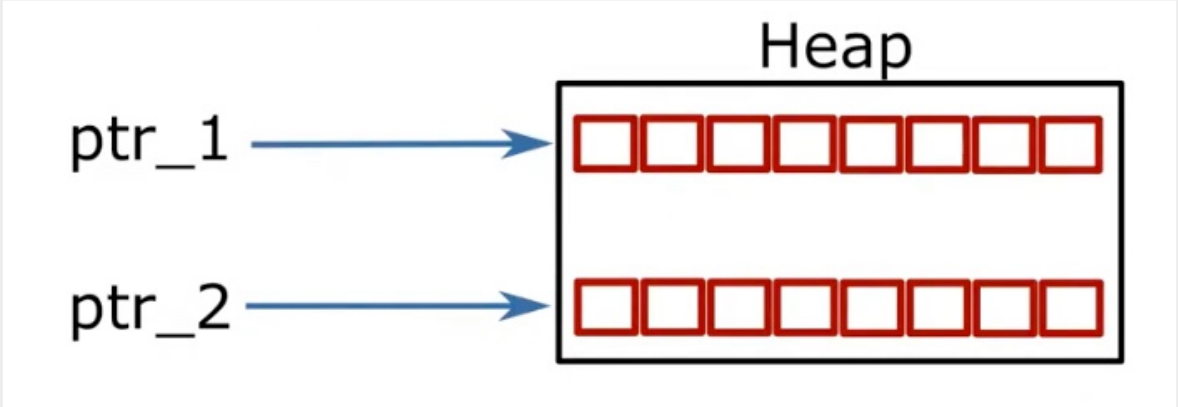
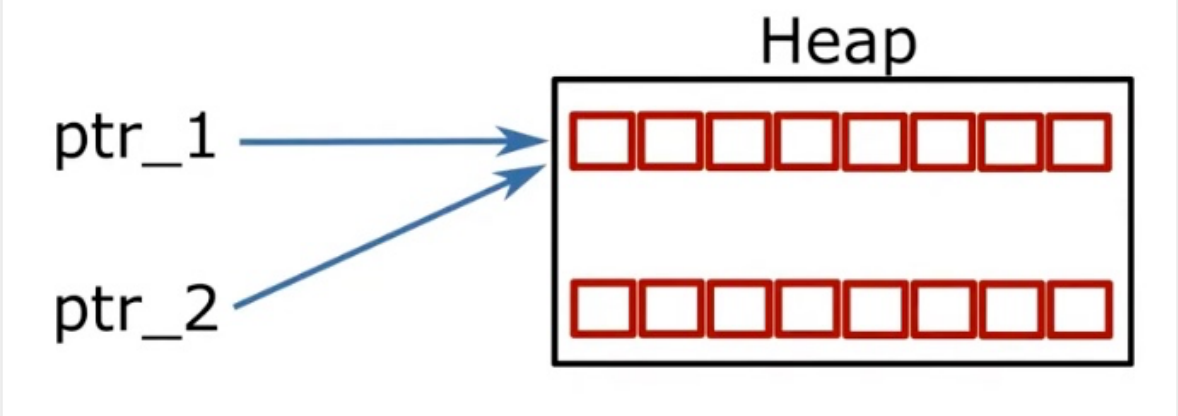
Memory Issues

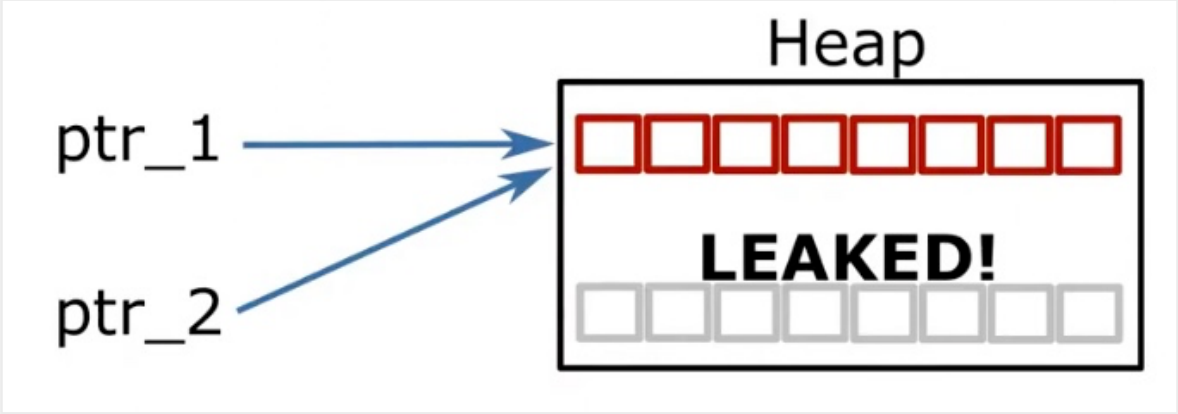
Memory Leak

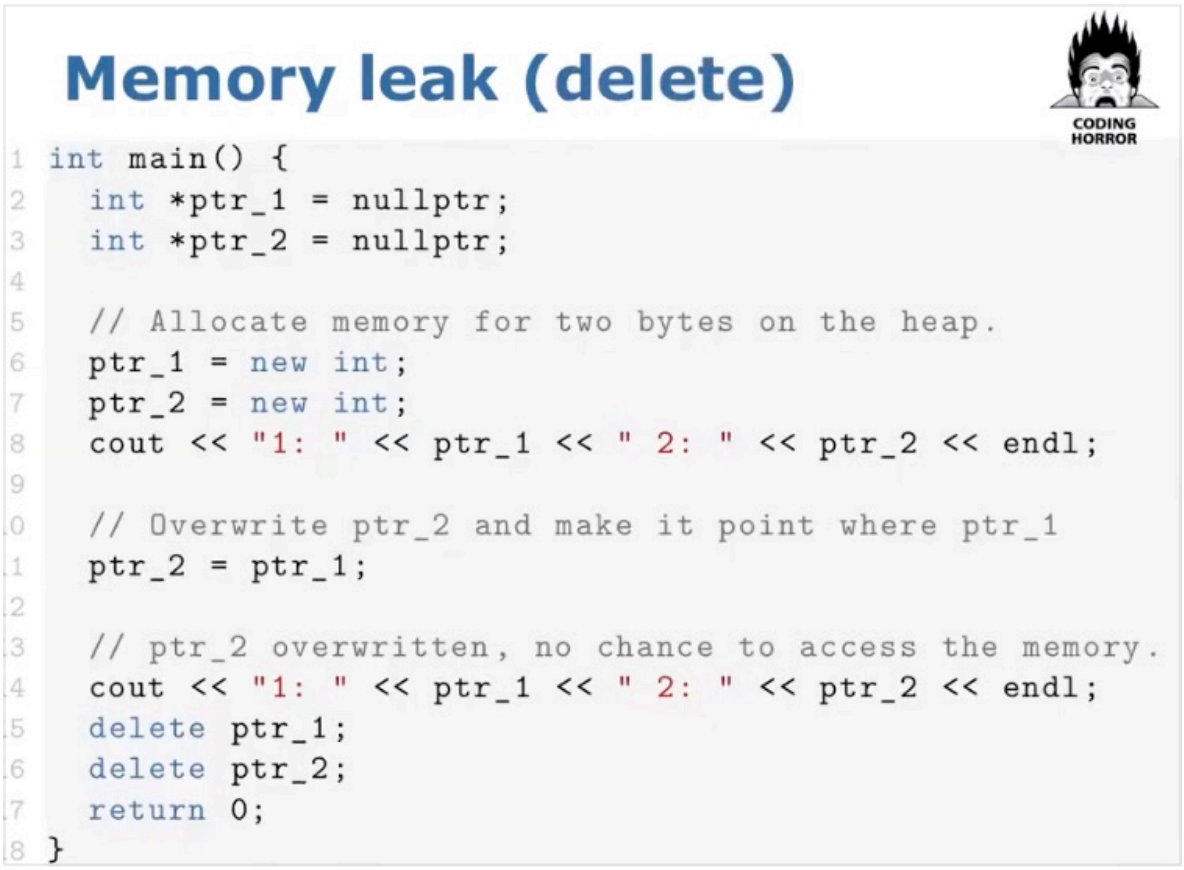
- Can happen when working with heap memory if we are not careful
- Memory leak: memory allocated on Heap access to which has been lost





Reference lost!



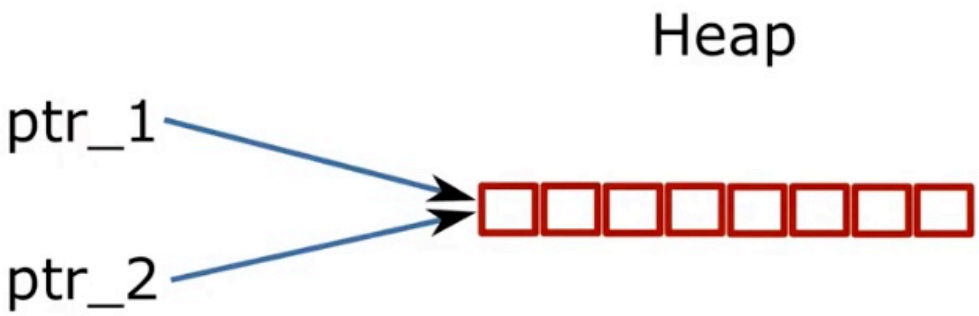


Dangling Pointer

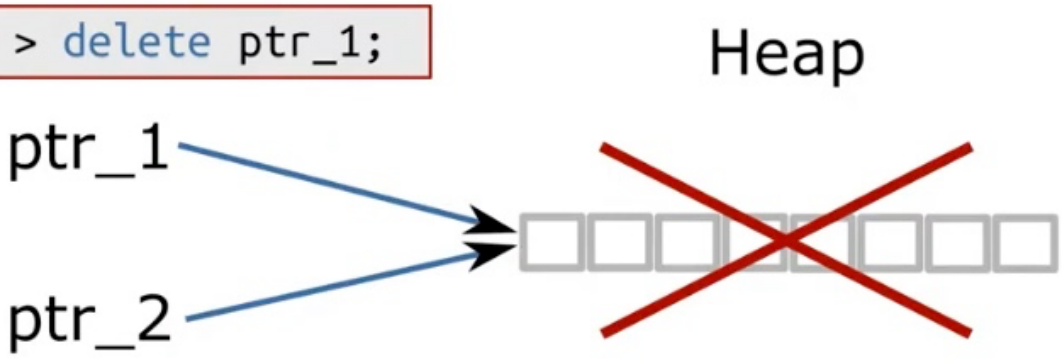
- Dangling Pointer: pointer to a freed memory
- Think of it as the opposite of memory leak
- Dereferencing a dangling pointer causes undefined behaviour

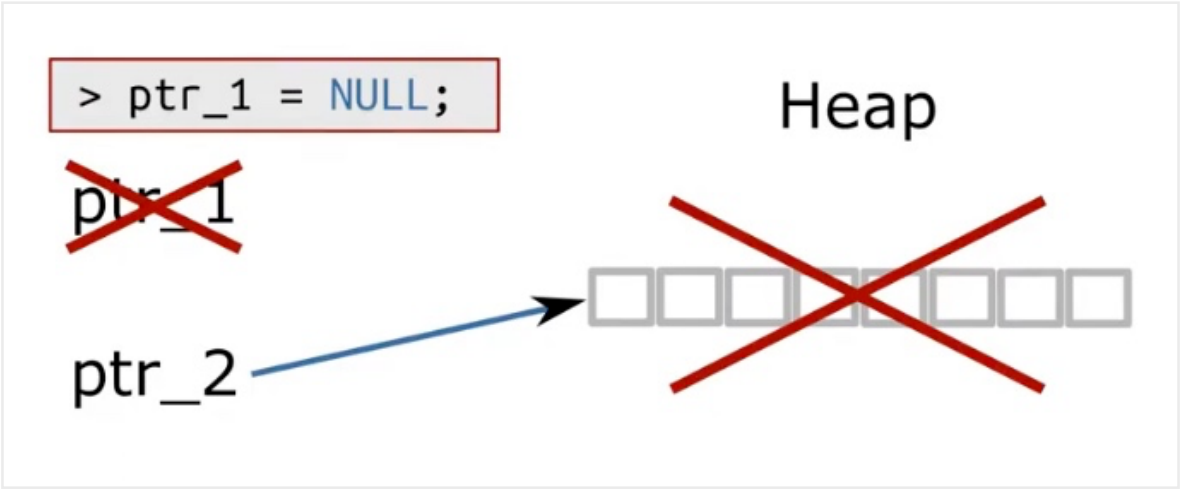
Dangling pointer

```
1 int* ptr_1 = some_heap_address;
2 int* ptr_2 = some_heap_address;
3 delete ptr_1;
4 ptr_1 = nullptr;
5 // Cannot use ptr_2 anymore! Behavior undefined!
```




```
> delete ptr_1;
```





RAII

- Resource Allocation is Initialization
- New object -> allocate memory
- Remove object -> free memory
- Objects own their data!

```
class MyClass {
 public:
  MyClass() { data_ = new SomeOtherClass; }
  ~MyClass() {
    delete data_;
    data_ = nullptr;
  }
 private:
  SomeOtherClass* data_;
};
```

- Still cannot copy an object of MyClass!!

```cpp
struct SomeOtherClass {};
class MyClass {
 public:
  MyClass() { data_ = new SomeOtherClass; }
  ~MyClass() {
    delete data_;
    data_ = nullptr;
  }
 private:
  SomeOtherClass* data_;
};
int main() {
  MyClass a;
  MyClass b(a);
  return 0;
}
```


```
*** Error in `raii_example':
double free or corruption: 0x0000000000877c20 ***
```

Shallow vs Deep copy

- Shallow Copy: just copy pointers, not data
- Deep Copy: copy data, create new pointers
- Default copy constructor and assignment operator implement shallow copying
- RAII + shallow copying -> dangling pointer
- RAII + Rule of All or Nothing (define each special function) -> correct
- Use smart pointers instead!