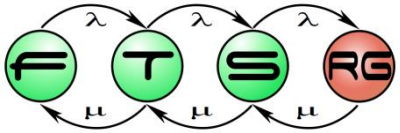
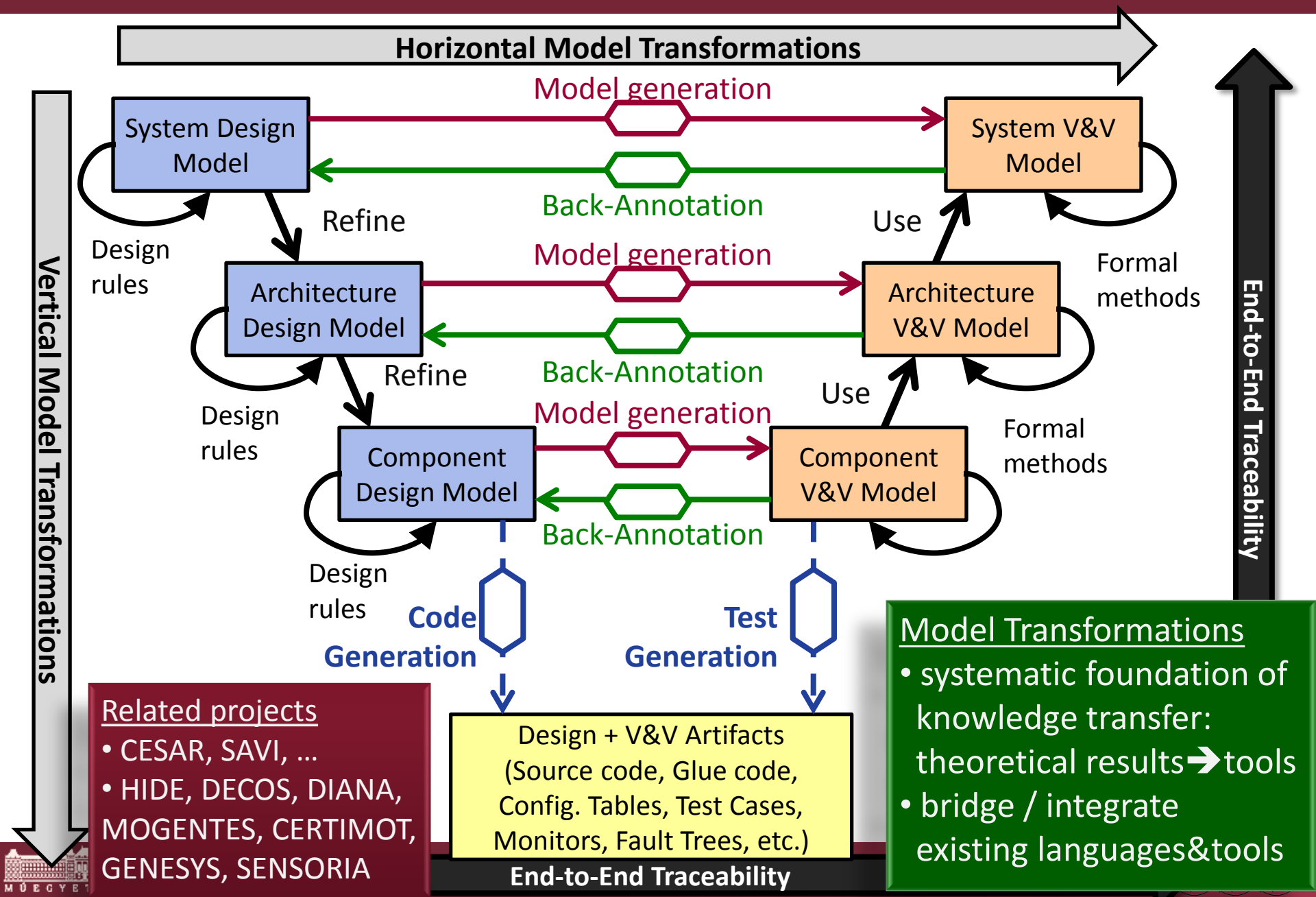
Foundations of Model Transformation

Model Driven Systems Development
Lecture 9-10



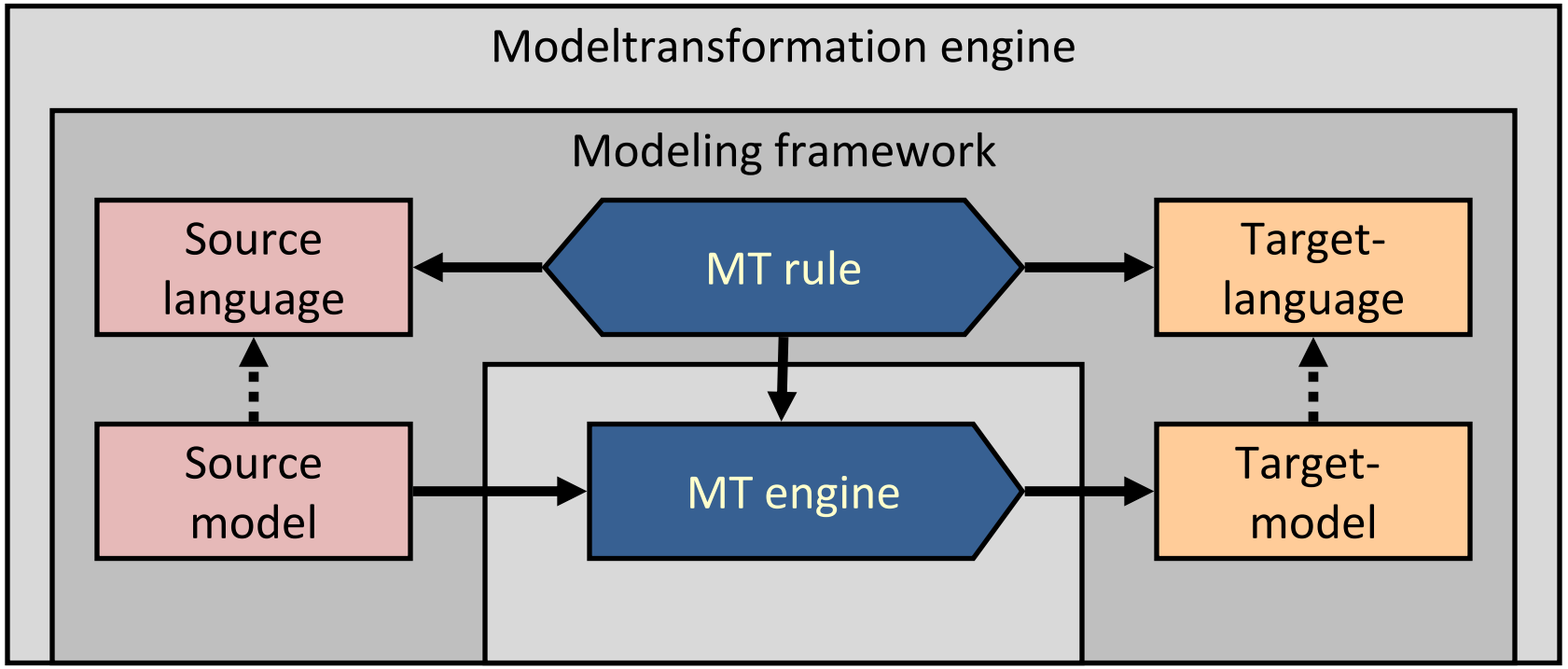
Models and Transformations in Critical Systems



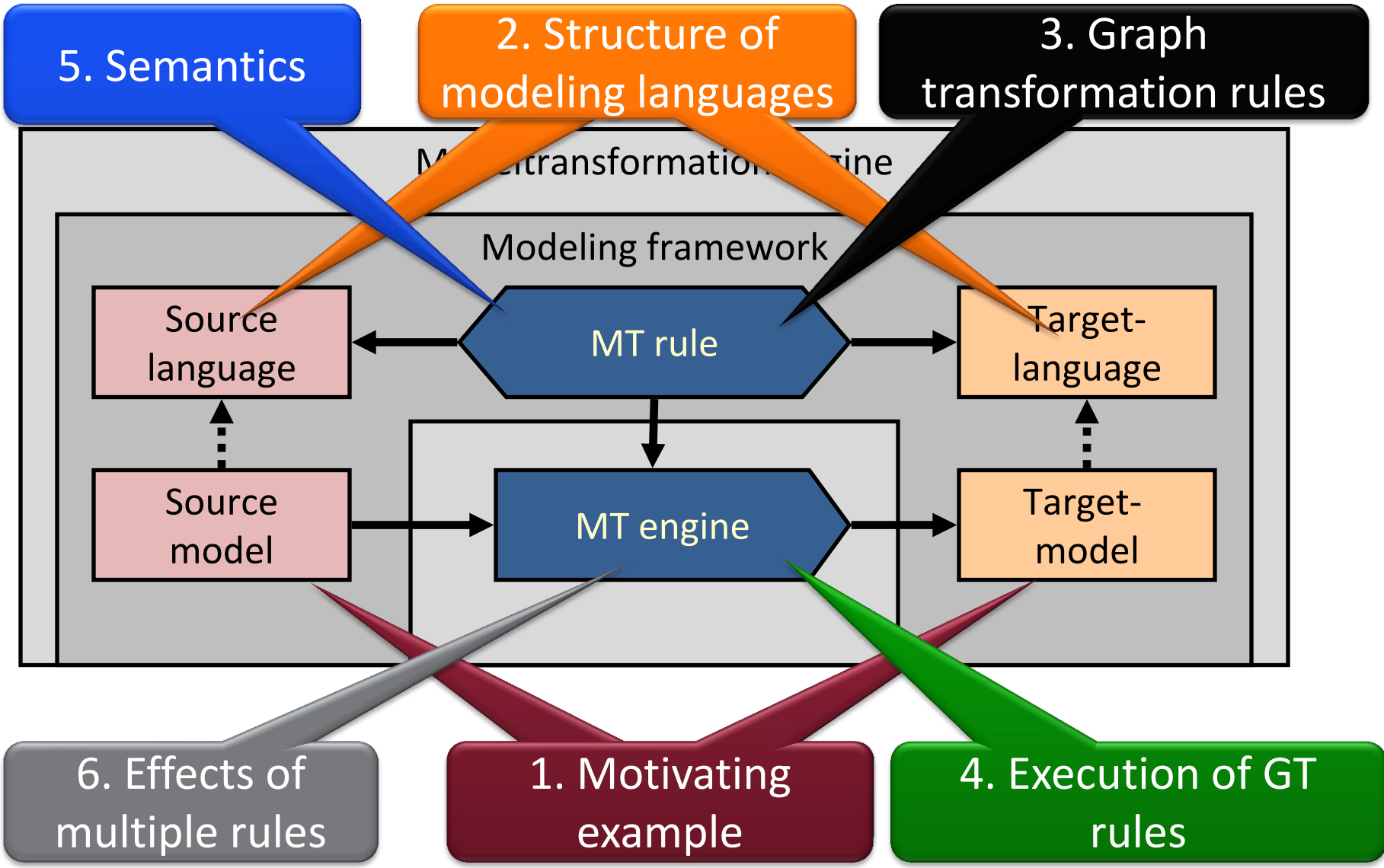
Related projects
- CESAR, SAVI, …
- HIDE, DECOS, DIANA, MOGENTES, CERTIMOT, GENESYS, SENSORIA

Model Transformations
- systematic foundation of knowledge transfer: theoretical results➔tools
- bridge / integrate existing languages&tools

Definition of Model Transformation



Overview

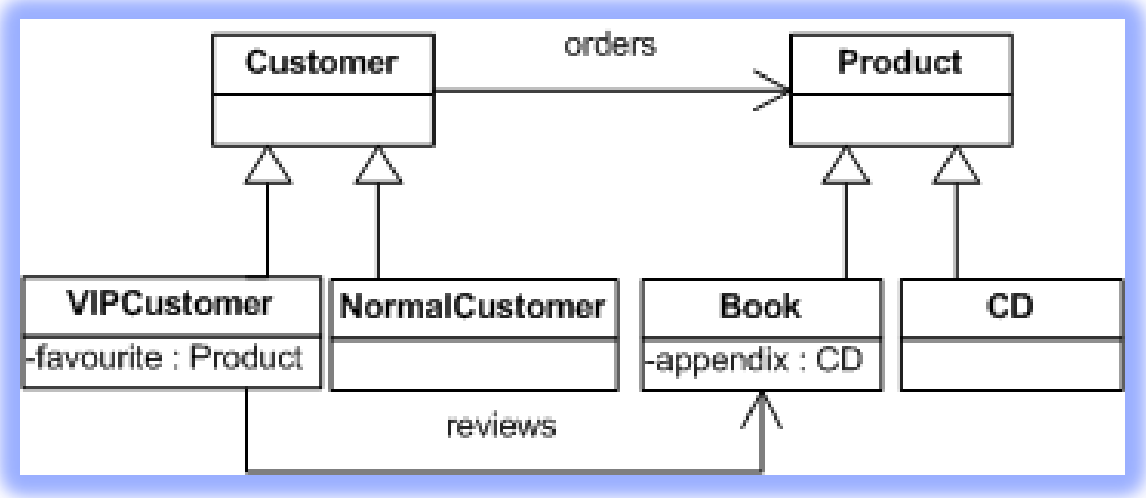

5. Semantics
2. Structure of modeling languages
3. Graph transformation rules
Modeling framework
Source language
MT rule
Target-language
Source model
MT engine
Target-model
6. Effects of multiple rules
1. Motivating example
4. Execution of GT rules

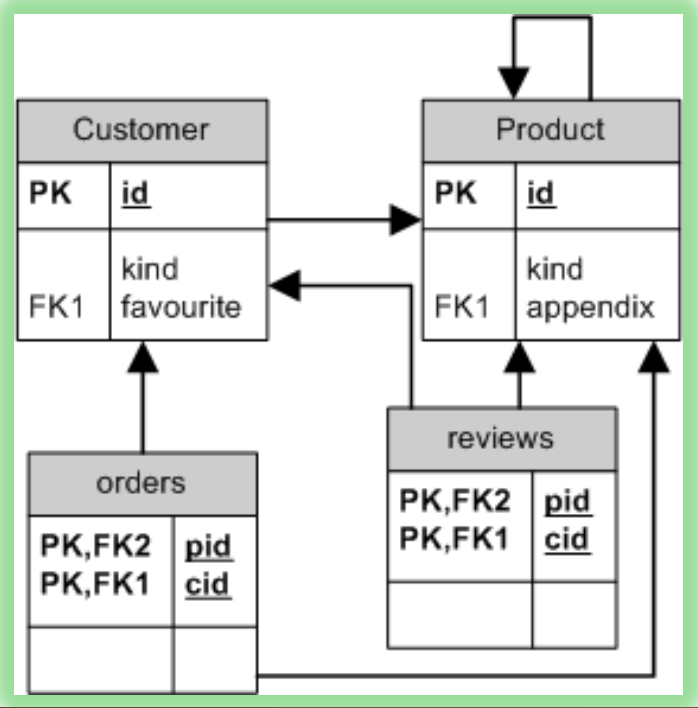
1. Motivating Example

Object Relational Schema mapping

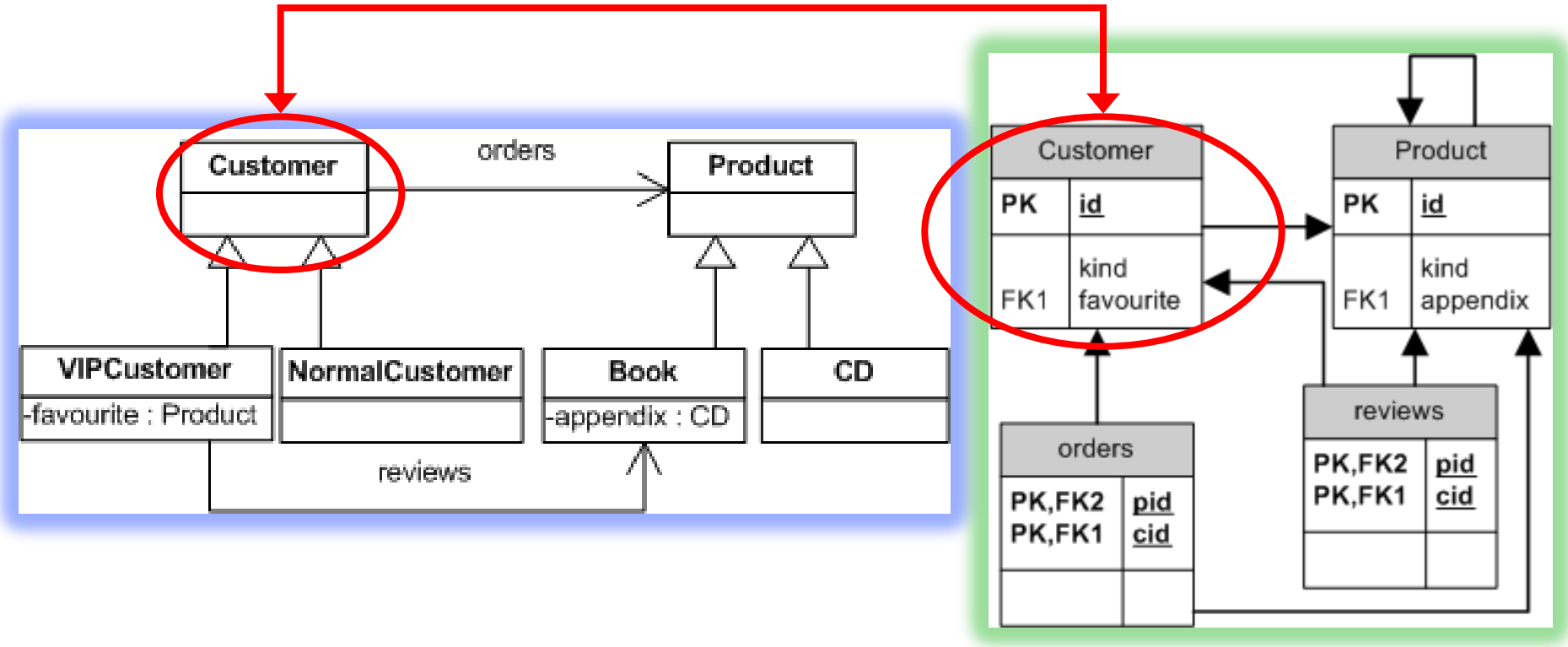
Example: Object-relational maping

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





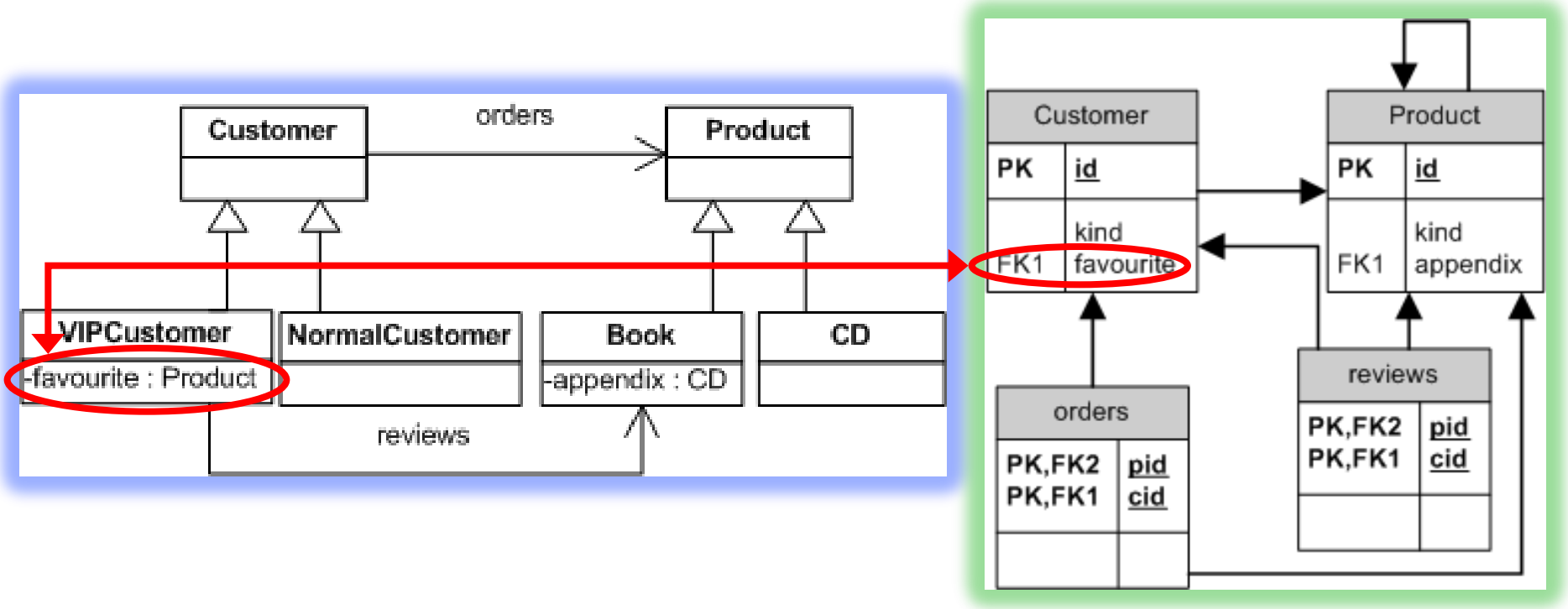
Informal definition of the MT rules of the mapping



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

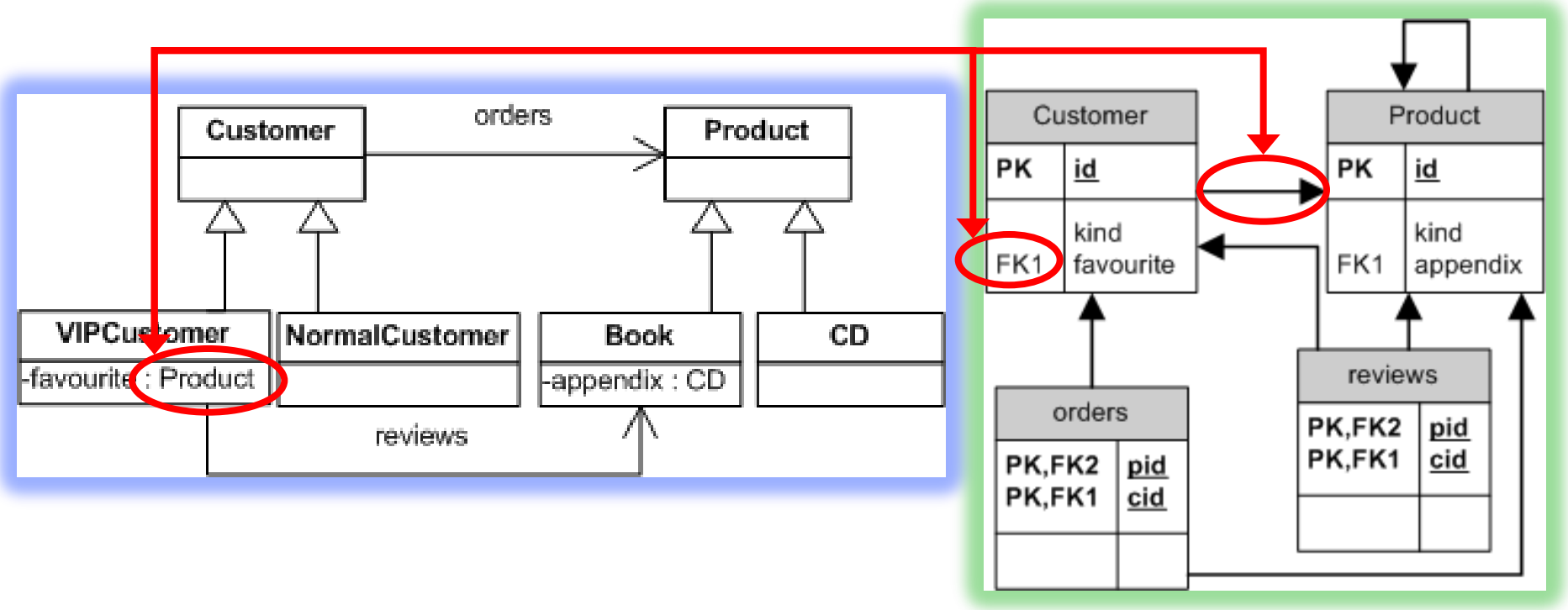
- Unique identifier (primary key),
- type definition

Informal definition of the MT rules of the mapping



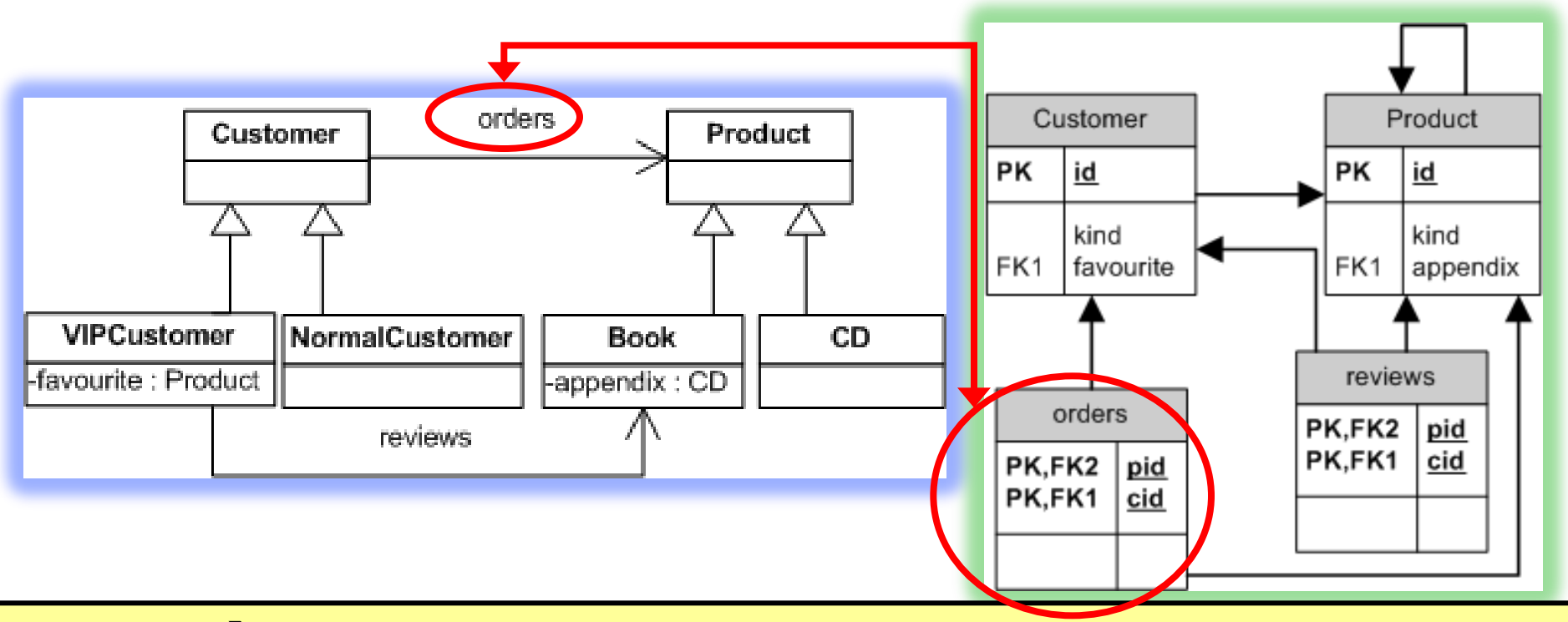
Class attributes ➔ (contained by the topmost classes) Column of the table

Informal definition of the MT rules of the mapping



Type of the attributes ➔ foreign key

Informal definition of the MT rules of the mapping



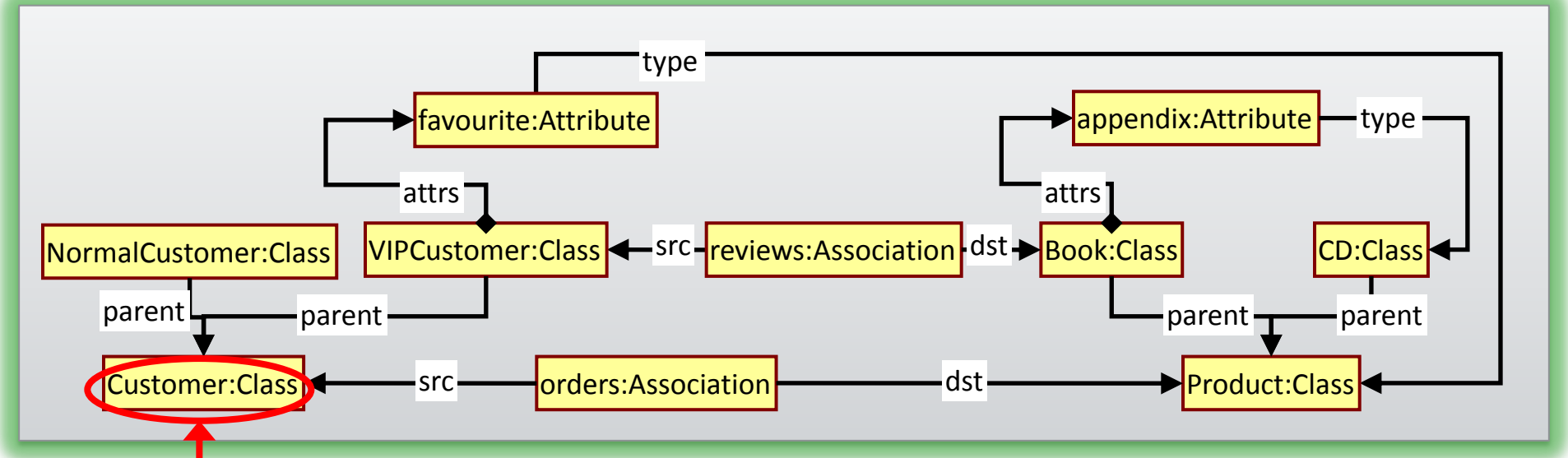
Association ➔ A table with two columns

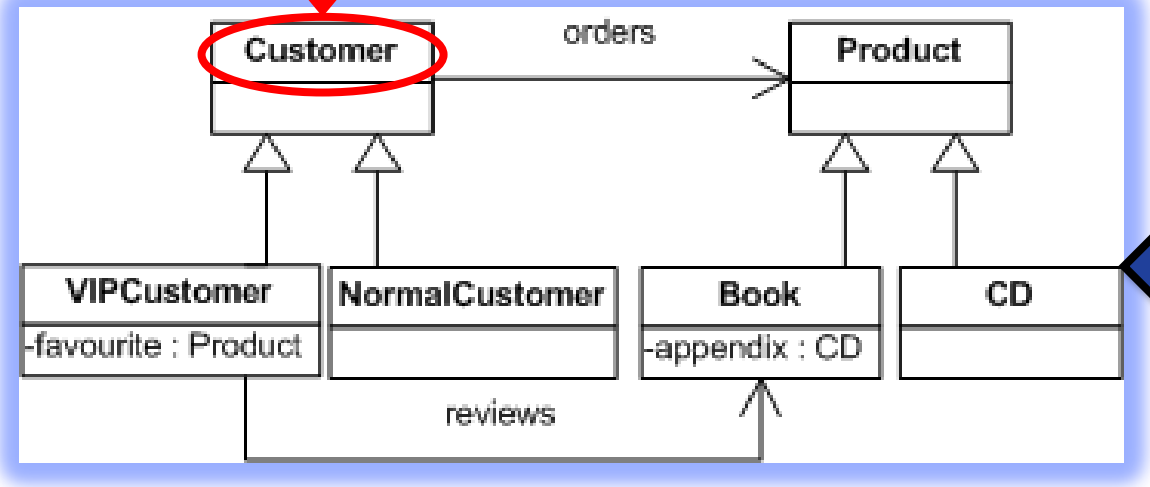
- source and target identifiers
- foreign keys (for consistency)

2. Structure of Modeling Languages

Revision

Structure of Modeling languages (UML)





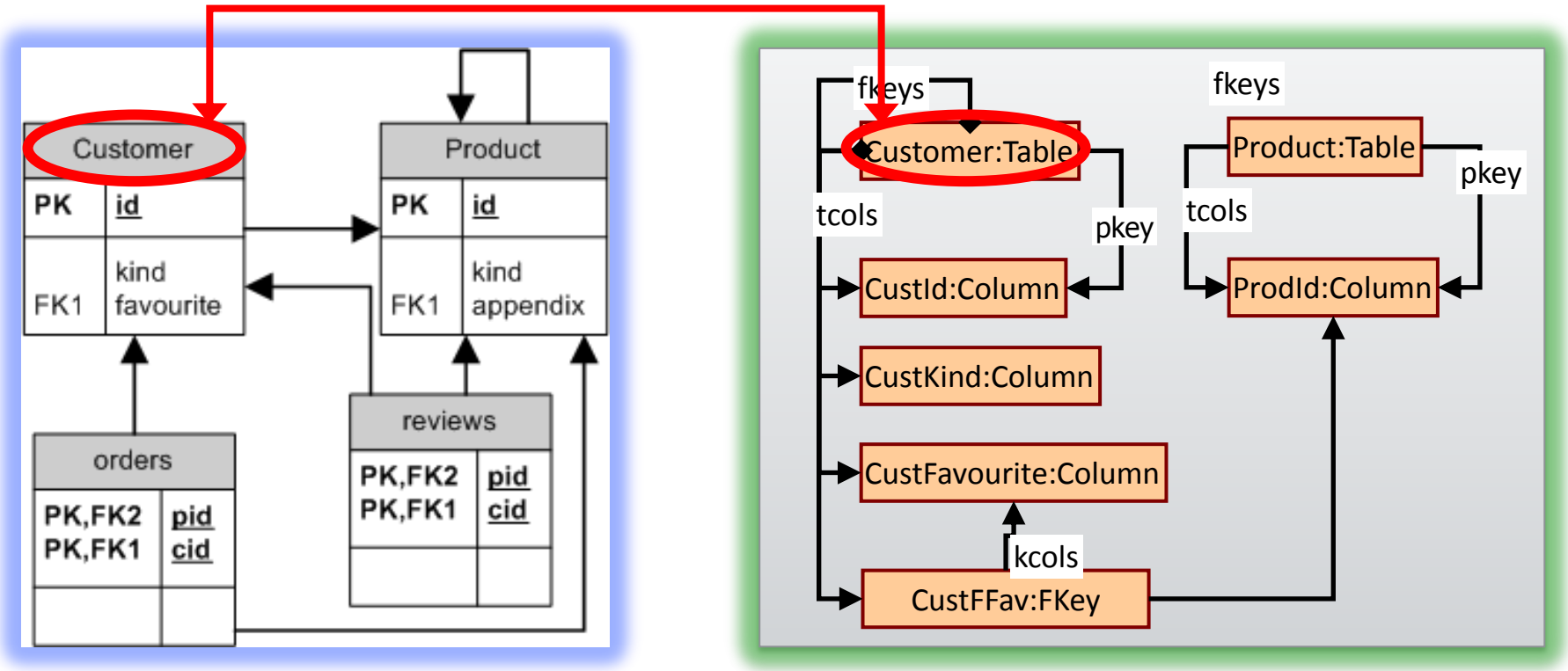
Abstract syntax

- Graph based model representation
- Machine readable

Concrete syntax

- Visual/textual representation
- Human readable

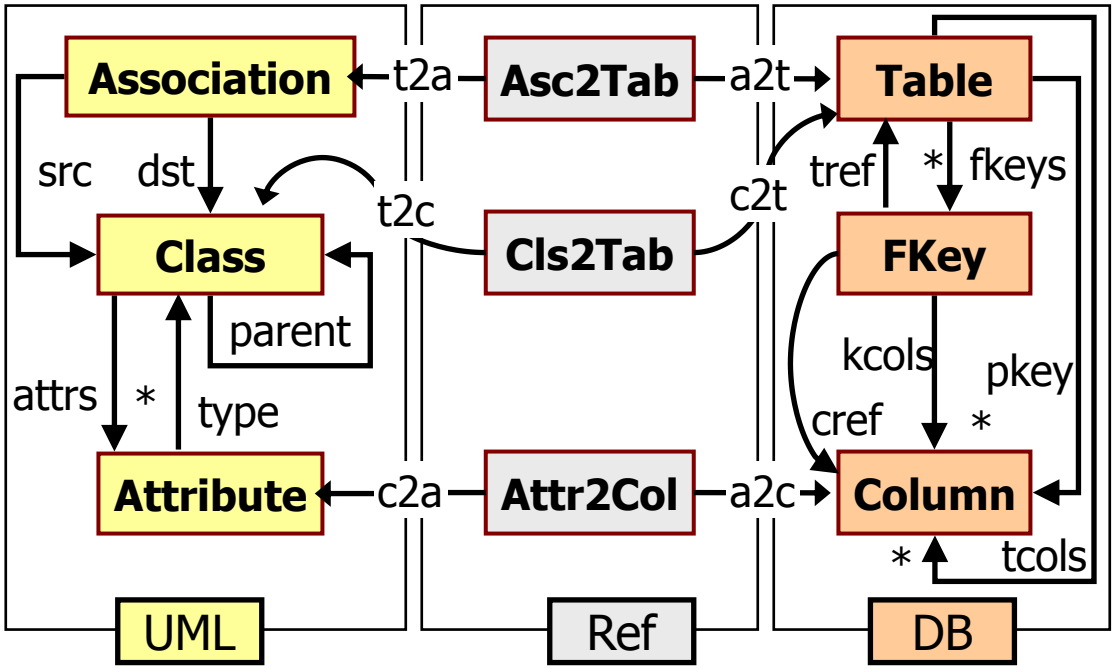
Structure of Modeling languages (RDBMS Schema)



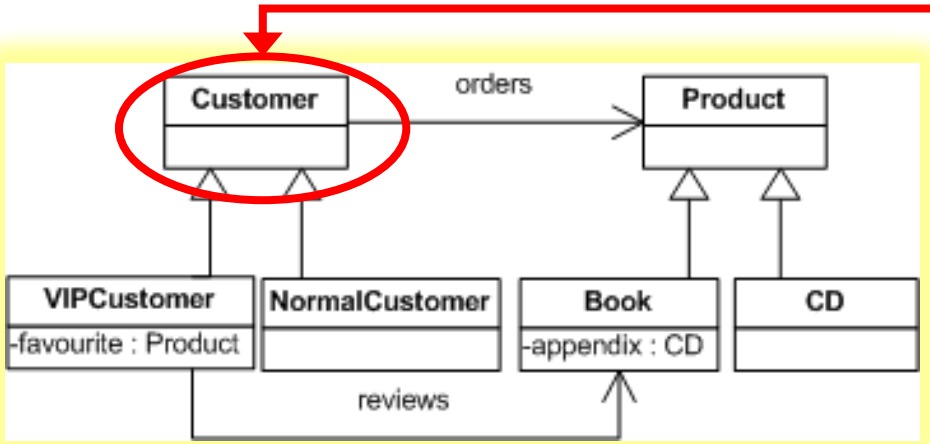
Concrete syntax

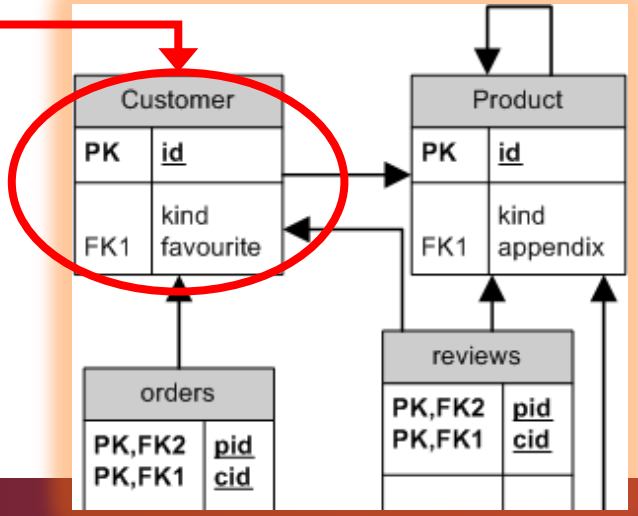
Abstract syntax

Metamodel of the O-R mapping



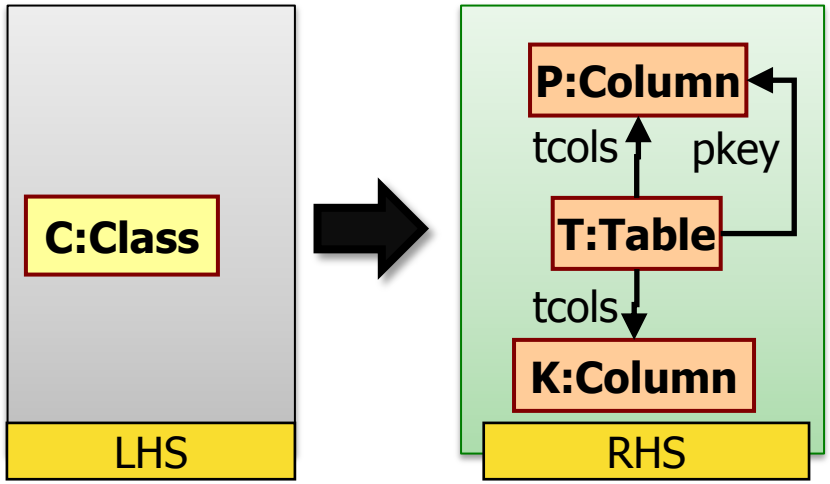
- Source + Target metamodel
- Traceability metamodel:
  - For saving the relations between the source and the target languages
- Motivation: critical embedded systems
  - Traceability
  - Requirement ➔ Source code





3. Graph Transformation Rules

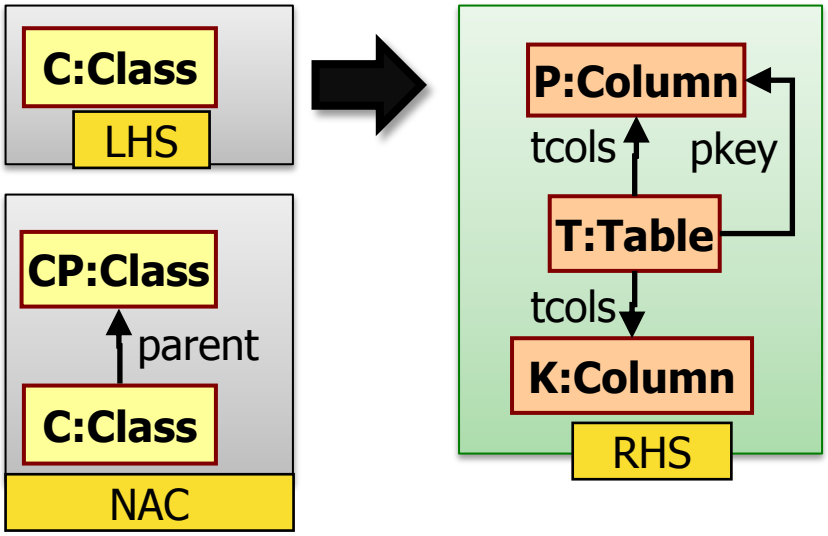
Structure of a GT rule



- **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)

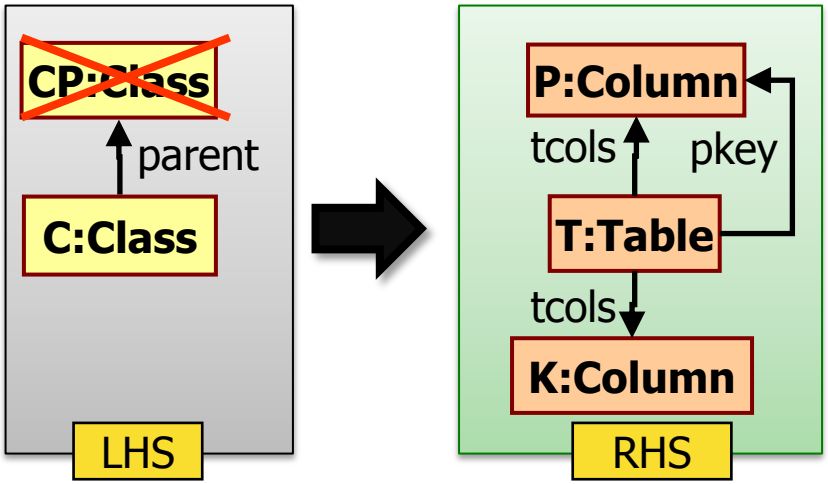
Structure of a GT rule



- **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

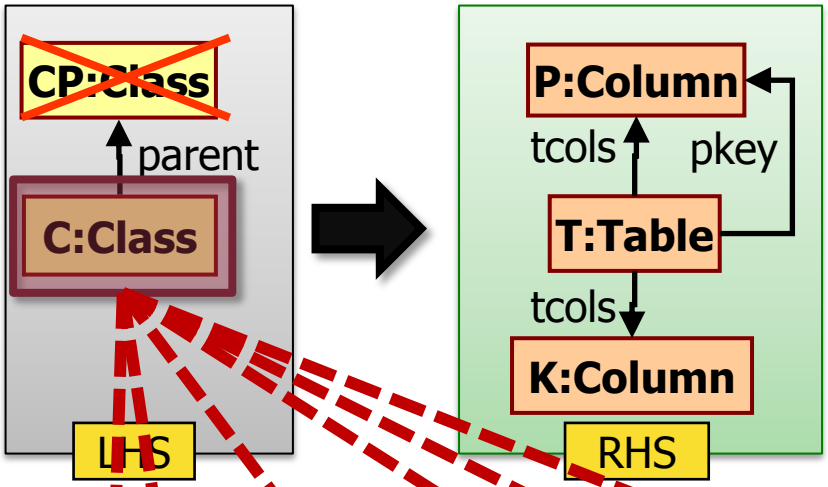


- **Graph Transformation (GT)**:
  - Declarative and formal paradigm
  - Rule base transformation
  - Match of the LHS➔ Image of the RHS
  - Generalization of Chomsky grammars (hierarchy) (text ➔ graph)

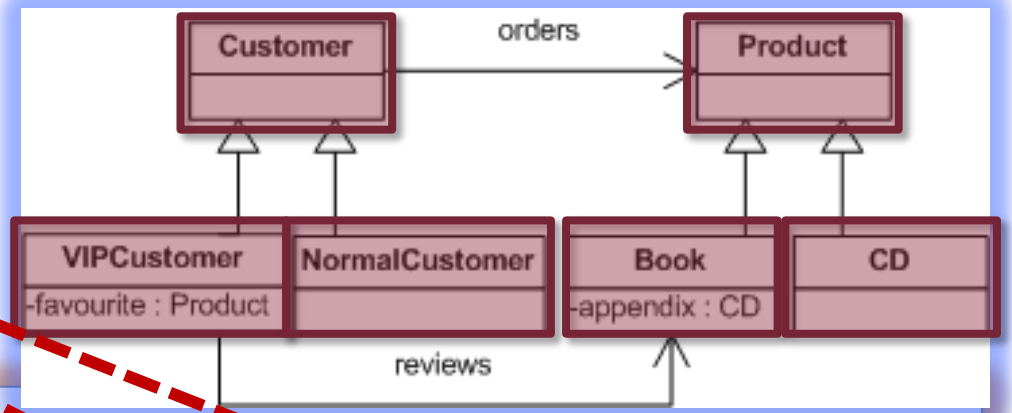
- **Graph Transformation Rules**
  - **Left hand side** - LHS
    - Graph pattern
    - Precondition for the rule application
  - **Right hand side** - RHS:
    - Graph pattern + LHS mapping
    - Declarative definition of the rule application
      - What we get (and not how we get it)
  - **Negative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

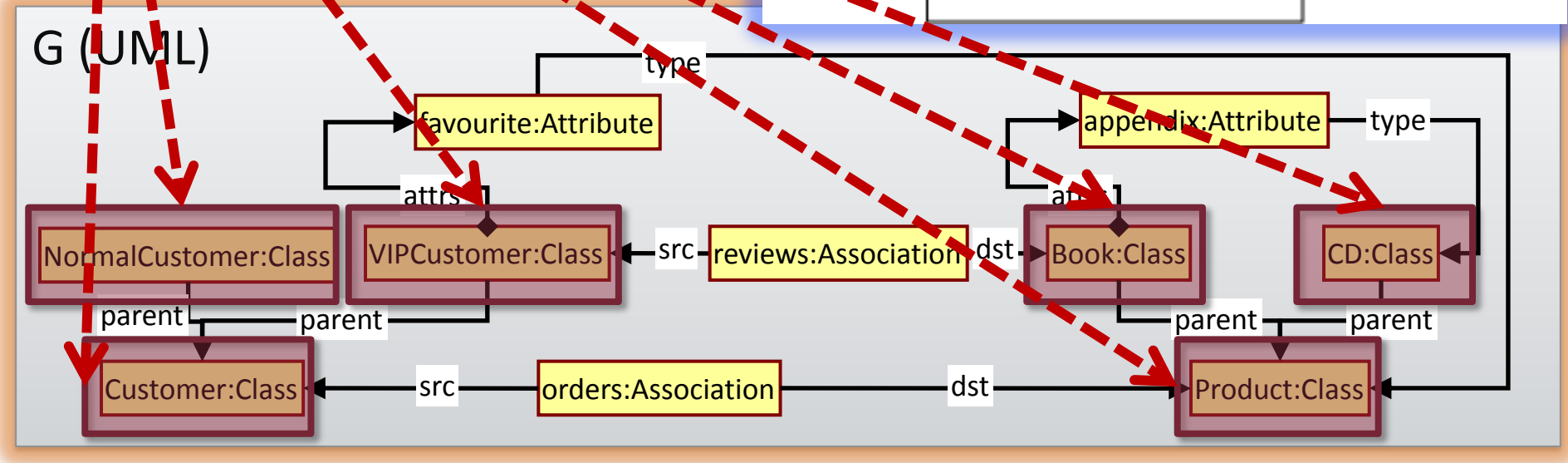
4. Application of Graph Transformation Rules

Application of GT rules

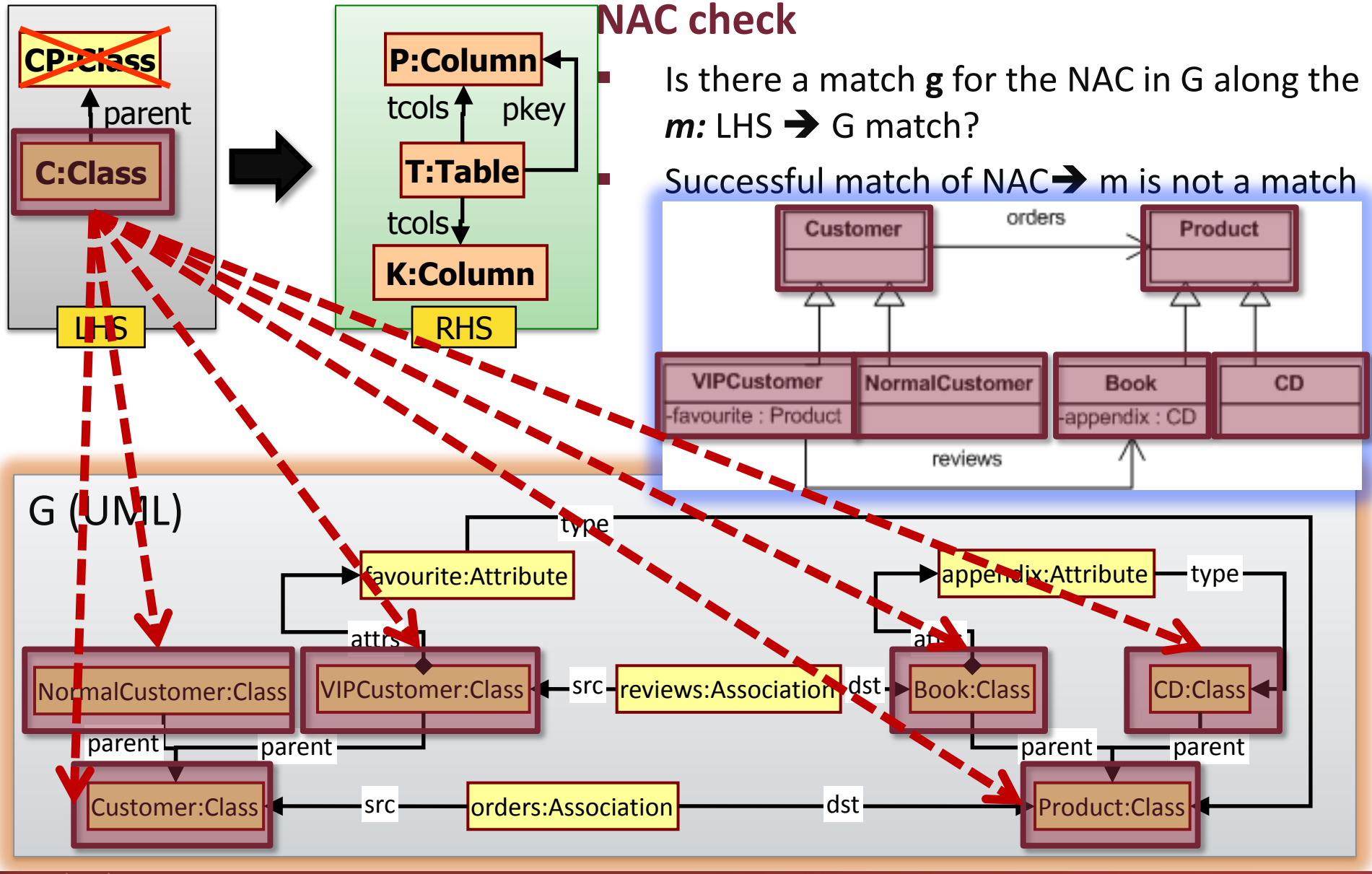


1. **Graph pattern matching**
   - Match of the LHS pattern in the underlying model
   - match ***m***: LHS ➔ G mapping

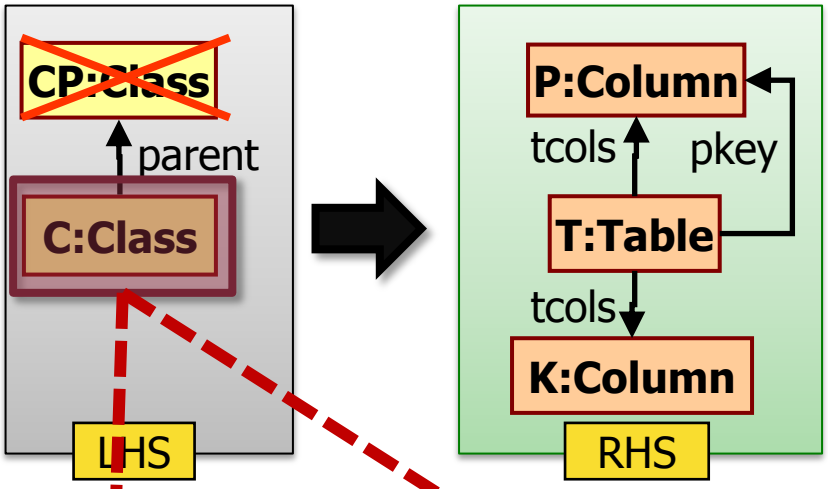




Application of GT rules

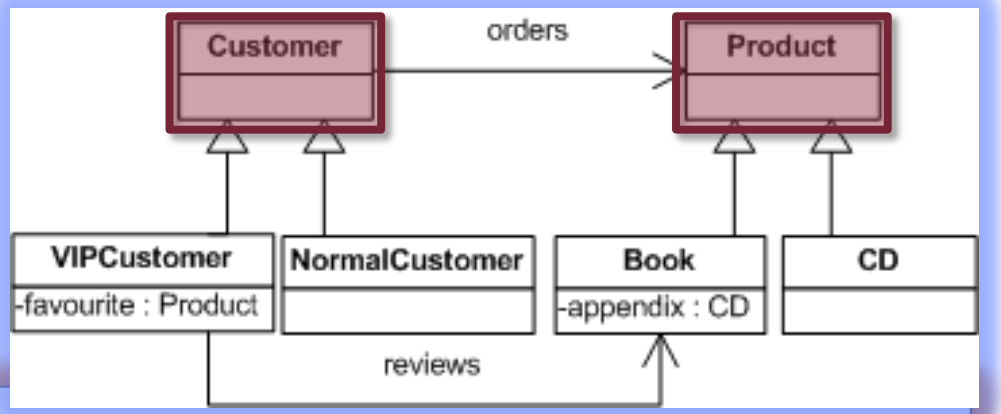


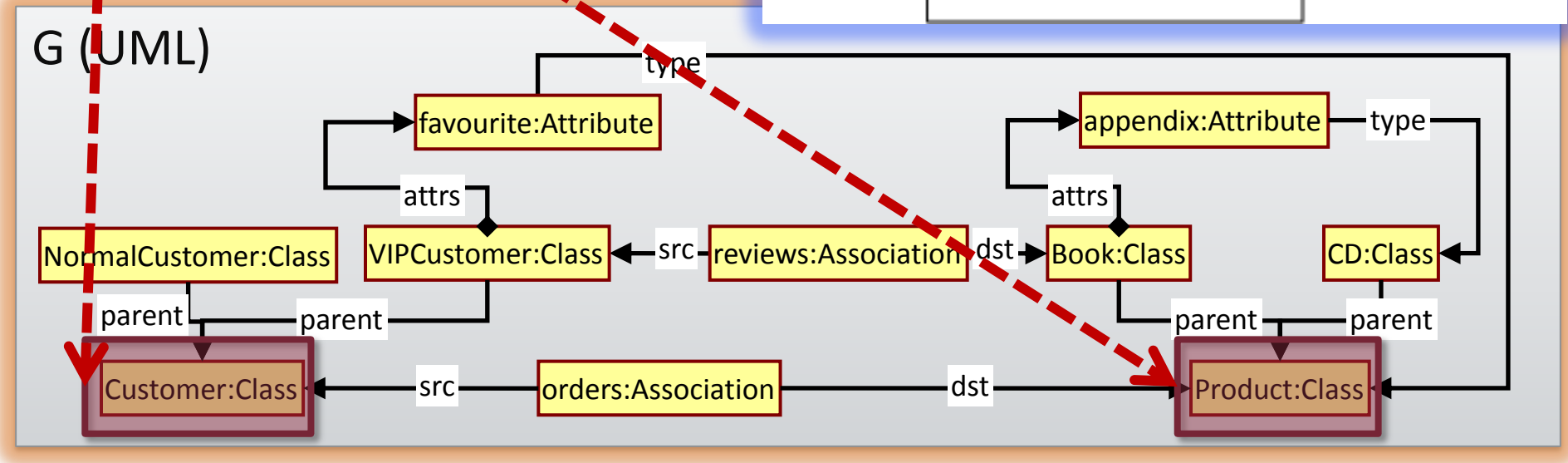
Application of GT rules



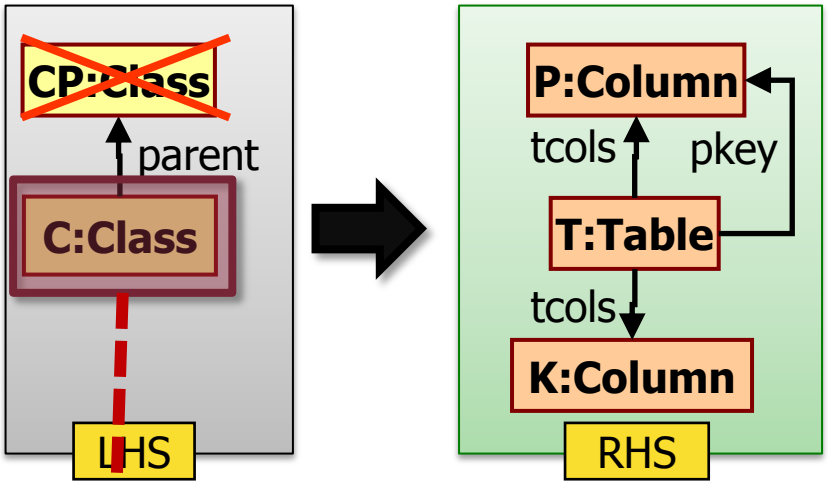
3. Non-deteministic selection

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



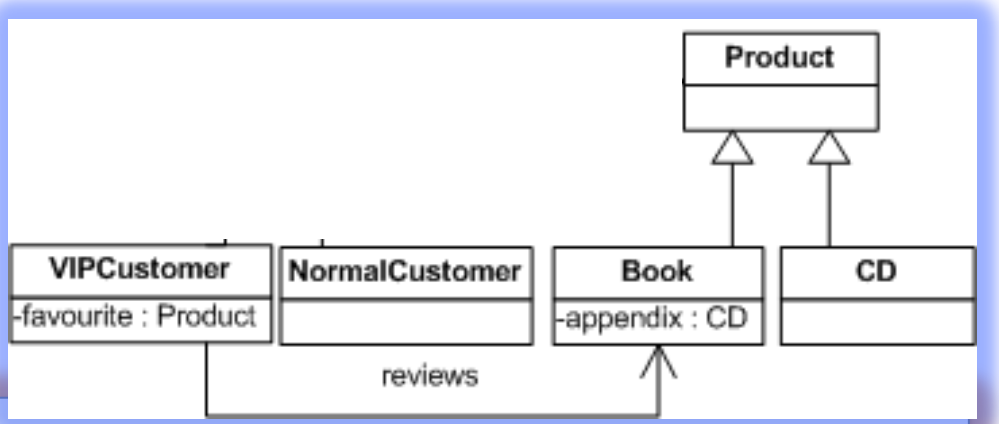


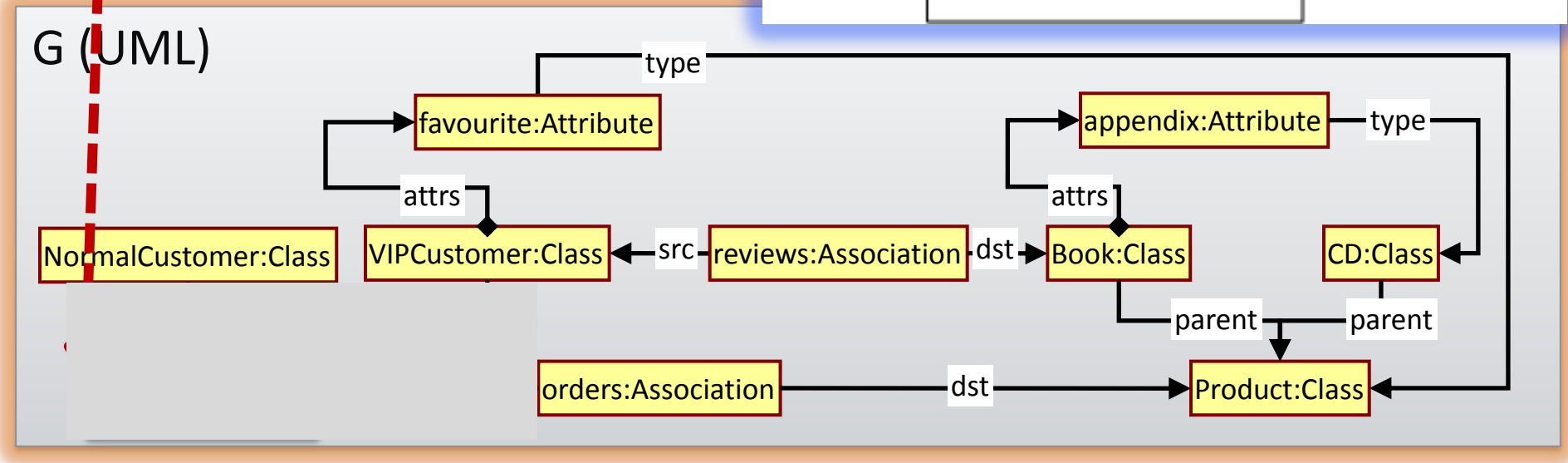
Application of GT rules



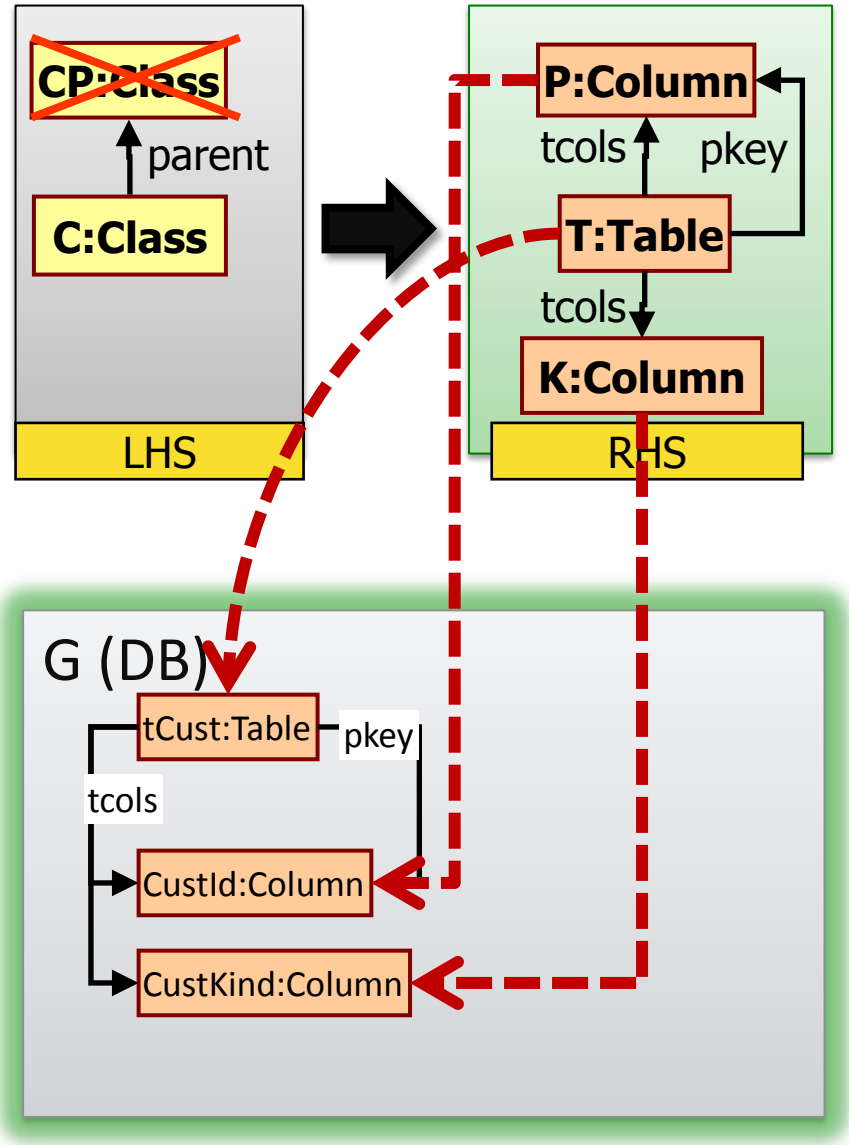
4. Deletion

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





Application of GT rules



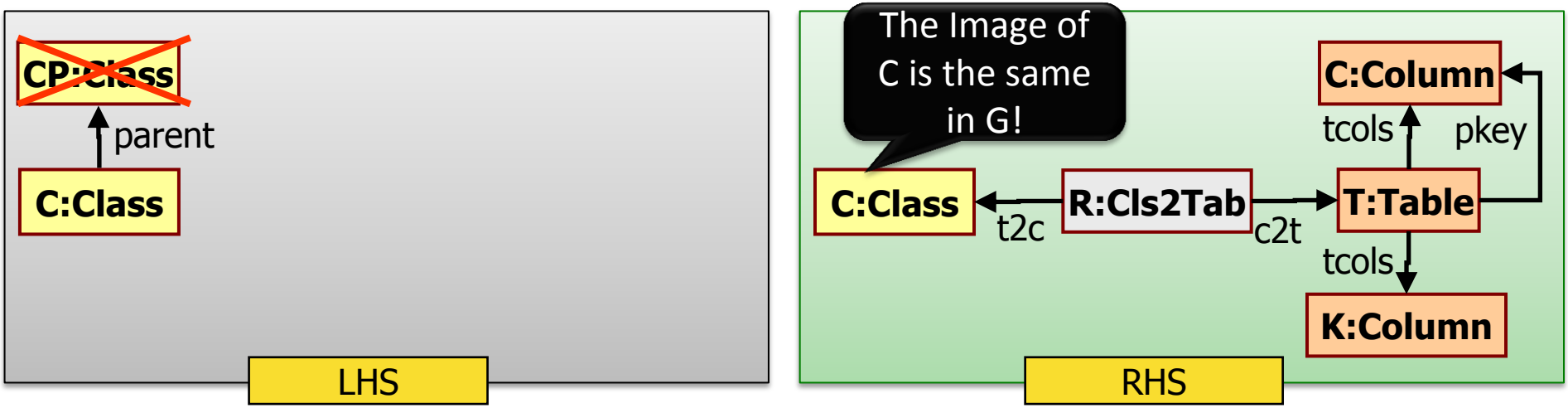
5. Creation (and binding)

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

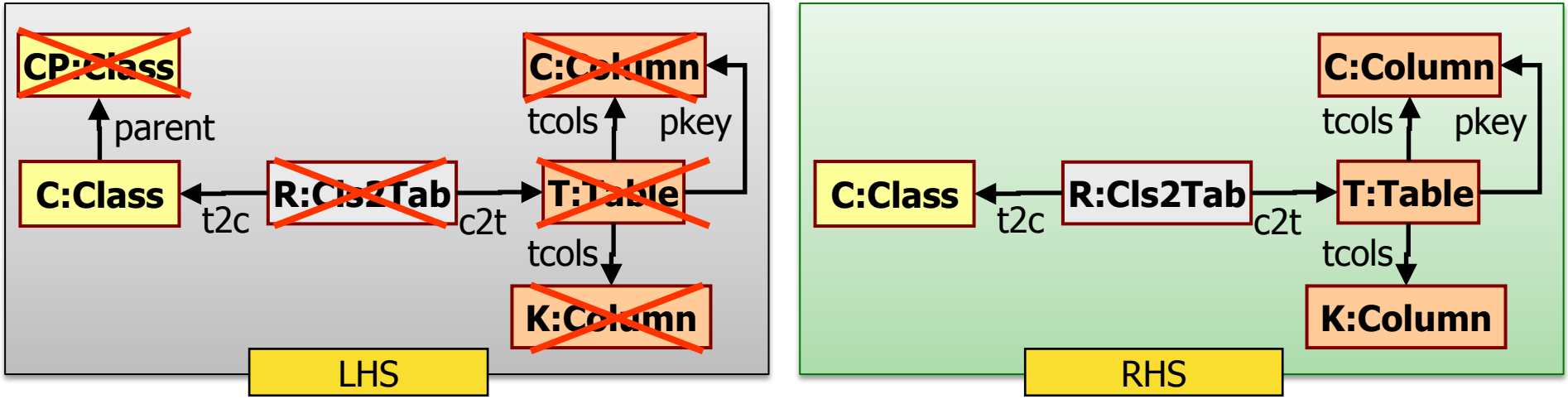
Customer	
PK	id
	kind

Typical problems…

1) Saving the source model, traceability

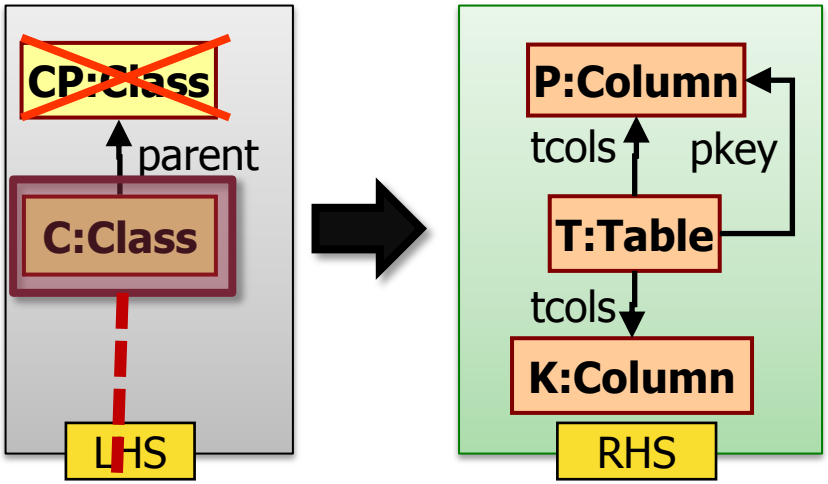


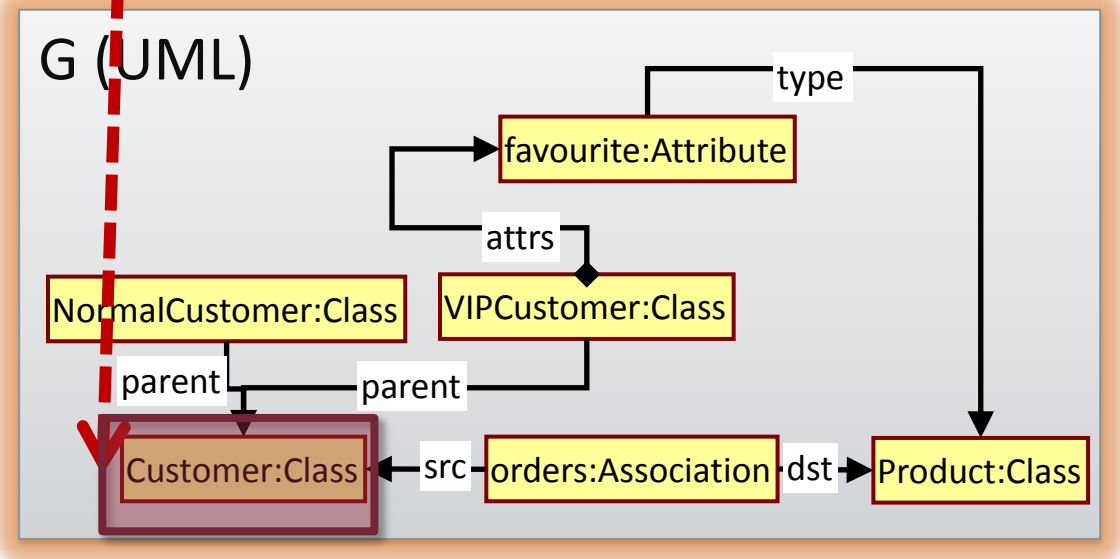
2) Application of the same rule along the same match



5. Different Semantics

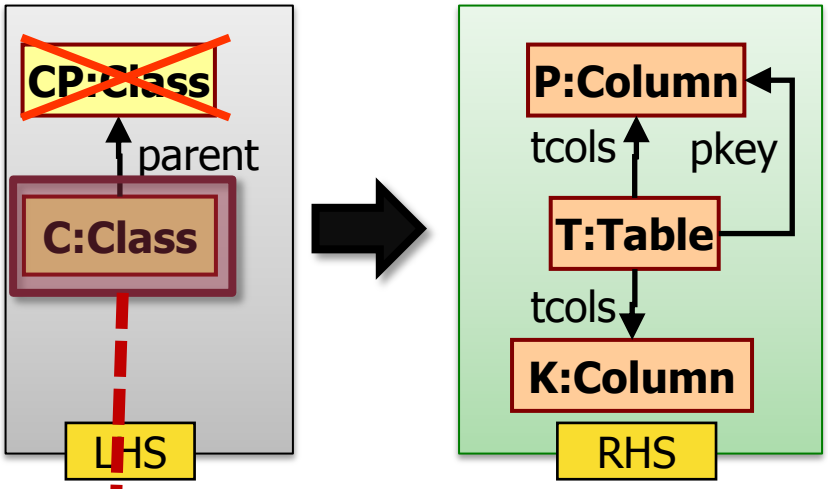
Semantics : Handling of Dangling edges

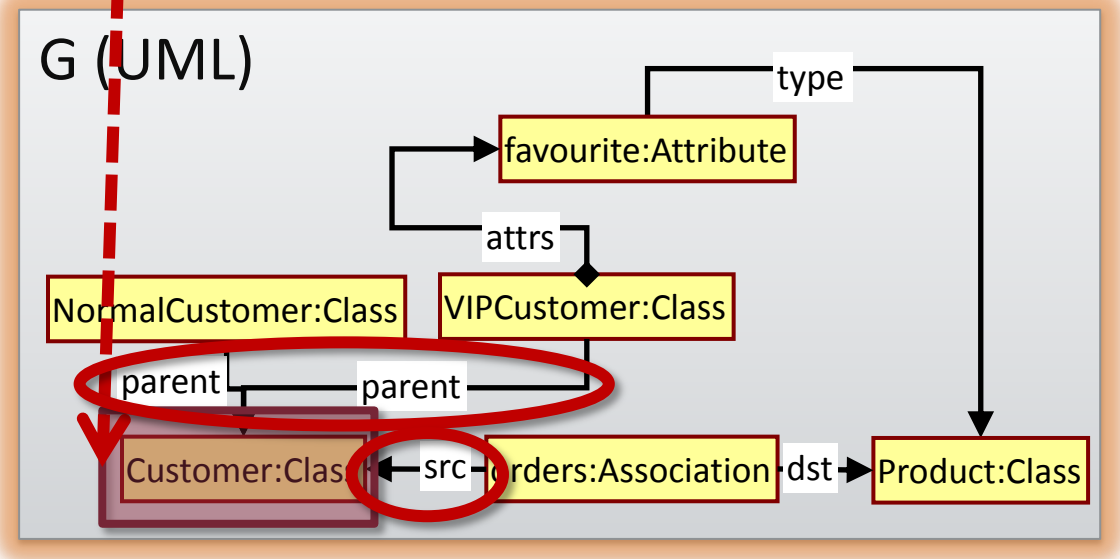




- **Dangling edges**:
  - Delete a node
    - What to do with the dangling edges?
- **Greedy approach**
  - Delete all dangling edges
  - **Pro**:
    - Intuitive for engineers
    - Easy to implement
  - **Con**:
    - Verification is hard (side effect of rules)

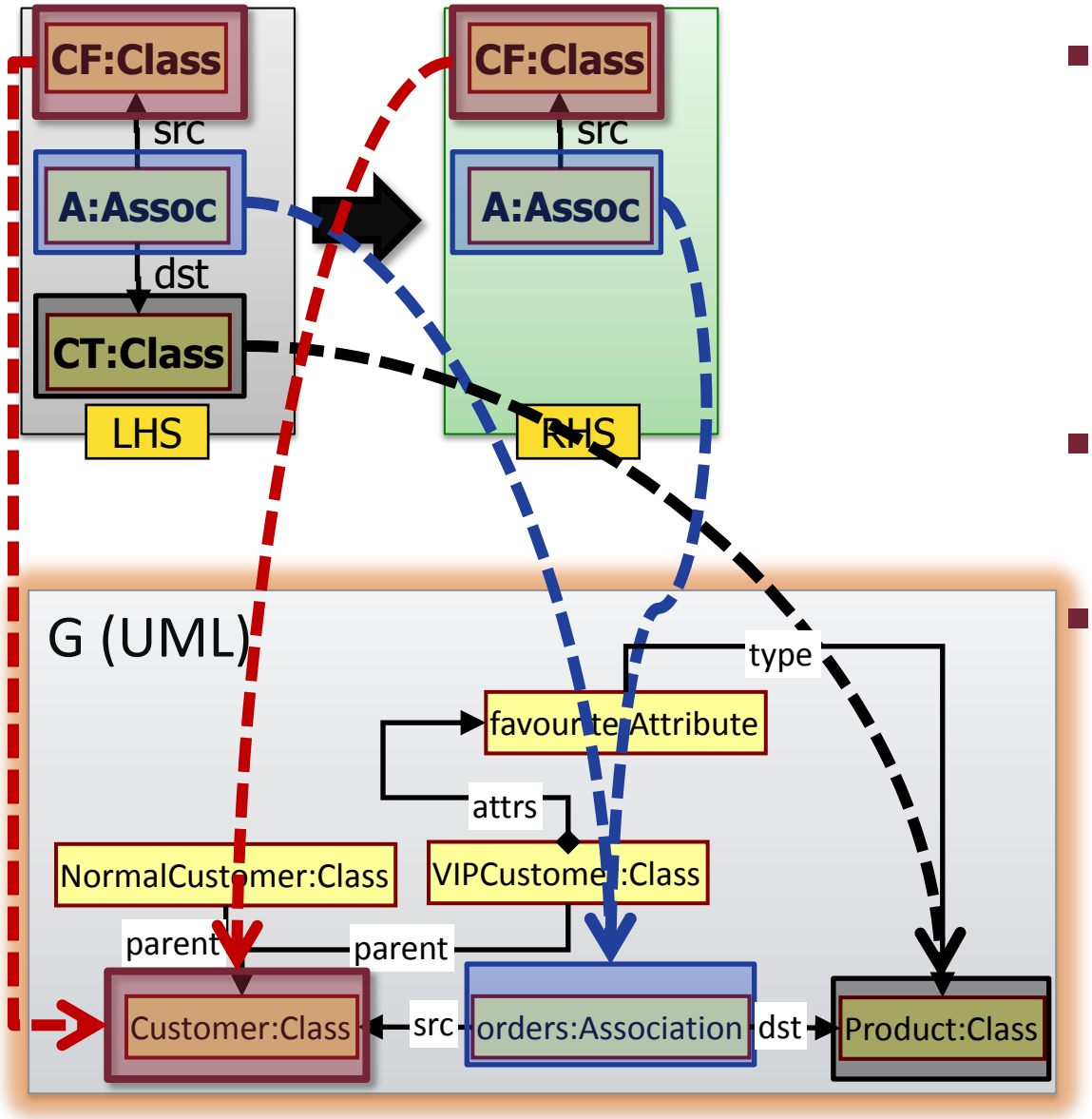
Semantics : Handling of Dangling edges





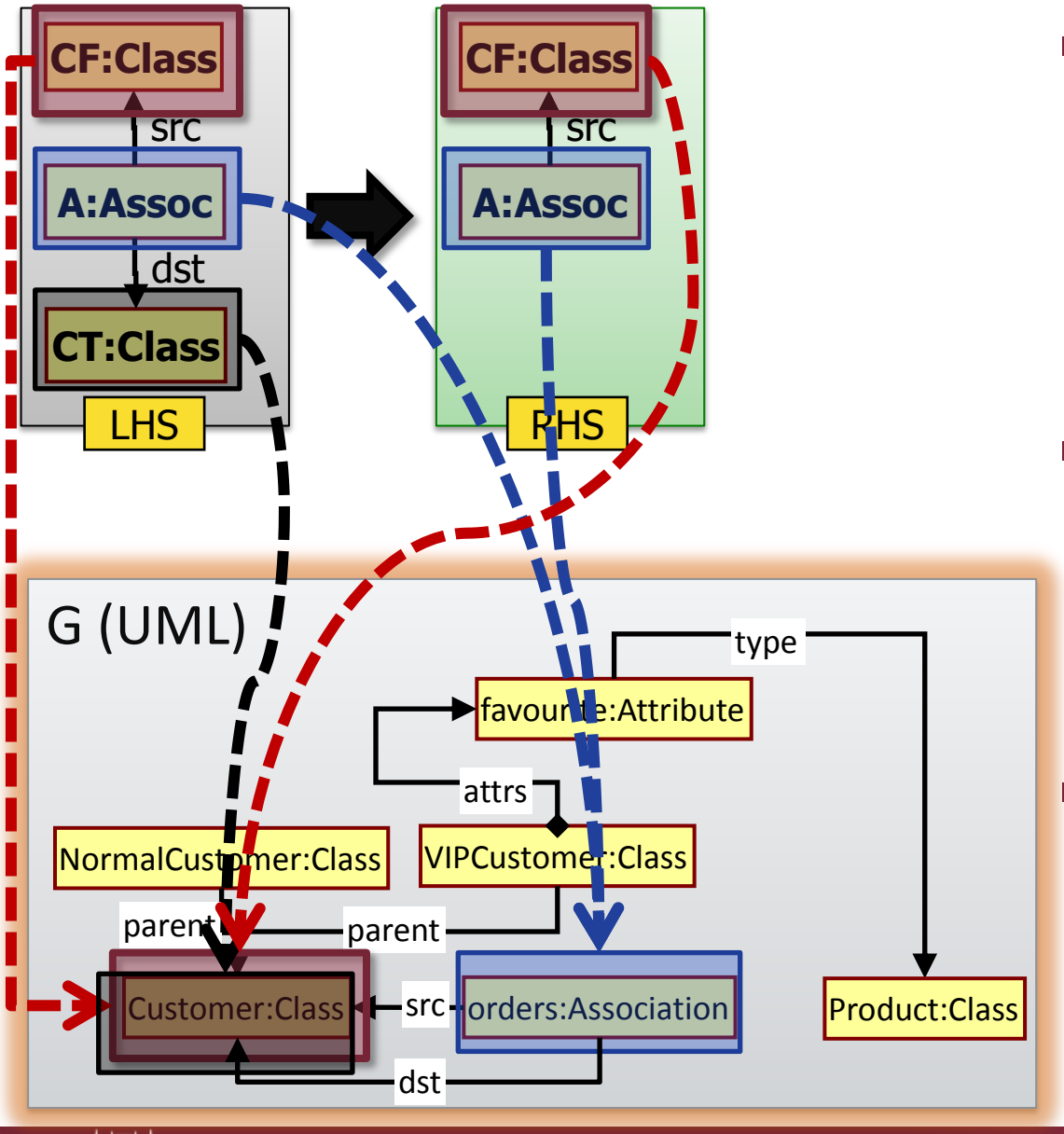
- **Dangling edges**:
  - Delete a node
    - What to do with the dangling edges?
- **Conservative approach**
  - The rule cannot be applied if it would produce a dangling edge
  - **Pro**:
    - Side effect free rules
    - Helps verification
  - **Con**:
    - Harder to implement
    - What is its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

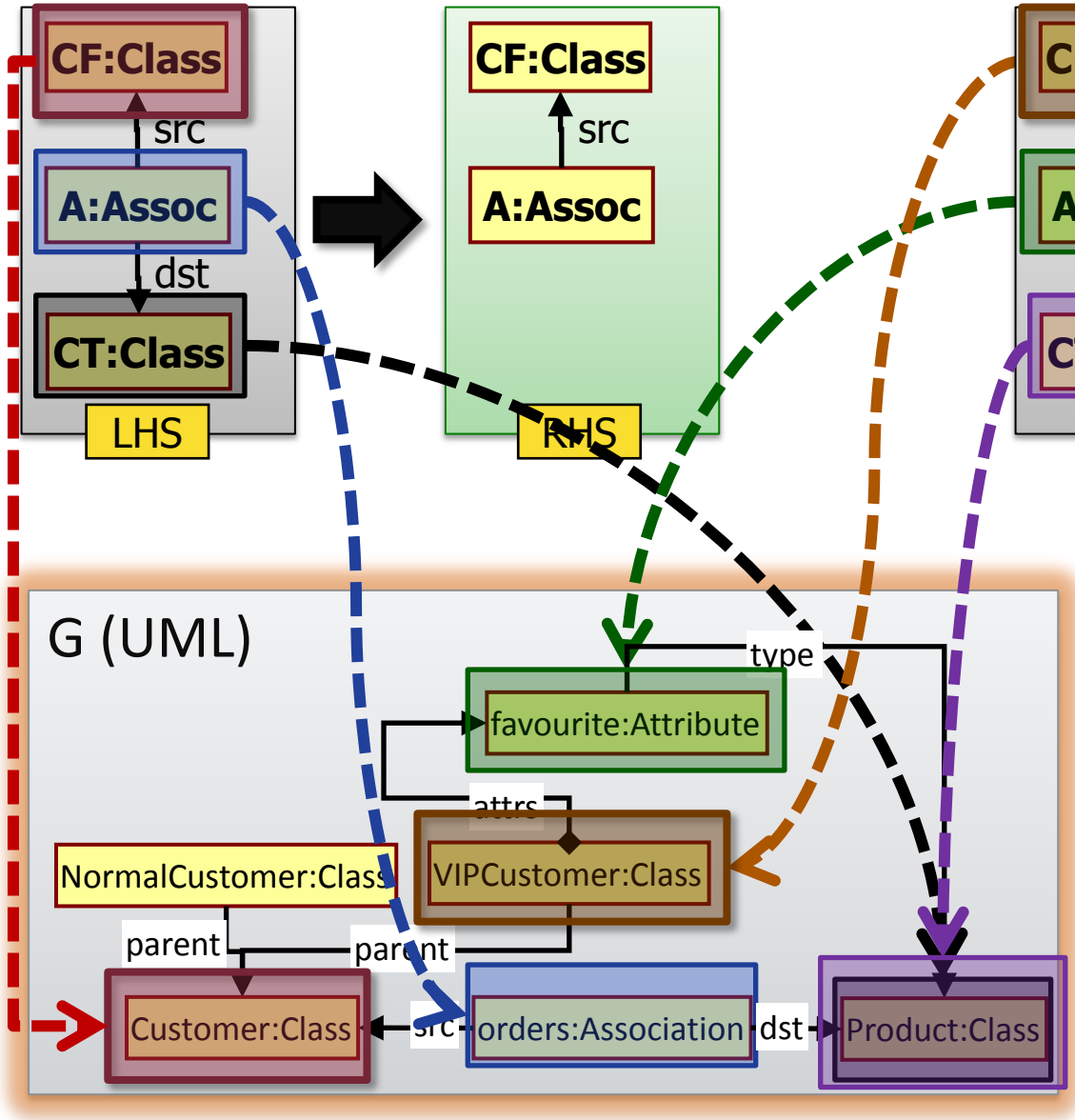
Semantics: Non-injective matching



- **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 injective semantics

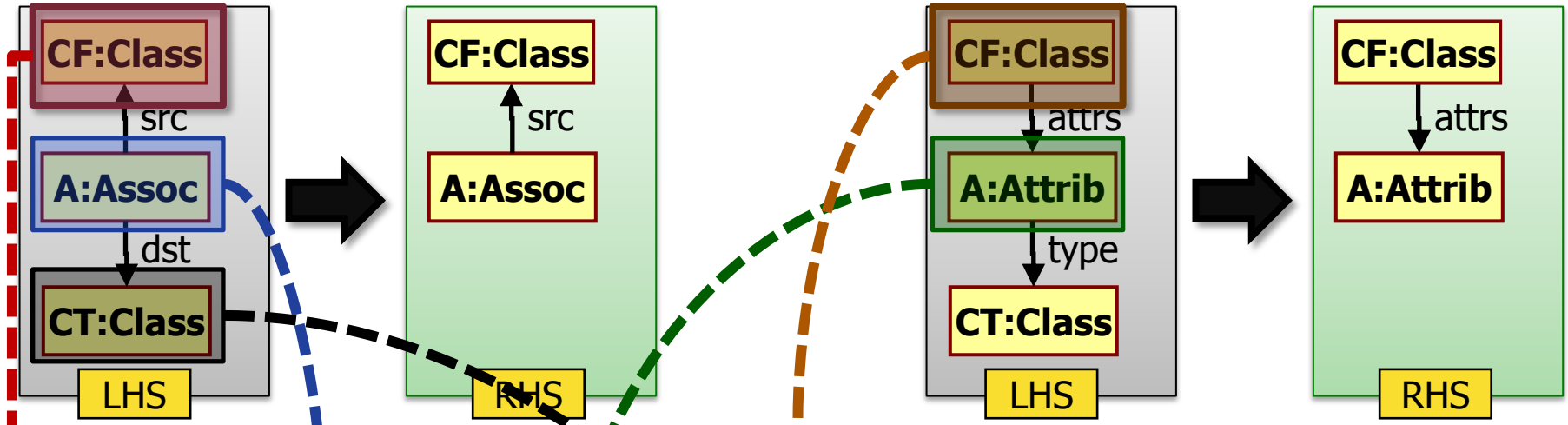
6. Effects of Multiple GT Rules

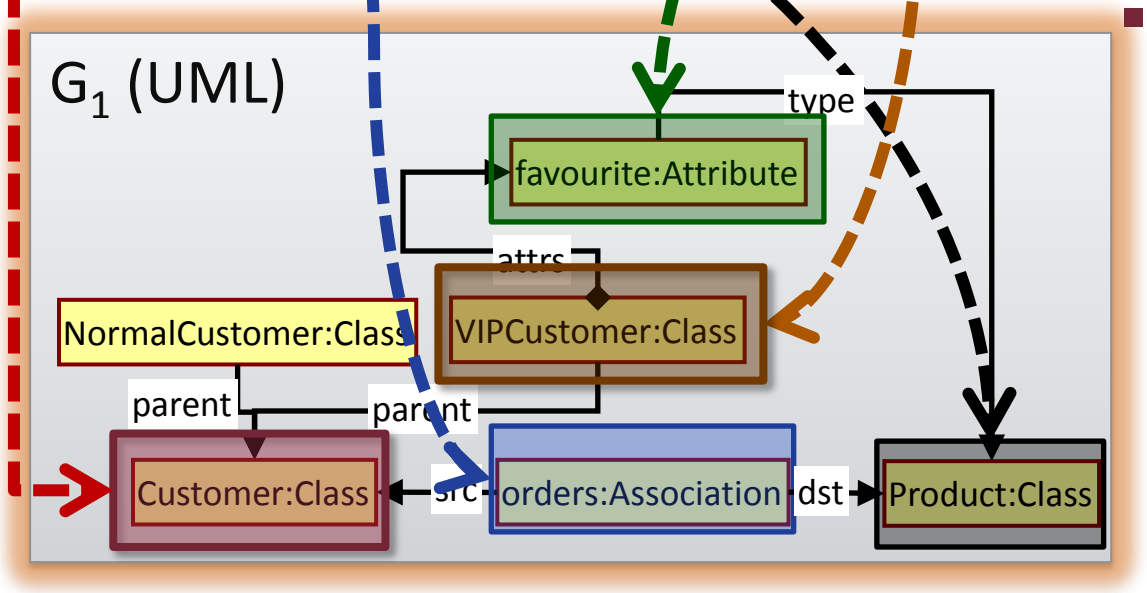
Conflict / Parallel independence



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

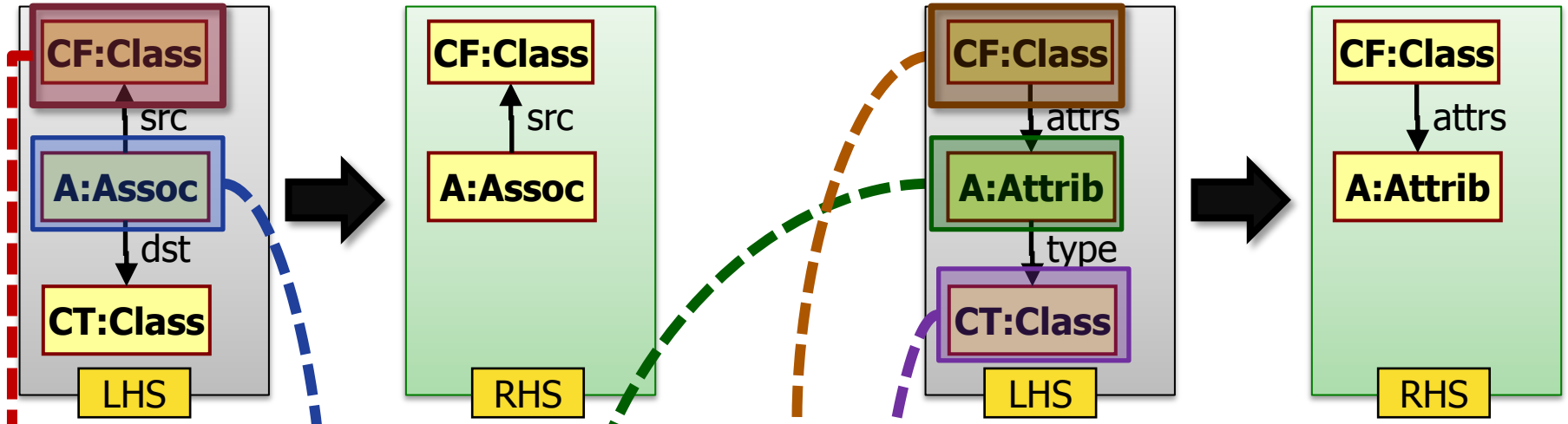
Sequential independence

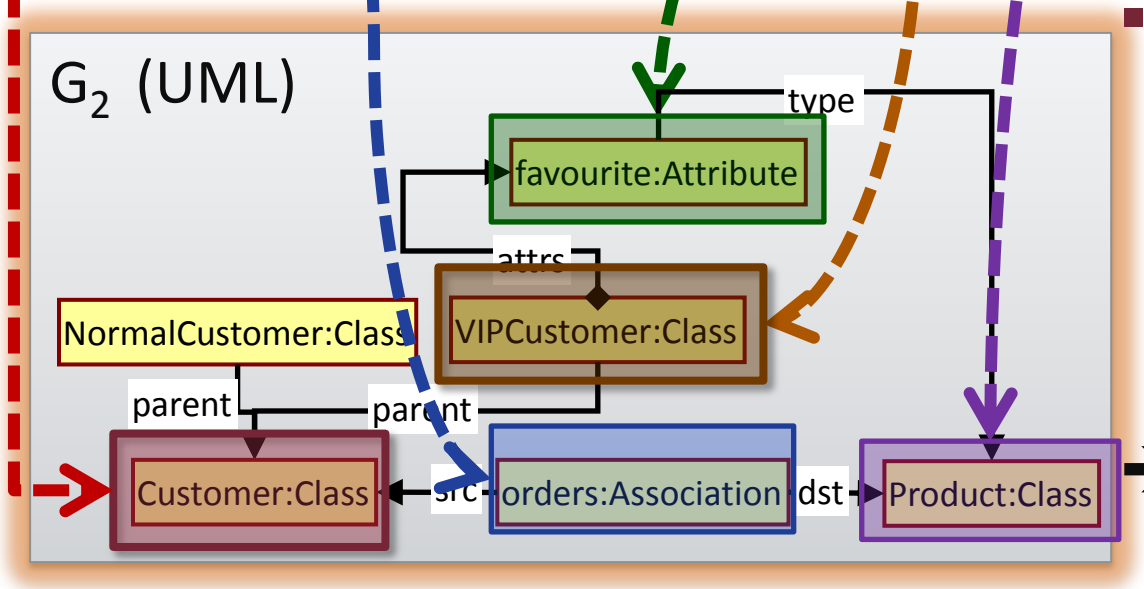




- **Sequential independence** (two following rule applications)
  - Their order can be swapped without any effect on their final result

Sequential independence

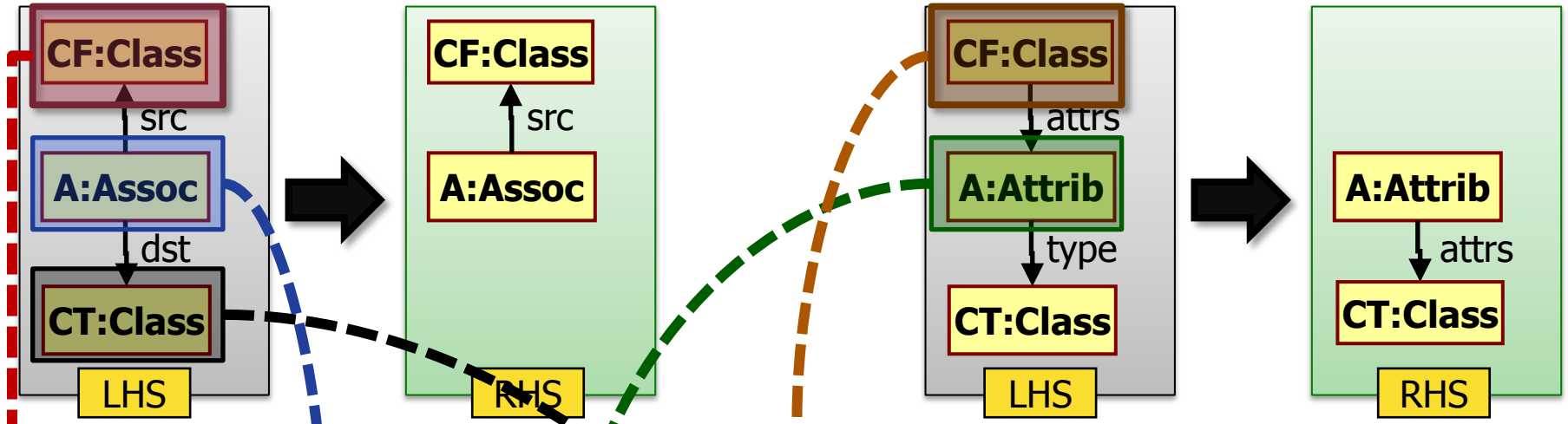


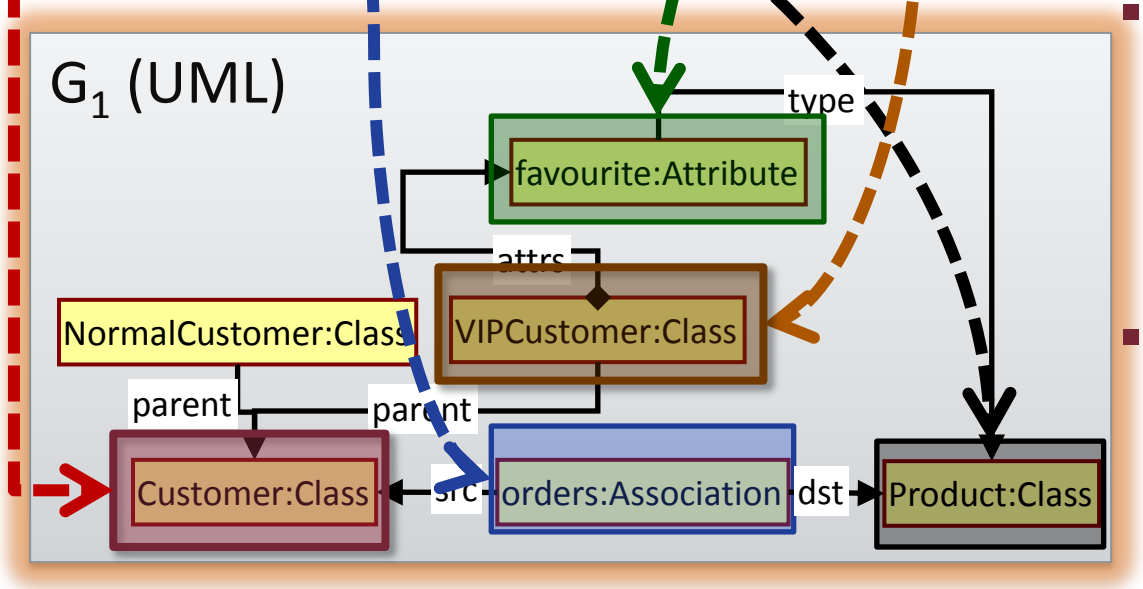


- **Sequential independence** (two following rule applications)
  - Their order can be swapped without any effect on their final result

→ Example

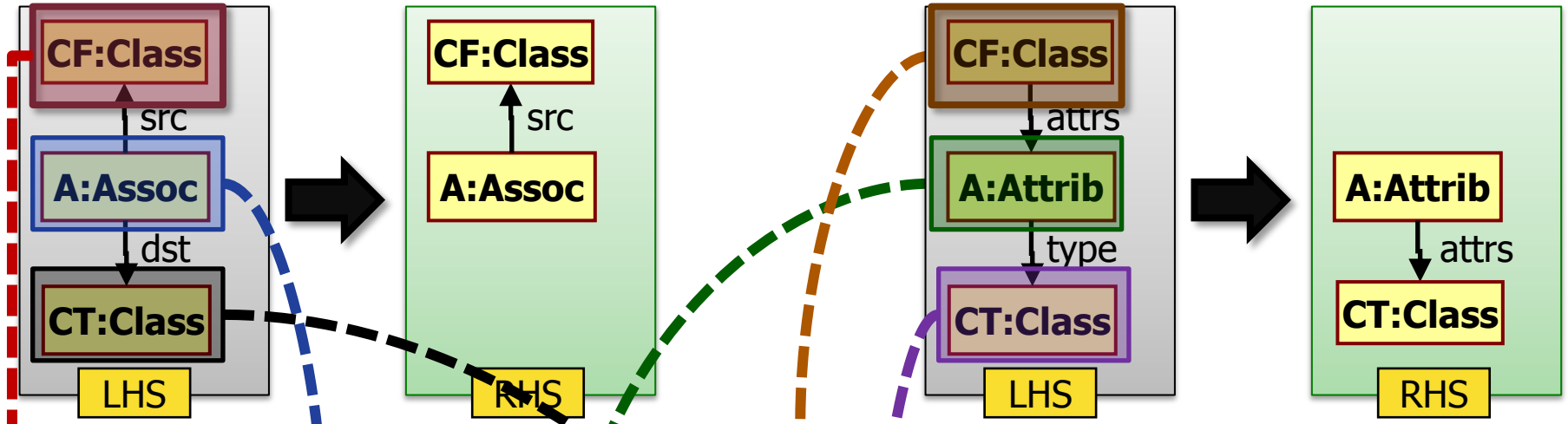
Causal dependence I.

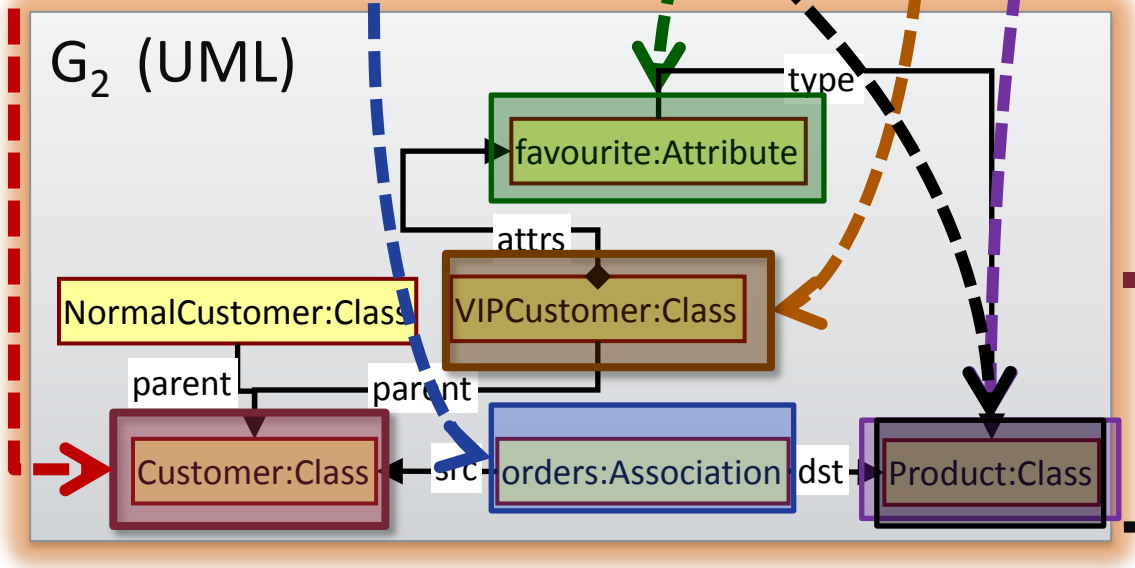




- **Sequential independence**
  (two following rule applications)
  - Their order can be swapped without any effect on their final result
- **Causally dependent**
  (two following rule applications)
  - If they are not serial independent

Causally dependence II.



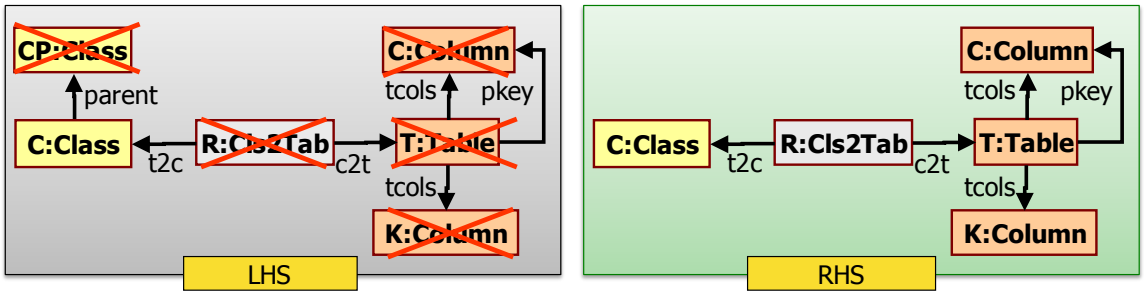


- **Serial independence**
  (two following rule applications)
  - Their order can be swapped without any effect on their final result
- **Causally dependent**
  (two following rule applications)
  - If they are not serial independent

→ Example

Summary

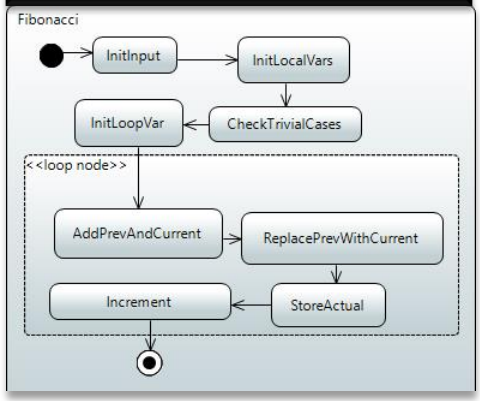
- **Graphtransformation**,
  as a modeltransformation paradigm
  - Rule and pattern based formal specification
  - Querying and manipulating graph based models
  - Intuitive graph based specification
- **Structure**
  - LHS graph pattern: precondition
  - RHS graph pattern: postcondition
  - NAC: negative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



Chaining and Traceability of Model Transformations

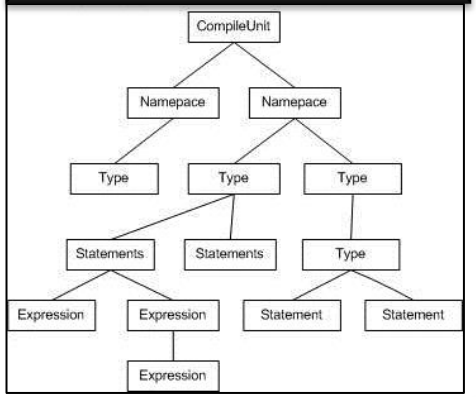
Code Generation by Model Transformations

Source Model



M2M

Code DOM/AST



M2T

Target Code

```
#include<iostream>
using namespace std;

int main() {
int number, reverse = 0;
 cout<<"Input a Number to Reverse:  ";
 cin>> number;

   for( ; number!= 0 ; )
   {
      reverse = reverse * 10;
      reverse = reverse + number%10;
      number = number/10;
   }
   cout<<"New Reversed Number is:  "<<reverse;

   return 0;
}
```

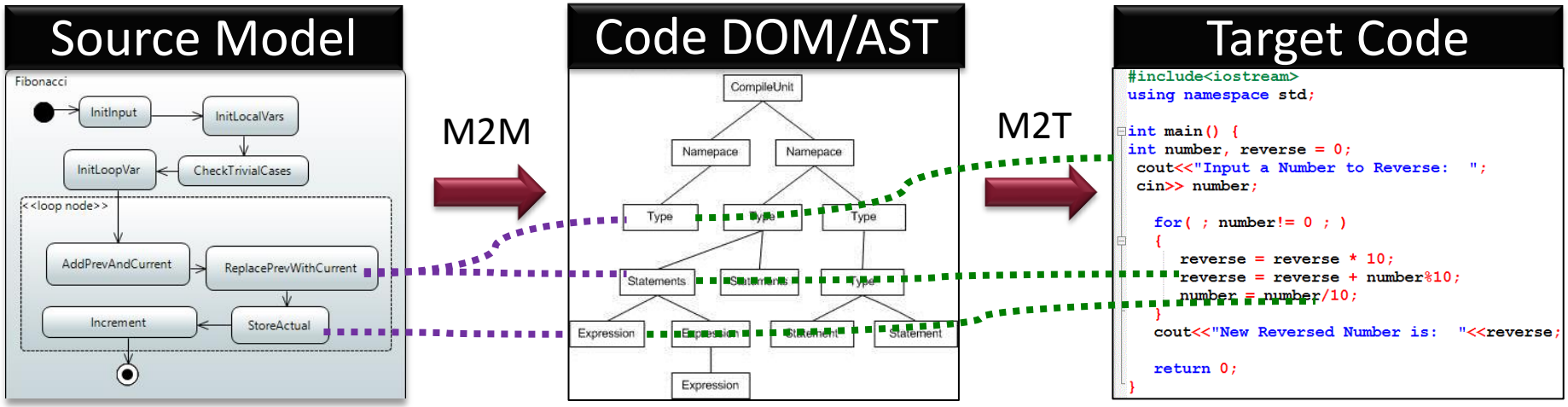
Model-to-Model (M2M) Transformation

- SRC: In-memory model (objects)
- TRG: In-memory model (objects)

Model-to-Text (M2T) Transformation

- SRC: In-memory model (objects)
- TRG: Textual code (string)

Traceability in Model Transformations



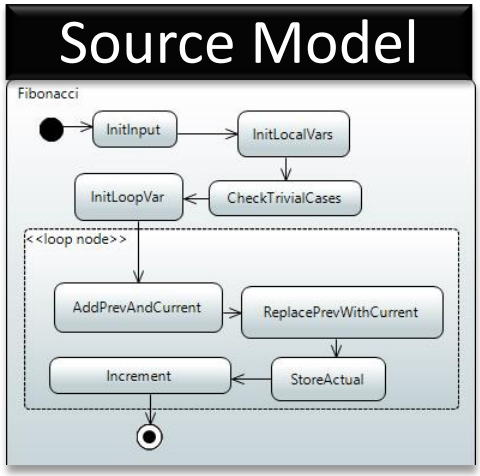
Traceability links:

- Additional links (edges)
- Connect SRC and TRG models

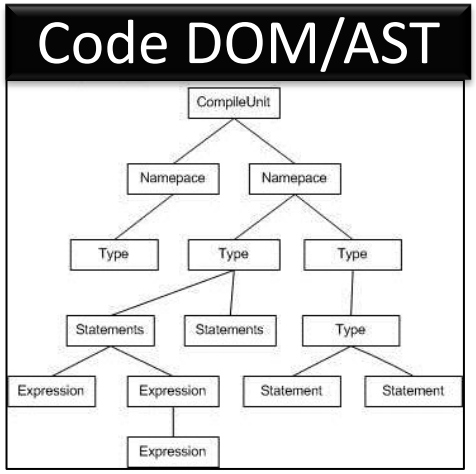
Objective:

- Support end-to-end traceability
- REQ ⇔ Model ⇔ Code

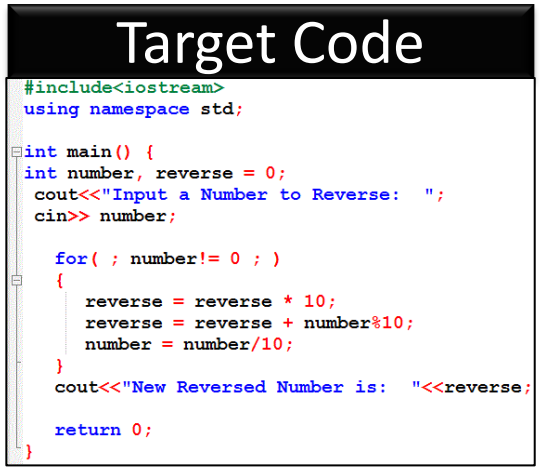
Chaining of Model Transformations



M2M

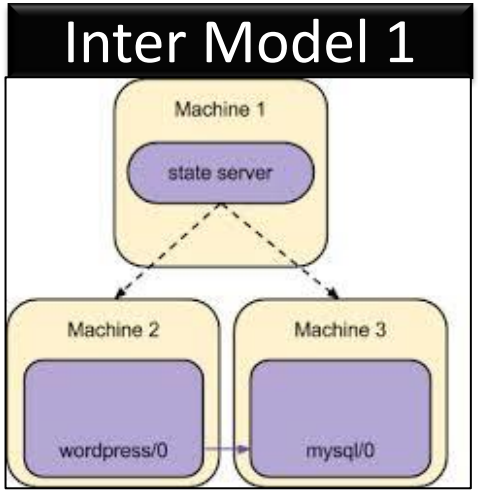


M2T

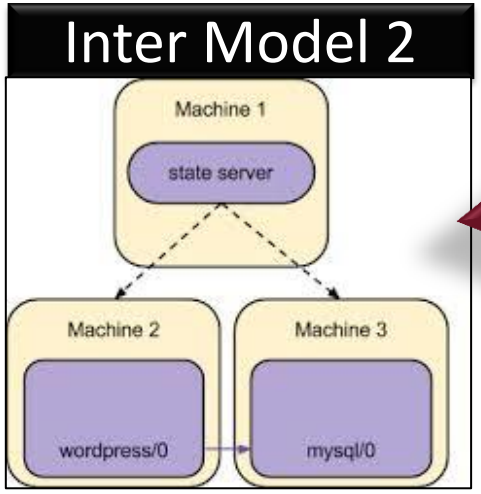


M2M

M2M



M2M

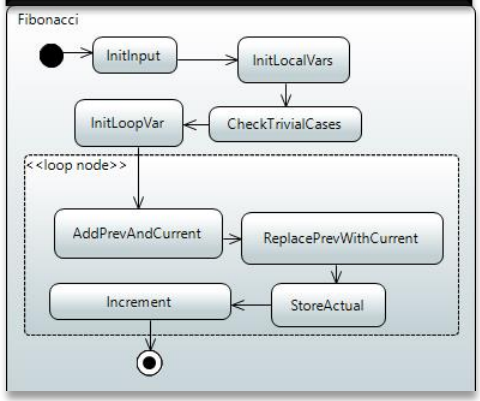


Goal:

- Reduce abstraction gap by „divide and conquer”
- Intermediate models
- Chain of model transformations

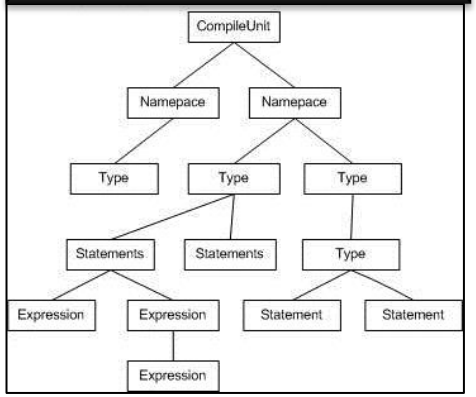
Model Transformation Flows / Chains

Source Model



M2M

Code DOM/AST



M2T

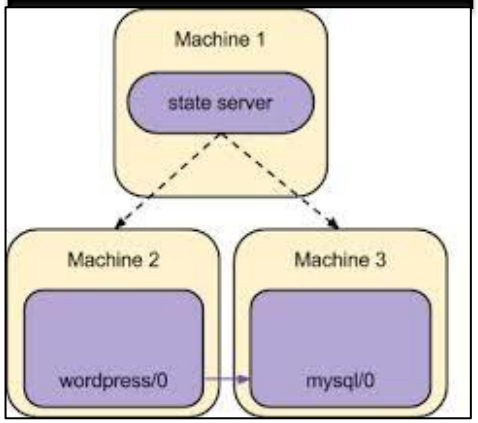
Target Code

```
#include<iostream>
using namespace std;

int main() {
int number, reverse = 0;
 cout<<"Input a Number to Reverse:  ";
 cin>> number;

   for( ; number!= 0 ; )
   {
      reverse = reverse * 10;
      reverse = reverse + number%10;
      number = number/10;
   }
   cout<<"New Reversed Number is:  "<<reverse;

   return 0;
}
```

M2M

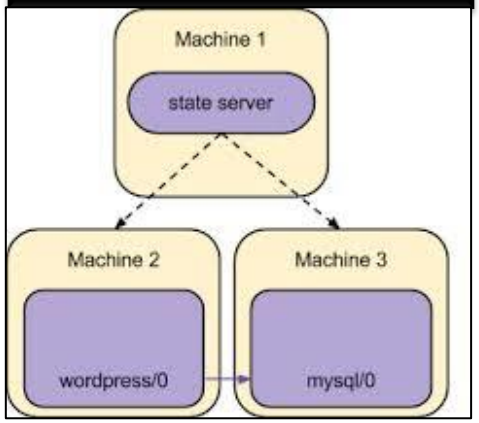
Joint optimization steps

Inter Model 1



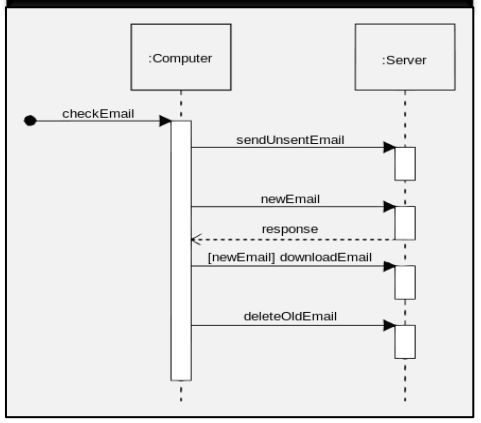
M2M

Inter Model 1



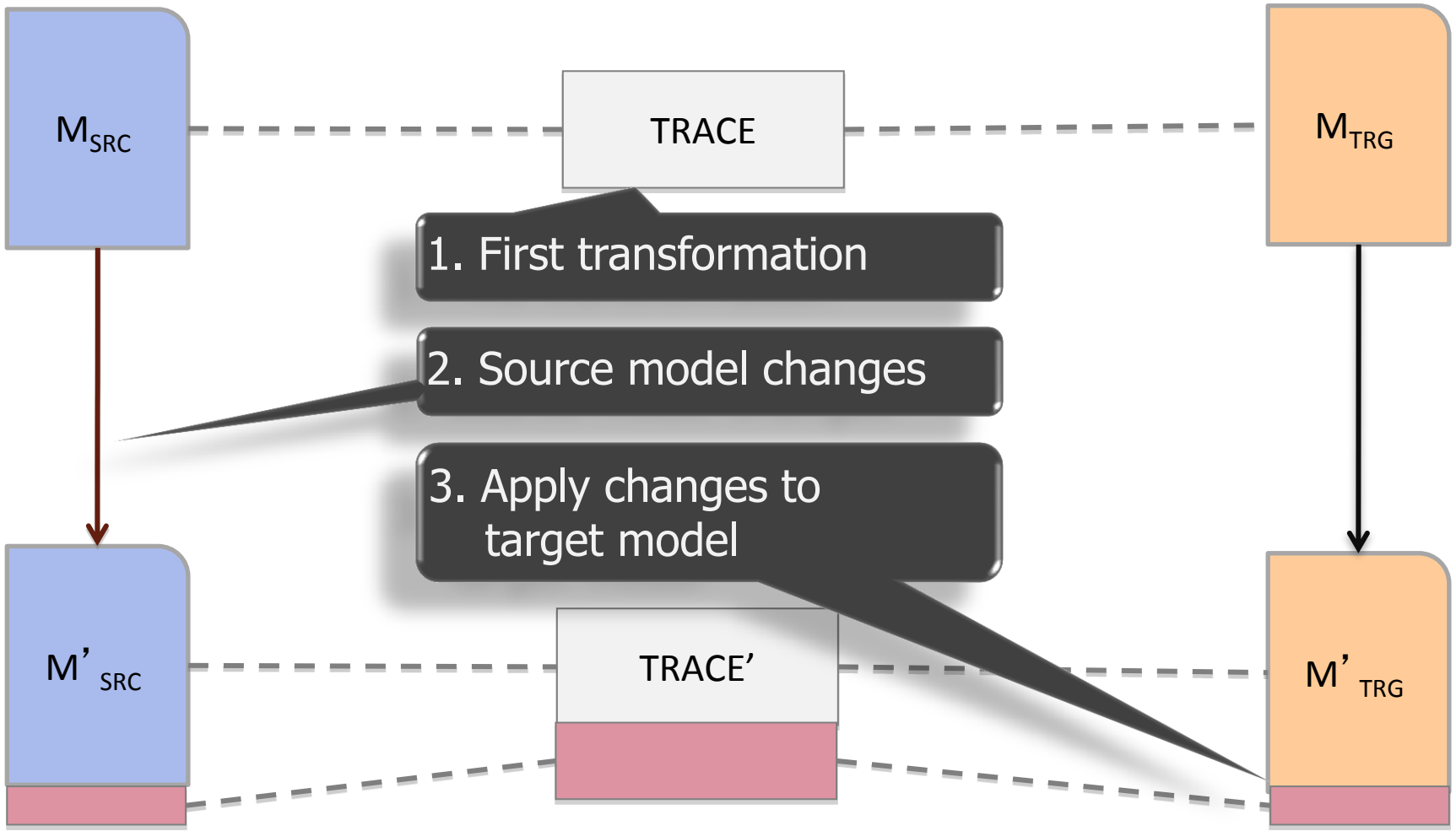
M2M

Source Model 2



Incrementality in model transformations

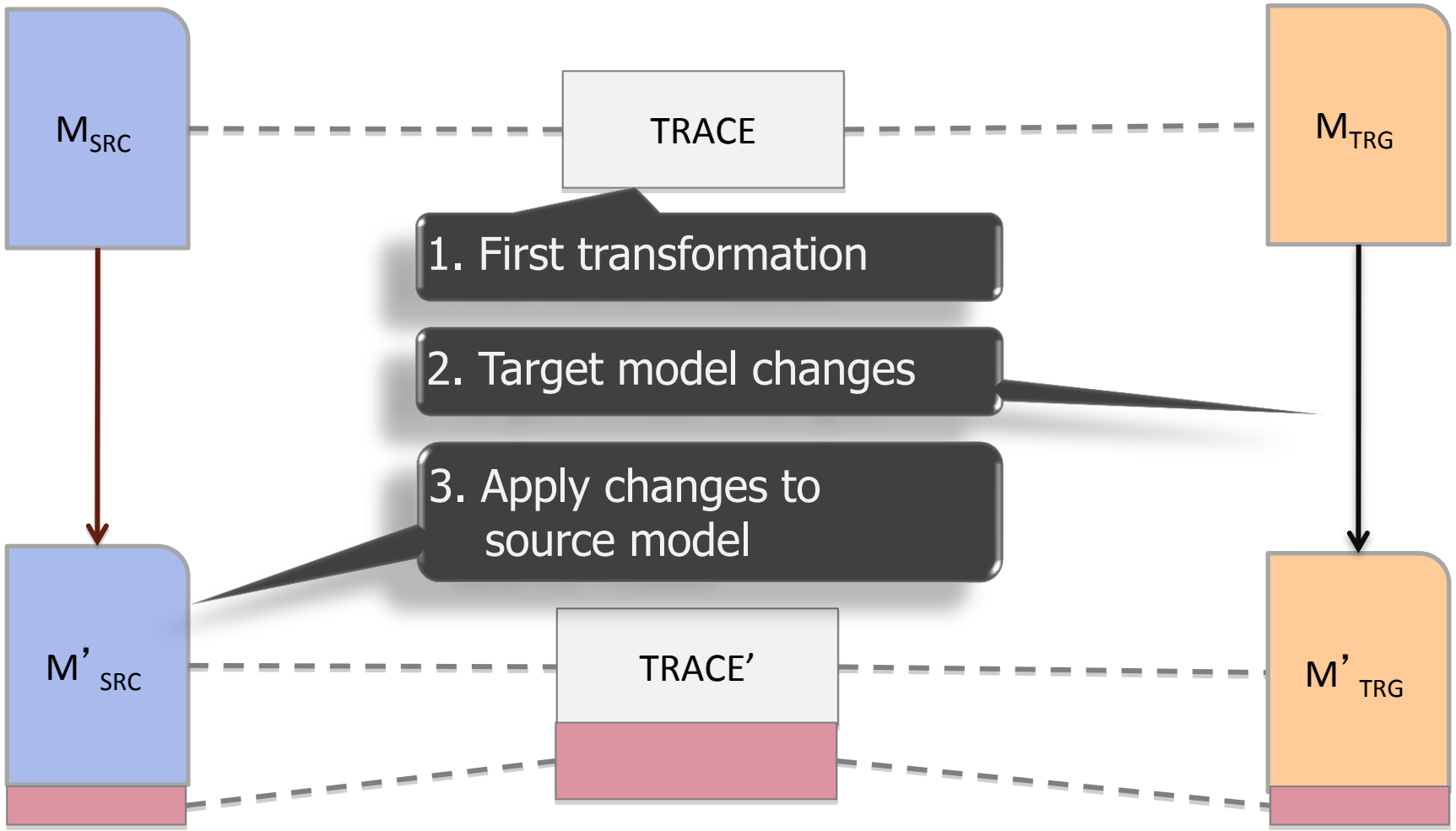
Incremental Forward Transformation



Practical application scenarios:
- Incremental model synchronization
- Tool integration

Solutions:
- Bidirectional transformations
- Change-driven transformations

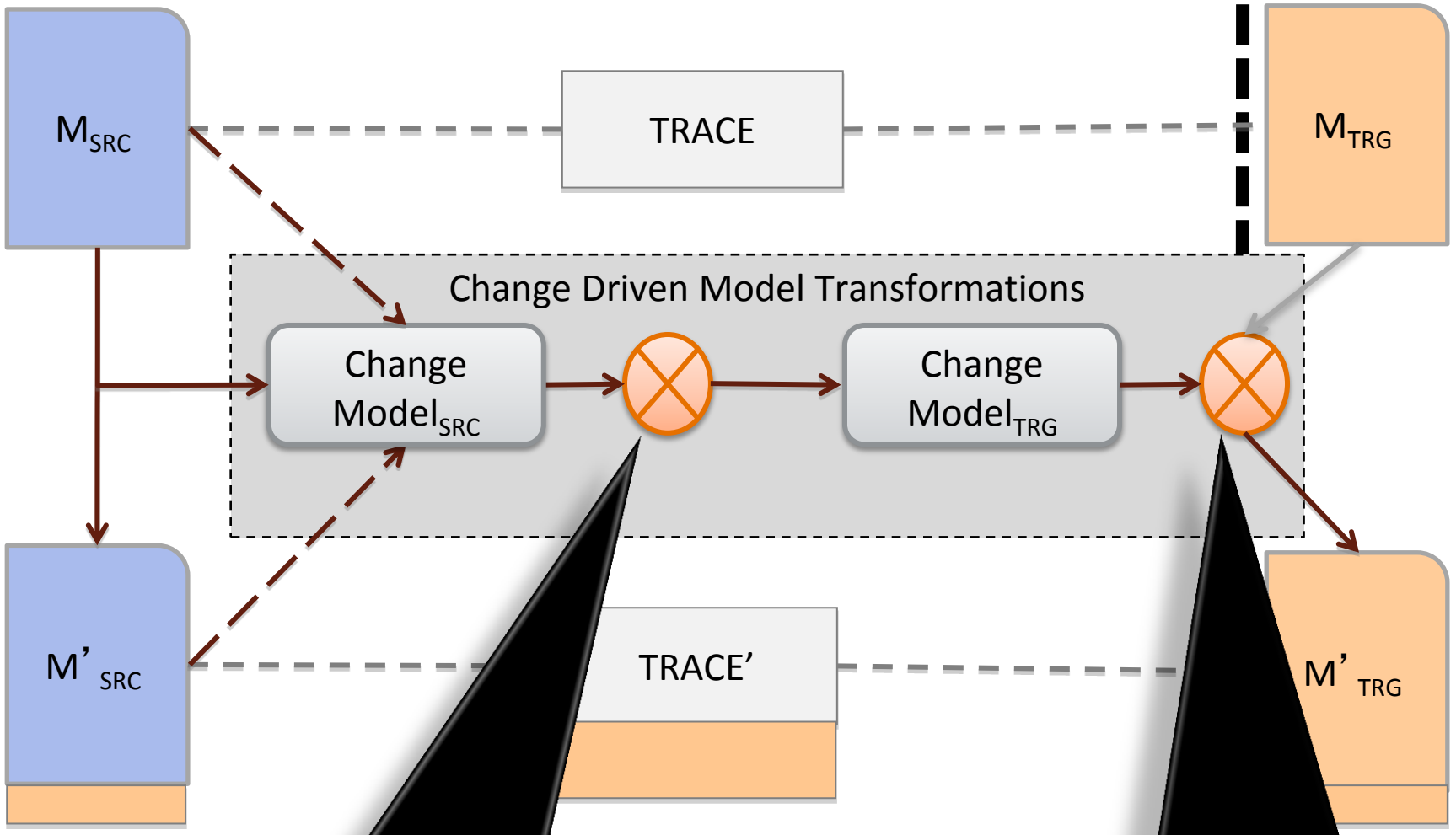
Incremental Backward Transformation



Challenge:
SRC→TRG specified
TRG→SRC inferred

Recent Approaches:
A. Schürr, P. Stevens, N. Foster, T. Hettel, Cicchetti&Pierantonio, Czarnecki&Diskin

Change Driven Model Transformations

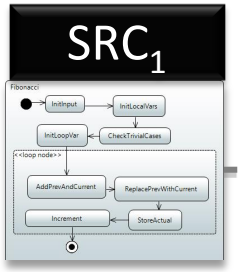


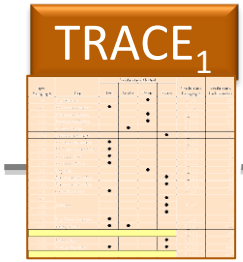
Change Driven Transformation
- Input: consumes change model
- Output: produces change model

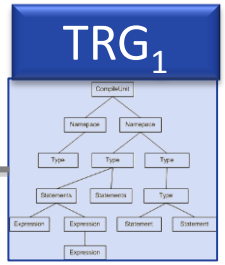
Apply Target Change Model
- via an API with little trace info
- target model is not materialized!

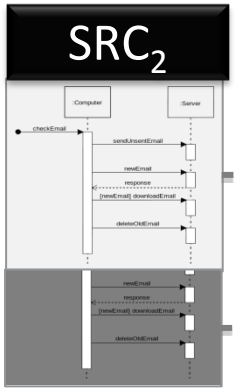
Levels of Incrementality in Model Transformations

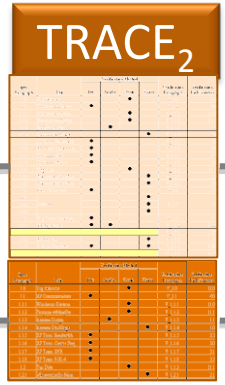
No Incrementality: Batch Transformations

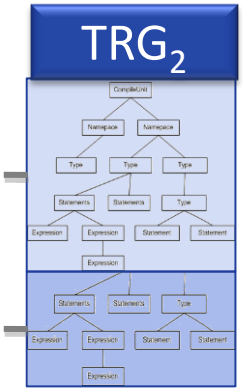






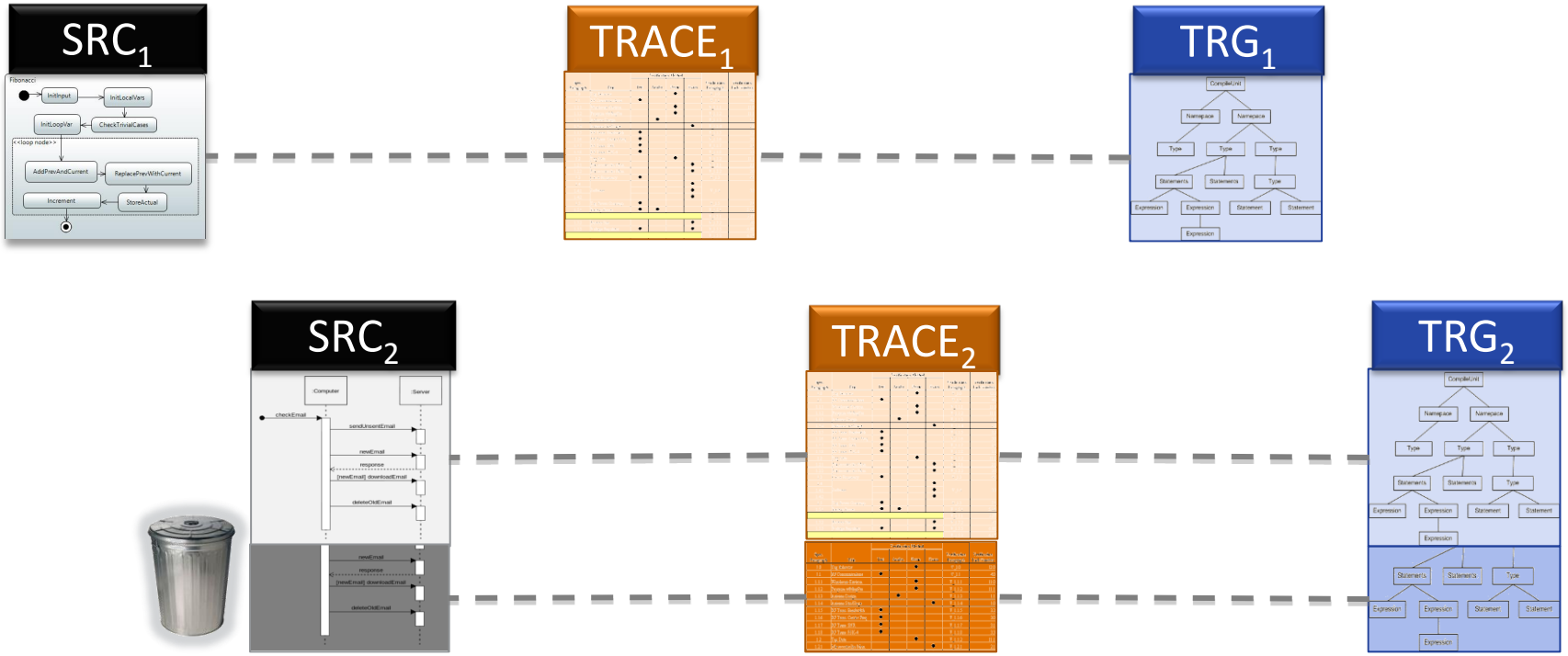






1. First transformation
2. Source model changes
3. Re-execute from scratch for all source models

Dirty Incrementality



Pros:

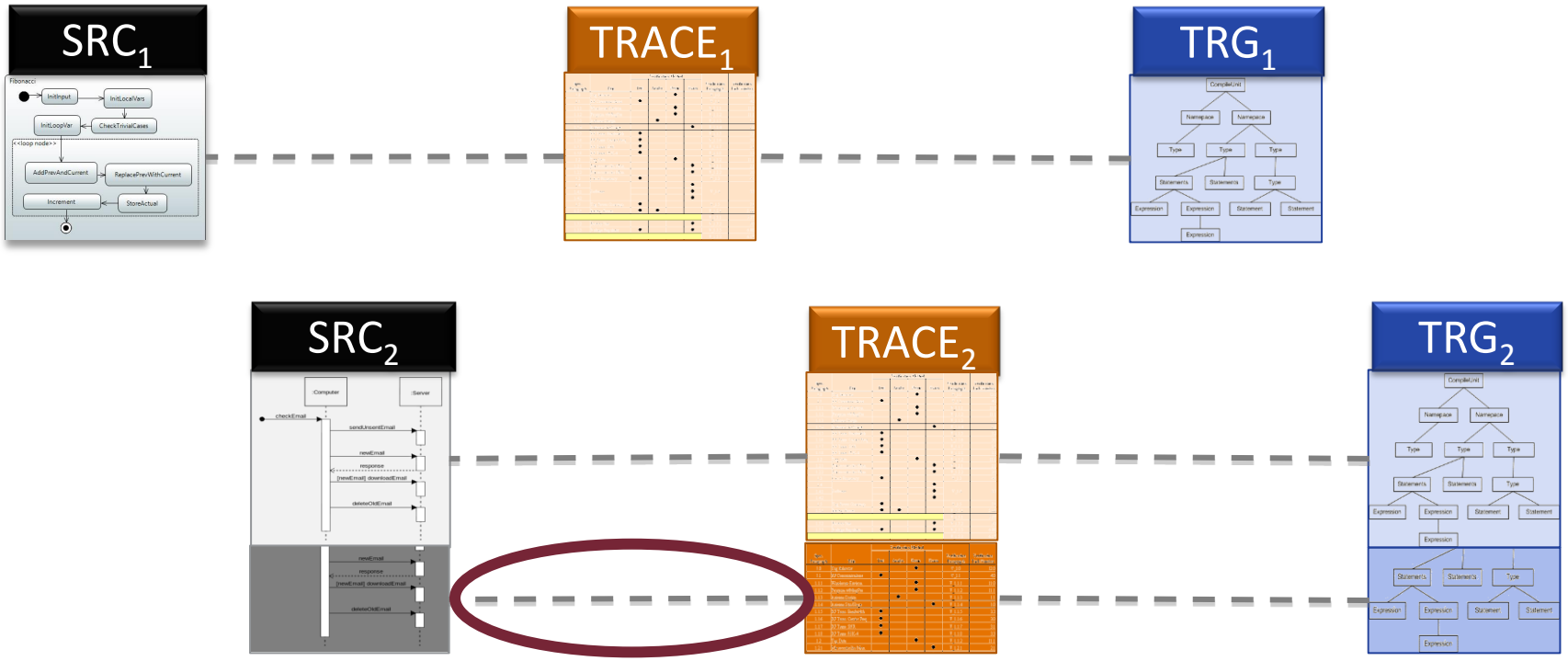
- Large-step incrementality
- Avoids continuous exec.

Cons:

- Complex MT can be slow
- Cleanup (after an error)?
- Chaining?

1. First transformation
2. Source model changes
3. Re-execute from scratch only for changed models

Incrementality by Traceability

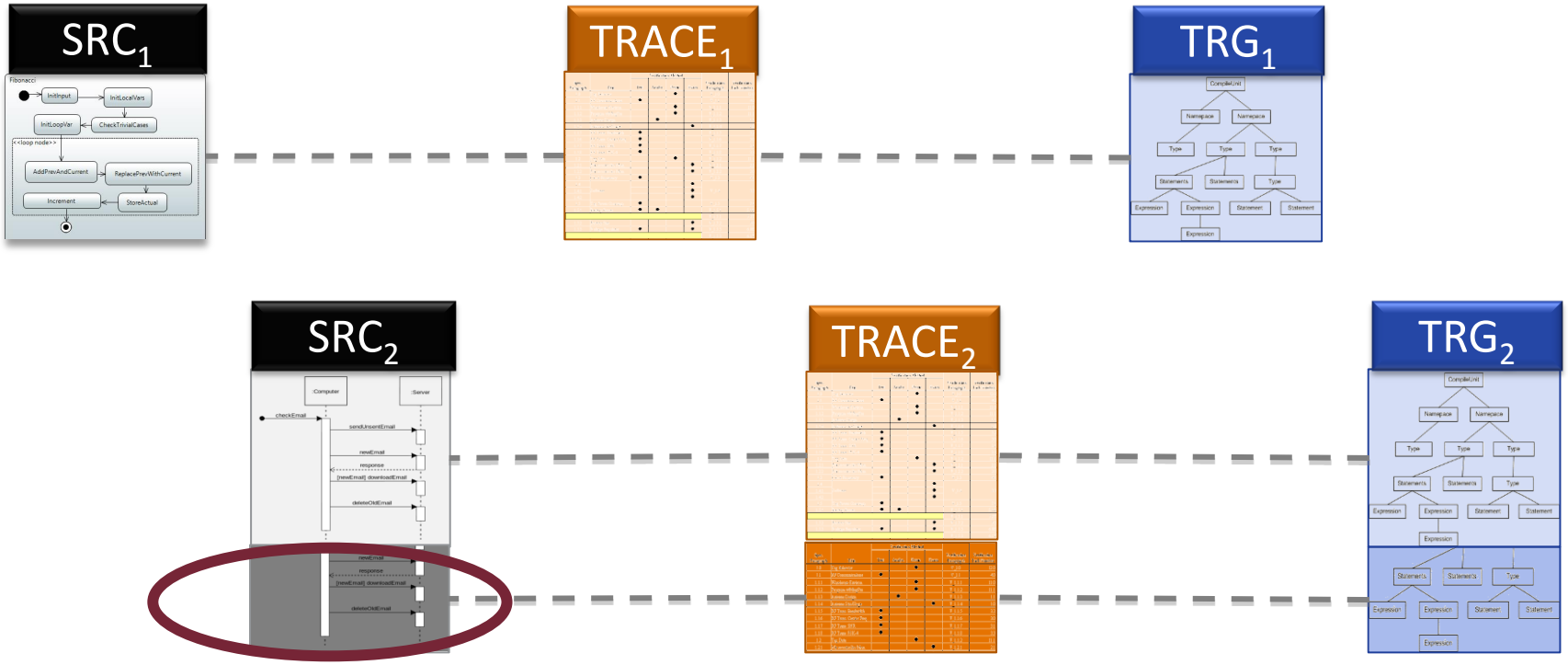


Pros:
- Small-step incrementality
- Better performance

Cons:
- Highly depends on traceability links
- Smart matcher needed

1. First transformation
2. Source model changes
3. Detect missing trace links
4. Re-execute MT only for untraceable elements

Event Driven Transformations



Pros:

- Refined context: driven by changes of query result set
- Chaining
- Avoids continuous comp.

Cons:

- Language-level restrictions

1. First transformation
2. Source model changes
3. Detect new activations
4. Fire rule activations (in relevant context)

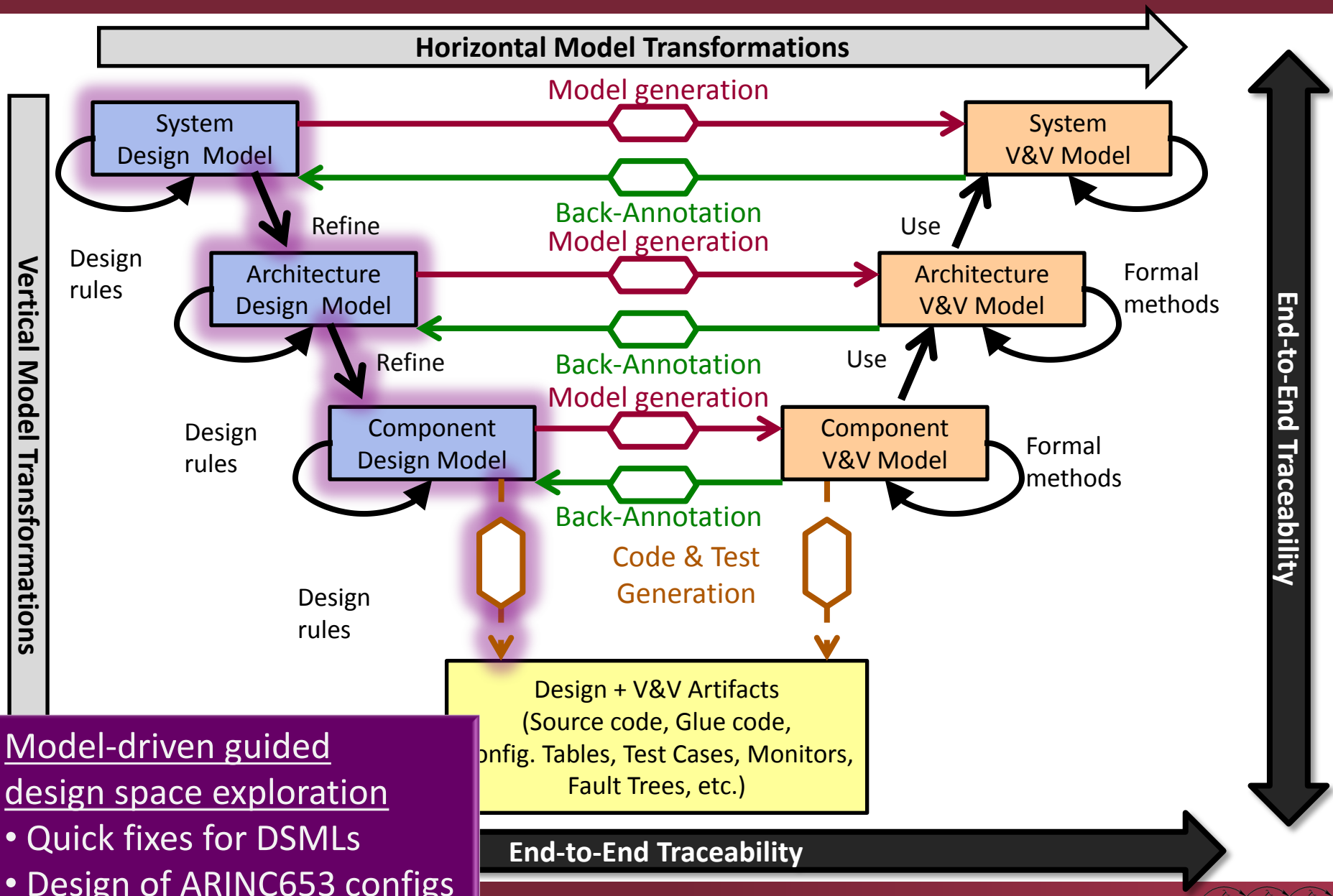
IncQuery

Design Space Exploration

Á. Hegedüs, Á. Horváth, D. Varró:
A model-driven framework for guided design space exploration.
Automated Software Engineering (August 2014)

DOI: 10.1007/s10515-014-0163-1

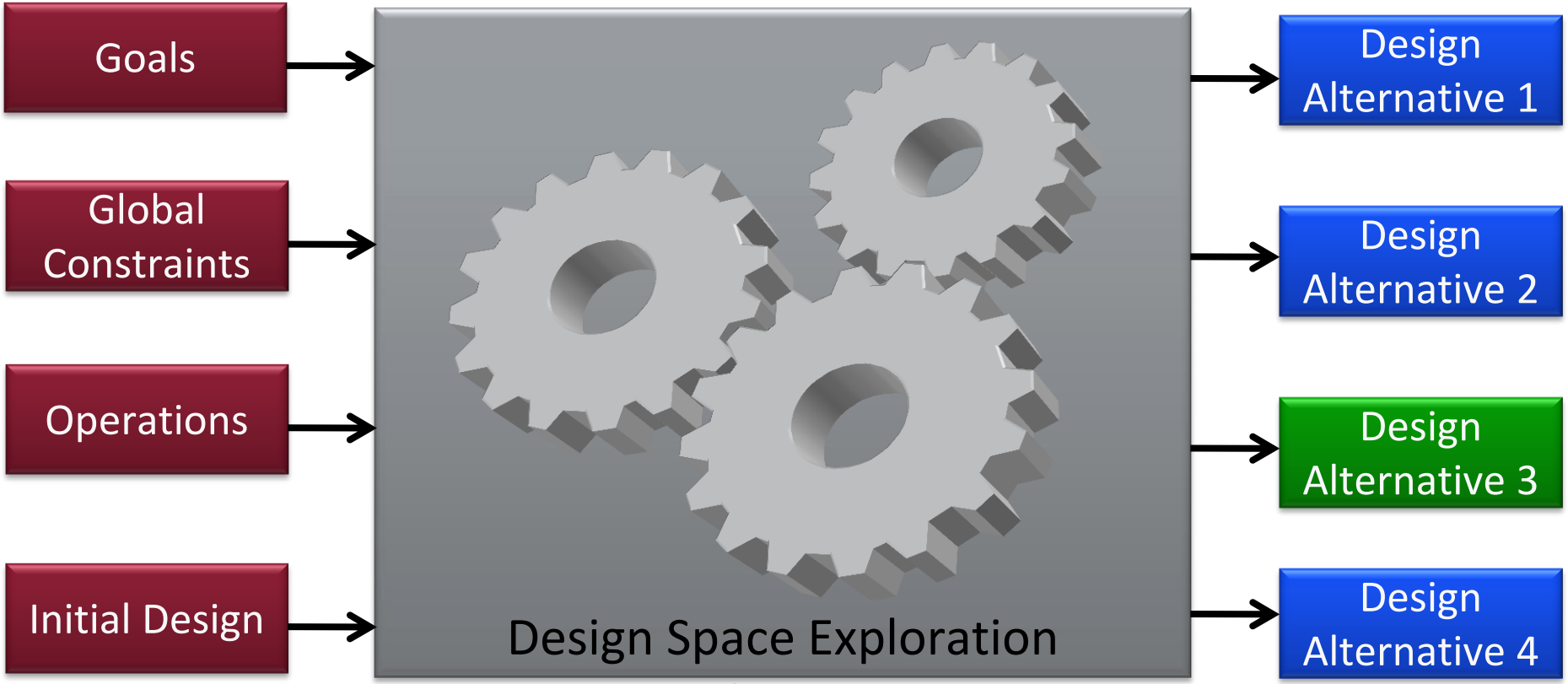
Model-Driven Guided Design Space Exploration



Model-driven guided design space exploration

- Quick fixes for DSMLs
- Design of ARINC653 configs

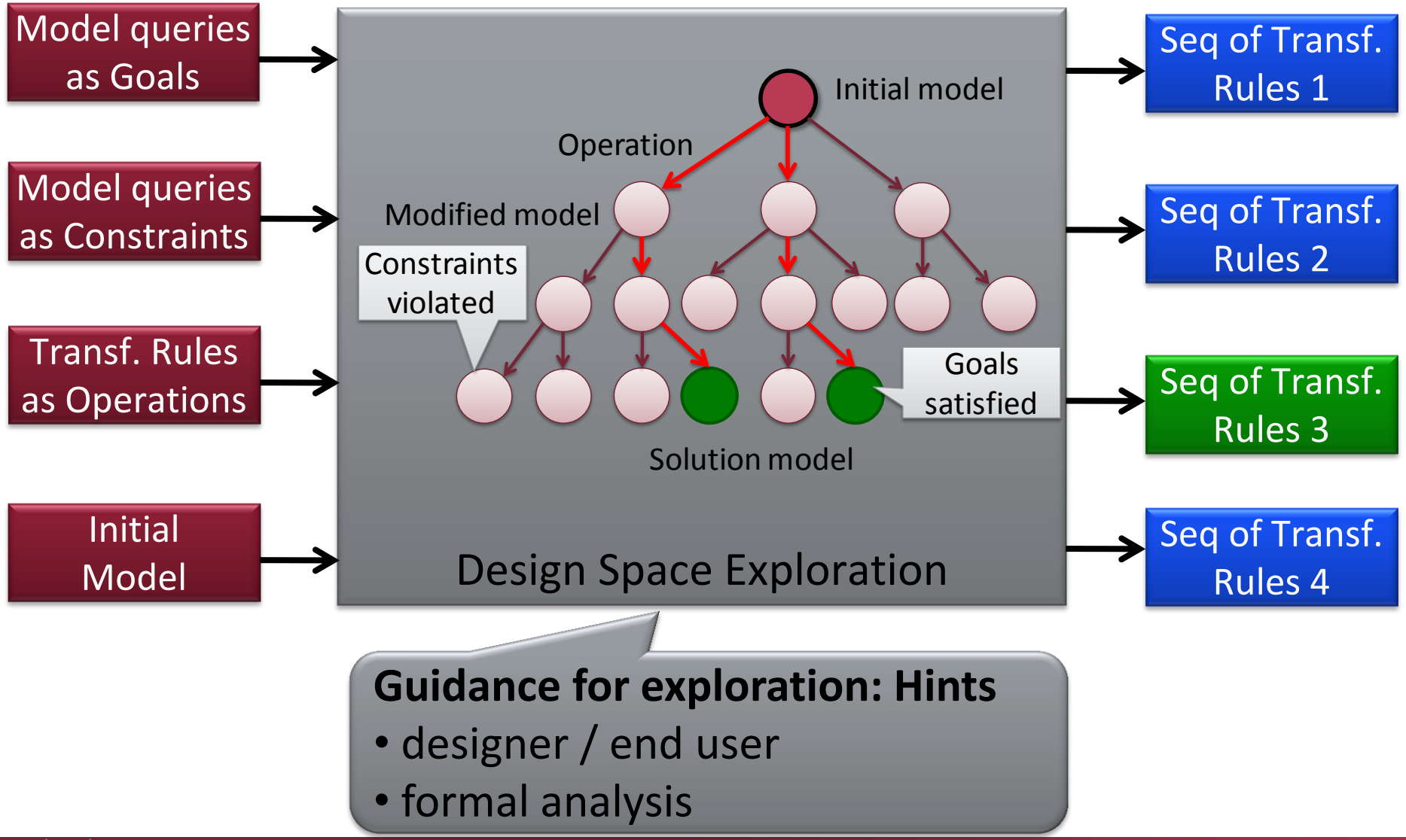
Design Space Exploration



Special state space exploration

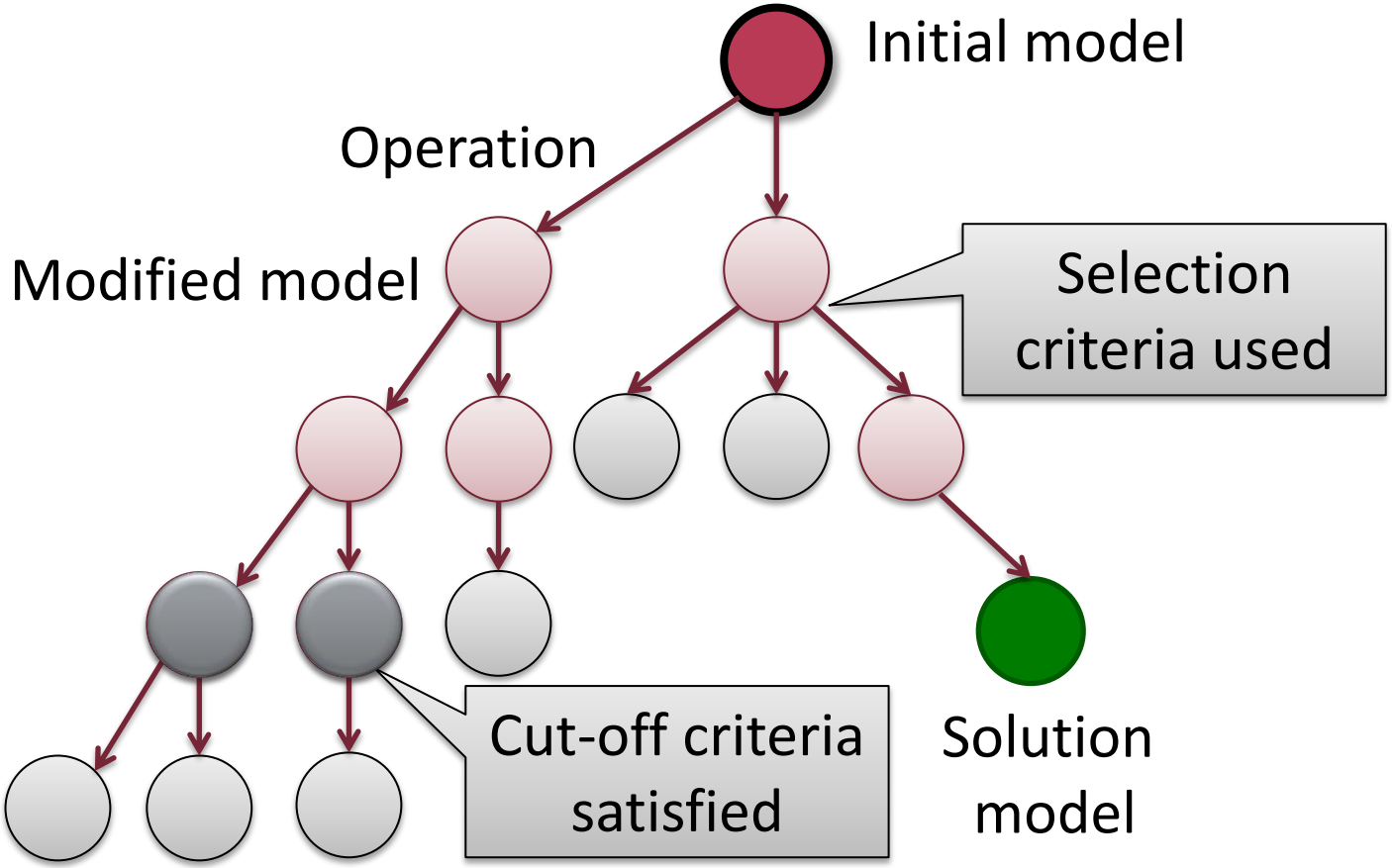
- potentially infinite state space
- „dense” solution space

Model Driven Guided Design Space Exploration

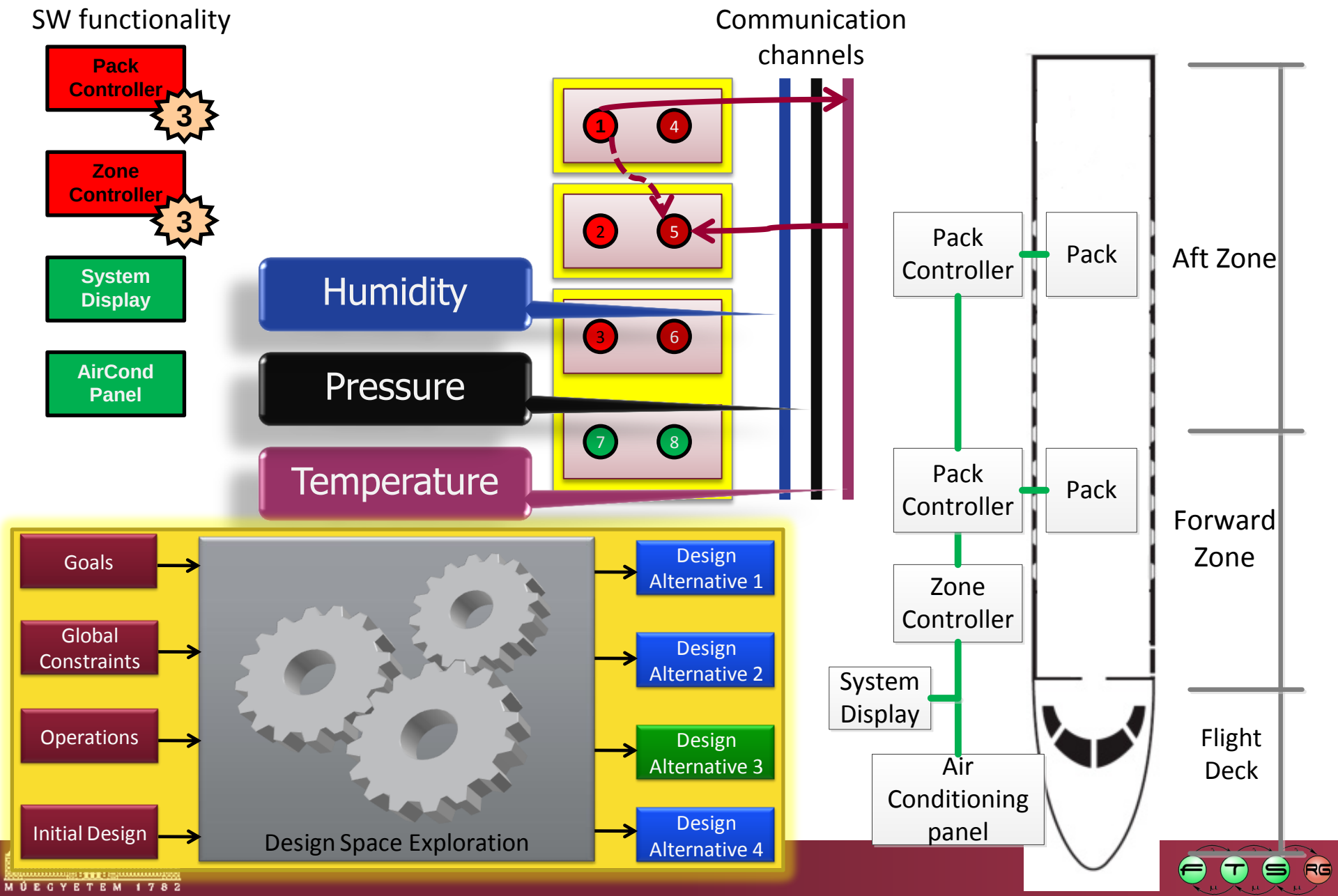


Guided Design Space Exploration

- High-level overview

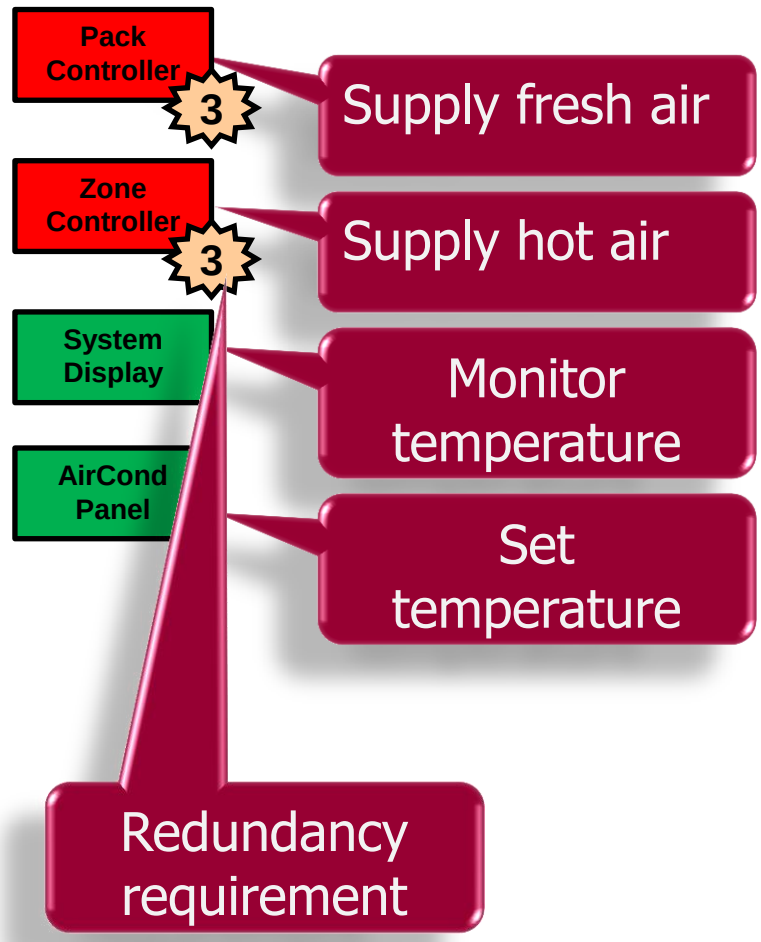


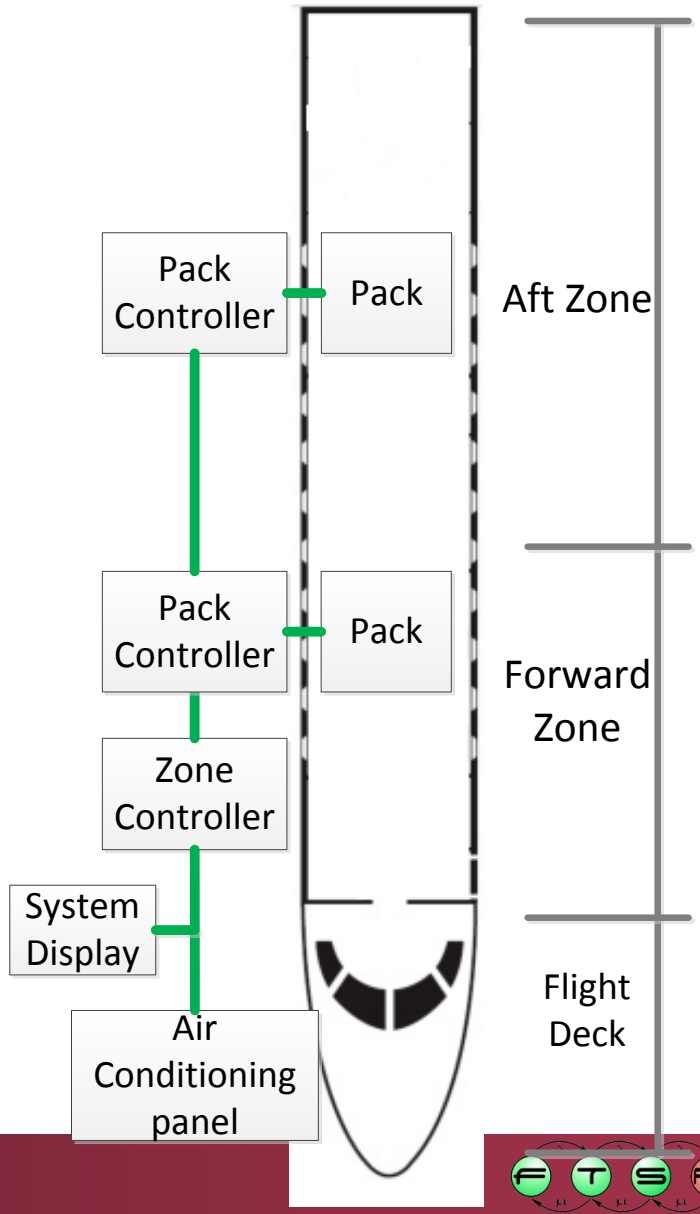
Design Space Exploration for IMA Config. Design



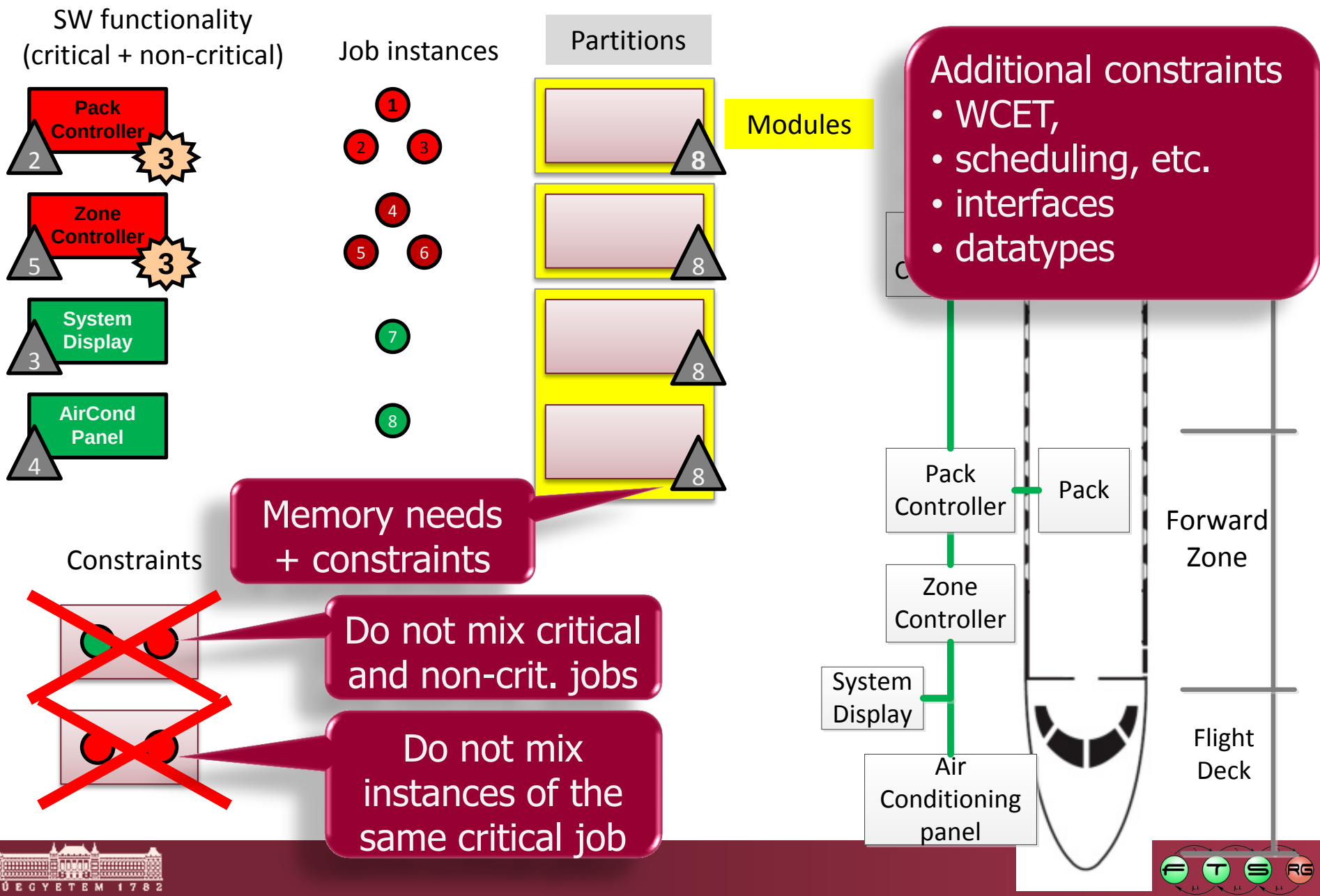
Designing ARINC653 configurations

SW functionality
(critical + non-critical)

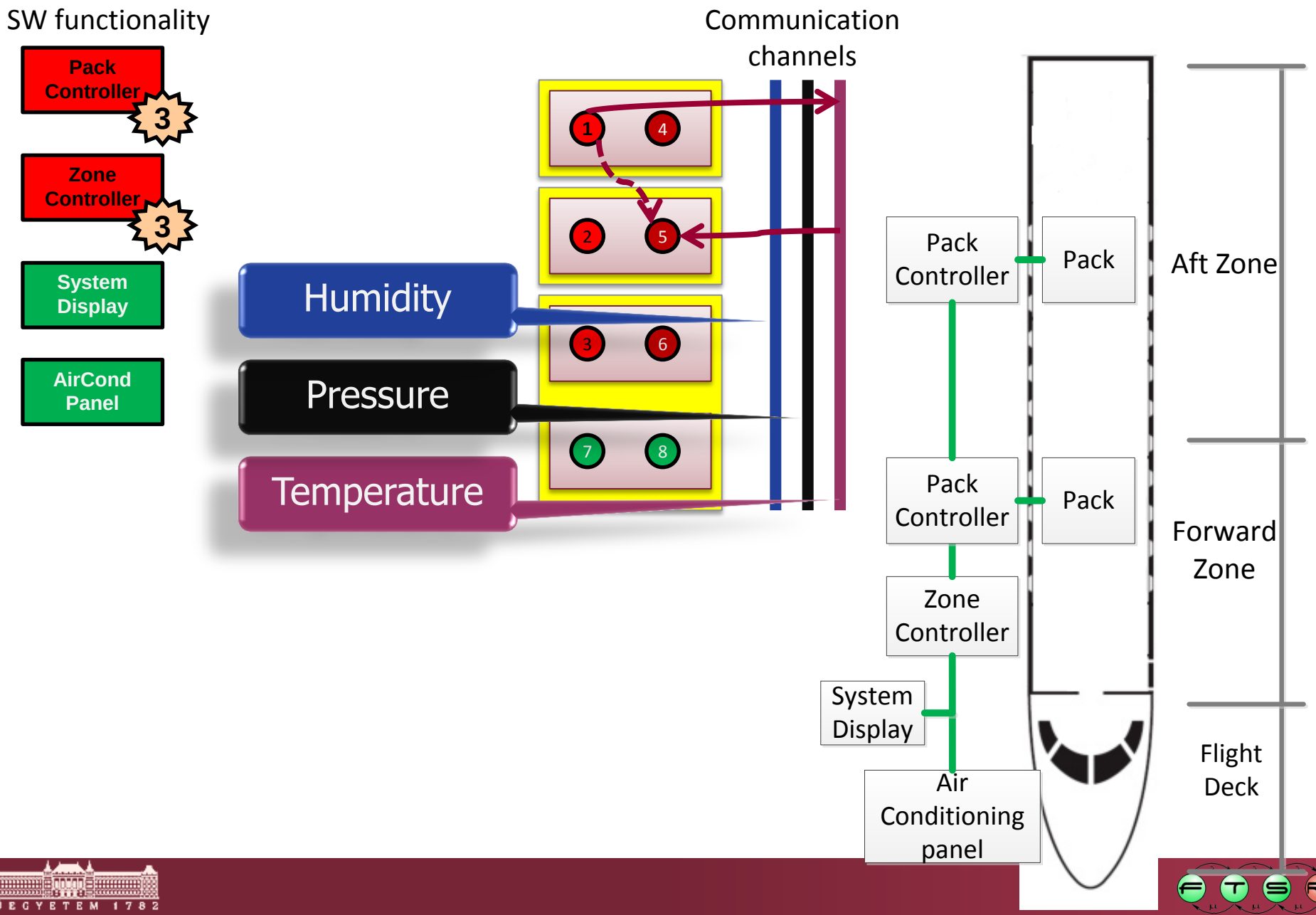




Job instances, Partitions, Modules

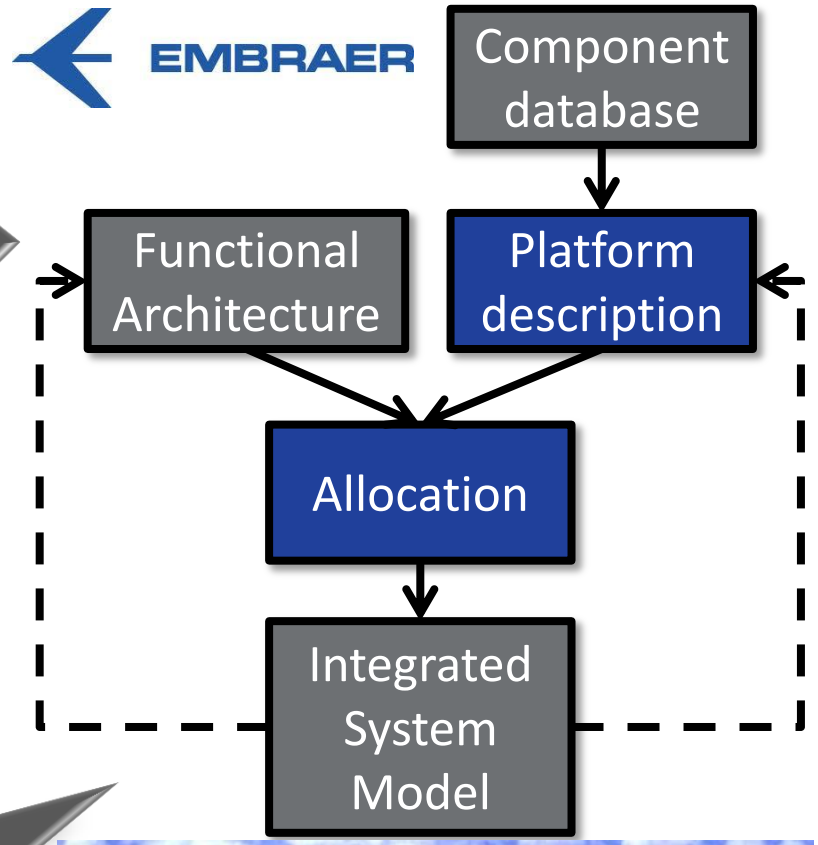


Allocating communication channels



Model Driven Development of IMA Configs

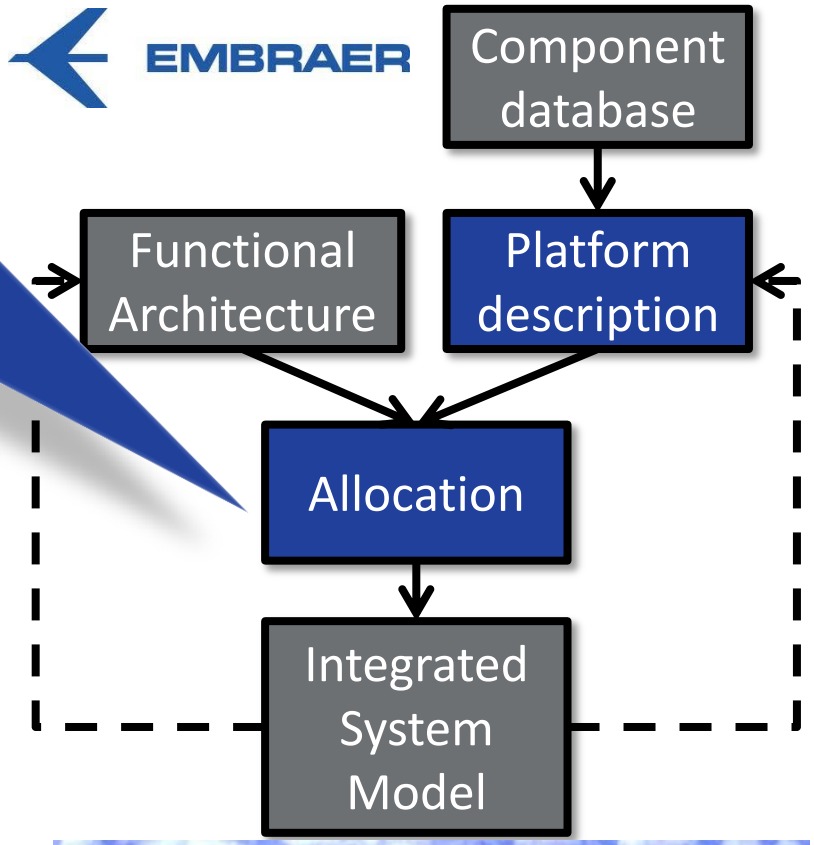
Inputs:
- Platform Independent Model (PIM) (functional + nonfunc. reqs; Simulink)
- Platform Description Model (PDM) for ARINC 653 (DSML)

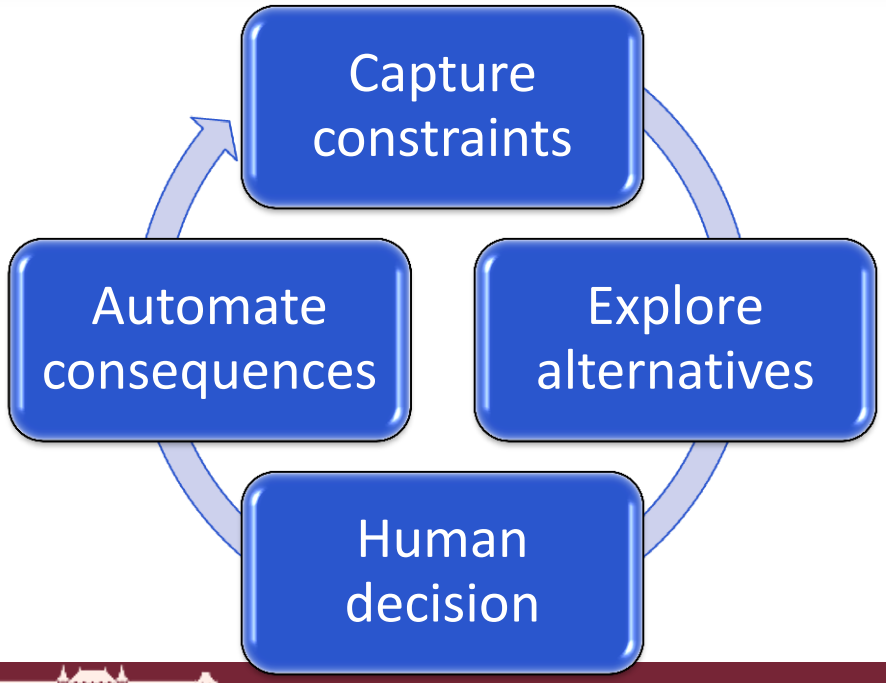


Output:
- Integrated system model
- Ready for simulation
- End-to-end traceability

Model Driven Development of IMA Configs

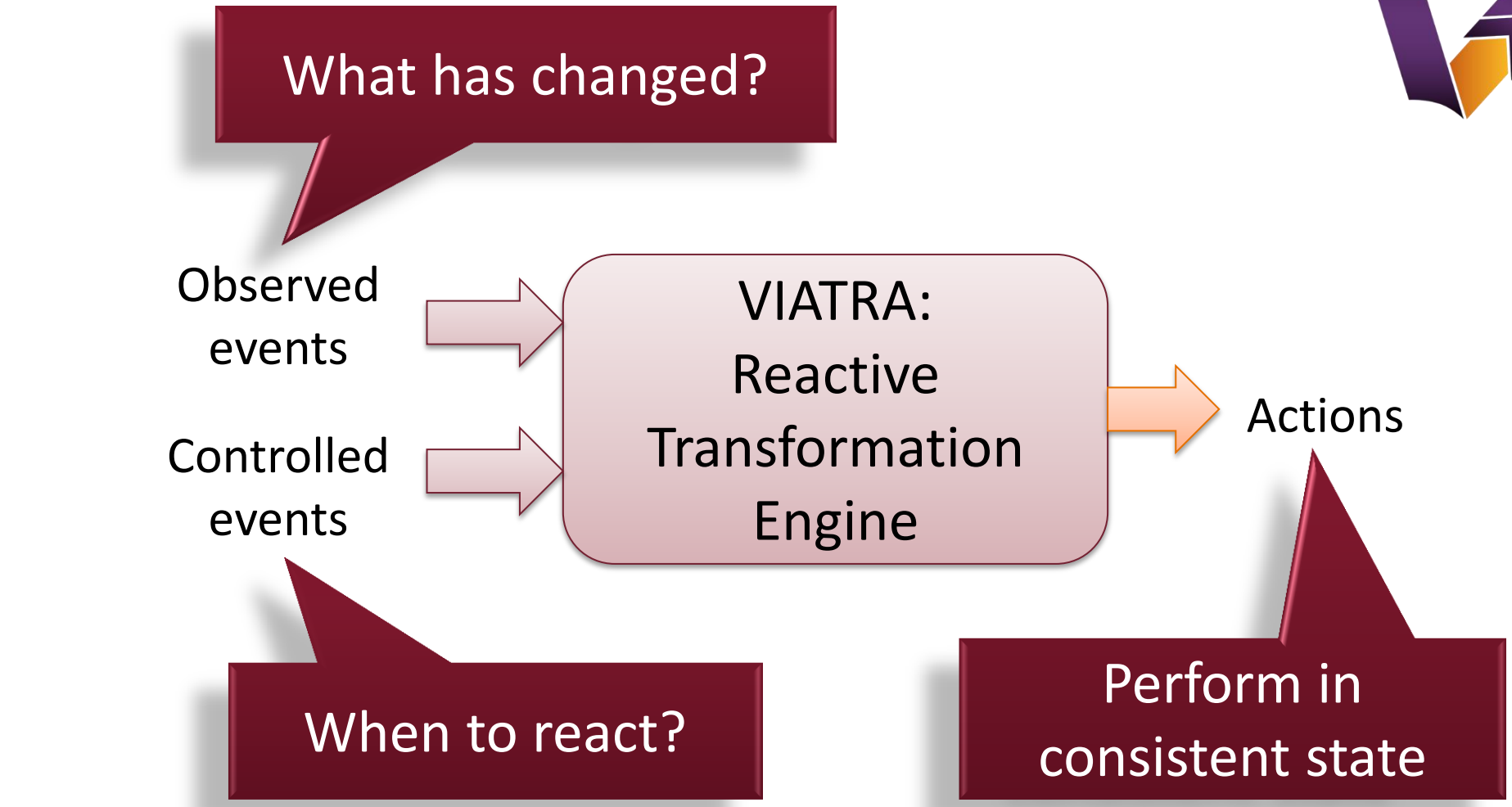
Model transformation chains:
- Designer-guided manual steps
- Automated steps
  - design space exploration
  - optimization
  - code generators
- Continuous validation of design rules



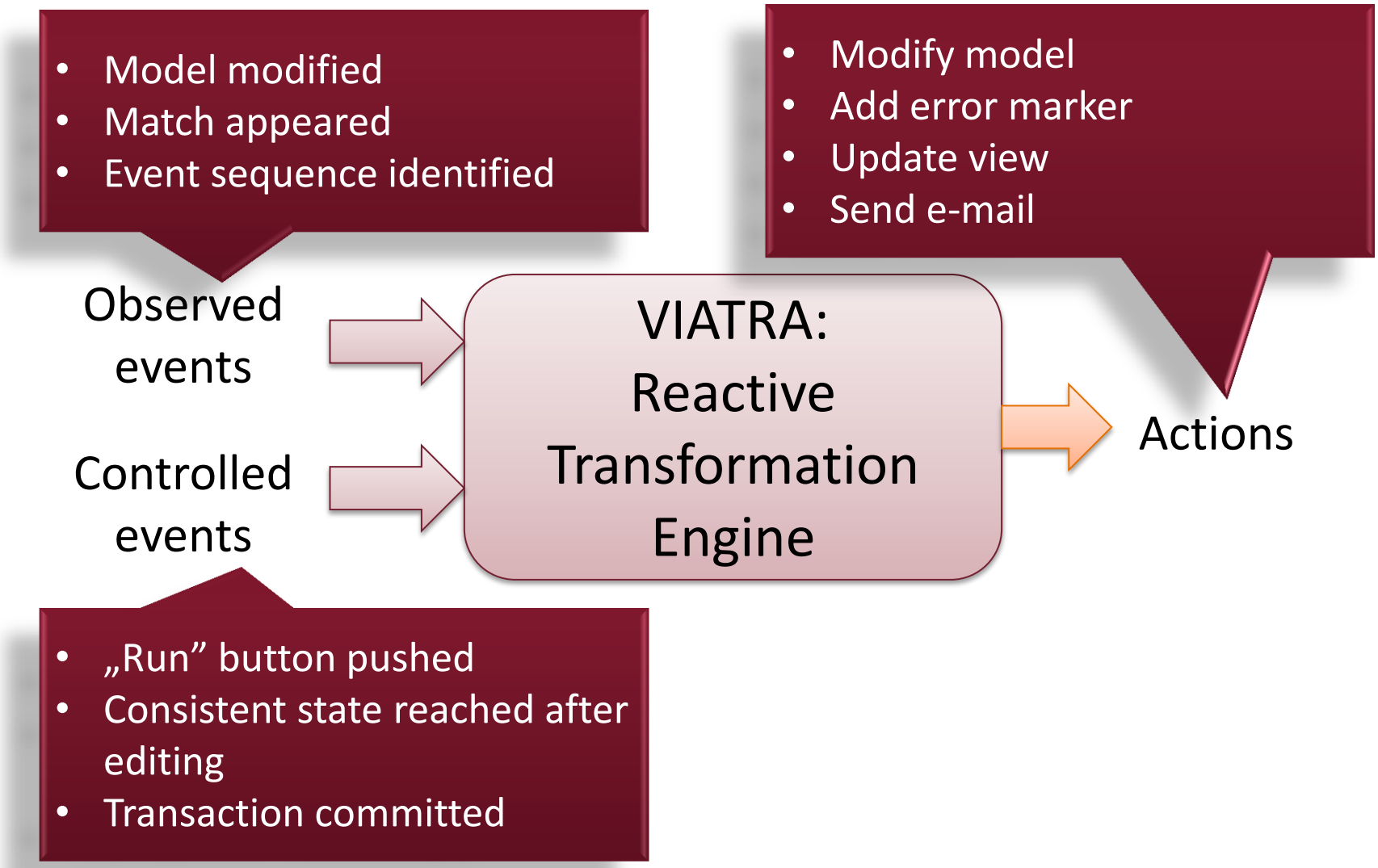


VIATRA: A Reactive Incremental Transformation Platform

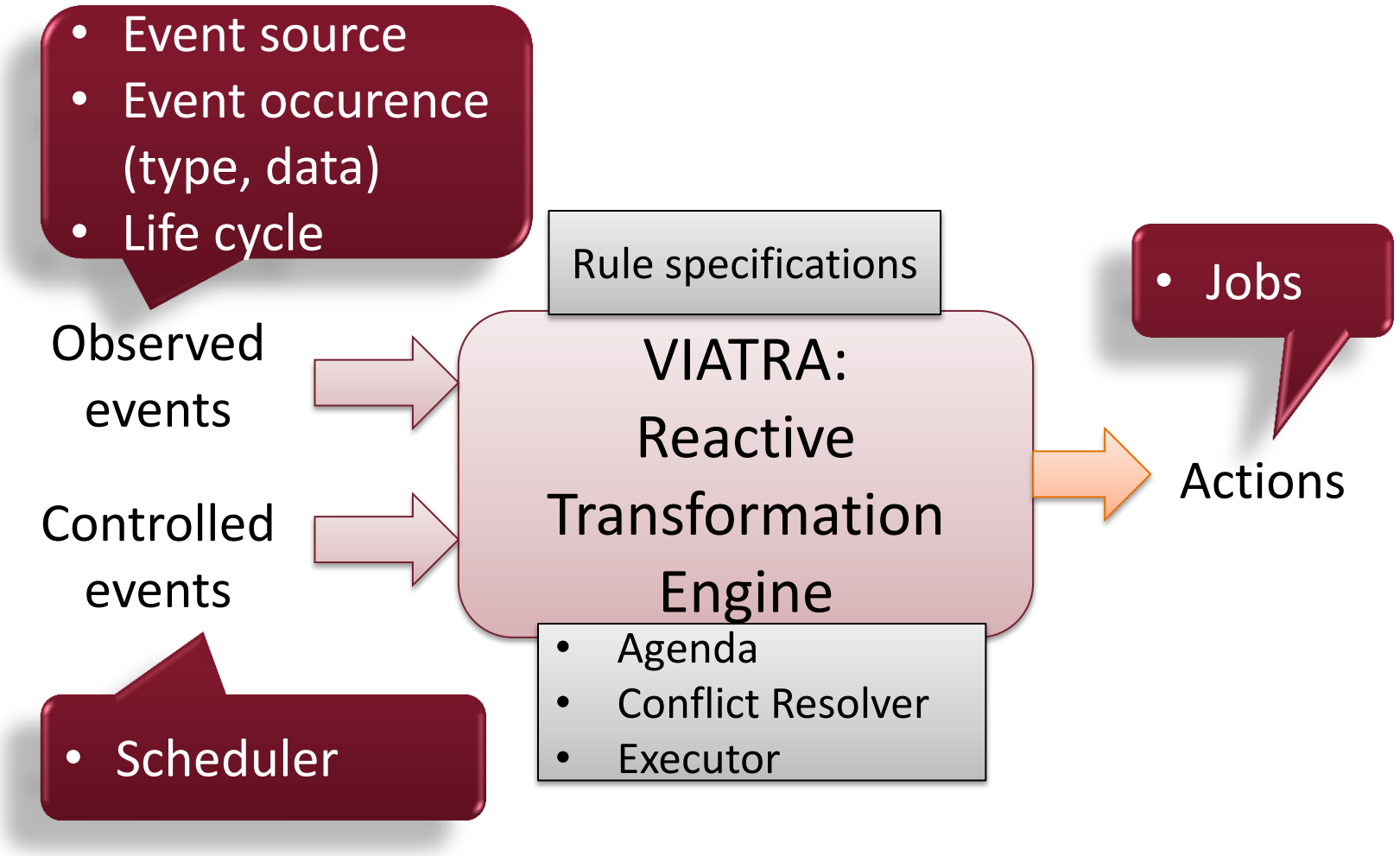
Reactive Event Driven Transformations



Reactive Event Driven Transformations



Reactive Event Driven Transformations



Language Example

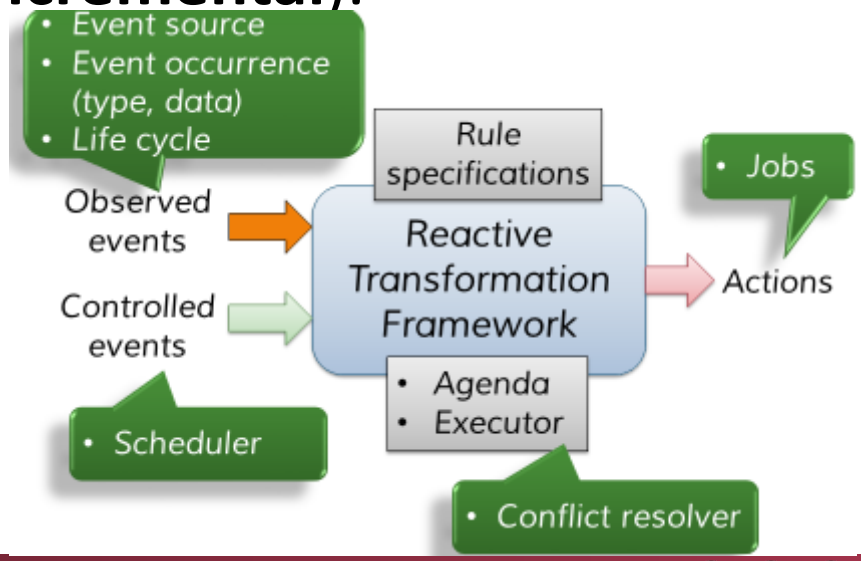
```
pattern someCondition( param1, param2 ) {...}
```

Query language

Xtend (Java)

```
val rule = createRule().precondit                    on).
  action[ match |  // perform action ].build
val incrRule = createRule().precondition(someCondition).
  lifecycle(ActivationLifecycles.incremental).
  action(::Appeared)[
   match | // perform action].
  action(::Disappeared)[
   match | // perform action].
  build
```

Event data



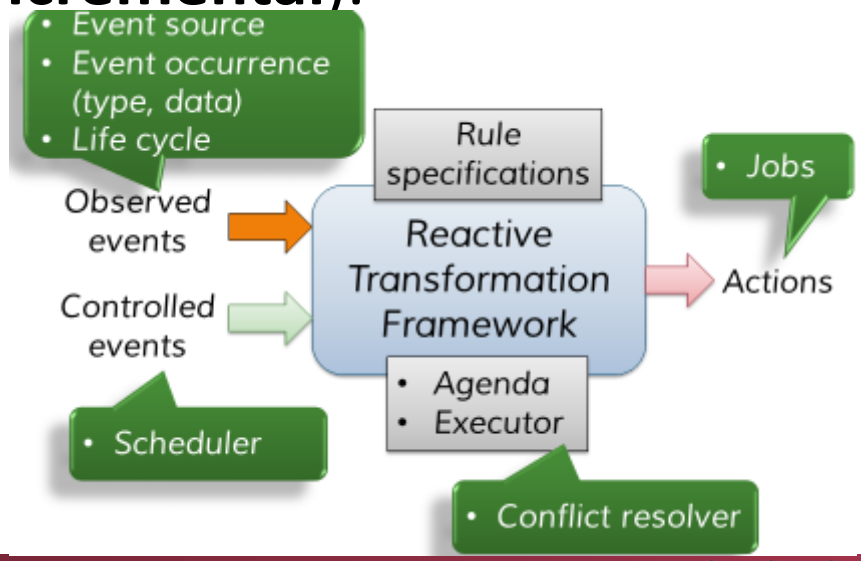
Language Example

pattern someCondition(param1, param

Rule specification

Xtend (Java)

```
val rule = createRule().precondition(someCondition).
  action[ match |  // perform action ].build
val incrRule = createRule().precondition(someCondition).
  lifecycle(ActivationLifecycles.incremental).
  action(::Appeared)[
   match | // perform action].
  action(::Disappeared)[
   match | // perform action].
  build
```



Language Example

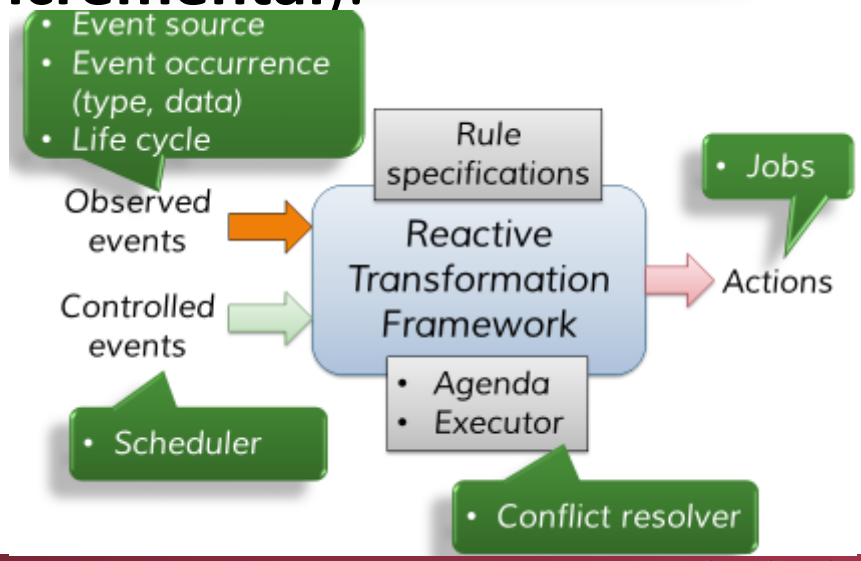
```
pattern someCondition( param1, param2 ) {...}
```

Query language

Xtend (Java)

```
val rule = createRule().precondition(someCondition).
  action[ match |  // perform action ].build
val incrRule = createRule().preconditic
  lifecycle(ActivationLifecycles.incremental).
  action(::Appeared)[
   match | // perform action].
  action(::Disappeared)[
   match | // perform action].
  build
```

Observed events



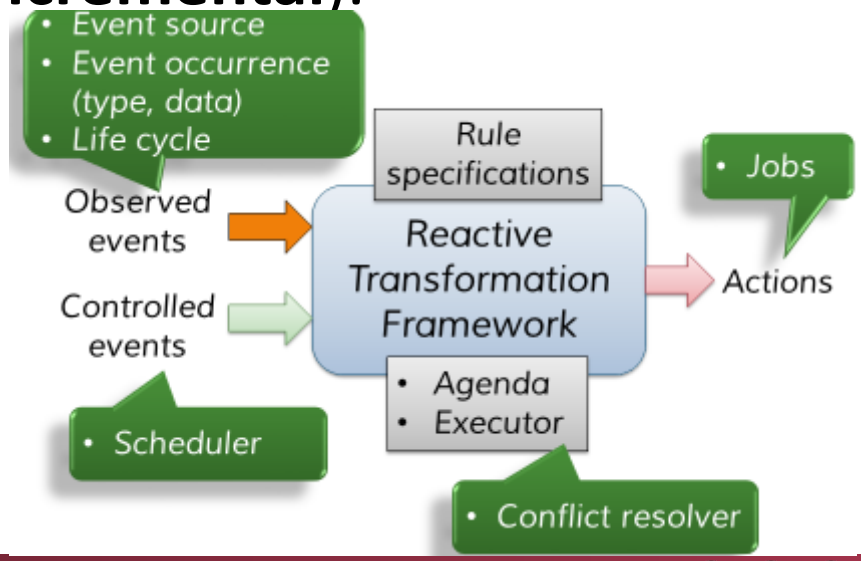
Language Example

```
pattern someCondition( param1, param2 ) {...}
```
Query language

Xtend (Java)

```
val rule = createRule().precondition(someCondition).
 action[ match |  // perform action ].build
val incrRule = createRule().precondition(someCondition).
  lifecycle(RelationLifecycles.incremental).
  actio
   match | // perform action].
  action(::Disappeared)[
   match | // perform action].
  build
```

Job specification

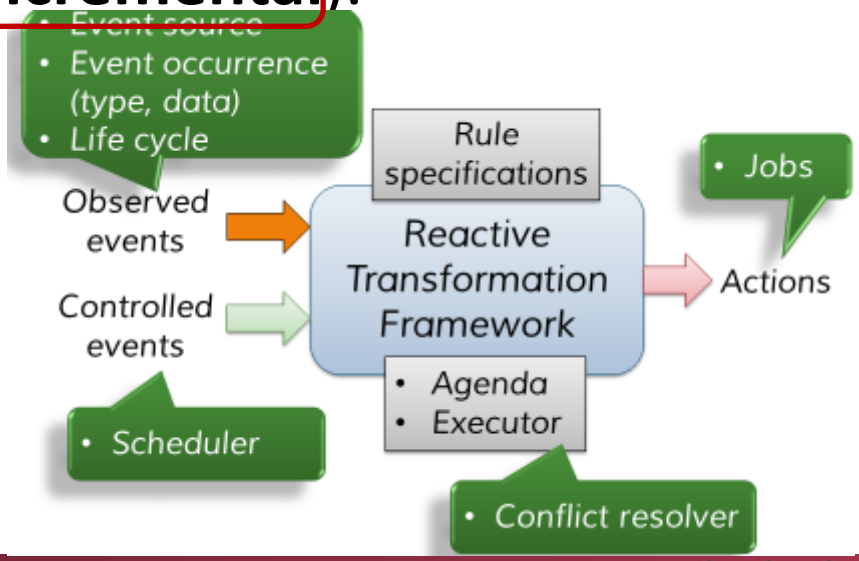


Language Example

```
pattern someCondition( param1, param2 ) {...}
```

```
val rule = createRule().precondition(some
  action[ match |  // perform action ].build
val incrRule = createRule().precondition(someCondition).
  lifecycle(ActivationLifecycles.incremental).
  action(::Appeared)[
   match | // perform action].
  action(::Disappeared)[
   match | // perform action].
  build
```

Activation state-event transitions



Language Example

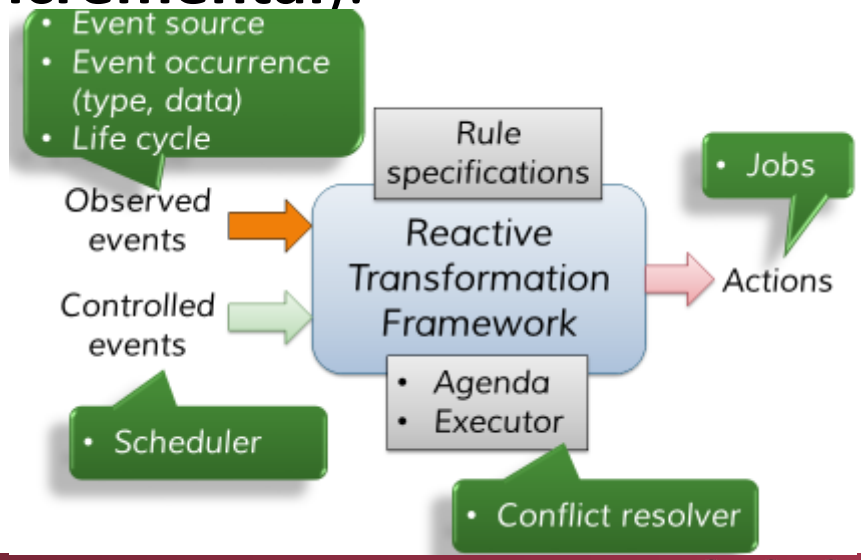
```
pattern someCondition( param1, param2 ) {...}
```

Query language

Xtend (Java)

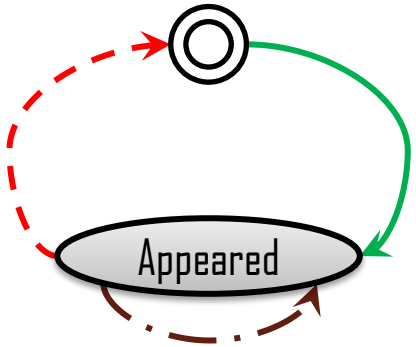
Jobs associated with event types

```
Rule().precondition(someCondition).
    // perform action ].build
var rRule = createRule().precondition(someCondition).
 lifecycle(ActivationLifecycles.incremental).
 action(::Appeared)[
  match | // perform action].
 action(::Disappeared)[
  match | // perform action].
 build
```

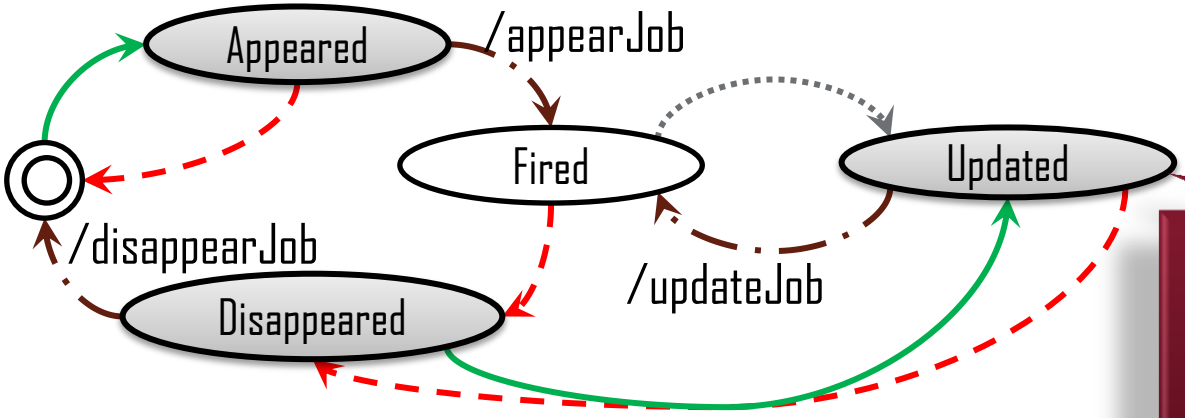


Activation Lifecycles

- Batch transformation



- Event-driven transformation



Phases

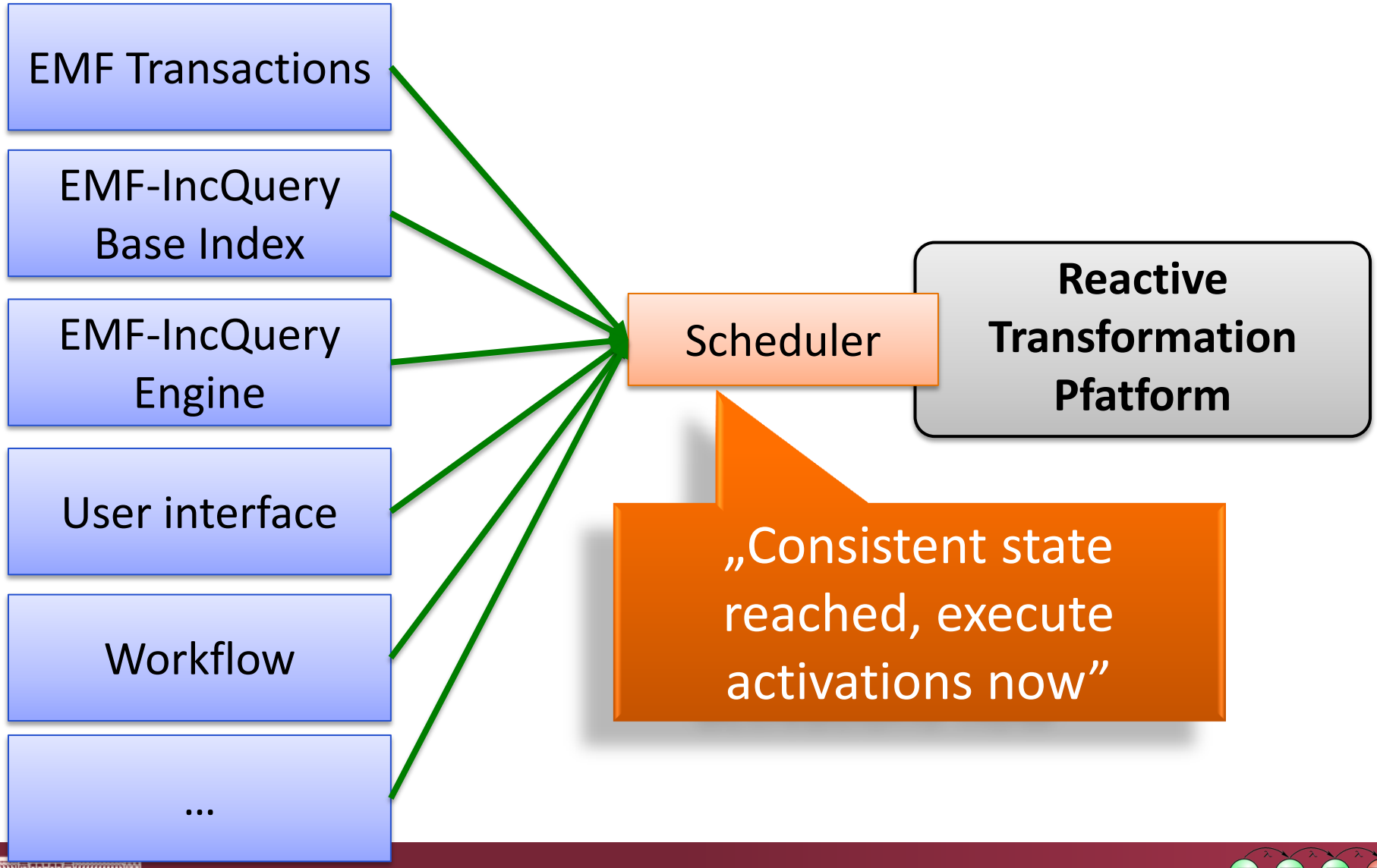
- Enabled
- Disabled
- Initial

Transitions

- Appear
- Update
- Disappear
- Fire

Only feature of event data object has changed

Scheduling


EMF Transactions
EMF-IncQuery Base Index
EMF-IncQuery Engine
User interface
Workflow
...
Scheduler
Reactive Transformation Pfatform
„Consistent state reached, execute activations now”

Conflict Resolution

- Multiple actions available
  - Different activations in the same rule
  - Activations of different rules
- Which activation to execute next?
- Conflict resolver can be selected
  - Global conflict set: deals with all rules
  - Scoped conflict set: selected rules

VIATRA: Overview of Features

- **Reactive MT Platform**
  - **MT Language**:
    - Internal DSL over Xtend
    - Transformation API
  - **MT Engine**:
    - Event-driven virtual machine
    - Batch + Incremental MTs
    - Control flow library
    - Compiles to Java
    - Debugger
    - High performance
  - Integrations:
    - EMF, IncQuery, Xtend, EMF-UML, …

Design Space Exploration

- Explore design model candidates
- Satisfying multiple criteria
- Rule based exploration
- Optimization

Complex Event Processing

- Detect complex event sequences
- Rule based reaction
- Xtext based language

Model Obfuscator

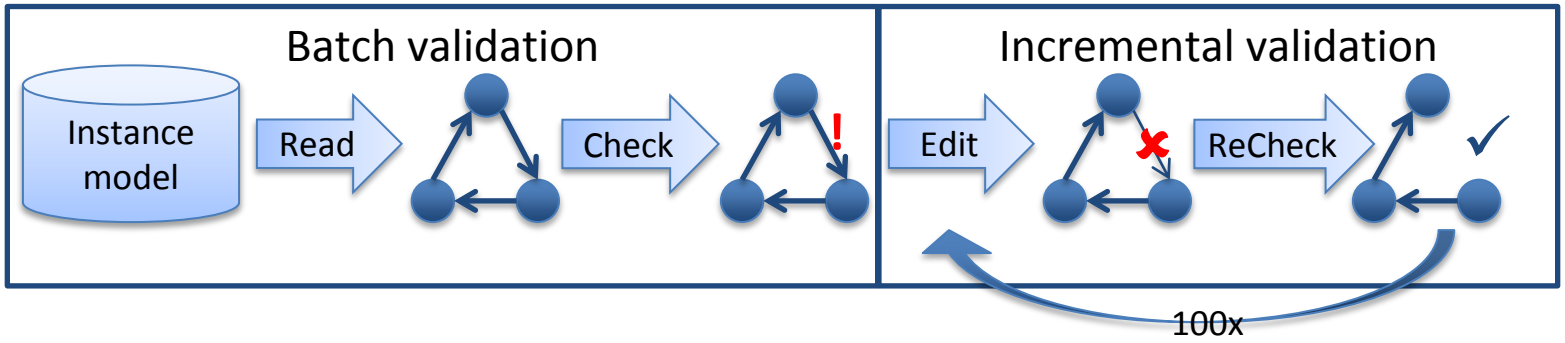
- Remove sensitive information from confidential models
- Original model ➔ Obfuscated model

Cross-technology benchmark for model validation

The Train Benchmark Case for Incremental Model Validation
(Transformation Tool Contest 2015)

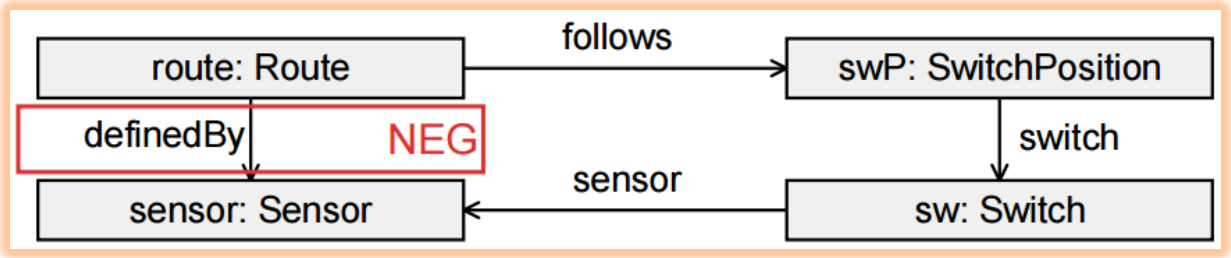
The Train Benchmark

- Model validation workload:
  - User edits the model
  - Instant validation of well-formedness constraints
  - Model is repaired accordingly
- Scenario:
  - Load + Check
  - Edit + Re-Check
- Models:
  - Randomly generated
  - Close to real world instances
    - Following different metrics
    - Customized distributions
  - Low number of violations
- Queries:
  - Two simple queries (<2 objects, attributes)
  - Two medium queries (4-7 joins, negation)
  - Two complex queries (7+ joins, transitive closure)
  - Validated match sets
- Transformations:
  - Modifies the model
  - Repair: remove existing violations
  - Inject: create new violations (matches)

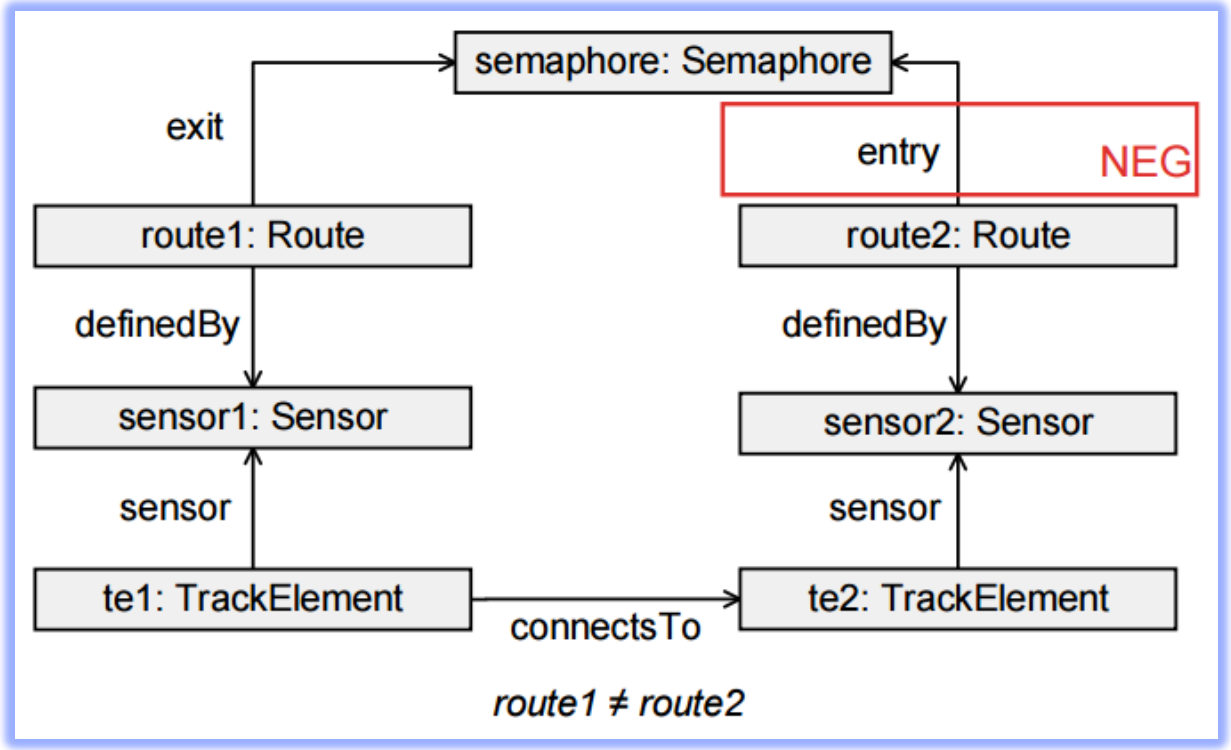


What Tools are Compared?

Selected Queries



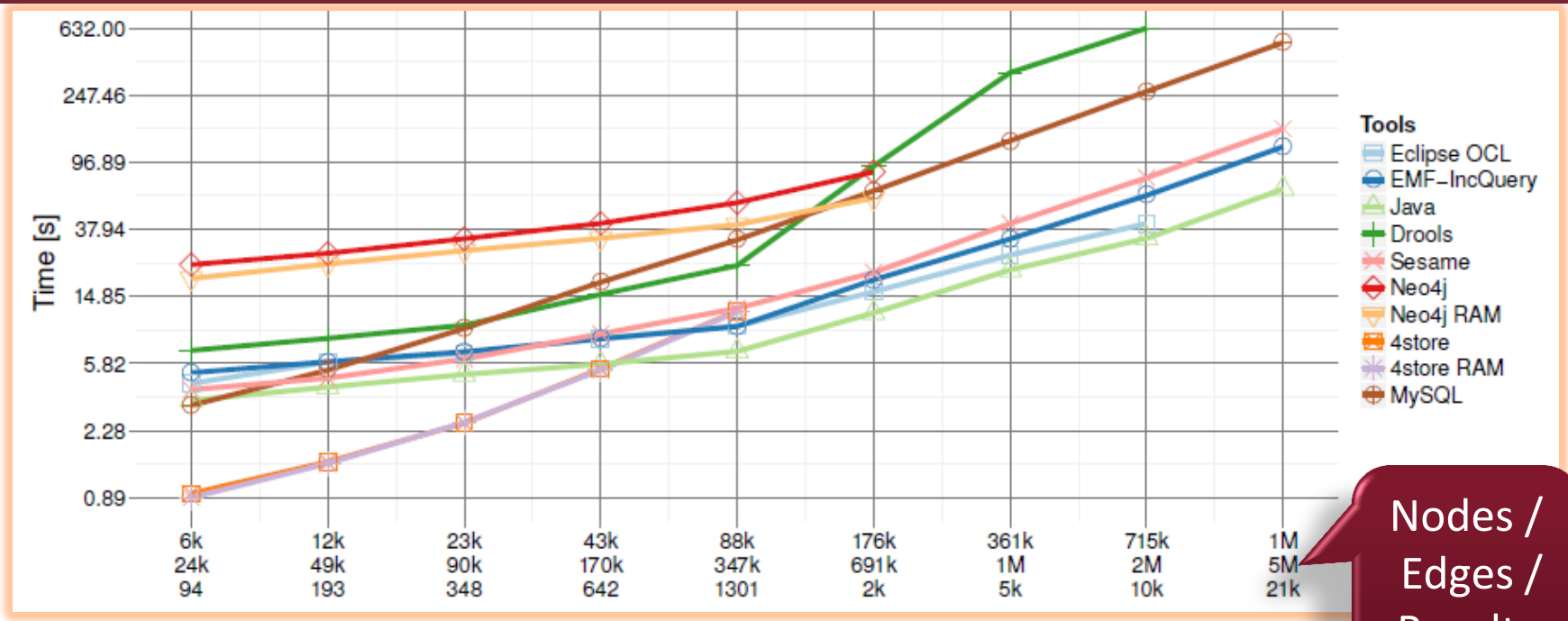
RouteSensor



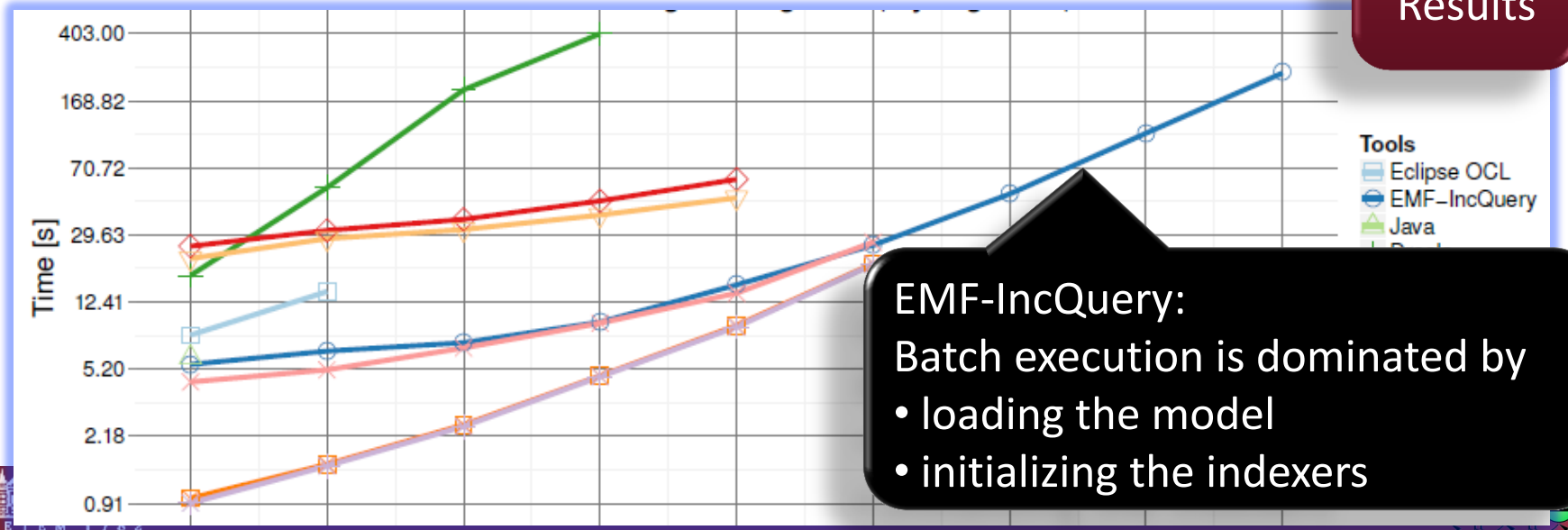
SemaphoreNeighbor

http://trainbenchmark.inf.mit.bme.hu for more details

Batch validation runtime

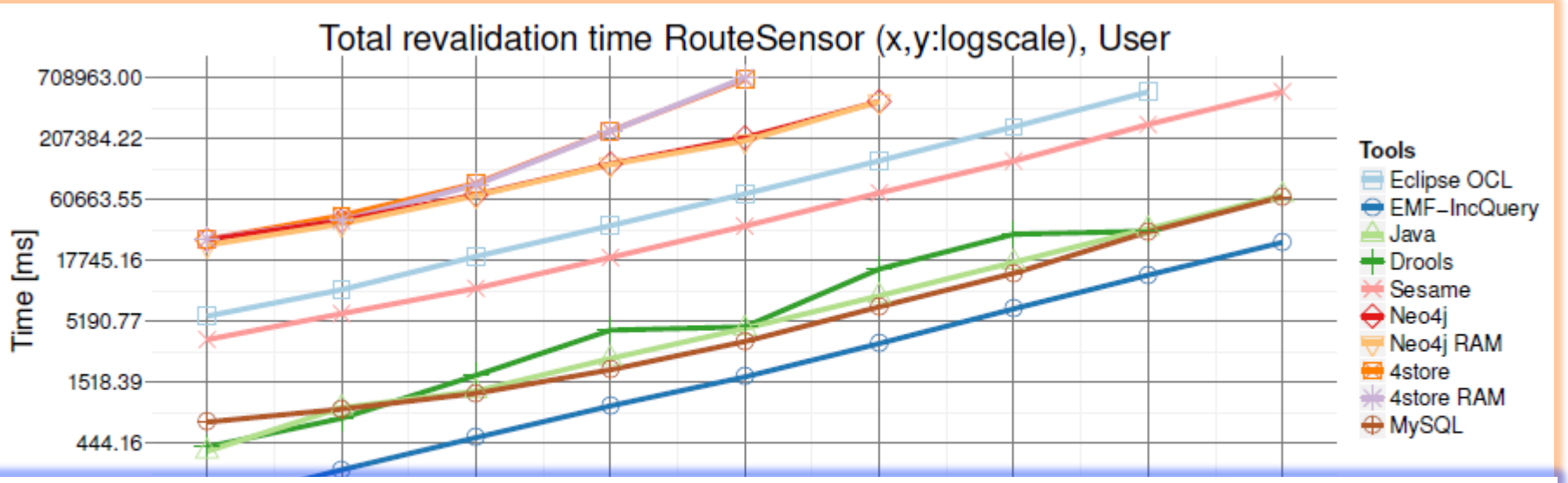


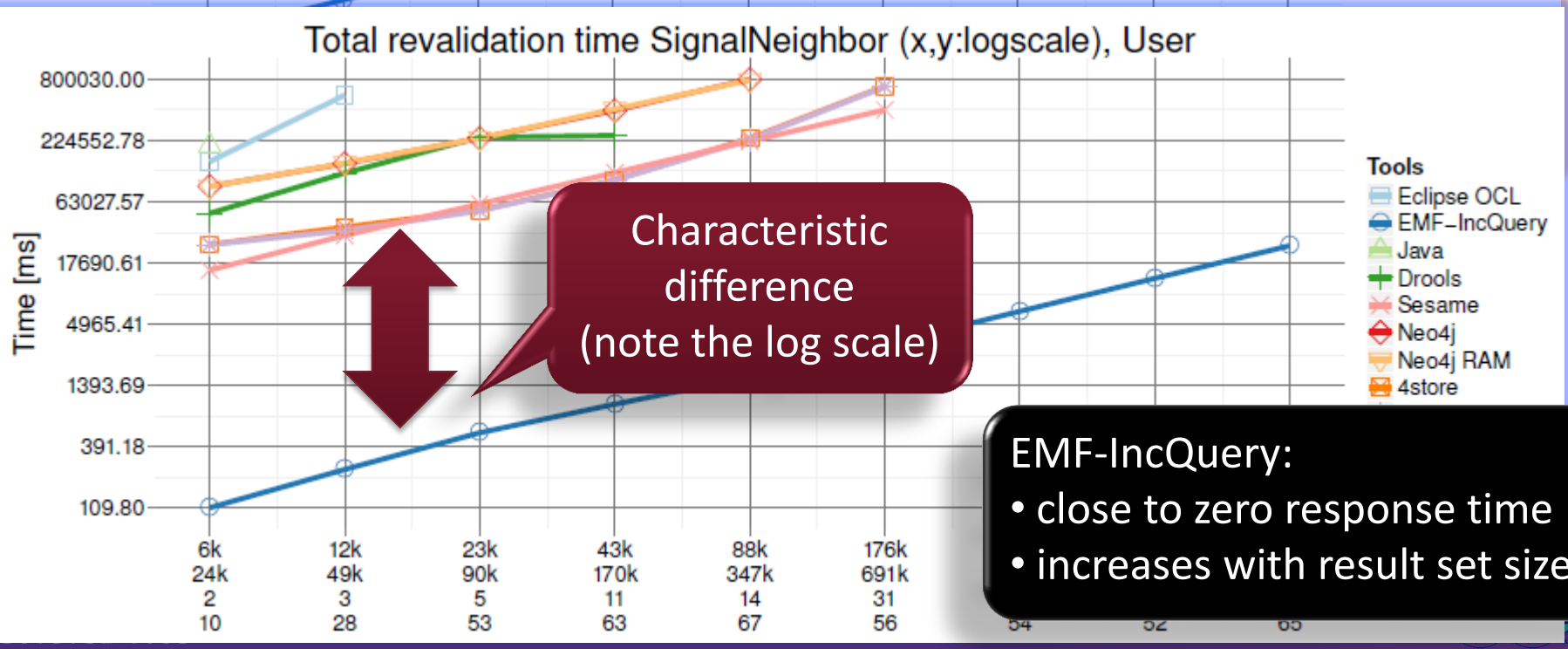
Nodes / Edges / Results



EMF-IncQuery:
Batch execution is dominated by
- loading the model
- initializing the indexers

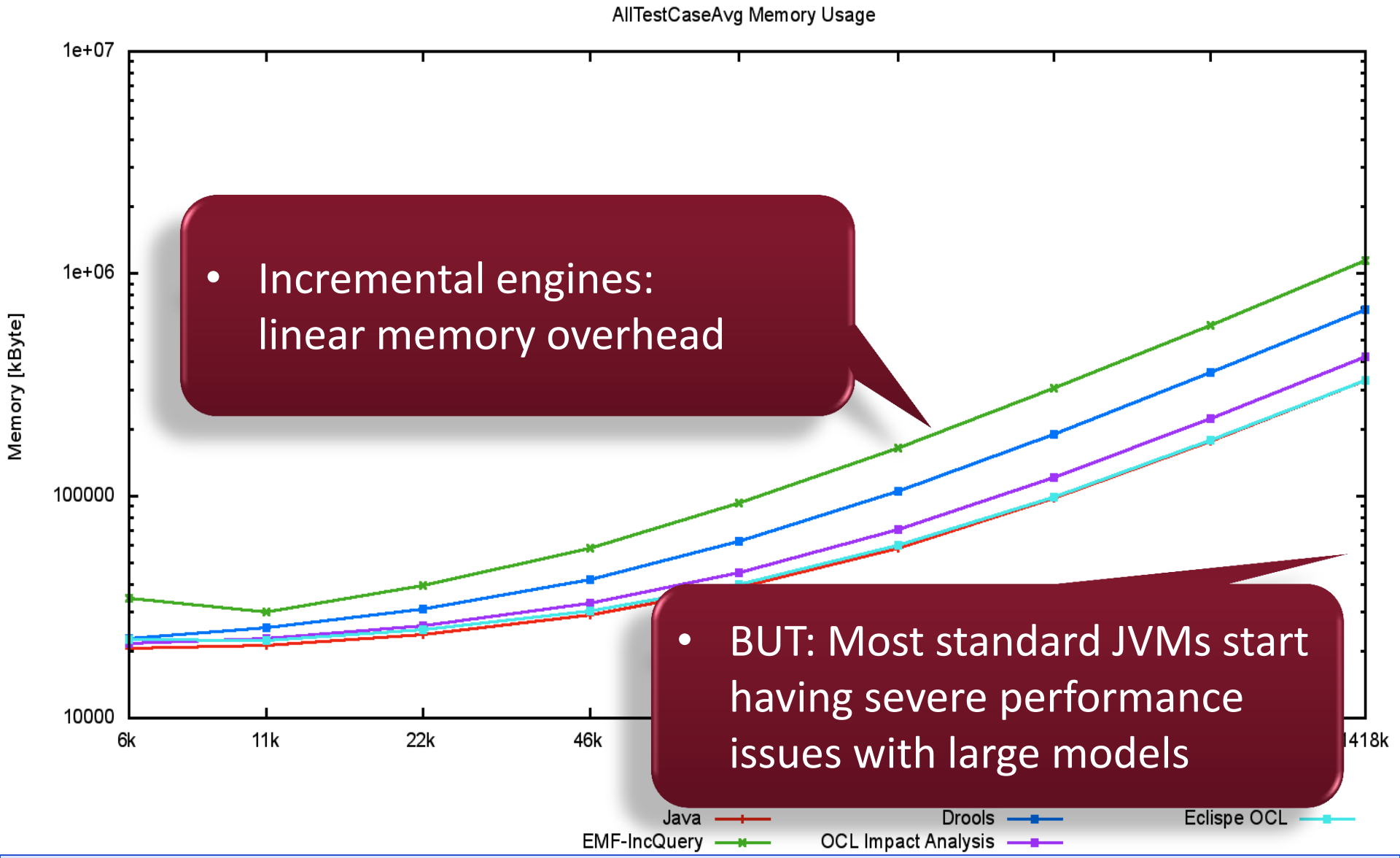
Re-validation time (complex queries)





EMF-IncQuery:
- close to zero response time
- increases with result set size

Memory usage



- Incremental engines: linear memory overhead
- BUT: Most standard JVMs start having severe performance issues with large models

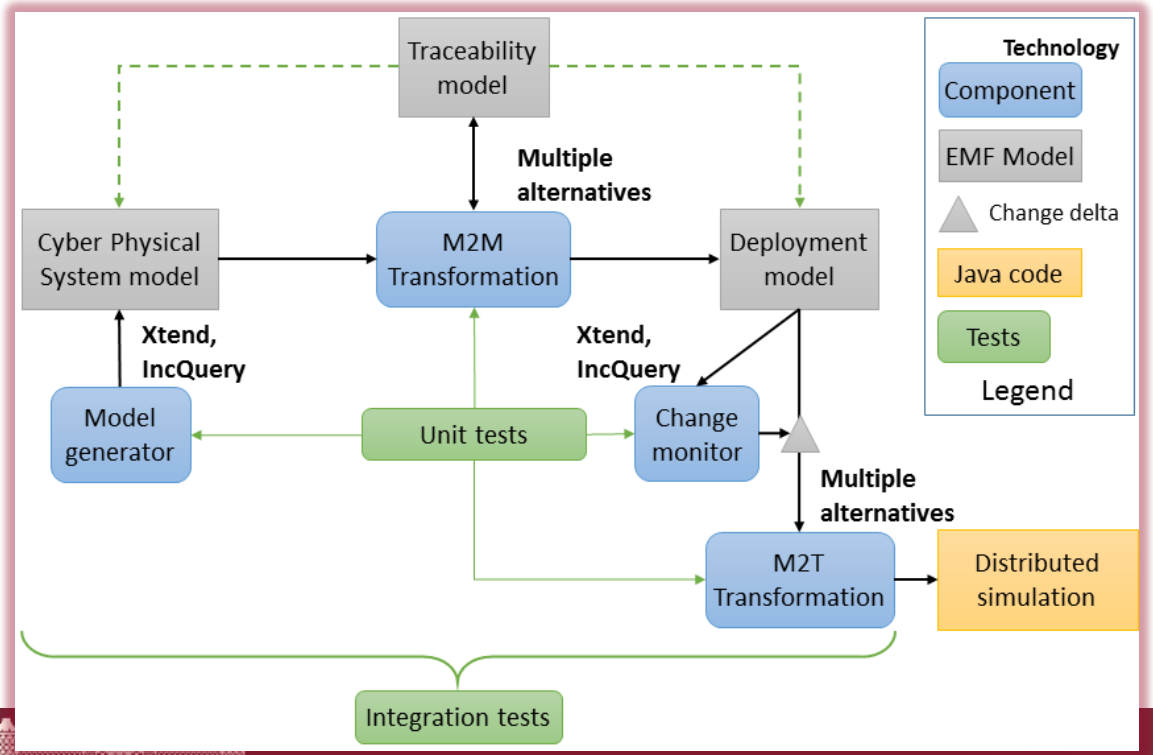
See also a MT benchmark: https://github.com/viatra/incquery-examples-cps /

Performance benchmarks

https://github.com/viatra/incquery-examples-cps

CPS Reallocation Benchmark

- Benchmark setup
  - Rule-based redeployment for cloud-based CPS
    - Model generator + Unit tests
    - M2M + M2T transformations
- Different target architecture / platform
  - Industrial (Low-Synch)
  - Client-Server
  - Publish-Subscribe



Test Scenario

- Different transformation variants
  - Batch
    - Xtend (2 versions)
    - IncQuery+Xtend
  - Incremental
    - Dirty (2 approaches)
    - Explicit traceability
    - Query-driven
    - Change-driven (VIATRA-EVM)
- Executions
  - First transformation execution
  - Small modification + (re)execution
- Environment
  - New machine with 16 GB RAM
- Parameters
  - 10 GB Heap
  - Maximum 10 minutes execution times for complete chain

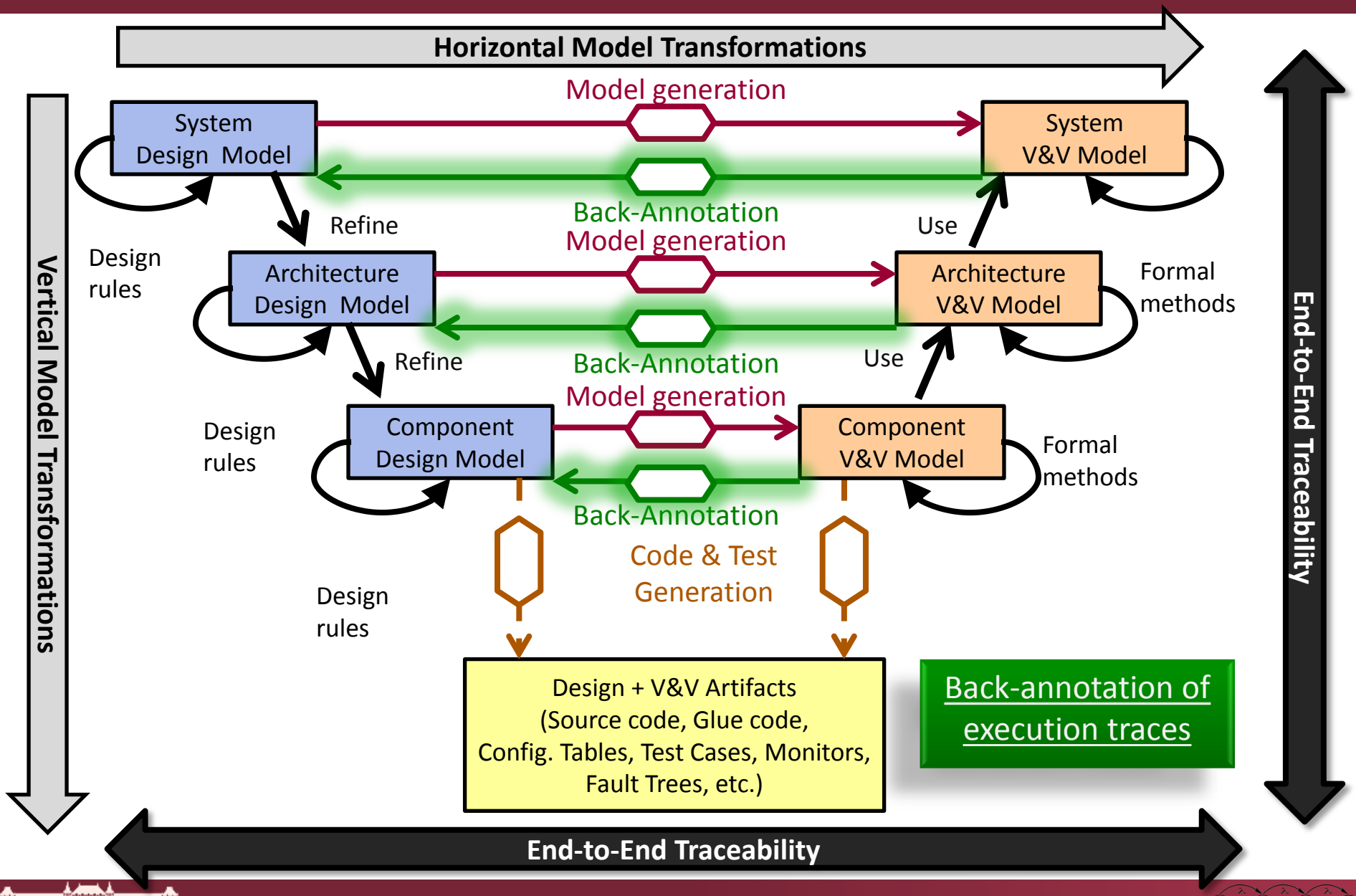
Scale	SRC Objects	SRC References	TRG Objects	TRG References	Trace Objects	Trace References	SUM Objects	SUM References
1	395	772	366	736	354	720	**1 115**	**2 228**
				5	762	1 535	**2 384**	**4 891**
				2	1 522	3 056	**4 750**	**10 725**
8	3 604	17 111		6 108	3 254	6 520	**10 124**	**29 739**
16	7 820	89 193	7 124	12 395	7 112	14 236	**22 056**	**115 824**
32	17 714	594 181	16 308	24 837	16 297	32 605	**50 319**	**651 623**
64	43 795	4 424 529	40 960	50 028	40 948	81 908	**125 703**	**4 556 465**

Trace model's size similar to target model

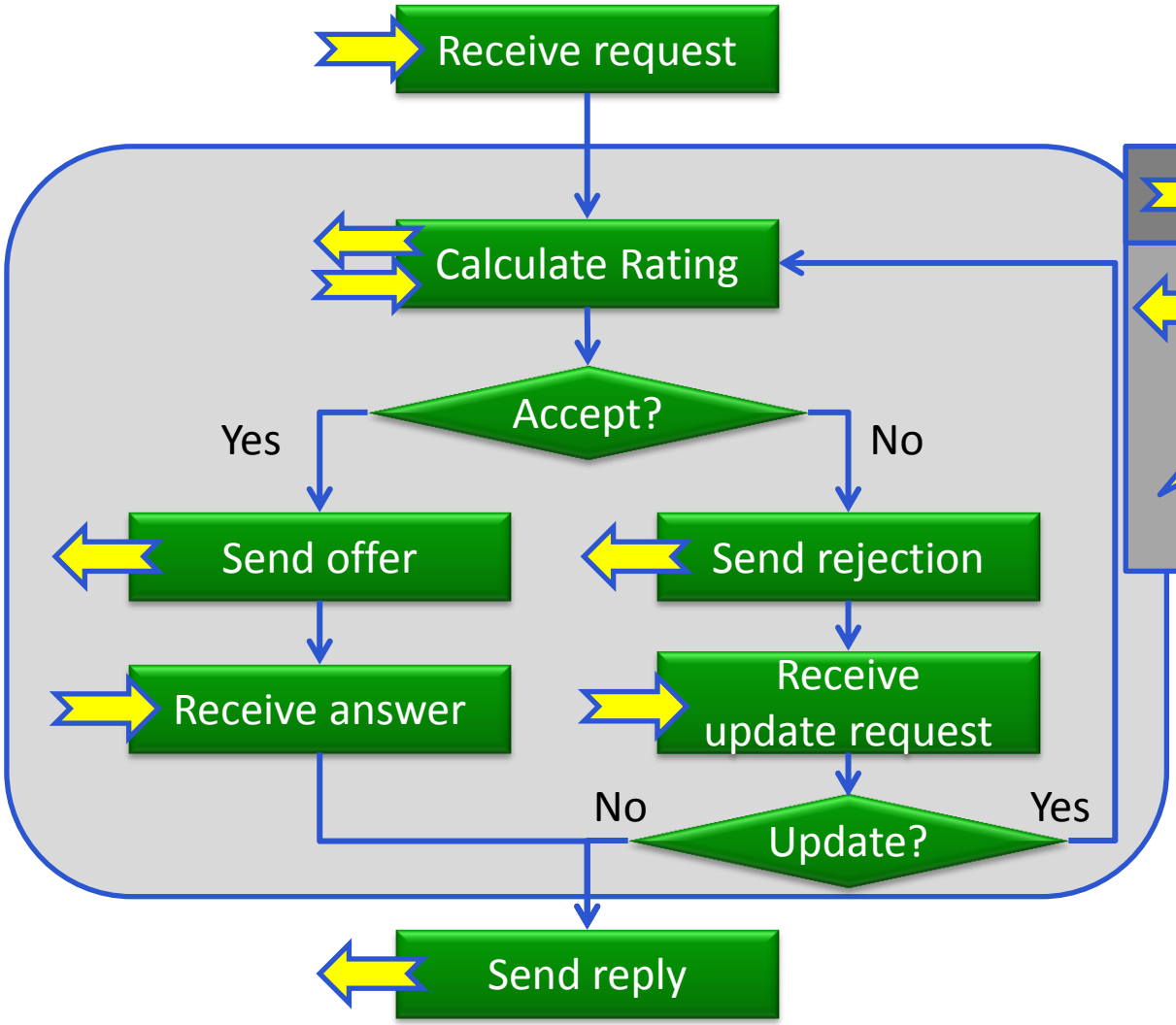
Back-annotation of Verification Results

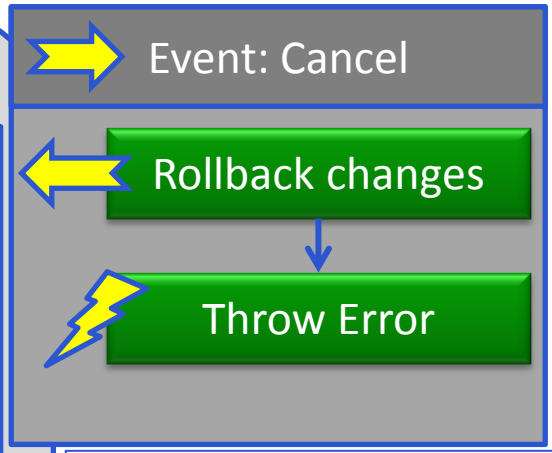
Based on Ábel Hegedüs' PhD thesis

Overview: Back-Annotation of Execution Traces



Model Analysis: Motivation for BPEL



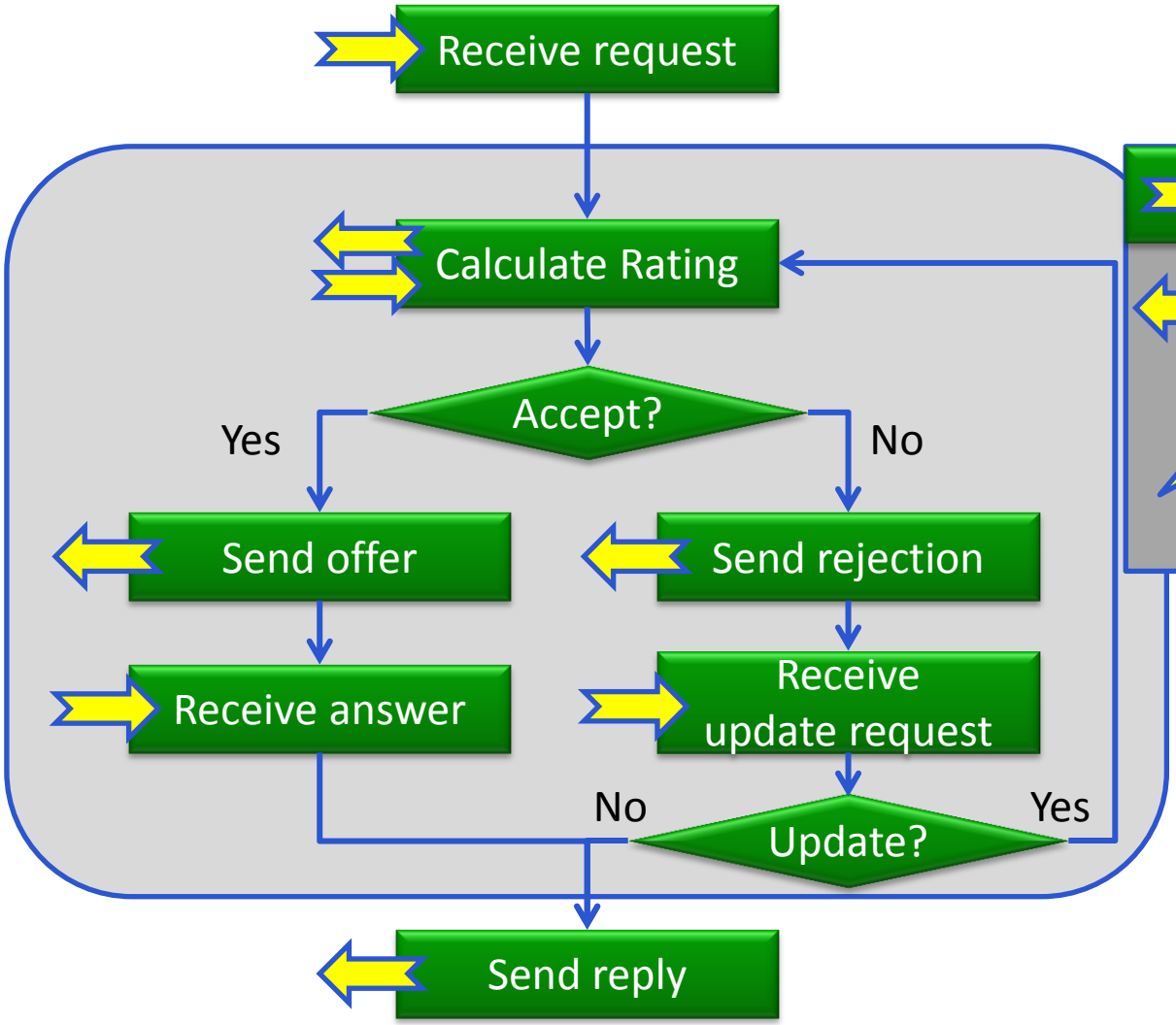


Requirement:

Every received request must result in a reply!

Will the business process assure this?

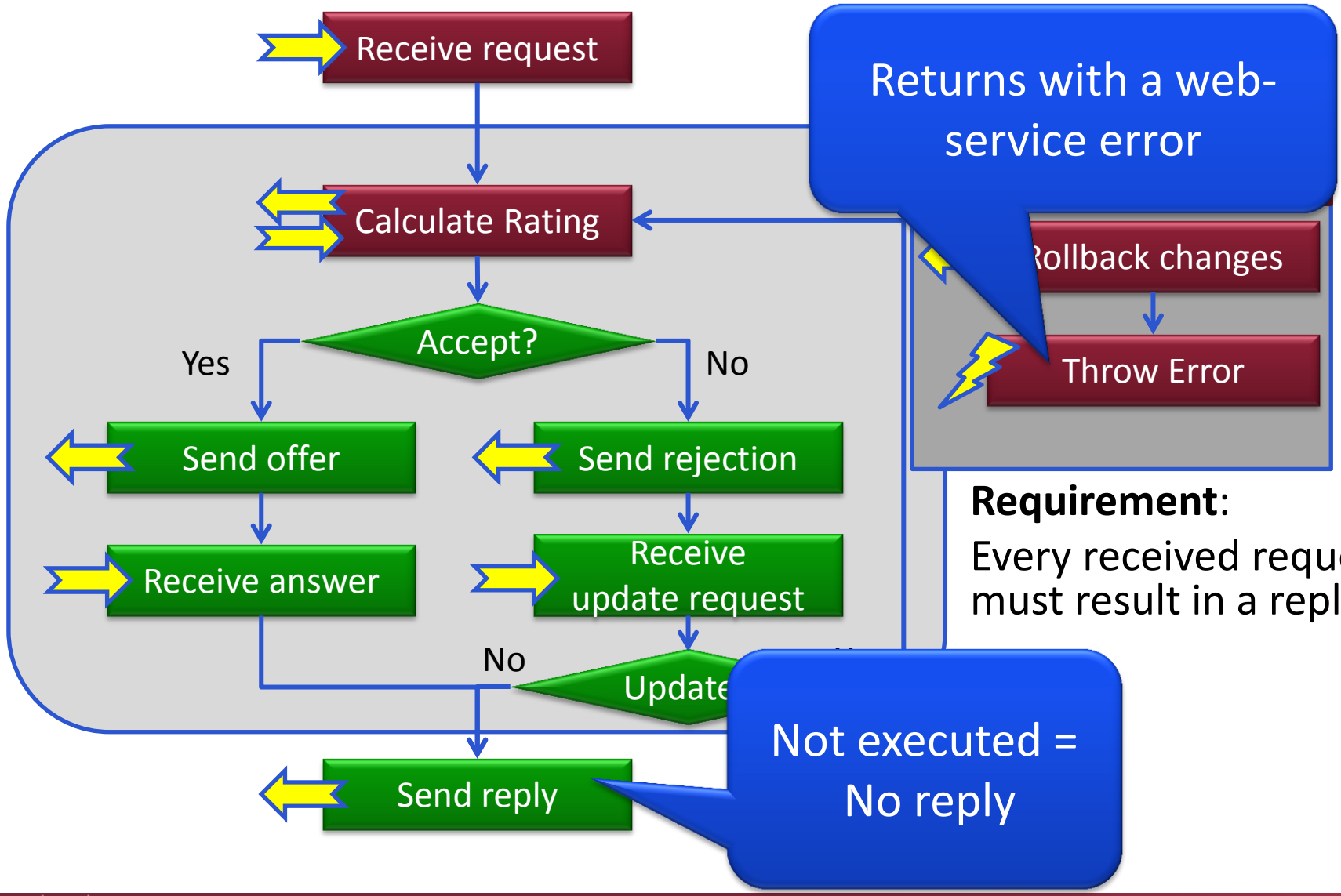
Motivating scenario (cont.)



Requirement:

Every received request must result in a reply!

Motivating scenario (cont.)



Requirement:

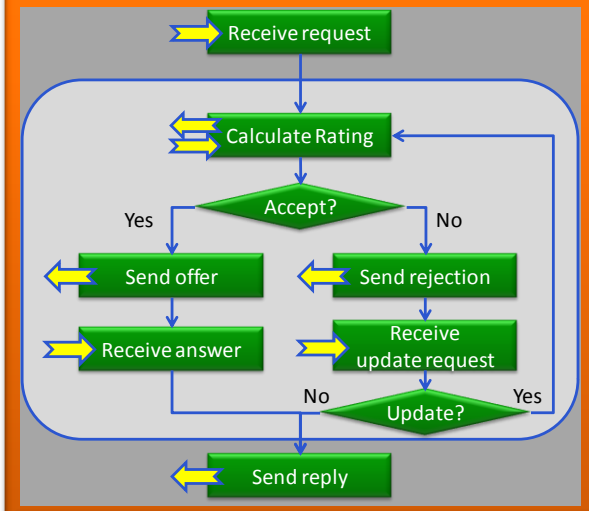
Every received request must result in a reply!

Model Based Analysis

System design

Mathematical analysis

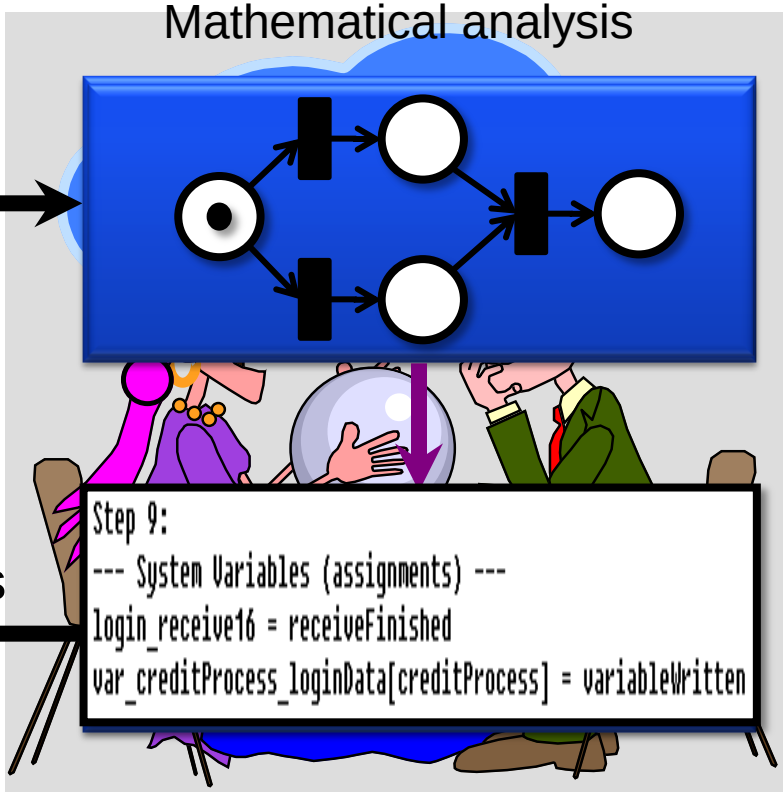
High-level System Model



Model generation

Fix problem

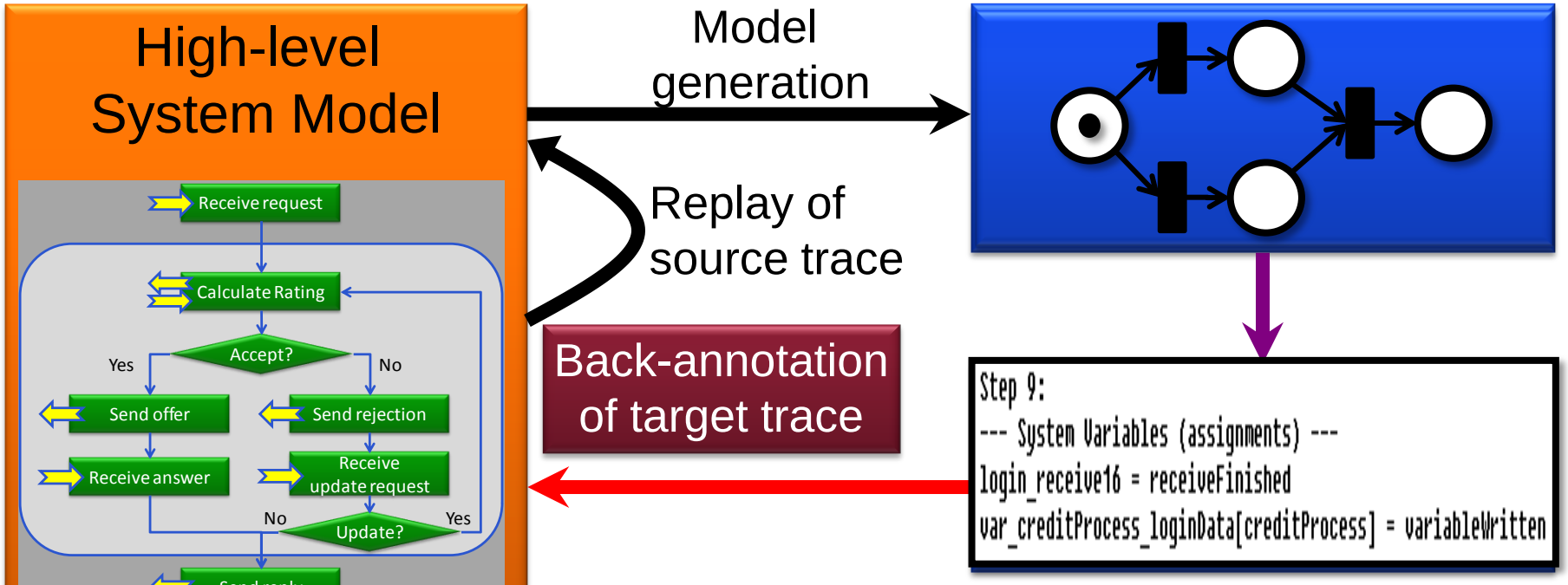
List of inconsistencies



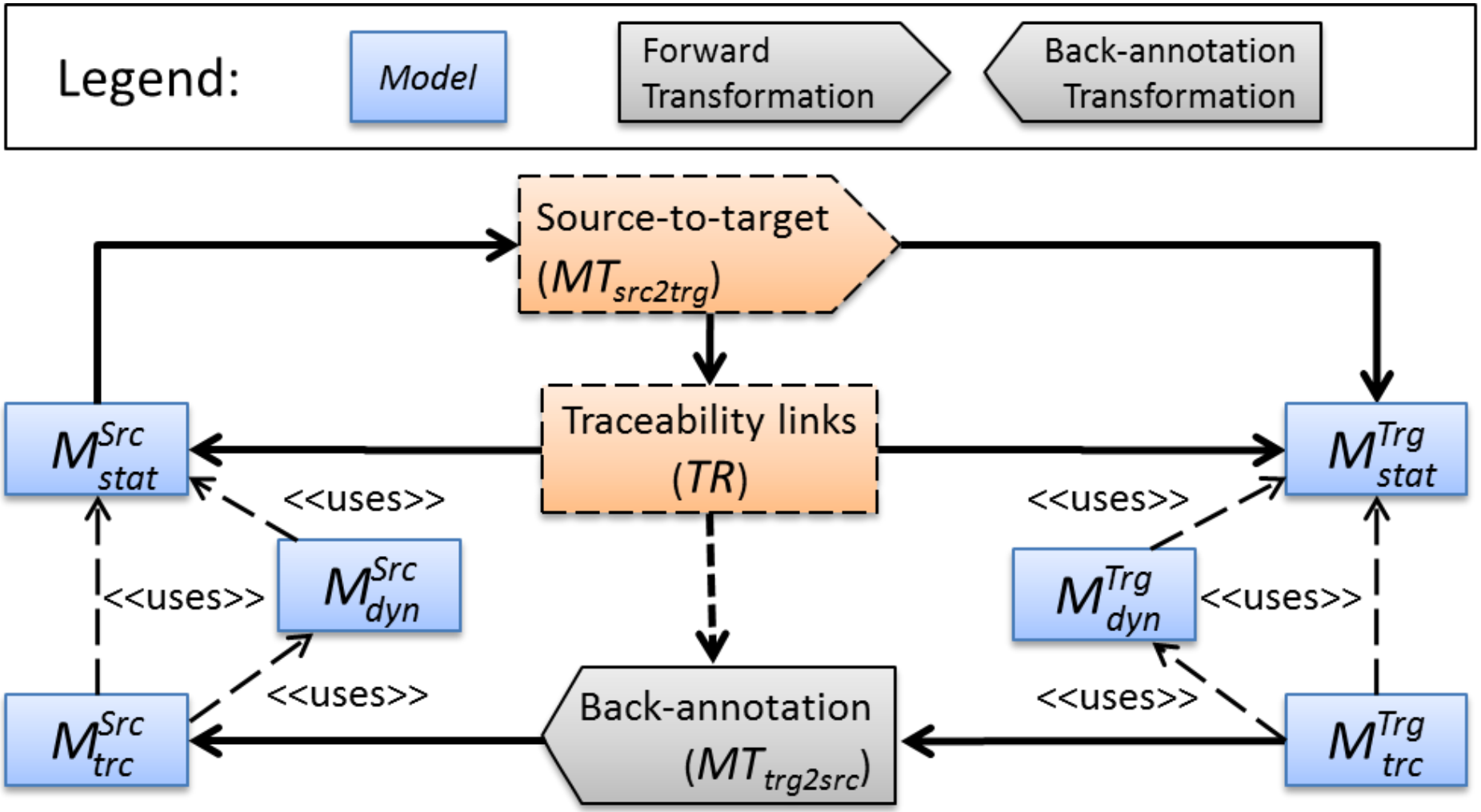
Counter-example / Execution traces

```
Counterexample:
========================
Path
========================
Step 0:
--- System Variables (assignments) ---
ba-pc!1 = 1
Accept_sequence36 = sequenceNotYetStartable
calculateOverAllRating_invoke33 = invokeNotYetStartable
Cancel_sequence22 = sequenceNotYetStartable
createNewCreditRequest_receive32 = receiveNotYetStartable
creditProcess_Process = scopeNotYetStartable
creditProcess_ProcessFaultCont = faultName_invalidVariables
creditProcess_ProcessFaultVar =
varType_tns_creditProcessRequestMessage
creditProcess_ProcessFinished = scopeNotFinished
cycleCore_sequence31 = sequenceNotYetStartable
Decline_sequence41 = sequenceNotYetStartable
else40 = elseNotYetStartable
getCustomerData_invoke17 = invokeNotYetStartable
login_receive16 = receiveNotYetStartable
Login_sequence15 = sequenceNotYetStartable
logout_invoke46 = invokeNotYetStartable
main_scope18 = scopeNotYetStartable
main_scope18FaultCont = faultName_invalidVariables
main_scope18FaultVar =
varType_tns_creditProcessResponseMessage
main_scope18Finished = scopeNotFinished
main_sequence14 = sequenceNotYetStartable
offer_reply38 = replyNotYetStartable
```

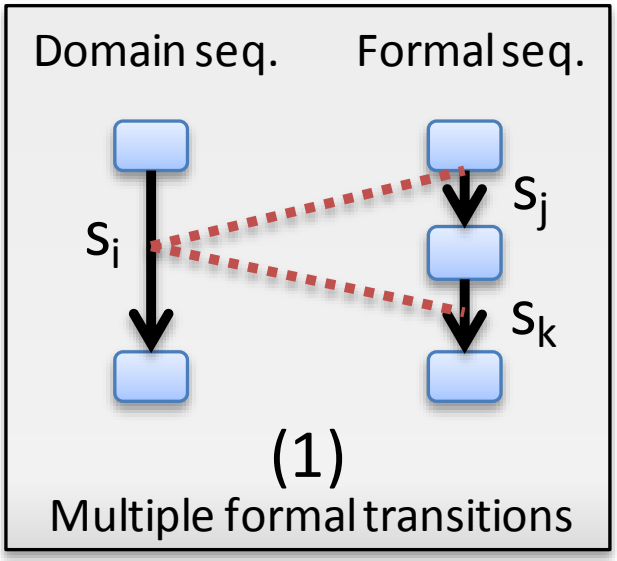
Back-Annotation of Counter Example Traces

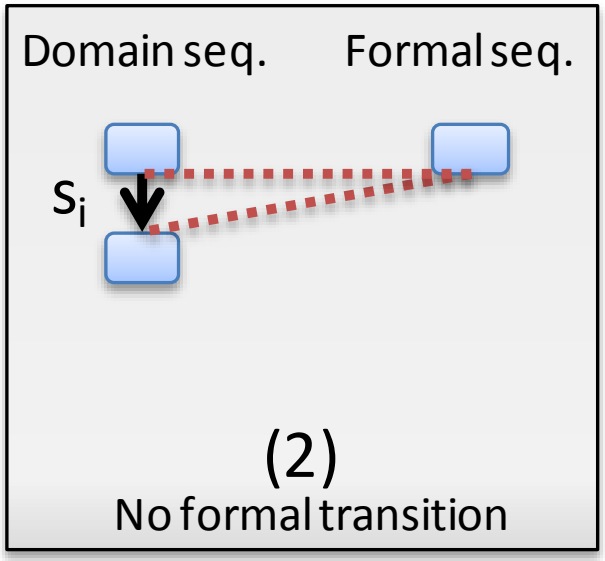


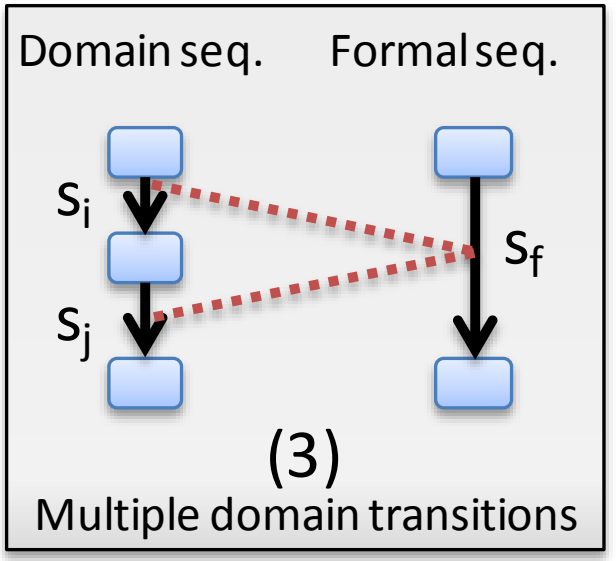
Overview of Back-annotation

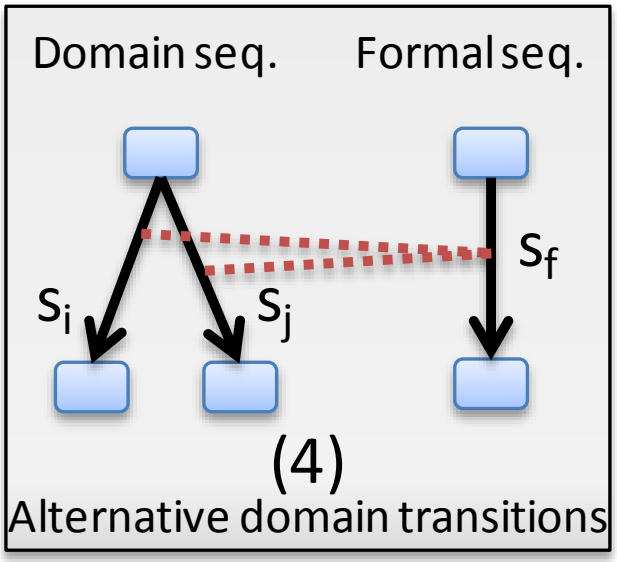


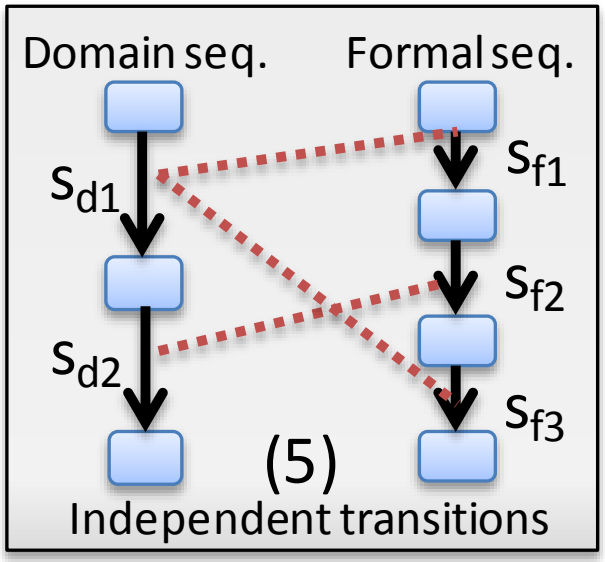
Challenges of Back-annotation

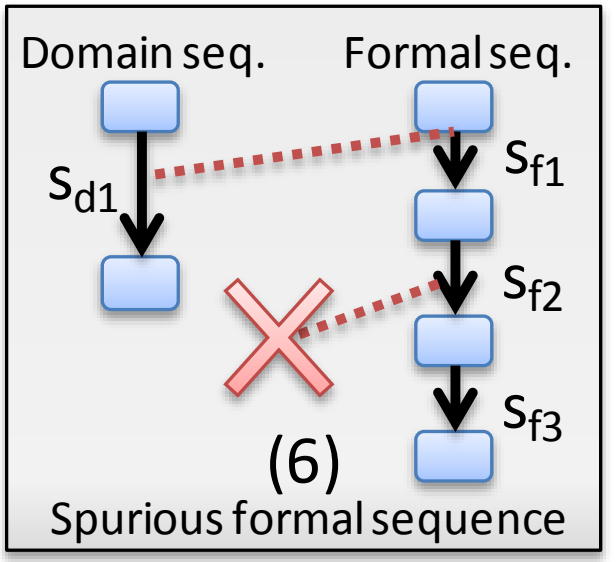




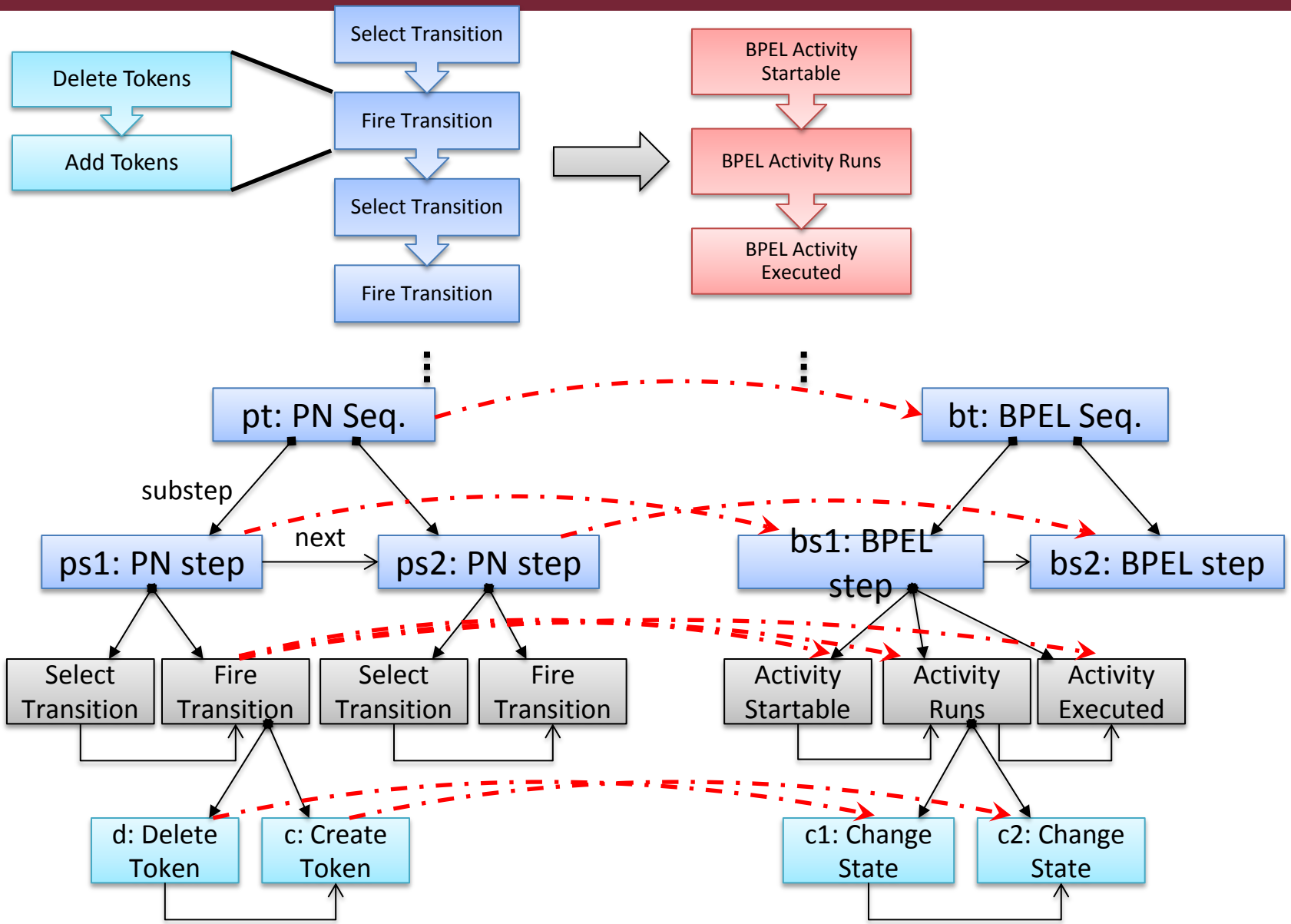




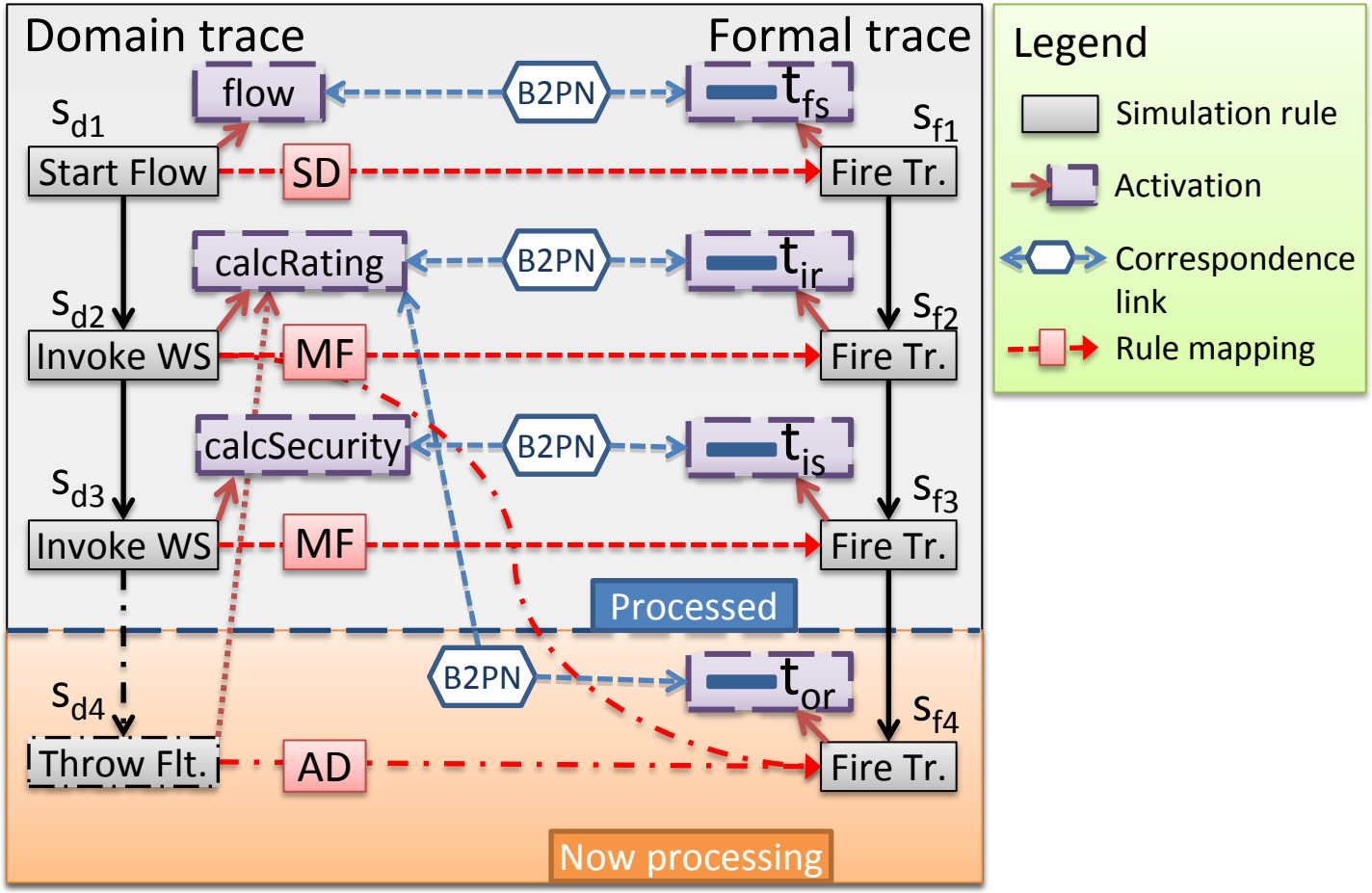




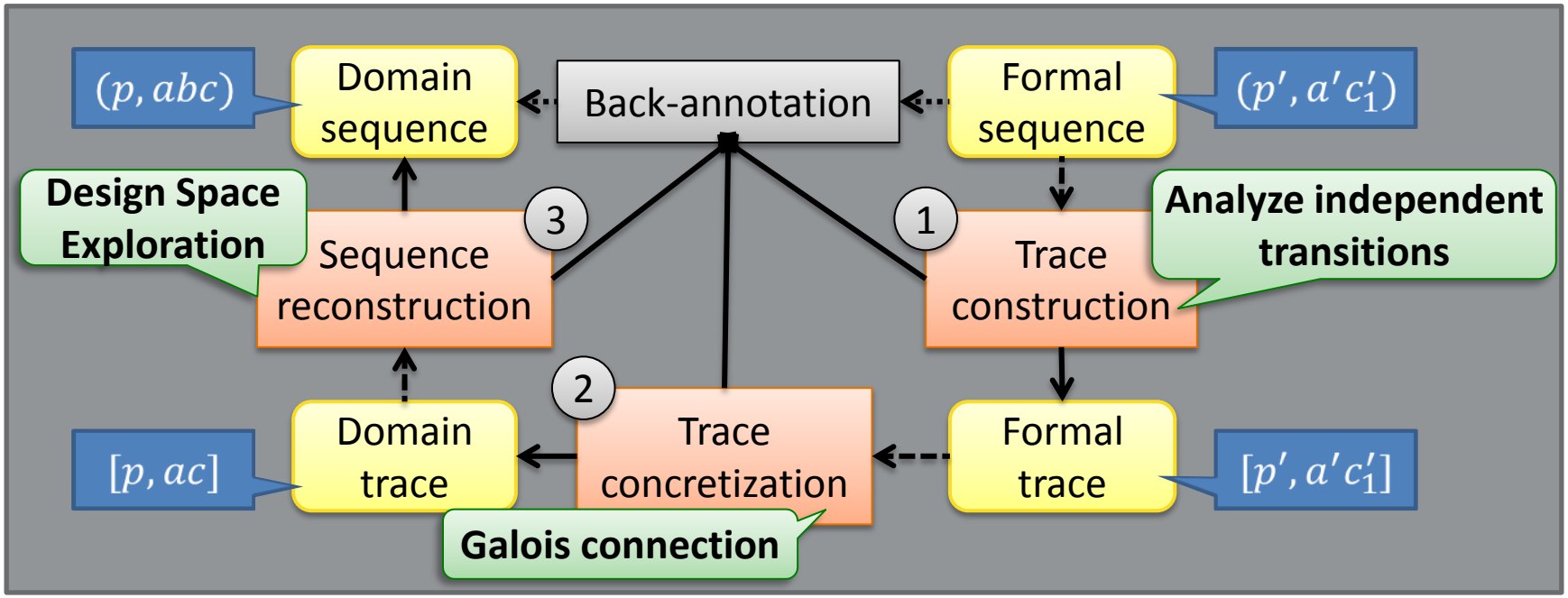
Example: Back-Annotation

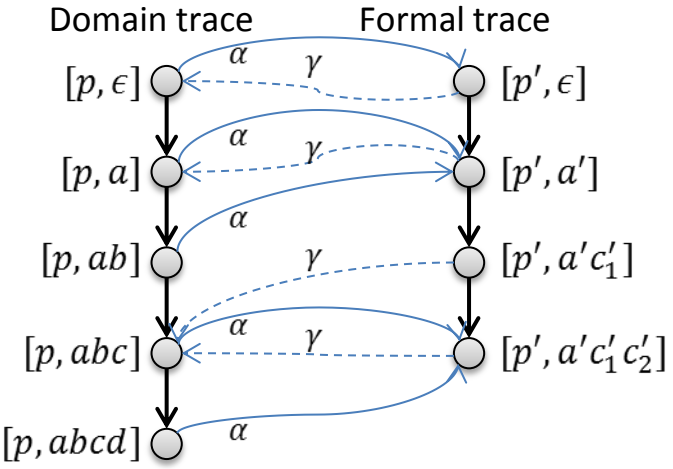


Example: Back-annotation



Our Back-Annotation Approach

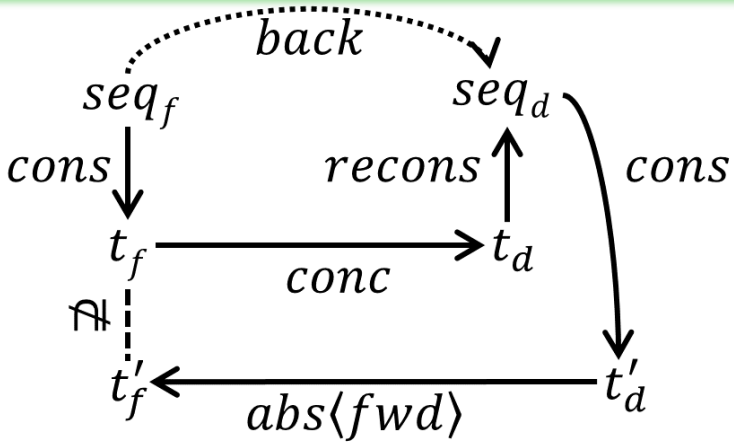




Semantic Properties of Back-Annotation

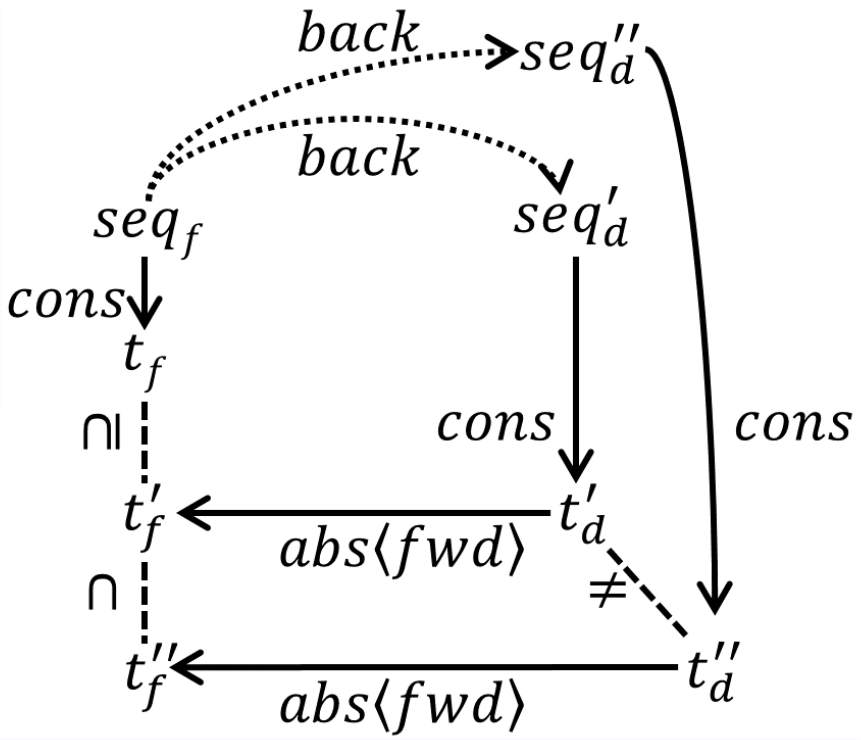
- Correctness

Definition 16. The *correctness of a forward transformation* is defined as a simulation property between domain and formal traces implied by the abstraction function $abs\langle fwd\rangle$. For each (stuttering equivalent) domain trace $t_d = [G_d, w_d]$, there is a *uniquely defined* formal trace $t_f = [G_f, w_f]$ with $t_f = abs\langle fwd\rangle(t_d)$.



Definition 17. A back-annotation $back : seq_f \rightarrow 2^{seq_d}$ is *correct* wrt. a correct abstraction function $abs\langle fwd\rangle$ if for each $seq_f = (G_f, w_f)$ and $seq_d = (G_d, w_d)$ with $seq_d \in back(seq_f)$, then $[G_f, w_f] \subseteq abs\langle fwd\rangle([G_d, w_d])$.

- Minimality



Definition 18. A back-annotation $back : seq_f \rightarrow 2^{seq_d}$ is *minimal* wrt. a correct abstraction function $abs\langle fwd\rangle$ if for each $seq_f = (G_f, w_f)$ and $seq_d = (G_d, w_d)$ with $seq_d \in back(seq_f)$, there does not exist another domain sequence $seq'_d = (G_d, w'_d)$ with $seq'_d \in back(seq_f)$ such that $[G_f, w_f] \subseteq abs\langle fwd\rangle([G_d, w'_d]) \subset abs\langle fwd\rangle([G_d, w_d])$.