Basics of Modeltransformation

Ákos Horváth

Dániel Varró

Model Driven Software Development

Lecture 8

Development Process for Avionics Systems

Unique Development Process (Traditional V-Model)

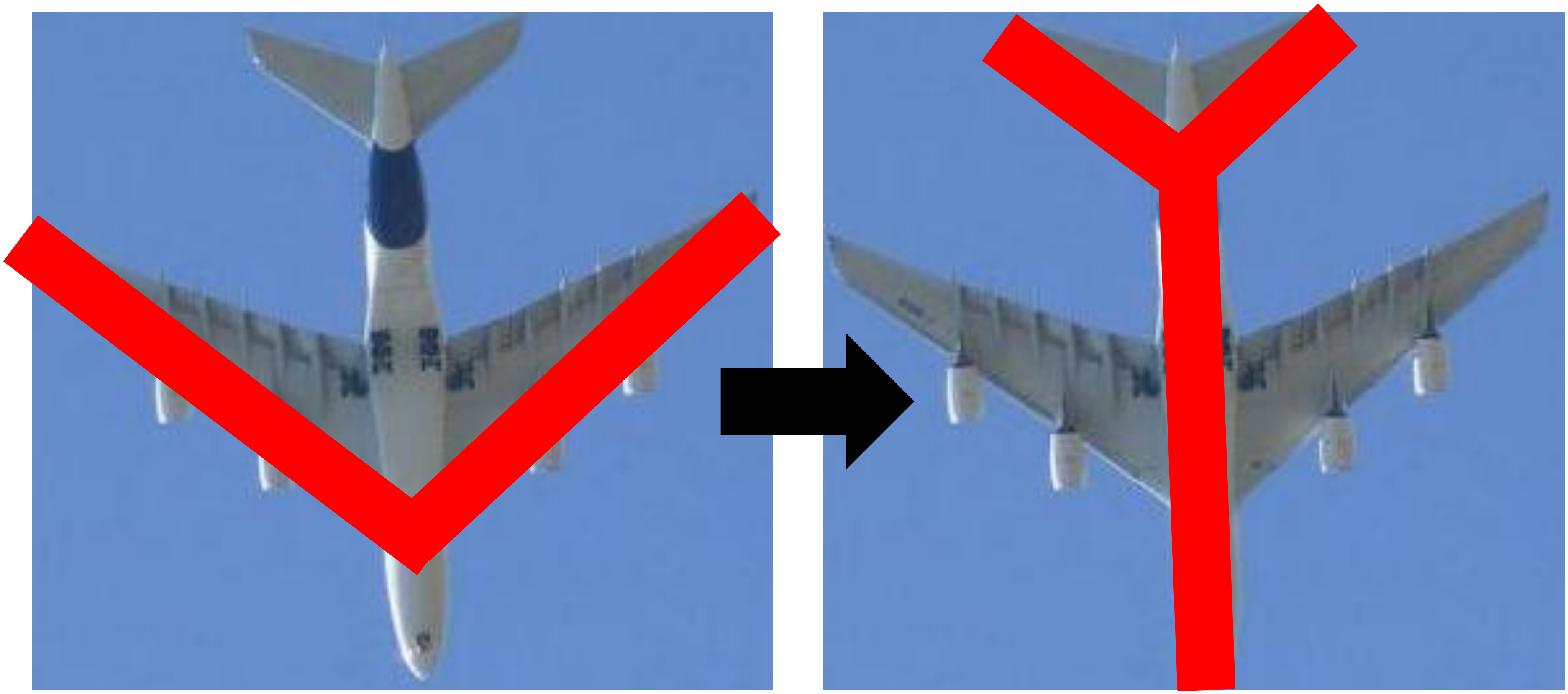
Avionics Systems Design

- requires a certification process
  - DO-178B
- to develop justified evidence
  - Certification artifacts
- that the system is free of flaws
  - Fulfils the requirements → traceability from requirements to synthesized source code

Certified tool → Fault-free output

Development Process for Avionics Systems

Traditional V-Model | Model-Driven Engineering

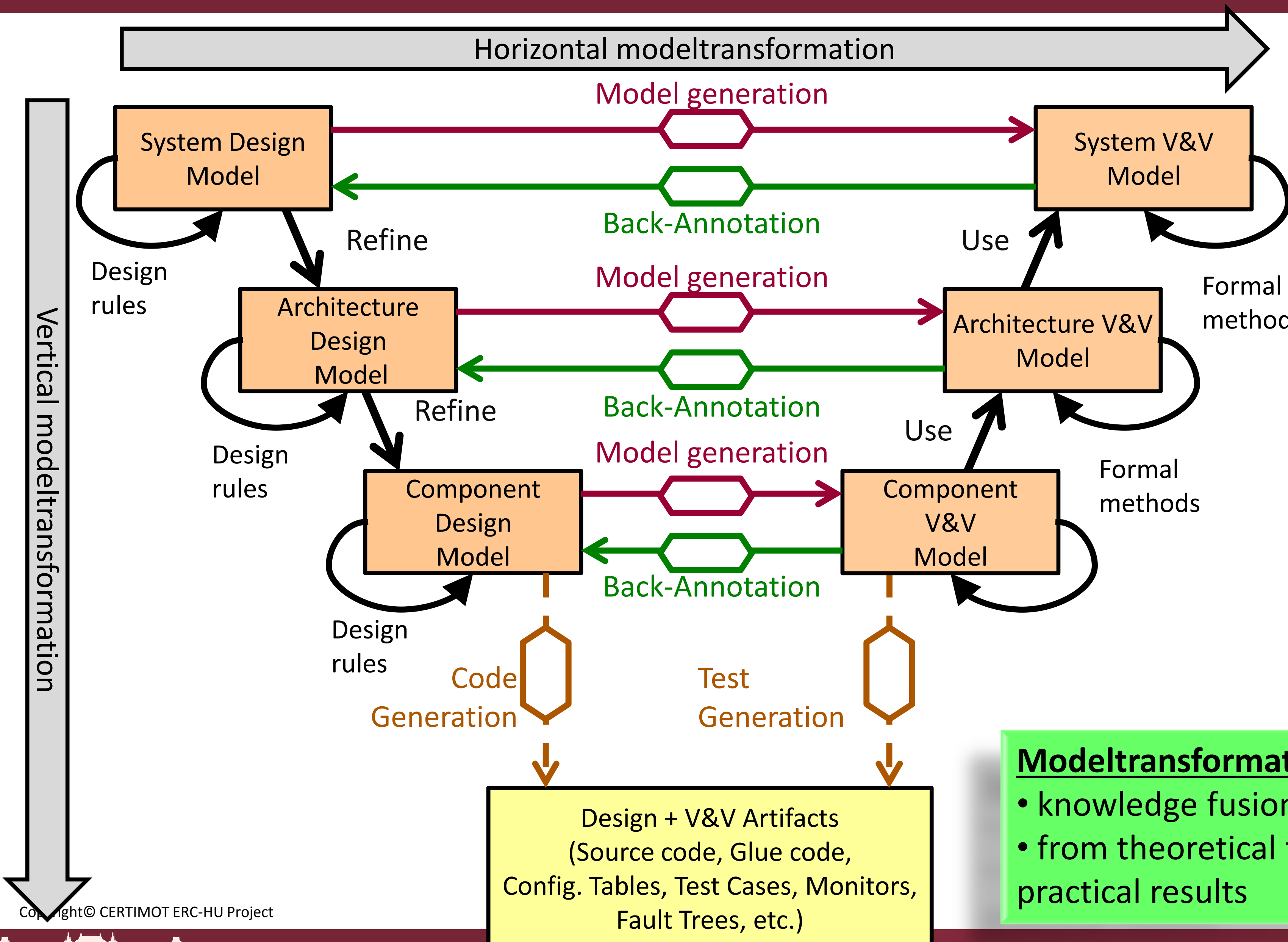


- DO-178B/C: Software Considerations in Airborne Systems and Equipment Certification (RTCA, EUROCAE)
- Steven P. Miller: Certification Issues in Model Based Development (Rockwell Collins)

Main ideas of MDE → DO-178C
- early validation of system models
- automatic source code generation

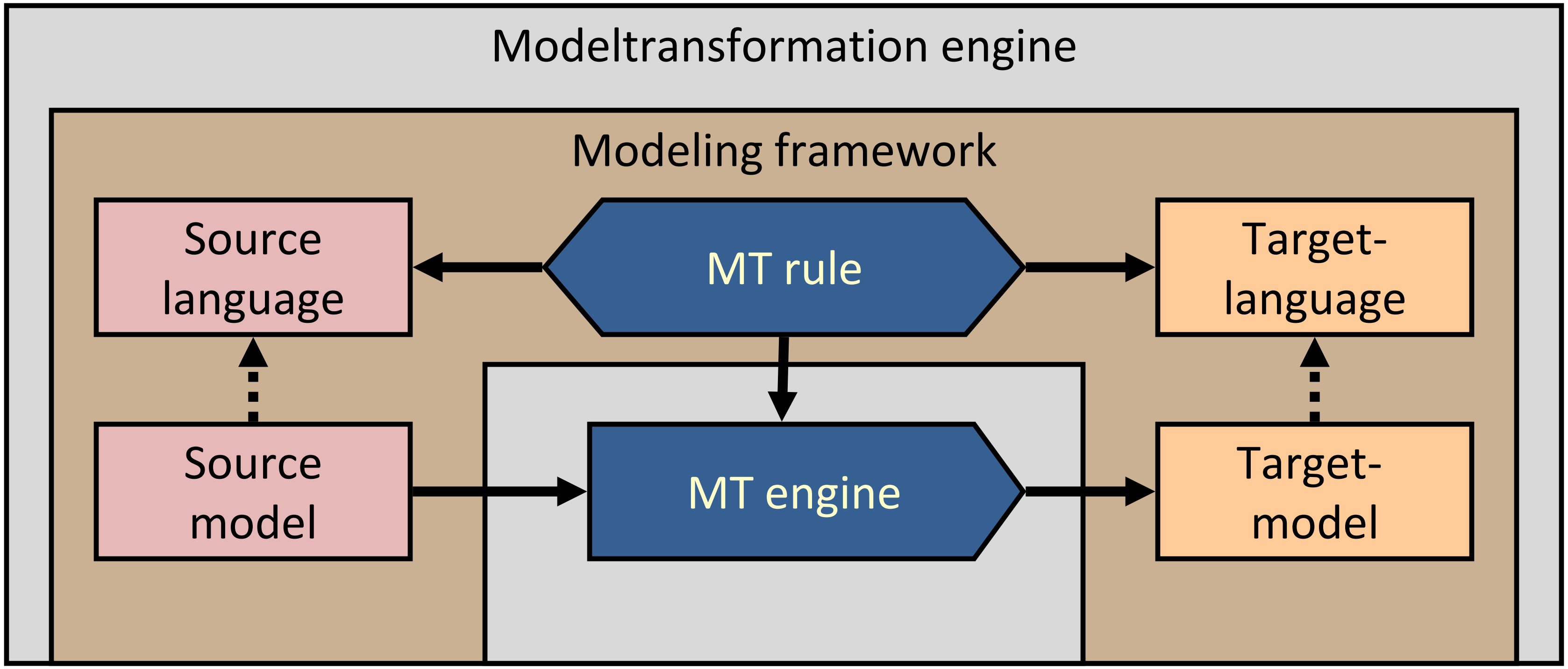
➔ reduce development costs

Models and Transformations in Avionics Systems Development

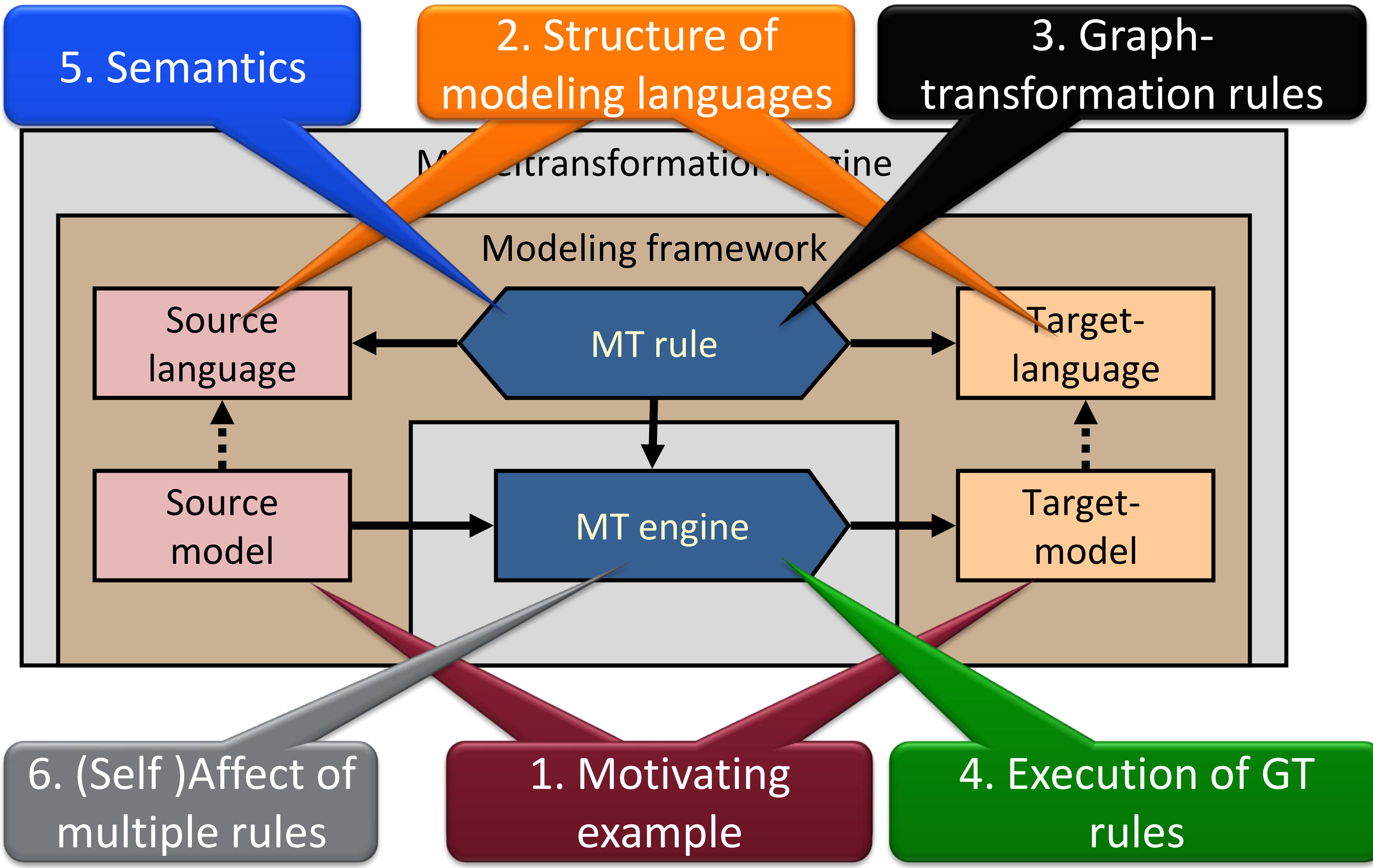


Modeltransformation
- knowledge fusion
- from theoretical to practical results

Definition of Modeltransformation



Overview



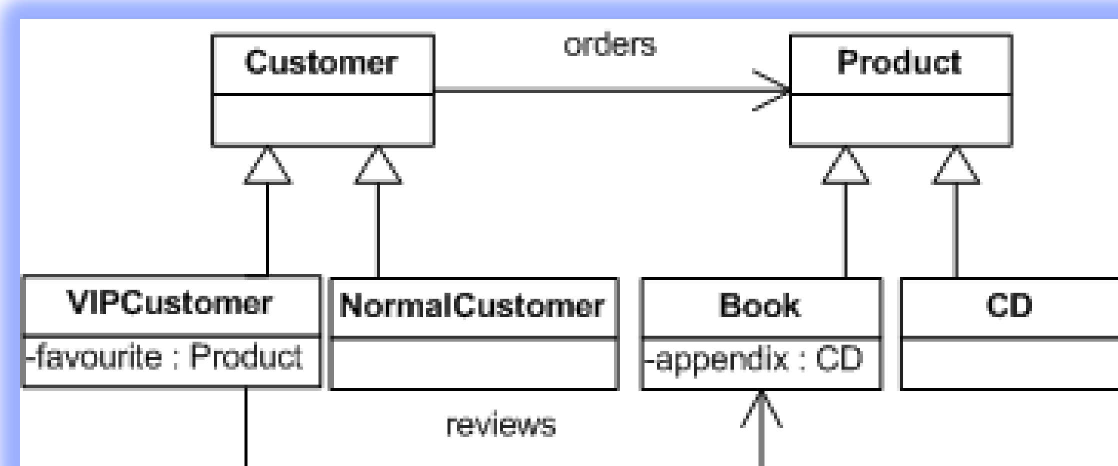
5. Semantics
2. Structure of modeling languages
3. Graph-transformation rules
Modeling framework
Source language
MT rule
Target-language
Source model
MT engine
Target-model
6. (Self)Affect of multiple rules
1. Motivating example
4. Execution of GT rules

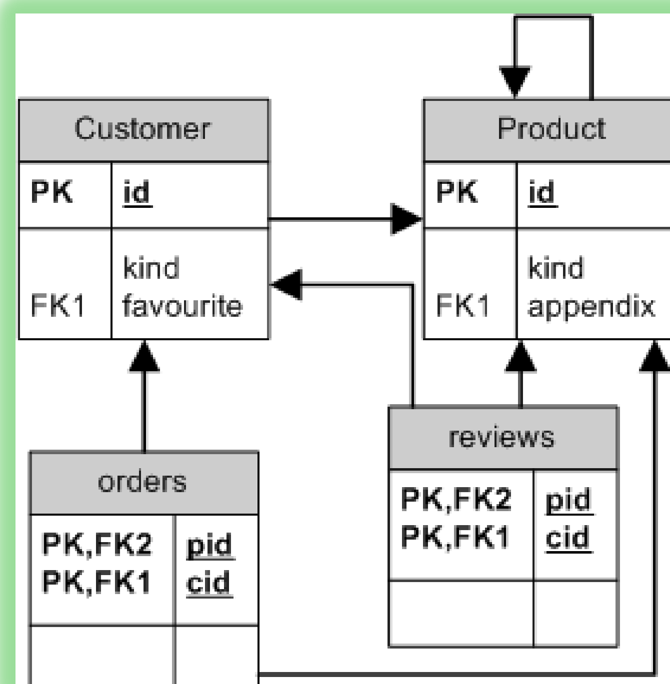
1. Motivating Example

Object Relation Schema mapping

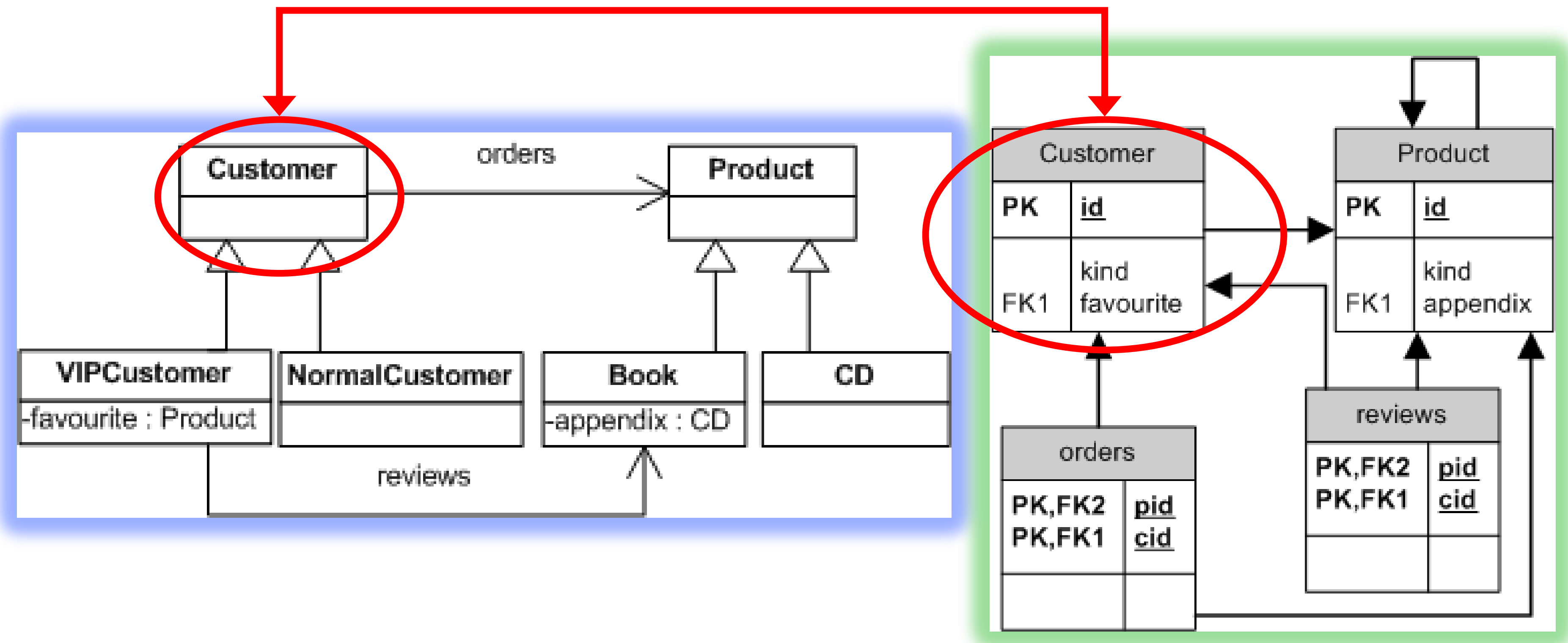
Example: Object-relational maping

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





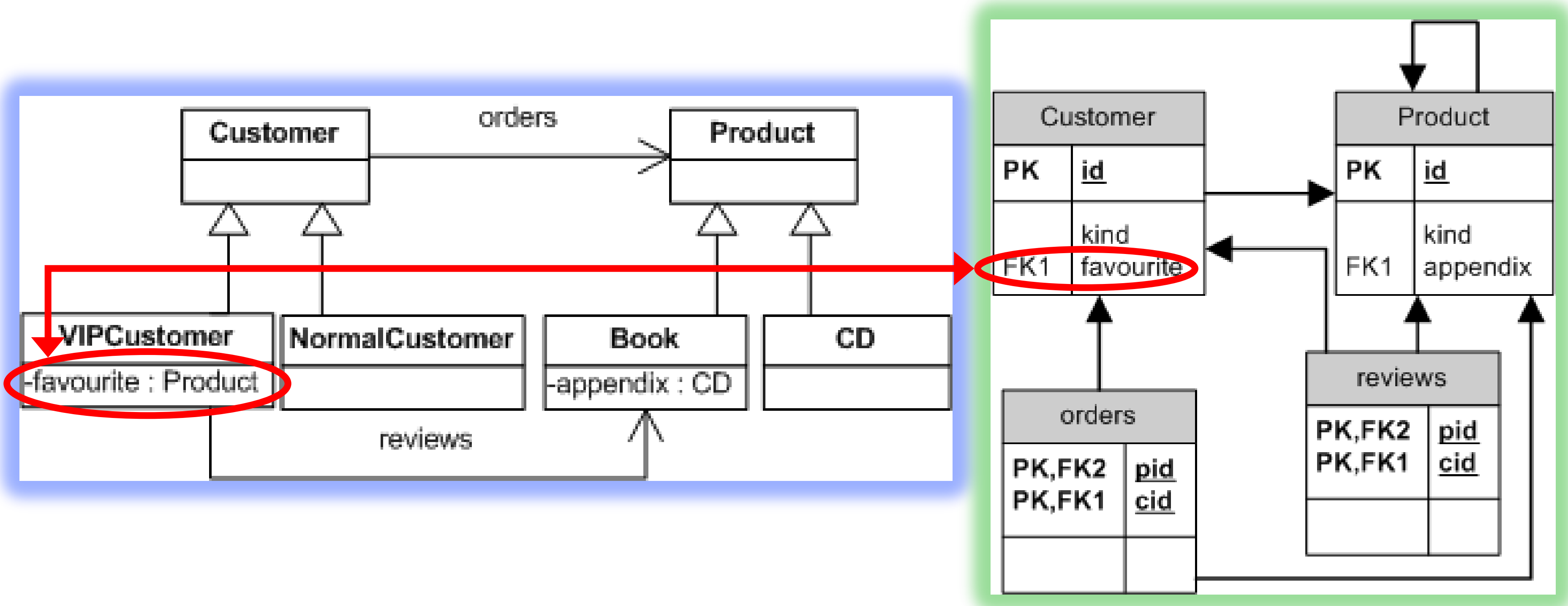
Informal definition of the MT rules of the mapping



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

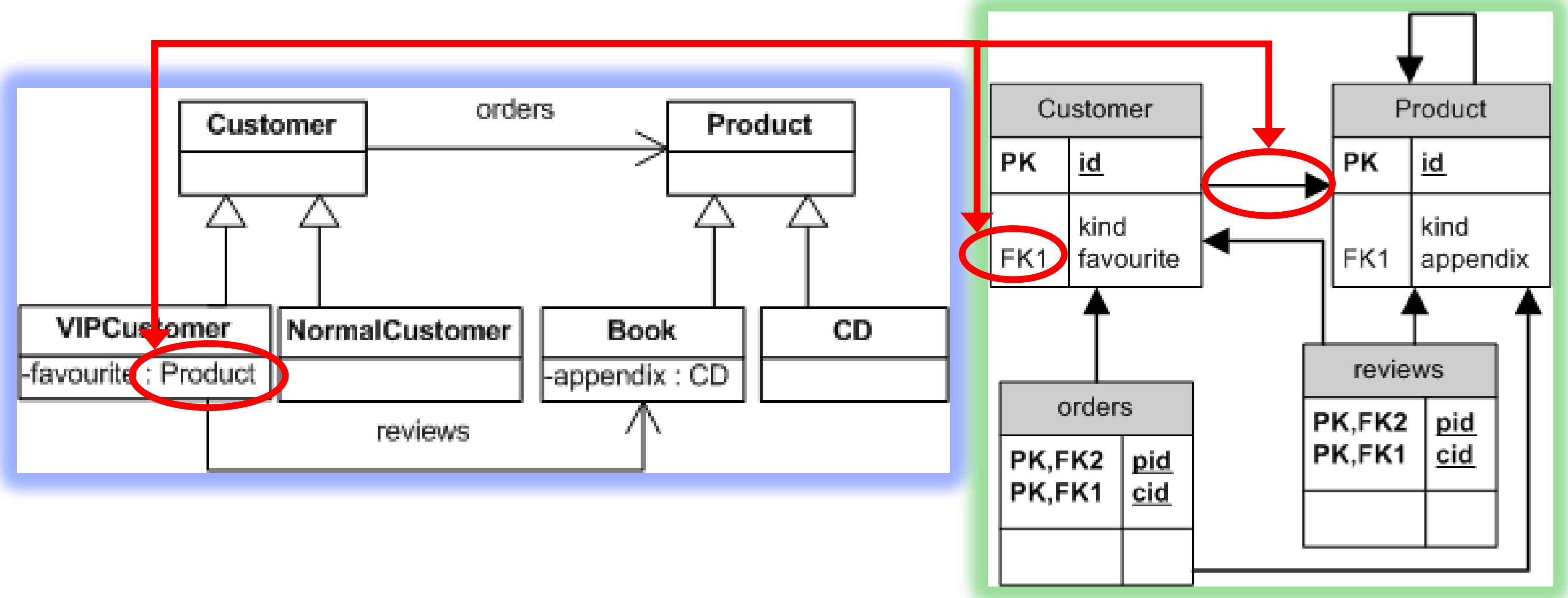
- Unique identifier (primary key),
- type definition

Informal definition of the MT rules of the mapping



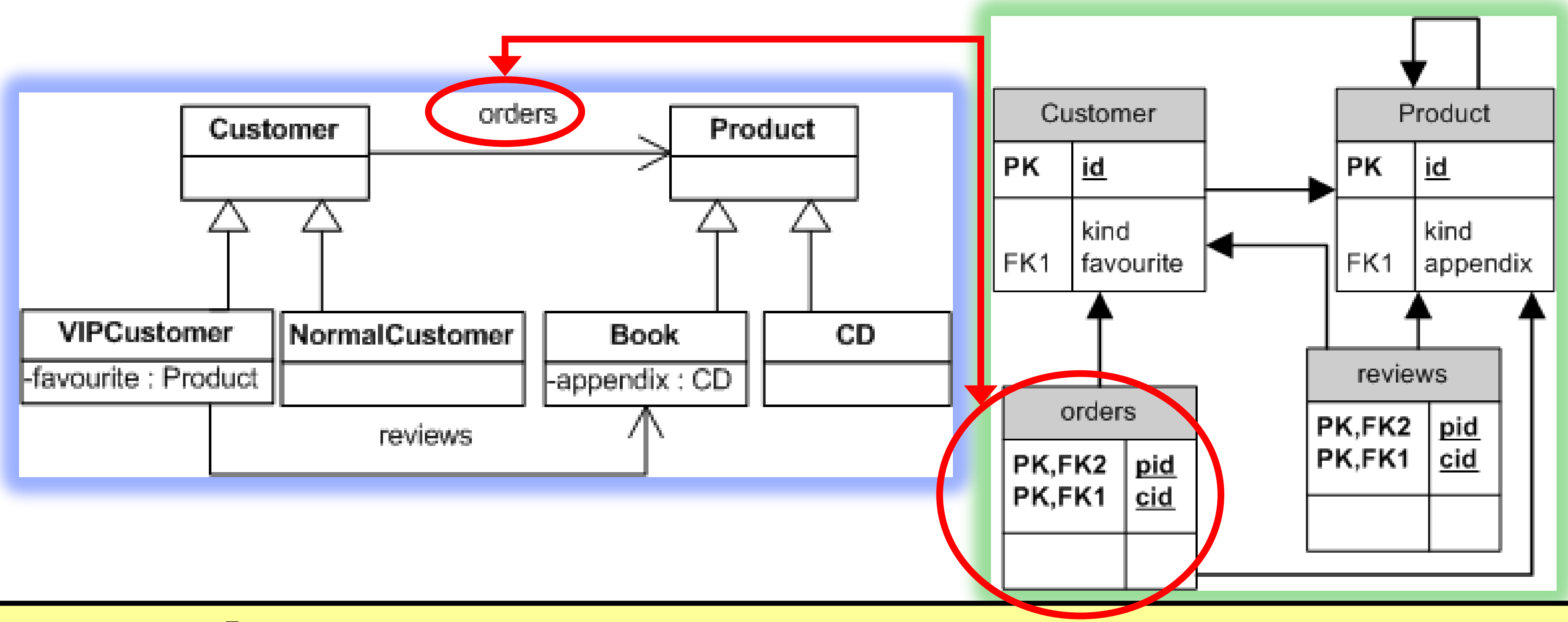
Class attributes ➔ (contained by the topmost classes) Column of the table

Informal definition of the MT rules of the mapping



Type of the attributes ➔ foreign key

Informal definition of the MT rules of the mapping



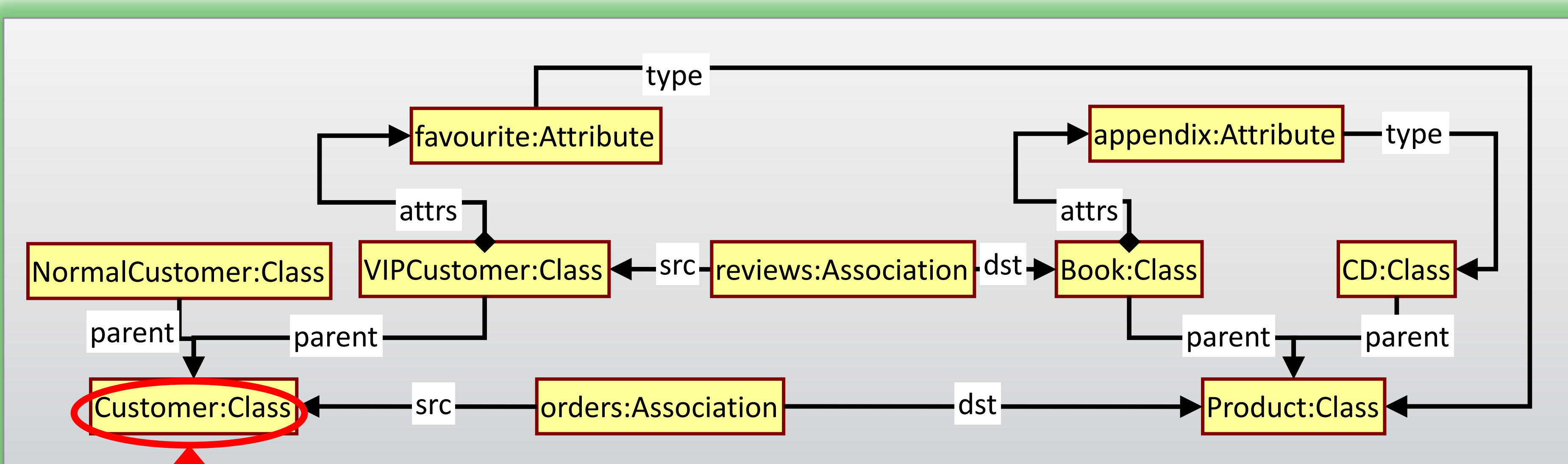
Association → A table with two columns

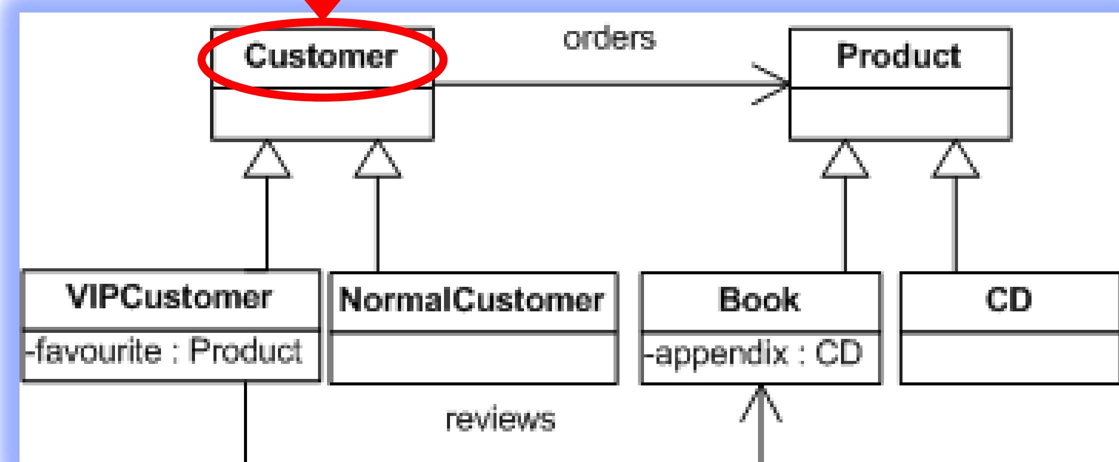
- source and target identifiers
- foreign keys (for consistency)

2. Structure of Modeling Languages

Overview

Structure of Modeling languages (UML)

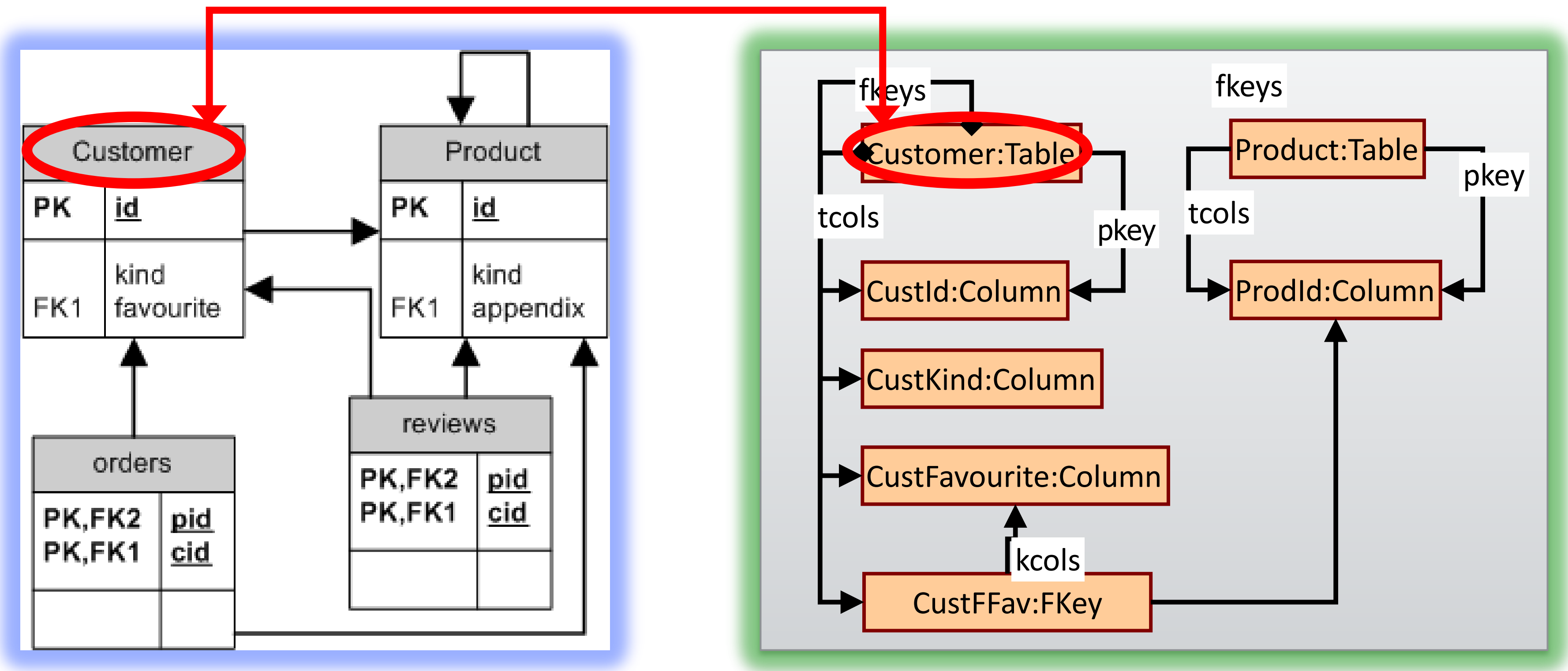




Abstract syntax
- Graph based model representation
- Machine readable

Concrete syntax
- Visual/textual representation
- Human readable

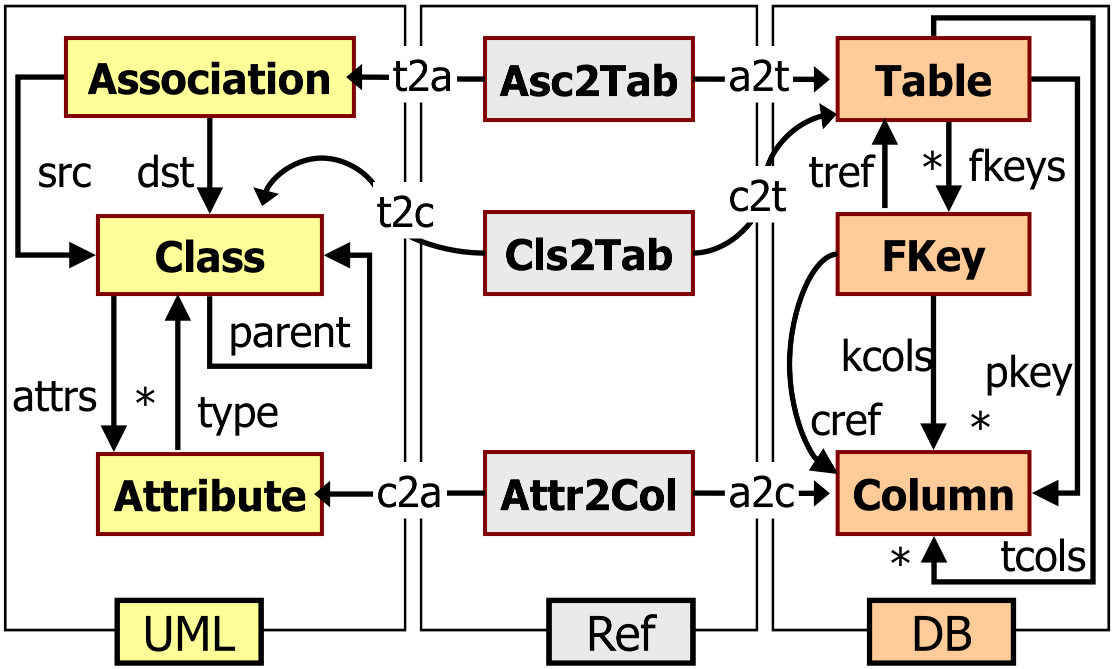
Structure of Modeling languages (RDBMS Schema)



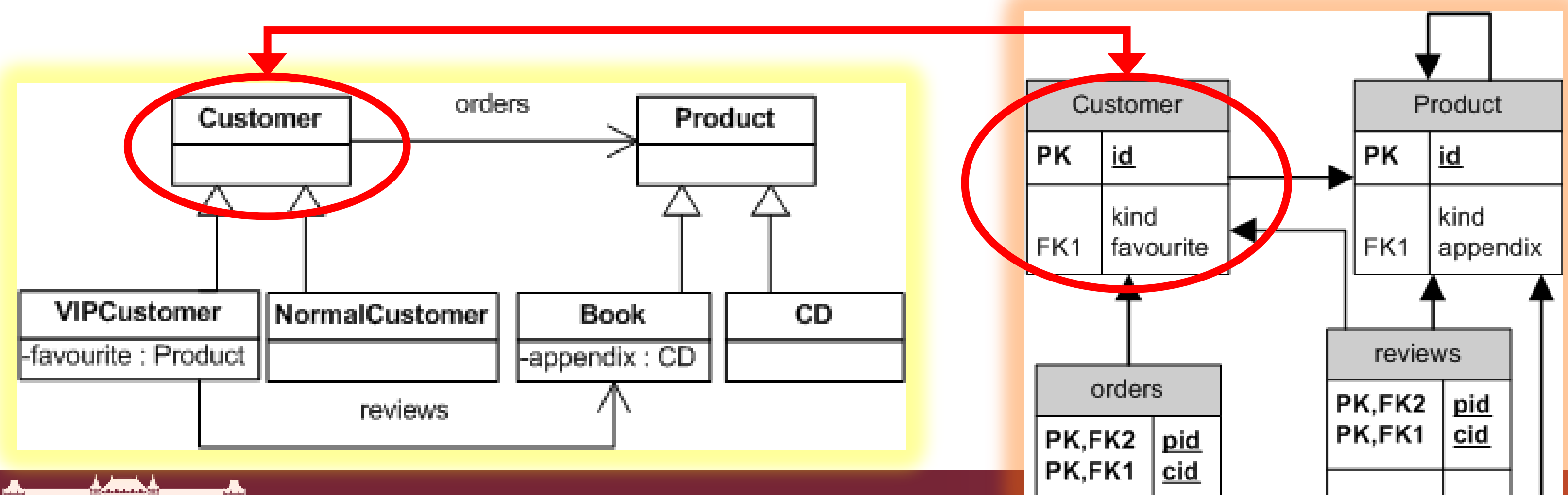
Concrete syntax

Abstract syntax

Metamodel of the O-R mapping

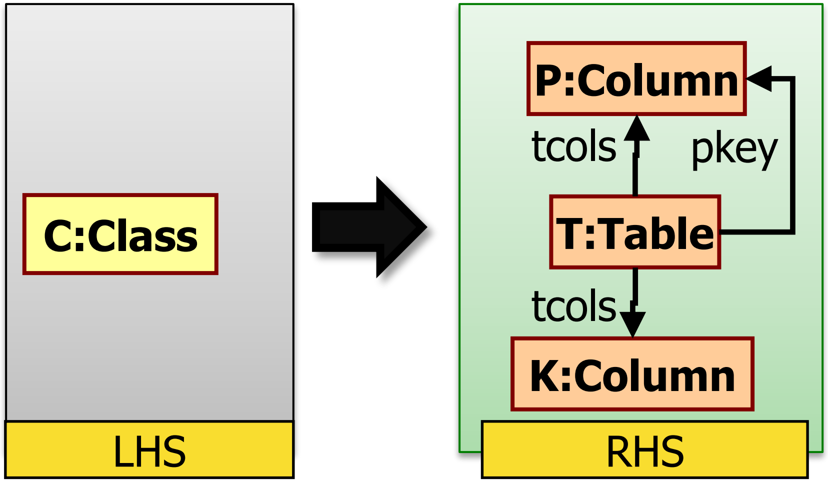


- Source + Target metamodel
- Traceability metamodel:
  - For saving the relations between the source and the target languages
- Motivation: critical embedded systems
  - Traceability
  - Requirement ➔ Source code



3. Graphtransformation rules

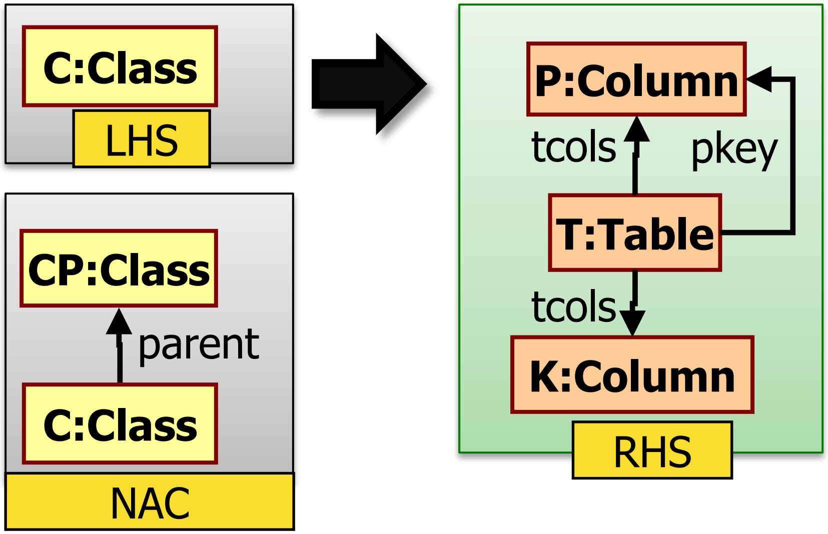
Structure of a GT rule



- **Graphtransformation 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)

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

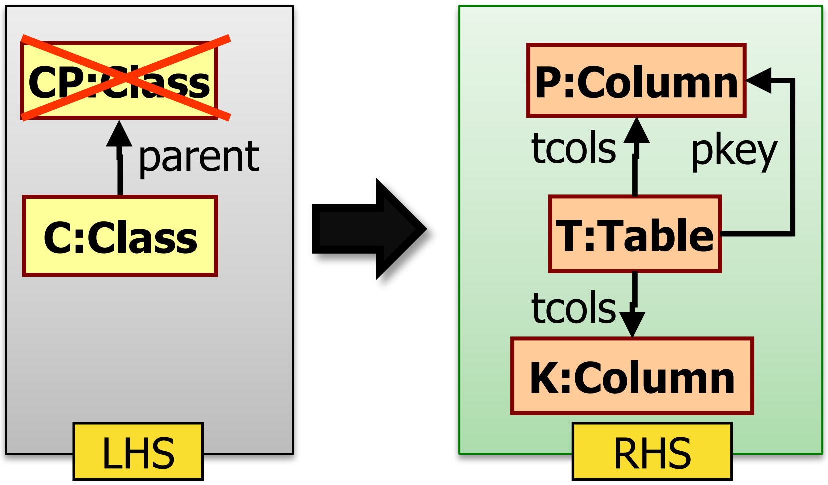
Structure of a GT rule



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

- **Graphtransformation 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



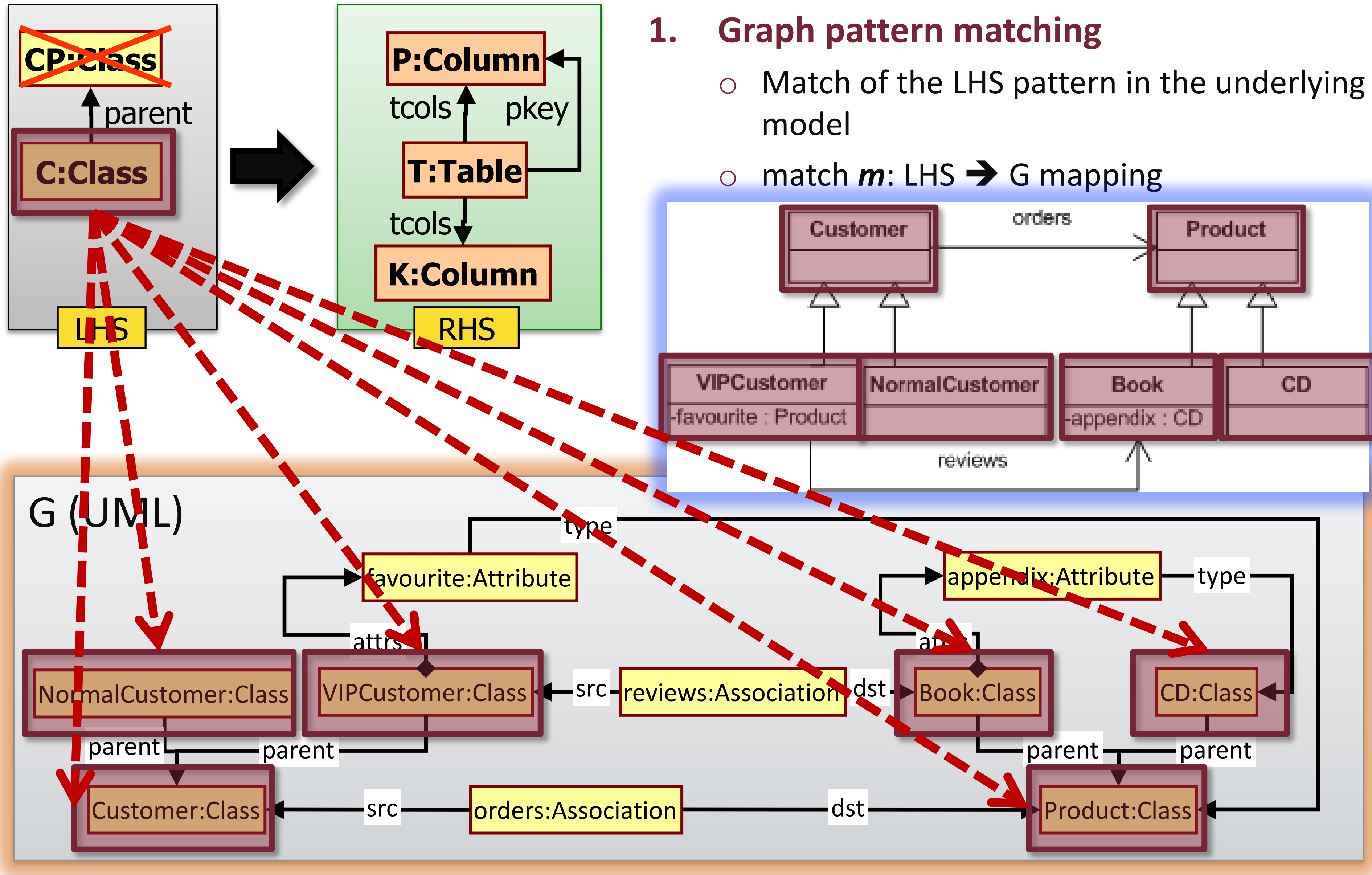
- **Graphtransformation (GT)**:
  - Declarative and formal paradigm
  - Rule base transformation
  - Match of the LHS➔ match of the RHS
  - Generalization of Chomsky grammars (hierarchy) (text ➔ graph)

- **Graphtransformation 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 Graphtransformation rules

Application of GT rules

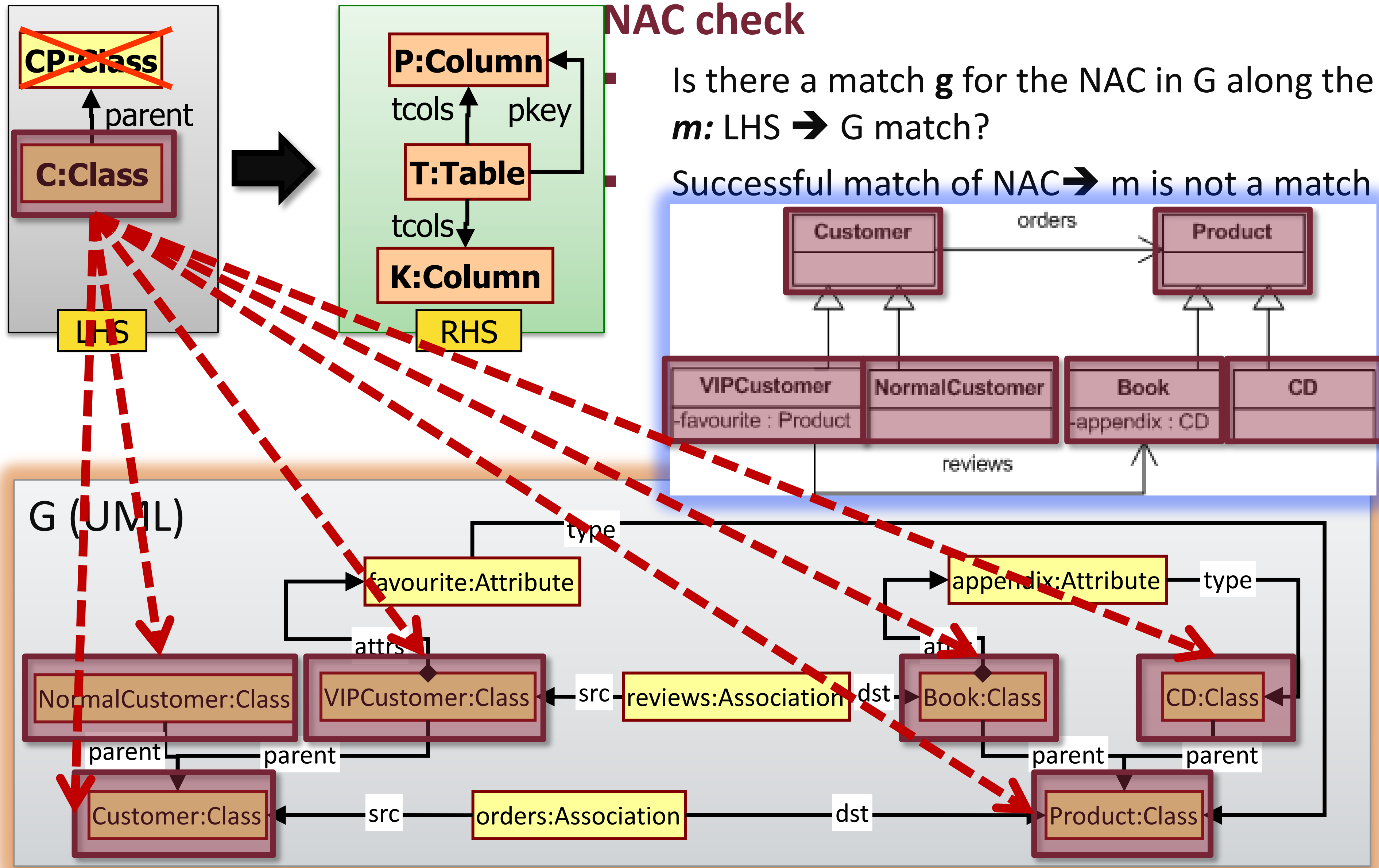
1. **Graph pattern matching**
   - Match of the LHS pattern in the underlying model
   - match *m*: LHS → G mapping



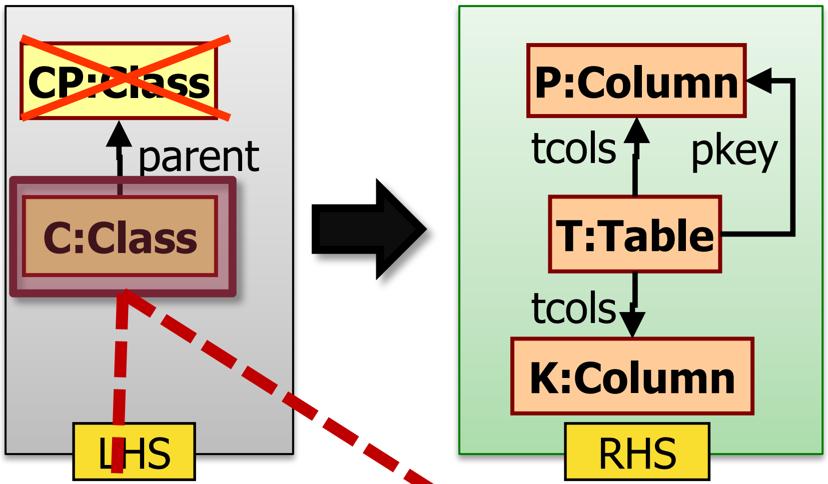
Application of GT rules

NAC check

- Is there a match **g** for the NAC in G along the ***m:*** LHS ➔ G match?
- Successful match of NAC➔ m is not a match

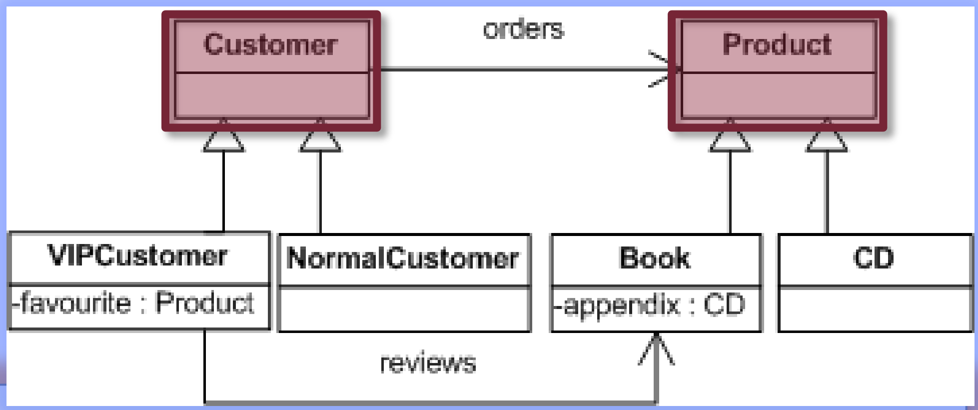


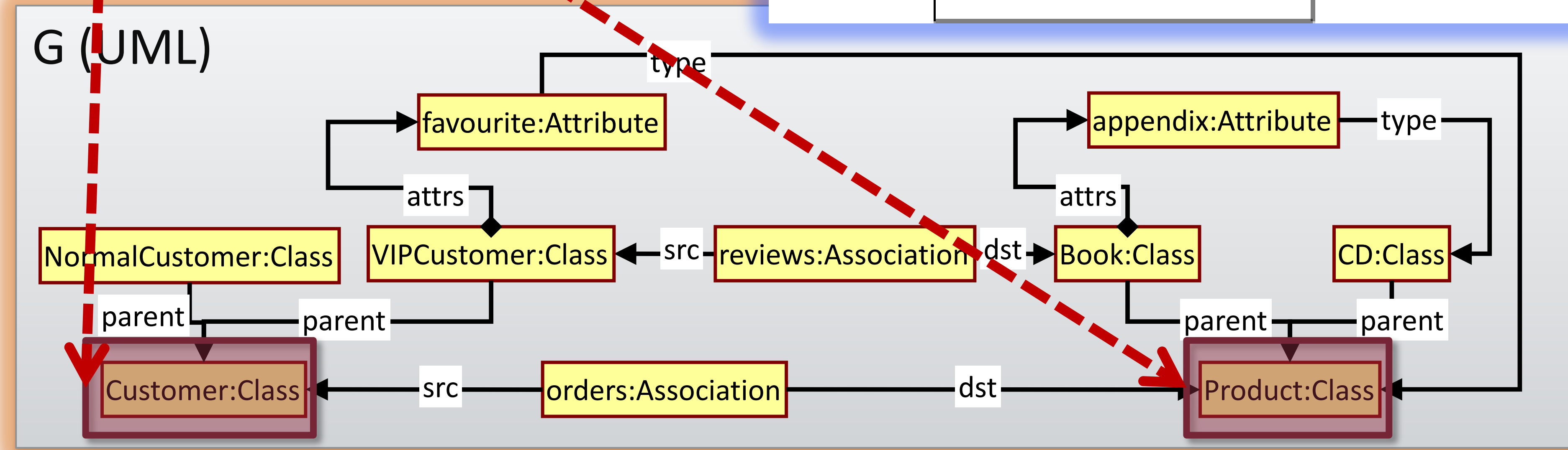
Application of GT rules



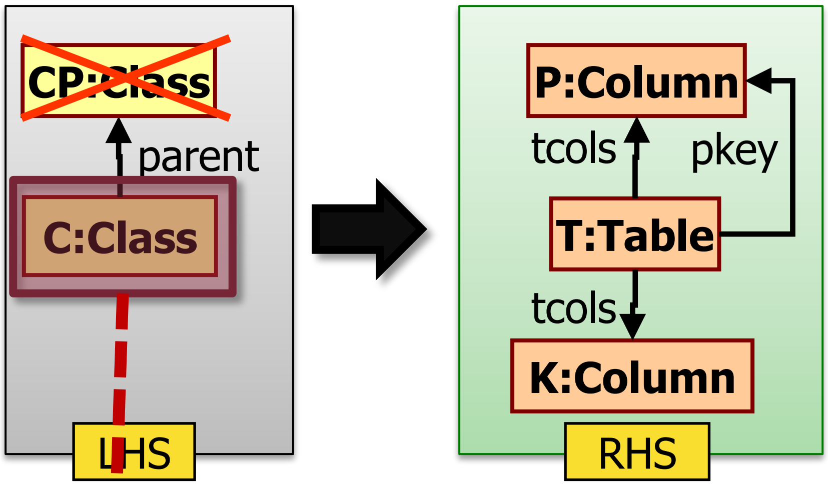
3. Nodeteministic selection

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



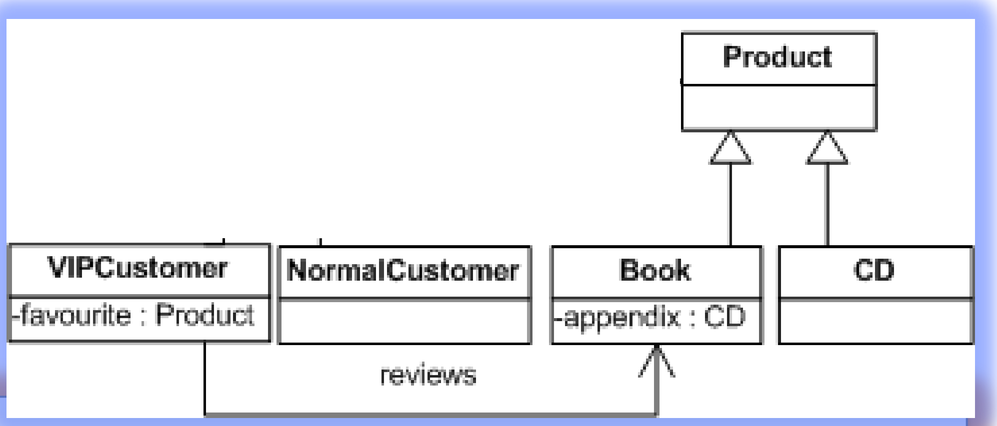


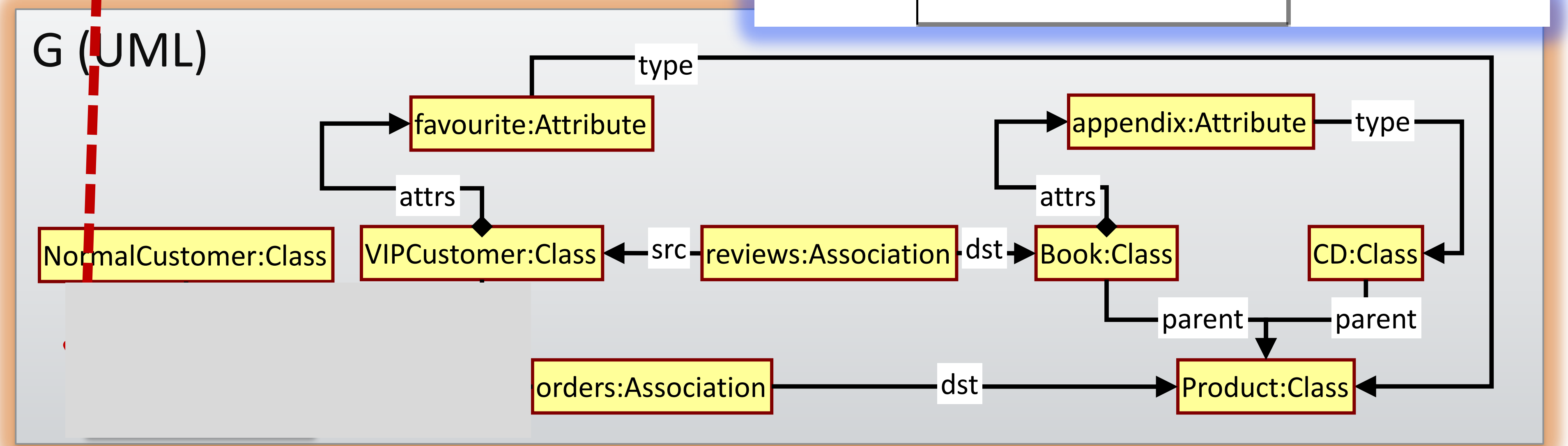
Application of GT rules



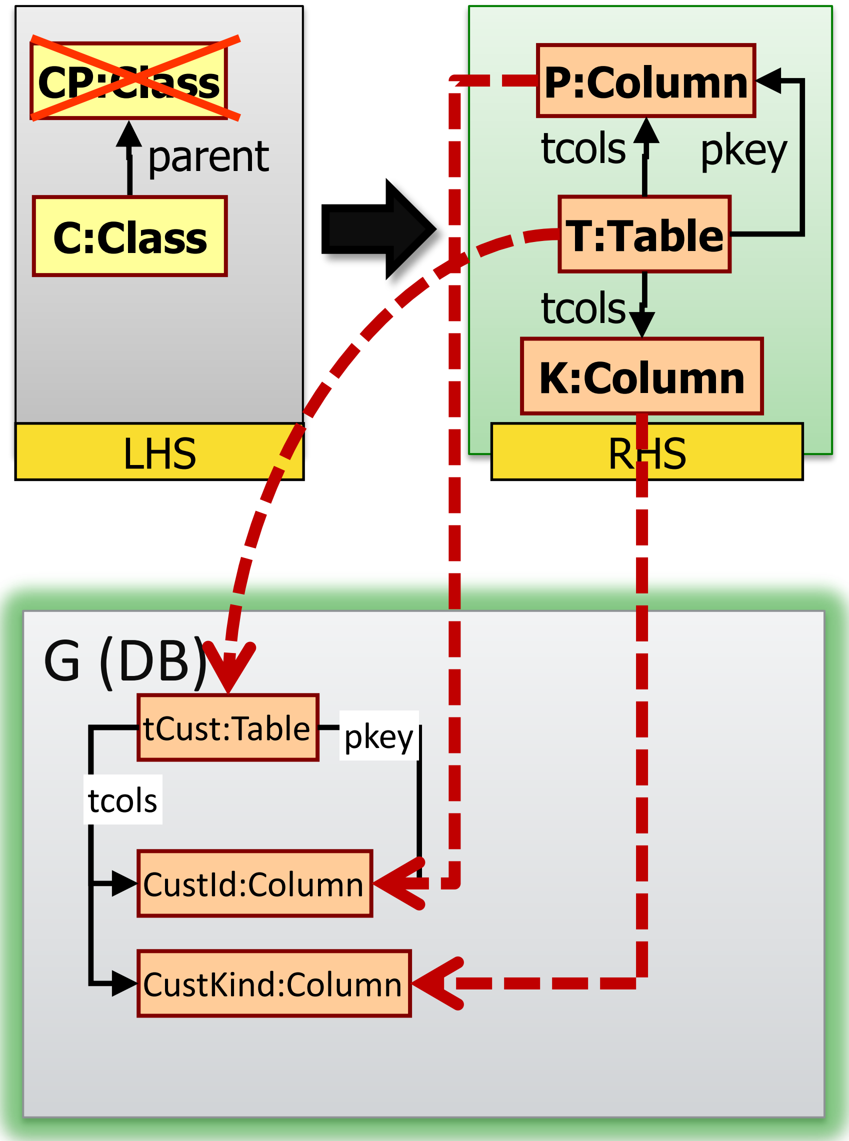
4. Deletion

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

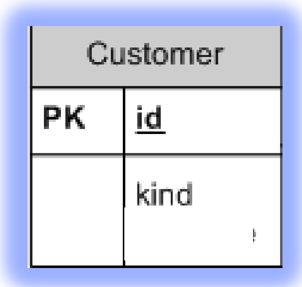




Application of GT rules

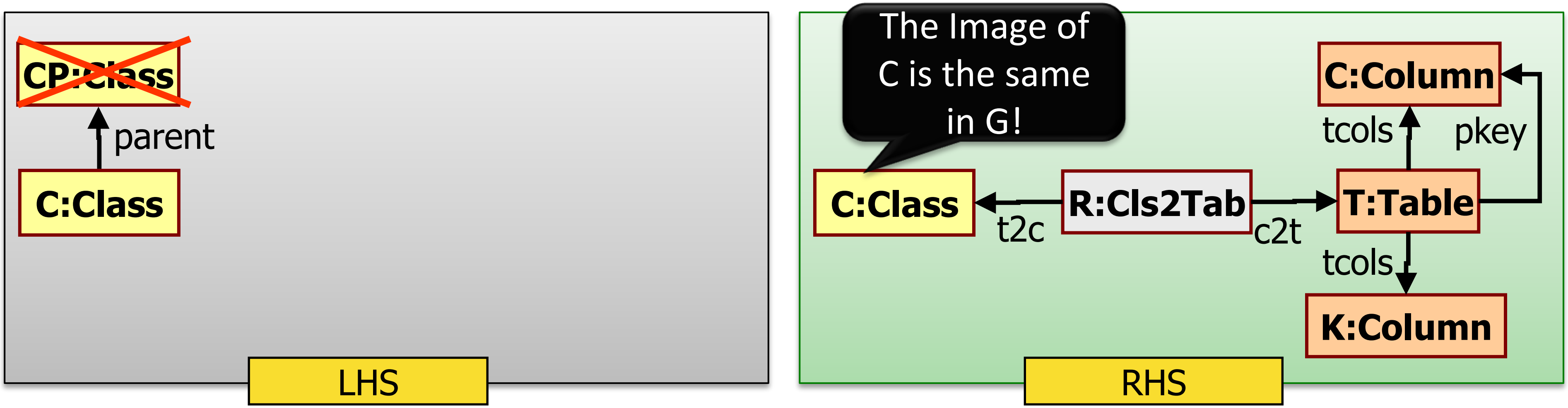


5. **Creation (and binding)**
   - Creation of RHS \ LHS in G with their corresponding relations
   - Output:
     a „match” of LHS in G

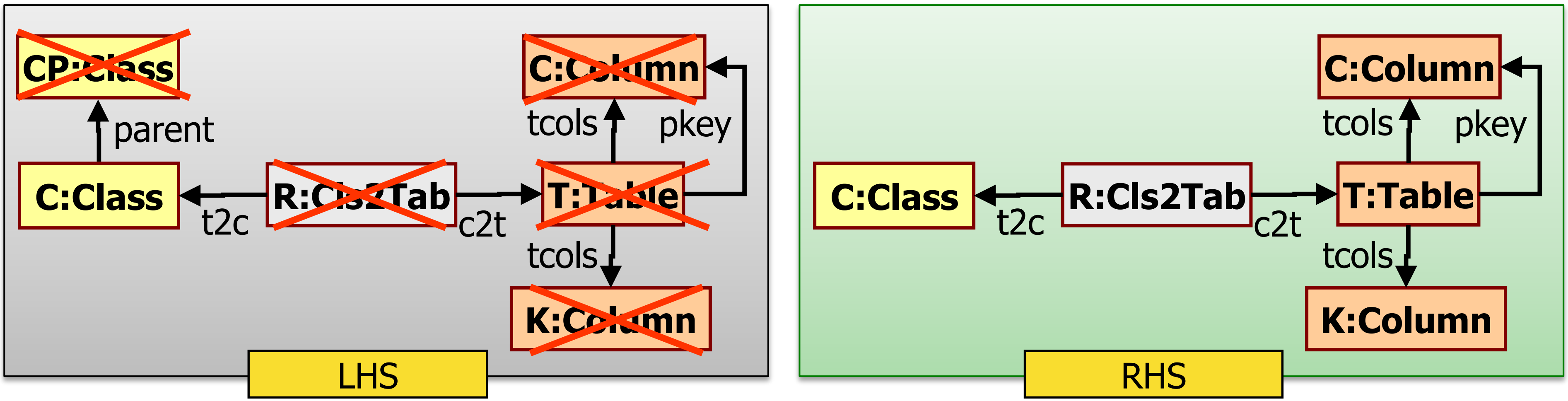


Typical problems…

1) Saving the source model, traceability

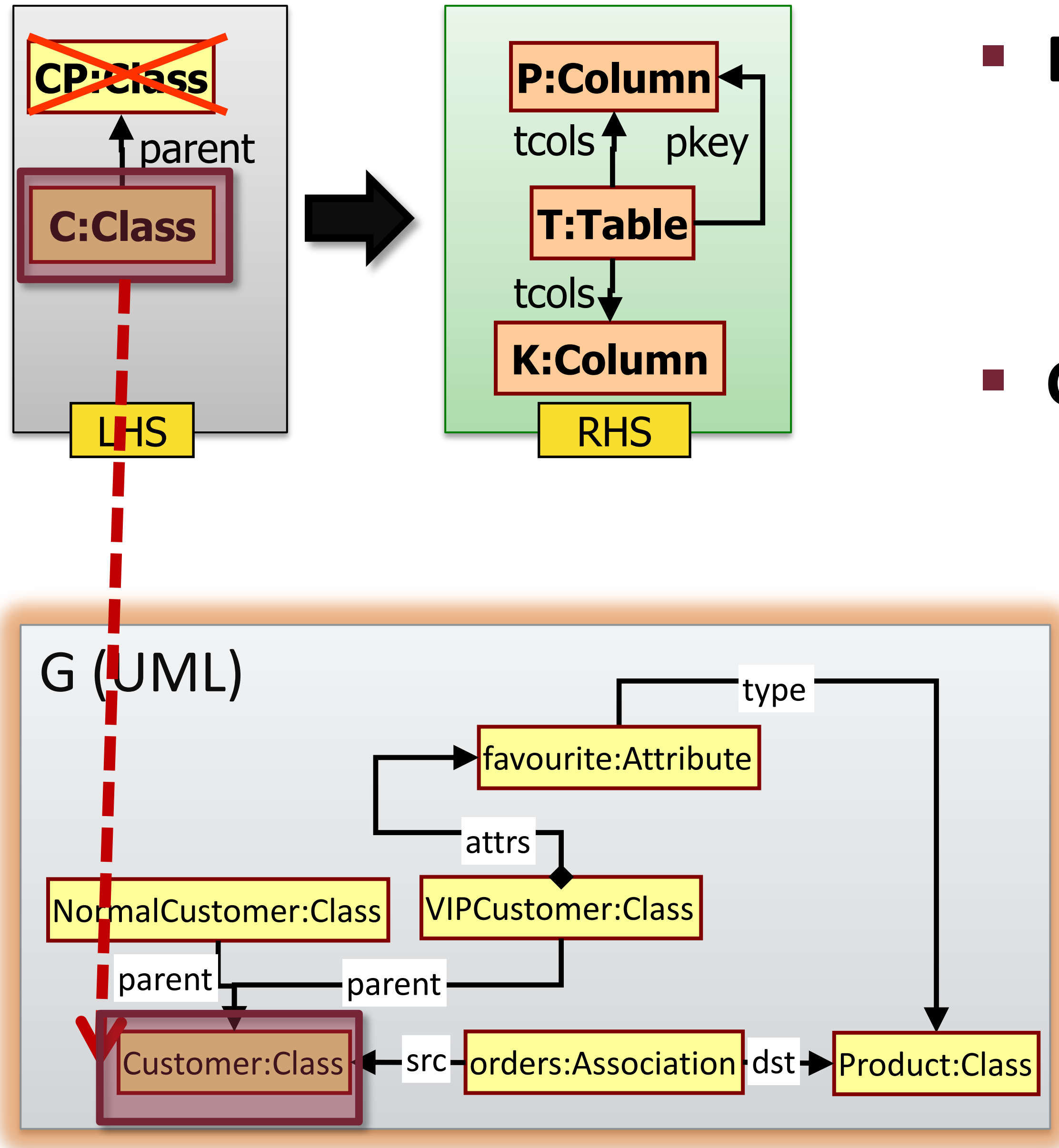


2) Application of the same rule along the same match



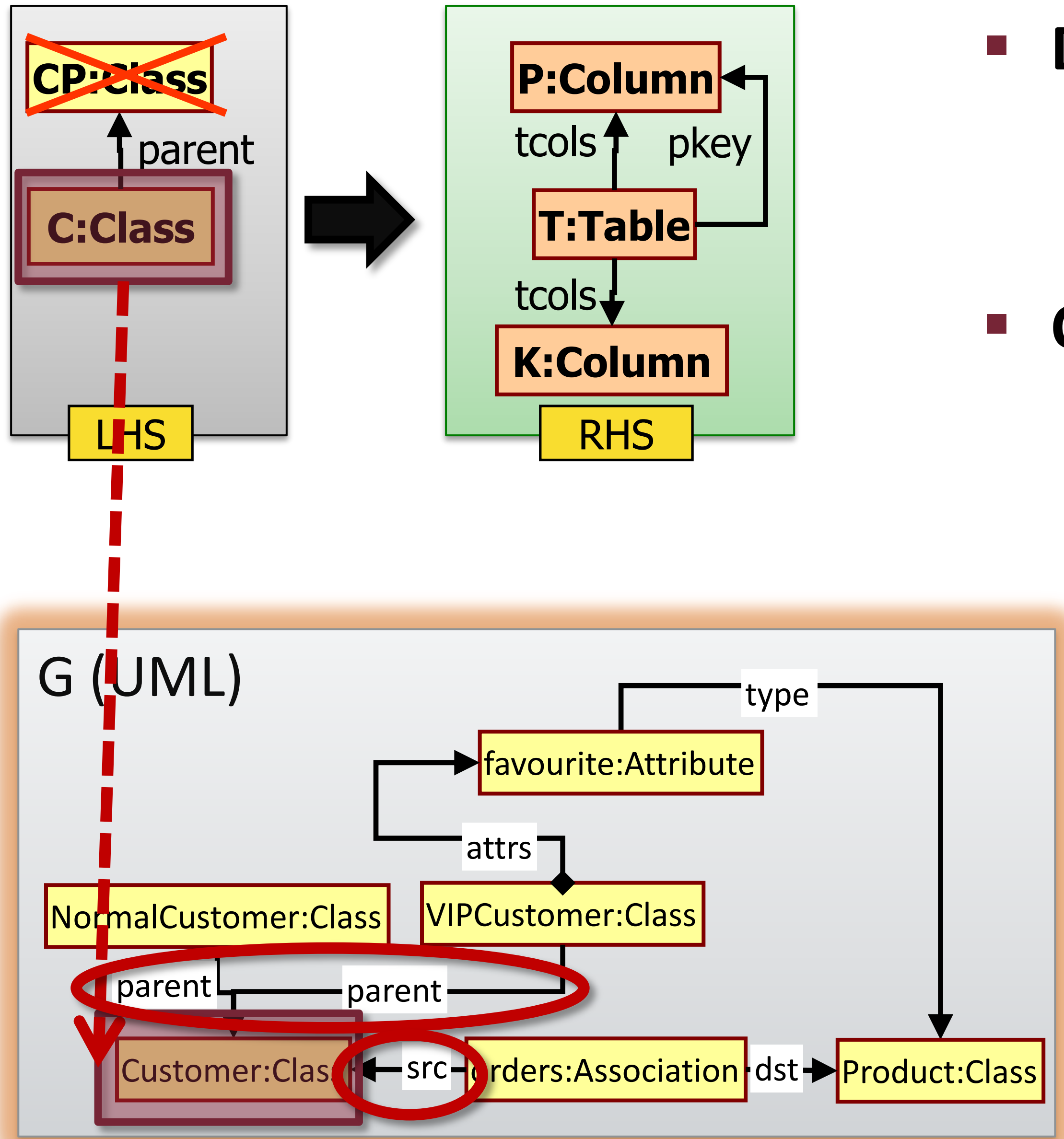
5. Different Semantics

Semantics : Handling of Dangling edges



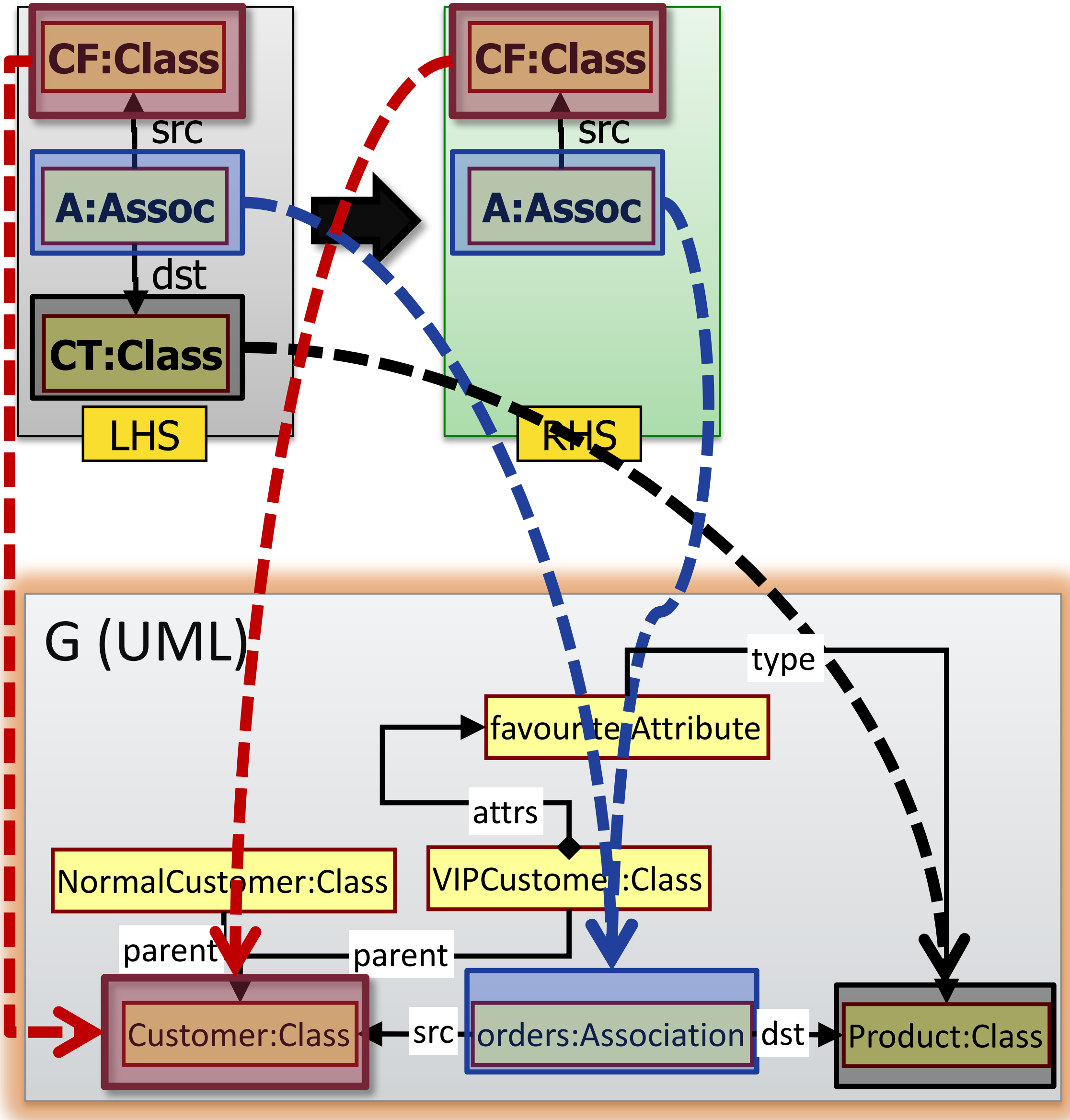
- **Dangling edges**:
  - Delete a node
    - What to do with the dangling edges?
- **Greedy approach**
  - Delete all dangling edges
  - **Pro**:
    - Intuitive for engineers
    - Easy to implement
  - **Con**:
    - Verification is hard (side effect of rules)

Semantics : Handling of Dangling edges



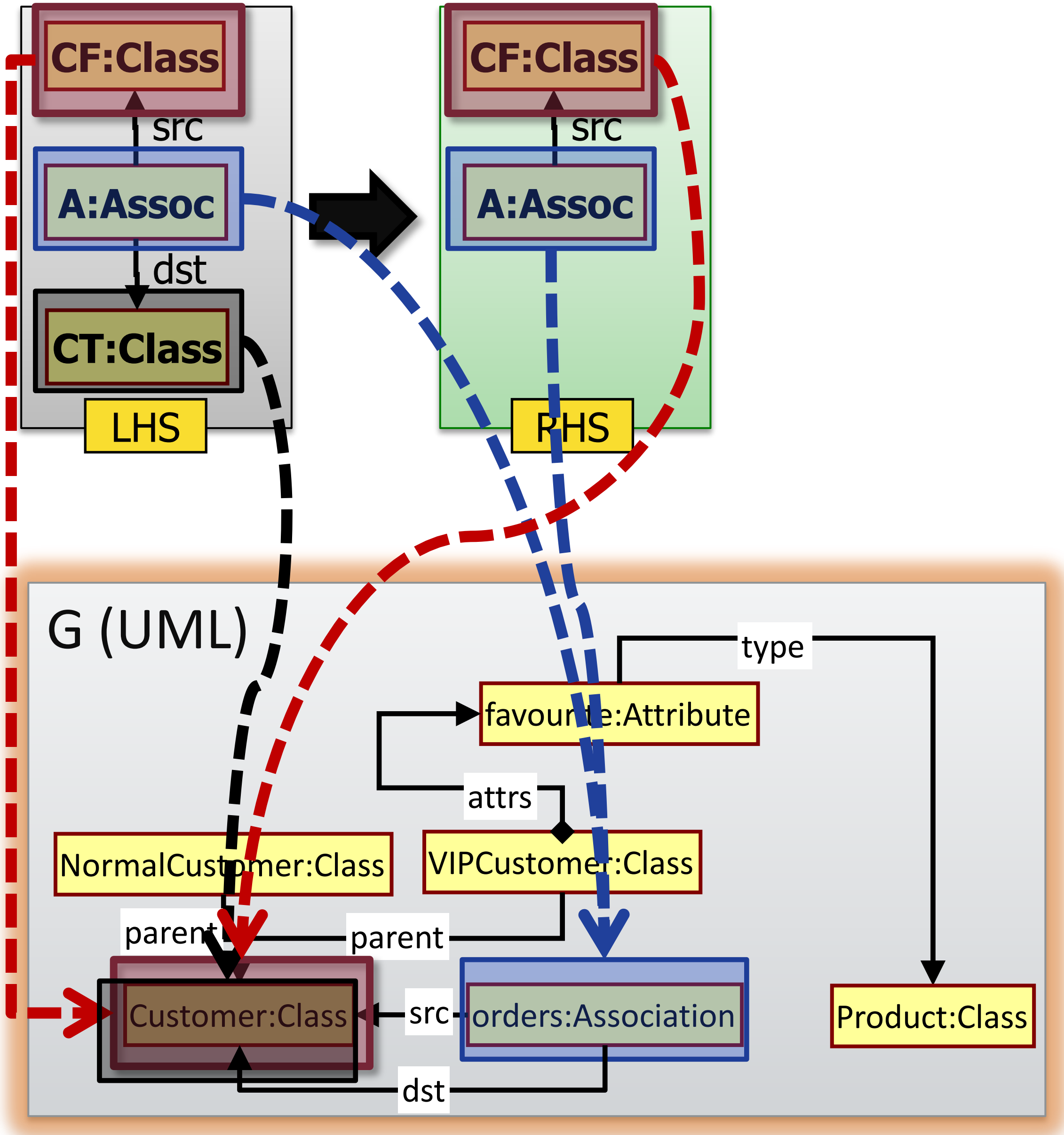
- **Dangling edges**:
  - Delete a node
    - What to do with the dangling edges?
- **Conservative approach**
  - The rule cannot be applied if it would produce a dangling edge
  - **Pro**:
    - Side effect free rules
    - Helps verification
  - **Con**:
    - Harder to implement
    - What is its meaning for engineers (not mathematicans)

Semantics: Injective matching



- **Injective matching („kisajátító”)**
  - For all nodes in the LHS➔ separate nodes are matched in G
- **Pro**:
  - Intuitive for engineers
- **Con**:
  - Verbose specification of rules
    (many alternate subrules)

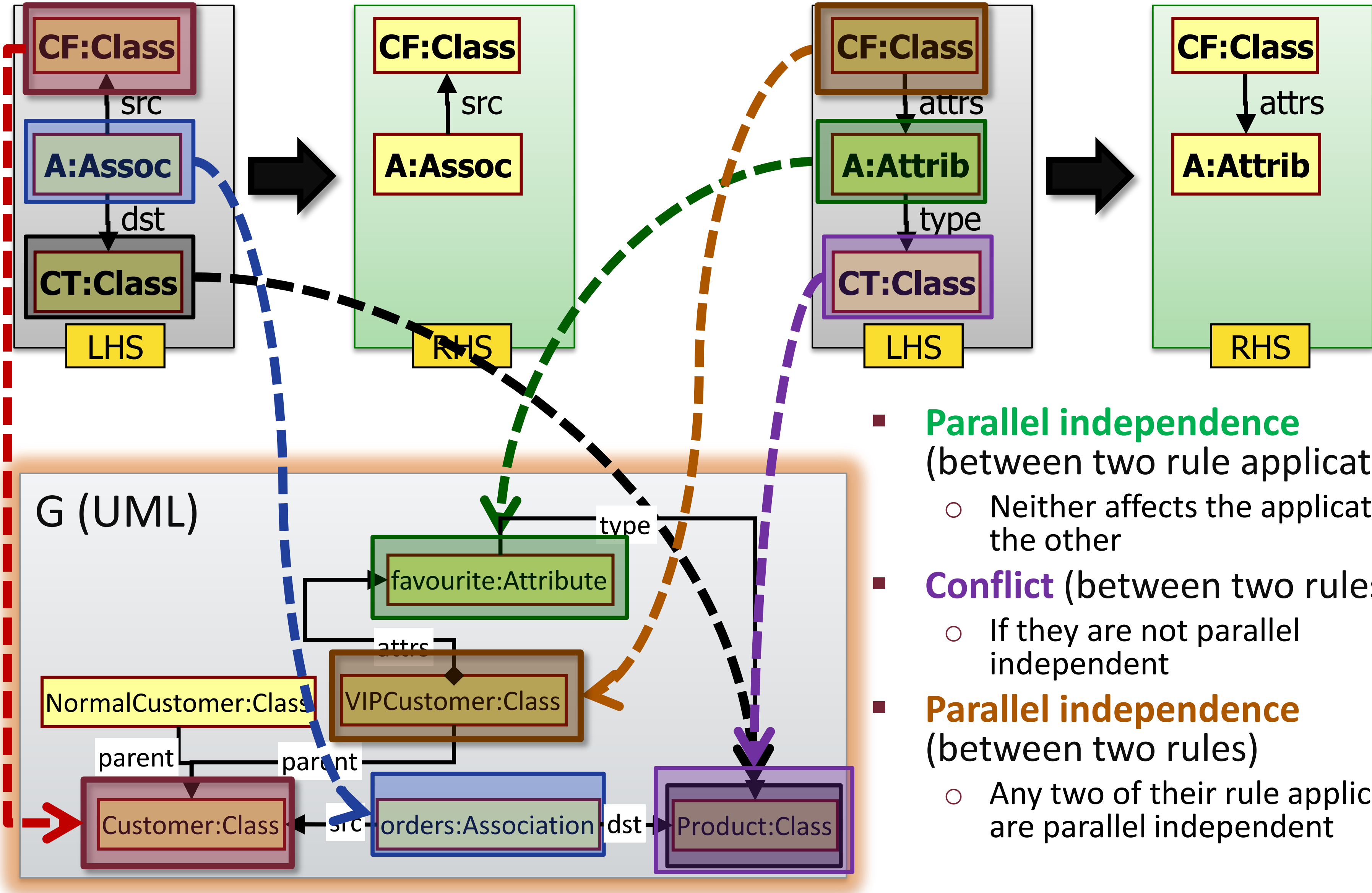
Semantics: Non-injective matching



- **Non-Injective matching („közösködő")**
  - For multiple nodes in the LHS ➔ the same node can be matched in G
- **Con**:
  - Contradictionary specification for a node
    - For **CF** : keep it
    - For **CT** : delete
- **Solution**:
  - Nodes to be deleted in LHS are matched with injectiv semantics

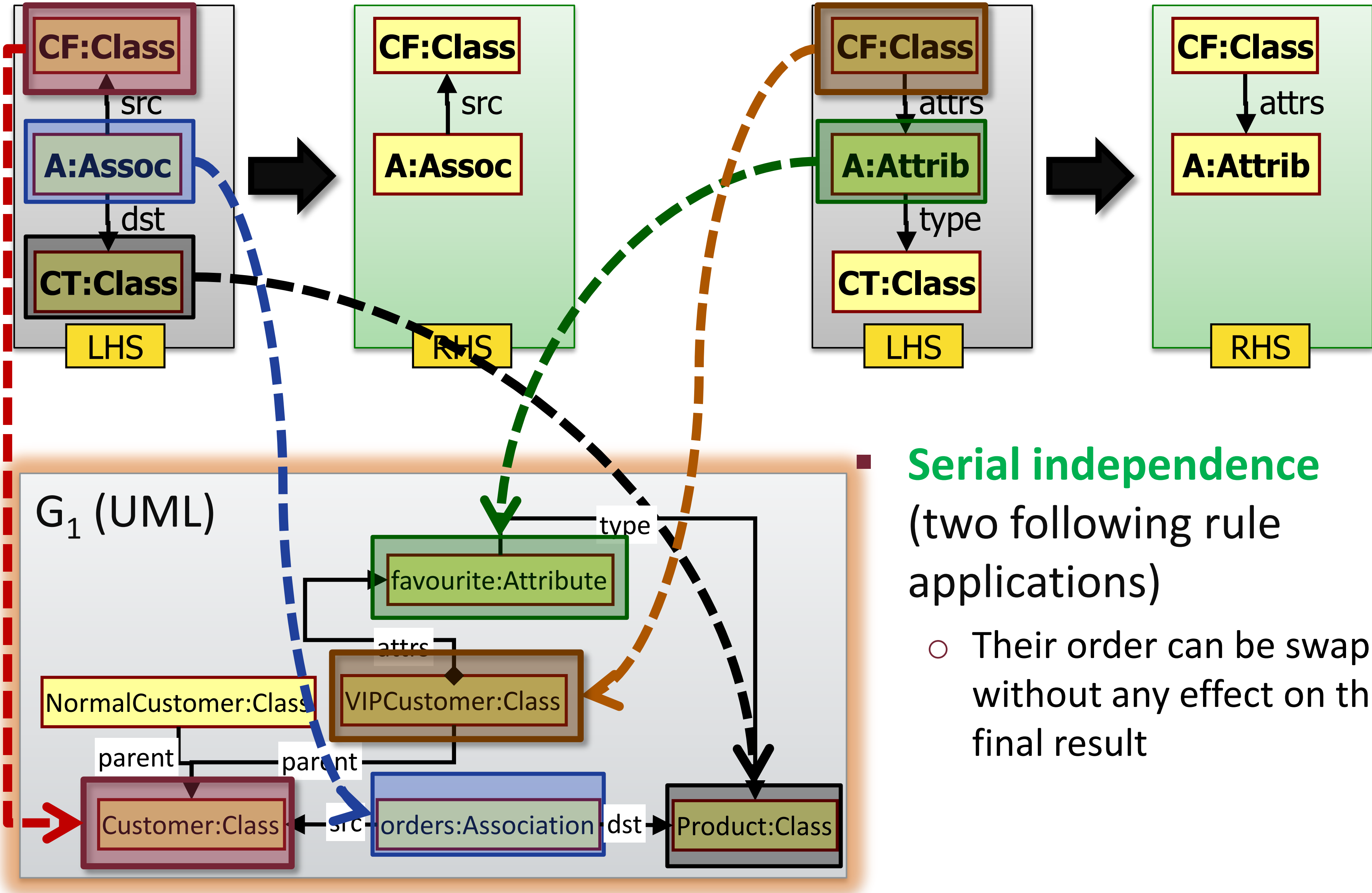
6. Affect of multiple GT rules

Conflict / Parallel independence



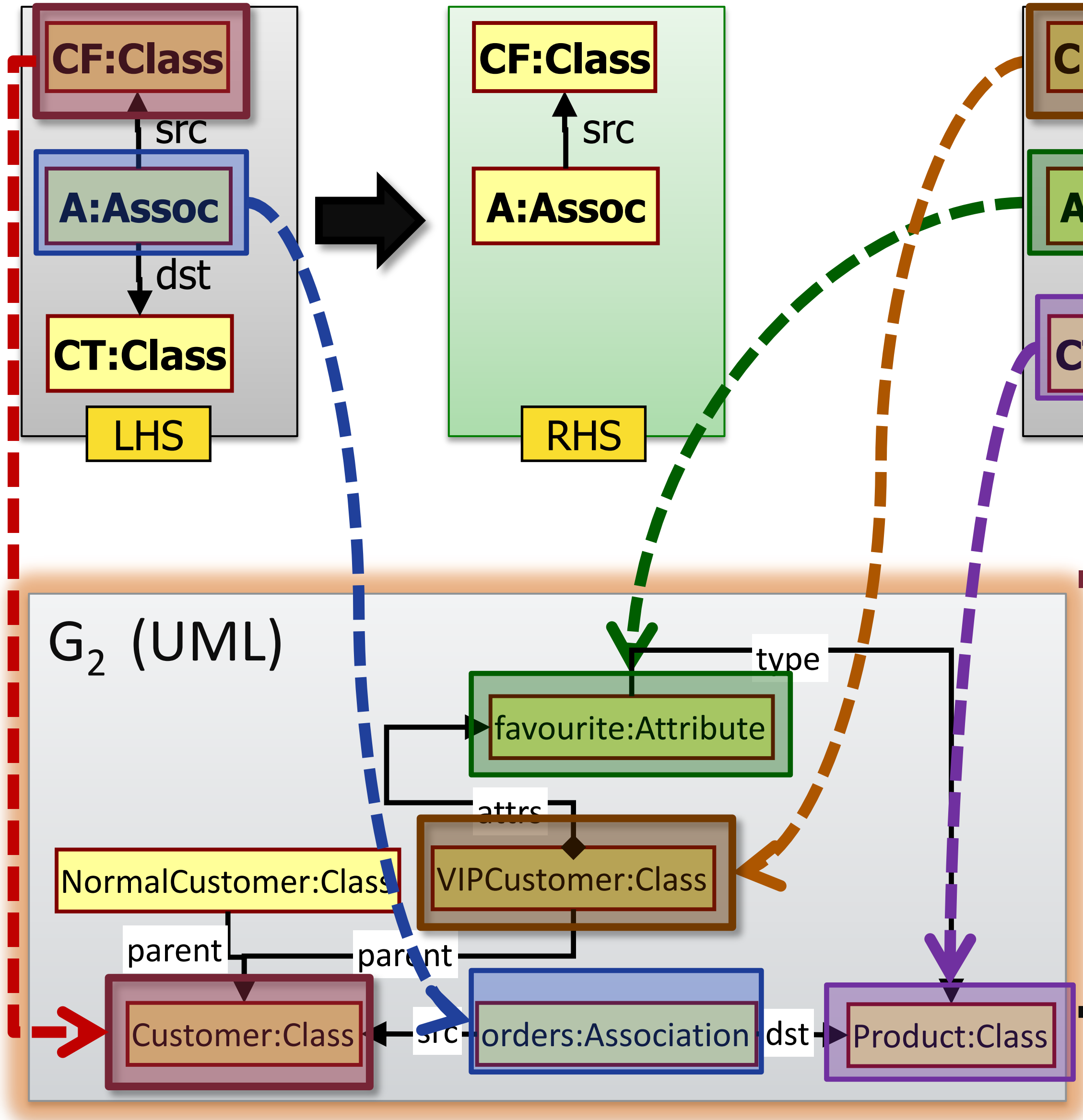
- **Parallel independence** (between two rule applications)
  - Neither affects the application of the other
- **Conflict** (between two rules)
  - If they are not parallel independent
- **Parallel independence** (between two rules)
  - Any two of their rule application are parallel independent

Serial independence



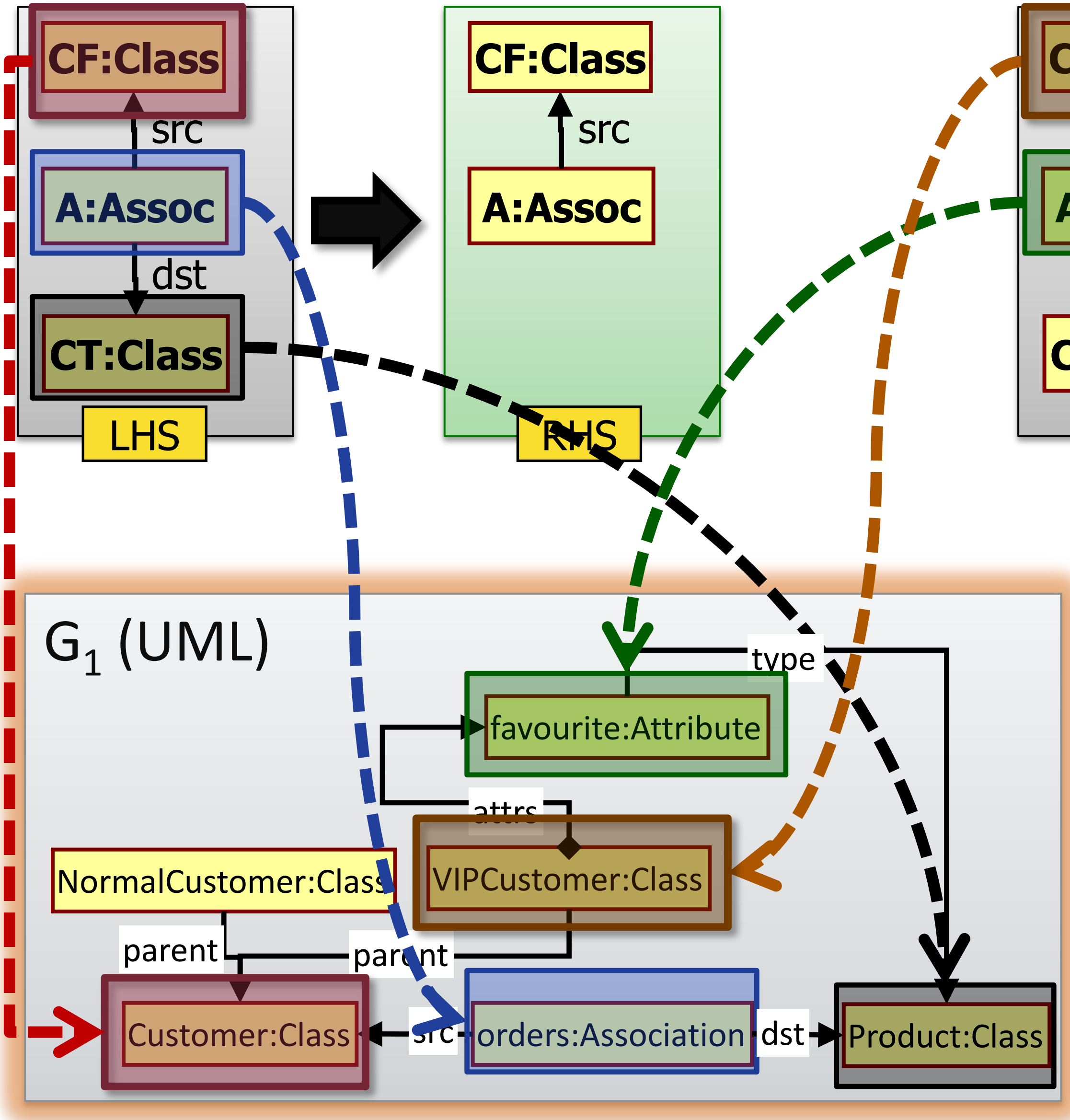
- **Serial independence** (two following rule applications)
  - Their order can be swapped without any effect on their final result

Serial independence



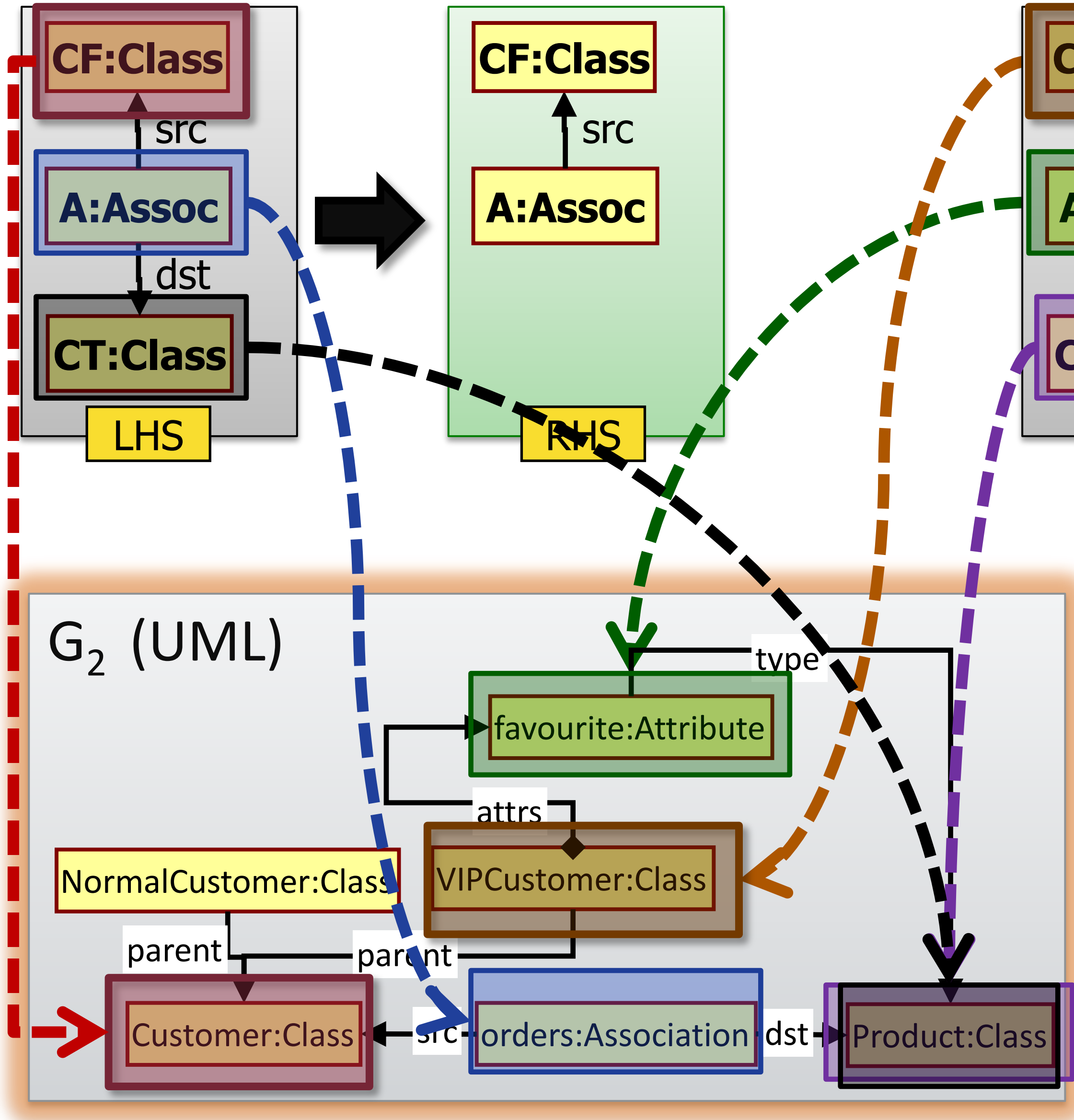
- **Serial independence** (two following rule applications)
  - Their order can be swapped without any effect on their final result
- → Example

Causally dependence I.



- **Serial independence** (two following rule applications)
  - Their order can be swapped without any effect on their final result
- **Causally dependent** (two following rule applications)
  - If they are not serial independent

Causally dependence II.

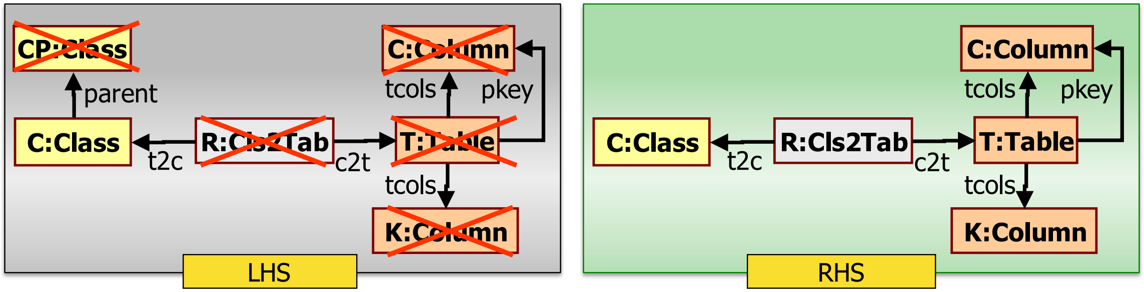


- **Serial independence** (two following rule applications)
  - Their order can be swapped without any effect on their final result
- **Causally dependent** (two following rule applications)
  - If they are not serial independent

→ Example

Summary

- **Graphtransformation,**
  as a modeltransformation paradigm
  - Rule and pattern based formal specification
  - Querying and manipulating graph based models
  - Intuitive graph based specification
- **Structure**
  - LHS graph pattern: precondition
  - RHS graph pattern: postcondition
  - NAC: negative condition
- **Rule application**
  - Graph pattern matching
  - Deletition + Creation
  - Dangling edges and injectivity
  - Affect of multiple rule application (conflicts and causality)



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

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
  - Viatra2

Other - XSLT

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

Implementing a Graph Transformation Engine

Implementing GT engines

- Key elements
  - Model Store
    - Storing typed graphs
    - Support easy import and export
  - Pattern Matching
    - Find match for LHS
  - Model manipulation
    - Fast model manipulation
    - Rollback
    - Notification

Pattern matching techniques

- Categories
  - Interpreted: AGG (Tiger), VIATRA, MOLA, Groove, ATL
    - underlying PM engine
  - Compiled: Fujaba, GReAT, PROGRES, Tiger
    - directly executed as a C or Java code (no PM engine)
- Base algorithms
  - Constraint satisfaction: AGG (Tiger)
    - variables + constraints
  - Local search: Fujaba, GReAT, PROGRES, VIATRA, MOLA, Groove, Tiger (Compiled)
    - step-by-step extension of the matching
  - Incremental: VIATRA, Tefkat
    - Updated cache mechanism

Constraint satisfaction based Pattern Matching

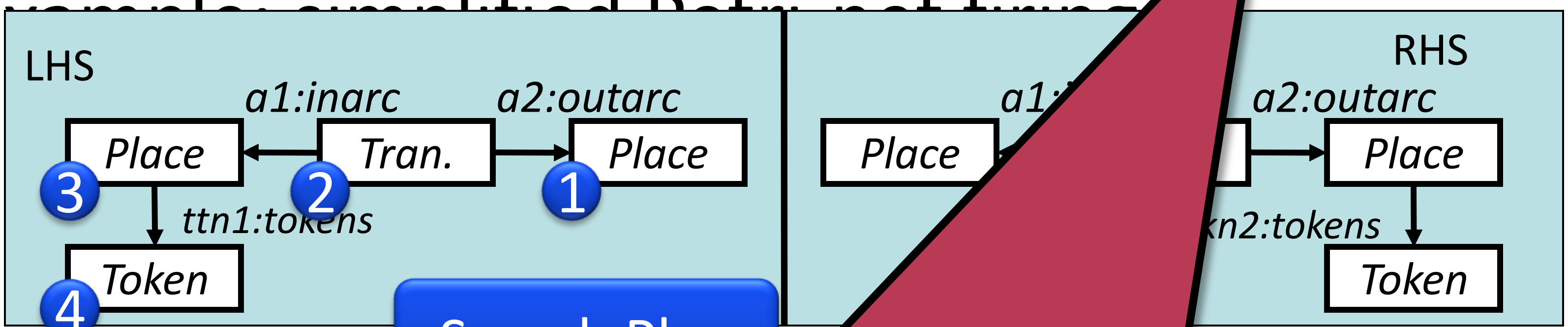
- Realization:
  - Nodes are handled as CSP variables
  - Constraints derived from edges
  - Type information as domain reduction
  - Traversal: backtracking algorithm
- Pros:
  - Adaptive algorithm
- Contras:
  - Handling large models
  - Scalability

Local Search based Pattern Matching

- Method
  - usually defined in design/compile time
  - simple search plan
  - hard wired precedence for constraint checking (NAC, injectivity, attribute, etc.)
- Good performance expected when:
  - Small patterns, bound input parameters

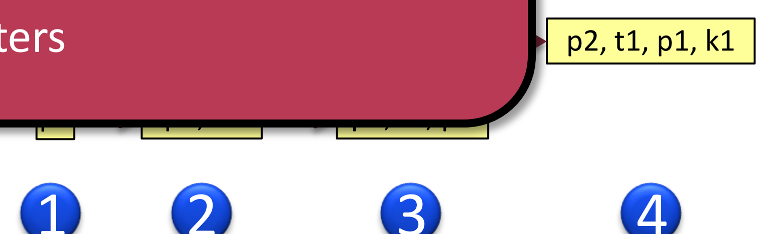
Pattern Matching: Local Search

- PM can be the most time-consuming part
- Most implementations perform **local search**
- Example: simplified Petri net firing



- Fujaba, GReAT, PROGRES, Groove, Tiger, GrGEN.NET…
- VIATRA2 also has a LS-based pattern matcher
- Good performance expected:
  - Small patterns, bound input parameters

st)

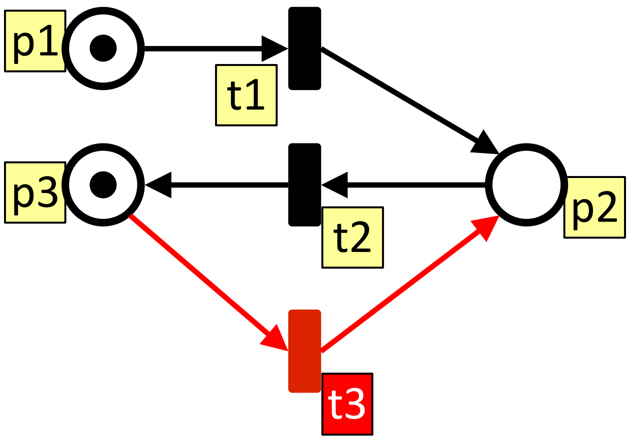


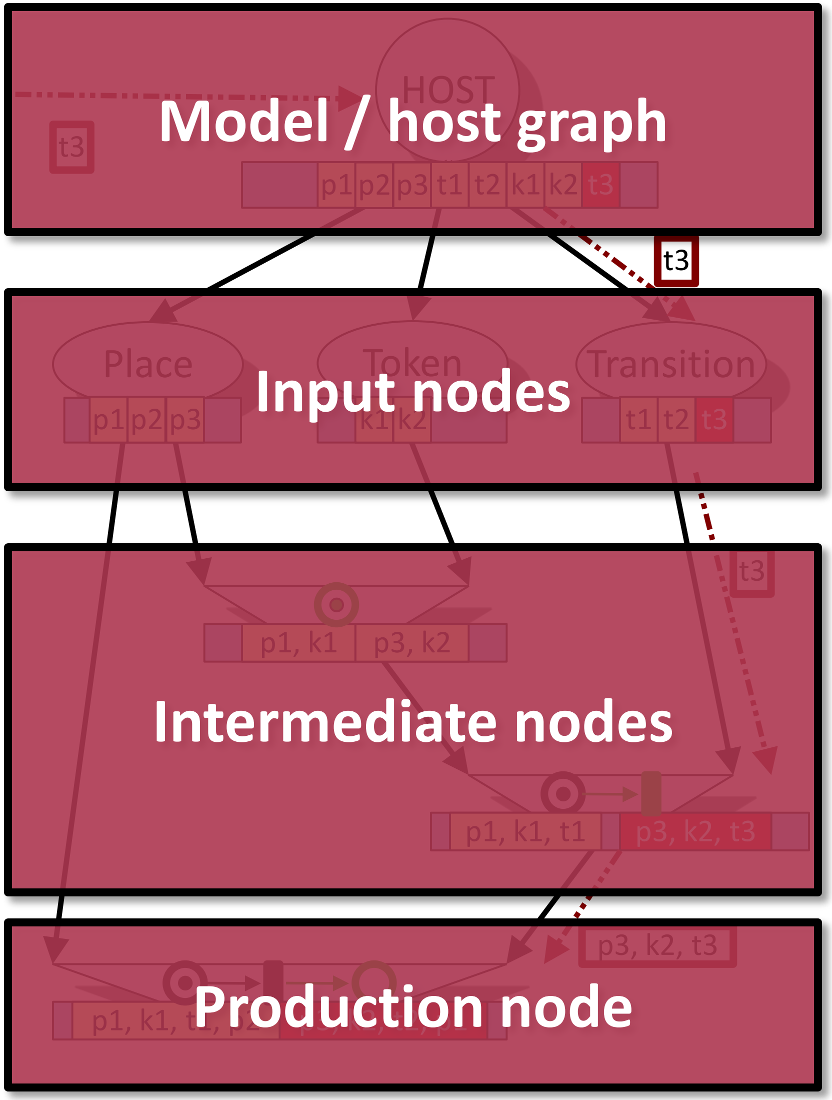
Incremental Pattern Matching

- Goal
  - **Store matching sets**
  - Incremental update
  - Fast response
- Good performance expected when:
  - frequent pattern matching
  - Small updates
- Possible application domain
  - E.g. synchronization, constraints, model simulation, etc.
- Example implementation (VIATRA): an adapted RETE algorithm

Incremental Pattern Matching by RETE

- RETE net
  - node: (sub)pattern
  - edge: change propagation
- Demostrating the principle
  - input: Petri-net
  - pattern: fireable transition
  - change: new transition





Hybrid pattern matching

- Combine local search-based and incremental pattern matching
- Motivation
  - Incremental PM is better for most cases, but…
    - Has memory overhead!
    - Has update overhead
  - → LS might be better in certain cases
    - Memory consumption (cache size)
    - Cache construction time penalty (overhead, simple navigation patterns)
    - Expensive updates (e.g., move operation)

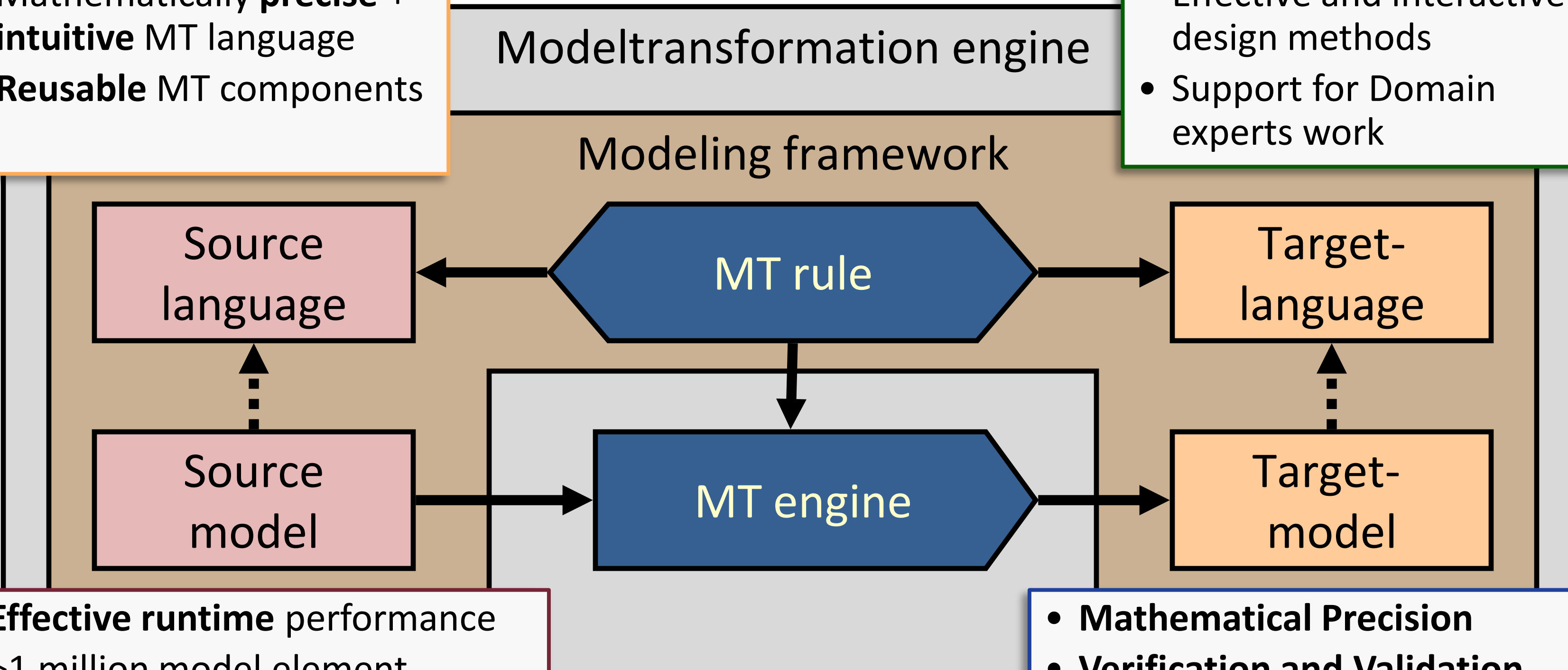
Our research: Design and analysis of modeltransformation

Specification of Modeltransformation

- Mathematically **precise** + **intuitive** MT language
- **Reusable** MT components

Design of Modeltransformation

- Effective and interactive design methods
- Support for Domain experts work



Application of Modeltransformation

- **Effective runtime** performance
- >1 million model element
- **IDE** integration

Verification of Modeltransformation

- **Mathematical Precision**
- **Verification and Validation** techniques