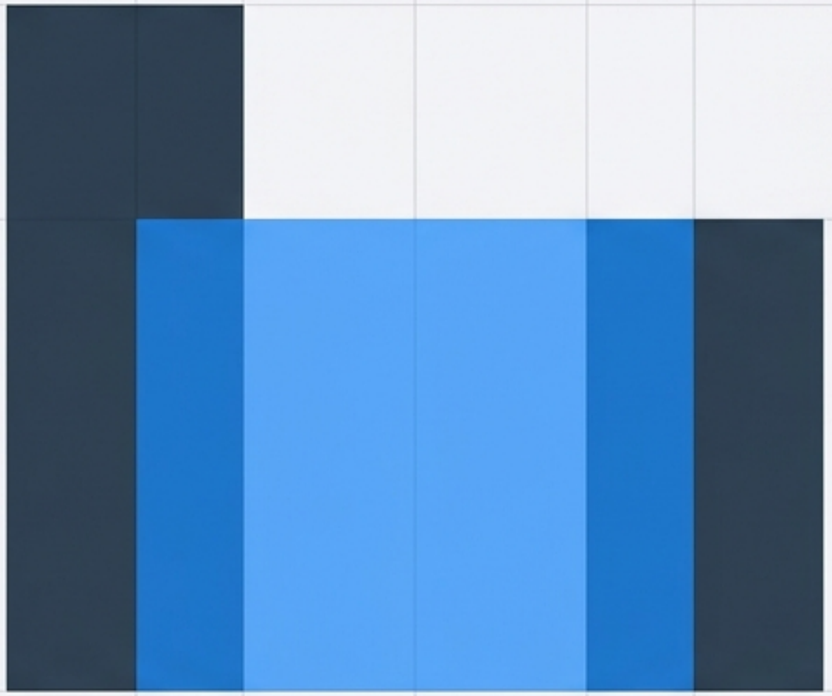
LeetCode 11: Container With Most Water

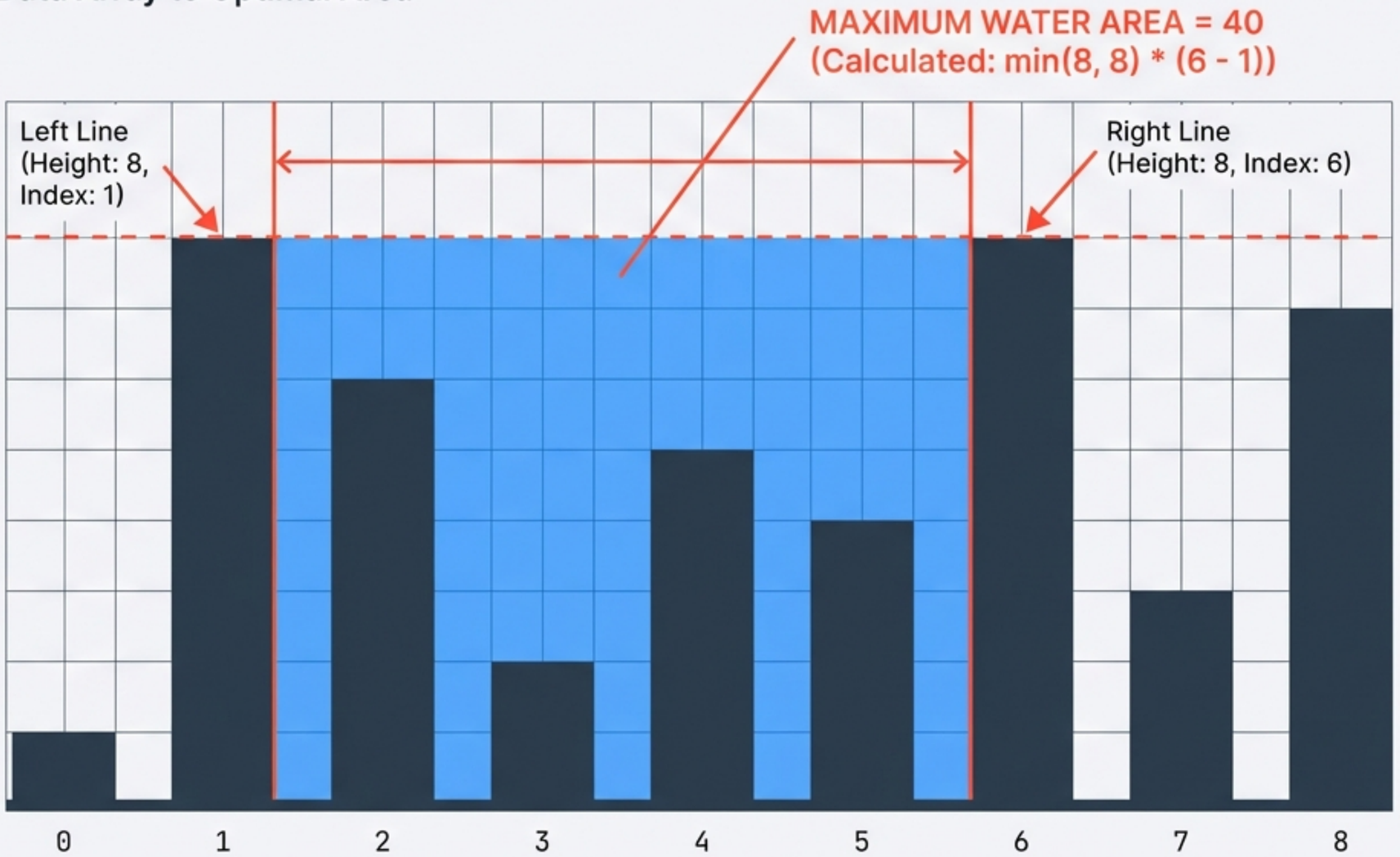


From Brute Force to Linear Time Optimization

MAX WATER CONTAINER VISUALIZATION

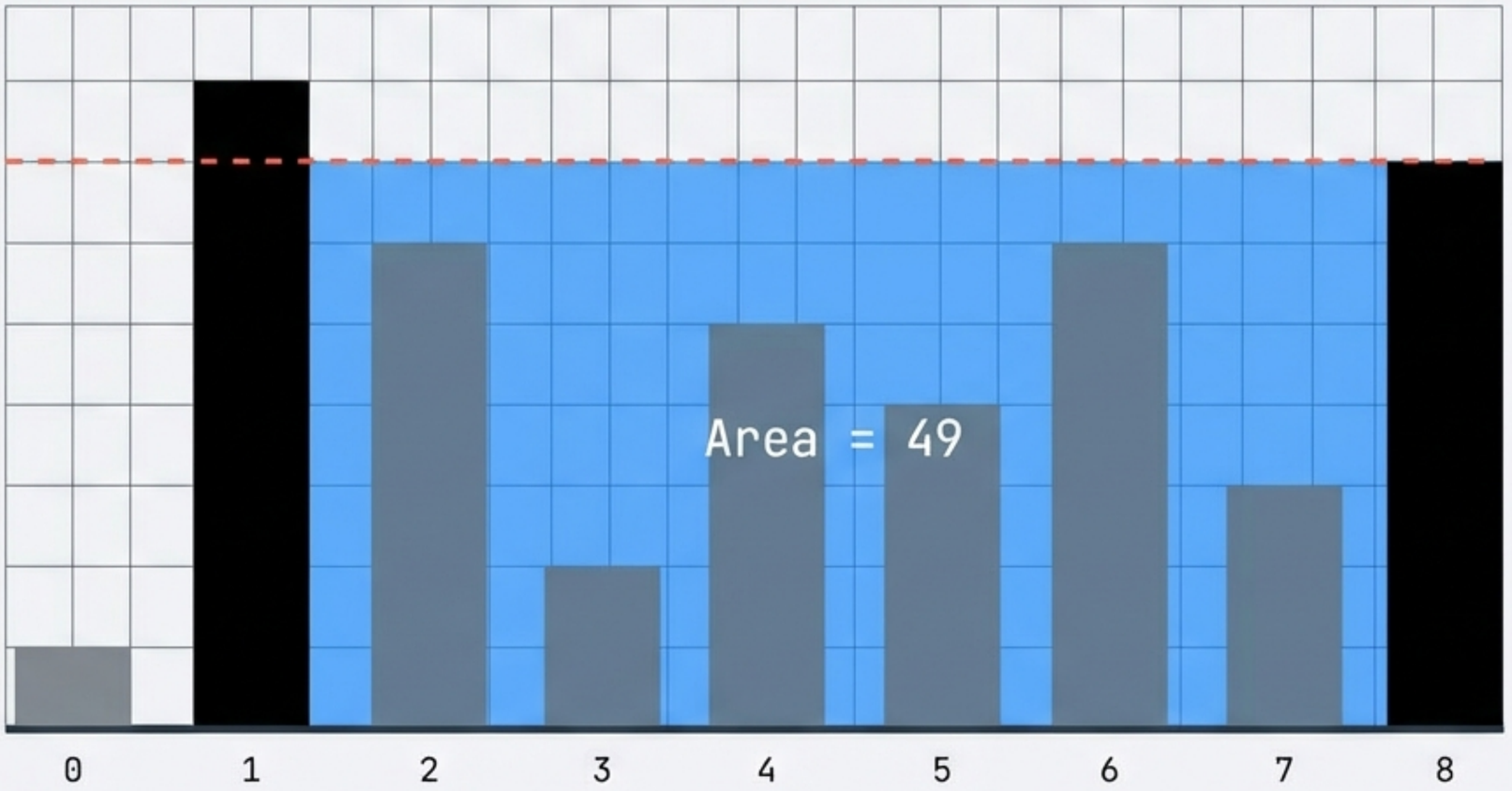
Swiss Technical Architectural | Data Array to Optimal Area

Input: An array of non-negative integers representing heights.
Goal: Find two lines that, together with the x-axis, form a container that holds the most water.



WATER AREA CALCULATION EXAMPLE

Swiss Technical Architectural | Detailed Container Analysis




```
Width
= Right Index (8) - Left Index (1)
= 7

Height
= min(Height[1], Height[8]) = 7

Area = 7 * 7 = 49
```

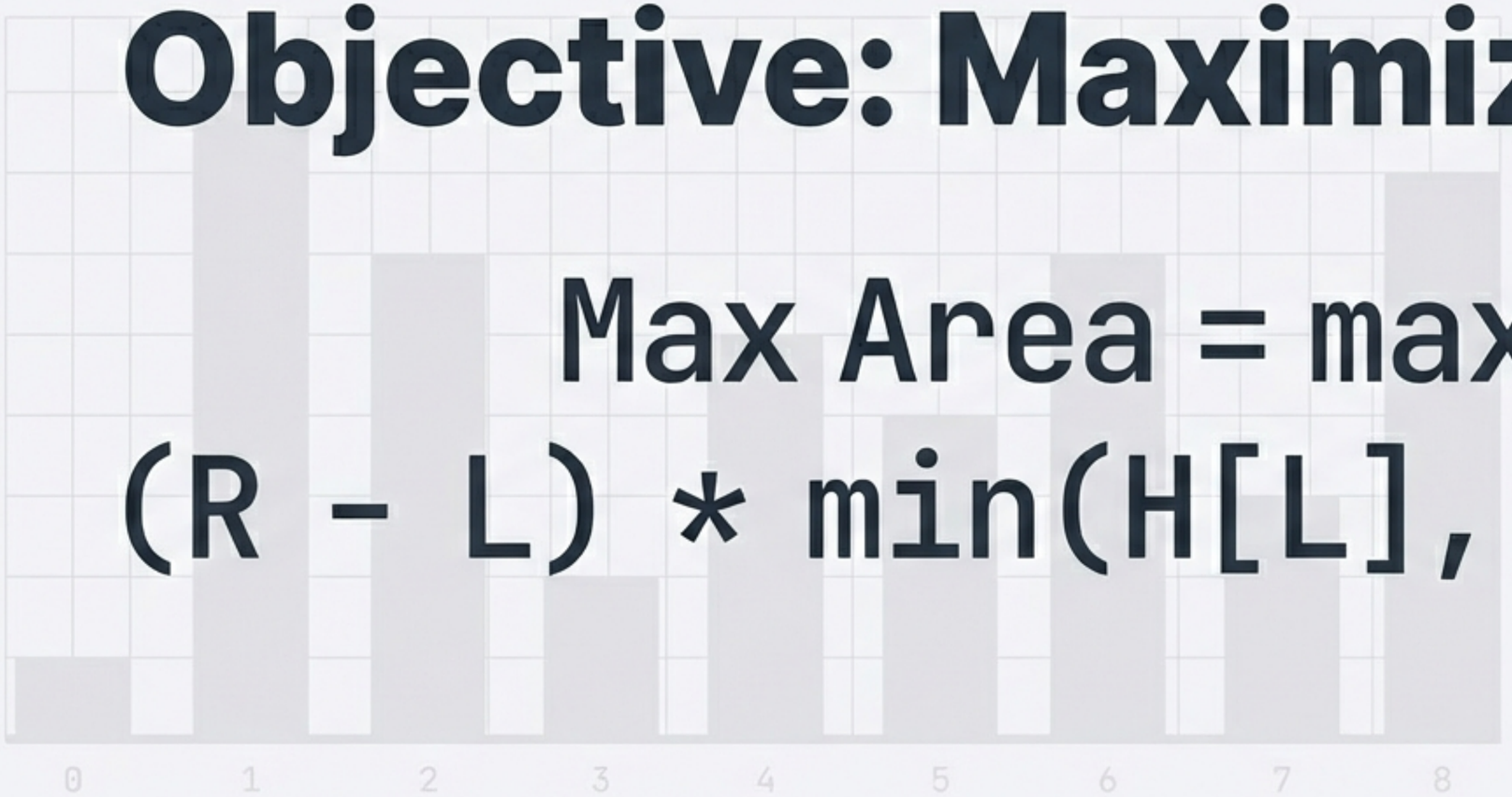
The container is limited by the shorter line (the bottleneck).

WATER AREA CALCULA

Swiss Technical Architectural | Detailed Container Analysis

Objective: Maximize Area

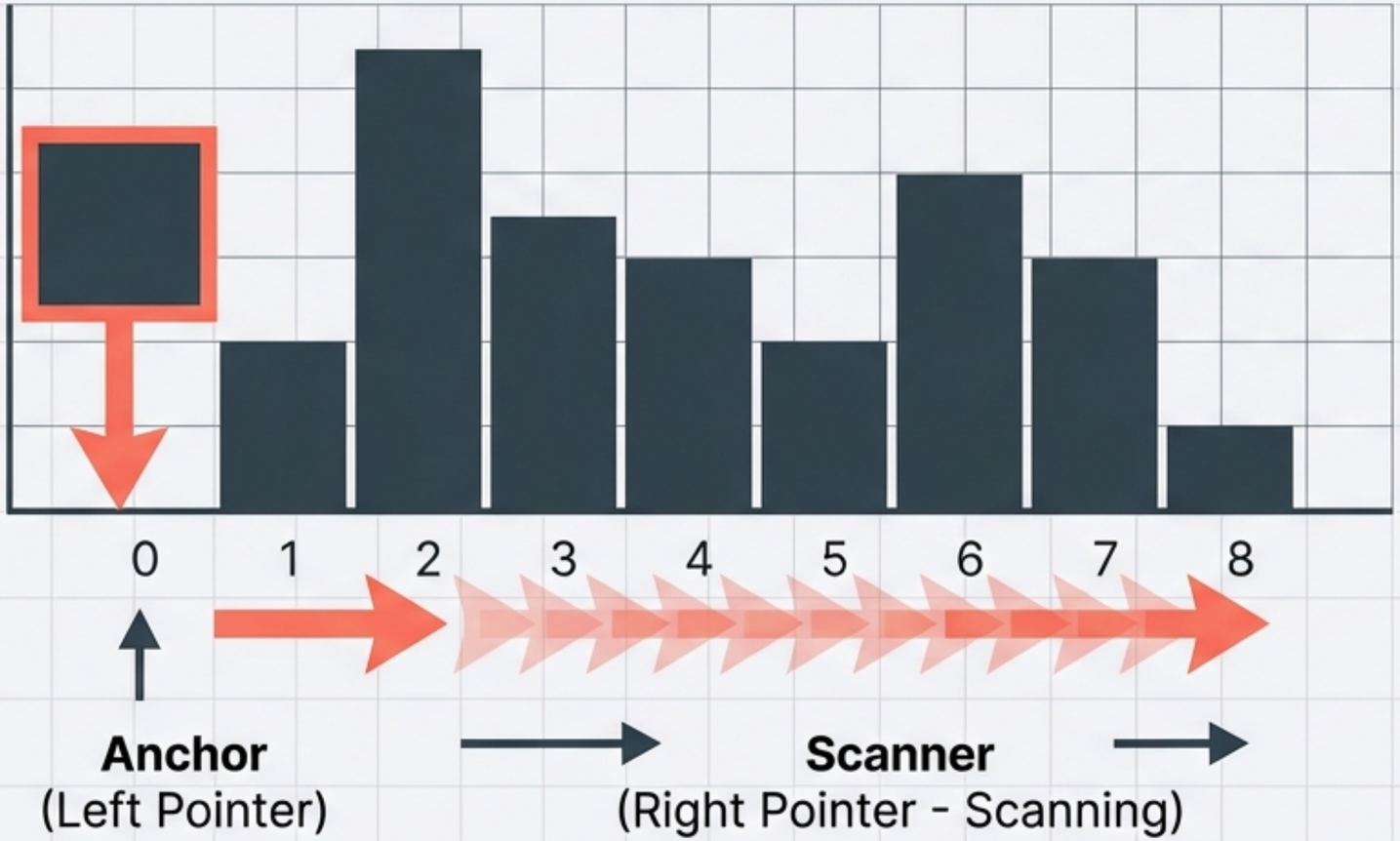
```
Max Area = max(
(R - L) * min(H[L], H[R]) )
```



Constraint: Vertical lines only. No slanting.

THE NAIVE APPROACH: BRUTE FORCE

Swiss Technical Architectural | Detailed Container Analysis



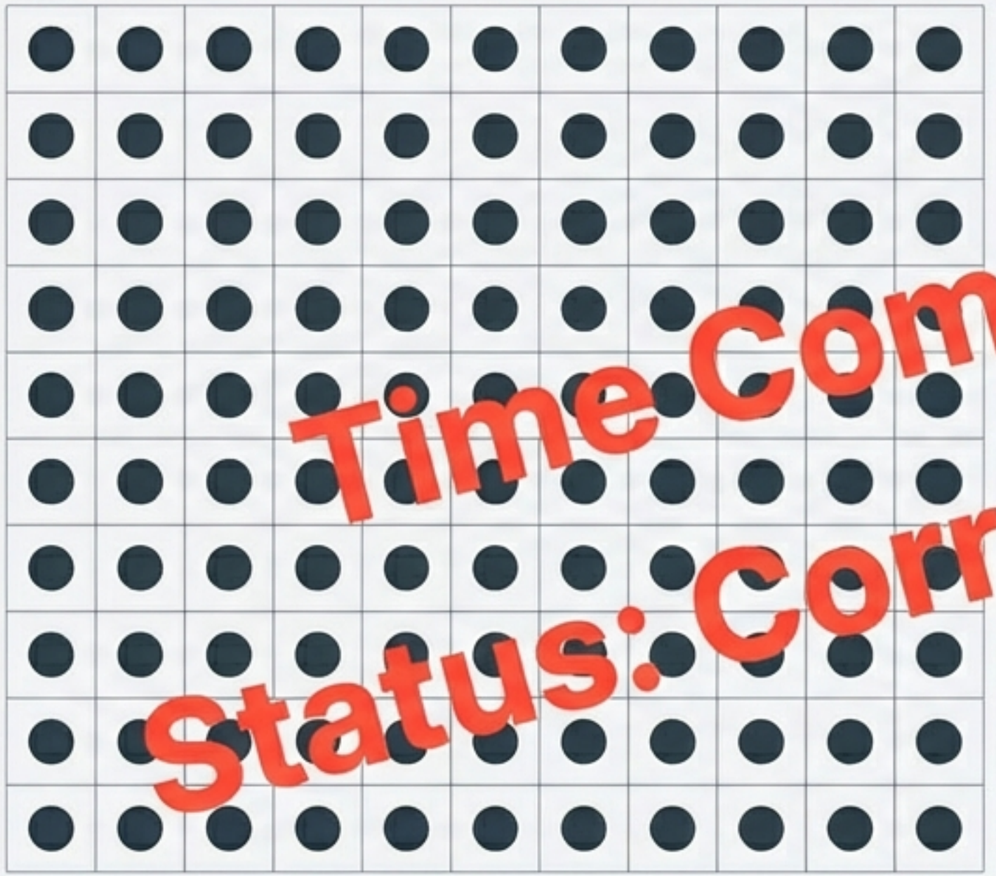
STRATEGY:

1. Fix Left pointer.
2. Scan every possible Right pointer.
3. Repeat for next Left pointer.

Check every single combination.

THE NAIVE APPROACH: COMPLEXITY ANALYSIS

Swiss Technical Architectural | Detailed Container Analysis



Time Complexity: $O(n^2)$

Status: Correct, but too slow.

N = 10 → ~45 Comparisons

N = 1000 → ~500,000 Comparisons

THE NAIVE APPROACH: BRUTE FORCE

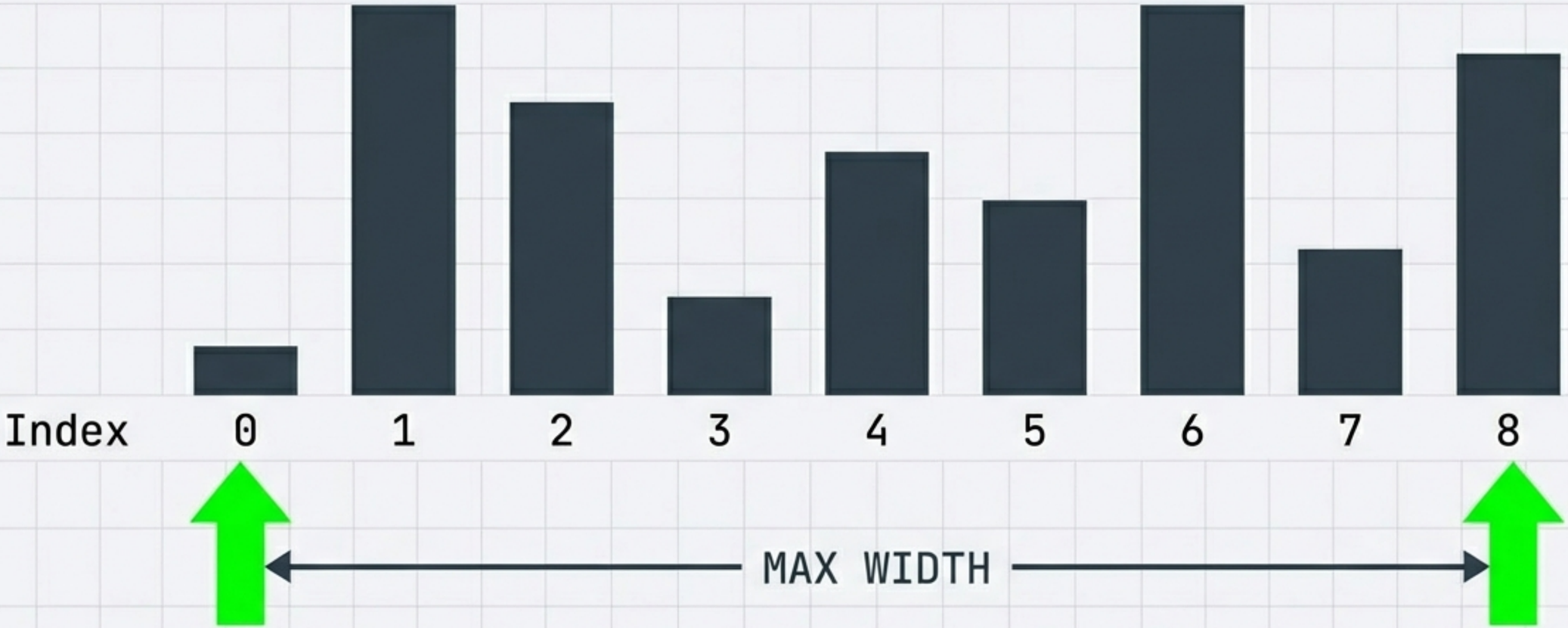
Swiss Technical Architectural | Detailed Container Analysis

Time Limit Exceeded (TLE)

We need a **paradigm shift.**
Can we solve this in a single pass O(n)?

THE TWO POINTER TECHNIQUE

Swiss Technical Architectural | Detailed Container Analysis

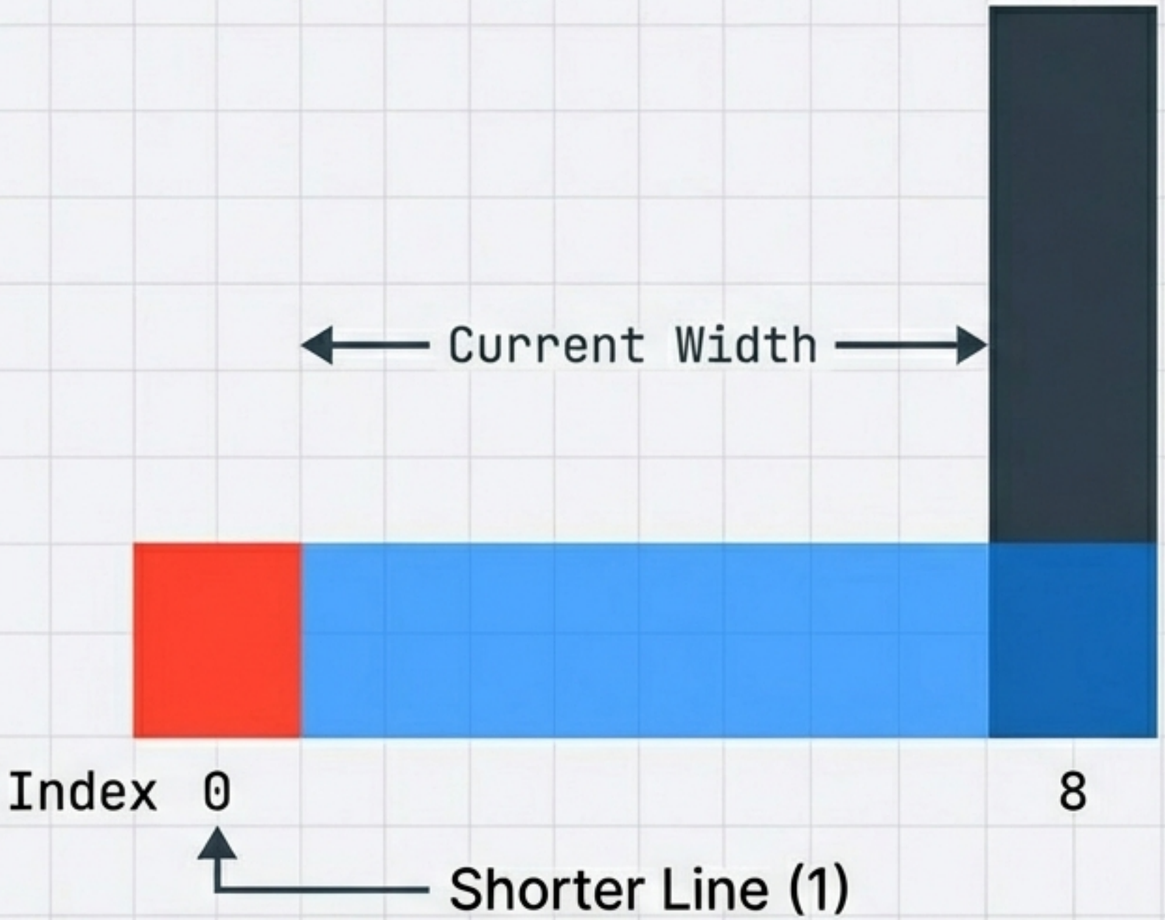


Start with the widest possible container.

Why? We maximize the width immediately. Now we only need to optimize for Height.

THE TWO POINTER TECHNIQUE

Swiss Technical Architectural | Detailed Container Analysis



THE BOTTLENECK LOGIC

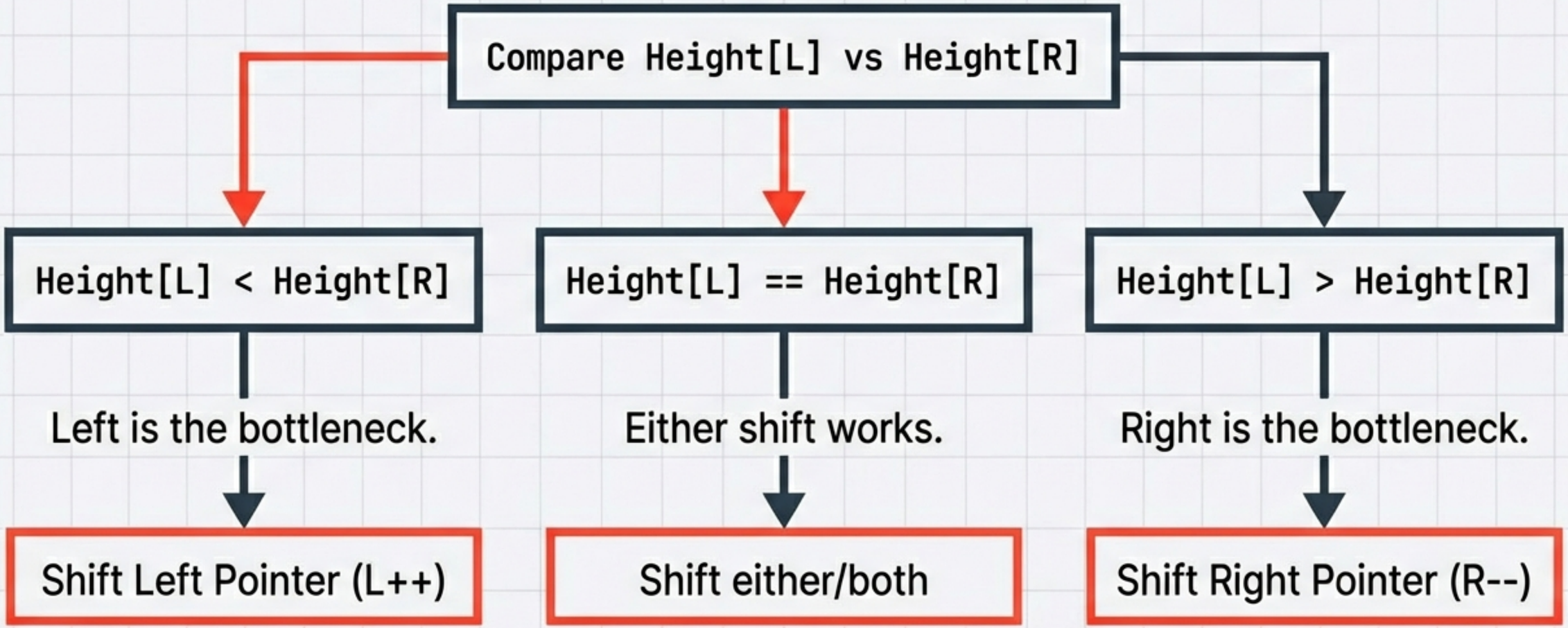
The area is strictly limited by the shorter line (1).

If we keep the Left bar and move the Right bar inward, the width decreases. Since height is capped at 1, the area can ONLY get smaller.

Therefore, keeping the Left bar is useless.

THE DECISION RULE

Swiss Technical Architectural | Detailed Container Analysis



SIMULATION STEP 1

Swiss Technical Architectural | Detailed Container Analysis

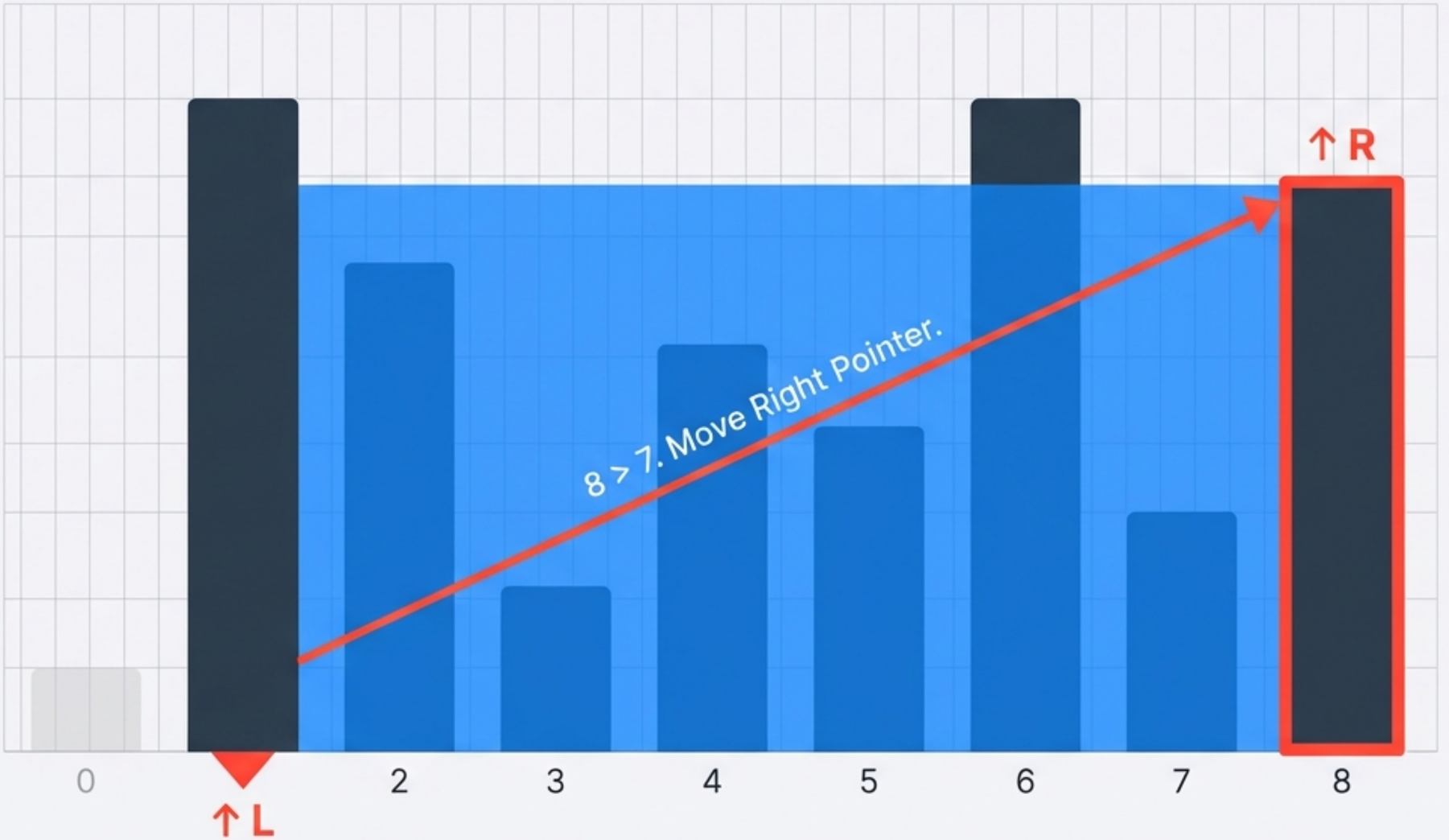
JetBrains Mono

```
Current L: 0 (Ht 1)
Current R: 8 (Ht 7)

Width:     8
Area:      8
MAX AREA:  8
```

SIMULATION STEP 2

Swiss Technical Architectural | Detailed Container Analysis

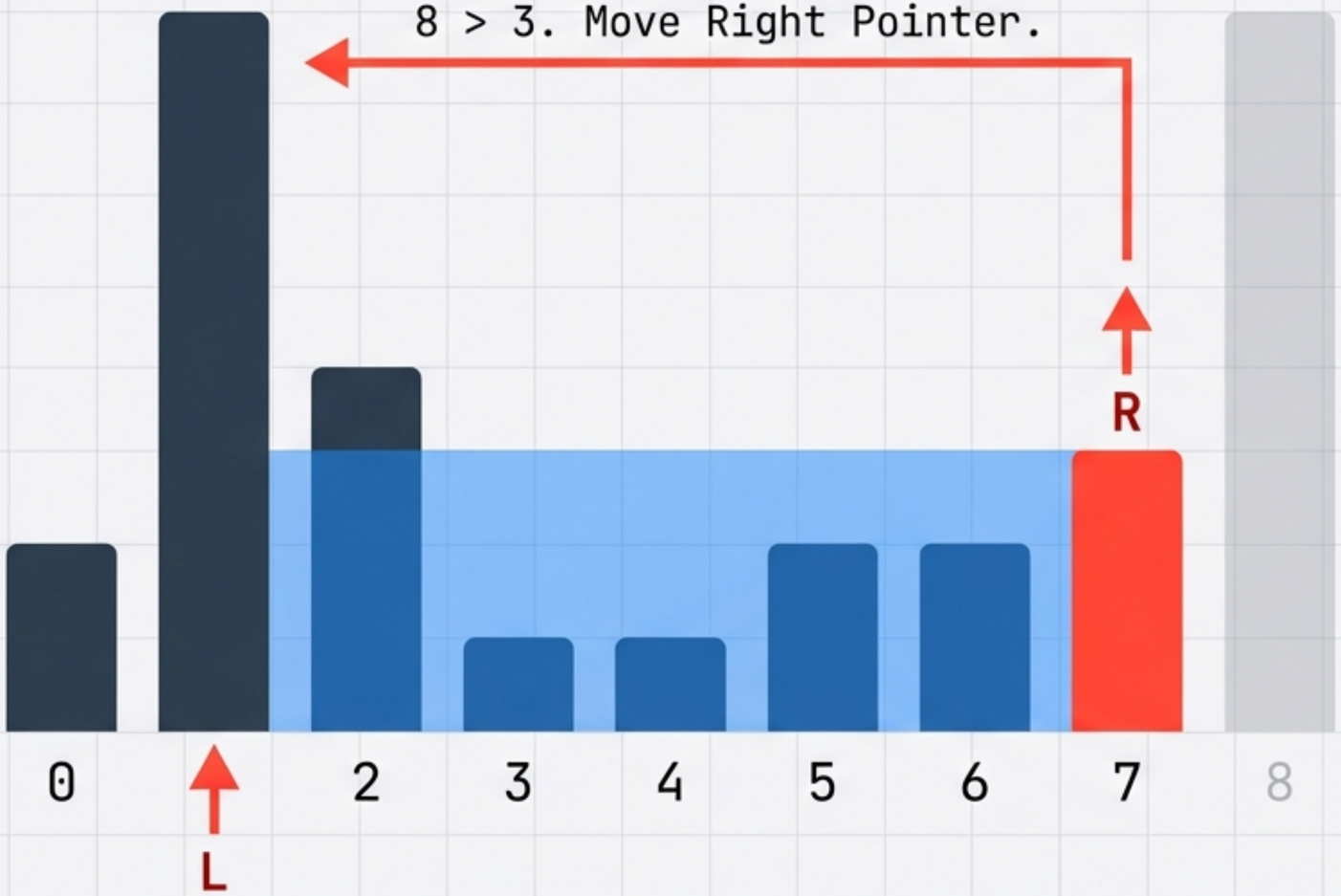


Dashboard Panel JetBrains Mono

Current L: 1 (Ht 8)
Current R: 8 (Ht 7)
Width: 7
Area: 49
MAX AREA: 49 New Record

SIMULATION STEP 3

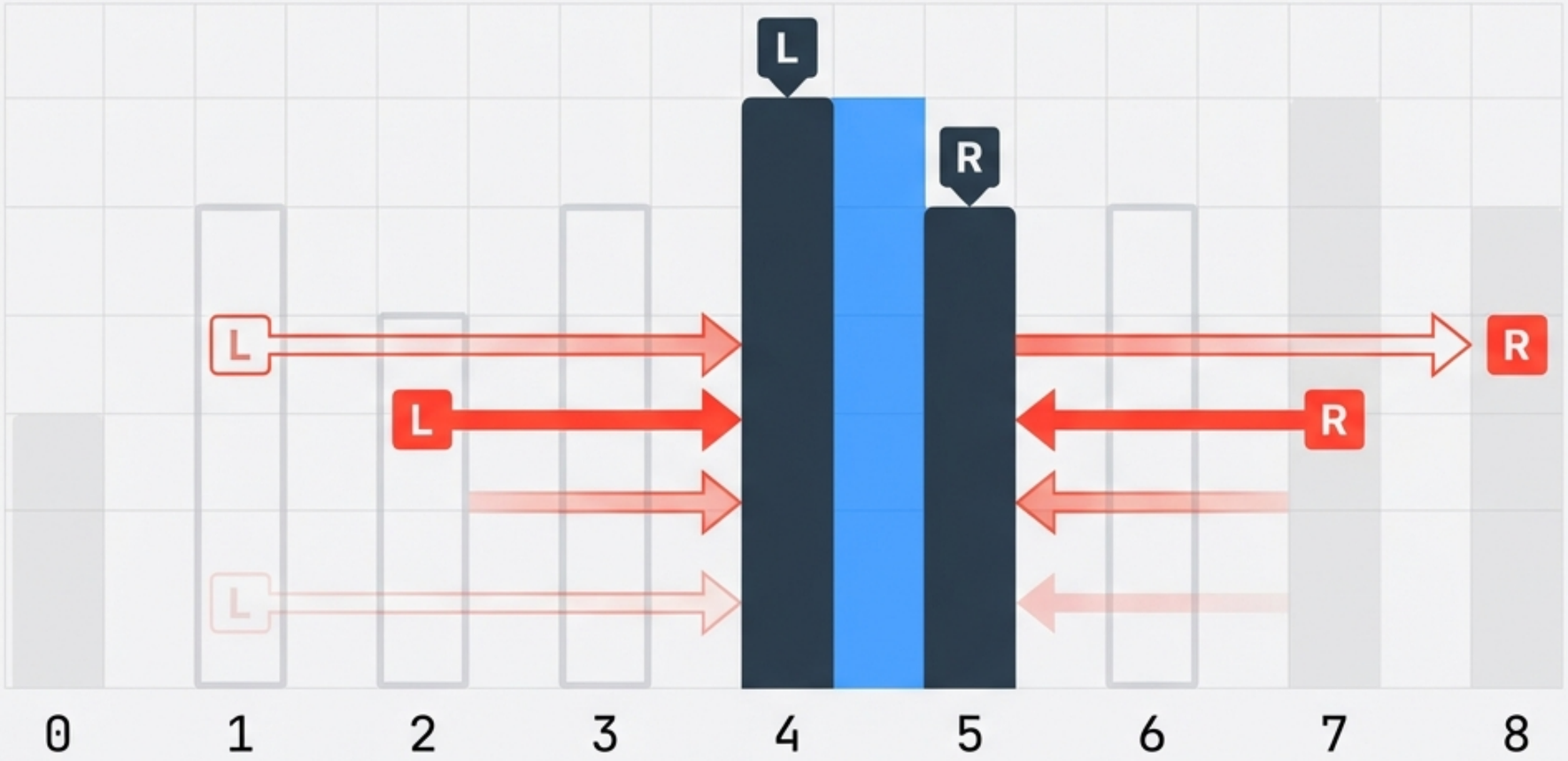
Swiss Technical Architectural | Detailed Container Analysis



Dashboard Panel
Current L: 1 (Ht 8)
Current R: 7 (Ht 3)
Width: 6
Area: 18
MAX AREA: 49

THE SHRINKING WINDOW

We continue discarding the shorter side until the pointers meet.

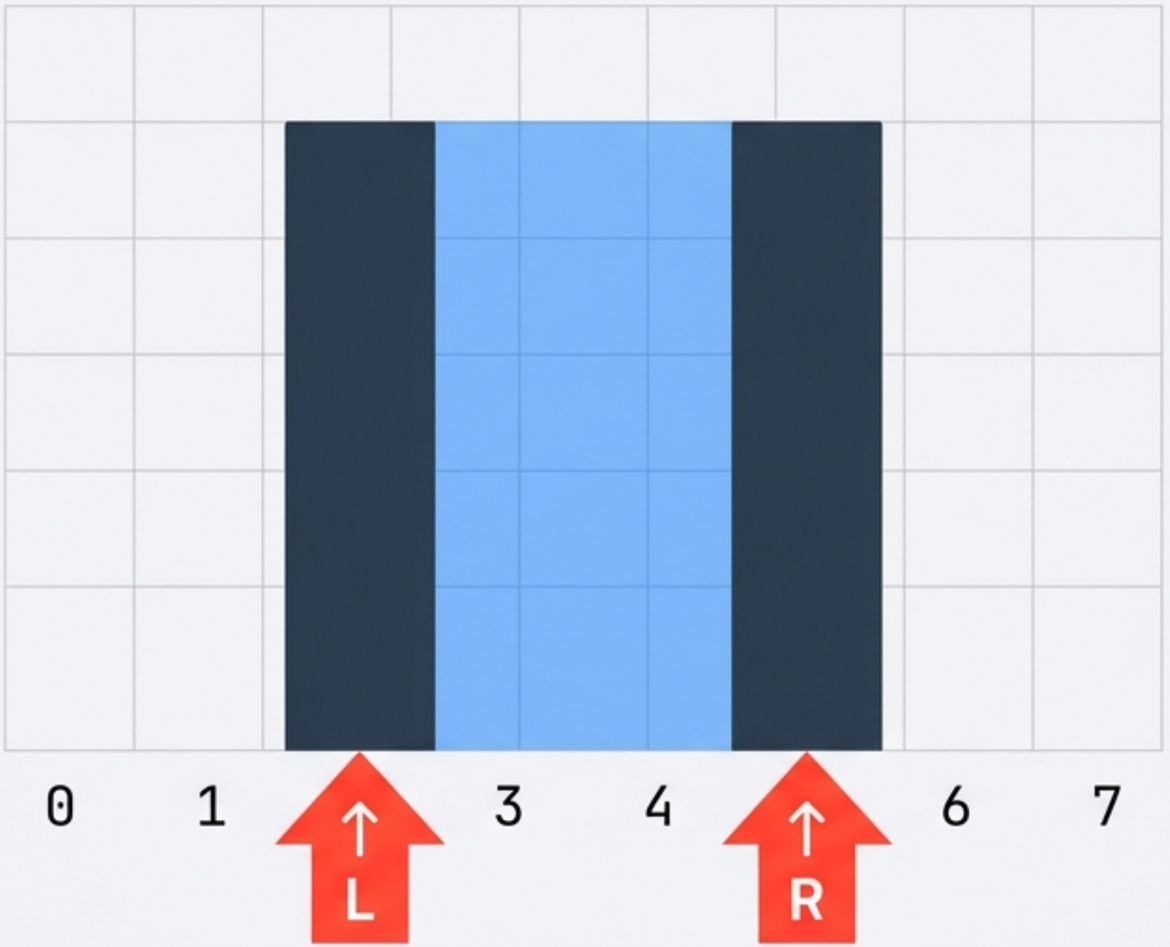


Dashboard Panel
JetBrains Mono

```
Current L: 4 (Ht 5)
Current R: 5 (Ht 4)
Width:     1
Area:      5
MAX AREA:  49
```

EDGE CASE: EQUAL HEIGHTS

Swiss Technical Architectural | Detailed Container Analysis

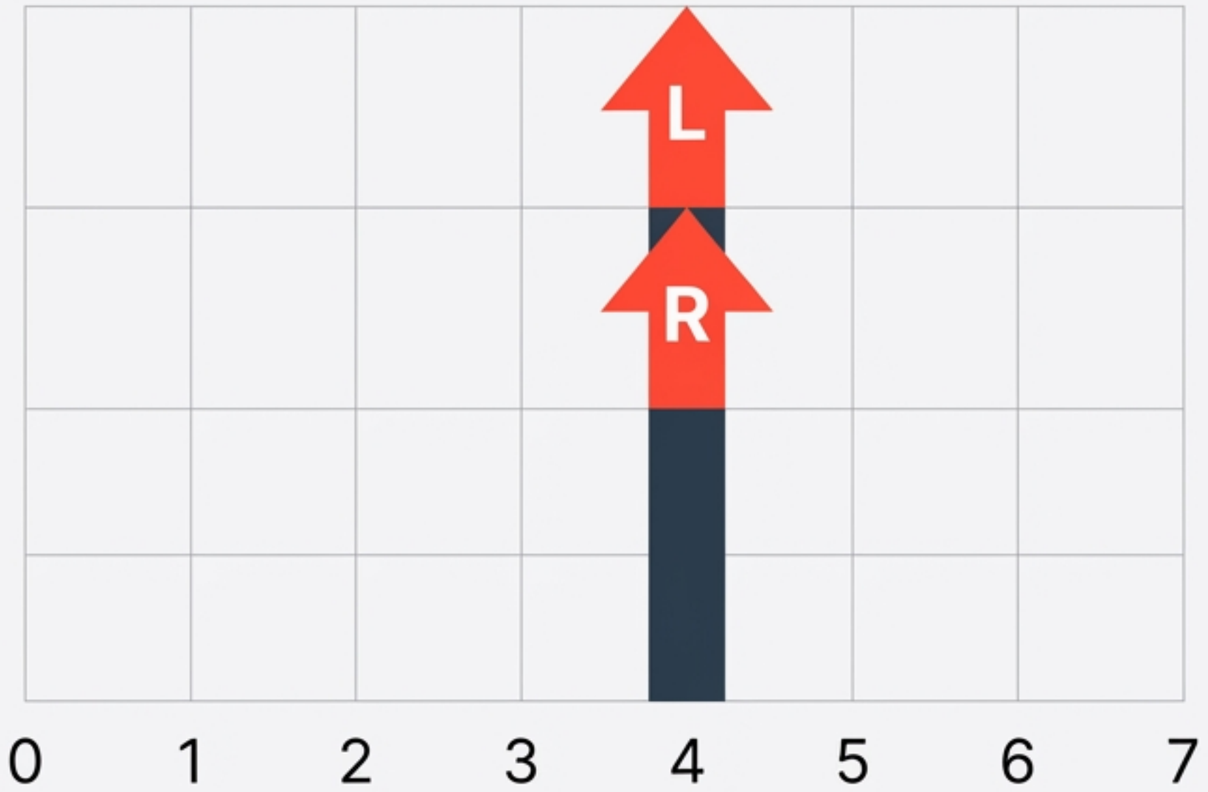


What if Height[L] == Height[R]?

- **Logic:** It implies neither is strictly "better" to keep, or both are limiting.
- **Optimization:** The code handles this naturally by treating it as a "less than" or "greater than" case. We can shift either pointer.

Termination Condition

Stop when Left >= Right



Reason: A container cannot exist with width 0.

Result: Return Max Area (49).

The Solution (Python)

```python
def maxArea(height):
    res = 0
    l, r = 0, len(height) - 1

    while l < r:
        area = (r - l) * min(height[l], height[r])
        res = max(res, area)

        if height[l] < height[r]:
            l += 1
        else:
            r -= 1

    return res
```

Complexity Analysis

Time

O(n) - Linear Time

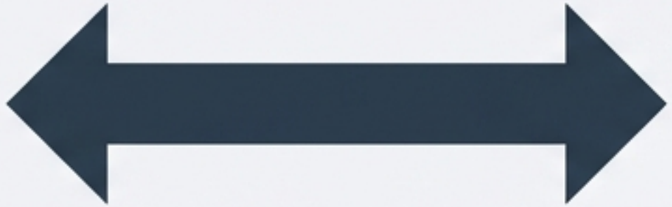
We visit each element at most once. The pointers only move inward, never backtracking.

Space

O(1) - Constant Space

We only store two pointers (`l`, `r`) and a result variable. No extra arrays or hash maps.

Summary: The Pattern



1. Maximize Width

Start at both ends.

2. Identify Bottleneck

Find the shorter height.

3. Discard & Shrink

Move past the bottleneck.

We safely discard the bottleneck because it can never support a larger area as the width shrinks.

Mastering Two Pointers turns $O(n^2)$ bottlenecks into $O(n)$ solutions.

LEETCODE 11 SOLVED