Search and Intersection

O'Rourke, Chapter 7

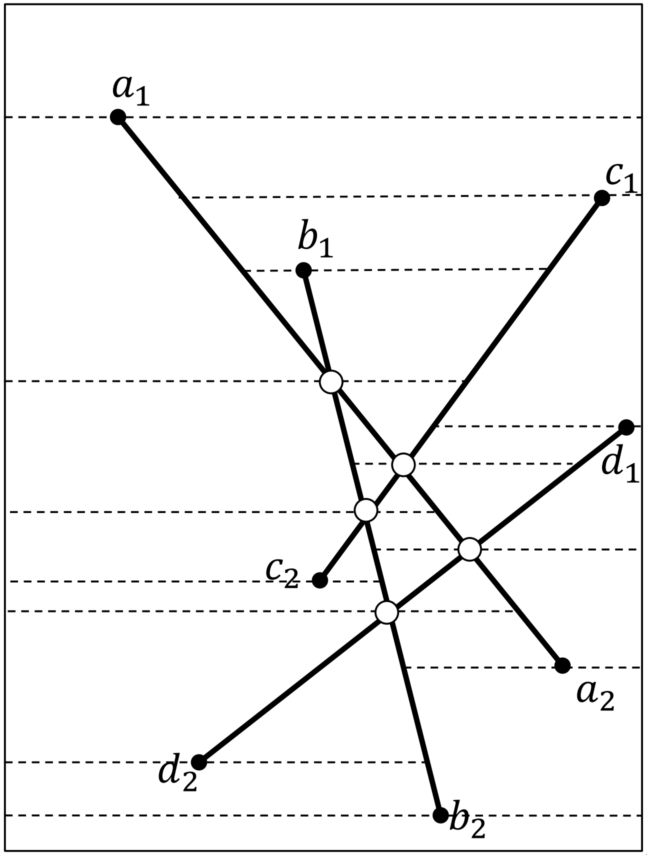
Outline

- Trapezoidal Decomposition
- Extreme Points (2D)
- Extreme Points (3D)

Trapezoidal Decomposition

Goal:

Given a set of line segments partition space into trapezoids with horizontal tops/bottoms so that each trapezoid has unique left/right neighbors.*



*Assume no line segment is vertical.

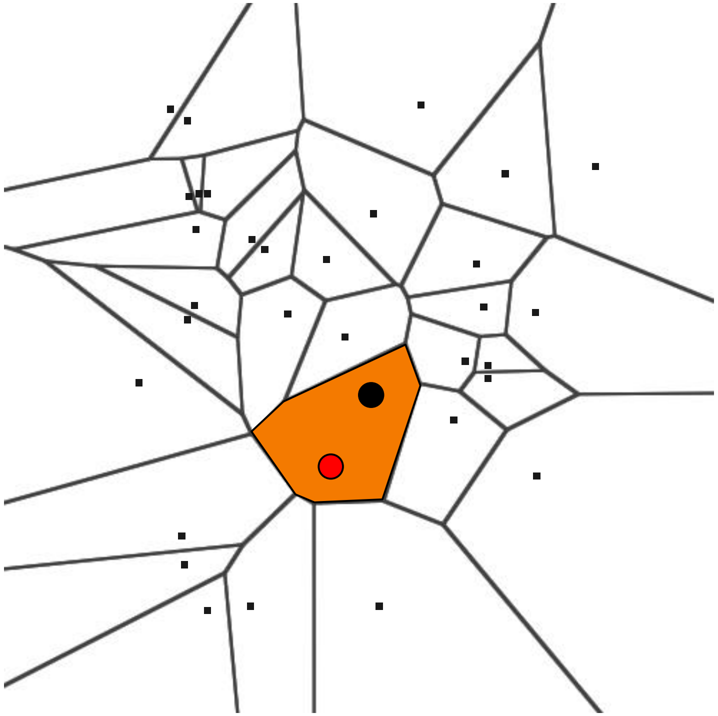
Trapezoidal Decomposition

Goal:

Given a partition of 2D space into polygons, (efficiently) compute a (compact) data-structure that enables fast point-in-polygon queries.

Example (Nearest-Neighbor):

Given the Voronoi diagram of a set of points, we would like to quickly determine to which Voronoi cell a point belongs.

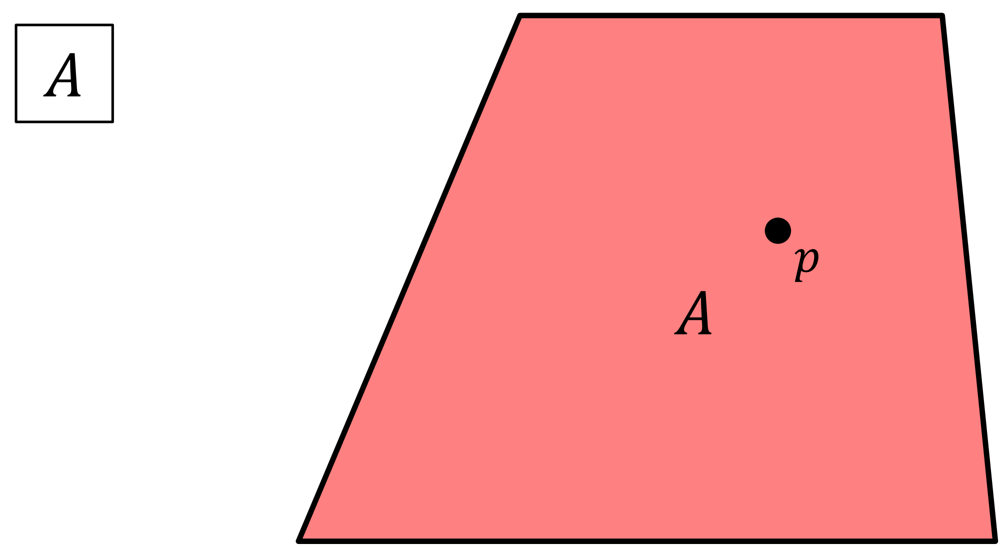


Trapezoidal Decomposition

Approach:

Construct the partition iteratively, adding new line-segments into existing partition:

- Add end-point, performing top/bottom split of containing trapezoid.

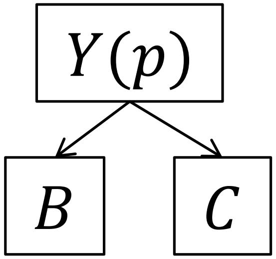


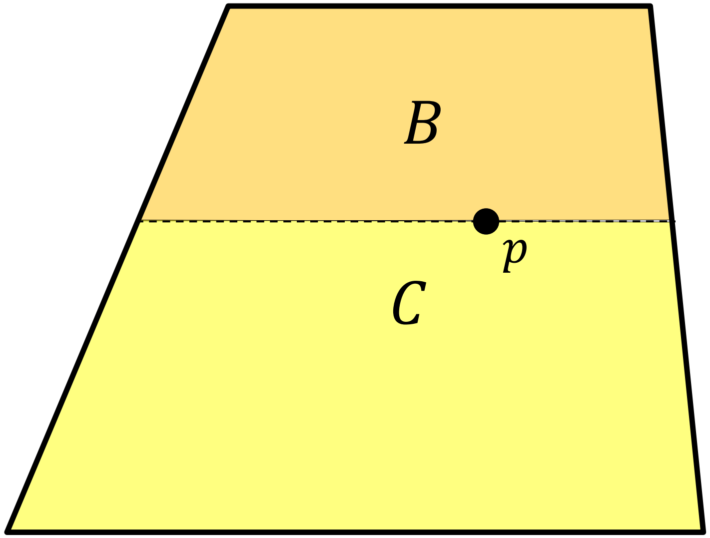
Trapezoidal Decomposition

Approach:

Construct the partition iteratively, adding new line-segments into existing partition:

- Add end-point, performing top/bottom split of containing trapezoid.



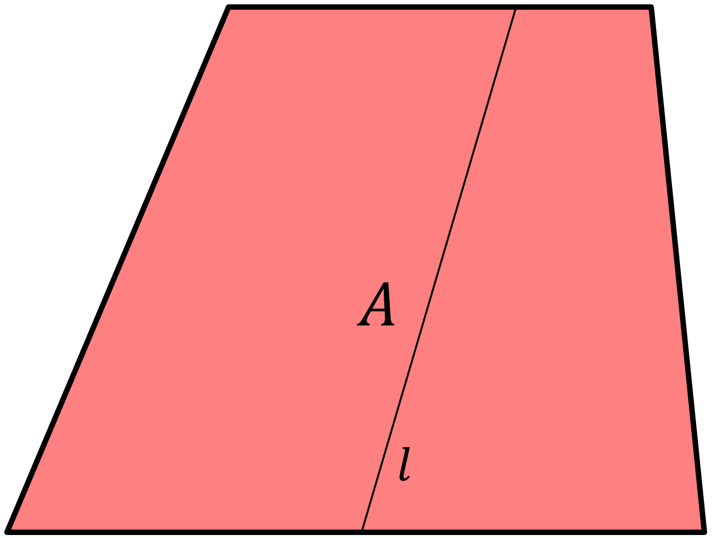


Trapezoidal Decomposition

Approach:

Construct the partition iteratively, adding new line-segments into existing partition:

- Add end-point.
- Add line segment, splitting the trapezoid into 2, 3, or 4 sub-trapezoids.

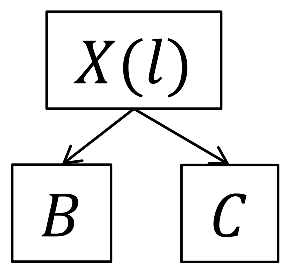


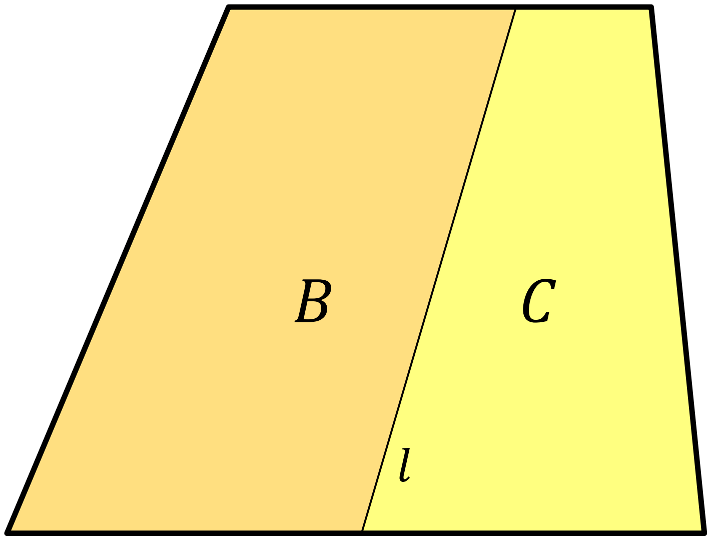
Trapezoidal Decomposition

Approach:

Construct the partition iteratively, adding new line-segments into existing partition:

- Add end-point.
- Add line segment, splitting the trapezoid into 2, 3, or 4 sub-trapezoids.



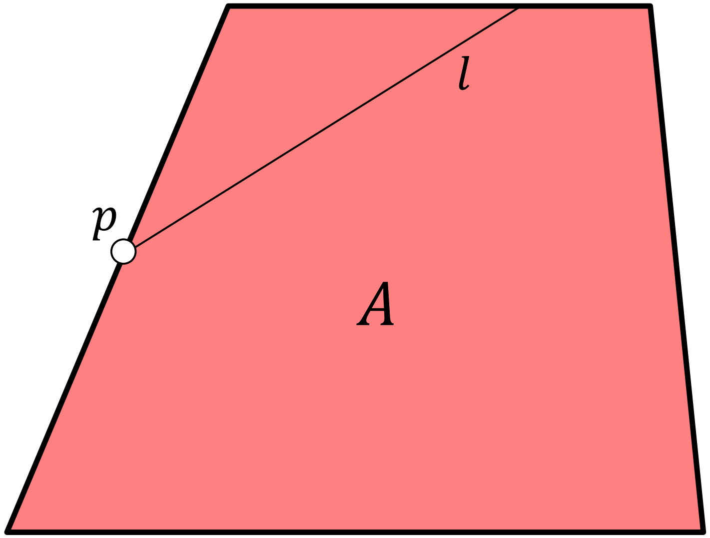


Trapezoidal Decomposition

Approach:

Construct the partition iteratively, adding new line-segments into existing partition:

- Add end-point.
- Add line segment, splitting the trapezoid into 2, 3, or 4 sub-trapezoids.

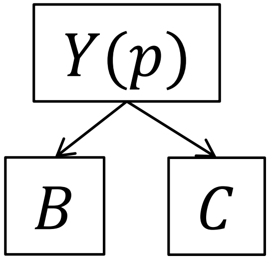


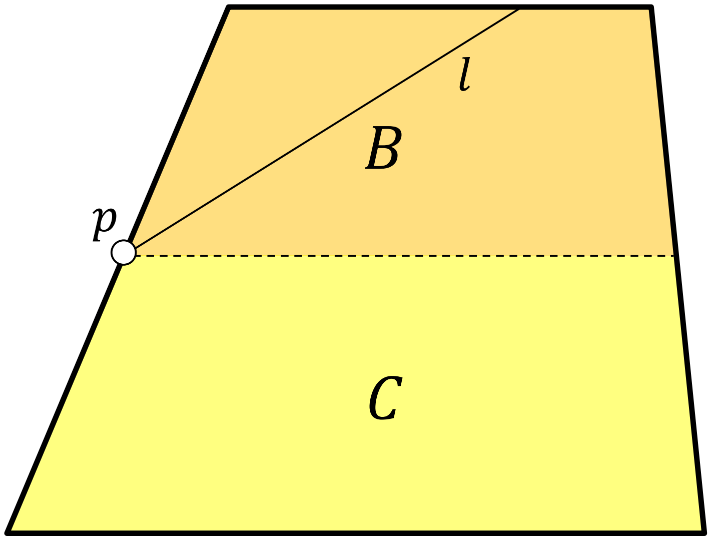
Trapezoidal Decomposition

Approach:

Construct the partition iteratively, adding new line-segments into existing partition:

- Add end-point.
- Add line segment, splitting the trapezoid into 2, 3, or 4 sub-trapezoids.



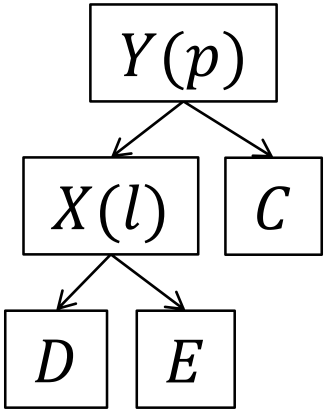


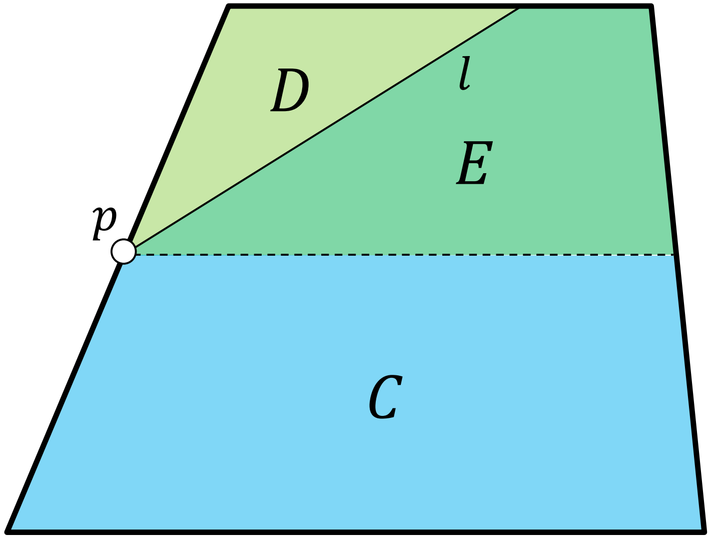
Trapezoidal Decomposition

<u>Approach</u>:

Construct the partition iteratively, adding new line-segments into existing partition:

- Add end-point.
- Add line segment, splitting the trapezoid into 2, 3, or 4 sub-trapezoids.



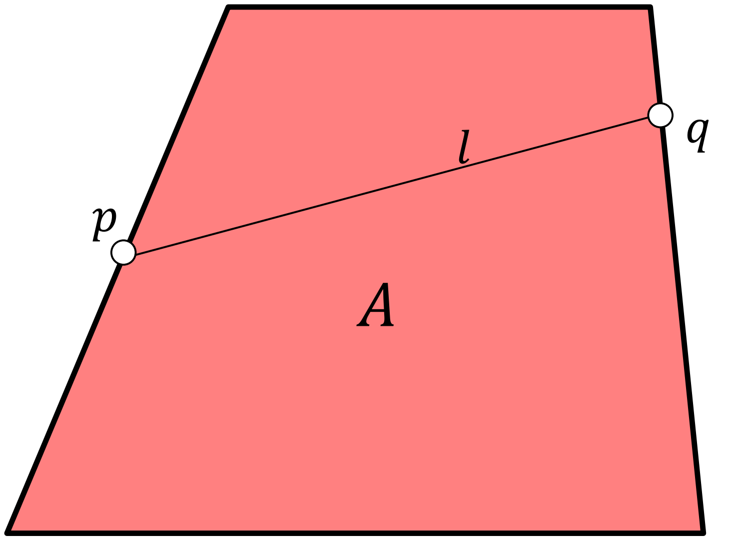


Trapezoidal Decomposition

Approach:

Construct the partition iteratively, adding new line-segments into existing partition:

- Add end-point.
- Add line segment, splitting the trapezoid into 2, 3, or 4 sub-trapezoids.

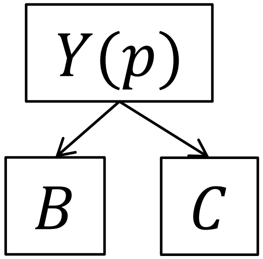


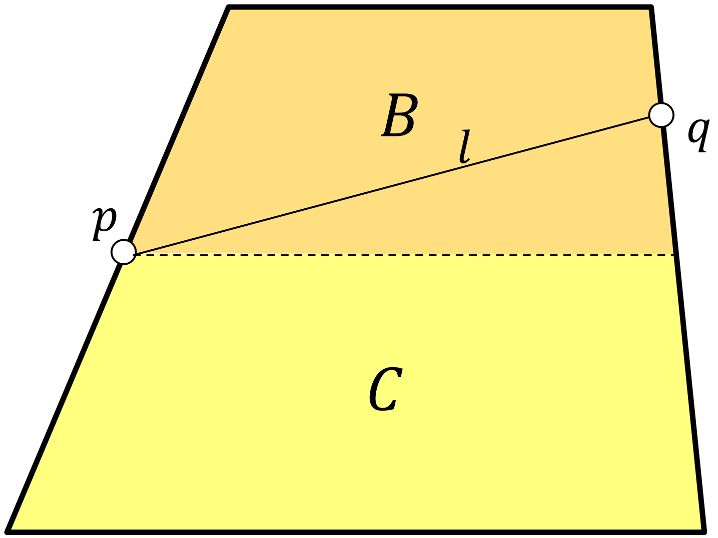
Trapezoidal Decomposition

Approach:

Construct the partition iteratively, adding new line-segments into existing partition:

- Add end-point.
- Add line segment, splitting the trapezoid into 2, 3, or 4 sub-trapezoids.



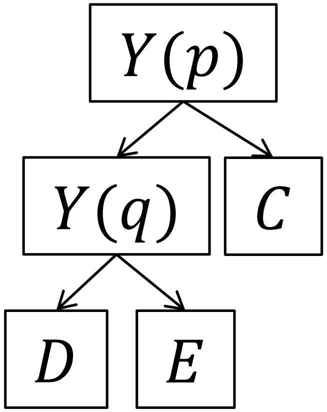


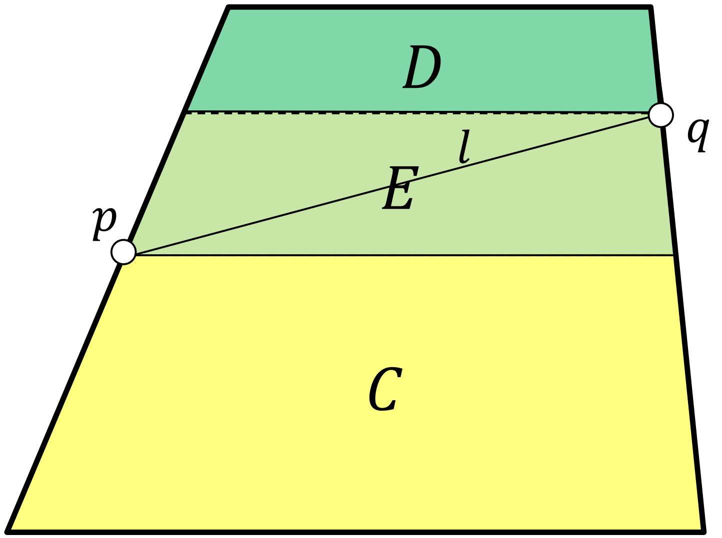
Trapezoidal Decomposition

<u>Approach</u>:

Construct the partition iteratively, adding new line-segments into existing partition:

- Add end-point.
- Add line segment, splitting the trapezoid into 2, 3, or 4 sub-trapezoids.



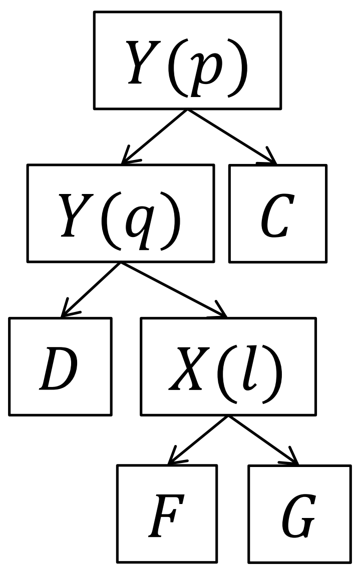


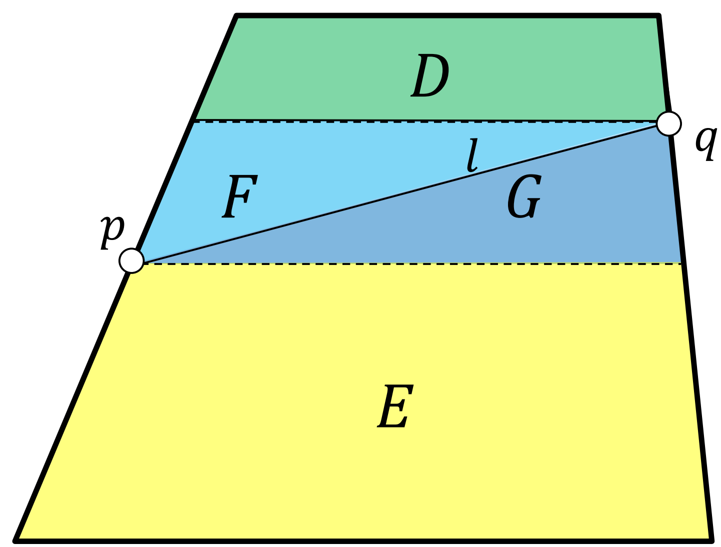
Trapezoidal Decomposition

<u>Approach</u>:

Construct the partition iteratively, adding new line-segments into existing partition:

- Add end-point.
- Add line segment, splitting the trapezoid into 2, 3, or 4 sub-trapezoids.



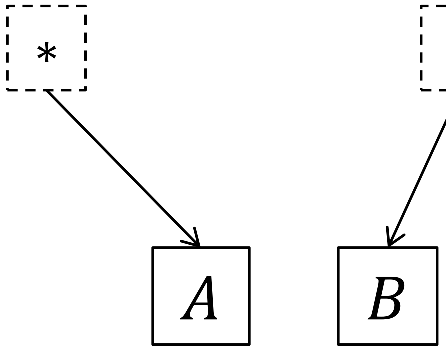


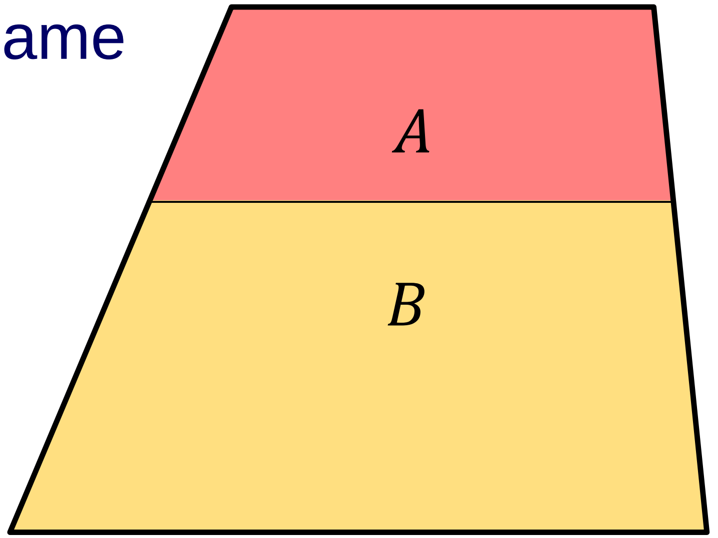
Trapezoidal Decomposition

<u>Approach</u>:

Construct the partition iteratively, adding new line-segments into existing partition:

- Add end-point.
- Add line segment.
- Merge vertically adjacent trapezoids if they are bounded by the same edges (in the same order).



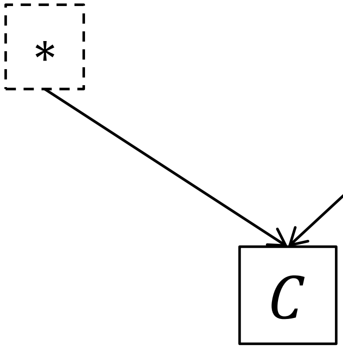


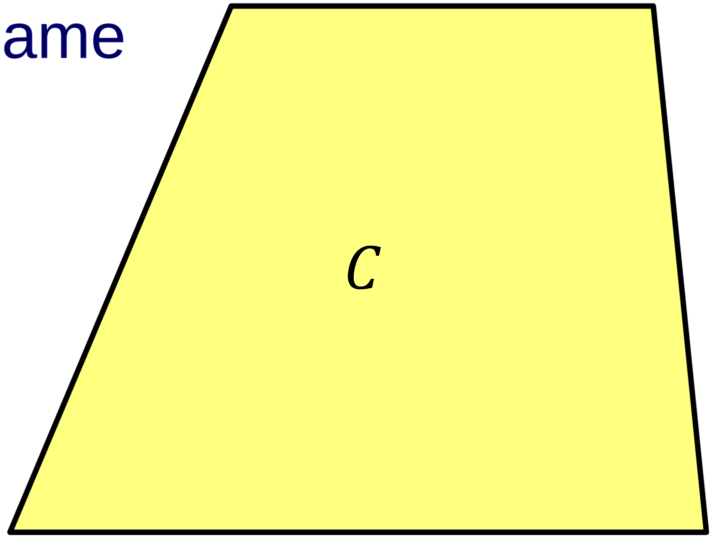
Trapezoidal Decomposition

Approach:

Construct the partition iteratively, adding new line-segments into existing partition:

- Add end-point.
- Add line segment.
- Merge vertically adjacent trapezoids if they are bounded by the same edges (in the same order).





Trapezoidal Decomposition

Approach:

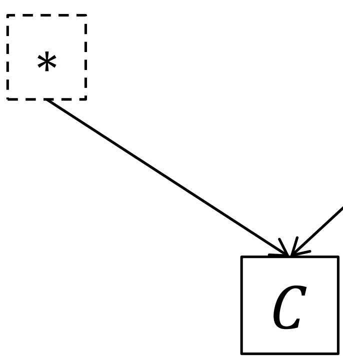
We get a directed binary tree with:

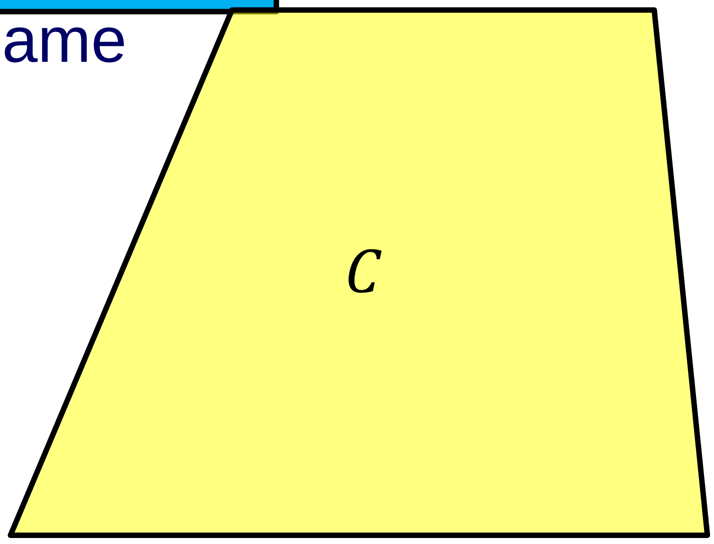
- interior nodes ⇒ left/right or top/bottom partitions
- leaves ⇒ trapezoids

- Add line
- Merge v
  if they are bounded by the same edges (in the same order).

Left child: above/left

Right child: below/right

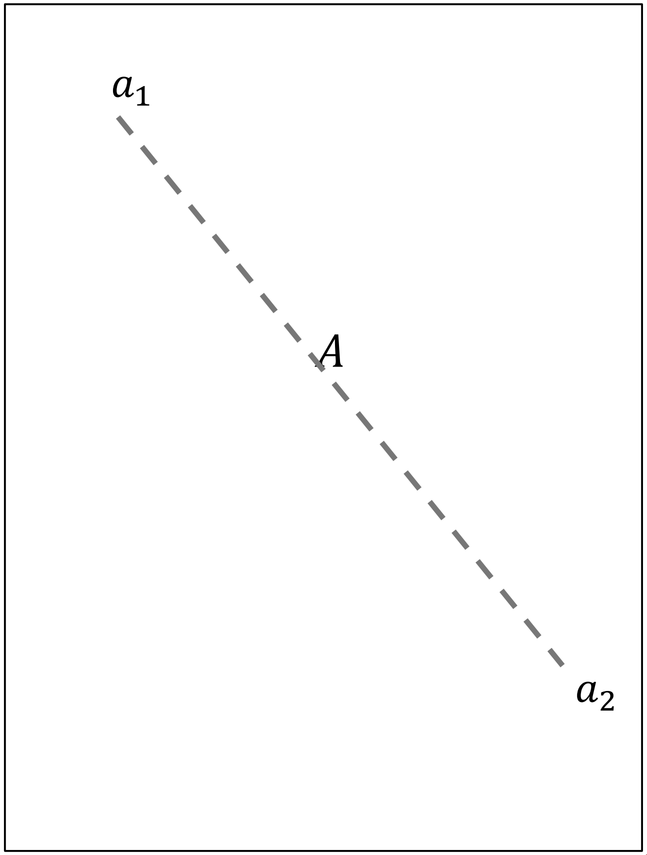




Trapezoidal Decomposition

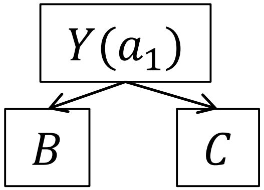
- Add end-point
- Add line segment
- Merge adjacent trapezoids

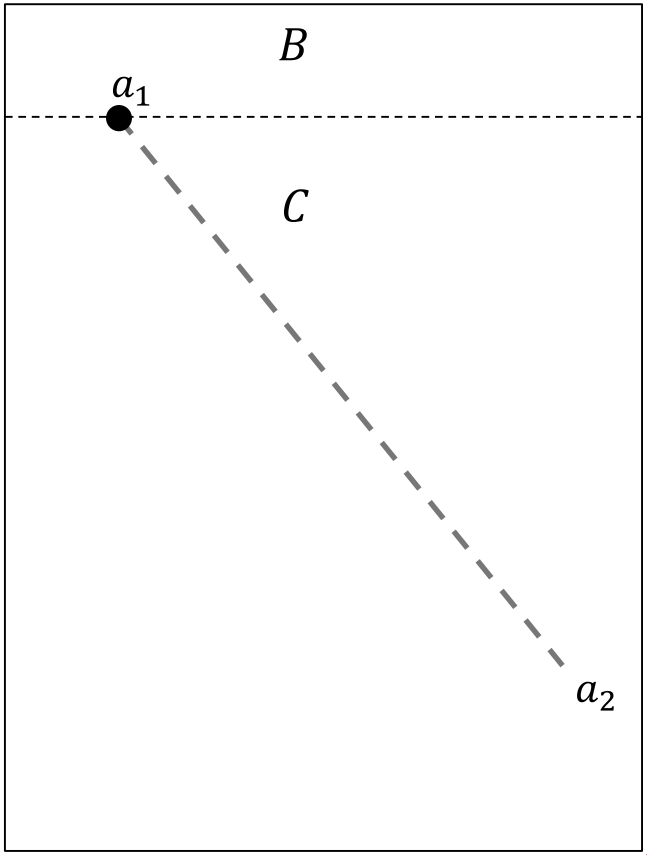
$\boxed{A}$



Trapezoidal Decomposition

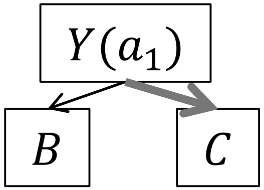
- **Add end-point**
- Add line segment
- Merge adjacent trapezoids

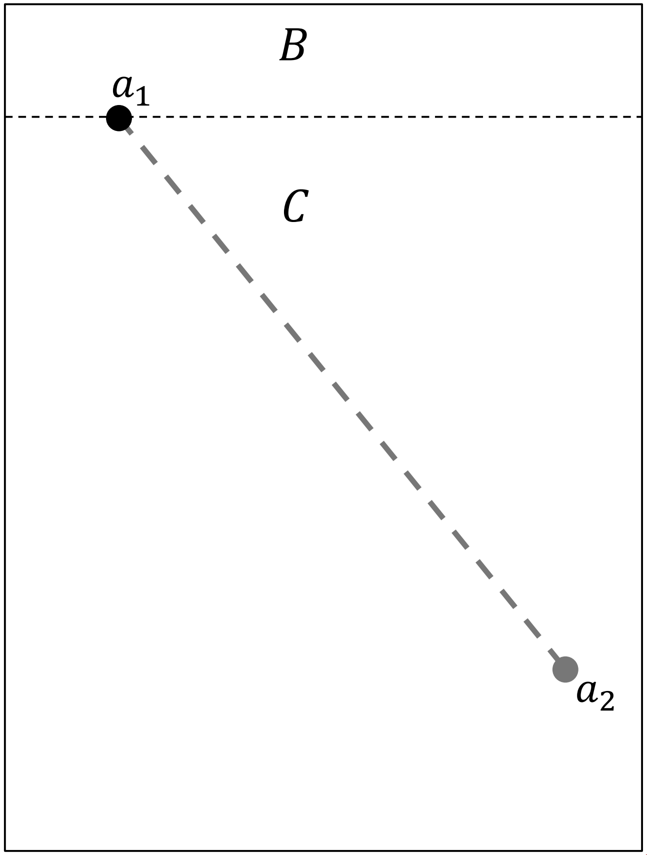




Trapezoidal Decomposition

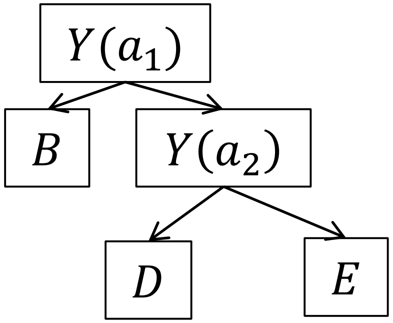
- **Add end-point**
- Add line segment
- Merge adjacent trapezoids

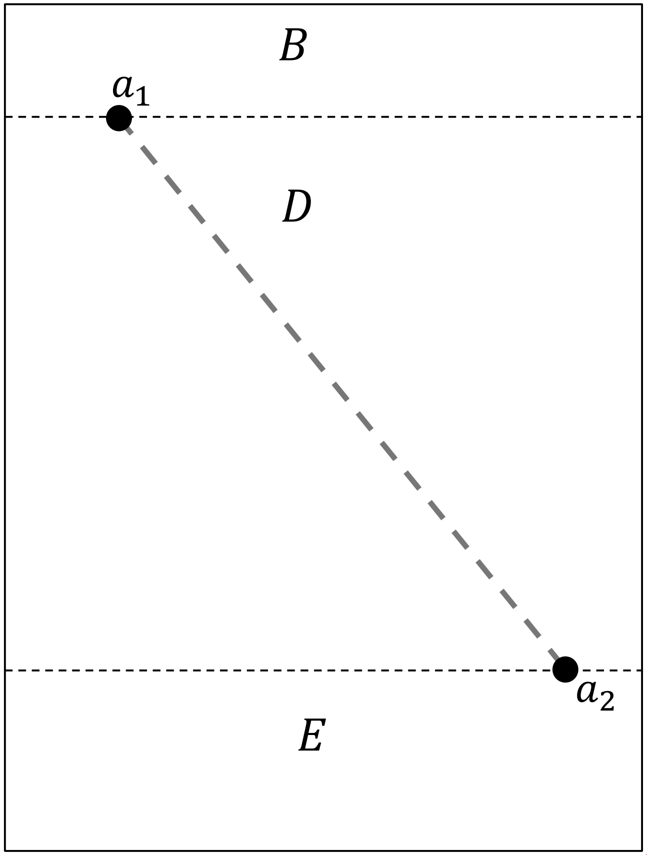




Trapezoidal Decomposition

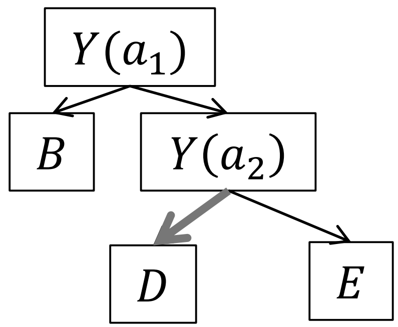
- **Add end-point**
- Add line segment
- Merge adjacent trapezoids

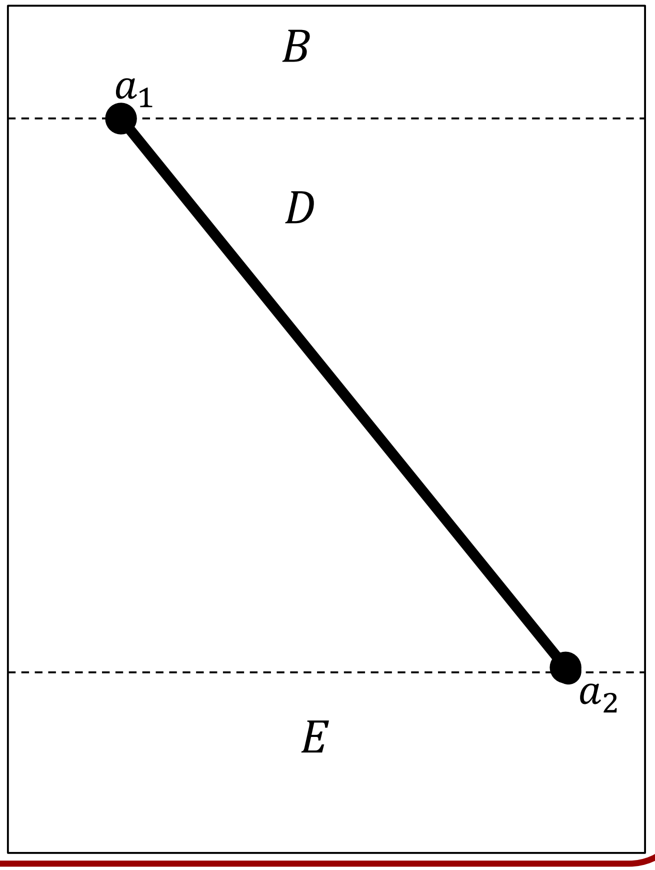




Trapezoidal Decomposition

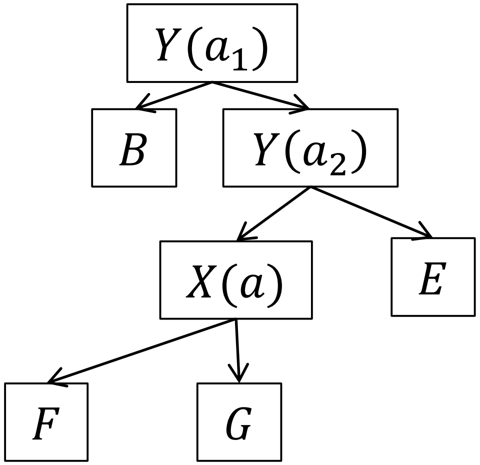
- Add end-point
- **Add line segment (2)**
- Merge adjacent trapezoids

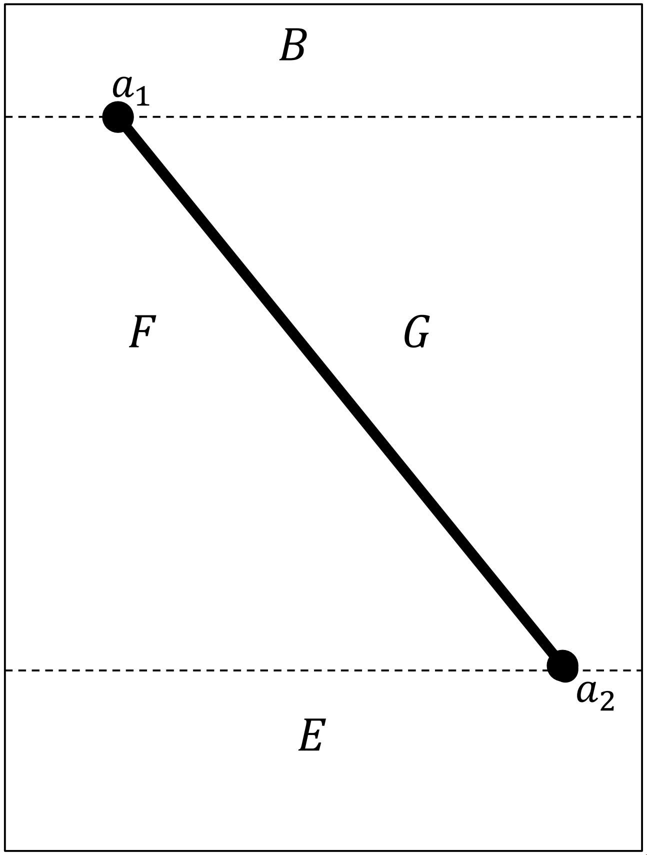




Trapezoidal Decomposition

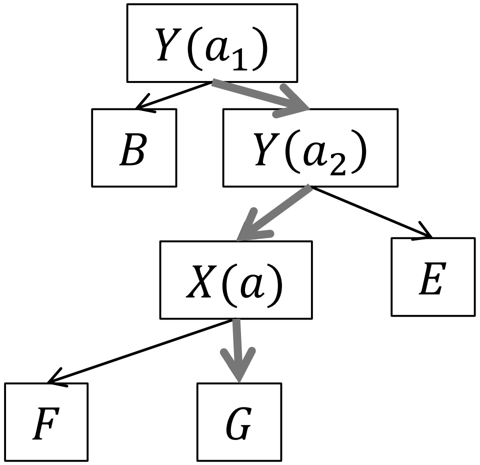
- Add end-point
- **Add line segment (2)**
- Merge adjacent trapezoids

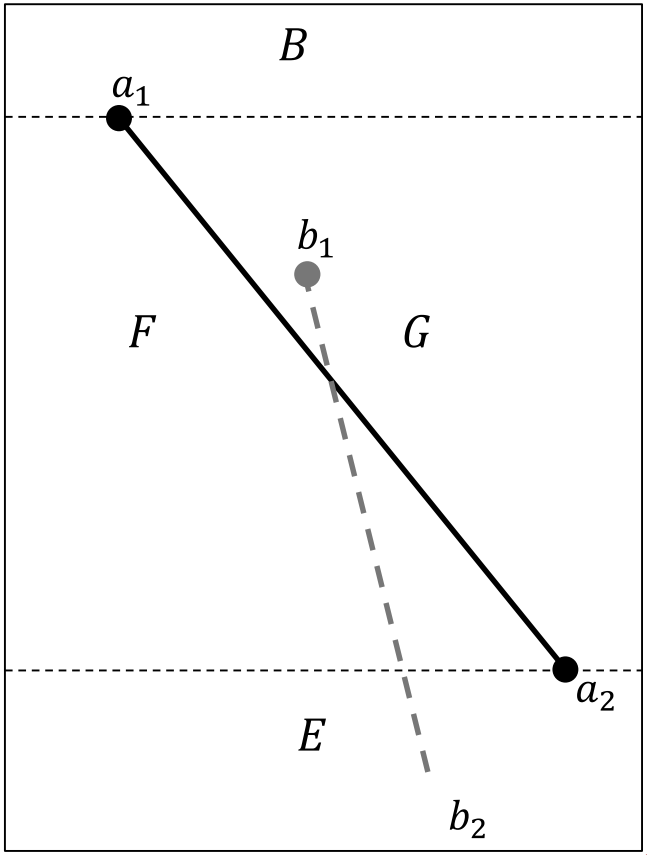




Trapezoidal Decomposition

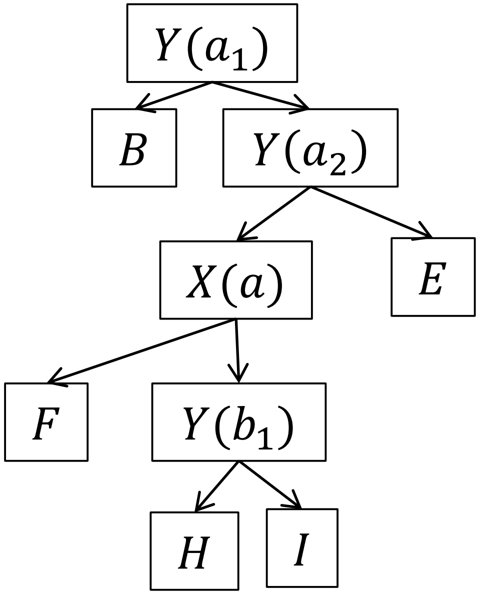
- **Add end-point**
- Add line segment
- Merge adjacent trapezoids

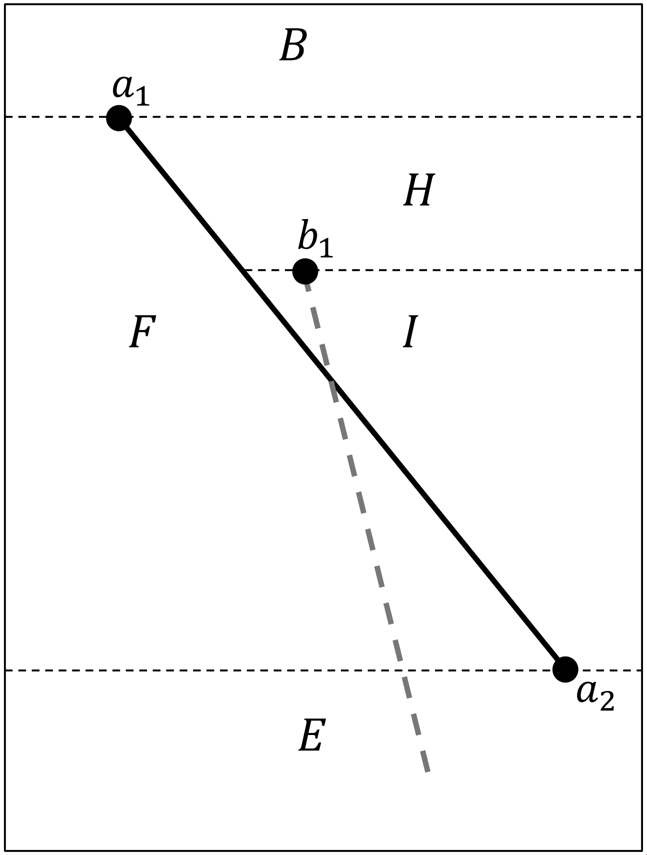




Trapezoidal Decomposition

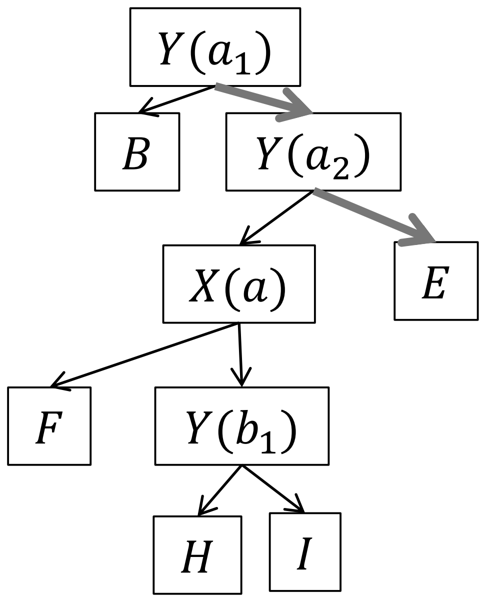
- **Add end-point**
- Add line segment
- Merge adjacent trapezoids

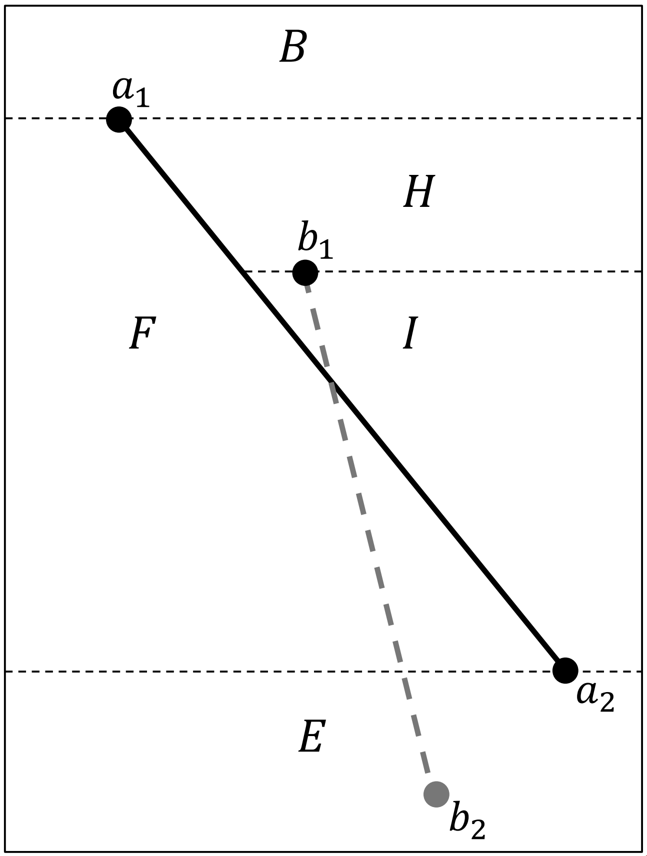




Trapezoidal Decomposition

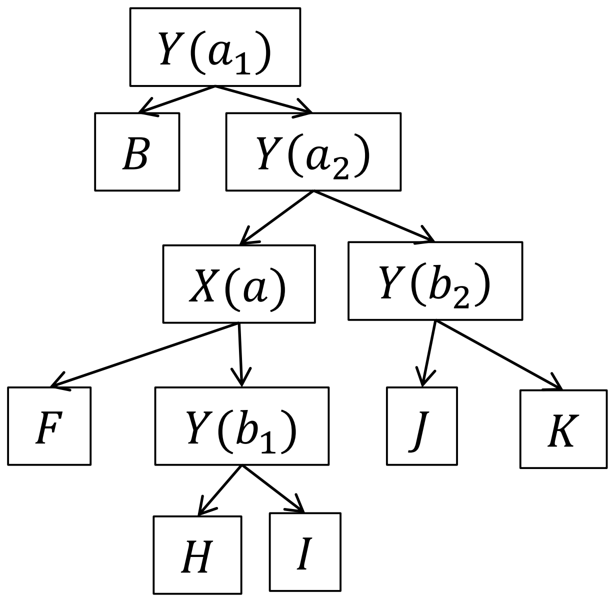
- **Add end-point**
- Add line segment
- Merge adjacent trapezoids

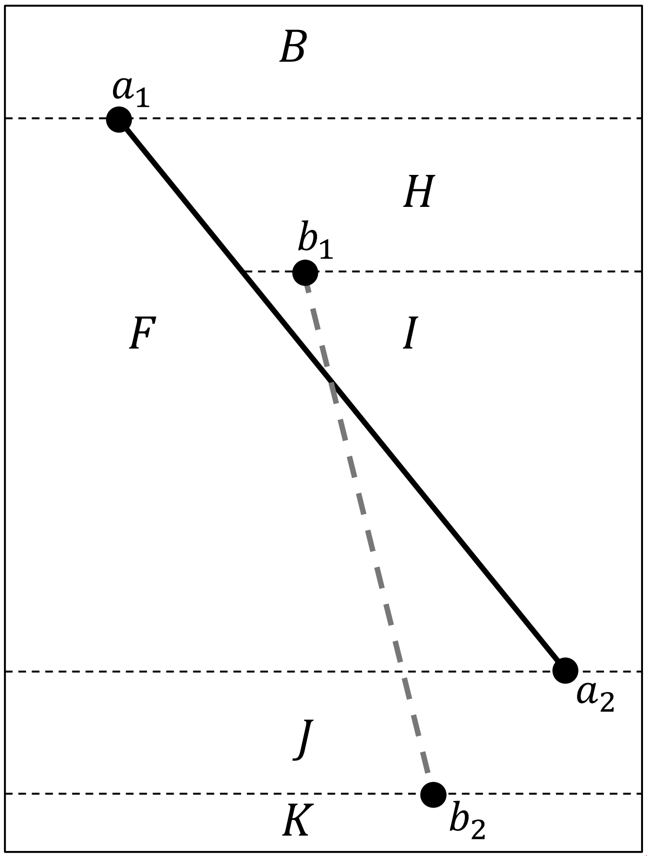




Trapezoidal Decomposition

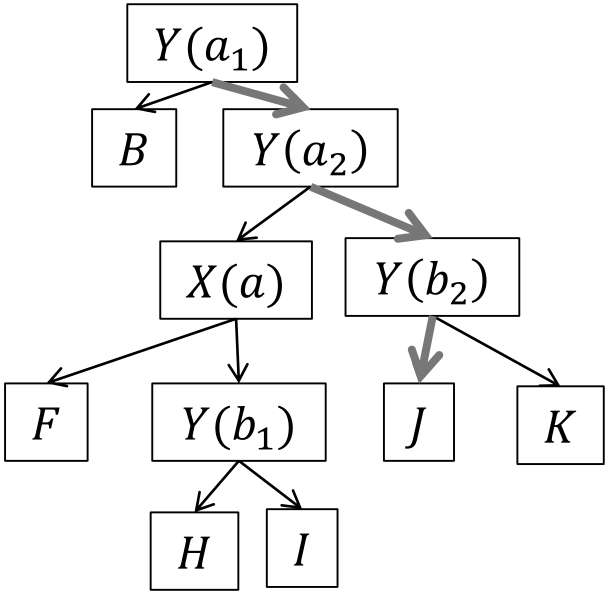
- **Add end-point**
- Add line segment
- Merge adjacent trapezoids

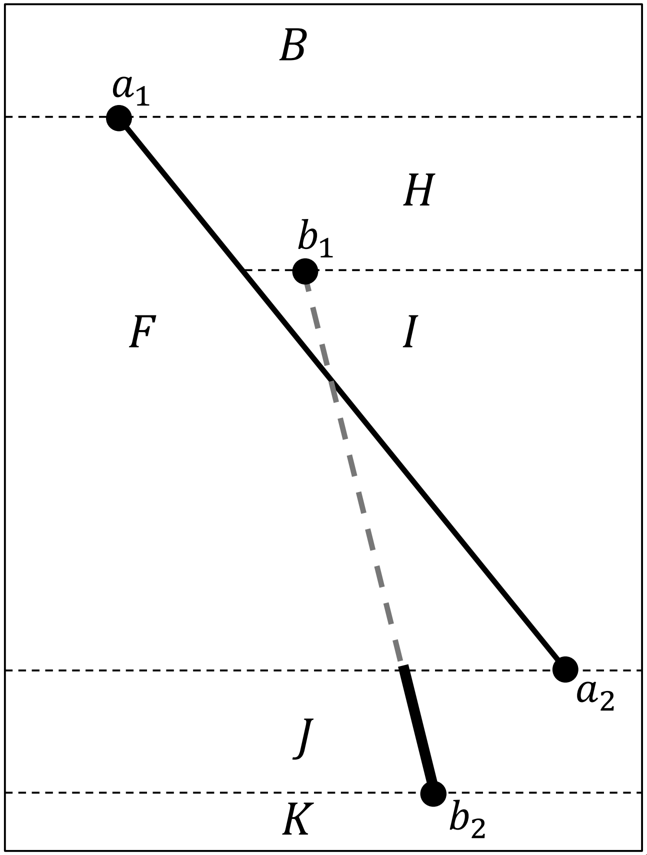




Trapezoidal Decomposition

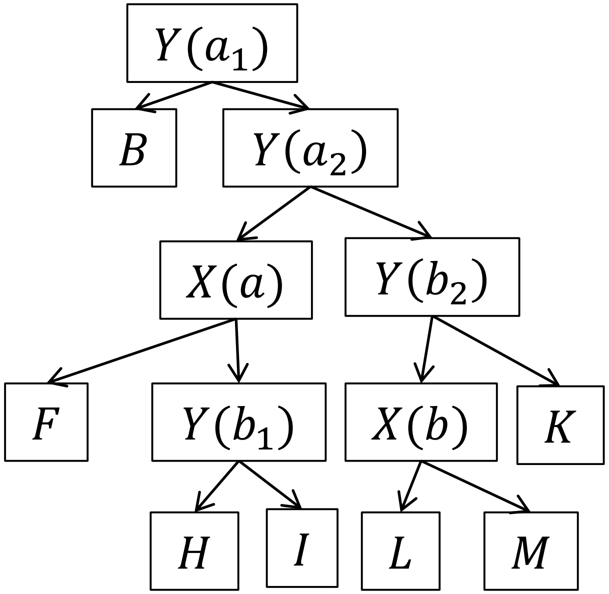
- Add end-point
- **Add line segment (2)**
- Merge adjacent trapezoids

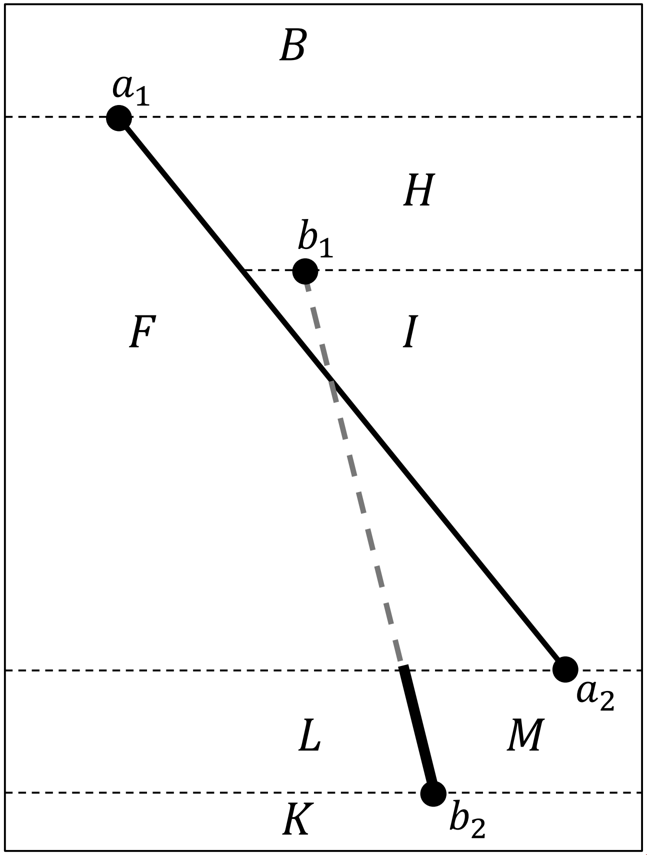




Trapezoidal Decomposition

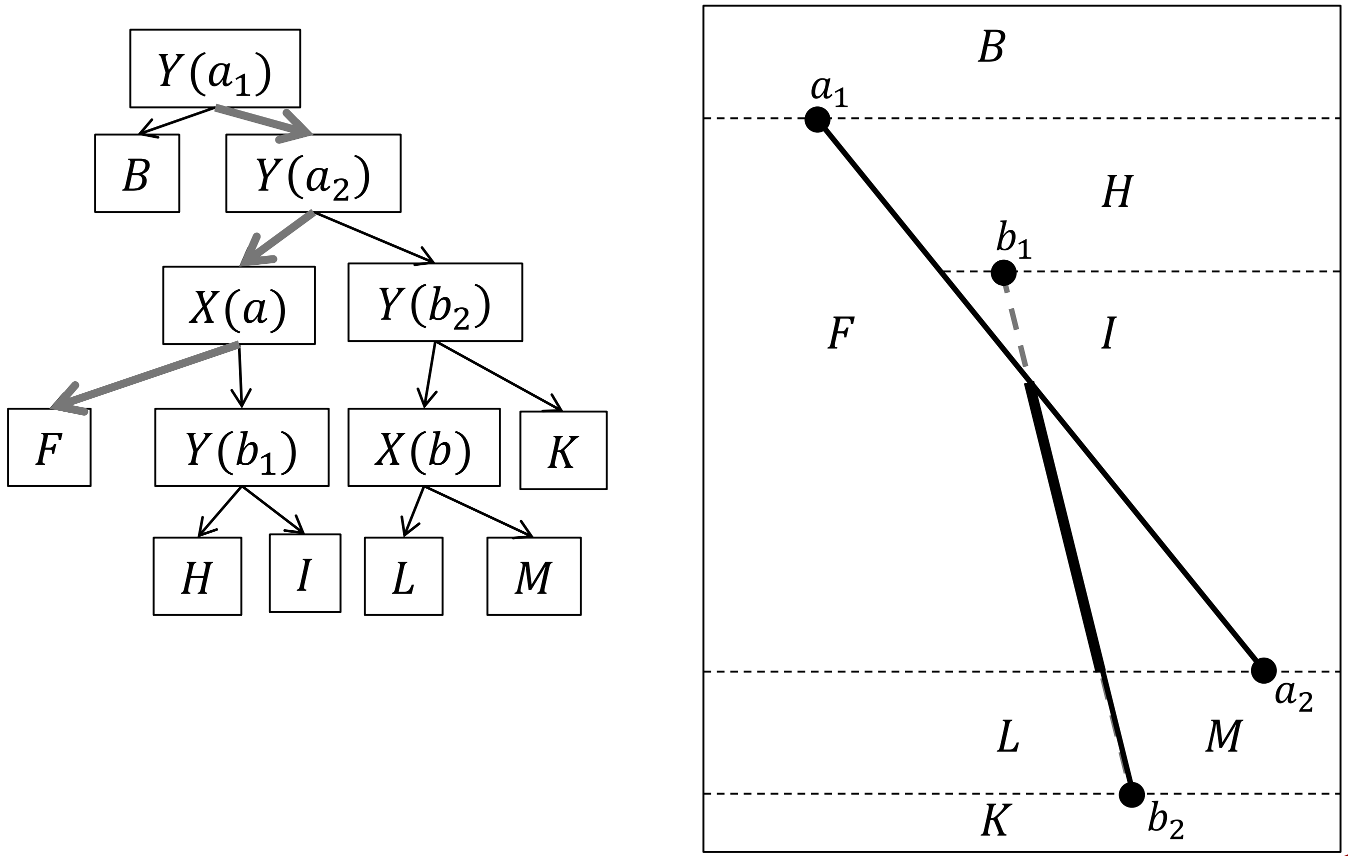
- Add end-point
- **Add line segment (2)**
- Merge adjacent trapezoids





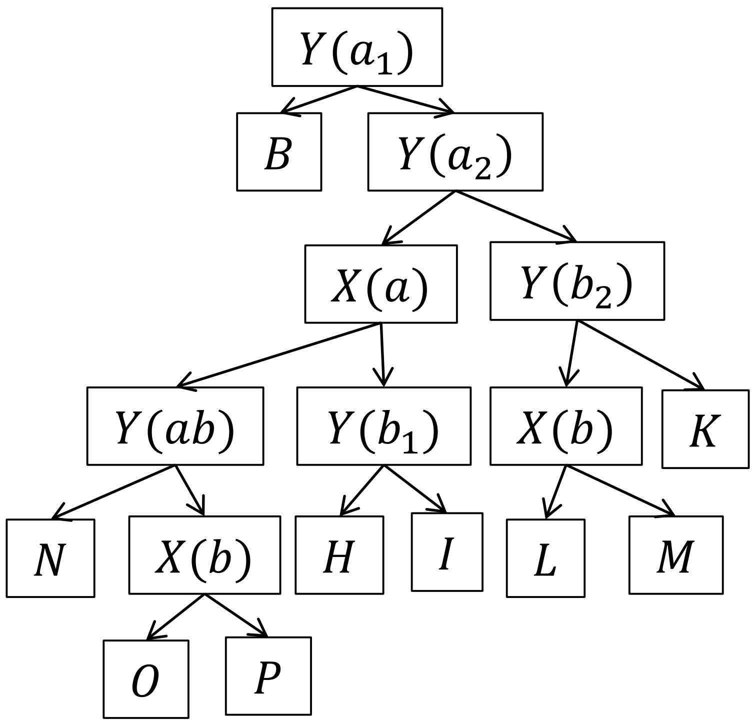
Trapezoidal Decomposition

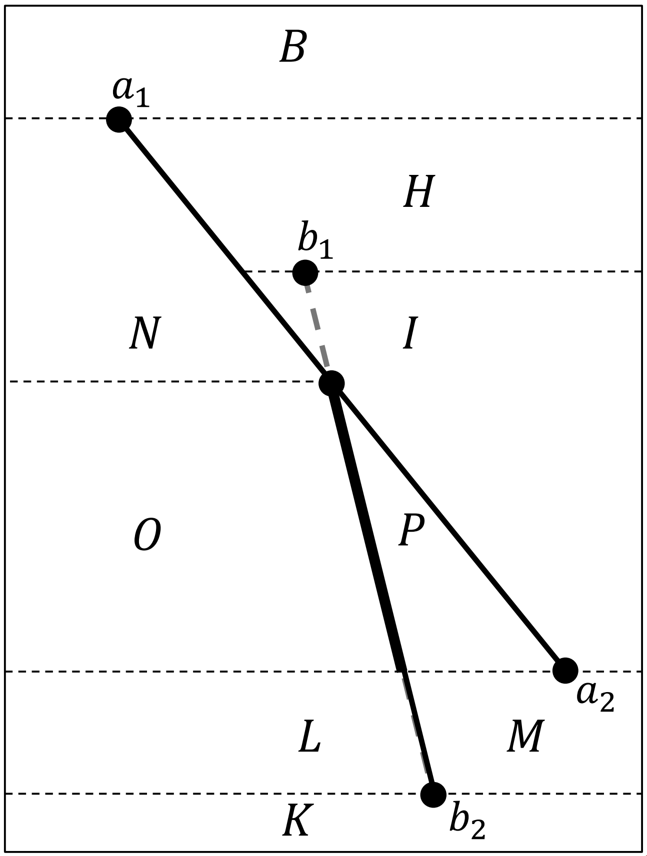
- Add end-point
- **Add line segment (3)**
- Merge adjacent trapezoids



Trapezoidal Decomposition

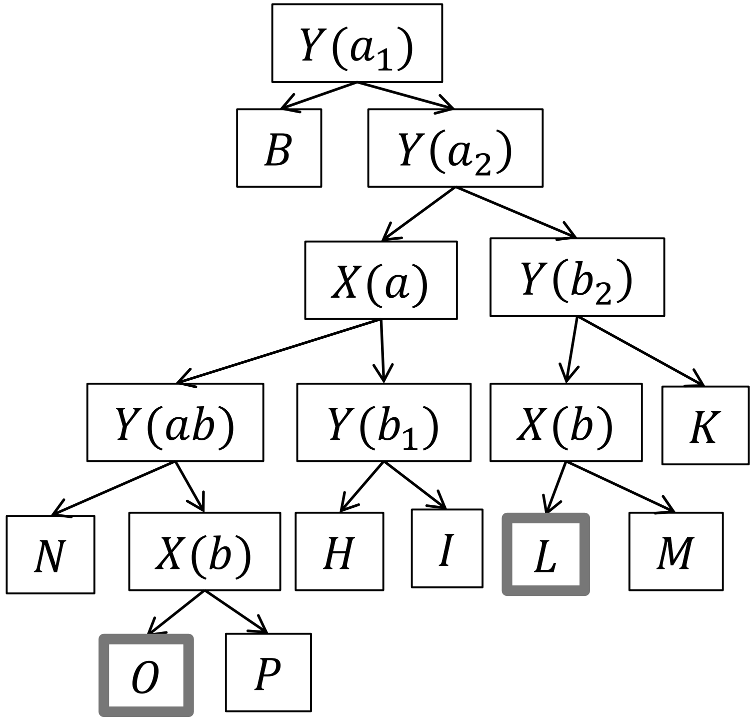
- Add end-point
- **Add line segment (3)**
- Merge adjacent trapezoids

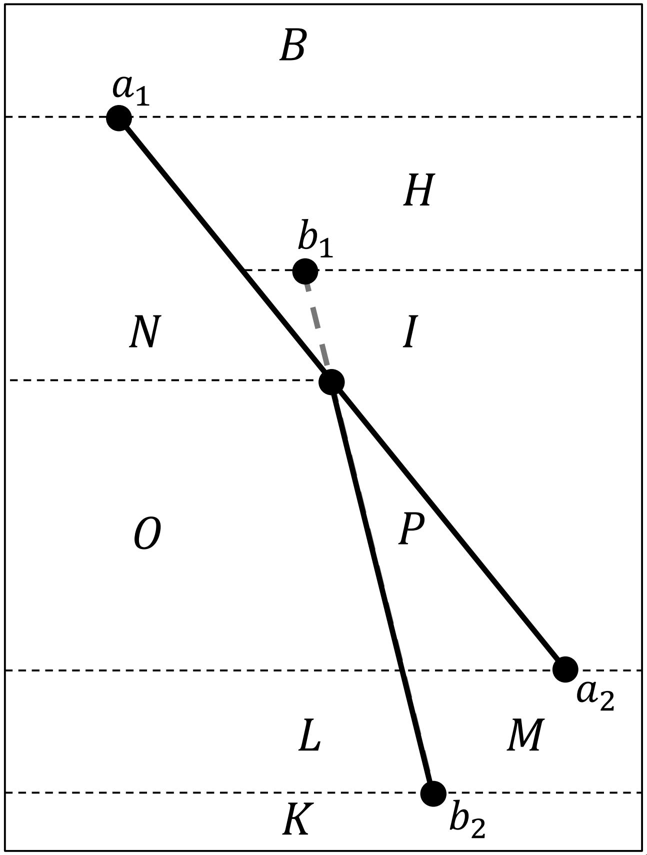




Trapezoidal Decomposition

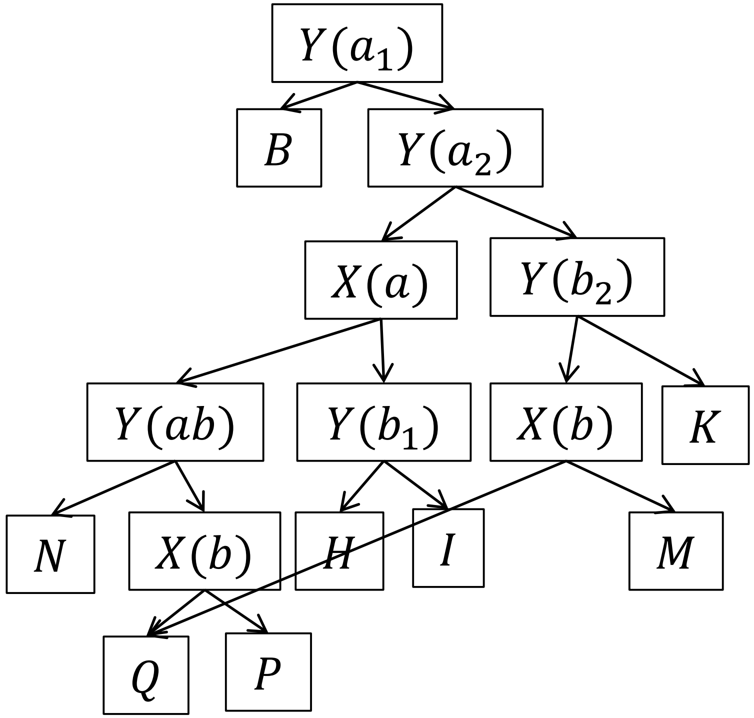
- Add end-point
- Add line segment
- **Merge adjacent trapezoids**

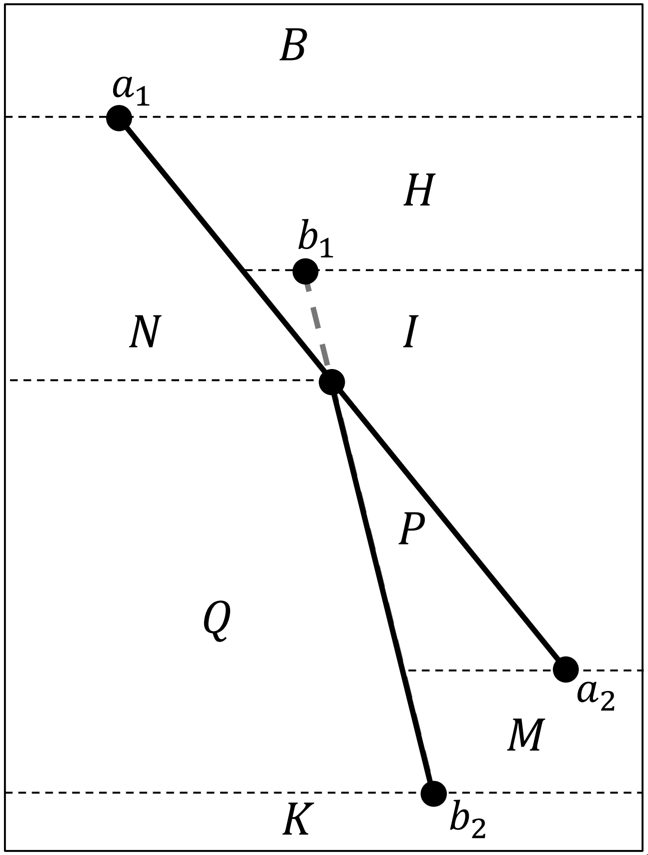




Trapezoidal Decomposition

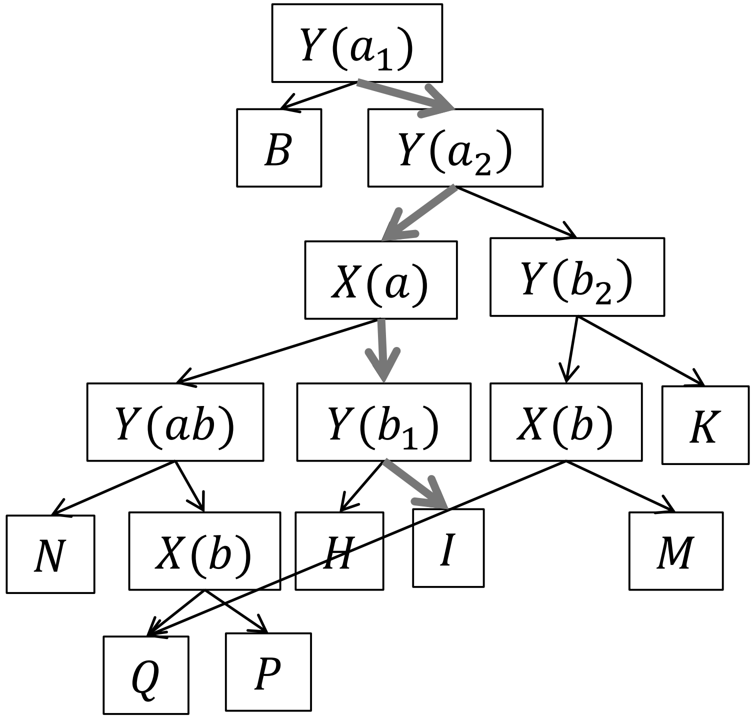
- Add end-point
- Add line segment
- **Merge adjacent trapezoids**

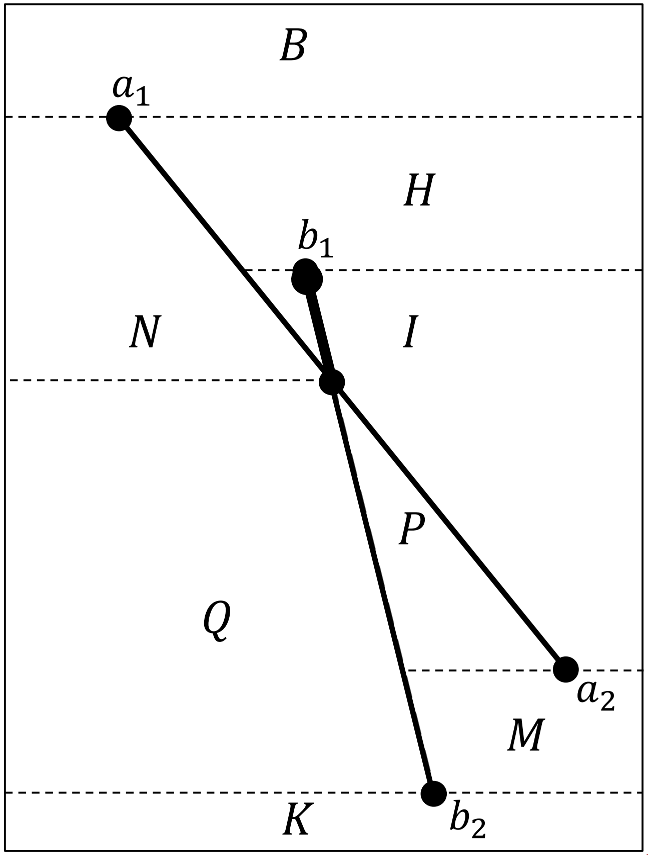




Trapezoidal Decomposition

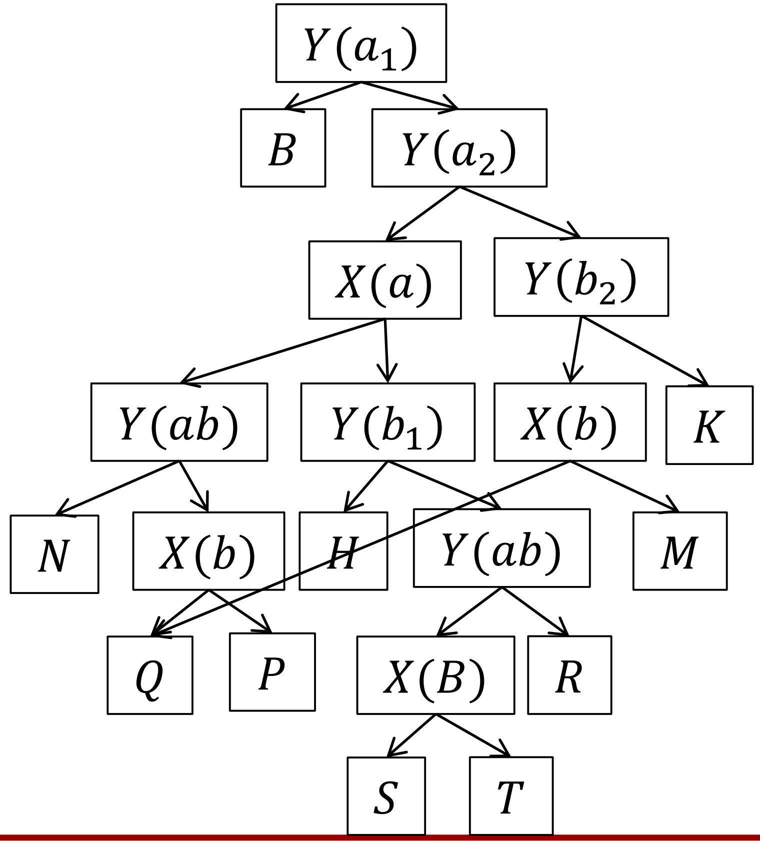
- Add end-point
- **Add line segment (3)**
- Merge adjacent trapezoids

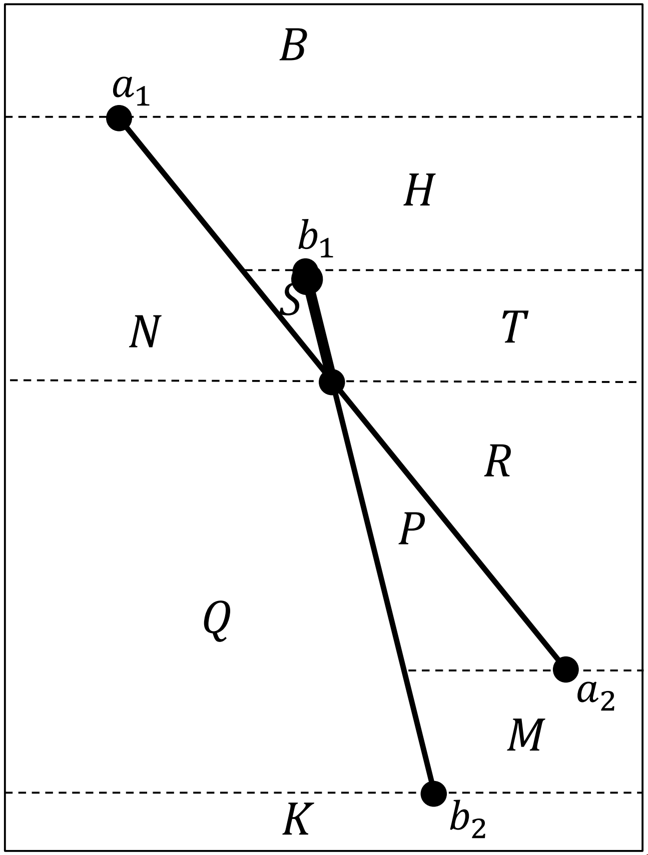




Trapezoidal Decomposition

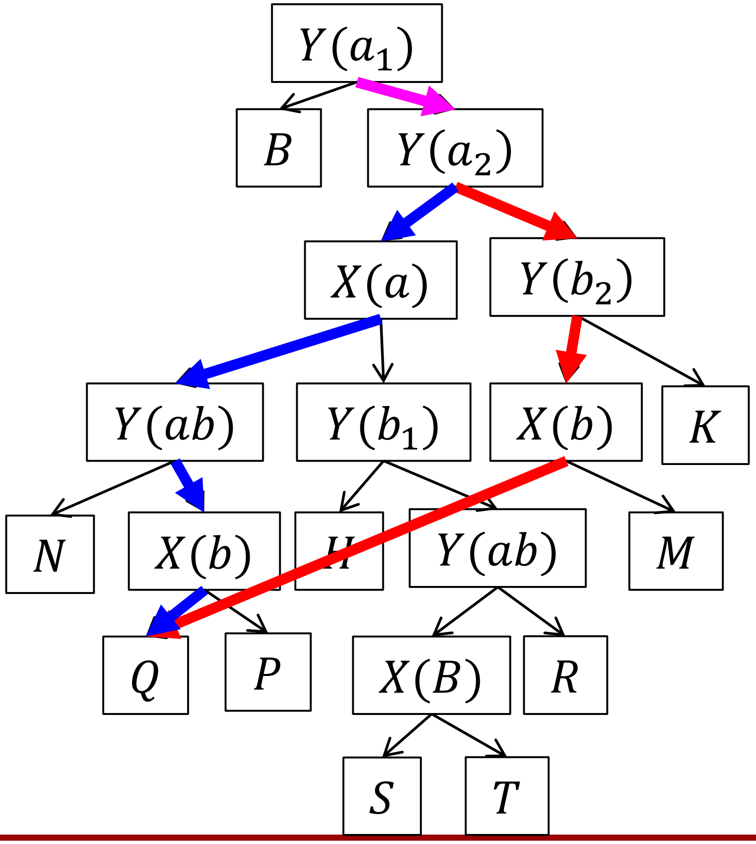
- Add end-point
- **Add line segment (3)**
- Merge adjacent trapezoids

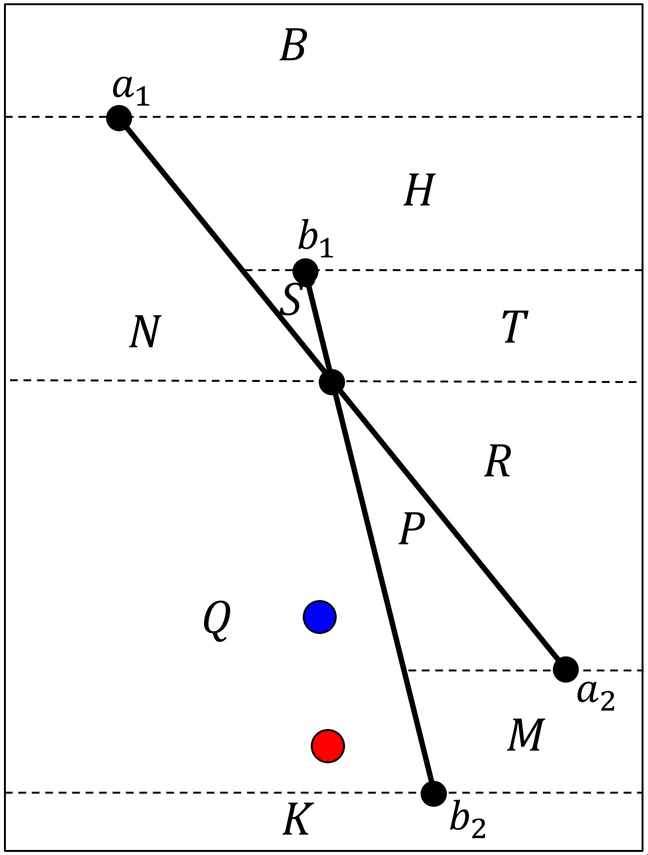




Trapezoidal Decomposition

Because of merging, we can have multiple paths into the same trapezoid.



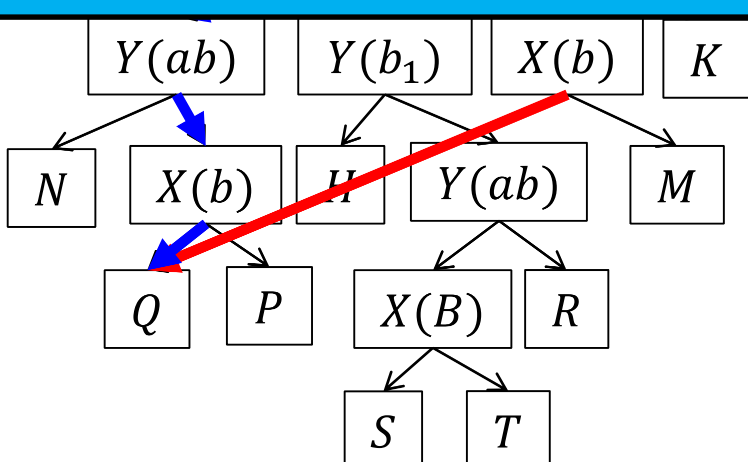


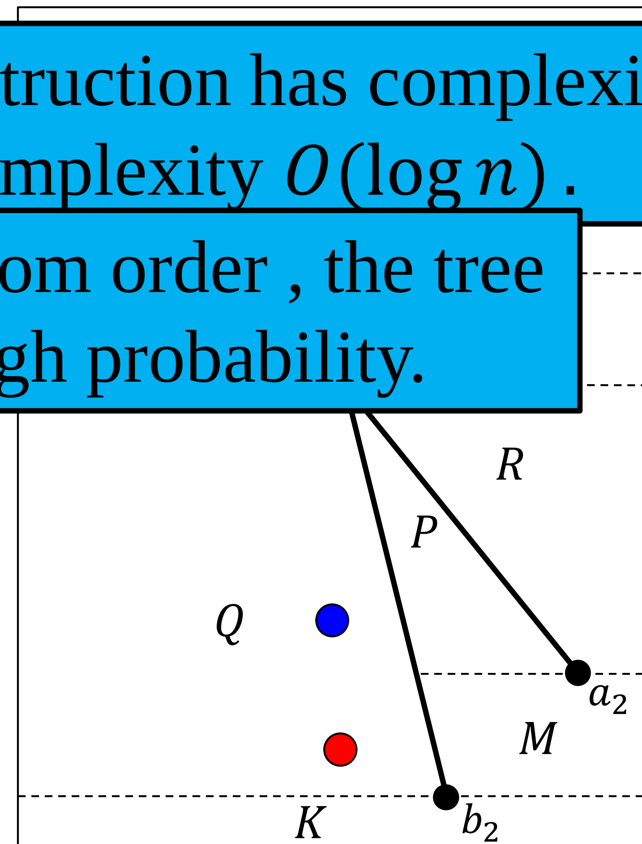
Trapezoidal Decomposition

Because of merging, we can have multiple paths into the same trapezoid.

Assuming the tree stays balanced, construction has complexity $O((n+k)\log n)$ and query has complexity $O(\log n)$.

If line segments are added in random order, the tree will be well-balanced, with high probability.





Outline

- Trapezoidal Decomposition
- Extreme Points (2D)
- Extreme Points (3D)

Extreme Points

Linear Programming:

Given a set of linear constraints:

$$C_i = \{p \mid \langle p, n_i \rangle \geq d_i\}$$

and a linear energy function:

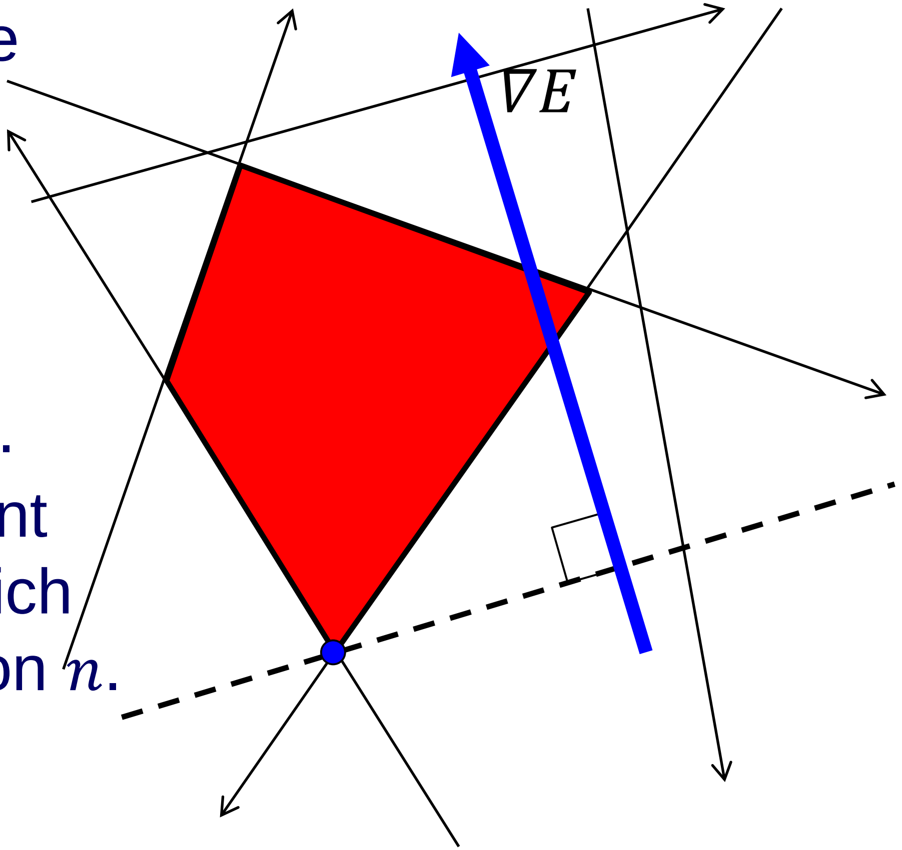
$$E(p) = \langle p, n \rangle + d$$

we would like to find the point p that satisfies the constraints and minimizes the energy.

Extreme Points

Linear Programming:

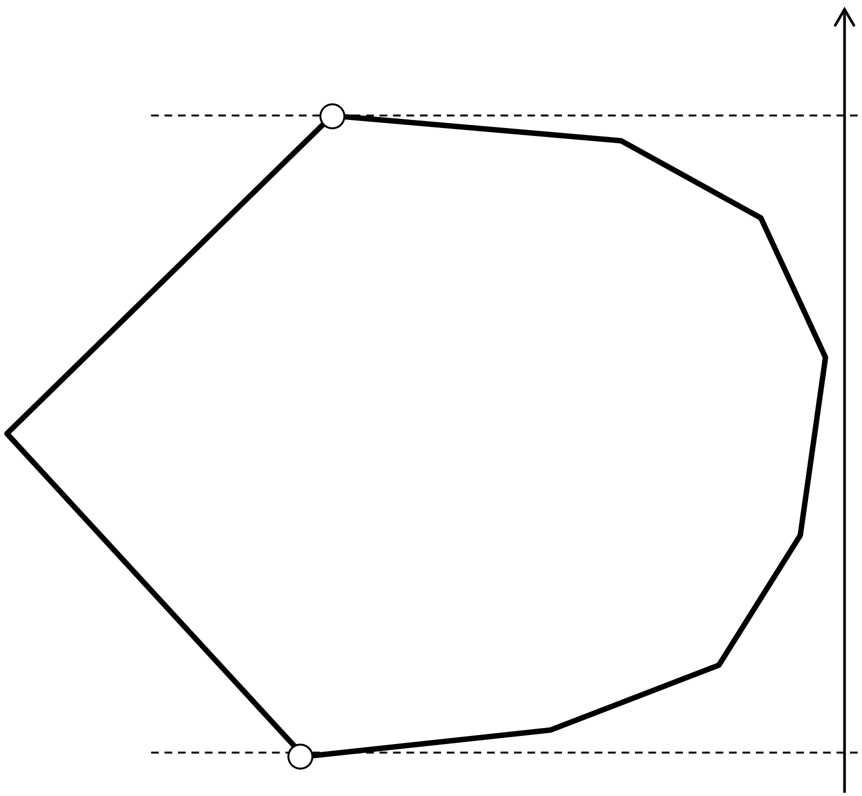
- Since the constraints are linear, each one defines a half-space of valid solutions.
- The intersection of these half-spaces is convex.
- Since the energy is linear, it has a constant gradient ∇E pointing away from the minimum.
- The minimizer is the point in the convex region which is extreme along direction n.



Extreme Points (2D)

Given a convex polygon P, we would like to find the extreme points along a particular direction.

Without loss of generality, we can assume that the direction is vertical.



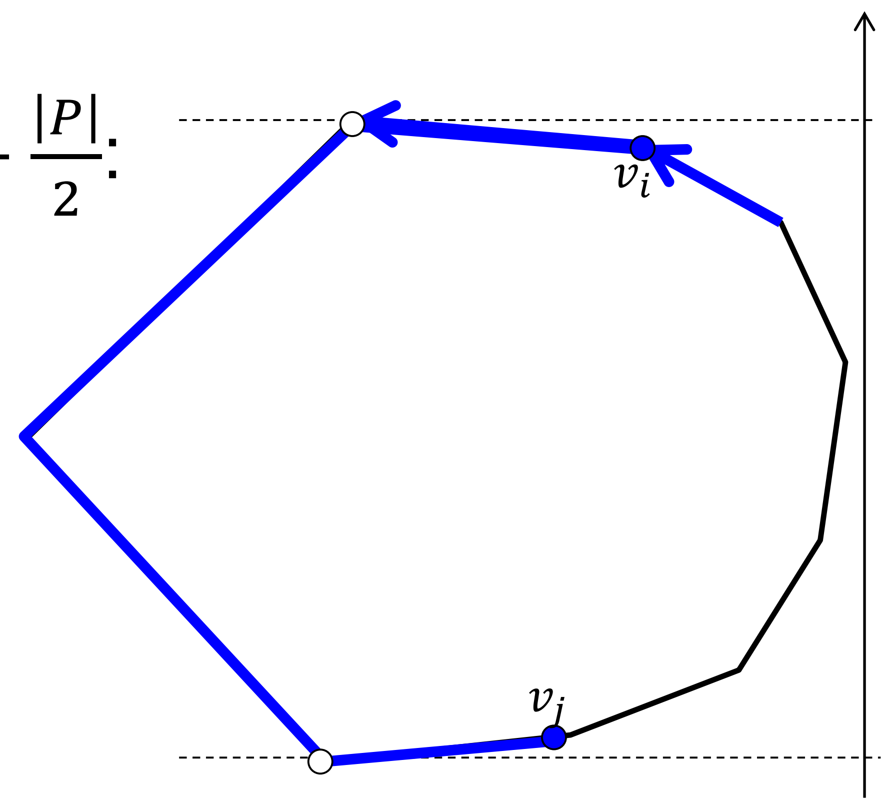
Extreme Points (2D)

Pick a vertex v_i at random:

- Both edges rising ⇒ Right
- Both edges falling ⇒ Left
- Otherwise, extremal

Consider vertex v_j, $j = i + \frac{|P|}{2}$:

- If v_i is left and v_j right:
  ⇒ max ∈ $[j, i]$ min ∈ $[i, j]$
- If v_i right and v_j left: …
- If v_i and v_j right:
  - » If v_i above v_j:
    - – Both extrema in $[j, i]$
  - » …



Extreme Points (2D)

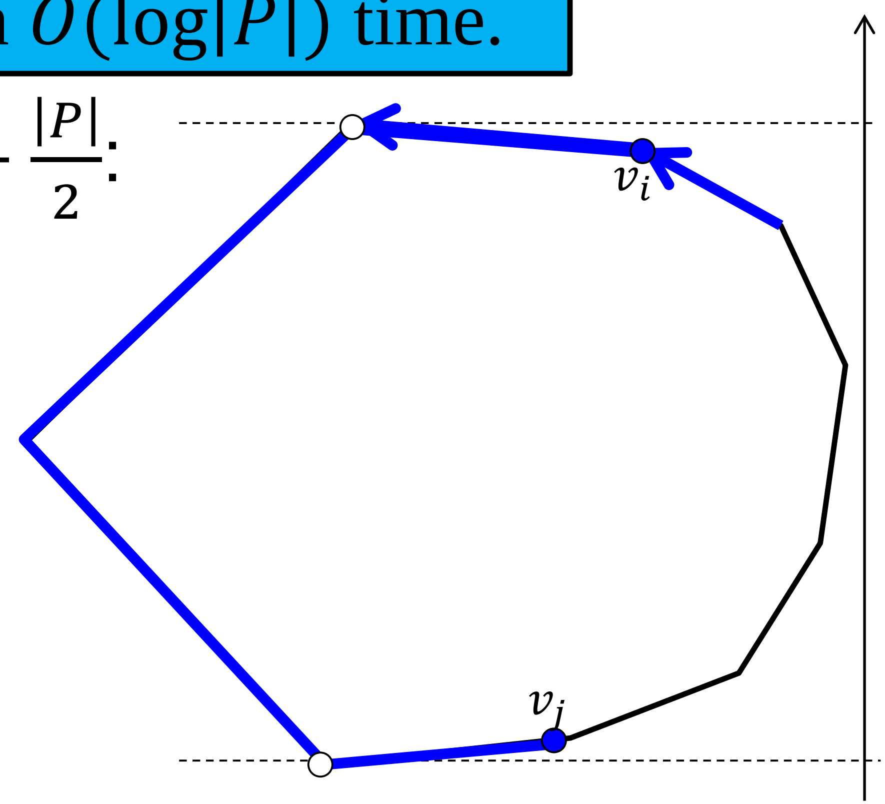
Pick a vertex v_i at random:

- Both edges rising ⇒ Right
- Bot
- Oth

With repeated bisection, we can find the two extrema in $O(\log|P|)$ time.

Consider vertex v_j, $j = i + \frac{|P|}{2}$:

- If v_i is left and v_j right:
  ⇒ max ∈ $[j, i]$ min ∈ $[i, j]$
- If v_i right and v_j left: …
- If v_i and v_j right:
  - » If v_i above v_j:
    - – Both extrema in $[j, i]$
  - » …



Outline

- Trapezoidal Decomposition
- Extreme Points (2D)
- Extreme Points (3D)

Extreme Points (3D)

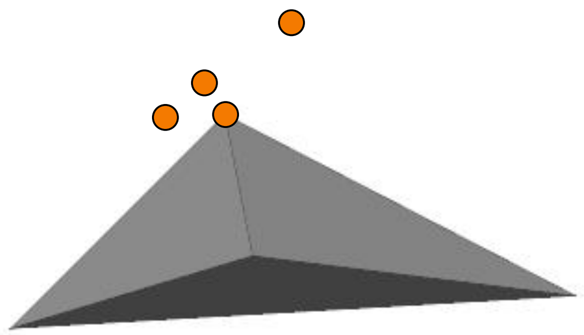
Given a convex polyhedron P, we would like to find the (without loss of generality) highest point.

[Kirkpatrick, 1983]

Compute a hierarchy of nested polytopes, compute the highest point on the coarsest polytopes and use that to efficiently compute the highest point on the next polytope.

Extreme Points (3D)

Compute a hierarchy of nested polytopes, $\{P_0 = P \supset$



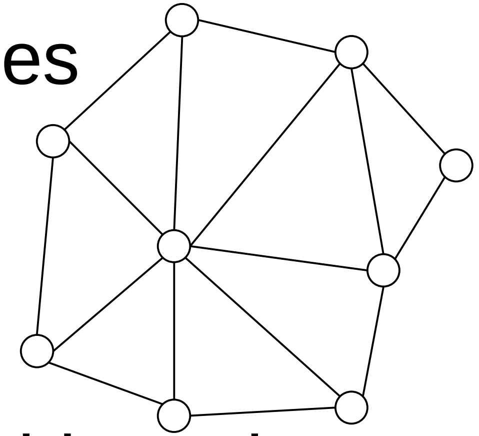
Constructing the Hierarchy

Definition:

Given a graph, a set of vertices is said to be *independent* if there is no edge in the graph that connects vertices in the set.

Key Idea:

Identify an independent set of vertices on P_k with low degree, remove those, and set P_{k+1} to the convex hull of what's left.



Repeat for subsequent levels of the hierarchy.

Constructing the Hierarchy

Greedy Algorithm:

- While not done
  - Find a vertex with degree ≤ 8.
  - If none of its neighbors have been marked as independent, mark it as independent.

Claim:

This algorithm will mark a at least 1/18 of the vertices as independent.

Constructing the Hierarchy

Proof:

By Euler's formula, for a triangulated polyhedron:

$$E = 3V - 6.$$

⇒ The sum of the degrees of the vertices, Σ, is equal to twice the number of edges, and hence:

$$\Sigma = 6V - 12.$$

⇒ There are at least $V/2$ vertices with degree ≤ 8. Otherwise, there are at least $V/2$ vertices with degree ≥ 9 and the rest have degree at least 3:

$$\Sigma \geq \frac{9V}{2} + \frac{3V}{2} = 6V > 6V - 12 = \Sigma \qquad (\Rightarrow\Leftarrow)$$

Constructing the Hierarchy

<u>Proof</u>:

There are at least $V/2$ vertices with degree ≤ 8.

$\Rightarrow$ Marking a low-degree vertex as independent, we invalidate (at most) 8 other vertices.

$\Rightarrow$ Repeating, we will mark at least 1/9 of the low-degree vertices as independent

$\Rightarrow$ 1/18 of all vertices will be independent.

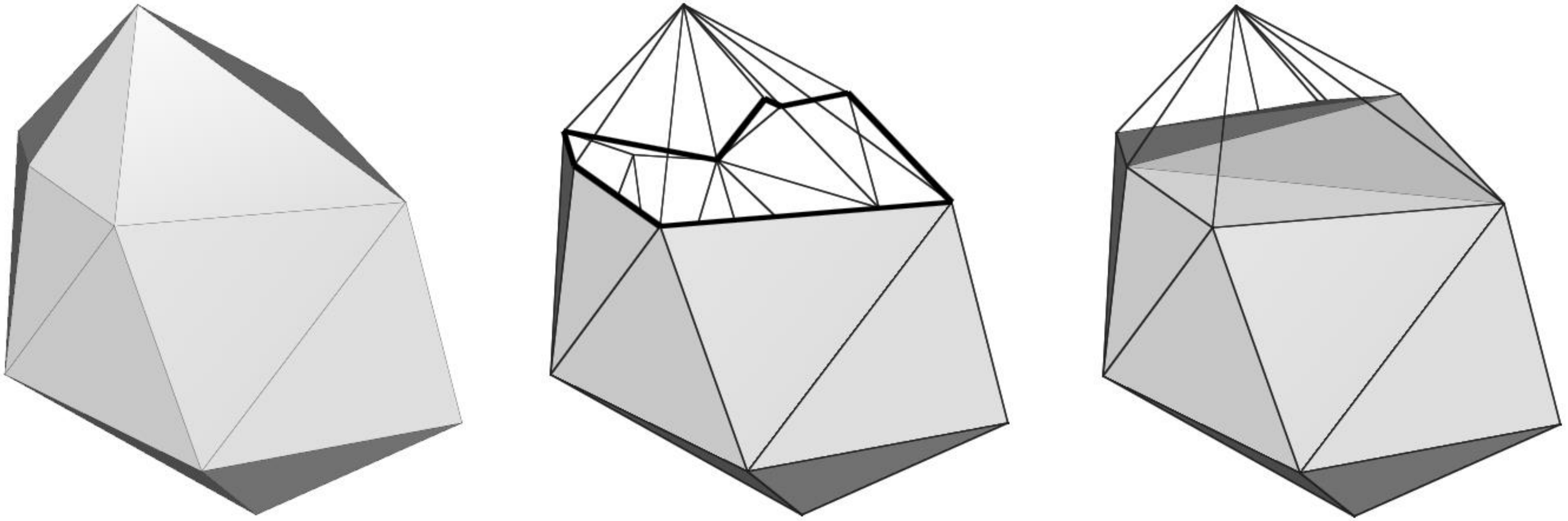
Using this to construct our polytope hierarchy:

- We will have $O(\log |P|)$ levels.
- We will require $O(|P|)$ storage.

Constructing the Hierarchy

Claim:

If we remove a point on a polytope, the convex hull of the remaining points can be obtained by computing the convex hull of the points on the boundary and using the “outer” half of the triangles.



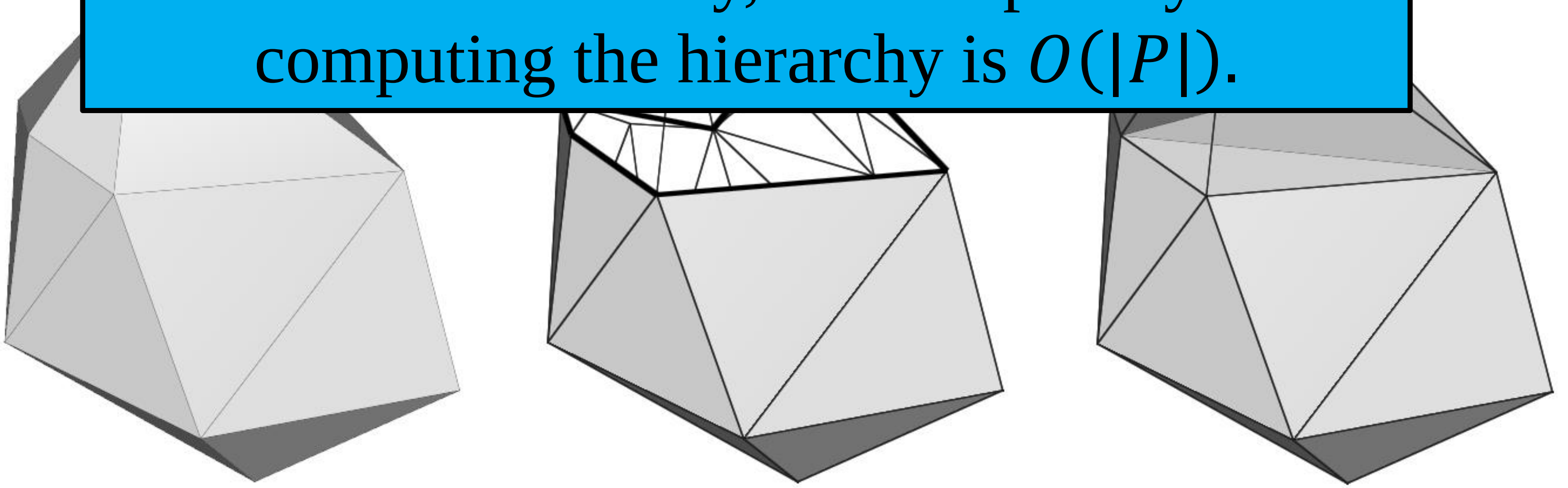
Constructing the Hierarchy

Claim:

Since the removed vertices are independent and have degree ≤ 8, the coarser convex hull can be computed in time proportional to the number of removed points.

bou gles.

Since a removed vertex does not appear later on in the hierarchy, the complexity of computing the hierarchy is $O(|P|)$.

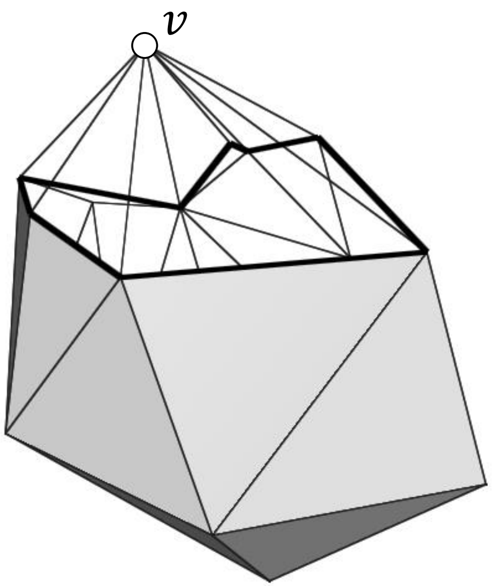


Constructing the Hierarchy

<u>Claim</u>:

After remove the highest vertex $v \in P$, the next highest vertex w has to be in the one-ring of v.

<u>Proof</u>: (by contradiction)



Assume w is interior.

$\Rightarrow$ The closed loop of neighbors of w are below w.

$\Rightarrow$ P must be below the cone apexed at w and going through its neighbors.

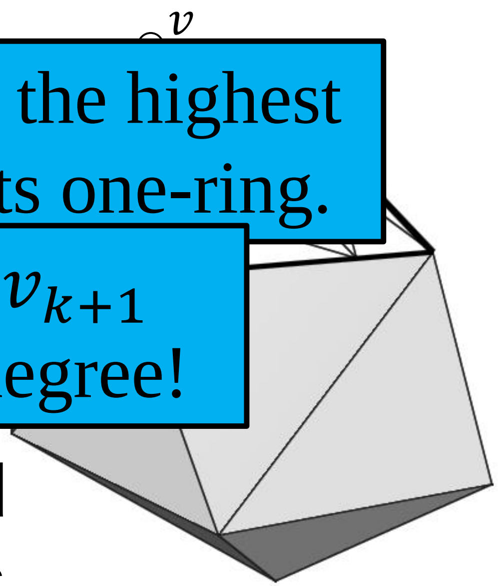
$\Rightarrow$ w was above v. ($\Rightarrow\Leftarrow$)

Constructing the Hierarchy

Claim:

After remove the highest vertex $v \in P$, the next highest vertex w has to be in the one-ring of v.

Proof: (by contradiction)



Given the highest vertex, $v_{k+1} \in P_{k+1}$ the highest vertex $v_k \in P_k$ is either v_{k+1} or is in its one-ring.

⇒ The

of w a

We can't test all neighbors of v_{k+1} because v_{k+1} may have large degree!

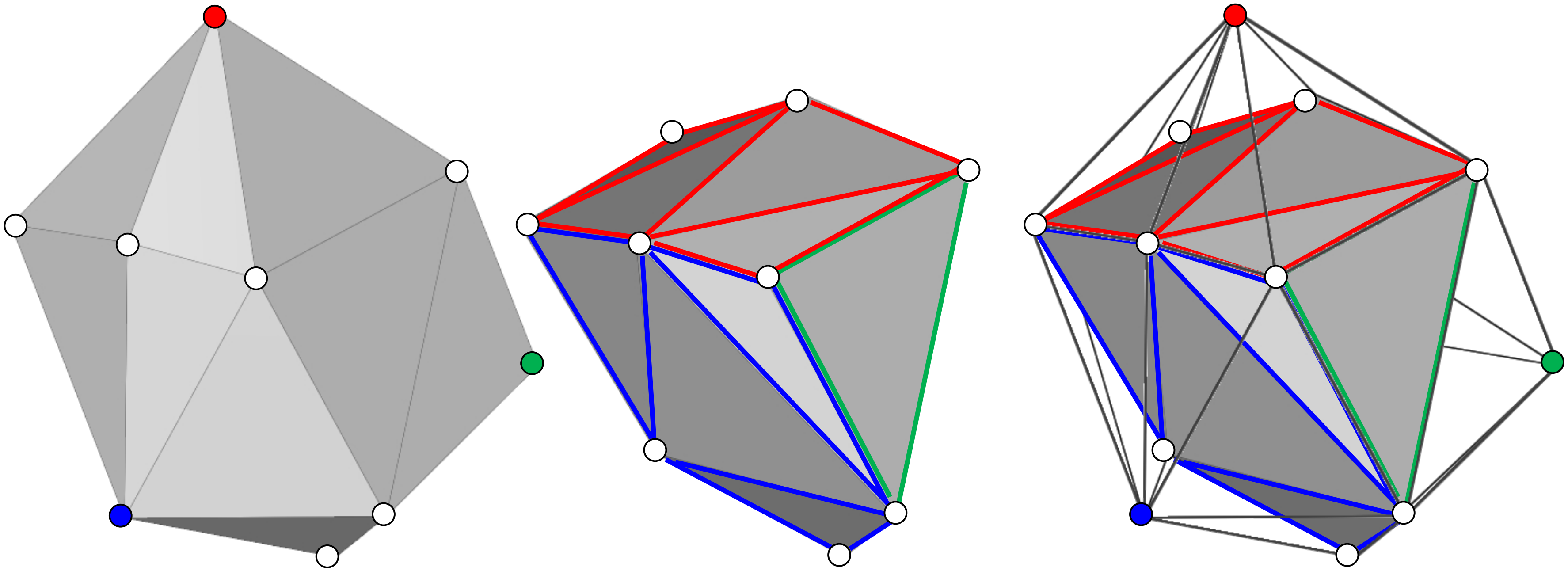
⇒ P must be below the cone apexed at w and going through its neighbors.

⇒ w was above v. (⇒⇐)

Constructing the Hierarchy

Definition:

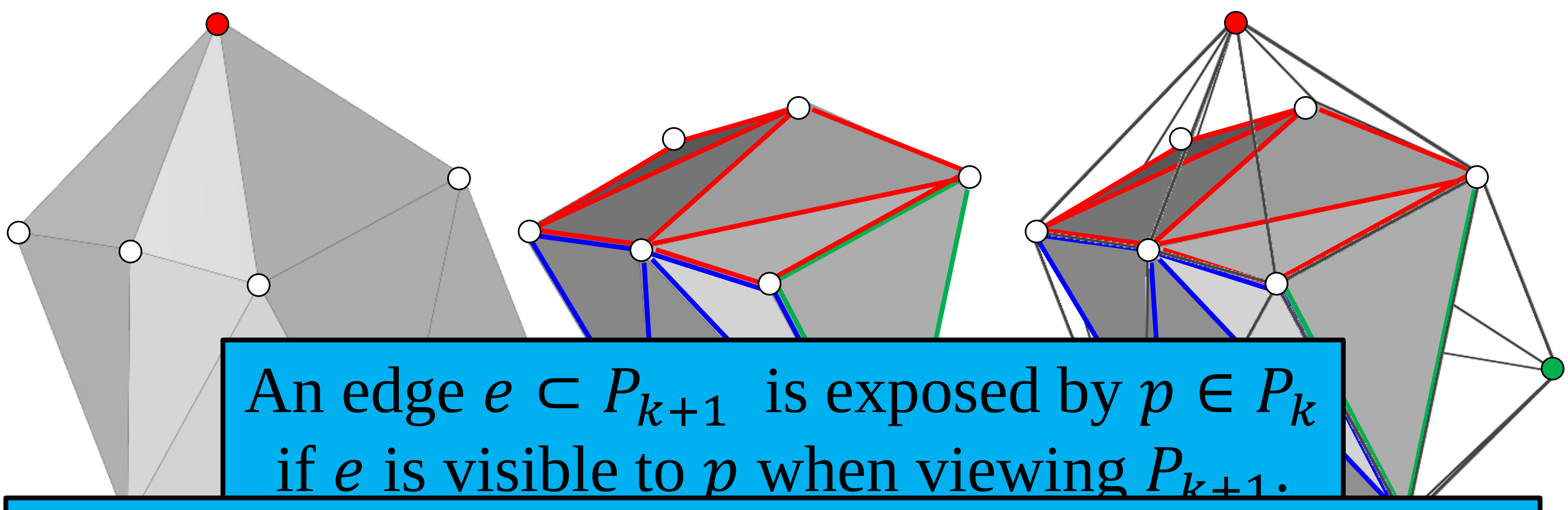
An edge e on P_{k+1} is *exposed* by a vertex $p \in P_k$, if e is in the triangulation of the hole resulting from the removal of p.



Constructing the Hierarchy

Definition:

An edge e on P_{k+1} is *exposed* by a vertex $p \in P_k$, if e is in the triangulation of the hole resulting from the removal of p.



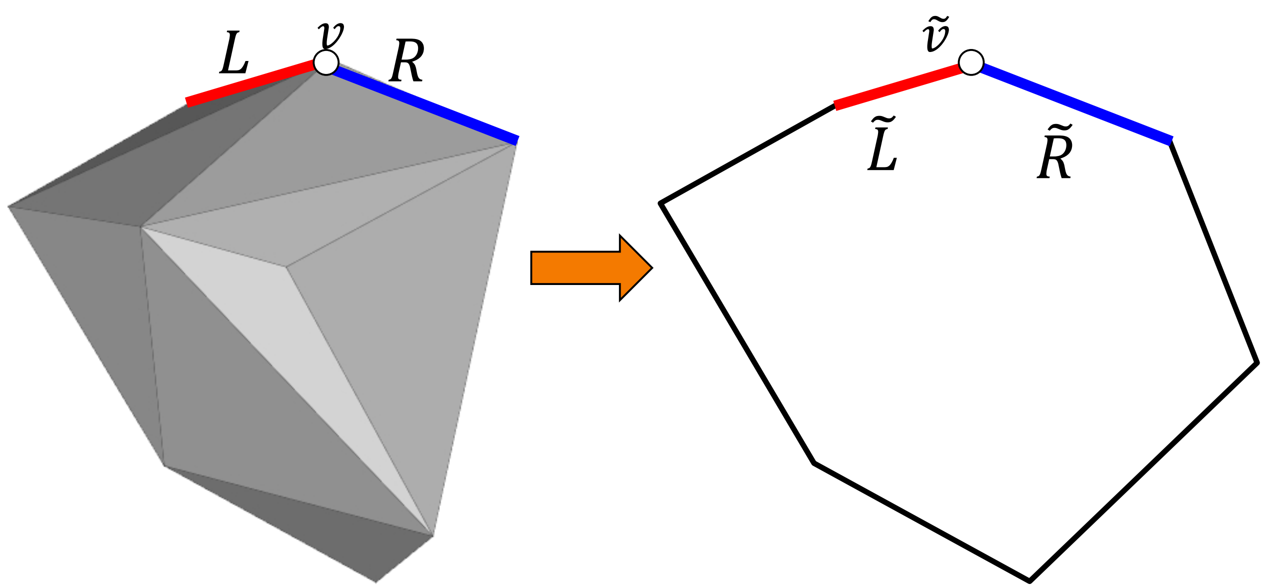
An edge $e \subset P_{k+1}$ is exposed by $p \in P_k$ if e is visible to p when viewing P_{k+1}.

An edge in P_{k+1} can be exposed by at most two vertices.

Constructing the Hierarchy

<u>Notation</u>:

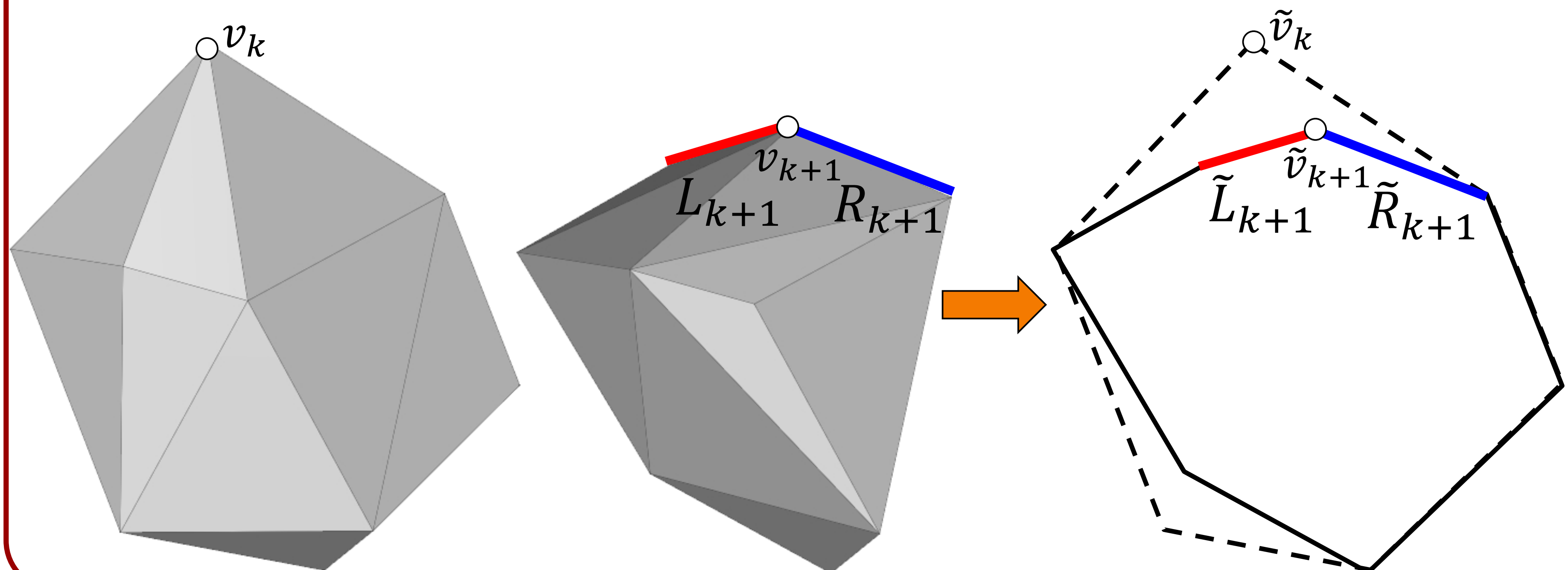
Given a polytope P, we can project it onto the yz-plane. We denote by L and R the edges that project on to the left-most and right-most edges coming out of the highest vertex.



Constructing the Hierarchy

<u>Claim</u>:

Given the highest vertex, $v_{k+1} \in P_{k+1}$ the highest vertex $v_k \in P_k$ is either v_{k+1}, or the vertex that exposes one of L_{k+1} and R_{k+1}.

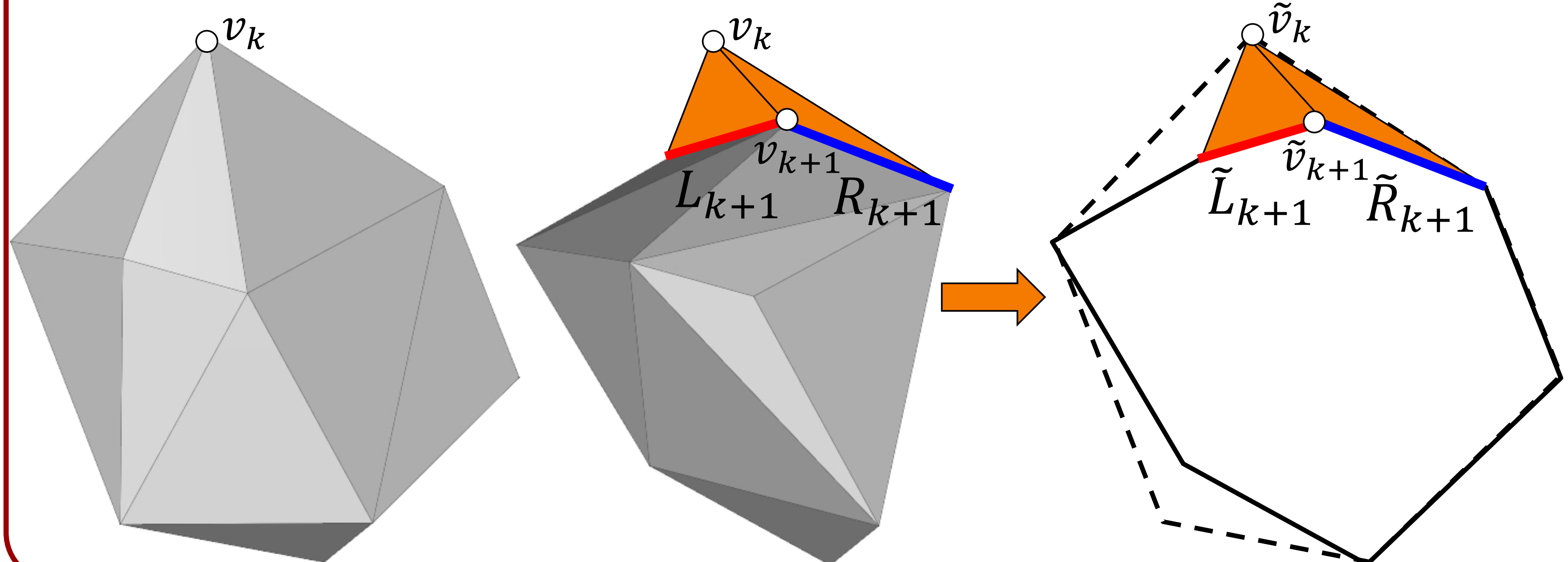


Constructing the Hierarchy

Proof:

- Draw triangles (v_k, L_{k+1}) and (v_k, R_{k+1}).
- One of these cannot intersect P_{k+1} (otherwise its projection would intersect).

$\Rightarrow$ One of L_{k+1} or R_{k+1} is exposed by v_k.



Constructing the Hierarchy

Proof:

If we know the highest vertex $v_{k+1} \in P_{k+1}$ and we know L_{k+1} and R_{k+1}, then we get the highest vertex $v_k \in P_k$ in $O(1)$.

If v_{k+1} is removed at level $k+2$:
$\Rightarrow$ v_{k+1} has degree ≤ 8
$\Rightarrow$ We can find L_{k+1} and R_{k+1} with exhaustive search.
Otherwise, we can use L_{k+2} and R_{k+2} to compute L_{k+1} and R_{k+1} in time $O(1)$.

- We can construct the polytope hierarchy in $O(|P|)$ time.
- We can find the extreme point, with respect to an arbitrary direction, in $O(\log|P|)$.