M^3: an Open Model for Measuring Code Artifacts

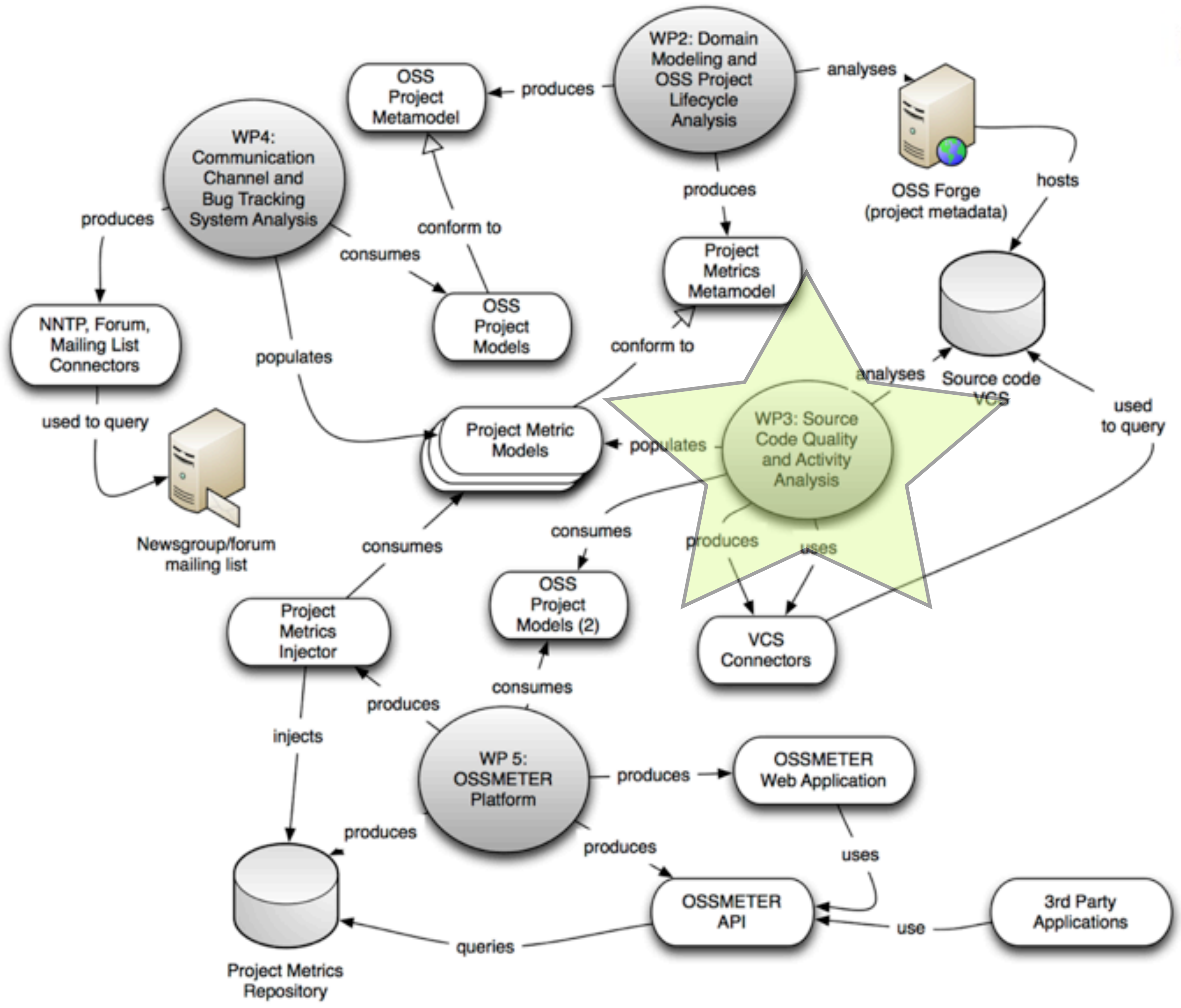
Anastasia Izmaylova, Paul Klint, Ashim Shahi, Jurgen Vinju

SWAT

Centrum Wiskunde & Informatica (CWI)

OSSMETER:

FP7 STREP Project: Automated Measurement and Analysis of Open Source Software.



Consortium

- The Open Group (UK) (Project co-ordinator)
- University of York (UK) (Technical co-ordinator)
- University of Manchester (UK)
- Centrum Wiskunde & Informatica (NL)
- University of L'Aquila (IT)
- Tecnalia (ES)
- Softeam (FR)
- Uninova (PT)
- Unparallel Innovation (PT)

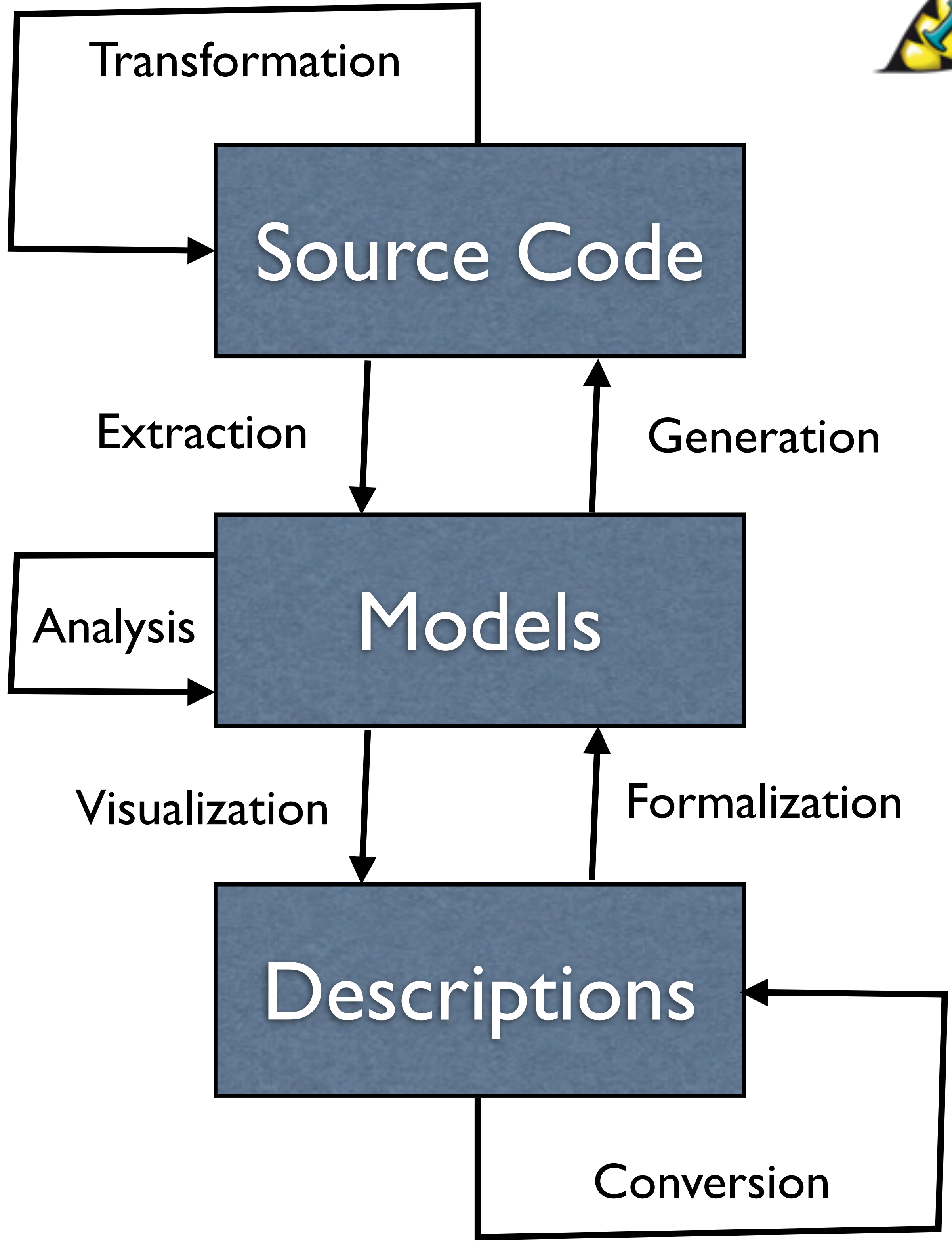
OSSMETER Objectives

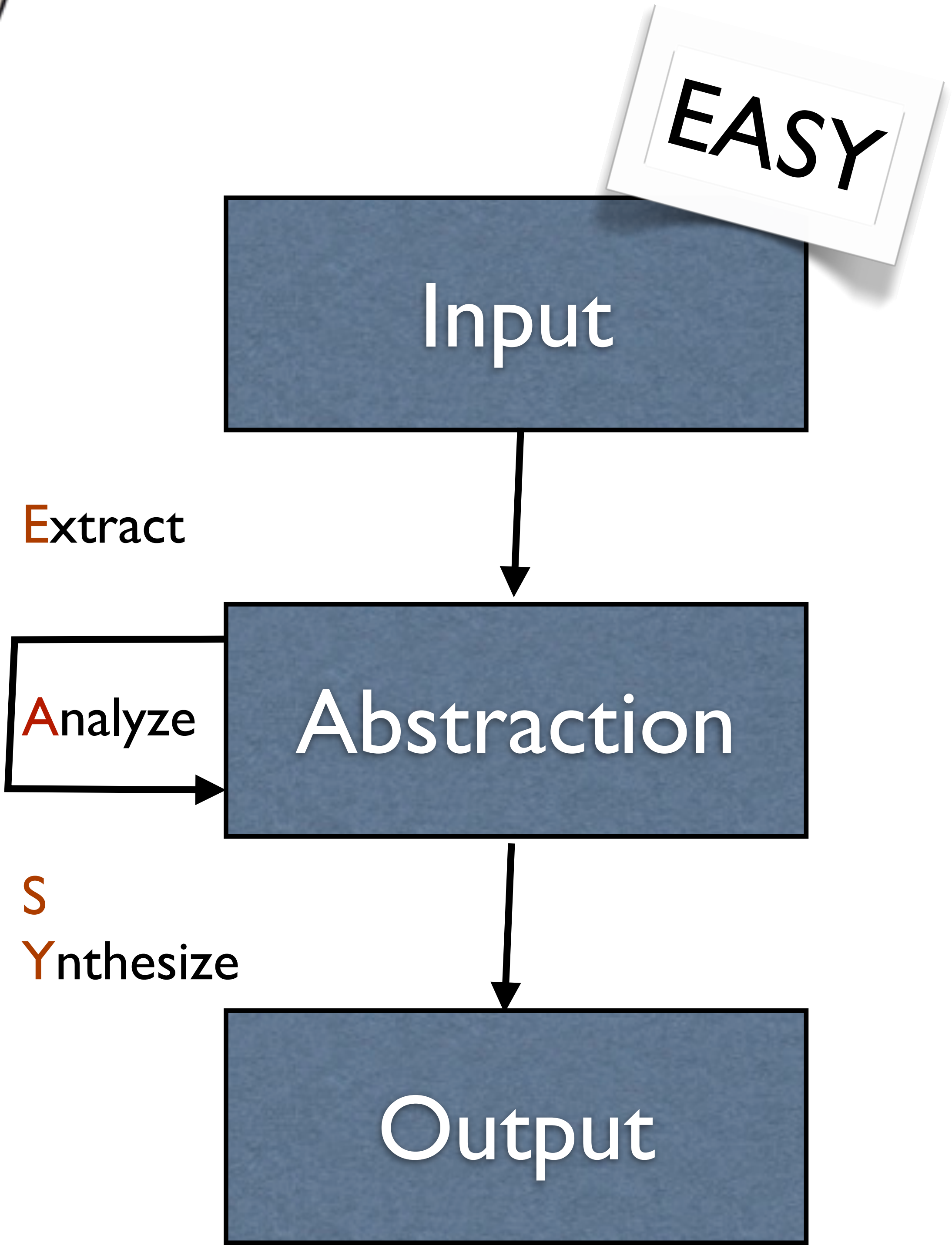
- Objectives
  - Get insight in source code quantity & quality
  - Get insight into programmer activity
- Means
  - Measure source code
  - Measure source code version "deltas"

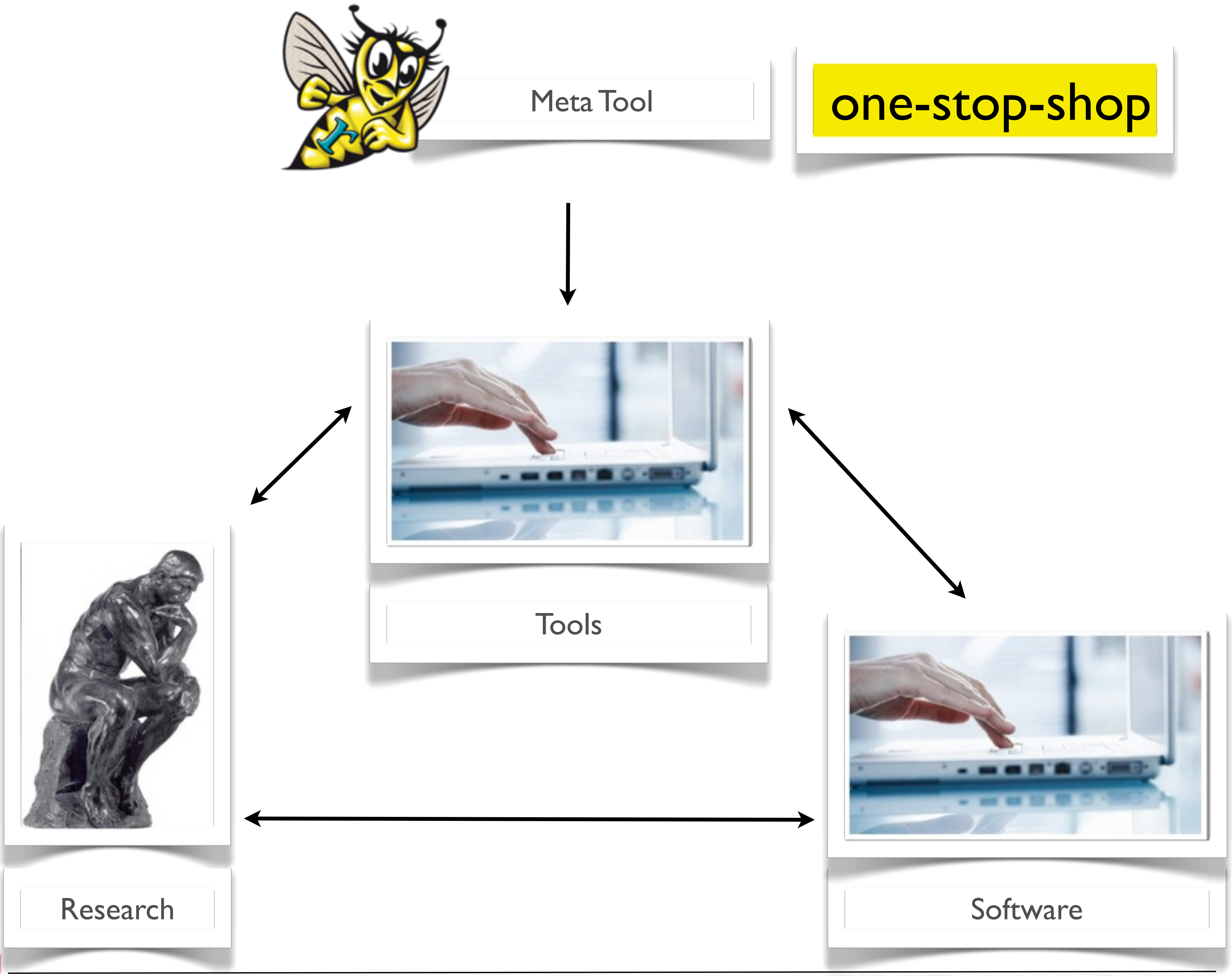
SWAT & Rascal

- Software exploration, transformation, generation, visualization, specification: software *-ation :-)
  - Domain specific languages
  - Analyzing software
- Teaching Evolution @ UvA et al.

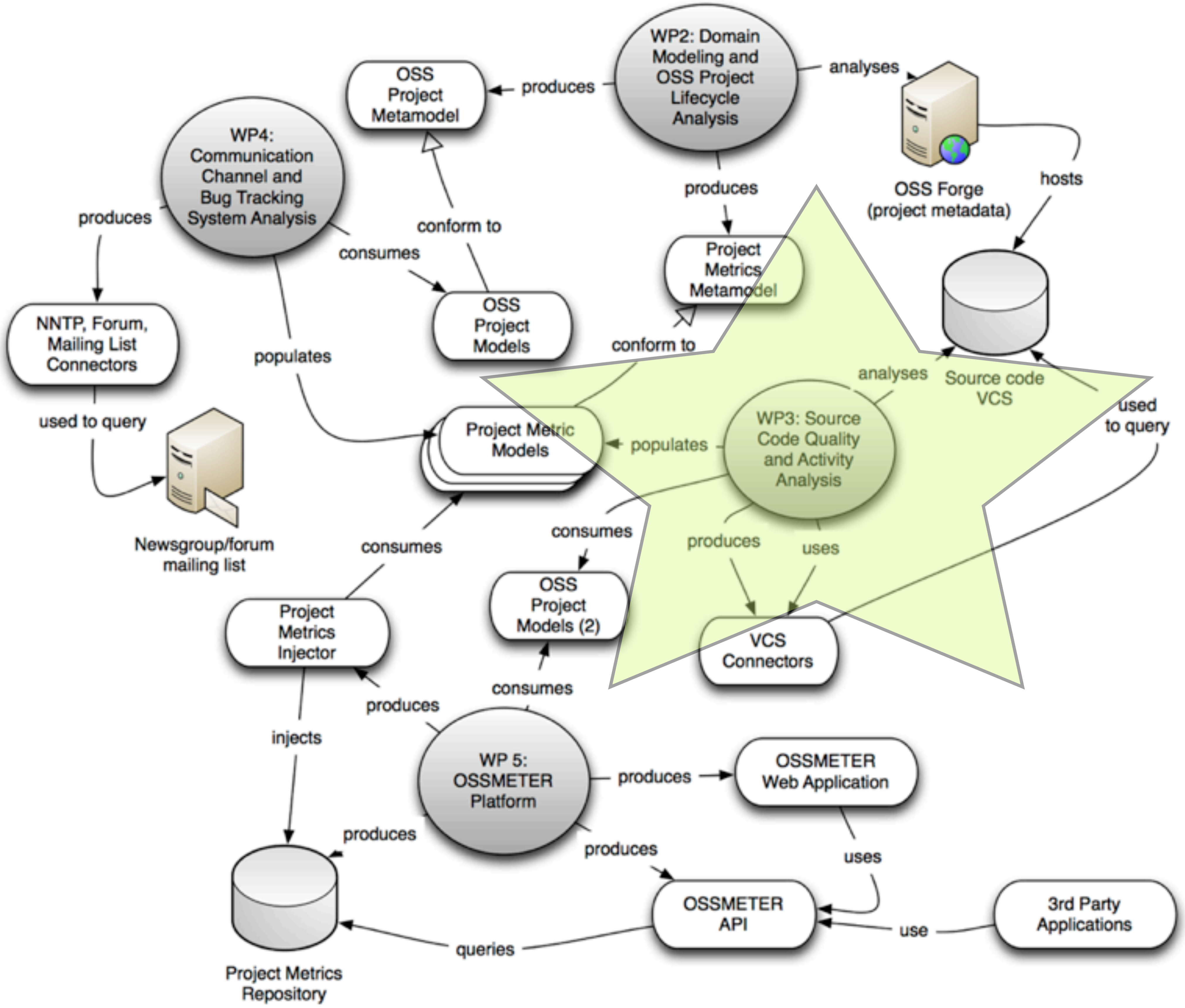
- Rascal is a DSL
- Lab infra-structure
- “one stop shop”
- Safe & simple


Transformation
Source Code
Extraction
Generation
Models
Analysis
Visualization
Formalization
Descriptions
Conversion


EASY
Input
Extract
Abstraction
Analyze
S
Ynthesize
Output


Meta Tool
one-stop-shop
Tools
Research
Software

OSSMETER: EU FP7 STREP Project on Automated Measurement and Analysis of Open Source Software.

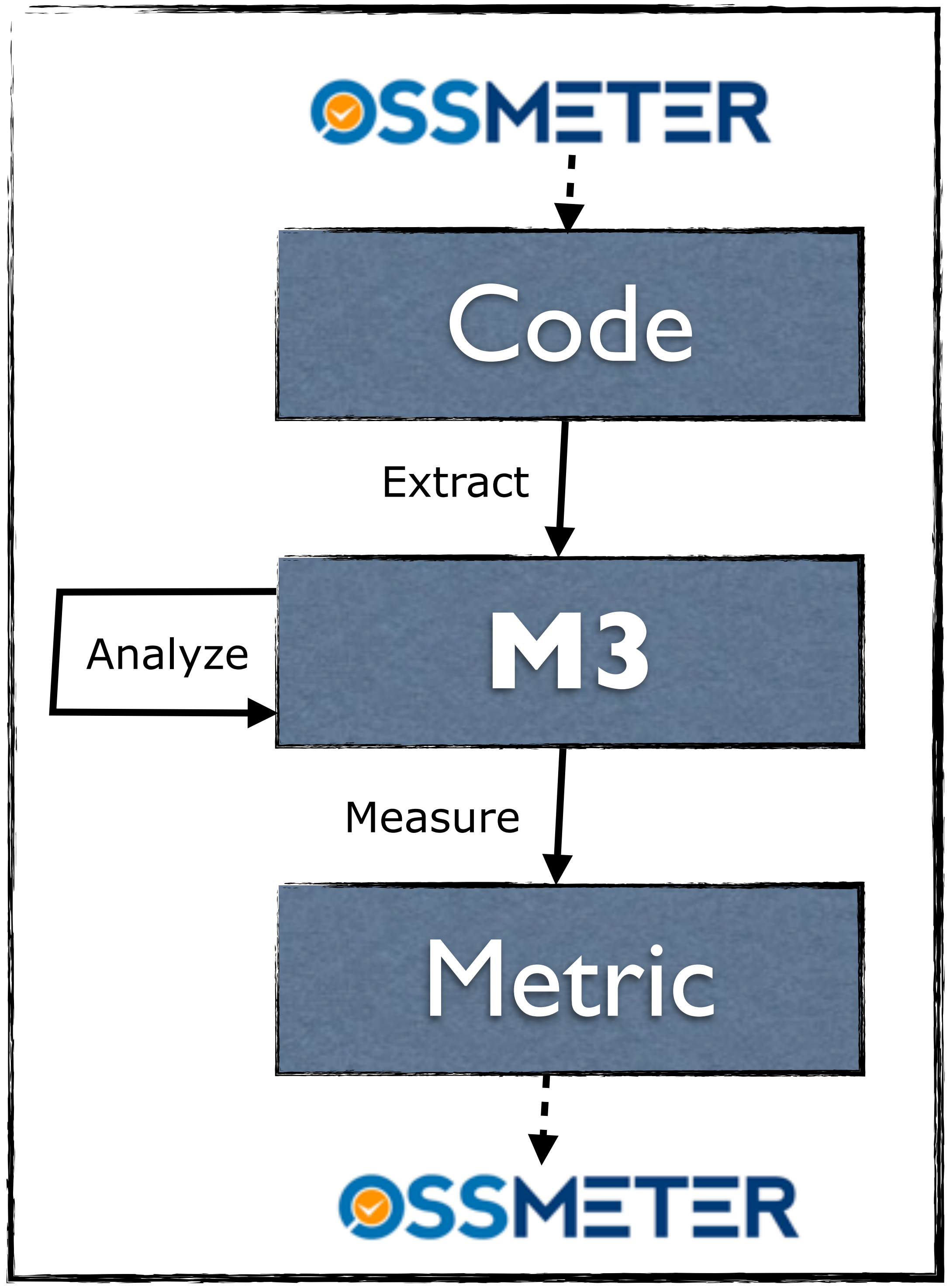


Challenges

precision vs. efficiency
deep vs. shallow
constraints vs. types vs. CFGs vs. RegEx

variety of languages & metrics:
reuse, *consistency* and <u>scale</u>

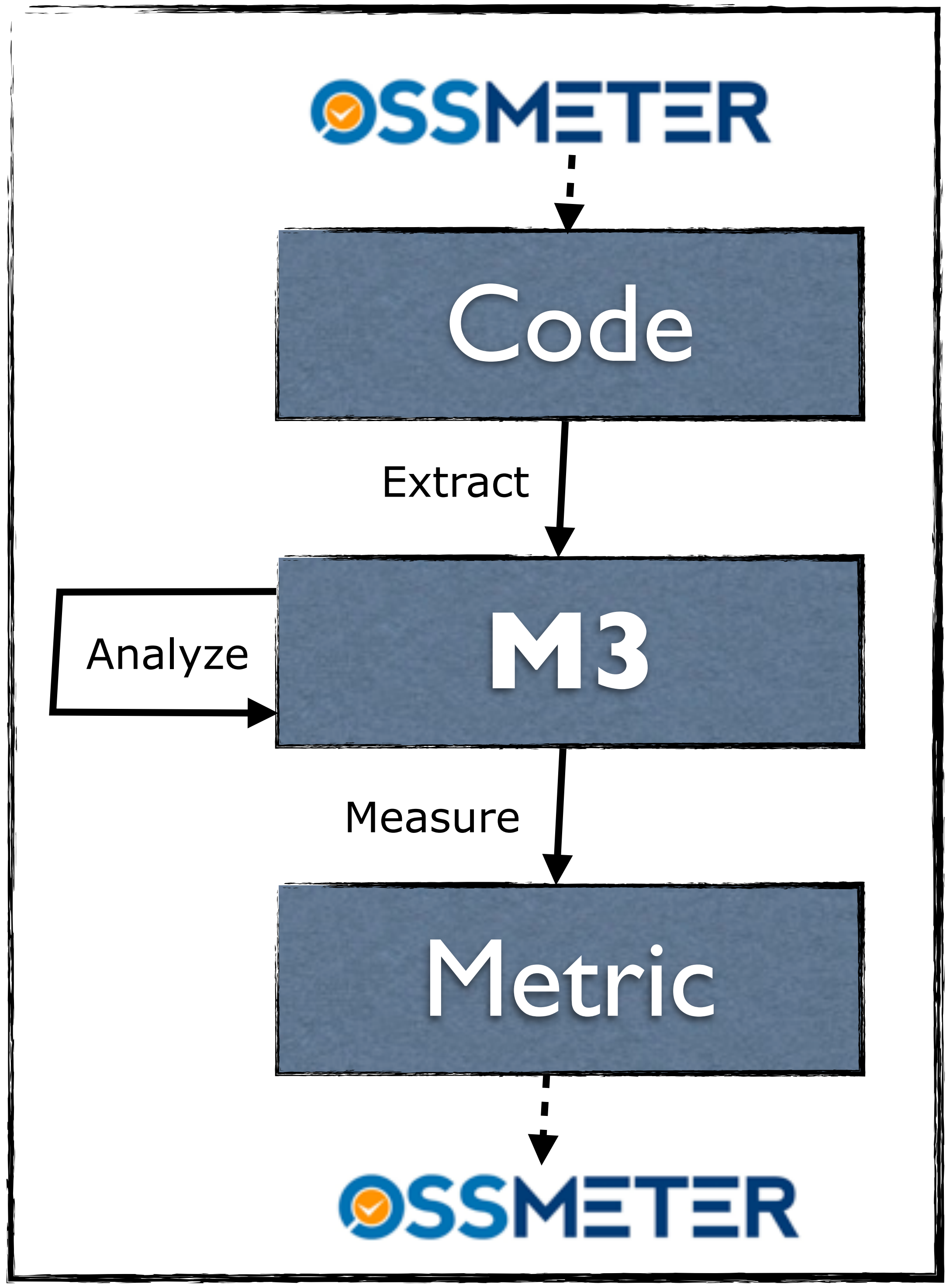
M3



[inspired by FAMIX, SOUL, and others]

M3

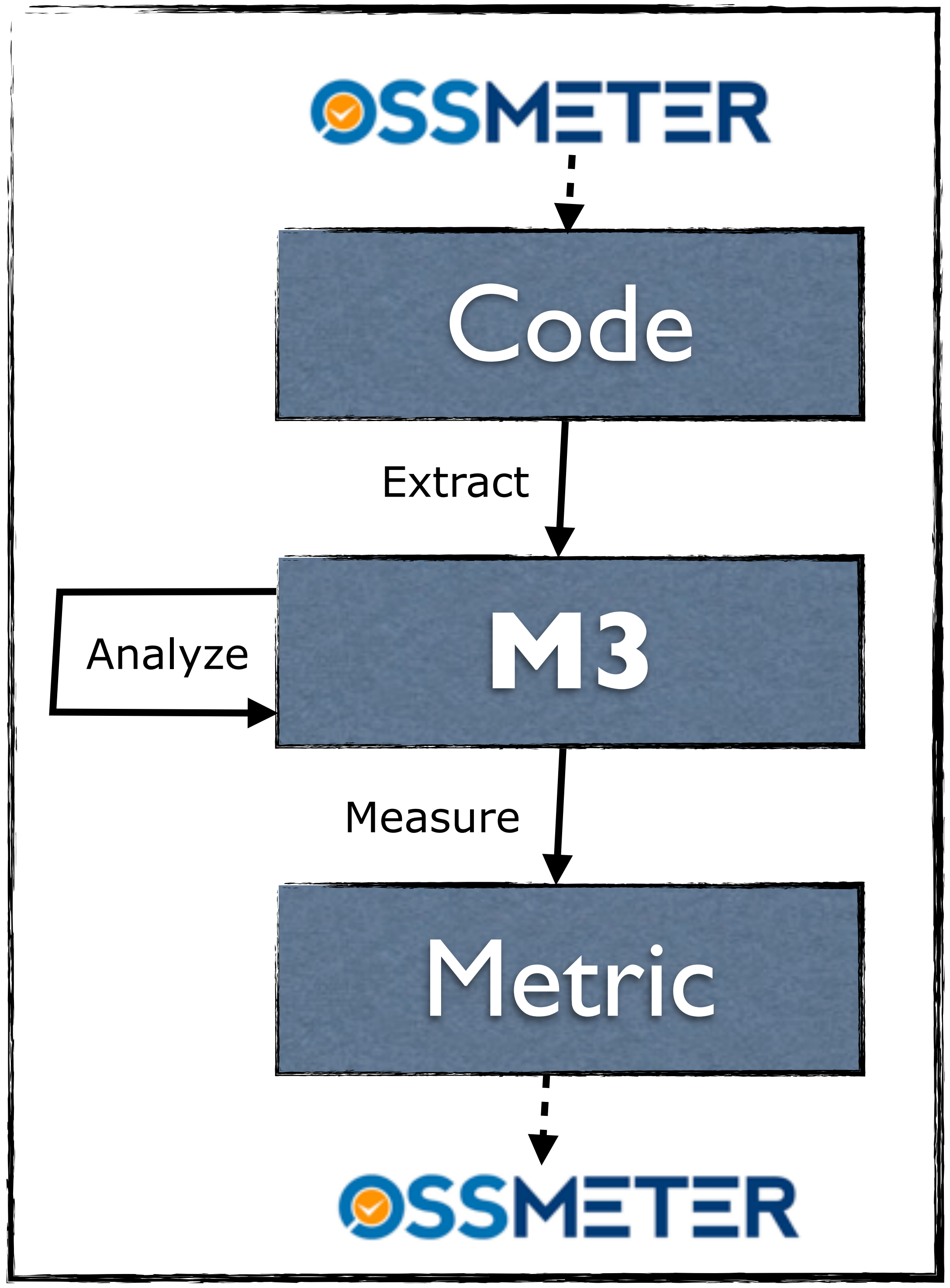
- Language-parametrized meta-model for source code metrics on syntax and semantics



[inspired by FAMIX, SOUL, and others]

M3

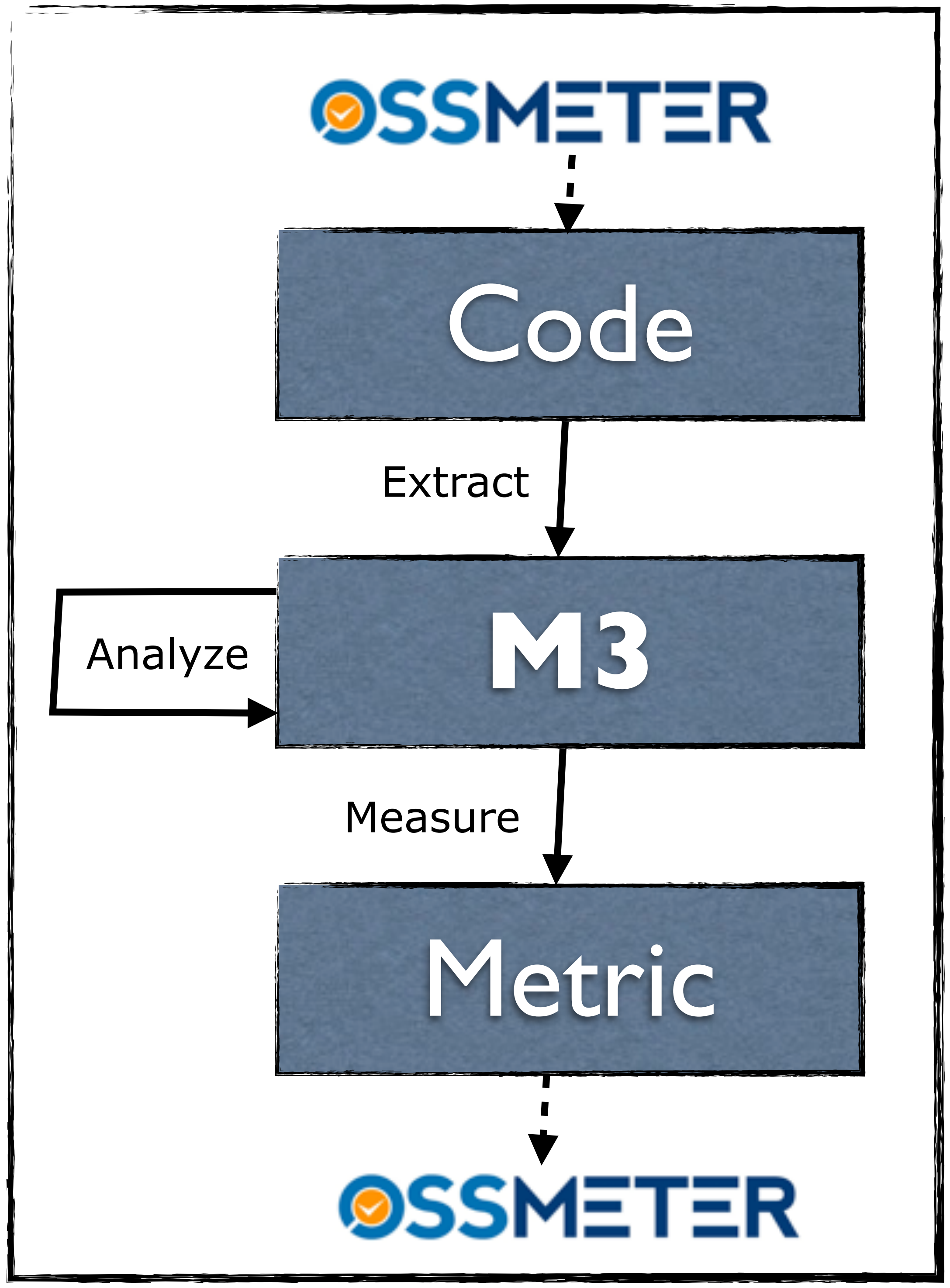
- Language-parametrized meta-model for source code metrics on syntax and semantics
- General & simple



[inspired by FAMIX, SOUL, and others]

M3

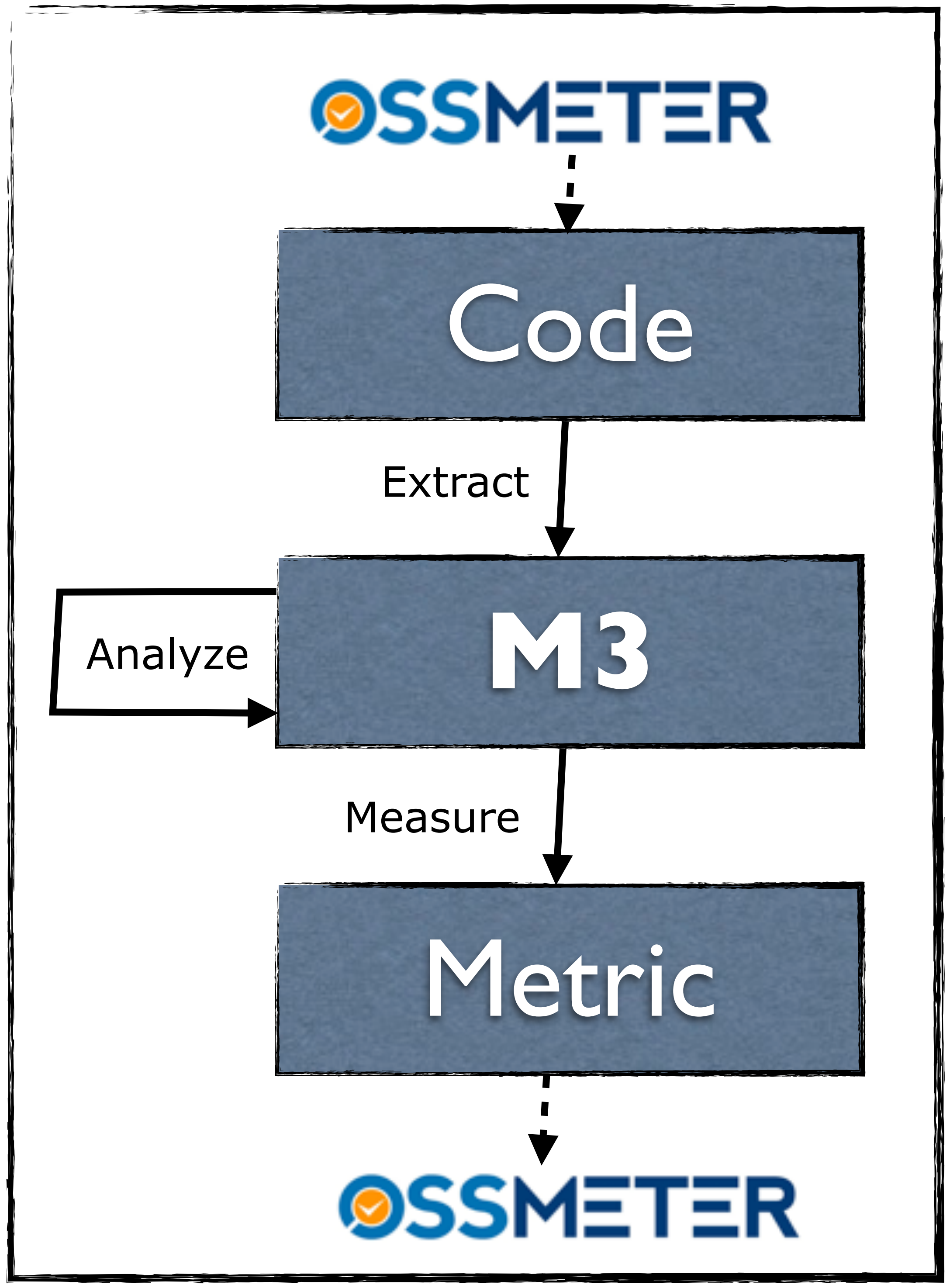
- Language-<u>parametrized</u> meta-model for source code metrics on syntax and semantics
- General & simple
- Formal



[inspired by FAMIX, SOUL, and others]

M3

- Language-parametrized meta-model for source code metrics on syntax and semantics
- General & simple
- Formal
- Detailed



[inspired by FAMIX, SOUL, and others]

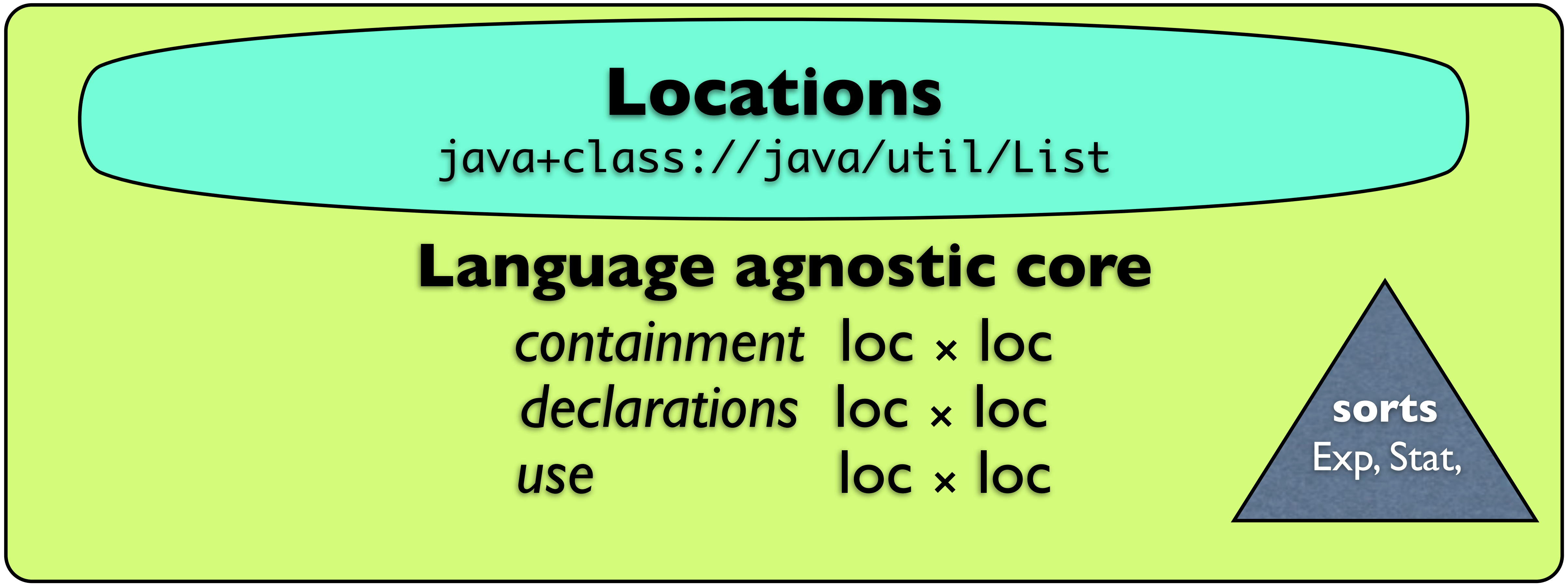
M3 = URI + Relations + Trees

M3 = URI + Relations + Trees

Locations

`java+class://java/util/List`

M3 = URI + Relations + Trees



M3 = URI + Relations + Trees

Locations

`java+class://java/util/List`

Language agnostic core

containment	loc × loc
declarations	loc × loc
use	loc × loc

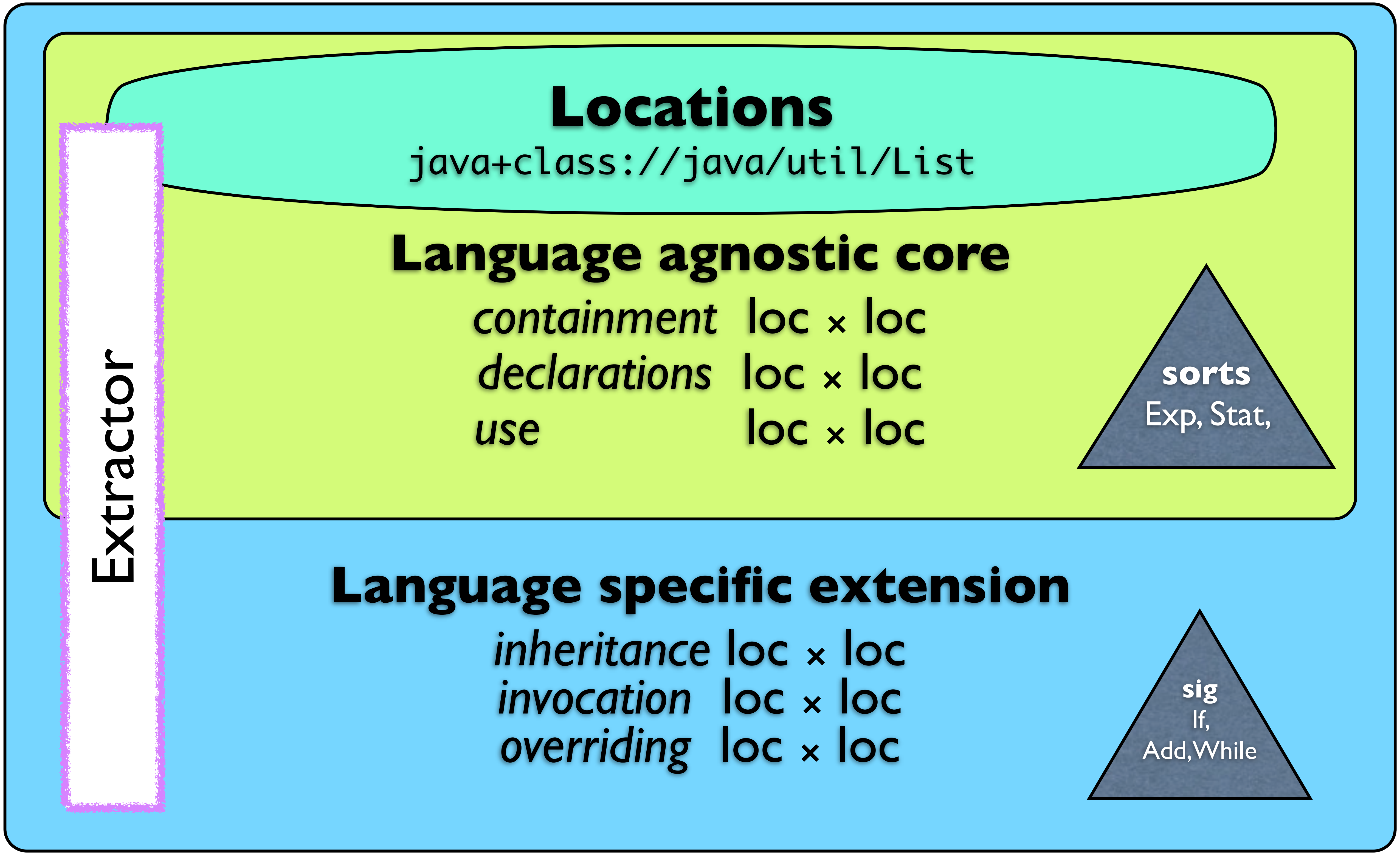
sorts
Exp, Stat,

Language specific extension

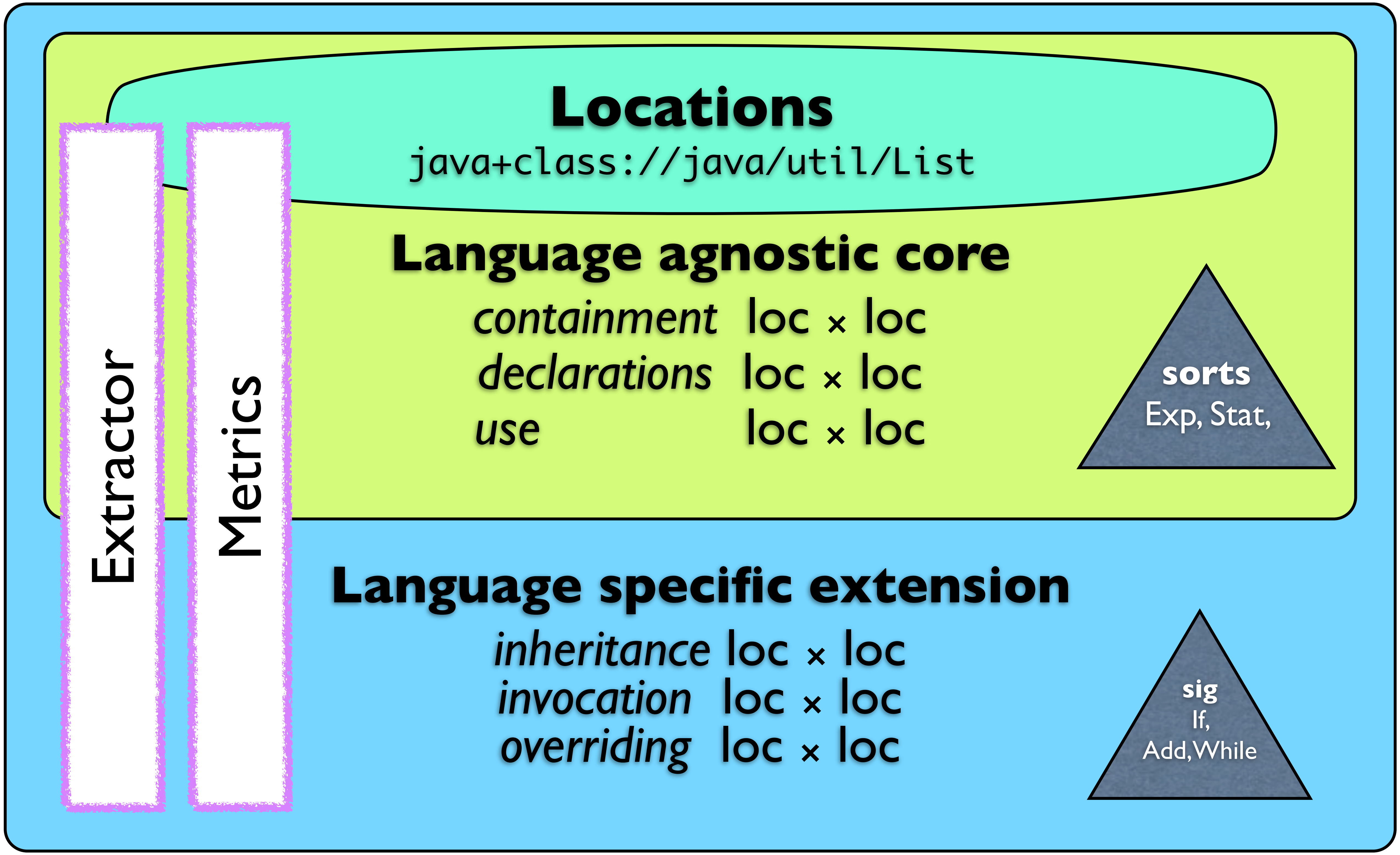
inheritance	loc × loc
invocation	loc × loc
overriding	loc × loc

sig
If,
Add, While

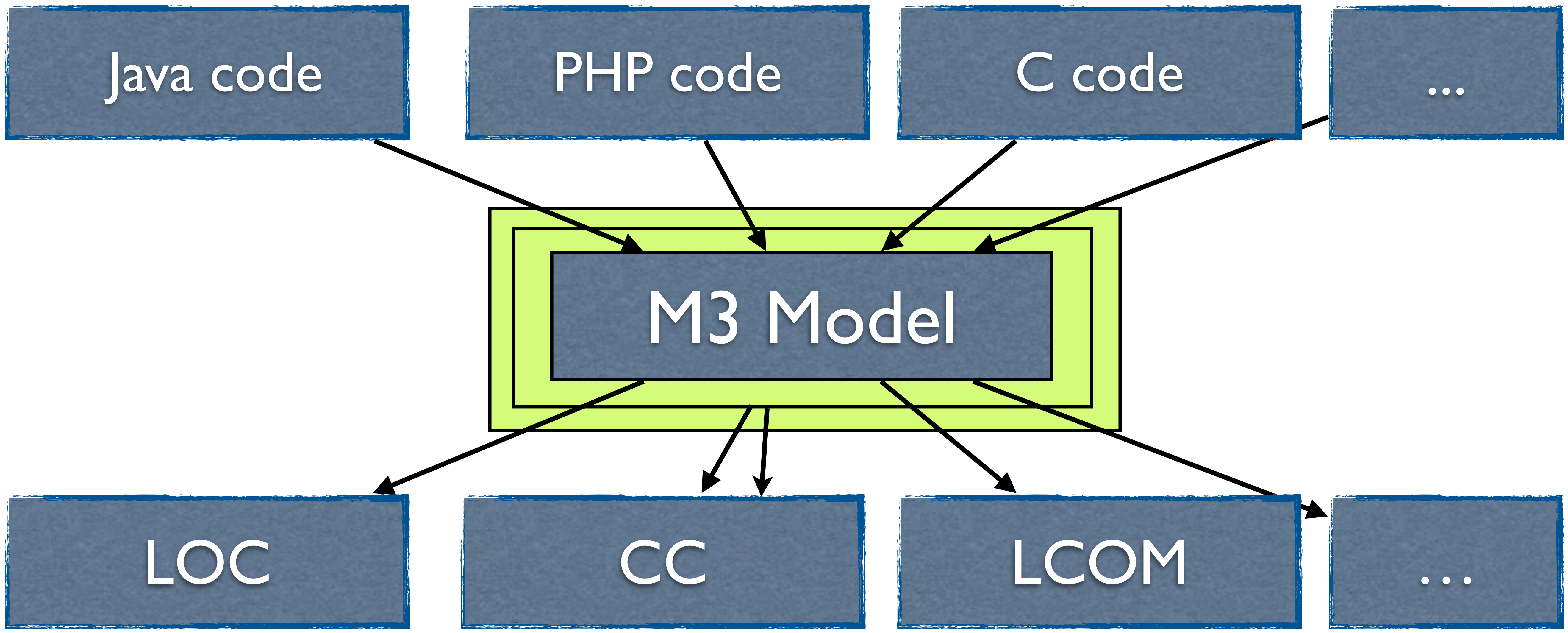
$M3 = URI + Relations + Trees$



M3 = URI + Relations + Trees



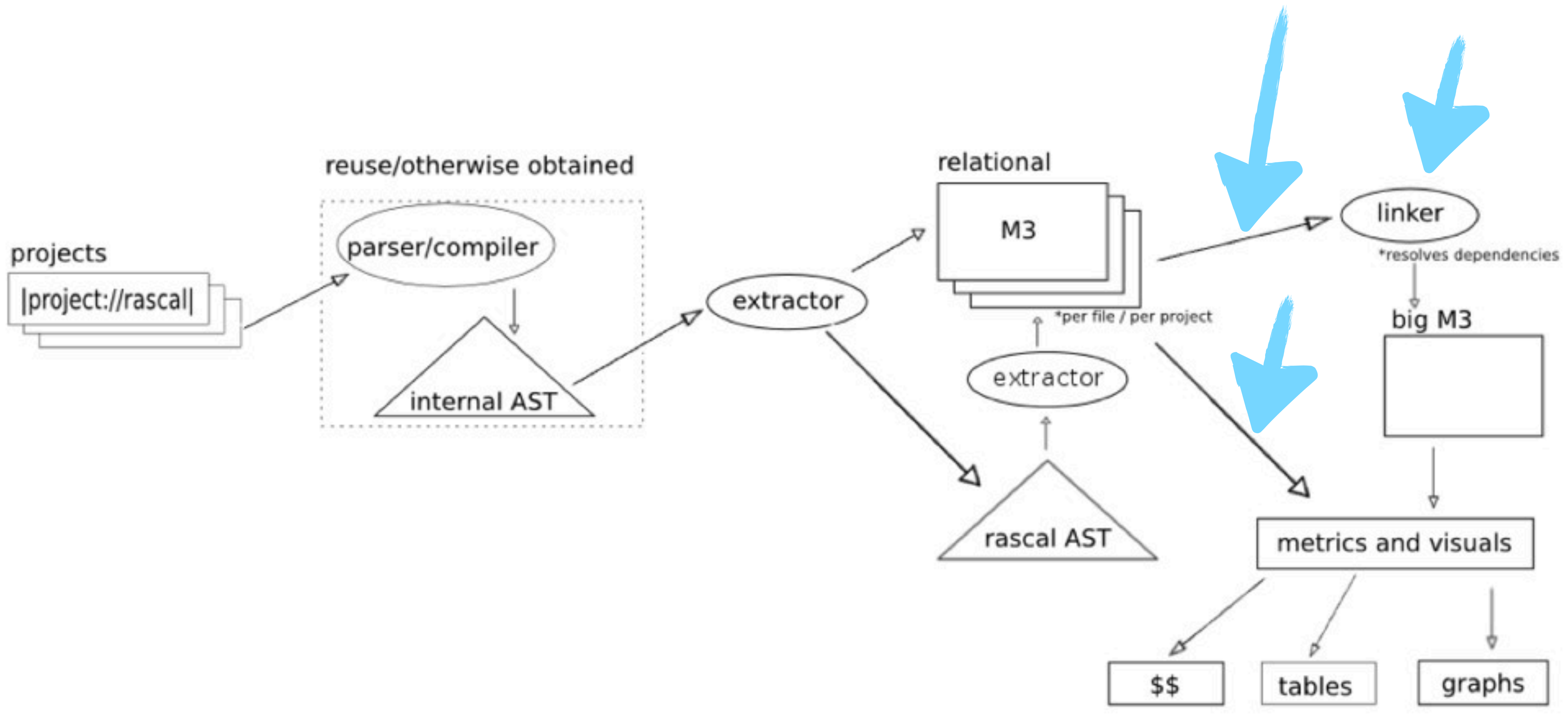
Managing variety by uniformity



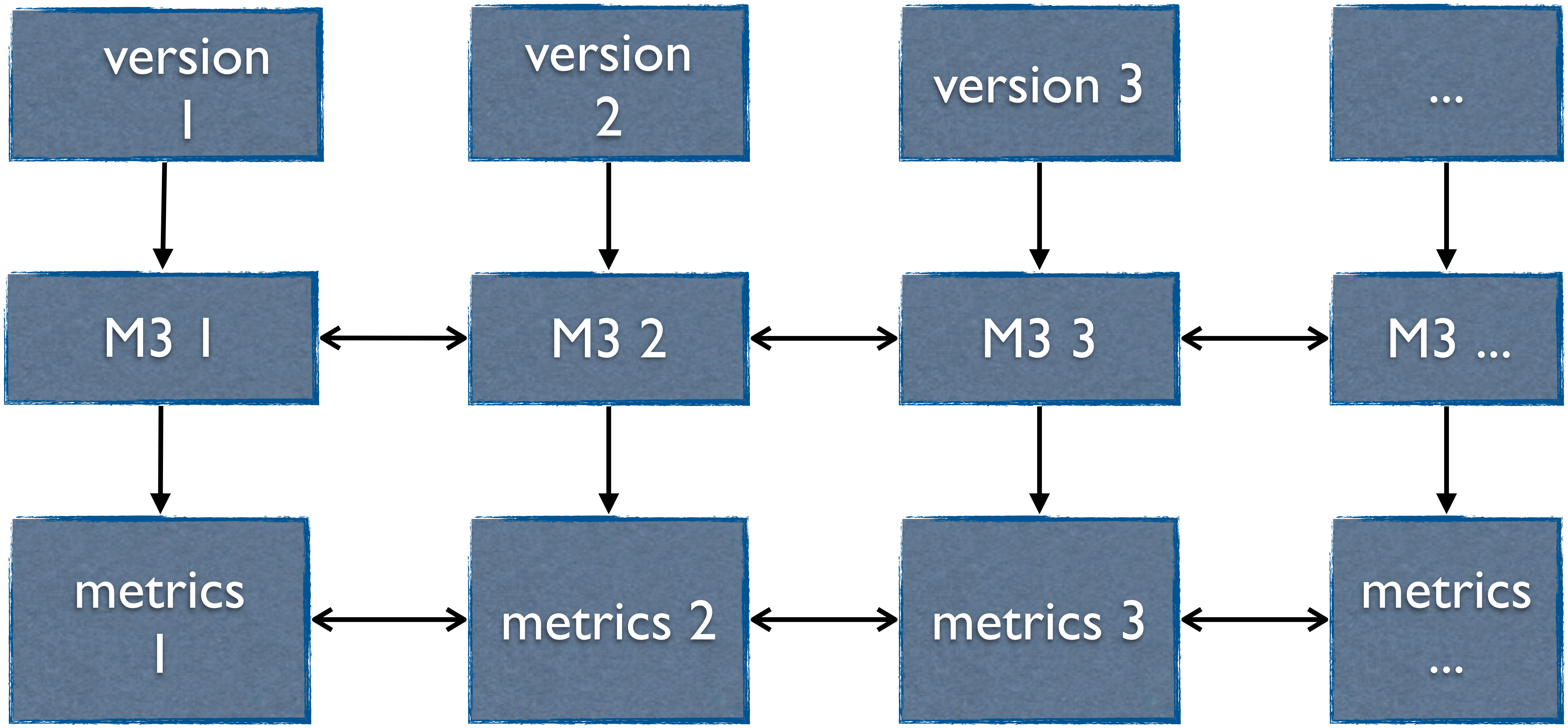
Caveat: details matter!

Uniformity helps reuse but does not guarantee it

M3 context

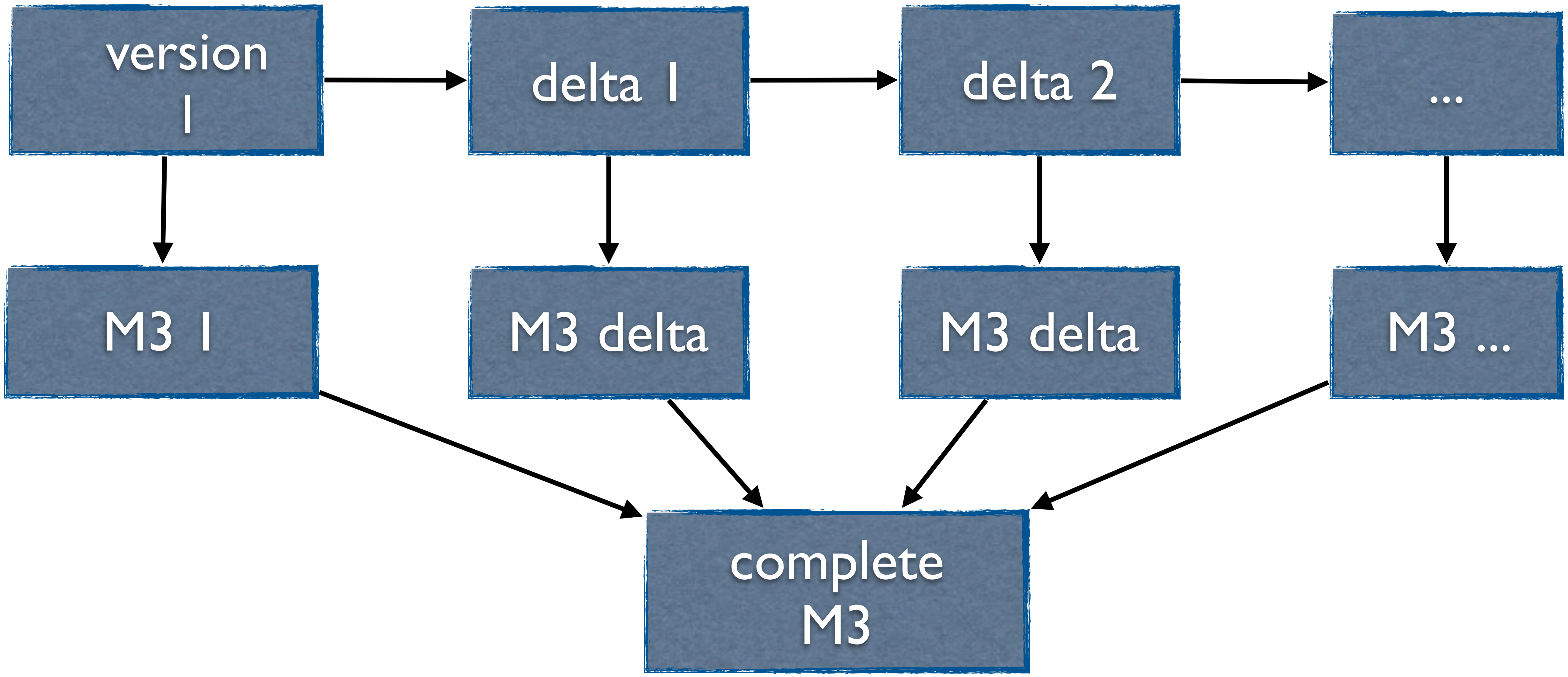

projects
|project://rascal|
reuse/otherwise obtained
parser/compiler
internal AST
extractor
relational
M3
*per file / per project
extractor
rascal AST
linker
*resolves dependencies
big M3
metrics and visuals
$$
tables
graphs

Representing evolution

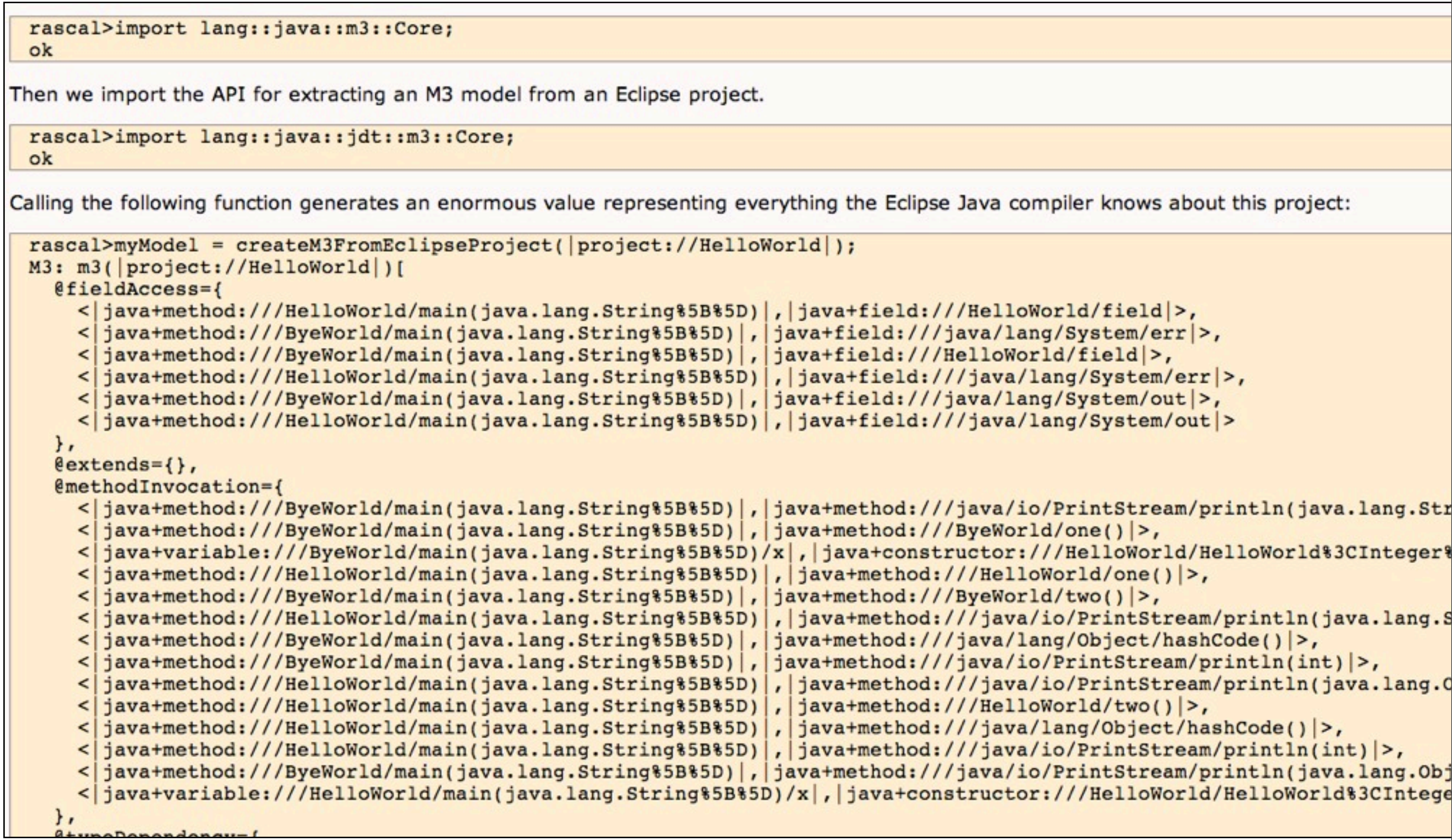


M3 enables comparability and safe composition
measure churn, growth, spikes, degradation, etc.

Incremental model extraction



Key design elements



```
rascal>import lang::java::m3::Core;
ok
```

Then we import the API for extracting an M3 model from an Eclipse project.

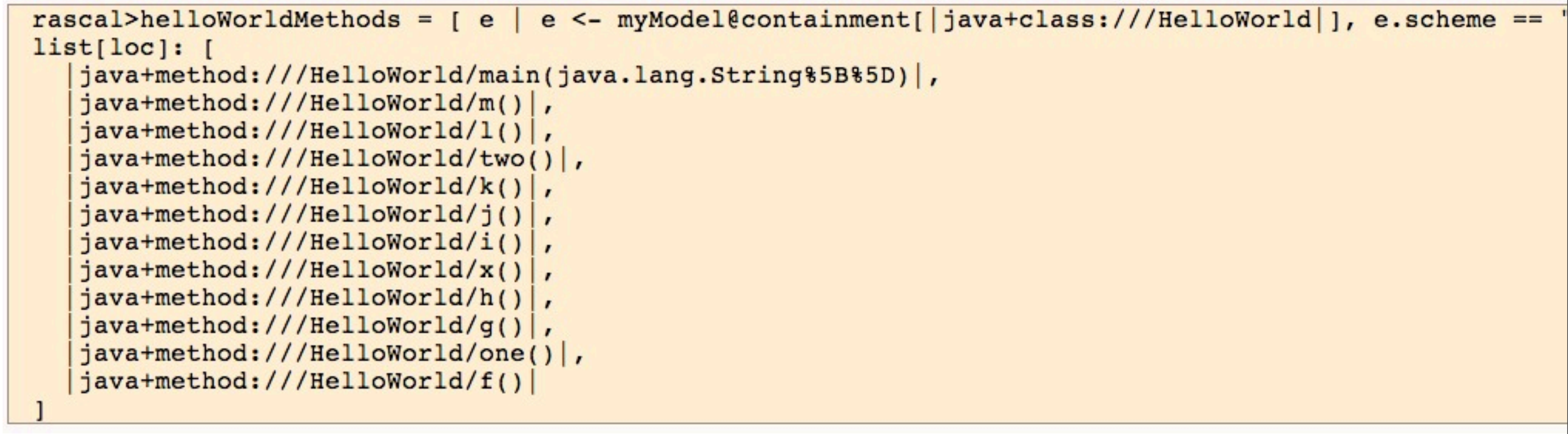
```
rascal>import lang::java::jdt::m3::Core;
ok
```

Calling the following function generates an enormous value representing everything the Eclipse Java compiler knows about this project:

```
rascal>myModel = createM3FromEclipseProject(|project://HelloWorld|);
M3: m3(|project://HelloWorld|)[
  @fieldAccess={
    <|java+method:///HelloWorld/main(java.lang.String%5B%5D)|,|java+field:///HelloWorld/field|>,
    <|java+method:///ByeWorld/main(java.lang.String%5B%5D)|,|java+field:///java/lang/System/err|>,
    <|java+method:///ByeWorld/main(java.lang.String%5B%5D)|,|java+field:///HelloWorld/field|>,
    <|java+method:///HelloWorld/main(java.lang.String%5B%5D)|,|java+field:///java/lang/System/err|>,
    <|java+method:///ByeWorld/main(java.lang.String%5B%5D)|,|java+field:///java/lang/System/out|>,
    <|java+method:///HelloWorld/main(java.lang.String%5B%5D)|,|java+field:///java/lang/System/out|>
  },
  @extends={},
  @methodInvocation={
    <|java+method:///ByeWorld/main(java.lang.String%5B%5D)|,|java+method:///java/io/PrintStream/println(java.lang.Str
    <|java+method:///ByeWorld/main(java.lang.String%5B%5D)|,|java+method:///ByeWorld/one()|>,
    <|java+variable:///ByeWorld/main(java.lang.String%5B%5D)/x|,|java+constructor:///HelloWorld/HelloWorld%3CInteger%
    <|java+method:///HelloWorld/main(java.lang.String%5B%5D)|,|java+method:///HelloWorld/one()|>,
    <|java+method:///ByeWorld/main(java.lang.String%5B%5D)|,|java+method:///ByeWorld/two()|>,
    <|java+method:///HelloWorld/main(java.lang.String%5B%5D)|,|java+method:///java/io/PrintStream/println(java.lang.S
    <|java+method:///ByeWorld/main(java.lang.String%5B%5D)|,|java+method:///java/lang/Object/hashCode()|>,
    <|java+method:///ByeWorld/main(java.lang.String%5B%5D)|,|java+method:///java/io/PrintStream/println(int)|>,
    <|java+method:///HelloWorld/main(java.lang.String%5B%5D)|,|java+method:///java/io/PrintStream/println(java.lang.O
    <|java+method:///HelloWorld/main(java.lang.String%5B%5D)|,|java+method:///HelloWorld/two()|>,
    <|java+method:///HelloWorld/main(java.lang.String%5B%5D)|,|java+method:///java/lang/Object/hashCode()|>,
    <|java+method:///HelloWorld/main(java.lang.String%5B%5D)|,|java+method:///java/io/PrintStream/println(int)|>,
    <|java+method:///ByeWorld/main(java.lang.String%5B%5D)|,|java+method:///java/io/PrintStream/println(java.lang.Obj
    <|java+variable:///HelloWorld/main(java.lang.String%5B%5D)/x|,|java+constructor:///HelloWorld/HelloWorld%3CIntege
  },
```

Textual values

Key design elements



```
rascal>helloWorldMethods = [ e | e <- myModel@containment[|java+class:///HelloWorld|], e.scheme == '
list[loc]: [
  |java+method:///HelloWorld/main(java.lang.String%5B%5D)|,
  |java+method:///HelloWorld/m()|,
  |java+method:///HelloWorld/l()|,
  |java+method:///HelloWorld/two()|,
  |java+method:///HelloWorld/k()|,
  |java+method:///HelloWorld/j()|,
  |java+method:///HelloWorld/i()|,
  |java+method:///HelloWorld/x()|,
  |java+method:///HelloWorld/h()|,
  |java+method:///HelloWorld/g()|,
  |java+method:///HelloWorld/one()|,
  |java+method:///HelloWorld/f()|
]
```

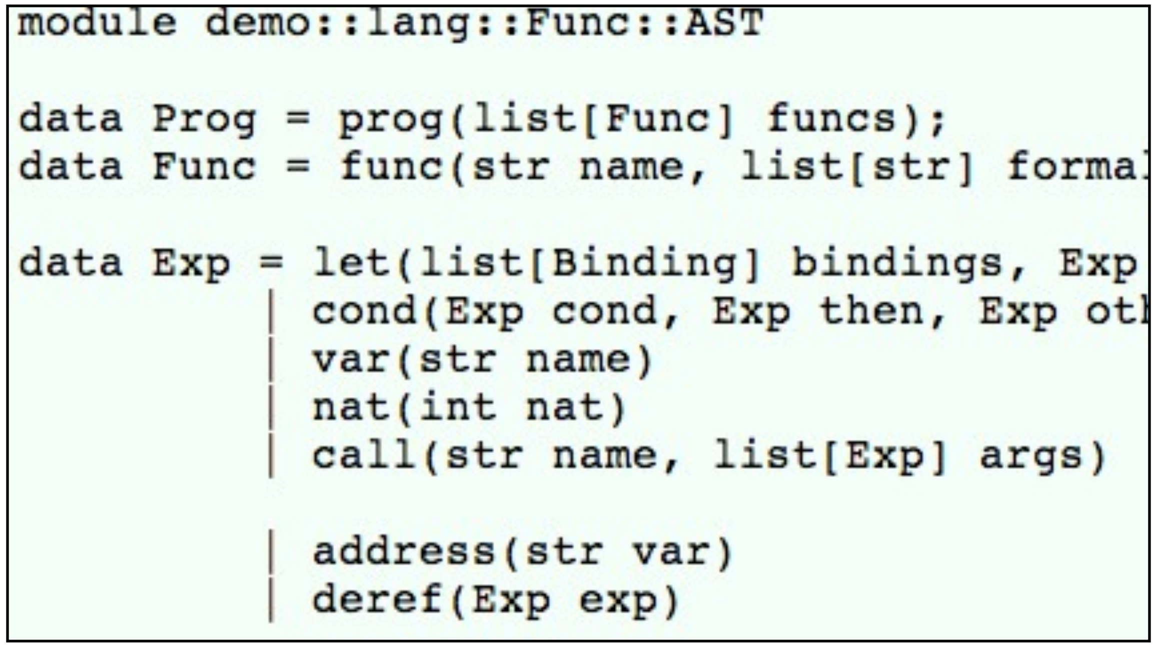
URI = Qualified Names and Hyperlinks

Key design elements

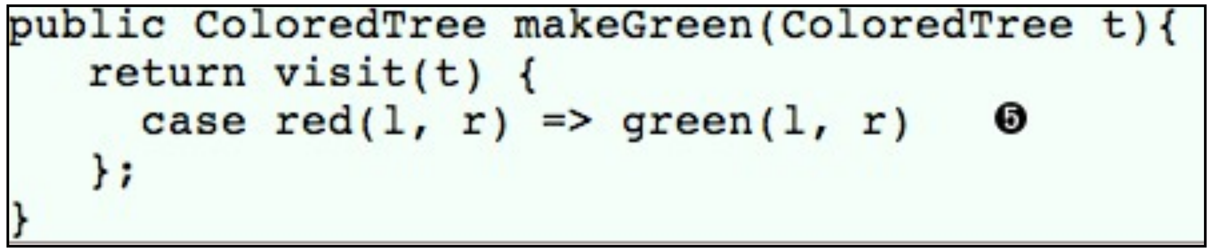
```
reachable = m@containment+

m@invocations += m@invocations o m@overrides

subtypes = m@extends + m@implements
```

Relations = query and elaborate

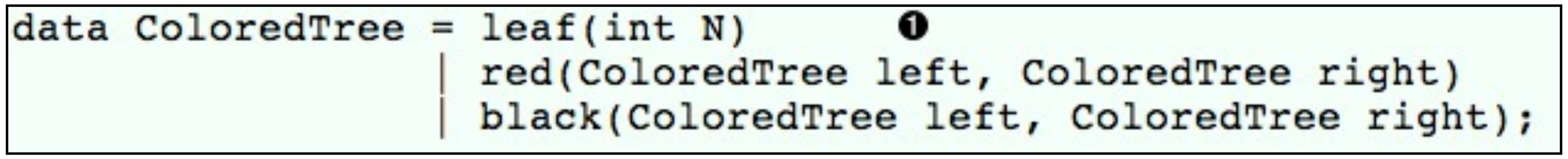
Key design elements



```
module demo::lang::Func::AST

data Prog = prog(list[Func] funcs);
data Func = func(str name, list[str] forma

data Exp = let(list[Binding] bindings, Exp
         | cond(Exp cond, Exp then, Exp ot
         | var(str name)
         | nat(int nat)
         | call(str name, list[Exp] args)

         | address(str var)
         | deref(Exp exp)
```



```
public ColoredTree makeGreen(ColoredTree t){
   return visit(t) {
     case red(l, r) => green(l, r)    ❺
   };
}
```



```
data ColoredTree = leaf(int N)        ❶
                 | red(ColoredTree left, ColoredTree right)
                 | black(ColoredTree left, ColoredTree right);
```

algebraic data-types are great for ASTs

[pattern matching for open dispatch: term rewriting "inside"]

Rascal/M3... the point?

- Rascal - CWI SWAT's lab
- OSSMETER - open source quality platform
- Diversity
  - Lots of languages
  - Lots of metrics
- Simple, extensible model, immutable/formal
- Caveat emptor & feedback/use welcome