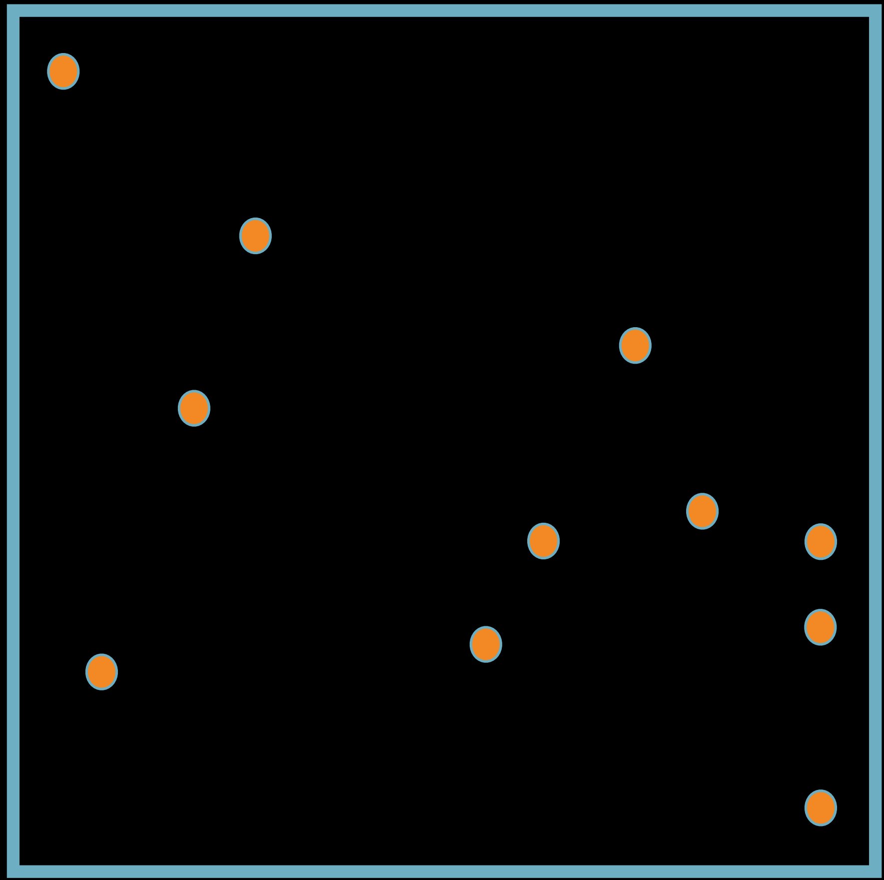
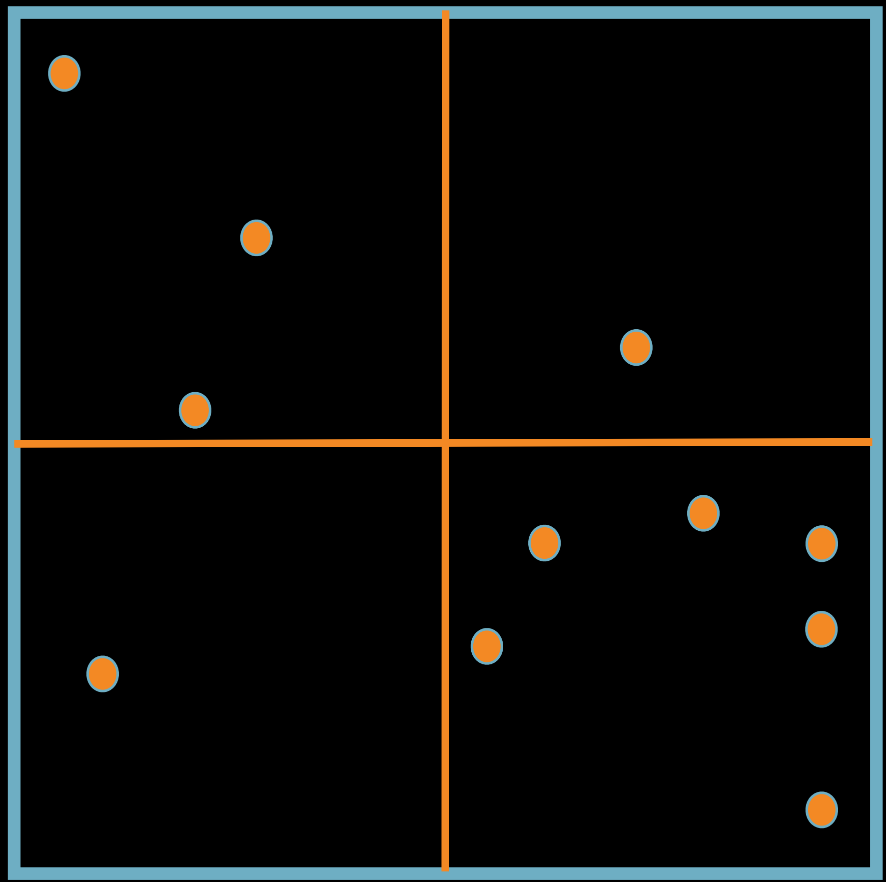
Quadtrees

João Comba

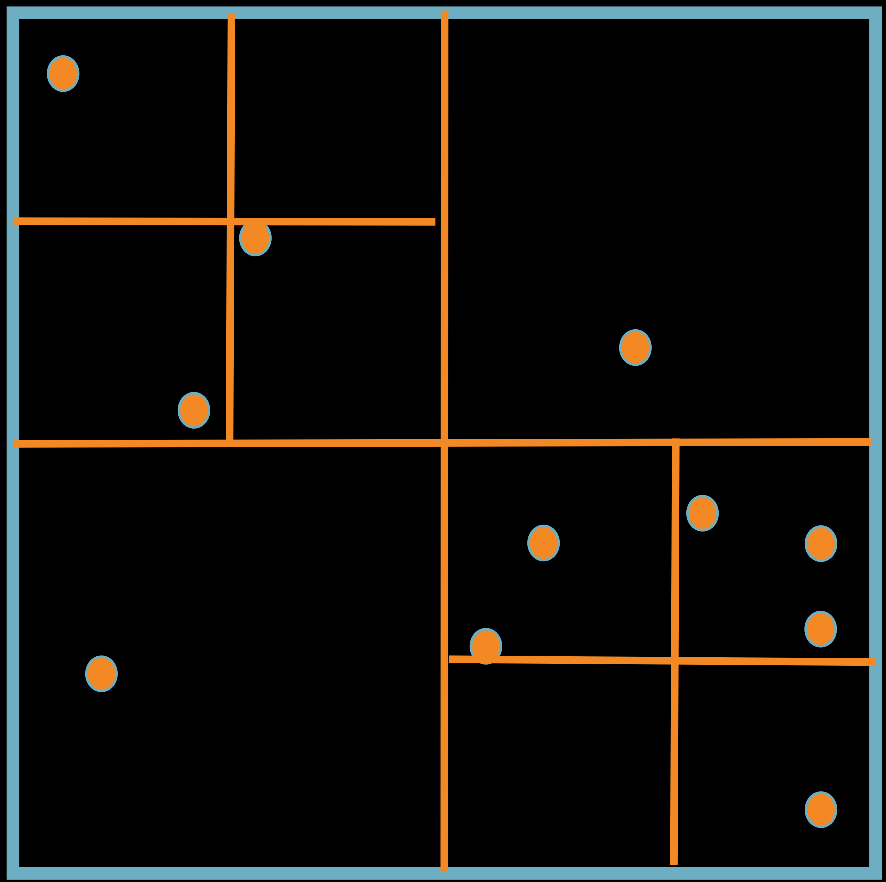
Quadtrees



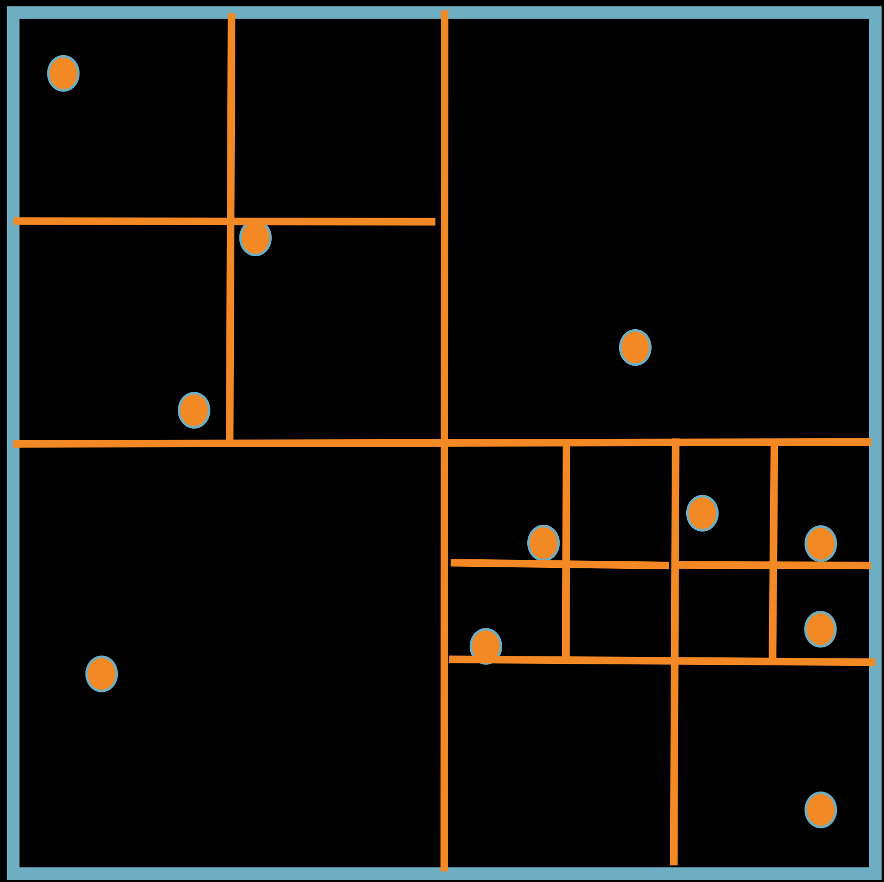
Quadtrees



Quadtrees



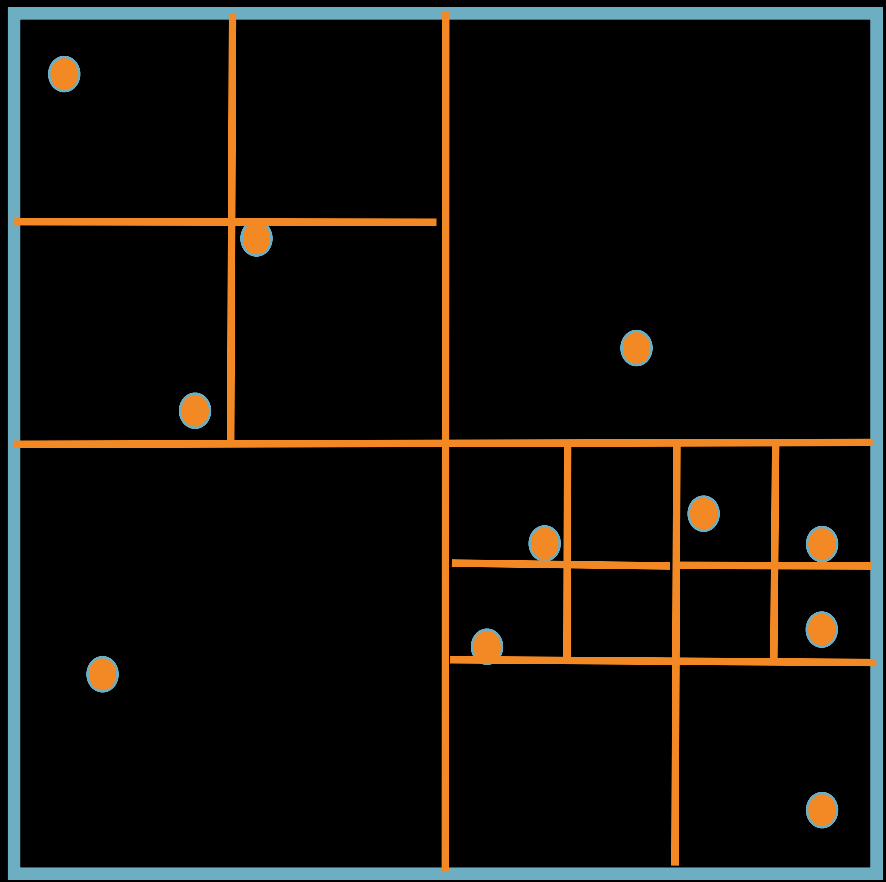
Quadtrees

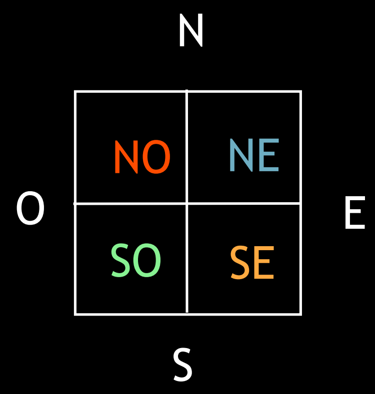


Quadtrees

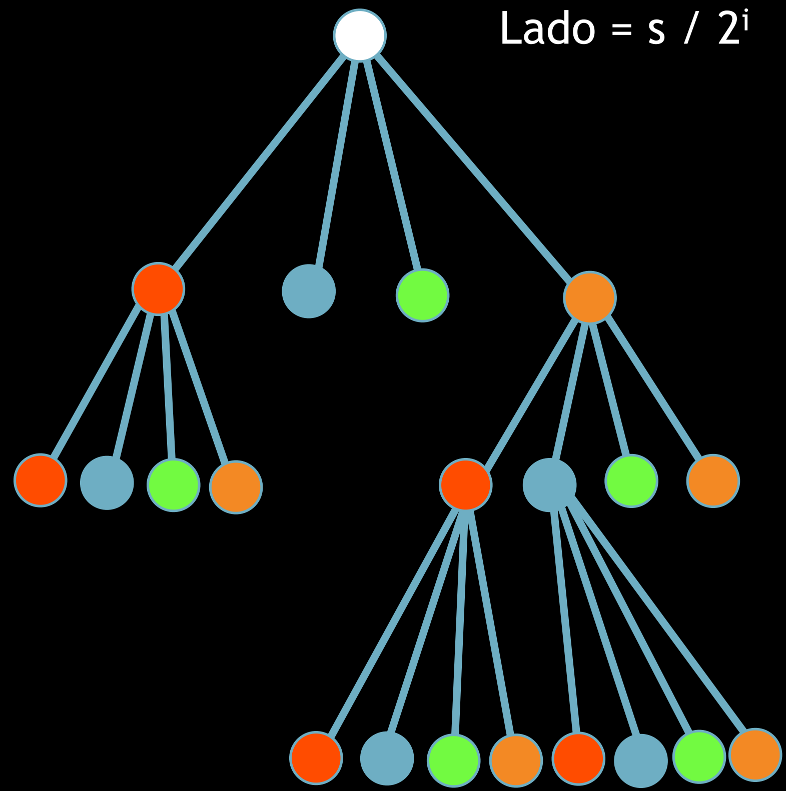
- Tipo de dados representados (pontos, linhas, regioes, volumes, outros)
- Principio guiando a subdivisao
- Resolucao (variavel ou nao)

Quadtrees Pontuais



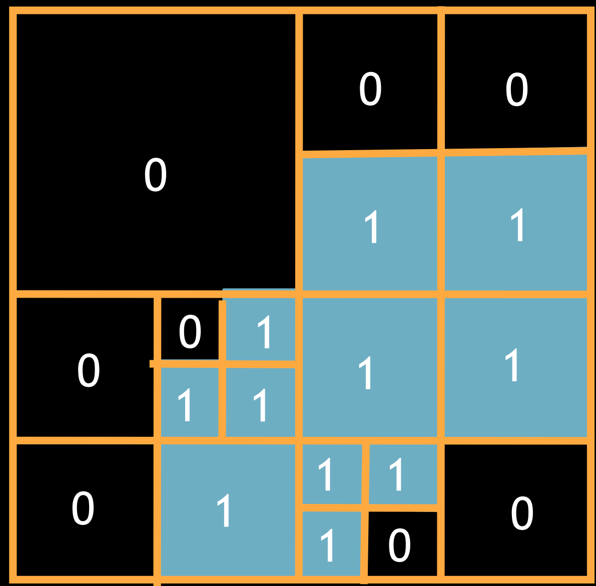


Lado = $s / 2^i$

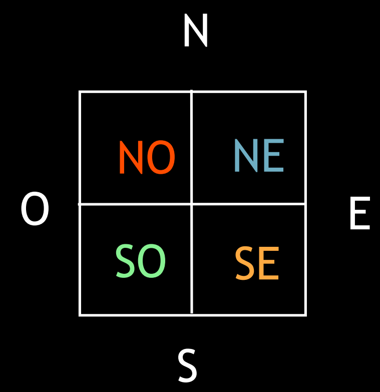


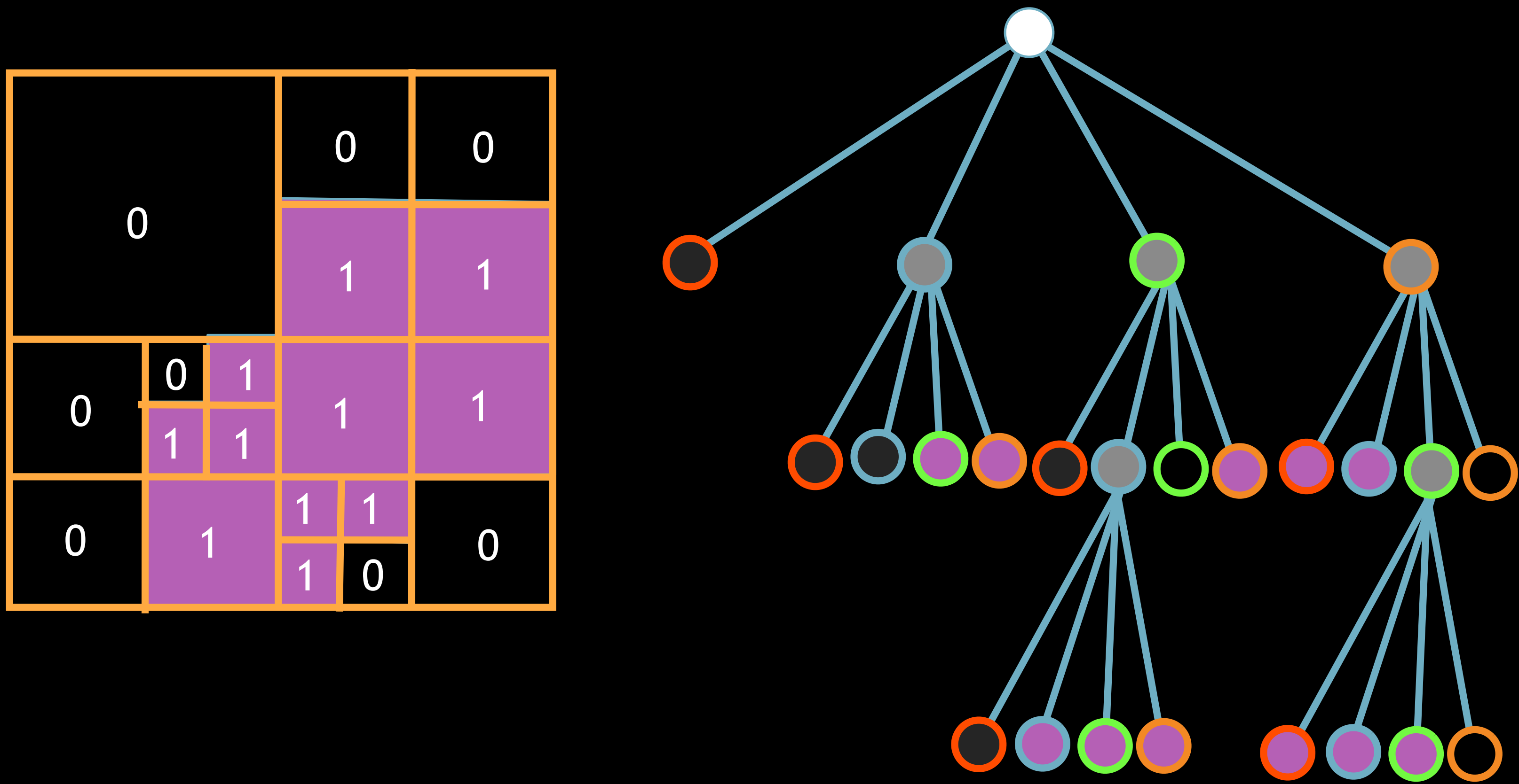
Quadtrees de Regioes

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1
0	0	0	0	1	1	1	1
0	0	0	1	1	1	1	1
0	0	1	1	1	1	1	1
0	0	1	1	1	1	0	0
0	0	1	1	1	0	0	0

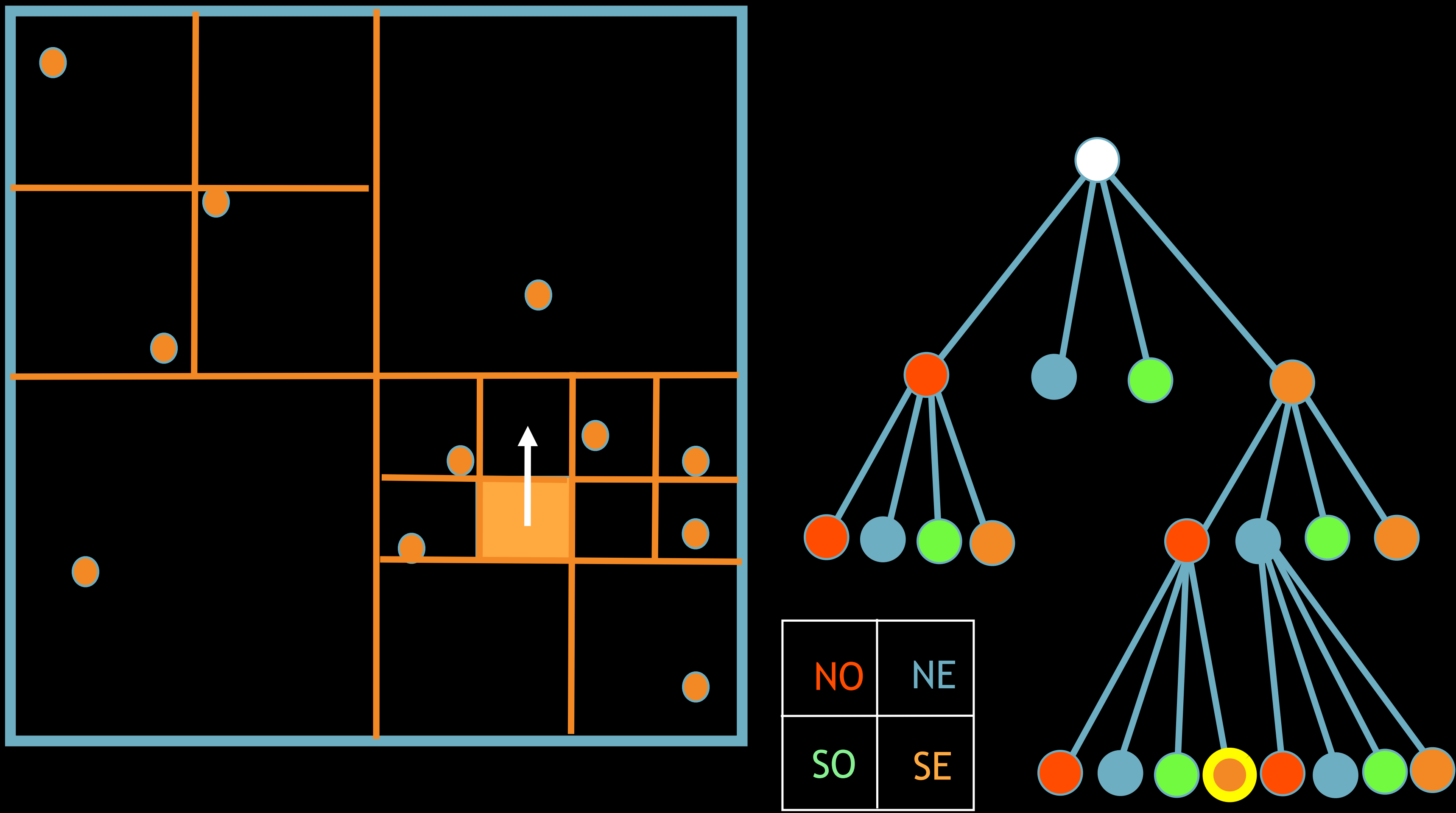


Quadtrees de Regioes

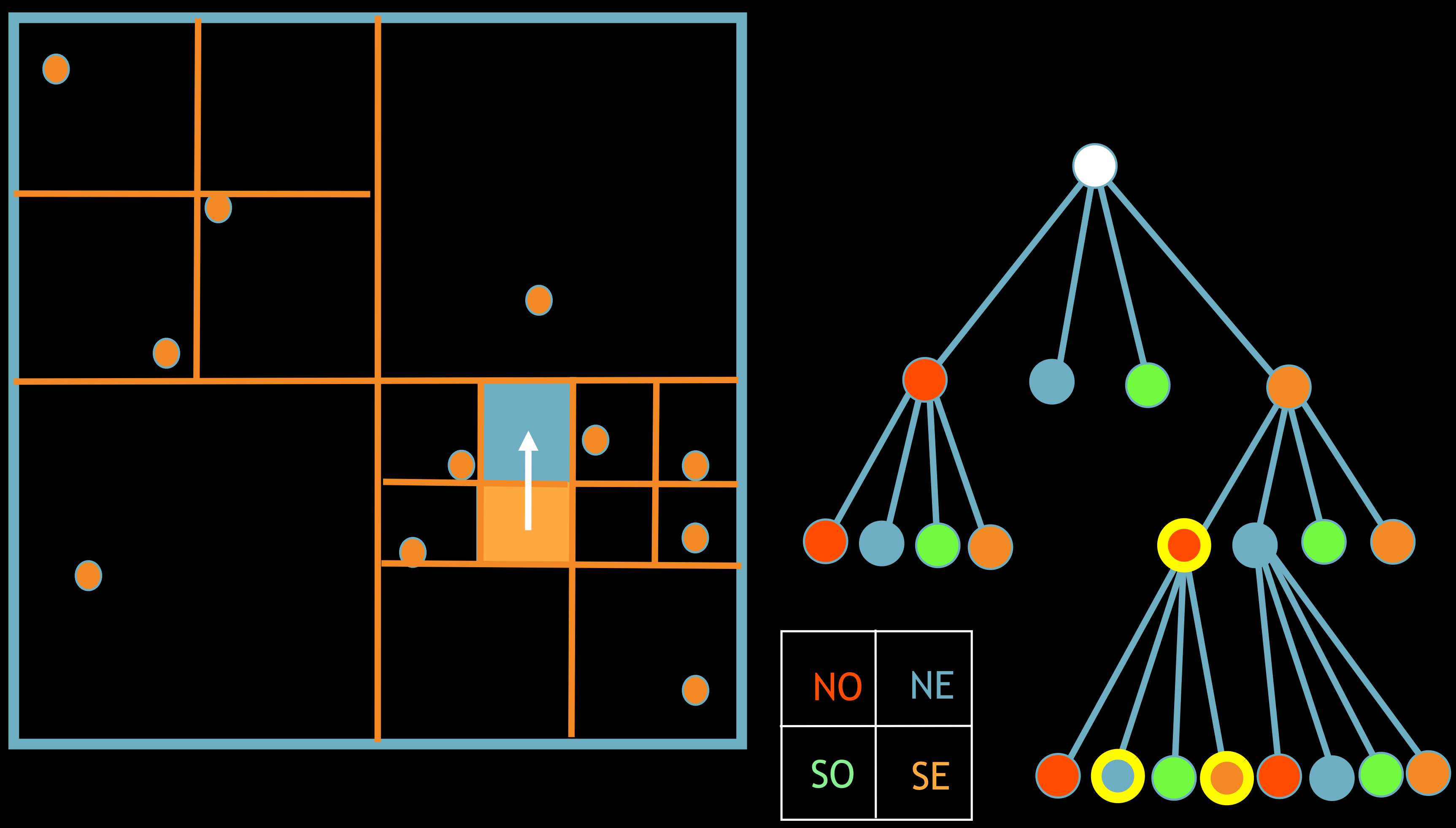




Busca de Vizinhos

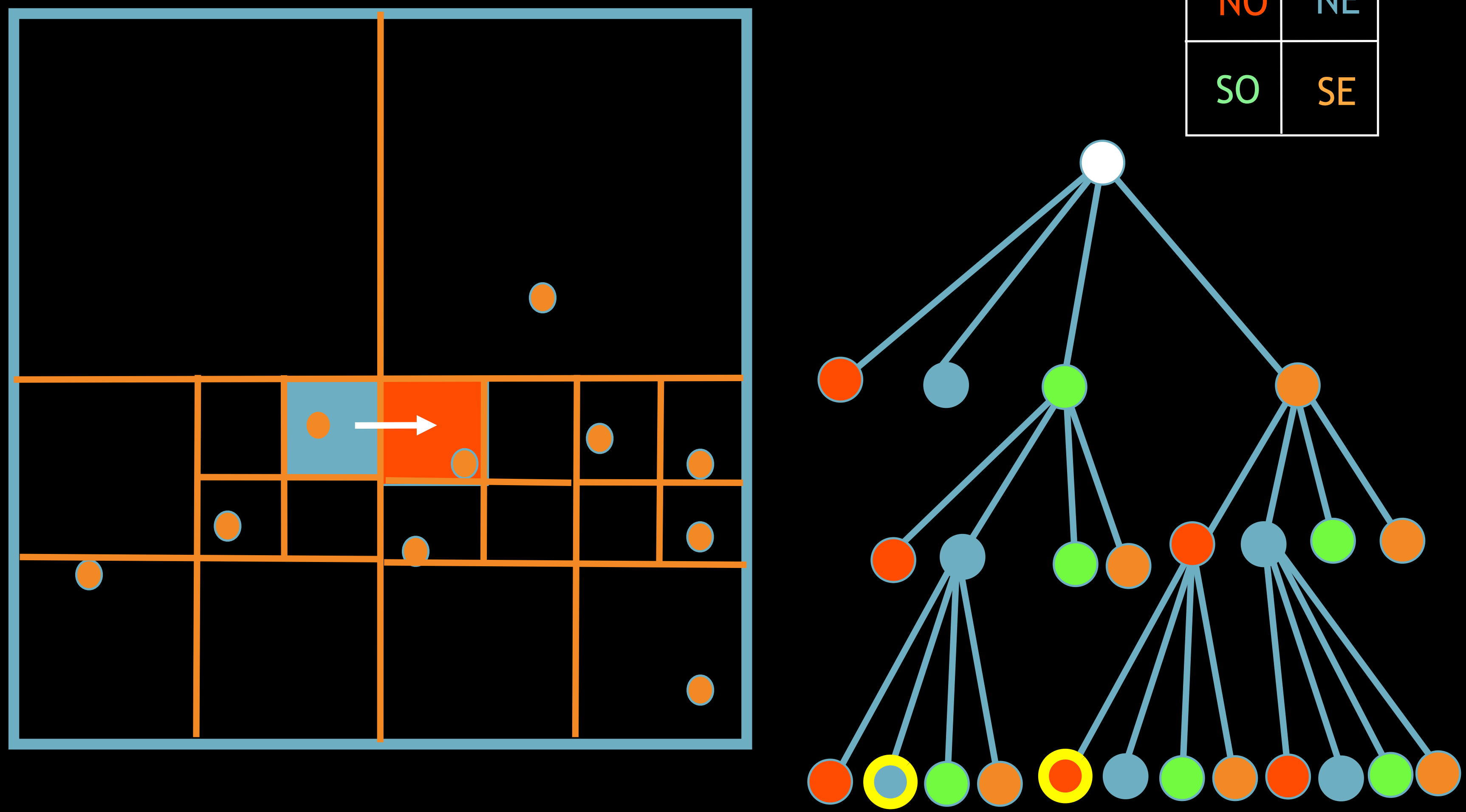


Busca de Vizinhos



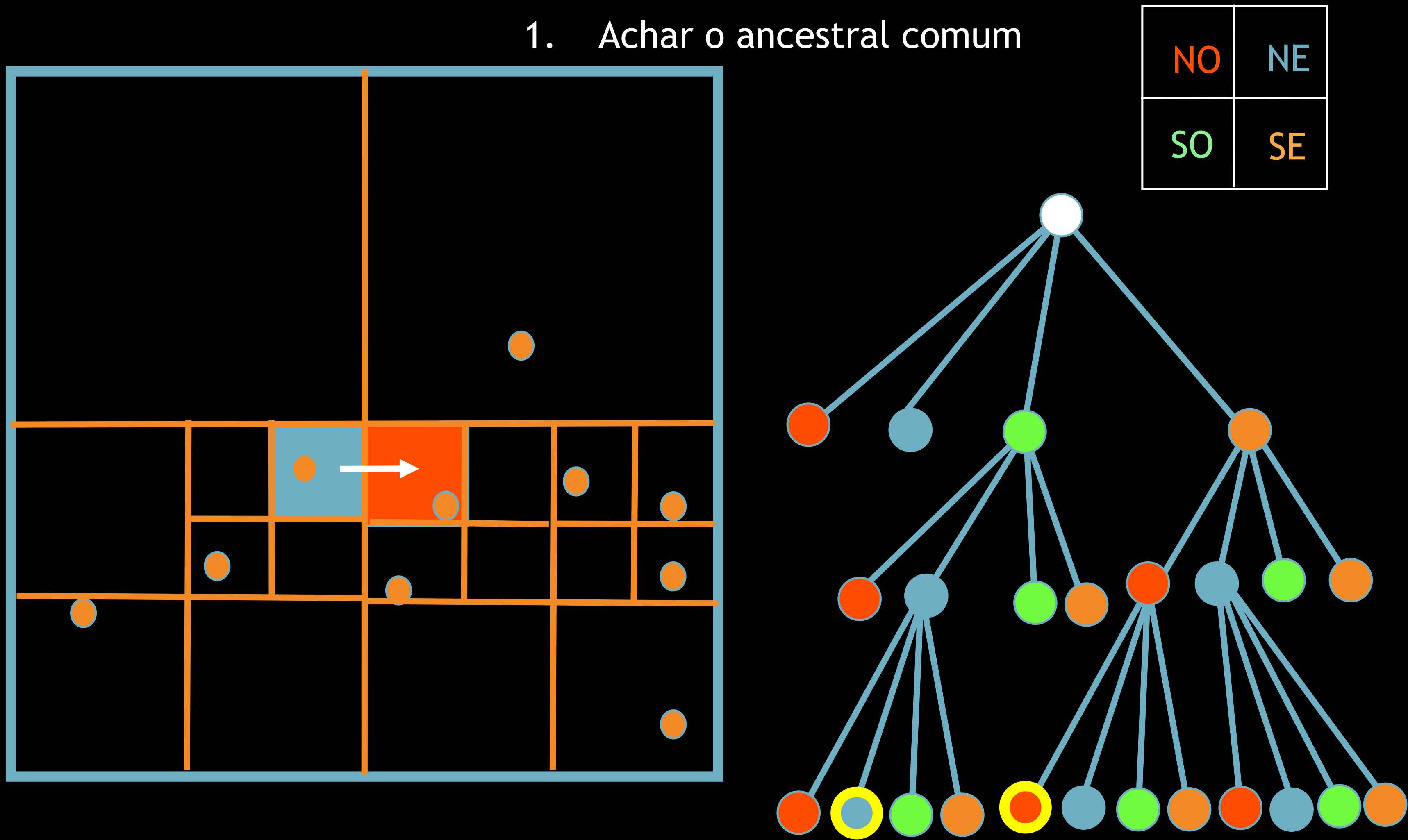
Busca de Vizinhos (Por Aresta)

NO	NE
SO	SE



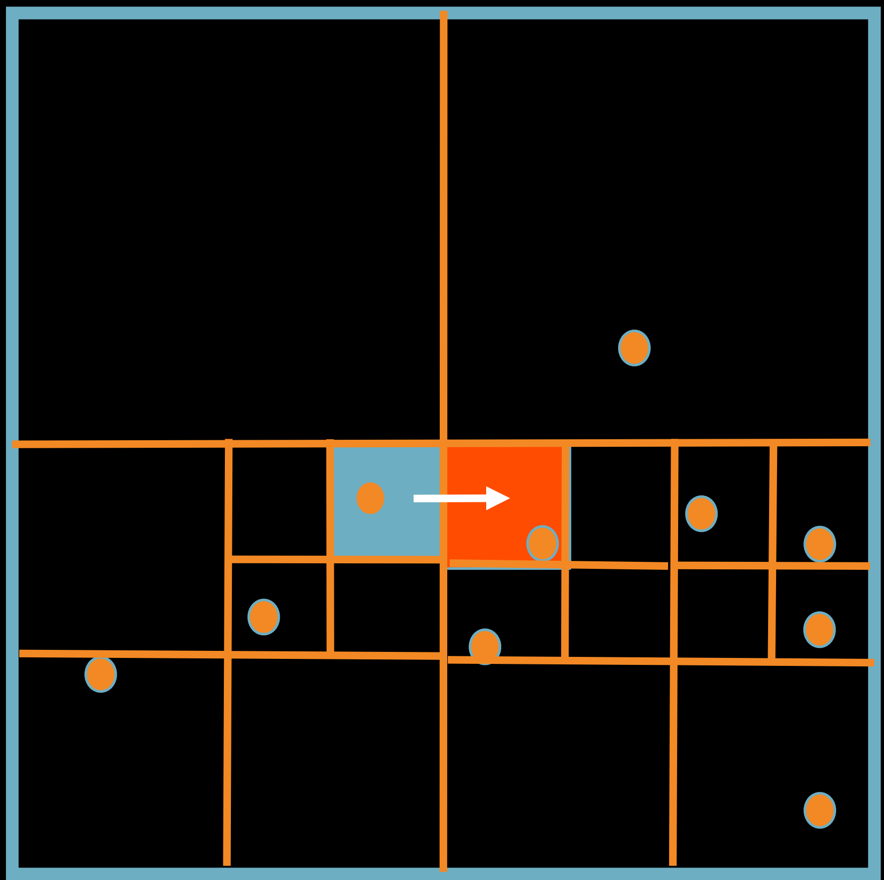
Busca de Vizinhos (Por Aresta)

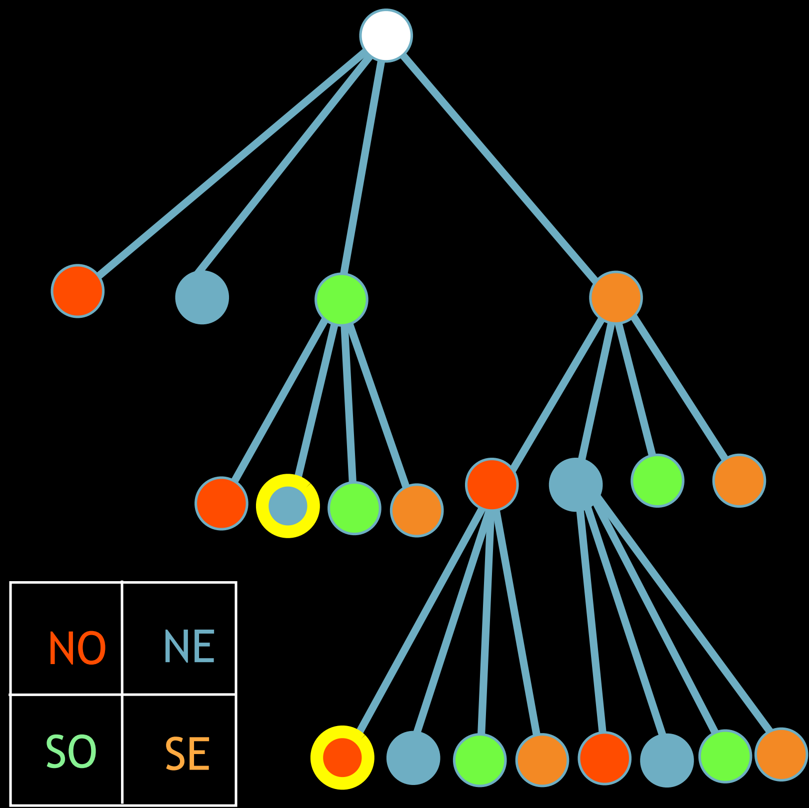
1. Achar o ancestral comum



Busca de Vizinhos

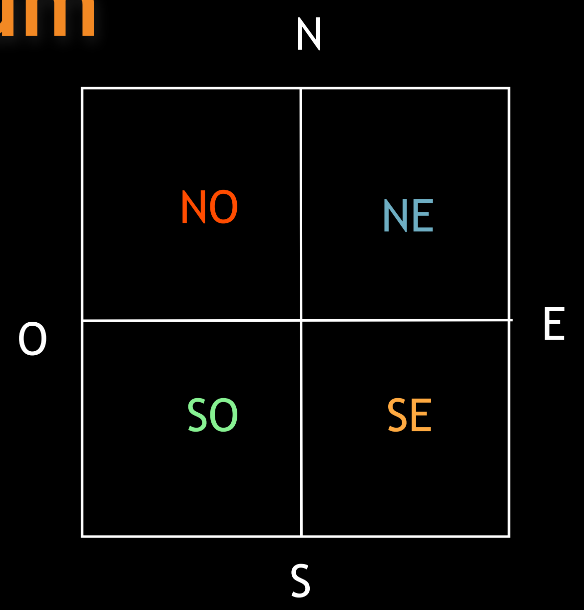
1. Achar o ancestral comum





Achar o ancestral comum

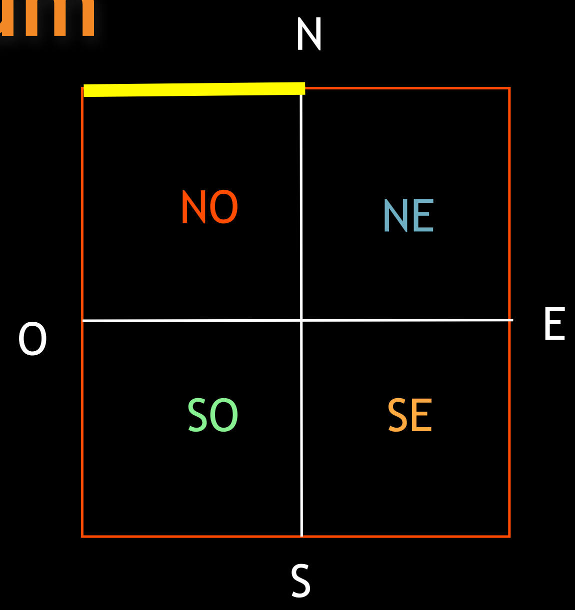
DIR(i) X QUAD(o)	NO	NE	SO	SE
N	T	T	F	F
E	F	T	F	T
S	F	F	T	T
O	T	F	T	F
NO	T	F	F	F
NE	F	T	F	F
SO	F	F	T	F
SE	F	F	F	T



ADJ(I,O) = TRUE, se e somente se se o quadrante O e' adjacente a aresta ou vertice i do bloco contendo O

Achar o ancestral comum

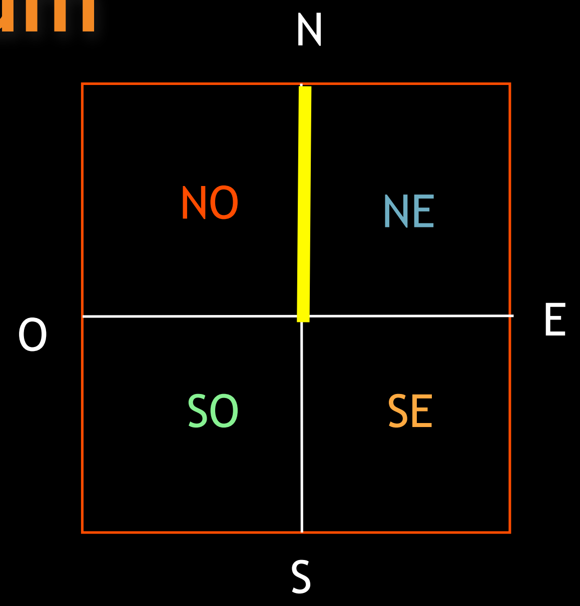
DIR(i) X QUAD(o)	NO	NE	SO	SE
N	T	T	F	F
E	F	T	F	T
S	F	F	T	T
O	T	F	T	F
NO	T	F	F	F
NE	F	T	F	F
SO	F	F	T	F
SE	F	F	F	T



ADJ(I,O) = TRUE, se e somente se se o quadrante O e' adjacente a aresta ou vertice i do bloco contendo O

Achar o ancestral comum

DIR(i) X QUAD(o)	NO	NE	SO	SE
N	T	T	F	F
E	F	T	F	T
S	F	F	T	T
O	T	F	T	F
NO	T	F	F	F
NE	F	T	F	F
SO	F	F	T	F
SE	F	F	F	T



ADJ(I,O) = TRUE, se e somente se se o quadrante O e' adjacente a aresta ou vertice i do bloco contendo O

Achar o ancestral comum

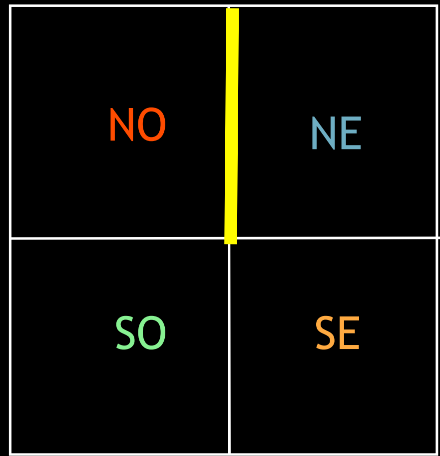
DIR(i) X QUAD(o)	NO	NE	SO	SE
N	SO	SE	NO	NE
E	NE	NO	SE	SO
S	SO	SE	NO	NE
O	NE	NO	SE	SO
NO	SE	SO	NE	NO
NE	SE	SO	NE	NO
SO	SE	SO	NE	NO
SE	SE	SO	NE	NO

NO	NE
SO	SE

REFLECT(I,O) =tipo do bloco de tamanho igual (nao necessariamente irmao) compartilhando a aresta

Achar o ancestral comum

DIR(i) X QUAD(o)	NO	NE	SO	SE
N	SO	SE	NO	NE
E	NE	NO	SE	SO
S	SO	SE	NO	NE
O	NE	NO	SE	SO
NO	SE	SO	NE	NO
NE	SE	SO	NE	NO
SO	SE	SO	NE	NO
SE	SE	SO	NE	NO



REFLECT(I,O) =tipo do bloco de tamanho igual (nao necessariamente irmao) compartilhando a aresta

Achar o ancestral comum

DIR(i) X QUAD(o)	NO	NE	SO	SE
NO	-	N	O	-
NE	N	-	-	E
SO	O	-	-	S
SE	-	E	S	-

NO	NE
SO	SE

ARESTA_COMUM(I,O) =retorna o tipo da aresta comum ao bloco contendo o quadrante O e seu vizinho na direcao de I

Busca de Vizinho por aresta

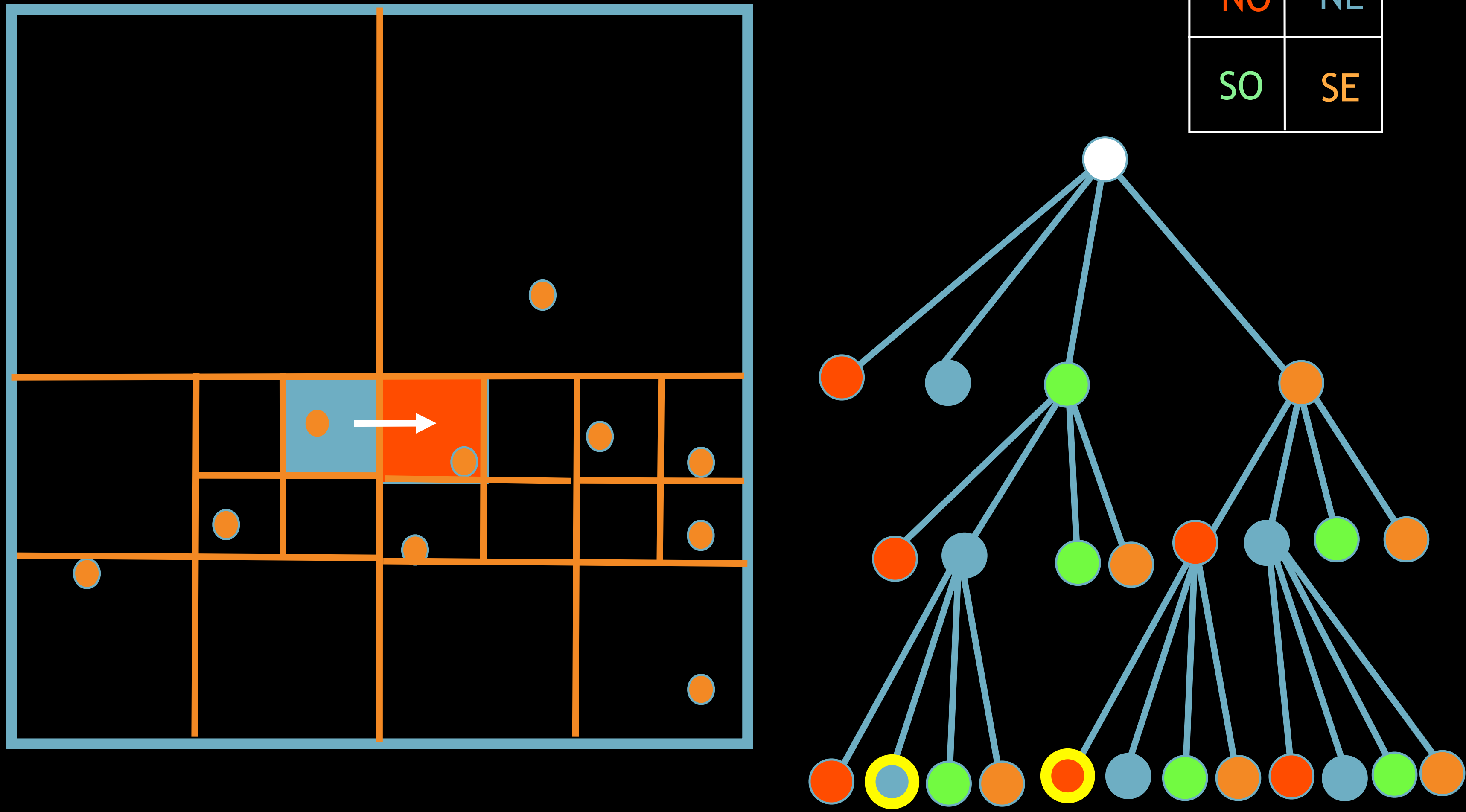
1. Localizar o mais proximo ancestral comum. Guardar o caminho
2. Refletir o caminho de subida para descer na arvore

```
PROCEDURE VizinhoPorAresta(P,I)
1.   RETURN (SON(IF  ADJ(I,TIPO(P))
                      THEN VizinhoPorAresta(PAI(P),I)
                      ELSE PAI(P),
                      REFLECT(I,TIPO(P))));
```

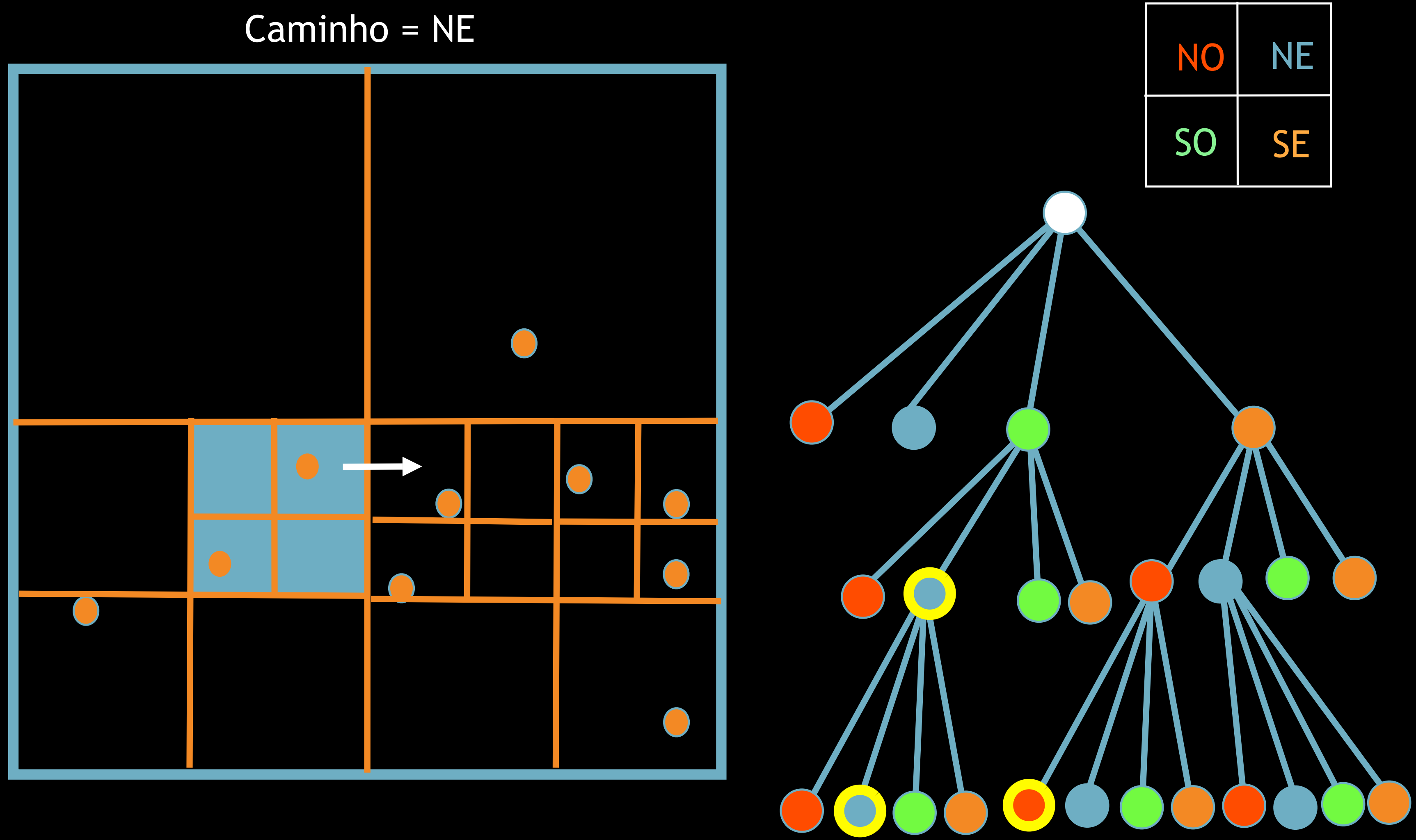
Busca de Vizinhos (Por Aresta)

1. Achar o ancestral comum

NO	NE
SO	SE



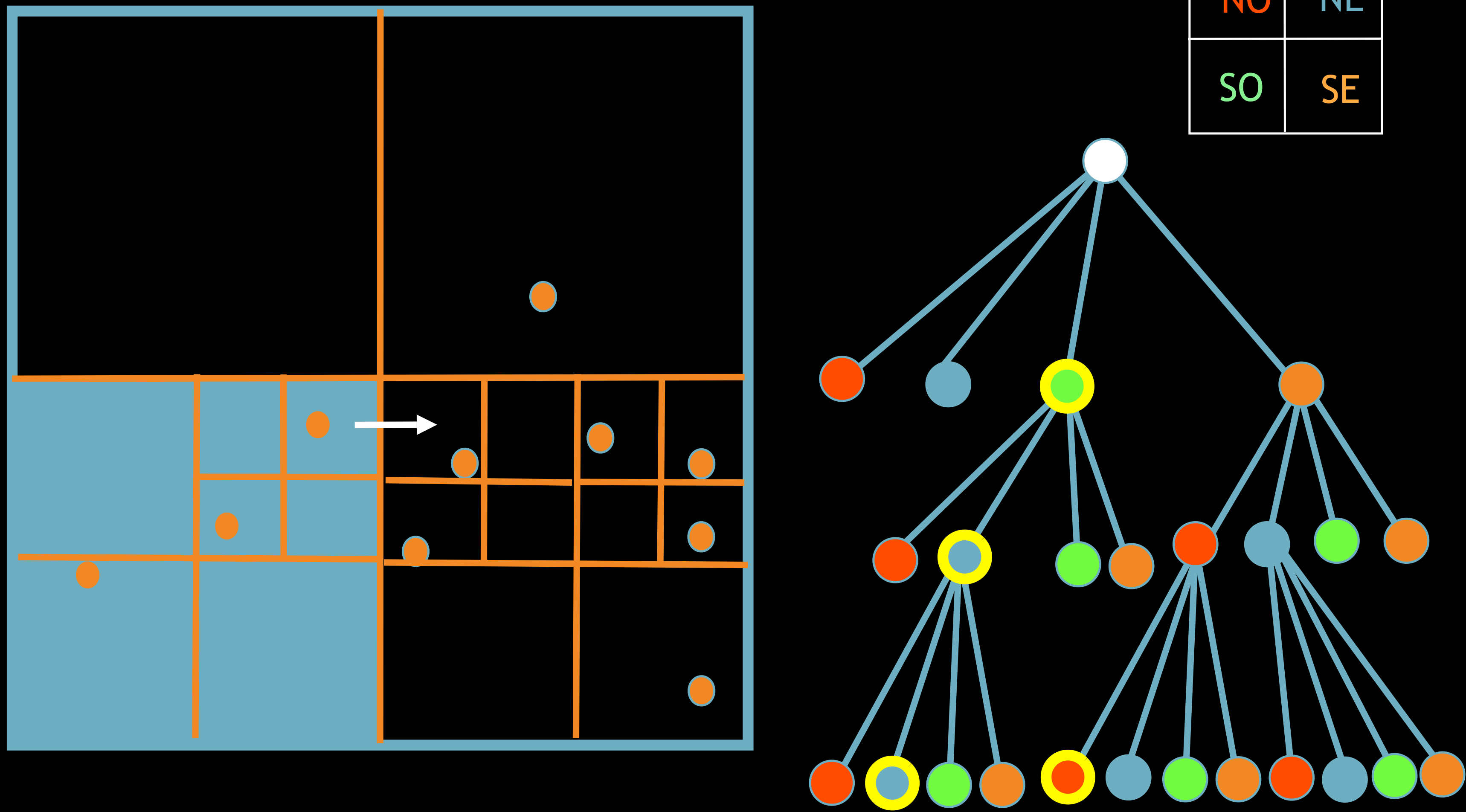
Busca de Vizinhos (Por Aresta)



Busca de Vizinhos (Por Aresta)

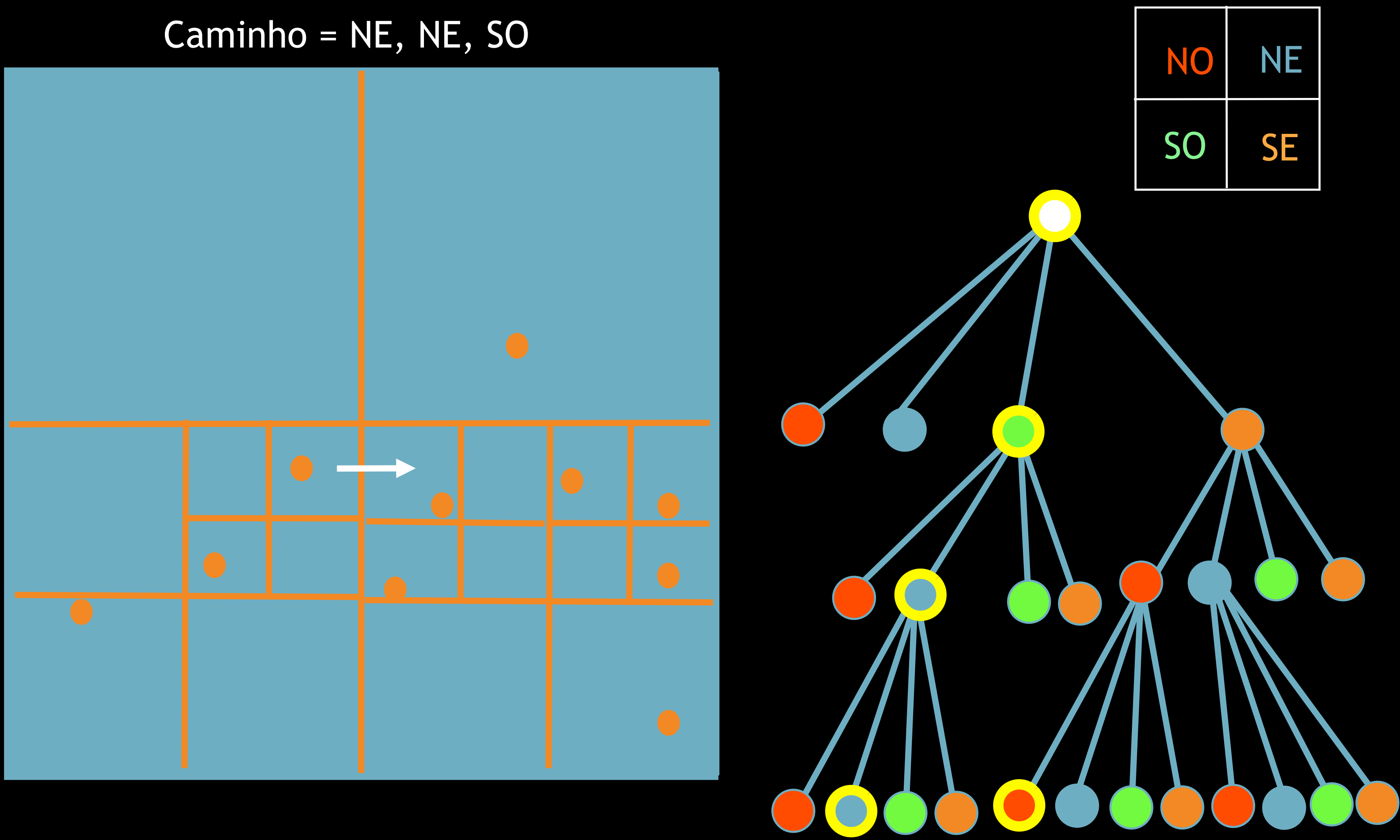
Caminho = NE, NE

NO	NE
SO	SE



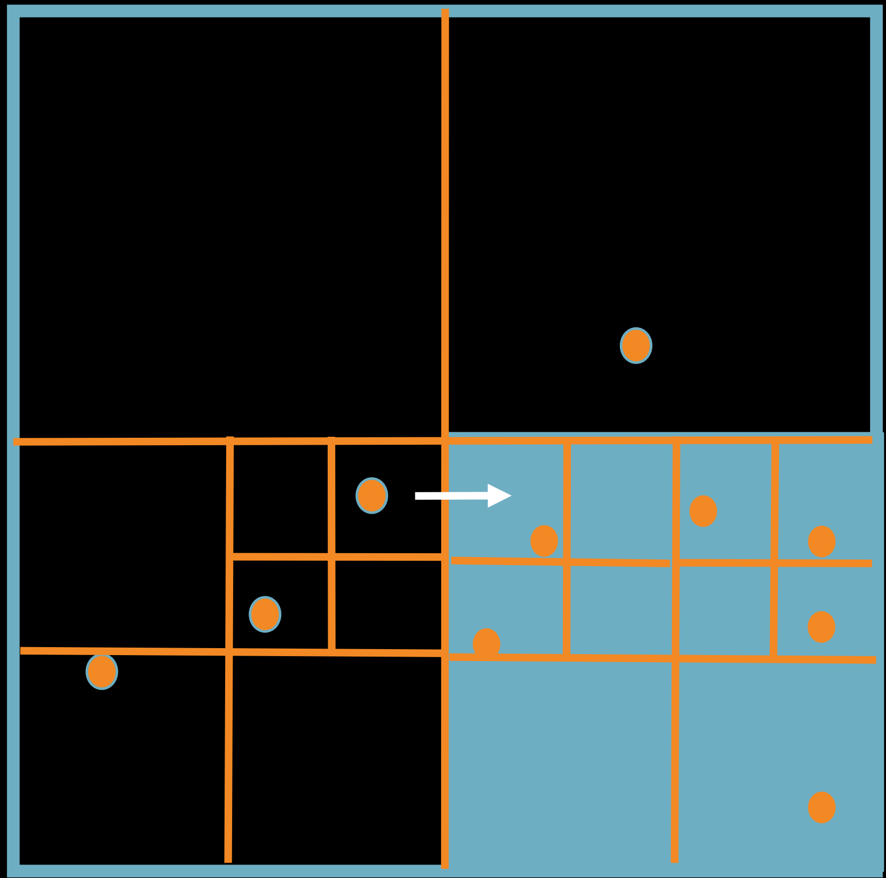
Busca de Vizinhos (Por Aresta)

Caminho = NE, NE, SO



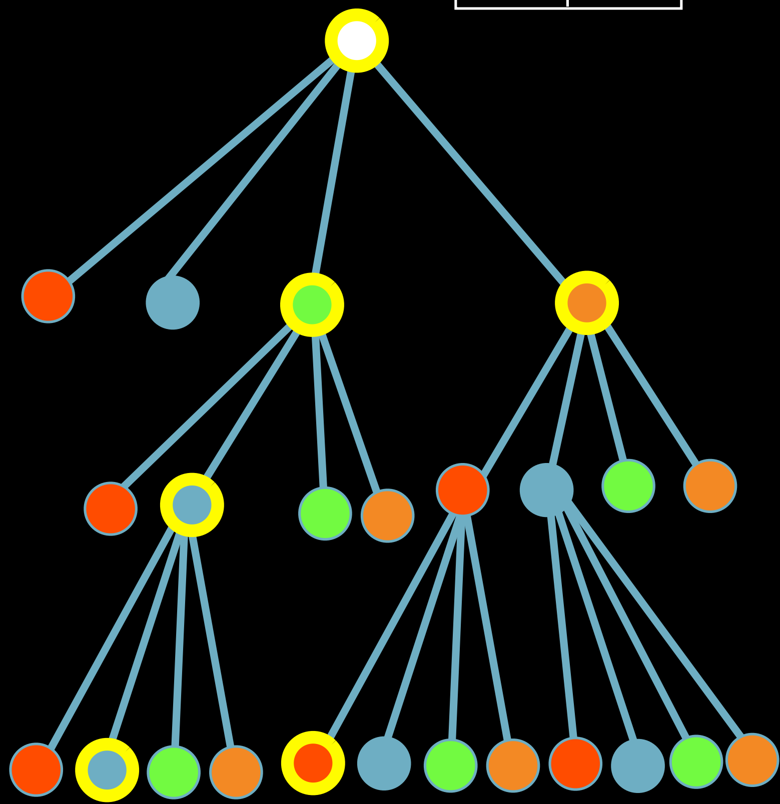
Busca de Vizinhos (Por Aresta)

Caminho = NE, NE, SO



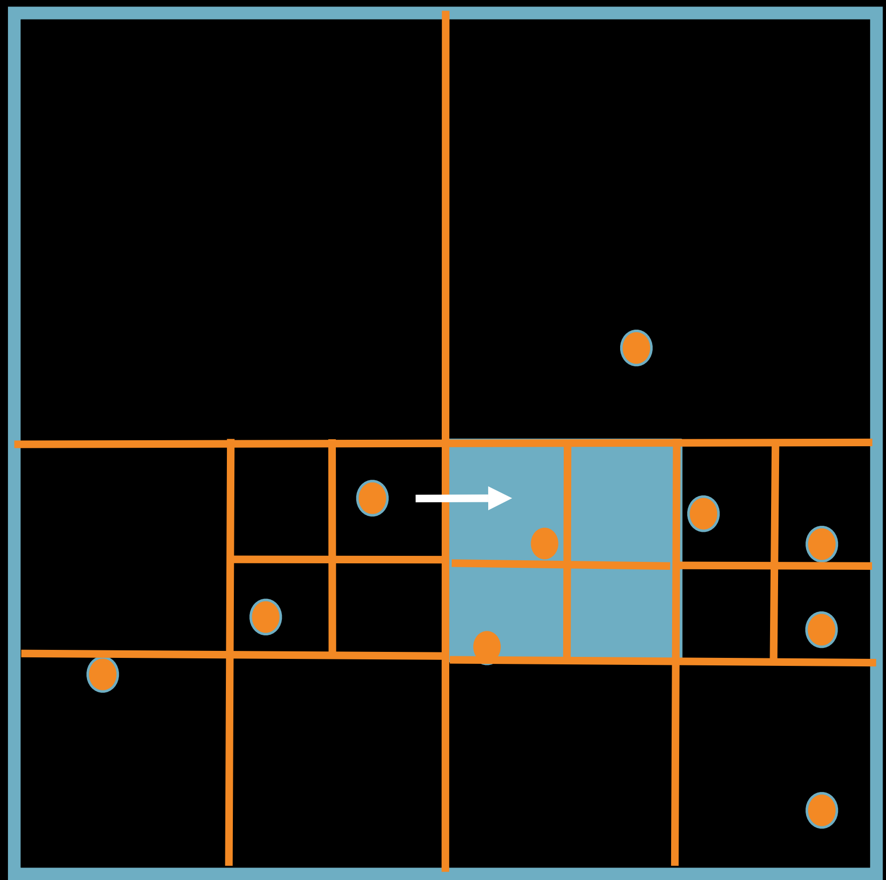
Caminho Reverso = SE

NO	NE
SO	SE



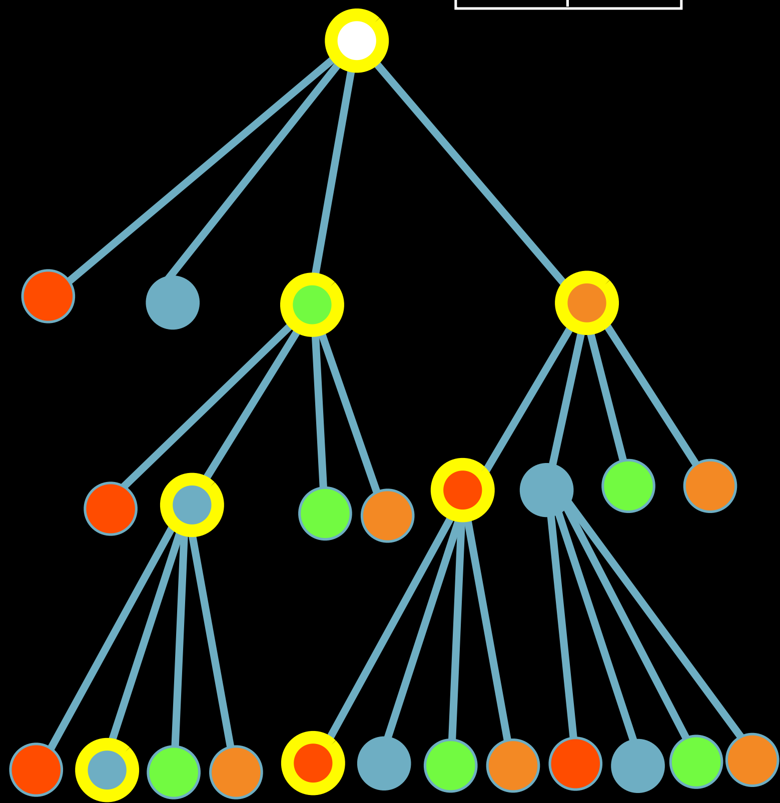
Busca de Vizinhos (Por Aresta)

Caminho = NE, NE, SO



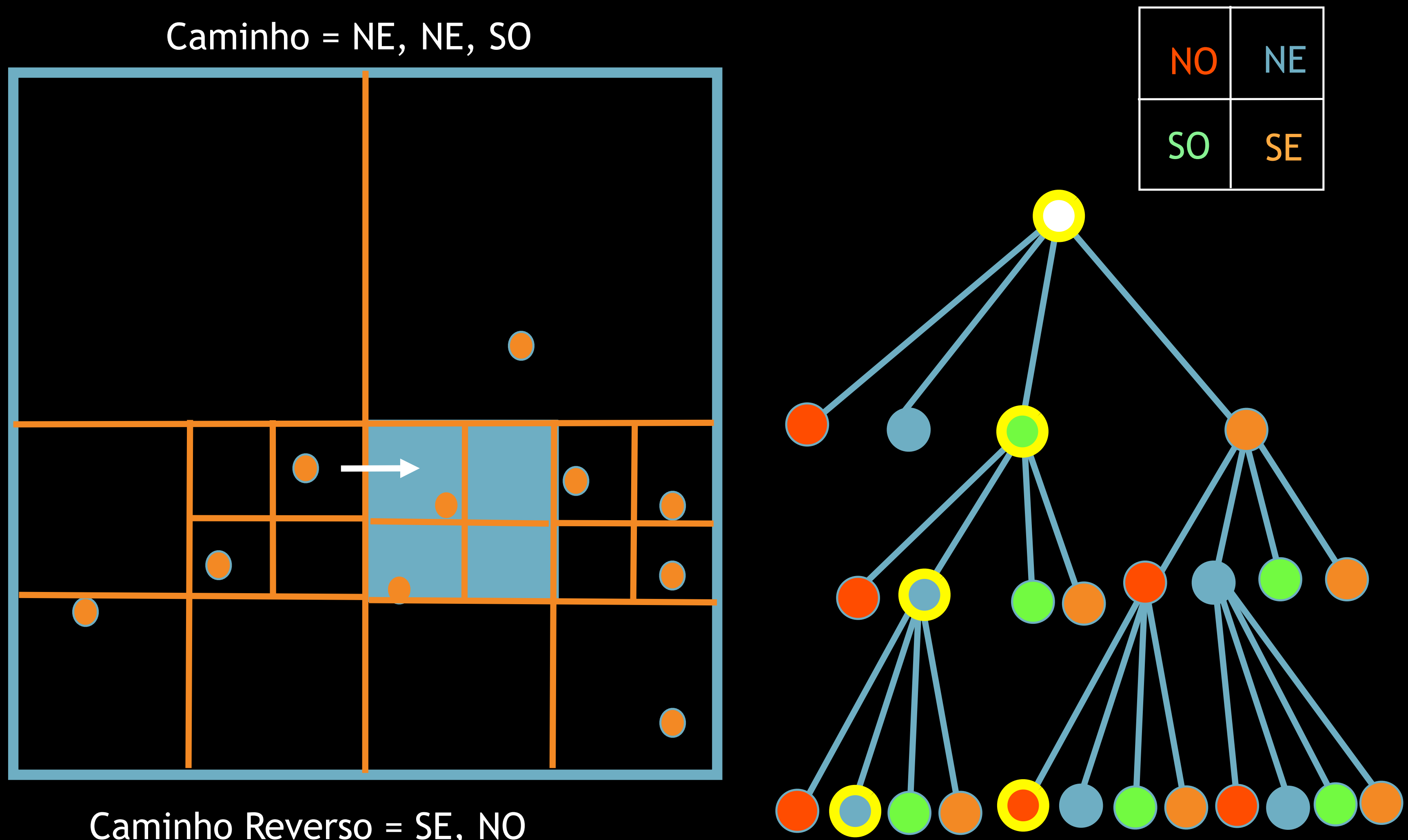
Caminho Reverso = SE, NO

NO	NE
SO	SE



Busca de Vizinhos (Por Aresta)

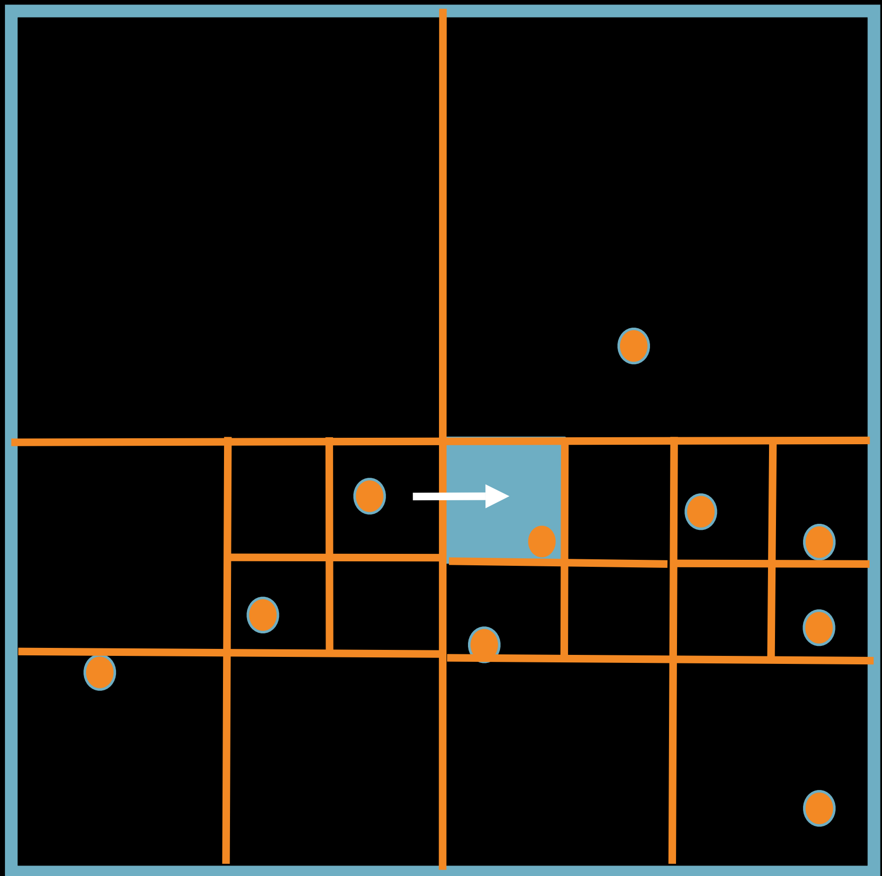
Caminho = NE, NE, SO

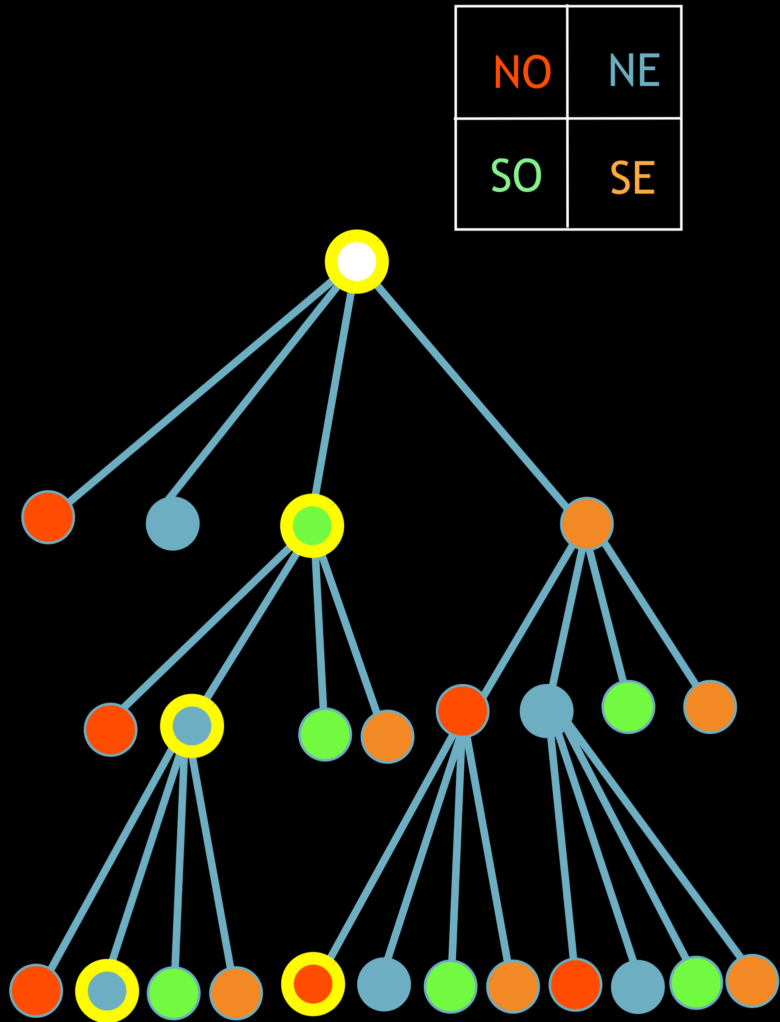


Caminho Reverso = SE, NO

Busca de Vizinhos (Por Aresta)

Caminho = NE, NE, SO



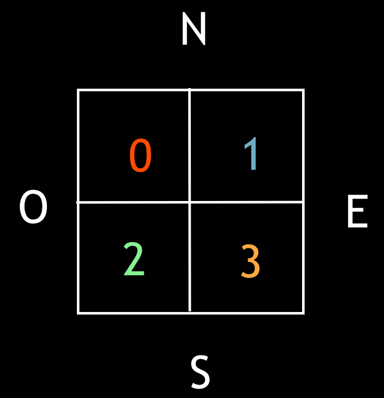


Caminho Reverso = SE, NO, NO

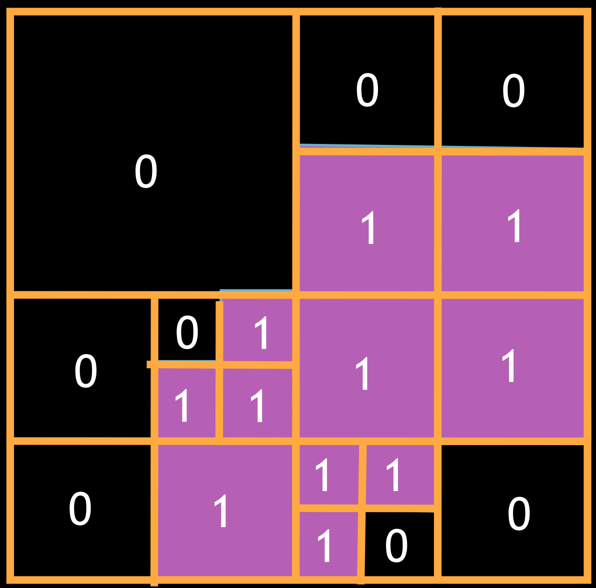
Busca de Vizinho por vértice

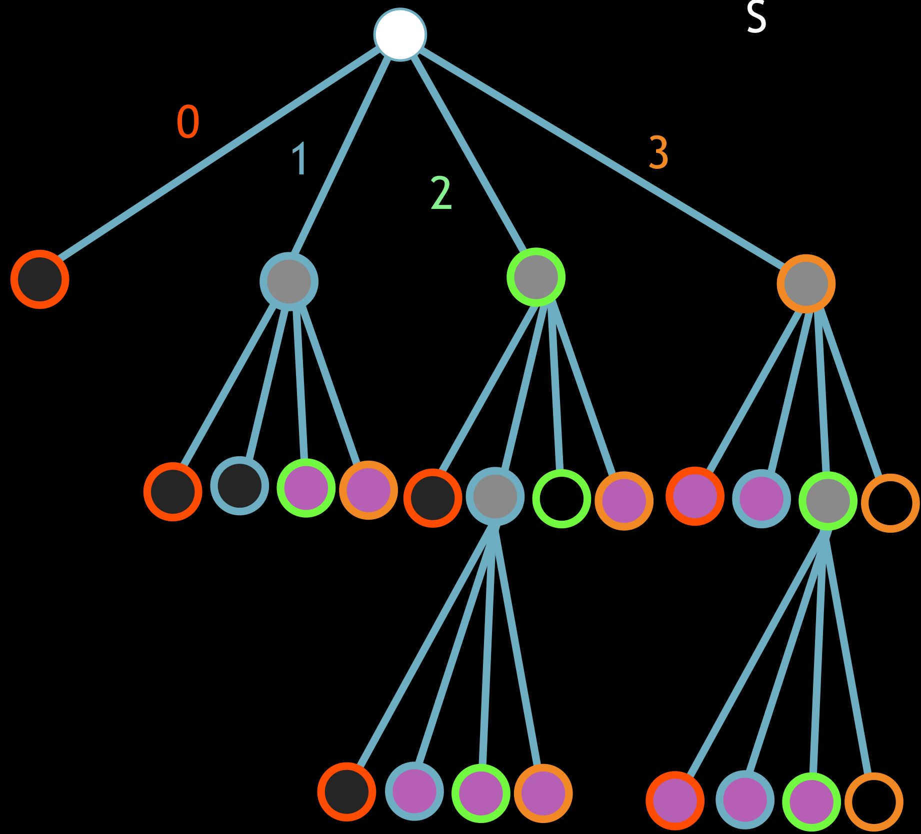
```
PROCEDURE VizinhoPorVertice(P,I)
1.    RETURN (SON(IF  ADJ(I,TIPO(P))
                   THEN VizinhoPorVertice(PAI(P),I)
                    ELSIF ARESTA_COMUM(I,TIPO(P)) != NULL
                        THEN VizinhoPorAresta(
                               PAI(P), ARESTA_COMUM(I,TIPO(P))
                        ELSE PAI(P),
                   REFLECT(I,TIPO(P))));
```

Representacao Linearizada

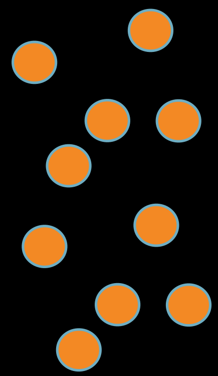


Q = {12, 13, 211, 212, 213, 23, 30, 31, 320, 321, 322}

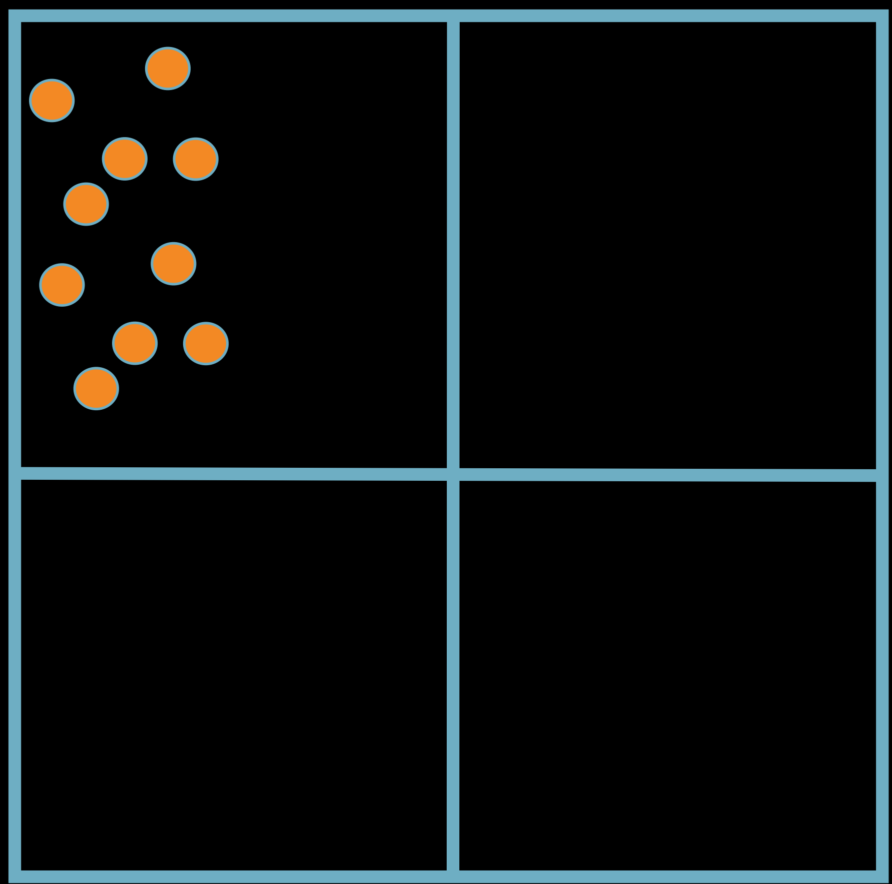




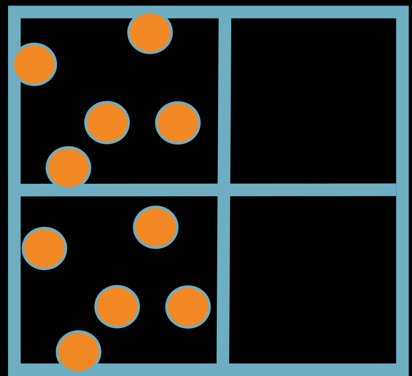
Achar o dominio inicial



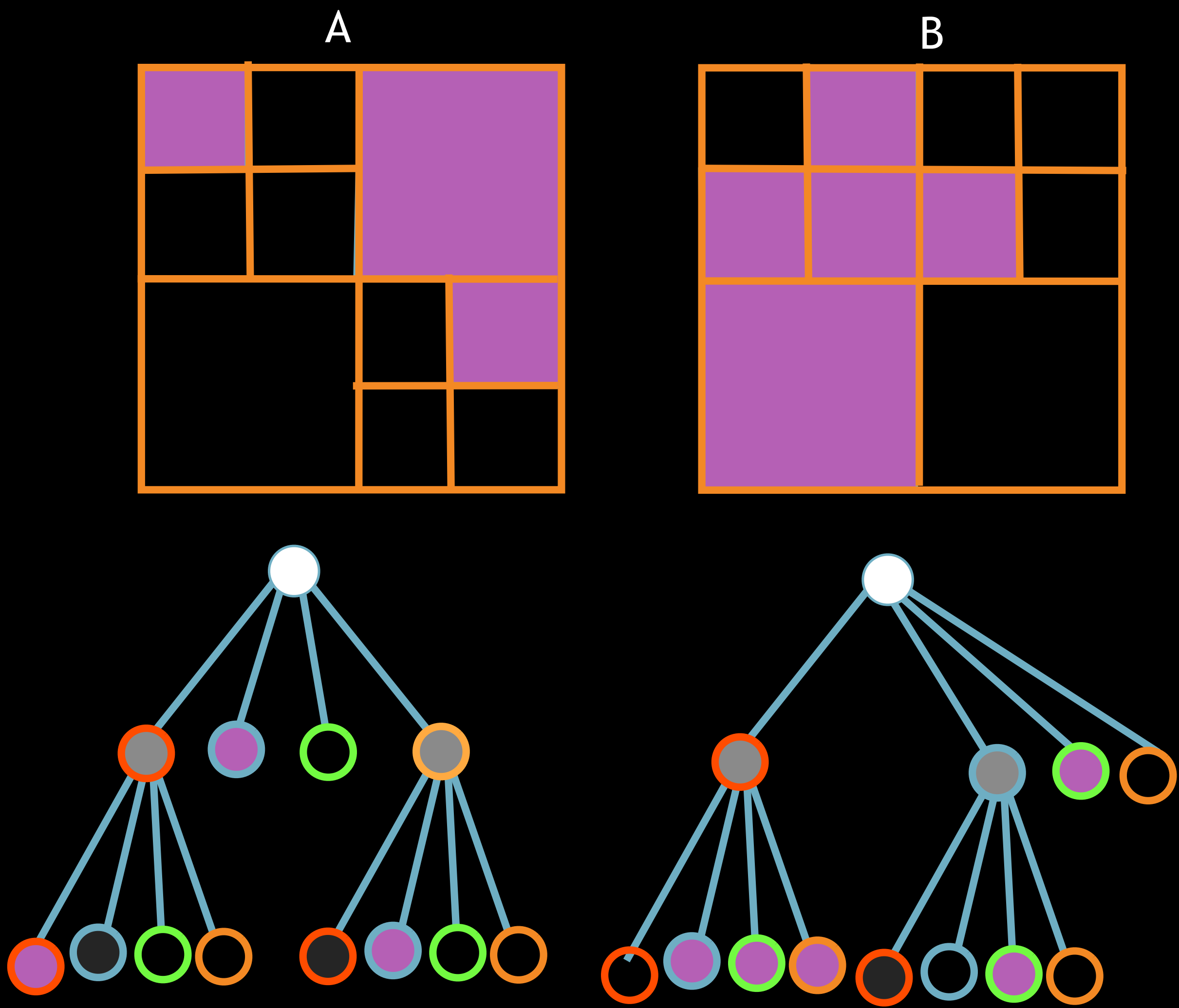
Achar o dominio inicial



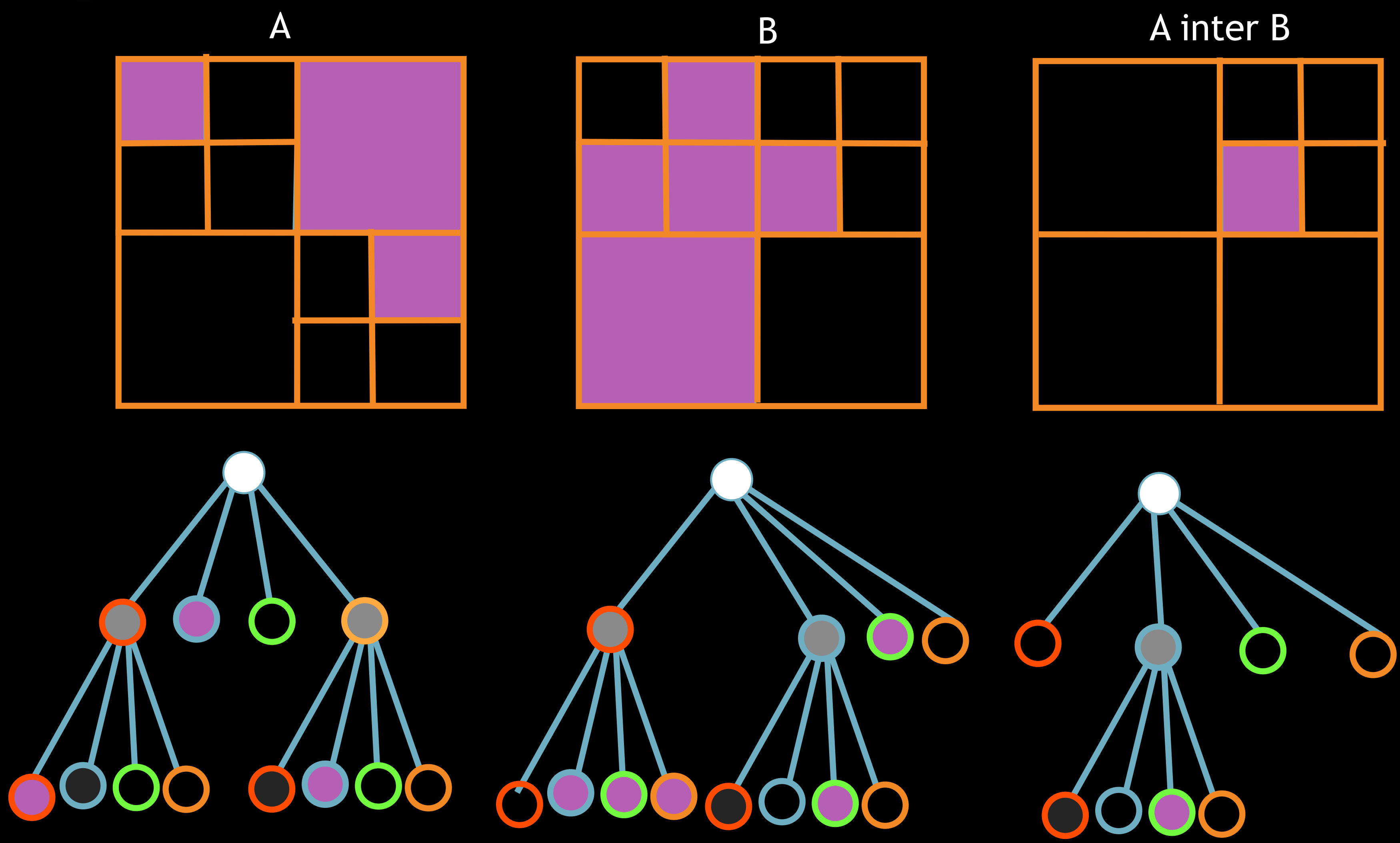
Achar o dominio inicial



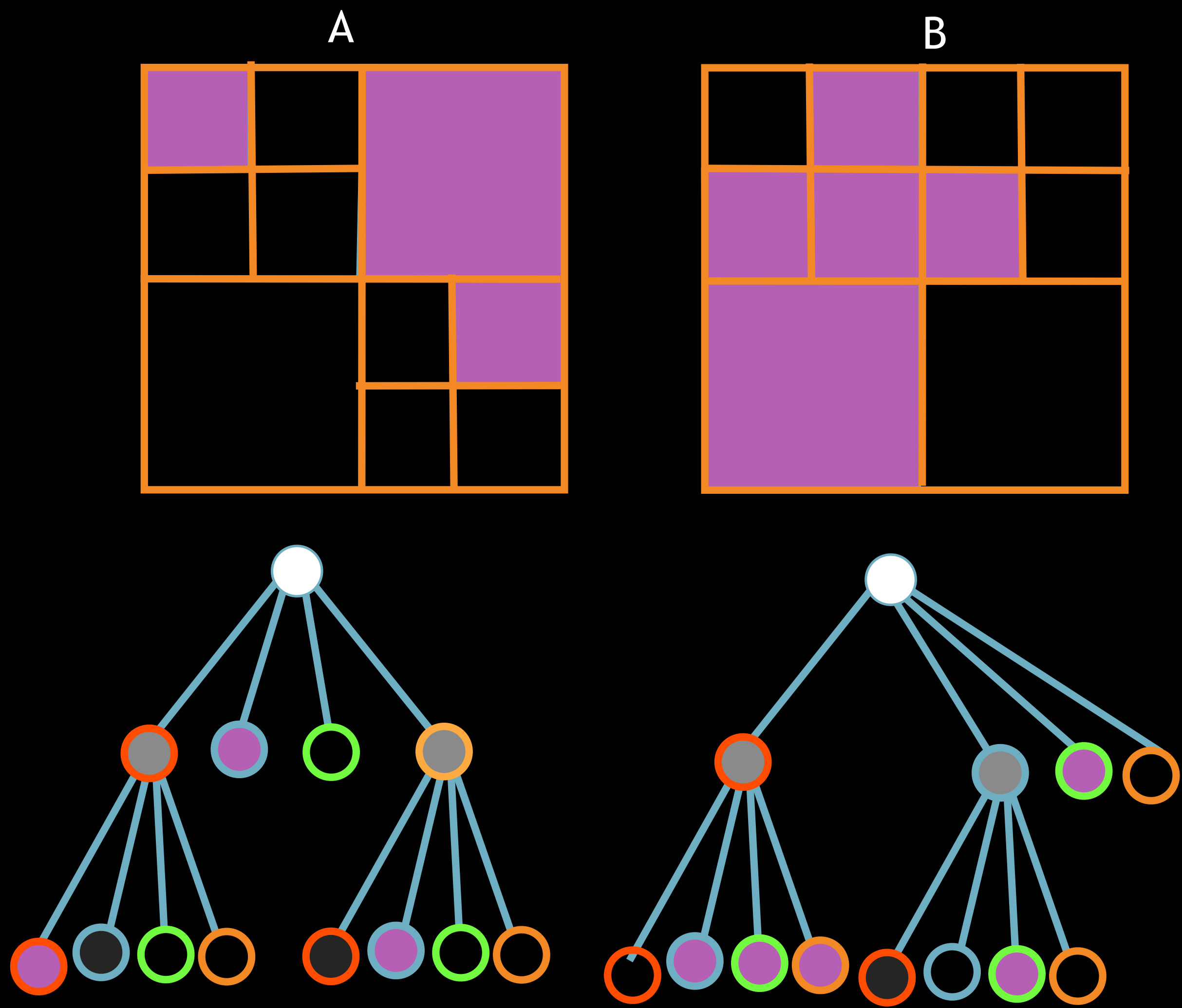
Operacoes Booleanas



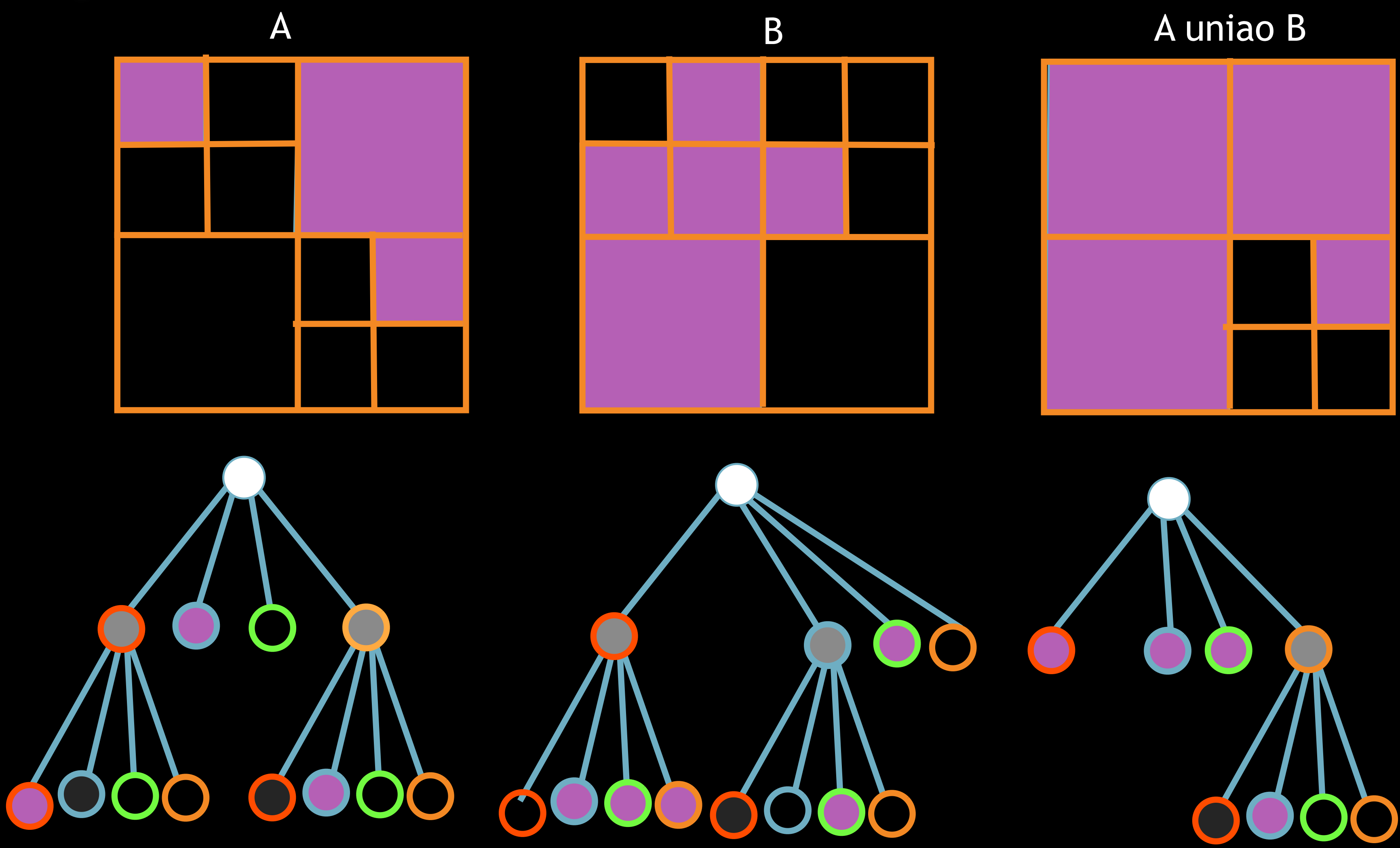
Operacoes Booleanas



Operacoes Booleanas



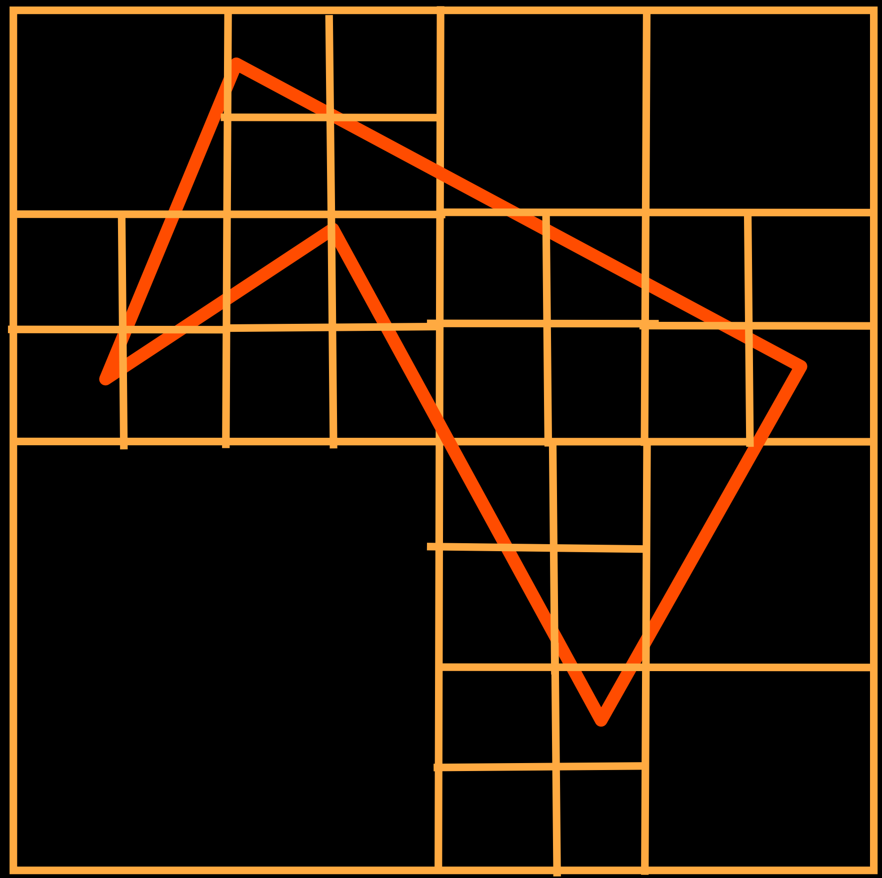
Operacoes Booleanas



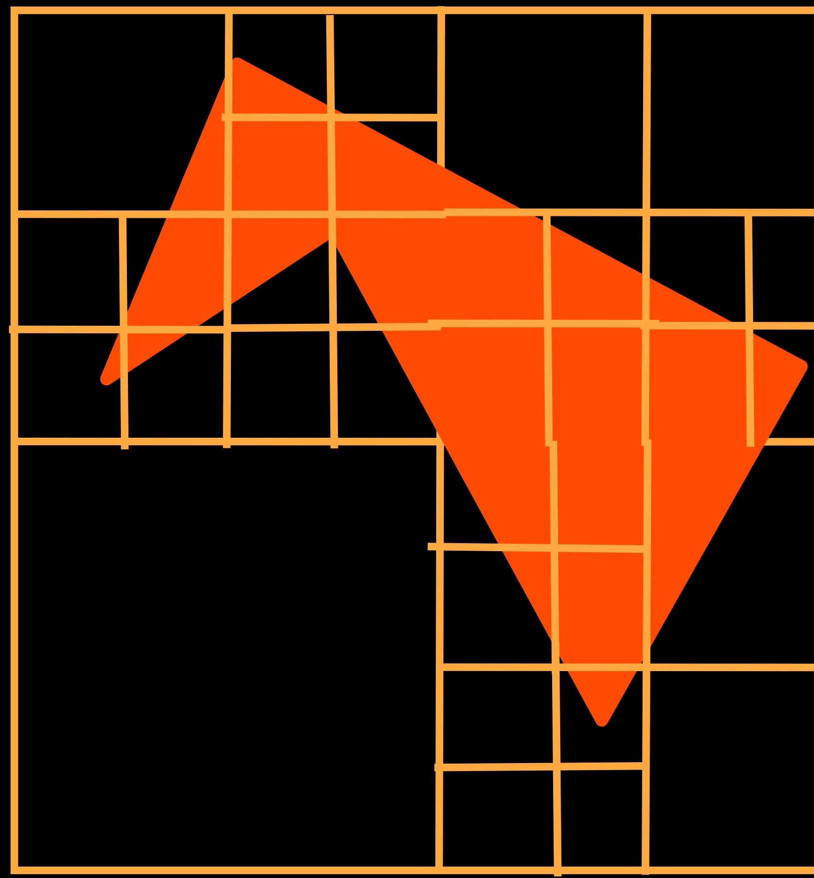
Operacoes Booleanas

```
ALGORITMO Intersecao(s,t)
IF (VAZIO(S) ou VAZIO(T))
THEN RETURN NODO(NULL,NULL,VAZIO)
ELSE IF (CHEIO(S)) THEN  RETURN COPY_TREE(t)
ELSE IF (CHEIO(T)) THEN RETURN COPY_TREE(s)
ELSE
   FOR i=0 TO 3
      p[i] = Intersecao(son(s,i),son(t,i));
   IF (VAZIO(P[0]) e VAZIO(P[1]) e VAZIO(P[2]) e VAZIO(P[2]))
      RETURN NODO(NULL,NULL, VAZIO)
   ELSE
      q = NODO(NULL,NULL, CINZA);
      FOR i=0 TO 3
        FILHO(q,i) = p[i];
        PAI(p[i]) = q;
      RETURN q;
```

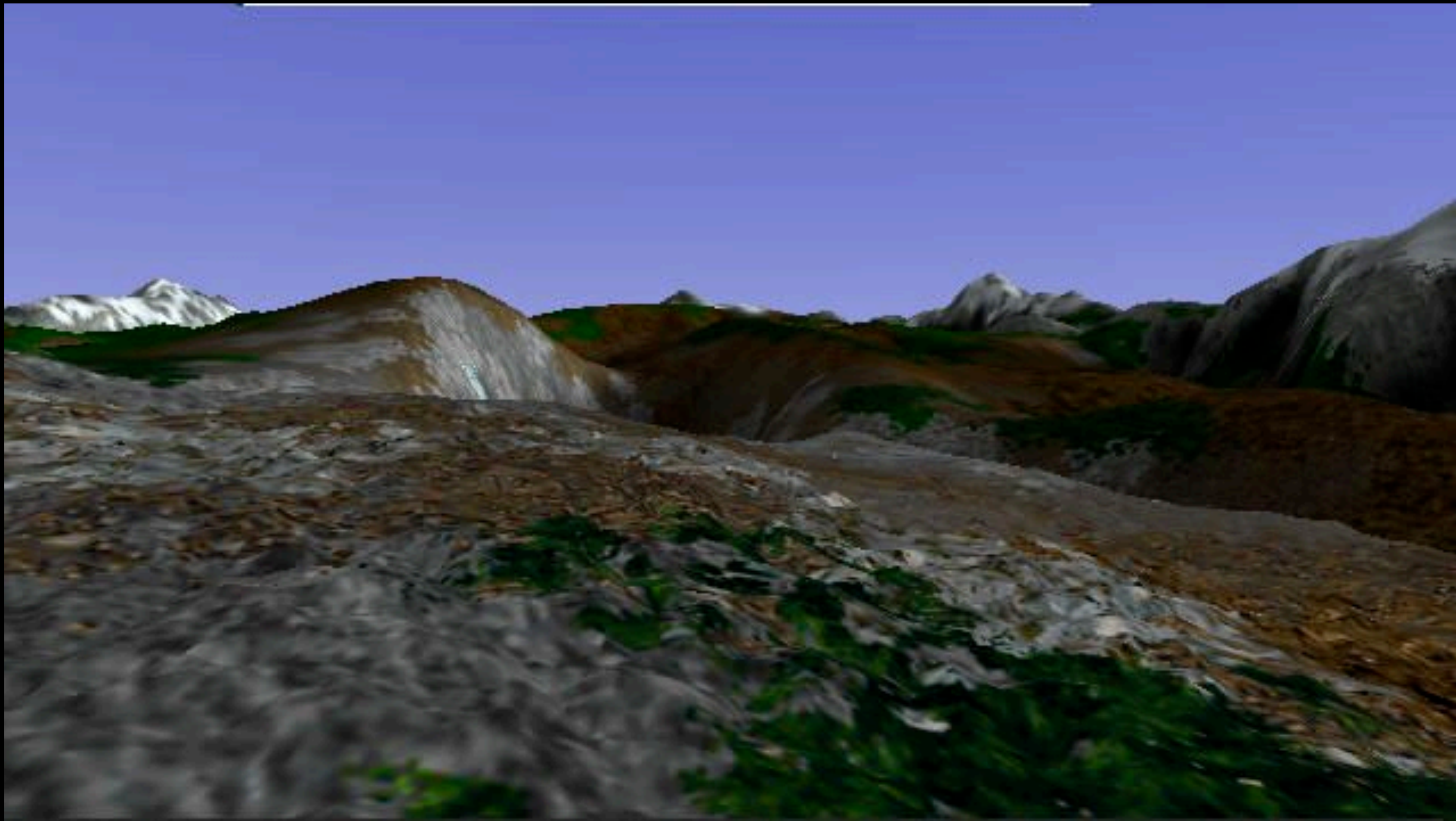
Polygonal Map Quadtrees

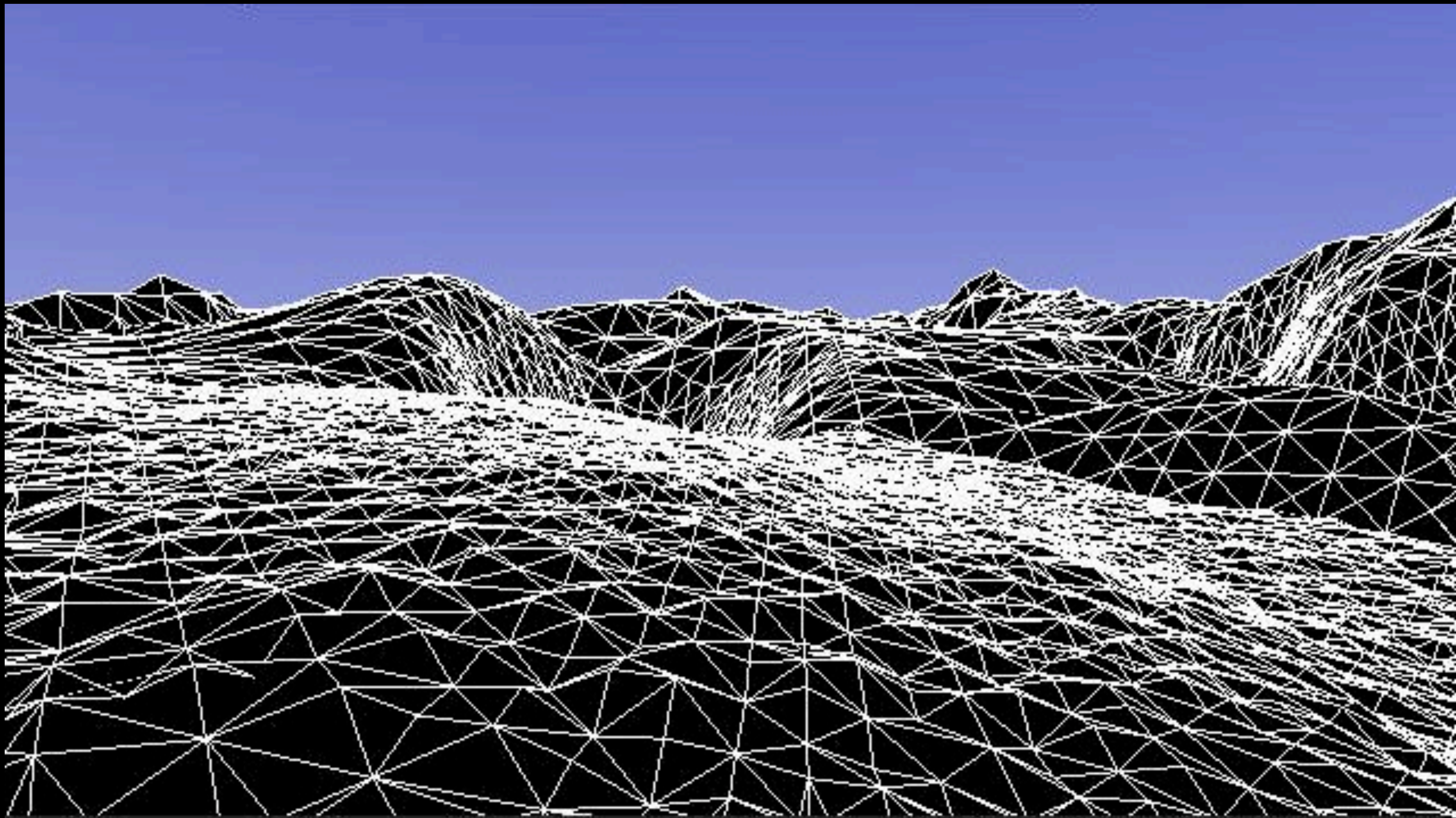


Region Quadtrees



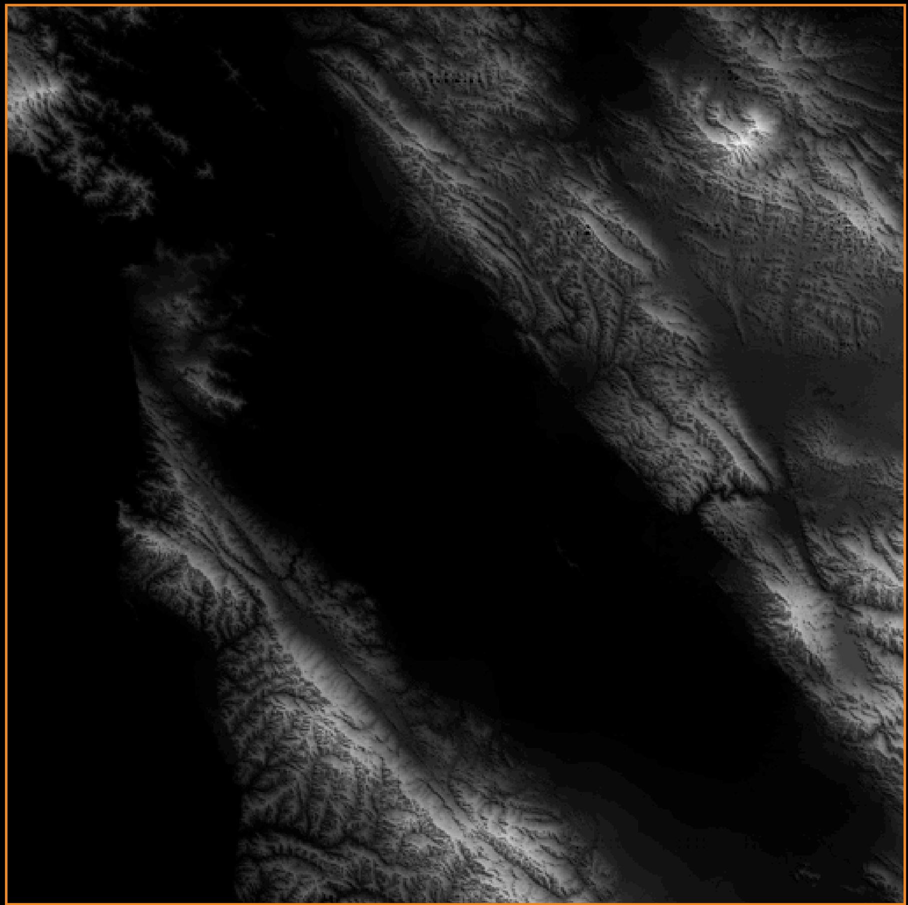
Simuladores



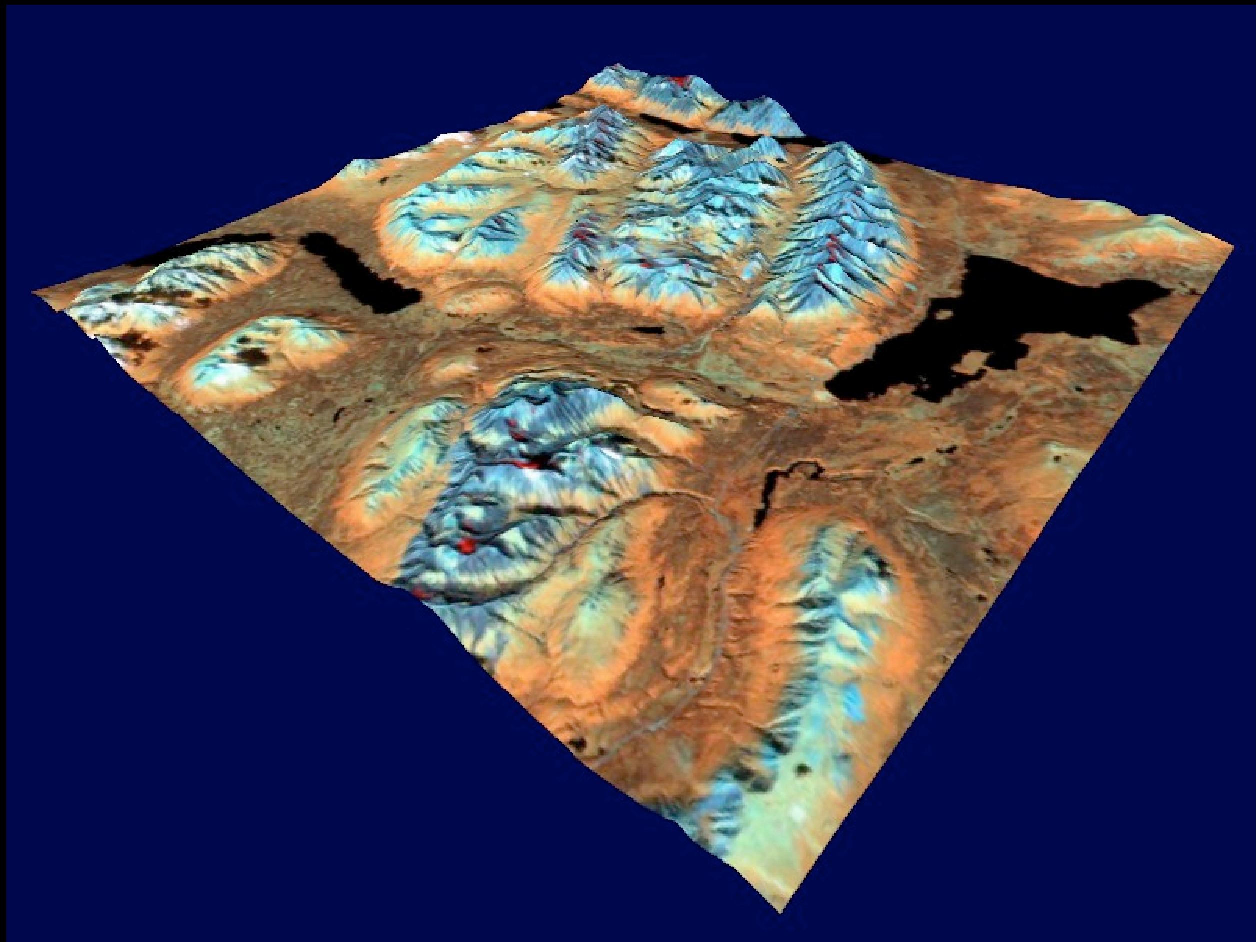


Representation: Height Fields

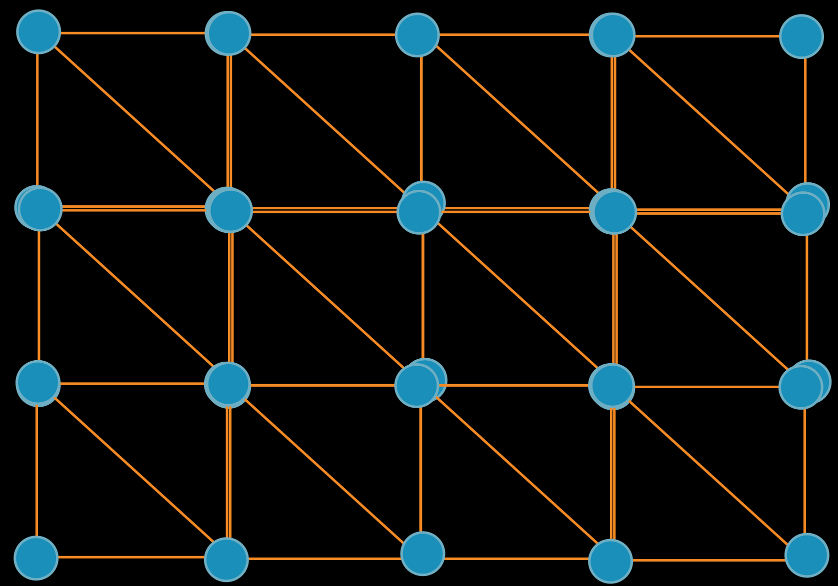
$w(v) = (x, y, z(x, y))$



Height Fields



Height fields



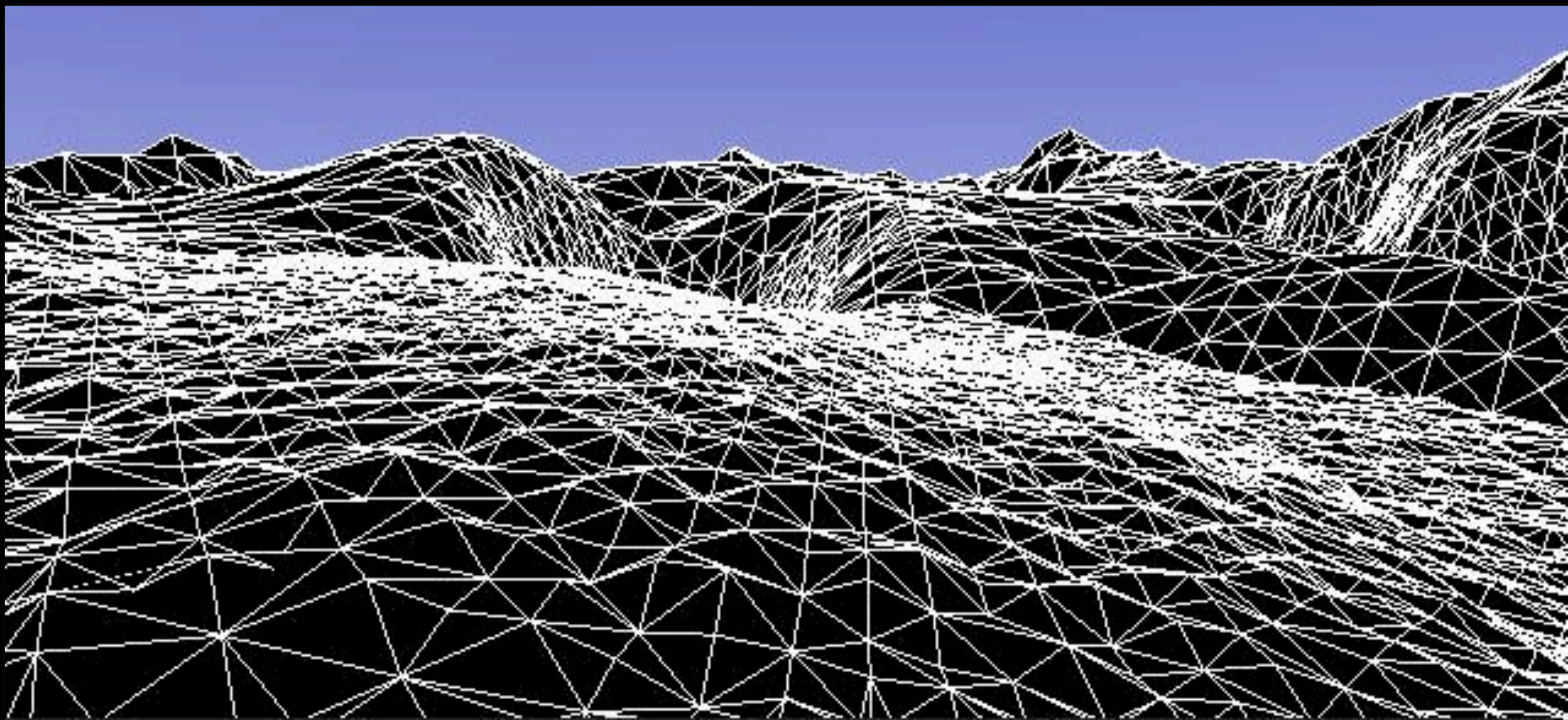
Brute force

Basic nature of LOD

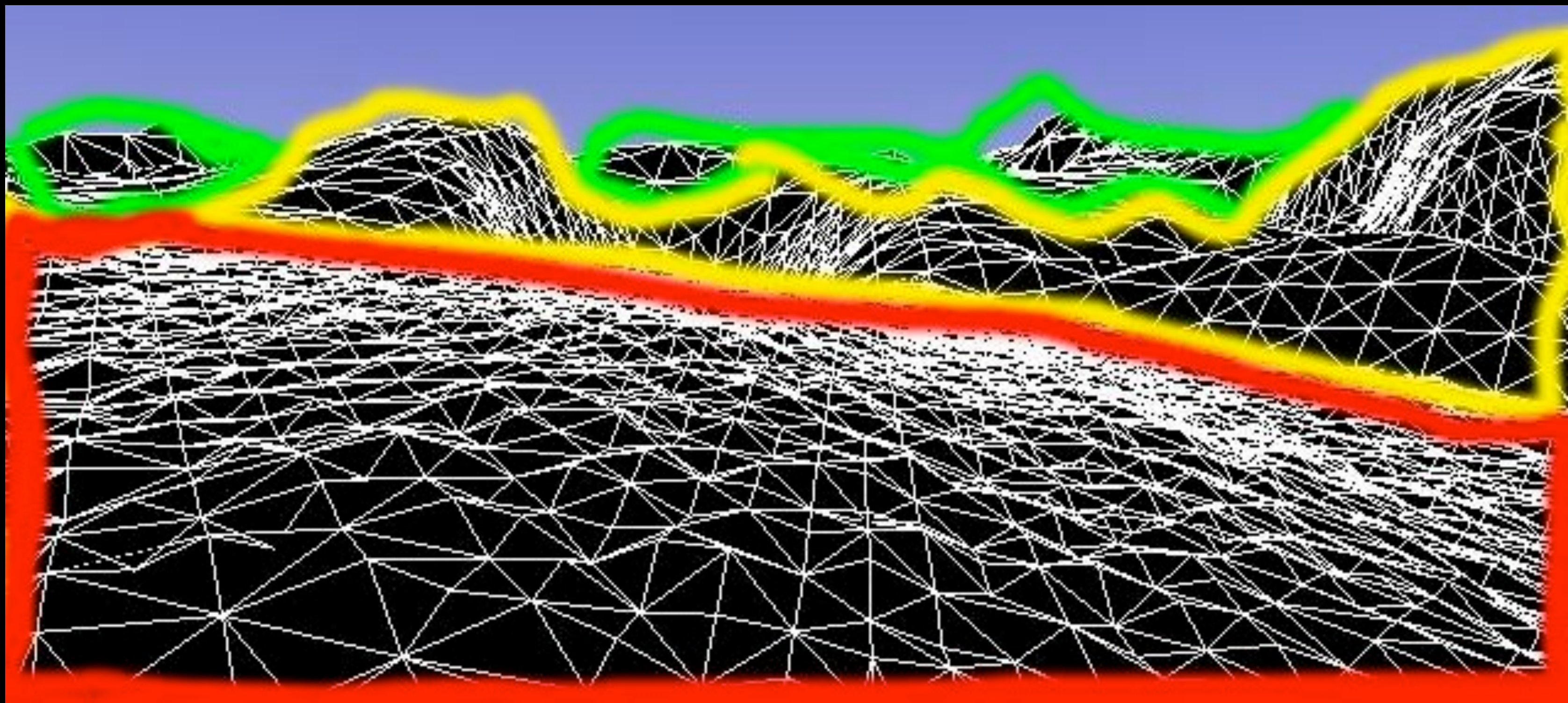
An ideally LOD'd scene: all polygons are roughly the same size in screen pixels.

By the nature of projection of 3D space, this means many more polygons are close to you than are far from you.

In an ideally LOD'd scene, all polygons are roughly the same size in screen pixels.



A large percentage of polygons are small and close (50%? 60%?)



Terrain engine design goals

Large terrain that looks good and is detailed

Can see forever (no walls of fog)

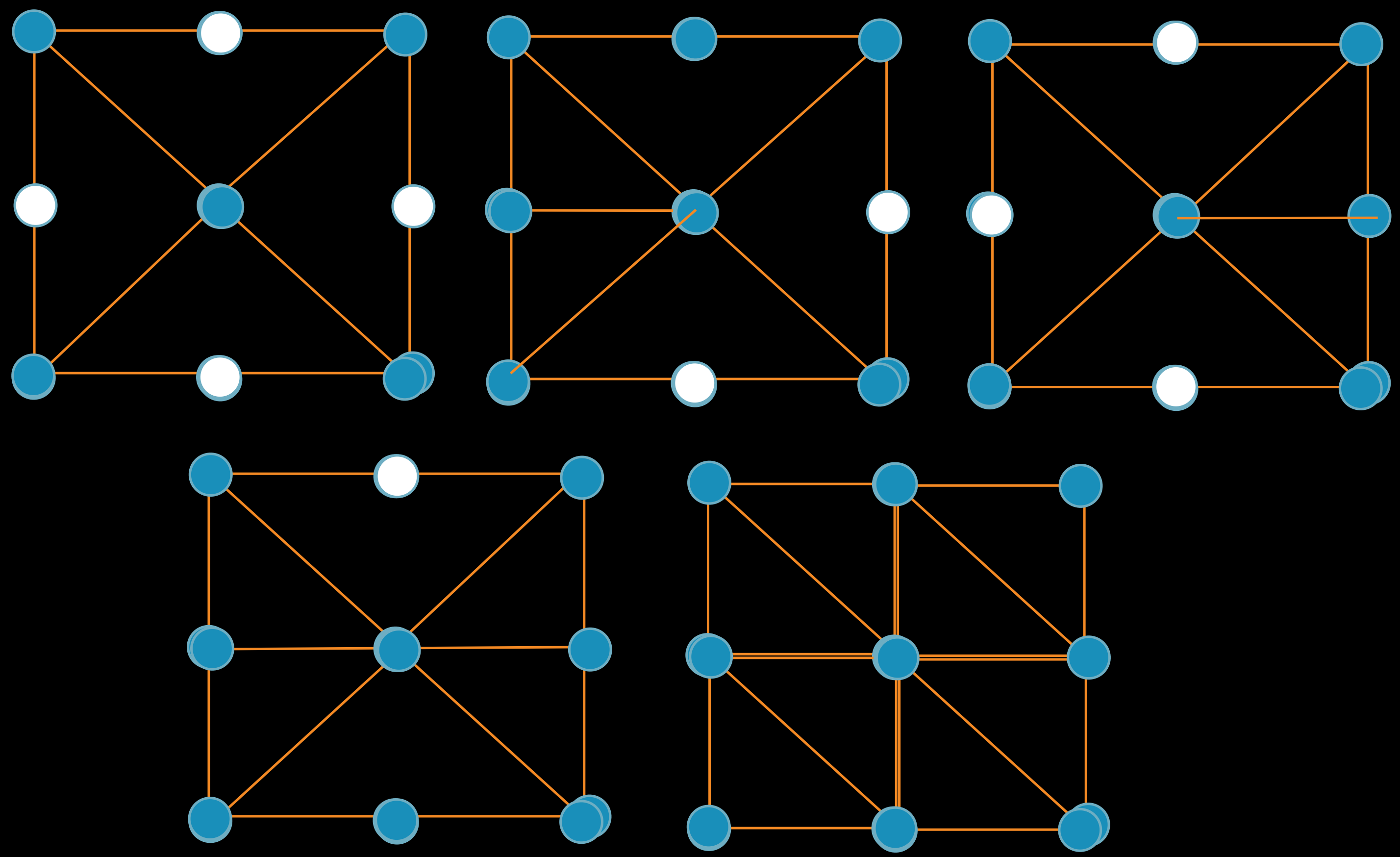
Runs at decent frame rates

Lindstrom’s algorithm

- First presented at SIGGRAPH ‘96; paper available on the Web.
- Bottom-up, works on a height field (elevation map)

Lindstrom's algorithm

Choosing which vertices to render

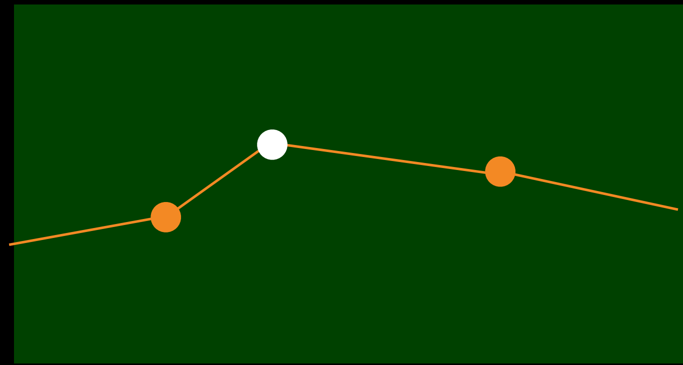


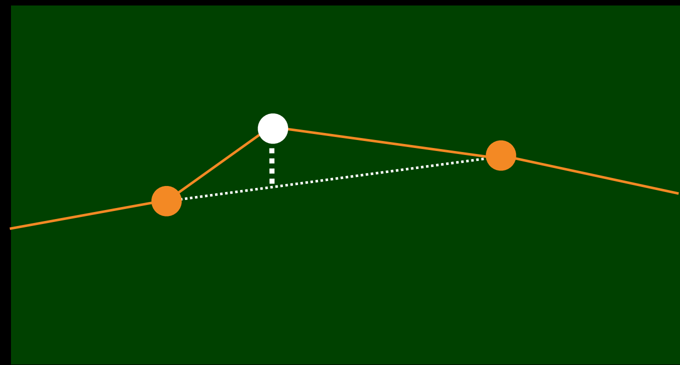
Lindstrom's algorithm

Each vertex has an associated "error height"

We estimate the size of this height projected to the screen.

If it exceeds some threshold, we "enable" the vertex.



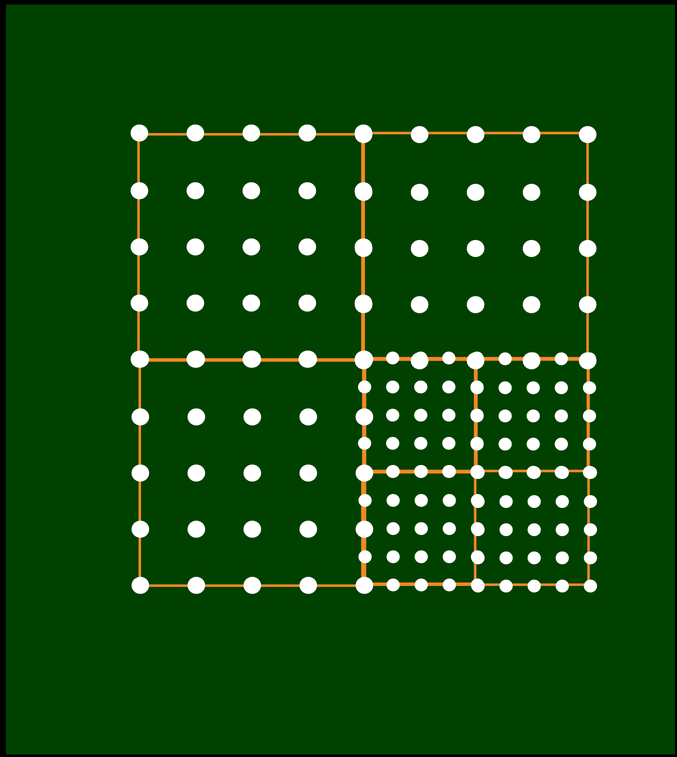


Lindstrom’s algorithm

Vertices are grouped into blocks, sorted by error value.

Frame coherence is used to limit the range of vertices that are tested each frame.

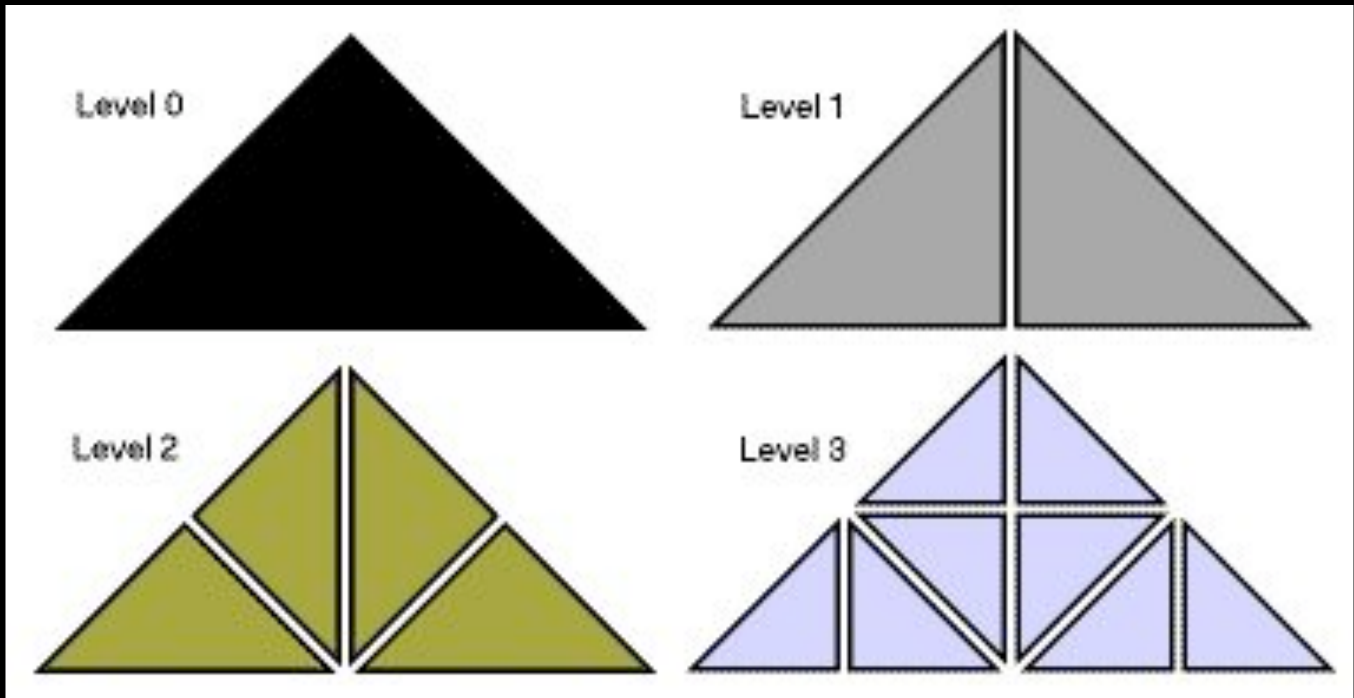
This gives Lindstrom much of its speed.



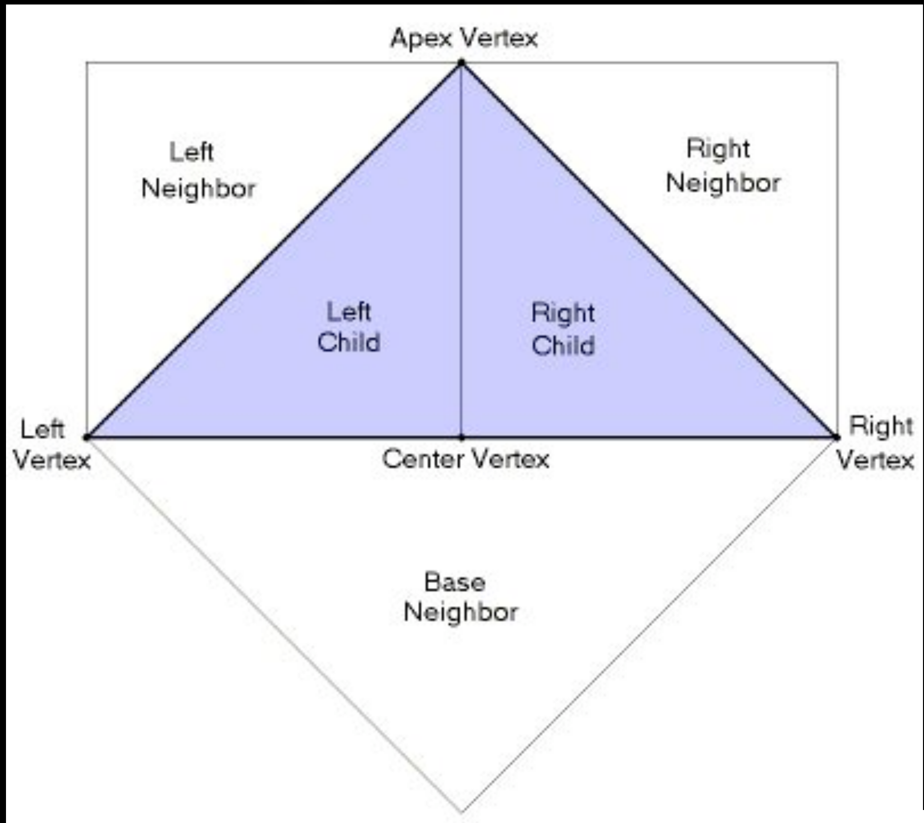
ROAM algorithm

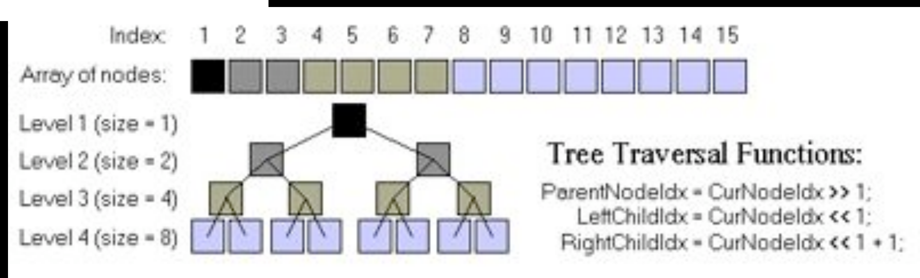
- Presented at IEEE Visualization '97, paper available on Web
- Top-down; progressively refines bounding volumes

Triangular Bintree

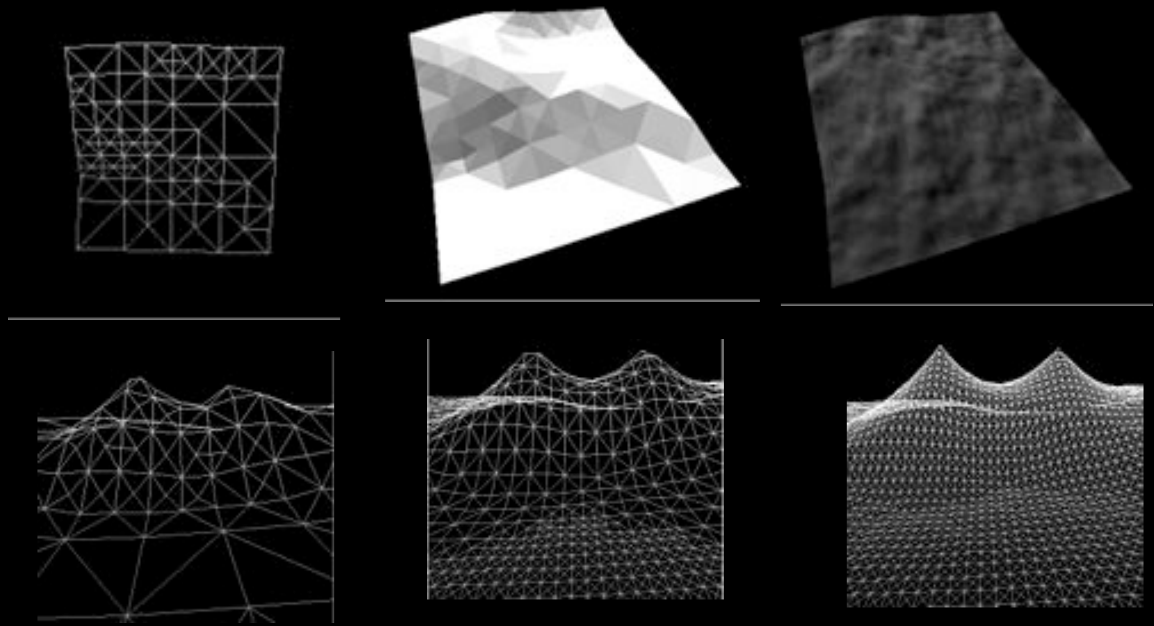


ROAM algorithm

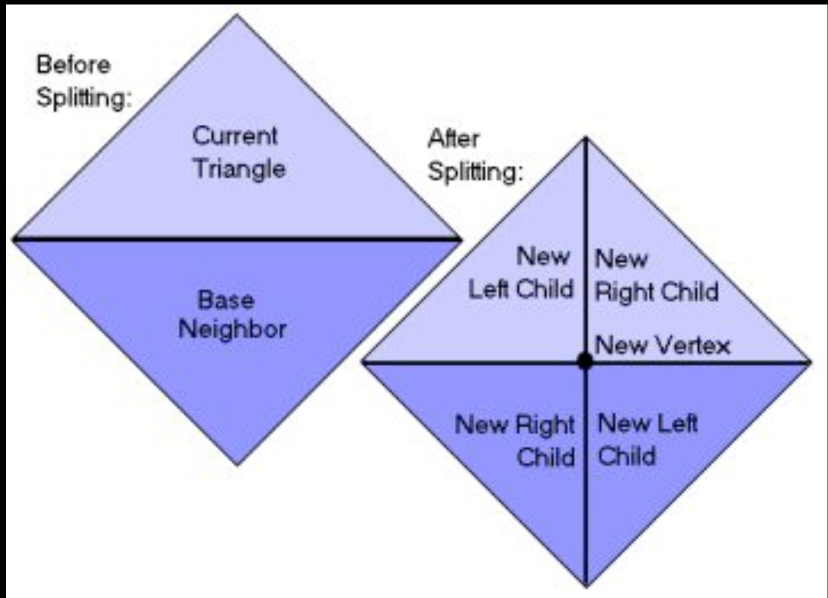




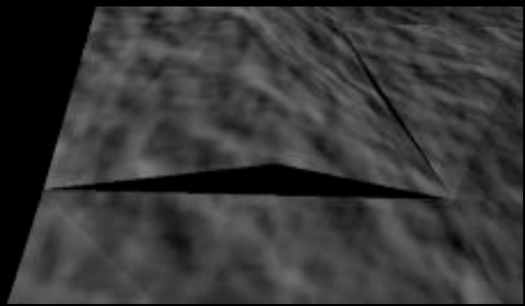
ROAM algorithm

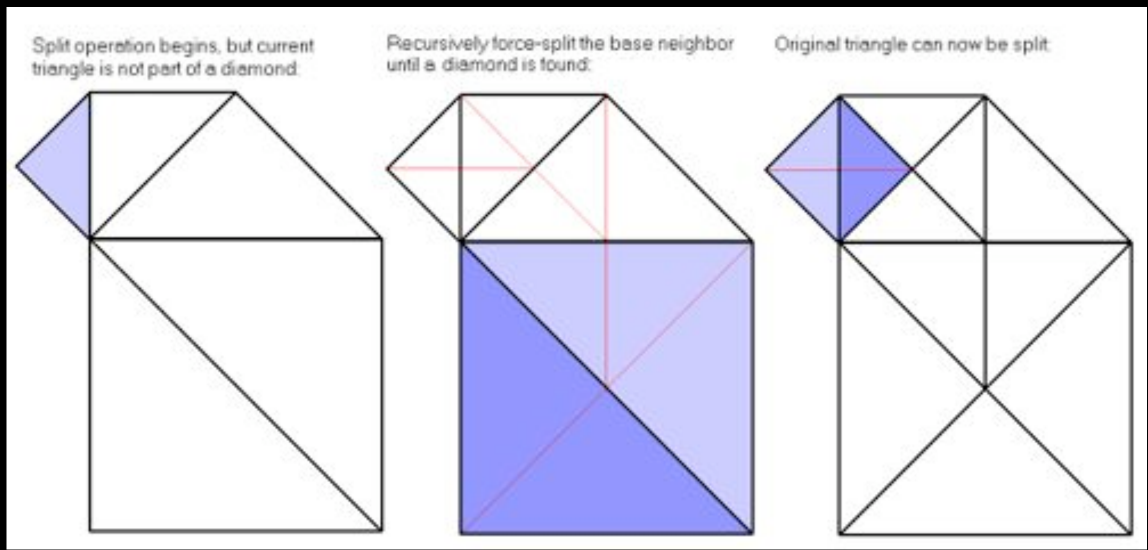


ROAM algorithm



ROAM algorithm





ROAM algorithm

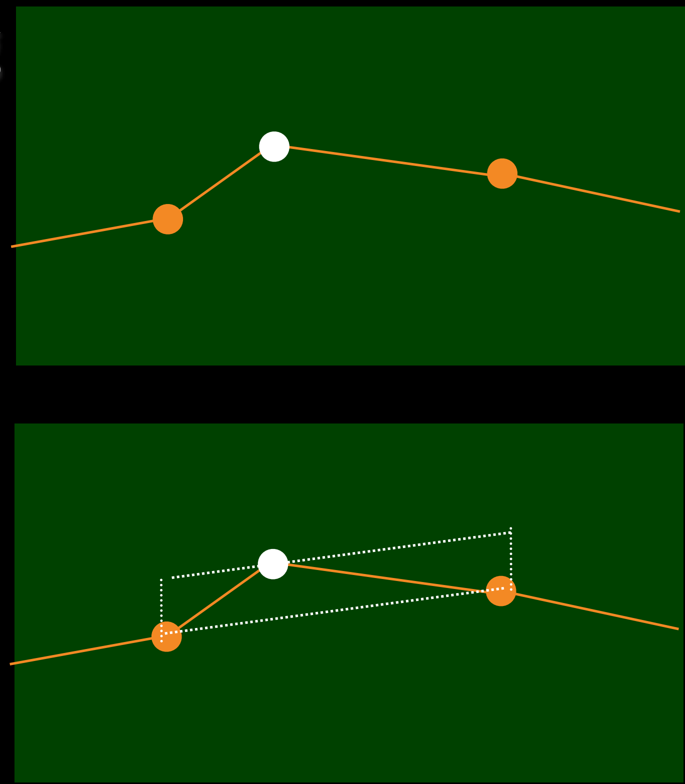
1. Find the highest priority triangle T in the queue
2. Force split T
3. Update the queue by removing T and other split triangles
4. Add any new triangles to the queue

ROAM algorithm

The thickness of each bounding volume, projected to the screen, controls when it is subdivided

No block structure

These bounding volumes are prioritized; priorities decay as the viewpoint moves.



Metrics

- Backface detail reduction
- Normal distortions
- Texture-coordinate distortion
- Silhouette edges
- View Frustum
- Atmospheric Obscurance
- Object Positioning