UNIVERSITEIT VAN AMSTERDAM

Language Engineering

Software Construction 2018

Dr. Vadim Zaytsev aka @grammarware

raincode

LABS

compiler experts

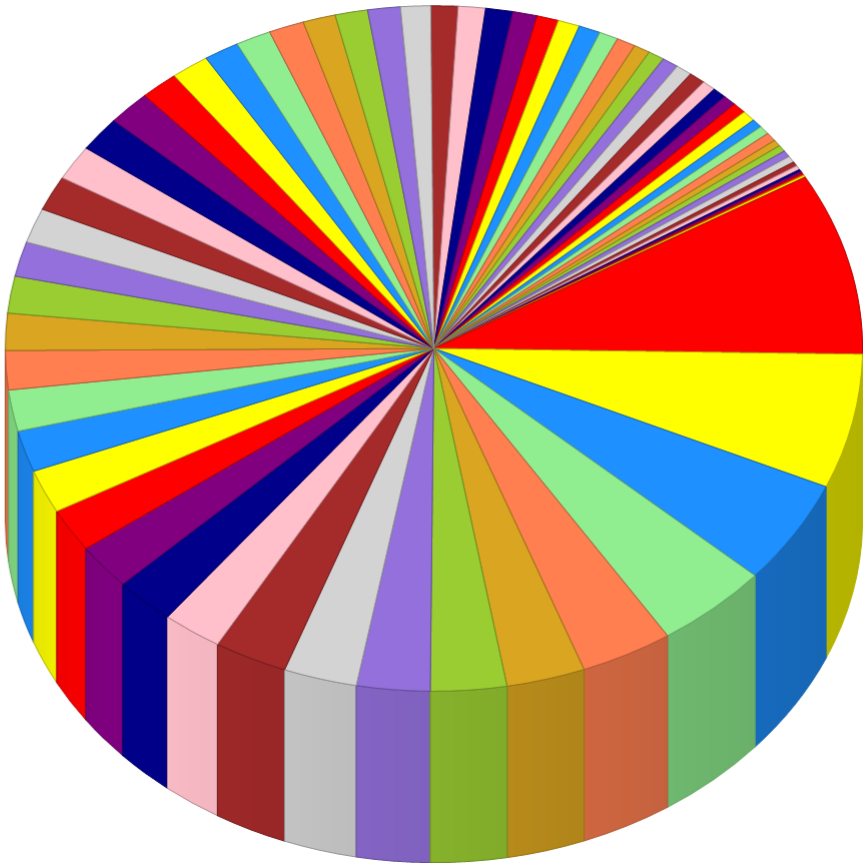
The most expensive is spelling

- Maintenance (12x)
- Maintenance. (3x)
- Maintanance (4x)
- Maintenence (1x)
- Maintainance (4x)
- Maintentance (1x)
- The maintenance (1x)
- Neglecting maintenance (1x)
- Testing / debugging (3x)
- Code programming (1x)

Double+ the number of commits

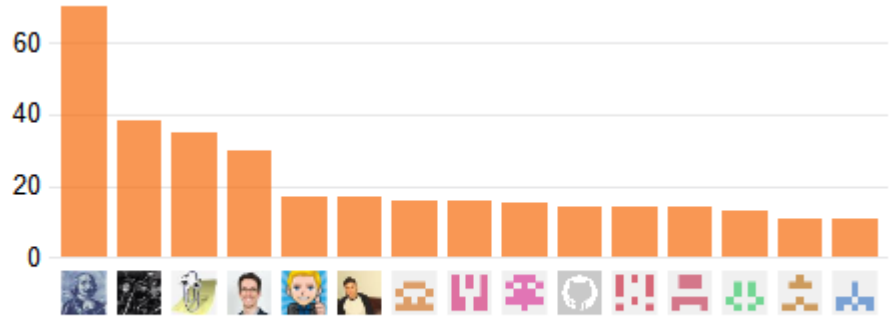
977 Commits

84 Nico Tromp
62 lauwdude
48 rmathijn
42 hasan
34 Peter Takacs
29 Remi van Veen
28 Simon Schneider
27 Cornelius Ries
27 GrimGerbil
27 Mihai Onofrei
22 Elias
22 carlyhill6895
21 rashadaoud
20 Metchu
20 Niels Boerkamp
19 Edwin
19 George Vletsas
18 Jordy Bottelier
17 Herczeg
17 Jorick van Rhenen
16 Joanna Roczniak
16 Jovan Maric
16 Meess
16 Tim Nederveen
16 Unknown
16 ighmelene
14 DennisvanderWerf
14 Jouke
13 Deepa Karra
13 Michael de Lang
13 porke
12 Dylan Bartels
12 Thijs Klaver
12 bicker
11 Scoudem
10 Jaap Koetsier
10 Joana Correia Magalhães Sousa
10 Nick
9 olimoli9160
8 V-Jong
8 bramo
8 jewelEarthDeveloper
7 TerryvanWalen
7 evanscharrenburg
6 Dennis Kruidenberg
6 MichielBoswijk
6 Sangam
6 Stevan Pavlicic
6 tdobber
5 Hector Stenger
5 Laurens de Gilde
5 Leó Gunnar vidisson
5 Tim
5 sara
4 Ioana Mirică

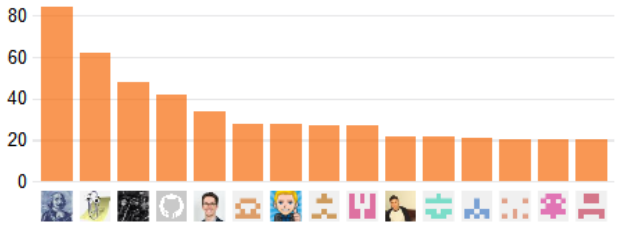


100kLOC; Python, TS, C# on the rise

Excluding merges, **61 authors** have pushed **523 commits** to master and **551 commits** to all branches. On master, **1,438 files** have changed and there have been **51,379 additions** and **21,217 deletions**.



Excluding merges, **70 authors** have pushed **946 commits** to master and **976 commits** to all branches. On master, **1,558 files** have changed and there have been **102,663 additions** and **0 deletions**.



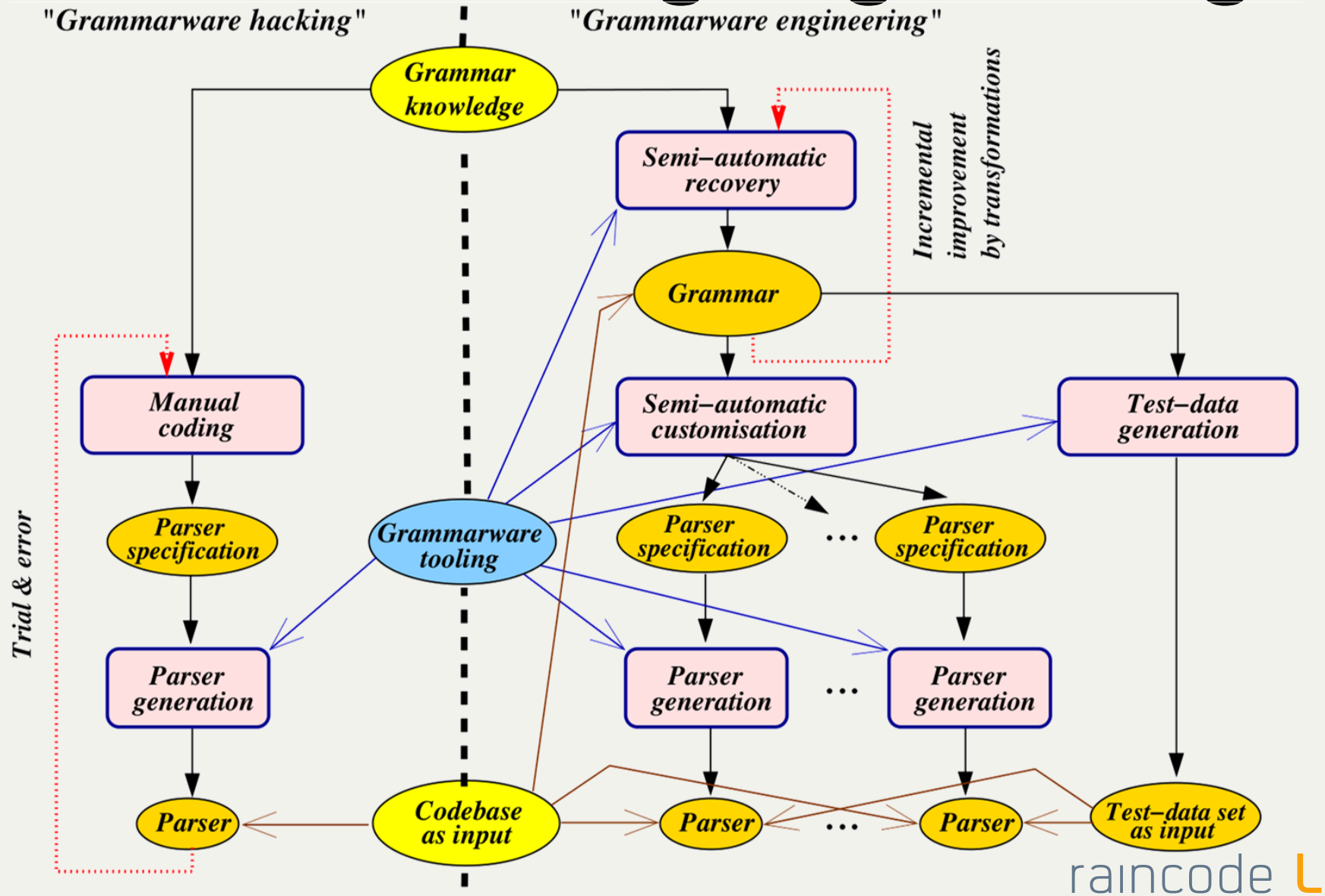
raincode LABS
compiler experts

Software Language Engineering

- engineering of software languages and related artefacts
- engineering of languages and their tools, as software
- discipline of engineering languages and their tools required for the creation of software

https://github.com/slebok/slebok/blob/master/terms/software%20language%20engineering.md

Grammarware hacking/engineering

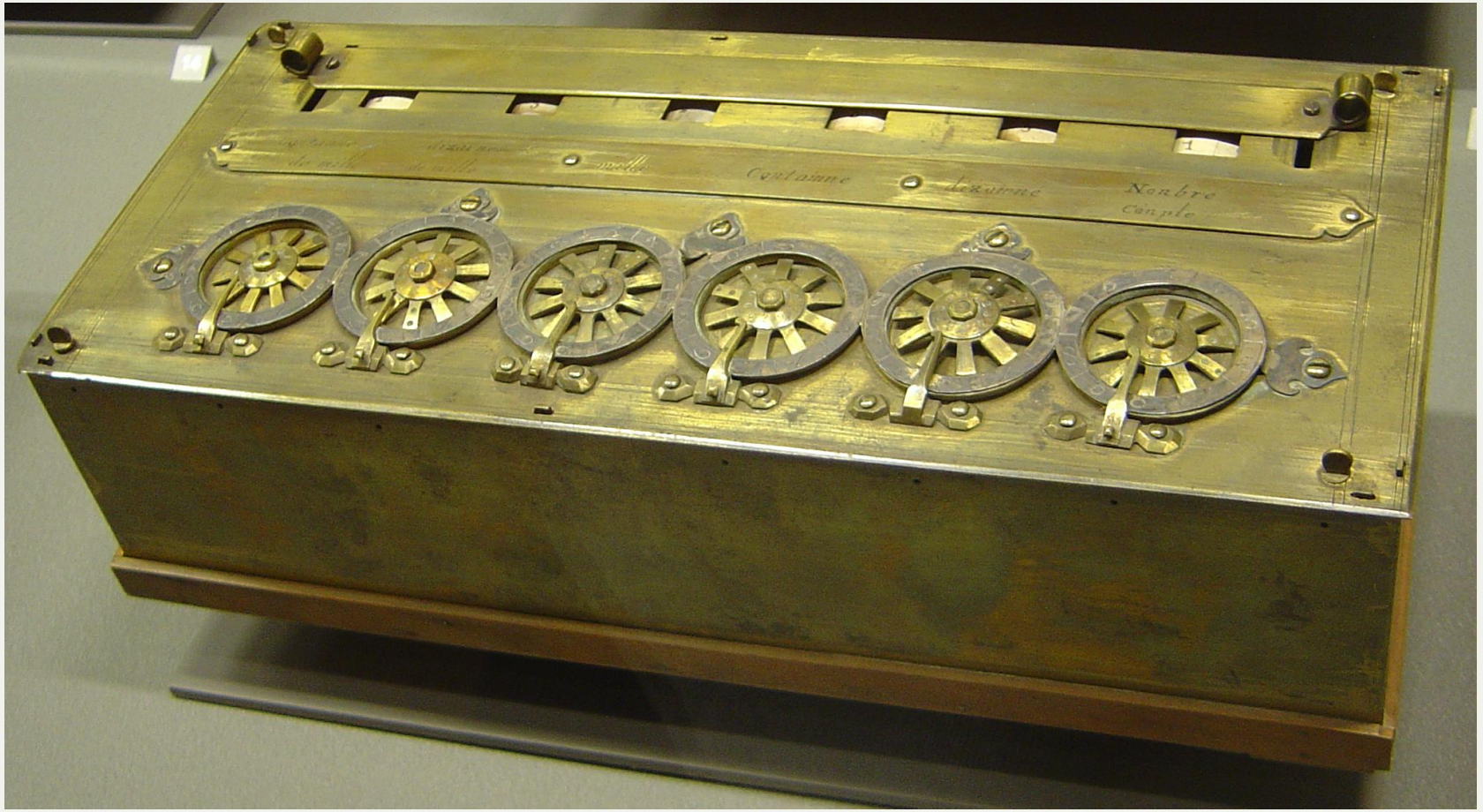


Klint, Lämmel, Verhoef, *Toward an Engineering Discipline for Grammarware*, ToSEM 2005.

SLE is multidisciplinary

- Grammarware
- Domain-Specific Languages
- Model-Driven Engineering
- Bidirectional Transformations
- XML
- Metaprogramming
- Generative Programming
- …

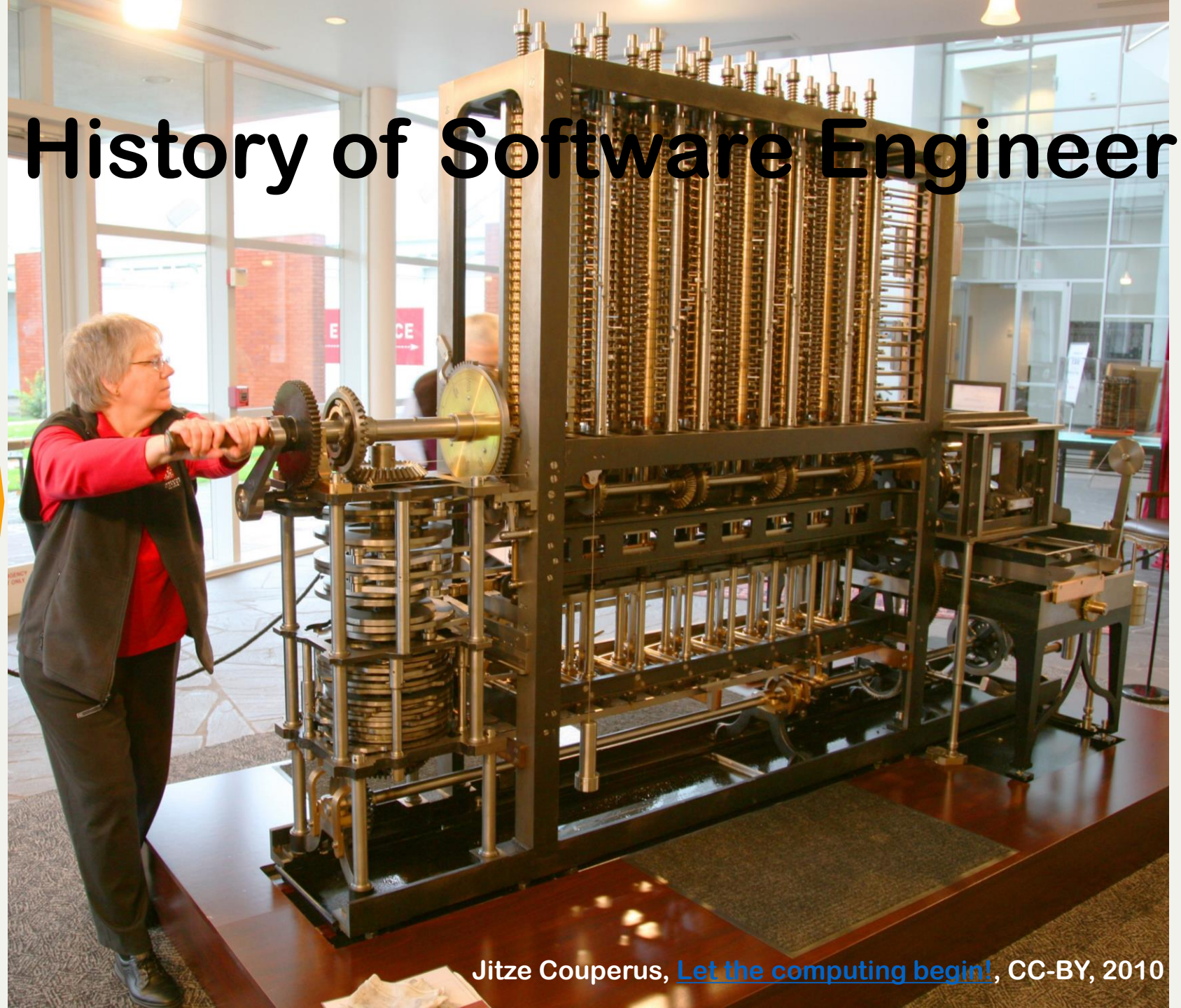
History of Software Engineering: 1642



David Monniaux, Arts et Metiers Pascaline, CC-BY-SA, 2005

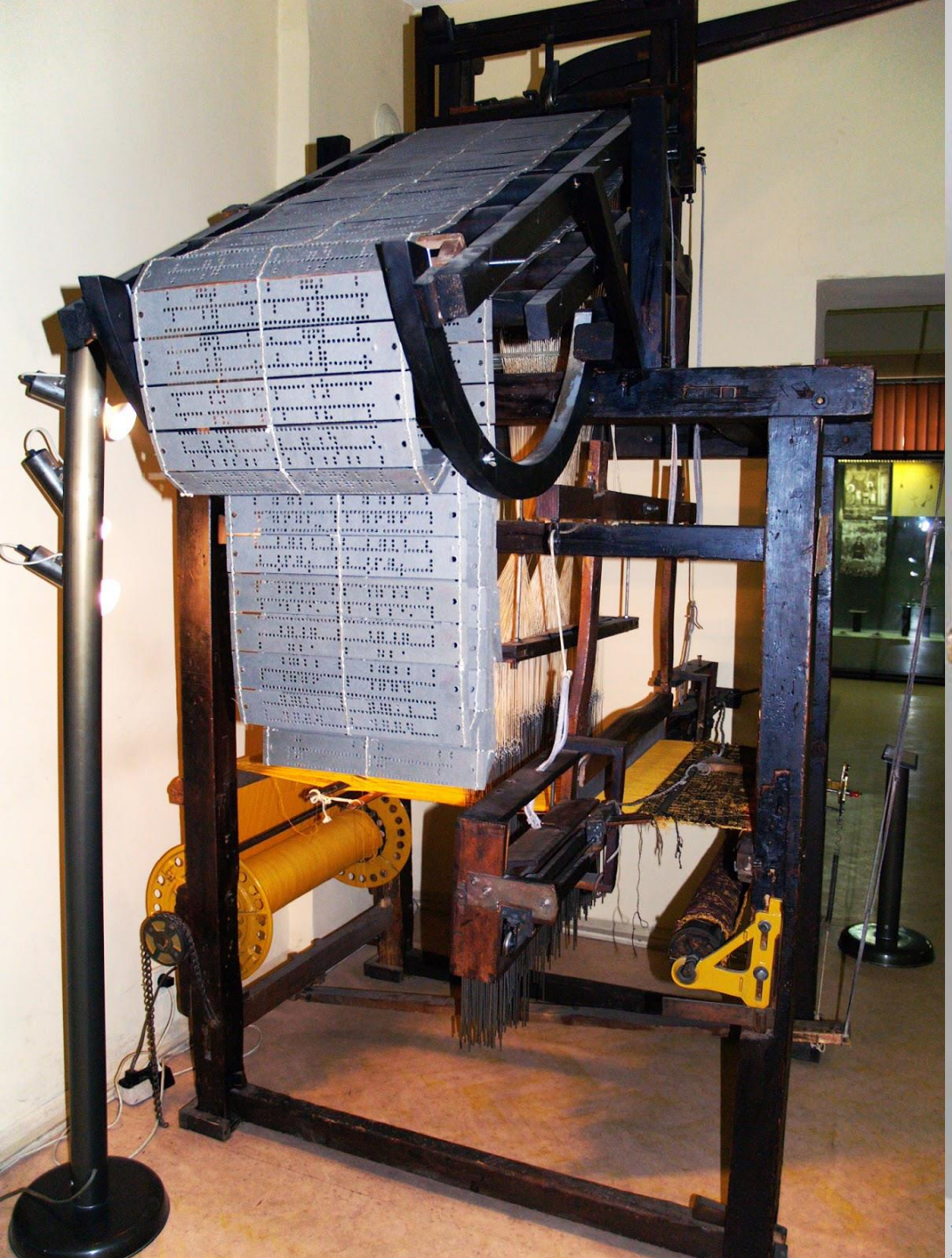
raincode LABS
compiler experts

History of Software Engineering: 1822



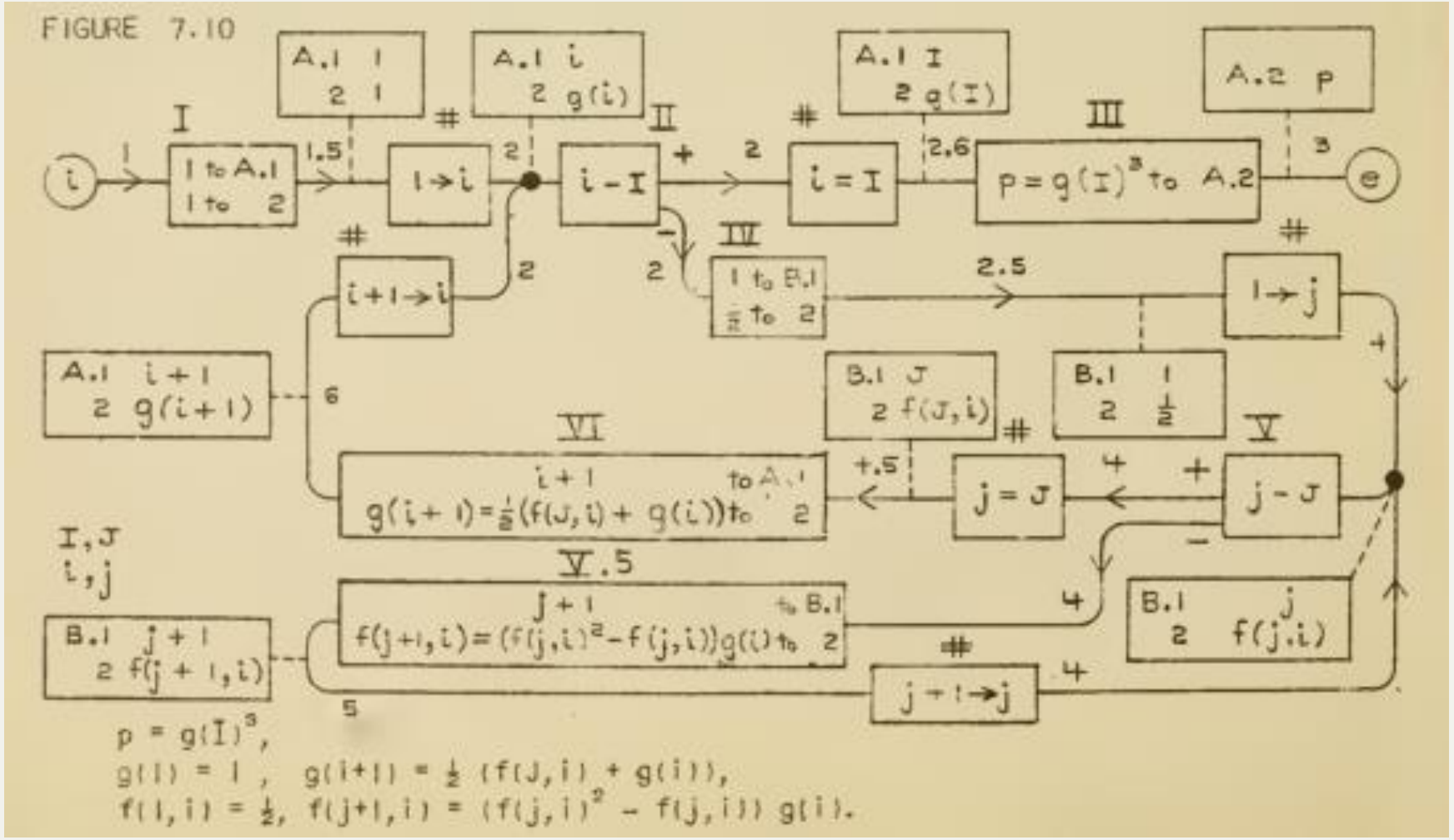
Jitze Couperus, Let the computing begin!, CC-BY, 2010

raincode LABS
compiler experts



https://commons.wikimedia.org/wiki/File:Hand-driven-jacquard-loom.jpg (CC-BY-SA, Edal Anton Lefterov)
https://commons.wikimedia.org/wiki/File:M0354_1951-40-004_2.jpg (CC-BY-SA, Jacques Monnin)

History of Software Engineering: 1947



Goldstine, von Neumann, *Planning and Coding of Problems for an Electronic Computing Instrument*, IAS 1947

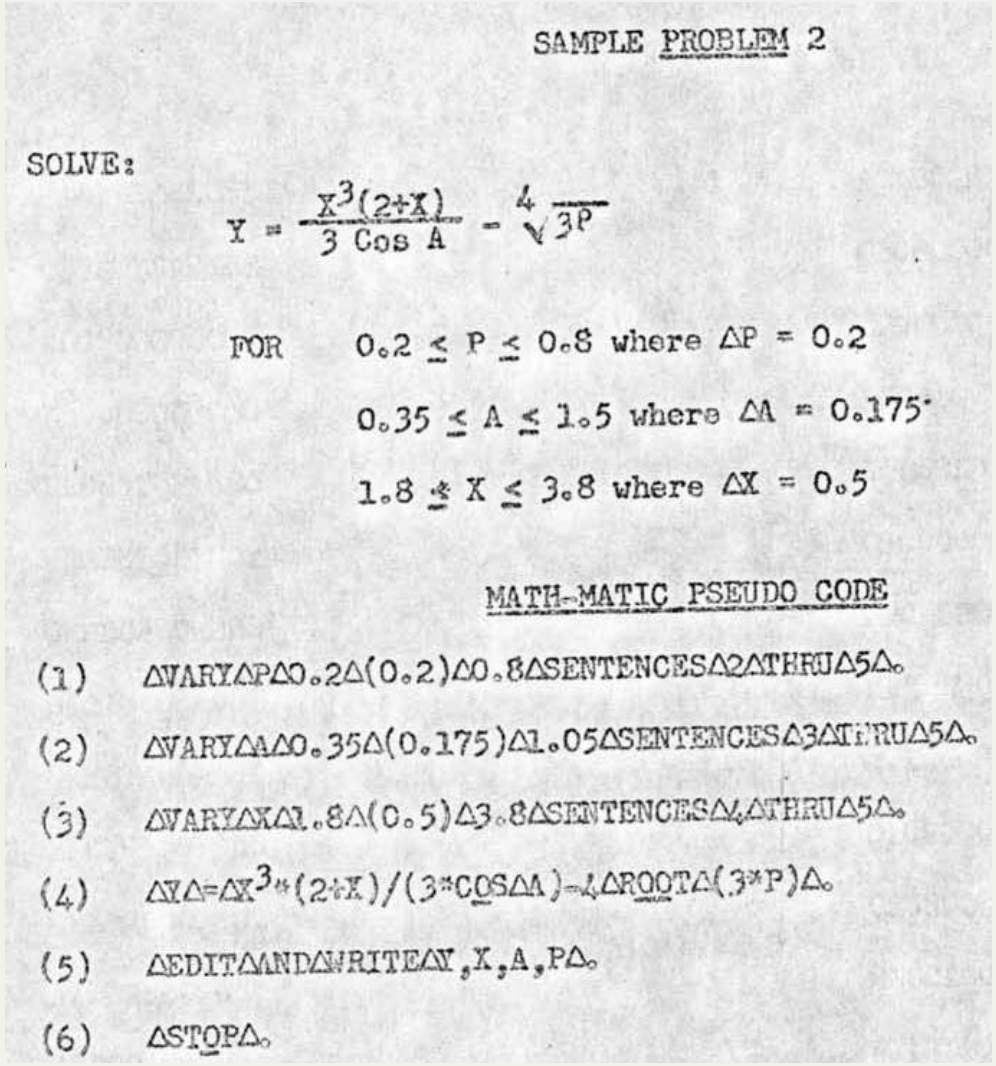
raincode LABS
compiler experts

History of Software Engineering: 1950

		Equations	Coded representation
00		i = 10	00 00 00 W0 03 Z2
01	0:	y = (√ abs t) + 5 cube t	T0 02 07 Z5 11 T0
02			00 Y0 03 09 20 06
03		y 400 if≤to 1	00 00 00 Y0 Z3 41
04		i print, 'TOO LARGE' print-and-return	00 00 Z4 59 W0 58
05		0 0 if=to 2	00 00 00 Z0 Z0 72
06	1:	i print, y print-and-return	00 00 Y0 59 W0 58
07	2:	T0 U0 shift	00 00 00 T0 U0 99
08		i = i-1	00 W0 03 W0 01 Z1
09		0 i if≤to 0	00 00 00 Z0 W0 40
10		stop	00 00 00 00 ZZ 08

Mauchly, Schmitt, Short Code

History of Software Engineering: 1957



SAMPLE PROBLEM 2

SOLVE:

$$Y = \frac{X^3(2+X)}{3 \cos A} - \sqrt[4]{3P}$$

FOR $0.2 \leq P \leq 0.8$ where $\Delta P = 0.2$

$0.35 \leq A \leq 1.5$ where $\Delta A = 0.175$

$1.8 \leq X \leq 3.8$ where $\Delta X = 0.5$

MATH-MATIC PSEUDO CODE

(1) ΔVARYΔPΔ0.2Δ(0.2)Δ0.8ΔSENTENCESΔ2ΔTHRUΔ5Δ.

(2) ΔVARYΔAΔ0.35Δ(0.175)Δ1.05ΔSENTENCESΔ3ΔTHRUΔ5Δ.

(3) ΔVARYΔXΔ1.8Δ(0.5)Δ3.8ΔSENTENCESΔ4ΔTHRUΔ5Δ.

(4) ΔYΔ=ΔX3*(2+X)/(3*COSΔA)-4ΔROOTΔ(3*P)Δ.

(5) ΔEDITΔANDΔWRITEΔY,X,A,PΔ.

(6) ΔSTOPΔ.

```
(1) VARY P 0.2 (0.2) 0.8
        SENTENCES 2 THRU 5.
(2) VARY A 0.35 (0.175) 1.05
        SENTENCES 3 THRU 5.
(3) VARY X 1.8 (0.5) 3.8
        SENTENCES 4 THRU 5.
(4) Y = X³*(2+X)/(3*COS A)
        - 4 ROOT (3*P).
(5) EDIT AND WRITE Y, X, A, P.
(6) STOP.
```

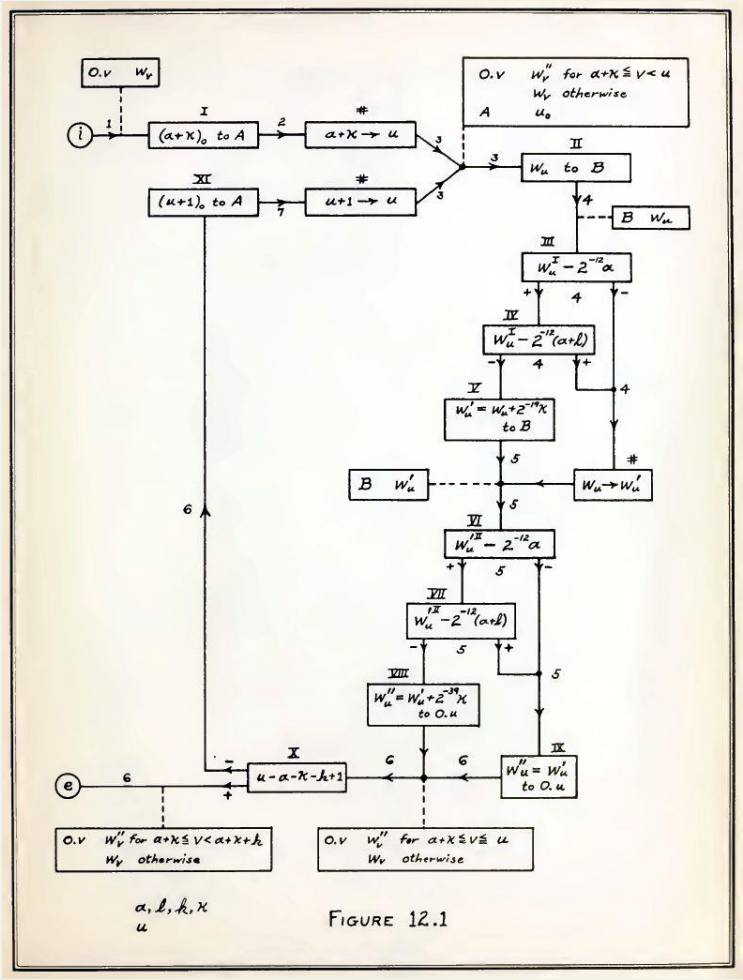
Ash, Broadwin, Dell Valle, Greene, Jenny, Katz, Yu, *Preliminary Manual for MATH-MATIC and ARITH-MATIC System for Algebraic Translation and Compilation for UNIVAC I and II*, 1957

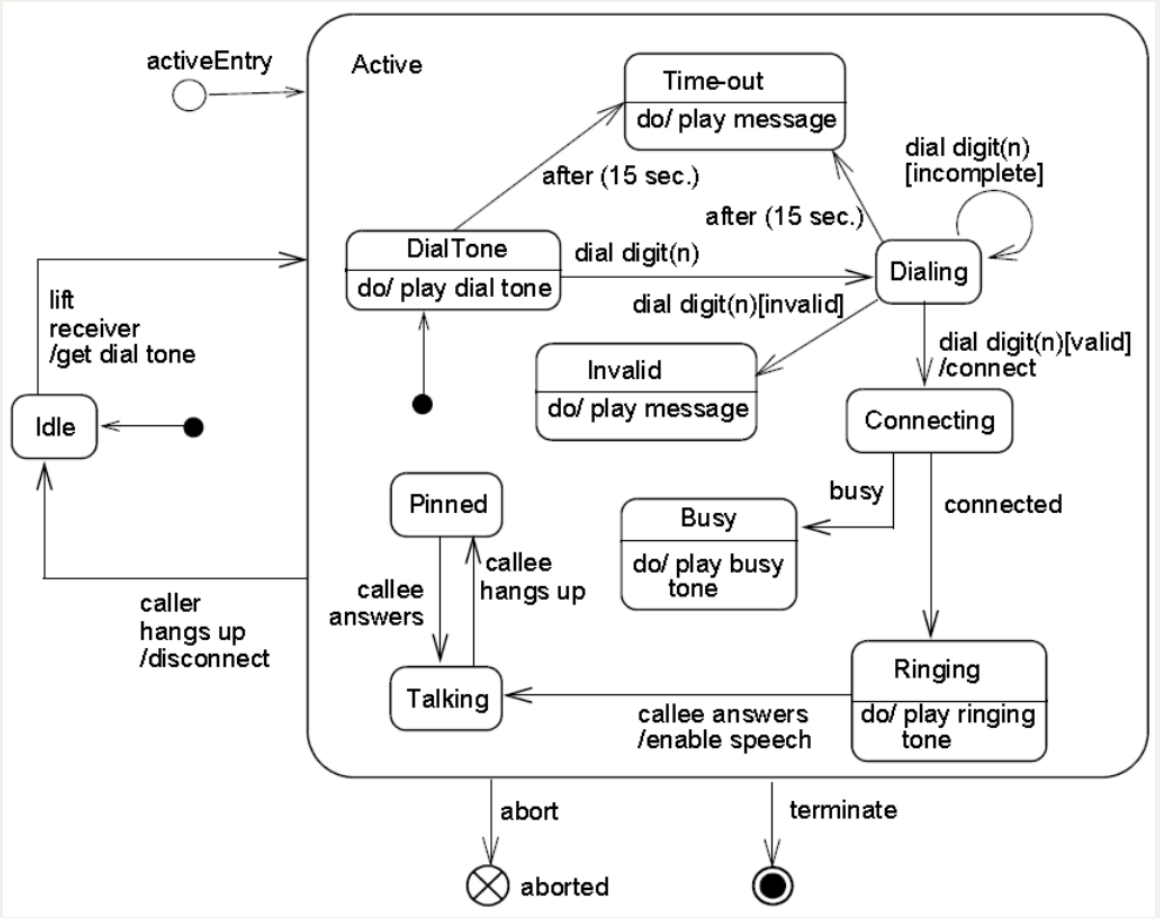
raincode LABS
compiler experts

History of Software Engineering: 1959

SI Neg. 83-14878. Date: na...Grace Murray Hopper at the UNIVAC keyboard, c. 1960. CC-BY-SA 2004

History of Software Engineering: 2015



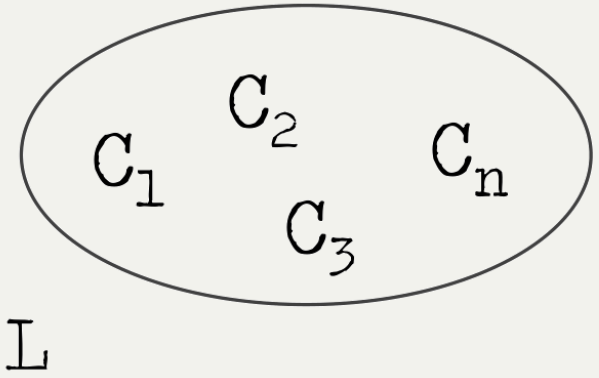


http://www.omg.org/spec/UML/2.5/ (Figure 14.36 from page 334)

raincode LABS
compiler experts

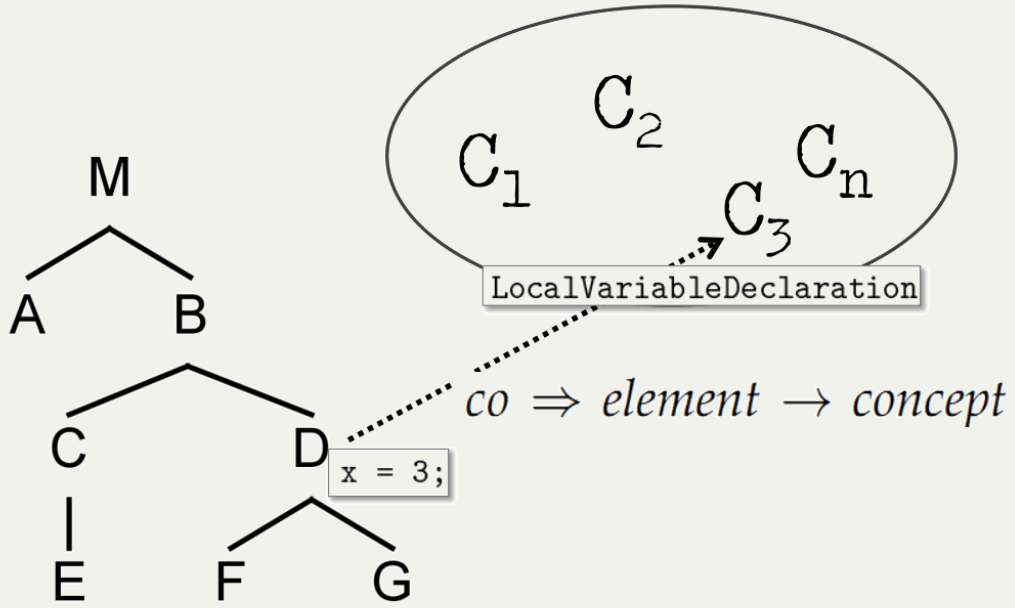
Languages are sets of concepts

Languages are
sets of concepts

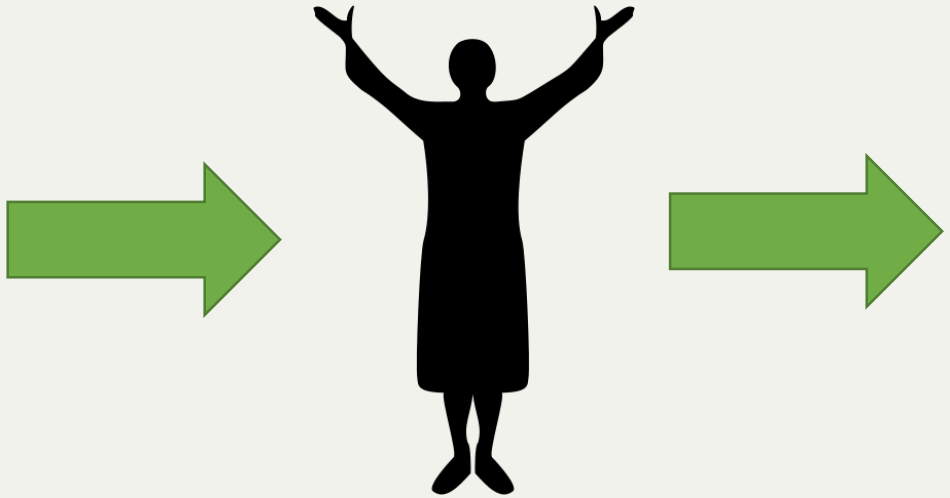


Languages are sets of concepts

Programs
and languages



Expectation vs Reality

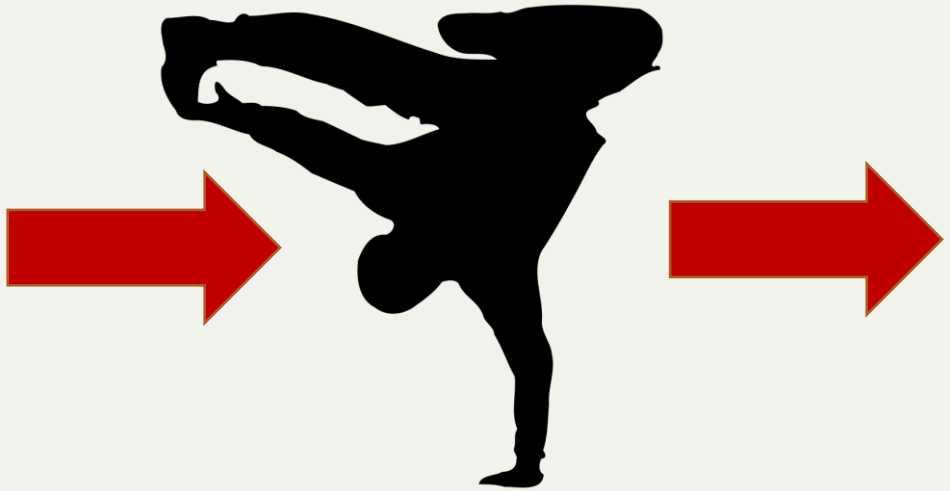


```
public class TypeChecker {
    private List<String> errors;
    private List<String> warnings;
    public TypeChecker() {
        this.errors     = new ArrayList<String>();
        this.warnings   = new ArrayList<String>();
    }
    public List<Identifier> checkIdentifiers(Form form)
    {
        ConflictingIdChecker checker = new ConflictingIdChecker(form, errors);
            return checker.getDuplicates();
    }
    public List<Identifier> checkReferences(Form form, Map<String,Type> symbolTable)
    {
        ReferenceChecker checker = new ReferenceChecker(form, symbolTable, errors);
            return checker.getUndefinedReferences();
    }
    public List<Expression> checkConditions(Form form, Map<String,Type> symbolTable)
    {
        ConditionChecker checker = new ConditionChecker(form, symbolTable, errors);
            return checker.getInvalidConditions();
    }
    public List<Operator> checkOperands(Form form, Map<String,Type> symbolTable)
    {
        OperandChecker checker = new OperandChecker(form, symbolTable, errors);
            return checker.getIllegalOperations();
    }
    public List<String> checkLabels(Form form)
    {
        DuplicateLabelChecker checker = new DuplicateLabelChecker(form, warnings);
            return checker.getDuplicates();
    }
    public boolean hasErrors() {
        return !errors.isEmpty();
    }
    public List<String> getErrors() {
        return errors;
    }
    public boolean hasWarnings() {
        return !errors.isEmpty();
    }
    public List<String> getWarnings() {
        return warnings;
    }
    public void printErrors() {
        for(String msg : errors) System.err.println("ERROR: " + msg);
    }
    public void printWarnings() {
        for(String msg : warnings) System.out.println("WARNING: " + msg);
    }
}
```

Domain **Programmer** **Code**

raincode LABS
compiler experts

Expectation vs Reality



Domain

Programmer

Code

Programming is lossy

- encoding
- obfuscating
- encrypting
- dispersing
- tangling
- distorting

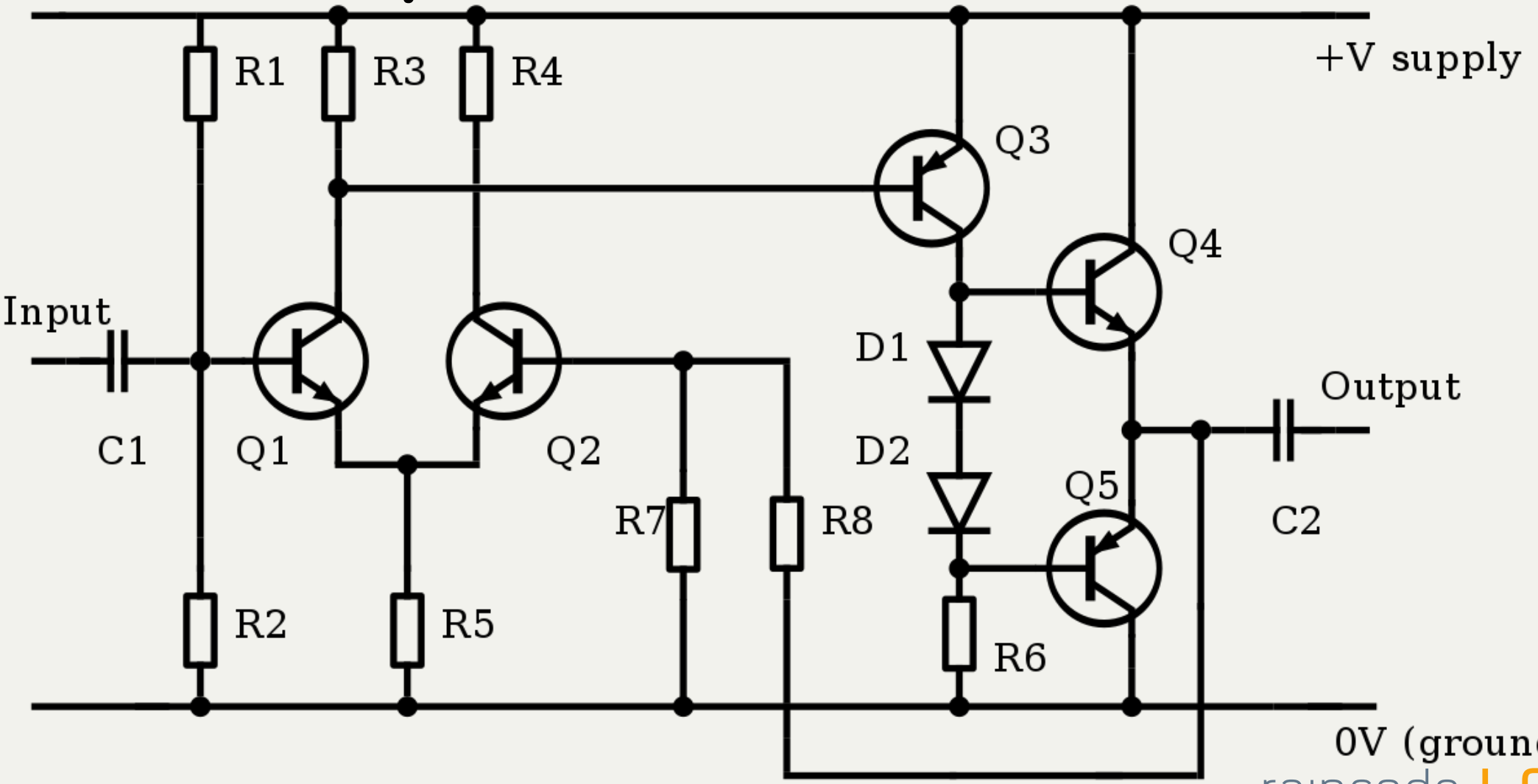
Quentin Tarantino, *Pulp Fiction*, 1994.

raincode LABS
compiler experts

Problems

- notations are unwieldy for domain experts
- lack of useful abstractions
- general-purpose analysis/verification/simulation limited
- too much code
- not enough documentation
- productivity is low
- maintenance is expensive
- portability is… somewhere

Domain-specific notations!



raincode LABS
compiler experts

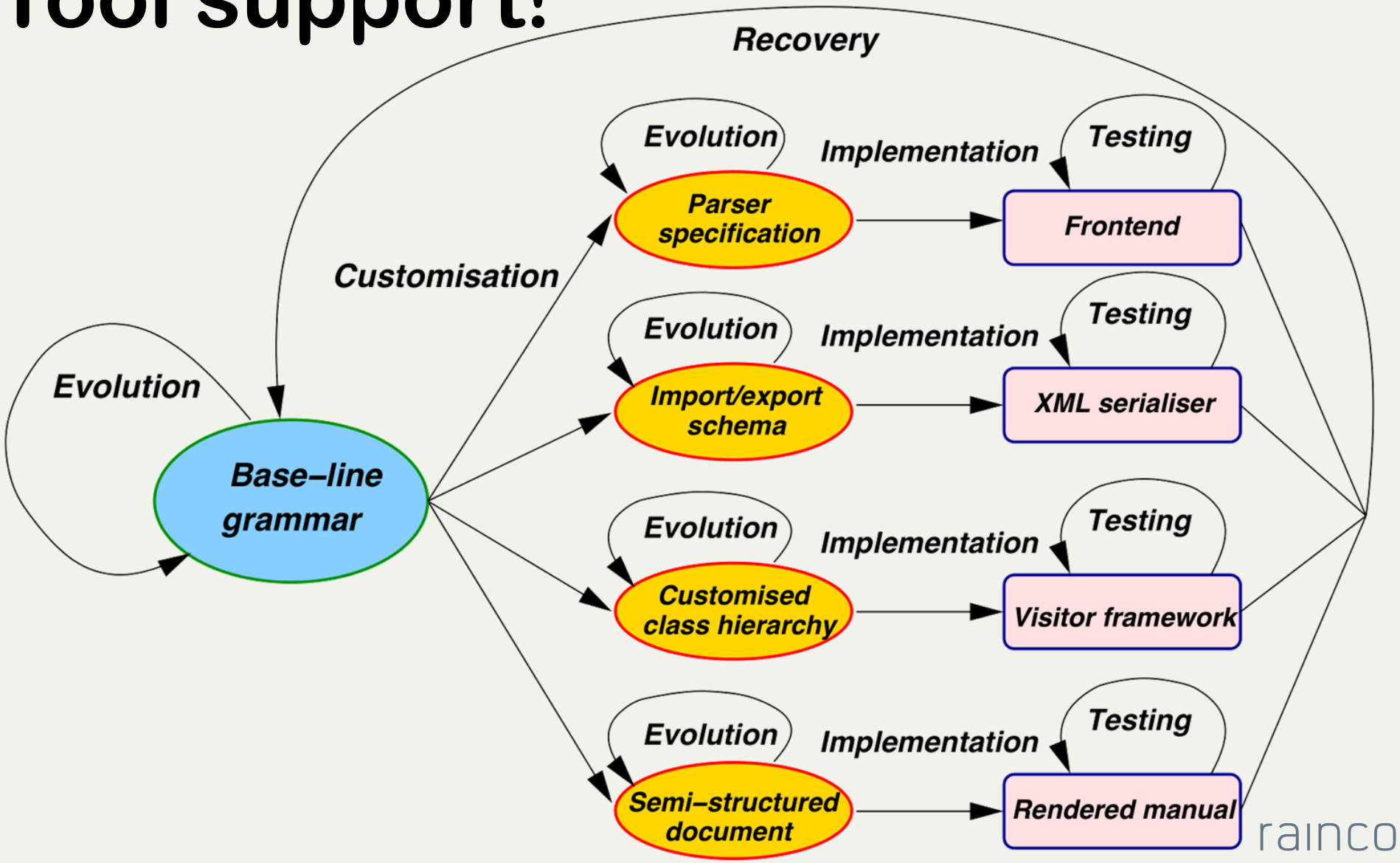
Abstractions fitting the domain!

```
 1  img {border:0;}
 2  .last {width:100%; text-align:right;}
 3  .last img {border:0; width:88px; height:31px;}
 4  br {clear:left;}
 5  div.c {text-align:center;}
 6  div.c div {border:1px solid black; display:inline-block;}
 7  div.c div img {vertical-align:middle;}
 8  .foto {width:100%; text-align:center; display:block;}
 9  a:hover {text-decoration:underline;}
10  body {background-color:#9C9; color:#033;}
11  a:link {color:#039;}
12  a:visited {color:#006;}
13  ul li {list-style:disc;}
14  ul li ul li, dl dd dl dd, .col {font-size:smaller;}
15  dl.boldies dt {font-weight:bold;}
16  .grey {color:#466;}
17  .red {color:#400;}
18  .grey .red a:link, .red a:link, .red a:visited, .grey .red a:visited {color:#633;}
19  .grey a:link {color:#46C;}
20  .grey a:visited {
21  .abox {background-color:#0FF;}
```

Tool support!

raincode LABS
compiler experts

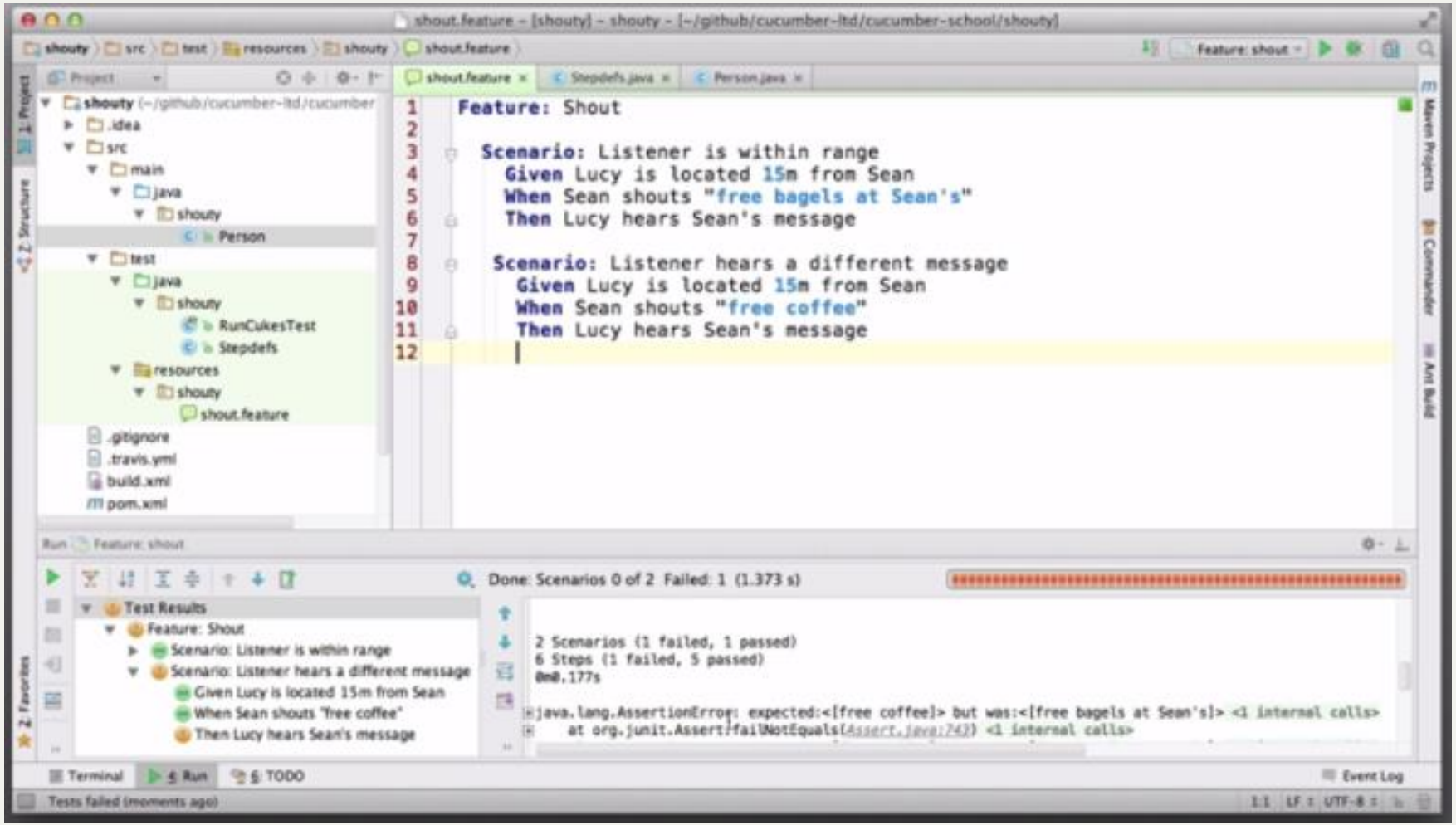
Tool support!



Klint, Lämmel, Verhoef, *Toward an Engineering Discipline for Grammarware*, ToSEM 2005.

raincode LABS
compiler experts

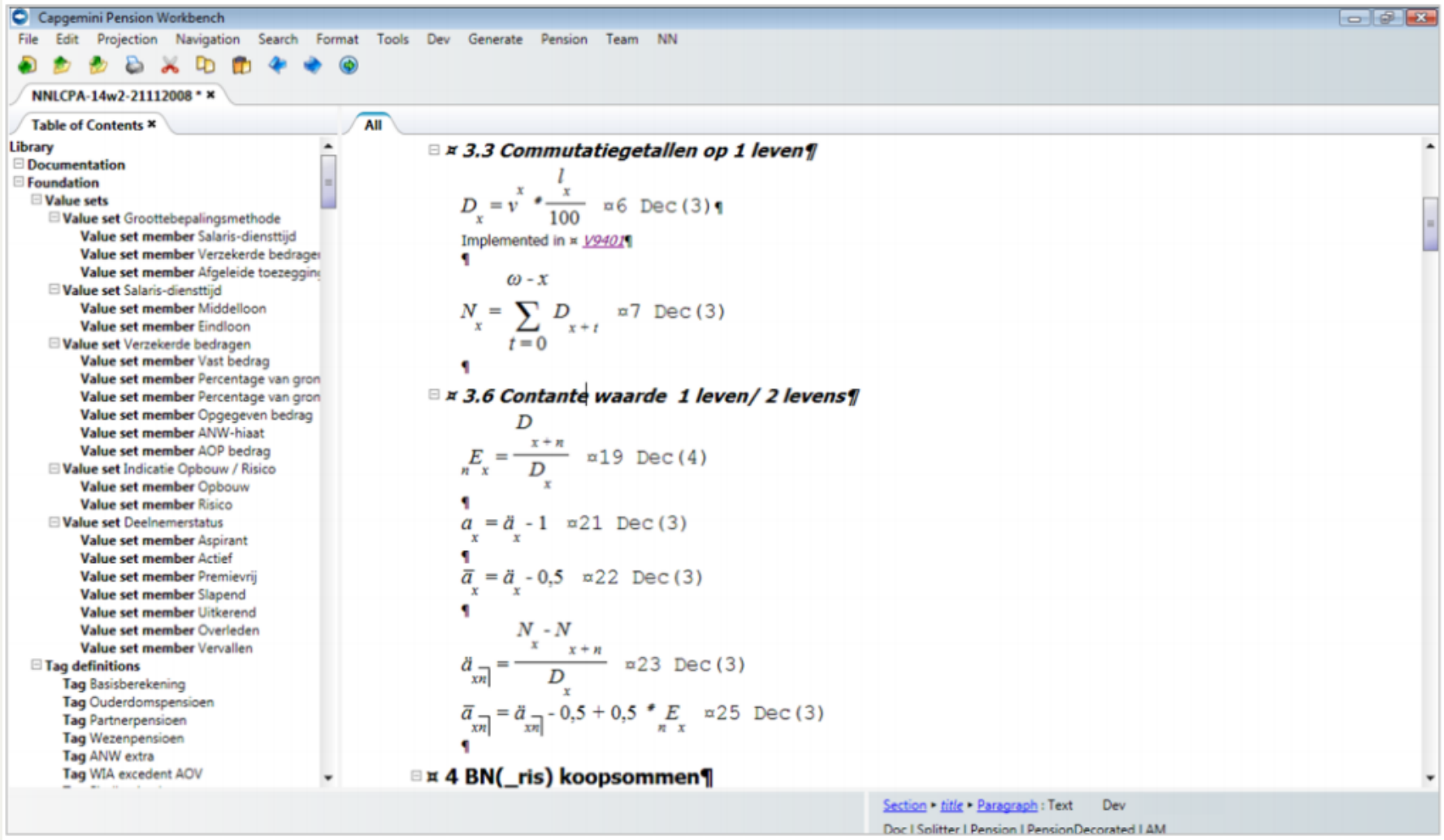
Readable! Concise!



https://cucumber.io/school

raincode LABS
compiler experts

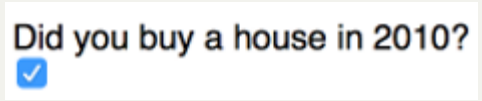
Productive! Comfortable!



Markus Völter, https://github.com/slecourse/slecourse/wiki/dslDesign

raincode LABS
compiler experts

Portability follows abstraction level

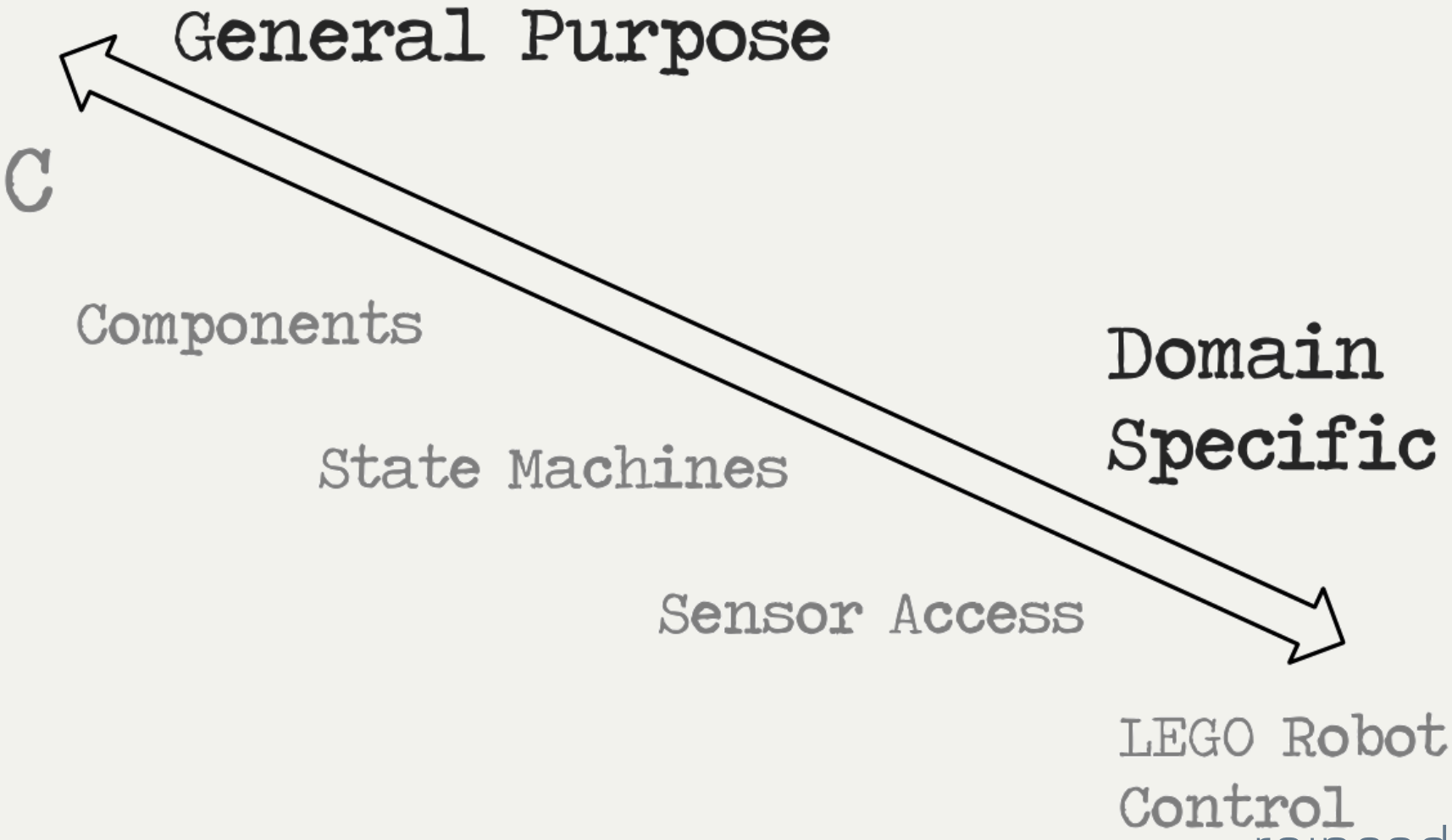
- “Open a window” is portable
  - new JFrame(…) … .getContentPane().add(…) … .setVisible(true)
- “\section{…}” is portable
  - Arial 12pt Semi-bold Italics … numbering … table of contents
- “‘Did you sell a house?’ soldHouse: boolean” is portable
  - 

 is not
- “all: paper.pdf // %.pdf: %.bib // pdflatex $*” is portable
  - .sh / .bat / .exe are not

Classify languages

- Makefiles
- Java
- UML in MDA [link]
- UML on a whiteboard
- HTML and CSS
- MediaWiki
- English
- Controlled natural language [link]

GPL-DSL is a spectrum



Markus Völter, https://github.com/slecourse/slecourse/wiki/dslDesign

Internal-External

- External: standalone and base language-independent
  - use its own syntax
  - better IDE support
  - make, Ant, ANTLR, yacc, XPath, SQL, sed, awk, CSS, …
- Internal: embedded in the host language
  - no special tools required
  - shared syntax means intermixability
  - limited expressivity, all host constraints inherited
  - popular in LISP, Scheme, Ruby, Smalltalk, JavaScript
  - jQuery, pypy, rake, rspec, Parsec, PetitParser
- Embedded

Internal DSL example (parsec)

Rather than using the highly parameterized *ParsecT* type, we specialize the types of the parser combinators readily to *P* as used in the present contribution.

```
-- Sequential composition
(>>=) :: P a -> (a -> P b) -> P b

-- Sequential composition; forget first result
(>>) :: P a -> P b -> P b

-- Predictive choice
(<|>) :: P a -> P a -> P a

-- EBNF "*"
many :: P a -> P [a]
```

Taxonomy for internal, external and embedded languages

	Syntax	Semantics	Host Integration	Tool Integration	Description
Internal	○	○	●	●	Internal languages make a creative use of the host language. They integrate seamlessly into the host language and tools, but their syntax and semantics is strictly constrained.
External	●	●	○	○	External languages are independent of the host language. This makes them difficult to integrate into the host language and development tools.
Embedded	●	●	●	●	Embedded languages combine the advantages of internal and external languages. Ideally an embedded language uses the same executable representation as the host and integrates well with tools.

Lukas Renggli, *Dynamic Language Embedding*, PhD thesis, 2010
By: Ralf Lämmel, http://softlang.wikidot.com/course:sle1718

Homogeneous-Heterogeneous

```
module CounterExample from counterd imports nothing {

  var int theI;

  var boolean theB;

  var boolean hasBeenReset;

  statemachine Counter {
    in start() <no binding>
       step(int[0..10] size) <no binding>
    out someEvent(int[0..100] x, boolean b) <no binding>
        resetted() <no binding>
    vars int[0..10] currentVal = 0
         int[0..100] LIMIT = 10
    states (initial = initialState)
      state initialState {
        on start [ ] -> countState { send someEvent(100, true && false || true); }
      }
      state countState {
        on step [currentVal + size > LIMIT] -> initialState { send resetted(); }
        on step [currentVal + size <= LIMIT] -> countState { currentVal = currentVal + size; }
        on start [ ] -> initialState {  }
      }
  } end statemachine

  var Counter c1;

  exported test case test1 {
    initsm(c1);
    assert(0) isInState<c1, initialState>;
    trigger(c1, start);
    assert(1) isInState<c1, countState>;
  } test1(test case)
}
```

Heterogeneous

C

Statemachines

Testing

Example

Exten
ded C

Ralf Lämmel, http://www.softlang.org/book, Internal DSL style
Lukas Renggli, *Dynamic Language Embedding*, PhD thesis, 2010

Internal DSL implementation patterns

```
Computer c = new Computer();
Processor p = new Processor();
p.setCores(2);
p.setType(Processor.Type.i386);
Disk d = new Disk();
d.setSize(150);
p.getDisks().add(d);
c.getProcessors().add(p);
```

Imperative Function Sequence

```
new Computer(
 new Processor(2, Processor.i386,
  new Disk(150, Disk.UNK_RPMS)));
```

Function Nesting

```
['computer',
 ['processor',
  ['cores', 2],
  ['type', '386']],
 ['disk',
  ['size', 150]]]
```

Literal Sequence

```
Computer c =
 computer()
 .processor()
  .cores(2)
  .type(386)
  .end()
 .disk()
  .size(150)
  .end()
 .end();
```

Fluent / Chaining

```
Computer c = new Computer();
Processor p = new Processor(2, Processor.i386);
p.getDisks().add(new Disk(150, Disk.UNK_RPMS));
c.getProcessors().add(p);
```

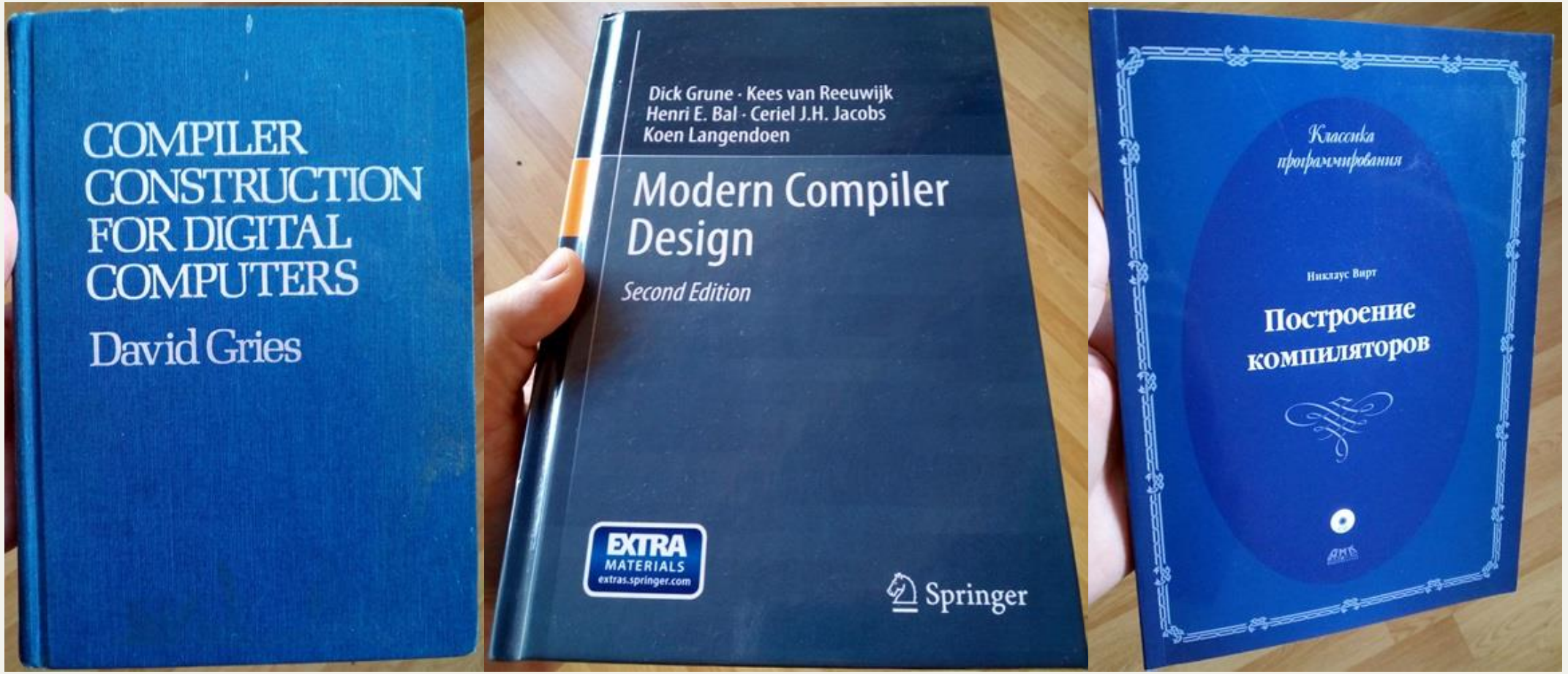
Functional Constructors

Languages support for styles

	C	C++	C#	Java	Javascript	Lisp	Haskell	Ruby	Smalltalk
2.1.1 Function Sequence	●	●	●	●	●	●	●	●	●
2.1.2 Function Nesting	●	●	●	●	●	●	●	●	●
2.1.3 Function Chaining	●	●	●	●	●	●	●	●	●
2.1.4 Higher-Order Functions	◐	◐	◐	◐	●	●	●	●	●
2.1.5 Language Literals	●	●	●	●	●	●	●	●	●
2.1.6 Operator Overloading	○	●	●	○	○	●	●	●	●
2.1.7 Meta-Annotations	○	○	●	●	○	◐	○	○	●
2.1.8 Program Generation	○	◐	◐	○	○	●	◐	○	◐
2.1.9 Macro Programming	◐	◐	○	○	○	●	◐	○	◐

Lukas Renggli, *Dynamic Language Embedding*, PhD thesis, 2010

External DSL implementation patterns



Ralf Lämmel, http://www.softlang.org/book,
External DSL style: https://youtu.be/MxJXVRqg1aI

raincode LABS
compiler experts

Embedded DSL implementation choices

- Extensible compilers
  - toolboxes with entry points to add to the toolchain
- Meta-programming systems
  - MetaOCaml, not MPS
- Language workbenches
  - Rascal, Spoofax, MPS
- Language transformations
  - ASF+SDF, TXL
- Modelling languages
  - Xtext, Kermeta, MetaEdit+

MetaOCaml

```
let even n = (n mod 2) = 0;;
let square x = x * x;;
let rec powerS n x =
  if n = 0 then .<1>.
  else if even n
       then .<square .~(powerS (n/2) x)>.
       else .<.~x *  .~(powerS (n-1) x)>.;;

let power5 = !. .<fun x -> .~(powerS 5 .<x>.)>.;;
```

Yannis Smaragdakis, GTTSE 2015 Tutorial

Lukas Renggli, *Dynamic Language Embedding*, PhD thesis, 2010

Expression Problem

	Const	Add	Neg
print()			
eval()			
count()			

Philip Wadler, *The Expression Problem*, 1998

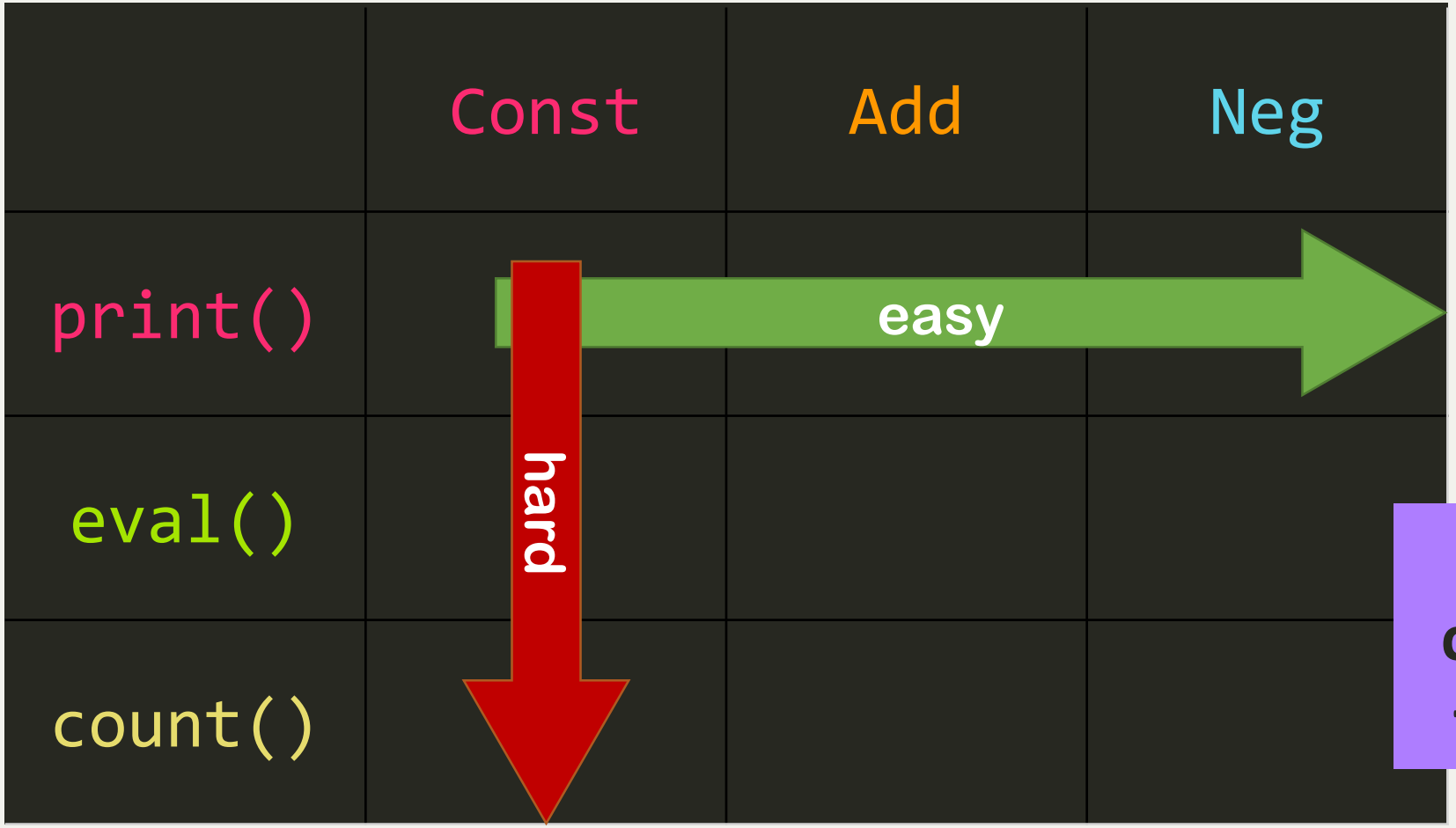
Java

```
public interface Exp {}
public class Const implements Exp {private int info;}
public class Add implements Exp {private Exp left, right;}
```

```
public interface Exp
   { public abstract String print(); }
public class Const implements Exp
   { private int info;
     @Override public String print()
       { return Integer.toString(info); } }
public class Add implements Exp
   { private Exp left, right;
     @Override public String print()
       { return left.print() + " + " + right.print(); } }
```

```
public class Neg implements Exp
   { private Exp operand;
     @Override public String print()
       { return "-" + operand.print(); } }
```

Expression Problem



Haskell

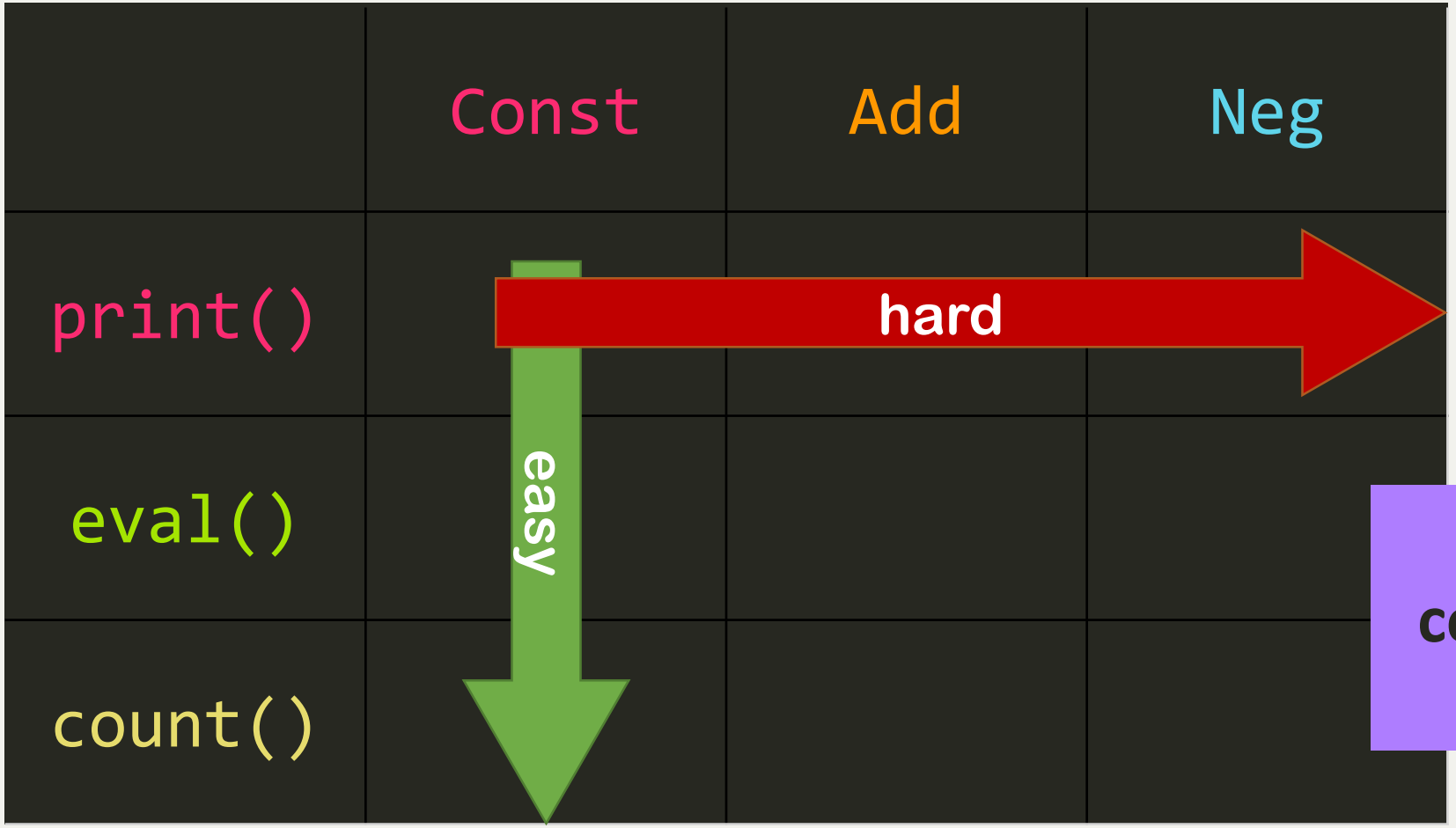
```
data Expr = Const Int
          | Add Expr Expr
```

```
pprint :: Expr -> String
pprint (Const i) = show i
pprint (Add l r) = pprint l ++ " + " ++ pprint r
```

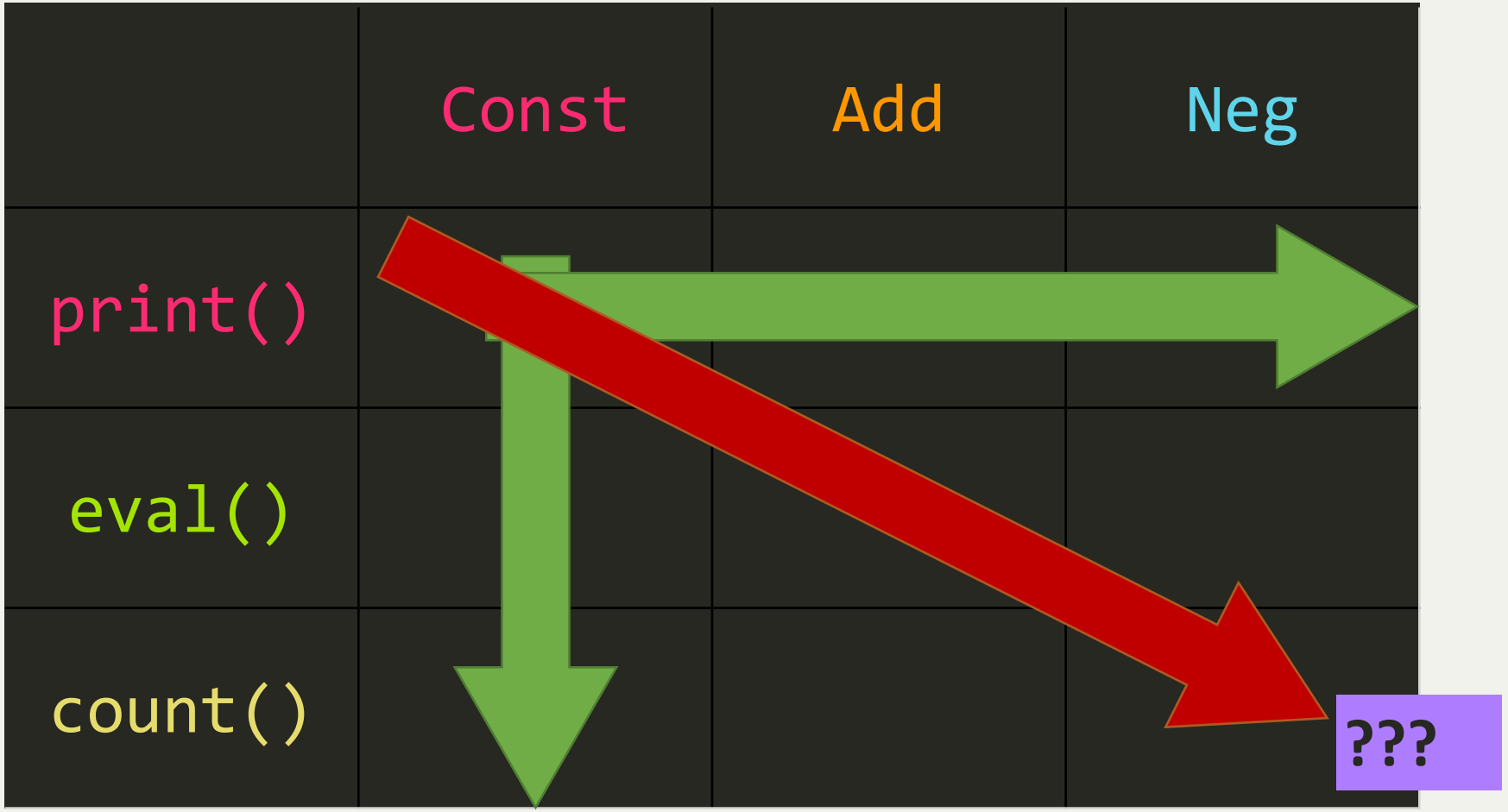
```
evaluate :: Expr -> Int
evaluate (Const i) = i
evaluate (Add l r) = evaluate l + evaluate r
```

```
countOps :: Expr -> Int
countOps (Const i) = 0
countOps (Add l r) = 1 + countOps l + countOps r
```

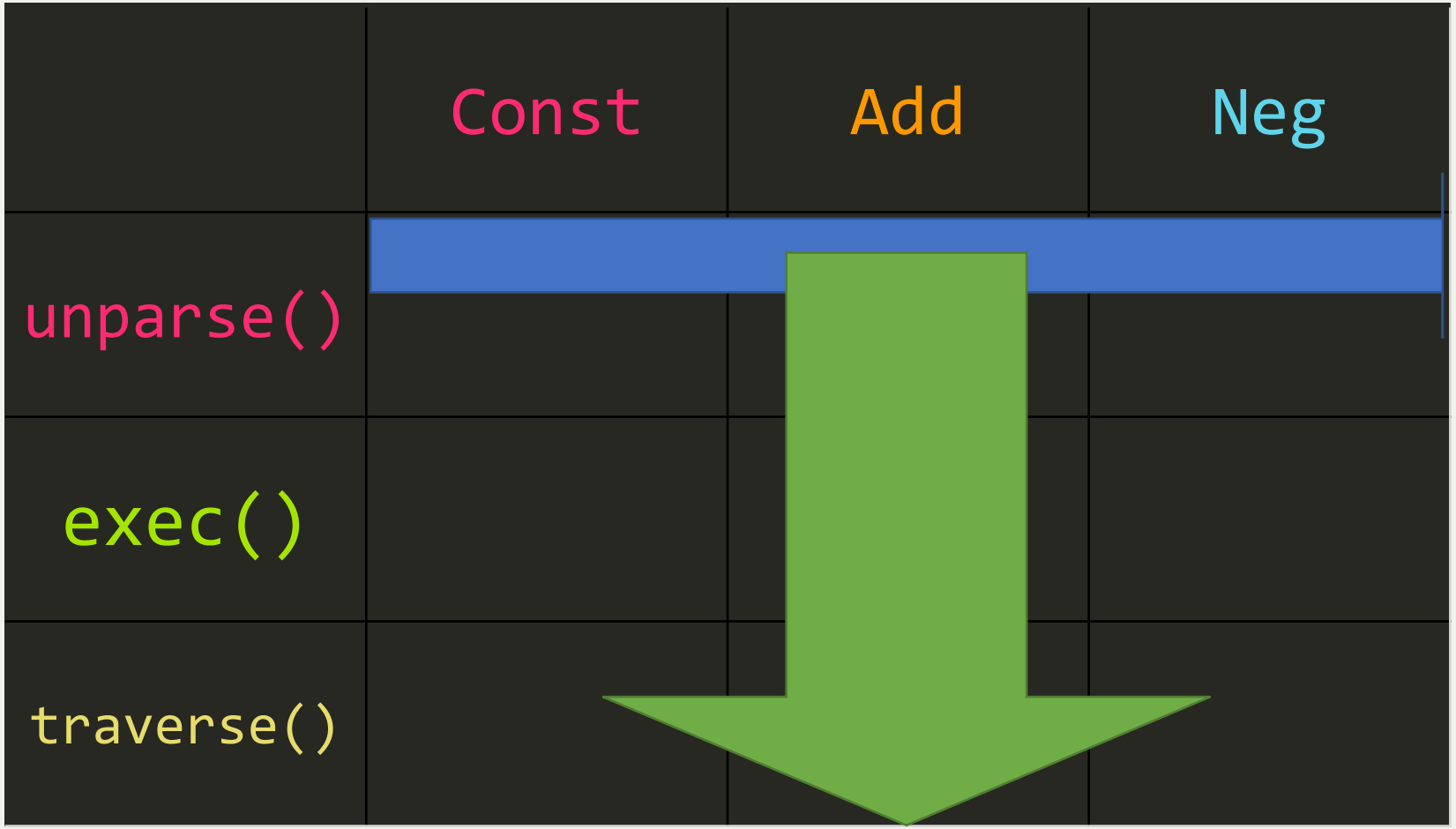
Expression Problem



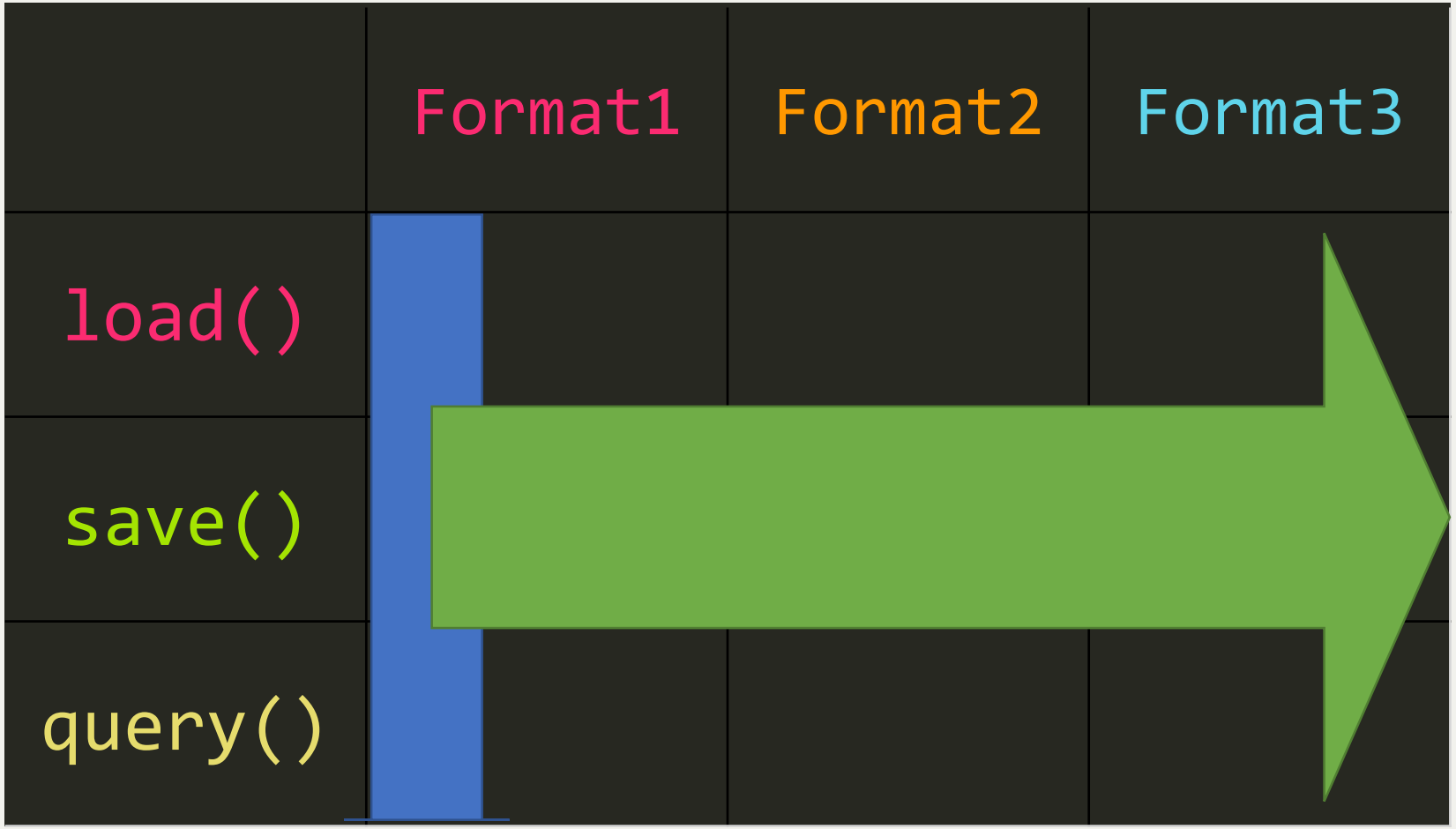
Expression Problem



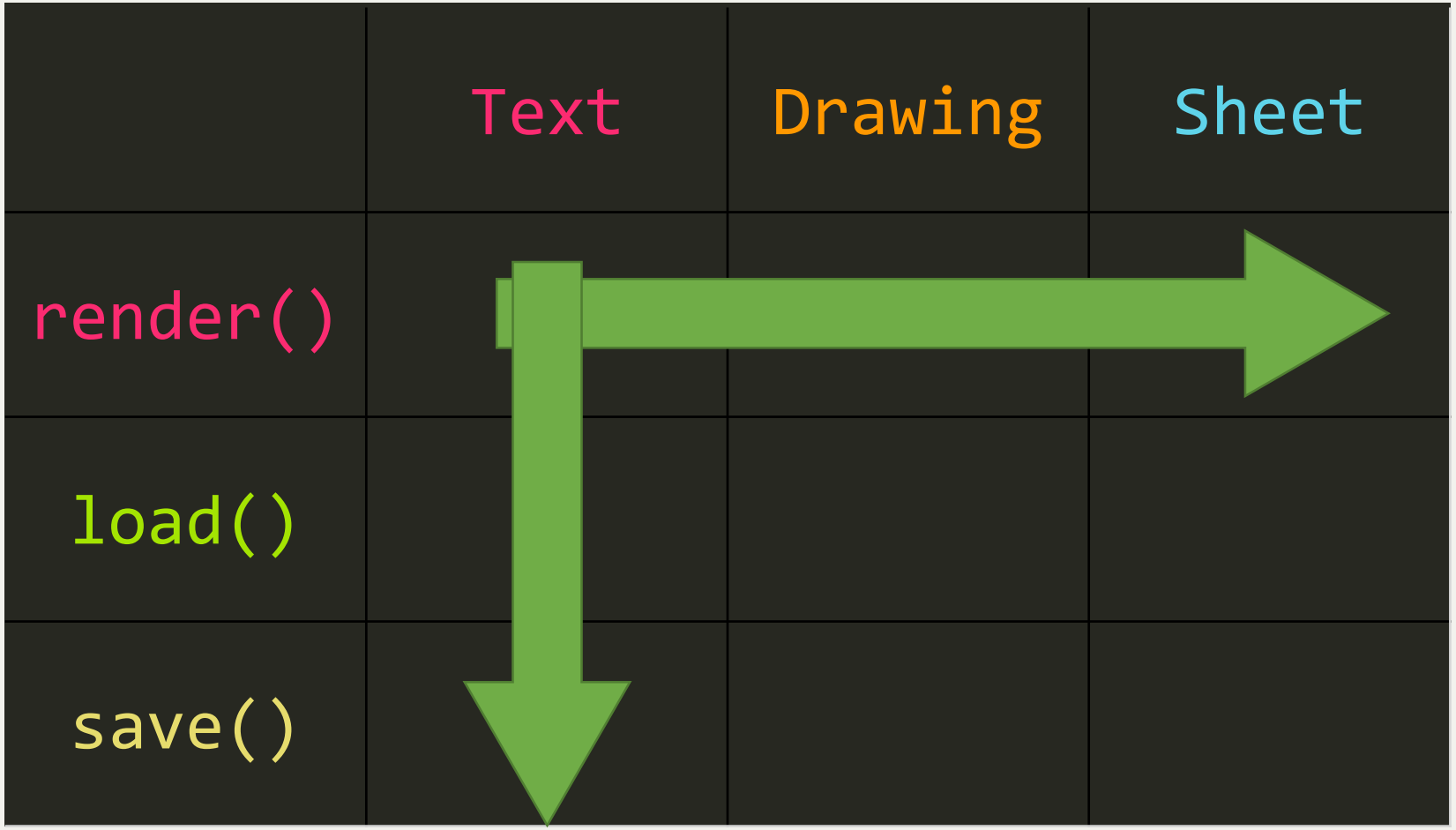
Expression Problem



Expression Problem



Expression Problem



Bob Nystrom, Solving *the Expression Problem*, 2010

Expression Problem

	Num	Add	Neg
show	EP1		EP3
eval	EP2		EP4

Scala

```
trait EP1
{ trait Exp { def show: String }
  class Num(value: Int) extends Exp
      { def show = value.toString() }
  class Add(left: Exp, right: Exp) extends Exp
      { def show = left.show + "+" + right.show } }
```

```
trait EP2 extends EP1
{ class Neg(term: Exp) extends Exp
      { def show = "-(" + term.show + ")" } }
```

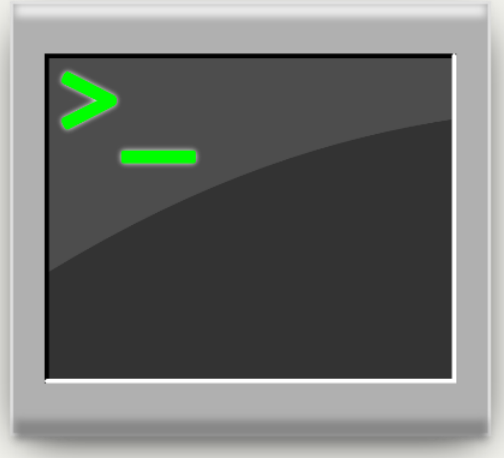
```
trait EP3 extends EP1
{ trait Exp extends super.Exp { def eval: Int }
  class Num(value: Int) extends super.Num(value) with Exp
      { def eval = value }
  class Add(left: Exp, right: Exp) extends
      super.Add(left, right) with Exp
      { def eval = left.eval + right.eval } }
```

```
trait EP4 extends EP2 with EP3
{ class Neg(term: Exp) extends super.Neg(term) with Exp
      { def eval = -term.eval } }
```

Görel Hedin, *The Expression Problem*, 2012

Conclusion

- The history of SE is in languages
- DSLs are good:
  - domain-specific notations
  - domain-specific abstractions
  - tool support for analysis, verification, optimisation, simulation, visualisation, animation, …
  - conciseness and self-documentation
  - productivity, maintainability, reliability, portability
- Internal/External/Embedded, Homo/Hetero
- Implementation patterns
- Expression Problem



raincode LABS
compiler experts

WOS - DIVIDE ET IMPERA (all seeing eye) by MalleimMain