

RTX
OFF

239 FPS



23 FPS

Vulkan

OpenCL™

NVIDIA
CUDA

Вычисления на видеокартах

Лекция 9 - Real-time BVH для Ray Tracing

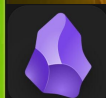
- Ray Tracing, AABB, BVH, Z curve, Morton Code
- Linear BVH (LBVH), H-PLOC
- Как читать научные статьи

@UnicornGlade

@PolarNick239

polarnick239@gmail.com

Николай Полярный



Obsidian

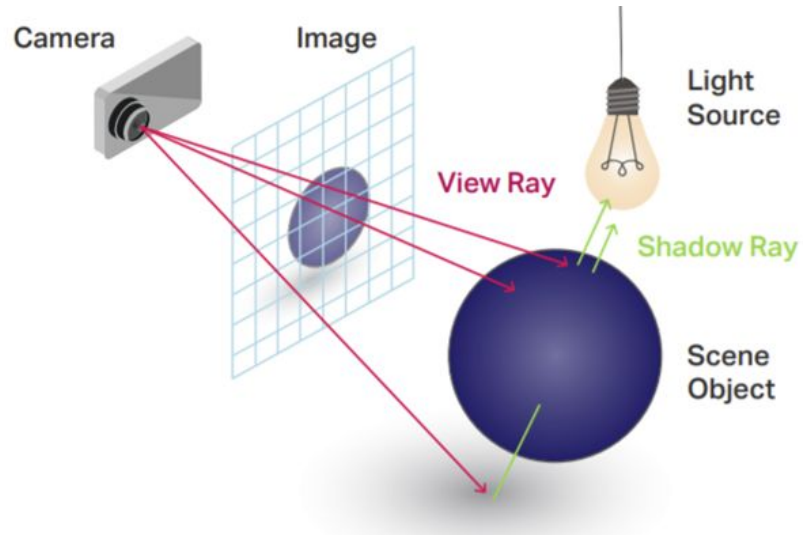


RTX
ON

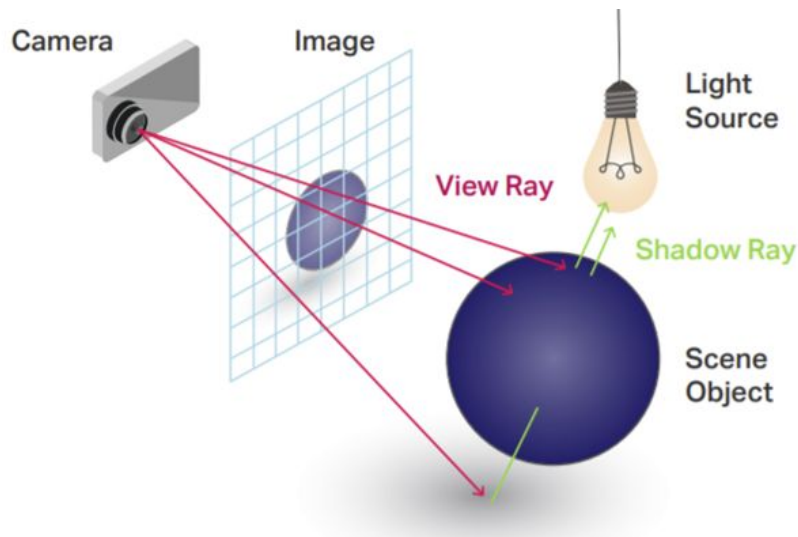
Глава 1: Ray Tracing

Axis-Aligned Bounding Boxes (AABB),
Bounding Volume Hierarchy (BVH)

Ray Tracing

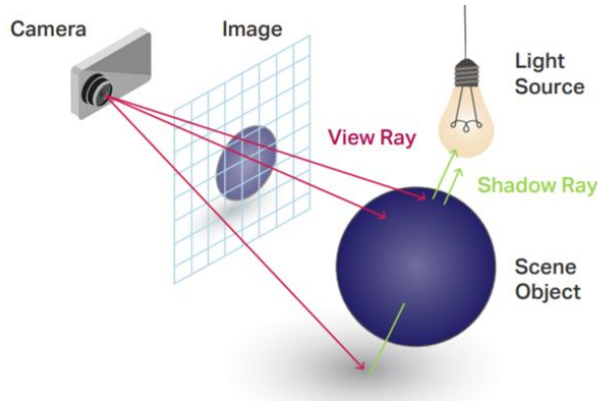


Ray Tracing

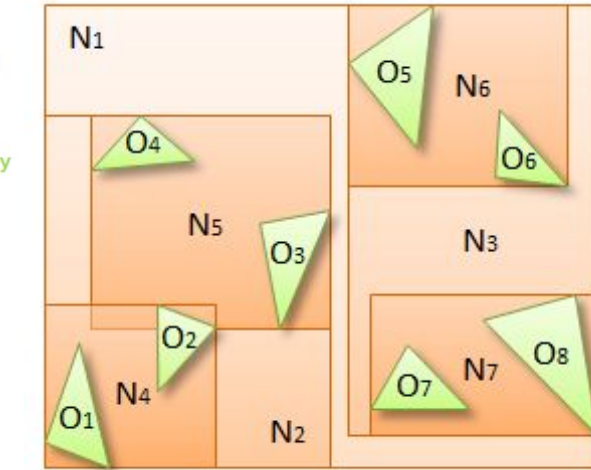


Как найти пересечение с треугольником?
Перебором для каждого луча всех
треугольников?

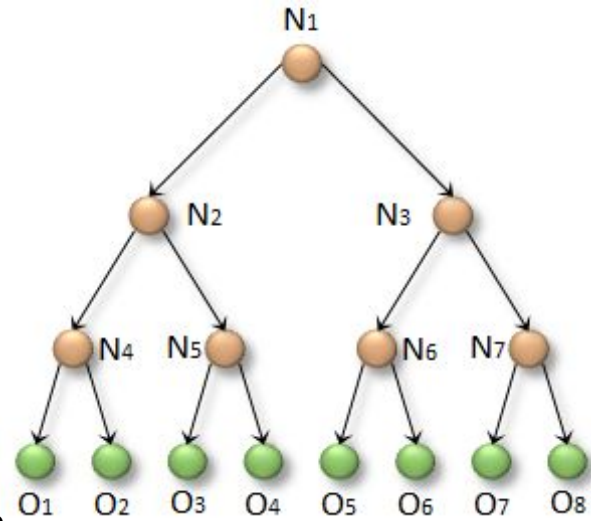
Ray Tracing



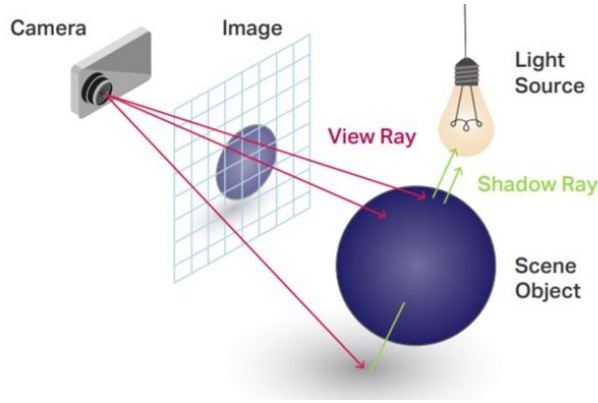
BVH - Bounding Volume Hierarchy



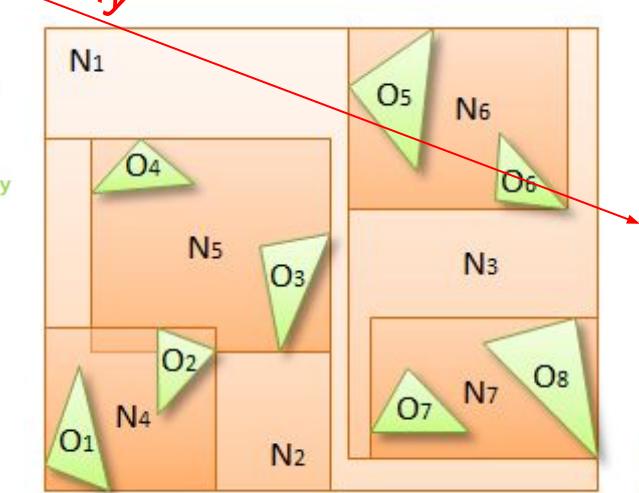
Axis-Aligned Bounding Boxes



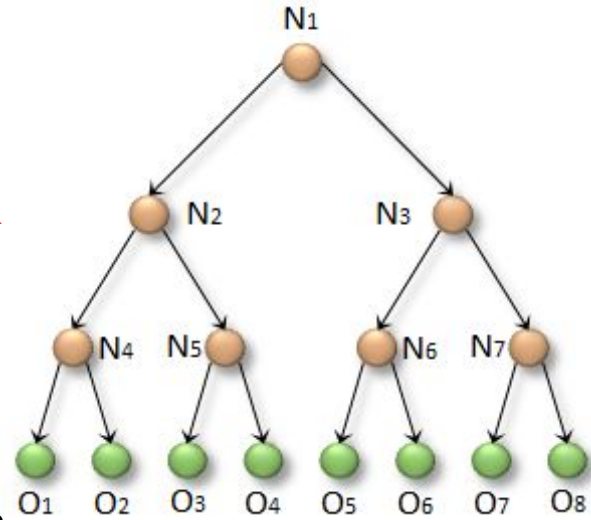
Ray Tracing



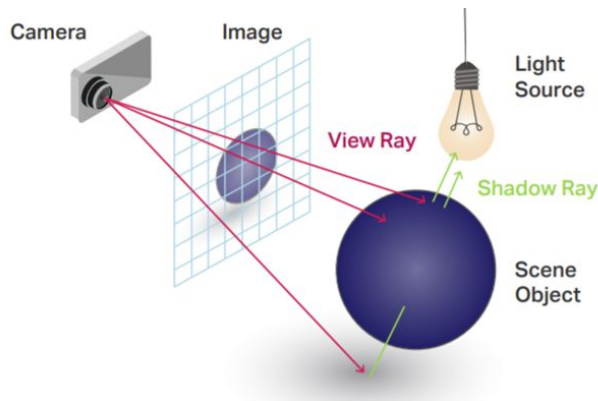
BVH - Bounding Volume Hierarchy



Axis-Aligned Bounding Boxes

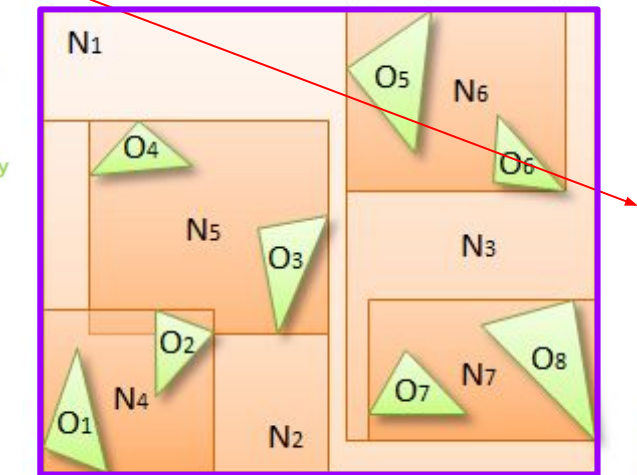


Ray Tracing

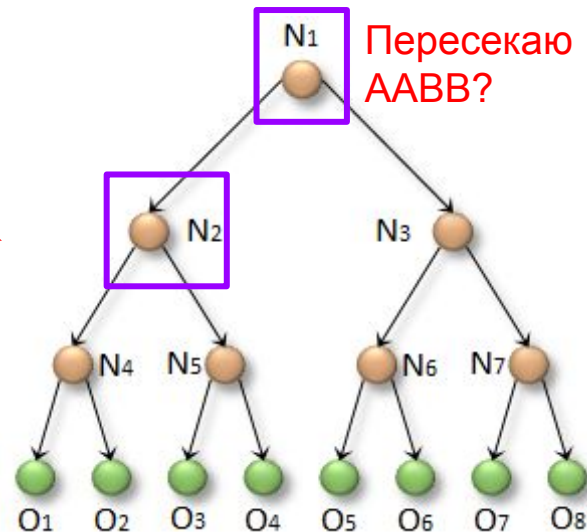


View Ray

BVH - Bounding Volume Hierarchy

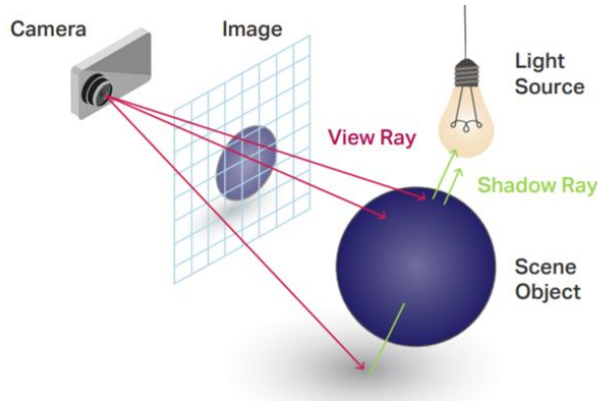


Axis-Aligned Bounding Boxes

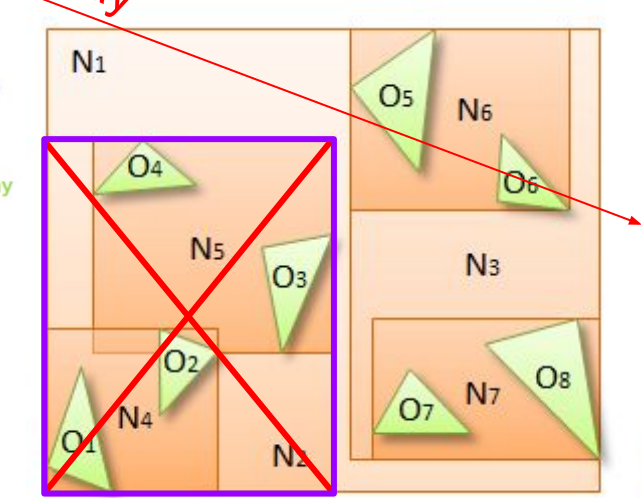


Пересекает
AABB?

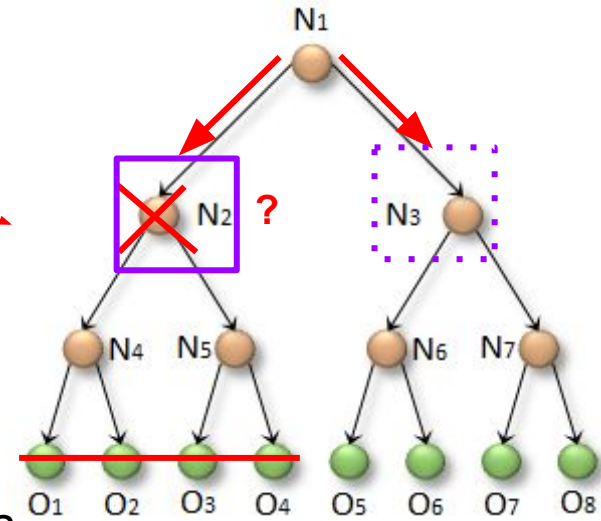
Ray Tracing



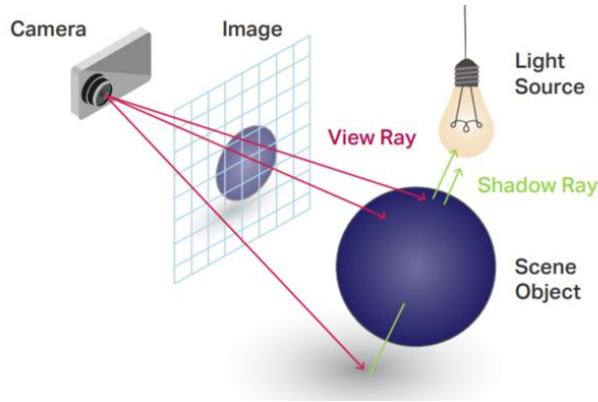
BVH - Bounding Volume Hierarchy



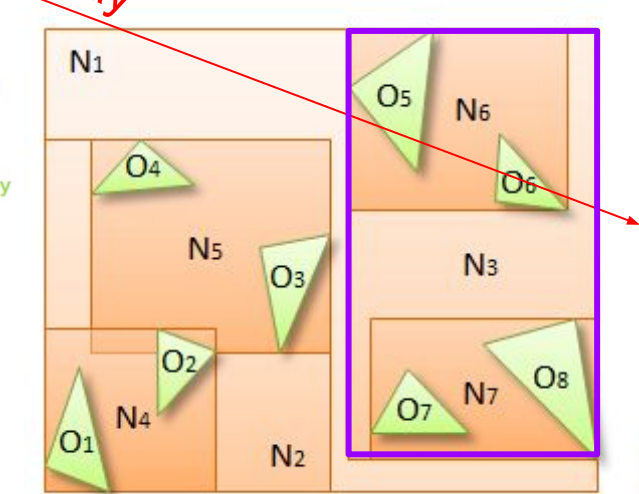
Axis-Aligned Bounding Boxes



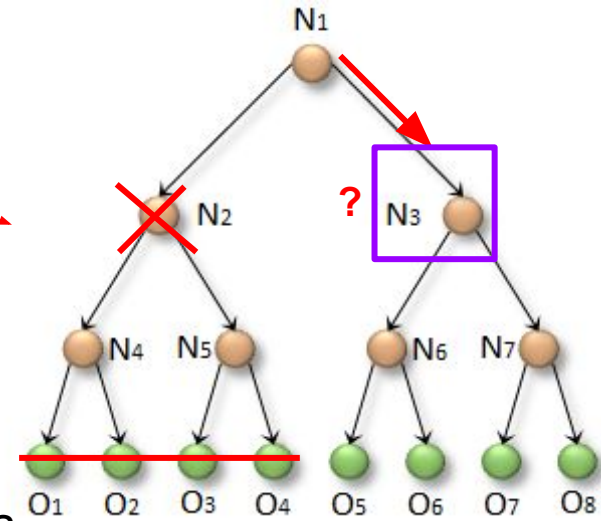
Ray Tracing



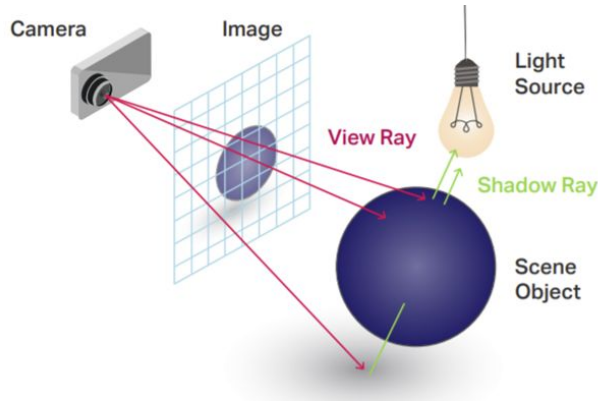
BVH - Bounding Volume Hierarchy



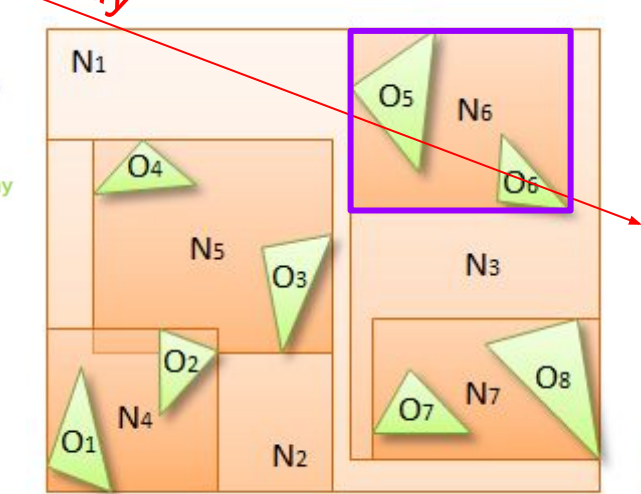
Axis-Aligned Bounding Boxes



Ray Tracing



BVH - Bounding Volume Hierarchy



Axis-Aligned Bounding Boxes

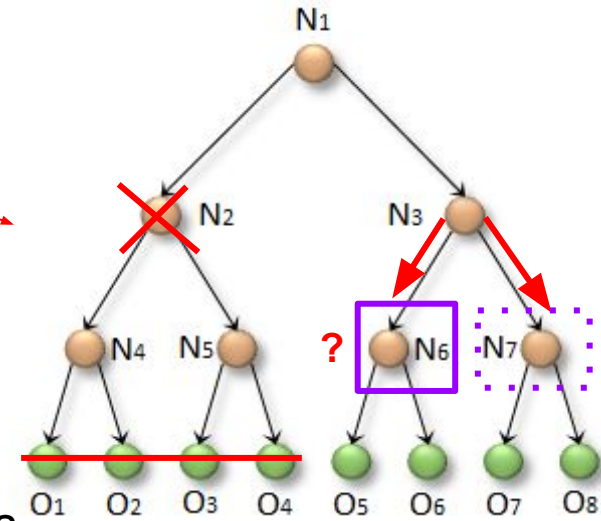


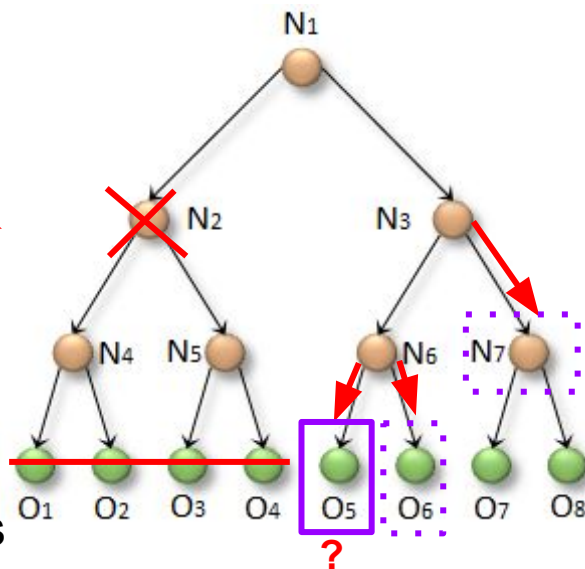
Diagram illustrating the Ray Tracing process:

- Camera:** The source of the View Rays.
- Image:** A grid representing the image plane where the rays intersect.
- View Ray:** Red lines originating from the camera and passing through the image plane to the scene object.
- Light Source:** A yellow light bulb emitting Shadow Rays.
- Shadow Ray:** Green lines originating from the light source and passing through the scene object.
- Scene Object:** A blue sphere representing the object being rendered.

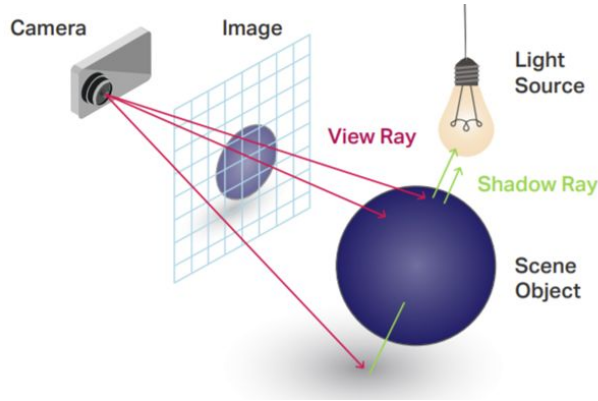


The diagram illustrates a hierarchical tree structure. The root node is N1, which branches into N2 and N3. N2 branches into N4 and N5. N3 branches into N6 and N7. N4 branches into O1 and O2. N5 branches into O3 and O4. N6 branches into O5 and O6. N7 branches into O7 and O8. A red arrow points from N1 to N3, and a purple box highlights N5.

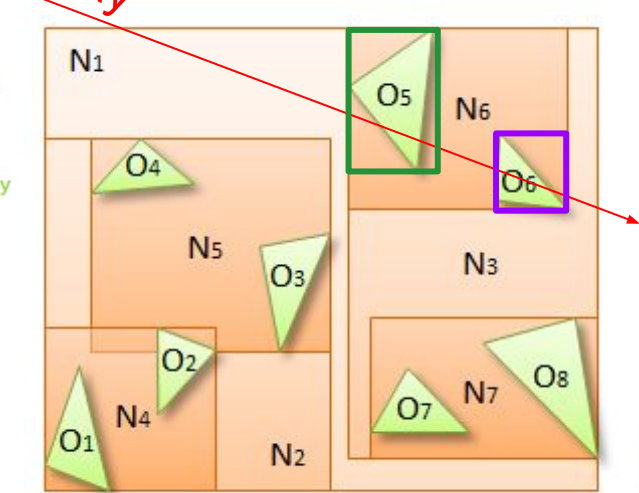
Axis-Aligned Bounding Boxes



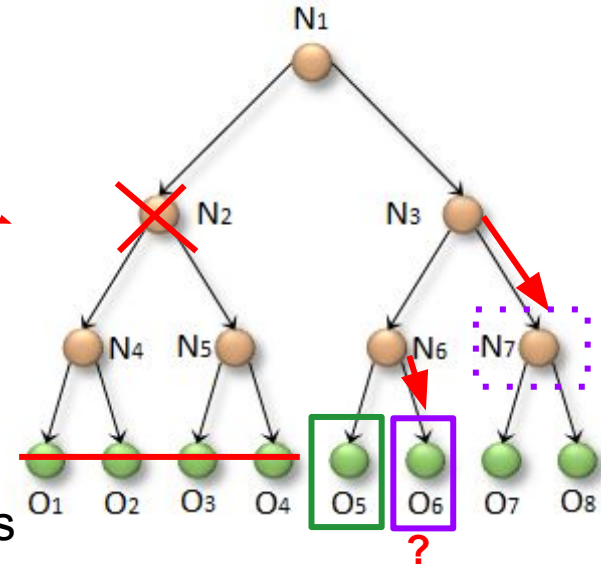
Ray Tracing



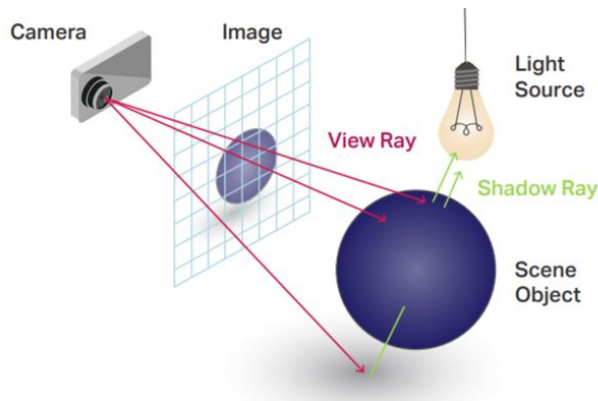
BVH - Bounding Volume Hierarchy



Axis-Aligned Bounding Boxes

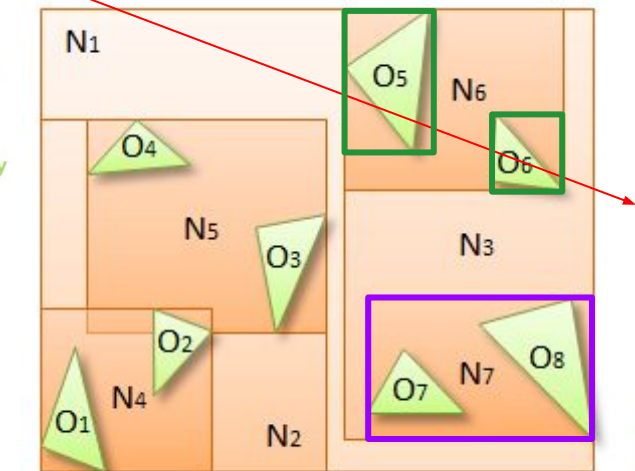


Ray Tracing

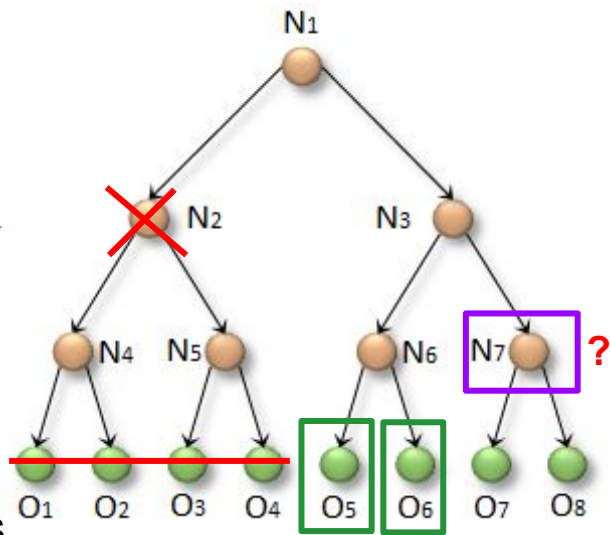


BVH - Bounding Volume Hierarchy

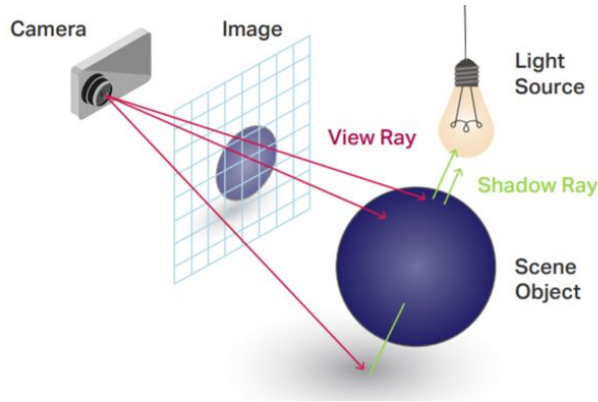
View Ray



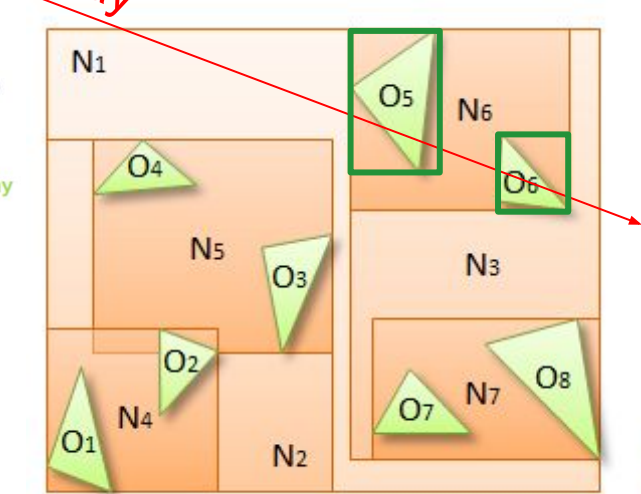
Axis-Aligned Bounding Boxes



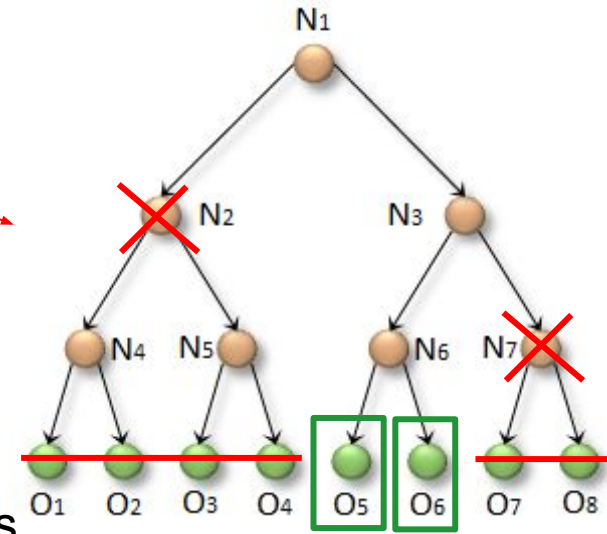
Ray Tracing



BVH - Bounding Volume Hierarchy

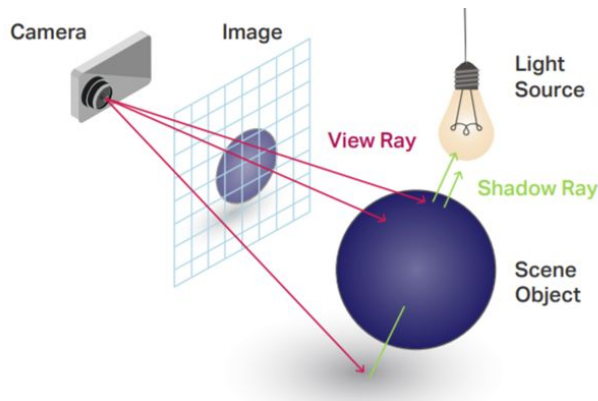


Axis-Aligned Bounding Boxes

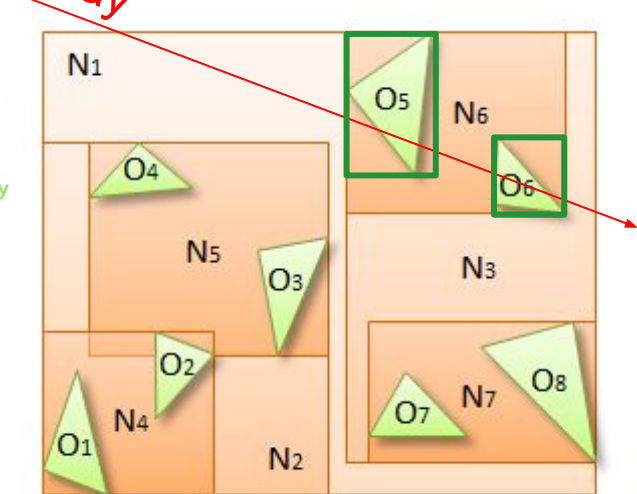


Вопросы?

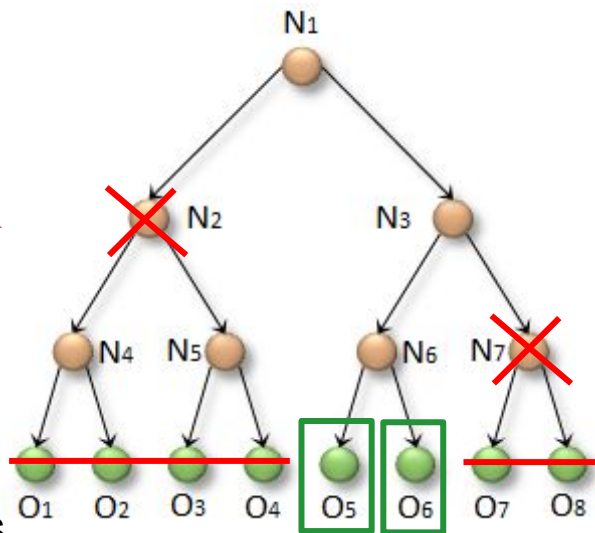
Ray Tracing



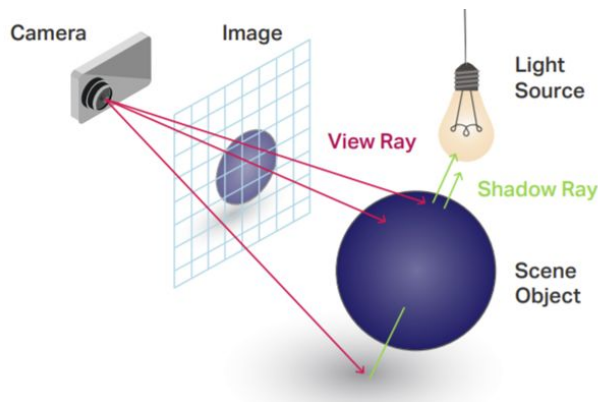
BVH - Bounding Volume Hierarchy



Axis-Aligned Bounding Boxes

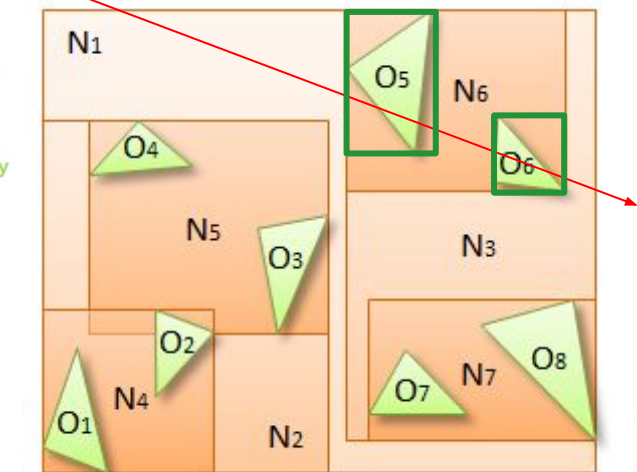


Ray Tracing

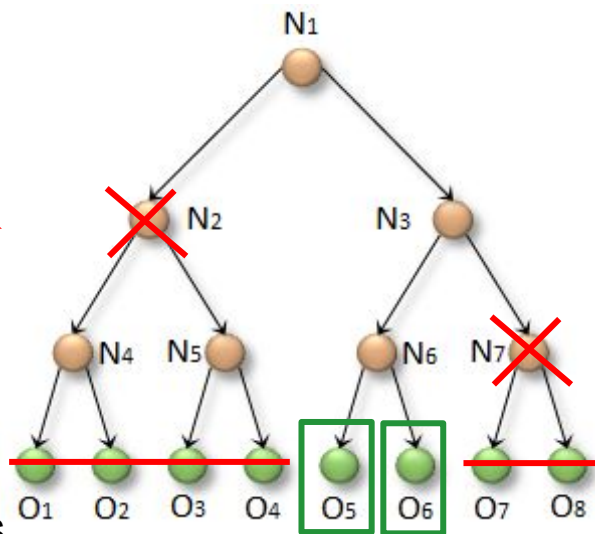


View Ray

BVH - Bounding Volume Hierarchy



Axis-Aligned Bounding Boxes



Как это будет выглядеть в коде?

BVH обход (рекурсивный)

```
void traverseRecursive(AABB
```

```
    int
```

```
    const __global BVHNode* node,
```

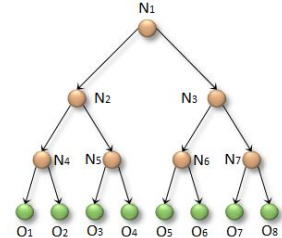
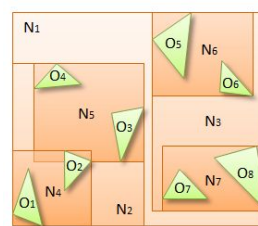
```
    __local CollisionList* list)
```

```
{
```

```
    if (checkOverlap(node->getAABB(), queryAABB)) {
```

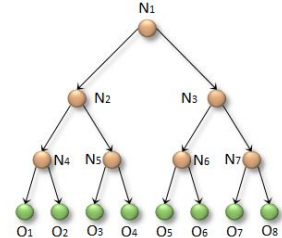
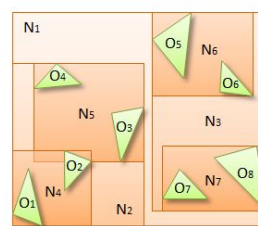
```
        queryAABB,
```

```
        queryObjectIdx,
```



BVH обход (рекурсивный)

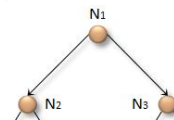
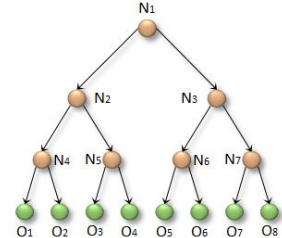
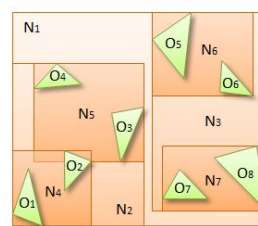
```
void traverseRecursive(AABB queryAABB,
                      int queryObjectIdx,
                      const __global BVHNode* node,
                      __local CollisionList* list)
{
    if (checkOverlap(node->getAABB(), queryAABB)) {
        if (node->isLeaf()) {
            list.add(queryObjectIdx, node->getObjectIdx());
        }
    }
}
```



BVH обход (рекурсивный)

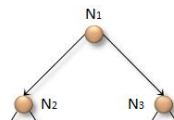
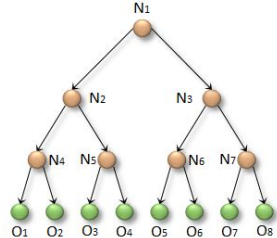
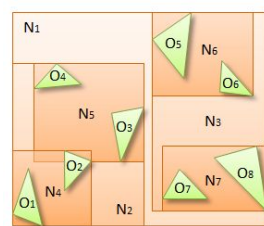
```
void traverseRecursive(AABB queryAABB,
                      int queryObjectIdx,
                      const __global BVHNode* node,
                      __local CollisionList* list)
{
    if (checkOverlap(node->getAABB(), queryAABB)) {
        if (node->isLeaf()) {
            list.add(queryObjectIdx, node->getObjectIdx());
        } else {
            const __global BVHNode childL = node->getLeftChild();
            const __global BVHNode childR = node->getRightChild();

```



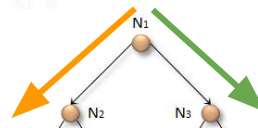
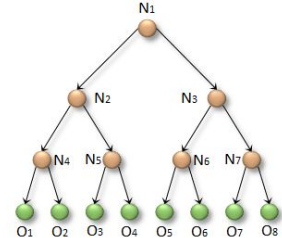
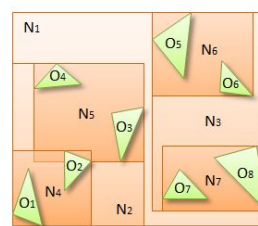
BVH обход (рекурсивный)

```
void traverseRecursive(AABB queryAABB,
                      int queryObjectIdx,
                      const __global BVHNode* node,
                      __local CollisionList* list)
{
    if (checkOverlap(node->getAABB(), queryAABB)) {
        if (node->isLeaf()) {
            list.add(queryObjectIdx, node->getObjectIdx());
        } else {
            const __global BVHNode childL = node->getLeftChild();
            const __global BVHNode childR = node->getRightChild();
            traverseRecursive(queryAABB, queryObjectIdx,
                             childL, list);
            traverseRecursive(queryAABB, queryObjectIdx,
                             childR, list);
        }
    }
}
```



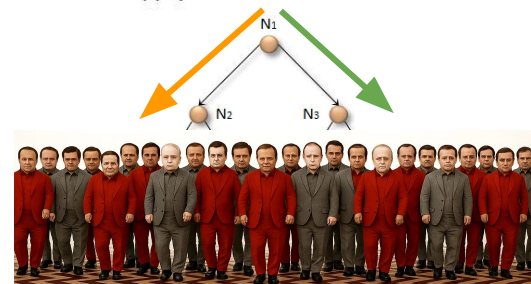
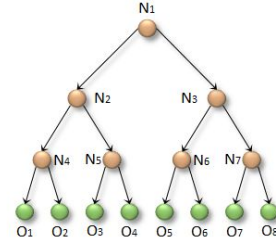
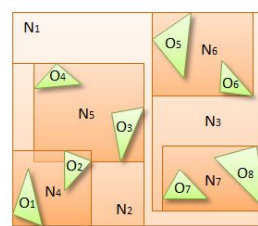
BVH обход (рекурсивный)

```
void traverseRecursive(AABB queryAABB,
                      int queryObjectIdx,
                      const __global BVHNode* node,
                      __local CollisionList* list)
{
    if (checkOverlap(node->getAABB(), queryAABB)) {
        if (node->isLeaf()) {
            list.add(queryObjectIdx, node->getObjectIdx());
        } else {
            const __global BVHNode childL = node->getLeftChild();
            const __global BVHNode childR = node->getRightChild();
            traverseRecursive(queryAABB, queryObjectIdx,
                             childL, list);
            traverseRecursive(queryAABB, queryObjectIdx,
                             childR, list);
        }
    }
}
```



BVH обход (рекурсивный)

```
void traverseRecursive(AABB queryAABB,
                      int queryObjectIdx,
                      const __global BVHNode* node,
                      __local CollisionList* list)
{
    if (checkOverlap(node->getAABB(), queryAABB)) {
        if (node->isLeaf()) {
            list.add(queryObjectIdx, node->getObjectIdx());
        } else {
            const __global BVHNode childL = node->getLeftChild();
            const __global BVHNode childR = node->getRightChild();
            traverseRecursive(queryAABB, queryObjectIdx,
                             childL, list);
            traverseRecursive(queryAABB, queryObjectIdx,
                             childR, list);
        }
    }
}
```



Code divergence!

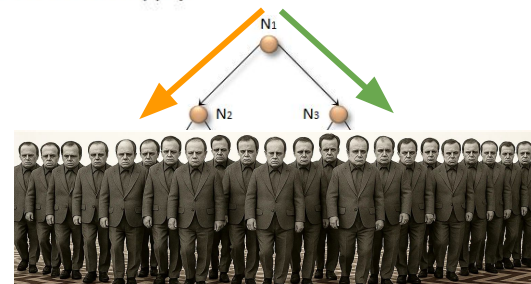
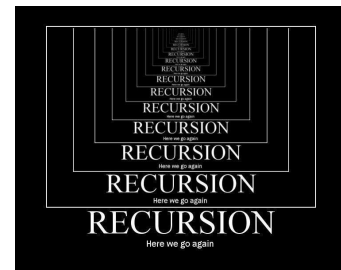
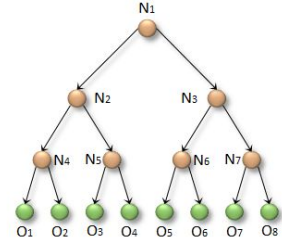
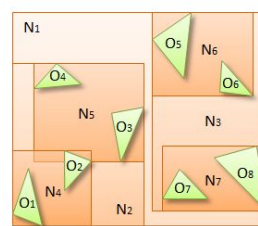
BVH обход (рекурсивный)

```

void traverseRecursive(AABB
    int
    const __global BVHNode* node,
    __local CollisionList* list)

{
    if (checkOverlap(node->getAABB(), queryAABB)) {
        if (node->isLeaf()) {
            list.add(queryObjectIdx, node->getObjectIdx());
        } else {
            const __global BVHNode childL = node->getLeftChild();
            const __global BVHNode childR = node->getRightChild();
            traverseRecursive(queryAABB, queryObjectIdx,
                             childL, list);
            traverseRecursive(queryAABB, queryObjectIdx,
                             childR, list);
        }
    }
}
    
```

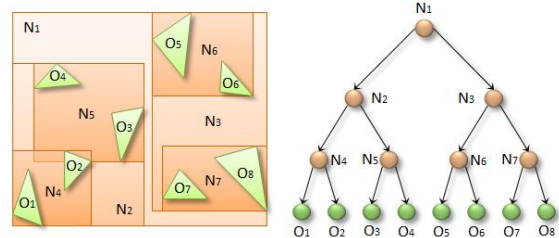
Что делать?



Code divergence!

BVH обход (на стеке)

```
BVHNode* stack[MAX_STACK_SIZE];  
BVHNode* node = bvhRoot;  
do {
```



BVH обход (на стеке)

```
BVHNode* stack[MAX_STACK_SIZE];
```

```
BVHNode* node = bvhRoot;
```

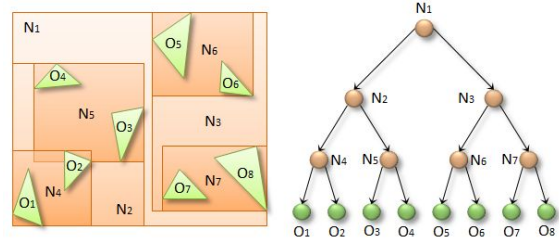
```
do {
```

```
    BVHNode* childL = node->getLeftChild();
```

```
    BVHNode* childR = node->getRightChild();
```

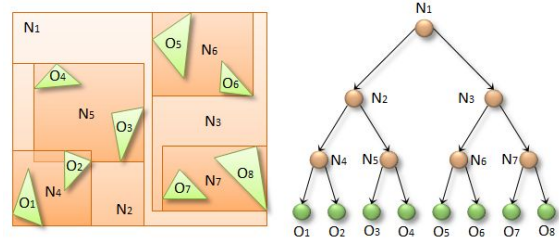
```
    bool overlapL = checkOverlap(queryAABB, childL->getAABB());
```

```
    bool overlapR = checkOverlap(queryAABB, childR->getAABB());
```



BVH обход (на стеке)

```
BVHNode* stack[MAX_STACK_SIZE];  
BVHNode* node = bvhRoot;  
do {  
    BVHNode* childL = node->getLeftChild();  
    BVHNode* childR = node->getRightChild();  
    bool overlapL = checkOverlap(queryAABB, childL->getAABB());  
    bool overlapR = checkOverlap(queryAABB, childR->getAABB());  
  
    if (overlapL && childL->isLeaf())  
        list.add(queryObjectIdx, childL->getObjectIdx());  
    if (overlapR && childR->isLeaf())  
        list.add(queryObjectIdx, childR->getObjectIdx());  
}
```



BVH обход (на стеке)

```
BVHNode* stack[MAX_STACK_SIZE];
```

```
BVHNode* node = bvhRoot;
```

```
do {
```

```
    BVHNode* childL = node->getLeftChild();
```

```
    BVHNode* childR = node->getRightChild();
```

```
    bool overlapL = checkOverlap(queryAABB, childL->getAABB());
```

```
    bool overlapR = checkOverlap(queryAABB, childR->getAABB());
```

```
    if (overlapL && childL->isLeaf())
```

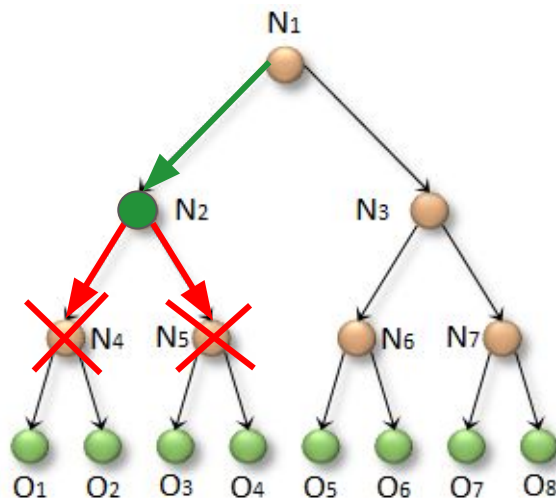
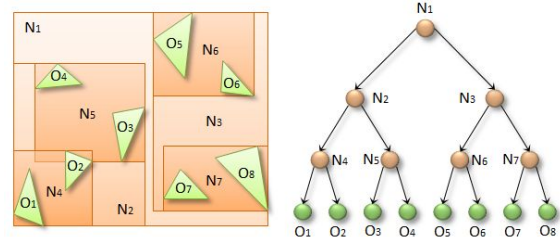
```
        list.add(queryObjectIdx, childL->getObjectIdx());
```

```
    if (overlapR && childR->isLeaf())
```

```
        list.add(queryObjectIdx, childR->getObjectIdx());
```

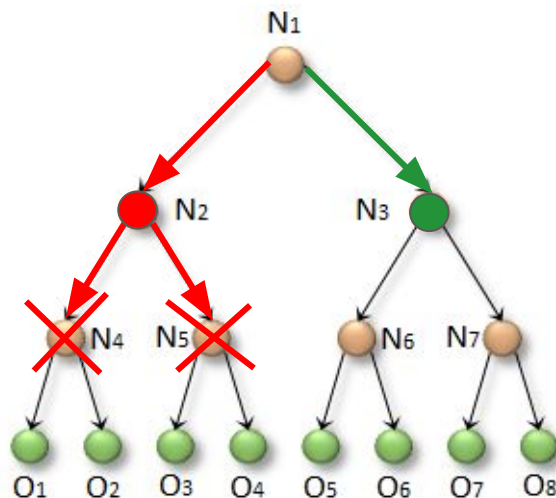
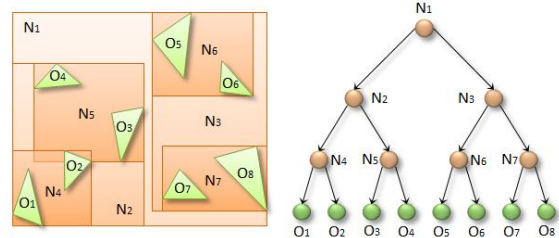
```
    if (!traverseL && !traverseR) {
```

куда нам тогда?



BVH обход (на стеке)

```
BVHNode* stack[MAX_STACK_SIZE];  
BVHNode* node = bvhRoot;  
do {  
    BVHNode* childL = node->getLeftChild();  
    BVHNode* childR = node->getRightChild();  
    bool overlapL = checkOverlap(queryAABB, childL->getAABB());  
    bool overlapR = checkOverlap(queryAABB, childR->getAABB());  
  
    if (overlapL && childL->isLeaf())  
        list.add(queryObjectIdx, childL->getObjectIdx());  
    if (overlapR && childR->isLeaf())  
        list.add(queryObjectIdx, childR->getObjectIdx());  
  
    if (!traverseL && !traverseR) {  
        node = stack.pop();  
    }  
}
```



BVH обход (на стеке)

```
BVHNode* stack[MAX_STACK_SIZE];
```

```
BVHNode* node = bvhRoot;
```

```
do {
```

```
    BVHNode* childL = node->getLeftChild();
```

```
    BVHNode* childR = node->getRightChild();
```

```
    bool overlapL = checkOverlap(queryAABB, childL->getAABB());
```

```
    bool overlapR = checkOverlap(queryAABB, childR->getAABB());
```

```
    if (overlapL && childL->isLeaf())
```

```
        list.add(queryObjectIdx, childL->getObjectIdx());
```

```
    if (overlapR && childR->isLeaf())
```

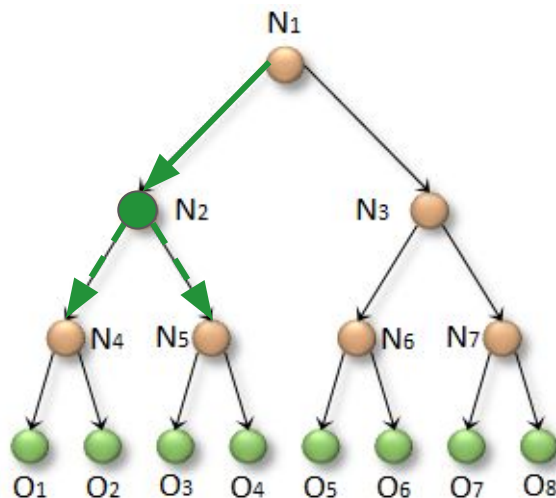
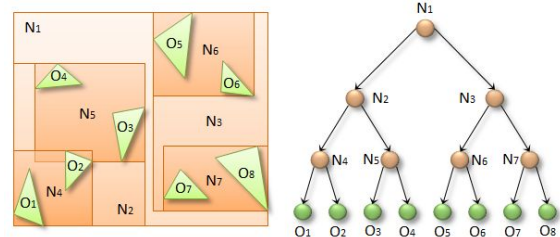
```
        list.add(queryObjectIdx, childR->getObjectIdx());
```

```
    if (!traverseL && !traverseR) {
```

```
        node = stack.pop();
```

```
    } else {
```

а что тогда?



BVH обход (на стеке)

```
BVHNode* stack[MAX_STACK_SIZE];
```

```
BVHNode* node = bvhRoot;
```

```
do {
```

```
    BVHNode* childL = node->getLeftChild();
```

```
    BVHNode* childR = node->getRightChild();
```

```
    bool overlapL = checkOverlap(queryAABB, childL->getAABB());
```

```
    bool overlapR = checkOverlap(queryAABB, childR->getAABB());
```

```
    if (overlapL && childL->isLeaf())
```

```
        list.add(queryObjectIdx, childL->getObjectIdx());
```

```
    if (overlapR && childR->isLeaf())
```

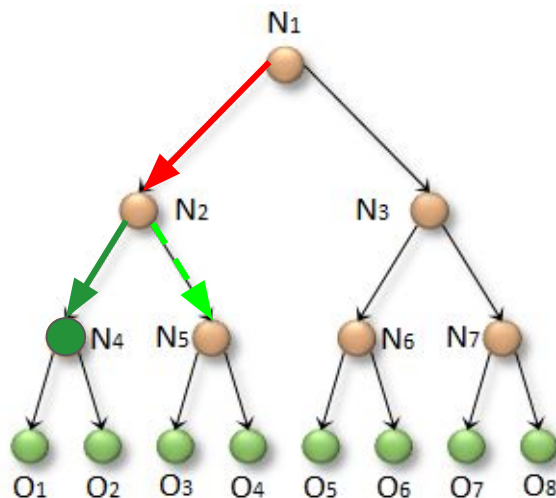
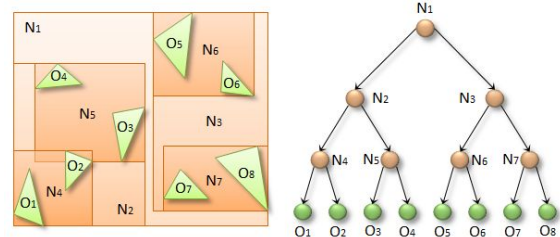
```
        list.add(queryObjectIdx, childR->getObjectIdx());
```

```
    if (!traverseL && !traverseR) {
```

```
        node = stack.pop();
```

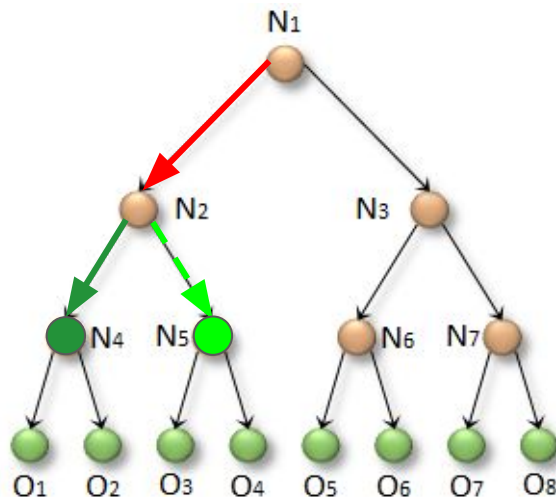
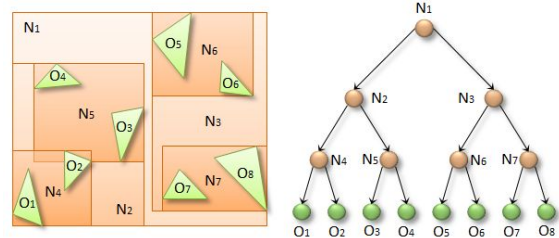
```
    } else {
```

```
        node = traverseL ? childL : childR;
```



BVH обход (на стеке)

```
BVHNode* stack[MAX_STACK_SIZE];  
BVHNode* node = bvhRoot;  
do {  
    BVHNode* childL = node->getLeftChild();  
    BVHNode* childR = node->getRightChild();  
    bool overlapL = checkOverlap(queryAABB, childL->getAABB());  
    bool overlapR = checkOverlap(queryAABB, childR->getAABB());  
  
    if (overlapL && childL->isLeaf())  
        list.add(queryObjectIdx, childL->getObjectIdx());  
    if (overlapR && childR->isLeaf())  
        list.add(queryObjectIdx, childR->getObjectIdx());  
  
    if (!traverseL && !traverseR) {  
        node = stack.pop();  
    } else {  
        node = traverseL ? childL : childR;  
        if (traverseL && traverseR)  
            stack.push(childR);  
    }  
}
```



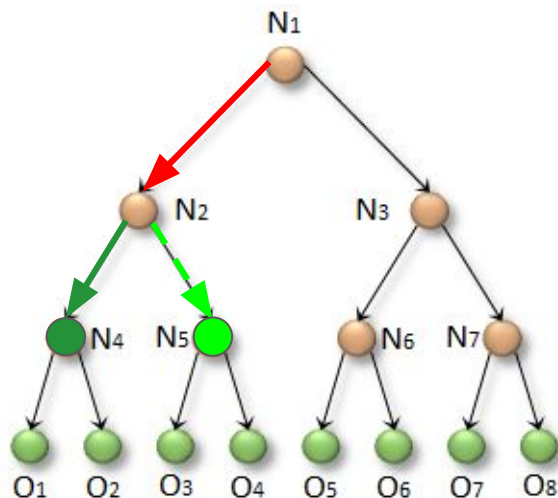
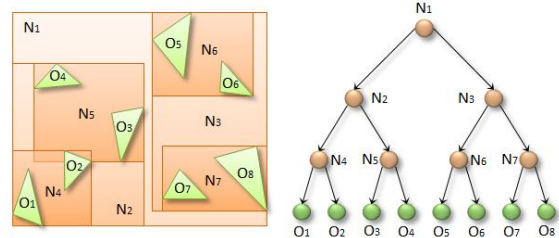
BVH обход (на стеке)

```
BVHNode* stack[MAX_STACK_SIZE];
BVHNode* node = bvhRoot;

do {
    BVHNode* childL = node->getLeftChild();
    BVHNode* childR = node->getRightChild();
    bool overlapL = checkOverlap(queryAABB, childL->getAABB());
    bool overlapR = checkOverlap(queryAABB, childR->getAABB());

    if (overlapL && childL->isLeaf())
        list.add(queryObjectIdx, childL->getObjectIdx());
    if (overlapR && childR->isLeaf())
        list.add(queryObjectIdx, childR->getObjectIdx());

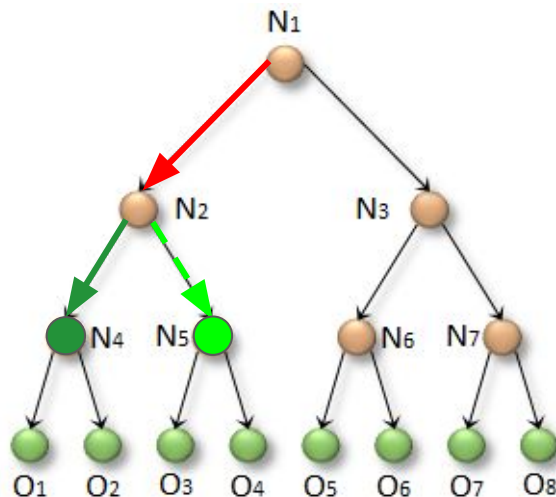
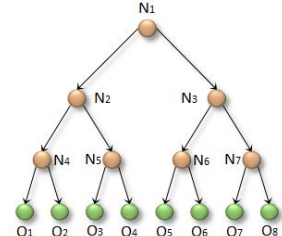
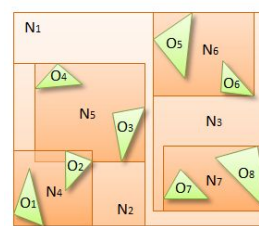
    if (!traverseL && !traverseR) {
        node = stack.pop();
    } else {
        node = traverseL ? childL : childR;
        if (traverseL && traverseR)
            stack.push(childR);
    }
} while (node != NULL);
```



BVH обход (на стеке)

```
BVHNode* stack[MAX_STACK_SIZE];  
BVHNode* node = bvhRoot;  
do {  
    BVHNode* childL = node->getLeftChild();  
    BVHNode* childR = node->getRightChild();  
    bool overlapL = checkOverlap(queryAABB, childL->getAABB());  
    bool overlapR = checkOverlap(queryAABB, childR->getAABB());  
  
    if (overlapL && childL->isLeaf())  
        list.add(queryObjectIdx, childL->getObjectIdx());  
    if (overlapR && childR->isLeaf())  
        list.add(queryObjectIdx, childR->getObjectIdx());  
  
    if (!traverseL && !traverseR) {  
        node = stack.pop();  
    } else {  
        node = traverseL ? childL : childR;  
        if (traverseL && traverseR)  
            stack.push(childR);  
    }  
} while (node != NULL);
```

Code divergence остался только
в виде числа итераций цикла.

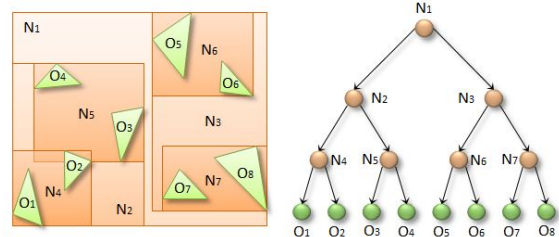


BVH обход (на стеке)

```
BVHNode* stack[MAX_STACK_SIZE];
BVHNode* node = bvhRoot;
do {
    BVHNode* childL = node->getLeftChild();
    BVHNode* childR = node->getRightChild();
    bool overlapL = checkOverlap(queryAABB, childL->getAABB());
    bool overlapR = checkOverlap(queryAABB, childR->getAABB());

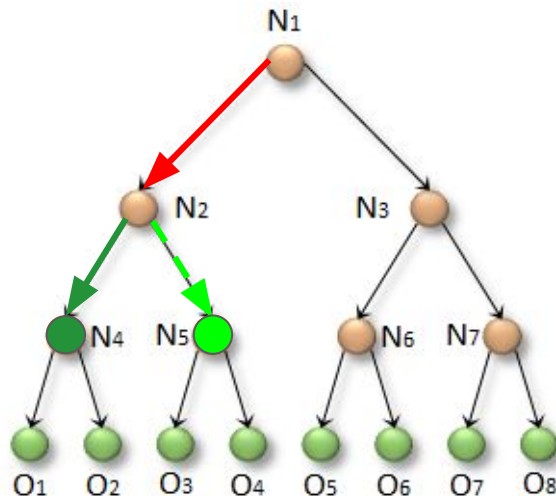
    if (overlapL && childL->isLeaf())
        list.add(queryObjectIdx, childL->getObjectIdx());
    if (overlapR && childR->isLeaf())
        list.add(queryObjectIdx, childR->getObjectIdx());

    if (!traverseL && !traverseR) {
        node = stack.pop();
    } else {
        node = traverseL ? childL : childR;
        if (traverseL && traverseR)
            stack.push(childR);
    }
} while (node != NULL);
```



Code divergence остался только
в виде числа итераций цикла.

Data divergence увеличивает
количество запрашиваемых
кеш-линий.



BVH обход (на стеке)

```
BVHNode* stack[MAX_STACK_SIZE];
```

```
BVHNode* node = bvhRoot;
```

```
do {
```

```
    BVHNode* childL = node->getLeftChild();
```

```
    BVHNode* childR = node->getRightChild();
```

```
    bool overlapL = checkOverlap(queryAABB, childL->getAABB());
```

```
    bool overlapR = checkOverlap(queryAABB, childR->getAABB());
```

```
    if (overlapL && childL->isLeaf())
```

```
        list.add(queryObjectIdx, childL->getObjectIdx());
```

```
    if (overlapR && childR->isLeaf())
```

```
        list.add(queryObjectIdx, childR->getObjectIdx());
```

```
    if (!traverseL && !traverseR) {
```

```
        node = stack.pop();
```

```
    } else {
```

```
        node = traverseL ? childL : childR;
```

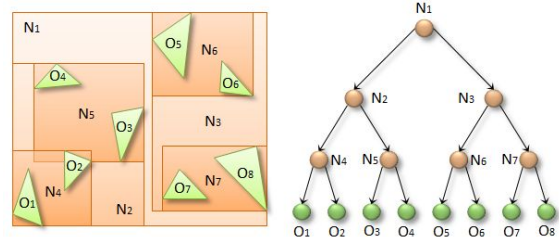
```
        if (traverseL && traverseR)
```

```
            stack.push(childR);
```

```
    }
```

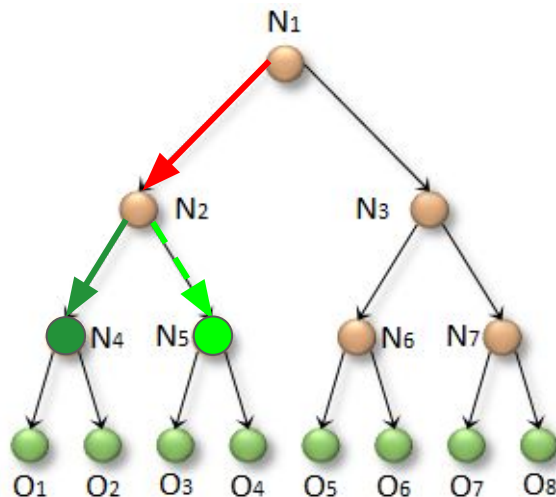
```
} while (node != NULL);
```

Как увеличить
когерентность по данным?



Code divergence остался только
в виде числа итераций цикла.

Data divergence увеличивает
количество запрашиваемых
кеш-линий.



BVH обход (на стеке)

```
BVHNode* stack[MAX_STACK_SIZE];
```

```
BVHNode* node = bvhRoot;
```

```
do {
```

```
    BVHNode* childL = node->getLeftChild();
```

```
    BVHNode* childR = node->getRightChild();
```

```
    bool overlapL = checkOverlap(queryAABB, childL->getAABB());
```

```
    bool overlapR = checkOverlap(queryAABB, childR->getAABB());
```

```
    if (overlapL && childL->isLeaf())
```

```
        list.add(queryObjectIdx, childL->getObjectIdx());
```

```
    if (overlapR && childR->isLeaf())
```

```
        list.add(queryObjectIdx, childR->getObjectIdx());
```

```
    if (!traverseL && !traverseR) {
```

```
        node = stack.pop();
```

```
    } else {
```

```
        node = traverseL ? childL : childR;
```

```
        if (traverseL && traverseR)
```

```
            stack.push(childR);
```

```
    }
```

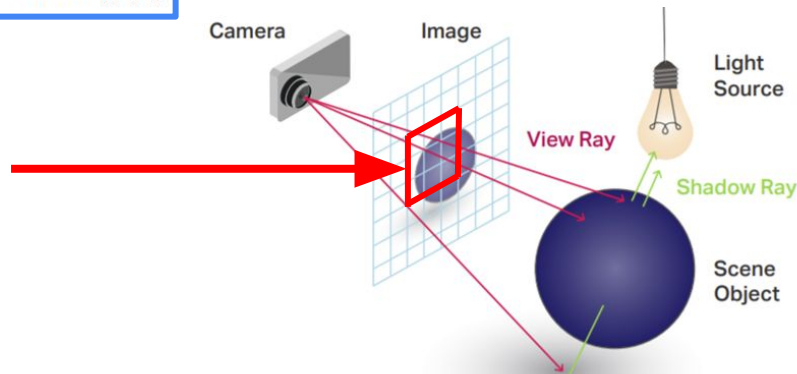
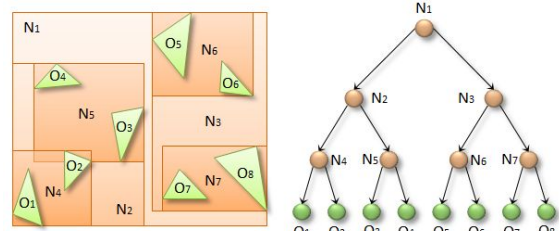
```
} while (node != NULL);
```

**Как увеличить
когерентность по данным?**

Code divergence остался только
в виде числа итераций цикла.

Data divergence увеличивает
количество запрашиваемых
кеш-линий.

**Данные когерентнее если в
одном warp - пучок лучей!**



BVH обход (на стеке)

Вопросы?

```
BVHNode* stack[MAX_STACK_SIZE];
```

```
BVHNode* node = bvhRoot;
```

```
do {
```

```
    BVHNode* childL = node->getLeftChild();
```

```
    BVHNode* childR = node->getRightChild();
```

```
    bool overlapL = checkOverlap(queryAABB, childL->getAABB());
```

```
    bool overlapR = checkOverlap(queryAABB, childR->getAABB());
```

```
    if (overlapL && childL->isLeaf())
```

```
        list.add(queryObjectIdx, childL->getObjectIdx());
```

```
    if (overlapR && childR->isLeaf())
```

```
        list.add(queryObjectIdx, childR->getObjectIdx());
```

```
    if (!traverseL && !traverseR) {
```

```
        node = stack.pop();
```

```
    } else {
```

```
        node = traverseL ? childL : childR;
```

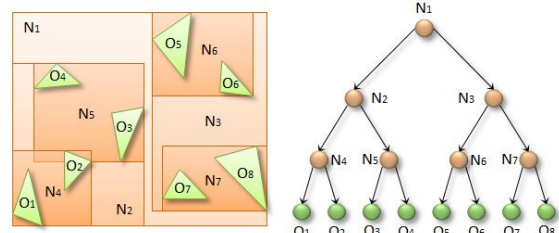
```
        if (traverseL && traverseR)
```

```
            stack.push(childR);
```

```
    }
```

```
} while (node != NULL);
```

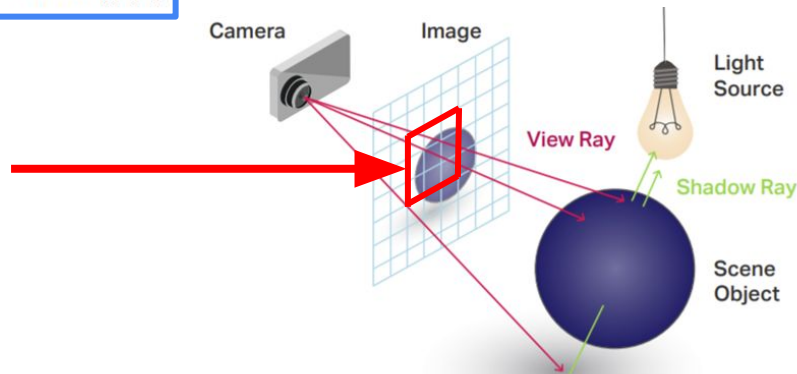
Как увеличить
когерентность по данным?



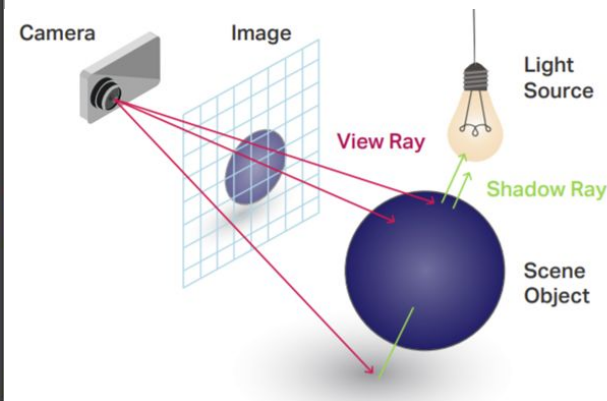
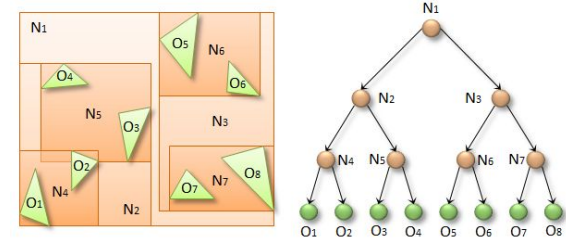
Code divergence остался только
в виде числа итераций цикла.

Data divergence увеличивает
количество запрашиваемых
кеш-линий.

Данные когерентнее если в
одном warp - пучок лучей!



Ray Tracing Cores



- 1) Обход BVH иерархии
- 2) Пересечение луча с треугольником

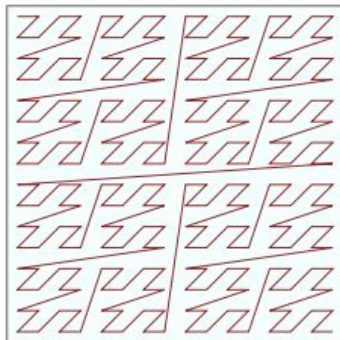
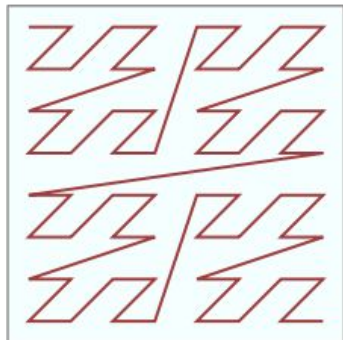
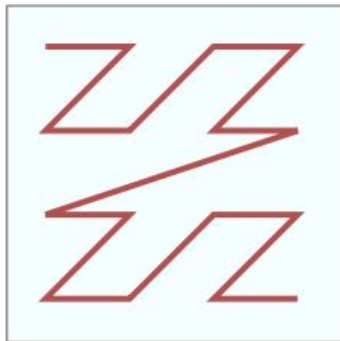


Глава 2:

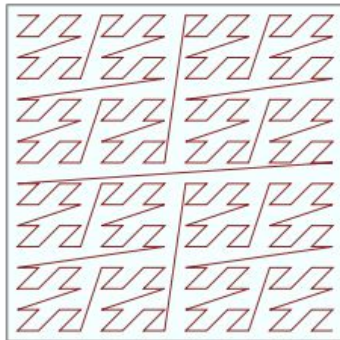
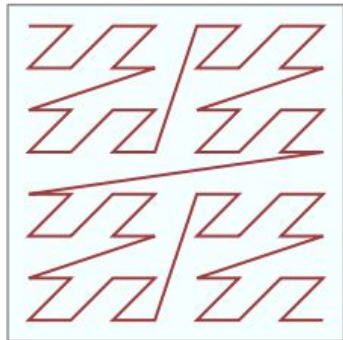
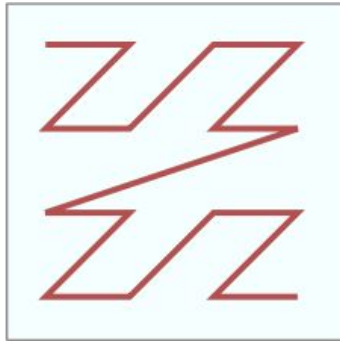
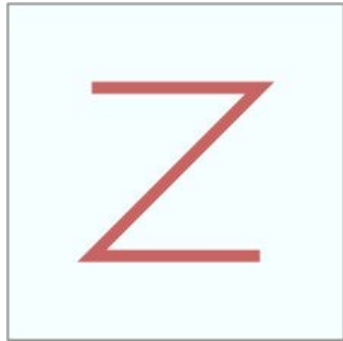
Z curve, Morton Code



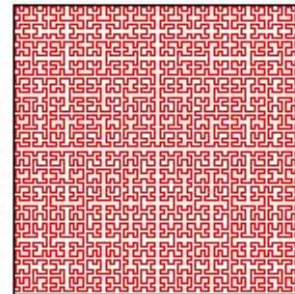
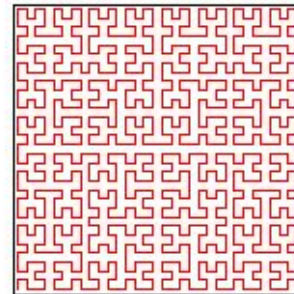
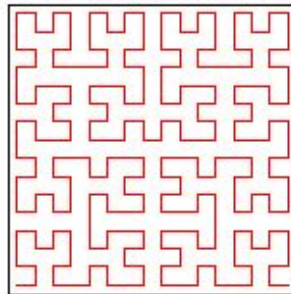
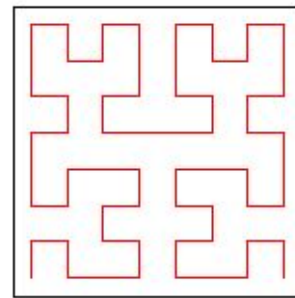
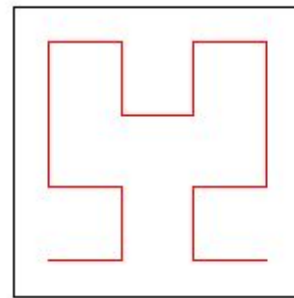
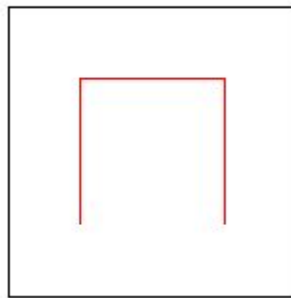
Z curve



Z curve

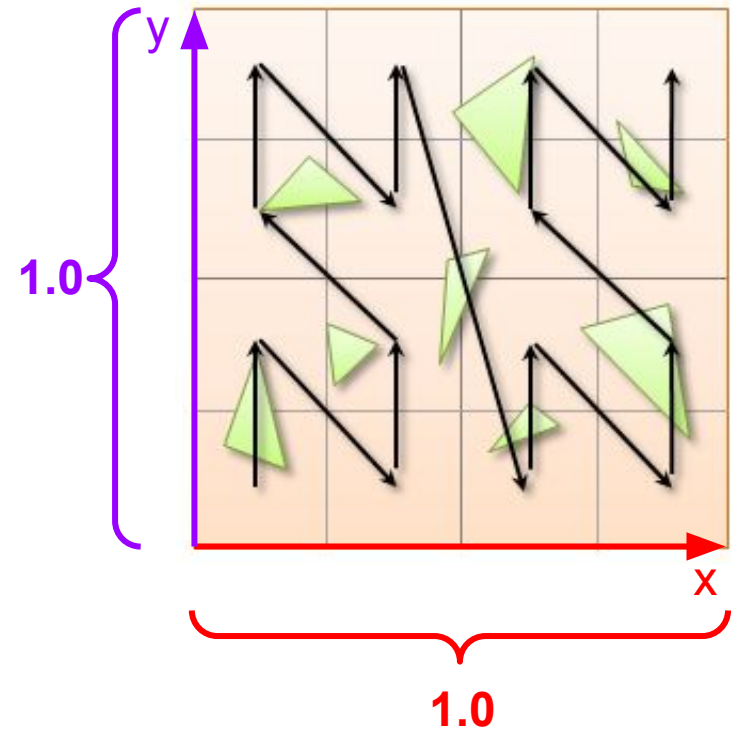


Hilbert curve



Z curve, Morton Code

```
// Calculates a 30-bit Morton code for the  
// given 3D point located within the unit cube [0,1].  
unsigned int morton3D(float x, float y, float z)
```



Z curve, Morton Code

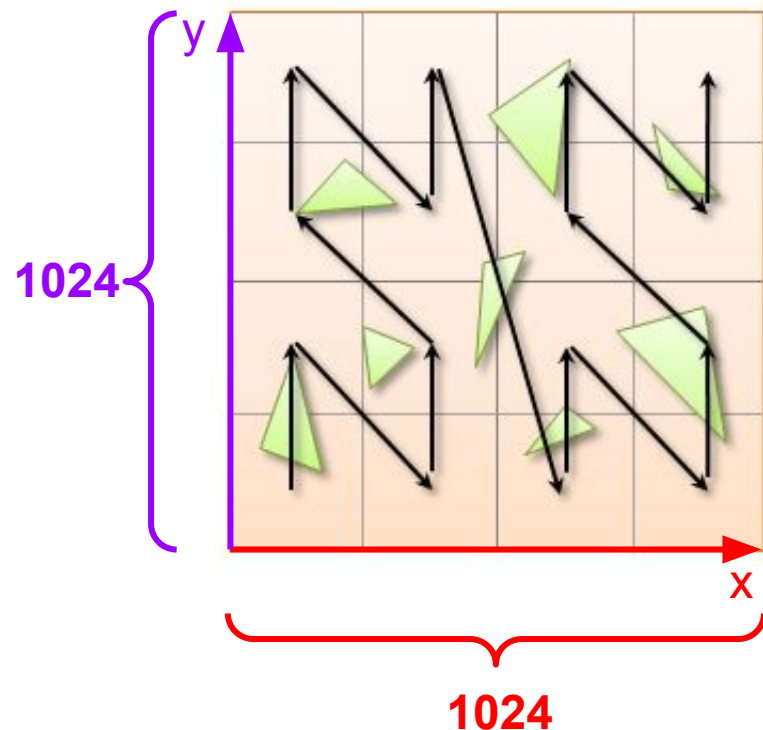
```
// Calculates a 30-bit Morton code for the  
// given 3D point located within the unit cube [0,1].  
unsigned int morton3D(float x, float y, float z)
```

```
{
```

```
    x = min(max(x * 1024.0f, 0.0f), 1023.0f);
```

```
    y = min(max(y * 1024.0f, 0.0f), 1023.0f);
```

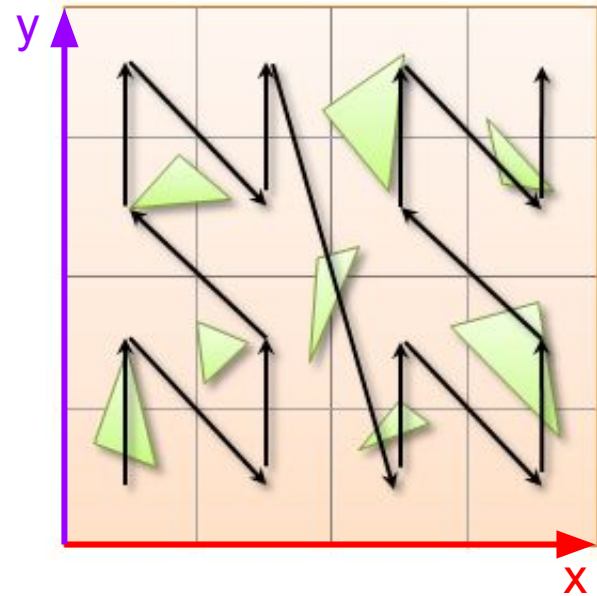
```
    z = min(max(z * 1024.0f, 0.0f), 1023.0f);
```



**Сколько бит нужно для кодирования 2D
кода Мортон?**

Z curve, Morton Code

$p_x = 0.1010$
 $p_y = 0.0111$
 $p_z = 0.1100$



```
// Calculates a 30-bit Morton code for the  
// given 3D point located within the unit cube [0,1].  
unsigned int morton3D(float x, float y, float z)
```

```
{
```

```
    x = min(max(x * 1024.0f, 0.0f), 1023.0f);
```

```
    y = min(max(y * 1024.0f, 0.0f), 1023.0f);
```

```
    z = min(max(z * 1024.0f, 0.0f), 1023.0f);
```

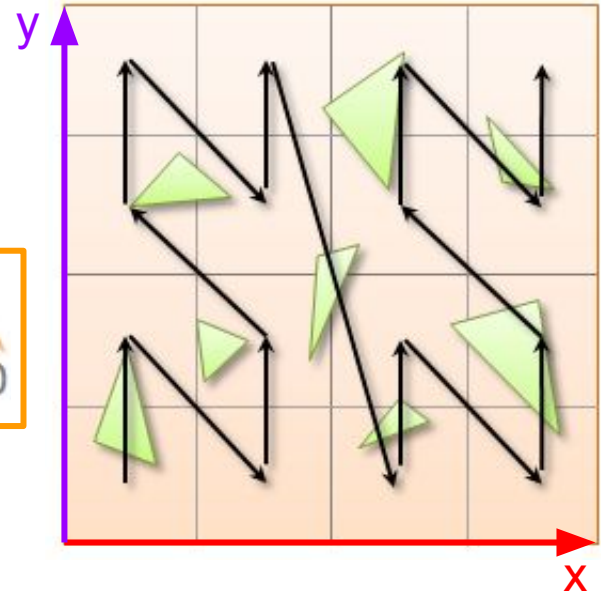
Z curve, Morton Code

$p_x = 0.1010$

$p_y = 0.0111$

$p_z = 0.1100$

$p_x = 0.$ 1 0 1 0
 $p_y = 0.$ 0 1 1 1
 $p_z = 0.$ 1 1 0 0



```
// Calculates a 30-bit Morton code for the  
// given 3D point located within the unit cube [0,1].  
unsigned int morton3D(float x, float y, float z)
```

```
{
```

```
    x = min(max(x * 1024.0f, 0.0f), 1023.0f);
```

```
    y = min(max(y * 1024.0f, 0.0f), 1023.0f);
```

```
    z = min(max(z * 1024.0f, 0.0f), 1023.0f);
```

```
    unsigned int xx = expandBits((unsigned int)x);
```

```
    unsigned int yy = expandBits((unsigned int)y);
```

```
    unsigned int zz = expandBits((unsigned int)z);
```

Z curve, Morton Code

```
// Expands a 10-bit integer into 30 bits  
// by inserting 2 zeros after each bit.
```

```
unsigned int expandBits(unsigned int v)
```

```
{
```

```
    v = (v * 0x00010001u) & 0xFF0000FFu;
```

```
    v = (v * 0x00000101u) & 0x0F00F00Fu;
```

```
    v = (v * 0x00000011u) & 0xC30C30C3u;
```

```
    v = (v * 0x00000005u) & 0x49249249u;
```

```
    return v;
```

```
}
```

```
// Calculates a 30-bit Morton code for the
```

```
// given 3D point located within the unit cube [0,1].
```

```
unsigned int morton3D(float x, float y, float z)
```

```
{
```

```
    x = min(max(x * 1024.0f, 0.0f), 1023.0f);
```

```
    y = min(max(y * 1024.0f, 0.0f), 1023.0f);
```

```
    z = min(max(z * 1024.0f, 0.0f), 1023.0f);
```

```
    unsigned int xx = expandBits((unsigned int)x);
```

```
    unsigned int yy = expandBits((unsigned int)y);
```

```
    unsigned int zz = expandBits((unsigned int)z);
```

$p_x = 0.1010$

$p_y = 0.0111$

$p_z = 0.1100$

$p_x = 0.$

1

0

1

0

1

0

1

0

$p_y = 0.$

0

1

1

1

1

0

1

0

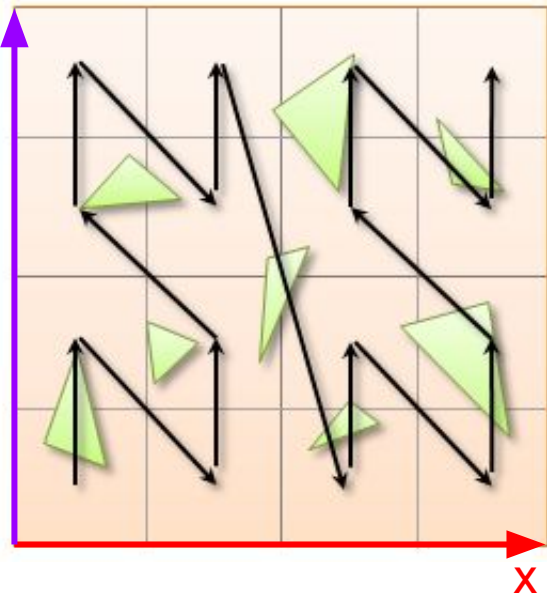
$p_z = 0.$

1

1

0

0



Z curve, Morton Code

```
// Expands a 10-bit integer into 30 bits  
// by inserting 2 zeros after each bit.
```

```
unsigned int expandBits(unsigned int v)
```

```
{
```

```
    v = (v * 0x00010001u) & 0xFF0000FFu;
```

```
    v = (v * 0x00000101u) & 0x0F00F0Fu;
```

```
    v = (v * 0x00000011u) & 0xC30C30Cu;
```

```
    v = (v * 0x00000005u) & 0x49249249u;
```

```
    return v;
```

```
}
```

```
// Calculates a 30-bit Morton code for the
```

```
// given 3D point located within the unit cube [0,1].
```

```
unsigned int morton3D(float x, float y, float z)
```

```
{
```

```
    x = min(max(x * 1024.0f, 0.0f), 1023.0f);
```

```
    y = min(max(y * 1024.0f, 0.0f), 1023.0f);
```

```
    z = min(max(z * 1024.0f, 0.0f), 1023.0f);
```

```
    unsigned int xx = expandBits((unsigned int)x);
```

```
    unsigned int yy = expandBits((unsigned int)y);
```

```
    unsigned int zz = expandBits((unsigned int)z);
```

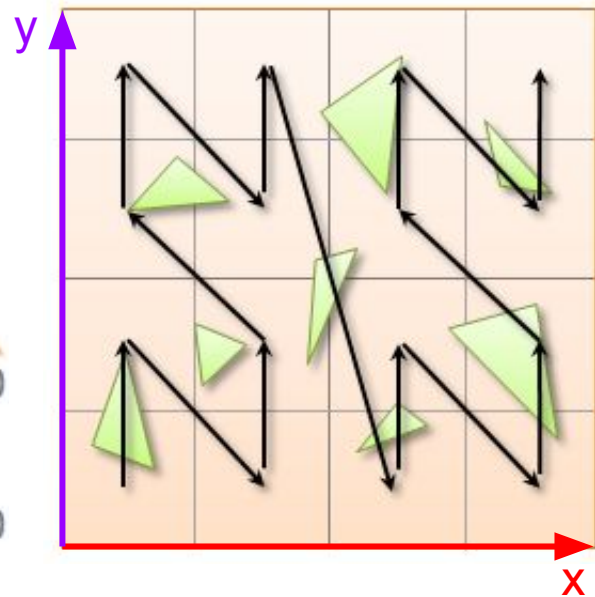
```
    return xx * 4 + yy * 2 + zz;
```

```
}
```

$p_x = 0.1010$
 $p_y = 0.0111$
 $p_z = 0.1100$

$p_x = 0.$ 1 0 1 0
 $p_y = 0.$ 0 1 1 1
 $p_z = 0.$ 1 1 0 0

code = 1 0 1 0 1 1 1 1 0 0 1 0



Z curve, Morton Code

// Expands a 10-bit integer into 30 bits
// by inserting 2 zeros after each bit.

```
unsigned int expandBits(unsigned int v)
```

```
{
```

```
    v = (v * 0x00010001u) & 0xFF0000FFu;
```

```
    v = (v * 0x00000101u) & 0xF00F00Fu;
```

```
    v = (v * 0x00000011u) & 0xC30C30C3u;
```

```
    v = (v * 0x00000005u) & 0x49249249u;
```

```
    return v;
```

```
}
```

// Calculates a 30-bit Morton code for the

// given 3D point located within the unit cube [0,1].

```
unsigned int morton3D(float x, float y, float z)
```

```
{
```

```
    x = min(max(x * 1024.0f, 0.0f), 1023.0f);
```

```
    y = min(max(y * 1024.0f, 0.0f), 1023.0f);
```

```
    z = min(max(z * 1024.0f, 0.0f), 1023.0f);
```

```
    unsigned int xx = expandBits((unsigned int)x);
```

```
    unsigned int yy = expandBits((unsigned int)y);
```

```
    unsigned int zz = expandBits((unsigned int)z);
```

```
    return xx * 4 + yy * 2 + zz;
```

```
}
```

$p_x = 0.1010$

$p_y = 0.0111$

$p_z = 0.1100$

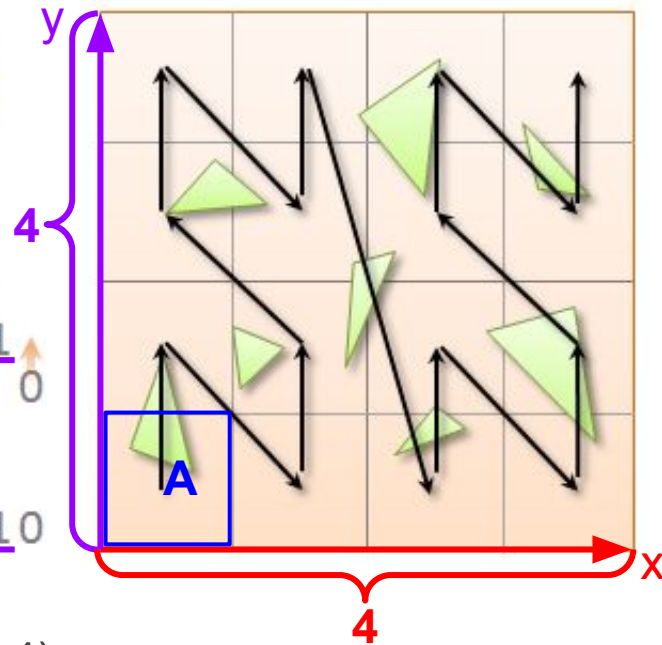
$p_x = 0.$

$p_y = 0.$

$p_z = 0.$

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

code = 101011110010



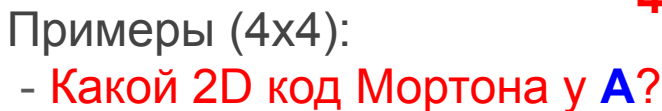
Примеры (4x4):

- Какой 2D код Мортонa у A?


```
// Expands a 10-bit integer into 30 bits
// by inserting 2 zeros after each bit.
unsigned int expandBits(unsigned int v)
```

```
{
    v = (v * 0x00010001u) & 0xFF0000FFu;
    v = (v * 0x00000101u) & 0x0F00F00Fu;
    v = (v * 0x00000011u) & 0xC30C30C3u;
    v = (v * 0x00000005u) & 0x49249249u;
    return v;
}
```

```
{
    x = min(max(x * 1024.0f, 0.0f), 1023.0f);
    y = min(max(y * 1024.0f, 0.0f), 1023.0f);
    z = min(max(z * 1024.0f, 0.0f), 1023.0f);
    unsigned int xx = expandBits((unsigned int)x);
    unsigned int yy = expandBits((unsigned int)y);
    unsigned int zz = expandBits((unsigned int)z);
    return xx * 4 + yy * 2 + zz;
}
```



Z curve, Morton Code

// Expands a 10-bit integer into 30 bits
// by inserting 2 zeros after each bit.

```
unsigned int expandBits(unsigned int v)
```

```
{
```

```
    v = (v * 0x00010001u) & 0xFF0000FFu;
```

```
    v = (v * 0x00000101u) & 0x0F00F0Fu;
```

```
    v = (v * 0x00000011u) & 0xC30C30Cu;
```

```
    v = (v * 0x00000005u) & 0x49249249u;
```

```
    return v;
```

```
}
```

// Calculates a 30-bit Morton code for the
// given 3D point located within the unit cube [0,1].

```
unsigned int morton3D(float x, float y, float z)
```

```
{
```

```
    x = min(max(x * 1024.0f, 0.0f), 1023.0f);
```

```
    y = min(max(y * 1024.0f, 0.0f), 1023.0f);
```

```
    z = min(max(z * 1024.0f, 0.0f), 1023.0f);
```

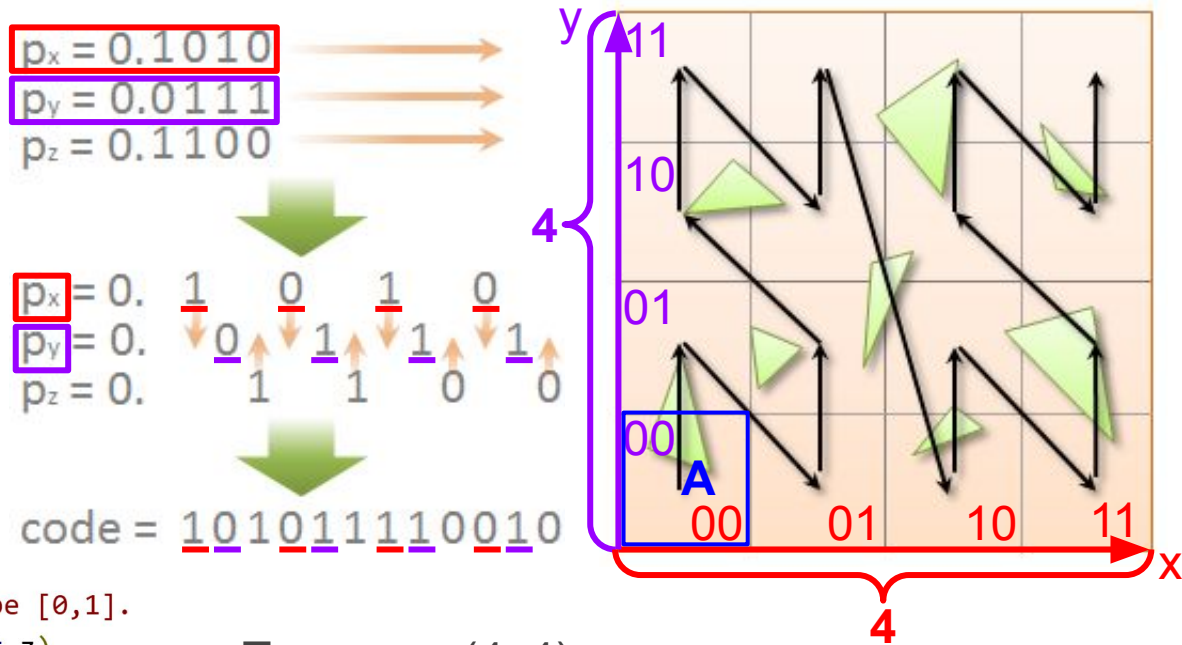
```
    unsigned int xx = expandBits((unsigned int)x);
```

```
    unsigned int yy = expandBits((unsigned int)y);
```

```
    unsigned int zz = expandBits((unsigned int)z);
```

```
    return xx * 4 + yy * 2 + zz;
```

```
}
```



Примеры (4x4):

- **A** = swizzling(**x**=00, **y**=00) = ????

Z curve, Morton Code

// Expands a 10-bit integer into 30 bits
// by inserting 2 zeros after each bit.

```
unsigned int expandBits(unsigned int v)
```

```
{
```

```
    v = (v * 0x00010001u) & 0xFF0000FFu;
```

```
    v = (v * 0x00000101u) & 0x0F00F0Fu;
```

```
    v = (v * 0x00000011u) & 0xC30C30Cu;
```

```
    v = (v * 0x00000005u) & 0x49249249u;
```

```
    return v;
```

```
}
```

// Calculates a 30-bit Morton code for the

// given 3D point located within the unit cube [0,1].

```
unsigned int morton3D(float x, float y, float z)
```

```
{
```

```
    x = min(max(x * 1024.0f, 0.0f), 1023.0f);
```

```
    y = min(max(y * 1024.0f, 0.0f), 1023.0f);
```

```
    z = min(max(z * 1024.0f, 0.0f), 1023.0f);
```

```
    unsigned int xx = expandBits((unsigned int)x);
```

```
    unsigned int yy = expandBits((unsigned int)y);
```

```
    unsigned int zz = expandBits((unsigned int)z);
```

```
    return xx * 4 + yy * 2 + zz;
```

```
}
```

$p_x = 0.1010$

$p_y = 0.0111$

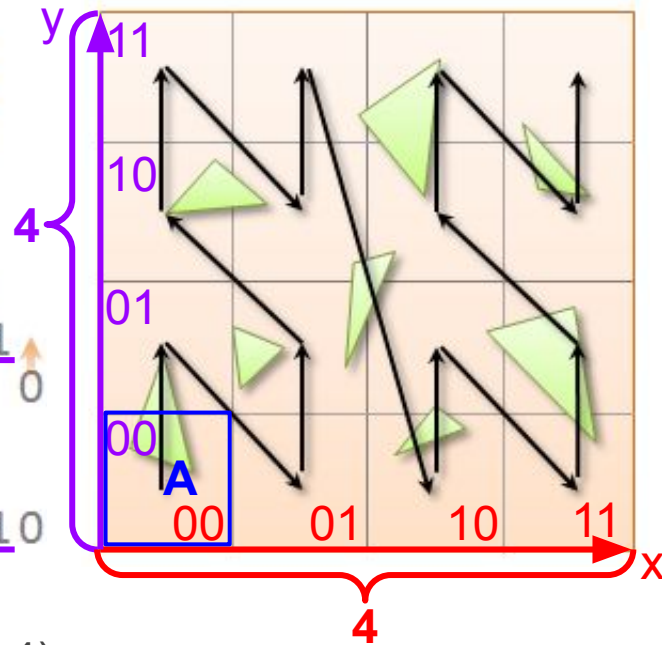
$p_z = 0.1100$

$p_x = 0.$

$p_y = 0.$

$p_z = 0.$

code = 101011110010



Примеры (4x4):

- **A** = swizzling(**x**=00, **y**=00) = 0000

Z curve, Morton Code

```
// Expands a 10-bit integer into 30 bits  
// by inserting 2 zeros after each bit.
```

```
unsigned int expandBits(unsigned int v)
```

```
{
```

```
    v = (v * 0x00010001u) & 0xFF0000FFu;
```

```
    v = (v * 0x00000101u) & 0xF00F00Fu;
```

```
    v = (v * 0x00000011u) & 0xC30C30Cu;
```

```
    v = (v * 0x00000005u) & 0x49249249u;
```

```
    return v;
```

```
}
```

```
// Calculates a 30-bit Morton code for the
```

```
// given 3D point located within the unit cube [0,1].
```

```
unsigned int morton3D(float x, float y, float z)
```

```
{
```

```
    x = min(max(x * 1024.0f, 0.0f), 1023.0f);
```

```
    y = min(max(y * 1024.0f, 0.0f), 1023.0f);
```

```
    z = min(max(z * 1024.0f, 0.0f), 1023.0f);
```

```
    unsigned int xx = expandBits((unsigned int)x);
```

```
    unsigned int yy = expandBits((unsigned int)y);
```

```
    unsigned int zz = expandBits((unsigned int)z);
```

```
    return xx * 4 + yy * 2 + zz;
```

```
}
```

$p_x = 0.1010$

$p_y = 0.0111$

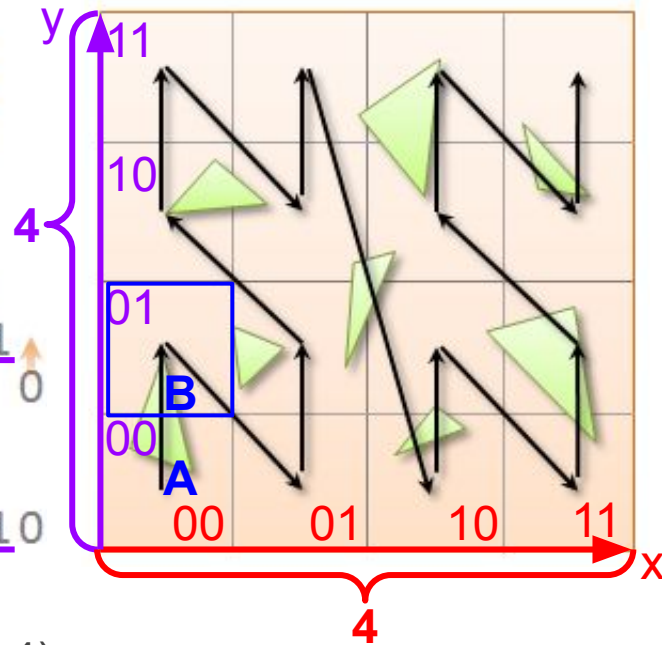
$p_z = 0.1100$

$p_x = 0.$

$p_y = 0.$

$p_z = 0.$

code = 101011110010



Примеры (4x4):

- **A** = swizzling(**x**=00, **y**=00) = 0000
- Какой 2D код Мортонa у **B**?

Z curve, Morton Code

```
// Expands a 10-bit integer into 30 bits  
// by inserting 2 zeros after each bit.
```

```
unsigned int expandBits(unsigned int v)
```

```
{
```

```
    v = (v * 0x00010001u) & 0xFF0000FFu;
```

```
    v = (v * 0x00000101u) & 0xF00F00Fu;
```

```
    v = (v * 0x00000011u) & 0xC30C30C3u;
```

```
    v = (v * 0x00000005u) & 0x49249249u;
```

```
    return v;
```

```
}
```

```
// Calculates a 30-bit Morton code for the
```

```
// given 3D point located within the unit cube [0,1].
```

```
unsigned int morton3D(float x, float y, float z)
```

```
{
```

```
    x = min(max(x * 1024.0f, 0.0f), 1023.0f);
```

```
    y = min(max(y * 1024.0f, 0.0f), 1023.0f);
```

```
    z = min(max(z * 1024.0f, 0.0f), 1023.0f);
```

```
    unsigned int xx = expandBits((unsigned int)x);
```

```
    unsigned int yy = expandBits((unsigned int)y);
```

```
    unsigned int zz = expandBits((unsigned int)z);
```

```
    return xx * 4 + yy * 2 + zz;
```

```
}
```

$p_x = 0.1010$

$p_y = 0.0111$

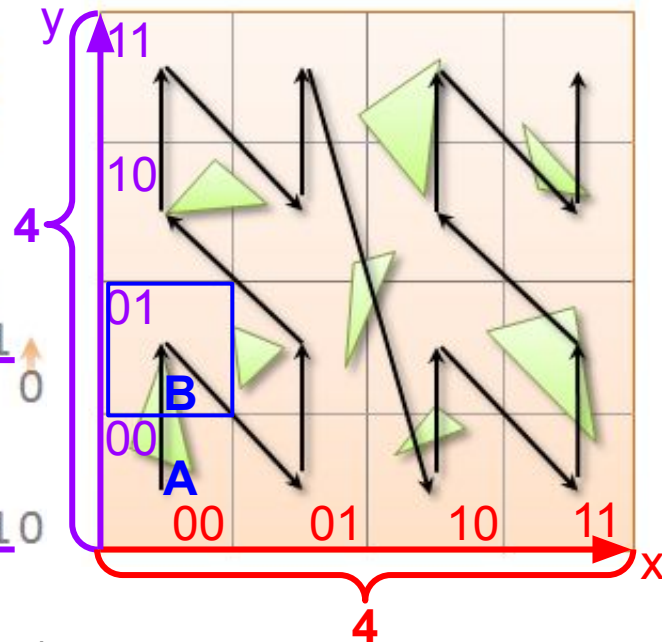
$p_z = 0.1100$

$p_x = 0.$

$p_y = 0.$

$p_z = 0.$

code = 101011110010



Примеры (4x4):

- **A** = swizzling(**x**=00, **y**=00) = 0000
- **B** = swizzling(**x**=00, **y**=01) = ????

Z curve, Morton Code

```
// Expands a 10-bit integer into 30 bits  
// by inserting 2 zeros after each bit.
```

```
unsigned int expandBits(unsigned int v)
```

```
{
```

```
    v = (v * 0x00010001u) & 0xFF0000FFu;
```

```
    v = (v * 0x00000101u) & 0x0F00F0Fu;
```

```
    v = (v * 0x00000011u) & 0xC30C30Cu;
```

```
    v = (v * 0x00000005u) & 0x49249249u;
```

```
    return v;
```

```
}
```

```
// Calculates a 30-bit Morton code for the
```

```
// given 3D point located within the unit cube [0,1].
```

```
unsigned int morton3D(float x, float y, float z)
```

```
{
```

```
    x = min(max(x * 1024.0f, 0.0f), 1023.0f);
```

```
    y = min(max(y * 1024.0f, 0.0f), 1023.0f);
```

```
    z = min(max(z * 1024.0f, 0.0f), 1023.0f);
```

```
    unsigned int xx = expandBits((unsigned int)x);
```

```
    unsigned int yy = expandBits((unsigned int)y);
```

```
    unsigned int zz = expandBits((unsigned int)z);
```

```
    return xx * 4 + yy * 2 + zz;
```

```
}
```

$p_x = 0.1010$

$p_y = 0.0111$

$p_z = 0.1100$

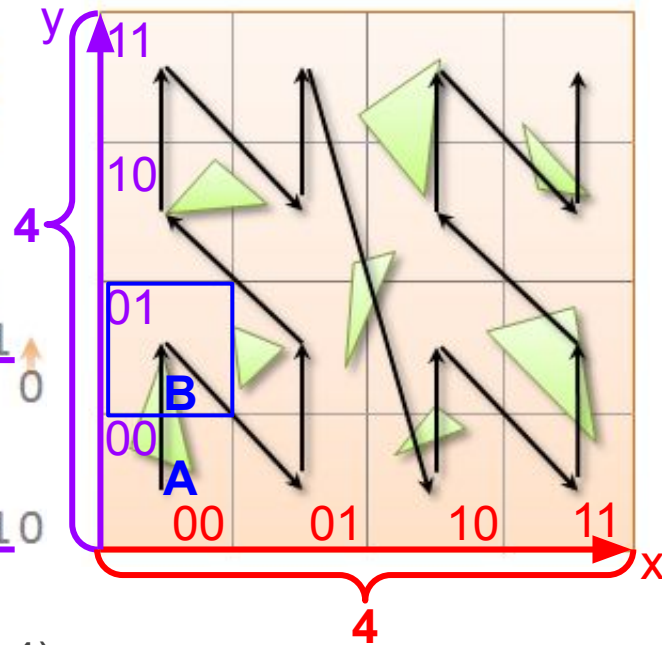
$p_x = 0.$

$p_y = 0.$

$p_z = 0.$

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

code = 101011110010



Примеры (4x4):

- **A** = swizzling(**x**=00, **y**=00) = 0000

- **B** = swizzling(**x**=00, **y**=01) = 0001

Z curve, Morton Code

// Expands a 10-bit integer into 30 bits
// by inserting 2 zeros after each bit.

```
unsigned int expandBits(unsigned int v)
```

```
{
```

```
    v = (v * 0x00010001u) & 0xFF0000FFu;
```

```
    v = (v * 0x00000101u) & 0xF00F00Fu;
```

```
    v = (v * 0x00000011u) & 0xC30C30Cu;
```

```
    v = (v * 0x00000005u) & 0x49249249u;
```

```
    return v;
```

```
}
```

// Calculates a 30-bit Morton code for the
// given 3D point located within the unit cube [0,1].

```
unsigned int morton3D(float x, float y, float z)
```

```
{
```

```
    x = min(max(x * 1024.0f, 0.0f), 1023.0f);
```

```
    y = min(max(y * 1024.0f, 0.0f), 1023.0f);
```

```
    z = min(max(z * 1024.0f, 0.0f), 1023.0f);
```

```
    unsigned int xx = expandBits((unsigned int)x);
```

```
    unsigned int yy = expandBits((unsigned int)y);
```

```
    unsigned int zz = expandBits((unsigned int)z);
```

```
    return xx * 4 + yy * 2 + zz;
```

```
}
```

$p_x = 0.1010$

$p_y = 0.0111$

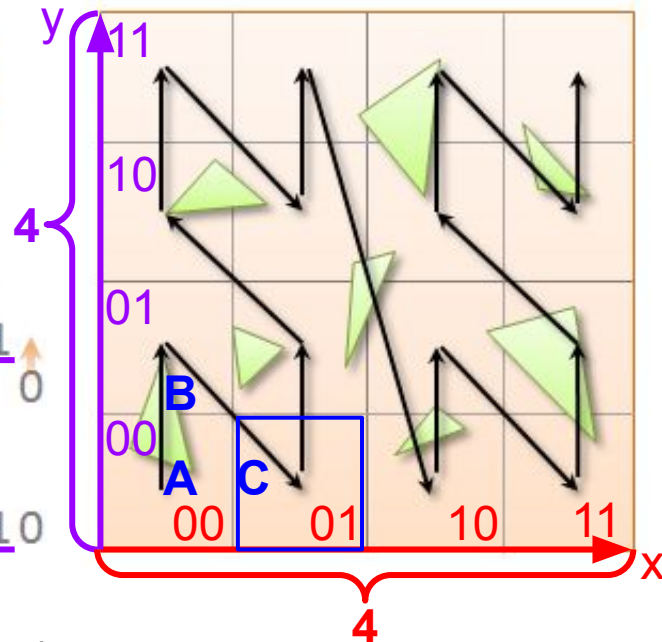
$p_z = 0.1100$

$p_x = 0.$

$p_y = 0.$

$p_z = 0.$

code = 101011110010



Примеры (4x4):

- **A** = swizzling($x=00$, $y=00$) = 0000
- **B** = swizzling($x=00$, $y=01$) = 0001
- **C** = swizzling($x=??$, $y=??$) = ????

```
// Expands a 10-bit integer into 30 bits
// by inserting 2 zeros after each bit.
unsigned int expandBits(unsigned int v)
```

```
{
    v = (v * 0x00010001u) & 0xFF0000FFu;
    v = (v * 0x00000101u) & 0x0F00F00Fu;
    v = (v * 0x00000011u) & 0xC30C30C3u;
    v = (v * 0x00000005u) & 0x49249249u;
    return v;
}
```

```
{
    x = min(max(x * 1024.0f, 0.0f), 1023.0f);
    y = min(max(y * 1024.0f, 0.0f), 1023.0f);
    z = min(max(z * 1024.0f, 0.0f), 1023.0f);
    unsigned int xx = expandBits((unsigned int)x);
    unsigned int yy = expandBits((unsigned int)y);
    unsigned int zz = expandBits((unsigned int)z);
    return xx * 4 + yy * 2 + zz;
}
```



- 56


```
// Expands a 10-bit integer into 30 bits
// by inserting 2 zeros after each bit.
unsigned int expandBits(unsigned int v)
```

```
{
    v = (v * 0x00010001u) & 0xFF0000FFu;
    v = (v * 0x00000101u) & 0x0F00F00Fu;
    v = (v * 0x00000011u) & 0xC30C30C3u;
    v = (v * 0x00000005u) & 0x49249249u;
    return v;
}
```

```
{
    x = min(max(x * 1024.0f, 0.0f), 1023.0f);
    y = min(max(y * 1024.0f, 0.0f), 1023.0f);
    z = min(max(z * 1024.0f, 0.0f), 1023.0f);
    unsigned int xx = expandBits((unsigned int)x);
    unsigned int yy = expandBits((unsigned int)y);
    unsigned int zz = expandBits((unsigned int)z);
    return xx * 4 + yy * 2 + zz;
}
```



- 57

```
// Expands a 10-bit integer into 30 bits
// by inserting 2 zeros after each bit.
unsigned int expandBits(unsigned int v)
```

```
{
    v = (v * 0x00010001u) & 0xFF0000FFu;
    v = (v * 0x00000101u) & 0x0F00F00Fu;
    v = (v * 0x00000011u) & 0xC30C30C3u;
    v = (v * 0x00000005u) & 0x49249249u;
    return v;
}
```

```
{
    x = min(max(x * 1024.0f, 0.0f), 1023.0f);
    y = min(max(y * 1024.0f, 0.0f), 1023.0f);
    z = min(max(z * 1024.0f, 0.0f), 1023.0f);
    unsigned int xx = expandBits((unsigned int)x);
    unsigned int yy = expandBits((unsigned int)y);
    unsigned int zz = expandBits((unsigned int)z);
    return xx * 4 + yy * 2 + zz;
}
```



- 58

Z curve, Morton Code

// Expands a 10-bit integer into 30 bits
// by inserting 2 zeros after each bit.

```
unsigned int expandBits(unsigned int v)
```

```
{
```

```
    v = (v * 0x00010001u) & 0xFF0000FFu;
```

```
    v = (v * 0x00000101u) & 0x0F00F0Fu;
```

```
    v = (v * 0x00000011u) & 0xC30C30Cu;
```

```
    v = (v * 0x00000005u) & 0x49249249u;
```

```
    return v;
```

```
}
```

// Calculates a 30-bit Morton code for the

// given 3D point located within the unit cube [0,1].

```
unsigned int morton3D(float x, float y, float z)
```

```
{
```

```
    x = min(max(x * 1024.0f, 0.0f), 1023.0f);
```

```
    y = min(max(y * 1024.0f, 0.0f), 1023.0f);
```

```
    z = min(max(z * 1024.0f, 0.0f), 1023.0f);
```

```
    unsigned int xx = expandBits((unsigned int)x);
```

```
    unsigned int yy = expandBits((unsigned int)y);
```

```
    unsigned int zz = expandBits((unsigned int)z);
```

```
    return xx * 4 + yy * 2 + zz;
```

```
}
```

$p_x = 0.1010$

$p_y = 0.0111$

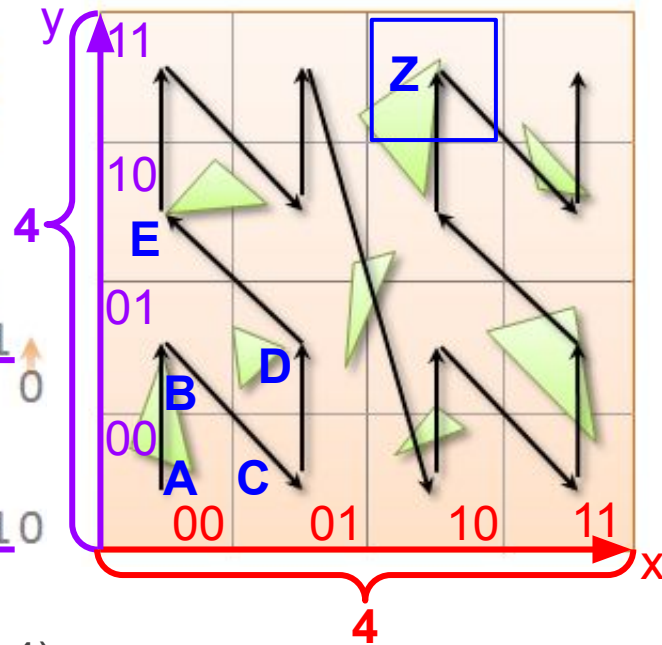
$p_z = 0.1100$

$p_x = 0.$

$p_y = 0.$

$p_z = 0.$

code = 101011110010



Примеры (4x4):

- **A** = swizzling(**x**=00, **y**=00) = 0000
- **B** = swizzling(**x**=00, **y**=01) = 0001
- **C** = swizzling(**x**=01, **y**=00) = 0010
- **D** = swizzling(**x**=01, **y**=01) = 0011
- **E** = swizzling(**x**=00, **y**=10) = 0100
- **Z** = swizzling(**x**=10, **y**=11) = 1101

Z curve, Morton Code

```
// Expands a 10-bit integer into 30 bits  
// by inserting 2 zeros after each bit.
```

```
unsigned int expandBits(unsigned int v)
```

```
{
```

```
    v = (v * 0x00010001u) & 0xFF0000FFu;
```

```
    v = (v * 0x00000101u) & 0xF00F00Fu;
```

```
    v = (v * 0x00000011u) & 0xC30C30C3u;
```

```
    v = (v * 0x00000005u) & 0x49249249u;
```

```
    return v;
```

```
}
```

```
// Calculates a 30-bit Morton code for the
```

```
// given 3D point located within the unit cube [0,1].
```

```
unsigned int morton3D(float x, float y, float z)
```

```
{
```

```
    x = min(max(x * 1024.0f, 0.0f), 1023.0f);
```

```
    y = min(max(y * 1024.0f, 0.0f), 1023.0f);
```

```
    z = min(max(z * 1024.0f, 0.0f), 1023.0f);
```

```
    unsigned int xx = expandBits((unsigned int)x);
```

```
    unsigned int yy = expandBits((unsigned int)y);
```

```
    unsigned int zz = expandBits((unsigned int)z);
```

```
    return xx * 4 + yy * 2 + zz;
```

```
}
```

$p_x = 0.1010$

$p_y = 0.0111$

$p_z = 0.1100$

$p_x = 0.$

$p_y = 0.$

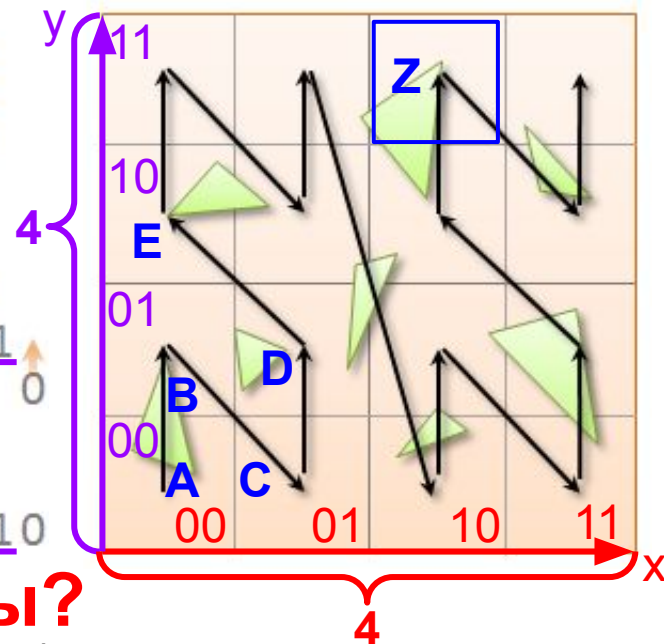
$p_z = 0.$

code = 101011110010

Вопросы?

Примеры (4x4):

- **A** = swizzling(**x**=00, **y**=00) = 0000
- **B** = swizzling(**x**=00, **y**=01) = 0001
- **C** = swizzling(**x**=01, **y**=00) = 0010
- **D** = swizzling(**x**=01, **y**=01) = 0011
- **E** = swizzling(**x**=00, **y**=10) = 0100
- **Z** = swizzling(**x**=10, **y**=11) = 1101



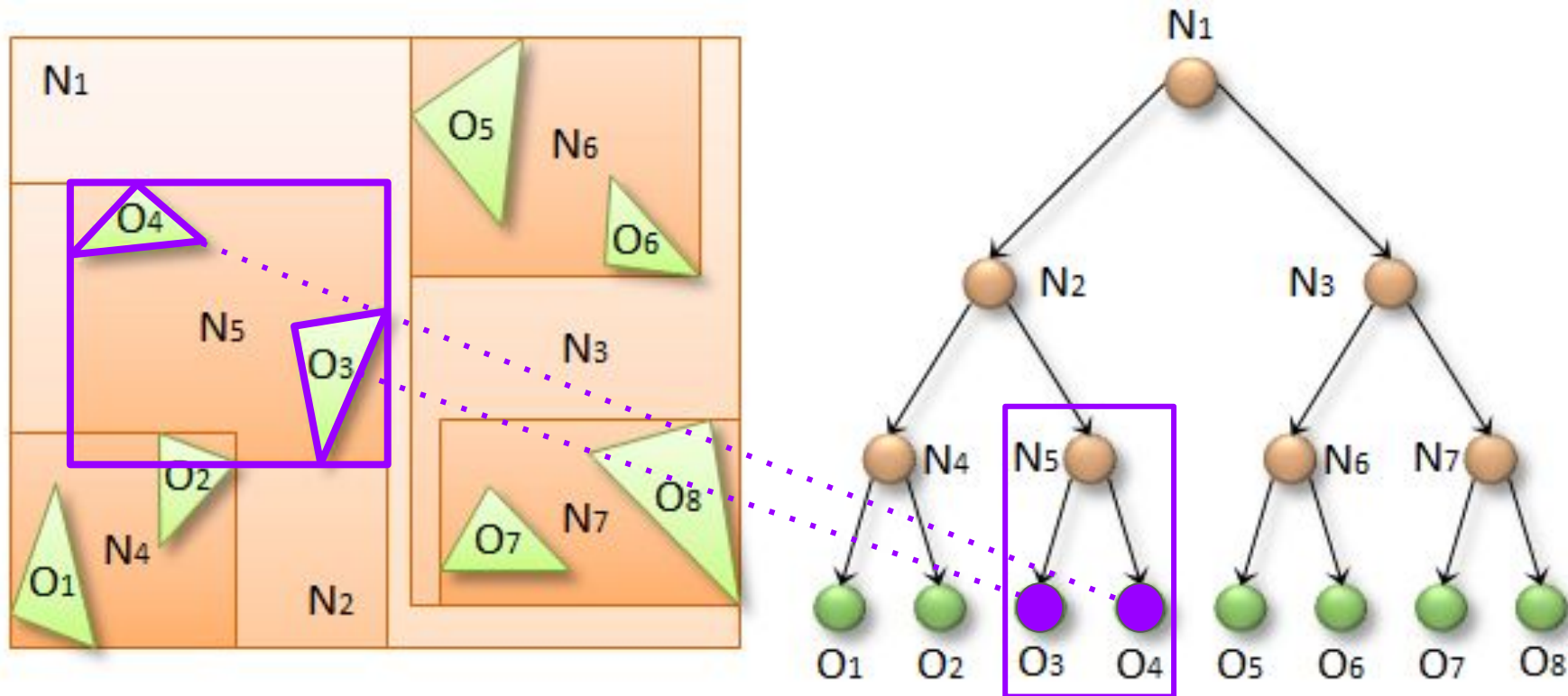
Глава 3: Жесть какая крутая

Maximizing Parallelism in the Construction of BVHs,
Octrees, and k-d Trees

Tero Karras, NVIDIA Research, circa 2012 colorized

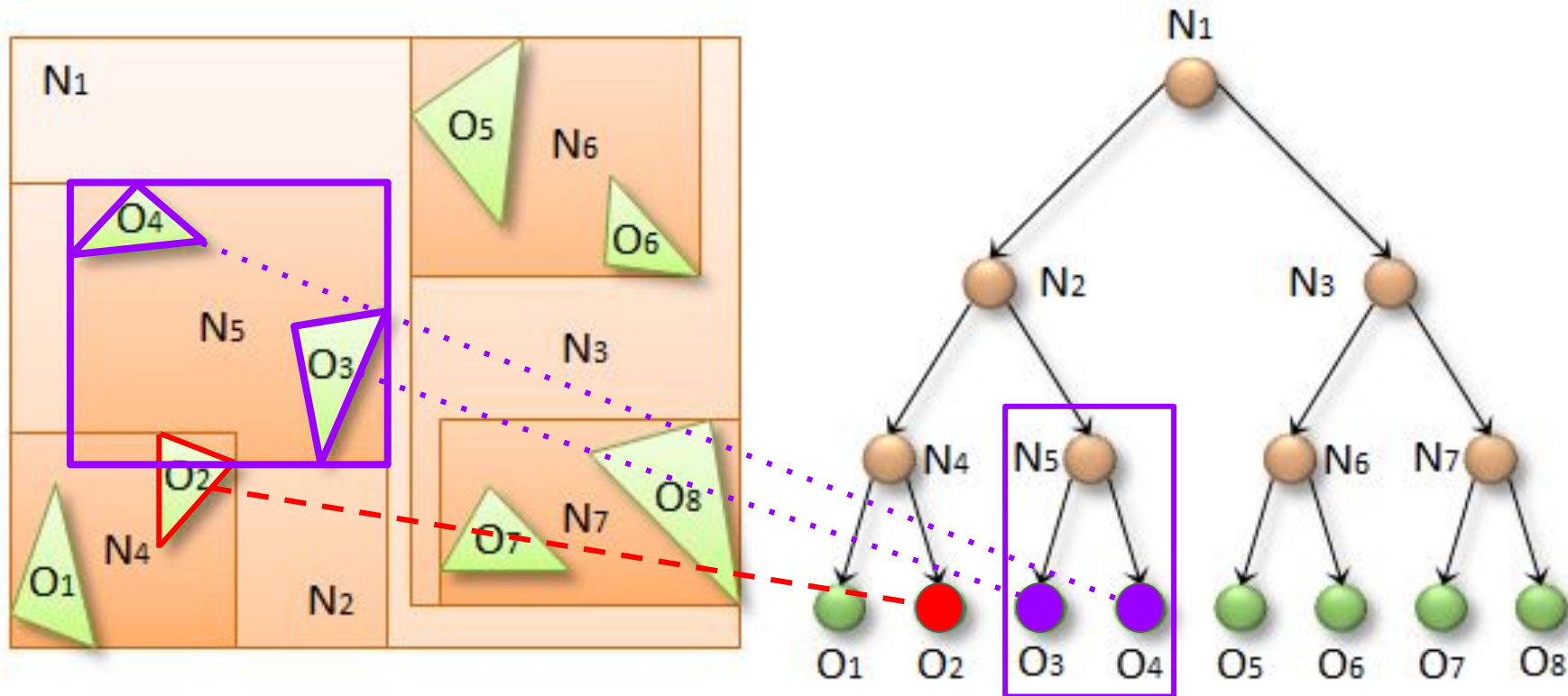


Linear BVH (LBVH)



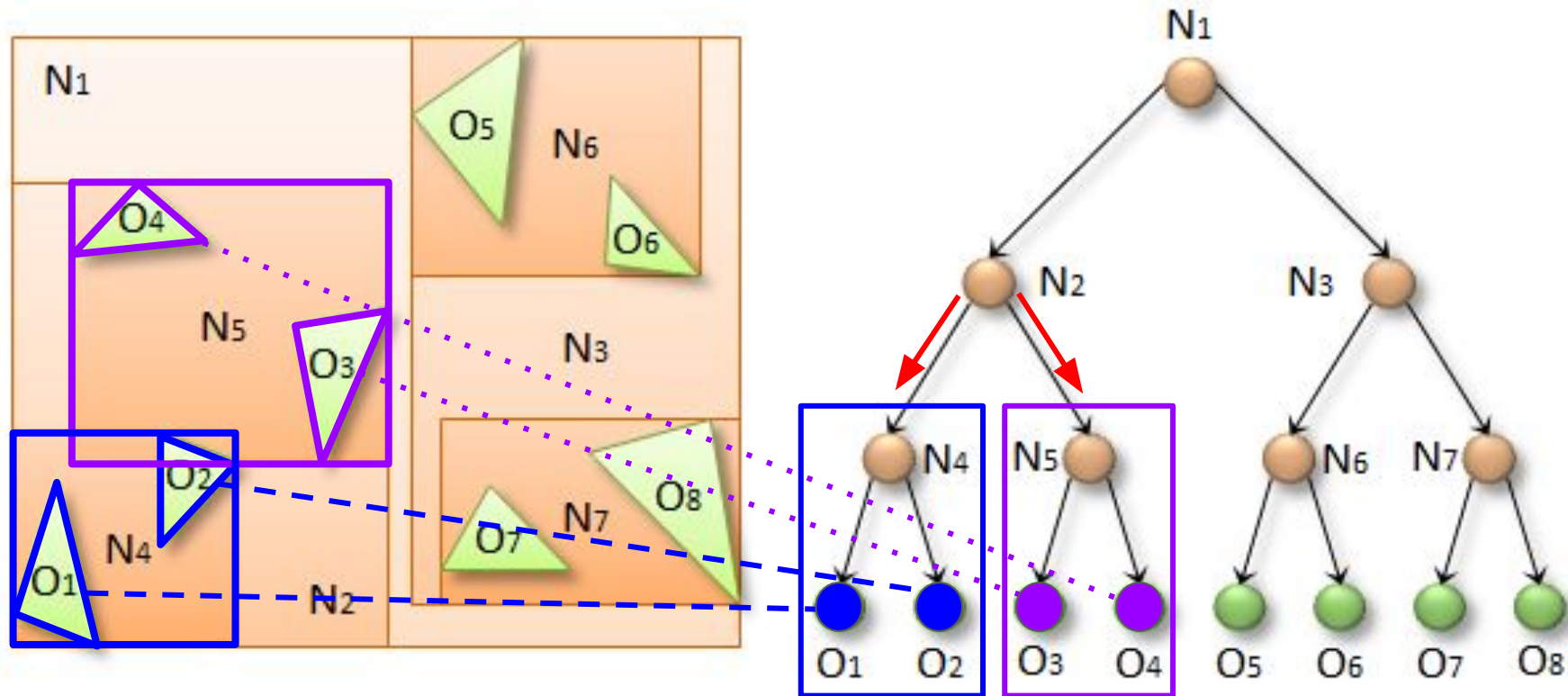
Linear BVH (LBVH)

А почему корректно не
включать O2 в N5 AABB?



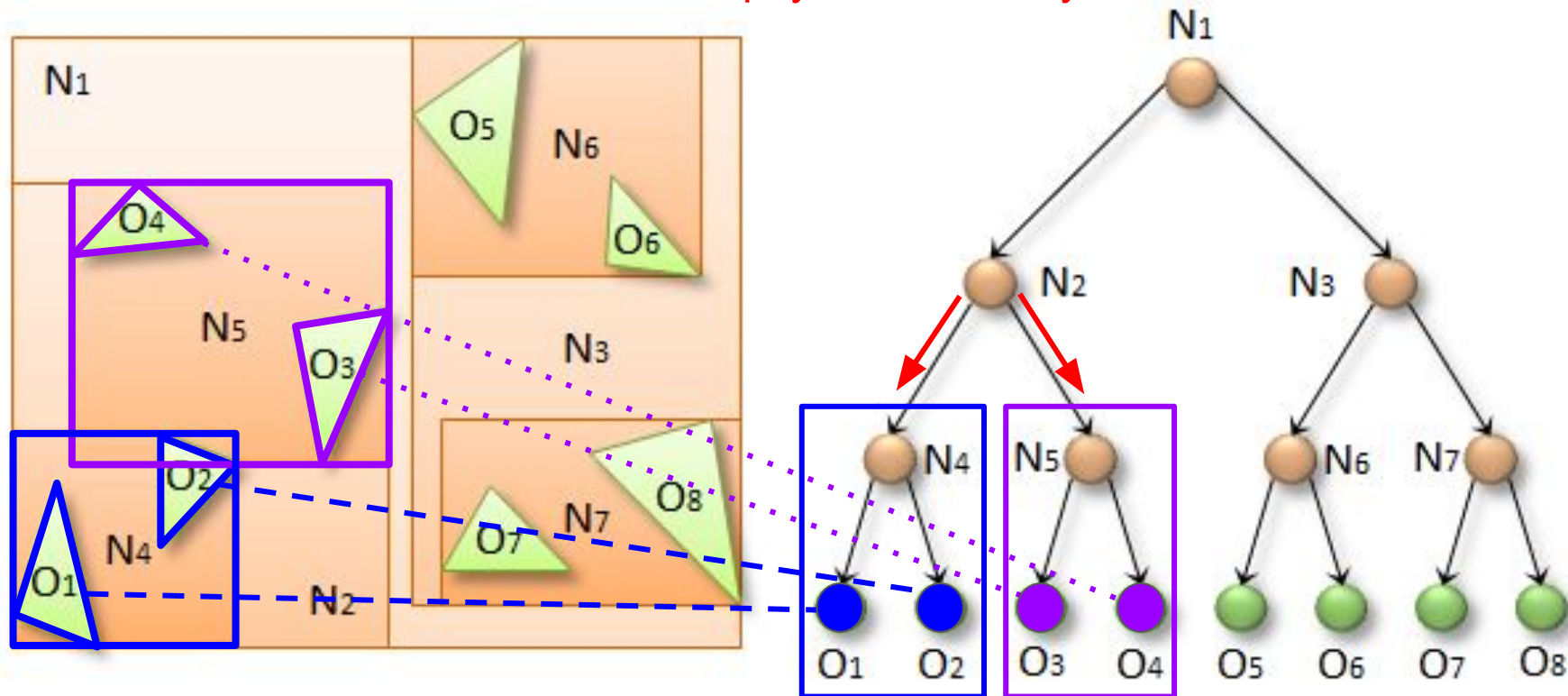
Linear BVH (LBVH)

А почему корректно не
включать O2 в N5 AABB?



Linear BVH (LBVH)

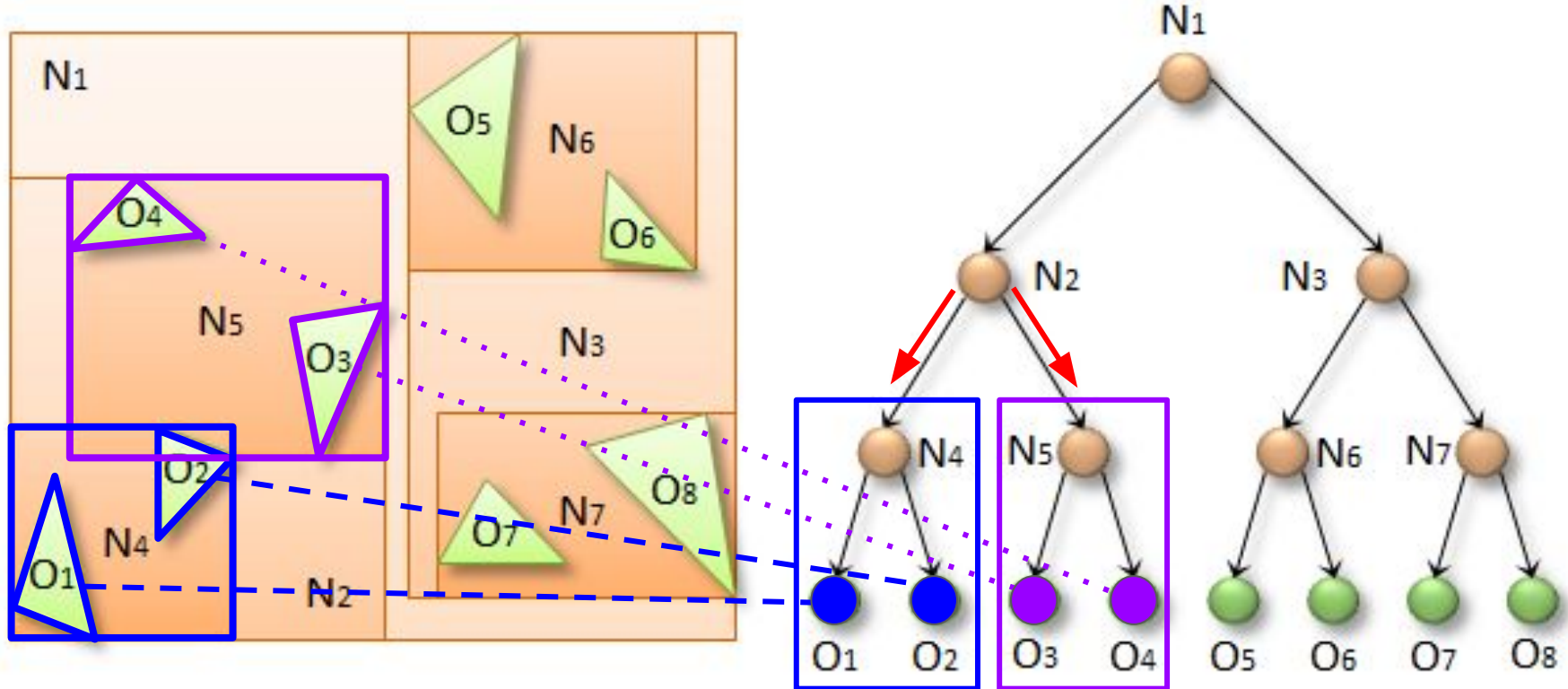
Можем ли мы **случайно** распределить
треугольники по узлам BVH?



Построение BVH не однозначно! Хотим чтобы узлы часто отсекались по **AABB** целиком, т.е. **??? Какая метрика качества BVH?**

Linear BVH (**LBVH**)

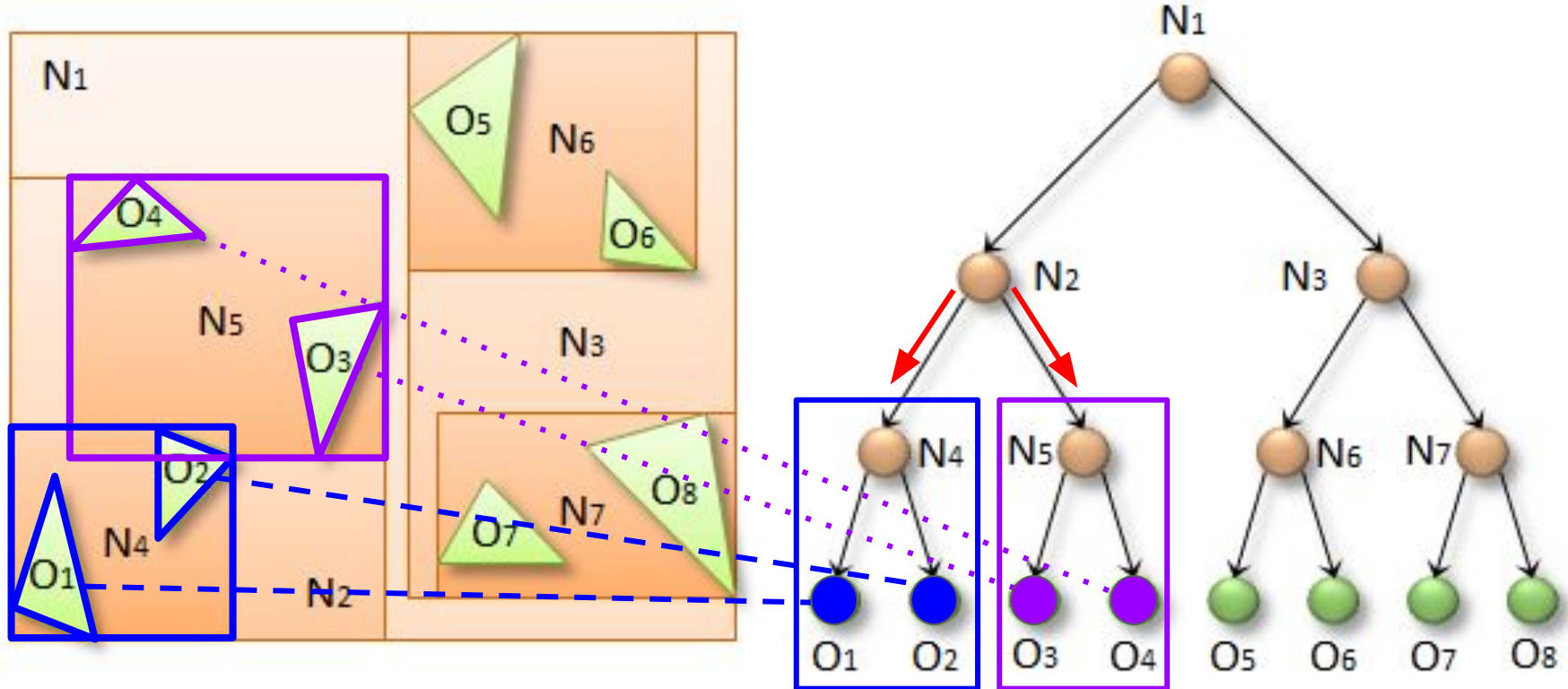
Можем ли мы **случайно** распределить треугольники по узлам BVH?



Построение BVH не однозначно! Хотим чтобы узлы часто отсекались по **AABB целиком**, т.е. хотим точности AABB - например минимизации объема.

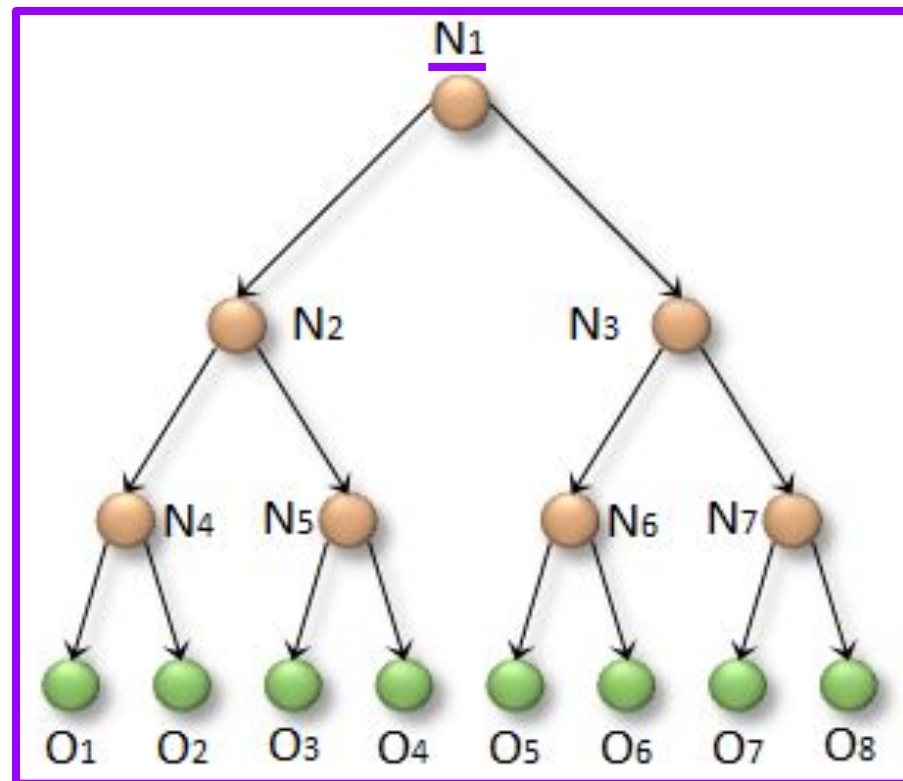
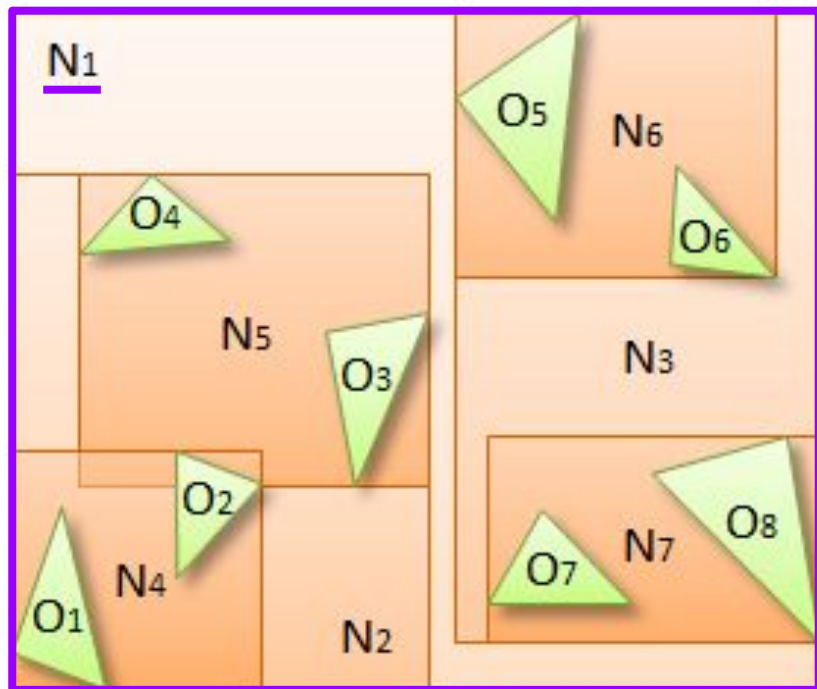
Linear BVH (**LBVH**)

Можем ли мы **случайно** распределить треугольники по узлам BVH?



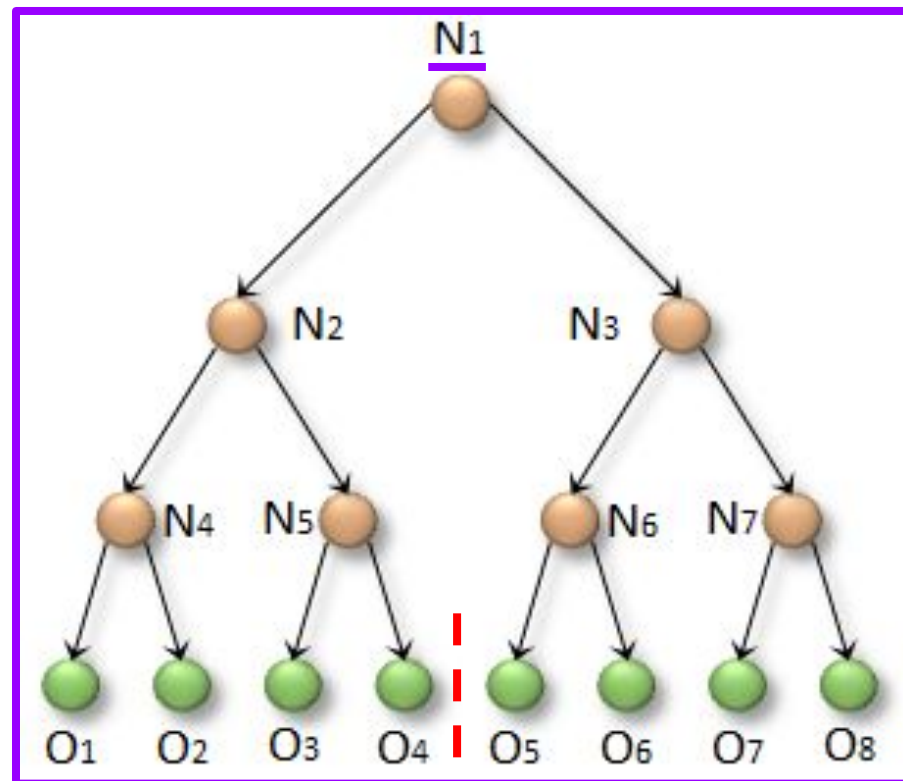
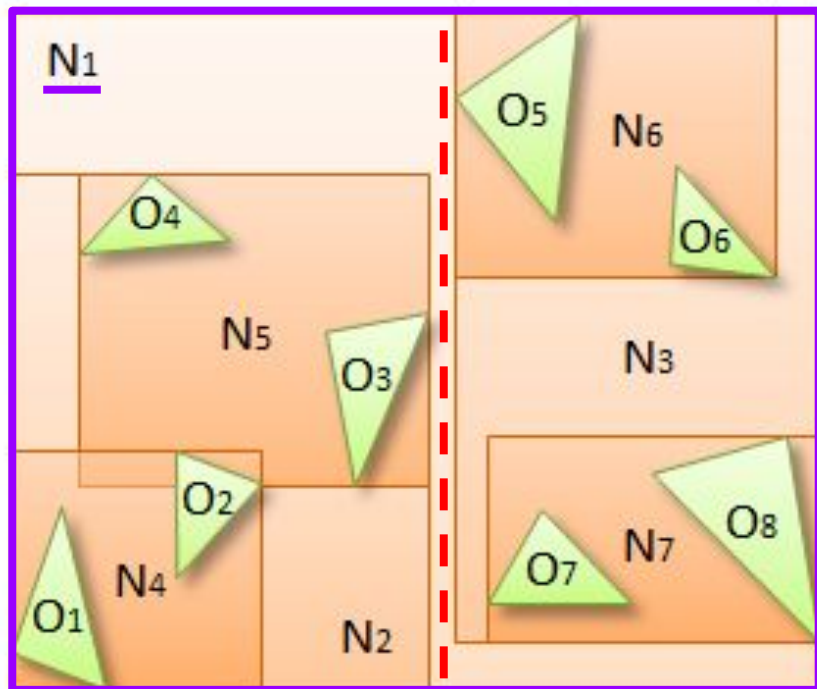
Построение BVH не однозначно! Хотим чтобы узлы часто отсекались по **AABB** **целиком**, т.е. хотим точности AABB - например минимизации объема. **Как добиться?**

Linear BVH (**LBVH**)



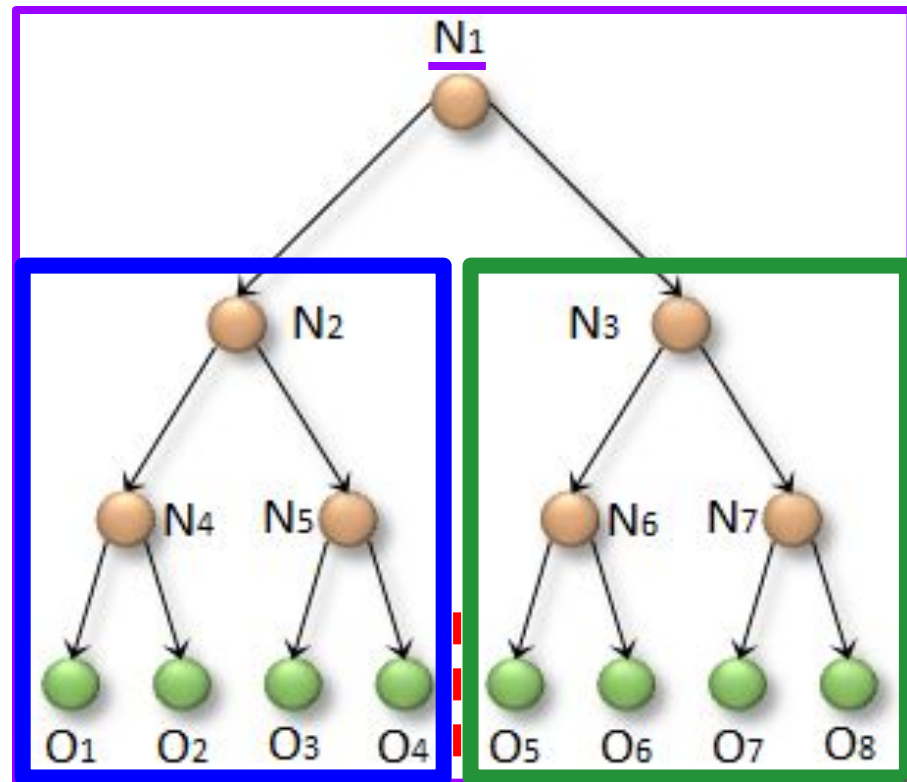
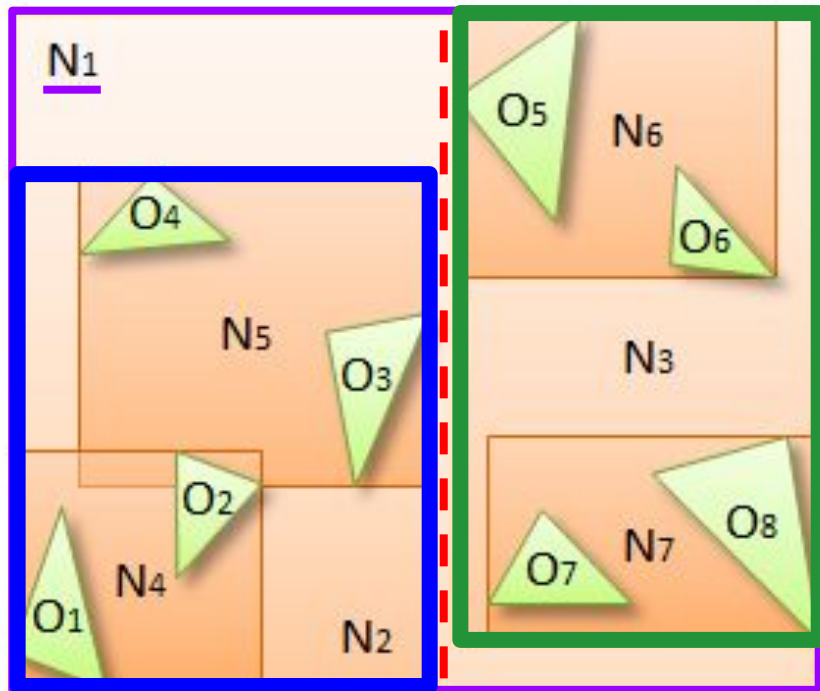
Построение BVH не однозначно! Хотим чтобы узлы часто отсекались по **AABB** **целиком**, т.е. хотим точности AABB - например минимизации объема. **Как добиться?**

Linear BVH (**LBVH**)

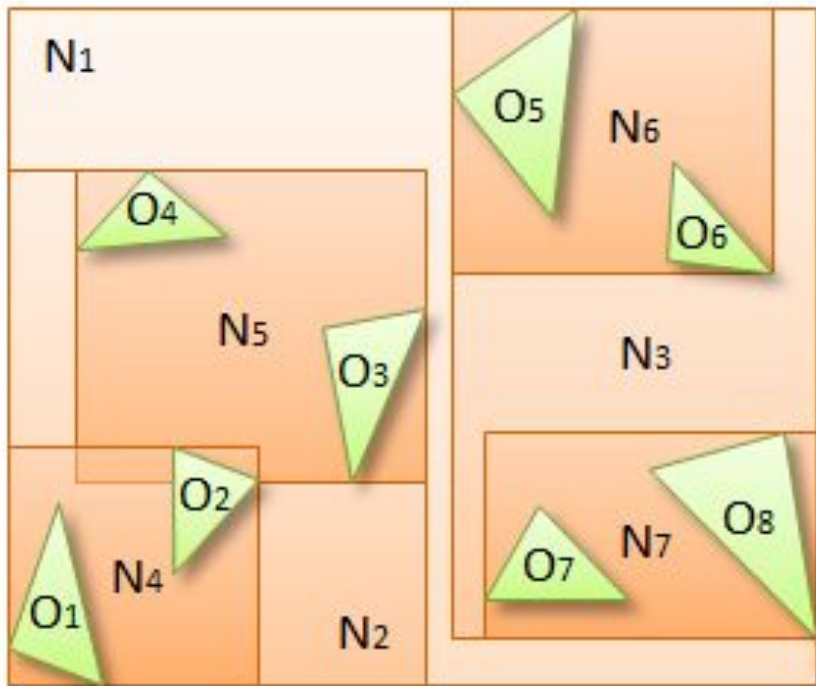


Построение BVH не однозначно! Хотим чтобы узлы часто отсекались по **AABB** **целиком**, т.е. хотим точности AABB - например минимизации объема. **Как добиться?**

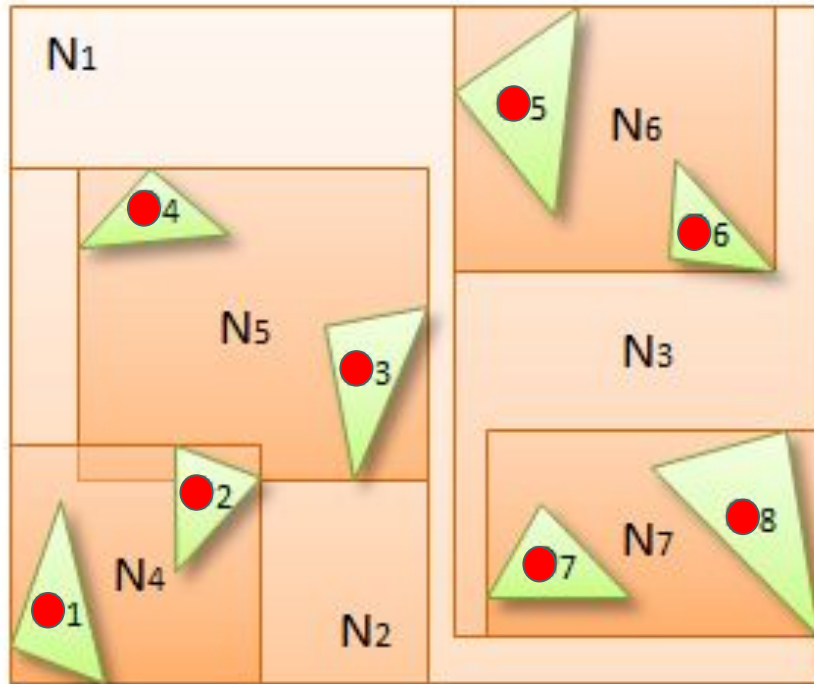
Linear BVH (**LBVH**)



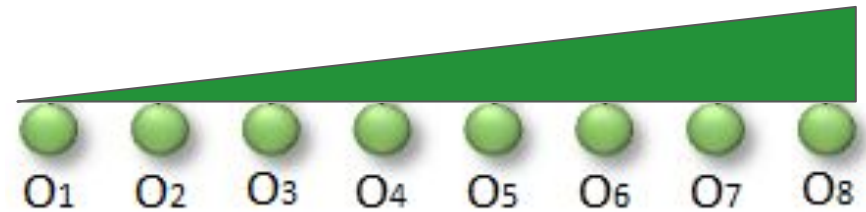
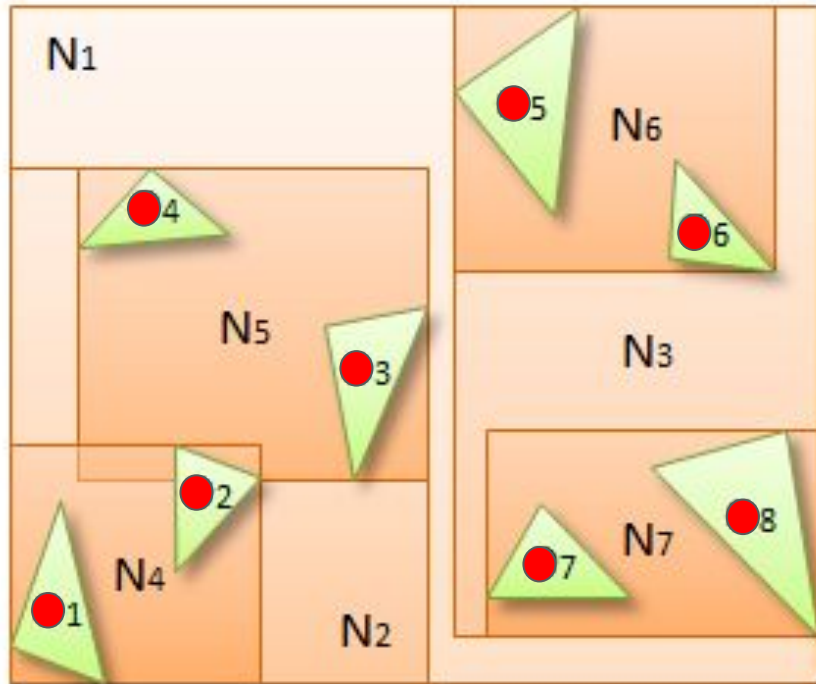
1) Есть много треугольников, каждый со своим AABV, это *листья*



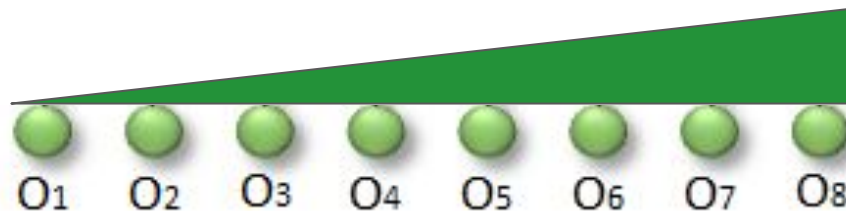
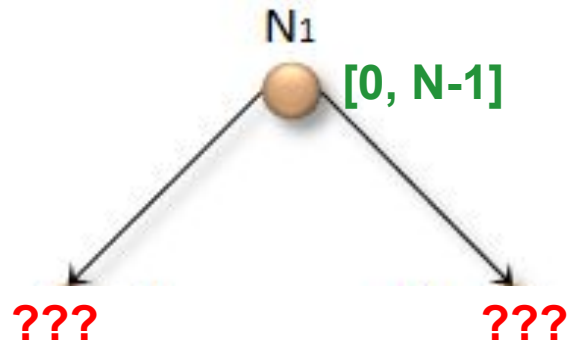
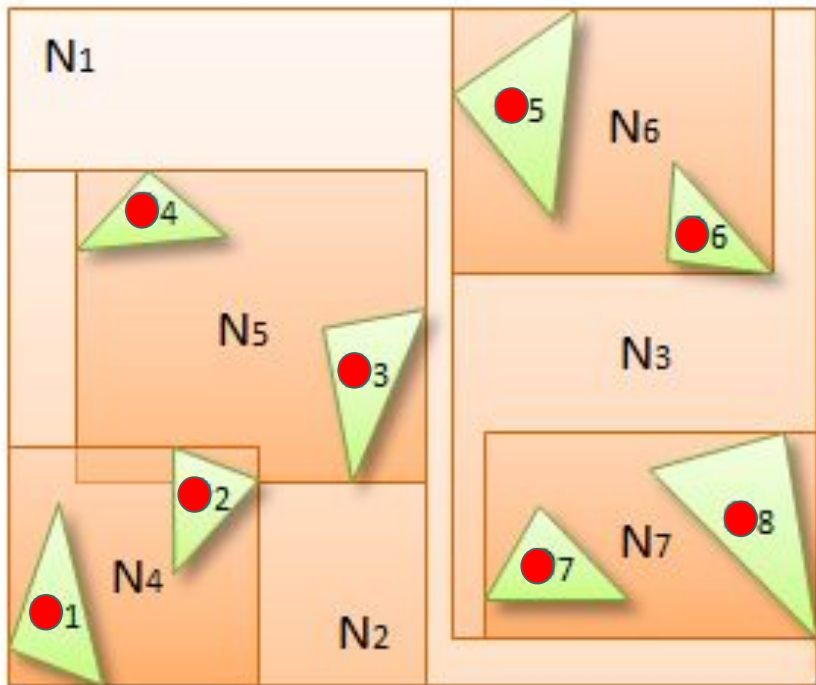
- 1) Есть много треугольников, каждый со своим AABV, это **листья**
- 2) Вычисляем у каждого **листа** код мортона (**по точке-центроиду**)



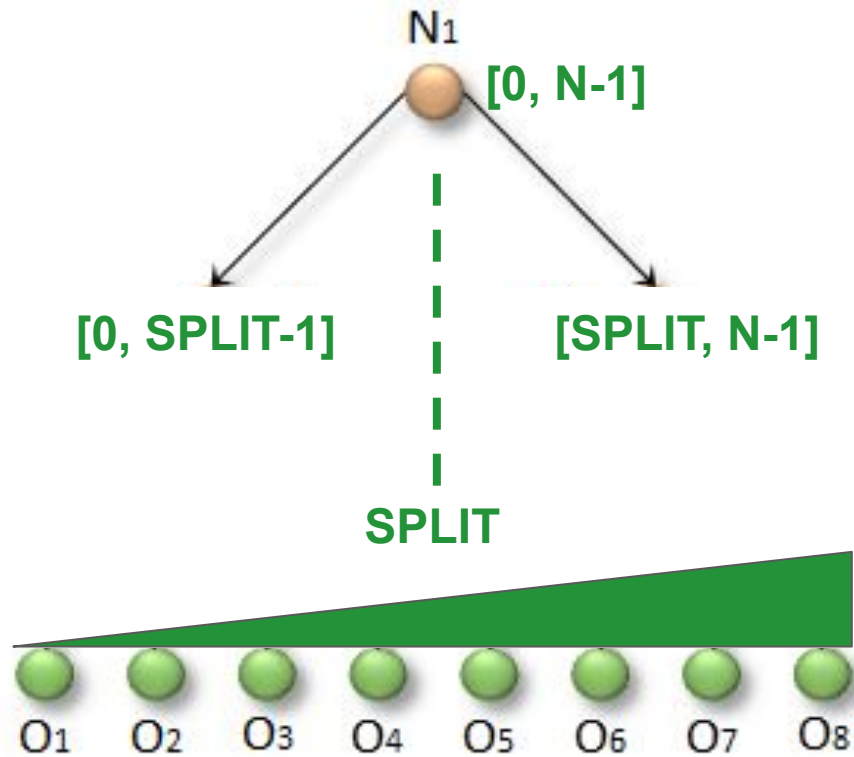
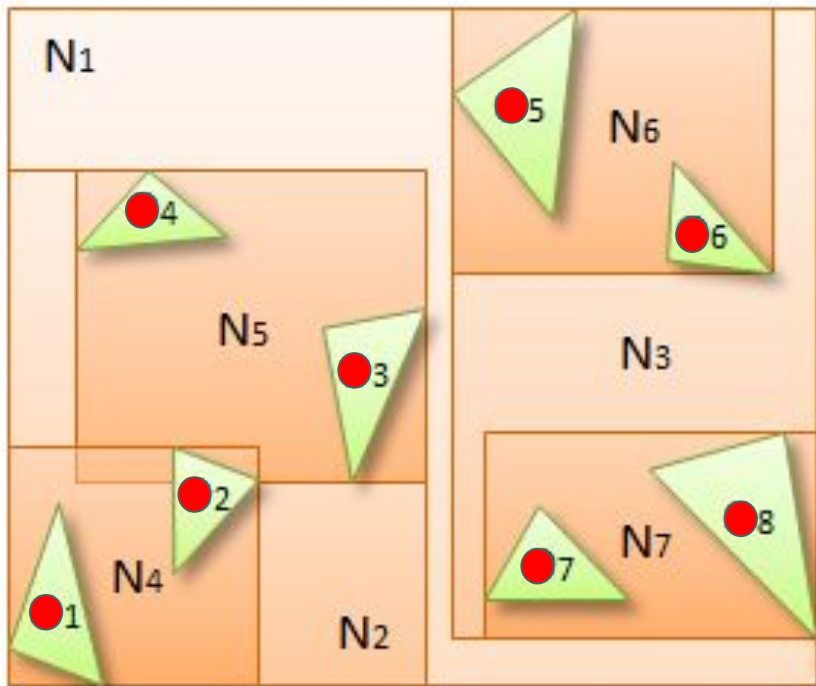
- 1) Есть много треугольников, каждый со своим AABV, это **листья**
- 2) Вычисляем у каждого **листа** код мортона (по точке-центроиду)
- 3) Сортируем **листья** по коду мортона



- 1) Есть много треугольников, каждый со своим AABV, это **листья**
- 2) Вычисляем у каждого **листа** код мортона (по точке-центроиду)
- 3) Сортируем **листья** по коду мортона

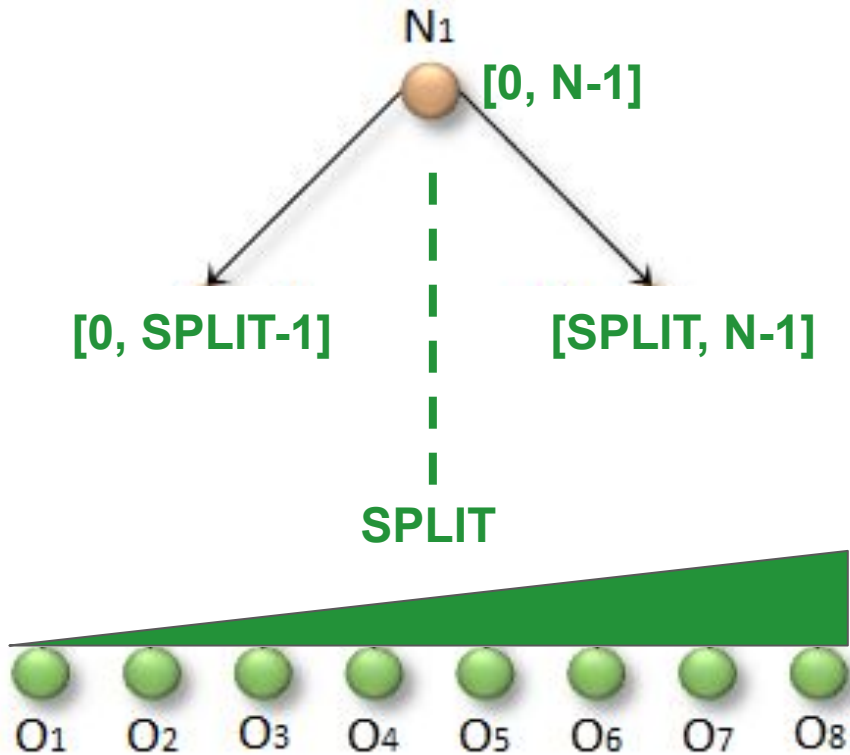
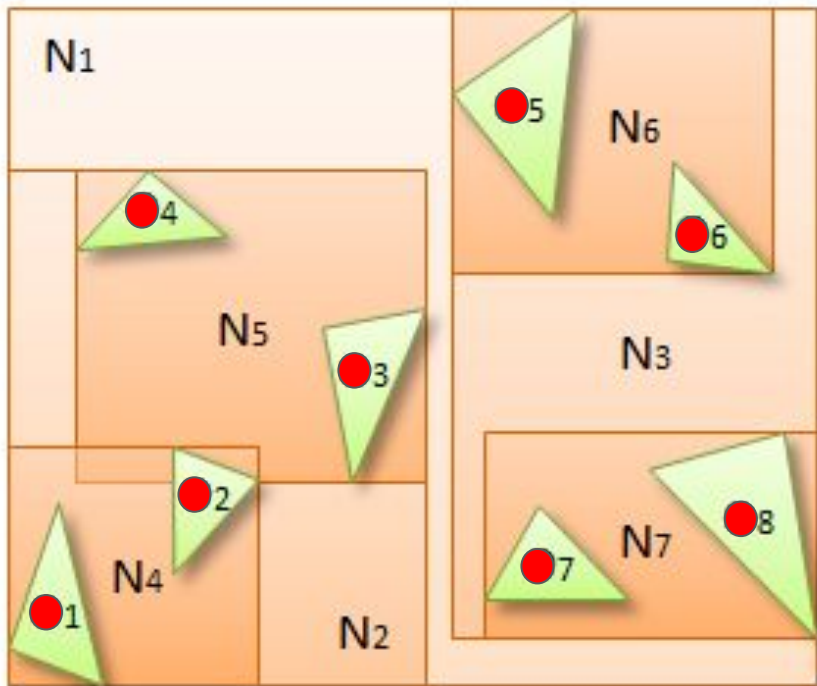


- 1) Есть много треугольников, каждый со своим AABV, это **листья**
- 2) Вычисляем у каждого **листа** код мортона (по точке-центроиду)
- 3) Сортируем **листья** по коду мортона



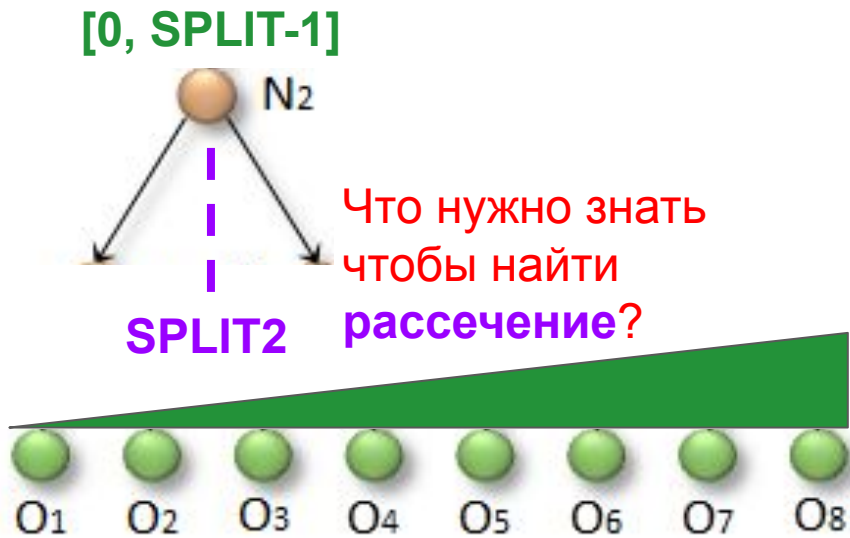
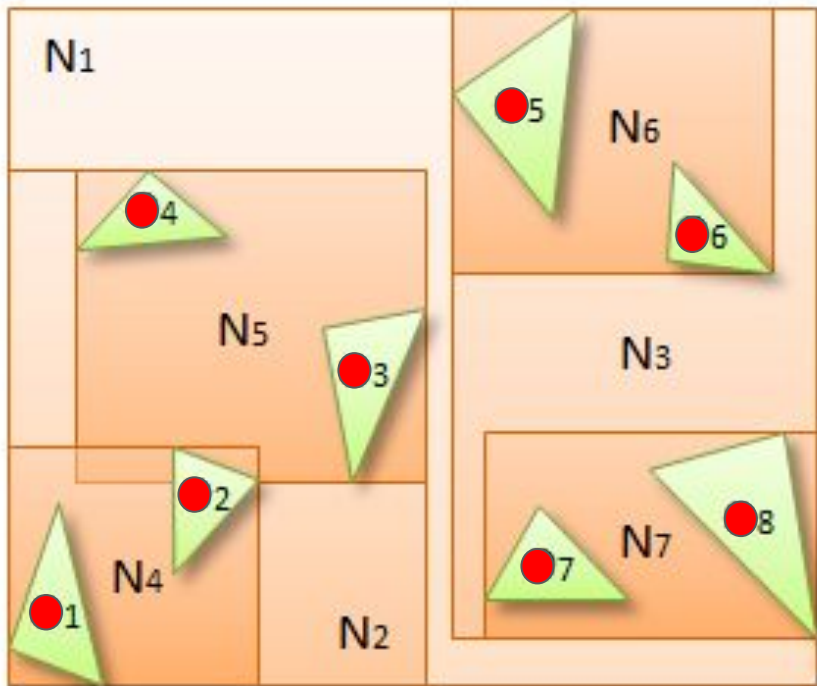
- 1) Есть много треугольников, каждый со своим AABV, это **листья**
- 2) Вычисляем у каждого **листа** код мортона (по точке-центроиду)
- 3) Сортируем **листья** по коду мортона

Нужен массовый параллелизм!



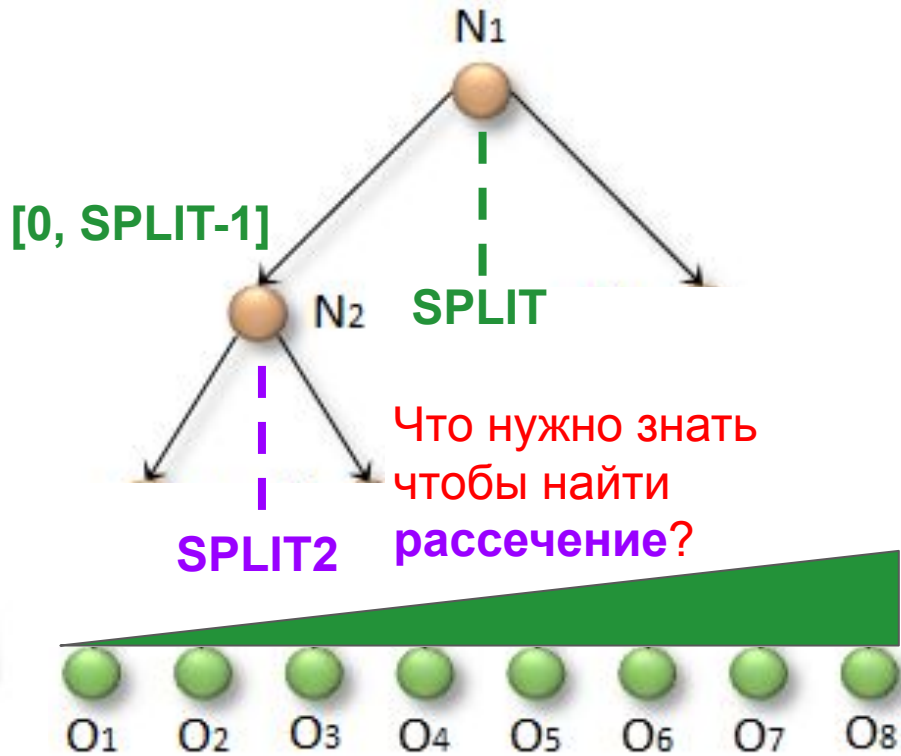
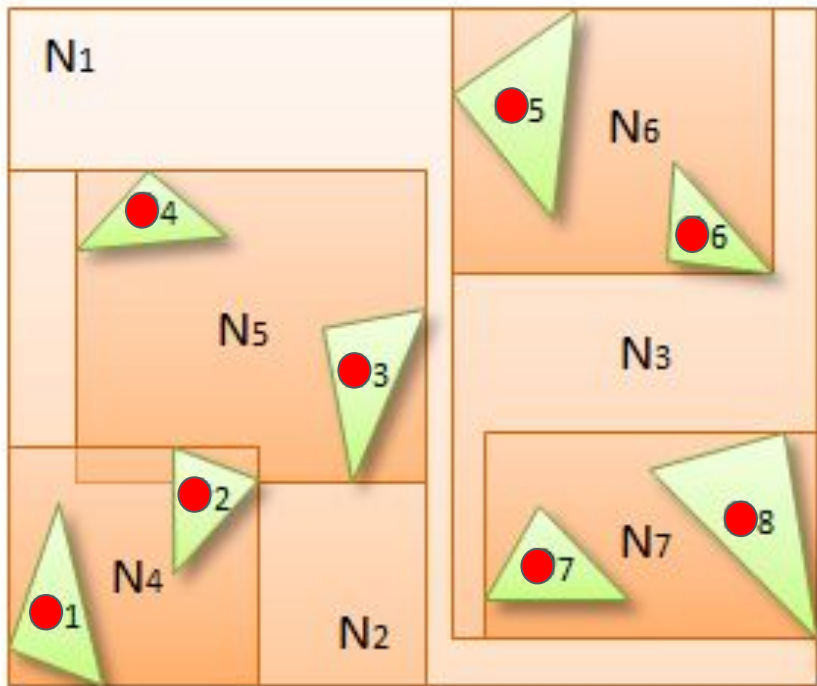
- 1) Есть много треугольников, каждый со своим AABV, это **листья**
- 2) Вычисляем у каждого **листа** код мортона (по точке-центроиду)
- 3) Сортируем **листья** по коду мортона

Нужен массовый параллелизм!



- 1) Есть много треугольников, каждый со своим AABV, это **листья**
- 2) Вычисляем у каждого **листа** код мортона (по точке-центроиду)
- 3) Сортируем **листья** по коду мортона

Нужен массовый параллелизм!



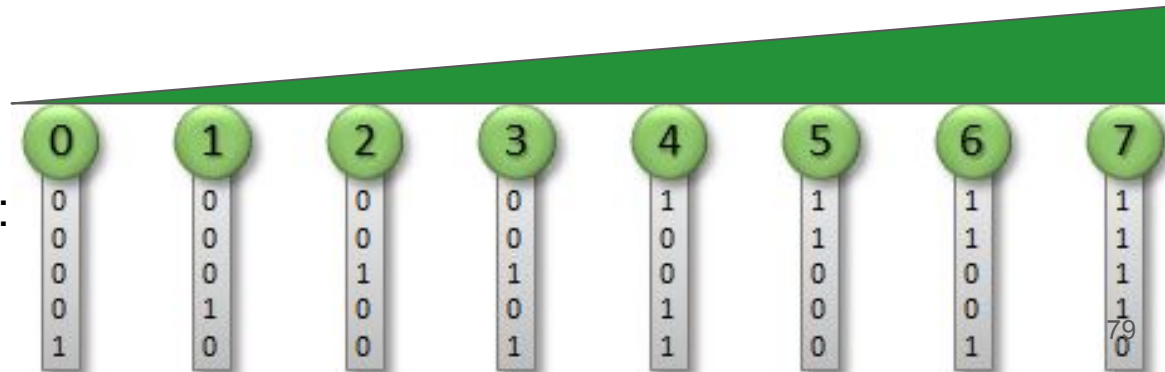
3) ... Сортируем *листья* по коду мортона

Нужен массовый параллелизм!

4) Нужно построить промежуточные узлы LBVH:

Как???

Сортированные Morton Codes:

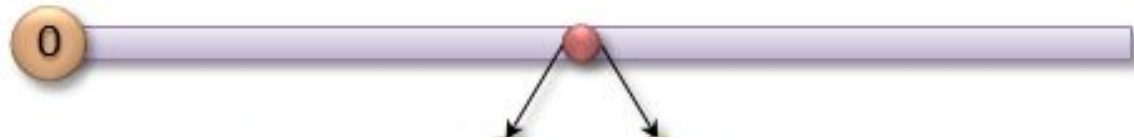


3) ... Сортируем **листья** по коду мортона

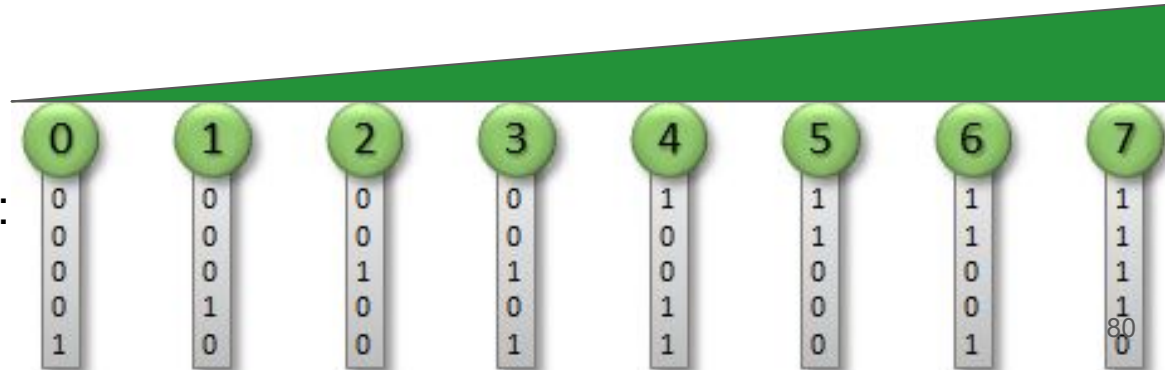
Нужен массовый параллелизм!

4) Нужно построить промежуточные узлы LBVN:

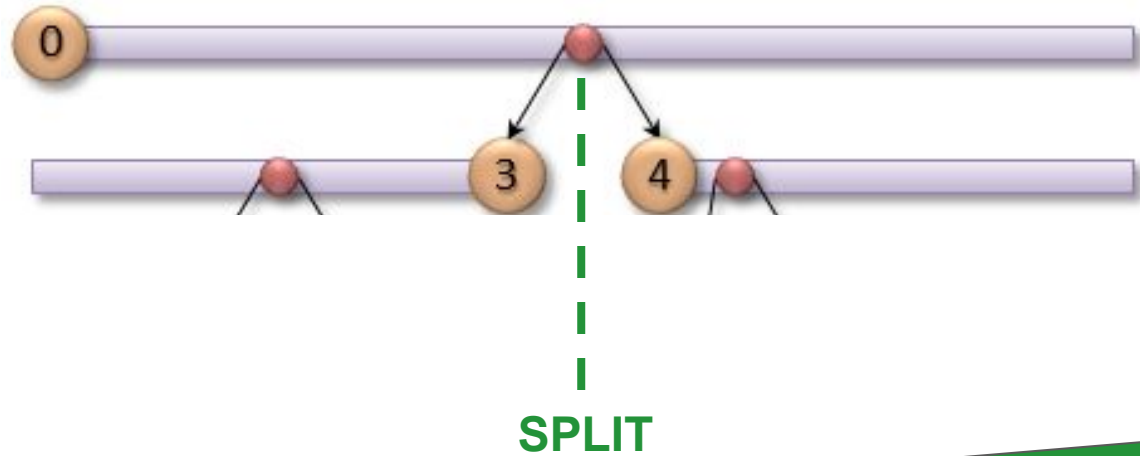
4.1) Каждый **узел** покрывает **подотрезок листьев**, корень покрывает **[0, N-1]**



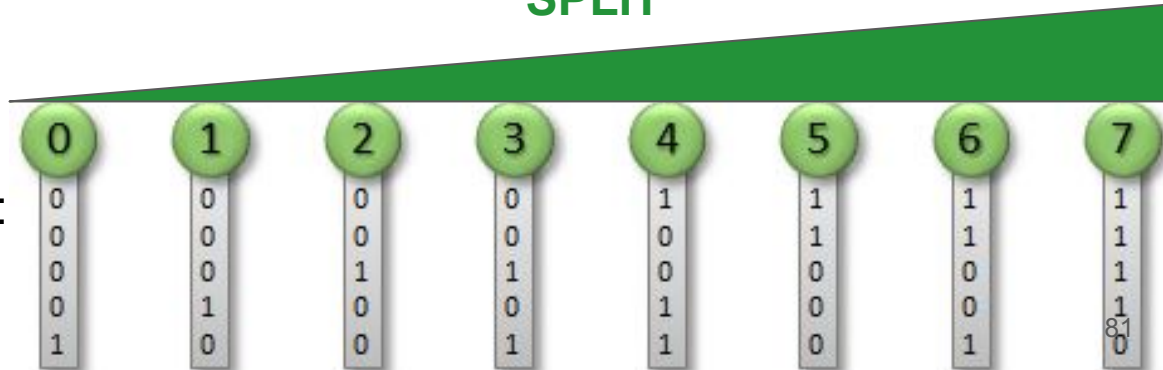
Сортированные Morton Codes:



4.2) Подотрезки *детей* корня покрывают половины рассеянные по **SPLIT**



Сортированные Morton Codes:

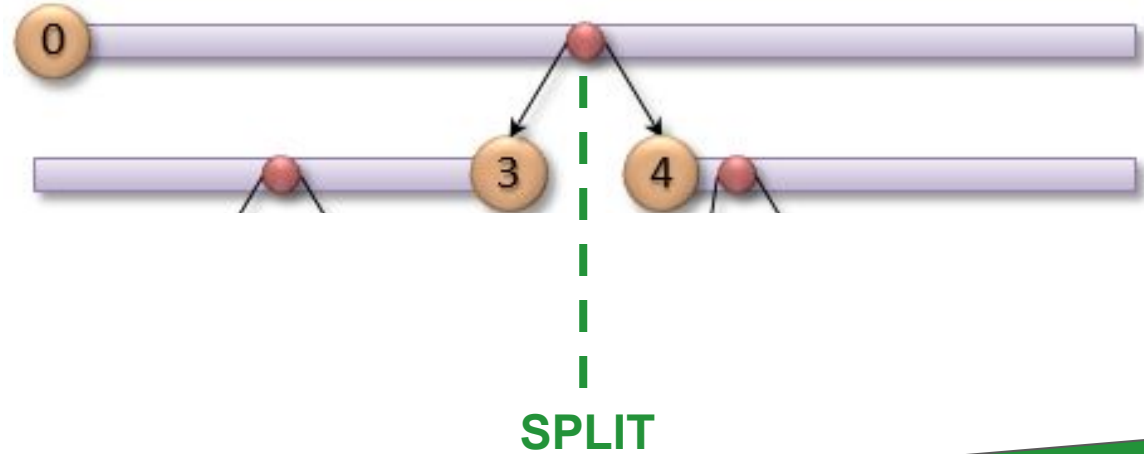


4.2) Подотрезки *детей* корня покрывают половины рассеянные по **SPLIT**

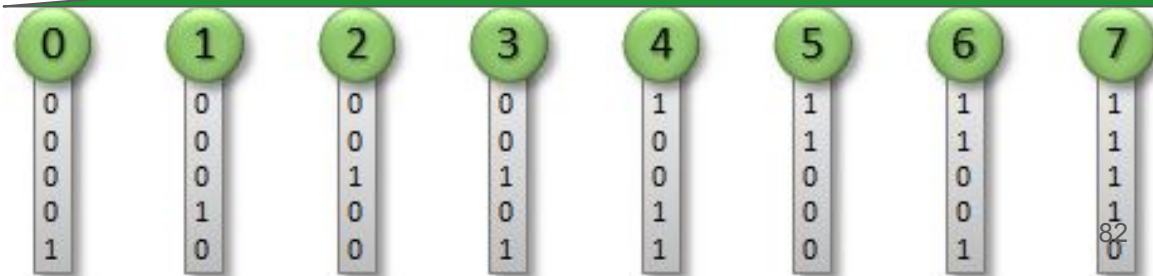
Как его выбрать?

Детерминированно!

Чтобы узлы смогли понять!

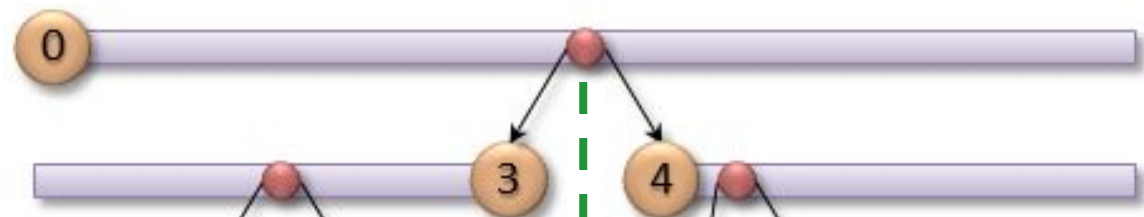


Сортированные Morton Codes:

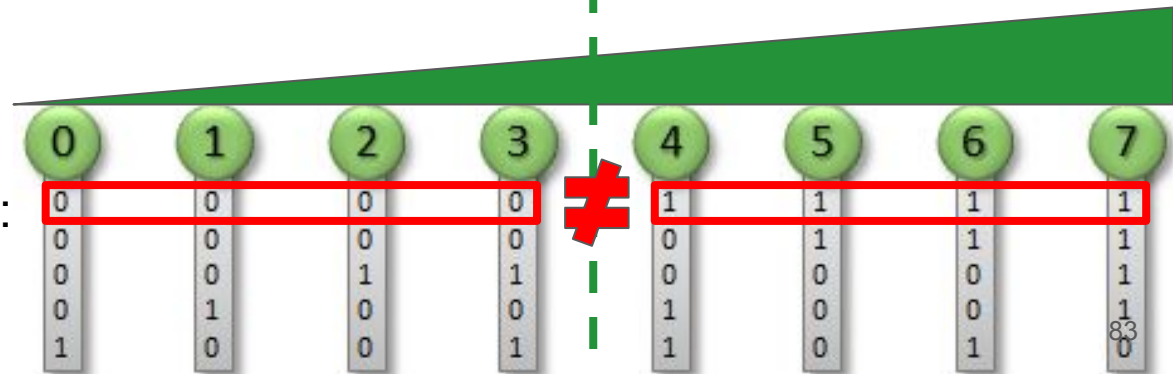


4.2) Подотрезки *детей* корня покрывают половины рассеянные по **SPLIT**

4.3) **SPLIT** - по старшему биту который различается на подотрезке



SPLIT



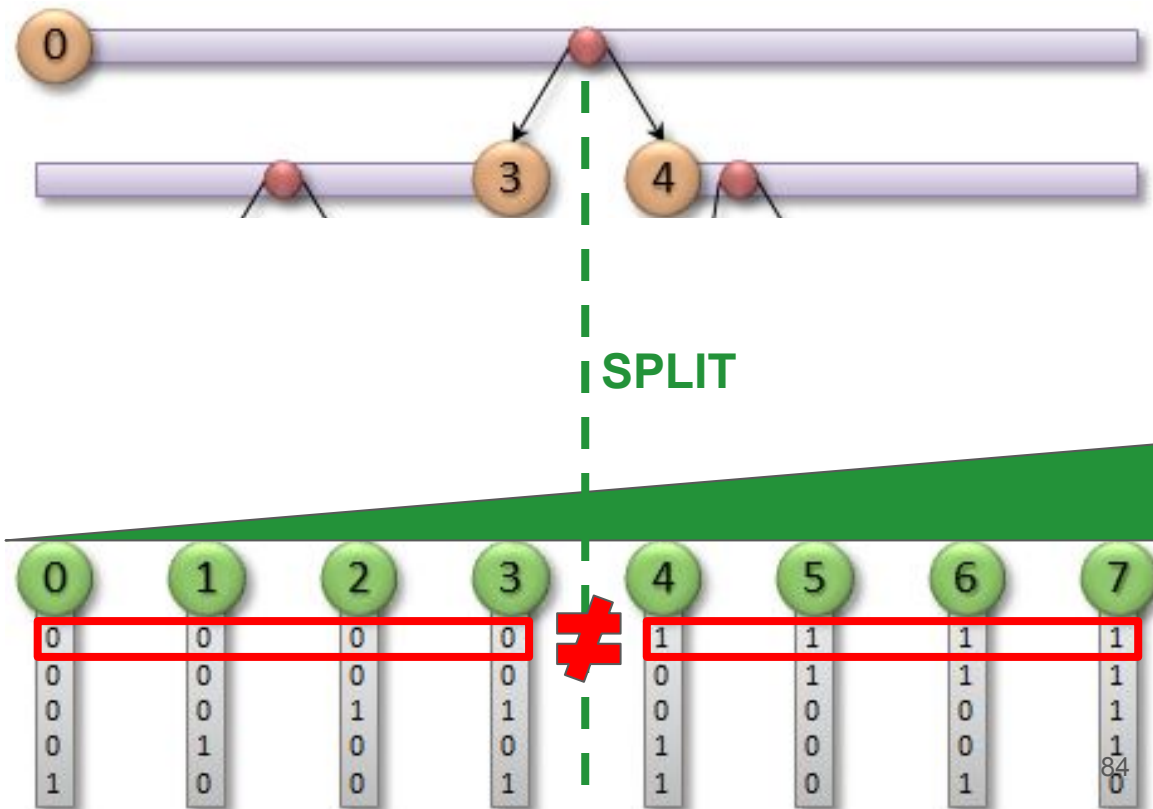
Сортированные Morton Codes:

0	1	2	3	4	5	6	7
0	0	0	0	1	1	1	1
0	0	0	0	0	1	1	1
0	0	1	0	0	0	0	1
0	1	0	0	1	0	0	1
1	0	0	1	1	0	1	0

4.2) Подотрезки *детей* корня покрывают половины рассеянные по **SPLIT**

4.3) **SPLIT** - по старшему биту который различается на подотрезке - как?

Какие коды мортон нужно загрузить из VRAM?

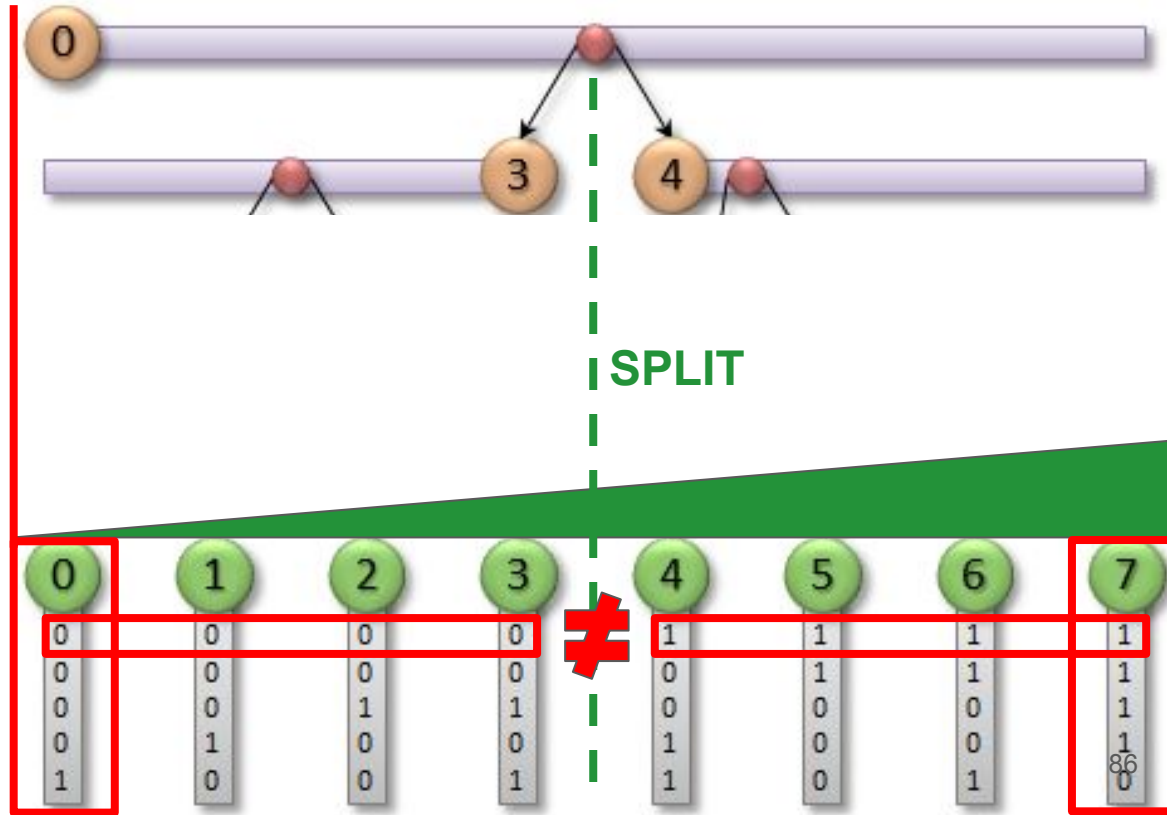


Сортированные Morton Codes:


```

1 int findSplit(unsigned int* sortedMortonCodes, int first, int last)
2 {
3     // Identical Morton codes => split the range in the middle.
4     unsigned int firstCode = sortedMortonCodes[first];
5     unsigned int lastCode = sortedMortonCodes[last];
6
7     if (firstCode == lastCode)
8         return (first + last) >> 1;
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33

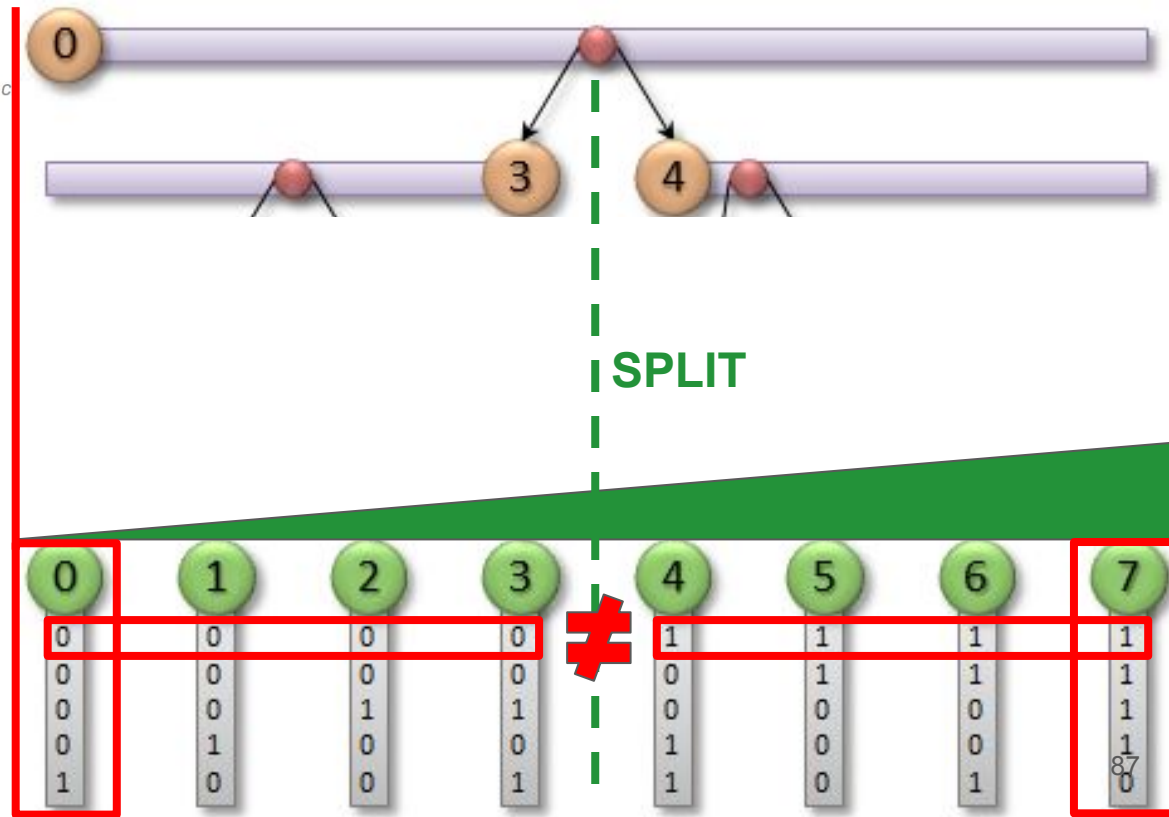
```



```

1 int findSplit(unsigned int* sortedMortonCodes, int first, int last)
2 {
3     // Identical Morton codes => split the range in the middle.
4     unsigned int firstCode = sortedMortonCodes[first];
5     unsigned int lastCode = sortedMortonCodes[last];
6
7     if (firstCode == lastCode)
8         return (first + last) >> 1;
9
10    // Calculate the number of highest bits that are the same
11    // for all objects, using the count-leading-zeros intrinsic
12    int commonPrefix = __clz(firstCode ^ lastCode);
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33

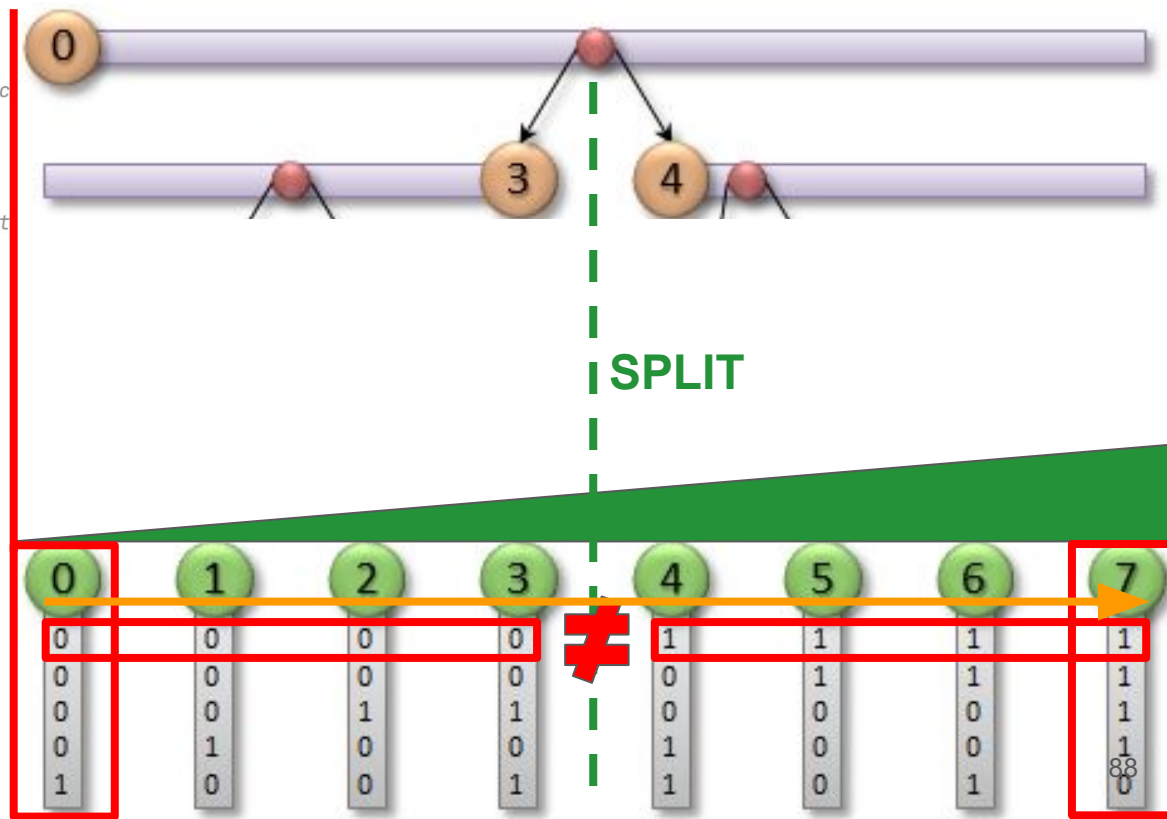
```




```

1 int findSplit(unsigned int* sortedMortonCodes, int first, int last)
2 {
3     // Identical Morton codes => split the range in the middle.
4     unsigned int firstCode = sortedMortonCodes[first];
5     unsigned int lastCode = sortedMortonCodes[last];
6
7     if (firstCode == lastCode)
8         return (first + last) >> 1;
9
10    // Calculate the number of highest bits that are the same
11    // for all objects, using the count-leading-zeros intrinsic
12    int commonPrefix = __clz(firstCode ^ lastCode);
13
14    // Use binary search to find where the next bit differs.
15    // Specifically, we are looking for the highest object that
16    // shares more than commonPrefix bits with the first one.
17
18    int split = first; // initial guess
19    int step = last - first;
20    // →
21    do {
22
23
24
25
26
27
28
29
30
31    } while (step > 1);
32
33    return split;

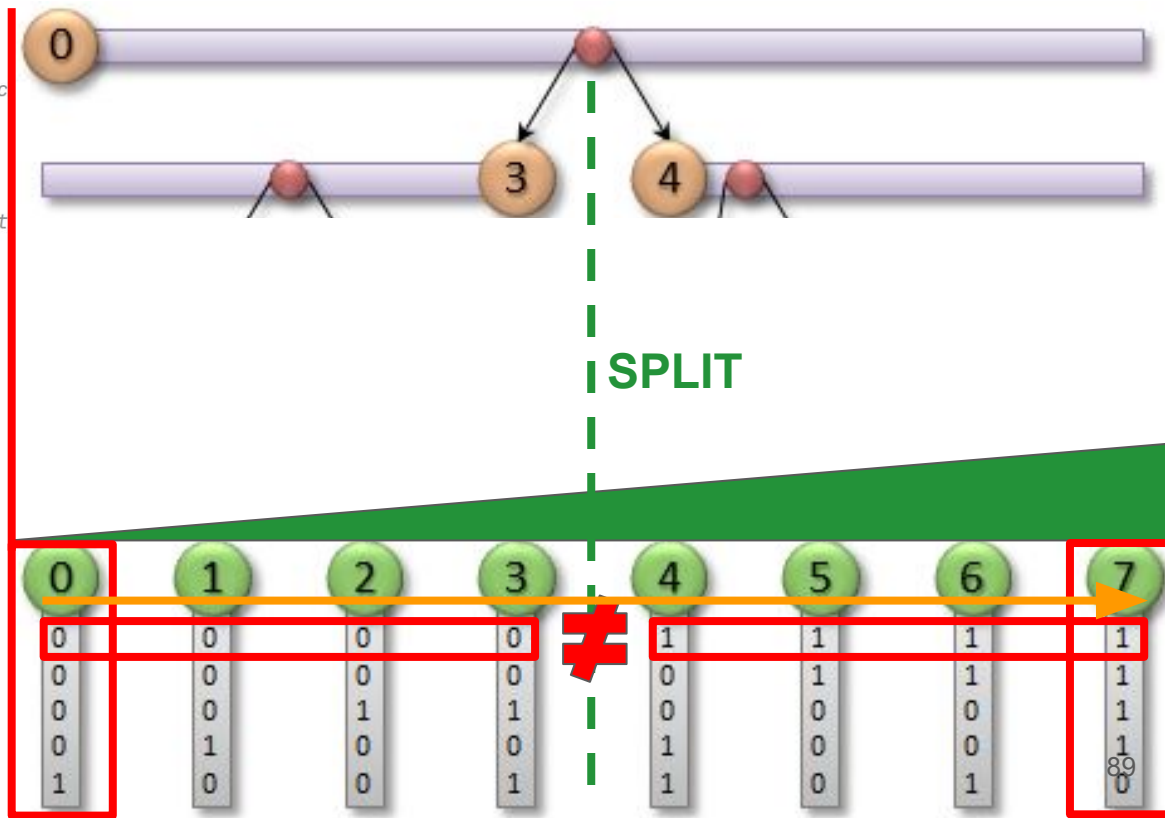
```



```

1 int findSplit(unsigned int* sortedMortonCodes, int first, int last)
2 {
3     // Identical Morton codes => split the range in the middle.
4     unsigned int firstCode = sortedMortonCodes[first];
5     unsigned int lastCode = sortedMortonCodes[last];
6
7     if (firstCode == lastCode)
8         return (first + last) >> 1;
9
10    // Calculate the number of highest bits that are the same
11    // for all objects, using the count-leading-zeros intrinsic
12    int commonPrefix = __clz(firstCode ^ lastCode);
13
14    // Use binary search to find where the next bit differs.
15    // Specifically, we are looking for the highest object that
16    // shares more than commonPrefix bits with the first one.
17
18    int split = first; // initial guess
19    int step = last - first;
20    // →
21    do {
22        step = (step + 1) >> 1; // exponential decrease
23        int newSplit = split + step; // proposed new position
24
25        // 0 1 2 3 4 5 6 7
26        // 0 0 0 0 1 1 1 1
27        // 0 0 0 0 0 1 1 1
28        // 0 0 1 0 0 0 0 1
29        // 0 1 0 0 1 0 0 1
30        // 0 0 0 1 0 0 0 0
31    } while (step > 1);
32
33    return split;

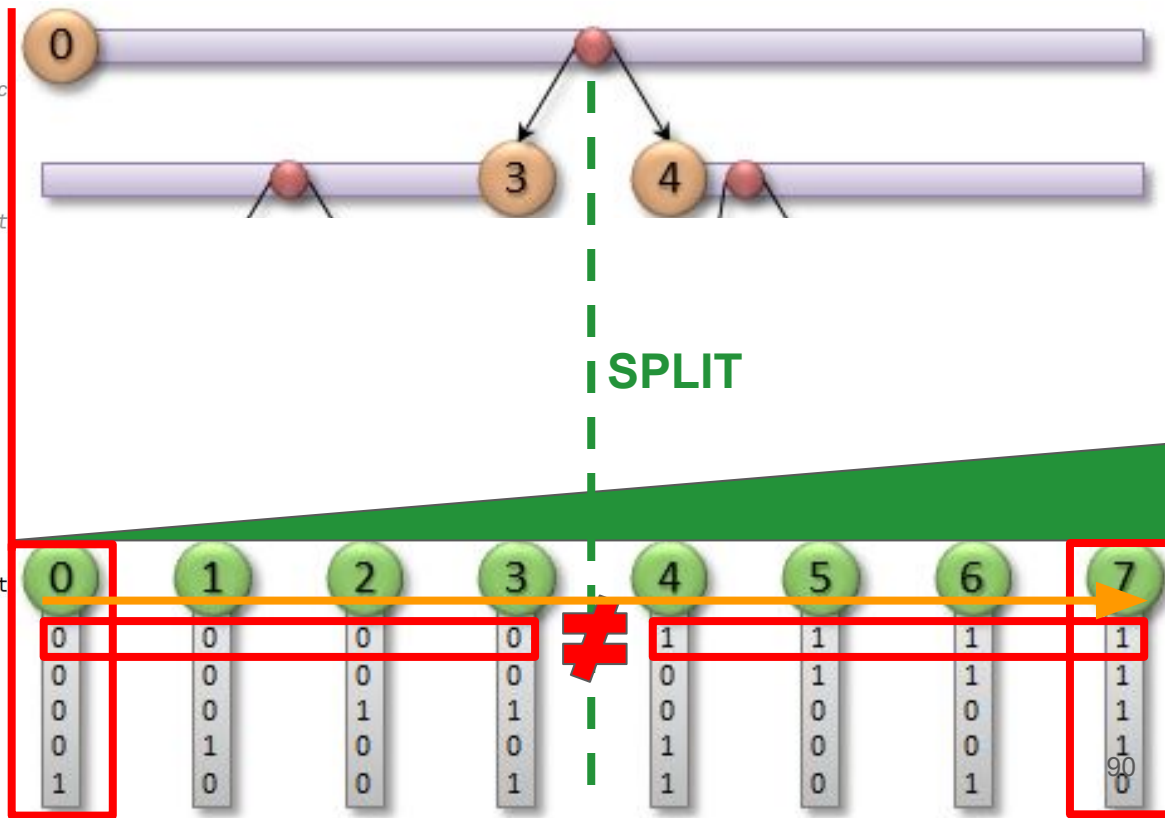
```



```

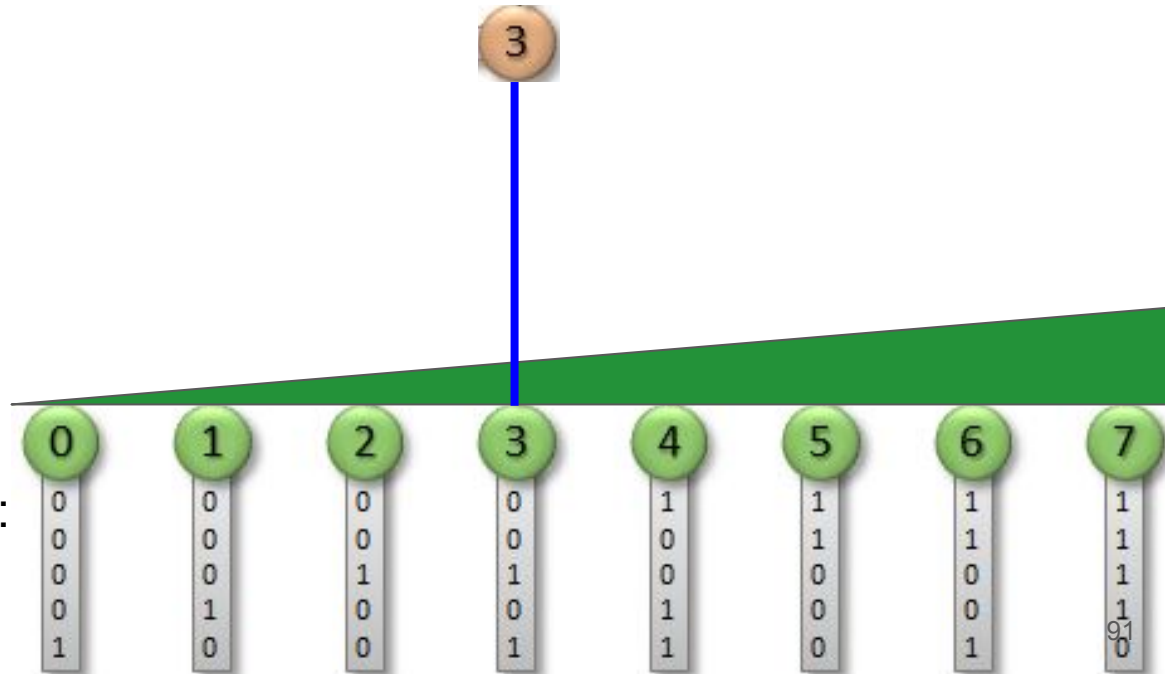
1 int findSplit(unsigned int* sortedMortonCodes, int first, int last)
2 {
3     // Identical Morton codes => split the range in the middle.
4     unsigned int firstCode = sortedMortonCodes[first];
5     unsigned int lastCode = sortedMortonCodes[last];
6
7     if (firstCode == lastCode)
8     {
9         return (first + last) >> 1;
10    }
11    // Calculate the number of highest bits that are the same
12    // for all objects, using the count-leading-zeros intrinsic
13    int commonPrefix = __clz(firstCode ^ lastCode);
14    // Use binary search to find where the next bit differs.
15    // Specifically, we are looking for the highest object that
16    // shares more than commonPrefix bits with the first one.
17
18    int split = first; // initial guess
19    int step = last - first;
20    // →
21    do {
22        step = (step + 1) >> 1; // exponential decrease
23        int newSplit = split + step; // proposed new position
24
25        if (newSplit < last) {
26            unsigned int splitCode = sortedMortonCodes[newSplit];
27            int splitPrefix = __clz(firstCode ^ splitCode);
28            if (splitPrefix > commonPrefix)
29            {
30                split = newSplit; // accept proposal
31            }
32        }
33    } while (step > 1);
34
35    return split;

```



- 4.2) Подотрезки **детей** корня покрывают половины рассеянные по **SPLIT**
- 4.3) **SPLIT** - по старшему биту который различается на **подотрезке** - **бин.п.**
- 4.4) Но сначала каждый **узел** находит **???**

Сортированные Morton Codes:



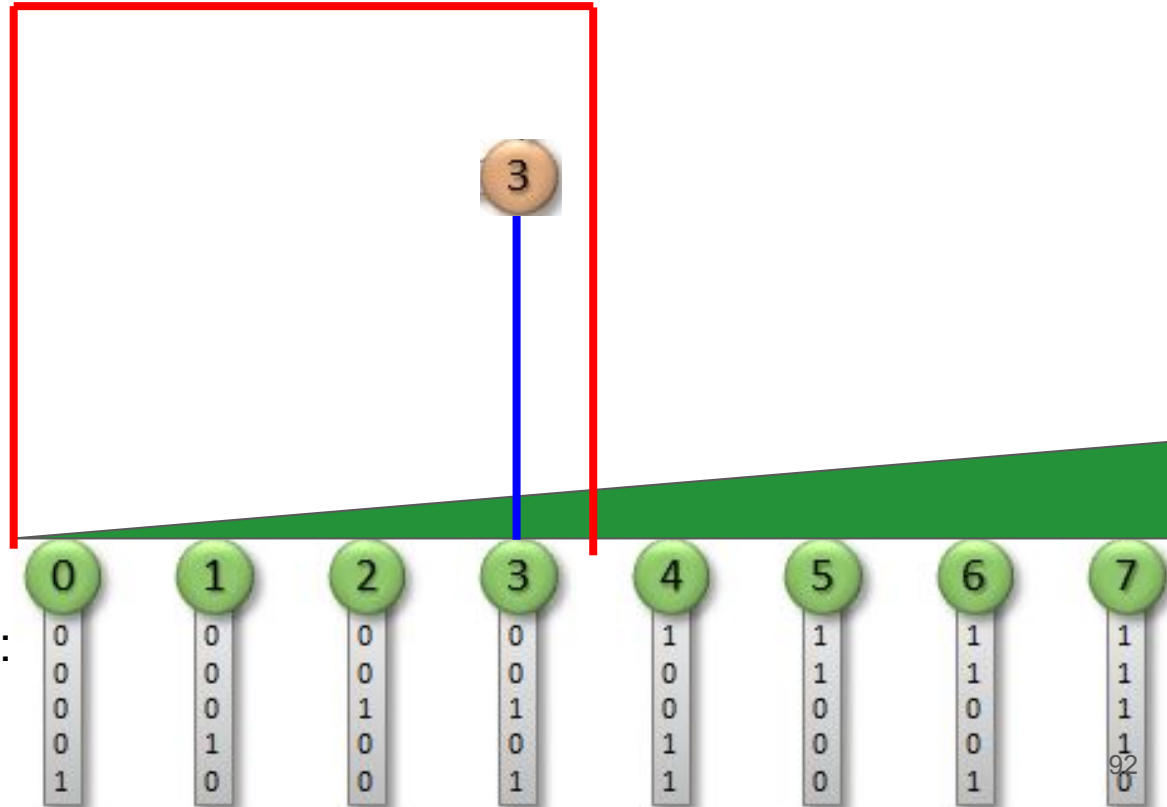
4.2) Подотрезки **детей** корня покрывают половины рассеянные по **SPLIT**

4.3) **SPLIT** - по старшему биту который различается на подотрезке - **бин.п.**

4.4) Но сначала каждый **узел** находит свой **подотрезок** ... **как???**

Какие могут быть варианты?

Сортированные Morton Codes:



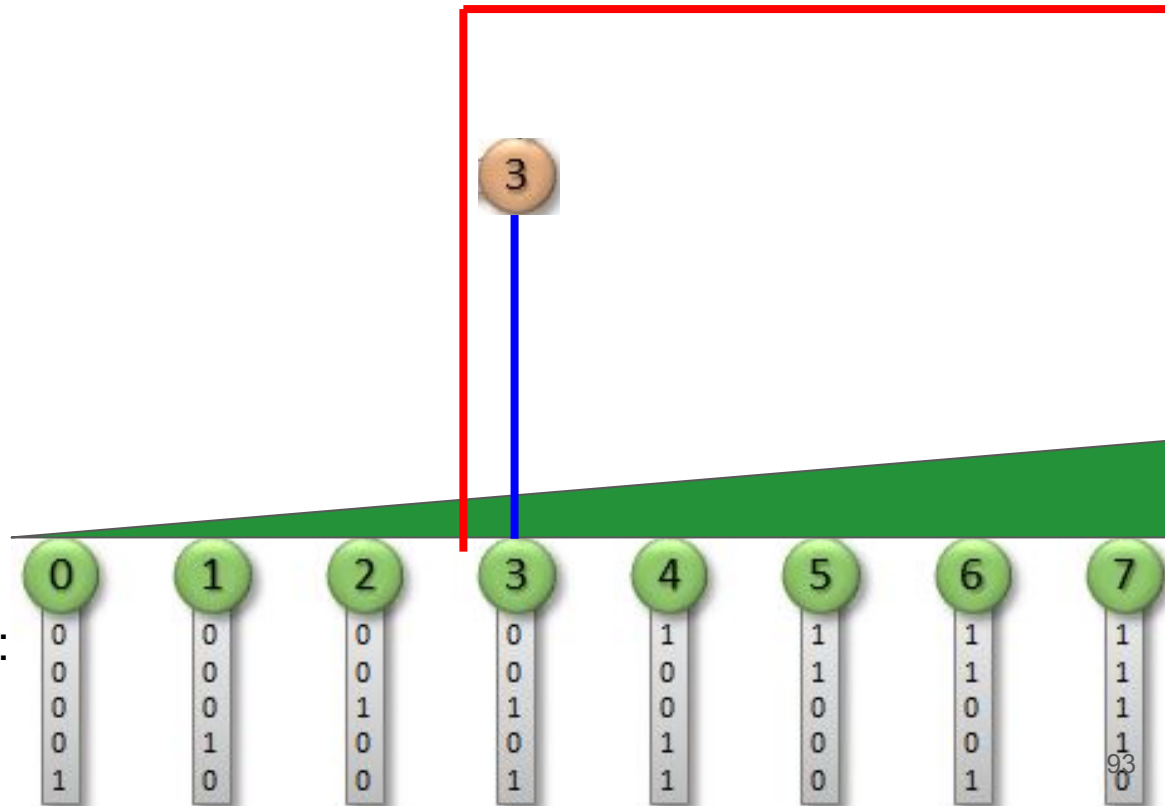
4.2) Подотрезки **детей** корня покрывают половины рассеянные по **SPLIT**

4.3) **SPLIT** - по старшему биту который различается на **подотрезке** - **бин.п.**

4.4) Но сначала каждый **узел** находит свой **подотрезок** ... **как???**

Какие могут быть варианты?

Сортированные Morton Codes:



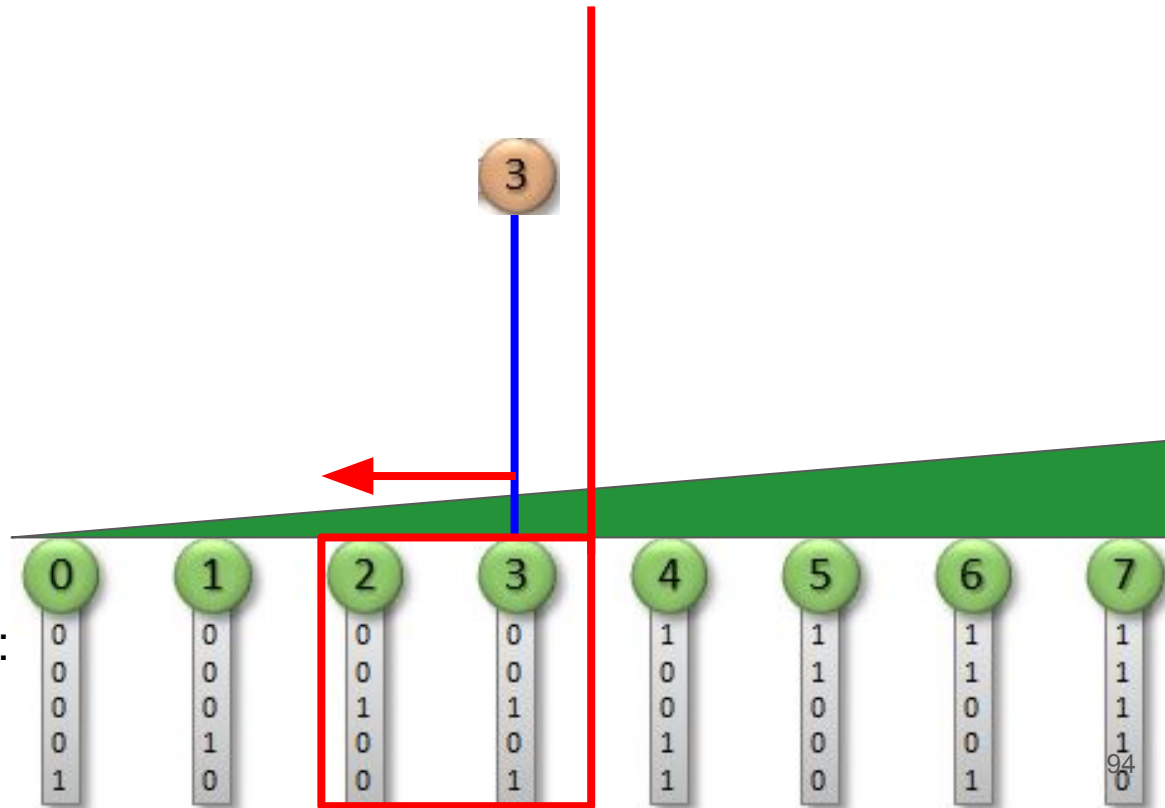
4.2) Подотрезки **детей** корня покрывают половины рассеянные по **SPLIT**

4.3) **SPLIT** - по старшему биту который различается на подотрезке - **бин.п.**

4.4) Но сначала каждый **узел** находит свой **подотрезок** ... **как???**

Какие могут быть варианты?

Сортированные Morton Codes:



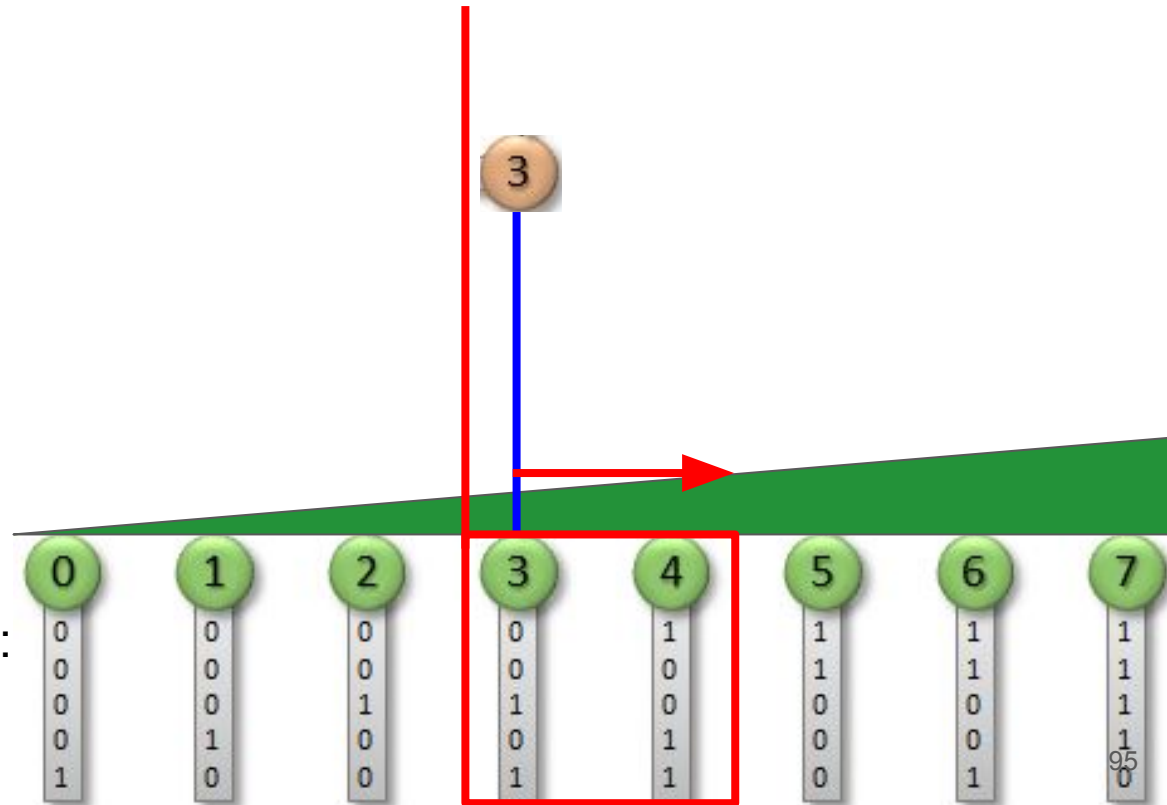
4.2) Подотрезки **детей** корня покрывают половины рассеянные по **SPLIT**

4.3) **SPLIT** - по старшему биту который различается на **подотрезке** - **бин.п.**

4.4) Но сначала каждый **узел** находит свой **подотрезок** ... **как???**

Какие могут быть варианты?

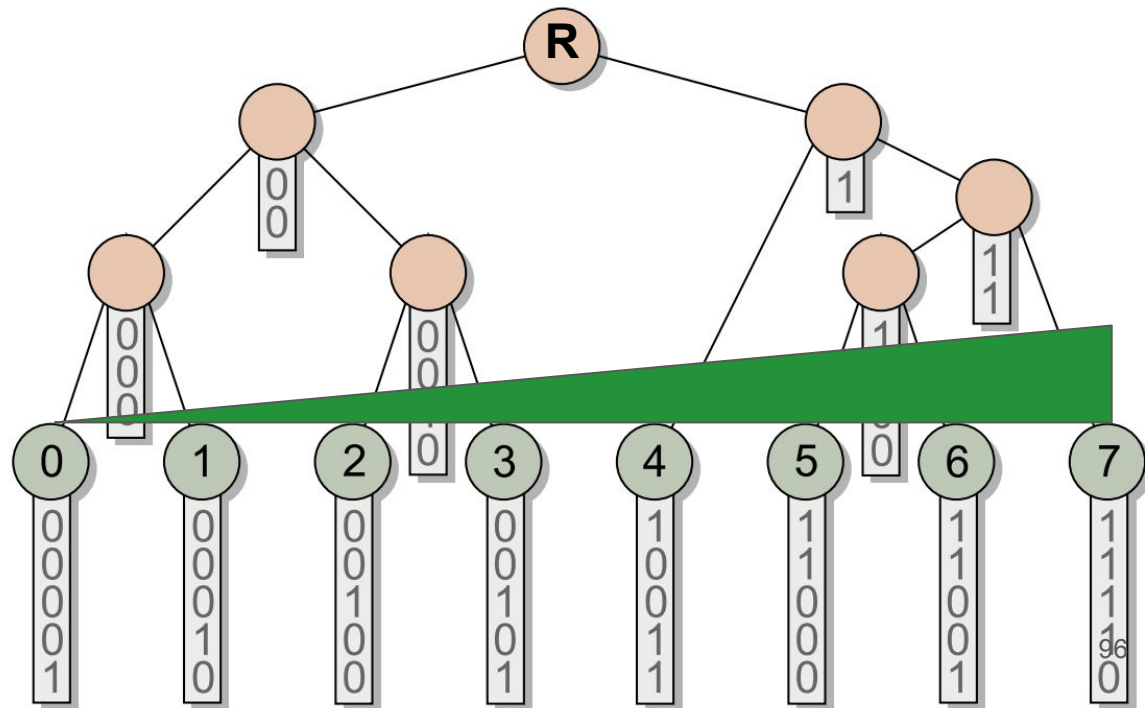
Сортированные Morton Codes:



4.2) Подотрезки **детей** корня покрывают половины рассеянные по **SPLIT**

4.3) **SPLIT** - по старшему биту который различается на подотрезке - **бин.п.**

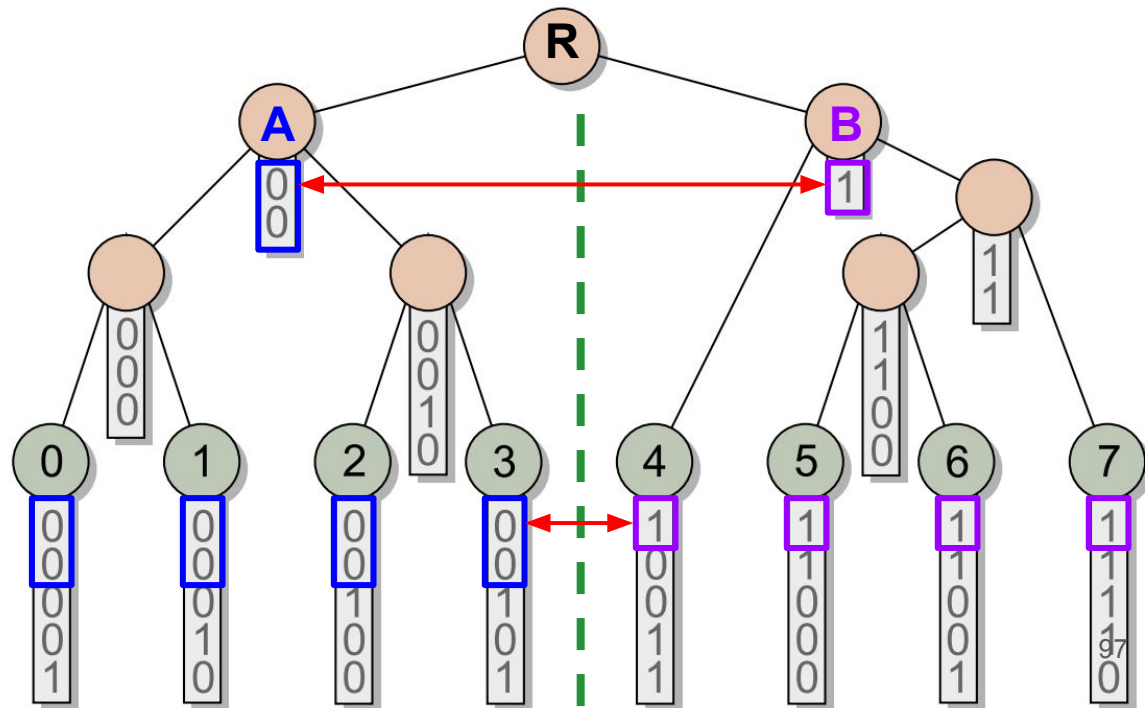
4.4) Но сначала каждый **узел** находит свой **подотрезок** ... **как???**



4.2) Подотрезки **детей** корня покрывают половины рассеянные по **SPLIT**

4.3) **SPLIT** - по старшему биту который различается на подотрезке - **бин.п.**

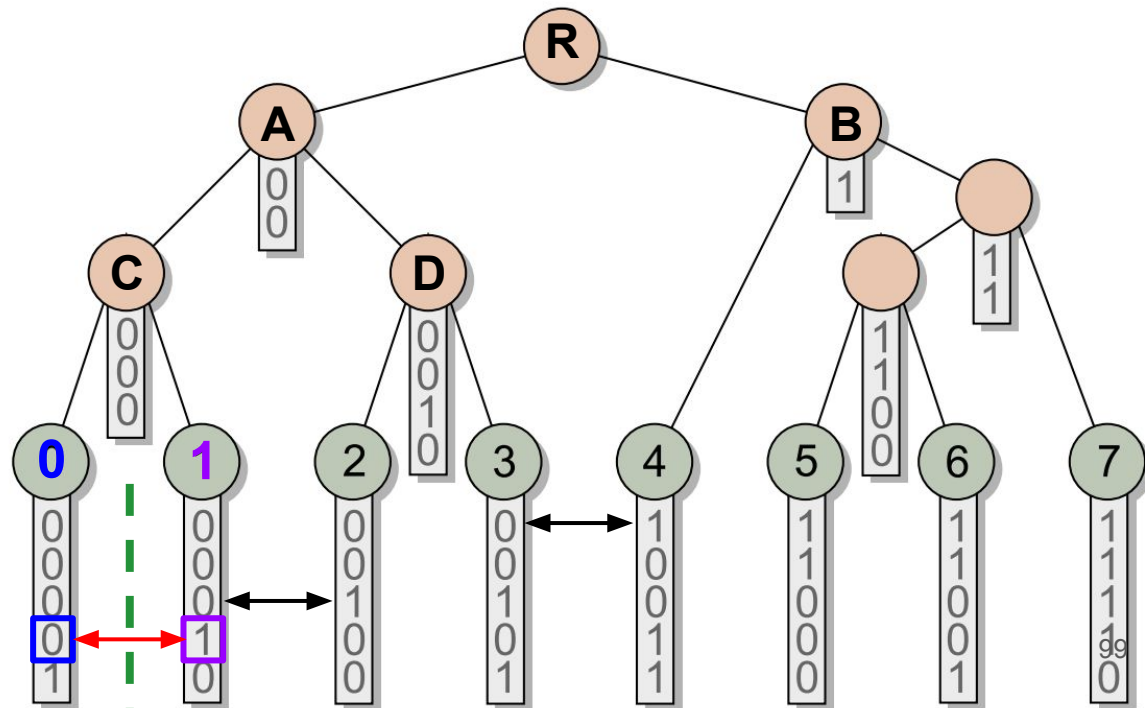
4.4) Но сначала каждый **узел** находит свой подотрезок ... **как???**



4.2) Подотрезки **детей** корня покрывают половины рассеянные по **SPLIT**

4.3) **SPLIT** - по старшему биту который различается на подотрезке - **бин.п.**

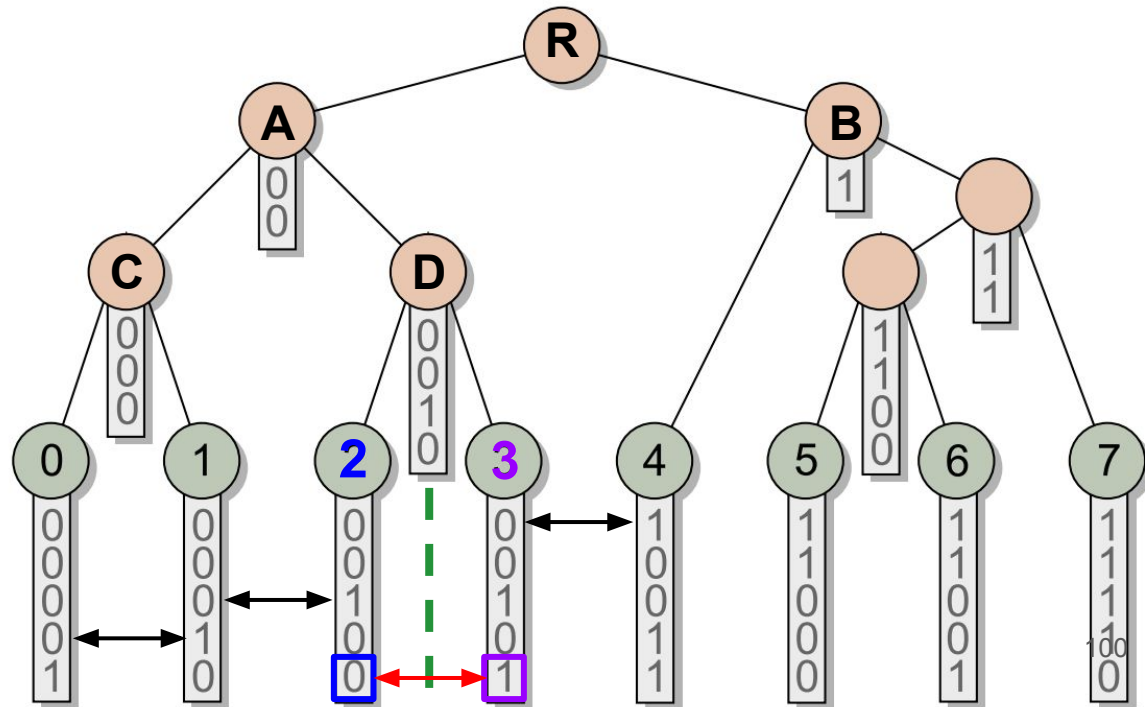
4.4) Но сначала каждый **узел** находит свой **подотрезок** ... **как???**



4.2) Подотрезки **детей** корня покрывают половины рассеянные по **SPLIT**

4.3) **SPLIT** - по старшему биту который различается на подотрезке - **бин.п.**

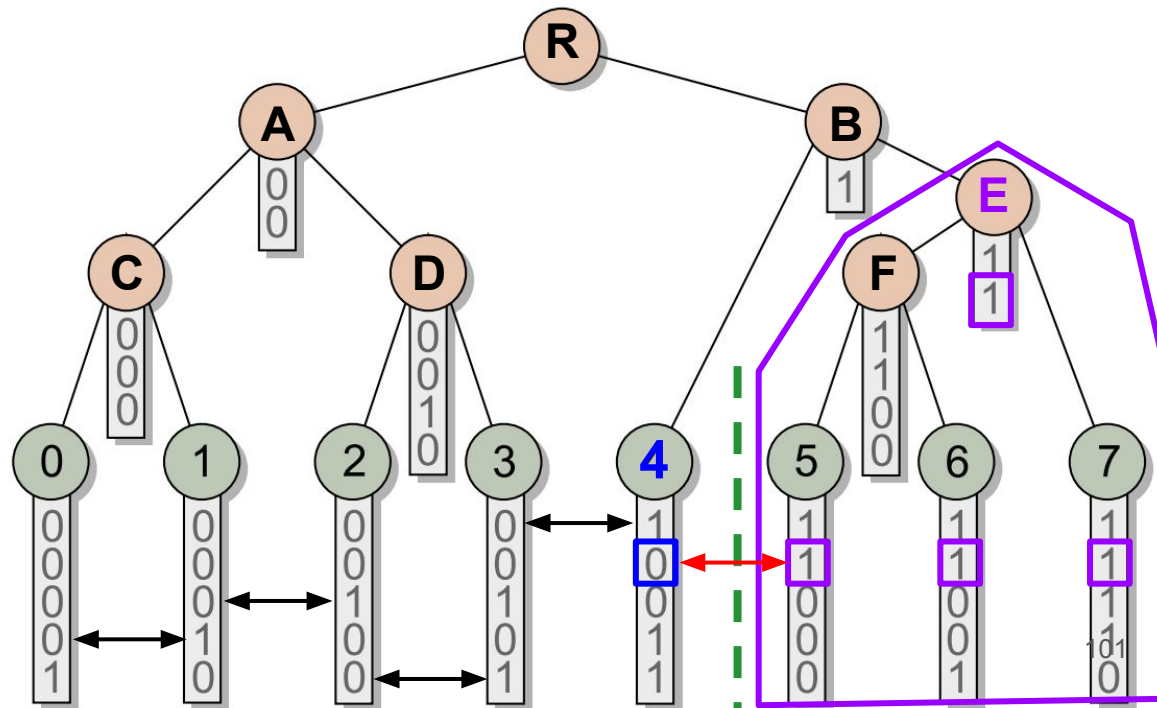
4.4) Но сначала каждый **узел** находит свой подотрезок ... **как???**



4.2) Подотрезки **детей** корня покрывают половины рассеянные по **SPLIT**

4.3) **SPLIT** - по старшему биту который различается на подотрезке - **бин.п.**

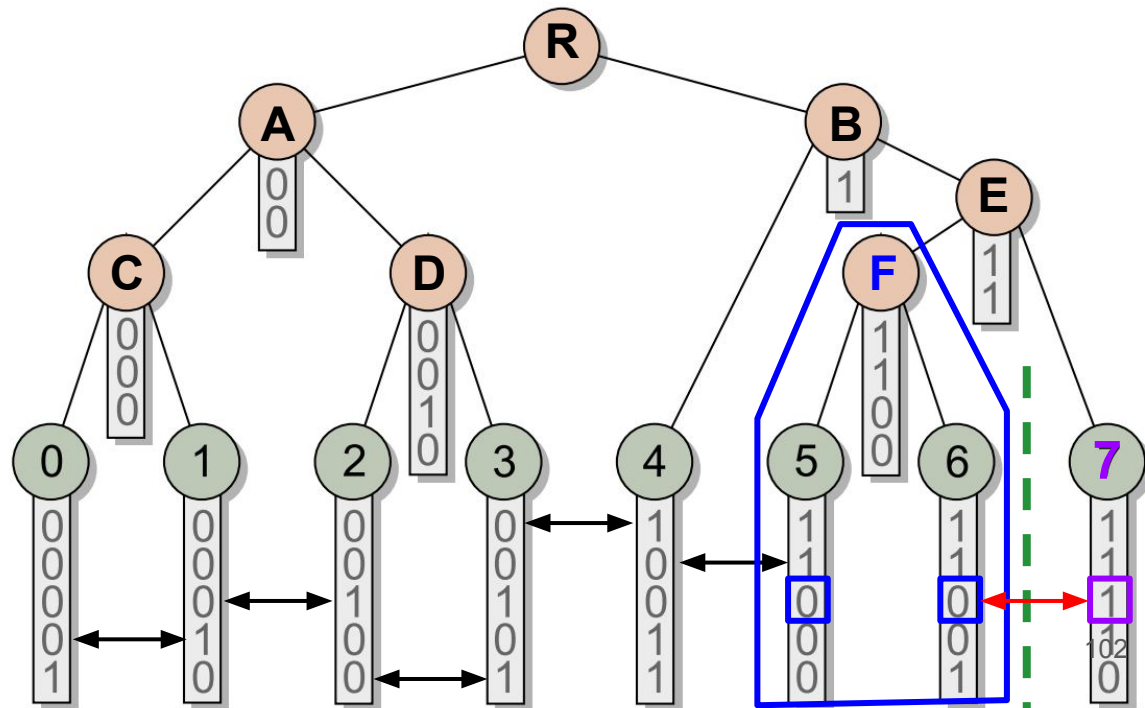
4.4) Но сначала каждый **узел** находит свой подотрезок ... **как???**



4.2) Подотрезки **детей** корня покрывают половины рассеянные по **SPLIT**

4.3) **SPLIT** - по старшему биту который различается на подотрезке - **бин.п.**

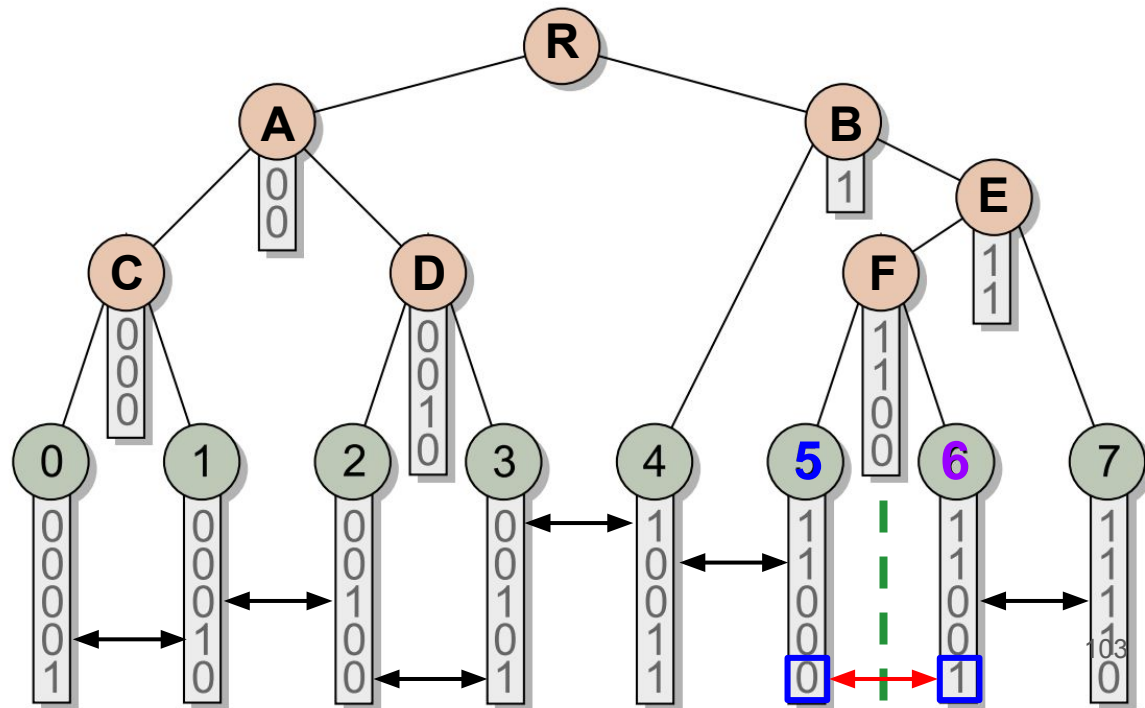
4.4) Но сначала каждый **узел** находит свой подотрезок ... **как???**



4.2) Подотрезки **детей** корня покрывают половины рассеянные по **SPLIT**

4.3) **SPLIT** - по старшему биту который различается на подотрезке - **бин.п.**

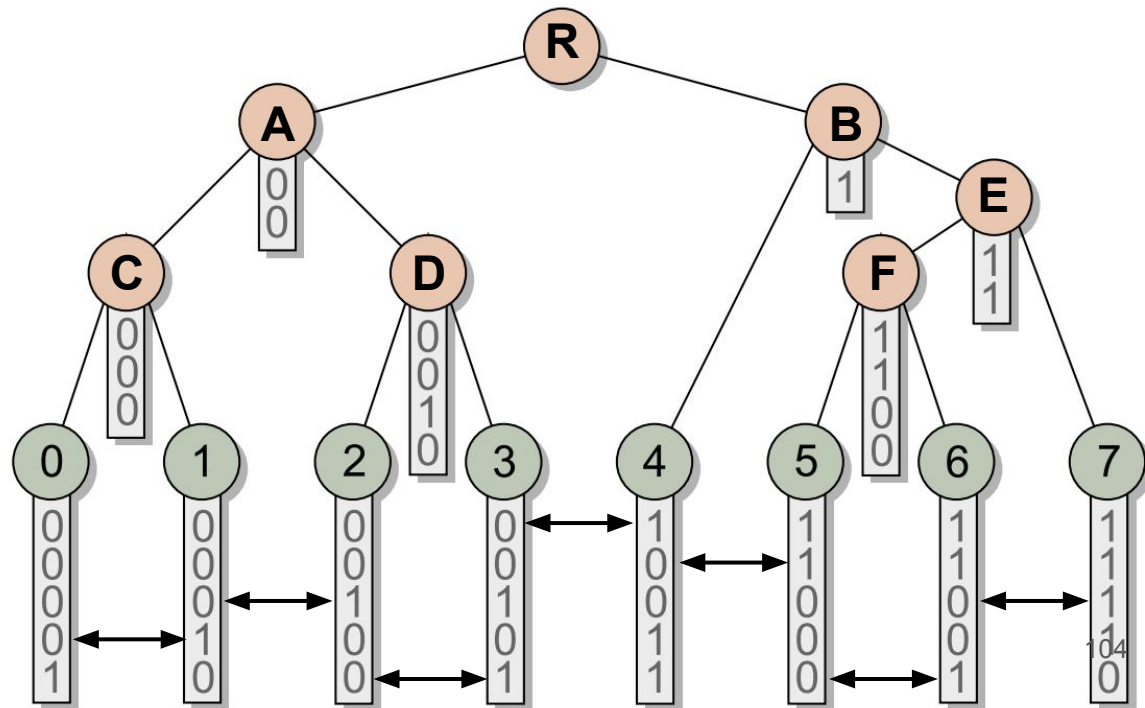
4.4) Но сначала каждый **узел** находит свой **подотрезок** ... **как???**



4.2) Подотрезки **детей** корня покрывают половины рассеянные по **SPLIT**

4.3) **SPLIT** - по старшему биту который различается на подотрезке - **бин.п.**

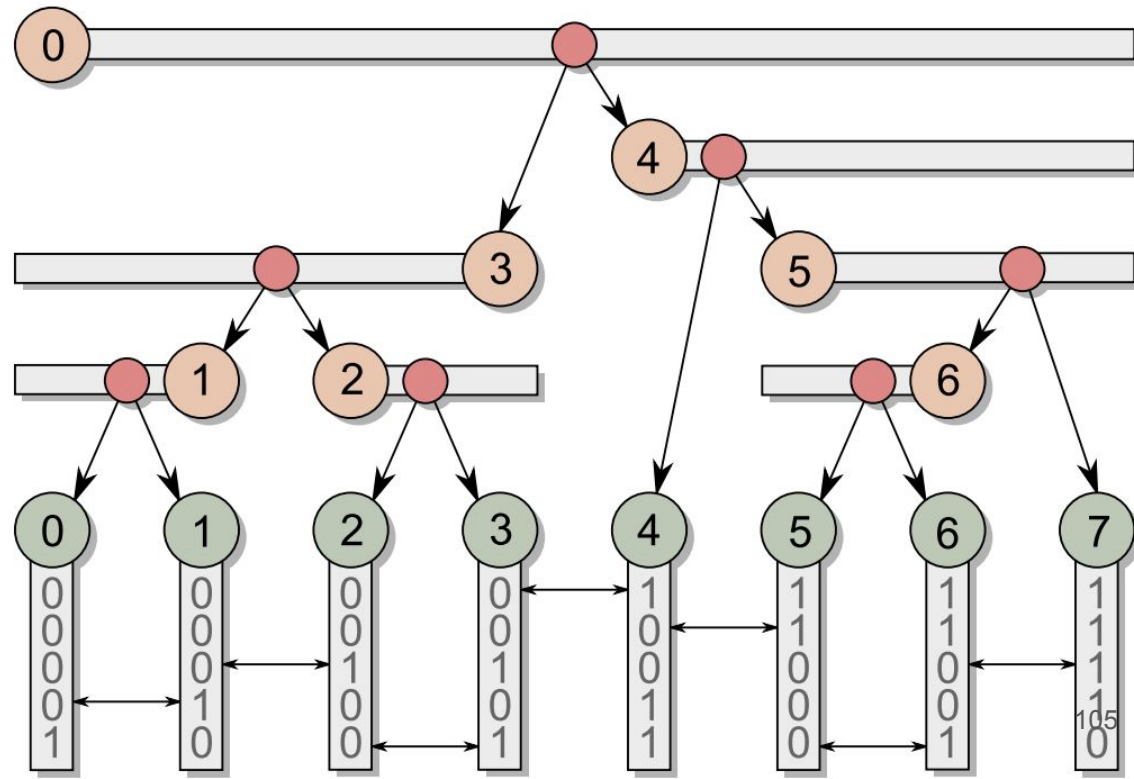
4.4) Но сначала каждый **узел** находит свой подотрезок ... **как???**



4.2) Подотрезки **детей** корня покрывают половины рассеянные по **SPLIT**

4.3) **SPLIT** - по старшему биту который различается на подотрезке - **бин.п.**

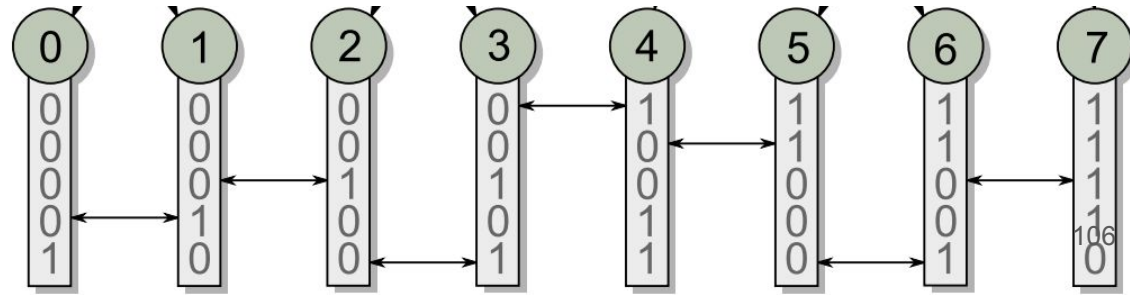
4.4) Но сначала каждый **узел** находит свой подотрезок ... **как???**



4.2) Подотрезки **детей** корня покрывают половины рассеянные по **SPLIT**

4.3) **SPLIT** - по старшему биту который различается на подотрезке - **бин.п.**

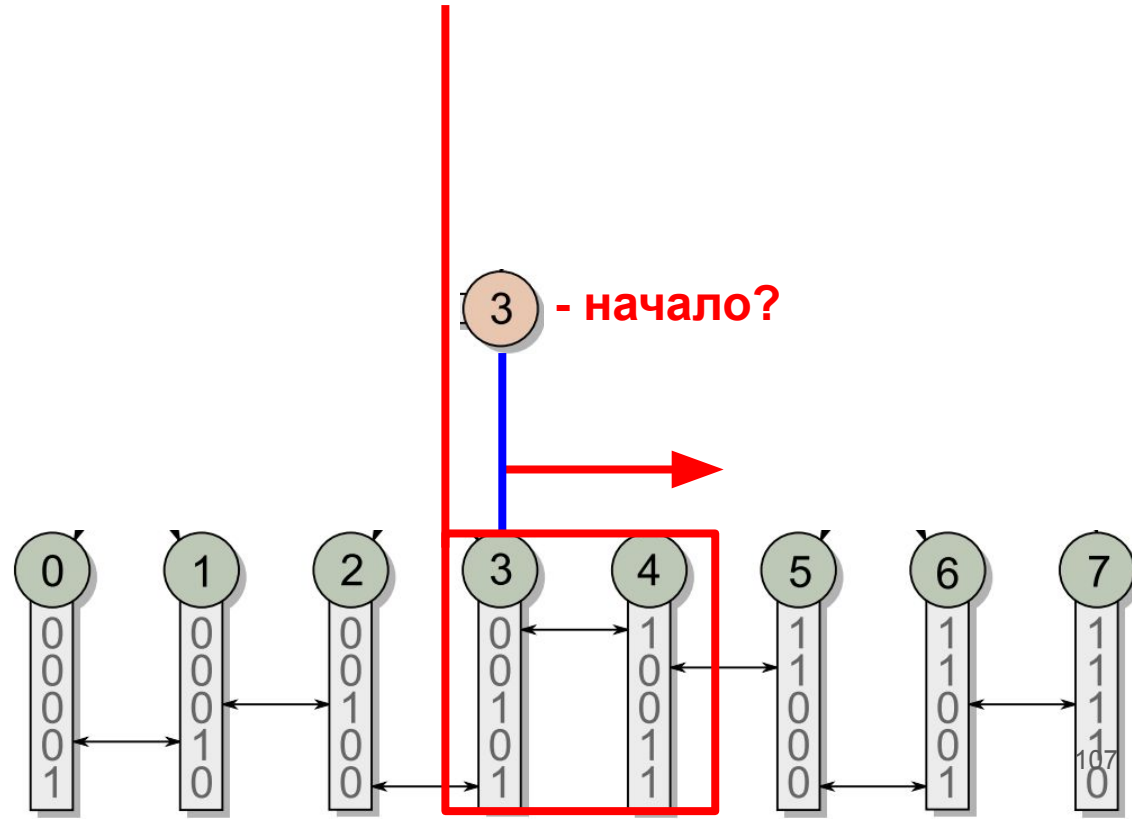
4.4) Но сначала каждый **узел** находит свой подотрезок ... **как???**



4.2) Подотрезки **детей** корня покрывают половины рассеянные по **SPLIT**

4.3) **SPLIT** - по старшему биту который различается на подотрезке - **бин.п.**

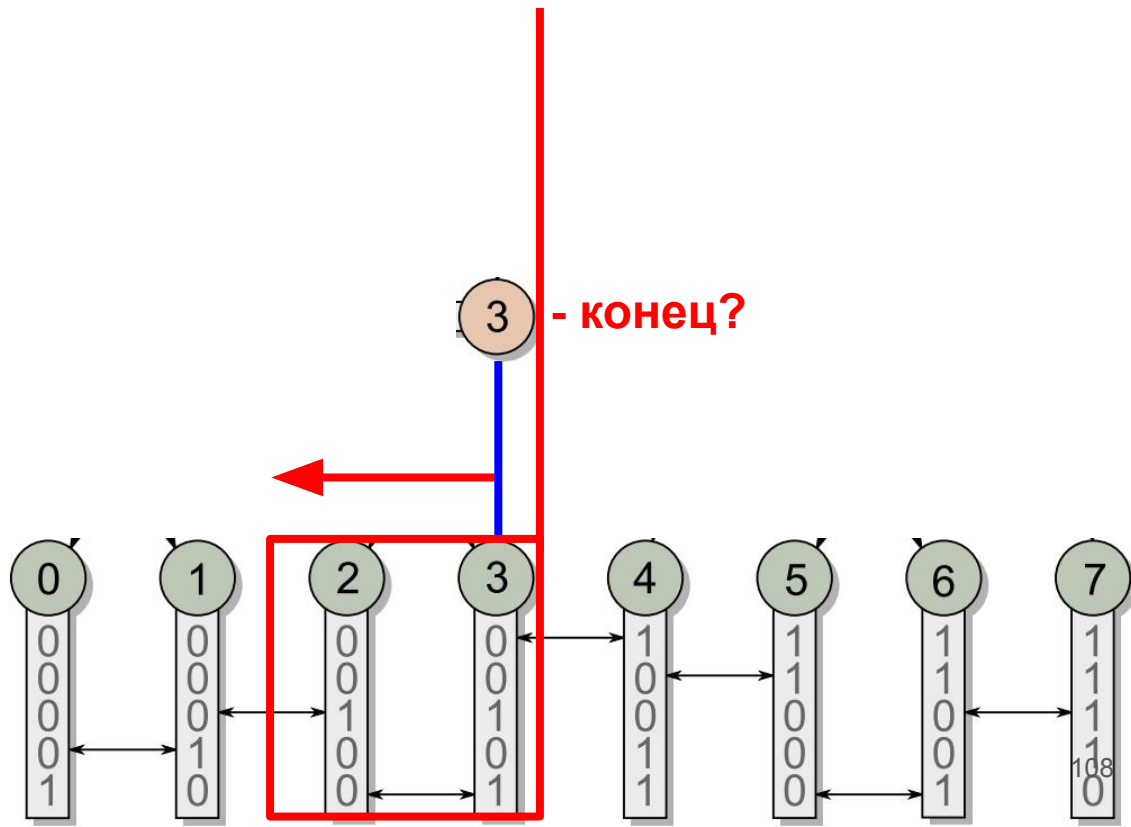
4.4) Но сначала каждый **узел** находит свой подотрезок ... **как???**



4.2) Подотрезки **детей** корня покрывают половины рассеянные по **SPLIT**

4.3) **SPLIT** - по старшему биту который различается на подотрезке - **бин.п.**

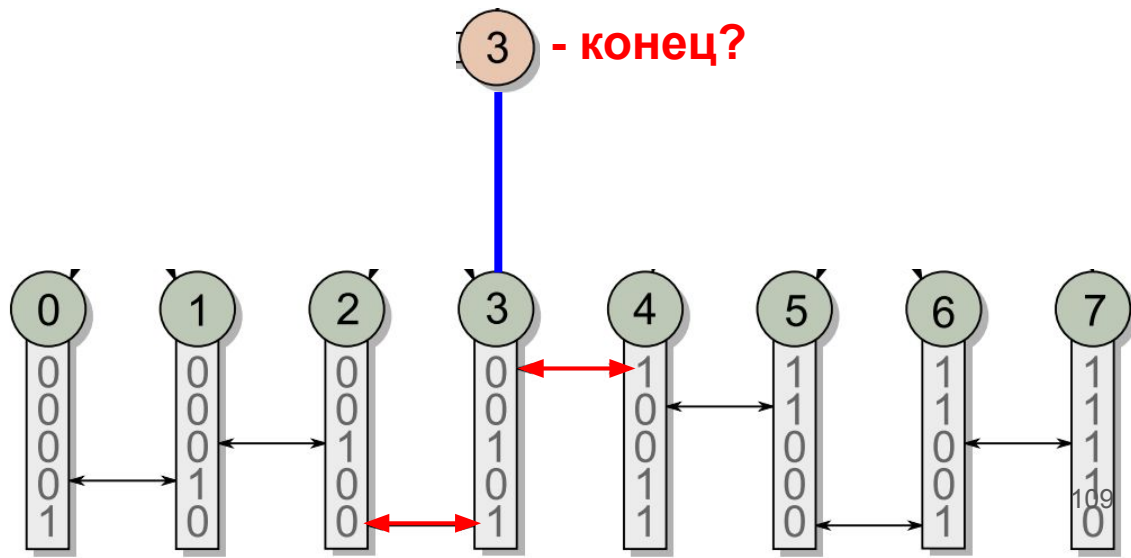
4.4) Но сначала каждый **узел** находит свой подотрезок ... **как???**



4.2) Подотрезки **детей** корня покрывают половины рассеянные по **SPLIT**

4.3) **SPLIT** - по старшему биту который различается на подотрезке - **бин.п.**

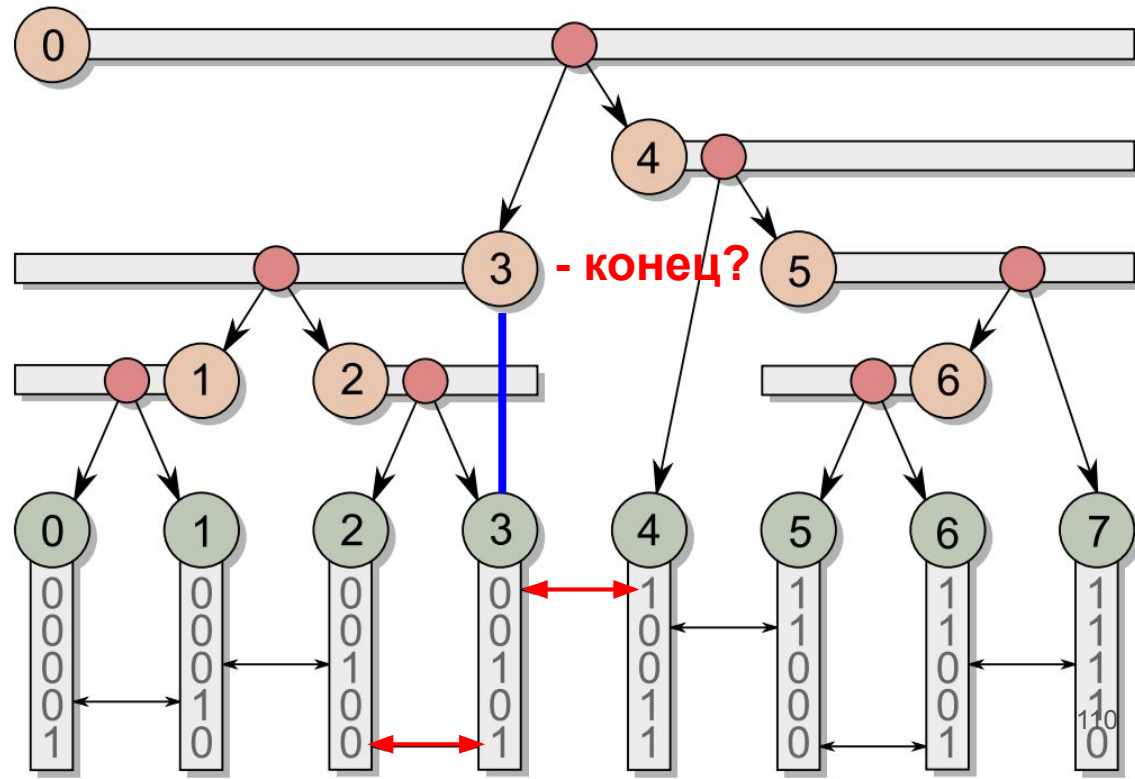
4.4) Но сначала каждый **узел** находит свой подотрезок ... **как???**



4.2) Подотрезки **детей** корня покрывают половины рассеянные по **SPLIT**

4.3) **SPLIT** - по старшему биту который различается на подотрезке - **бин.п.**

4.4) Но сначала каждый **узел** находит свой подотрезок ... **как???**

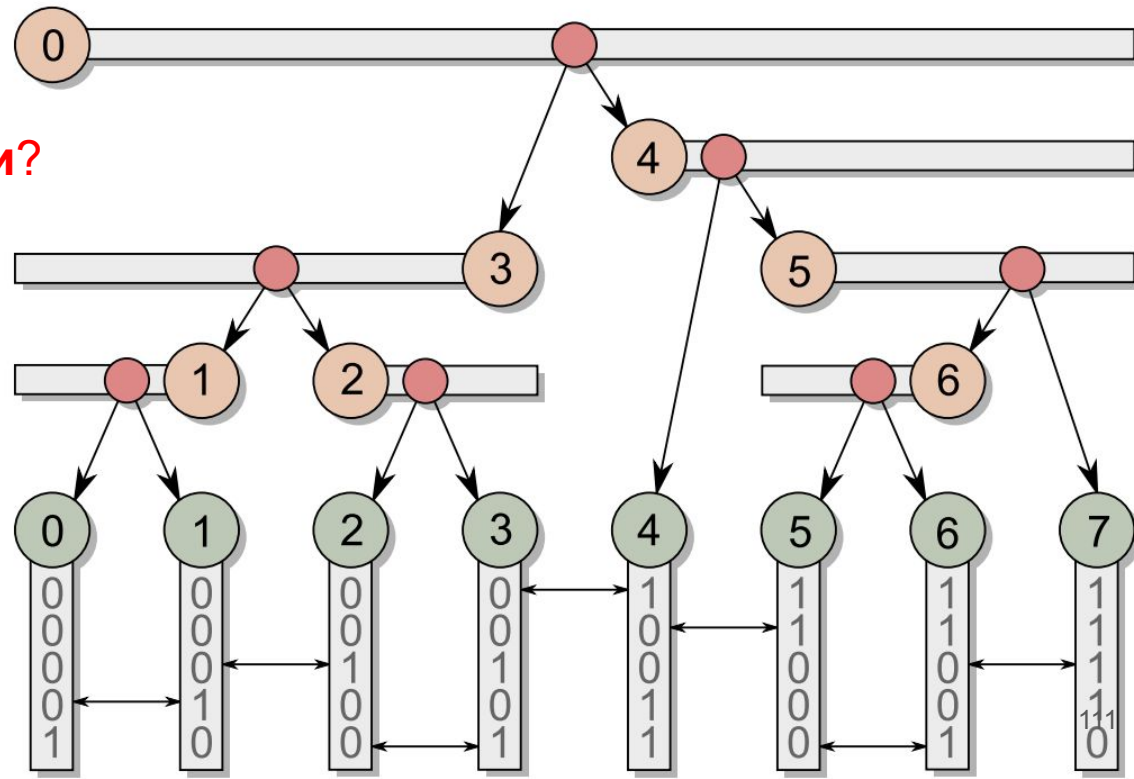


4.2) Подотрезки **детей** корня покрывают половины рассеянные по **SPLIT**

4.3) **SPLIT** - по старшему биту который различается на подотрезке - **бин.п.**

4.4) Но сначала каждый **узел** находит свой подотрезок ... **как???**

Какие **узлы** являются **концами**?

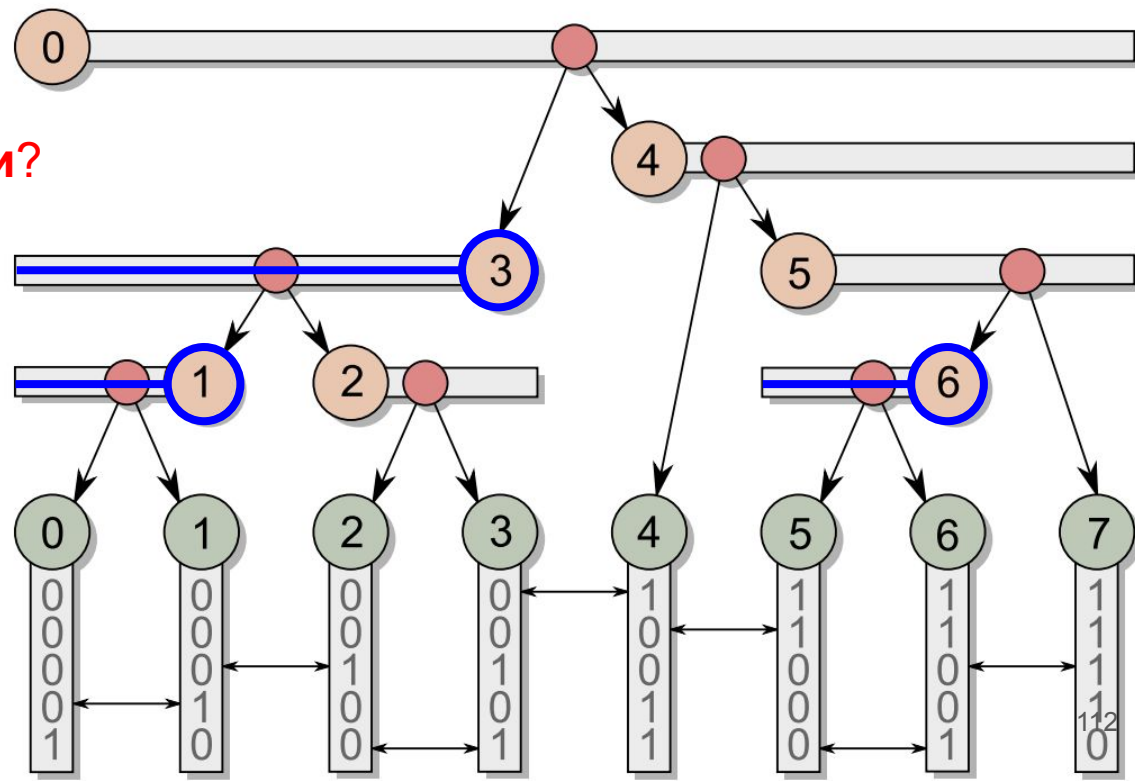


4.2) Подотрезки **детей** корня покрывают половины рассеянные по **SPLIT**

4.3) **SPLIT** - по старшему биту который различается на подотрезке - **бин.п.**

4.4) Но сначала каждый **узел** находит свой подотрезок ... **как???**

Какие **узлы** являются **концами**?

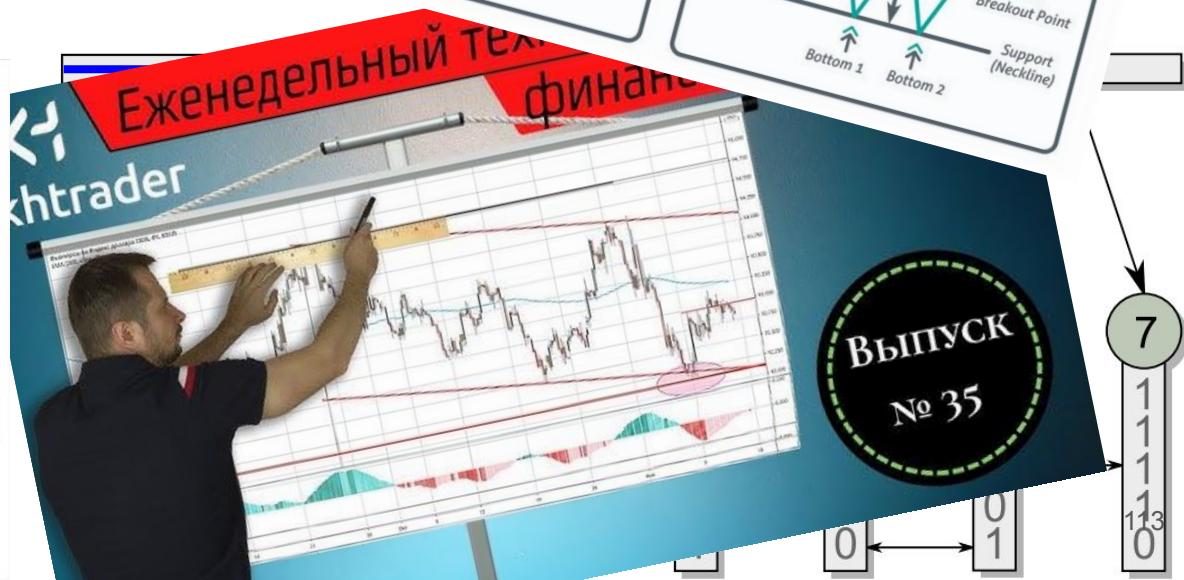
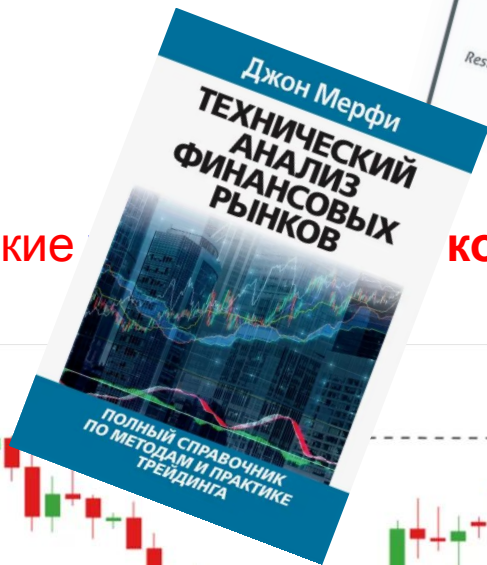
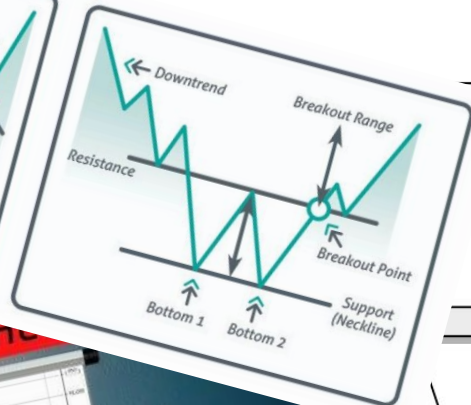


...чается на подотрезке - бин.п.

как???

Какие

концами?

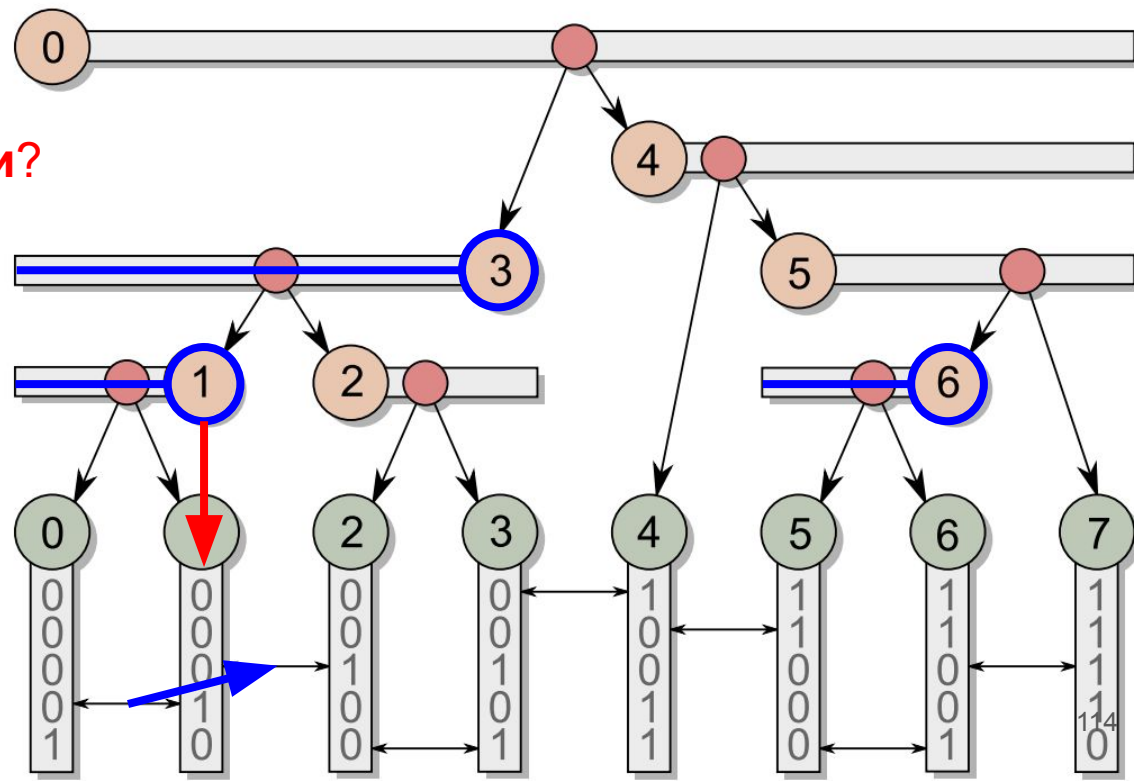


4.2) Подотрезки **детей** корня покрывают половины рассеянные по **SPLIT**

4.3) **SPLIT** - по старшему биту который различается на подотрезке - **бин.п.**

4.4) Но сначала каждый **узел** находит свой подотрезок ... **как???**

Какие **узлы** являются **концами**?

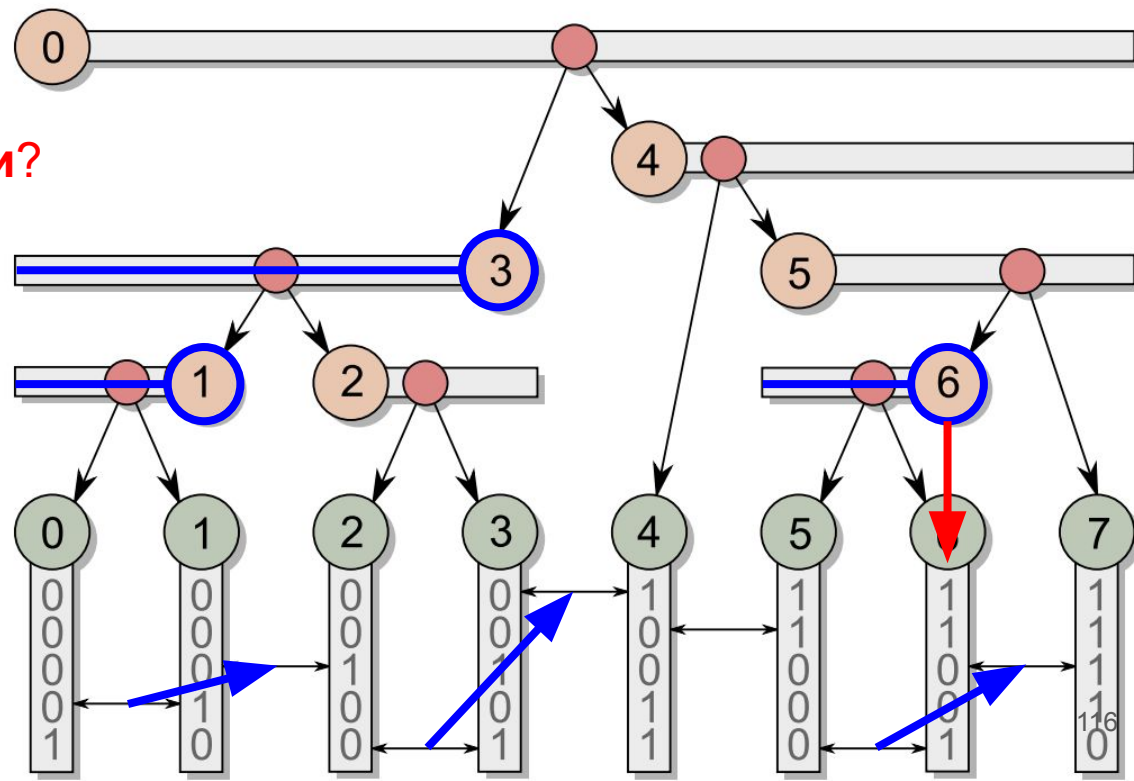


4.2) Подотрезки **детей** корня покрывают половины рассеянные по **SPLIT**

4.3) **SPLIT** - по старшему биту который различается на подотрезке - **бин.п.**

4.4) Но сначала каждый **узел** находит свой **подотрезок** ... **как???**

Какие **узлы** являются **концами**?



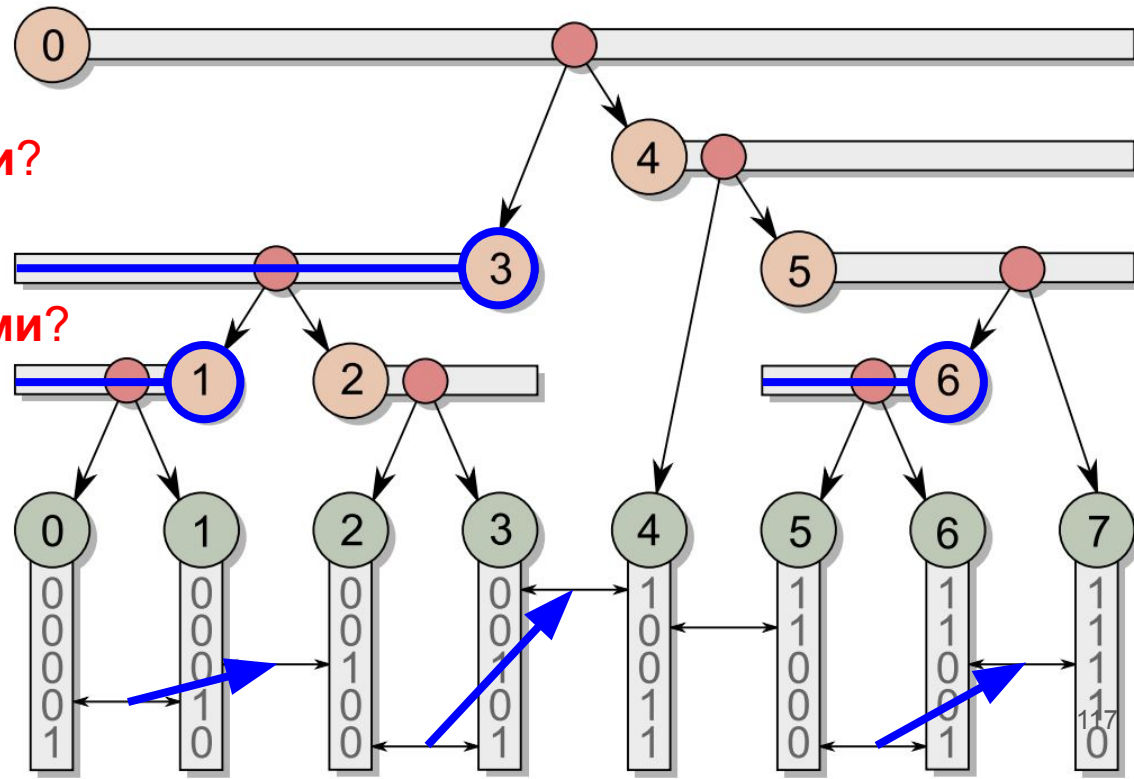
4.2) Подотрезки **детей** корня покрывают половины рассеченные по **SPLIT**

4.3) **SPLIT** - по старшему биту который различается на подотрезке - **бин.п.**

4.4) Но сначала каждый **узел** находит свой **подотрезок** ... **как???**

Какие **узлы** являются **концами**?

Какие **узлы** являются **началами**?



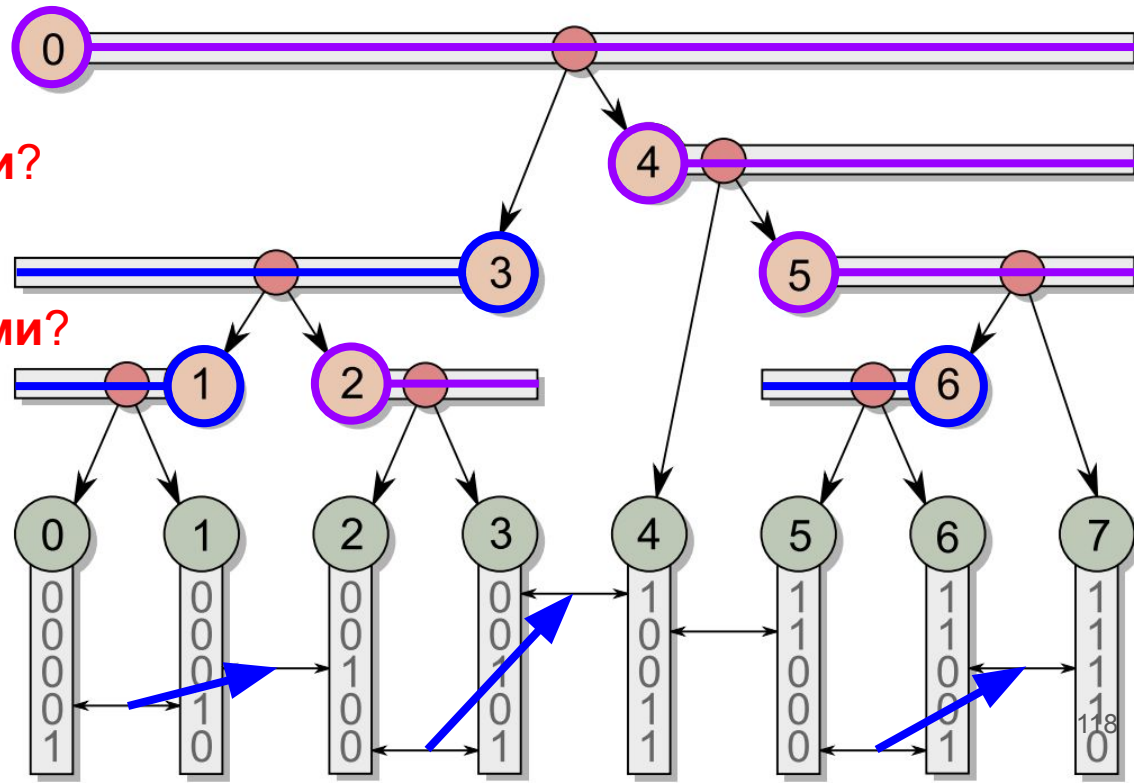
4.2) Подотрезки **детей** корня покрывают половины рассеченные по **SPLIT**

4.3) **SPLIT** - по старшему биту который различается на подотрезке - **бин.п.**

4.4) Но сначала каждый **узел** находит свой подотрезок ... **как???**

Какие **узлы** являются **концами**?

Какие **узлы** являются **началами**?



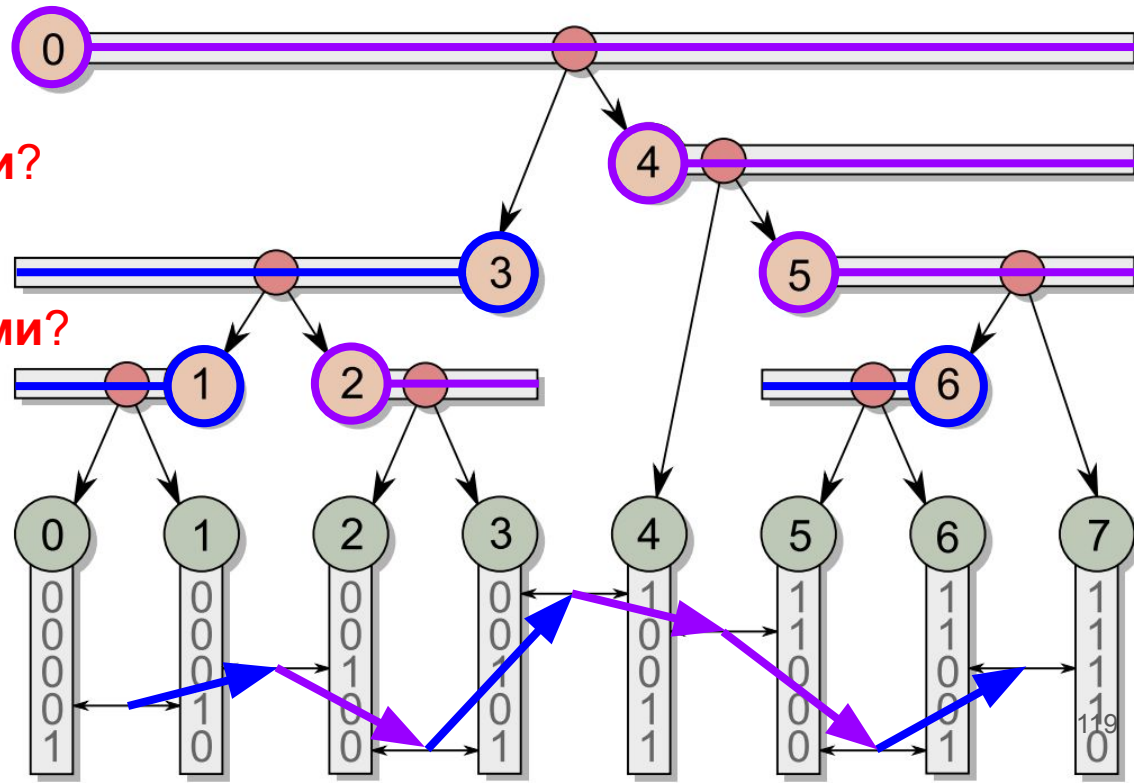
4.2) Подотрезки **детей** корня покрывают половины рассеченные по **SPLIT**

4.3) **SPLIT** - по старшему биту который различается на подотрезке - **бин.п.**

4.4) Но сначала каждый **узел** находит свой **подотрезок** ... **как???**

Какие **узлы** являются **концами**?

Какие **узлы** являются **началами**?



4.2) Подотрезки **детей** корня покрывают половины рассеченные по **SPLIT**

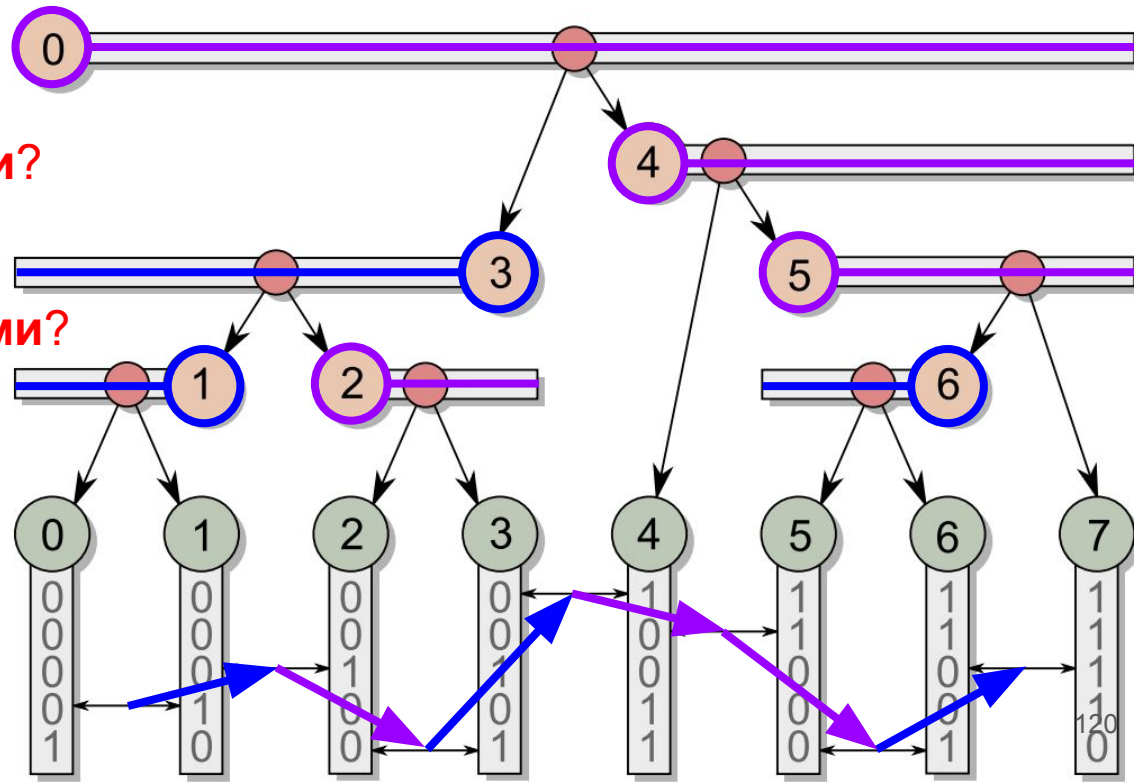
4.3) **SPLIT** - по старшему биту который различается на подотрезке - **бин.п.**

4.4) Но сначала каждый **узел** находит свой подотрезок ... **как???**

Какие **узлы** являются **концами**?

Какие **узлы** являются **началами**?

Почему так?



4.2) Подотрезки **детей** корня покрывают половины рассеянные по **SPLIT**

4.3) **SPLIT** - по старшему биту который различается на подотрезке - **бин.п.**

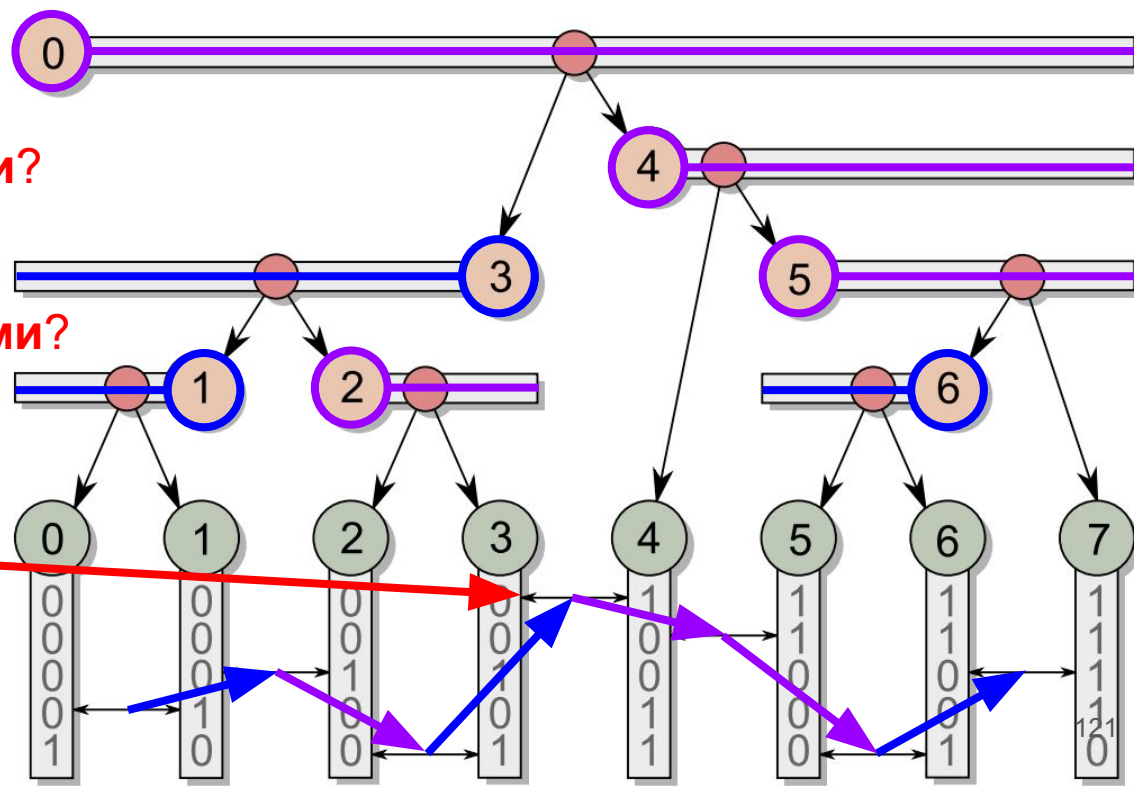
4.4) Но сначала каждый **узел** находит свой подотрезок ... **как???**

Какие **узлы** являются **концами**?

Какие **узлы** являются **началами**?

Почему так?

Почему именно тут - пик?



4.2) Подотрезки **детей** корня покрывают половины рассеченные по **SPLIT**

4.3) **SPLIT** - по старшему биту который различается на подотрезке - **бин.п.**

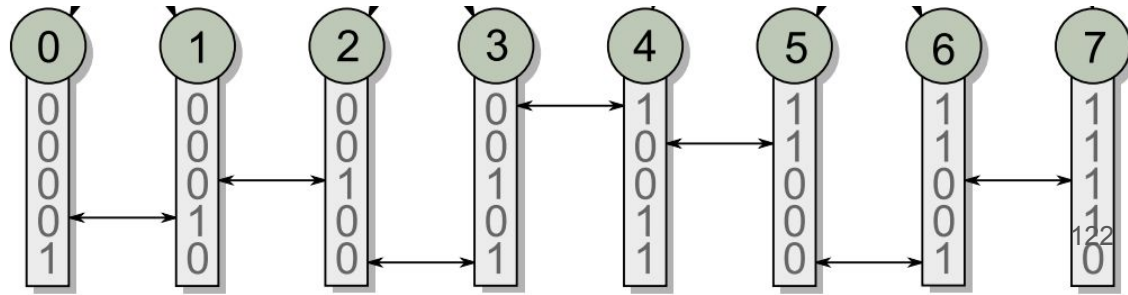
4.4) Но сначала каждый **узел** находит свой подотрезок ... **как???**

Какие **узлы** являются **концами**?



Какие **узлы** являются **началами**?

Почему так?

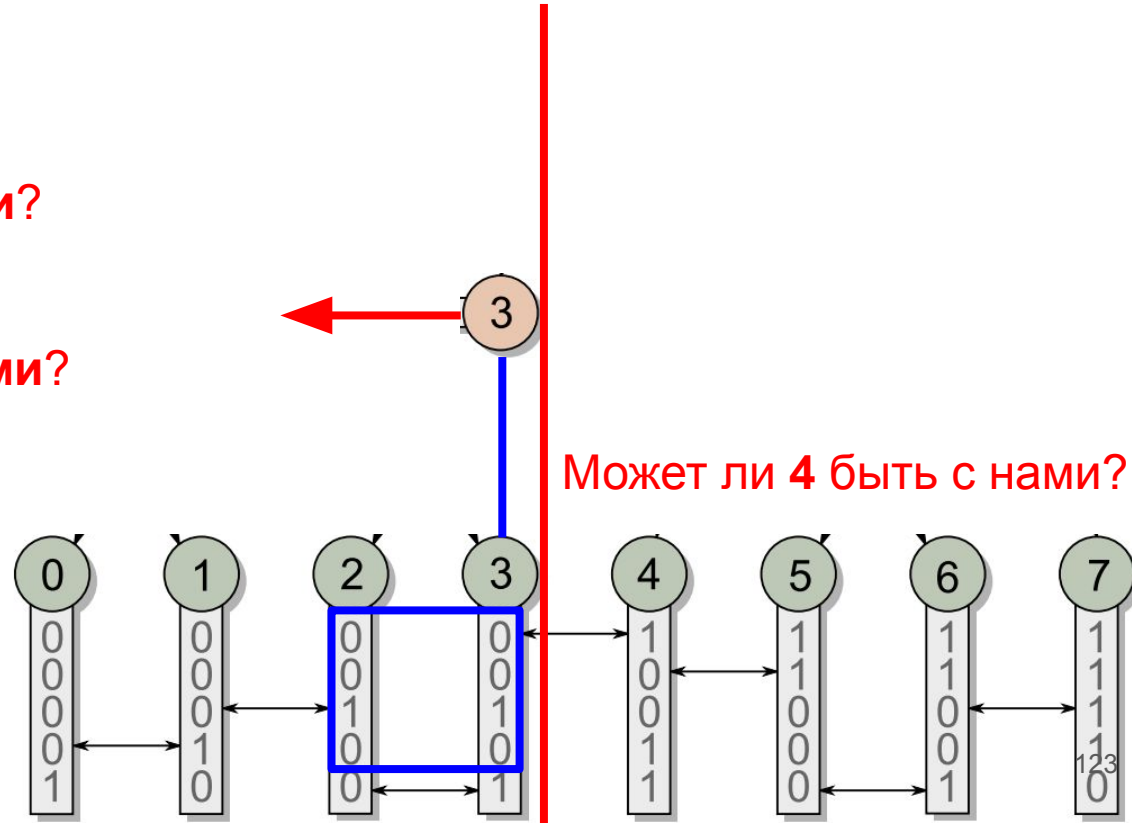


- 4.2) Подотрезки **детей** корня покрывают половины рассеченные по **SPLIT**
- 4.3) **SPLIT** - по старшему биту который различается на подотрезке - **бин.п.**
- 4.4) Но сначала каждый **узел** находит свой подотрезок ... **как???**

Какие **узлы** являются **концами**?

Какие **узлы** являются **началами**?

Почему так?



4.2) Подотрезки **детей** корня покрывают половины рассеченные по **SPLIT**

4.3) **SPLIT** - по старшему биту который различается на подотрезке - **бин.п.**

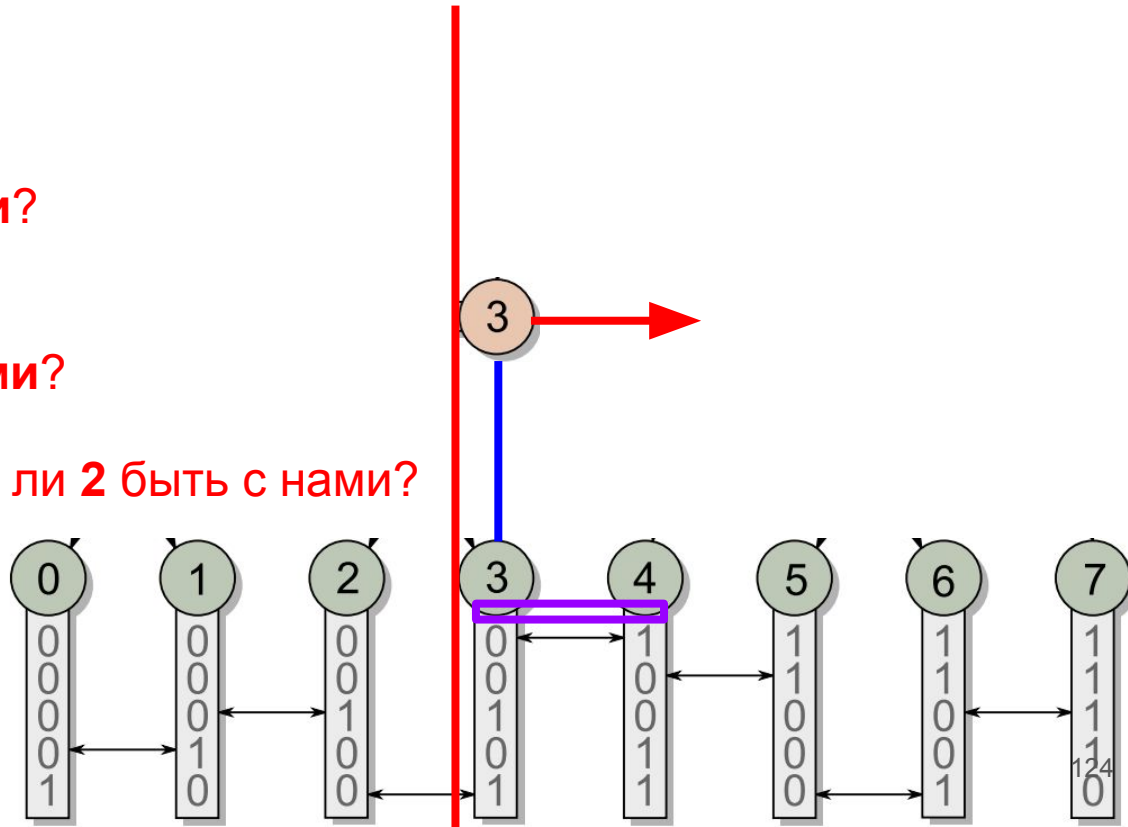
4.4) Но сначала каждый **узел** находит свой подотрезок ... **как???**

Какие **узлы** являются **концами**?

Какие **узлы** являются **началами**?

Почему так?

Может ли **2** быть с нами?



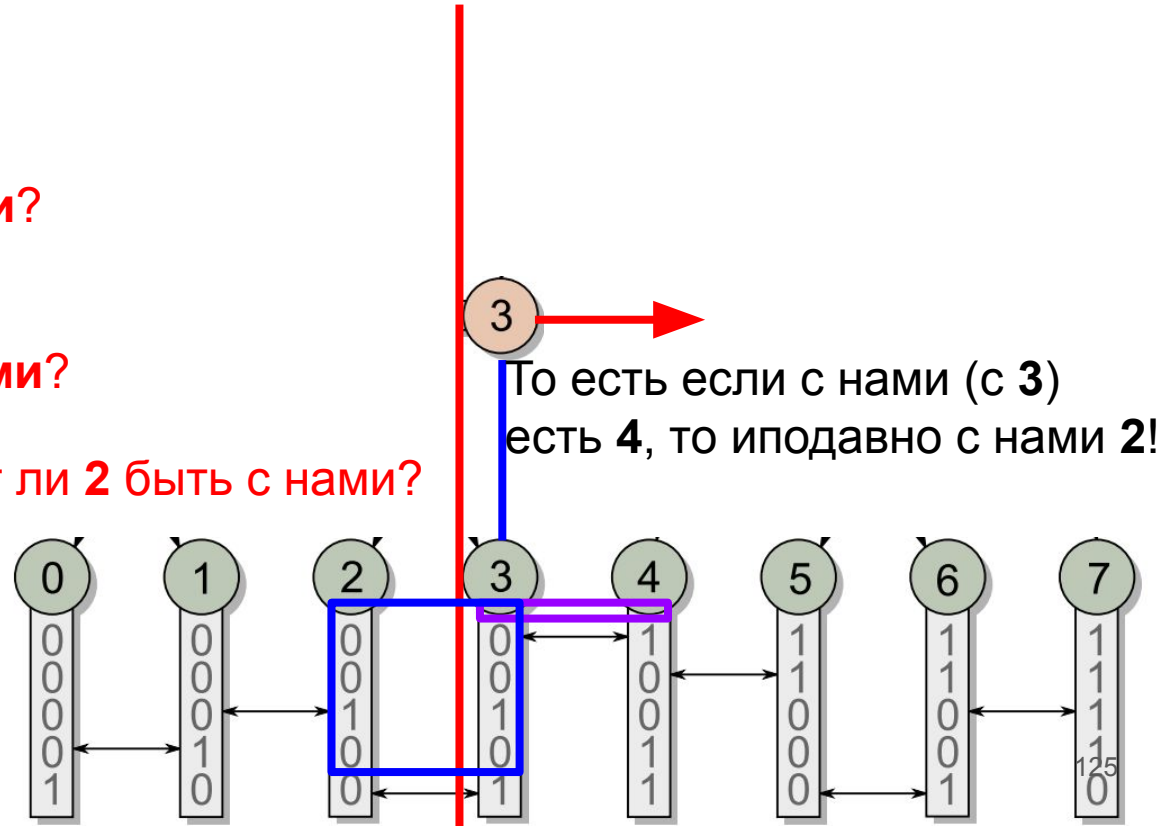
- 4.2) Подотрезки **детей** корня покрывают половины рассеянные по **SPLIT**
- 4.3) **SPLIT** - по старшему биту который различается на подотрезке - **бин.п.**
- 4.4) Но сначала каждый **узел** находит свой подотрезок ... **как???**

Какие **узлы** являются **концами**?

Какие **узлы** являются **началами**?

Почему так?

Может ли **2** быть с нами?

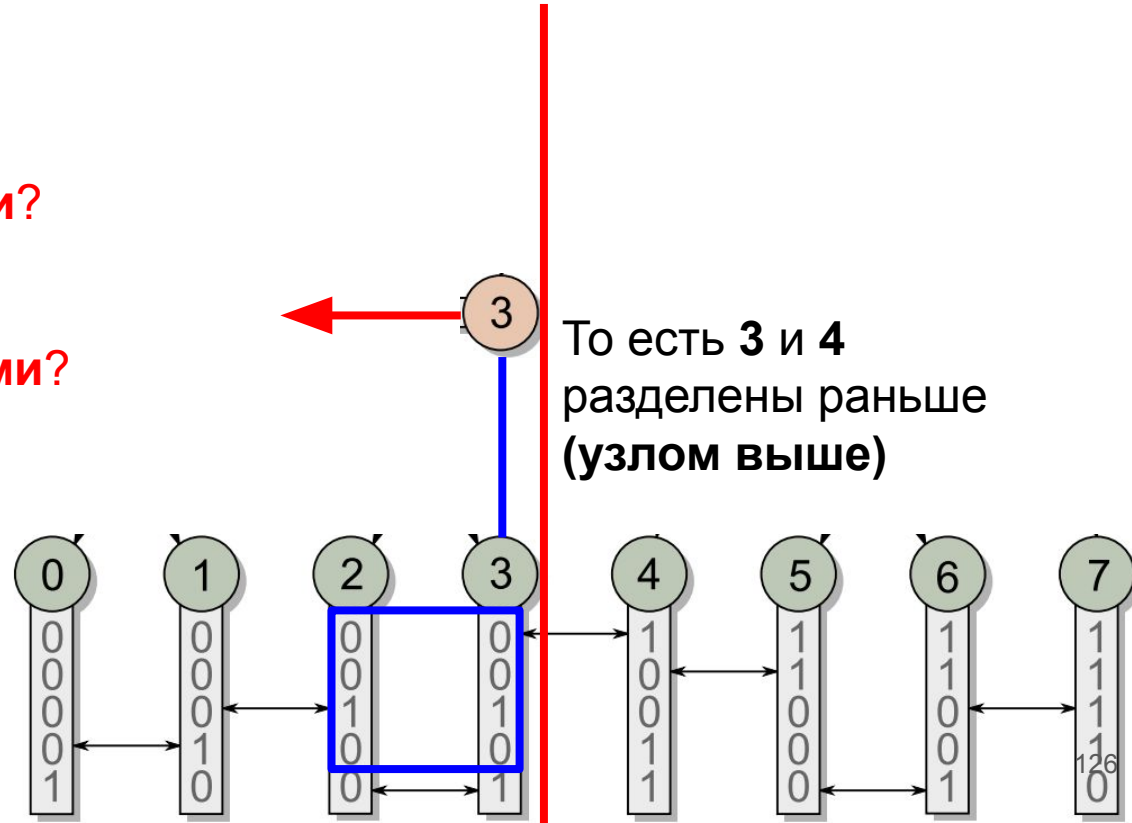


- 4.2) Подотрезки **детей** корня покрывают половины рассеянные по **SPLIT**
- 4.3) **SPLIT** - по старшему биту который различается на подотрезке - **бин.п.**
- 4.4) Но сначала каждый **узел** находит свой подотрезок ... **как???**

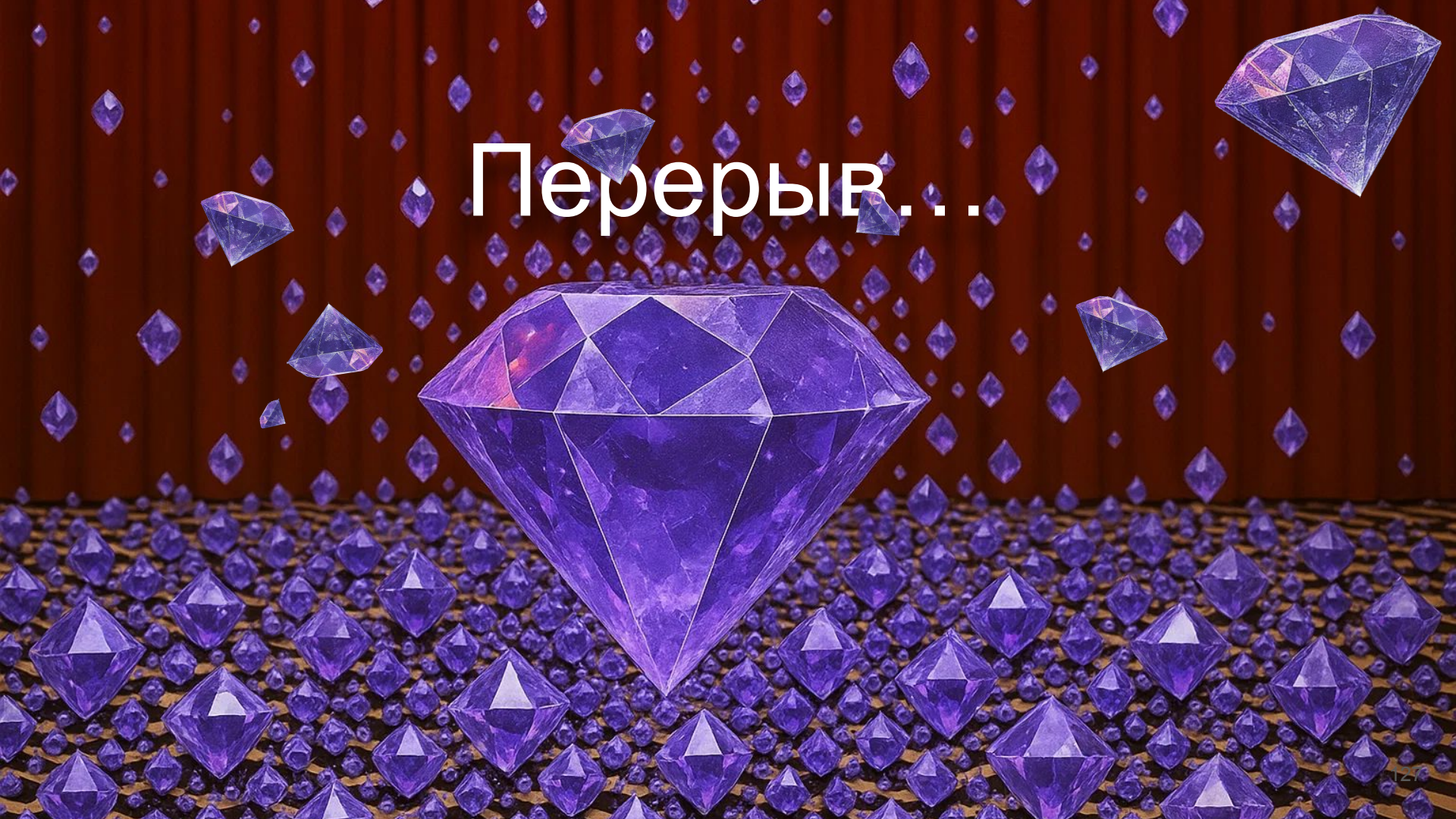
Какие **узлы** являются **концами**?

Какие **узлы** являются **началами**?

Почему так?



Перерыв...

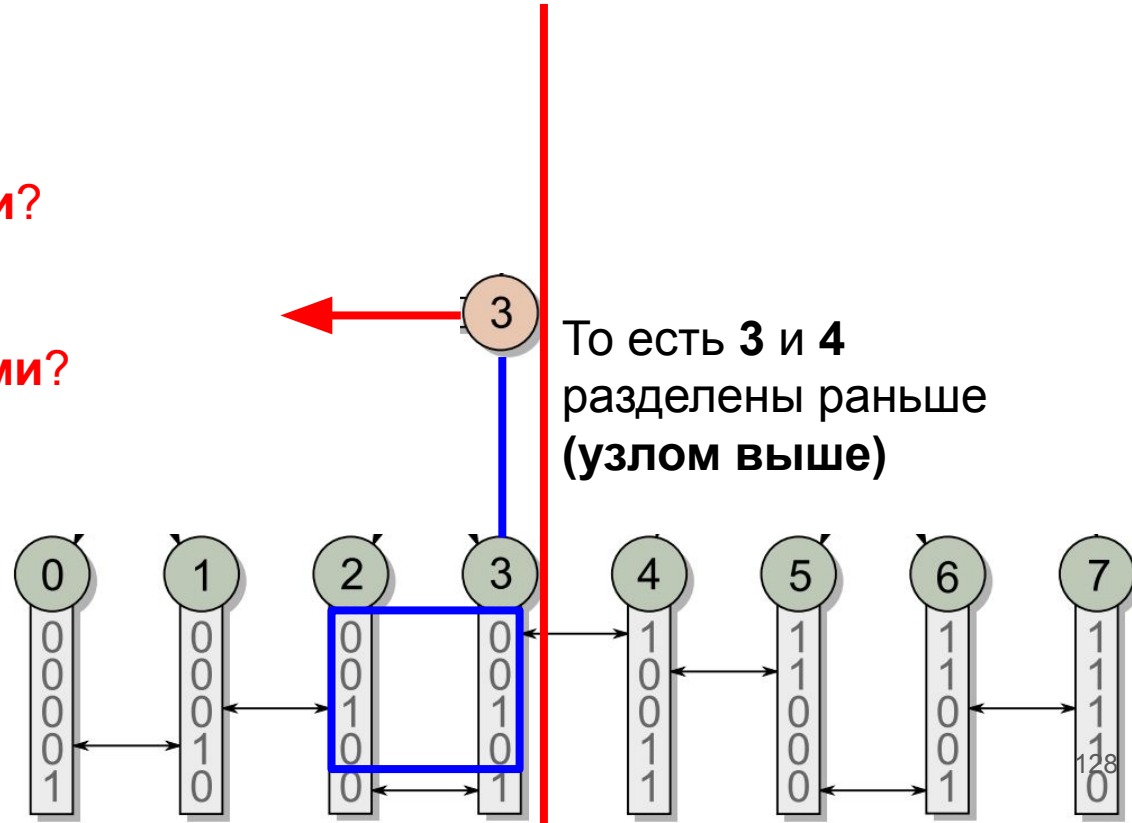


- 4.2) Подотрезки **детей** корня покрывают половины рассеянные по **SPLIT**
- 4.3) **SPLIT** - по старшему биту который различается на подотрезке - **бин.п.**
- 4.4) Но сначала каждый **узел** находит свой подотрезок ... **как???**

Какие **узлы** являются **концами**?

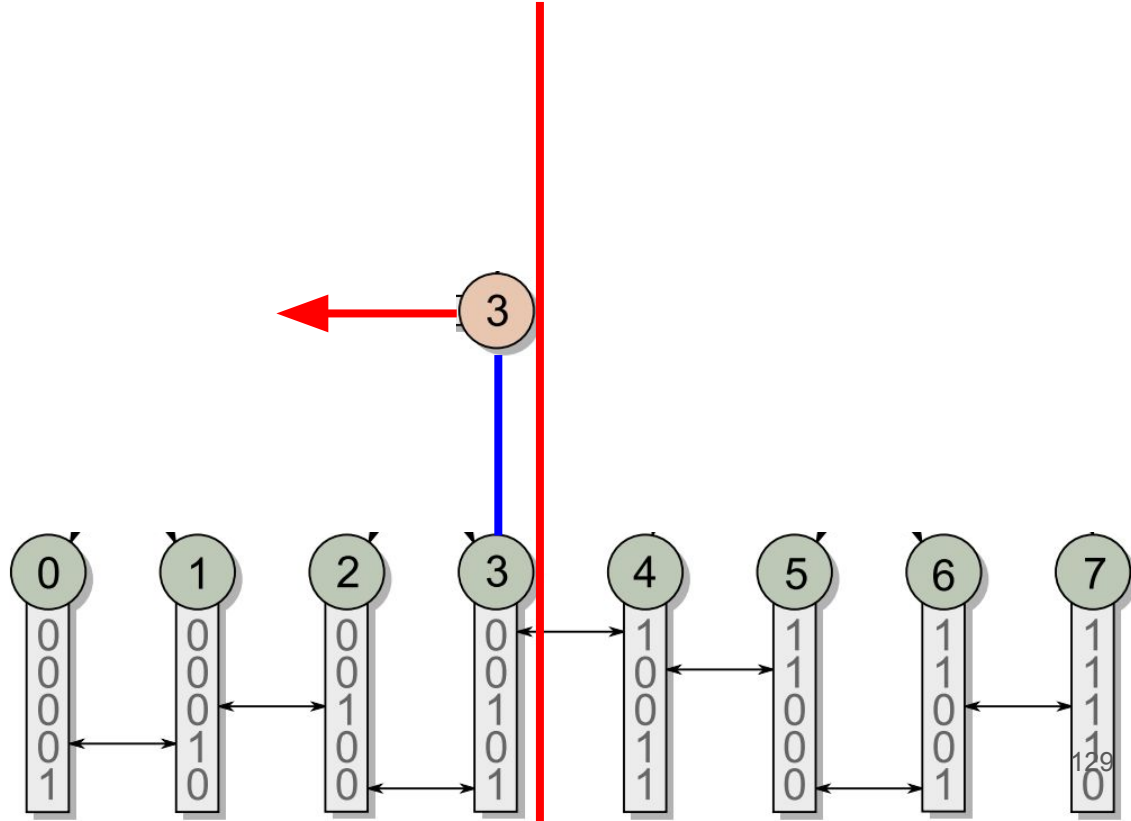
Какие **узлы** являются **началами**?

Почему так?



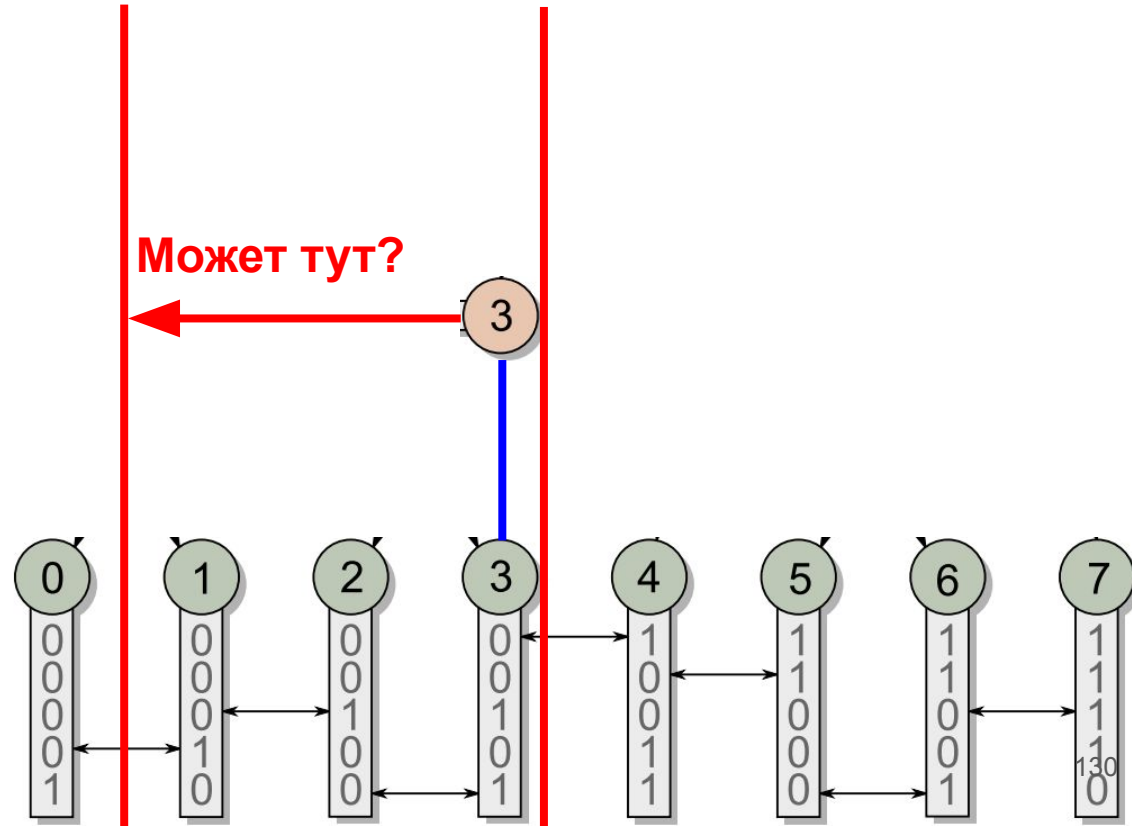
4.4) Но сначала каждый **узел** находит направление **подотрезка**

4.5) Как найти **конец** подотрезка?

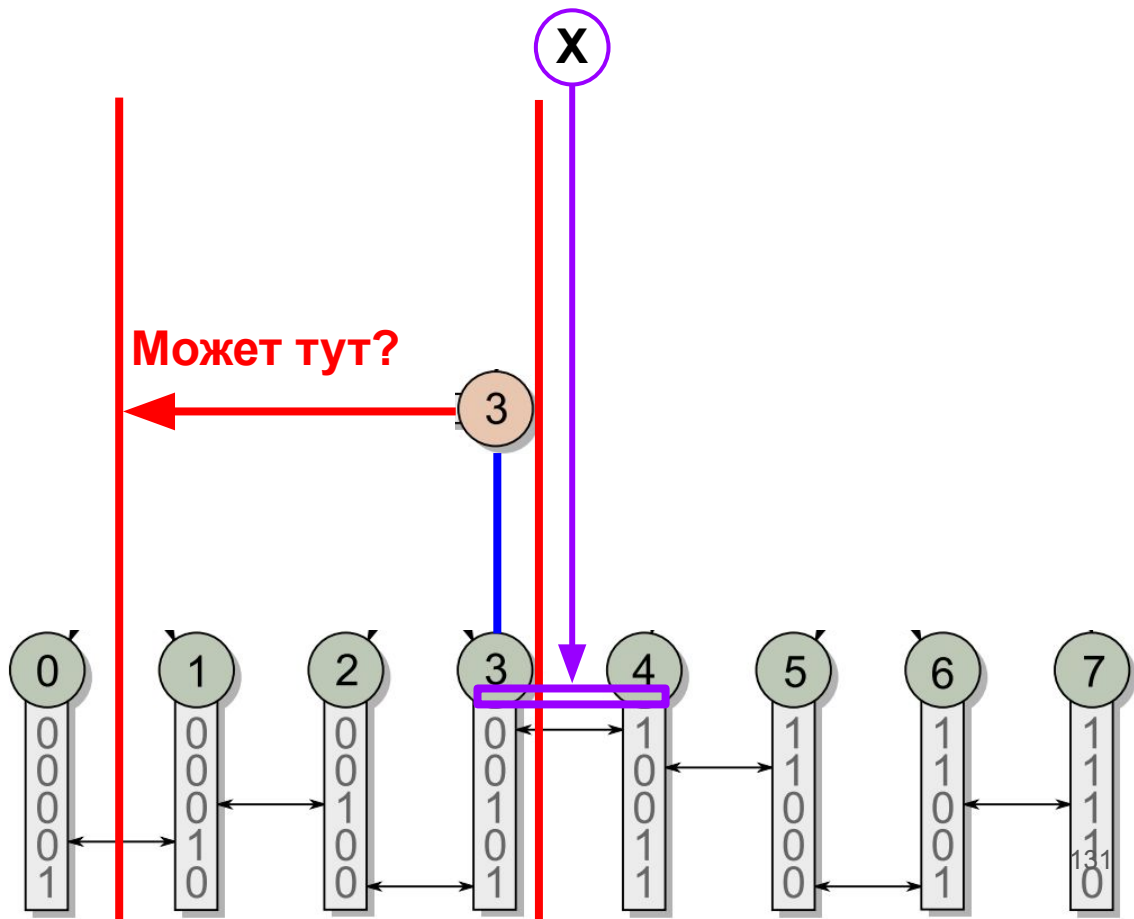


4.4) Но сначала каждый **узел** находит направление **подотрезка**

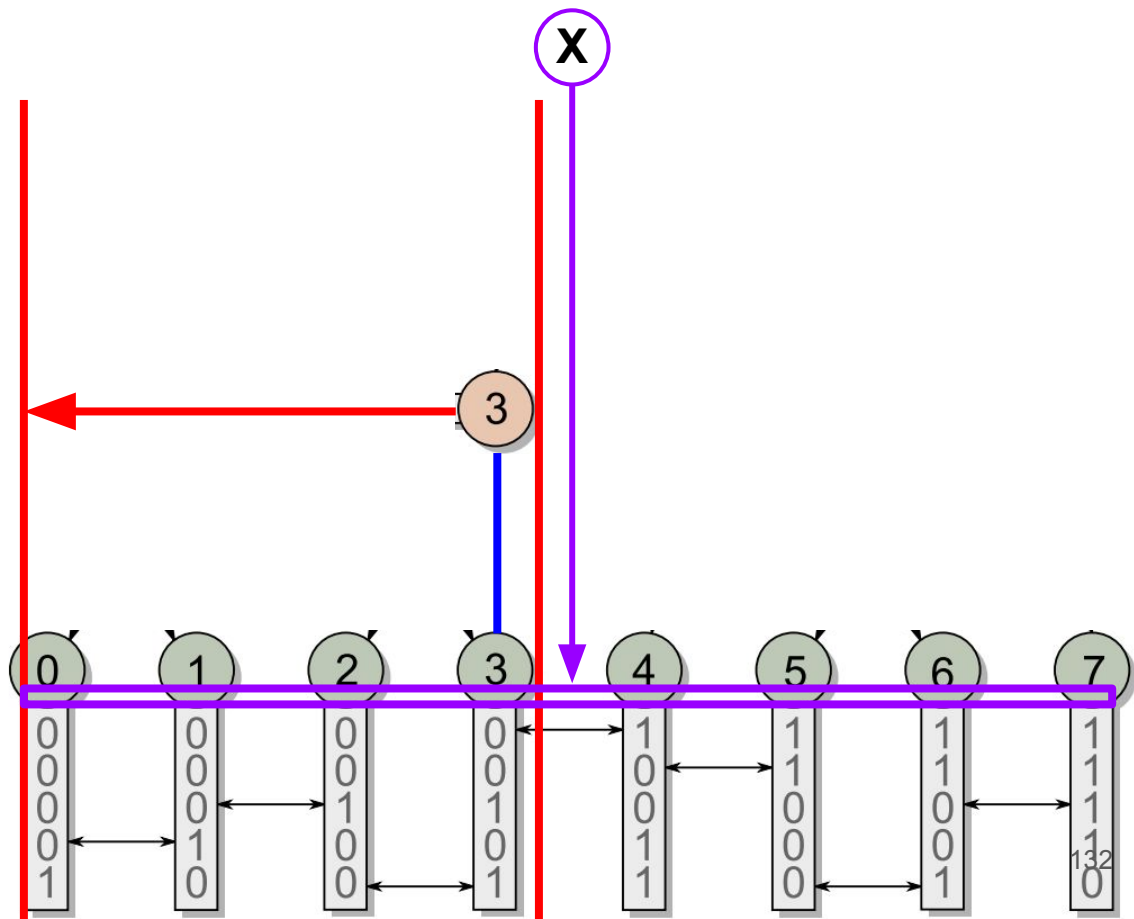
4.5) Как найти **конец** подотрезка?



- 4.4) Но сначала каждый **узел** находит направление **подотрезка**
- 4.5) Как найти **конец** подотрезка?

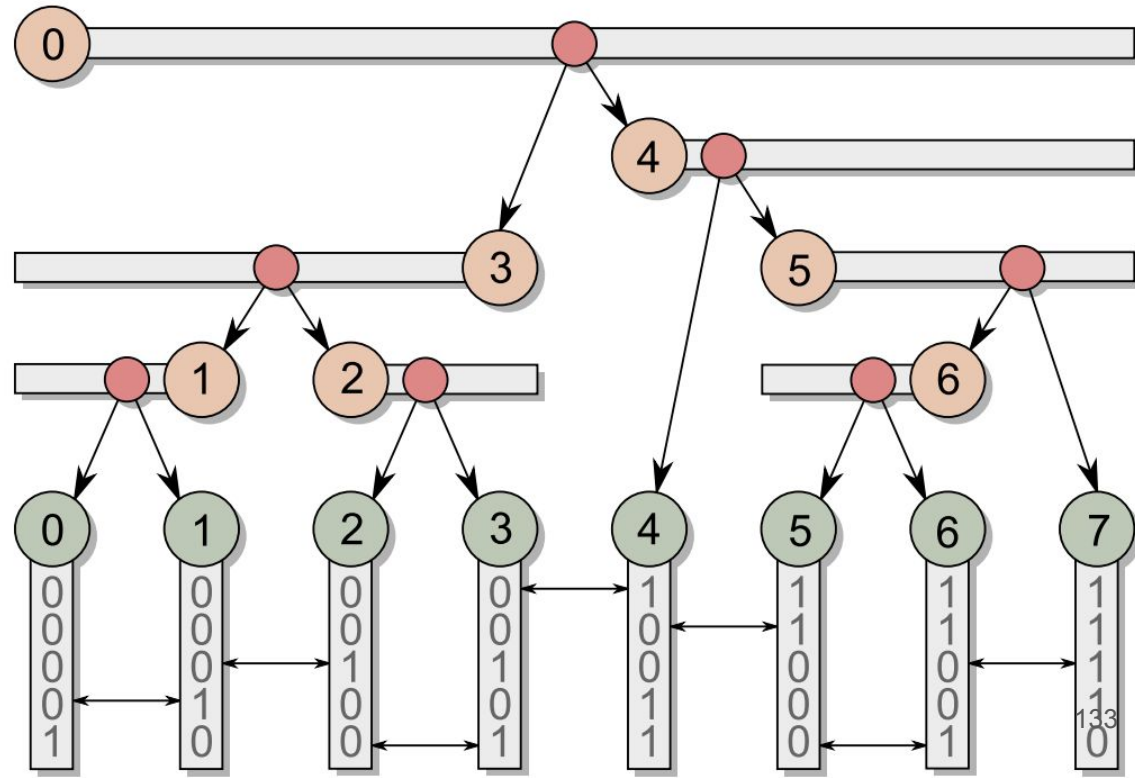


- 4.4) Но сначала каждый **узел** находит направление **подотрезка**
- 4.5) Как найти **конец** подотрезка?

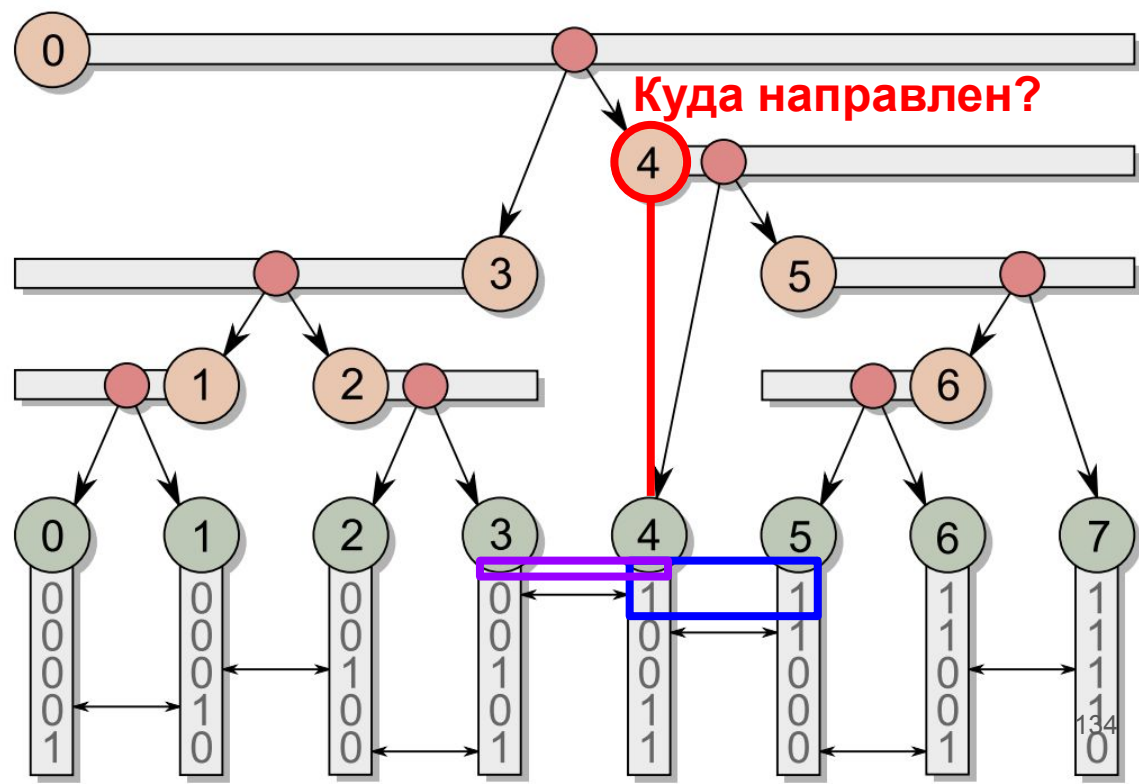


4.4) Но сначала каждый **узел** находит направление **подотрезка**

4.5) Как найти **конец** подотрезка?



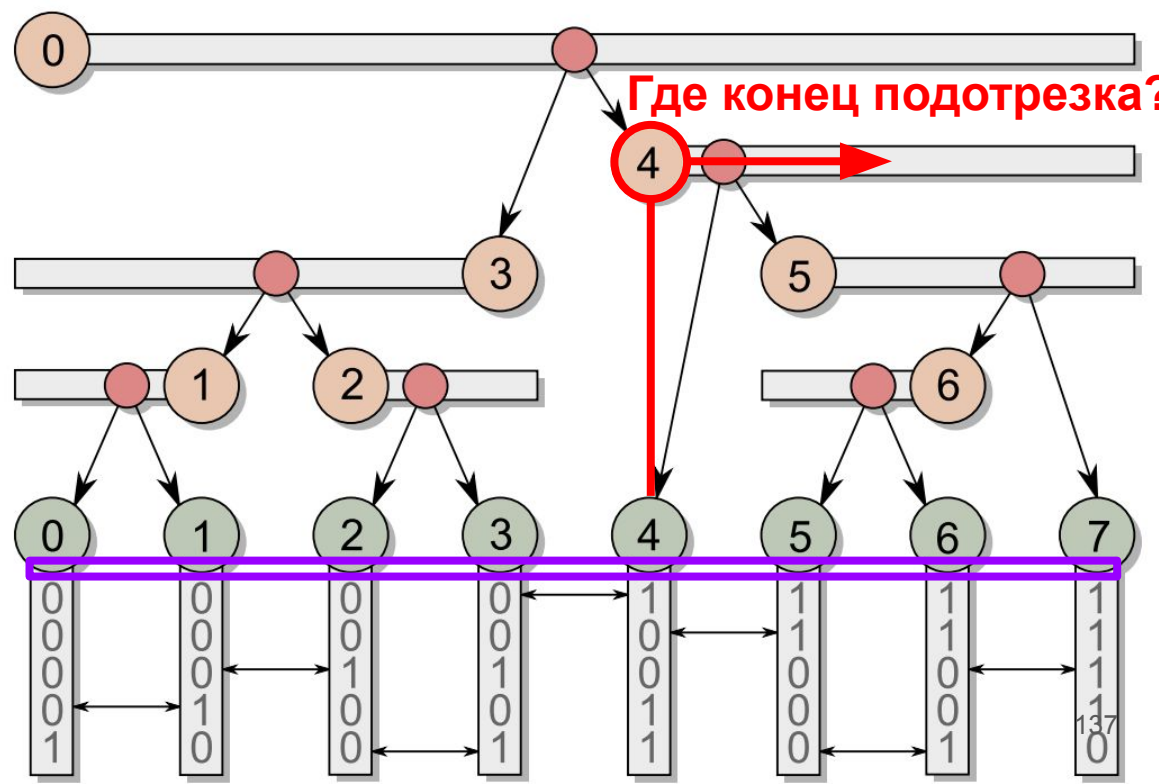
- 4.4) Но сначала каждый **узел** находит направление **подотрезка**
- 4.5) Как найти **конец** подотрезка?



4.5) Как найти **конец** подотрезка?

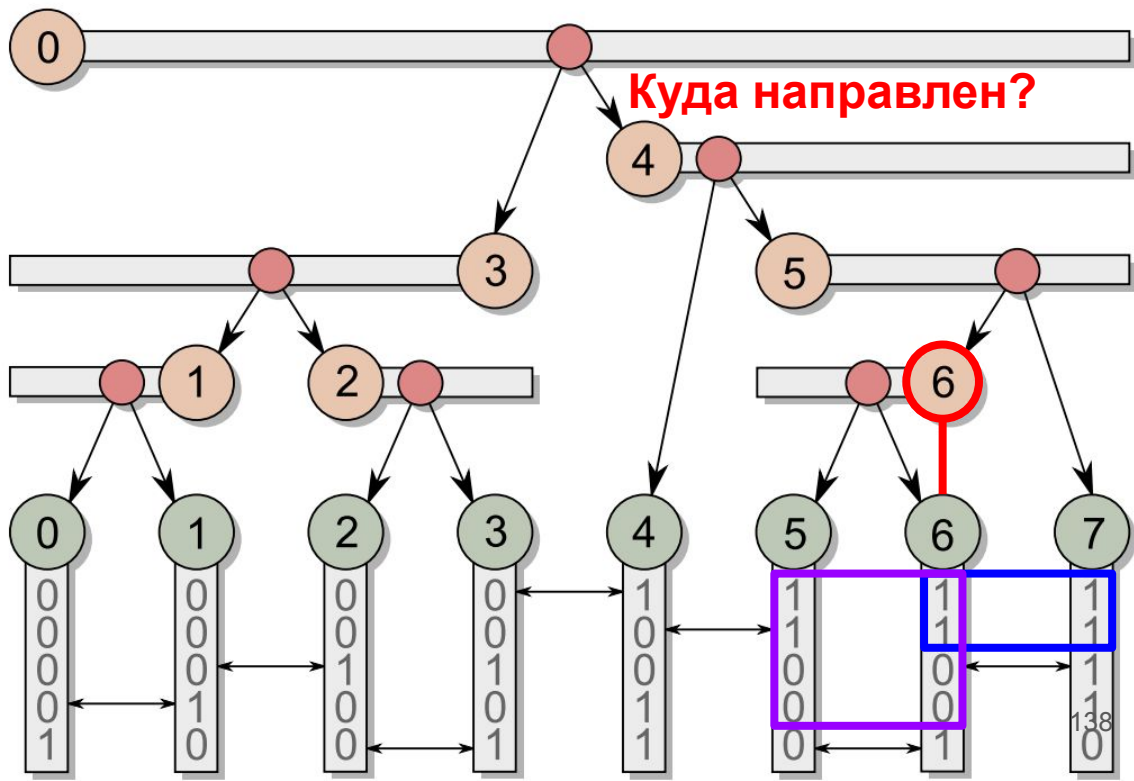
4.5) Как найти **конец** подотрезка?

- 4.4) Но сначала каждый **узел** находит направление **подотрезка**
- 4.5) Как найти **конец** подотрезка?



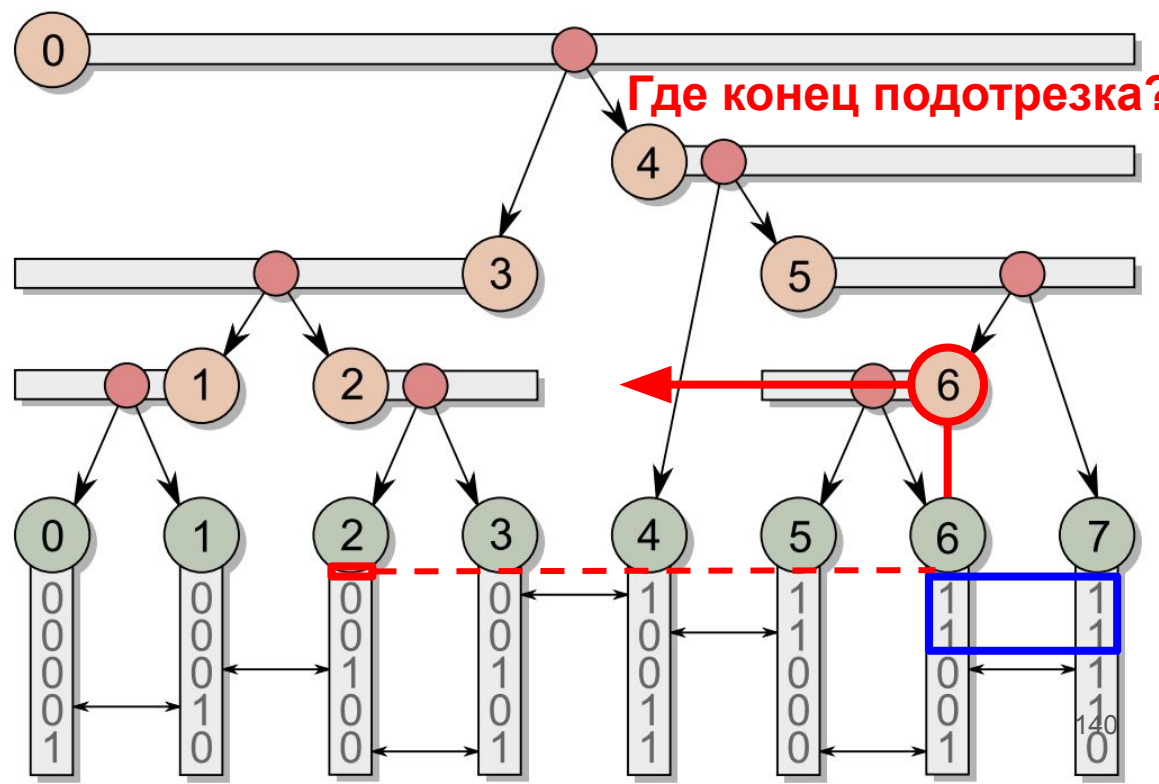
4.4) Но сначала каждый **узел** находит направление **подотрезка**

4.5) Как найти **конец** подотрезка?



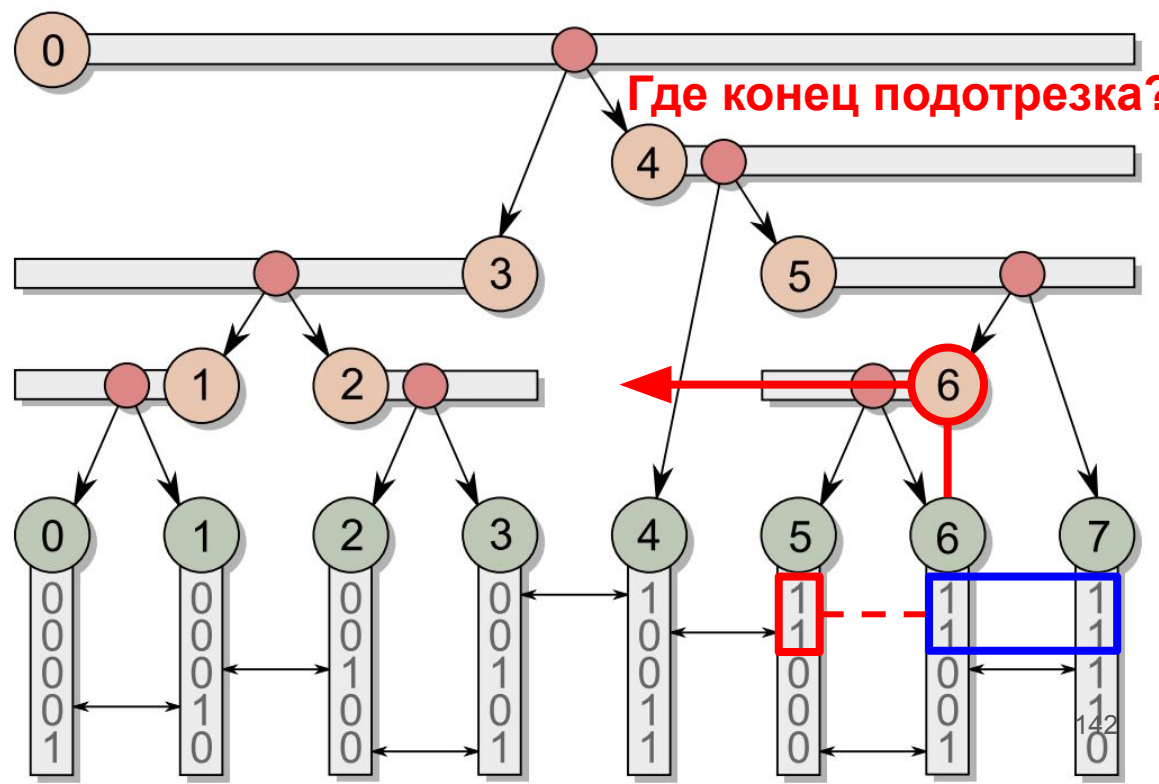
4.5) Как найти **конец** подотрезка?

- 4.4) Но сначала каждый **узел** находит направление **подотрезка**
- 4.5) Как найти **конец** подотрезка?



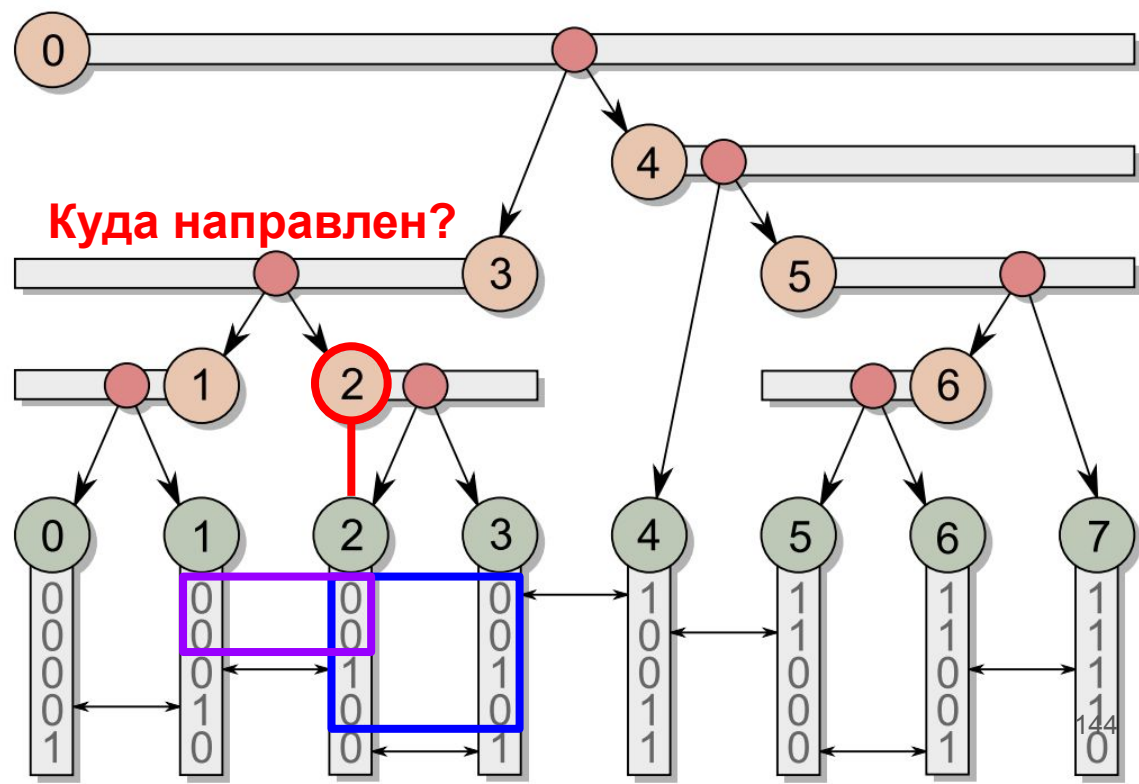
4.5) Как найти **конец** подотрезка?

- 4.4) Но сначала каждый **узел** находит направление **подотрезка**
- 4.5) Как найти **конец** подотрезка?



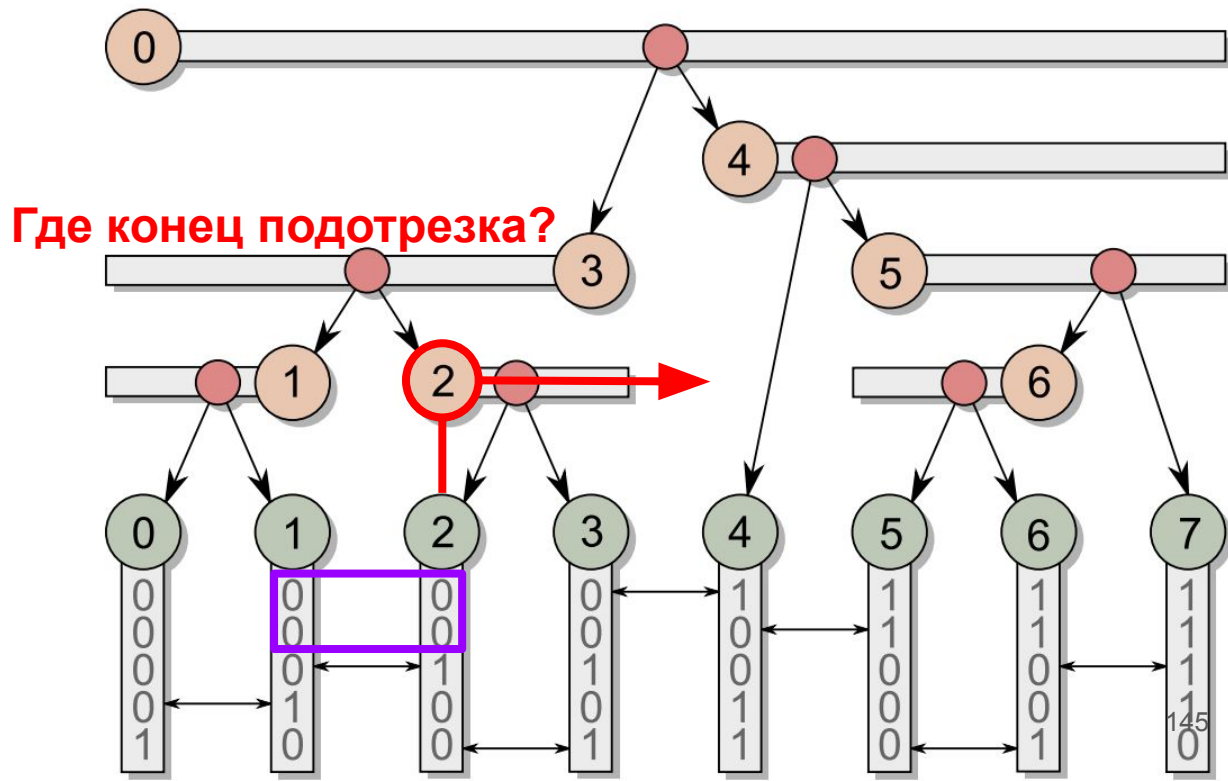
4.5) Как найти **конец** подотрезка?

- 4.4) Но сначала каждый **узел** находит направление **подотрезка**
- 4.5) Как найти **конец** подотрезка?



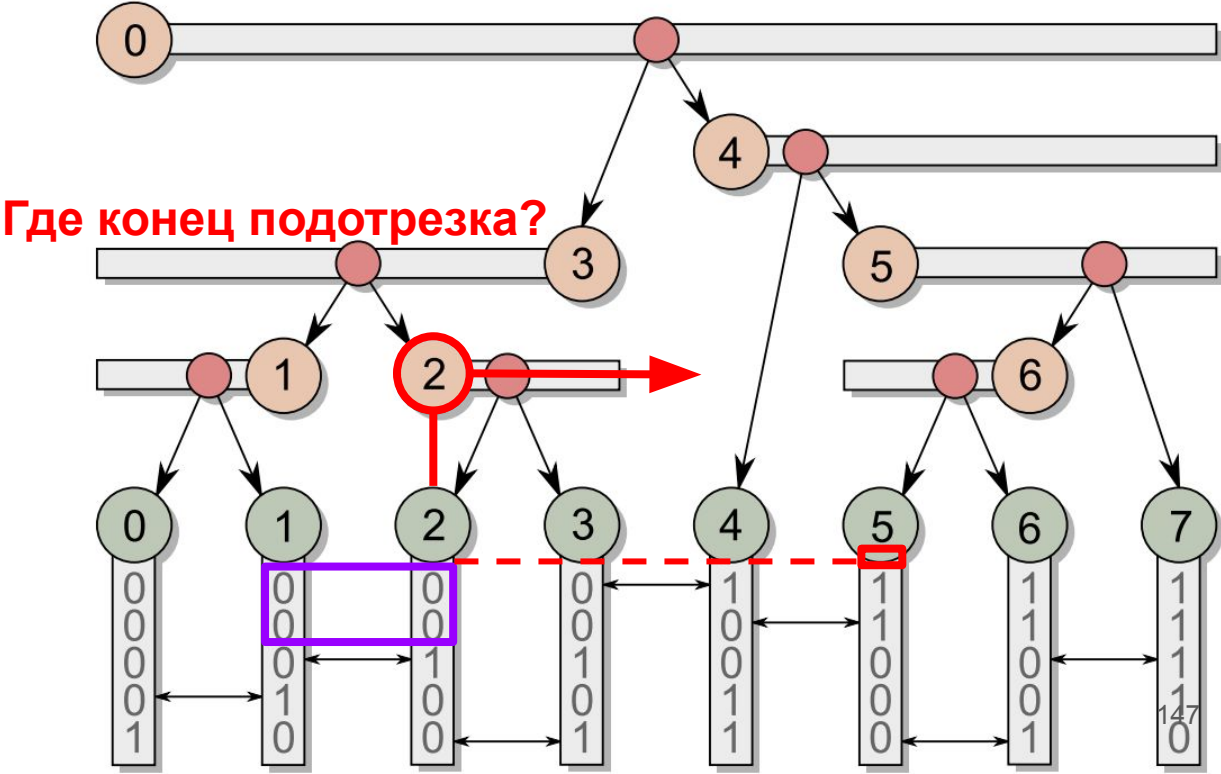
4.4) Но сначала каждый **узел** находит направление **подотрезка**

4.5) Как найти **конец** подотрезка?

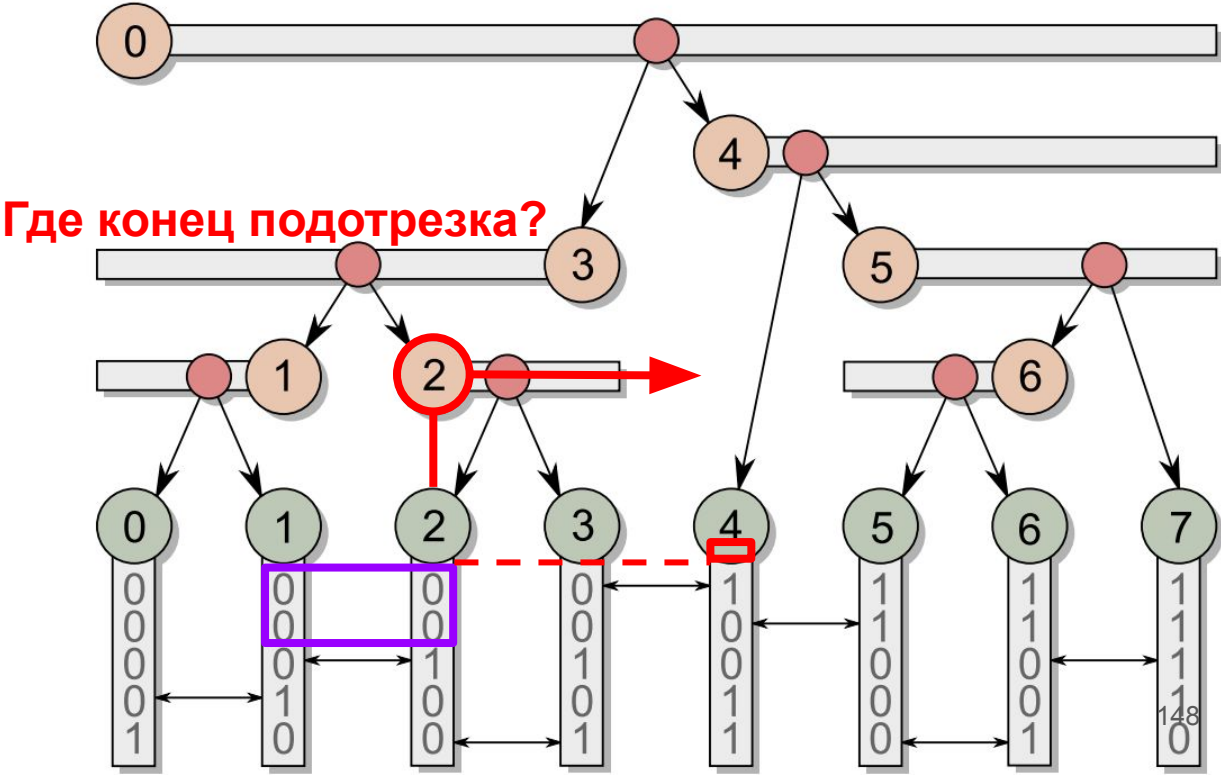


4.5) Как найти **конец** подотрезка?

- 4.4) Но сначала каждый **узел** находит направление **подотрезка**
- 4.5) Как найти **конец** подотрезка?

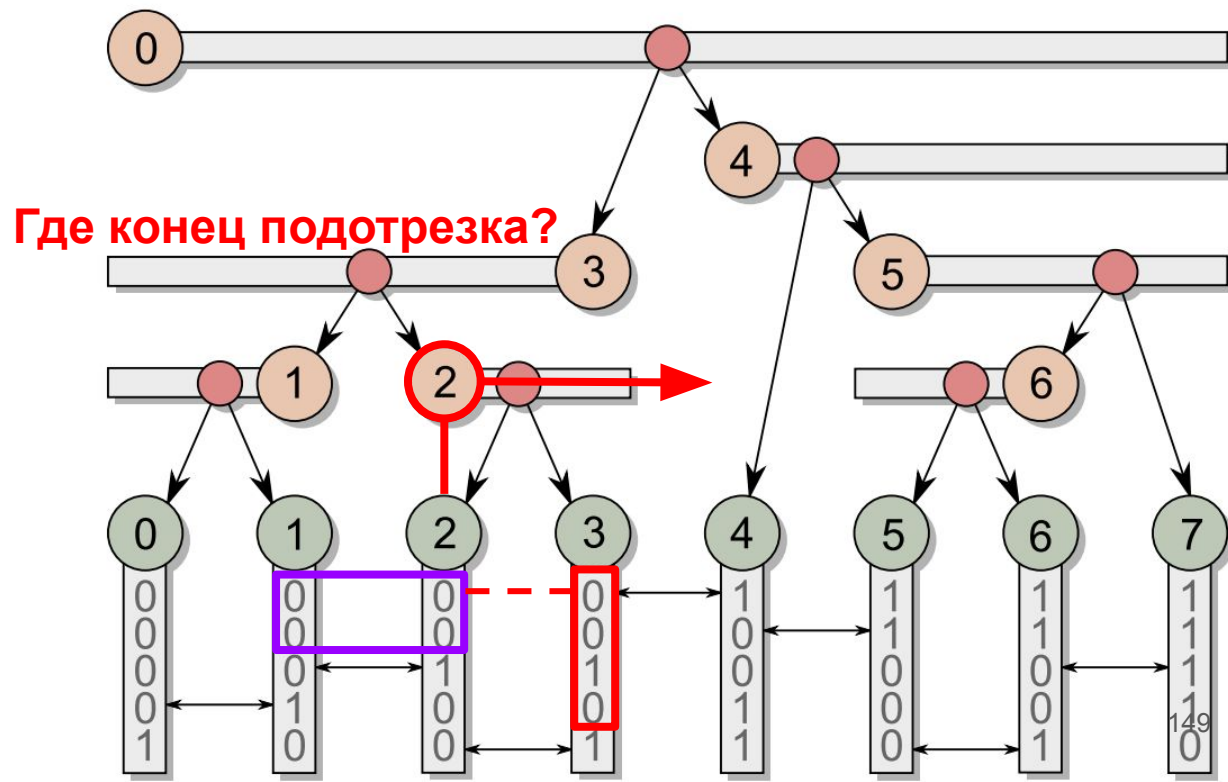


- 4.4) Но сначала каждый **узел** находит направление **подотрезка**
- 4.5) Как найти **конец** подотрезка?

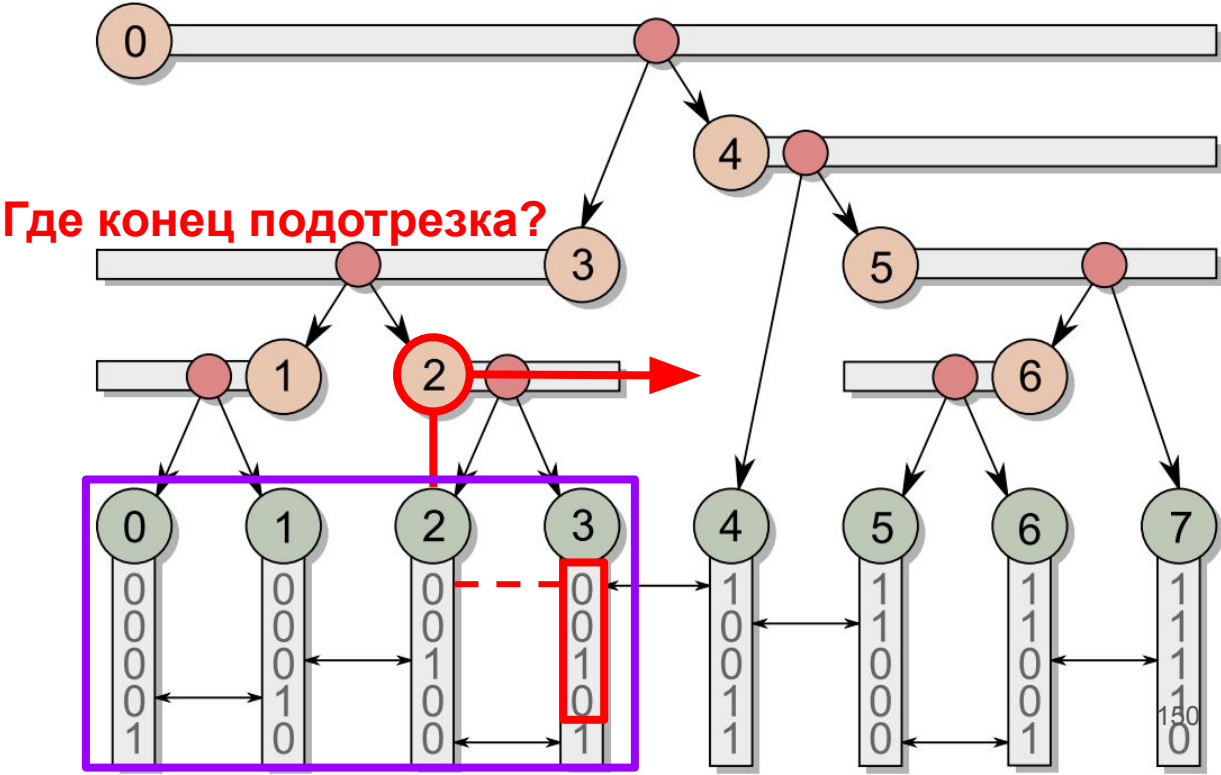


4.4) Но сначала каждый **узел** находит направление **подотрезка**

4.5) Как найти **конец** подотрезка?



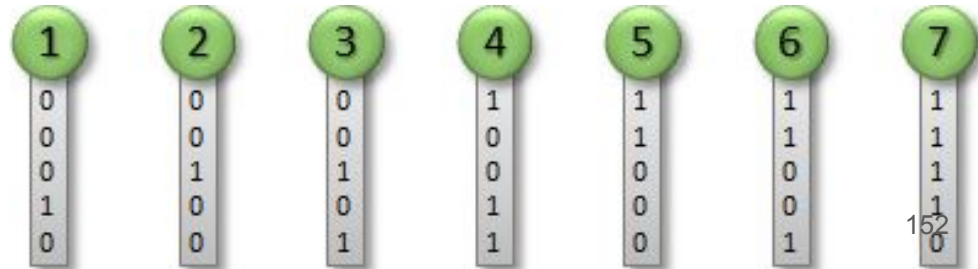
- 4.4) Но сначала каждый **узел** находит направление **подотрезка**
- 4.5) Как найти **конец** подотрезка?



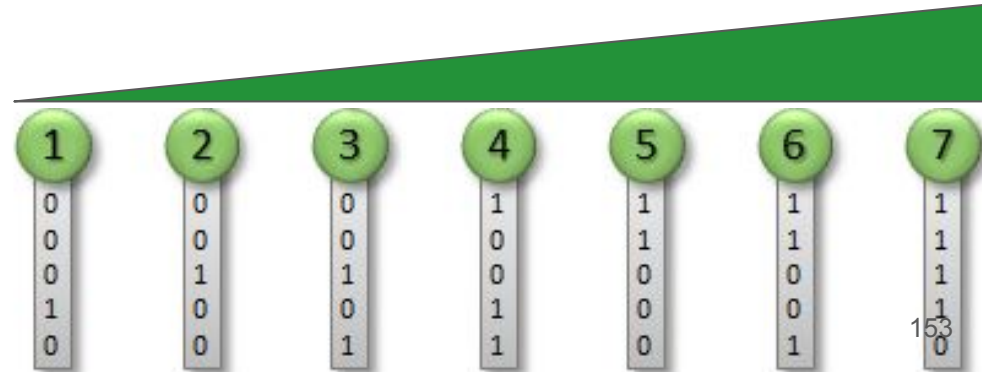
1) Есть много треугольников, каждый со своим AABV, это *листья*



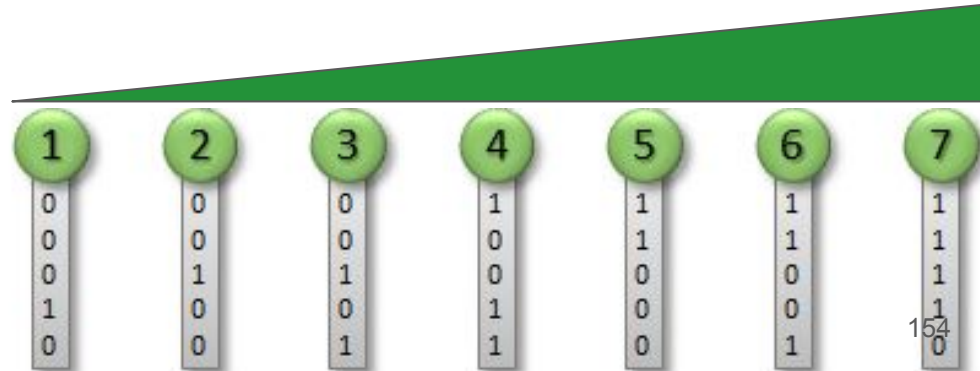
- 1) Есть много треугольников, каждый со своим AABV, это **листья**
- 2) Вычисляем у каждого **листа** код мортона (по точке-центроиду)



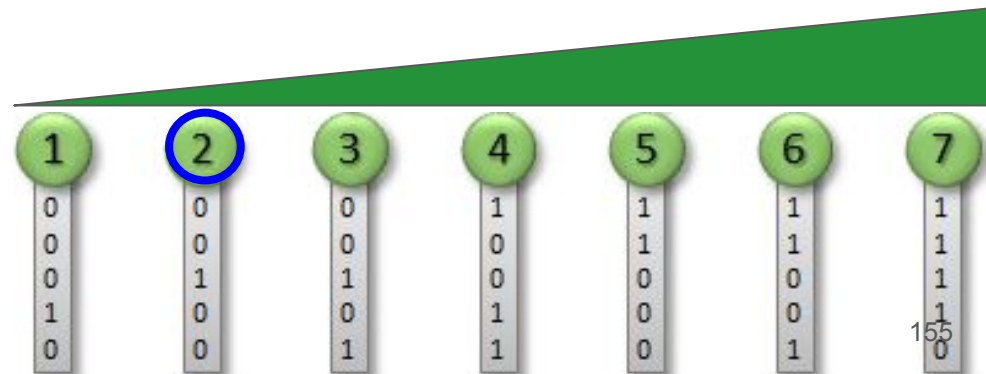
- 1) Есть много треугольников, каждый со своим AABV, это **листья**
- 2) Вычисляем у каждого **листа** код мортона (по точке-центроиду)
- 3) Сортируем **листья** по коду мортона



- 1) Есть много треугольников, каждый со своим AABV, это **листья**
- 2) Вычисляем у каждого **листа** код мортона (по точке-центроиду)
- 3) Сортируем **листья** по коду мортона
- 4) Нужно построить промежуточные узлы LBVH (параллельно)

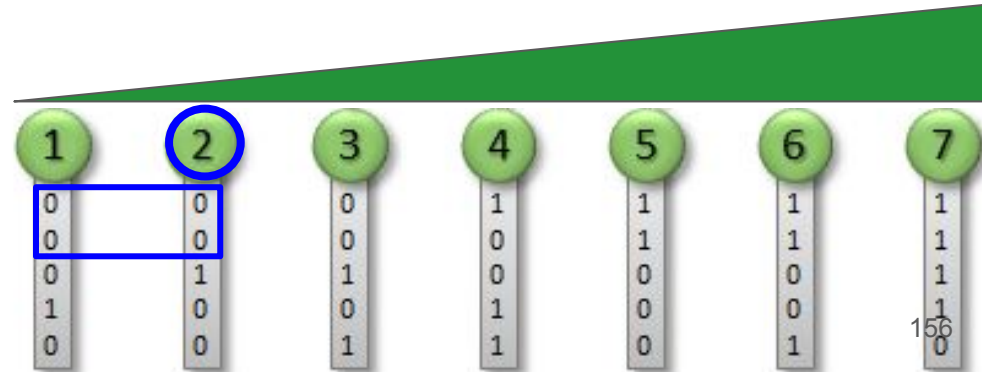


4) Для каждого узла i :



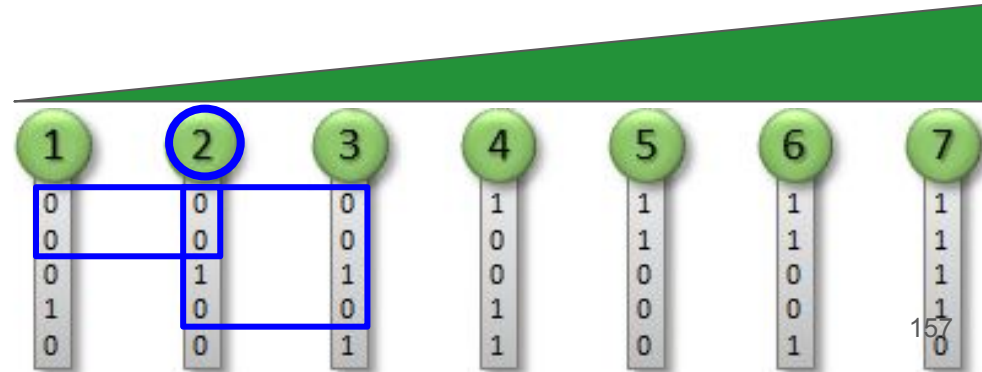
4) Для каждого **узла** i :

4.1) Находим общий префикс с $i-1$



4) Для каждого узла i :

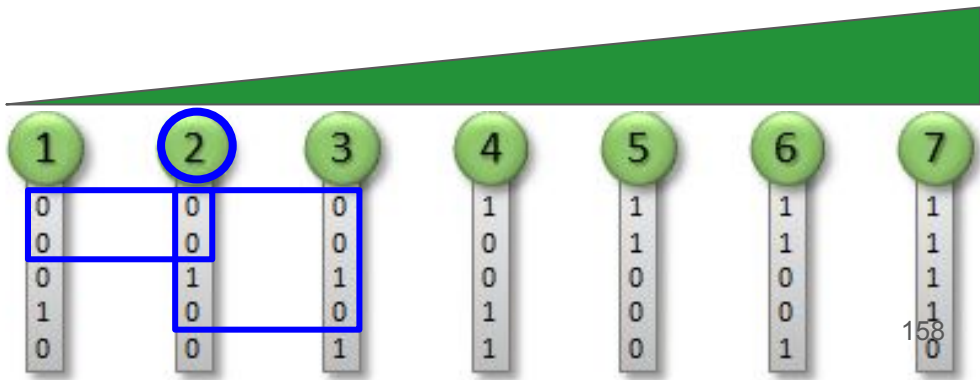
4.1) Находим общий префикс с $i-1$ и $i+1$



```
2: // Determine direction of the range (+1 or -1)
3: d ← sign(δ(i, i + 1) − δ(i, i − 1))
```



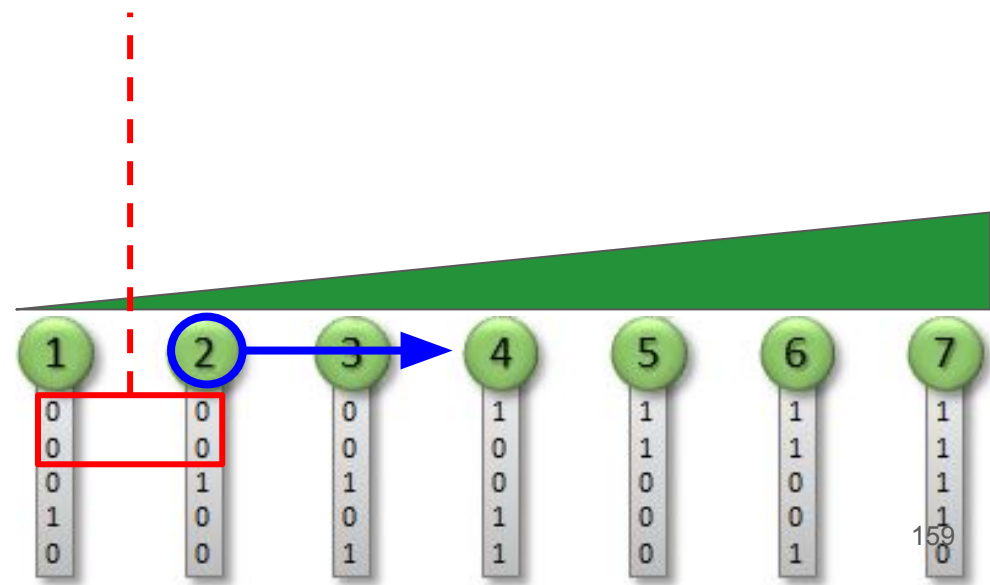
- 4) Для каждого **узла i** :
- 4.1) Находим общий префикс с **$i-1$** и **$i+1$**
- 4.2) Находим направление **подотрезка**



```
2: // Determine direction of the range (+1 or -1)
3: d ← sign(δ(i, i + 1) − δ(i, i − 1))
```



- 4) Для каждого **узла** i :
- 4.1) Находим общий префикс с $i-1$ и $i+1$
- 4.2) Находим направление **подотрезка**



2: // Determine direction of the range (+1 or -1)
3: $d \leftarrow \text{sign}(\delta(i, i+1) - \delta(i, i-1))$



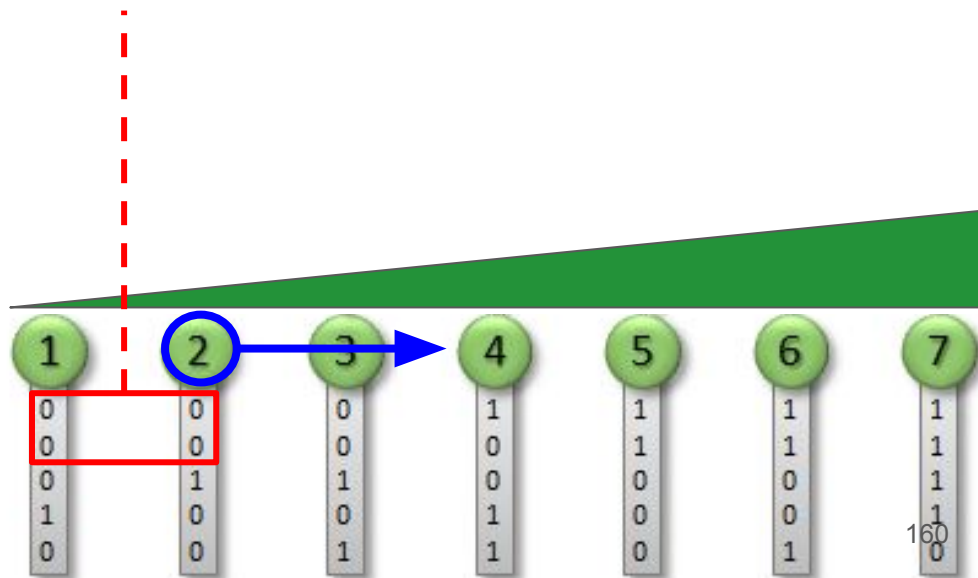
4) Для каждого **узла** i :

4.1) Находим общий префикс с $i-1$ и $i+1$

4.2) Находим направление **подотрезка**

4.4) Находим конец **подотрезка**

Как?



```

2: // Determine direction of the range (+1 or -1)
3:  $d \leftarrow \text{sign}(\delta(i, i+1) - \delta(i, i-1))$ 
4: // Compute upper bound for the length of the range
5:  $\delta_{\min} \leftarrow \delta(i, i-d)$ 
6:  $l_{\max} \leftarrow 2$ 
7: while  $\delta(i, i + l_{\max} \cdot d) > \delta_{\min}$  do
8:    $l_{\max} \leftarrow l_{\max} \cdot 2$ 

```



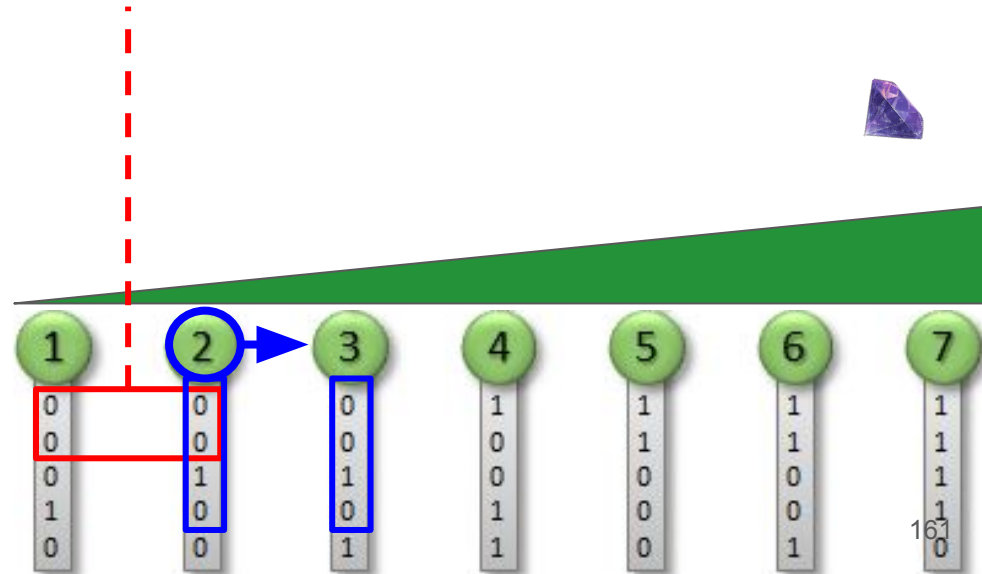
4) Для каждого **узла** i :

4.1) Находим общий префикс с $i-1$ и $i+1$

4.2) Находим направление **подотрезка**

4.4) Находим конец **подотрезка**

Как?




```

2: // Determine direction of the range (+1 or -1)
3:  $d \leftarrow \text{sign}(\delta(i, i+1) - \delta(i, i-1))$ 
4: // Compute upper bound for the length of the range
5:  $\delta_{\min} \leftarrow \delta(i, i-d)$ 
6:  $l_{\max} \leftarrow 2$ 
7: while  $\delta(i, i + l_{\max} \cdot d) > \delta_{\min}$  do
8:    $l_{\max} \leftarrow l_{\max} \cdot 2$ 

```



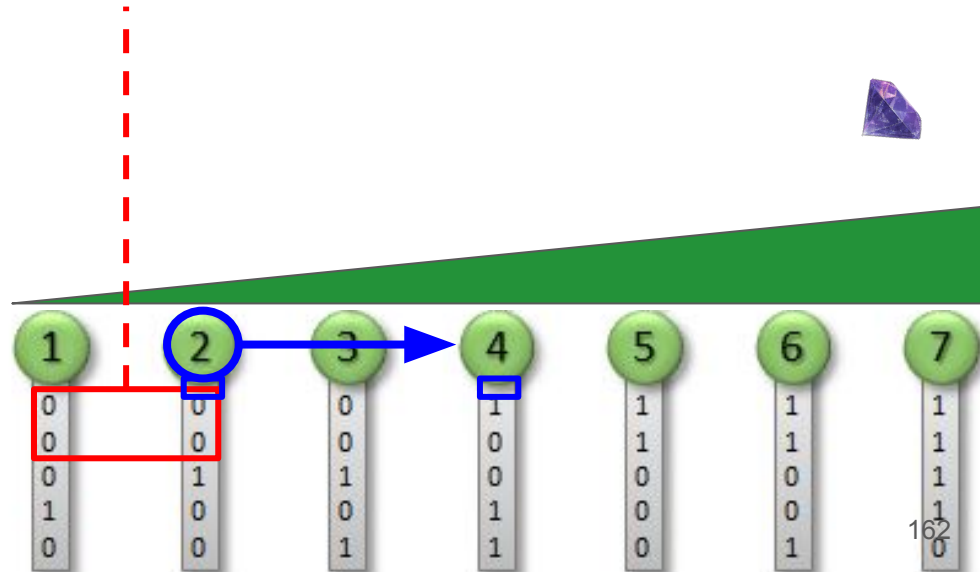
4) Для каждого **узла** i :

4.1) Находим общий префикс с $i-1$ и $i+1$

4.2) Находим направление **подотрезка**

4.4) Находим конец **подотрезка**

Как?



```

2: // Determine direction of the range (+1 or -1)
3:  $d \leftarrow \text{sign}(\delta(i, i+1) - \delta(i, i-1))$ 
4: // Compute upper bound for the length of the range
5:  $\delta_{\min} \leftarrow \delta(i, i-d)$ 
6:  $l_{\max} \leftarrow 2$ 
7: while  $\delta(i, i + l_{\max} \cdot d) > \delta_{\min}$  do
8:    $l_{\max} \leftarrow l_{\max} \cdot 2$ 

```



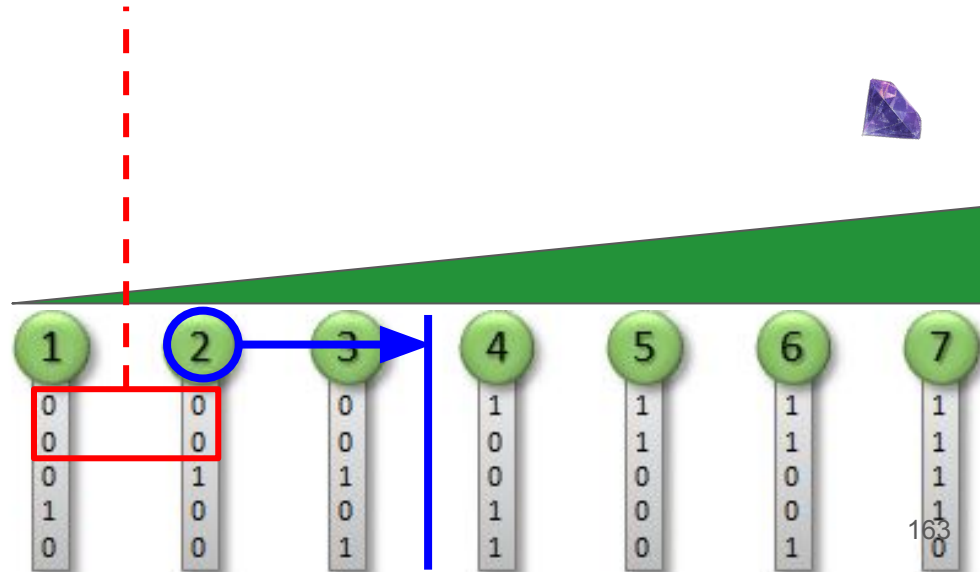
4) Для каждого **узла** i :

4.1) Находим общий префикс с $i-1$ и $i+1$

4.2) Находим направление **подотрезка**

4.4) Находим конец **подотрезка**

Как?



```

2: // Determine direction of the range (+1 or -1)
3:  $d \leftarrow \text{sign}(\delta(i, i+1) - \delta(i, i-1))$ 
4: // Compute upper bound for the length of the range
5:  $\delta_{\min} \leftarrow \delta(i, i-d)$ 
6:  $l_{\max} \leftarrow 2$ 
7: while  $\delta(i, i + l_{\max} \cdot d) > \delta_{\min}$  do
8:    $l_{\max} \leftarrow l_{\max} \cdot 2$ 
9: // Find the other end using binary search
10:  $l \leftarrow 0$ 
11: for  $t \leftarrow \{l_{\max}/2, l_{\max}/4, \dots, 1\}$  do
12:   if  $\delta(i, i + (l+t) \cdot d) > \delta_{\min}$  then
13:      $l \leftarrow l + t$ 
14:  $j \leftarrow i + l \cdot d$ 

```



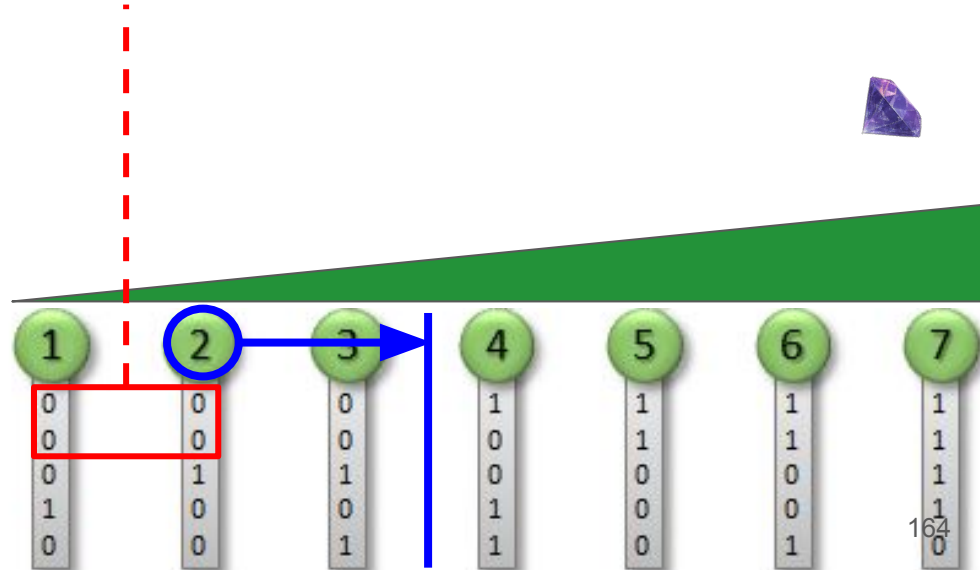
4) Для каждого **узла** i :

4.1) Находим общий префикс с $i-1$ и $i+1$

4.2) Находим направление **подотрезка**

4.4) Находим конец **подотрезка**

Как?



```

2: // Determine direction of the range (+1 or -1)
3:  $d \leftarrow \text{sign}(\delta(i, i+1) - \delta(i, i-1))$ 
4: // Compute upper bound for the length of the range
5:  $\delta_{\min} \leftarrow \delta(i, i-d)$ 
6:  $l_{\max} \leftarrow 2$ 
7: while  $\delta(i, i + l_{\max} \cdot d) > \delta_{\min}$  do
8:    $l_{\max} \leftarrow l_{\max} \cdot 2$ 
9: // Find the other end using binary search
10:  $l \leftarrow 0$ 
11: for  $t \leftarrow \{l_{\max}/2, l_{\max}/4, \dots, 1\}$  do
12:   if  $\delta(i, i + (l+t) \cdot d) > \delta_{\min}$  then
13:      $l \leftarrow l + t$ 
14:  $j \leftarrow i + l \cdot d$ 

```

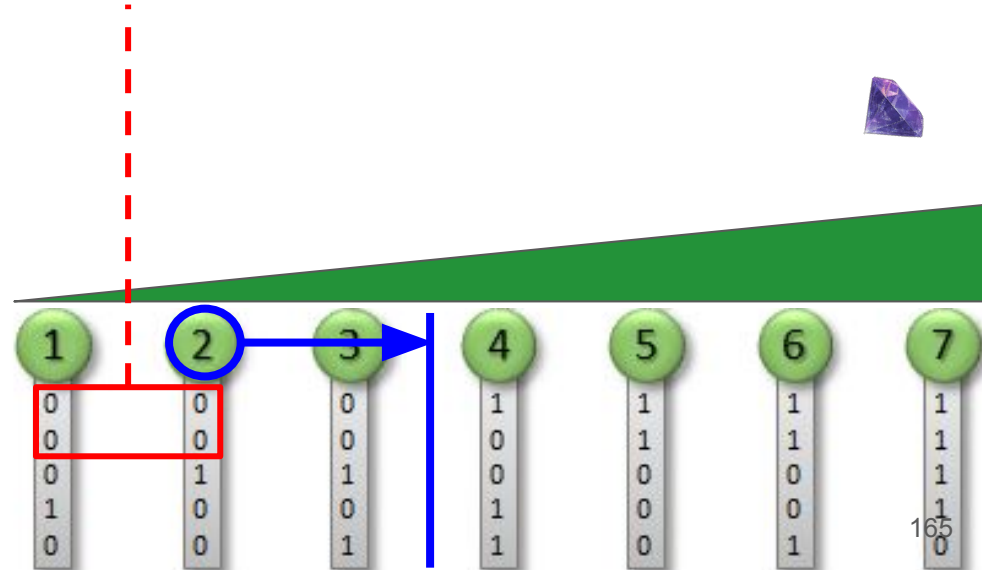


4) Для каждого **узла** i :

4.1) Находим общий префикс с $i-1$ и $i+1$

4.2) Находим направление **подотрезка**

4.4) Находим конец **подотрезка** (2 x бин.п.)



```

2: // Determine direction of the range (+1 or -1)
3:  $d \leftarrow \text{sign}(\delta(i, i+1) - \delta(i, i-1))$ 
4: // Compute upper bound for the length of the range
5:  $\delta_{\min} \leftarrow \delta(i, i-d)$ 
6:  $l_{\max} \leftarrow 2$ 
7: while  $\delta(i, i + l_{\max} \cdot d) > \delta_{\min}$  do
8:    $l_{\max} \leftarrow l_{\max} \cdot 2$ 
9: // Find the other end using binary search
10:  $l \leftarrow 0$ 
11: for  $t \leftarrow \{l_{\max}/2, l_{\max}/4, \dots, 1\}$  do
12:   if  $\delta(i, i + (l+t) \cdot d) > \delta_{\min}$  then
13:      $l \leftarrow l + t$ 
14:  $j \leftarrow i + l \cdot d$ 

```



4) Для каждого **узла** i :

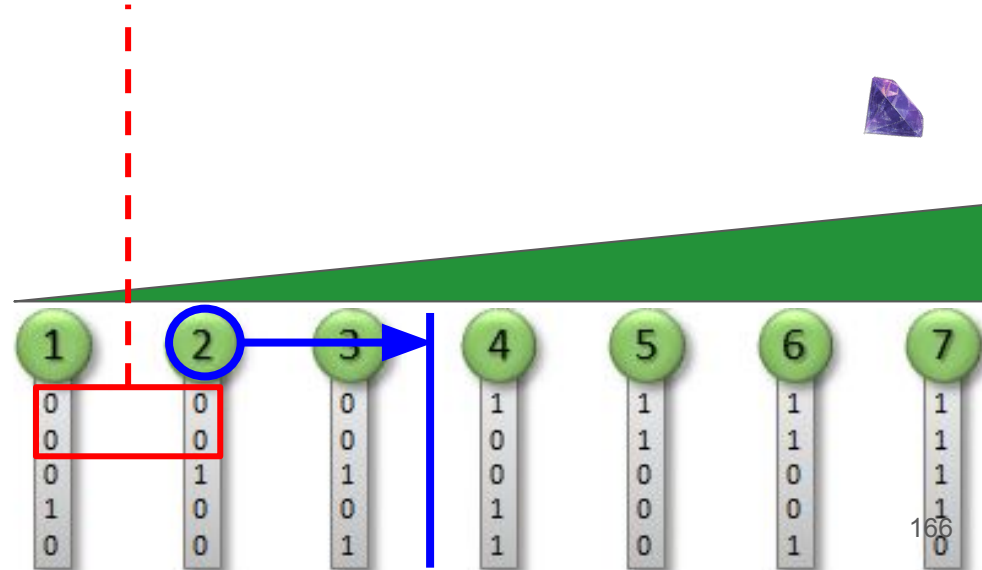
4.1) Находим общий префикс с $i-1$ и $i+1$

4.2) Находим направление **подотрезка**

4.4) Находим конец **подотрезка** (2 x бин.п.)

4.5) Находим **SPLIT** подотрезка:

как?



```

2: // Determine direction of the range (+1 or -1)
3:  $d \leftarrow \text{sign}(\delta(i, i+1) - \delta(i, i-1))$ 
4: // Compute upper bound for the length of the range
5:  $\delta_{\min} \leftarrow \delta(i, i-d)$ 
6:  $l_{\max} \leftarrow 2$ 
7: while  $\delta(i, i+l_{\max} \cdot d) > \delta_{\min}$  do
8:    $l_{\max} \leftarrow l_{\max} \cdot 2$ 
9: // Find the other end using binary search
10:  $l \leftarrow 0$ 
11: for  $t \leftarrow \{l_{\max}/2, l_{\max}/4, \dots, 1\}$  do
12:   if  $\delta(i, i+(l+t) \cdot d) > \delta_{\min}$  then
13:      $l \leftarrow l+t$ 
14:  $j \leftarrow i+l \cdot d$ 

```

4) Для каждого **узла** i :

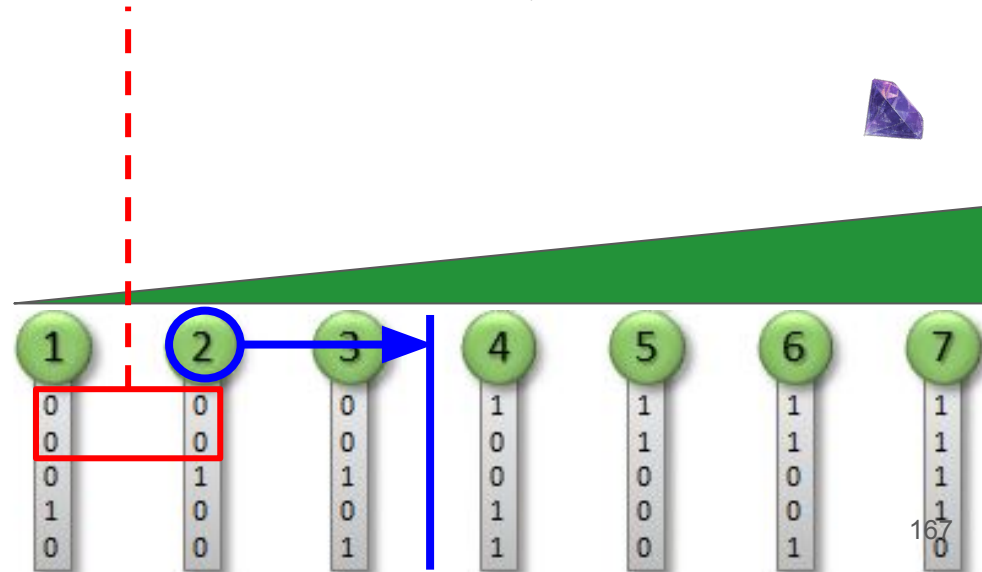
4.1) Находим общий префикс с $i-1$ и $i+1$

4.2) Находим направление **подотрезка**

4.4) Находим конец **подотрезка** (2 x бин.п.)

4.5) Находим **SPLIT** подотрезка:

поиск старшего различающегося бита (бин.п.)




```

2: // Determine direction of the range (+1 or -1)
3:  $d \leftarrow \text{sign}(\delta(i, i+1) - \delta(i, i-1))$ 
4: // Compute upper bound for the length of the range
5:  $\delta_{\min} \leftarrow \delta(i, i-d)$ 
6:  $l_{\max} \leftarrow 2$ 
7: while  $\delta(i, i+l_{\max} \cdot d) > \delta_{\min}$  do
8:    $l_{\max} \leftarrow l_{\max} \cdot 2$ 
9: // Find the outer end using binary search
10:  $l \leftarrow 0$ 
11: for  $t \leftarrow \{l_{\max}/2, l_{\max}/4, \dots, 1\}$  do
12:   if  $\delta(i, i+(l+t) \cdot d) > \delta_{\min}$  then
13:      $l \leftarrow l+t$ 
14:  $j \leftarrow i+l \cdot d$ 
15: // Find the split position using binary search
16:  $\delta_{\text{node}} \leftarrow \delta(i, j)$ 
17:  $s \leftarrow 0$ 
18: for  $t \leftarrow \{\lceil l/2 \rceil, \lceil l/4 \rceil, \dots, 1\}$  do
19:   if  $\delta(i, i+(s+t) \cdot d) > \delta_{\text{node}}$  then
20:      $s \leftarrow s+t$ 
21:  $\gamma \leftarrow i+s \cdot d + \min(d, 0)$ 

```

4) Для каждого **узла** i :

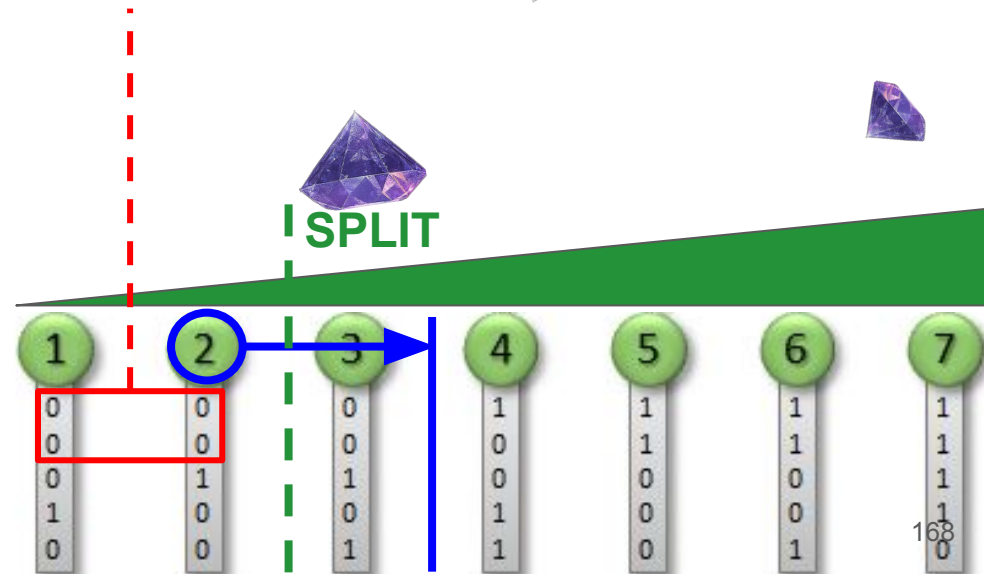
4.1) Находим общий префикс с $i-1$ и $i+1$

4.2) Находим направление **подотрезка**

4.4) Находим конец **подотрезка** (2 x бин.п.)

4.5) Находим **SPLIT** подотрезка:

поиск старшего различающегося бита (бин.п.)



```

2: // Determine direction of the range (+1 or -1)
3:  $d \leftarrow \text{sign}(\delta(i, i+1) - \delta(i, i-1))$ 
4: // Compute upper bound for the length of the range
5:  $\delta_{\min} \leftarrow \delta(i, i-d)$ 
6:  $l_{\max} \leftarrow 2$ 
7: while  $\delta(i, i+l_{\max} \cdot d) > \delta_{\min}$  do
8:    $l_{\max} \leftarrow l_{\max} \cdot 2$ 
9: // Find the other end using binary search
10:  $l \leftarrow 0$ 
11: for  $t \leftarrow \{l_{\max}/2, l_{\max}/4, \dots, 1\}$  do
12:   if  $\delta(i, i+(l+t) \cdot d) > \delta_{\min}$  then
13:      $l \leftarrow l+t$ 
14:  $j \leftarrow i+l \cdot d$ 
15: // Find the split position using binary search
16:  $\delta_{\text{node}} \leftarrow \delta(i, j)$ 
17:  $s \leftarrow 0$ 
18: for  $t \leftarrow \{\lceil l/2 \rceil, \lceil l/4 \rceil, \dots, 1\}$  do
19:   if  $\delta(i, i+(s+t) \cdot d) > \delta_{\text{node}}$  then
20:      $s \leftarrow s+t$ 
21:  $\gamma \leftarrow i+s \cdot d + \min(d, 0)$ 
22: // Output child pointers
23: if  $\min(i, j) = \gamma$  then  $\text{left} \leftarrow L_{\gamma}$  else  $\text{left} \leftarrow L_{\gamma}$ 
24: if  $\max(i, j) = \gamma+1$  then  $\text{right} \leftarrow L_{\gamma+1}$  else  $\text{right} \leftarrow L_{\gamma+1}$ 
25:  $I_i \leftarrow (\text{left}, \text{right})$ 

```

4) Для каждого **узла** i :

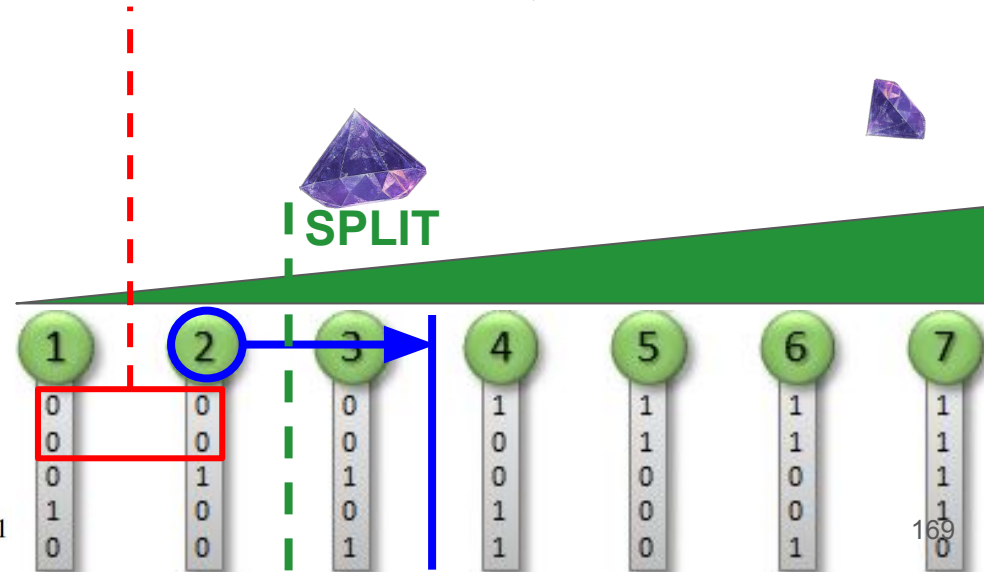
4.1) Находим общий префикс с i -

4.2) Находим направление подотр

4.4) Находим конец подотрезка (2 x бин.п.)

4.5) Находим **SPLIT** подотрезка:

поиск старшего различающегося бита (бин.п.)



Глава 4: Вечерние чтения

Semantic Scholar
Surface Area Heuristic
Triangle splitting



Исследуем дальше через [semanticscholar.org](https://www.semanticscholar.org)



SEMANTIC SCHOLAR

Maximizing Parallelism in the Construction of BVHs, Octrees, and k-d Trees

Search

Sign In

Create Free Account

DOI: 10.2312/EGGH/HPG12/033-037 • Corpus ID: 16898234

Maximizing parallelism in the construction of BVHs, octrees, and k-d trees

Tero Karras • Published in EGGH-HPG'12 25 June 2012 • Computer Science

TLDR This work presents a novel approach that improves scalability by constructing the entire tree in parallel, an in-place algorithm for constructing binary radix trees, which is used as a building block for other types of trees.[Expand](#)

[View on ACM](#)

[\[PDF\] research.nvidia.com](#)

[Save to Library](#)

[Create Alert](#)

[Cite](#)

[Share](#)

249 Citations

Highly Influential Citations ⓘ 35

Background Citations 58

Methods Citations 105

Results Citations 4

[View All](#)

Figures and Tables

Topics

249 Citations

10 References

Related Papers

Figures and Tables from this paper

Исследуем дальше через [semanticscholar.org](https://www.semanticscholar.org)



SEMANTIC SCHOLAR

Maximizing Parallelism in the Construction of BVHs, Octrees, and k-d Trees

Search

Sign In

Create Free Account

DOI: 10.2312/EGGH/HPG12/033-037 • Corpus ID: 16898234

Share

Maximizing parallelism in the construction of BVHs, octrees, and k-d trees

Tero Karras • Published in EGGH-HPG'12 25 June 2012 • Computer Science

TLDR This work presents a novel approach that improves scalability by constructing the entire tree in parallel, an in-place algorithm for constructing binary radix trees, which is used as a building block for other types of trees.[Expand](#)

[View on ACM](#)

[PDF](#) research.nvidia.com

[Save to Library](#)

[Create Alert](#)

[Cite](#)

249 Citations

Highly Influential Citations 35

Background Citations 58

Methods Citations 105

Results Citations 4

[View All](#)

Figures and Tables

Topics

249 Citations

10 References

Related Papers

Figures and Tables from this paper

249 Citations



Date Range ▾

Citation Type ▾

Has PDF

Author ▾

More Filters

Sort by Citation Count ▾

Expressive Body Capture: 3D Hands, Face, and Body From a Single Image

G. Pavlakos

Vasileios Choutas

+4 authors

Michael J. Black

Computer Science

Computer Vision and Pattern Recognition · 2019

TLDR This work uses the new method, SMPLify-X, to fit SMPL-X to both controlled images and images in the wild, and evaluates 3D accuracy on a new curated dataset comprising 100 images with pseudo ground-truth. [Expand](#)

1,947

[PDF]

Save

GRAB: A Dataset of Whole-Body Human Grasping of Objects

Omid Taheri

N. Ghorbani

Michael J. Black

Dimitrios Tzionas

Computer Science

European Conference on Computer Vision · 2020

TLDR This work collects a new dataset, called GRAB (GRasping Actions with Bodies), of whole-body grasps, containing full 3D shape and pose sequences of 10 subjects interacting with 51 everyday objects of varying shape and size, and trains GrabNet, a conditional generative network, to predict 3D handgrasps for unseen 3D object shapes. [Expand](#)

431

[PDF]

1 Excerpt

Save

Pulsar: Efficient Sphere-based Neural Rendering

Christoph Lassner

M. Zollhöfer

Computer Science

Computer Vision and Pattern Recognition · 2021

TLDR Pulsar is an efficient sphere-based differentiable rendering module that is orders of magnitude faster than competing techniques, modular, and easy-to-use due to its tight integration with PyTorch, and enables a plethora of applications, ranging from 3D reconstruction to neural rendering. [Expand](#)

196

PDF

1 Excerpt

Save

Fast parallel construction of high-quality bounding volume hierarchies

Tero Karras

Timo Aila

Computer Science

High Performance Graphics · 2013

TLDR A new massively parallel algorithm for constructing high-quality bounding volume hierarchies (BVHs) for ray tracing, based on modifying an existing BVH to improve its quality, and executes in linear time at a rate of almost 40M triangles/sec on NVIDIA GTX Titan. [Expand](#)

158

PDF

Save

LEAP: Learning Articulated Occupancy of People

Marko Mihajlovic

Yan Zhang

Michael J. Black

Siyu Tang

Computer Science

Computer Vision and Pattern Recognition · 2021

TLDR Experiments show that the canonicalized occupancy estimation with the learned LBS functions greatly improves the generalization capability

249 Citations



Date Range ▾

Citation Type ▾

Has PDF

Author ▾

More Filters

Sort by Citation Count ▾

Expressive Body Capture: 3D Hands, Face, and Body From a Single Image

G. Pavlakos

Vasileios Choutas

+4 authors

Michael J. Black

Computer Science

· [Computer Vision and Pattern Recognition](#) · 2019

TLDR This work uses the new method, SMPLify-X, to fit SMPL-X to both controlled images and images in the wild, and evaluates 3D accuracy on a new curated dataset comprising 100 images with pseudo ground-truth. [Expand](#)

1,947

[PDF]

Save

GRAB: A Dataset of Whole-Body Human Grasping of Objects

Omid Taheri

N. Ghorbani

Michael J. Black

Dimitrios Tzionas

Computer Science

· [European Conference on Computer Vision](#) · 2020

TLDR This work collects a new dataset, called GRAB (GRasping Actions with Bodies), of whole-body grasps, containing full 3D shape and pose sequences of 10 subjects interacting with 51 everyday objects of varying shape and size, and trains GrabNet, a conditional generative network, to predict 3D handgrasps for unseen 3D object shapes. [Expand](#)

431

[PDF]

1 Excerpt

Save

Pulsar: Efficient Sphere-based Neural Rendering

Christoph Lassner

M. Zollhöfer

Computer Science

· [Computer Vision and Pattern Recognition](#) · 2021

TLDR Pulsar is an efficient sphere-based differentiable rendering module that is orders of magnitude faster than competing techniques, modular, and easy-to-use due to its tight integration with PyTorch, and enables a plethora of applications, ranging from 3D reconstruction to neural rendering. [Expand](#)

196

PDF

1 Excerpt

Save

Fast parallel construction of high-quality bounding volume hierarchies

Tero Karras

Timo Aila

Computer Science

· [High Performance Graphics](#) · 2013

TLDR A new massively parallel algorithm for constructing high-quality bounding volume hierarchies (BVHs) for ray tracing, based on modifying an existing BVH to improve its quality, and executes in linear time at a rate of almost 40M triangles/sec on NVIDIA GTX Titan. [Expand](#)

158

PDF

Save

LEAP: Learning Articulated Occupancy of People

Marko Mihajlovic

Yan Zhang

Michael J. Black

Siyu Tang

Computer Science

· [Computer Vision and Pattern Recognition](#) · 2021

TLDR Experiments show that the canonicalized occupancy estimation with the learned LBS functions greatly improves the generalization capability

Fast Parallel Construction of High-Quality Bounding Volume Hierarchies

Tero Karras

Timo Aila

NVIDIA

Abstract

We propose a new massively parallel algorithm for constructing high-quality bounding volume hierarchies (BVHs) for ray tracing. The algorithm is based on modifying an existing BVH to improve its quality, and executes in linear time at a rate of almost 40M triangles/sec on NVIDIA GTX Titan. We also propose an improved approach for parallel splitting of triangles prior to tree construction. Averaged over 20 test scenes, the resulting trees offer over 90% of the ray tracing performance of the best offline construction method (SBVH), while previous fast GPU algorithms offer only about 50%. Compared to state-of-the-art, our method offers a significant improvement in the majority of practical workloads that need to construct the BVH for each frame. On the average, it gives the best overall performance when tracing between 7 million and 60 billion rays per frame. This covers most interactive applications, product and architectural design, and even movie rendering.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Raytracing;

Keywords: ray tracing, bounding volume hierarchies

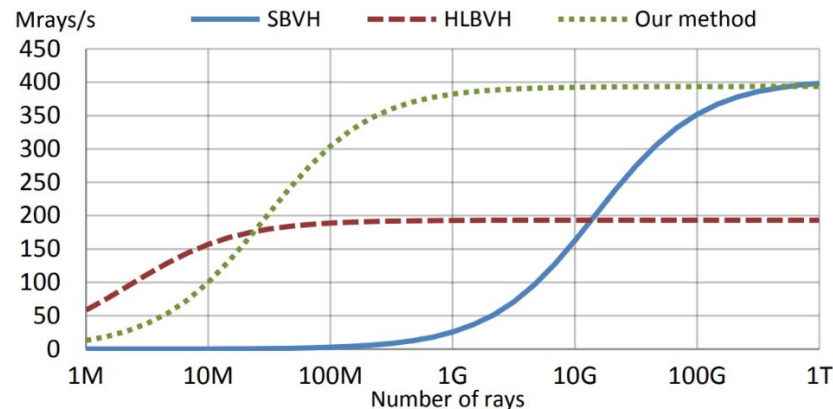


Figure 1: Performance of constructing a BVH and then casting a number of diffuse rays with NVIDIA GTX Titan in SODA (2.2M triangles). SBVH [Stich et al. 2009] yields excellent ray tracing performance, but suffers from long construction times. HLBVH [Garanzha et al. 2011a] is very fast to construct, but reaches only about 50% of the performance of SBVH. Our method is able to reach 97% while still being fast enough to use in interactive applications. In this particular scene, it offers the best quality–speed tradeoff for workloads ranging from 30M to 500G rays per frame.

KARRAS, T. 2012. Maximizing parallelism in the construction of BVHs, octrees, and k-d trees. In *Proc. High-Performance Graphics 2012*, 33–37.

KENSLER, A. 2008. Tree rotations for improving bounding volume hierarchies. In *Proc. IEEE Symposium on Interactive Tracing*, 73–76.

Записываем себе - что такое SAH?
(вместо одного треугольника на лист)

Without Triangle Splitting We first analyze the performance without triangle splitting by comparing against six comparison methods: greedy top-down sweep SAH (SweepSAH) [MacDonald and Booth 1990], LBBVH [Lauterbach et al. 2009; Karras 2012], HLBVH [Garanzha et al. 2011a], GridSAH [Garanzha et al. 2011b], tree rotations with hill climbing (Kensler) [Kensler 2008], and iterative reinsertion (Bittner) [Bittner et al. 2013], where the last two are initialized using LBBVH. We use the authors' original implementations for HLBVH and GridSAH, and our own implementations for the rest of the methods. We choose SweepSAH as the baseline because it continues to be the de facto standard in BVH construction. For LBBVH, we optimize the leaf nodes by collapsing subtrees into leaves to minimize SAH cost (instead of using single-triangle leaves). This is a well-known method, takes a negligible amount of time, and considerably improves the LBBVH results. Of the comparison methods LBBVH, HLBVH and GridSAH execute on the GPU, SweepSAH and Kensler are parallelized over four CPU cores, and Bittner runs on a single CPU core. Our method uses a fixed set of parameters for all test scenes: $n = 7$, $\gamma = 7$, we perform 3 rounds of optimization, and γ is doubled after every round.

Table 3 gives numbers for ray tracing performance, SAH cost, and build time for four scenes in addition to the average numbers over the 20 scenes. On average, our builder produces trees that offer 96% of the maximum achievable ray tracing performance of SweepSAH (min 86% max 113%), while being significantly faster to construct. LBBVH offers 3–4 times faster build than our method, but provides only 69% of the maximum ray tracing performance on the average, with significant scene-dependent variation: the performance is only 40–50% for all city scenes but over 80% for highly tessellated objects. Interestingly HLBVH and GridSAH result in

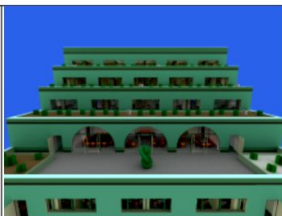
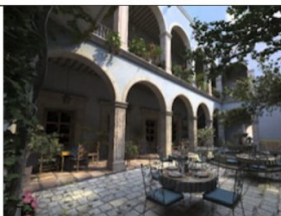
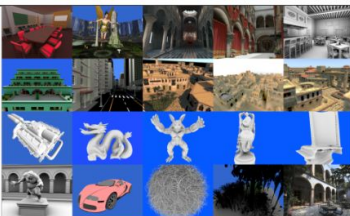
															
	CONFERENCE 282K tris			SODA 2.2M tris			DRAGON 870K tris			SANMIGUEL 10.5M tris			AVERAGE OF 20 SCENES relative to SweepSAH		
Builder	Perf Mrays/s	SAH cost	Build time	Perf Mrays/s	SAH cost	Build time	Perf Mrays/s	SAH cost	Build time	Perf Mrays/s	SAH cost	Build time	Perf %	SAH %	Build %
SweepSAH	258.4	46.50	848 ms	330.2	78.26	8 s	229.7	56.74	3 s	86.7	20.13	50 s	100.0	100.0	100.00
Our	275.6	38.48	9 ms	301.8	71.04	56 ms	213.1	60.41	24 ms	83.5	17.38	274 ms	96.0	94.4	1.03
LBVH	179.4	64.59	2 ms	203.0	106.06	16 ms	188.5	70.00	7 ms	55.3	26.71	74 ms	69.4	131.5	0.28
HLBVH	185.0	60.91	6 ms	193.5	105.13	12 ms	189.1	68.54	8 ms	39.9	29.08	32 ms	67.4	129.3	0.57
GridSAH	187.8	61.96	7 ms	200.1	104.58	15 ms	193.8	66.12	11 ms	39.8	29.26	35 ms	68.1	129.1	0.70
Kensler	250.5	42.69	4 s	258.8	76.08	44 s	211.3	62.04	10 s	79.1	18.50	197 s	91.5	99.6	552.26
Bittner	278.7	36.51	1 s	356.5	59.39	60 s	215.6	57.26	50 s	98.2	15.69	334 s	103.3	87.7	1083.24

Table 3: Measurements for seven different BVH builders in absence of triangle splitting. Performance is given in millions of diffuse rays per second, and the build time includes everything from receiving a list of triangles to being ready to cast the rays. The last three columns give average results for the full set of 20 test scenes relative to SweepSAH.

Table 3 gives numbers for ray tracing performance, SAH cost, and build time for four scenes in addition to the average numbers over the 20 scenes. On average, our builder produces trees that offer 96% of the maximum achievable ray tracing performance of SweepSAH (min 86% max 113%), while being significantly faster to construct. **LBVH offers 3–4 times faster build than our method**, but provides only 69% of the maximum ray tracing performance on the average, with significant scene-dependent variation: the performance is only 40–50% for all city scenes but over 80% for highly tessellated objects. Interestingly HLBVH and GridSAH result in

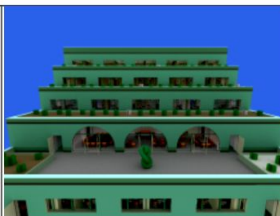
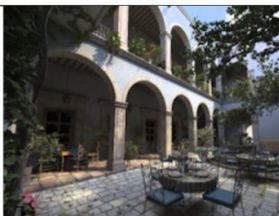
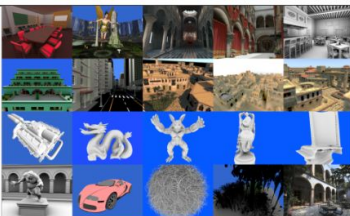
															
	CONFERENCE 282K tris			SODA 2.2M tris			DRAGON 870K tris			SANMIGUEL 10.5M tris			AVERAGE OF 20 SCENES relative to SweepSAH		
Builder	Perf Mrays/s	SAH cost	Build time	Perf Mrays/s	SAH cost	Build time	Perf Mrays/s	SAH cost	Build time	Perf Mrays/s	SAH cost	Build time	Perf %	SAH %	Build %
SweepSAH	258.4	46.50	848 ms	330.2	78.26	8 s	229.7	56.74	3 s	86.7	20.13	50 s	100.0	100.0	100.00
Our	275.6	38.48	9 ms	301.8	71.04	56 ms	213.1	60.41	24 ms	83.5	17.38	274 ms	96.0	94.4	1.03
LBVH	179.4	64.59	2 ms	203.0	106.06	16 ms	188.5	70.00	7 ms	55.3	26.71	74 ms	69.4	131.5	0.28
HLBVH	185.0	60.91	6 ms	193.5	105.13	12 ms	189.1	68.54	8 ms	39.9	29.08	32 ms	67.4	129.3	0.57
GridSAH	187.8	61.96	7 ms	200.1	104.58	15 ms	193.8	66.12	11 ms	39.8	29.26	35 ms	68.1	129.1	0.70
Kensler	250.5	42.69	4 s	258.8	76.08	44 s	211.3	62.04	10 s	79.1	18.50	197 s	91.5	99.6	552.26
Bittner	278.7	36.51	1 s	356.5	59.39	60 s	215.6	57.26	50 s	98.2	15.69	334 s	103.3	87.7	1083.24

Table 3: Measurements for seven different BVH builders in absence of triangle splitting. Performance is given in millions of diffuse rays per second, and the build time includes everything from receiving a list of triangles to being ready to cast the rays. The last three columns give average results for the full set of 20 test scenes relative to SweepSAH.

Table 3 gives numbers for ray tracing performance, SAH cost, and build time for four scenes in addition to the average numbers over the 20 scenes. On average, our builder produces trees that offer 96% of the maximum achievable ray tracing performance of SweepSAH (min 86% max 113%), while being significantly faster to construct. LBVH offers 3–4 times faster build than our method, but provides only 69% of the maximum ray tracing performance on the average, with significant scene-dependent variation: the performance is only 40–50% for all city scenes but over 80% for highly tessellated objects. Interestingly HLBVH and GridSAH result in

2 Related Work

Surface Area Heuristic Ray tracing performance is most commonly estimated using the surface area cost model, first introduced by Goldsmith and Salmon [1987] and later formalized by MacDonald and Booth [1990]. The SAH cost of a given acceleration structure is defined as the expected cost of tracing a non-terminating random ray through the scene:

$$C_i \sum_{n \in I} \frac{A(n)}{A(\text{root})} + C_l \sum_{l \in L} \frac{A(l)}{A(\text{root})} + C_t \sum_{l \in L} \frac{A(l)}{A(\text{root})} N(l), \quad (1)$$

where I and L are the sets of internal nodes and leaf nodes, respectively, and C_i and C_l are their associated traversal costs. C_t is the cost of a ray-triangle intersection test, and $N(l)$ denotes the number of triangles referenced by leaf node l . The surface area of the bounding volume in node n is indicated by $A(n)$, and the ratio $A(n)/A(\text{root})$ corresponds to the conditional probability that a random ray intersecting the root is also going to intersect n . In this paper, we use $C_i = 1.2$, $C_l = 0$, and $C_t = 1$, which we have verified experimentally to give the highest correlation with the measured performance.

The classic approach for constructing BVHs is based on greedy top-down partitioning of triangles that aims to minimize the SAH cost at every step [MacDonald and Booth 1990]. At each node, the triangles are classified to either side of an axis-aligned split plane according to the centroids of their axis-aligned bounding boxes (AABBs). The split plane is chosen by evaluating the SAH cost of the resulting child nodes for each potential plane, and selecting the one that results in the lowest cost. Leaf nodes are created when the SAH cost can no longer be improved through partitioning, i.e., the benefit of creating a new internal node is outweighed by its cost.

Another well-known approach is to start from the leaves and proceed in a bottom-up fashion by merging the nodes iteratively [Walter et al. 2008]. Even though this approach is often able to produce trees with a very low SAH cost, it tends to lose to the top-down algorithm in practical ray tracing performance.

Approximate Methods The most widely adopted simplification of the full top-down partitioning is the *binned SAH* [Wald 2007; Wald 2012], which limits the split planes considered at each node to a fixed number. The planes are placed uniformly along each axis to cover the spatial extent of the node, which makes it possible to *bin* the triangles into intervals between the planes according to their centroids. A further simplification is to first perform the binning globally using a fixed-size grid, and then reuse the same results for all nodes that overlap a given cell [Garanzha et al. 2011b].

Another approach is to use a linear BVH (LBVH) [Lauterbach et al. 2009], which can be constructed very quickly on the GPU. The idea is to first sort the triangles along a space-filling curve, and then partition them recursively so that each node ends up representing a linear range of triangles [Pantaleoni and Luebke 2010; Garanzha et al. 2011a]. Karras [2012] showed that every stage of the construction can be parallelized completely over the entire tree, which makes the rest of the stages practically free compared to the sorting.

Although linear BVHs are fast to construct, their ray tracing performance tends to be unacceptably low — usually around 50% of the gold standard. This necessitates using hybrid methods that construct important parts of the tree using a high-quality algorithm while using a fast algorithm for the expensive parts. HLBVH [Garanzha et al. 2011a], for instance, uses binned SAH for the top-most nodes while relying on linear BVH for the remaining ones.

BVH Optimization Closely related to our work, there has been some amount of research on optimizing *existing* BVHs as a post-process. Kensler [2008] proposed using local tree rotations to improve high-quality BVHs beyond the gold standard on the CPU, and Kopta et al. [2012] applied the same idea to refine BVHs during animation to combat their inherent degradation in quality. A similar approach was used in NVIDIA OptiX [2012] to improve the quality of HLBVH on the GPU. The main weakness of tree rotations is that they are prone to getting stuck in a local optimum in many scenes. To overcome this effect, one has to resort to stochastic methods that converge too slowly to be practical [Kensler 2008].

Recently, Bittner et al. [2013] presented an alternative algorithm based on iteratively removing nodes from the tree and inserting them back at optimal locations. Since there are a large number of options for modifying the tree at each step, the algorithm is able to improve the quality significantly before getting stuck. However, since the method is fundamentally serial, it is unclear whether it can be implemented efficiently on the GPU.

3 Overview

Our goal is to construct high-quality BVHs from scratch as quickly as possible. For maximum performance, we target NVIDIA Kepler GPUs using CUDA. Our approach is motivated by the insight offered by Bittner et al. [2013] that it is possible to take an existing low-quality BVH and modify it to match the quality of the best top-down methods. While Bittner et al. hypothesize that the individual tree modifications have to be global in nature for this to be possible, our intuition is that the *number* of possible modifications plays a more important role. For example, tree rotations [Kensler 2008] offer only 6 ways of modifying the tree per node. Once all of these modifications are exhausted, i.e., none of them is able to reduce the SAH cost any further, the optimization gets stuck.

Instead of looking at individual nodes, we extend the concept of tree rotations to larger neighborhoods of nodes. We define a *treelet* as the collection of immediate descendants of a given treelet root, consisting of n treelet leaves and $n - 1$ treelet internal nodes (Figure 2). We require the treelet to constitute a valid binary tree on its own, but it does not necessarily have to extend all the way down to the leaves of the BVH. In other words, the children of every internal node of the treelet must be contained in the treelet as well, but its leaves can act as representatives of arbitrarily large subtrees.

3 Overview

Our goal is to construct high-quality BVHs from scratch as quickly as possible. For maximum performance, we target NVIDIA Kepler GPUs using CUDA. Our approach is motivated by the insight offered by Bittner et al. [2013] that it is possible to take an existing low-quality BVH and modify it to match the quality of the best top-down methods. While Bittner et al. hypothesize that the individual tree modifications have to be global in nature for this to be possible, our intuition is that the *number* of possible modifications plays a more important role. For example, tree rotations [Kensler 2008] offer only 6 ways of modifying the tree per node. Once all of these modifications are exhausted, i.e., none of them is able to reduce the SAH cost any further, the optimization gets stuck.

Instead of looking at individual nodes, we extend the concept of tree rotations to larger neighborhoods of nodes. We define a *treelet* as the collection of immediate descendants of a given treelet root, consisting of n treelet leaves and $n - 1$ treelet internal nodes (Figure 2). We require the treelet to constitute a valid binary tree on its own, but it does not necessarily have to extend all the way down to the leaves of the BVH. In other words, the children of every internal node of the treelet must be contained in the treelet as well, but its leaves can act as representatives of arbitrarily large subtrees.

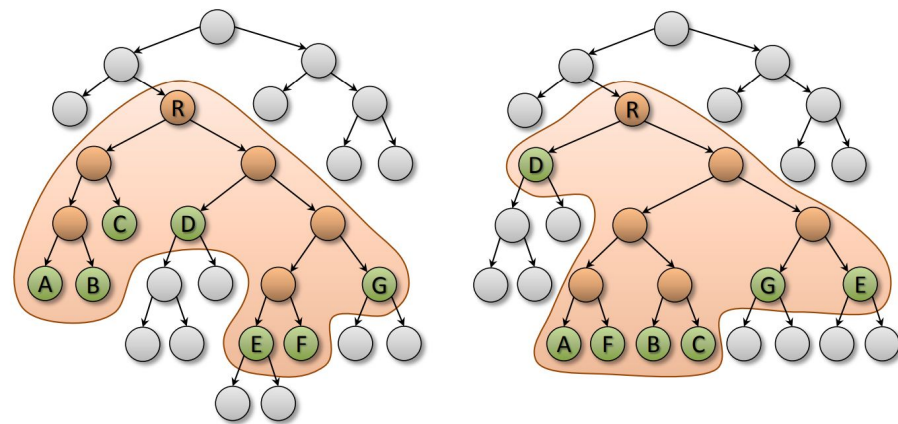


Figure 2: Left: *Treelet* consisting of 7 leaves (A–G) and 6 internal nodes, including the root (R). The leaves can either be actual leaf nodes of the BVH (A, B, C, F), or they can represent arbitrary subtrees (D, E, G). Right: Reorganized treelet topology to minimize the overall SAH cost. Descendants of the treelet leaves are kept intact, but their location in the tree is allowed to change.

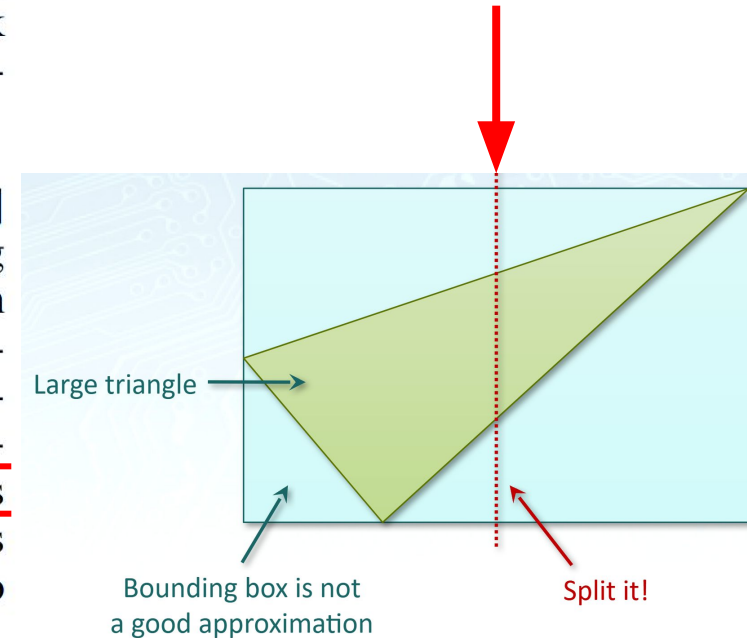
Triangle Splitting Several authors have noted that splitting large triangles can improve the quality of BVHs significantly in scenes that contain large variation in triangle sizes. Ernst and Greiner [2007] propose to split triangles along their longest axis if their surface area exceeds a pre-defined threshold, and Dammertz and Keller [2008] propose similar scheme to split triangle edges based on the volume of their axis-aligned bounding boxes. However, neither of these methods has been proven to work reliably in practice — they can actually end up decreasing the performance in certain scenes.

A better approach, proposed independently by Stich et al. [2009] (SBVH) and Popov et al. [2009], is to incorporate triangle splitting directly into the top-down construction algorithm. SBVH, which yields the highest ray tracing performance to date, works by considering *spatial splits* in addition to conventional partitioning of triangles. Spatial splits correspond to duplicating triangles that intersect a given split plane, so that each resulting *triangle reference* lies strictly on either side of the plane. The choice between spatial splits and triangle partitioning is made on a per-node basis according to which alternative is the most effective in reducing the SAH cost.

Fast parallel construction of high-quality bounding volume hierarchies, Karras et al, 2013

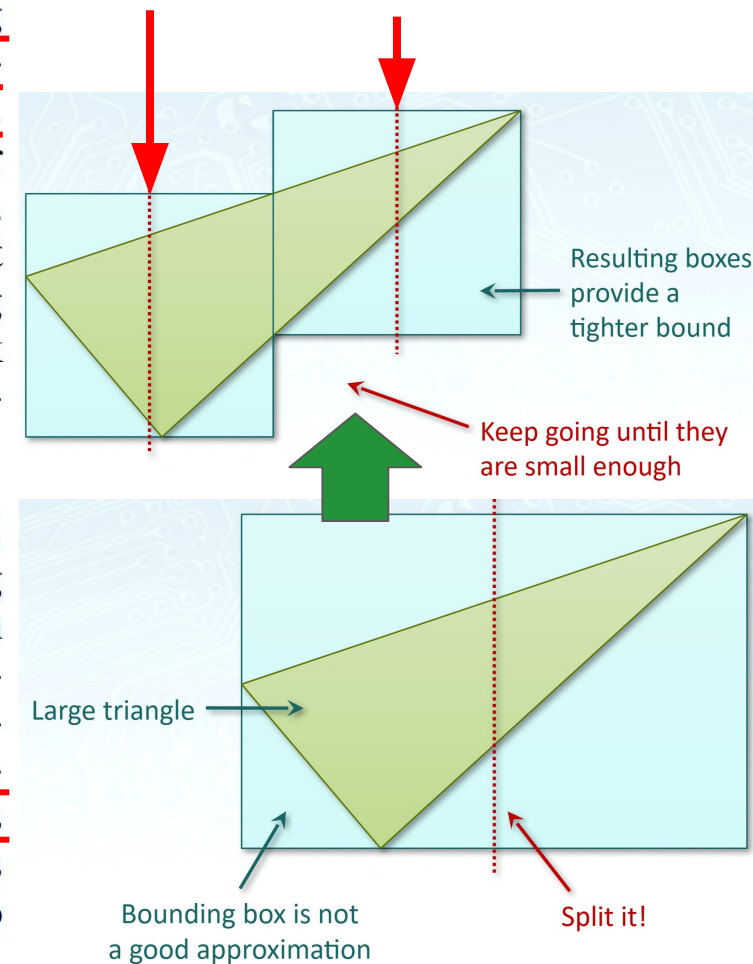
Triangle Splitting Several authors have noted that splitting large triangles can improve the quality of BVHs significantly in scenes that contain large variation in triangle sizes. Ernst and Greiner [2007] propose to split triangles along their longest axis if their surface area exceeds a pre-defined threshold, and Dammertz and Keller [2008] propose similar scheme to split triangle edges based on the volume of their axis-aligned bounding boxes. However, neither of these methods has been proven to work reliably in practice — they can actually end up decreasing the performance in certain scenes.

A better approach, proposed independently by Stich et al. [2009] (SBVH) and Popov et al. [2009], is to incorporate triangle splitting directly into the top-down construction algorithm. SBVH, which yields the highest ray tracing performance to date, works by considering *spatial splits* in addition to conventional partitioning of triangles. Spatial splits correspond to duplicating triangles that intersect a given split plane, so that each resulting *triangle reference* lies strictly on either side of the plane. The choice between spatial splits and triangle partitioning is made on a per-node basis according to which alternative is the most effective in reducing the SAH cost.



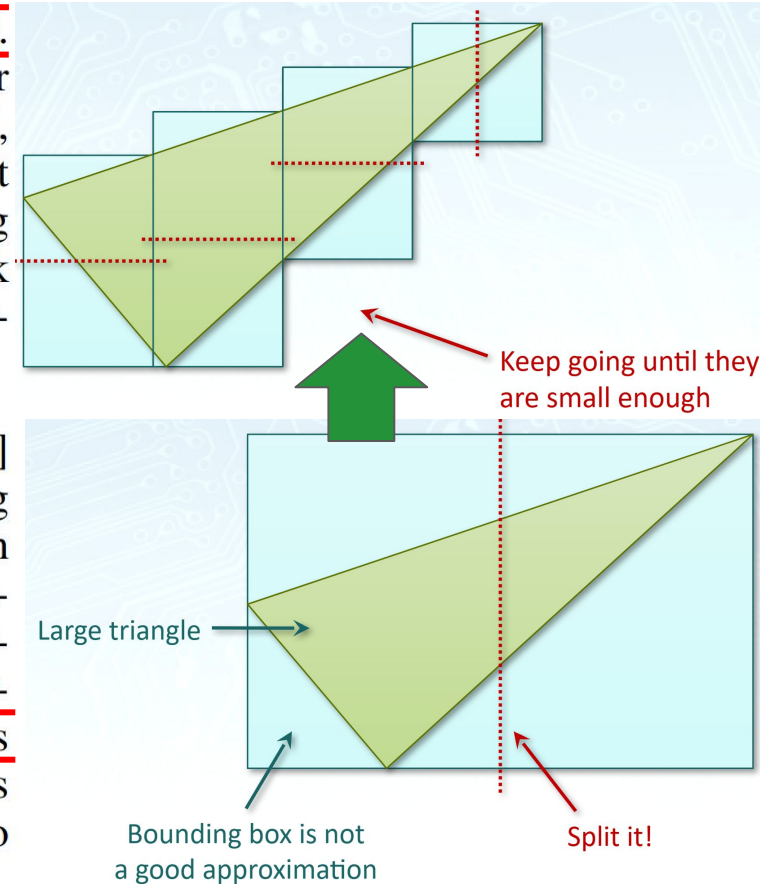
Triangle Splitting Several authors have noted that splitting large triangles can improve the quality of BVHs significantly in scenes that contain large variation in triangle sizes. Ernst and Greiner [2007] propose to split triangles along their longest axis if their surface area exceeds a pre-defined threshold, and Dammertz and Keller [2008] propose similar scheme to split triangle edges based on the volume of their axis-aligned bounding boxes. However, neither of these methods has been proven to work reliably in practice — they can actually end up decreasing the performance in certain scenes.

A better approach, proposed independently by Stich et al. [2009] (SBVH) and Popov et al. [2009], is to incorporate triangle splitting directly into the top-down construction algorithm. SBVH, which yields the highest ray tracing performance to date, works by considering *spatial splits* in addition to conventional partitioning of triangles. Spatial splits correspond to duplicating triangles that intersect a given split plane, so that each resulting *triangle reference* lies strictly on either side of the plane. The choice between spatial splits and triangle partitioning is made on a per-node basis according to which alternative is the most effective in reducing the SAH cost.



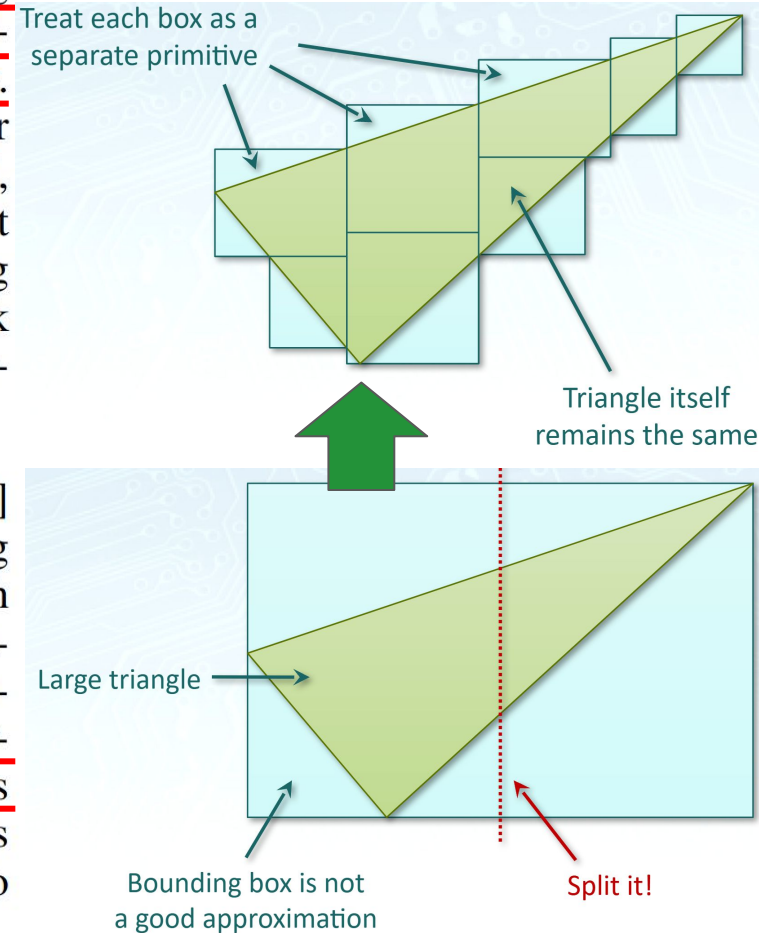
Triangle Splitting Several authors have noted that splitting large triangles can improve the quality of BVHs significantly in scenes that contain large variation in triangle sizes. Ernst and Greiner [2007] propose to split triangles along their longest axis if their surface area exceeds a pre-defined threshold, and Dammertz and Keller [2008] propose similar scheme to split triangle edges based on the volume of their axis-aligned bounding boxes. However, neither of these methods has been proven to work reliably in practice — they can actually end up decreasing the performance in certain scenes.

A better approach, proposed independently by Stich et al. [2009] (SBVH) and Popov et al. [2009], is to incorporate triangle splitting directly into the top-down construction algorithm. SBVH, which yields the highest ray tracing performance to date, works by considering *spatial splits* in addition to conventional partitioning of triangles. Spatial splits correspond to duplicating triangles that intersect a given split plane, so that each resulting *triangle reference* lies strictly on either side of the plane. The choice between spatial splits and triangle partitioning is made on a per-node basis according to which alternative is the most effective in reducing the SAH cost.



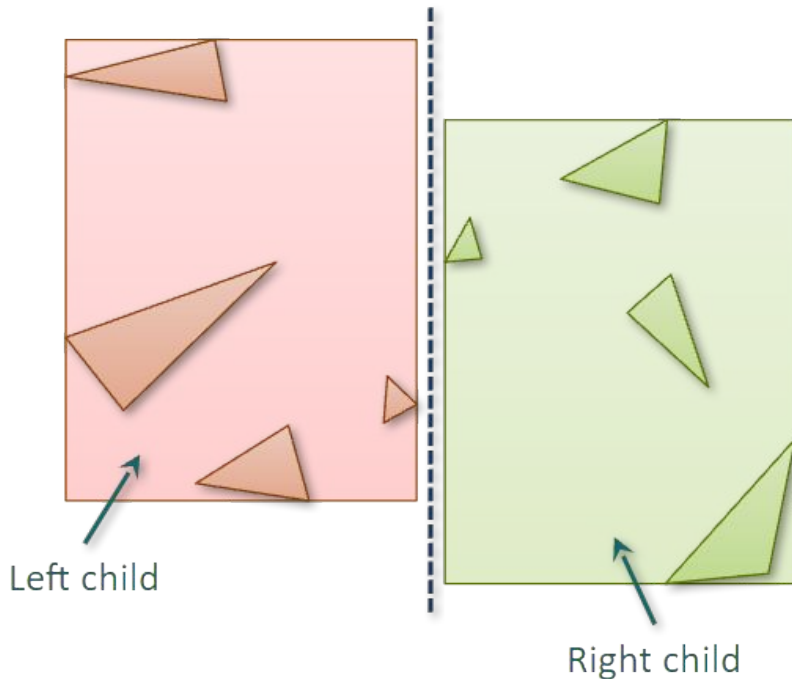
Triangle Splitting Several authors have noted that splitting large triangles can improve the quality of BVHs significantly in scenes that contain large variation in triangle sizes. Ernst and Greiner [2007] propose to split triangles along their longest axis if their surface area exceeds a pre-defined threshold, and Dammertz and Keller [2008] propose similar scheme to split triangle edges based on the volume of their axis-aligned bounding boxes. However, neither of these methods has been proven to work reliably in practice — they can actually end up decreasing the performance in certain scenes.

A better approach, proposed independently by Stich et al. [2009] (SBVH) and Popov et al. [2009], is to incorporate triangle splitting directly into the top-down construction algorithm. SBVH, which yields the highest ray tracing performance to date, works by considering *spatial splits* in addition to conventional partitioning of triangles. Spatial splits correspond to duplicating triangles that intersect a given split plane, so that each resulting *triangle reference* lies strictly on either side of the plane. The choice between spatial splits and triangle partitioning is made on a per-node basis according to which alternative is the most effective in reducing the SAH cost.



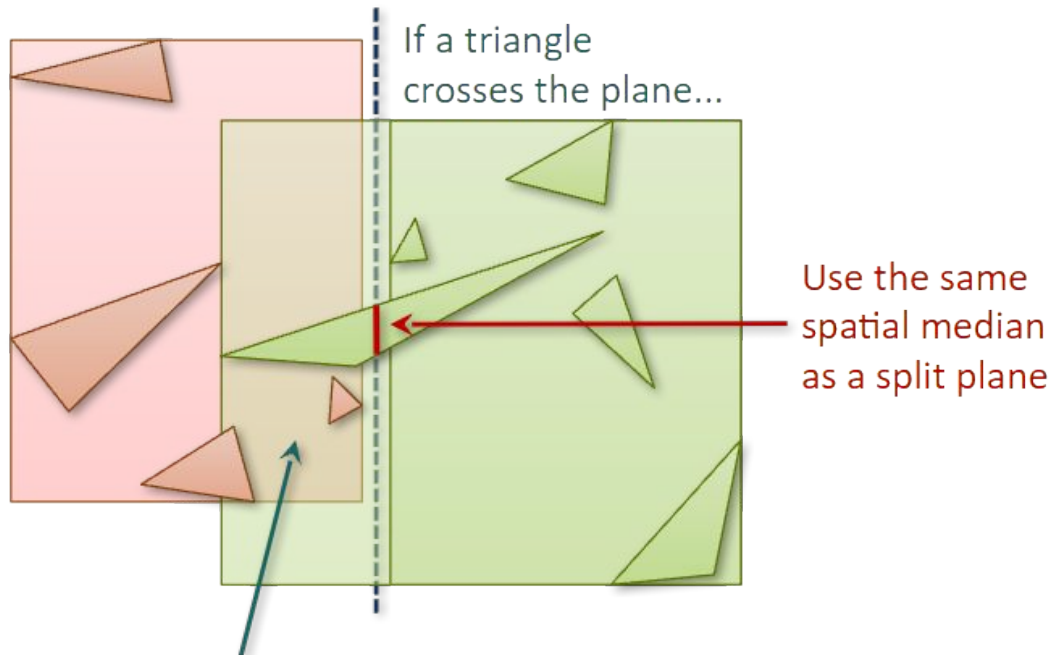
Split plane selection

- Reduce node overlap in the initial BVH



Split plane selection

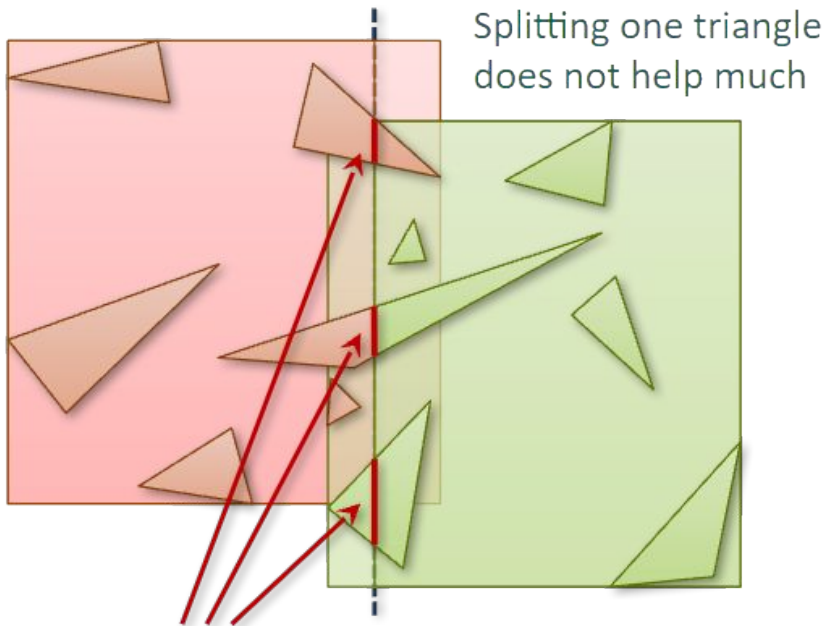
- Reduce node overlap in the initial BVH



...the bounding boxes will overlap

Split plane selection

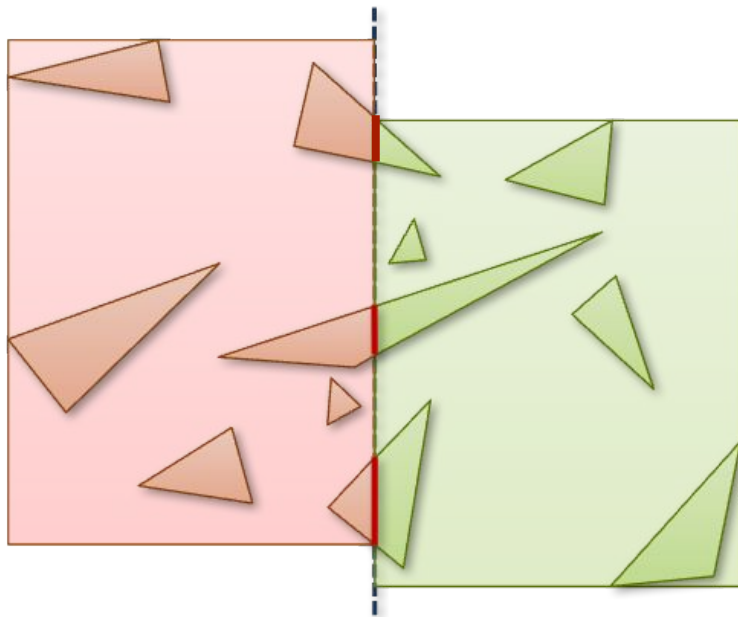
- Reduce node overlap in the initial BVH



Need to split them all
to get the benefits

Split plane selection

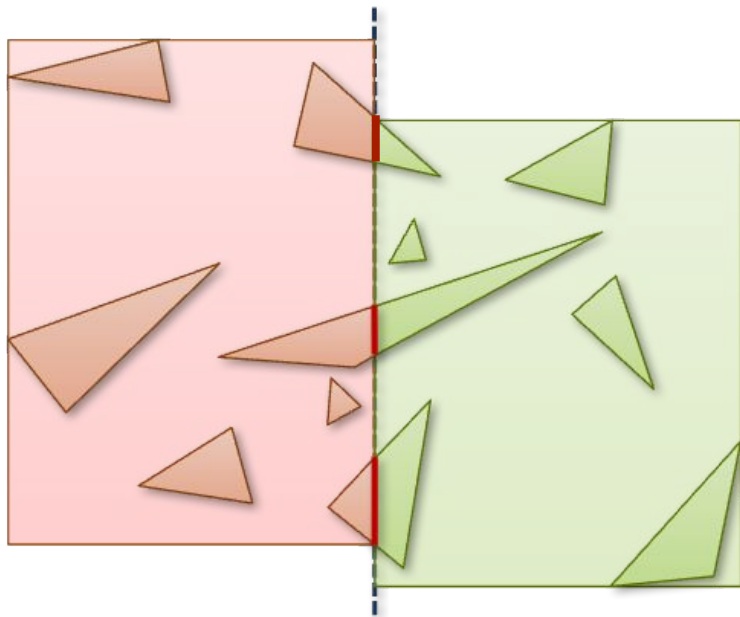
- Reduce node overlap in the initial BVH



Split plane selection

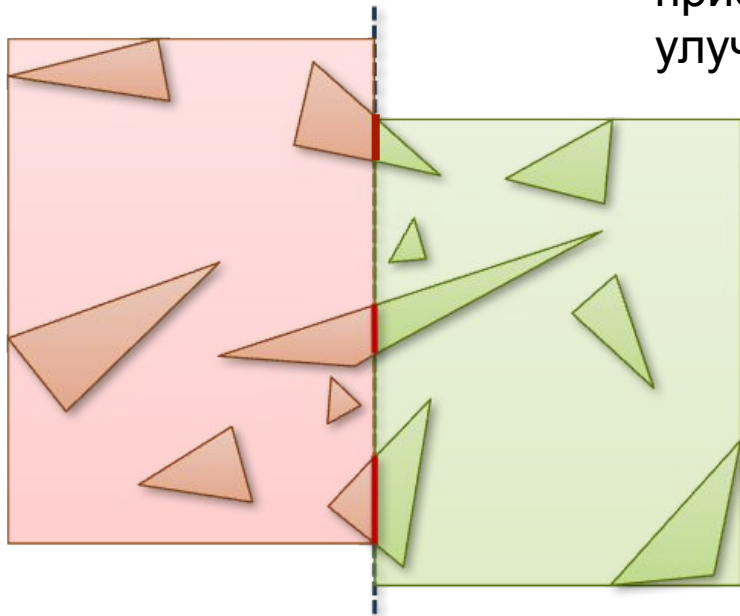
- Reduce node overlap in the initial BVH

Фиксированный бюджет рассечений!
Что делать?



Split plane selection

- Reduce node overlap in the initial BVH



Фиксированный бюджет рассечений!

Рассекаем тех у кого наивысший приоритет - оценка потенциального улучшения **SAH**.



Fast parallel construction of high-quality bounding volume hierarchies

Tero Karras, Timo Aila · Published in [High Performance Graphics](#) 19 July 2013 · Computer Science

158 Citations

Search authors, publications



Date Range

Citation Type

Has PDF

Author

More Filters

Sort by Citation Co...

Sort by Relevance

Sort by Most Influenced Paper

Sort by Citation Count

Sort by Recency

Neural Fields in Visual Computing and Beyond

Yiheng Xie

Towaki Takikawa

+7 authors

Srinath Sridhar

Computer Science, Mathematics · Computer graphics forum (Print)

TLDR A review of the literature on neural fields shows the breadth of topics already covered in visual computing, both historically and in current incarnations, and highlights the improved quality, flexibility, and capability brought by neural field methods. [Expand](#)

696



Save

Embreree

I. Wald

Sven Woop

Carsten Benthin

Gregory S. Johnson

M. Ernst

Computer Science · [ACM Transactions on Graphics](#) 2014

TLDR The design goals and software architecture of Embree are described, and it is shown that for secondary rays in particular, the performance is competitive with (and often higher than) existing state-of-the-art methods on CPUs and GPUs. [Expand](#)

344



Save

Mixture of volumetric primitives for efficient neural rendering

Stephen Lombardi

T. Simon

Gabriel Schwartz

Michael Zollhoefer

Yaser Sheikh

Jason M. Saragih

Computer Science ·

[ACM Transactions on Graphics](#) 2021

TLDR Mixture of Volumetric Primitives (MVP), a representation for rendering dynamic 3D content that combines the completeness of volumetric representations with the efficiency of primitive-based rendering, is presented. [Expand](#)

325



Save

Pulsar: Efficient Sphere-based Neural Rendering

Christoph Lassner

M. Zollhöfer

Computer Science · [Computer Vision and Pattern Recognition](#) 2021

TLDR Pulsar is an efficient sphere-based differentiable rendering module that is orders of magnitude faster than competing techniques, modular, and easy-to-use due to its tight integration with PyTorch, and enables a plethora of applications, ranging from 3D reconstruction to neural rendering. [Expand](#)

196



1 Excerpt Save

Scalable Metropolis Monte Carlo for simulation of hard shapes

Joshua A. Anderson

M. E. Irgang

S. Glotzer

Computer Science, Physics · [Computer Physics Communications](#) 2015

87



1 Excerpt Save

A Survey on Bounding Volume Hierarchies for Ray Tracing

Daniel Meister

Shinji Ogaki

Carsten Benthin

Michael J. Doyle

M. Guthe

Jiří Bittner

Computer Science · Computer graphics forum (Print) 2021

Как читать статьи

- 1) Как найти хоть одну статью-зацепку?

Как читать статьи

- 1) **Как найти хоть одну статью-зацепку?** Популярны **open-source** решения, **лекции**, **слайды**, искать статьи по хоть каким-то **ключевым словам**, спросить кого-то из области (хотя бы смежной).

Как читать статьи

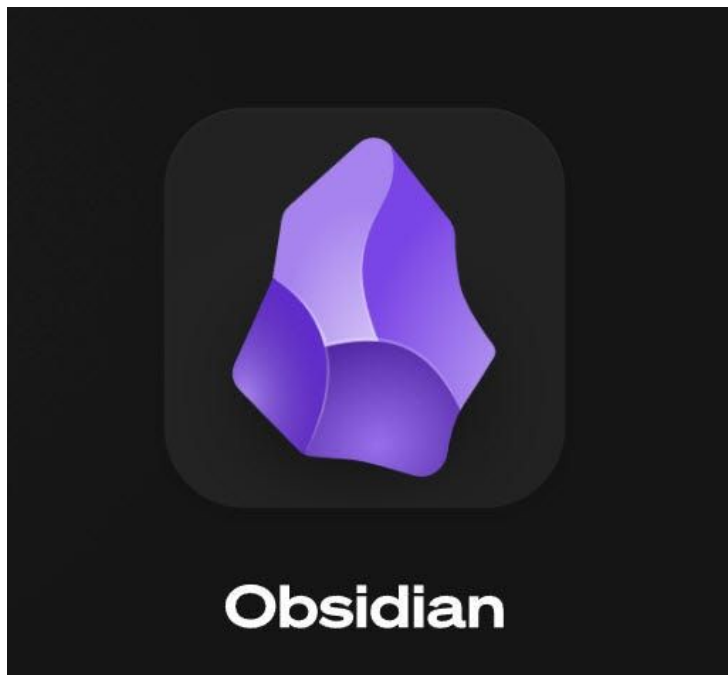
- 1) **Как найти хоть одну статью-зацепку?** Популярные **open-source** решения, **лекции**, **слайды**, искать статьи по хоть каким-то **ключевым словам**, спросить кого-то из области (хотя бы смежной).

Как не растерять зацепки ведь это бесконечная рекурсия, code divergence и ХТОНИЧЕСКИЙ УЖАС?!

Как читать статьи

- 1) **Как найти хоть одну статью-зацепку?** Популярные **open-source** решения, **лекции**, **слайды**, искать статьи по хоть каким-то **ключевым словам**, спросить кого-то из области (хотя бы смежной).

Как не растерять зацепки ведь это бесконечная рекурсия, code divergence и ХТОНИЧЕСКИЙ УЖАС?!



Как читать статьи

- 1) **Как найти хоть одну статью-зацепку?** Популярные **open-source** решения, **лекции**, **слайды**, искать статьи по хоть каким-то **ключевым словам**, спросить кого-то из области (хотя бы смежной).
- 2) **Как по зацепке найти В-С-Ё что есть?**

Как читать статьи

- 1) **Как найти хоть одну статью-зацепку?** Популярны **open-source** решения, **лекции**, **слайды**, искать статьи по хоть каким-то **ключевым словам**, спросить кого-то из области (хотя бы смежной).
- 2) **Как по зацепке найти В-С-Ё что есть?** Через [semanticscholar.org](https://www.semanticscholar.org) путешествовать в прошлое (**references**), в будущее (**citations**), смотреть статьи тех же **авторов**, постепенно уточнять **ключевые слова** поиска.

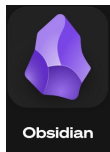
Как читать статьи

- 1) **Как найти хоть одну статью-зацепку?** Популярные **open-source** решения, **лекции**, **слайды**, искать статьи по хоть каким-то **ключевым словам**, спросить кого-то из области (хотя бы смежной).
- 2) **Как по зацепке найти В-С-Ё что есть?** Через [semanticscholar.org](https://www.semanticscholar.org) путешествовать в прошлое (**references**), в будущее (**citations**), смотреть статьи тех же **авторов**, постепенно уточнять **ключевые слова** поиска.

Как не растерять зацепки ведь это бесконечная рекурсия, code divergence и ХТОНИЧЕСКИЙ УЖАС?!

Как читать статьи

- 1) **Как найти хоть одну статью-зацепку?** Популярны **open-source** решения, **лекции**, **слайды**, искать статьи по хоть каким-то **ключевым словам**, спросить кого-то из области (хотя бы смежной).
- 2) **Как по зацепке найти В-С-Ё что есть?** Через semanticscholar.org путешествовать в прошлое (**references**), в будущее (**citations**), смотреть статьи тех же **авторов**, постепенно уточнять **ключевые слова** поиска.



Как не растерять зацепки ведь это бесконечная рекурсия, code divergence и ХТОНИЧЕСКИЙ УЖАС?!

Как читать статьи

- 1) **Как найти хоть одну статью-зацепку?** Популярные **open-source** решения, **лекции**, **слайды**, искать статьи по хоть каким-то **ключевым словам**, спросить кого-то из области (хотя бы смежной).
- 2) **Как по зацепке найти В-С-Ё что есть?** Через [semanticscholar.org](https://www.semanticscholar.org) путешествовать в прошлое (**references**), в будущее (**citations**), смотреть статьи тех же **авторов**, постепенно уточнять **ключевые слова** поиска.
- 3) **На какие статьи обращать внимание в этих путешествиях?**

Как читать статьи

- 1) **Как найти хоть одну статью-зацепку?** Популярны **open-source** решения, **лекции**, **слайды**, искать статьи по хоть каким-то **ключевым словам**, спросить кого-то из области (хотя бы смежной).
- 2) **Как по зацепке найти В-С-Ё что есть?** Через [semanticscholar.org](https://www.semanticscholar.org) путешествовать в прошлое (**references**), в будущее (**citations**), смотреть статьи тех же **авторов**, постепенно уточнять **ключевые слова** поиска.
- 3) **На какие статьи обращать внимание в этих путешествиях?** **Обзорные** (сравнивающие многие методы), **most influenced, most cited**.

Как читать статьи

- 1) **Как найти хоть одну статью-зацепку?** Популярны **open-source** решения, **лекции**, **слайды**, искать статьи по хоть каким-то **ключевым словам**, спросить кого-то из области (хотя бы смежной).
- 2) **Как по зацепке найти В-С-Ё что есть?** Через [semanticscholar.org](https://www.semanticscholar.org) путешествовать в прошлое (**references**), в будущее (**citations**), смотреть статьи тех же **авторов**, постепенно уточнять **ключевые слова** поиска.
- 3) **На какие статьи обращать внимание в этих путешествиях?** **Обзорные** (сравнивающие многие методы), **most influenced**, **most cited**.
- 4) **Как справиться с ситуацией “не могу понять статью”?**

Как читать статьи

- 1) **Как найти хоть одну статью-зацепку?** Популярные **open-source** решения, **лекции**, **слайды**, искать статьи по хоть каким-то **ключевым словам**, спросить кого-то из области (хотя бы смежной).
- 2) **Как по зацепке найти В-С-Ё что есть?** Через [semanticscholar.org](https://www.semanticscholar.org) путешествовать в прошлое (**references**), в будущее (**citations**), смотреть статьи тех же **авторов**, постепенно уточнять **ключевые слова** поиска.
- 3) **На какие статьи обращать внимание в этих путешествиях?** **Обзорные** (сравнивающие многие методы), **most influenced**, **most cited**.
- 4) **Как справиться с ситуацией “не могу понять статью”?** Прыгнуть в прошлое на **most influenced** предтечи, там часто подробнее и проще описано. Поискать **open-source** реализацию (иногда код понять проще), поискать **слайды/презентацию/лекцию**.

Как читать статьи

- 1) **Как найти хоть одну статью-зацепку?** Популярные **open-source** решения, **лекции**, **слайды**, искать статьи по хоть каким-то **ключевым словам**, спросить кого-то из области (хотя бы смежной).
- 2) **Как по зацепке найти В-С-Ё что есть?** Через semanticscholar.org путешествовать в прошлое (**references**), в будущее (**citations**), смотреть статьи тех же **авторов**, постепенно уточнять **ключевые слова** поиска.
- 3) **На какие статьи обращать внимание в этих путешествиях?** **Обзорные** (сравнивающие многие методы), **most influenced**, **most cited**.
- 4) **Как справиться с ситуацией “не могу понять статью”?** Прыгнуть в прошлое на **most influenced** предтечи, там часто подробнее и проще описано. Поискать **open-source** реализацию (иногда код понять проще), поискать **слайды/презентацию/лекцию**.
- 5) **Как не упустить ни одной идеи автора?**

Как читать статьи

- 1) **Как найти хоть одну статью-зацепку?** Популярны **open-source** решения, **лекции**, **слайды**, искать статьи по хоть каким-то **ключевым словам**, спросить кого-то из области (хотя бы смежной).
- 2) **Как по зацепке найти В-С-Ё что есть?** Через [semanticscholar.org](https://www.semanticscholar.org) путешествовать в прошлое (**references**), в будущее (**citations**), смотреть статьи тех же **авторов**, постепенно уточнять **ключевые слова** поиска.
- 3) **На какие статьи обращать внимание в этих путешествиях?** **Обзорные** (сравнивающие многие методы), **most influenced**, **most cited**.
- 4) **Как справиться с ситуацией “не могу понять статью”?** Прыгнуть в прошлое на **most influenced** предтечи, там часто подробнее и проще описано. Поискать **open-source** реализацию (иногда код понять проще), поискать **слайды/презентацию/лекцию**.
- 5) **Как не упустить ни одной идеи автора?** Презумпция таланта и экономии места. Если что-то в статье - в этом есть смысл, если не ясно какой - значит **что-то упускаете**.

Как

1)

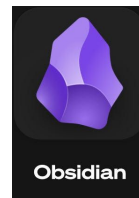
2)

3)

4)



- 5) **Как не упустить ни одной идеи автора?** Презумпция таланта и экономии места. Если что-то в статье - в этом есть смысл, если не ясно какой - значит **что-то упускаете**.

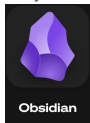


Как читать статьи

- 1) **Как найти хоть одну статью-зацепку?** Популярны **open-source** решения, **лекции**, **слайды**, искать статьи по хоть каким-то **ключевым словам**, спросить кого-то из области (хотя бы смежной).
- 2) **Как по зацепке найти В-С-Ё что есть?** Через semanticscholar.org путешествовать в прошлое (**references**), в будущее (**citations**), смотреть статьи тех же **авторов**, постепенно уточнять **ключевые слова** поиска.
- 3) **На какие статьи обращать внимание в этих путешествиях?** **Обзорные** (сравнивающие многие методы), **most influenced**, **most cited**.
- 4) **Как справиться с ситуацией “не могу понять статью”?** Прыгнуть в прошлое на **most influenced** предтечи, там часто подробнее и проще описано. Поискать **open-source** реализацию (иногда код понять проще), поискать **слайды/презентацию/лекцию**.
- 5) **Как не упустить ни одной идеи автора?** Презумпция таланта и экономии места. Если что-то в статье - в этом есть смысл, если не ясно какой - значит **что-то упускаете**.
- 6) **Как не упустить ни одной своей идеи?**

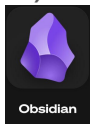
Как читать статьи

- 1) **Как найти хоть одну статью-зацепку?** Популярны **open-source** решения, **лекции**, **слайды**, искать статьи по хоть каким-то **ключевым словам**, спросить кого-то из области (хотя бы смежной).
- 2) **Как по зацепке найти В-С-Ё что есть?** Через semanticscholar.org путешествовать в прошлое (**references**), в будущее (**citations**), смотреть статьи тех же **авторов**, постепенно уточнять **ключевые слова** поиска.
- 3) **На какие статьи обращать внимание в этих путешествиях?** **Обзорные** (сравнивающие многие методы), **most influenced, most cited**.
- 4) **Как справиться с ситуацией “не могу понять статью”?** Прыгнуть в прошлое на **most influenced** предтечи, там часто подробнее и проще описано. Поискать **open-source** реализацию (иногда код понять проще), поискать **слайды/презентацию/лекцию**.
- 5) **Как не упустить ни одной идеи автора?** Презумпция таланта и экономии места. Если что-то в статье - в этом есть смысл, если не ясно какой - значит **что-то упускаете**.
- 6) **Как не упустить ни одной своей идеи? Записывать!** Появилась идея “а что было бы если?” - запишите чтобы попробовать. Появился вопрос “а почему здесь сказано/выбрано такое-то?” - запишите чтобы найти ответ в другой статье.



Как читать статьи

- 1) Как найти хоть одну статью-зацепку? Популярные **open-source** решения, лекции, слайды, искать статьи по хоть каким-то **ключевым словам**, спросить кого-то из области (хотя бы смежной).
- 2) Как по зацепке найти **В-С-Ё** что есть? Через semanticscholar.org путешествовать в прошлое (**references**), в будущее (**citations**), смотреть статьи тех же **авторов**, постепенно уточнять **ключевые слова** поиска.
- 3) На какие статьи обращать внимание в этих путешествиях? **Обзорные** (сравнивающие многие методы), **most influenced, most cited**.
- 4) Как справиться с ситуацией “не могу понять статью”? Прыгнуть в прошлое на **most influenced** предтечи, там часто подробнее и проще описано. Поискать **open-source** реализацию (иногда код понять проще), поискать **слайды/презентацию/лекцию**.
- 5) Как не упустить ни одной идеи автора? Презумпция таланта и экономии места. Если что-то в статье - в этом есть смысл, если не ясно какой - значит **что-то упускаете**.
- 6) Как не упустить ни одной своей идеи? **Записывать!** Появилась идея “а что было бы если?” - запишите чтобы попробовать. Появился вопрос “а почему здесь сказано/выбрано такое-то?” - запишите чтобы найти ответ в другой статье.



Глава 5: H-PLOC



H-PLOC Hierarchical Parallel Locally-Ordered Clustering for BVH Construction

H-PLOC

Hierarchical Parallel Locally-Ordered Clustering for Bounding Volume Hierarchy Construction

CARSTEN BENTHIN, Advanced Micro Devices, Inc., Germany

DANIEL MEISTER, Advanced Micro Devices, Inc., Japan

JOSHUA BARCZAK, Advanced Micro Devices, Inc., USA

ROHAN MEHALWAL, Advanced Micro Devices, Inc., USA

JOHN TSAKOK, Advanced Micro Devices, Inc., USA

ANDREW KENSLER, Advanced Micro Devices, Inc., USA

We propose a novel GPU-oriented approach for constructing binary bounding volume hierarchies (BVHs) based on the parallel locally-ordered clustering (*PLOC/PLOC++*) algorithm. Compared to competing high-performance GPU BVH build algorithms (*PLOC++* or *ATRBVH*), our method provides similar BVH quality in just a single kernel launch while achieving 1.1-3.6 $\times$ lower construction times for the entire BVH build and 1.6-13 $\times$ lower for just the binary BVH construction phase. Additionally, we propose an efficient algorithm to convert a binary BVH to an n -wide BVH with just a single kernel launch. Besides being extremely efficient, our proposed algorithms are simple to implement, allowing easy integration into existing frameworks.

H-PLOC Hierarchical Parallel Locally-Ordered Clustering for BVH Construction

	Misc [ms]	Quad [ms]	Sort [ms]	BVH2 [ms]	Conv [ms]	Total [ms]	Build Perf MTris/s	BVH4 SAH PT	
Sponza 0.3M Triangles									
PRBVH	0.10 (<0.1%)	0.19 (<0.1%)	0.09 (<0.1%)	632 (99.9%)	0.19 (<0.1%)	633	0.4 (< 0.01)	1.00	1.00
ATRBVH	0.09 (3.4%)	0.19 (7.2%)	0.09 (3.8%)	2.09 (78.9%)	0.18 (6.8%)	2.65	105 (0.25)	1.10	1.11
PLOC++	0.12 (9%)	0.19 (15.0%)	0.09 (6.8%)	0.71 (53.4%)	0.20 (15.8%)	1.33	208 (0.49)	1.11	1.10
<u>LBVH</u>	0.09 (13.6%)	0.19 (30.3%)	0.09 (13.6%)	0.08 (12.1%)	0.19 (30.3%)	0.66	422 (1.0)	1.33	1.28
<u>H-PLOC</u>	0.10 (14.9%)	0.19 (25.7%)	0.09 (12.2%)	0.16 (21.6%)	0.20 (25.7%)	0.74	377 (0.89)	1.09	1.11
Powerplant 12.7M Triangles									
PRBVH	0.81 (<0.1%)	2.5 (<0.1%)	1.09 (<0.1%)	38007 (99.9%)	5.59 (<0.1%)	38017	0.3 (< 0.01)	1.00	1.00
ATRBVH	0.77 (2.2%)	2.5 (7%)	1.09 (2.2%)	25.38 (72%)	5.54 (15.8%)	35.23	362 (0.32)	1.11	1.10
PLOC++	0.98 (6.8%)	2.5 (17.3%)	1.09 (7.6%)	4.44 (31%)	5.32 (37.2%)	14.31	892 (0.80)	1.12	1.11
<u>LBVH</u>	0.76 (6.6%)	2.5 (21.8%)	1.09 (9.4%)	1.86 (16.2%)	5.26 (45.9%)	11.46	1112 (1.0)	1.38	1.30
<u>H-PLOC</u>	0.98 (7.7%)	2.5 (19.7%)	1.09 (8.5%)	2.75 (21.6%)	5.39 (42.4%)	12.71	1004 (0.90)	1.12	1.13

H-PLOC Hierarchical Parallel Locally-Ordered Clustering for BVH Construction

	Misc [ms]	Quad [ms]	Sort [ms]	BVH2 [ms]	Conv [ms]	Total [ms]	Build Perf MTris/s	BVH4 SAH PT	
Sponza 0.3M Triangles									
PRBVH	0.10 (<0.1%)	0.19 (<0.1%)	0.09 (<0.1%)	632 (99.9%)	0.19 (<0.1%)	633	0.4 (< 0.01)	1.00	1.00
ATRBVH	0.09 (3.4%)	0.19 (7.2%)	0.09 (3.8%)	2.09 (78.9%)	0.18 (6.8%)	2.65	105 (0.25)	1.10	1.11
PLOC++	0.12 (9%)	0.19 (15.0%)	0.09 (6.8%)	0.71 (53.4%)	0.20 (15.8%)	1.33	208 (0.49)	1.11	1.10
<u>LBVH</u>	0.09 (13.6%)	0.19 (30.3%)	0.09 (13.6%)	0.08 (12.1%)	0.19 (30.3%)	0.66	422 (1.0)	1.33	1.28
<u>H-PLOC</u>	0.10 (14.9%)	0.19 (25.7%)	0.09 (12.2%)	0.16 (21.6%)	0.20 (25.7%)	0.74	377 (0.89)	1.09	1.11
Powerplant 12.7M Triangles									
PRBVH	0.81 (<0.1%)	2.5 (<0.1%)	1.09 (<0.1%)	38007 (99.9%)	5.59 (<0.1%)	38017	0.3 (< 0.01)	1.00	1.00
ATRBVH	0.77 (2.2%)	2.5 (7%)	1.09 (2.2%)	25.38 (72%)	5.54 (15.8%)	35.23	362 (0.32)	1.11	1.10
PLOC++	0.98 (6.8%)	2.5 (17.3%)	1.09 (7.6%)	4.44 (31%)	5.32 (37.2%)	14.31	892 (0.80)	1.12	1.11
<u>LBVH</u>	0.76 (6.6%)	2.5 (21.8%)	1.09 (9.4%)	1.86 (16.2%)	5.26 (45.9%)	11.46	1112 (1.0)	1.38	1.30
<u>H-PLOC</u>	0.98 (7.7%)	2.5 (19.7%)	1.09 (8.5%)	2.75 (21.6%)	5.39 (42.4%)	12.71	1004 (0.90)	1.12	1.13

H-PLOC Hierarchical Parallel Locally-Ordered Clustering for BVH Construction

	Misc [ms]	Quad [ms]	Sort [ms]	BVH2 [ms]	Conv [ms]	Total [ms]	Build Perf MTris/s	BVH4 SAH PT	
Sponza 0.3M Triangles									
<u>PRBVH</u>	0.10 (<0.1%)	0.19 (<0.1%)	0.09 (<0.1%)	632 (99.9%)	0.19 (<0.1%)	633	0.4 (< 0.01)	1.00	1.00
ATRBVH	0.09 (3.4%)	0.19 (7.2%)	0.09 (3.8%)	2.09 (78.9%)	0.18 (6.8%)	2.65	105 (0.25)	1.10	1.11
PLOC++	0.12 (9%)	0.19 (15.0%)	0.09 (6.8%)	0.71 (53.4%)	0.20 (15.8%)	1.33	208 (0.49)	1.11	1.10
<u>LBVH</u>	0.09 (13.6%)	0.19 (30.3%)	0.09 (13.6%)	0.08 (12.1%)	0.19 (30.3%)	0.66	422 (1.0)	1.33	1.28
<u>H-PLOC</u>	0.10 (14.9%)	0.19 (25.7%)	0.09 (12.2%)	0.16 (21.6%)	0.20 (25.7%)	0.74	377 (0.89)	1.09	1.11
Powerplant 12.7M Triangles									
<u>PRBVH</u>	0.81 (<0.1%)	2.5 (<0.1%)	1.09 (<0.1%)	38007 (99.9%)	5.59 (<0.1%)	38017	0.3 (< 0.01)	1.00	1.00
ATRBVH	0.77 (2.2%)	2.5 (7%)	1.09 (2.2%)	25.38 (72%)	5.54 (15.8%)	35.23	362 (0.32)	1.11	1.10
PLOC++	0.98 (6.8%)	2.5 (17.3%)	1.09 (7.6%)	4.44 (31%)	5.32 (37.2%)	14.31	892 (0.80)	1.12	1.11
<u>LBVH</u>	0.76 (6.6%)	2.5 (21.8%)	1.09 (9.4%)	1.86 (16.2%)	5.26 (45.9%)	11.46	1112 (1.0)	1.38	1.30
<u>H-PLOC</u>	0.98 (7.7%)	2.5 (19.7%)	1.09 (8.5%)	2.75 (21.6%)	5.39 (42.4%)	12.71	1004 (0.90)	1.12	1.13

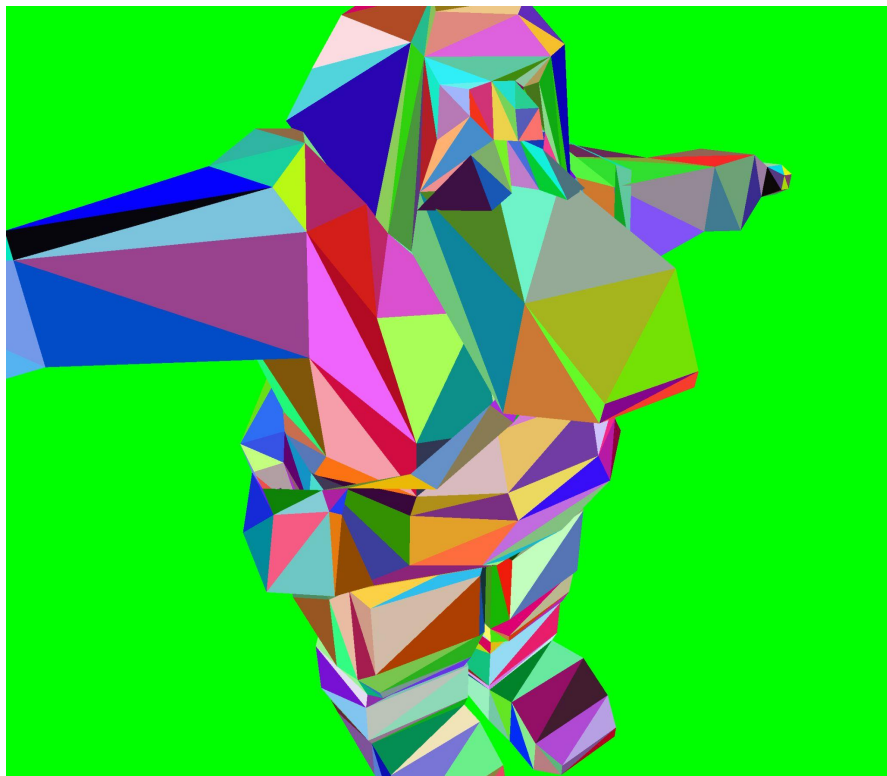
5 Conclusion and Future Work

H-PLOC uses *LBVH* bottom-up construction as a guide and combines it with efficient *PLOC++*-based merging of wave-sized lists of clusters. This results in tremendous efficiency advantages, as *H-PLOC* outperforms competing binary BVH build algorithms by several factors while providing similar BVH quality. Considering the total BVH build cost, including all phases, the overhead of *H-PLOC* to the speed-of-light reference *LBVH* is only about 15%, while offering significantly better BVH quality. Combined with its implementation simplicity, which we believe is a strong argument for adoption, and the efficient binary to n -wide BVH conversion, *H-PLOC* could be the algorithm of choice for high-quality real-time BVH construction. In the future, we are interested in incorporating insertion-based refinement in the binary BVH construction phase to further lower the quality gap to *PRBVH*.

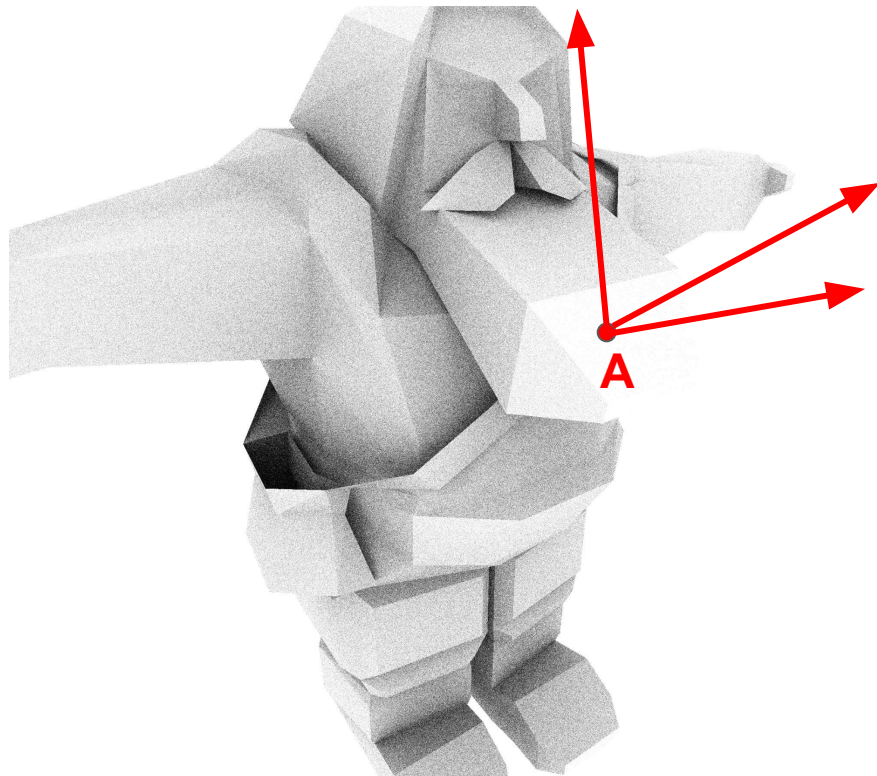
5 Conclusion and Future Work

H-PLOC uses *LBVH* bottom-up construction as a guide and combines it with efficient *PLOC++*-based merging of wave-sized lists of clusters. This results in tremendous efficiency advantages, as *H-PLOC* outperforms competing binary BVH build algorithms by several factors while providing similar BVH quality. Considering the total BVH build cost, including all phases, the overhead of *H-PLOC* to the speed-of-light reference *LBVH* is only about 15%, while offering significantly better BVH quality. Combined with its implementation simplicity, which we believe is a strong argument for adoption, and the efficient binary to n -wide BVH conversion, *H-PLOC* could be the algorithm of choice for high-quality real-time BVH construction. In the future, we are interested in incorporating insertion-based refinement in the binary BVH construction phase to further lower the quality gap to *PRBVH*.

Задание: ускорить трассировку лучей через **LBVH**

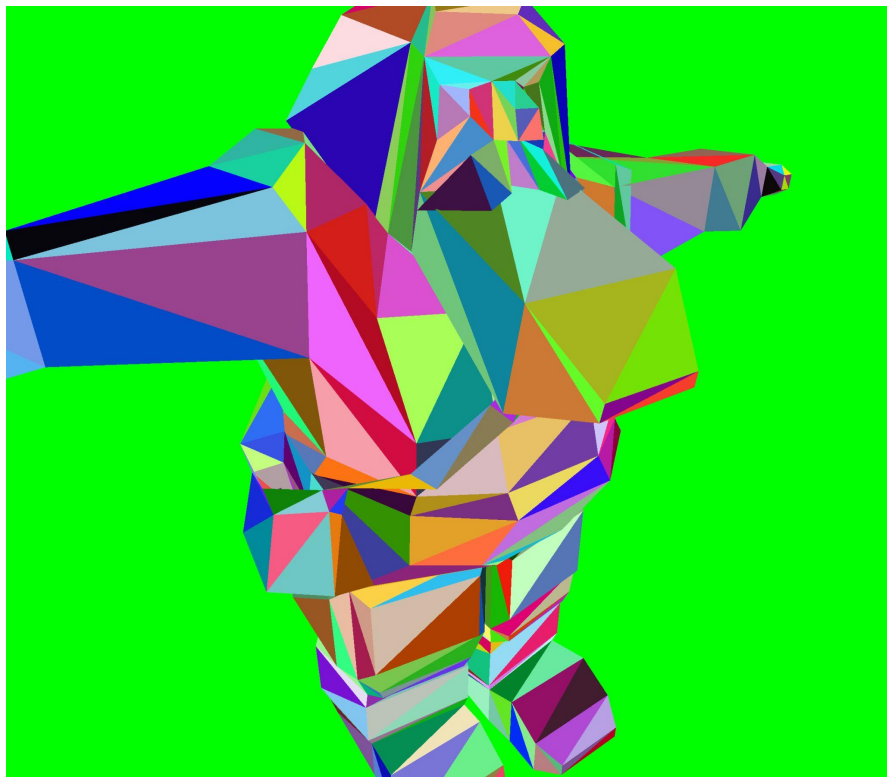


Face Index Framebuffer

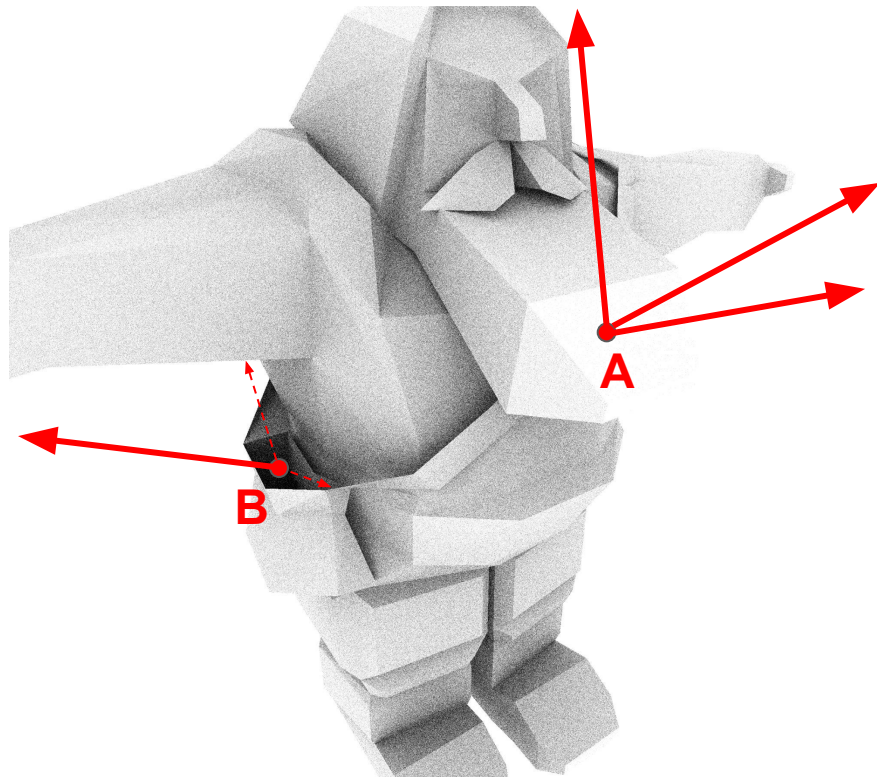


Ambient Occlusion

Задание: ускорить трассировку лучей через **LBVH**

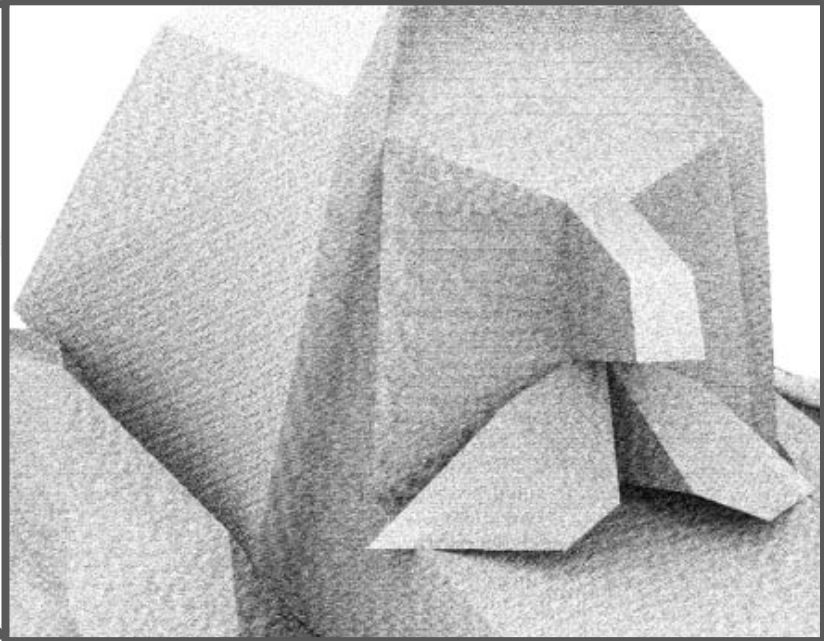
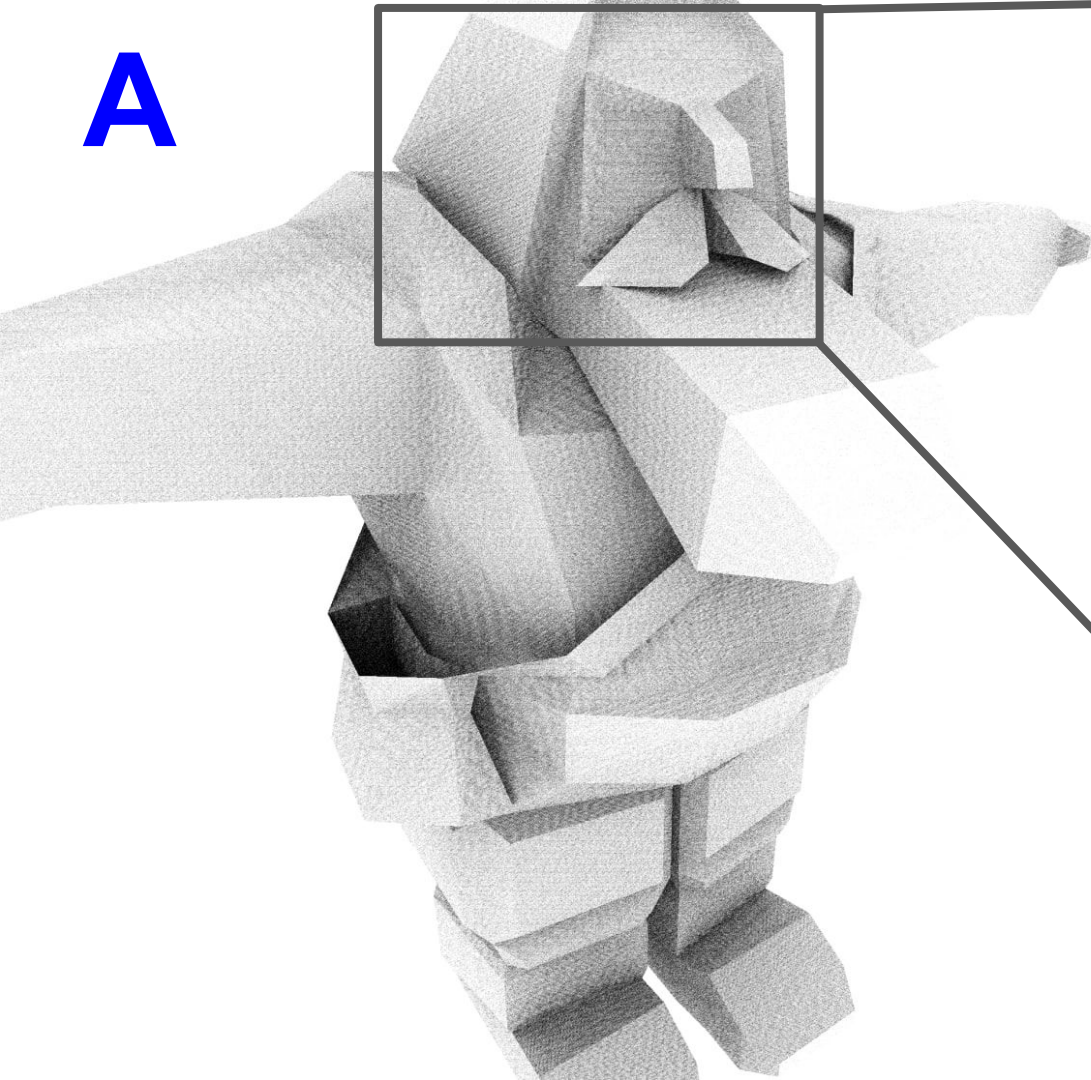


Face Index Framebuffer



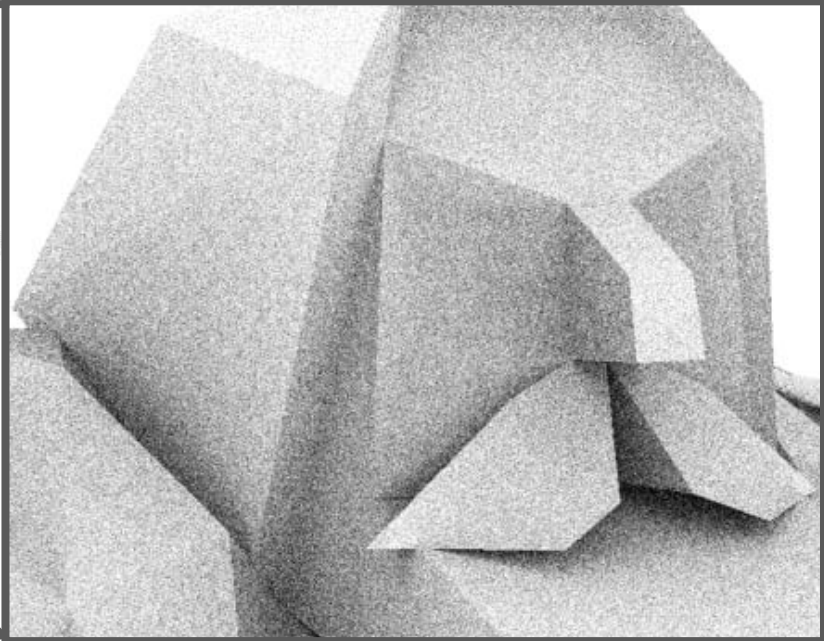
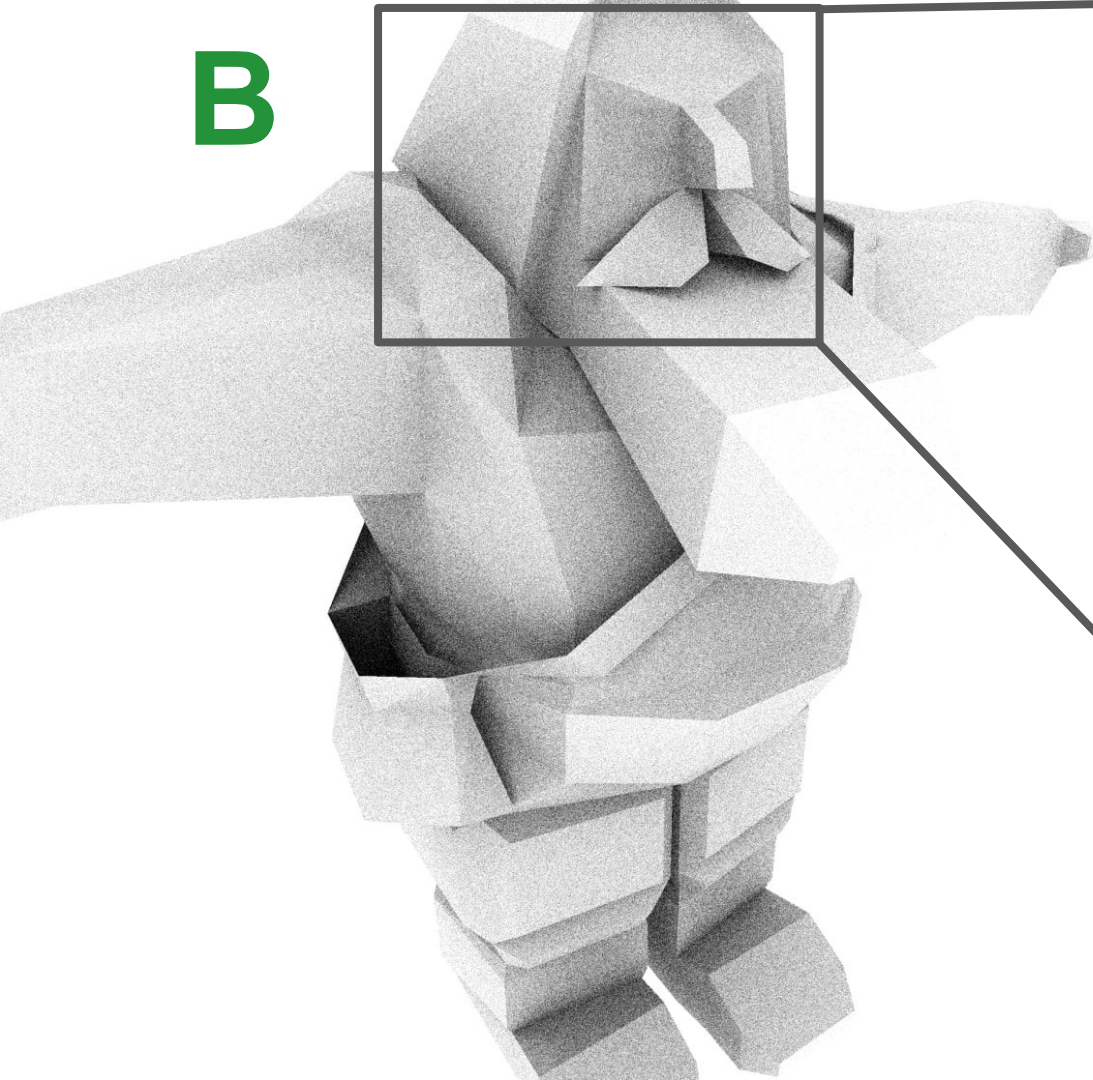
Ambient Occlusion

A



Чем отличаются **A** и **B**?

В



Чем отличаются **А** и **В**?



Вопросы?

 [@UnicornGlade](https://t.me/UnicornGlade)

 [@PolarNick239](https://t.me/PolarNick239)

 polarnick239@gmail.com

 Николай Полярный



CS Space

Клуб технологий и науки