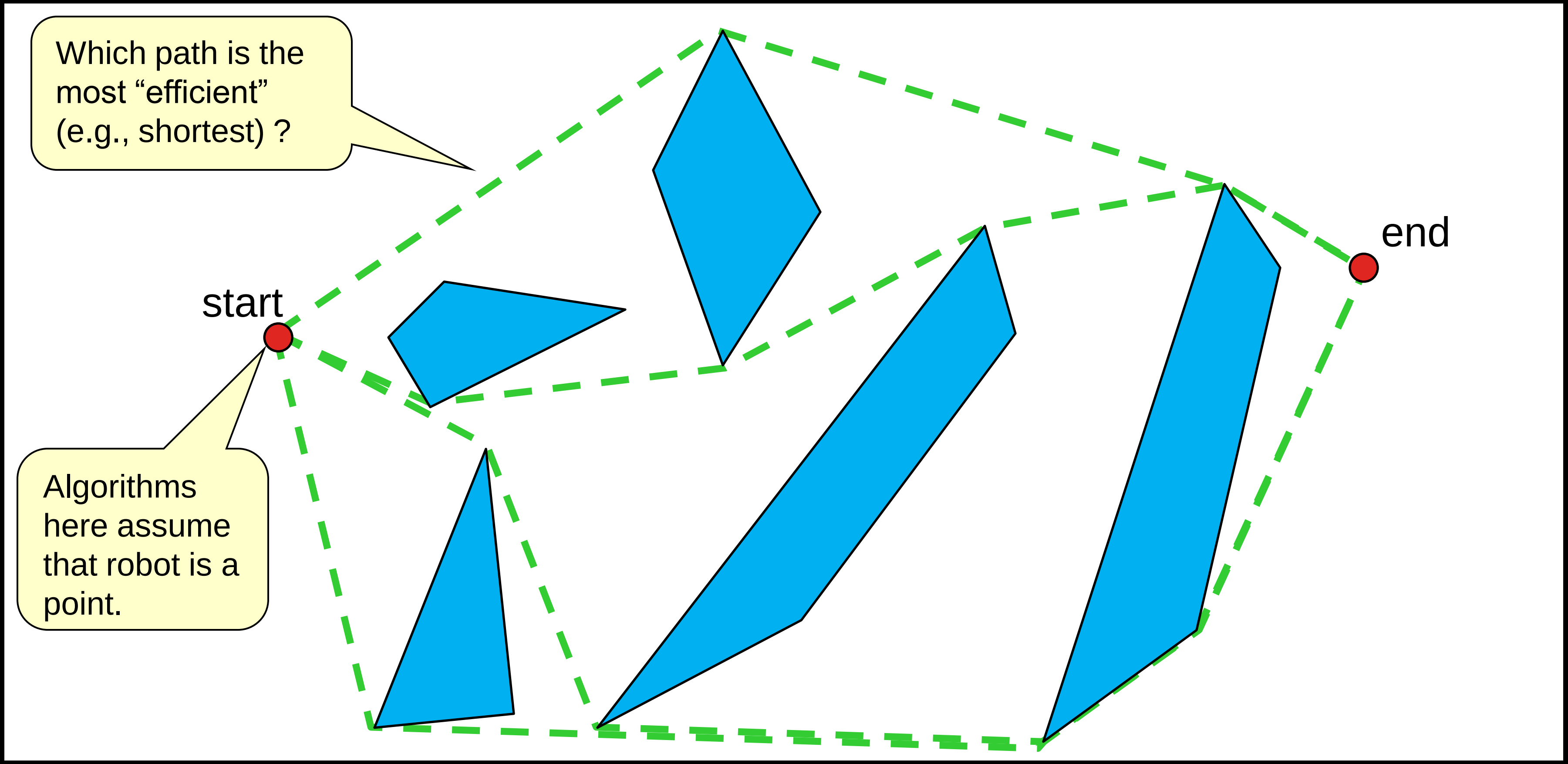
# Path Planning

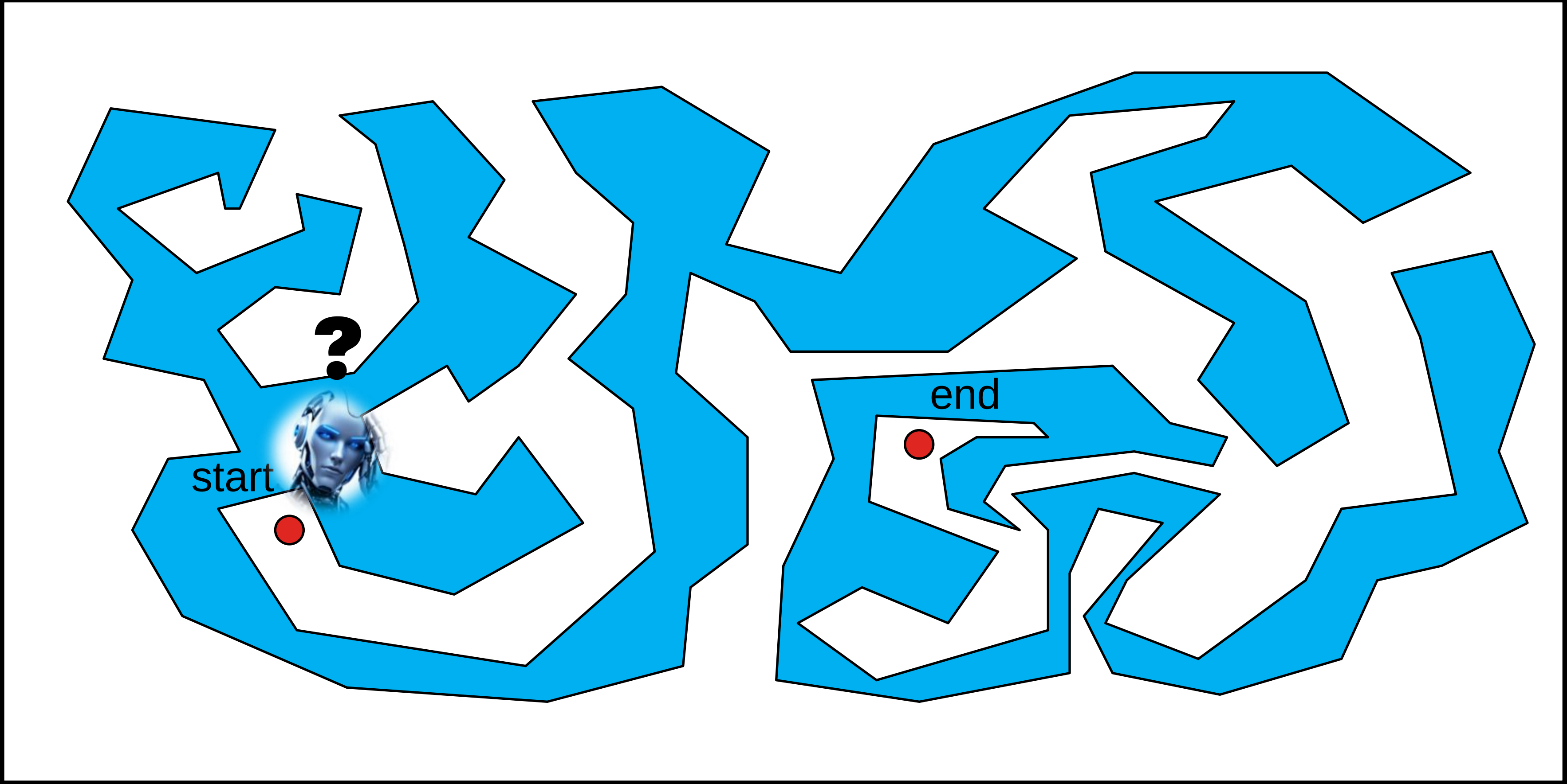
# Path Planning – Convex Obstacles

- How do we get a robot to plan a path around objects efficiently from one location to another ?



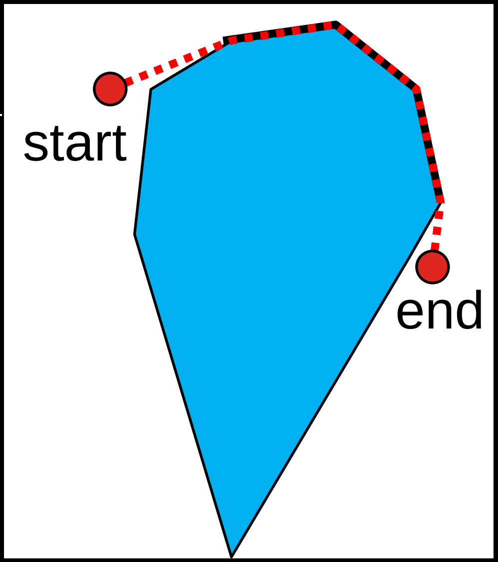
# Path Planning – Non-Convex Obst.

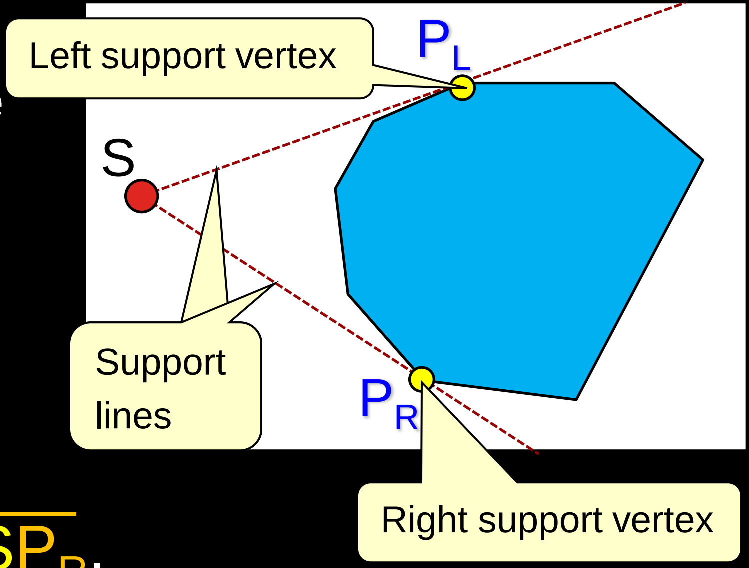
- Solution is not as obvious with non-convex obstacles. We will consider this is another lab.



# Shortest Paths

- Shortest path actually travels around obstacles, “hugging” the boundary.
- If an obstacle is in the way, robot will go around it by heading towards the left or right *support vertices*:
  - most “extreme” vertices of obstacle with respect to some point, S.
  - like “grab points” for picking up obstacle with two arms.
  - obstacle always lies completely on one side of support lines $\overline{SP_L}$ and $\overline{SP_R}$.



# Shortest Path Properties

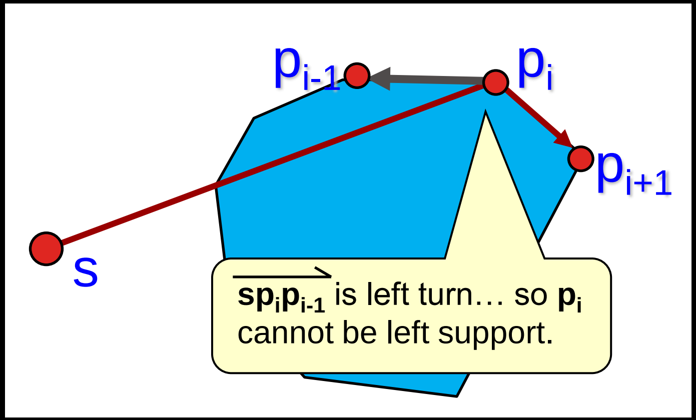
- Can find $P_L$ and $P_R$ by checking each vertex using a "left/right turn test" … for convex polygons:

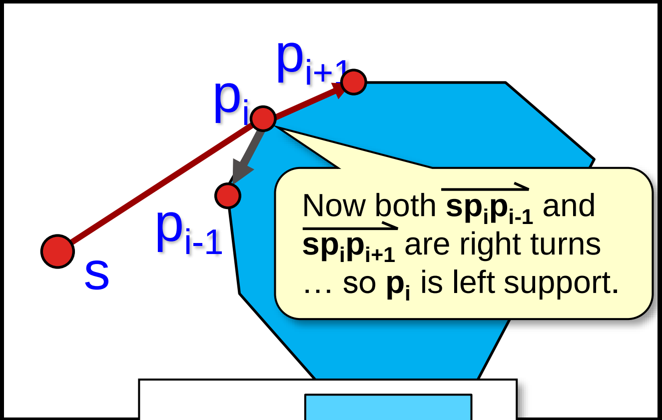
  $P_L = p_i$ if and only if both $sp_ip_{i-1}$ and $sp_ip_{i+1}$ are right turns.

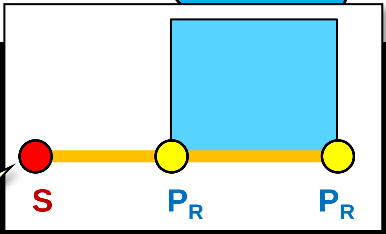
  $P_R = p_i$ if and only if both $sp_ip_{i-1}$ and $sp_ip_{i+1}$ are left turns.

  – Check all polygon vertices:

```
s = (xs,ys) is the source point
FOR (each vertex pi = (xi,yi) of the polygon) {
    pi+1 = (xi+1,yi+1) // polygon vertex after pi
    pi-1 = (xi-1,yi-1) // polygon vertex before pi
    t1 = (xi-xs)*(yi+1-ys) - (yi-ys)*(xi+1-xs)
    t2 = (xi-xs)*(yi-1-ys) - (yi-ys)*(xi-1-xs)
    IF ((t1 ≤ 0) AND (t2 ≤ 0)) THEN
        pi is the left support vertex, so add it
    IF ((t1 ≥ 0) AND (t2 ≥ 0)) THEN
        pi is the right support vertex, so add it
}
```

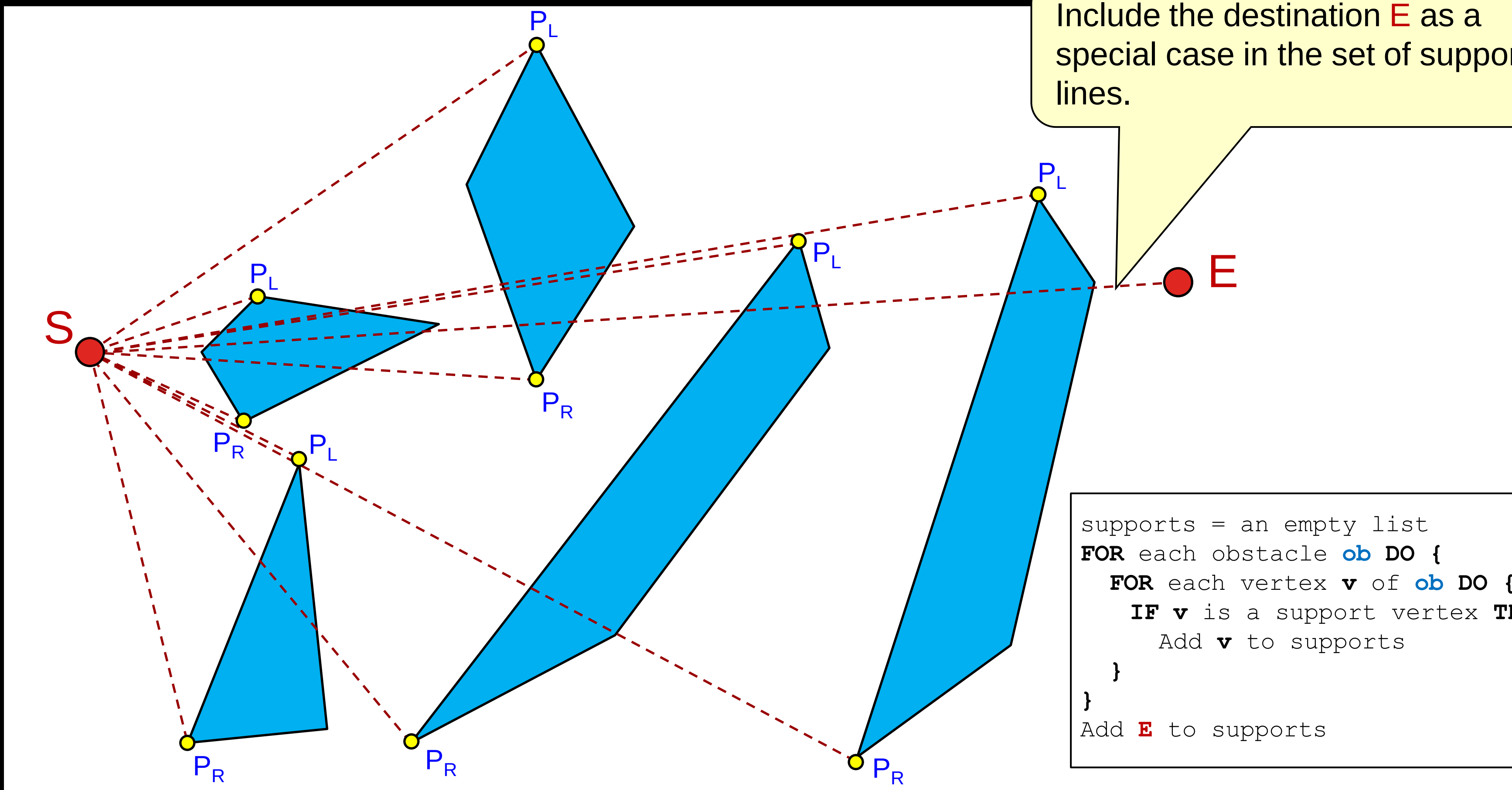


# Finding All Support Vertices

- The first step towards computing a shortest path is to find all obstacle support vertices:



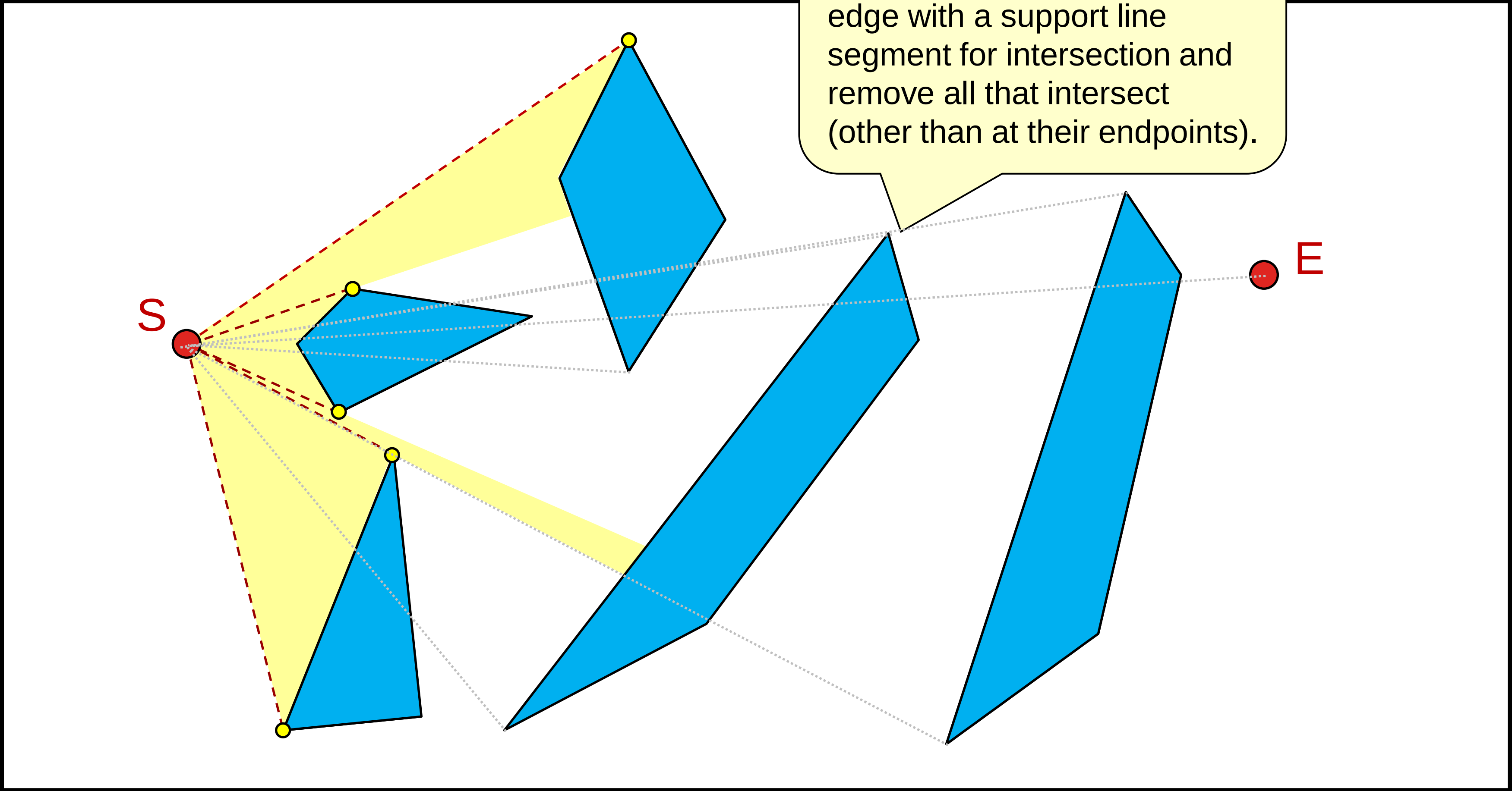

Include the destination E as a special case in the set of support lines.

```
supports = an empty list
FOR each obstacle ob DO {
  FOR each vertex v of ob DO {
    IF v is a support vertex THEN
      Add v to supports
  }
}
Add E to supports
```

# Visible Support Vertices

- Then eliminate any support vertices that are not visible from S:

Can compare every polygon edge with a support line segment for intersection and remove all that intersect (other than at their endpoints).



# Eliminating Support Vertices

- Just need to add an IF statement before adding:

```
supports = an empty list
FOR each obstacle ob DO {
    FOR each vertex v of ob DO {
        IF v is a support vertex THEN {
            IF SupportLineIntersectsObstacle(S, v, obstacles) is false THEN
                Add v to supports
        }
    }
}
IF SupportLineIntersectsObstacle(S, E, obstacles) is false THEN
    Add E to supports
```

S → v is a support line

Add this function call

S → E is a support line

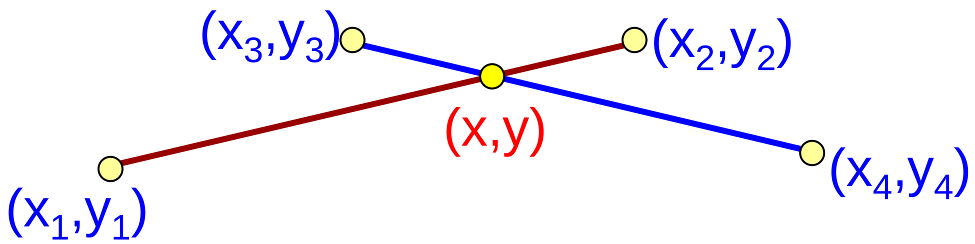
Add this function call

- Function checks each support line with all obstacles:

```
SupportLineIntersectsObstacle(S, supportPoint, obstacles) {

    FOR each obstacle ob of obstacles DO {
        FOR each vertex v of ob DO {
            va = vertex of ob after v
            IF support line from S to supportPoint intersects obstacle edge v → va THEN
                RETURN true
        }
    }
    RETURN false

}
```

# Line Intersection test

- How do we check for line-segment intersection ?




- Can use well-known equation of a line:

$$y = m_a x + b_a \qquad y = m_b x + b_b$$

where

$$m_a = (y_2 - y_1) / (x_2 - x_1) \qquad m_b = (y_4 - y_3) / (x_4 - x_3)$$

$$b_a = y_1 - x_1 m_a \qquad b_b = y_3 - x_3 m_b$$

Must handle special case where lines are vertical. (i.e., $\mathbf{x_1} == \mathbf{x_2}$ or $\mathbf{x_3} == \mathbf{x_4}$)

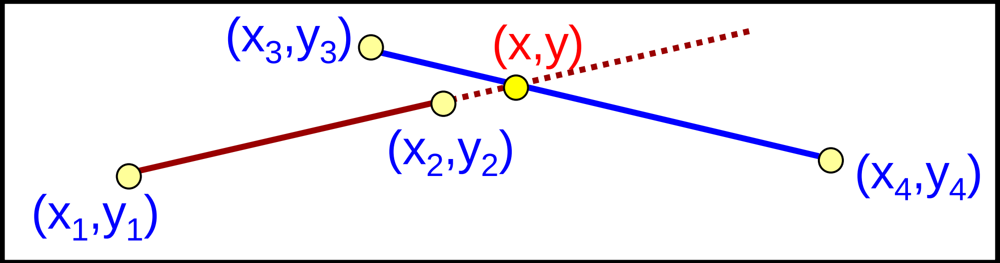
- Intersection occurs when these are equal:

$$m_a x + b_a = m_b x + b_b \quad \rightarrow \quad x = (b_b - b_a) / (m_a - m_b)$$

If ($m_a == m_b$) the lines are parallel and there is no intersection

# Line Intersection test

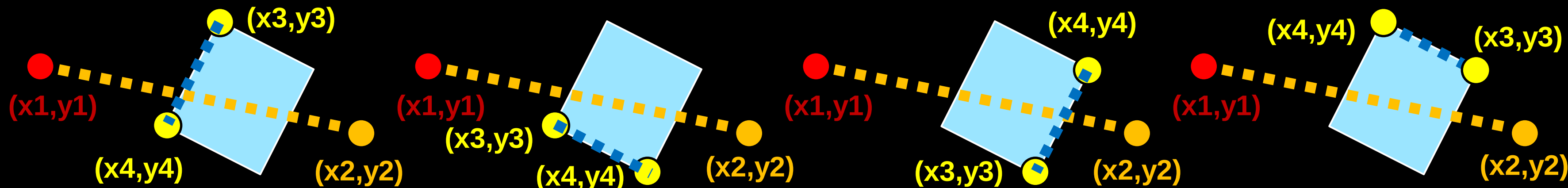
- Final test is to ensure that intersection ($x$, $y$) lies on line segment … just make sure that each of these is true:
  - $\max(x_1, x_2) \geq x \geq \min(x_1, x_2)$
  - $\max(x_3, x_4) \geq x \geq \min(x_3, x_4)$




- In java, we have a nice function to do all this for us:

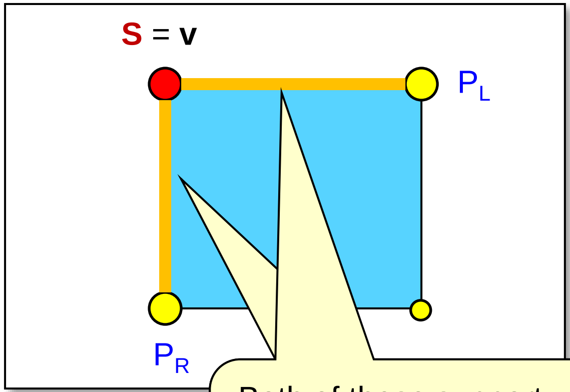
```
java.awt.geom.Line2D.Double.linesIntersect(x1,y1,x2,y2,x3,y3,x4,y4)
```

- You will be checking intersection of a support line with each edge of an obstacle:



# Handling Special Cases: 1

```
supports = an empty list

FOR each obstacle ob DO {

    FOR each vertex v of ob DO {

       IF S has the same coordinates as v THEN {
          Add vertex of ob before v to supports
          Add vertex of ob after v to supports
       }

       OTHERWISE {

          IF v is a support vertex THEN {
             IF SupportLineIntersectsObstacle(S, v, obstacles) is false THEN
                Add v to supports
          }

       }
    }
}
IF SupportLineIntersectsObstacle(S, E, obstacles) is false THEN
    Add E to supports
```



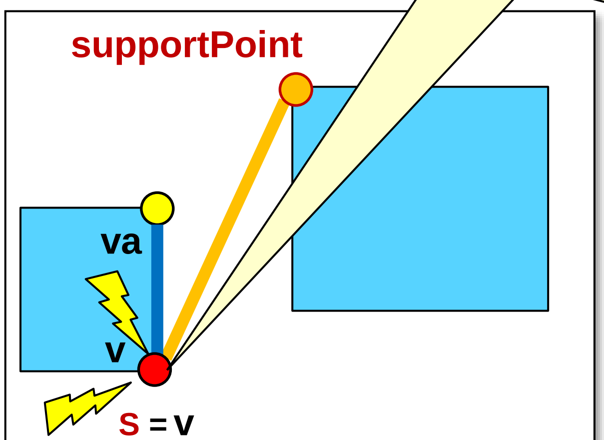

Both of these support lines are ok since they are on the same obstacle

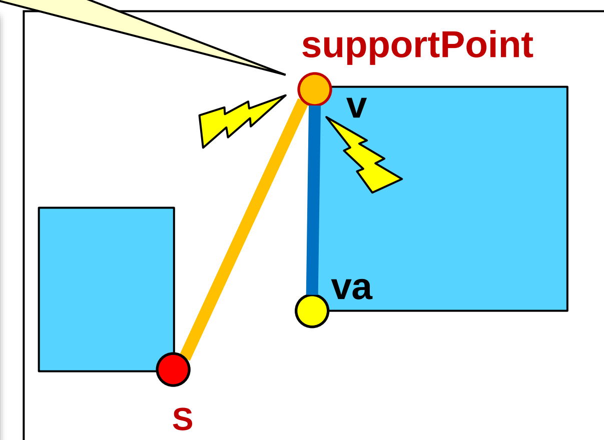
# Handling Special Cases: 2

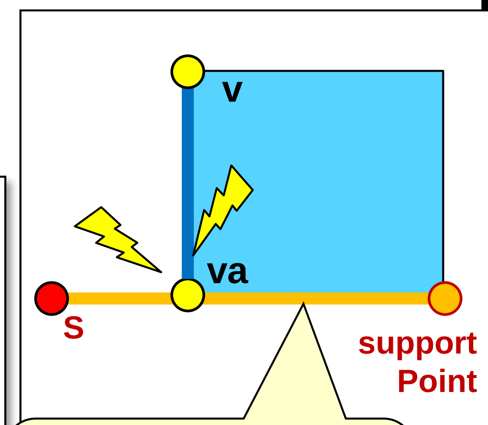
```
SupportLineIntersectsObstacle(S, supportPoint, obstacles) {

    FOR each obstacle ob of obstacles DO {
        FOR each vertex v of ob DO {
            va = vertex of ob after v

            IF [(support line from S to supportPoint intersects obstacle edge v → va) AND
                (S is not the same coordinate as v or va) AND
                (supportPoint is not the same coordinate as v or va)] THEN {
                RETURN true
            }
        }
    }
    RETURN false
}
```

Be careful with your logic here. Many students get this wrong.

By definition, the support line intersects this obstacle at the vertex. But this is ok.



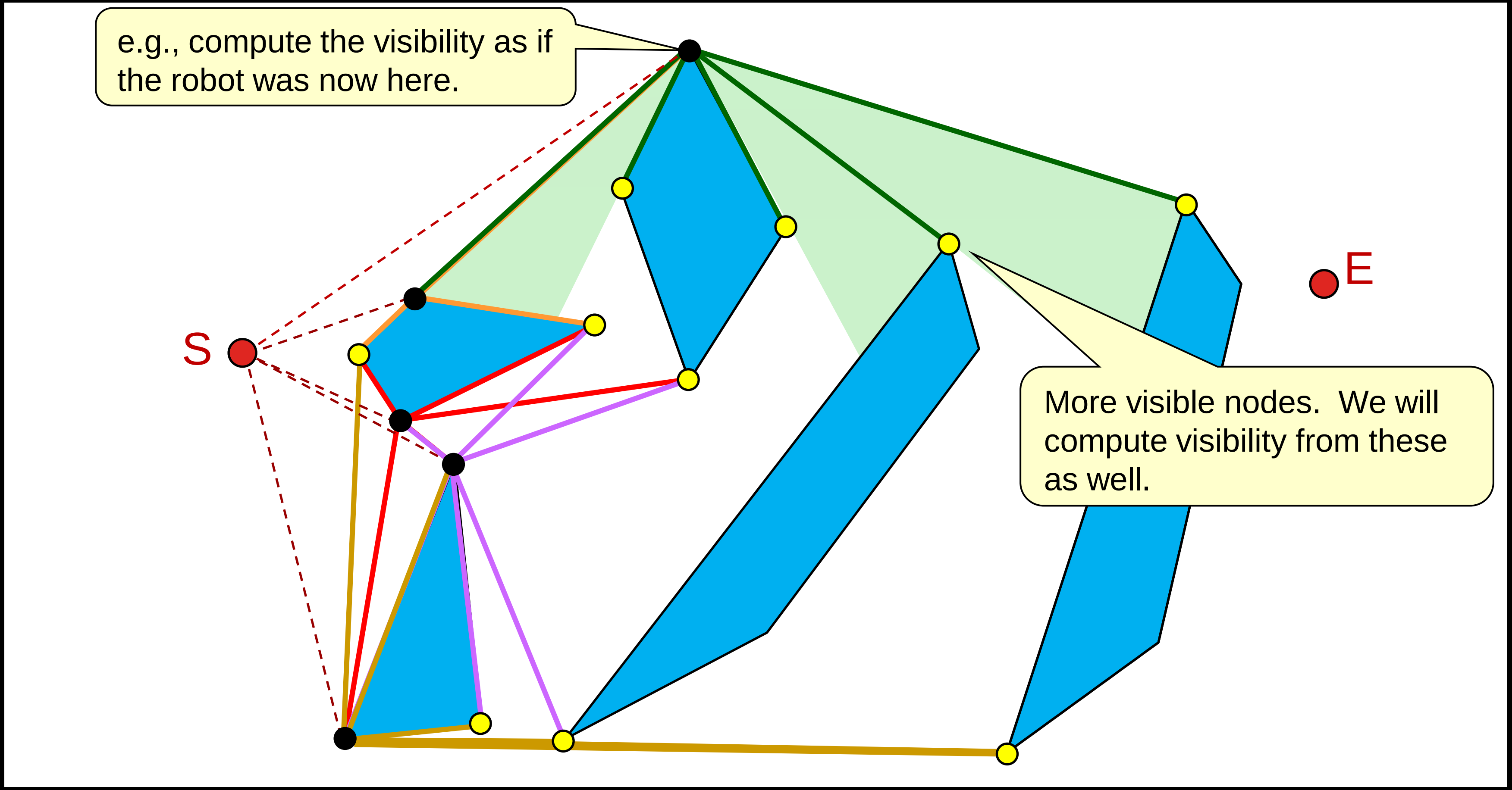




This support line running parallel to the obstacle edge will intersect and will be removed.

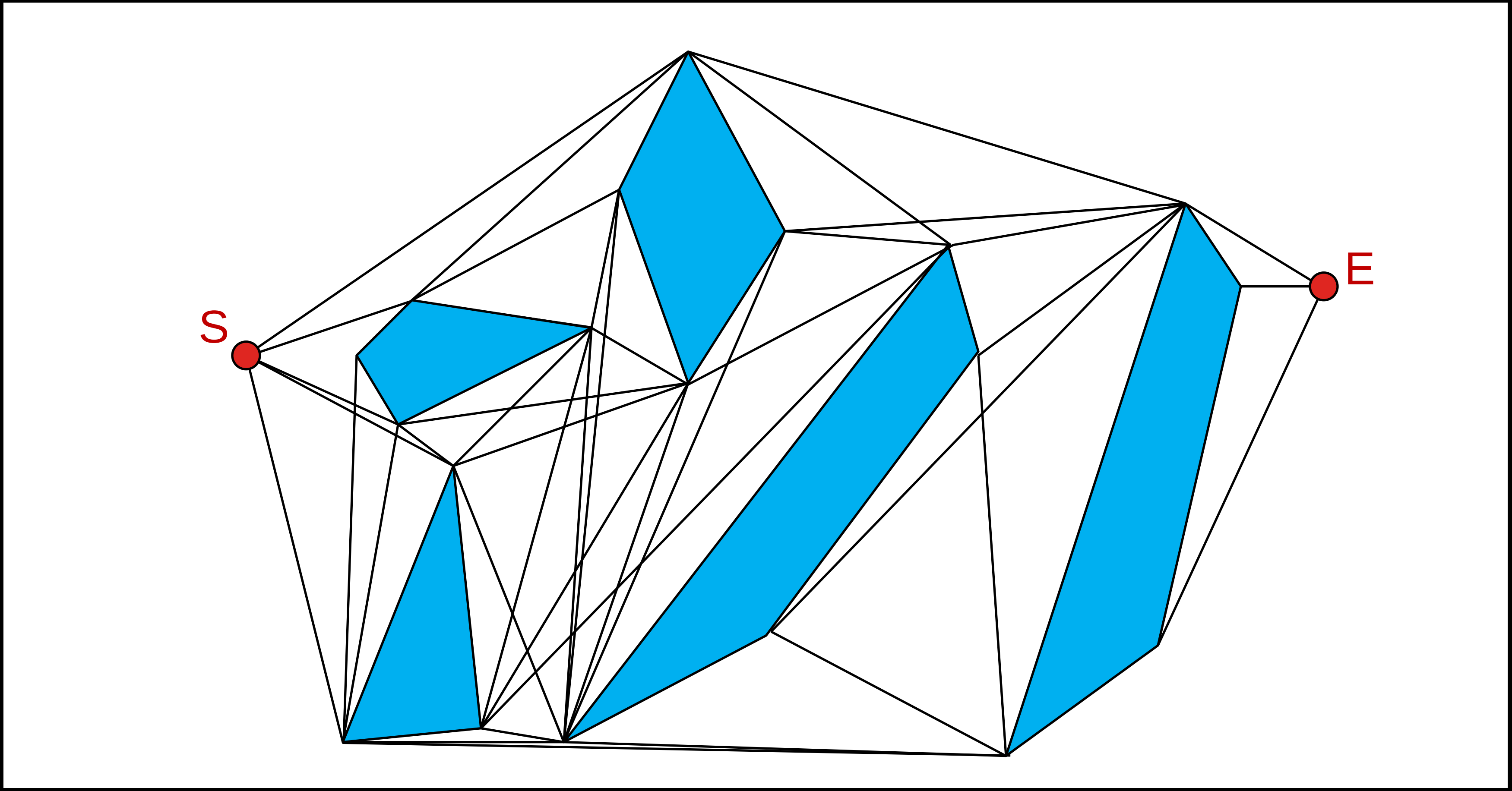
# Iterating Through Support Vertices

- We will now repeat this process from each obstacle vertex (as if robot traveled to those vertices):



# The Visibility Graph

- By appending all these visible segments together, a visibility graph is obtained:



# The Pseudocode

```
computeVisibilityGraph() {
    graph = an empty graph

    Add S as a Node of the graph
    Add E as a Node of the graph


    FOR each obstacle ob of obstacles DO {
        FOR each vertex v of ob DO {
            IF v is not already a Node in the graph THEN
                Add v as a Node in the graph
        }
    }


    FOR each Node n of the graph DO {

        Find all visible support points from n

        FOR each visible support point p that we found DO {
            m = find the node at point p in the graph

            IF ((m was found) AND (n!=m)) THEN
                Add an Edge in the graph from Node n to Node m
        }
    }
}
```

S and E are the start and end points of our environment

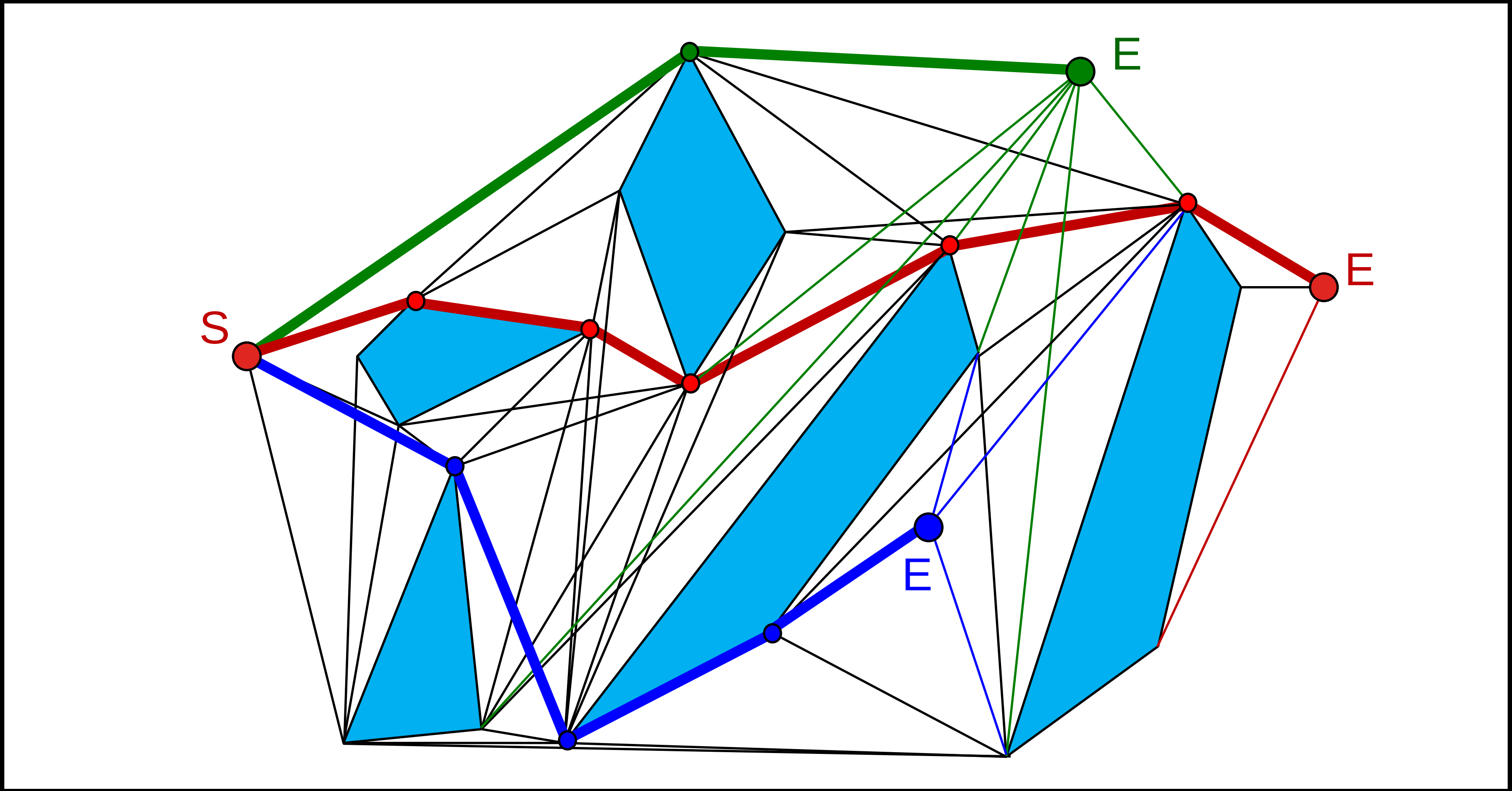
These are the obstacles of our environment

This is all our hard work from before

Don’t check coordinate values here … just make sure that **n** is not the same identical node as **m**.

# Visibility Graph Paths

- Shortest paths from the start to the end location will <u>always</u> travel along visibility graph edges:



# Start the Lab …