EMF-IncQuery: A high-performance model query engine for EMF models

István Ráth

rath@mit.bme.hu

Budapest University of Technology and Economics

Fault Tolerant Systems Research Group

Overview

- A model query engine
  - Supports batch queries
  - Optimized for incremental queries!
- Incrementality
  - *Compute once, update afterwards*
  - Gain: Instant re-evaluation
  - Price: Uses some more memory
    - Manageable with proper life cycles
- Why?
  - Queries are everywhere
    - UI (views), business logic (transformations), tooling (validation)
  - Existing technology has performance issues with large models
  - Current query languages are difficult (to reuse)

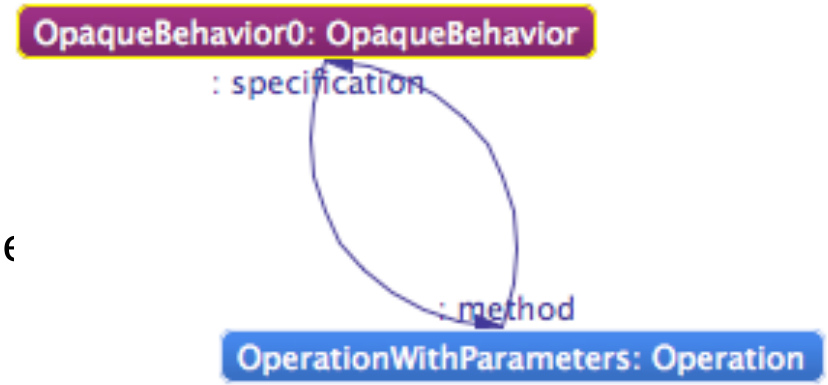
Query: a piece of code that looks/iterates through certain parts of an (EMF) instance model

Benefits

- Makes on-the-fly well-formedness validation, view maintenance, … feasible over really large instance models
  - Large: >1M elements
  - Difference is noticeable for a few hundred elements too, depending on your queries
- Simplifies writing really complex queries
  - Graph pattern language
  - Highly reusable, you can even build query libraries
  - Performance does not (significantly) depend on query complexity!
- Easy-to-integrate into existing apps
  - works with any EMF DSL
  - Query engine can be attached to any Notifier or TransactionalEditingDomain

EXAMPLE A simple UML validation constraint

- "All Behaviors must have an Operation as their specification."
  - Otherwise they do not have any "interface" through which they could be accessed → "dead code"



- Bad case:

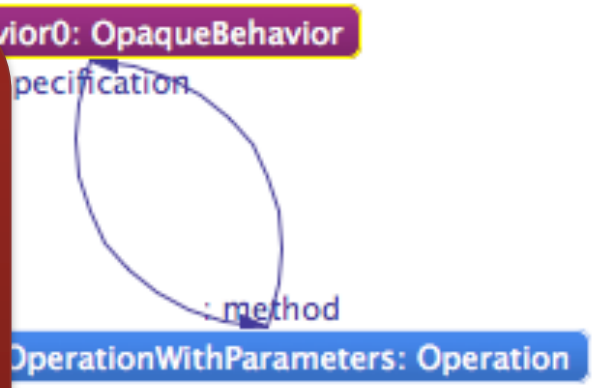
```
@Constraint(mode="problem",location="Behavior",
    message="The behavior $Behavior$ has no specification operation.")
pattern OpaqueBehaviorWithoutOperation(Behavior) = {
    OpaqueBehavior(Behavior);

    neg pattern behaviorHasSpecification(Behavior) = {
        Behavior(Behavior);
        Behavior.specification(SpecRel,Behavior,Specification);
        BehavioralFeature.method(BFRel,Specification,Behavior);
        Operation(Specification);
    }
}
```

EXAMPLE A simple UML validation constraint

Expressive declarative query language by graph patterns

- Capture local + global queries
- Compositionality + Reusabilility
- „Arbitrary” Recursion, Negation



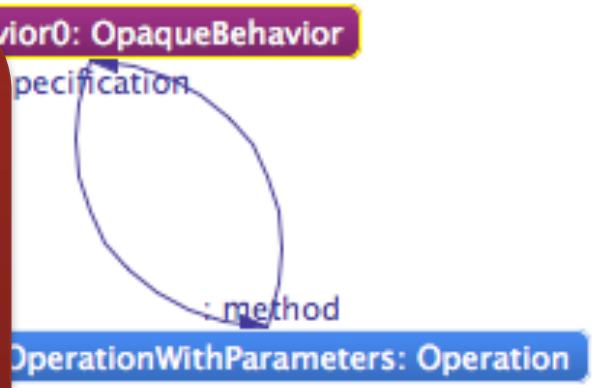

```
cification operation.")
pattern u aviorWithoutOperation(Behavior) = {
    OpaqueBehavior(Behavior);

    neg pattern behaviorHasSpecification(Behavior) = {
        Behavior(Behavior);
        Behavior.specification(SpecRel,Behavior,Specification);
        BehavioralFeature.method(BFRel,Specification,Behavior);
        Operation(Specification);
    }
}
```

EXAMPLE A simple UML validation constraint

Expressive declarative query language by graph patterns

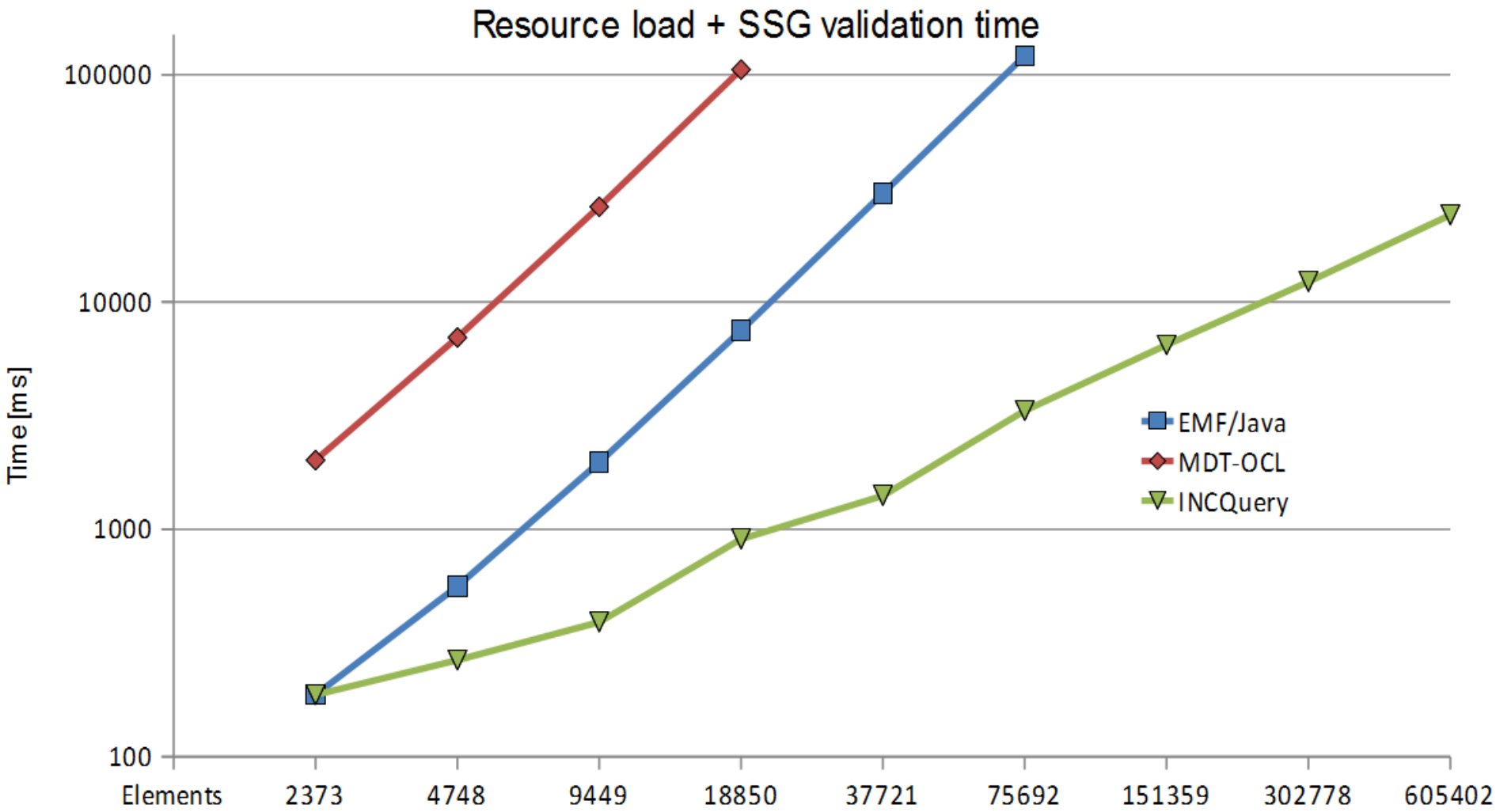
- Capture local + global queries
- Compositionality + Reusabilility
- „Arbitrary” Recursion, Negation



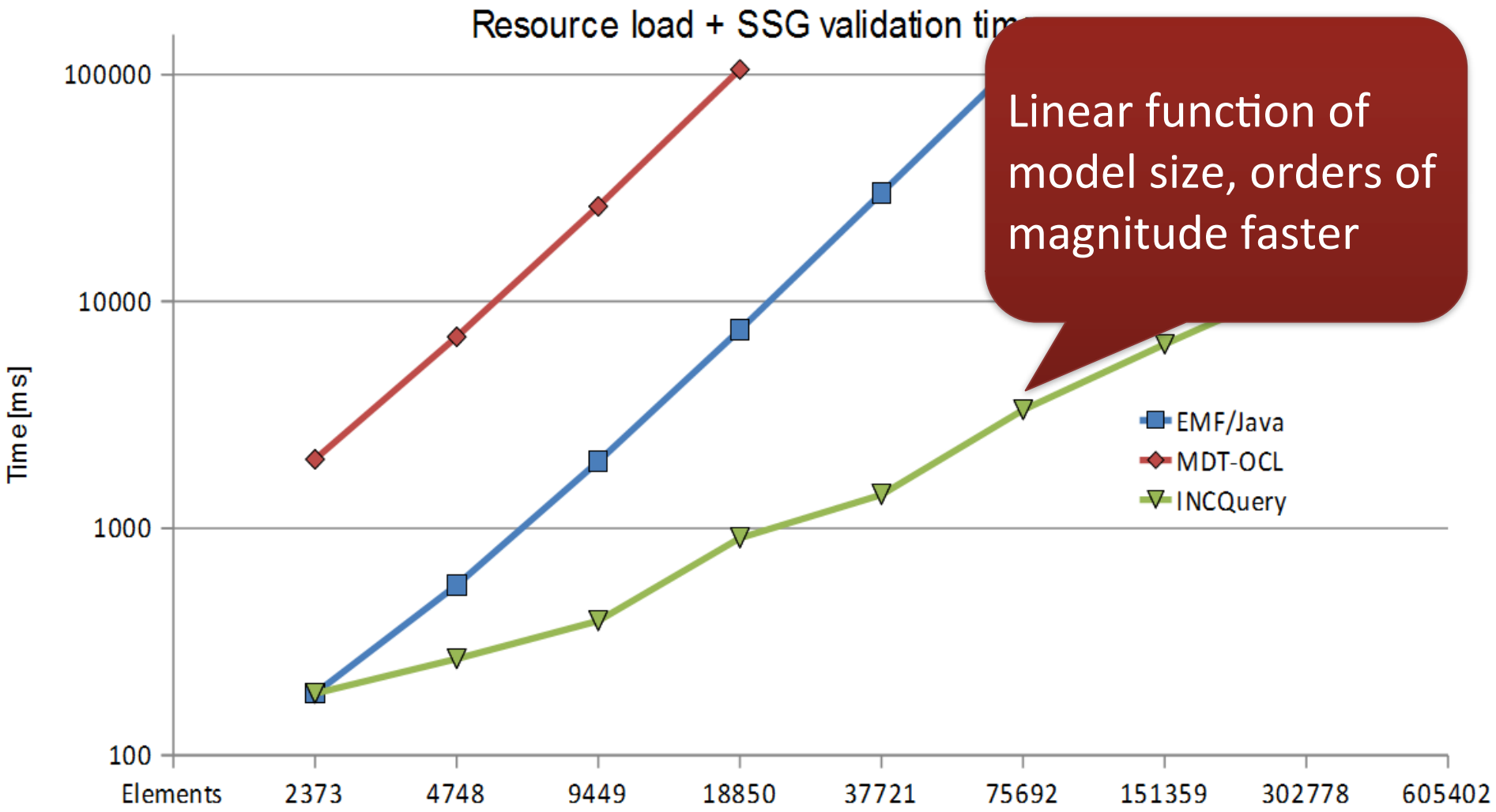

```
cification operation.")
pattern   aviorWithoutOperation(Behavior) = {
    OpaqueBehavior(Behavi
    neg patte
        Behav
        Behav
        Behav
        Opera
    }
}
```

- Uses the IncQuery Validation Engine to manage validation markers on-the-fly
- No manual coding necessary
- Generically integrates into EMF tree editors and GMF editors

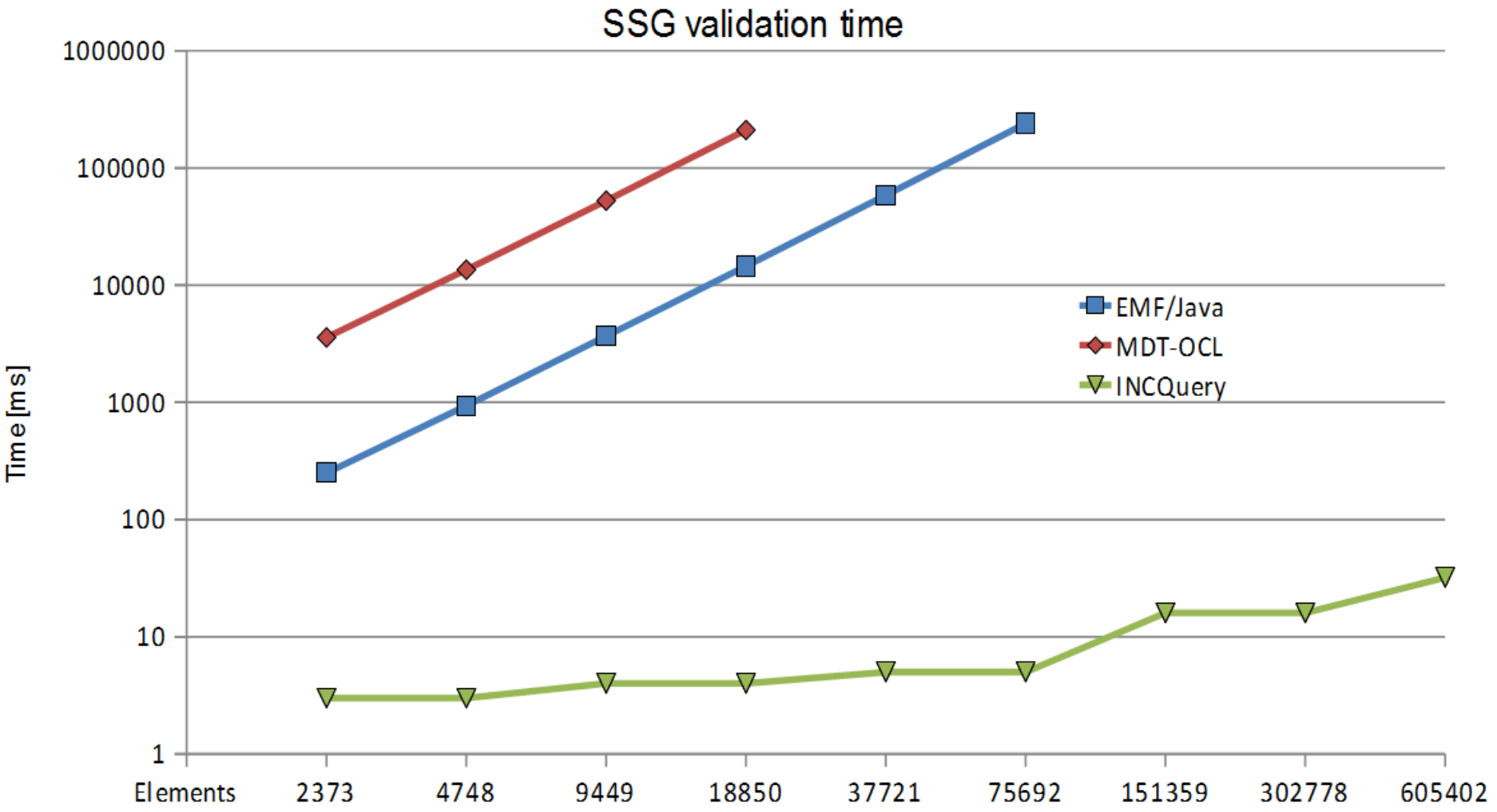
Batch mode


Resource load + SSG validation time
Time [ms]
100000
10000
1000
100
Elements
2373
4748
9449
18850
37721
75692
151359
302778
605402
EMF/Java
MDT-OCL
INCQuery

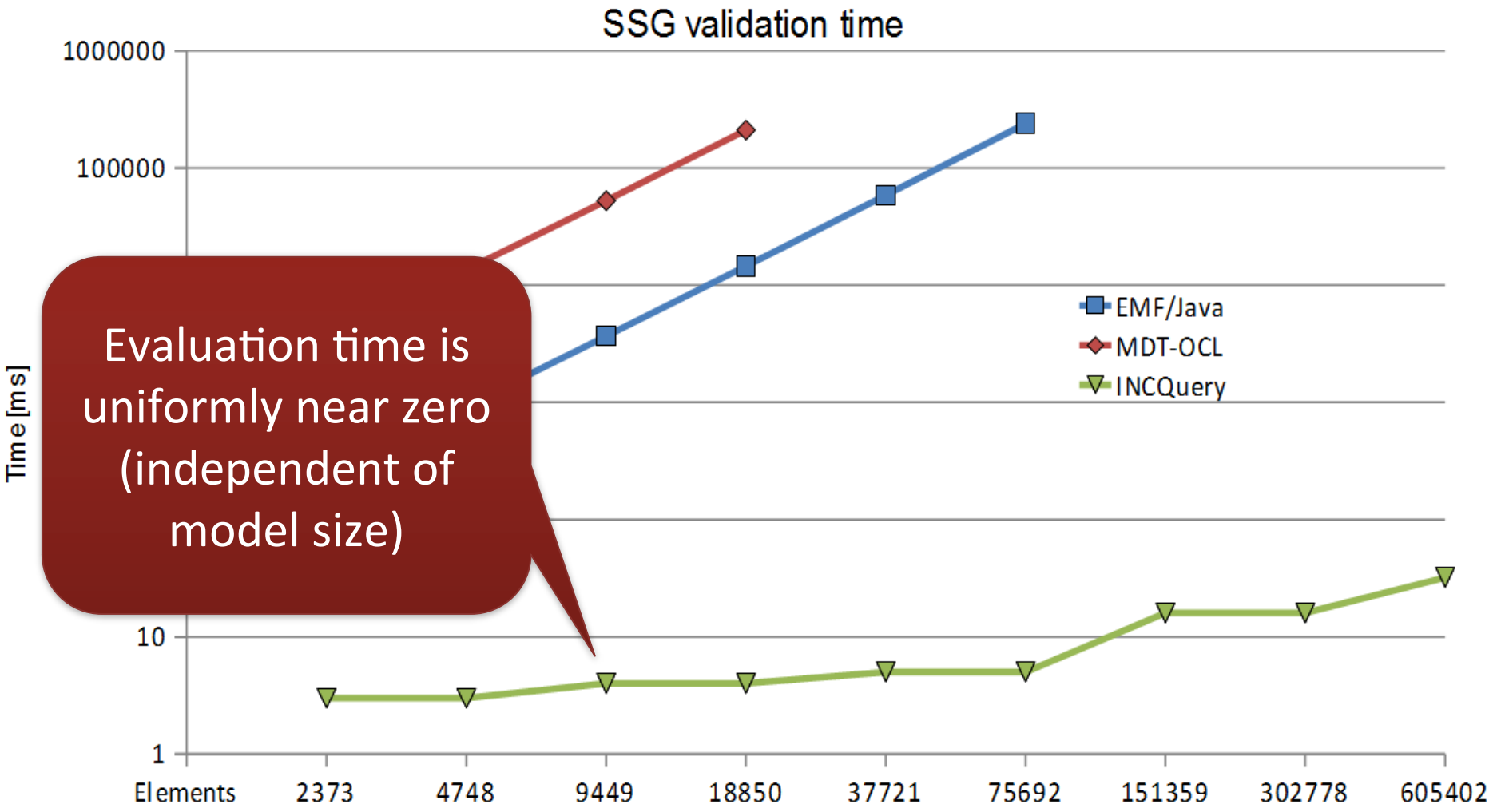
Batch mode


Resource load + SSG validation tim
100000
10000
1000
100
Time [ms]
Elements
2373
4748
9449
18850
37721
75692
151359
302778
605402
EMF/Java
MDT-OCL
INCQuery
Linear function of model size, orders of magnitude faster

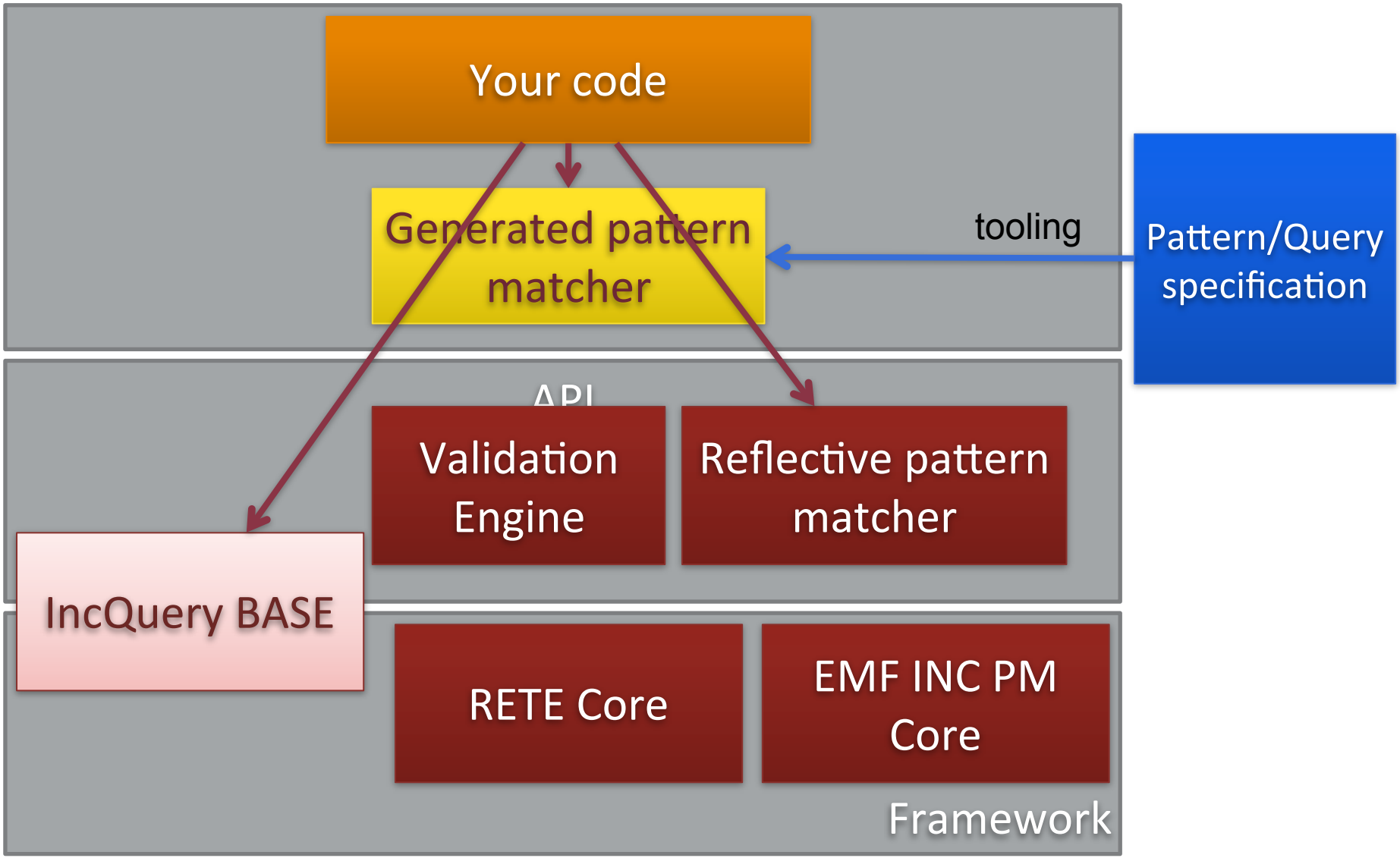
Incremental mode



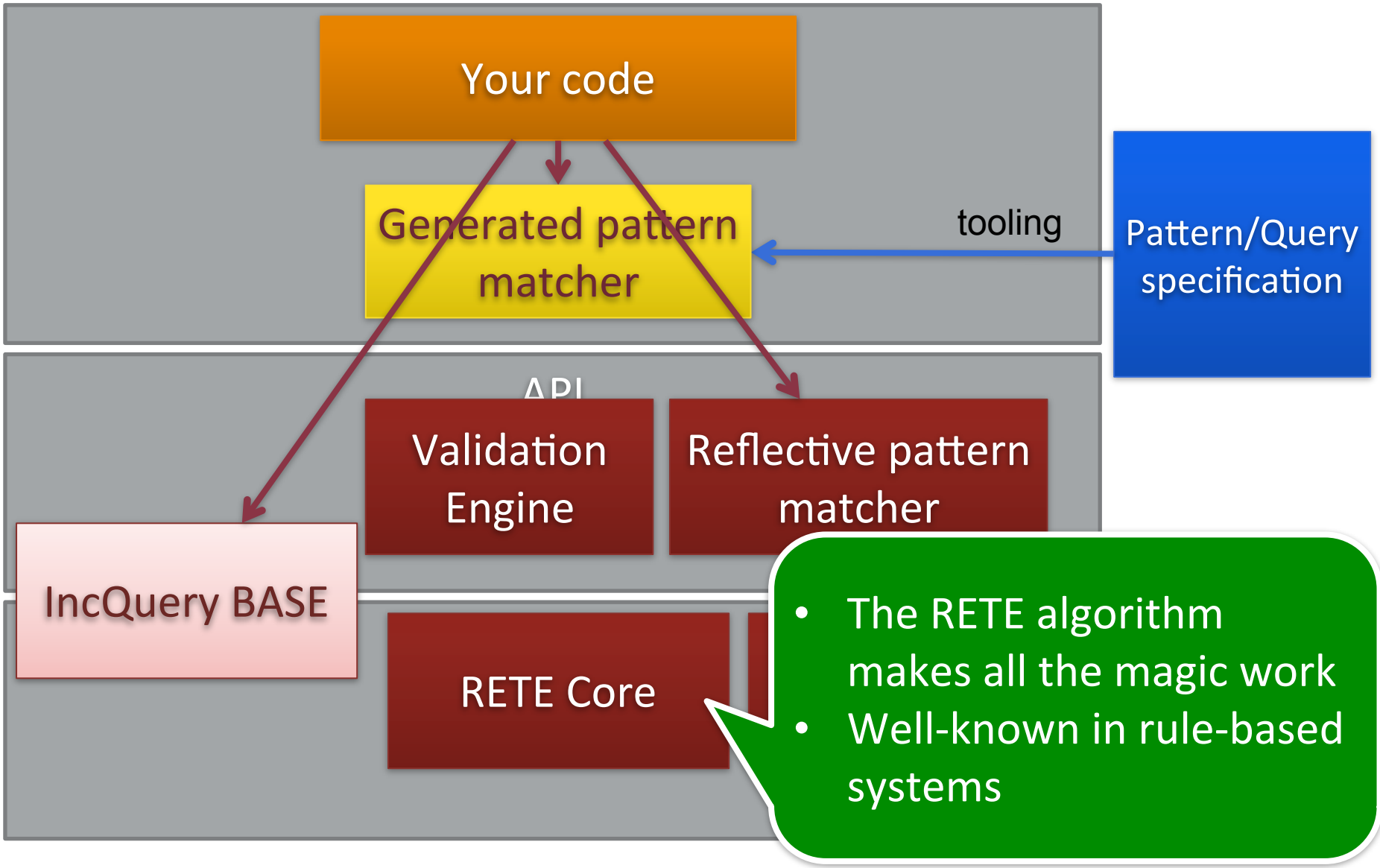
Incremental mode



EMF-IncQuery Architecture v0.7



EMF-IncQuery Architecture v0.7



IncQuery BASE

- Light-weight Java library for simple (yet very powerful) EMF model queries, with **incremental evaluation**
- Supports
  - Get all instance elements by type
  - Reverse navigation along references
  - Get model elements by attribute value/type
- Very easy to integrate into any EMF tool (pure Java) – **standalone!**
- Same high performance and scalability as IncQuery
- Incremental transitive closure
  - Computation of e.g. reachability regions, connected model partitions, …
  - Innovative new algorithm for general graphs

Development roadmap: IncQuery v0.7

- Tooling
  - Xtext2-based tooling
  - Incorporating
    - Unlimited recursion and transitive closure
    - Short attribute notation
    - Aggregate functions
    - Match (or even exceed) most of the expressive power of OCL, while providing better re-use and more concise notation
- Runtime
  - Reflective queries
    - Build and execute queries on-the-fly, using Java and IQ PL
  - RETE construction optimizations
    - Goal: to significantly reduce memory footprint

Further plans & collaboration opportunities

- Integrate IncQuery Base and IncQuery to EMF modeling tools
  - Support for Epsilon, ATL, OCL, …
- Performance boost for derived EFeatures
  - and notifications!

Check it out!

- Pointers
  - http://viatra.inf.mit.bme.hu/incquery
    - Documentation, language reference
    - Tutorials
    - Examples
  - http://viatra.inf.mit.bme.hu/incquery/base
  - rath@mit.bme.hu
  - viatra-dev@inf.mit.bme.hu