2. Simple Patterns for Variability

Dr.-Ing. Sebastian Götz

Software Technology Group
Department of Computer Science
Technische Universität Dresden
WS 16/17, 18.10.2016

1) **Basic Template-And-Hook Patterns**
2) **Faceted Objects with Bridges**
3) **Layered Objects**
4) **Dimensional Systems**
5) **Layered Systems**

Literature (To Be Read)

- V. Caisin. **Creational Patterns.** Paper in Design Pattern seminar, IDA, 2001. Available at home page.
- GOF, **Chapters on Creational and Structural Patterns**
- Another good book:

  Head First Design Patterns. Eric Freeman & Elisabeth Freeman, mit Kathy Sierra & Bert Bates. O'Reilly, 2004, ISBN 978-0-596-00712-6

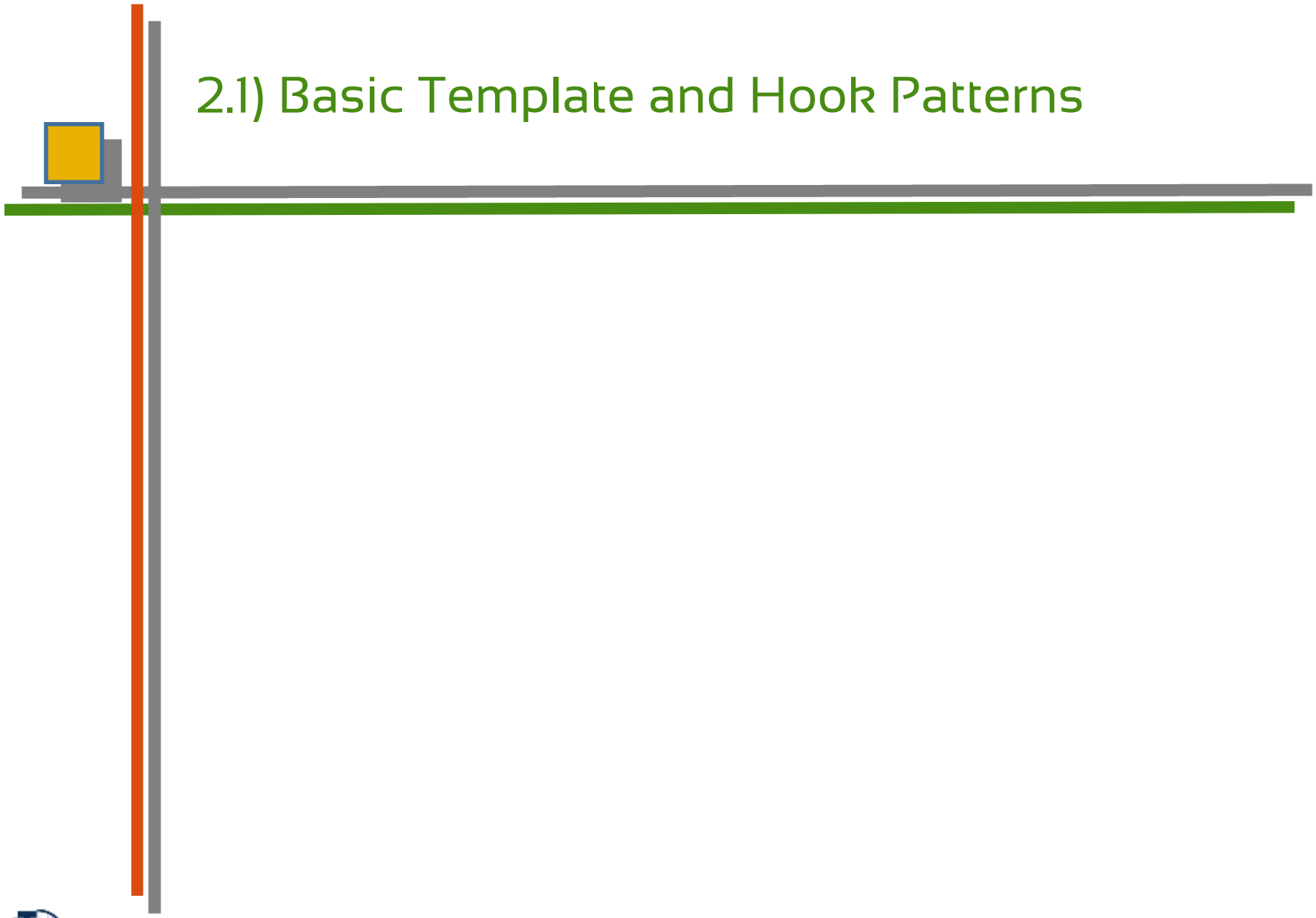
  German Translation
  Entwurfsmuster von Kopf bis Fuß. Eric Freeman & Elisabeth Freeman, mit Kathy Sierra & Bert Bates. O'Reilly, 2005, ISBN 978-3-89721-421-7

Secondary Literature

- D. Karlsson. **Metapatterns**. Seminar Design Patterns, IDA, Linköpings universitet, 2001.
- W. Pree. **Design Patterns for Object-Oriented Software Development**. Addison-Wesley, 1995. Unfortunately out of print.
- W. Zimmer. **Relationships Between Design Patterns**. Pattern Languages of Programming (PLOP) 1995.
- Uta Priss. **Faceted Information Representation**. Electronic Transactions in Artificial Intelligence (ETAI). 2000(4):21-33.
- R. Prieto-Diaz, P. Freeman. **Classifying Software for Reusability**. IEEE Software, Jan 1987. Prieto-Diaz has introduced facet-based classifications in software engineering. Surf also on citeseer for facets.
- Don Batory, Vivek Singhal, Jeff Thomas, Sankar Dasari, Bart Geraci, and Marty Sirkin. **The GenVoca Model of Software-System Generation**. IEEE Software, 11 (5), Sept. 1994, pp 89—94.

Goal

1. Understanding Templates and Hooks
   - Template Method vs Template Class
   - Dimensional Class Hierarchy
2. Understanding why Bridges implement faceted objects
3. Understanding layered systems



2.1) Basic Template and Hook Patterns

The Problem

The Problem

The Problem

The Problem

The Problem

The Problem

Variable Part
Can be exchanged

HOOK

Common Part
Stays fixed

TEMPLATE

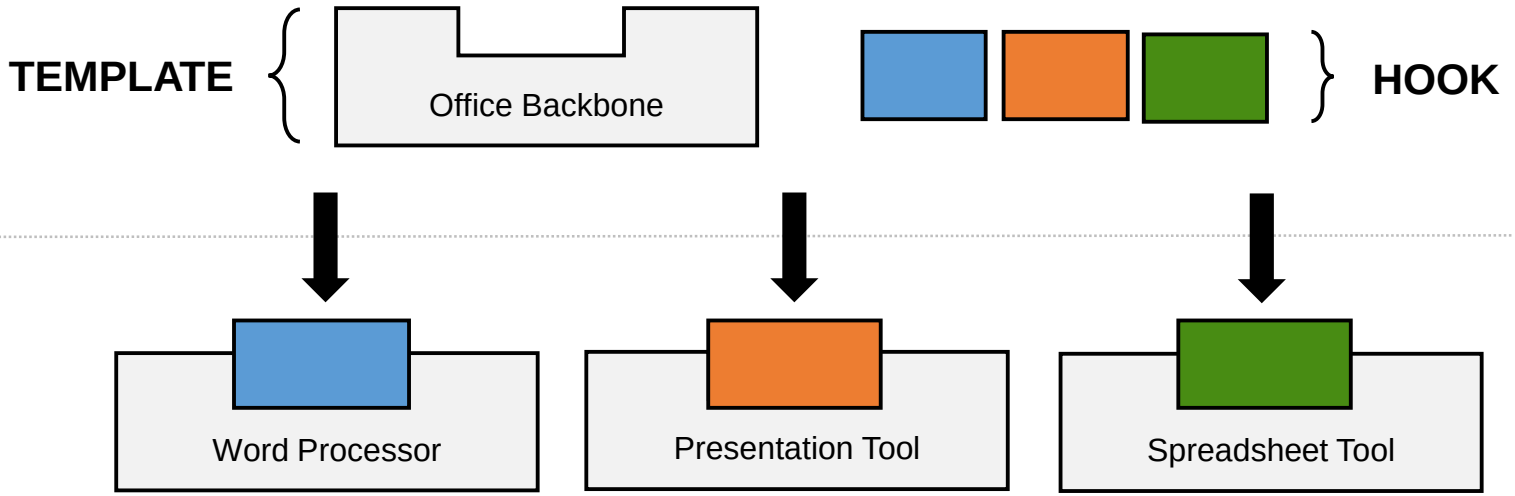
The Problem

- How to produce several products from one code base?
- Design patterns often center around
  - Things that are common to several applications

    Commonalities lead to *frameworks* or *product lines*

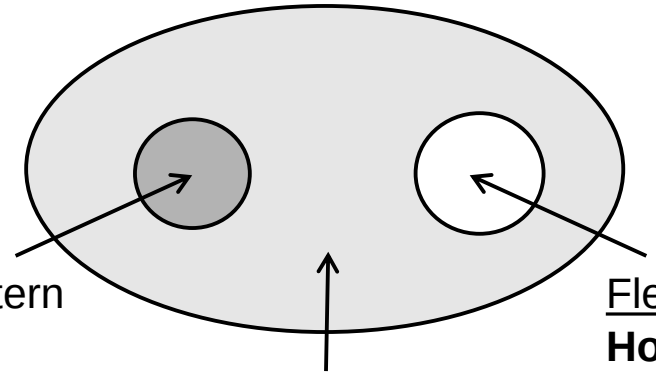
  - Things that are different from application to application

    Variabilities to *products* of a product line



Pree's Template&Hook Conceptual Pattern

- Pree invented a t*emplate-and-hook (T&H)* concept for the communality/variability knowledge in design patterns [Pree] [Karlsson]
- *Templates contain skeleton code*
  - Common for the entire product line
- *Hooks* are placeholders for the instance-specific code
  - Only for one product
  - Also called *slots, hotspots*

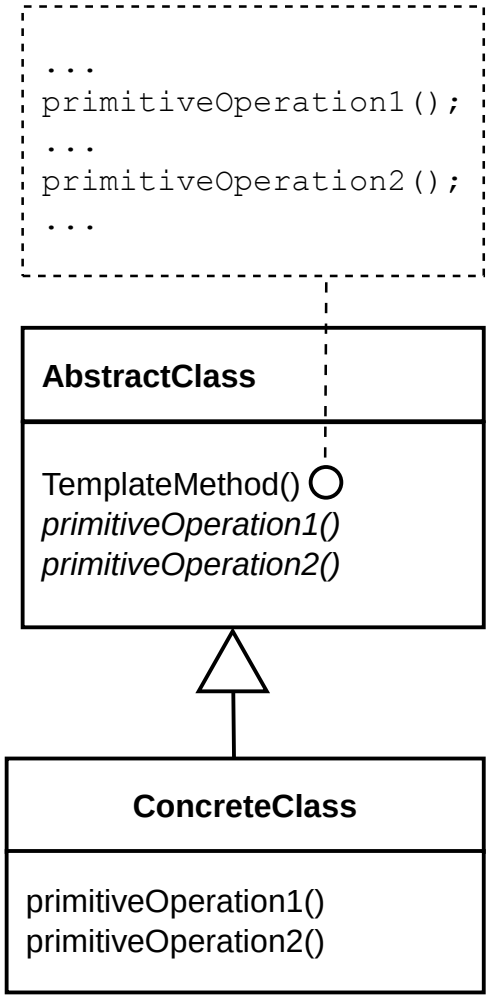


Fixed part of design pattern
Template: commonality

Flexible part of design pattern
Hook: variability

Context

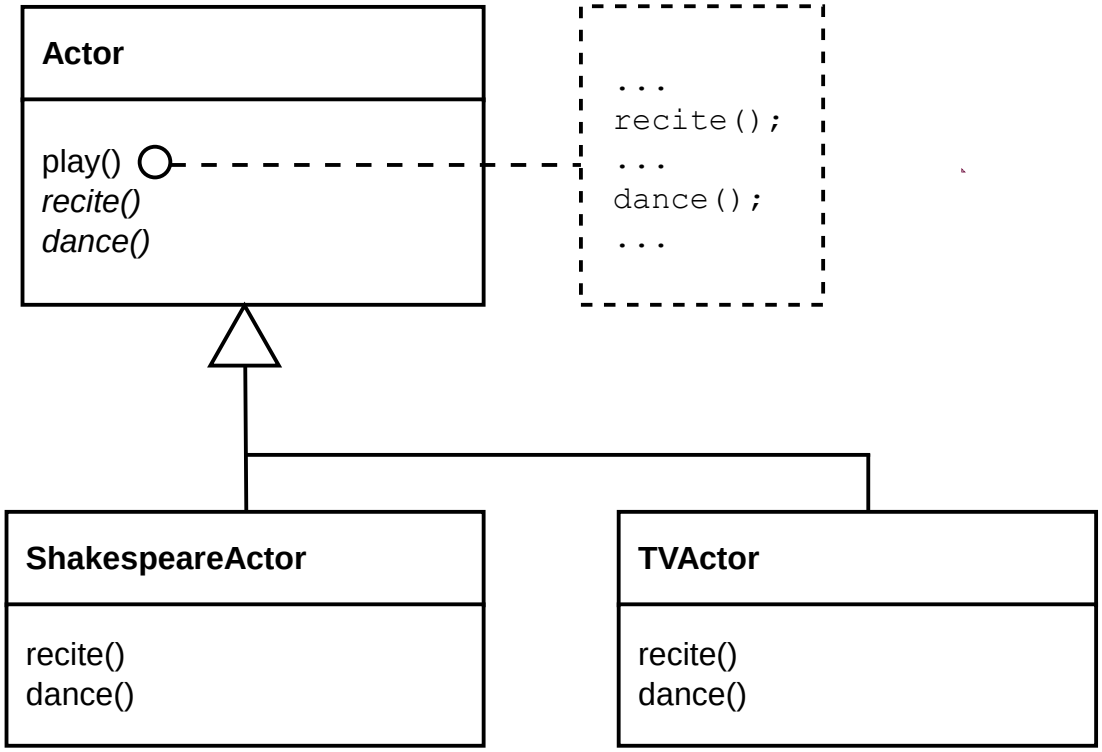
The Template Method Design Pattern



- Define the skeleton of an algorithm (*template method*)
  - The template method is concrete
- Delegate parts to abstract *hook methods (slot methods)* that are filled by subclasses
  - Requires inheritance
- Implements template and hook with the same class, but different methods

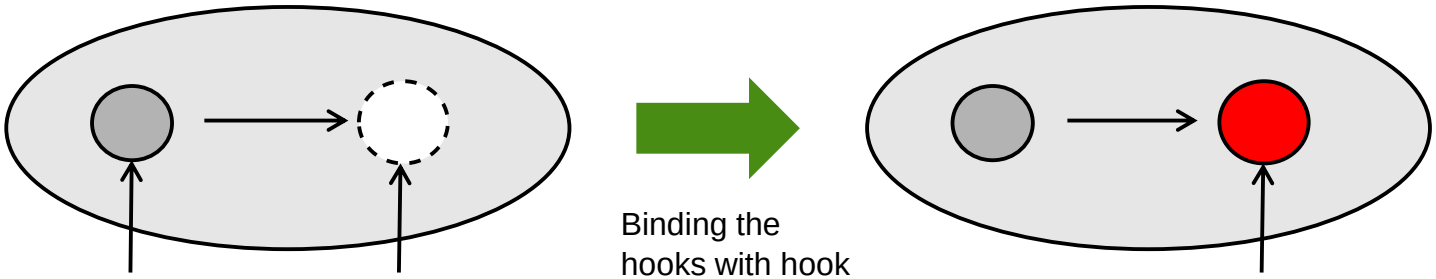
Template Method Example: Actors & Genres

- Binding an Actor's hook to be a ShakespeareActor or a TV actor



Variability with Template Method

- Allows for varying behavior
  - Separate invariant parts from variant parts of an algorithm
  - TemplateMethod differs slighly from polymorphism:
    → For a polymorphic method, one needs several subclasses
- Binding the hook (slot) means to
  - Derive a concrete subclass from the abstract superclass, providing the implementation of the hook method
- Controlled extension by only allowing for binding hook methods, but not overriding template methods



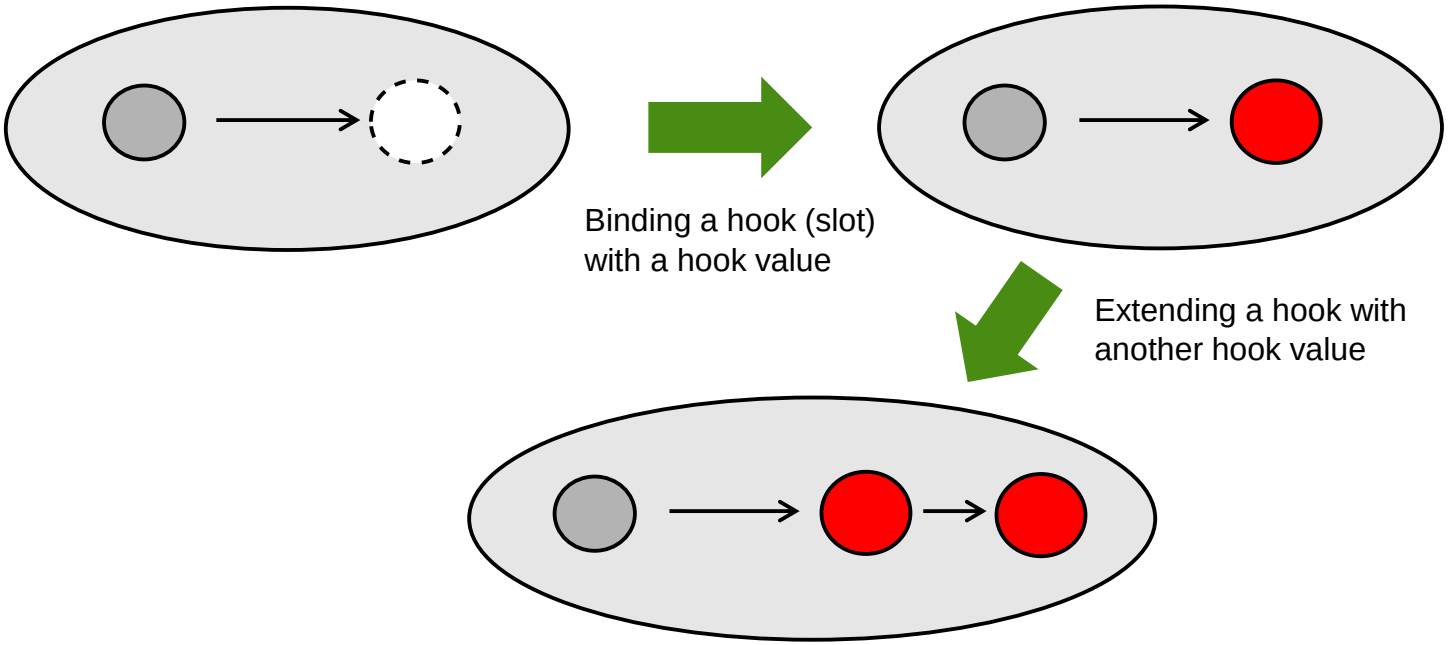
Consequences

- The design pattern **TemplateMethod** realizes the conceptual pattern **T&H** on the level of Methods
  - TemplateMethod – HookMethod

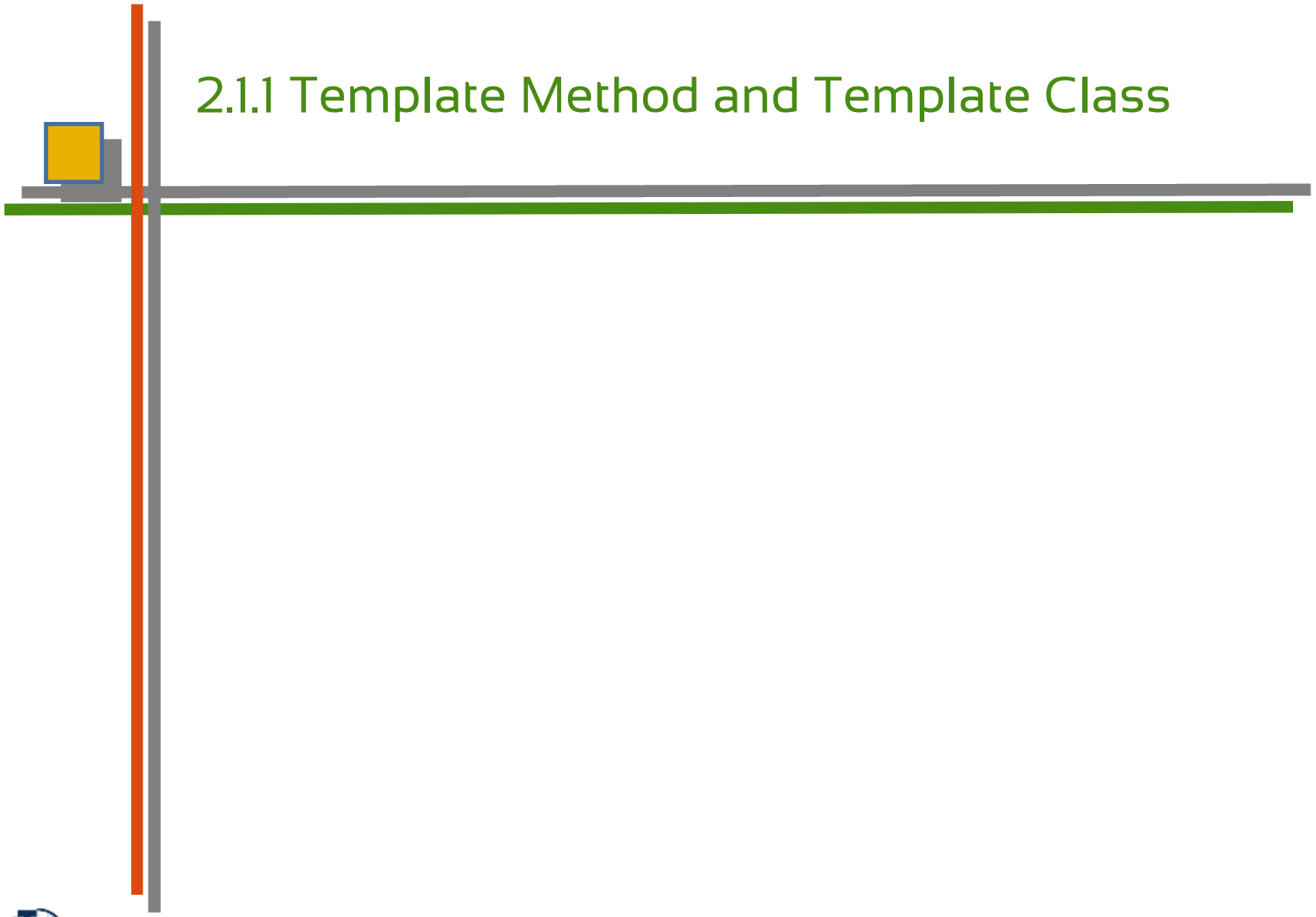
- Basis for other design patterns:
  - FactoryMethod
  - TemplateClass

Variability vs Extension

- The T&H concept occurs in two basic variants
  - Binding of hooks or extension of hooks:
    → a *slot*, is a hook that can be bound only once (unextensible hook, only bindable)
  - Hooks can be extensible
  - Extension patterns are treated later



2.1.1 Template Method and Template Class

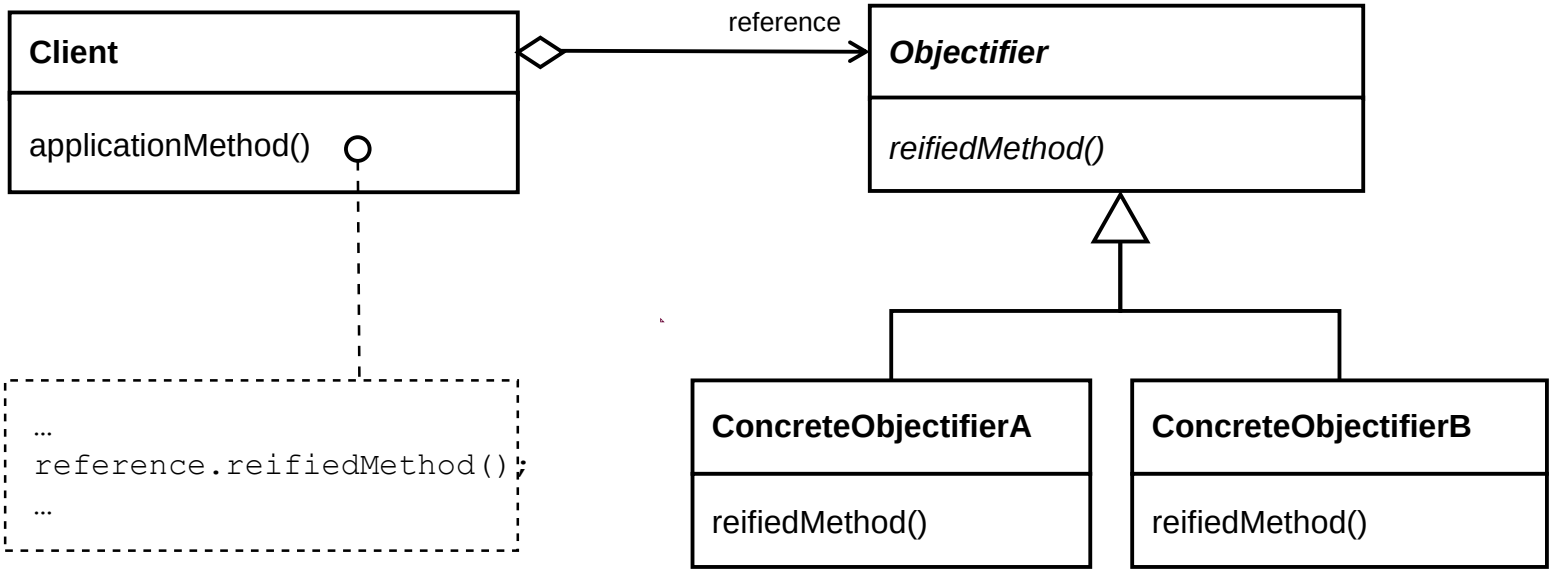


What Happens If We Reify the Hook Method?

- Methods can be reified,
  → represented as objects
- In the TemplateMethod, the hook method can be split out of the class and put into a separate object
- Reification is done by another basic pattern, **the Objectifier** [Zimmer]

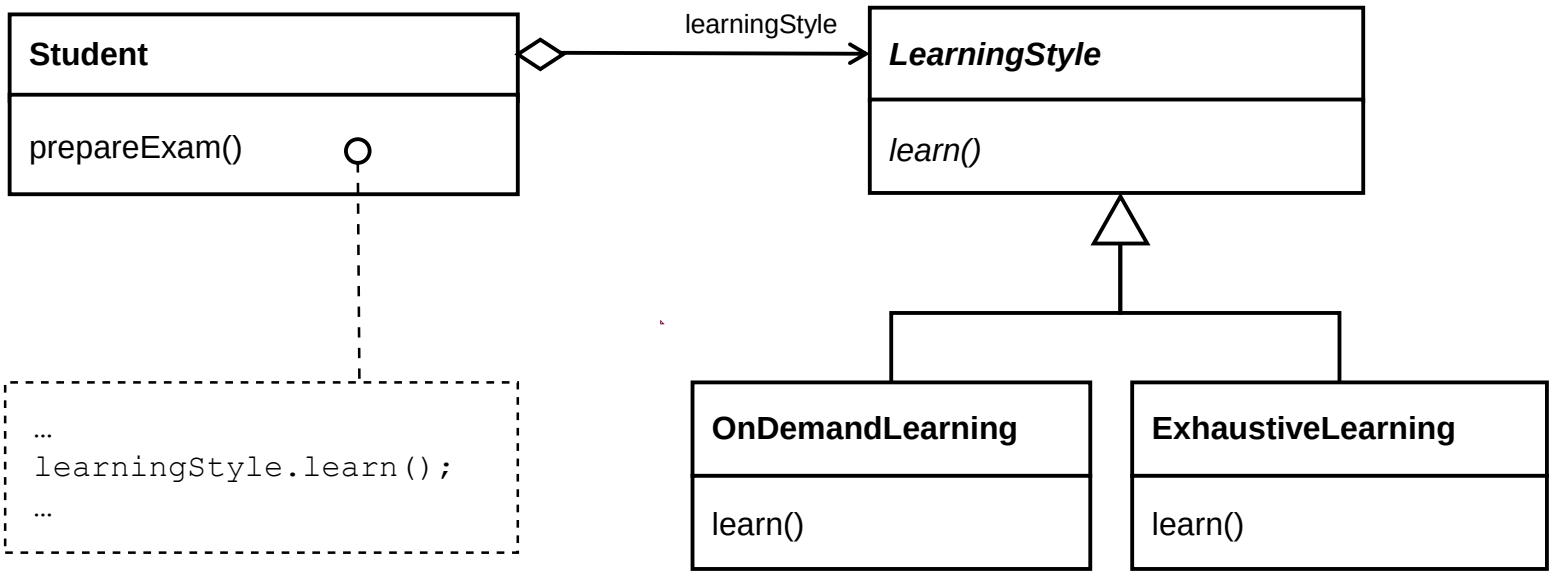
The Objectifier Pattern

- The pattern talks about **basic polymorphism** with objects (delegation)
  - Combined with an *abstract class* and *abstract method*
  - Clients call objects polymorphically



Objectifier Example: Different Students

- When preparing an exam, students may use different learning styles
- Instead of a method learn(),
  an objectified method (`LearningStyle` class) can be used



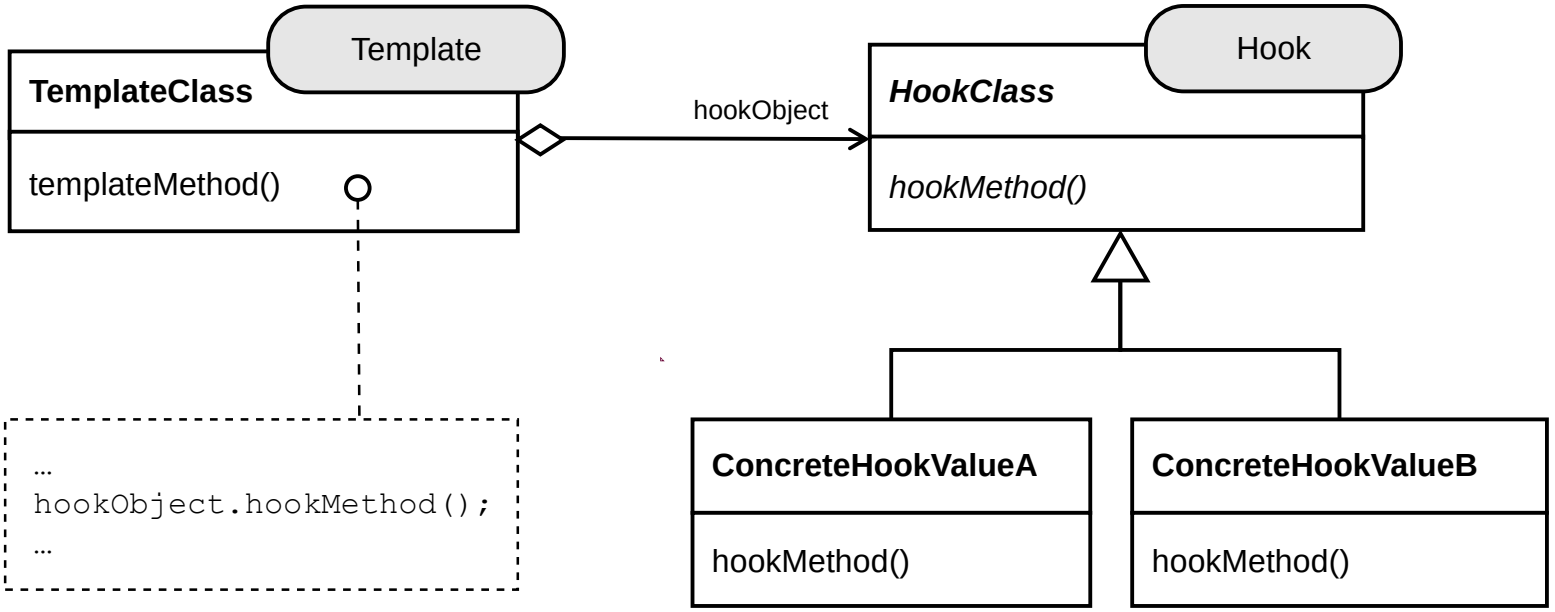
T&H on the Level of Classes

- With the Objectifier, we can build now Template&Hook classes
  - Additional roles for some classes
    - The template role
    - The hook role

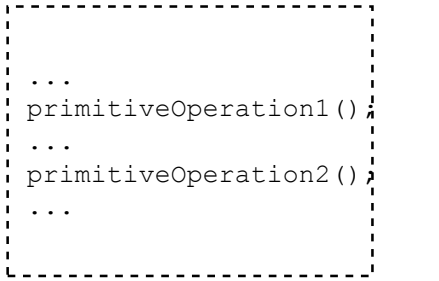
- Resulting patterns:
  - Template Class
  - Generic Template Class
  - Dimensional Class Hierarchies for variability with parallel class hierarchies
    - Implementation of facets
    - Bridge, Visitor

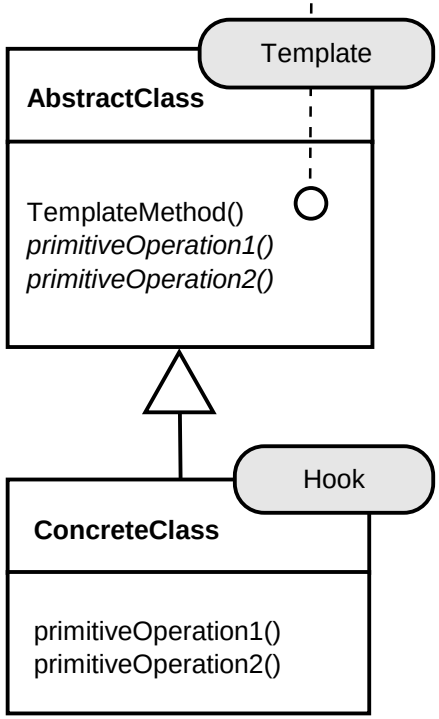
Template Class

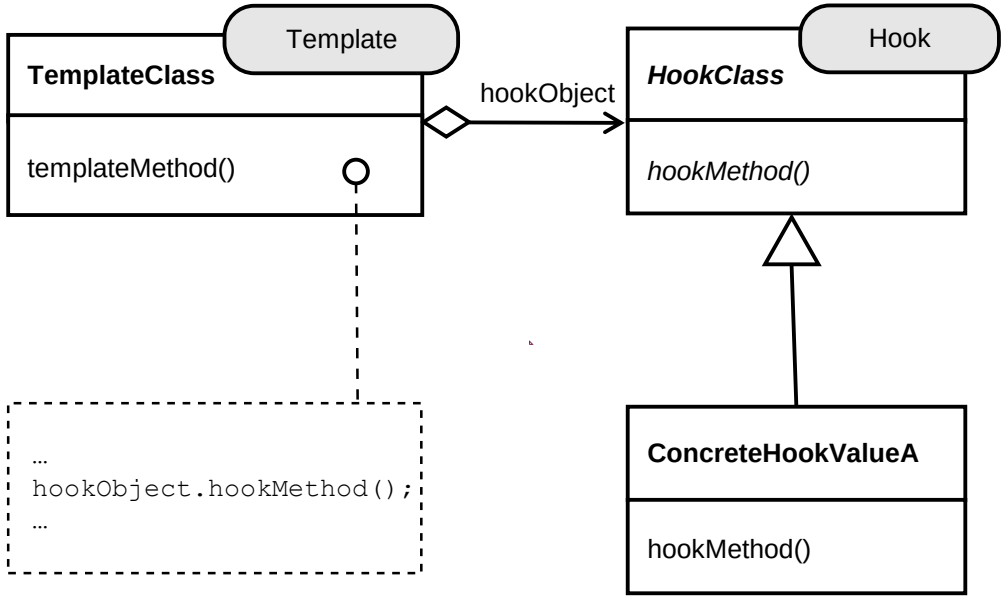
- Is combined from **TemplateMethod** + **Objectifier**
  - Explicitly fix a template class in the Objectifier
  - Template method and hook method are found in different classes



Template Method vs Template Class







Template Class

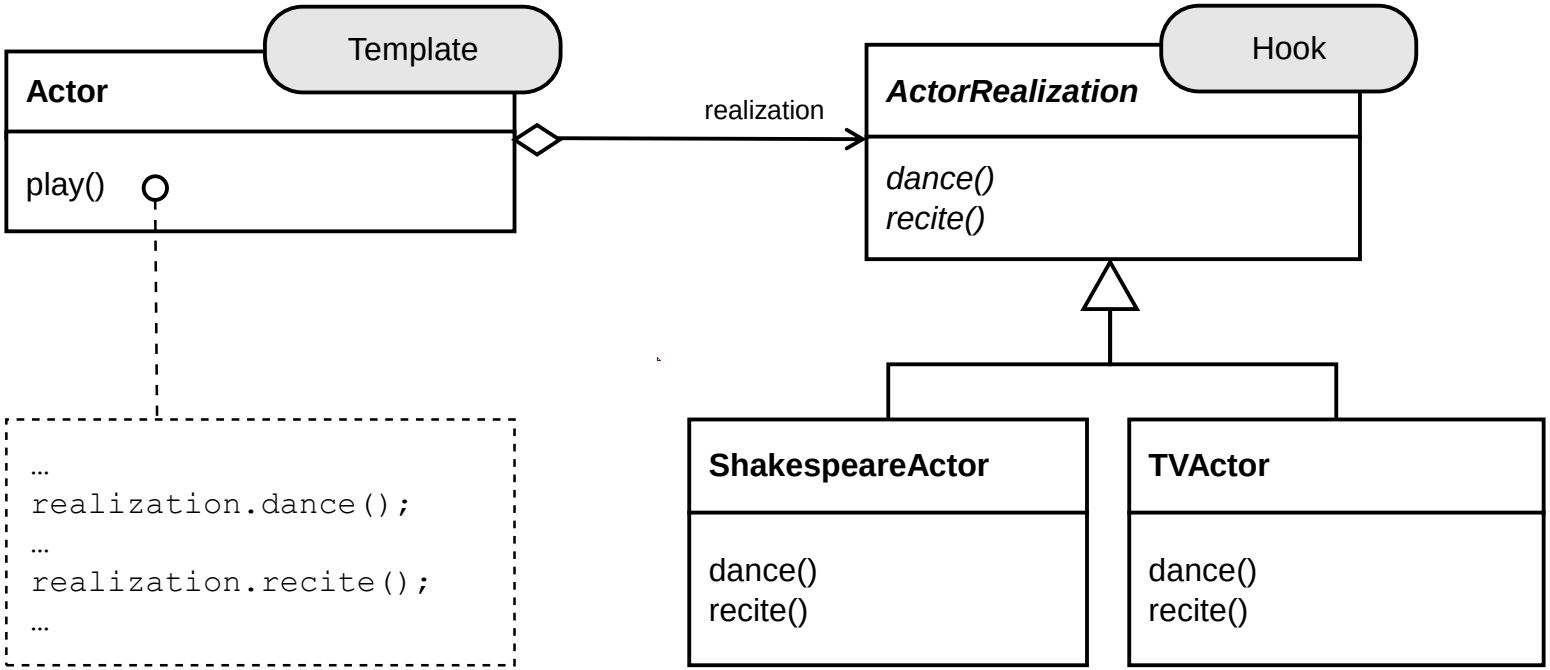
- Similar to TemplateMethod, but
  - Hook objects can be exchanged at run time
  - Exchanging a set of methods at the same time
  - Consistent exchange of several parts of an algorithm (not only one method)

Template Class

- This pattern is basis of
  - Bridge
  - Builder
  - Command
  - Iterator
  - Observer
  - Prototype
  - State
  - Strategy
  - Visitor

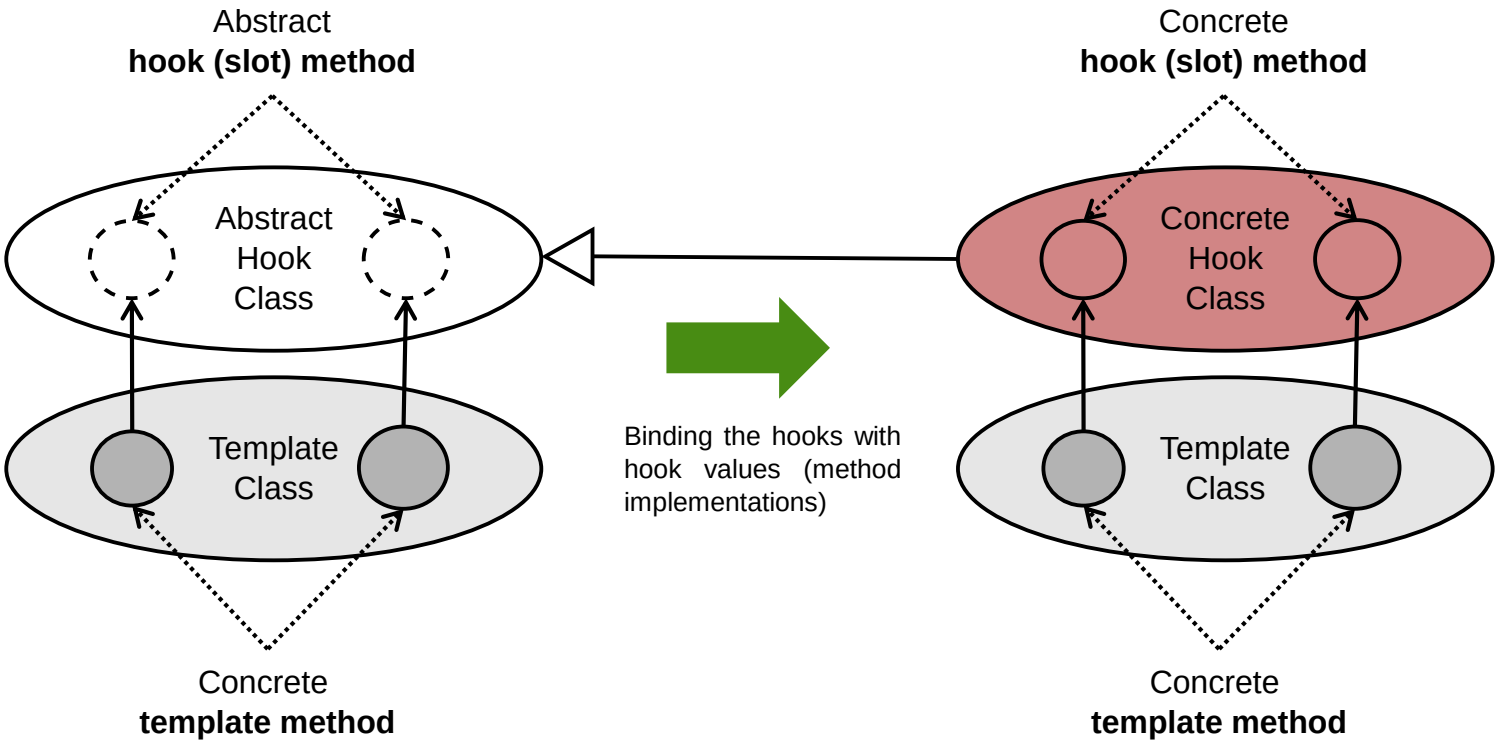
Actors and Genres as Template Class

- Consistent exchange of recitation and dance behavior possible



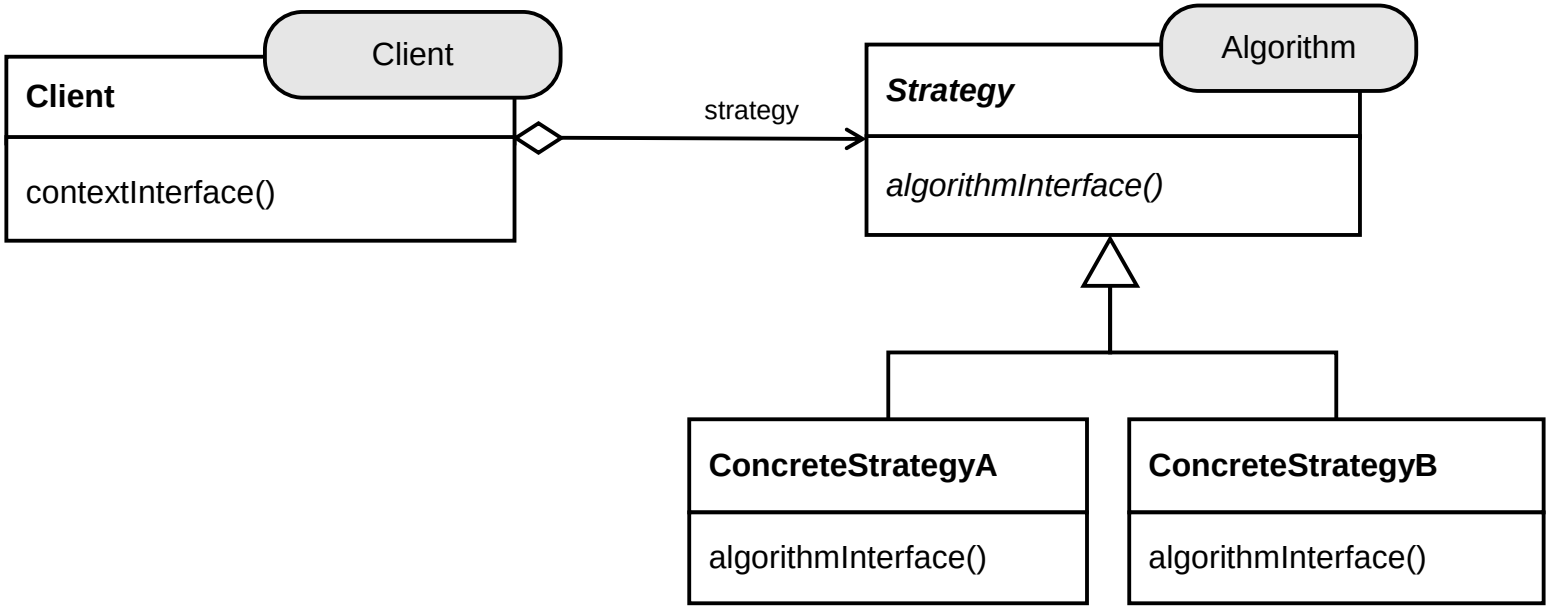
Variability with TemplateClass

- Binding the hook means to
  - Concrete Class provides the implementation of the hook method
  - Derive concrete subclass from the **abstract hook superclass**



The GOF-Pattern Strategy

- Variant of Template Class
- Hands out the roles *client and algorithm*



GOF-Strategy is related to Template Class

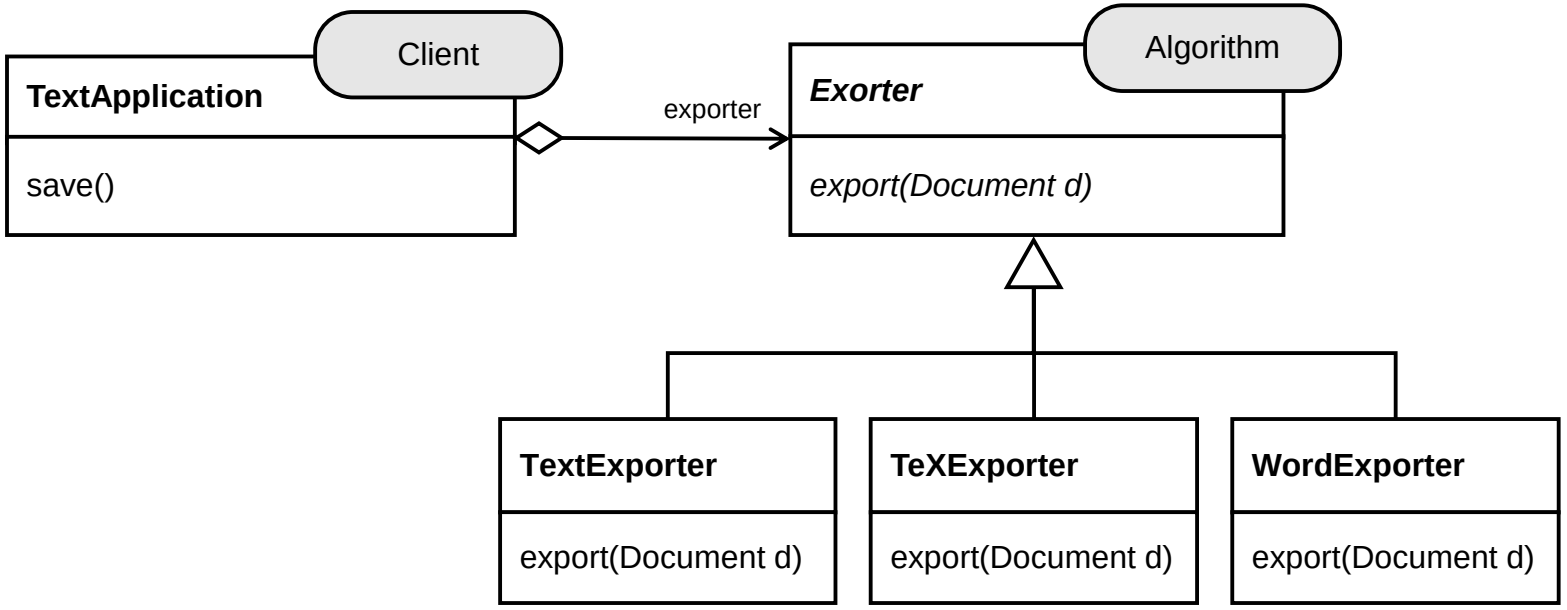
- GOF Strategy has the same structure as Objectifier
  - **Different incentive !**
  - It is not for reifying methods, but for varying methods only
- TemplateClass also has a different incentive
  - Hence, TemplateClass hands out other roles for the classes
  - Client class is the template class
  - Strategy class is the hook class

Design Patterns may have the same structure and/or behavior,

but can have a different incentive

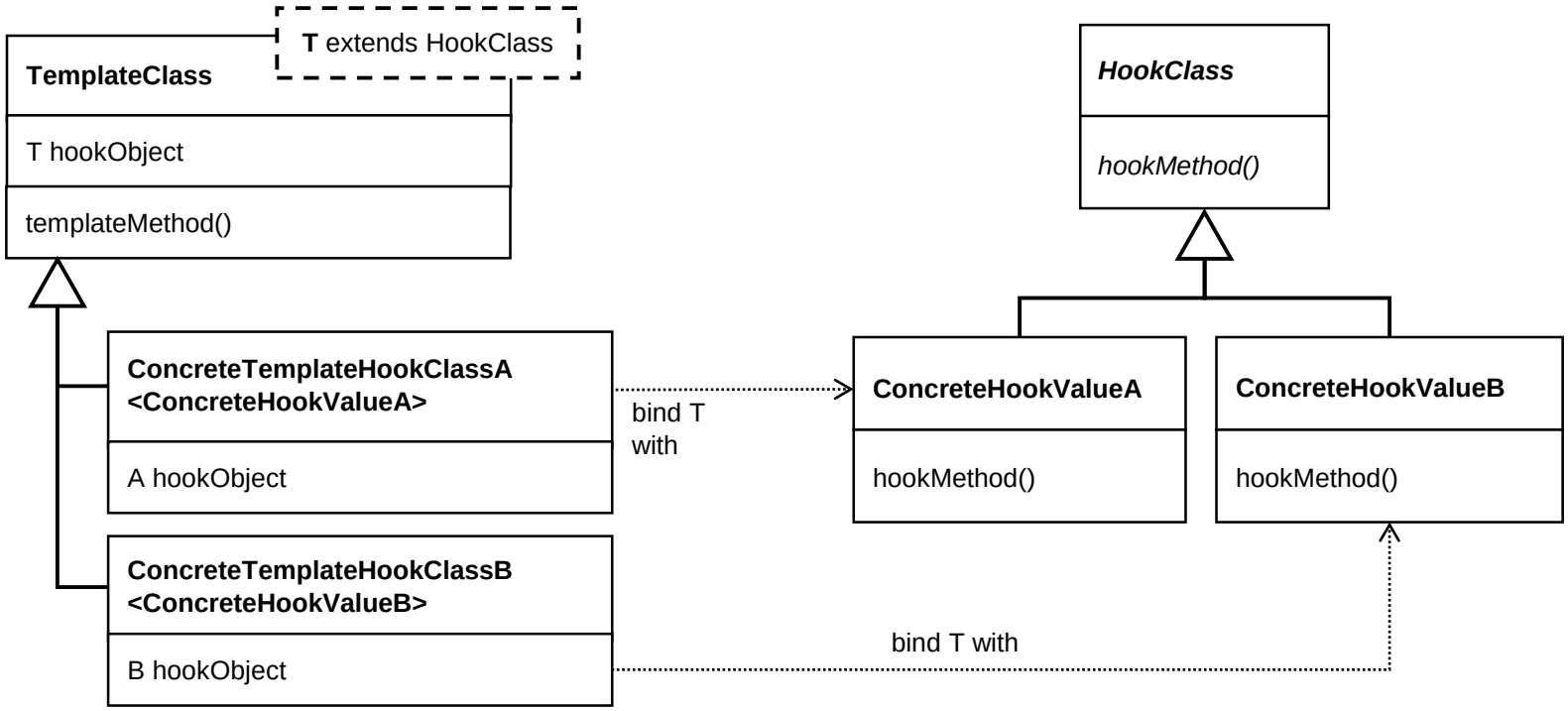
Example for Strategy

- Encapsulate formatting algorithms

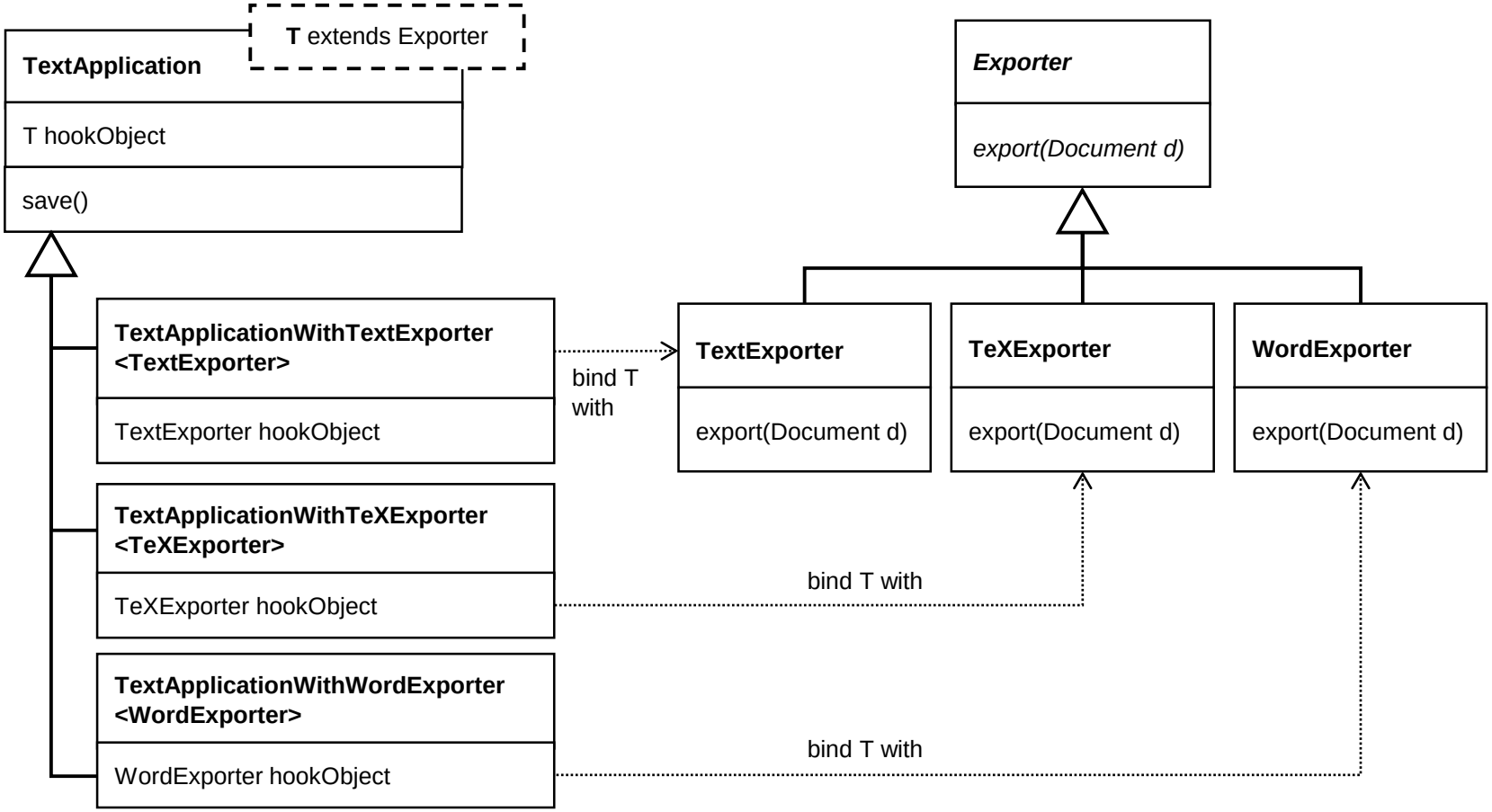


Generic Template Class

- TemplateClass can be realized with GenericTemplateClass
  - In languages with generic classes (C++, Java 1.5, C#, Sather, Cecil, Eiffel)
- The subclassing of the hook hierarchy is replaced by static generic expansion
  - Hence, more type safety, less runtime dispatch



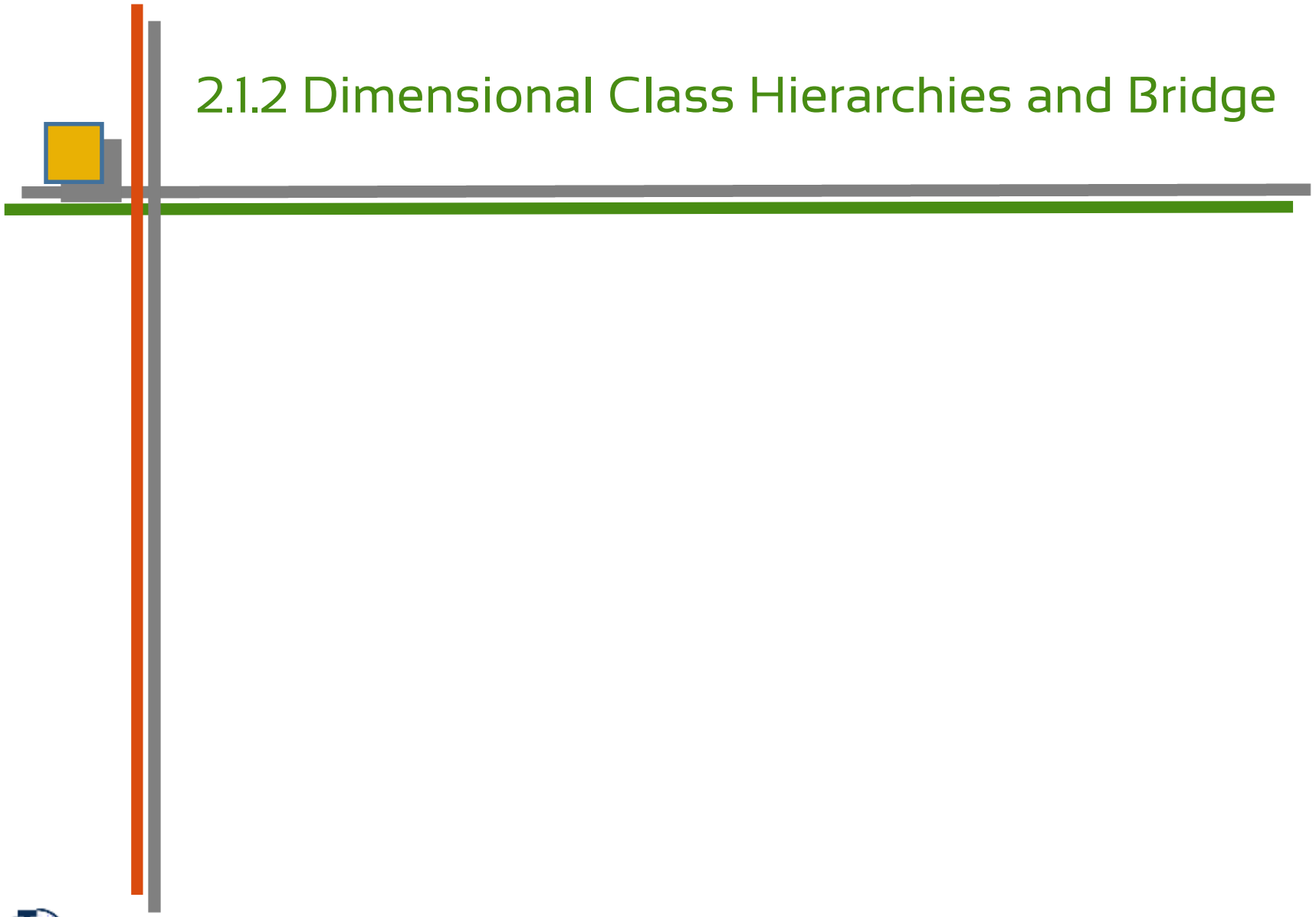
Generic Text Exporter



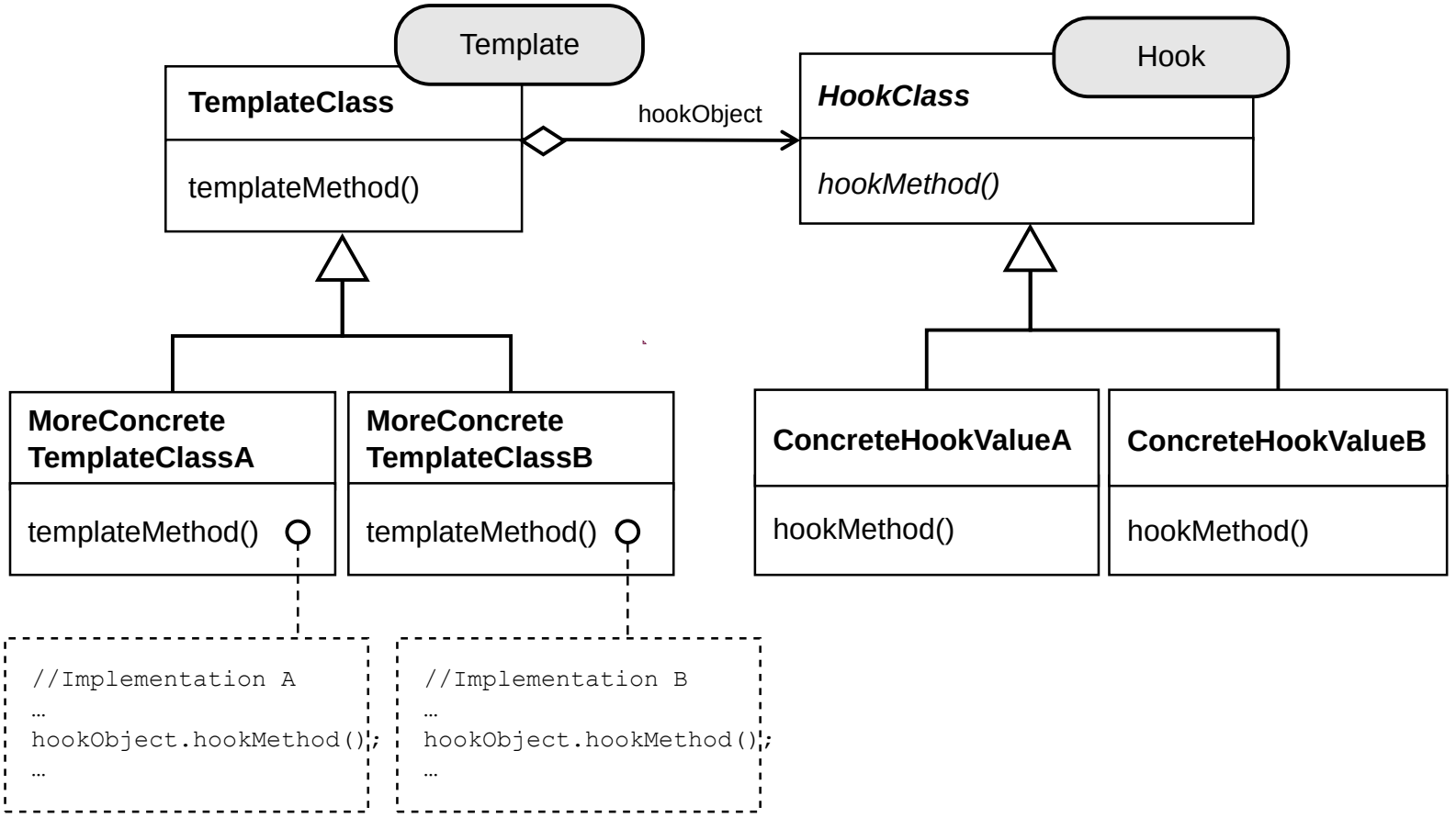
Further Work on Generic Template Parameterization

- See course CBSE
- GenVoca [Batory]
  - Generic template instantiation method for nested generics
  - Parameterization on many levels
  - Layered systems result
  - Realizable with nested C++ templates
  - See later
- Template Metaprogramming (www.boost.org)
  - Using template parameter for other purposes than hook classes

2.1.2 Dimensional Class Hierarchies and Bridge



Variability Pattern Dimensional Class Hierarchies



Dimensional Class Hierarchies

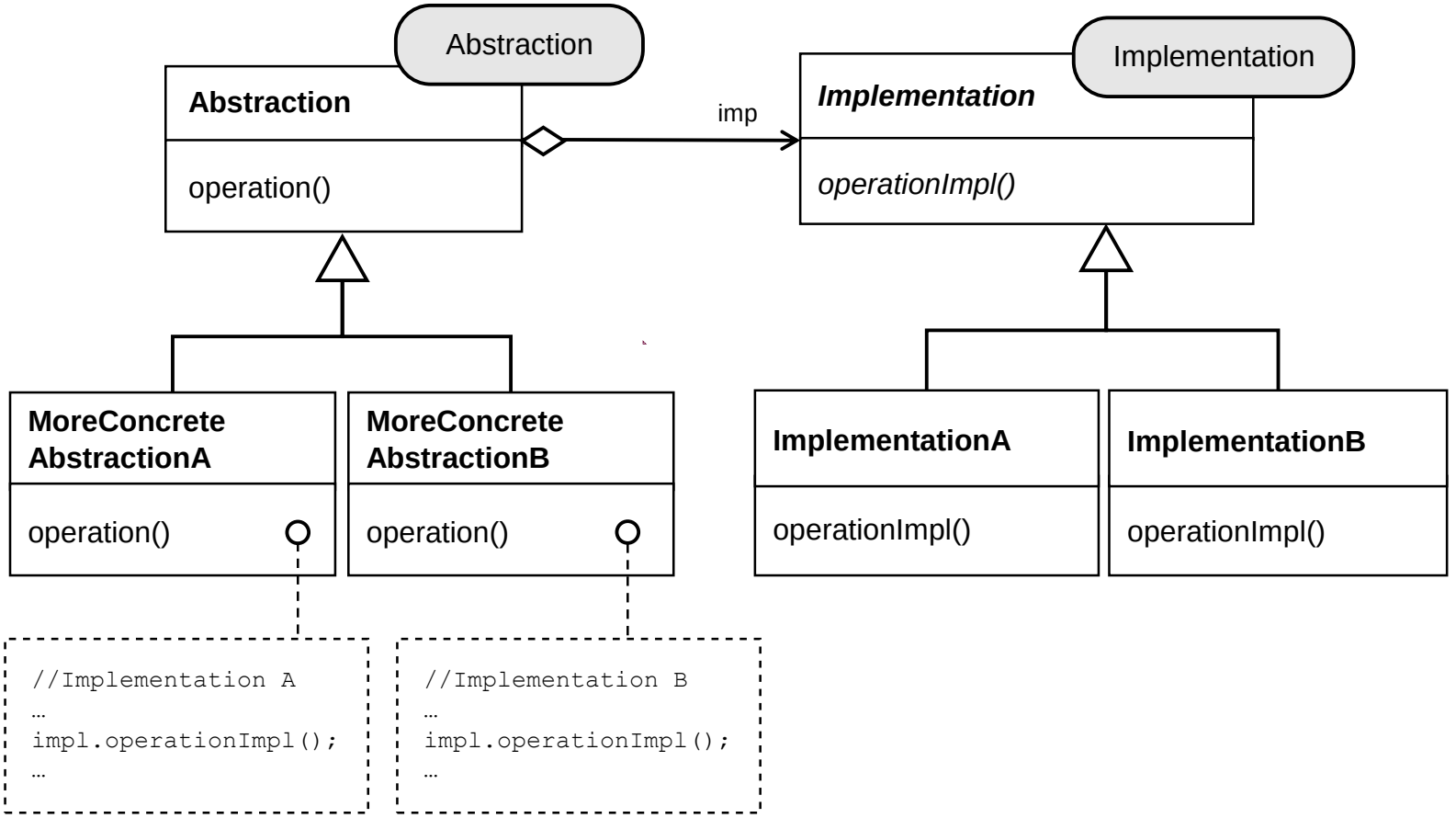
- Vary also the template class in a class hierarchy
  - The sub-template classes can change the template algorithm
  - Template method still calls the hook methods
  - ***Important***: sub-template classes must fulfil the *contract* of the superclass
    - Implementation can be changed,
    - Interface and visible behavior must be the same
- Upper and lower layer (dimension)
  - Template method (upper layer) calls hook methods (lower layer)
- Both hierarchies can be varied independently
  - Factoring (orthogonalization)
  - Reuse is increased
- Basis for other patterns (Bridge, Visitor, …)
- Basis for implementation of facets

Bridge Pattern

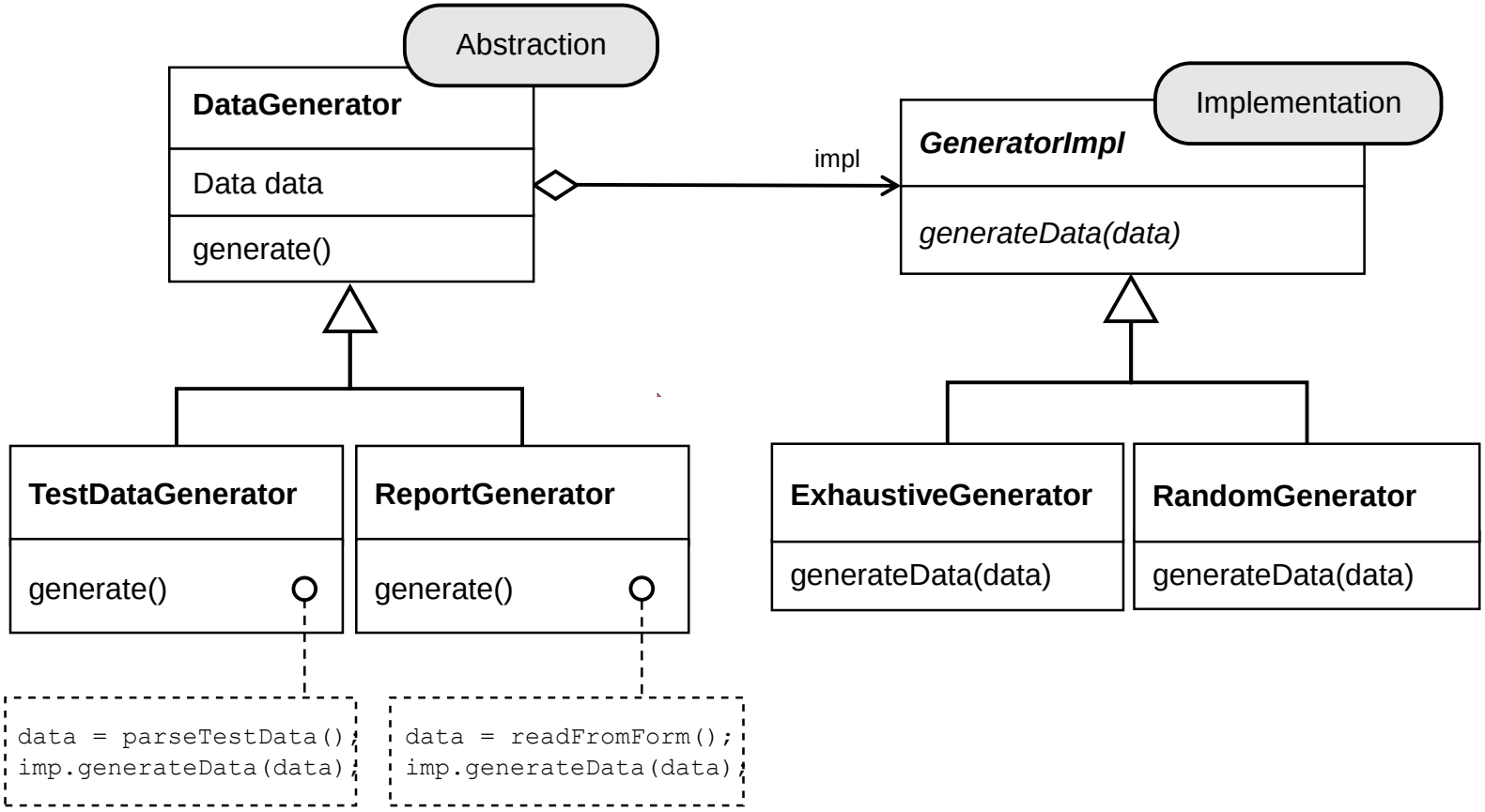
- The Bridge pattern is a variant of Dimensional Class Hierarchies (different incentive)
- The left hierarchy is called *abstraction hierarchy*,
  the right *implementation*
  - Also *handle* vs *body*
- Separation of two hierarchies

Bridge

- Different incentive (Abstraction/Implementation)



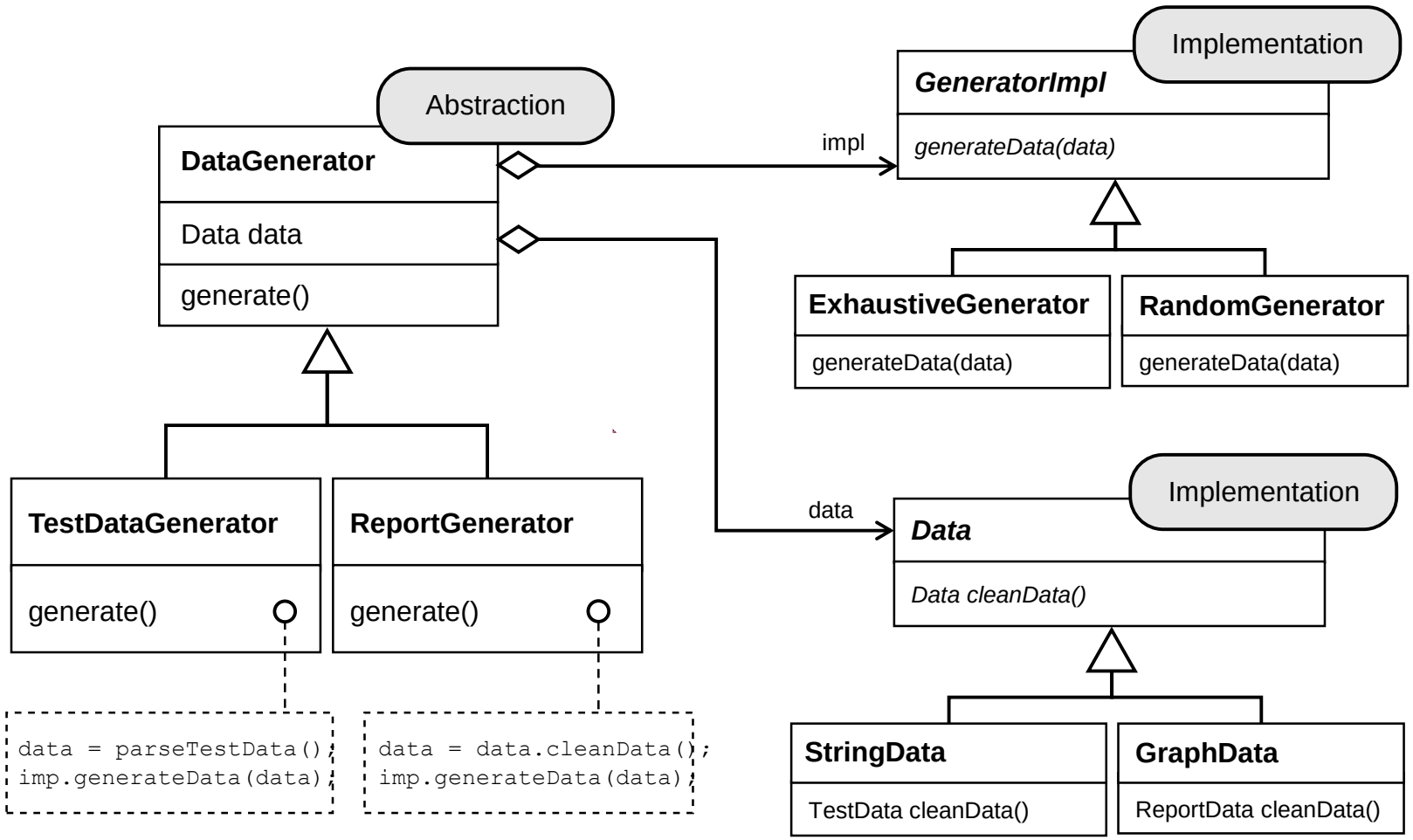
Example: DataGenerator as Bridge



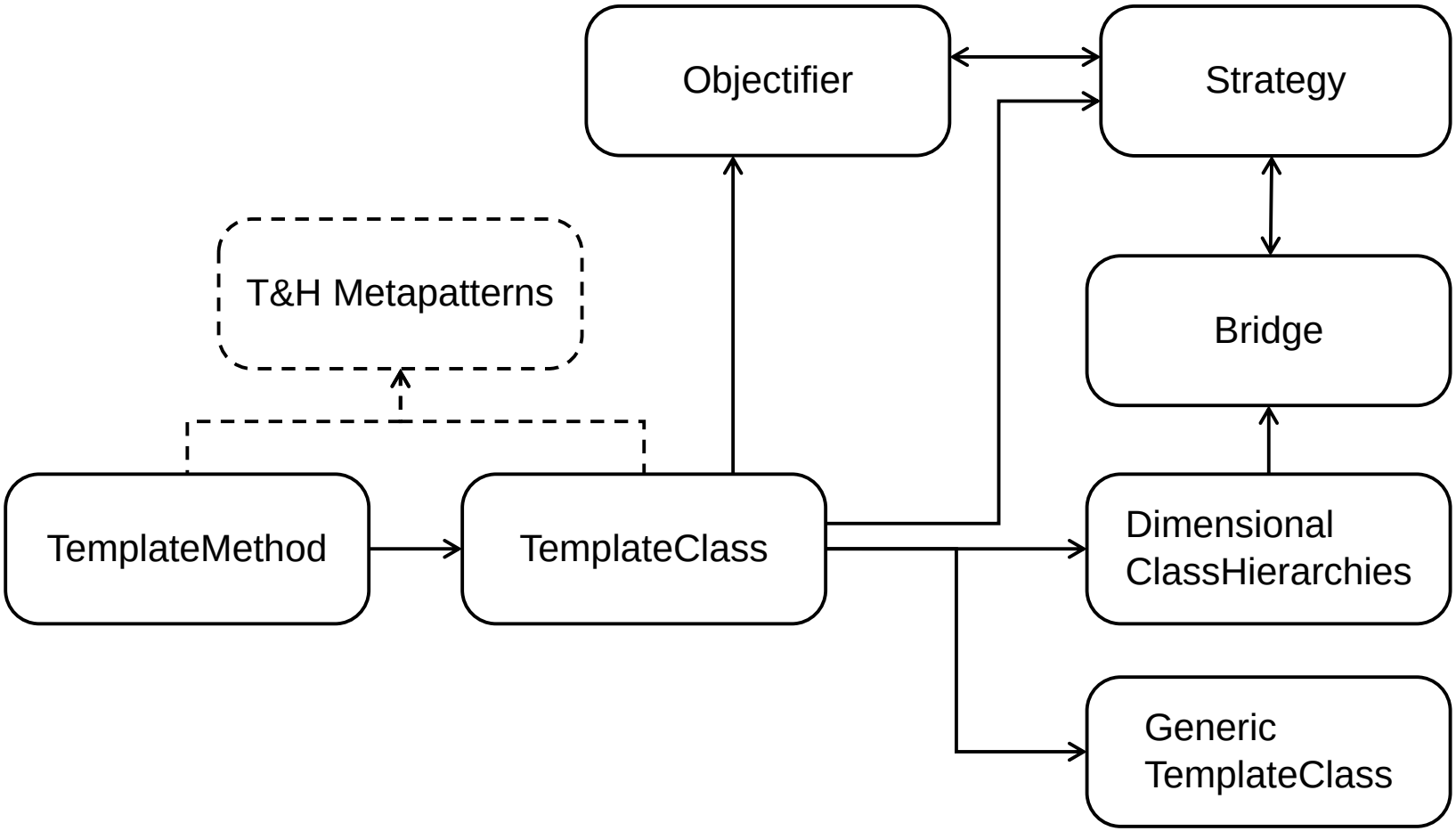
Bridge

- Both hierarchies can be varied independently
  - Factoring (orthogonalization)
  - Reuse is increased
- An abstraction can have several Bridges
  - Bridges can be replicated
  - Basis for implementation of *facets*

Example: DataGenerator as Bridge



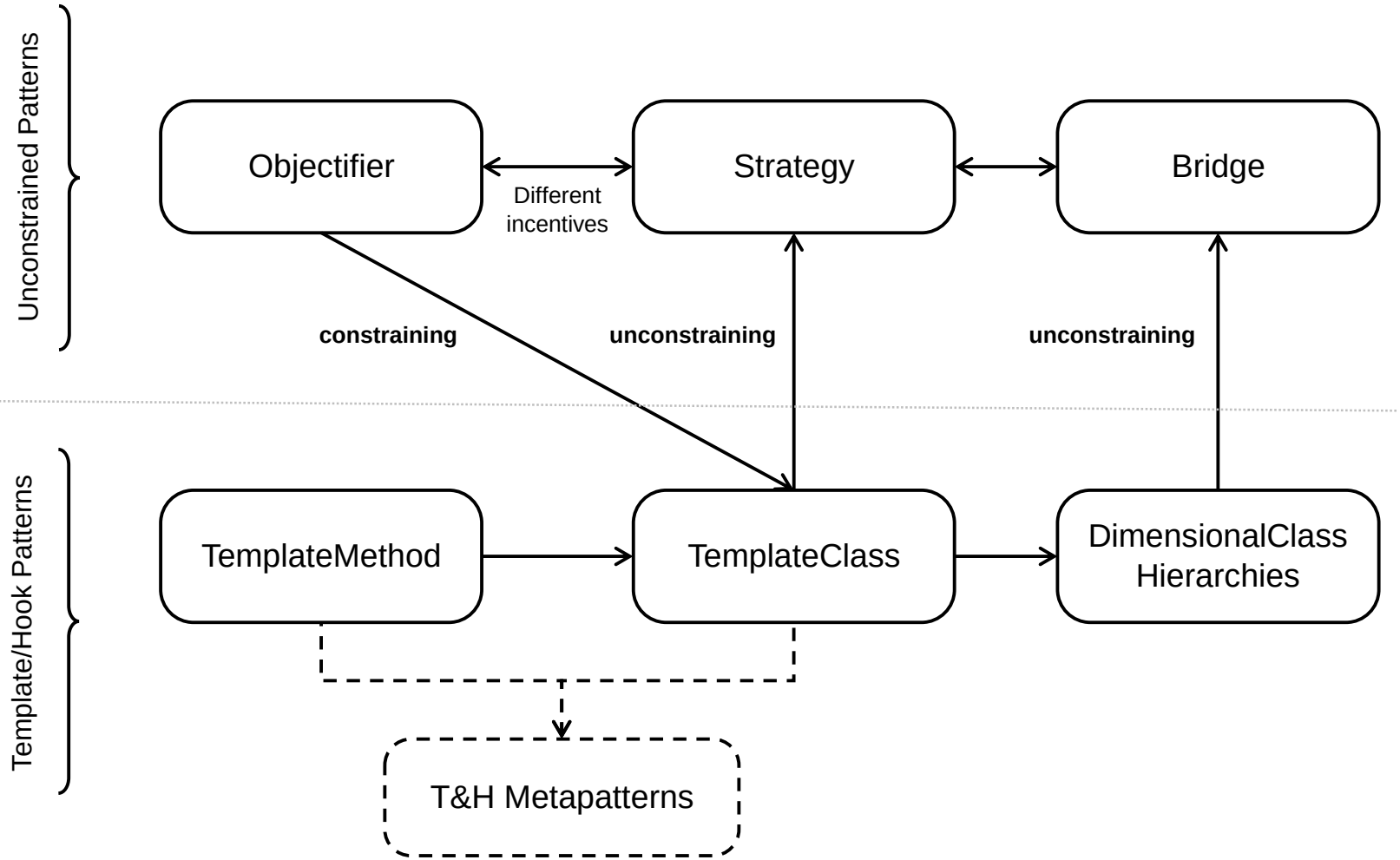
Basic Variability Patterns - Overview



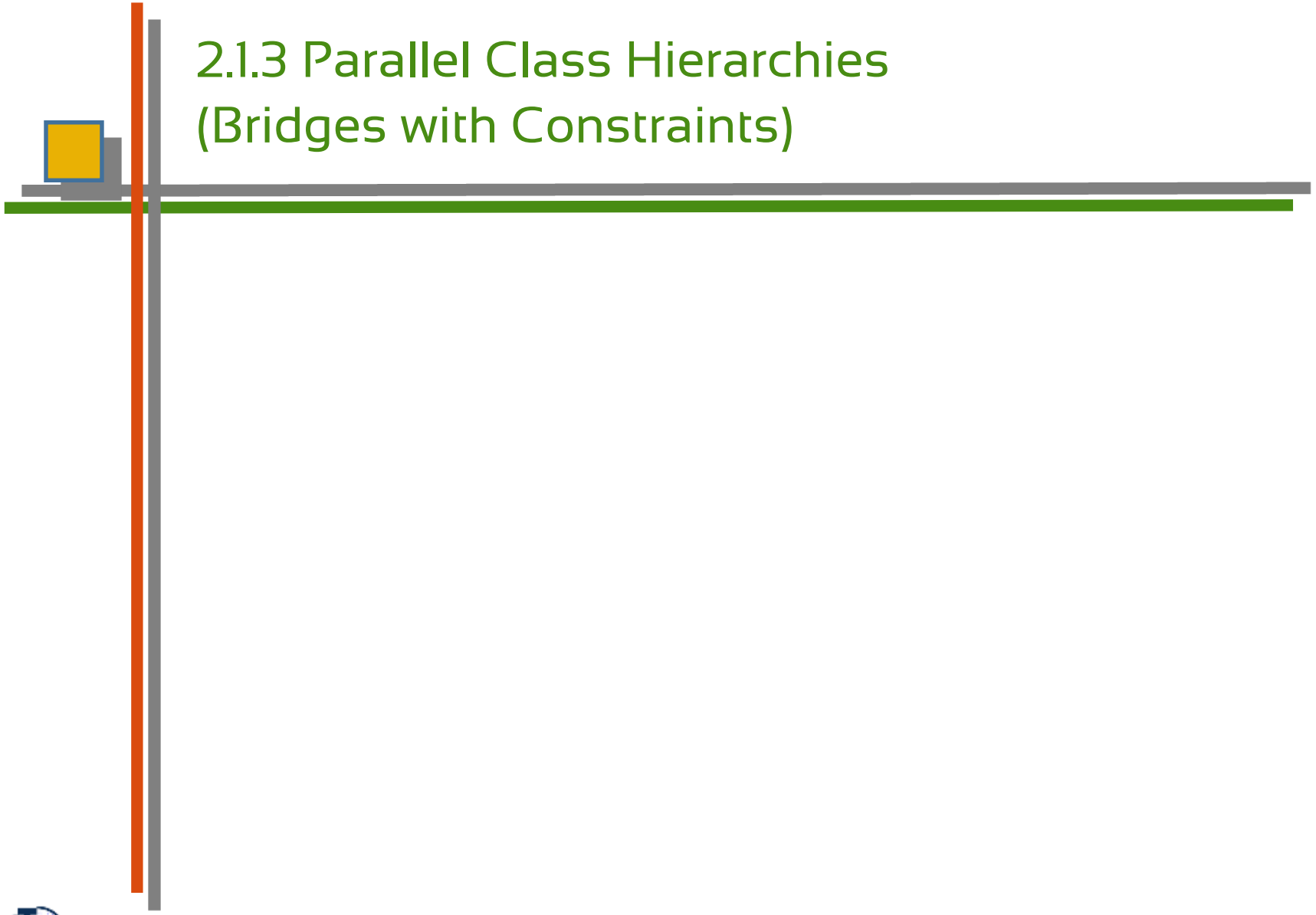
Relations of Basic Patterns

- Pree book vs Gamma book
  - Pree and the GOF worked together for some time,
    but then they published two different books
  - Pree's focus was on templates and hooks (framework patterns)
  - GOF on arbitrary patterns in arbitrary context
- One can take any GOF pattern and make it a framework pattern by introducing the template-and-hook constraint
  - Or if you take away the template-hook constraint from a framework pattern,
    you get an unconstrained general pattern

Relations


Unconstrained Patterns
Objectifier
Different incentives
Strategy
Bridge
constraining
unconstraining
unconstraining
Template/Hook Patterns
TemplateMethod
TemplateClass
DimensionalClass Hierarchies
T&H Metapatterns

2.1.3 Parallel Class Hierarchies (Bridges with Constraints)

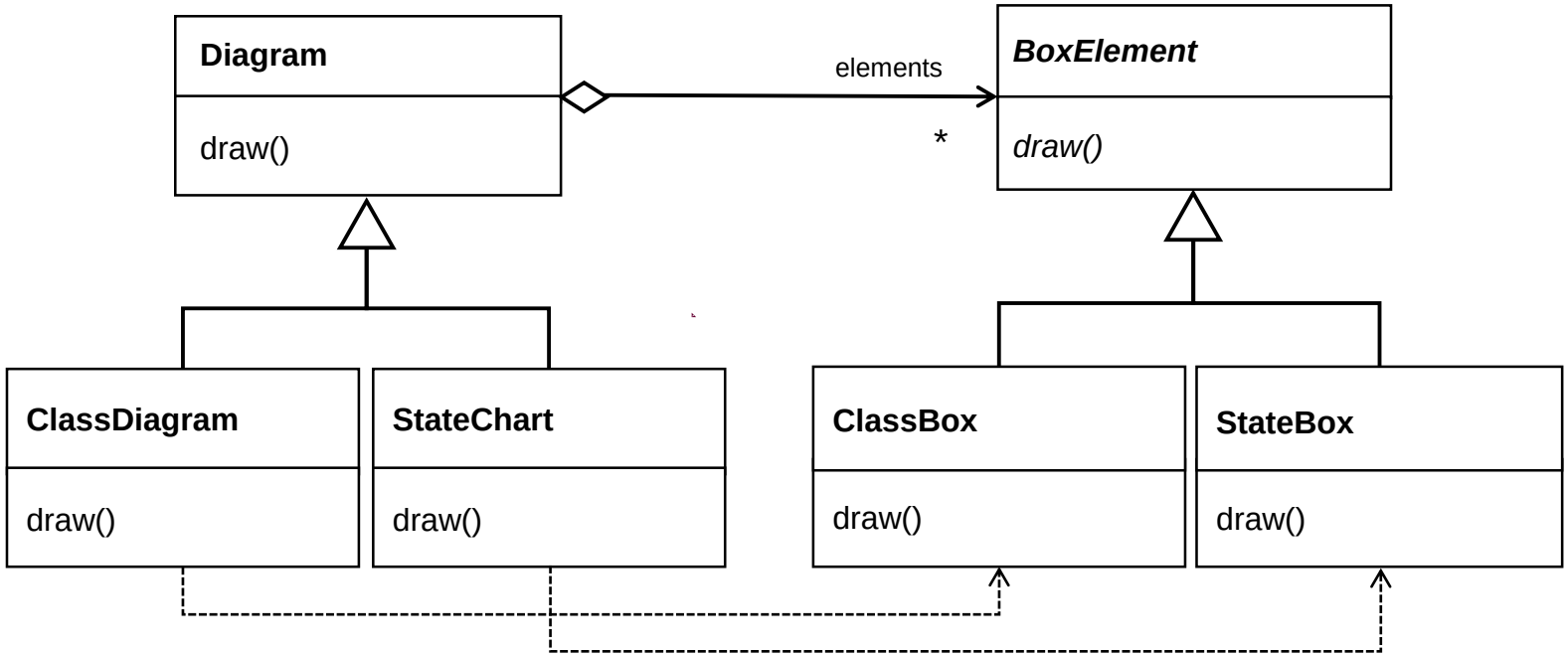


When the Dimensions cannot be Independently Varied

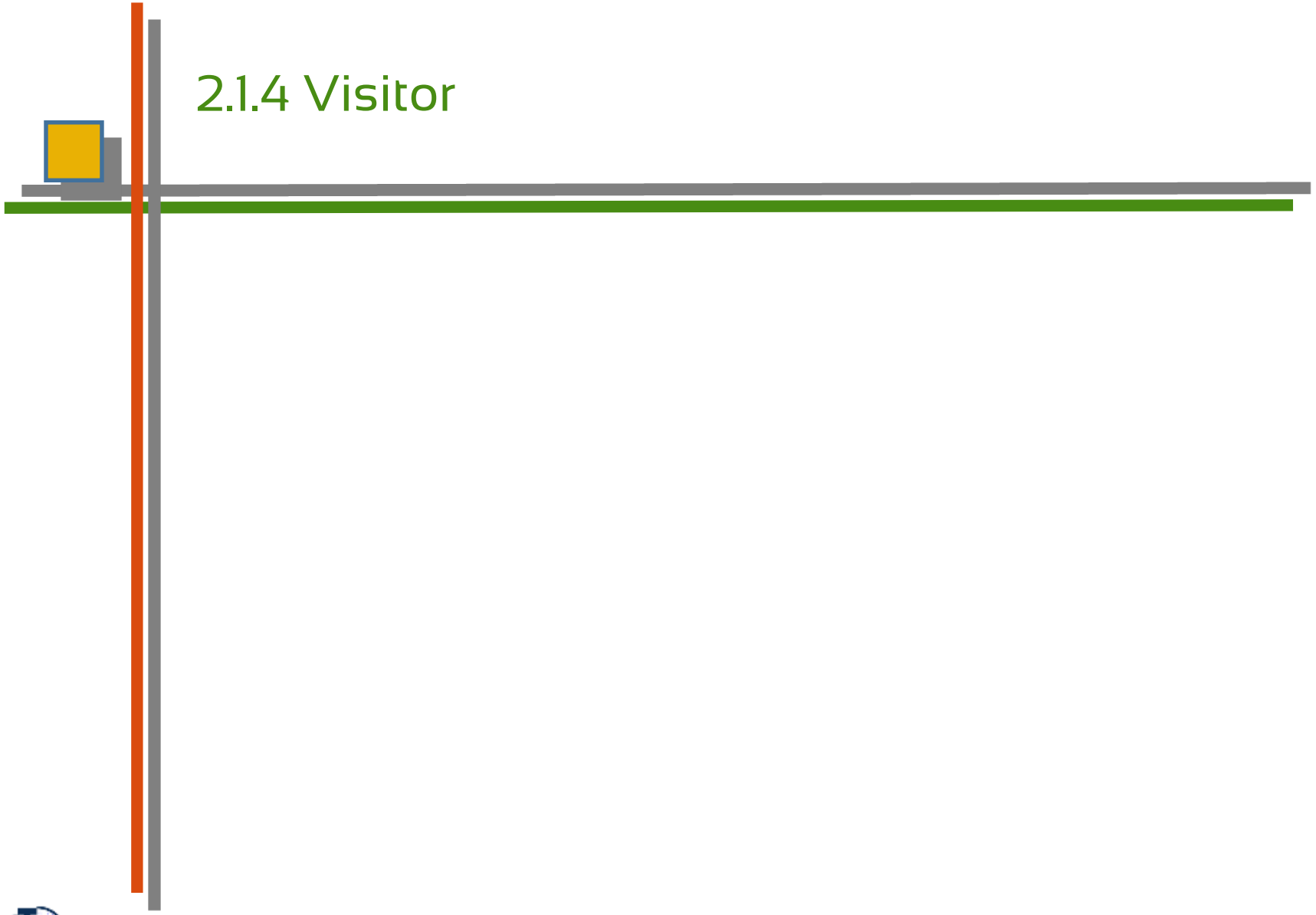
- Dimensions of Dimensional Class Hierarchies sometimes ***not independent***
  - If one is varied, another must also be varied
- Dimensions have equal size and structure, i.e., are isomorphic
- Typically Example: Container classes and their elements
  - UML diagrams and their node and edge types
  - Figures and their figure elements
  - Record lists and their record types

Parallel Hierarchies with Parallelism Constraint

- Both hierarchies, must be varied consistently



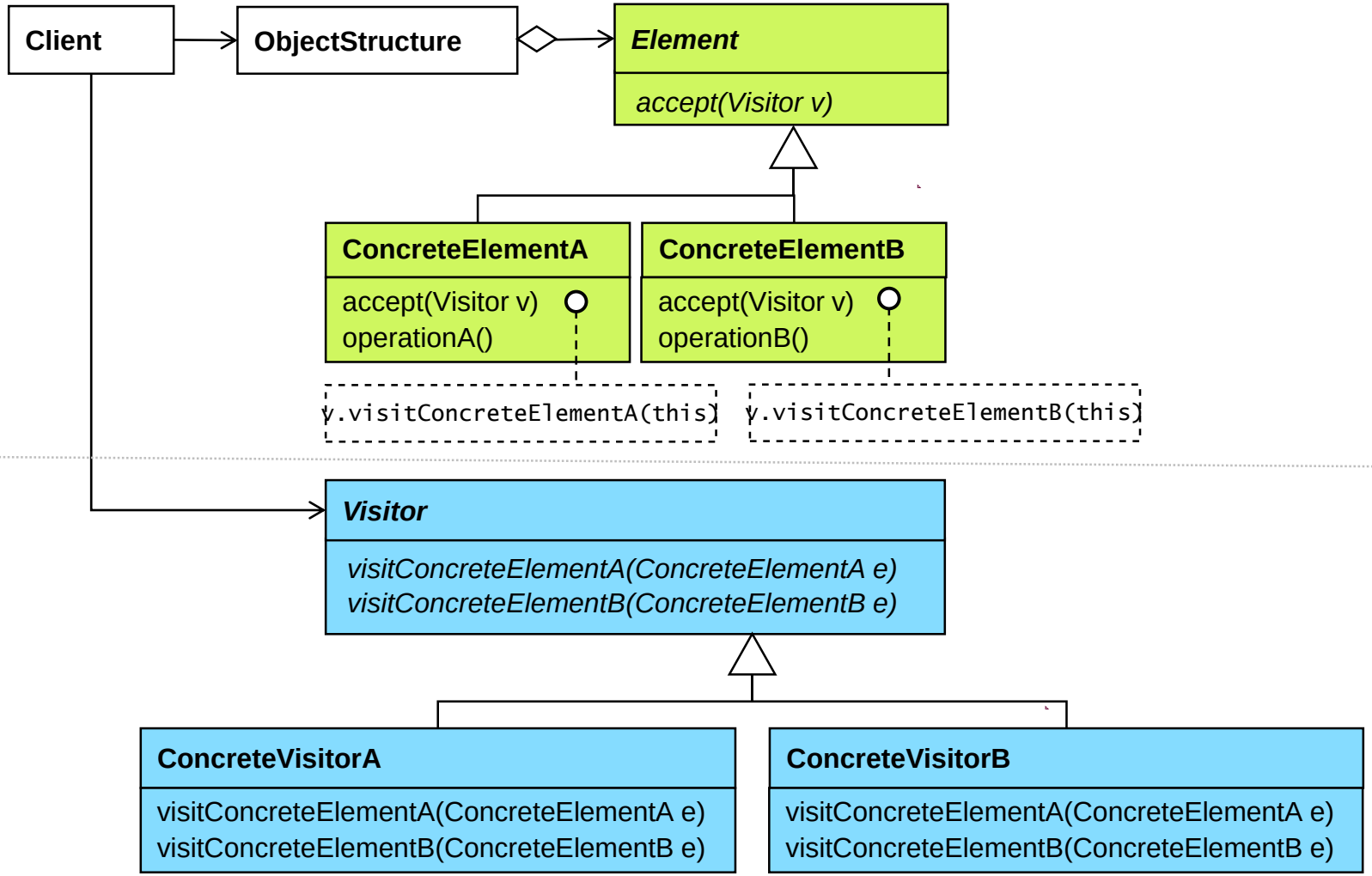
2.1.4 Visitor



Visitor

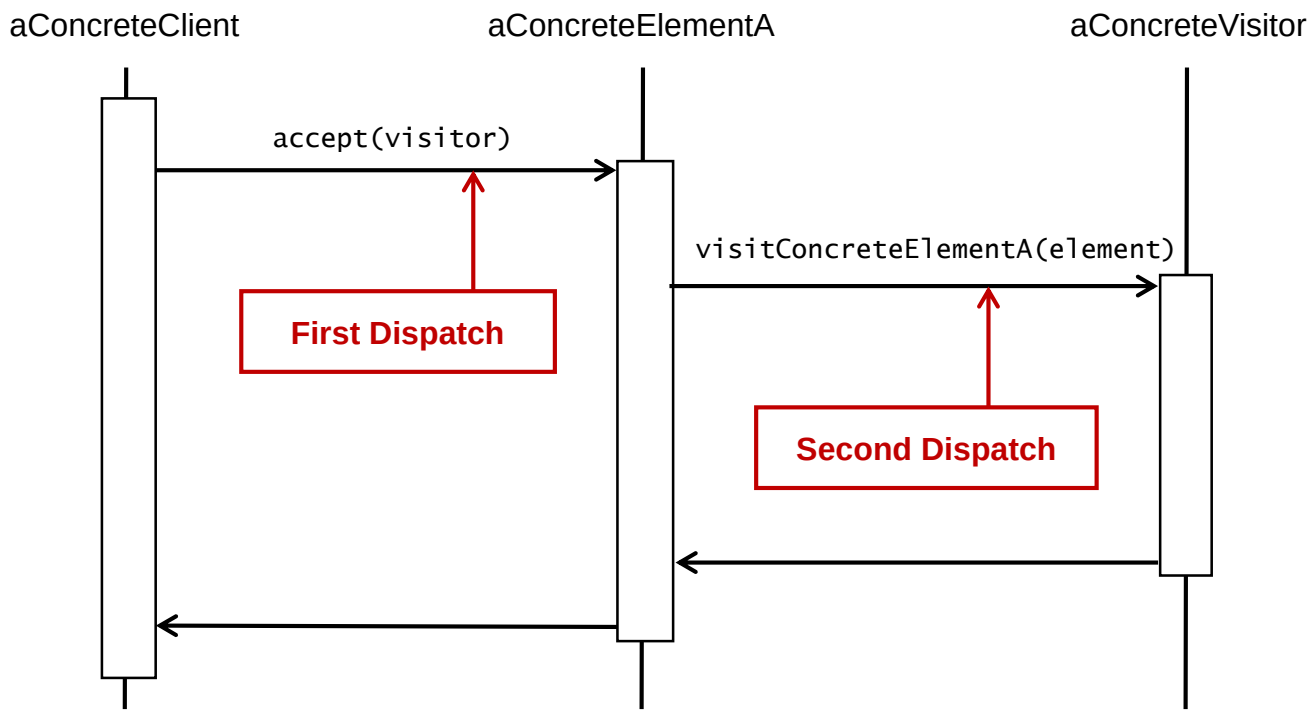
- Variant of DimensionalClassHierarchies
  - Template class hierarchy models a polymorphic data structure
  - In most cases a tree
- Hook hierarchy models a polymorphic algorithm
  - Encapsulate an operation on a collection (tree) as an object
  - Hook is an objectifier pattern (reified method)
- Separate tree inheritance hierarchy from command hierarchy
  - Simple extensibility of both hierarchies
  - Factoring (orthogonalization): simpler inheritance structures, otherwise multiplication of classes

Structure for Visitor

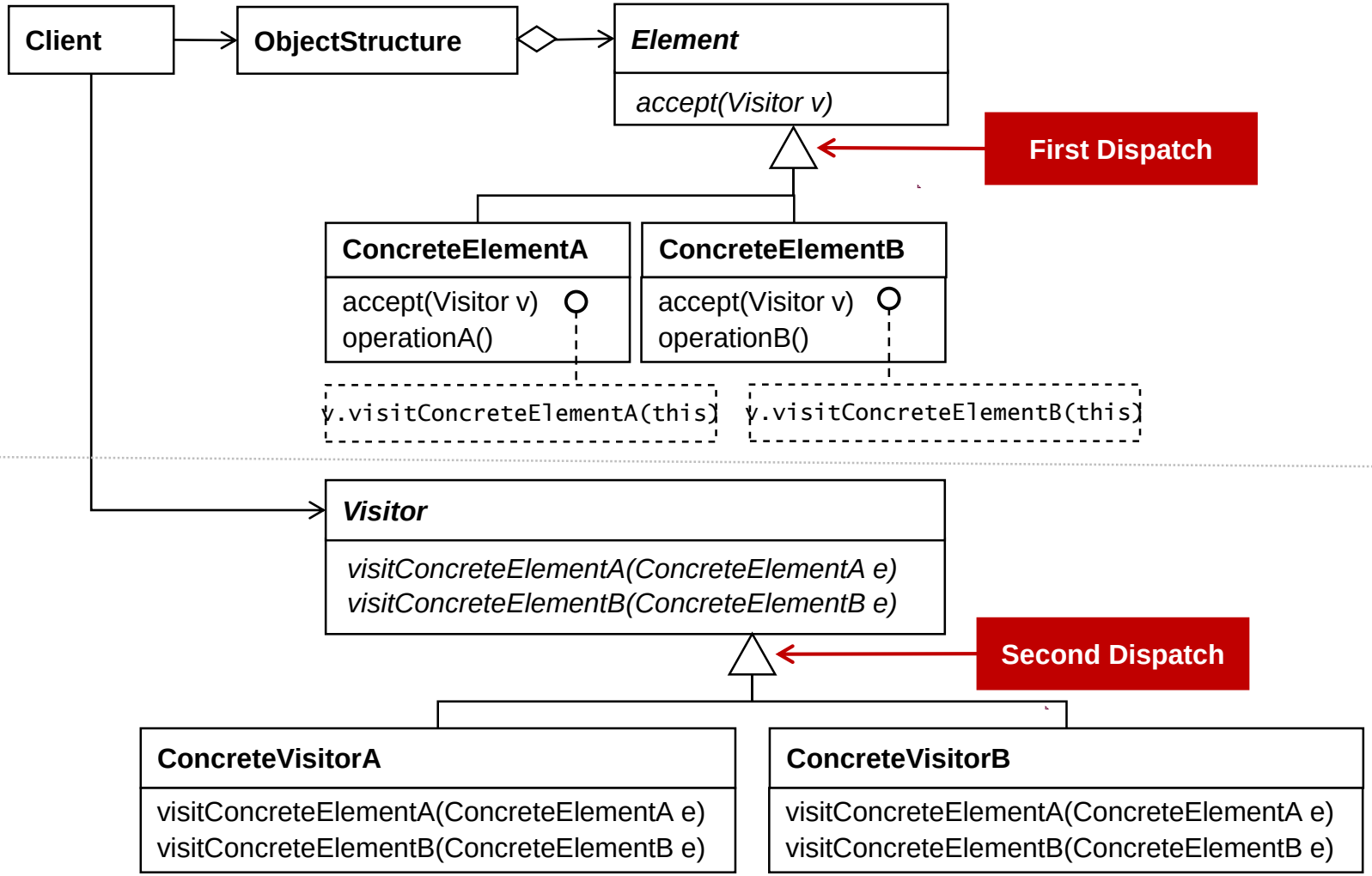


Sequence Diagram Visitor

- First dispatch on data, then on visitor



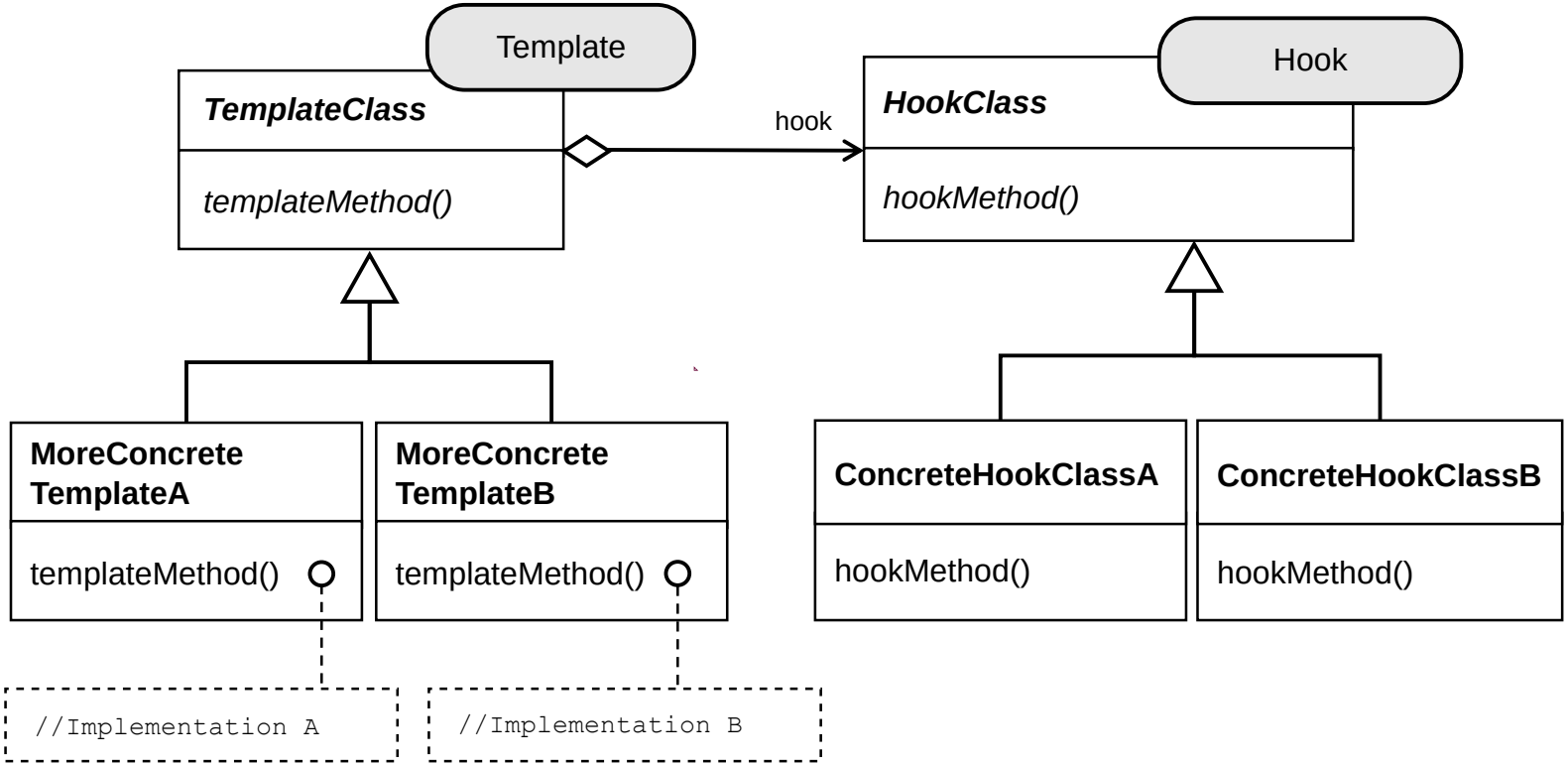
Structure for Visitor



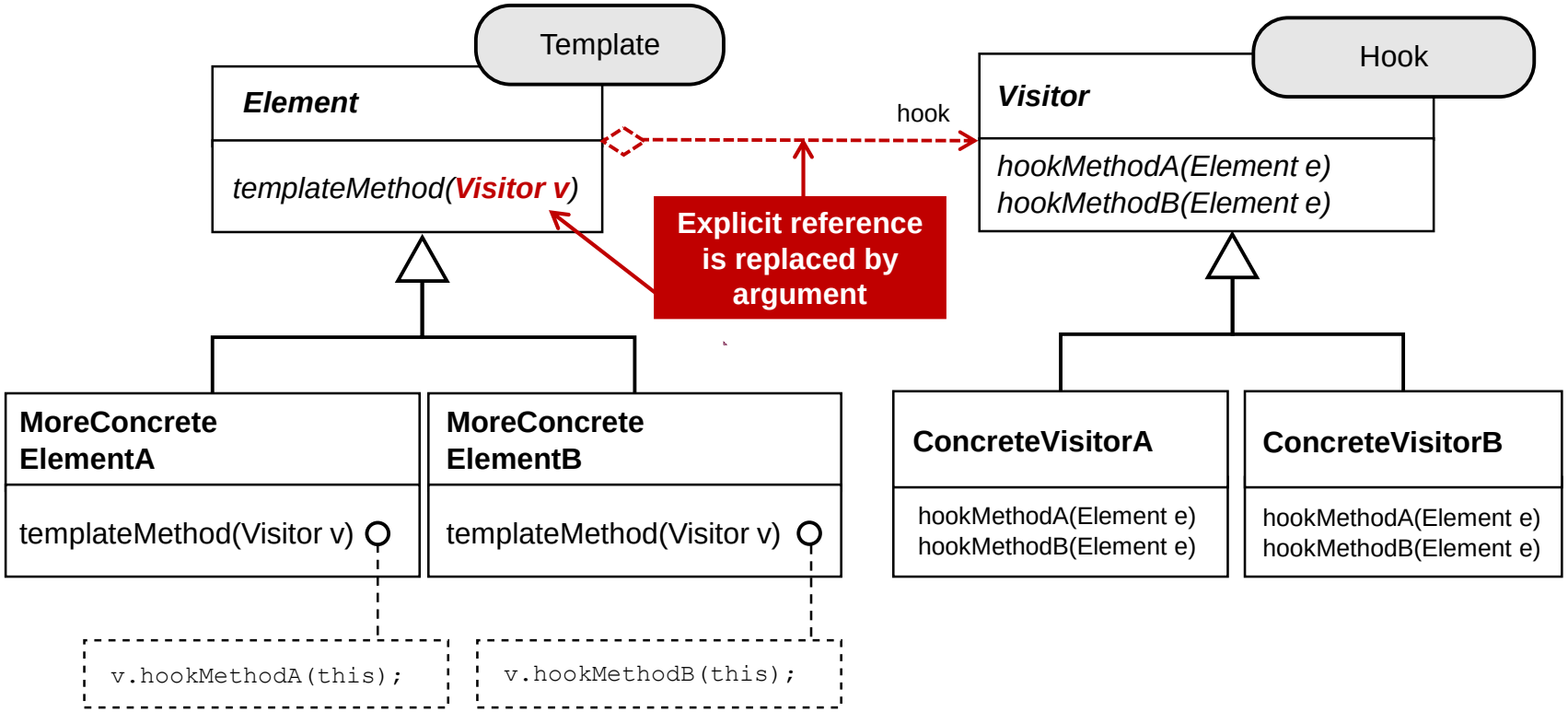
Visitor

- Implementation of a dimensional structure
  - First dispatch on dimension 1 *(data structure)*
  - Second dispatch on dimension 2 *(algorithm)*
  - Dimensions are not independent (no facets)
    - Chosen implementation of the algorithm depends on the chosen implementation of the data
- Abbreviation for **multimethods**
  - Dispatch/polymorphism on two arguments, not only the first (double dispatch)
  - First on data object (method *accept*), second on operation object (method *visit*)

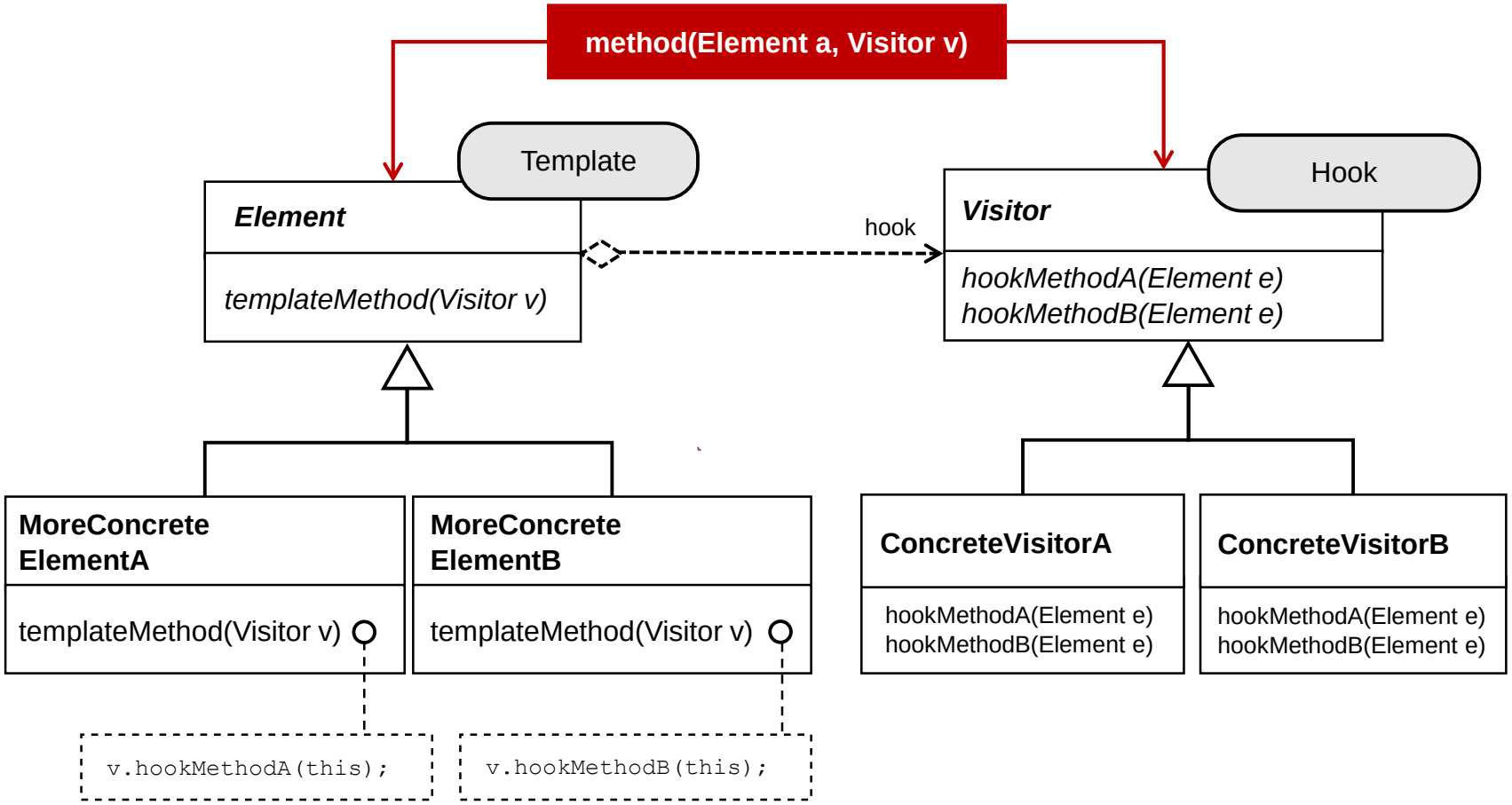
Remember: DimensionalClassHierarchies



Remember: DimensionalClassHierarchies

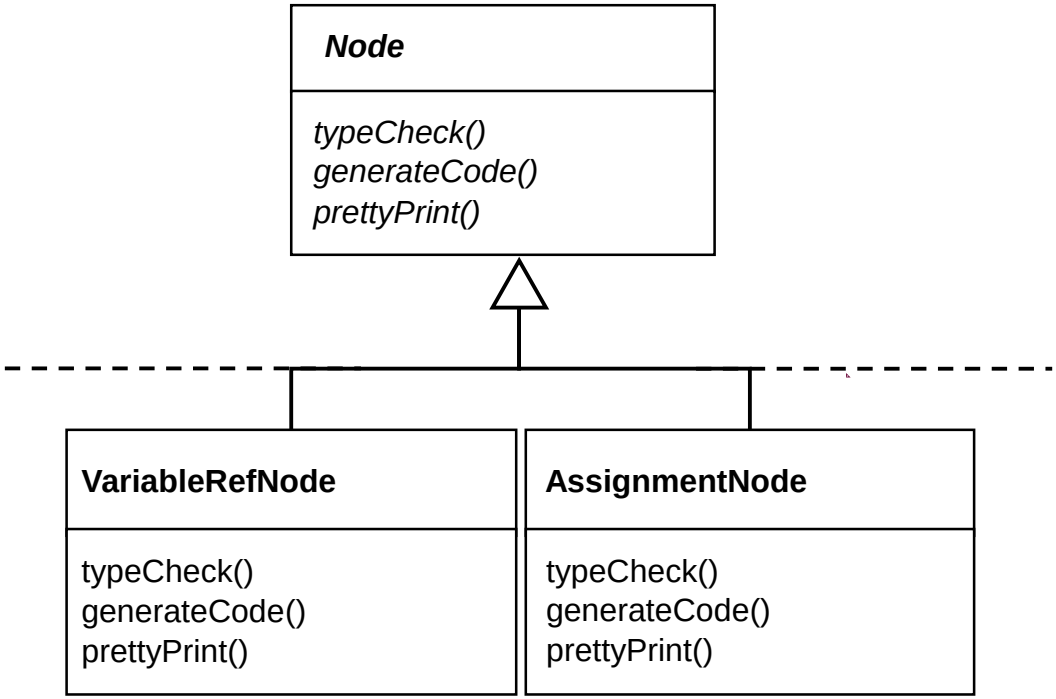


Visitor As Multimethod

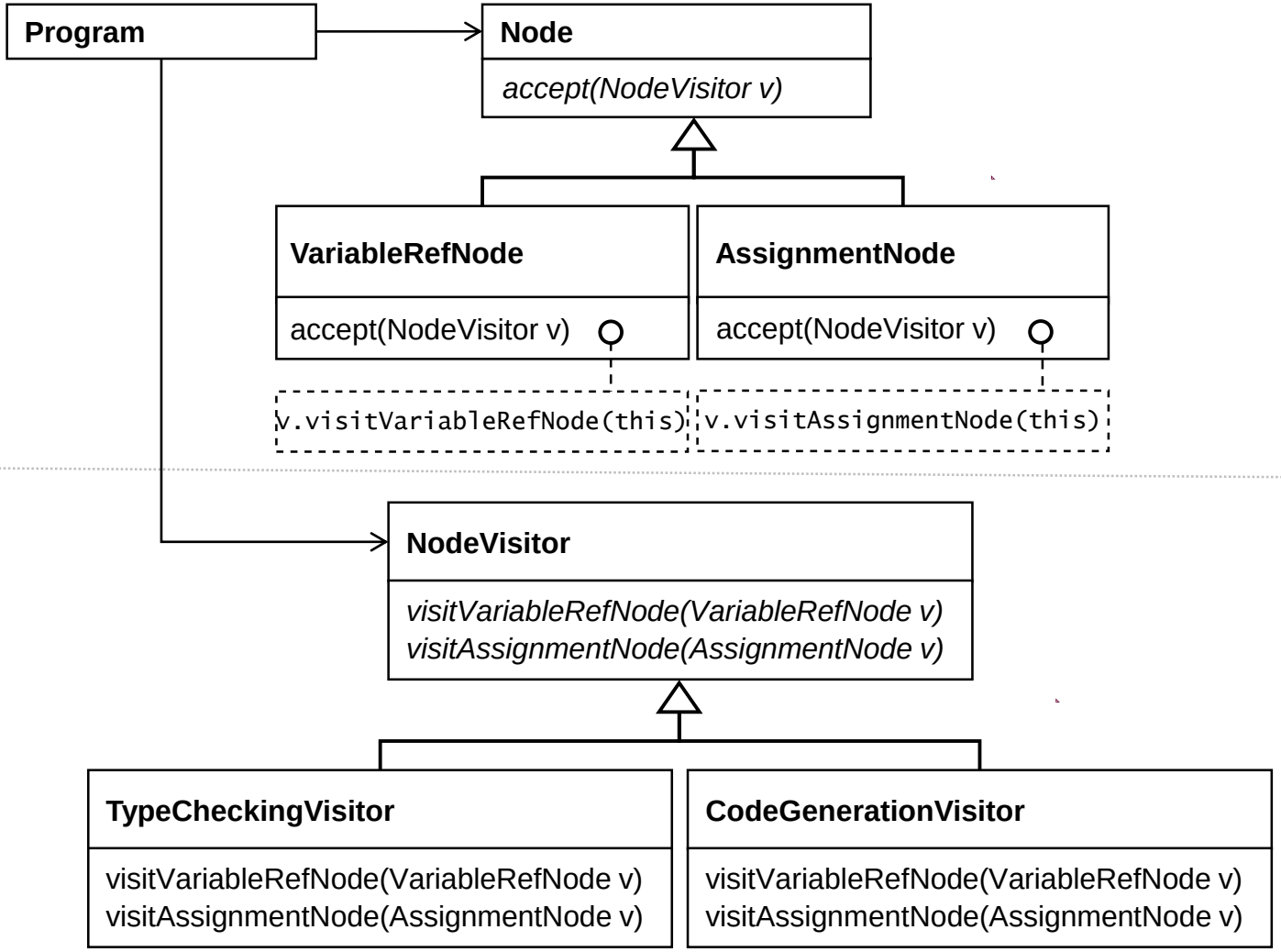


Example: Compiler Abstract Syntax Trees (ASTs)

- The operations are distributed over the classes.
- In case of extensions, all classes must be extender



Abstract Syntax Trees with Visitors



2) Dimensional Class Hierarchies (Bridges) as an Implementation of Facet Classifications

... in the following, we use the patterns Bridge and DimensionalClassHierarchies interchangeably

Facet Classifications

- A *facet* is an orthogonal dimension of a model
  - Every facet has its separate model
  - All facet classes are abstract
- Facets factorize inheritance hierarchies
  - Hence, facets simplify inheritance hierarchies
- Concrete classes in the (combined) model inherit from every dimension (every facet)
  - All classes in facets are *independent (*i.e., don't know of each other)
  - A concrete class offers the union of all features

62

Dijkstra on Aspects

"Let me try to explain to you, what to my taste is *characteristic for all intelligent thinking*. It is, that one is willing to *study in depth an aspect of one's subject* matter in isolation for the sake of its own consistency, all the time knowing that one is occupying oneself only *with one of the aspects*. We know that a program must be correct and we can study it from that viewpoint only; we also know that it should be efficient and we can study its efficiency on another day, so to speak. In another mood we may ask ourselves whether, and if so: why, the program is desirable. *But nothing is gained --on the contrary!-- by tackling these various aspects simultaneously*."

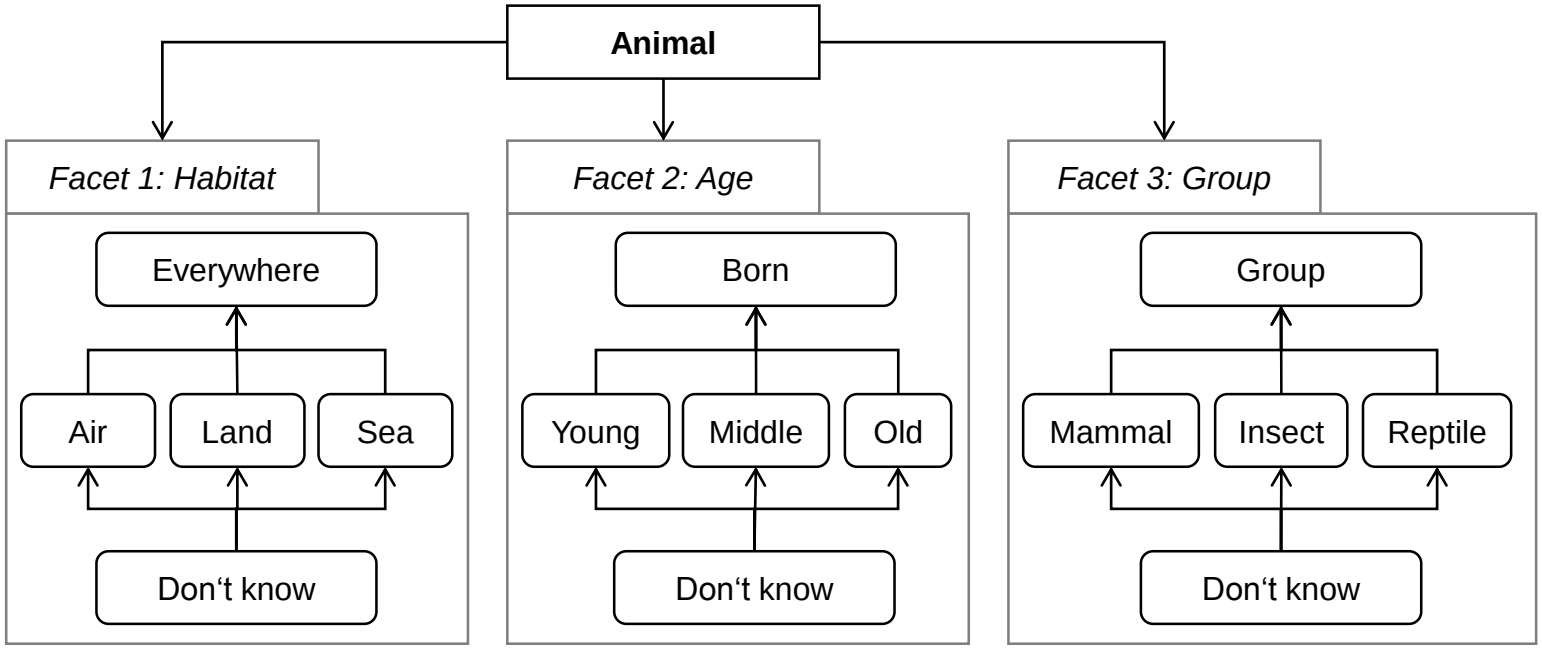
E. W. Dijkstra, On the Role of Scientific Thought, EWD 447 Selected Writings on Computing: A Personal Perspective, pages 60–66, 1982.

Separation of Concerns (SoC)

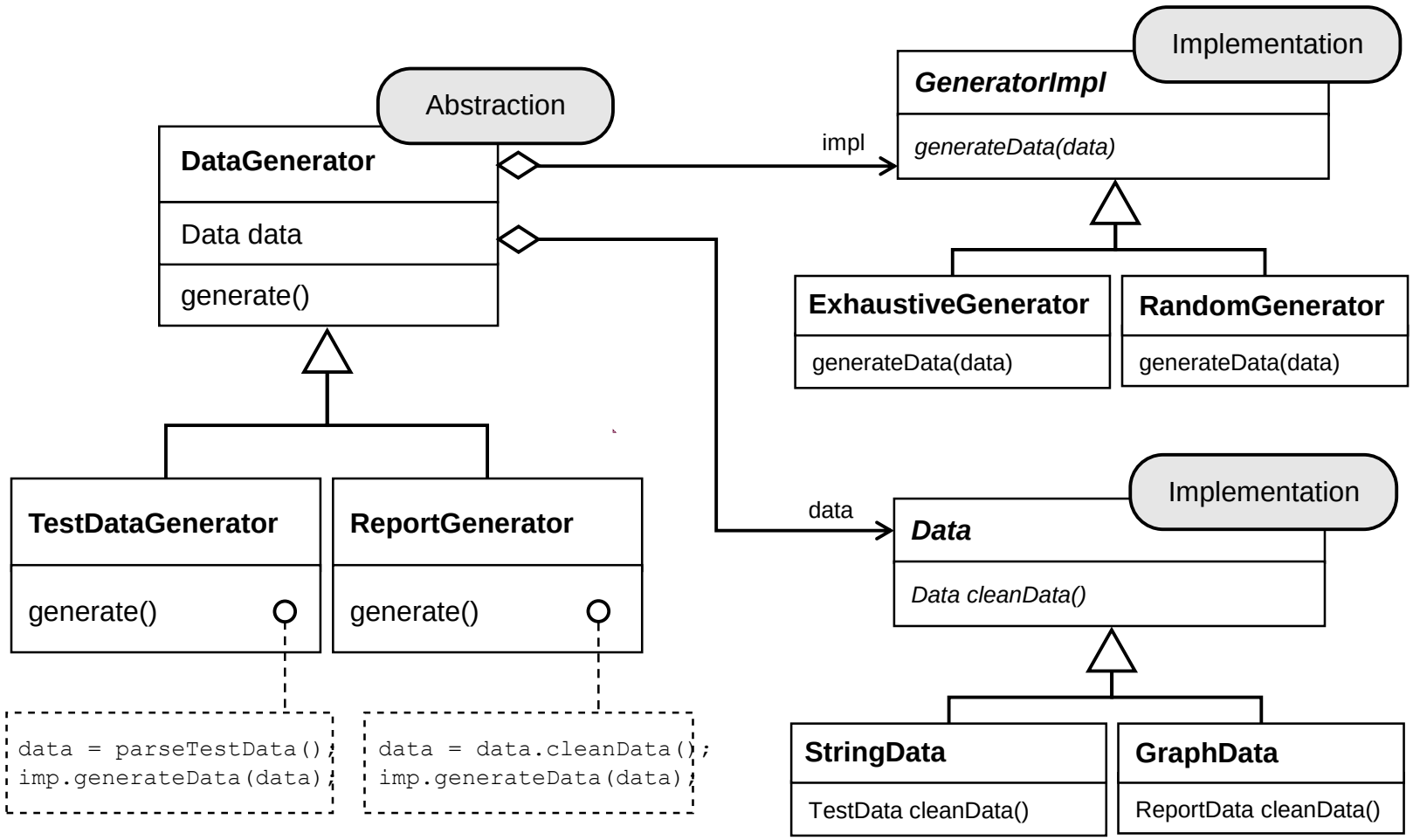
It is what I sometimes have called "the separation of concerns", which, even if not perfectly possible, is yet the *only available technique for effective ordering of one's thoughts*, that I know of. This is what I mean by "focussing one's attention upon some aspect": it does not mean ignoring the other aspects, it is just doing justice to the fact that *from this aspect's point of view, the other is irrelevant*. It is being one- and multiple-track minded simultaneously.

Facets in Living Beings

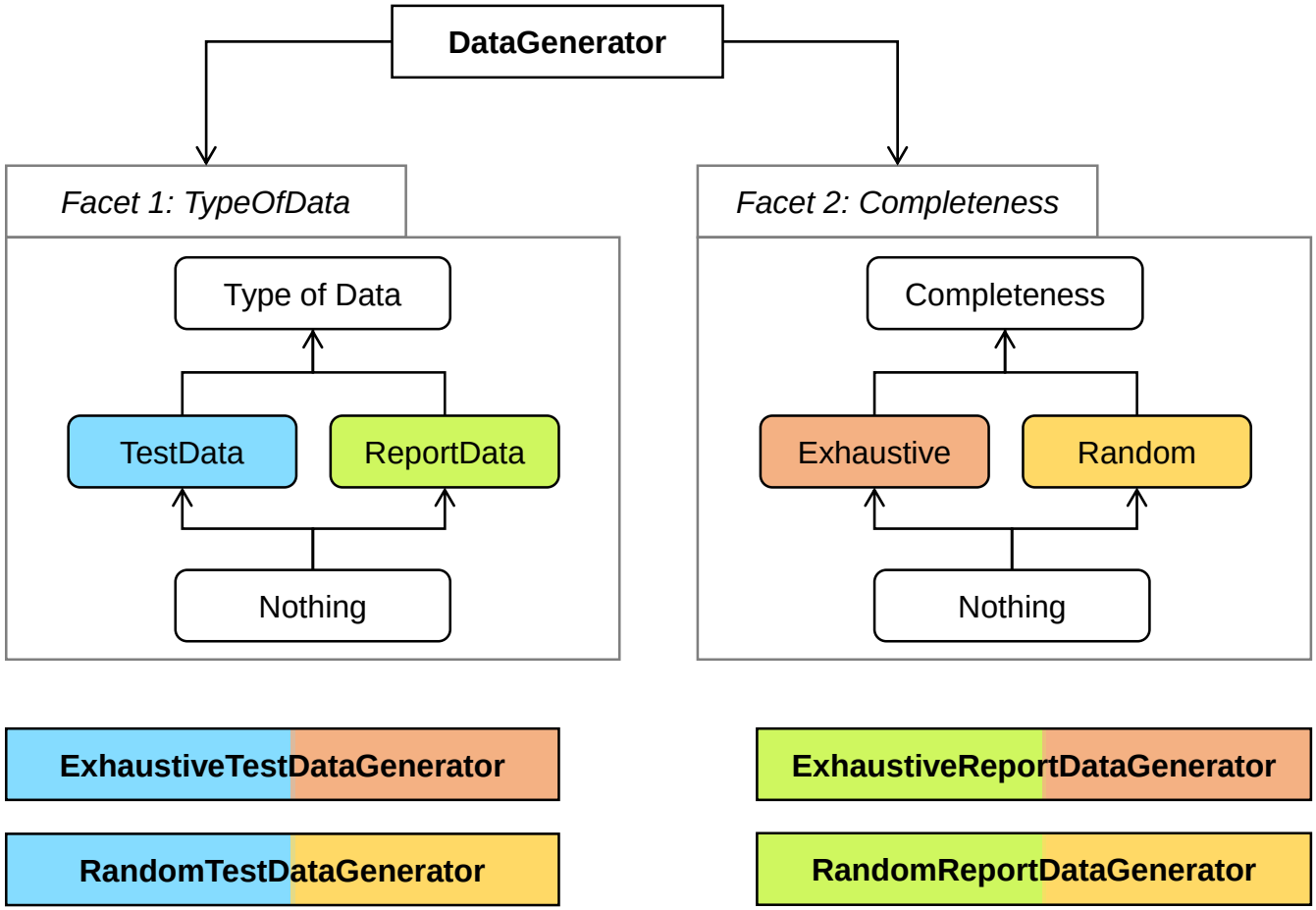
- The following model of Living Beings has 3 facets
  - Domain (where does an animal live?); Age; Group of Animal
- Final, concrete classes inherit from all facets.
- Facets Factorize Models: A full model would multiply all classes (3^n)



Remember: DataGenerator as Bridge



Facets of the Data Generator

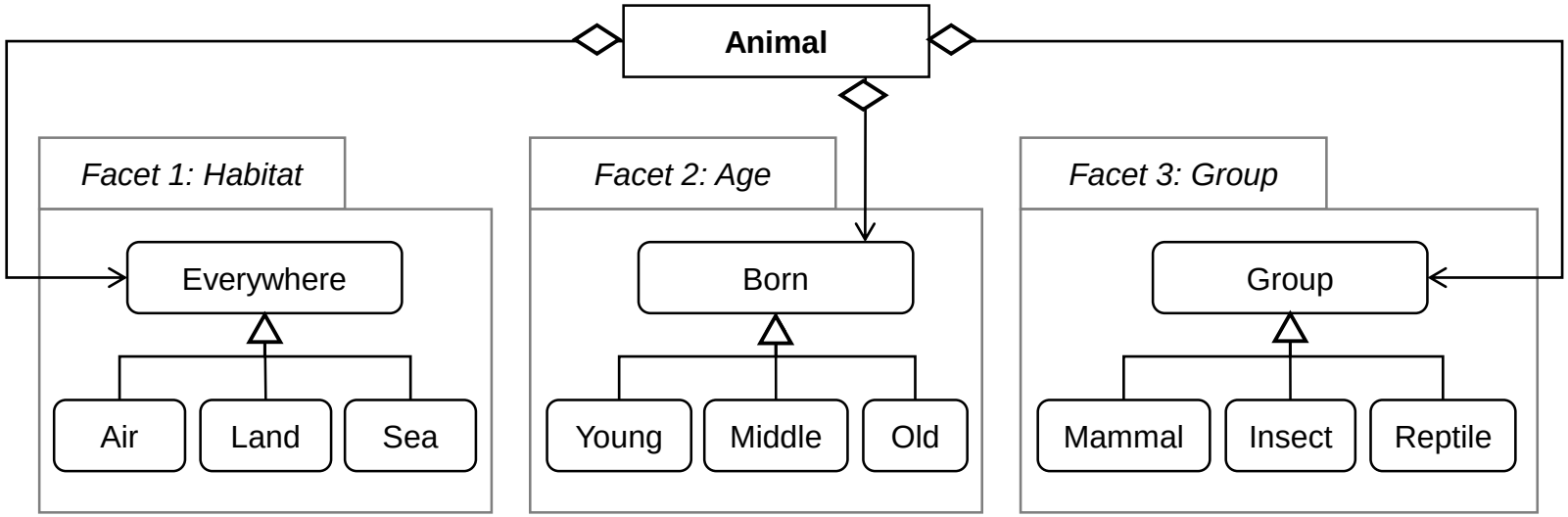


Facets Can Be Implemented by Multi-Bridges

- One central facet (abstraction)
- Others are delegatees in bridges (habitate, age, group, etc.)
- Advantage
  - + All facets can be varied independently
  - + Simple models
- Restriction: facets model only one logical object
  - With several physical objects

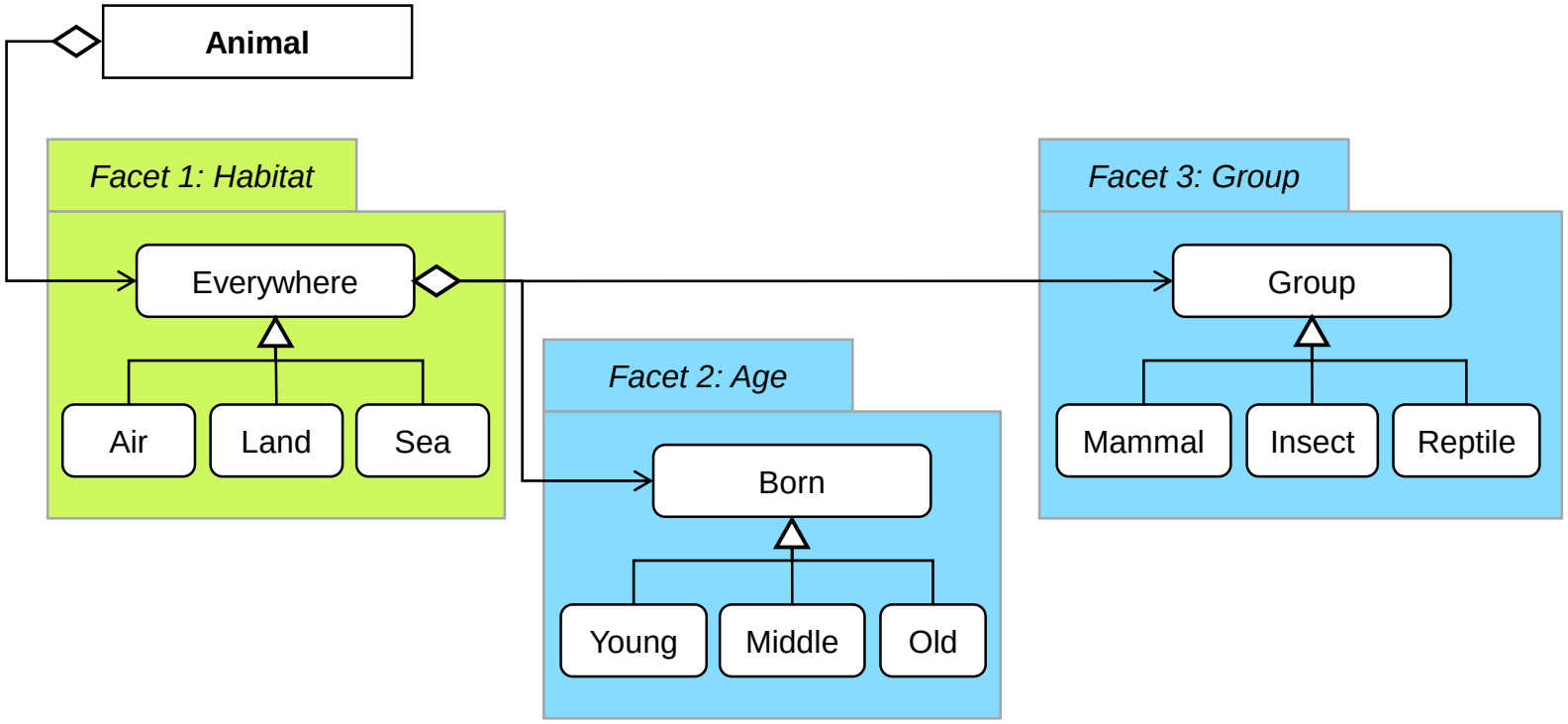
Multi-Bridge with Core Facet

- Animal as core facet, all others are *hook classes*
- *Bottom (e.g., "Don't know")* dispappears (transferred to multiplicity)
  - Multiplicity 1 = no bottom construct
  - Multiplicity 0 … 1 = bottom available



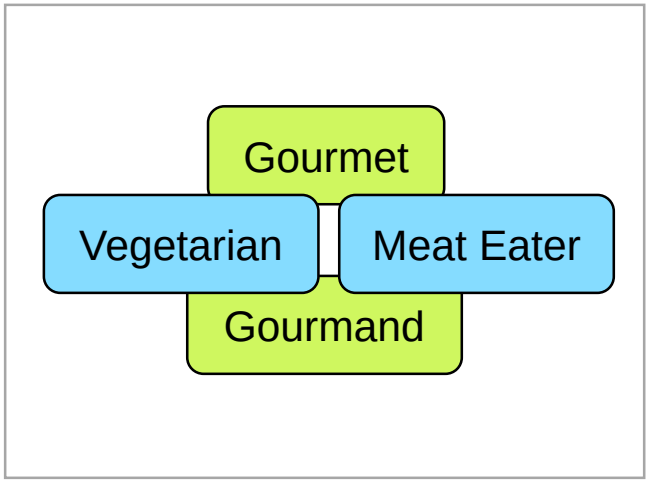
Multiple Bridge Without Core

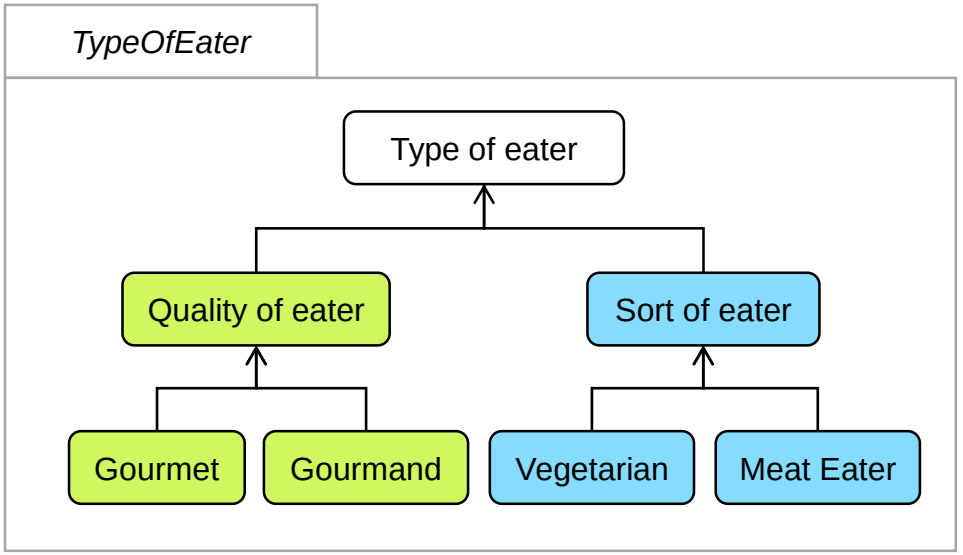
- Select a primary facet, relate others by bridges (n-Bridge)
- Problem: primary facet *knows* the others



How Can I Recognize Facets in Modelling?

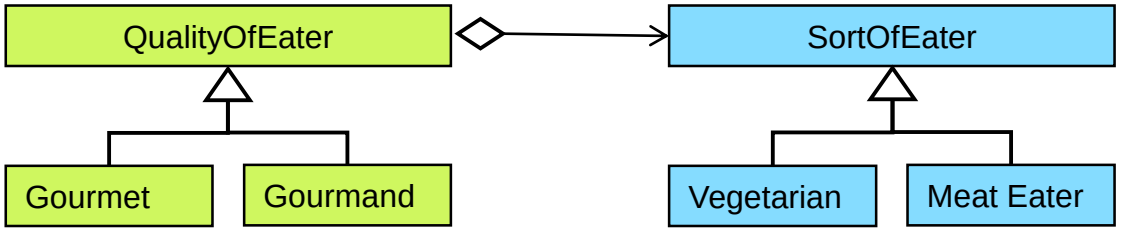
- If a class has *several different partitions*
- A model is ***not*** a facet model, if some class exists,
  whose heirs ***do not partition*** the class (non-partitioned inheritance)



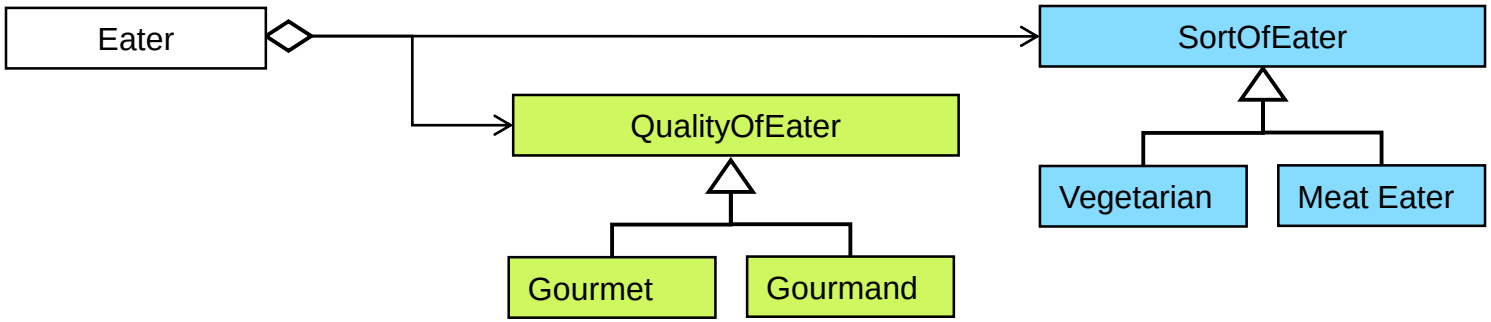


Resolve with DimensionalClassHierarchies (Bridge)

Simple Bridge



Double Bridge with Core



Modeling of Facet with Bridges

- Advantages:
  - + Dynamic variations possible
  - + Fewer classes
- Disadvantages:
  - – No type check on product classes
  - – No control over which combinations are created (illegal ones or undefined ones)
  - – Object schizophrenia
  - – Memory consumption with allocations
  - – Speed
  - – --> not for embedded systems!

Example: Classification of Research Papers after Shaw

- How to classify a research paper?
- When is it bad, when is it good?
- Mary Shaw proposed a facet-based classification with the facets
  - Research question
  - Result
  - Evaluation

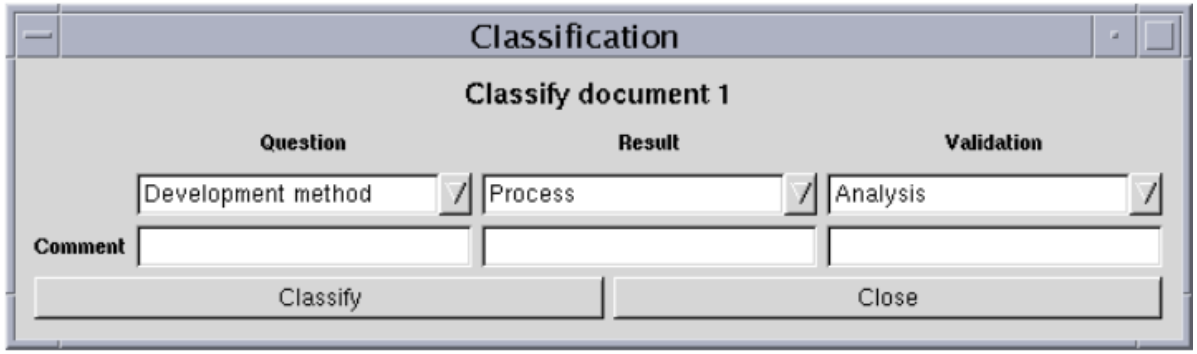
Classification of Research Papers

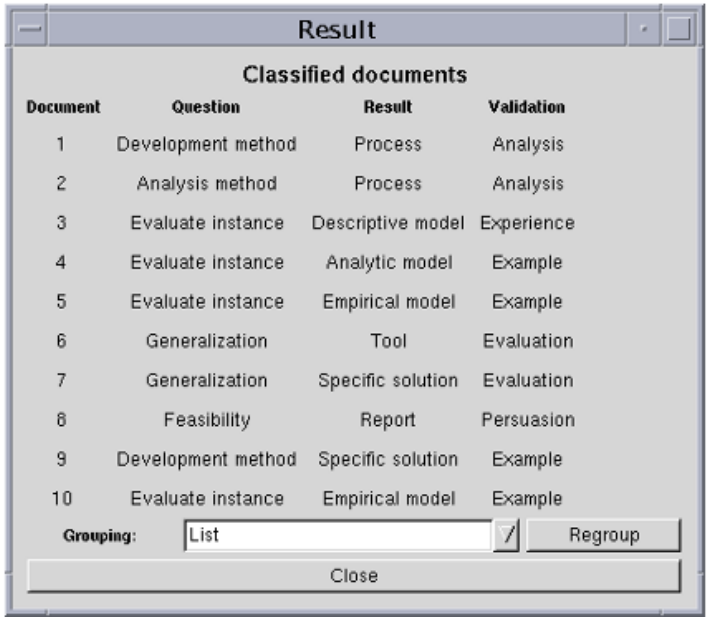
- 5 x 7 x 5 facet classes→175 product classes (types of research papers)

Question	Result	Validation
Development method	Procedure or technique	Analysis
Analysis method	Qualitative or decriptive model	Experience
Design or evaluation	Analytic model	Example
Generalization	Emperical model	Evaluation
Feasability study	Tool or notation	Persuasion
	Specification	
	Report	

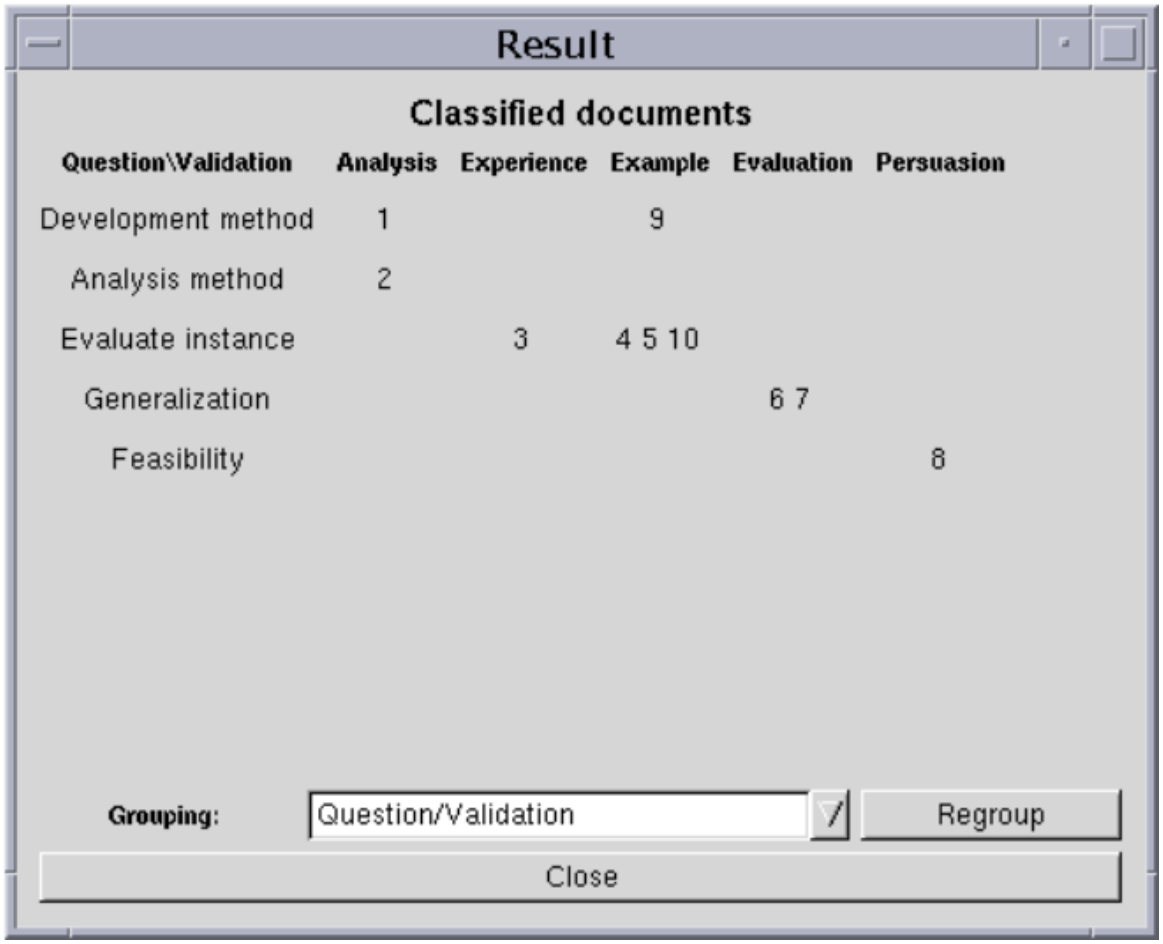
Mary Shaw, *Writing good software engineering research papers: minitutorial*, In Proceedings of the 25th International Conference on Software Engineering, 2003, Pages 726-736, IEEE Computer Society Washington, DC, USA

Classification of Research Papers





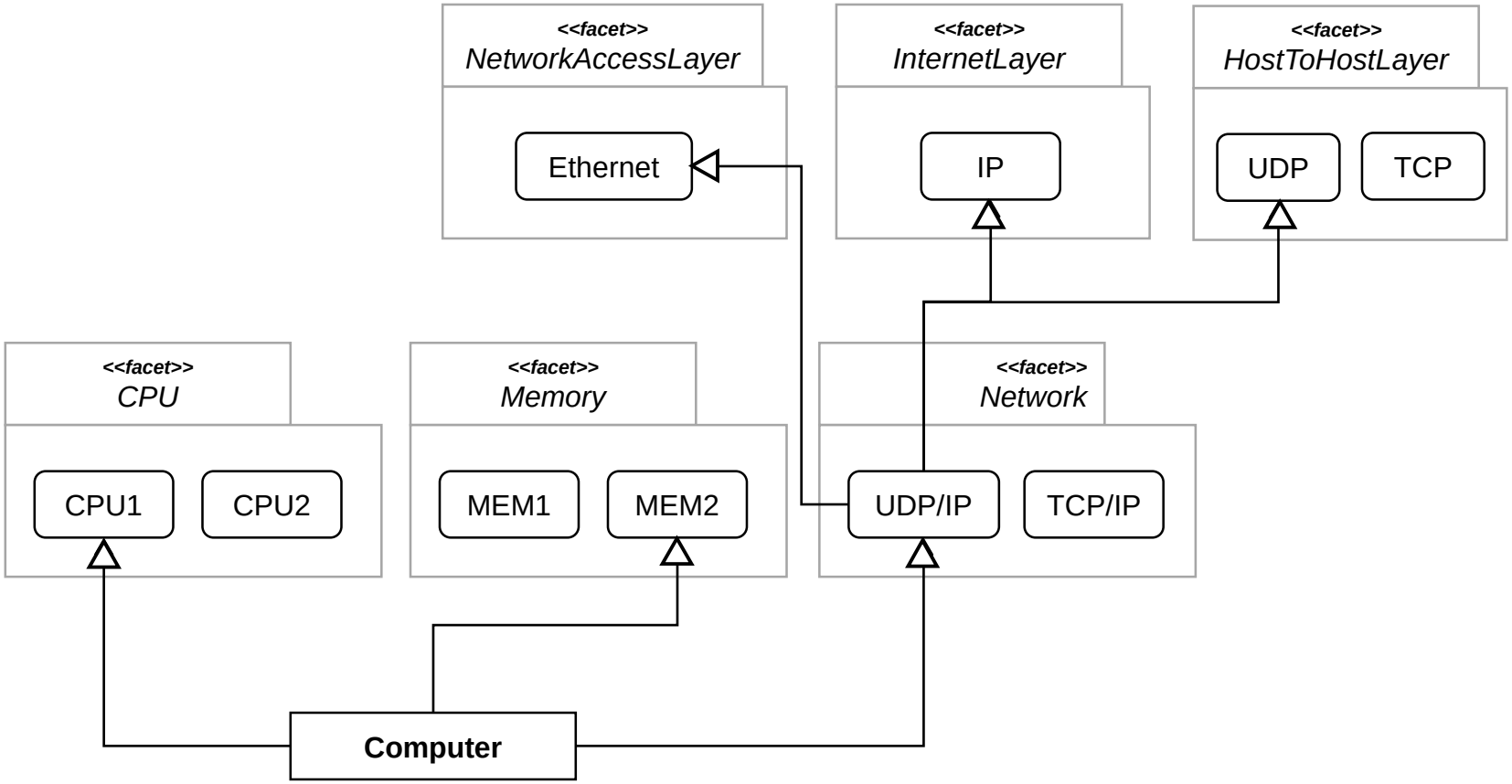
Classification of Research Papers



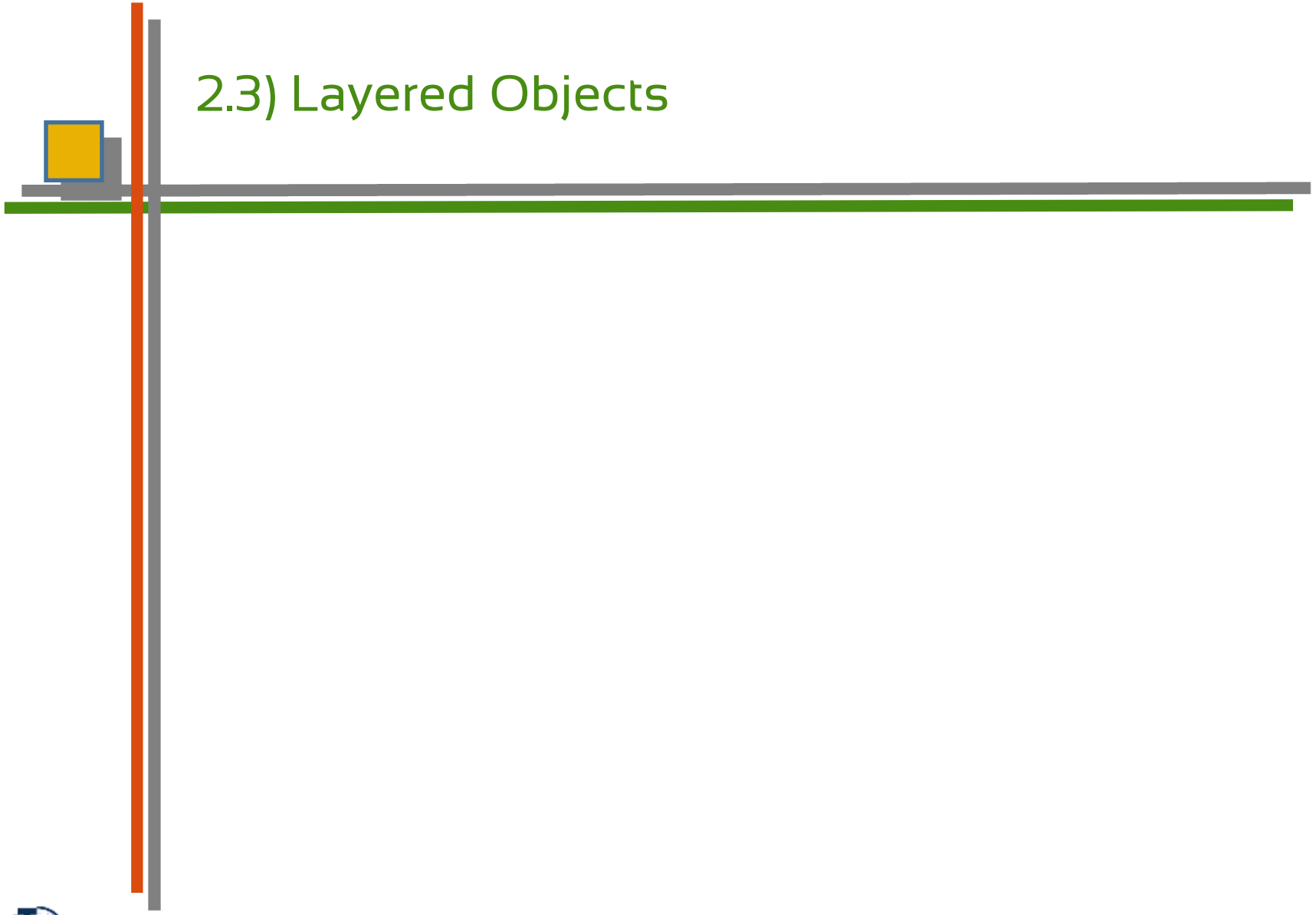
When to Use Facet-based Models

- When the model consists of independent dimensions
- When the model is very complicated
- Realizations:
  - *Use multiple inheritance*,
    when type checking is necessary (e.g., in frameworks)
  - *Use Bridge*,
    when language does not support multiple inheritance,
    or dimensions change dynamically

Several Facet Groups are Possible



2.3) Layered Objects

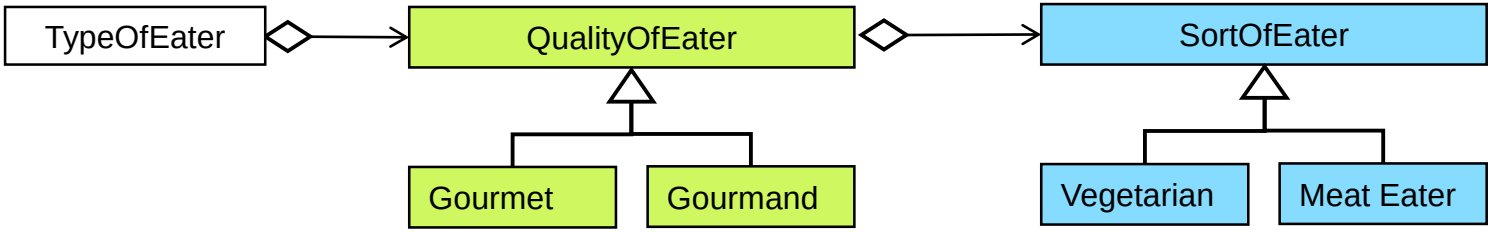


Be Aware

- If you meet a Bridge, you may have a facet classification
- Only question: are the dimensions independent?
- Sometimes, dependencies exist (e.g., one dimension calls another)
  - This requires an interface (contract) between the dimensions

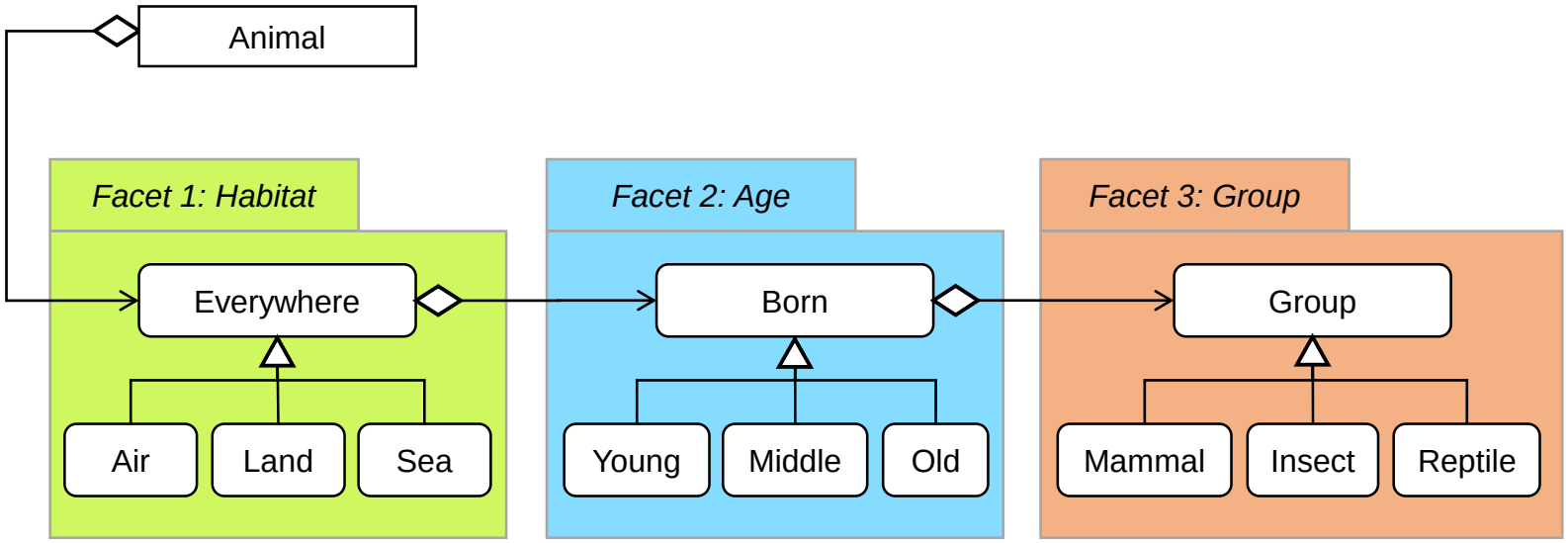
Layered Objects with Chain-Bridge

Chain Bridge with Core



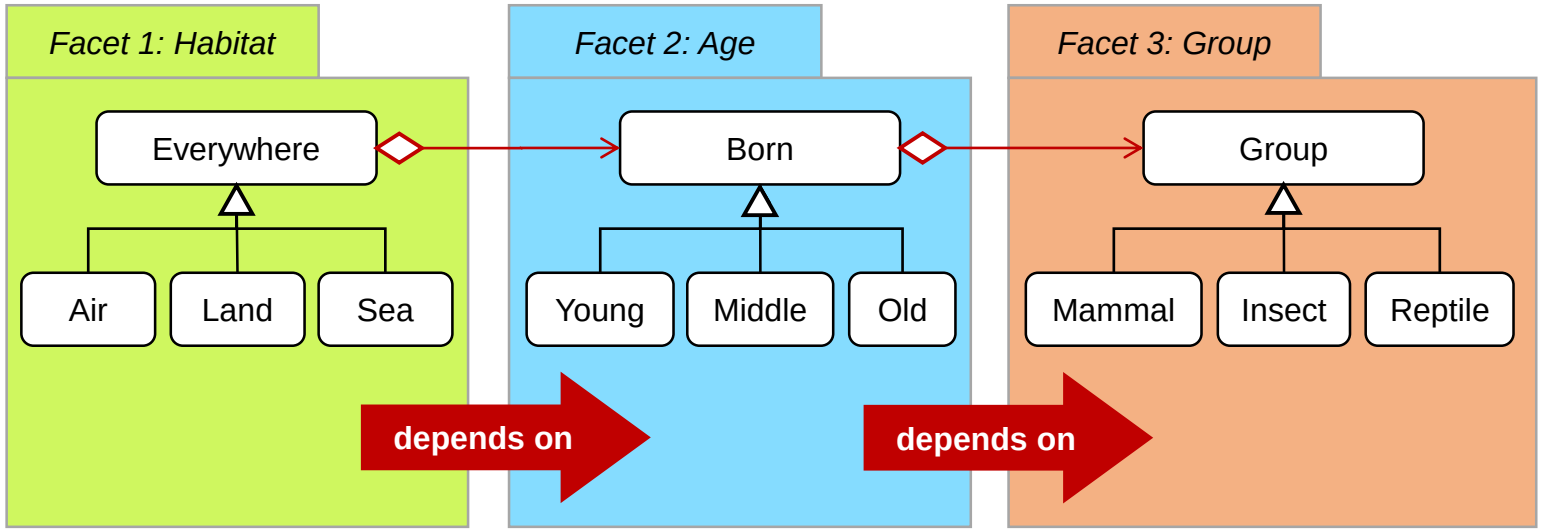
Chain-Bridge for Layered Object Implementation

- Select a primary facet, relate others by chain-Bridges
- Here without core

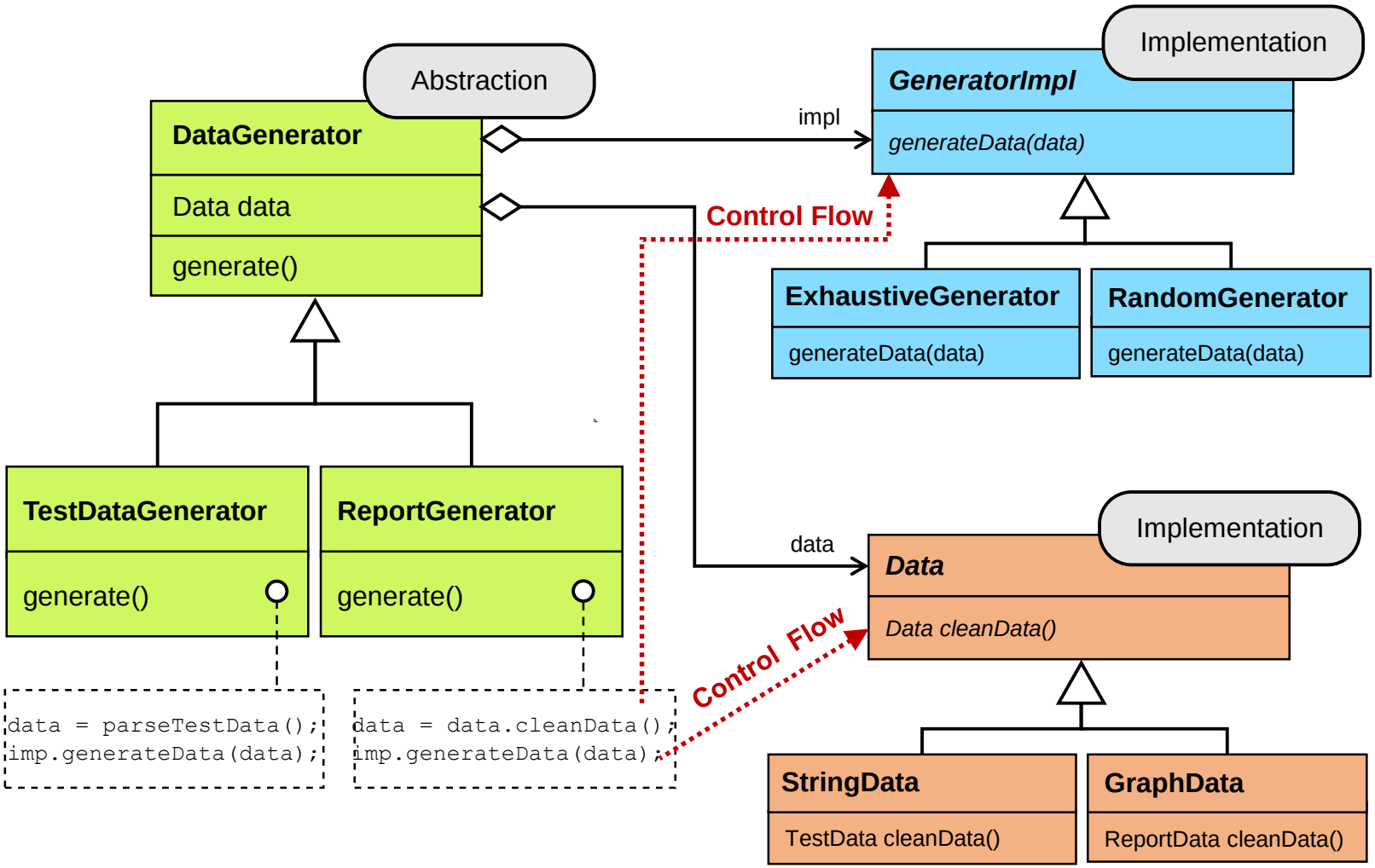


Layered Objects

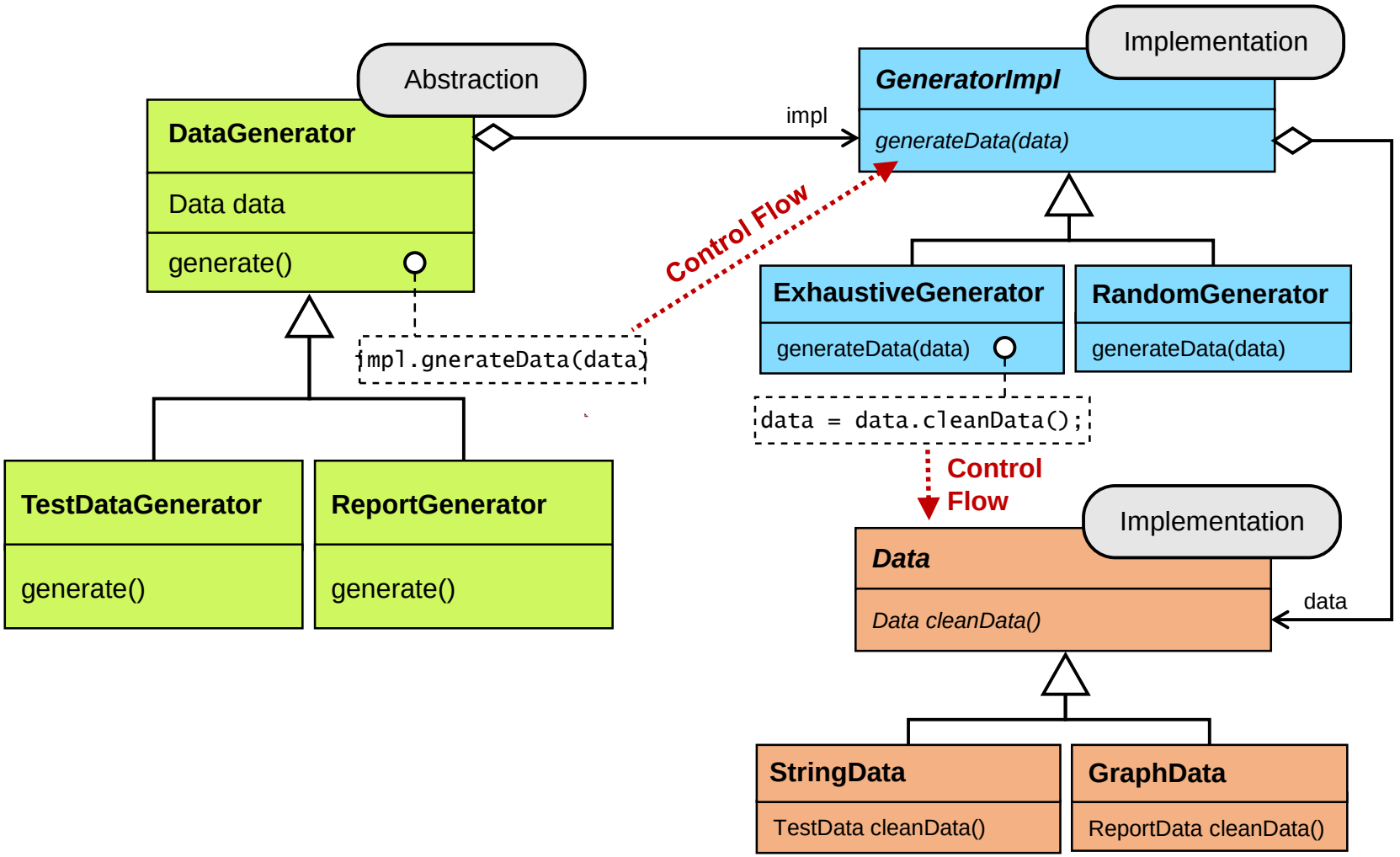
- Upper layers depend on information of lower layers



Remember: DataGenerator as Bridge

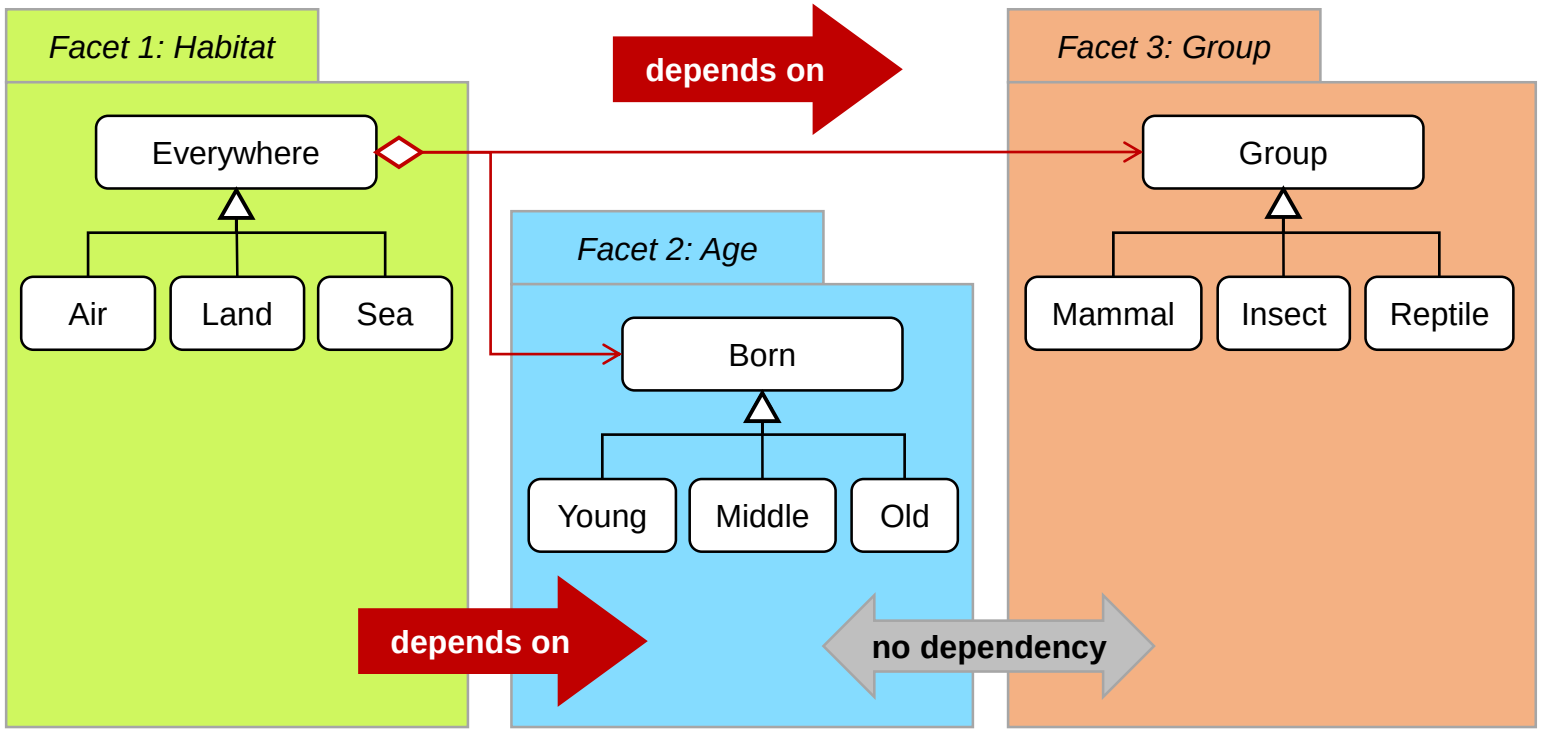


DataGenerator as 2-Chain-Bridge



Compare to Facets

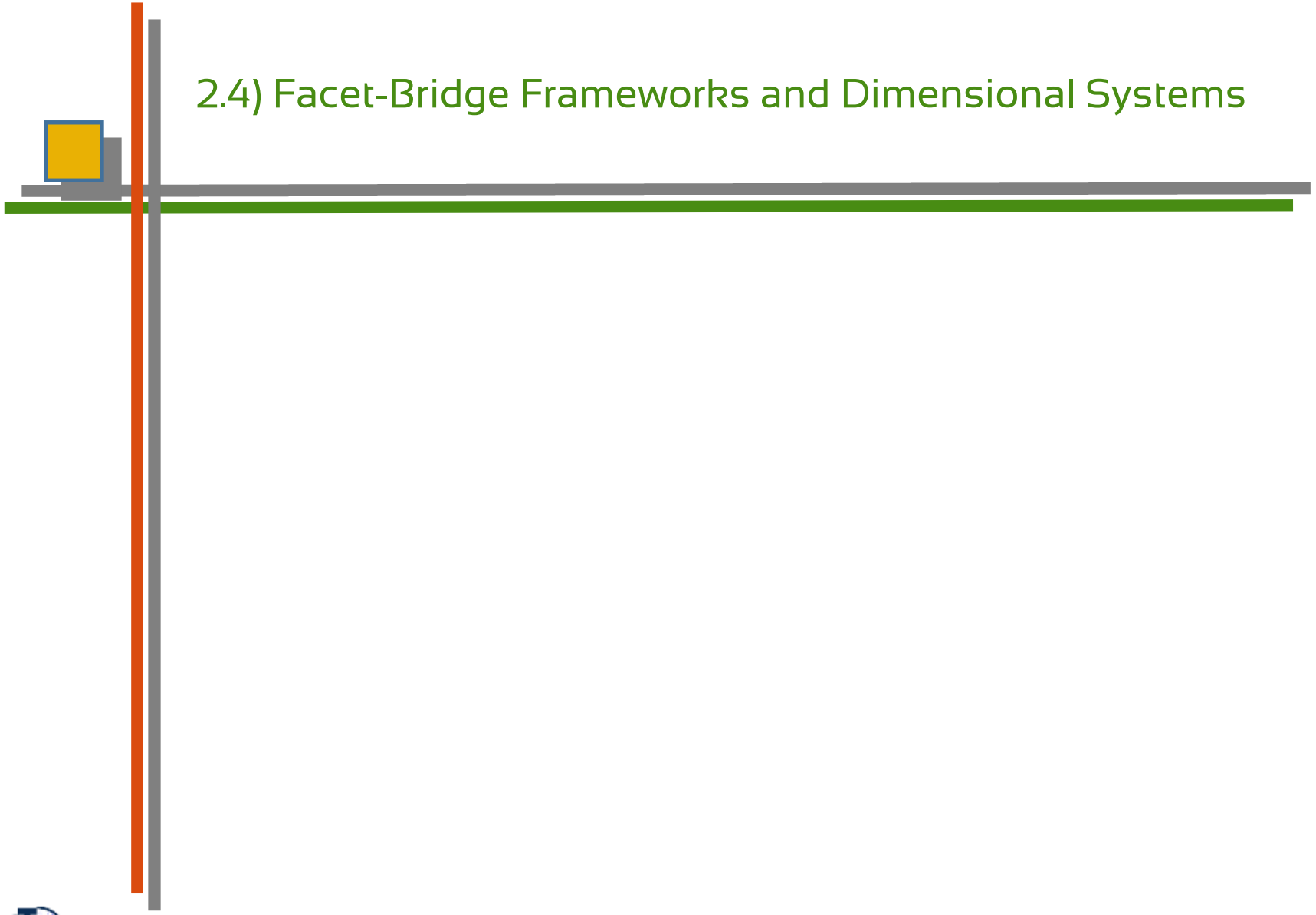
- Dimensions do not depend on information of others



Layered Object Spaces

- A layered object space is *similar* to a facet space
- Layers exchange information in a *directed way*
  - *Upper layers* ***call*** *lower layers*,
    which *deliver* information to upper layers
  - Abstract topmost classes in a layer provide abstract methods
    that can be called from other layers
  - The dependencies are *directed and acyclic* (form a DAG)
- All classes in a layer can be exchanged (must conform to the interface)
- Layered object spaces are much broader applicable than facet spaces

2.4) Facet-Bridge Frameworks and Dimensional Systems

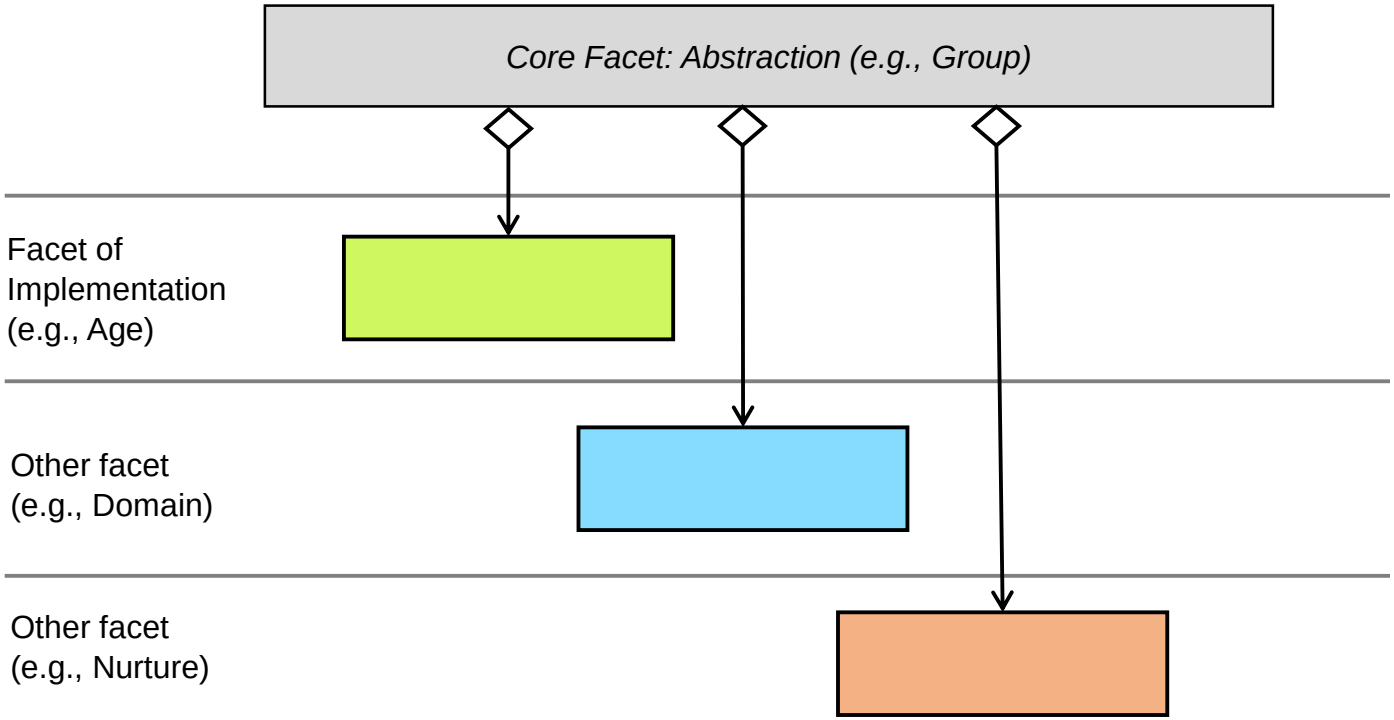


Multiple Bridges for Facet-Based Systems

- So far, we looked at implementations of faceted or layered objects, i.e., models of complex objects
- Facet classifications and layered objects can be generalized to facet-based or dimensional *frameworks* and *systems*

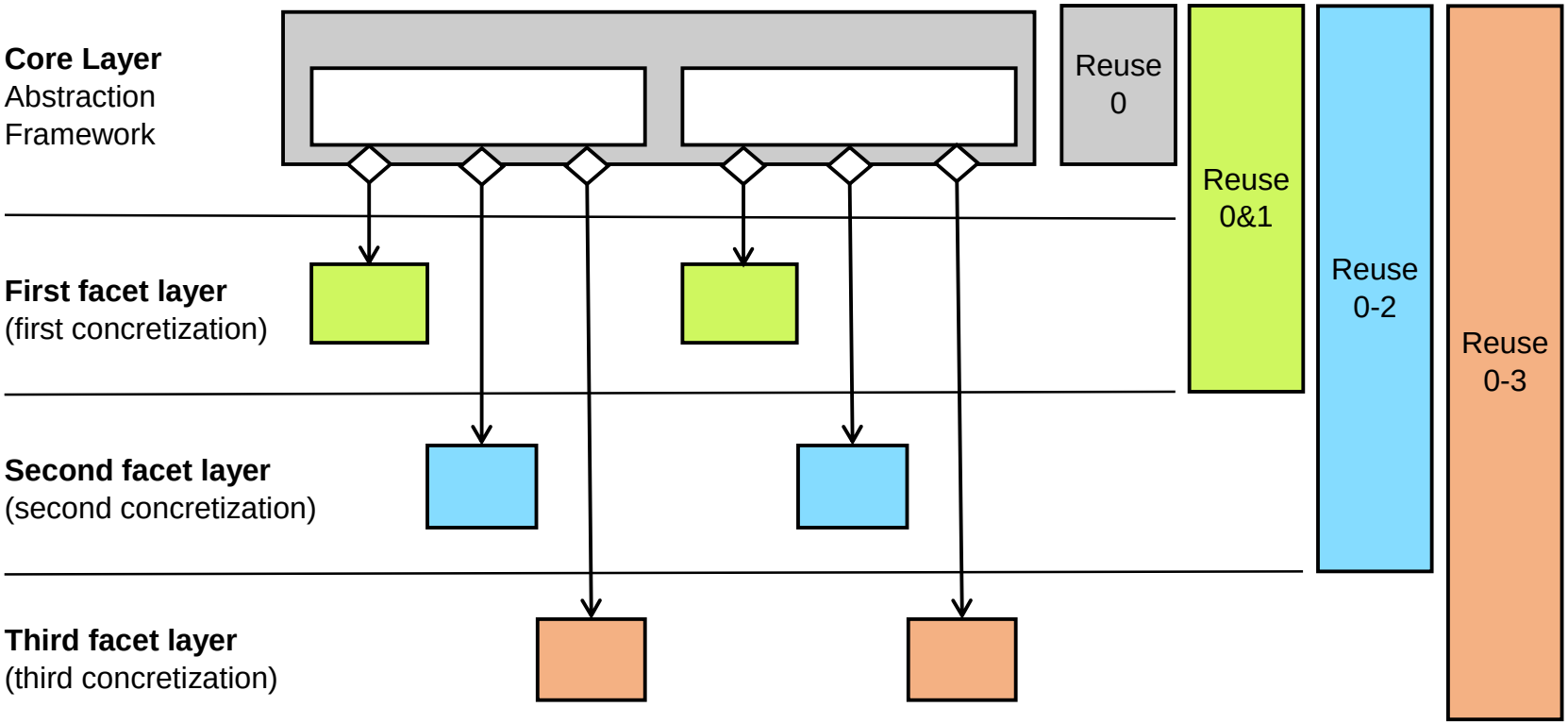
Facet-Bridge Dimensional Systems

- Bridge patterns can be divided upon different dimensions
- Here: a triple Bridge with core and 3 dimensions, all independent



Facet-Bridge Frameworks for Facet-Based Systems

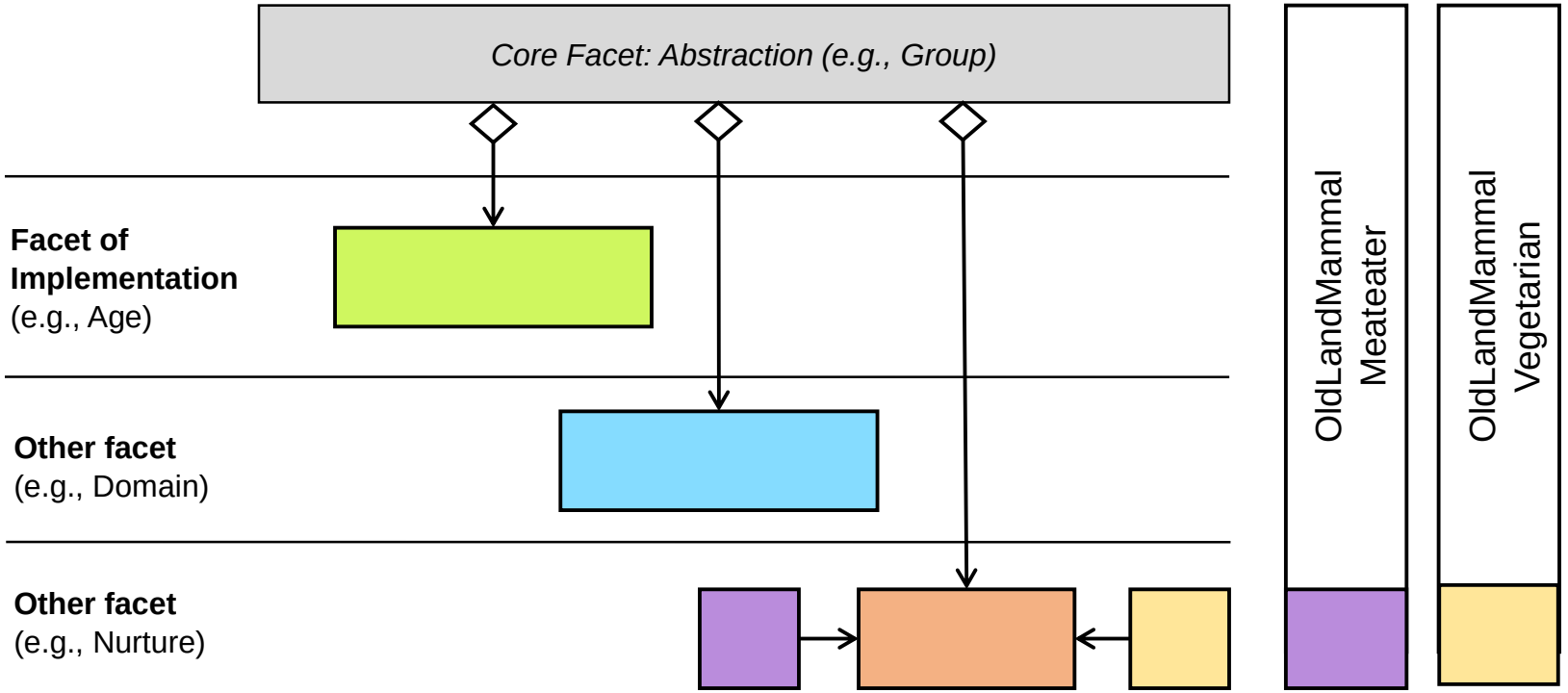
- If one or several layers are fixed, and the rest is variable, *facet frameworks* result



Facet-Bridge Frameworks for Facet-Based Systems

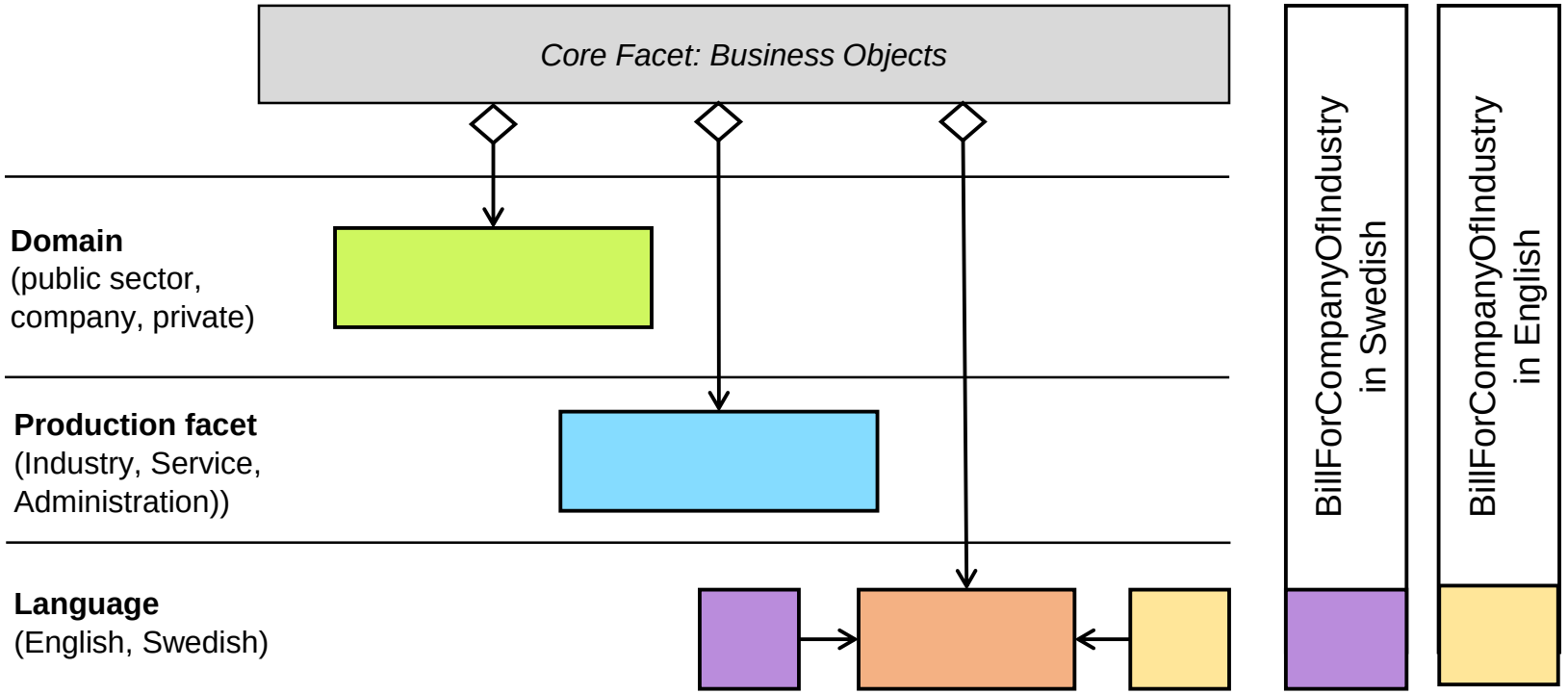
OldLandMammalVegetarian

- Products can be instantiated on every level
- Every dimension provides additional reuse



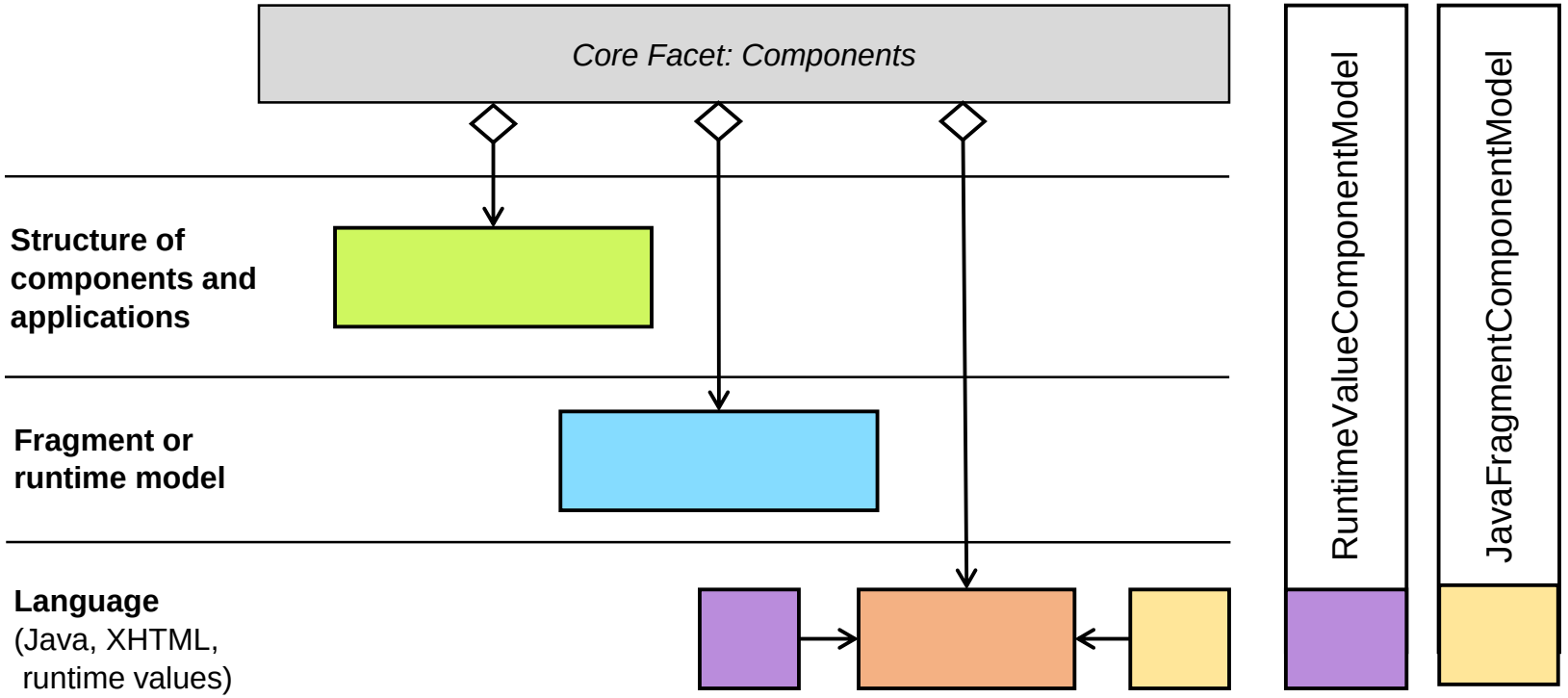
Facet-Bridge Frameworks for Business Software

- SAP, Peoplesoft, Intentia, IFS



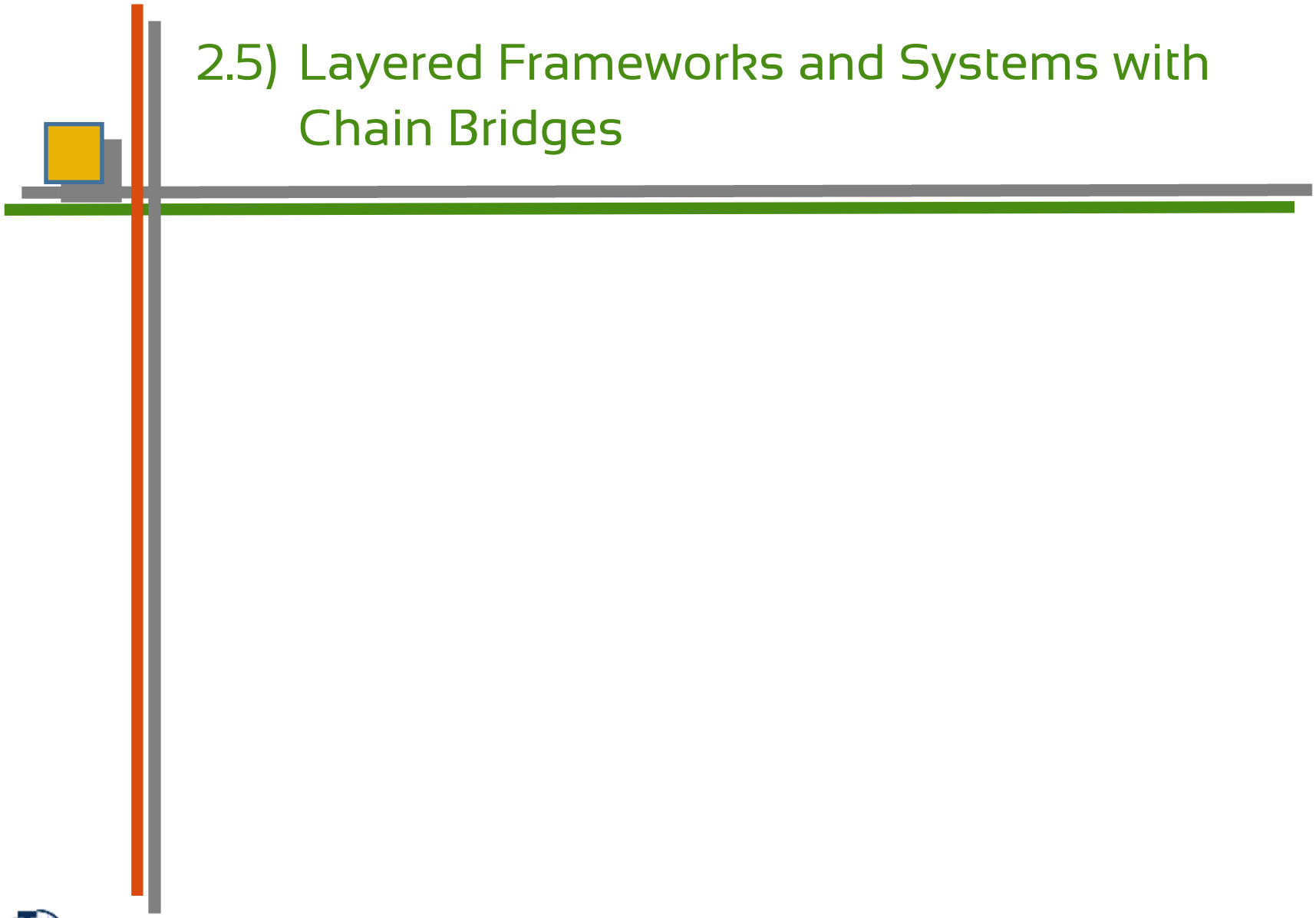
Facet-Bridge Frameworks for Component Composition

- COMPOST is a framework for constructing component systems (a framework for frameworks)



Facet-based Design and Frameworks

- Best practice: whenever you have a huge class hierarchy (that is not completely based on partitioning)
  - Find out the facets
  - Factor the inheritance hierarchy into the facets
  - Choose a core facet
  - And implement the facets with a facet framework with Bridges.
  - For an n-dimensional facet problem you need at least n-1 Bridges
- If the “facets” are not independent, introduce layers
  - And implement them with Chain-Bridges



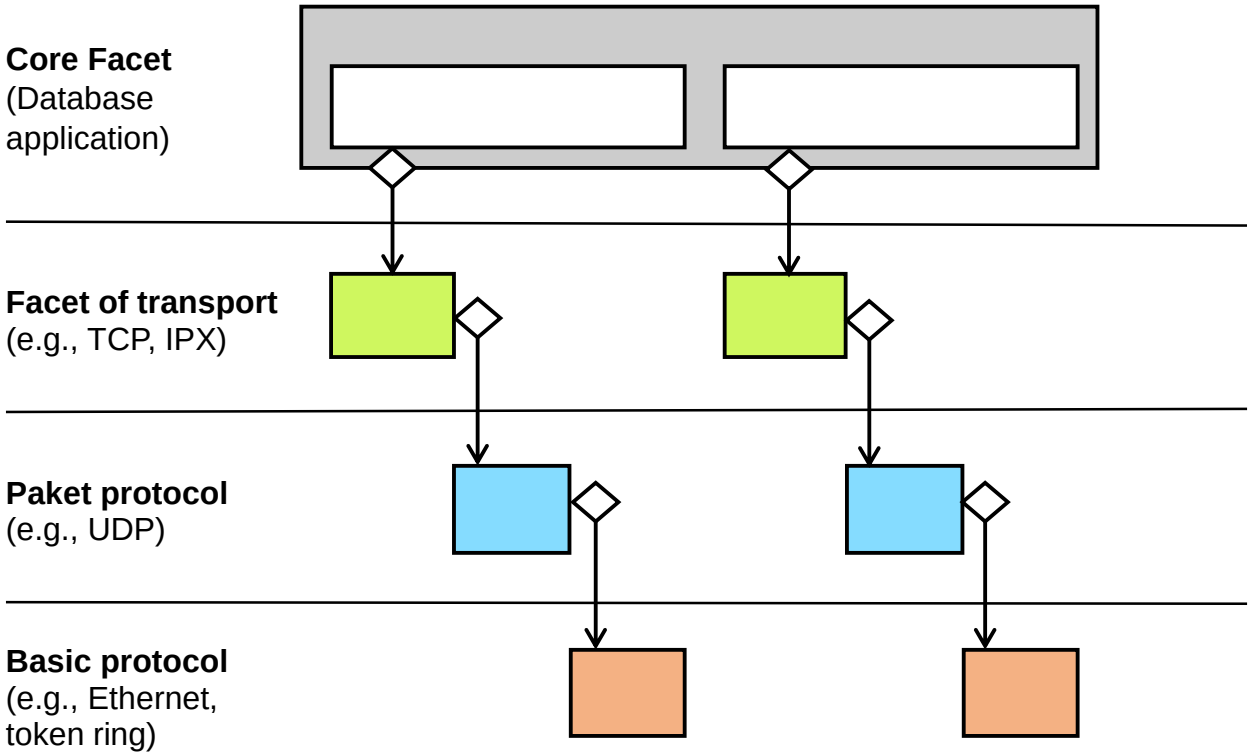
2.5) Layered Frameworks and Systems with Chain Bridges

Layered Frameworks for Layered Systems

- Whenever a system is a layered architecture (stack architecture),
  a *layered (object) framework* can be used
  - Chain-Bridge can implement them, if the layers are independent of each other
    (*layered chain-bridge framework)*
  - The layering is an abstraction layering
    (more detailed things appear as lower layer)
- Modelling criterion: every class must contribute to every layer of a layered system
  - Classes *crosscut the layers*
  - In general, layered system do not meet this criterion
- Different products can be configured easily by varying the dimensions

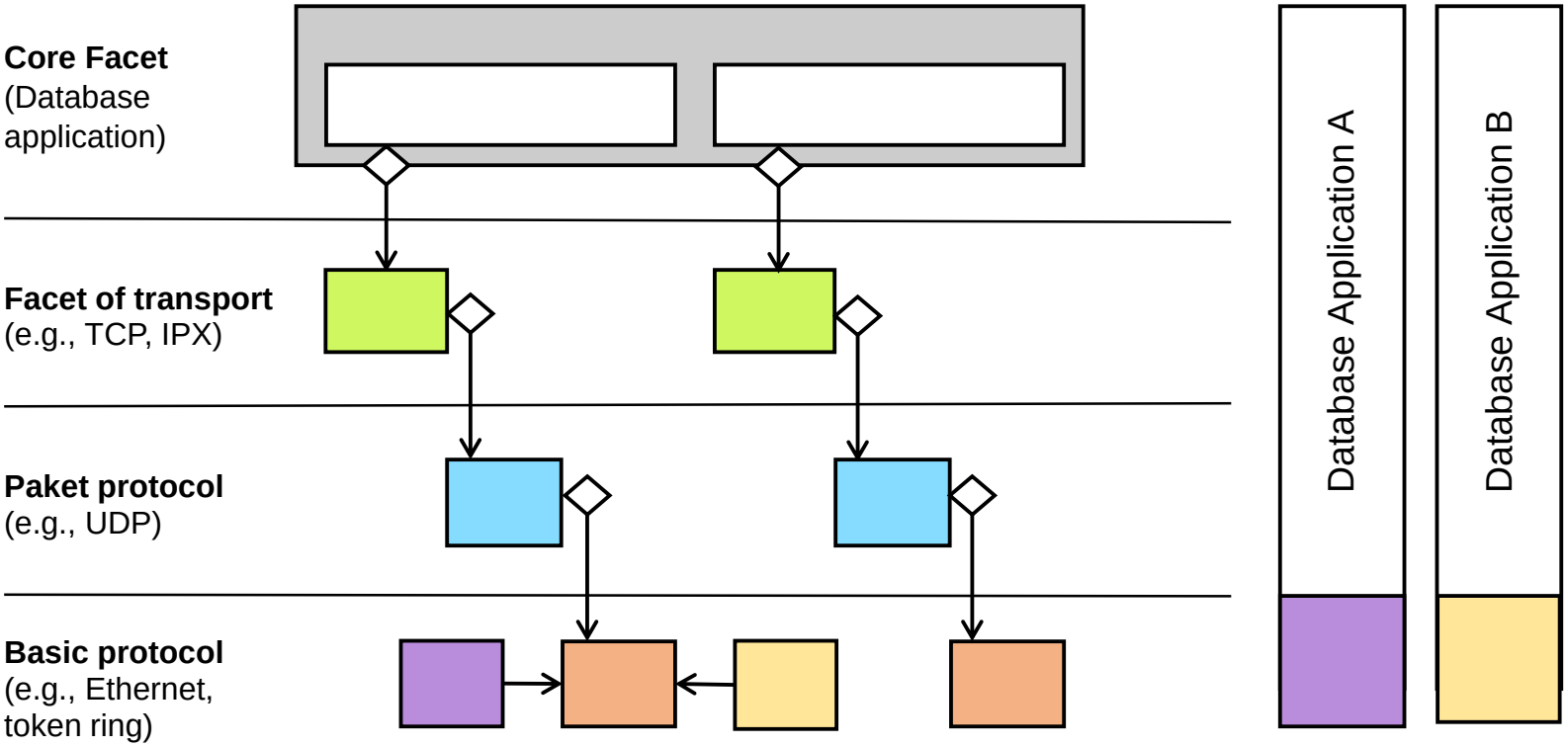
Network Stacks as Layered Bridge System

- ISO/OSI has 7 layers (leads to a 7-Bridge)
- Every layer knows the next underlying
- All partial objects call partial objects in lower layers



Databases and Layered Bridge Frameworks

- An object-oriented database should be a layered bridge framework



The Role of Layered Frameworks

- Layered frameworks are a very important structuring method for large systems
  - Parameterization
  - Variation
  - Extension
- On every layer, reuse is possible
  - Enourmous variability
- Every layer corresponds to an *aspect* of the application
  - All layers form *stacked aspects*
- A large system must be *reducible* or *layered*
  - Hence, layered frameworks provide a wonderful, very general methods for product lines of very large products
  - And additionally, for *extensible* systems

The Role of Layered Frameworks

- Currently, there are three competing implementation technologies
  - Aspect-oriented weaving
  - View-based weaving (hyperslice programming)
    (see Component-Based Software Engineering, summer semester)
  - Hand programming
    - Chain-Bridges
    - Role Object Pattern (see later)
- Layered frameworks are one of *the most important* software engineering concepts

The End

Figures

Slide 6	http://furniture-blog.de/wp-content/uploads/2013/12/Schraube-im-Holz.jpg
Slide 7	https://farm4.staticflickr.com/3804/10033443254_8671330728_k.jpg
Slide 8	http://www.duden.de/_media_/full/S/Schraubenkopf-201020591820.jpg
Slide 9	https://www.flickr.com/photos/eastgermanpics/4381203299