`Programming Languages shape Computational Thinking'

Intreerede prof. dr. Eelco Visser

Hoogleraar Programming Languages

17 Januari 2014

This Lecture

```
repeat n times
  variation-of(same-argument)

for(p in important-people)
  thank-you(p)

for(p in audience)
  individual-quiz(p)


def same-argument =
   “Programming Languages shape Computational Thinking”
```

Automatisering beperkt wachttijd rijexamen

Ondernemingsraden in banksector willen automatisering onder controle krijgen

Automatisering landbouw teveel zaak van koplopers

Automatisering mislukt bij helft bedrijven

GROTE BEHOEFTE AAN PERSONEEL AUTOMATISERING

De PC, een algemeen gebruikt, maar toch ingewikkeld gereedschap

Artsen bekijken nut computer

Mensen zijn geen verlengstuk van de computer

Kosten automatisering overheid stijgen snel

Vrouw en computer

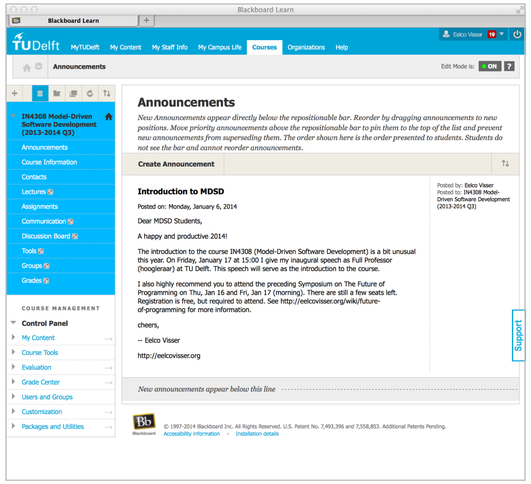
Automatisering politie is verkapte bezuiniging

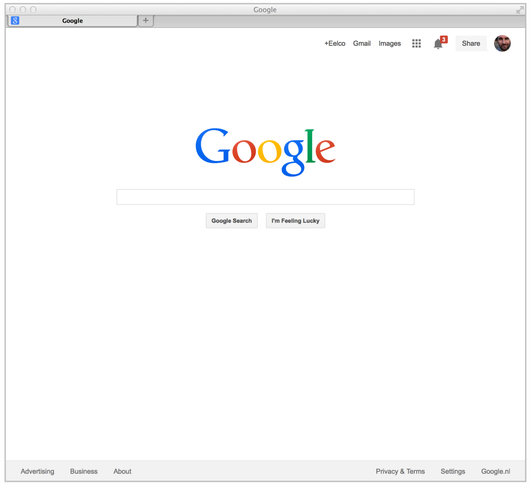
WERKNEMERS ONVOLDOENDE BIJ AUTOMATISERING BETROKKEN

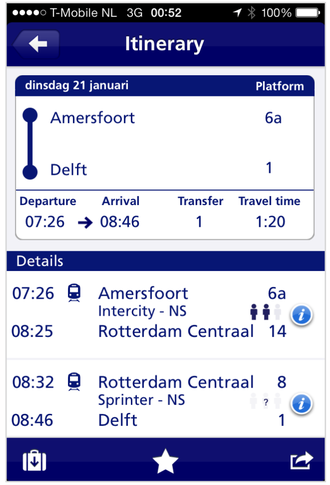
Source: http://kranten.kb.nl

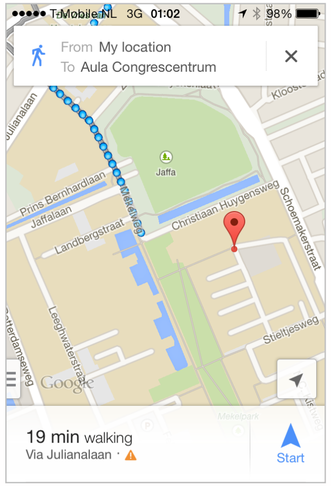
TU Delft

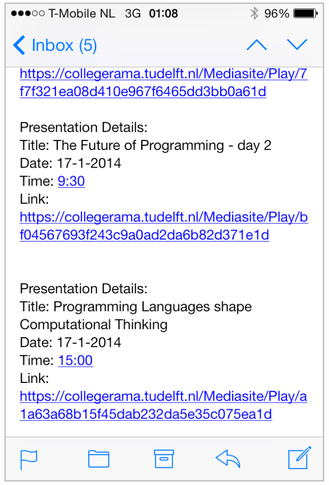
Software systems are the engines of modern information society

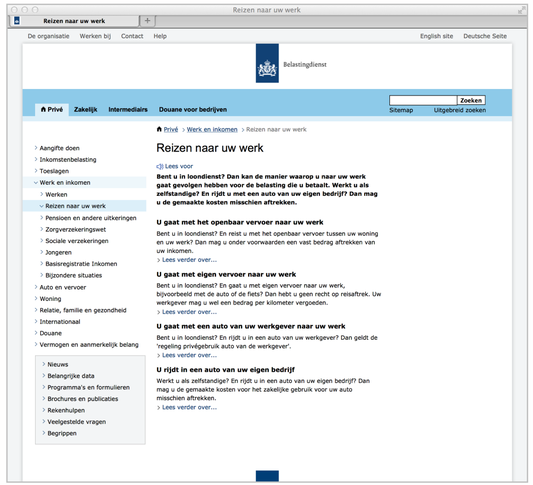






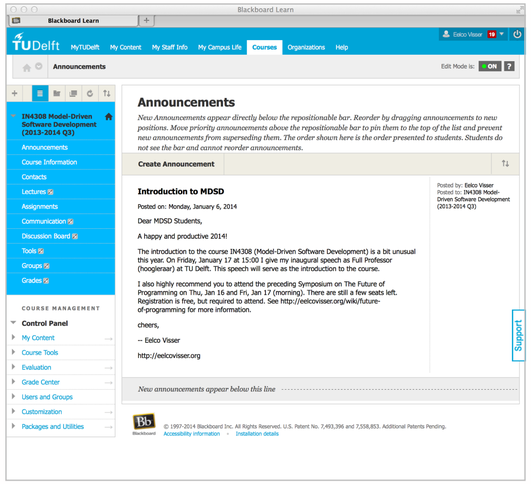


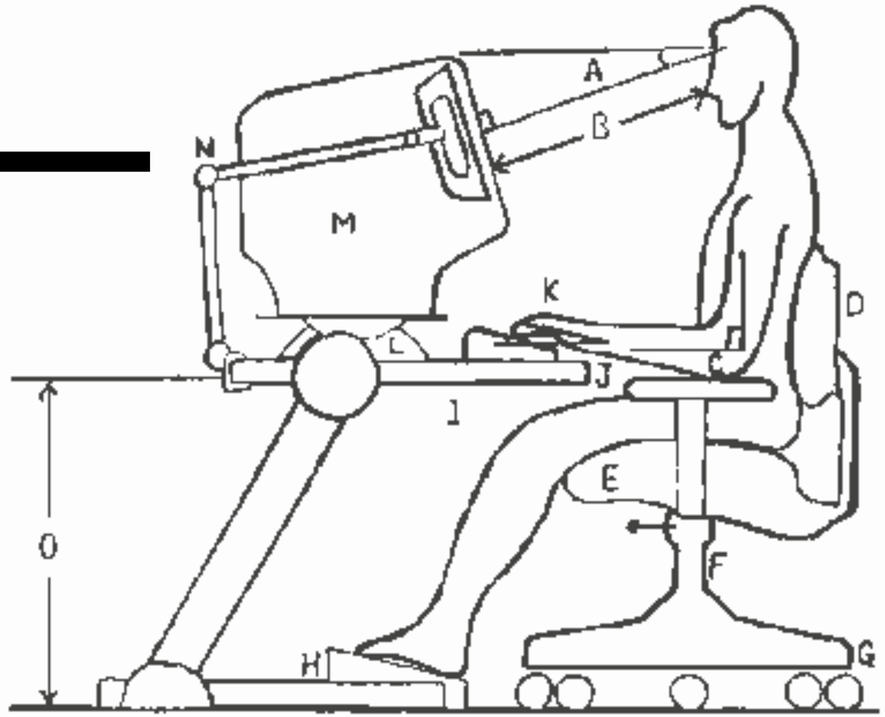




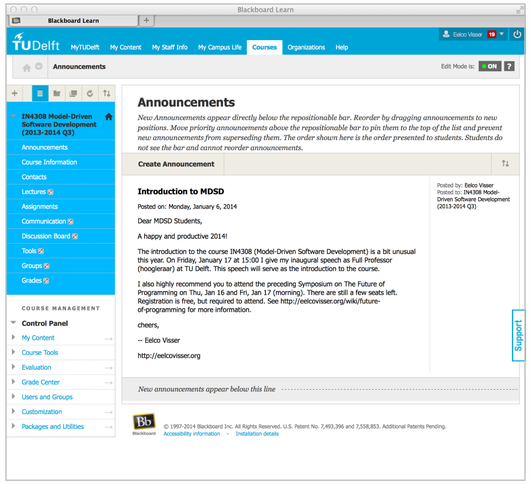
TU Delft

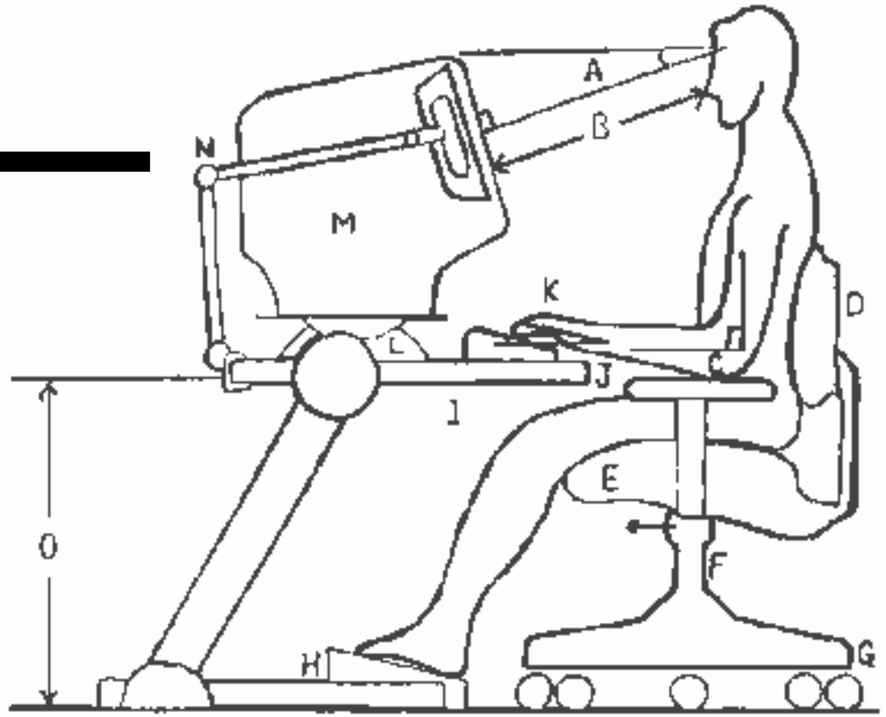
Software systems are for people





Software should ‘just work’

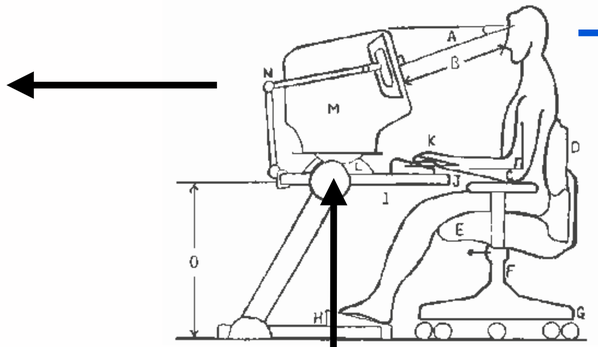




- [x] easy to use
- [x] safe
- [x] cheap (free!)
- [x] responsive
- [x] scale
- [x] reliable
- [x] don’t lose data
- [x] keep data private
- [x] help catch terrorists

TU Delft

Software is encoded computational thinking

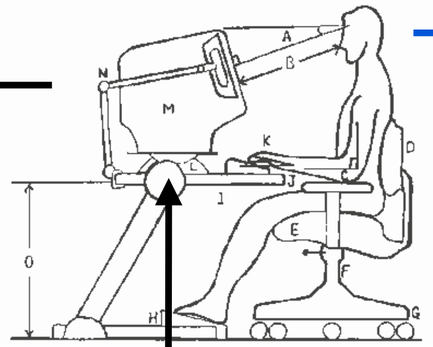


software

```
_fib:
        pushq %rbp
        movq  %rsp, %rbp
        subq  $16, %rsp
        movl  %edi, -4(%rbp)
        movl  -4(%rbp), %eax
        cmpl  $1, %eax
        jg    LBB1_2
        movl  $1, -12(%rbp)
        jmp   LBB1_3
LBB1_2:
        movl  -4(%rbp), %eax
        subl  $2, %eax
        movl  %eax, %edi
        callq _fib
        movl  %eax, %ecx
        movl  -4(%rbp), %edx
        subl  $1, %edx
        movl  %ecx, -16(%rbp)
        movl  %edx, %edi
        callq _fib
        movl  %eax, %ecx
        movl  -16(%rbp), %edx
        addl  %ecx, %edx
        movl  %edx, -12(%rbp)
LBB1_3:
        movl  -12(%rbp), %eax
        movl  %eax, -8(%rbp)
        movl  -8(%rbp), %eax
        addq  $16, %rsp
        popq  %rbp
        ret
```

TU Delft

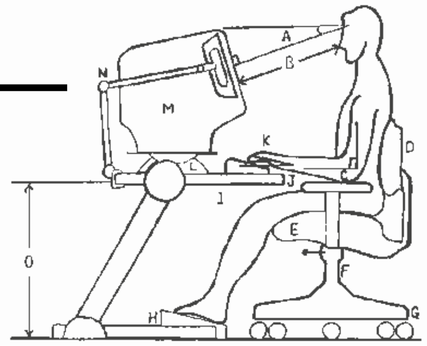
Software engineers are the architects and mechanics of modern information society



computer user

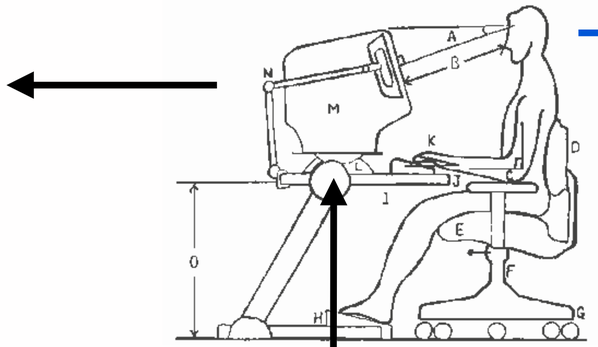
```
_fib:
        pushq %rbp
        movq  %rsp, %rbp
        subq  $16, %rsp
        movl  %edi, -4(%rbp)
        movl  -4(%rbp), %eax
        cmpl  $1, %eax
        jg    LBB1_2
        movl  $1, -12(%rbp)
        jmp   LBB1_3
LBB1_2:
        movl  -4(%rbp), %eax
        subl  $2, %eax
        movl  %eax, %edi
        callq _fib
        movl  %eax, %ecx
        movl  -4(%rbp), %edx
        subl  $1, %edx
        movl  %ecx, -16(%rbp)
        movl  %edx, %edi
        callq _fib
        movl  %eax, %ecx
        movl  -16(%rbp), %edx
        addl  %ecx, %edx
        movl  %edx, -12(%rbp)
LBB1_3:
        movl  -12(%rbp), %eax
        movl  %eax, -8(%rbp)
        movl  -8(%rbp), %eax
        addq  $16, %rsp
        popq  %rbp
        ret
```

software



software engineer

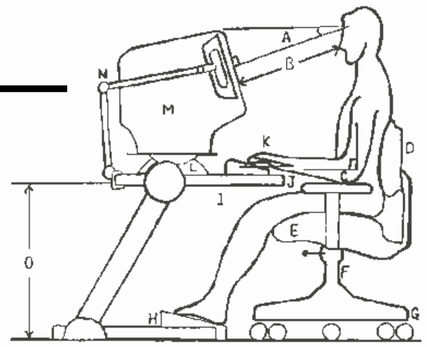
Software engineering provides the tools that let software engineers do a good job



computer user

```
_fib:
        pushq %rbp
        movq  %rsp, %rbp
        subq  $16, %rsp
        movl  %edi, -4(%rbp)
        movl  -4(%rbp), %eax
        cmpl  $1, %eax
        jg    LBB1_2
        movl  $1, -12(%rbp)
        jmp   LBB1_3
LBB1_2:
        movl  -4(%rbp), %eax
        subl  $2, %eax
        movl  %eax, %edi
        callq _fib
        movl  %eax, %ecx
        movl  -4(%rbp), %edx
        subl  $1, %edx
        movl  %ecx, -16(%rbp)
        movl  %edx, %edi
        callq _fib
        movl  %eax, %ecx
        movl  -16(%rbp), %edx
        addl  %ecx, %edx
        movl  %edx, -12(%rbp)
LBB1_3:
        movl  -12(%rbp), %eax
        movl  %eax, -8(%rbp)
        movl  -8(%rbp), %eax
        addq  $16, %rsp
        popq  %rbp
        ret
```

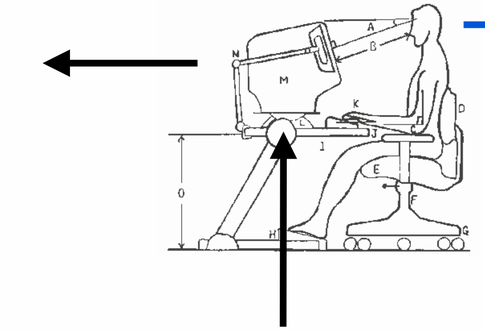
software



software engineer

design patterns
best practices
coding standards
code reviews
pair programming
(unit) testing
debugging

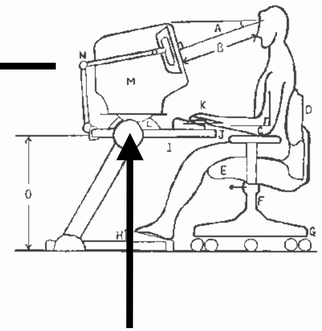
Programming languages are the key tools in the encoding process



computer user

software

```
_fib:
        pushq %rbp
        movq  %rsp, %rbp
        subq  $16, %rsp
        movl  %edi, -4(%rbp)
        movl  -4(%rbp), %eax
        cmpl  $1, %eax
        jg    LBB1_2
        movl  $1, -12(%rbp)
        jmp   LBB1_3
LBB1_2:
        movl  -4(%rbp), %eax
        subl  $2, %eax
        movl  %eax, %edi
        callq _fib
        movl  %eax, %ecx
        movl  -4(%rbp), %edx
        subl  $1, %edx
        movl  %ecx, -16(%rbp)
        movl  %edx, %edi
        callq _fib
        movl  %eax, %ecx
        movl  -16(%rbp), %edx
        addl  %ecx, %edx
        movl  %edx, -12(%rbp)
LBB1_3:
        movl  -12(%rbp), %eax
        movl  %eax, -8(%rbp)
        movl  -8(%rbp), %eax
        addq  $16, %rsp
        popq  %rbp
        ret
```



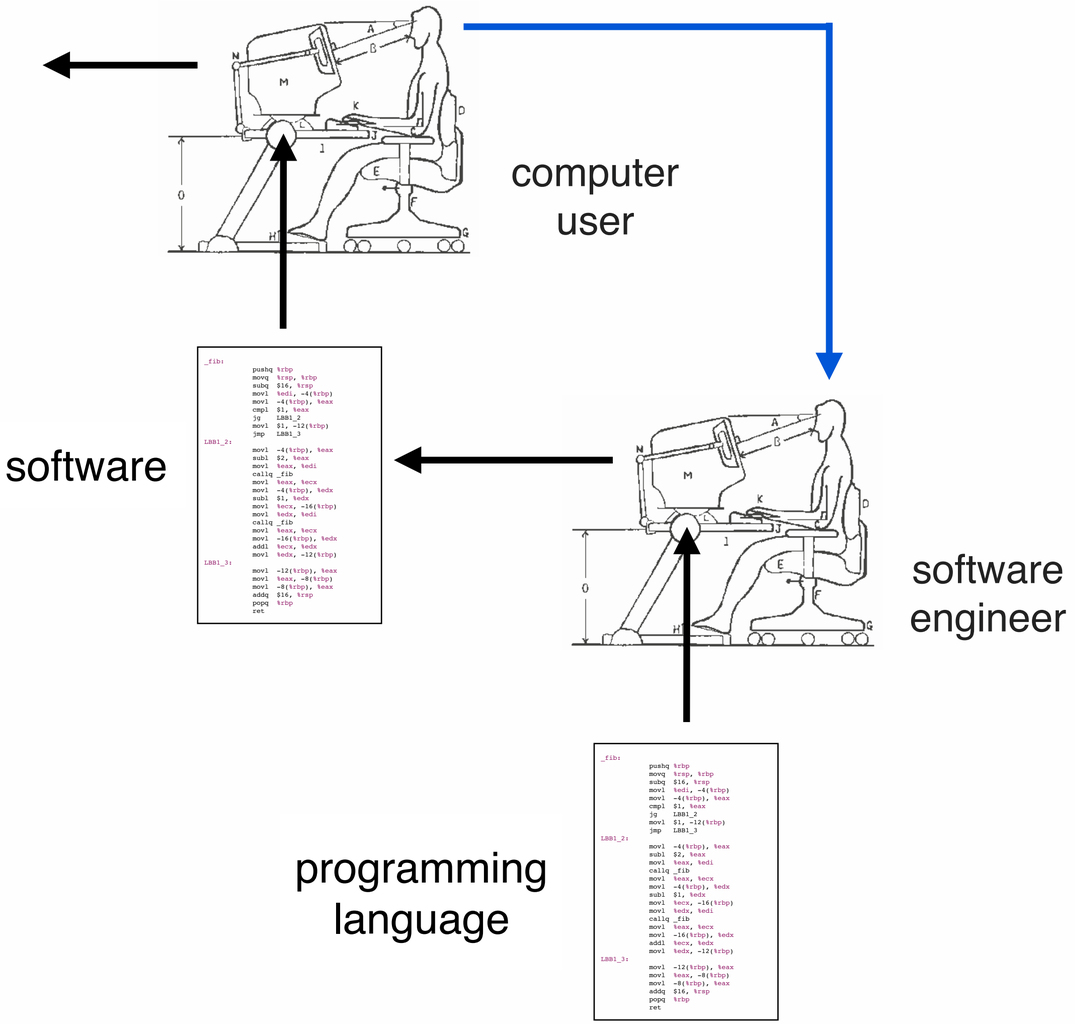
software engineer

programming language

```
_fib:
        pushq %rbp
        movq  %rsp, %rbp
        subq  $16, %rsp
        movl  %edi, -4(%rbp)
        movl  -4(%rbp), %eax
        cmpl  $1, %eax
        jg    LBB1_2
        movl  $1, -12(%rbp)
        jmp   LBB1_3
LBB1_2:
        movl  -4(%rbp), %eax
        subl  $2, %eax
        movl  %eax, %edi
        callq _fib
        movl  %eax, %ecx
        movl  -4(%rbp), %edx
        subl  $1, %edx
        movl  %ecx, -16(%rbp)
        movl  %edx, %edi
        callq _fib
        movl  %eax, %ecx
        movl  -16(%rbp), %edx
        addl  %ecx, %edx
        movl  %edx, -12(%rbp)
LBB1_3:
        movl  -12(%rbp), %eax
        movl  %eax, -8(%rbp)
        movl  -8(%rbp), %eax
        addq  $16, %rsp
        popq  %rbp
        ret
```

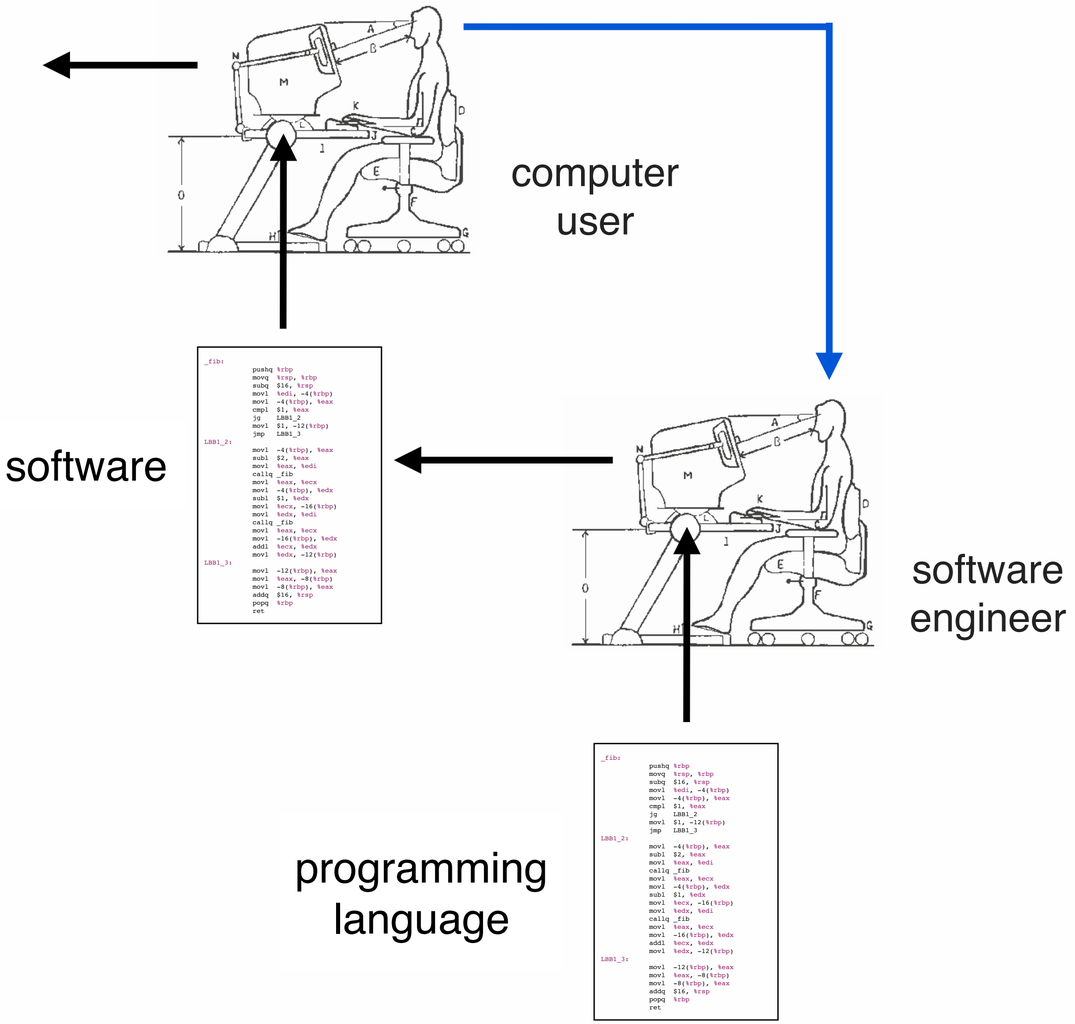
TU Delft

Programming languages are software systems too



compiler
interpreter
type checker
editor (IDE)
refactorings
bug finders

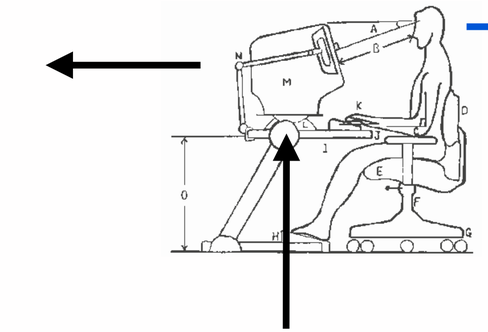
Programming languages should ‘just work’



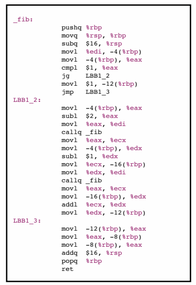
- ☑ easy to use
- ☑ safe
- ☑ cheap (free!)
- ☑ responsive
- ☑ scale
- ☑ reliable

TU Delft

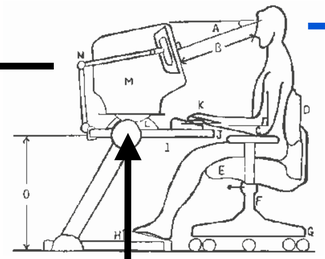
Language designers create programming languages



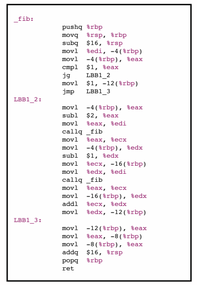
computer user



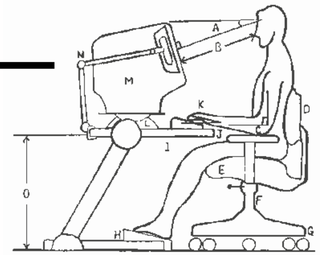
software



software engineer



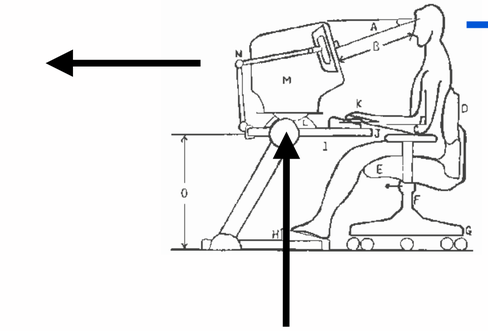
programming language



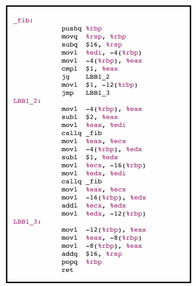
language designer

TU Delft

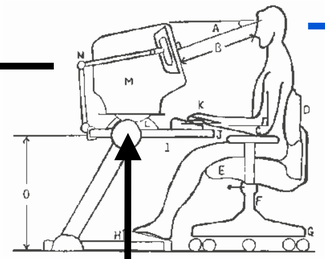
Language engineering provides the tools that let language designers do a good job



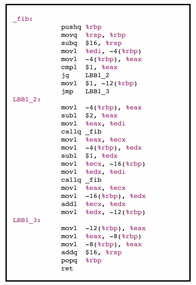
computer user



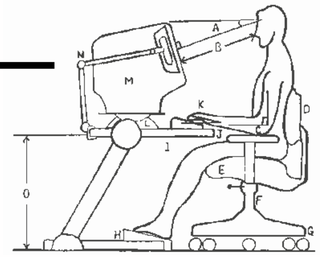
software



software engineer



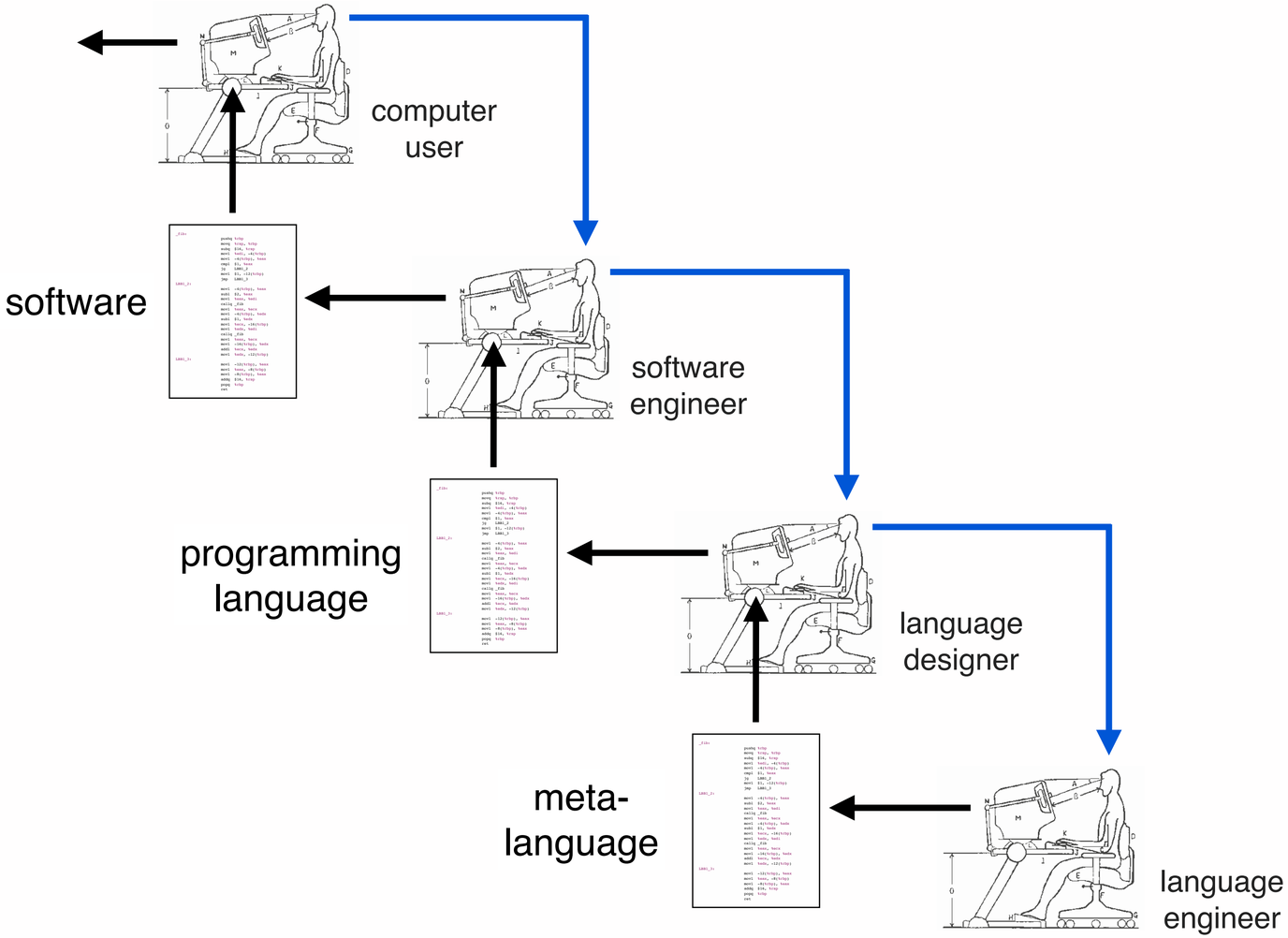
programming language



language designer

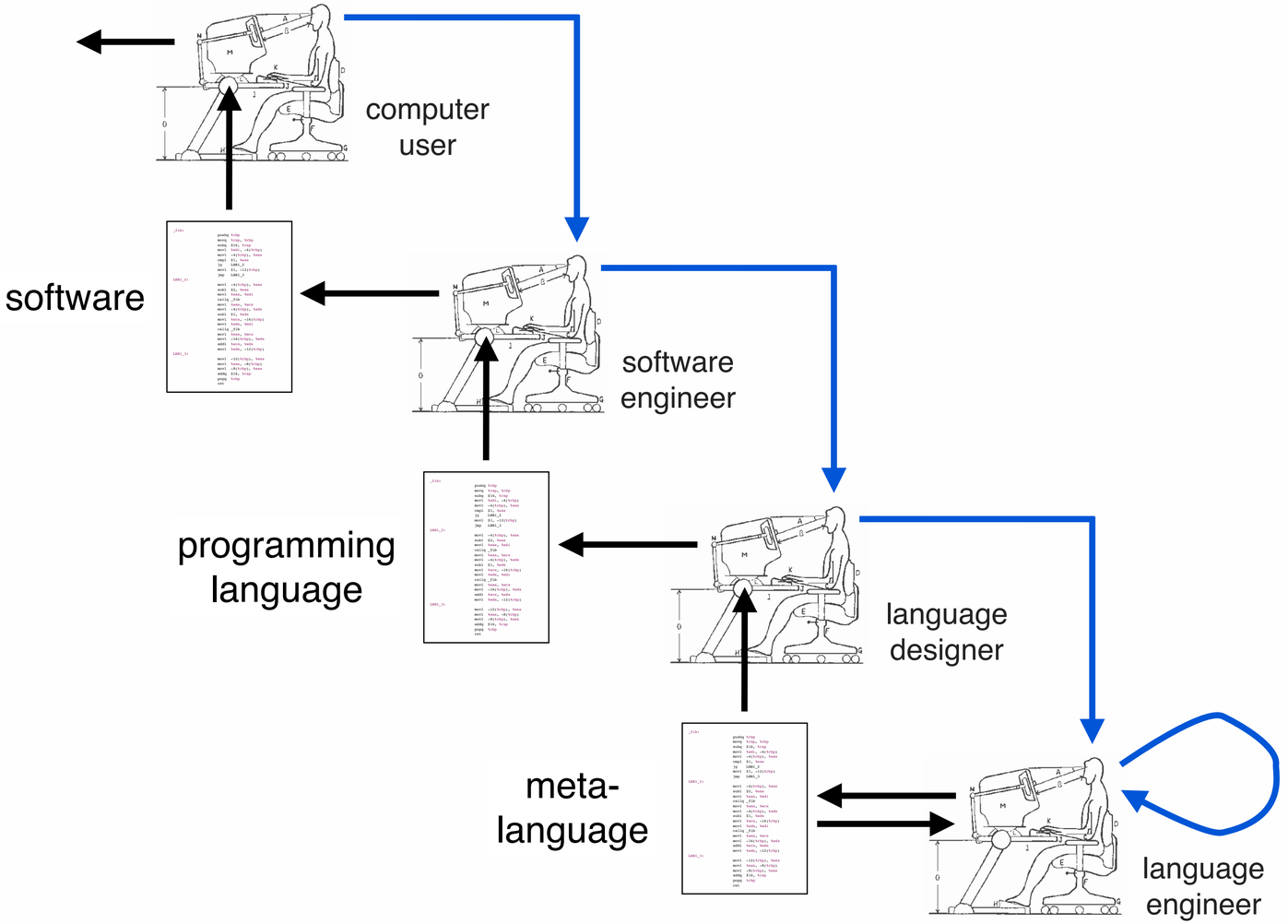
TU Delft

Meta-languages support the encoding of language designs

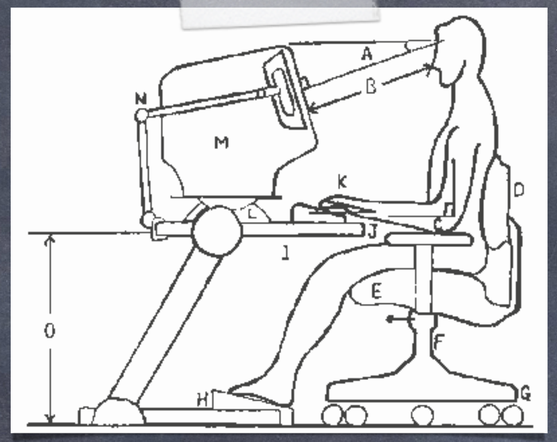


TU Delft

Language engineers eat their own dogfood



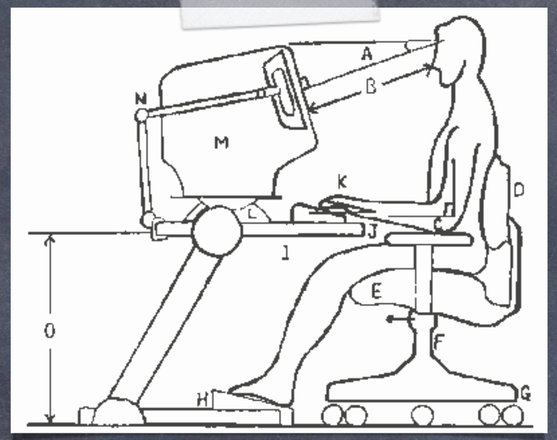
Software is encoded computational thinking



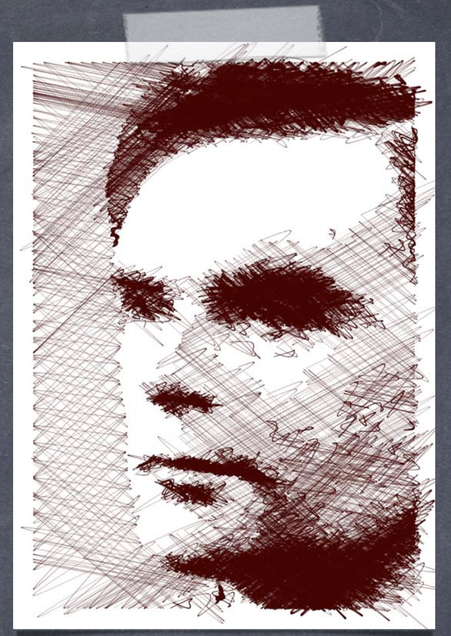

```
_fib:
        pushq %rbp
        movq  %rsp, %rbp
        subq  $16, %rsp
        movl  %edi, -4(%rbp)
        movl  -4(%rbp), %eax
        cmpl  $1, %eax
        jg    LBB1_2
        movl  $1, -12(%rbp)
        jmp   LBB1_3
LBB1_2:
        movl  -4(%rbp), %eax
        subl  $2, %eax
        movl  %eax, %edi
        callq _fib
        movl  %eax, %ecx
        movl  -4(%rbp), %edx
        subl  $1, %edx

        movl  %ecx, -16(%rbp)
        movl  %edx, %edi
        callq _fib
        movl  %eax, %ecx
        movl  -16(%rbp), %edx
        addl  %ecx, %edx
        movl  %edx, -12(%rbp)
LBB1_3:
        movl  -12(%rbp), %eax
        movl  %eax, -8(%rbp)
        movl  -8(%rbp), %eax
        addq  $16, %rsp
        popq  %rbp
        ret
```

TU Delft

Software is encoded computational thinking




```
_fib:
        pushq %rbp
        movq  %rsp, %rbp
        subq  $16, %rsp
        movl  %edi, -4(%rbp)
        movl  -4(%rbp), %eax
        cmpl  $1, %eax
        jg    LBB1_2
        movl  $1, -12(%rbp)
        jmp   LBB1_3
LBB1_2:
        movl  -4(%rbp), %eax
        subl  $2, %eax
        movl  %eax, %edi
        callq _fib
        movl  %eax, %ecx
        movl  -4(%rbp), %edx
        subl  $1, %edx

        movl  %ecx, -16(%rbp)
        movl  %edx, %edi
        callq _fib
        movl  %eax, %ecx
        movl  -16(%rbp), %edx
        addl  %ecx, %edx
        movl  %edx, -12(%rbp)
LBB1_3:
        movl  -12(%rbp), %eax
        movl  %eax, -8(%rbp)
        movl  -8(%rbp), %eax
        addq  $16, %rsp
        popq  %rbp
        ret
```

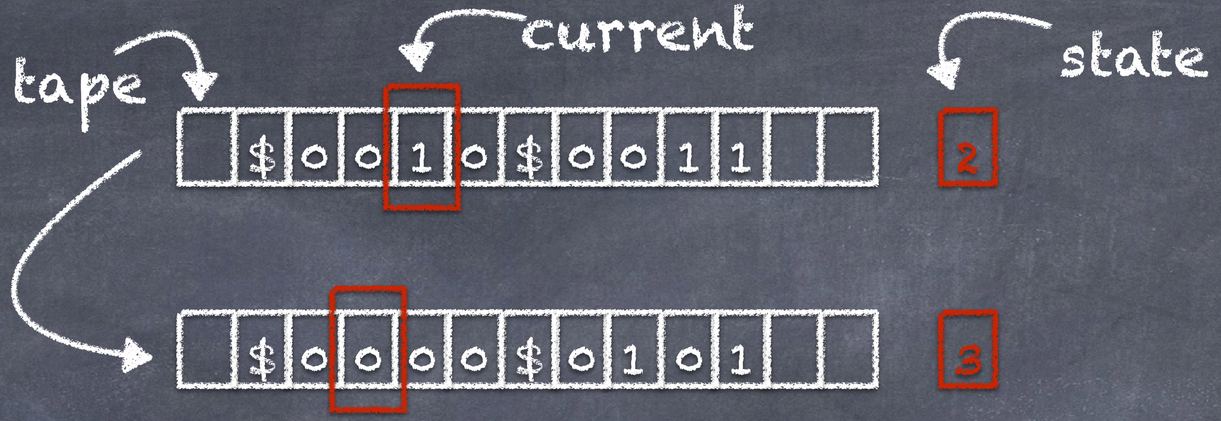


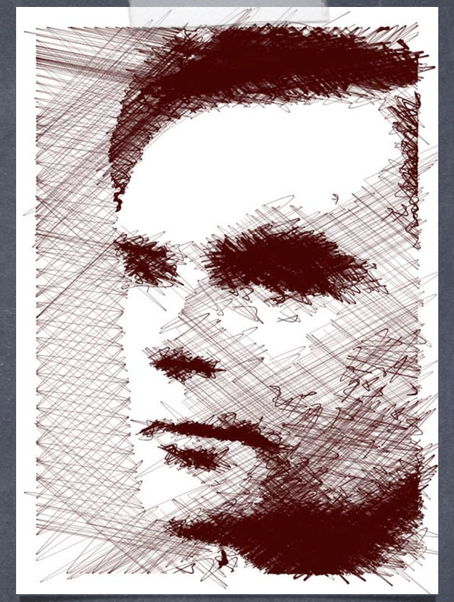
Alan Turing
1912 - 1954

modeling mechanical computation

TU Delft

Turing Machine





Alan Turing
1936

1	$	$	R	2
2	o	o	R	2
2	1	1	L	3
2	$	$	R	H
3	o	o	L	3
3	1	1	L	3
3	$	$	R	4

current state

next state

current symbol

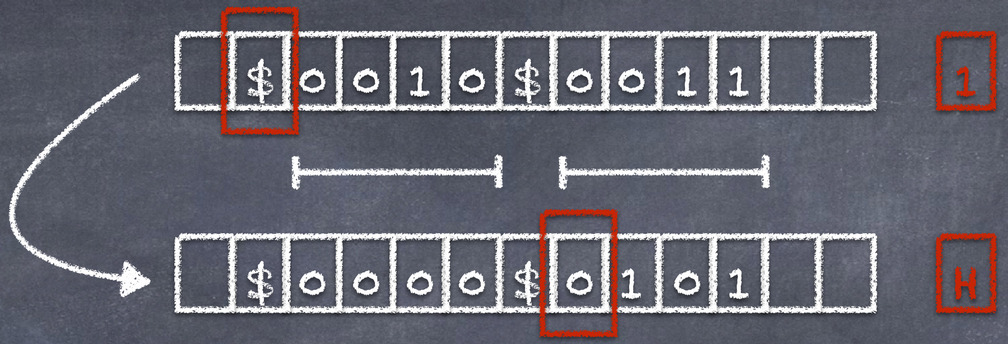
new symbol

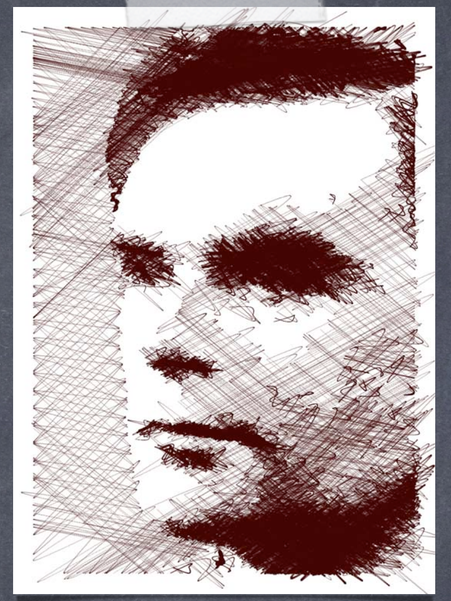
move tape

Church-Turing Thesis: Any effective computation can be expressed with a Turing Machine

TU Delft

Turing Machine





Alan Turing
1936

Add two binary numbers

1	$	$	R	2
2	0	0	R	2
2	1	1	L	3
2	$	$	R	H
3	0	0	L	3
3	1	1	L	3
3	$	$	R	4

is zero?

4	0	1	R	4
4	1	0	R	4
4	$	$	L	5
5	1	0	L	5
5	0	1	L	6
6	0	0	L	6
6	1	1	L	6
6	$	$	R	7
7	0	1	R	7
7	1	0	R	7
7	$	$	R	8

subtract one

8	0	0	R	8
8	1	1	R	8
8	$	$	L	9
9	1	0	L	9
9	0	1	L	10
10	0	0	L	10
10	1	1	L	10
10	$	$	L	11
11	0	0	L	11
11	1	1	L	11
12	$	$	R	2

add one

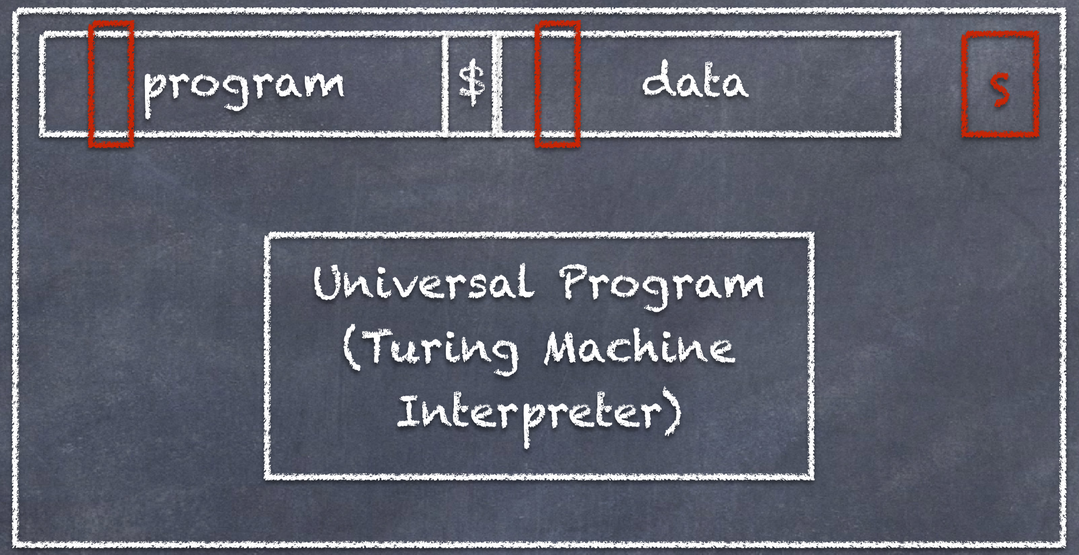
back

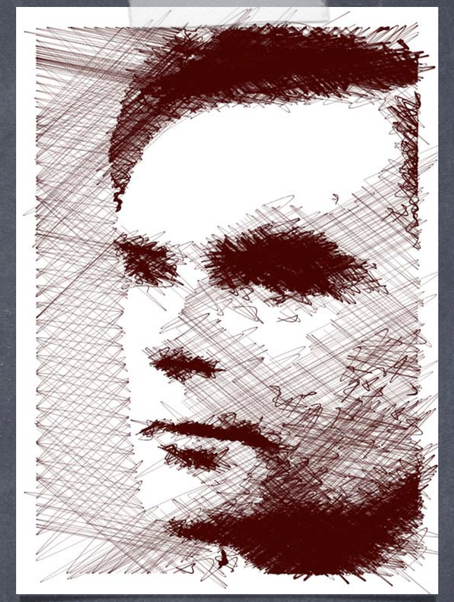
repeat

Church-Turing Thesis: Any effective computation can be expressed with a Turing Machine

TU Delft

Universal Turing Machine





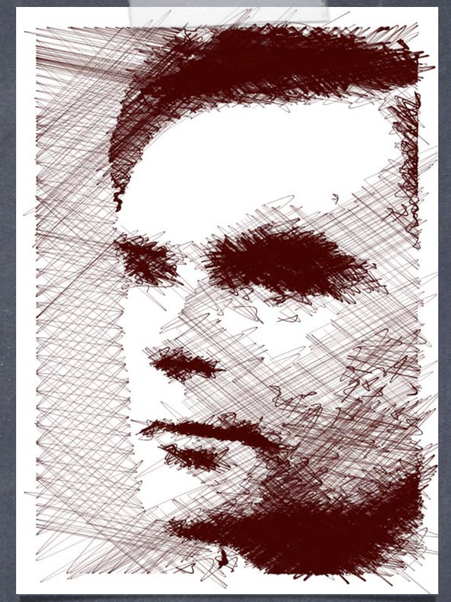
Alan Turing
1936

Halting problem: given a Turing Machine program P, does P halt for any input data?

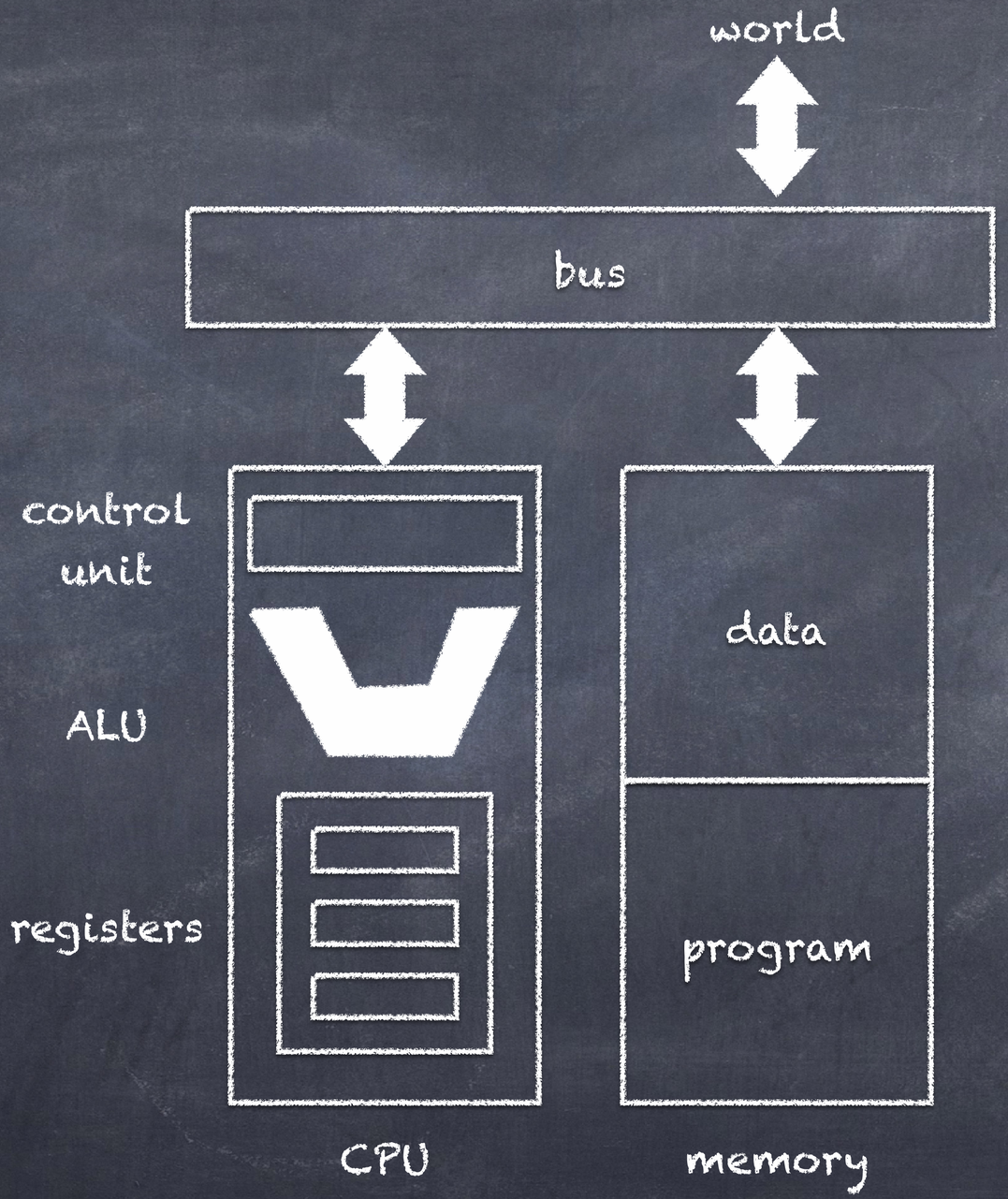
Undecidable!

TU Delft

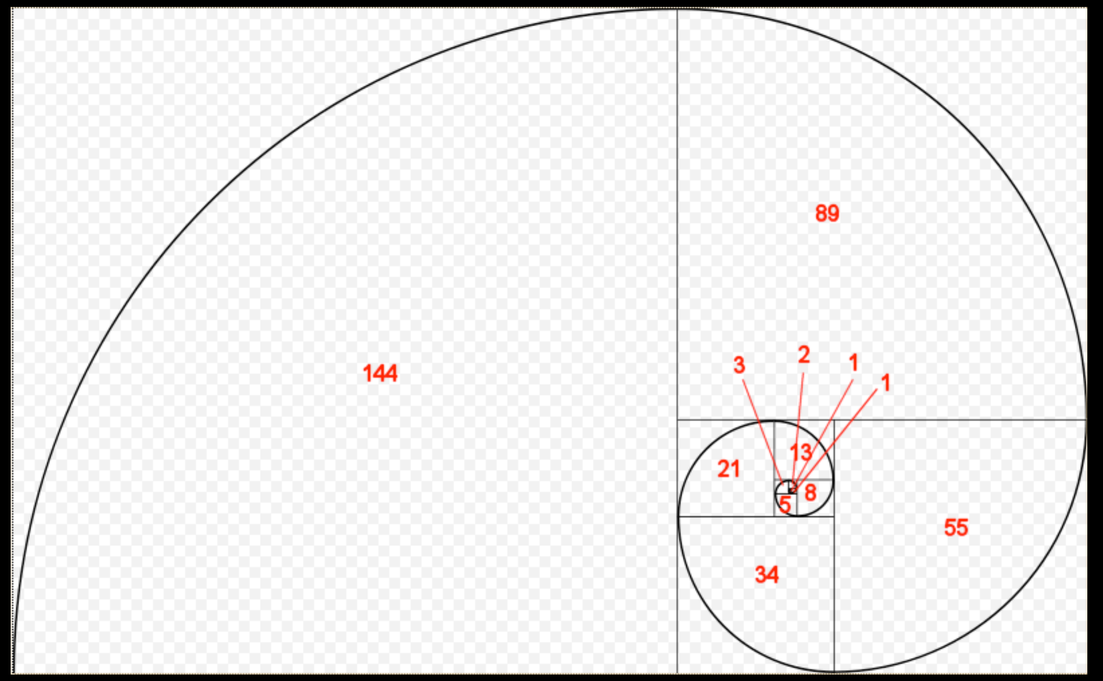
Von Neumann



Alan Turing

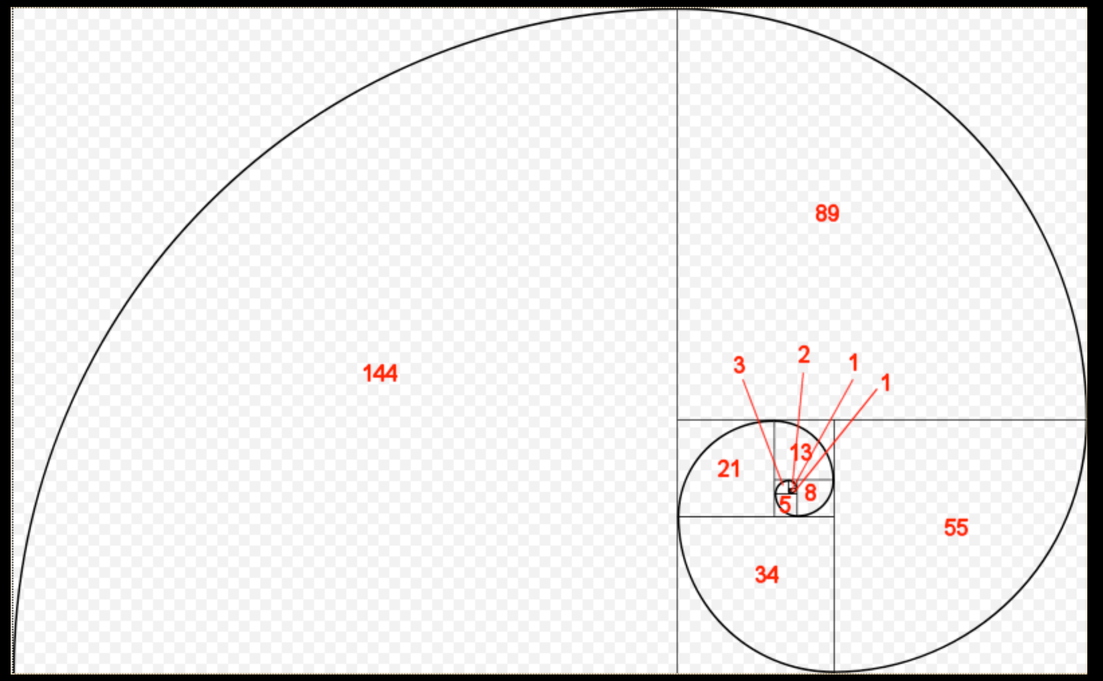


Simple Instructions
add, subtract
fetch, store
...



n	n-th fibonacci
0	1
1	1
2	2 = 1 + 1
3	3 = 1 + 2
4	5 = 2 + 3
5	8 = 3 + 5
6	13 = 5 + 8
7	21 = 8 + 13
8	34 = 13 + 21

Example: a procedure for computing Fibonacci numbers



n	n-th fibonacci
0	1
1	1
2	2 = 1 + 1
3	3 = 1 + 2
4	5 = 2 + 3
5	8 = 3 + 5
6	13 = 5 + 8
7	21 = 8 + 13
8	34 = 13 + 21

Example: a procedure for computing Fibonacci numbers

compute n-th fibonacci number

if n less or equal to 1 result is 1

compute (n-2)-nd fibonacci number

compute (n-1)-th fibonacci number

add the results

return to caller

```
_fib:
    pushq   %rbp
    movq    %rsp, %rbp
    subq    $16, %rsp
    movl    %edi, -4(%rbp)
    movl    -4(%rbp), %eax
    cmpl    $1, %eax
    jg      LBB1_2
    movl    $1, -12(%rbp)
    jmp     LBB1_3
LBB1_2:
    movl    -4(%rbp), %eax
    subl    $2, %eax
    movl    %eax, %edi
    callq   _fib
    movl    %eax, %ecx
    movl    -4(%rbp), %edx
    subl    $1, %edx
    movl    %ecx, -16(%rbp)
    movl    %edx, %edi
    callq   _fib
    movl    %eax, %ecx
    movl    -16(%rbp), %edx
    addl    %ecx, %edx
    movl    %edx, -12(%rbp)
LBB1_3:
    movl    -12(%rbp), %eax
    movl    %eax, -8(%rbp)
    movl    -8(%rbp), %eax
    addq    $16, %rsp
    popq    %rbp
    ret
```

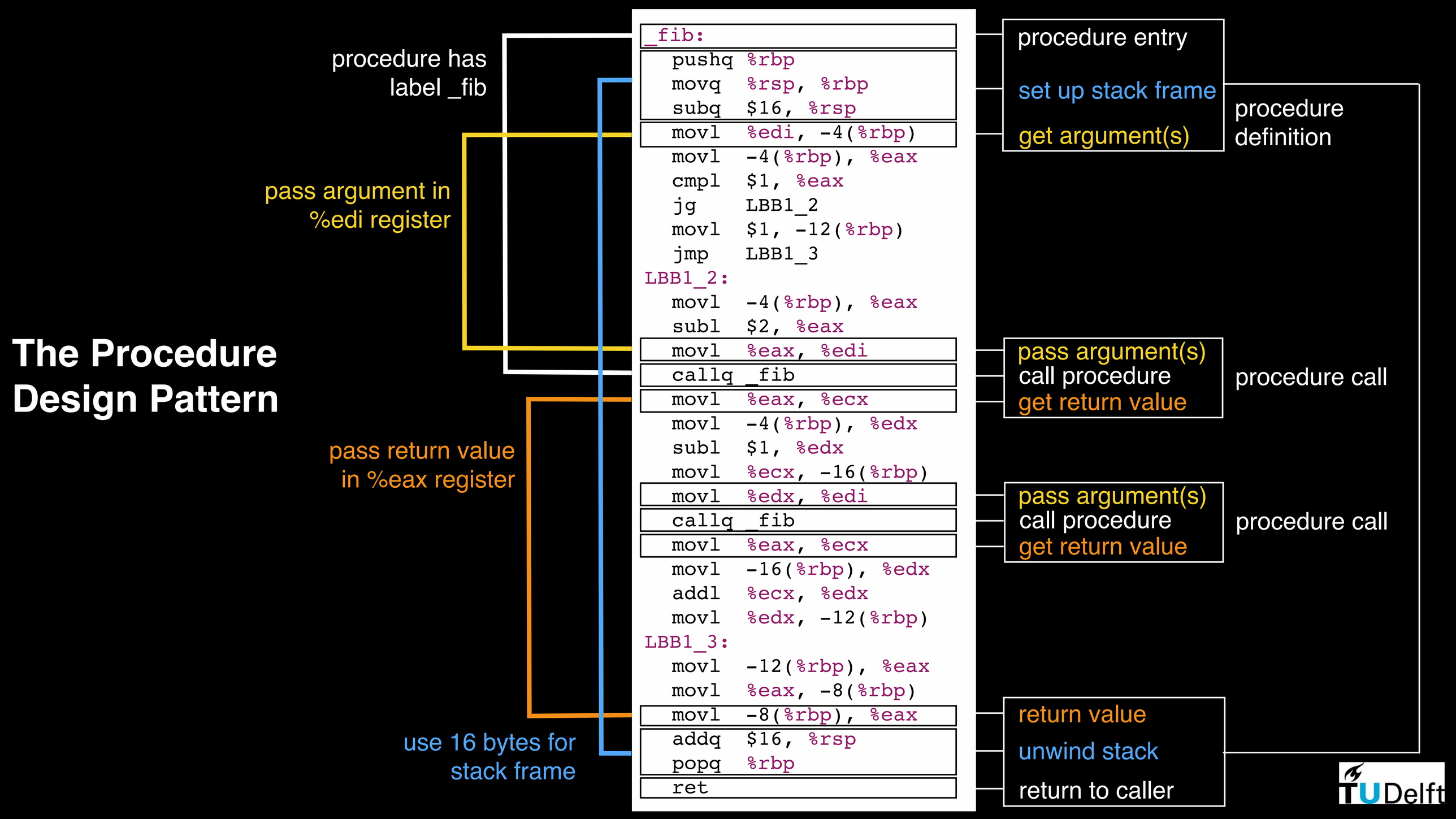
by calling _fib

by calling _fib

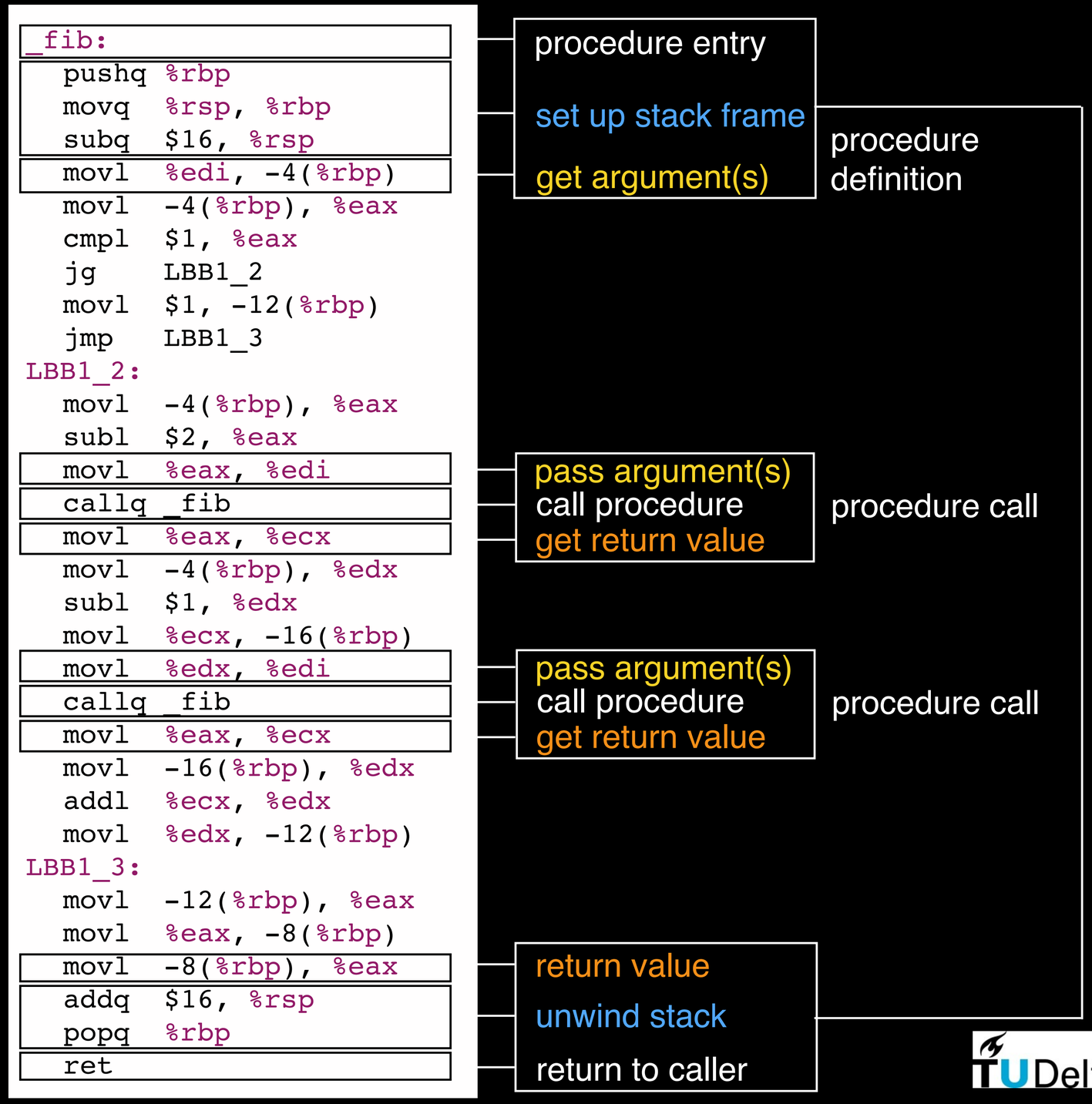
n	n-th fibonacci
0	1
1	1
2	2 = 1 + 1
3	3 = 1 + 2
4	5 = 2 + 3
5	8 = 3 + 5
6	13 = 5 + 8
7	21 = 8 + 13
8	34 = 13 + 21

TU Delft

The Procedure Design Pattern



design patterns
best practices
coding standards
code reviews
pair programming
(unit) testing
bug finders
program annotations
debugging



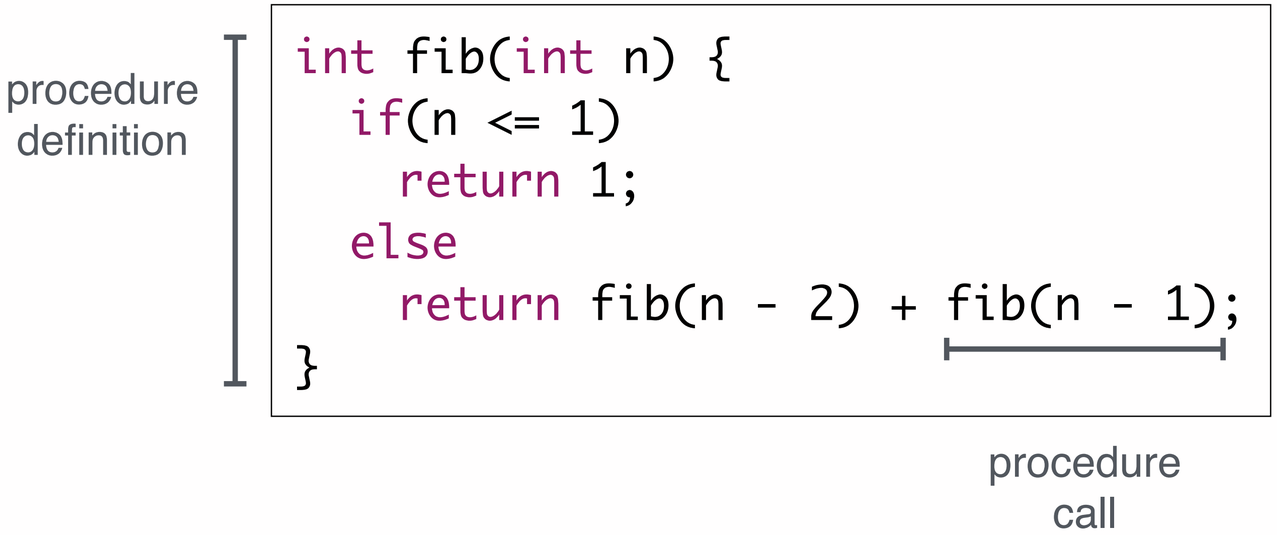
Procedural Abstraction

```
int fib(int n) {
  if(n <= 1)
    return 1;
  else
    return fib(n - 2) + fib(n - 1);
}
```

The C Programming Language - Kernighan & Ritchie (1988)

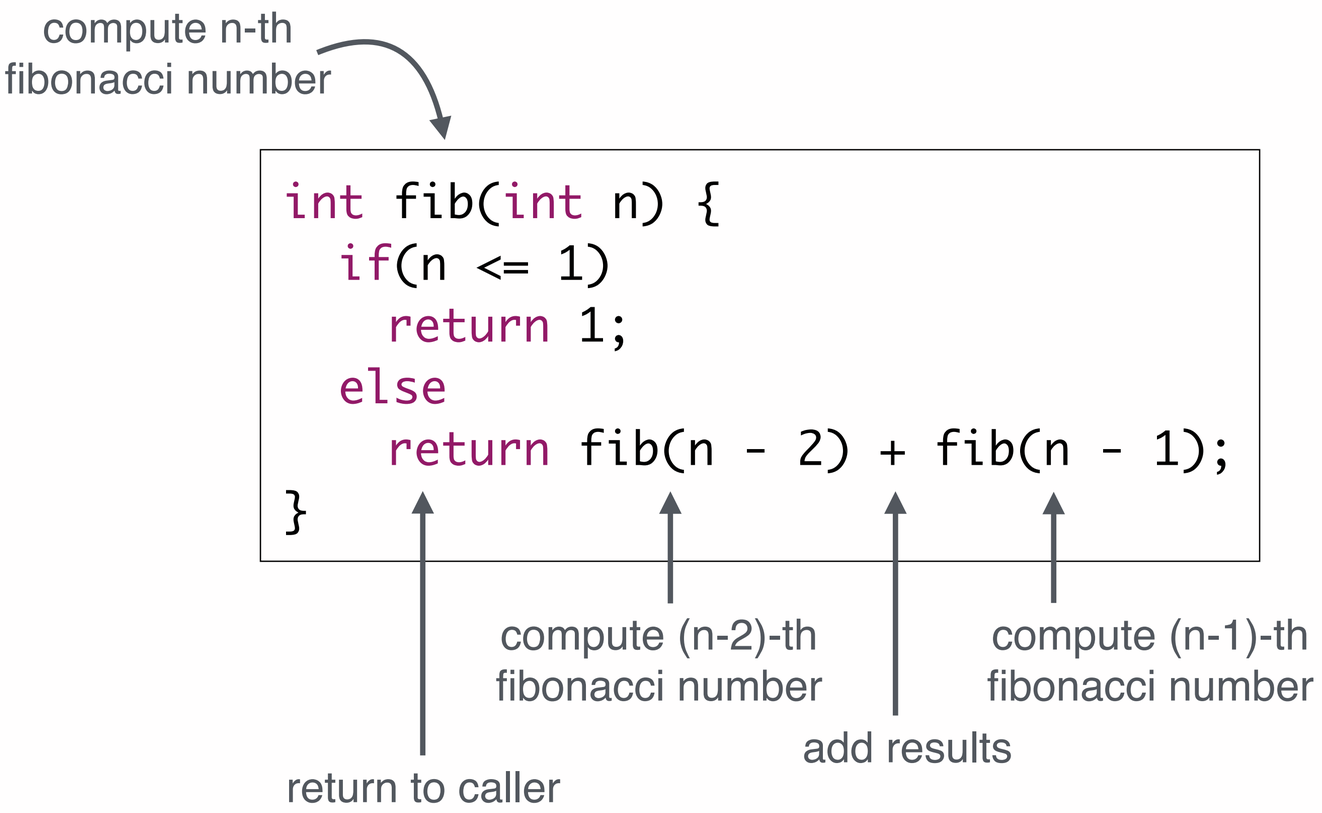
A declarative language for specifying procedures

Procedural Abstraction



Procedure design pattern captured in linguistic abstraction

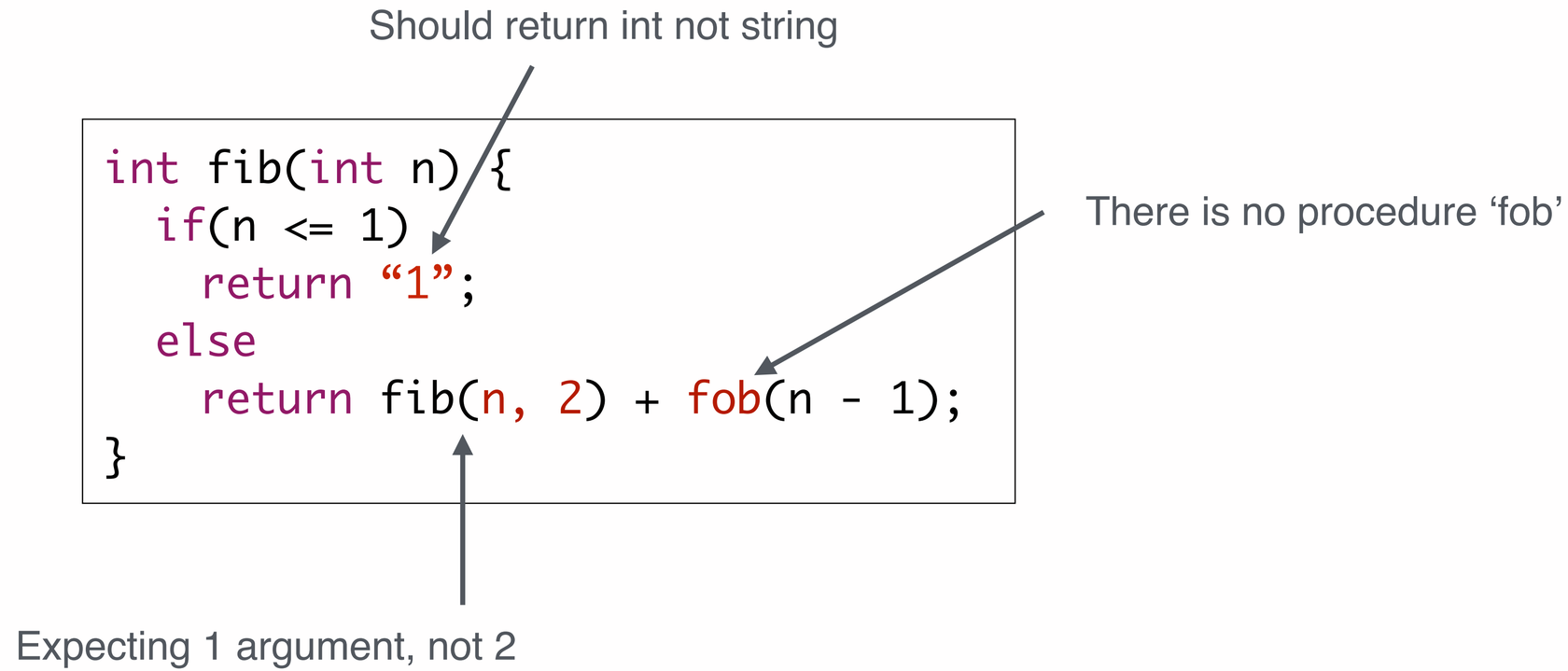
Procedural Abstraction



n	n-th fibonacci
0	1
1	1
2	2 = 1 + 1
3	3 = 1 + 2
4	5 = 2 + 3
5	8 = 3 + 5
6	13 = 5 + 8
7	21 = 8 + 13
8	34 = 13 + 21

Self explanatory code!

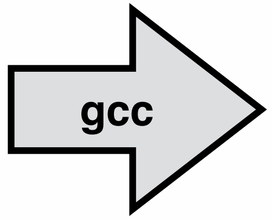
Procedural Abstraction



Automatic consistency checks!

Procedural Abstraction

```
int fib(int n) {
  if(n <= 1)
    return 1;
  else
    return fib(n - 2) + fib(n - 1);
}
```




```
_fib:
    pushq %rbp
    movq  %rsp, %rbp
    subq  $16, %rsp
    movl  %edi, -4(%rbp)
    movl  -4(%rbp), %eax
    cmpl  $1, %eax
    jg    LBB1_2
    movl  $1, -12(%rbp)
    jmp   LBB1_3
LBB1_2:
    movl  -4(%rbp), %eax
    subl  $2, %eax
    movl  %eax, %edi
    callq _fib
    movl  %eax, %ecx
    movl  -4(%rbp), %edx
    subl  $1, %edx
    movl  %ecx, -16(%rbp)
    movl  %edx, %edi
    callq _fib
    movl  %eax, %ecx
    movl  -16(%rbp), %edx
    addl  %ecx, %edx
    movl  %edx, -12(%rbp)
LBB1_3:
    movl  -12(%rbp), %eax
    movl  %eax, -8(%rbp)
    movl  -8(%rbp), %eax
    addq  $16, %rsp
    popq  %rbp
    ret
```

Automatic generation of correct code!

Procedural Abstraction

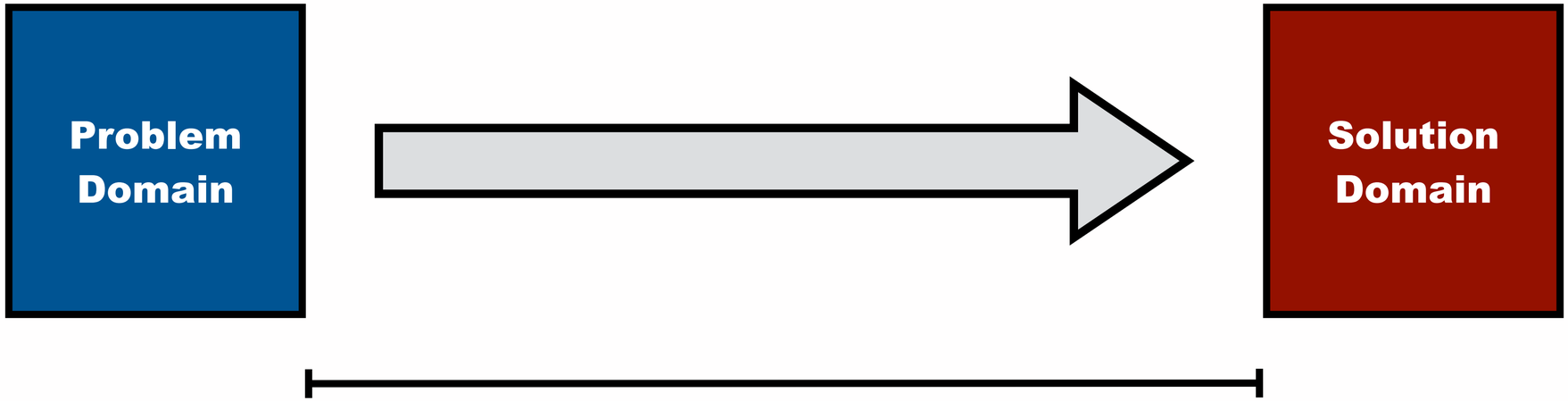
Is this a correct implementation of Fibonacci?

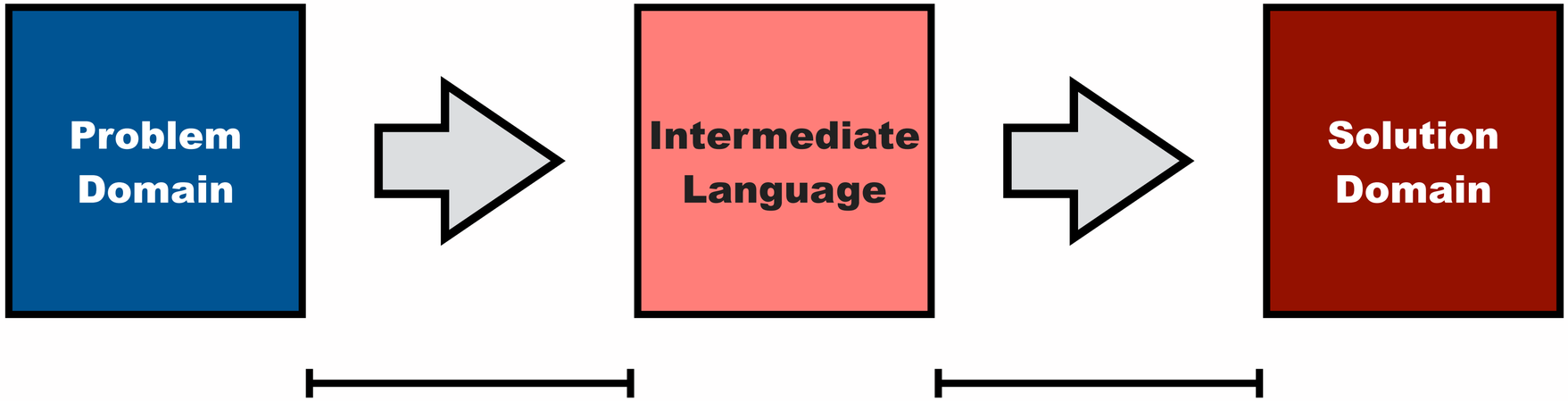
design patterns
best practices
coding standards
code reviews
pair programming
(unit) testing
bug finders
program annotations
debugging

```
int fib(int n) {
  if(n <= 1)
    return 1;
  else
    return fib(n - 2) + fib(n - 1);
}
```

Yes, but it is not a very *responsive* implementation of Fibonacci; its complexity is O(2^n)

Reasoning about behaviour!


Problem
Domain
Solution
Domain



linguistic abstraction | liNG'gwistik ab'strakSHən |

noun

1. a programming language construct that captures a programming design pattern
   - *the linguistic abstraction saved a lot of programming effort*
   - *he introduced a linguistic abstraction for page navigation in web programming*
2. the process of introducing linguistic abstractions
   - *linguistic abstraction for name binding removed the algorithmic encoding of name resolution*

Procedural abstraction

Structured control-flow

Automatic memory management

Data abstraction

Modules

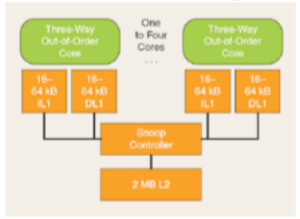
and many more

Church-Turing Thesis++: Any effective computation can be expressed with a *C program*

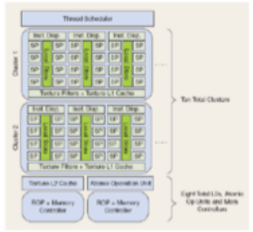
“A programming language is low level when its programs require attention to the irrelevant”. -- Alan Perlis, 1982

And There Will Be Variety ...

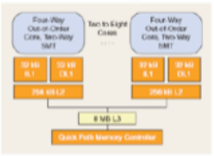
Arm Cortex A9



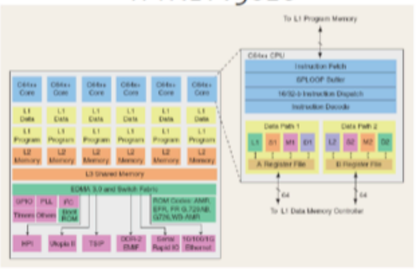
Nvidia G200



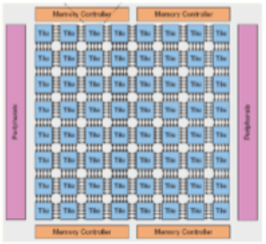
Core i7



TI TNETV3020



Tilera Tile64



Source: IEEE SP Magazine, Vol. 26, November 2009

Optimization for Register Locality and ILP

```
// straightforward code
for(i = 0; i < N; i += 1)
  for(j = 0; j < N; j += 1)
    for(k = 0; k < N; k += 1)
      c[i][j] += a[i][k]*b[k][j];
```

Removes aliasing

Enables register allocation and instruction scheduling

Compiler typically does not do:

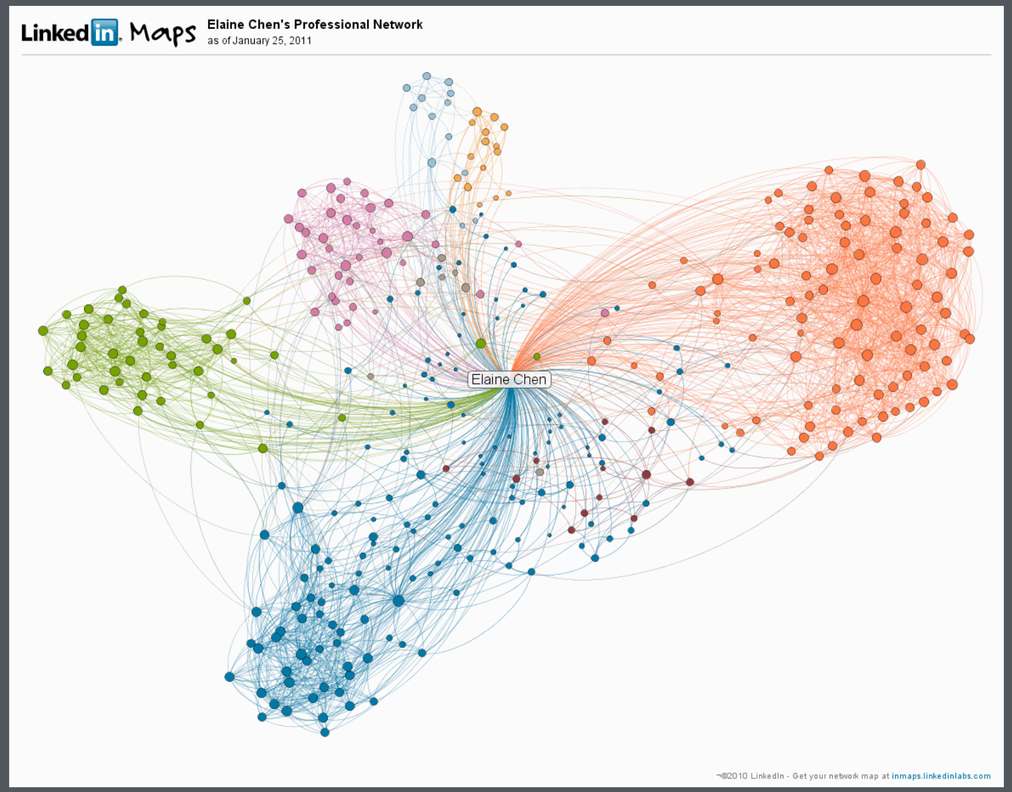
- often illegal
- many choices

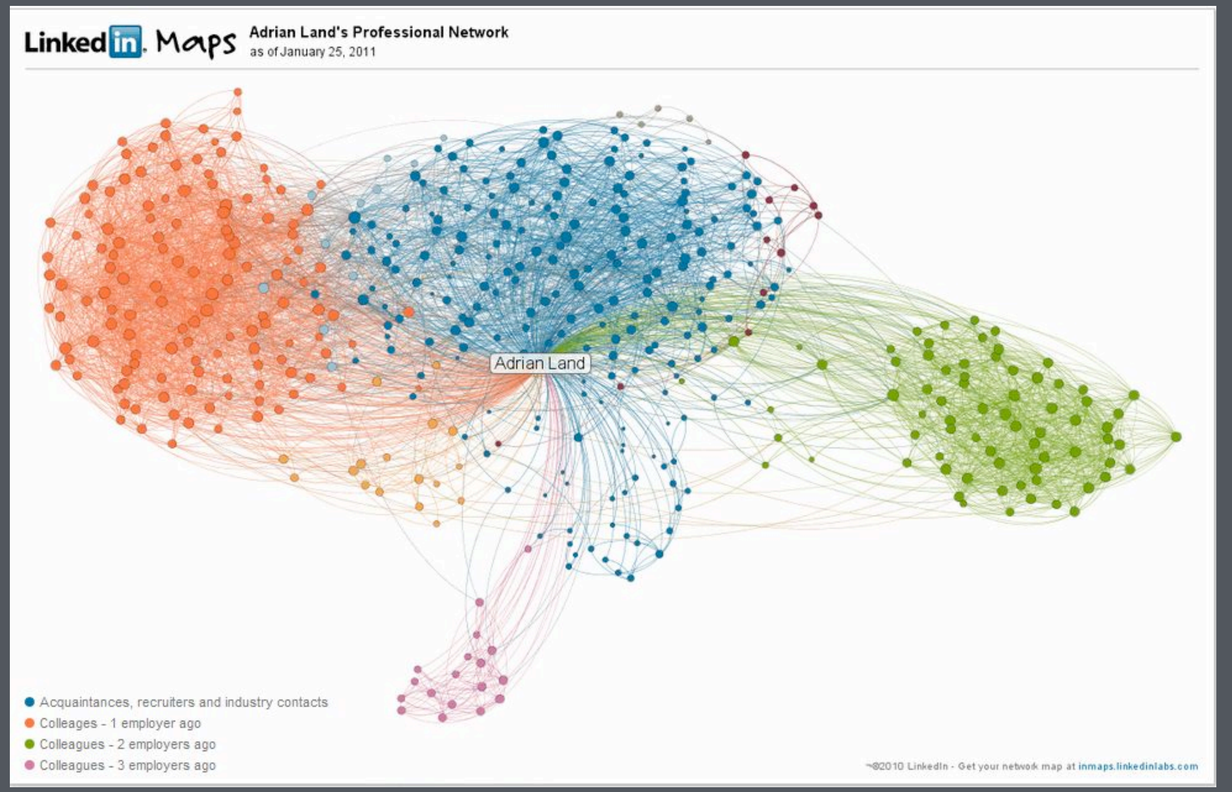
```
// unrolling + scalar replacement
for(i = 0; i < N; i += MU) {
    for(j = 0; j < N; j += NU) {
      for(k = 0; k < N; k += KU) {
        t1 = A[i*N + k];                    } load
        t2 = A[i*N + k + 1];
        t3 = A[i*N + k + 2];
        t4 = A[i*N + k + 3];
        t5 = A[(i + 1)*N + k];
        <more copies>

        t10 = t1  * t9;                     } compute
        t17 = t17 + t10;
        t21 = t1  * t8;
        t18 = t18 + t21;
        t12 = t5  * t9;
        t19 = t19 + t12;
        t13 = t5  * t8;
        t20 = t20 + t13;
        <more ops>

        C[i*N + j]         = t17;           } store
        C[i*N + j + 1]     = t18;
        C[(i+1)*N + j]     = t19;
        C[(i+1)*N + j + 1] = t20;
      }
    }
}
```


"Current practice: Thousands of programmers re-implement and reoptimize the same functionality for every new processor and for every new processor generation."

Markus Püschel

TU Delft

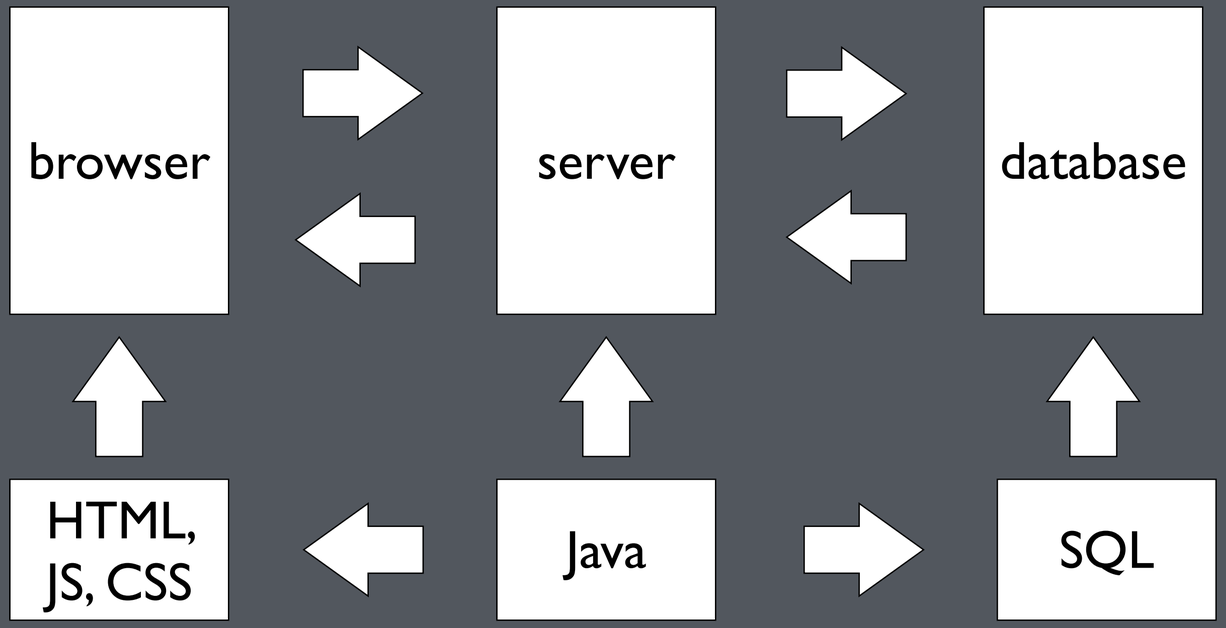




"The enormous growth in data leads to large underlying graphs which require huge amounts of computational power to analyze. While modern commodity computer systems provide a significant amount of computational power measured in the tera-flops, efficient execution of graph analysis algorithms on large datasets remains difficult on these systems [26]. ... **it is often challenging to implement a graph analysis algorithm in an efficient way**."

Sungpack Hong et al. (2012)

TU Delft



Data Persistence

Routing

Search

Access Control

Injection Attacks

Data Binding

Data Validation

XSS

...

...

Late Failure Detection in Web Applications

	Manifestation			Retraceability			Clarity		
Category	**Ra**	**Se**	**Li**	**Ra**	**Se**	**Li**	**Ra**	**Se**	**Li**
Data model									
Properties of non-existing types	R	C	C	-	+	+	-	+	+
Invalid inverse properties	R	D	C	-	+	+	+/-	+	+
Invalid data validation	R	C/D	C/D	-	+/-	+/-	-	+	+
User interface									
Invalid page elements	R	R	R	+	+	-	-	+	-
Invalid element nesting	B	B	B	-	-	-	-	-	-
Invalid references to data model	R	R	R	+	-	+	-	+	+
Invalid links to pages	R	B	B	+	-	-	-	-	-
Invalid links to actions	R	R	R	-	+	-	+	-	-
Application logic									
Invalid references to data model	R	C	C	+	+	+	-	+	+
Invalid redirect from actions	R	R	R	-	-	-	-	-	-
Invalid data binding	R	NA	NA	-	NA	NA	-	NA	NA
Access control									
References to data model	R	R	C	+	-	+	-	-	+

Ra = Ruby on Rails, Se = Seam, Li = Lift

B = Browser, C = Compile, D = Deploy

NA = Not applicable, R = Runtime

Zef Hemel, Danny M. Groenewegen, Lennart C. L. Kats, Eelco Visser. **Static consistency checking of web applications with WebDSL**. Journal of Symbolic Computation, 46(2):150-182, 2011.

TU Delft

parallel programming (multi-core machines)

distributed systems (web, mobile)

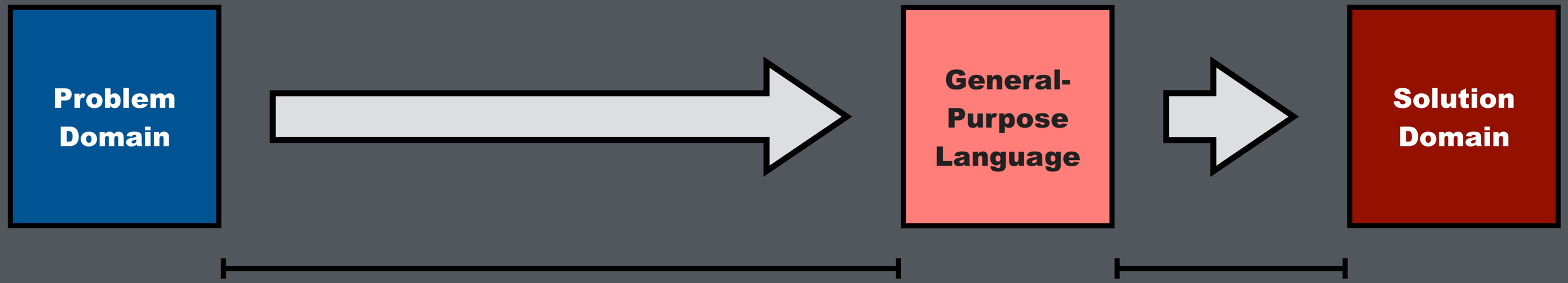
data persistence / services / invariants

security / authentication / authorization / access control

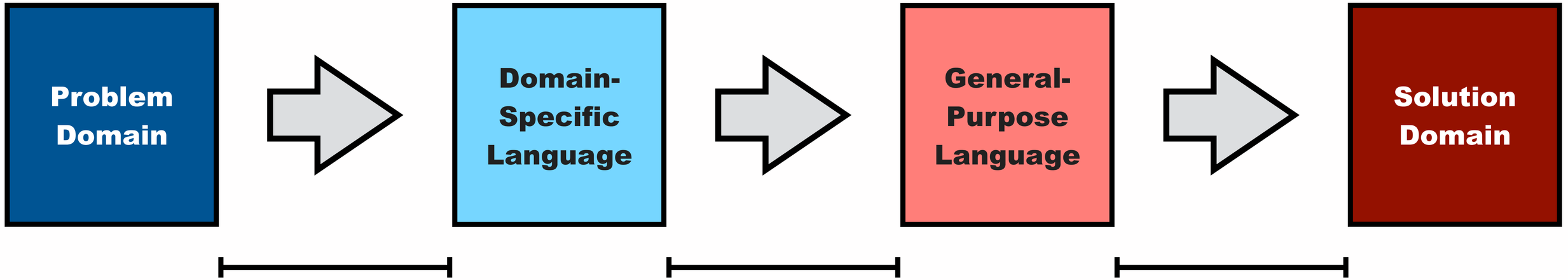
embedded software (resource constraints)

big data analytics (scalability)

programming quantum computers



"A programming language is low level when its programs require attention to the irrelevant". -- Alan Perlis, 1982



Domain-specific language (DSL)

noun

1. a programming language that provides notation, analysis, verification, and optimization specialized to an application domain
2. result of linguistic abstraction beyond general-purpose computation

Program Generation in Spiral

Transform $\mathbf{DFT}_8$

Decomposition rules (algorithm knowledge)

Algorithm *(SPL)*

$$(\mathbf{DFT}_2 \otimes I_4)\, T_4^8\, (I_2 \otimes ((\mathbf{DFT}_2 \otimes I_2)\, T_2^4\, (I_2 \otimes \mathbf{DFT}_2)\, L_2^4))\, L_2^8$$

↻ parallelization vectorization

Algorithm *(Σ-SPL)*

$$\sum (S_j\ \mathbf{DFT}_2\ G_j) \sum (\sum (S_{k,l}\, \mathrm{diag}(\mathrm{t}_{k,l})\ \mathbf{DFT}_2\ G_l)\ \sum (S_m\, \mathrm{diag}(\mathrm{t}_m)\ \mathbf{DFT}_2\ G_{k,m}))$$

↻ locality optimization

C Program

```
void sub(double *y, double *x) {
double f0, f1, f2, f3, f4, f7, f8, f10, f11;
    f0 = x[0] - x[3];
    f1 = x[0] + x[3];
    f2 = x[1] - x[2];
    f3 = x[1] + x[2];
    f4 = f1 - f3;
    y[0] = f1 + f3;
    y[2] = 0.7071067811865476 * f4;
    f7 = 0.9238795325112867 * f0;
< more lines>
```

↻ code style
code level optimization

ETH Eidgenössische Technische Hochschule Zürich Swiss Federal Institute of Technology Zurich

Markus Püschel

```
Procedure Compute_BC(
 G: Graph, BC: Node_Prop<Float>(G)) {
  G.BC = 0;          // initialize BC
  Foreach(s: G.Nodes) {
   // define temporary properties
    Node_Prop<Float>(G) Sigma;
    Node_Prop<Float>(G) Delta;
    s.Sigma = 1; // Initialize Sigma for root
   // Traverse graph in BFS-order from s
    InBFS(v: G.Nodes From s)(v!=s) {
      // sum over BFS-parents
      v.Sigma = Sum(w: v.UpNbrs) {w.Sigma};
    }
   // Traverse graph in reverse BFS-order
    InRBFS(v!=s) {
      // sum over BFS-children
      v.Delta = Sum (w:v.DownNbrs) {
         v.Sigma / w.Sigma * (1+ w.Delta)
      };
      v.BC += v.Delta @s; //accumulate BC
} } }
```

Green-Marl: A DSL for Easy and Efficient Graph Analysis

Sungpack Hong, Hassan Chafi, Eric Sedlar, Kunle Olukotun (ASPLOS 2012)

WebDSL: tier-less web programming

Separation of Concerns & Linguistic Integration

```
entity Status {
  text   : WikiText
  author : Person
  validate(text.length() <= 140, "Message can only be 140 characters")
}

extend entity Person {
  following : Set<Person>
}
```

data model with invariants (automatic data persistence)

```
template output(p: Person) {
  navigate profile(p) { output(p.name) }
}
page profile(p: Person) {
  // page body
}
```

statically checked navigation

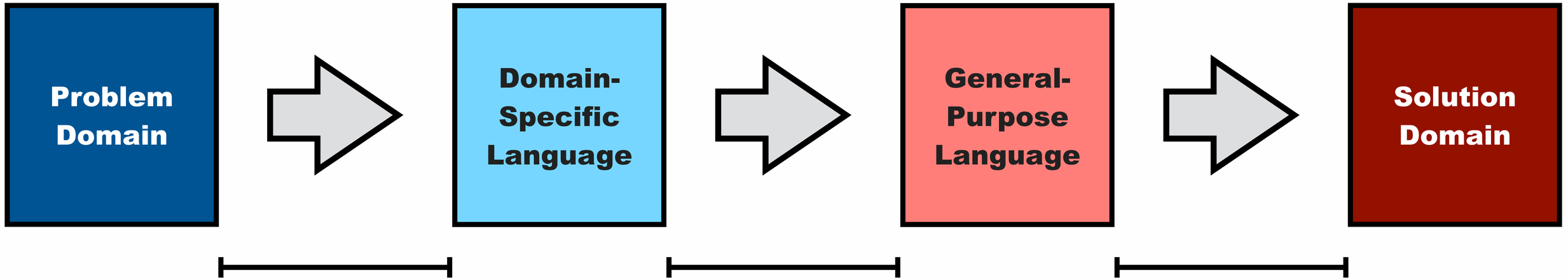
```
function searchPerson(name: String, lim: Int) : List<Person> {
  return select p from Person as p
          where p.name like ~("%" + name + "%")
          limit ~lim;
}
```

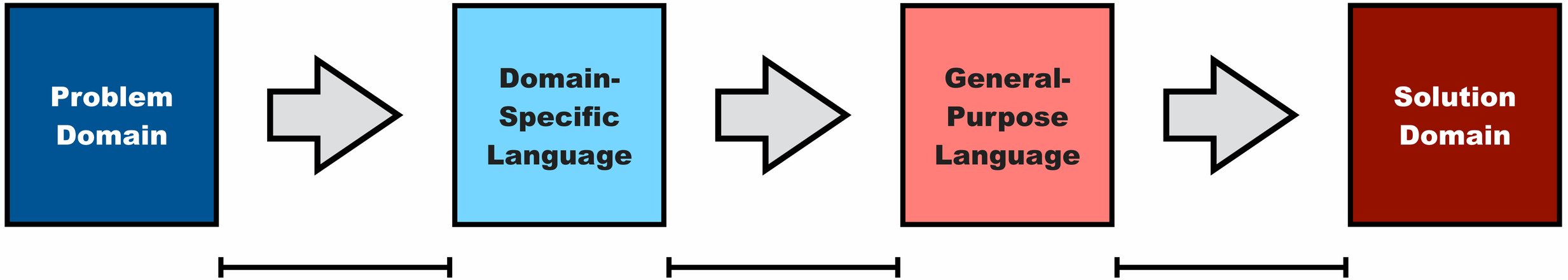
integrated queries (prevent injection attacks)

```
template update() {
  var status := Status{ author := principal() }
   form{
     input(status.text)
     submit action{} { "Share" }
   }
}
```

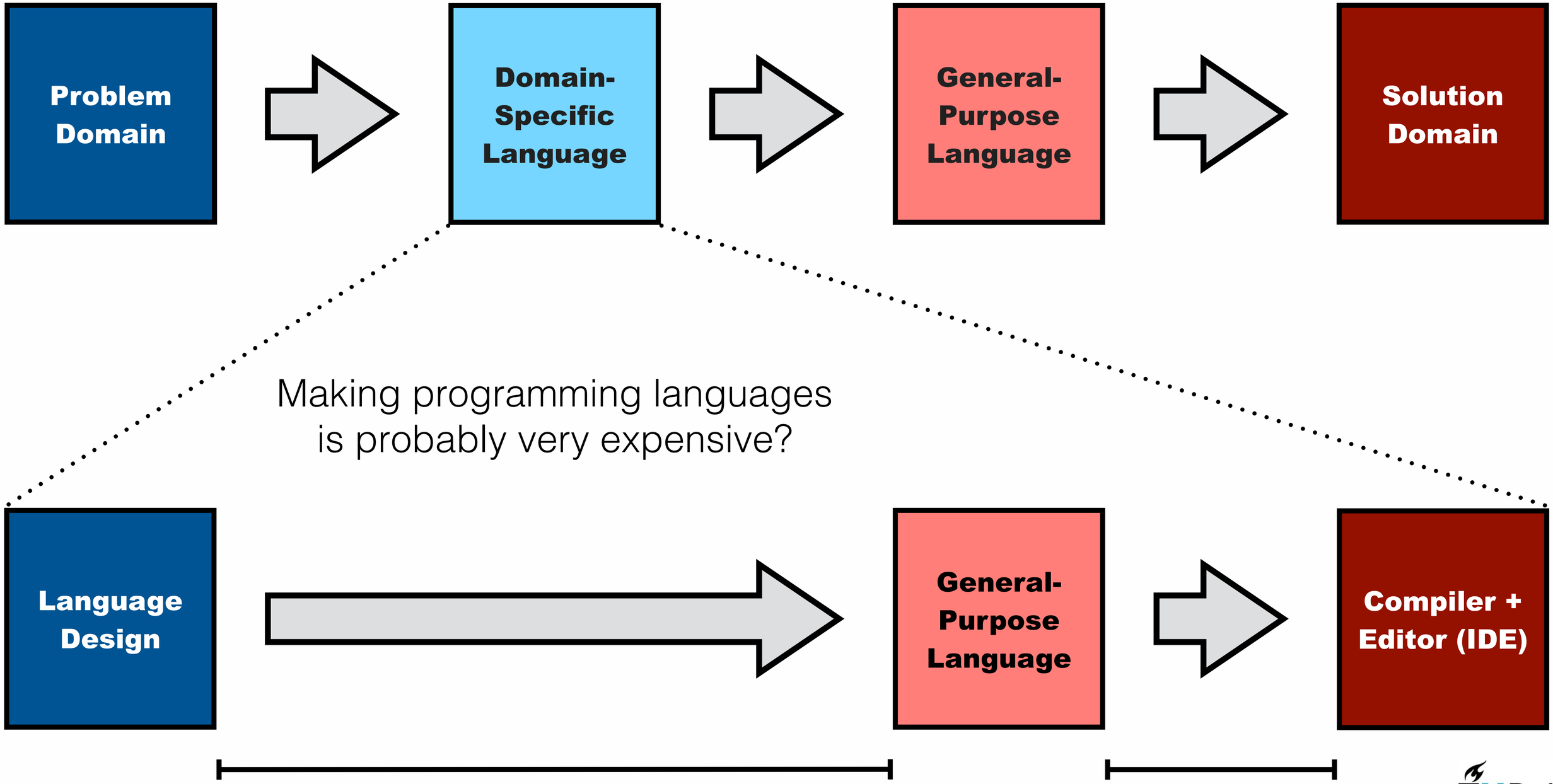
model-view pattern (no controllers)

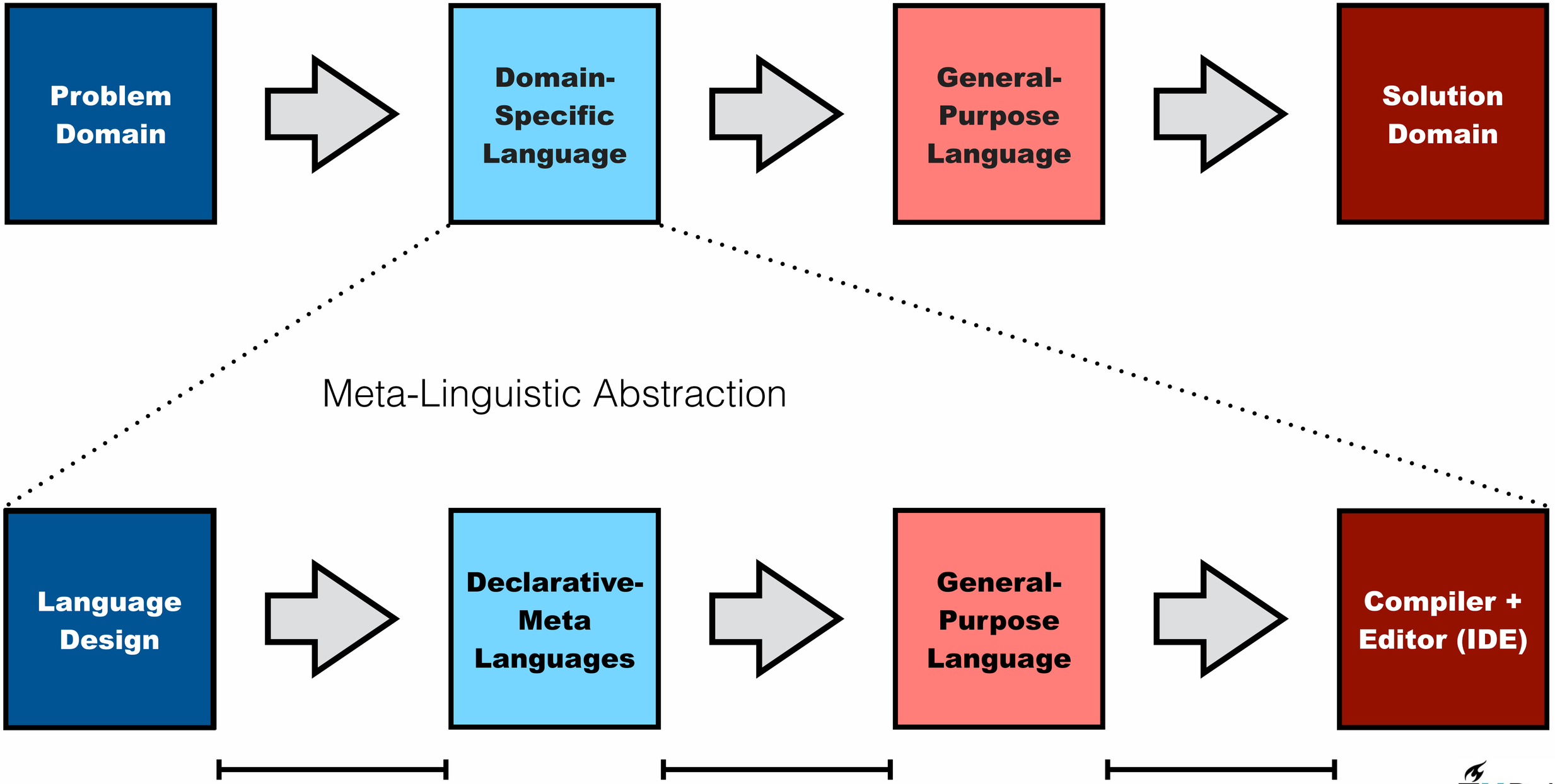
TU Delft


Problem Domain
Domain-Specific Language
General-Purpose Language
Solution Domain

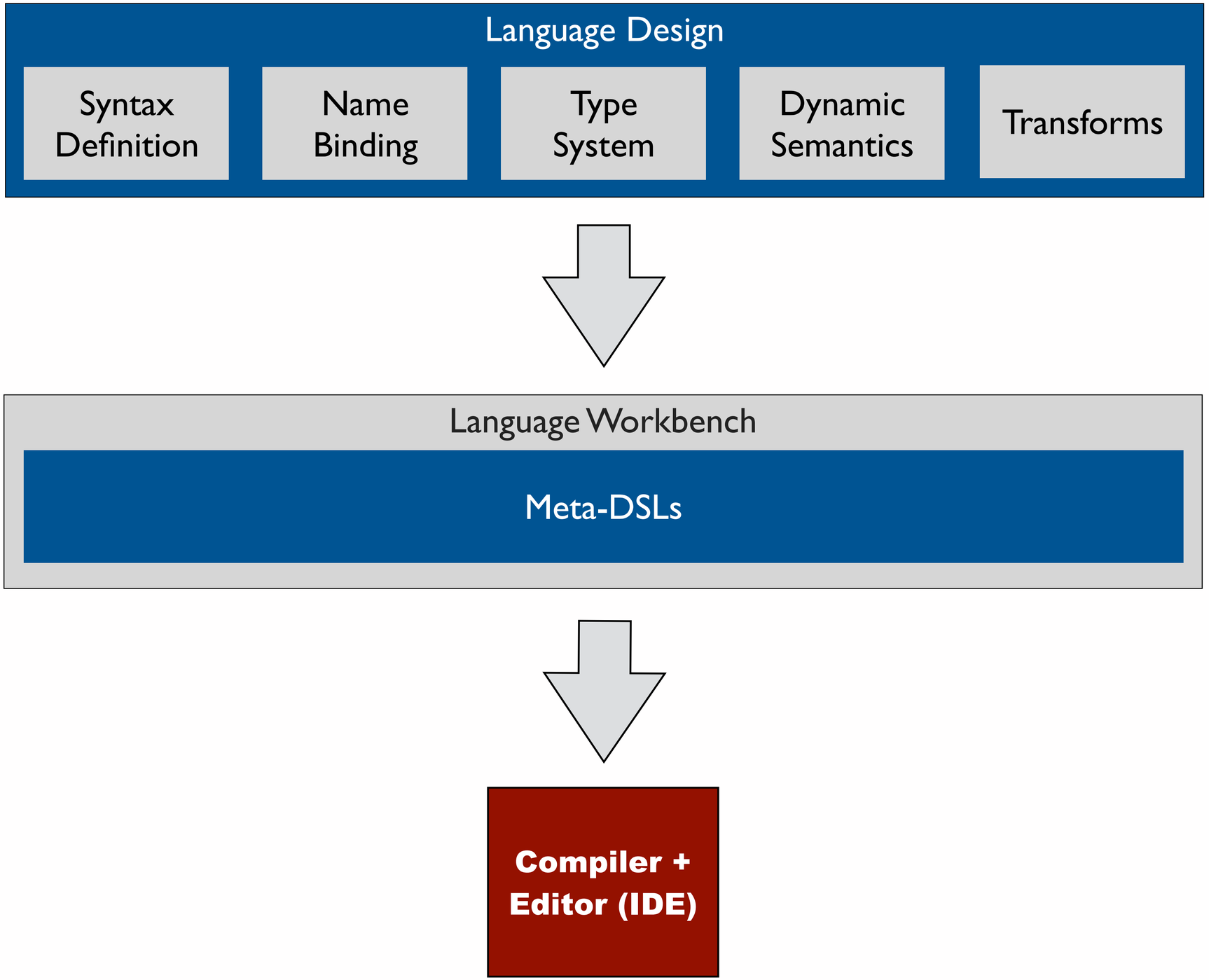


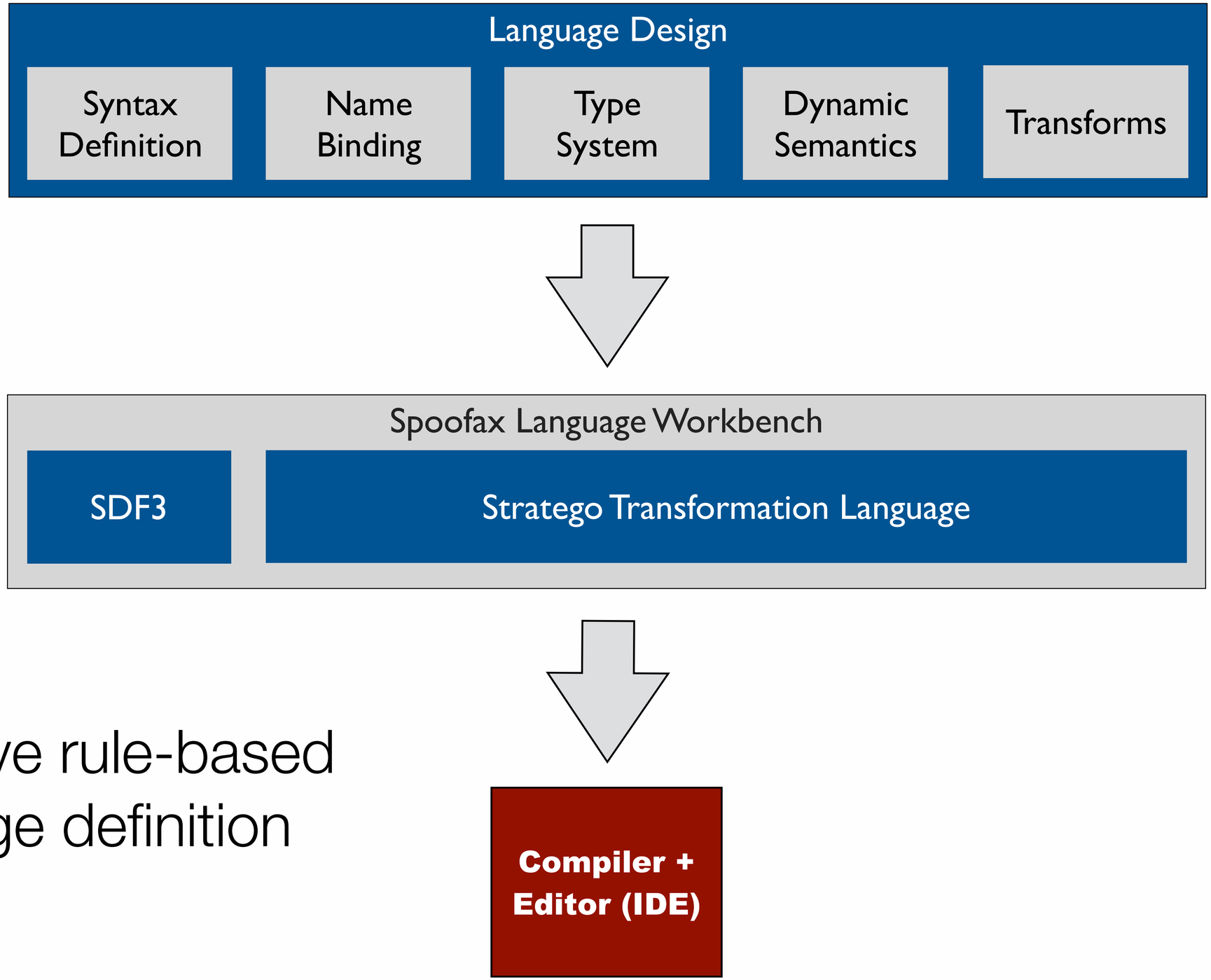
Making programming languages is probably very expensive?


Problem Domain
Domain-Specific Language
General-Purpose Language
Solution Domain
Making programming languages is probably very expensive?
Language Design
General-Purpose Language
Compiler + Editor (IDE)

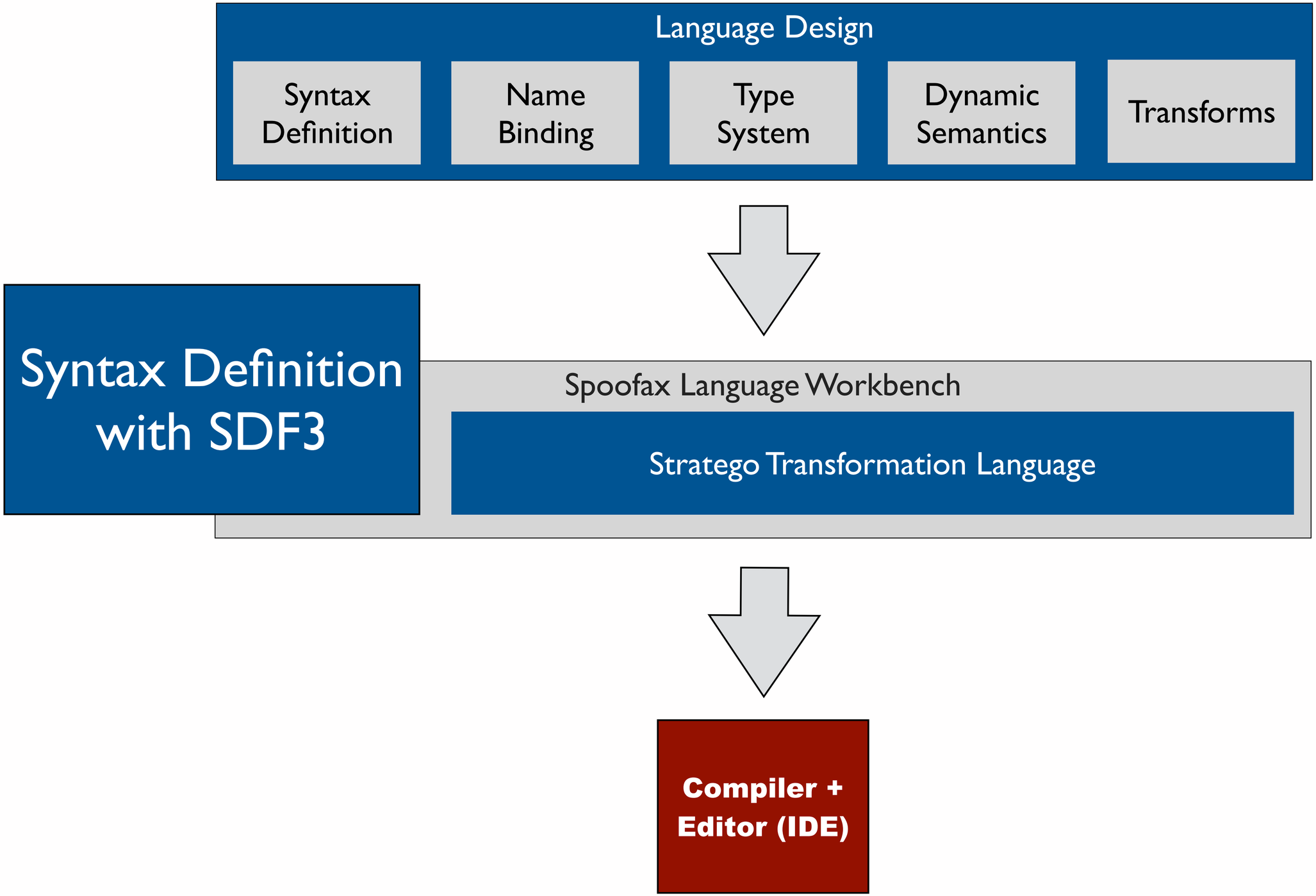

Problem Domain
Domain-Specific Language
General-Purpose Language
Solution Domain
Meta-Linguistic Abstraction
Language Design
Declarative-Meta Languages
General-Purpose Language
Compiler + Editor (IDE)


Language Design
Declarative-Meta Languages
Compiler + Editor (IDE)


Language Design
Syntax Definition
Name Binding
Type System
Dynamic Semantics
Transforms
Language Workbench
Meta-DSLs
Compiler + Editor (IDE)



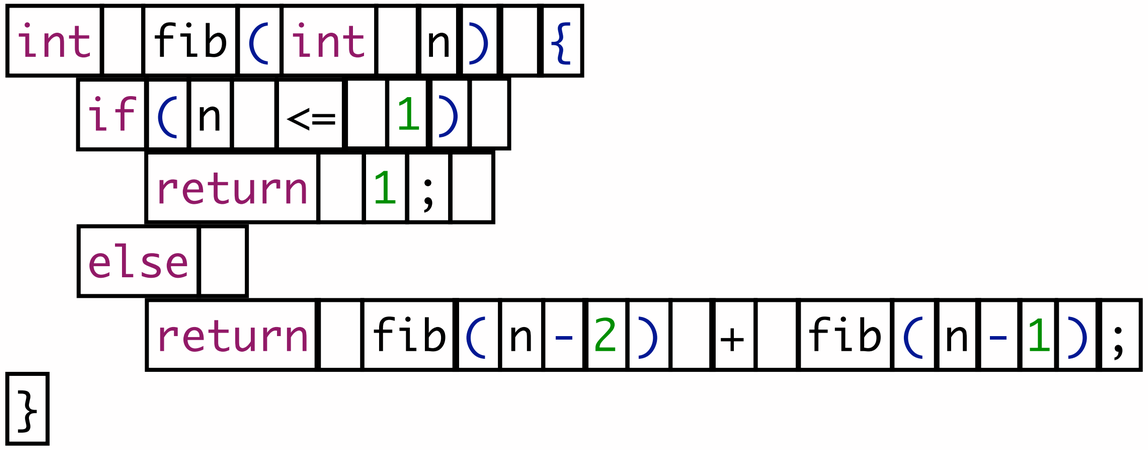
declarative rule-based language definition


Language Design
Syntax Definition
Name Binding
Type System
Dynamic Semantics
Transforms
Syntax Definition with SDF3
Spoofax Language Workbench
Stratego Transformation Language
Compiler + Editor (IDE)

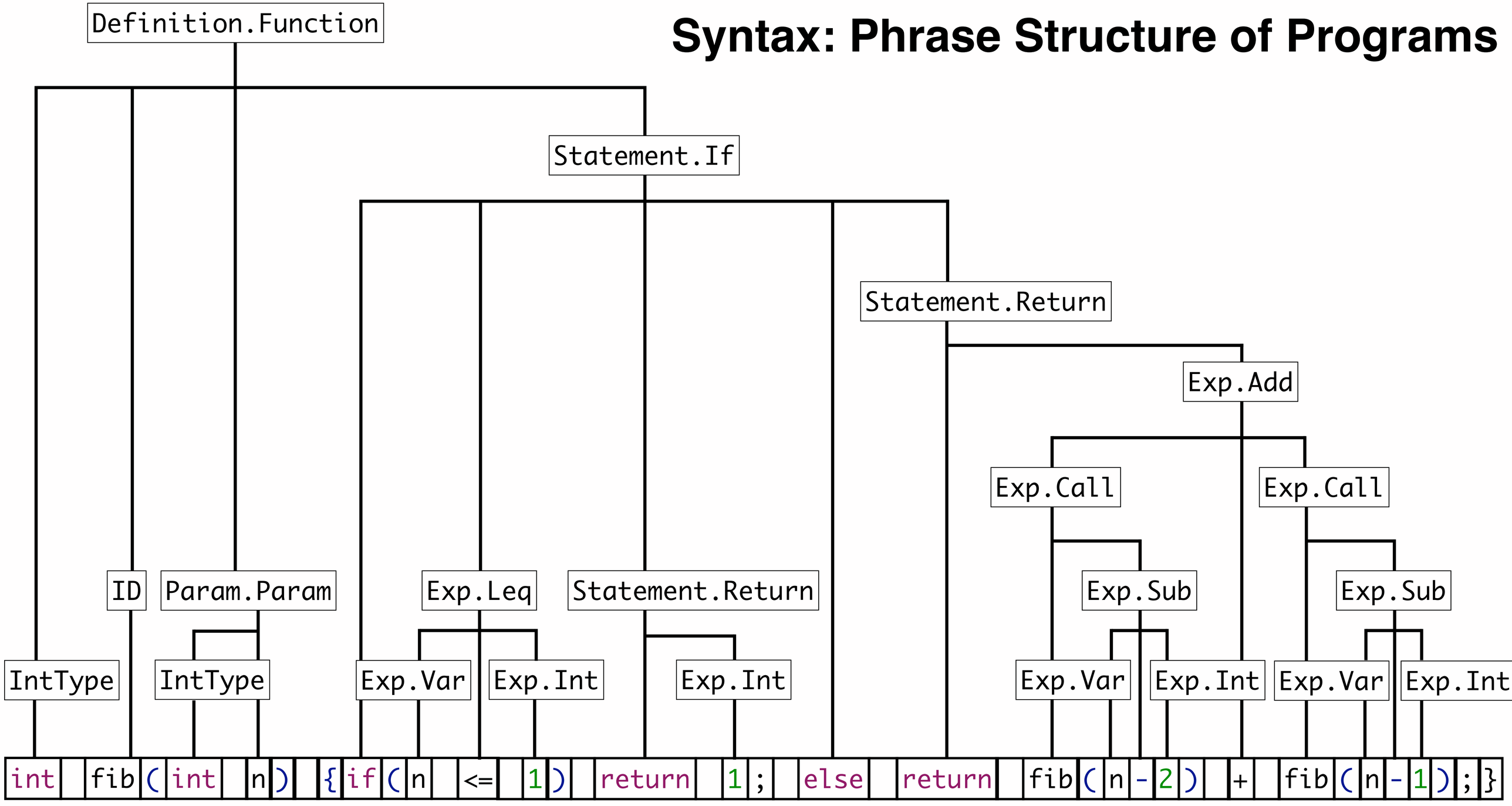
Syntax: Phrase Structure of Programs

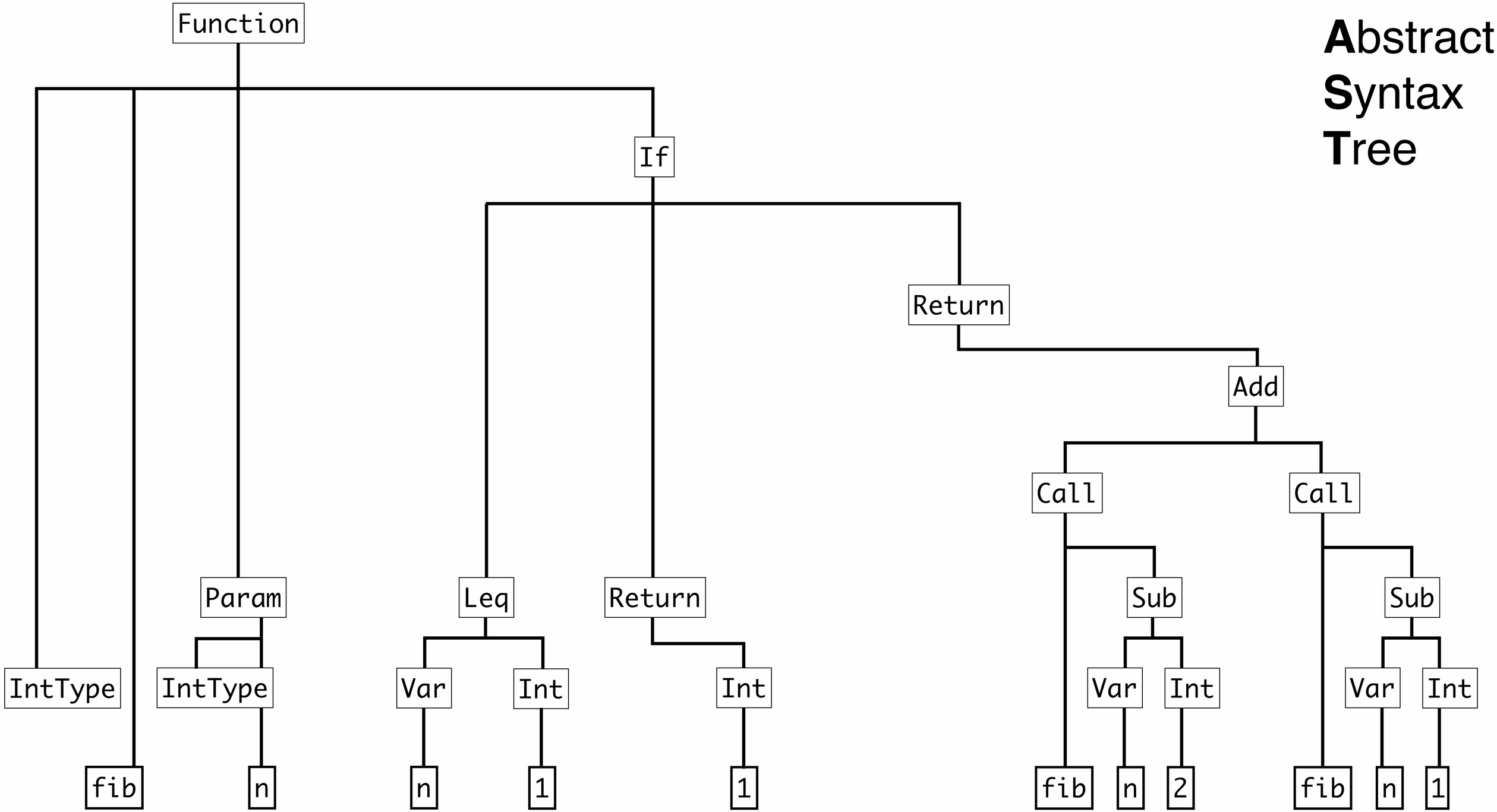
```
int fib(int n) {
  if(n <= 1)
    return 1;
  else
    return fib(n - 2) + fib(n - 1);
}
```

Syntax: Phrase Structure of Programs



Syntax: Phrase Structure of Programs



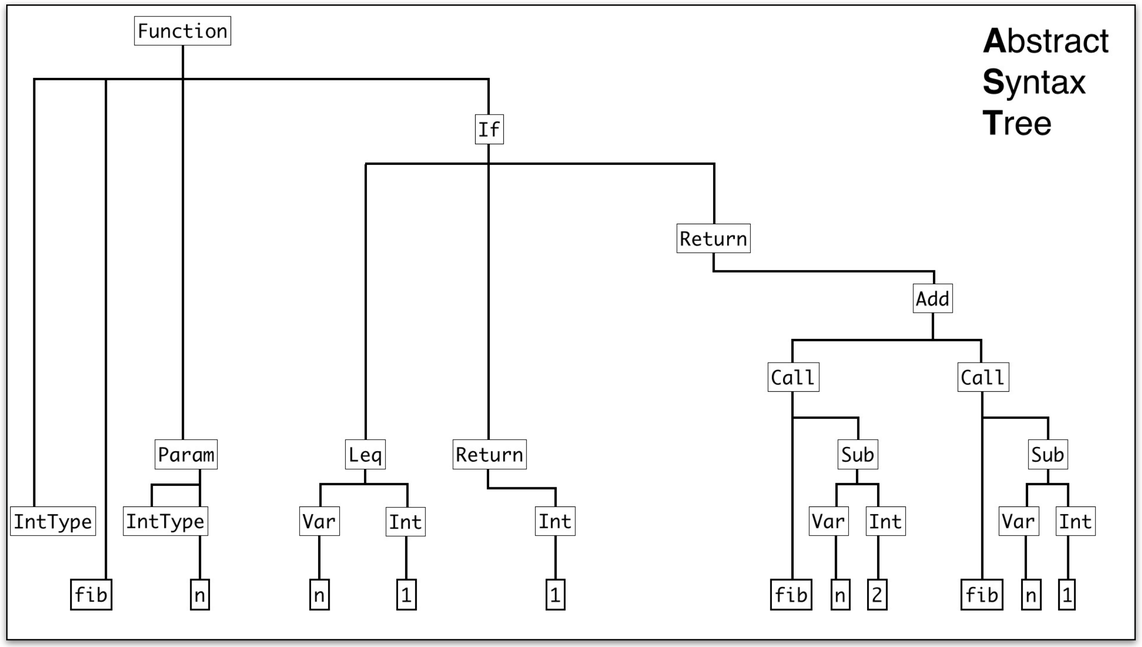


Abstract **S**yntax **T**ree

```
int fib(int n) {
  if(n <= 1)
    return 1;
  else
    return fib(n - 2) + fib(n - 1);
}
```

Text

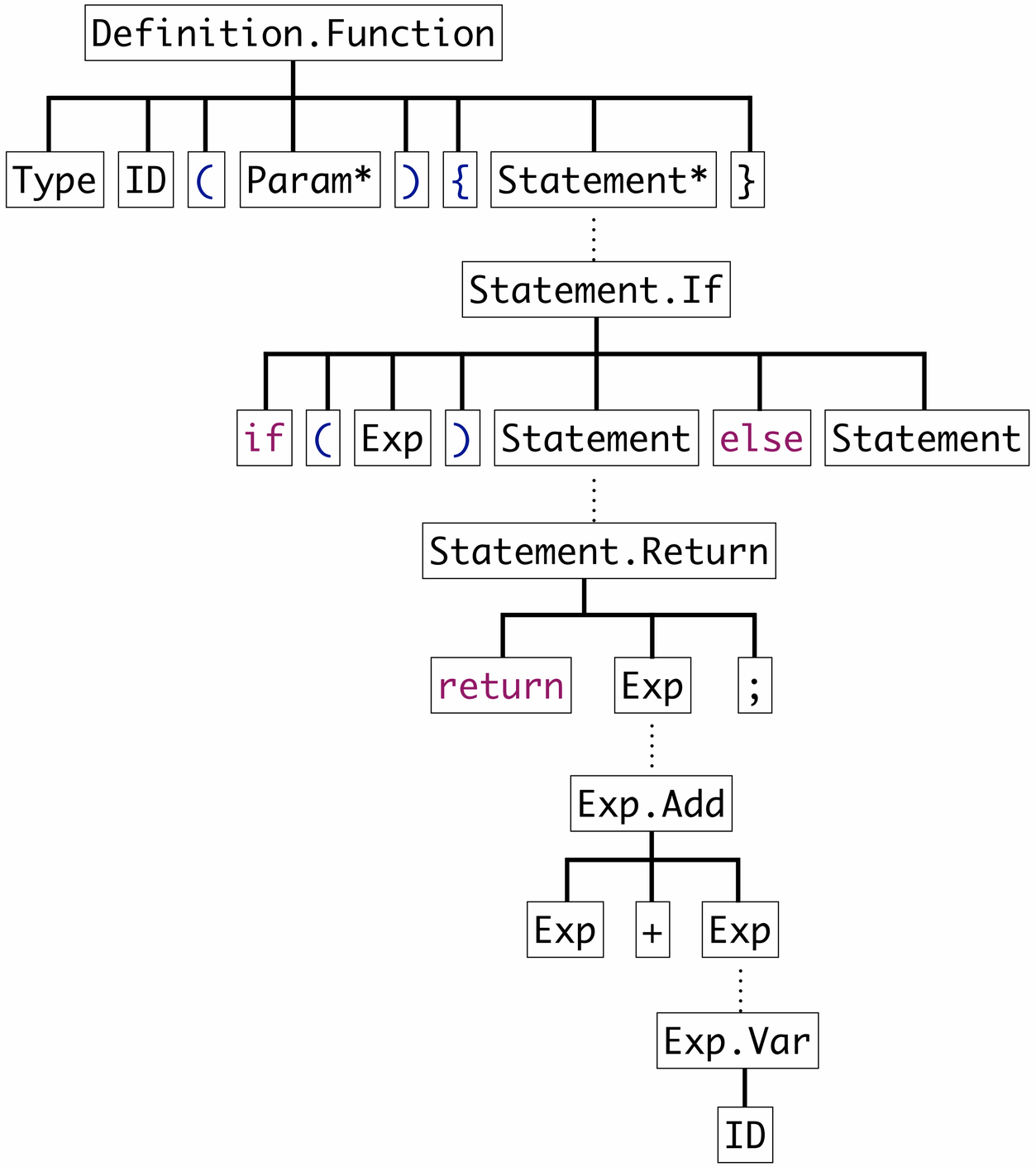
parse




```
Function(
  IntType()
, "fib"
, [Param(IntType(), "n")]
, [ If(
      Leq(Var("n"), Int("1"))
    , Int("1")
    , Add(
        Call("fib", [Sub(Var("n"), Int("2"))])
      , Call("fib", [Sub(Var("n"), Int("1"))])
      )
    )
  ]
)
```

Abstract **S**yntax **T**erm

TU Delft

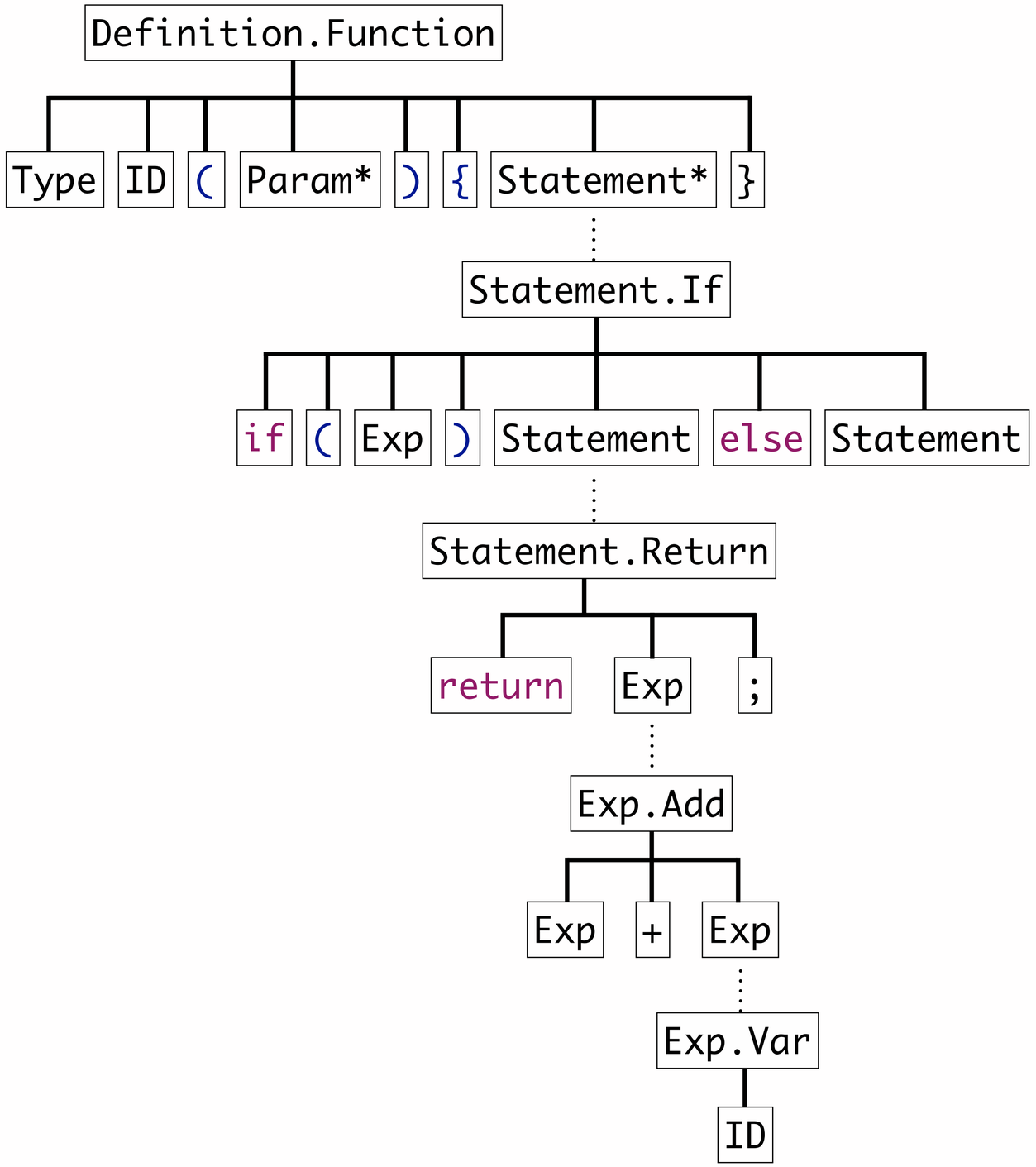


Understanding Syntax = Understanding Tree Structure

parse(prettyprint(t)) = t

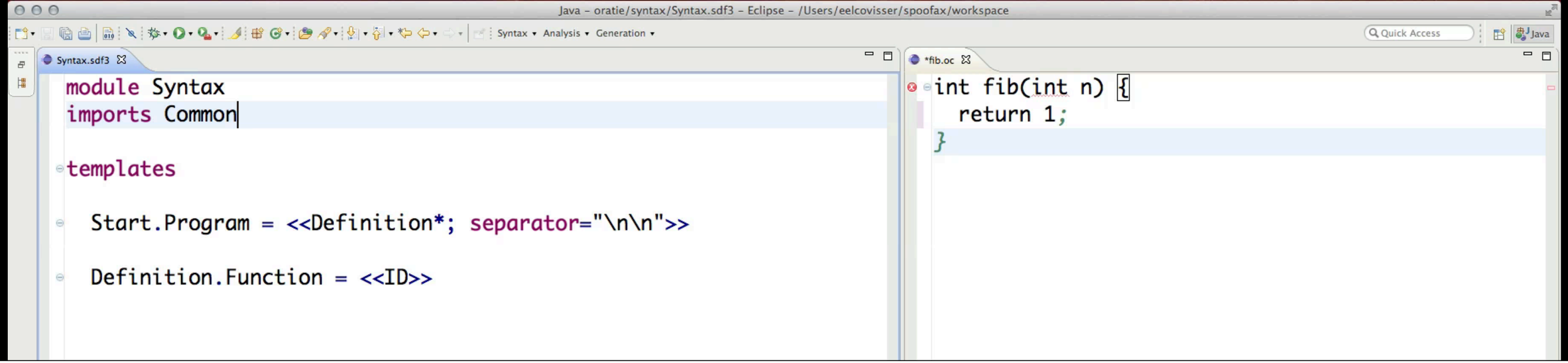
No need to understand how parse **works!**

TU Delft

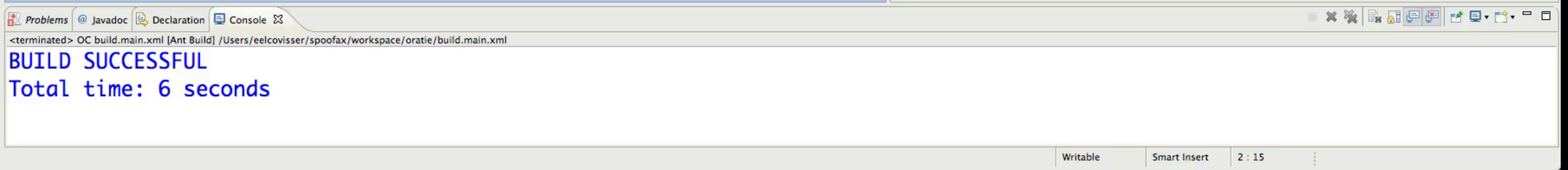


The Syntax Definition Formalism SDF3

```
templates

  Definition.Function = <
    <Type> <ID>(<Param*; separator=",">) {
      <Statement*; separator="\n">
    }
  >

  Statement.If = <
    if(<Exp>)
      <Statement>
    else
      <Statement>
  >

  Statement.Return = <return <Exp>;>

  Exp.Add  = <<Exp> + <Exp>>

  Exp.Var  = <<ID>>
```



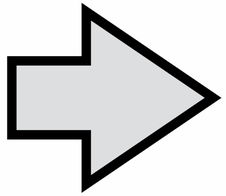
Demo: Syntax Definition in the Spoofax Language Workbench



Multi-Purpose Declarative Syntax Definition

```
Statement.If = <
  if(<Exp>)
    <Statement>
  else
    <Statement>
>
```

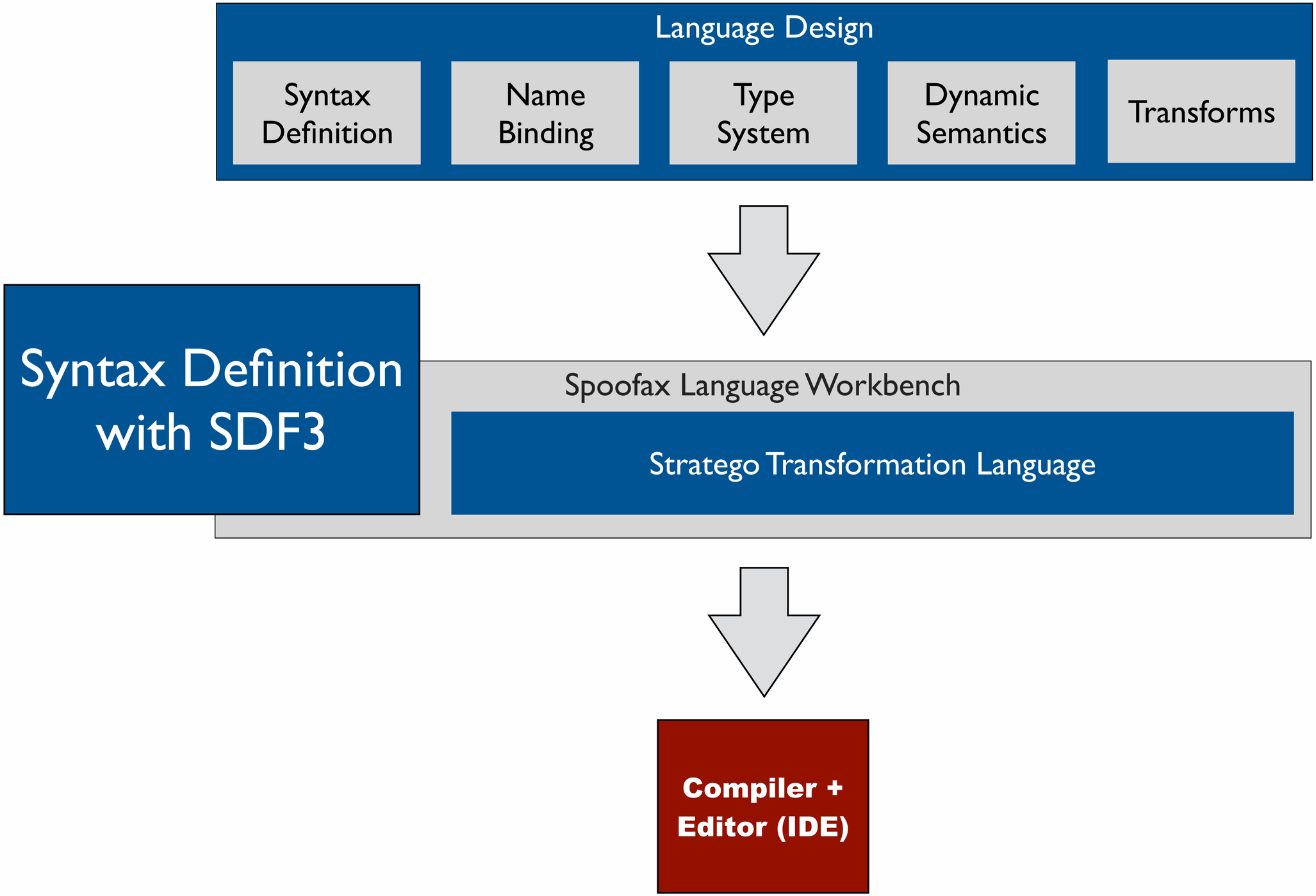
Syntax Definition

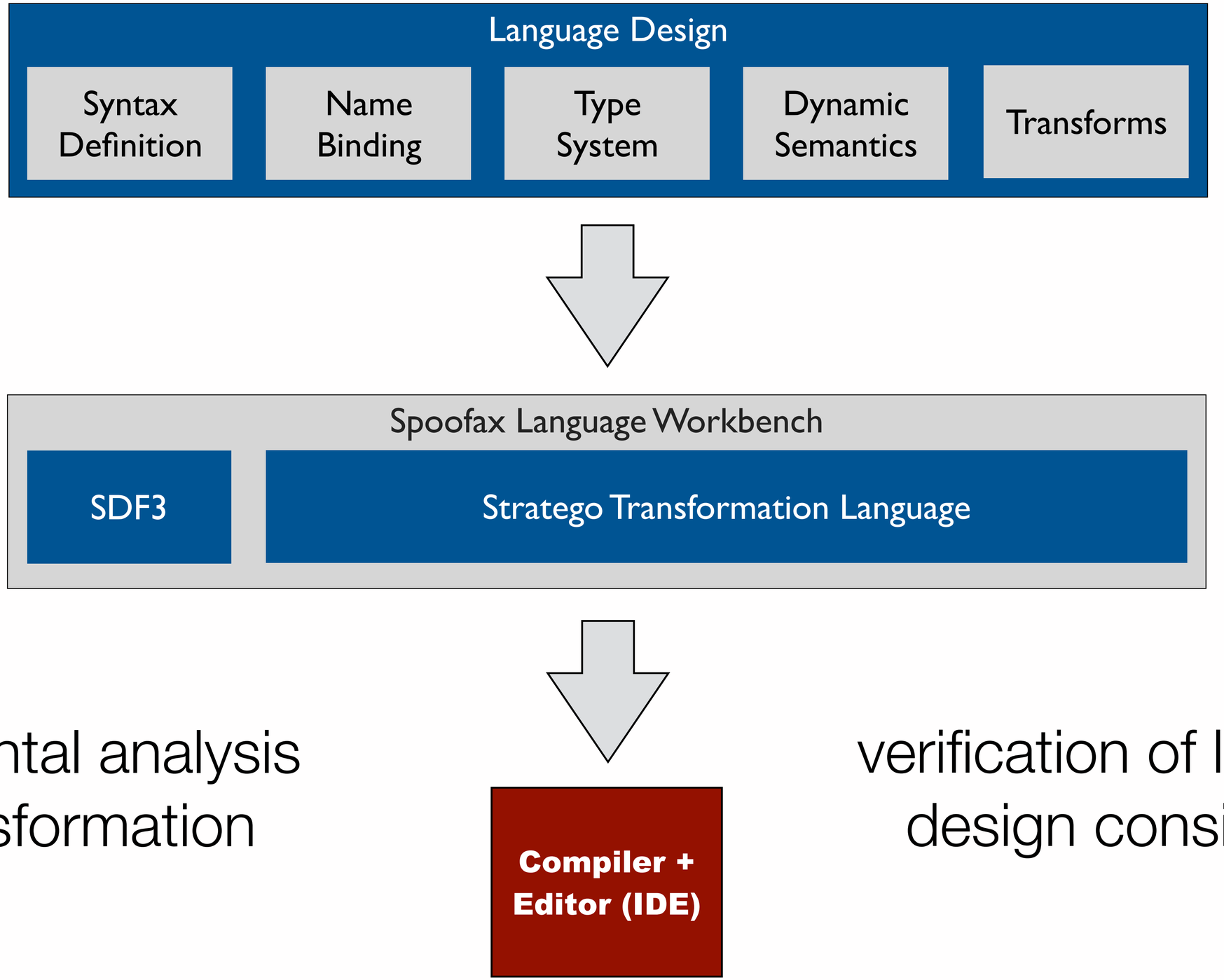


- Parser
- Error recovery rules
- Pretty-Printer
- Abstract syntax tree schema
- Syntactic coloring
- Syntactic completion templates
- Folding rules
- Outline rules

A very incomplete history of SDF

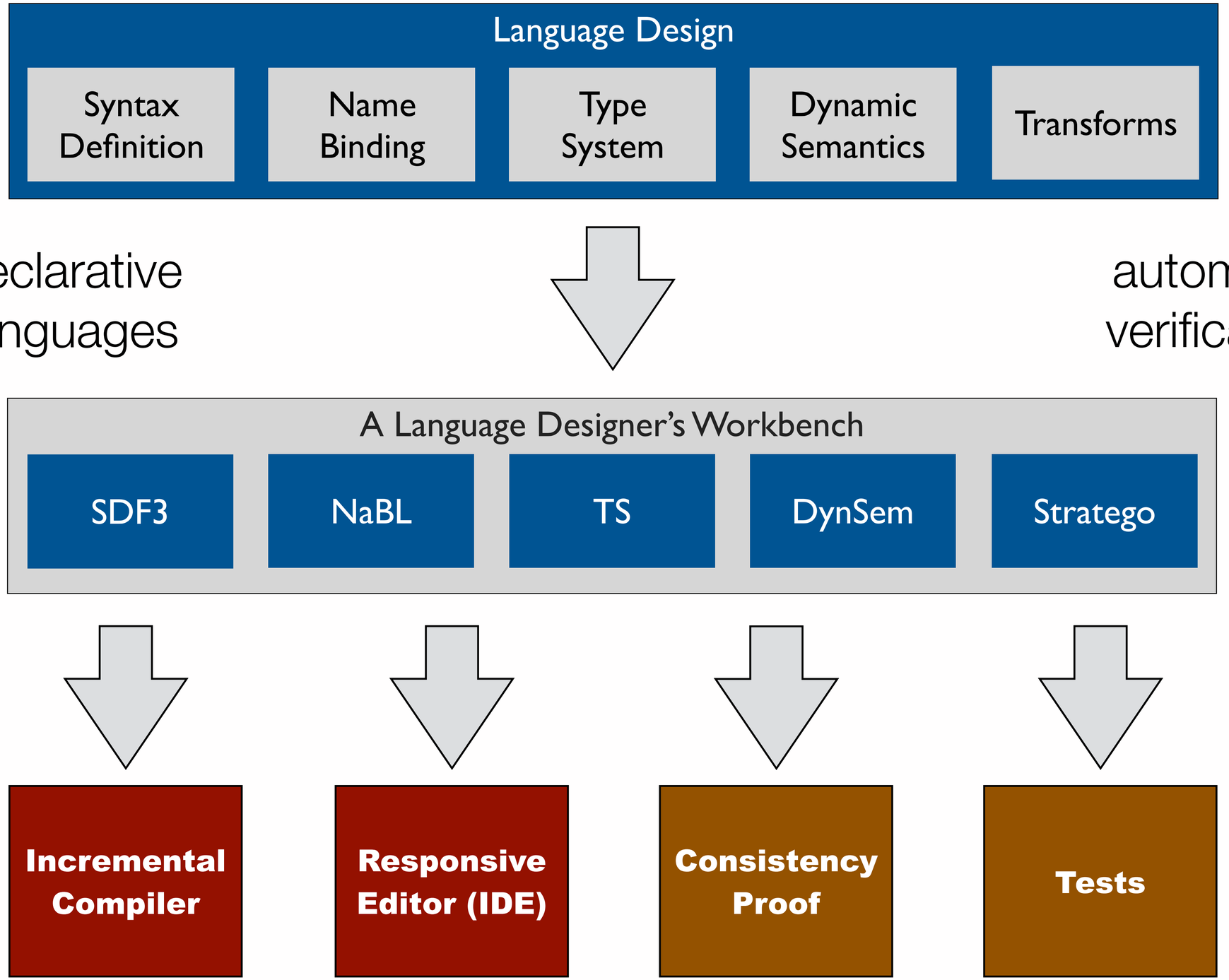
Context-free Grammars	1956	Chomsky
BNF	1963	Backus, Naur
Tomita parsing	1985	Tomita
The Syntax Definition Formalism **SDF**	1988	Heering, Hendriks, Klint, Rekers
Generalized LR Parsing	1992	Rekers
Character level grammars (**SDF2**)	1995	Visser
Scannerless Generalized LR Parsing	1997	Visser
Disambiguation filters	2002	van den Brand, Scheerder, Vinju, Visser
Language embedding	2004	Bravenboer, Visser
SGLR in Java (JSGLR)	2006	Kalleberg
Preventing injection attacks	2010	Bravenboer, Dolstra, Visser
The Spoofax Language Workbench	2010	Kats, Visser
Library-based syntactic language extensibility (SugarJ)	2011	Erdweg, Rendel, Kästner, Ostermann
Error recovery for SGLR	2012	De Jonge, Kats, Visser, Söderberg
Template-based grammar productions (**SDF3**)	2012	Vollebregt, Kats, Visser
Layout sensitive generalized parsing	2012	Erdweg, Rendel, Kästner, Ostermann

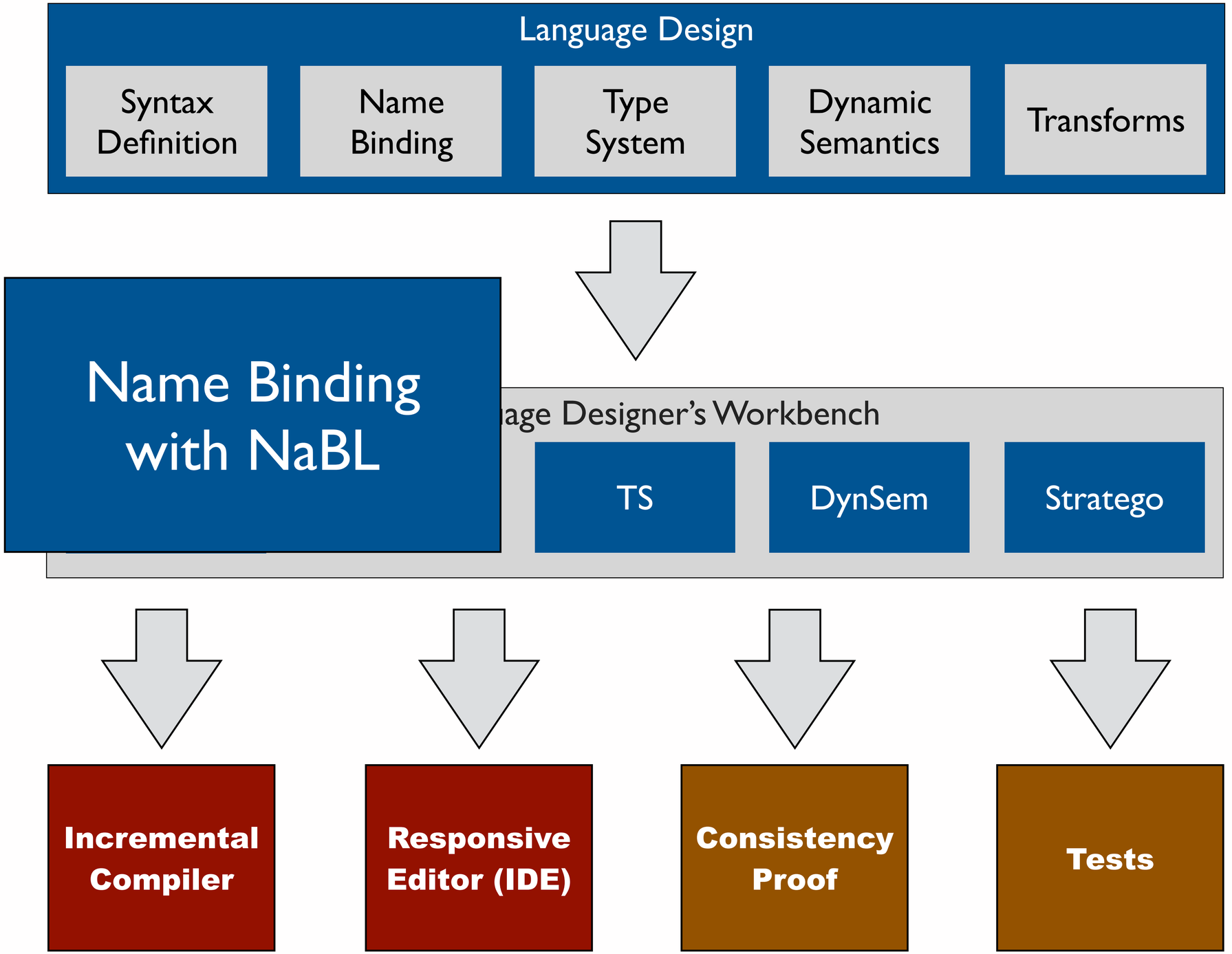

Language Design
Syntax Definition
Name Binding
Type System
Dynamic Semantics
Transforms
Syntax Definition with SDF3
Spoofax Language Workbench
Stratego Transformation Language
Compiler + Editor (IDE)



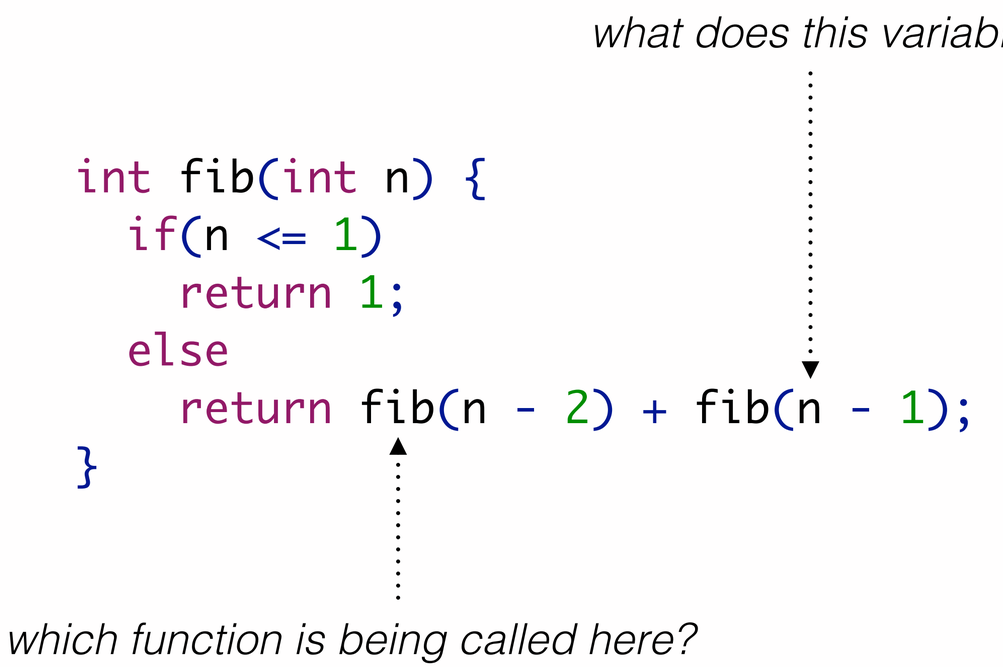
incremental analysis & transformation

verification of language design consistency


Language Design
Syntax Definition
Name Binding
Type System
Dynamic Semantics
Transforms
more declarative meta-languages
automatic verification
A Language Designer's Workbench
SDF3
NaBL
TS
DynSem
Stratego
Incremental Compiler
Responsive Editor (IDE)
Consistency Proof
Tests


Language Design
Syntax Definition
Name Binding
Type System
Dynamic Semantics
Transforms
Name Binding with NaBL
age Designer's Workbench
TS
DynSem
Stratego
Incremental Compiler
Responsive Editor (IDE)
Consistency Proof
Tests

Name Binding & Scope Rules



Needed for

- checking correct use of names and types
- lookup in interpretation and compilation
- navigation in IDE
- code completion

State-of-the-art

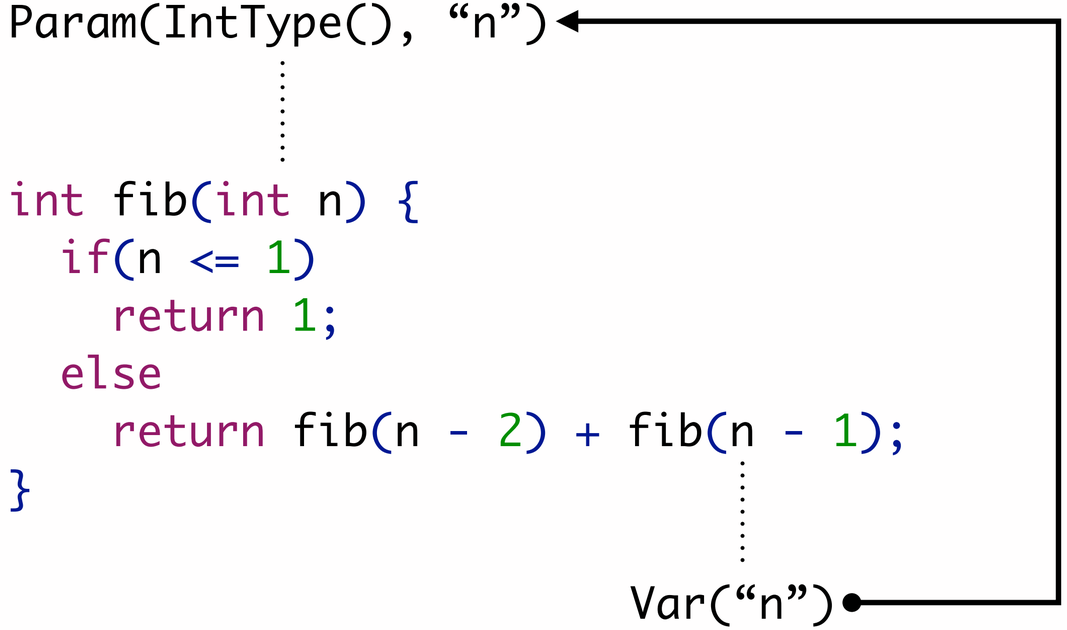
- programmatic encoding of name resolution algorithms

Our contribution

- declarative language for name binding & scope rules
- generation of incremental name resolution algorithm

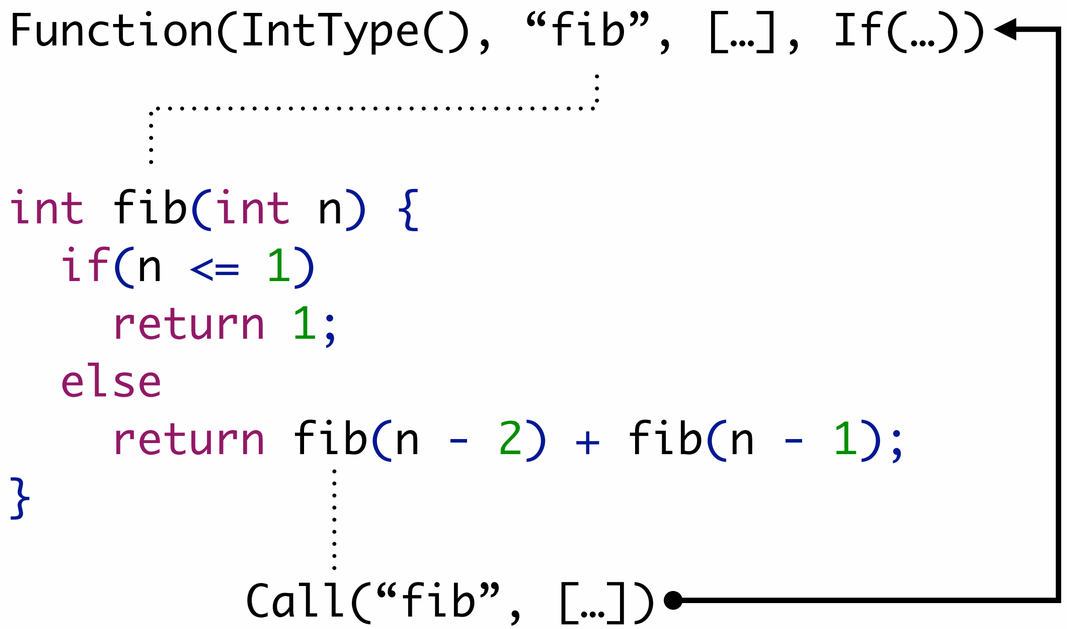
- Konat, Kats, Wachsmuth, Visser (SLE 2012)
- Wachsmuth, Konat, Vergu, Groenewegen, Visser (SLE 2013)

Definitions and References




```
binding rules

  Param(t, name) :
    defines Variable name

  Var(name) :
    refers to Variable name
```

Definitions and References




```
binding rules

  Param(t, name) :
    defines Variable name

  Var(name) :
    refers to Variable name

  Function(t, name, param*, s) :
    defines Function name

  Call(name, exp*) :
    refers to Function name
```

Scope

```
int fib(int n) {
  if(n <= 1)
    return 1;
  else
    return fib(n - 2) + fib(n - 1);
}

int power(int x, int n) {
  if(x <= 0)
    return 1;
  else
    return x * power(x, n - 1);
}
```

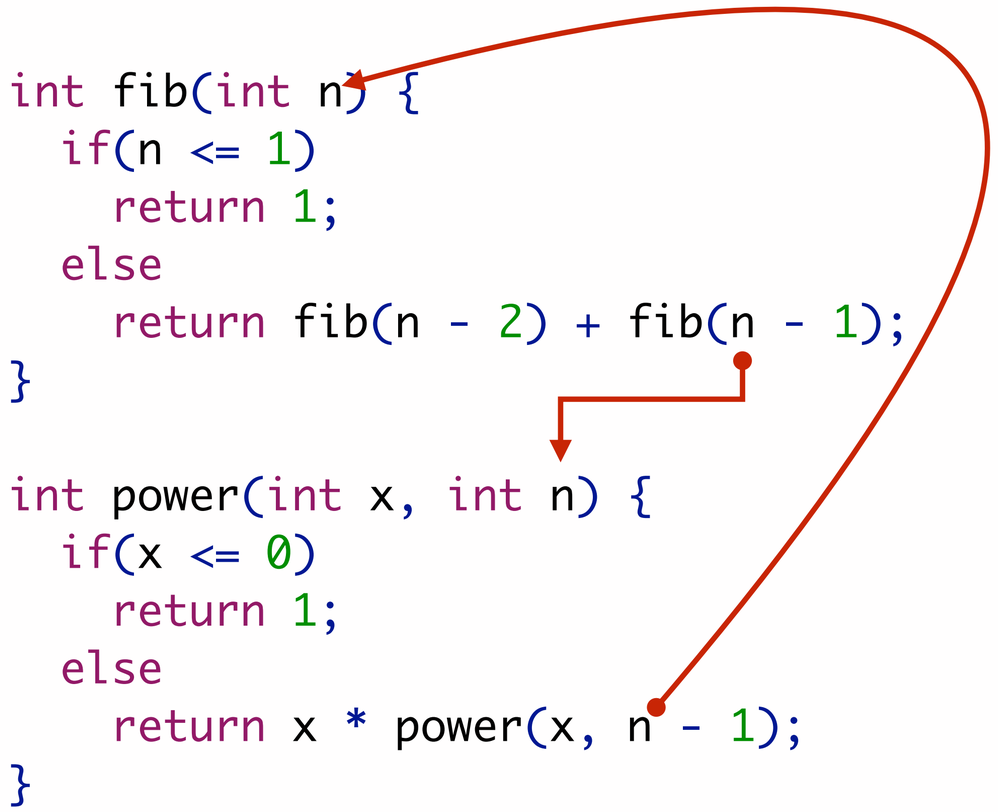
```
binding rules

  Param(t, name) :
    defines Variable name

  Var(name) :
    refers to Variable name

  Function(t, name, param*, s) :
    defines Function name

  Call(name, exp*) :
    refers to Function name
```

Scope

```
int fib(int n) {
  if(n <= 1)
    return 1;
  else
    return fib(n - 2) + fib(n - 1);
}

int power(int x, int n) {
  if(x <= 0)
    return 1;
  else
    return x * power(x, n - 1);
}
```


```
binding rules

  Param(t, name) :
    defines Variable name

  Var(name) :
    refers to Variable name

  Function(t, name, param*, s) :
    defines Function name

  Call(name, exp*) :
    refers to Function name
```

Scope



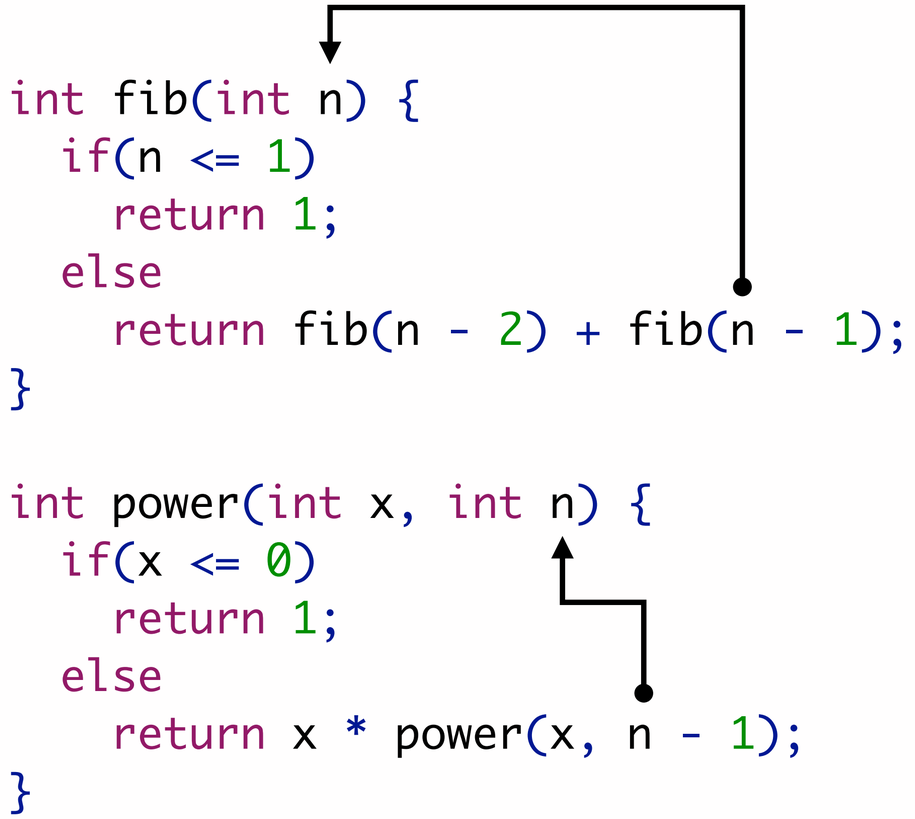
```
int fib(int n) {
  if(n <= 1)
    return 1;
  else
    return fib(n - 2) + fib(n - 1);
}

int power(int x, int n) {
  if(x <= 0)
    return 1;
  else
    return x * power(x, n - 1);
}
```

Wrong!

binding rules

```
Param(t, name) :
  defines Variable name

Var(name) :
  refers to Variable name

Function(t, name, param*, s) :
  defines Function name

Call(name, exp*) :
  refers to Function name
```

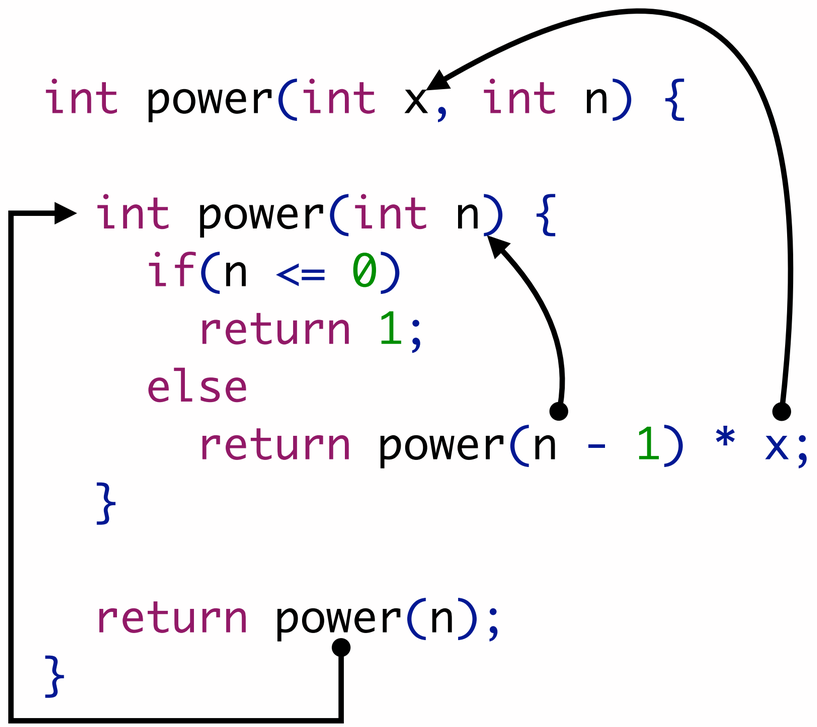
Scope



```
int fib(int n) {
  if(n <= 1)
    return 1;
  else
    return fib(n - 2) + fib(n - 1);
}

int power(int x, int n) {
  if(x <= 0)
    return 1;
  else
    return x * power(x, n - 1);
}
```

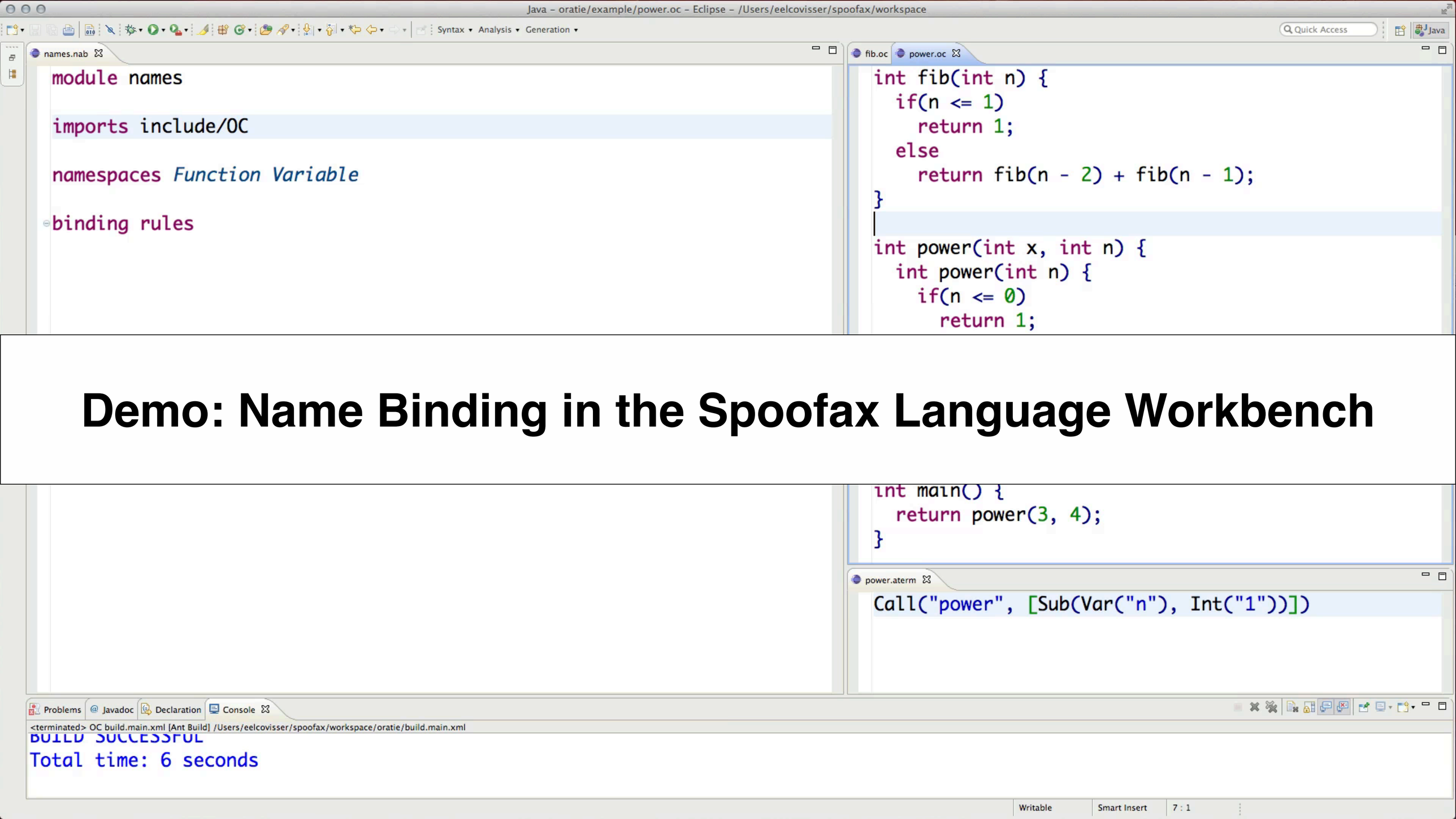
```
binding rules

  Param(t, name) :
    defines Variable name

  Var(name) :
    refers to Variable name

  Function(t, name, param*, s) :
    defines Function name
    scopes Variable

  Call(name, exp*) :
    refers to Function name
```

Scope



binding rules

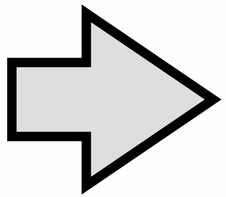
```
Param(t, name) :
  defines Variable name

Var(name) :
  refers to Variable name

Function(t, name, param*, s) :
  defines Function name
  scopes Variable, Function

Call(name, exp*) :
  refers to Function name
```



Demo: Name Binding in the Spoofax Language Workbench

Declarative Name Binding and Scope Rules

```
binding rules

  Param(t, name) :
    defines Variable name

  Var(name) :
    refers to Variable name

  Function(t, name, param*, s) :
    defines Function name
    scopes Variable, Function

  Call(name, exp*) :
    refers to Function name
```

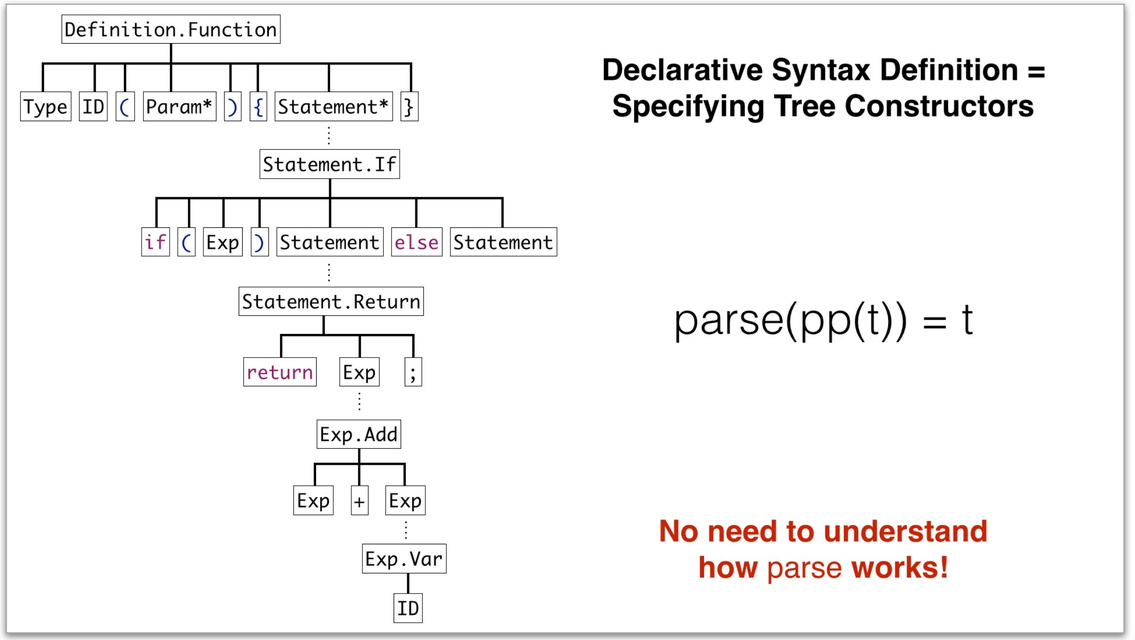


- Incremental name resolution algorithm
- Name checks
- Reference resolution
- Semantic code completion

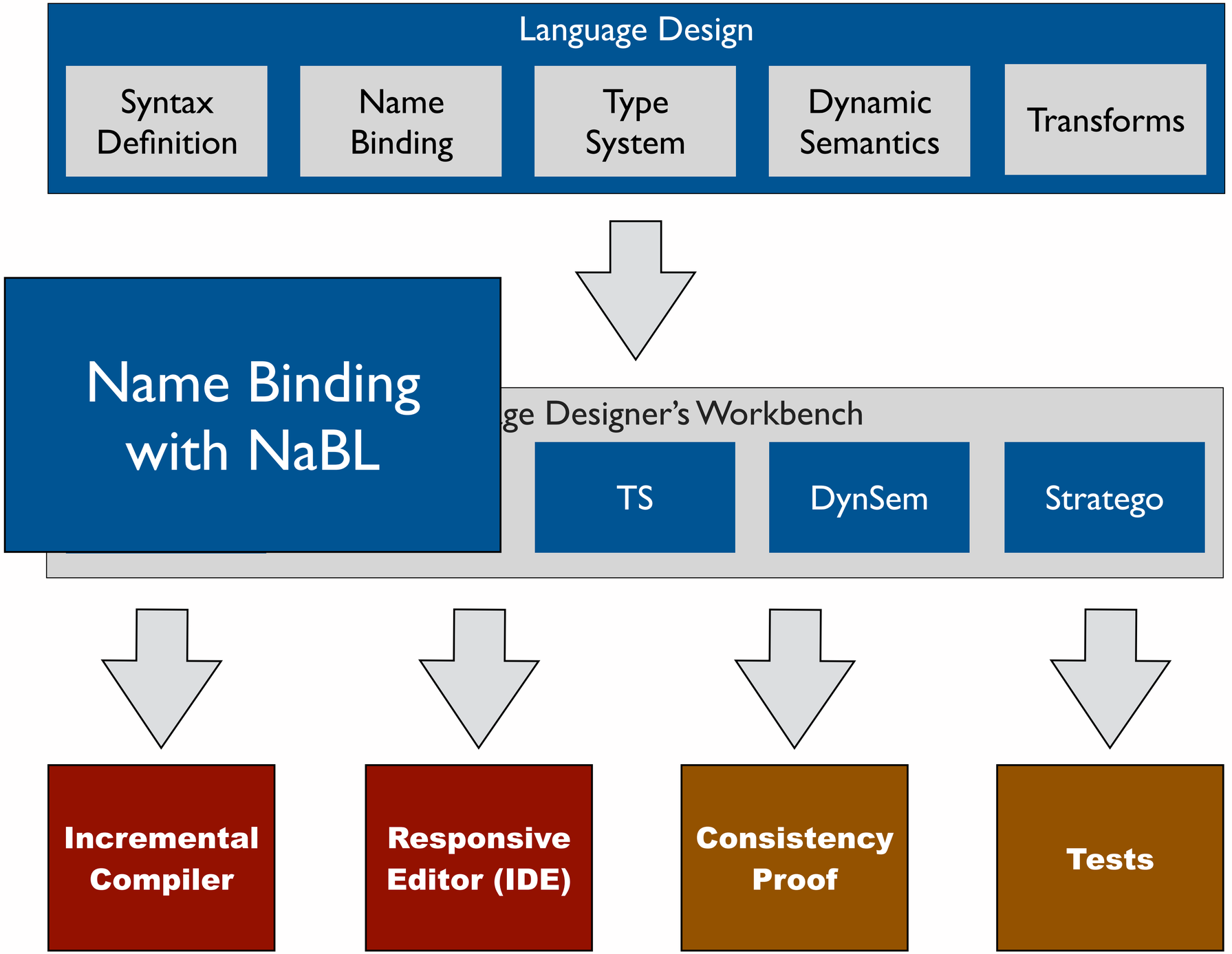
Semantics of Name Binding?

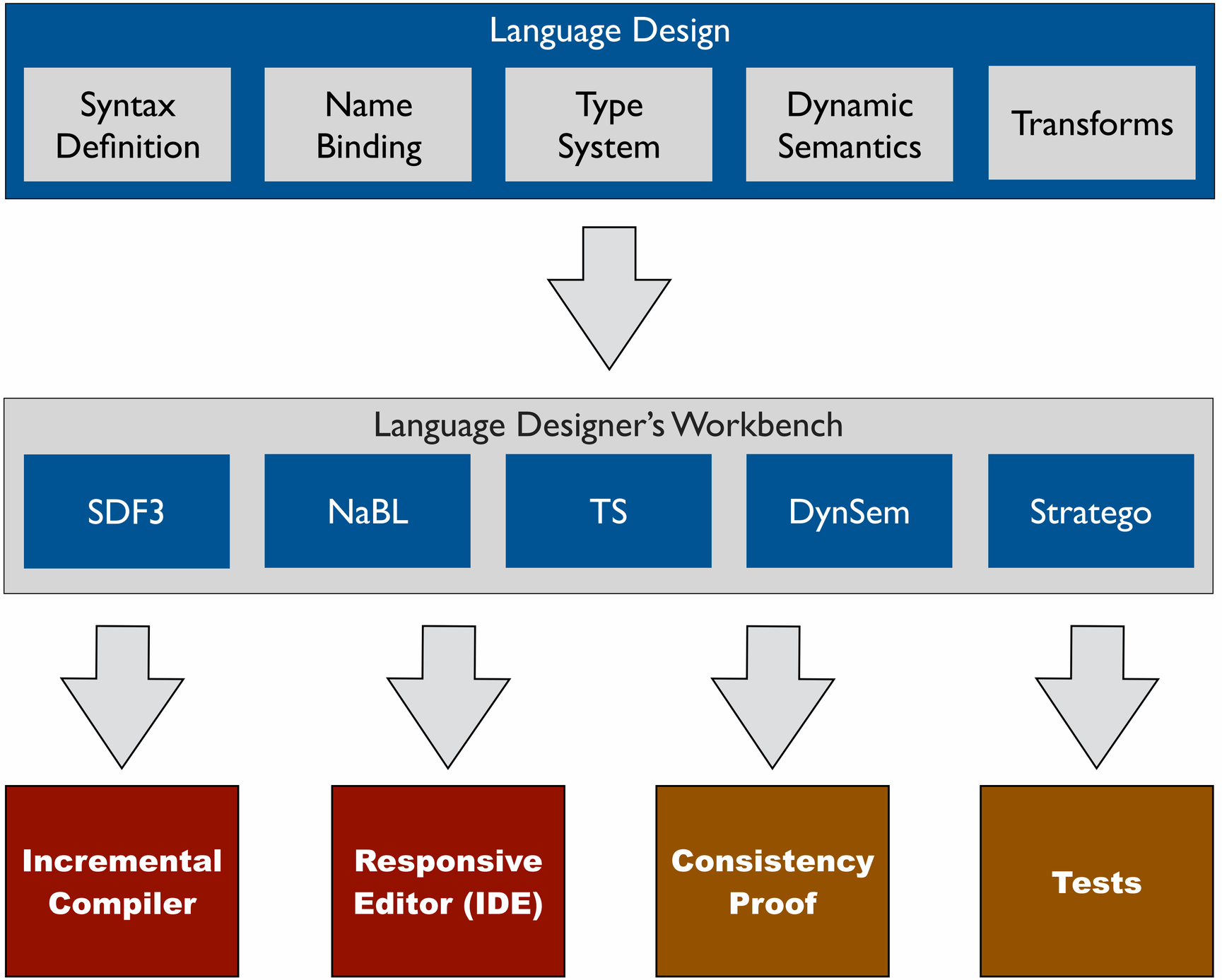
```
binding rules

  Param(t, name) :
    defines Variable name

  Var(name) :
    refers to Variable name

  Function(t, name, param*, s) :
    defines Function name
    scopes Variable, Function

  Call(name, exp*) :
    refers to Function name
```

Research: how to characterize correctness of the result of name resolution without appealing to the algorithm itself?



Analogy: declarative semantics of syntax definition


Language Design
Syntax Definition
Name Binding
Type System
Dynamic Semantics
Transforms
Name Binding with NaBL
ge Designer's Workbench
TS
DynSem
Stratego
Incremental Compiler
Responsive Editor (IDE)
Consistency Proof
Tests


Language Design
Syntax Definition
Name Binding
Type System
Dynamic Semantics
Transforms
Language Designer's Workbench
SDF3
NaBL
TS
DynSem
Stratego
Incremental Compiler
Responsive Editor (IDE)
Consistency Proof
Tests

Spoofax 2.0: Prototype the Language Designer's Workbench

capturing understanding of language definition

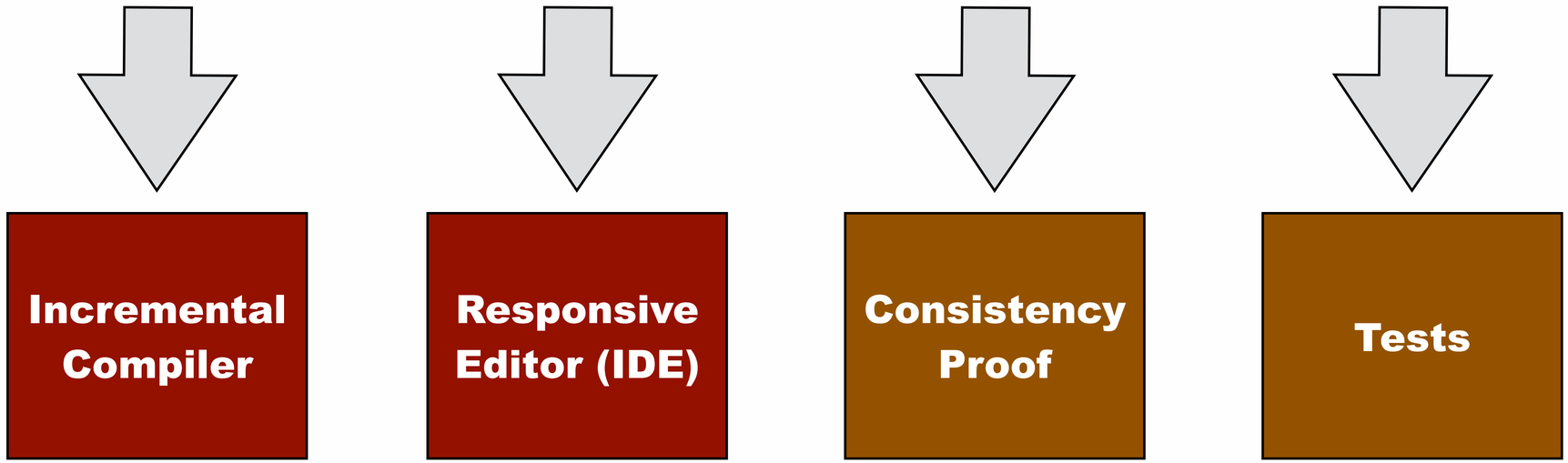
```
templates // statements

  Statement.Return =
    <return <Exp>;>

  Statement.If = <
    if(<Exp>)
      <Statement>
    else
      <Statement>
  >
```

```
binding rules

  Param(t, name) :
    defines Variable name
         of type t

  Var(name) :
    refers to Variable name
```

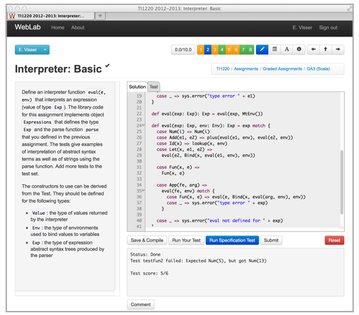
```
type rules

  Call(name, exp*) : t
  where definition of
                name : (t*, t)
    and exp* : t_exp*
    and t_exp* == t*
```

```
rules

  e --> True(), s1 --> v
  ----------------------
  If(e, s1, s2) --> v

  e --> False(), s2 --> v
  -----------------------
  If(e, s1, s2) --> v
```

```
rules

  translate :
    Call(f, e*) -> [
      <translate> e*,
      Callq(f),
      Movl(Reg("r"), Reg("t"))
    ]
```



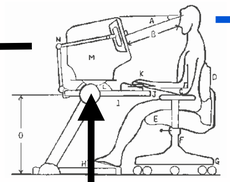
TU Delft



WebLab, researchr,
YellowGrass, ...

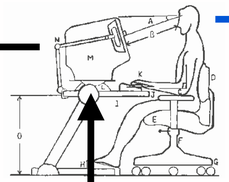
WebDSL, KeyBridge,
Green-Marl, SugarJ

Spoofax, SDF3, Stratego
NaBL, TS, DynSem



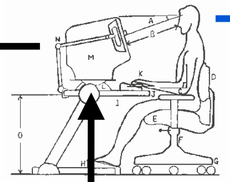
computer
user

software



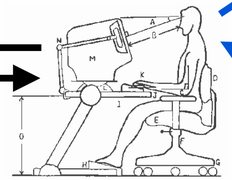
software
engineer

programming
language



language
designer

meta-
language



language
engineer

TU Delft

Education

Software languages are the instruments that allow us to turn computational thinking into computation

A computer science education should prepare its students to co-evolve with the changing language landscape, learn new technologies as they emerge, contribute to their development, and apply them in new contexts

That requires students to develop an understanding of the **fundamentals of computing**, rather than just acquire a set of specific skills

Modernizing Programming Education

Concepts of Programming Languages
- Study modern programming languages
- Functional + OO (Scala)
- Prototype based inheritance (JavaScript)
- *Connections with other disciplines*

Model-Driven Software Development
- Language design project

Compiler Construction
- Using language workbench
- Complete compiler + IDE in one semester
- Connection with research

Extending Curriculum
- *program analysis, formal semantics, verification, ...*

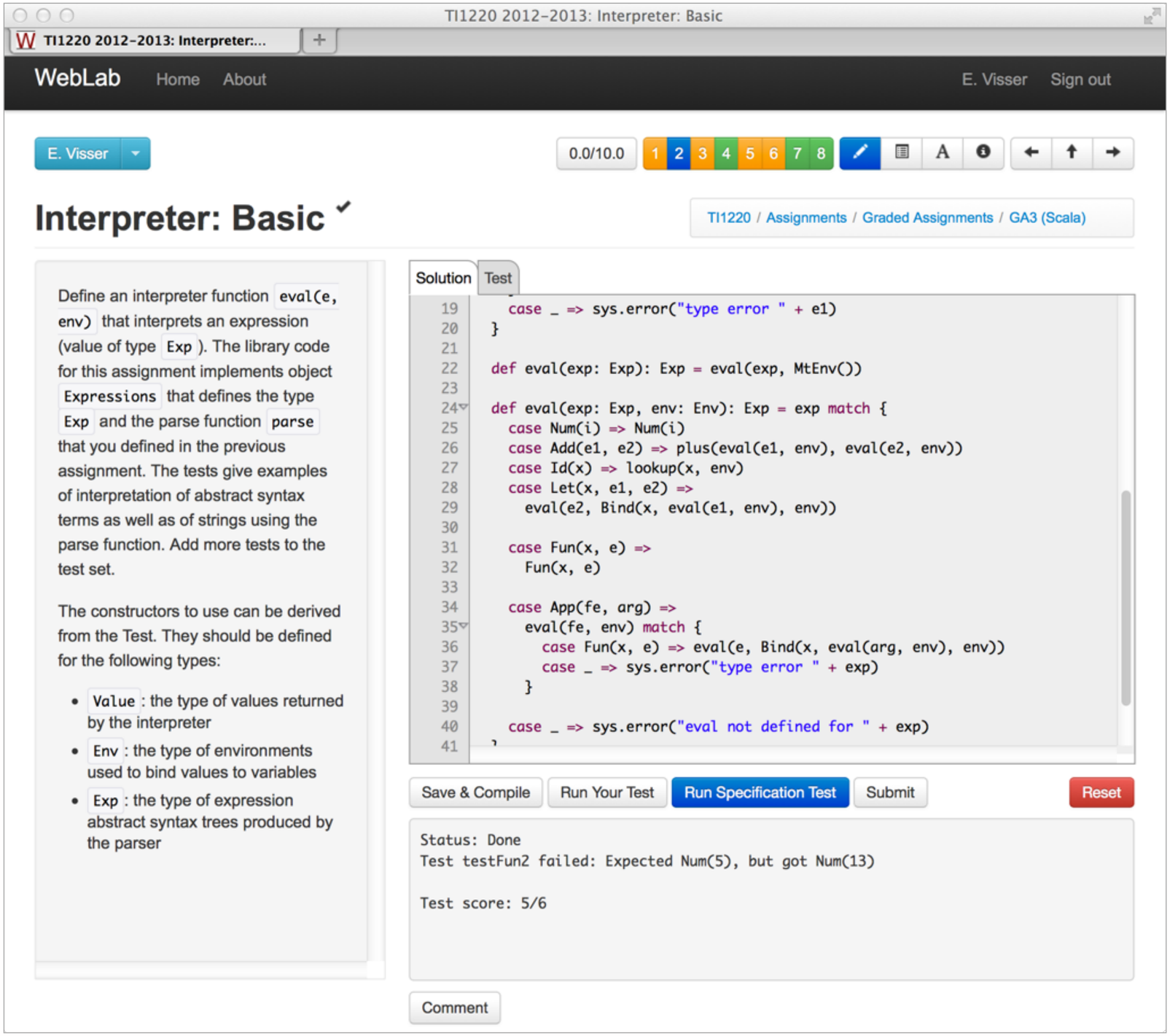
‘ICT in Education’

EvaTool: automate the course evaluation workflow

researchr: bibliography management for literature surveys

WebLab: learning management system for programming education

Improving efficiency and effectiveness of organization



WebLab

Programming education in the browser

Program in browser
Compile on server

Automatic grading with unit testing

Instructor uses same environment

TU Delft

Thanks!

Family

Connie Visser

Wim Visser

Janine Mets

Jelle Visser

Bente Visser

Employers

Board TU Delft

EEMCS Faculty

AvL Committee

Paul Klint (UvA)

Andrew Tolmach (OGI)

Doaitse Swierstra (UU)

Arie van Deursen

Henk Sips

Rob Fastenau

Karel Luyben

Research Group

Stratego/XT
Nix
(1998-2006)

Anya Bagge
Martin Bravenboer
Eelco Dolstra
Merijn de Jonge

Karl Kalleberg
Karina Olmos
Rob Vermaas

Sander Mak
Roy van den Broek
Armijn Hemel
Jory van Zessen
Bogdan Dumitriu
Remko van Beusekom
Rene de Groot
Niels Janssen
Arthur van Dam

Jozef Kruger
Jonne van Wijngaarden
Alan van Dam
Robert Anisko
Lennart Swart
Hedzer Westra
Arne de Bruijn

Spoofax
WebDSL
DisNix
Hydra
(2006-2013)

Sander van der Burg
Eelco Dolstra
Danny Groenewegen
Maartje de Jonge
Zef Hemel

Lennart Kats
Rob Vermaas
Sander Vermolen
Guido Wachsmuth

Richard Vogelij
Oskar van Rest
Chris Gersen
Elmer van Chastelet
Nathan Bruning
Ricky Lindeman
André Miguel Simões Dias Vieira
Tobi Vollebregt
Vlad Vergu

Gabriël Konat
Sverre Rabbelier
Nami Nasserazad
Ruben Verhaaf
Wouter Mouw
Michel Weststrate
Jippe Holwerda
Nicolas Pierron
Jonathan Joubert

Language
Designer's
Workbench
(2013-2018)

Luís Amorim
Elmer van Chastelet
Danny Groenewegen
Gabriël Konat

Pierre Neron
Augusto Passalaqua
Vlad Vergu
Guido Wachsmuth

Mircea Voda
Daco Harkes

TU Delft

Sponsors

Oracle Labs

Philips Research

SERC

Jacquard, Open Competition, VICI

TU Delft

The Future of Programming

Arie van Deursen
Brandon Hill
Daan Leijen
Erik Meijer
Guido Wachsmuth
Harry Buhrman
Herman Geuvers
John Hughes
Manuel Serrano
Markus Püschel
Markus Völter
Sebastian Erdweg
Stefan Hanenberg
Tiark Rompf

Delft University of Technology