Garbage Collection

COMP360

Memory Management

- Program Objects/Data occupy memory
- How does the runtime system *efficiently* create and recycle memory on behalf of the program?
  - What makes this problem important?
  - What makes this problem hard?
  - Why are researchers still working on it?

Dynamic memory allocation and reclamation

- *Heap* contains dynamically allocated objects
- Object allocation: **malloc**, **new**
- Deallocation:
  - Manual/explicit: **free, delete**
  - automatic: **garbage collection**

Explicit Memory Management Challenges

- More code to maintain
- Correctness
  - Free an object too soon - Error
  - Free an object too late - waste space
  - Never free - at best waste, at worst fail
- Efficiency can be very high
- Gives programmers “control”

Garbage collection:
Automatic memory management

- reduces programmer burden
- eliminates sources of errors
- integral to modern object-oriented languages, i.e., Java, C#, .net
- now part of mainstream computing
- Challenge:
  - performance efficiency

Key Issues

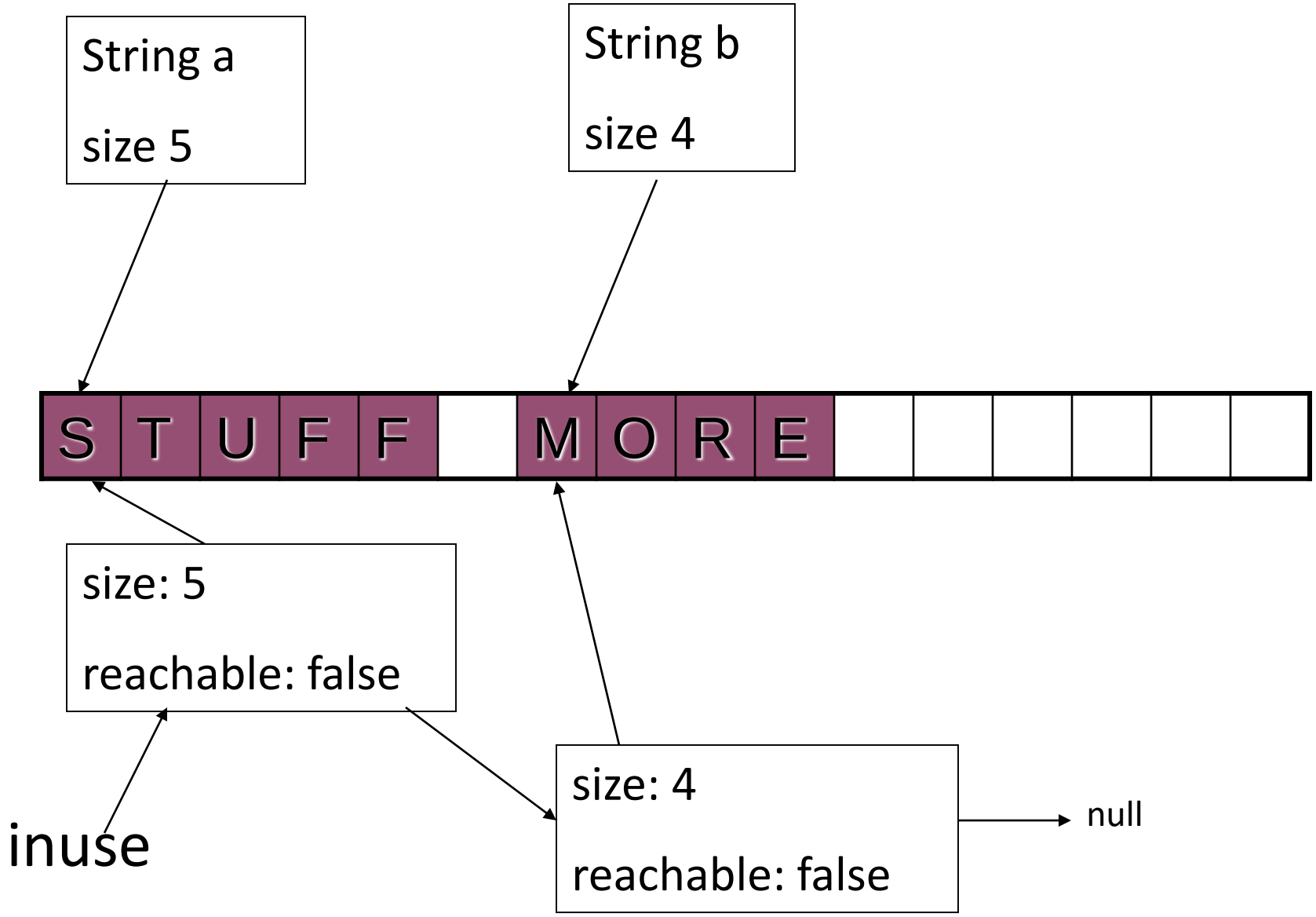
- For both
  - Fast allocation
  - Fast reclamation
  - Low fragmentation (wasted space)
  - How to organize the memory space
- Garbage Collection
  - Discriminating live objects and garbage

Garbage Collection Notes

- Observation
  - Most objects die young
  - A small percentage long lived objects
- Avoid copying long-lived objects several times
- Generational GC segregates objects by age
  - Older object collects less often

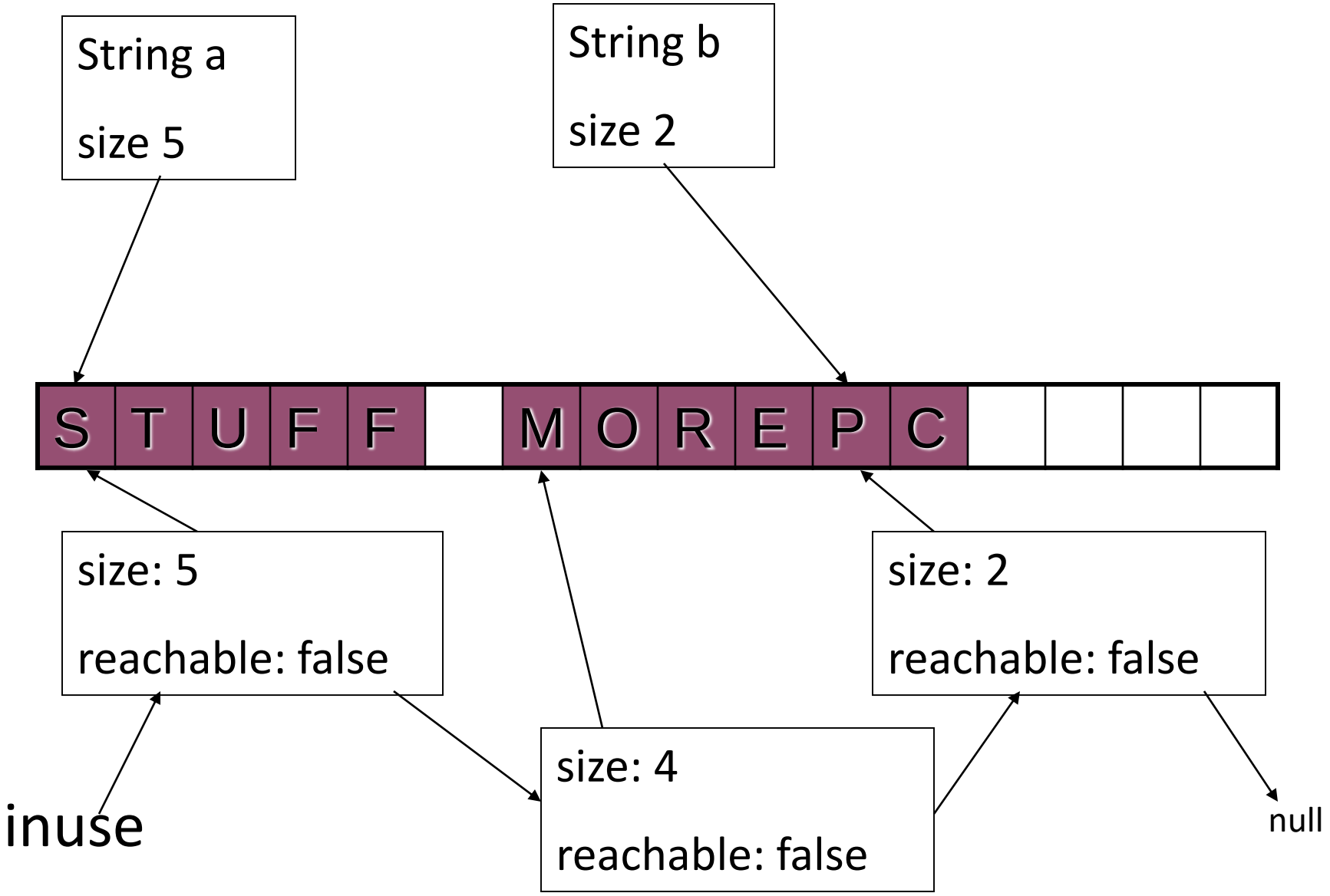
Initial Setup of Example

```
String a;
String b;
a = "STUFF";
b = "MORE";
```


String a
size 5
String b
size 4
S T U F F M O R E
size: 5
reachable: false
inuse
size: 4
reachable: false
null

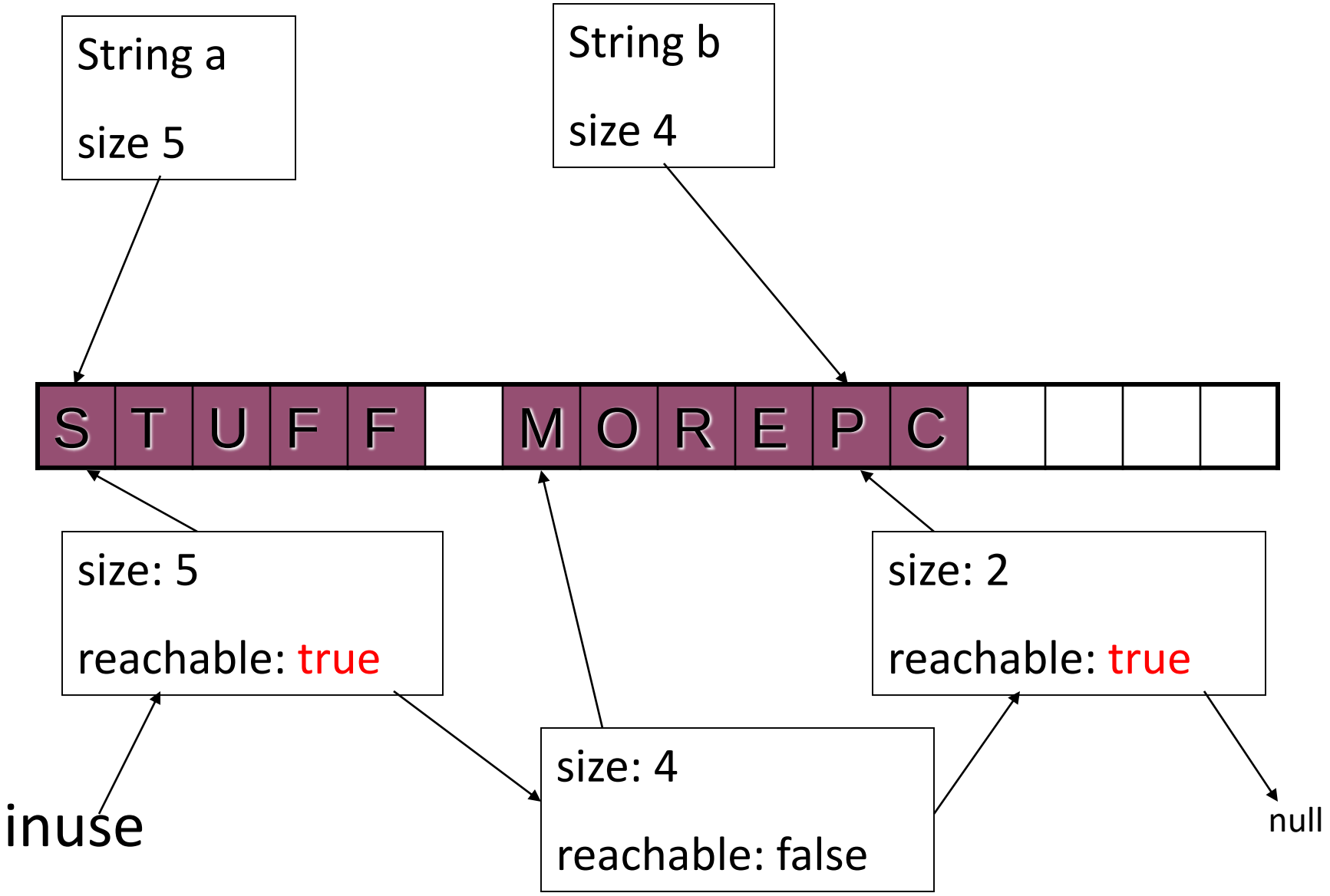
Reassignment of a variable

```
String a;
String b;
a = "STUFF";
b = "MORE";
b = "PC";
```


String a
size 5
String b
size 2
S T U F F M O R E P C
size: 5
reachable: false
size: 2
reachable: false
size: 4
reachable: false
inuse
null

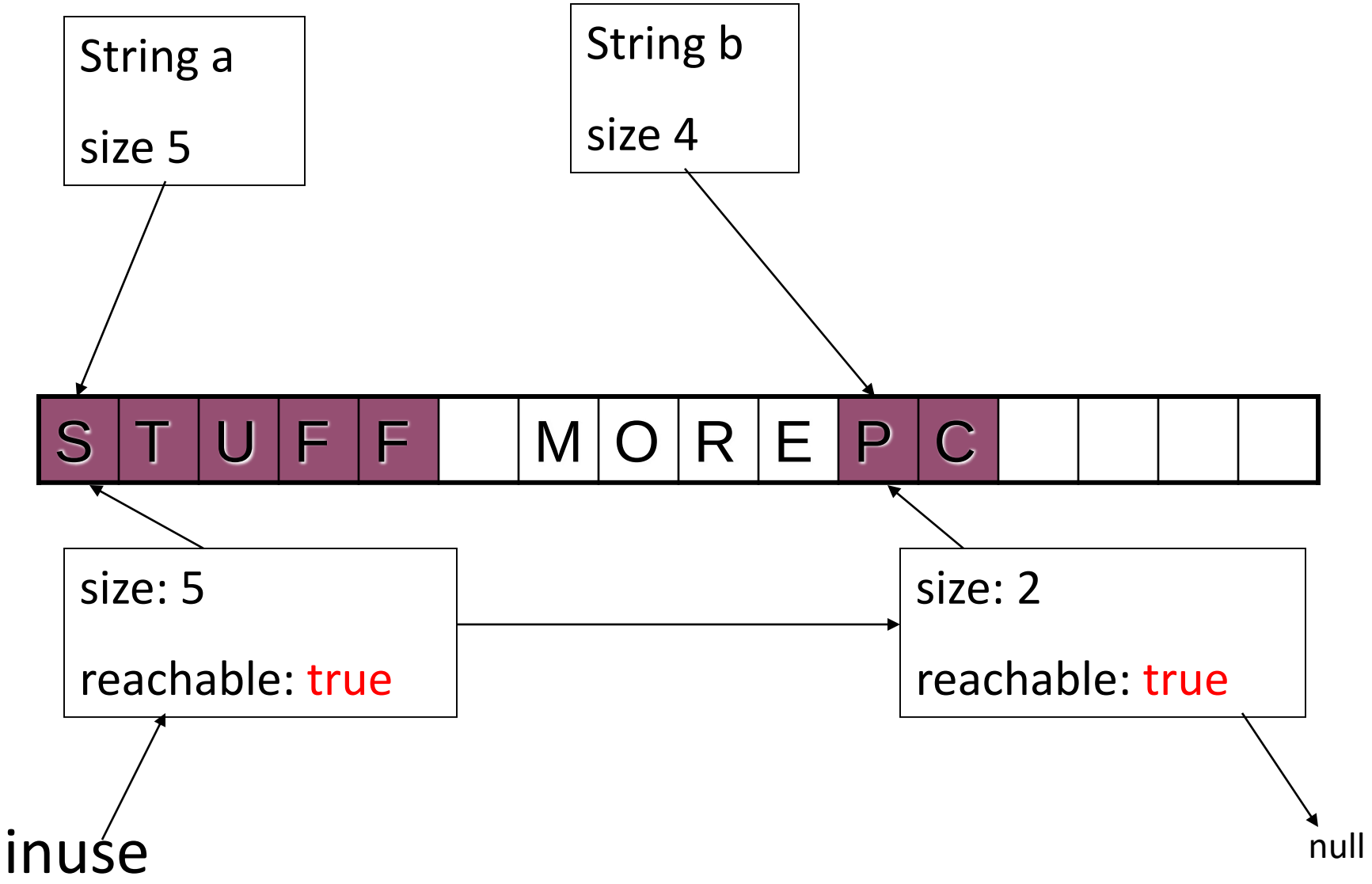
Garbage Collection

- For each string, search the list of inuse space to find allocation node that points to this string.
- Set the **`reachable`** flag to true;


String a
size 5
String b
size 4
S T U F F M O R E P C
size: 5
reachable: true
size: 2
reachable: true
size: 4
reachable: false
inuse
null

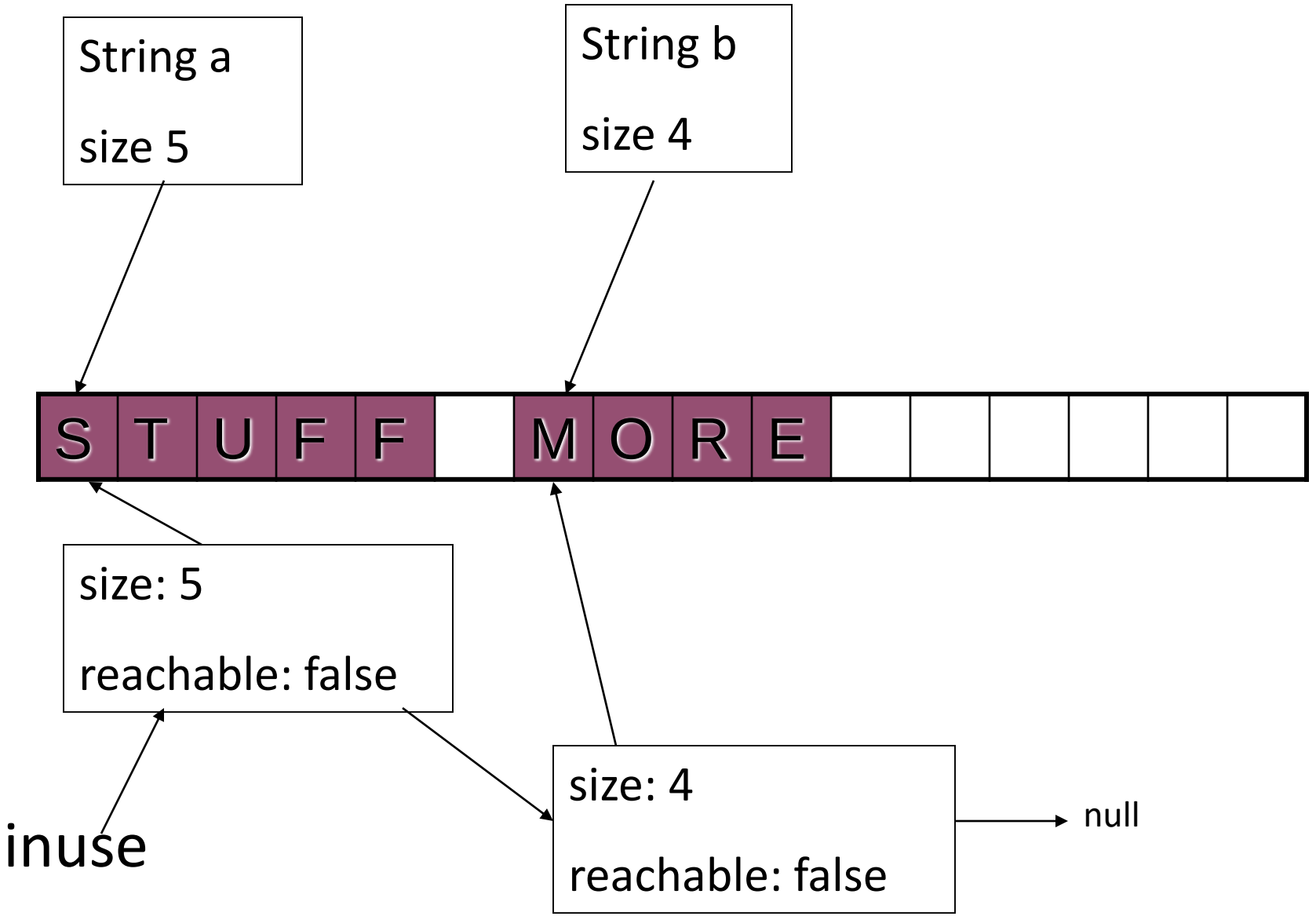
Take out the garbage

- Traverse the inuse list.
- For each allocation node whose **`reachable`** flag is false, return that space to the free pool.


String a
size 5
String b
size 4
S T U F F M O R E P C
size: 5
reachable: true
size: 2
reachable: true
inuse
null

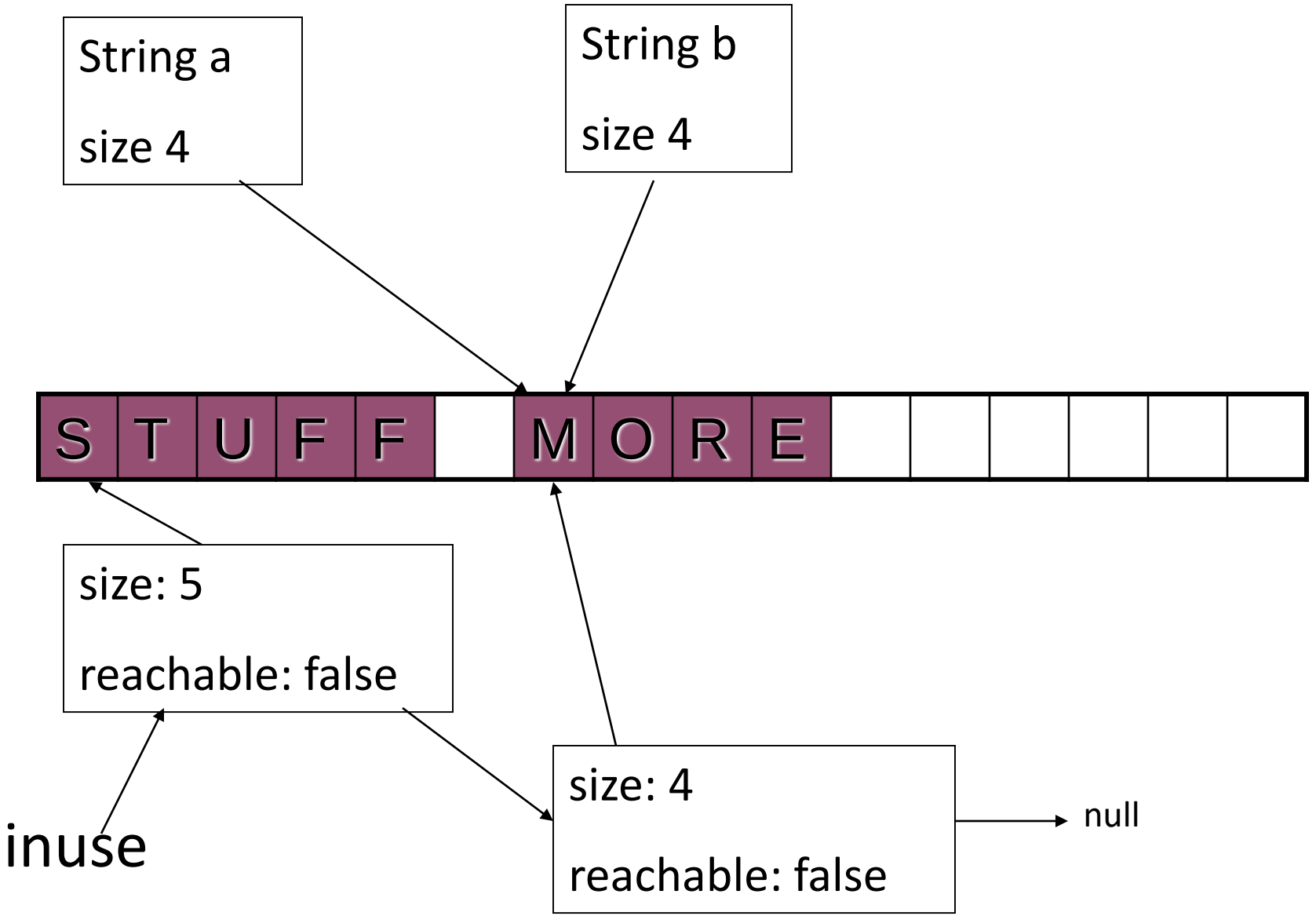
Another Example

```
String a;
String b;
a = "STUFF";
b = "MORE";
```


String a
size 5
String b
size 4
S T U F F M O R E
size: 5
reachable: false
size: 4
reachable: false
null
inuse

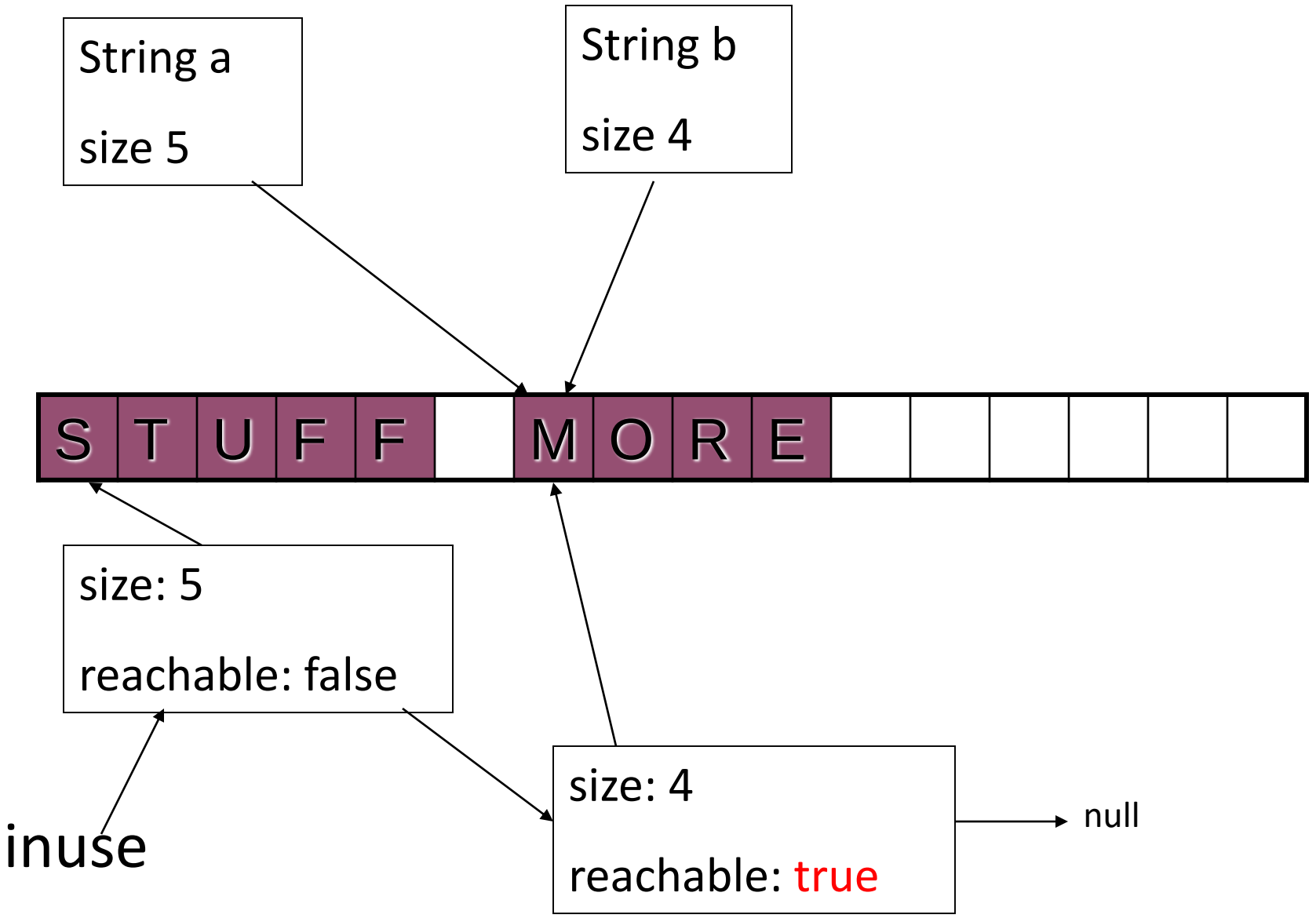
Set one variable equal to another

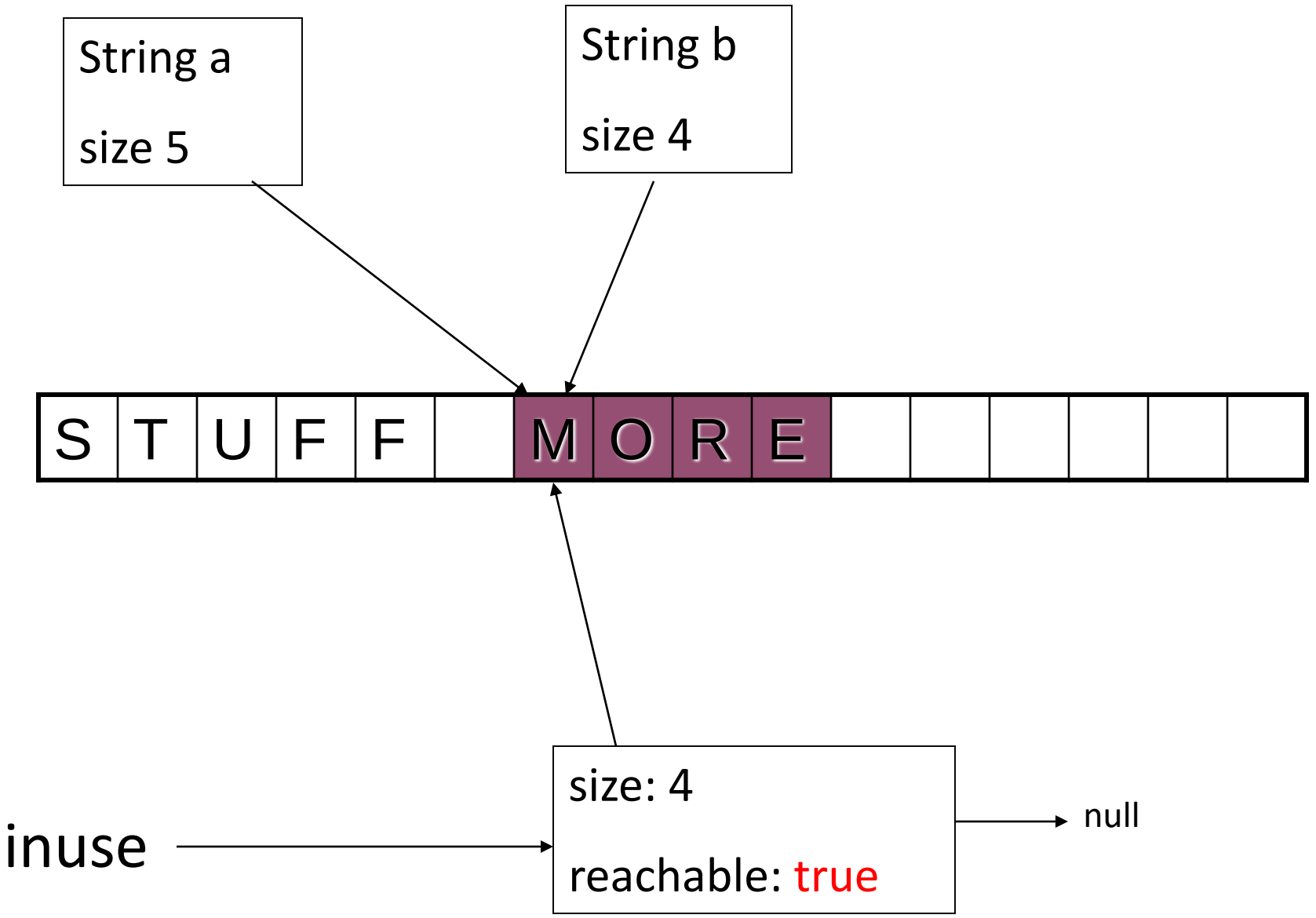
```
String a;
String b;
a = "STUFF";
b = "MORE";

a = b;
```


String a
size 4
String b
size 4
S T U F F M O R E
size: 5
reachable: false
size: 4
reachable: false
null
inuse

Garbage Collect

- Traverse the inuse list.
- For each allocation node whose **`reachable`** flag is false, return that space to the free pool.


String a
size 5
String b
size 4
S T U F F M O R E
size: 5
reachable: false
size: 4
reachable: true
null
inuse


String a
size 5
String b
size 4
S T U F F M O R E
inuse
size: 4
reachable: true
null