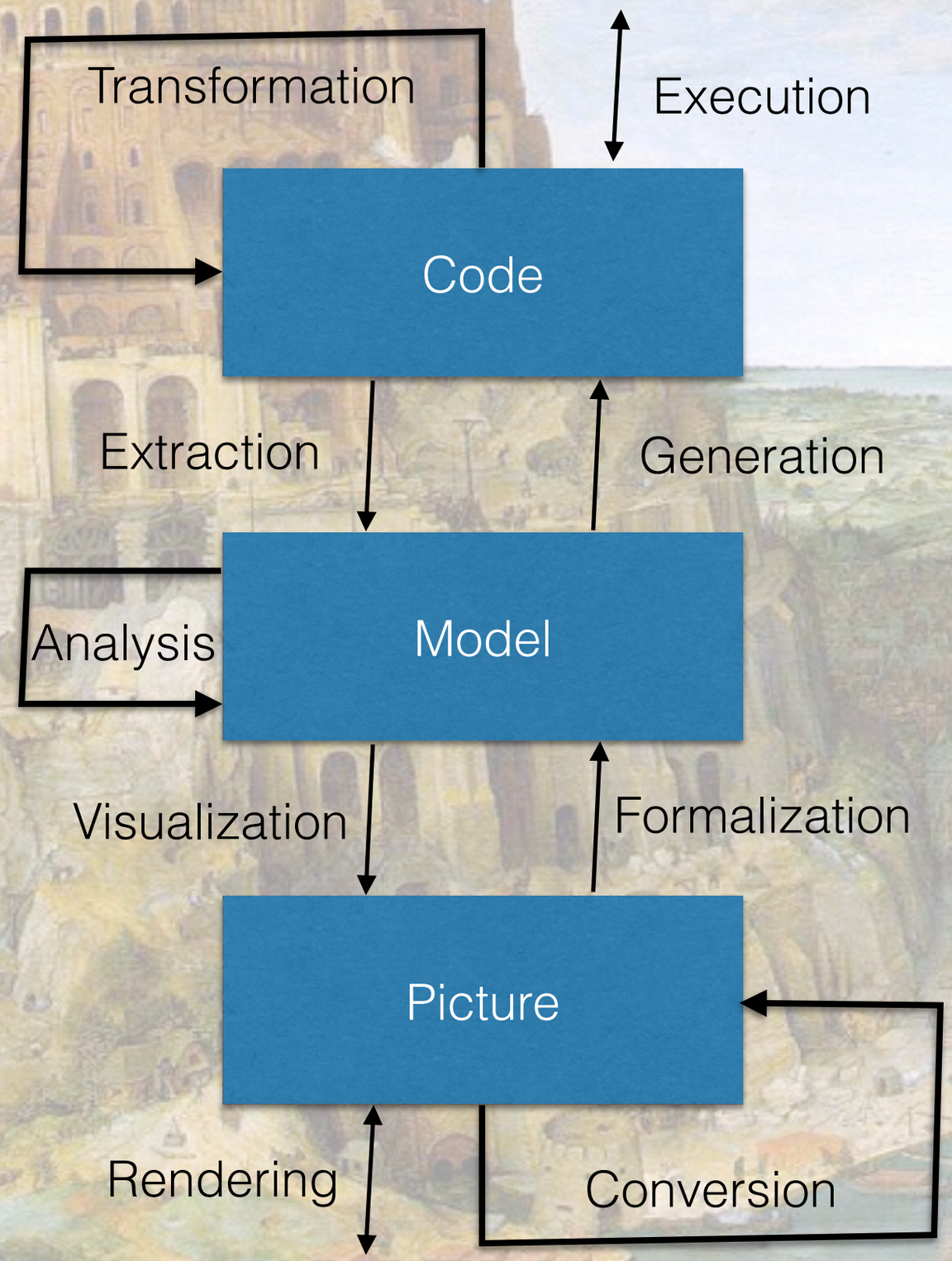
Tips & Tricks for TUE students doing Architecture Reconstruction with Rascal

Jurgen Vinju
Davy Landman

https://gist.github.com/jurgenvinju/8972255
http://update.rascal-mpl.org/{un,}stable

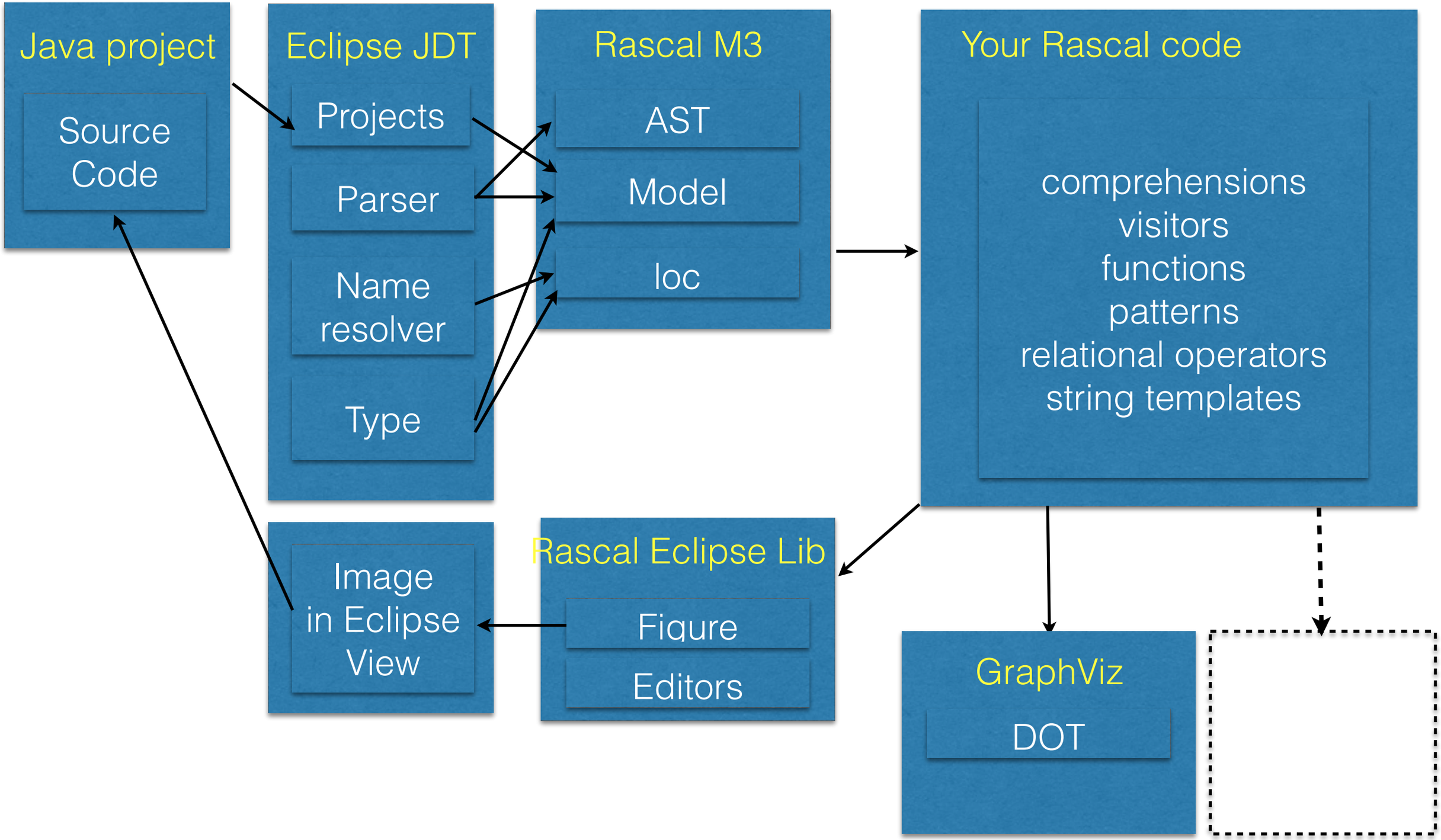
See peach assignment 0 for install instruction
And assignment 2 and Tonella & Potrich

Rascal is a DSL for **meta programming**

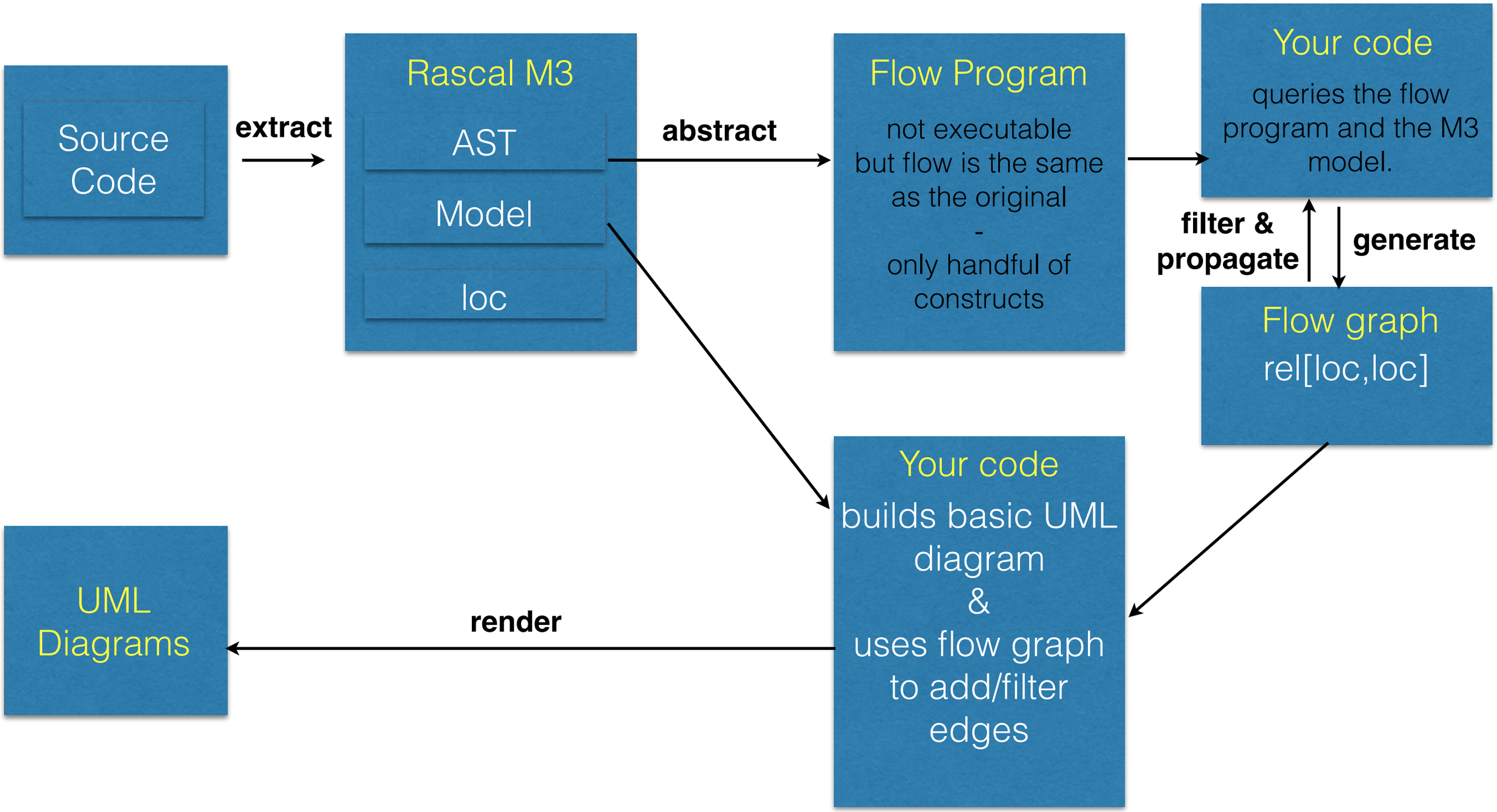


(Brueghel, Tower of Babel)

Extract Analyze SYnthesize



UML & Object Flow



Identifying code

- Source code locations: loc type
  - `|project://MyProj/src/org/myproj/Fruit.java|`
- Structured names: loc type
  - |java+class://java/util/List|
  - |java+class://java/util/List| + “this” == |java+class://java/util/List/this|
- Back to code
  - hyperlinks (if you extracted a full M3 model)
  - IO::readFile(loc) produces a str with the contents

Getting facts

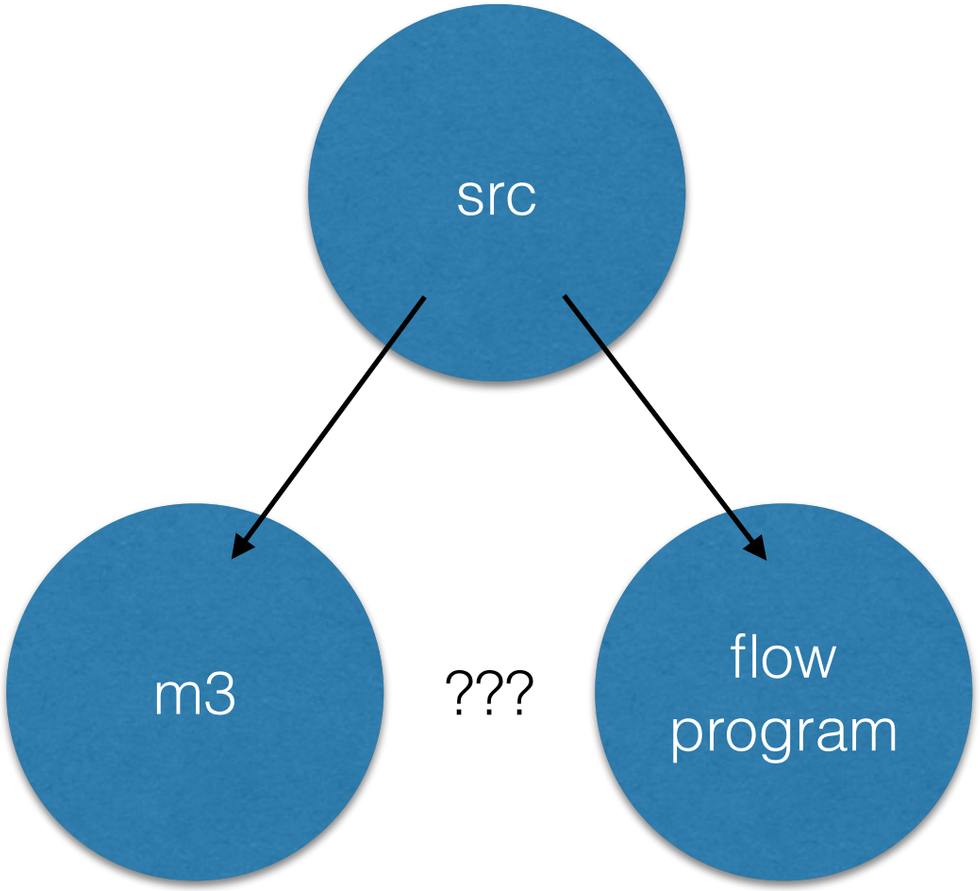
- First have a compileable Eclipse project “eLib”
- import lang::java::jdt::m3::Core; import lang::java::jdt::m3::AST;
- m = createM3FromEclipseProject(|project://eLib|);
- import lang::ofg::ast::Java2OFG; import lang::ofg::ast::FlowLanguage;
- p = createOFG(|project://eLib|);
- Now you have:
  - an M3 model in m
  - a flow program in p
  - hyperlinks

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

rascal>import lang::ofg::ast::Java2OFG;
ok

rascal>m = createM3FromEclipseProject(|project://eLib|)
>>>>>>>;
M3: m3(|project://eLib|)[
  @fieldAccess={
    <|java+constructor:///User/User(java.lang.String,java.lang.String,java.lang.String)|,|java+field:///User/fullName|>,
    <|java+variable:///Library/printAllLoans()/i|,|java+field:///Library/loans|>,
    <|java+constructor:///User/User(java.lang.String,java.lang.String,java.lang.String)|,|java+field:///User/userCode|>,
    <|java+variable:///Library/searchDocumentByTitle(java.lang.String)/i|,|java+field:///Library/documents|>,
    <|java+method:///Main/rmUser(java.lang.String)|,|java+field:///Main/lib|>,
```

```
rascal>p = createOFG(|project://eLib|);
Getting decls
Getting stms
Program: program(
  {
    attribute(|java+field:///InternalUser/internalId|),
    method(
      |java+method:///Main/addBook(java.lang.String)|,
      [|java+parameter:///Main/addBook(java.lang.String)/cmd|]),
    method(
      |java+method:///User/authorizedUser()|,
      [])
```



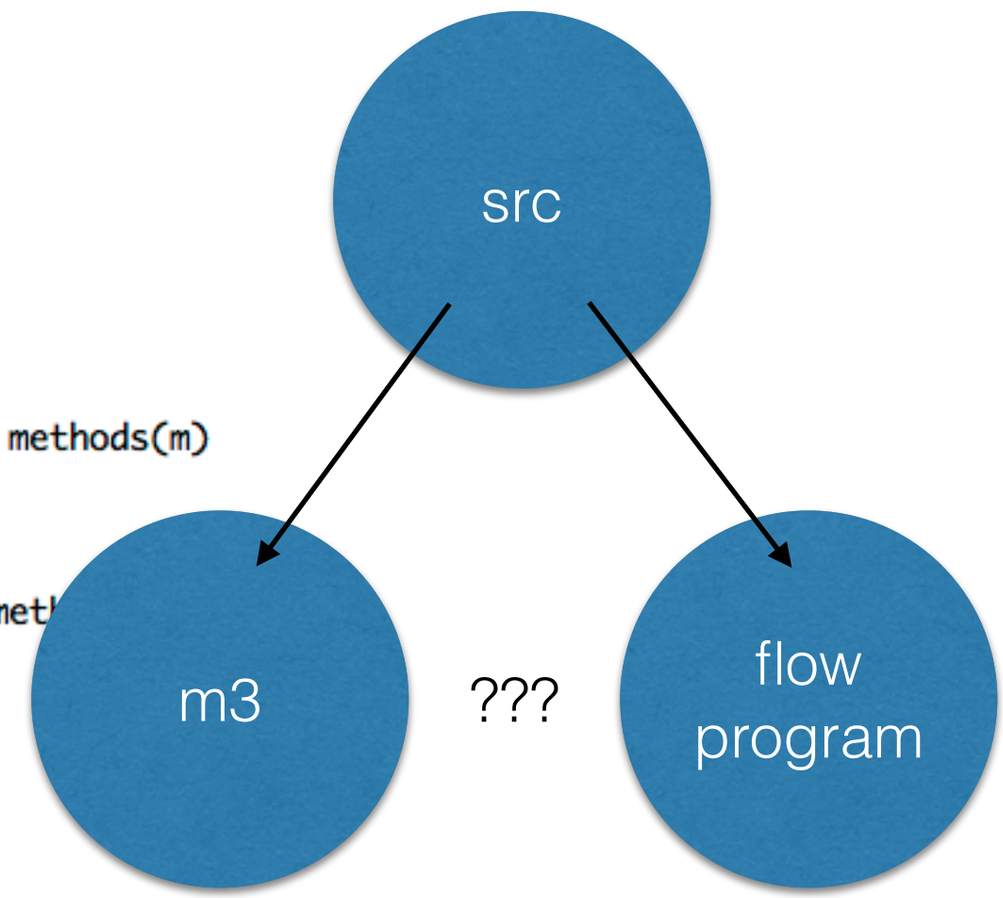
```
rascal>methods(m)
set[loc]: {
  |java+method:///User/getAddress()|,
  |java+method:///TechnicalReport/authorizedLoan(User)|,
  |java+method:///User/getName()|,
  |java+method:///Main/rmUser(java.lang.String)|,
  |java+method:///Main/getArgs(java.lang.String)|,
  |java+method:///Journal/authorizedLoan(User)|,
  |java+constructor:///TechnicalReport/TechnicalReport(java.lang.String,java.lang.String,java.lang.String)|,
  |java+method:///Document/printAvailability()|,
```

```
rascal>p.decls
set[Decl]: {
  attribute(|java+field:///InternalUser/internalId|),
  method(
    |java+method:///Main/addBook(java.lang.String)|,
    [|java+parameter:///Main/addBook(java.lang.String)/cmd|]),
```

```
rascal>{l | method(l, _) <- p.decls} == methods(m)
bool: false

rascal>{l | method(l, _) <- p.decls} + {l | constructor(l,_) <- p.decls} == methods(m)
bool: false

rascal>{l | method(l, _) <- p.decls} + {l | constructor(l,_) <- p.decls} - meth
set[loc]: {
  |java+constructor:///Library/Library()|,
  |java+constructor:///Main/Main()|
}
```



Initial flow graph

```
module Hallo

import lang::ofg::ast::FlowLanguage;
import lang::ofg::ast::Java2OFG;
import List;
import Relation;

alias OFG = rel[loc from, loc to];

OFG buildGraph(Program p)
  = { <as[i], fps[i]> | newAssign(x, cl, c, as) <- p.statemens, constructor(c, fps) <- p.decls, i <- index(as) }
  + { <cl + "this", x> | newAssign(x, cl, _, _) <- p.statemens }
  + { <y, x> | assign(x, _, y) <- p.statemens}
```

- a relation as a graph
- uses generators and matching to project out of program
- uses “+” on locs to build new implicit identifiers
- uses “index” from List to pair formal/actual parameters

Flow propagation sketch

```
OFG prop(OFG g, rel[loc,loc] gen, rel[loc,loc] kill, bool back) {
  OFG IN = { };
  OFG OUT = gen + (IN - kill);
  gi = g<to,from>;
  set[loc] pred(loc n) = gi[n];
  set[loc] succ(loc n) = g[n];

  solve (IN, OUT) {
    IN = { <n,\o> | n <- carrier(g), p <- (back ? pred(n) : succ(n)), \o <- OUT[p] };
    OUT = gen + (IN - kill);
  }

  return OUT;
}
```

- use “solve” for fixed point
- projection “g[n]” is succ, revert and project “g<to,from>[n]” i
  pred
- comprehensions for next solution

Debugging

- IO::println, iprintln,
- util::ValueUI::text,tree,graph
- interactive debugger (just put a breakpoint)
- online manual?! http://tutor.rascal-mpl.org, also in Eclipse Tutor View
- online questions?! http://ask.rascal-mpl.org
- Issue tracker at github.org

Caveats

- No type checker (coming soon)
- No incremental parsing (one error at a time)
- Slowness (compiler coming soon)
- But: consider doing this from scratch :-)

Demo

- Inheritance diagram from Eclipse Java project
- Get code into an Eclipse Java project
- Start a Rascal project
- Start a Rascal console
- Browse library code and tutorial
- Script the tool
  - https://gist.github.com/jurgenvinju/4999479