

04.L

栈与队列

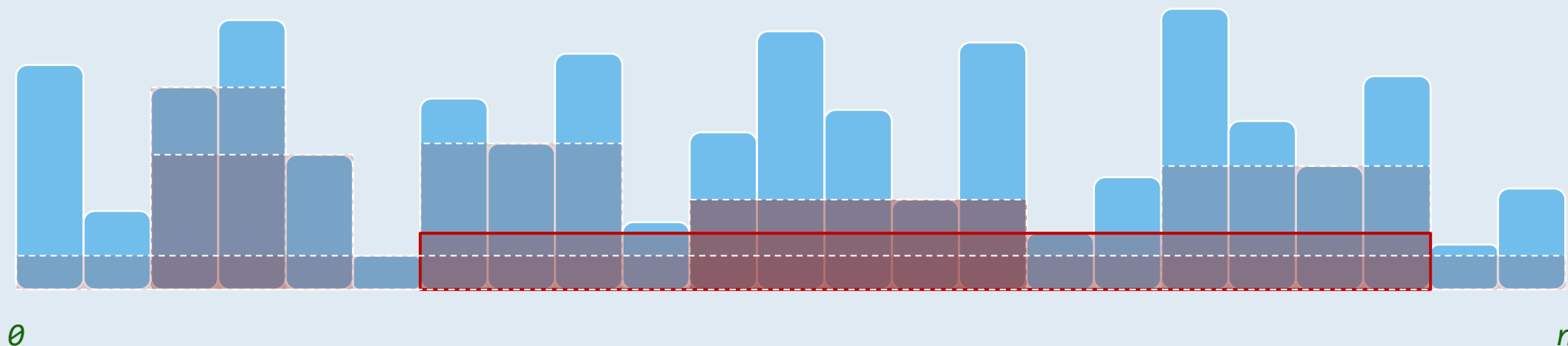
直方图内最大矩形

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...

邓俊辉

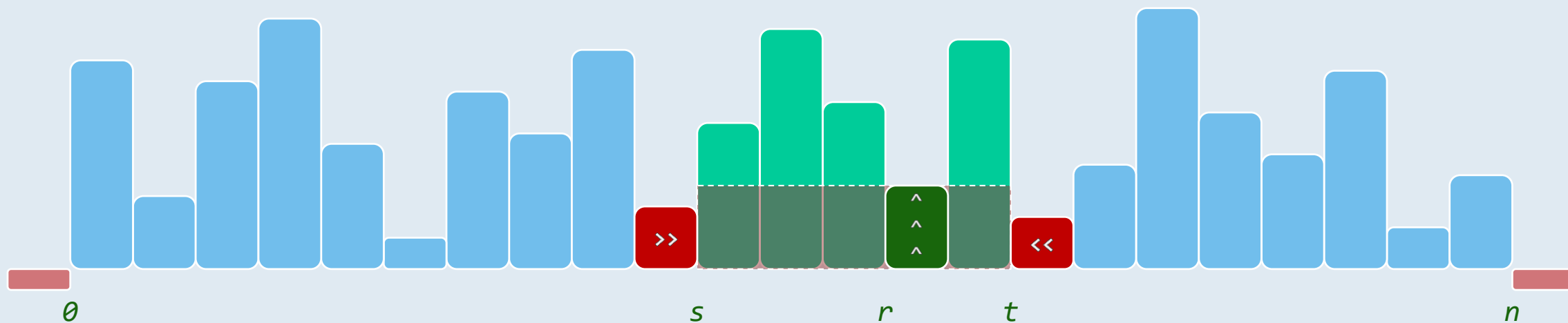
deng@tsinghua.edu.cn

# 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



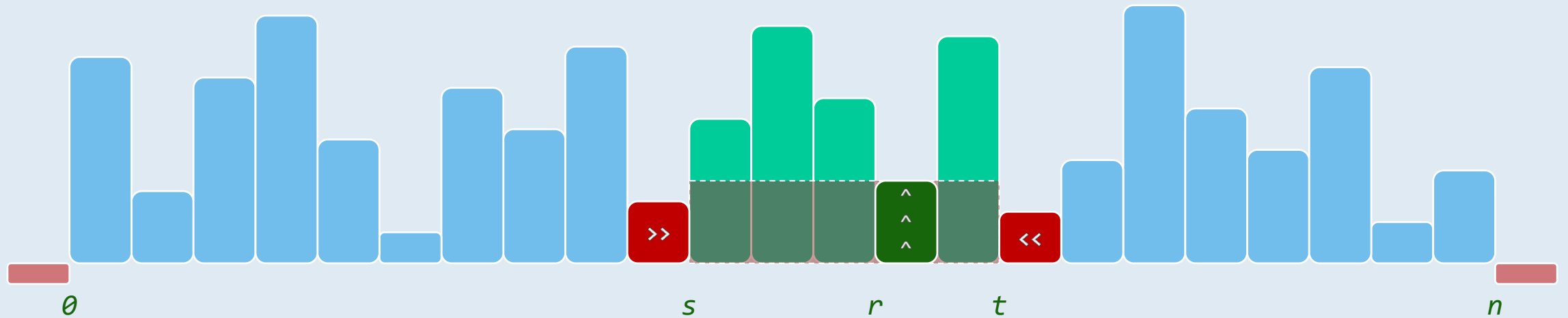
❖ Maximal rectangle supported by  $H[r]$ :  $maxRect(r) = H[r] \cdot (t(r) - s(r))$

where

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

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

## Brute-force



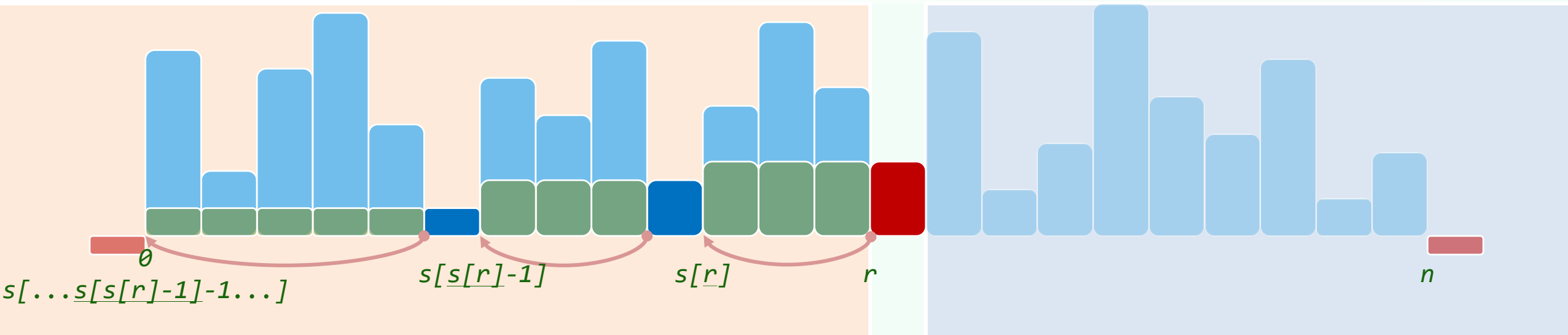
❖ Determining  $s(r)$  and  $t(r)$  for all  $r$ 's requires  $\mathcal{O}(n^2)$  time

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

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

❖ Actually, we can do this more efficiently ...

## 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;`

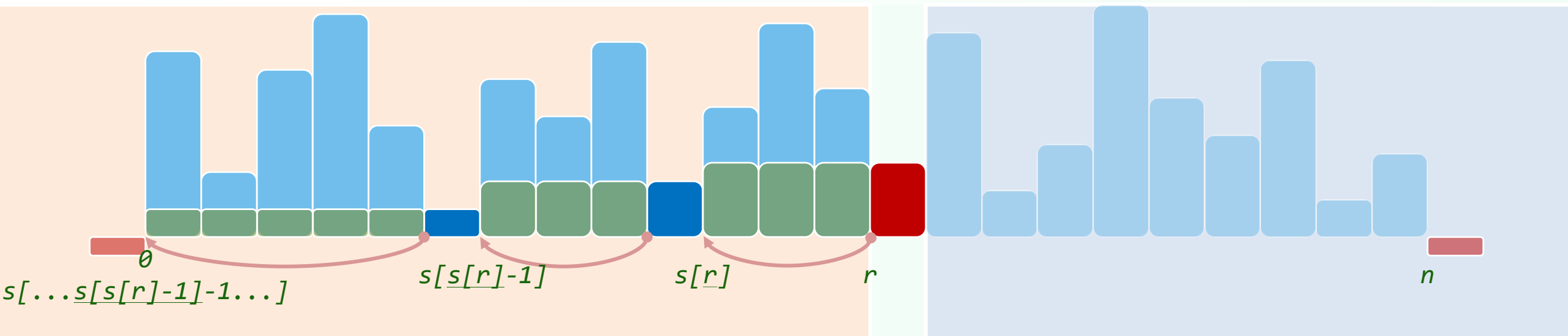
`for ( int r = 0; r < n; r++ ) //try using SENTINEL for simplicity by yourself`

`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



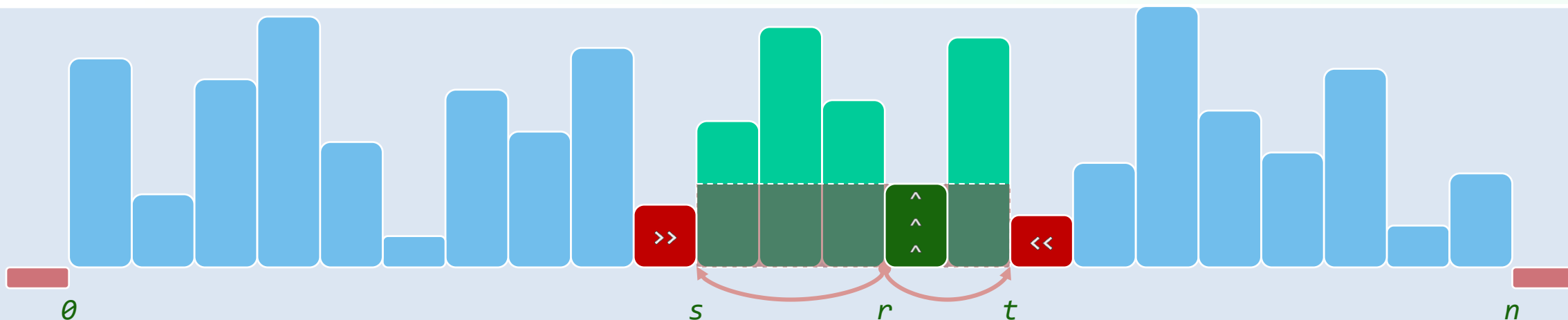
- ❖ All bars are scanned and pushed into  $S$  **in turn**
- ❖ After each iteration of the outer loop,  $S$  stores a **chain** in terms of  $s[]$

$$S[S.size() - 1] = S.top() = r \quad \text{and} \quad \forall 0 \leq k < S.size(), \quad S[k - 1] + 1 = s[S[k]]$$

- ❖ Every bar is popped when it will be of **no use** thereafter, i.e.

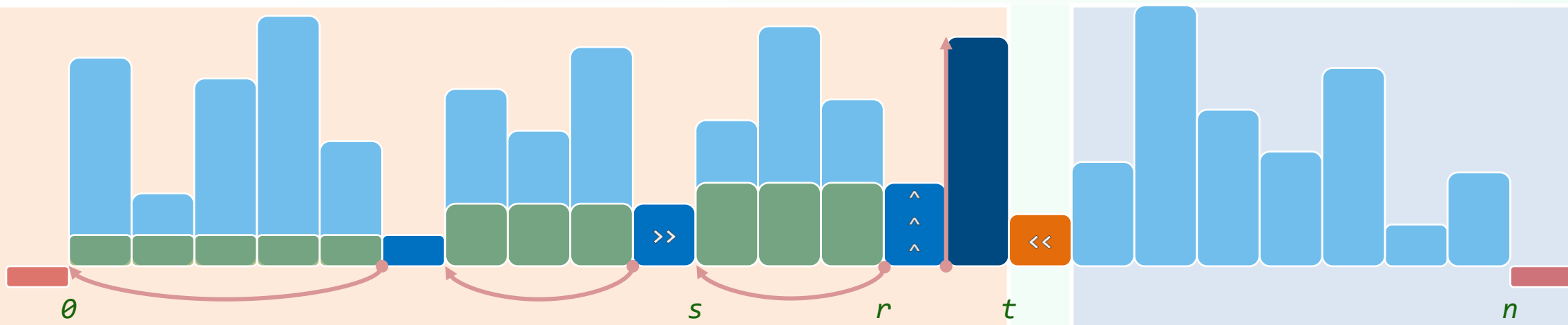
$$\forall 0 \leq k < r \text{ but } k \notin S, \quad \nexists r \leq t \text{ s.t. } k + 1 = s(t)$$

## 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() == s(r)-1 & SR.top() == r
```

```
❖ for ( int t = 0; t <= n; t++ ) //amortized- $\mathcal{O}(n)$ 
```

```
while ( !SR.empty() && ( t == n || H[SR.top()] > H[t] ) )
```

```
int r = SR.pop(); int s = SR.empty() ? 0 : SR.top() + 1;
```

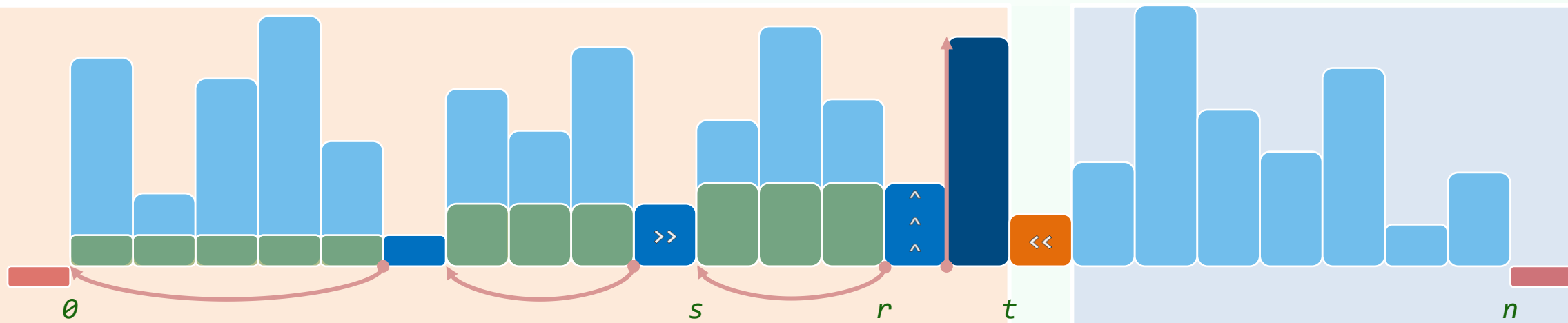
```
maxRect = max( maxRect, H[r] * ( t - s ) );
```

```
if ( t < n ) SR.push( t );
```

```
return maxRect;
```



## One-Pass Scan: Loop Invariant & Correctness



❖ Again, **at** each iteration of the **outer** loop, we always have

$$\forall 0 \leq k < SR.size(), \quad SR[k-1] + 1 = s[SR[k]]$$

- ❖ For each bar  $r$  popped in the inner loop, we have

$$t(r) = t \quad \mathbf{and} \quad s[r] = SR.top() + 1$$