Smart pointer

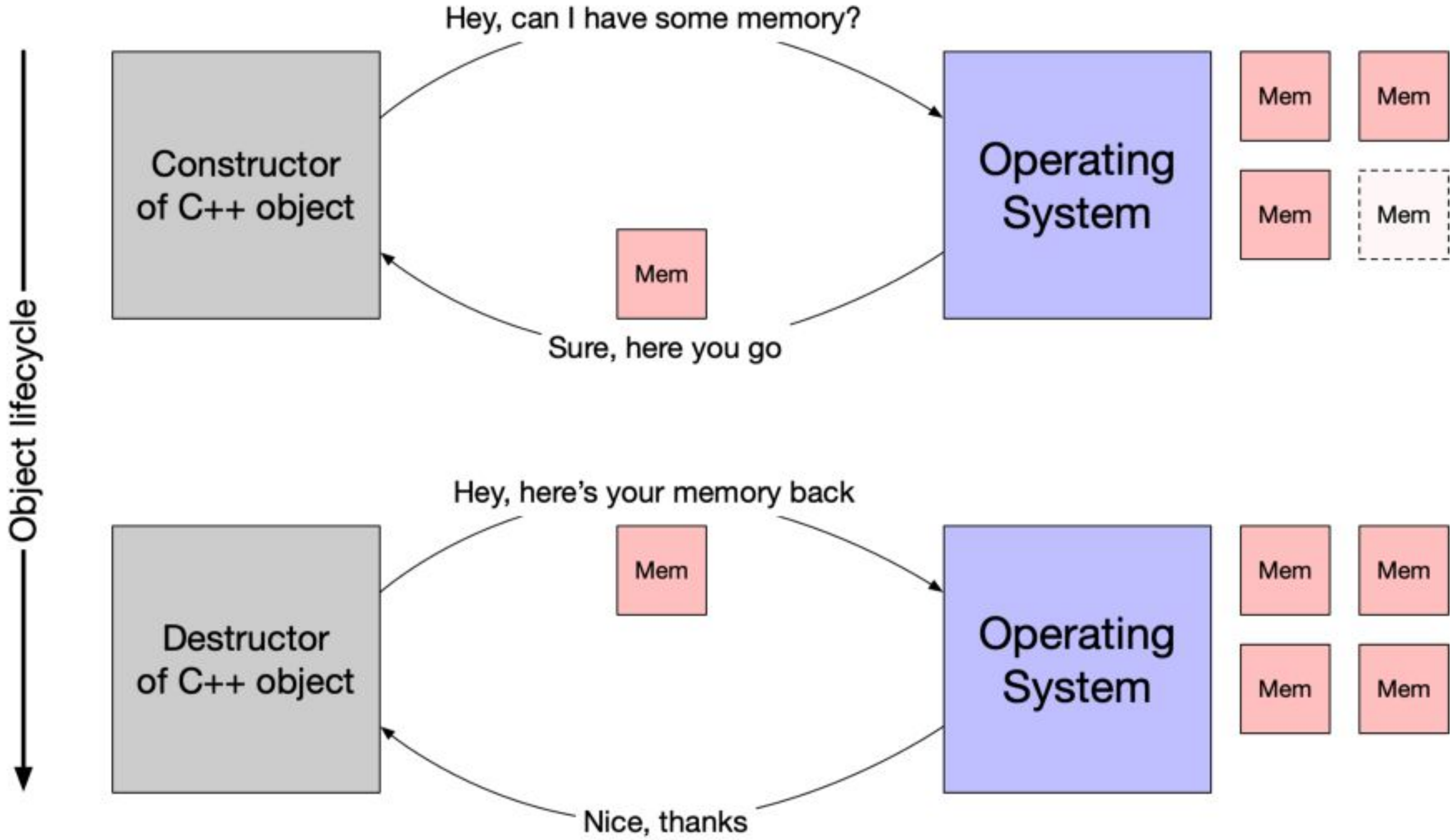
1. RAII
2. unique_ptr
3. shared_ptr
4. weak_ptr

*R*esource *A*llocation *I*s *I*nitialization

```
// Problem #1
{
 int *arr = new int[dynamicSize];
} // arr goes out of scope but we didn't delete it, we have a memory leak


// Problem #2
std::mutex globalMutex;
void funcCalledInMultipleThreads() {
 globalMutex.lock();
 // Code that runs in multiple threads...
} // We never unlocked the mutex, so this function will deadlock
```

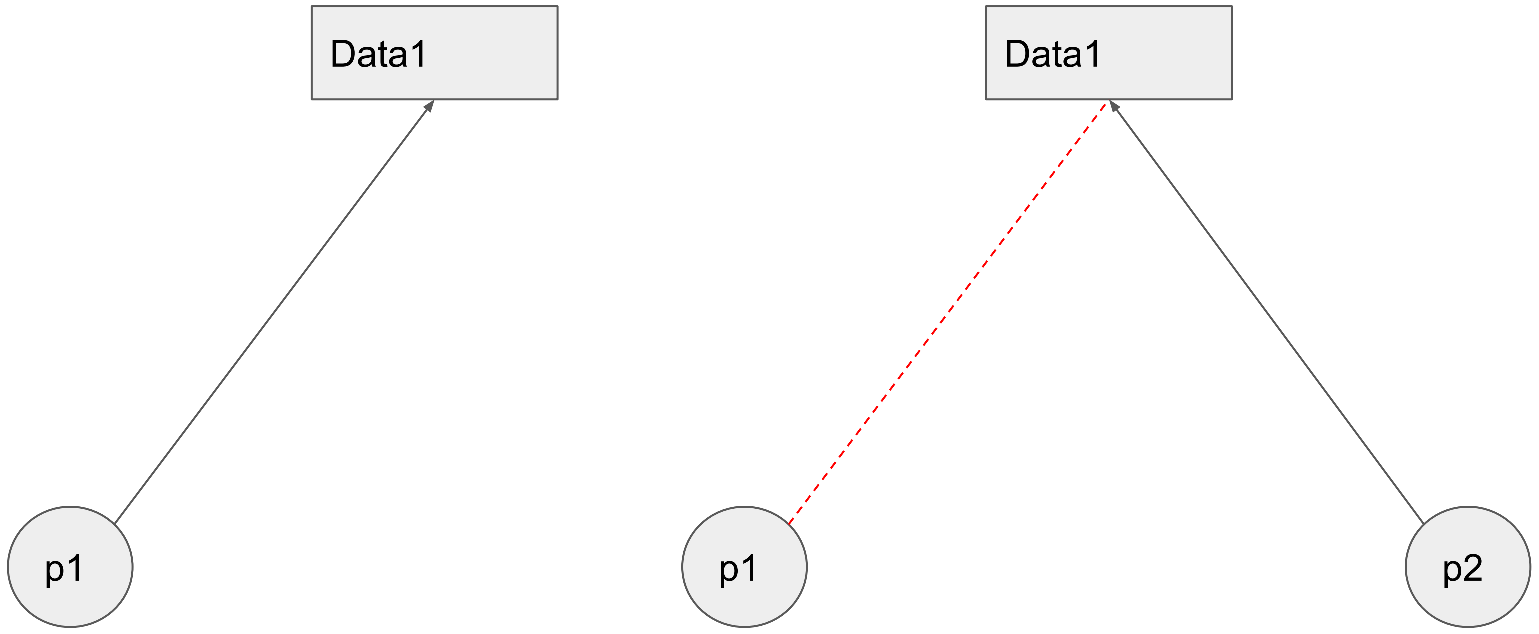
```
// Problem #3
{
 std::thread t1([]() {
   std::cout << "In a thread" << std::endl;
   // Do some stuff...
   return 5;
 });
} // Thread goes out of scope and is joinable, std::terminate
is called
```


Hey, can I have some memory?
Constructor of C++ object
Operating System
Mem
Mem
Mem
Mem
Mem
Sure, here you go
Object lifecycle
Hey, here's your memory back
Mem
Destructor of C++ object
Operating System
Mem
Mem
Mem
Mem
Nice, thanks

History

C++ 98	std::auto_ptr ❌	- Exclusive - Move - Cannot be used with STL ("copy-constructible" and "assignable")
C++ 11	std::unique_ptr	- Exclusive - Moves during copy - Deals with noncopyable
	std::shared_ptr	- Can be shared - Ref count mechanism
	std::weak_ptr	- Borrows - Breaks cyclic references - Reference count don’t changes

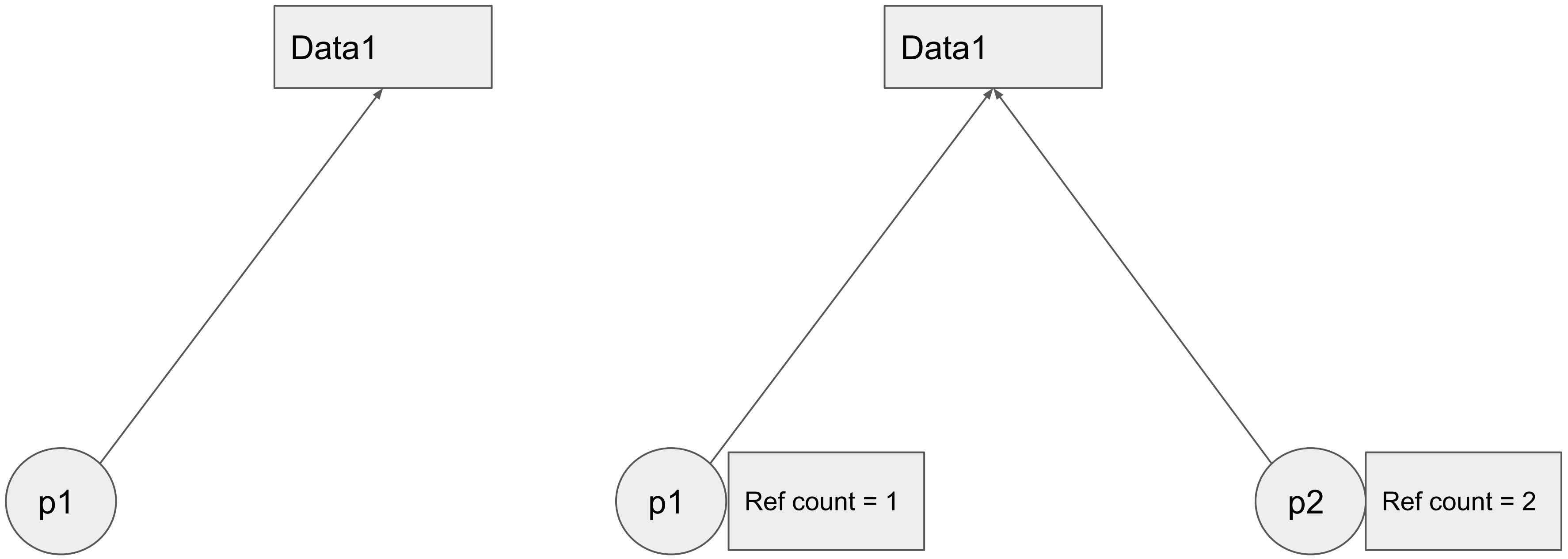
std::unique_ptr



std::unique_ptr

Function	Description
`uniq.release()`	Returns a pointer to the resource and releases it
`uniq.get()`	Returns a pointer to the resource
`uniq.reset(ptr)`	▪ Resets the resource to a new one ▪ Deletes the old resource
`uniq.get_deleter()`	Returns the deleter
`std::make_unique(....)`	Creates the resource and wraps it in a `std::unique_ptr`

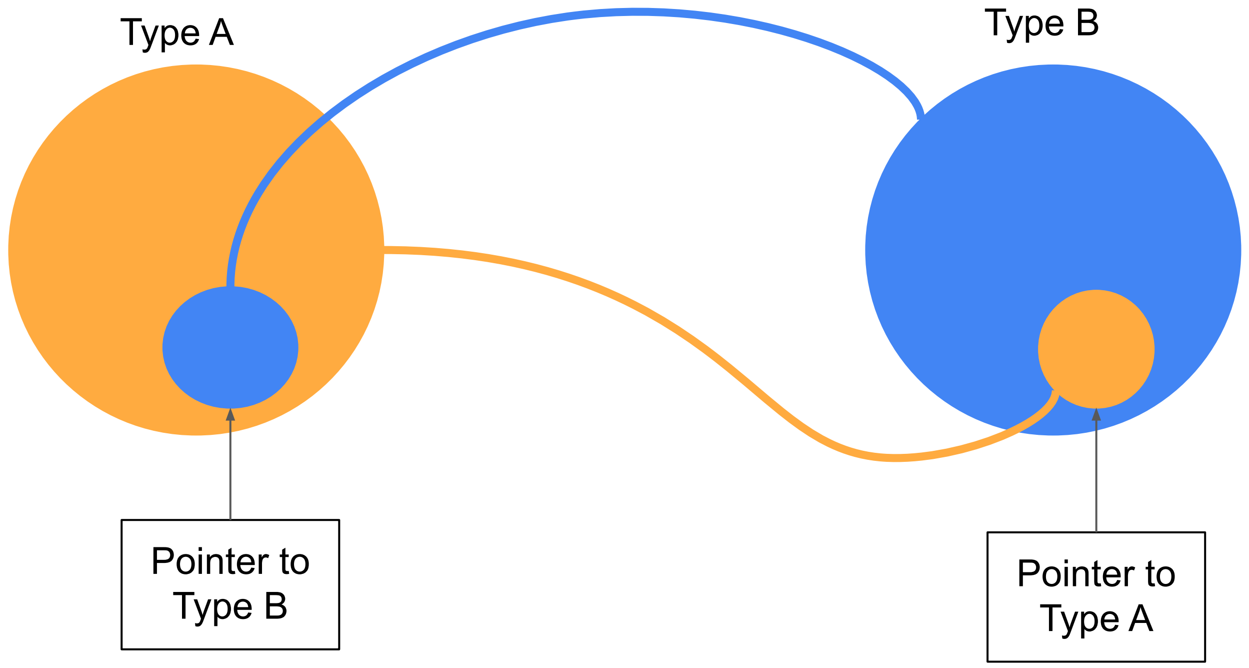
std::shared_ptr

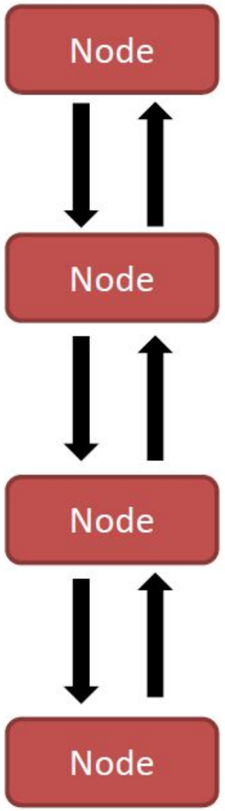


std::shared_ptr

Function	Description
`sha.unique()`	Checks if the `std:shared_ptr` is the unique owner of the resource
`sha.use_count()`	Returns the value of the reference counter
`sha.get()`	Returns a pointer to the resource
`sha.reset(ptr)`	▪ Resets the resource ▪ Deletes eventually the resource
`sha.get_deleter()`	Returns the deleter
`std::make_shared(....)`	Creates the resource and wraps it in a `std::shared_ptr`

Cyclic dependency





std::weak_ptr

- Doesn't owns the resource rather borrows it from a ***shared_ptr***
- Cannot access the resource

Function	Description
`wea.expired()`	Checks if the resource exists
`wea.use_count()`	Returns the value of the reference counter
`wea.lock()`	Creates a `std::shared_ptr` to the resource if available
`wea.reset()`	Releases the resource

Performance

```
E:\projects\smart_ptr>g++ performance.cpp && a
new         : 1.08428 s
unique_ptr  : 1.76321 s
make_unique : 1.76131 s
shared_ptr  : 2.23858 s
make_shared : 3.22648 s
```

```
E:\projects\smart_ptr>g++ -std=c++17 performance.cpp && a
new         : 1.1149 s
unique_ptr  : 1.69143 s
make_unique : 1.70458 s
shared_ptr  : 2.22804 s
make_shared : 3.21749 s
```

```
E:\projects\smart_ptr>g++ -std=c++14 performance.cpp && a
new         : 0.887534 s
unique_ptr  : 1.46738 s
make_unique : 1.51619 s
shared_ptr  : 2.02721 s
make_shared : 3.01955 s
```

```
E:\projects\smart_ptr>g++ -std=c++20 performance.cpp && a
new         : 2.4557 s
unique_ptr  : 3.04152 s
make_unique : 3.06957 s
shared_ptr  : 3.64781 s
make_shared : 4.44141 s
```

```
E:\projects\smart_ptr>g++ -std=c++23 performance.cpp && a
new         : 2.4278 s
unique_ptr  : 3.0326 s
make_unique : 3.13849 s
shared_ptr  : 3.58022 s
make_shared : 4.48865 s
```

What’s next...

1. deleter
2. Concurrency - atomic

References

https://youtu.be/sQCSX7vmmKY

https://docs.microsoft.com/en-us/cpp/cpp/how-to-create-and-use-unique-ptr-instances?view=msvc-170