



# 11. Design Patterns as Role Models

---

1

Prof. Dr. U. Aßmann

Chair for Software  
Engineering

Faculty of Informatics

Dresden University of  
Technology

WS 16/17 - Dec 1, 2016

**Lecturer:** Dr. Sebastian Götz

- 1) Design Patterns as Role Models
- 2) Composition of Design Patterns with Role Models
- 3) Effects of Role Modeling in Frameworks
- 4) Optimization of Design Patterns



# Literature (To Be Read)

2

- ▶ D. Riehle, T. Gross. Role Model Based Framework Design and Integration. Proc. 1998 Conf. On Object-oriented Programming Systems, Languages, and Applications (OOPSLA 98) ACM Press, 1998. <http://citeseer.ist.psu.edu/riehle98role.html>
- ▶ Dirk Riehle. Bureaucracy. In Robert Martin, Dirk Riehle, and Frank Buschmann, editors, Pattern Languages of Program Design 3, pages 163-185. Addison Wesley, 1998.
  - <http://dirkriehle.com/computer-science/research/1996/europlop-1996-bureaucracy.pdf>

# Remark

3

- ▶ Many role models and figures have been taken from:  
Dirk Riehle: **A Role-Based Design Pattern Catalog of Atomic and Composite Patterns Structured by Pattern Purpose.** In: Ubilab Technical Report 97-1-1
- ▶ <http://dirkriehle.com/computer-science/research/1997/ubilab-tr-1997-1-1.pdf>

# Goal

4

- ▶ Understand design patterns as role models
- ▶ Understand application of design patterns as merging role models into class models
- ▶ Understand composite design patterns
  - Understand how to mine composite design patterns
- ▶ Understand layered frameworks as role models
- ▶ Understand how to optimize layered frameworks and design patterns



# 11.1 Design Patterns as Role Models

---

---

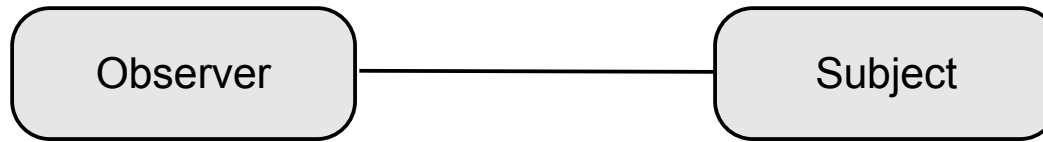
5



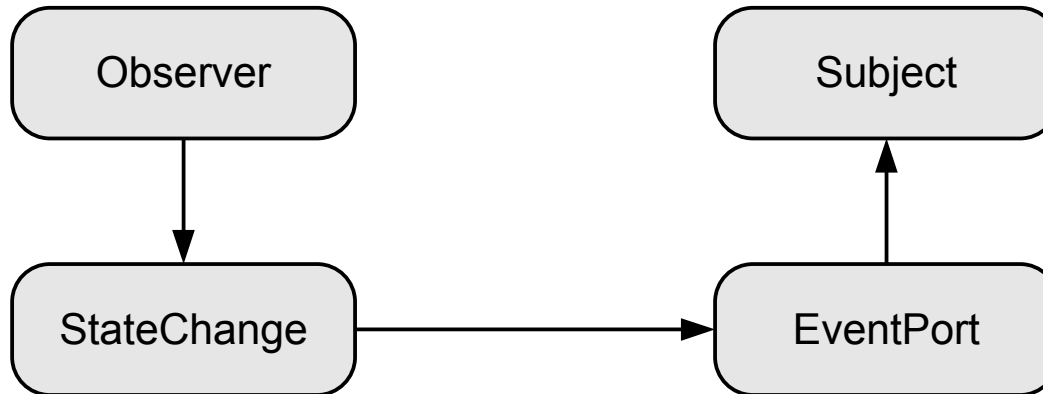
# Design Patterns have Role Models

6

## ▶ Observer role model



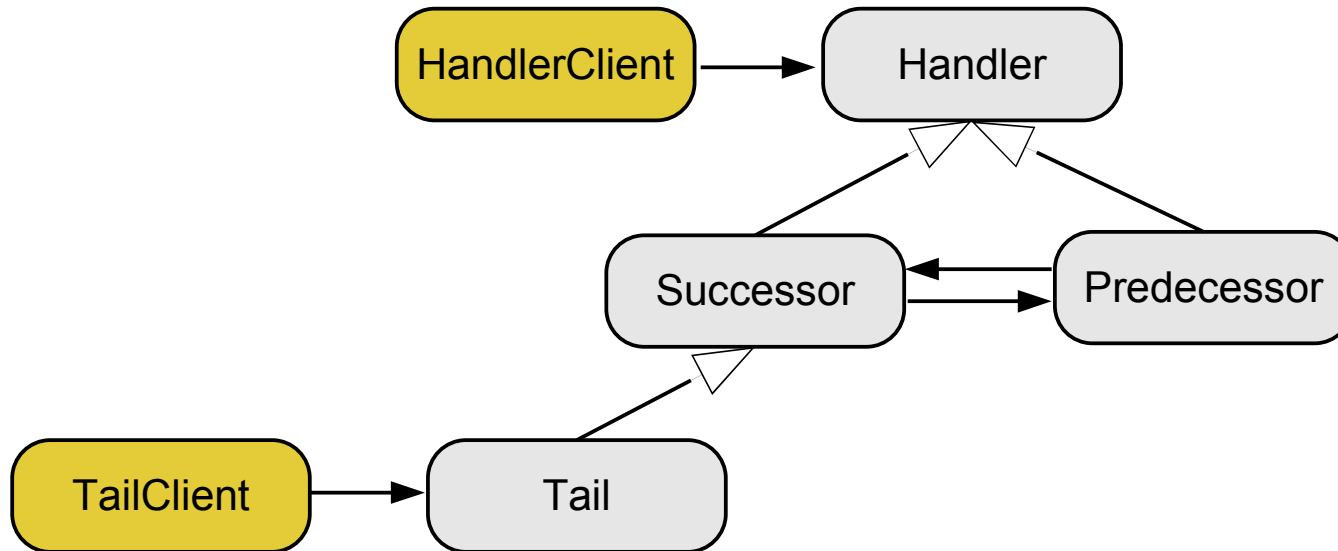
## ▶ Event Notification role model



# Participants of Patterns form Role Models

7

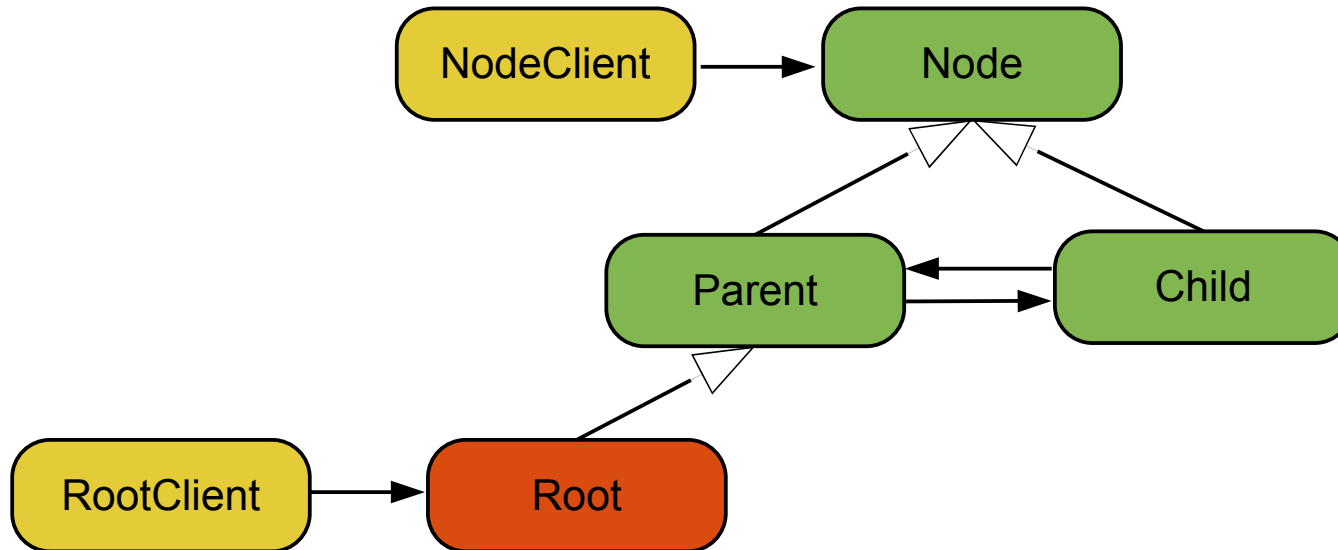
- ▶ The “participant” section of a GOF pattern is a *role model*



# Role Model of Composite

8

- ▶ Root role is not in the GOF pattern description

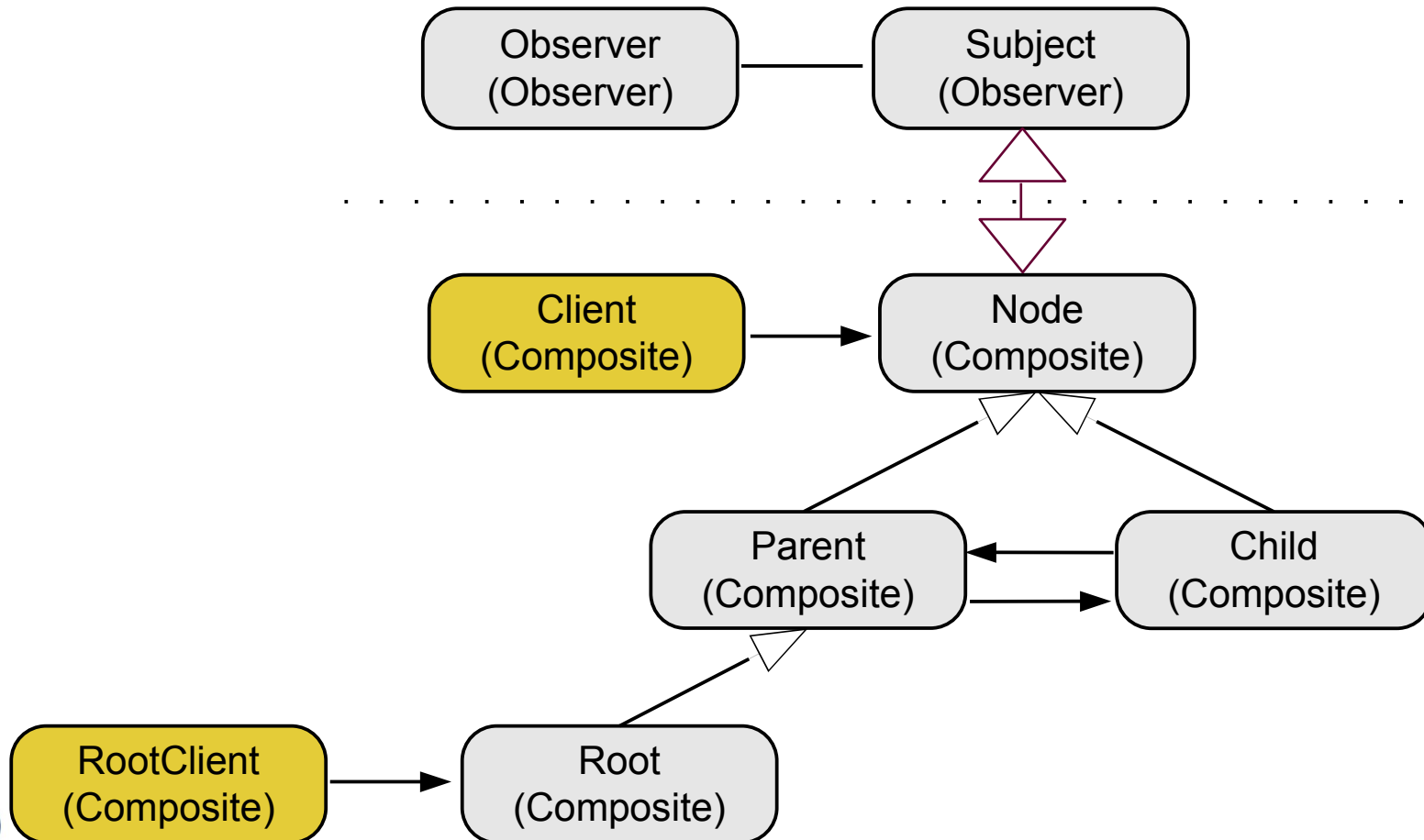




# Composing Role Models

9

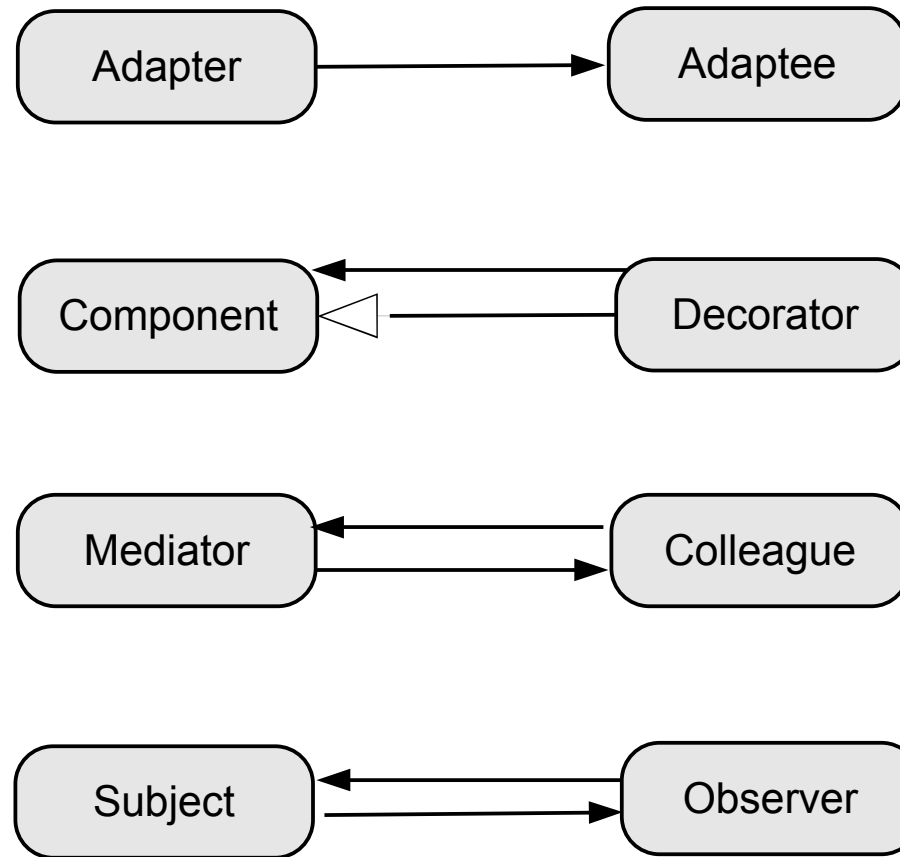
- ▶ Overlaying the Composite with the Observer role model by role equivalence



# Core Role Diagrams of Several Patterns

10

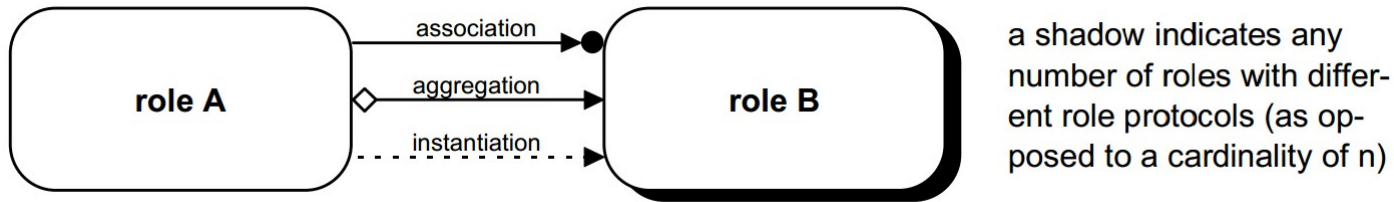
- ▶ Many of them are quite similar



# Adjustments to Riehle's Notation

11

- ▶ Riehle used special relationships between roles



- ▶ We do not use these!
- ▶ Roles have no identity and, hence, cannot aggregate
- ▶ Riehle used role aggregation to express the need to introduce an aggregation between the classes the roles are mapped to.



# 11.2 Composite Design Patterns with Role Model Composition

12

.. how to create bigger design patterns as composed role models..



# 11.2.1 Bureaucracy

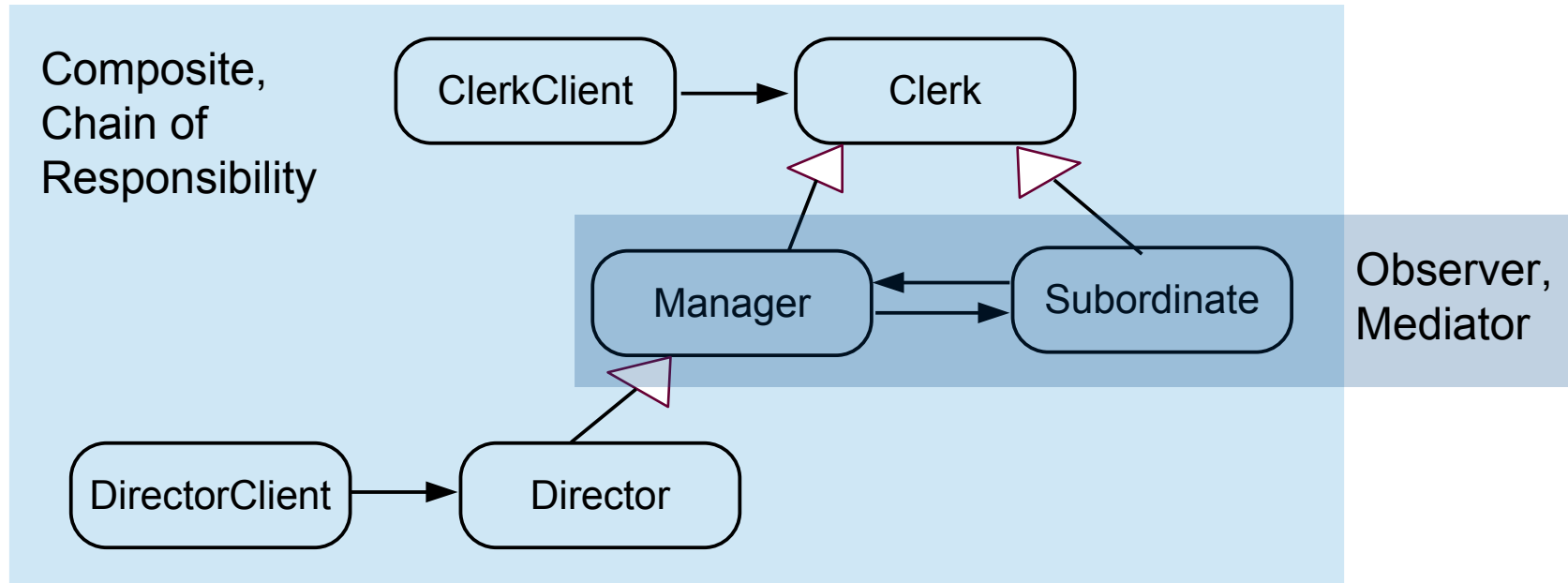
13



# Example: Bureaucracy

14

- ▶ A pattern to model organizations that have a tree-like structure (as opposed to matrix organizations)
  - Composed of the role models:  
Composite, Mediator, Chain of Responsibility and Observer



# Example: Bureaucracy

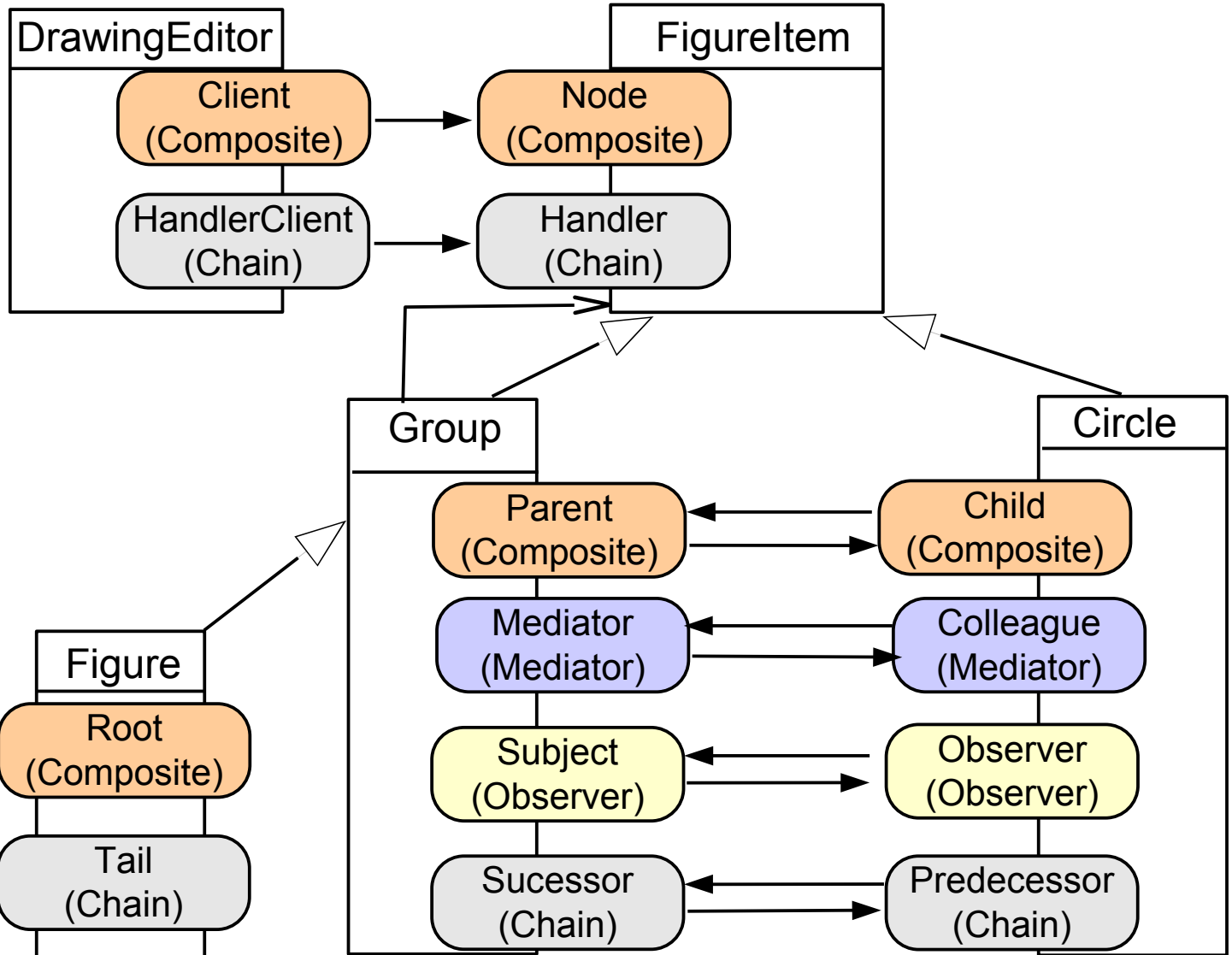
15

- ▶ The *Composite* defines the organizational hierarchy of managers
- ▶ The *Mediator* is used to let colleagues talk to their siblings via a parent (mediator role)
- ▶ The *Chain* handles requests of clients
  - Every node may handle requests
  - If a node cannot handle a request, it is passed up in the hierarchy (on the path to the root)
- ▶ The *Observer* is used to listen to actions of a subordinate node
  - If a subordinate node (subject) changes something, its manager (observer) is notified

# Bureaucracy Applied to Figure Example

16

Prof. Dr. G. Smann, Design Patterns and Frameworks





# Application of Bureaucracy

17

- ▶ For all hierarchies
  - Figures in graphic and interactive applications
  - Widgets in GUIs
  - Documents in office systems
  - Piece lists in production management and CAD systems
  - Hierarchical tools in TAM (see later)
  - Modeling organizations in domain models: companies, governments, clubs



# 11.2.2 Model-View-Controller (MVC)

---

---

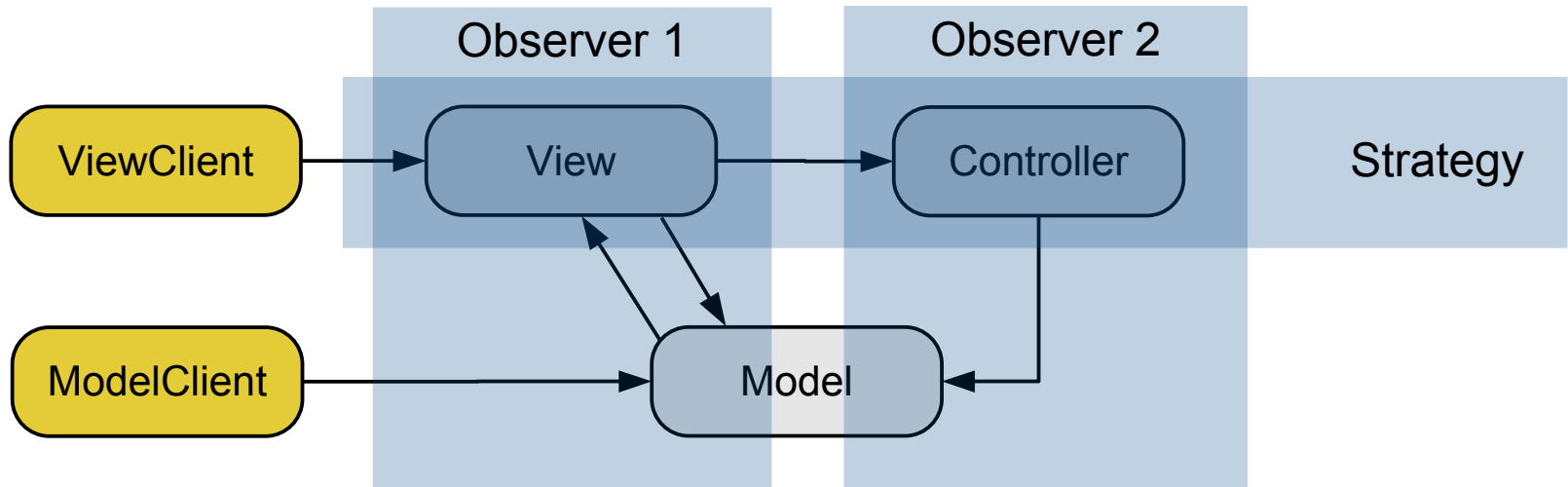
18



# Role Model of MVC

19

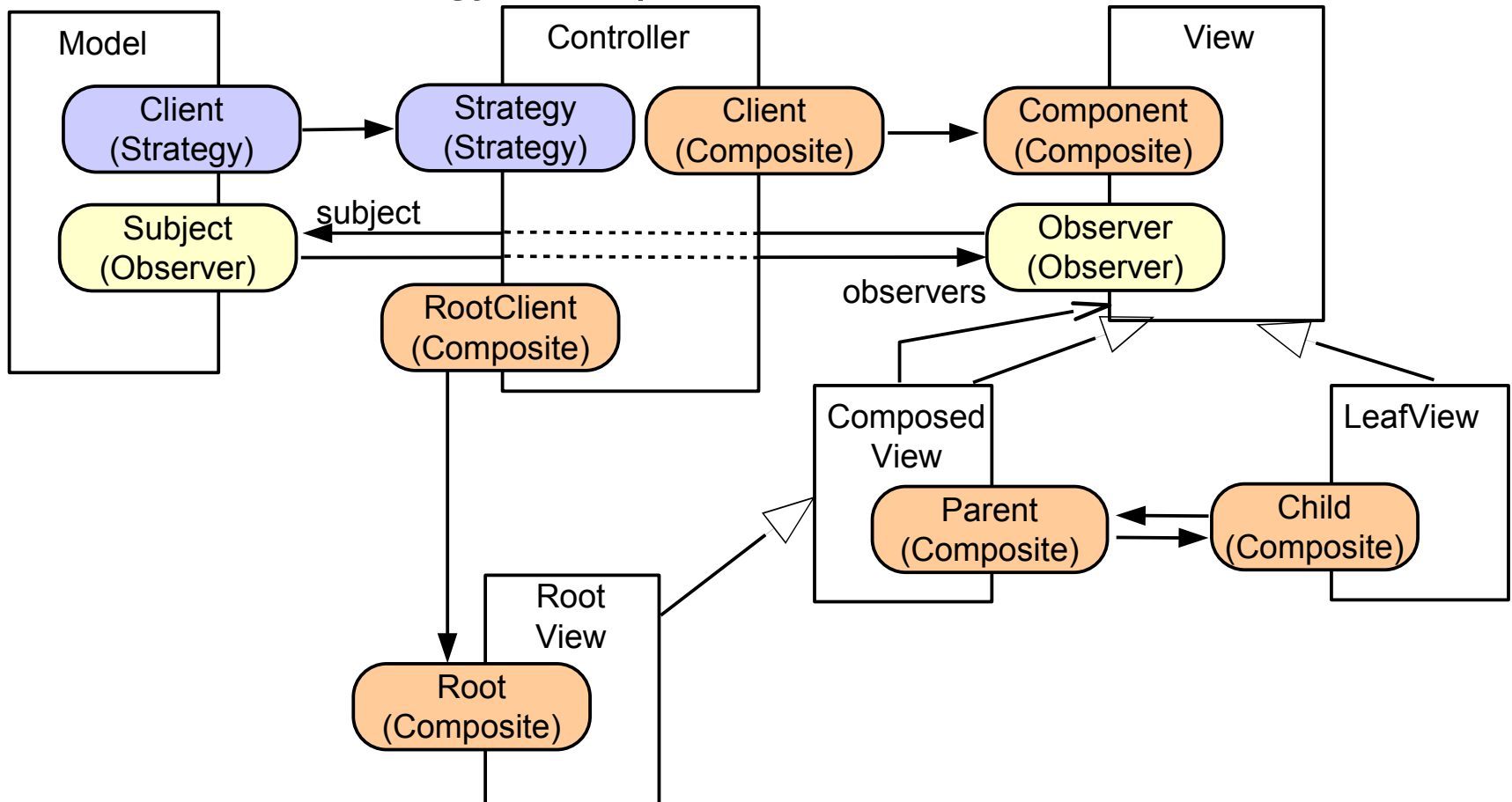
- ▶ MVC originates from Trygve Reenskaug and Adele Goldberg
- ▶ MVC role model can be composed from the role models of Observer and Strategy
  - Views and Controllers observe the model
  - Controllers can be varied (Strategy)
  - Extension with Composite Pattern for views possibles



# Role-Class Model of MVC

20

- ▶ MVC originates from Trygve Reenskaug and Adele Goldberg
- ▶ MVC role model can be composed from the role models of Observer, Strategy, Composite



# This Closes a Big Loop

21


- ▶ Remember, Reenskaug developed MVC 1978 with Goldberg, while working on Smalltalk-78 port for Norway
- ▶ Starting from his MVC pattern, Reenskaug has worked on role-based design
- ▶ 1998, Riehle/Gross transferred role-based models to design patterns
- ▶ Today, MVC can be explained as composed role model of other design patterns

# Riehle-Gross Law On Composite Design Patterns

*The role model of a composite design patterns is composed of the role models of their component design patterns*

## ► Consequences

- Complex patterns can be easily split into simpler ones (decomposition)
- Variants of patterns can more easily be related to each other (variability of patterns)
  - e.g., ClassAdapter and ObjectAdapter
- Template&Hook conceptual pattern can be explained as role model (see next chapter)



# 11.2.3 Composition of Simple Variability Patterns

---

---

23



# Warning

24

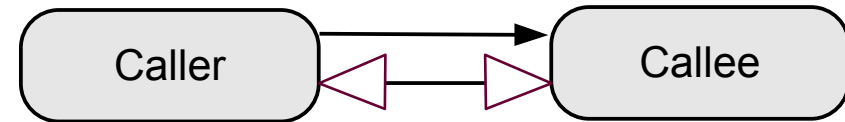
- ▶ The following is an attempt to build up the basic GOF patterns from simple role models based on the findings by Riehle and colleagues
- ▶ The compositions of patterns depend on the concrete form of their role models
- ▶ It explains why Strategy is different from Bridge and TemplateClass, etc.



# Derived Method

25

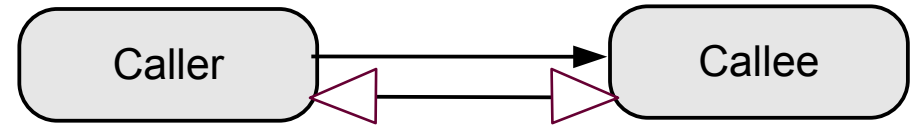
- ▶ In a class,
  - A *kernel method* implements the feature directly on the attributes of the class, calling no other method
  - A *derived method* is implemented by calling only kernel methods
- ▶ Caller and callee role have to be bound to the same class (as the purpose is to have class-internal method calls)



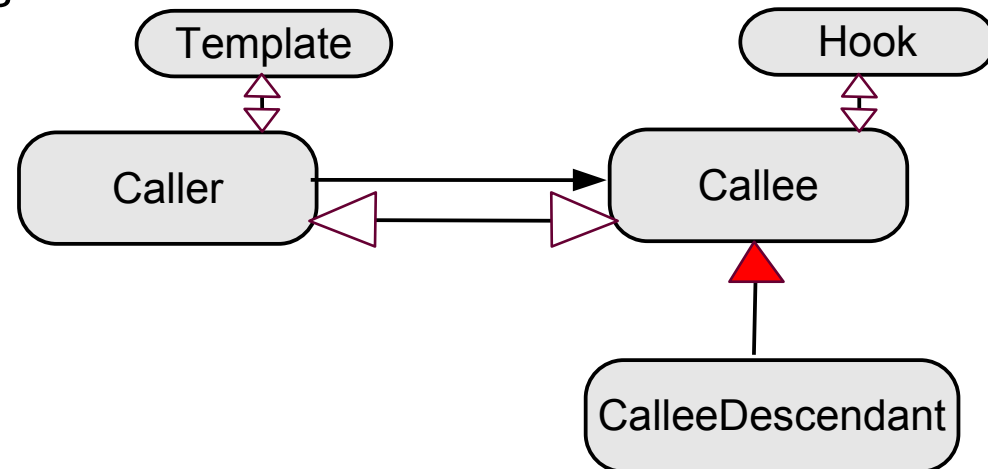
# Derived Method and TemplateMethod

26

- ▶ TemplateMethod is a DerivedMethod that has
  - an additional Template/HookMethod role model
  - Inheritance hierarchy on right side (implied by role-class inheritance constraint)
  - The template role implies no hierarchy on left side



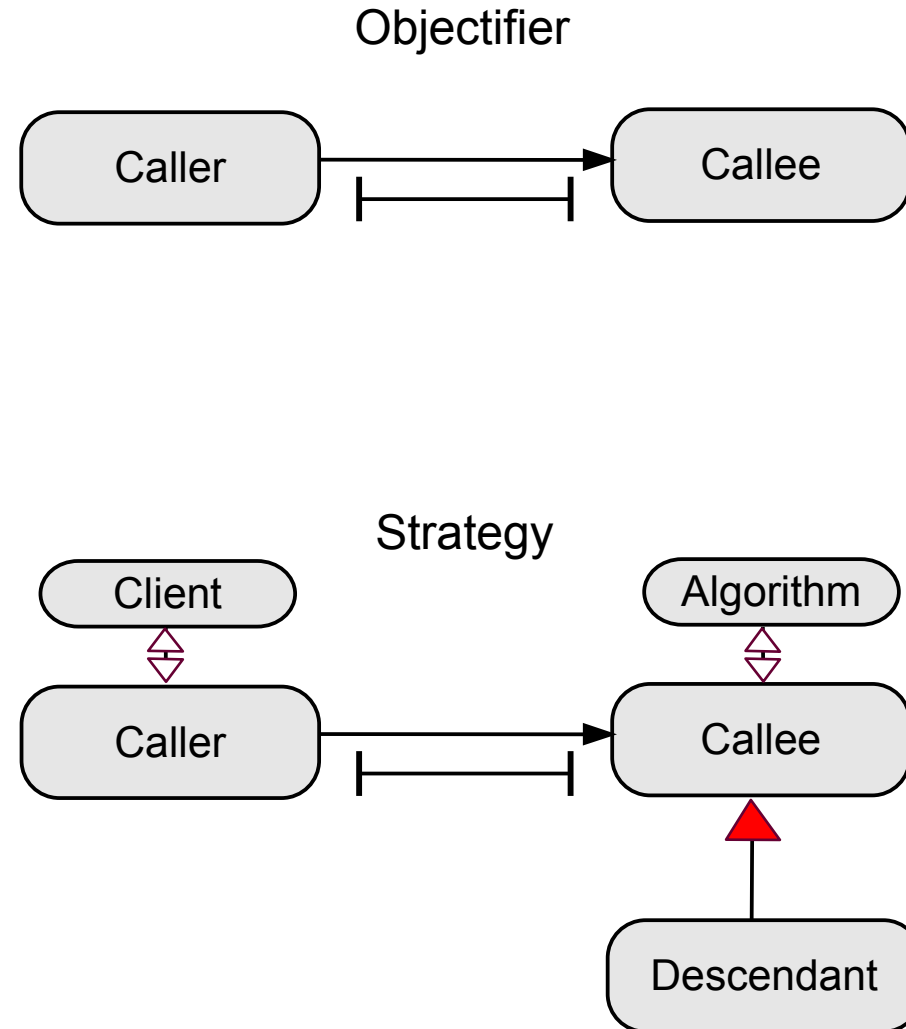
## TemplateMethod



# Objectifier and Strategy

27

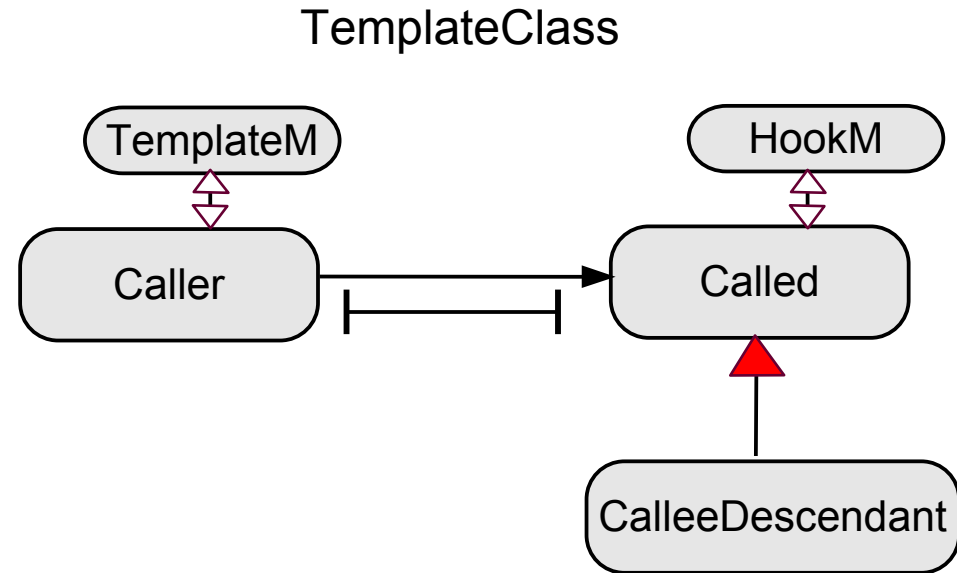
- ▶ Objectifier has
  - A prohibition constraint on Caller and Callee (instead of equivalence)
  - No template role
- ▶ Strategy is an Objectifier with
  - Client role
  - Algorithm role
  - Hierarchy on right side
  - No template role



# TemplateClass

28

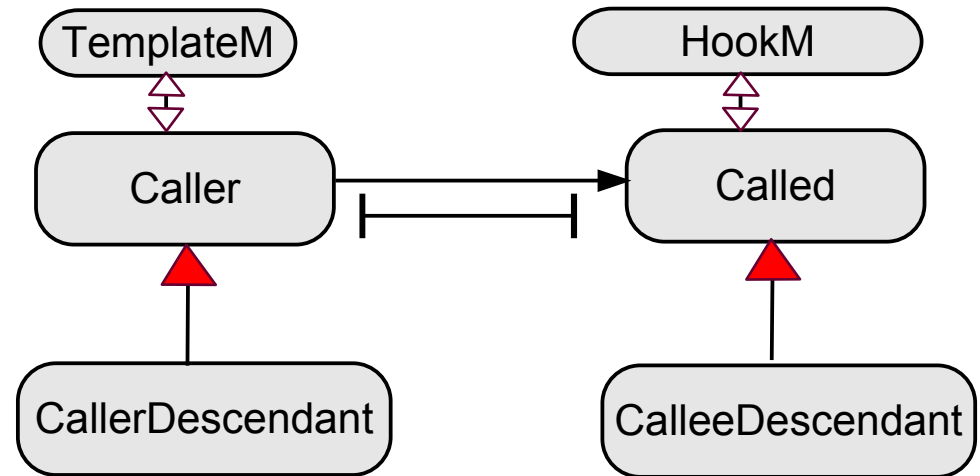
- ▶ TemplateClass is an Objectifier with
  - An additional Template/ HookMethod role model
  - TemplateMethod role implies no hierarchy on left side
  - HookMethod role implies inheritance hierarchy on right side
  - *No client or algorithm role, otherwise like Strategy*



# DimensionalClassHierarchies

29

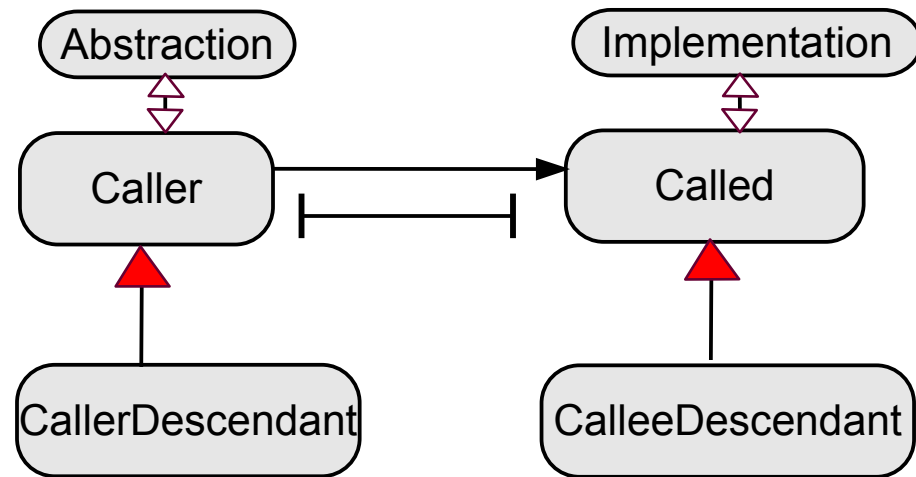
- ▶ DimensionalClassHierarchies is a TemplateClass
  - With left hierarchy constraint



# Bridge

30

- ▶ Bridge is a DimensionalHierarchies with
  - Abstraction/Implementation roles instead of T&H
  - *No template/hook role*

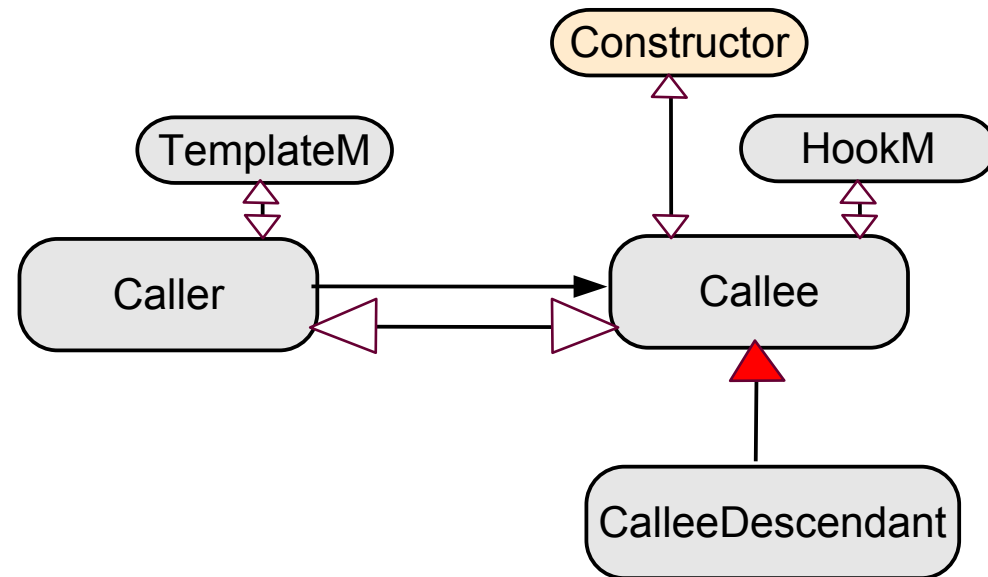


# Creational Patterns

31

- ▶ Add more roles with semantics about creation
- ▶ E.g., FactoryMethod is a TemplateMethod with a creational role model

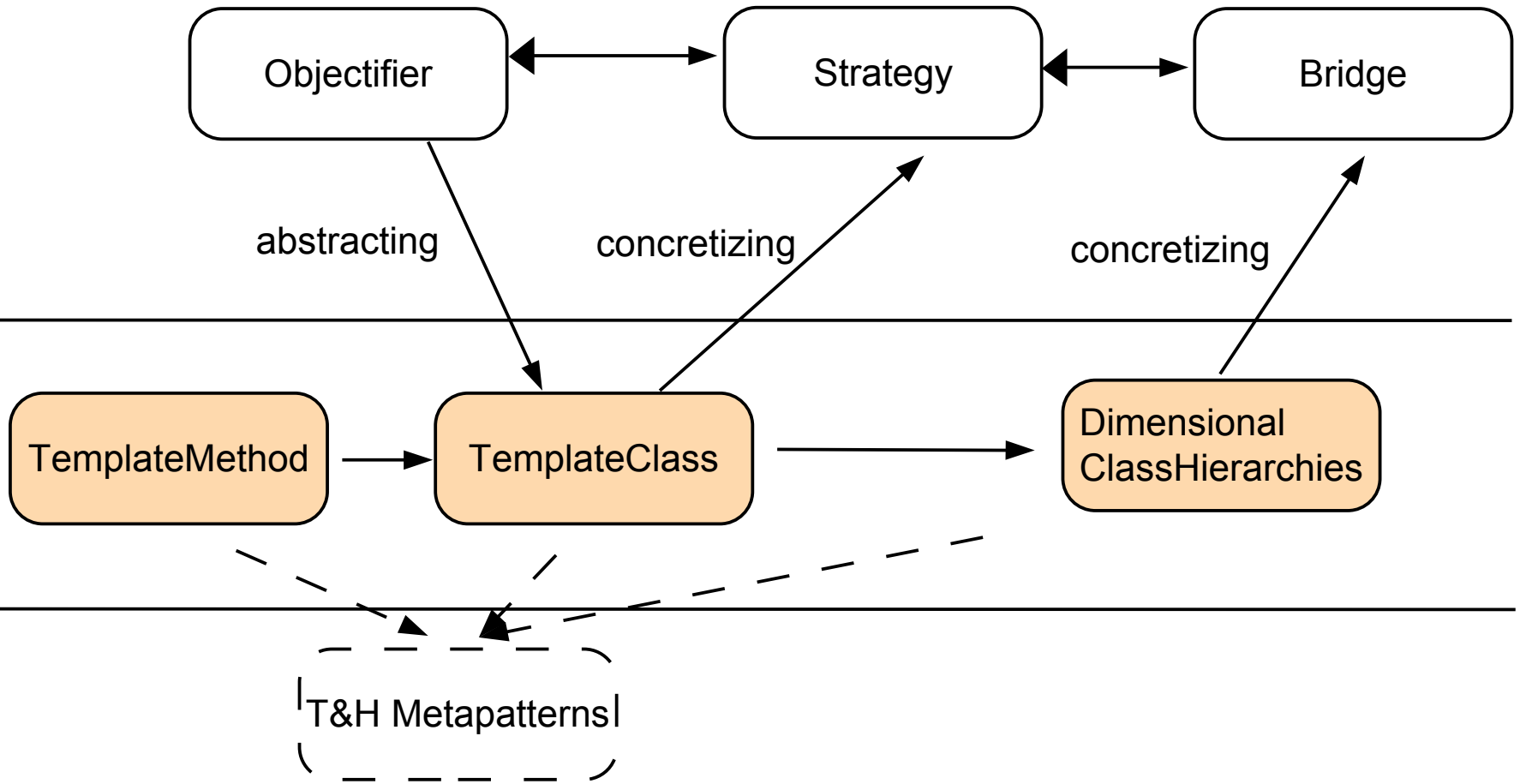
## FactoryMethod



# Remember: Relation TemplateMethod, TemplateClass, Strategy, Observer

32

More specific patterns (with more intent, more pragmatics, specific role denotations)



Framework Patterns (with TemplateM/HookM role model)





# 11.2.5 Consequences of the Riehle/Gross Law

---

---

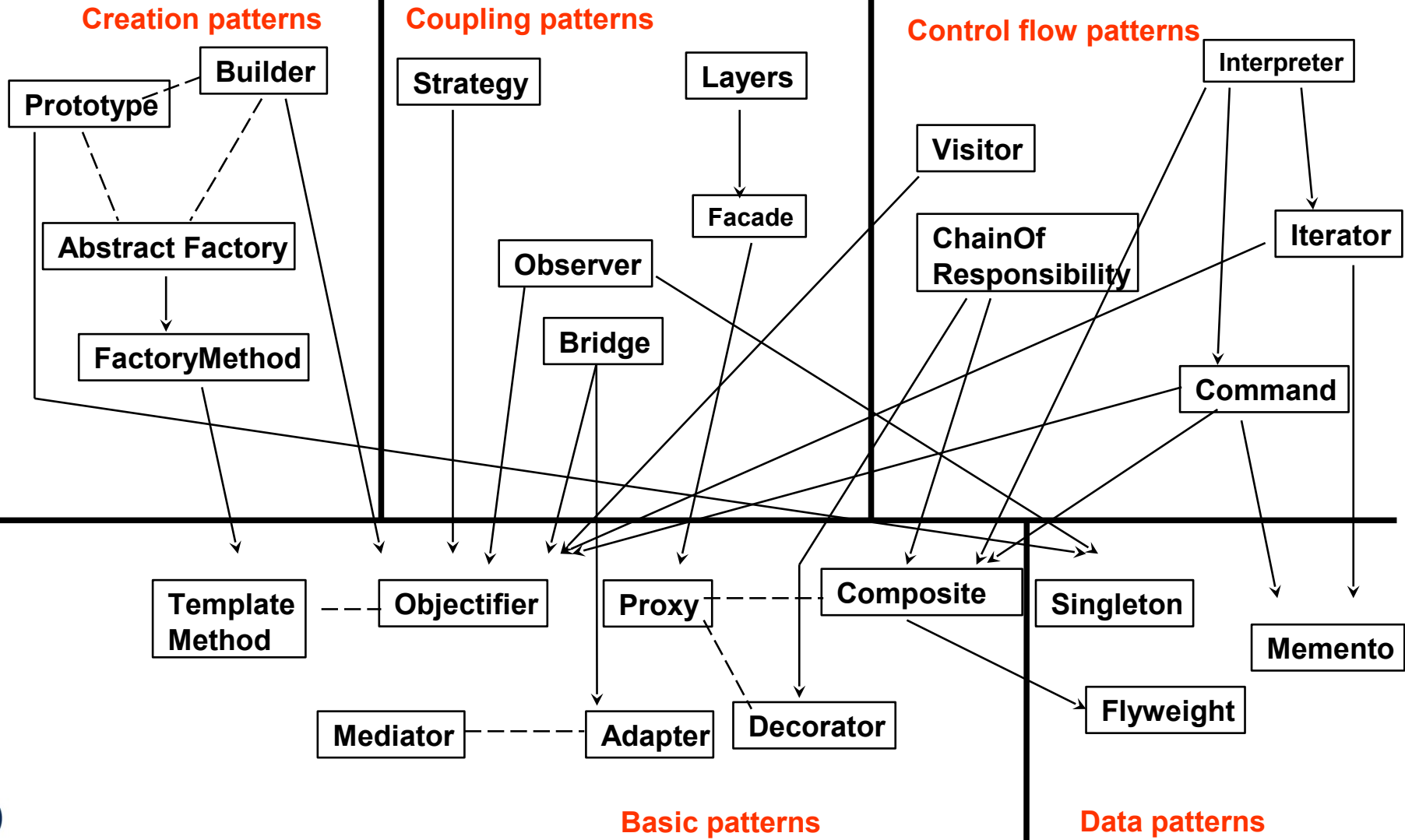
33



# Relations between Patterns

## [Zimmer, PLOP 1]

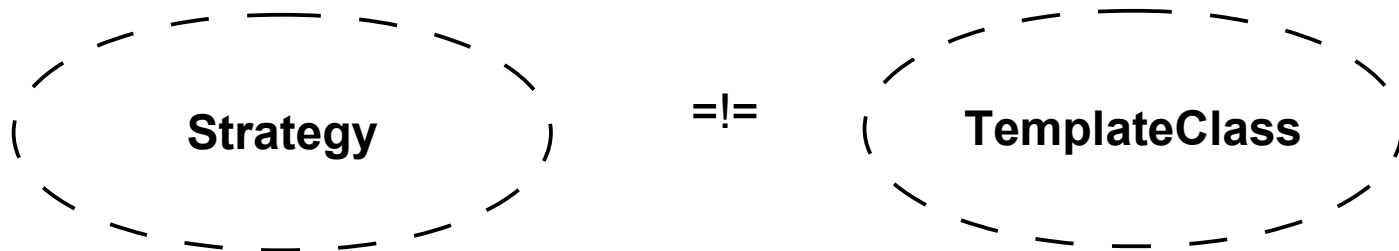
34



# Vision for Pattern-Based Design

35

- ▶ With different role models, the fine semantic differences between several patterns can be expressed syntactically
  - A role model can capture *intent (pragmatics)* of a pattern
  - While patterns can have the same structure, the intent may be different
  - It is possible to distinguish a Strategy, TemplateClass, a Bridge or DimensionalClassHierarchy
- ▶ This makes designs more explicit, precise, and formal
- ▶ Prerequisite: role types have to be formally specified (which is current research)



# 11.3 Effects of Role-Based Design Patterns on Frameworks and Applications

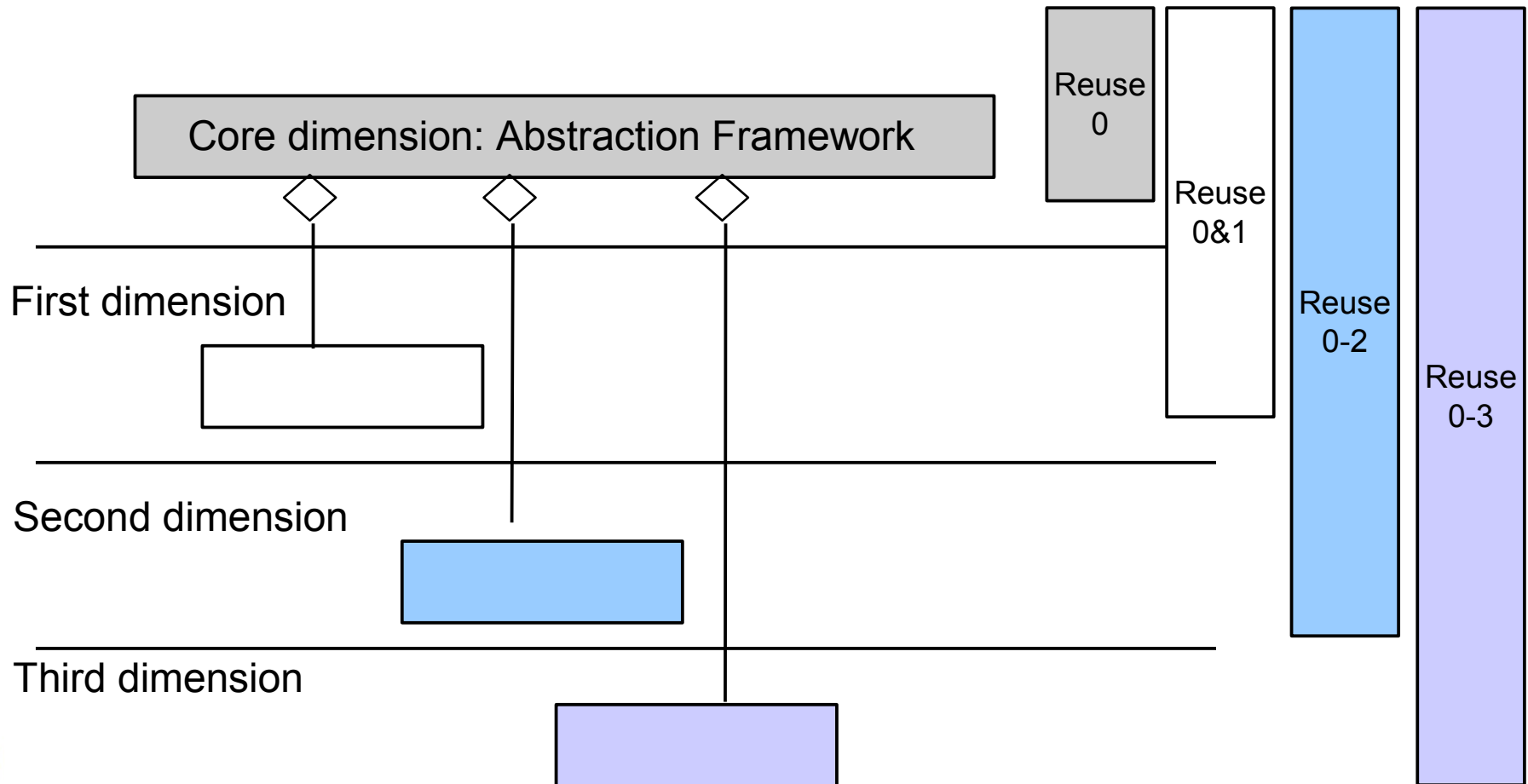
36



# Role Models and Facet/Layered Frameworks

37

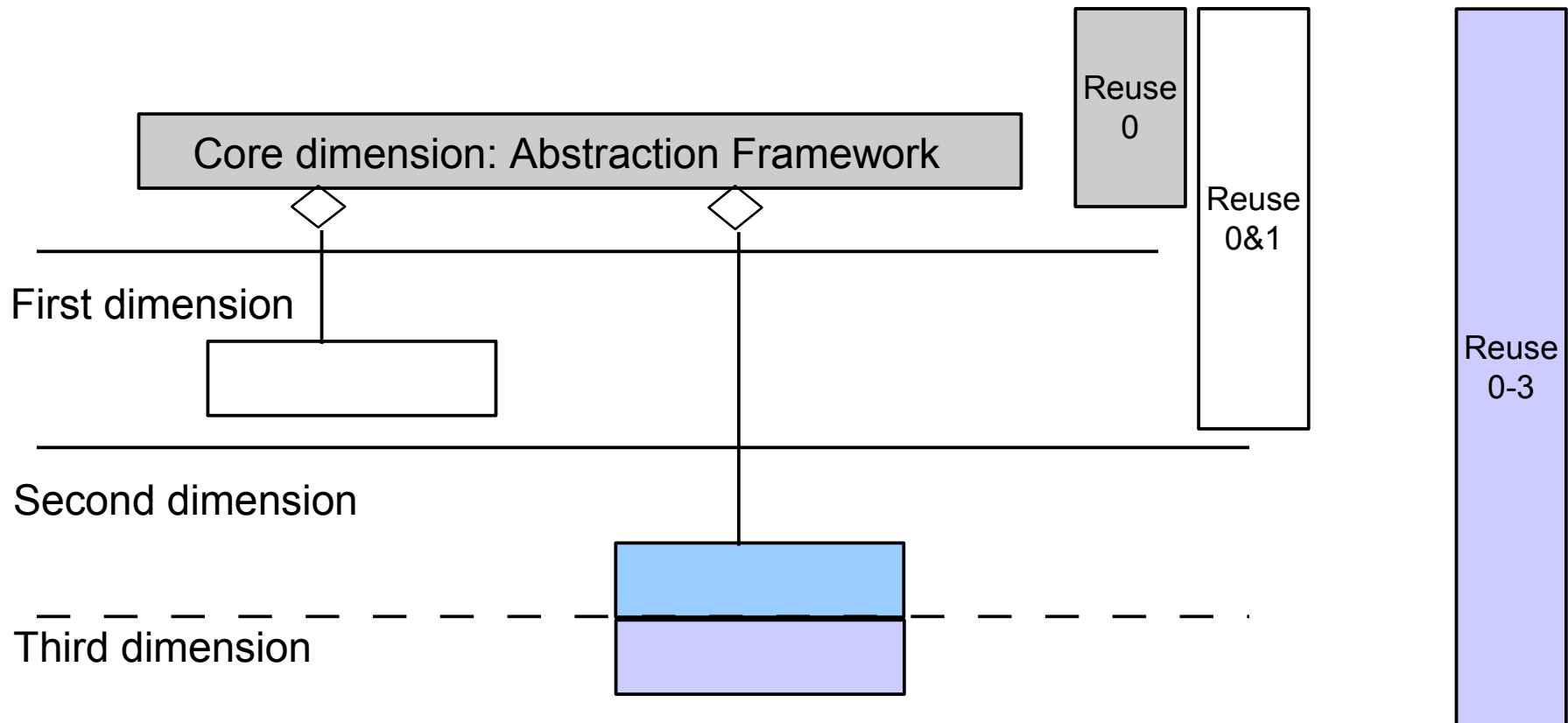
- ▶ Remember: An n-Bridge framework maintains roles (role models) in every facet (because a facet model is based on a class-role model)
- ▶ Similar for Chain-Bridges and layered frameworks



# Merging dimensions of Facet/Layered Frameworks

38

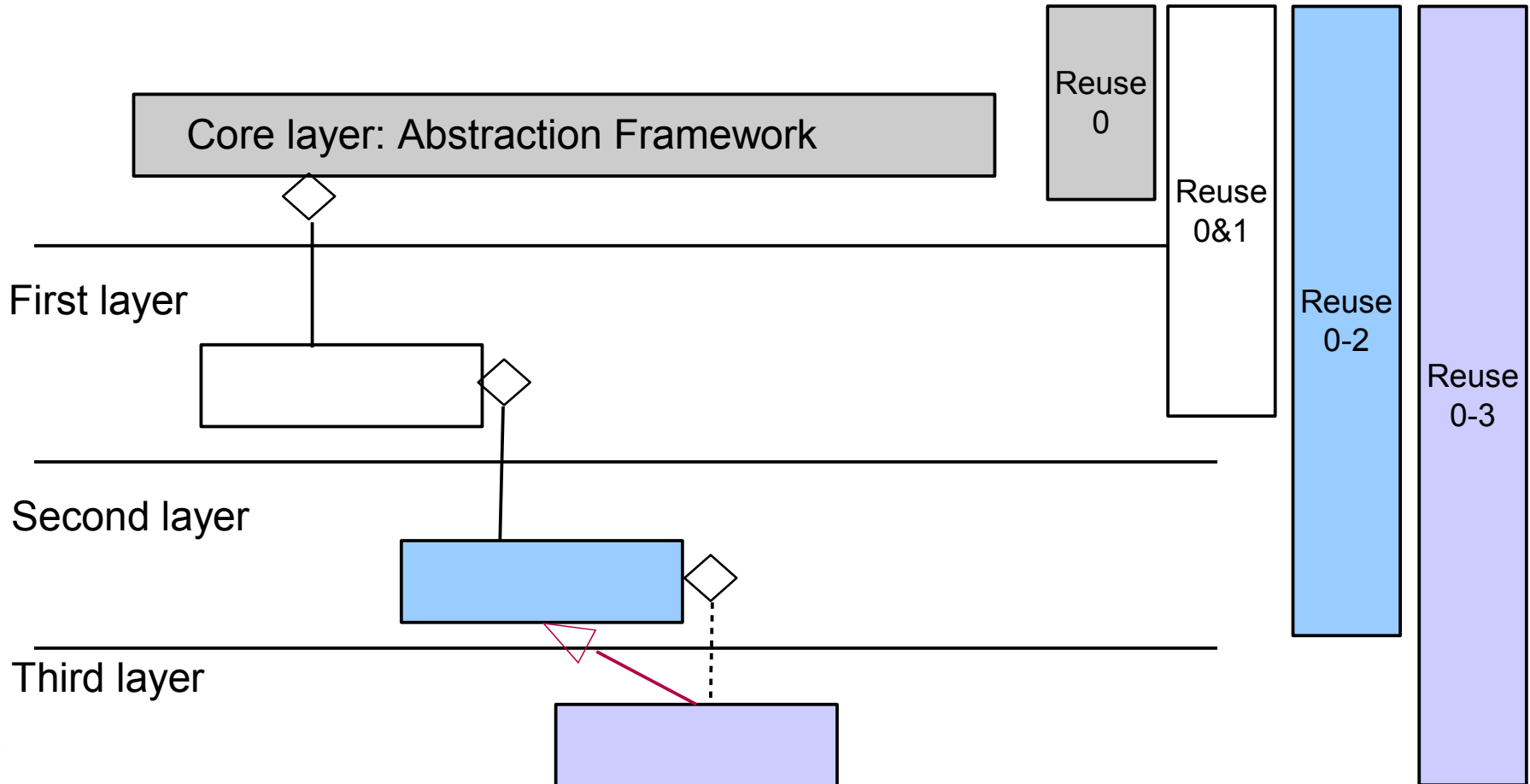
- ▶ If the dimensions are seen as role models, it can be chosen to merge them, i.e., the role models
- ▶ Here: merge second and third dimension into one physical implementation



# Role Models and Layered Frameworks

39

- ▶ Similar for Chain-Bridges and layered frameworks



# Merging Dimensions/Layers of Dimensional/Layered Frameworks

40

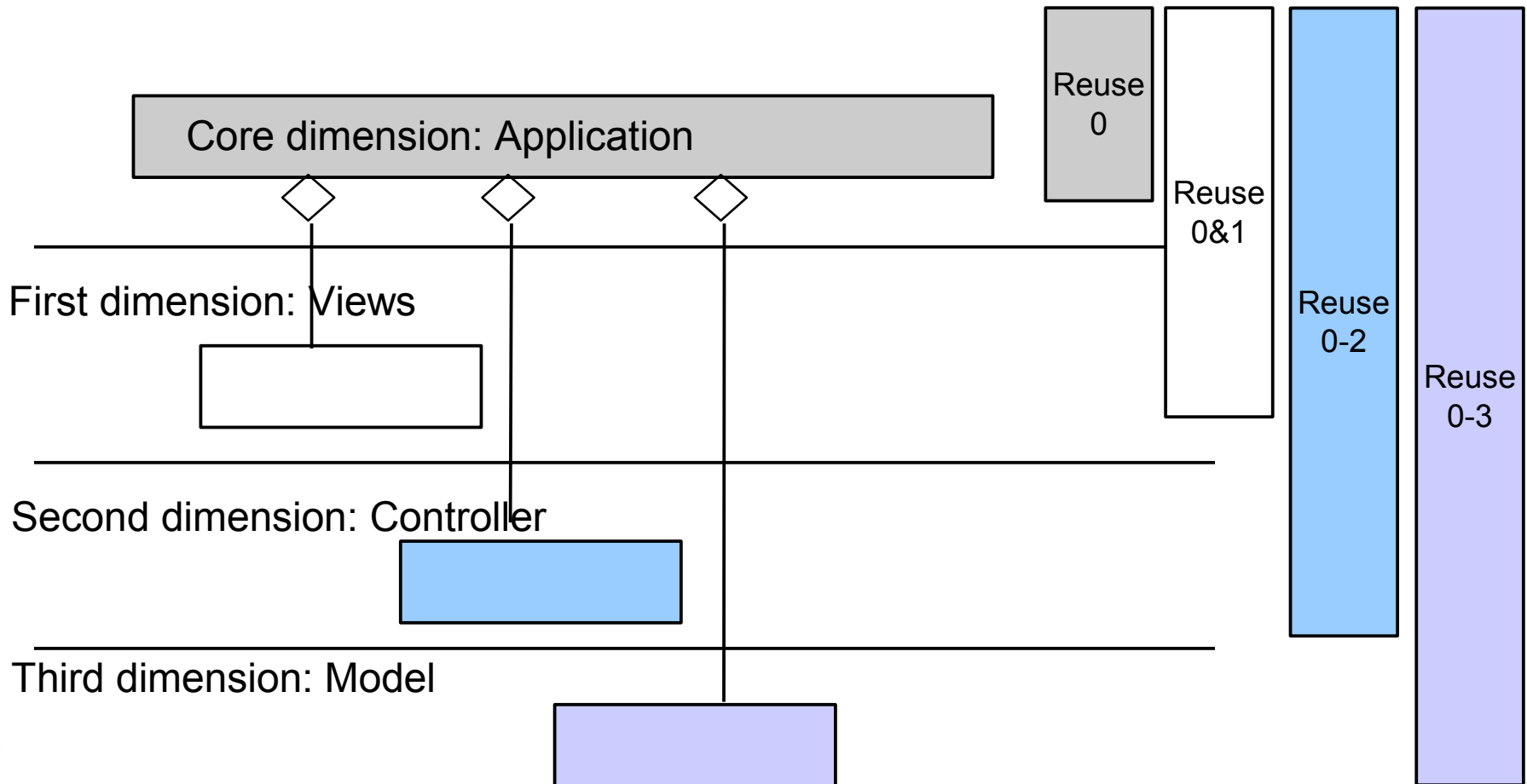
- ▶ When two layers are merged, the variability of a framework decreases
- ▶ But its applications are more efficient:
  - Less delegations (less bridges)
  - Less allocations (less physical objects)
  - Less runtime flexibility (less dynamic variation)



# MVC as Multi-Bridge Framework

41

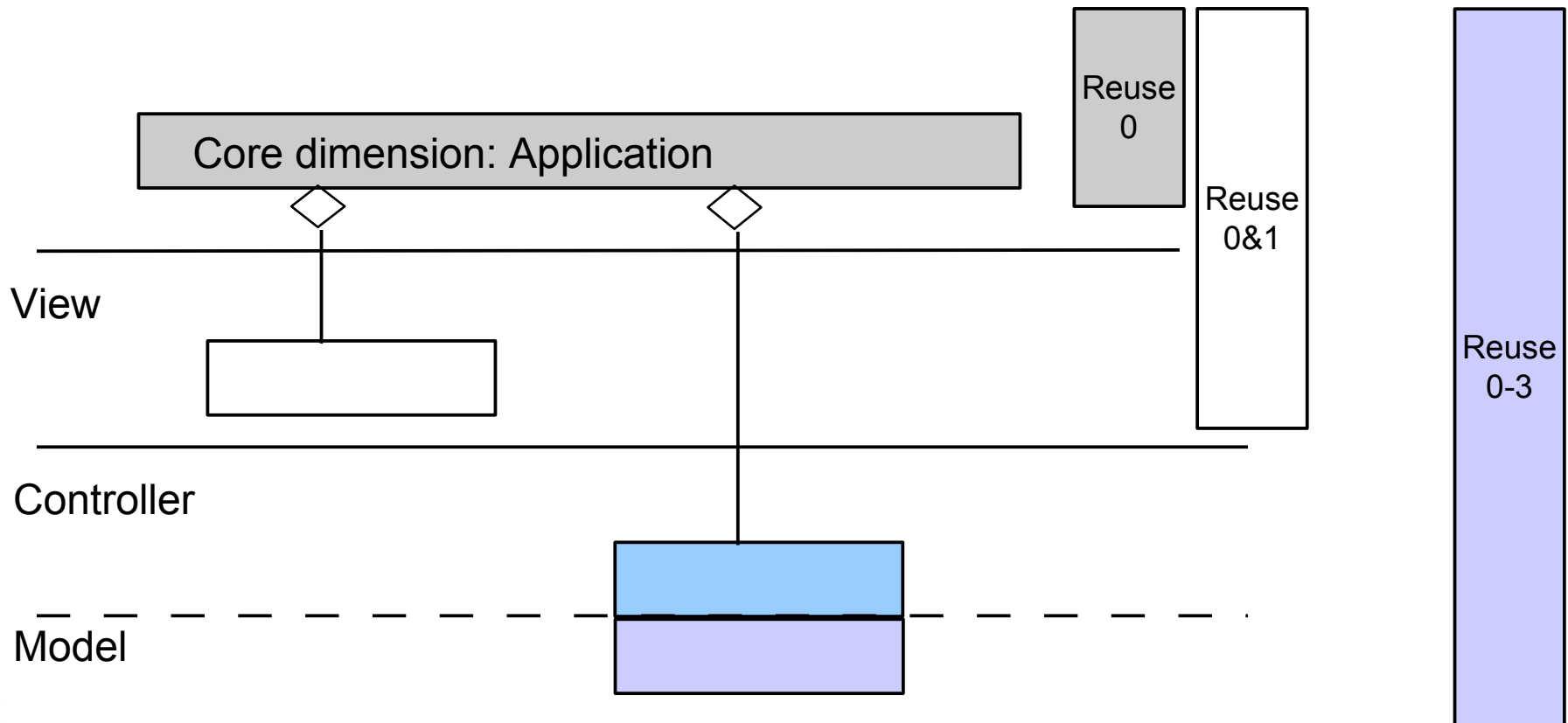
- ▶ The roles of MVC can be ordered in a n-Bridge framework



# MVC as Optimized Multi-Bridge Framework

42

- ▶ Model and Controller layer can be merged
- ▶ Less variability, but also less runtime objects





# 11.4 Optimization of Design Patterns with Role Models

---

---

43



# Optimization for Design Patterns

44

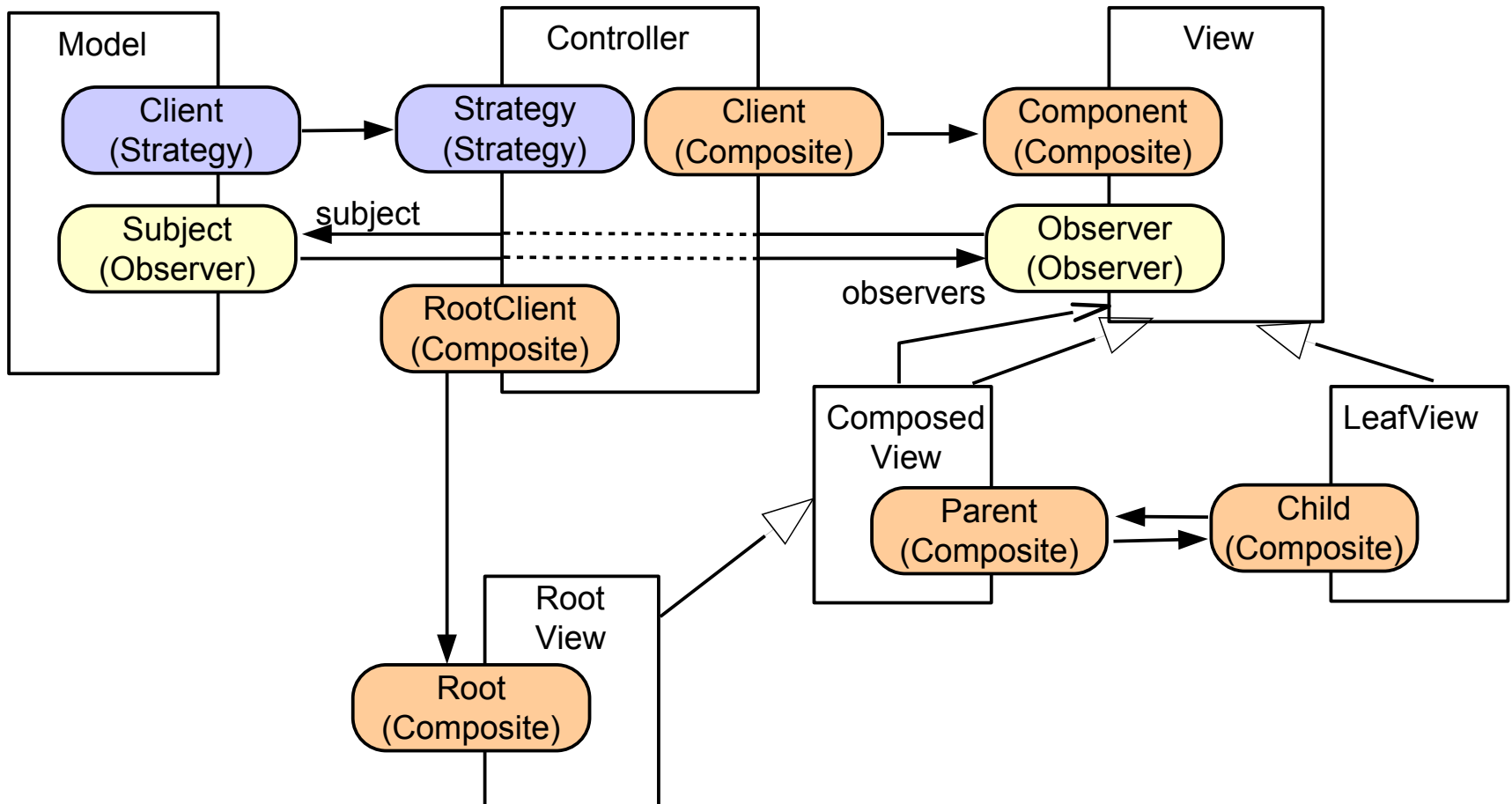
Whenever you need a variant of a design pattern that is more efficient, investigate its role model and try to merge the classes of the roles

- ▶ Effect:
  - Less variability
  - Less runtime objects
  - Less delegations

# Original Role-Class Model of MVC

45

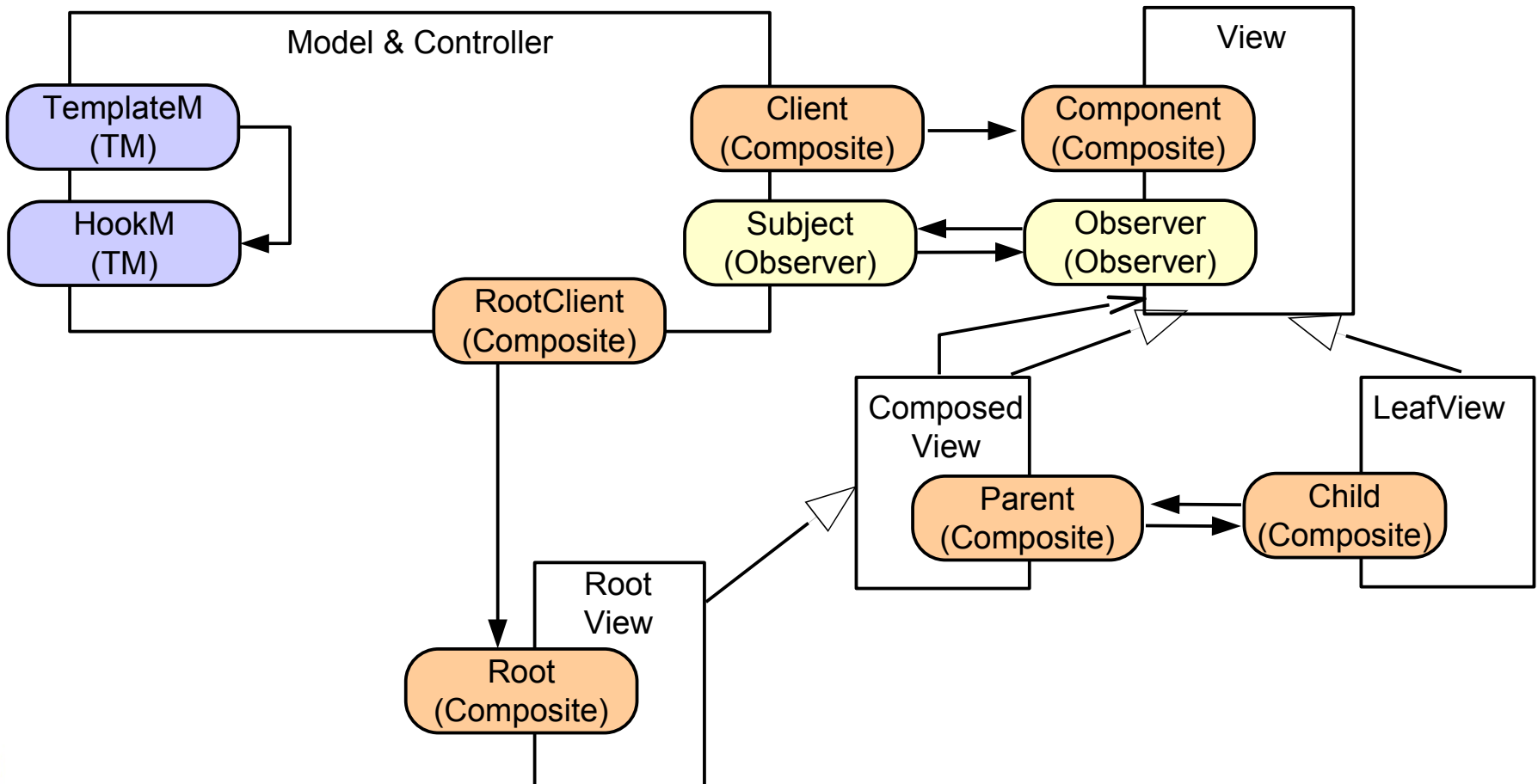
- ▶ Separate classes for almost each role



# Optimized Role-Class Model of MVC

46

- ▶ Merging model and controller class leads to less delegations (i.e., a performance improvement)
- ▶ Strategy pattern is to be exchanged with TemplateMethod pattern



# The End: Summary

47

- ▶ Roles are important for design patterns
  - If a design pattern occurs in an application, some class of the application plays the role of a class in the pattern
- ▶ Role mapping is the process of allocating roles to concrete implementation classes
- ▶ Hence, role mapping decides how the classes of the design pattern are allocated to implementation classes (and this can be quite different)
- ▶ Composite design patterns are based on role model composition
- ▶ Layered frameworks and design patterns can be optimized by role merging