LOSE GEKOPPELT WIE NIE

DI VS. IOC

Saxonia Systems
So geht Software.

Der Sprecher

Hendrik Lösch

Senior Consultant & Coach

Hendrik.Loesch@saxsys.de

@HerrLoesch

Just-About.Net

WPF-Anwendungen mit MVVM und Prism

Modulare Architekturen verstehen und umsetzen

Windows 8 Store Apps mit MVVM und Prism

XAML-Entwurfsmuster, Bootstrapping, Navigation, Messaging

Test Driven Development – Praxisworkshop

Business-Applikationen testgetrieben entwickeln

Inversion of Control und Dependency Injection

Prinzipien der modernen Software-Architektur ...

Test Driven Development mit C#

Grundlagen, Frameworks, best Practices

Automatisiertes Testen mit Visual Studio 2012

Grundlagen, Testarten und Strategien

Saxonia Systems
So geht Software.

WAS MACHT SCHLECHTES DESIGN AUS?

1. It is hard to change because every change affects too many other parts of the system. (Rigidity)

2. When you make a change, unexpected parts of the system break. (Fragility)

3. It is hard to reuse in another application because it cannot be disentangled from the current application. (Immobility)

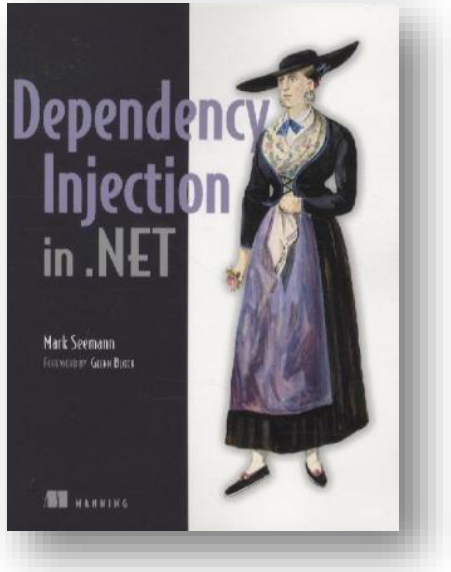
Quelle: http://www.objectmentor.com/resources/articles/dip.pdf

ARTEN VON ABHÄNGIGKEITEN

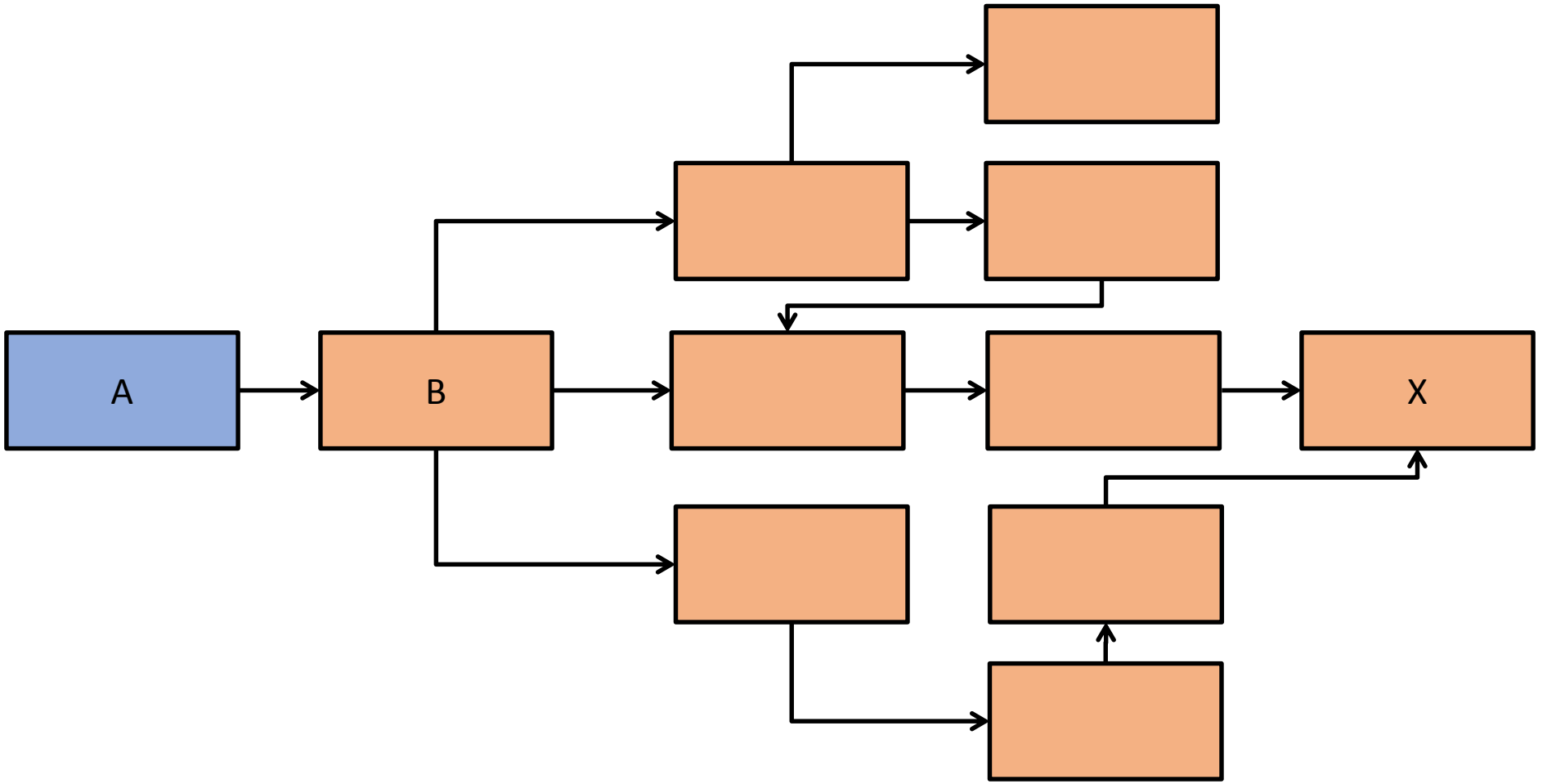
beständig

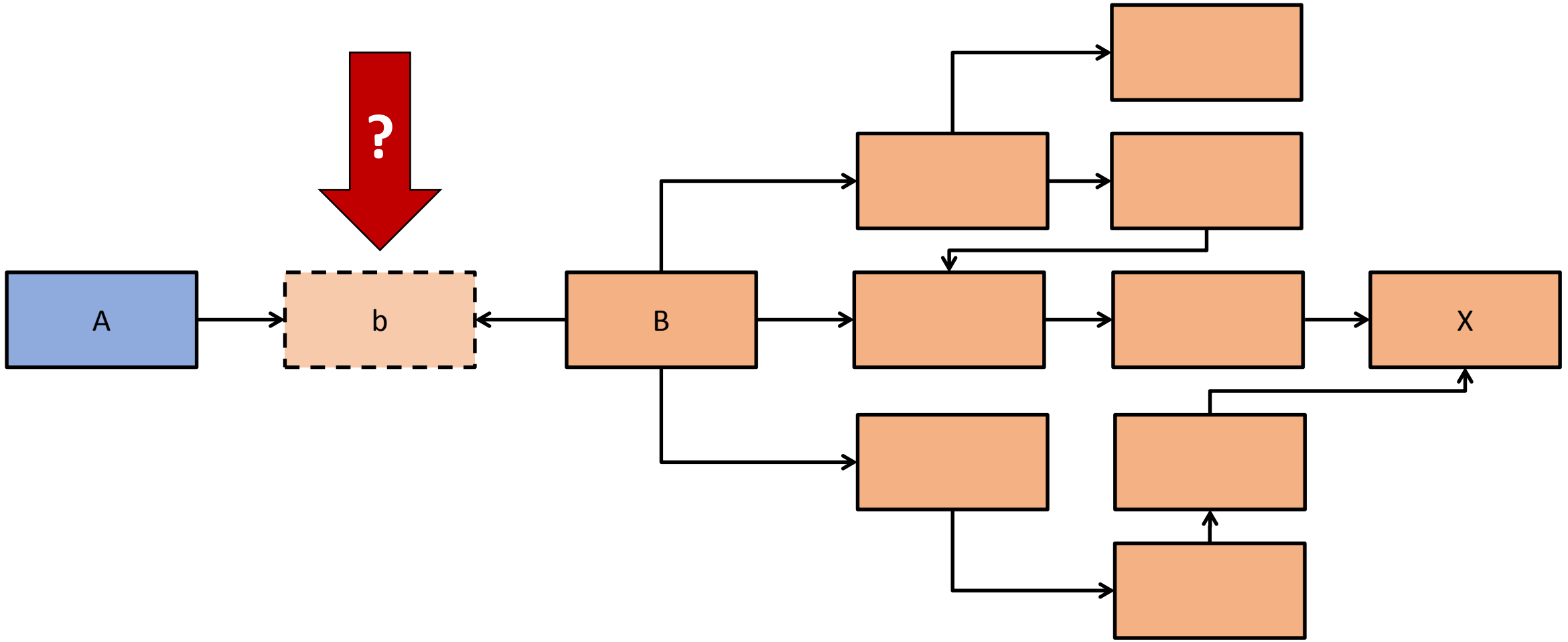
unbeständig

- Basistechnologien (.Net, Java, Ruby, …)
- Basisdatentypen (int, string, char…)
- Datenhaltungsklassen und Transferobjekte
- Domänen spezifische Algorithmen und Datenbanken
- UI Logik
- …



Saxonia Systems
So geht Software.


A
B
X


?
A
b
B
X

```
public class A
{
    IB b;

    public A(IB b)
    {
        this.b = b;
    }
 }
```

Saxonia Systems
So geht Software.

> “The SOLID principles are not rules. They are not laws. They are not perfect truths. They are statements on the order of *“An apple a day keeps the doctor away.”* This is a good principle, it is good advice, but it’s not a pure truth, nor is it a rule.”

Robert C. Martin
(Uncle Bob)

Quelle: https://sites.google.com/site/unclebobconsultingllc/getting-a-solid-start

S ingle Responsibility Principle

O pen Closed Principle

L iskov Substitution Principle

I nterface Segregation Principle

D ependency Inversion Principle

S ingle Responsibility Principle

O pen Closed Principle

L iskov Substitution Principle

I nterface Segregation Principle

D ependency Inversion Principle

Eine Klasse sollte nur eine Verantwortlichkeit haben.

Quelle: http://www.clean-code-developer.de/

S ingle Responsibility Principle

O pen Closed Principle

L iskov Substitution Principle

I nterface Segregation Principle

D ependency Inversion Principle

Eine Klasse sollte offen für Erweiterungen, jedoch geschlossen für Modifikationen sein.

Quelle: http://www.clean-code-developer.de/

Saxonia Systems
So geht Software.

S ingle Responsibility Principle

O pen Closed Principle

L iskov Substitution Principle

I nterface Segregation Principle

D ependency Inversion Principle

Abgeleitete Klassen sollten sich so verhalten wie es von ihren Basistypen erwartet wird.

Quelle: http://www.clean-code-developer.de/

Saxonia Systems
So geht Software.

S ingle Responsibility Principle

O pen Closed Principle

L iskov Substitution Principle

I nterface Segregation Principle

D ependency Inversion Principle

Interfaces sollten nur die Funktionalität wiederspiegeln die ihre Klienten erwarten.

Quelle: http://www.clean-code-developer.de/

Saxonia Systems
So geht Software.

S ingle Responsibility Principle

O pen Closed Principle

L iskov Substitution Principle

I nterface Segregation Principle

D **ependency Inversion Principle**

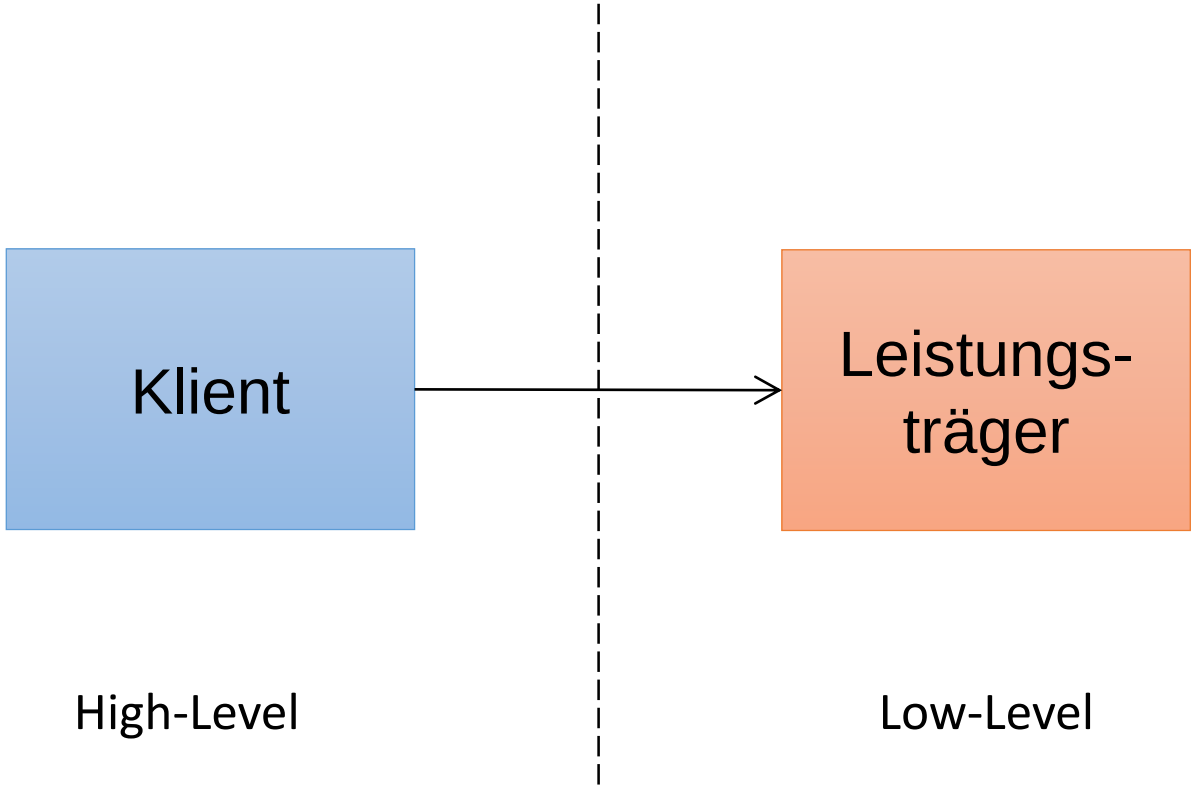
High-Level Klassen sollen nicht von Low-Level Klassen abhängig sein, sondern beide von Abstraktionen.
Abstraktionen sollen nicht von Details abhängig sein, sondern Details von Abstraktionen.

Quelle: Wikipedia.org

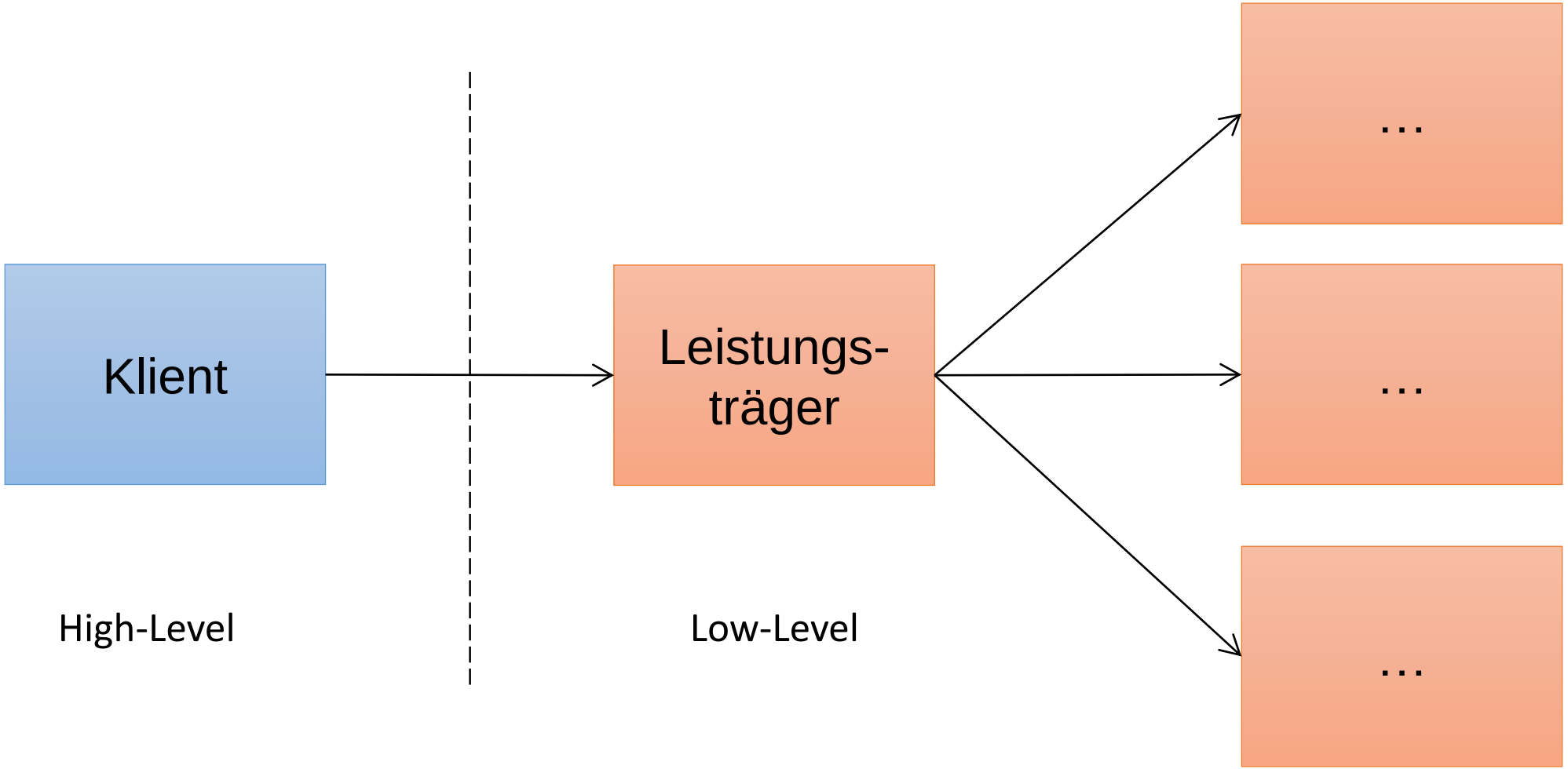
High-Level Klassen sollen nicht von Low-Level Klassen abhängig sein, sondern beide von Abstraktionen.

Abstraktionen sollen nicht von Details abhängig sein, sondern Details von Interfaces.

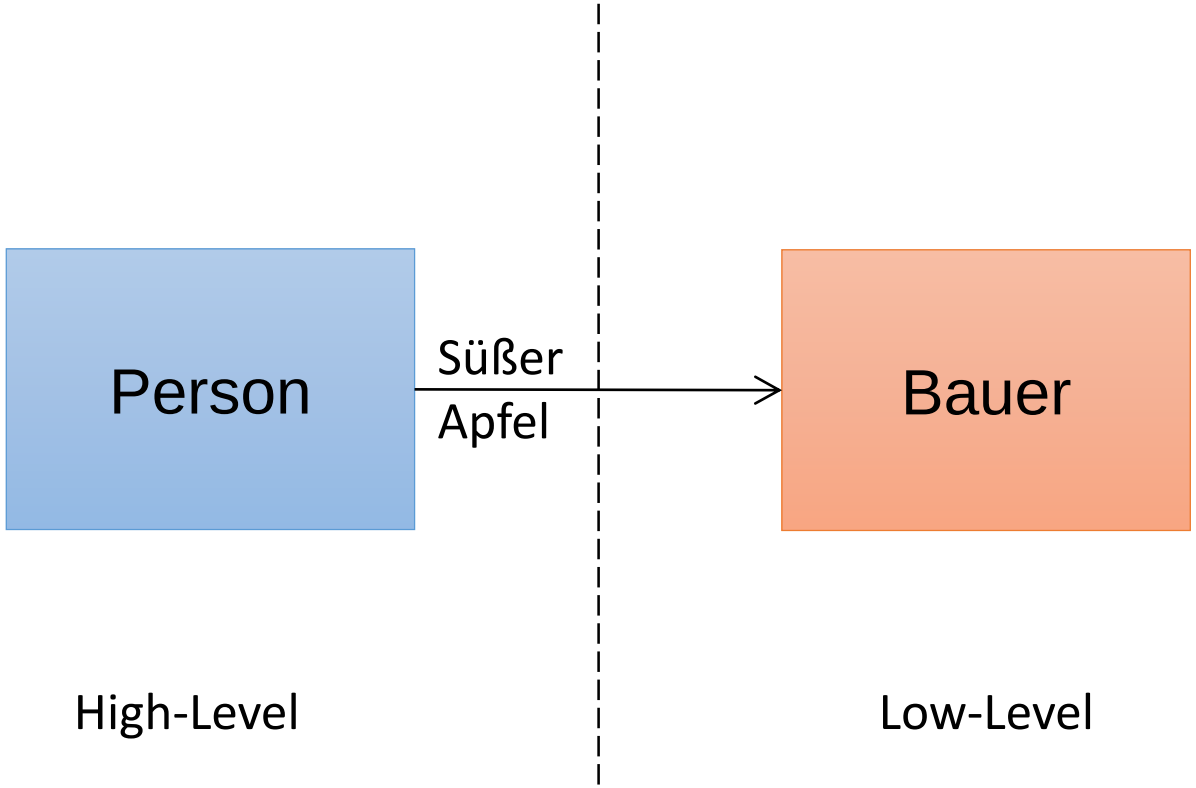
High-Level Klassen sollen nicht von Low-Level Klassen abhängig sein…



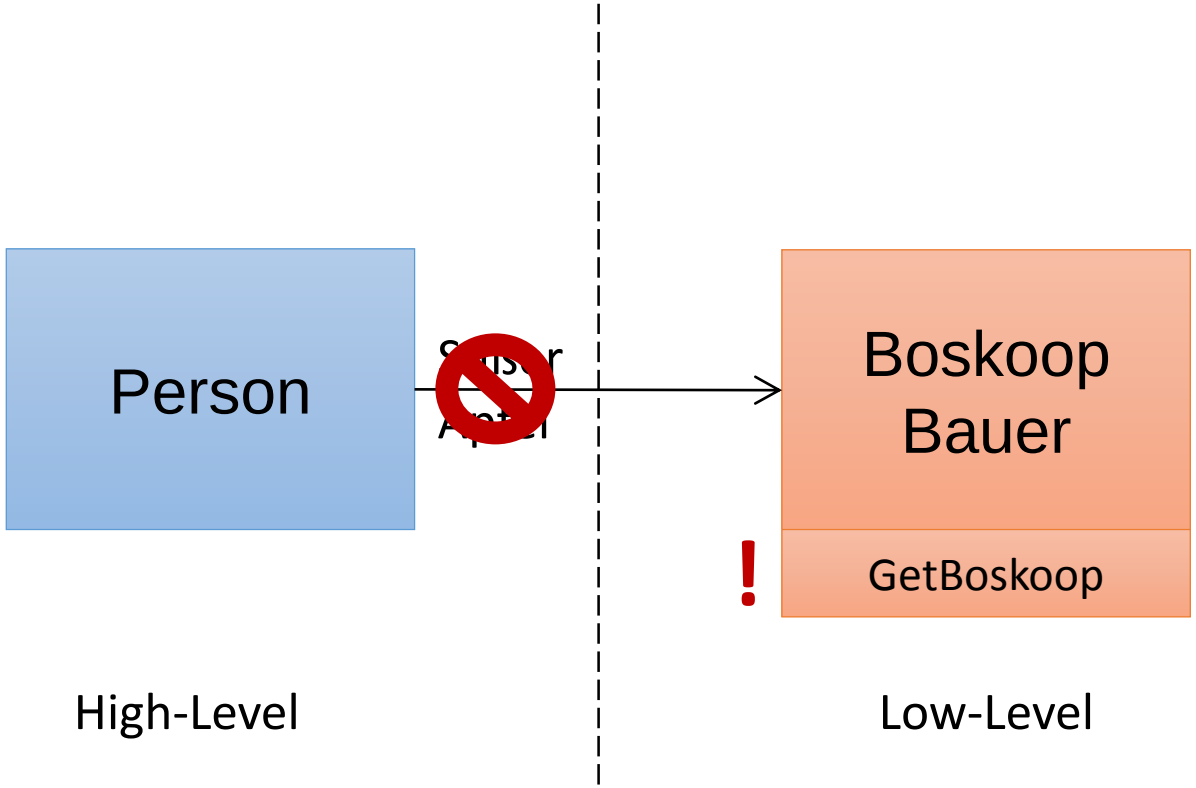
High-Level Klassen sollen nicht von Low-Level Klassen abhängig sein…



High-Level Klassen sollen nicht von Low-Level Klassen abhängig sein…



High-Level Klassen sollen nicht von Low-Level Klassen abhängig sein…



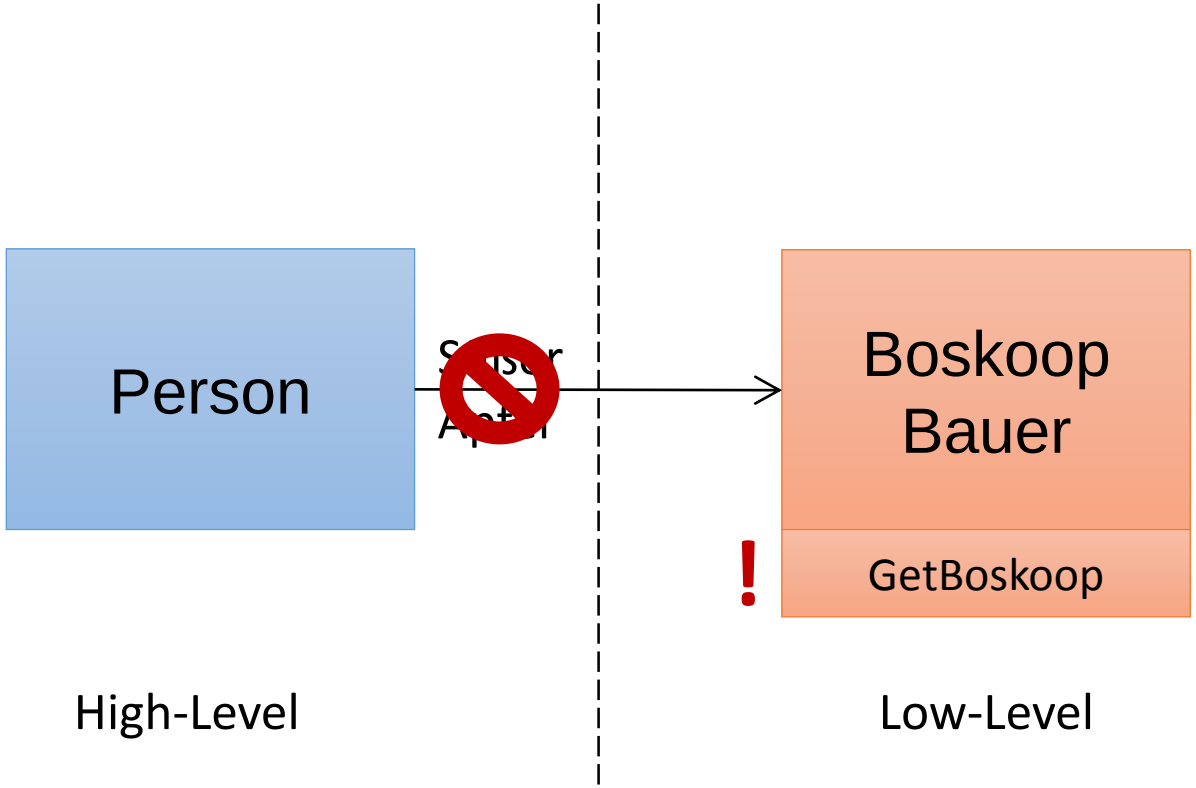
High-Level Klassen sollen nicht von Low-Level Klassen abhängig sein, sondern beide von Abstraktionen.

Abstraktionen sollen nicht von Details abhängig sein, sondern Details von Interfaces.

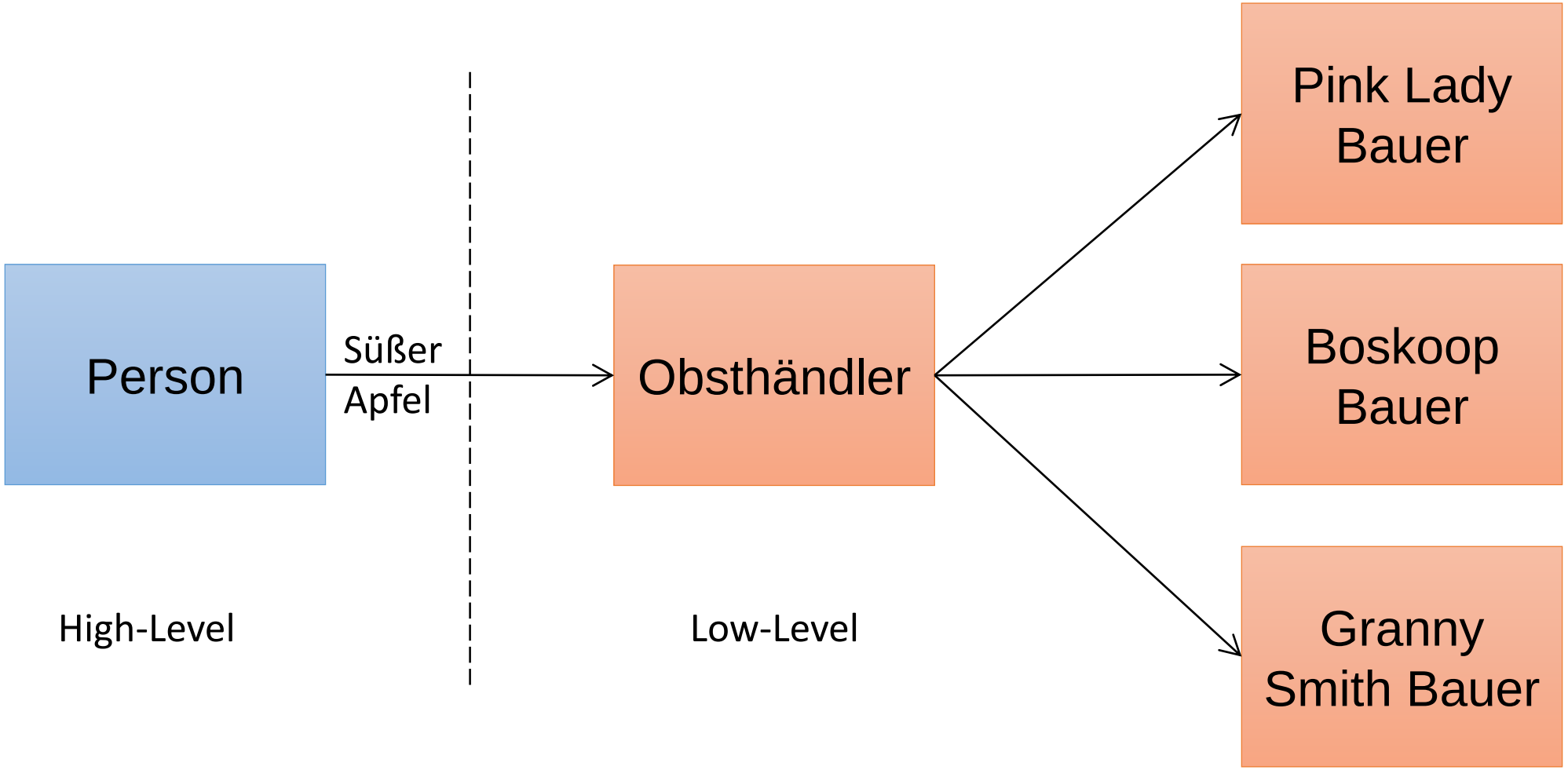
High-Level Klassen sollen nicht von Low-Level Klassen abhängig sein, sondern beide von Abstraktionen.

Abstraktionen sollen nicht von Details abhängig sein, sondern Details von Interfaces.

High-Level Klassen sollen nicht von Low-Level Klassen abhängig sein, sondern beide von Abstraktionen…



High-Level Klassen sollen nicht von Low-Level Klassen abhängig sein, sondern beide von Abstraktionen…



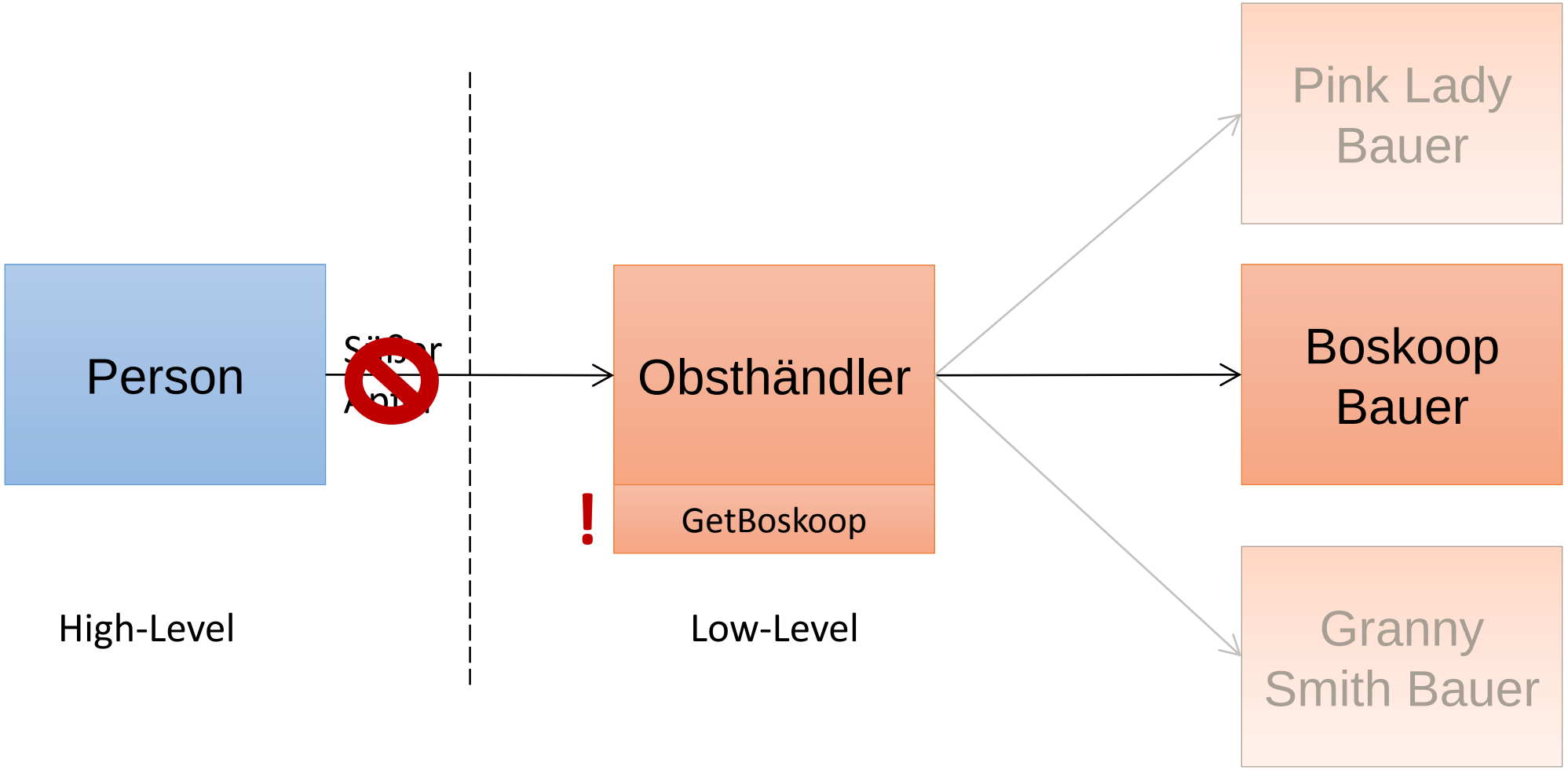
High-Level Klassen sollen nicht von Low-Level Klassen abhängig sein, sondern beide von Abstraktionen.

Abstraktionen sollen nicht von Details abhängig sein, sondern Details von Abstraktionen.

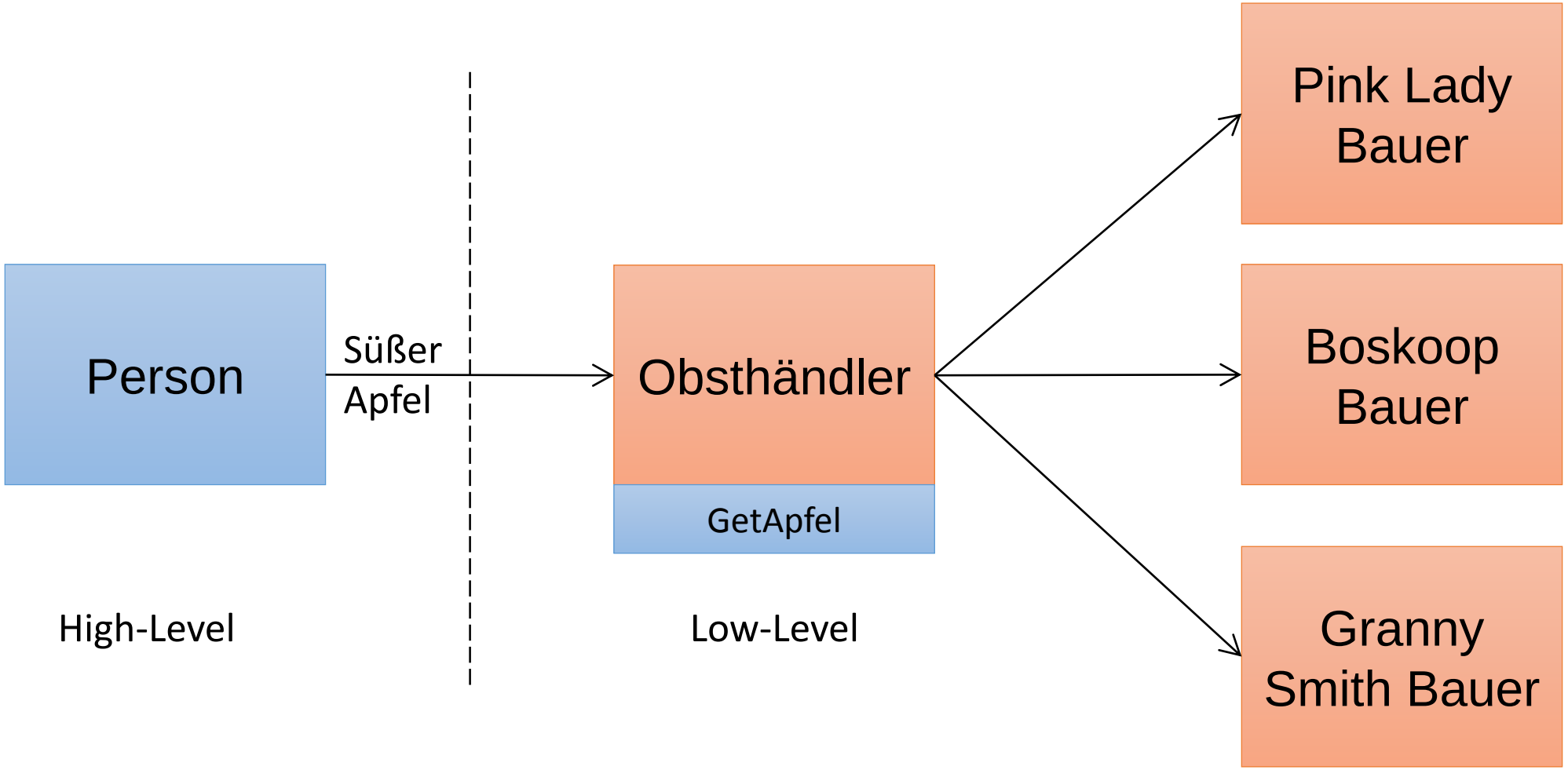
High-Level Klassen sollen nicht von Low-Level Klassen abhängig sein, sondern beide von Abstraktionen.

Abstraktionen sollen nicht von Details abhängig sein, sondern Details von Abstraktionen.

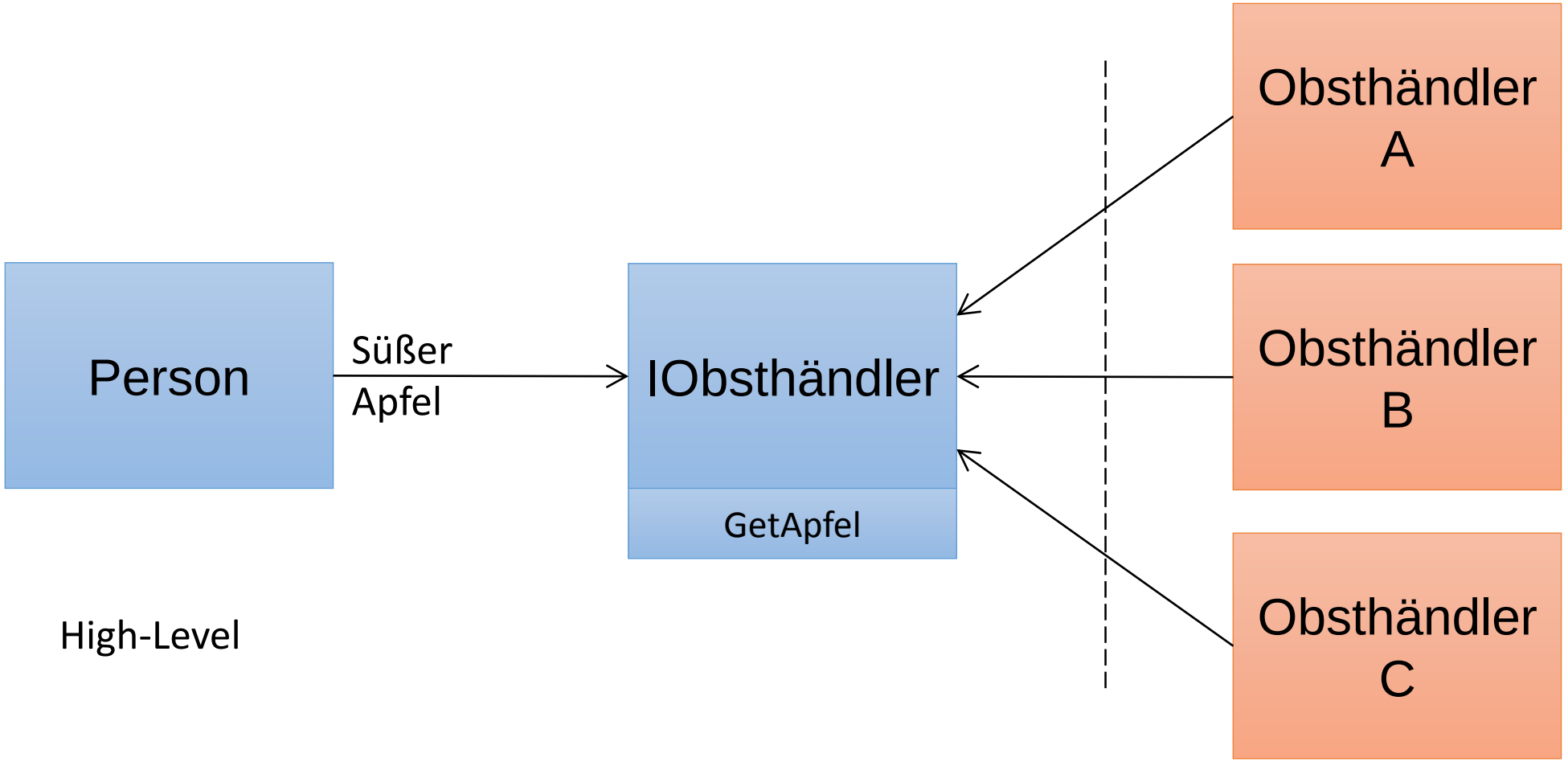
Abstraktionen sollen nicht von Details abhängig sein, sondern Details von Abstraktionen.



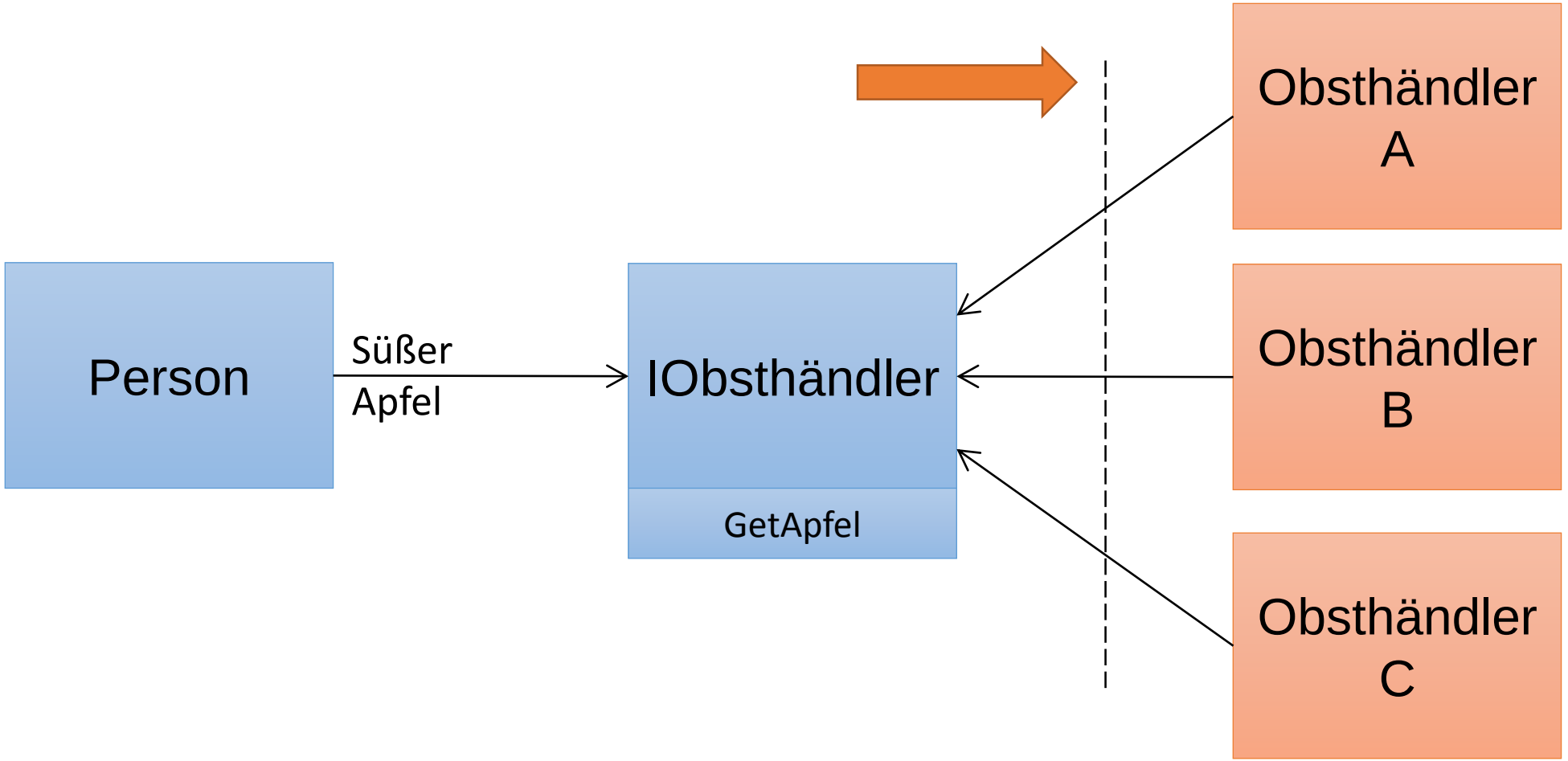
Abstraktionen sollen nicht von Details abhängig sein, sondern Details von Abstraktionen.



Abstraktionen sollen nicht von Details abhängig sein, sondern Details von Abstraktionen.



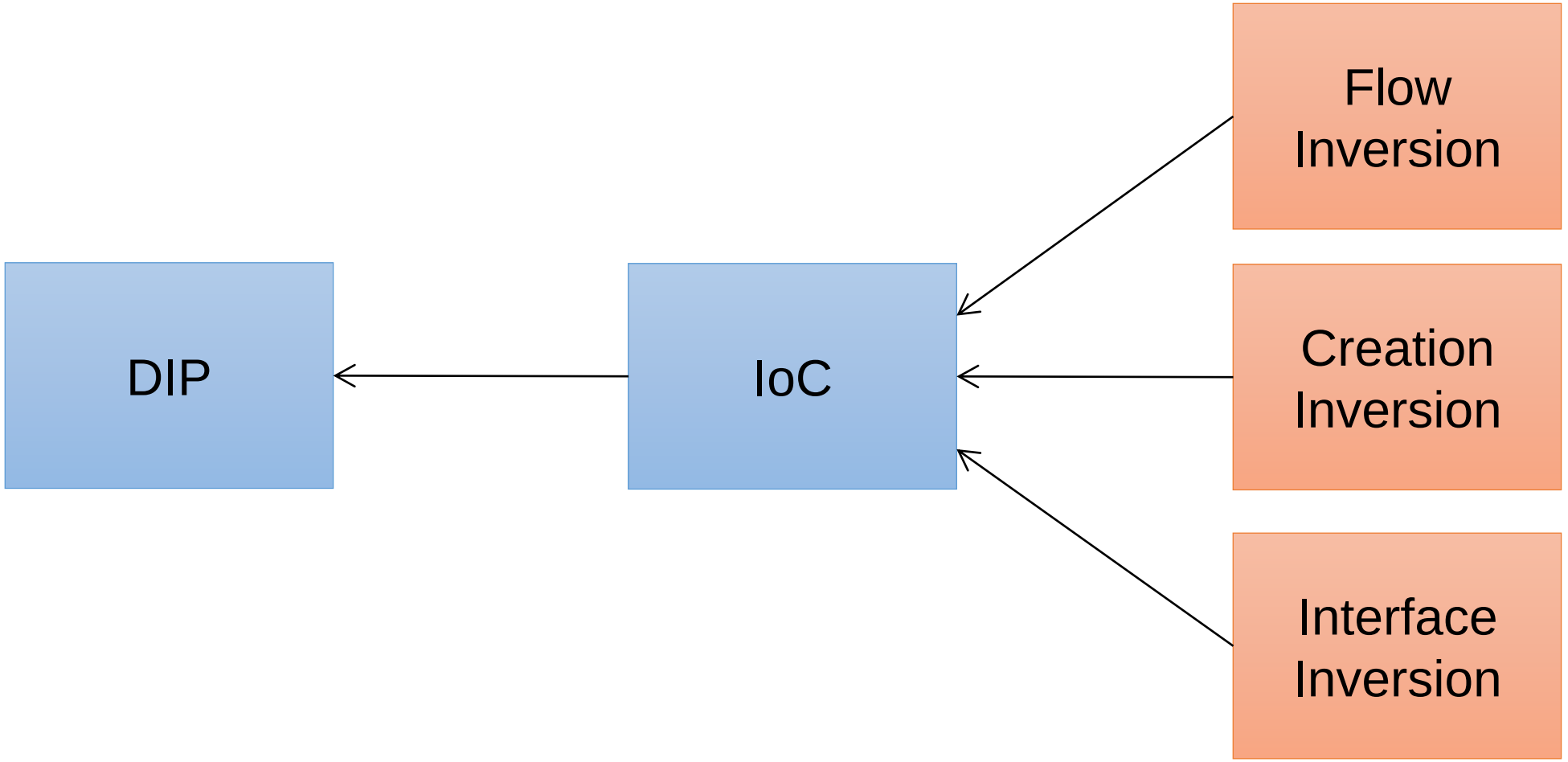
Was ist Inversion of Control???

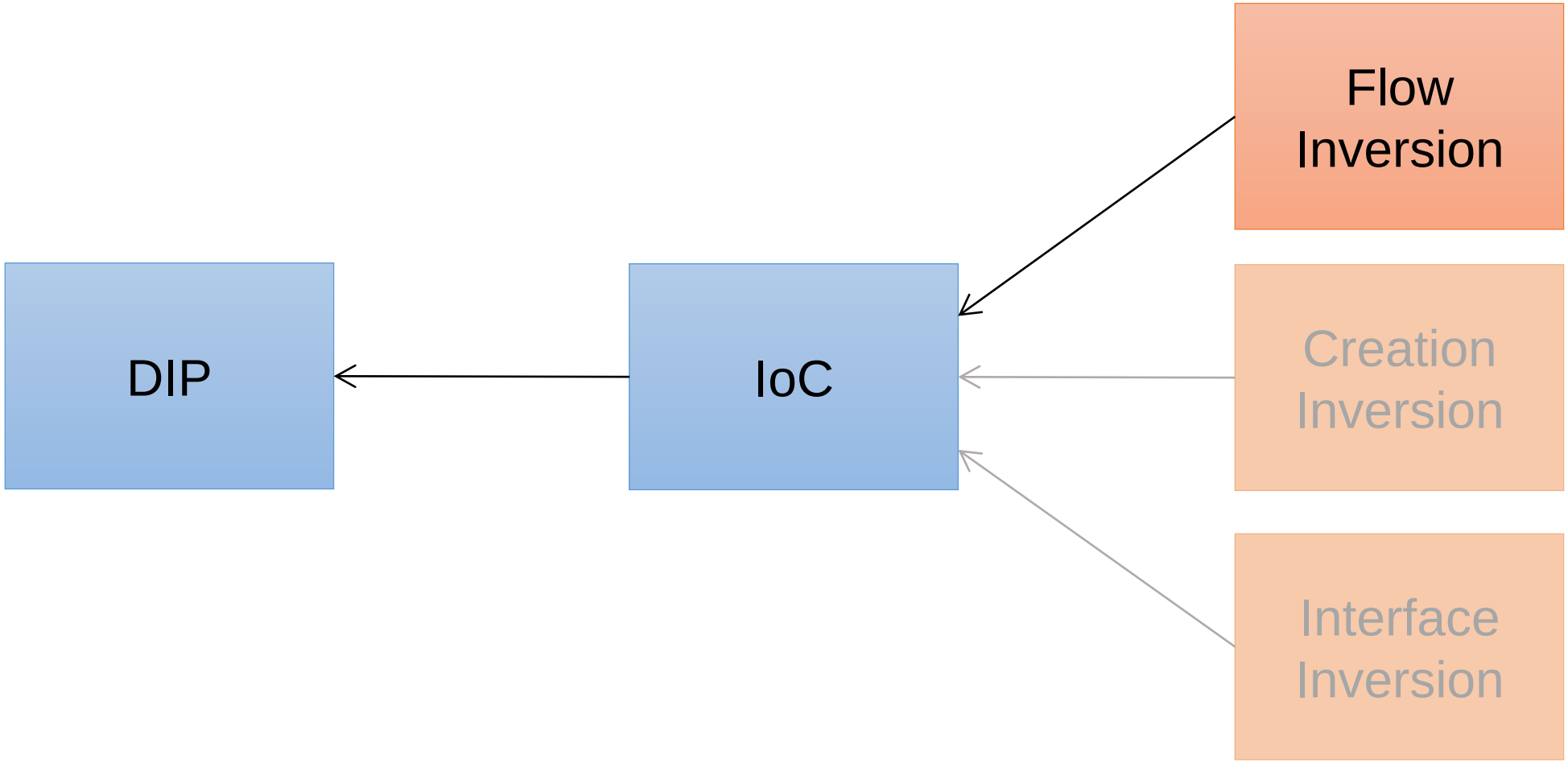


Saxonia Systems
So geht Software.

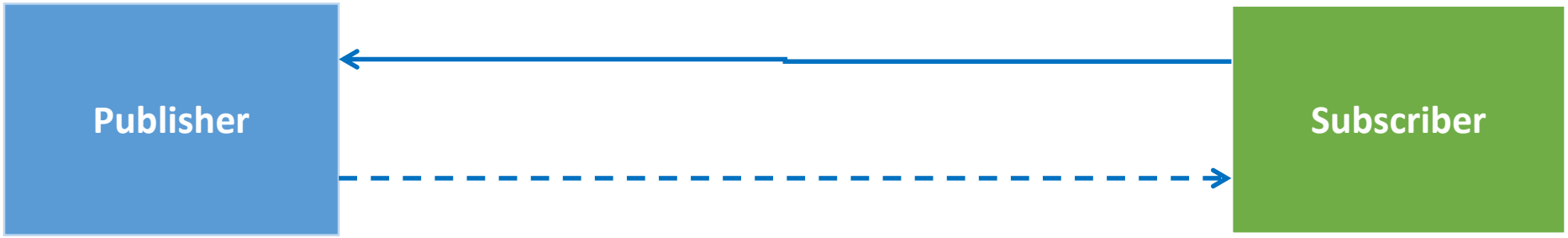
ohne IoC

mit IoC


DIP
IoC
Flow Inversion
Creation Inversion
Interface Inversion

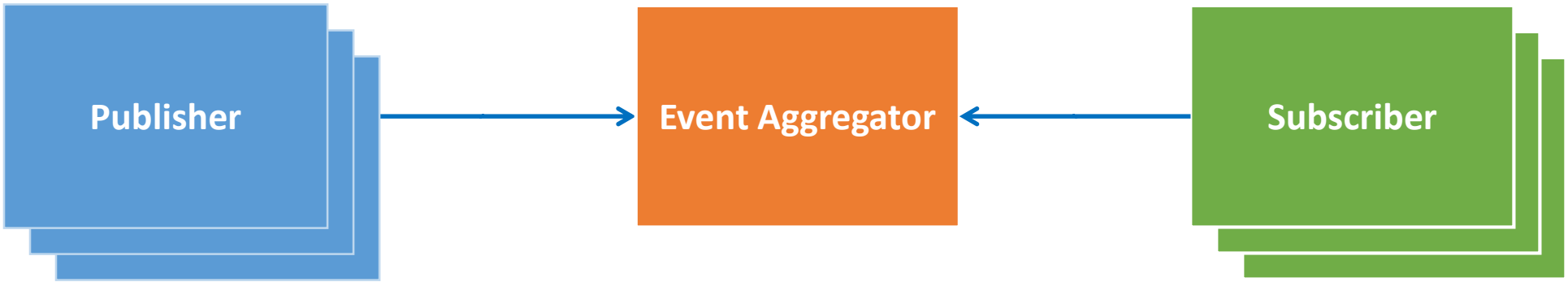

Flow Inversion
DIP
IoC
Creation Inversion
Interface Inversion

PUB SUB PATTERN

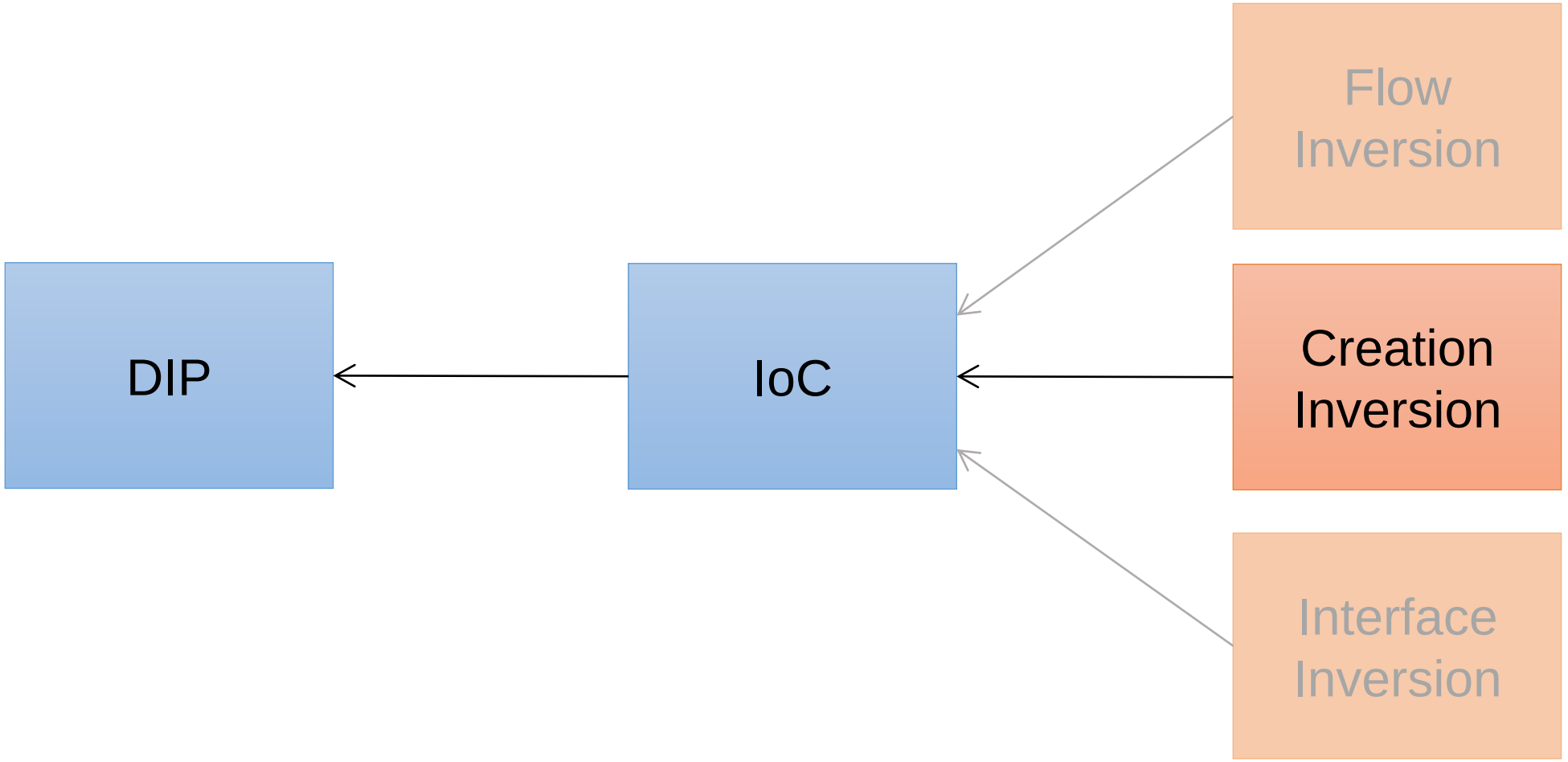


Flow Inversion

EVENT AGGREGATOR / MESSAGE BUS / PUB SUB SERVICE



Flow Inversion


DIP
IoC
Flow Inversion
Creation Inversion
Interface Inversion

IOC CONTAINER

Creation
Inversion

```
public class StudentContext : DbContext
{
    public StudentContext() : base()
    {}

    public DbSet<StudentEntity> Students { get; set; }
}
```

Creation
Inversion

```
public class StudentContext : DbContext, IStudentContext
{
    public StudentContext() : base()
    {}

    public DbSet<StudentEntity> Students { get; set; }
}
```

```
public class IStudentContext
{
    DbSet<StudentEntity> Students { get; set; }
}
```

Creation
Inversion

```
public class StudentManagementViewModel
{
    StudentContext context;

    public StudentEntity Student { get; set; }

    public StudentManagementViewModel()
    {
        this.context = new StudentContext();
    }

    public void ShowStudent(string name)
    {
        this.Student =
           this.context.Students.FirstOrDefault<StudentEntity>
           (s => s.StudentName == name);
    }
}
```

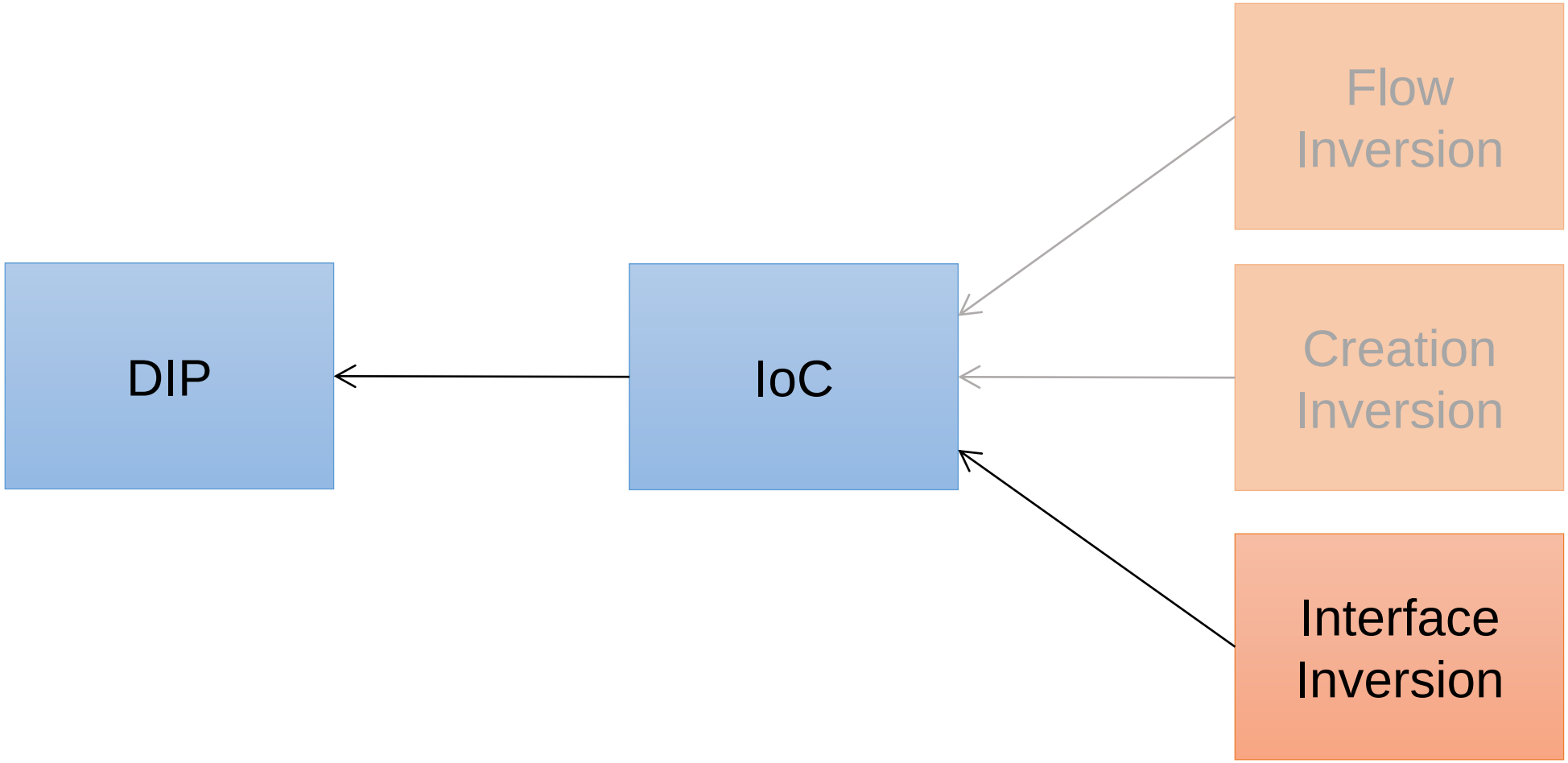
Creation
Inversion

```
public class StudentManagementViewModel
{
    IStudentContext context;

    public StudentEntity Student { get; set; }

    public StudentManagementViewModel()
    {
        this.context = ServiceLocator.Create<IStudentContext>();
    }

    public void ShowStudent(string name)
    {
        this.Student =
           this.context.Students.FirstOrDefault<StudentEntity>
           (s => s.StudentName == name);
    }
}
```

Creation
Inversion

```
public class StudentManagementViewModel
{
    IStudentContext context;

    public StudentEntity Student { get; set; }

    public StudentManagementViewModel(IStudentContext context)
    {
        this.context = context;
    }

    public void ShowStudent(string name)
    {
        this.Student =
           this.context.Students.FirstOrDefault<StudentEntity>
           (s => s.StudentName == name);
    }
}
```

Creation
Inversion


DIP
IoC
Flow Inversion
Creation Inversion
Interface Inversion

```
public class StudentContext : DbContext, IStudentContext
{
    public StudentContext() : base()
    {}

    public DbSet<StudentEntity> Students { get; set; }
}
```

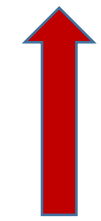
```
public class IStudentContext
{
    DbSet<StudentEntity> Students { get; set; }
}
```

Interface Inversion

Saxonia Systems
So geht Software.

```
public class StudentContext : DbContext, IStudentRepository
{
    public StudentContext() : base()
    {
        // ...
    }

    private DbSet<StudentEntity> students;

    public IQueryable<StudentEntity> Students { get{ return this.students} }
}
```

```
public class IStudentRepository
{
    IQueryable<StudentEntity> Students { get; set; }
}
```



Interface Inversion

Saxonia Systems
So geht Software.

```
[Table("StudentInfo")]
public class StudentEntity
{
    public StudentEntity() { }

    [Key]
    public int SID { get; set; }

    [Column("Name", TypeName = "ntext")]
    [MaxLength(20)]
    public string StudentName { get; set; }

    [Column("BDate", TypeName = "datetime")]
    public DateTime BirthDate { get; set; }

    [NotMapped]
    public int? Age { get;}

}
```

Interface Inversion

```
public abstract class Student
{
    public string Name { get; set; }

    public DateTime BirthDate { get; set; }

    public int? Age { get; }
}

public class IStudentRepository
{
    IQueryable<Student> Students { get; set; }

    Student CreateStudent();
}
```

```
internal class StudentEntity : Student
{
    public int SID { get; set; }
}
```

Interface Inversion

```
public class StudentManagementViewModel
{
    IStudentContext context;

    public StudentEntity Student { get; set; }

    public StudentManagementViewModel(IStudentContext context)
    {
        this.context = context;
    }

    public void ShowStudent(string name)
    {
        this.Student =
           this.context.Students.FirstOrDefault<StudentEntity>
           (s => s.StudentName == name);
    }
}
```

Interface Inversion

```
public class StudentManagementViewModel
{
    IStudentRepository repository;

    public Student Student { get; set; }

    public StudentManagementViewModel(IStudentRepository repository)
    {
        this.repository = repository;
    }

    public void ShowStudent(string name)
    {
        this.Student =
           this.repository.Students.FirstOrDefault<Student>
           (s => s.StudentName == name);
    }
}
```

Interface Inversion

Pseudocode!!!

Saxonia Systems
So geht Software.

```
public class IStudentRepository
{
    IQueryable<Student> Students { get; set; }

    Student CreateStudent();
}
public interface IManageStudents : IQueryable<Student>
{
    Student Create();

    void Add(Student student);

        ...
}

internal class StudentsFromDatabase : IManageStudents, DbSet {...}

this.Student = this.students.FirstOrDefault(s => s.StudentName == name);
```

Interface Inversion

Siehe auch: cessor.de

Header Interfaces

```
public class IStudentContext
{
    DbSet<StudentEntity> Students { get; set; }
}
```

Role Interfaces

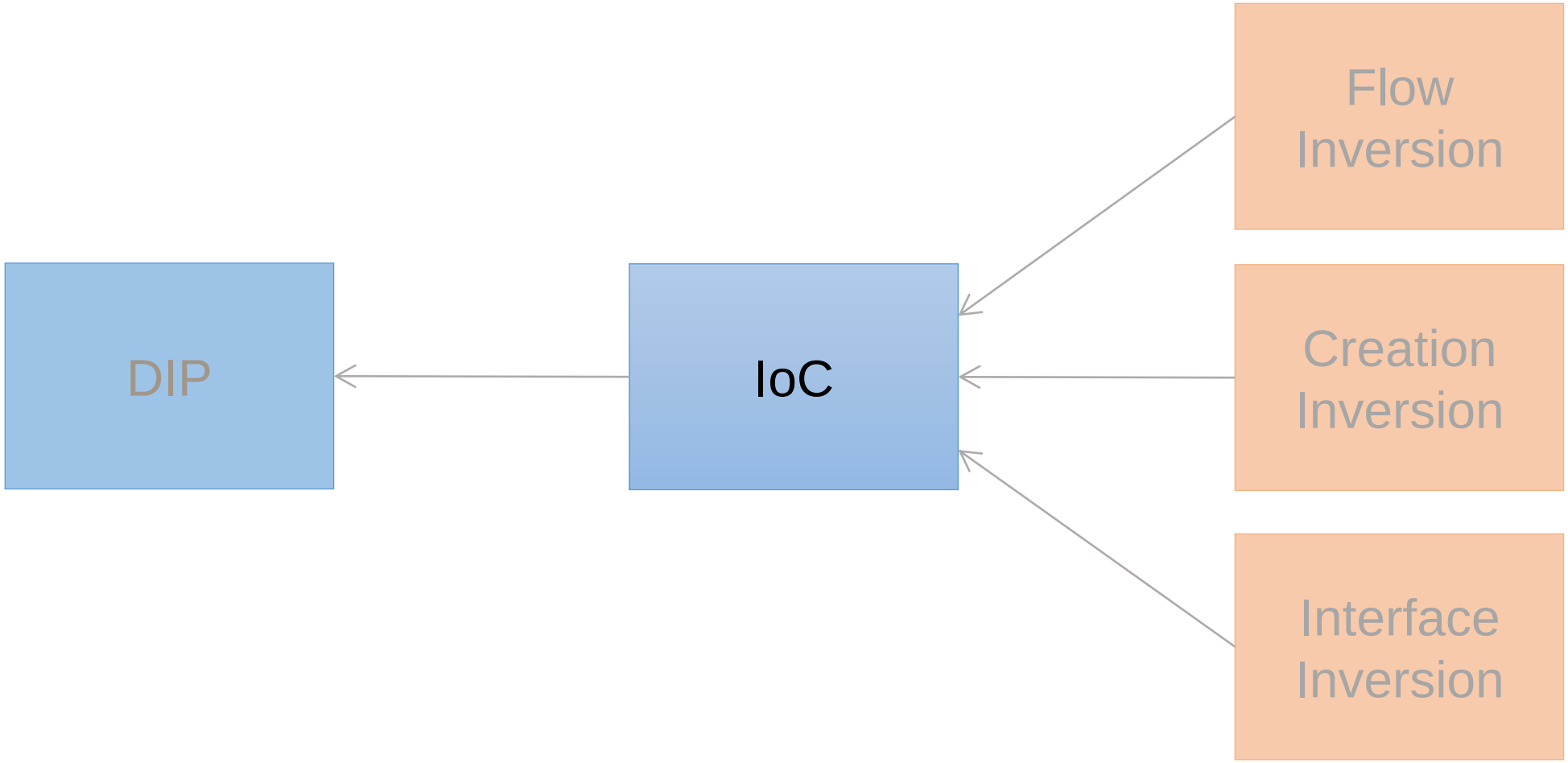
```
public interface IManageStudents : IQueryable<Student>
{
    Student Create();

    void Add(Student student);

        ...
}
```

```
Student student = this.repository.Students.Single(s => s.StudentName == name);

Student stud = this.repository.Students.Single(s => s.StudentName == name);

var student = this.repository.Students.Single(s => s.StudentName == name);
```

Interface Inversion

FAZIT

Saxonia Systems
So geht Software.


DIP
IoC
Flow Inversion
Creation Inversion
Interface Inversion

Was bringt uns Inversion of Control?

- Entkopplung der Ausführung einer Aktion von ihrer Implementierung.
- Fokussierung von Bestandteilen auf deren eigentliche Aufgabe.
- Grundstrukturen werden durch Verträge festgehalten.
- Zulieferer können einfacher durch andere ersetzt werden.

Was macht schlechtes Design aus?

1. It is hard to change because every change affects too many other parts of the system. (Rigidity)

2. When you make a change, unexpected parts of the system break. (Fragility)

3. It is hard to reuse in another application because it cannot be disentangled from the current application. (Immobility)

Quelle: http://www.objectmentor.com/resources/articles/dip.pdf

Was macht gutes Design aus?

Man kann Teile entfernen ohne, dass eine Welt zusammenbricht!

Der Sprecher

Hendrik Lösch

Senior Consultant & Coach

Hendrik.Loesch@saxsys.de

Just-About.Net

WPF-Anwendungen mit MVVM und Prism

Modulare Architekturen verstehen und umsetzen

Windows 8 Store Apps mit MVVM und Prism

XAML-Entwurfsmuster, Bootstrapping, Navigation, Messaging

Test Driven Development – Praxisworkshop

Business-Applikationen testgetrieben entwickeln

Inversion of Control und Dependency Injection

Prinzipien der modernen Software-Architektur ...

Test Driven Development mit C#

Grundlagen, Frameworks, best Practices

Automatisiertes Testen mit Visual Studio 2012

Grundlagen, Testarten und Strategien

Saxonia Systems

So geht Software.