栈与队列

直方图内最大矩形

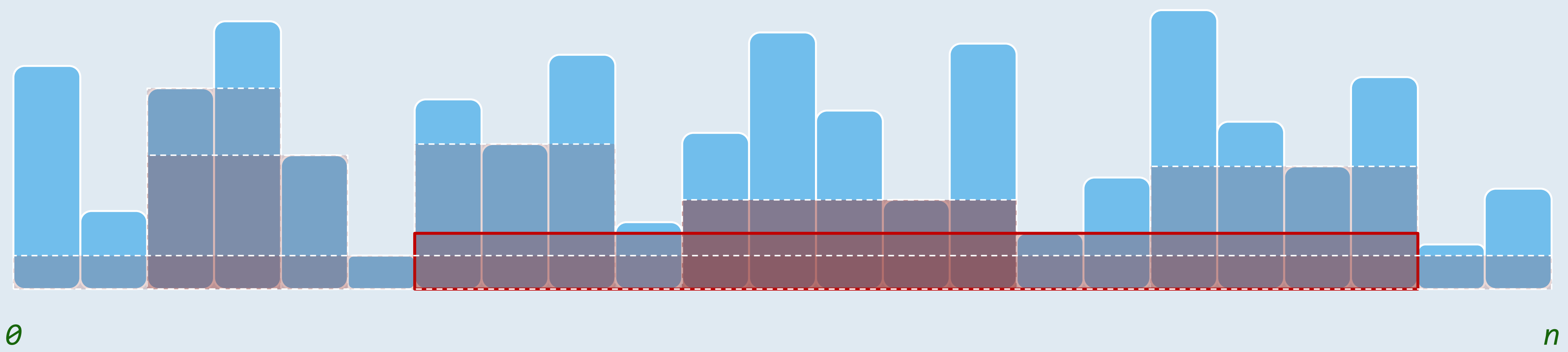
04-XC

就这么着，我有了一所严丝密缝、涂抹灰泥的木板房子，七英尺宽，十五英尺长，立柱有八英尺高...

这时，公共智慧的结果便产生理智与意志在社会体中的结合，也才有了各个部分的密切配合，以及最后全体的最大力量。

邓俊辉
deng@tsinghua.edu.cn

Maximum Rectangle



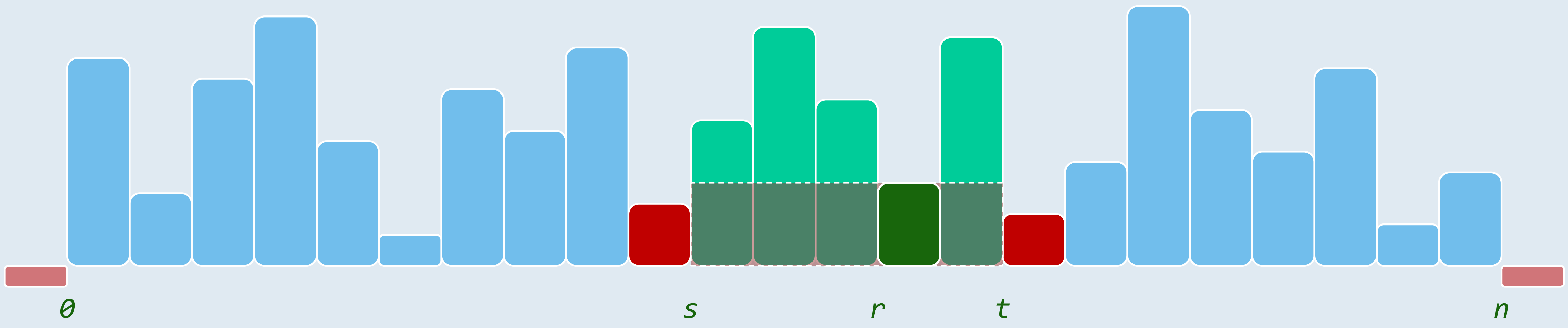
❖ Let H[0,n) be a histogram of non-negative integers

❖ How to find the largest orthogonal rectangle in H[]?

❖ To eliminate possible ambiguity

we can, for example, choose the **LEFTMOST** one

Maximal Rectangles



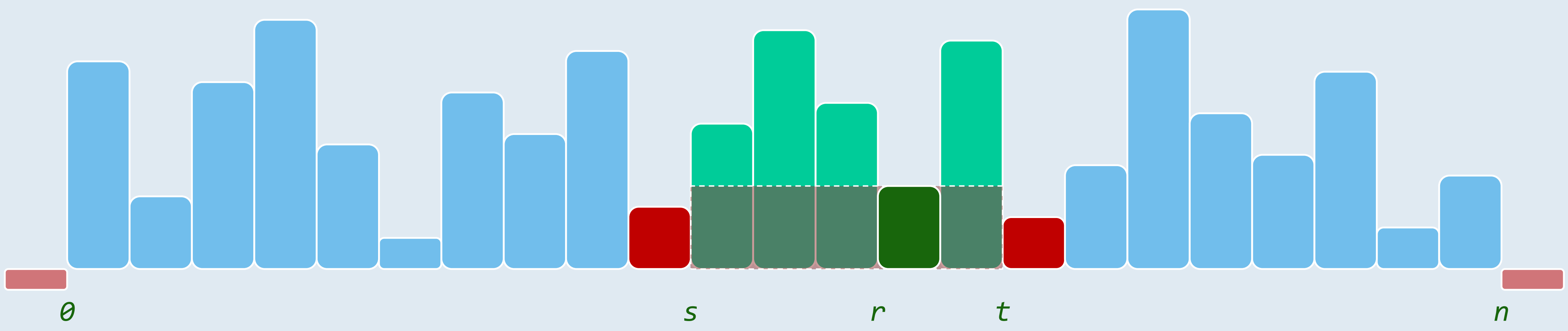
❖ **Maximal rectangle supported by H[r]:** $maxRect(r) = H[r] \cdot (t(r) - s(r))$

$$s(r) = \max\{\ k \mid 0 \le k \le r \text{ and } H[k-1] < H[r]\ \}$$

where

$$t(r) = \min\{\ k \mid r < k \le n \text{ and } H[r] > H[k]\ \}$$

Brute-force



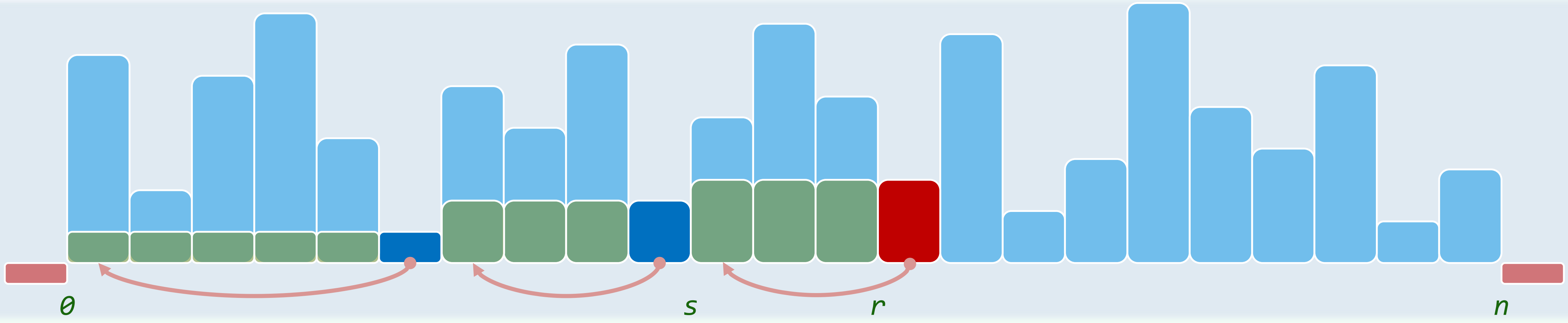
❖ Determining s(r) and t(r) for all r's requires $\mathcal{O}(n^2)$ time

$$s(r) = \max\{\ k \mid 0 \le k \le r \text{ and } H[k-1] < H[r]\ \}$$

$$t(r) = \min\{\ k \mid r < k \le n \text{ and } H[r] > H[k]\ \}$$

❖ Actually, we can do this even faster ...

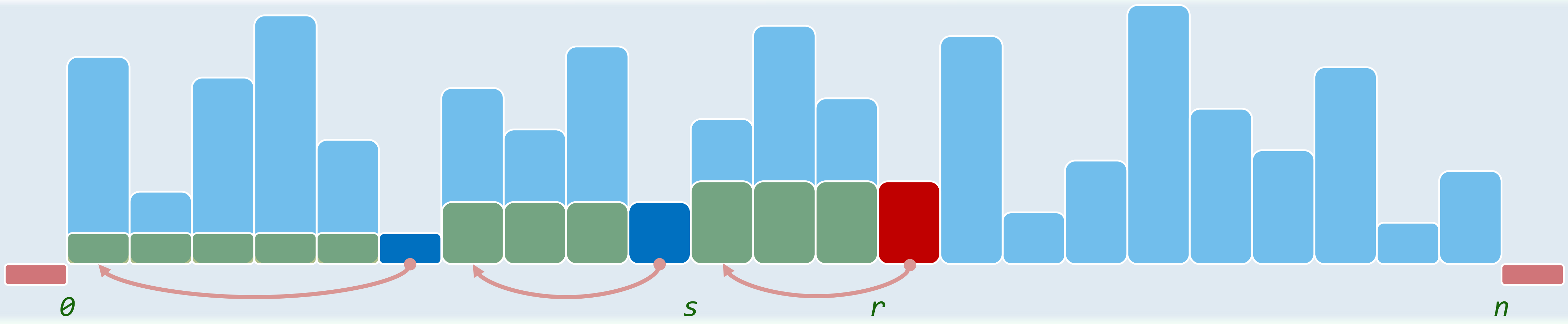
Using Stack: Algorithm



- All s(r)'s can be determined by a **LINEAR** scan of the histogram

```
int* s = new int[n]; Stack<Rank> S;
for ( int r = 0; r < n; r++ ) //try using SENTINEL for simplicity by yourself
   while ( !S.empty() && H[S.top()] >= H[r] ) S.pop(); //until H[top] < H[r]
   s[r] = S.empty() ? 0 : 1 + S.top();    S.push(r); //S is always ASCENDING
while( !S.empty() ) S.pop();
```

Using Stack: Loop Invariant & Correctness



- **After each iteration of the outer loop, we always have**

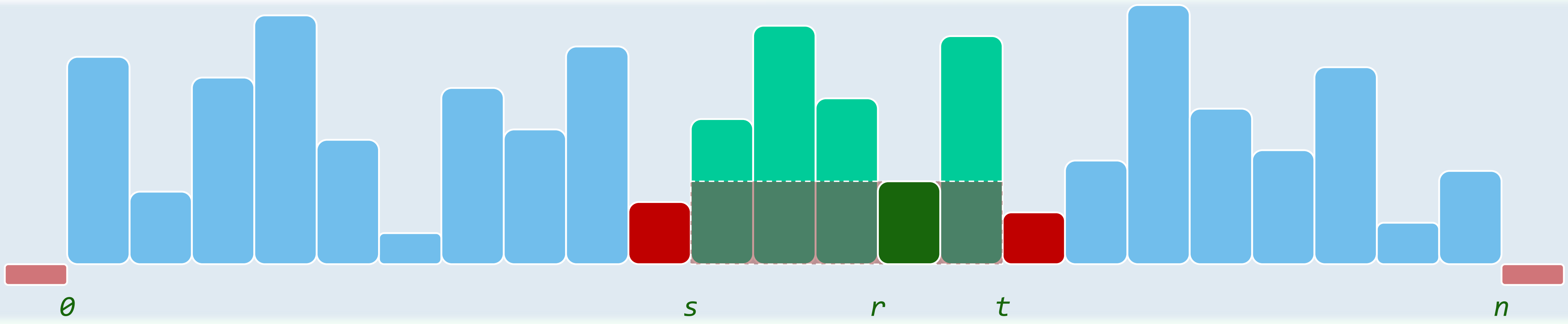
$$S[S.size()-1] = S.top() = r \qquad \textbf{and}$$

$$S[i-1] = s[S[i]] - 1 = \max\{\, k \mid 0 \le k < S[i] \text{ and } H[k] < H[S[i]] \,\} \quad (\text{when } 0 \le i < S.size())$$

- **Each r should be pushed into the stack and right before that, we have**

$$s[r] = 1 + S.top()$$

Using Stack: Complexity



❖ And t(r)'s can be determined by another scan in the **REVERSED** direction

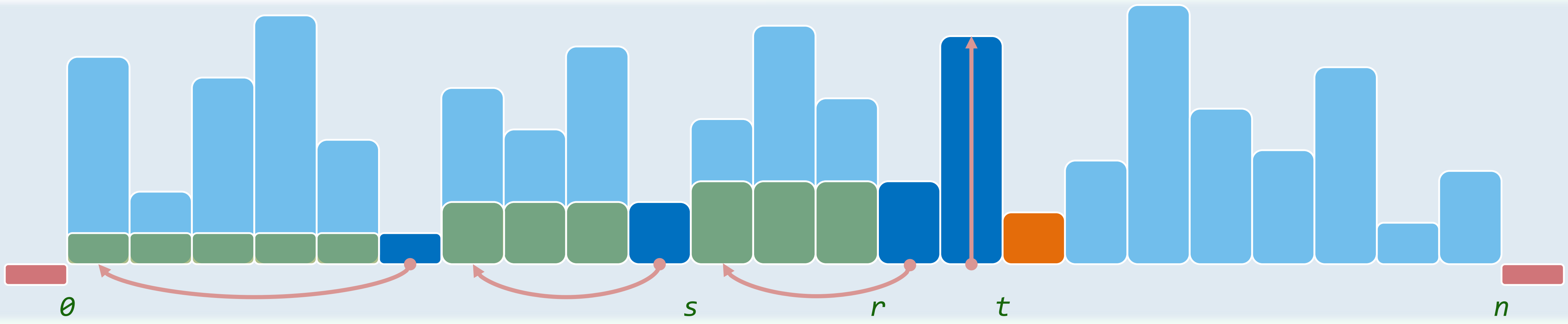
❖ Hence all maximal rectangles can be computed in $\mathcal{O}(n)$ time (by AMORTIZATION)

❖ However, what if the histogram is given in an **ON-LINE** manner?

In this case, the t(r)'s **CAN'T** be determined until the **ENTIRE** input is ready

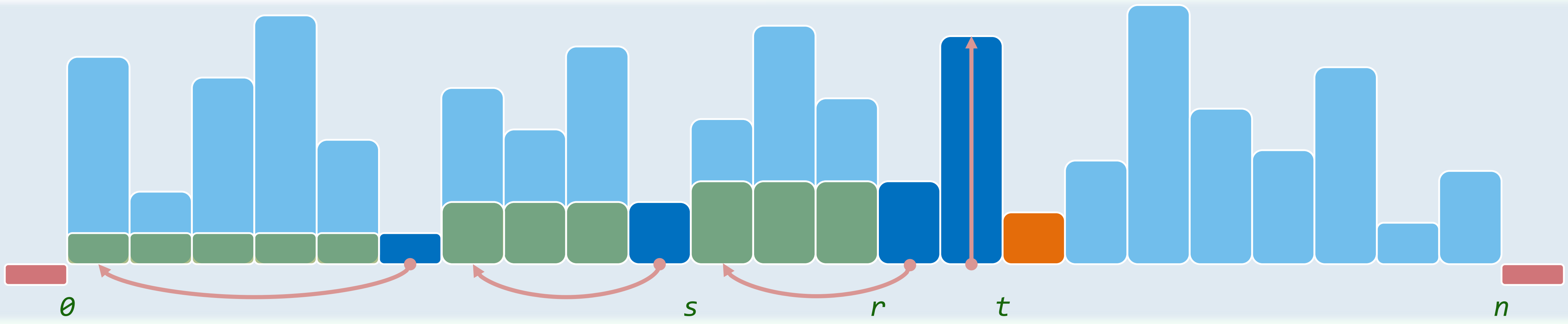
❖ Is it possible to compute **BOTH** s(r)'s and t(r)'s by a **SINGLE** scan? //on-fly

One-Pass Scan: Algorithm




```
❖ Stack<int> SR;  __int64 maxRect = 0; //SR.2ndTop() == r(s) & SR.top() == r
❖ for ( int t = 0; t <= n; t++ ) //amortized-O(n)
     while ( !SR.empty() && ( t == n || H[SR.top()] > H[t] ) )
        int r = SR.pop(); int s = SR.empty() ? 0 : SR.top() + 1;
        maxRect = max( maxRect, H[r] * ( t - s ) );
     if ( t < n ) SR.push( t );
  return maxRect;
```

One-Pass Scan: Loop Invariant & Correctness



- Again, after each iteration of the outer loop, we always have

$$S[S.size() - 1] = S.top() = t \quad \text{and}$$

$$S[i-1] = s[S[i]] - 1 \quad (\text{when } 0 \leq i < S.size())$$

- At the end of each iteration of the inner loop, we have

$$t[S.top()] = t \quad \text{and} \quad s[S.top()] = 1 + S[S.size() - 2]$$