07-F

BST Application

Segment Tree

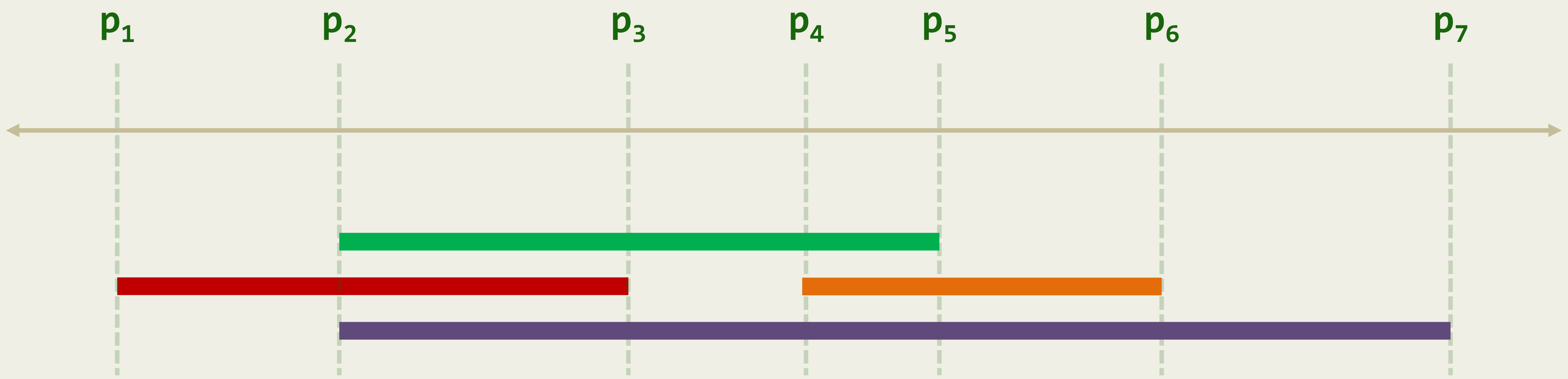
把一条线分割成不相等的两段，再把这两段按照同样的比例再分成两个部分。假设第一次分出来的两段中，一段代表可见世界，另一段代表理智世界。然后再看第二次分成的两段，他们分别代表清楚与不清楚的程度，你便会发现，可见世界那一段的第一部分是它的影像。

邓俊辉
deng@tsinghua.edu.cn

Elementary Intervals

❖ Let $I = \{ [x_i, x_i'] \mid i = 1, 2, 3, \ldots, n \}$ be n intervals on the x-axis

❖ Sort all the endpoints into $\{ p_1,\ p_2,\ p_3,\ \ldots,\ p_m \}$, $m \leq 2n$



❖ m+1 elementary intervals are hence defined as:

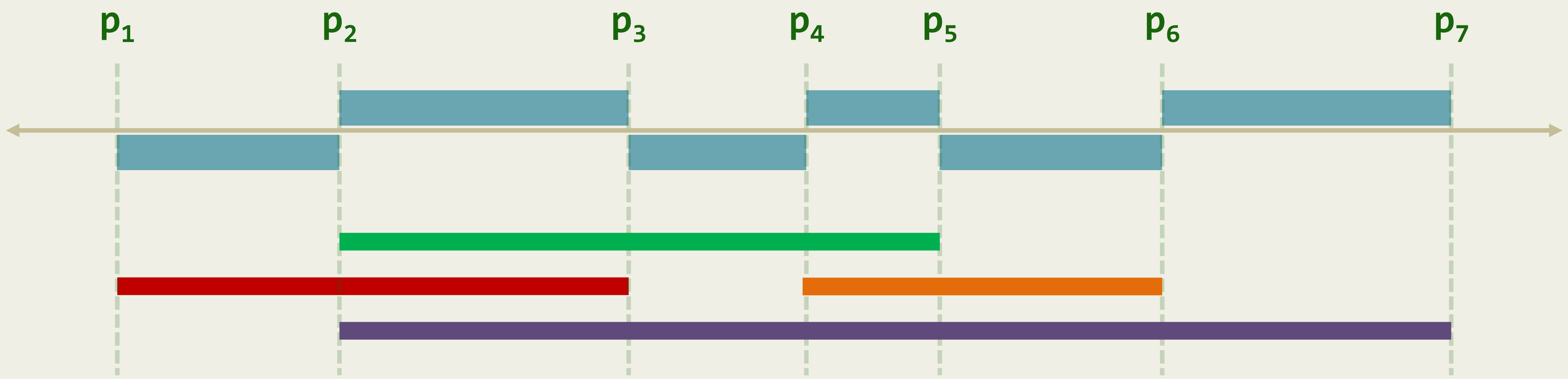
$$(-\infty, p_1],\ (p_1, p_2],\ (p_2, p_3],\ \ldots,\ (p_{m-1}, p_m],\ (p_m, +\infty]$$

Discretization

👁 Within each EI, all stabbing queries share a same output

∴ If we sort all EI's into a vector and

store the corresponding output with each EI, then ...

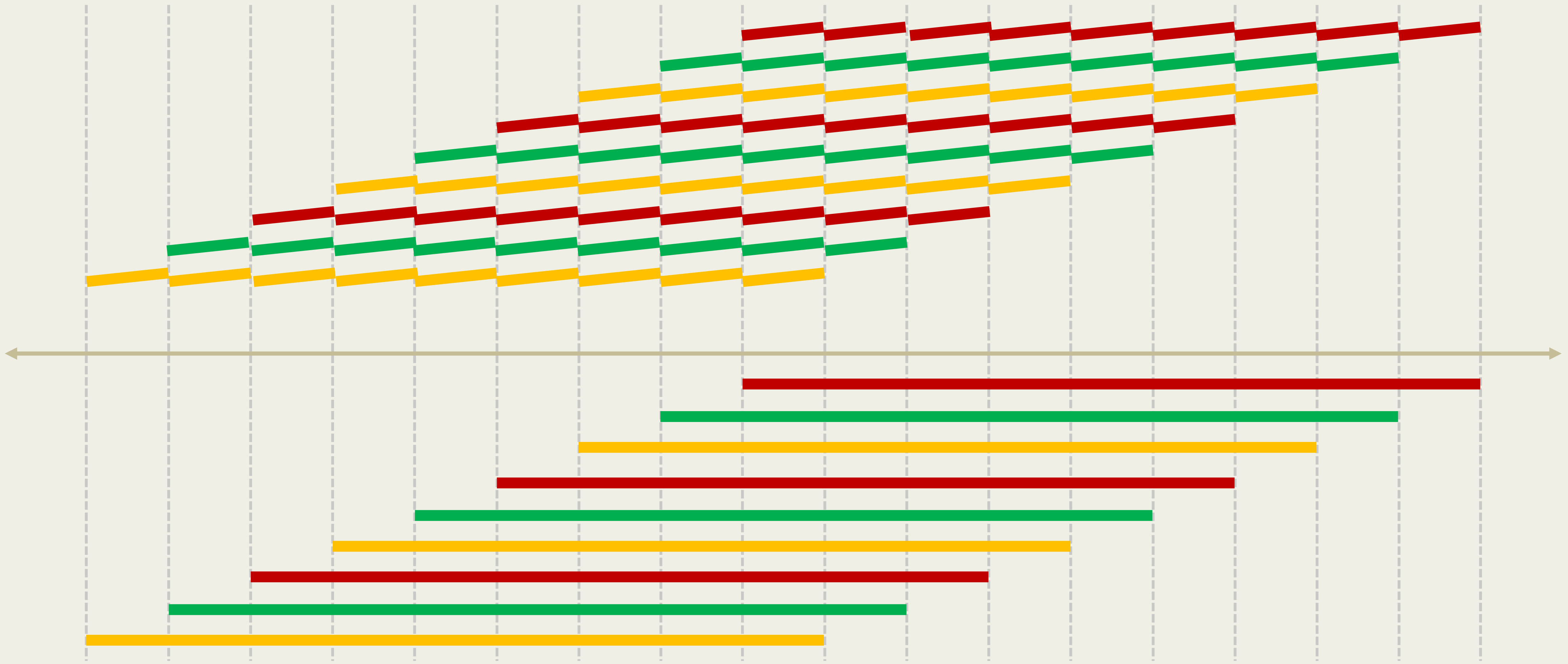


∴ Once a query position is determined, //by an $\mathcal{O}(\log n)$ time binary search

the output can then be returned directly //$\mathcal{O}(r)$

Worst Case

❖ Every interval spans $\Omega(n)$ EI's and a total space of $\Omega(n^2)$ is required

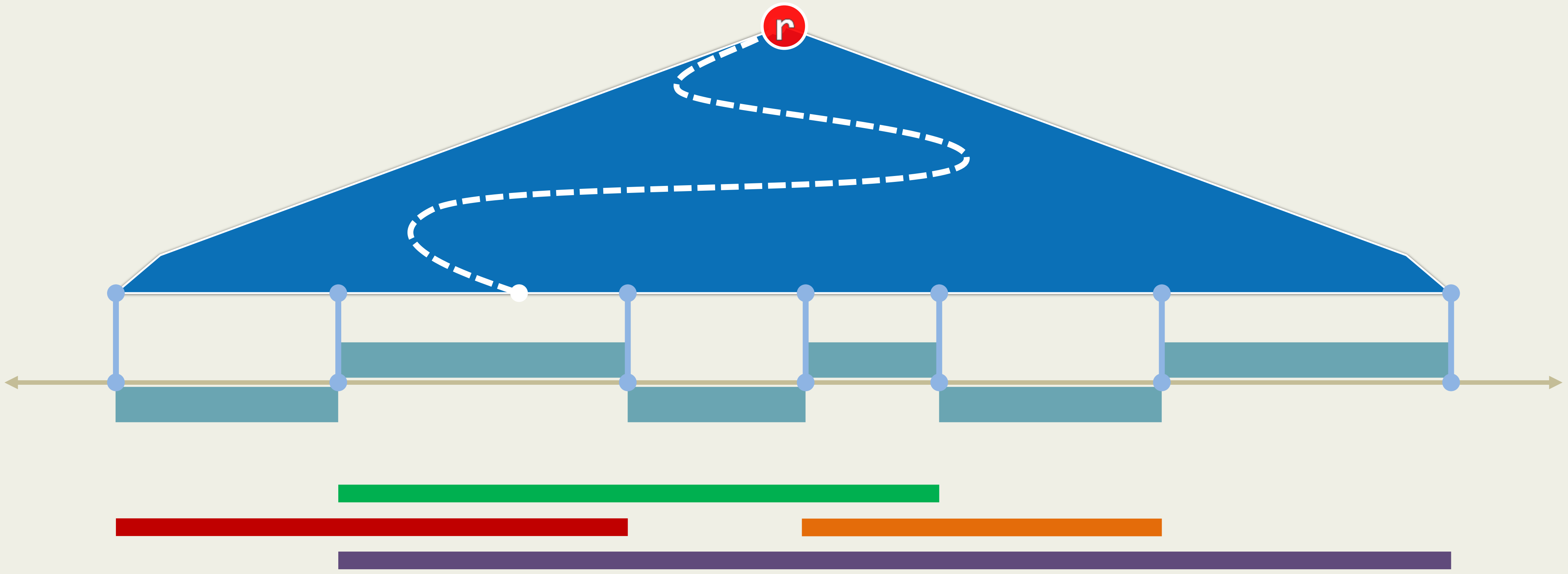


Sorted Vector --> BBST

❖ For each leaf v,

denote the corresponding elementary interval as R(v), //range of domination

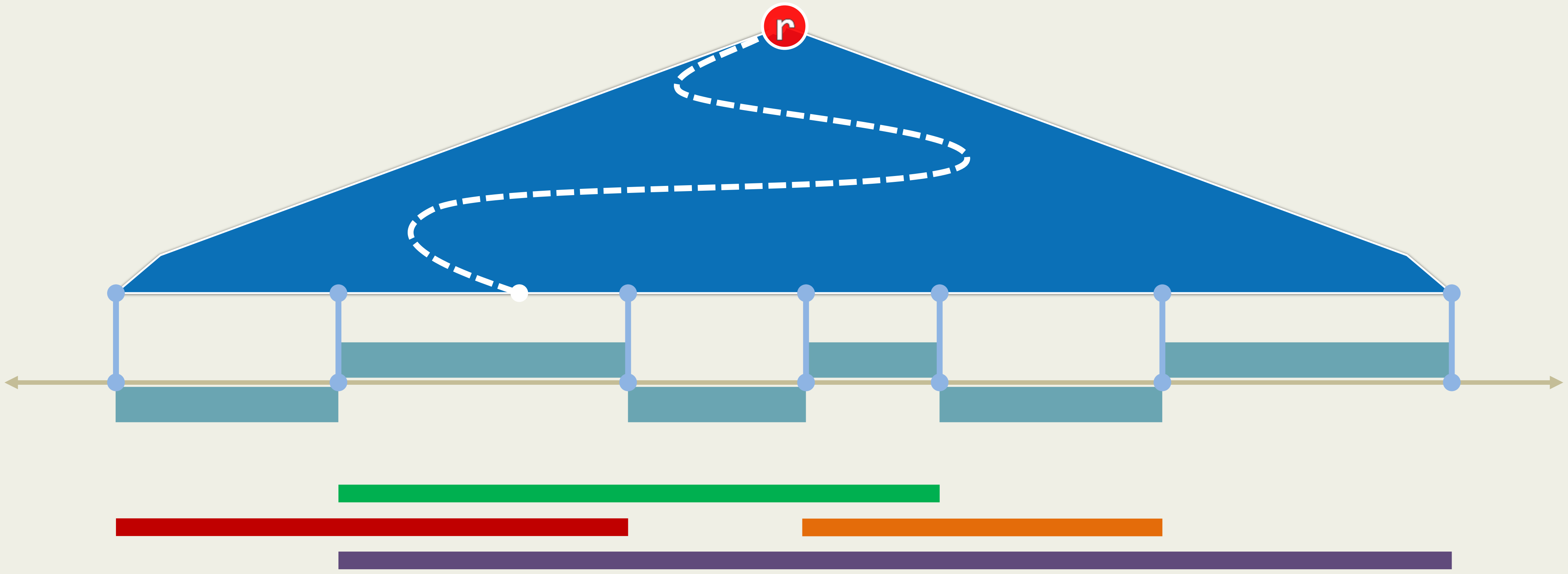
denote the subset of intervals spanning R(v) as Int(v) and store Int(v) at v



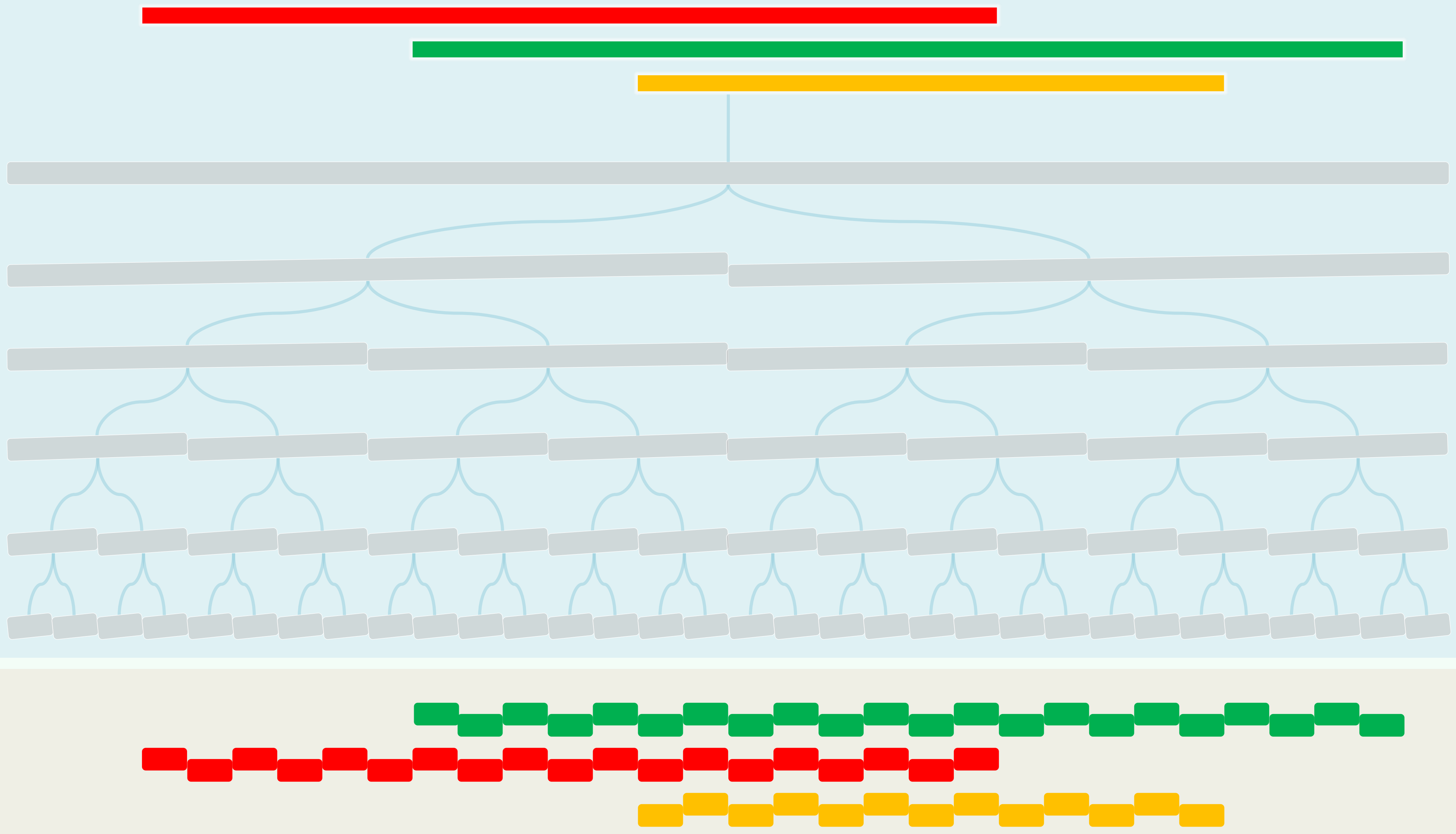
1D Stabbing Query with BBST

❖ To find all intervals containing q_x, we can

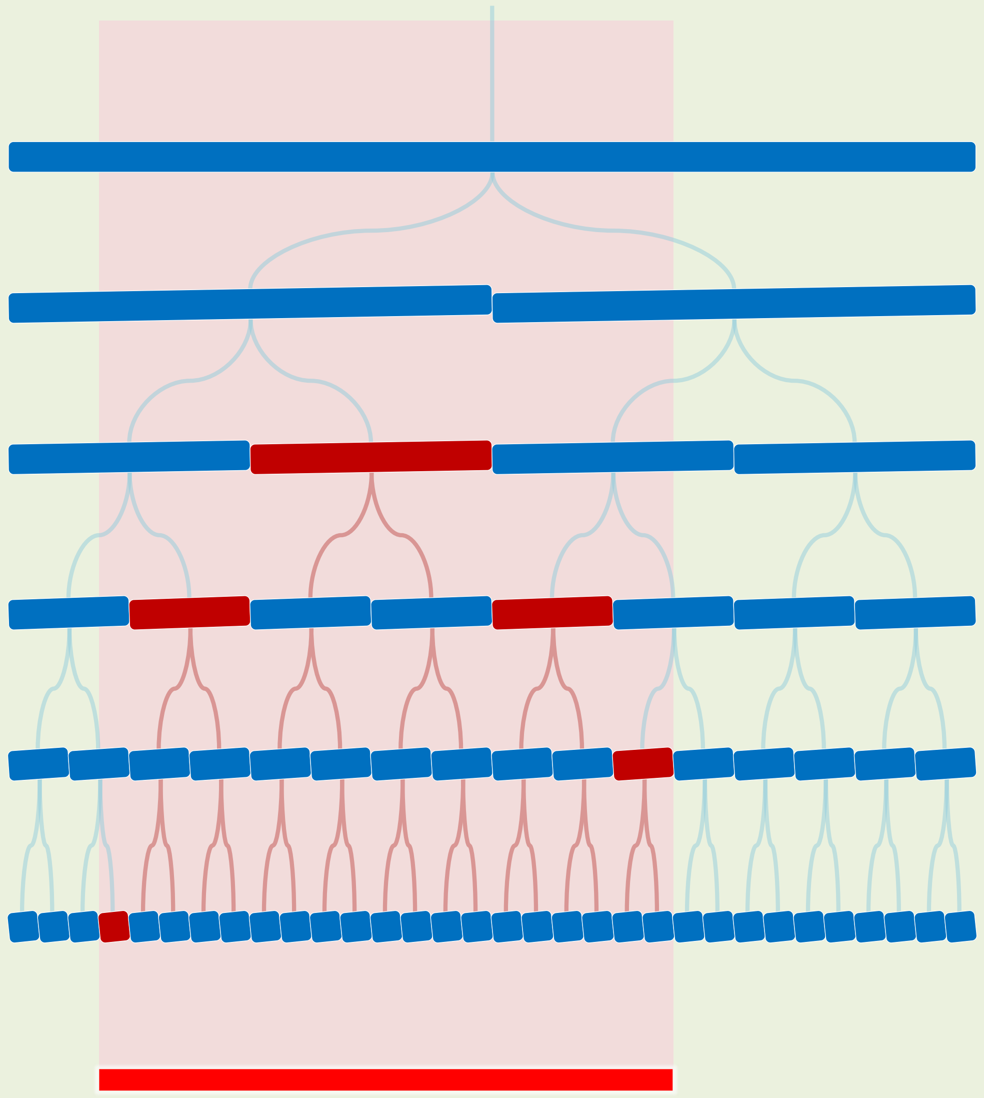
- find the leaf v whose R(v) contains q_x //$\mathcal{O}$(logn) time for a BBST
- and then report Int(v) //$\mathcal{O}$(1 + r) time

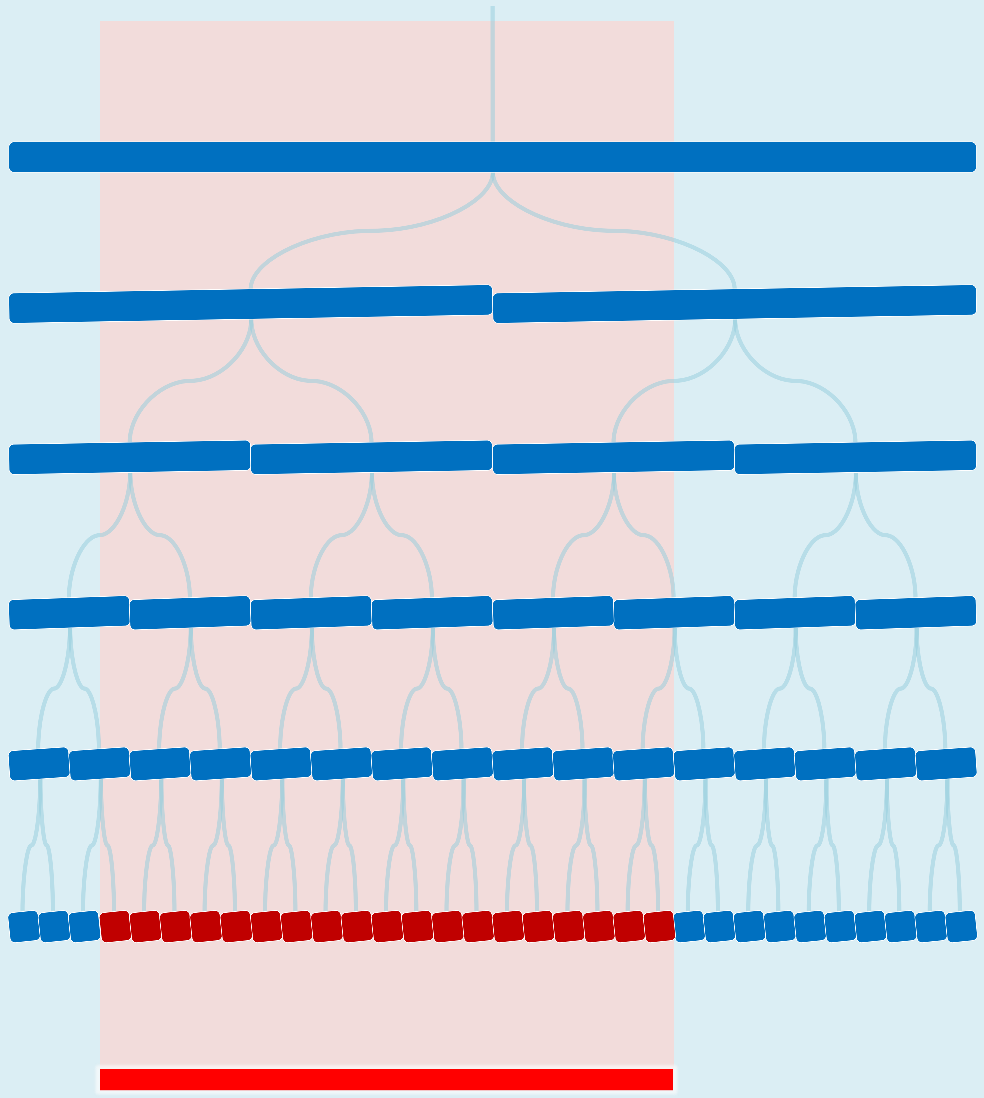


$\Omega(n^2)$ Total Space In The Worst Cases

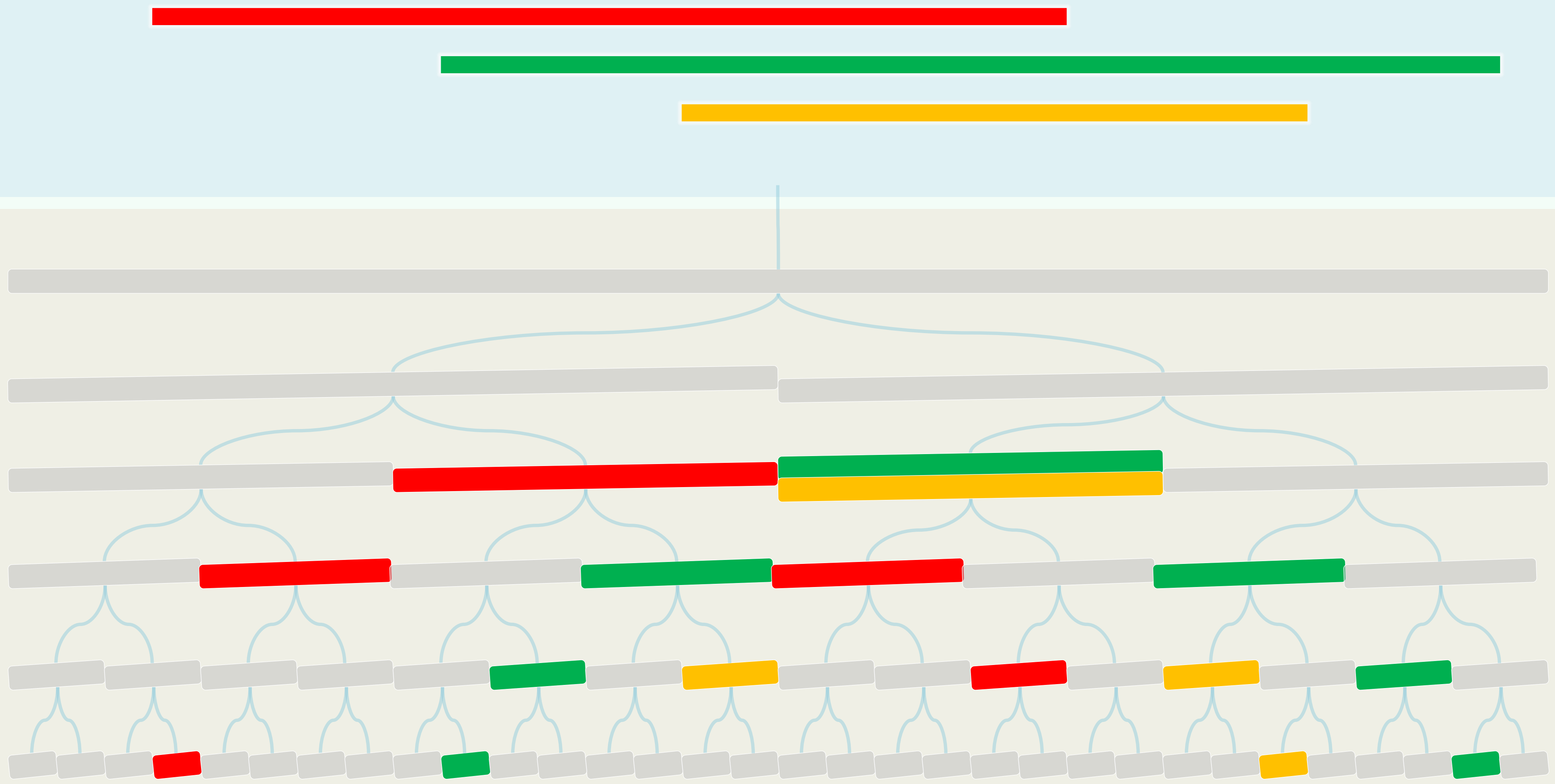


Store each interval at $\mathcal{O}(\log n)$ common ancestors by greedy merging



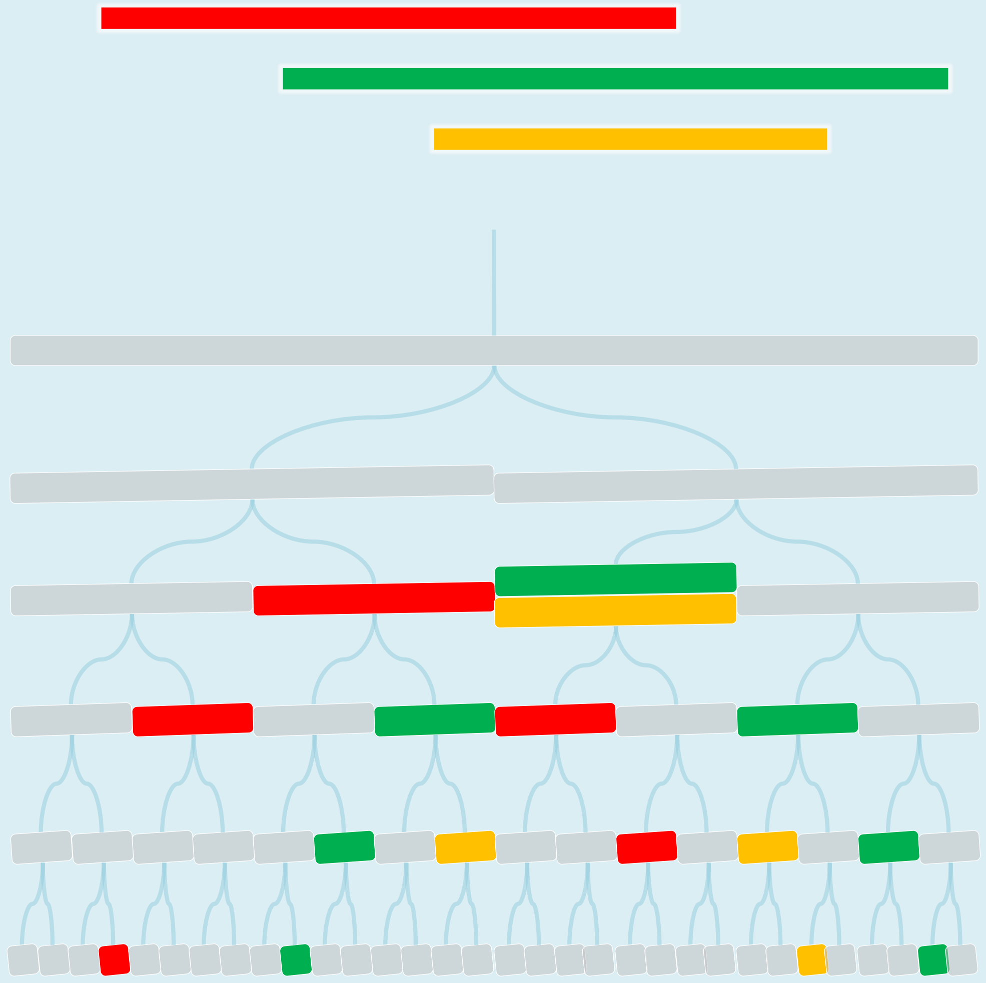


Canonical Subsets with $\mathcal{O}(n\log n)$ Space



BuildSegmentTree(I)

```
Sort all endpoints in I before
   determining all the EI's //O(nlogn)
Create T a BBST on all the EI's //O(n)
Determine R(v) for each node v
   //O(n) if done in a bottom-up manner
For each s of I
   InsertSegment( T.root, s )
```



InsertSegment(v , s)

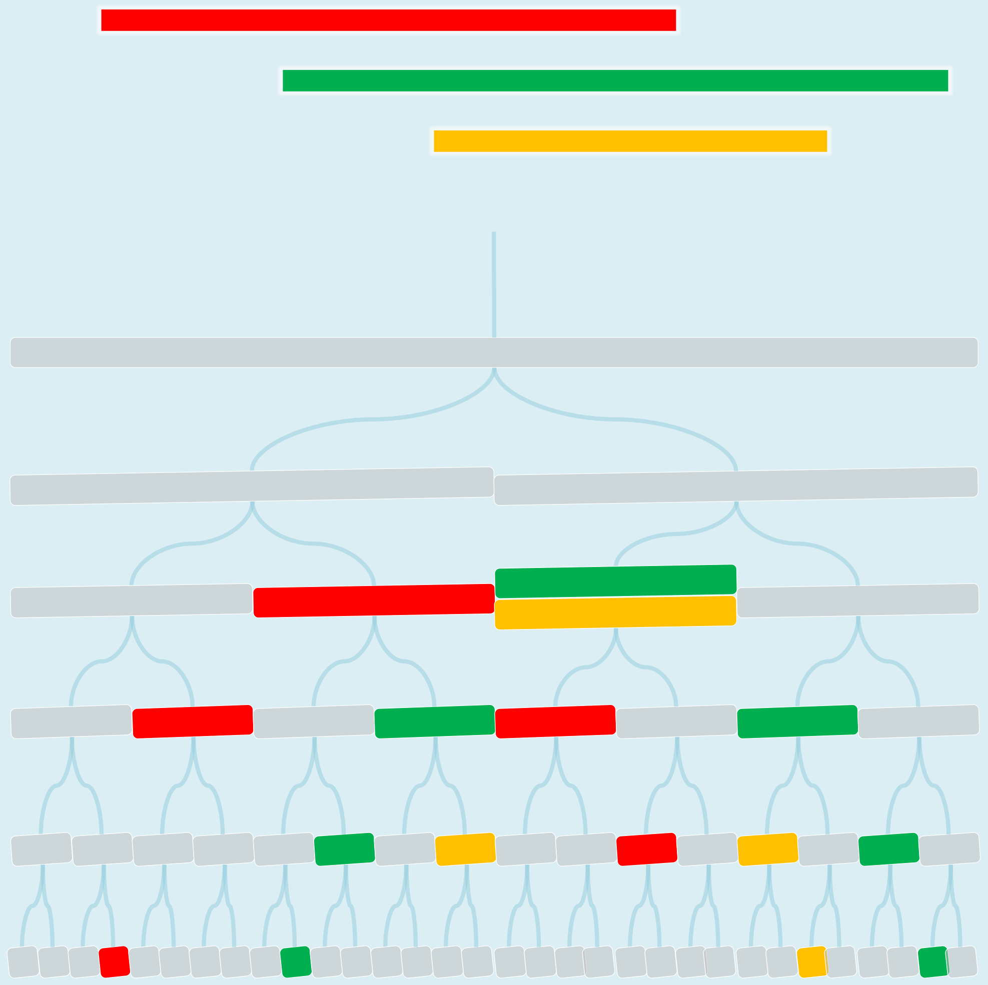
```
if ( R(v) ⊆ s ) //greedy by top-down
   store s at v and return;
if ( R( lc(v) ) ∩ s ≠ ∅ ) //recurse
   InsertSegment( lc(v), s );
if ( R( rc(v) ) ∩ s ≠ ∅ ) //recurse
   InsertSegment( rc(v), s );
```

👁 At each level,

$\leq$ 4 nodes are visited

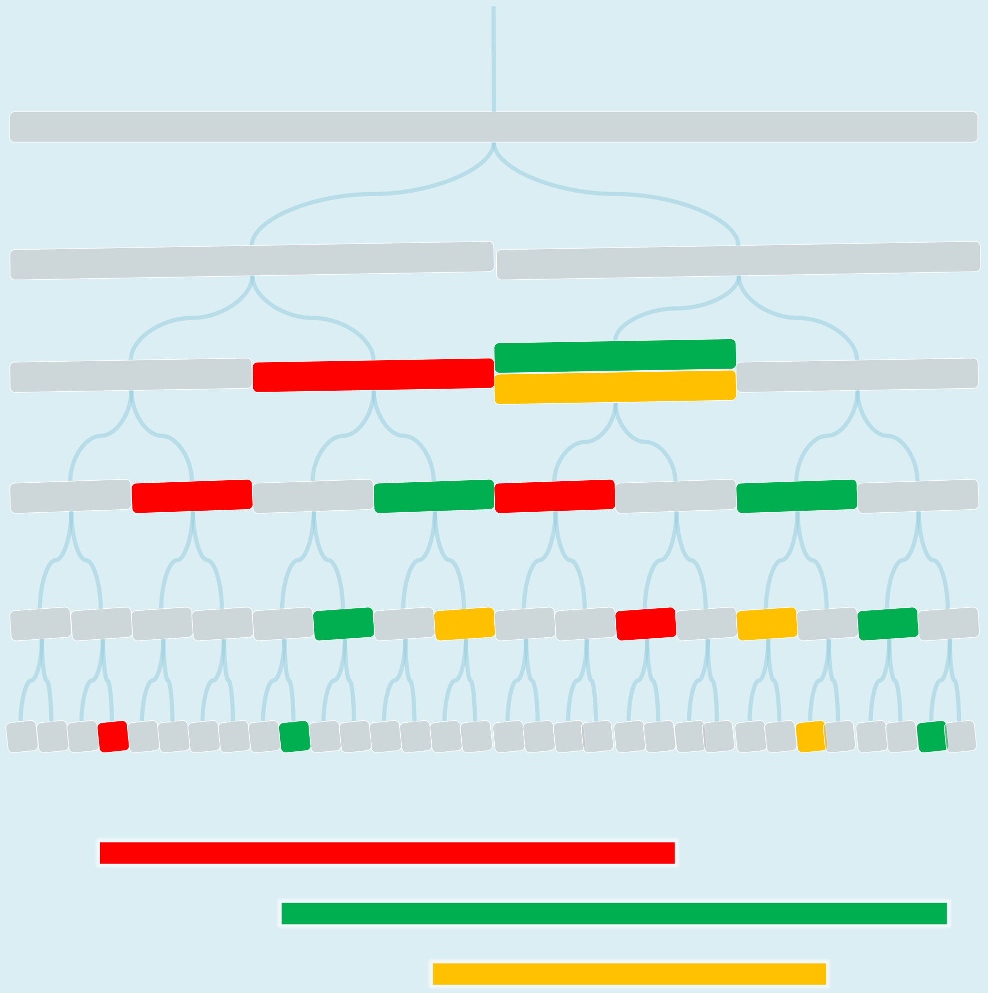
(2 stores + 2 recursions)

∴ $\mathcal{O}(\log n)$ time



Query(v , q_x)

```
report all the intervals in Int(v)
if ( v is a leaf )
    return
if ( $q_x \in$ R( lc(v) ) )
    Query( lc(v), $q_x$ )
else //$q_x \in$ R( rc(v) )
    Query( rc(v), $q_x$ )
```



$\mathcal{O}(r + \log n)$

- Only one node is visited per level,

  altogether $\mathcal{O}(\log n)$ nodes

- At each node v

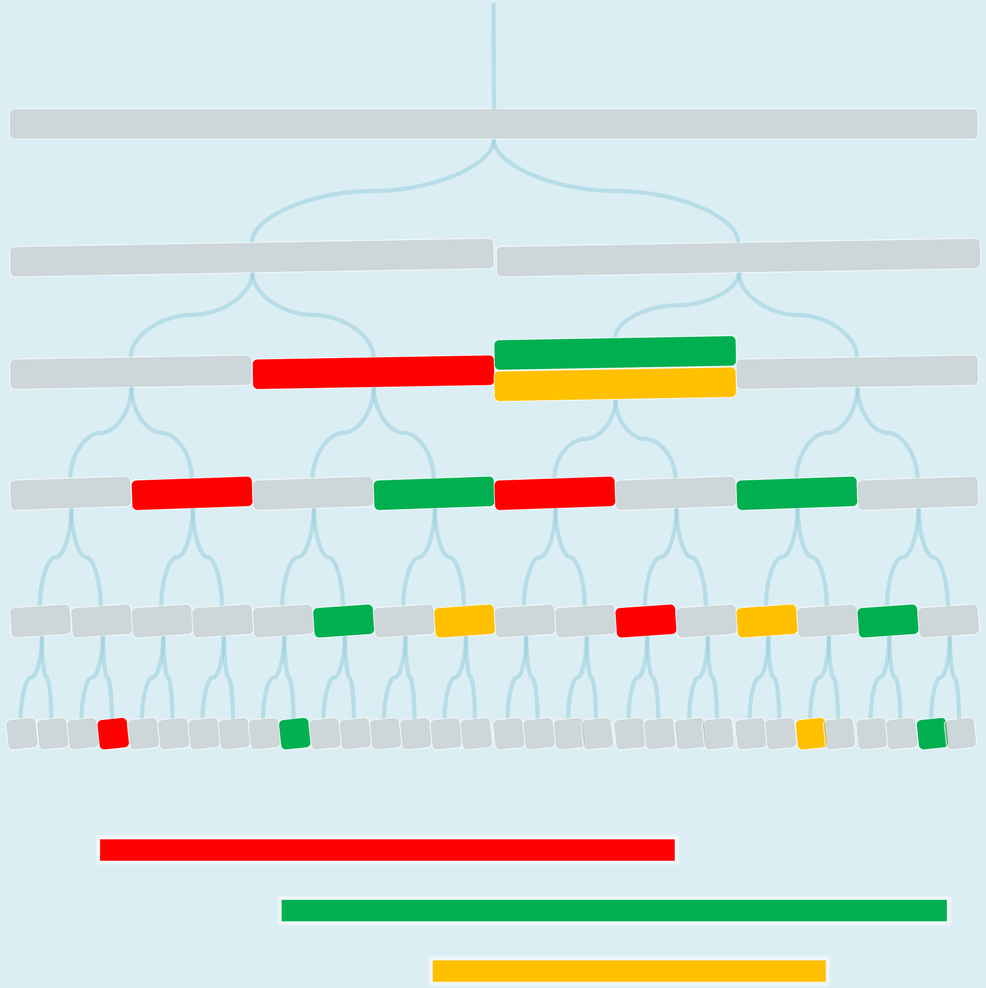
  - the CS Int(v) is reported

  - in time

    $$1 + |\text{Int}(v)| = \mathcal{O}(1 + r_v)$$

∴ Reporting all the intervals

costs $\mathcal{O}(r)$ time



Conclusion

❖ For a set of n intervals,

- a segment tree of size $\mathcal{O}(n\log n)$
- can be built in $\mathcal{O}(n\log n)$ time
- which reports all intervals
  containing a query point
  in $\mathcal{O}(r + \log n)$ time