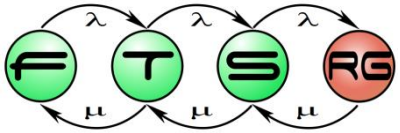
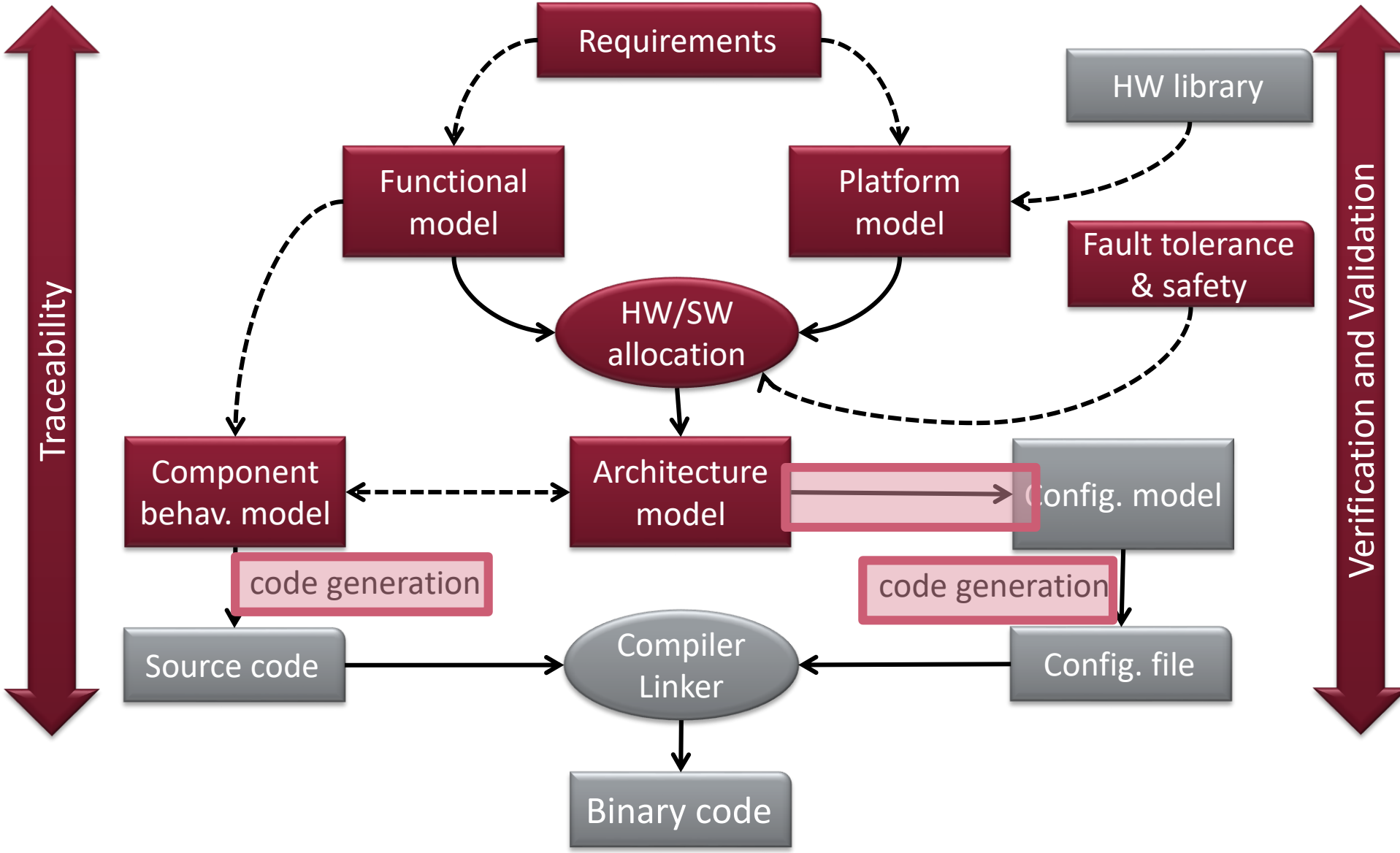
Code generation and model transformation approaches

Systems Engineering BSc Course



Budapest University of Technology and Economics
Department of Measurement and Information Systems

Platform-based systems design



Learning Objectives

Model and code generation approaches

- Brief overview on model transformation approaches
- Overview on code generation concepts
- Summary of currently available technologies

Case-study

- Complex modeling and transformation case study from the avionics domain

Code generation (text synthesis)

Why?

- Let's shorten Development time!
- Use our **models/requirements/plans** to derive…
  - Documentation
  - Source code
  - Configuration descriptors
  - Communication messages
  - Object Serialization
  - …
- Need to support designing „text" synthesis

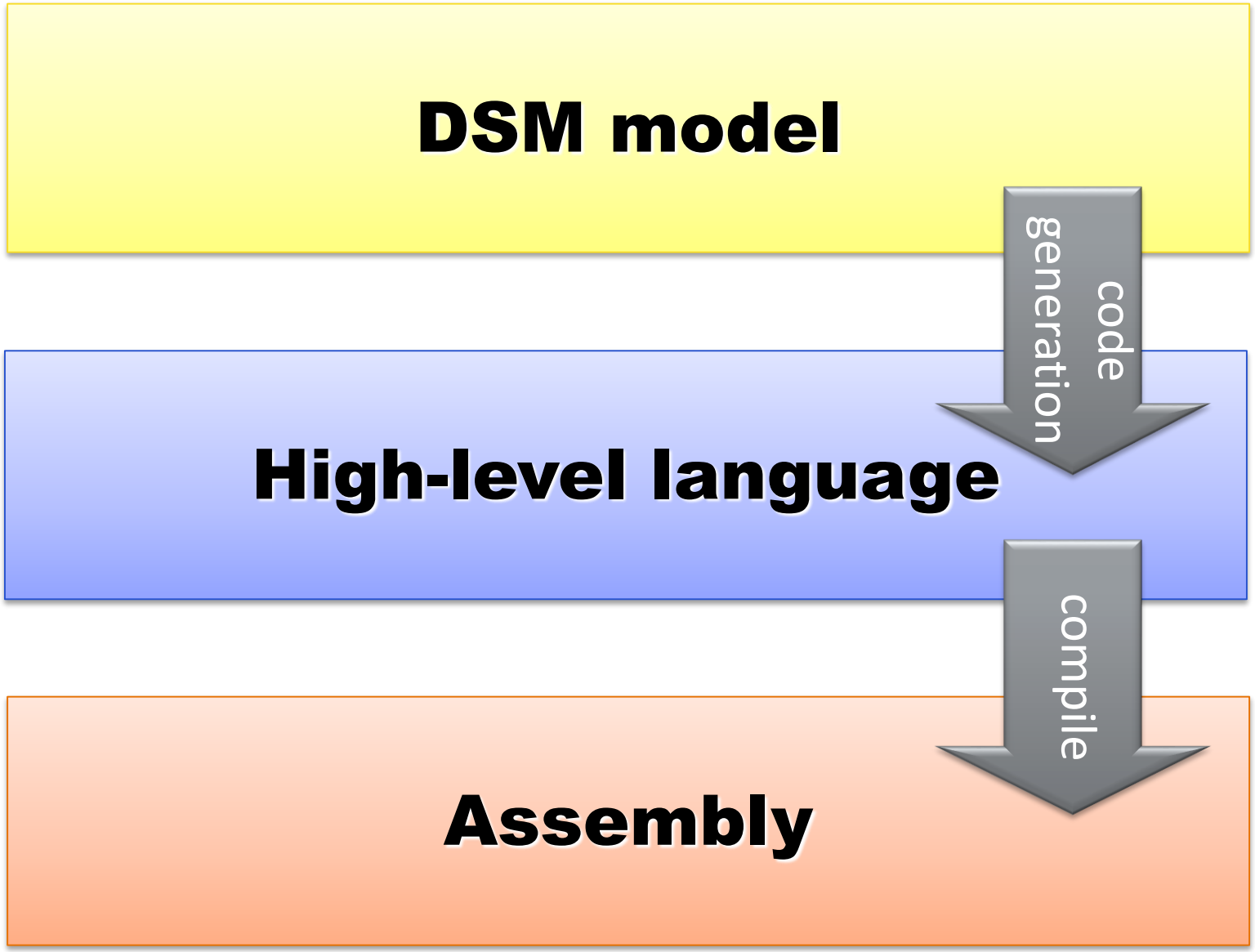
Text synthesis

- The realization of a high-level model on an implementation platform
- A choice between certain attributes – compromise between:
  - Compatibility
  - Performance
  - Maintainability
  - Reusability

Similarity with compilers

- Mapping between abstraction levels
  - e.g., From C to assembly
- Usage of design patterns
  - e.g., function calls in C
- Many similarities, NOT a strict separation
  - pl. C++ templates, automatically generated ctor+dtor
- Prediction:
  - yesterday's design pattern → today's code generation feature → tomorrow's language element
- Domain-specific instead of universal languages

Example: Source Code generation in MDE



Major Approaches

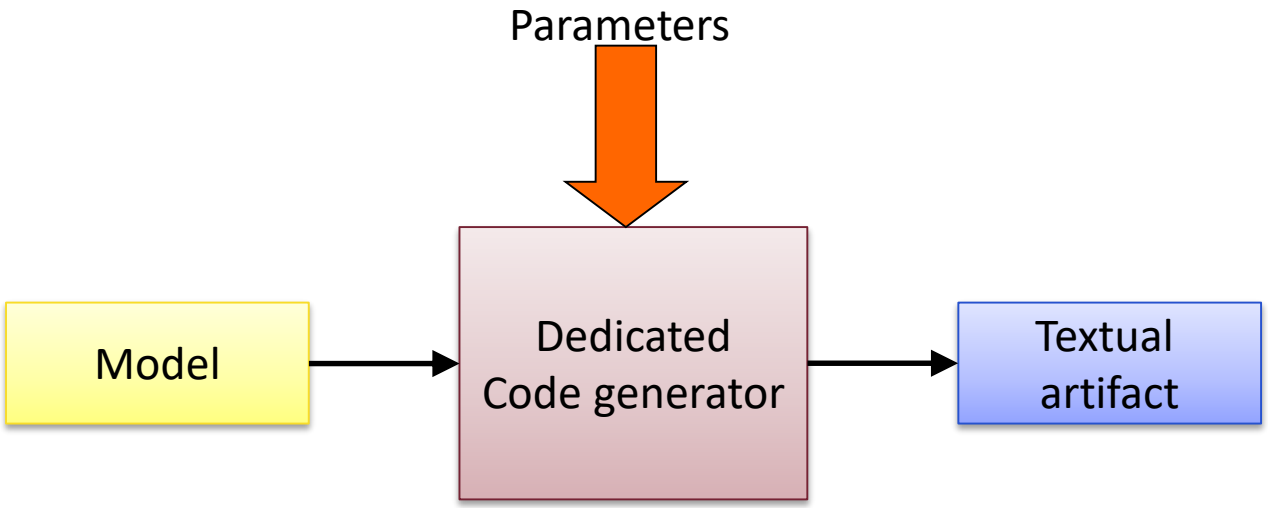
- Dediacated
  - Specific, ad-hoc
  - Using a dedicated code generator
- Template based

Specific, ad-hoc

```
sourceFile.write("   temp = ((AIDA_PARTITION_TYPE*) selfModule.partitions.elements);\n" )
i = 0
for partition in partitions:
 numPorts = getNumberOfAllCommPorts_Partition(currModuleComm, interPartitionComm, partition.partitionName)
 sourceFile.write("   temp[" + str(i) + "].partition_id = " + str(partition.partitionID) + ";\n" )
 sourceFile.write("   strcpy( &temp[" + str(i) + "].partition_name[0], \"" + str(partition.partitionName) + "\");\n")
 sourceFile.write("   temp[" + str(i) + "].ports.type = CONST_AIDA_PORTS_TYPE;\n")
 sourceFile.write("   temp[" + str(i) + "].ports.elements = &mem_ports_" + str(partition.partitionName) + "[0];\n")
 sourceFile.write("   temp[" + str(i) + "].ports.numOfElements = " + str(numPorts) + ";\n")
 sourceFile.write("\n")
 i = i + 1
## end for
sourceFile.write("\n")
```

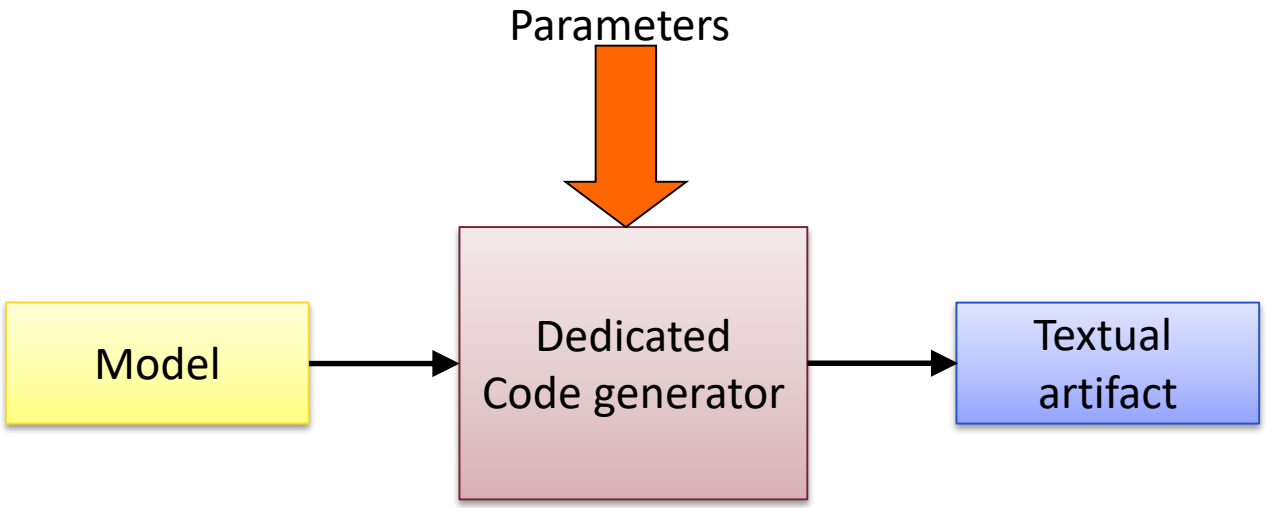
- Designed for the specific problem domain:
  - Best performance
  - Quick and dirty
  - Long development, hard maintainability
  - Zero reusability
  - Dedicated problem domains
    - Minimal changes during support cycle (safety critical embedded system, defense)
    - Certifiability
  - Example:
    - ARINC653 Multistatic configuration generator (python script)

Dedicated code generator



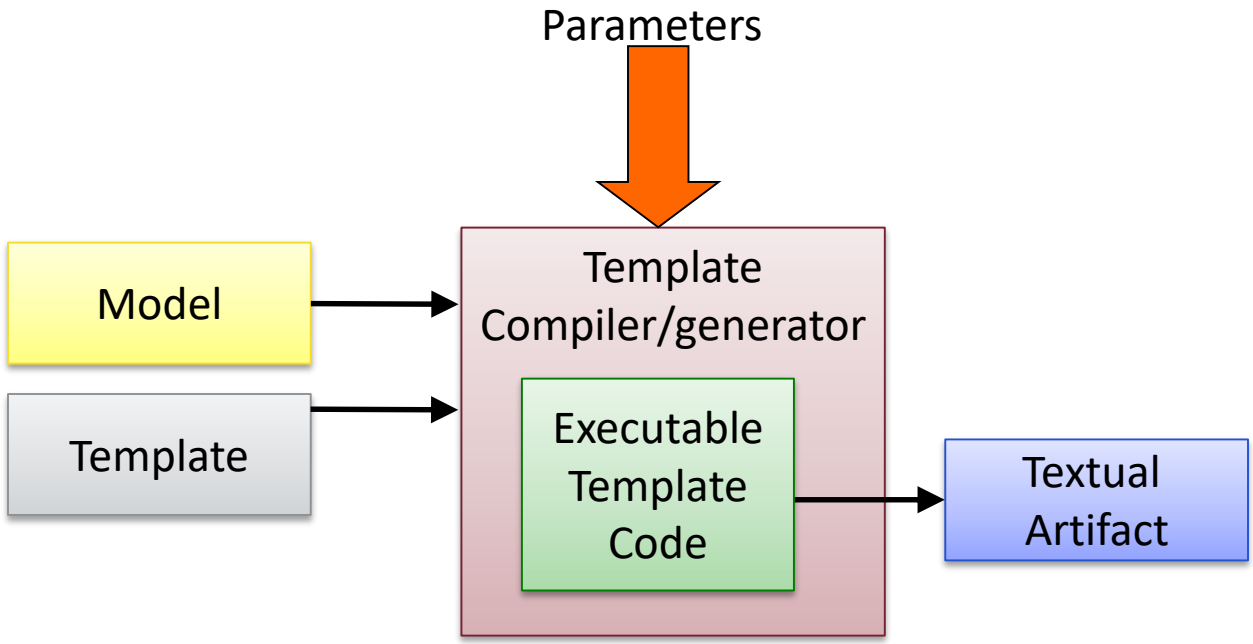
- Based on a framework:
  - Faster development time
  - Slower performance, better reusability
  - Embedded systems, moderate changes during project lifecycle

Dedicated code generator

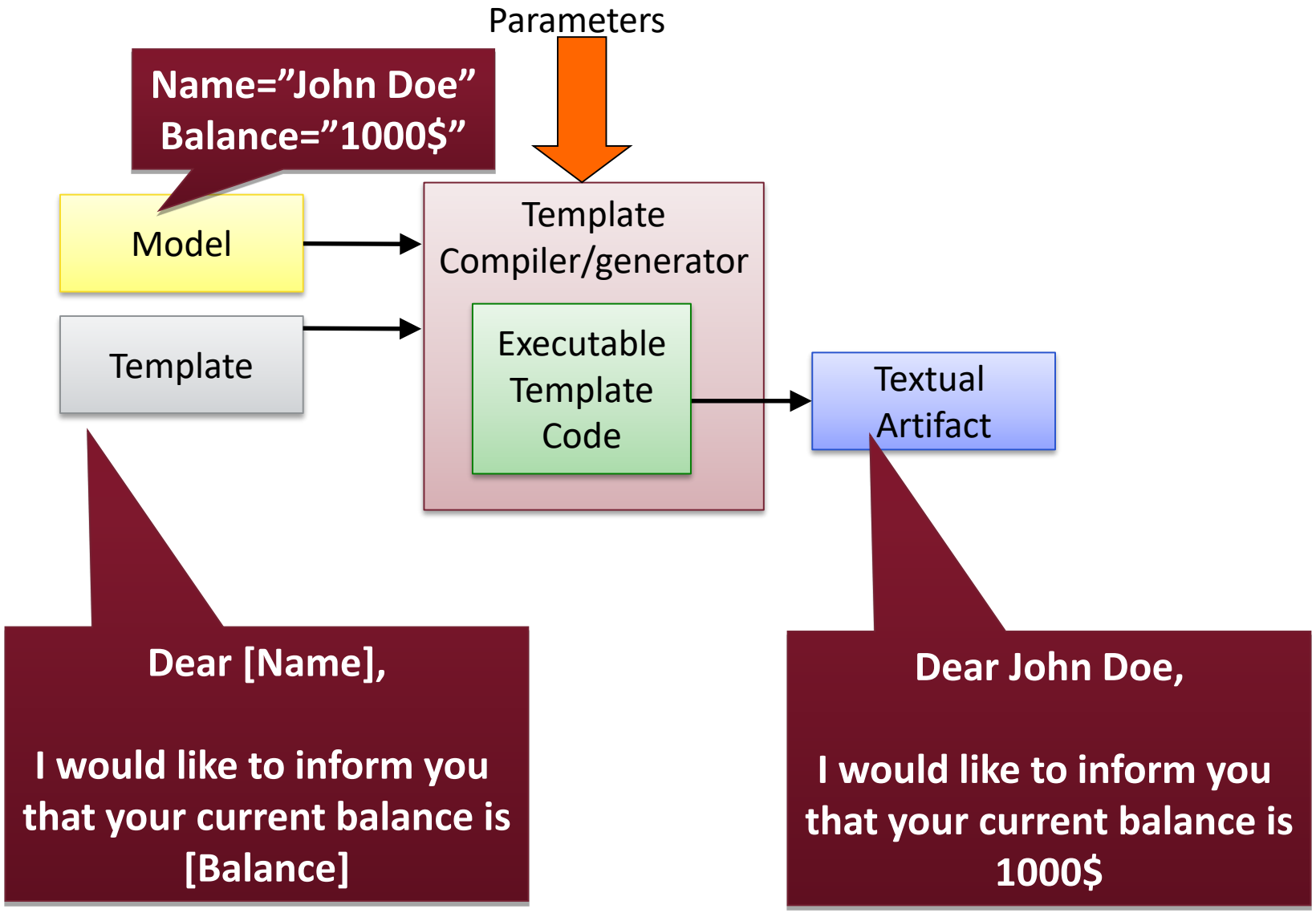


- Examples:
  - IBM Rational Software Architect
  - VASP (DO-178B Level A) Display graphics in avionics
  - Mathworks
  - Matlab Simulink
  - Esterel Scade suite

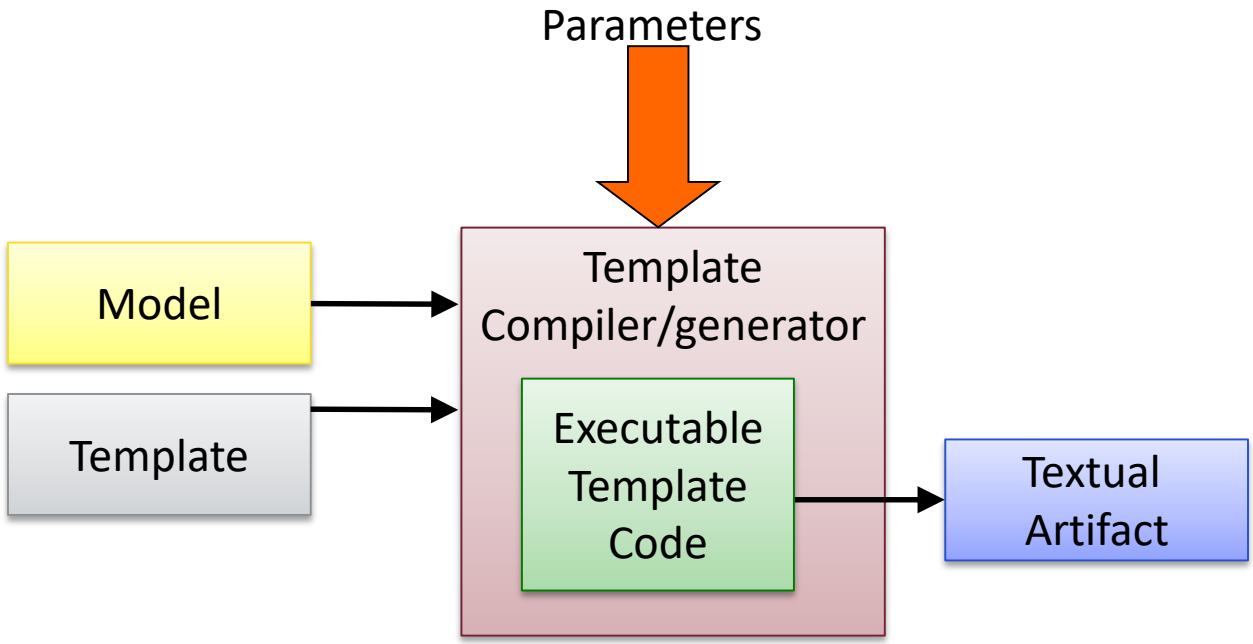
Template based approach



Template based approach

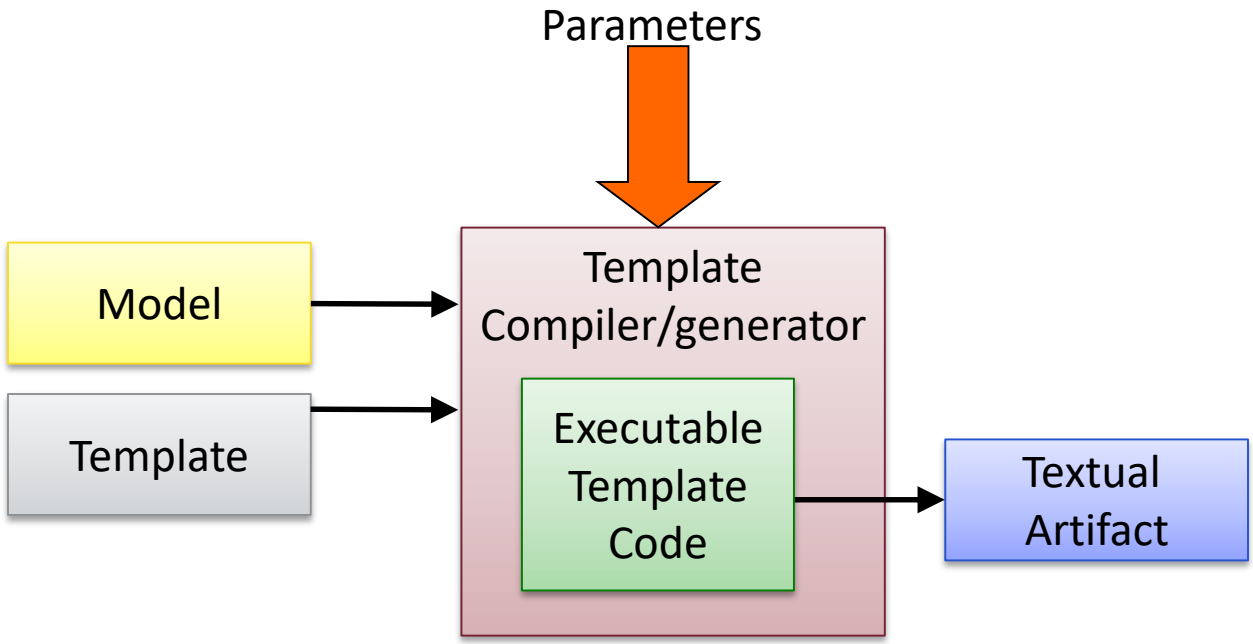


Template based approach



- Fastest development time
- „Slowest” performance, highest reusability
- Fast changing environments (e.g., web based technologies)
- Complex changes during project lifecycle
  - Models and templates can be changed independently

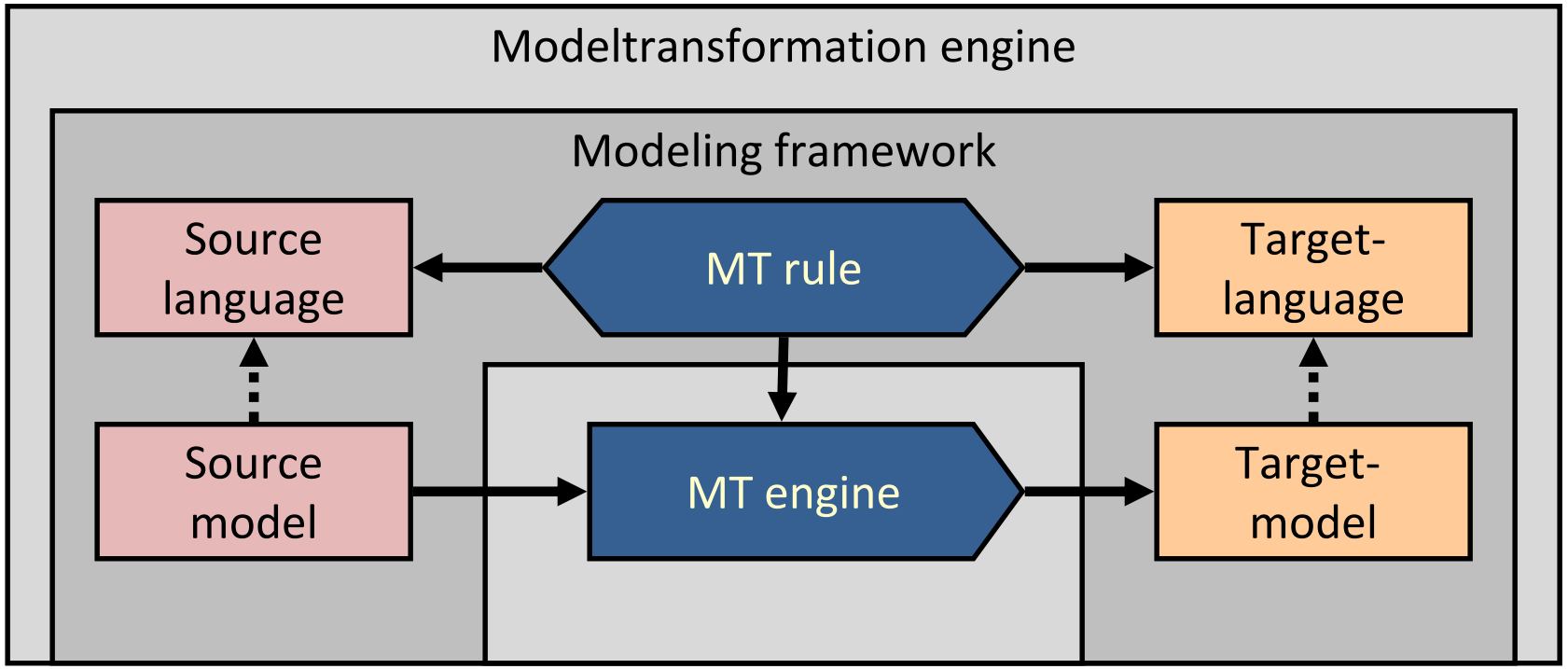
Template based approach



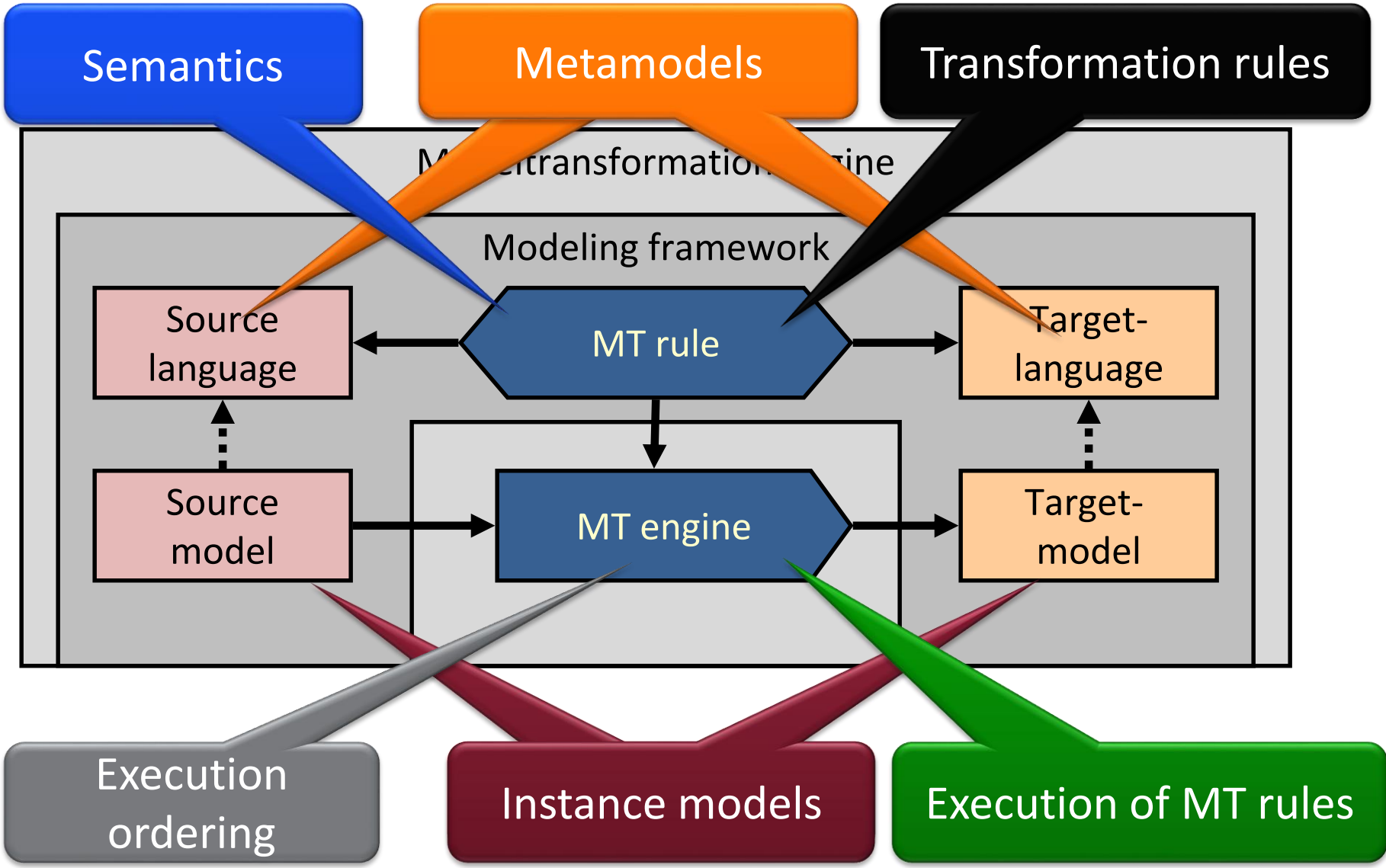
- Examples:
  - JET (for EMF models)
  - Velocity (/JSP)
  - Xtend, Acceleo (MDE approach in Eclipse)
  - AutoFilter (Kalman filters)
  - Smarty (php)

Model Transformation

Definition of Model Transformation



Overview

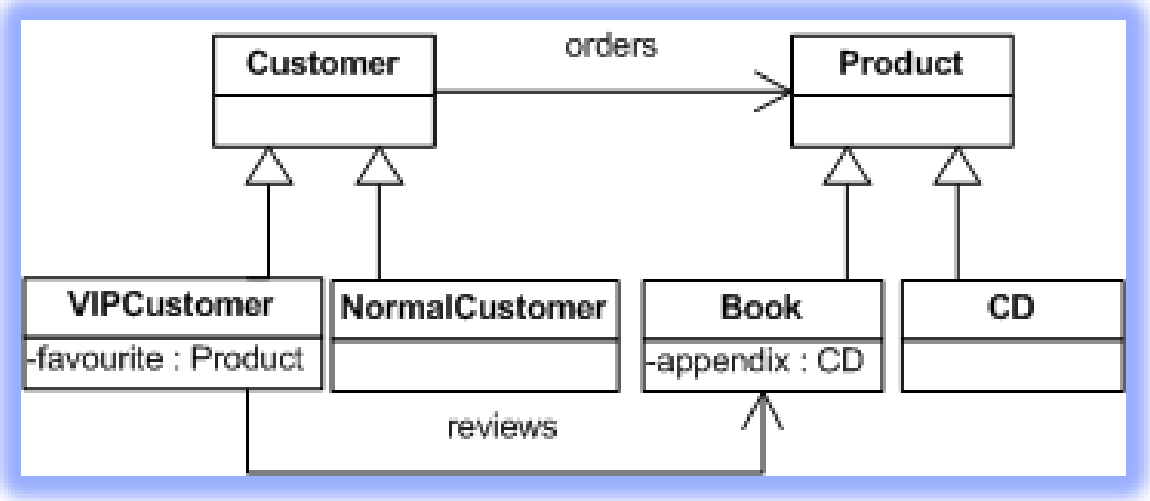

Semantics
Metamodels
Transformation rules
Modeling framework
Source language
MT rule
Target-language
Source model
MT engine
Target-model
Execution ordering
Instance models
Execution of MT rules

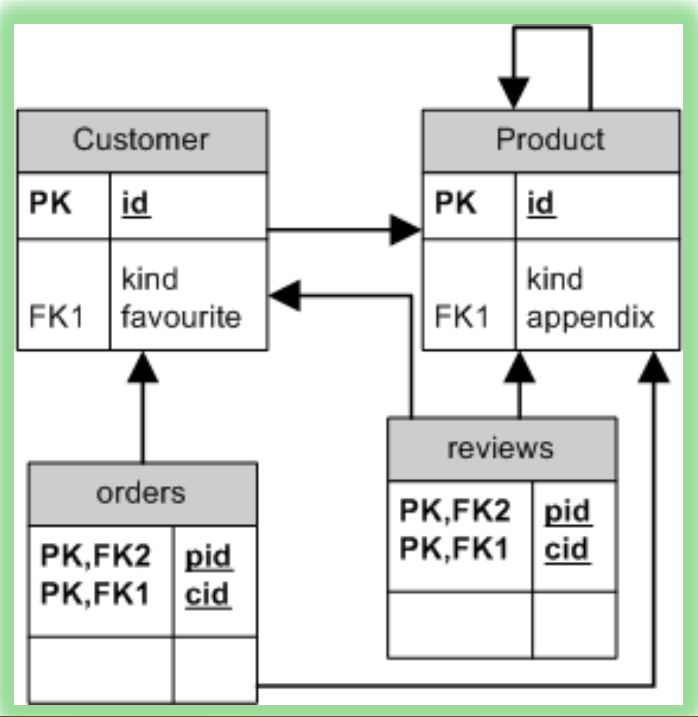
1. Motivating Example

Object Relational Schema mapping

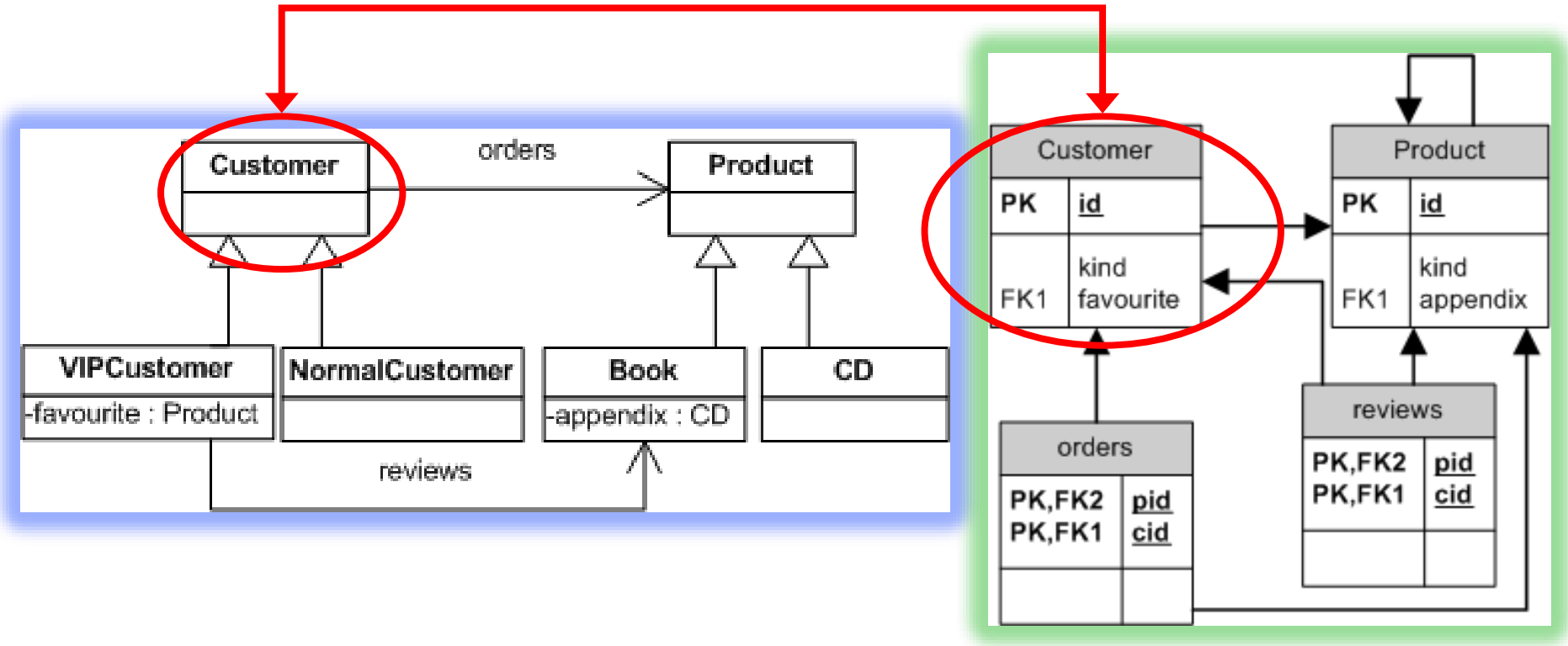
Example: Object-relational maping

- Important as:
  - Model transformation benchmark
  - Most widely used industrial model transformation (pl. Hibernate, EJB, CDO)
- Objective:
  - **Input**: UML class diagram
  - **Output** Relational database schema





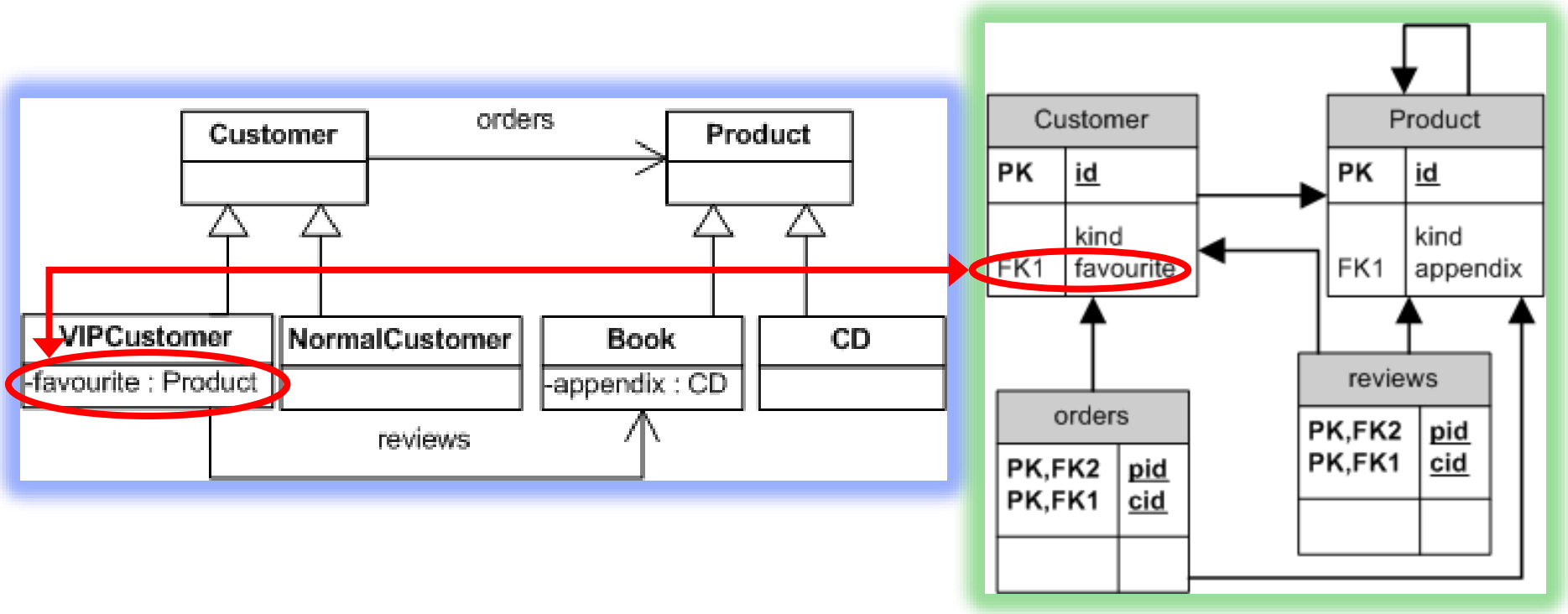
Informal definition of the MT rules of the mapping



Topmost (generalization) classes ➔ Database table + 2 column:

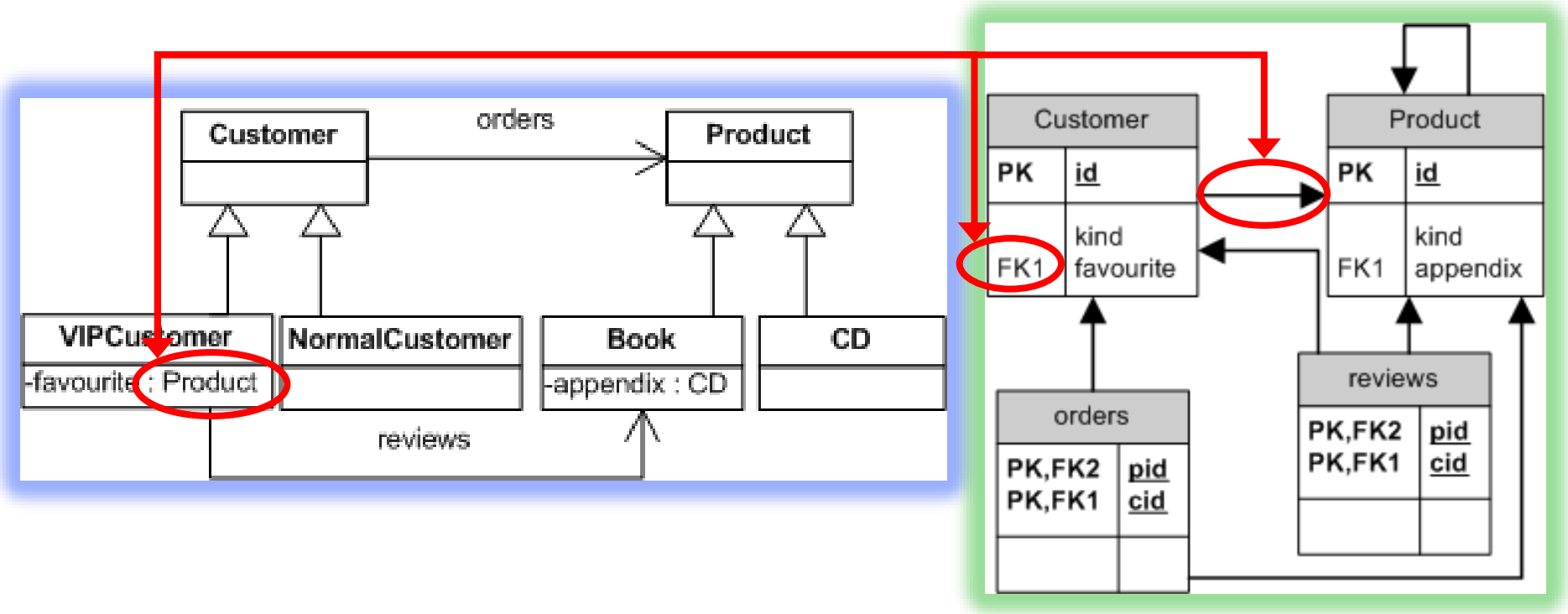
- Unique identifier (primary key),
- type definition

Informal definition of the MT rules of the mapping



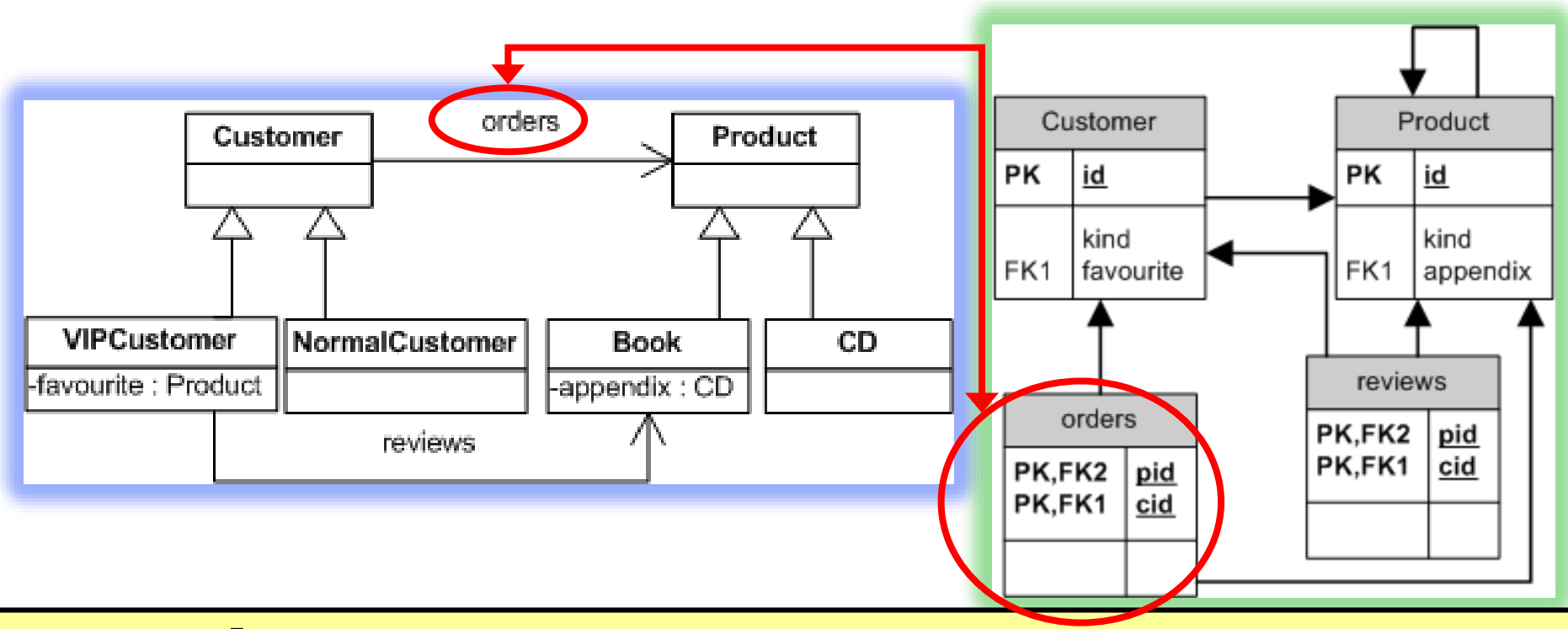
Class attributes ➔ (contained by the topmost classes) Column of the table

Informal definition of the MT rules of the mapping



Type of the attributes ➔ foreign key

Informal definition of the MT rules of the mapping



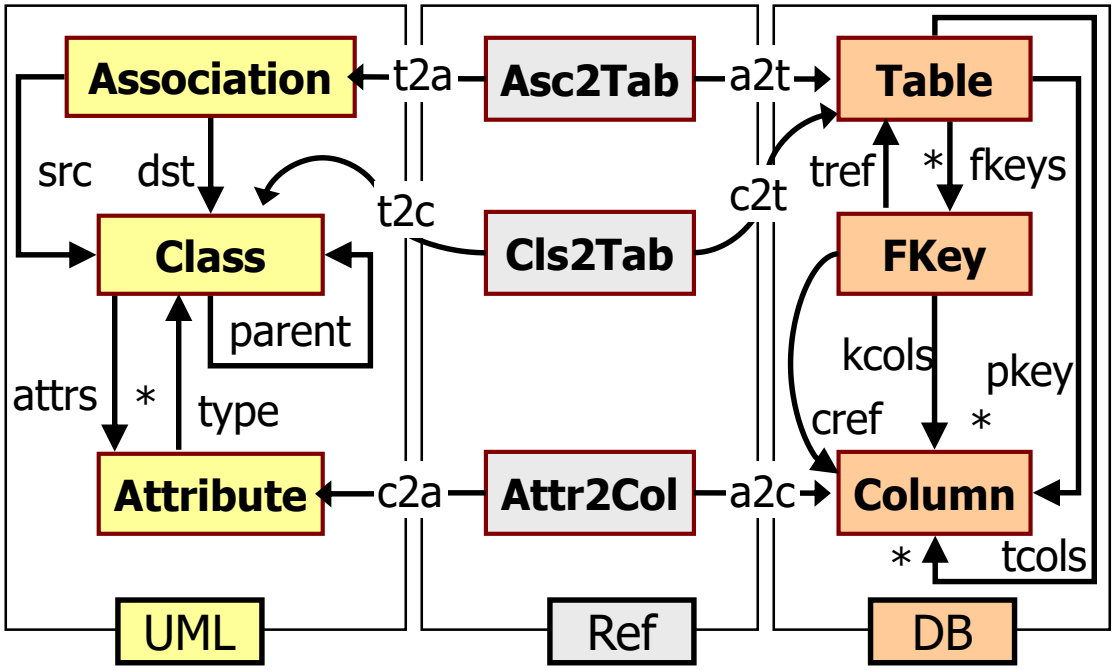
Association ➔ A table with two columns

- source and target identifiers
- foreign keys (for consistency)

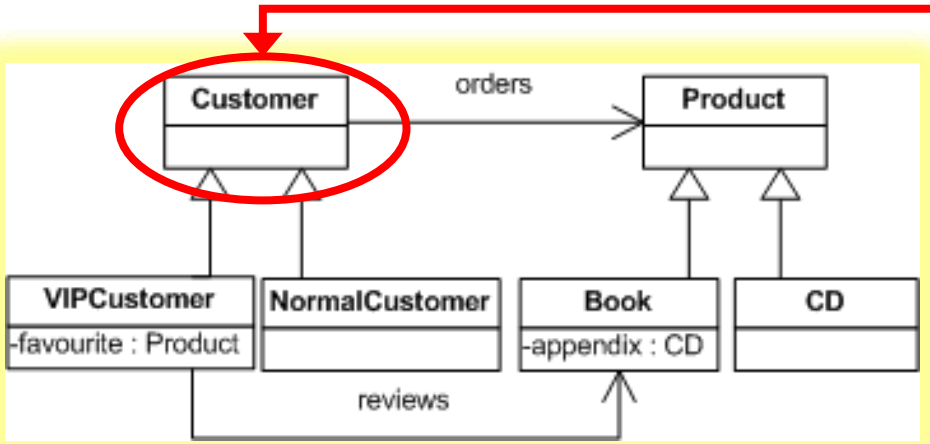
2. Structure of Modeling Languages

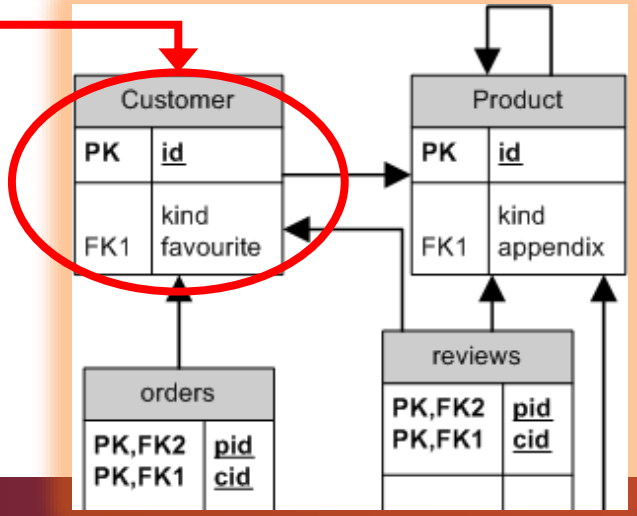
Revision

Metamodel of the O-R mapping



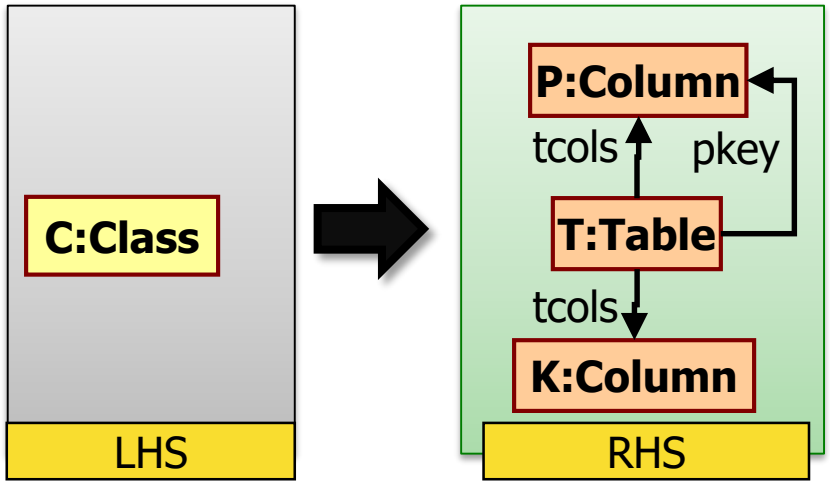
- Source + Target metamodel
- Traceability metamodel:
  - For saving the relations between the source and the target languages
- Motivation: critical embedded systems
  - Traceability
  - Requirement ➔ Source code





3. Graph Transformation Rules

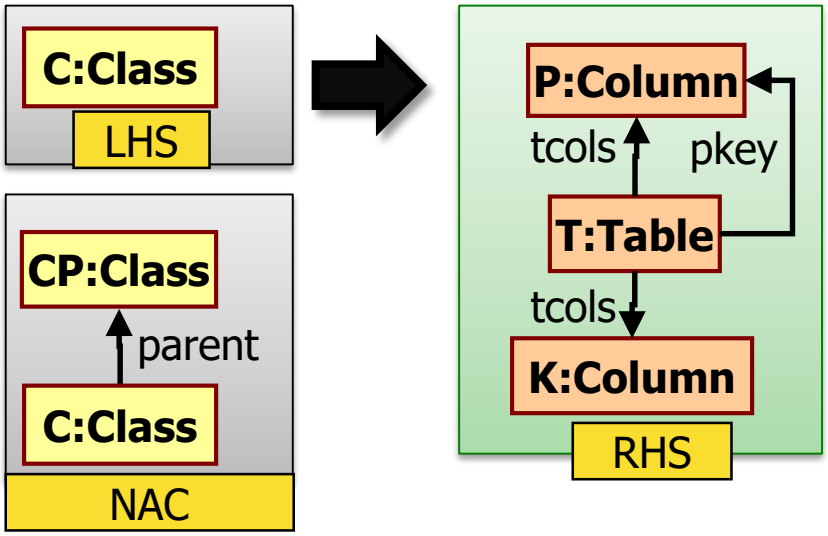
Structure of a GT rule



- **Graph Transformation (GT)**:
  - Declarative and formal paradigm
  - Rule base transformation
  - Match of the LHS➔ match of the RHS
  - Generalization of Chomsky grammars (hierarchy) (text ➔ graph)

- **Graph Transformation Rules**
  - **Left hand side** - LHS
    - Graph pattern
    - Precondition for the rule application
  - **Right hand side** - RHS:
    - Graph pattern + LHS mapping
    - Declarative definition of the rule application
      - What we get (and not how we get it)

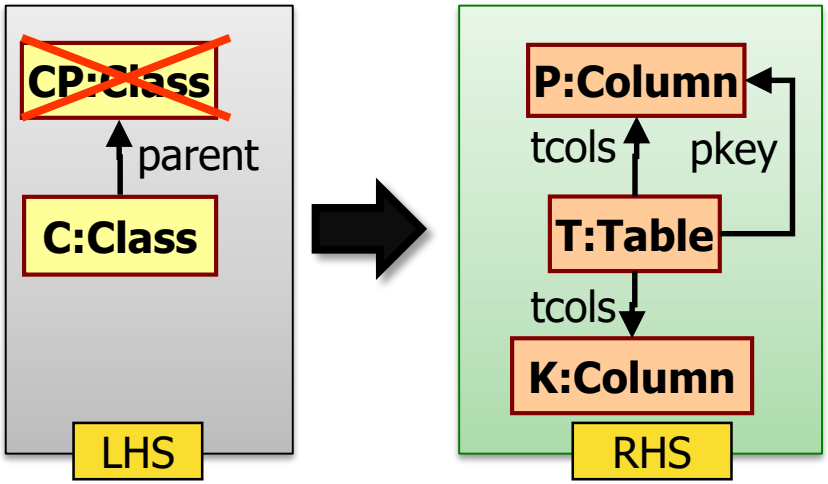
Structure of a GT rule



- **Graph Transformation (GT)**:
  - Declarative and formal paradigm
  - Rule base transformation
  - Match of the LHS➔ Image of the RHS
  - Generalization of Chomsky grammars (hierarchy) (text ➔ graph)

- **Graph Transformation Rules**
  - **Left hand side** - LHS
    - Graph pattern
    - Precondition for the rule application
  - **Right hand side** - RHS:
    - Graph pattern + LHS mapping
    - Declarative definition of the rule application
      - What we get (and not how we get it)
  - **Negative Application Condition**(NAC):
    - Graph pattern + LHS mapping
    - Negative precondition of the rule application
    - If it can be made true➔ the rule cannot be applied
    - Multiple NACs ➔ only one is true ➔ rule cannot be applied

Structure of a GT rule

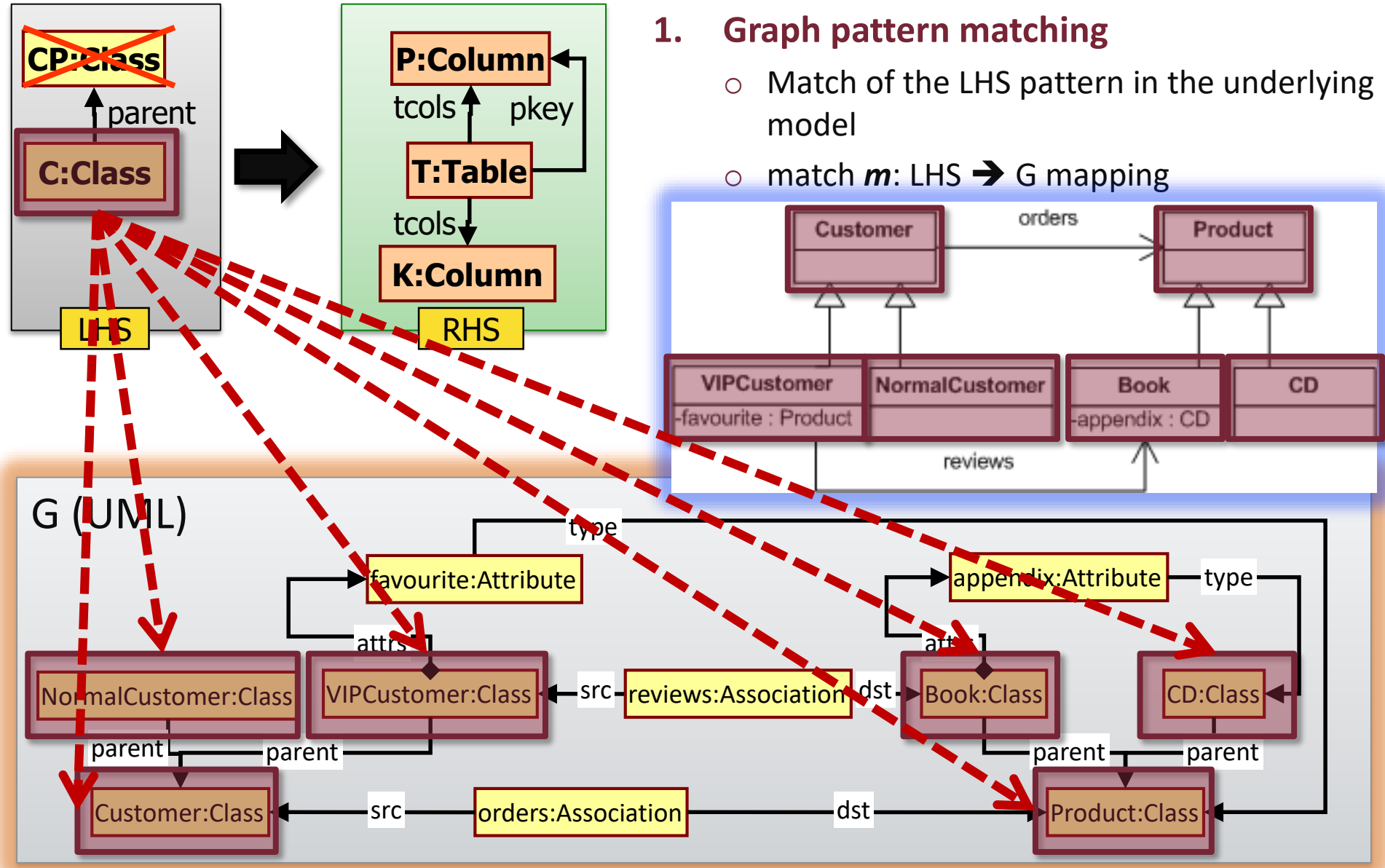


- **Graph Transformation (GT)**:
  - Declarative and formal paradigm
  - Rule base transformation
  - Match of the LHS➔ Image of the RHS
  - Generalization of Chomsky grammars (hierarchy) (text ➔ graph)

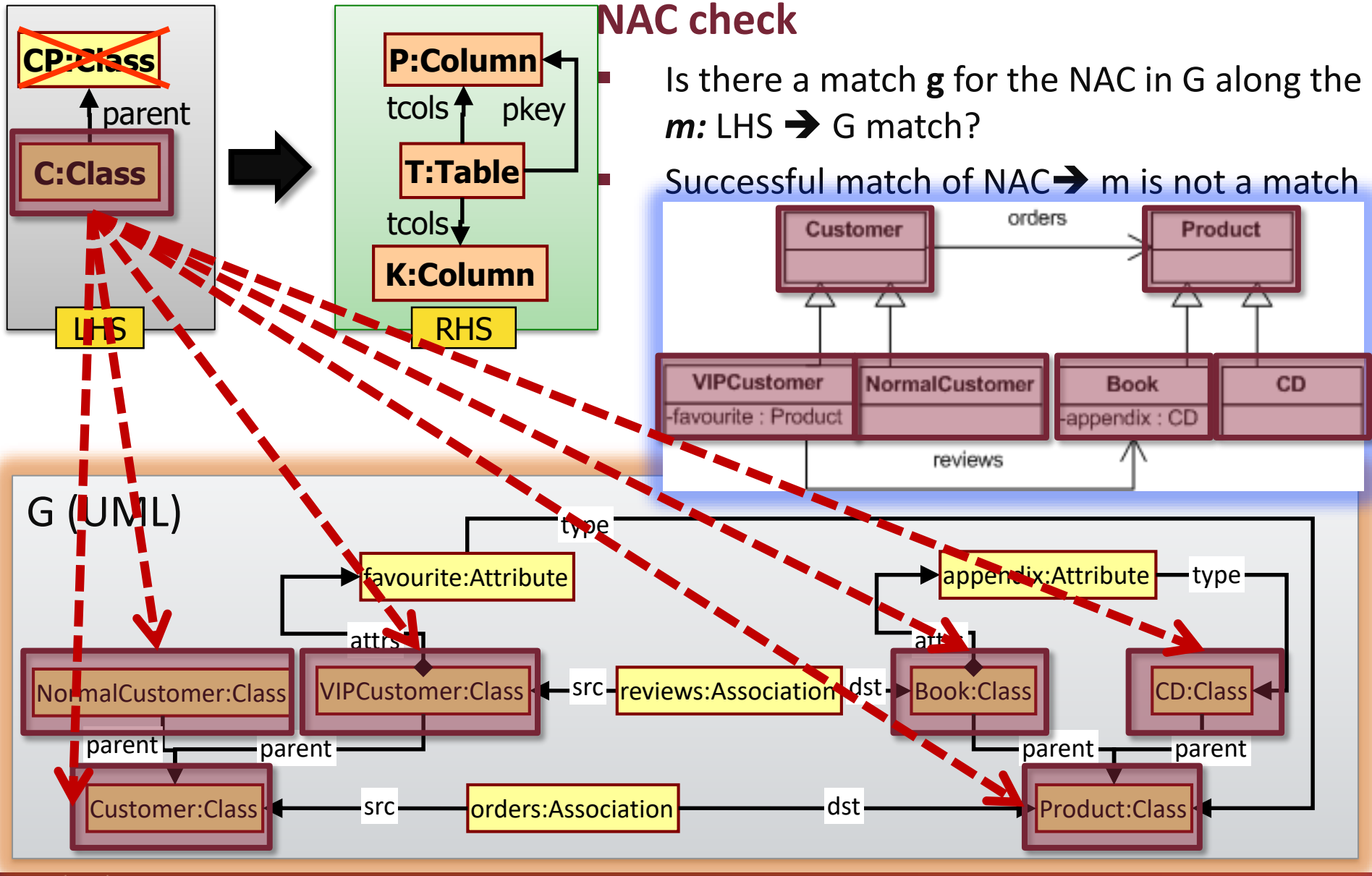
- **Graph Transformation Rules**
  - **Left hand side** - LHS
    - Graph pattern
    - Precondition for the rule application
  - **Right hand side** - RHS:
    - Graph pattern + LHS mapping
    - Declarative definition of the rule application
      - What we get (and not how we get it)
  - **Negative Application Condition**(NAC):
    - Graph pattern + LHS mapping
    - Negative precondition of the rule application
    - If it can be made true➔ the rule cannot be applied
    - Multiple NACs ➔ only one is true ➔ rule cannot be applied

4. Application of Graph Transformation Rules

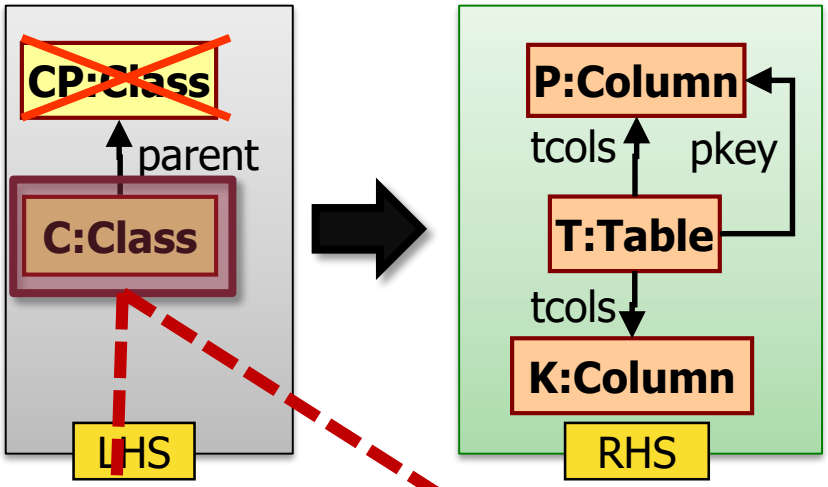
Application of GT rules



Application of GT rules

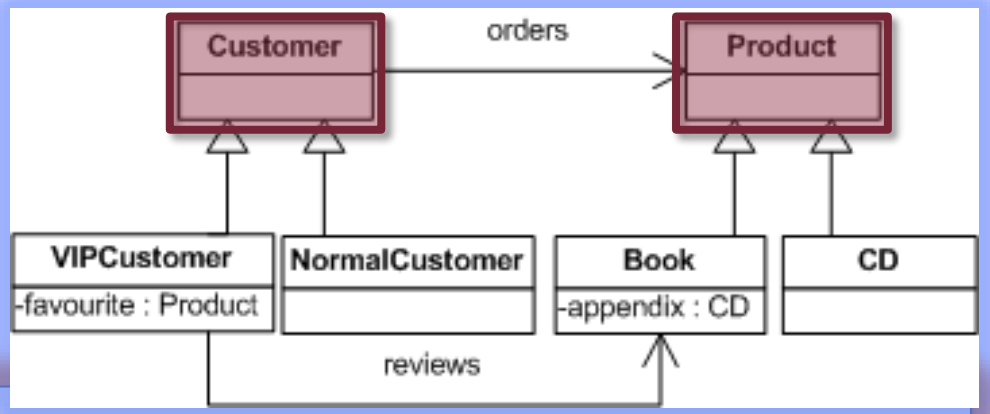


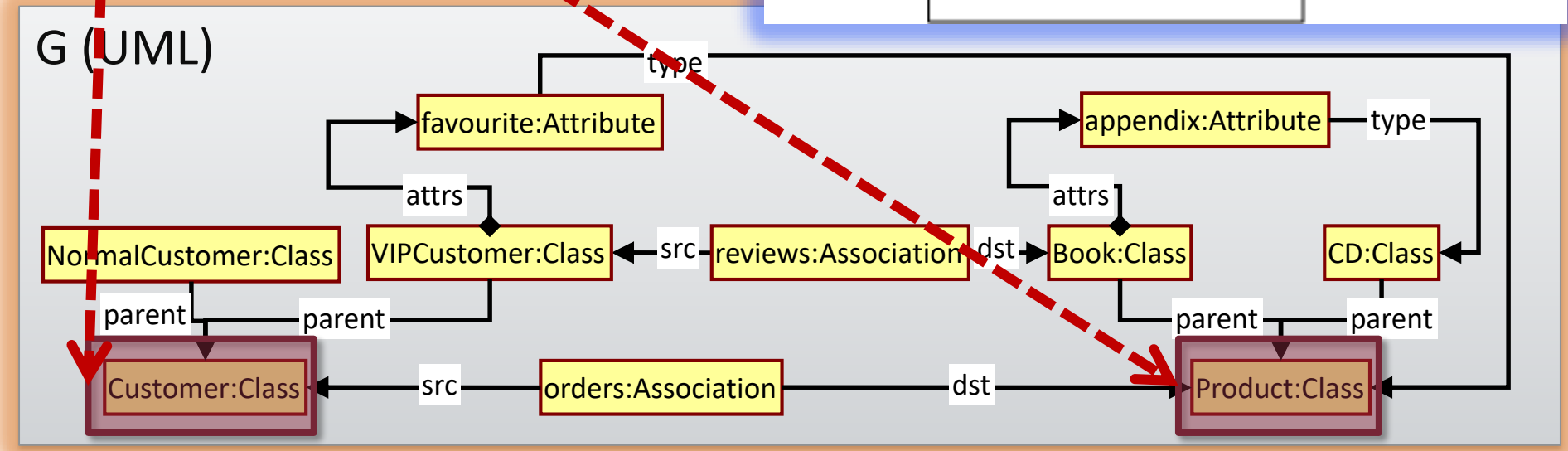
Application of GT rules



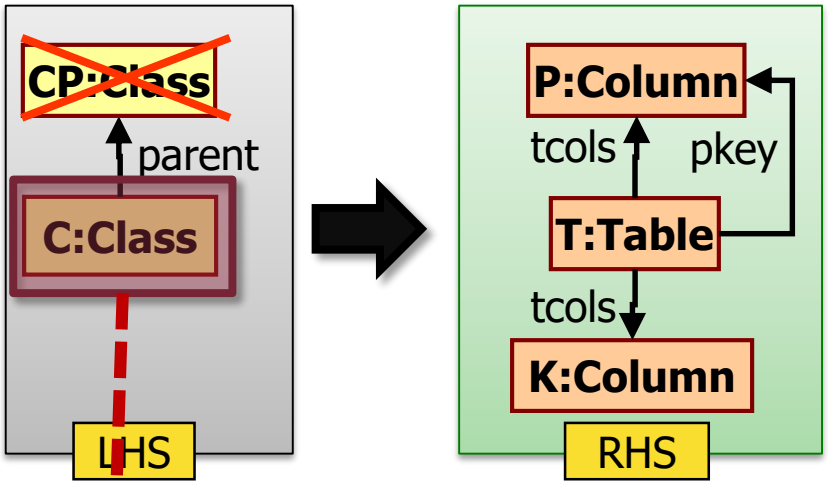
3. Non-deteministic selection

- Random selection of a match (if more than one)
- No match➔ rule fails



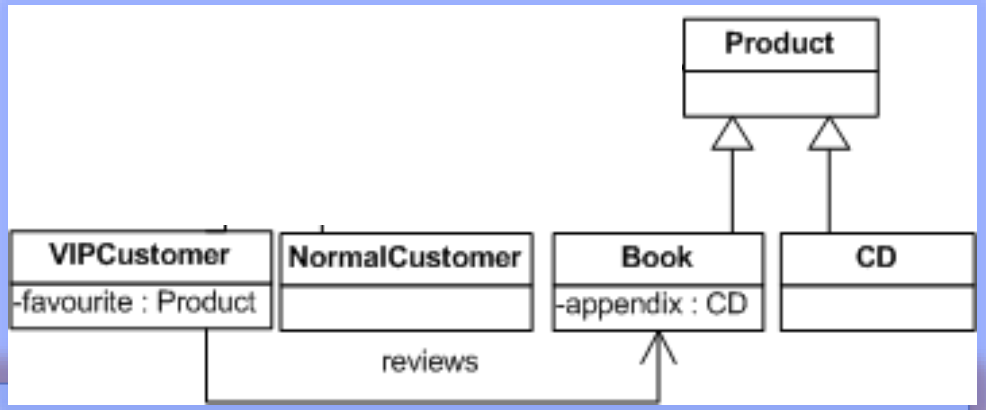


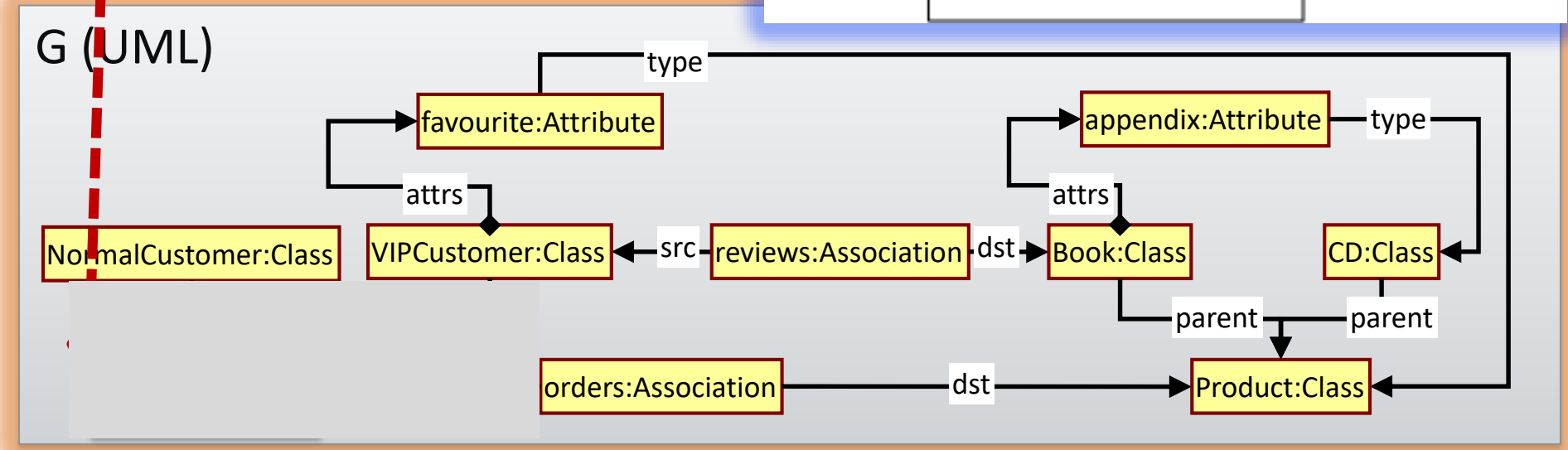
Application of GT rules



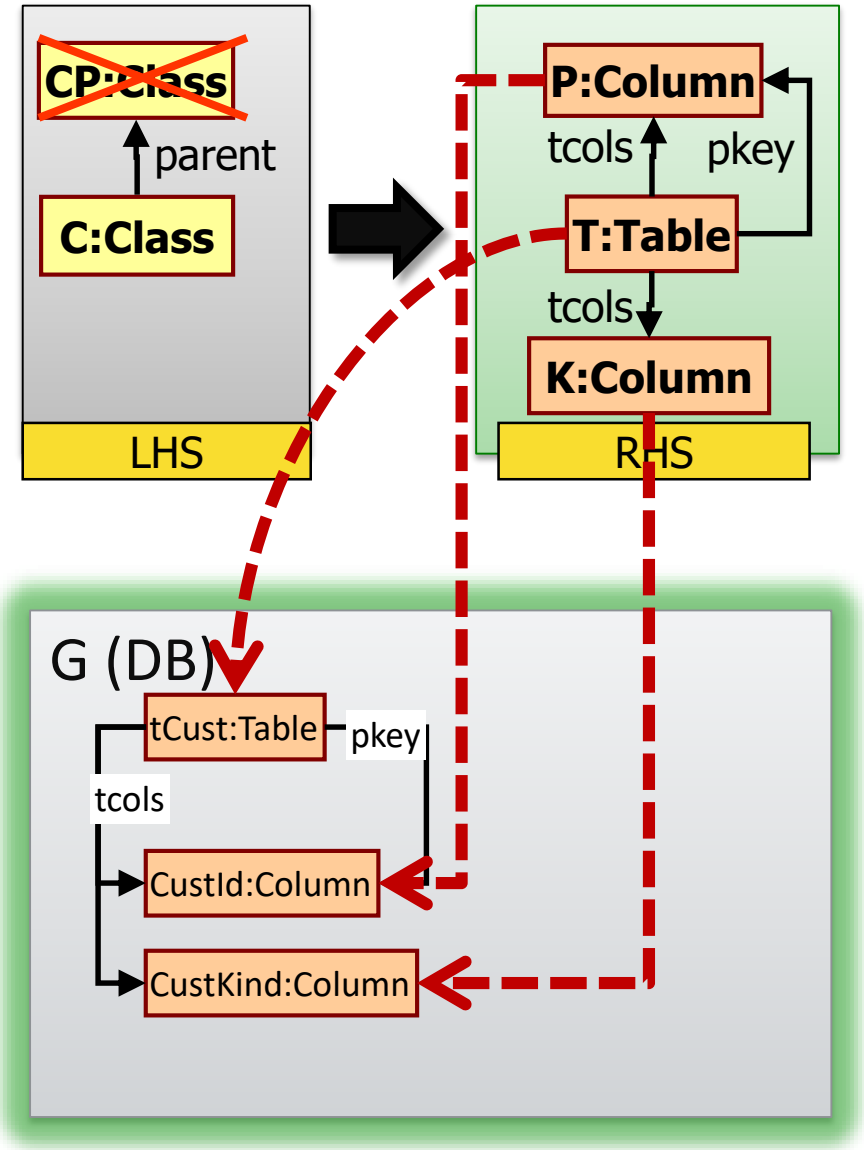
4. Deletion

- Deletion of LHS \ RHS from G
- In LHS yes, in RHS no





Application of GT rules



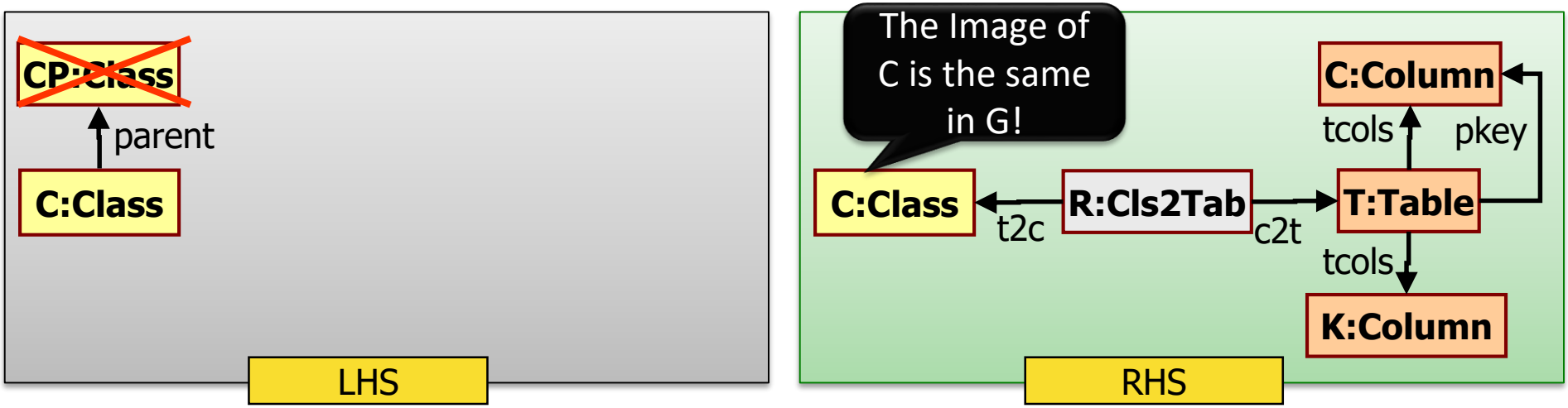
5. Creation (and binding)

- Creation of RHS \ LHS in G with their corresponding relations
- Output:
  a „match” of RHS in G

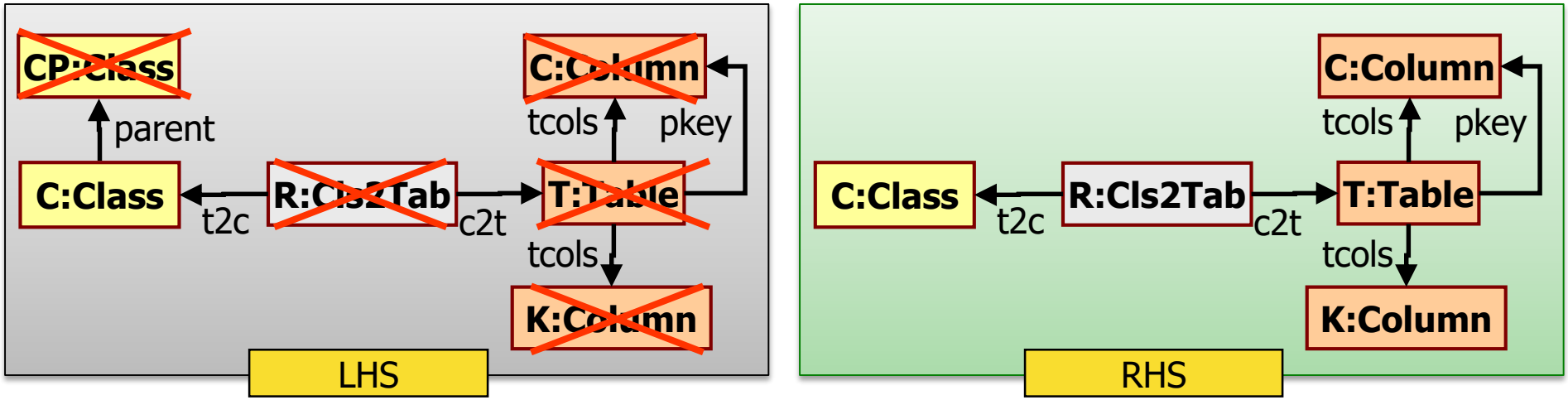
Customer	
PK	id
	kind

Typical problems…

1) Saving the source model, traceability



2) Application of the same rule along the same match



Model transformation approaches

MT: categories

- Model-to-Code (M2C) → ☺
  - Text generation
  - AST generation → special case of M2M
  - Ad-hoc, dedicated, template based, etc.
- Model-to-Model (M2M)
  - Between models
    - Intra-domain transformation
      (e.g., simulation, refactoring, validation)
    - Inter-domain transformation
      (PIM-to-PSM mapping, model analysis)
  - Bridging semantical gaps

Model Transformation approaches

- Direct Model Manipulation
- Relational
- Graph Transformation based
- Hybrid
- Other

Direct Model Manipulation

- Models stored in a Model Space
- Manipulation through API
- Queries hand coded

- Examples:
  - Base EMF
  - Jamda
  - SiTra

Relational Approaches

- Based on mathematical relations
  - Defined as constraints
  - Constraint logic programming
- Queries captured as constraints
- Model manipulation handled by *labeling*
- Fully declarative definition

- Example:
  - QVT

Graph Transformation based

- Model are graphs → use Graph Transformation
- Declarative definition
- Precise formal semantics
- Queries as graph patterns
- Model manipulation as graph transformation rules

- Examples:
  - AGG
  - GreAT
  - ATOM
  - GrGen.Net

Hybrid approaches

- Combines declarative and imperative definition
- ”Developer friendly”
- Typically
  - Queries → declarative
  - Control Structure → imperative
- Complex language
- Largest transformations are using this approach

- Example:
  - ATL
  - Viatra

Other - XSLT

- Models as XMI files
- Model Transformation as XSLT programs
- Hard to maintain
- XMI representations are
  - verbose
  - poor readability

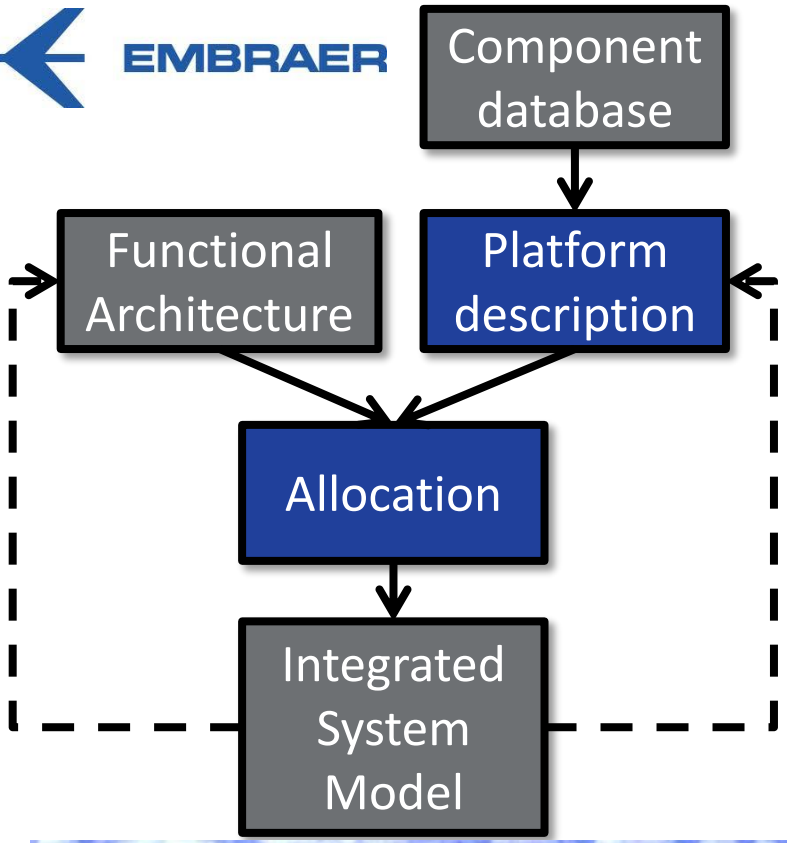
Model driven development of ARINC653 configuration tables

A case study

Recent Project

Goal: Allocate SW components to ARINC653 compliant IMA platform

EMBRAER



DECOS

indexys

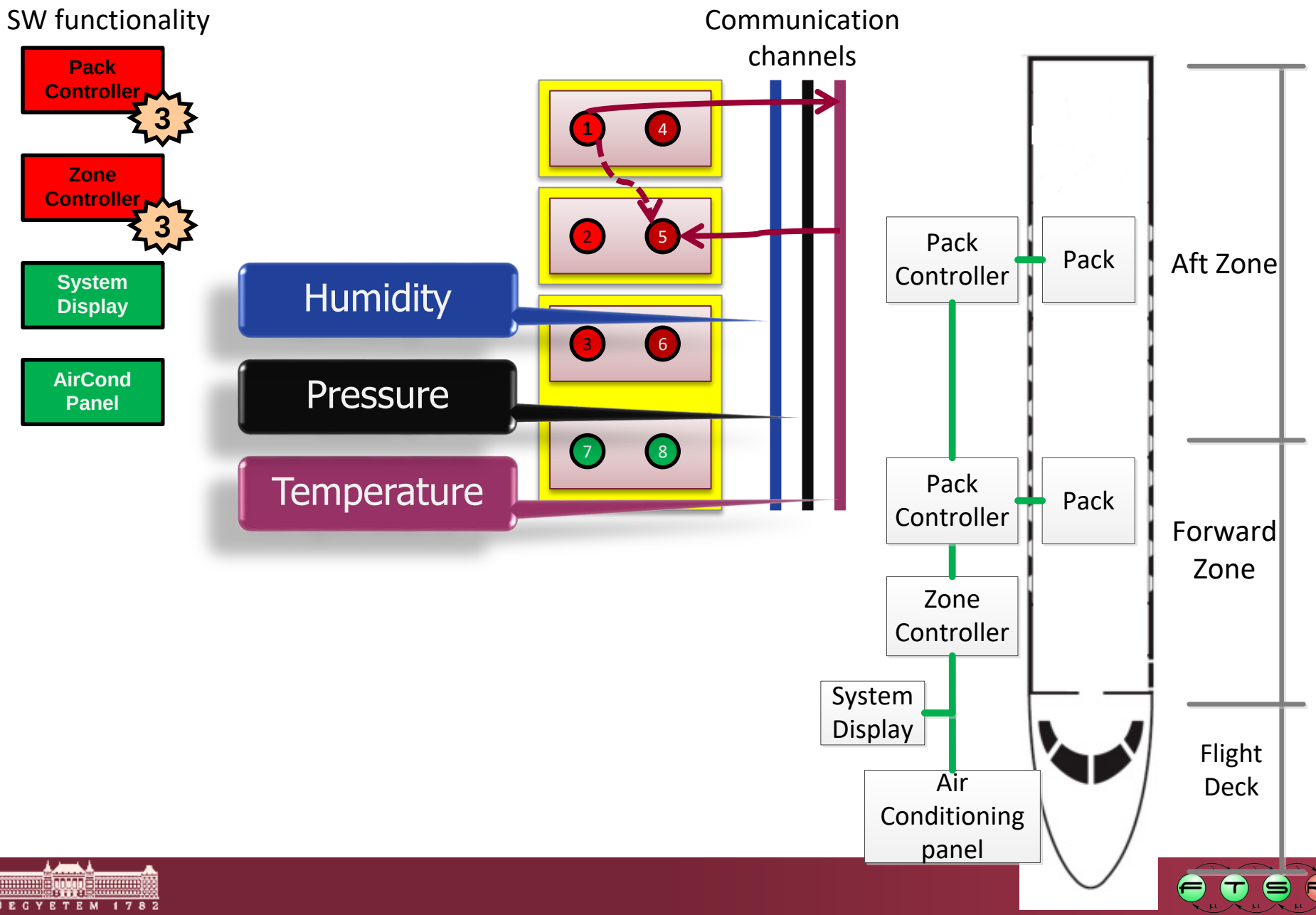
MOGENTES

DIANA

SENSORIA

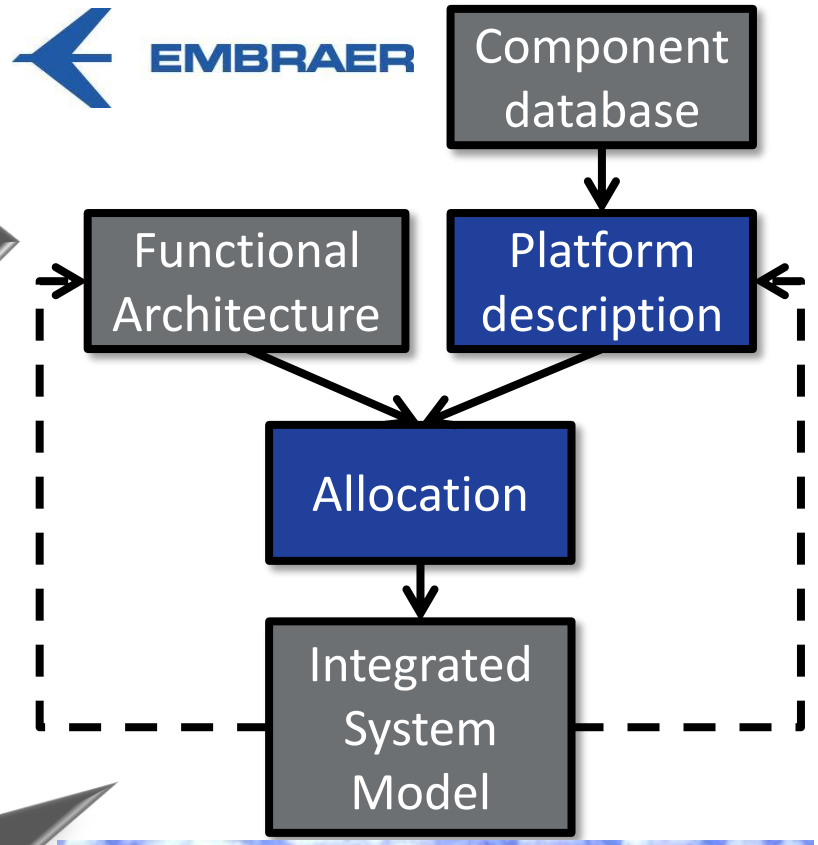
secure CHANGE

Allocating communication channels



Model Driven Development of IMA Configs

Inputs:
- Platform Independent Model (PIM) (functional + nonfunc. reqs; Simulink)
- Platform Description Model (PDM) for ARINC 653 (DSML)

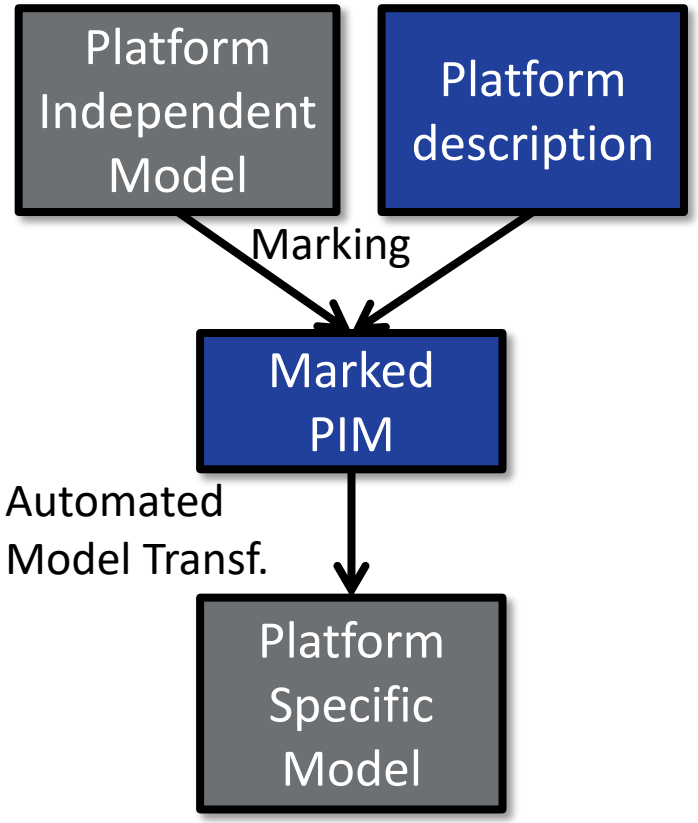


Output:
- Integrated system model
- Ready for simulation
- End-to-end traceability

Traditional MDA Theory?

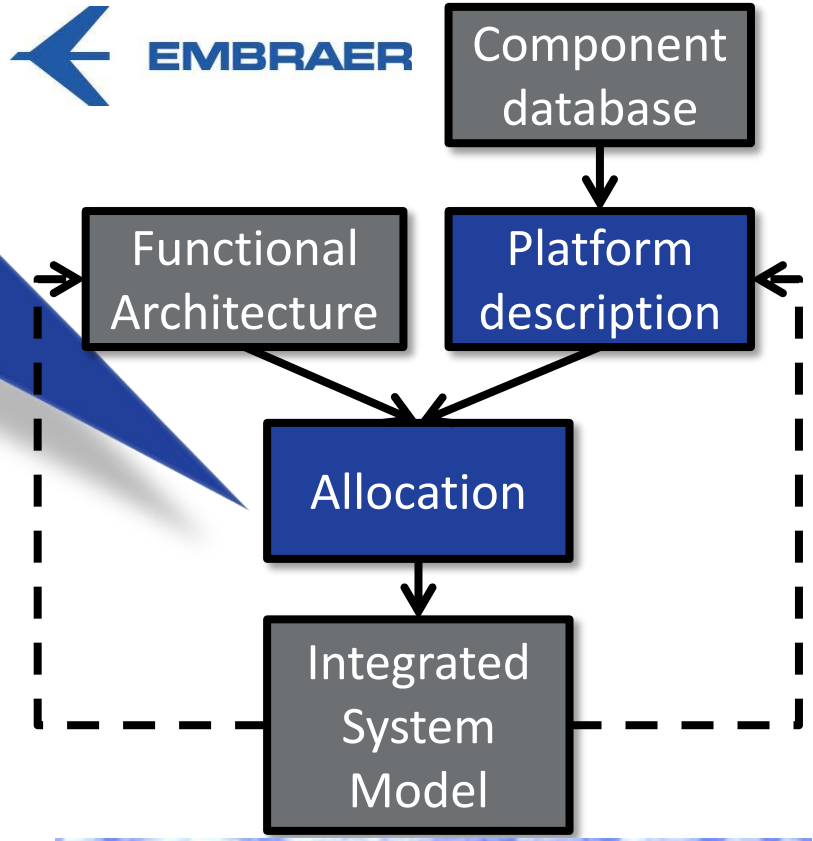
Problems:

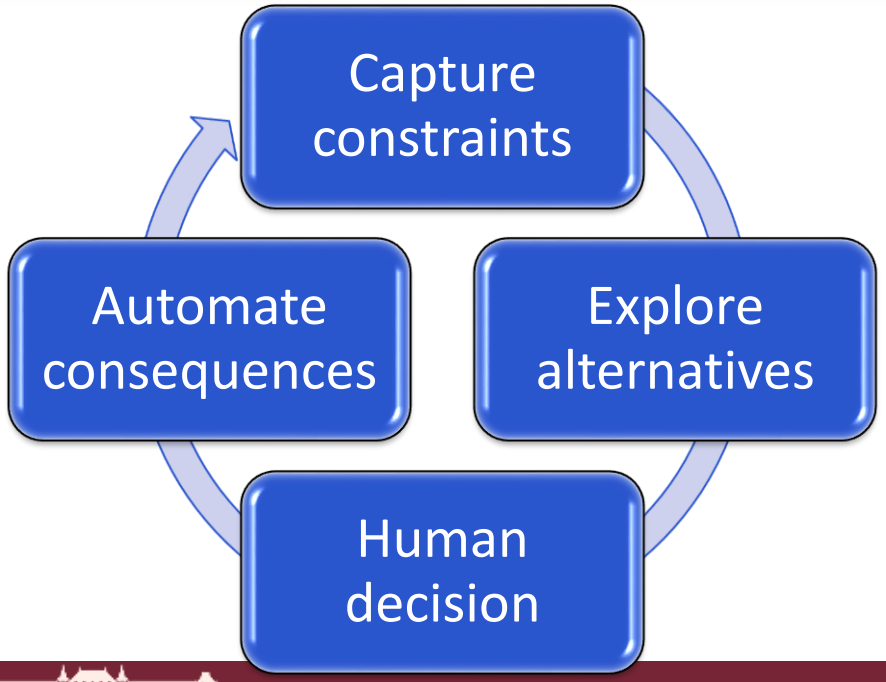
- Marking is too complex
- Not all MT steps can be automated



Model Driven Development of IMA Configs

Model transformation chains:
- Designer-guided manual steps
- Automated steps
  - design space exploration
  - optimization
  - code generators
- Continuous validation of design rules

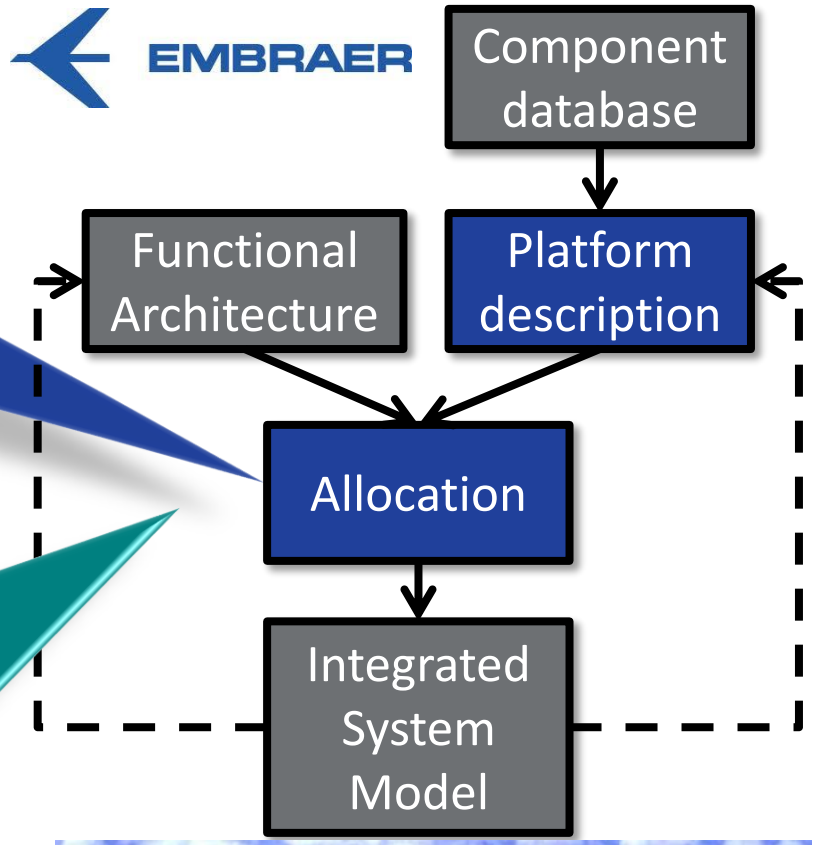




Model Driven Development of IMA Configs

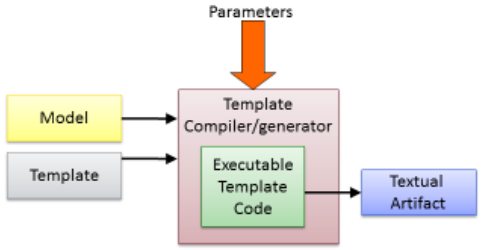
Precise development workflow:
- Aligned with certification-compliant development process
- Monitors design phases
  - completed steps
  - incomplete steps

End-to-end traceability:
- Traceability models
  - linking FAM and PDM to IAM
  - integration with requirements tool (e.g. DOORS)
- Soft interconnection of models by incremental model queries

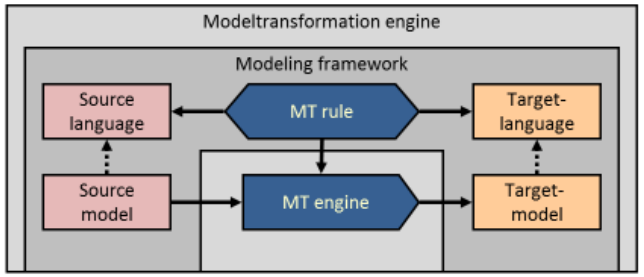


Summary

Template based approach

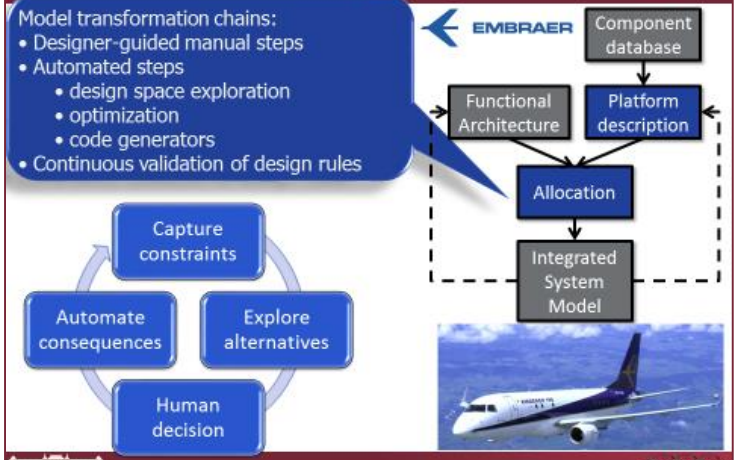


Definition of Model Transformation



Model Driven Development of IMA Configs

Model transformation chains:
- Designer-guided manual steps
- Automated steps
  - design space exploration
  - optimization
  - code generators
- Continuous validation of design rules



Platform-based systems design