# 12. Frameworks and Patterns - Framework Variation Patterns



Prof. Dr. U. Aßmann

Software Engineering

Faculty of Informatics

Dresden University of Technology

WS 16/17 - Dec 7, 2016

**Lecturer**: Dr. Sebastian Götz

1. Open Role Framework Hooks
2. Framework Hook Patterns
3. Delegation-Based Framework Hook Patterns
4. Recursion-Based Framework Hook Patterns
5. Unification-Based
6. Inheritance-Based

# Literature (To Be Read)



- W. Pree. **Framework Development and Reuse Support**. In Visual Object-Oriented Programming, Manning Publishing Co., editors M. M. Burnett and A. Goldberg and T. G. Lewis, Pp, 253-268, 1995.
  www.softwareresearch.net/publications/J003.pdf
  - Or: D. Karlsson. Metapatterns. Paper in Design Pattern seminar, IDA, 2001. Available at home page.
- 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

# Secondary Literature



- W. Pree. **Design Patterns for Object-oriented Software Development.** Addison-Wesley 1995.
- M. Fontoura, W. Pree, B. Rumpe. **The UML Profile for Framework Architectures.** Addison-Wesley, Object Technology Series. 2002.

# Goal

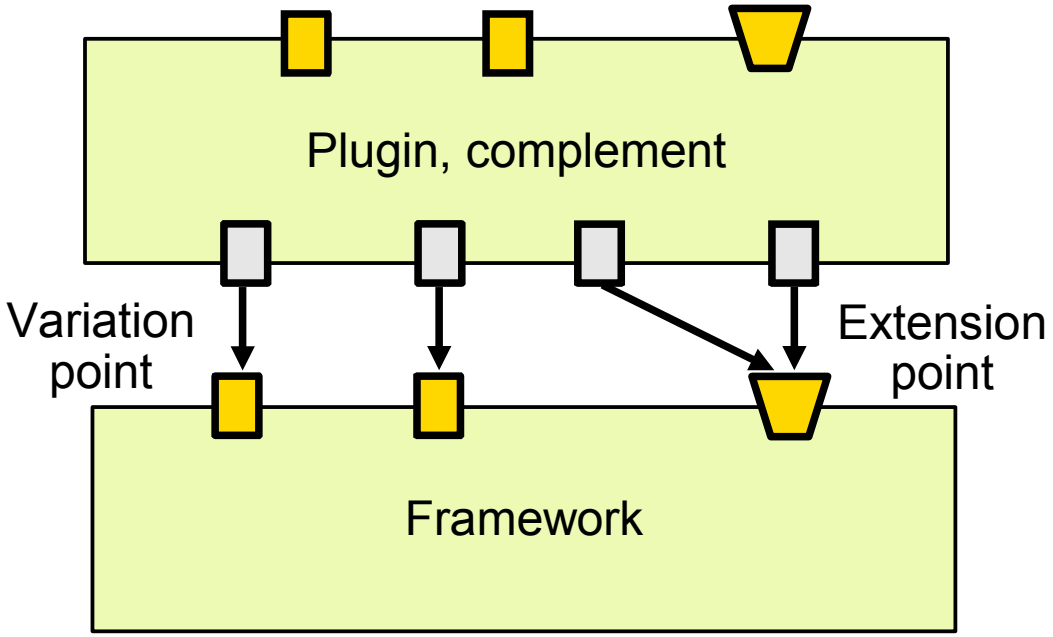


- What's a framework?
- Studying variabilities of frameworks with the T&H concept
- Introducing different types of hooks for frameworks and components (TH patterns)
- Understand framework hook patterns
  - The box-like notation for frameworks and framework hook patterns
- More types of dimensional frameworks

# Plugins and Extensions Points



- Frameworks are completed to products with **plugins (complements)**. Frameworks carry
  - framework extension hooks, **extension points**, which can be extended (bound) many times
  - framework variation hooks, **variation points,** which can be bound only once
- Plugins can be frameworks themselves (layered frameworks)



# Design Patterns and Frameworks



- Historically, design patterns were discovered during framework development
  - Smalltalk MVC [Goldberg, Reenskaug]
  - ET++ [Gamma]
  - Interviews [Vlissides]
- Design patterns are *building blocks of frameworks*
  - Framework developers vary and extend classes of the framework
- Design patterns create the products of a product line
  - Application developers vary and extend classes of the framework
  - Variability design patterns can be used as *framework variation points (framework variation hooks)*
  - Extensibility design patterns can be used as *framework extension points (framework extension hooks)*

# 12.1 Framework Instantiation and Merging With Open Roles



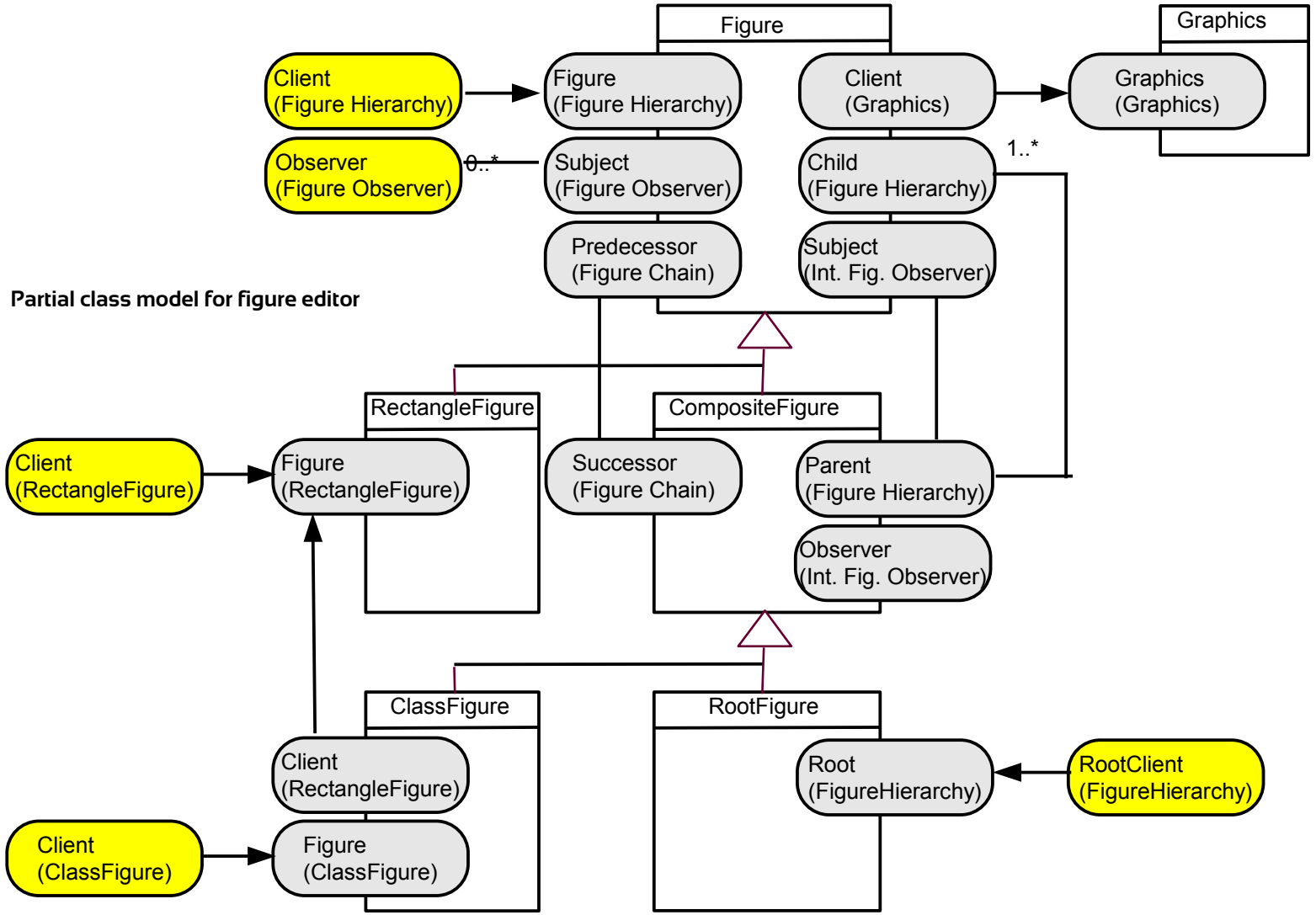
# Framework Instantiation with Open Roles



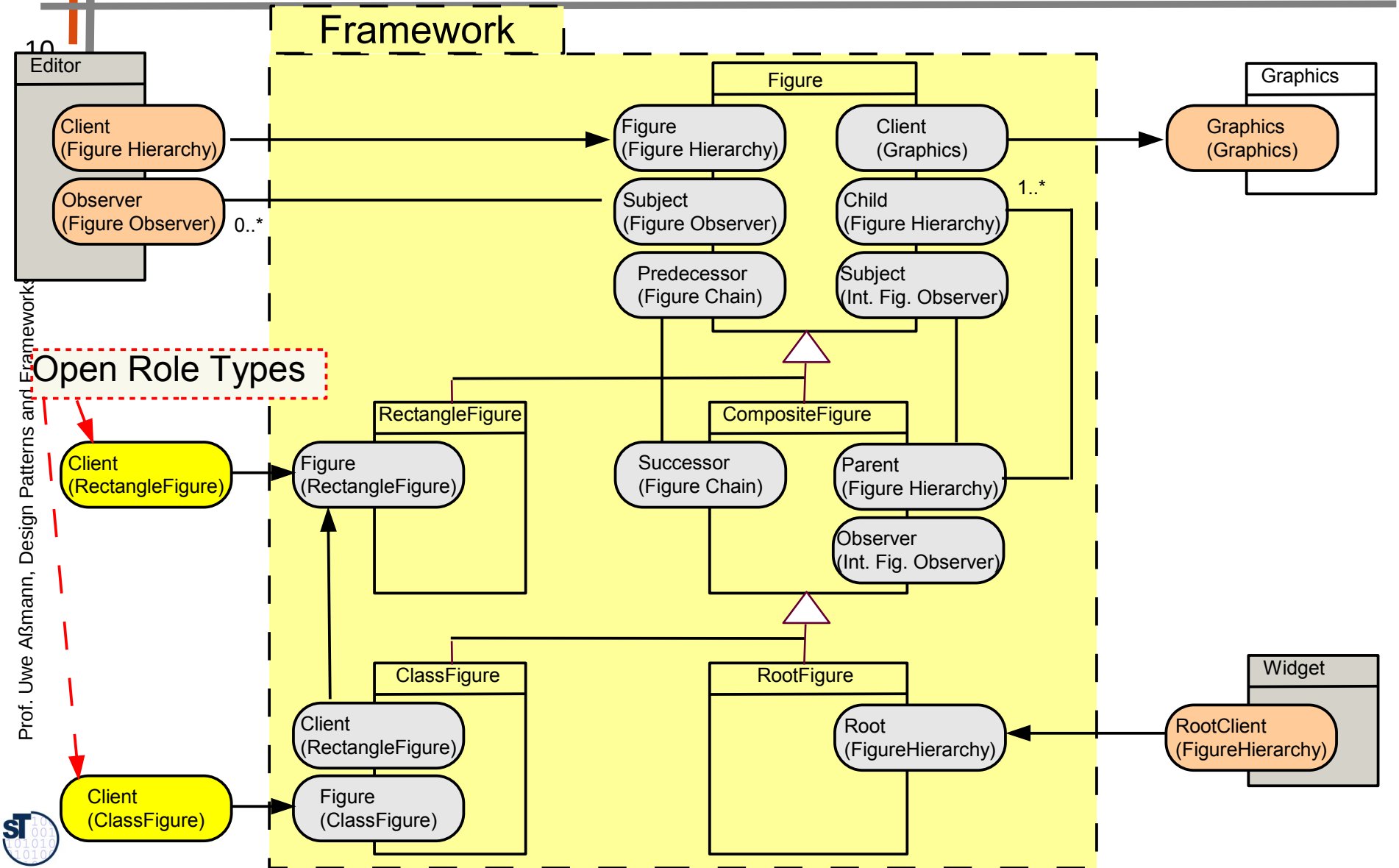
- The most simple form of framework instantiation is Riehle/Gross' *open role instantiation* based on association
  - Here, frameworks are class models with *open role hooks (free, unbound roles),* role types that have not yet been assigned to classes
- The open roles form an *integration repertoire*
  - the set of role types, by which the framework can be integrated into an application (*framework hooks, framework variation points)*
- A framework is *instantiated* by binding its open roles to classes
  - Role constraints have to be respected
- Hence, role models play the bridge between a framework and its clients

# Remember: The Partial Figure Model, a Standard Role-Class Model



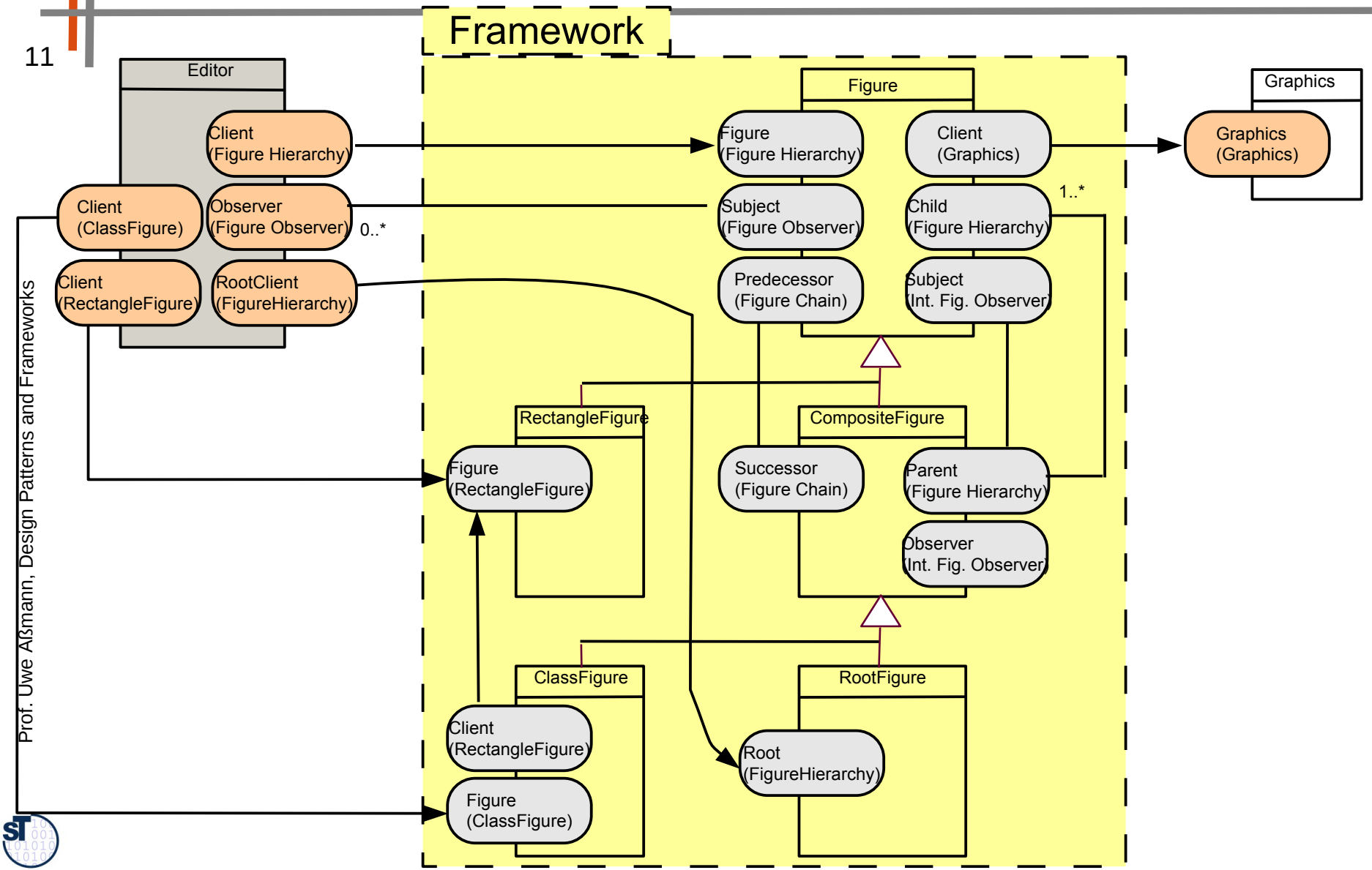


# The Figure Framework, Partially Instantiated

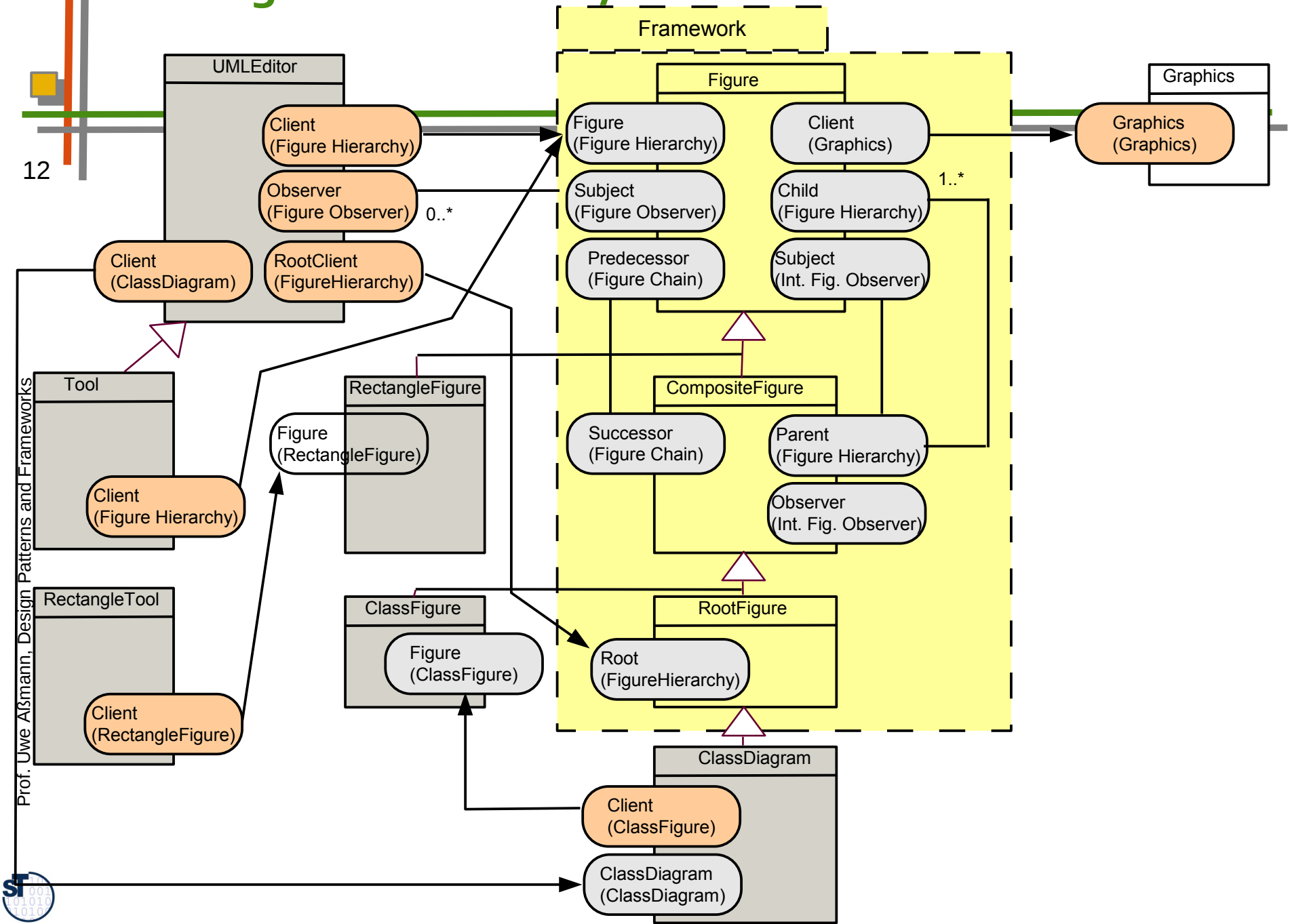


# The Figure Framework, Fully Instantiated to an Editor





# The Figure Framework, Instantiated to an UML Editor



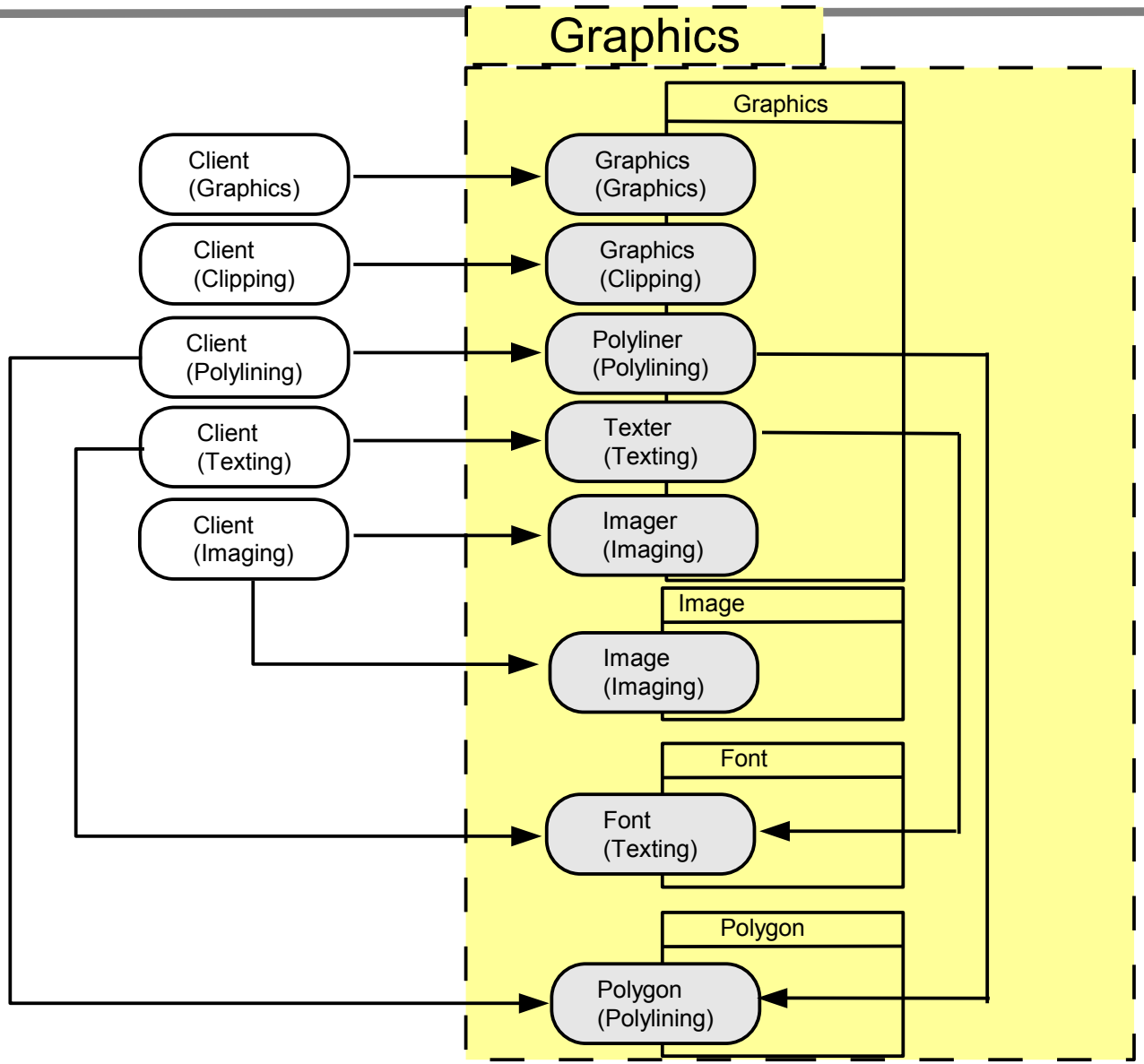
# Merging of Frameworks



- Two frameworks are *merged* by binding the open roles of framework A to classes of framework B
  - Role constraints have to be respected
- Hence, role models play the bridge between different frameworks, too.
  - Leads to layers of frameworks

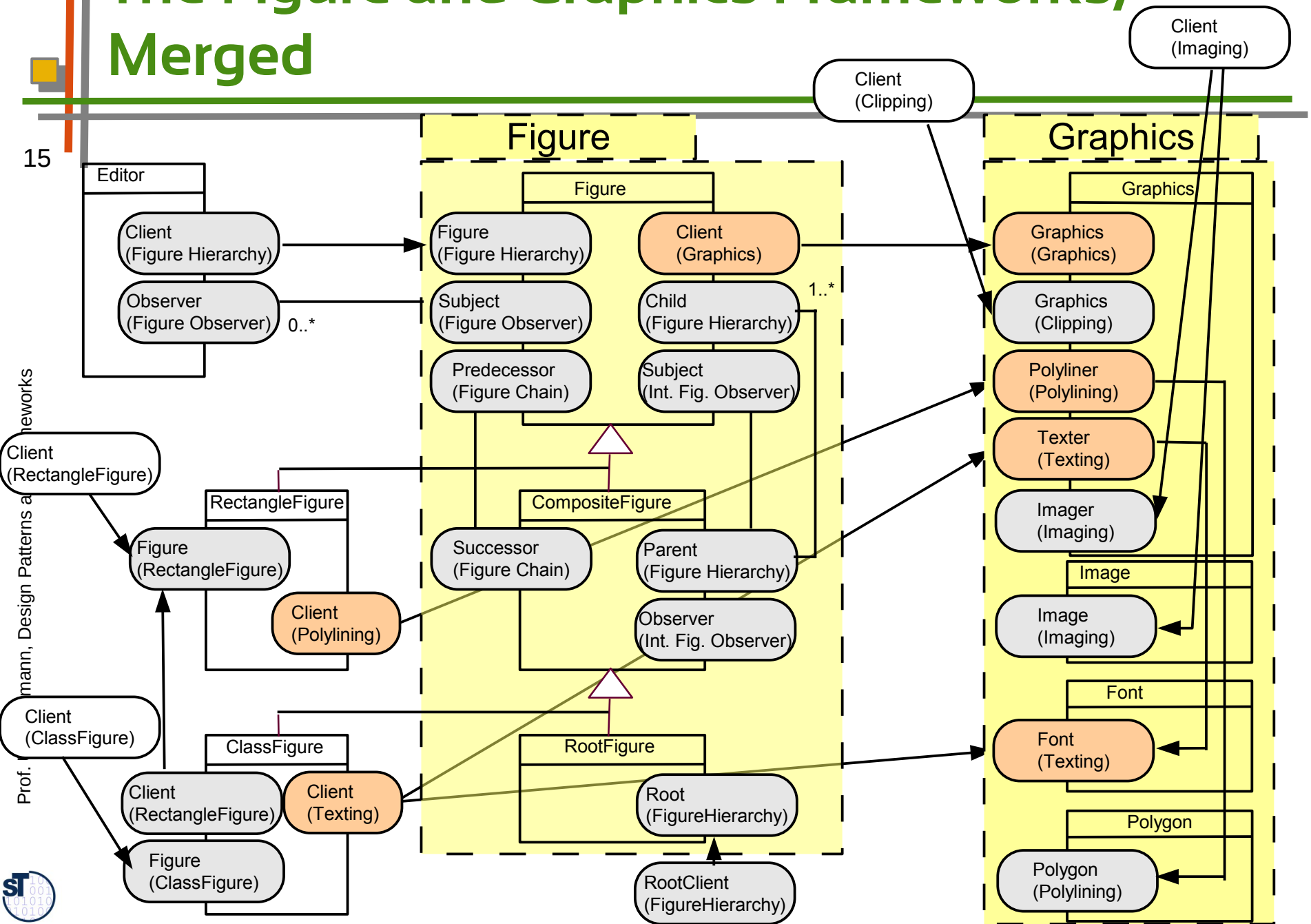
# A Graphics Framework





# The Figure and Graphics Frameworks, Merged





# Limitations of Open Role Instantiation



- [Riehle/Gross] role-based framework instantiation relies on simple role binding, with role constraints
- Role binding for framework instantiation and merging can be even more elaborated

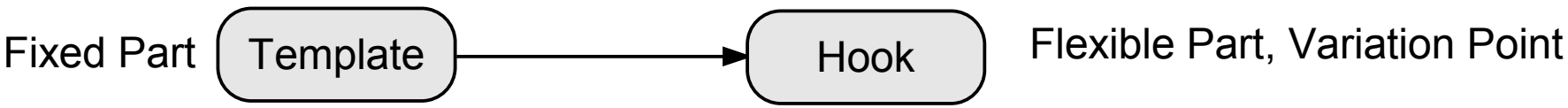
# 12.2 Framework Hook Patterns



# Pree's Framework Hook Patterns (Template&Hook Role Models)



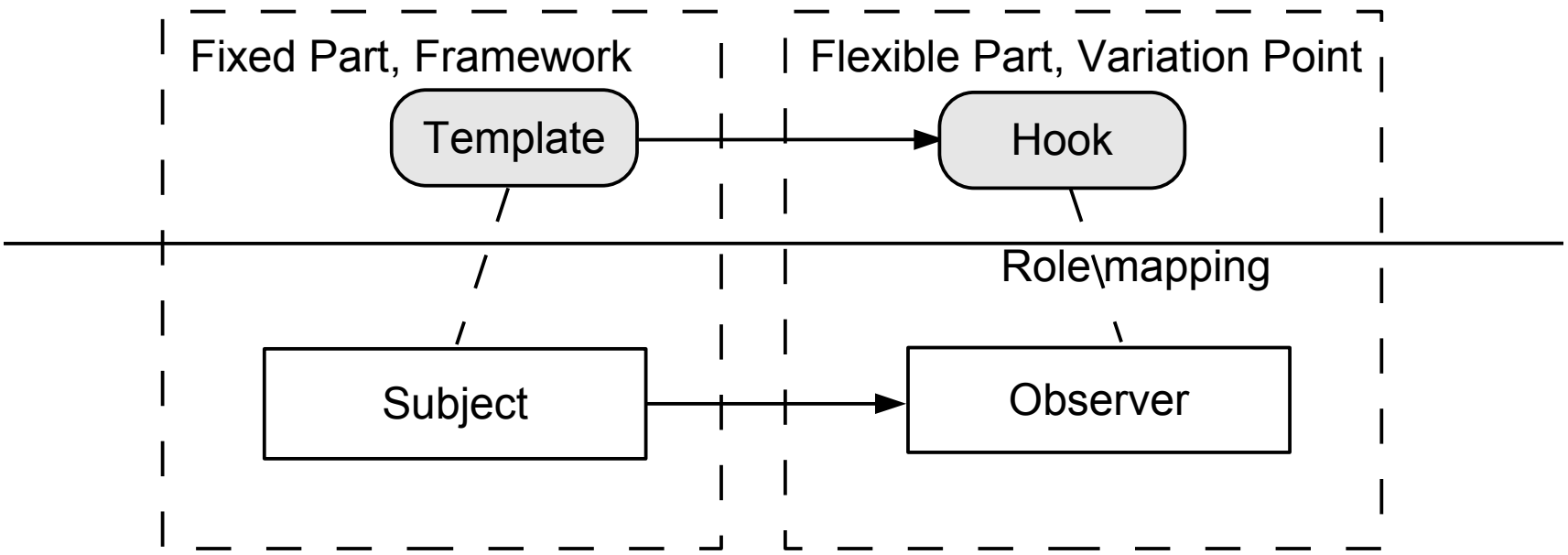
- In Pree's work, *framework hooks* are characterized by design patterns *(framework hook patterns)*
  - They describe the roles of classes on the *border* of the framework
  - The framework hook pattern determines the way how the classes interact with each other at the border of the framework
- A framework variation point is characterized with a *Template&Hook conceptual pattern*
  - Pree called this a *T&H metapattern*, we call this a *T&H role model*
- A T&H role model has 2 parts:
  - A template class (or *template role type)*, which gives the skeleton algorithm of the framework: Fix, grasps commonalities
  - A hook class, which can be exchanged (or: a *hook role type* which can be bound to a client class): Variable, even extensible, grasps variability and extension



# T&H Patterns and Standard Patterns



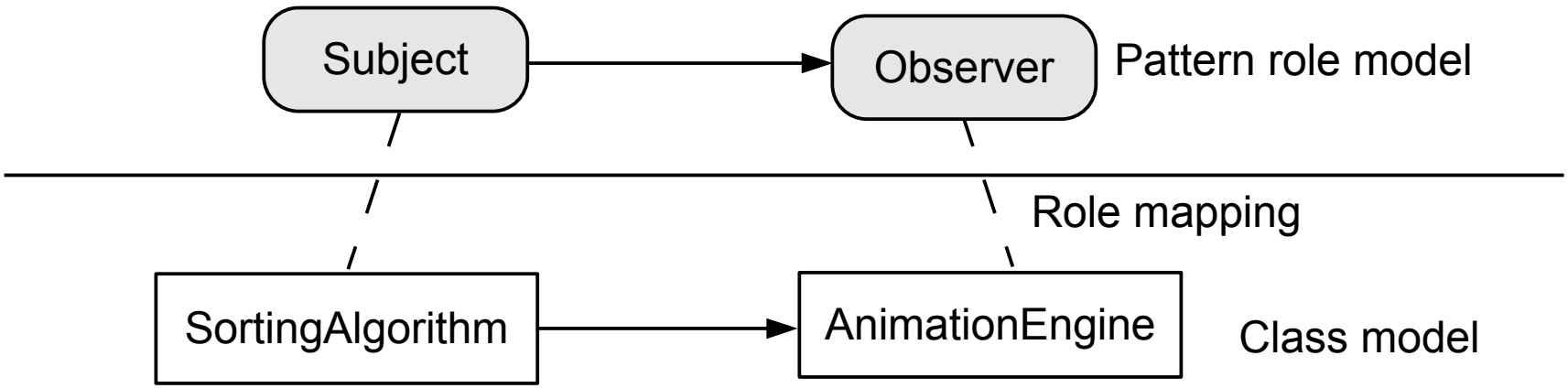
- A TH-role model *overlays* another pattern (hence Pree called it a *metapattern*)
  - The template part fixes parts of the pattern
  - The hook part keeps parts of the pattern variable, i.e., open for binding.



# T&H in Standard Design Patterns



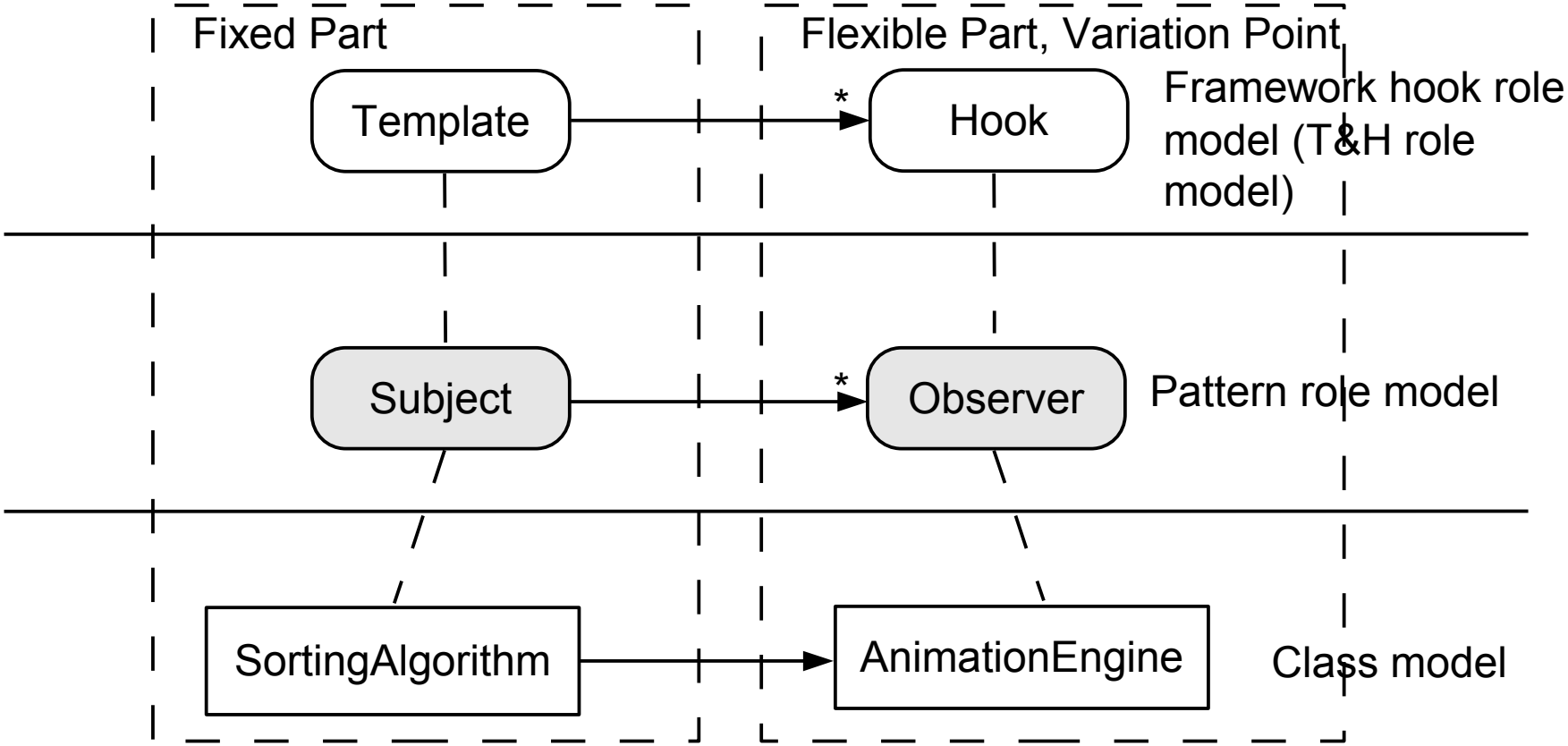
- Subject and Observer can vary; nothing is fixed
  - SortingAlgorithm and AnimationEngine can be exchanged



# T&H in Framework Hook Patterns



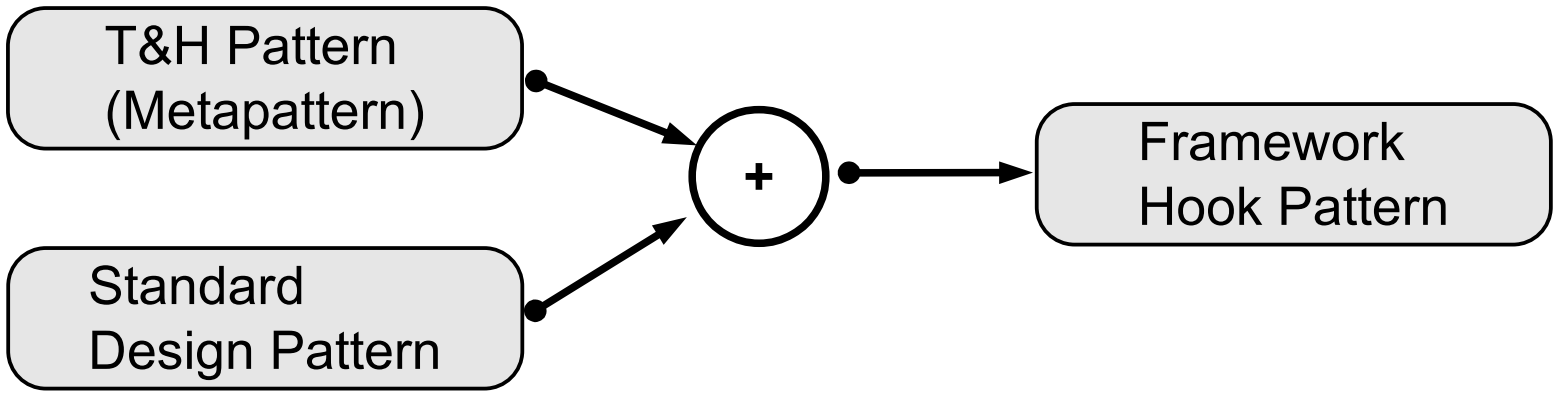
- Subject can no longer vary; it is fixed
  - SortingAlgorithm cannot be exchanged (exception: DimensionalClassHierarchies)



# Why T&H Patterns add More to Standard Patterns



- Due to Riehle and Gross, we know that metapatterns are role models that overlay the role models of design patterns
  - Metapatterns are very general role models that can be mixed into every design pattern
  - As design patterns describe application models, metapatterns describe design patterns
- In [Pree], roles are not considered. Pree has only hook classes and hook methods. Here, we combine [Pree] and [Riehle/Gross]
- If a metapattern is overlayed to a role model of a design pattern, it adds commonality/variability knowledge, describing a *framework variation point*
  - The template part characterizes the framework's fixed parts
  - The hook part characterizes the framework's variation point
- Hence we call a design pattern with metapattern information **framework hook pattern**



# Framework Hook Patterns

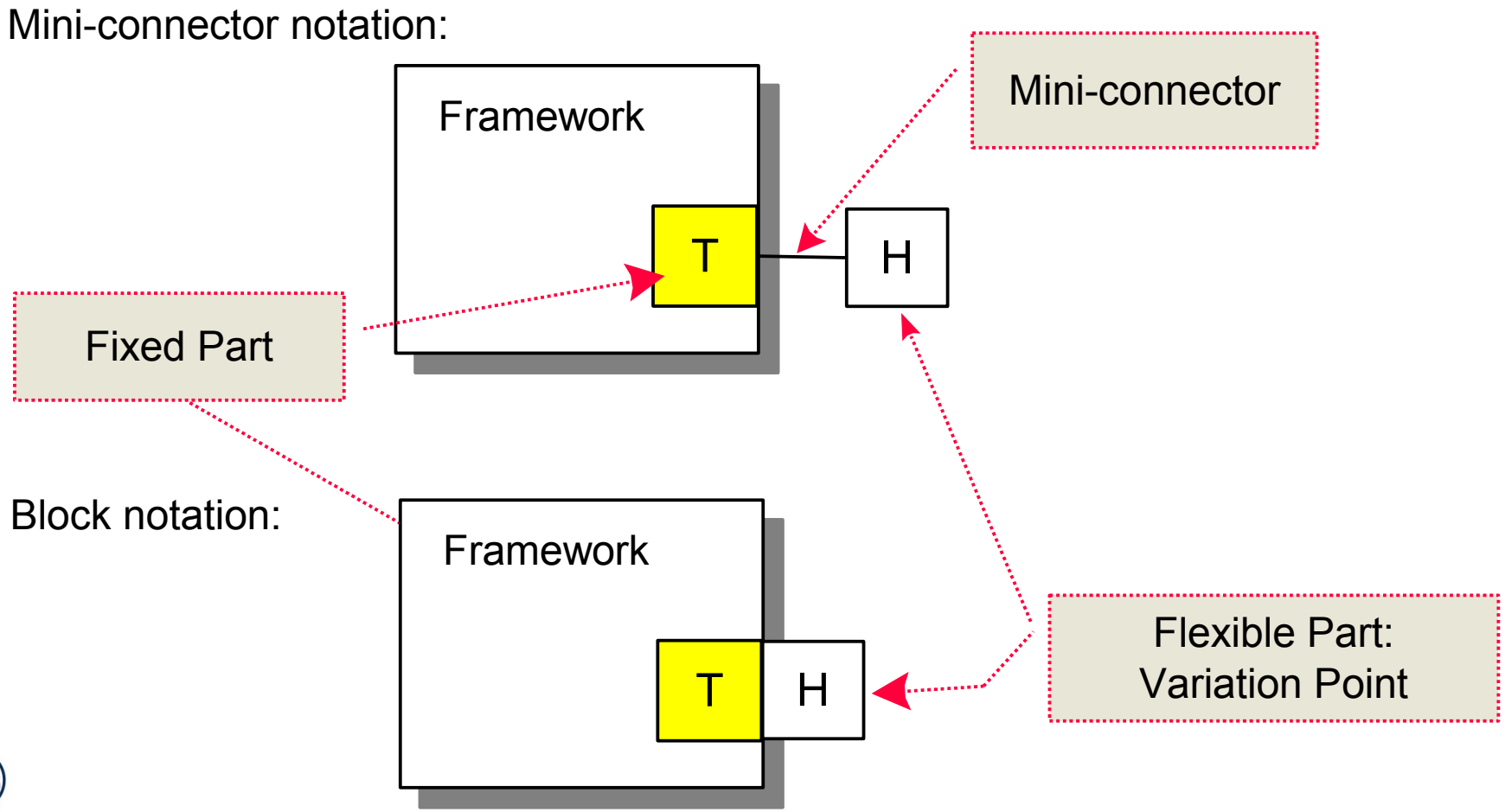


- The template-hook role model
  - adds more pragmatics to a standard design pattern: **information about commonality and variability**. Hence, framework variation points are described
  - The template-hook role model adds more *constraints* to a standard design pattern. Some things can no longer be exchanged
- Pree discovered 7 framework hook patterns, i.e., 7 template-hook role models for framework hooks
  - The template-hook role models describe the parameterization of the framework by *open roles*
  - They include Riehle's open roles, but add more variants
  - There are even other ones (see next chapter)
- Framework T&H patters are defined only **at the border** of frameworks

# Two Simple Notations for Framework Hook Patterns



- Mini-connector notation: shows T, H, mini-connector
- Block notation: Shows T, H



# 12.3 Delegation-Based Framework Hook Patterns

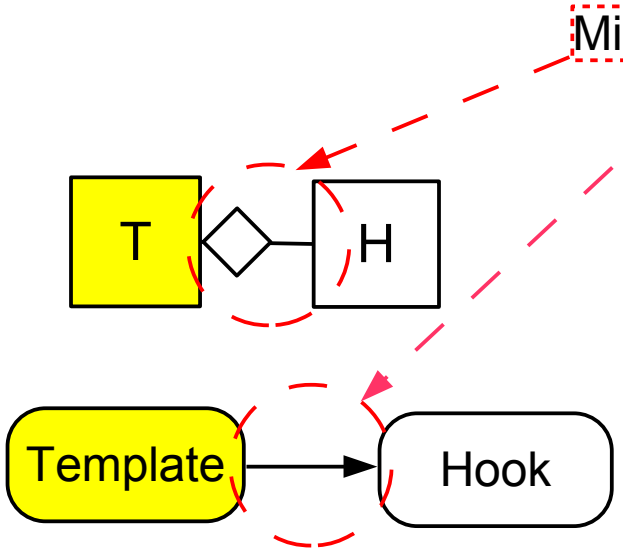


# T—H Connection Pattern

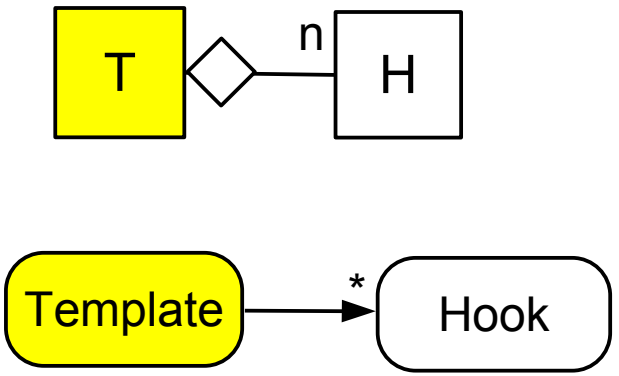


- T&H *connection pattern* (T--H framework hook)
  - Similar to Riehle/Gross open role type, but with aggregation instead of association
  - T and H classes are coupled by a template-hook role model, the hook is a delegatee (the relation is called a *mini-connector)*
  - “Whole” is in the framework, “Part” is in the plugin

**1-T—H (aggregated open role hook)**
H part of T



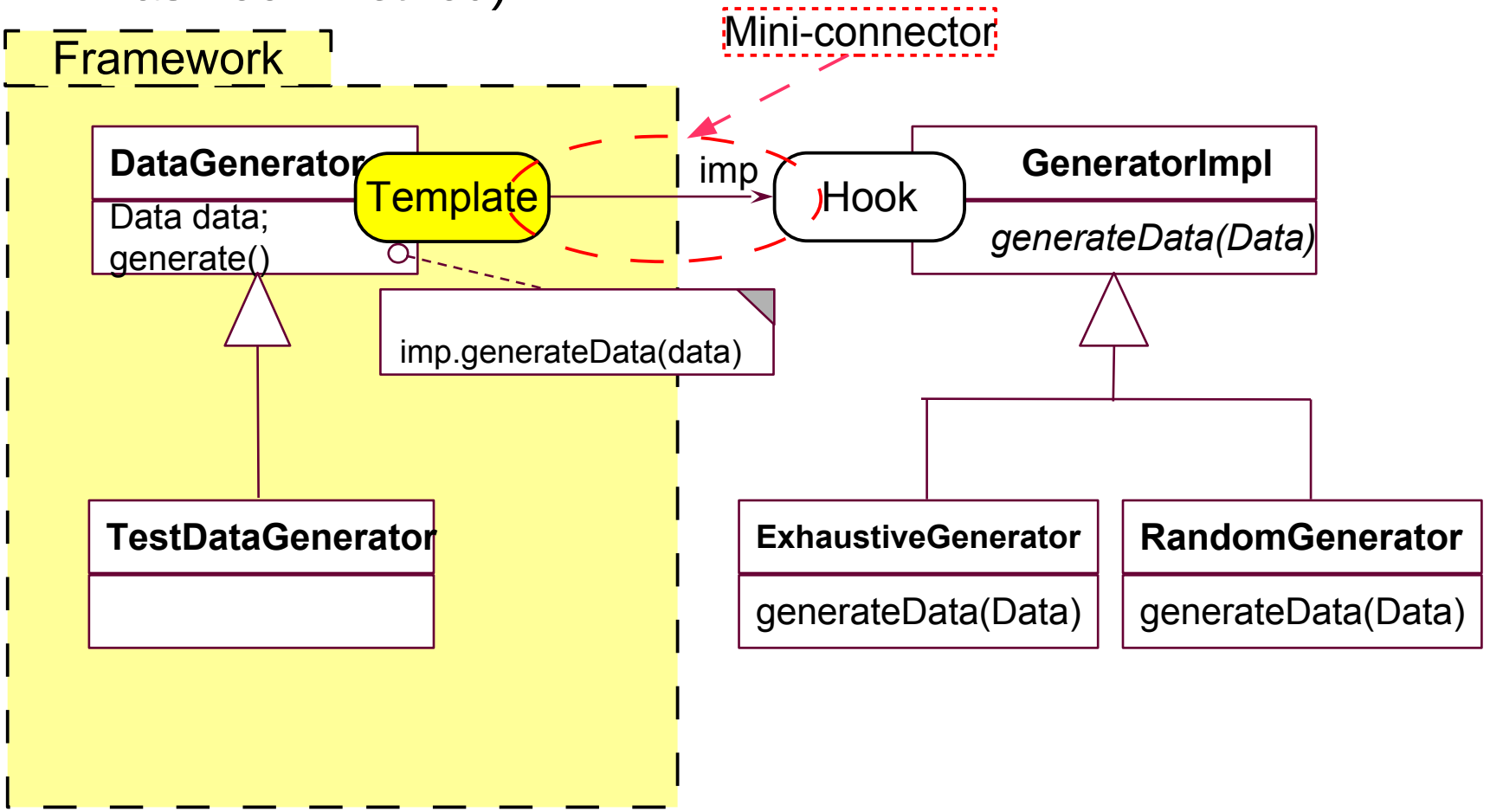

**n-T—H (flat extension)**
T has n H parts, n is dynamic



# TemplateClass with 1-T--H

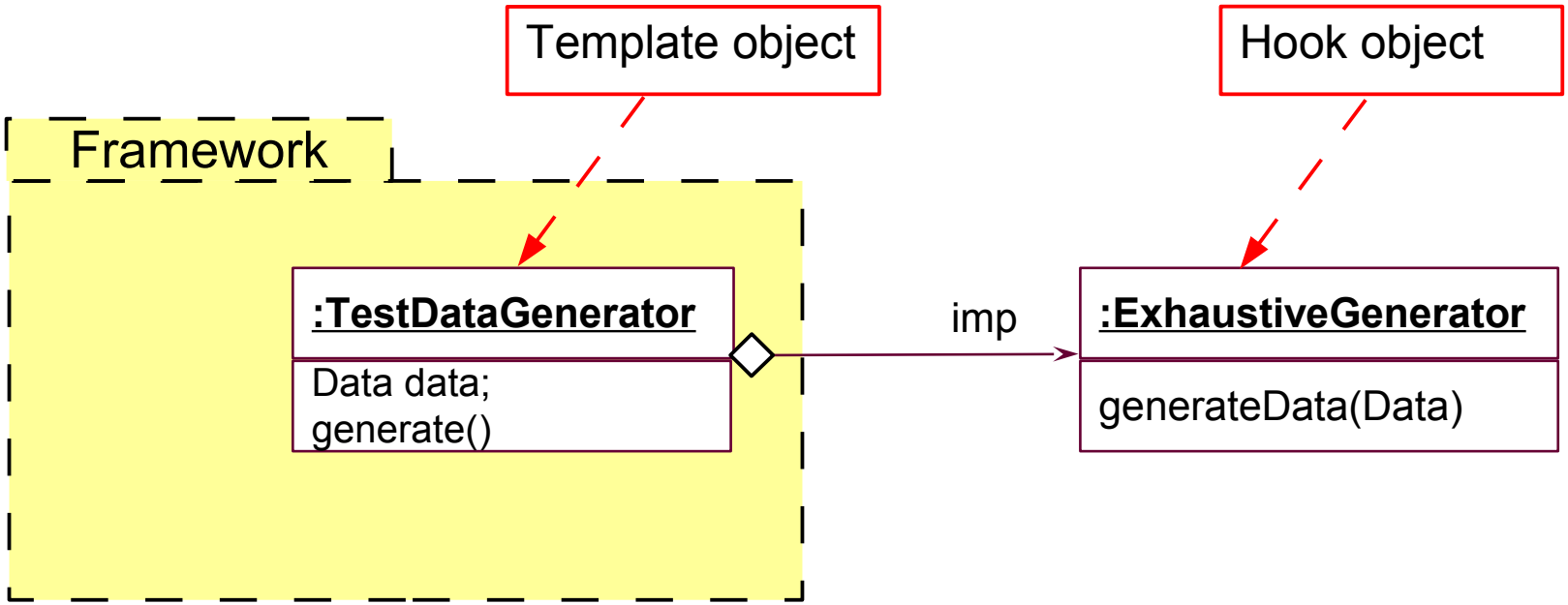


- Attention: in this case, the Template role also carries the TemplateM role (framework has template method, application has hook method)



# TemplateClass Runtime Scenario

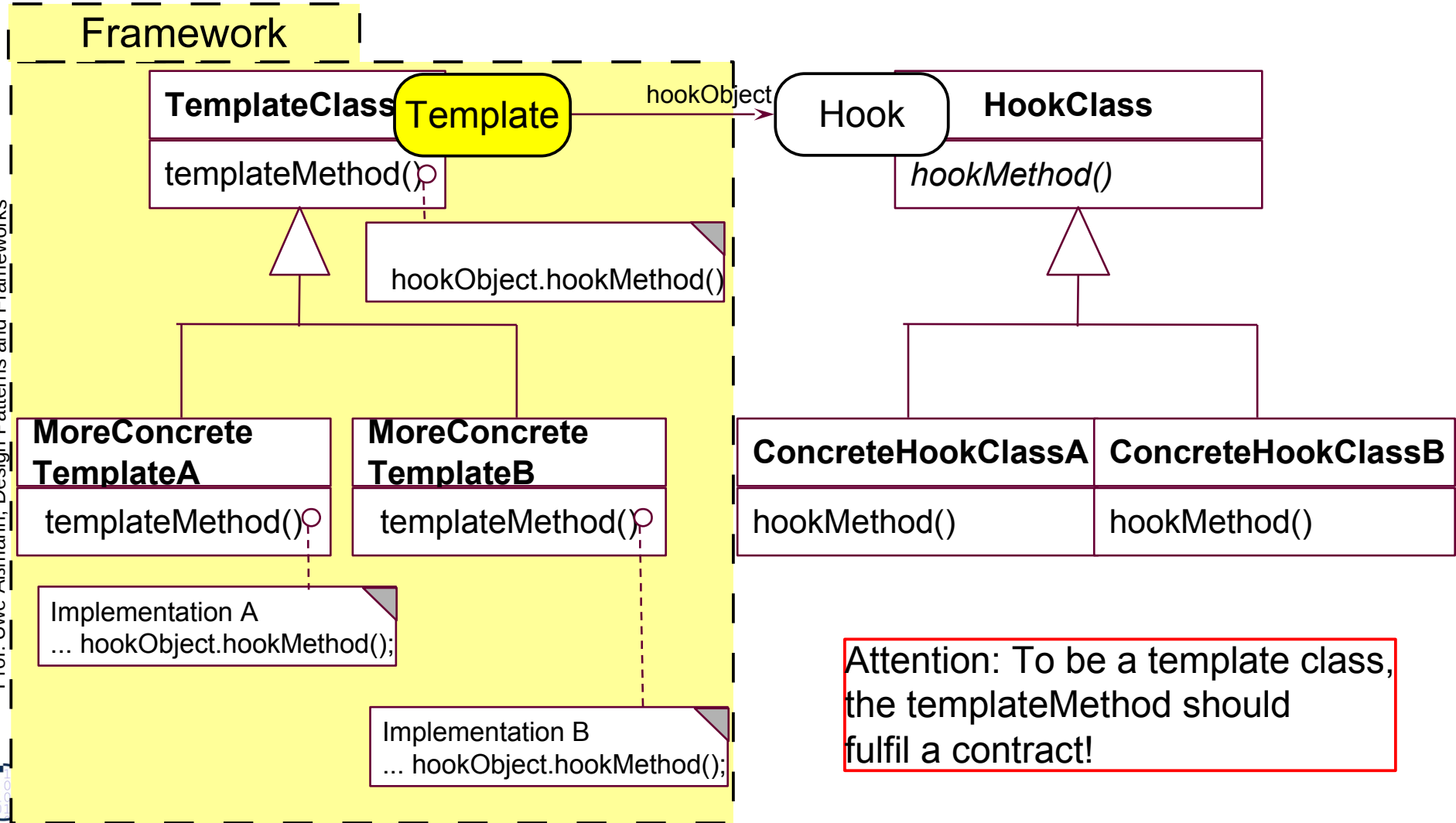




# Dimensional Hierarchies with 1-T--H (Bridge with Template/Hook Constraint)

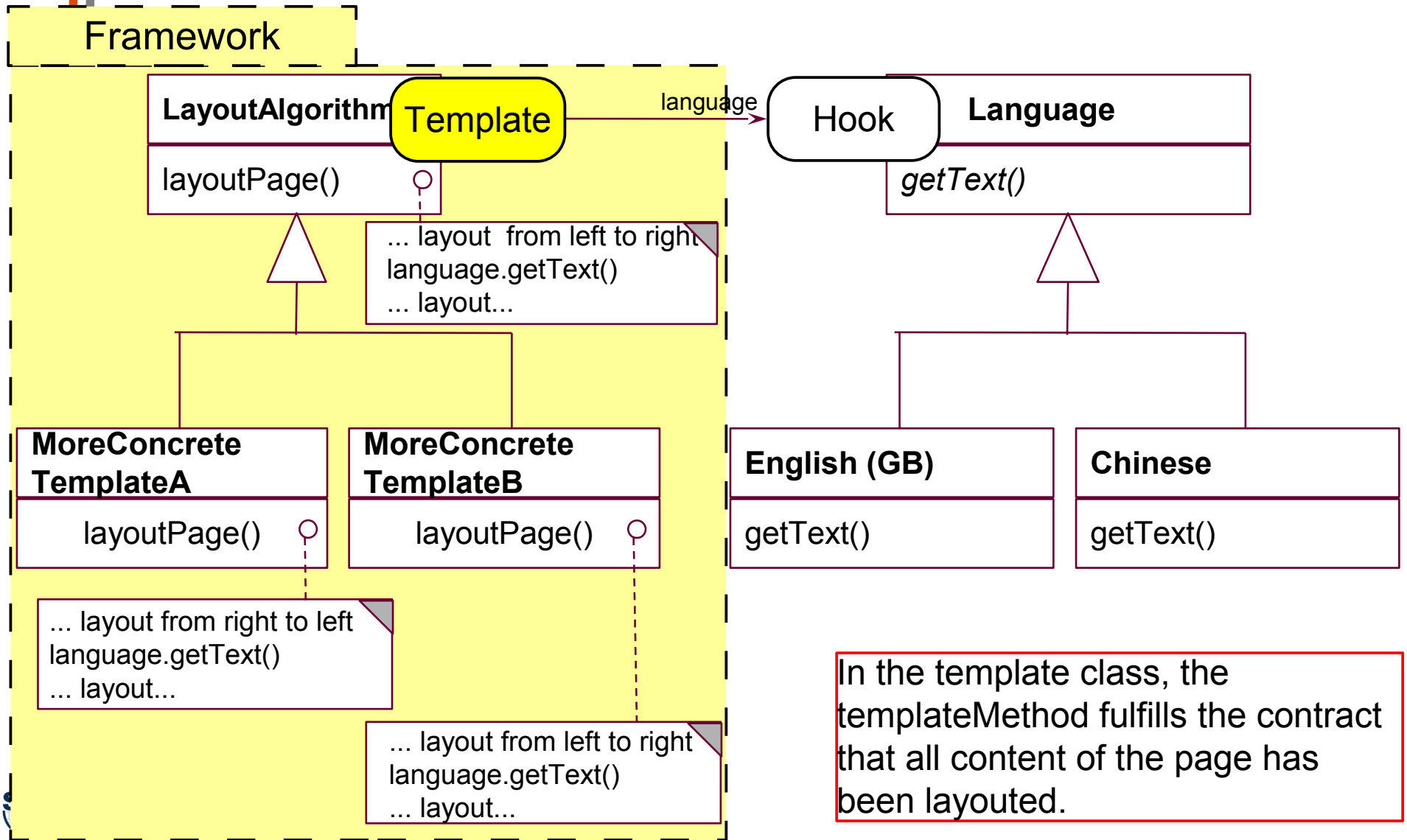


- Template classes cannot be varied *by the client*, only the hook class




Attention: To be a template class, the templateMethod should fulfil a contract!

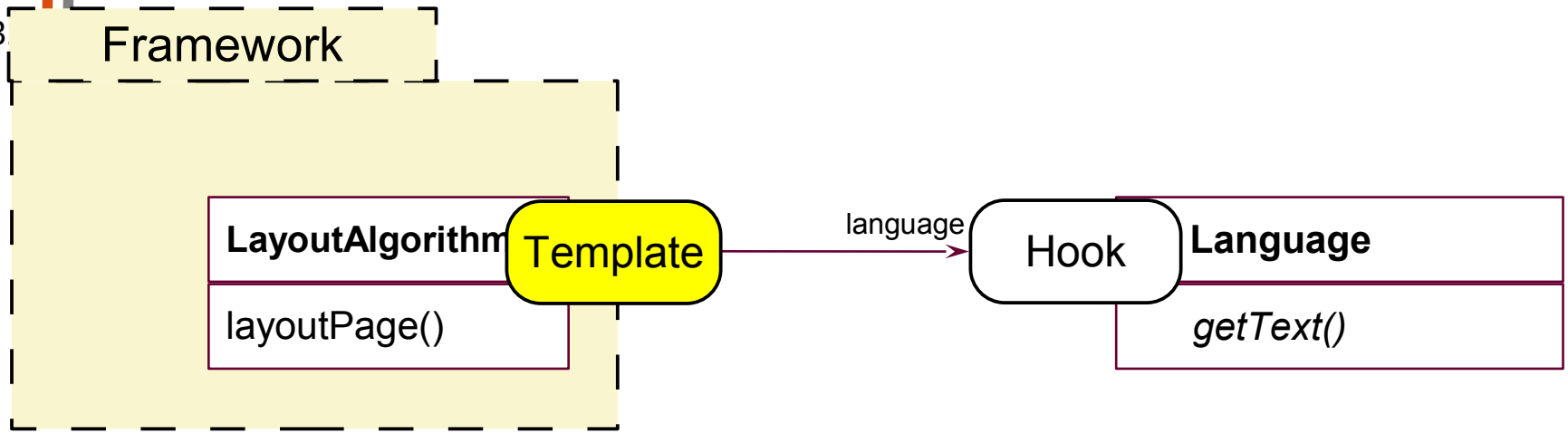
# Ex.: Internationalization as Dimensional Class Hierarchy with 1-T--H



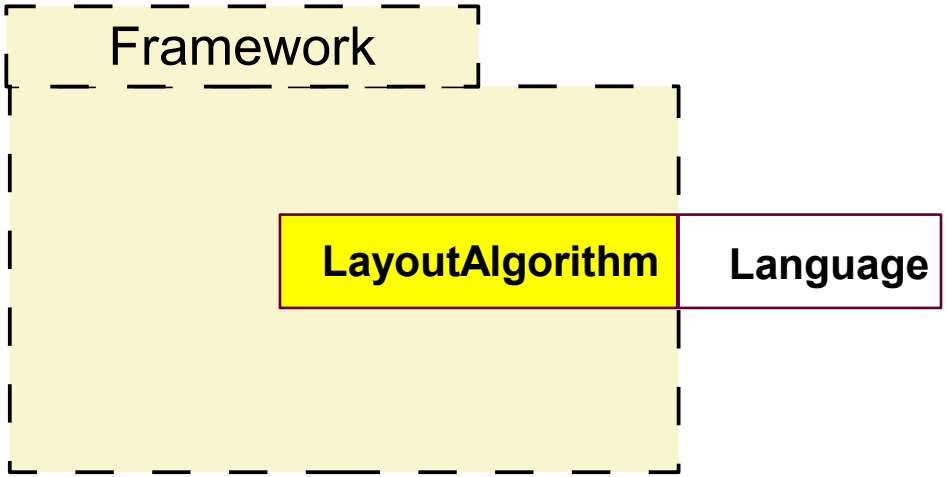

In the template class, the templateMethod fulfills the contract that all content of the page has been layouted.

# Ex.: Internationalization of Frameworks with Dimensional Class Hierarchy with 1-T--H

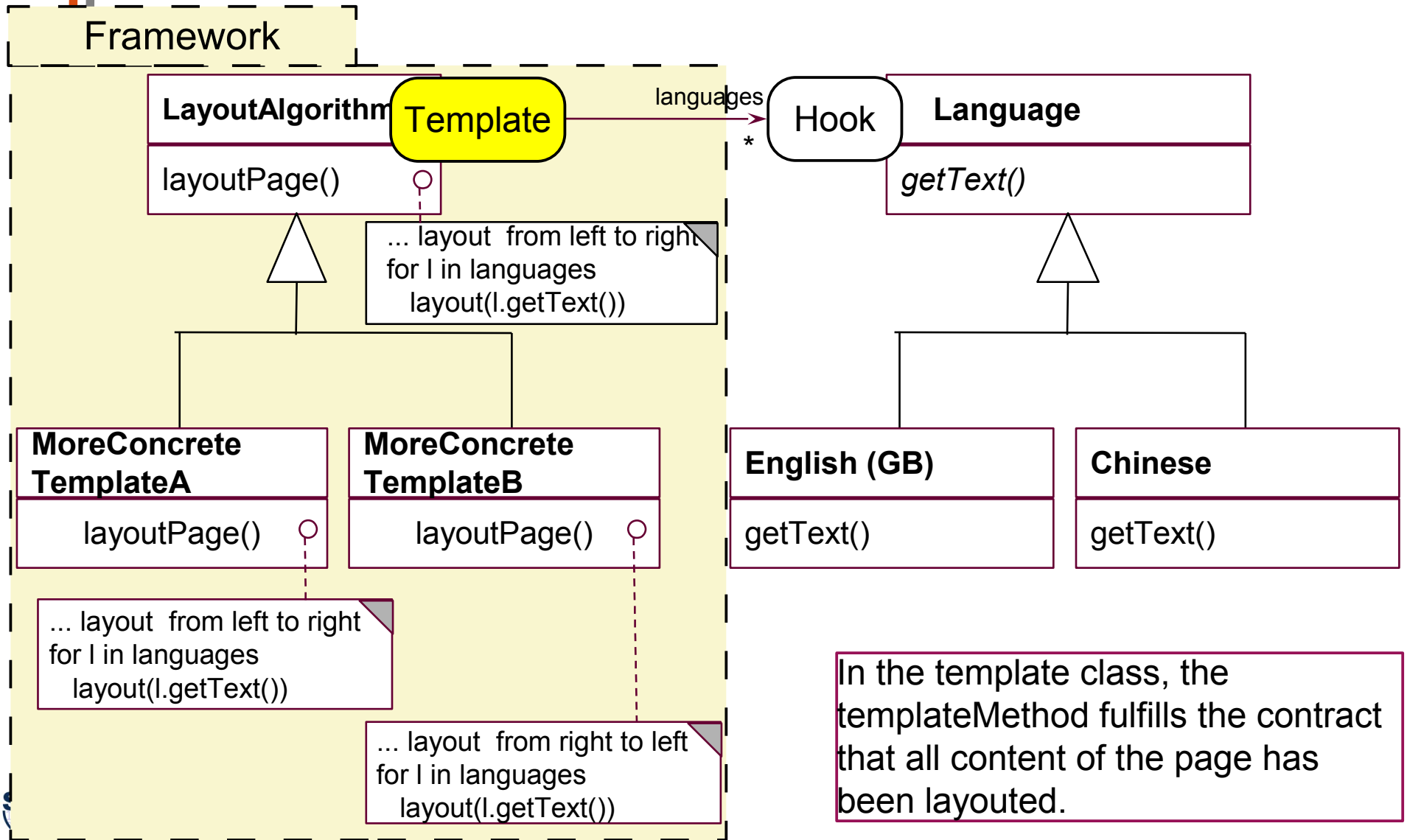





- may be abbreviated with block notation to:



# Ex.: Multiple Internationalization as Dimensional Class Hierarchy with n-T--H



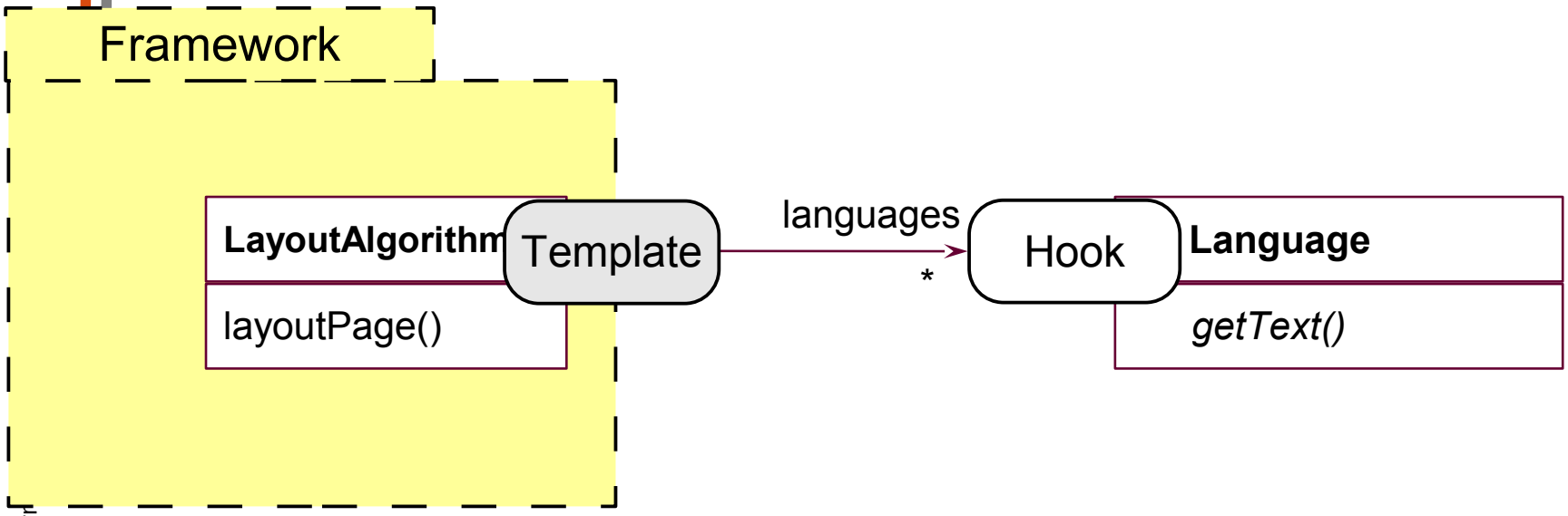

In the template class, the templateMethod fulfills the contract that all content of the page has been layouted.

# Ex.: Multiple Internationalization as Dimensional Class Hierarchy with n-T--H

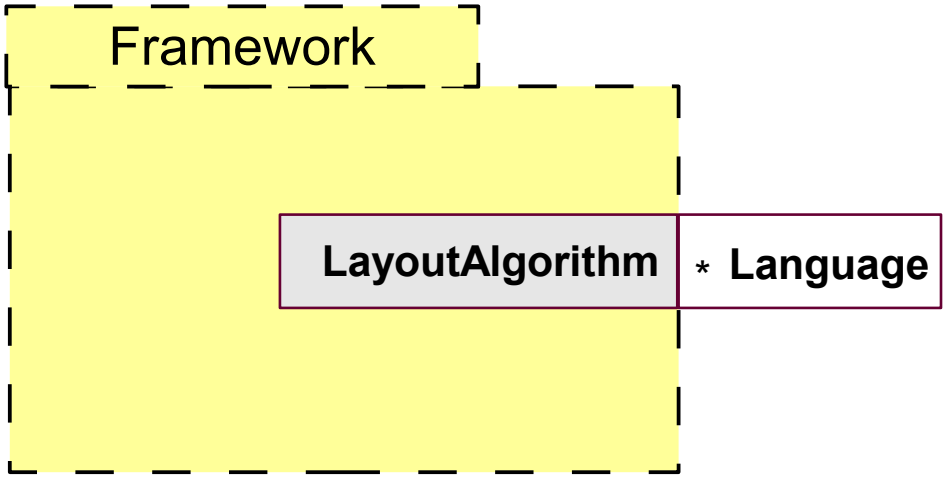


- n-T—H is based on *-Bridge pattern
- This framework hook allows for multiple internationalized texts
  - An application can layout several languages at the same time
- The layout algorithm can be coupled with different languages that use the same layout (multiple internationalization)
- Here, you can see the power of the T—H concept:
  - 1-T--H: dynamic variability
  - n-T—H: dynamic extension (flat, non-recursive)

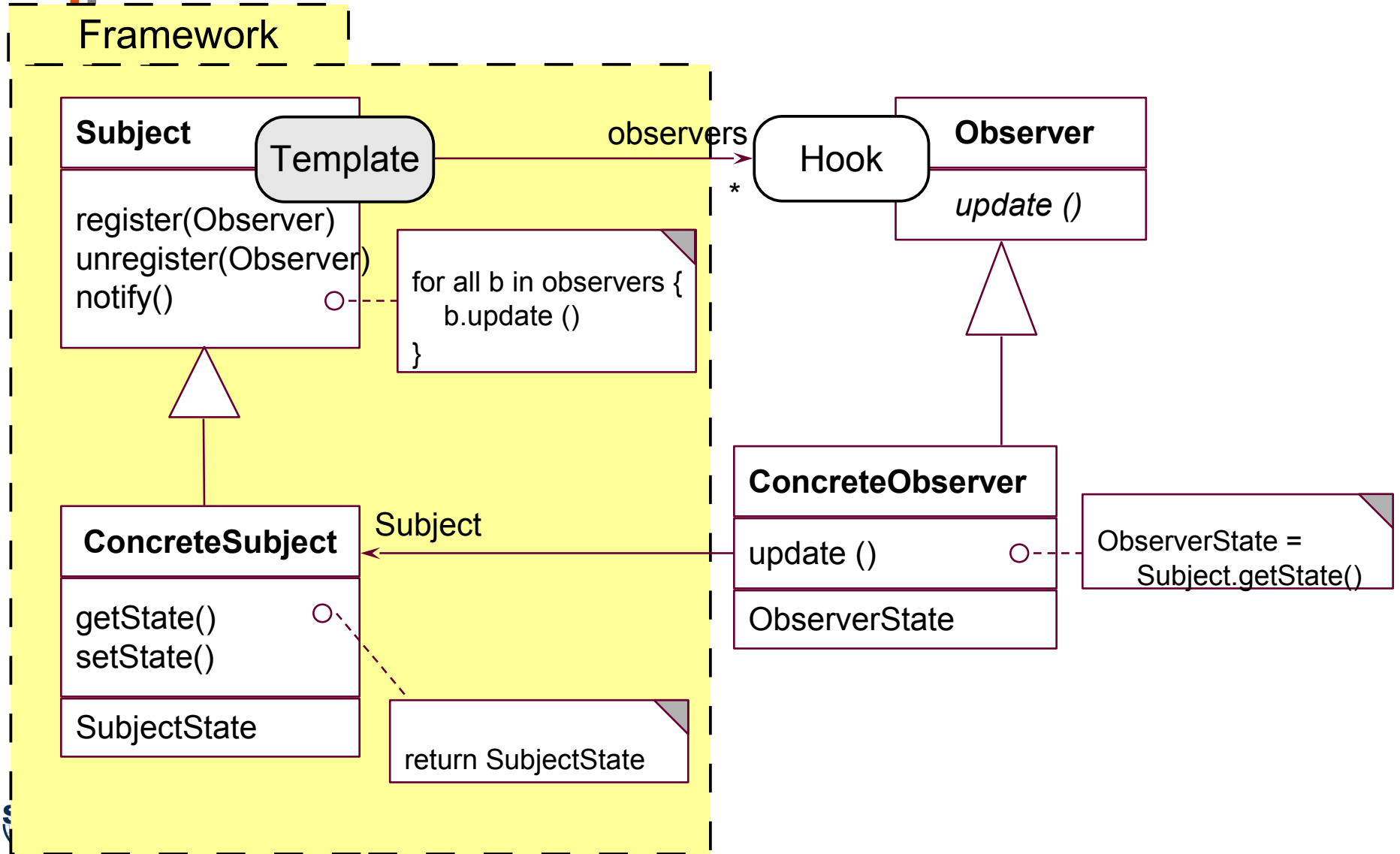
# Ex.: Multiple Internationalization as n-T—H Dimensional Hierarchy




Block notation:

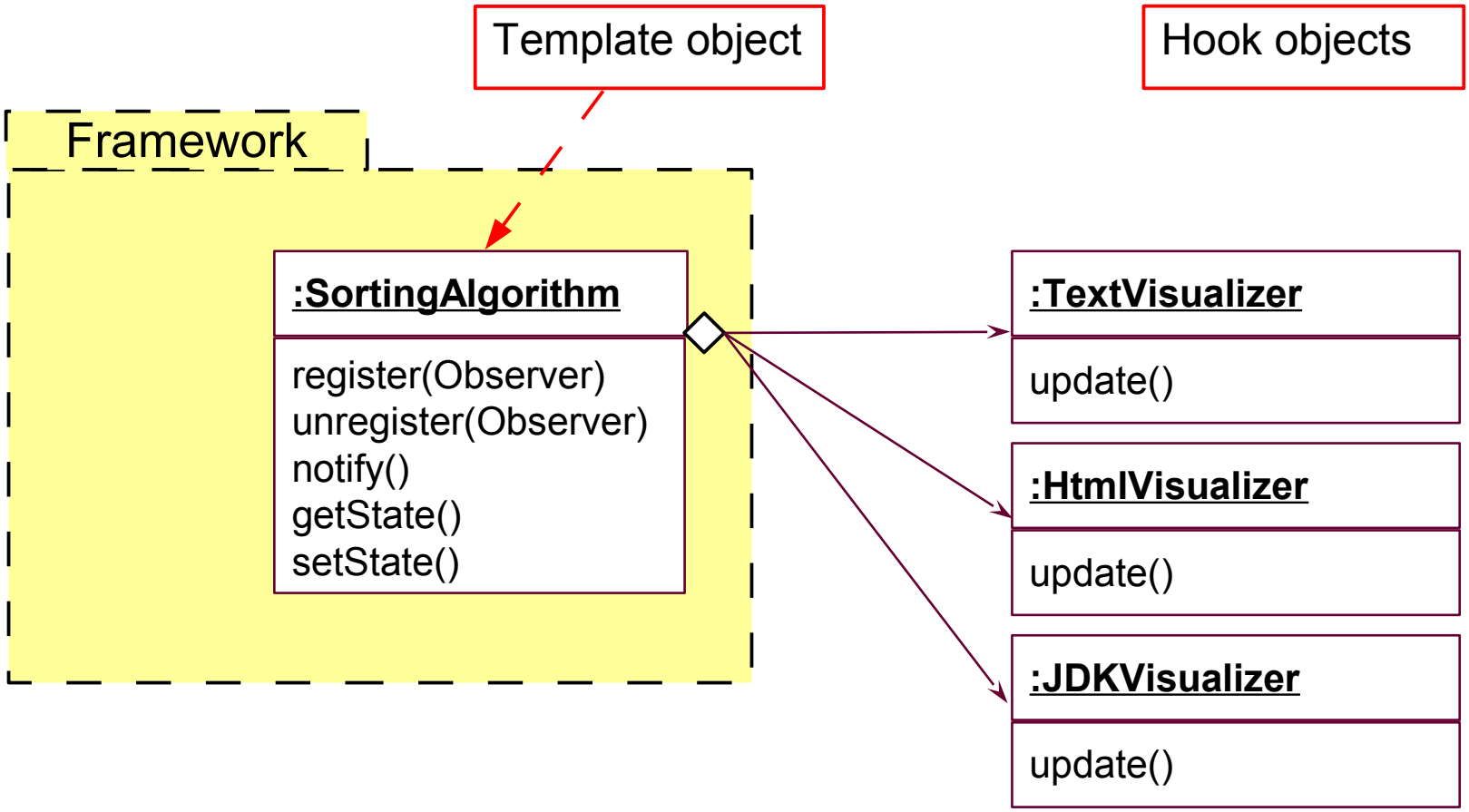


# Observer as n-T—H of a Framework

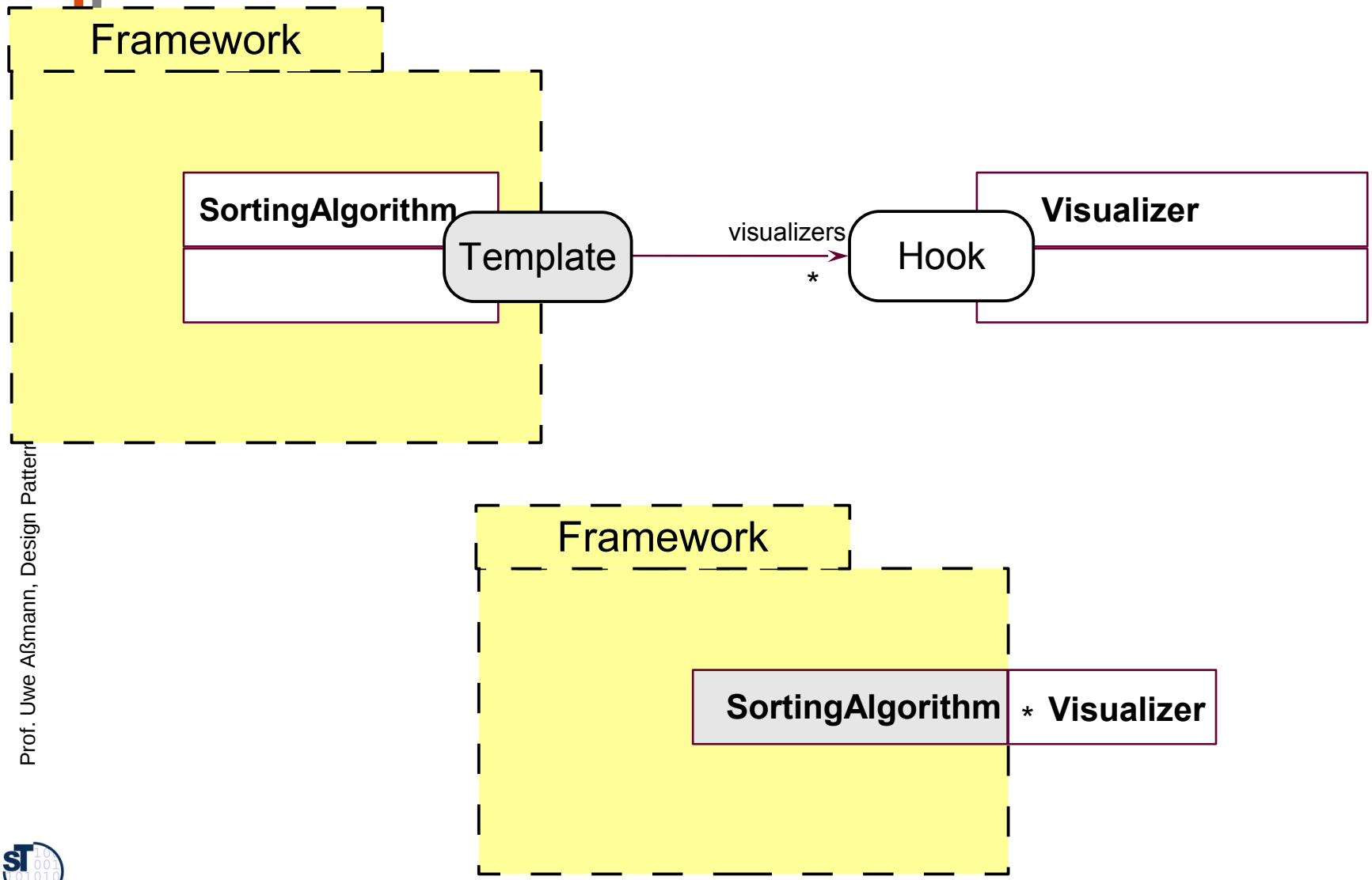


# Observer Runtime Scenario: Several Visualizers in Parallel





# Observer-Based Extensible Frameworks



# Observer

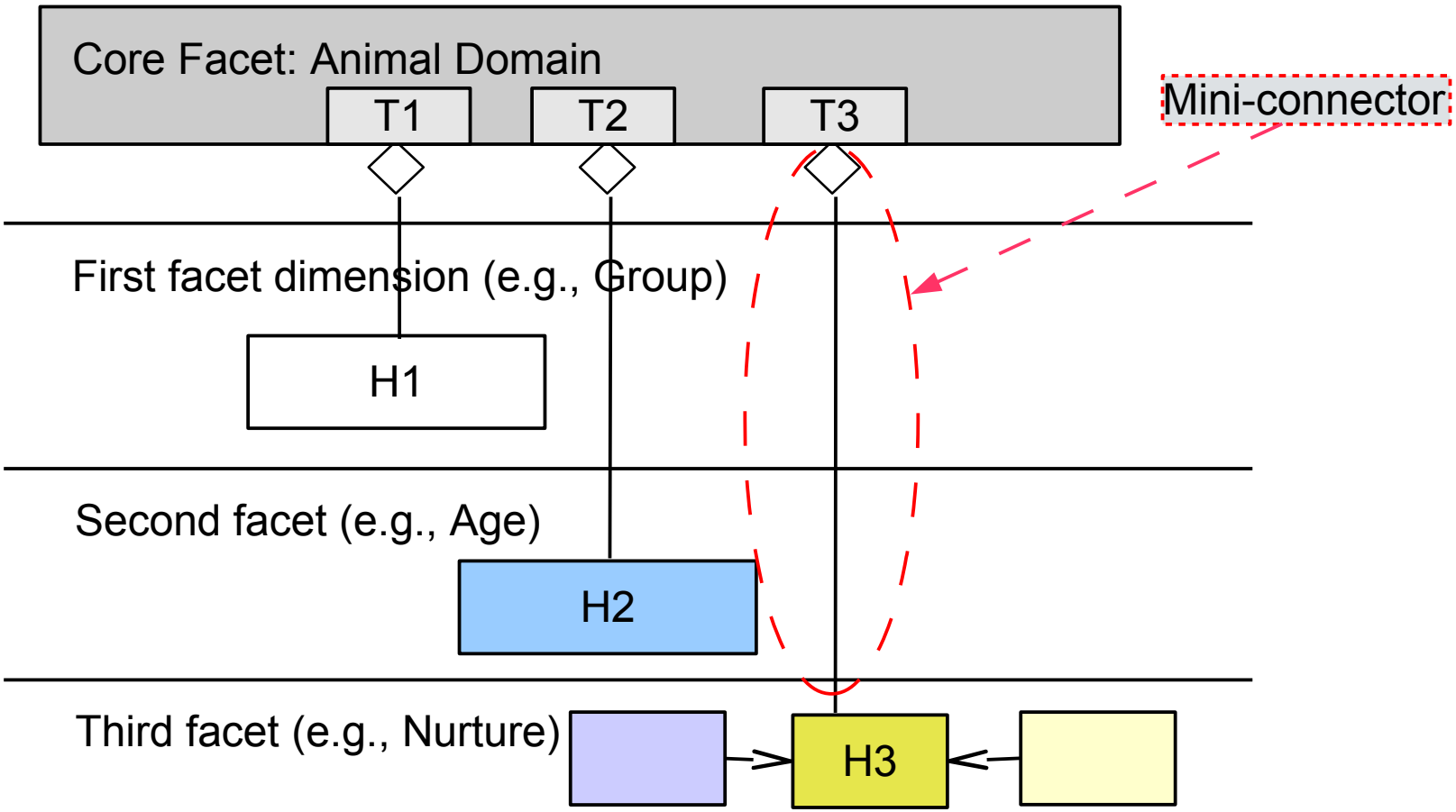


- The Observer pattern is used for extensibility
- With T&H, it becomes clear that Observers are a perfect way to achieve product lines with new feature extensions:
  - Model a critical template algorithm as Subject (template of the n-T--H)
  - Model an extension as a new Observer (hook of the n-T--H)

# Bridge Frameworks Have T—H Hooks

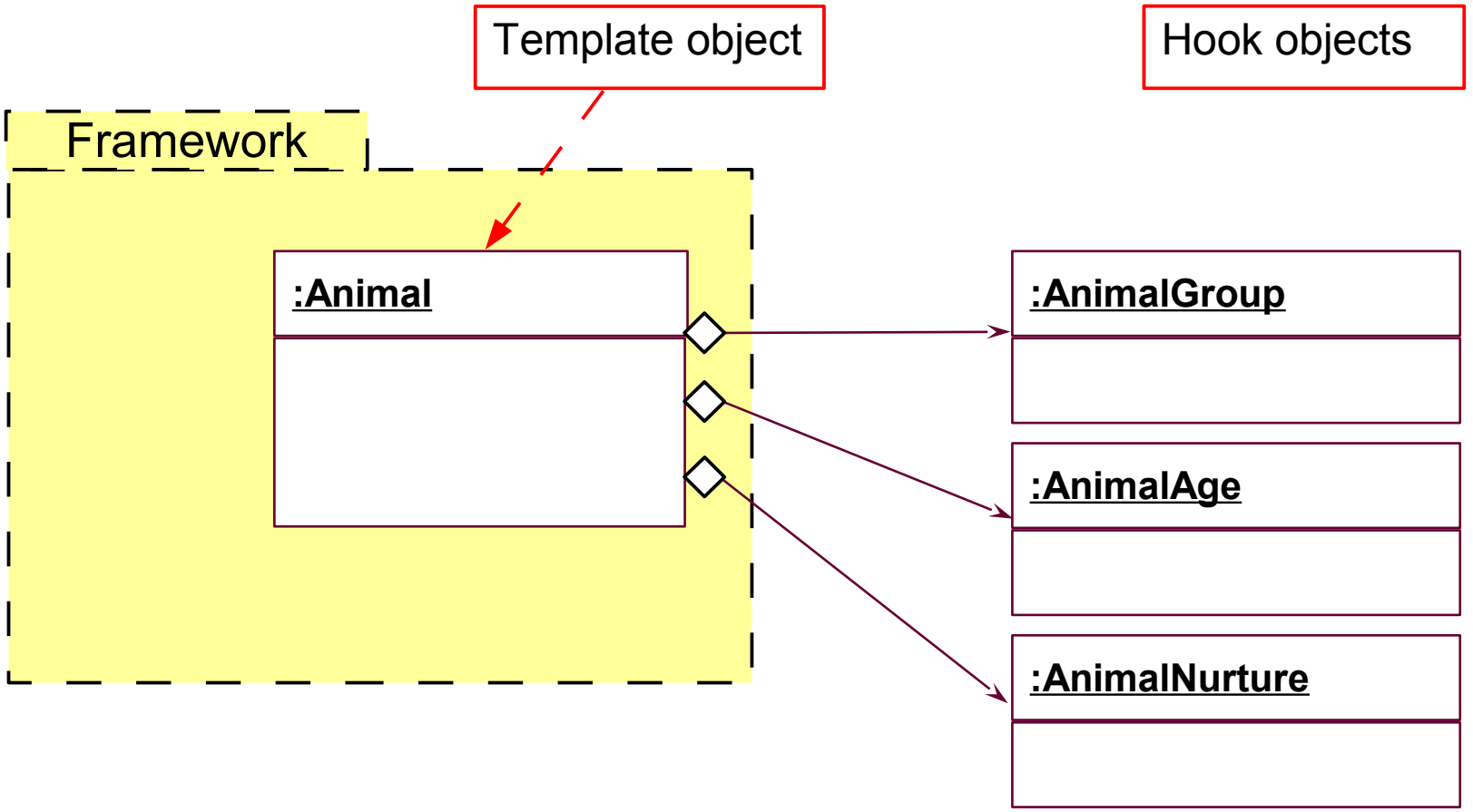


- Every dimension corresponds to a T—H hook
- Bridges, Strategy, Adapter can be used as mini-connectors



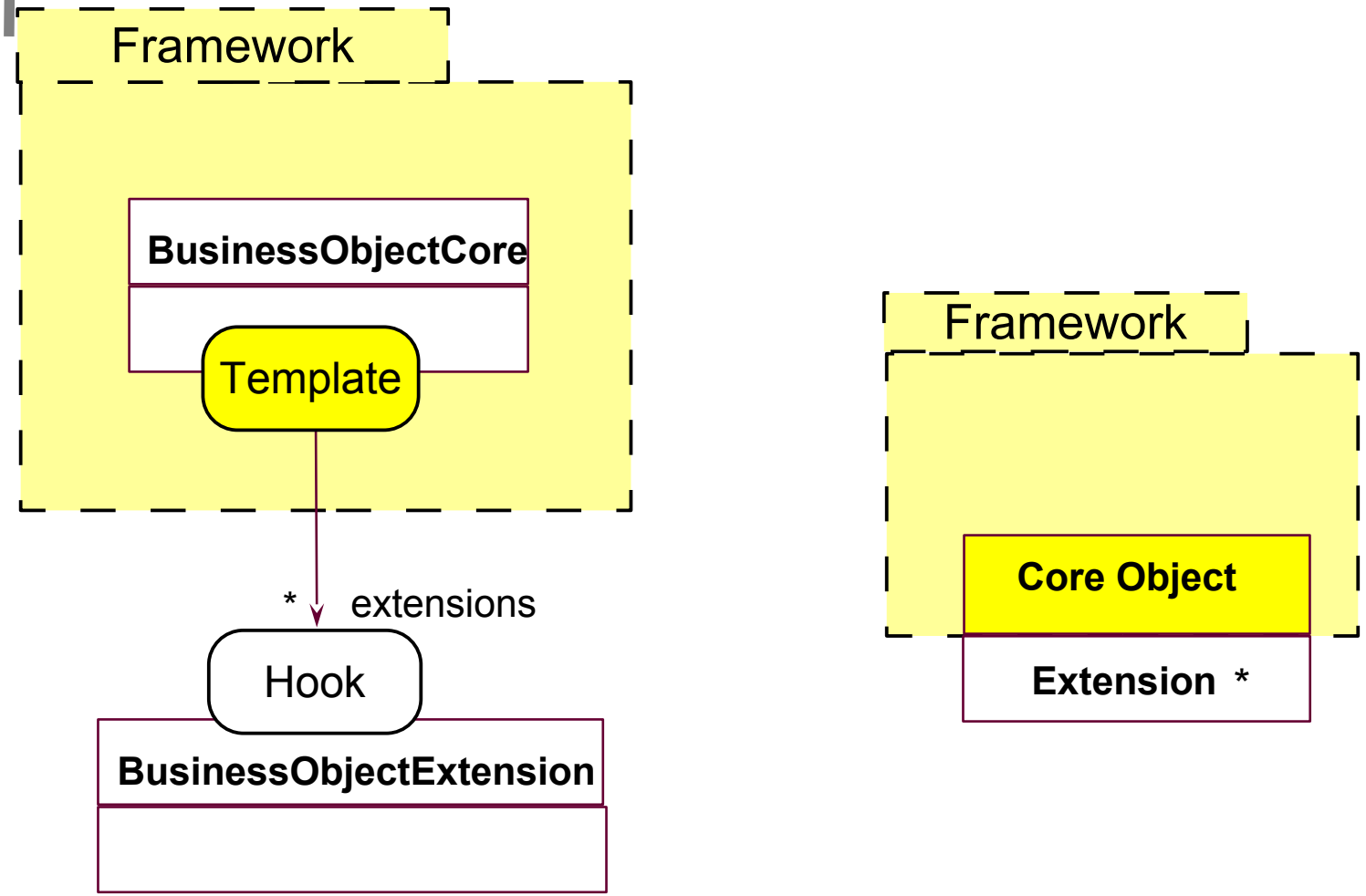
# Ex.: Bridge Framework Runtime Scenario





# Extensible Bridge Framework with n-T--H





# n-T—H Makes Bridge Frameworks Extensible



- An n-T—H framework hook makes dimensional bridge frameworks extensible with new dimensions *at run time*
- New extensions in new dimensions can be added and removed on-the-fly
- Applications
  - Business applications
  - System software
  - 3- and n-tier architectures

# T—H Patterns Result in Blackbox Frameworks



- The main relation between T and H is *delegation*.
- Hence, when overriding and instantiating H, the framework is untouched (*blackbox framework*)
- 1-T—H gives variability
- n-T—H gives extensibility

# 12.4 The H<=T Recursion Metapattern

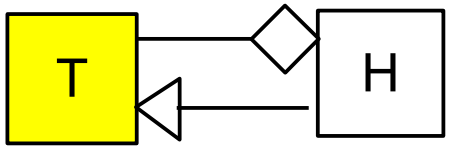


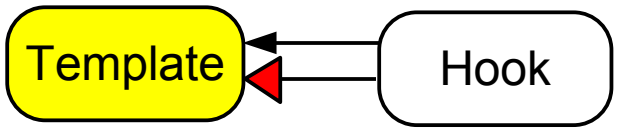
# H<=T Recursive Connection



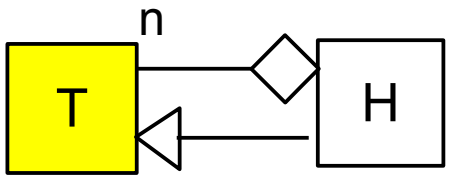
- T&H *recursive connection pattern* (H<=T framework hook, *deep extension pattern*)
  - with 1- or n-ObjectRecursion
  - H-class inherits from T; T is part of H
  - H is decorator of T (1:1) or a composed class in a composite pattern (1:n)

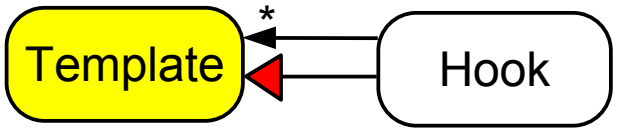
**H<=T (deep list extension)**
T part of H
H inherits from T
1-ObjectRecursion/Decorator




**n-H<=T (deep graph extension)**
H has n T parts
H inherits from T
n-ObjectRecursion/Composite

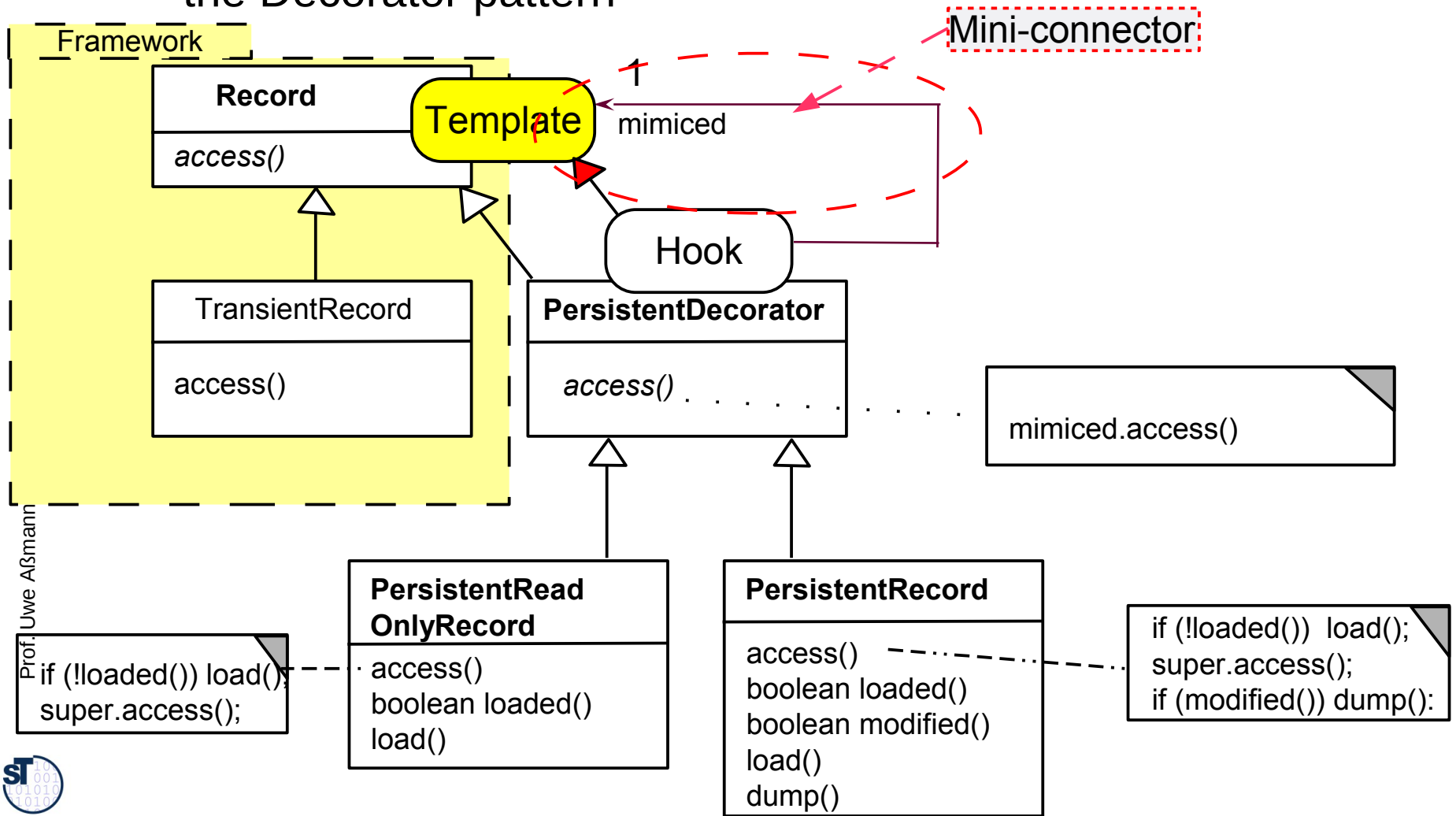


# Decorator as 1-H<=T



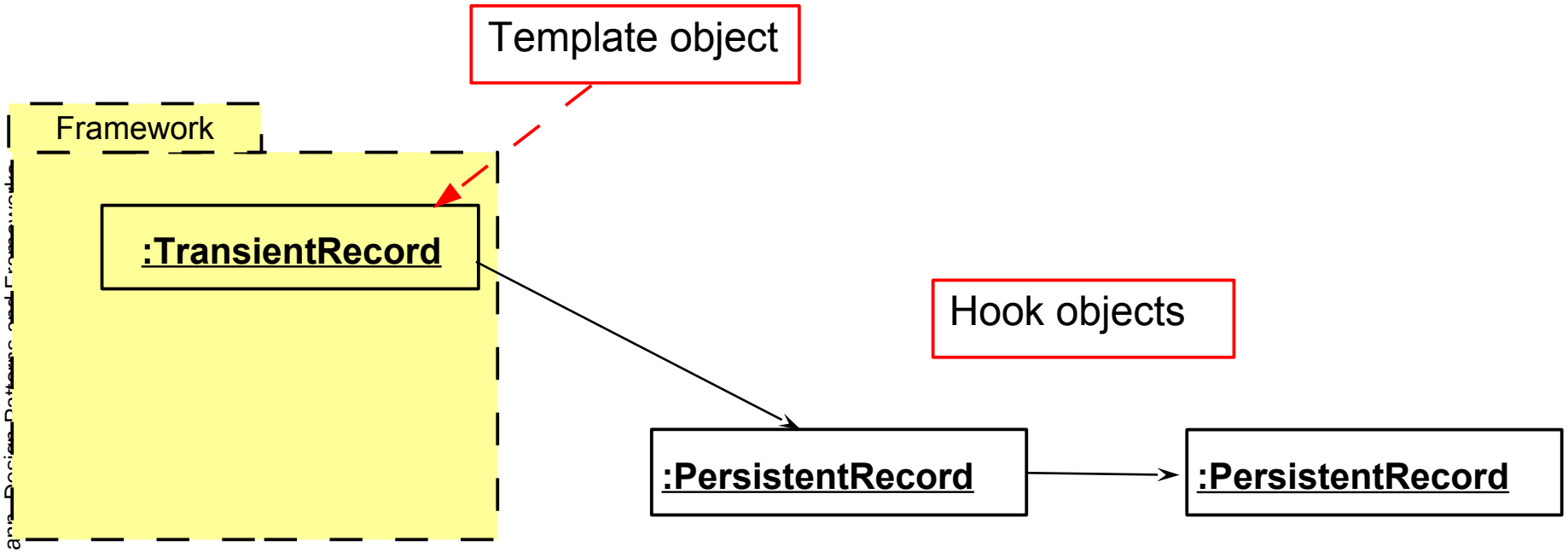
- All decorator objects have to conform to the template class of the Decorator pattern



# Ex.: Run-Time Snapshot of Decorator as Framework Hook Pattern



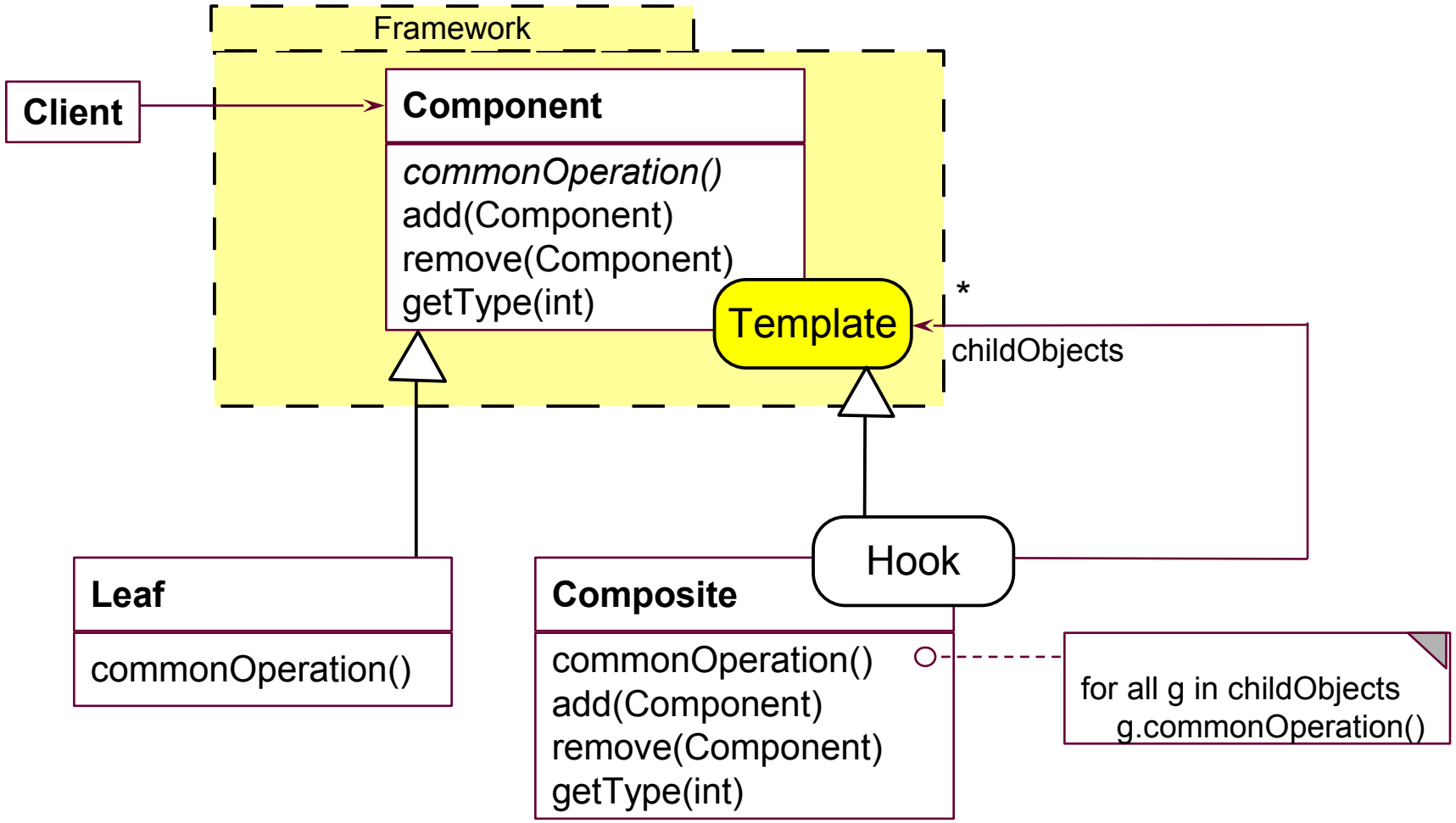
- Lists extend the framework



# Composite as n-H<=T



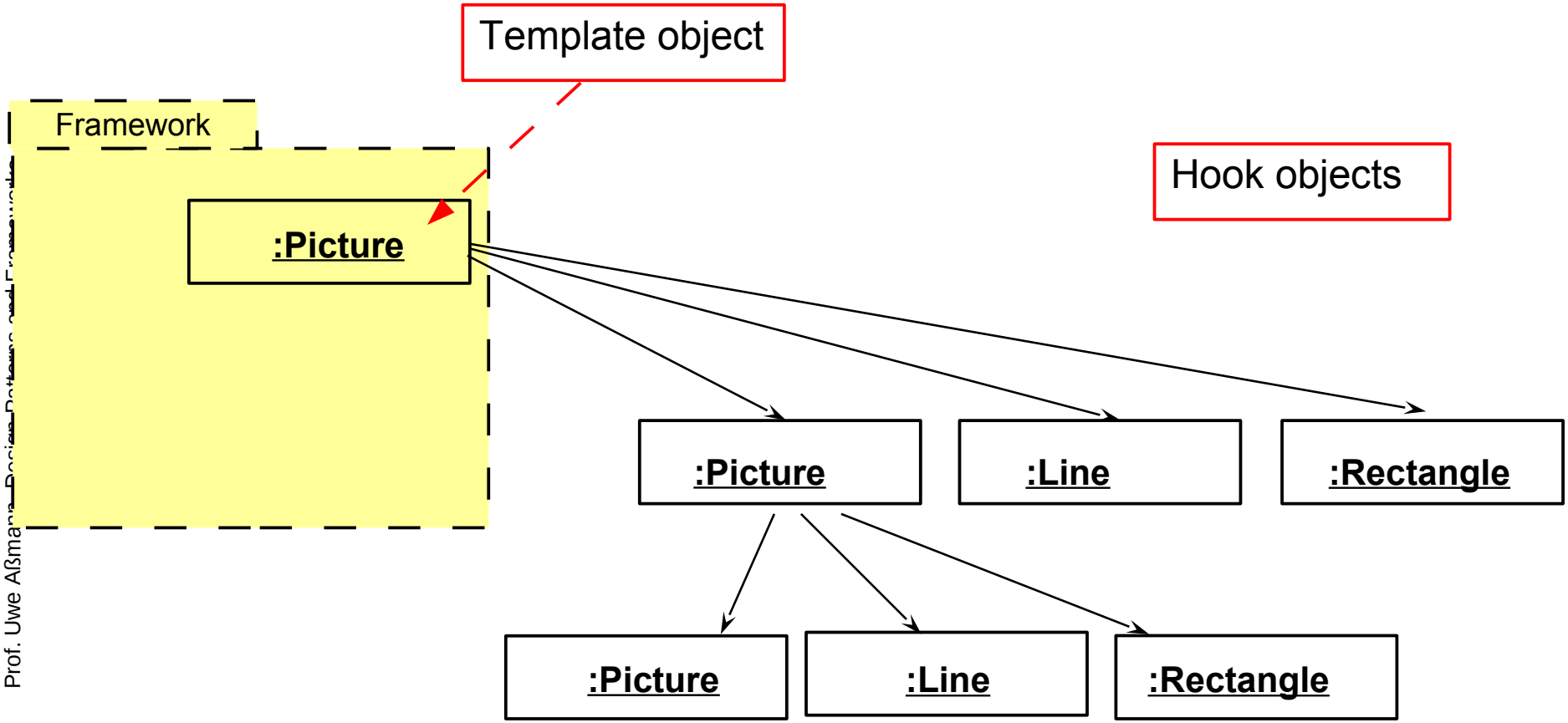
- Composite is an instance of n-ObjectRecursion and n-H<=T



# Ex. Run-Time Snapshot of Composite as Framework Hook Pattern



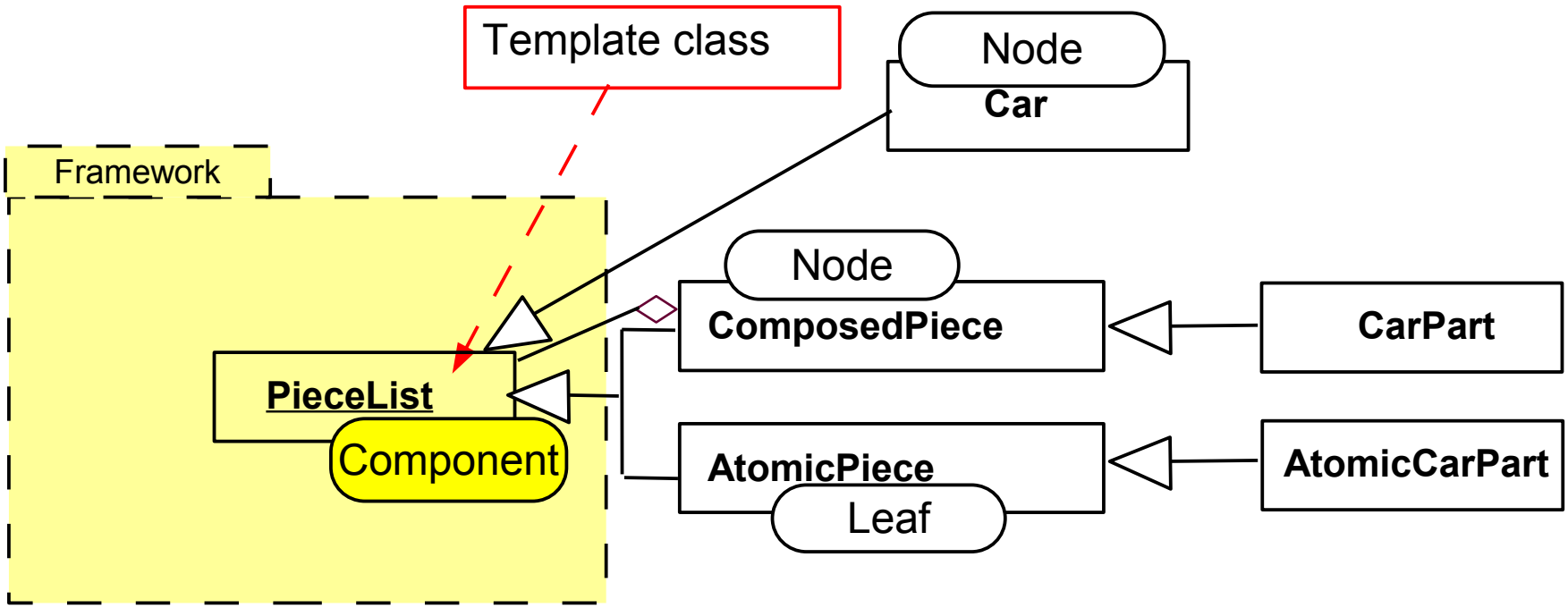
- Part/Whole hierarchies extend the framework



# Production Data Systems



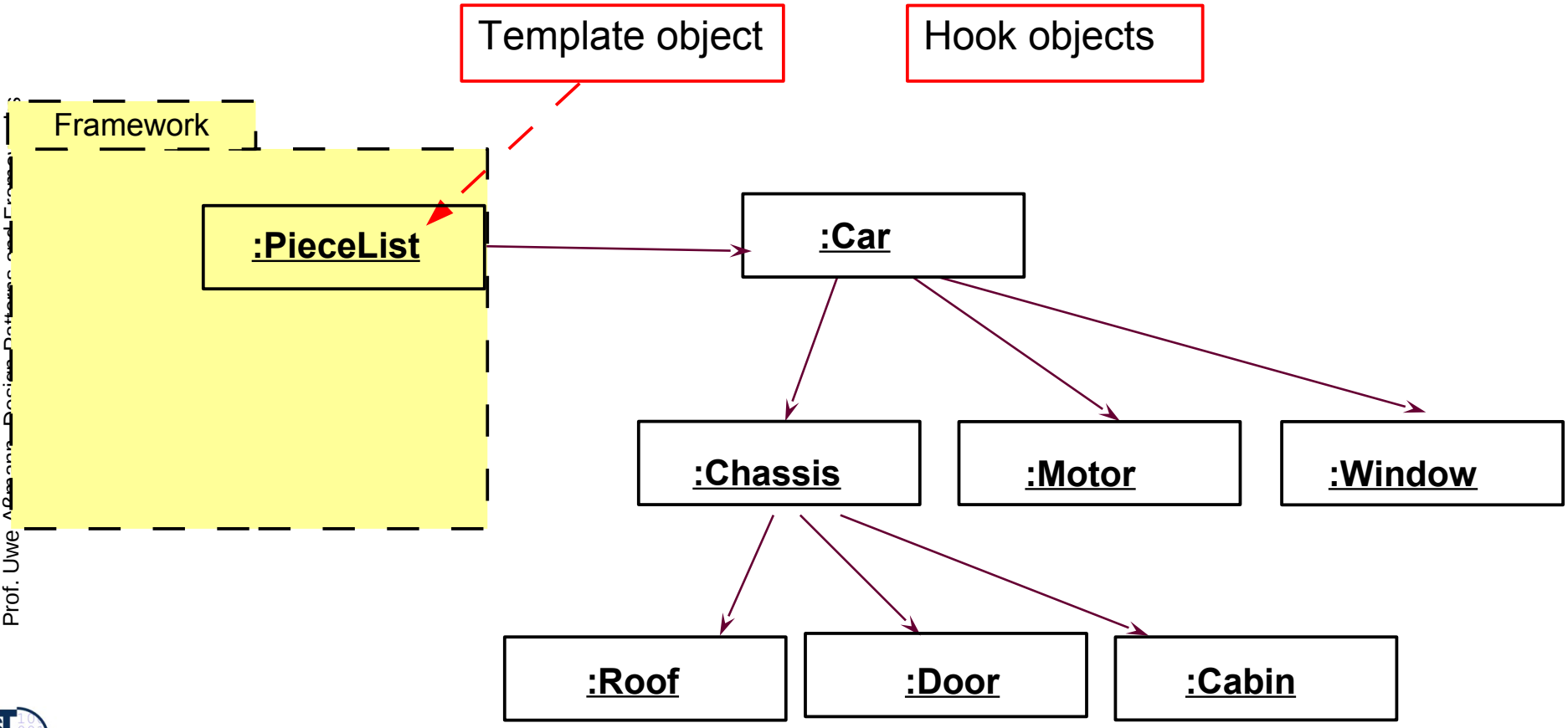
- Piece lists are part/whole hierarchies of technical artifacts in production
- The roles of a composite form the hook of the framework



# Ex. Snapshot of a Production Data System



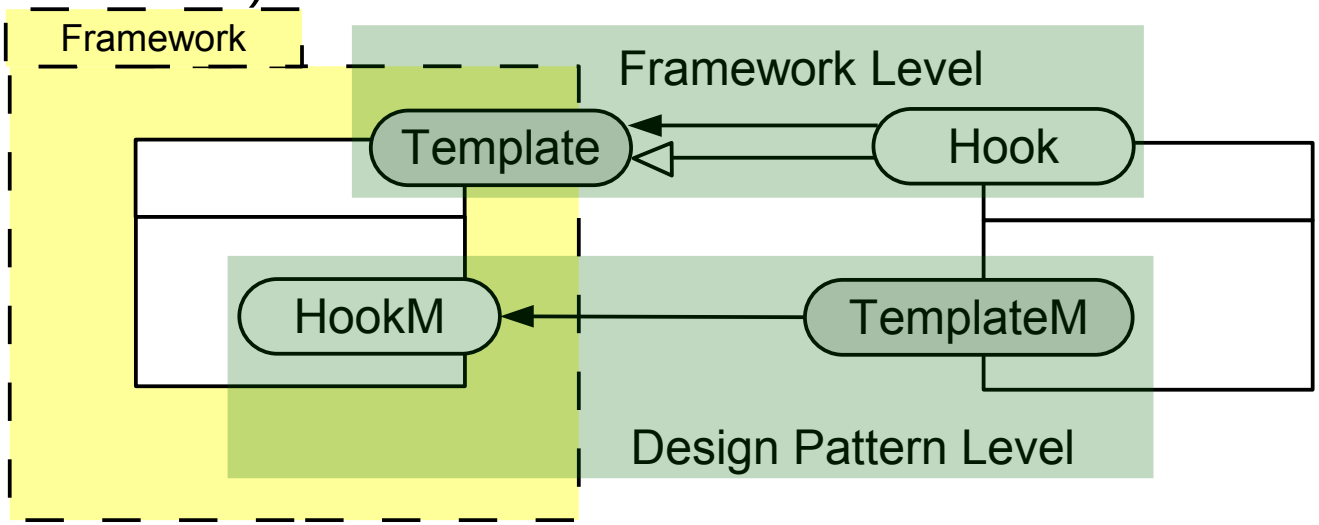
- Piece lists are part/whole hierarchies of technical artefacts in production
- Example: SAP PDM module, IBM San Francisco



# H<=T



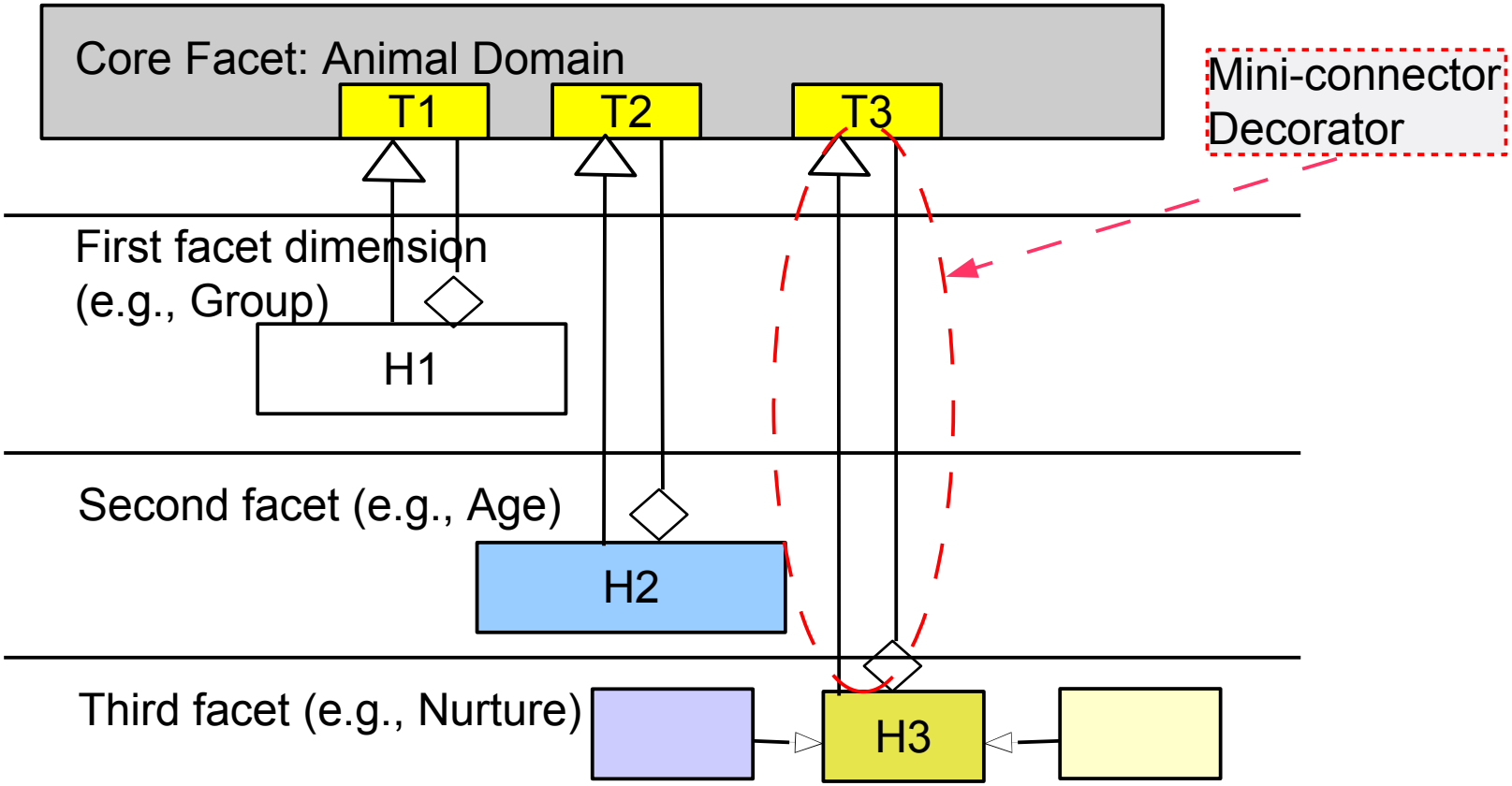
- H<=T framework hooks result in frameworks between black-box and white-box
- Mini-connector H<=T is used
- Attention: The class with the Template role carries the HookM role, the class with the Hook role carries TemplateM role
  - The template (fixed) class in the framework is called from the hook class in the application (which carries the template method role)



# Bridge Frameworks Can Be Done with H<=T (Bridge H<=T Framework)



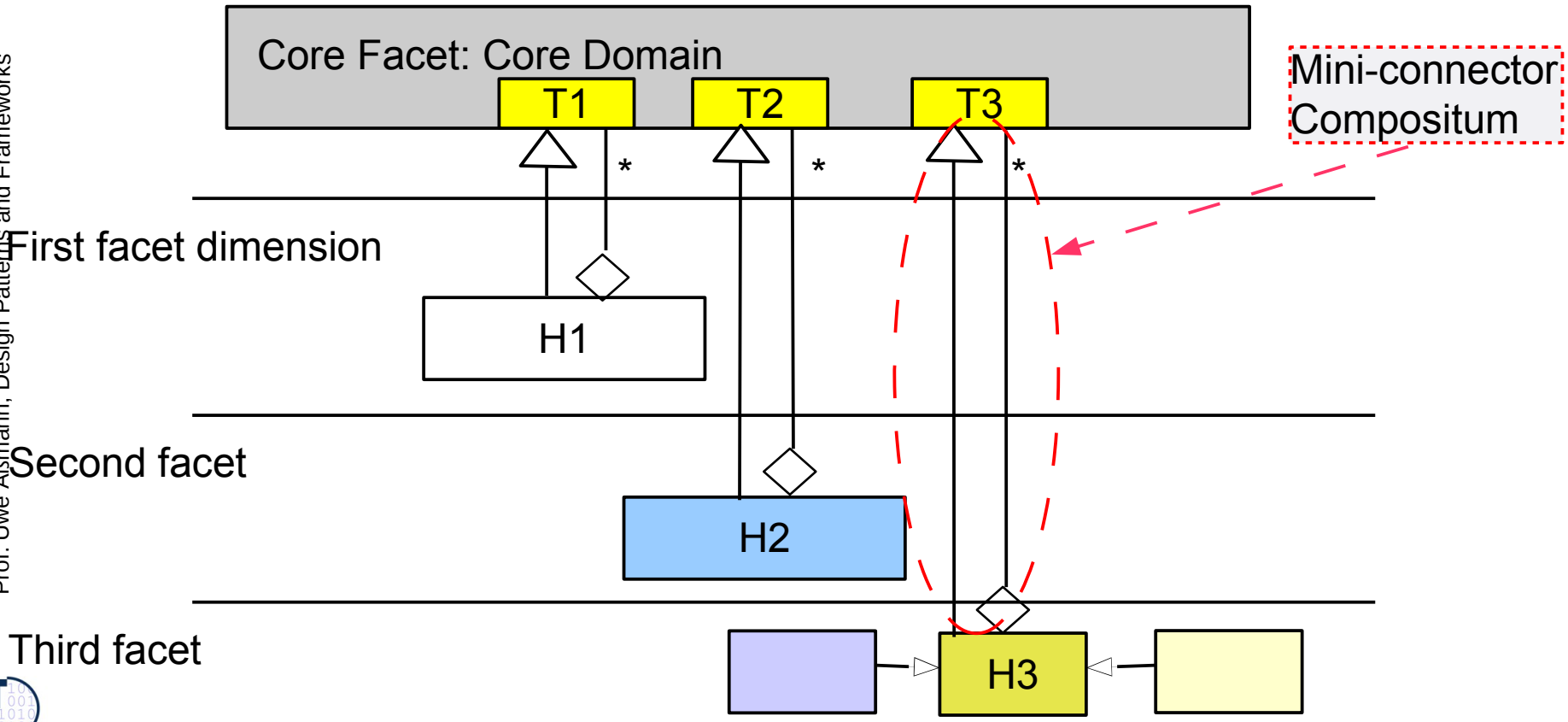
- A dimension may correspond to a H<=T hook of the core framework
- Composite, Decorator, Bureaucracy can be used as mini-connectors



# Bridge Frameworks Can Be Done with H<=T (Bridge H<=T Framework)



- Composite as mini-connector



# 12.5 The TH Unification Metapattern



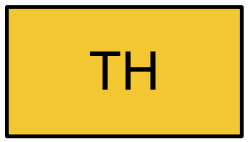
Unification Hooks replace a framework object by a plugin object

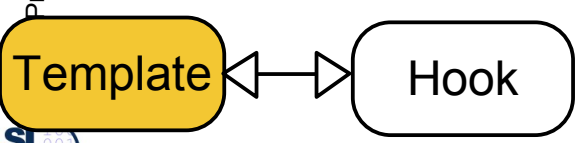
# TH



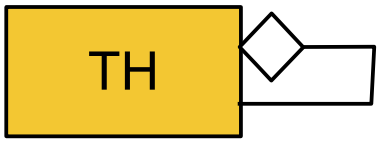
- Unified T&H pattern (TH framework hook)
  - T-class == H-class

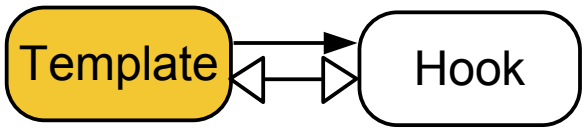
**TH**
T == H
TH part of TH
Decorator



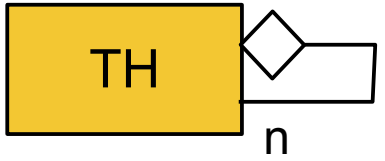



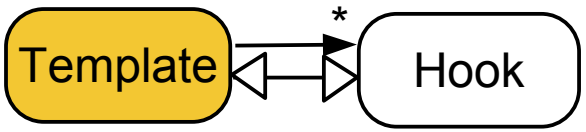

**1-TH (deep list extension)**
T == H
TH part of TH
Decorator




**n-TH (deep tree extension)**
T == H
TH has n TH parts
1:n-Composite

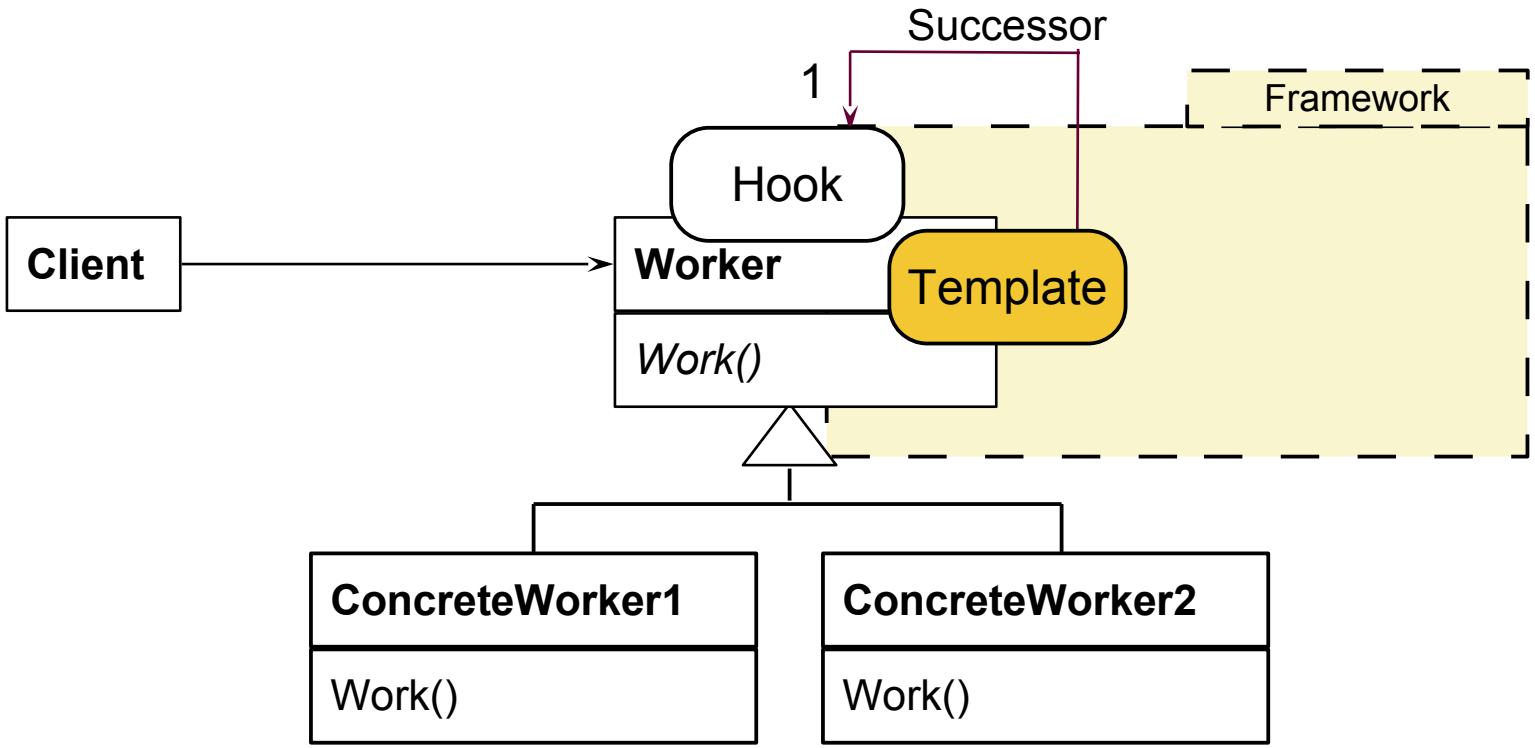


# ChainOfResponsibility as 1-TH

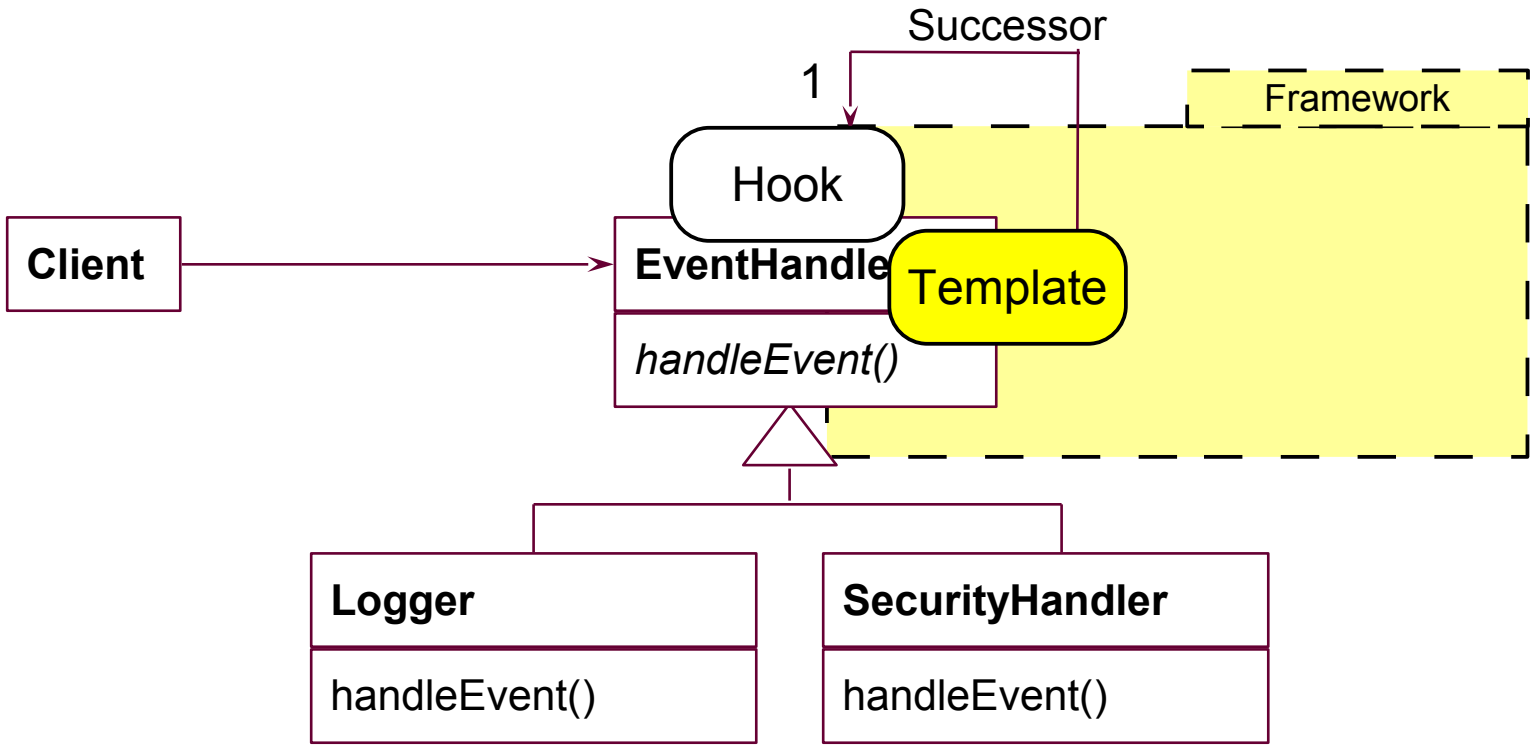


- A Chain is recursing on the abstract super class, i.e.,
  - All classes in the inheritance tree know they hide some other class (unlike the ObjectRecursion)



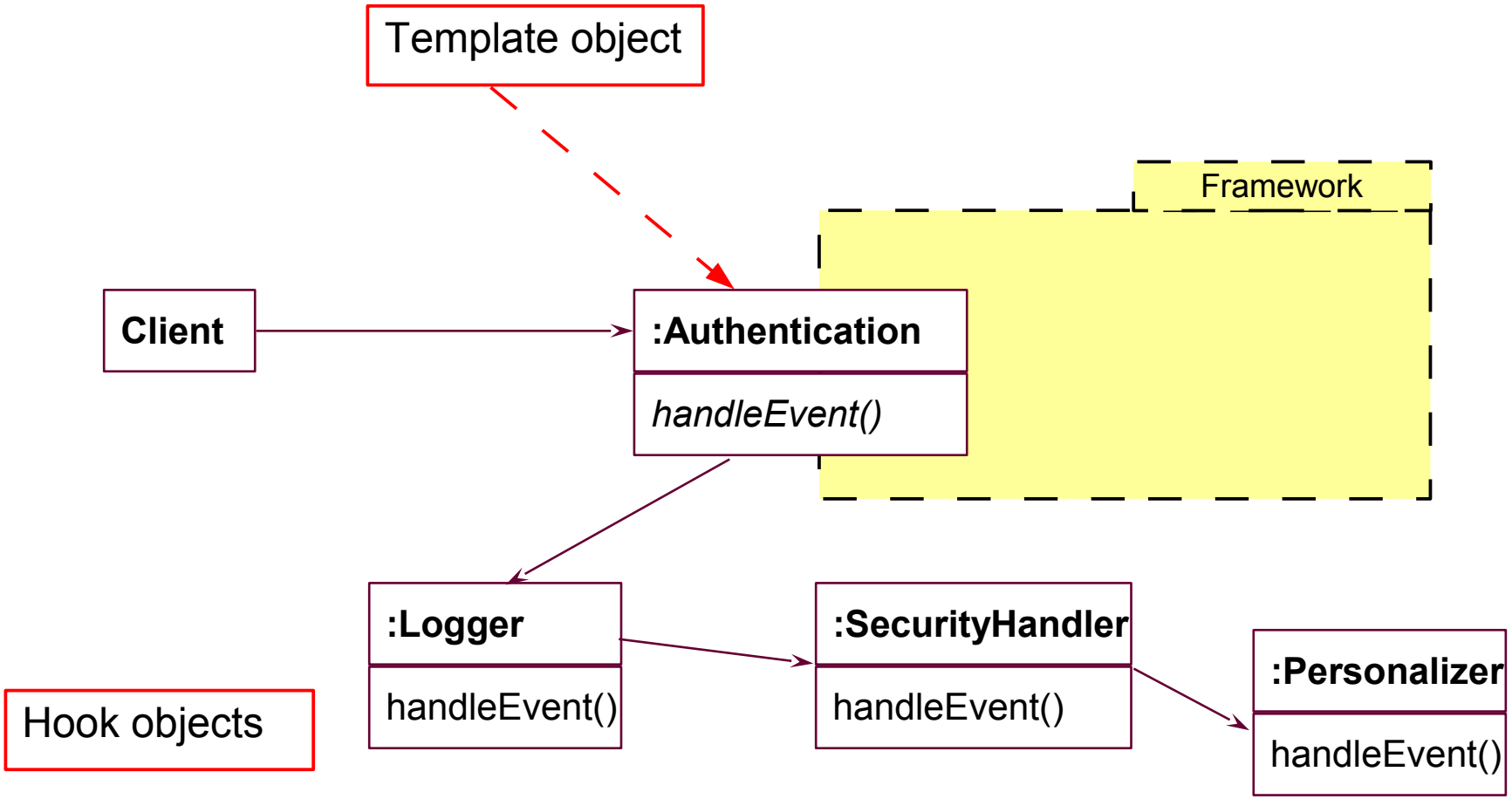
# Ex.: Event Handlers





# Ex.: Snapshot of Event Handlers





# Why TH Unification Makes Sense

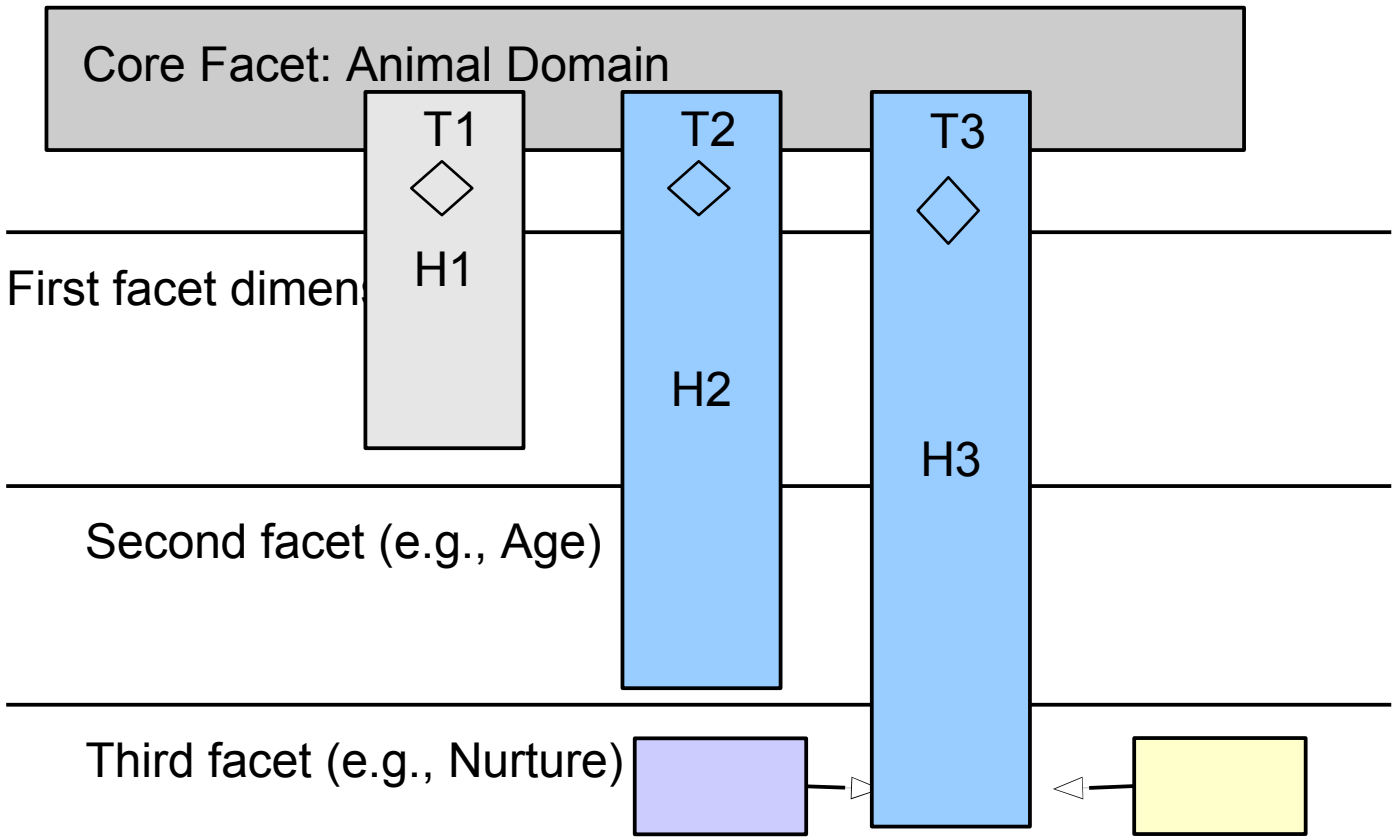


- If a hook class is the same as the template class,
  - Some methods are template methods, others are hook methods
  - Together with the template, the hooks can be exchanged
- Template methods  in the template class are not abstract
  - They are build from referencing hook methods of the hook class
- As we saw in the last chapter, merging role types in one class can make an application faster, but less flexible

# Bridge Frameworks Can Be Done with TH (Bridge TH Framework)



- A dimension may correspond to a H<=T hook
- Chain can be used as mini-connector



# 12.6 The H<T Whitebox Inheritance Metapattern



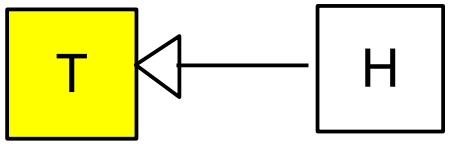
- The object of a plugin, typed by the subclass, replaces the object of the framework, typed by the superclass

# H<T



- If H inherits from T, an H<T framework results (whitebox framework pattern)
  - Whitebox reuse of T in the framework, while deriving H in the application
  - (not of Pree, earlier known)
- If a hook class inherits from a template class, it inherits the skeleton algorithm
  - Template methods  in the template class are not abstract, but concrete
  - They are build from referencing hook methods of the hook class
- An H<T framework hook means a *whitebox framework*

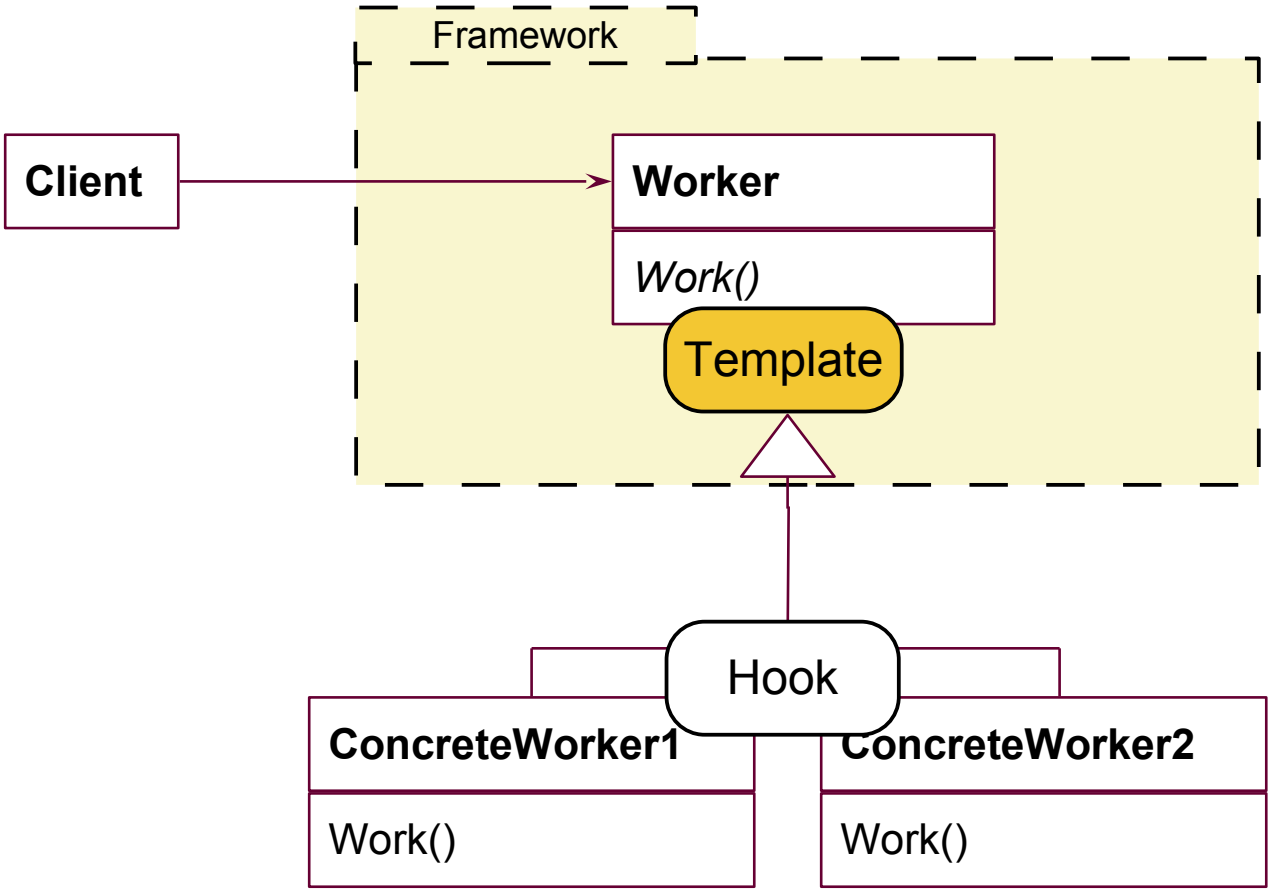
H<T



# Whitebox Framework with H<T Framework Hook



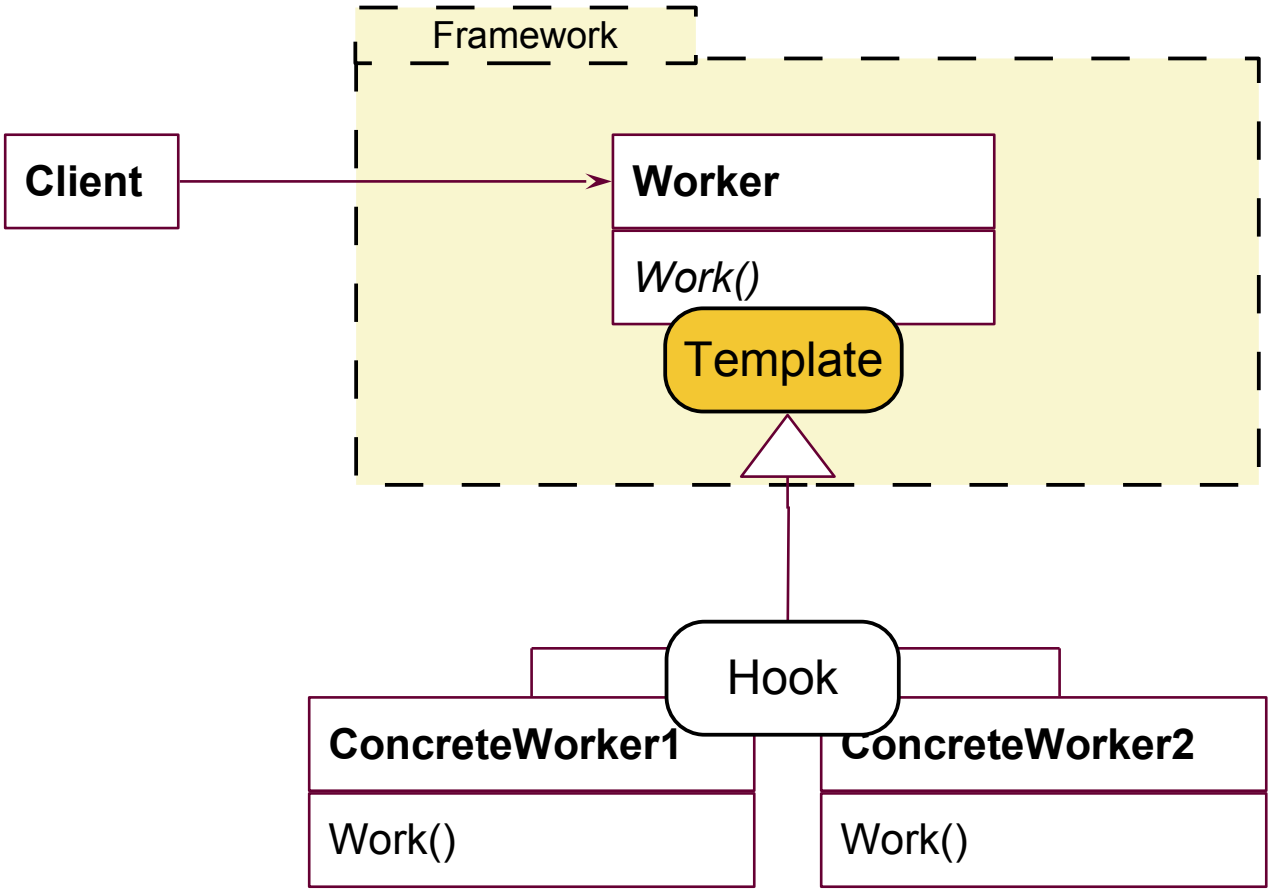
- Also TemplateMethod can be applied (HookM <= TemplateM)



# Whitebox Framework with H<T Framework Hook



- Also TemplateMethod can be applied (HookM <= TemplateM)



# Summary of T&H Patterns and Framework Hooks

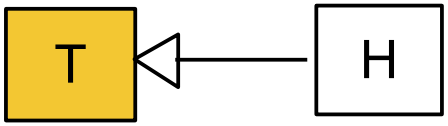


# Framework Hook Patterns



## Inheritance

**H<T**
H inherit from T
whitebox

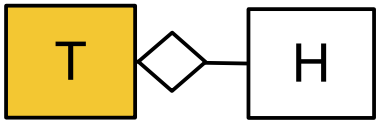



## Unification

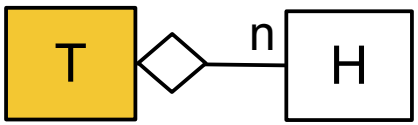
**TH**
T == H


## Aggregation/Association

**T--H**
H part of T
T is core class of
complex object

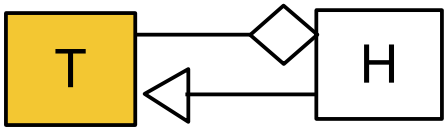



**n-T--H**
T has n H parts
T is core class of
complex object

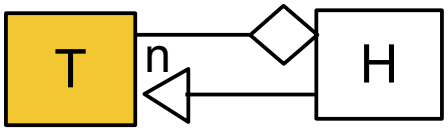



## Recursion

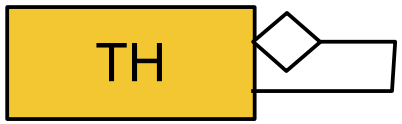
**H<=T**
T part of H
H inherit from T
Decorator



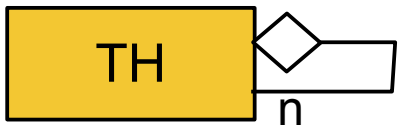

**n-H<=T**
H has n T parts
H inherit from T
1:n-Decorator




**1-TH**
T == H
TH part of TH
Decorator

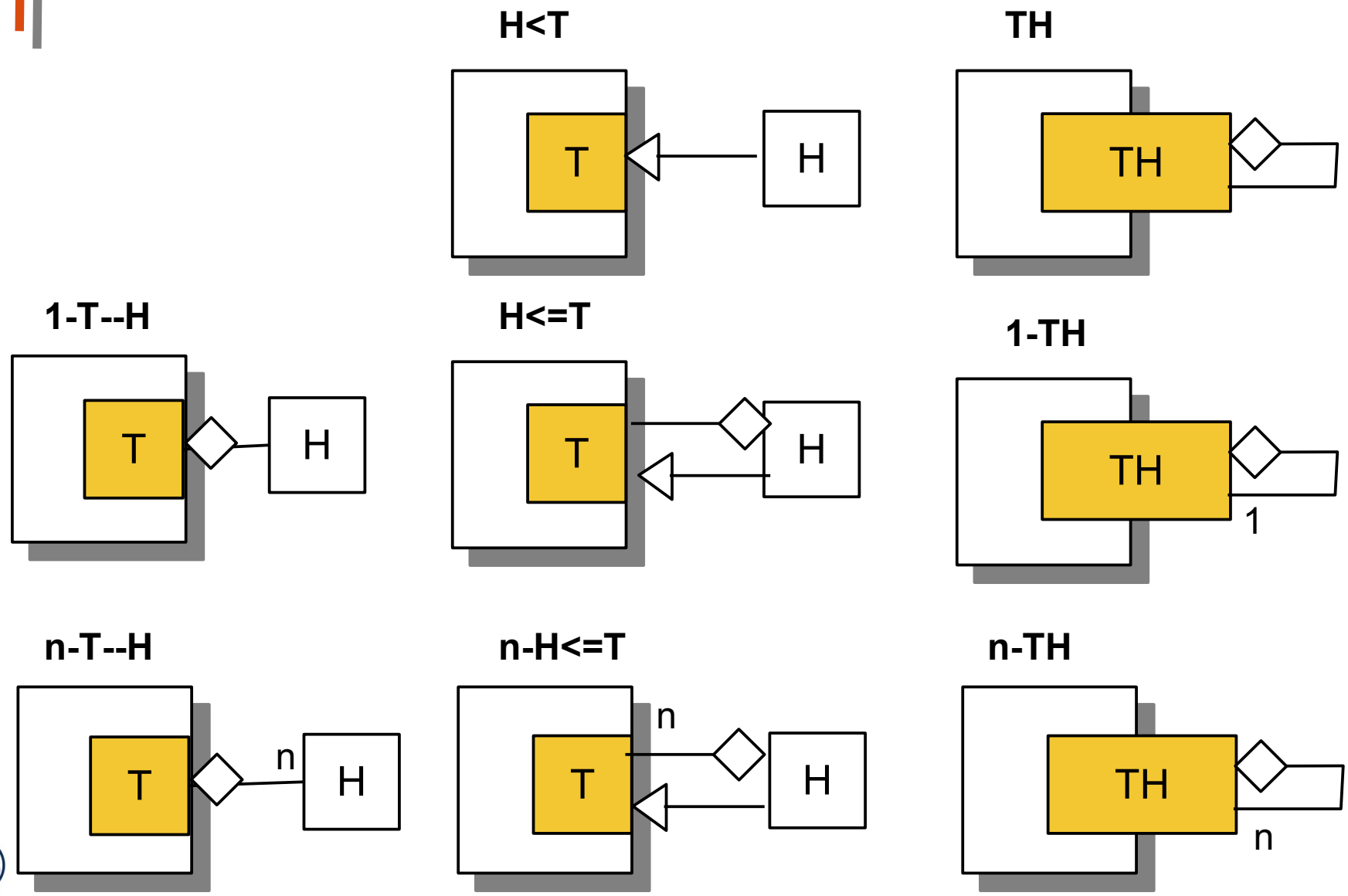



**n-TH**
T == H
TH has n TH parts
1:n-Composite



# Mini-Connector Notation for Framework Hooks

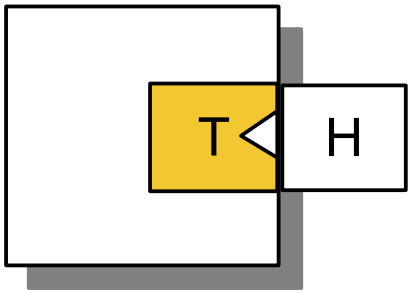




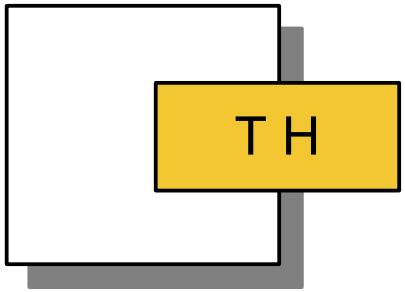
# Block Notation for Framework Hooks



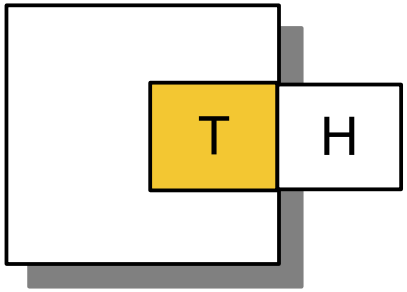
H<T



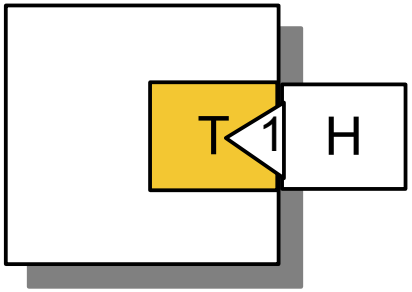

TH



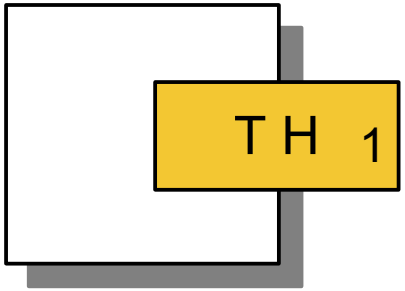

1-T--H



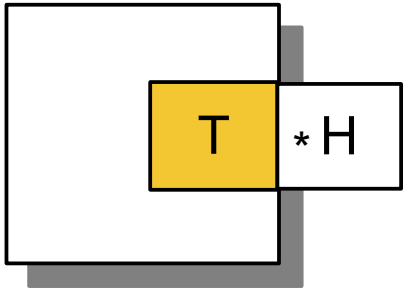

H<=T



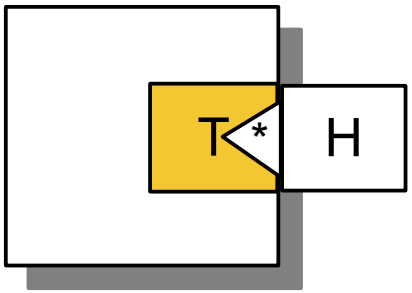

1-TH



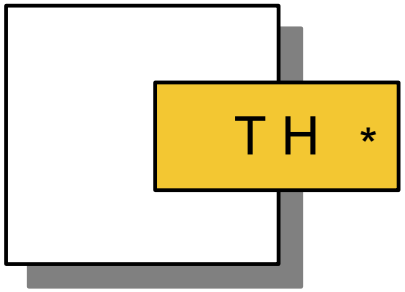

n-T--H




n-H<=T




n-TH



# Summary



- When overlayed  with a T--H metapattern, a design pattern becomes a *framework hook pattern* for the interface of a framework
- These are *mini-connectors* between a framework and its application classes
  - More flexible that just generic classes (generic frameworks) or delegation (blackbox) or inheritance (whitebox)
- The framework hook patterns determine very precisely how a framework is to be instantiated
- There are more kinds of dimensional frameworks
  - Dimensional T—H (n-Bridge LF), H<=T, TH, T>H  dimensional frameworks
- 1-T&H framework hook patterns can be used for variability of the framework
- n-T&H for extensibility.

# The End