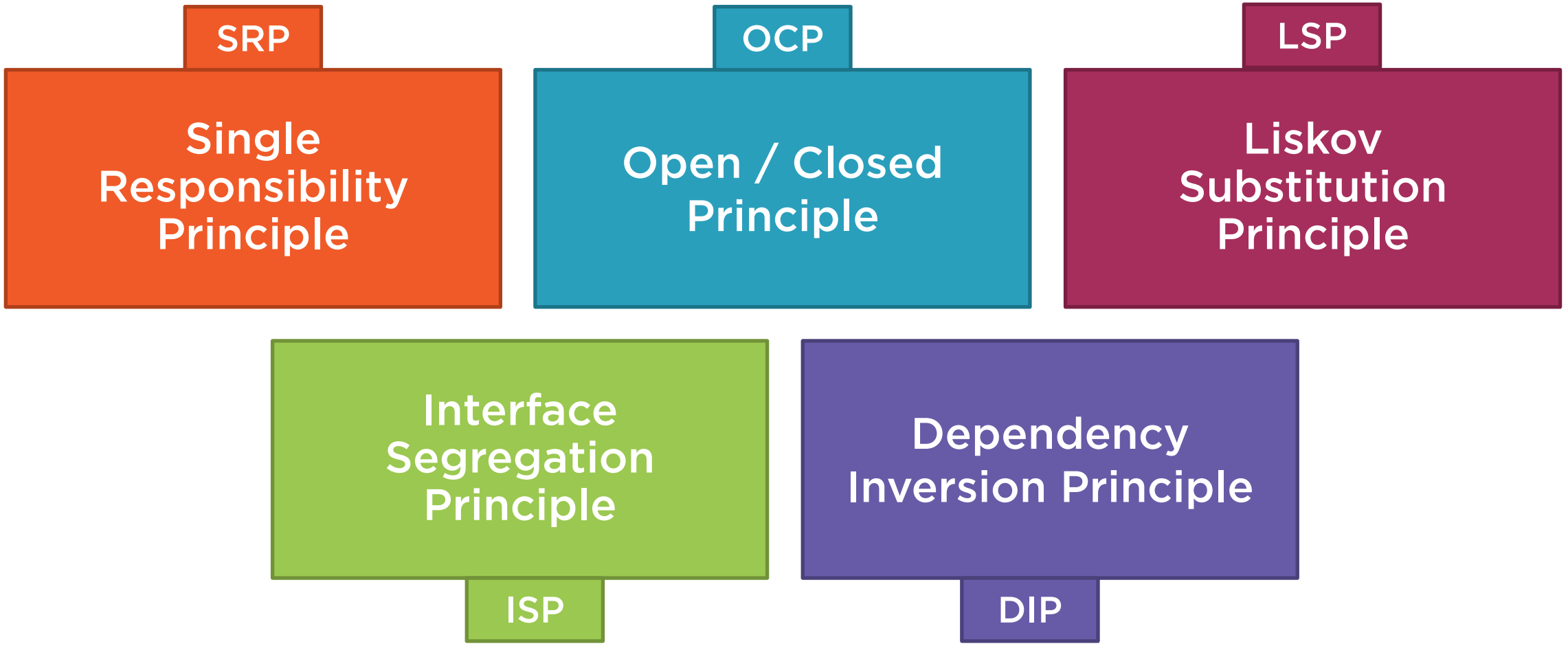
Dependency Inversion Principle

Steve Smith
FORCE MULTIPLIER FOR DEV TEAMS
@ardalis | ardalis.com | weeklydevtips.com

SOLID Principles



Dependency Inversion Principle

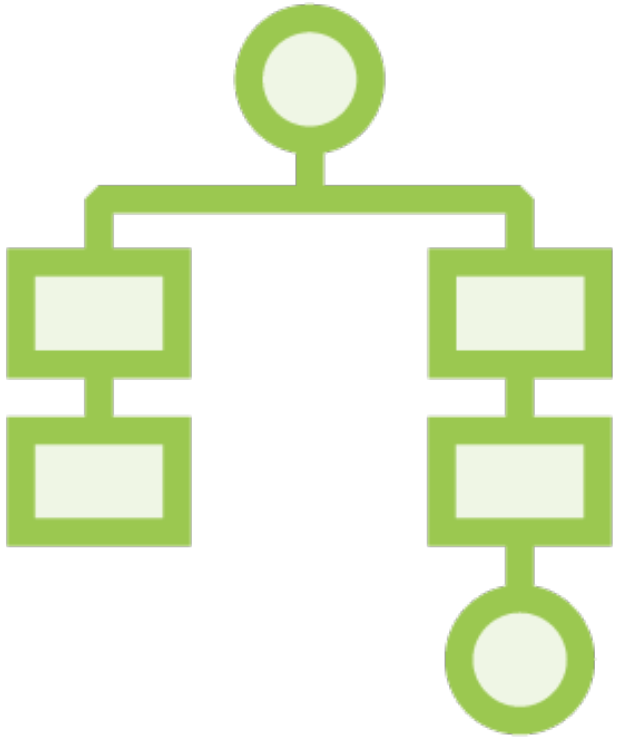
High-level modules should not depend on low-level modules. Both should depend on abstractions.

Abstractions should not depend on details.

Details should depend on abstractions.

How do I know if something depends on something else?

Dependencies in C#



References required to compile

References required to run

Learn More

Microsoft Reference Application + eBook

- github.com/dotnet-architecture/eShopOnWeb

Clean Architecture Solution Template

- github.com/ardalis/CleanArchitecture

On Pluralsight

- “Creating N-Tier Applications in C#”
- “Domain-Driven Design Fundamentals”

What's the difference between "high-level" and "low-level"?

High Level

More abstract

Business rules

Process-oriented

Further from input/output (I/O)

Low Level

Closer to I/O

"Plumbing" code

Interacts with specific external systems and hardware

Separation of Concerns

Keep plumbing code separate from high level business logic

What's an abstraction?

(in C#, in this context)

Abstractions in C#

Interfaces

Abstract base classes

“Types you can’t instantiate”

What about *details*?

Abstractions should not depend on *details*.
Details should depend on abstractions.

Details and Abstractions

Abstractions shouldn't be coupled to details

Abstractions describe what

- Send a message
- Store a Customer record

Details specify how

- Send an SMTP email over port 25
- Serialize Customer to JSON and store in a text file

Depending on Details

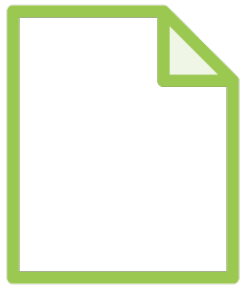
```
public interface IOrderDataAccess
{
    SqlDataReader ListOrders(SqlParameterCollection params);
}
```

Abstractions Should Not Depend on Details

```
public interface IOrderDataAccess
{
    List<Order> ListOrders(Dictionary<string,string> params);
}
```

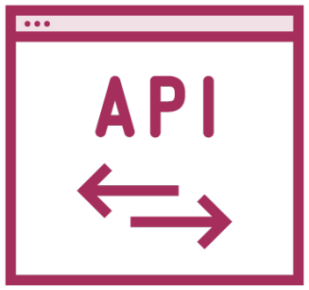
Low Level Dependencies

Database

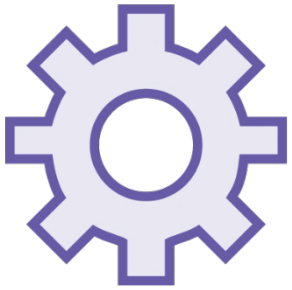


File system

Email



Web APIs



Configuration

Clock

Hidden Direct Dependencies

Direct use of low level dependencies

Static calls and `new`

Causes pain

- **Tight coupling**
- **Difficult to isolate and unit test**
- **Duplication**

New Is Glue

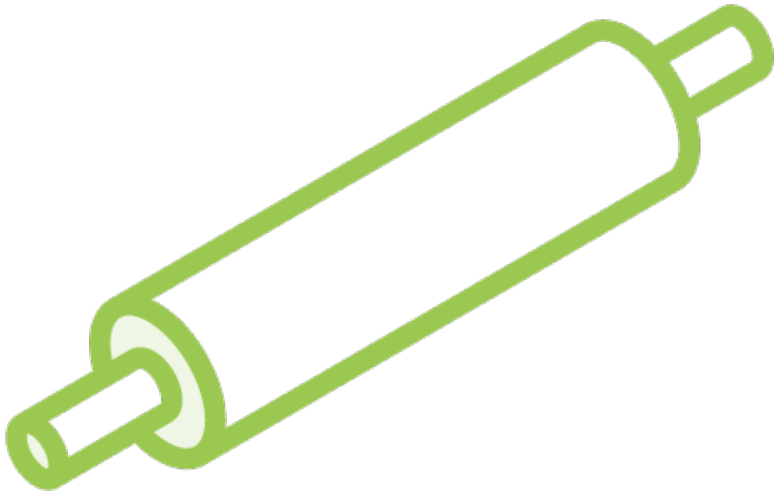
Using new to create dependencies glues your code to that dependency

- ardalis.com/new-is-glue

New isn't bad – just bear in mind the coupling it creates

- Do you need to specify the implementation?
- Could you use an abstraction instead?

Explicit Dependencies Principle

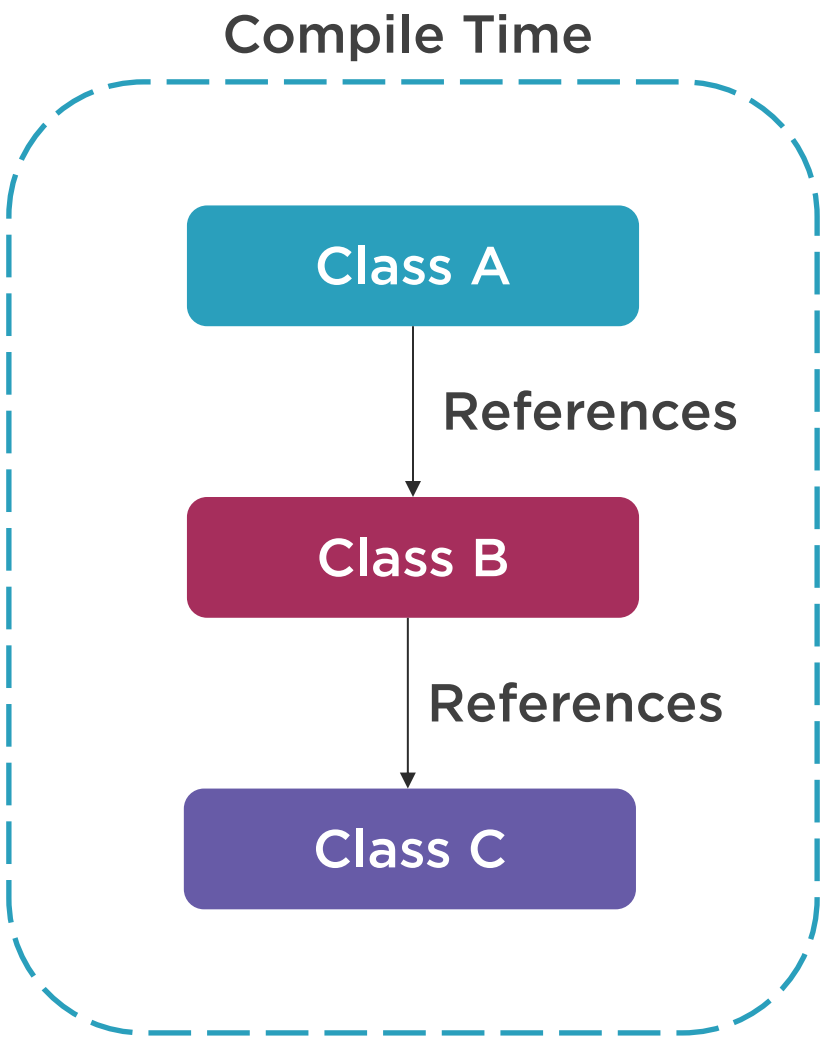


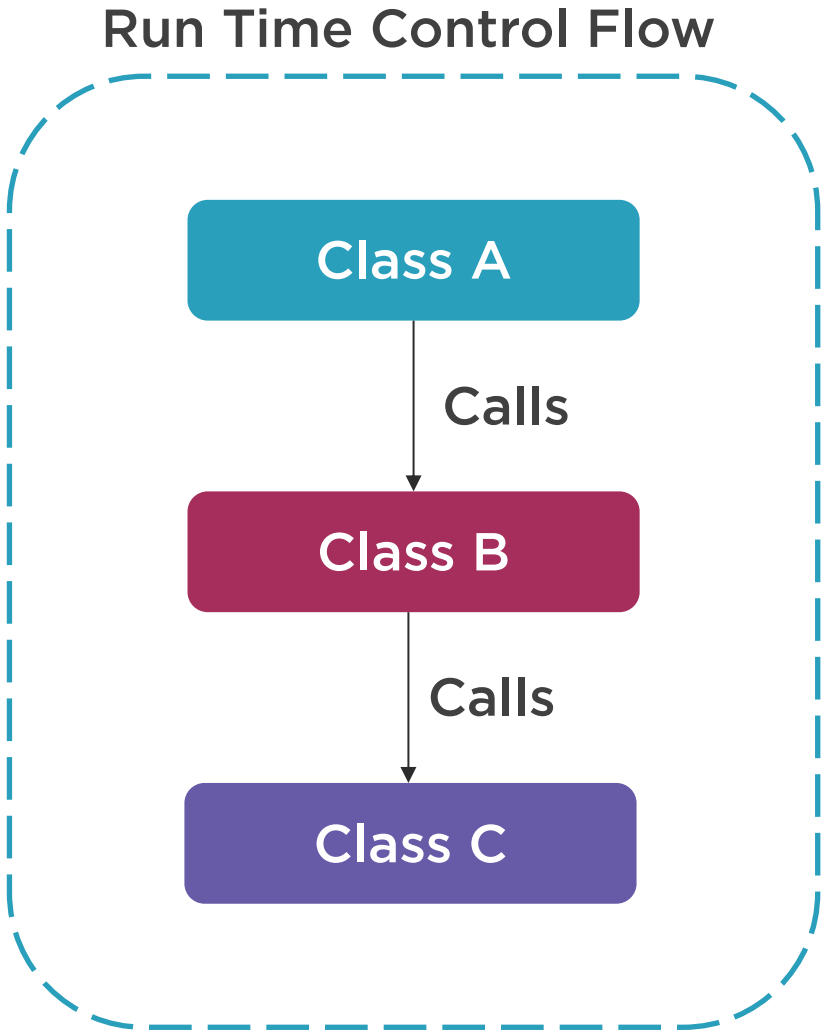
Your classes shouldn't surprise clients with dependencies

List them up front, in the constructor

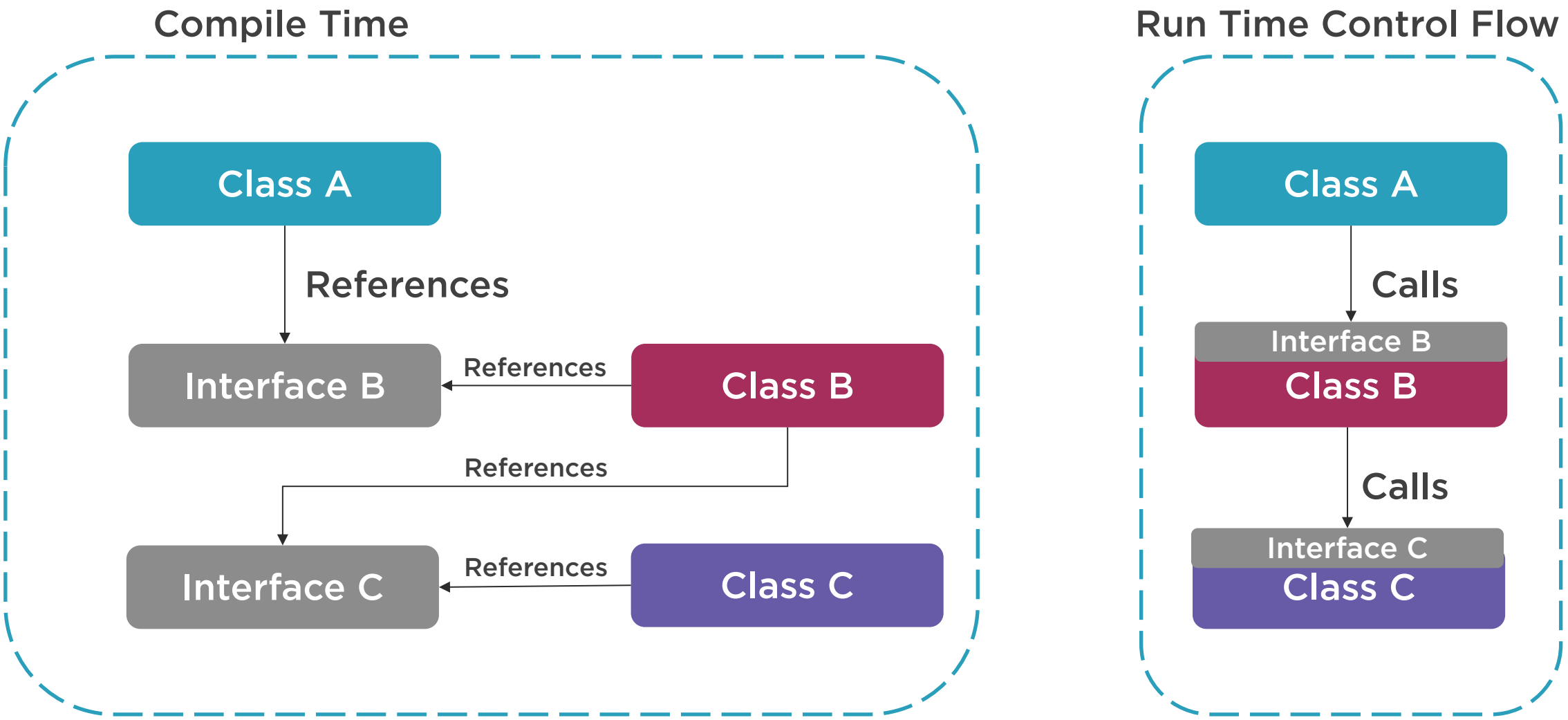
Think of them as ingredients in a cooking recipe

Dependencies without Abstractions





Dependencies with Abstractions



Learn More

Microsoft Docs: Architectural Principles

- bit.ly/2tmcebj

Explicit Dependencies Principle

- bit.ly/2lea2Nu

Dependency Injection

Don't create your own dependencies

- Depend on abstractions
- Request dependencies from client

Client injects dependencies as

- Constructor arguments
- Properties
- Method arguments

See also: Strategy Design Pattern

Tip: Prefer Constructor Injection

Follows Explicit Dependencies Principle

Classes are never in uninitialized state

Can leverage an IOC container to construct types and their dependencies

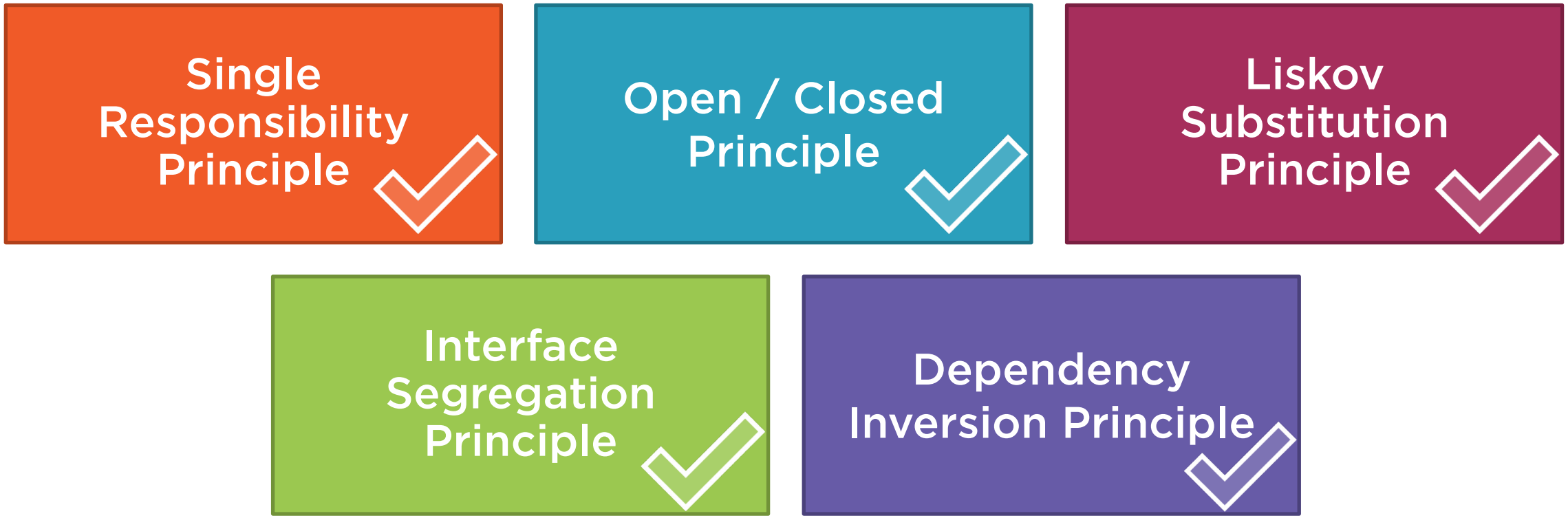
IOC, or “Inversion of Control” containers are sometimes called “dependency injection” (DI) containers or simply services containers.

Demo

Applying DIP to ArdalisRating

Available at
https://github.com/ardalis/solidsample

SOLID Principles



Too many files?

Use folders!

Demo

Organizing ArdalisRating and Supporting a Web Front End

Available at https://github.com/ardalis/solidsample

Key Takeaways

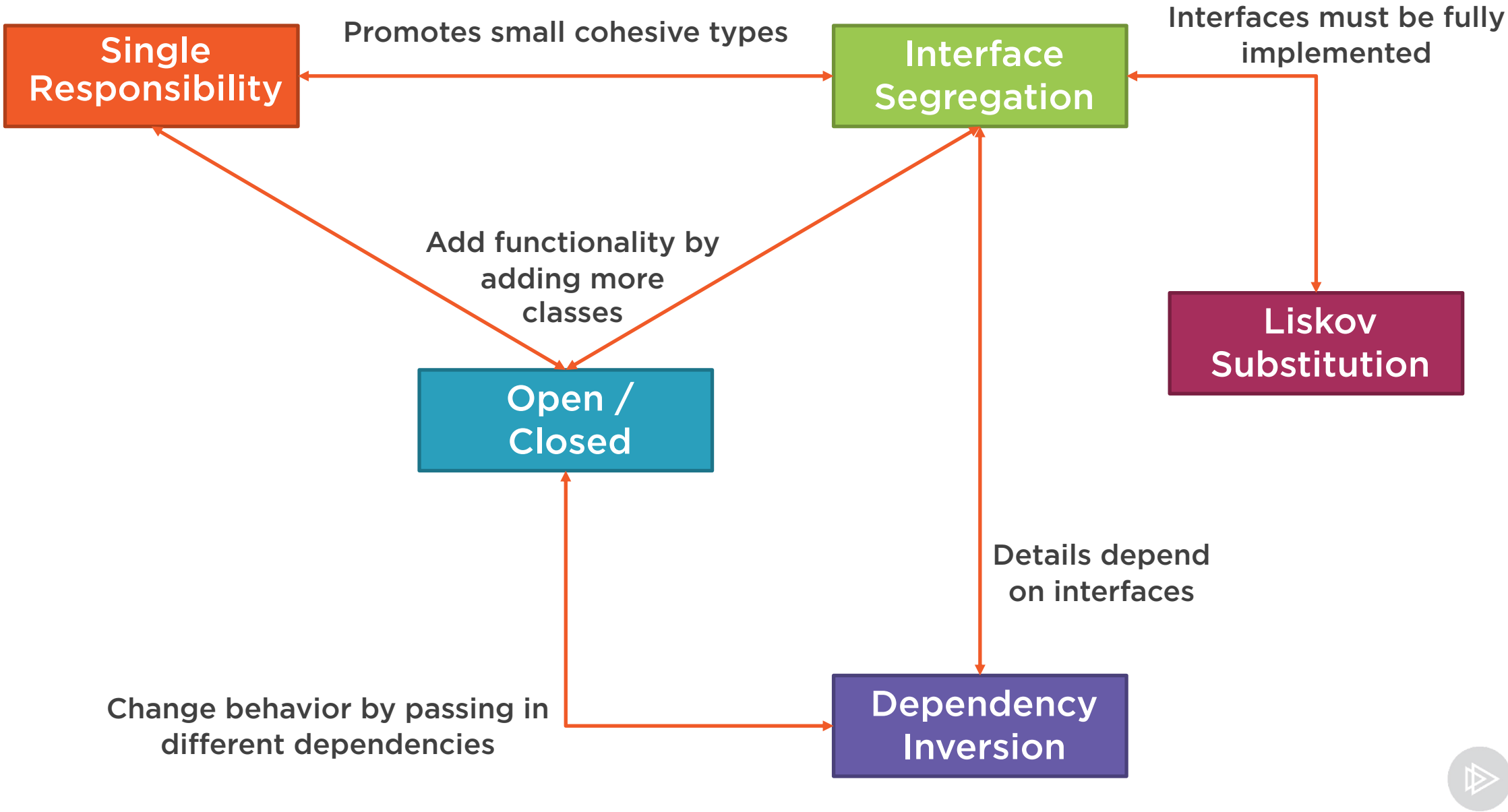
- Most classes should depend on abstractions, not implementation details
- Abstractions shouldn't leak details
- Classes should be explicit about their dependencies
- Clients should inject dependencies when they create other classes
- Structure your solutions to leverage dependency inversion

Course Summary

Review principles – see next slide

Code Samples

- https://github.com/ardalis/solidsample


Single Responsibility
Promotes small cohesive types
Interface Segregation
Interfaces must be fully implemented
Add functionality by adding more classes
Liskov Substitution
Open / Closed
Details depend on interfaces
Change behavior by passing in different dependencies
Dependency Inversion

SOLID Principles for C# Developers

Steve Smith
FORCE MULTIPLIER FOR DEV TEAMS
@ardalis | ardalis.com | weeklydevtips.com