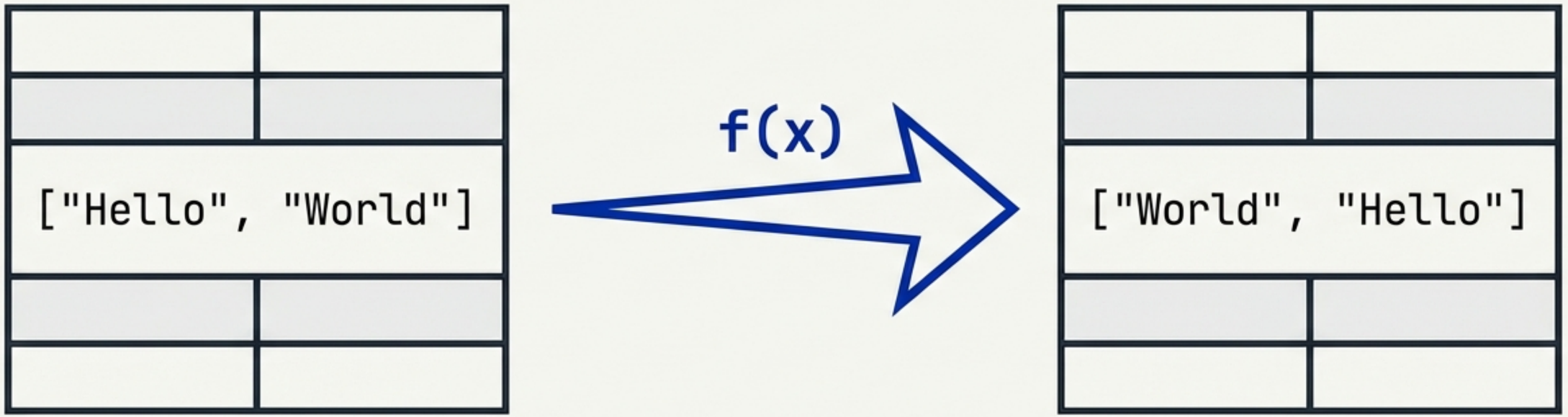
DAY 6/90: STRING ALGORITHMS

Cracking LeetCode 151: Reverse Words in a String

Mastering String Manipulation & The 'Double Reverse' Pattern

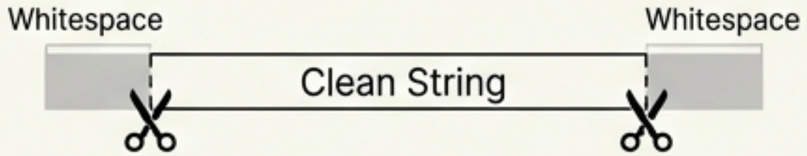


The Problem Statement

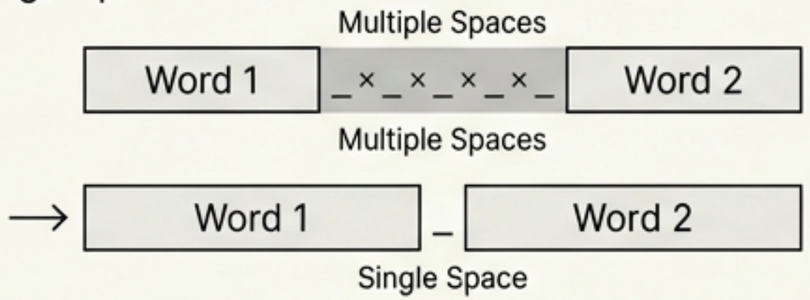
Core Task: Given an input string `s`, reverse the order of the words.

The Catch (Constraints):

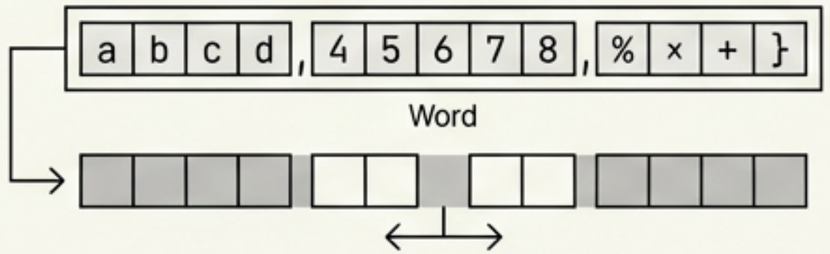
1. **Trim:** Remove all leading and trailing whitespace.



2. **Reduce:** Collapse multiple spaces between words into a single space.



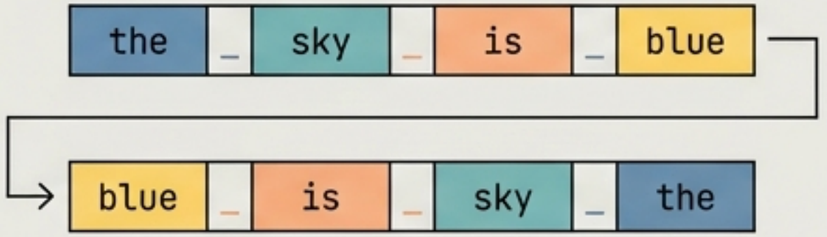
3. **Definition:** A 'word' is a sequence of non-space characters.



Input / Output Examples

Example 1

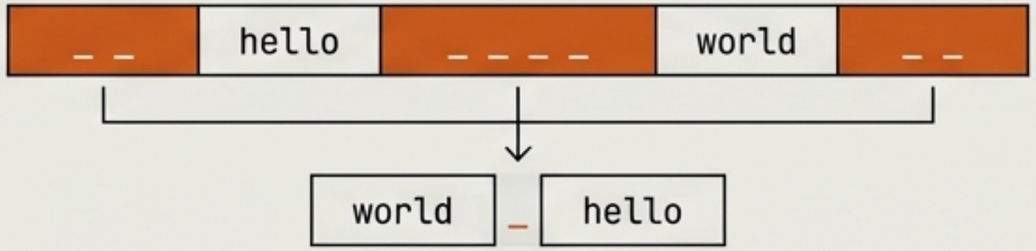
```
Input: 'the_sky_is_blue'
```




```
Output: 'blue_is_sky_the'
```

Example 2 (The Edge Case)

```
Input:  '__hello___world__'
```

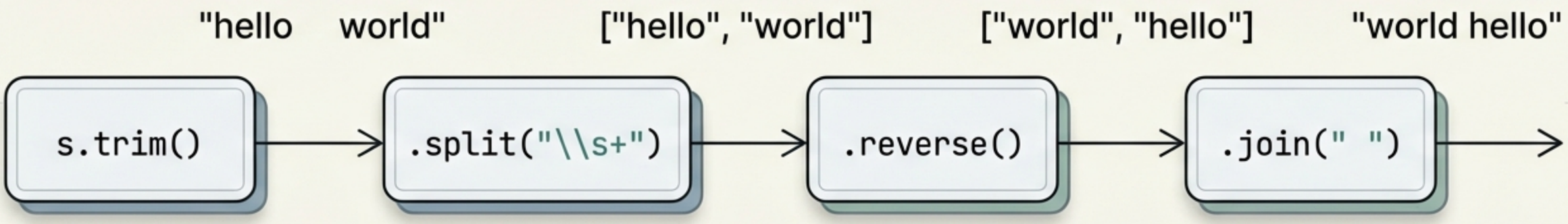



```
Output: 'world_hello'
```

Key Insight: The challenge is not just reversing; it is data sanitation.

Approach 1: The Library "One-Liner"

Production-ready, but interview-risky.




```
return s.trim().split("\\s+").reverse().join(" ");
```

Why the 'One-Liner' Fails the Interview

⚠ The Critique

Space Overhead: `split()` creates a completely new array of strings. `trim()` creates a string copy. This allocates O(N) auxiliary space multiple times.

Dependency: Relies on heavy Regex engines.

The Interviewer's Check

> "That demonstrates library knowledge, not algorithmic thinking. Solve this treating the string as a mutable character array. No `split`, no `trim`."

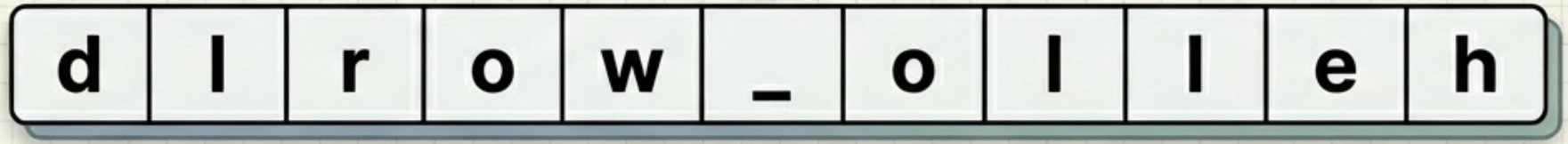
The Goal: Manually manage pointers to clean, reverse, and optimize.

Approach 2: The ‘Double Reverse’ Strategy

A geometric pattern to reverse order without auxiliary arrays.

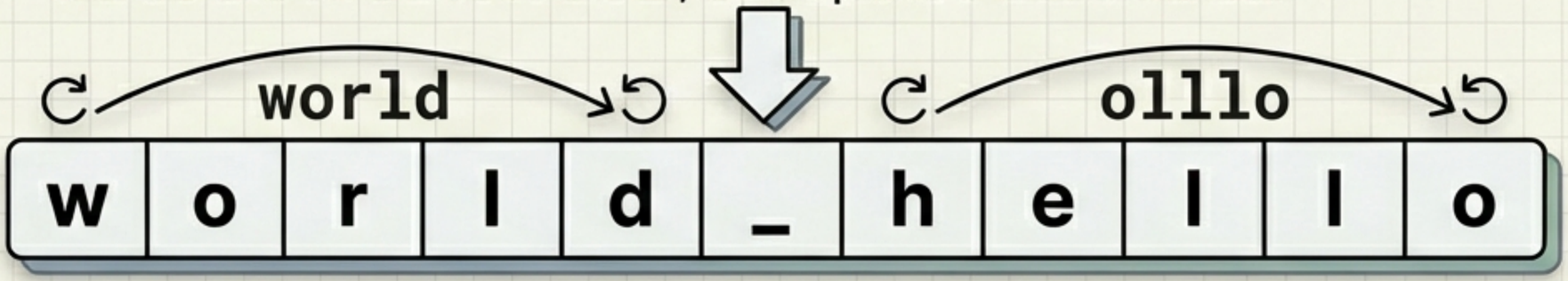
Sanitize Spaces

Reverse Entire String (0 to N-1)



Words are in correct slots, but spelled backwards.

Reverse Individual Words



Step 1: Cleaning the Data (The Logic)

Goal: Convert `'__hello___world__'` → `'hello world'`

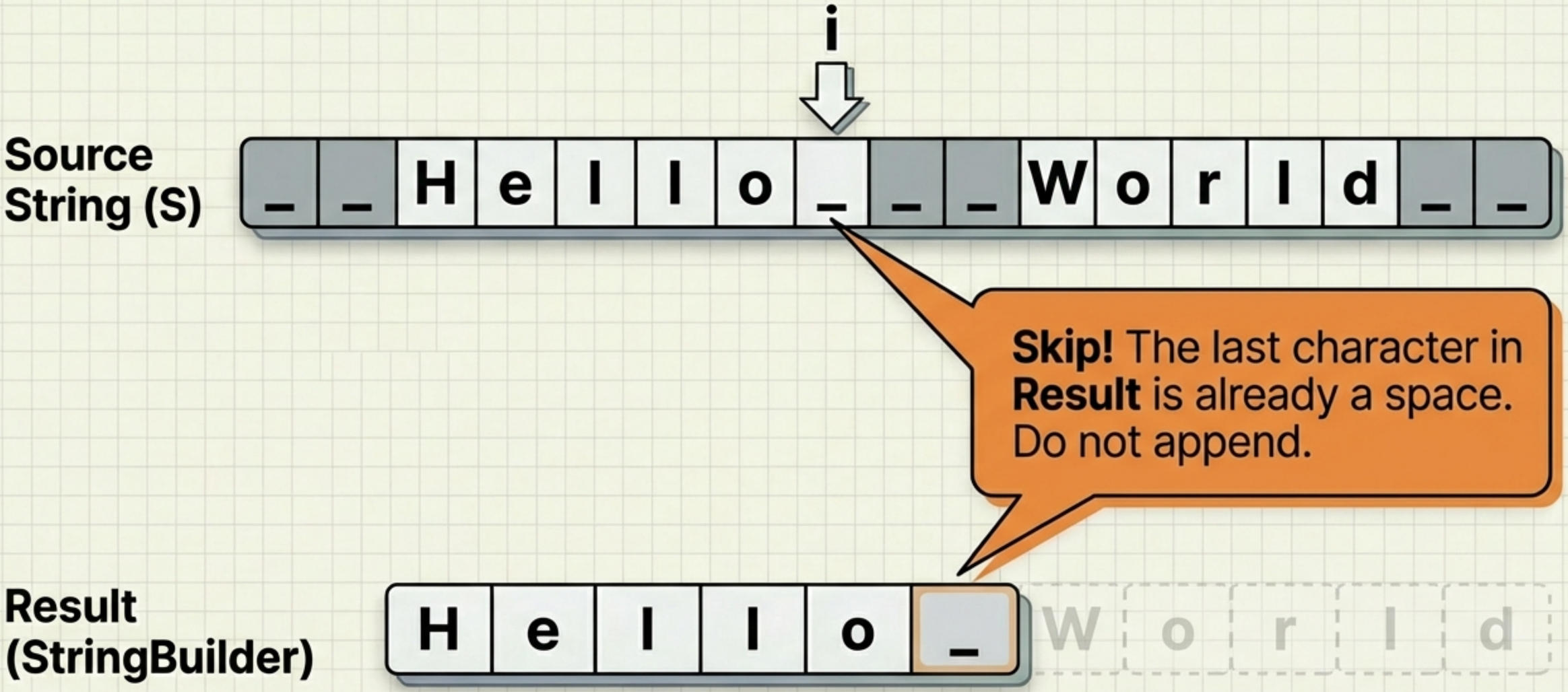
Iterate through source string with pointer L.

- **Rule 1:** If character is not a space: **Append**.
- **Rule 2:** If character IS a space: Only append if the *previous* character in result was **not** a space.

```java
// Java (using StringBuilder)
if (c != ' ') {
    sb.append(c);
} else if (sb.length() > 0 &&
  sb.charAt(sb.length() - 1) != ' ') {
    sb.append(' ');
}
```

This logic automatically handles leading, trailing, and multiple spaces in one pass.

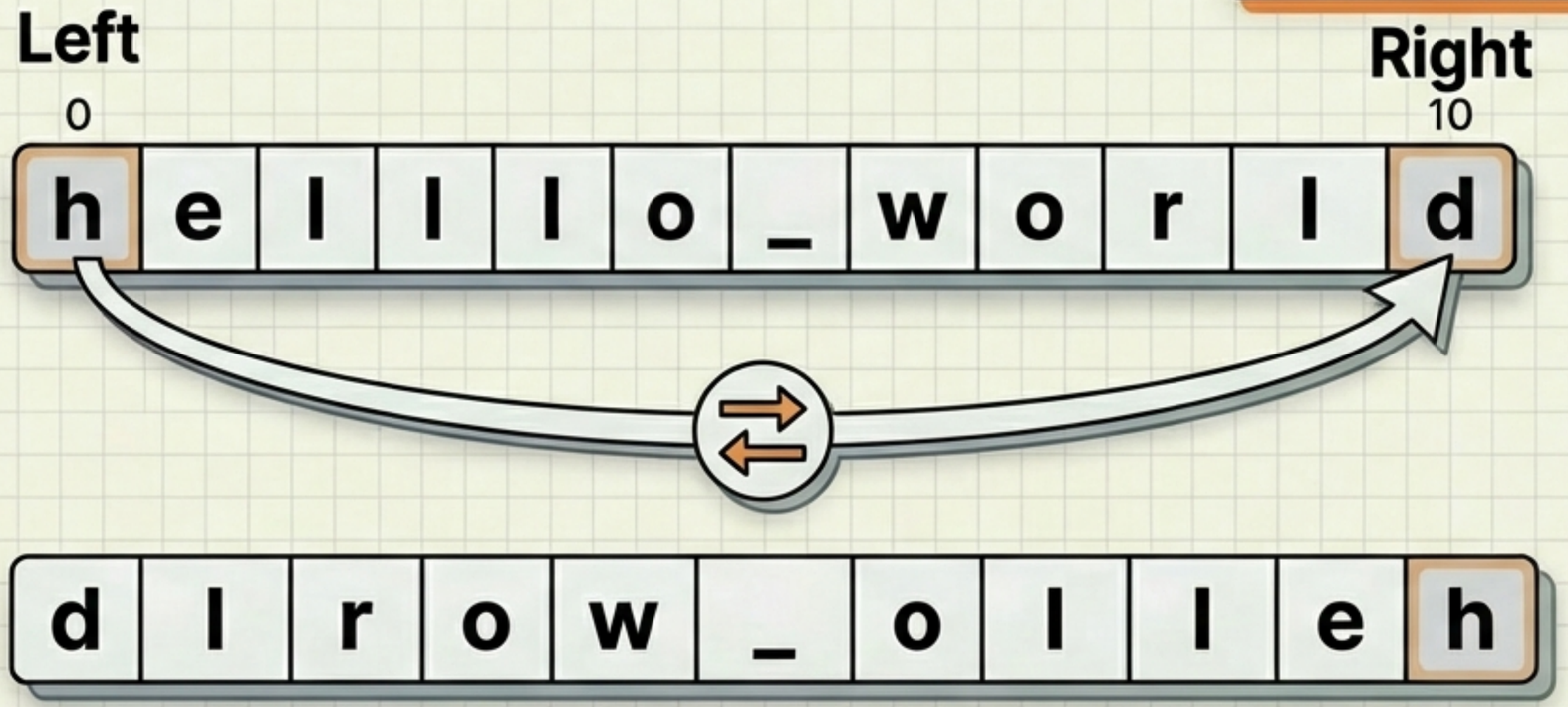
Visualizing the ‘Clean’ Phase



Step 2: Reverse the Whole String

Visualizing the global reverse.

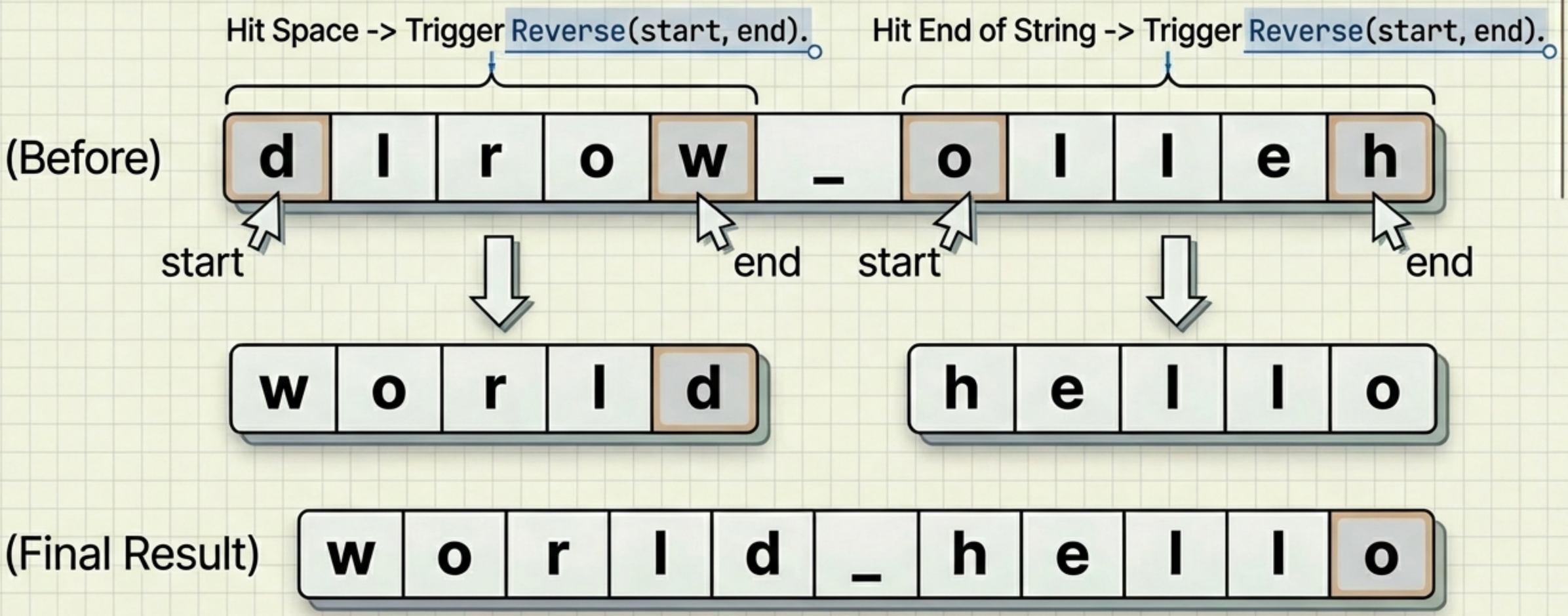
```
reverse(sb, 0, length - 1);
```



Relative Position Fixed: ‘World’ characters are now at the front. ‘Hello’ characters are at the back.

Step 3: Reverse Each Word

Visualizing the word-by-word reverse.



Implementation Structure (Java)

```java
public String reverseWords(String s) {
    // 1. Convert to mutable structure & Clean
    StringBuilder sb = trimSpaces(s);

    // 2. Reverse the whole string
    reverse(sb, 0, sb.length() - 1);

    // 3. Reverse each word
    reverseEachWord(sb);

    return sb.toString();
}
```

Helper Function Logic

- `reverse(sb, i, j)`

  Standard two-pointer swap. Used for both whole string and individual words.

- `trimSpaces(s)`

  Handles the constraints (leading/trailing/multiple spaces).

Complexity Analysis

Time Complexity

O(N)

Pass 1 (Clean): Traverse N characters.
Pass 2 (Global Reverse): Traverse N/2 times.
Pass 3 (Word Reverse): Traverse N times total.

3N ≈ O(N).

Space Complexity

Java

O(N)

Strings are immutable. Must allocate `StringBuilder`.

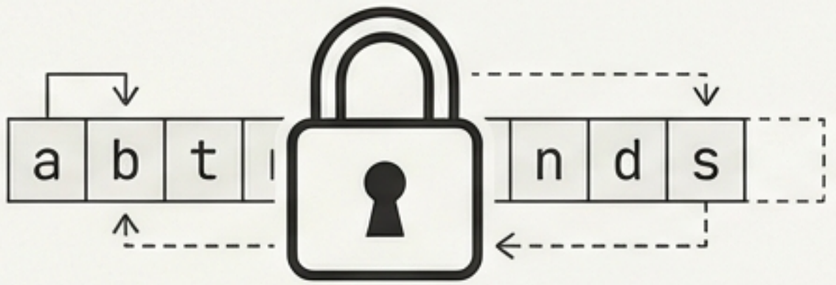
C++

O(1)

Strings are mutable. Algorithm can be performed in-place.

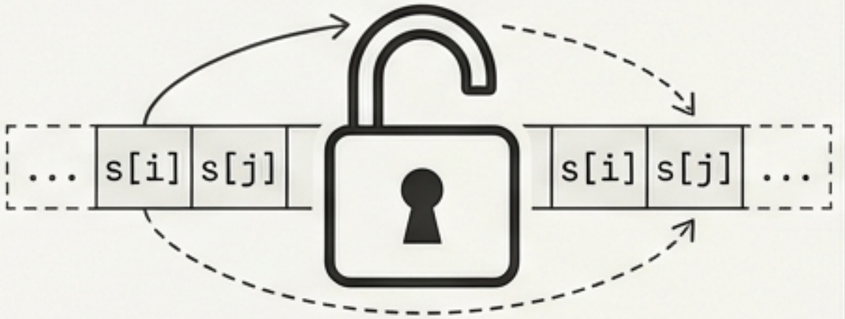
The “Immutability” Caveat

String s



Immutable. Cannot change characters in place.

StringBuilder



Mutable. Supports in-place swaps.

Interview Pro-Tip

Explicitly mention to your interviewer: “I am using O(N) space for the StringBuilder because Java strings are immutable, but the logic itself is O(1) space.”

The “Double Reverse” Pattern Summary

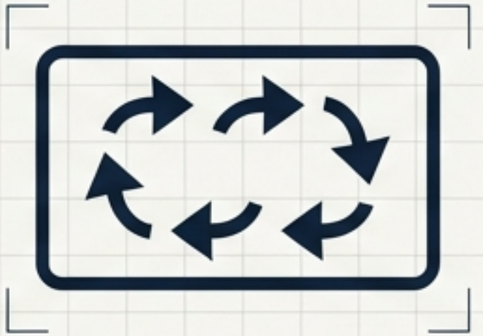
1. Sanitize

Merge & trim spaces

2. Global Flip

Fix relative word positions

3. Local Flip



Fix internal word spelling

Versatility: This pattern also solves **‘Rotate Array’** (**LeetCode 189**).

Homework & Next Steps

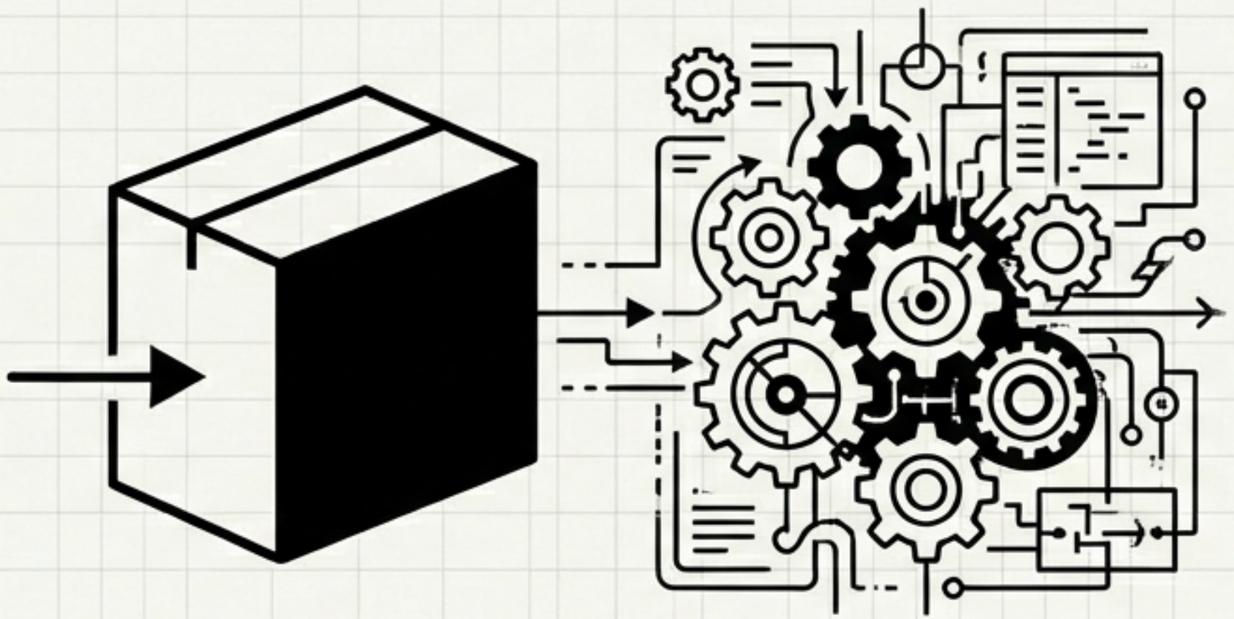
- **Practice:** Reverse Words in a String II

  Input is a mutable `Char Array`. Use this exact logic for `O(1)` space.

- **Practice:** Reverse Words in a String III

  Reverse characters in words but preserve whitespace/order.

- **Implement:** `cleanSpaces` function from scratch.



Libraries Algorithms

“Libraries are great for production, but Algorithms are for understanding.”

Mastering pointers and state manipulation is more valuable than knowing the `.split()` function.

Based on 'Day 6/90: String Algorithms' by CTO Bhaiva