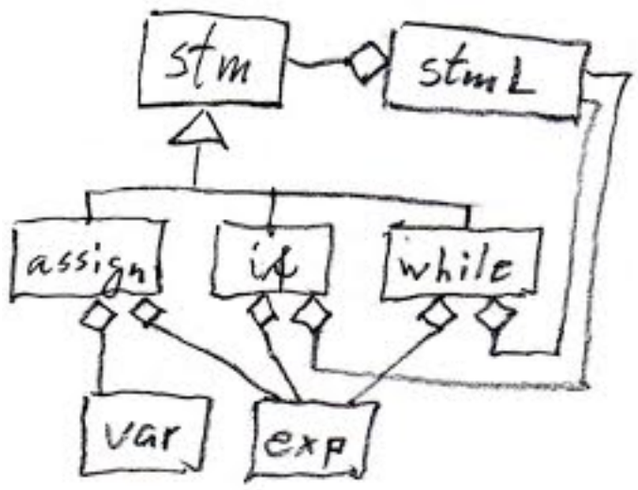
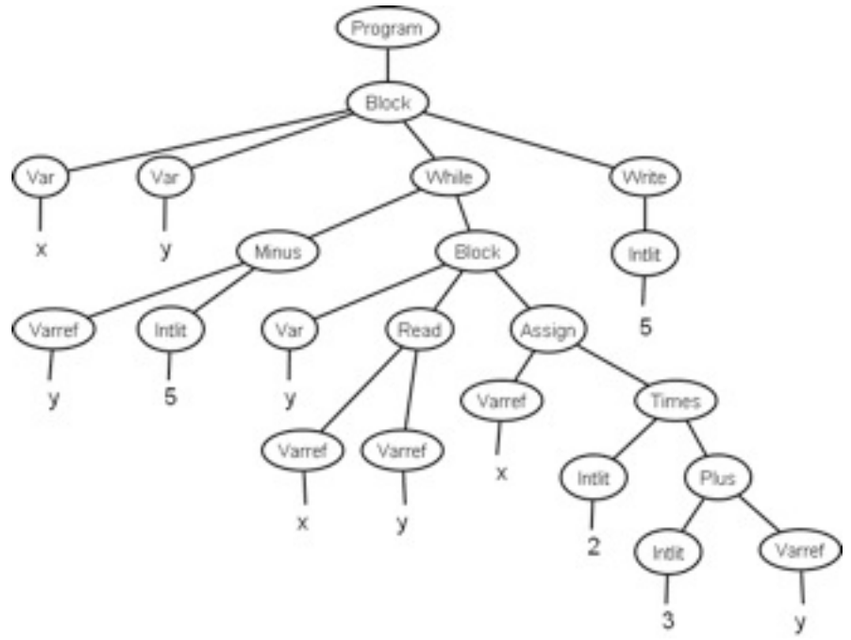
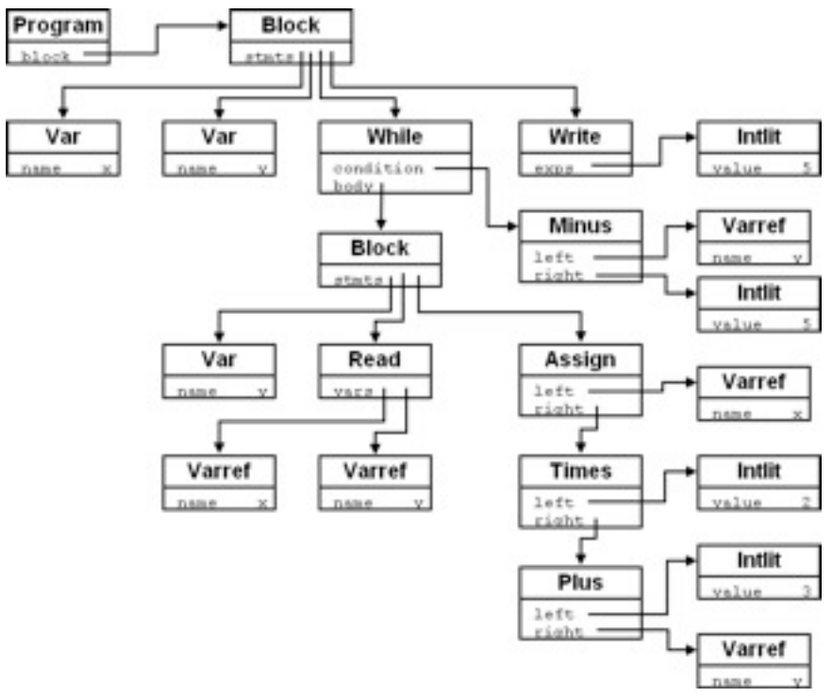




Abstract Syntax Sucks!

deconstruction (?), allegory (?), ...

Tijs van der Storm



Alfred Aho
(contributed to *lex*)

Alfred Aho
(contributed to *lex*)

Scanners suck!

Jurgen Vinju

“Deconstruction”

- Turn hierarchies up-side-down
- Bring margins to the center

Fundamental Concepts in Programming Languages

CHRISTOPHER STRACHEY

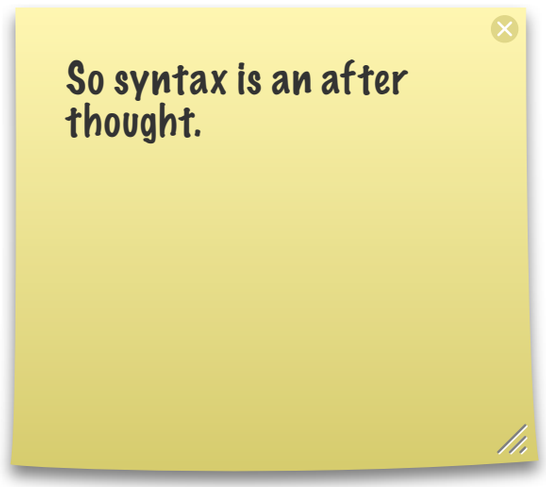
In a rough and ready sort of way it seems to me fair to think of the semantics as being what we want to say and the syntax as how we have to say it. In these terms the urgent task in programming languages is to explore the field of semantic possibilities. When we have discovered the main outlines and the principal peaks we can set about devising a suitably neat and satisfactory notation for them, and this is the moment for syntactic questions.

NB: 39 pages, just 5 occurences of the word syntax

http://www.itu.dk/courses/BPRD/E2009/fundamental-1967.pdf

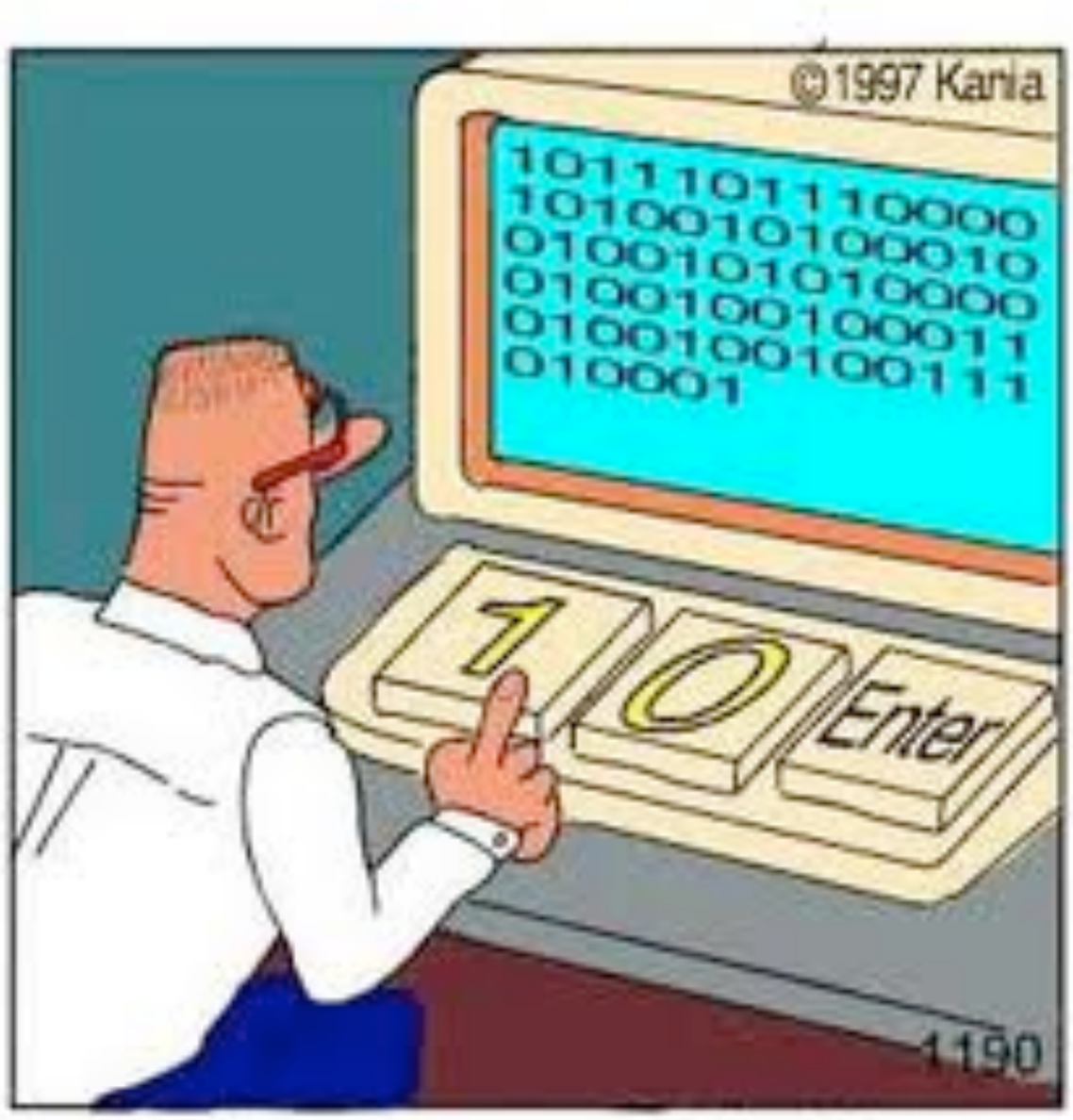
From the original notes...

Basic irrelevance of syntax and primacy of semantics.



http://fexpr.blogspot.nl/2011/06/primacy-of-syntax.html

But syntax is the UI!?!



Real programmers code in binary.

And syntax is part of that, NOT all of it, but nonetheless a very visible aspect of it.

Topsy turvy

- Concrete syntax over abstract syntax
- Implementation/engineering over semantics
- Comments, whitespace, layout, etc.

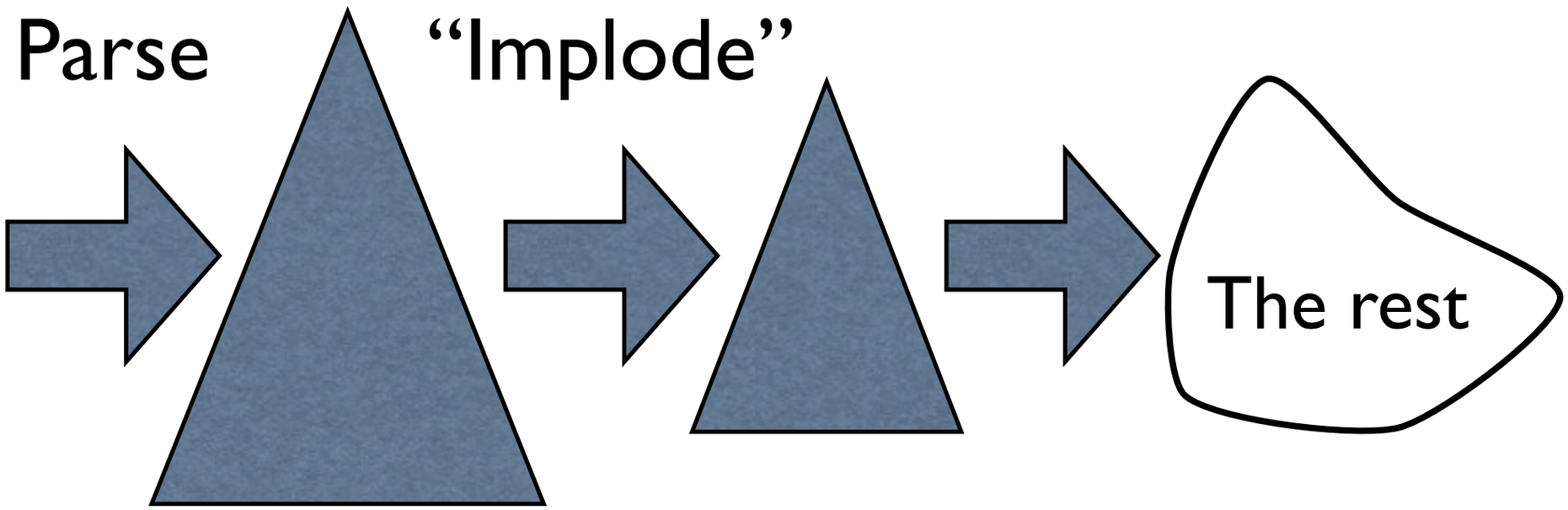
A compiler pipeline...

```
public X getX(int i) {
	return rows.get(i);
}

public X getY(int i) {
	return cols.get(i);
}

public int lastIndexX() {
	return rows.size() - 1;
}

public int lastIndexY() {
	return cols.size() - 1;
}
```



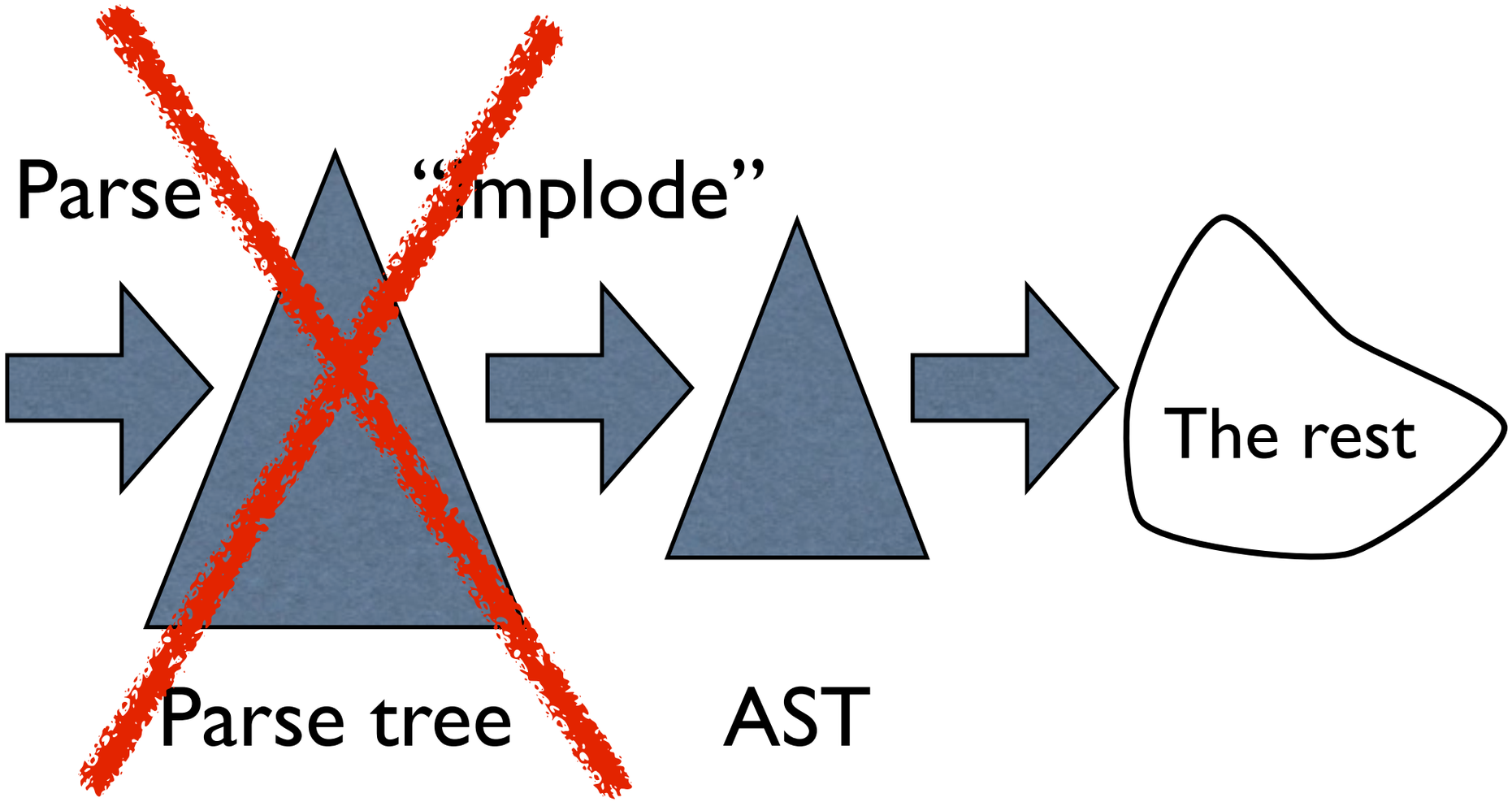
Really...

```
public X getX(int i) {
	return rows.get(i);
}

public X getY(int i) {
	return cols.get(i);
}

public int lastIndexX() {
	return rows.size() - 1;
}

public int lastIndexY() {
	return cols.size() - 1;
}
```



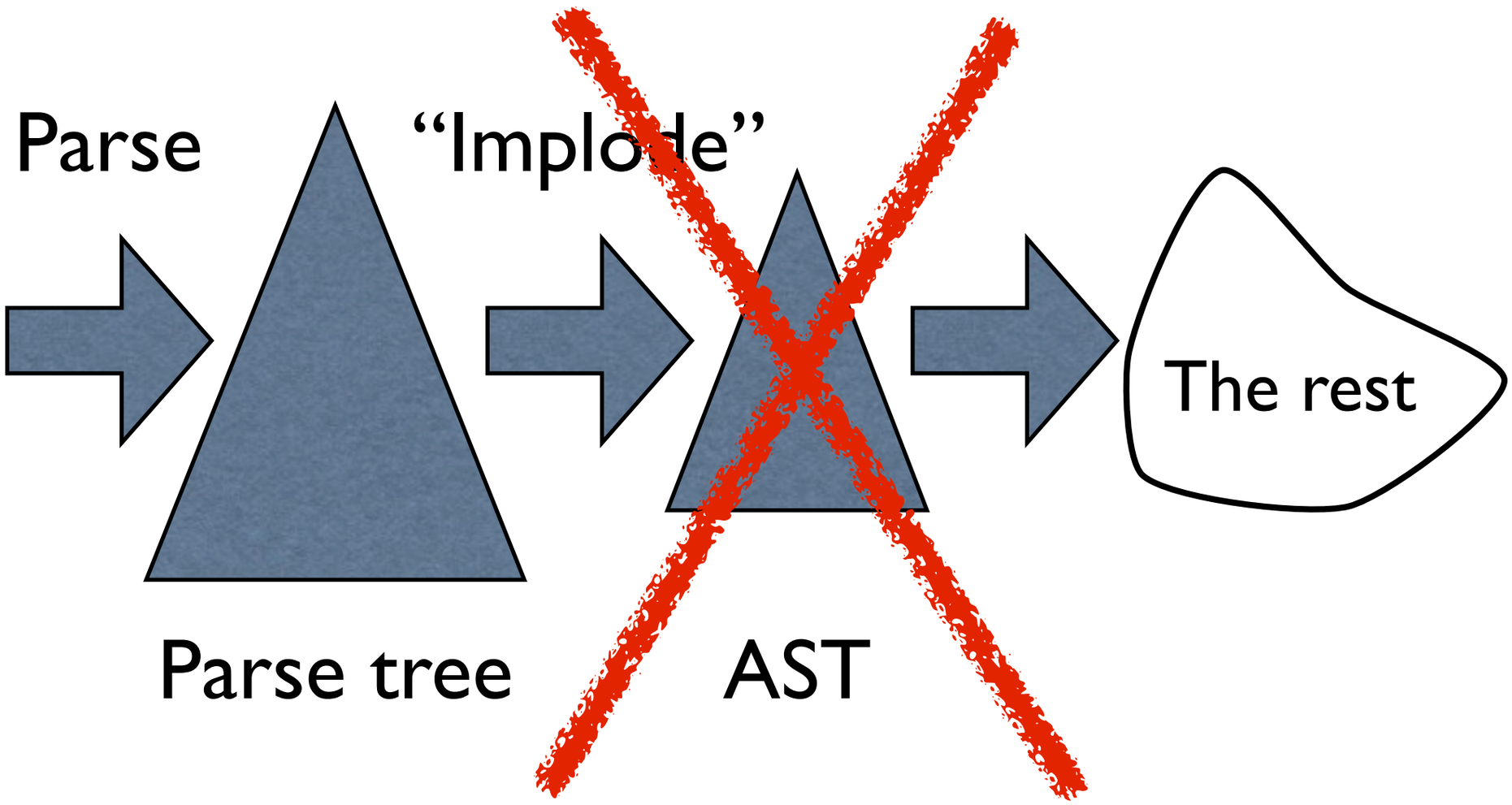
This talk

```
	public X getX(int i) {
		return rows.get(i);
	}

	public X getY(int i) {
		return cols.get(i);
	}

	public int lastIndexX() {
		return rows.size() - 1;
	}

	public int lastIndexY() {
		return cols.size() - 1;
	}
```



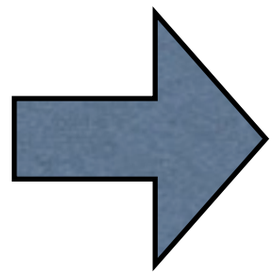
Rascal Language Workbench

- "Functional" meta-programming language
- DSL implementation and program understanding/renovation
- Source code in, source code out
- Built-in context-free grammars
- Pattern matching, traversal, comprehensions, relation algebra, ...

http://www.rascal-mpl.org

Language Workbench Challenge

```
form taxOfficeExample {
  "Did you sell a house in 2010?"
    boolean hasSoldHouse
  "Did you buy a house in 2010?"
    boolean hasBoughtHouse
  "Did you enter a loan?"
    boolean hasMaintLoan
  if (hasSoldHouse) {
    "What was the selling price?"
      money sellingPrice
    "Private debts for the sold house:"
      money privateDebt
    "Value residue:"
      money valueResidue =
        (sellingPrice - privateDebt)
  }
}
```



Did you sell a house in 2010?

Yes

Did you buy a house in 2010?

Choose an answer

Did you enter a loan?

Choose an answer

What was the selling price?

100

Private debts for the sold house:

200

Value residue:

-100.00

Submit taxOfficeExample

22 co-auteurs.

The State of the Art in Language Workbenc

Conclusions from the Language Workbench Challen

Sebastian Erdweg[1], Tijs van der Storm[2,3], Markus Völter[4], Meinte Boersma[5], Remi Bosman[6], William R. Cook[7], Albert Gerritsen[6], Angelo Hulshout[8], Steven Kelly[9], Alex Loh[7], Gabriël Konat[10], Pedro J. Molina[11], Martin Palatnik[6], Risto Pohjonen[9], Eugen Schindler[6], Klemens Schindler[6], Riccardo Solmi[12], Vlad Vergu[10], Eelco Visser[10], Kevin van der Vlist[13], Guido Wachsmuth[10], and Jimi van der Woning[13]

[1] TU Darmstadt, Germany [2] CWI, Amsterdam, The Netherlands [3] INRIA Lille Nord Europe, Lille, France [4] voelter.de, Stuttgart, Germany [5] DSL Consultancy, Leiden, The Netherlands [6] Sioux, Eindhoven, The Netherlands [7] University of Texas, Austin, US [8] Delphino Consultancy, Best, The Netherlands [9] MetaCase, Jyväskylä, Finland [10] TU Delft, The Netherlands [11] Icinetic, Sevilla, Spain [12] Independent, Bologna, Italy [13] Universiteit van Amsterdam

SLE 2013 6th International Conference on Software Language Engineering with SPLASH 2013 October 2013 @ Idianapolis, USA

Rascal's concrete syntax feature

- Grammar non-terminals are types
- Parse trees are values
- Concrete syntax pattern matching and construction
- All parse trees subtype of *Tree*
  - (an ADT for parse trees)

Syntax definition

```
module Syntax
extend lang::std::Layout;

start syntax Controller =
    controller:
        Events events
        ResetEvents? resets
        Commands? commands
        State+ states;

syntax Events
  = "events" Event* "end";
syntax ResetEvents
  = "resetEvents" Id* "end";
syntax Commands
  = "commands" Command* "end";
```

Syntax definition

```
module Syntax
extend lang::std::Layout;

start syntax Controller =
    controller:
        Events events
        ResetEvents? resets
        Commands? commands
        State+ states;

syntax Events
  = "events" Event* "end";
syntax ResetEvents
  = "resetEvents" Id* "end";
syntax Commands
  = "commands" Command* "end";
```

standard Layout

Syntax definition

start symbol

standard Layout

```
module Syntax
extend lang::std::Layout;

start syntax Controller =
    controller:
        Events events
        ResetEvents? resets
        Commands? commands
        State+ states;

syntax Events
  = "events" Event* "end";
syntax ResetEvents
  = "resetEvents" Id* "end";
syntax Commands
  = "commands" Command* "end";
```

Syntax definition

```
module Syntax
extend lang::std::Layout;

start syntax Controller =
    controller:
        Events events
        ResetEvents? resets
        Commands? commands
        State+ states;

syntax Events
  = "events" Event* "end";
syntax ResetEvents
  = "resetEvents" Id* "end";
syntax Commands
  = "commands" Command* "end";
```

standard Layout

start symbol

production label

Syntax definition

```
module Syntax
extend lang::std::Layout;

start syntax Controller =
    controller:
        Events events
        ResetEvents? resets
        Commands? commands
        State+ states;

syntax Events
  = "events" Event* "end";
syntax ResetEvents
  = "resetEvents" Id* "end";
syntax Commands
  = "commands" Command* "end";
```

standard Layout

start symbol

production label

subelement labels

Lexical syntax

```
lexical Id
  = ([a-zA-Z][a-zA-Z0-9_]* !>> [a-zA-Z0-9_])
  \ Reserved ;

keyword Reserved
  = "events"
  | "end"
  | "resetEvents"
  | "state"
  | "actions" ;
```

Lexical syntax

lexicals don't get layout

```
lexical Id
  = ([a-zA-Z][a-zA-Z0-9_]* !>> [a-zA-Z0-9_])
  \ Reserved ;

keyword Reserved
  = "events"
  | "end"
  | "resetEvents"
  | "state"
  | "actions" ;
```

Lexical syntax

lexicals don't get layout

character class

```
lexical Id
  = ([a-zA-Z][a-zA-Z0-9_]* !>> [a-zA-Z0-9_])
  \ Reserved ;

keyword Reserved
  = "events"
  | "end"
  | "resetEvents"
  | "state"
  | "actions" ;
```

Lexical syntax

lexicals don't get layout

follow restriction

character class

```
lexical Id
  = ([a-zA-Z][a-zA-Z0-9_]* !>> [a-zA-Z0-9_])
  \ Reserved ;

keyword Reserved
  = "events"
  | "end"
  | "resetEvents"
  | "state"
  | "actions" ;
```

Lexical syntax

lexicals don't get layout

follow restriction

character class

```
lexical Id
  = ([a-zA-Z][a-zA-Z0-9_]* !>> [a-zA-Z0-9_])
  \ Reserved ;
```

keyword reservation

```
keyword Reserved
  = "events"
  | "end"
  | "resetEvents"
  | "state"
  | "actions" ;
```

Lexical syntax

lexicals don't get layout

follow restriction

character class

```
lexical Id
  = ([a-zA-Z][a-zA-Z0-9_]* !>> [a-zA-Z0-9_])
  \ Reserved ;
```

keyword reservation

```
keyword Reserved
  = "events"
  | "end"
  | "resetEvents"
  | "state"
  | "actions" ;
```

keyword class

Parse trees

Everything is there!

Demo

- 1: parse trees in Rascal
- 2: concrete matching and construction
- 3: “analyzing” comments

WYSIWYG

- Concrete: x + y
- Abstract: add(var(“x”), var(“y”))

“Pretty” printing for free

- With ASTs: printing = pretty printing
  - AKA: inventing layout
- Parse trees can be *unparsed*
- Text can be highlighted based on parse trees

High-fidelity transformation

- Need to preserve comments/layout for
- Refactoring
- Renovation

Comments fly 1st class

```
if(hasSoldHouse) {
   /*
    * We only ask for <sellingPrice> and <privateDebt>
    * if <hasSoldHouse == true>
    */
   "What was the selling price of the house?"
     money sellingPrice
   "Private debts for the sold house:"
     money privateDebt
   "Value residue:"
     money valueResidue = (sellingPrice - privateDebt)
}
```

Comments fly 1st class

```
if(hasSoldHouse) {
  /*
   * We only ask for <sellingPrice> and <privateDebt>
   * if <hasSoldHouse > 1>
   */
  "What was the selling price of the house?"
    money sellingPrice
  "Private debts for the sold house:"
    money privateDebt
  "Value residue:"
    money valueResidue = (sellingPrice - privateDebt)
}
```

Summary

- ASTs: discard layout/comments
- Parse trees: contain all of it
  - high-fidelity transformation
  - “comments are part of the language too”
- Rascal:
  - parse trees, concrete matching, *Tree*

Conclusion

- Abstract syntax sucks ;)
- Anything you can do abstract, I can do concrete...
- ... and more.
- Abstraction is great, but has its cost.