TreeCalc
Computations in the Insurance business

Dr. E. Hackhofer
EDV-Softwareberatung Ges.m.b.H.
1070 Wien
http://www.hackhofer.com

Hackhofer

TU WIEN

FAKULTÄT FÜR INFORMATIK
Faculty of Informatics

computer languages

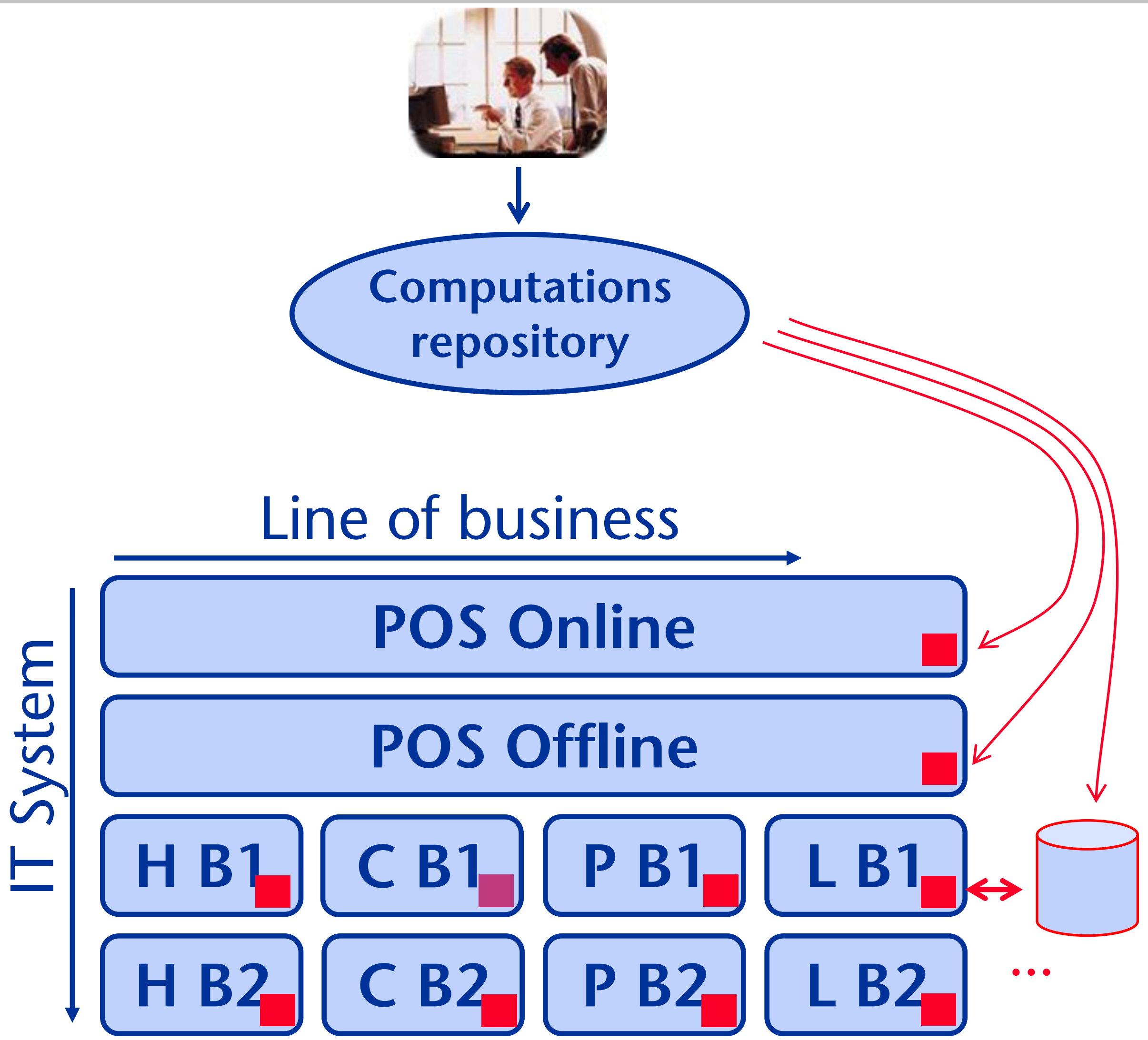
Schedule

- **Motivation + environment**
- **Domain specific language**
- **TreeCalc**
  - Language
  - Compiler
  - Demo

Computations / Business Rules

- **Core**
  - Premium calculation
- **User Interface**
  - Choices (Listbox, ...)
  - Plausibility checks
  - UI control
- **Others**
  - Print data
  - Interface data
  - Compensation calculation

IT architecture



Computations repository

Line of business

IT System

POS Online

POS Offline

H B1

C B1

P B1

L B1

H B2

C B2

P B2

L B2

...

IT environment

- **Heterogenous platforms**
  - Windows, Linux
    - 32bit, 64bit
    - Java, .NET, VB, VBA, ...
  - z/OS
    - COBOL, PL/I
    - IMS, CICS, Batch
  - AIX
- **Performance**
  - Offline: Laptops
  - Online: ~1000 computations / min
  - Batch-jobs: time+cost critical

Implementation options

- **Hardcoded**
- **Database driven**
- **Customization**
  - Code (C, Java, ..)
  - Simple language
- **Domain specific language (DSL)**
  - Declarative
  - Domain experts handle the rules
  - Write once, use/call everywhere
  - Uniform interface to IT; communication!

DSL - Examples

- **select * from customer order by name**
- **td{border:1px solid gray; padding:3px; }**
- **calc: calc.c**
  $(CC) $(CFLAGS) -O3 -o $@ $<
- **/^Record/ { counter++ }**
  END { print “nr. of records: “ counter }

DSL - Implementation Options

- **Internal DSL / Embedded DSL**
  - Lisp, Ruby, (Template) Haskell, (Meta)OCaml
  - Fluent interface: Java, C#, ...
  - Highly dependent on host language
- **External DSL**
  - Custom syntax, custom parsing
  - Semantic model
  - Interpretation / Code generation

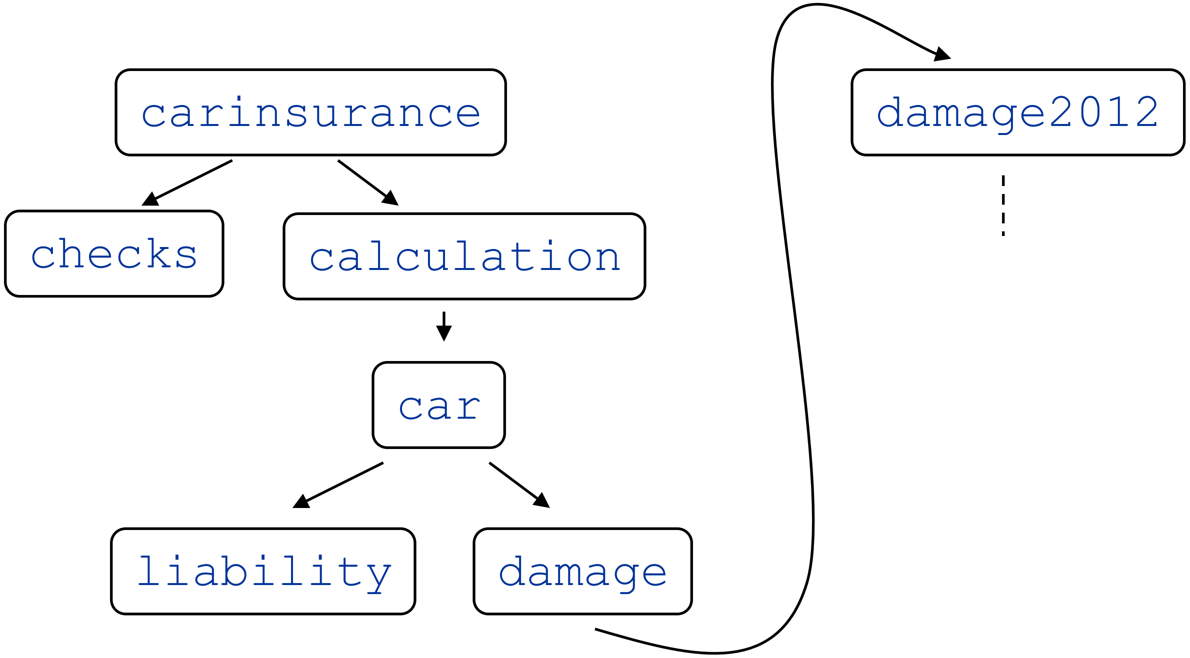
```
cust.add(“Martin“)
    .born(1981)....
```

DSL „TreeCalc“

- Text format
- Calculations organized in Trees / DAGs
- Declarative, no side-effects
- Data is part of the „model“

TreeCalc - Tree

```
TREE carinsurance {
    NODE checks ;
    NODE calculation {
         NODE car TIMES I_CarCounter {
            NODE liability ;
            NODE damage IF I_Damage_YN {
                LINK damage2012;
            }
         }
    }
}
```



TreeCalc - Calculations

```
CALC carinsurance.calculation {
    RX_Prem = R_Prem
                *
              IF I_Discount_YN THEN
                   0.8
              ELSE
                   1
              ENDIF ;
}

CALC carinsurance.calculation.car.liability {
    R_Prem = I_kw * T_Area[I_Area].fact ;
    ...
}

CALC damage2012.calculation {
    R_Prem = ...
}
```

TreeCalc - Table

```
TABLE T_Mortality (age, qx, qy) {
   16, 0.0006380, 0.0003980 ;
   17, 0.0007200, 0.0004160 ;
   18, 0.0007760, 0.0004060 ;
   19, 0.0008060, 0.0003720 ;
   20, 0.0008400, 0.0003580 ;
    ...
}

TABLE T_Liability_Sum (key, text) {
   1, "€  6.000.000,-" ;
   2, "€ 12.000.000,-" ;
}
```

TreeCalc - Function

```
FUNC F_LI_Lx(age, sex, risk) =
   IF age <= 0 THEN
      100000
   ELSE
      F_LI_Lx(age - 1, sex, risk)
      *
      (1 - F_LI_qx(age - 1, sex, risk))
   ENDIF
 ;

FUNC F_LI_qx(age, sex, riskq) =
     sex = 1
     ? min(T_Mortality[age].qx * (1 + riskq), 1)
     : min(T_Mortality[age].qy * (1 + riskq), 1)
;
```

TreeCalc Prototype

- jflex + BYacc
- Helper methods to construct AST

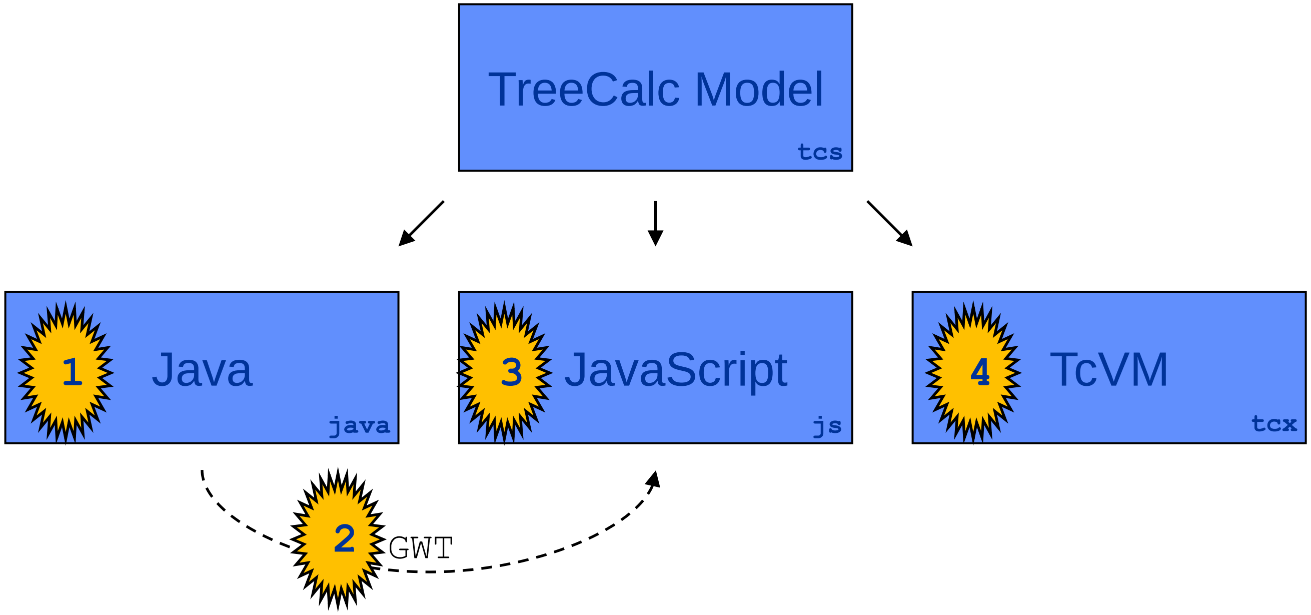
```
tabrows:
  tabrow           { $$ = getAstTableRows($1); }
| tabrows tabrow { $$ = getAstTableRows($1, $2); }
```

- Irregular Heterogeneous AST

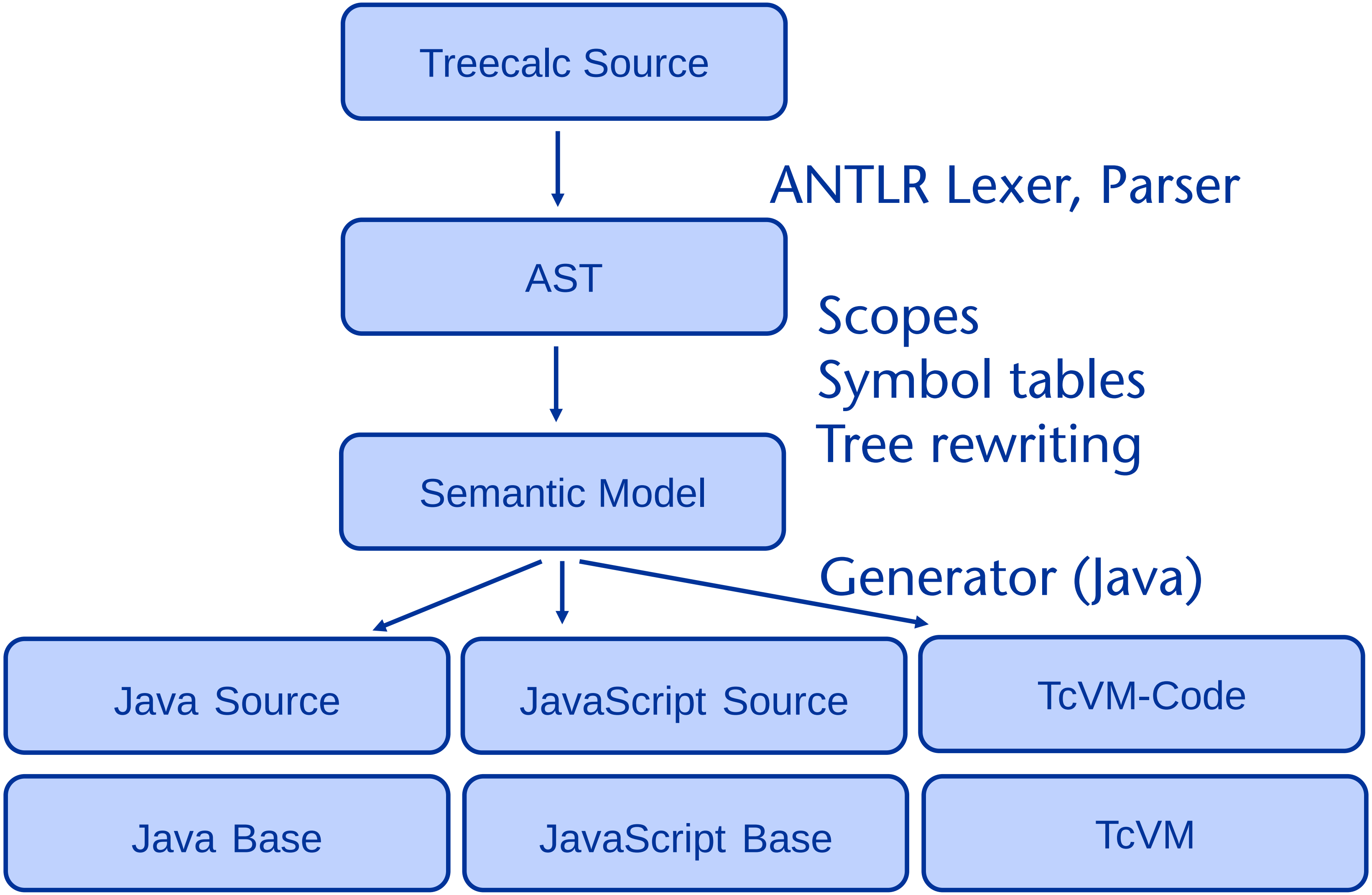
```
class AstBinop extends Ast {
      Ast left;
      Ast right;
```

- Interpretation of AST
  - Binary format: Java serialization

Generation targets



TreeCalc Implementation



TreeCalc - Parsing

- **ANTLR**
  - Lexer + Parser + Tree construction
  - LL(*), semantic/syntactic predicates
  - DFA to scan ahead+decide
- **Homogeneous AST**

```
class Tree {
   List<Tree> children;
   int getType();
   String getText();
```

TreeCalc ANTLR Snipplet

```
compilationunit: def+ -> ^(TT_COMPUNIT def*) ;
def:
   'TREE' nodepath '{' nodeinfo* '}'
      -> ^('TREE' nodepath nodeinfo*)
 | 'CALC' nodepath '{' resultdef* '}'
      -> ^('CALC'  nodepath resultdef*)
 | 'INPUT' id (('{' resultdef* '}') | ';')
      -> ^('INPUT' id resultdef*)
 | 'FUNC'^ resultdef
 | 'TABLE'^ id '('! colnames ')'! '{'! tabline* '}'!
 ;
tableline: tablecell (',' tablecell)* ';'
      -> ^(TT_TABLELINE tablecell*)
 ;

NUMBER : NUMBER_INT
       | NUMBER_INT '.' NUMBER_INT EXPONENT? ;
fragment NUMBER_INT: '0'..'9'+;
```

lex+yacc vs. ANTLR

- **ANTLR**
  - Nicer grammar (parsing expr. grammar)
  - Automatic error recovery
  - Declarative tree construction
  - ANTLRWorks, Eclipse plugin, used by XText, …
  - Java framework: trees, …
- **lex+yacc**
  - Smaller (factor 5) and faster
  - lex better than ANTLR lexer
  - expressions: assoc. & precedance nice

TreeCalc – Code generation

- **out.print(...) ☺**
  - Alternative: e.g. StringTemplate
- **Simple because of Semantic model + AST**
- **Formulas**
  - intermediate vars _1, _2, ...
  - AST node

    → optional: out.print(...)

    → returns expression string (short expr. or varname)

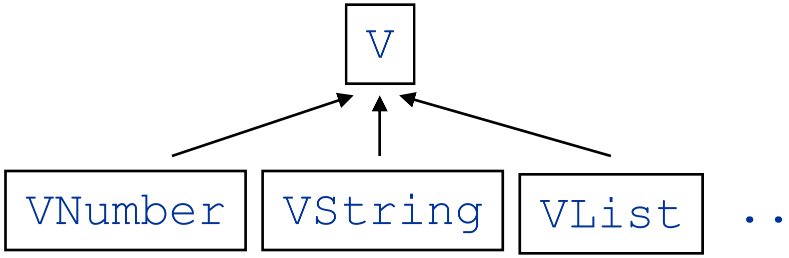
TreeCalc – Java formula

```
static final V F_LI_LX(S _s, V age, V sex, V risk) {
   Object cacheKey = _s.getCacheKey(8156345, age, sex, risk);
   V ret = _s.readCache(cacheKey);
   if (ret!=null) { return ret; }
   V _1;
   V _2 = age.smleq(_i0);
   if (_2.booleanValue()) {
      _1 = _i100000;
   } else {
      V _3 = age.sub(_i1);
      V _4 = age.sub(_i1);
      V _5 = _i1.sub(F.F_LI_QX(_s, _4, sex, risk));
      V _6 = F.F_LI_LX(_s, _3, sex, risk).mult(_5);
      _1 = _6;
   }
   ret = _1;
   _s.writeCache(cacheKey, ret);
   return ret;
}
```

```
FUNC F_LI_Lx(age, sex, risk) =
   IF age <= 0 THEN
      100000
   ELSE
      F_LI_Lx(age-1, sex, risk)
      *
      (1-F_LI_qx(age-1, sex, risk))
   ENDIF
 ;
```

TreeCalc - Data types

- **Dynamic conversions + checks**
- **Data Types**
  - String
  - Number
  - List
  - Date: String (Y-M-D, D.M.Y, M/D/Y)
  - Boolean: Number (false=0, true=1)
  - Internal: Function ref, Table ref, Null



TreeCalc - Java highlights

- **Generated Java code quite nice**
- **Nasty bytecode limits**
- **Performance**
  - big switch faster than reflection
  - Dynamic type conversions + a lot of objects
  - LRU Cache instead of HashMap
  - static analysis to exclude simple formulas from caching

GWT (Java to JavaScript)

- **Changes**
  - Missing libraries (Regex, NumberFormat, …)
  - Optimized implementations (e.g. BitSet)
- **Conclusions**
  - JavaScript almost „for free“
  - Quite big JavaScript (base libraries etc.)
  - „Pure“ JavaScript preferred

JavaScript

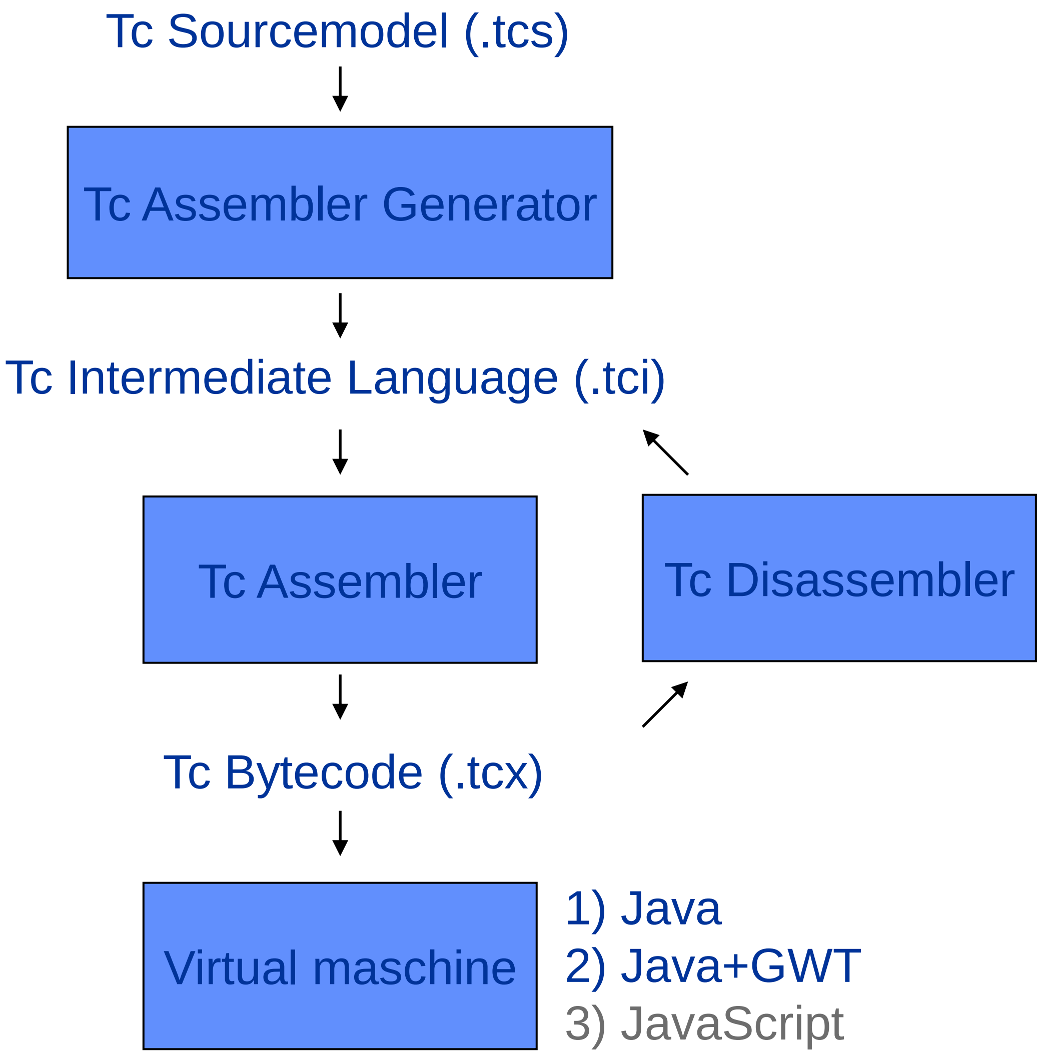
- **… the easy part**
  - dynamic constructs → no switch() needed
  - Base functions
    - Number and String enhanced
- **… the hard part**
  - no HashMap, … → Strings for property access
  - no NumberFormat etc → additional implementation

JavaScript - Example

```
tc.f = {
 F_LI_LX: function(_s, age, sex, risk) {
  var cacheKey = _s.getCacheKey(1564575033, age, sex, risk);
  var ret = _s.readCache(cacheKey);
  if(ret!=undefined) { return ret; }
  var _1;
  var _2 = age.smleq(tc.c._i0);
  if (_2) {
    _1 = tc.c._i100000;
  } else {
    var _3 = age.subtract(tc.c._i1);
    var _4 = age.subtract(tc.c._i1);
    var _5 = tc.c._i1.subtract(tc.f.F_LI_QX(_s, _4, sex, risk));
    var _6 = tc.f.F_LI_LX(_s, _3, sex, risk).mult(_5);
    _1 = _6;
  }
  ret = _1;
  _s.writeCache(cacheKey, ret);
  return ret;
 }, ... }
```

```
FUNC F_LI_Lx(age, sex, risk) =
   IF age <= 0 THEN
      100000
   ELSE
      F_LI_Lx(age-1, sex, risk)
      *
      (1-F_LI_qx(age-1, sex, risk))
   ENDIF
 ;
```

TreeCalc - Virtual machine



Tc Sourcemodel (.tcs)

Tc Assembler Generator

Tc Intermediate Language (.tci)

Tc Assembler

Tc Disassembler

Tc Bytecode (.tcx)

Virtual maschine

1) Java
2) Java+GWT
3) JavaScript

Tc Intermediate Language

```
F_LI_Layers =
  F_LI_Indexation_Perc > 0 ? F_LI_TariffDuration : 1
```

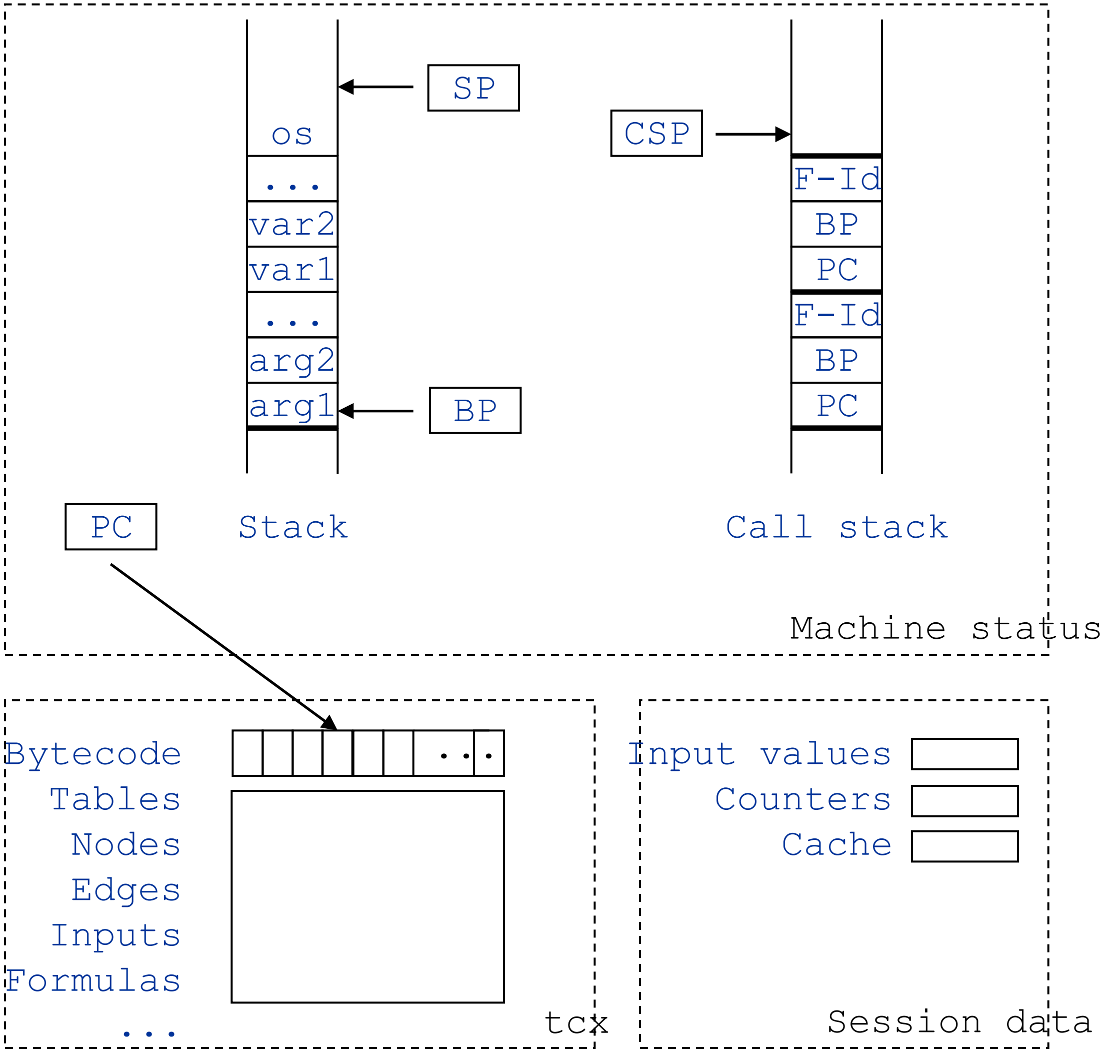
tcs

Tc Assembler Generator

```
.formula formula=506 simple=false ; line 3182
   //start of if statement, line 3182
   : callfunc 65 0 ; F_LI_INDEXATION_PERC
   : pushconst 0
   : cmpbig
   : iffalse L0
   : callfunc 90 0 ; F_LI_TARIFFDURATION
   : goto L1
 L0:
   : pushconst 1
 L1:
   //end of if statement
   : return
.formuladone
```

tci

Tc virtual machine - Data



TcVM - Highlights

- **TcVM Java**
  - Rapid development
  - Experiments with adaptive memoization
  - Base classes reused
  - Interpreter in Java awkward + slow
- **TcVM JavaScript**
  - Smaller+faster than by GWT

Summary

- External DSL implementation not that hard
- JavaScript getting better + faster
- Virtual Machine implementation very compact

References

- Domain Specific Languages, Martin Fowler, Addison-Wesley, 2010
- Language Implementation Patterns, Terence Parr, Pragmatic Bookshelf, 2009
- The Definitive ANTLR Reference, Terence Parr, Pragmatic Bookshelf, 2007
- http://www.antlr.org/
- http://jflex.de/
- http://byaccj.sourceforge.net/

- stefan.neubauer@hackhofer.at