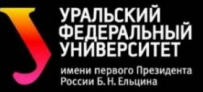
Возможности ускорения GiST: патчи, хаки, твики

GiST speedup: patches, hacks and tweaks

Andrey Borodin

Software engineer at Octonica

PhD, Associate Professor at URFU



Multidimensional access methods

Windows search

select *
from table
where table.point
is inside some region

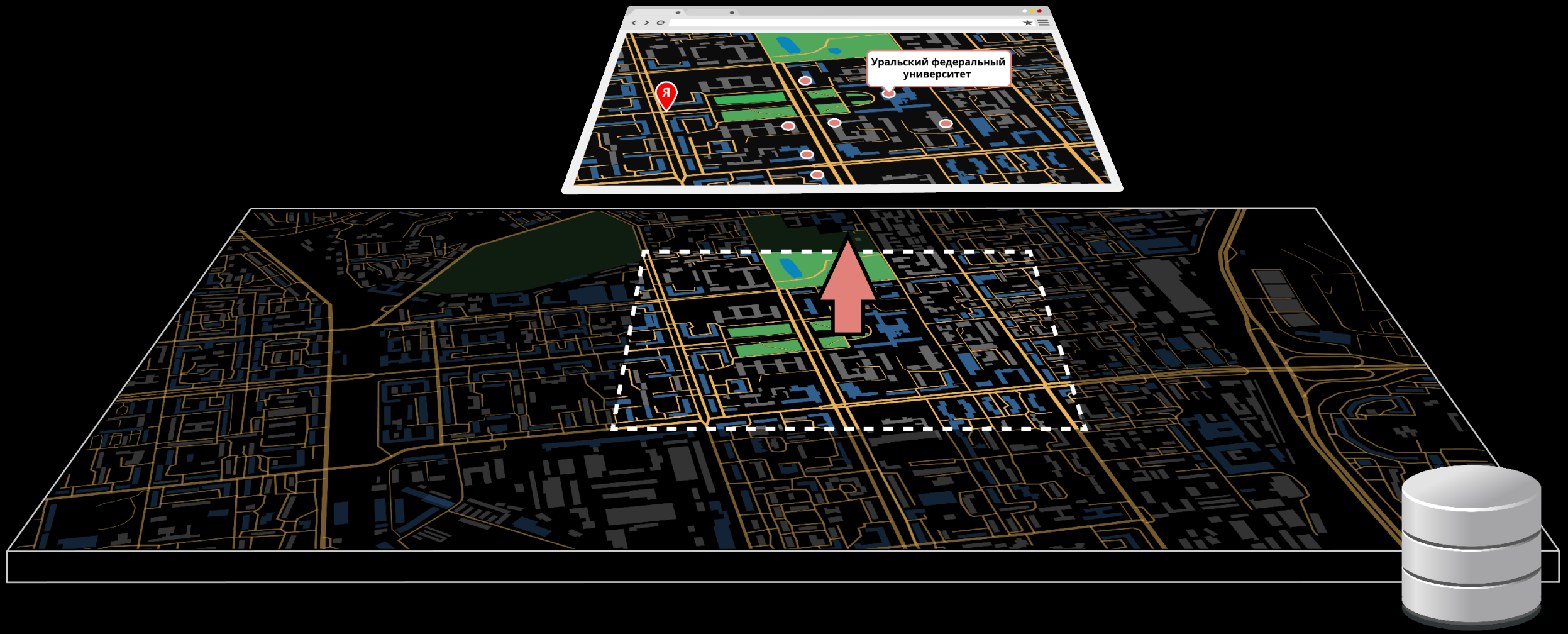
Aggregation

select aggregate(column)
from table
where table.point
is inside some region

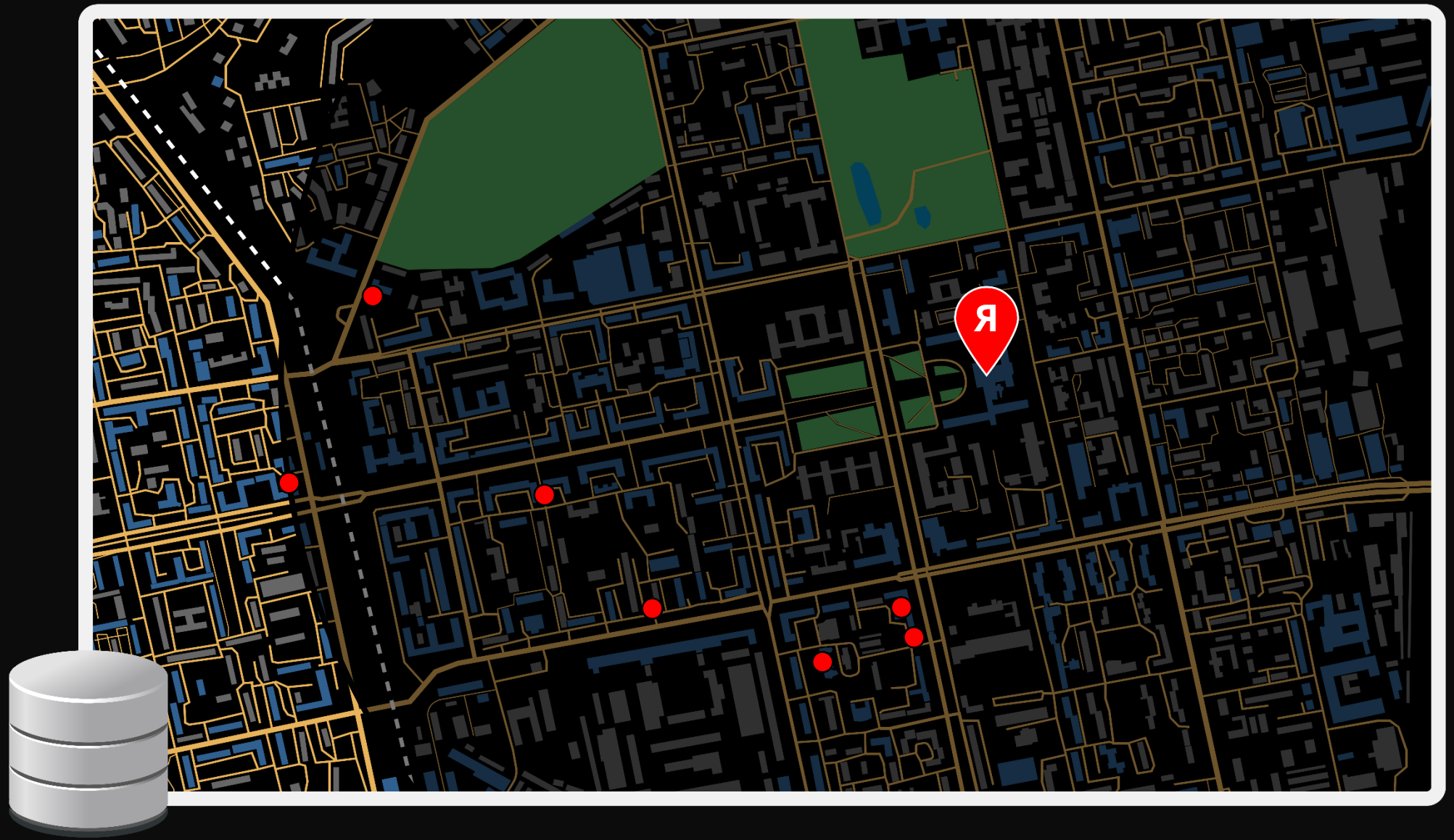
kNN (k Nearest Newburgh)

select *
from table
order by distance from some point
limit N

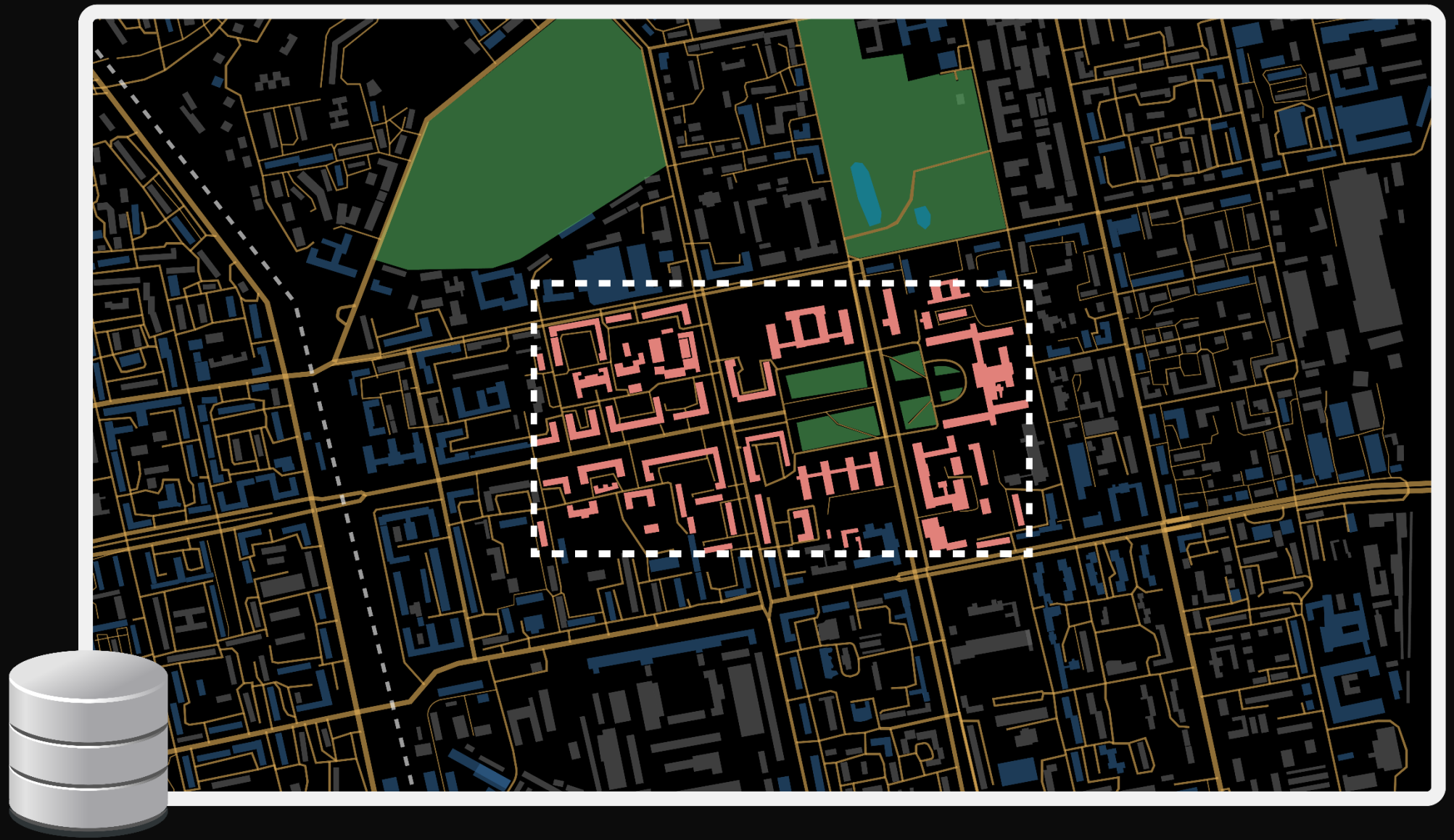
Windows search



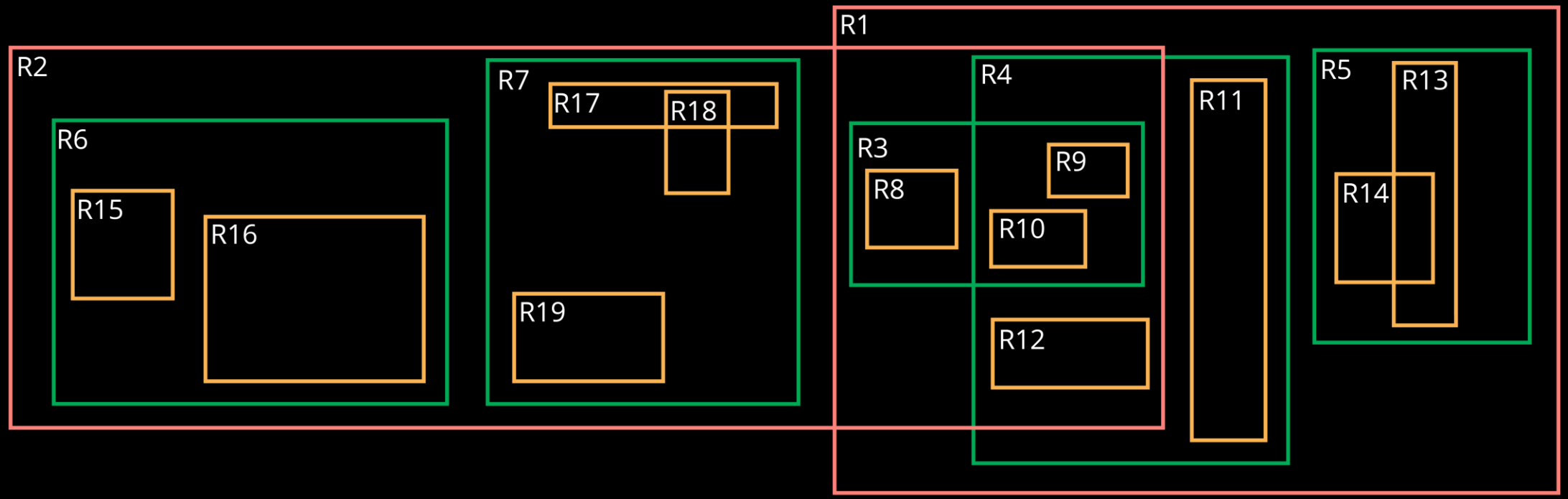
kNN search

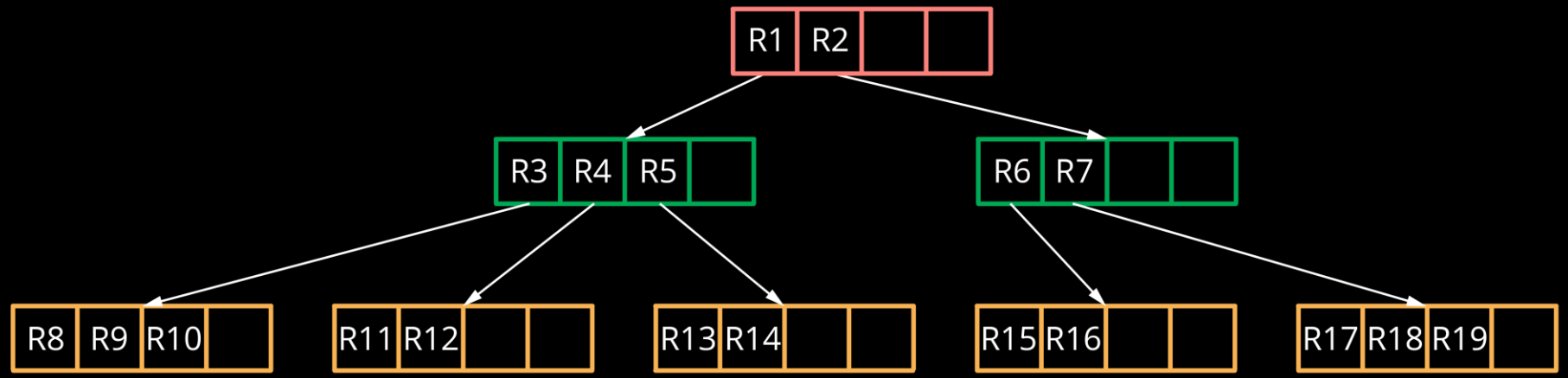


Aggregate Query (OLAP)

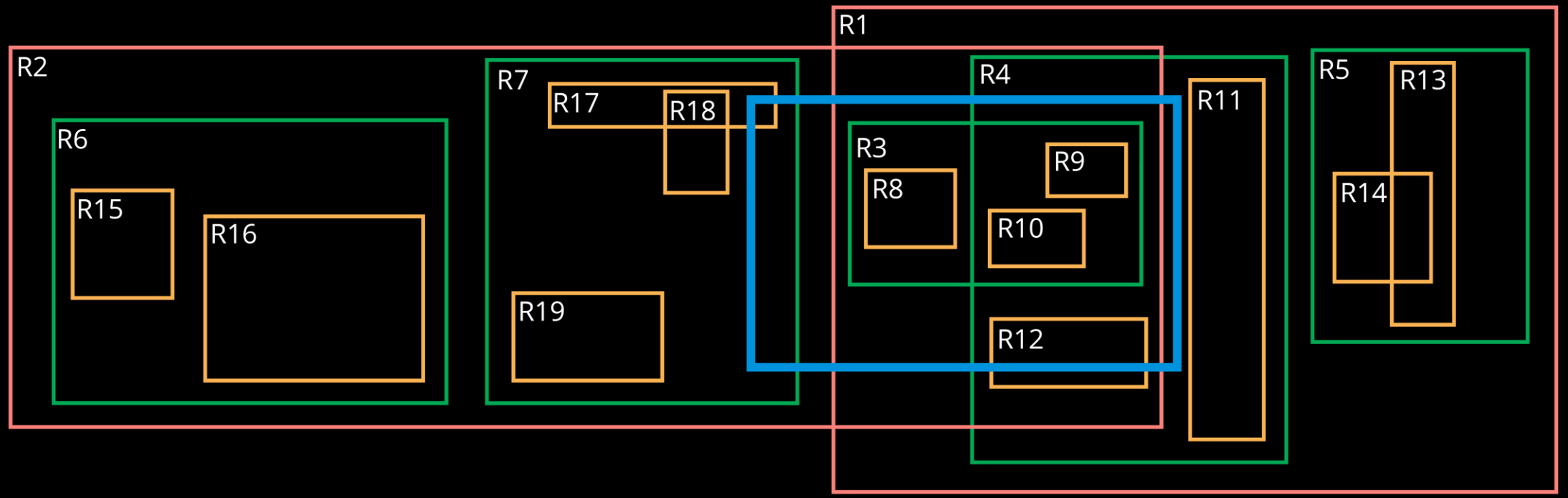


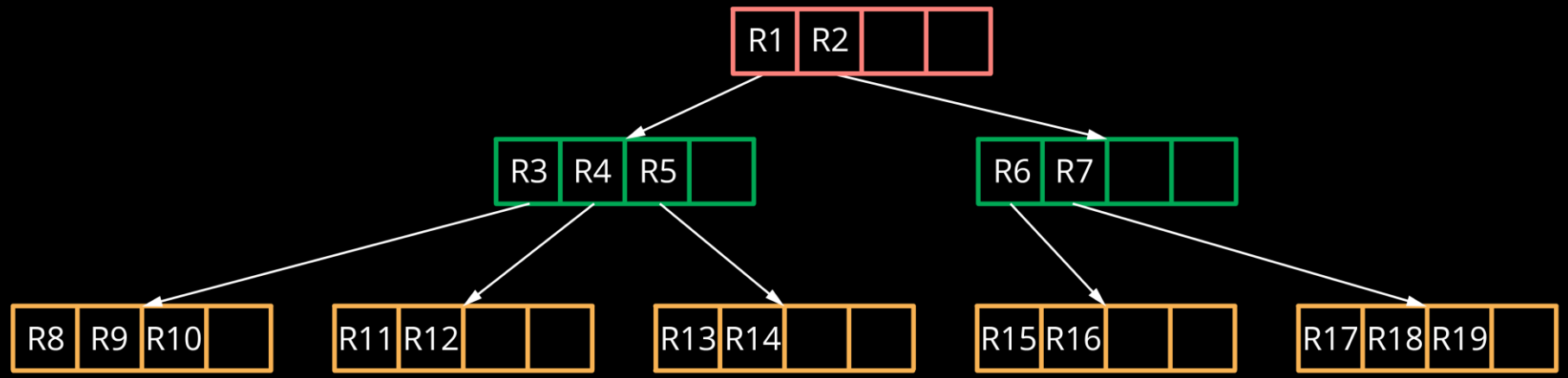
R-tree


R1
R2
R6
R15
R16
R7
R17
R18
R19
R4
R3
R8
R9
R10
R12
R11
R5
R13
R14

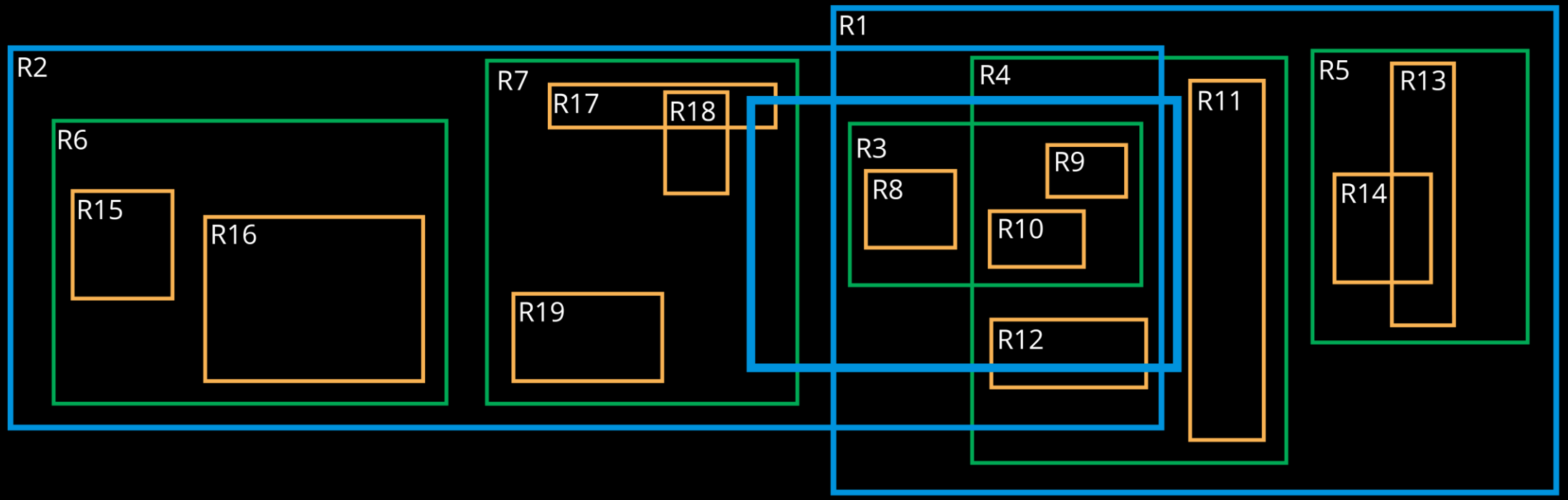

R1 R2
R3 R4 R5
R6 R7
R8 R9 R10
R11 R12
R13 R14
R15 R16
R17 R18 R19

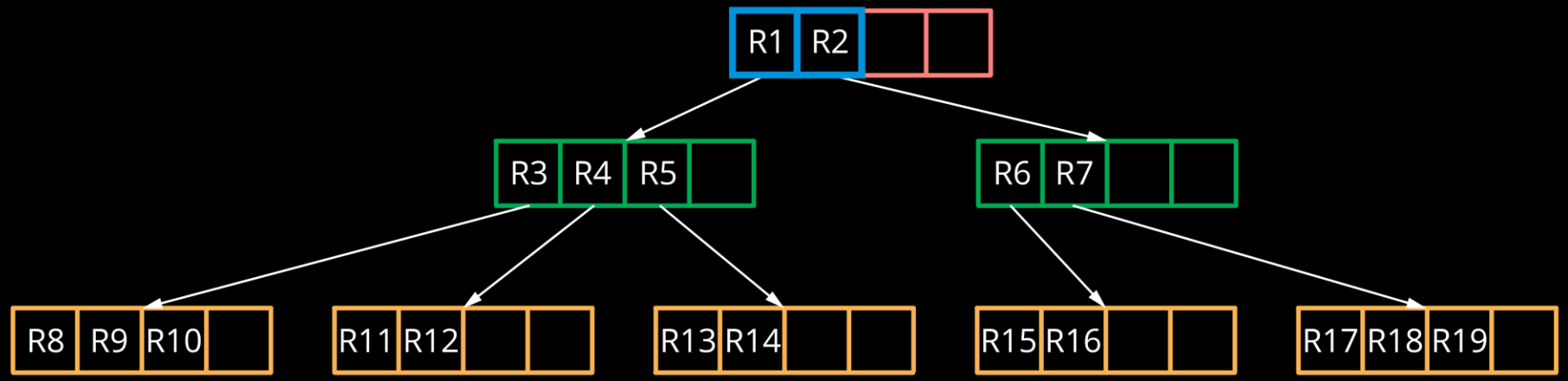
R-tree


R1
R2
R6
R15
R16
R7
R17
R18
R19
R4
R3
R8
R9
R10
R12
R11
R5
R13
R14

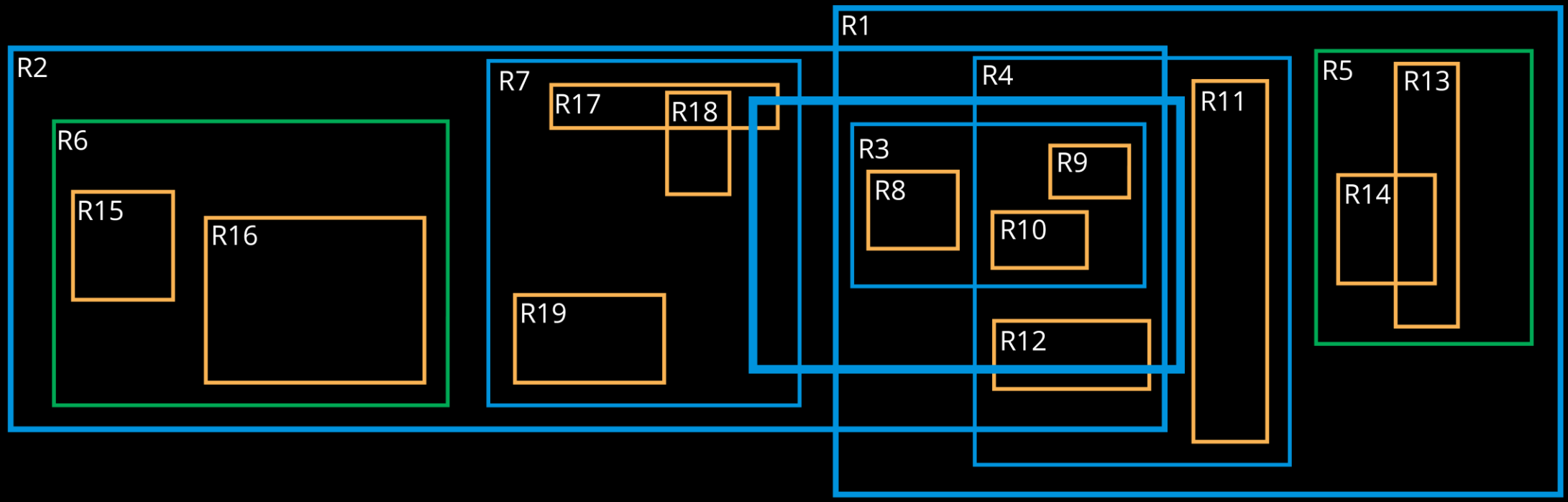

R1 R2
R3 R4 R5
R6 R7
R8 R9 R10
R11 R12
R13 R14
R15 R16
R17 R18 R19

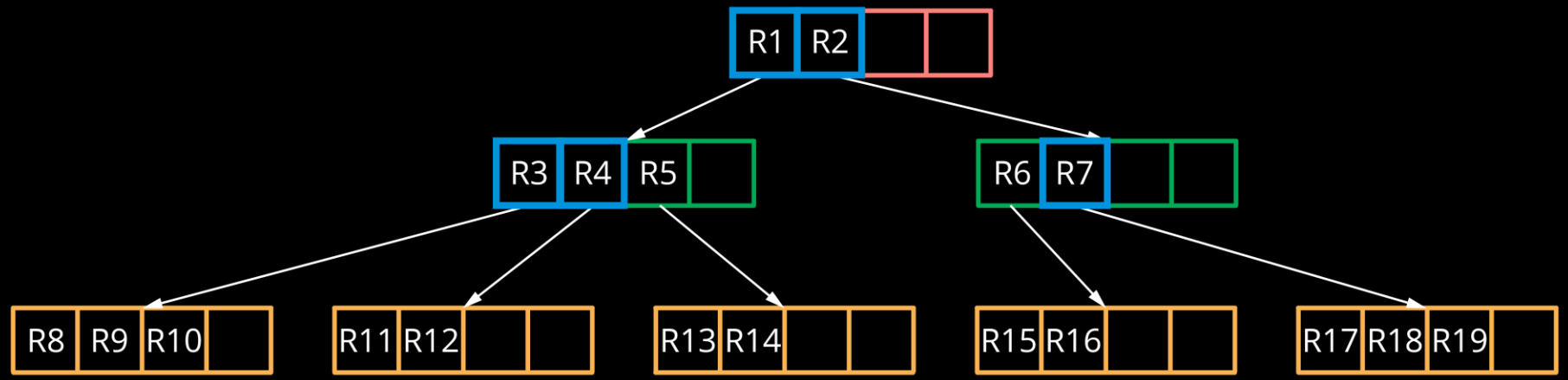
R-tree


R1
R2
R6
R15
R16
R7
R17
R18
R19
R4
R3
R8
R9
R10
R12
R11
R5
R13
R14

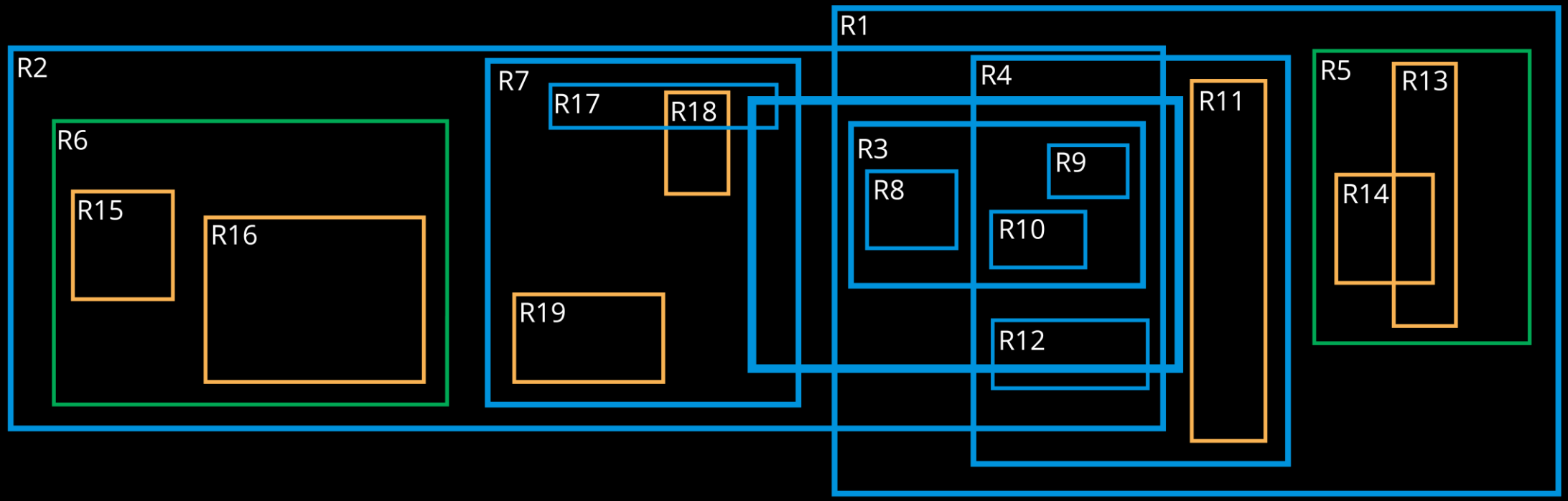

R1
R2
R3
R4
R5
R6
R7
R8
R9
R10
R11
R12
R13
R14
R15
R16
R17
R18
R19

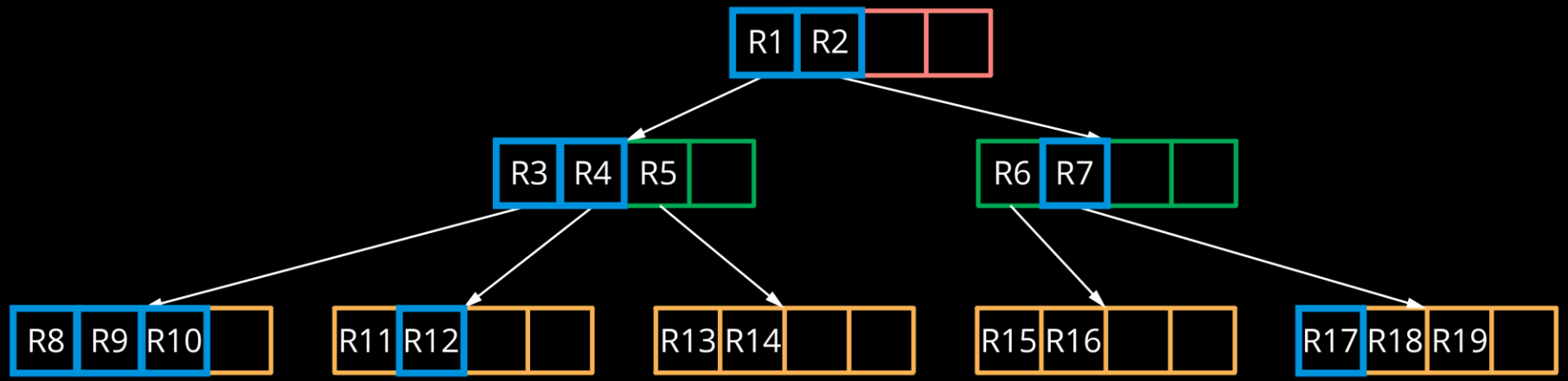
R-tree


R1
R2
R6
R15
R16
R7
R17
R18
R19
R4
R3
R8
R9
R10
R12
R11
R5
R13
R14

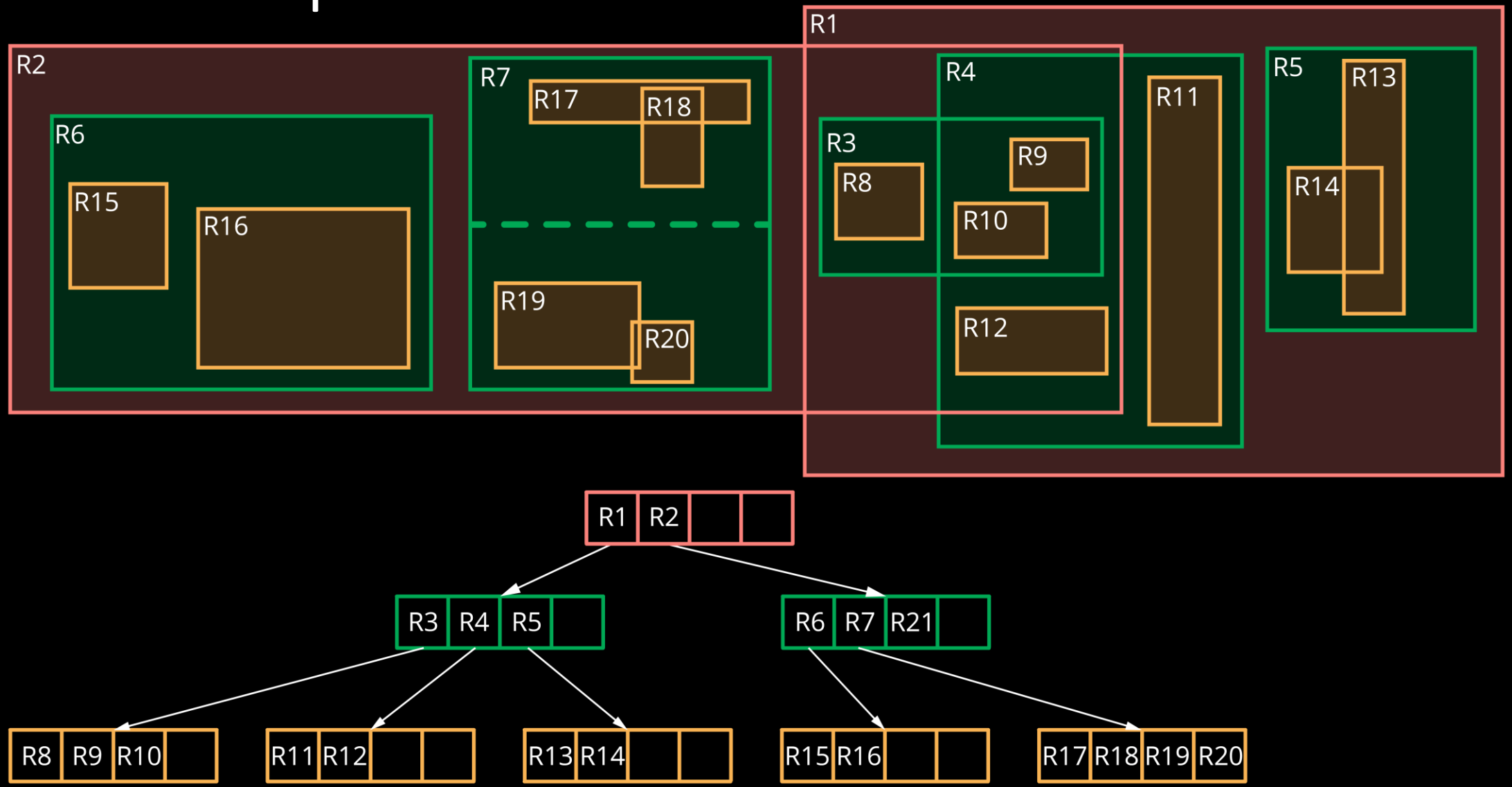

R1
R2
R3
R4
R5
R6
R7
R8
R9
R10
R11
R12
R13
R14
R15
R16
R17
R18
R19

R-tree

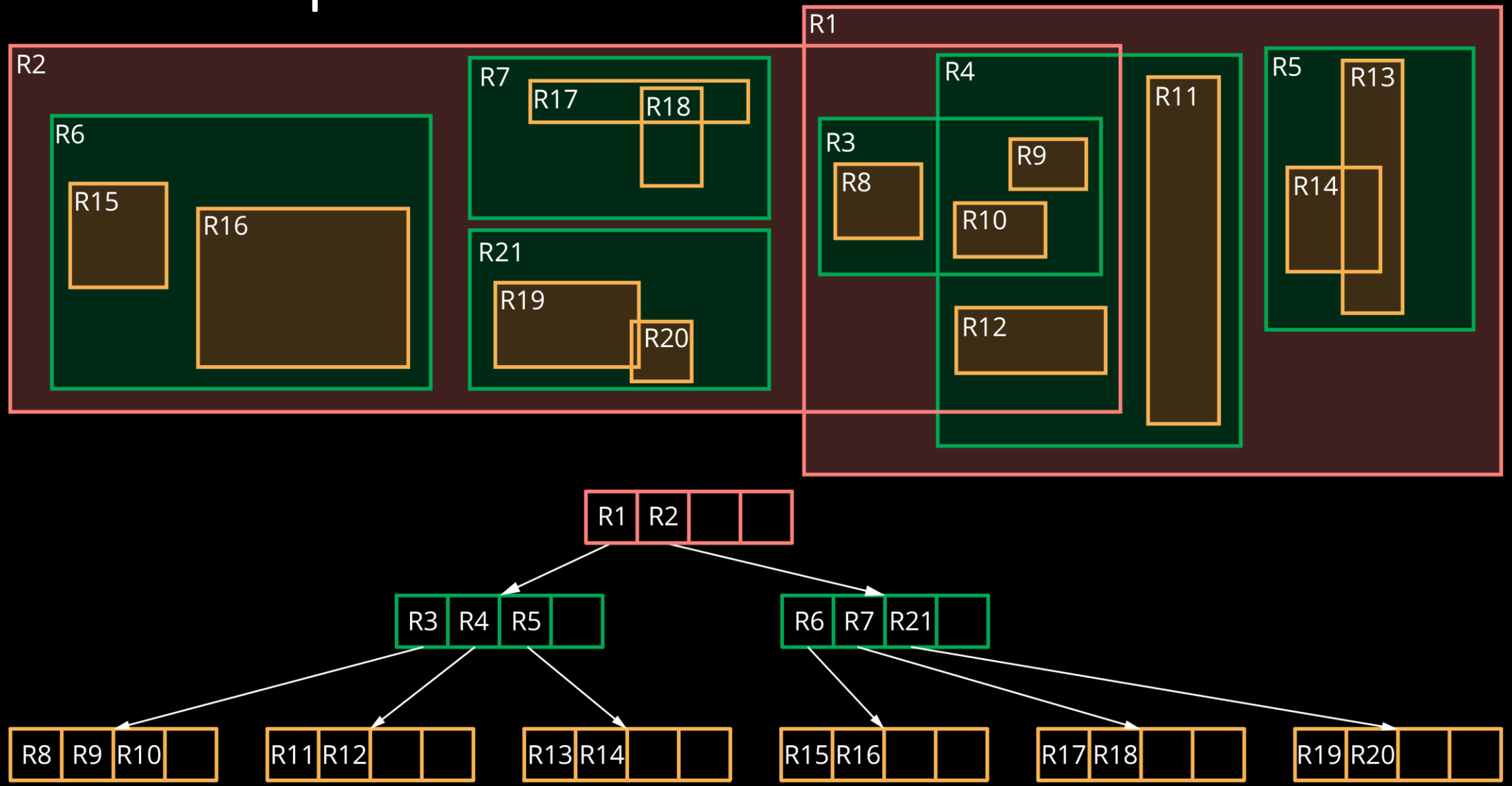

R1
R2
R6
R15
R16
R7
R17
R18
R19
R3
R8
R4
R9
R10
R12
R11
R5
R13
R14


R1
R2
R3
R4
R5
R6
R7
R8
R9
R10
R11
R12
R13
R14
R15
R16
R17
R18
R19

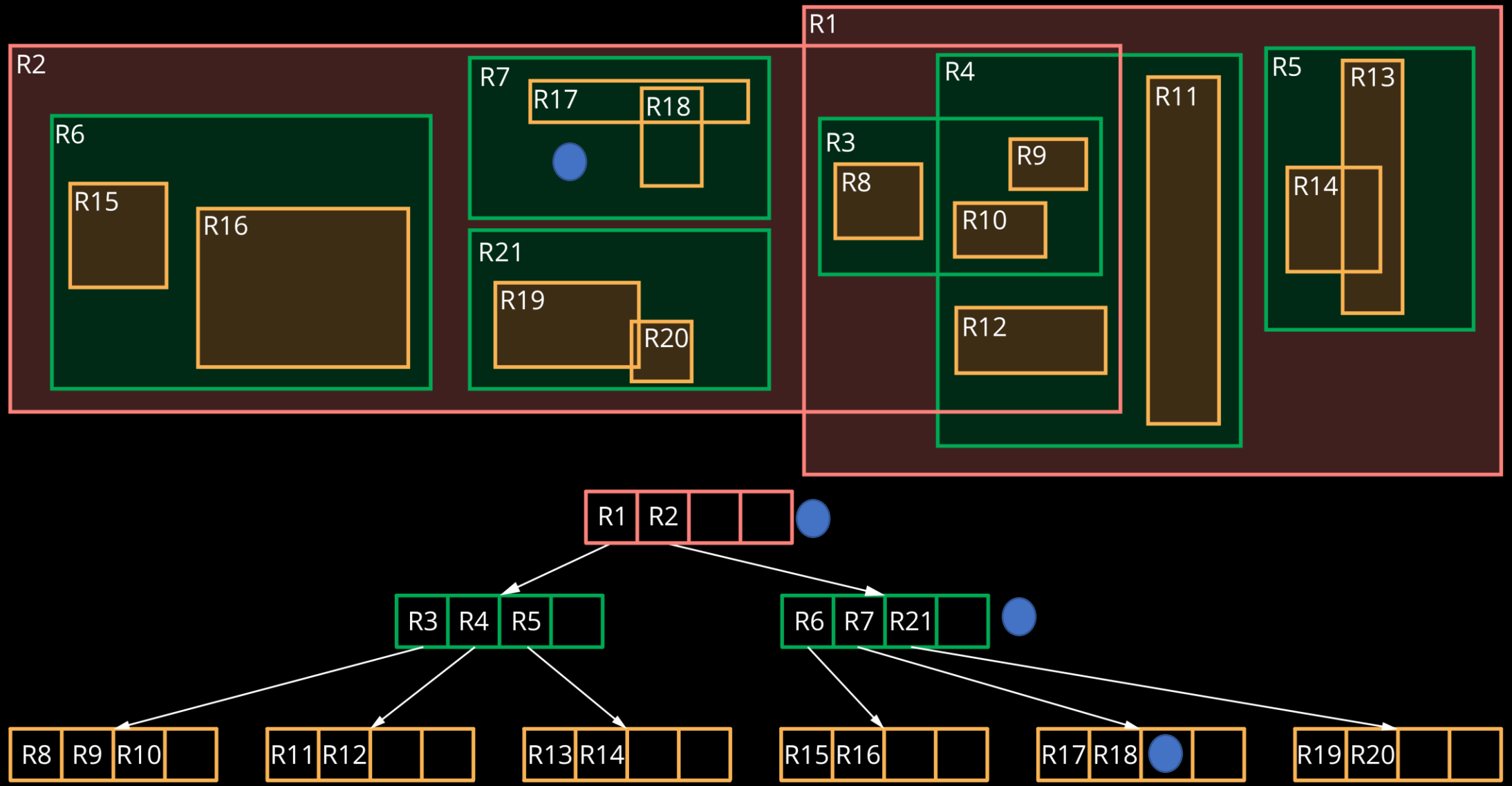
R-tree: split


R1
R2
R6
R15
R16
R7
R17
R18
R19
R20
R4
R3
R8
R9
R10
R12
R11
R5
R13
R14
R1 R2
R3 R4 R5
R6 R7 R21
R8 R9 R10
R11 R12
R13 R14
R15 R16
R17 R18 R19 R20

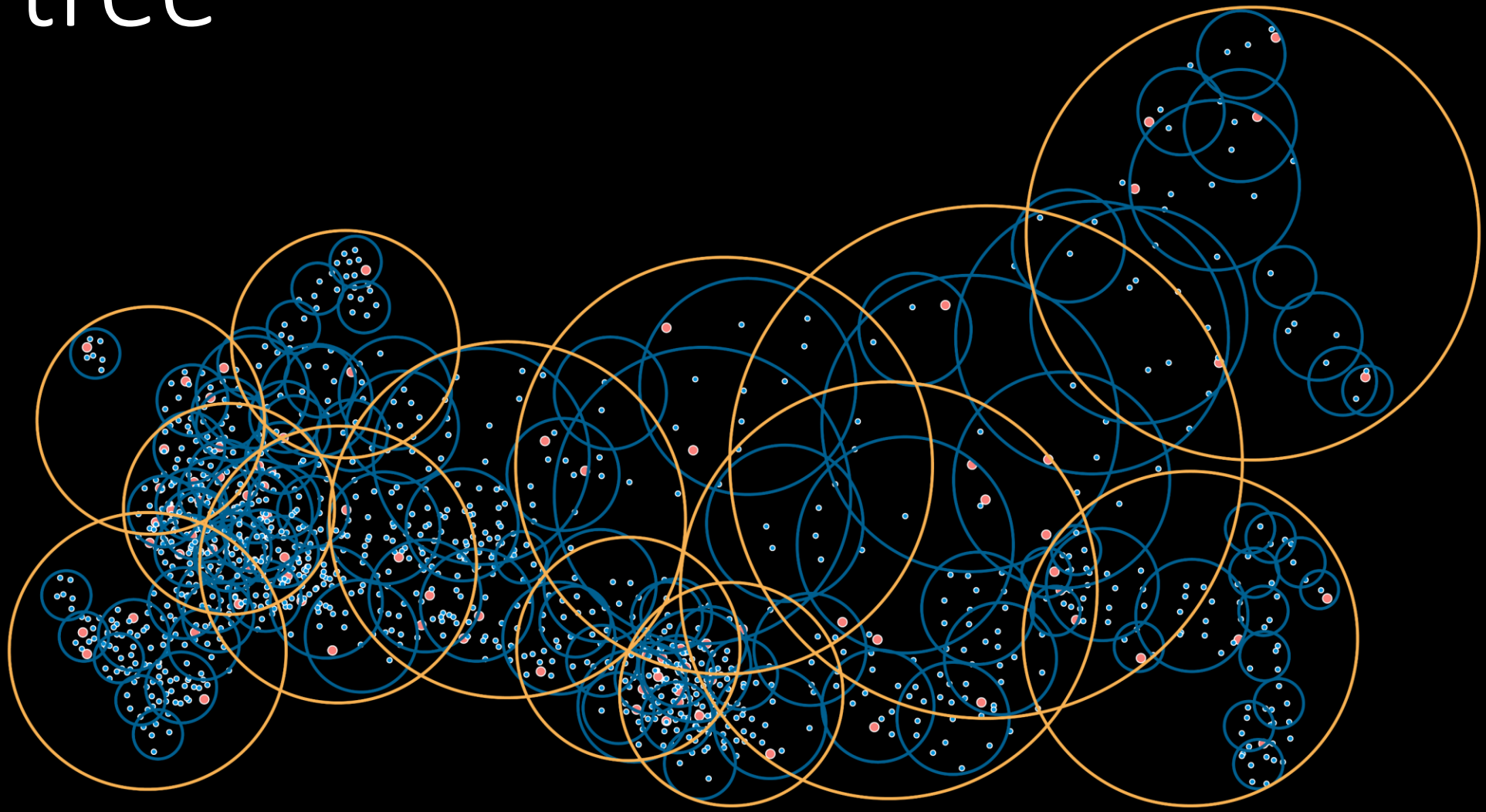
R-tree: split


R1
R2
R7
R17
R18
R6
R15
R16
R21
R19
R20
R4
R3
R8
R9
R10
R12
R11
R5
R13
R14
R1 R2
R3 R4 R5
R6 R7 R21
R8 R9 R10
R11 R12
R13 R14
R15 R16
R17 R18
R19 R20

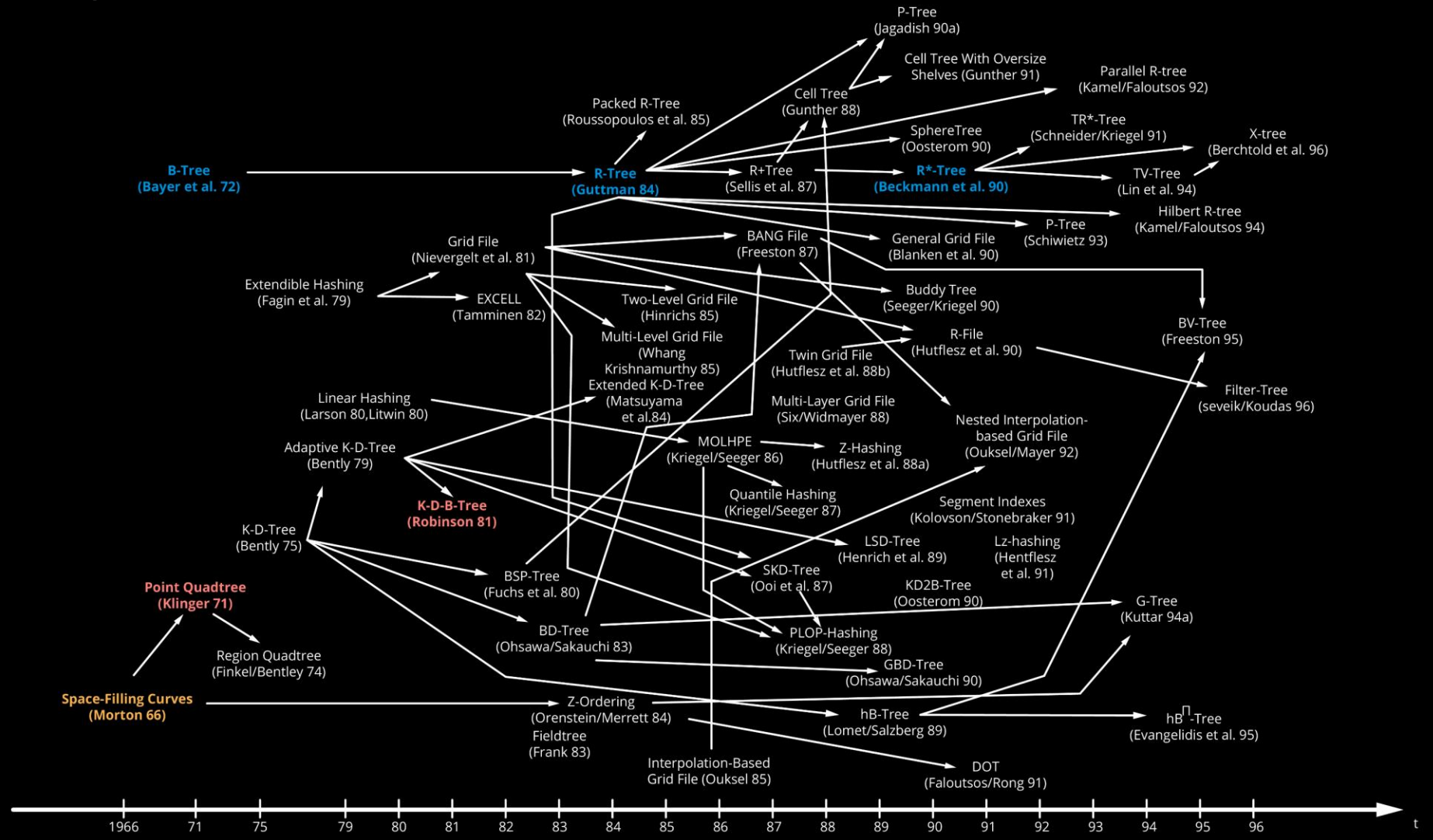
R-tree: choose subtree



M-tree

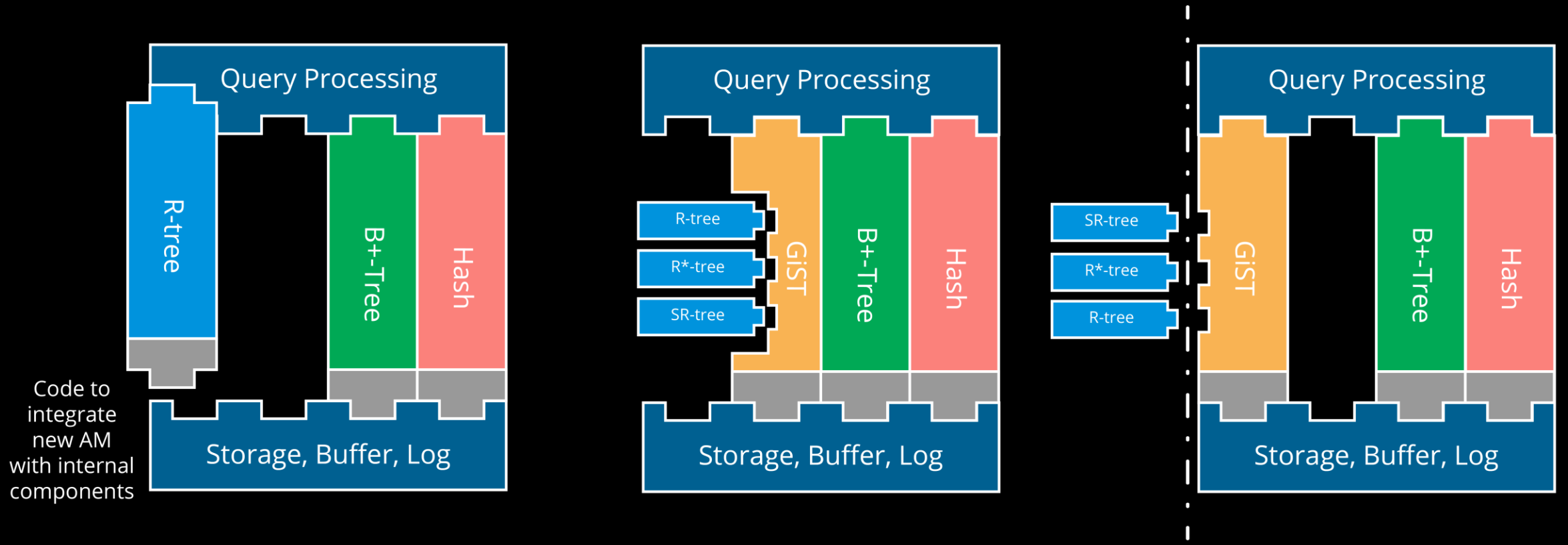


History of multidimensional access methods

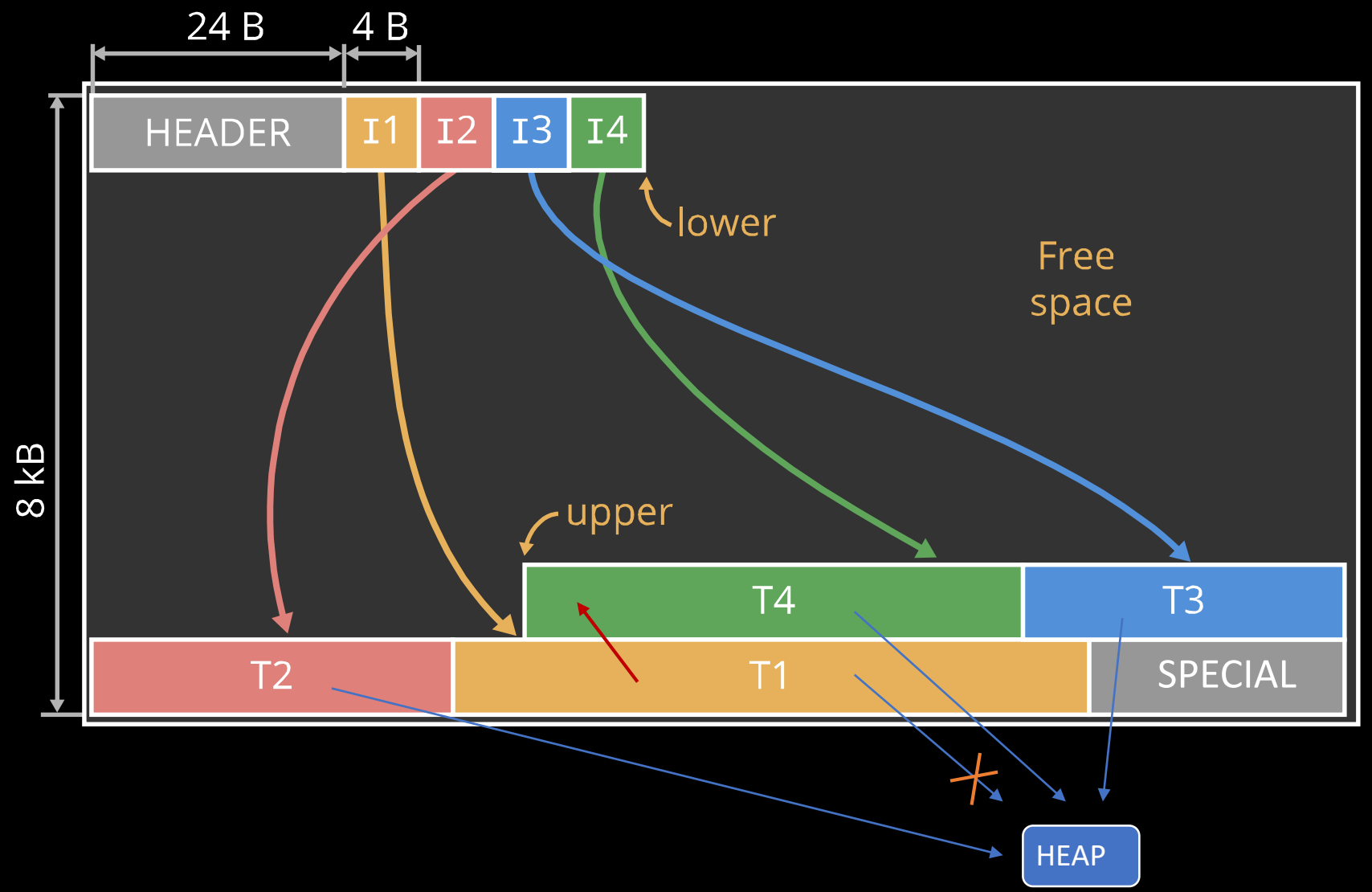


Based on work of O. Gunther and V. Gaede

Generalized index search trees



Intra-page indexing: skiptuple approach



Theoretically optimal fan-out

In a tree with fan-out *f* query returning exactly 1 row has to touch every tuple in a node, but in optimal tree path from root to leaf will take exactly 1 page on each level, so key-compares count *K* is equal to tree height *h* multiplied by *f*.

$$K = f \cdot h = [f \cdot \log_f N],$$

which is optimal with $f \rightarrow e \approx 3$.

Without loss of generality this holds for cache lines count instead of key compares.

As we saw above this does not work for *f = 2.* Probably, due to differences of constraints (experiment involved more than one row returned by aggregate query)

Performance improvement estimation
(rough upper bound)

- On a page with $n \approx 243$ entries optimal regrouping for $f = 3$ will construct intra-page tree with 4 levels.
- One-row-query traversal will take 15 key compares instead of 243
- Overall tree height will be 57% higher due to reduced 1.5 times page capacity
- One-row-query performance will be better 243/15/1.57 = 10 times

- But in practice I only could gain 10%, as for now.

Tuples were reordered with updates

- Before update

- After update of T3

IndexTupleOverwrite patch

- https://commitfest.postgresql.org/10/661/
- Test:

```
insert into dataTable(c)
    select cube(array[x/100,y/100,z/100])
      from generate_series(1,1e2,1) x,
           generate_series(1,1e2,1) y,
           generate_series(1,1e2,1) z;
```

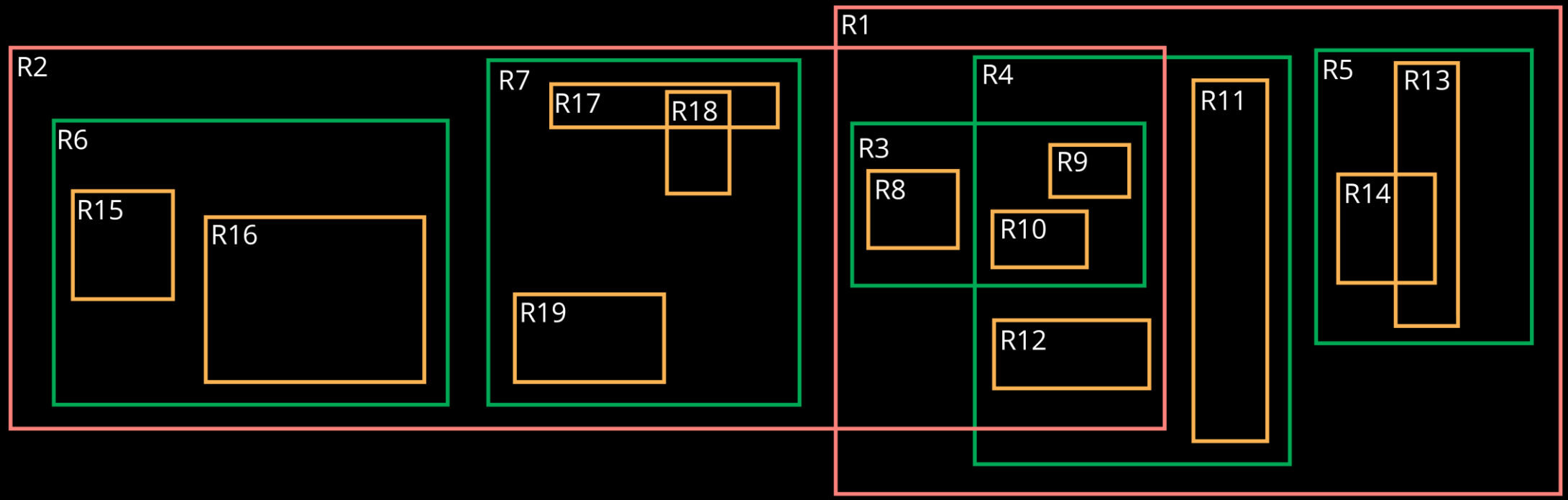
- Results: up to 2 times faster, 3 times smaller index

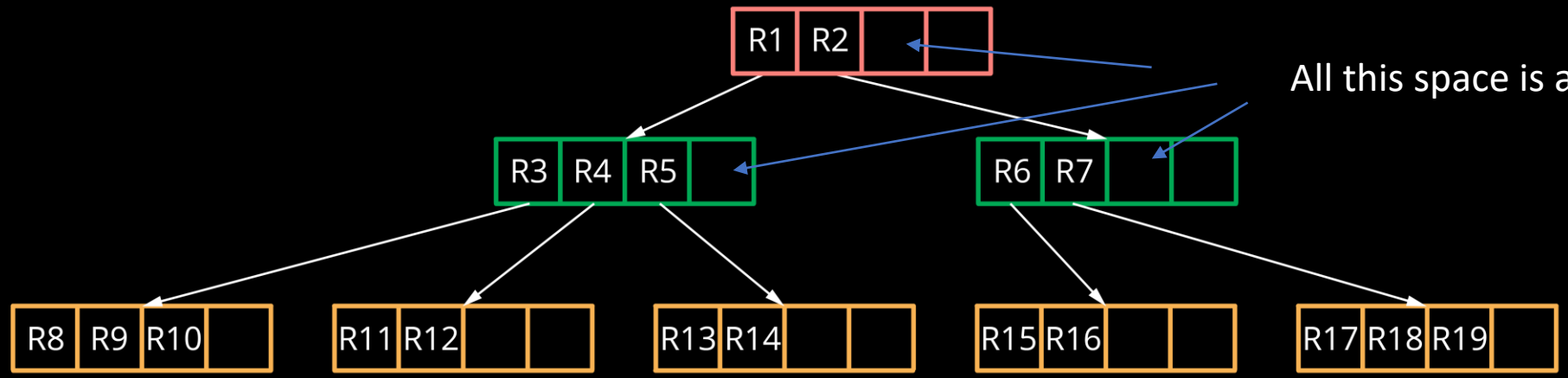
Only on ordered data!

Randomized data insertions speed improvement is around 10%-15%

Randomized data insertions already was on this speeds, it is ordered data that was slow

Fractal tree for GiST





All this space is a buffer!

Fractal tree for GiST

On 1 million of insertions classic GiST is 8% faster

On 3M performance is equal

On 30M lazy GiST if 12% faster

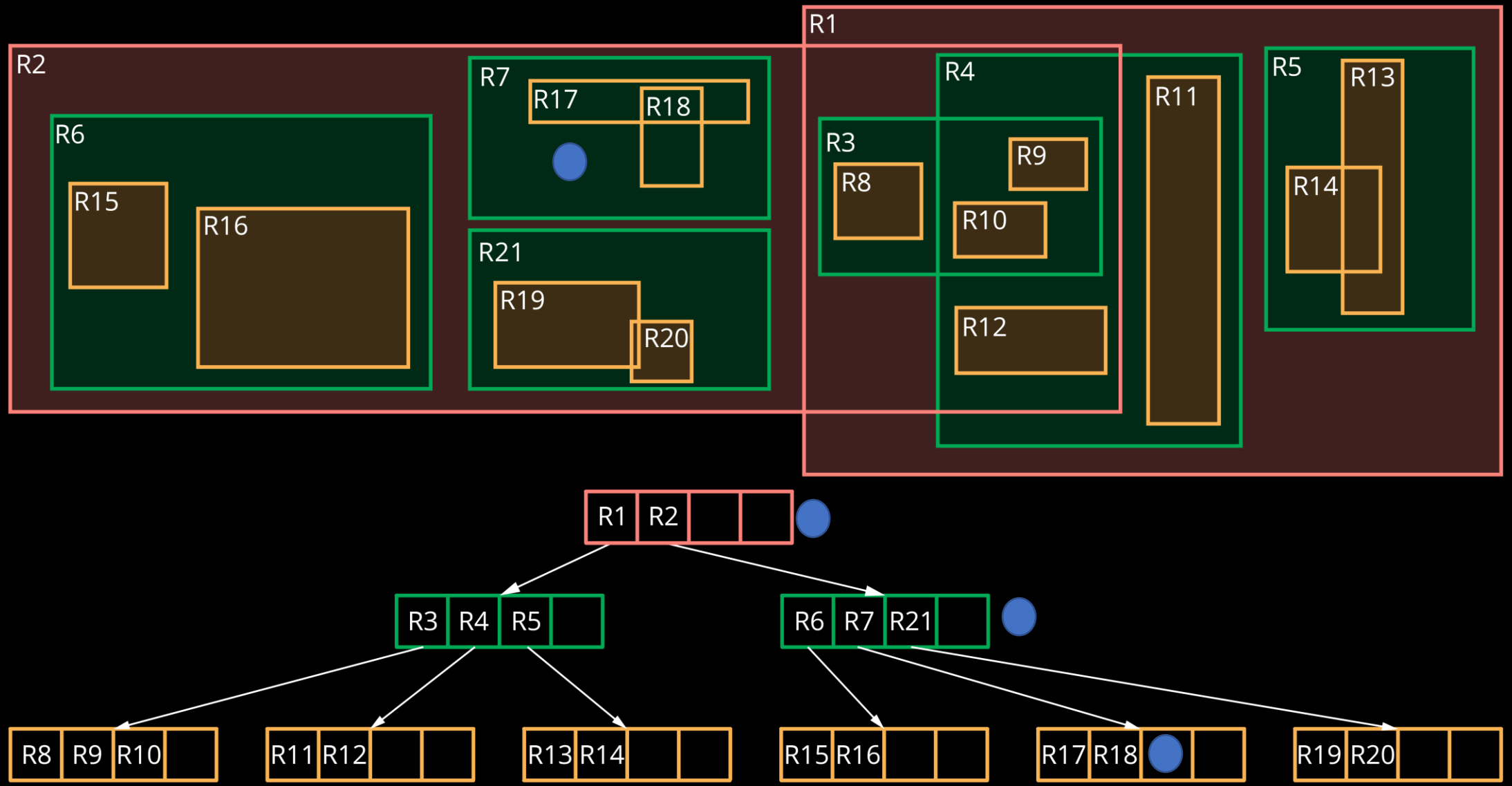
On my machine with SSD disks,

With HDDs IO is more important and number may change in favor of fractal tree

More details and patch:

https://www.postgresql.org/message-id/CAJEAwVE_8VDTV1Utfe1CmbVRCvVPtvHZnSKFdMf0rB5mELSLeA%40mail.gmail.com

R-tree: choose subtree



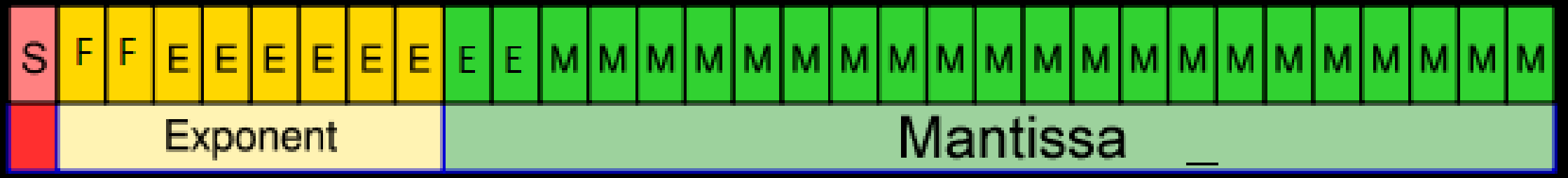
Current penalty function from *cube* extension

```
/*
** The GiST Penalty method for boxes
** As in the R-tree paper, we use change in area as our penalty metric
*/
Datum
g_cube_penalty(PG_FUNCTION_ARGS)
{
        GISTENTRY  *origentry = (GISTENTRY *) PG_GETARG_POINTER(0);
        GISTENTRY  *newentry = (GISTENTRY *) PG_GETARG_POINTER(1);
        float      *result = (float *) PG_GETARG_POINTER(2);
        NDBOX      *ud;
        double          tmp1,
                        tmp2;

        ud = cube_union_v0(DatumGetNDBOX(origentry->key),
                                        DatumGetNDBOX(newentry->key));
        rt_cube_size(ud, &tmp1);
        rt_cube_size(DatumGetNDBOX(origentry->key), &tmp2);
        *result = (float) (tmp1 - tmp2);

        PG_RETURN_FLOAT8(*result);
}
```

Realms

```
/* REALM 0: No extension is required, volume is zero, return edge       */
/* REALM 1: No extension is required, return nonzero volume             */
/* REALM 2: Volume extension is zero, return nonzero edge extension     */
/* REALM 3: Volume extension is nonzero, return it                      */
```

2 flag bits inside IEEE 754 float



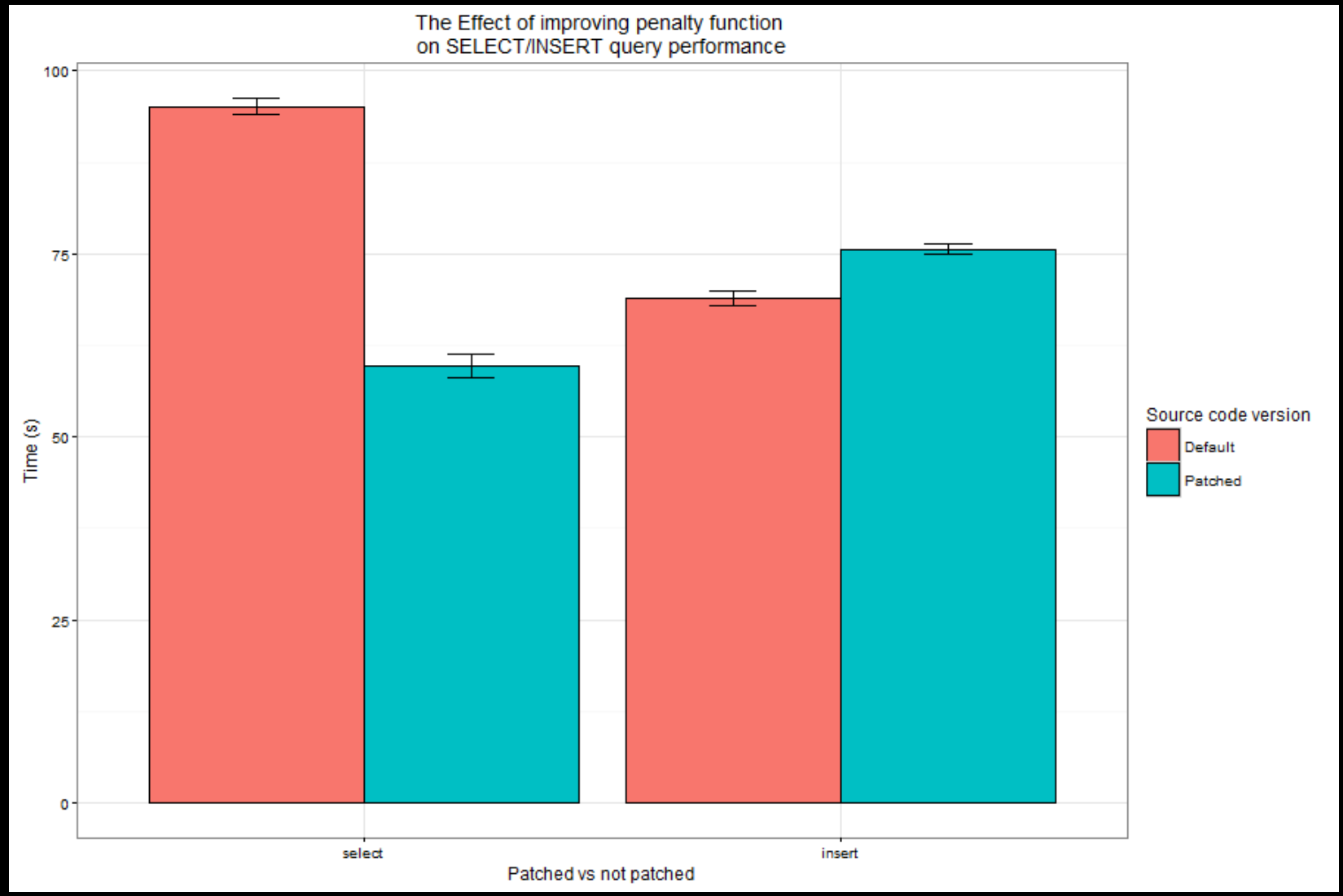
M

M

Packing float: code

```
typedef union { float fp; int i; } U;
float pack_float(const float v, const int r) {
  const U a = { .fp = v };
  const U b = { .i = (a.i >> 2) + r * (INT32_MAX / 4) };
  return b.fp;
}
```

Benchmarking



Cube extension GiST penalty function improvement

- https://commitfest.postgresql.org/10/782/

“Returned with feedback”

Revised R*-tree

- Split algorithm (with small deviations):

https://github.com/x4m/pggistopt/blob/rrsplit_pack/contrib/cube/cube.c#L539

Up to 50% faster small SELECTs for cube extension

- Choose subtree algorithm:

Currently impossible in GiST API

As Norbert Beckman, author of RR*-tree, put it in the discussion: “Overlap optimization is one of the main elements, if not the main query performance tuning element of the RR*-tree. You would fall back to old R-Tree times if that would be left off.”

GiST API: collision check function

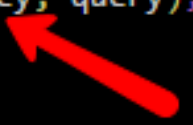
- Collision check currently returns a binary result:

1. Query may be collides with subtree MBR
2. Query do not collides with subtree

- This result may be augmented with a third state: subtree is totally within the query. In this case, GiST scan can scan down subtree without key checks.

GiST API: collision check function

```
bool
g_cube_internal_consistent(NDBOX *key,
                                                NDBOX *query,
                                                StrategyNumber strategy)

{
        bool            retval;

        /*
         * fprintf(stderr, "internal_consistent, %d\n", strategy);
         */
        switch (strategy)
        {
                case RTOverlapStrategyNumber:
                        retval = (bool) cube_overlap_v0(key, query);
                        break;
                case RTSameStrategyNumber:
                case RTContainsStrategyNumber:
                case RTOldContainsStrategyNumber:
                        retval = (bool) cube_contains_v0(key, query);
                        break;
                case RTContainedByStrategyNumber:
                case RTOldContainedByStrategyNumber:
                        retval = (bool) cube_overlap_v0(key, query);
                        break;
                default:
                        retval = FALSE;
        }
        return (retval);
}
```



GiST API advancement

1. Allow choose subtree not via penalty calculation
2. Extend consistency API

GiST API advancement

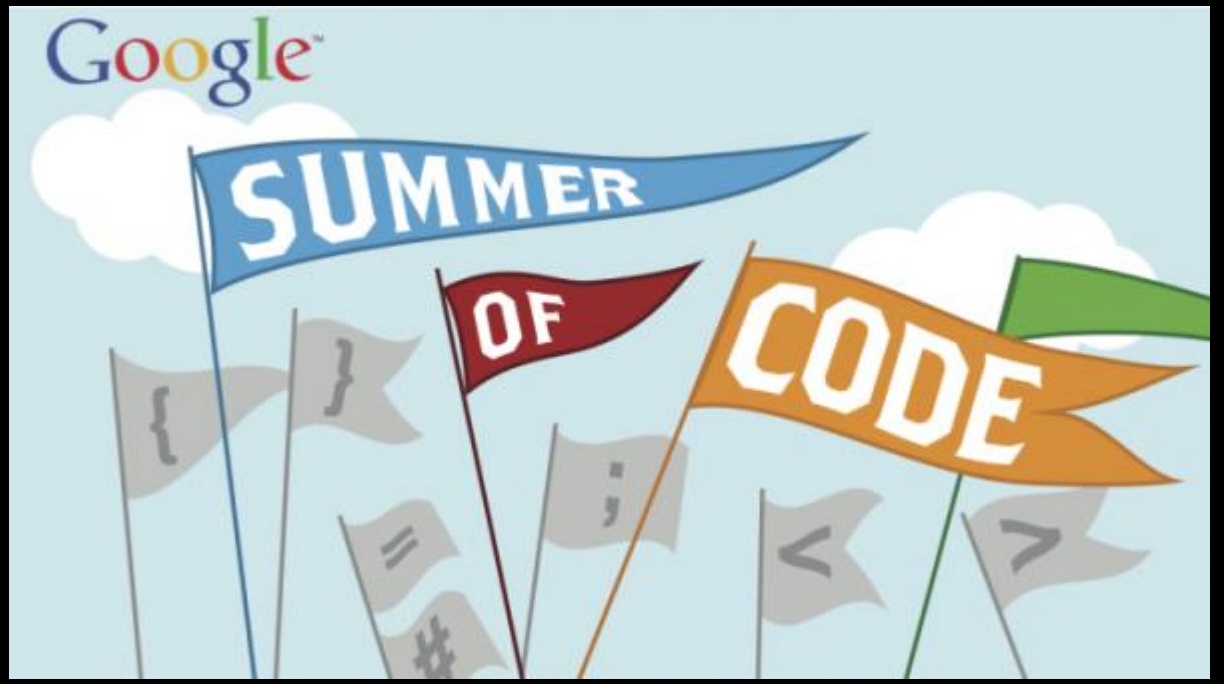
1. Allow choose subtree not via penalty calculation
2. Extend consistency API

Any students here?

GiST API advancement

1. Allow choose subtree not via penalty calculation
2. Extend consistency API

GSoC 2017 applications are open!



Read more on
https://wiki.postgresql.org/wiki/GSoC_2017#GiST_API_advancement

Questions?

Contacts

amborodin@acm.org

vk.com/amborodin

github.com/x4m

Andrey Borodin

Software engineer at Octonica

PhD, Associate Professor at Ural Federal University