Memory Allocation for Virtual Methods and Overriding in C#

# 🧠 Concept Summary

When you declare a method as 'virtual' in a base class and override it in a derived class, the runtime determines which method to call based on the actual object instance type, not the reference type. This behavior is implemented using the Virtual Method Table (vtable).

# 🔧 How Memory is Managed

1. Virtual Method in Base Class (`Vehicle`)

- A virtual method in the base class is added to the vtable.  
- The vtable is associated with the class type, not each object.  
- It's a lookup table holding method pointers for that class.

2. Override in Derived Class (`Car`, `ElectricScooter`)

- The derived class replaces the base method's vtable entry with its own method.  
- It does not alter the base class.  
- Each derived class has its own vtable pointing to its own method.

# 🧩 Visual Representation

[Vehicle vtable]  
- Start() → Vehicle.Start()  
  
[Car vtable]  
- Start() → Car.Start()  
  
[ElectricScooter vtable]  
- Start() → ElectricScooter.Start()

# 📌 Runtime Behavior Example

Vehicle v1 = new Car();  
v1.Start(); // Output: "Starting a car with a key."

Even though v1 is of type Vehicle, because it points to a Car object, the CLR looks up Car's vtable and invokes Car.Start().

# 📂 Memory Notes

- Virtual method definitions are stored once per method in the method table.  
- Object instances contain a reference to their type's vtable.  
- No duplicate memory for methods; only vtable pointers are updated in derived classes.

# ✅ Conclusion

- Base class defines the virtual method and adds it to the vtable.  
- Derived classes override the vtable entry to point to their own method.  
- Method calls are resolved at runtime using the vtable of the actual object type.