

# 13. Frameworks and Patterns - Framework Extension Patterns

1

Prof. Dr. U. Aßmann  
Software Engineering Group  
Department of Computer Science  
Technische Universität Dresden

Version WS16/17 - Jan 15, 2017

**Lecturer:** Dr. Sebastian Götz

- 1) Extension Object Pattern
- 2) Large Layered Frameworks
- 3) Role Object Pattern
- 4) GenVoca Pattern
- 5) Mixin Layer Pattern



# Literature (To Be Read)

2

- ▶ E. Gamma. **The Extension Objects Pattern**. Conf. On Pattern Languages of Programming (PLOP) 97, ACM. <http://portal.acm.org/citation.cfm?id=273448.273455>
- ▶ Y. Smaragdakis and D. Batory. **Mixin layers: an object-oriented implementation technique for refinements and collaboration-based designs**. ACM Transactions on Software Engineering and Methodology, 11(2):215–255, 2002. <http://dl.acm.org/citation.cfm?id=505148>
- ▶ D. Bäumer, D. Riehle, W. Silberski, M. Wulf. **Role Object**. Conf. On Pattern Languages of Programming (PLOP) 97. <http://hillside.net/plop/plop97/Proceedings/riehle.pdf>
- ▶ 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://dl.acm.org/citation.cfm?id=286951>
- ▶ D. Bäumer, G. Gryczan, C. Lilienthal, D. Riehle, H. Züllighoven. **Framework Development for Large Systems**. Communications of the ACM 40(10), Oct. 1997. <http://dl.acm.org/citation.cfm?id=262804>

# Further Literature

3

- ▶ D. Bäumer. Softwarearchitekturen für die rahmenwerkbasierte Konstruktion grosser Anwendungssysteme. PhD thesis, 1997, Universität Hamburg.  
<http://ediss.sub.uni-hamburg.de/volltexte/1998/12/pdf/Beleg.pdf>
- ▶ JWAM sites
  - <http://www.c1-wps.de/forschung-und-lehre/fachpublikationen/>
  - [www.jwam.de](http://www.jwam.de)
  - <http://sourceforge.net/projects/jwamtoolconstr/>
- ▶ U. Aßmann. Composing Frameworks and Components for Families of Semantic Web Applications. Lecture Notes In Computer Science, vol. 2901, Nov. 2003.  
[http://link.springer.com/chapter/10.1007/978-3-540-24572-8\\_1](http://link.springer.com/chapter/10.1007/978-3-540-24572-8_1)
- ▶ U. Aßmann, J. Johannes, J. Henriksson, and I. Savga. Composition of rule sets and ontologies. In F. Bry, editor, Reasoning Web, Second Int. Summer School 2006, number 4126 in LNCS, pages 68-92, Sept 2006. Springer.  
[http://link.springer.com/chapter/10.1007%2F11837787\\_3](http://link.springer.com/chapter/10.1007%2F11837787_3)
- ▶ Y. Smaragdakis, D. Batory. Implementing layered designs with mixin layers. In Lecture Notes in Computer Science (LNCS) 1998, Springer-Verlag.  
<http://link.springer.com/chapter/10.1007/BFb0054107>

# Goal

4

- ▶ Studying extensible framework hook patterns
  - Understand patterns:
    - Extensions Object,
    - Role Object, and
    - Genvoca
  - See how layered frameworks can be implemented by
    - Role Object and
    - Genvoca
- ▶ Understand these patterns as extension points of frameworks, i.e., framework hook patterns

# Frameworks Must Be Extensible

5

- ▶ Frameworks must evolve, be adapted
- ▶ Idea: instead of variability hooks, use extensibility hooks
  - based on basic extensibility patterns
- ▶ Presented in this lecture:
  - Gamma's Extension Object Pattern (EOP)
  - Layered frameworks
    - Riehle/Züllighoven's RoleObject pattern (ROP)
    - Batory's mixin layer pattern (GenVoca pattern)



# 13.1 The ExtensionObjects Pattern (EOP)

---

6

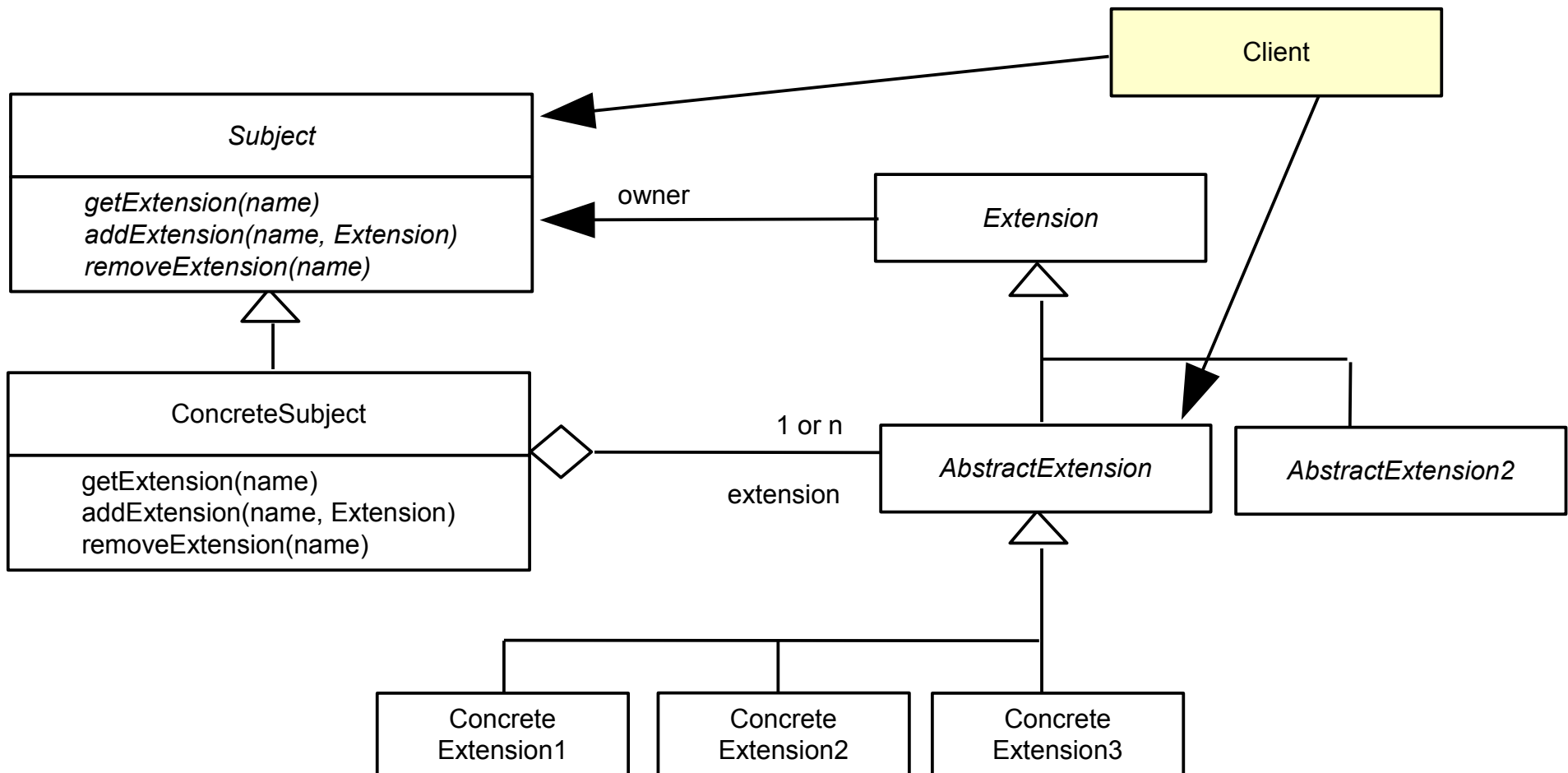
Extensions of Objects, visible for the Client



# Structure of ExtensionObjects

7

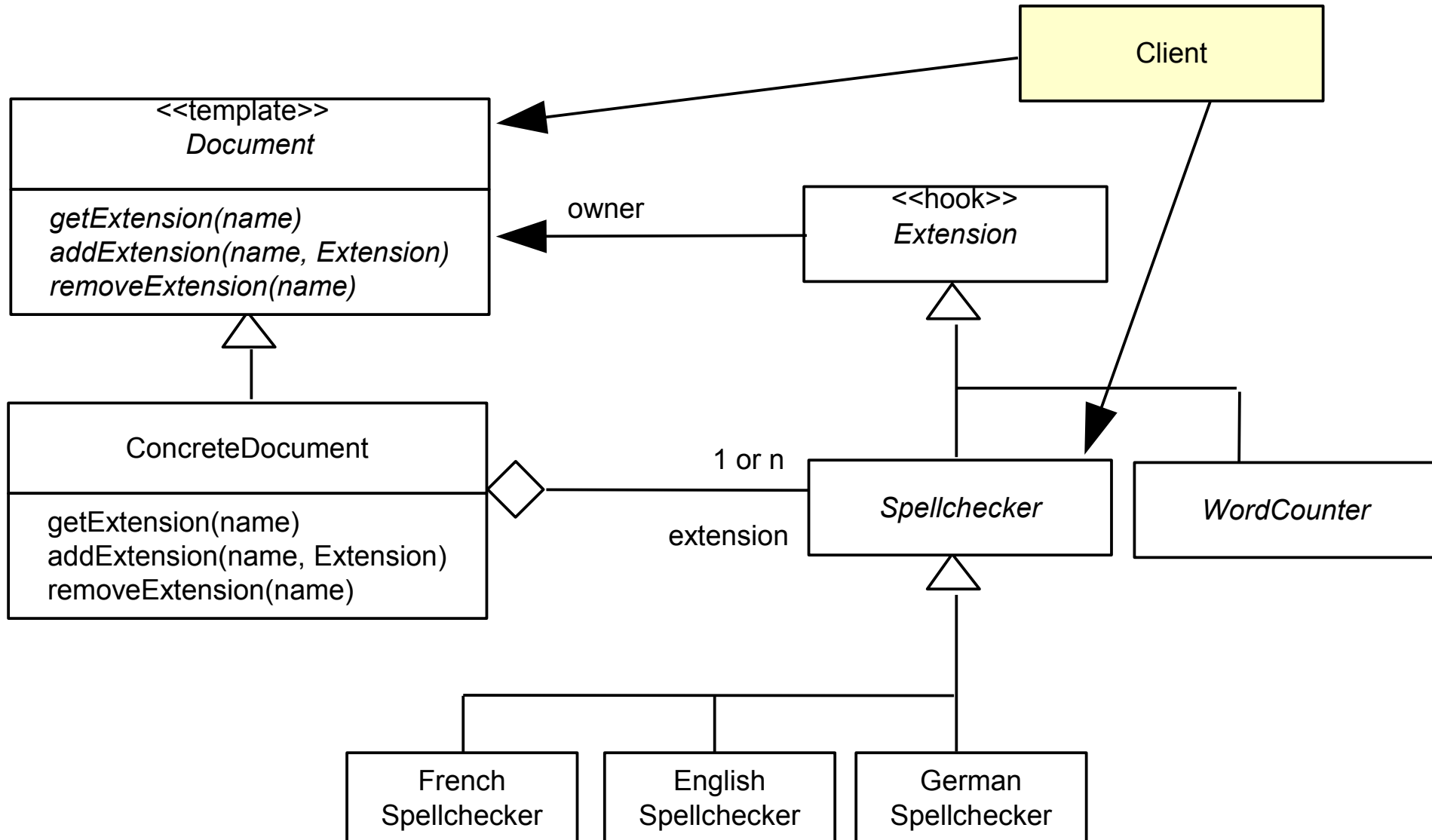
- ▶ Whenever a complex object has non-mandatory parts that can be added, if necessary
- ▶ *Extension* is the base class of all extensions
- ▶ *AbstractExtension* defines an interface for a concrete hierarchy of extension objects
- ▶ Extensions can be added, retrieved, and removed by clients



# Example: Spellcheckers in Document Models

8

- ▶ E.g., OpenDoc or OLE documents





# Discussion of EOP

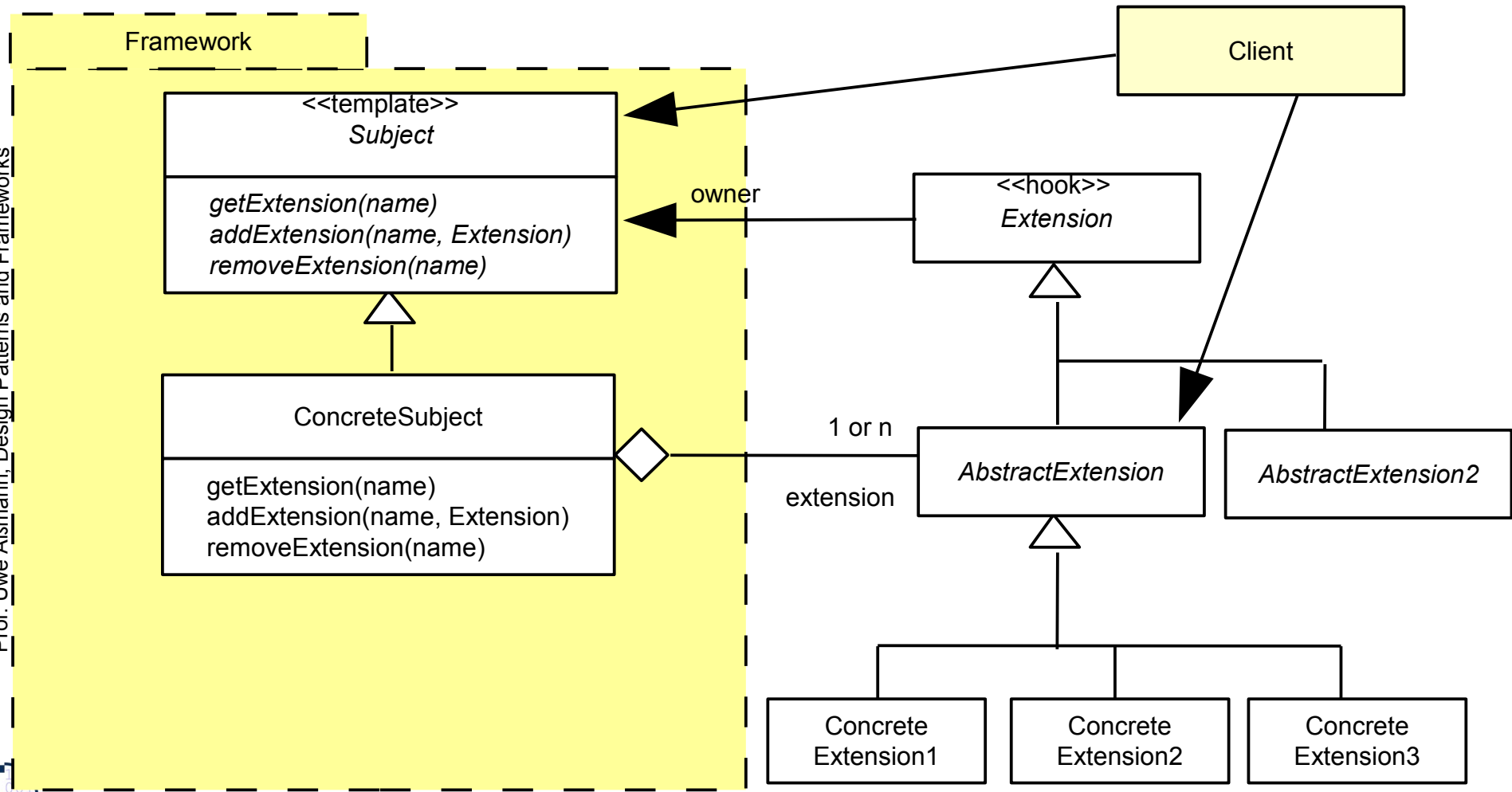
9

- ▶ If there is 1 extension object, naming is not necessary
- ▶ If there are n extension objects, a dictionary (map) has to map names to extension objects
- ▶ Advantages
  - Complex objects can be split into simpler parts
  - Extensions can model (optional) roles of objects
  - Extensions can be added dynamically and unforeseen
- ▶ Disadvantage
  - **Clients have to manage extension objects themselves**, and hence, are more complex
  - Extension objects suffer from the *object schizophrenia* problem: the logical *this* of an extension object is the subject, but the physical *this* is the extension object
- ▶ **Relations to Other Patterns**
  - If many objects of an application have the same roles that are realized by extension objects, ExtensionObjects can be generalized to the Role Object Pattern

# ExtensionObjects at Framework Borders

10

- ▶ Since with EOP, clients have to manage extensions themselves, the use of the template object in the framework does not help to use the hook objects



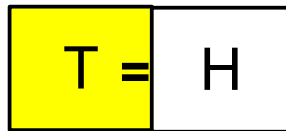
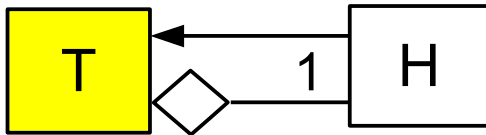
# EOP as Framework Hook Pattern

11

- ▶ Since the hook object is not mandatory, also 1-H=T is a real extensibility pattern for frameworks

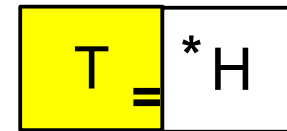
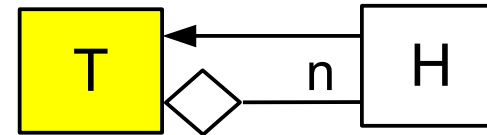
**1-H=T**

T has 1 H part  
T owns H



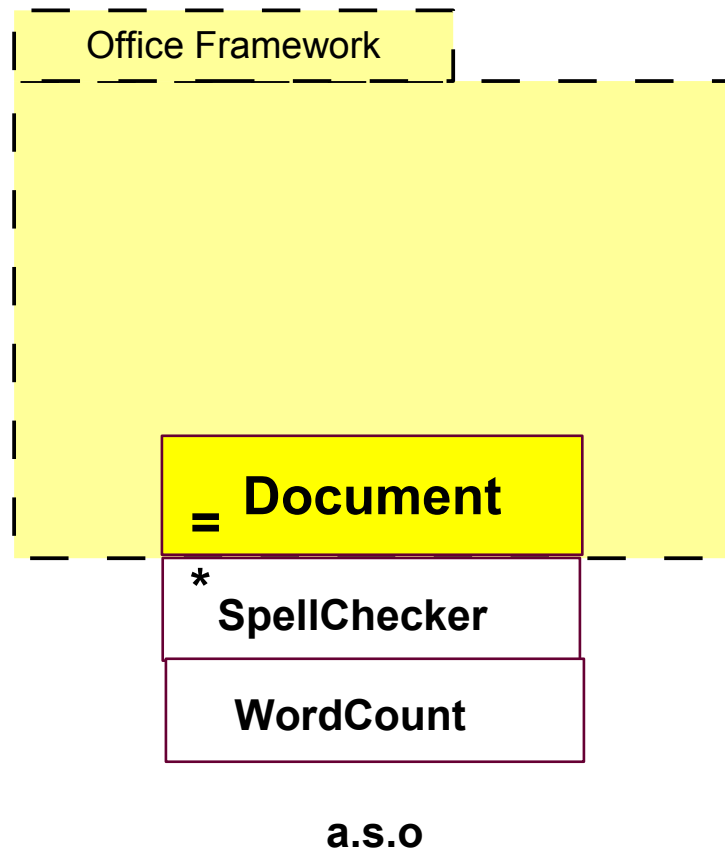
**n-H=T**

T has n H parts  
T owns H parts



# Optional Tools for Documents in an Office Framework

12





# 13.2 Extensibility of Frameworks with Layers

---

13

... with Layered Role Object Frameworks



# Case Study GEBOS

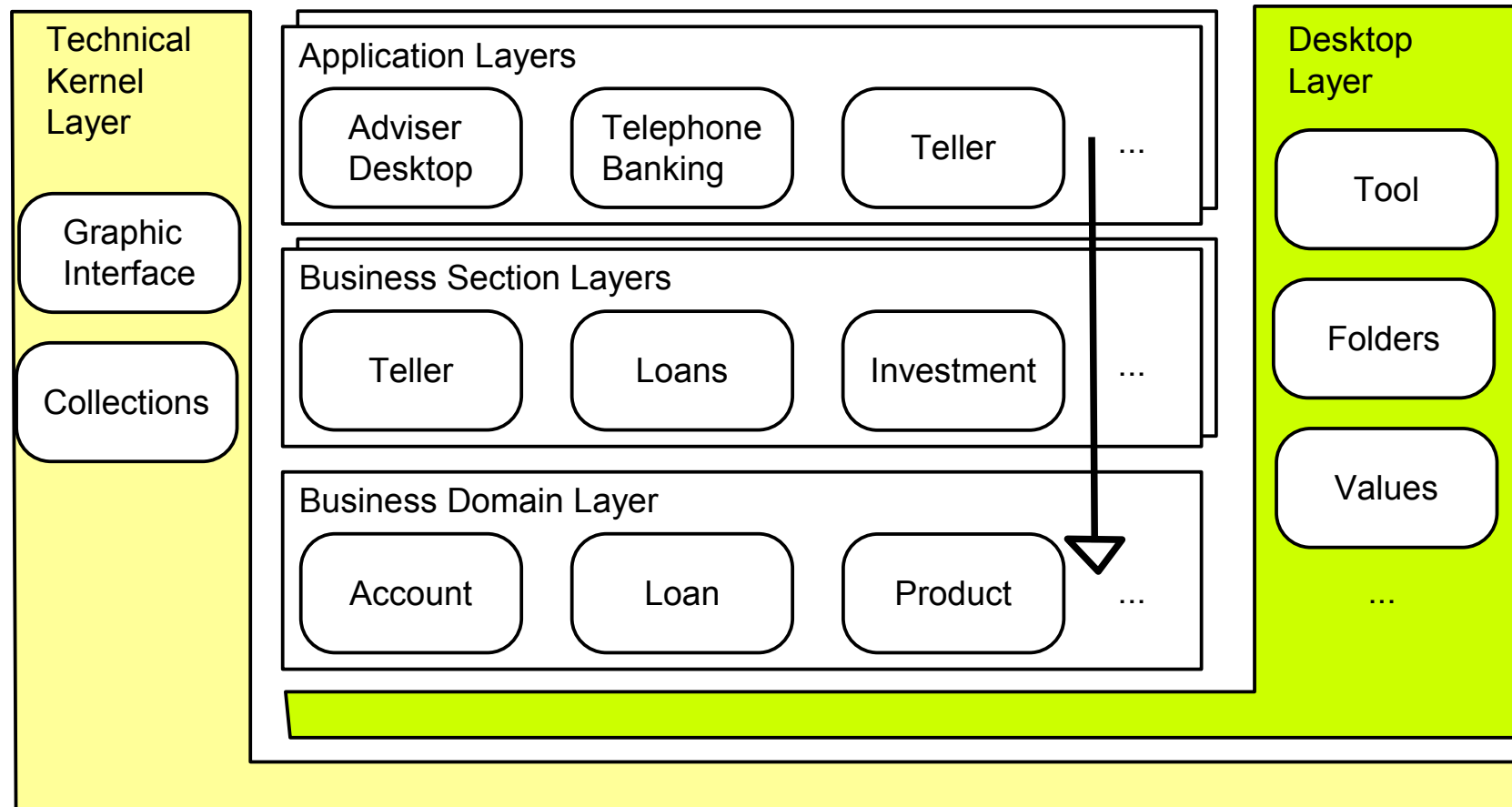
14

- ▶ GEBOS is a banking application for RWG banking group with 450 banks, south of Germany
  - Banking applications, with services: tellers, loans, stocks, investment, self-service
  - 2500 C++ classes, arranged in frameworks, Arranged in layers
- ▶ Concepts of the bank application domain
  - Banks organize themselves in **business sections** (tellers, loans, etc.)
    - Department of specialists that have a certain expertise
  - **Workplace contexts**
    - Service centers offer customers an all-in-one service
    - Services of the business sections
    - Every workplace needs different application systems
  - **Business domain**
    - Business objects such as bill, order, account

# Application Framework Layers

15

- ▶ GEBOS demonstrates that it is advantageous to structure an application framework into layers
  - Application layers, Business Section layers, Business domain layers
  - Desktop Layer, Technical kernel layer



# Layers

16

## ▶ **Technical Kernel Layer**

- Service layer, independent of other layers
- Domain independent, application independent
- Is a framework itself
  - Collections, Middleware, Wrappers
  - Garbage collection, late creation, factories, trace support
- Is a blackbox framework

## ▶ **Desktop Layer**

- Support for interactive workplaces
- Contains a tool construction framework (for the Tools&Materials approach)
- MVC framework, Folder framework, Value framework for business and domain values
  - AccountNumber, ClientNumber, Money etc
- Look and feel, reusable for office domains with GUI applications



# Layers

17

- ▶ **Business Domain Layer** contains the business core concepts: Account, customer, product, value types
  - Shares knowledge for all business sections
  - Think about how to divide the knowledge between business domain layer and business section layers
- ▶ **Business Section Layers**
  - Subclassing business domain and desktop layers, “inherits” knowledge from both
  - Business section concepts: Borrower, investor, guarantor, loan, loan account, tools. Organizational entities and notions
  - Distinguish from business domain
- ▶ **Application Layers**
  - Application concepts
  - Separate from Business Sections, because workplaces need different functionality from different business sections
  - Uses (and inherits) from all other layers

# Goals in Framework Design of GEBOS

18

- ▶ Minimize coupling between frameworks and application systems
  - Frameworks should never be touched when developing an application system
- ▶ Model different facets of business sections, products, and business domain concepts
  - Use role-object design pattern
- ▶ Minimize coupling between the layers
  - Separate concepts from implementation
  - Move implementation to lower layers
- ▶ Achieved with the RoleObject pattern



# 13.3 The RoleObject Pattern

---

19



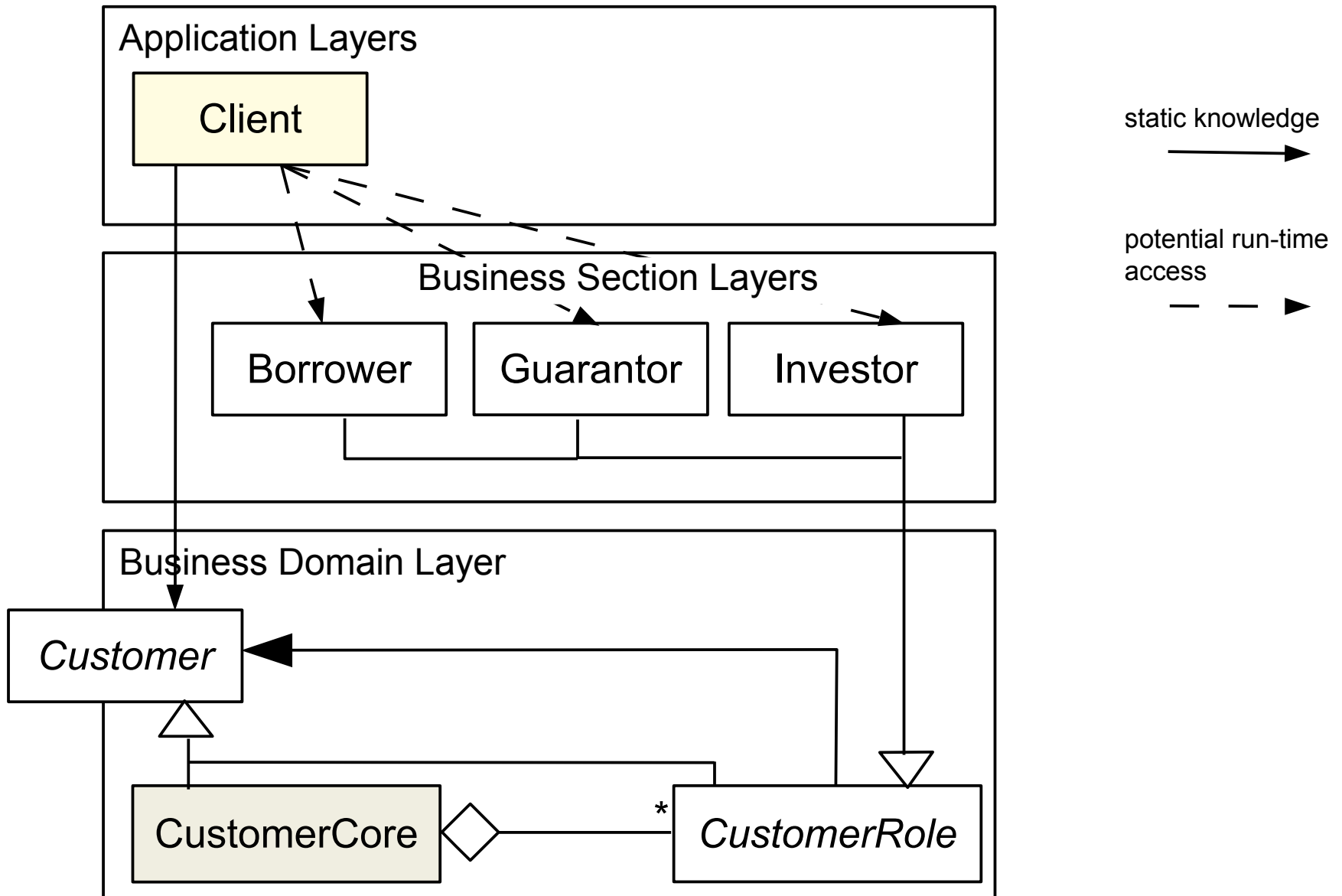
# Framework Extensibility with Riehles Role-Object Layers

20

- ▶ The Role-Object Pattern (ROP) is both a variability and extensibility pattern
  - Realizes the “dispatch on all layers” for application frameworks
  - Can easily be extended with new layers
- ▶ Extension of a core layer (a blackbox framework of core objects) with layers of delegates (role objects)
  - A **conceptual object (complex object, subject)** of the application is split over all layers
  - **Core** and **role** objects conceptually belong together to the **conceptual** object, but distribute over the layers
  - Role objects are *views* on the conceptual object

# Riehle/Züllighovens Role Object Pattern (ROP)

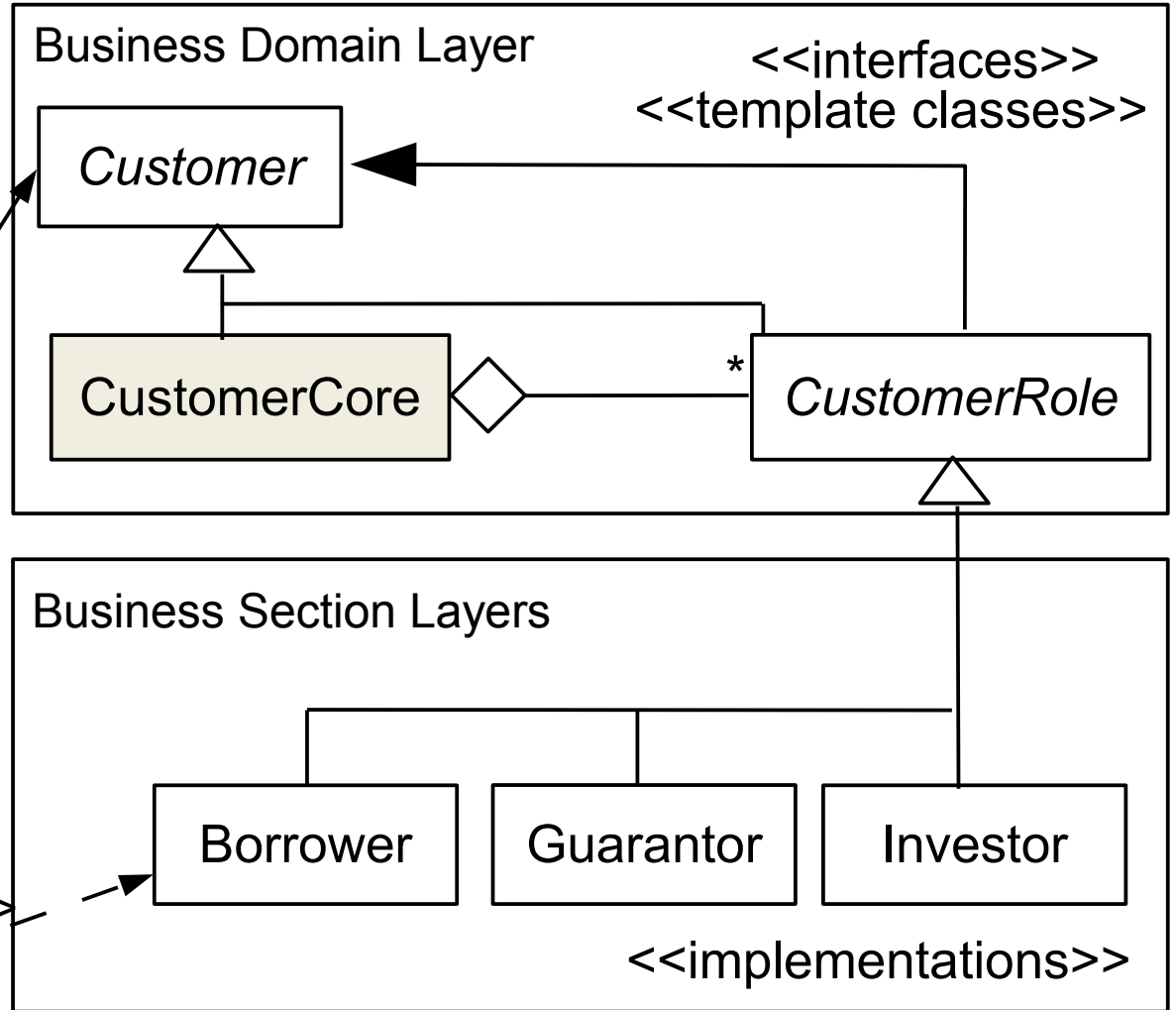
21



# Role Object Pattern with Inheritance Drawn Upwards

22

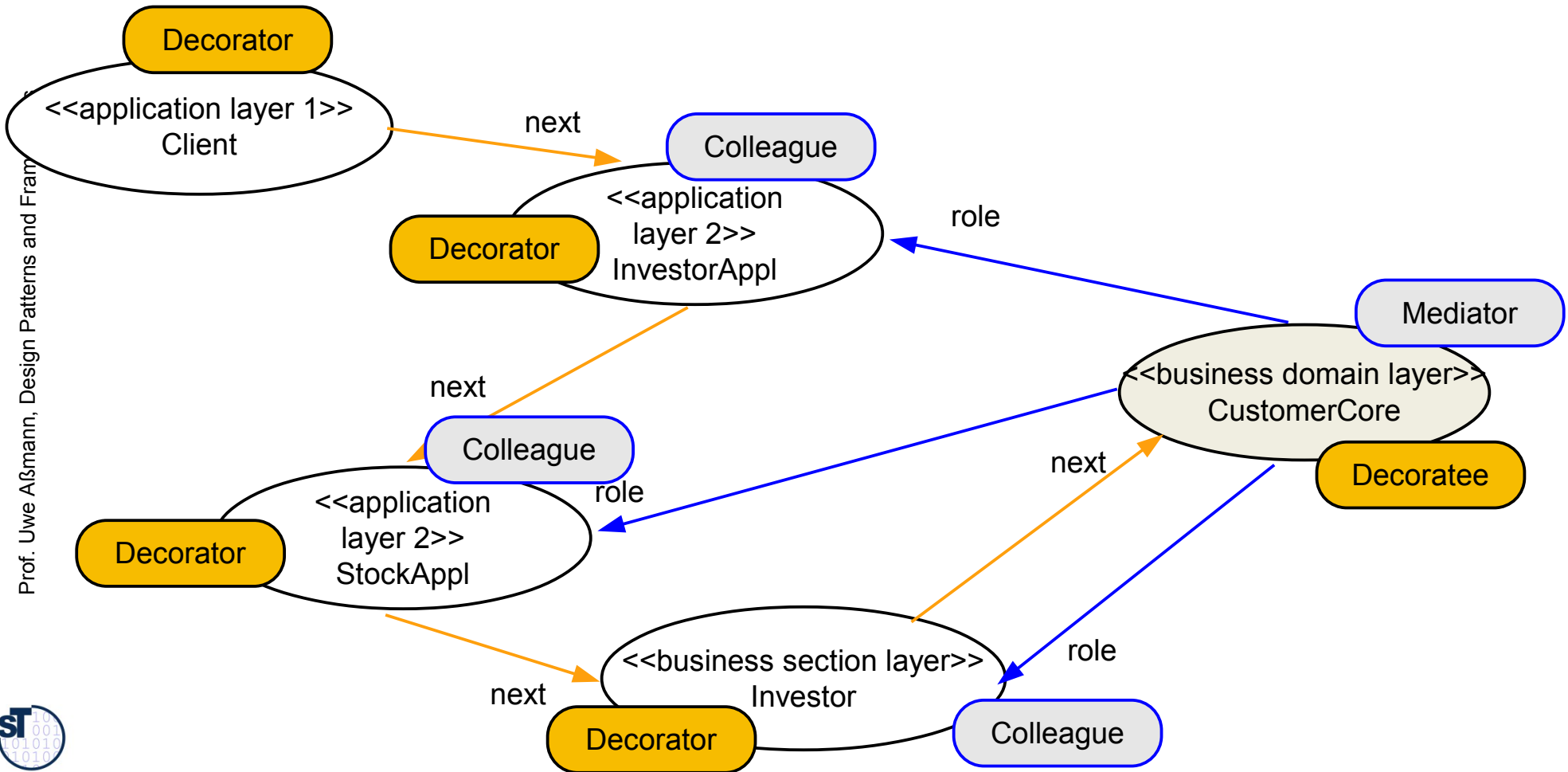
Juве Aßmann, Design Patterns and Frameworks



# Run-Time Structure: Deep Roles

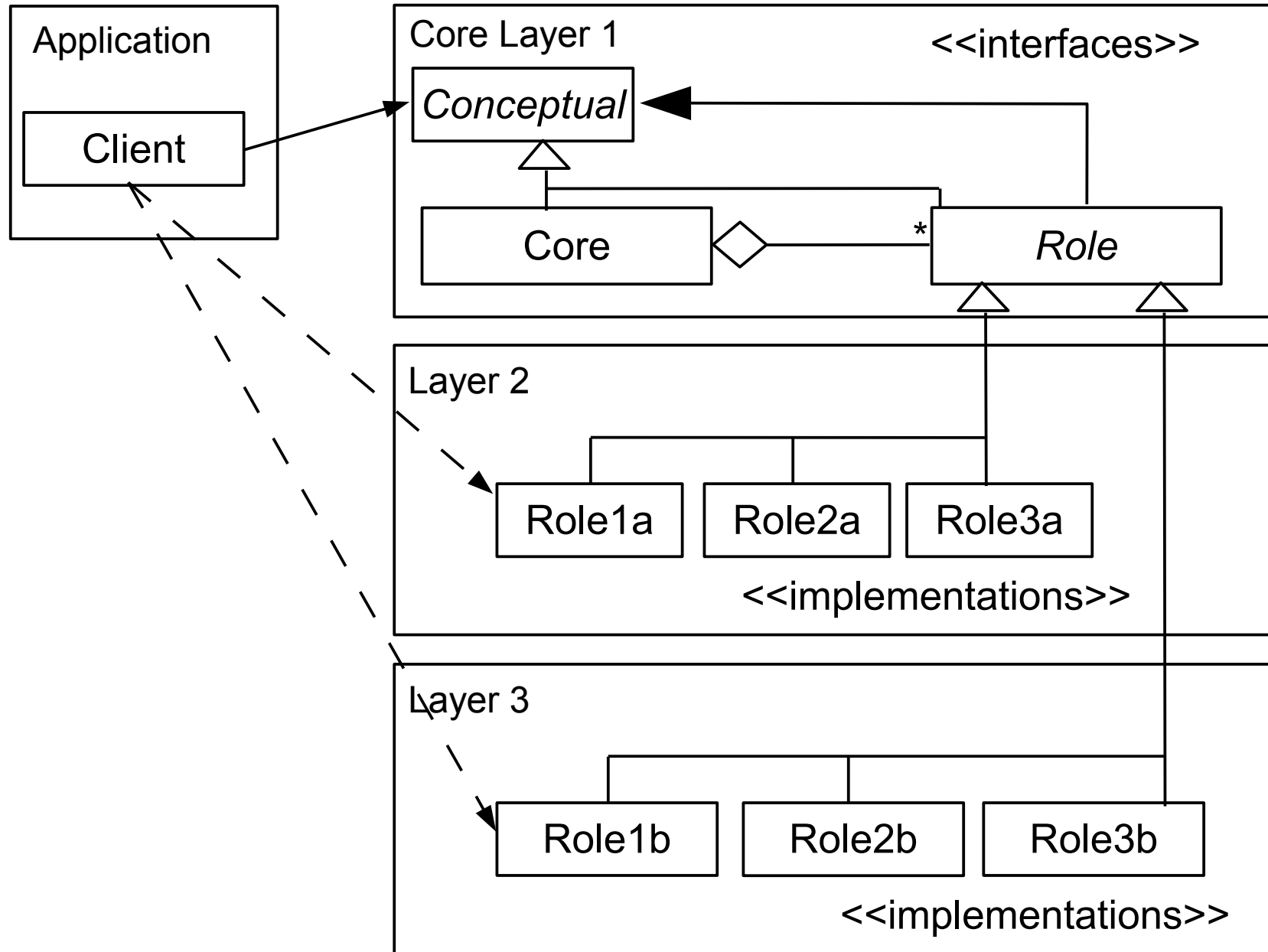
23

- ▶ At runtime, RoleObjects pass service requests (queries) on to the core
  - RoleObjects can be stacked in a Decorator chain (**deep roles**)
- ▶ The core knows all RoleObjects, and distributes requests (Mediator)
  - The core manages the RoleObjects in a *map* that can be dynamically extended



# Riehle/Züllighovens Role Object Pattern Abstracted ("Deep Roles")

24

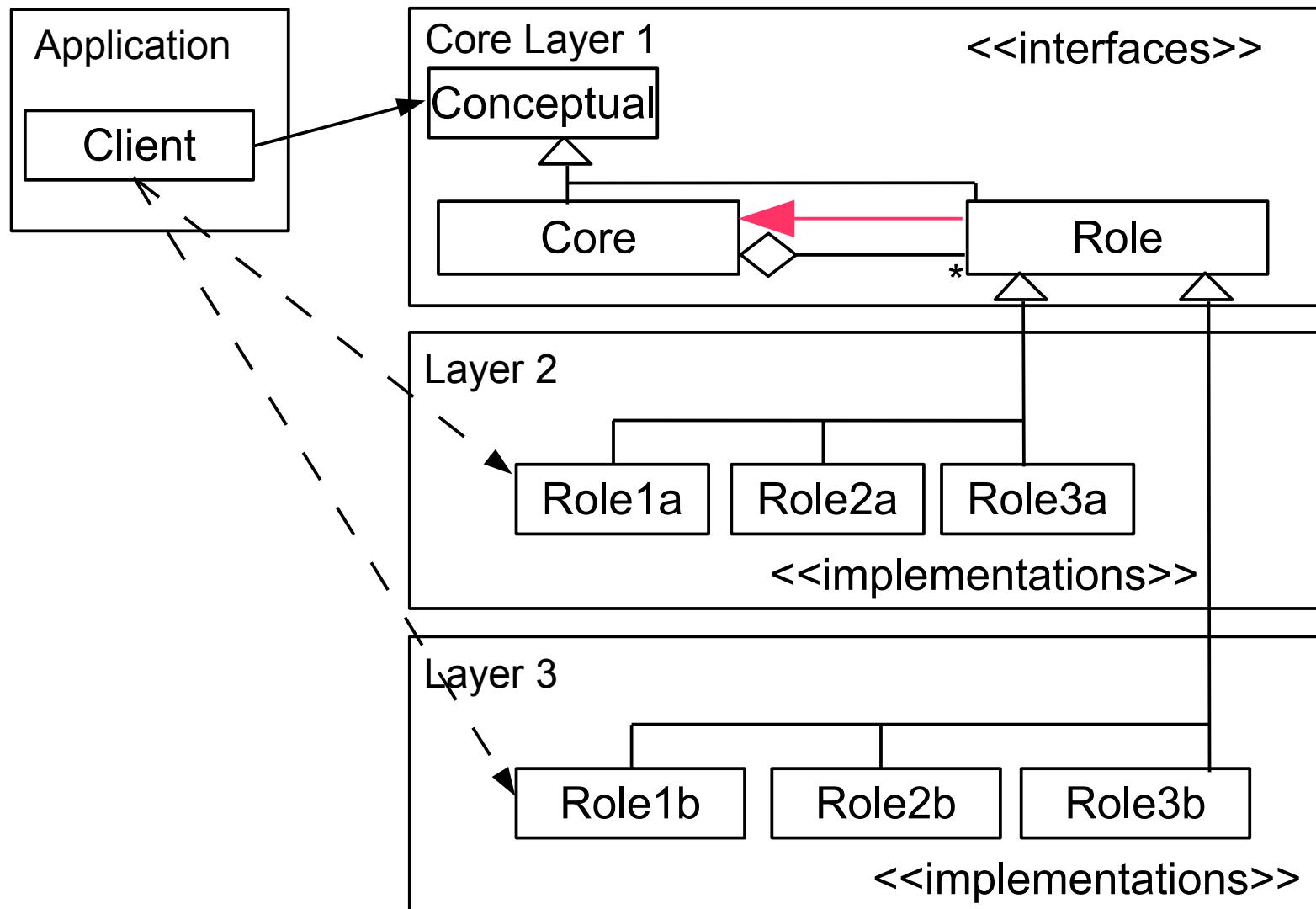




# Riehle/Züllighovens Role Object Pattern Variant 2 ("Flat Roles")

25

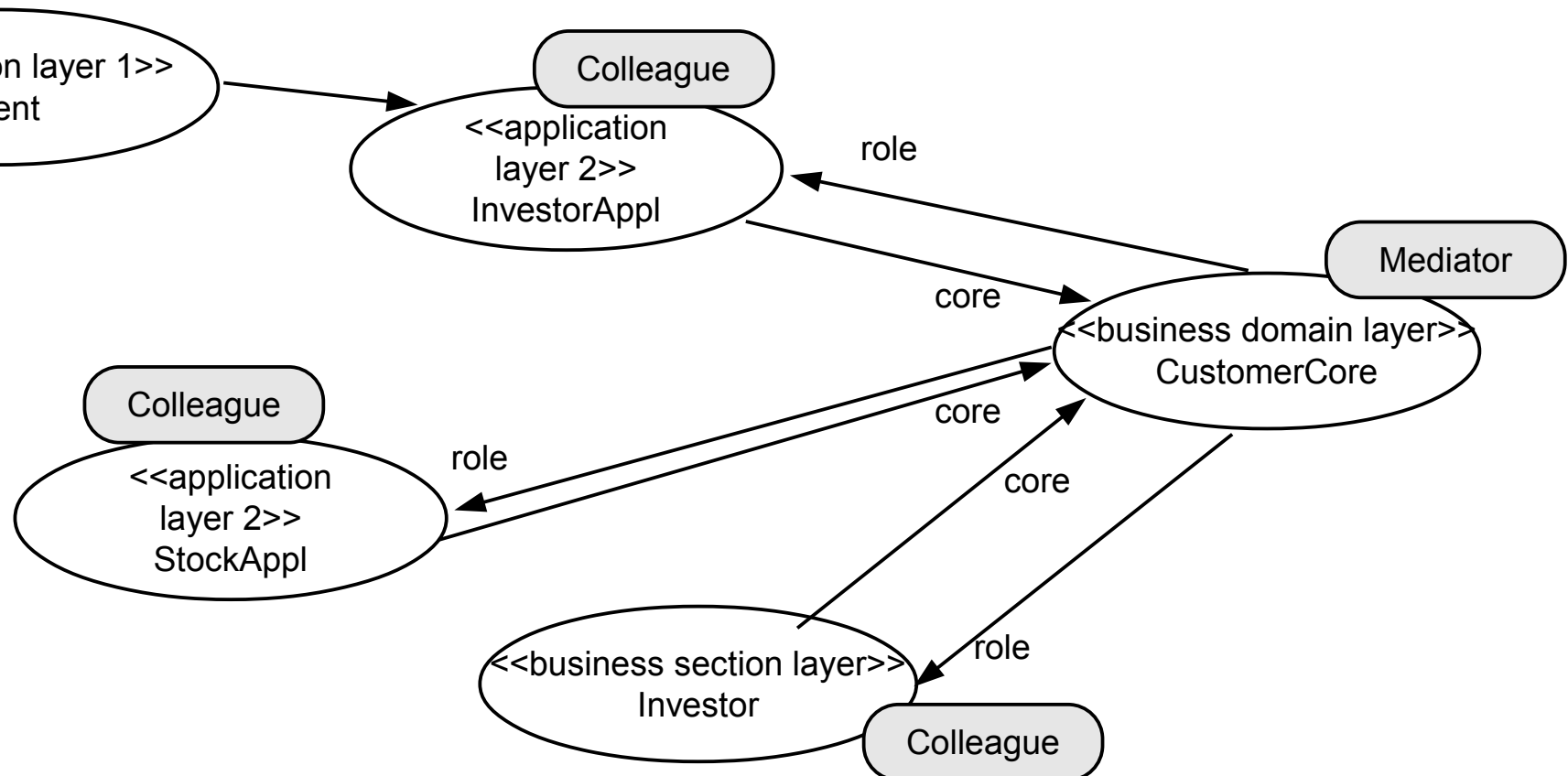
- ▶ Variant 2 has no Decorator; roles only know cores



# Variant "Flat Roles": Run-Time Structure

26

- ▶ At runtime, RoleObjects pass service requests (queries) on to the core
  - RoleObjects can be stacked in a Decorator chain
- ▶ The core knows all RoleObjects, and distributes requests (Mediator)
  - The core manages the RoleObjects in a *map* that can be dynamically extended
- ▶ At runtime, RoleObjects pass service requests (queries) on to the core
  - RoleObjects are directly linked to the core (**flat roles**)



# Run-time Behavior of ROP

27

## ▶ Change of role:

- Different Role Objects may belong to the same role type (e.g., working for multiple companies)
- Over time, the role object for a player may change
- This expresses states of the player in the application
  - E.g., Borrower --> UnsafeBorrower --> TrustedBorrower

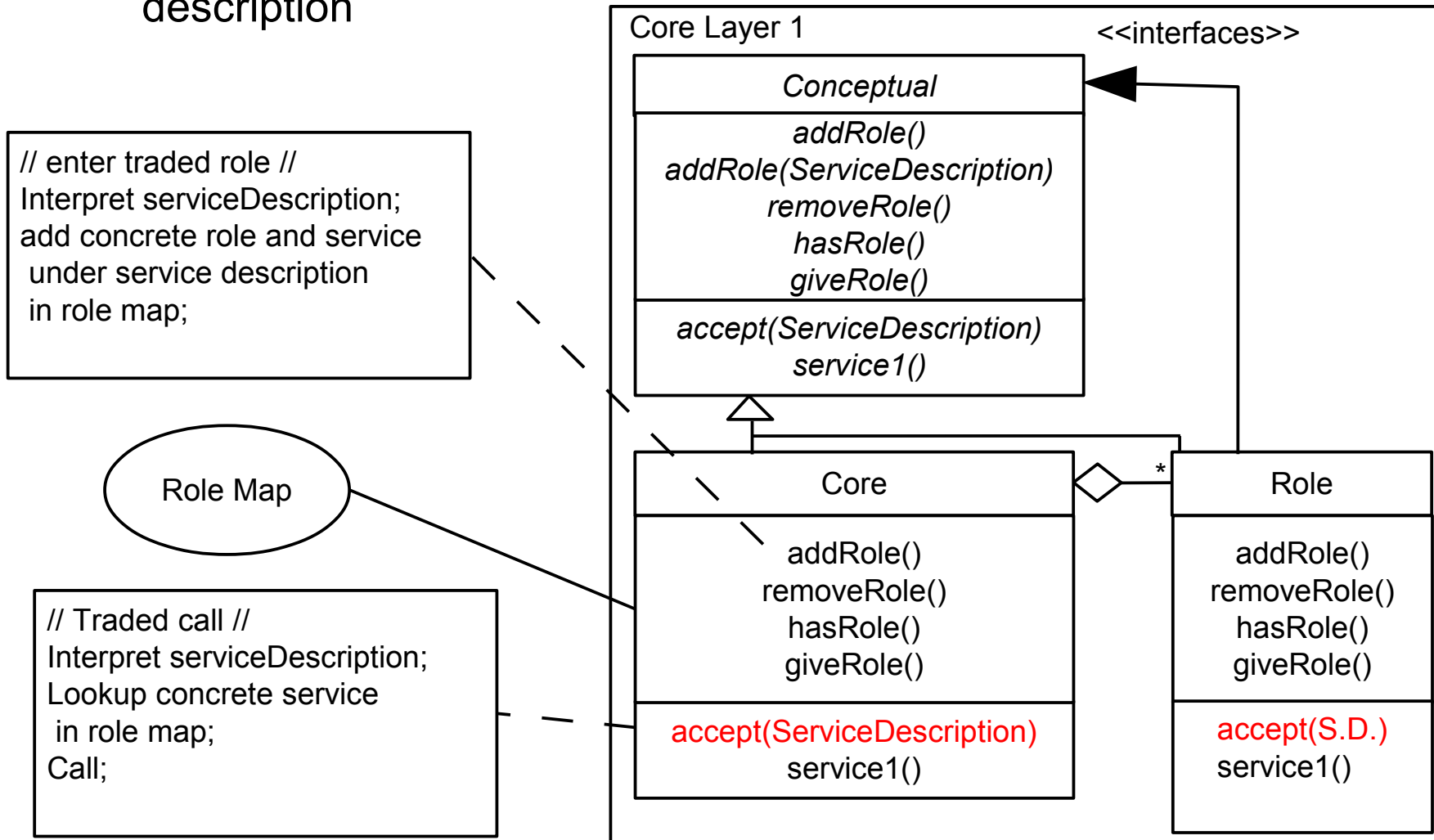
## ▶ Roles are **created on-demand**

- In the beginning, the Subject is *slim*, i.e., carries no roles.
- At service requests, the core creates roles and enters them in the role map

# Variant 3: Core Layer with Traded Call

28

- ▶ To add services dynamically (beyond the service interfaces in the conceptual object), add a *trader* to the core
  - A **trader** is a method that interprets a service request based on a service description



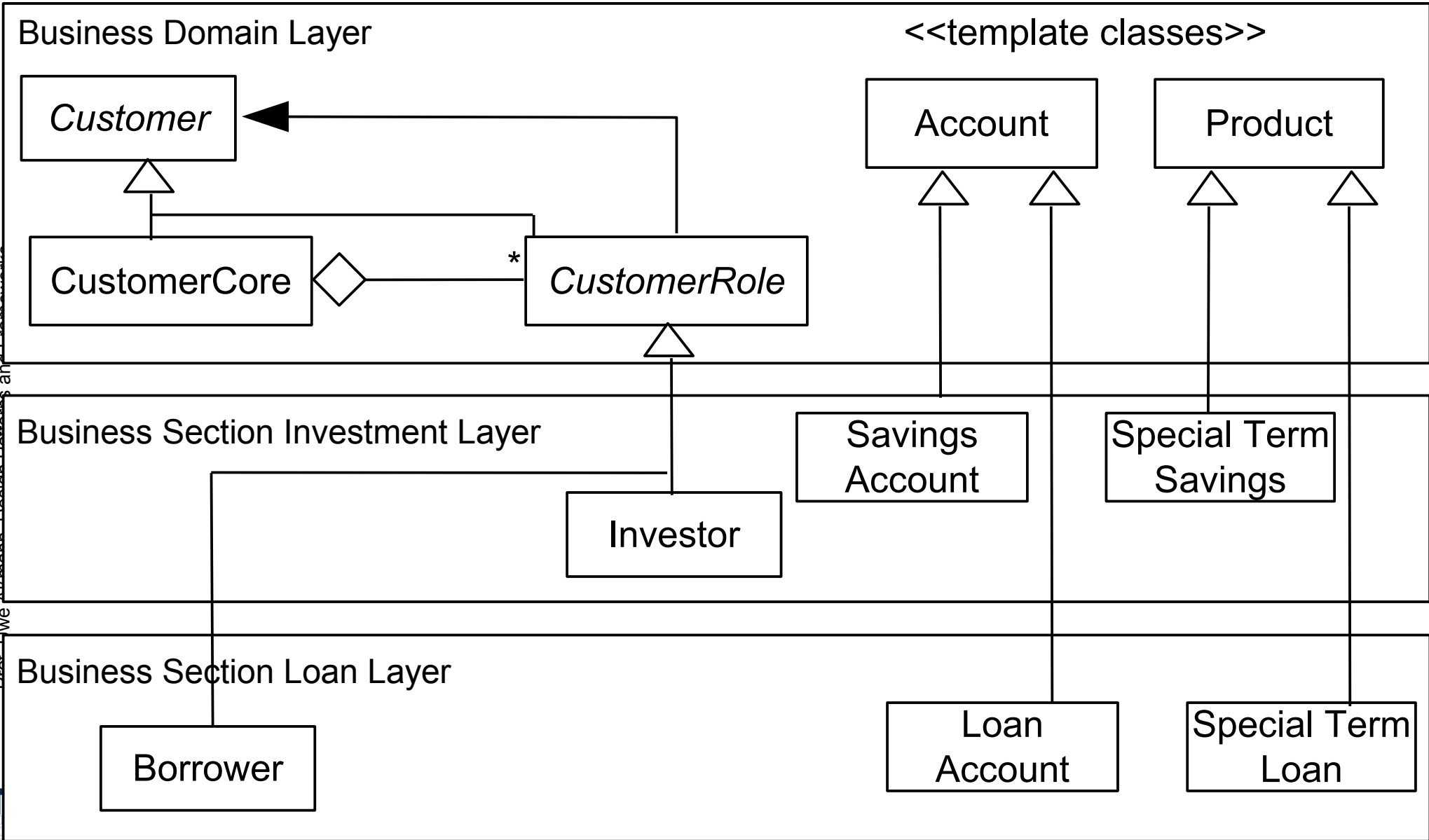
# RoleObjectPattern and Other Patterns

29

- ▶ ROP is not only a Decorator
  - It is based on 1-H $\leq$ T, i.e., 1-ObjectRecursion
    - All role objects inherit from the abstractum
  - Remember, 1-ObjectRecursion based patterns lend themselves to extension
  - And 1-H $\leq$ T framework hook patterns provide extensible frameworks
  - 1:n relationship between core and role objects
  - Role objects decorate the core object, and pass requests on to it

# Role Object Pattern Vs Inheritance (White-Box Framework Layers)

30

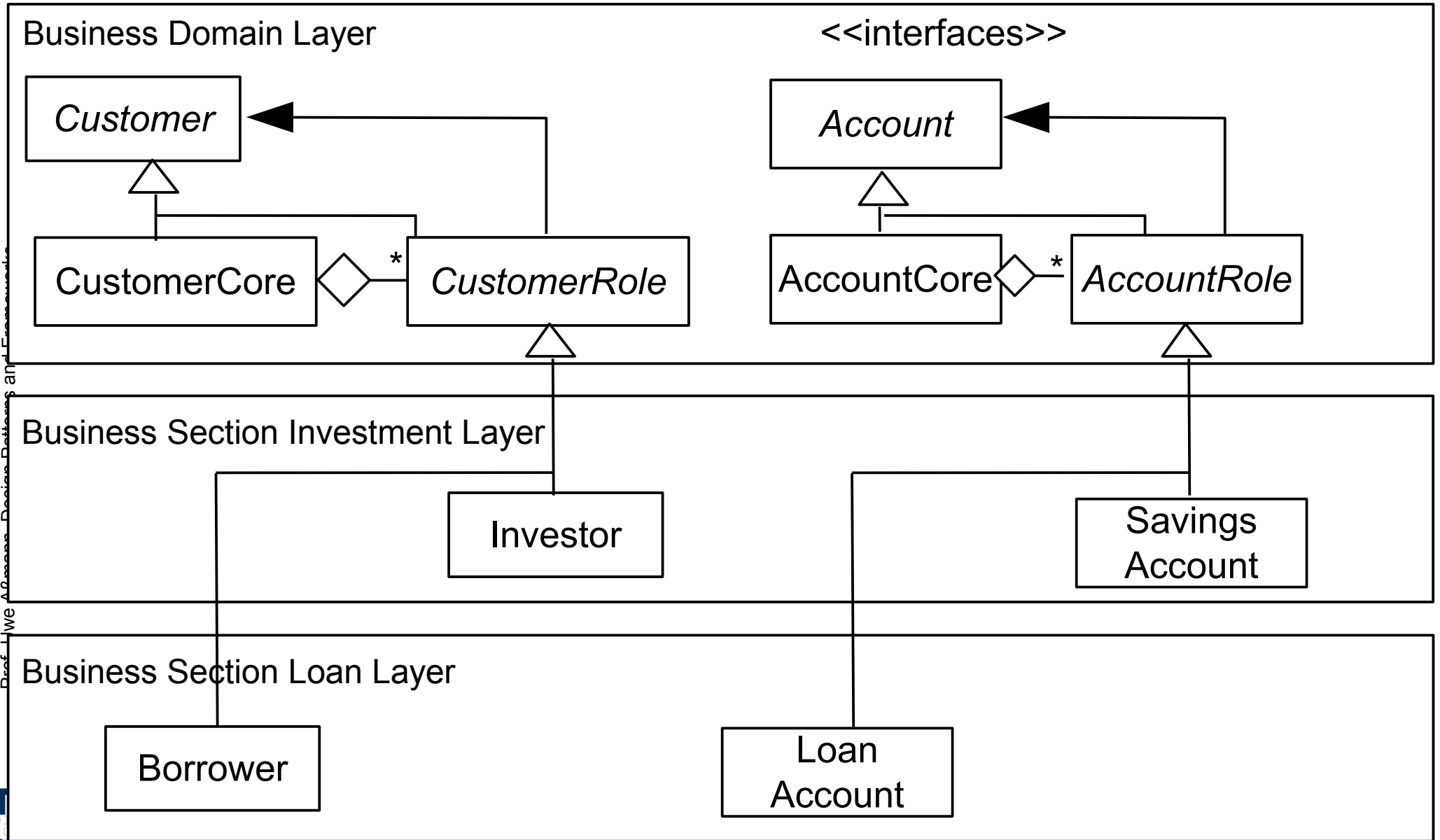


Dr. We Ahmed, Decision Patterns and Frameworks



# Role Object Pattern Vs Inheritance (White-Box Framework Layers)

31



Dr. We Ahmed, Decision Patterns and Frameworks



# Comparison of Role Objects with Inheritance

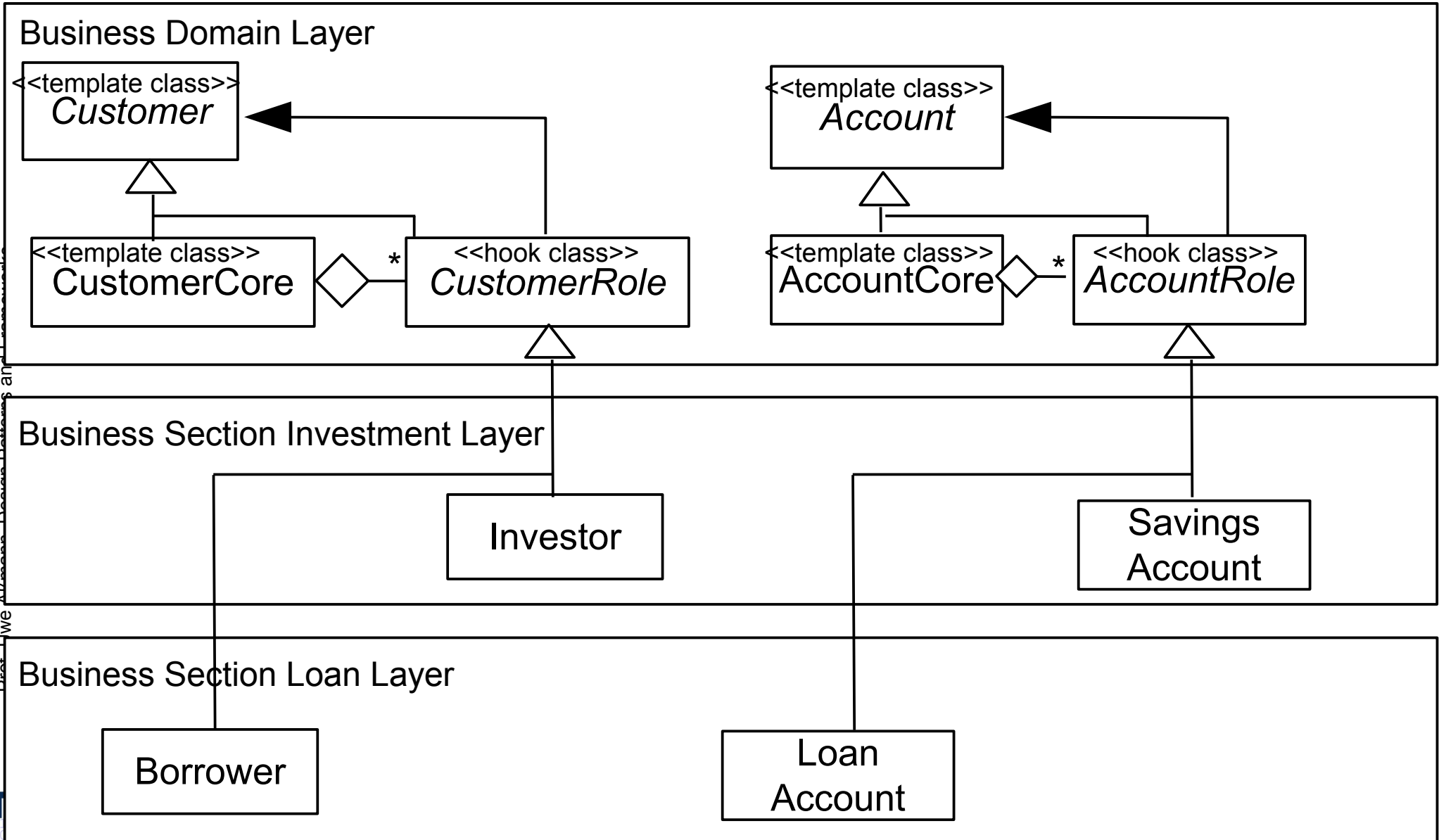
32

- ▶ Simple inheritance has one instance of a subclass at a time
  - Subclass can change over time (polymorphism)
- ▶ The role object has many of them at the same time
  - All role objects can change (role polymorphism)
- ▶ Only changes in the base layers (technical, presentation, business) affect other layers
  - Changes in the business section layers do not affect the business domain layers
- ▶ The relation of core and role objects is a special form of part-of (combined with inheritance)



# Role Object Pattern with Template and Hook Stereotypes

33



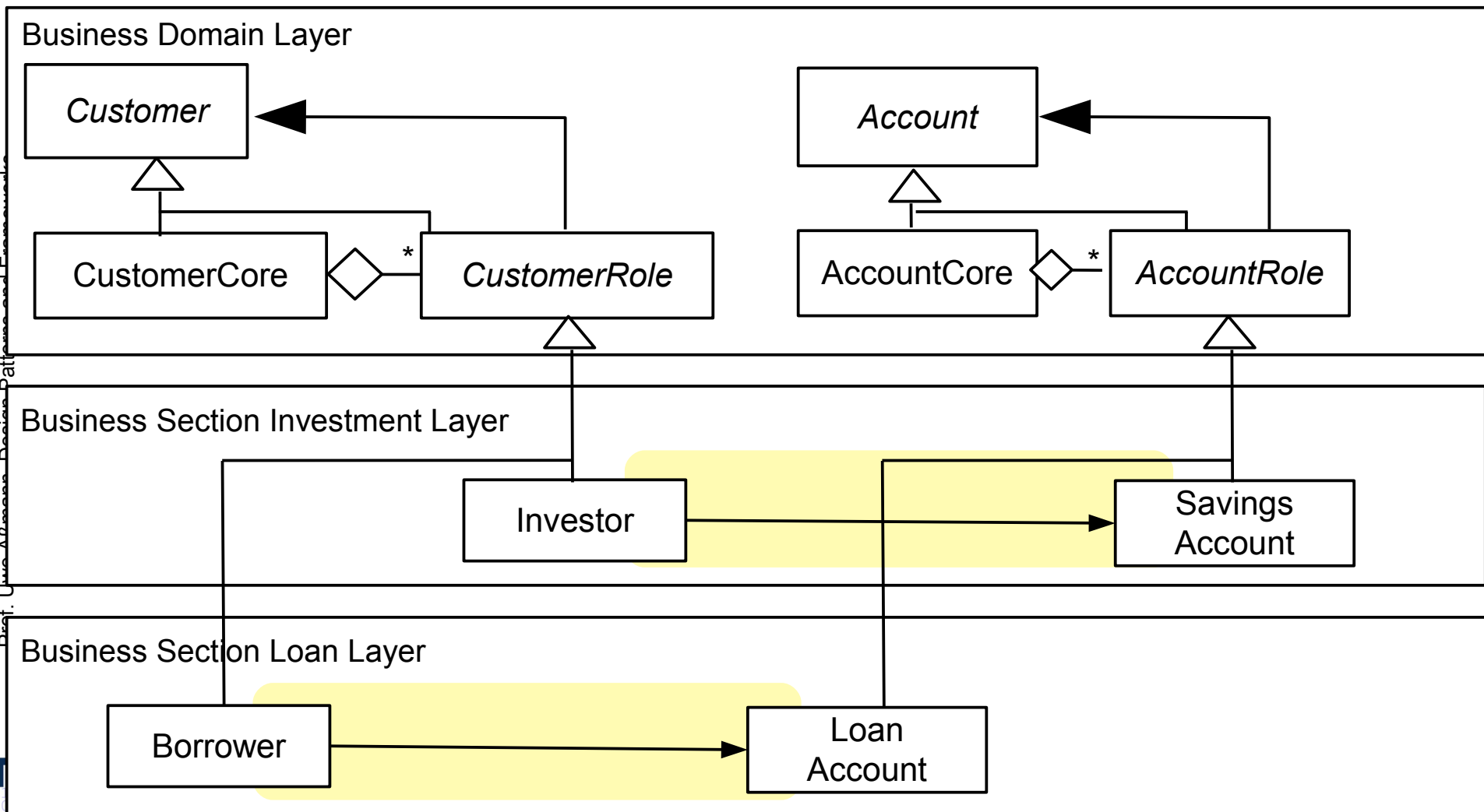
Prof. Dr. We Al-Deeb, Decision Patterns and Frameworks



# Role Object Pattern and Role Models on Role Layers

34

- ▶ Usually, roles of one subject talk to other roles of another subject on the same layer (within a role model)
- ▶ Cores never talk to each other directly



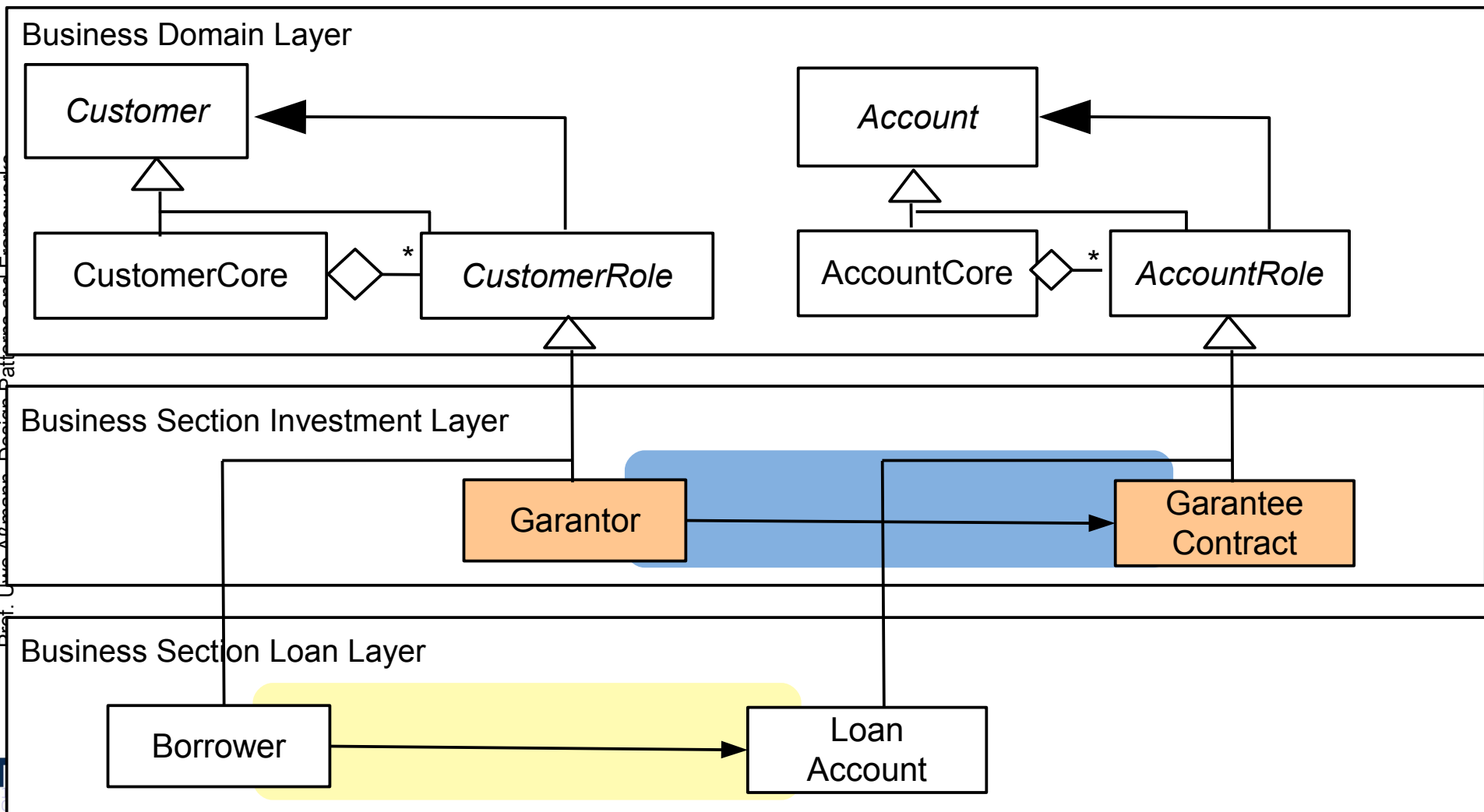
Dist. Univ. Alameda, Decision Patterns and Frameworks



# Switching Variable Role Layers

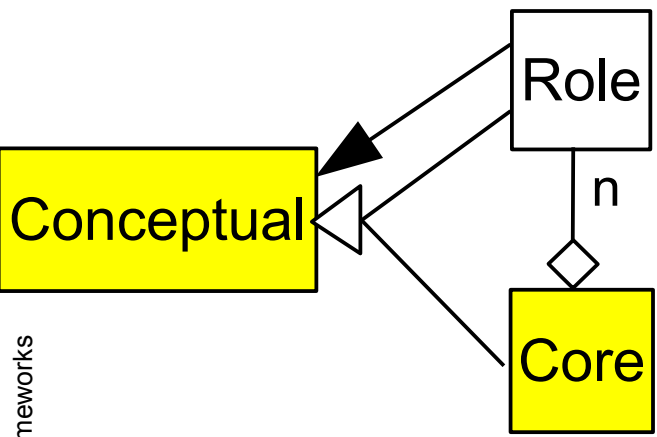
35

- ▶ At run time, entire role models on role layers can be exchanged (**variable role layers**)

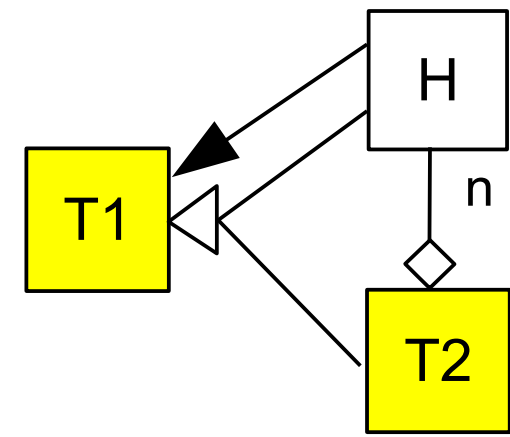


# Riehle/Züllighovens Layer Pattern As Framework Hook Pattern

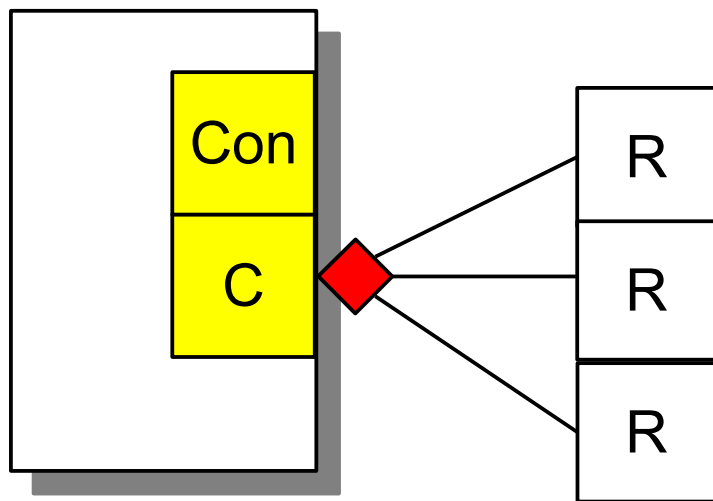
36



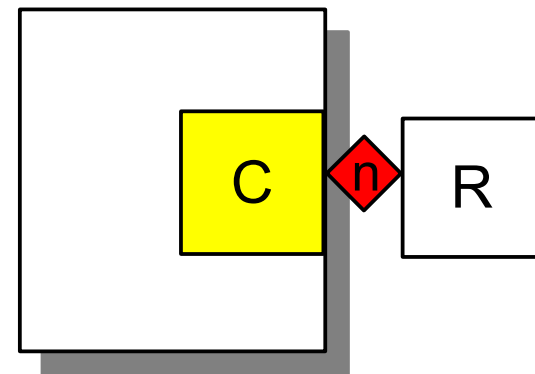
**n-TrH**  
 T2 has H parts  
 H and T2 inherit from T1  
 H knows T1



## Core-Role-Pattern



## n-TrH mini-connector



Special partOf

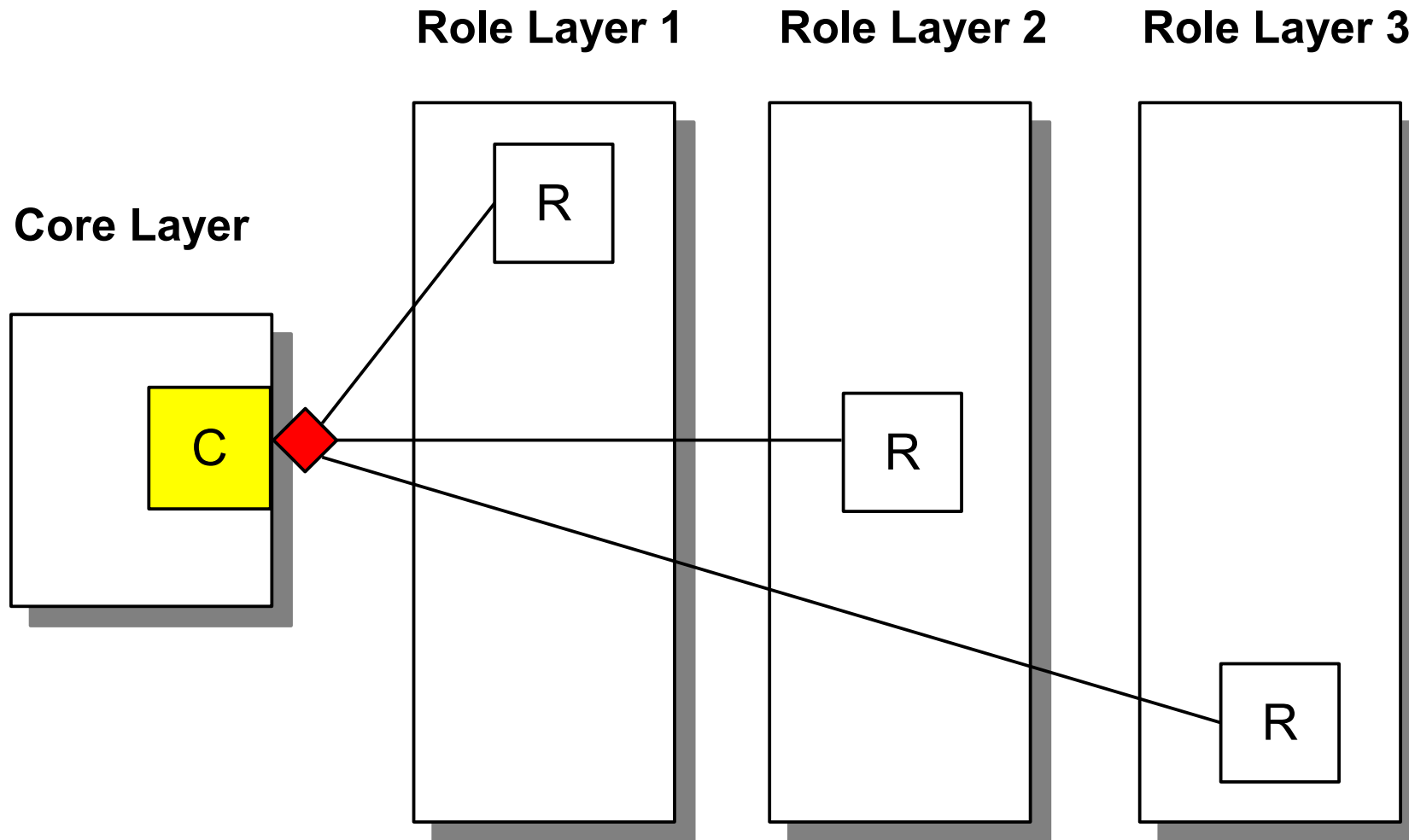
# ROP Ensures Extensibility

37

- ▶ The ROP lends itself not only to variability, but also to **dynamic extensibility**
  - If a framework hook is a role object pattern, the hook can be extended in unforeseen ways **without** changing the framework!
  - New layers of the application or the framework can be added at design time or runtime
- ▶ Powerful extension concept with ROP-Trader
  - Whenever you have to design something complex which should be **extensible in unforeseen ways**, consider Role Object

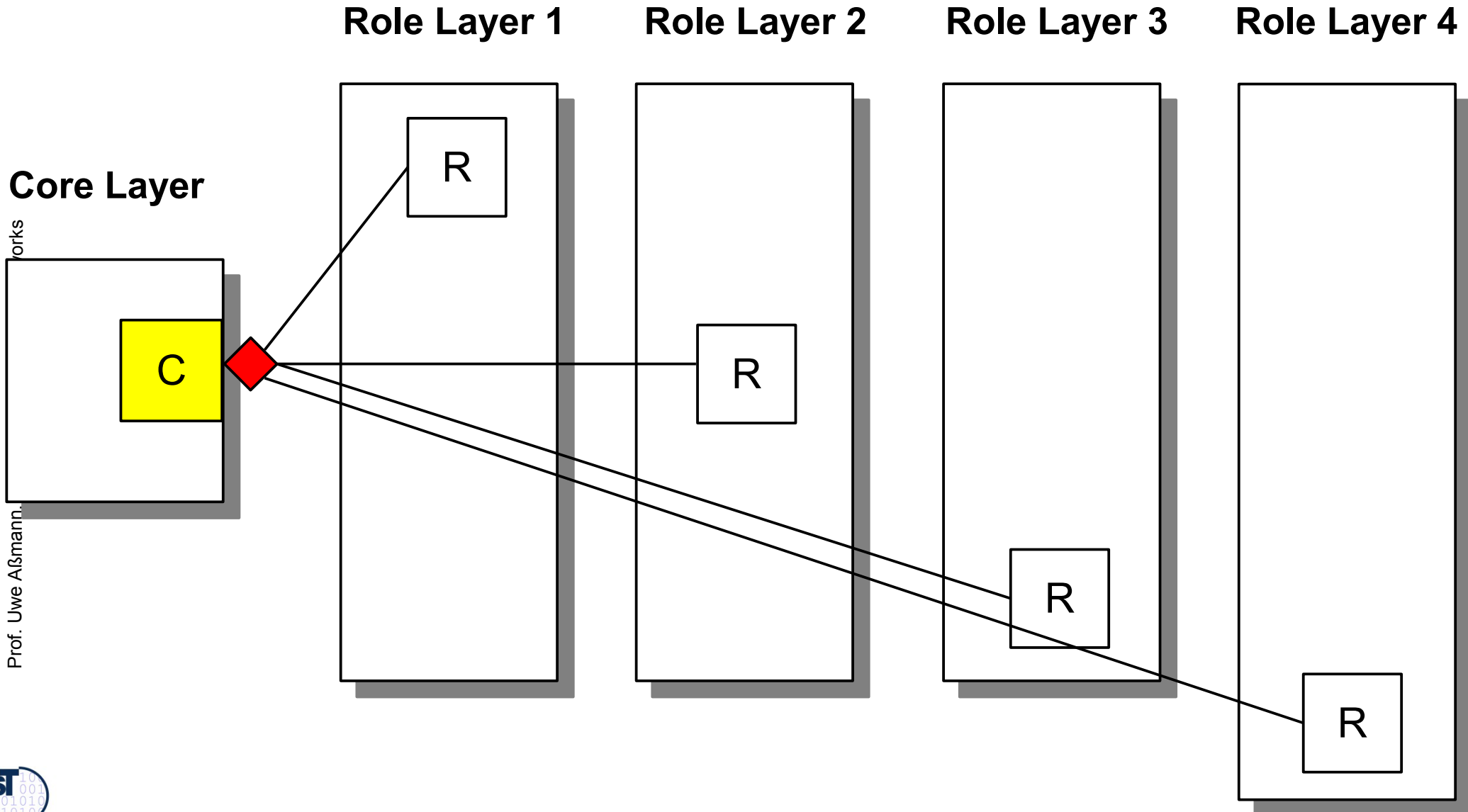
# Riehle/Züllighovens Layered Role Object Framework

38



# Extension in Layered Role Object Frameworks

39



Core Layer

Role Layer 1

Role Layer 2

Role Layer 3

Role Layer 4

R

R

R

R

orks

Prof. Uwe Aßmann

# ROP Can Implement Dimensions That Are Not Independent

40

- ▶ The role objects implement dimensions
  - Core object implements primary dimension
  - Role object secondary dimension
- ▶ Role objects realize *one conceptual object*, instead of a role model crosscutting several conceptual objects
  - Facets are independent dimensions of a conceptual object
  - Every dimension can be varied independently
- ▶ Comparison to the standard implementation of facets by Multi-Bridge (see Chapter “Simple Extensibility”)
  - Multi-Bridge has no inheritance between ConceptualObject, Core and Role
  - Multi-Bridge suffers from object schizophrenia, ROP can implement “this()” on itself without object schizophrenia
  - Calls to the role are not dispatched to the LogicalObject
  - Bridges must not inherit from each other, RoleObjects can



# Benefit of Layered ROP Frameworks

41

- ▶ Implements conceptual objects with layered dependent dimensions
  - Not only independent dimensions
- ▶ Together with layering,
  - Easily extensible
  - Enormous variability
  - Simple structure for extensible product line architecture results
- ▶ For instance: **Layered Frameworks for Business Software**
  - Dispatch on all layers is necessary
    - Implementation without multimethods (in standard languages) *very hard*. Only CLOS, Cecil, and MultiJava are good here
  - That is one reason why business frameworks are so hard
    - SanFrancisco business framework of IBM didn't make it though a dynamic extensibility pattern
    - That's also why these applications are so expensive

# 13.4 The GenVoca Pattern, Mixin Layers, and Layered Mixin Frameworks

42



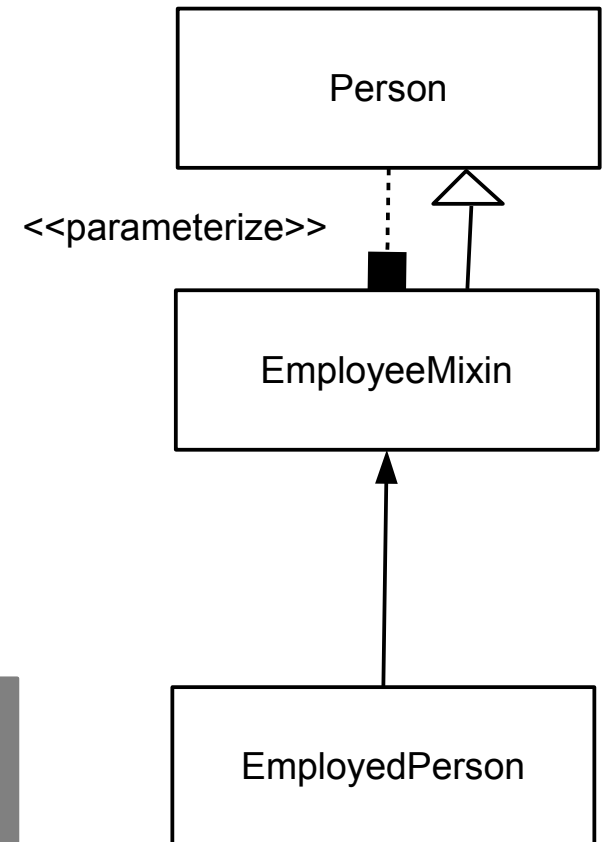
# The Mixin Concept

43

- ▶ A **mixin** is a partial class, for an extension of another class
- ▶ A **mixin-base** is a class with a generic super class, a mixin parameterizes this
- ▶ Some languages have mixins (Scala, C#, Eiffel); otherwise, mixins can be expressed as class fragments that can be parameterized with a superclass (C++)
- ▶ Mixins can implement (static) roles and facets

```
template <class S>
class EmployeeMixin extends S {
    // class extension..
    Salary salary;
    Employer emp;
}
```

```
EmployeeMixin<Person>    employedPerson;
EmployeeMixin<Student>  employedStudent;
EmployeeMixin<Puppy>    employedPuppy;
```



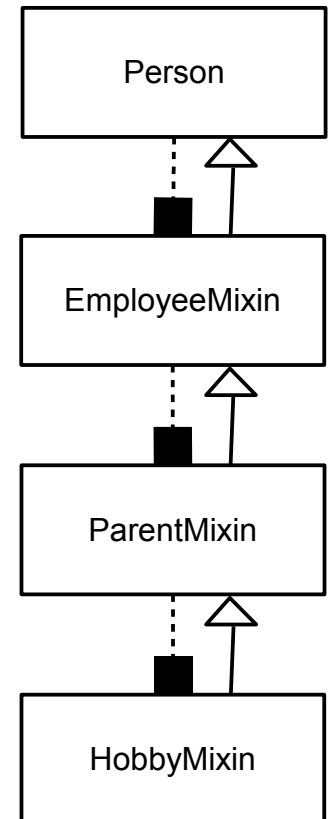
# The GenVoca Pattern

44

- ▶ If several mixin parameterizations are nested, the GenVoca pattern results [Batory]

```
template <class S> class EmployeeMixin extends S {
    Salary salary;
    Employer emp;
}
template <class S> class ParentMixin extends S {
    Child child;
    Money kindergeld;
}
template <class S> class HobbyMixin extends S {
    Hobby hobby;
}
// Persons composed with GenVoca pattern
HobbyMixin<ParentMixin<EmployeeMixin<Person>>>> assmann;
EmployeeMixin<ParentMixin<HobbyMixin<Person>>>> assmann2;

// Have assmann and assmann2 the same type?
```



# GenVoca Variations

45

- ▶ When different variants exist for an “abstraction layer”, parameterizations express configurations of a product line

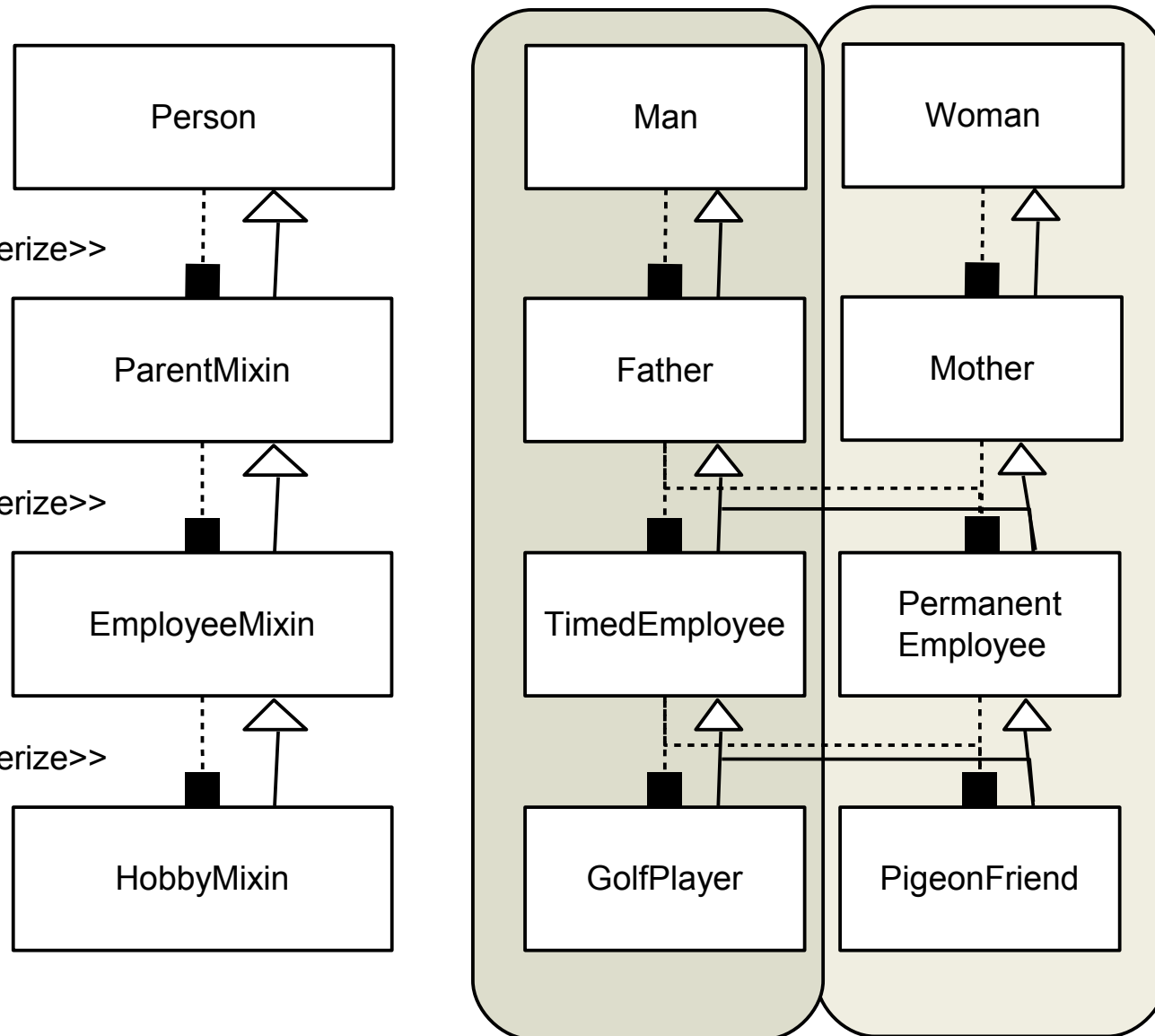
```
// Variants
Person: Man, Woman
ParentMixin: FatherMixin, MotherMixin
EmployeeMixin: TimedEmployee, PermanentEmployee
HobbyMixin: Gamer, Sportsman, GolfPlayer

// Compositions
GolfPlayer<PermanentEmployee<Father<Man>>>> assmann;
Gamer<TimedEmployee<Father<Man>>>> miller;
GolfPlayer<PermanentEmployee<Mother<Woman>>>> brown;
```

# Variations on Different Abstraction Layers form Product Variants

46

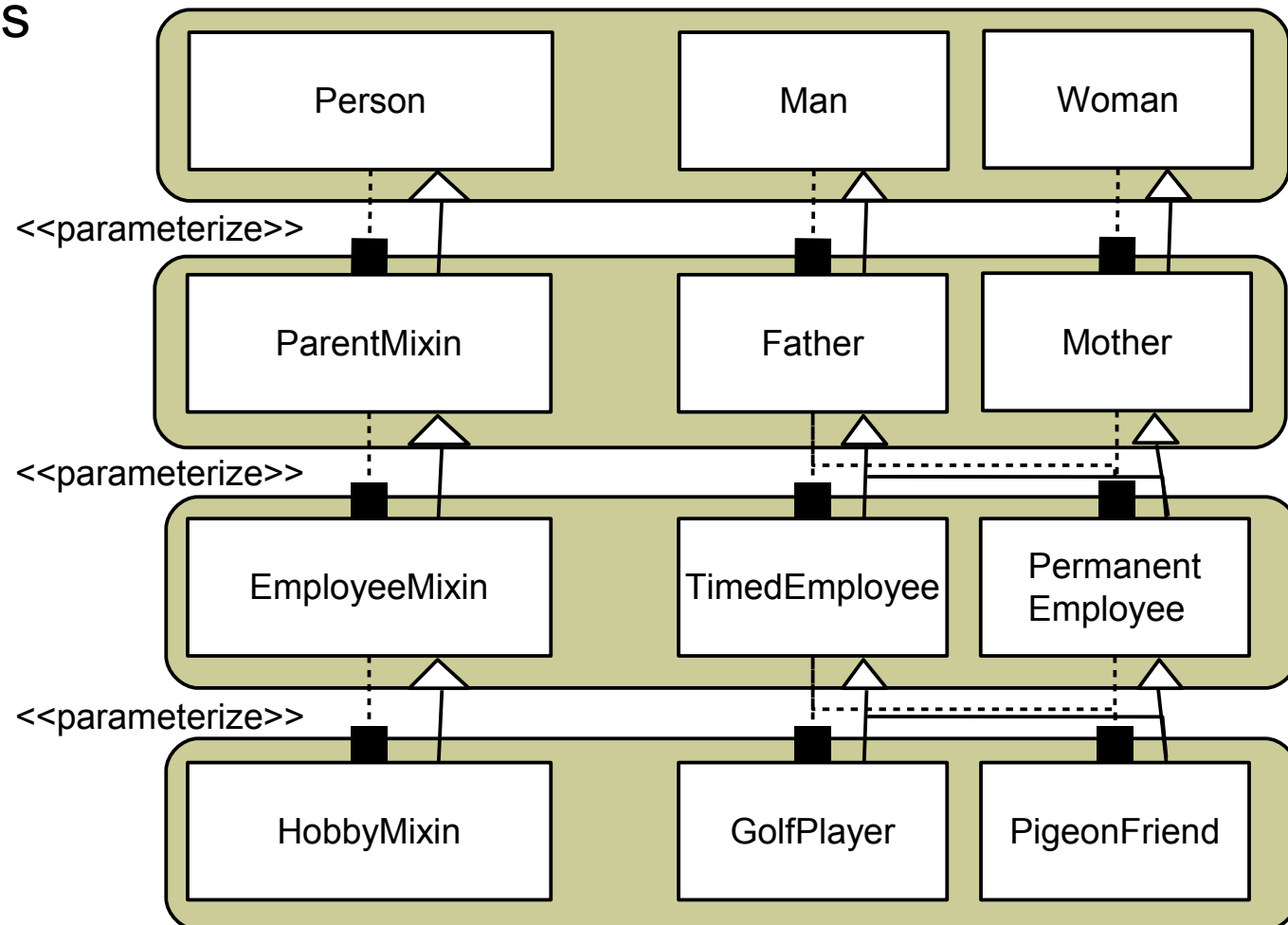
- ▶ Variants can be formed on every layer



# Variations on Different Role Layers

47

- ▶ Abstraction layers correspond to *role layers* of complex objects
- ▶ Roles *collaborate*, but are not implemented by role objects, but by mixins



# Discussion

48

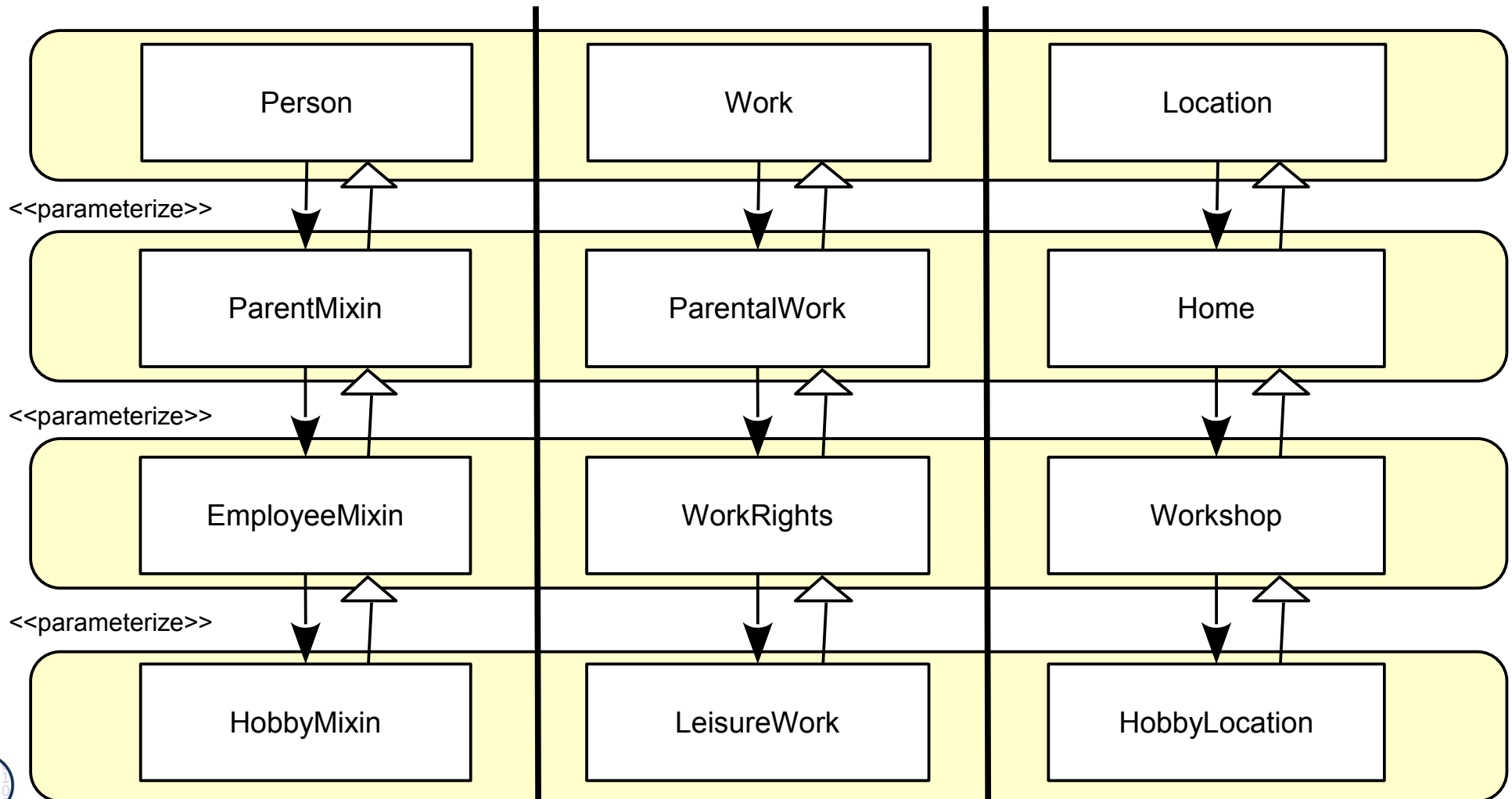
- ▶ A *mixin layer* groups all mixins of a role abstraction layer
- ▶ Mixins play in the GenVoca pattern the same role as role objects in the role object pattern and layered role frameworks
  - However, all role objects are *embedded* into one physical object
  - There is a physical identity for the entire logical object
  - No object schizophrenia to be avoided
  - GenVoca applications are more efficient, since they merge all roles together into one physical object
  - **But:** only static extensibility!
- ▶ Similarly to layered role object frameworks, layered GenVoca frameworks can model big product lines
  - Every abstraction layer (mixin layer) expresses variability
  - New mixin layers model extensibility



# 13.5 The Mixin Layer Pattern

49

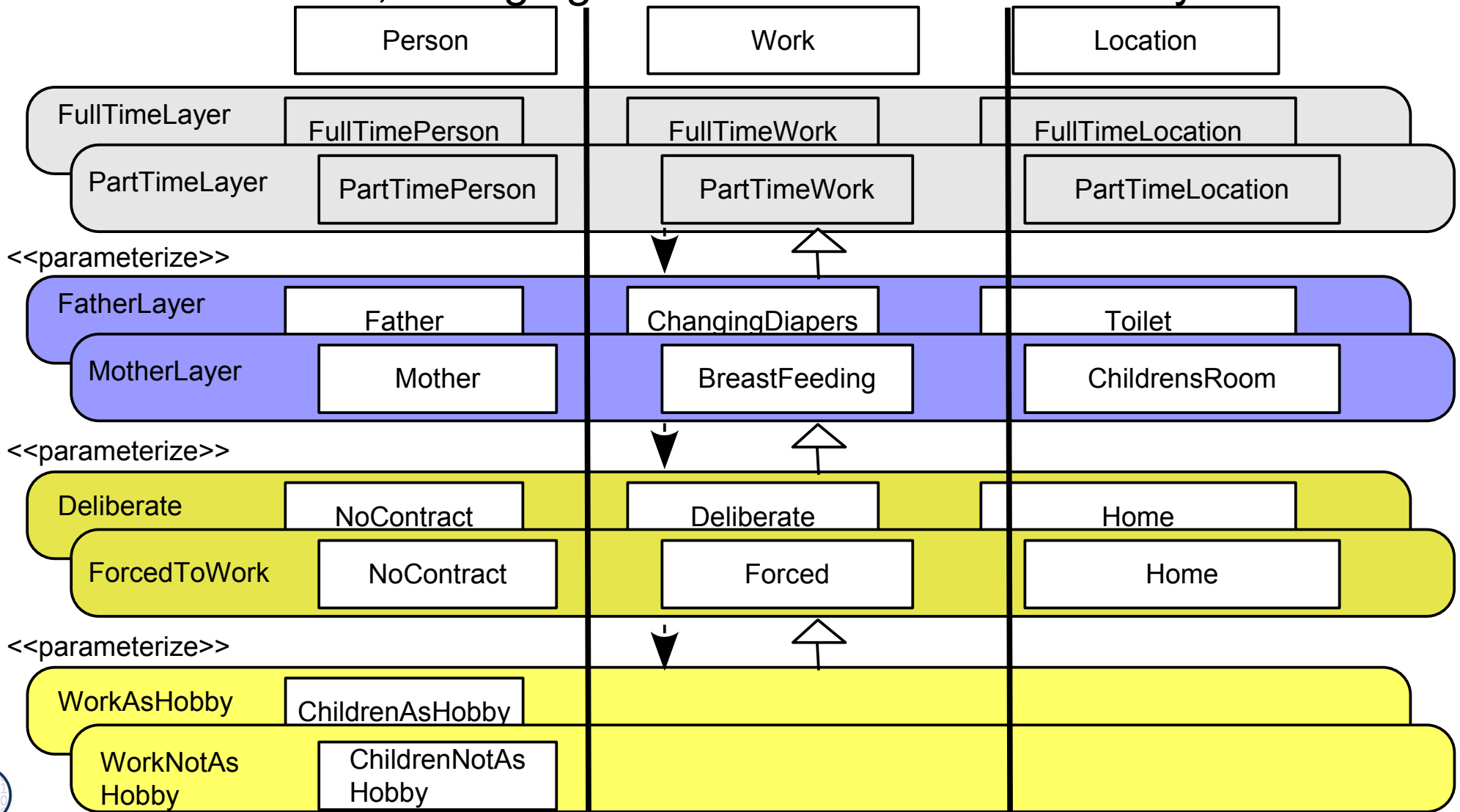
- ▶ While the GenVoca pattern deals with single stacking of parameterizations, the MixinLayer pattern groups all roles of an abstraction layer together and composes entire layers
- ▶ MixinLayer treats all logical objects of an application



# Mixin Layers as Compositional Unit

50

- ▶ A mixin layer gets a name and can be exchanged consistently for a variant, changing the behavior of the entire layer



# Composition of Mixin Layers

51

- ▶ Mixin layers are composed similarly to single GenVoca mixins
  - Meaning: All role classes are consistently exchanged with their layer

```
CoreLayer: FullTime, PartTime
ParentLayer: FatherLayer, MotherLayer
EmployeeLayer: Deliberate, ...
HobbyLayer: WorkAsHobby, Slave....

// This is now mixin layer composition!
WorkAsHobby<Deliberate<FatherLayer<FullTime>>>> assmann;
```

# Implementation of Mixin Layers with GenVoca Pattern and Inner Classes

52

- ▶ The role classes of upper layers form super classes of the layer class
- ▶ The following pattern allows for separate parameterization of all role mixins, *not* the layer as a whole

```
class Layer <class Super, class RoleSuper1, .., class RoleSupern>
  extends Super {
    class Role1 extends RoleSuper1 { .. }
    ..
    class Rolen extends RoleSupern { .. }

    .. additional classes..
  }
```

# Implementation of Mixin Layers with Designated Inner Classes

53

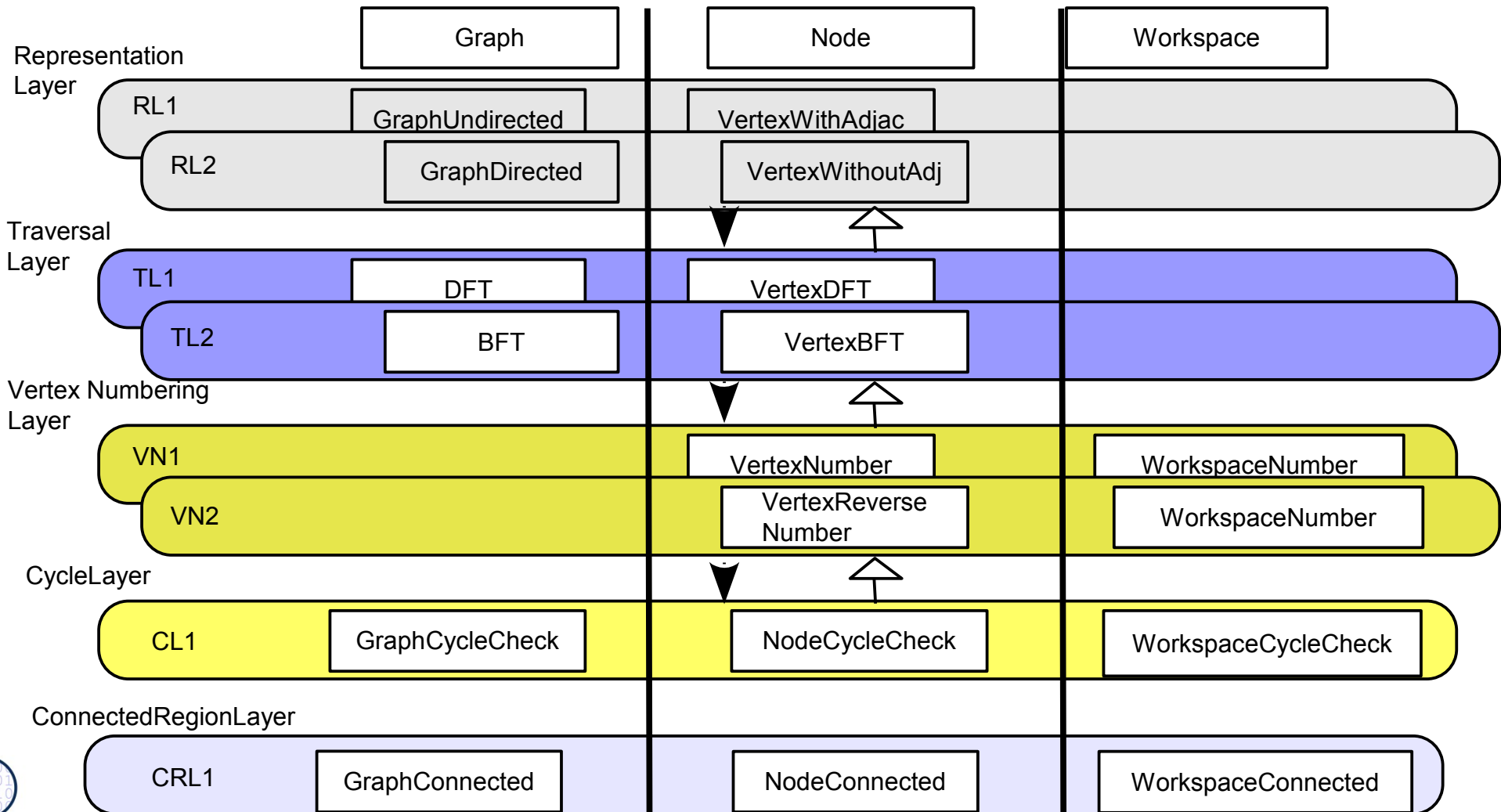
- ▶ If the target language permits to have inner classes that can be designated by an expression, mixin layers can be inherited as a whole
- ▶ The super mixin layer can be selected by one single expression  $L \langle L1 \rangle$

```
class Layer <class Super>
// The class Super has n inner role classes RoleSuper1, ...,
// RoleSupern
extends Super {
  class Role1 extends Super.RoleSuper1 { .. }
  ..
  class Rolen extends Super.RoleSupern { .. }
  .. additional classes..
}
```

# Example: A Graph Framework [Herrejon Batory]

54

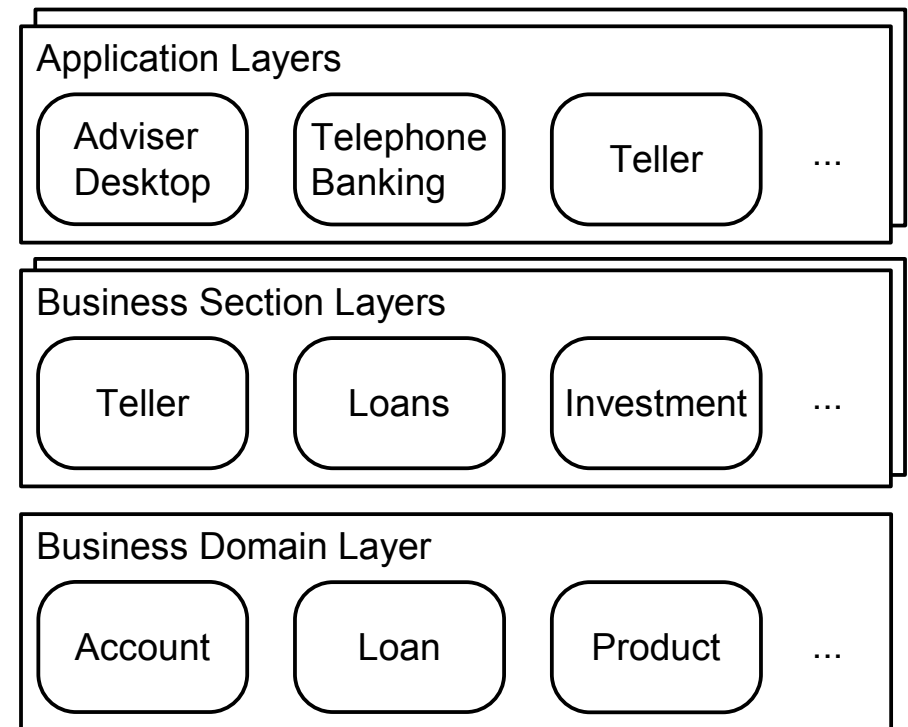
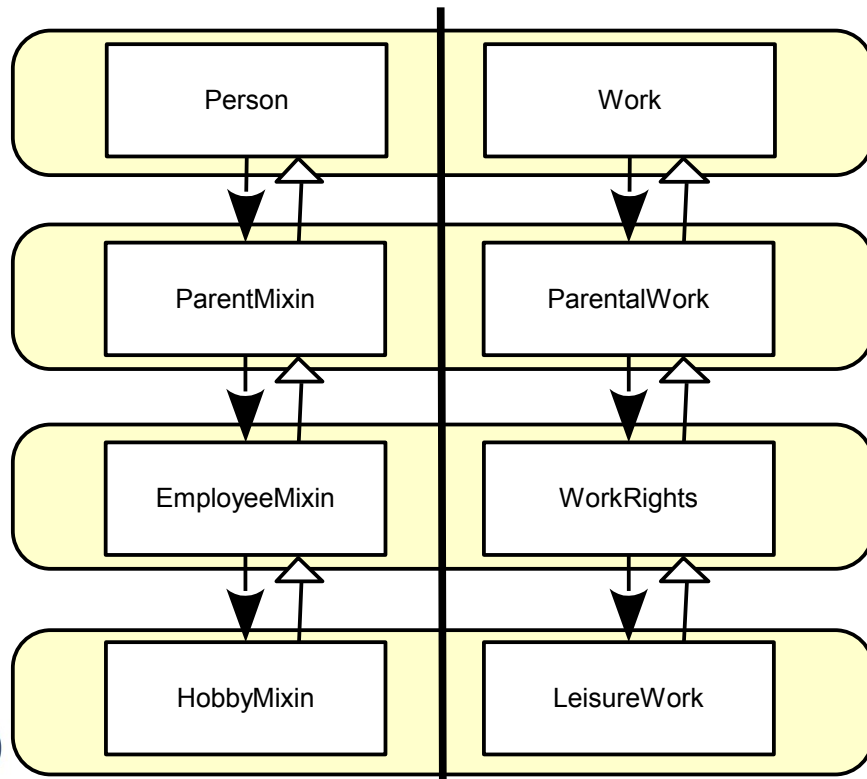
- ▶ Graph applications can be structured into mixin layers
- ▶ `ConnectedOnDFTUndirected = CRL1<CL1<VN1<TL1<RL1>>>>>`
- ▶ `ConnectedOnBFTRevDirected = CRL1<CL1<VN2<TL2<RL2>>>>>`



# Layered Mixin Frameworks vs Layered Role Object Frameworks

55

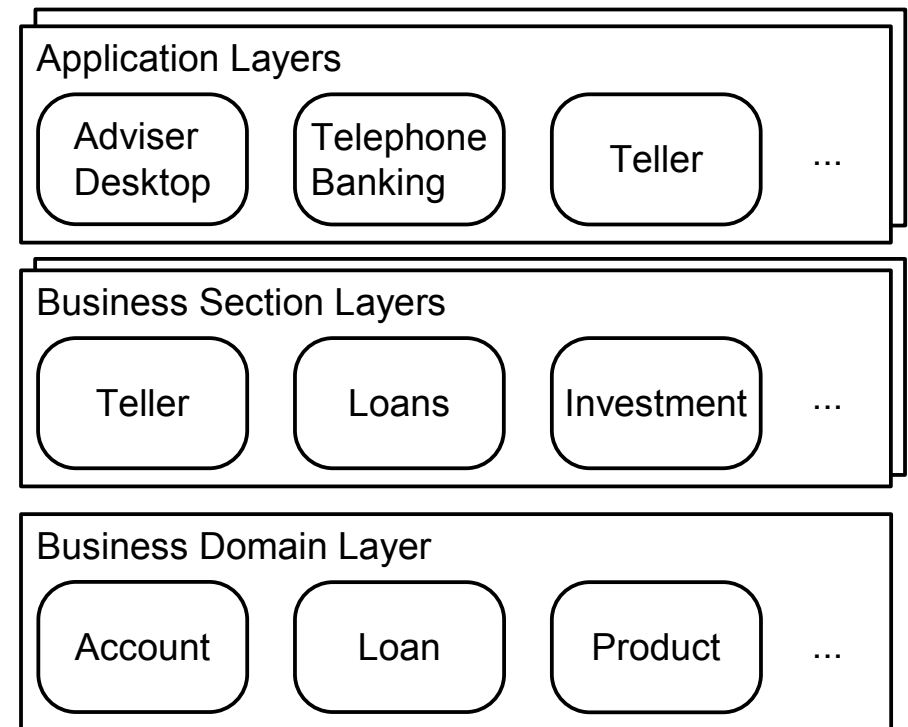
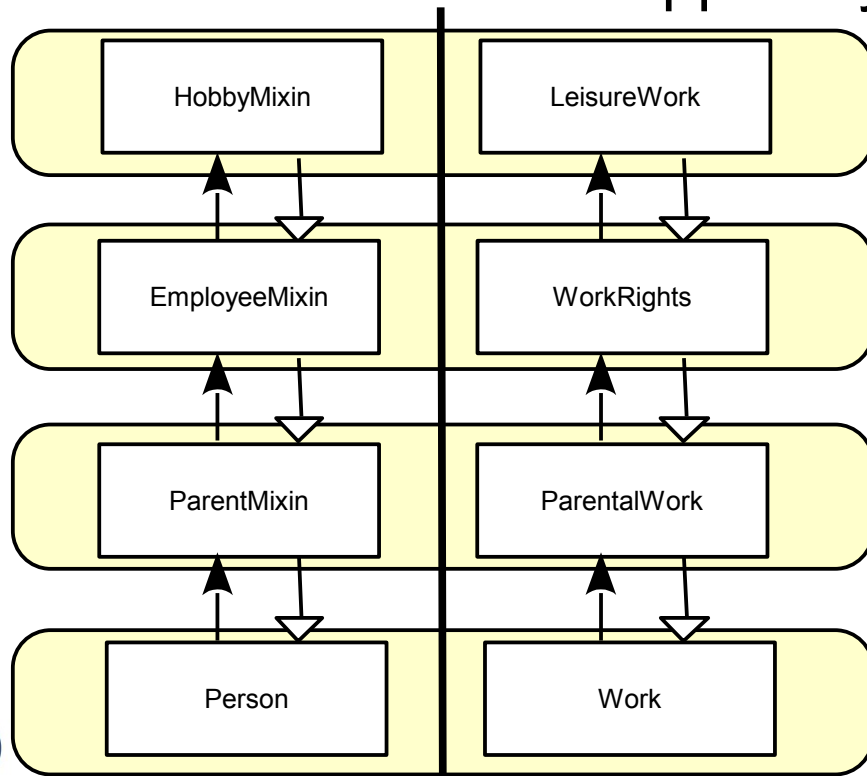
- ▶ Every mixin layer corresponds to a role layer
- ▶ Mixin layers form *frameworks* that can be extended by mixin layer composition towards applications
- ▶ Same variability effects for big product lines



# Layered Mixin Frameworks vs Layered Role Object Frameworks

56

- ▶ Unfortunately, the direction of generality is usually drawn in the opposite way in mixin layer frameworks and role object frameworks
- ▶ If we agree to put the “most general abstraction layer” downmost, the dependencies go into the same direction
- ▶ Features on the upper layers *depend* on the lower layers





# Layered Mixin Frameworks vs Layered Role Object Frameworks

57

- ▶ Essentially, layered role object frameworks and layered mixin frameworks provide the same concept for variability and extensibility
- ▶ Difference: dynamic extensibility
  - Layered role object frameworks allow for it
  - Layered mixin frameworks don't

# What Have We Learned?

58

- ▶ Extension Objects Pattern
- ▶ Extensible Framework Hook Patterns
  - Using Role Object Pattern
  - Using Genvoca (MixinLayers)
- ▶ Role Object Pattern for dynamic extensibility
- ▶ Genvoca for static extensibility

# The End

59