CWI

CWI Lectures

Understanding and Simplifying Software

with the Rascal Meta-Programming Language

Jurgen J. Vinju

CWI

Software is Complex

CWI

Software is Complex

- The principles of software are easy

CWI

Software is Complex

- The principles of software are easy
  - Just a bunch of computer instructions

CWI

Software is Complex

- The principles of software are easy
  - Just a bunch of computer instructions
  - IO, arithmetic, control, done.

CWI

Software is Complex

- The principles of software are easy
  - Just a bunch of computer instructions
  - IO, arithmetic, control, done.
- The practice of software is incomprehensible

CWI

Software is Complex

- The principles of software are easy
  - Just a bunch of computer instructions
  - IO, arithmetic, control, done.
- The practice of software is incomprehensible
  - There is too much code

CWI

Software is Complex

- The principles of software are easy
  - Just a bunch of computer instructions
  - IO, arithmetic, control, done.
- The practice of software is incomprehensible
  - There is too much code
  - There is too much diversity

CWI

Software is Complex

- The principles of software are easy
  - Just a bunch of computer instructions
  - IO, arithmetic, control, done.
- The practice of software is incomprehensible
  - There is too much code
  - There is too much diversity
  - The CPU is too fast

CWI

Software is Complex

- The principles of software are easy
  - Just a bunch of computer instructions
  - IO, arithmetic, control, done.
- The practice of software is incomprehensible
  - There is too much code
  - There is too much diversity
  - The CPU is too fast
  - There is too much memory

CWI

Instruments

CWI

Instruments

- Higher level programming
  - Model Driven Engineering
  - Domain Specific Languages

CWI

Instruments

- Higher level programming
  - Model Driven Engineering
  - Domain Specific Languages
- Program comprehension
  - Static & dynamic analysis
  - IDEs

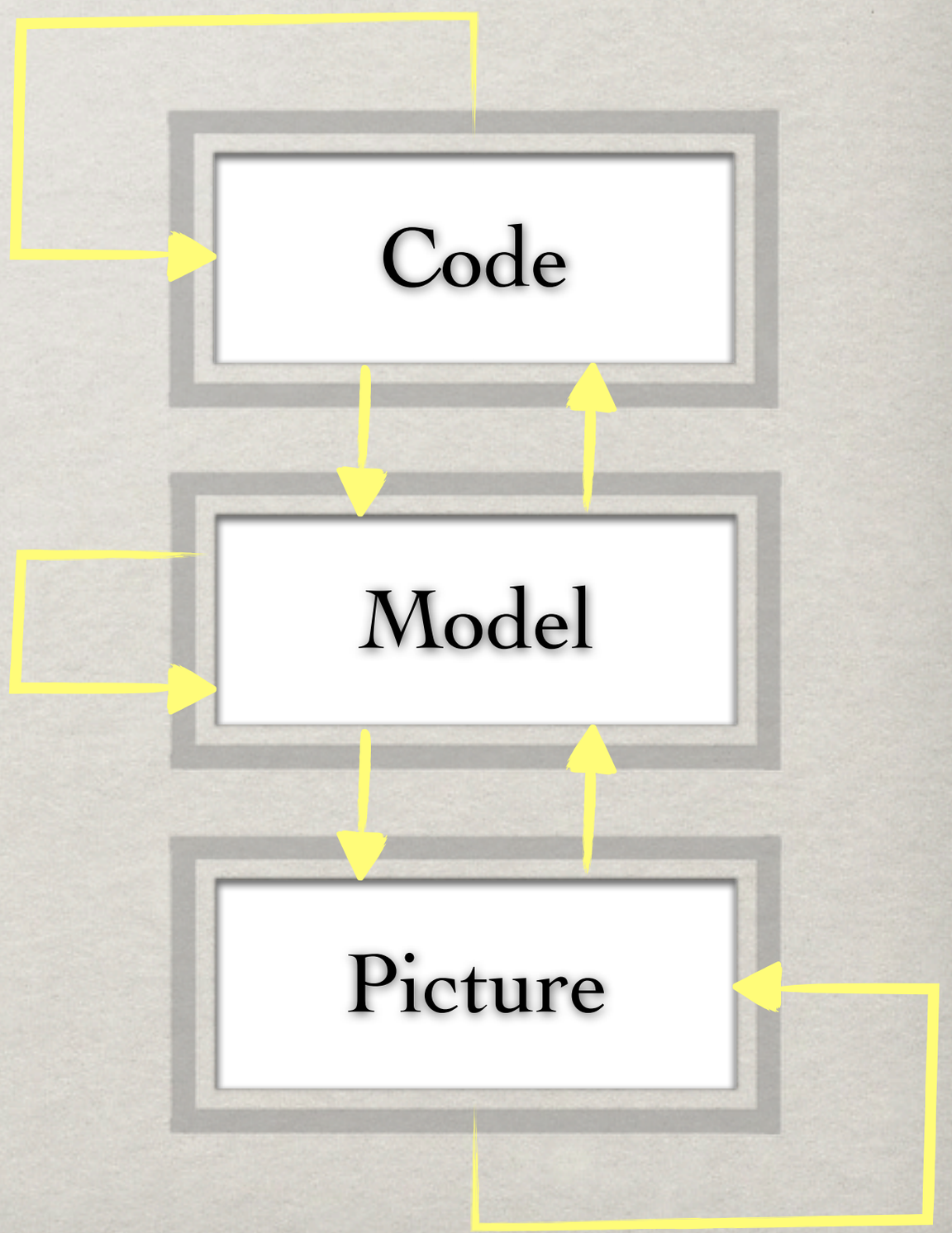
CWI

Instruments

- Higher level programming
  - Model Driven Engineering
  - Domain Specific Languages
- Program comprehension
  - Static & dynamic analysis
  - IDEs
- Observing Software
  - Mining Software Repositories
  - Measurement & benchmarking

Instruments

- Higher level programming
  - Model Driven Engineering
  - Domain Specific Languages
- Program comprehension
  - Static & dynamic analysis
  - IDEs
- Observing Software
  - Mining Software Repositories
  - Measurement & benchmarking



CWI

PROBLEM

CWI

PROBLEM

- Instruments are costly

CWI

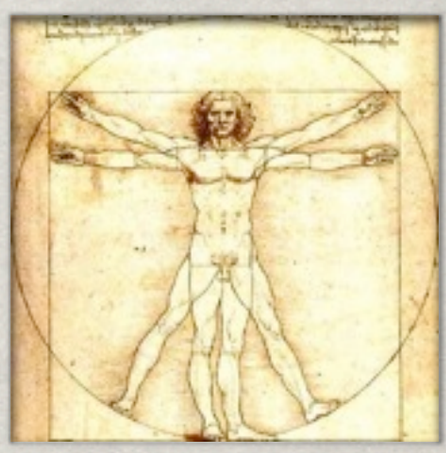
PROBLEM

- Instruments are costly
  - Diverse

CWI

PROBLEM

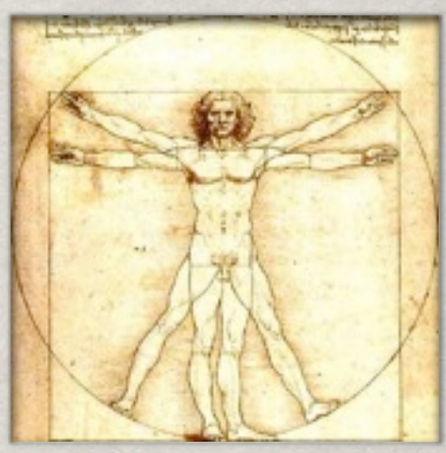
- Instruments are costly
  - Diverse
  - Multi-disciplinary



CWI

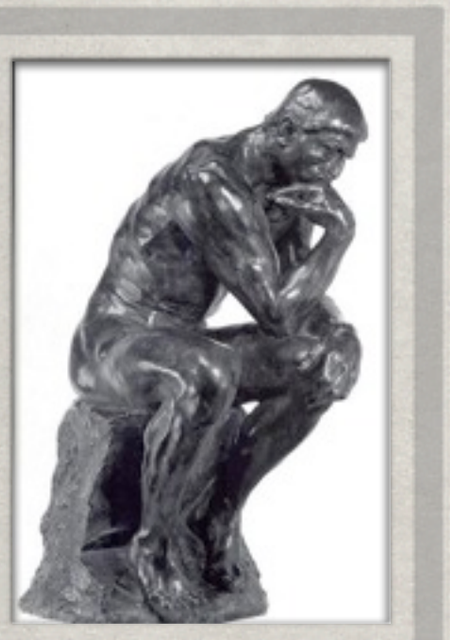
PROBLEM

- Instruments are costly
  - Diverse
  - Multi-disciplinary
  - Accurate/Efficient



Rascal

ENABLES



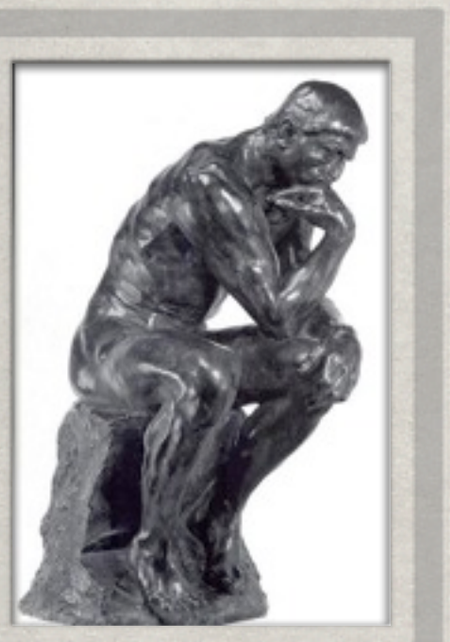
Research

Application

Rascal

A new tool every week

ENABLES



Research

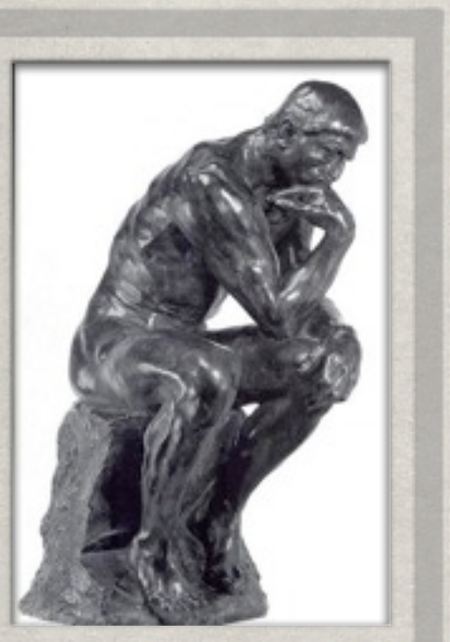
Application

Rascal

One-stop-shop

A new tool every week

ENABLES

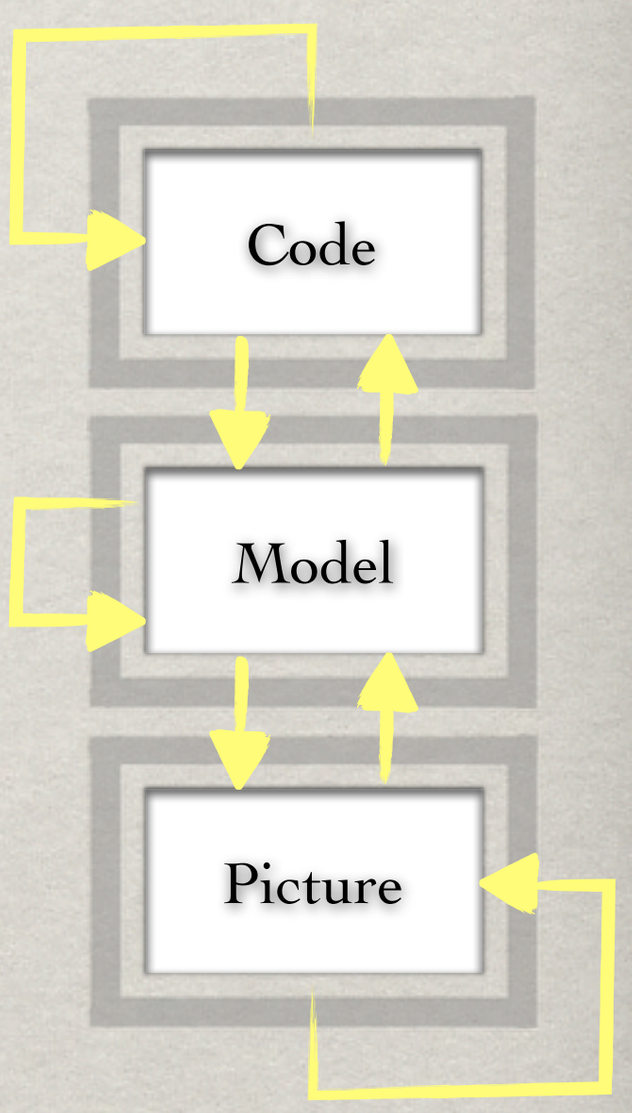


Research

Application

CWI

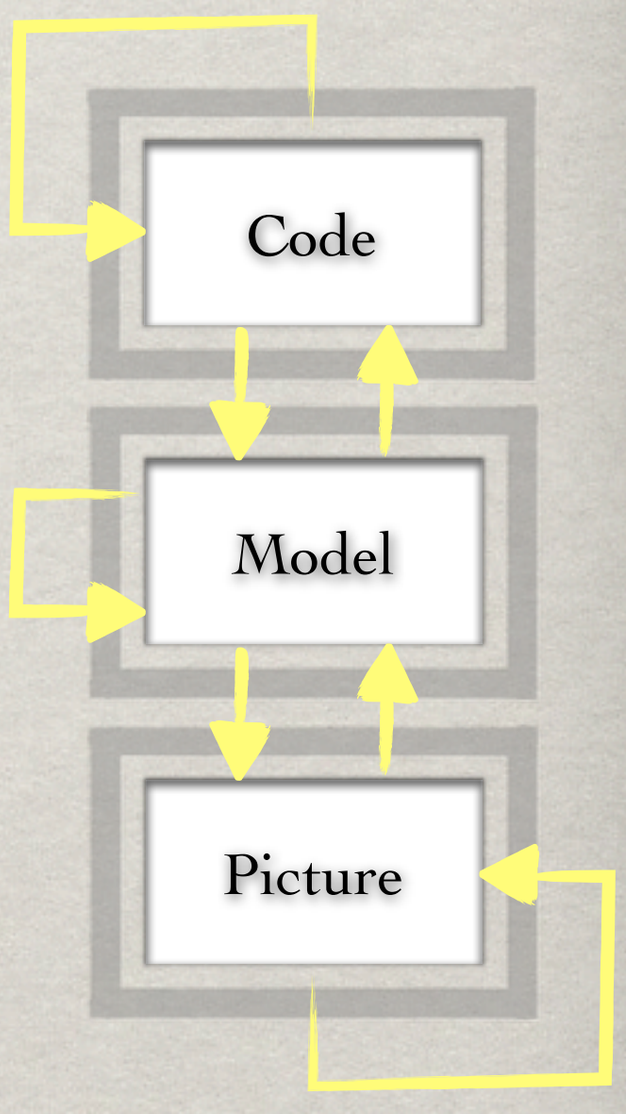
Rascal is EASY



CWI

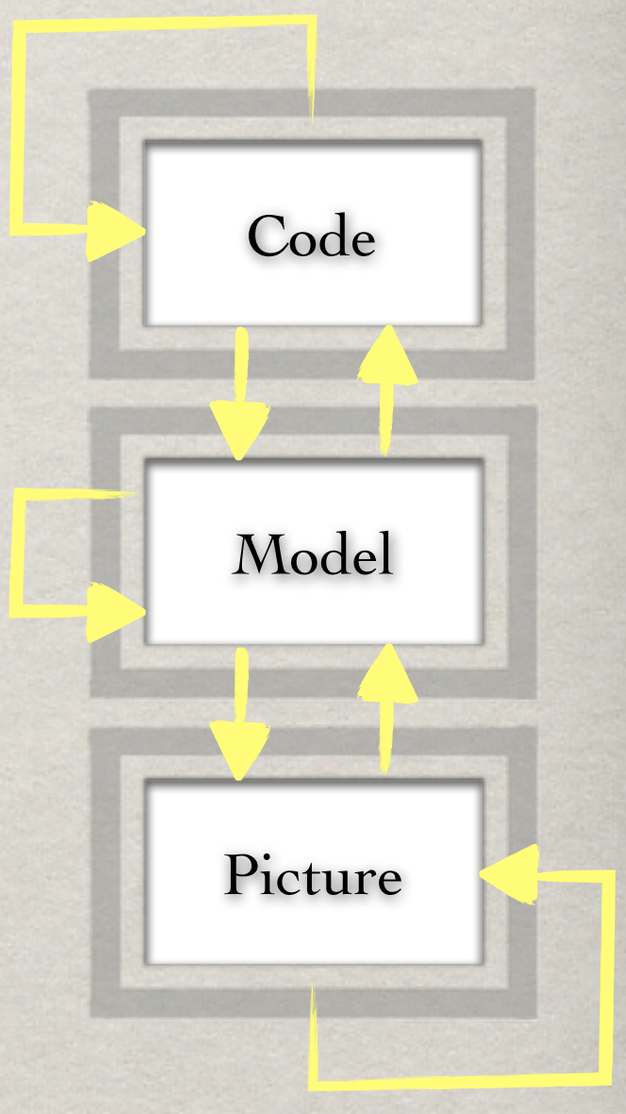
Rascal is EASY

- Extract,
  - Parse DSLs and (legacy) GPLs
  - Create abstract relational models (like call graphs)



Rascal is EASY

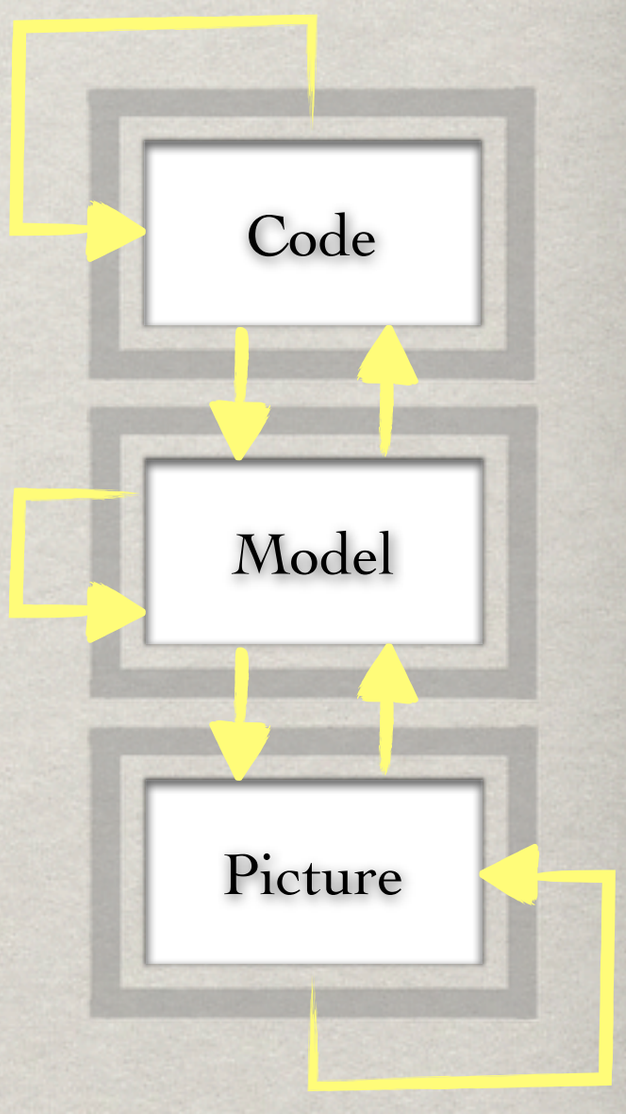
- Extract,
  - Parse DSLs and (legacy) GPLs
  - Create abstract relational models (like call graphs)
- Analyze
  - Query abstract/concrete syntax trees
  - Query relational models



CWI

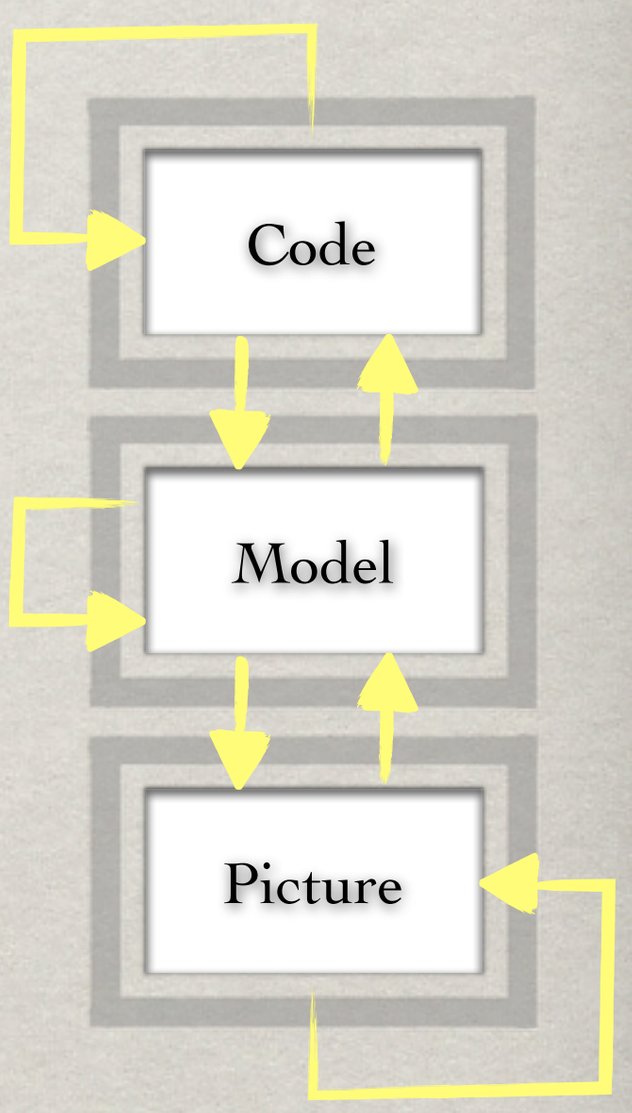
Rascal is EASY

- Extract,
  - Parse DSLs and (legacy) GPLs
  - Create abstract relational models (like call graphs)
- Analyze
  - Query abstract/concrete syntax trees
  - Query relational models
- **SY**nthesize
  - Generate code
  - Transform code
  - Produce data and visuals



CWI

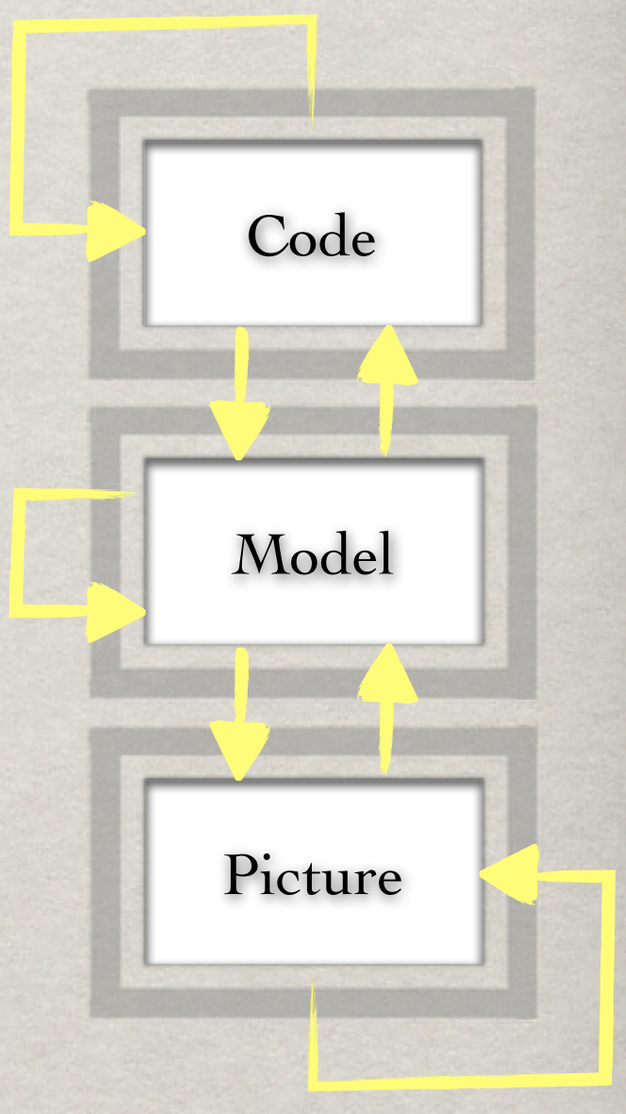
Rascal is EASY



CWI

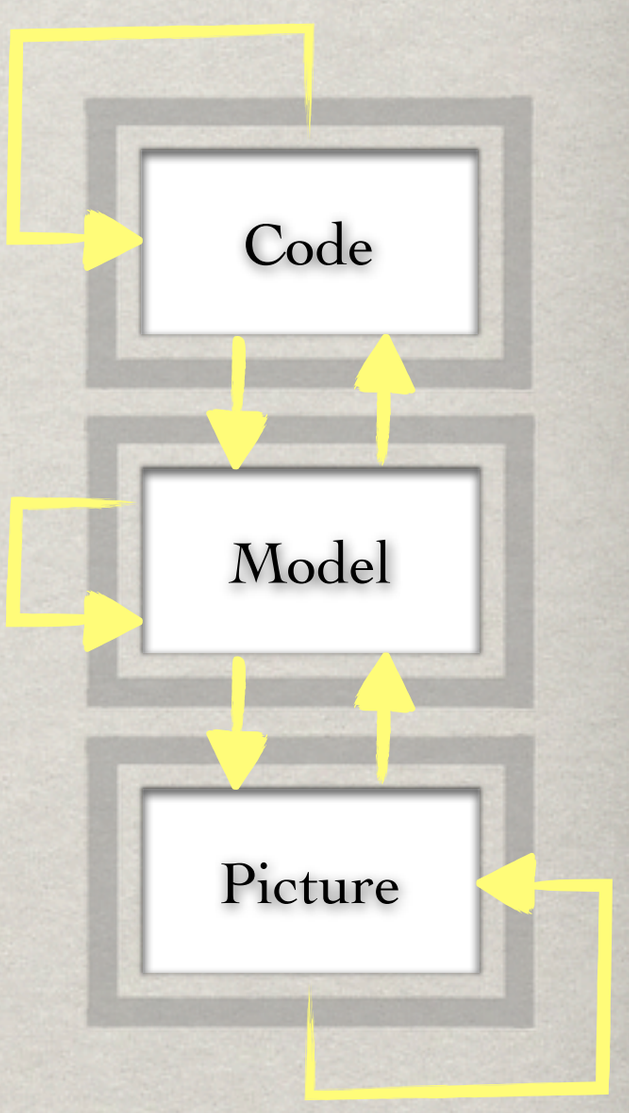
Rascal is EASY

- Extract,
  - Fast context-free general top-down parsing
  - Pattern matching & generic traversal



Rascal is EASY

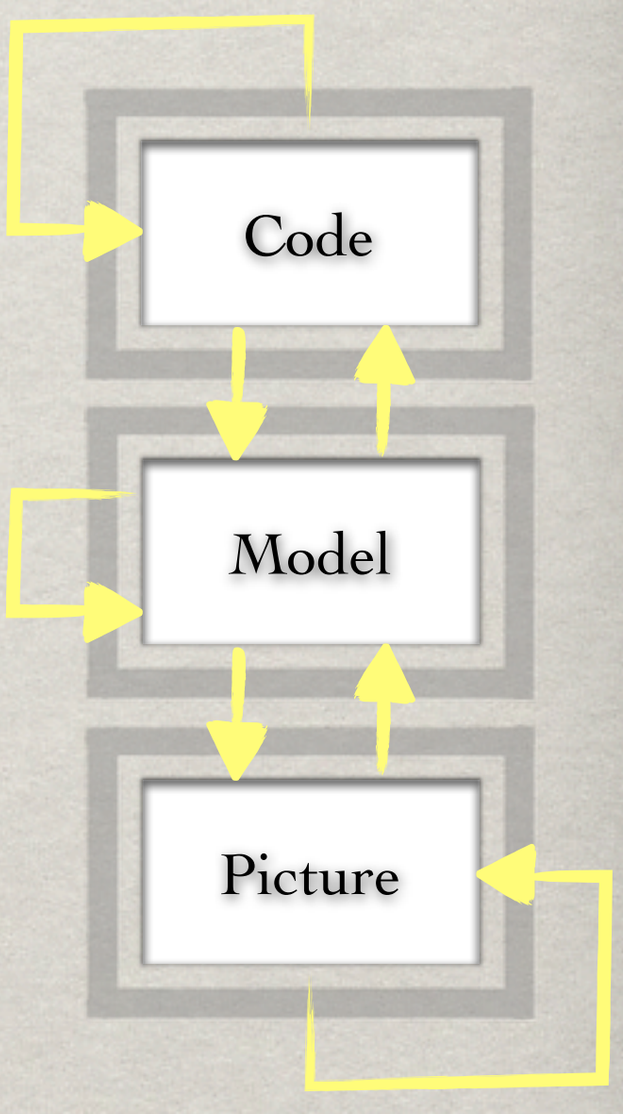
- Extract,
  - Fast context-free general top-down parsing
  - Pattern matching & generic traversal
- Analyze
  - Relational queries and comprehensions
  - Backtracking, fixed point computation, ...



CWI

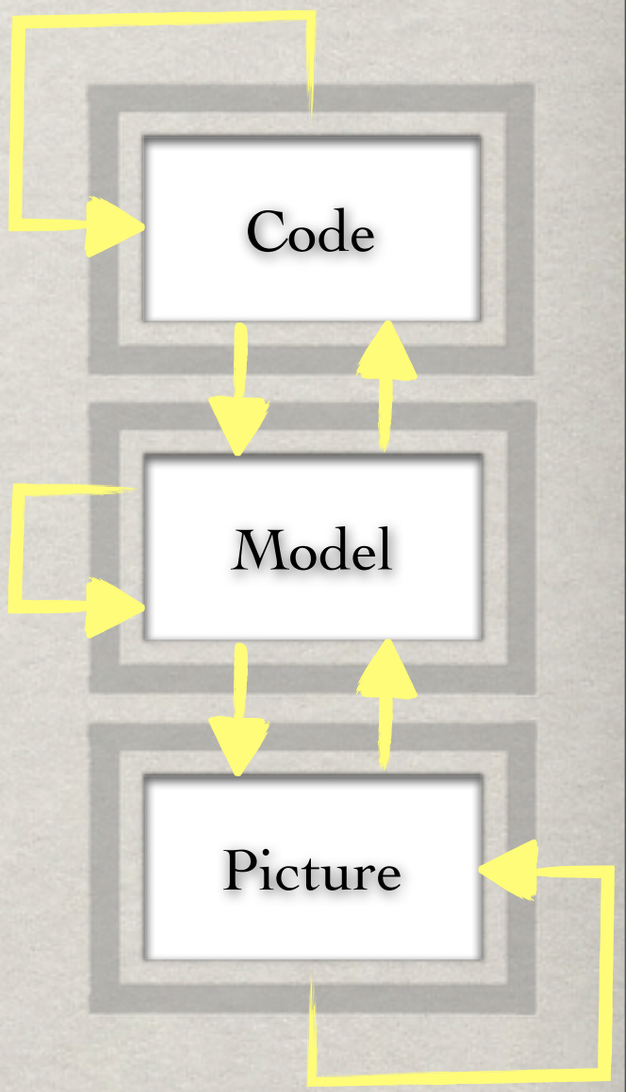
Rascal is EASY

- Extract,
  - Fast context-free general top-down parsing
  - Pattern matching & generic traversal
- Analyze
  - Relational queries and comprehensions
  - Backtracking, fixed point computation, ...
- **SY**nthesize
  - String templates
  - Concrete syntax
  - Interactive visualization generator



CWI

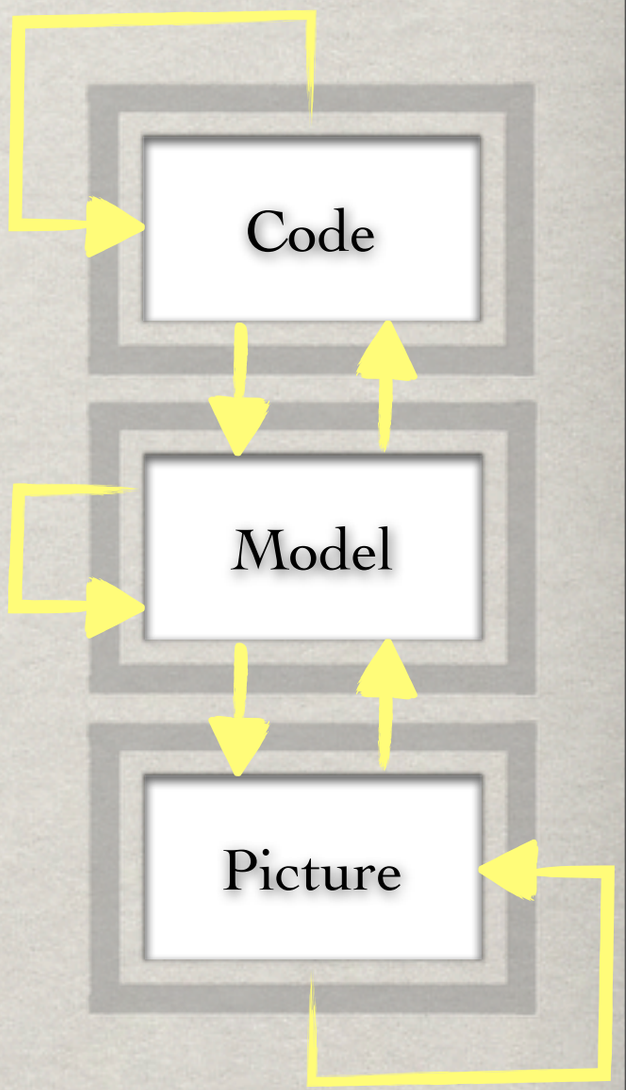
A one-slide DSL



CWI

A one-slide DSL

```
metro {
   Centraal Waterloo Weesperplein Wibautstraat Amstel;
   Amstel Spaklerweg Overamstel Rai Zuid;
   Rai Zuid Amstelveenseweg Lelylaan Sloterdijk;
   Centraal Rokin FerdinandBol Zuid;
}
```

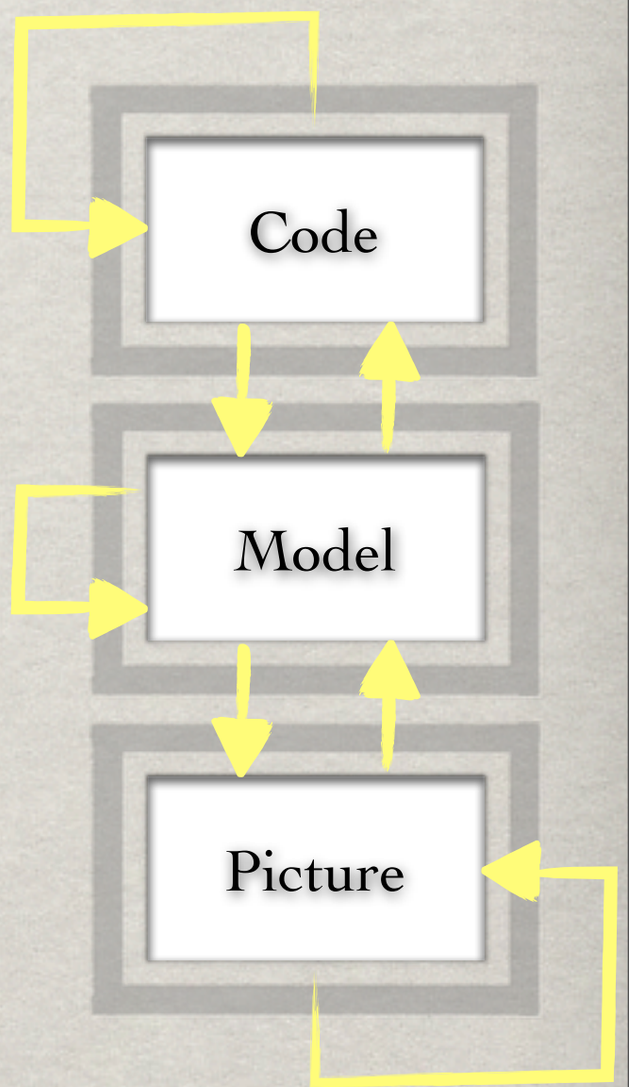


CWI

A one-slide DSL

```
metro {
   Centraal Waterloo Weesperplein Wibautstraat Amstel;
   Amstel Spaklerweg Overamstel Rai Zuid;
   Rai Zuid Amstelveenseweg Lelylaan Sloterdijk;
   Centraal Rokin FerdinandBol Zuid;
}
```

```
{ <“Centraal”, “Waterloo”>,
```

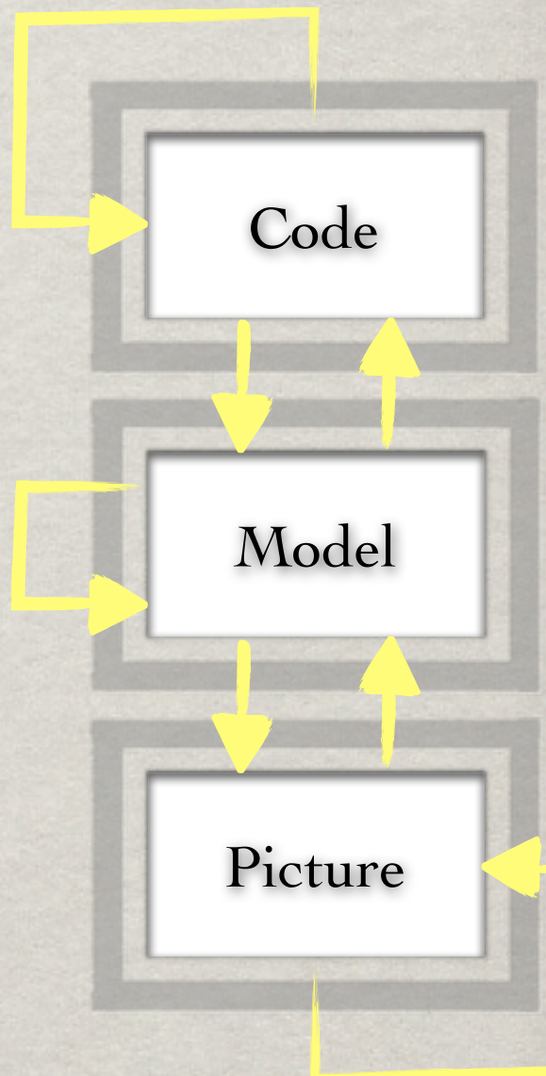


CWI

A one-slide DSL

```
metro {
   Centraal Waterloo Weesperplein Wibautstraat Amstel;
   Amstel Spaklerweg Overamstel Rai Zuid;
   Rai Zuid Amstelveenseweg Lelylaan Sloterdijk;
   Centraal Rokin FerdinandBol Zuid;
}
```

```
{ <“Centraal”, “Waterloo”>,
  <“Waterloo”,” Weesperplein”>, … }
```

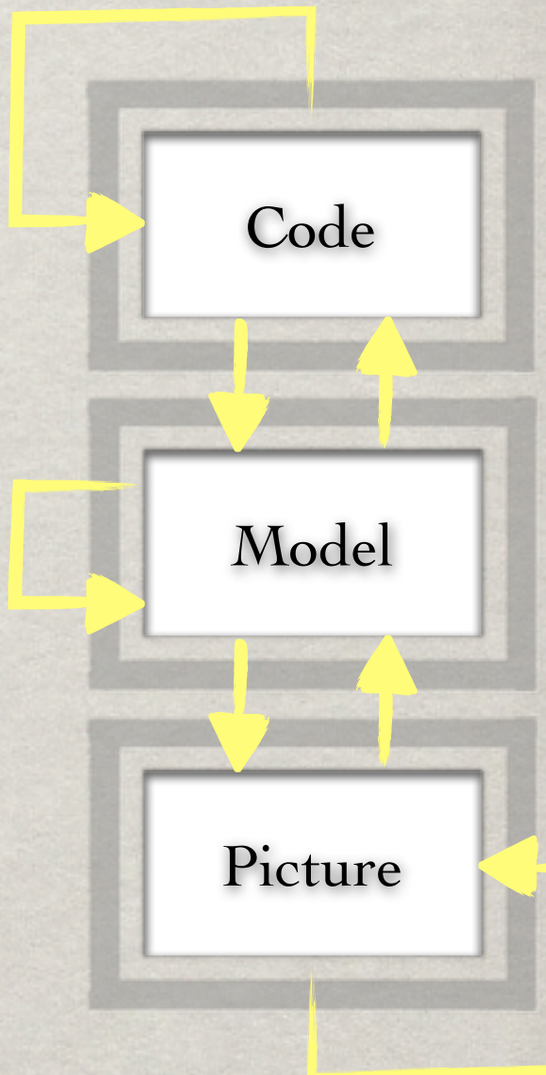


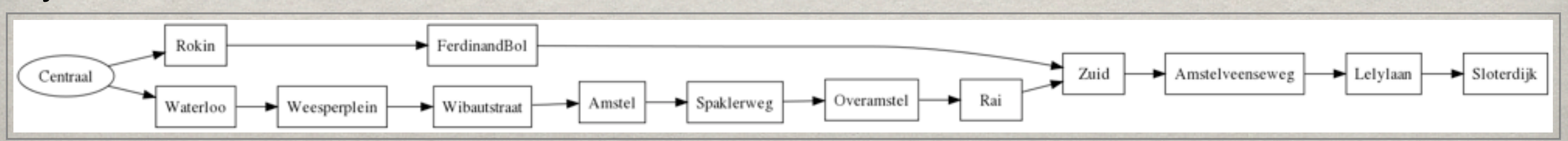
A one-slide DSL

```
metro {
   Centraal Waterloo Weesperplein Wibautstraat Amstel;
   Amstel Spaklerweg Overamstel Rai Zuid;
   Rai Zuid Amstelveenseweg Lelylaan Sloterdijk;
   Centraal Rokin FerdinandBol Zuid;
}
```

```
{ <“Centraal”, “Waterloo”>,
  <“Waterloo”,” Weesperplein”>, … }
```

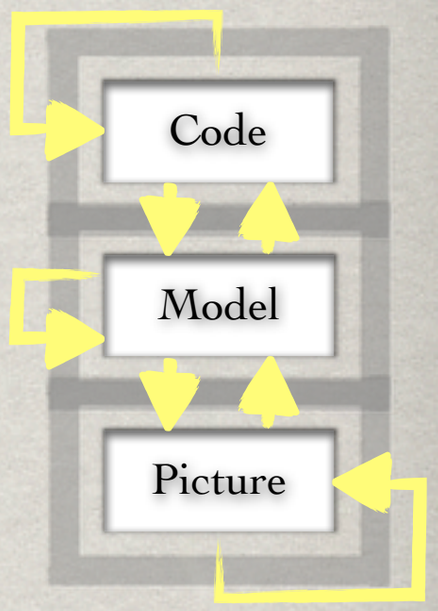
```
digraph Metro {
   node [shape=box]
   Centraal -> Waterloo
   Waterloo -> Weesperplein ...
   Centraal [shape=ellipse]
}
```





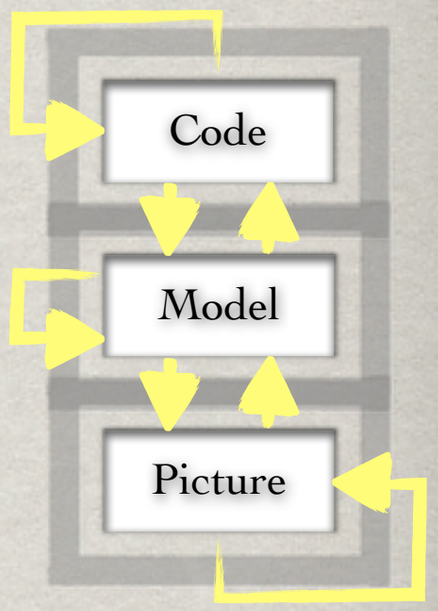
CWI

A one-slide DSL



CWI

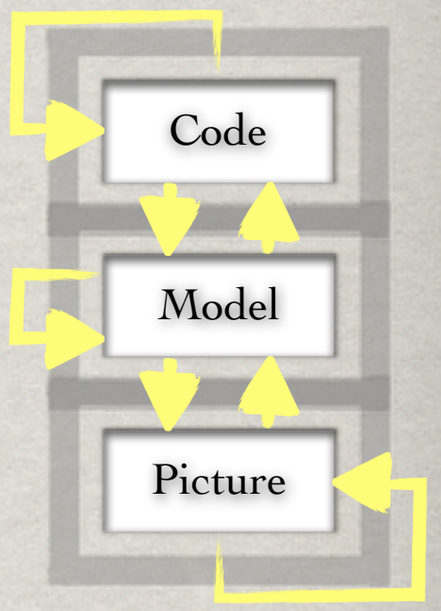
A one-slide DSL




```
module Metro
```

A one-slide DSL

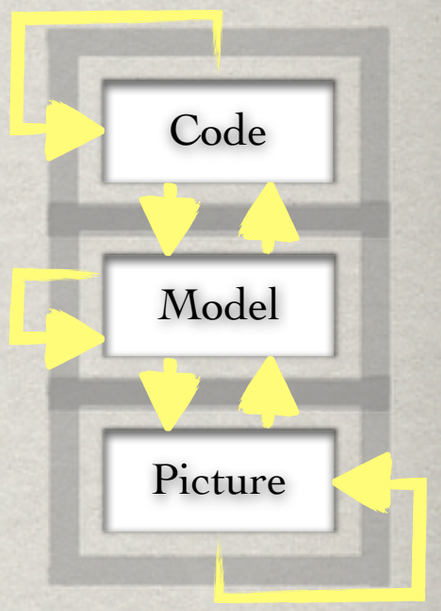
```
module Metro

start syntax System = “metro” “{“ Track* tracks “}”;
```



CWI

A one-slide DSL

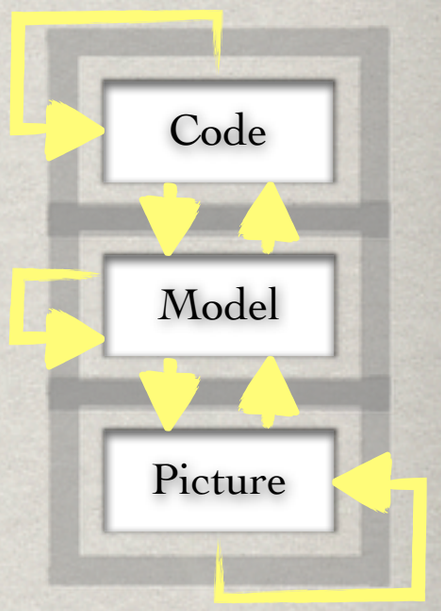
```
module Metro

start syntax System = “metro” “{“ Track* tracks “}”;

syntax Track = Id+ stations “;” ;
```



CWI

A one-slide DSL

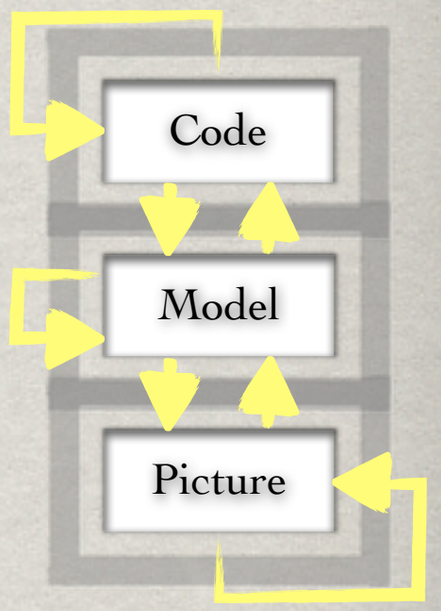
```
module Metro

start syntax System = “metro” “{“ Track* tracks “}”;

syntax Track = Id+ stations “;” ;

lexical Id = [A-Za-z][A-Za-z0-9]*;
```



CWI

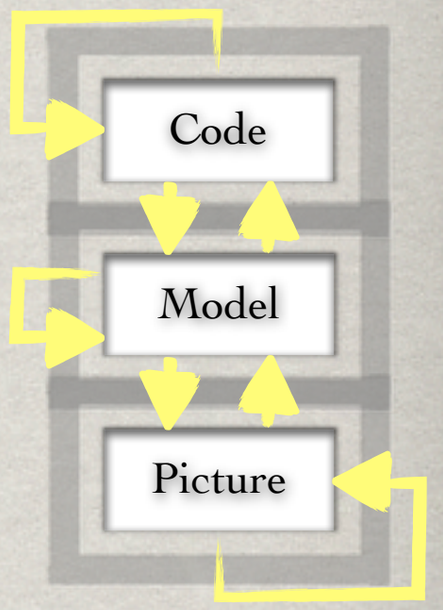
A one-slide DSL

```
module Metro

start syntax System = “metro” “{“ Track* tracks “}”;

syntax Track = Id+ stations “;” ;

lexical Id = [A-Za-z][A-Za-z0-9]*;

layout WS = [\ \t\n\r]*;
```



CWI

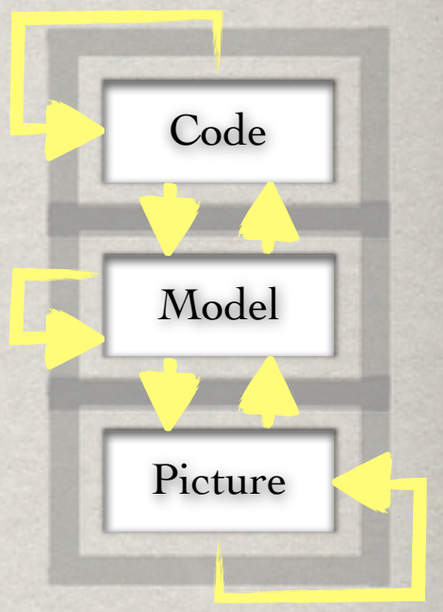
A ONE-SLIDE DSL




```
module Metro

start syntax System = “metro” “{“ Track* tracks “}”;

syntax Track = Id+ stations “;” ;

lexical Id = [A-Za-z][A-Za-z0-9]*;

layout WS = [\ \t\n\r]*;

rel[Id,Id] extractMetroGraph(loc source) =
```

CWI

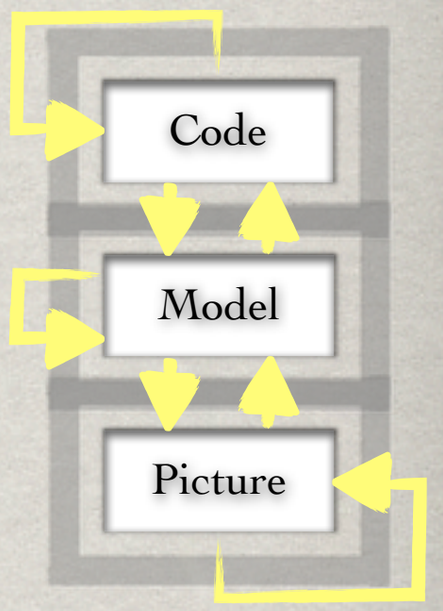
A ONE-SLIDE DSL




```
module Metro
start syntax System = “metro” “{“ Track* tracks “}”;
syntax Track = Id+ stations “;” ;
lexical Id = [A-Za-z][A-Za-z0-9]*;
layout WS = [\ \t\n\r]*;

rel[Id,Id] extractMetroGraph(loc source) =
  {<from, to> | /Track t := parse(#start[System], source),
```

CWI

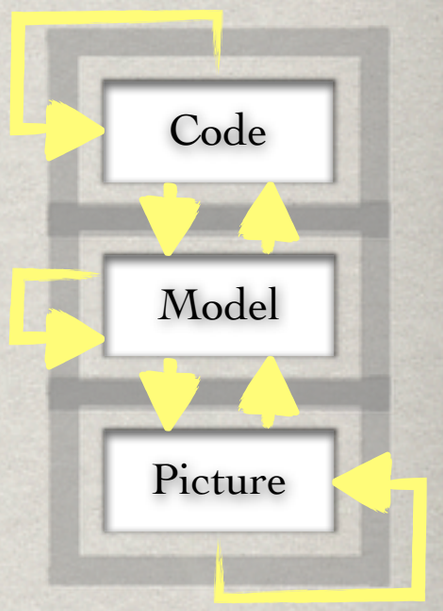
A ONE-SLIDE DSL




```
module Metro
start syntax System = “metro” “{“ Track* tracks “}”;
syntax Track = Id+ stations “;” ;
lexical Id = [A-Za-z][A-Za-z0-9]*;
layout WS = [\ \t\n\r]*;

rel[Id,Id] extractMetroGraph(loc source) =
  {<from, to> | /Track t := parse(#start[System], source),
       (Track) `<Id+ _> <Id from> <Id to> <Id+ _>;` := t};
```

CWI

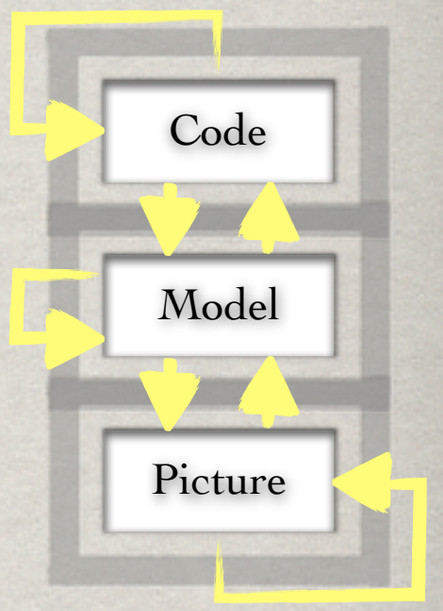
A ONE-SLIDE DSL




```
module Metro
start syntax System = “metro” “{“ Track* tracks “}”;
syntax Track = Id+ stations “;” ;
lexical Id = [A-Za-z][A-Za-z0-9]*;
layout WS = [\ \t\n\r]*;

rel[Id,Id] extractMetroGraph(loc source) =
  {<from, to> | /Track t := parse(#start[System], source),
        (Track) `<Id+ _> <Id from> <Id to> <Id+ _>;` := t};
bool isHub(rel[Id,Id] metro, Id station) = size(metro[Id]) > 1;
```

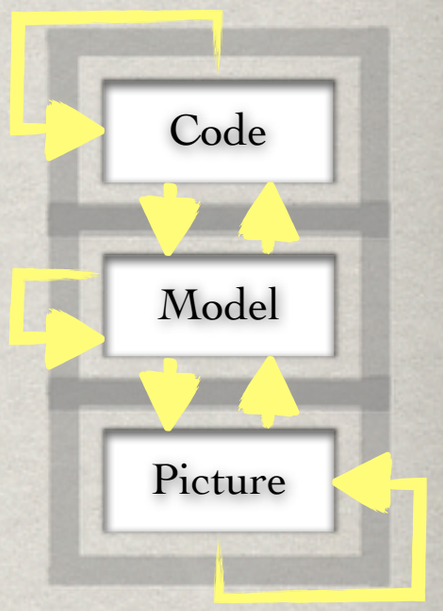
CWI

A ONE-SLIDE DSL



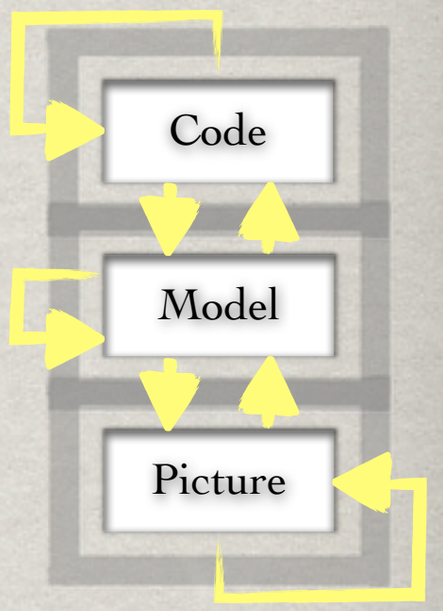

```
module Metro
start syntax System = “metro” “{“ Track* tracks “}”;
syntax Track = Id+ stations “;” ;
lexical Id = [A-Za-z][A-Za-z0-9]*;
layout WS = [\ \t\n\r]*;

rel[Id,Id] extractMetroGraph(loc source) =
  {<from, to> | /Track t := parse(#start[System], source),
        (Track) `<Id+ _> <Id from> <Id to> <Id+ _>;` := t};
bool isHub(rel[Id,Id] metro, Id station) = size(metro[Id]) > 1;

void synthesizeDotGraph(loc target, rel[Id from,Id to] metro) {
```

A ONE-SLIDE DSL



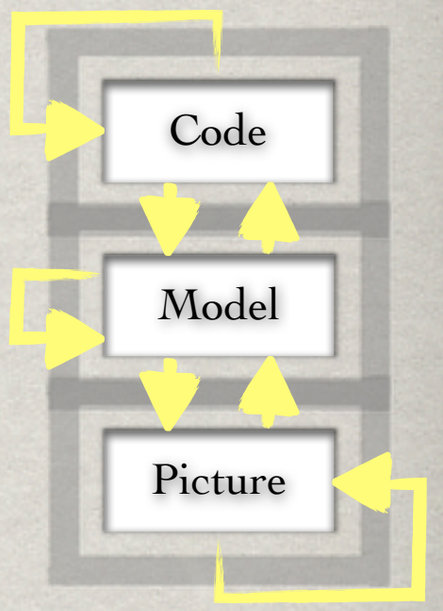

```
module Metro
start syntax System = “metro” “{“ Track* tracks “}”;
syntax Track = Id+ stations “;” ;
lexical Id = [A-Za-z][A-Za-z0-9]*;
layout WS = [\ \t\n\r]*;

rel[Id,Id] extractMetroGraph(loc source) =
  {<from, to> | /Track t := parse(#start[System], source),
        (Track) `<Id+ _> <Id from> <Id to> <Id+ _>;` := t};
bool isHub(rel[Id,Id] metro, Id station) = size(metro[Id]) > 1;

void synthesizeDotGraph(loc target, rel[Id from,Id to] metro) {
  writeFile(target,”digraph Metro { node [shape=box]
```

A ONE-SLIDE DSL



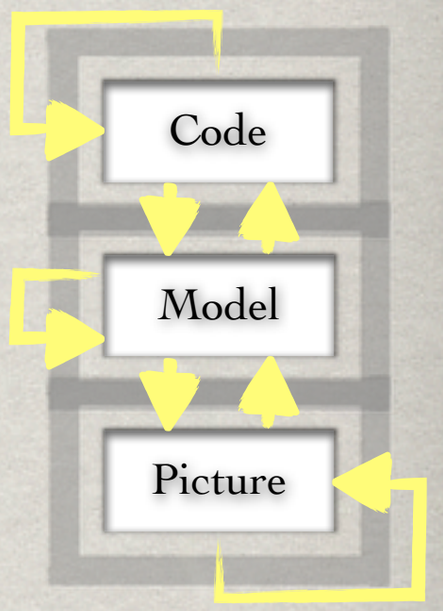

```
module Metro
start syntax System = “metro” “{“ Track* tracks “}”;
syntax Track = Id+ stations “;” ;
lexical Id = [A-Za-z][A-Za-z0-9]*;
layout WS = [\ \t\n\r]*;

rel[Id,Id] extractMetroGraph(loc source) =
  {<from, to> | /Track t := parse(#start[System], source),
        (Track) `<Id+ _> <Id from> <Id to> <Id+ _>;` := t};
bool isHub(rel[Id,Id] metro, Id station) = size(metro[Id]) > 1;

void synthesizeDotGraph(loc target, rel[Id from,Id to] metro) {
  writeFile(target,”digraph Metro { node [shape=box]
                    ‘<for (<from, to> <- metro) {>
```

A ONE-SLIDE DSL



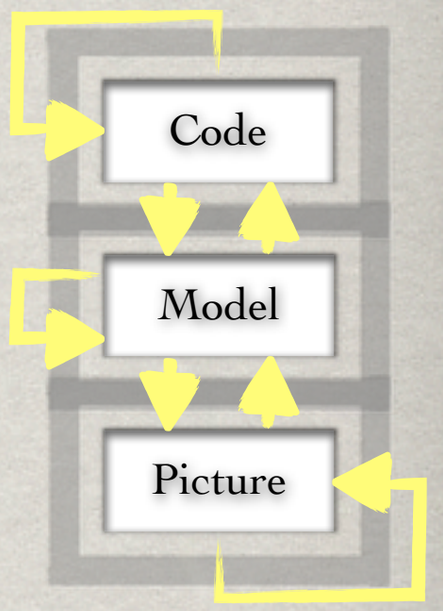

```
module Metro
start syntax System = “metro” “{“ Track* tracks “}”;
syntax Track = Id+ stations “;” ;
lexical Id = [A-Za-z][A-Za-z0-9]*;
layout WS = [\ \t\n\r]*;

rel[Id,Id] extractMetroGraph(loc source) =
  {<from, to> | /Track t := parse(#start[System], source),
        (Track) `<Id+ _> <Id from> <Id to> <Id+ _>;` := t};
bool isHub(rel[Id,Id] metro, Id station) = size(metro[Id]) > 1;

void synthesizeDotGraph(loc target, rel[Id from,Id to] metro) {
  writeFile(target,”digraph Metro { node [shape=box]
                  ‘<for (<from, to> <- metro) {>
                  ‘  <from> -\> <to><}>
```

A ONE-SLIDE DSL



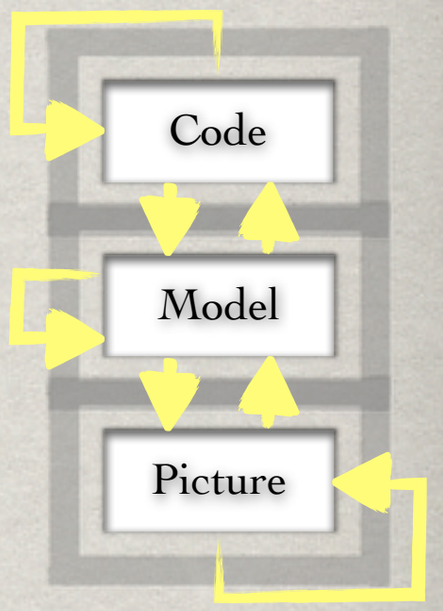

```
module Metro
start syntax System = “metro” “{“ Track* tracks “}”;
syntax Track = Id+ stations “;” ;
lexical Id = [A-Za-z][A-Za-z0-9]*;
layout WS = [\ \t\n\r]*;

rel[Id,Id] extractMetroGraph(loc source) =
  {<from, to> | /Track t := parse(#start[System], source),
        (Track) `<Id+ _> <Id from> <Id to> <Id+ _>;` := t};
bool isHub(rel[Id,Id] metro, Id station) = size(metro[Id]) > 1;

void synthesizeDotGraph(loc target, rel[Id from,Id to] metro) {
  writeFile(target,”digraph Metro { node [shape=box]
                    ‘<for (<from, to> <- metro) {>
                    ‘  <from> -\> <to><}>
                    ‘<for (st <- metro<from>, isHub(metro, st)){>
```

A ONE-SLIDE DSL



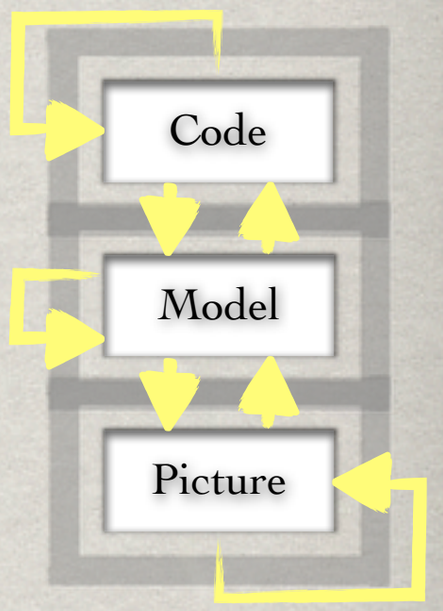

```
module Metro
start syntax System = “metro” “{“ Track* tracks “}”;
syntax Track = Id+ stations “;” ;
lexical Id = [A-Za-z][A-Za-z0-9]*;
layout WS = [\ \t\n\r]*;

rel[Id,Id] extractMetroGraph(loc source) =
  {<from, to> | /Track t := parse(#start[System], source),
        (Track) `<Id+ _> <Id from> <Id to> <Id+ _>;` := t};
bool isHub(rel[Id,Id] metro, Id station) = size(metro[Id]) > 1;

void synthesizeDotGraph(loc target, rel[Id from,Id to] metro) {
  writeFile(target,”digraph Metro { node [shape=box]
                    ‘<for (<from, to> <- metro) {>
                    ‘  <from> -\> <to><}>
                    ‘<for (st <- metro<from>, isHub(metro, st)){>
                    ‘  <st> [shape=ellipse]<}>
```

A ONE-SLIDE DSL




```
module Metro
start syntax System = “metro” “{“ Track* tracks “}”;
syntax Track = Id+ stations “;” ;
lexical Id = [A-Za-z][A-Za-z0-9]*;
layout WS = [\ \t\n\r]*;

rel[Id,Id] extractMetroGraph(loc source) =
  {<from, to> | /Track t := parse(#start[System], source),
        (Track) `<Id+ _> <Id from> <Id to> <Id+ _>;` := t};
bool isHub(rel[Id,Id] metro, Id station) = size(metro[Id]) > 1;

void synthesizeDotGraph(loc target, rel[Id from,Id to] metro) {
  writeFile(target,”digraph Metro { node [shape=box]
                    ‘<for (<from, to> <- metro) {>
                    ‘  <from> -\> <to><}>
                    ‘<for (st <- metro<from>, isHub(metro, st)){>
                    ‘  <st> [shape=ellipse]<}>
                    ‘}”);
```

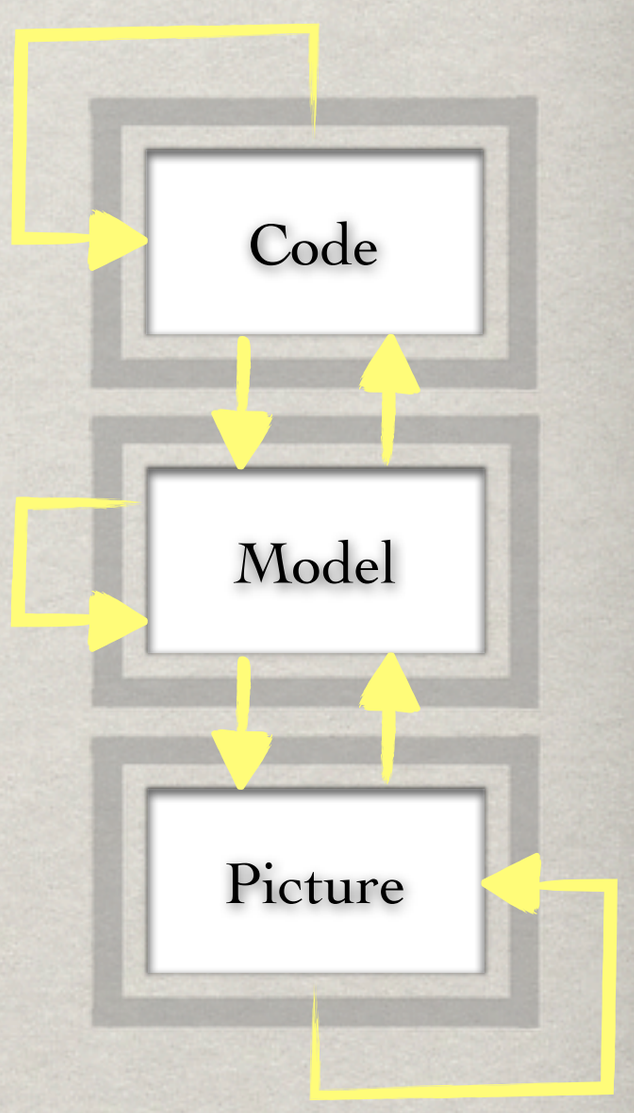
A ONE-SLIDE DSL




```
module Metro
start syntax System = “metro” “{“ Track* tracks “}”;
syntax Track = Id+ stations “;” ;
lexical Id = [A-Za-z][A-Za-z0-9]*;
layout WS = [\ \t\n\r]*;

rel[Id,Id] extractMetroGraph(loc source) =
  {<from, to> | /Track t := parse(#start[System], source),
        (Track) `<Id+ _> <Id from> <Id to> <Id+ _>;` := t};
bool isHub(rel[Id,Id] metro, Id station) = size(metro[Id]) > 1;

void synthesizeDotGraph(loc target, rel[Id from,Id to] metro) {
  writeFile(target,”digraph Metro { node [shape=box]
                    ‘<for (<from, to> <- metro) {>
                    ‘  <from> -\> <to><}>
                    ‘<for (st <- metro<from>, isHub(metro, st)){>
                    ‘  <st> [shape=ellipse]<}>
                    ‘}”);
```

CWI

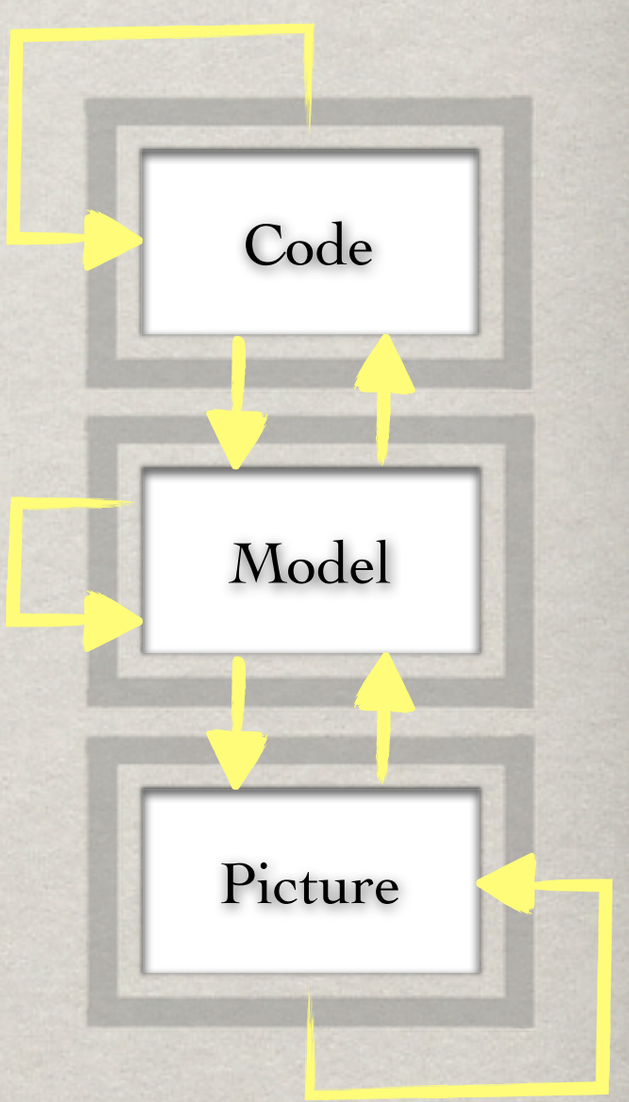
A ONE-SLIDE DSL



CWI

A ONE-SLIDE DSL

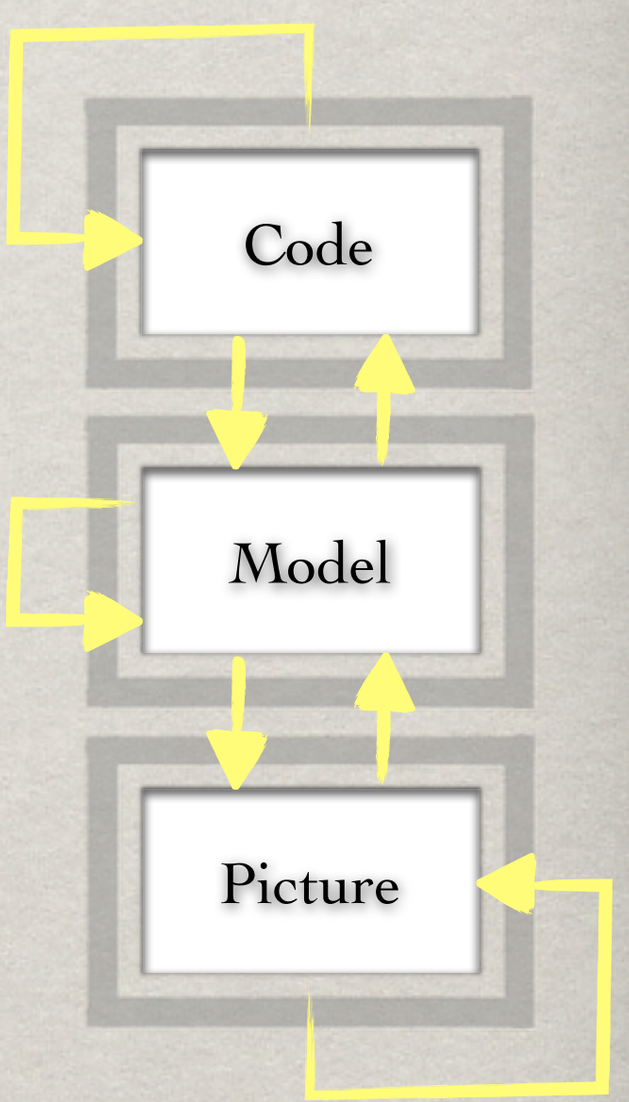
- What is the point?
  - Rapid tool development
  - No boilerplate
  - No glue
  - No magic
  - Done. Next!



CWI

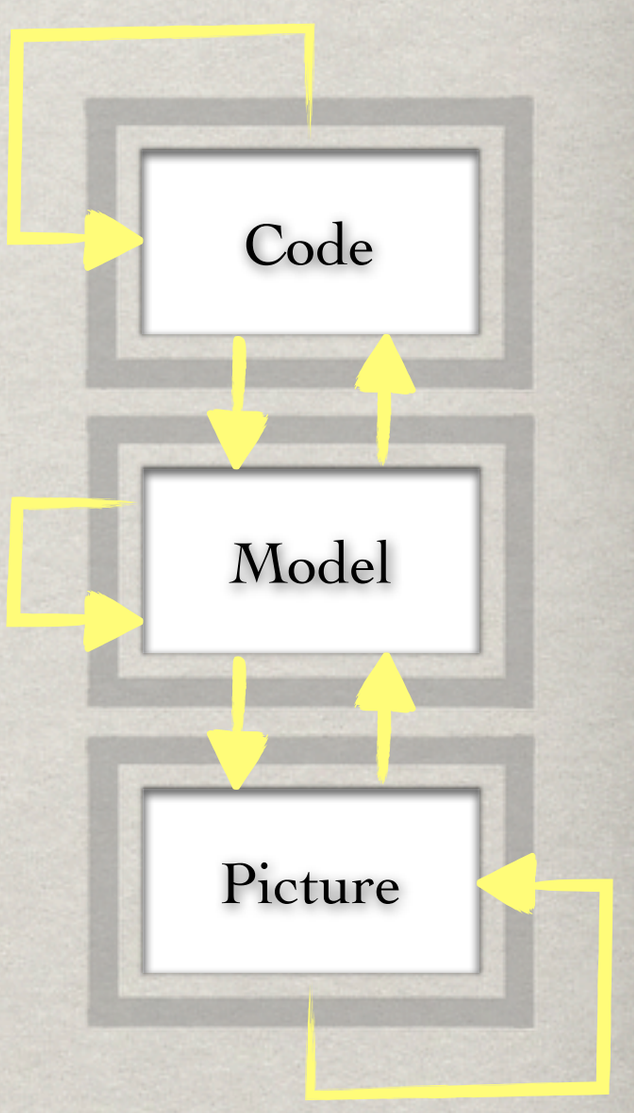
A ONE-SLIDE DSL

- What is the point?
  - Rapid tool development
  - No boilerplate
  - No glue
  - No magic
  - Done. Next!
- This works for
  - all kinds of meta-programming tools
  - all kinds of languages



CWI

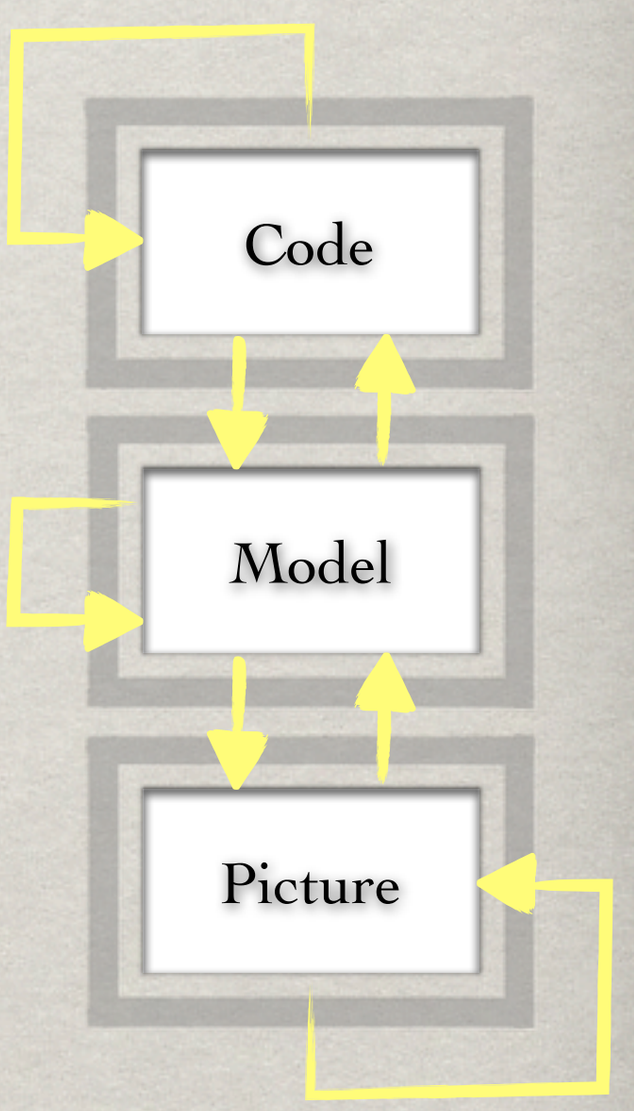
Library development



CWI

Library development

- **Type-safe** access to resources such as bug databases, version management systems, spreadsheets, webservices

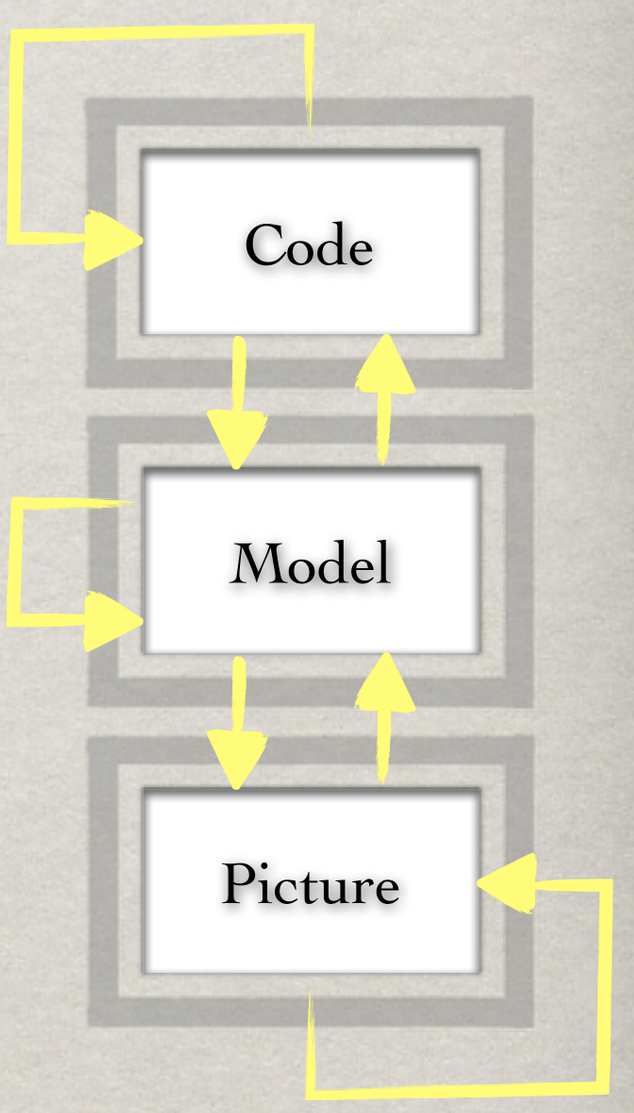


CWI

Library development

- **Type-safe** access to resources such as bug databases, version management systems, spreadsheets, webservices

- **Front-ends** for programming languages

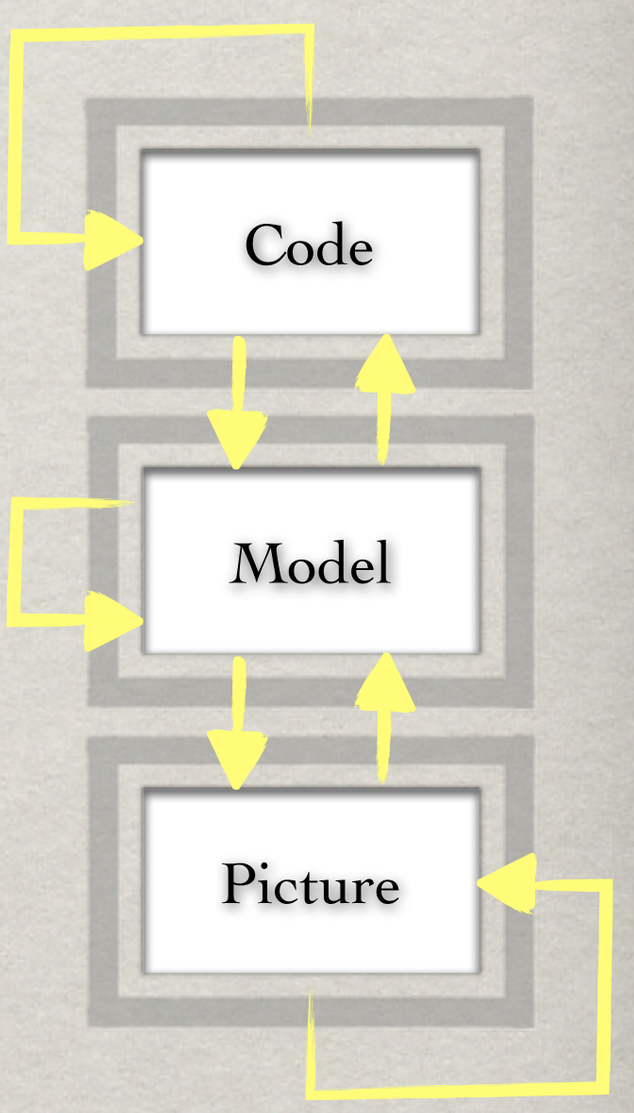


Library development

- **Type-safe** access to resources such as bug databases, version management systems, spreadsheets, webservices

- **Front-ends** for programming languages

- **Generic analyses**; statistics, constraints, satisfiability, …



CWI

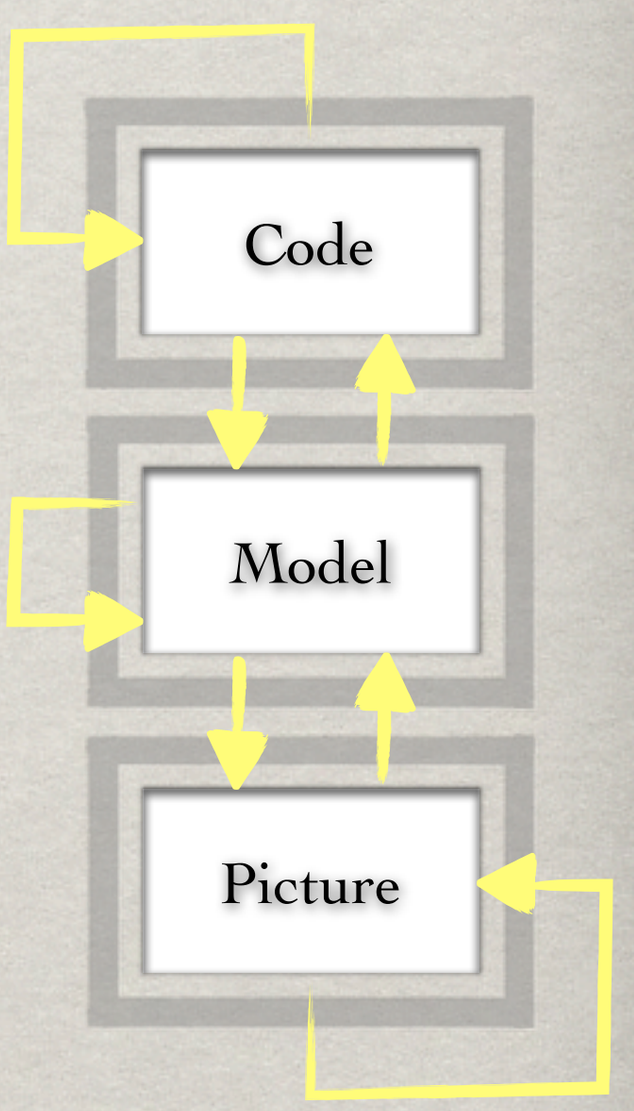
Library development

- **Type-safe** access to resources such as bug databases, version management systems, spreadsheets, webservices

- **Front-ends** for programming languages

- **Generic analyses**; statistics, constraints, satisfiability, …

- **Visualization:** one-stop-library for any visualization (graph, chart, browser, …)



CWI

State of the art

CWI

State of the art

- Rascal is **beta** quality, *alpha* guarantees

CWI

State of the art

- Rascal is **beta** quality, *alpha* guarantees
- Experience
- Software Evolution/Construction at UvA & OU
- Bachelors Amsterdam University College (AUC)
- Applications in SWAT (see **posters**)

CWI

State of the art

- Rascal is **beta** quality, *alpha* guarantees
- Experience
  - Software Evolution/Construction at UvA & OU
  - Bachelors Amsterdam University College (AUC)
  - Applications in SWAT (see **posters**)
- Currently applying lessons-learned

CWI

CWI

FOR RESEARCH IN ANALYSIS AND SYNTHESIS OF COMPLEX SOFTWARE ARTIFACTS

CWI

Software may be hard but Rascal is easy for research in analysis and synthesis of complex software artifacts

CWI

http://www.rascal-mpl.org

Software may be hard but Rascal is easy for research in analysis and synthesis of complex software artifacts