SOLID Principles

Architecture Engine Club Summer 2023

Goals

Cleaner Code

Isolates Bugs

Easier Testing

Better Reusability

Principles

Single Responsibility

Open-Closed

Liskov Substitution

Interface Segregation

Dependency Inversion

Single Responsibility Principle

SINGLE RESPONSIBILITY PRINCIPLE

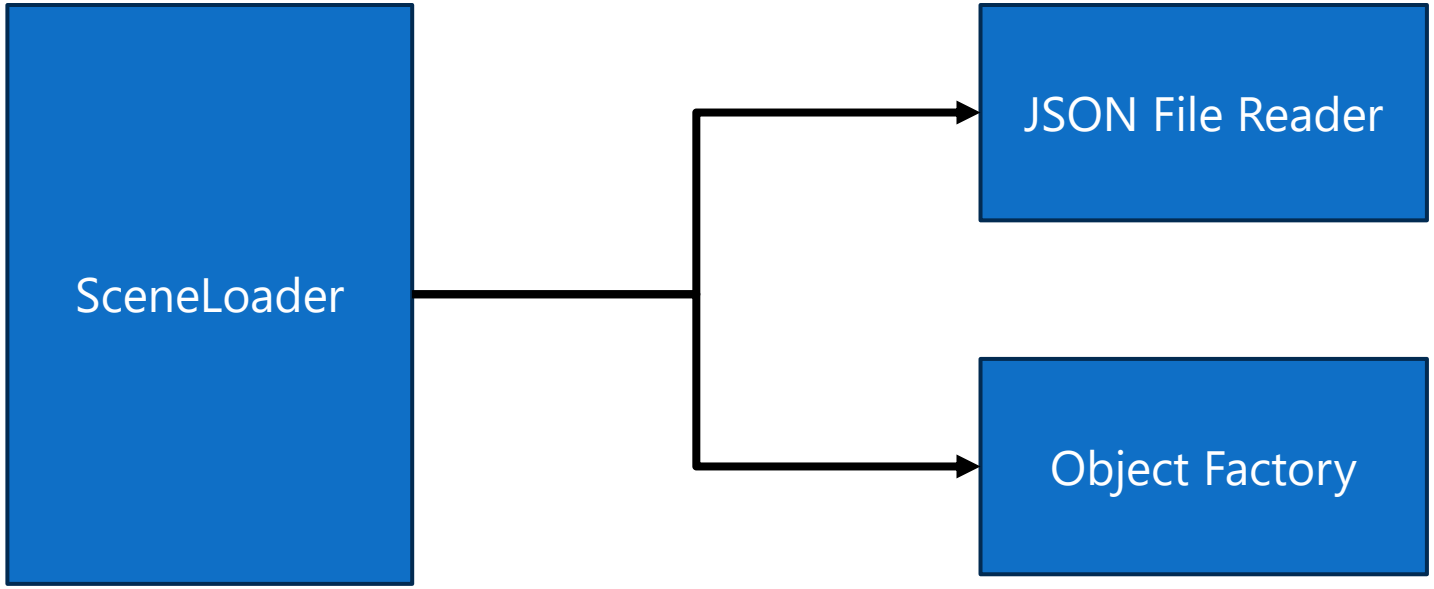
Definition

Every class should only ever have one and **only one responsibility**

SINGLE RESPONSIBILITY PRINCIPLE

Example

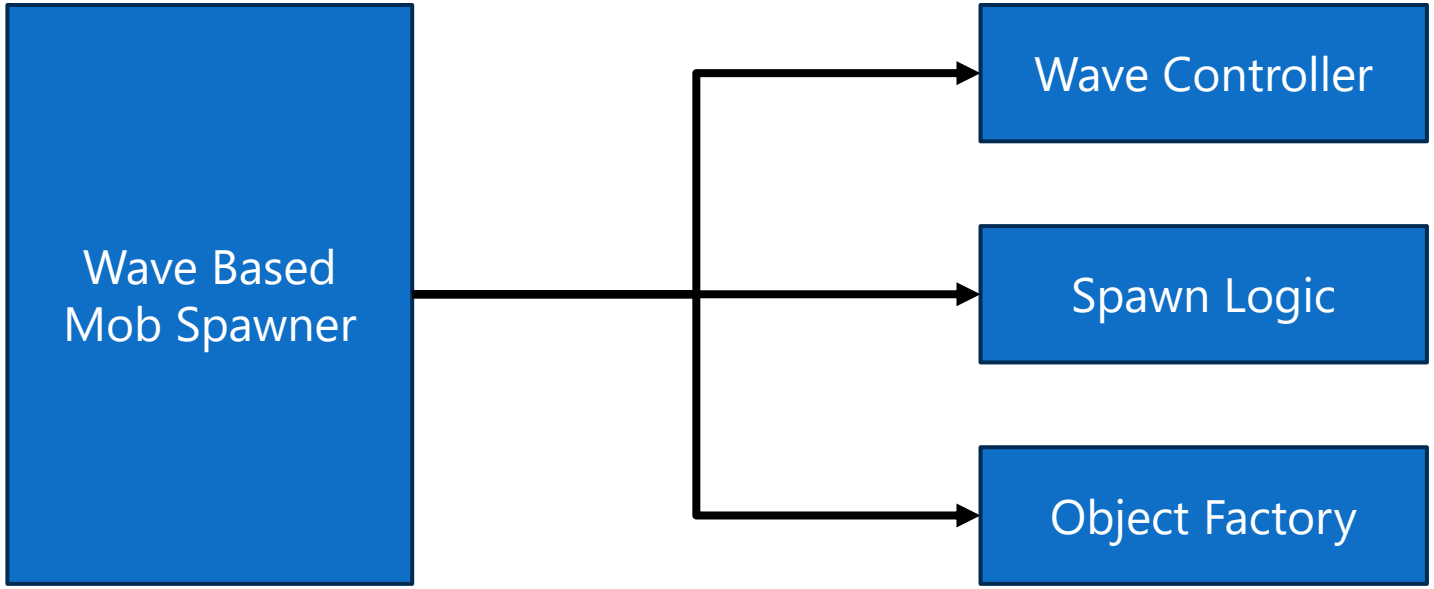
Split a class to individual features/responsibilities



SINGLE RESPONSIBILITY PRINCIPLE

Example

Split a class to individual features/responsibilities



Open-closed Principle

OPEN-CLOSED PRINCIPLE

Definition

Software entities should be **open for extension**
but **closed for modification**

OPEN-CLOSED PRINCIPLE

Example

What's wrong here?

```
class SecretNumber
{
  protected:
    int* secret;
  public:
    SecretNumber()
    : secret { new int(420) }
    {}
    int GetSecret() const { return *secret; }
};
class NottiBoy : public SecretNumber
{
  private:
    int bar;
  public:
    int DoStuff()
    {
        secret = nullptr;
        return bar;
    }
};
```

OPEN-CLOSED PRINCIPLE

Example

We should not expose variables that inheritors can modify to destroy existing behaviour

```
class SecretNumber
{
  private: // Changed to private
    int* secret;
  public:
    SecretNumber()
    : secret { new int(420) }
    {}
    int GetSecret() const { return *secret; }
};
class NottiBoy : public SecretNumber
{
  private:
    int bar;
  public:
    int DoStuff()
    {
        secret = nullptr; // Now we can't do this!
        return bar;
    }
};
```

Liskov Substitution Principle

LISKOV SUBSTITUITION PRINCIPLE

Definition

Functions that use pointers or references to base classes **must be able to use objects of derived** classes **without knowing it**

LISKOV SUBSTITUITION PRINCIPLE

Example

Base Class

- Object must be usable immediately after construction
- `Foo()` only throws `std::invalid_argument`

Bad Derived Class

- Object must call `SetUp()` before use
- Overriden `Foo()` also throws `std::out_of_range`

LISKOV SUBSTITUITION PRINCIPLE

Example

Base Class

- Object must be usable immediately after construction
- `Foo()` only throws `std::invalid_argument`

Good Derived Class

- `SetUp()` automatically called in constructor
- Overriden `Foo()` only throws `std::invalid_argument`

Interface Segregation Principle

INTERFACE SEGREGATION PRINCIPLE

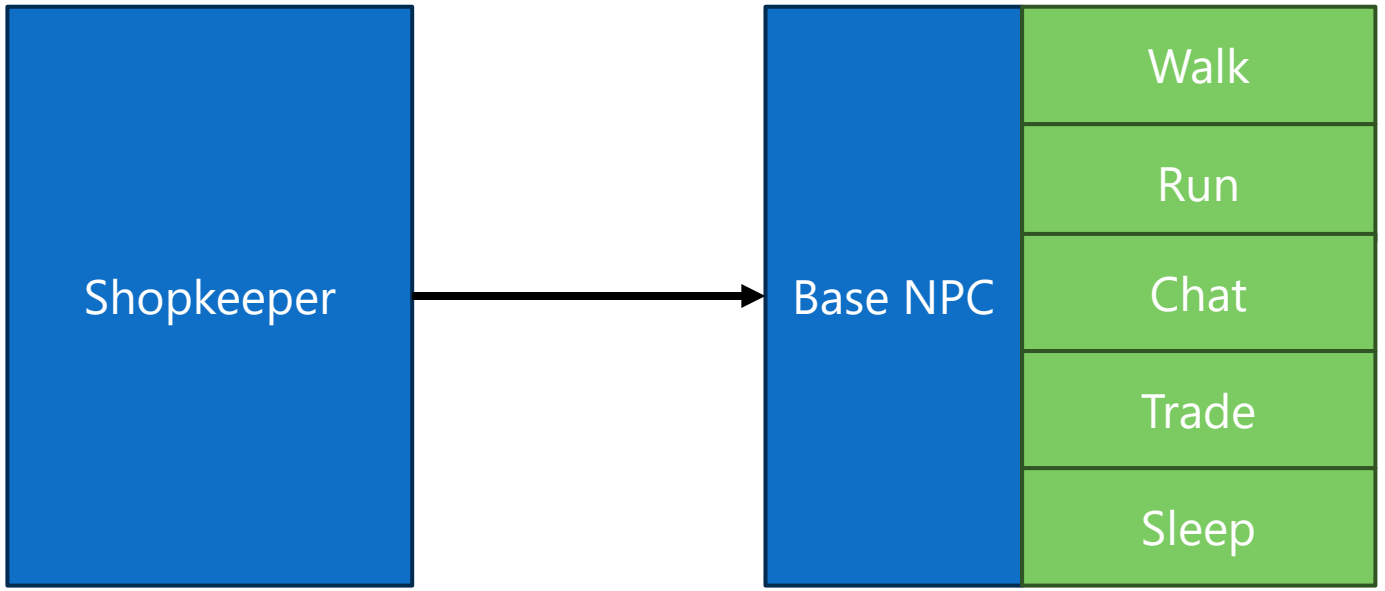
Definition

Clients should **not be forced to depend** on interfaces **that they do not use**

INTERFACE SEGREGATION PRINCIPLE

Example

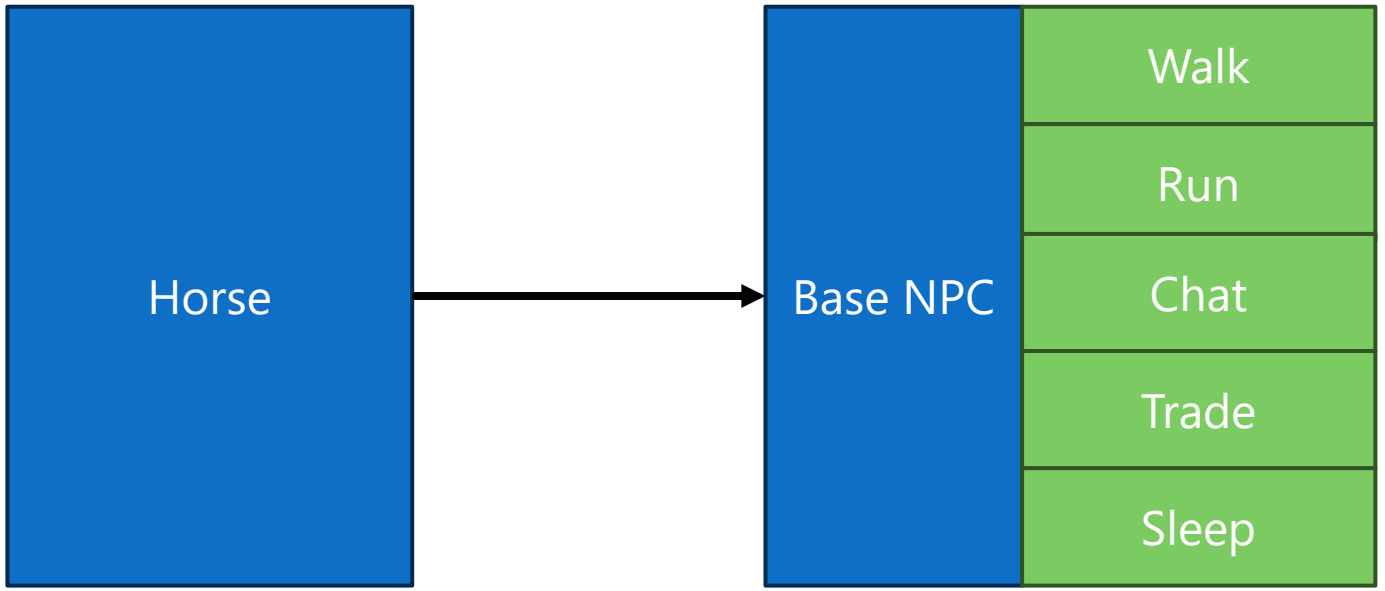
Looks about right for a shopkeeper, but…



INTERFACE SEGREGATION PRINCIPLE

Example

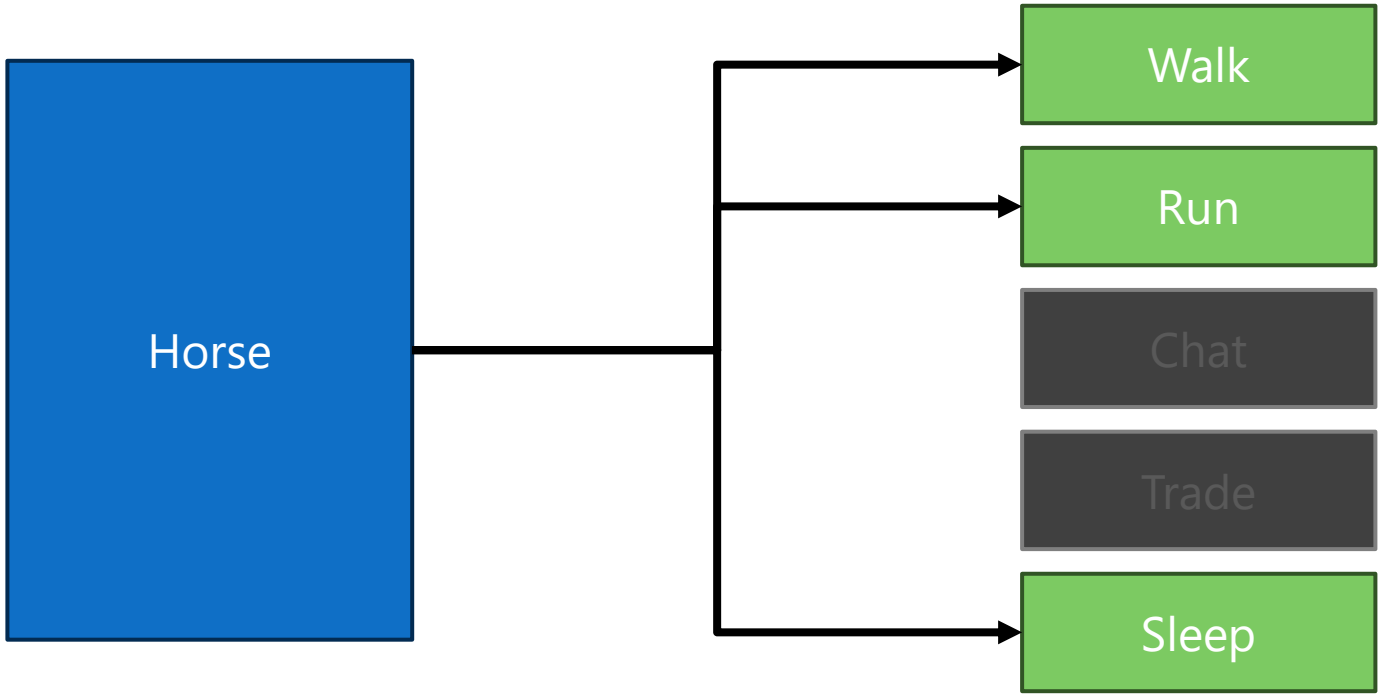
...not really for a horse, right?



INTERFACE SEGREGATION PRINCIPLE

Example

Hence, interface segregation



Dependency Inversion Principle

DEPENDENCY INVERSION PRINCIPLE

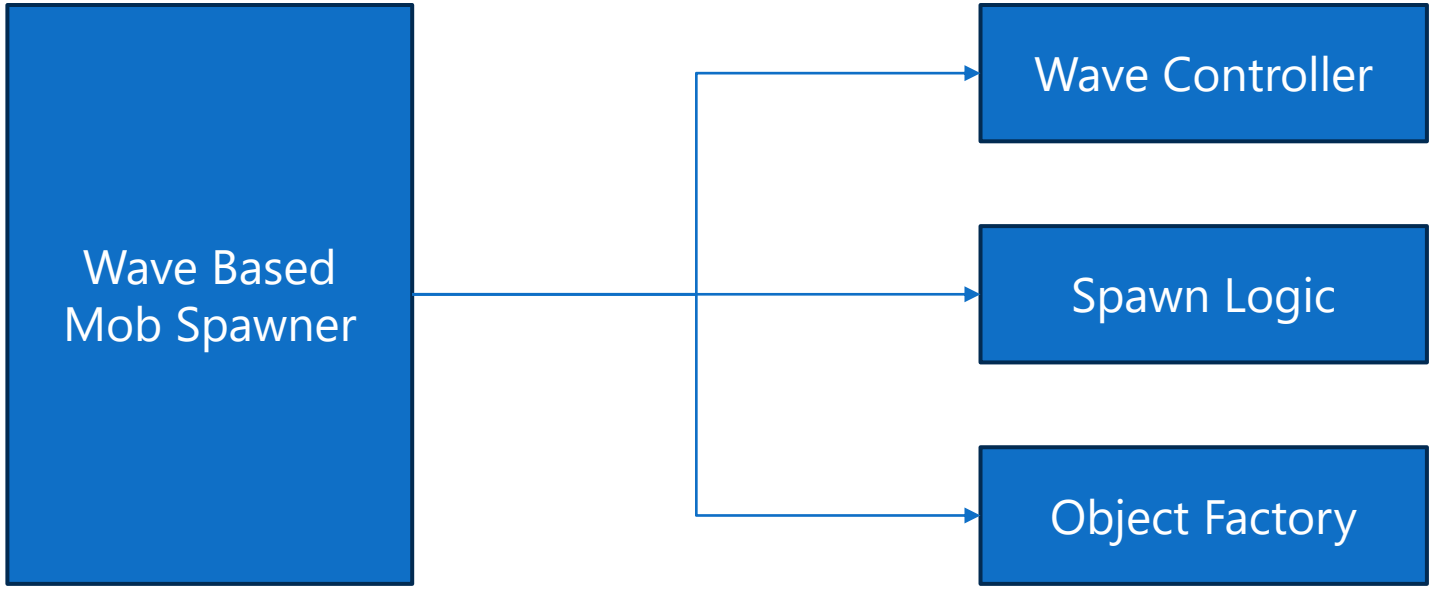
Definition

Depend upon abstractions, not implementations

DEPENDENCY INVERSION PRINCIPLE

Example

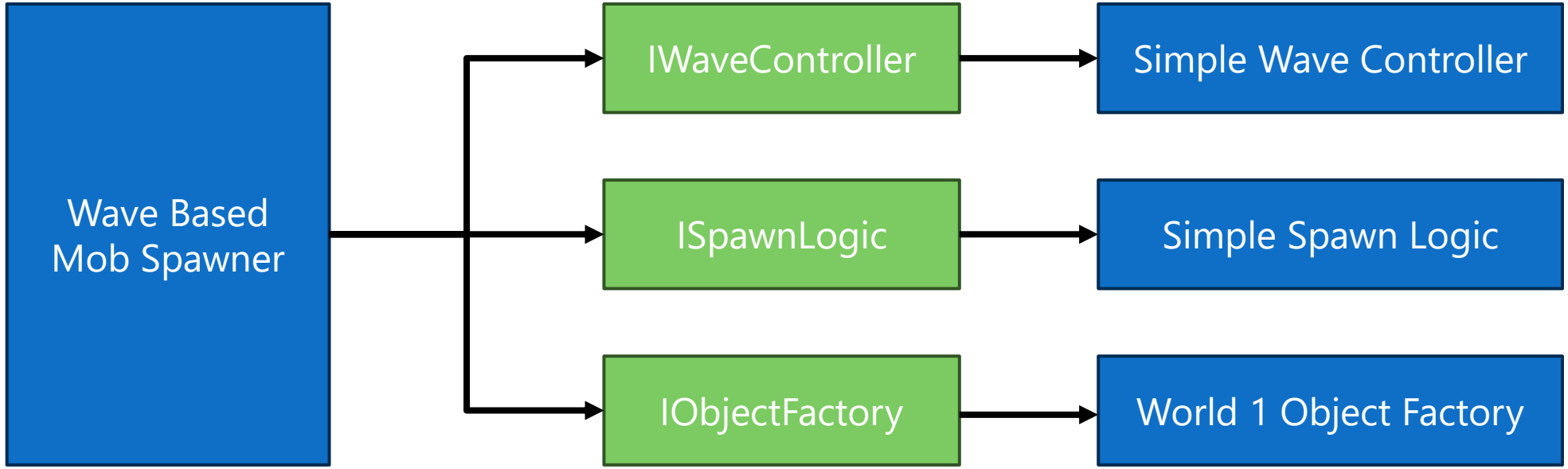
Now what if we want to swap out our behaviours?



DEPENDENCY INVERSION PRINCIPLE

Example

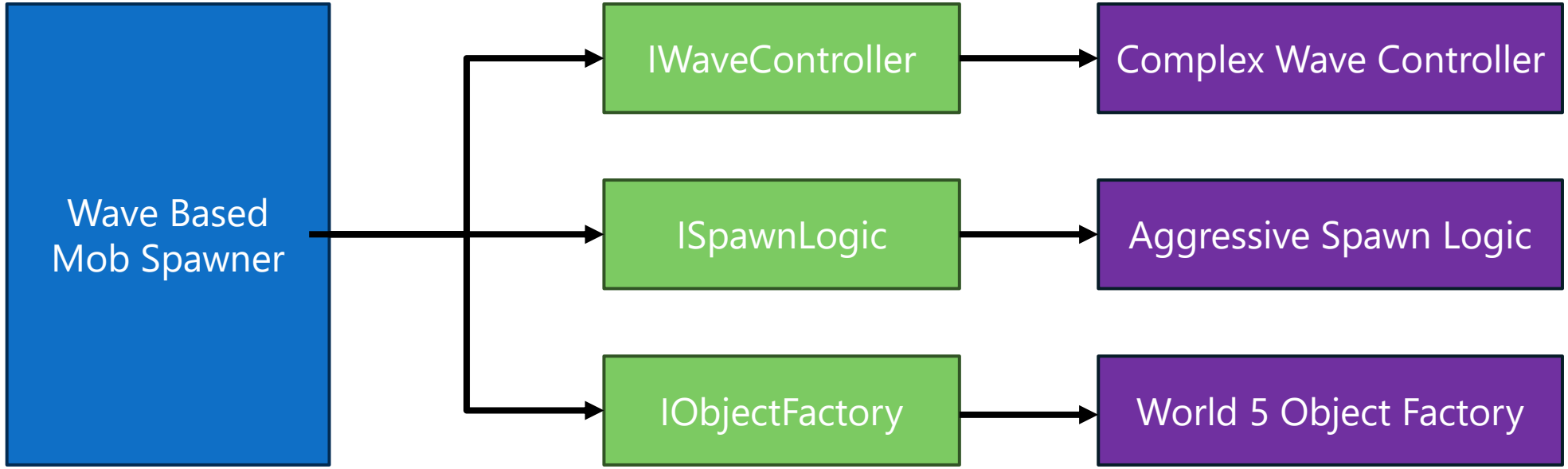
By practicing dependency inversion…



DEPENDENCY INVERSION PRINCIPLE

Example

We can essentially swap things out



Thanks for Listening

Any Questions?