Model Transformation and Graph Transformation

Horváth Ákos

Dániel Varró

Budapesti Műszaki és Gazdaságtudományi Egyetem
Méréstechnika és Információs Rendszerek Tanszék

Development Process for Critical Systems

**Unique Development Process
(Traditional V-Model)**

Critical Systems Design

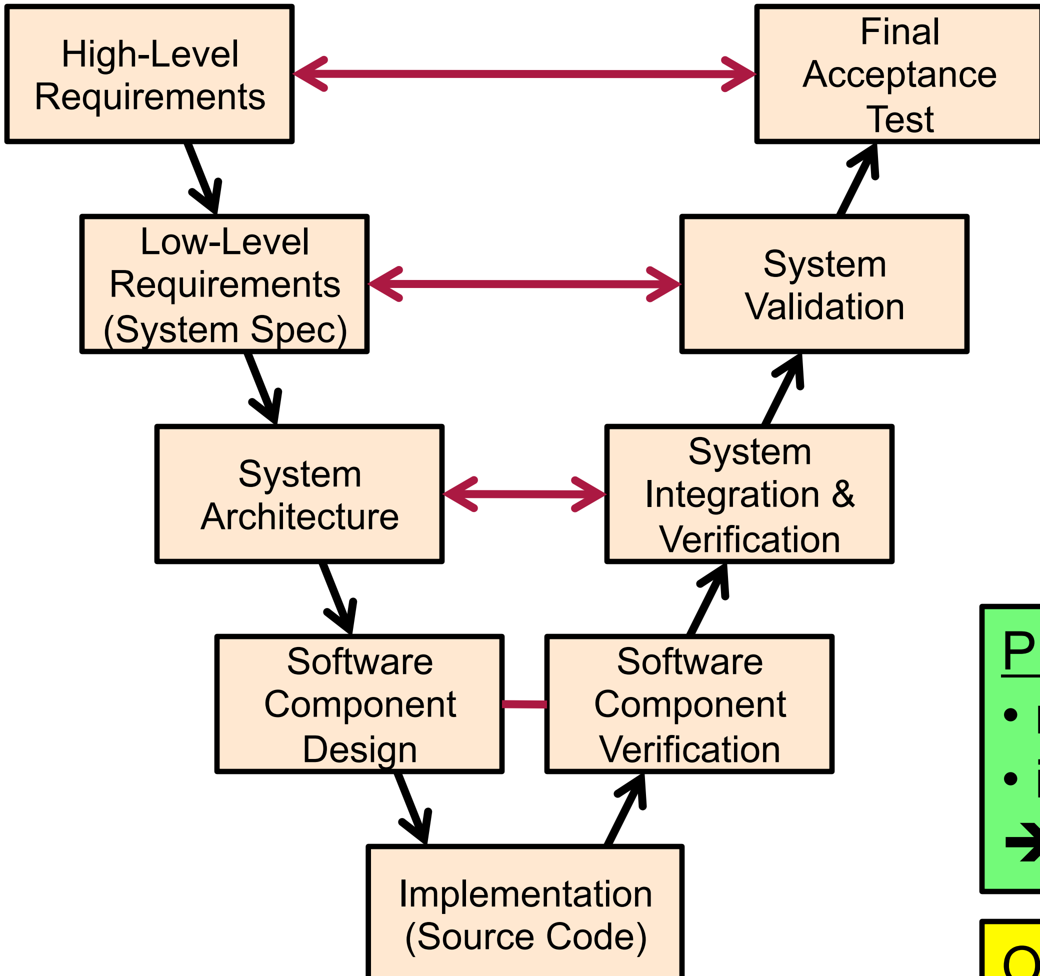
- requires a certification process
- to develop justified evidence
- that the system is free of flaws

Software Tool Qualification

- obtain certification credit
- for a software tool
- used in critical system design

Qualified Tool → Certified Output

Qualification of Software Tools



Development tools:
- input ➔ output deterministically
- introduce new errors

Verification tools:
- fail to detect errors

Promises of Tool Qualification
- reduce development + V&V cost
- increase quality and productivity

➔ reduce certification costs

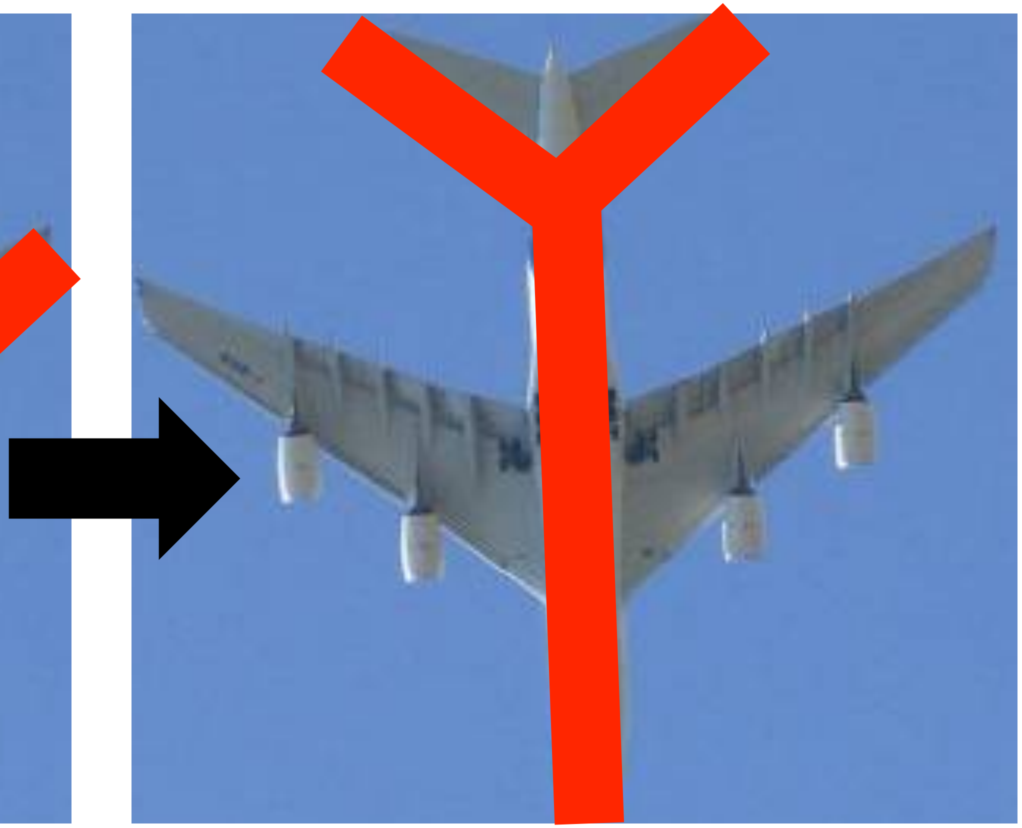
Obstacles for Tool Qualification
- hidden tool functionality
- complex V&V tasks

➔ extreme qualification costs

Model-Driven Engineering of Critical Systems

Traditional V-Model

Model-Driven Engineering

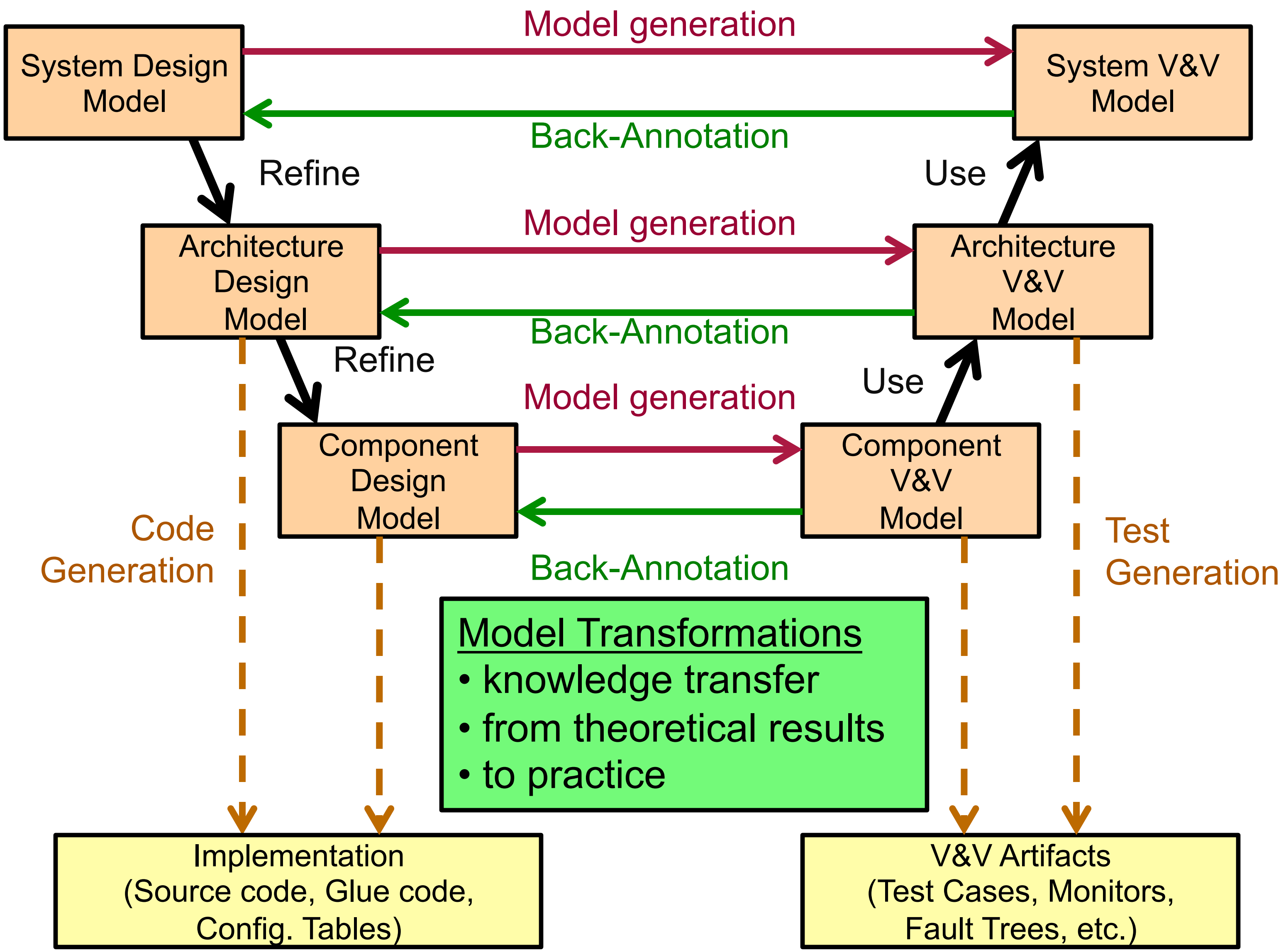


<u>Main ideas of MDE</u>

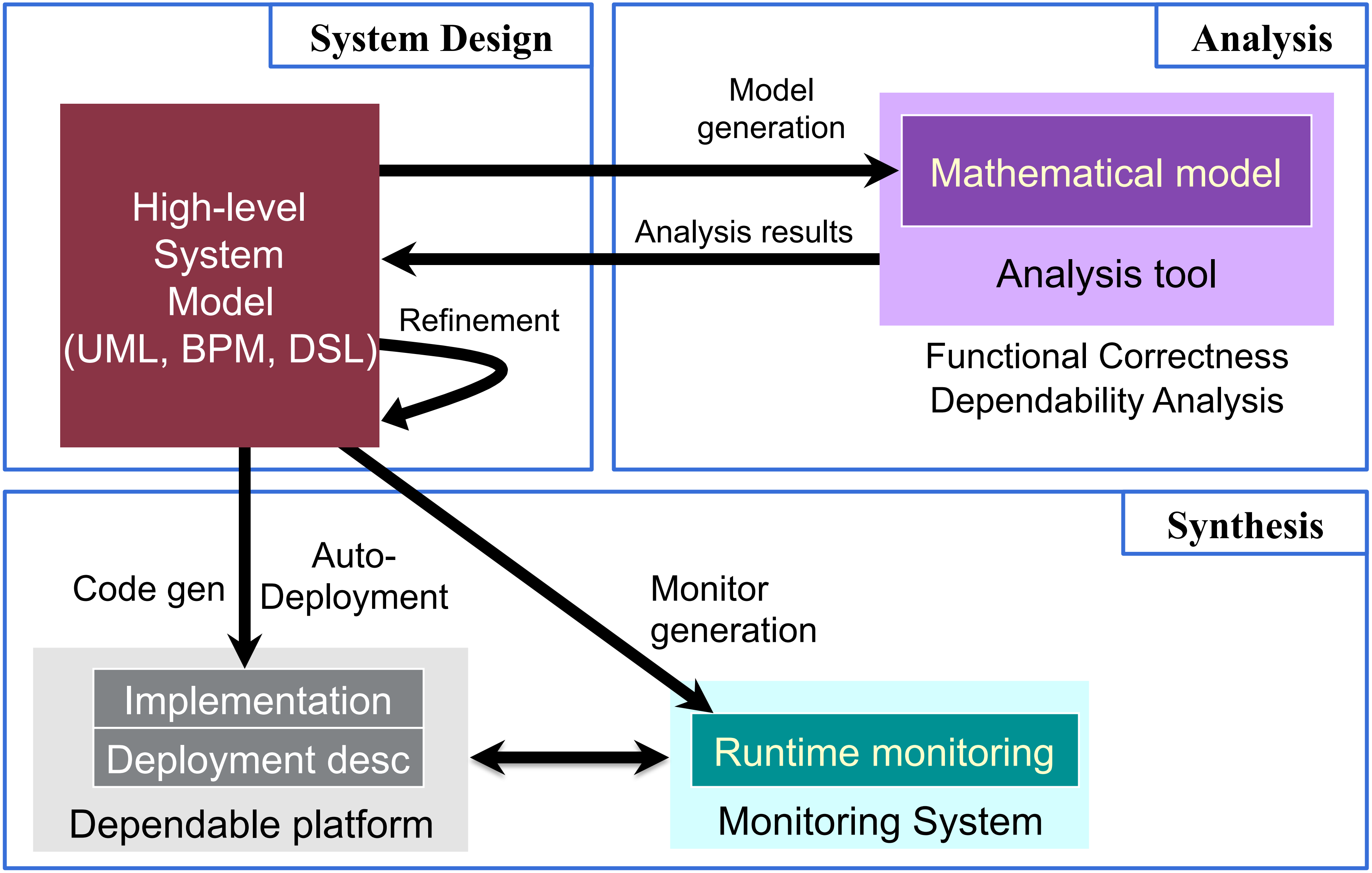
- early validation of system models
- automatic source code generation

➔ reduce development costs

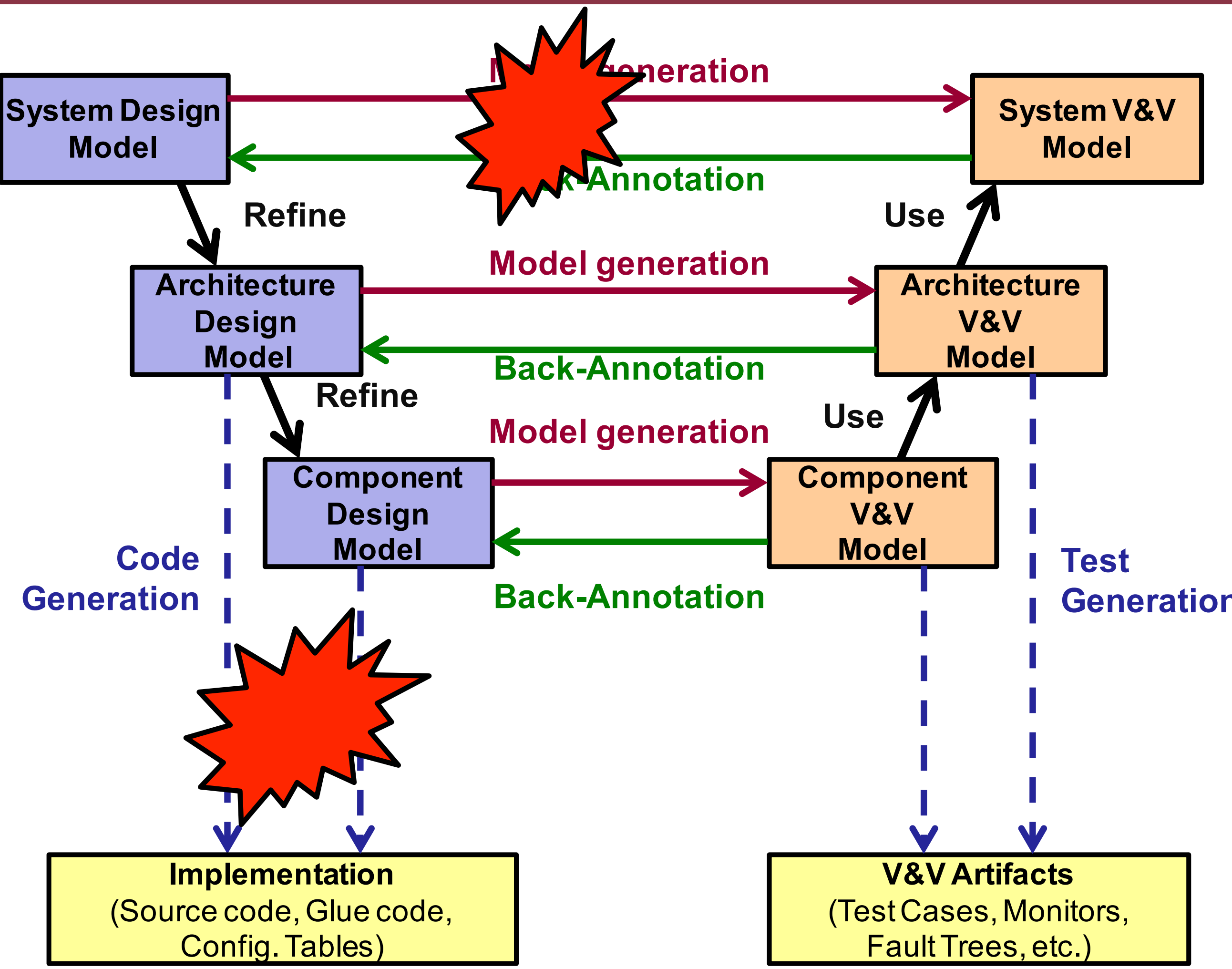
Models and Transformations in Critical Systems



Model-Driven Design and Analysis



Model Transformation Errors?



Code generator error
- model: OK, code: no

Model generator error
- model: OK, V&V: No
- model: No, V&V: OK

Agenda

- Model Transformation in general
- MT Approaches
- Graph Transformation
  - Example
  - Definition
  - Pattern matching
- Graph Transformation Systems

Model Transformation

MDA promises

- Improving quality
  - Designers can focus on a narrower area of expertise
  - Automation ⇒ fewer bugs
- Improving productivity
  - Modelling time increased
  - Integration, testing times greatly reduced
  - Large parts of code can be generated automatically
- Compatibility, reusability among platforms
  - Model level (PIM) > component level

Challenges in Model Based Engineering

- A typical design process of a large system involves
  - Many stakeholders, development teams, man months
  - Many tools:
    - Requirements, Analysis, Design, Testing, Maintenance, …
- Tool integration is a major challenge
  - Design of Embedded / Critical systems:
    Cost of tool integration ≈ Cost of the tools themselves
- Why?
  - Continuous evolution / changes of tools
  - Each having its own (modelling / programming) language
  - Difficult to build correct and robust bridges between them

Applications of Model Transformation

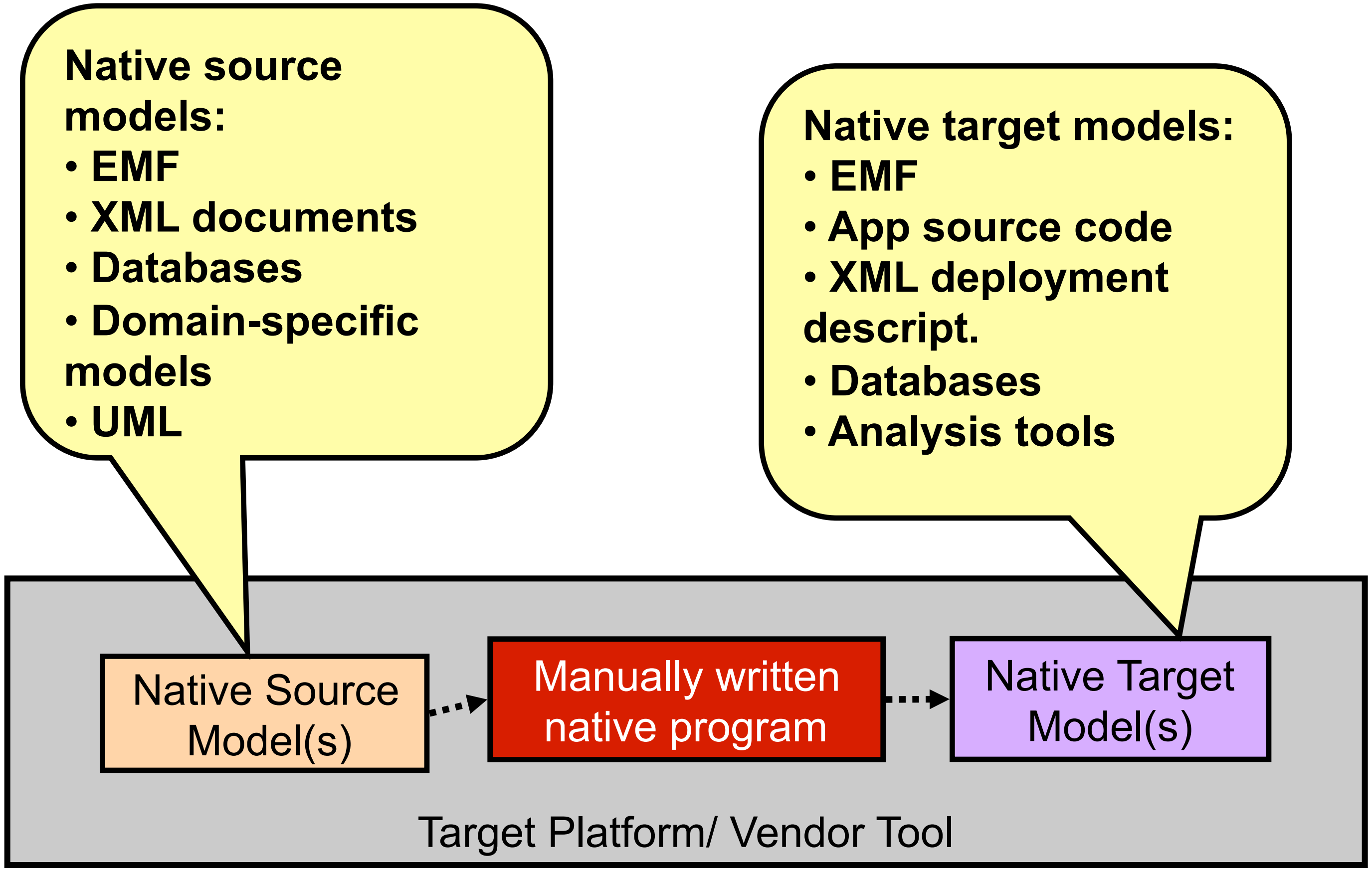
- Application of MT in MDE
  - Automation of development processes
    - Configuration generation for deployment
    - Design optimization
  - Model based verification and analysis
    - High level model → mathematical formalism → verify
    - Constraint checking over models
  - Tool integration
    - Merge, aggregate, map models of industrial tools
  - Domain Specific Modeling support
    - Integrated MT in modeling tools
    - Simulation of DSM languages
- **Needs to support all aspects** → common requirements
  - Easy definition → declarative
  - Usually, graph based models
  - Pattern matching
  - Elementary model manipulation

MT: categories

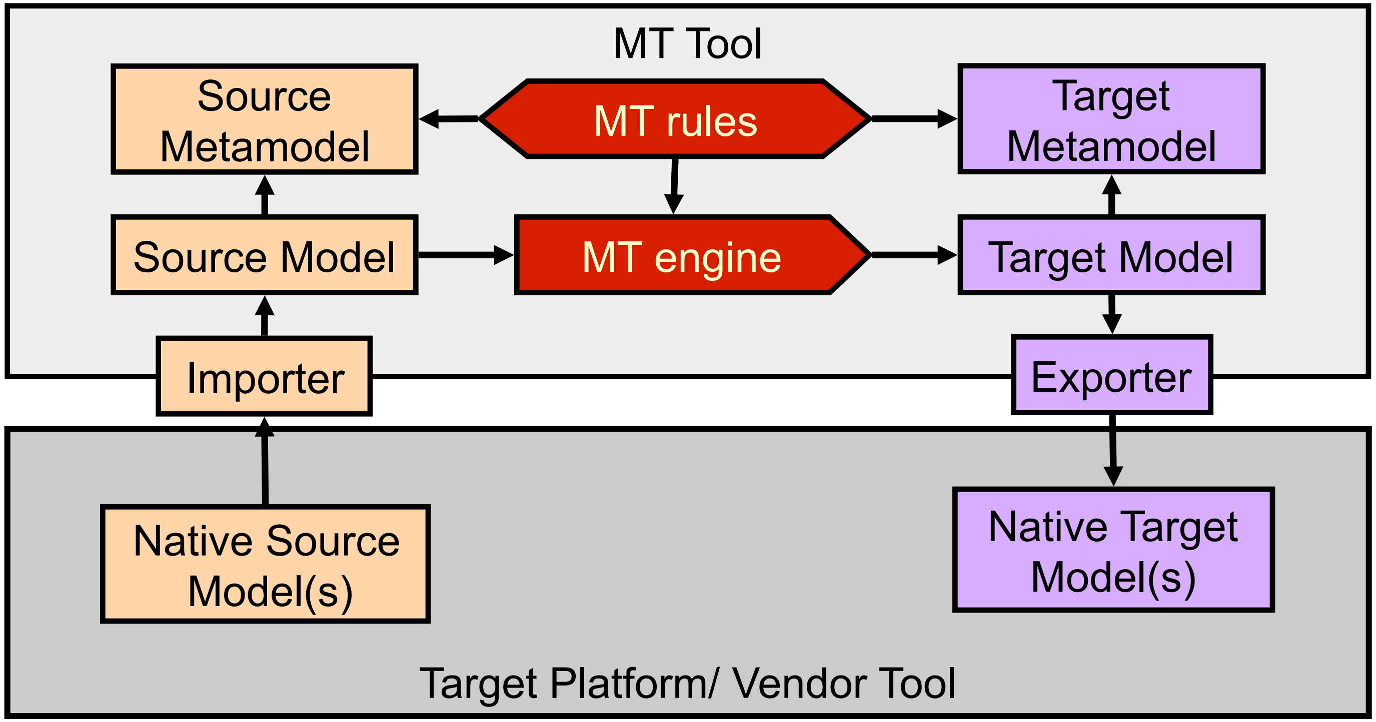
- Model-to-Code (M2C)
  - Text generation
  - AST generation → special case of M2M
  - Ad-hoc, dedicated, template based, etc.
- Model-to-Model (M2M)
  - Between models
    - Intra-domain transformation
      (e.g., simulation, refactoring, validation)
    - Inter-domain transformation
      (PIM-to-PSM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

Model Transformation Architecture

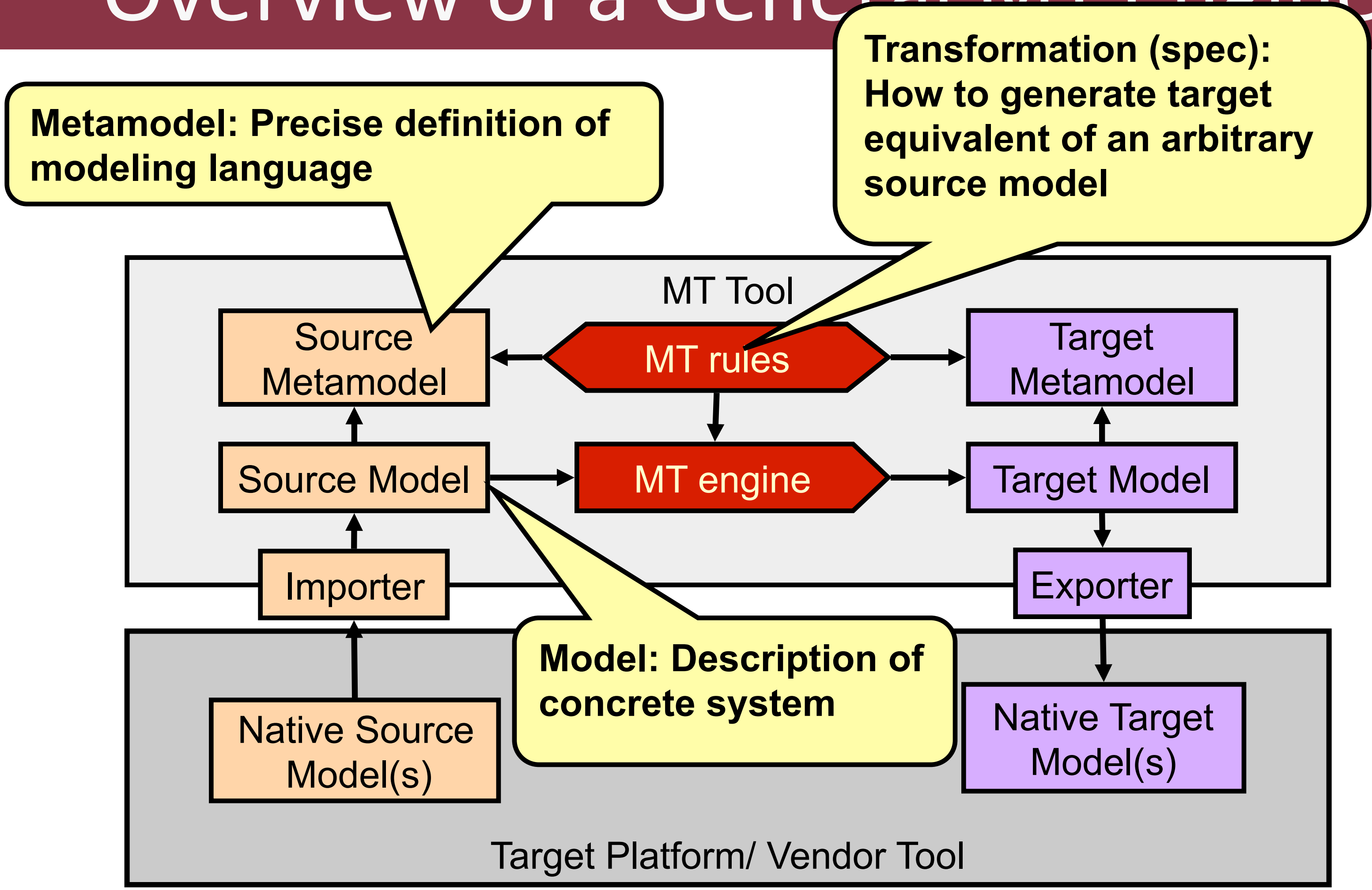
Add-hoc transformation



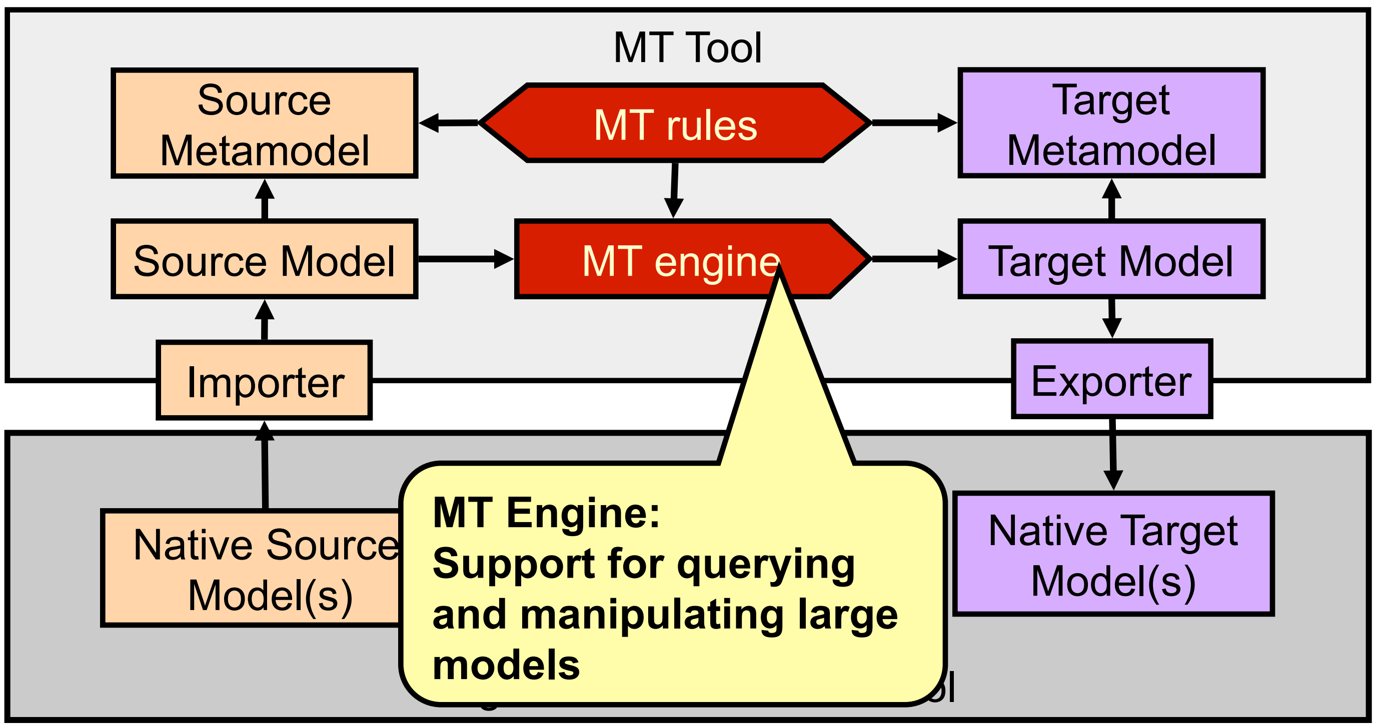
Overview of a General MT engine



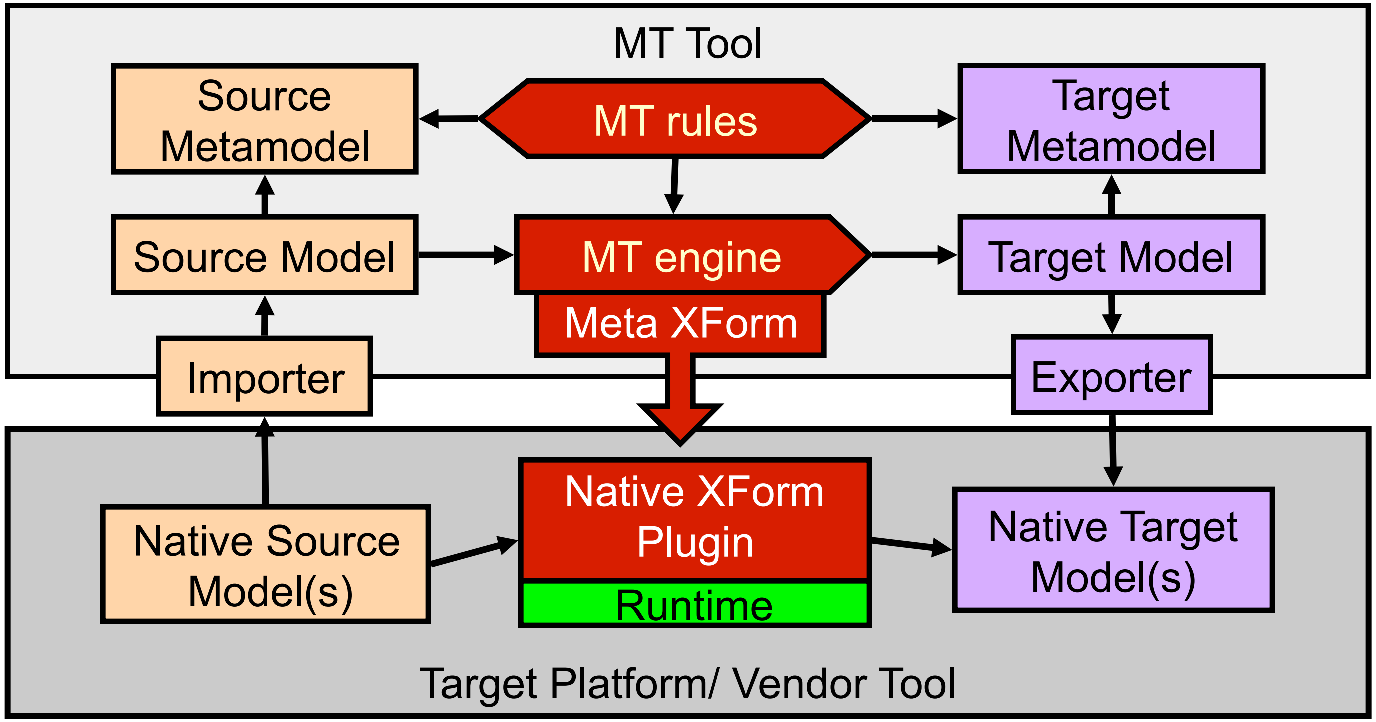
Overview of a General MT engine



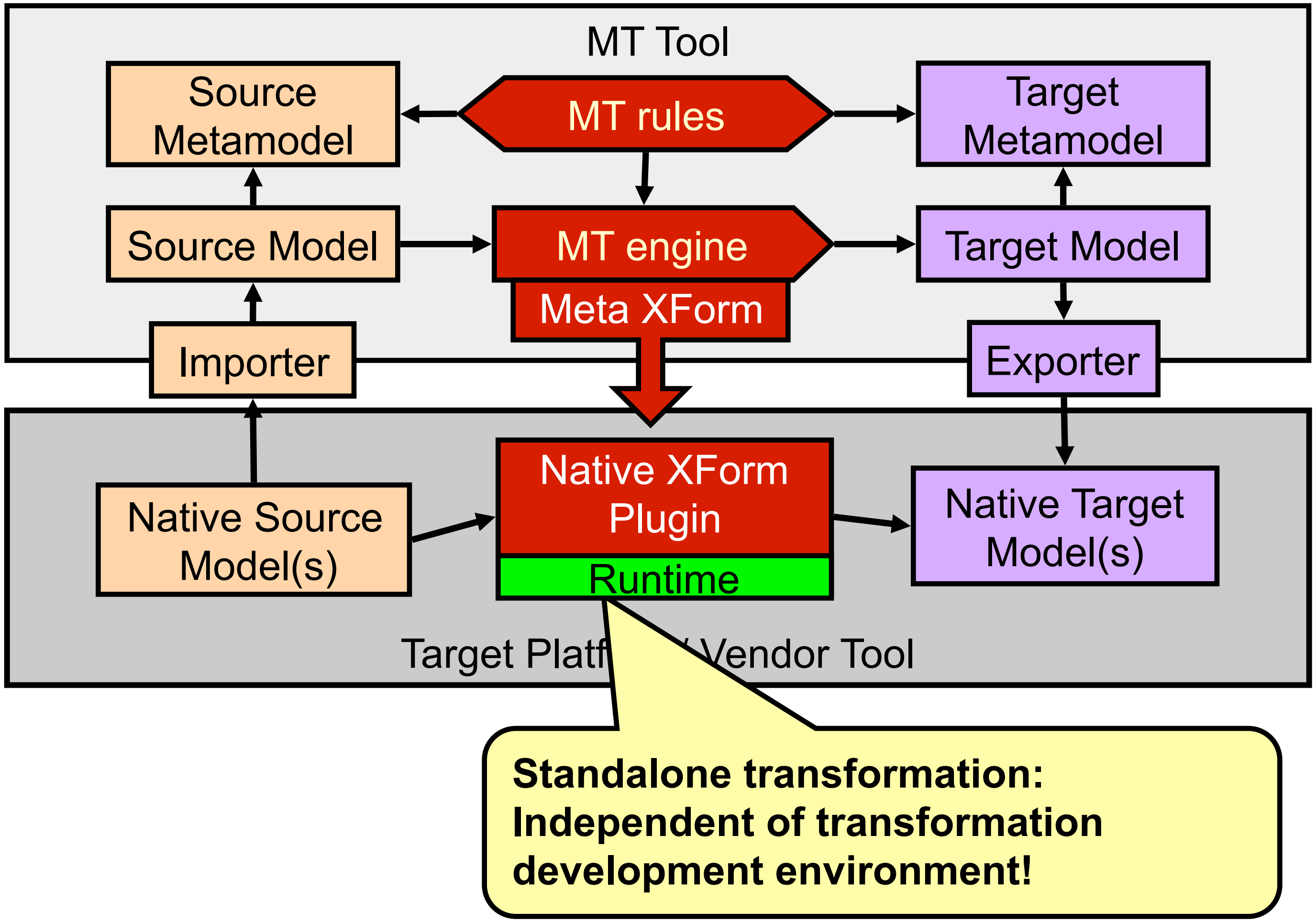
Overview of a General MT engine



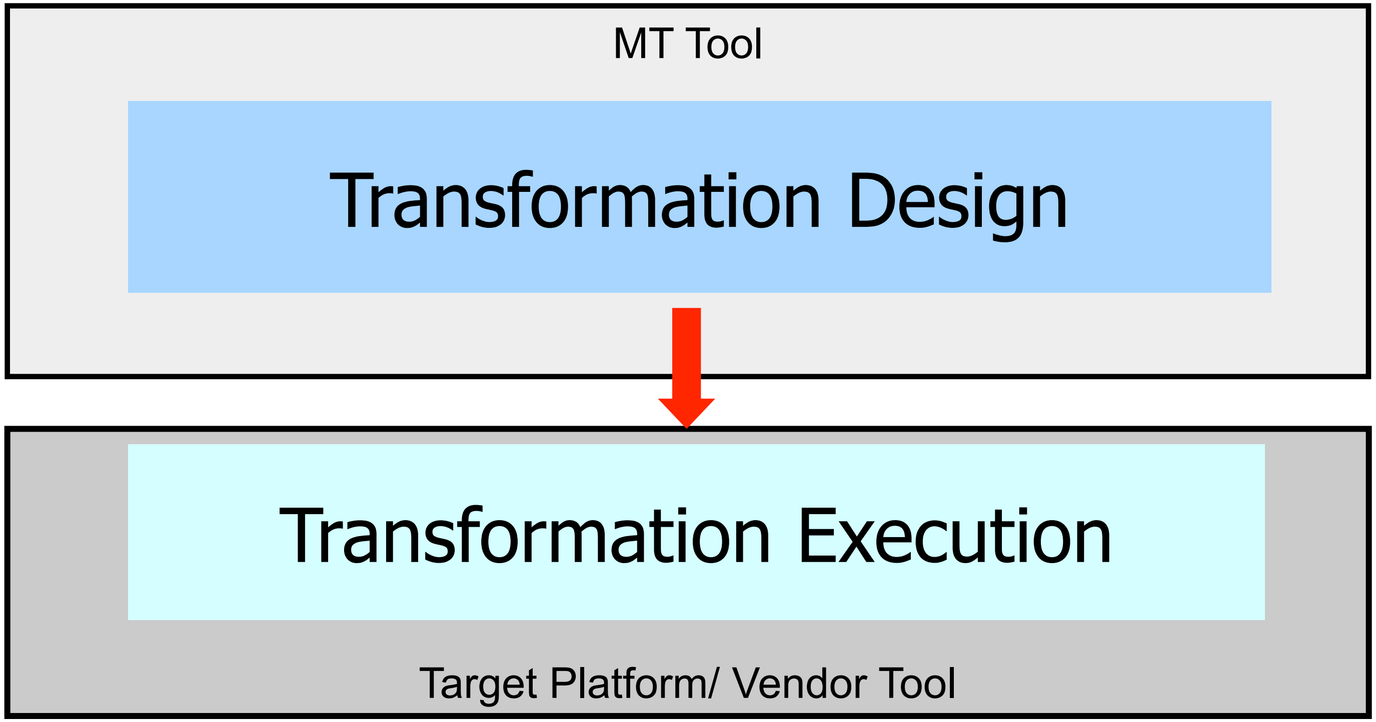
Overview of a General MT engine



Overview of a General MT engine



Overview of a General MT engine



Model Transformation Approaches

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

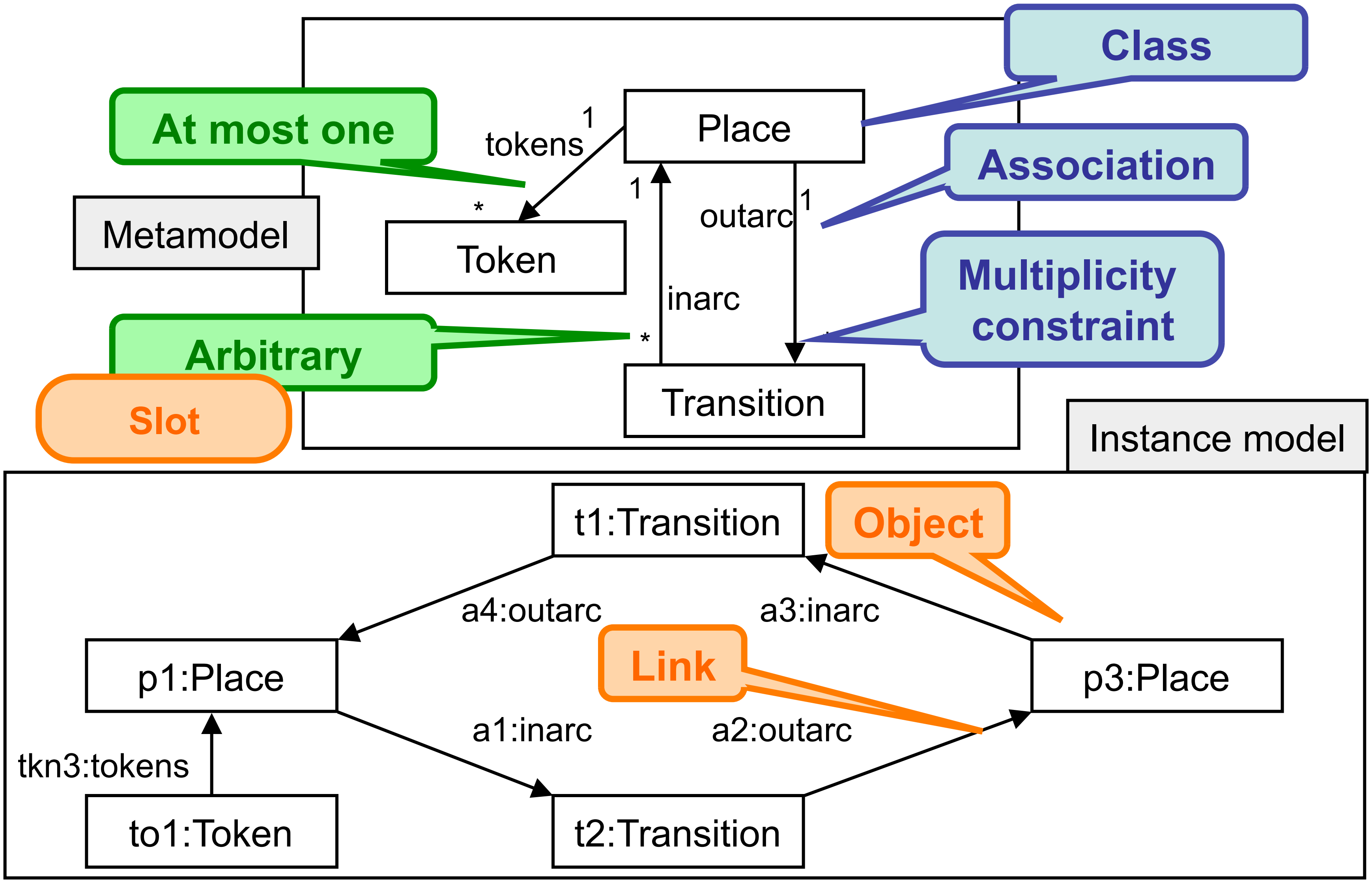
- Models as XMI files
- Model Transformation as XSLT programs
- Hard to maintain
- XMI representations are
  - verbose
  - poor readability

Graph Transformation

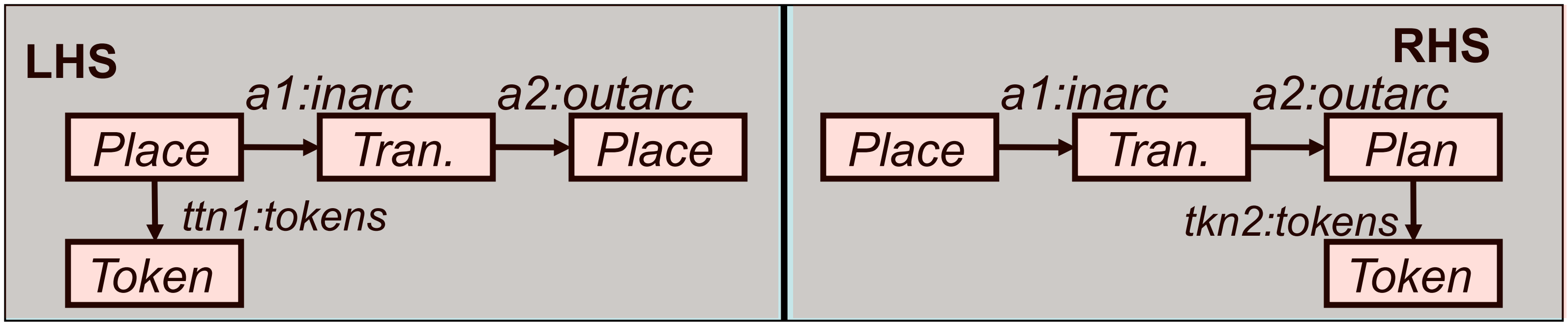
Graph Transformation

- Graph Transformation (GT)
  - A mathematical behaviour specification formalism behind model transformation (and much more)
  - Solid theoretical background, yet easy to explain
- Key concept: graph pattern
- Transition: graph transformation rule
  - Left hand side (LHS, precondition) – graph pattern
  - Right hand side (RHS, postcondition) – graph pattern
  - Application: replace one with the other!

Example: Petri net simulation



Example: Petri net simulation



Phases of GT matching

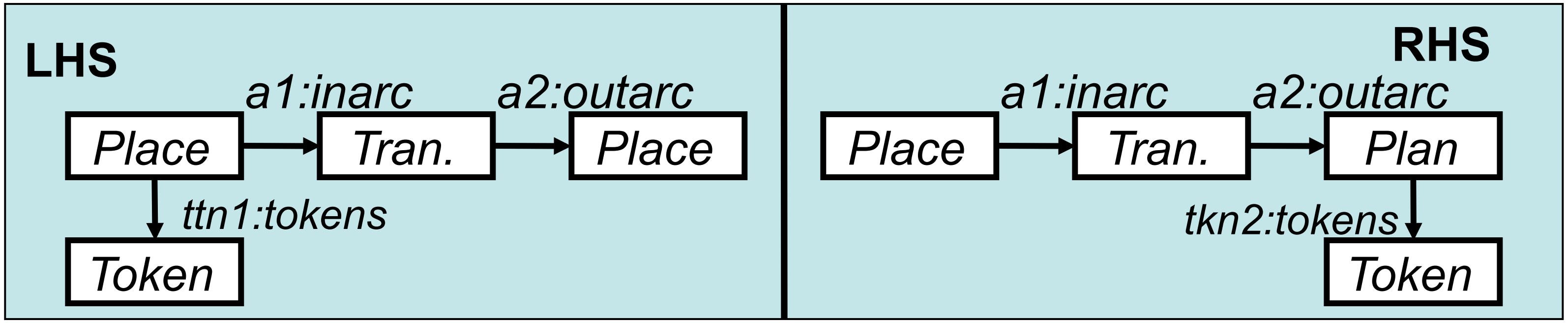
- Pattern Matching phase (non-determism)
- Updating phase: delete+ create

Definition of Graph Transformation rules

A GT rule is defined by

- Left hand side (LHS) pattern
  - Defines the precondition of the rule application
- Right hand side (RHS) pattern
  - Defines the postcondition of the rule

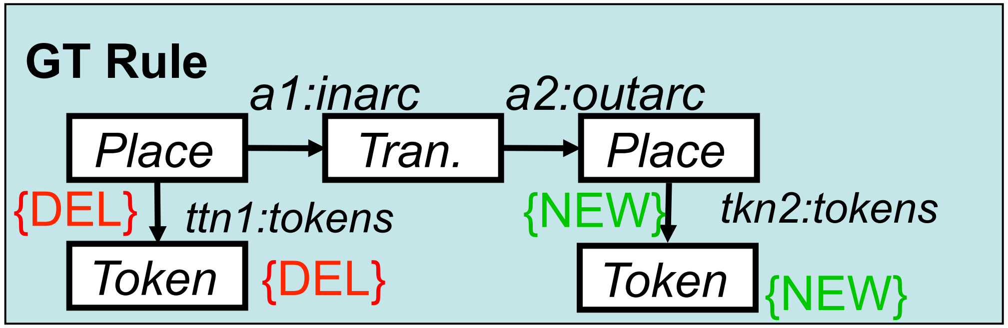
- **Semantics:** select rule p : LHS → RHS ;
  - occurrence o_L : LHS → G (instance graph) [pattern matching]
  - remove from G the occurrence of L \ R [update: delete]
  - add to result an occurrence of R \ L [update: create]



Definitions of GT Rules

GT rule P: LHS → RHS with LHS∩RHS well-defined,

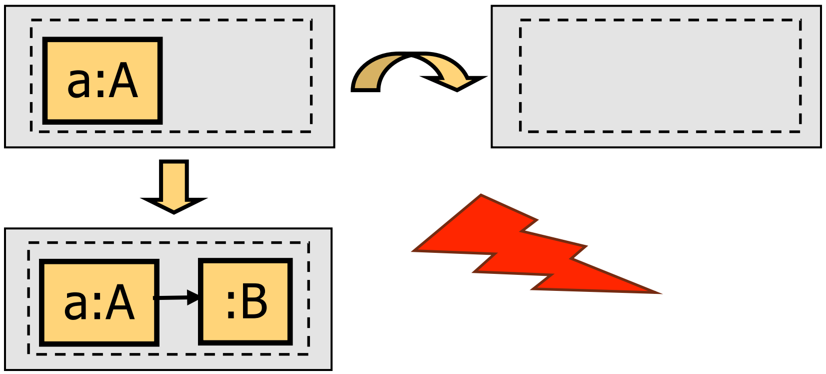
- different presentations
  - With explicit LHS and RHS representation
  - with L, R integrated [Fujaba]: LHS ∪ RHS and marking
    - L - R {del}
    - R - L {new}



Dangling edge options

- Double Push Out (DPO) approach (conservative):
  - don‘t delete if this causes „dangling edges“ → invertible transformations, no side-effects

- Example
  - Delete A node (horizontal)
  - Delete B node (vertical)

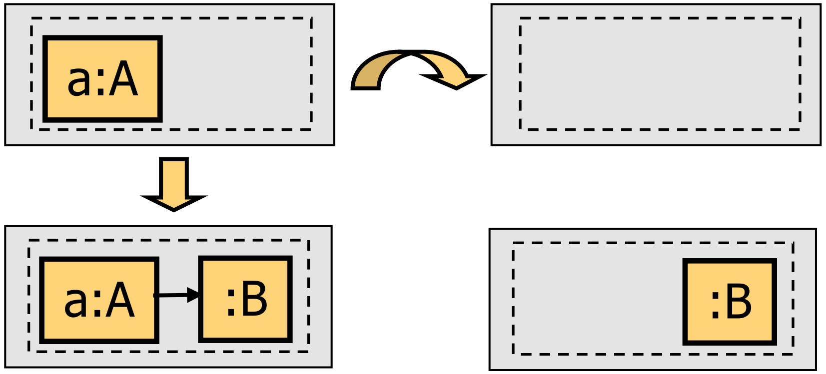


Dangling edge options

- Single Push Out (SPO) approach (greedy):
  - delete also the dangling edges

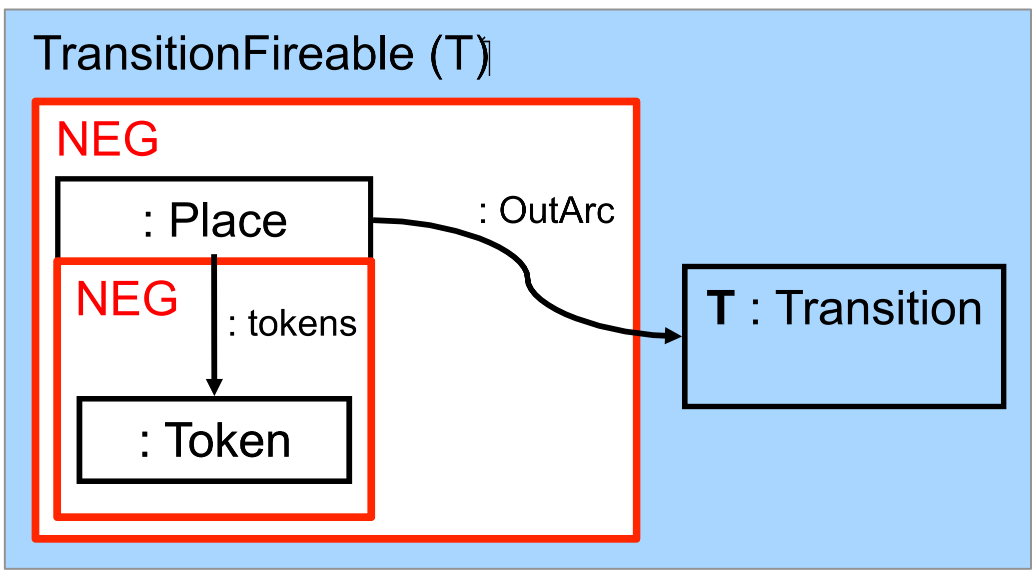
  → side-effects, BUT more intuitive + easier to implement

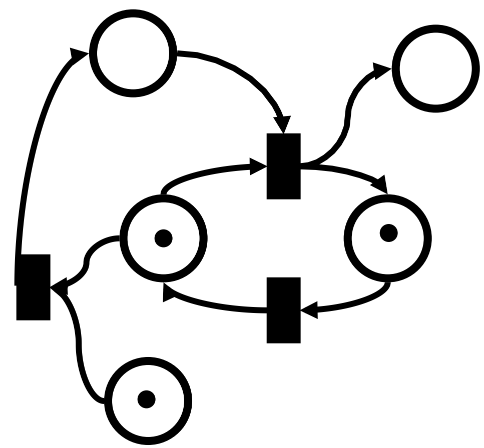
- Example
  - Delete A node (horizontal)
  - Delete B node (vertical)



Extension – Negative Application Condition

- Negative Application Condition
  - Also a graph pattern
  - If the NAC can be successfully matched then the application of the rule fails.
  - Example ☺ (NAC patterns can be embedded into each other)





Uses of GT

- **Model Transformation**
- Behavioural semantics of dynamic languages
  - see „Domain Specific Modeling” lecture
- Graph grammars
  - ~string grammars
  - context free, etc.
  - For defining, parsing graph languages
- Graph transformation systems
  - Formal behavior definition
  - For Model Checking

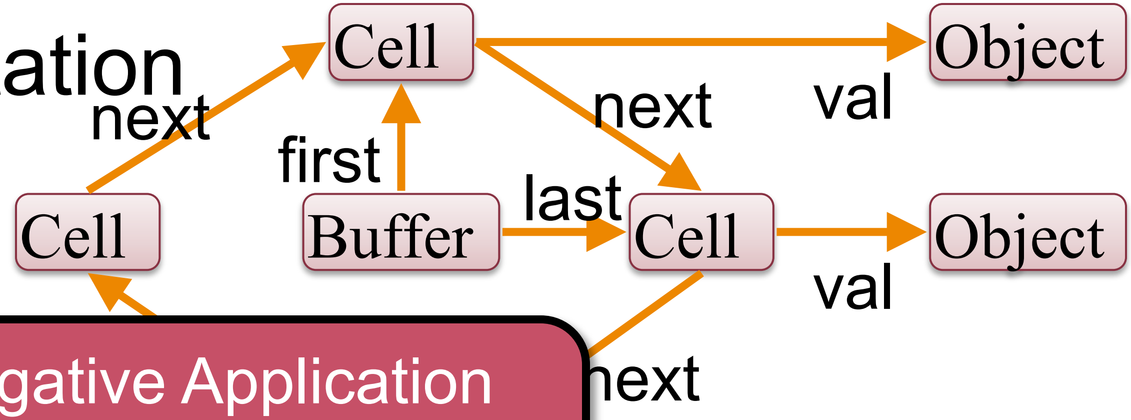
Graph Transformation System

GTS

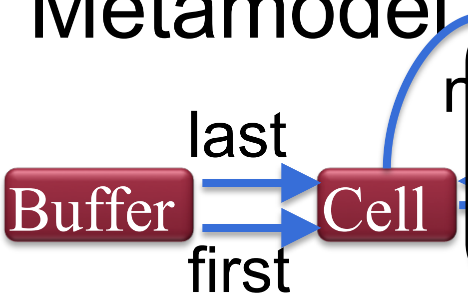
- GTS (Graph Transformation System)
  - $GTS = \langle G_0, \mathcal{P} \rangle$
  - $\mathcal{P}$: finite set of production operations
  - States: reachable from G_0 graph
  - Transitions: GT rule applications
- Transition Labeling: applied rule (→LTS)
- State Labeling: graph patterns (→~Kripke)
- When to use?
  - Algorithms on complex data structures
  - Concurrent, asynchronous systems

FIFO Circular Buffer

- Graph representation

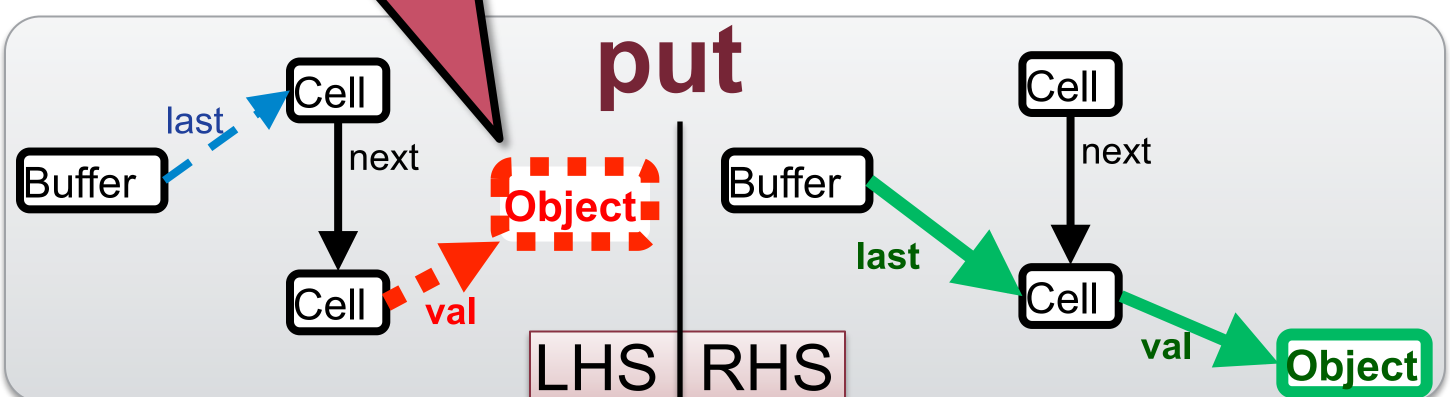


- Metamodel



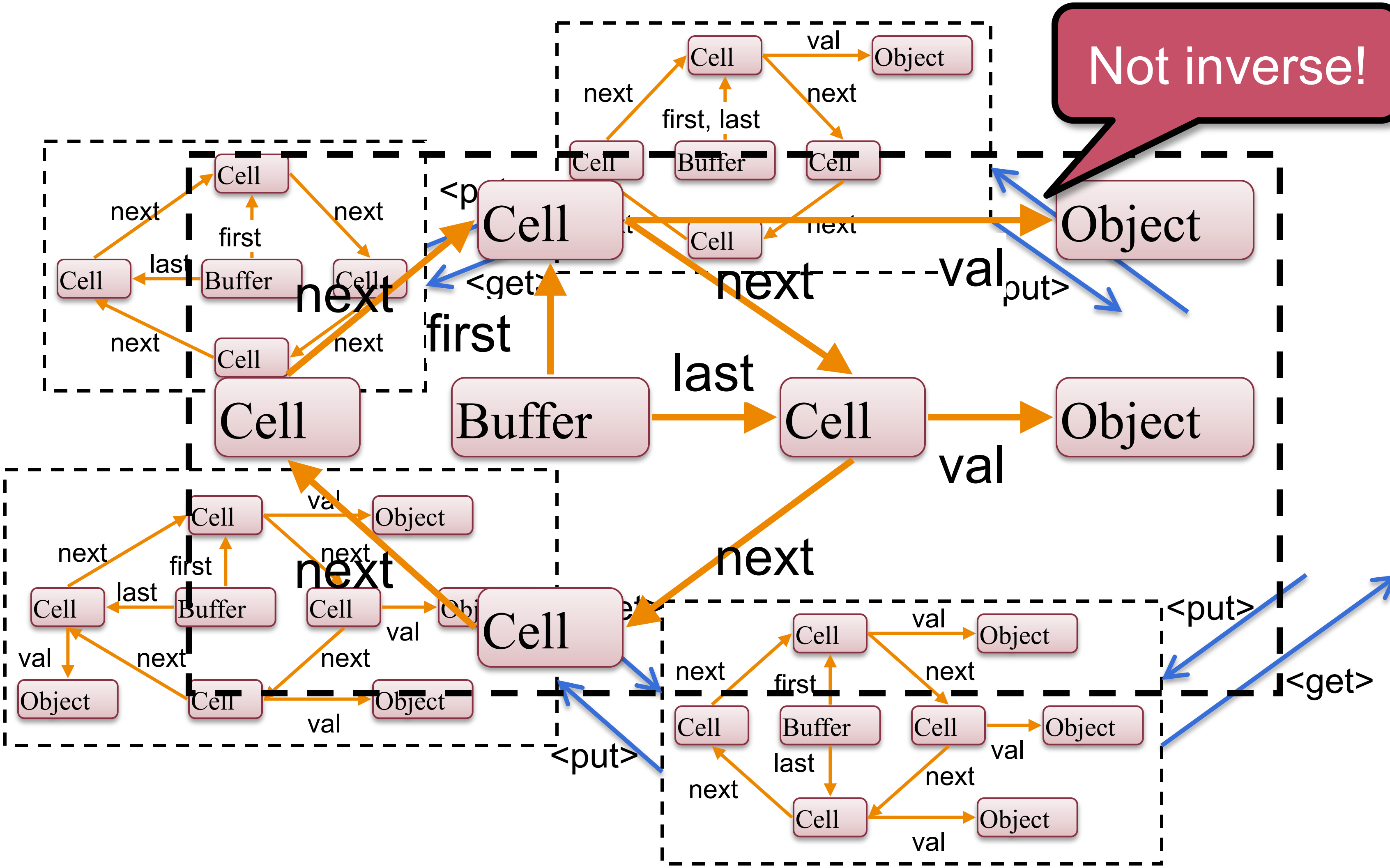
Negative Application Condition (NAC)

- Operation as GT rules:
  - get
  - **put**



Source: Lecture of Arend Rensink, 2005.03.04, NVTI day, Utrecht

State Space



Source: Lecture of Arend Rensink, 2005.03.04, NVTI day, Utrecht

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

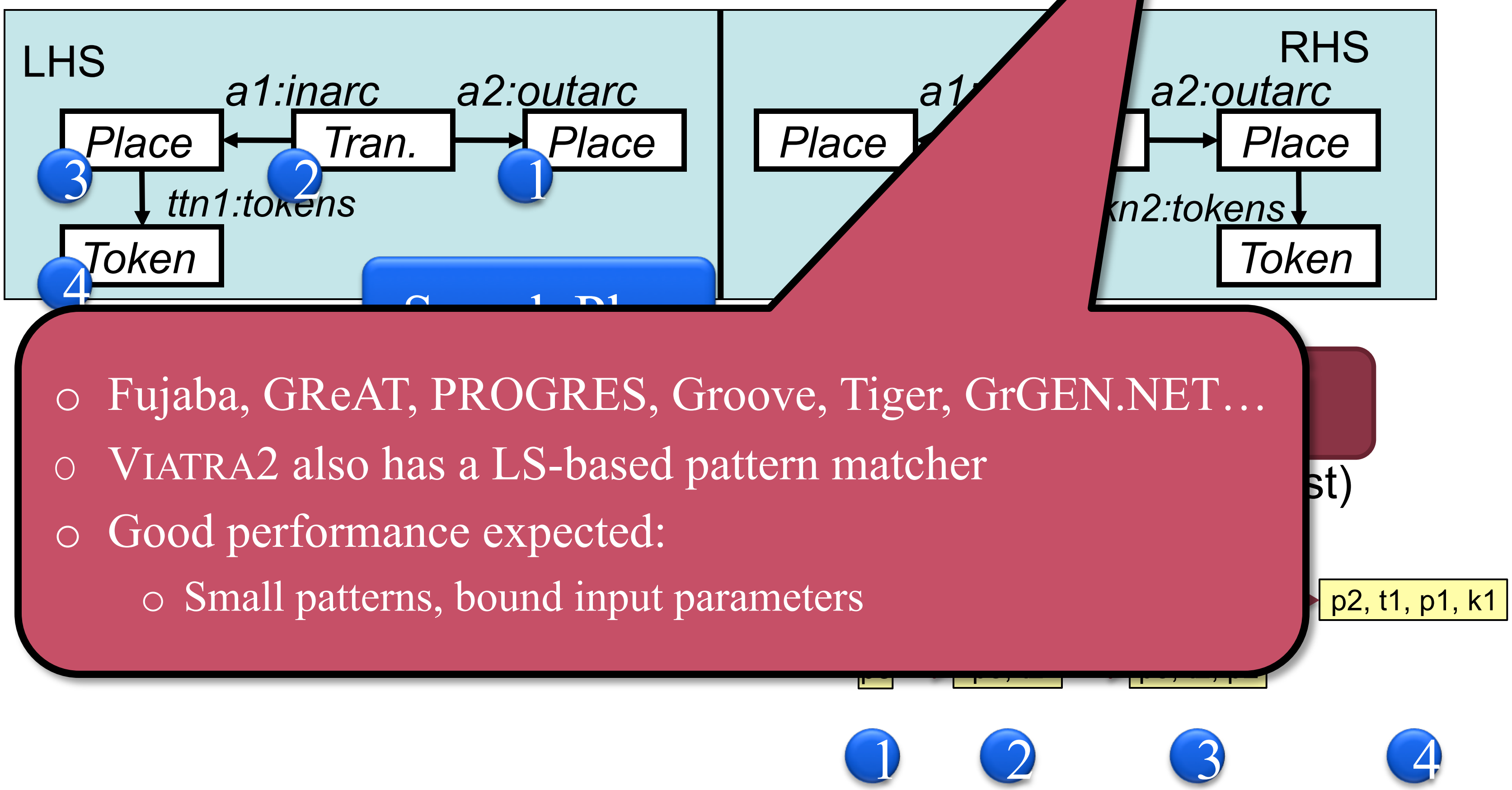
- Realization:
  - Nodes are handled as CSP variables
  - Constraints derived from edges
  - Type information as domain reduction
  - Traversal: backtracking algorithm
- Pros:
  - Adaptive algorithm
- Contras:
  - Handling large models
  - Scalability

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

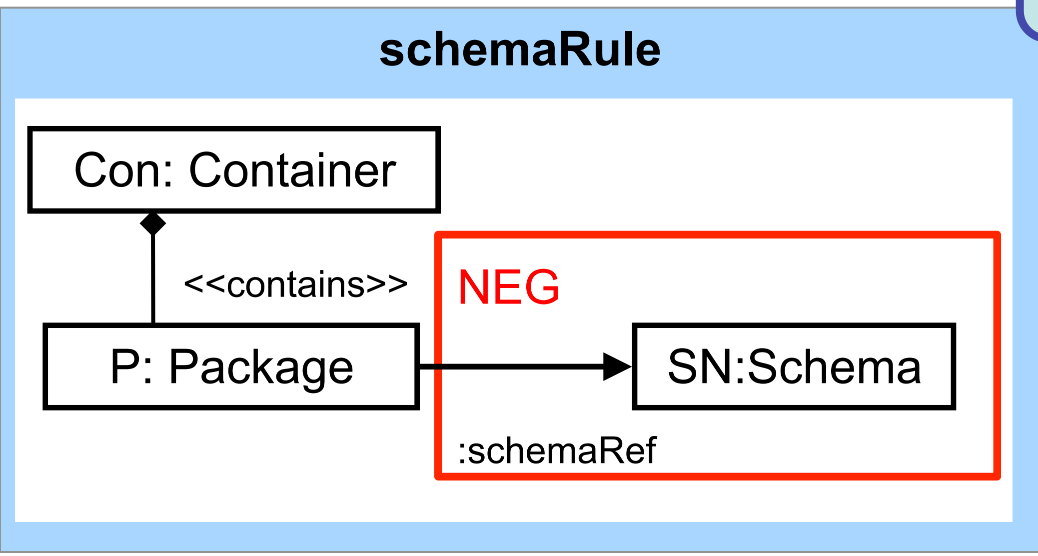


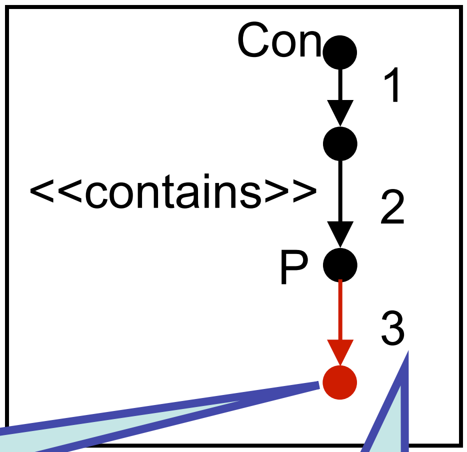
- Fujaba, GReAT, PROGRES, Groove, Tiger, GrGEN.NET…
- VIATRA2 also has a LS-based pattern matcher
- Good performance expected:
  - Small patterns, bound input parameters

Local Search based Pattern Matching Example

frequently used & efficient solution

Search plan





Search sequence

Constraint checking (NAC, injectivity, etc.)

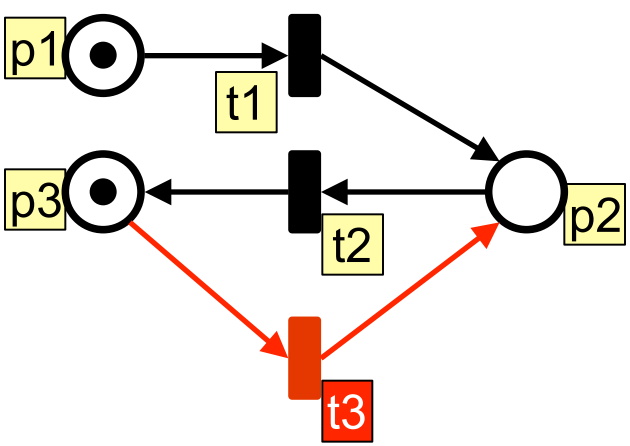
order of traversal in the search plan

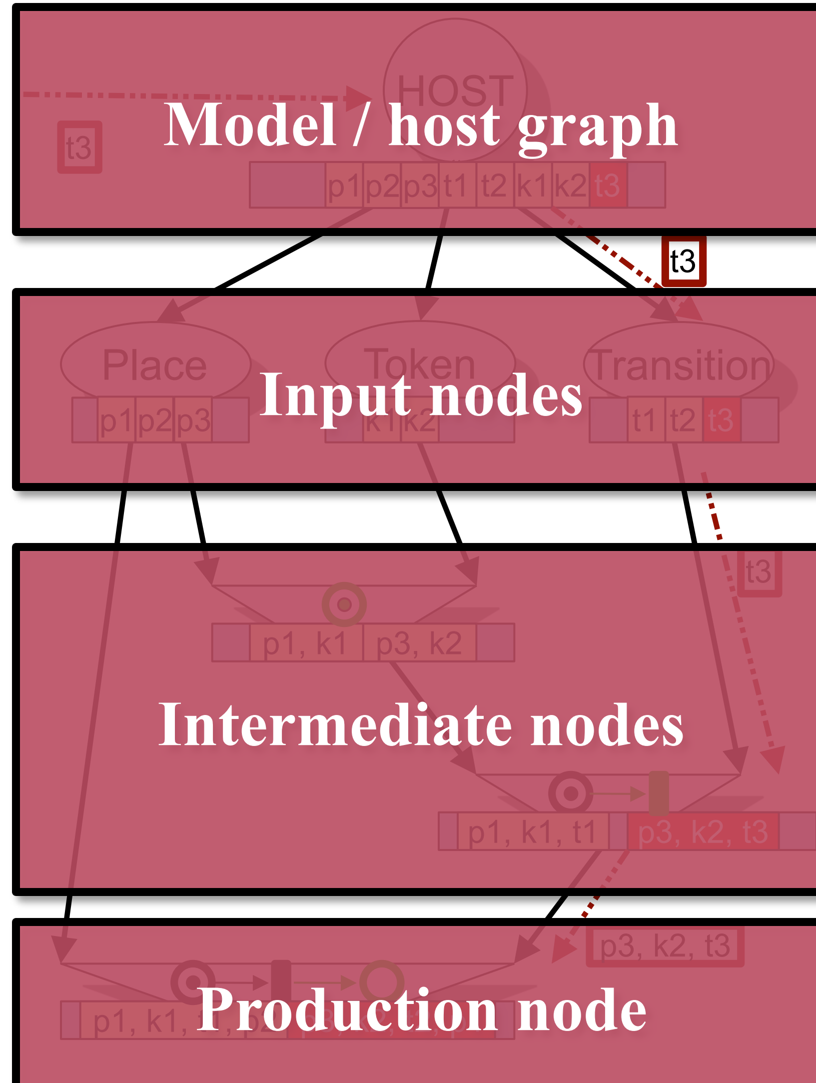
Incremental Pattern Matching

- 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

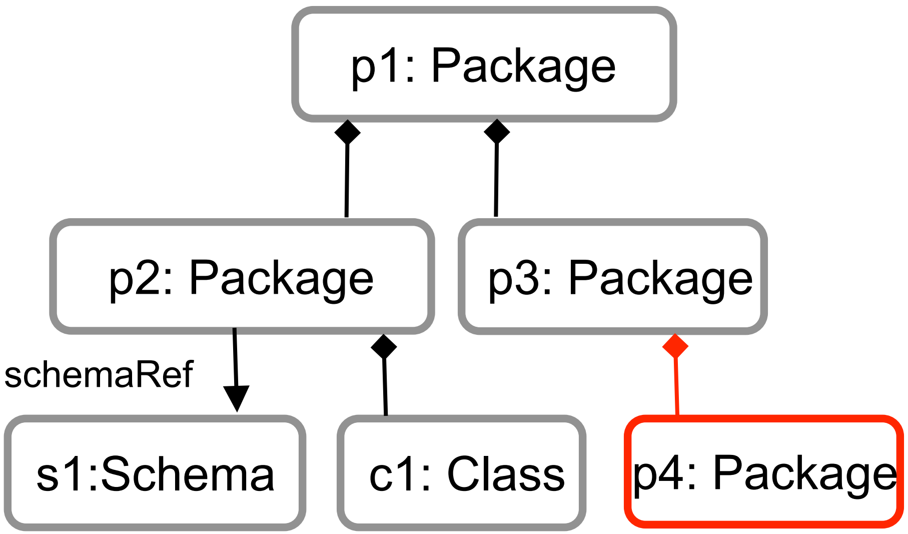
- RETE net
  - node: (sub)pattern
  - edge: change propagation
- Demostrating the principle
  - input: Petri-net
  - pattern: fireable transition
  - change: new transition

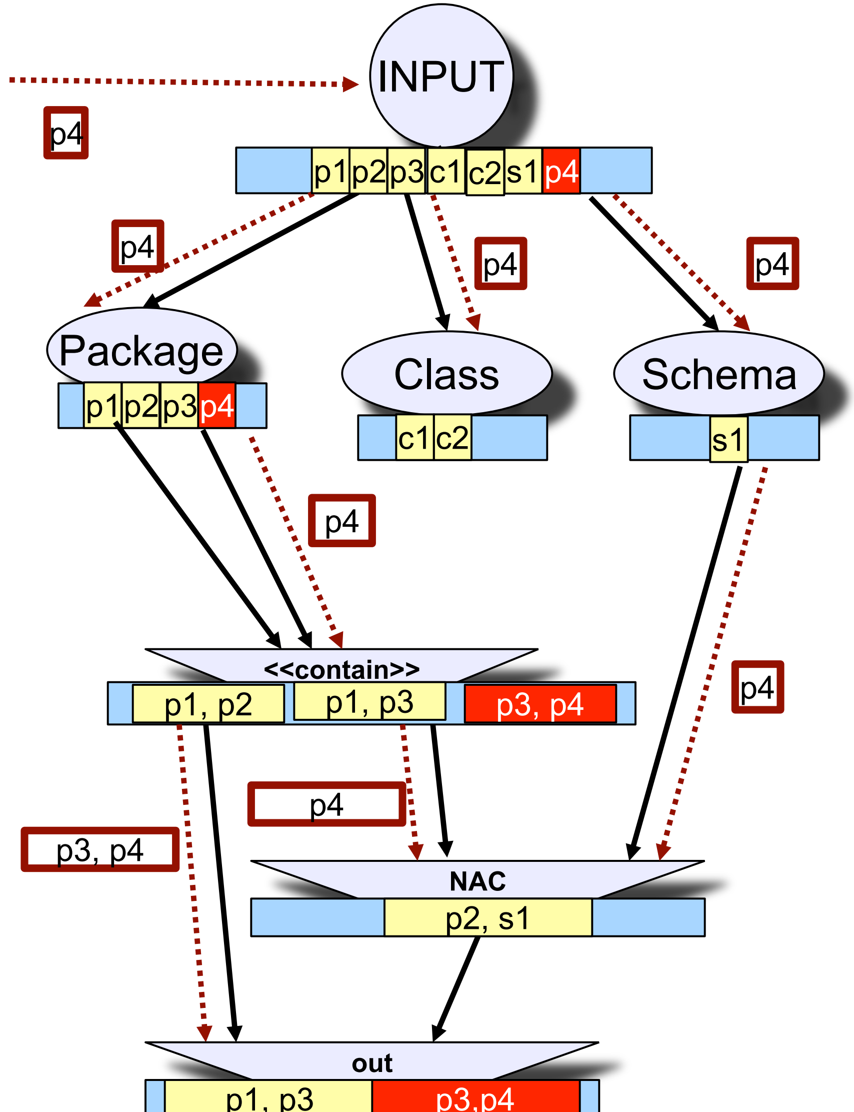




Incremental Pattern Matching Example

- RETE net
  - nodes: intermediate matchings
  - edge: update propagation
- Example
  - input: schemaRule pattern
  - pattern: contained Package
  - update: new package





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)

Summary

Summary

- MT is the backbone of MDE
- Getting into main stream
  (e.g., ATL, QVT: part of Eclipse modeling distrib)
- Key questions
  - Pattern matching
  - Model management (millions of elements)
  - Tooling support (debug, profiling, editors, etc.)
  - Testing and Verification
- Lot of research directions