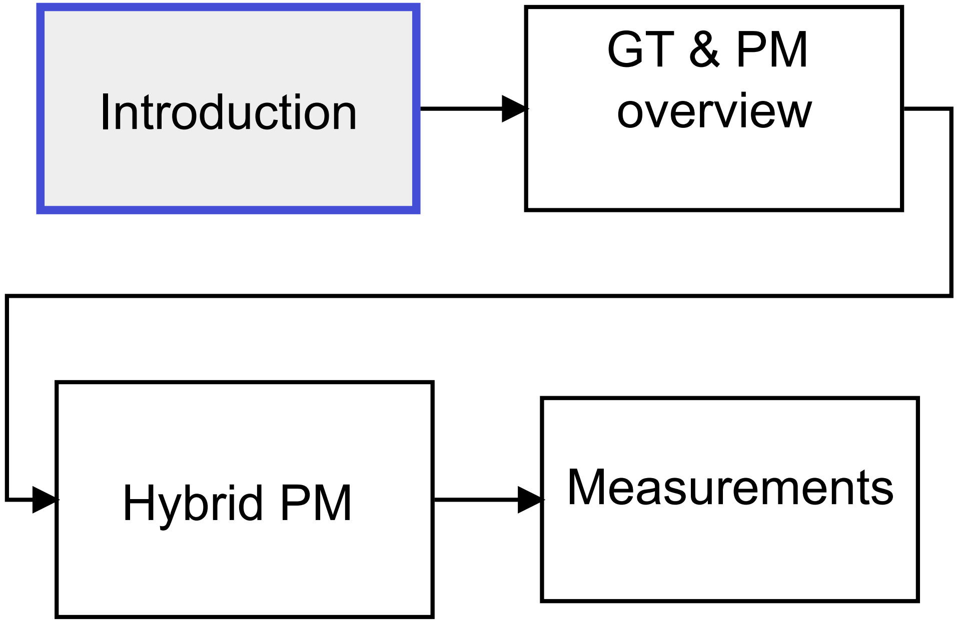
MŰEGYETEM 1782

Efficient Model Transformations by Combining Pattern Matching Strategies

Gábor Bergmann, Ákos Horváth,
István Ráth, Dániel Varró

Talk Overview



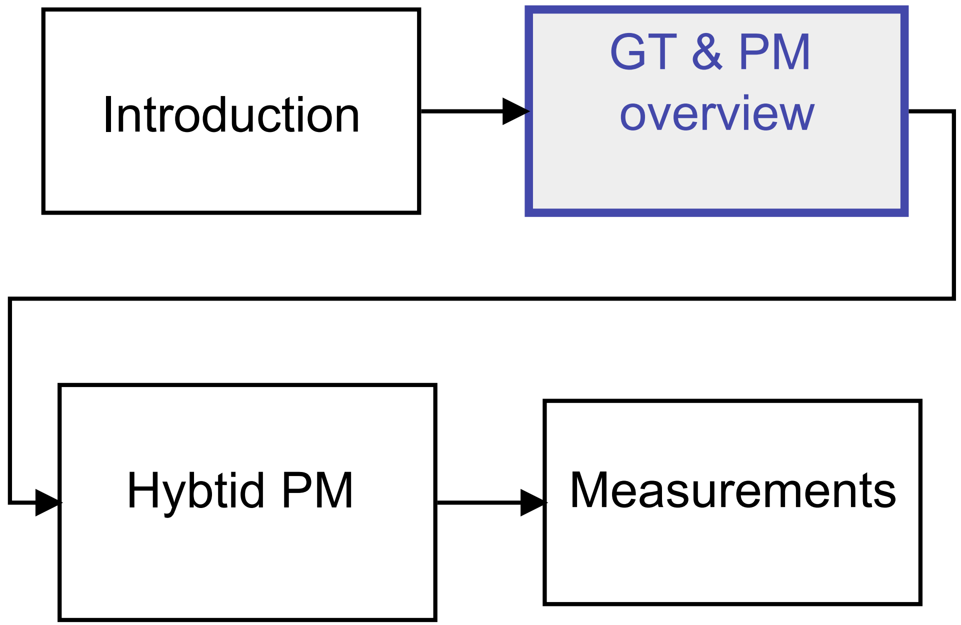
Introduction

- Common problem to be solved by model transformation tools:
  - Efficient query and manipulation of complex graph-based patterns
- One possible solution:
  - Graph transformation

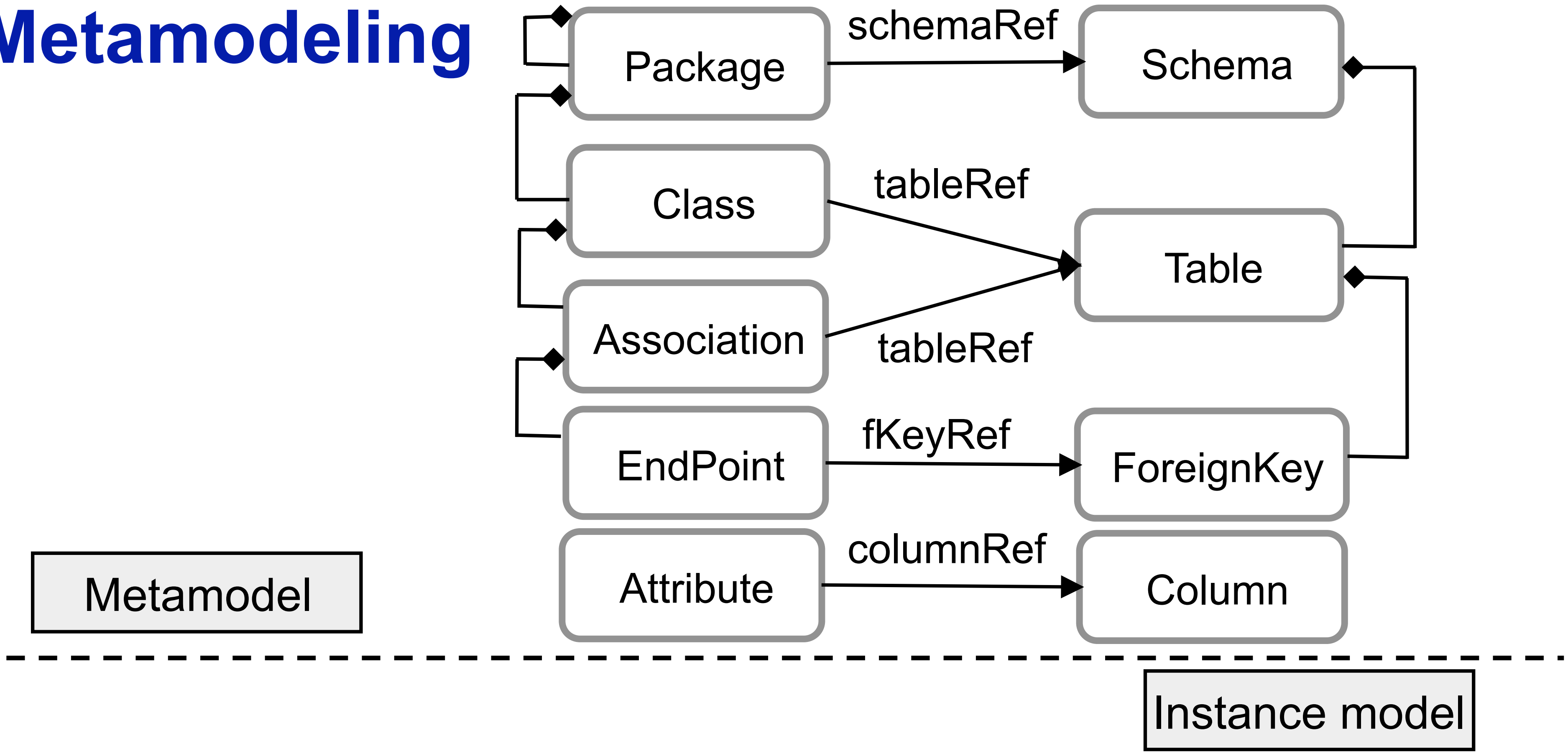
Benchmarking

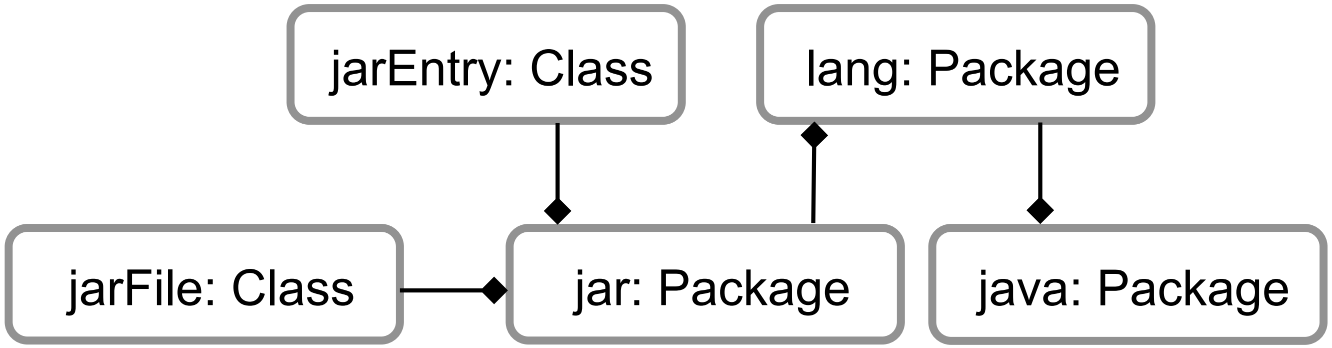
- Aim:
  - systematic and reproducible measurements
  - on performance
  - under varying and precisely defined circumstances
- Overall goal:
  - help transformation engineers in selecting tools
  - serve as reference for future research
- Popular approach in different fields
  - AI
  - relational databases
  - rule-based expert systems

Talk overview

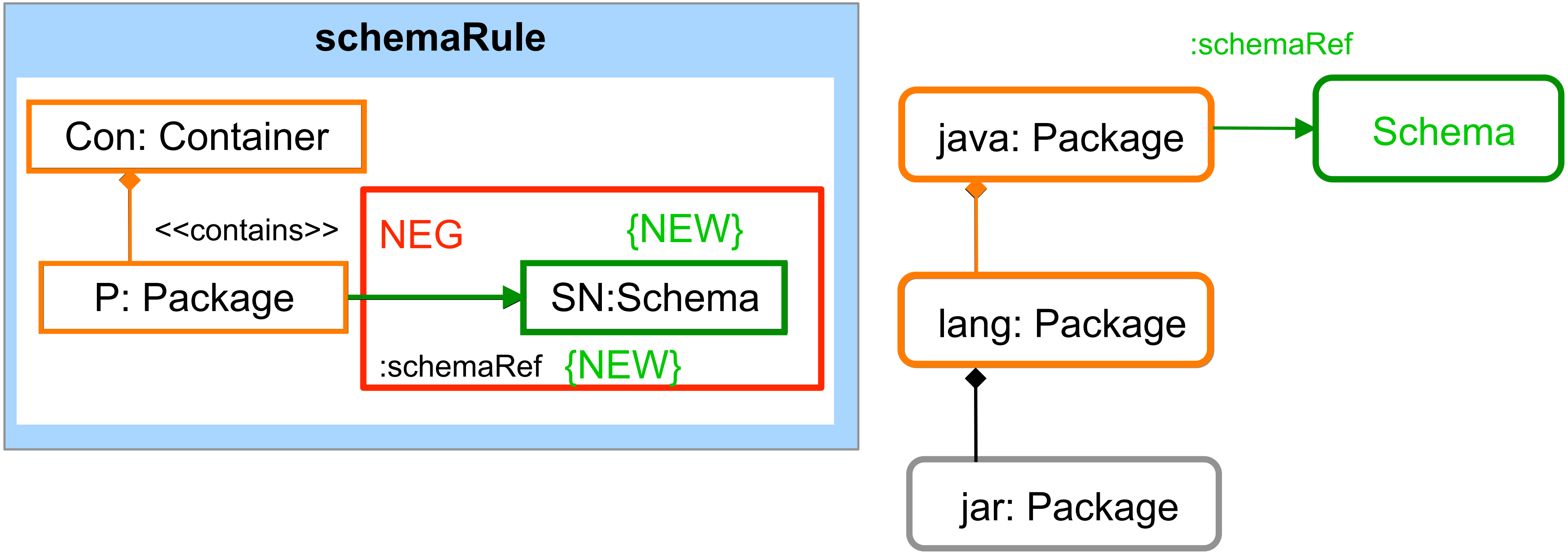


Metamodeling





Graph Transformation



Phases of GT matching
- Pattern Matching phase
- Updating phase: delete+ create

Pattern Matching is the most critical issue from the performance viewpoint (in our experience)

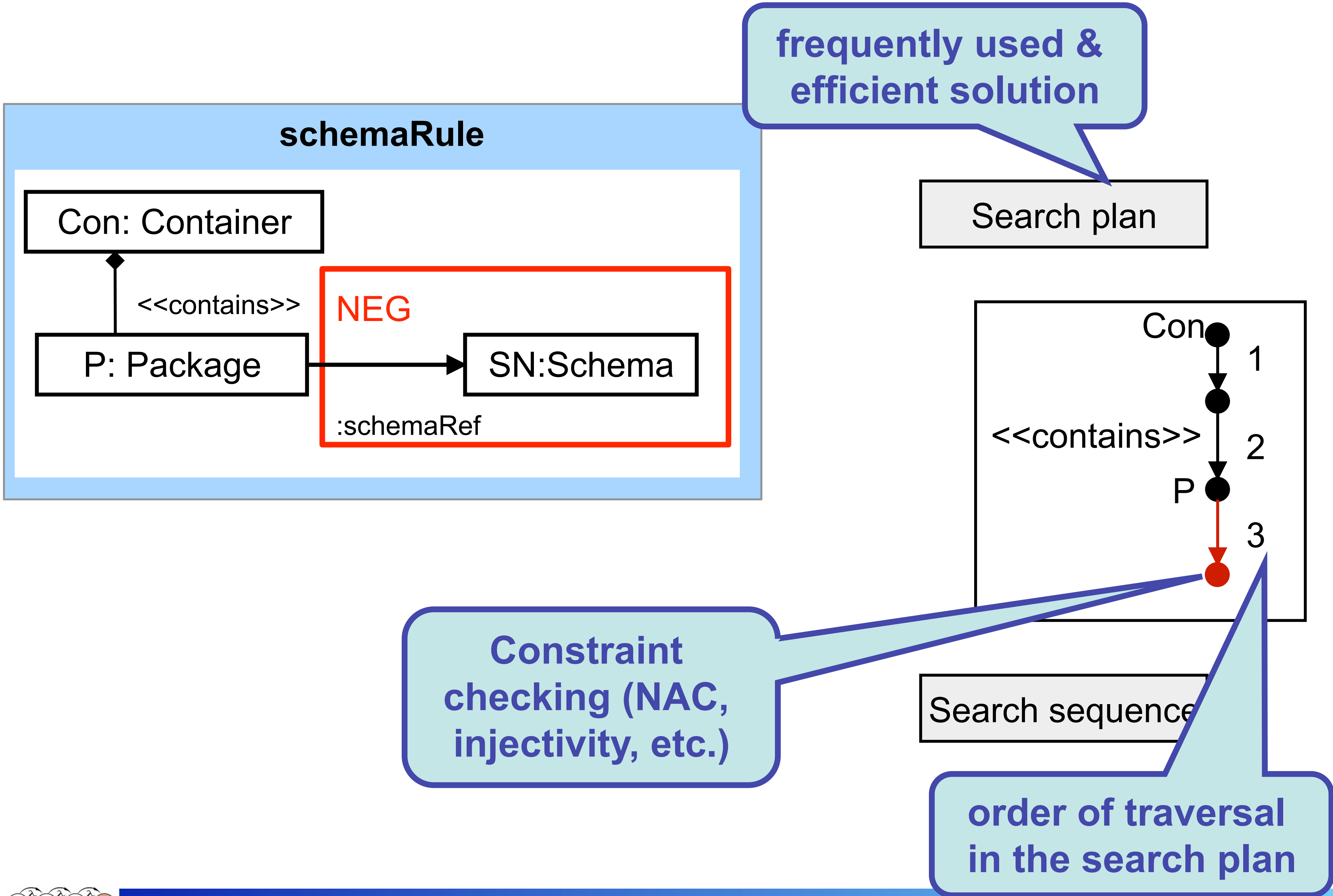
Pattern matching techniques

- Execution strategies
  - Interpreted: AGG (Tiger), VIATRA, MOLA, Groove, ATL
    - underlying PM engine
  - Compiled: Fujaba, GReAT, PROGRES, Tiger, VMTS, GrGEN.NET, ...
    - directly executed as C(#) or Java code
- Algorithms
  - Constraint satisfaction: AGG (Tiger)
    - variables + constraints
  - Local search (LS): Fujaba, GReAT, PROGRES, VIATRA, MOLA, Groove, Tiger (Compiled), GrGEN.NET, ...
    - step-by-step extension of the matching
  - Incremental (INC): VIATRA, Tefkat
    - Updated cache mechanism

Traditional Local Search-based pattern matching

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

Local Search based Pattern Matching Example

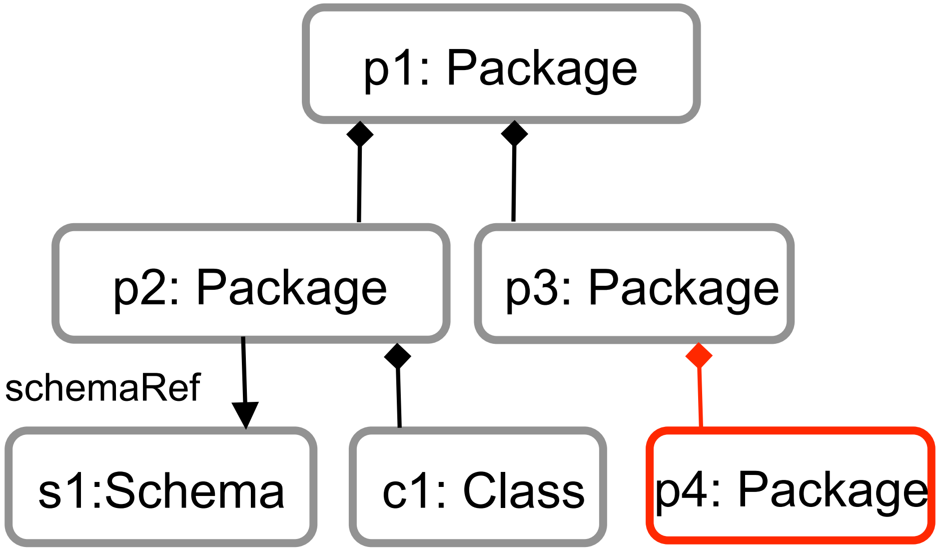


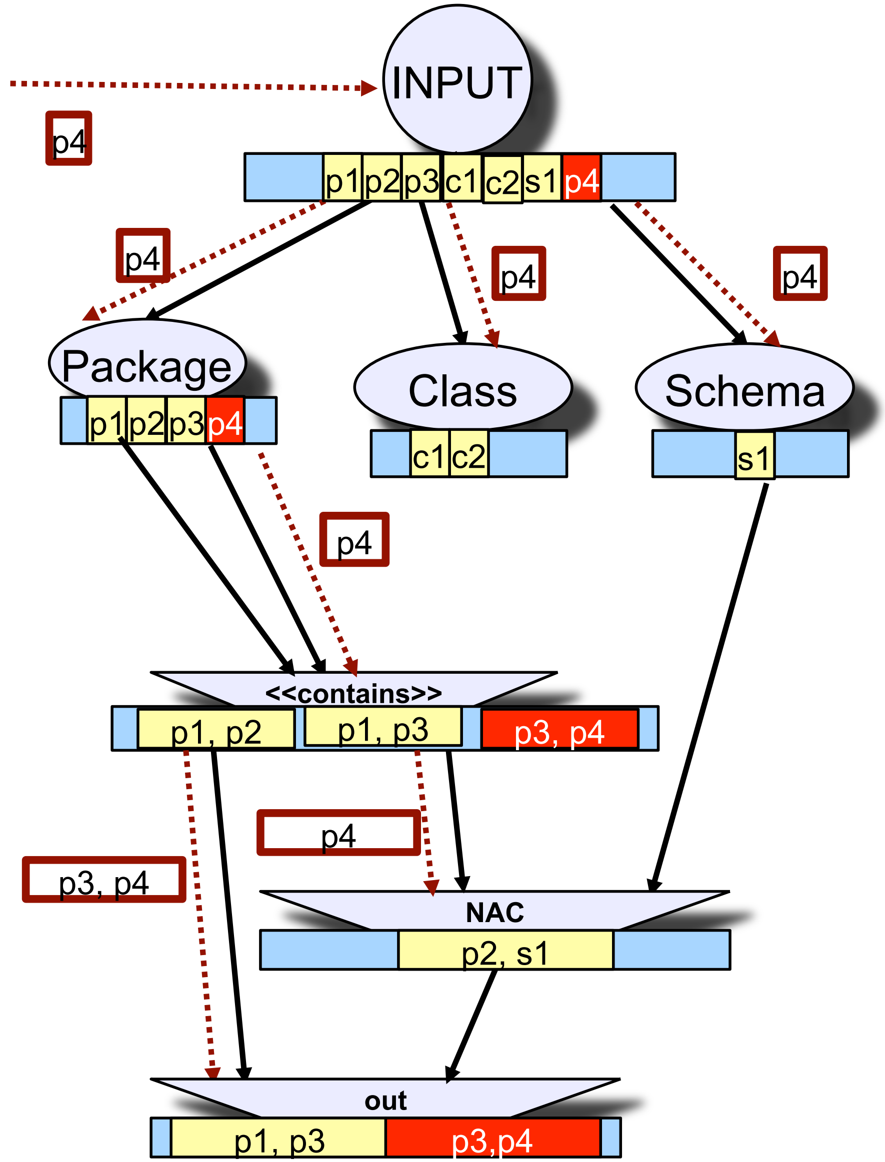
Incremental Pattern Matching

- 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.
- In VIATRA: an adapted RETE algorithm

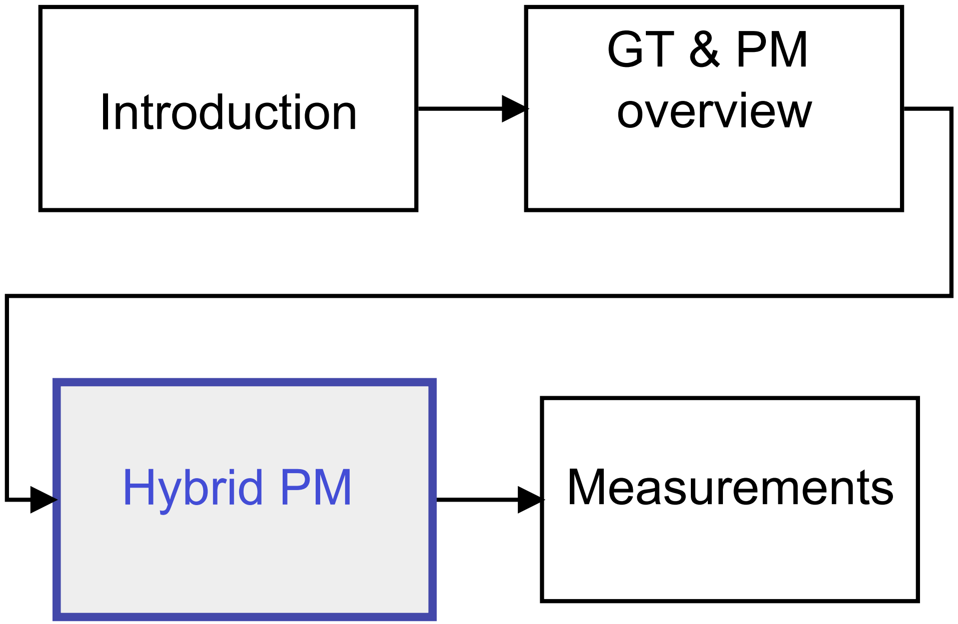
Incremental Pattern Matching Example

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





Talk overview



Hybrid pattern matching

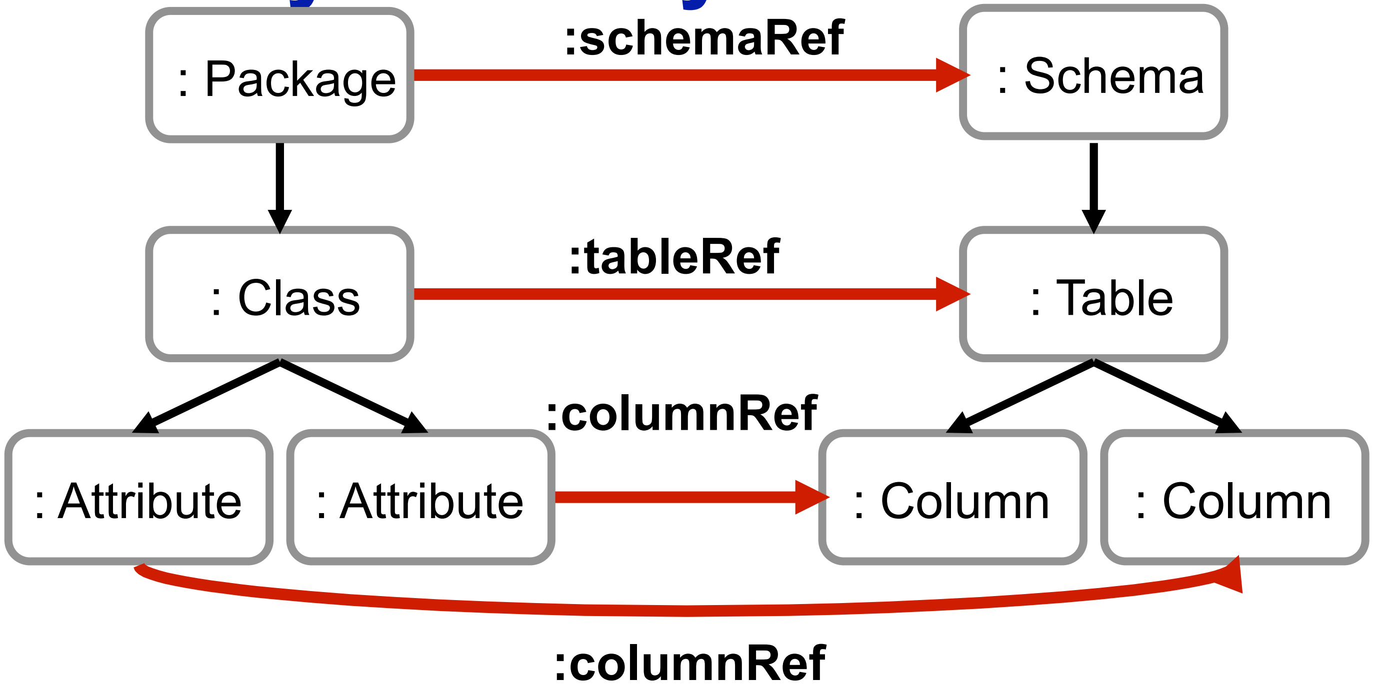
- Idea: combine local search-based and incremental pattern matching
- Motivation
  - Incremental PM is better for most cases, but…
    - Has memory overhead!
    - Has update overhead
  - → LS might be better in certain cases
- Based on experience with a ”real world” transformation application[1]

[1]Kovacs, M., Lollini, P., Majzik, I., Bondavalli, A.: *An Integrated Framework for the Dependability Evaluation of Distributed Mobile Applications* (SERENE’08)

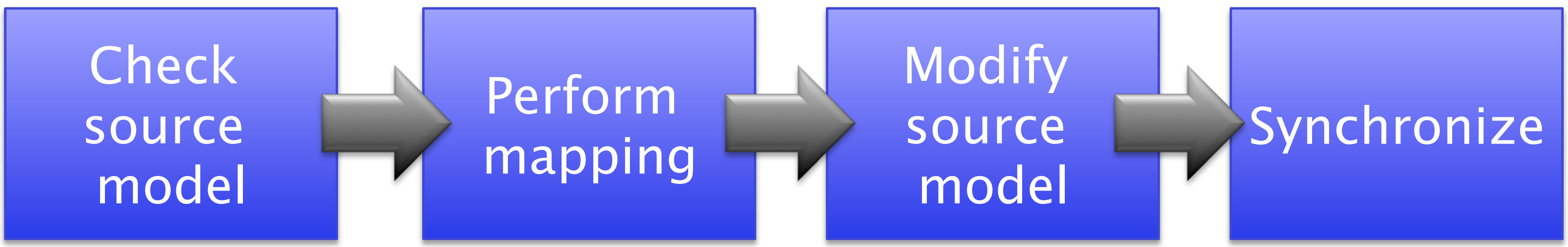
Where LS might be better…

- Memory consumption
  - RETE sizes grow roughly linearly with the model space
  - Constrained memory → trashing
- Cache construction time penalty
  - RETE networks take time to construct
  - „navigation patterns" can be matched quicker by LS
- Expensive updates
  - Certain patterns' matching set is HUGE
  - Depends largely on the transformation

Case study: ORM Synchronization



Transformation workflow



Phases

- Check Phase
  - Well-formedness checking
  - Static graph structure
  - No model manipulation
- Initial Transformation
  - Match reusability
  - Unidirectional
  - Complex rules
  - Batch like execution
- Refactoring
  - Single rule executed: move package in the hierarchy
  - Manual execution
- Synchronization
  - Match reusability
  - Unidirectional
  - Simple rules
  - Live execution

Phases

- Check Phase
  - Simple patterns, looking only for the first match
  - INC: cache construction penalty high
  - LS may be a better choice
- Initial Transformation
  - Processes the entire model (full traversal)
  - Match set may not fit into memory
  - Solution: decompose, use LS for certain patterns
- Refactoring
  - Move in containment hierarchy: very expensive cache update
  - LS can be significantly better
- Synchronization
  - INC significantly better (as demonstrated at ICGT08)

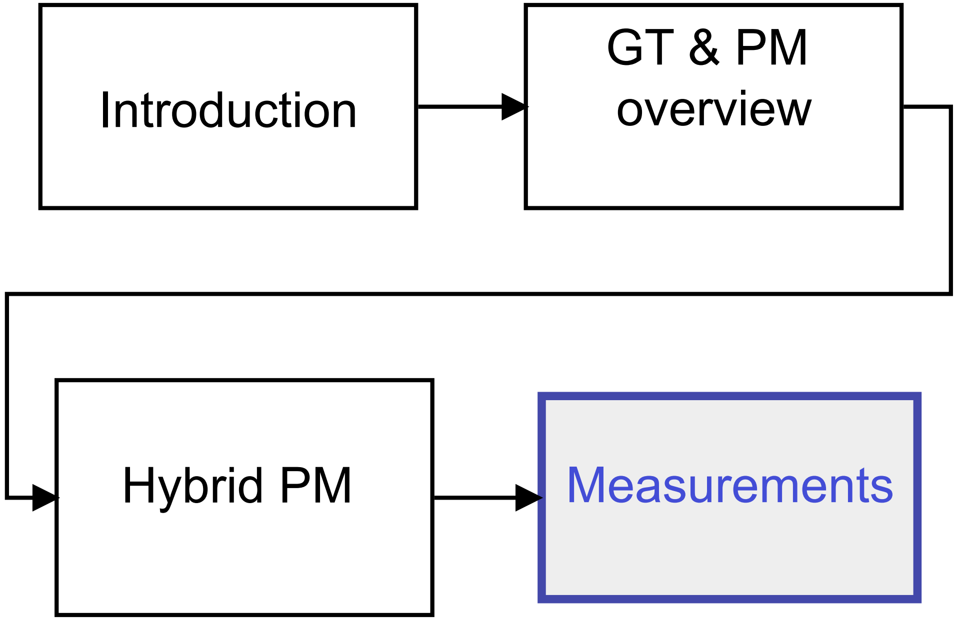
Hybrid PM in the source code

```
@incremental
pattern orphanTable(T) =
{
    table(T);
    neg pattern mapped(T) =
    {
        class(C);
        table(T);
        class.tableRef(REFN, C, T);
    } or {
        assoc(A);
        table(T);
        assoc.tableRef(REFN, A, T);
    }
}
```

Assign a PM implementation on a per-pattern (per-rule) basis → ability to fine tune performance on a very fine grained level.

```
@localsearch
pattern schemaRule_lhs(P) =
{
    package(P);
    neg pattern mapped(P, SN, REFN) = {
        package(P);
        schema(SN);
        package.schemaRef(REFN, P, SN);
    }
}
```

Talk overview



Environment

- Hardware and OS
  - 1.8 GHz Intel Core2 Duo
  - 2048 MB RAM
  - Windows XP SP3
  - Sun JVM 1.6.0_02 for VIATRA
- Tool related
  - VIATRA2 R3 Build 2009.02.03
  - Standard services of the default distribution

Composite ORM Synchronization benchmark

Considerations for selecting PM strategy

- Graph pattern static attributes
  - Number of patterns
  - Pattern size
  - Containment constraints
- Control structures
  - Parameter passing
  - Usage frequencies
  - Model update cost
- Model-dependent pattern characteristics
  - Model statistics (instance count)

Adaptive hybrid pattern matching

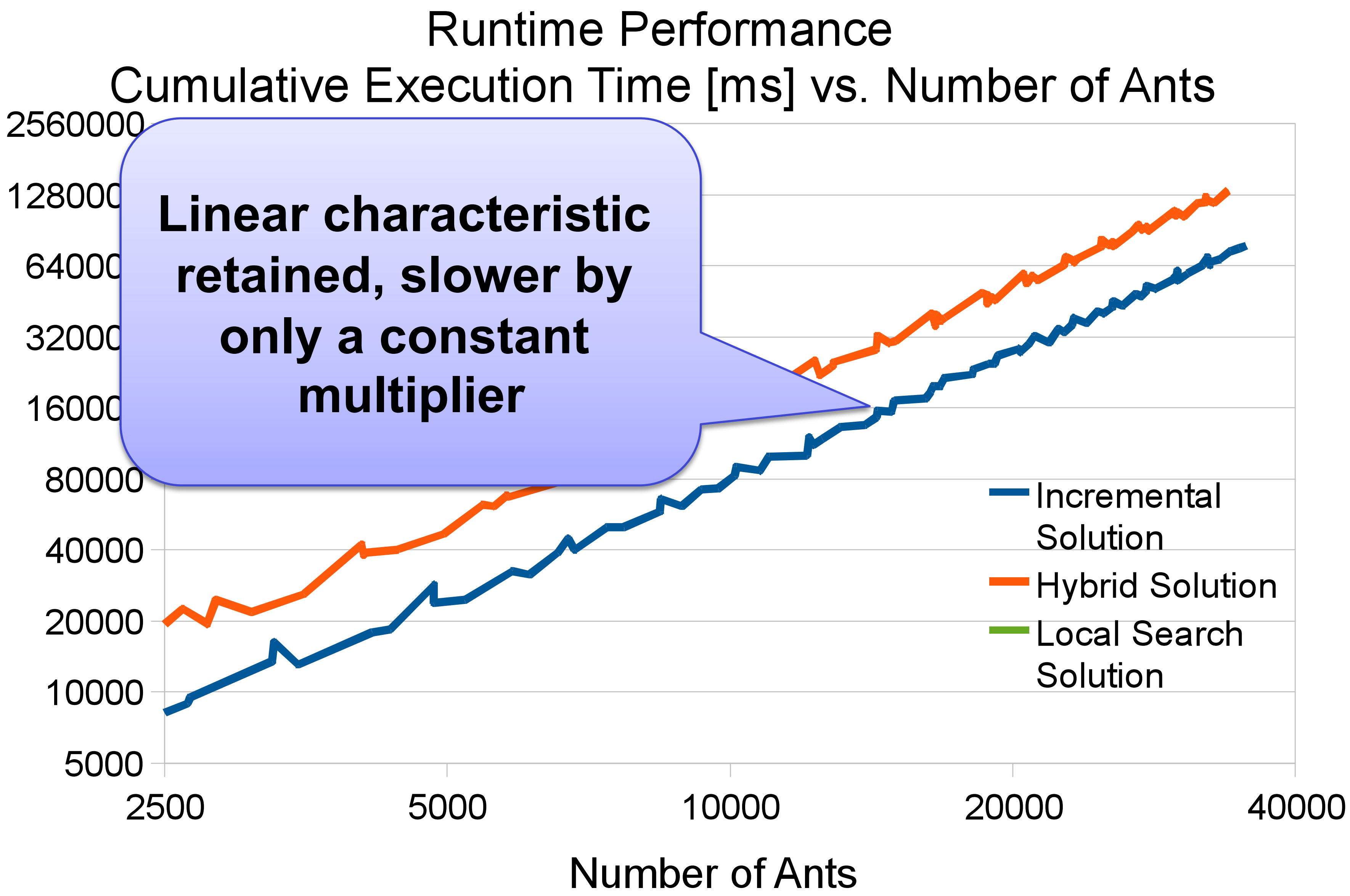
- Goal: provide (semi-) automatic aid for strategy selection
- Idea: monitor memory usage
  - JVM telemetry
- Prevent heap exhaustion
  - Destroy match set cache structures
  - Switch to LS

PM Strategy	Used heap [MB]	Transform phase execution time [s]
LS	201	77.1
INC	353	13.6
Static hybrid	220	10.9
Adaptive hybrid	**235**	**35.7**

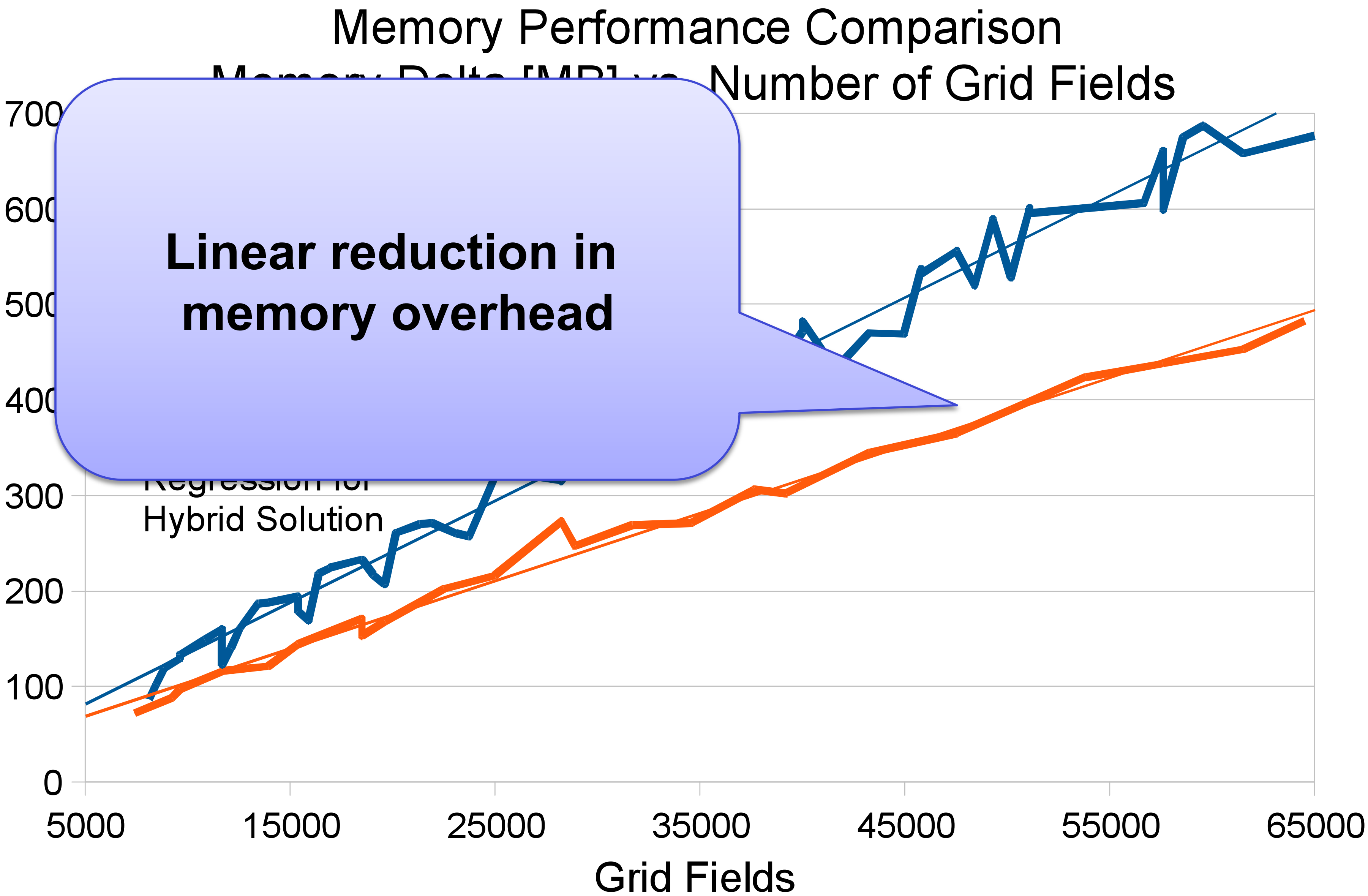
Further benchmarking

- Paper for a GRaBaTS 2008 special issue in STTT'09: Experimental assessment of combining pattern matching strategies with VIATRA2
- In-detail investigation
  - hybrid approach
  - Transformation language-level optimizations
- Optimization
  - Based on experience with ICMT'09 paper

AntWorld Results



Memory footprint



Summary

Optimization strategy	Performance	Memory footprint
LS	High order polynomial	Constant
Switch to INC	Polynomial order reduction	Linear increase with model size
Switch to Hybrid	Linear (~50%) reduction	Linear (~50%) reduction

- In short: you may get a linear decrease in memory for a linear increase in execution time → retains complexity class characteristics ☺