

OO Design Principles

Java Basic



Based On Java 21

Contents



- What is OO Design Principle
- SOLID Principle
 - Single Responsibility Principle
 - Open / Close Principle
 - Liskovs' Substitution Principle
 - Interface Segregation Principle
 - Dependency Inversion Principle

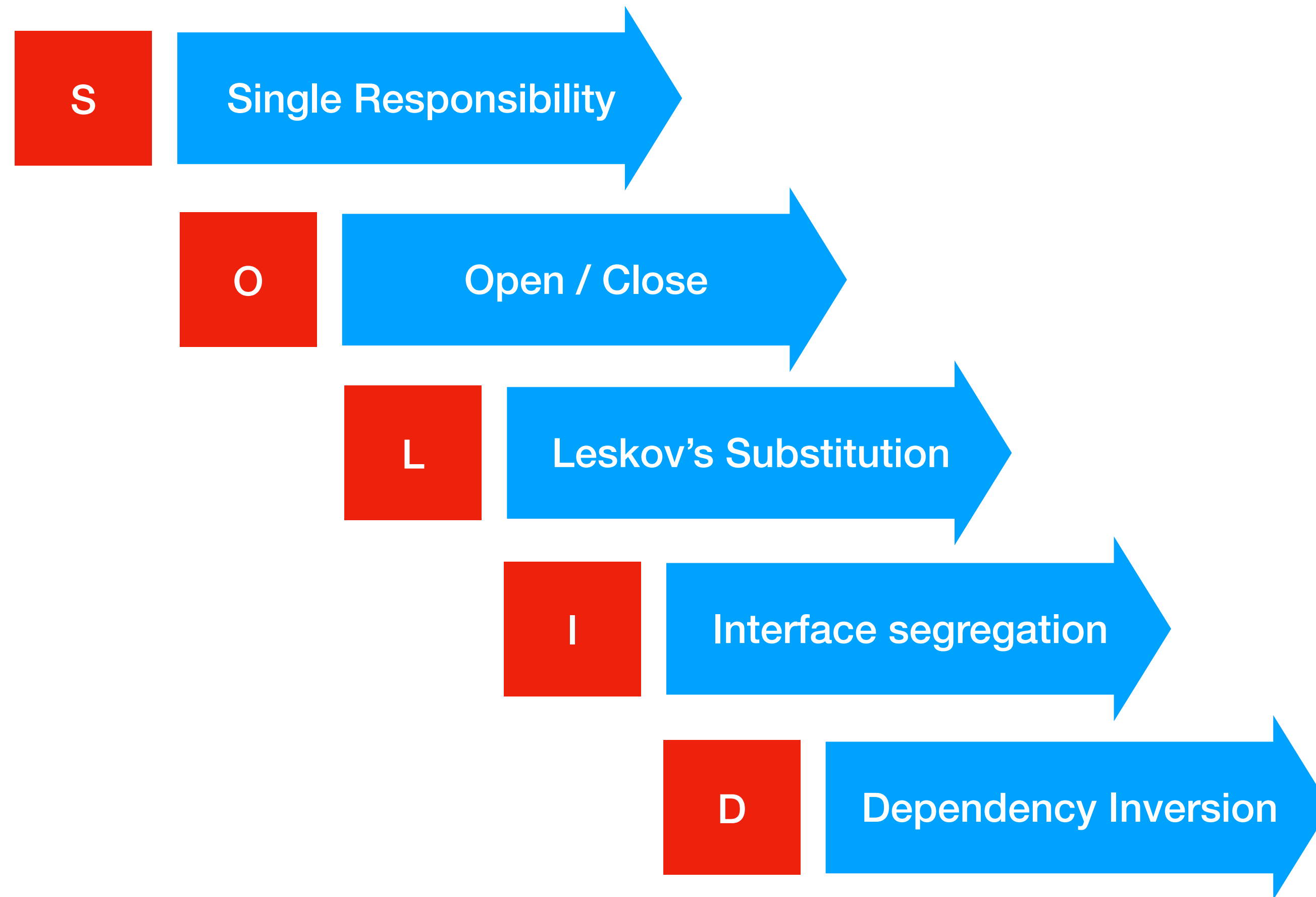
What is OO Design Principle

- OOP ကတော့ Object တွေမှာ ဒါတွေလုပ်လို့ရတယ်ဆိုတာကို ဖော်ပြထားပါတယ်
- OOP ကို အသုံးပြုခြင်းအားဖြင့် Code Reuse ကို ရရှိစေပြီး Developing ကို မြန်ဆန်စေနိုင်ပါတယ်
- Object Design Principle ဆိုတာကတော့ Object တွေကို အသုံးပြုပြီး Software တွေကို ရေးသားတဲ့ အခါမှာ လိုက်နာသင့်တဲ့ အချက်တွေပဲ ဖြစ်ပါတယ်
- Robert C Martin ရေးသားထားတဲ့ “Agile Software Development, Principles, Patterns, and Practices.” (2002) စာအုပ်ထဲမှာ OOP ကို အသုံးပြုပြီး ရေးသားတဲ့ အခါမှာ လိုက်နာသင့်တဲ့ စီးမျဉ်း စည်းကမ်းများကို စုစည်းဖော်ပြထားခဲ့ပါတယ်

What is a good design?

- Robert C Martin က “Good Design is not a bad design” လို့ ပြောပါတယ်
- မကောင်းတာတွေကို သိမှ ကောင်းအောင် ပြင်ရေးနိုင်မှာ ဖြစ်ပါတယ်
- Rigidity
Change effect other parts
- Fragility
Change breaks other parts
- Immobility
Can't move means can't reuse

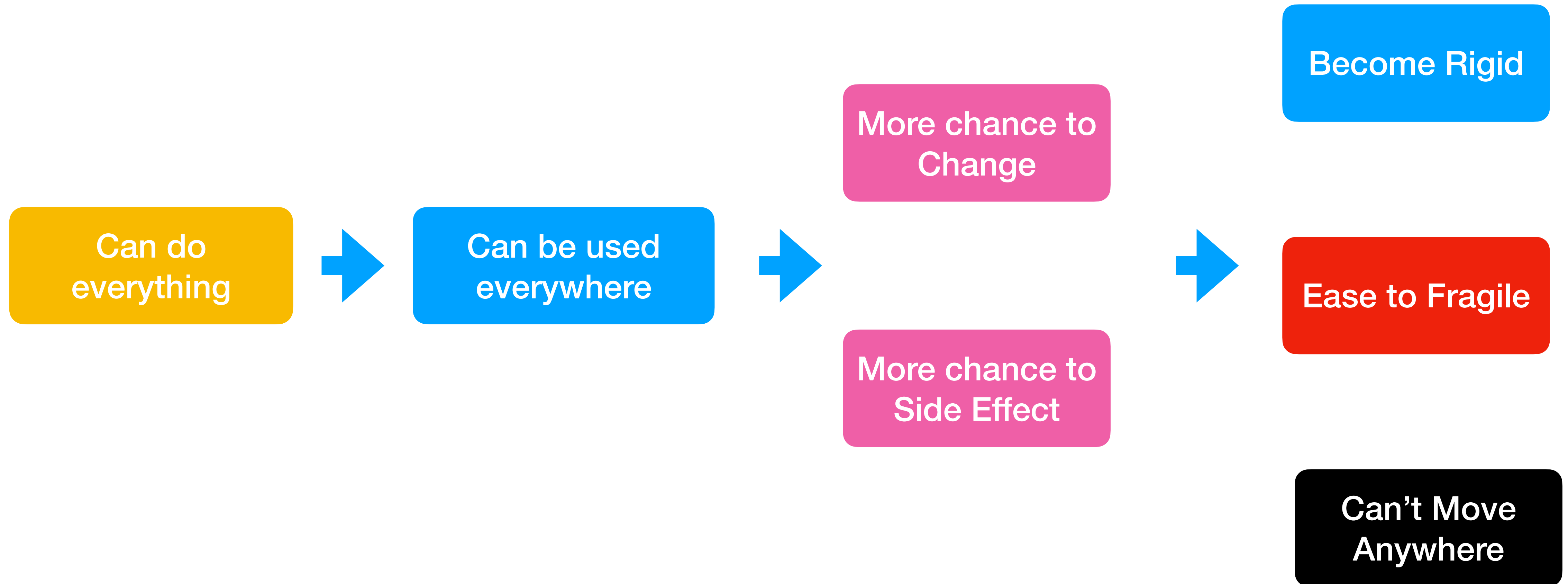
SOLID Principles



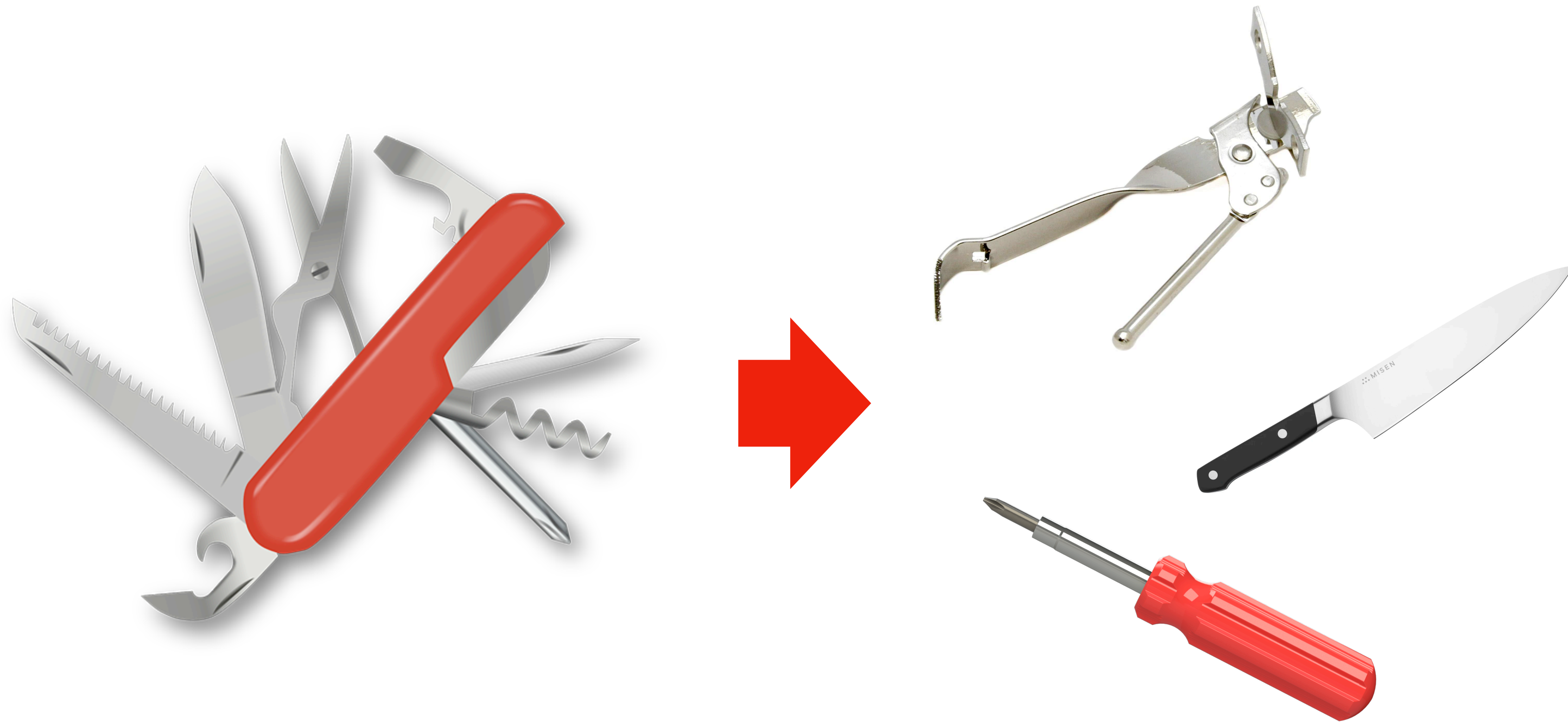
Single Responsibility

A class should have one and only one reason to change, meaning that a class should have only one job.

Don't try everything alone



Single Responsibility

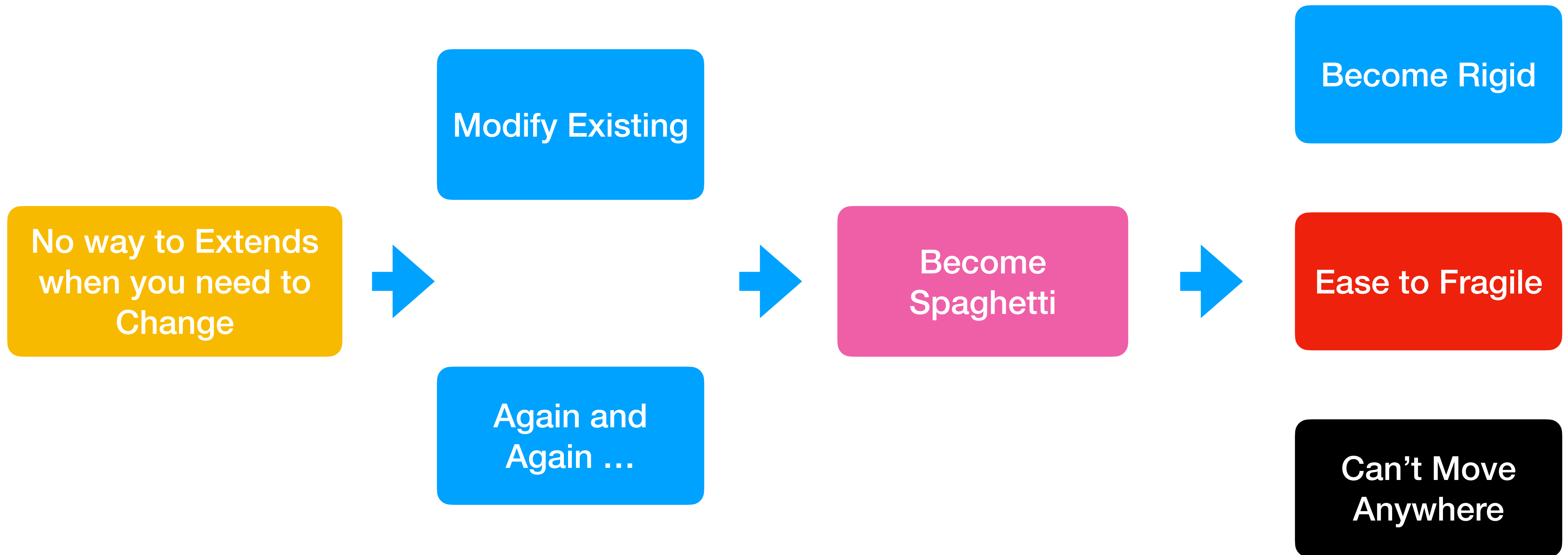


- Specification leads cohesiveness, less Side Effects & Higher Reusability

Open / Close Principle

Software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification.

Don't try to modify again and again



Extension over Modification



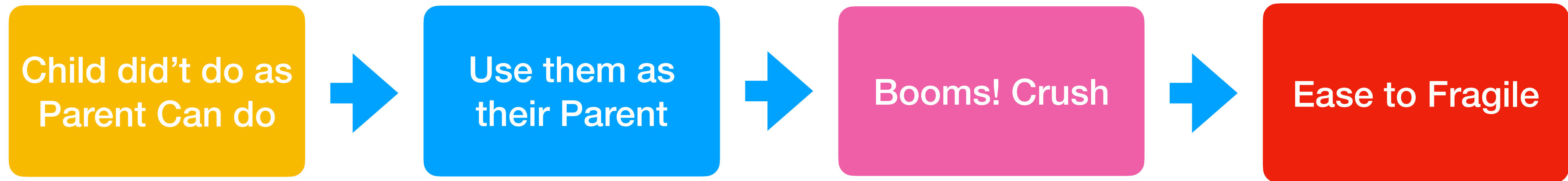
Reduce Complexity, rigidity and fragility!

Liskov Substitution Principle

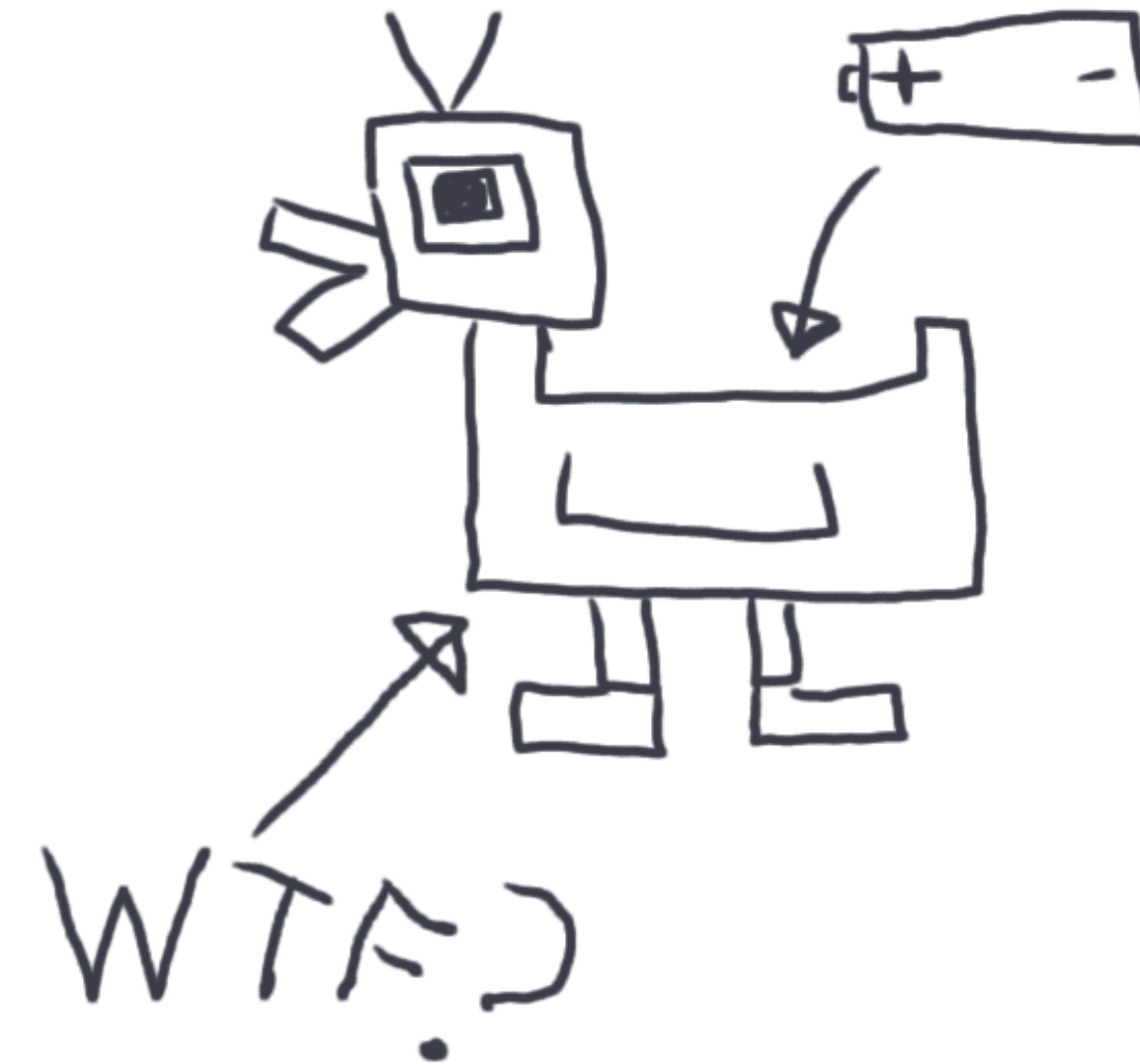
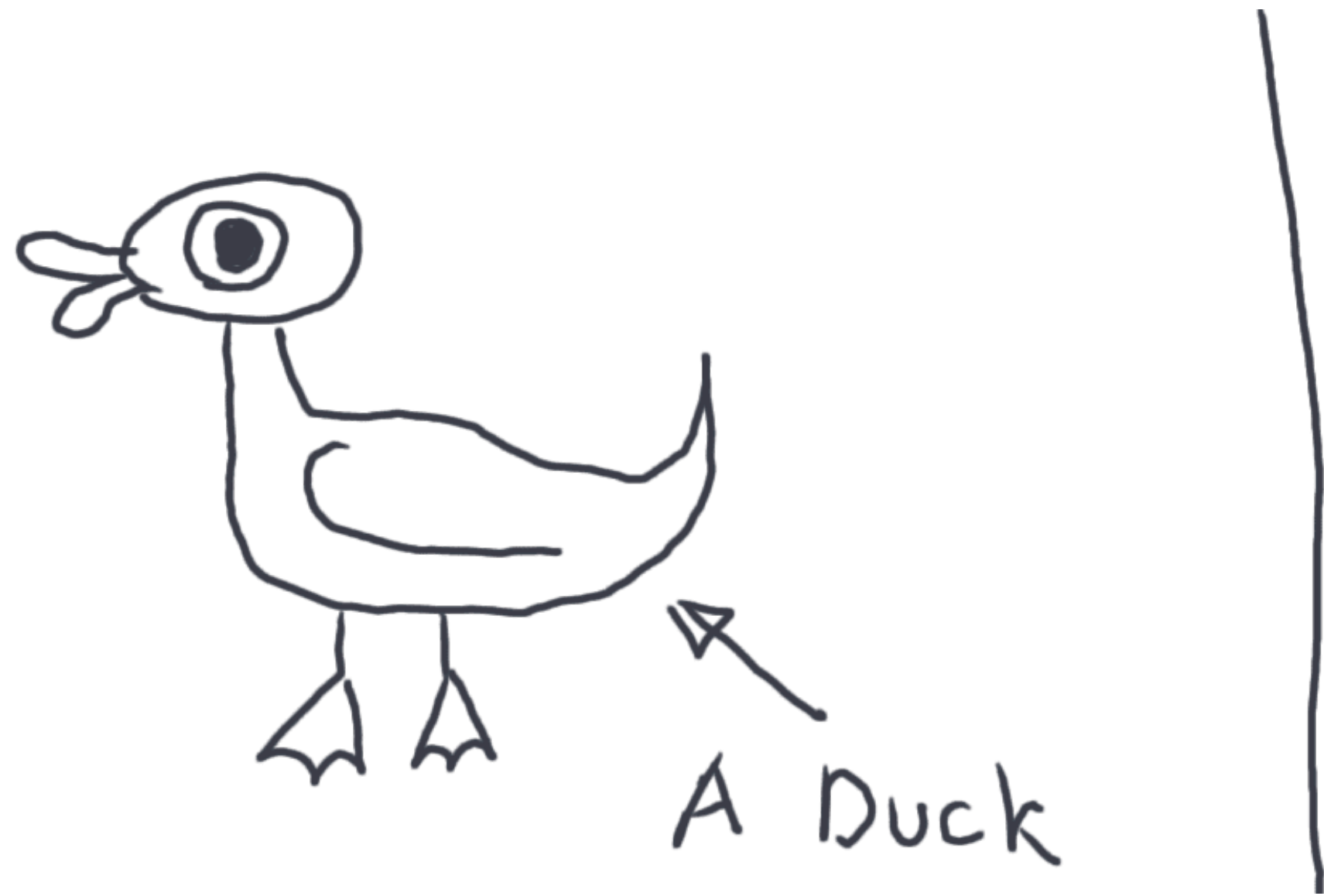
*Let $q(x)$ be a property provable
about objects x of type T .*

*Then $q(y)$ should be provable for
objects y of type S where S is a
subtype of T .*

Child should be obey what the parent say!



Child of Duck is also a duck!

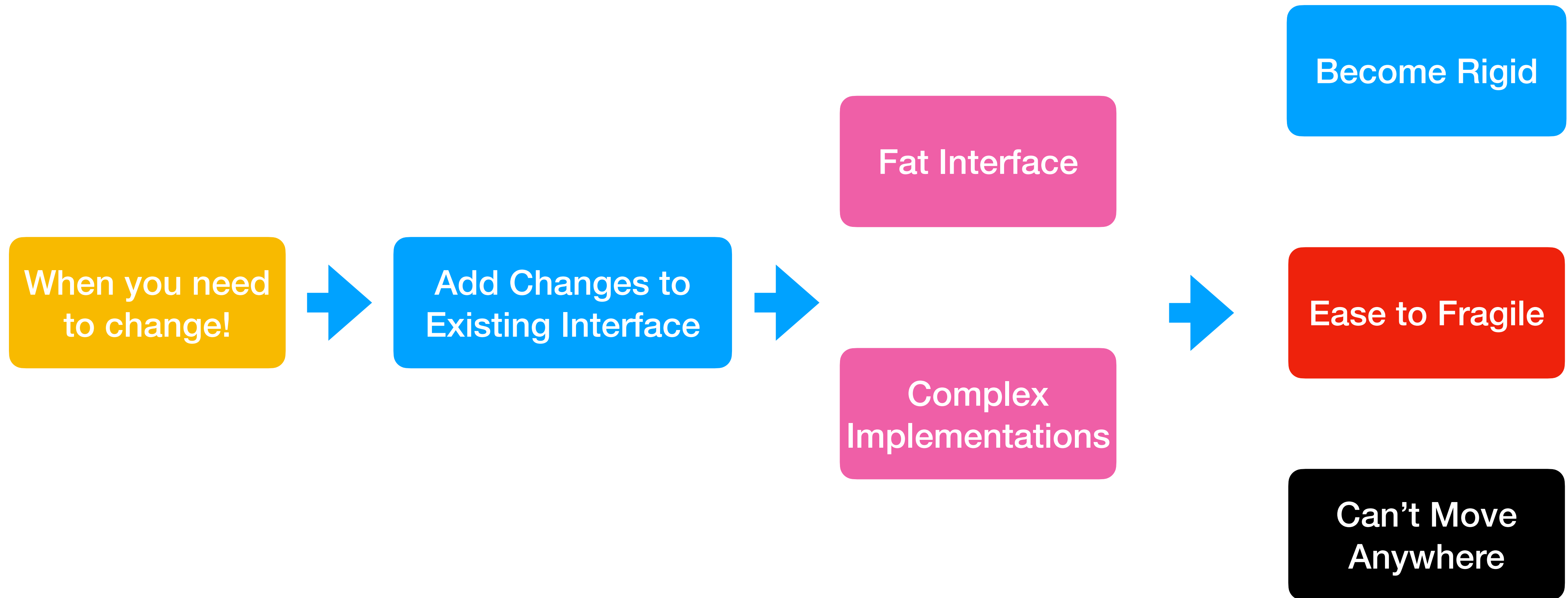


Terminate Strange things is forbidding Mistake

Interface Segregation Principle

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

Avoid fatty interfaces



Segregate interfaces



rawpixel

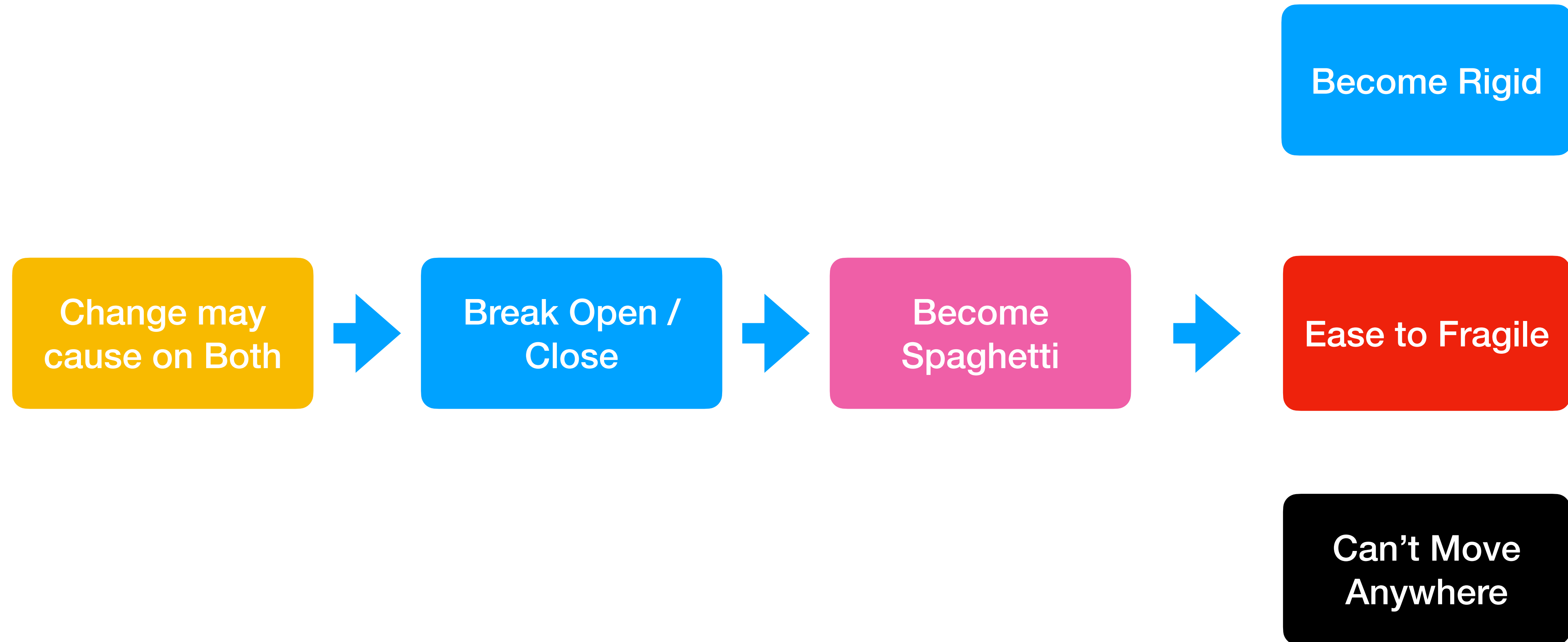
Limit functions reduce Complexity And promote Reusability

Dependency Inversion Principle

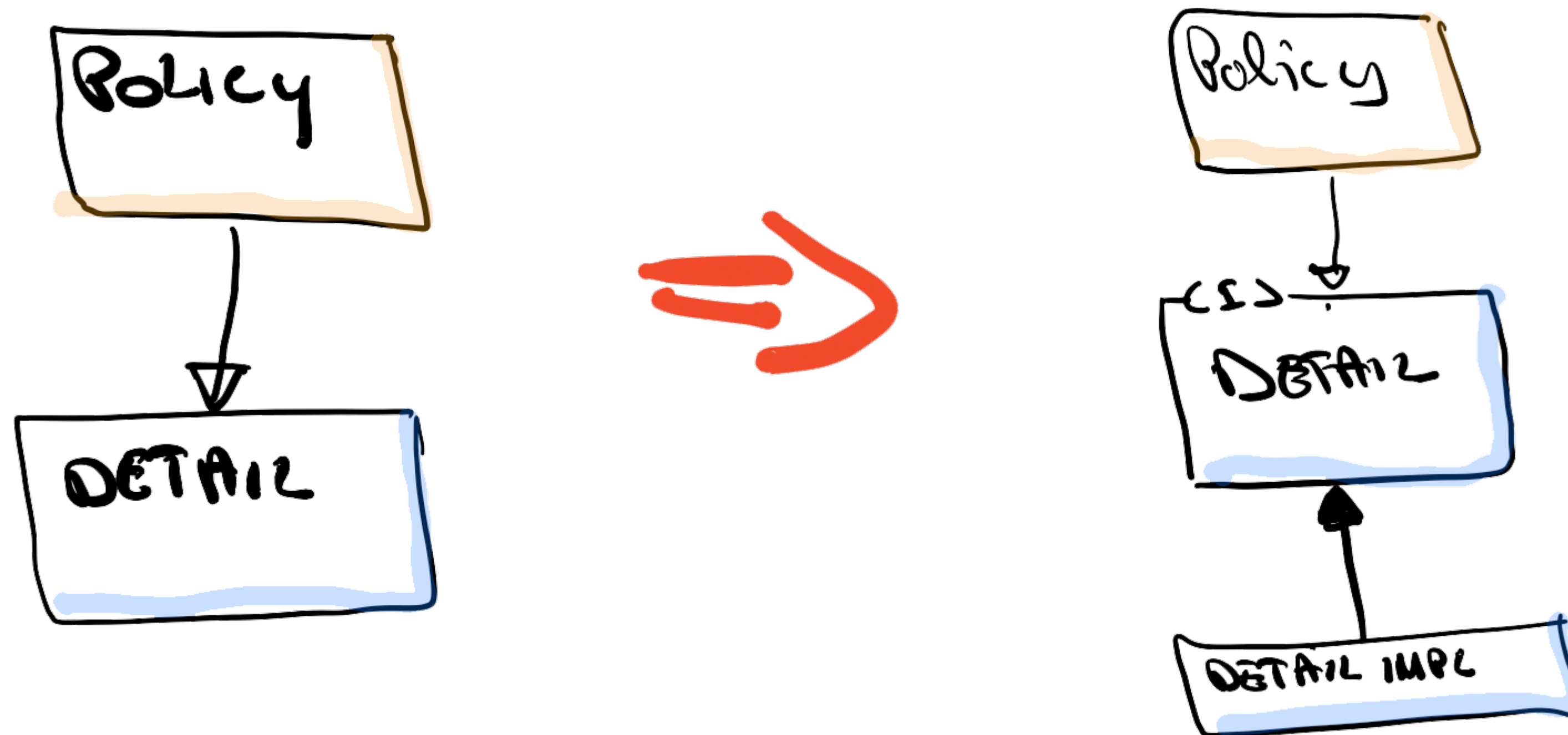
High level modules should not depend on low level modules; both should depend on abstractions.

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

When what depends on how!



Inverse Dependencies



By introducing Abstraction between What & How, It reduces coupling between them