The OO Design Principles

Agenda

- Need of Design
- Design Smells
- Class Design Principles
- Summary

Agenda

- Need of Design
- Design Smells
- Class Design Principles
- Summary

What Design Exactly About is..?

What's meaning of design..?

What's the difference compared to analysis..?

Why do we need design..?

To deliver faster

To manage change

To deal with complexity

Why do we Need Design..?

Agenda

- Need of Design
- Design Smells
- Class Design Principles
- Summary

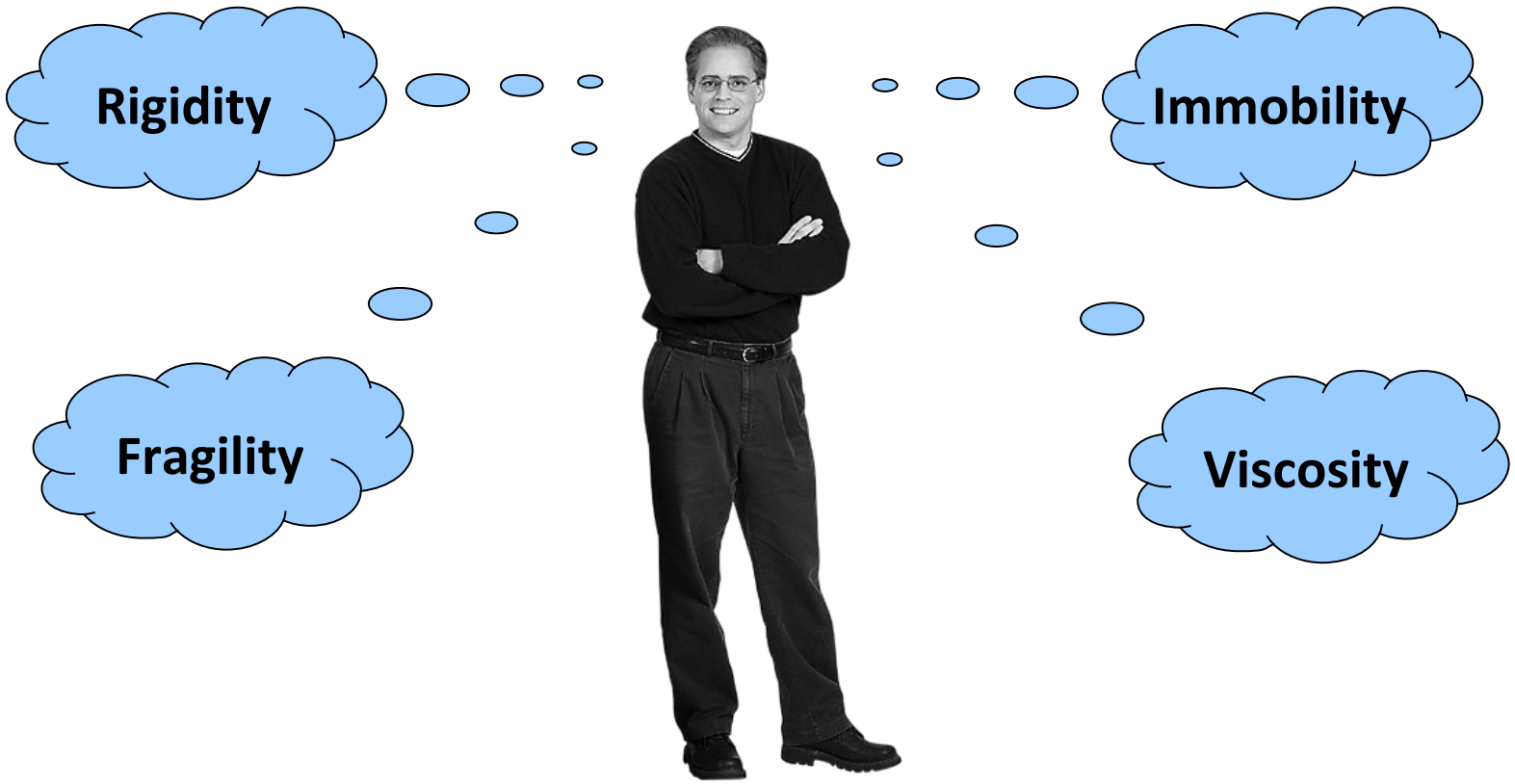
Design Smells

How do we know a Design is bad..?

Okay... I under stand the Importance of design

Are they any symptoms Of bad design...

Design Smells(contd…)



Rigidity

The design is hard to change

- Tendency for software to be difficult to change.
- Single change causes cascade of subsequent changes in dependent modules.
- The more module must be changed the more rigid the design.

Fragility

The design is easy to break

- Tendency of the software to break in many places every time it's changed .
- The breakage occurs in areas with no conceptual relationship
- On every fix the software breaks in unexpected way.

Immobility

Difficult to reuse

- Inability to reuse software from other projects or modules .
- The useful module have too many dependencies.
- Cost of rewriting is less compared to the risk faced to separate those part.

Viscosity

Hard to do the right thing

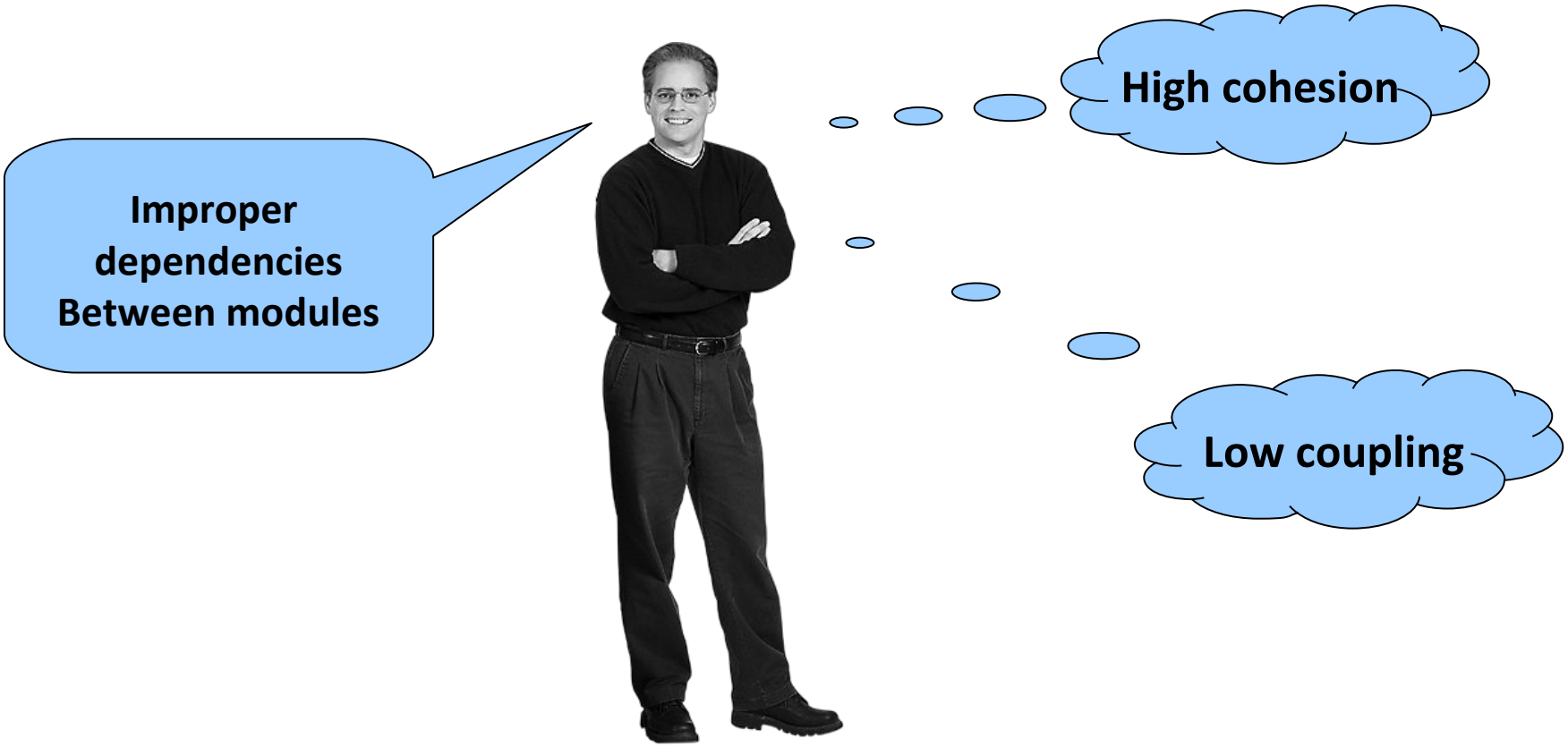
- It's easy to do the wrong thing, but hard to do the right thing.
- When the design preserving methods are more difficult to use than the hacks
- When dev environment is slow and inefficient developer will be tempted to do wrong things.

Design Characteristics

Is there any Characteristics For good design

Why design becomes rigid, Fragile Immobile and viscous

Design Characteristics



Design Characteristics

SRP

OCP

LSP

ISP

DIP

So how can we Achieve good Design..?

So let's go SOLID..

Single Responsibility Principle

This is hard to see, as we think Responsibility in group

- There should never be more then one reason to change a class.
- Many responsibility means many reason to change.

Benefits of SRP

1. Removes the **immobility Smell** from Design.
2. Deodorizes the **Rigidity Smell**

Achieved high cohesive design

Open Closed Principle

Abstraction is the key

- Software entities should be open for extension, but closed for modification.
- Keep the things that change frequently away from things that don't change.

Open Closed Principle (contd…)

Open for Extension

- Behavior of the module can be extended.

Closed for Modification

- The source of such a module is invisible.

OCP is the heart of object oriented design

- Make the module behave in new and different ways as the requirement of the application change.
- Resist making source change to it.

Further thinking of OCP

Is 100% Closer possible

Is there any techniques

- In Reality 100% Closer is not attainable.
- Closer must be Strategic.

Abstraction is the key

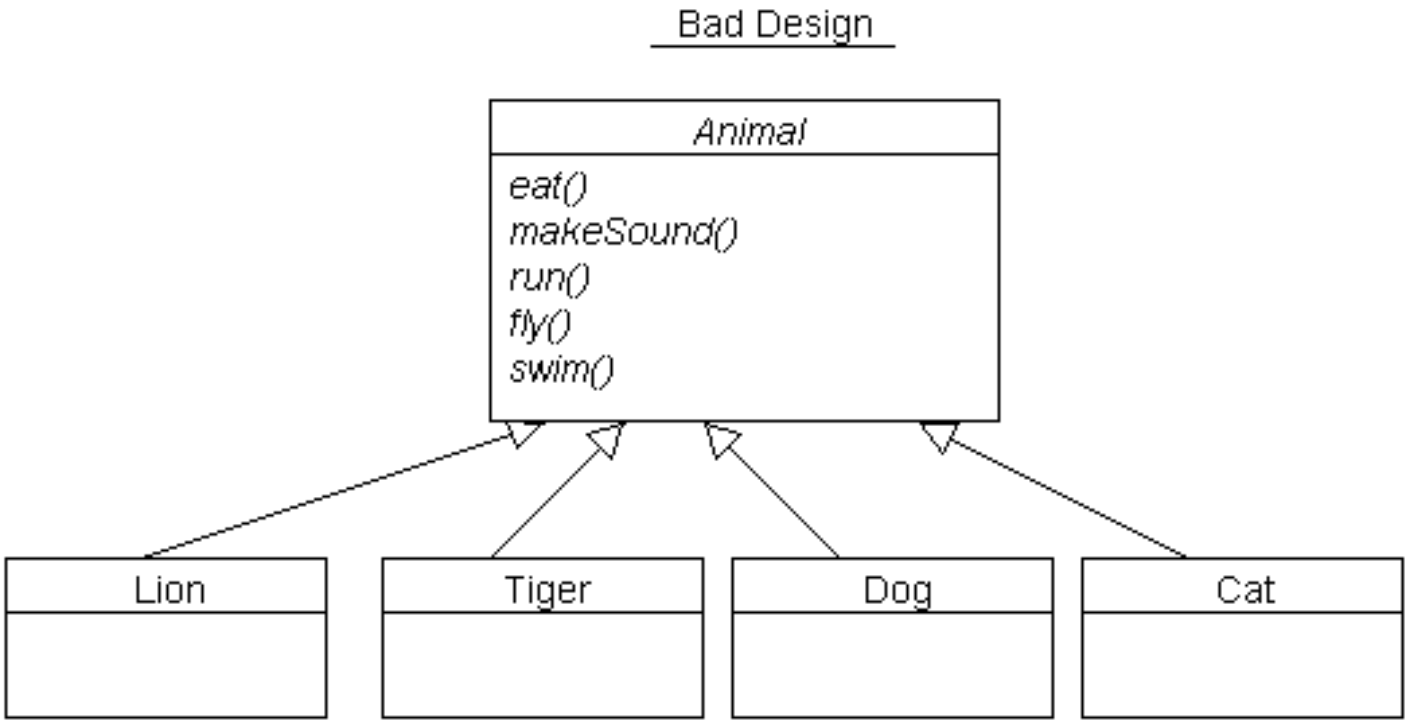
Flexibility, re usability and maintainability is The benefits

Liskov's SubstitutionPrinciple

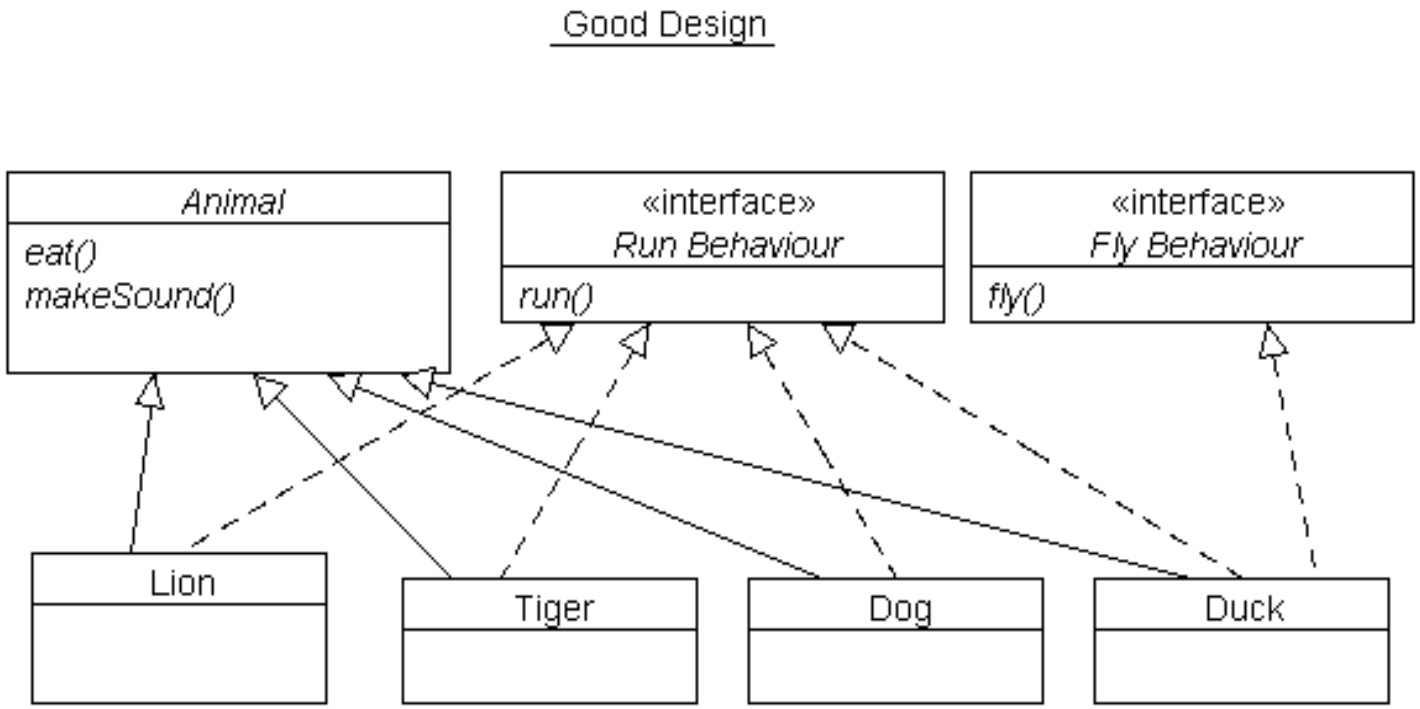
Subclass should be substitutable for their base class

- LSP reveals the problems with inheritance .
- It makes clear that in OO design IS-A relationship is about behavior; behavior that clients depend on.
- The LSP is all about well-designed inheritance
- Violating LSP makes code confusing.
- It's hard to understand the code that misuses inheritance

LSP Violation Example

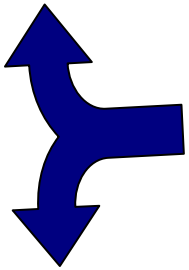


LSP Violation Example



Further thinking of LSP

- In order to be substitutable, the contract of the base class must be honored by the derived class.



Design by Contract

- Derived class substitutable for base class, if
  - Preconditions are no stronger than the base class
  - Preconditions are no weaker than the base class

Derived method should expect no more and provide no less

LSP Violation

The solution will likely to be put into an if else statement in the client side.

•Are you sure it works?
•Does lion fly?

Ah… may be we've allocate
The responsibility
Wrongly.

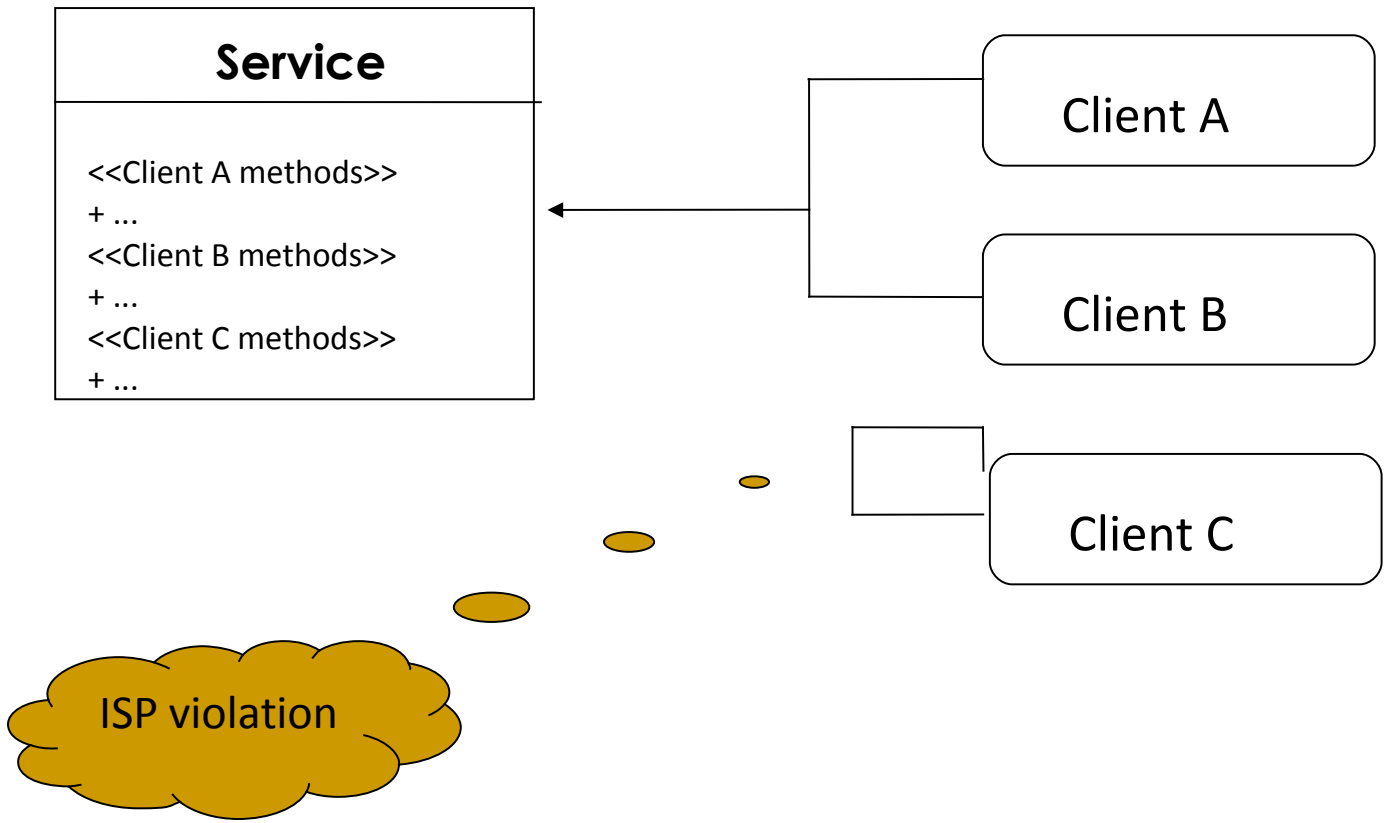
It violates LSP

Interface Segregation Principle

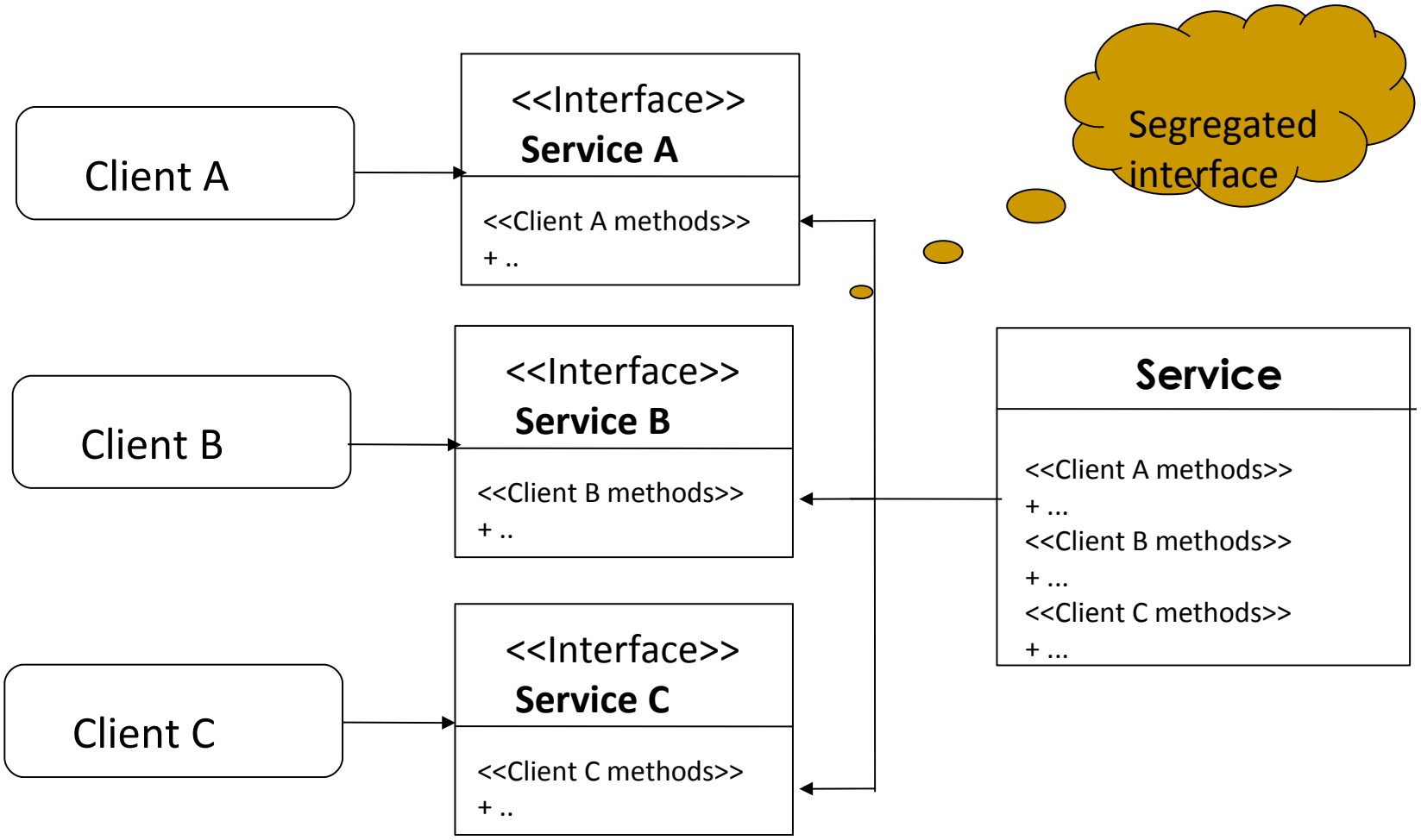
Many client specific interface are Better then one general purpose interface

- Client should not forced to depend on methods they do not use.
- ISP deals with designing cohesive interfaces and avoiding fat interfaces.
- What happen when the big class changes? All depending module must also change.

An violation of ISP example



An violation of ISP example: Solution

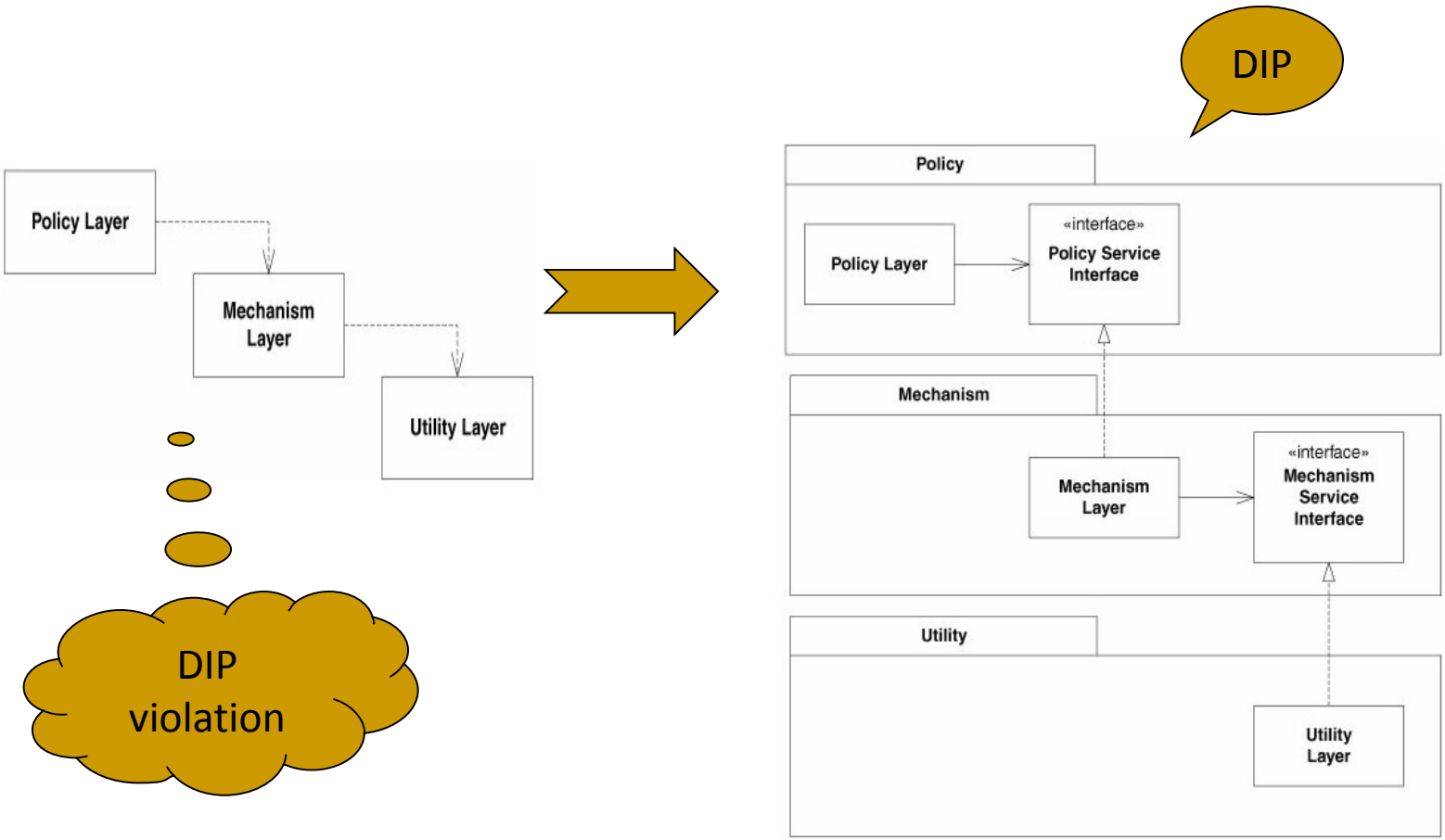


Dependency Inversion Principle

Depend upon abstraction. Do not depend Upon concretions

- High level module should not depend upon low level modules, both should depends upon abstraction
- Abstraction should not depends upon on details, details should depends on abstraction.

A DIP example



DIP Summary

- Inversion of dependencies is the hallmark of good object oriented design.
- If it's dependencies are inverted, it has an OO design If it's dependencies are not inverted it has a procedural design.
- Dependency injection is the core of the famous spring framework

Hollywood principle: “ Don't call us, we'll call you”.

Summary

SRP	• A class should have only one reason to change
OCP	• A Module should be open for extension but closed for modification.
LSP	• Subclass should be substitutable for their base class.
ISP	• Many client specific interfaces are better then one general purpose interface .
DIP	• Depends upon abstraction. Do not depends upon concretions.

What’s Next?

15 min

After the break:

GoF Patterns Revisited