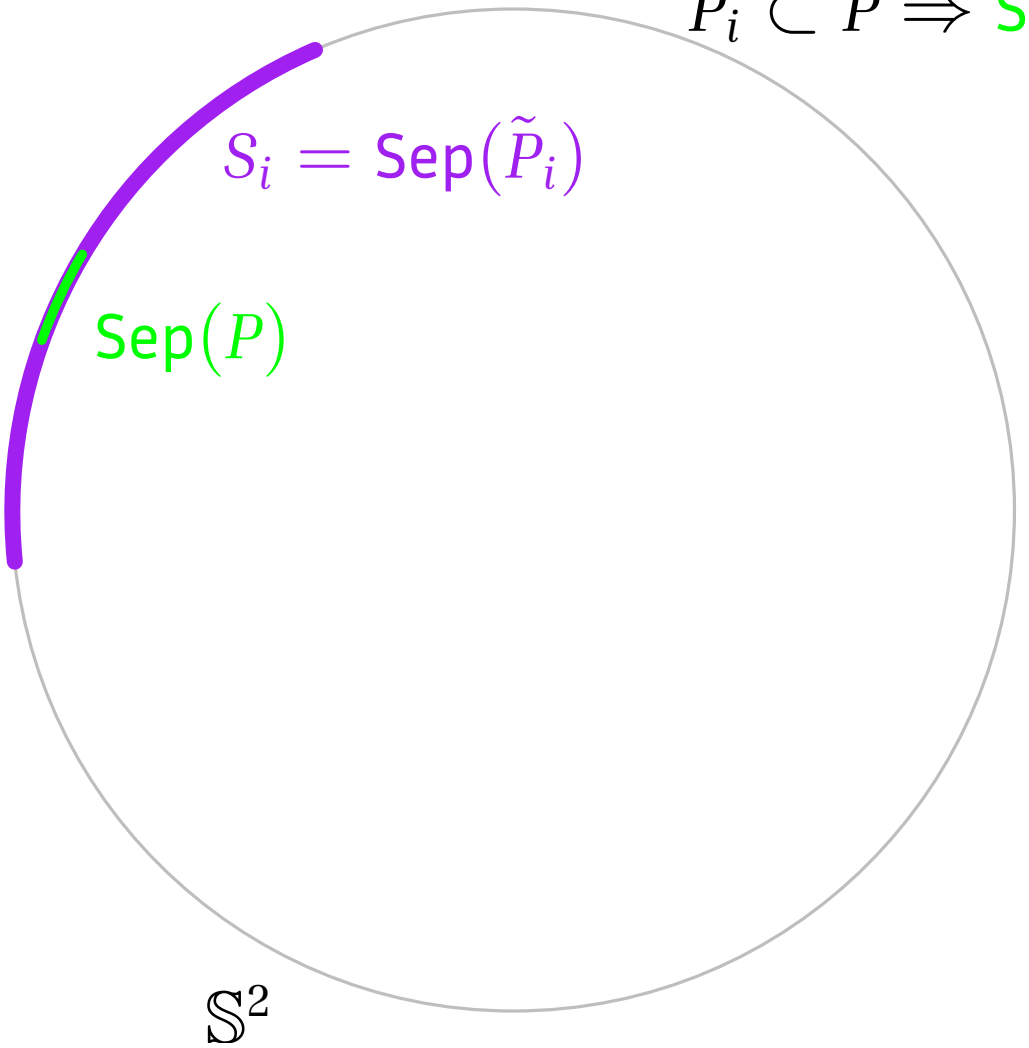
A convex spherical polygon

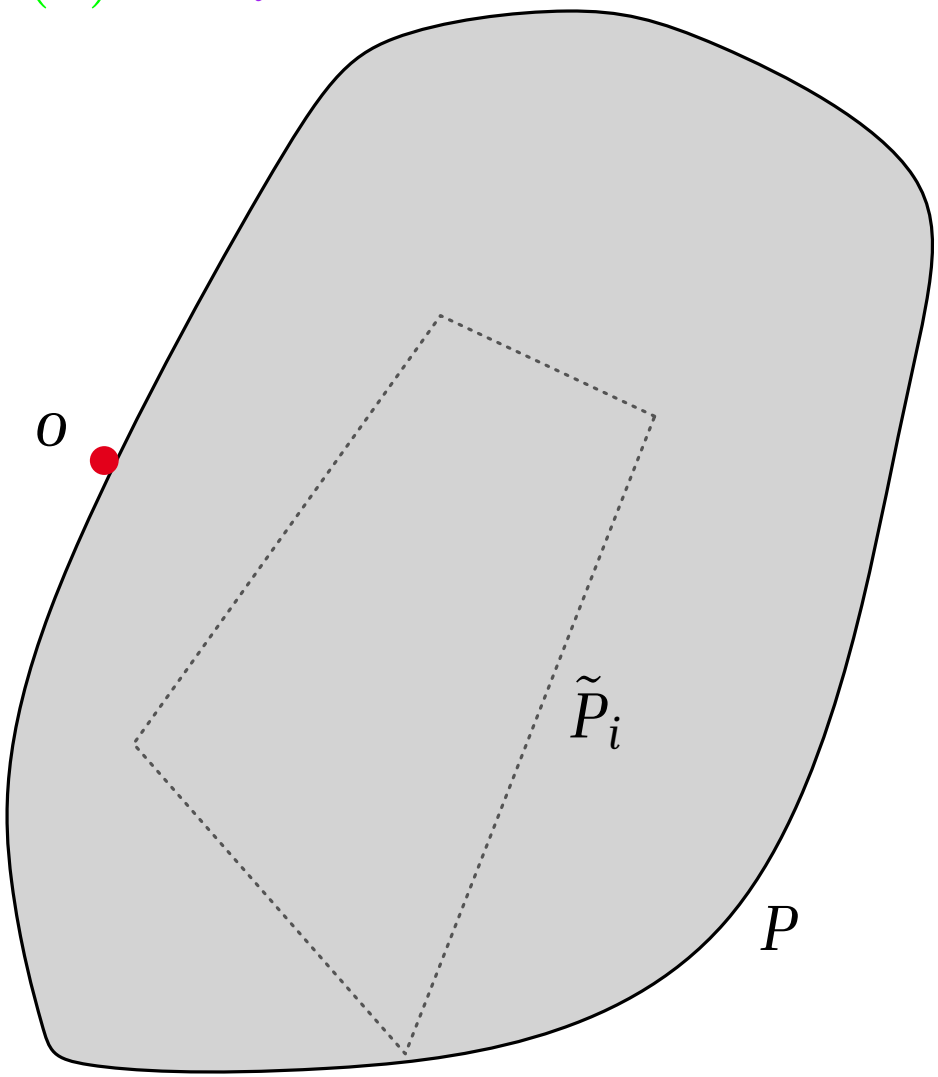
DSS when $o \notin P$

Maintain a search polygon S_i on $\mathbb{S}^2$ as a superset of $\mathsf{Sep}(P)$:

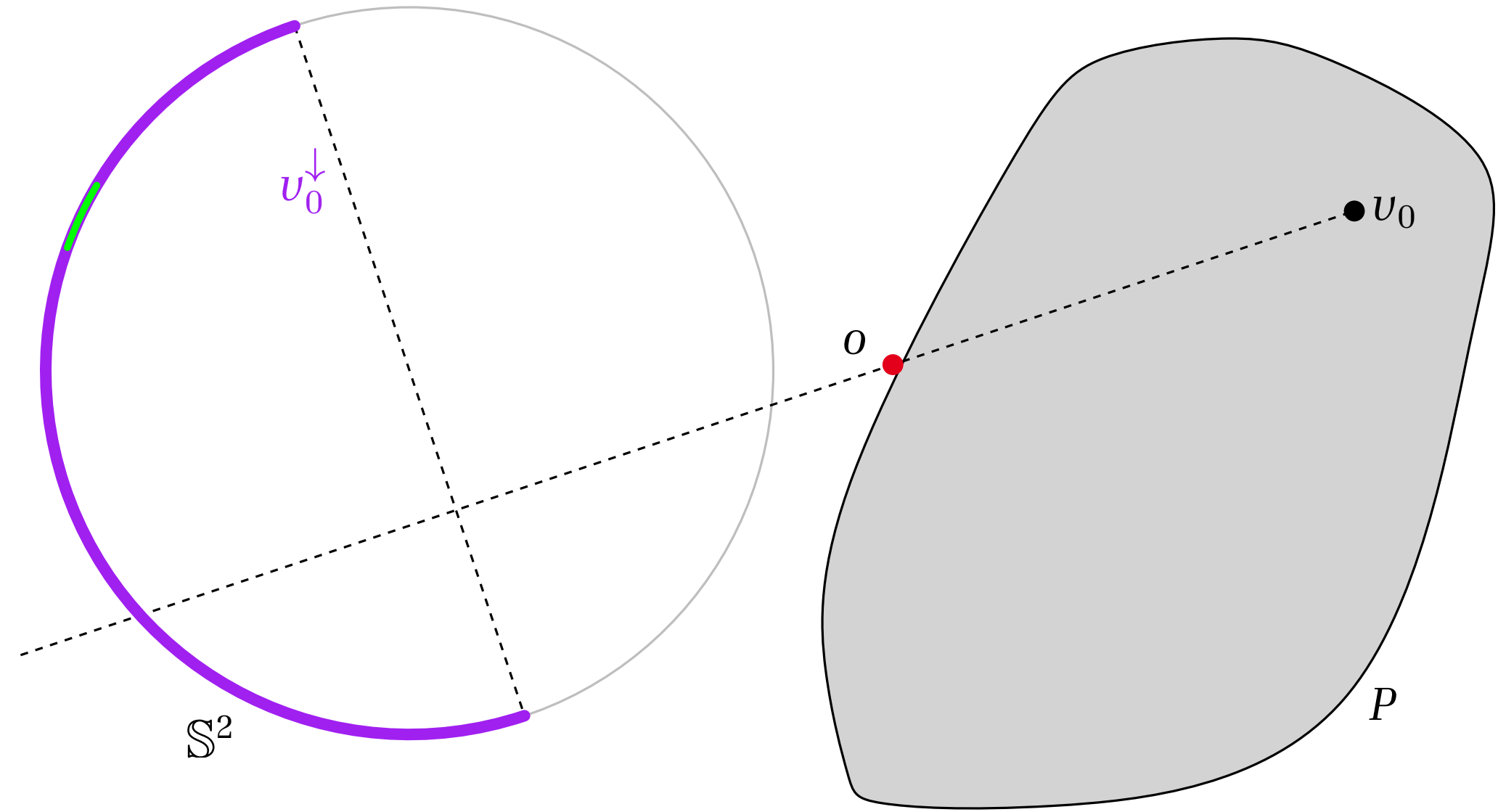
$$S_i = \mathsf{Sep}\left(\tilde{P}_i = \mathcal{H}(\{v_0, v_1, \ldots, v_i\})\right) = \bigcap_{k=0}^{i} v_k^{\downarrow}$$

$$\tilde{P}_i \subset P \Rightarrow \mathsf{Sep}(P) \subset S_i$$



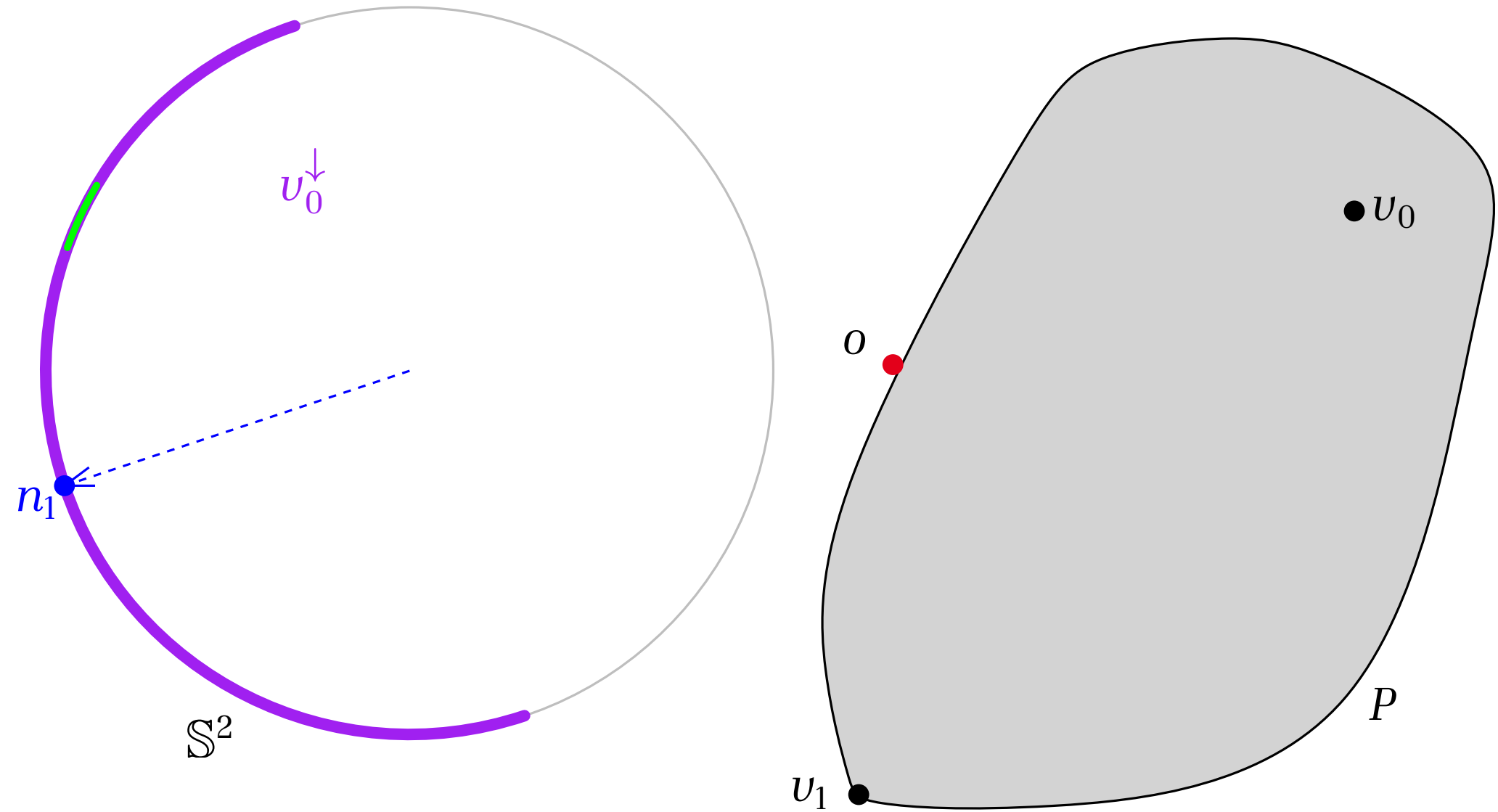


DSS when $o \notin P$



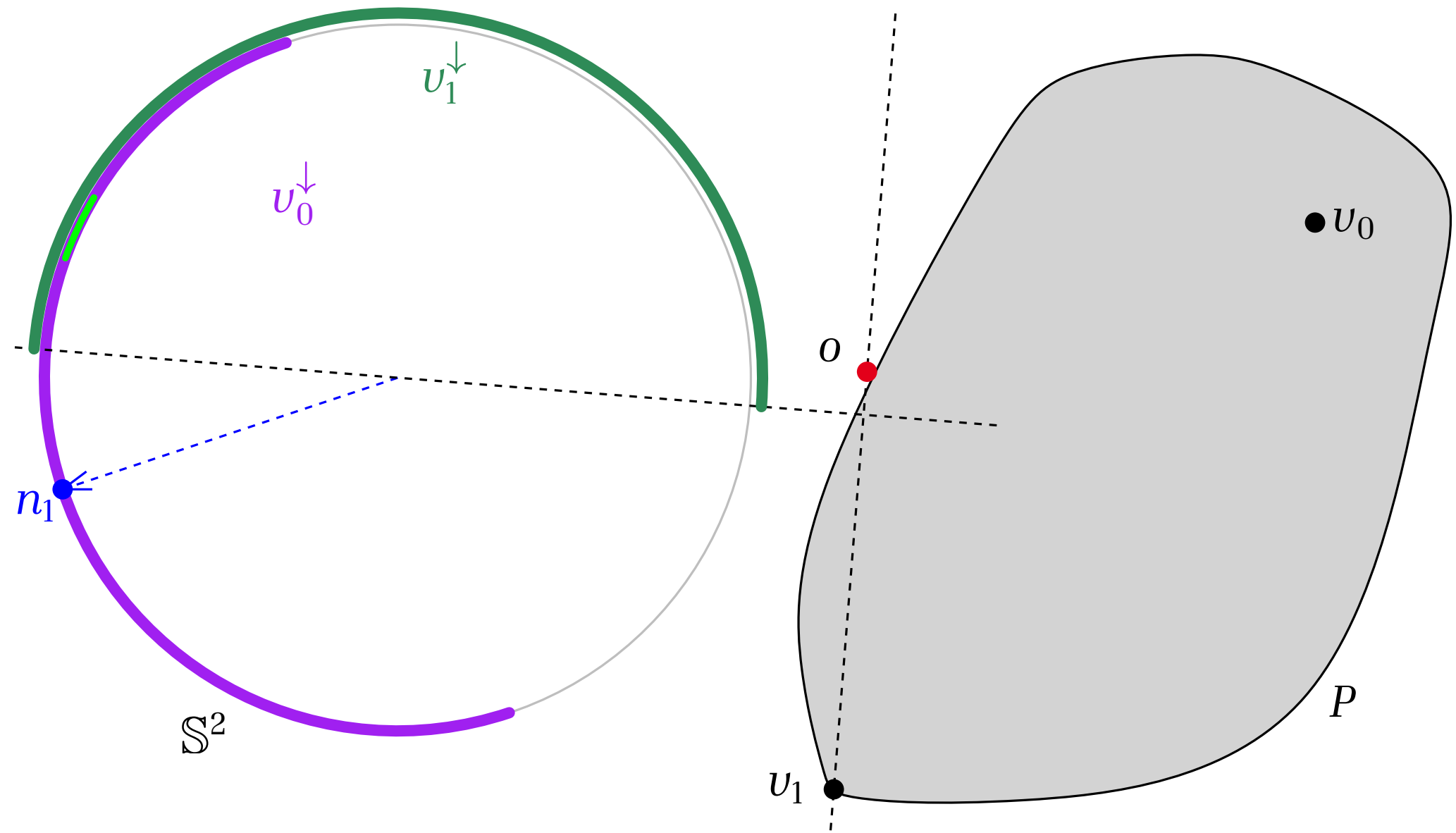
DSS when $o \notin P$

$v_1 \leftarrow \Sigma_P(n_1)$;
$h_P(n_1) \geq 0$



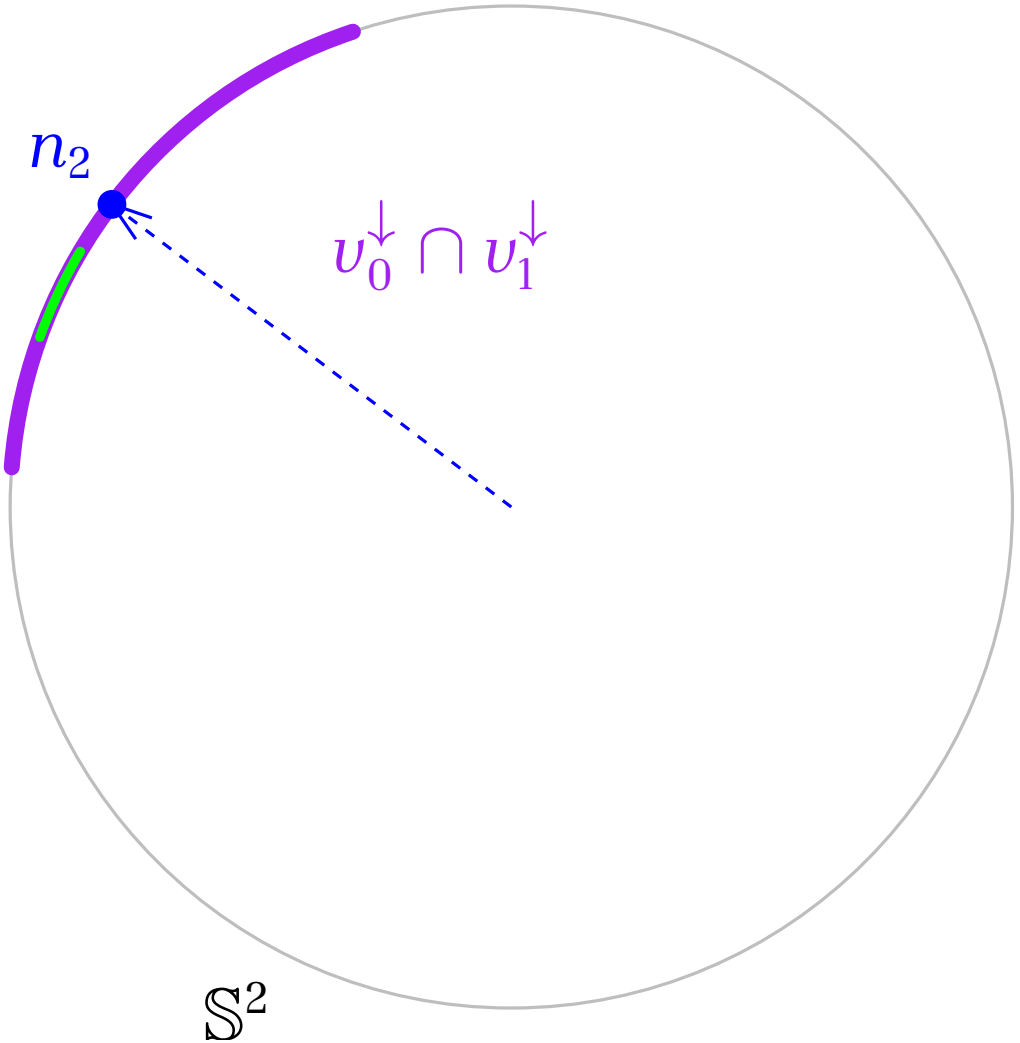
DSS when $o \notin P$

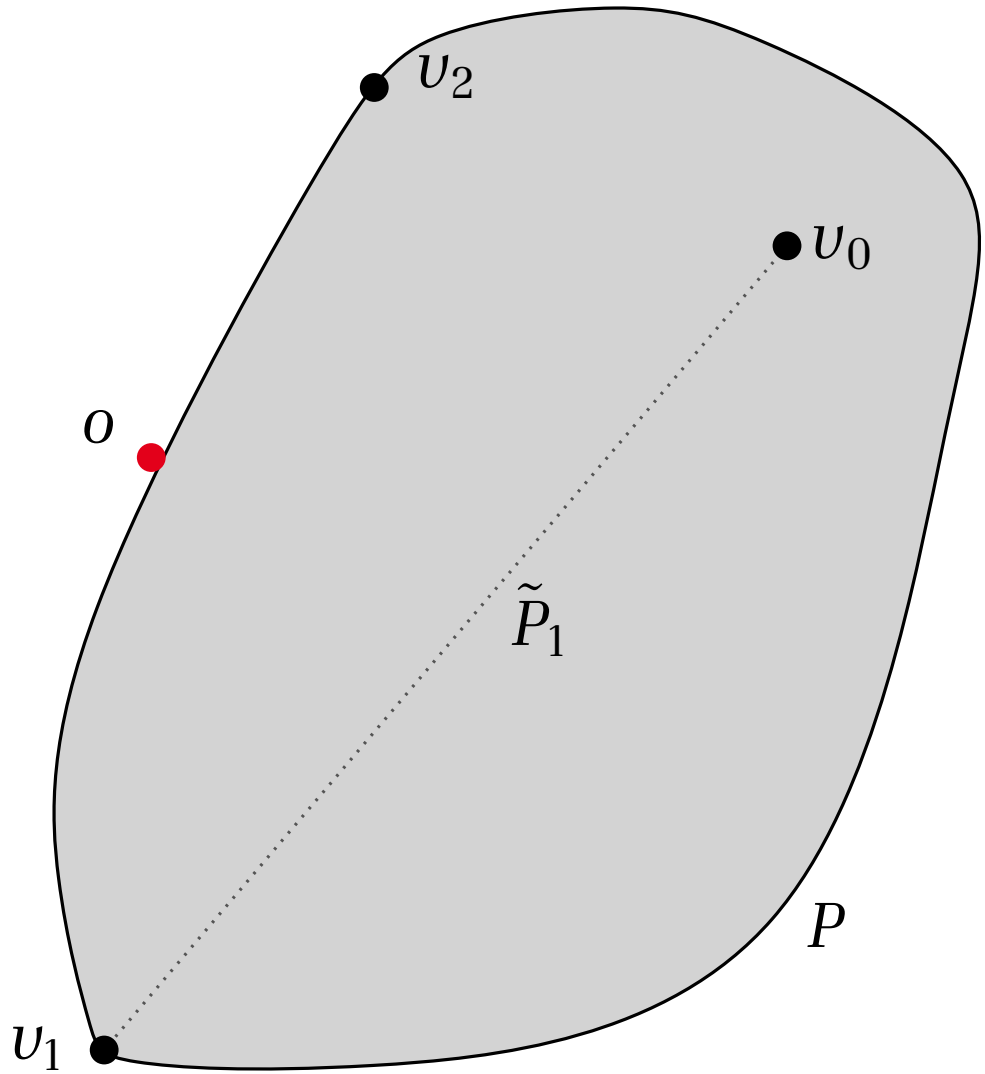
$v_1 \leftarrow \Sigma_P(n_1)$;
$h_P(n_1) \geq 0$



DSS when $o \notin P$

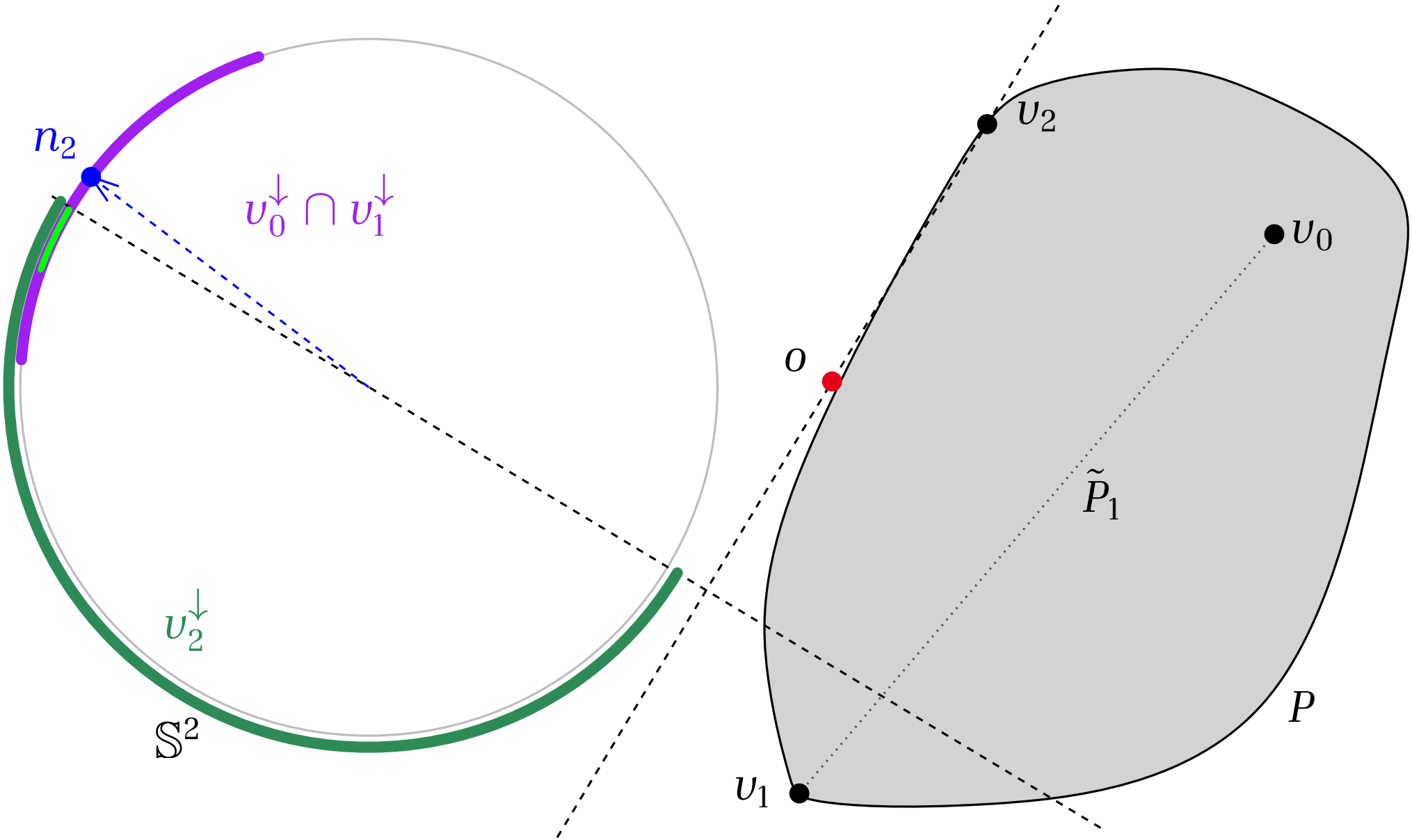
$v_2 \leftarrow \Sigma_P(n_2)$;
$h_P(n_2) \geq 0$





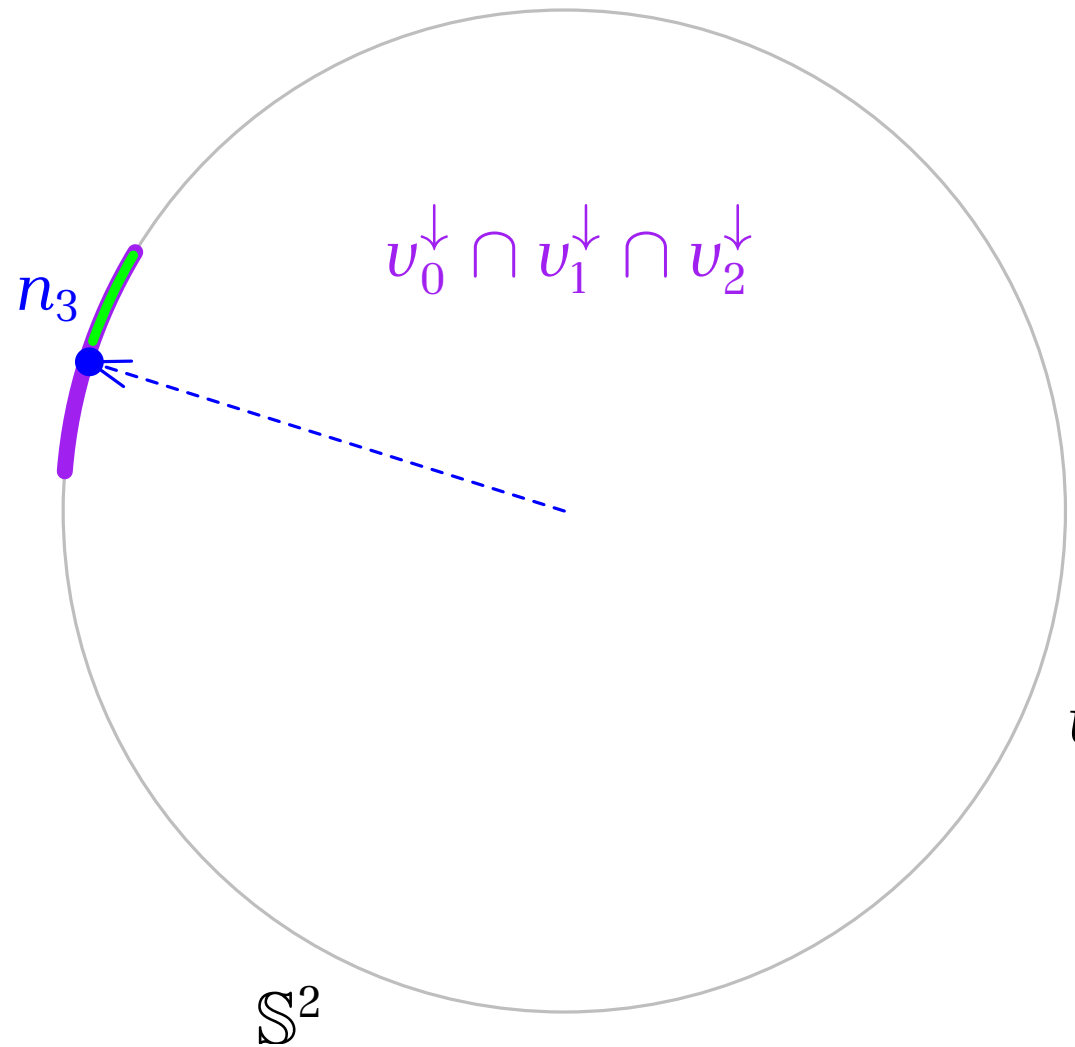
DSS when $o \notin P$

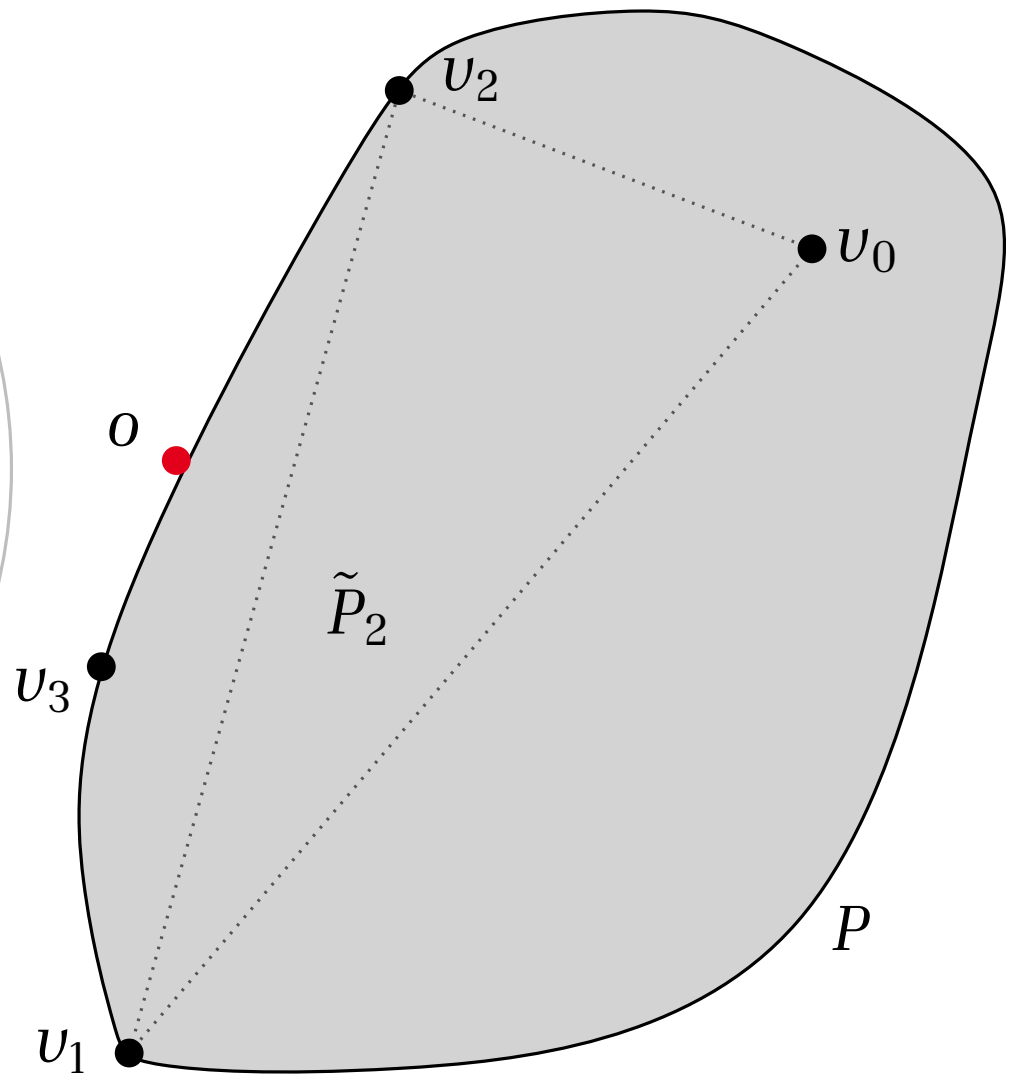
$v_2 \leftarrow \Sigma_P(n_2)$;
$h_P(n_2) \geq 0$



DSS when $o \notin P$

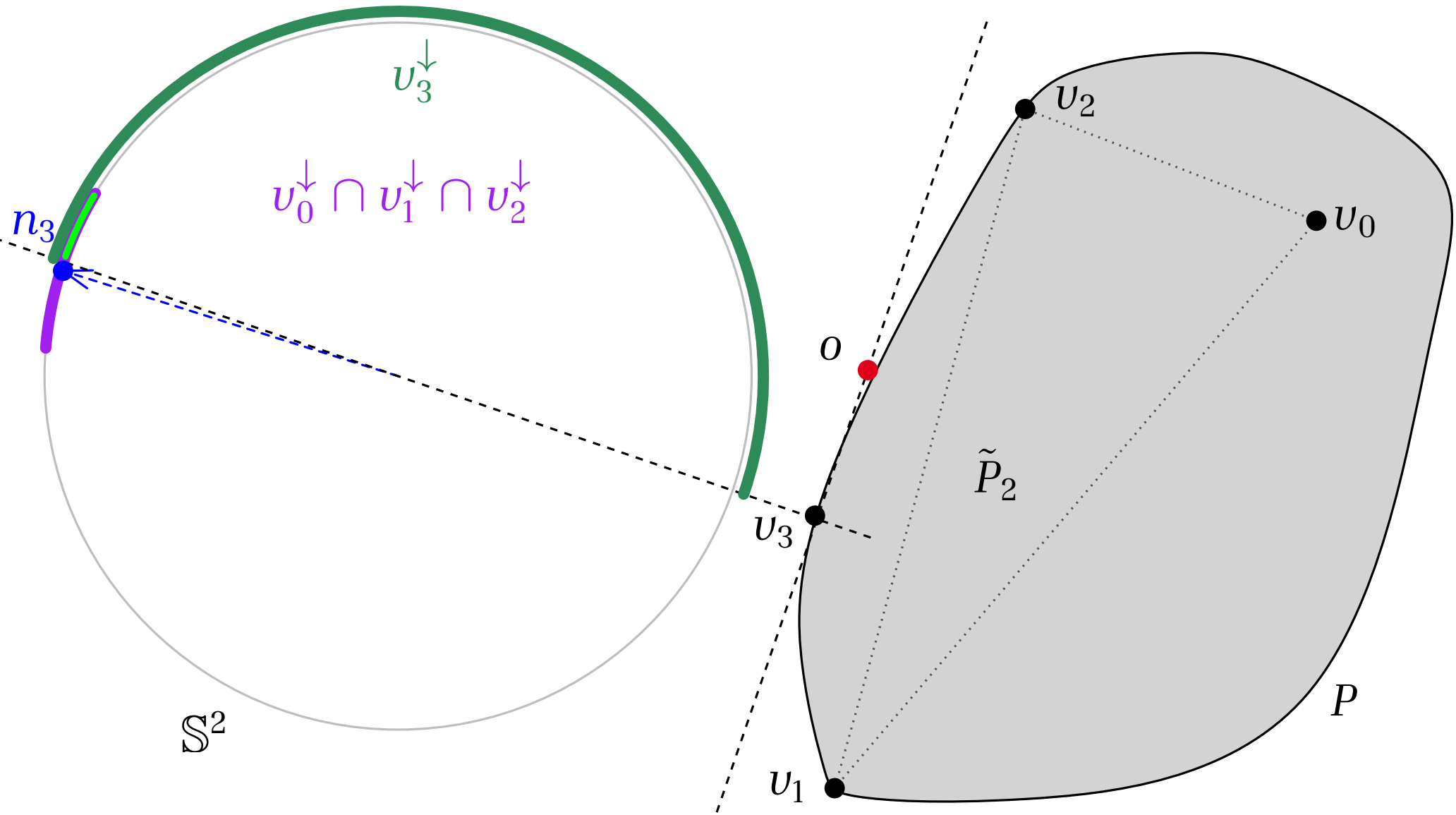
$v_3 \leftarrow \Sigma_P(n_3)$;
$h_P(n_3) \geq 0$





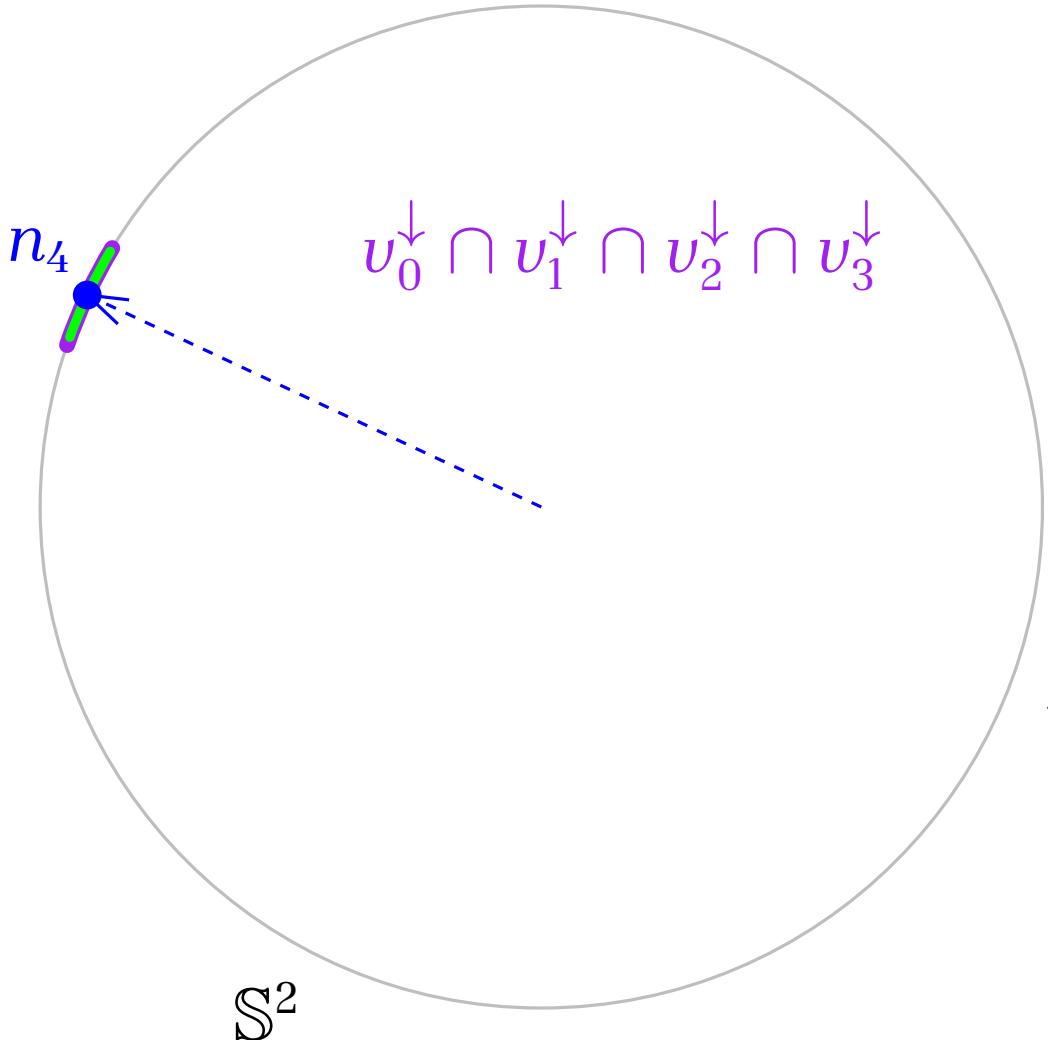
DSS when $o \notin P$

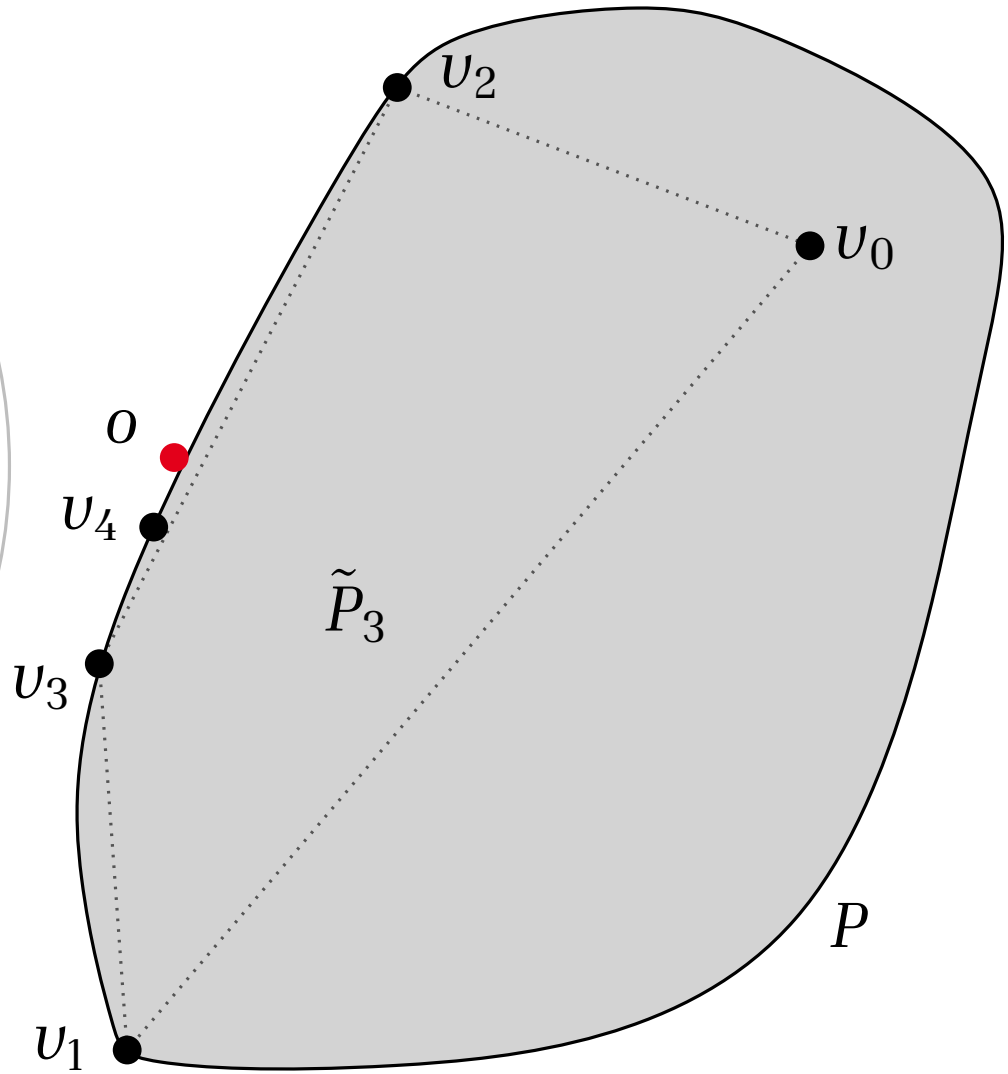
$$v_3 \leftarrow \Sigma_P(n_3);$$
$$h_P(n_3) \geq 0$$



DSS when $o \notin P$

$$v_4 \leftarrow \Sigma_P(n_4);$$
$$h_P(n_4) < 0$$

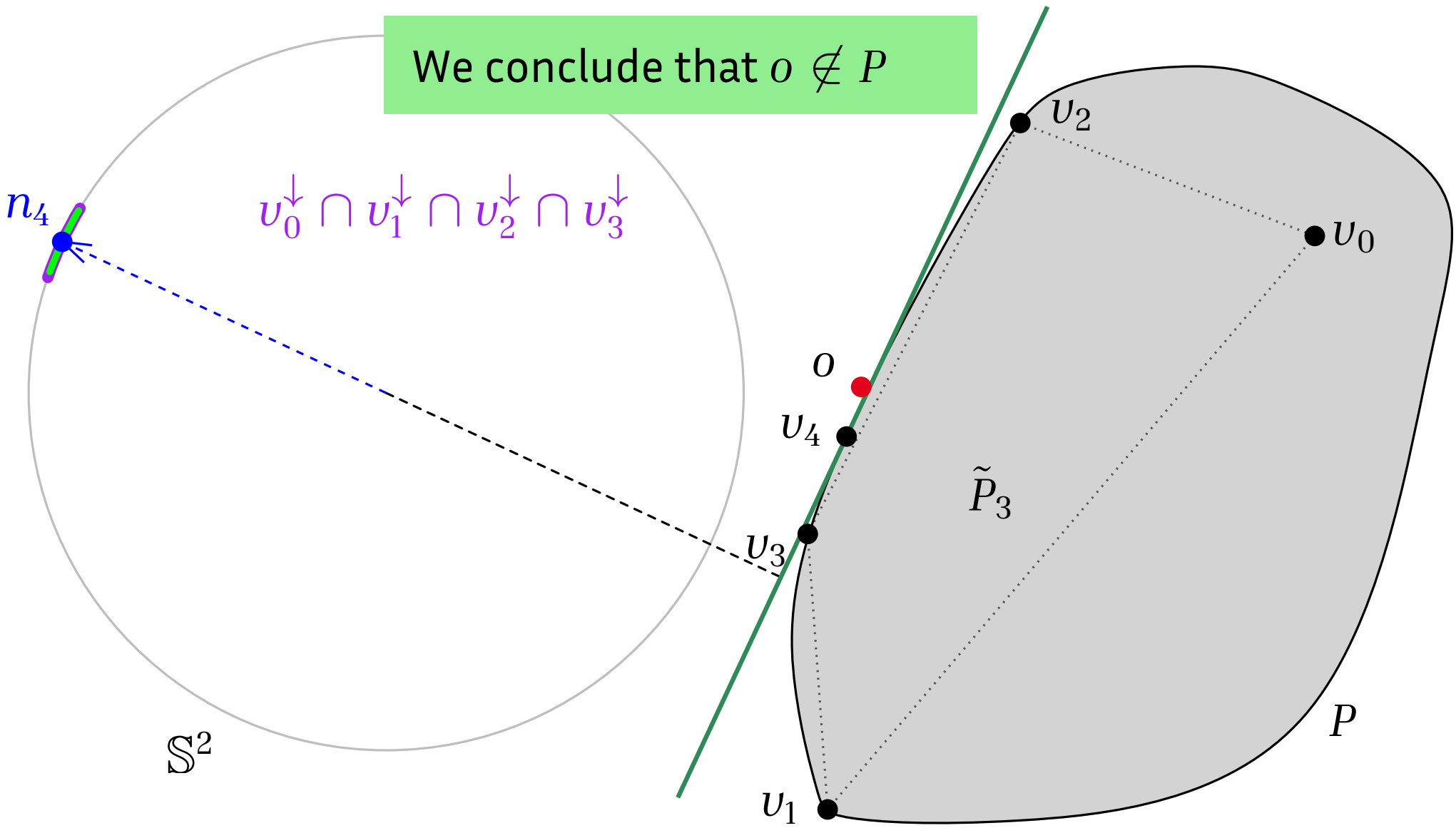




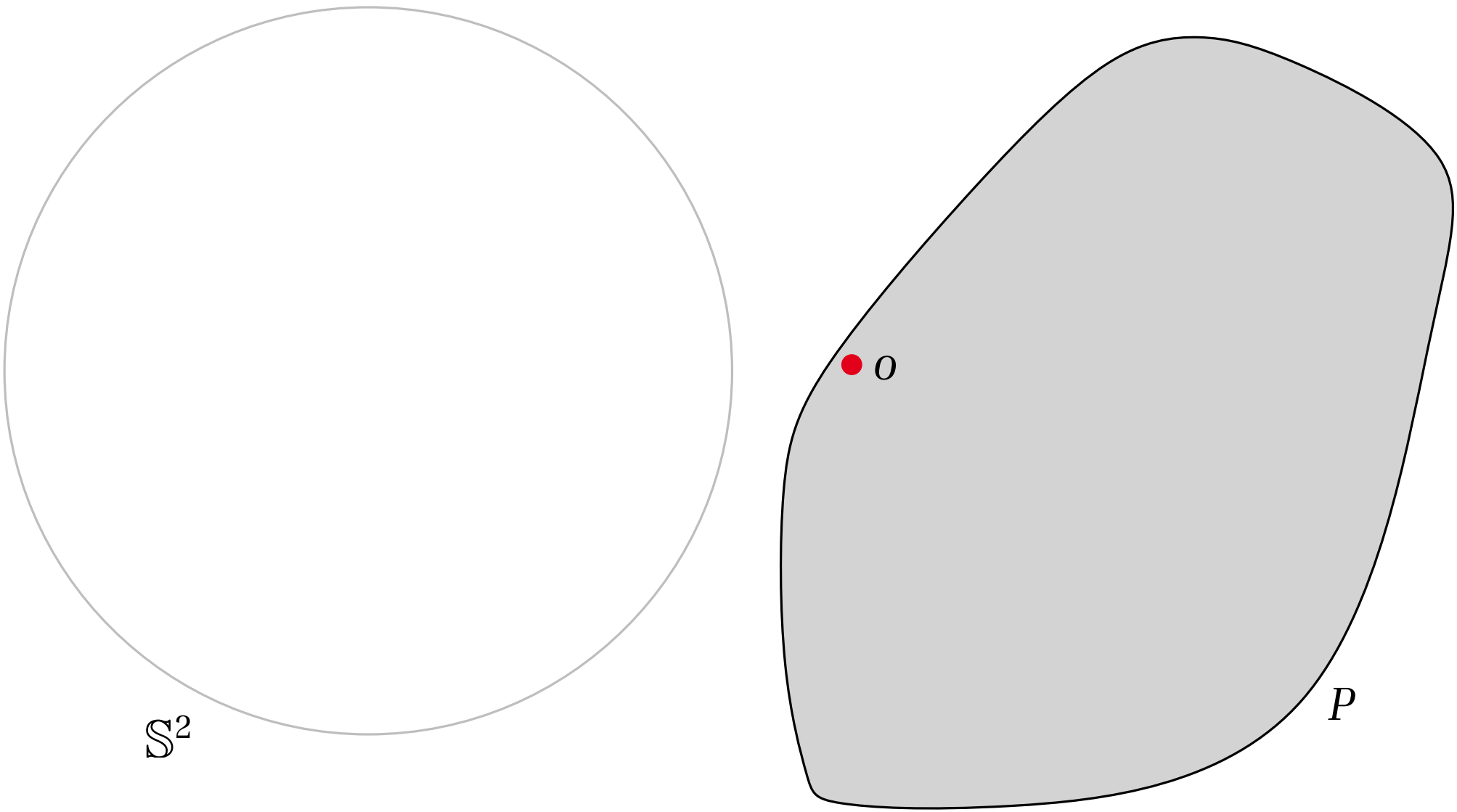
DSS when $o \notin P$

$$v_4 \leftarrow \Sigma_P(n_4);$$
$$h_P(n_4) < 0$$

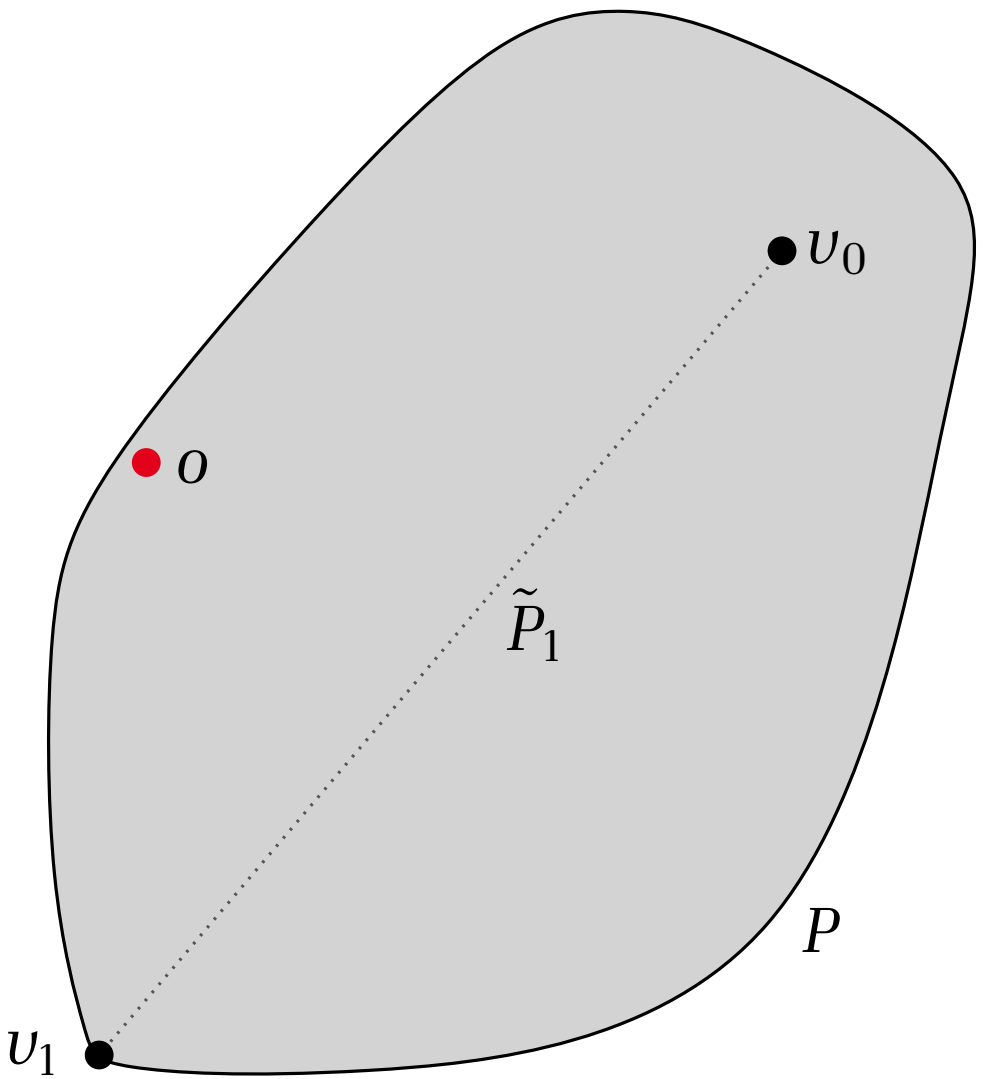
We conclude that $o \notin P$



DSS when $o \in P$

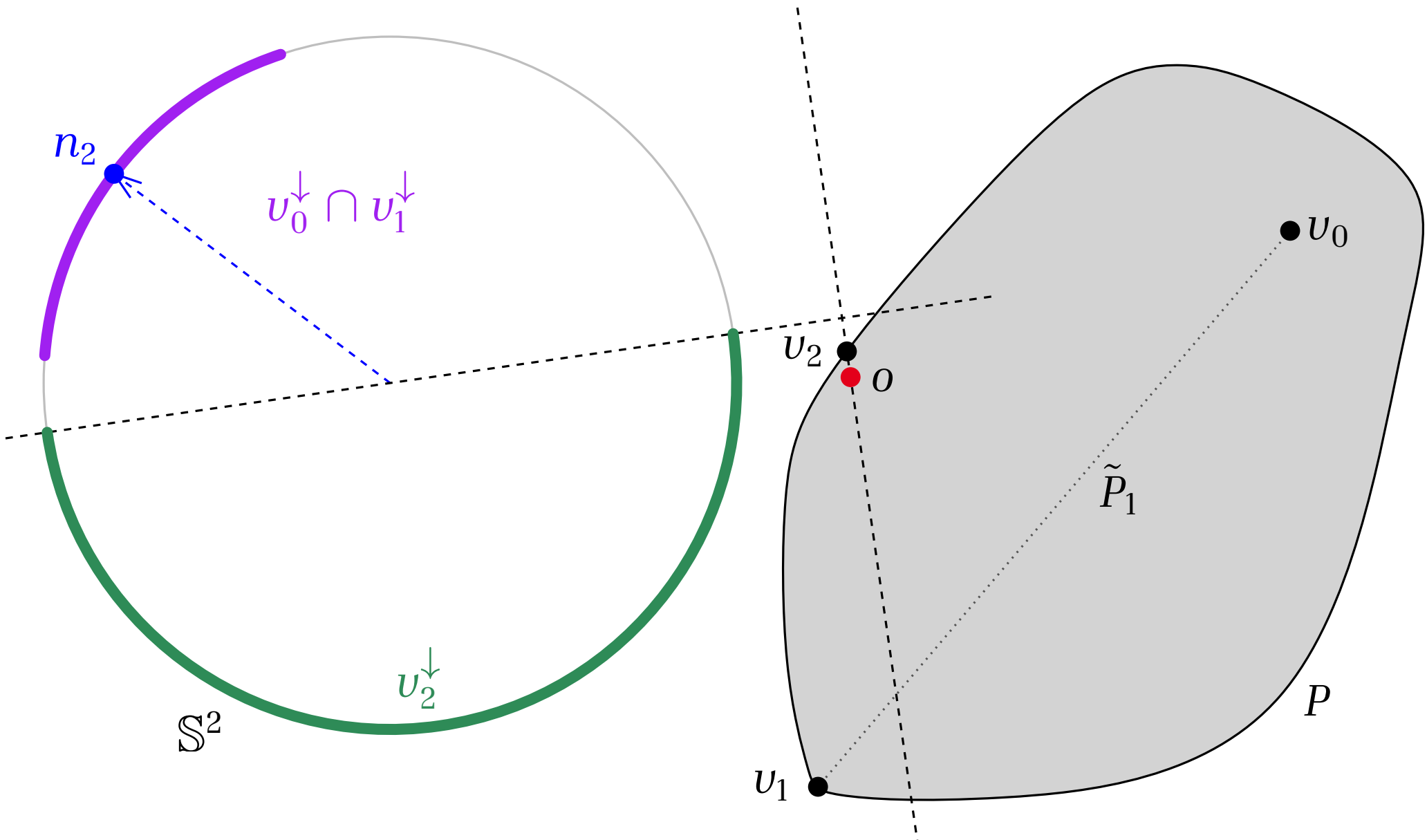


DSS when $o \in P$



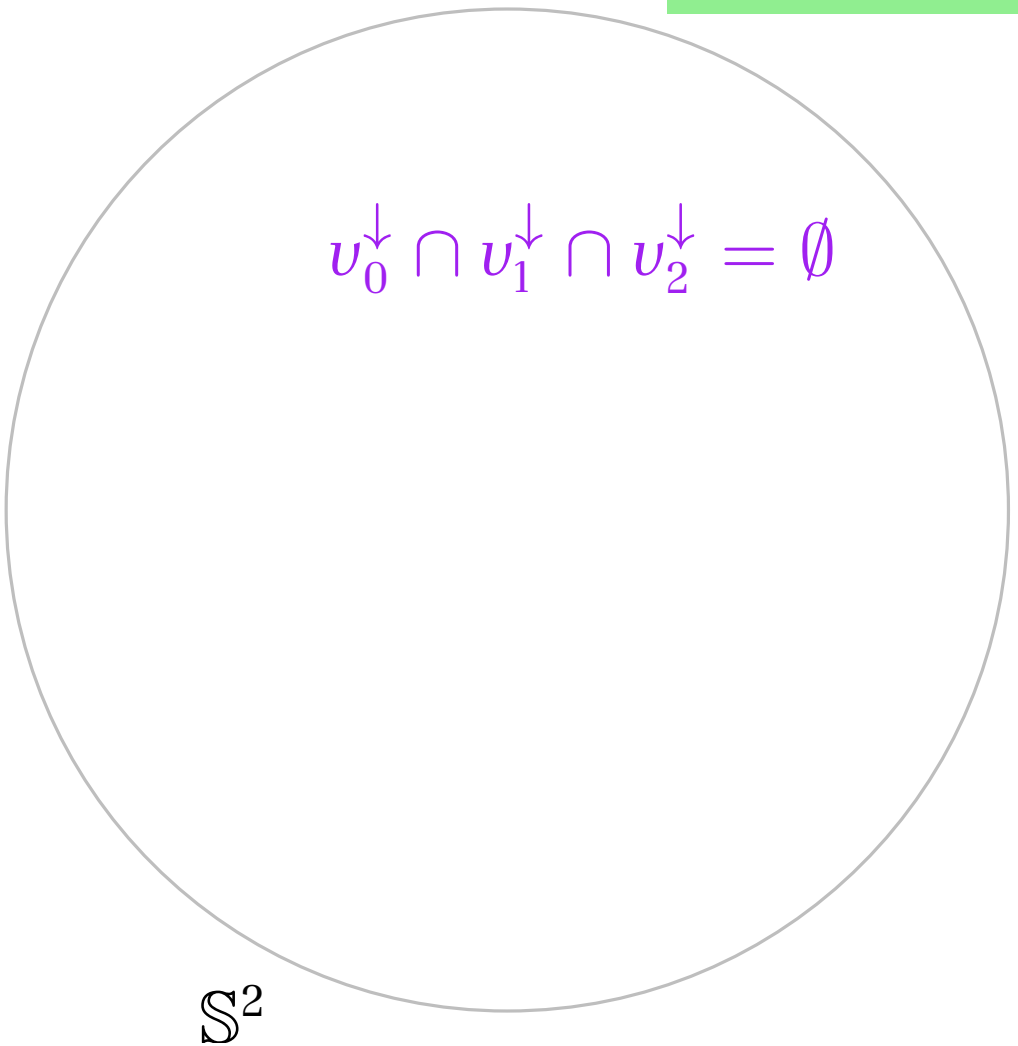
DSS when $o \in P$

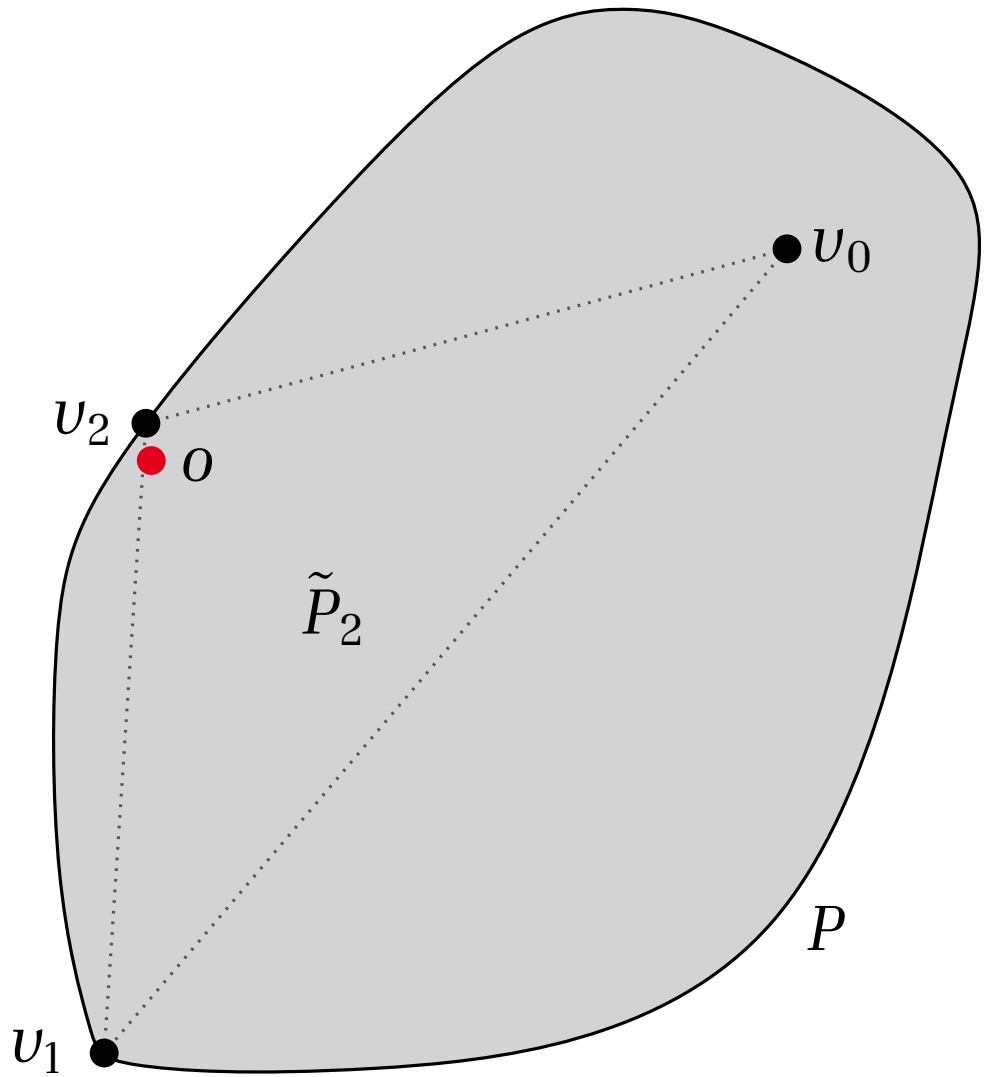
$v_2 \leftarrow \Sigma_P(n_2)$;
$h_P(n_2) \geq 0$



DSS when $o \in P$

$\mathsf{Sep}(\tilde{P}_2) = \emptyset \Rightarrow \mathsf{Sep}(P) = \emptyset$
We conclude that $o \in P$





Comparing DSS and GJK

DSS is optimized for yes/no. GJK is optimized to compute a distance.

1: **function** DSS_or_GJK (P, $\tilde{P}_i$)
2: $n \leftarrow$ a direction in the separating set of $\tilde{P}_i$
3: $v_{i+1} \leftarrow \Sigma_P(n)$
4: **if** $h_P(n) < 0$ **then return** "$o \notin P$"
5: $\tilde{P}_{i+1} \leftarrow$ better approximation of P using $\tilde{P}_i$ and v_{i+1}
6: **if** $o \in \tilde{P}_{i+1}$ **then return** "$o \in P$"
7: **return** DSS_or_GJK(P, $\tilde{P}_{i+1}$)

Comparing DSS and GJK

DSS is optimized for yes/no. GJK is optimized to compute a distance.

1: **function** DSS_or_GJK (P, $\tilde{P}_i$)
2: $n \leftarrow$ a direction in the separating set of $\tilde{P}_i$
3: $v_{i+1} \leftarrow \Sigma_P(n)$
4: **if** $h_P(n) < 0$ **then return** "$o \notin P$"
5: $\tilde{P}_{i+1} \leftarrow$ better approximation of P using $\tilde{P}_i$ and v_{i+1}
6: **if** $o \in \tilde{P}_{i+1}$ **then return** "$o \in P$"
7: **return** DSS_or_GJK(P, $\tilde{P}_{i+1}$)

Comparing DSS and GJK

DSS is optimized for yes/no. GJK is optimized to compute a distance.

1: **function** DSS_or_GJK (P, $\tilde{P}_i$)
2: $n \leftarrow$ a direction in the separating set of $\tilde{P}_i$
3: $v_{i+1} \leftarrow \Sigma_P(n)$
4: **if** $h_P(n) < 0$ **then return** "$o \notin P$"
5: $\tilde{P}_{i+1} \leftarrow$ better approximation of P using $\tilde{P}_i$ and v_{i+1}
6: **if** $o \in \tilde{P}_{i+1}$ **then return** "$o \in P$"
7: **return** DSS_or_GJK(P, $\tilde{P}_{i+1}$)

Comparing DSS and GJK

DSS is optimized for yes/no. GJK is optimized to compute a distance.

1: **function** DSS_or_GJK (P, $\tilde{P}_i$)
2: $n \leftarrow$ a direction in the separating set of $\tilde{P}_i$
3: $v_{i+1} \leftarrow \Sigma_P(n)$
4: **if** $h_P(n) < 0$ **then return** "$o \notin P$"
5: $\tilde{P}_{i+1} \leftarrow$ better approximation of P using $\tilde{P}_i$ and v_{i+1}
6: **if** $o \in \tilde{P}_{i+1}$ **then return** "$o \in P$"
7: **return** DSS_or_GJK(P, $\tilde{P}_{i+1}$)

Comparing DSS and GJK

DSS is optimized for yes/no. GJK is optimized to compute a distance.

1: **function** DSS_or_GJK (P, $\tilde{P}_i$)
2: $n \leftarrow$ a direction in the separating set of $\tilde{P}_i$
3: $v_{i+1} \leftarrow \Sigma_P(n)$
4: **if** $h_P(n) < 0$ **then return** "$o \notin P$"
5: $\tilde{P}_{i+1} \leftarrow$ better approximation of P using $\tilde{P}_i$ and v_{i+1}
6: **if** $o \in \tilde{P}_{i+1}$ **then return** "$o \in P$"
7: **return** DSS_or_GJK(P, $\tilde{P}_{i+1}$)

Comparing DSS and GJK

DSS is optimized for yes/no. GJK is optimized to compute a distance.

1: **function** DSS_or_GJK (P, $\tilde{P}_i$)
2: $n \leftarrow$ a direction in the separating set of $\tilde{P}_i$
3: $v_{i+1} \leftarrow \Sigma_P(n)$
4: **if** $h_P(n) < 0$ **then return** "$o \notin P$"
5: $\tilde{P}_{i+1} \leftarrow$ better approximation of P using $\tilde{P}_i$ and v_{i+1}
6: **if** $o \in \tilde{P}_{i+1}$ **then return** "$o \in P$"
7: **return** DSS_or_GJK(P, $\tilde{P}_{i+1}$)

Comparing DSS and GJK

DSS is optimized for yes/no. GJK is optimized to compute a distance.

1: **function** DSS_or_GJK (P, $\tilde{P}_i$)
2: $n \leftarrow$ a direction in the separating set of $\tilde{P}_i$
3: $v_{i+1} \leftarrow \Sigma_P(n)$
4: **if** $h_P(n) < 0$ **then return** "$o \notin P$"
5: $\tilde{P}_{i+1} \leftarrow$ better approximation of P using $\tilde{P}_i$ and v_{i+1}
6: **if** $o \in \tilde{P}_{i+1}$ **then return** "$o \in P$"
7: **return** DSS_or_GJK(P, $\tilde{P}_{i+1}$)

Comparing DSS and GJK

DSS is optimized for yes/no. GJK is optimized to compute a distance.

- DSS uses $\tilde{P}_i = \mathcal{H}\left(\{v_0, v_1, \ldots, v_i\}\right)$ (better for decision).
  GJK has $\left|\tilde{P}_i\right| \leq 4$ (better for the actual distance).

1: **function** DSS_or_GJK (P, $\tilde{P}_i$)
2: $n \leftarrow$ a direction in the separating set of $\tilde{P}_i$
3: $v_{i+1} \leftarrow \Sigma_P(n)$
4: **if** $h_P(n) < 0$ **then return** "$o \notin P$"
5: $\tilde{P}_{i+1} \leftarrow$ better approximation of P using $\tilde{P}_i$ and v_{i+1}
6: **if** $o \in \tilde{P}_{i+1}$ **then return** "$o \in P$"
7: **return** DSS_or_GJK(P, $\tilde{P}_{i+1}$)

Comparing DSS and GJK

DSS is optimized for yes/no. GJK is optimized to compute a distance.

- DSS uses $\tilde{P}_i = \mathcal{H}\left(\{v_0, v_1, \ldots, v_i\}\right)$ (better for decision).
  GJK has $\left|\tilde{P}_i\right| \leq 4$ (better for the actual distance).

- DSS: $S_{i+1} \leftarrow S_i \cap v_{i+1}^{\downarrow}$ ($\approx$ planar 2D polygon clipping).
  GJK: $K \leftarrow$ point of $\tilde{P}_i$ closest to o; $\tilde{P}_{i+1} \leftarrow \left\{f(K) \cup \{v_{i+1}\}\right\}$
  (complex code / numerical issue).

1: **function** DSS_or_GJK (P, $\tilde{P}_i$)
2: $n \leftarrow$ a direction in the separating set of $\tilde{P}_i$
3: $v_{i+1} \leftarrow \Sigma_P(n)$
4: **if** $h_P(n) < 0$ **then return** "$o \notin P$"
5: $\tilde{P}_{i+1} \leftarrow$ better approximation of P using $\tilde{P}_i$ and v_{i+1}
6: **if** $o \in \tilde{P}_{i+1}$ **then return** "$o \in P$"
7: **return** DSS_or_GJK(P, $\tilde{P}_{i+1}$)

Comparing DSS and GJK

DSS is optimized for yes/no. GJK is optimized to compute a distance.

- DSS uses $\tilde{P}_i = \mathcal{H}\left(\{v_0, v_1, \ldots, v_i\}\right)$ (better for decision).
  GJK has $\left|\tilde{P}_i\right| \leq 4$ (better for the actual distance).

- DSS: $S_{i+1} \leftarrow S_i \cap v_{i+1}^{\downarrow}$ ($\approx$ planar 2D polygon clipping).
  GJK: $K \leftarrow$ point of $\tilde{P}_i$ closest to o; $\tilde{P}_{i+1} \leftarrow \left\{f(K) \cup \{v_{i+1}\}\right\}$ (complex code / numerical issue).

- DSS: $o \in \tilde{P}_i \Leftrightarrow S_i = \emptyset$ (trivial to test).
  GJK: $o \in \tilde{P}_i$? by-product of previous step (numerical issue).

```
1: function DSS_or_GJK (P, P̃_i)
2:     n ← a direction in the separating set of P̃_i
3:     v_{i+1} ← Σ_P(n)
4:     if h_P(n) < 0 then return "o ∉ P"
5:     P̃_{i+1} ← better approximation of P using P̃_i and v_{i+1}
6:     if o ∈ P̃_{i+1} then return "o ∈ P"
7:     return DSS_or_GJK(P, P̃_{i+1})
```

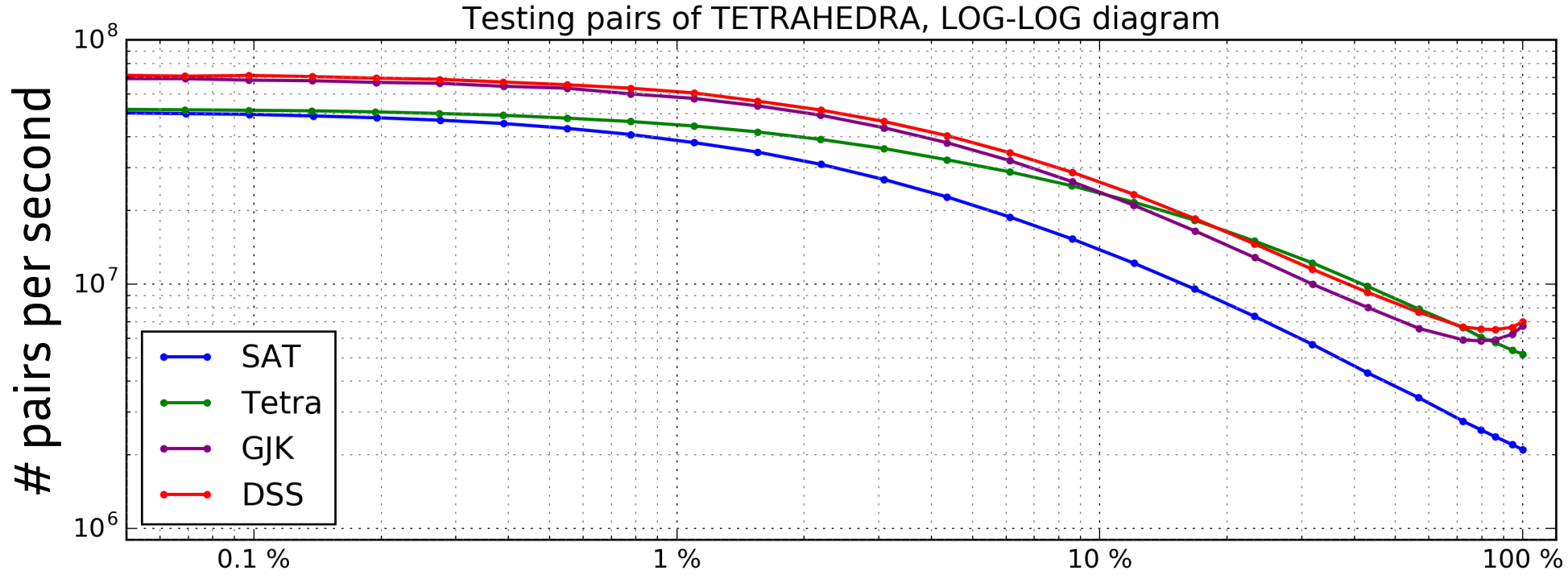
Benchmarks

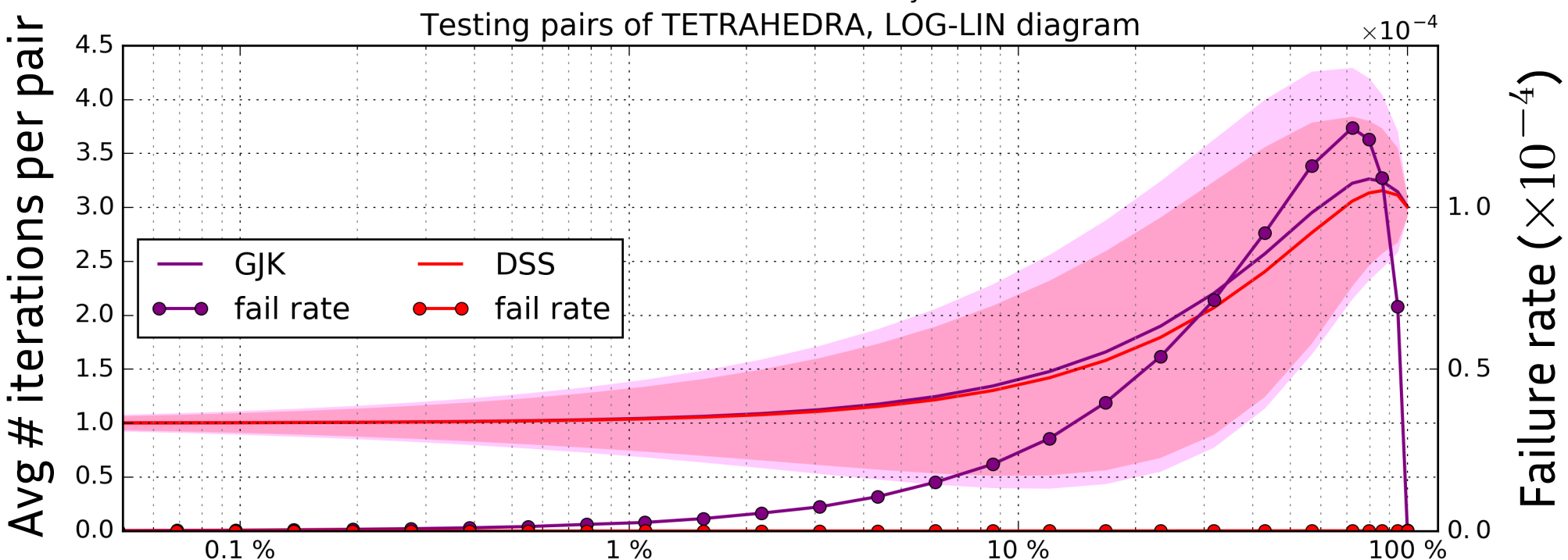
- Randomly generated shapes while controlling *collision density*
- Plot *average # of tested-pairs-per-second* v.s. density
- Plot *average # of iterations* v.s. density

Several kind of shapes:

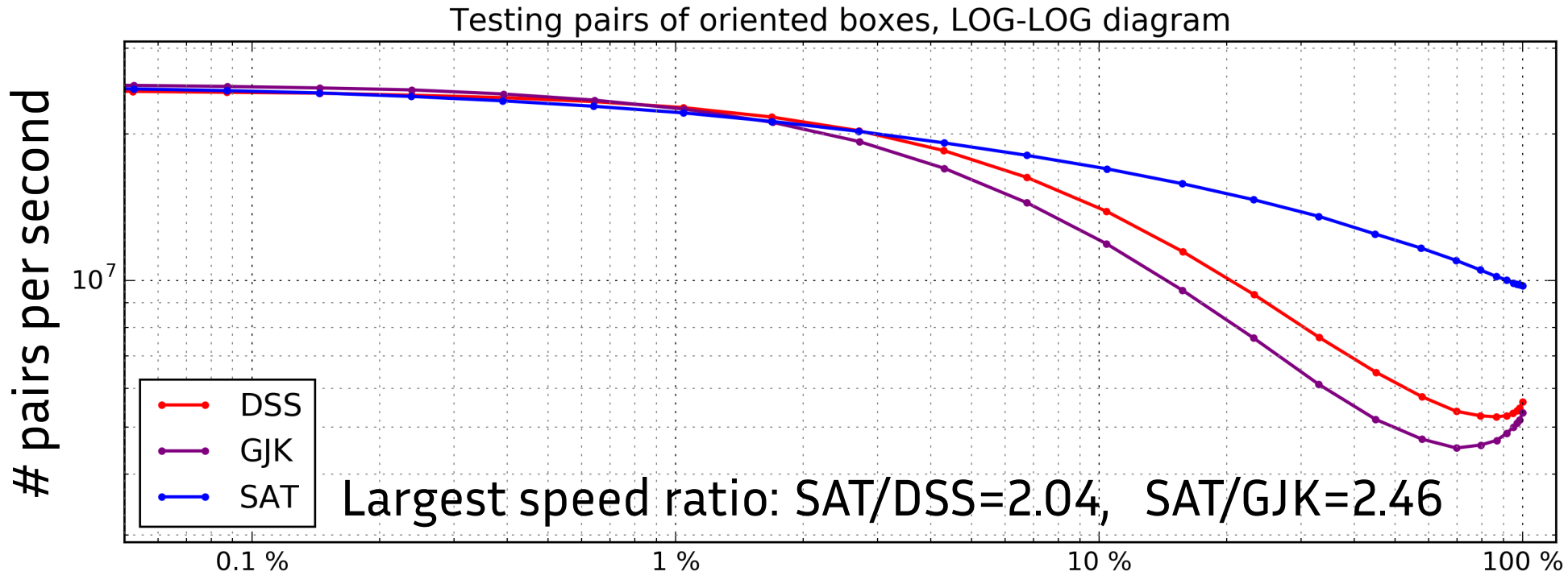
- tetrahedra
- oriented boxes
- polytopes with 16 vertices
- polytopes with increasing size (@ 50 % collision density)
- frustum culling axis-aligned boxes
- frustum culling spheres
- hybrid algorithm

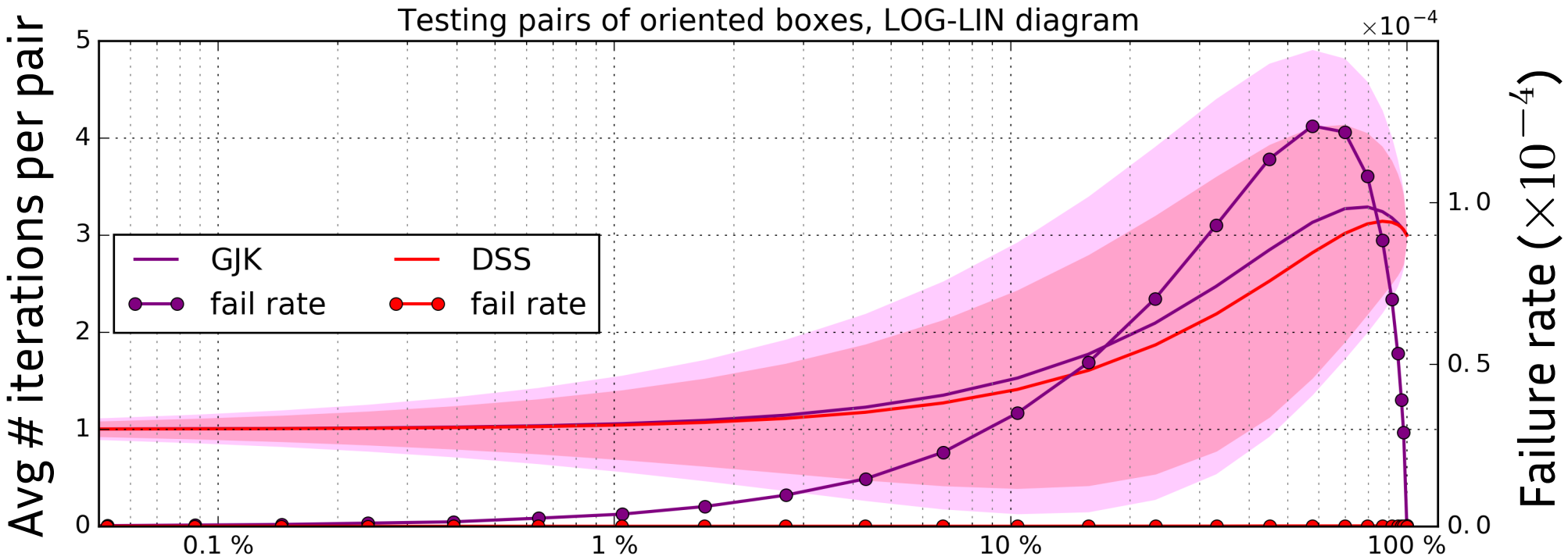
Tetrahedra


Testing pairs of TETRAHEDRA, LOG-LOG diagram
pairs per second
10^8
10^7
10^6
SAT
Tetra
GJK
DSS
0.1 %
1 %
10 %
100 %
Collision density


Testing pairs of TETRAHEDRA, LOG-LIN diagram
Avg # iterations per pair
Failure rate ($\times 10^{-4}$)
$\times 10^{-4}$
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
1.0
0.5
0.0
GJK
fail rate
DSS
fail rate
0.1 %
1 %
10 %
100 %

Oriented boxes





Thank you

C++ code, benchmark script & figure script:
https://github.com/horasio/intersection-detection

The paper on HAL:
https://hal.inria.fr/hal-01522903

How many iterations?

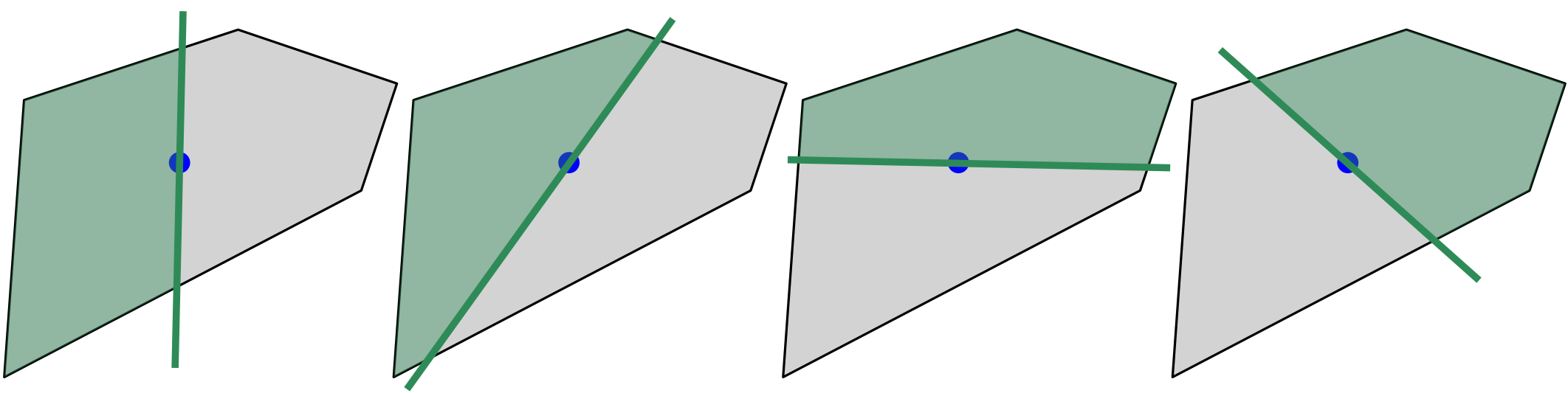
If we assume that we are able to find a centerpoint n of a convex spherical polygon S efficiently. Then, there exists a constant $\mu > 1$ such that $n \cdot v \geq 0 \Rightarrow \text{area}(S \cap v^{\downarrow}) \leq \text{area}(S)/\mu$.

After the k-th iteration, if a decision has not yet been reached, the search polygon S_k must still contain $\text{Sep}(A \ominus B)$.

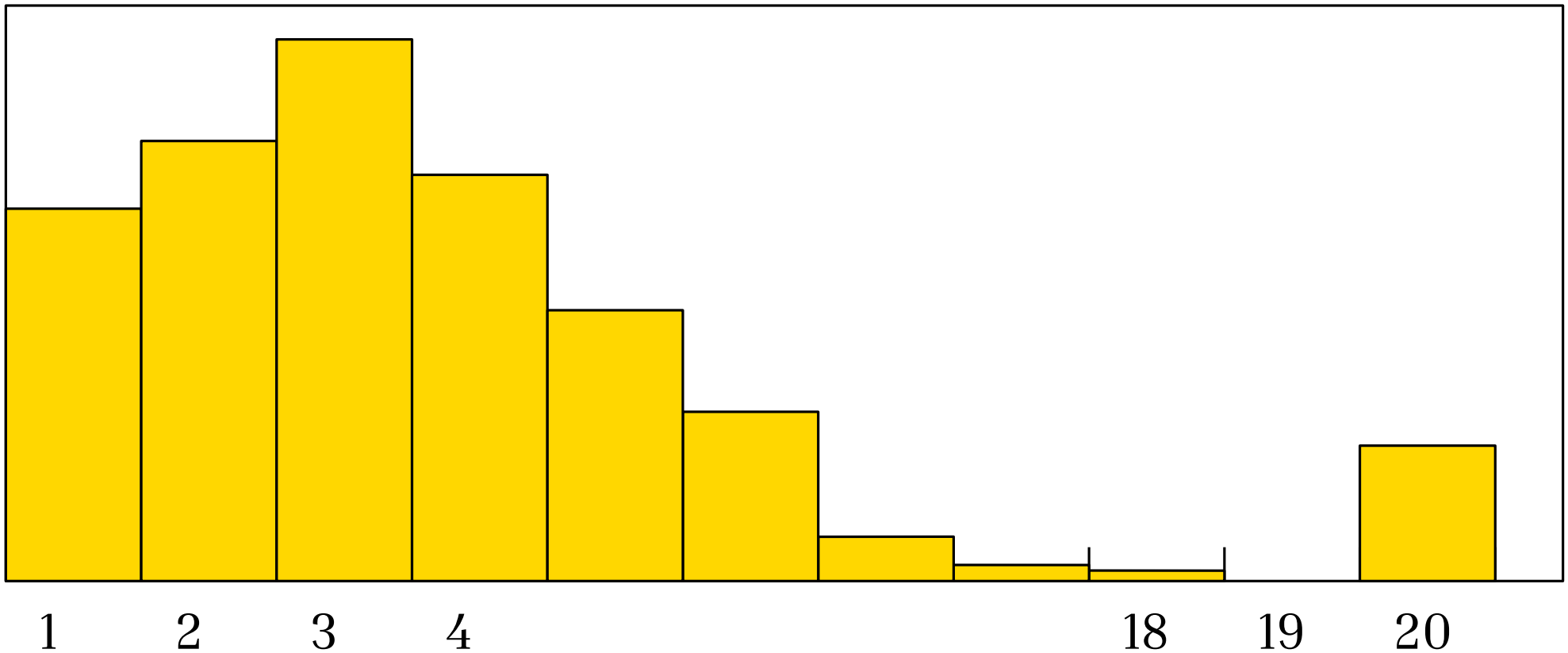
Thus, $\text{area}(\text{Sep}(A \ominus B)) \leq \text{area}(S_k) \leq 2\pi\mu^{-k}$.

This implies

$$k \leq \log_\mu \left(\frac{2\pi}{\text{area}(\text{Sep}(A \ominus B))} \right).$$



Infinite loop



Histogram of the number of iterations

More about the separating set

Observations:

1. $\mathbb{S}^{2^-}(\{p\}) = p^{\downarrow}$
2. $\mathsf{Sep}(X \cup Y) = \mathsf{Sep}(X) \cap \mathsf{Sep}(Y)$
3. $\mathsf{Sep}(\mathcal{H}(X)) = \mathsf{Sep}(X)$
4. $X \subset Y \Rightarrow \mathsf{Sep}(Y) \subset \mathsf{Sep}(X)$
5. $\forall \lambda > 0, \mathsf{Sep}(\lambda X) = \mathsf{Sep}(X)$

$$\mathsf{Sep}(P) = \bigcap_{p \in P} p^{\downarrow} \qquad S(P) = \bigcap_{s \in \mathsf{sil}(P)} s^{\downarrow}$$

Let $V_k = \{v_0, v_1, \ldots, v_k\}$ and $\mathcal{I}$ the indices of the silhouette vertices of $\mathcal{H}(V_k)$.

$$\mathsf{Sep}(\mathcal{H}(V_k)) = \bigcap_{i=0}^{k} v_i^{\downarrow} \qquad \mathsf{Sep}(\mathcal{H}(V_k)) = \bigcap_{i \in \mathcal{I}} v_i^{\downarrow}$$

How to compute $d(o, P\text{olyhedron})$?

Without pre-processing:

1. Compute the position of o w.r.t. each face of P.
2. Conclude.

How to compute $d(o, P\text{olyhedron})$?

Without pre-processing:

1. Compute the position of o w.r.t. each face of P.
2. Conclude.

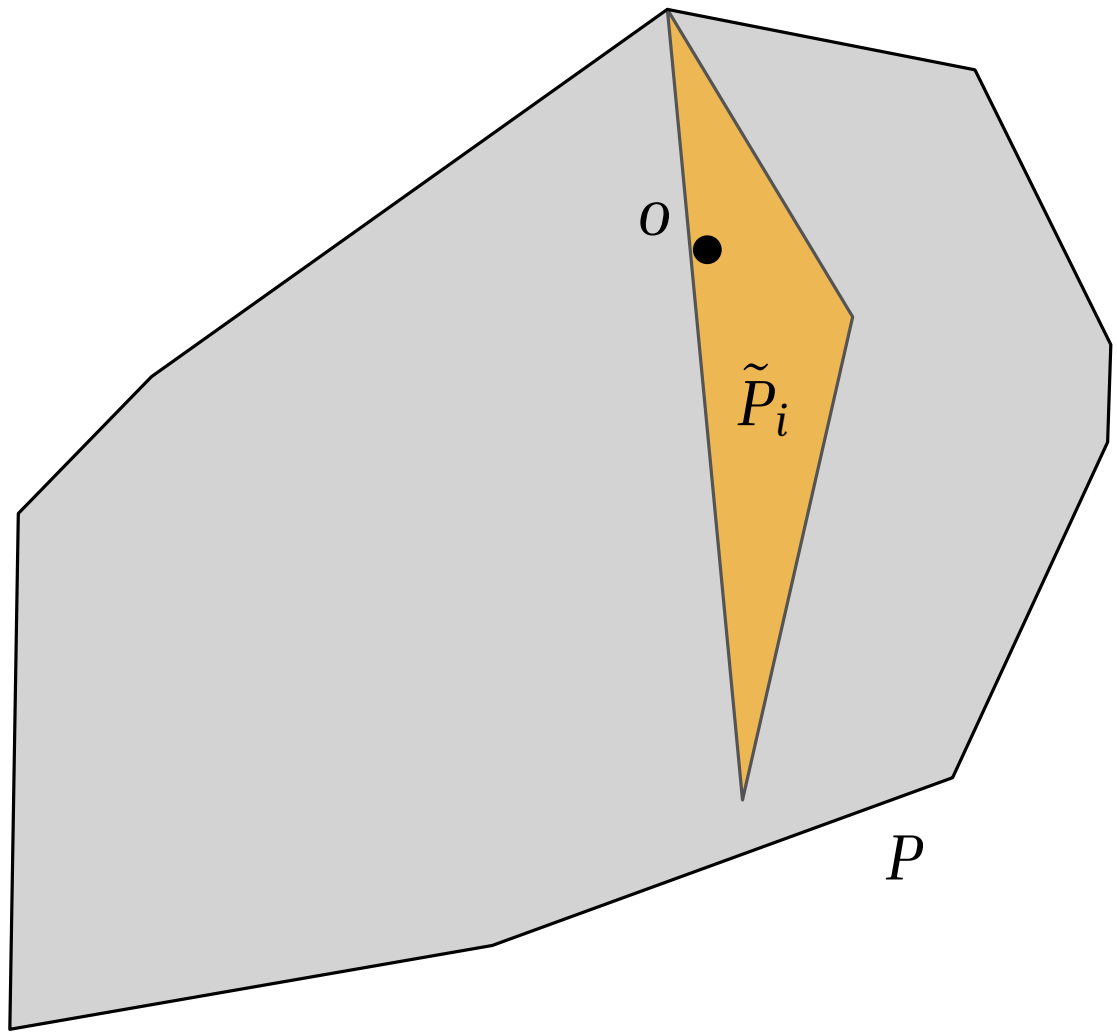
Complexity too high: $|A \ominus B| = O(|A| \times |B|)$

GJK for computing $d(o, P\text{olyhedron})$

GJK for computing $d(o, P\text{olyhedron})$

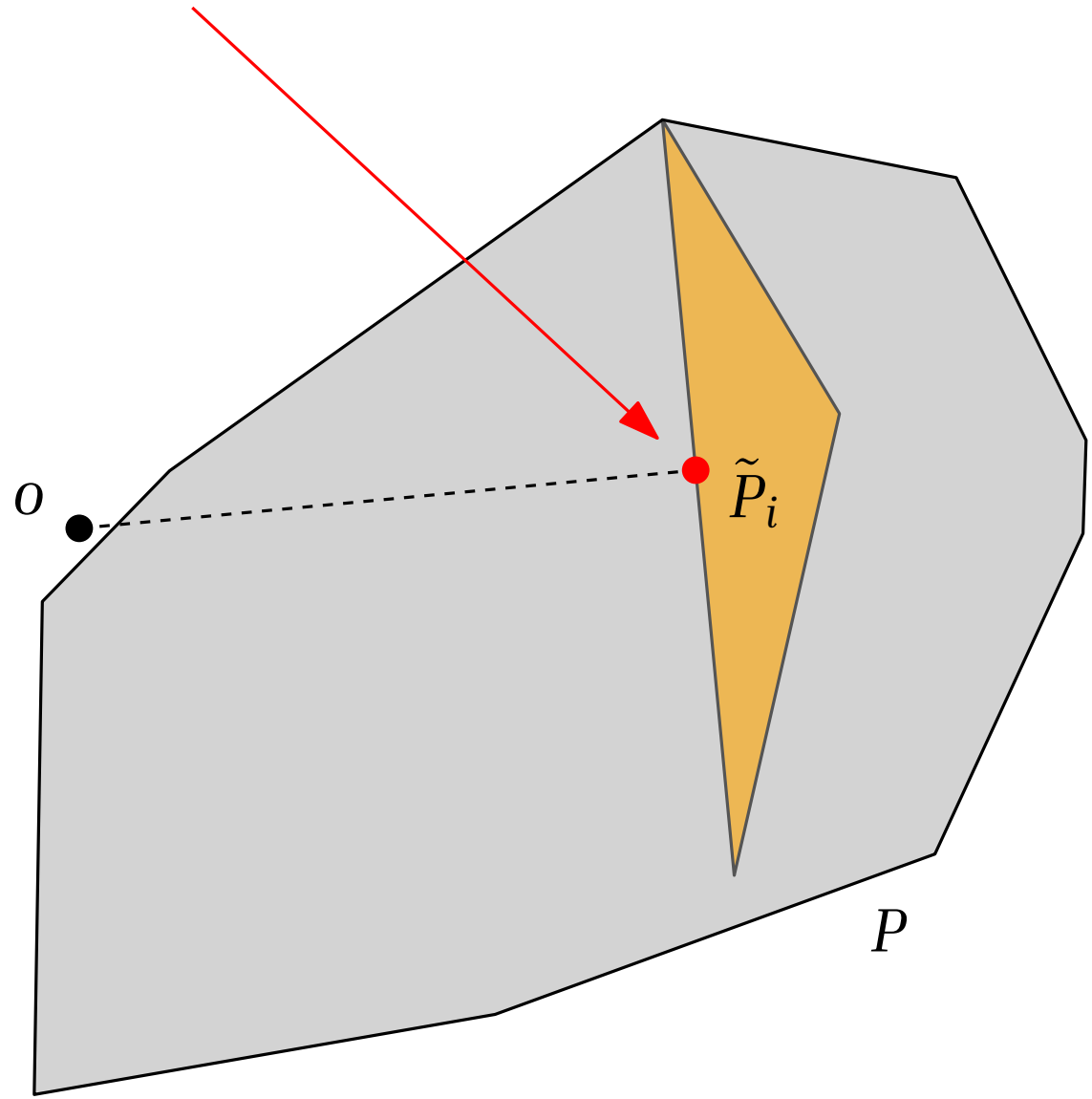
GJK uses a simplex $\tilde{P}_i \subset P$ as a proxy for P, $i = 0, 1, 2, \ldots$

1. If $o \in \tilde{P}_i$, conclude that $d(o, P) = 0$.



GJK for computing $d(o, P\text{olyhedron})$

2. Otherwise, compute K_i, the point of $\tilde{P}_i$ closest to the origin.



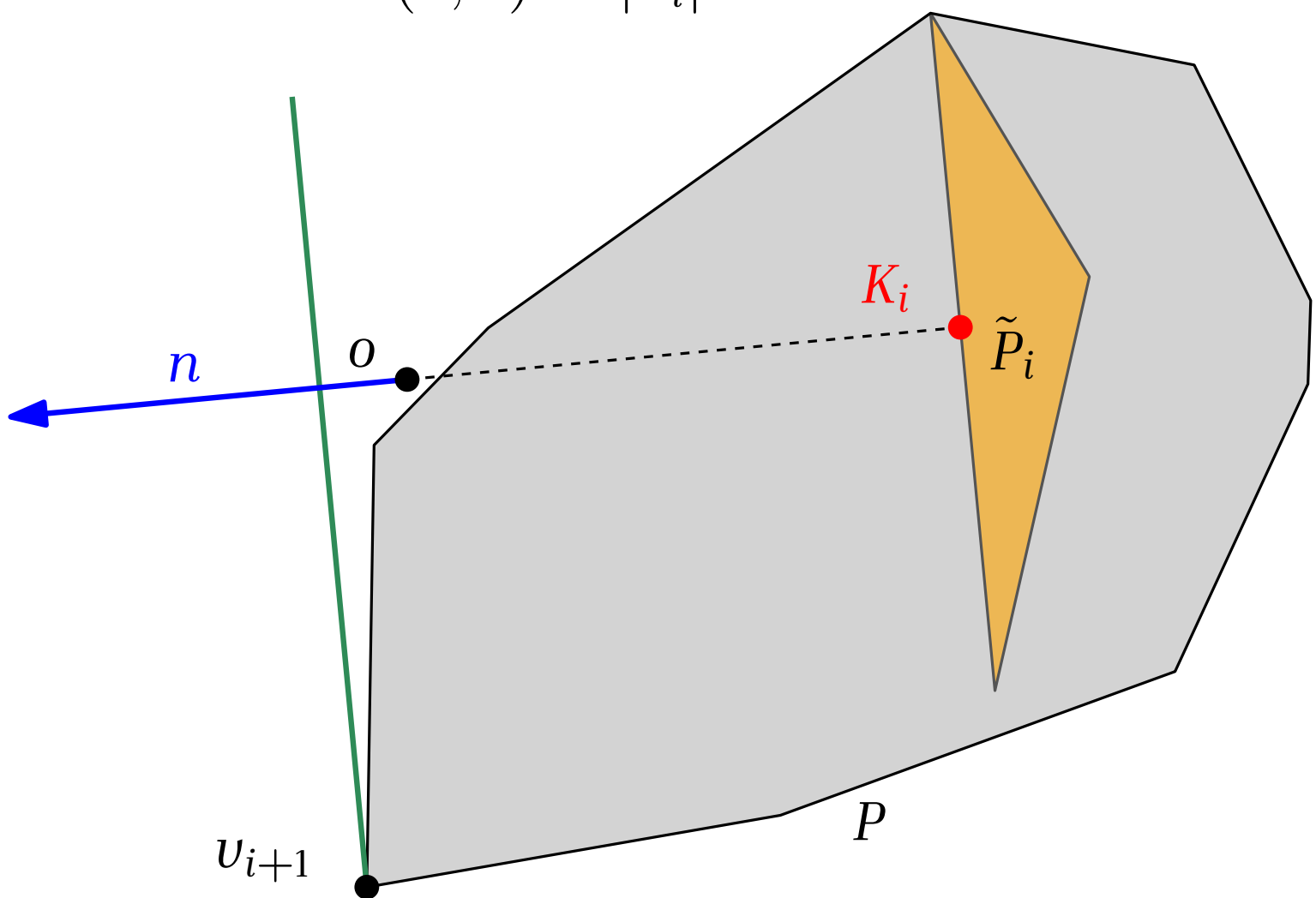
GJK for computing $d(o, P\text{olyhedron})$

3. $n \leftarrow -\nu(\tilde{P}_i)$

   $v_{i+1} \leftarrow \Sigma_P(n)$

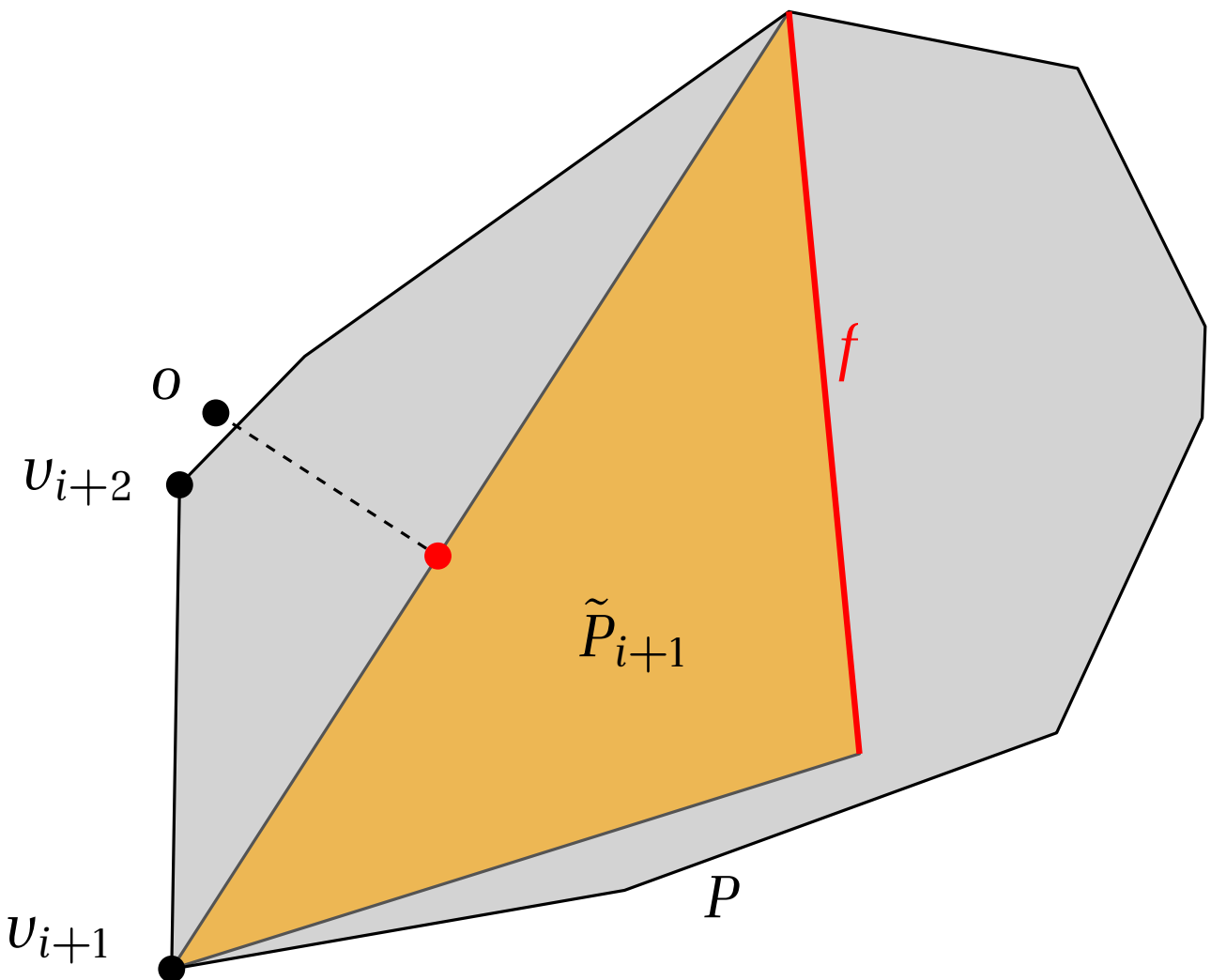
If $v_{i+1} \cdot K_i = |K_i|^2$ *//i.e.* K_i can not move closer to o

then we conclude that $d(o, P) = |K_i|$

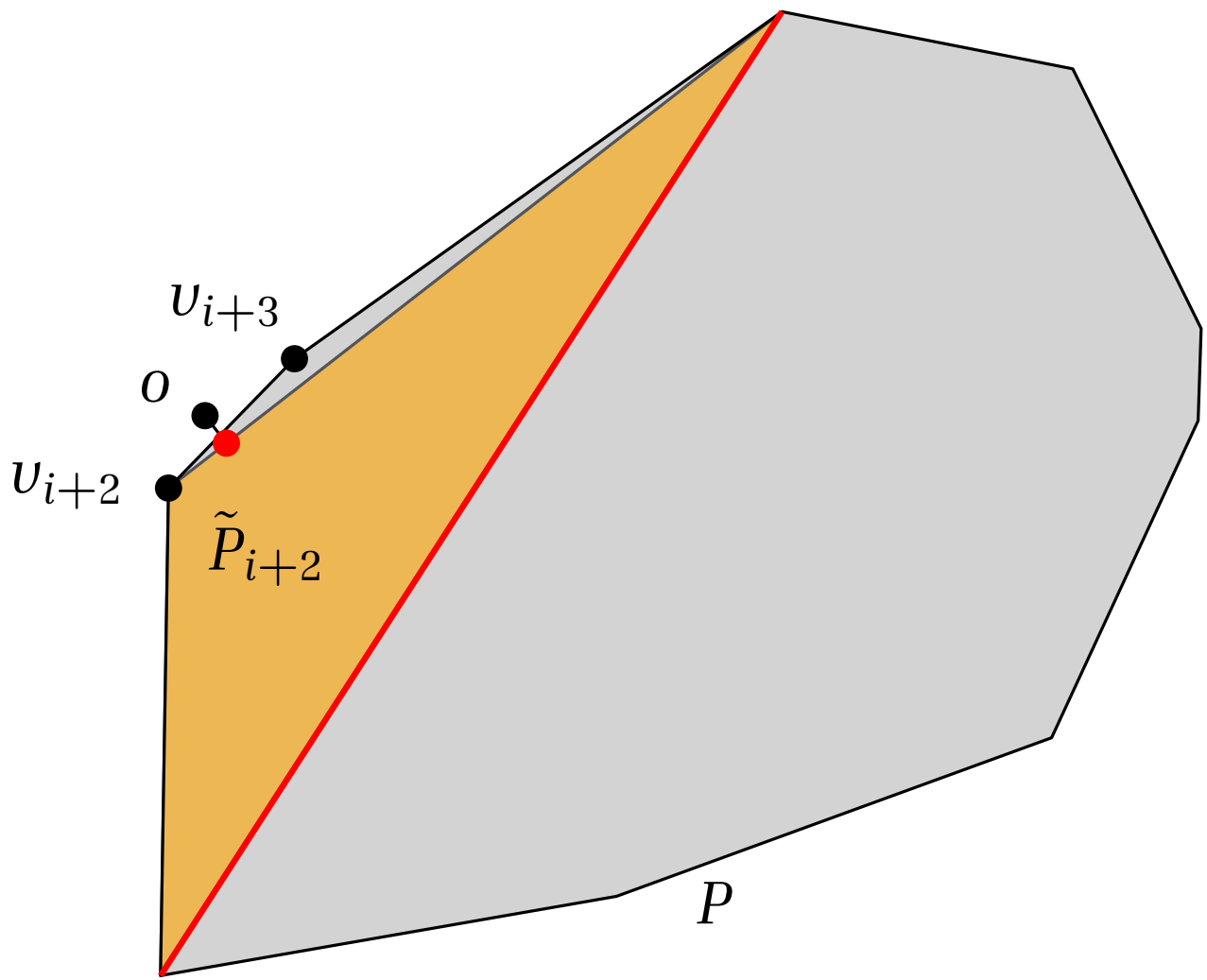


GJK for computing $d(o, P\text{olyhedron})$

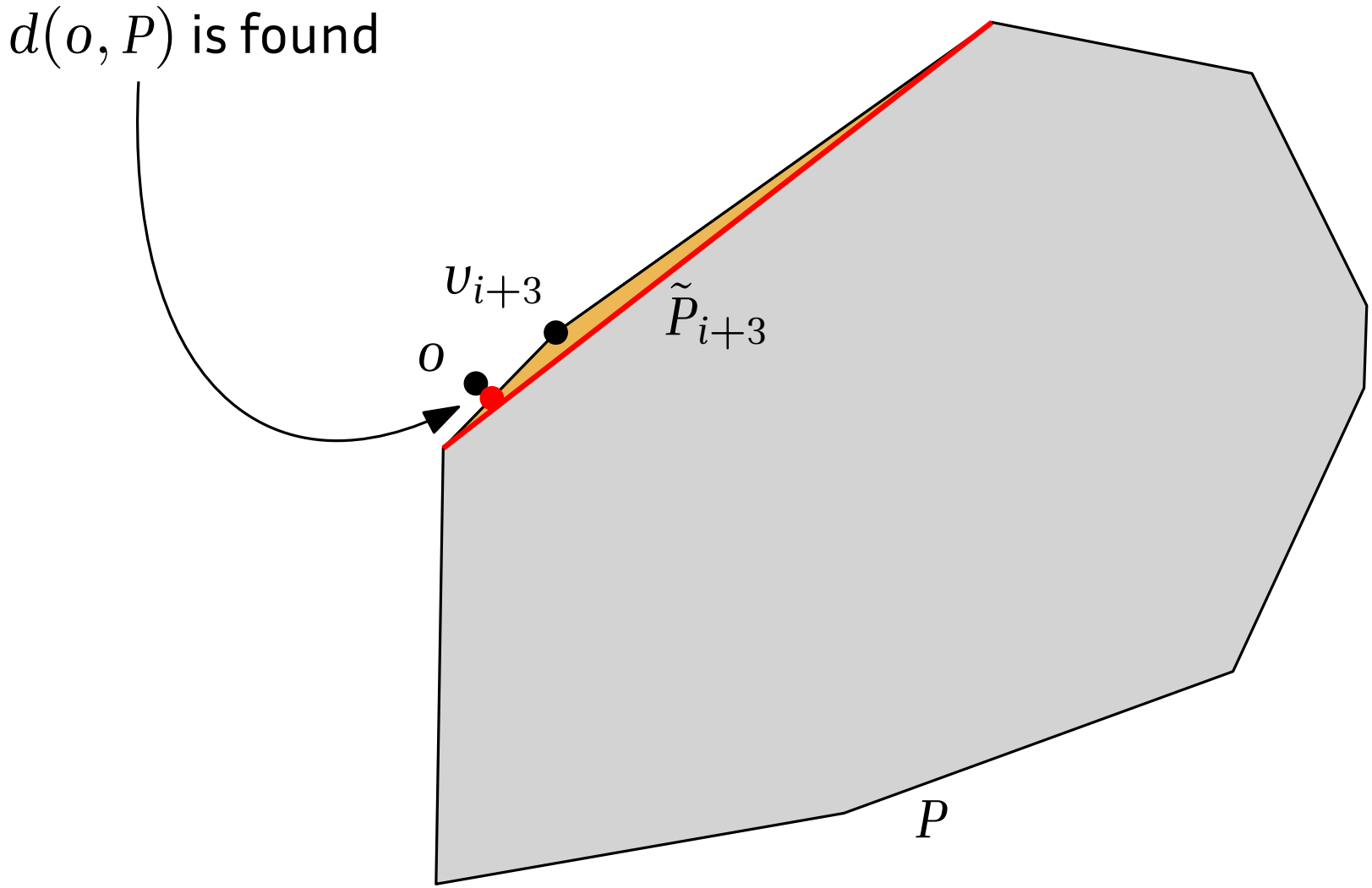
4. Otherwise, build $\tilde{P}_{i+1}$ from $\tilde{P}_i$ and v_{i+1}:
$\tilde{P}_{i+1} \leftarrow \mathcal{H}(f \cup \{v_{i+1}\})$



GJK for computing $d(o, P\text{olyhedron})$



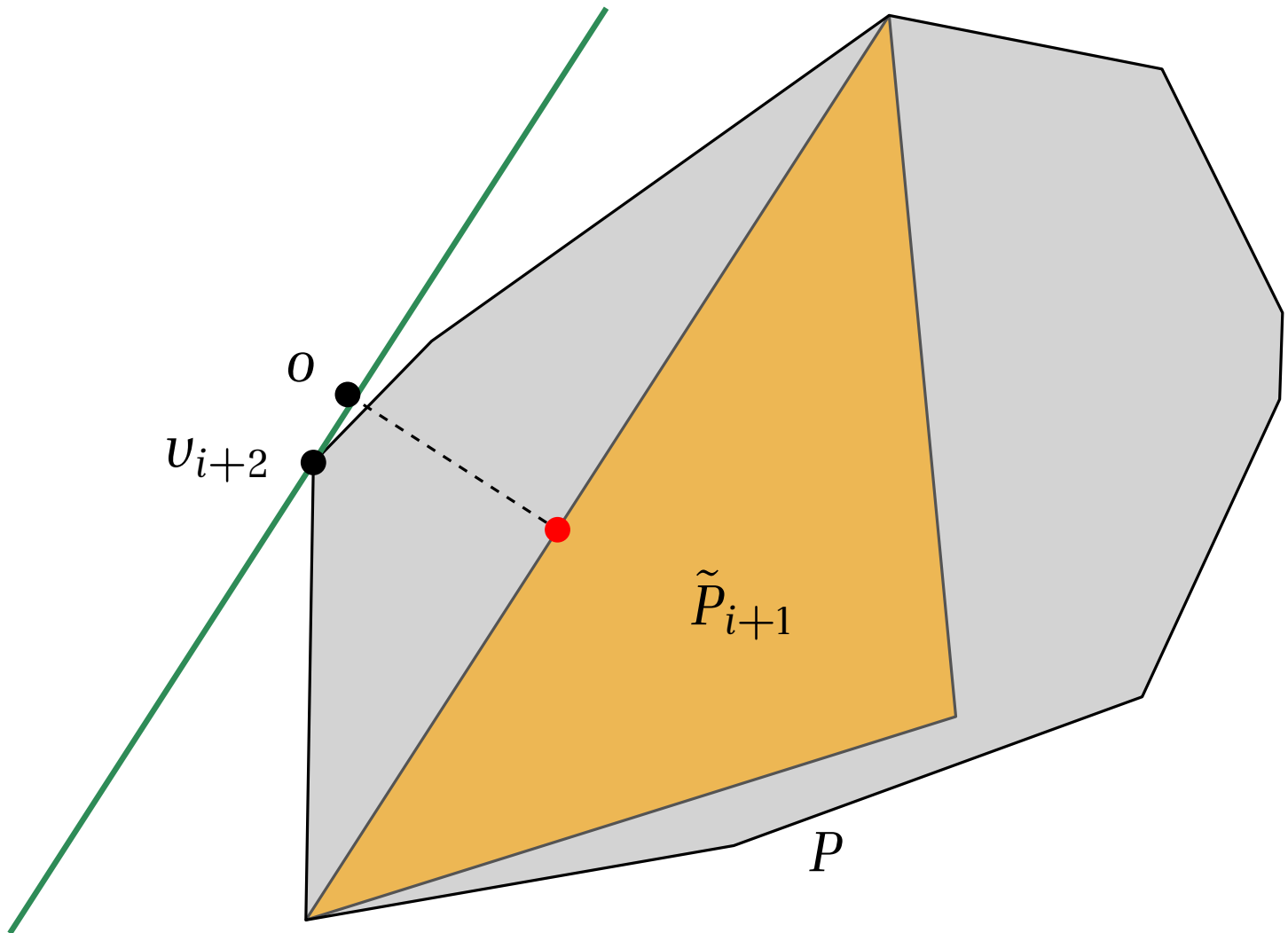
GJK for computing $d(o, P\text{olyhedron})$



GJK for deciding if $o \in P$

$n \leftarrow -K_i; \quad v_{i+1} \leftarrow \Sigma_P(n);$

If $v_{i+1} \cdot K_i = |K_i|^2 > 0$ then we have $o \notin P$



DSS vs GJK

1: **function** DSS_or_GJK(P, $\tilde{P}_i$)
2: $n \leftarrow$ a direction in the separating set of $\tilde{P}_i$
3: $v_{i+1} \leftarrow \Sigma_P(n)$
4: **if** $h_P(n) < 0$ **then return** "$o \notin P$"
5: $\tilde{P}_{i+1} \leftarrow$ better approximation of P using $\tilde{P}_i$ and v_{i+1}
6: **if** $o \in \tilde{P}_{i+1}$ **then return** "$o \in P$"
7: **return** DSS_or_GJK(P, $\tilde{P}_{i+1}$)

DSS vs GJK

1: **function** DSS_or_GJK(P, $\tilde{P}_i$)
2: $n \leftarrow$ a direction in the separating set of $\tilde{P}_i$
3: $v_{i+1} \leftarrow \Sigma_P(n)$
4: **if** $h_P(n) < 0$ **then return** "$o \notin P$"
5: $\tilde{P}_{i+1} \leftarrow$ better approximation of P using $\tilde{P}_i$ and v_{i+1}
6: **if** $o \in \tilde{P}_{i+1}$ **then return** "$o \in P$"
7: **return** DSS_or_GJK(P, $\tilde{P}_{i+1}$)

In both GJK and DSS:

- $\tilde{P}_0 = \{v_0\} \subset P$
- Vertices computed at line 3 form $V_i = \{v_0, v_1, \ldots, v_i\}$

DSS vs GJK

1: **function** DSS_or_GJK(P, $\tilde{P}_i$)
2: $n \leftarrow$ a direction in the separating set of $\tilde{P}_i$
3: $v_{i+1} \leftarrow \Sigma_P(n)$
4: **if** $h_P(n) < 0$ **then return** "$o \notin P$"
5: $\tilde{P}_{i+1} \leftarrow$ better approximation of P using $\tilde{P}_i$ and v_{i+1}
6: **if** $o \in \tilde{P}_{i+1}$ **then return** "$o \in P$"
7: **return** DSS_or_GJK(P, $\tilde{P}_{i+1}$)

DSS

The polyhedron $\tilde{P}_i$

GJK

$\tilde{P}_i = \mathcal{H}(V_i)$ **not** stored
$S_i = \text{Sep}(\tilde{P}_i)$ stored

Best possible approximation
$\tilde{P}_i \subset \tilde{P}_{i+1}$

$\tilde{P}_i = \mathcal{H}(W \subset V_i)$
$|W| \leq 4$

DSS vs GJK

1: **function** DSS_or_GJK(P, $\tilde{P}_i$)
2: $n \leftarrow$ a direction in the separating set of $\tilde{P}_i$
3: $v_{i+1} \leftarrow \Sigma_P(n)$
4: **if** $h_P(n) < 0$ **then return** "$o \notin P$"
5: $\tilde{P}_{i+1} \leftarrow$ better approximation of P using $\tilde{P}_i$ and v_{i+1}
6: **if** $o \in \tilde{P}_{i+1}$ **then return** "$o \in P$"
7: **return** DSS_or_GJK(P, $\tilde{P}_{i+1}$)

Test if $o \in \tilde{P}_{i+1}$?

DSS

$\Leftrightarrow S_{i+1} = \emptyset$
trivial to test

GJK

point-in-tetrahedron test
complex code
numerically sensitive
($\tilde{P}_{i+1} \approx$ flat tetrahedron)

DSS vs GJK

1: **function** DSS_or_GJK(P, $\tilde{P}_i$)
2: $n \leftarrow$ a direction in the separating set of $\tilde{P}_i$
3: $v_{i+1} \leftarrow \Sigma_P(n)$
4: **if** $h_P(n) < 0$ **then return** "$o \notin P$"
5: $\tilde{P}_{i+1} \leftarrow$ better approximation of P using $\tilde{P}_i$ and v_{i+1}
6: **if** $o \in \tilde{P}_{i+1}$ **then return** "$o \in P$"
7: **return** DSS_or_GJK(P, $\tilde{P}_{i+1}$)

DSS

Pick a direction $n \in \text{Sep}(\tilde{P}_i)$

GJK

average of vertices in S_i
(best: centerpoint)

closest point on tetrahedron to o

product of code from line 6 in previous iteration

DSS vs GJK

1: **function** DSS_or_GJK(P, $\tilde{P}_i$)
2: $n \leftarrow$ a direction in the separating set of $\tilde{P}_i$
3: $v_{i+1} \leftarrow \Sigma_P(n)$
4: **if** $h_P(n) < 0$ **then return** "$o \notin P$"
5: $\tilde{P}_{i+1} \leftarrow$ better approximation of P using $\tilde{P}_i$ and v_{i+1}
6: **if** $o \in \tilde{P}_{i+1}$ **then return** "$o \in P$"
7: **return** DSS_or_GJK(P, $\tilde{P}_{i+1}$)

DSS

Compute $\tilde{P}_{i+1}$

GJK

Update S_{i+1} by
2D (spherical) polygon clipping
(careful with the lunes)

$S_{i+1} \leftarrow S_i \cap v_{i+1}^{\downarrow}$

Update $\tilde{P}_i$ by
selecting $f \subset \tilde{P}_i$ $\quad |f| \leq 3$

$\tilde{P}_{i+1} \leftarrow \mathcal{H}(f \cup \{v_{i+1}\})$