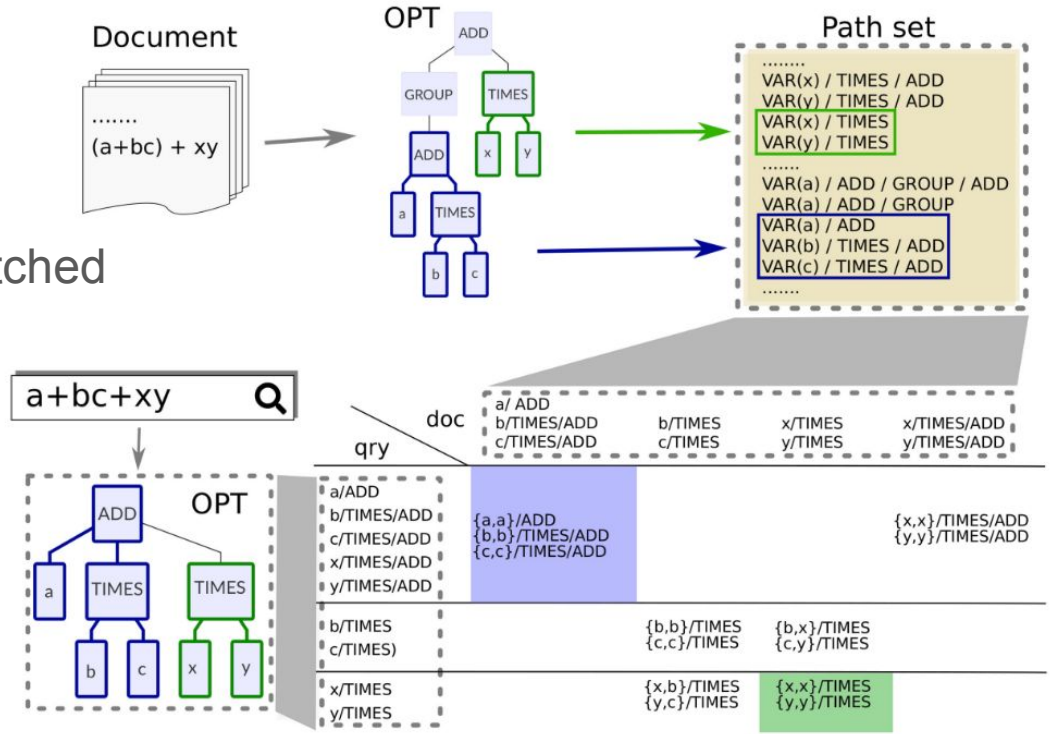
Section tree pruning for finding best matched subexpression

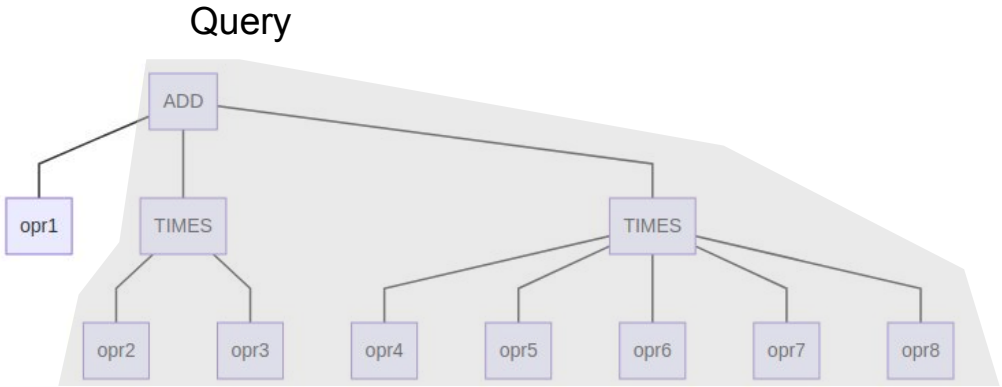
Wei Zhong

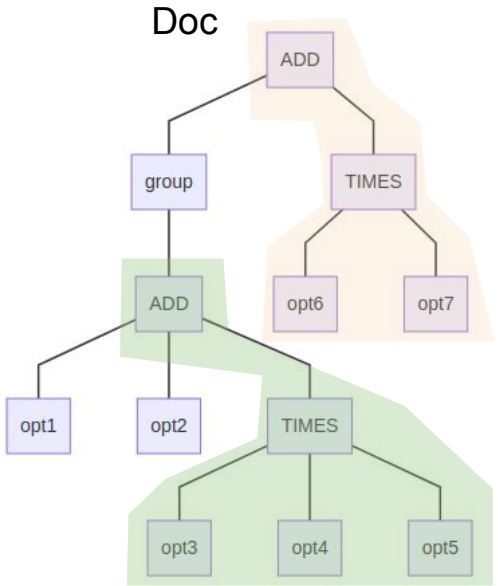
Recap

- We want to find the best-matched subexpressions.
- The experiment shows the model is computation-bounded.
- In the alignment algo, we need to examine all rows of table and all K best-matched subexpressions.
- Pruning idea: Each tree rooted at some node has upper-bound of matched nodes and we can skip rows with upperbound ≤ a threshold.



Sector tree





Sector tree: Subtrees with same leaf-root paths.

The maximum matching between two sector trees T1 and T2 is just min(leaves(T1), leaves(T2)).
Old algorithm tries to match tree paths one by one, if we index sector tree (no leaf IDs under it), it will speed up the matching algorithm.

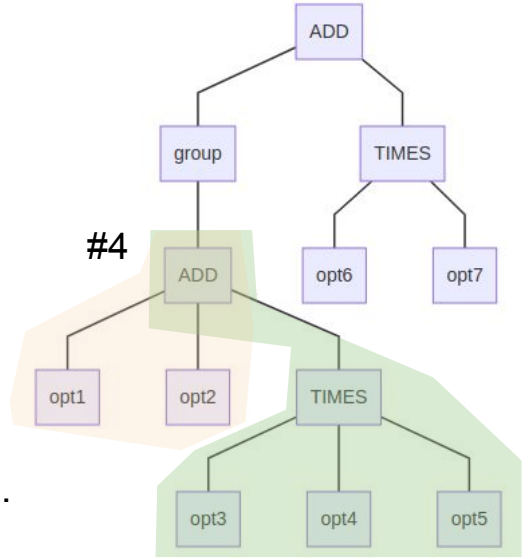
Sector tree

One rooted subtree can be decomposed into several sector trees.

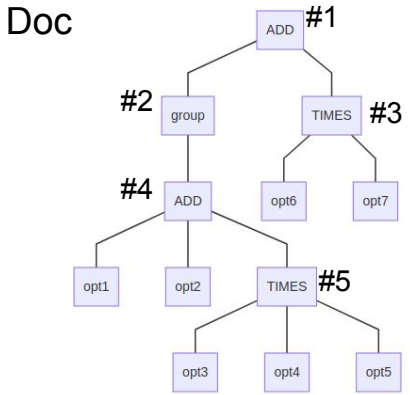
Each sector tree is stored in only one posting list, and it is impossible for one posting list sector trees to match any paths of sector trees from other posting lists.

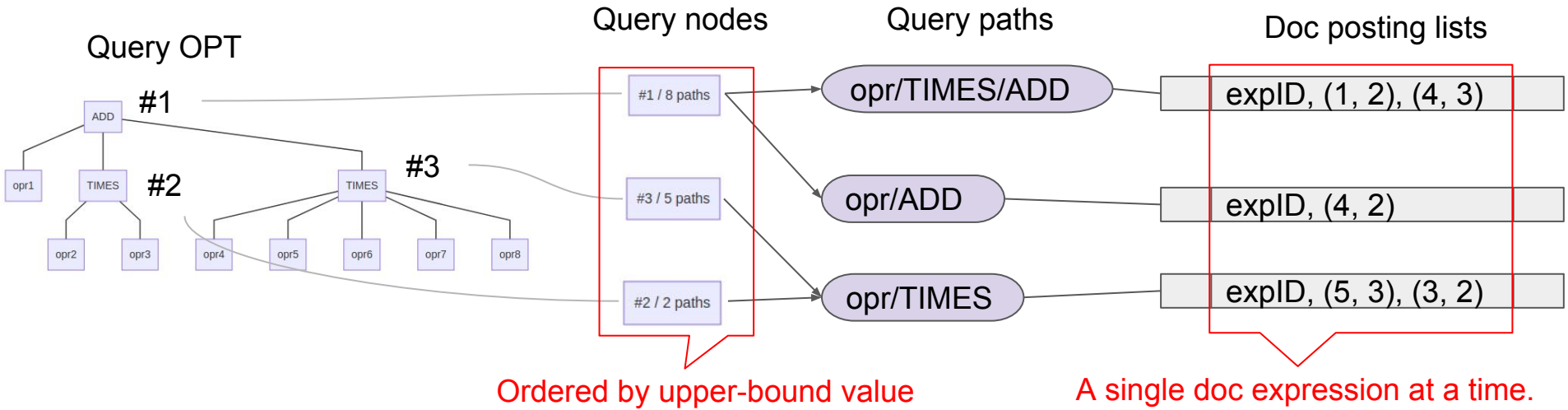
For matching, a pair of value is enough to store a sector subtree. One is the node ID of the subtree root-end node, the other is the “width” of the subtree. Use the notation **(rootID, width)** for sector tree.

For example, the subtree rooted at node#12 has two sector trees: **(4, 2)** and **(4, 3)**.



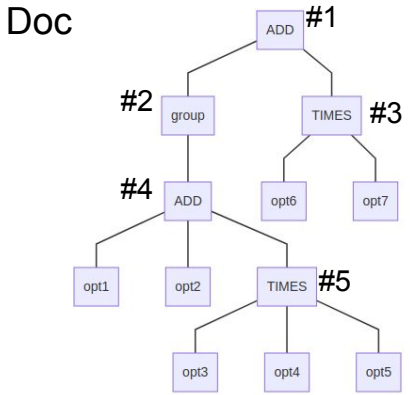
Sector tree posting lists

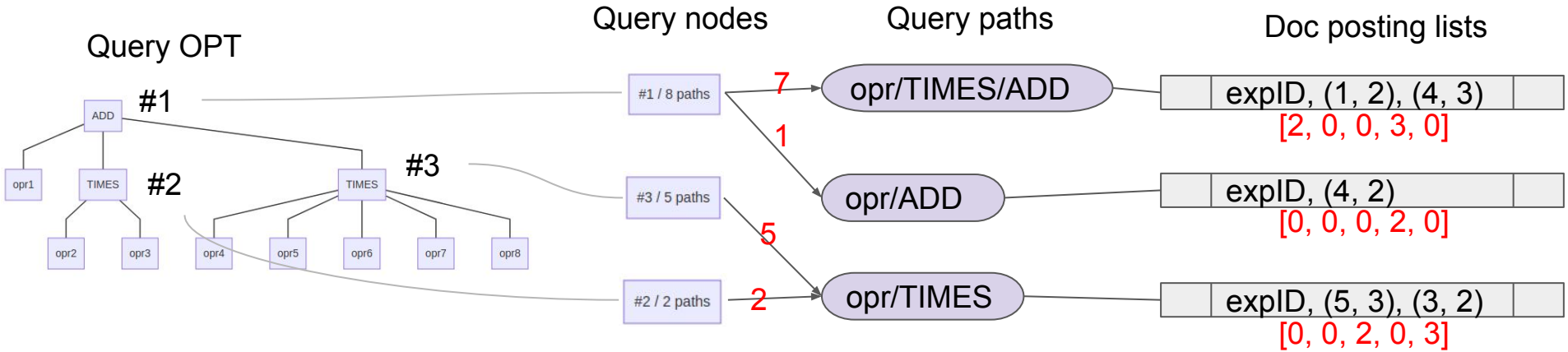




Construct sector posting lists, and utilize query node score upper-bounds.

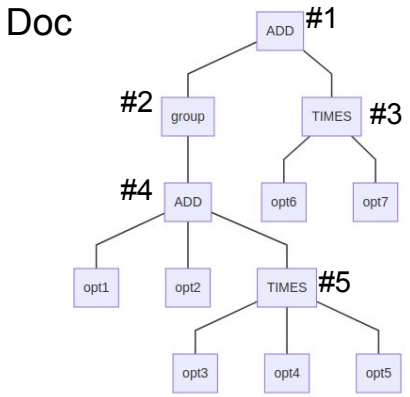
Sector-tree width values

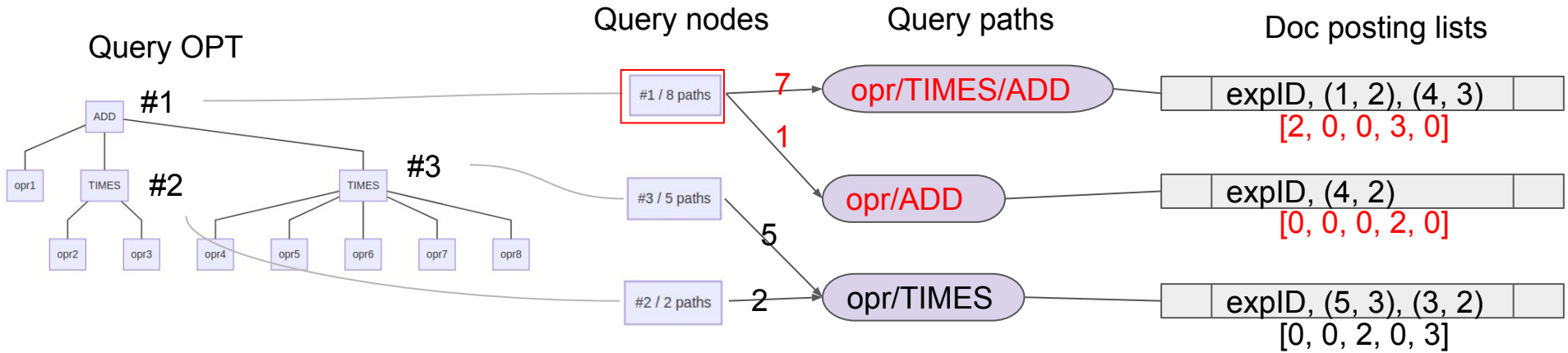




Assign *sector-tree width* values to query node -> paths relation, and use vector representation in doc space.

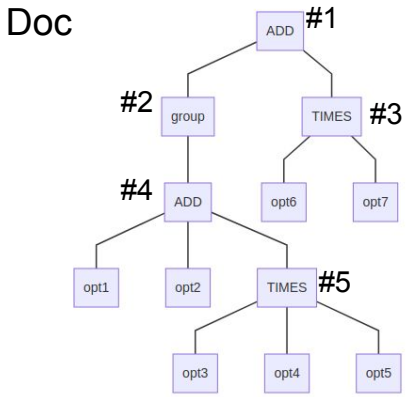
Sector-tree matching iterations

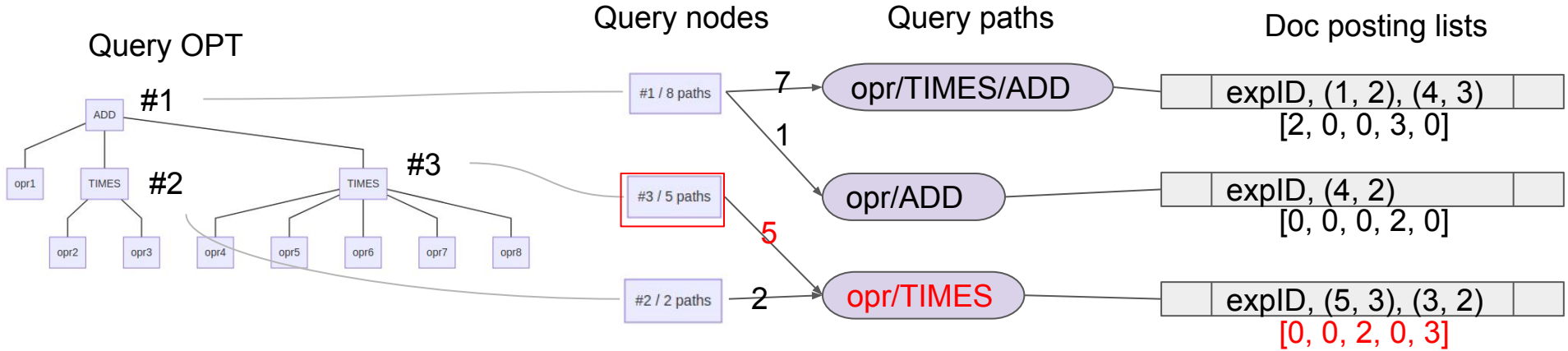




Iteration (1): min(7, [2, 0, 0, 3, 0]), unmatched query paths: 7 - max(2,3) = 4 (*dynamic upperbound* -> 8 - 4 = 4)
+ min(1, [0, 0, 0, 2, 0]), unmatched query paths: 1 - max(1) = 0
= [2, 0, 0, 4, 0], where the widest = 4.

Sector-tree matching iterations

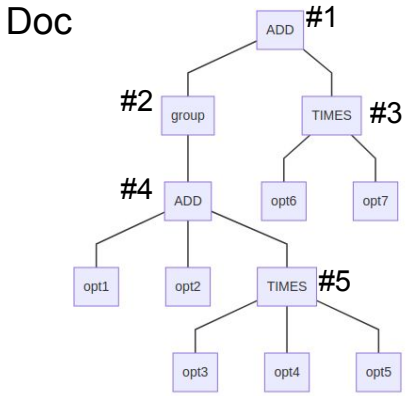


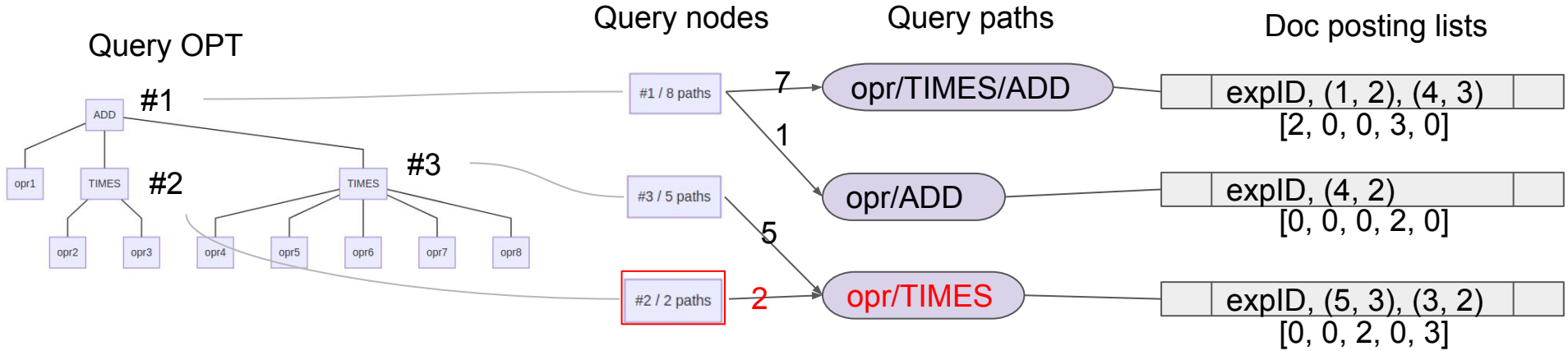


Iteration (2): min(5, [0, 0, 2, 0, 3]), unmatched query paths: 5 - max(2,3) = 2

= [0, 0, 2, 0, 3], update the current widest to max(previous widest, 3) = 4 (not changed)

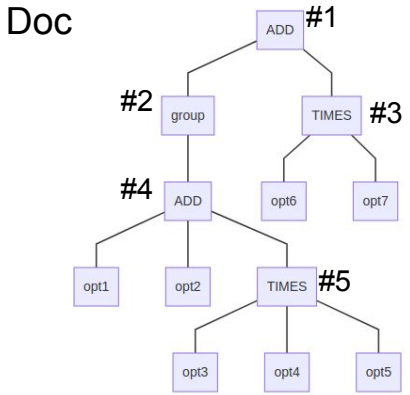
Sector-tree pruning (1)

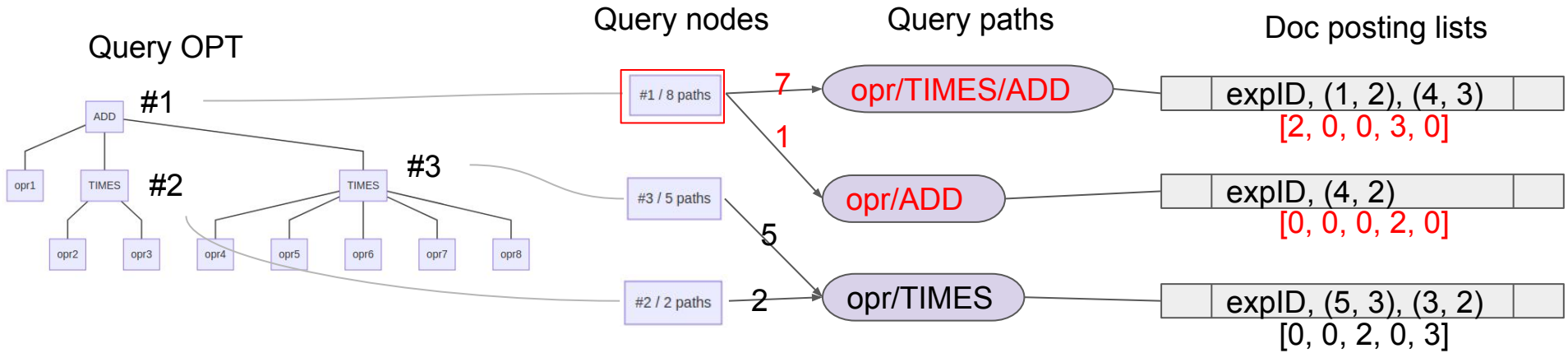




Iteration (3): Since this node has upper-bound value of 2 ≤ current widest value 4, we can skip this iteration.

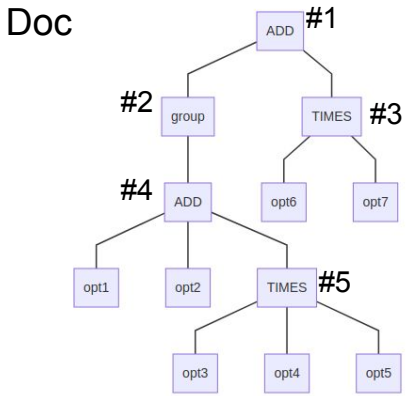
Sector-tree pruning (2)

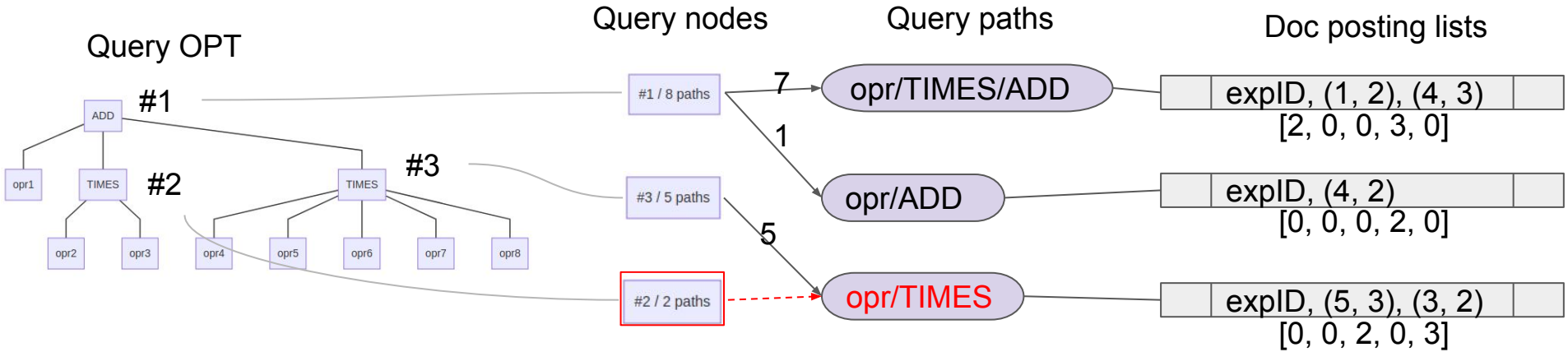




Go back to iteration (1):
From min(7, [2, 0, 0, 3, 0]), we know unmatched query paths: 7 - max(2,3) = 4 (*dynamic upperbound* -> 8 - 4 = 4)
If the *dynamic upper bound* value ≤ current widest value, we can also skip that iteration *as early as possible*.

Sector-tree pruning (3)





Also, assume we have a min-heap that stores currently top-K best scored hits, at the top of the min-heap is the lowest scored top-K hit, denote the matched leaves of it as threshold value “Θ”.
Then if *upper bound* value ≤ Θ, we can decrease the reference counter of associated posting list, and if this posting list reference counter goes ≤ 0, that posting list can be dropped completely for later merging.

Pseudo code

```
struct sector_tr {
        int rnode;
        int width;
};

struct posting {
        struct sector_tr *(*get_sector_trees)();
        int   (*get_min)();
        void  (*advance)();
        int expID;
        int refcnt;
};

struct pruner_node {
        int width; // each node is ordered by this key
        struct sector_tr secttr[64]; // each sect rnode is the same here
        struct posting *posting[64];
};


int main() // simulate one merge stage iteration
{
        struct posting all_post[64];
        struct pruner_node q_nodes[64];
        int theta = 123; /* best matched number of leaves */
        int q_widest_match = 0;
        for (int i = 0; i < 64 /* |T_q| */; i++) {
                struct pruner_node *q_node = q_nodes + i;
                int q_upperbound = q_node->width;
                int q_node_match = 0;
                int sumvec[128] = {0};

                if (q_upperbound <= theta) {
                        for (int j = 0; j < 64 /* |l(T_q)| */; j++) {
                                struct posting *post = q_node->posting[j]
                                post->refcnt --;
                        }
                        /* delete this pruner_node */;
                        continue;
                }

                for (int j = 0; j < 64 /* |l(T_q)| */; j++) {
                        struct sector_tr *q_secttr = q_node->secttr + j;
                        struct posting *post = q_node->posting[j];
                        struct sector_tr *d_secttr = post->get_sector_trees();
                        int qw = q_secttr->width;
                        int max = 0;

                        /* skip non-hit postings */
                        if (post->get_min() != post->expID)
                                continue;

                        /* vector addition */
                        for (int k = 0; k < 64 /* |l(T_d)| */; k++) {
                                int dw = d_secttr[k].width;
                                int rn = d_secttr[k].rnode;
                                if (qw > dw) {
                                        sumvec[rn] += dw;
                                        if (dw > max) max = dw;
                                } else {
                                        sumvec[rn] += qw;
                                        if (qw > max) max = qw;
                                }

                                if (sumvec[rn] > q_node_match) {
                                        q_node_match = sumvec[rn];
                                }
                        }

                        /* update upperbound for each vector addition */
                        if (max < qw) {
                                q_upperbound -= (qw - max);
                                if (q_upperbound <= theta)
                                        break;
                        }
                }

                if (q_node_match > q_widest_match)
                        q_widest_match = q_node_match;
        }
```