perplexity

if we changea a primitive data type, it does not override but changes its location in memory, doesnt this lead to memory wastage, what happens in other lang, explain in detail with let c = 10 example and then c= 50

When you **change the value of a primitive data type** (like `let c = 10;` followed by `c = 50;`), what happens in memory does **not** usually cause memory wastage, and the process differs based on the language and whether it's "primitive type" or "reference type."

What happens in a primitive assignment? (`let c = 10; c = 50;`)

- When you declare `let c = 10;`, the system creates a spot in memory for a variable called `c` and stores the value `10` there.
- If you later write `c = 50;`, the system **overwrites** the existing memory location with the new value `50`. The old value (`10`) is simply replaced; it doesn't reserve a new spot or leave the old one hanging around.
- This is the default behavior in most languages like C, C++, Java (for primitive types), and JavaScript[1] [2] [3].

In C/C++:

```
int c = 10; // Allocates 4 bytes (typically) on the stack, holds value 10
c = 50;     // Same 4 bytes: 10 is simply replaced by 50
```

> No new memory is allocated. There's no wastage; only the value at the address for `c` changes[4] [3] [5].

In Java:

```
int c = 10; // Reserves space on the stack for 'c', stores 10
c = 50;     // Overwrites 10 with 50 in the same location
```

> Primitives (`int`, `float`, etc.) are stored on the stack. Reassignment only updates the value, not the location[6] [2] [3].

In JavaScript:

- Very similar: the variable holds the value directly.
- When you assign `c = 50;`, the old value (`10`) is gone, replaced by `50` at the same memory spot[7] [8].

Does this create memory wastage?

- **No**, because the system just overwrites the value at the assigned memory location.
- The only exception could be certain languages or environments with "immutable" primitives or special string interning, but for typical `int`, `float`, `char` types, overwriting is standard. Non-primitive/reference types (like objects/arrays) behave differently.

How do reference types behave?

- *Reference (non-primitive) types* (objects, arrays, strings in some languages) store a memory address (pointer) to the actual data on the heap.
- If you assign a new value to a reference variable, the pointer is updated—so the variable now points somewhere else. The old data in memory is left alone until garbage collected (automatic memory management) or freed (manual).
- In languages with **garbage collection** (like Java, JavaScript, Python), old values may stay in memory until the garbage collector frees them—but that's for *objects*, not primitives[9].

Example side-by-side: Primitives vs Reference Types

Example	Memory Behavior
`let c = 10; c = 50;`	Overwrites 10 with 50 in same variable's memory slot. No new allocation. No wastage[2] [3].
`let arr = [^1][^10][^11]; arr = [^12][^6];`	Heap allocates new memory for `[^12][^6]`, pointer `arr` is updated. `[^1][^10][^11]`'s space remains until garbage collected (in JS, Java)[6] [9].

Key Points

- **Primitives:** Value replaced in the same spot (no memory wastage, no new allocation).
- **Reference types:** New location allocated for new data; old data is garbage collected. Some temporary "wastage" may occur until GC runs.

Summary: Your Example (`let c = 10; c = 50;`)

- **Result:** `c` always occupies just one memory slot—its value changes, but the slot remains the same. No wastage occurs for primitive types in most languages. For reference types, there can be temporary memory not in use until the GC collects it[2] [3] [7] [8] [9].

If you want a detailed illustration or code in a specific language, let me know!

⁂

1. https://www.devlane.com/blog/what-are-primitive-data-types
2. https://java-programming.mooc.fi/part-5/3-primitive-and-reference-variables/
3. https://www.scientecheasy.com/2020/06/memory-allocation-primitive-nonprimitive.html/
4. https://www.w3schools.com/c/c_memory_allocate.php
5. https://unstop.com/blog/dynamic-memory-allocation-in-cpp
6. https://www.linkedin.com/pulse/understanding-memory-allocation-primitive-vs-data-types-ankit-dubey-azbgc
7. https://ui.dev/primitive-vs-reference-values-in-javascript
8. https://stackoverflow.com/questions/69519676/if-i-change-a-variables-value-will-the-original-value-still-exist-inside-the-me
9. https://muneebdev.com/understanding-data-types/