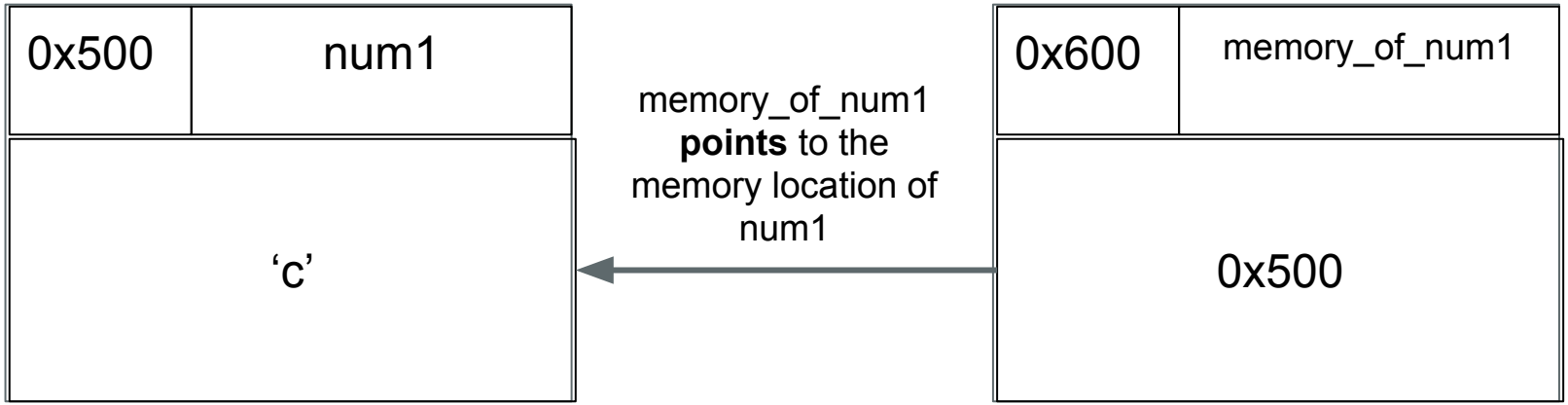
Pointers and Functions

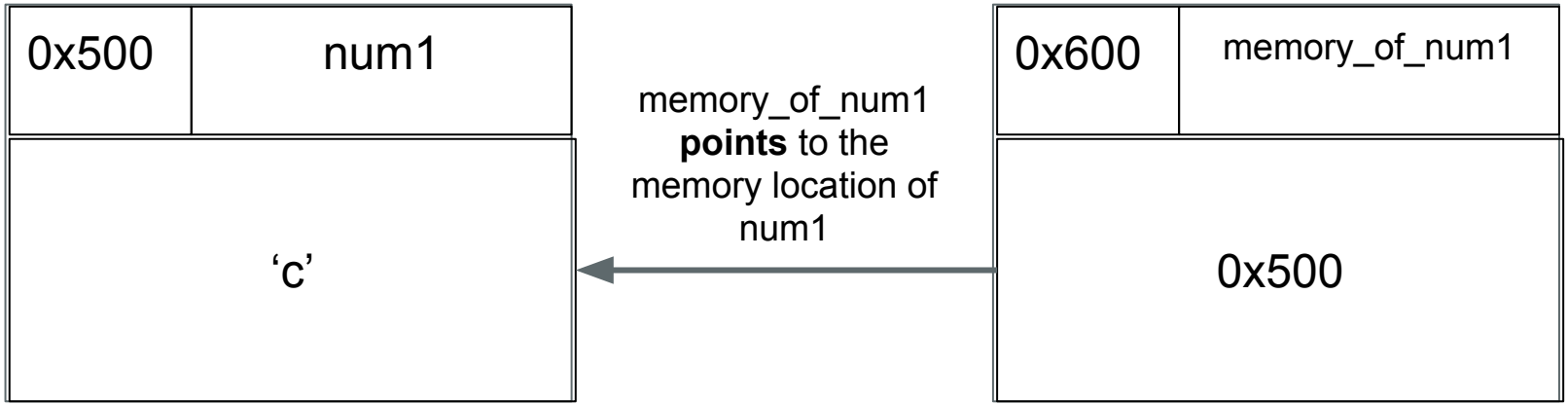
Pointer Review

- Pointers store the memory address of another variable
- Allow direct manipulation of variables in different memory locations



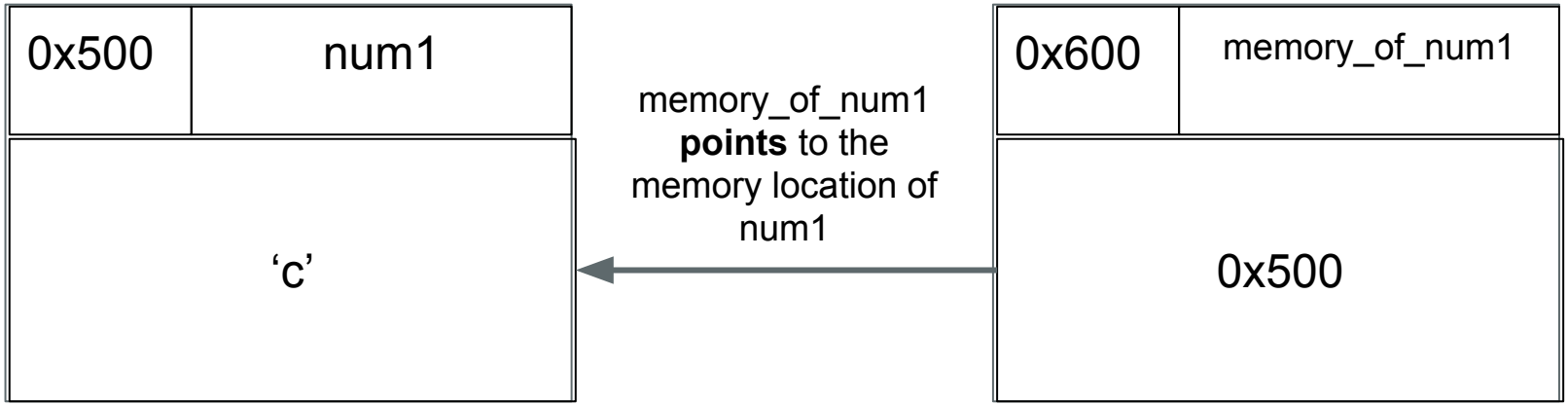
Pointer Review

- The memory address of a variable can be accessed through &
  - Ex: &num1 → 0x500



Pointer Review

- The object pointed to by a variable can be accessed through *
  - Ex: *memory_of_num1 → ‘c’



Pointers in main()

- Pointers found only in main() are allowed
- Can often add confusion to code…

- Pointers are usually found in three places: functions, arrays, and structs

Pointers in Functions

- Remember → C is a **pass-by-value** language
  - All inputs to a function will make a new memory location
  - Requires many copy steps

- C provides a challenge for software engineers
  - Difficult to keep track of consistently returning and assigning variables
  - Also fairly inefficient

Pointers in Functions

- Copying and returning is fairly inefficient → ends up being a 8-step process
  - 1) find original variable
  - 2) access stored value
  - 3) find another memory location
  - 4) store value here

Copying Step

Pointers in Functions

- Copying and returning is fairly inefficient → ends up being a 8-step process
  - 5) find returned variable
  - 6) access returned value
  - 7) find original memory location
  - 8) store value in OG memory

Return / Assignment Step

Pointers in Functions

- Pointers allow direct manipulation of variables within functions
- No need for return statements: removes HALF of that 8-step process
  - Therefore more efficient
  - Also can be easier to visualize for software engineers

Pointers in Functions

- Practically, they work the same way as pointers in main()
  - Memory address accessed by &
  - Contents of referenced memory addressed by *

- Differences
  - The **memory address** is passed into the function
  - The **referenced value** is manipulated within the function

Pointers in Functions

- The function invocation uses the **&** reference in order to pass the memory location
  - The memory location is copied to the function variables
  - The original memory is accessed using *

```
addone(&num1);

void add_one(int *num1){
    *num1 = *num1 + 1;
}
```

Pointers Coding Challenge: Part 1

Die Rolling Game

- If the code rolls doubles, you win. Otherwise, the computer wins.
- Generate two random numbers in main() between 1-6
- Create a function that checks and prints if you have won or not
    - Restriction: This function HAS to use pointers to refer to the two random numbers in main()
- Ask the user if they want to continue playing at the end of each turn
    - If ‘y’, keep going
    - If ‘n’, quit

Pointers Coding Challenge: Part 2

Die Rolling Game - with betting

- Same rules apply, except...
- The user has $100 to start
  - The user must place a bet on the game (this bet must be >= $0)
  - If the user wins, the amount of money gets added to their account
  - Otherwise, it is removed
- This money exchange must be completed in a function with pointers
- If the user reaches $0, the game stops