Clipping Primitives

Why?

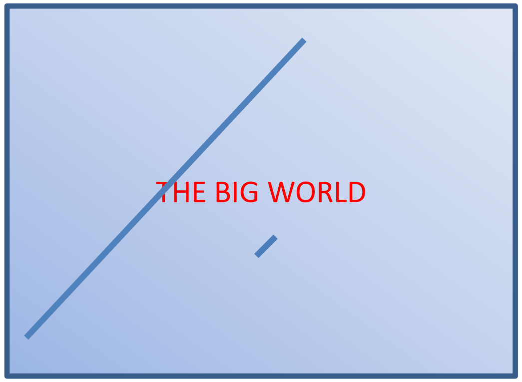
How?

THE “BIG REAL” WORLD

One possibility

THE BIG WORLD

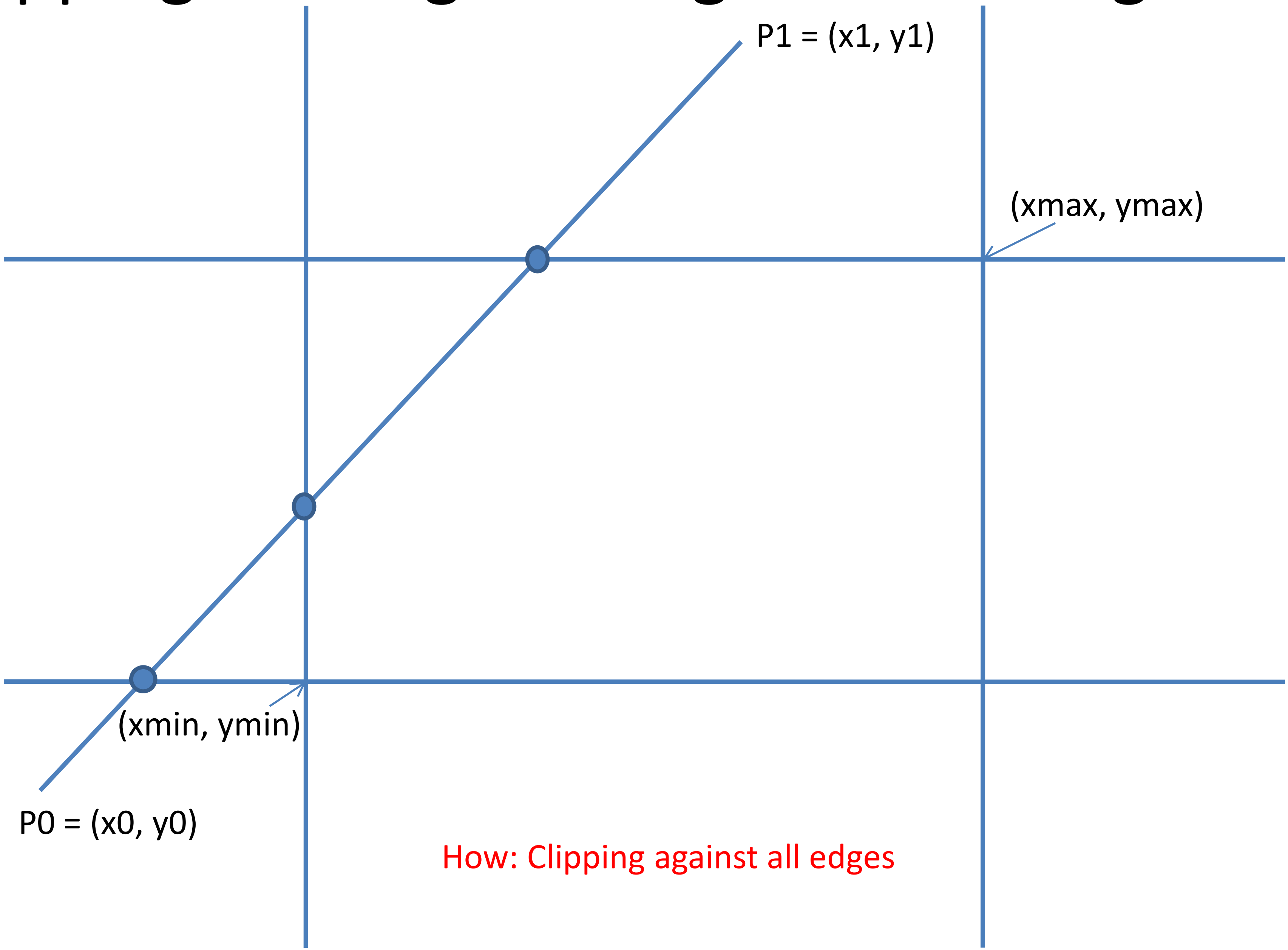
One size cannot fit all – zooming in too much makes primitives disappear



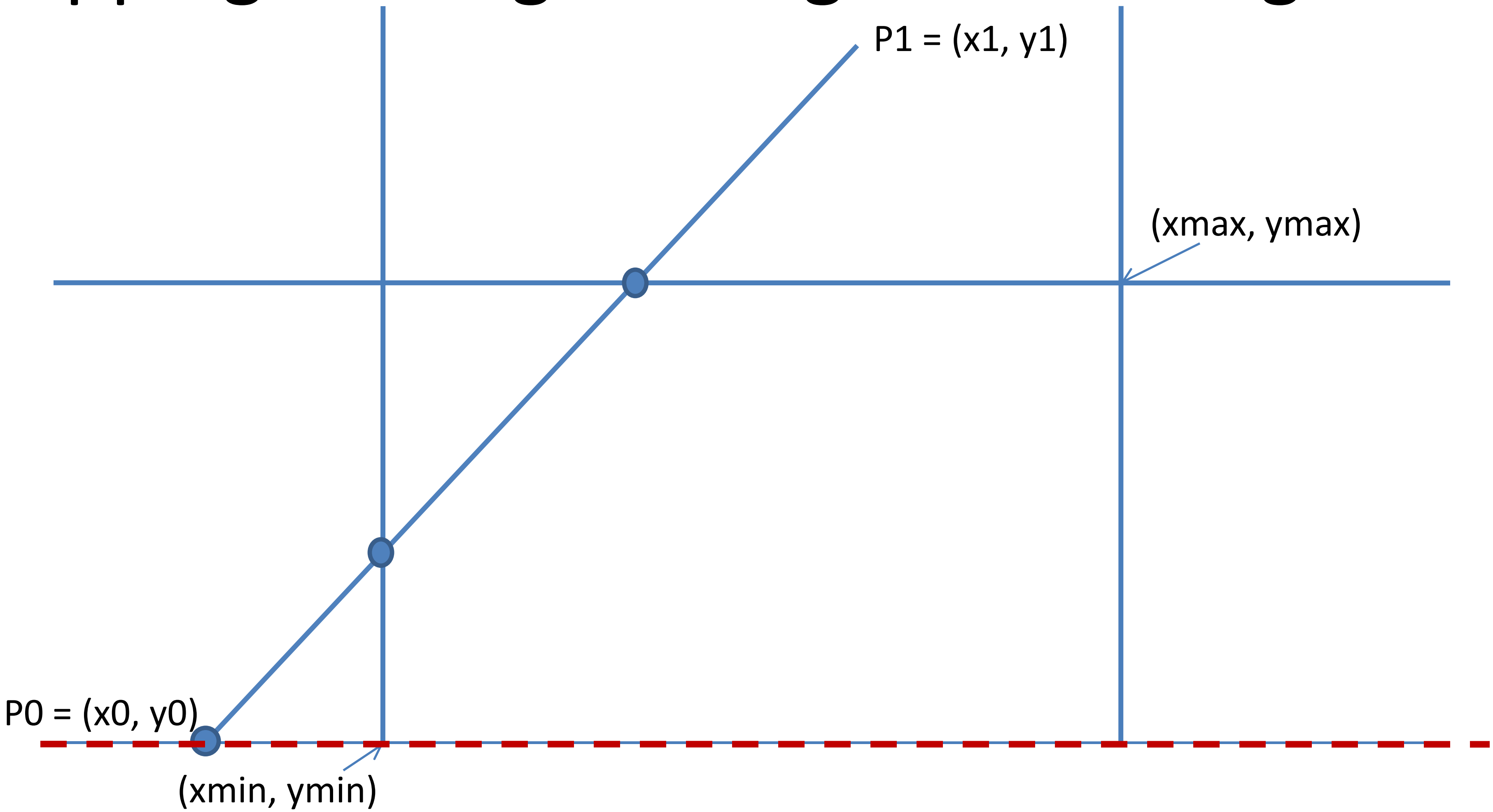
What needs to be done?

- Clipping against rectangle (screen is usually rectangle!)
- Clipping against multiple rectangles (window system GUI)
- Clipping of every primitive against rectangle (begin with line as it is the basic and many other primitives could use the algorithm)
- Here: clipping against ONE rectangle

Clipping line segment against rectangle

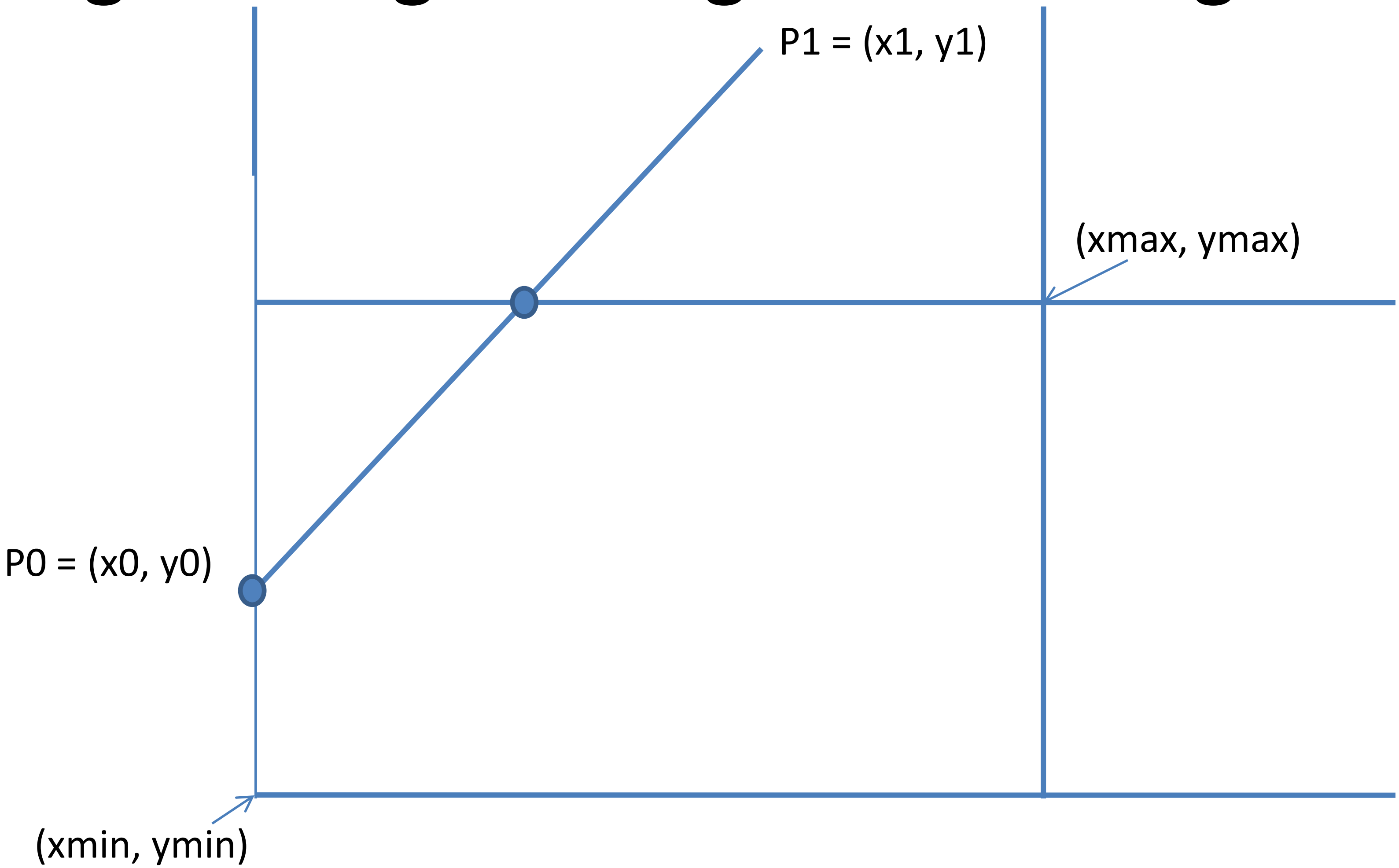


Clipping line segment against rectangle



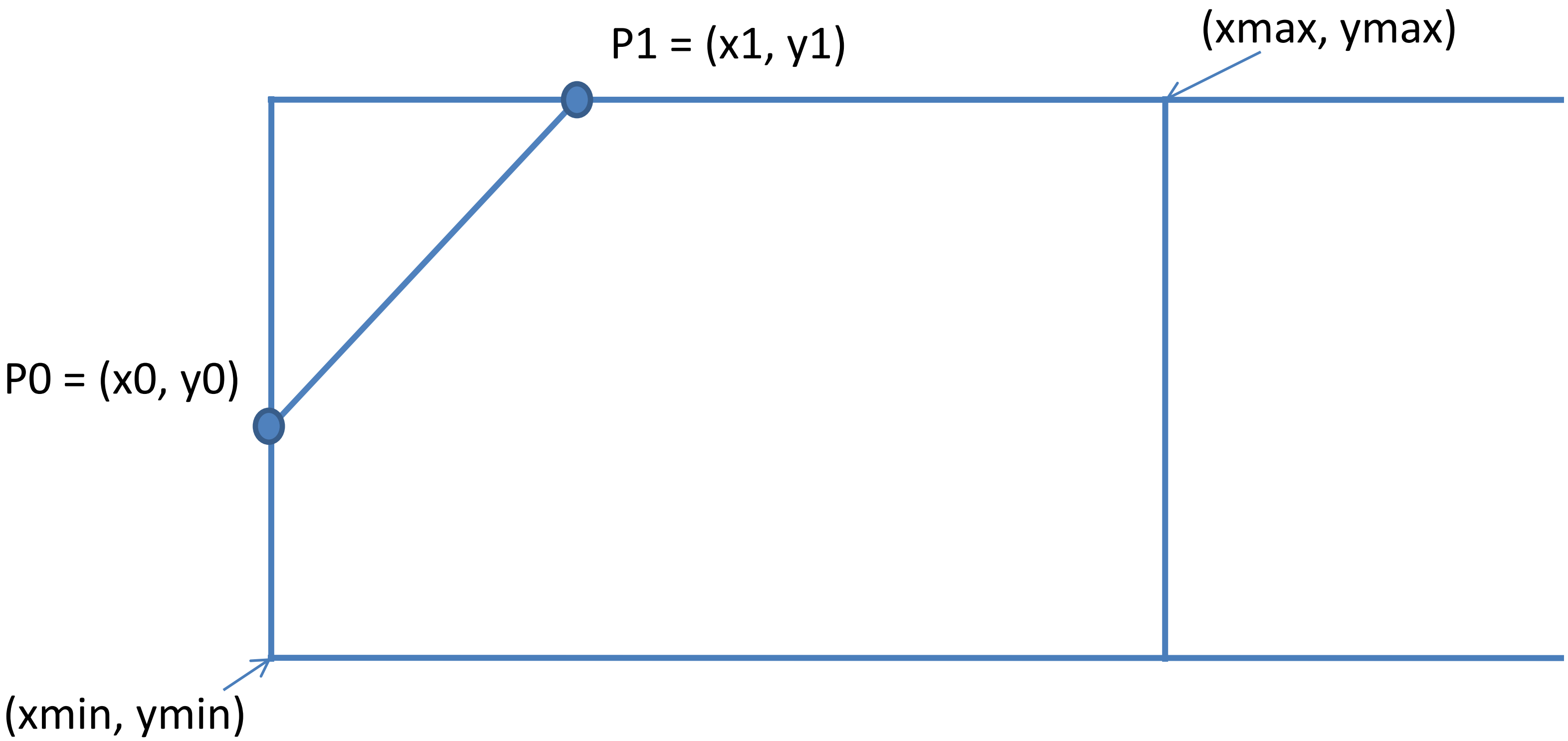
Clip against the bottom edge

Clipping line segment against rectangle



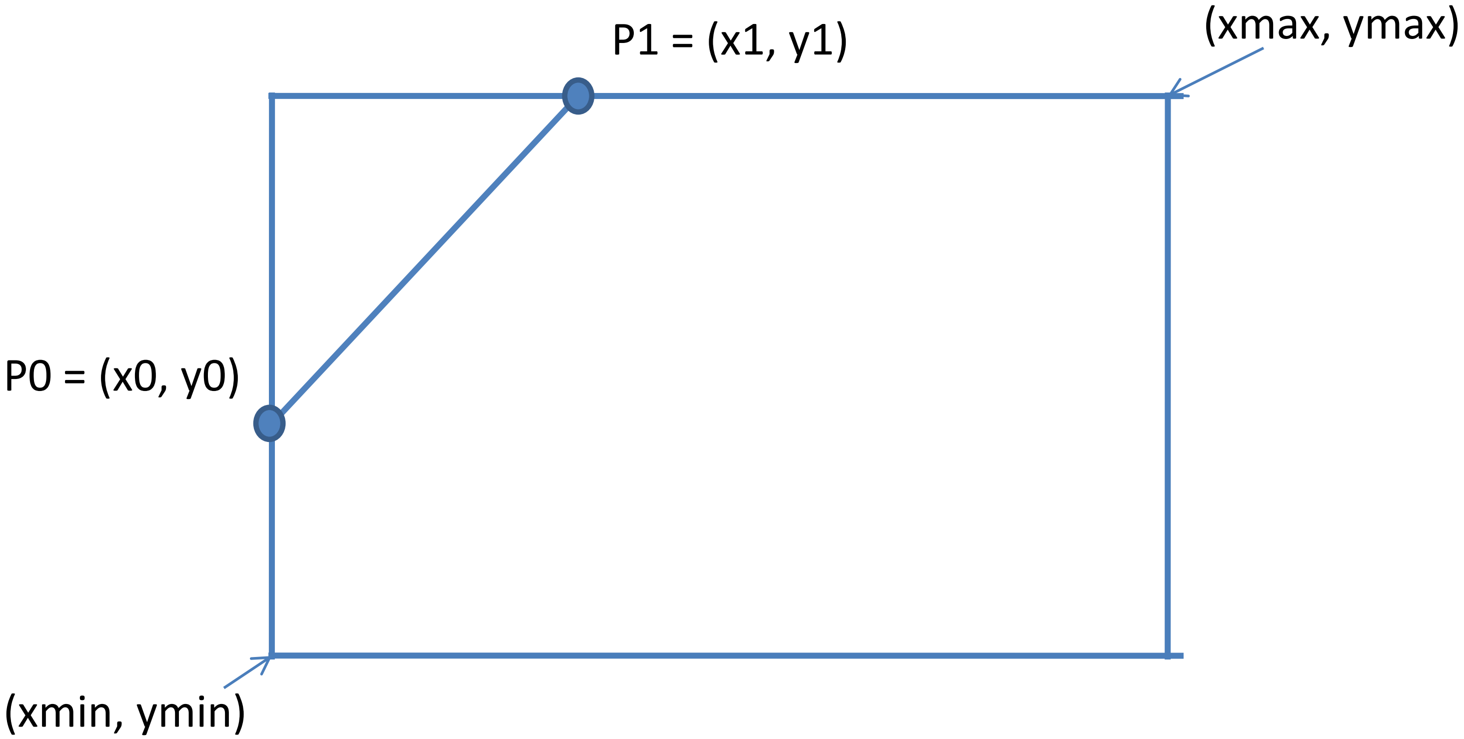
Clip against the bottom edge, and left edge

Clipping line segment against rectangle



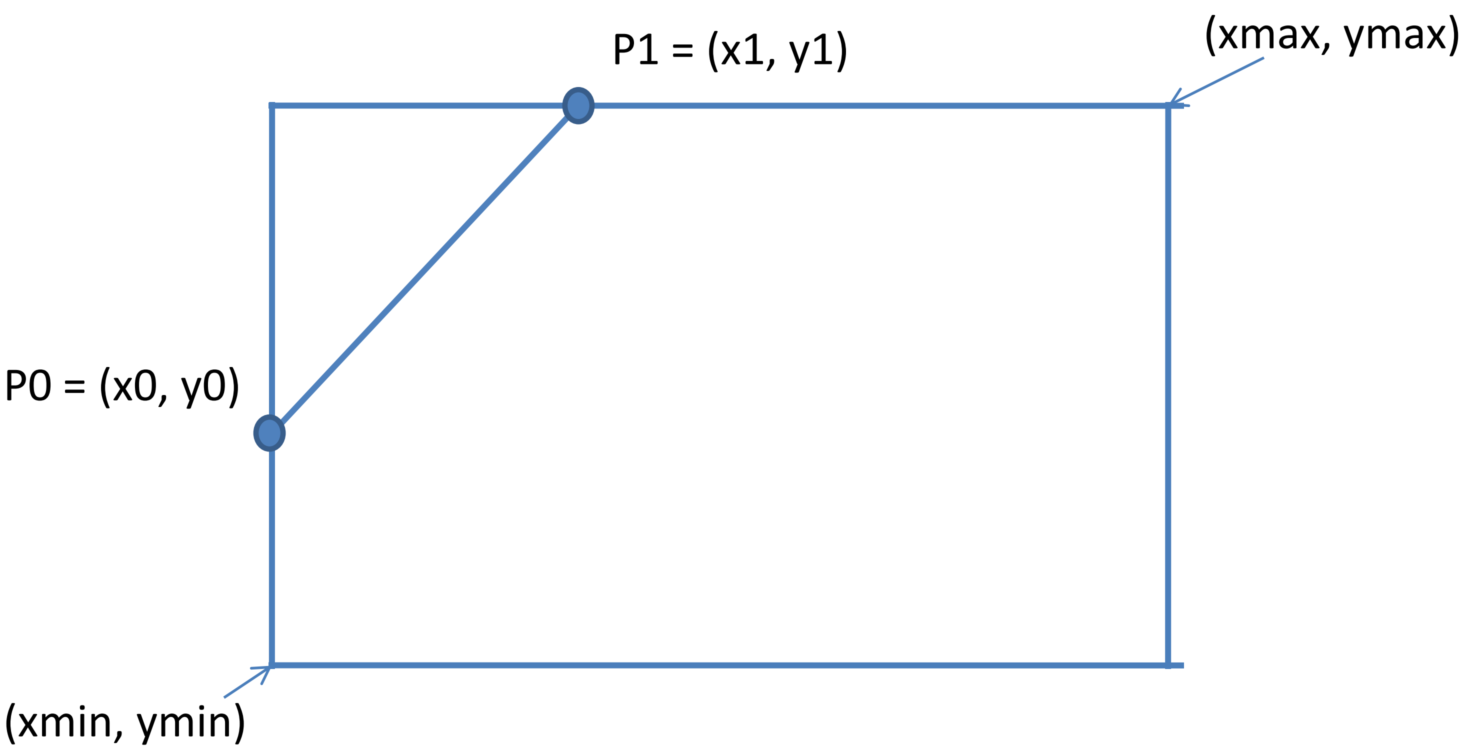
Clip against the bottom edge, left edge, and top edge

Clipping line segment against rectangle



Clip against all four edges

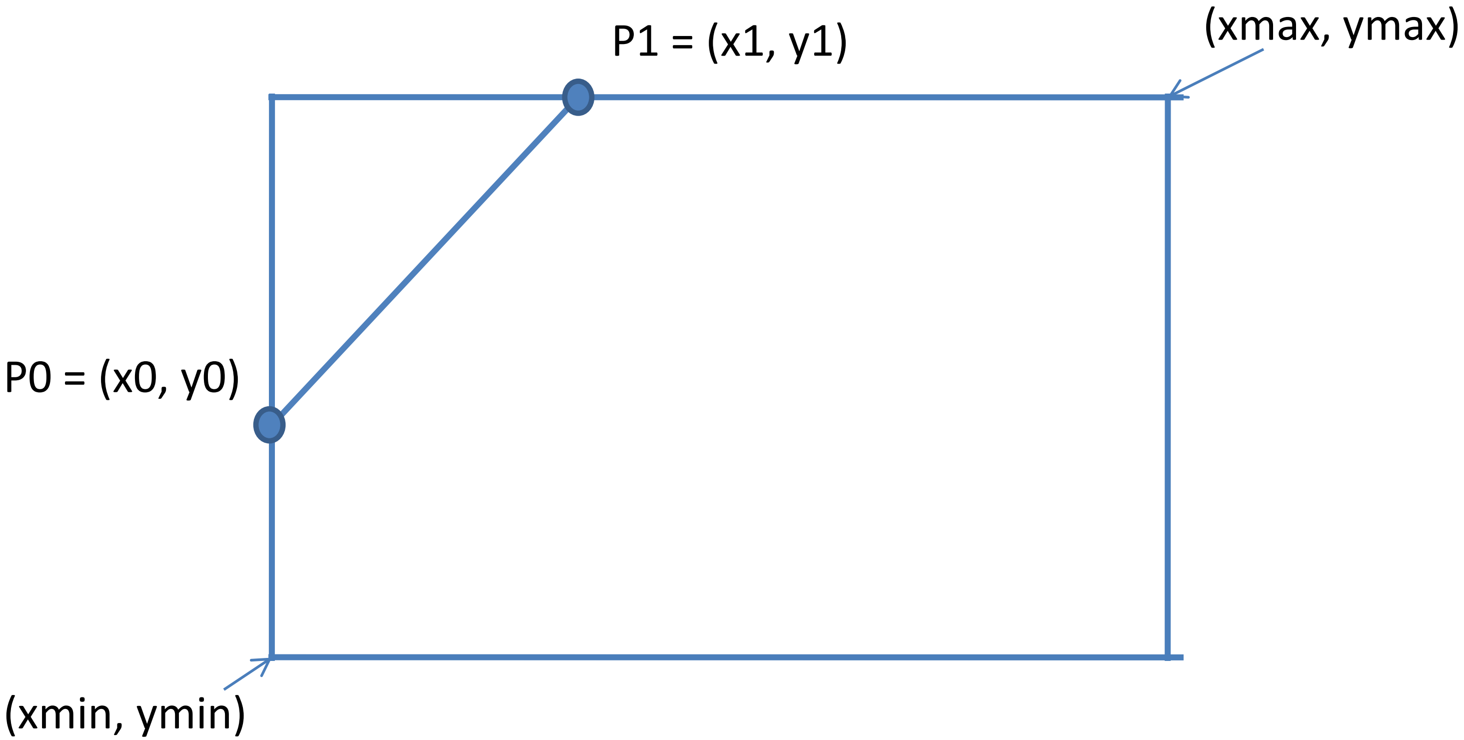
Clipping line segment against rectangle



Basic Ideas:

- determine the intersections (where does the line intersect with the extended edges?)
- repeatedly shrinks the line

Clipping line segment against rectangle



➔ solving equations to find the intersections of each edge with the line segment

For example: to find the intersection with the BOTTOME edge:

$y = ty0 + (1-t)y1 = ymin$ => $t = (ymin – y1)/(y0-y1)$

$x = tx0 + (1-t)x1$ => $x = x1 + (x0-x1)* (ymin – y1)/(y0-y1)$

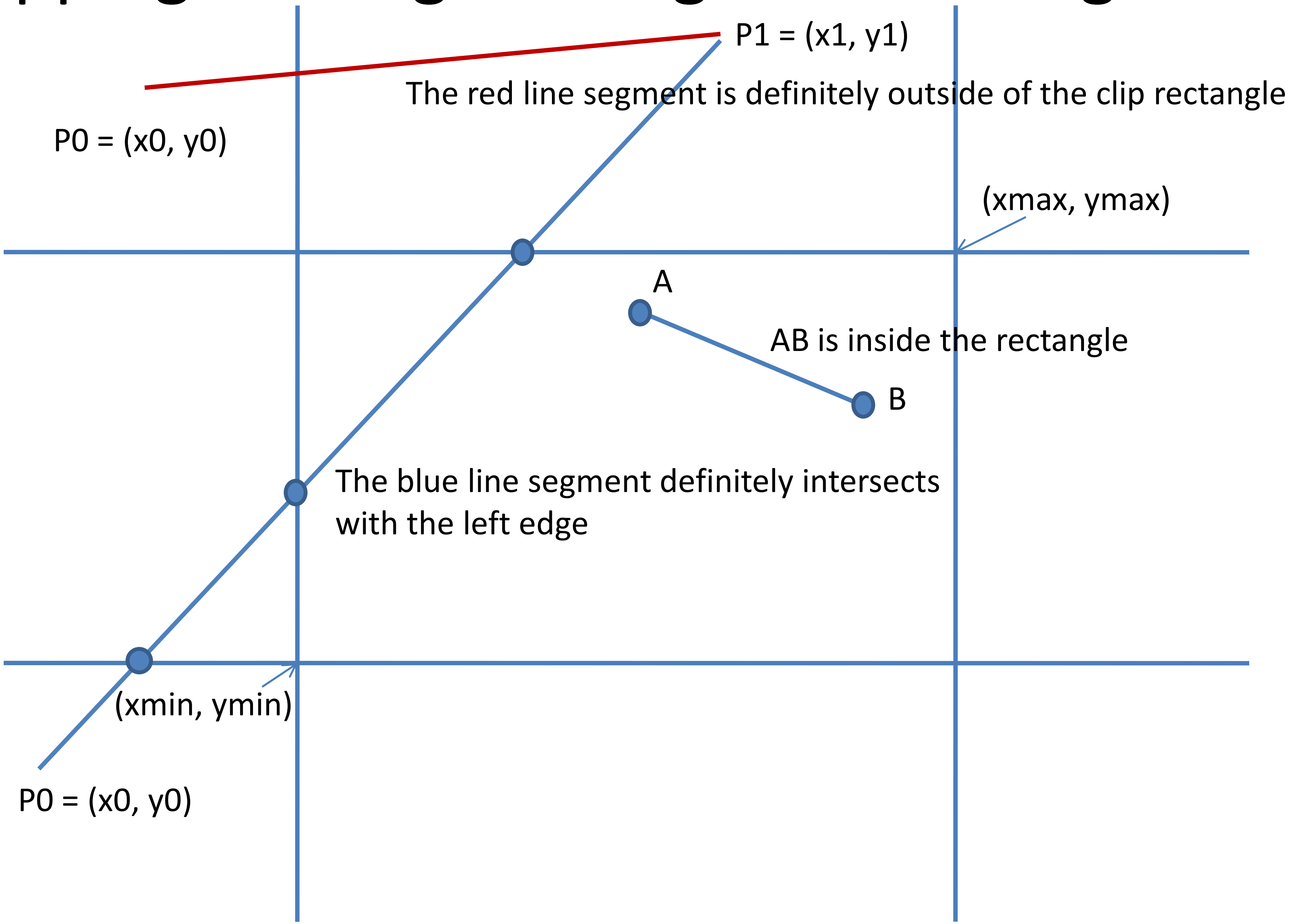
Then, check whether the intersections are inside or outside the line segment, the edge

Ideas and Questions

- Computing the intersections are simple but could be avoided in several situations
- There are situations in which a simple comparison is sufficient to decide whether the line intersects the edge

➔ Make use of the observation to eliminate some computations

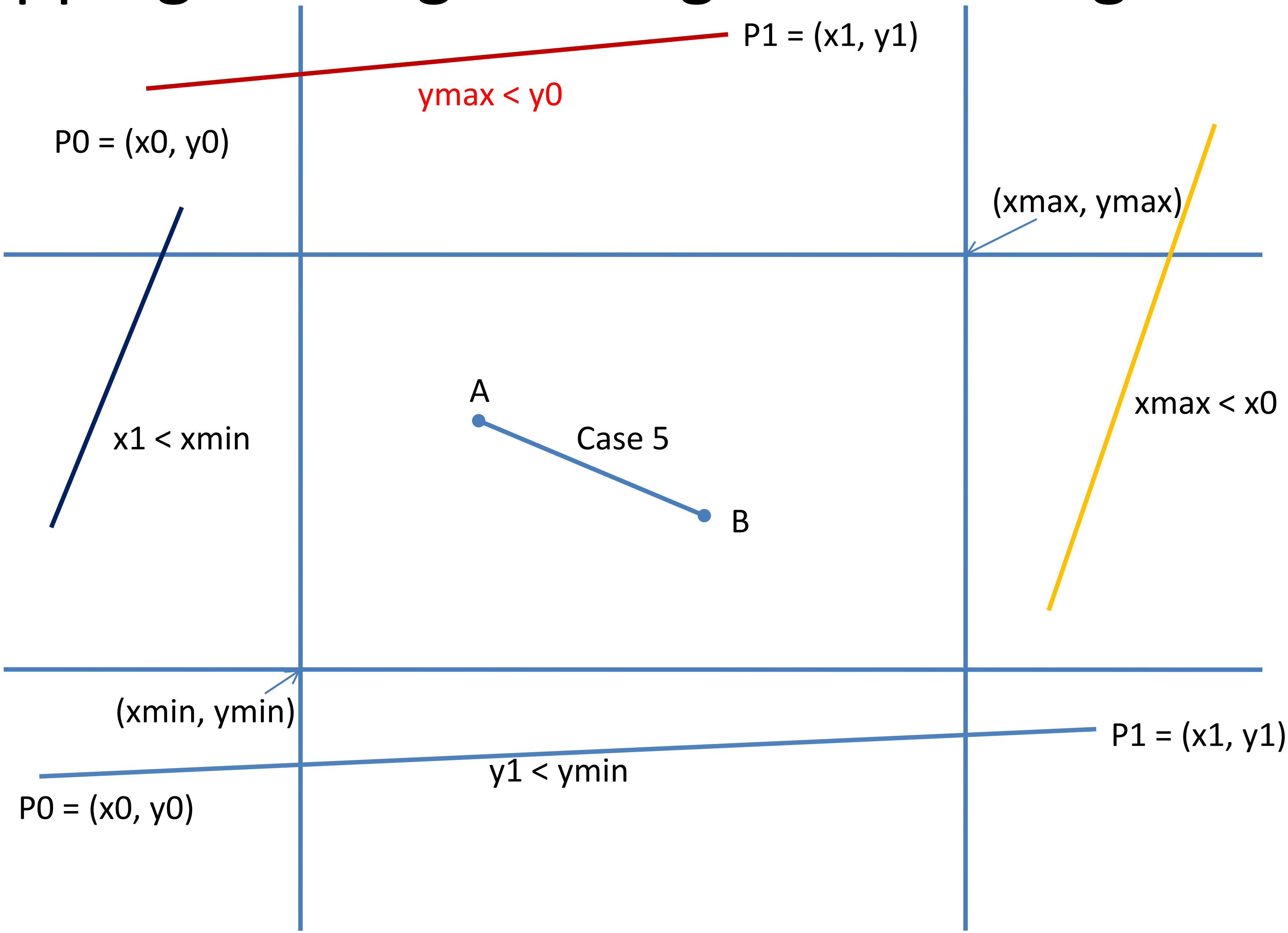
Clipping line segment against rectangle



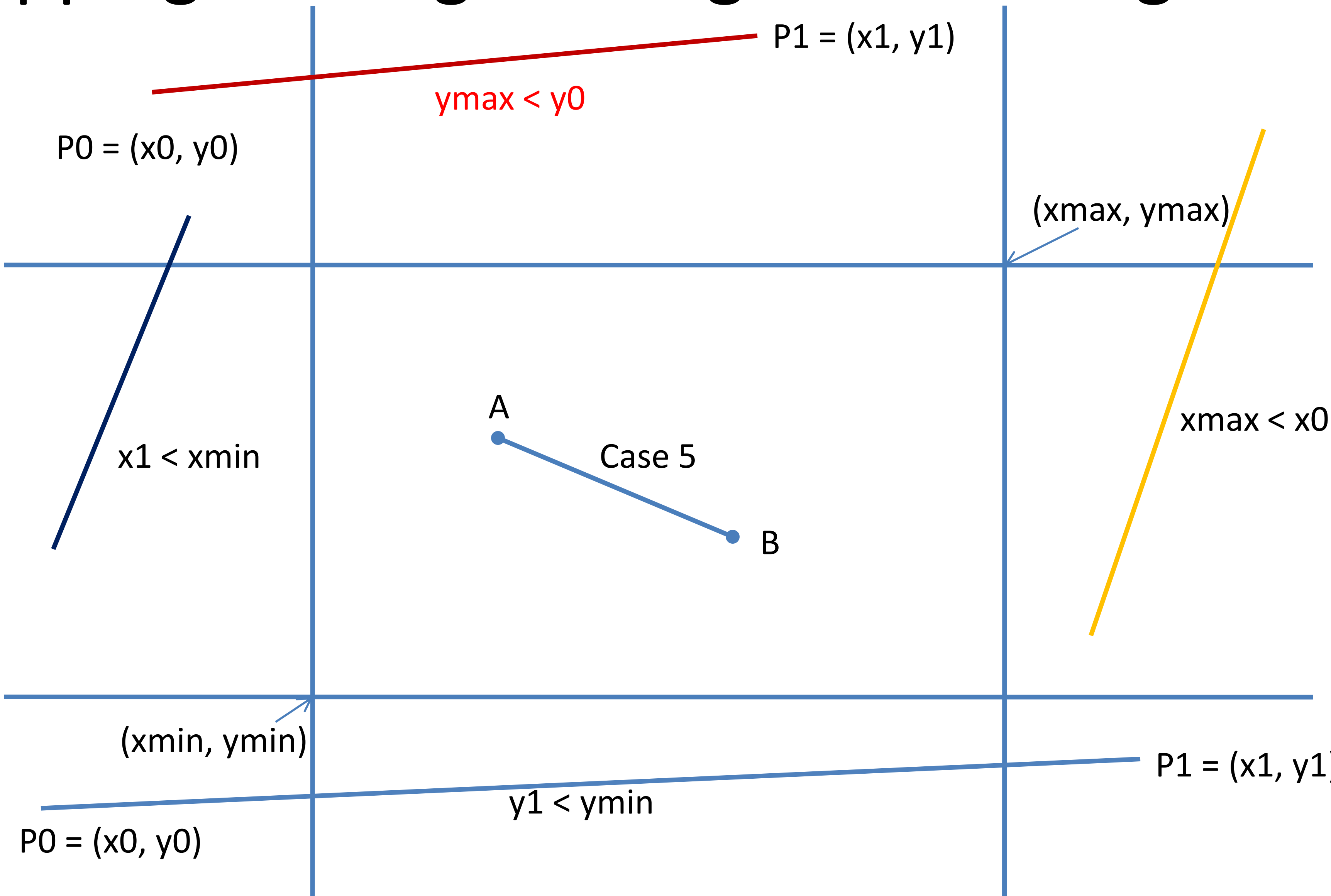
Some Simple Checks

- Line segment: (x0,y0) to (x1,y1)
  (assume that x0 < x1 and y0 < y1)
- Rectangle: (xmin,ymin) and (xmax,ymax)
- Special cases:
  - **Case 1**: x1 < xmin: no intersection
  - **Case 2**: xmax < x0: no intersection
  - **Case 3**: y1 < ymin: no intersection
  - **Case 4**: ymax < y0: no intersection
  - **Case 5**: xmin <= x0 < x1 <= xmax and
    ymin <= y0 < y1 <= ymax: inside

Clipping line segment against rectangle

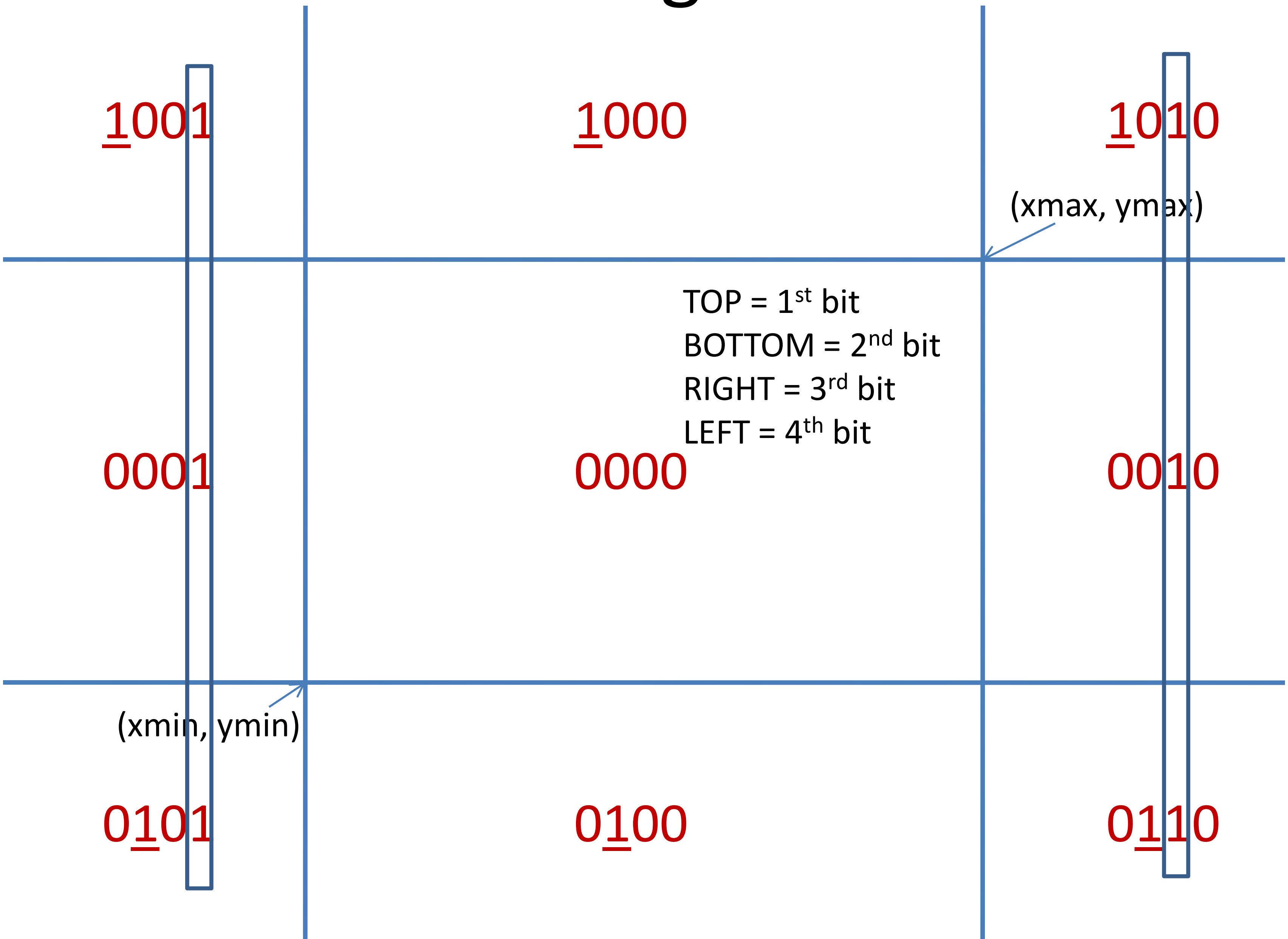


Clipping line segment against rectangle

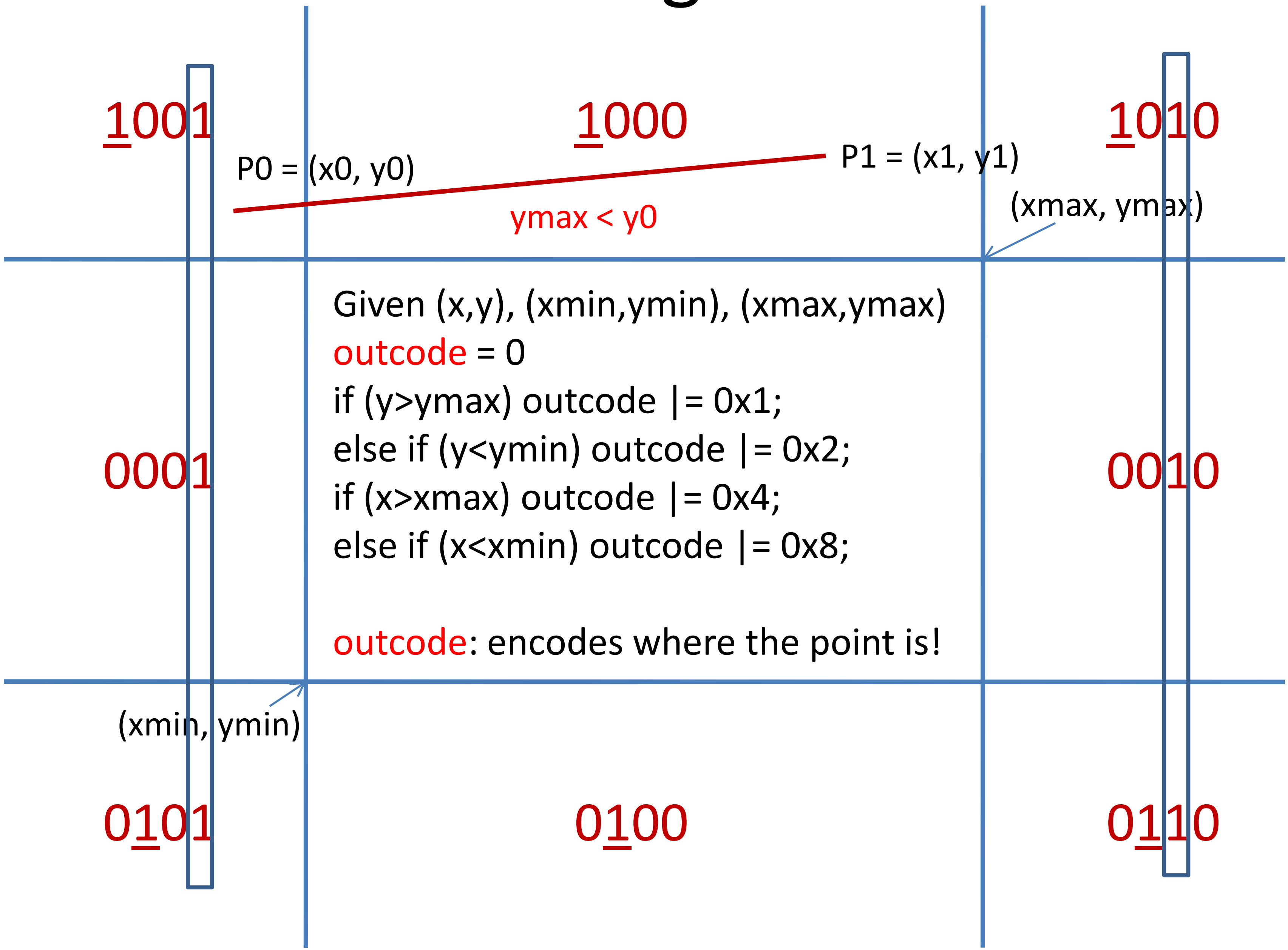


Maybe we can make this observations as part of the bigger picture!

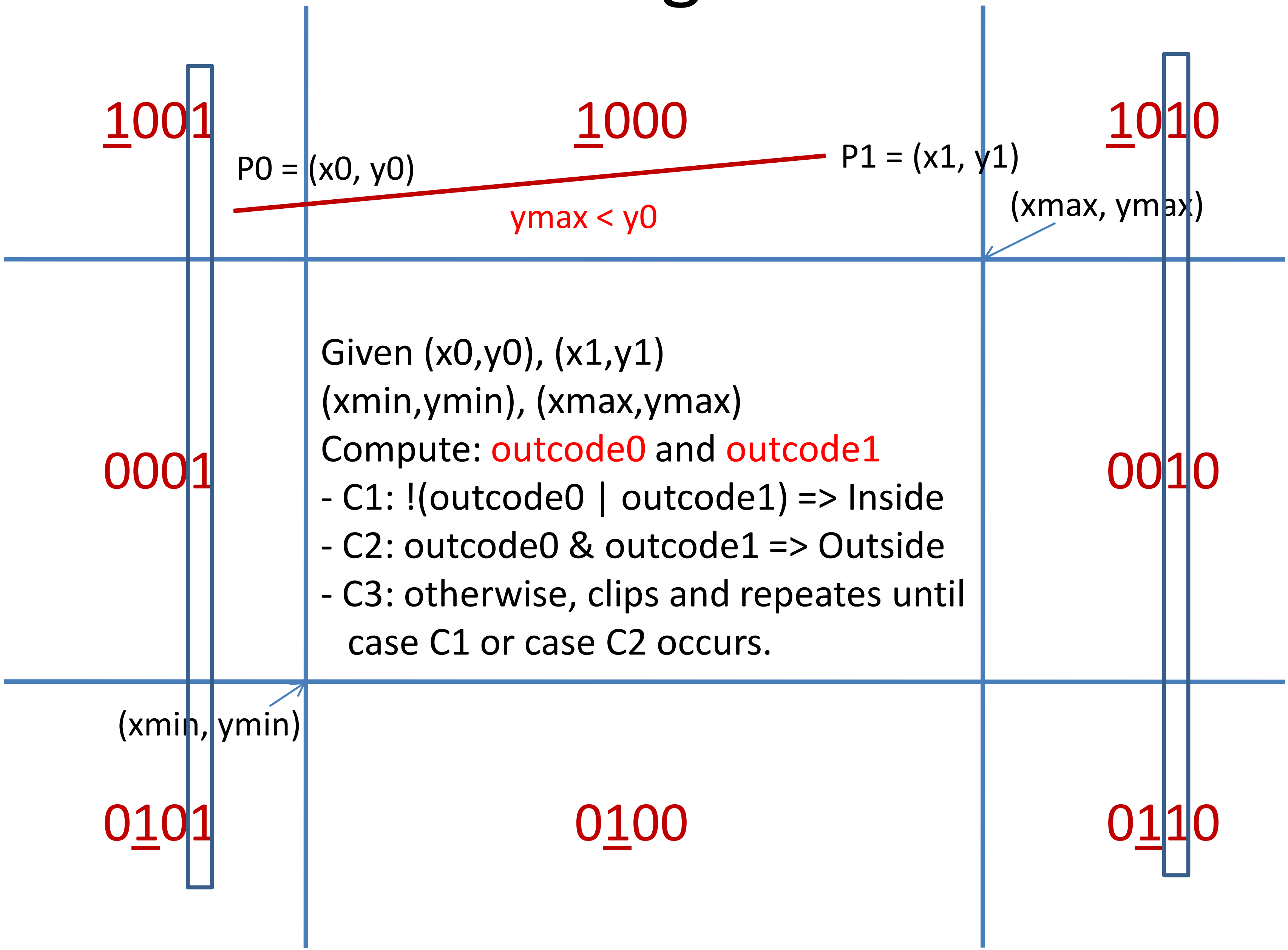
Cohen-Sutherland Algorithm



Cohen-Sutherland Algorithm



Cohen-Sutherland Algorithm



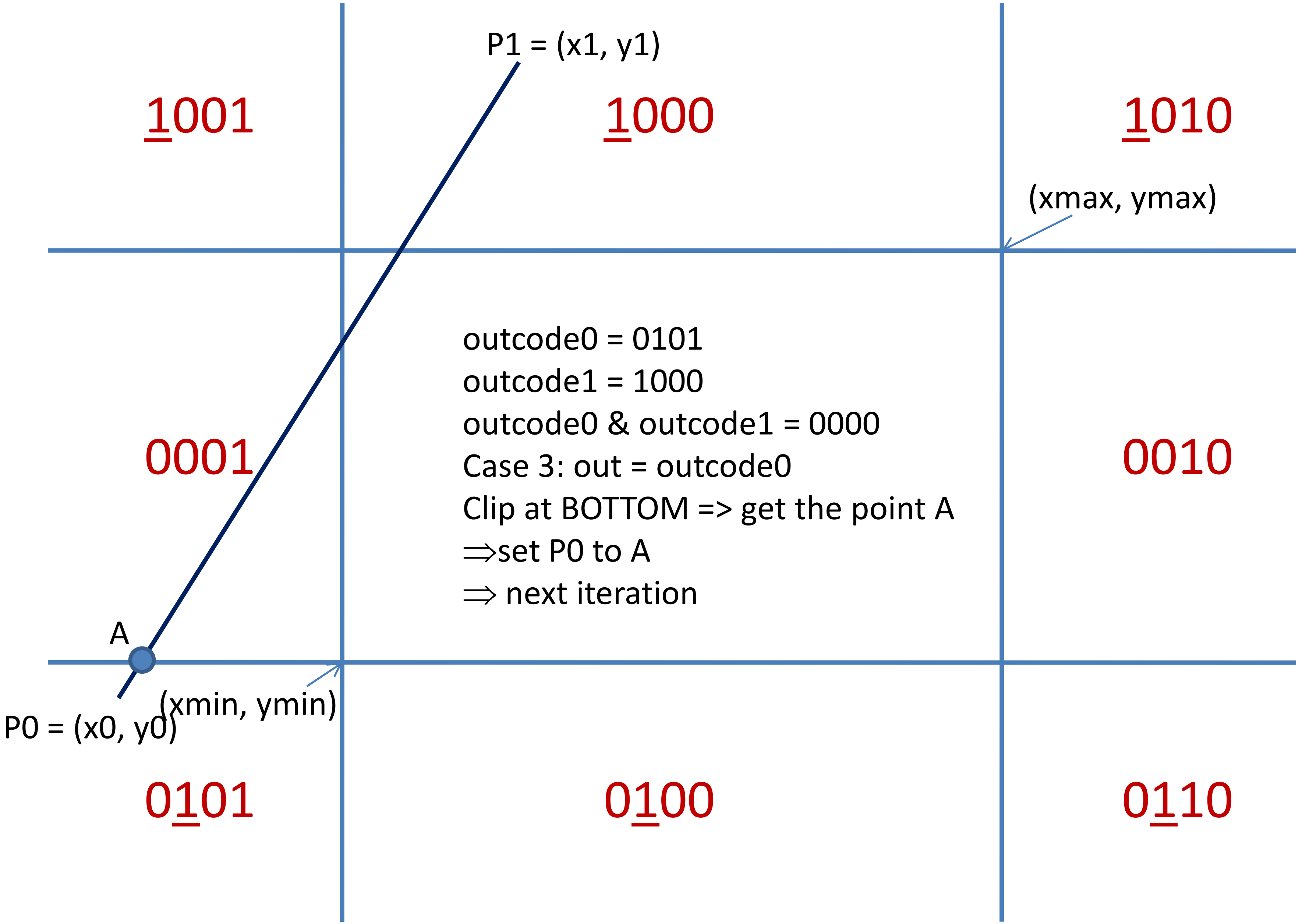
Cohen-Sutherland Algorithm

Given: (x0,y0), (x1,y1) and (xmin,ymin), (xmax,ymax)

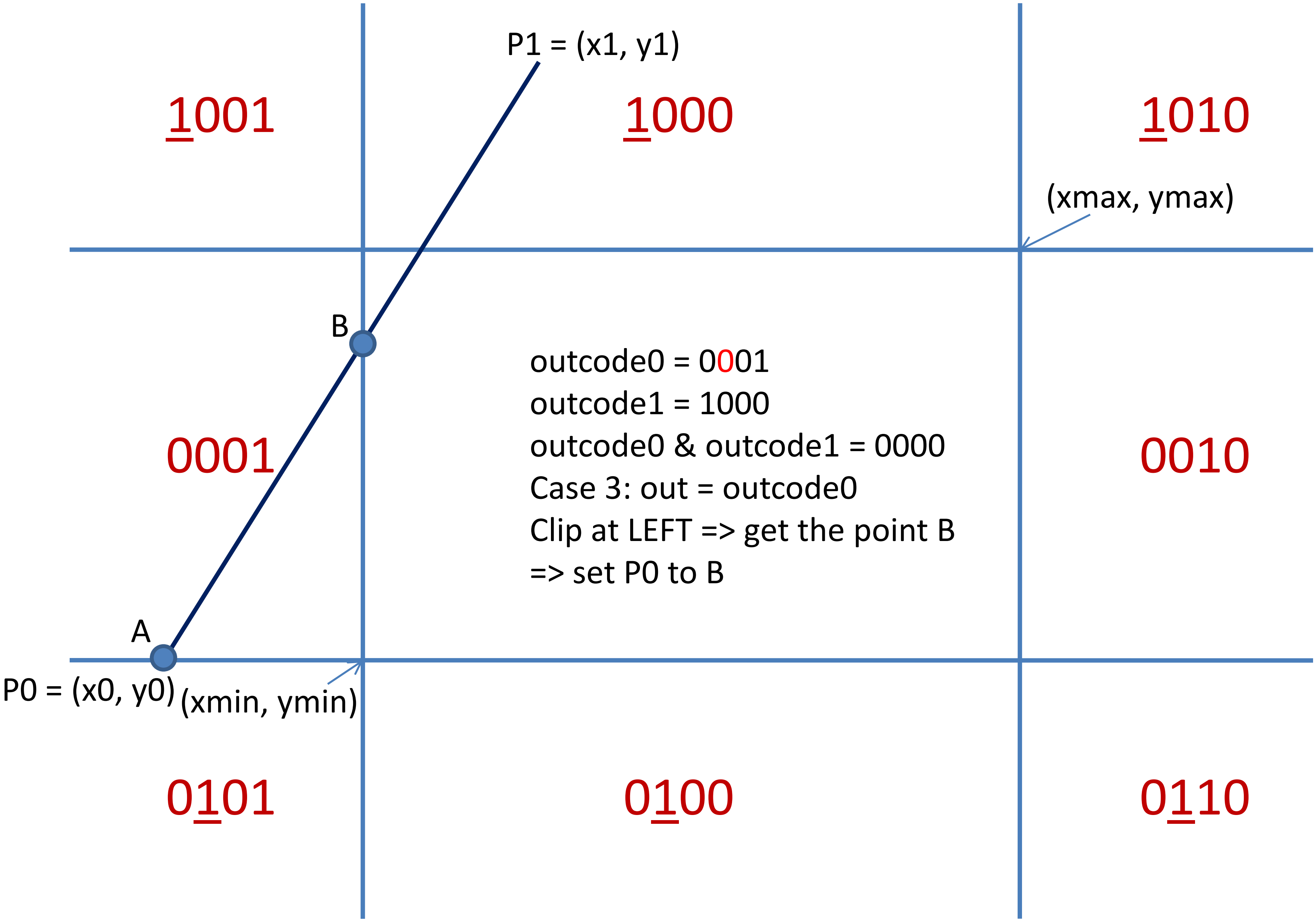
- Compute: outcode0 and outcode1
- C1: !(outcode0 | outcode1) => Inside
- C2: outcode0 & outcode1 => Outside
- C3: otherwise, clips and repeates until C1 or C2
  - set out = outcode0 ? outcode0 : outcode1;
  - if (out & 0x1) { x = x0+(x1-x0)*(ymax-y0)/(y1-y0) ;
    y = ymax} // clip on TOP

    .....
  - if (out==outcode0) {x0 = x; y0=y; => repeat} // clip from (x0,y0)
  - else {x1 = x; y1=y; => repeat} // clip from (x1,y1)

Cohen-Sutherland Algorithm – Example

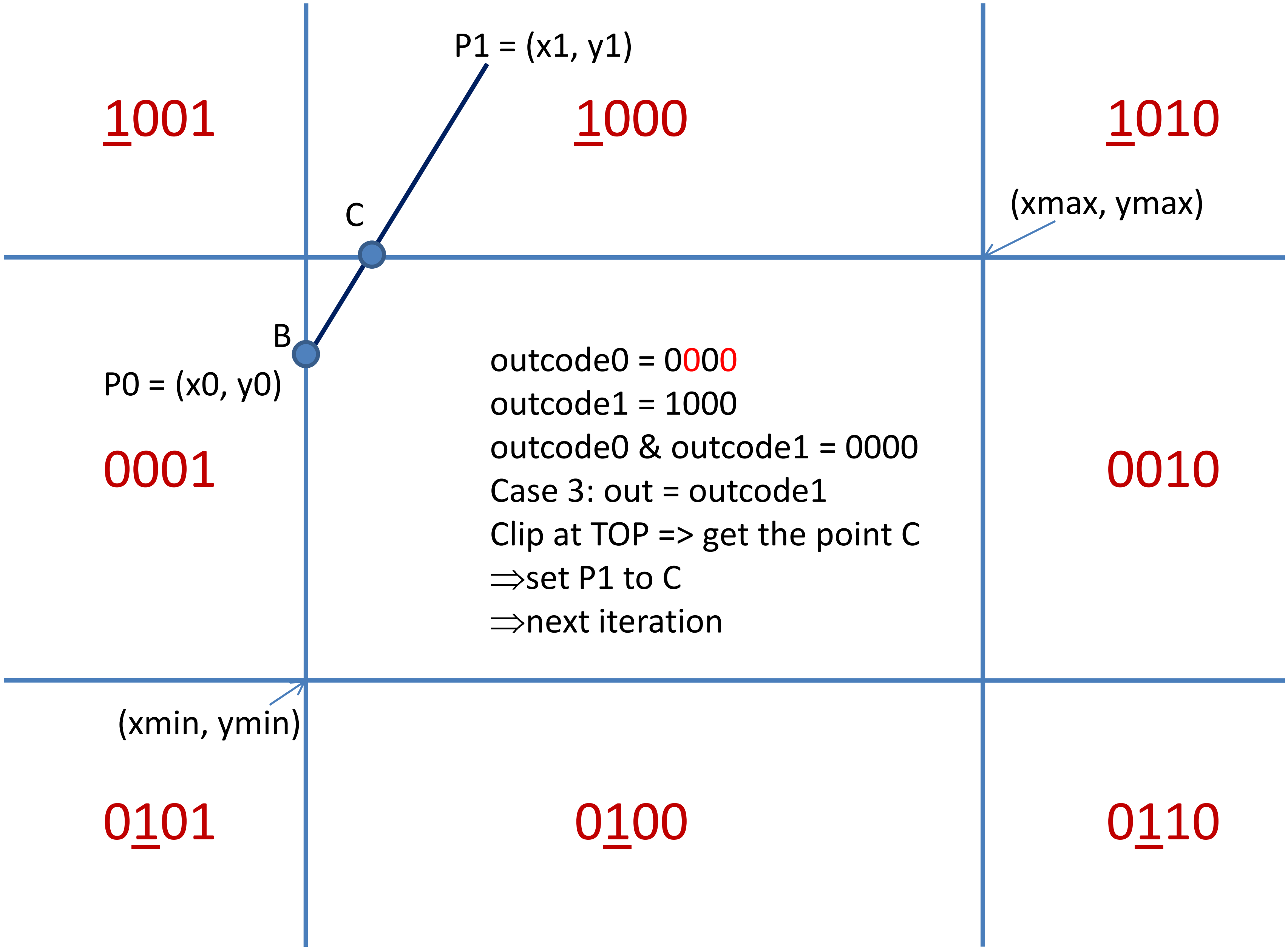


Cohen-Sutherland Algorithm – Example



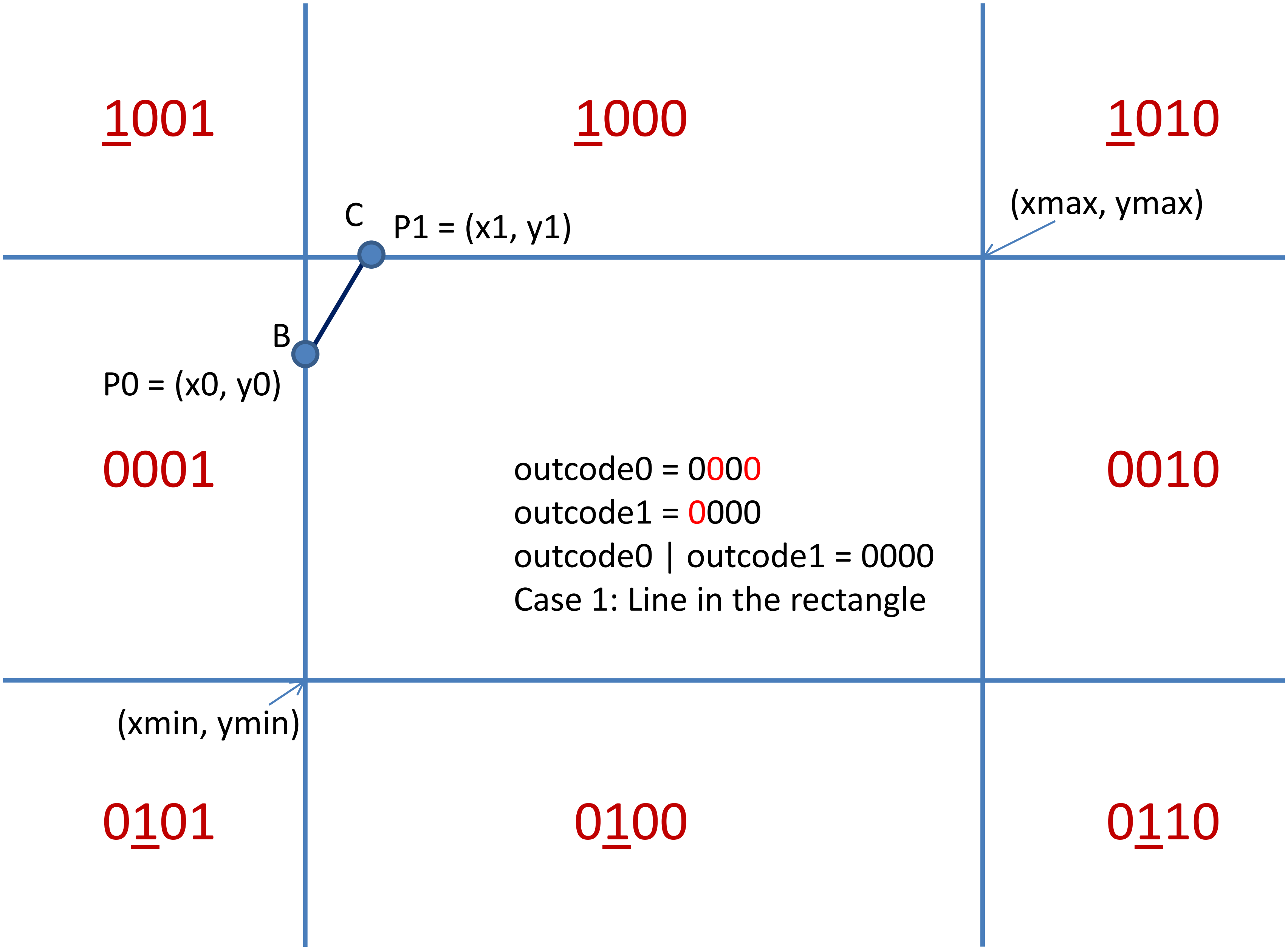
outcode0 = 0001
outcode1 = 1000
outcode0 & outcode1 = 0000
Case 3: out = outcode0
Clip at LEFT => get the point B
=> set P0 to B

Cohen-Sutherland Algorithm – Example



outcode0 = 0000
outcode1 = 1000
outcode0 & outcode1 = 0000
Case 3: out = outcode1
Clip at TOP => get the point C
⇒set P1 to C
⇒next iteration

Cohen-Sutherland Algorithm – Example



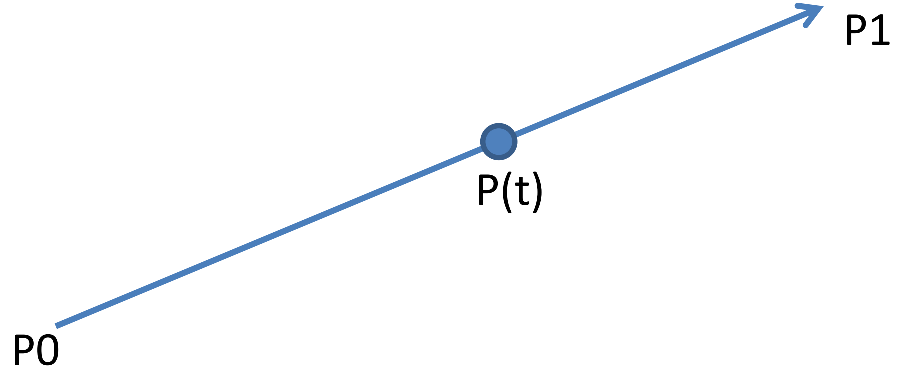
What good about Cohen-Sutherland algorithm?

- Simple
- We do not need to switch (x0,y0), (x1,y1)

Parametric Line Clipping Algorithm

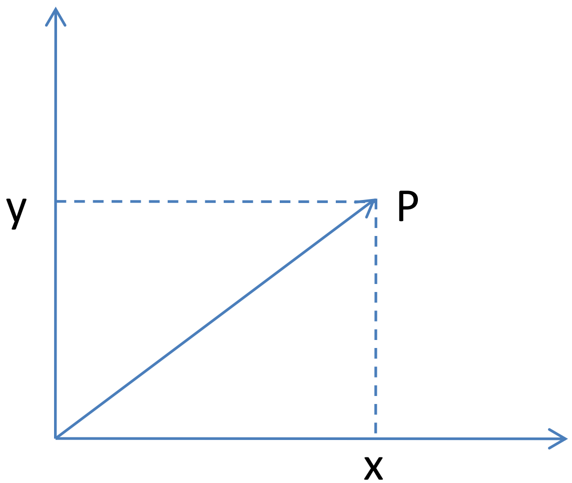
- Dot product of vectors
- Parametric line equation: P0 and P1 are vectors, 0 <= t <= 1

$$P(t) = P0 + (P1-P0)t$$



Vector Space

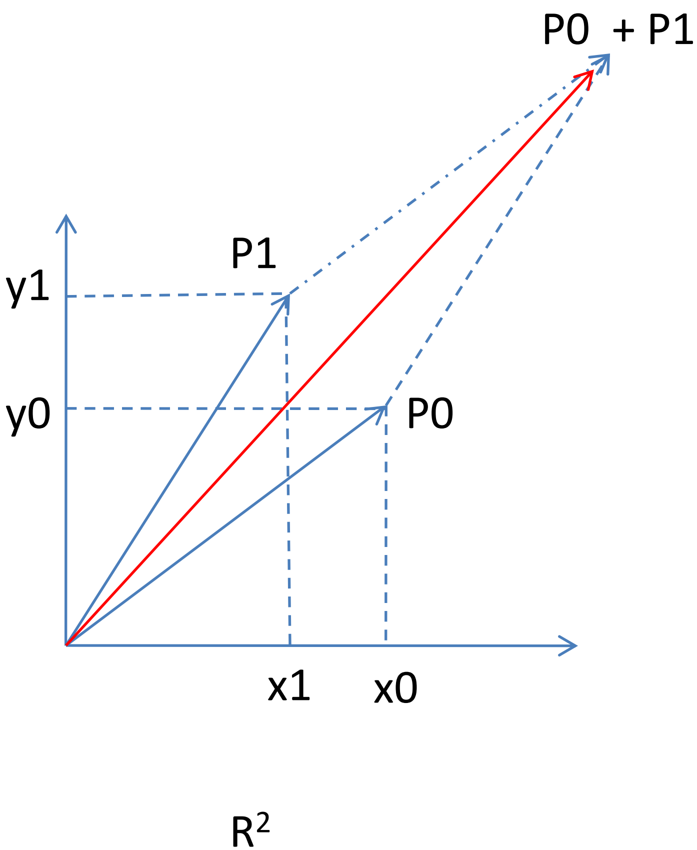
- A point in R^n can be viewed as a vector



R^2

$$P = \begin{bmatrix} x \\ y \end{bmatrix}$$

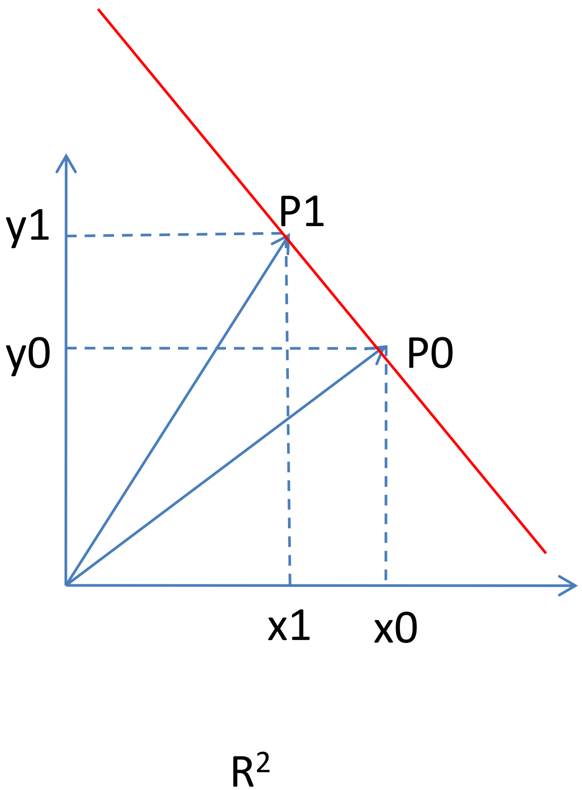
Addition of Vectors



$$P0 = \begin{pmatrix} X0 \\ y0 \end{pmatrix} \qquad P1 = \begin{pmatrix} X1 \\ y1 \end{pmatrix}$$

$$P0 + P1 = \begin{pmatrix} x0 + x1 \\ y0 + y1 \end{pmatrix}$$

Line Equation through P0 and P1



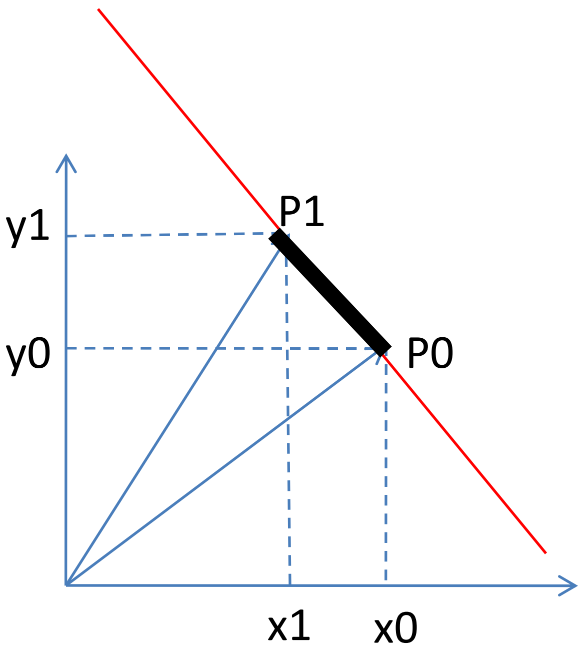
$$P0 = \begin{pmatrix} X0 \\ y0 \end{pmatrix} \qquad P1 = \begin{pmatrix} X1 \\ y1 \end{pmatrix}$$

$$(1\text{-}t)P0 + tP1 = \begin{pmatrix} (1\text{-}t)x0 + tx1 \\ (1\text{-}t)y0 + ty1 \end{pmatrix}$$

t is a real number

(Parametric Form of a Line through P0 and P1)

Line Equation through P0 and P1



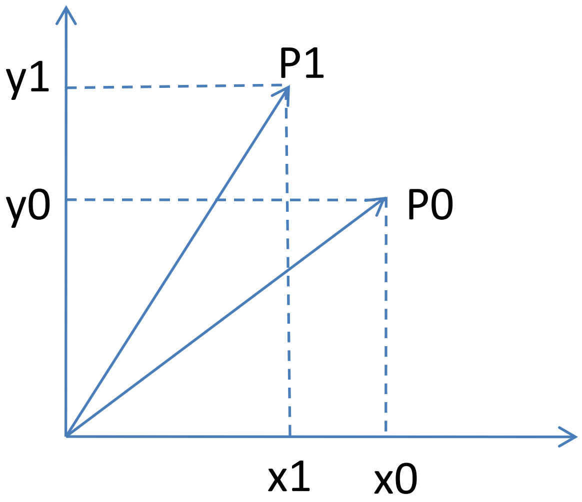
R^2

$$P0 = \begin{pmatrix} X0 \\ y0 \end{pmatrix} \qquad P1 = \begin{pmatrix} X1 \\ y1 \end{pmatrix}$$

$$(1\text{-}t)P0 + tP1 = \begin{pmatrix} (1\text{-}t)x0 + tx1 \\ (1\text{-}t)y0 + ty1 \end{pmatrix}$$

For the line segment from P0 to P1, $0 <= t <= 1$

Dot Product



R^2

$$P0 = \begin{pmatrix} X0 \\ y0 \end{pmatrix} \qquad P1 = \begin{pmatrix} X1 \\ y1 \end{pmatrix}$$

$P0 \bullet P1 = x0*x1 + y0*y1$

Dot product is not a vector – It is a scalar value (number)

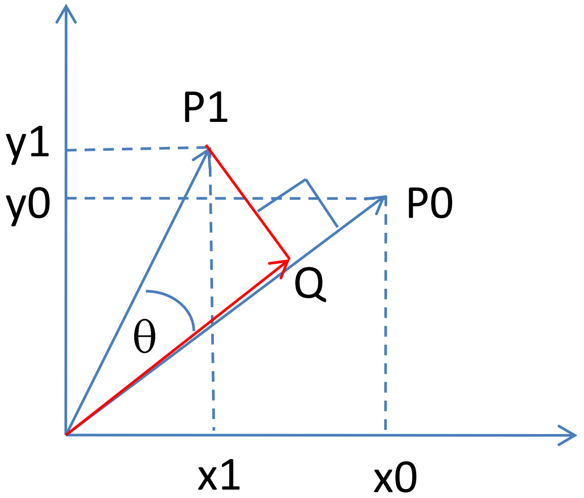
Distance from P0 to the origin (also called *length*) is

$||P0|| = sqrt(x0^2+y0^2)$ or $||P0|| = sqrt(P0 \bullet P0)$

Vector with length 1 is called *normalized (unit)* vector

$w/||w||$ is always a normalized vector

Dot Product



$$P0 = \begin{bmatrix} X0 \\ y0 \end{bmatrix} \qquad P1 = \begin{bmatrix} X1 \\ y1 \end{bmatrix}$$

P0 • P1 = x0*x1 + y0*y1

R^2 If P0 is a unit vector then Q is the cosine of the angle between P0 and P1

$$\cos^{-1} = \frac{P0 \bullet P1}{\| P0 \| \cdot \| P1 \|}$$

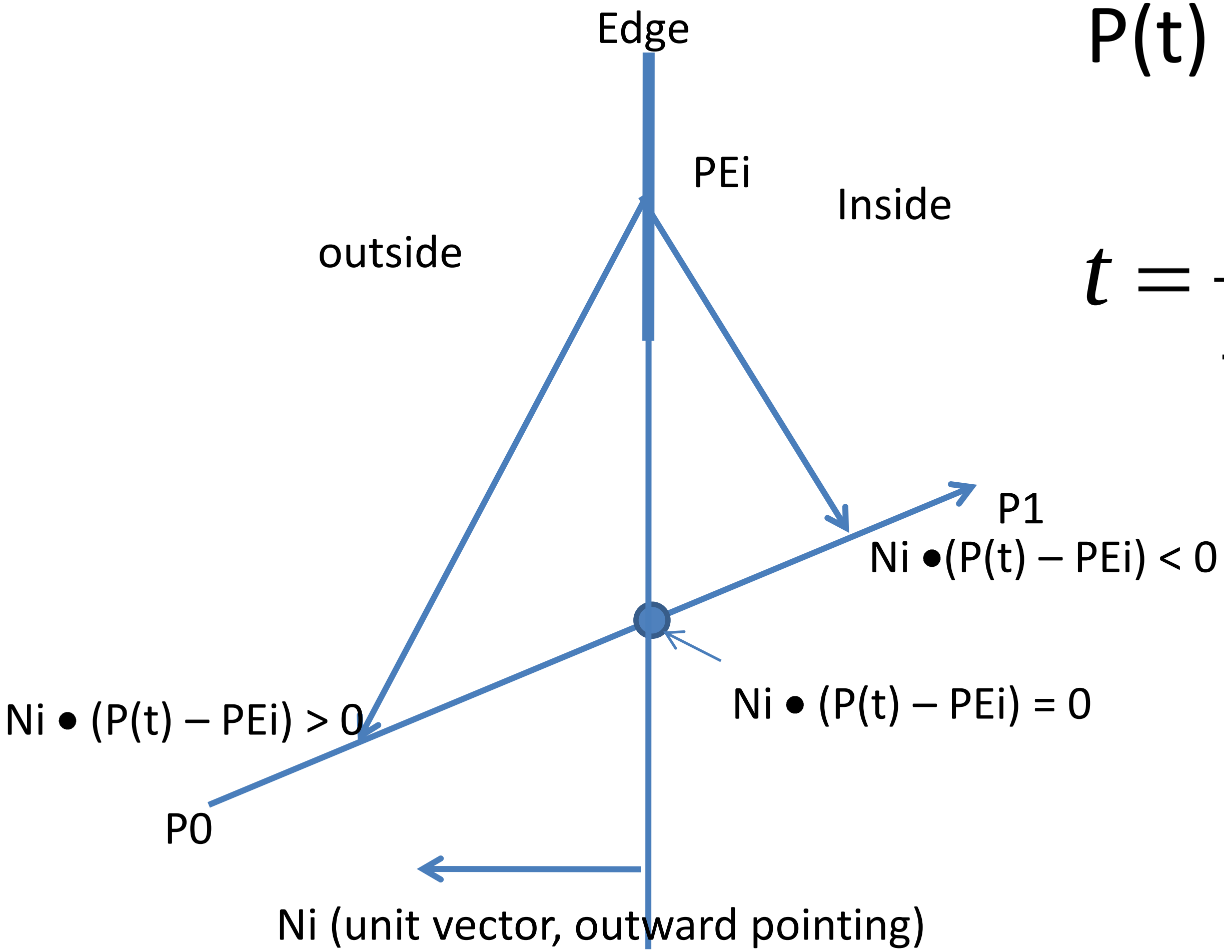
Application of Dot Product

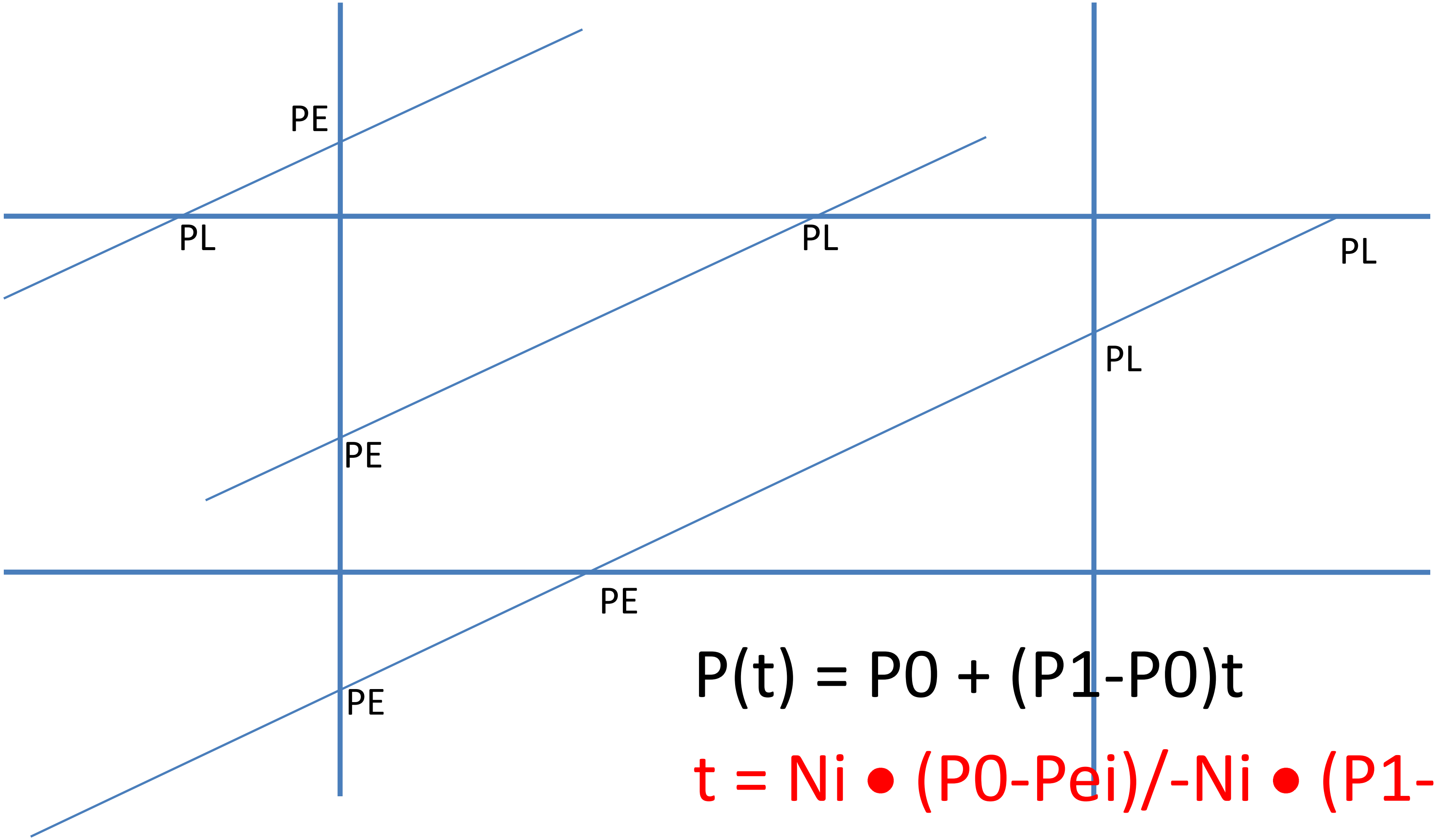
- Defining circle: $||Q - P|| = r$
- Defining line going through P and perpendicular to a vector v: $(Q - P) \bullet v = 0$

Parametric Line Clipping Algorithm

$$P(t) = P0 + (P1-P0)t$$

$$t = \frac{Ni \bullet (P0 - Pei)}{-Ni \bullet (P1 - P0)}$$





$$P(t) = P0 + (P1-P0)t$$

$$t = Ni \bullet (P0-Pei)/-Ni \bullet (P1-P0)$$

t < 0 or t > 1: intersection outside segment

Intersections can be classified by the sign of Ni • D where D = P0-Pei

If Ni • D < 0 the intersection is inside=>outside (potential leaving - PL)

If Ni • D > 0 the intersection is outside=>inside (potential entering - PE)

Algorithm

- Need to choose Ni, Pei

Clip Edge Ei	Ni	PEi	P0-PEi	t
LEFT x=xmin	(-1,0)	(xmin,y)	(x0-xmin,y0-y)	-(x0-xmin)/(x1-x0)
RIGHT x=xmax	(1,0)	(xmax,y)	(x0-xmax,y0-y)	(x0-xmax)/-(x1-x0)
BOTTOM y=ymin	(0,-1)	(x,ymin)	(x0-x,y0-ymin)	-(y0-ymin)/(y1-y0)
TOP y=ymax	(0,1)	(x,ymax)	(x0-x,y0-ymax)	(y0-ymax)/-(y1-y0)

- The algorithm then follows the idea of the Cohen-Sutherland algorithm, computing PE and PL intersections, moving the endpoints P0 and P1 until the entire segment is inside or outside the rectangle

Clipping Circles and Ellipses

- Trivial acceptance/rejection test (inside/outside)
- Test whether the extent of circle intersects with rectangle (could be testing for quadrant, octant)
- Compute the intersections
- Use scan conversion to draw the arc (or fill the arc)

Clipping Polygons (Sutherland-Hodgman)

- Idea: Clip the polygon against each *extended* edge (edge extended on both endpoints to become a line)
- Detail: see book

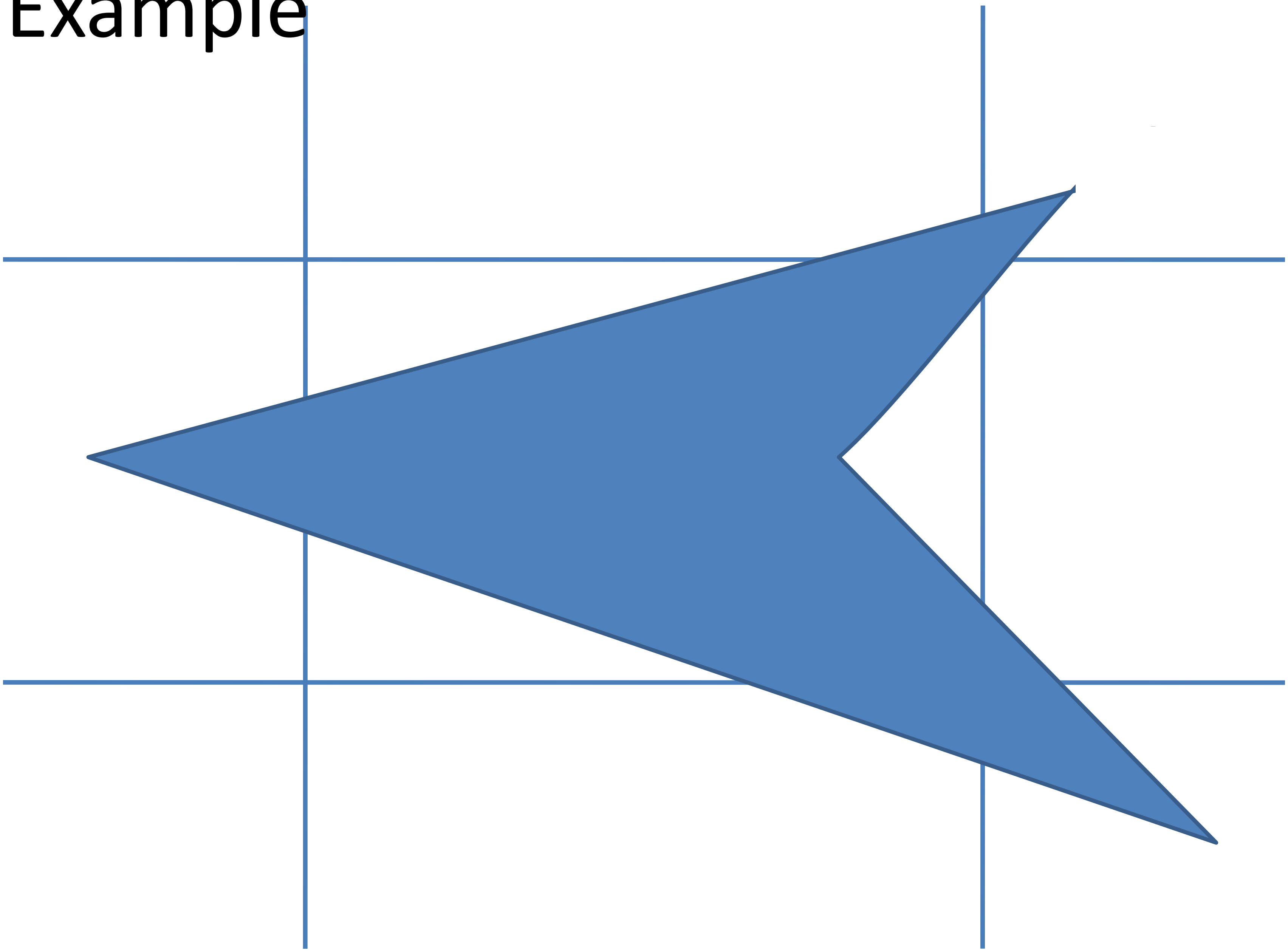
Clipping Polygons (Sutherland-Hodgman)

Given: Polygon v1, v1, … , vn

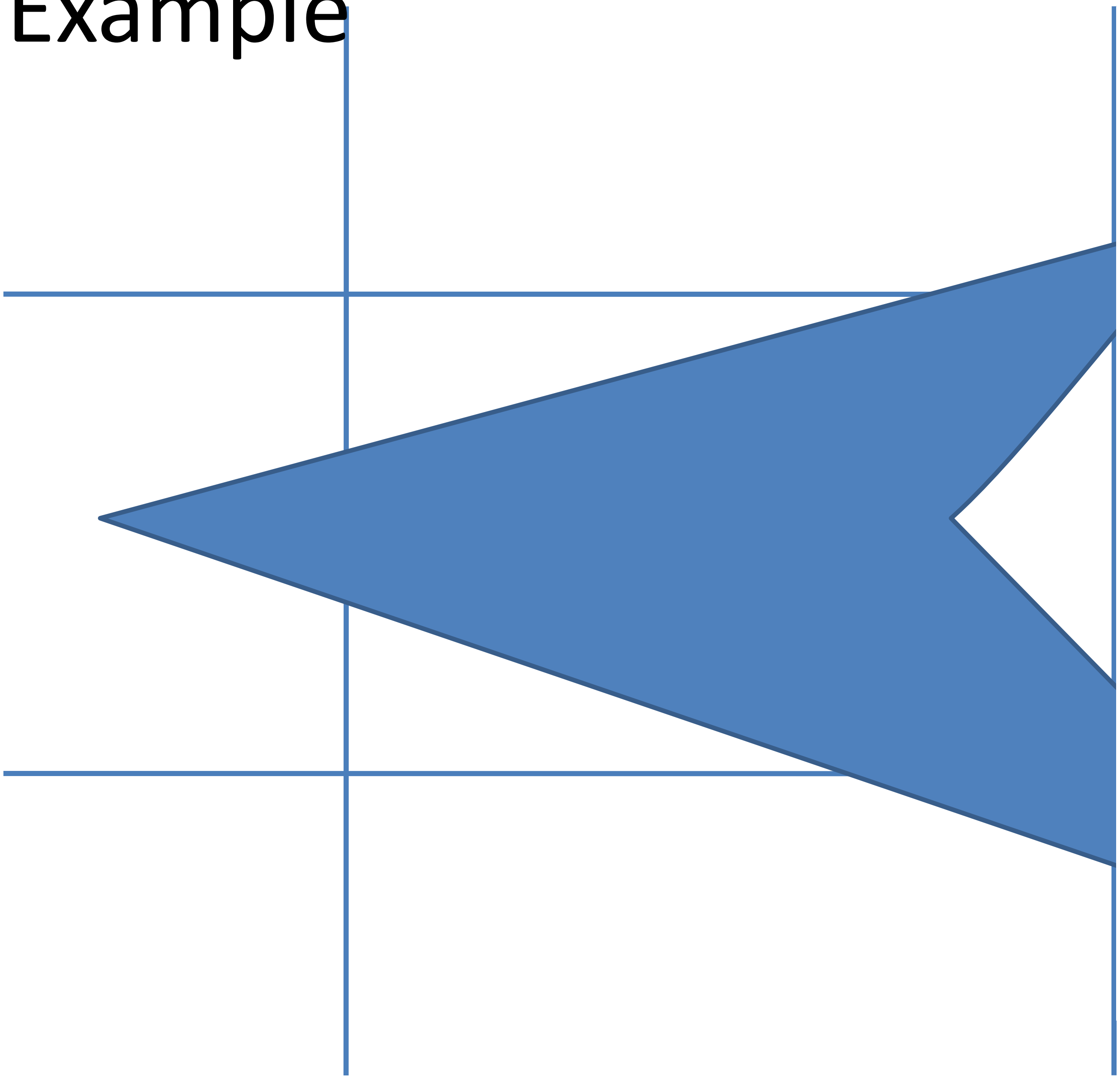
i = n (move from vn to v1, v2, …, vn)

```
for (i1 = 1; i1 < n; i1 ++) {
    Move from vi to vi1
    Clip the segment (zero, one, or two new
    vertices are added to the output list)
    i = i1
}
```

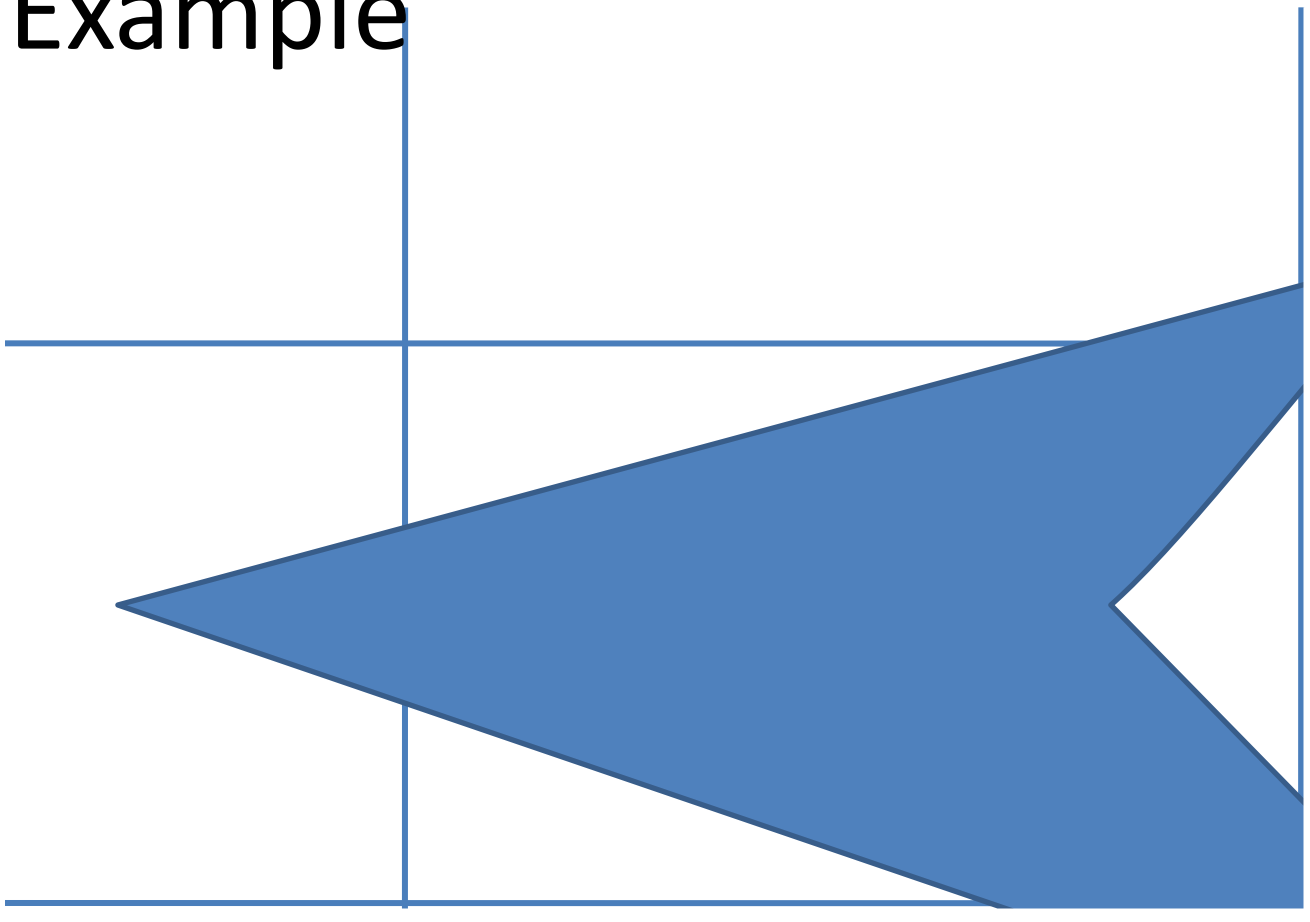
Example



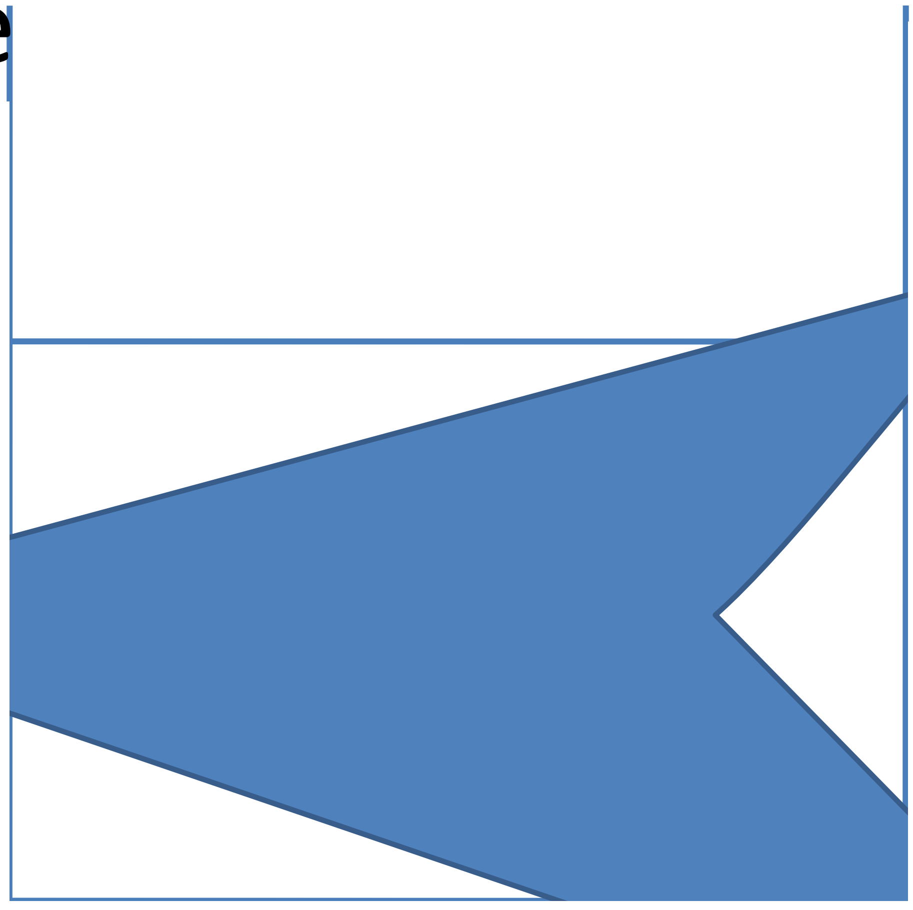
Example



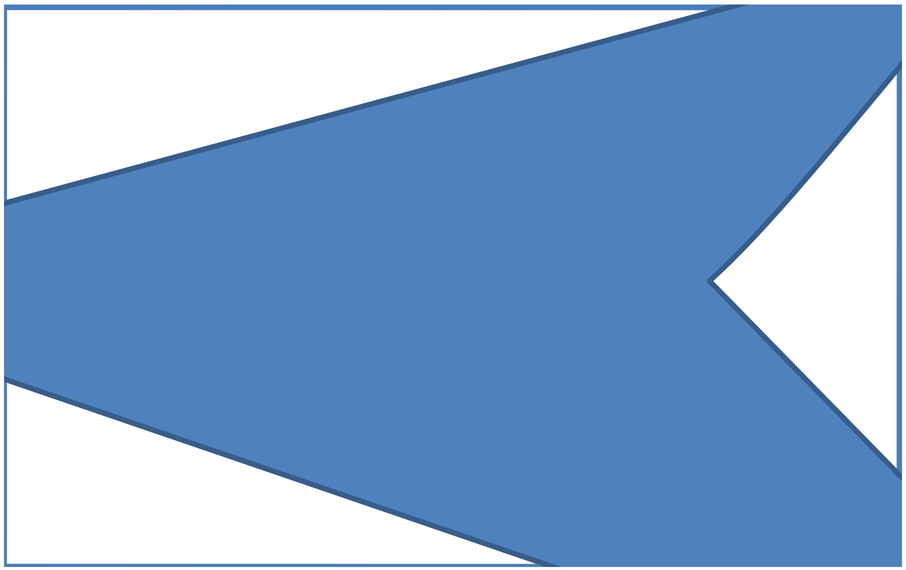
Example



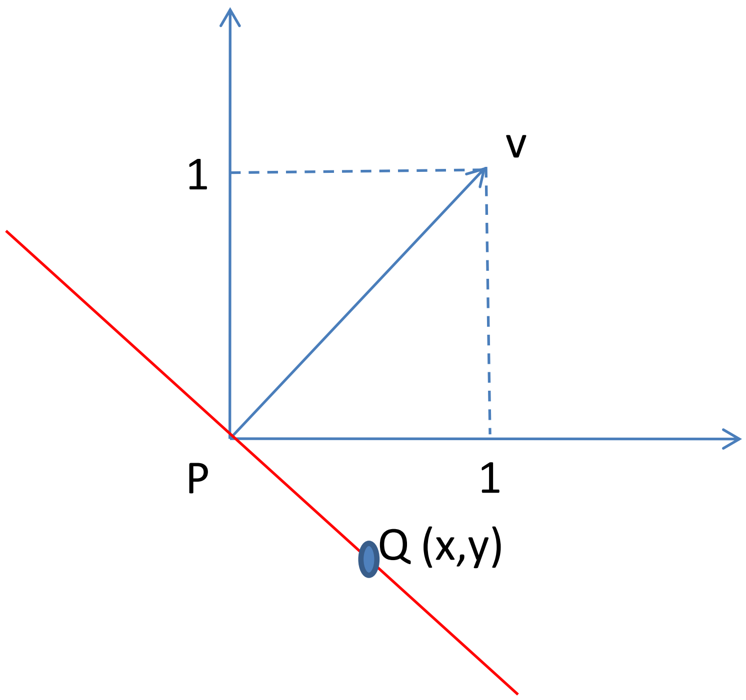
Example



Example



Few Examples of Dot Product

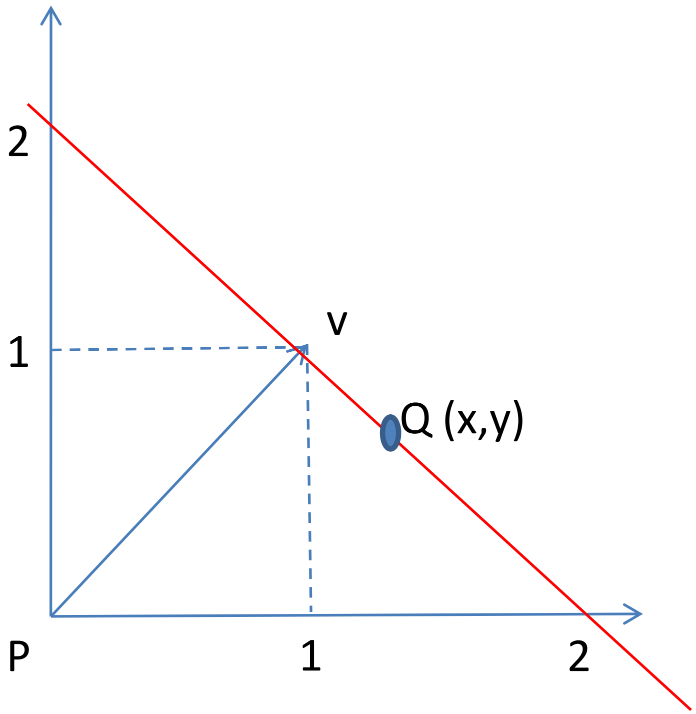


$$P = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \qquad v = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

$(Q\text{-}P) \bullet v = 0$

$$\begin{bmatrix} x-0 \\ y-0 \end{bmatrix} \bullet \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 0 \Rightarrow x + y = 0$$

Few Examples of Dot Product



$$P = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \qquad v = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

(Q-P) • v = 0

$$\begin{bmatrix} x-1 \\ y-1 \end{bmatrix} \bullet \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 0 \Rightarrow x + y = 2$$