Kompilatorteknik

Görel Hedin

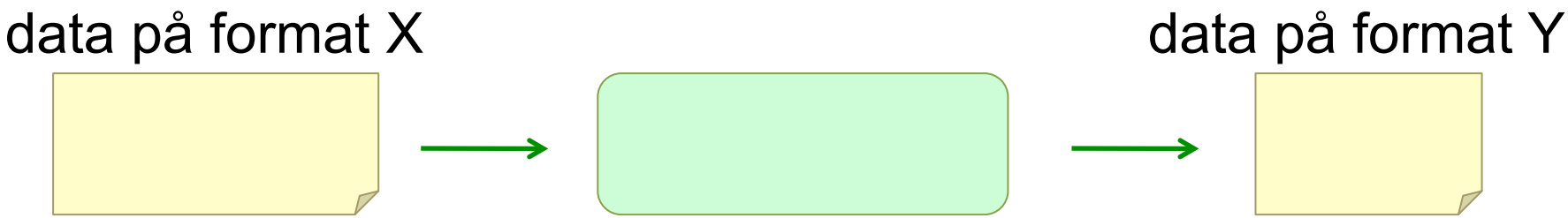
Datavetenskap

Lunds Tekniska Högskola

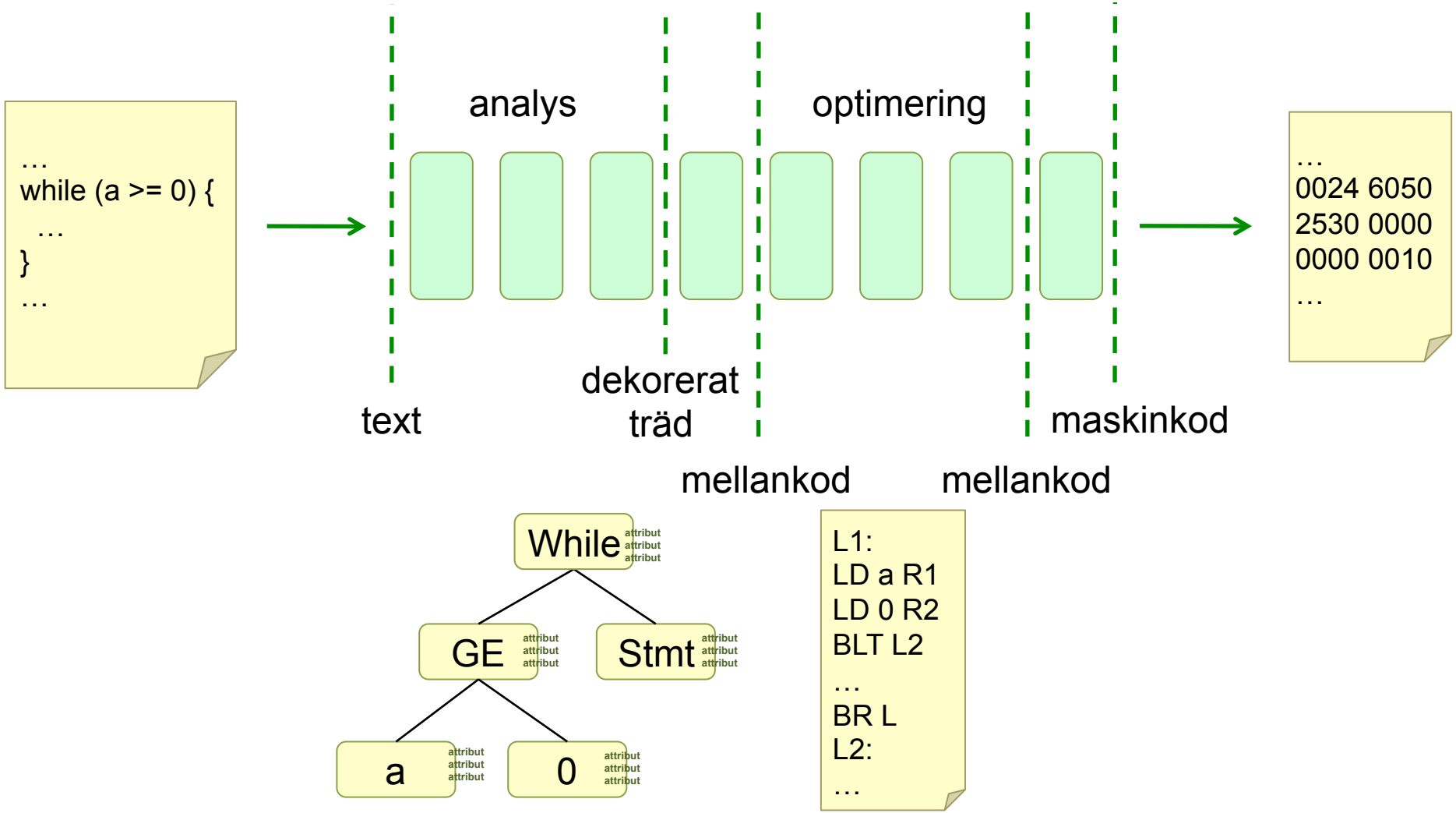
Temaföreläsning, Datorer i system, 2014

En typisk kompilator

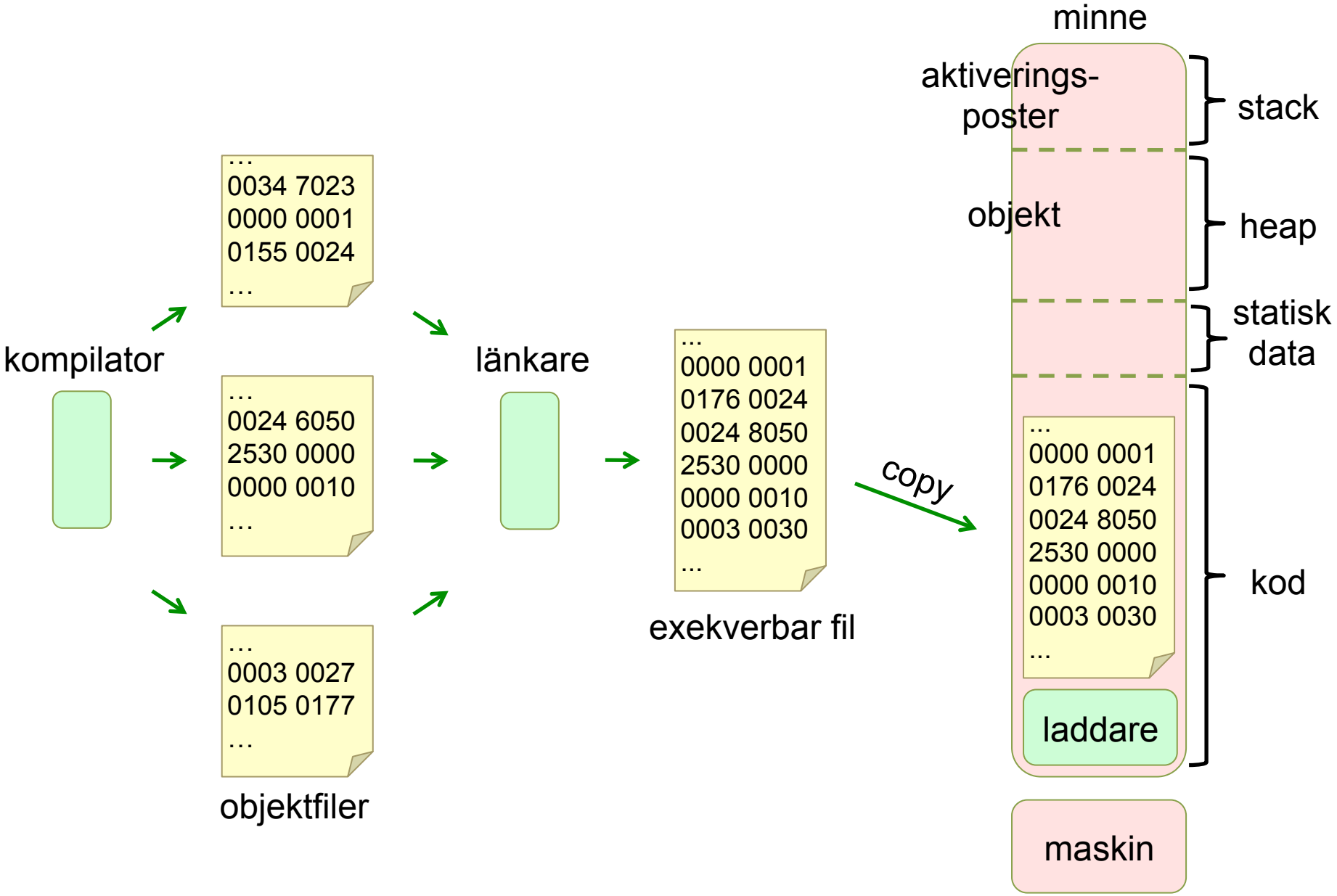
Men tekniken är generell:



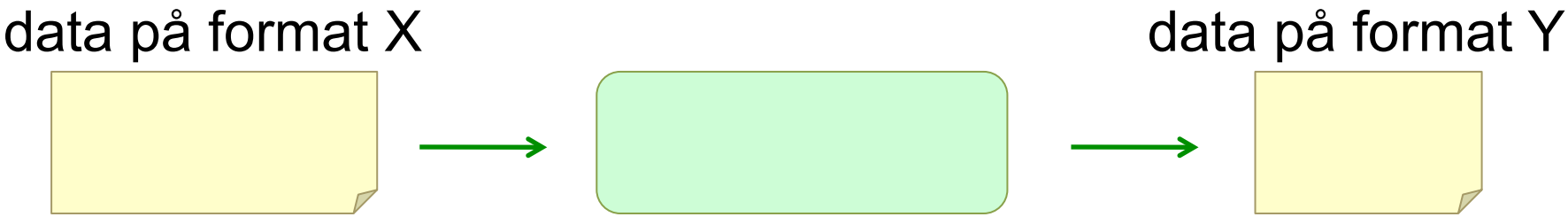
Typiska steg i en kompilator



Länkning, laddning, exekvering



Varför lära sig kompilatorteknik?



- Förstå programspråk
- Förstå programmeringsverktyg
- Förstå prestanda hos program (performance)
- Bygg ditt eget språk/verktyg
  - Mycket vanligare än du tror!!!
  - Många *domänspecifika språk* (domain specific languages)
- Generella användbara tekniker (inte bara för kompilatorer)

Gissa språket! 1958

Lisp

```
(defun reverse(L)
   (if (eq L ())
       ()
       (append (reverse (cdr L)) (list (car L)))))
```

- John McCarthy
- Första funktions-orienterade språket
- Rekursion: dynamisk stack-allokering
- Dynamiskt allokerade listor: heap-allokering och garbage collection
- Metaprogrammering: program manipulerar program

1960

Algol 60

```
procedure RSOK(y, start, slut, z, k);
value start, slut, z;
integer array y;
integer start, slut, z, k;
comment Värdet z söks upp i den sorterade vektorn y med hjälp
av intervall-halvering. Index för z returneras i variabeln k.
begin
   k := (start + slut) // 2;
   if k <> start then
   begin
      if z < y[k] then RSOK(y, start, k, s, k)
                  else RSOK(y, k, slut, z, k)
   end
end;
```

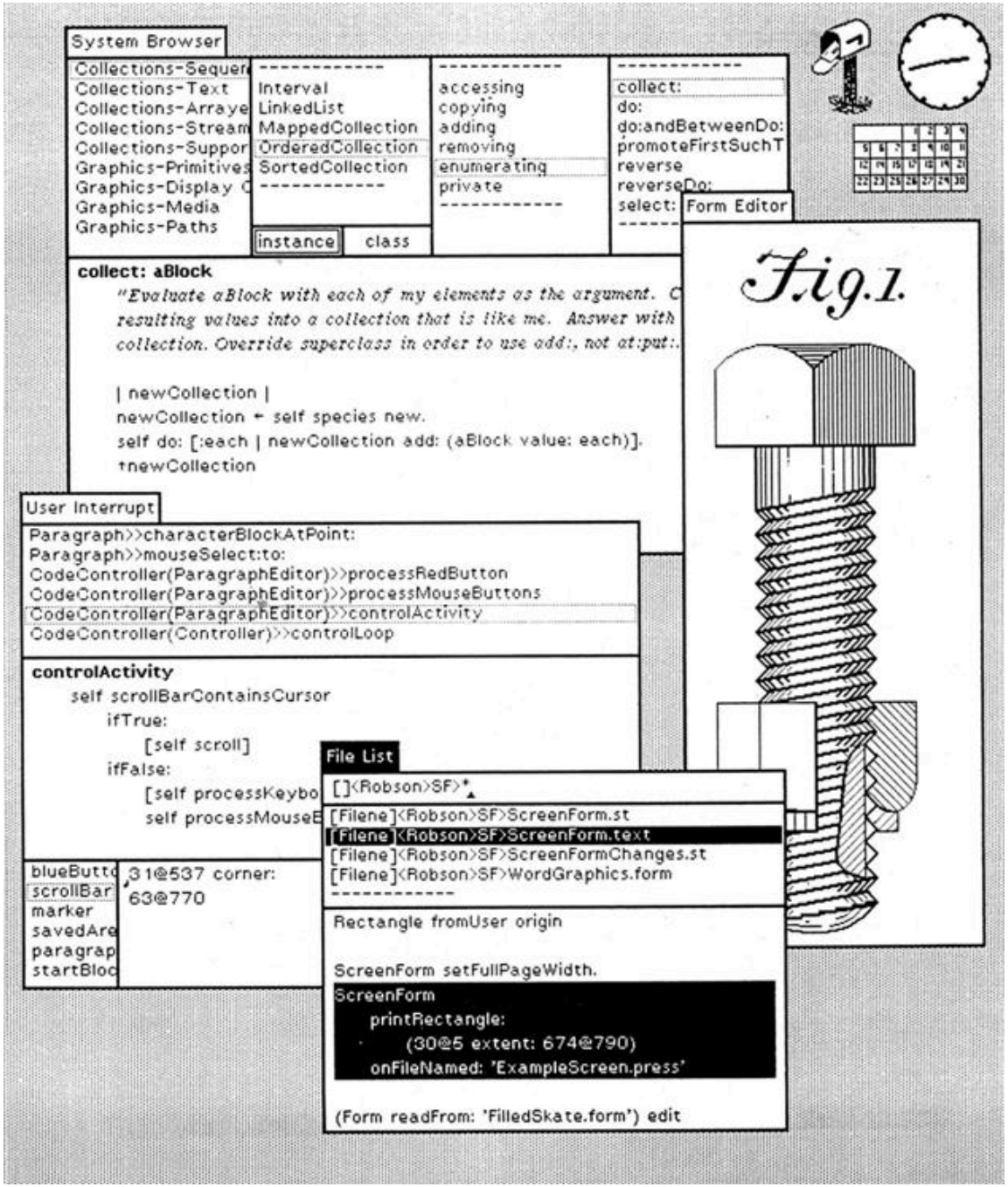
- John Backus, Peter Naur, m. fl.
- Nästlade funktioner, blockstruktur
- Rekursion: dynamisk stack-allokering.
- En Algol-kompilator för SMIL (Siffermaskinen i Lund) implementerades av univ. lektor Torgil Ekman, LTH, 1962.

```
class Glyph;
   virtual: procedure print;
begin
end;

Glyph class Char (c);
   character c;
begin
   procedure print;
     OutChar(c);
end;

Glyph class Line (elements);
   ref (Glyph) array elements;
begin
   procedure print;
   begin
      integer i;
      for i := 1 step 1 until UpperBound(elements, 1) do
         elements(i).print;
      OutImage;
   end;
end;
```

1967

Simula 67

- Kristen Nygaard och Ole Johan Dahl
- Första objekt-orienterade språket
- Klasser, ärvning, virtuella procedurer, co-rutiner: dynamisk heap-allokering och garbage collection.
- En Simula-kompilator implementerades vid LTH cirka 1980.

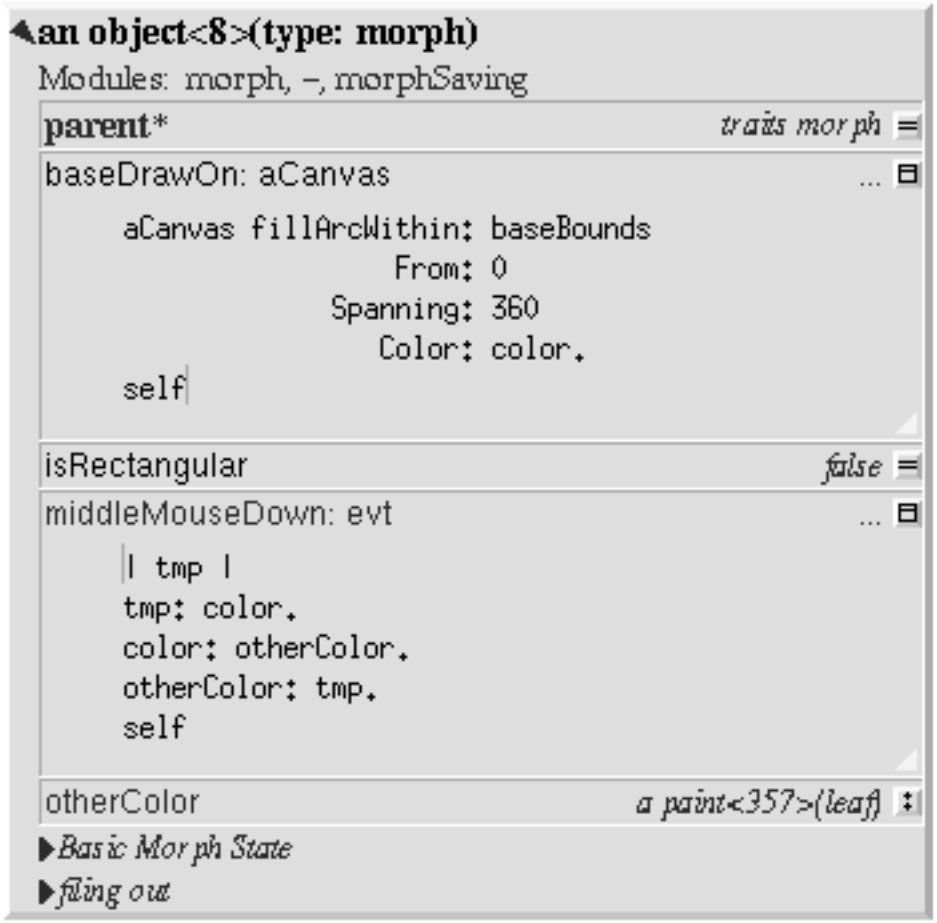


1972

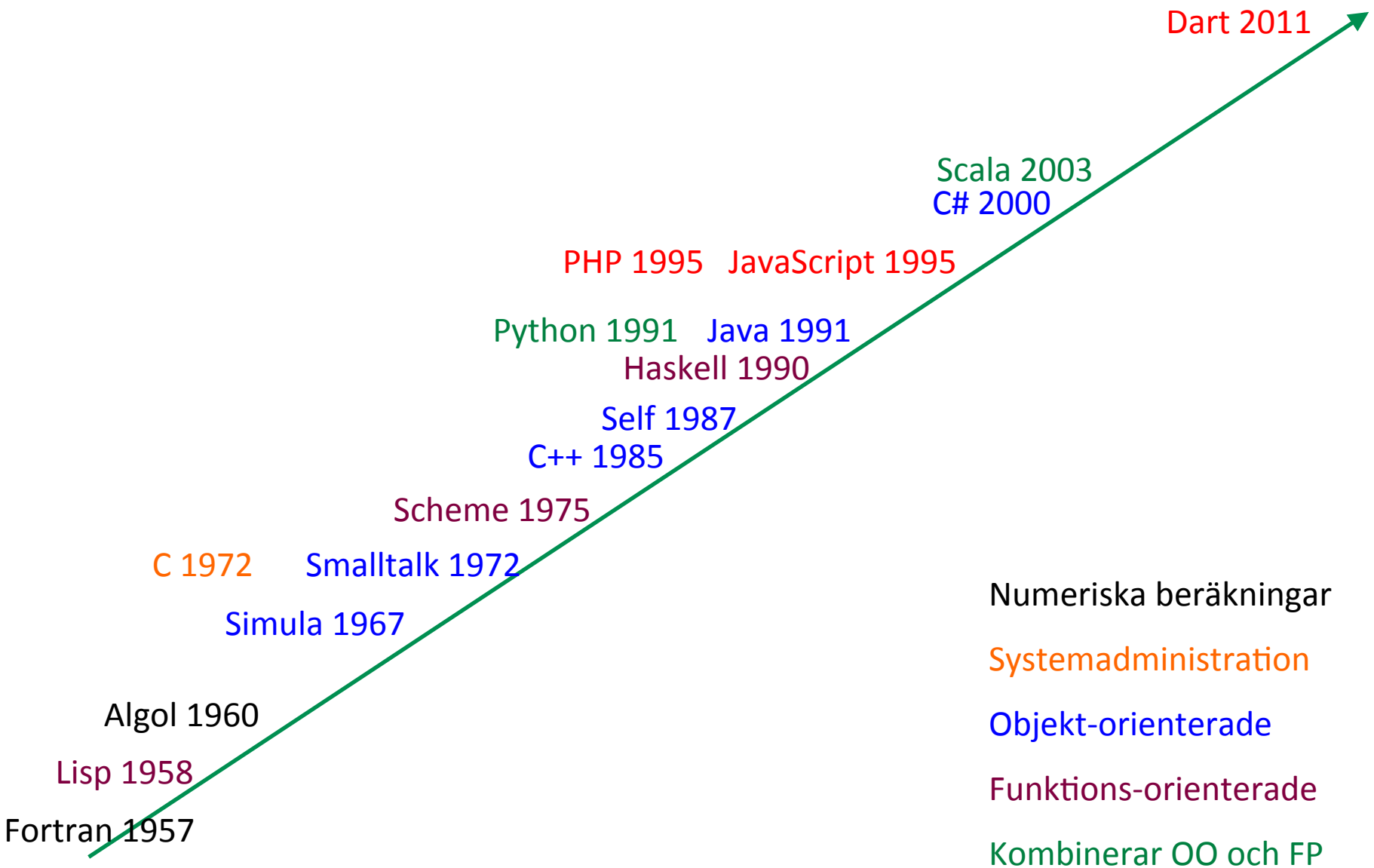
Smalltalk

- Smalltalk, av Alan Kay m. fl.
- Influenser från Lisp och Simula
- *"Everything is an object"*
- Dynamiskt typat
- Reflection
- Bytecode
- JIT: Just-in-time compilation (kompilera under runtime).

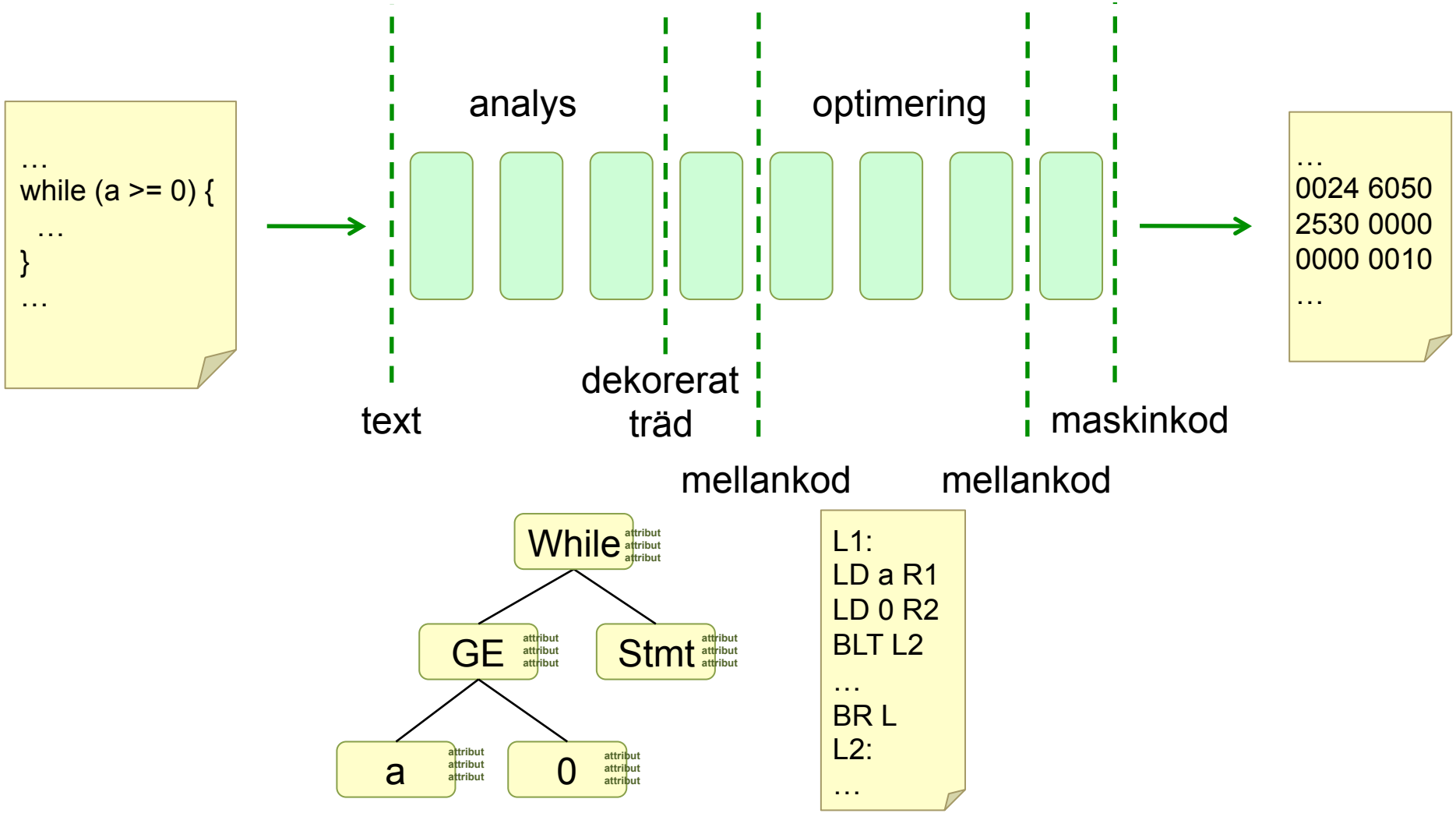
1987
Self



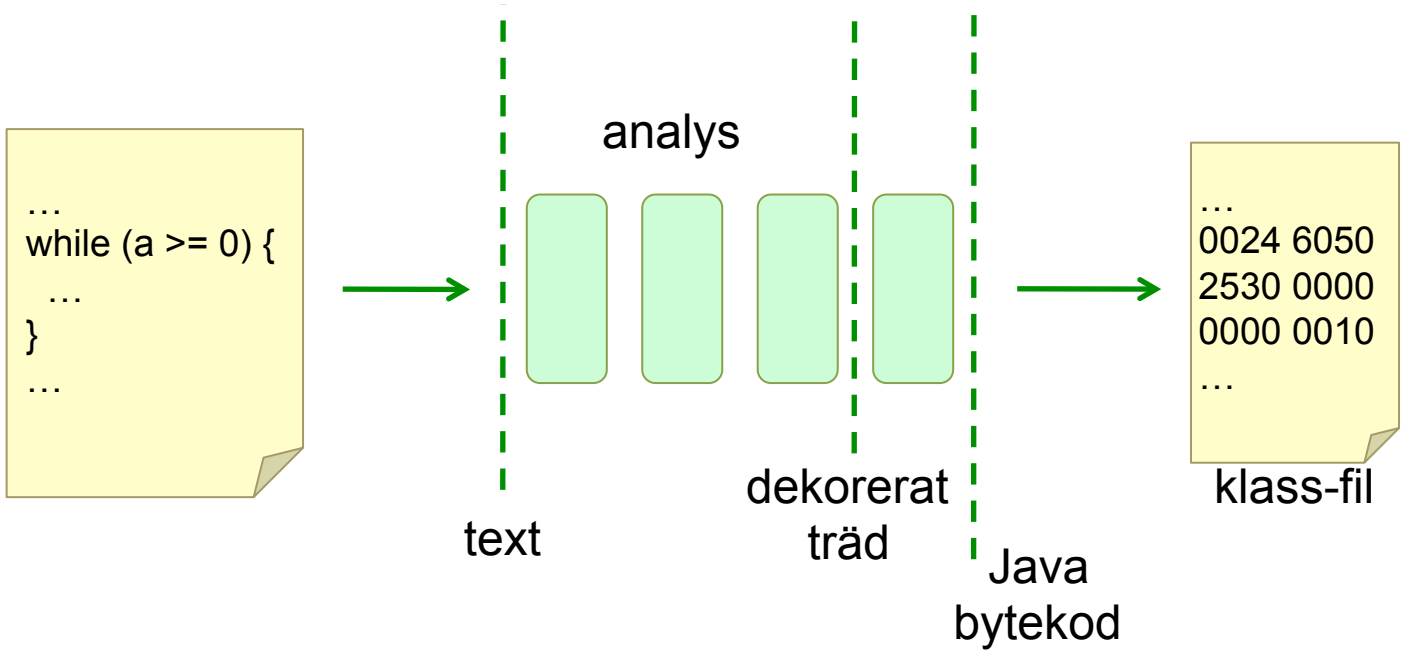
- David Ungar m. fl.
- Influenser från Smalltalk
- Prototyper i stället för klasser: klona objekt i stället för att göra "new"
- Adaptiv optimering: JIT + optimera under runtime.
- Delar av Self-teamet implementerade HotSpot JVM 1999. (Javas virtuella maskin med adaptiv optimering)
- Hotspot JVM är snabbare än C/C++ för vissa program.

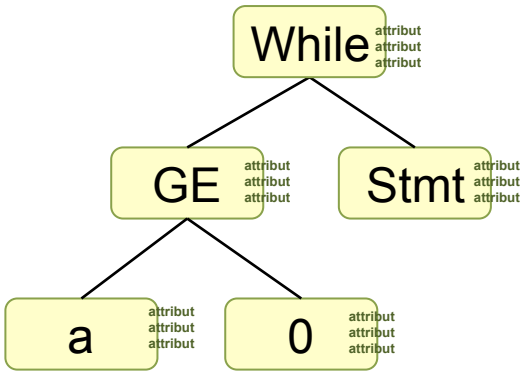

Dart 2011
Scala 2003
C# 2000
PHP 1995
JavaScript 1995
Python 1991
Java 1991
Haskell 1990
Self 1987
C++ 1985
Scheme 1975
C 1972
Smalltalk 1972
Simula 1967
Algol 1960
Lisp 1958
Fortran 1957
Numeriska beräkningar
Systemadministration
Objekt-orienterade
Funktions-orienterade
Kombinerar OO och FP
Webb-programmering

Typiska steg i en kompilator



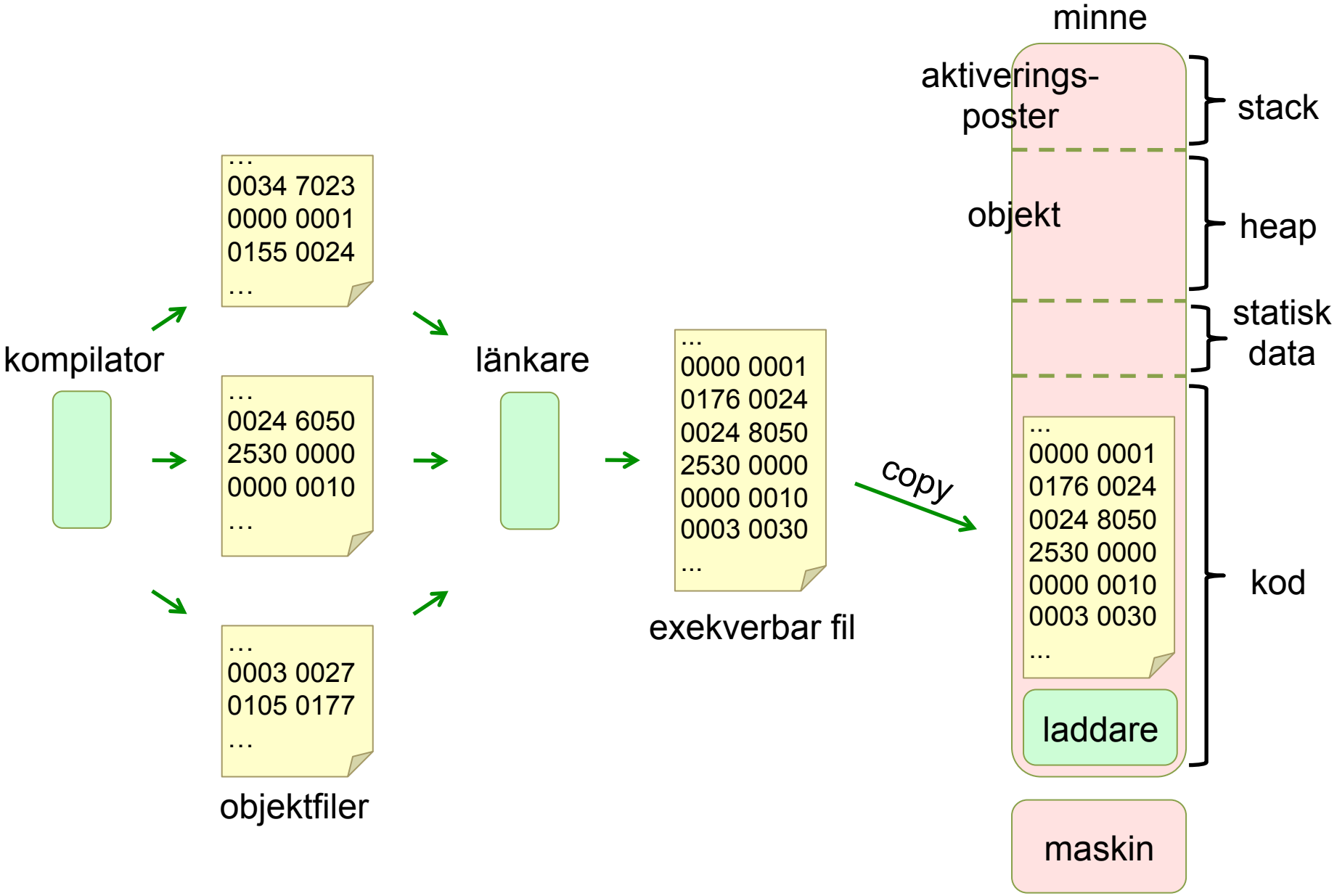
Java-kompilatorn



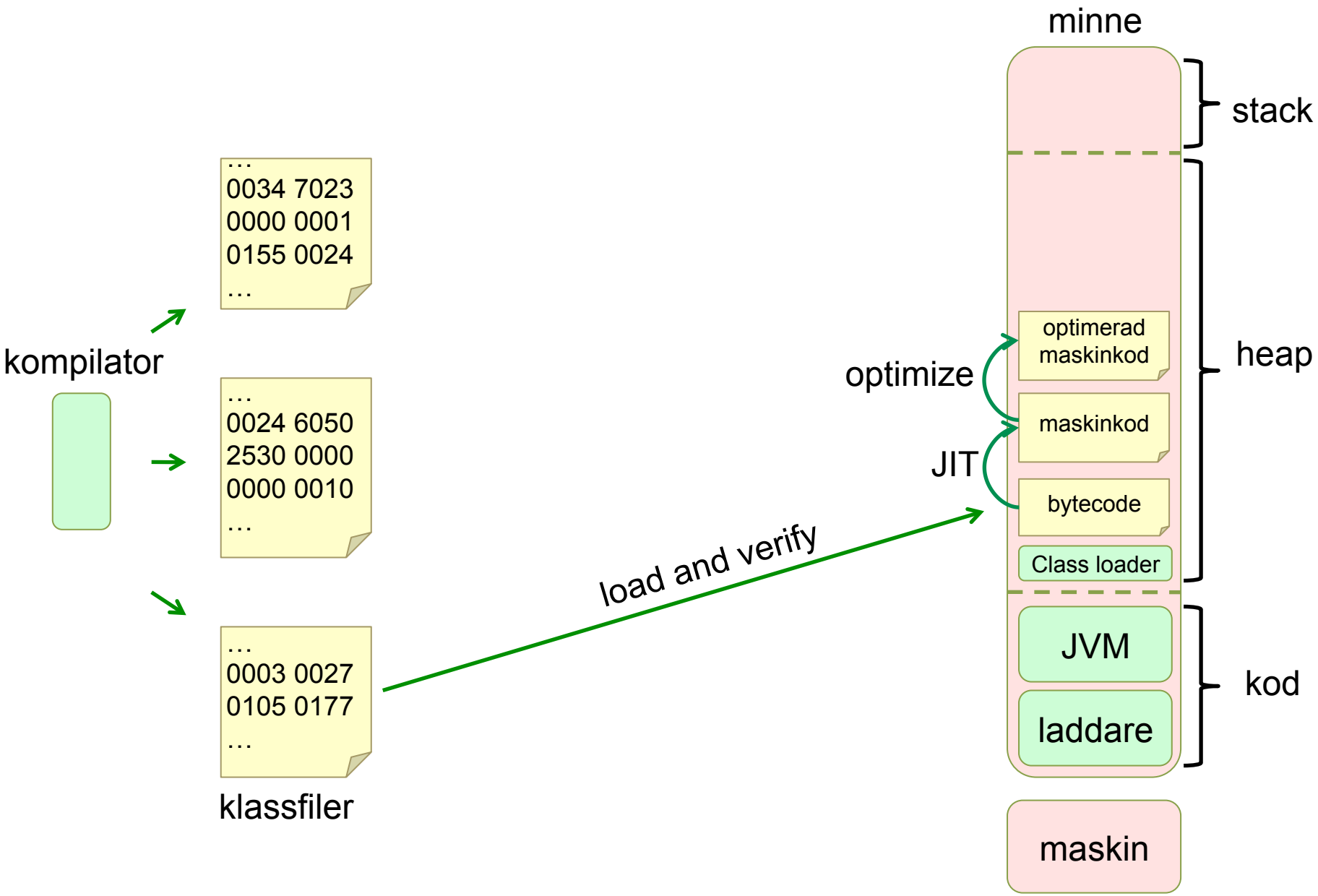


- Klassfiler: binärkod för en *virtuell maskin*
- Mellankodsnivå, ej optimerade

Länkning, laddning, exekvering



Java-exekvering



Implementation av programspråk

Språk	Välkänd kompilator	Implementationsspråk
C	gcc	C
Java	javac	Java
Scala	scalac	Scala
...		

Bootstrapping!

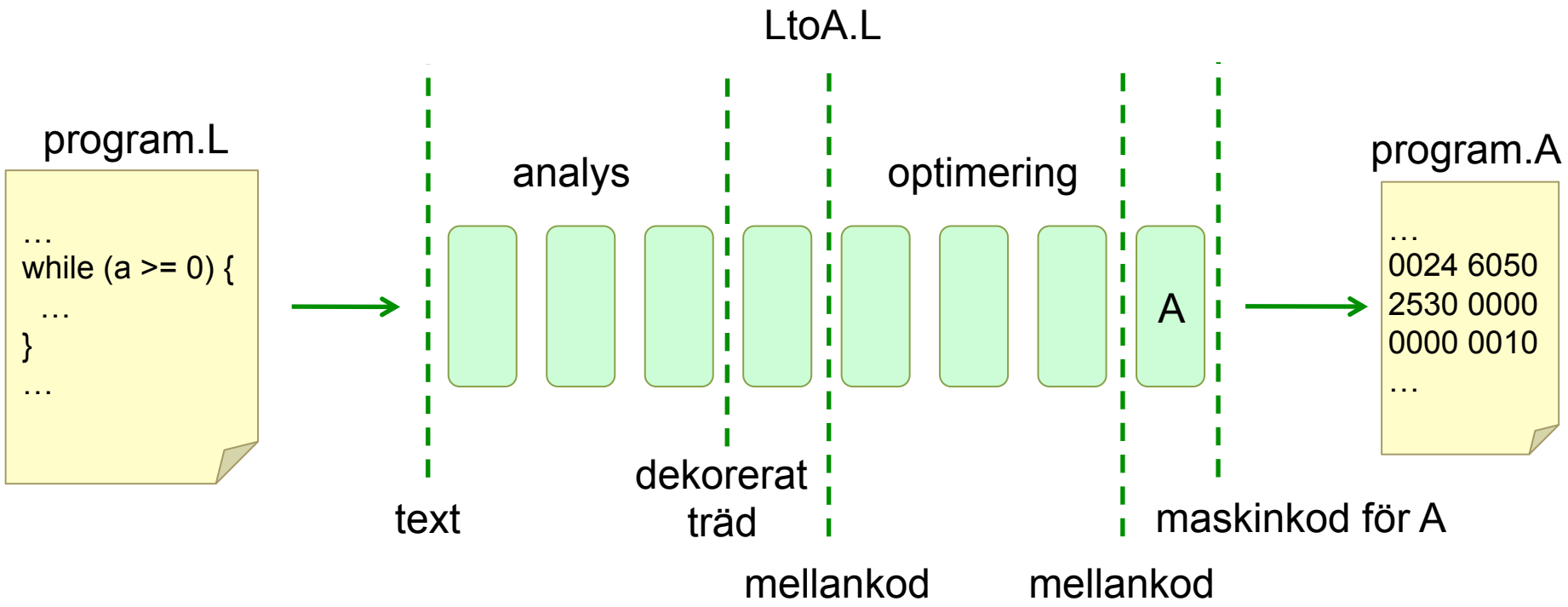
Bootstrapping

Språkversion	Implementeras i
C_1	Assembler
C_2	C_1
C_3	C_2
...	

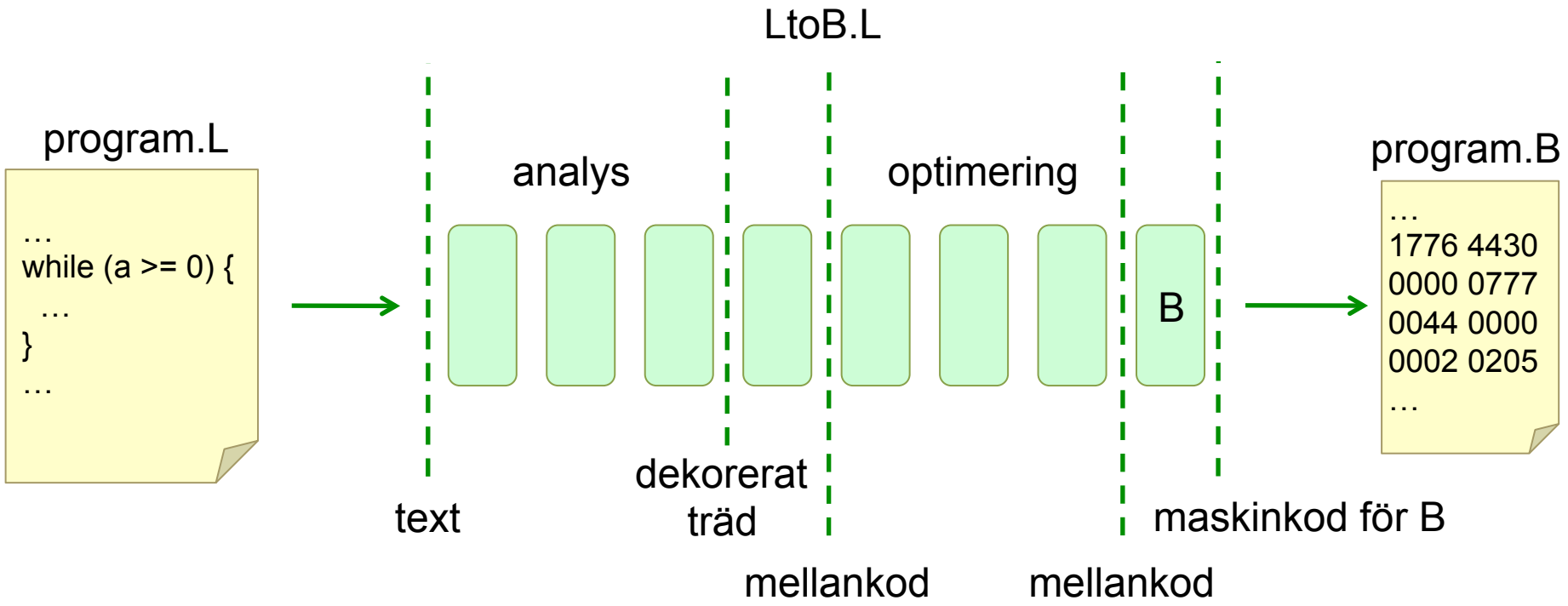
Språkversion	Implementeras i
$Java_1$	C
$Java_2$	$Java_1$
$Java_3$	$Java_2$
...	

Språkversion	Implementeras i
Pizza	Java
$Scala_1$	Pizza
$Scala_2$	$Scala_1$
...	

Kompilera till ny plattform



Kompilera till ny plattform



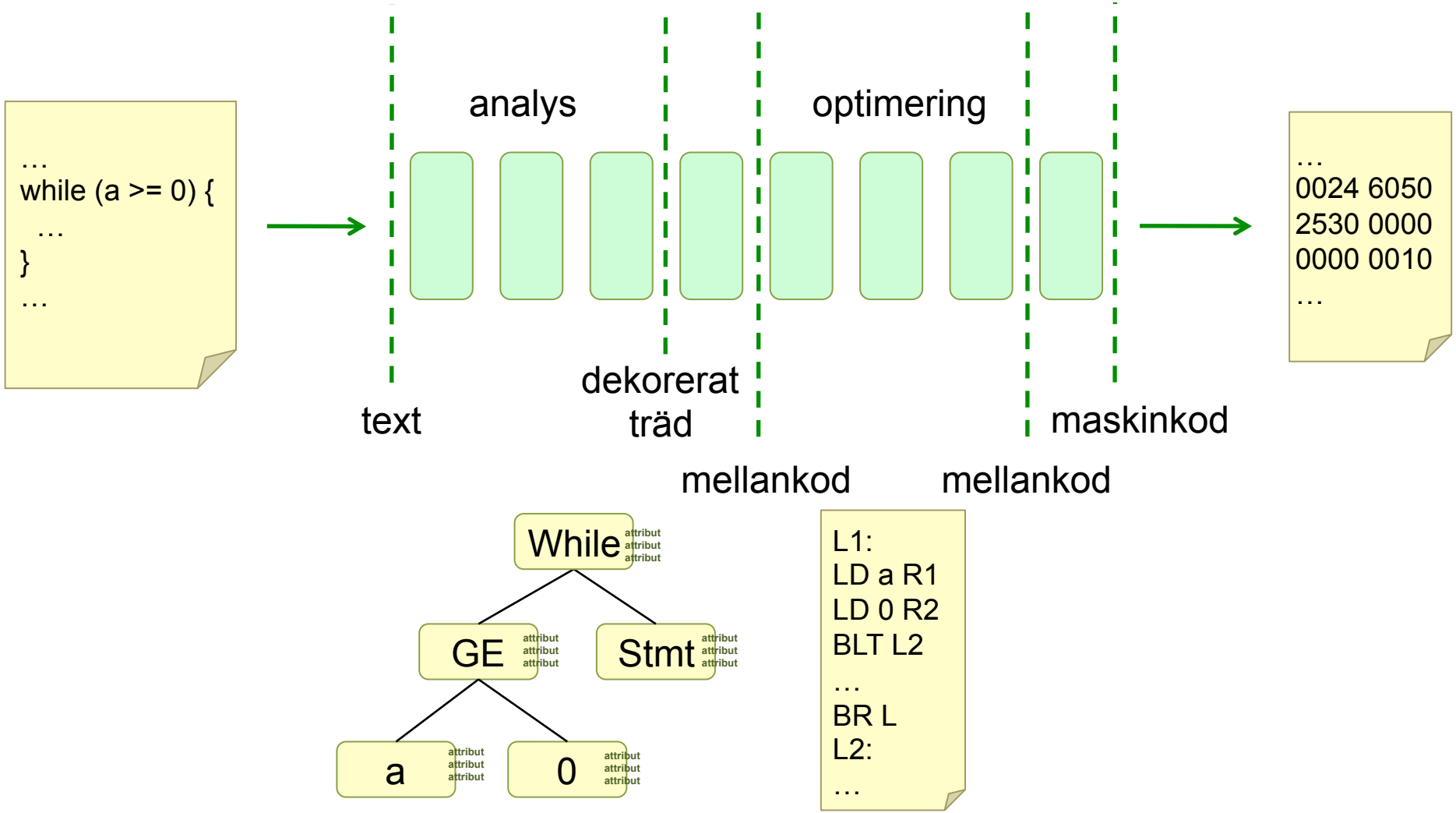
Endast sista steget (backend) behöver bytas

Korskompilering till ny plattform

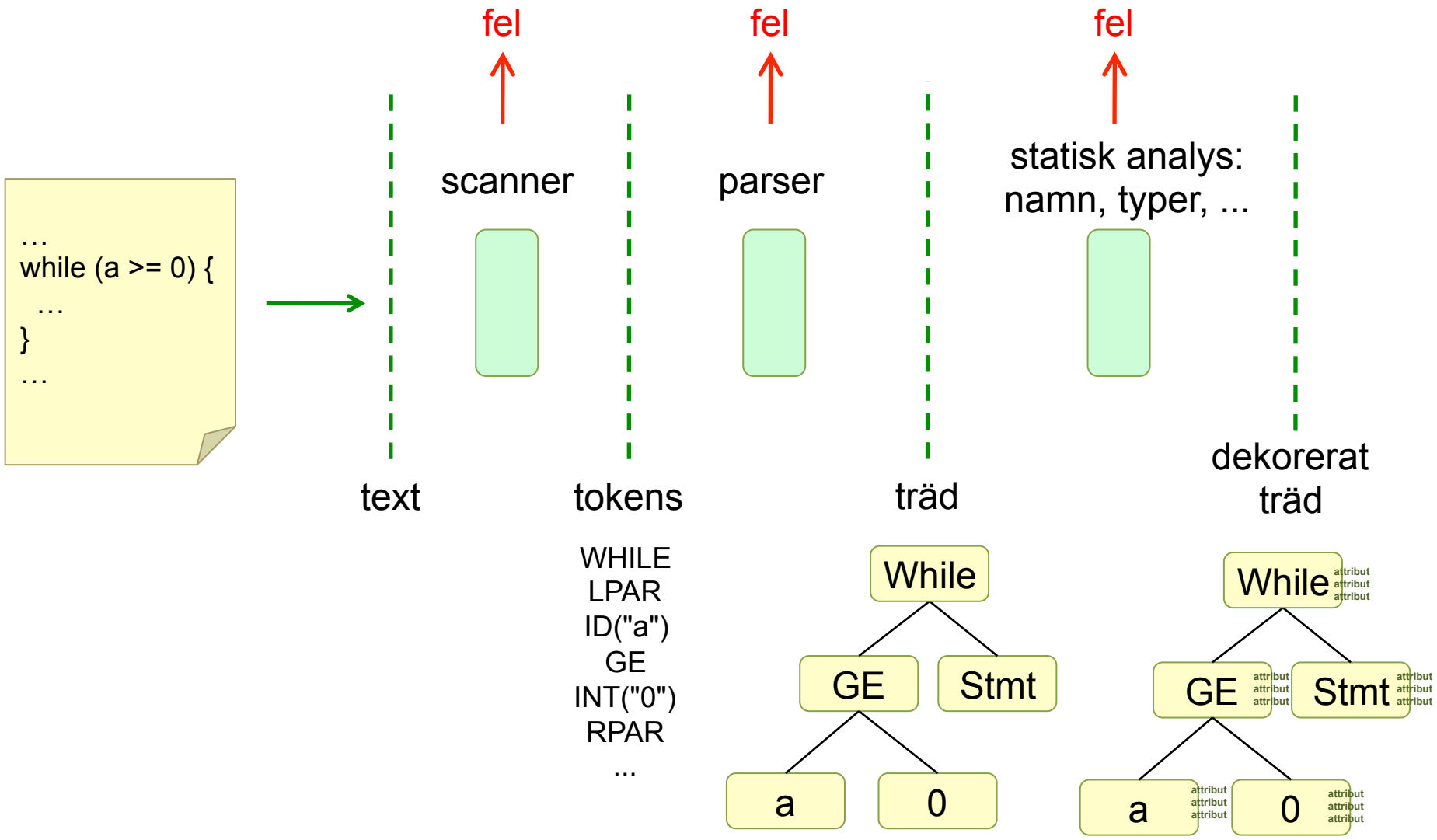
- Vi har LtoA.L
  - Kompilerar språket L till maskinen A
  - Implementerad i språket L självt.
- Vi har LtoA.A (skapad med bootstrapping)
- Vi har LtoB.L (implementerat ny backend för maskin B)
- Hur kan vi skapa LtoB.B (kompilerar till B-kod, på maskin B)?

- Kompilera LtoB.L med LtoA.A. Vi får LtoB.A – en *korskompilator*
- Kompilera LtoB.L med LtoB.A. Vi får LtoB.B.
- Nu kan vi kompilera L-program på maskin B och generera B-kod.

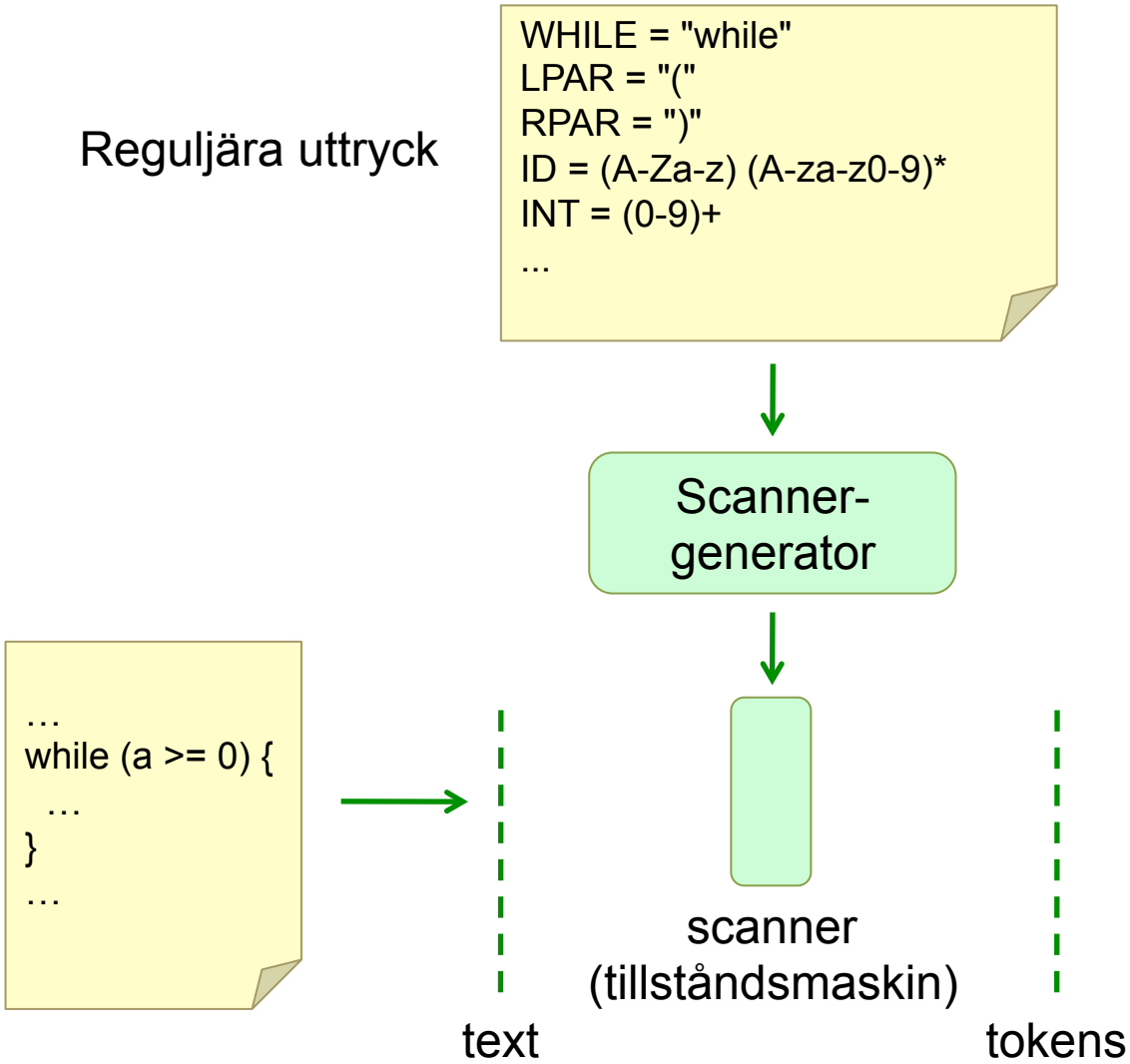
Mer detaljer i kompilatorn



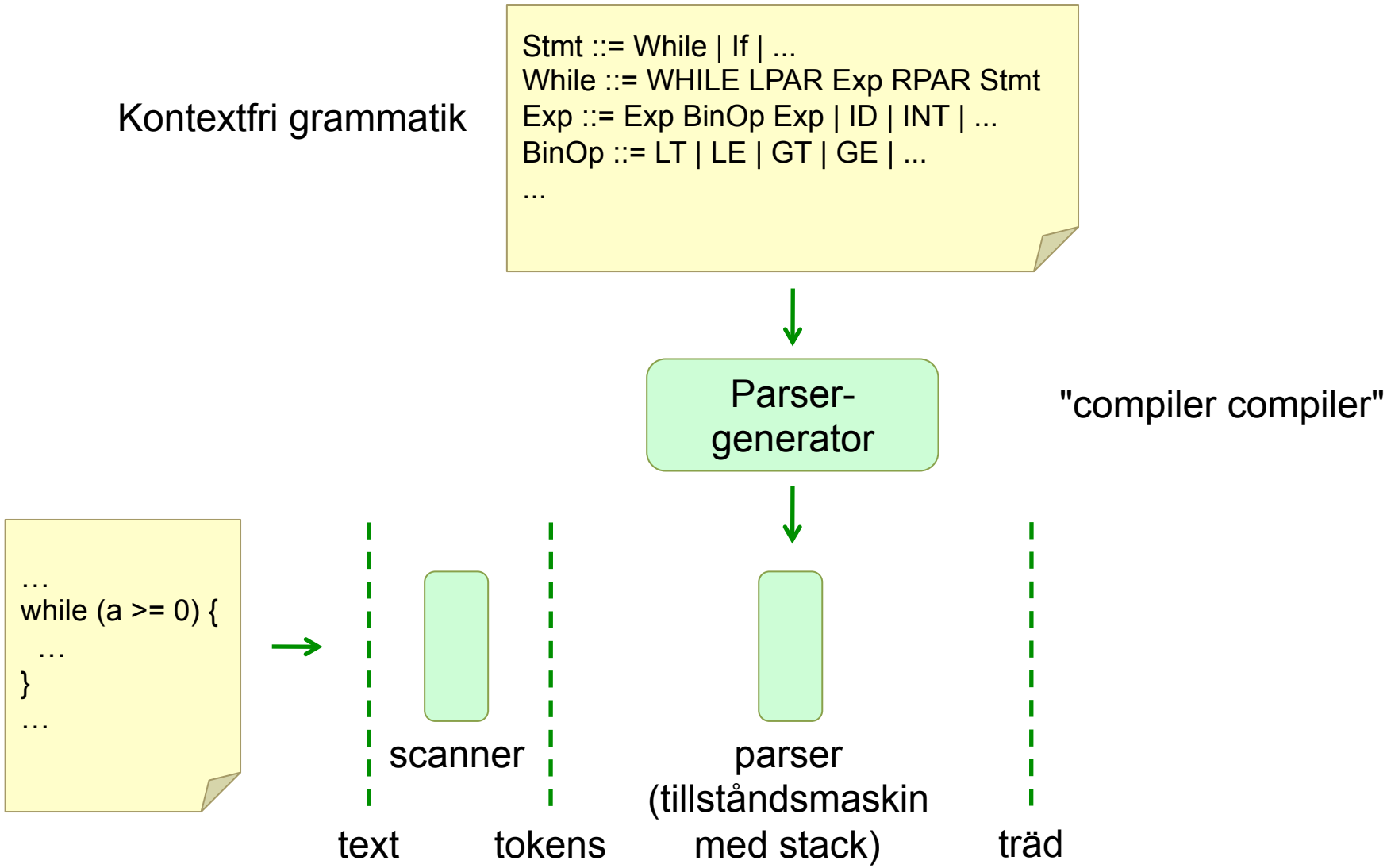
Analysdelen



Generering av Scanner



Generering av parser



Generering av statisk analysator

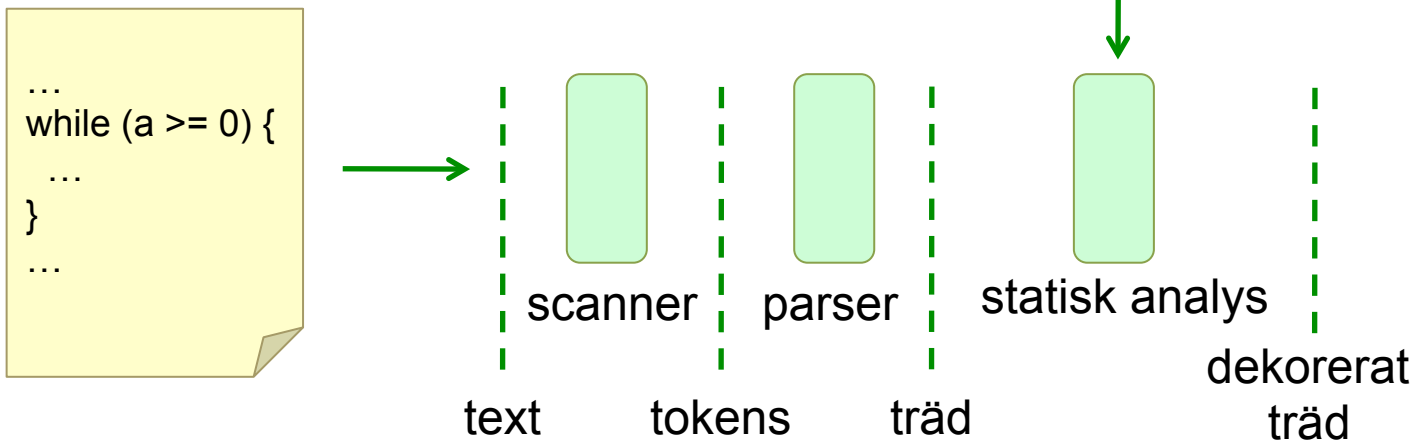
Attributgrammatik:
Trädnoderna förses med attribut som definieras av ekvationer.

```
syn Decl Use.decl = lookup(ID);
inh Decl Use.lookup(String s);
eq  Block.body.lookup(String s) = decls.localLookup(s);
syn Type Expr.type;
eq  Use.type = decl.type;
...
```

Attributevaluator-generator

jastadd

Forskningsprojekt vid Datavetenskap, LTH



Forskningsprojekt om utvidgningsbara kompilatorer


java 1.4


java 5


java 7


java 8

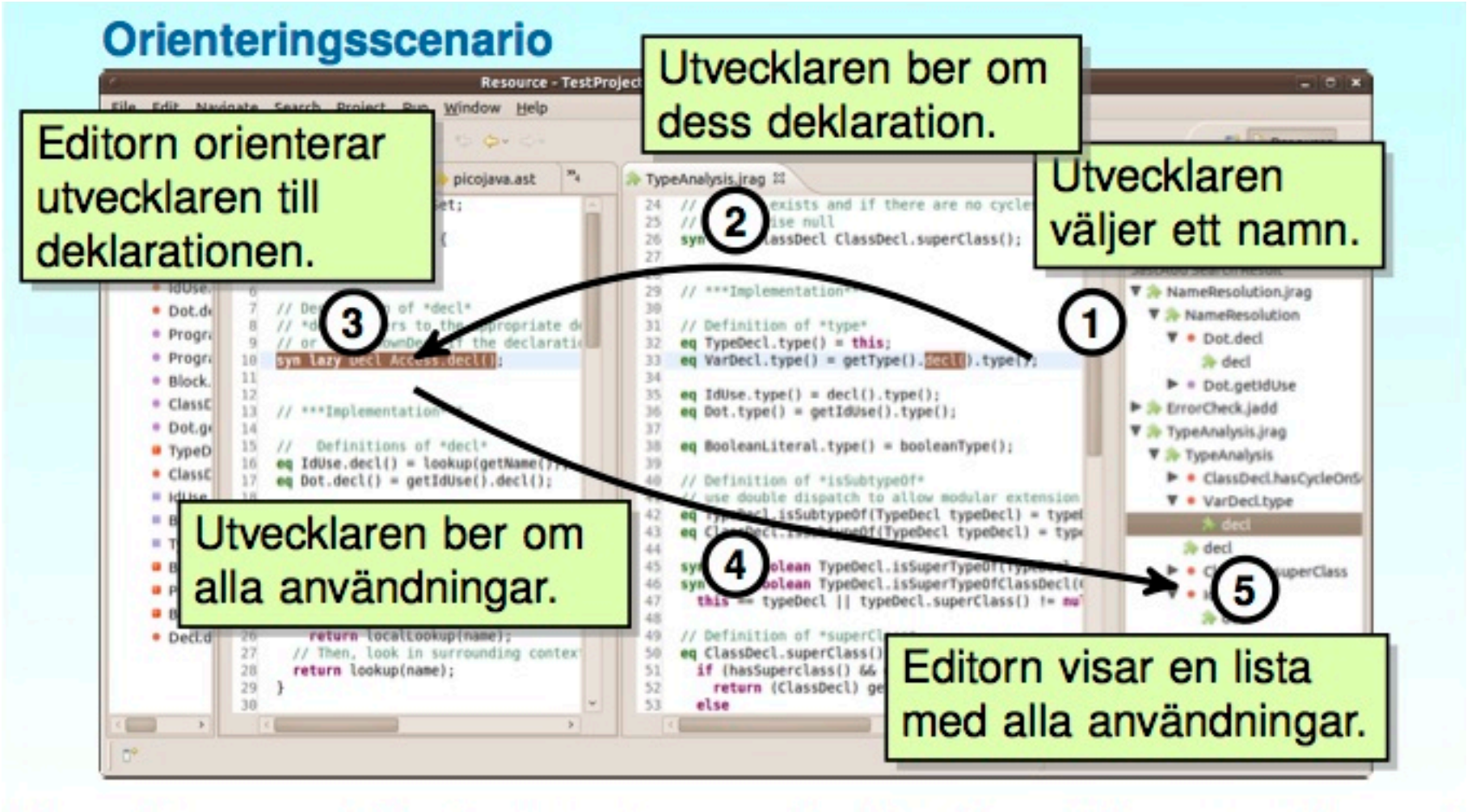

Refaktorisering


IDE-stöd

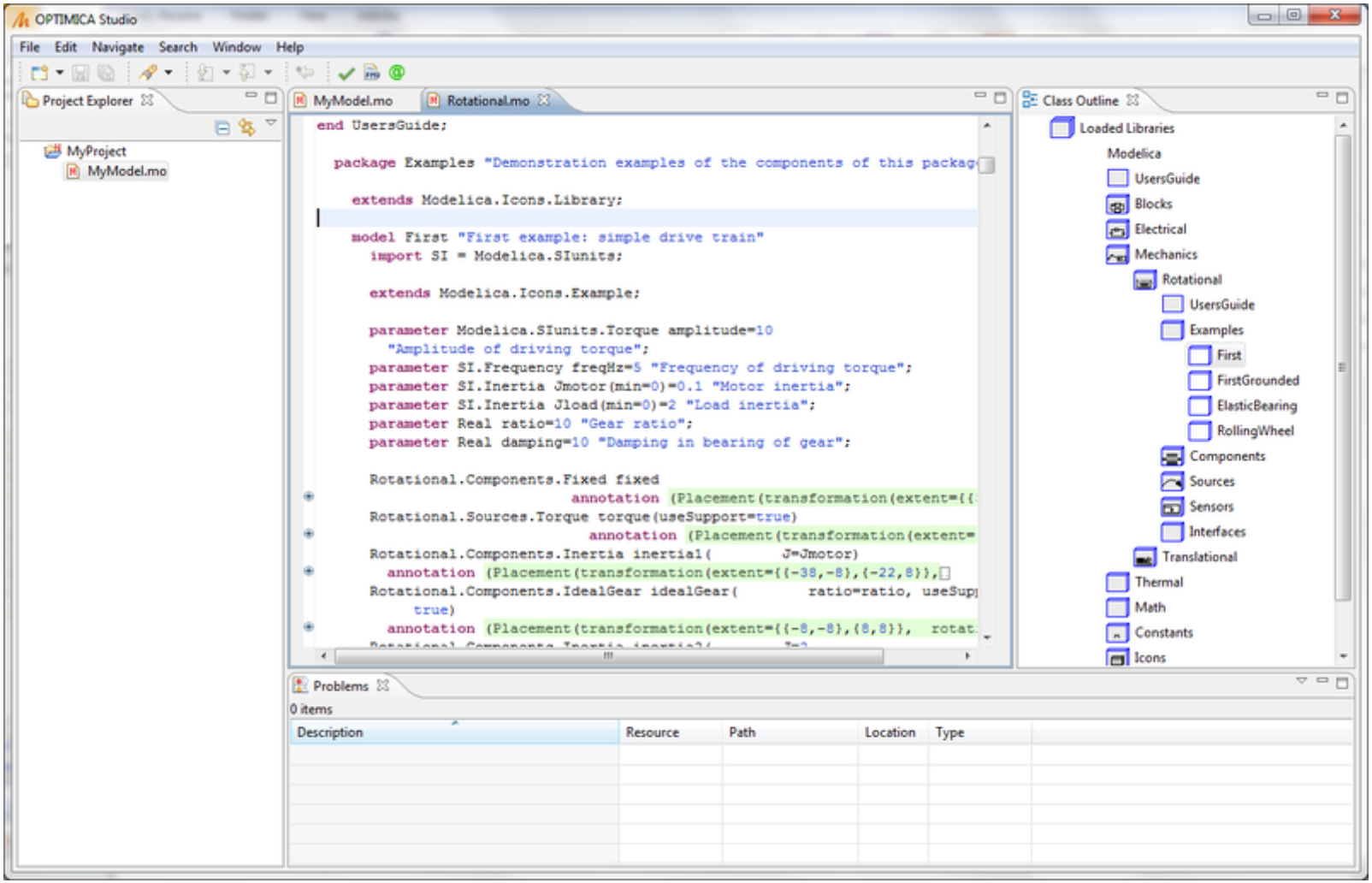

Modelica


Visuella språk

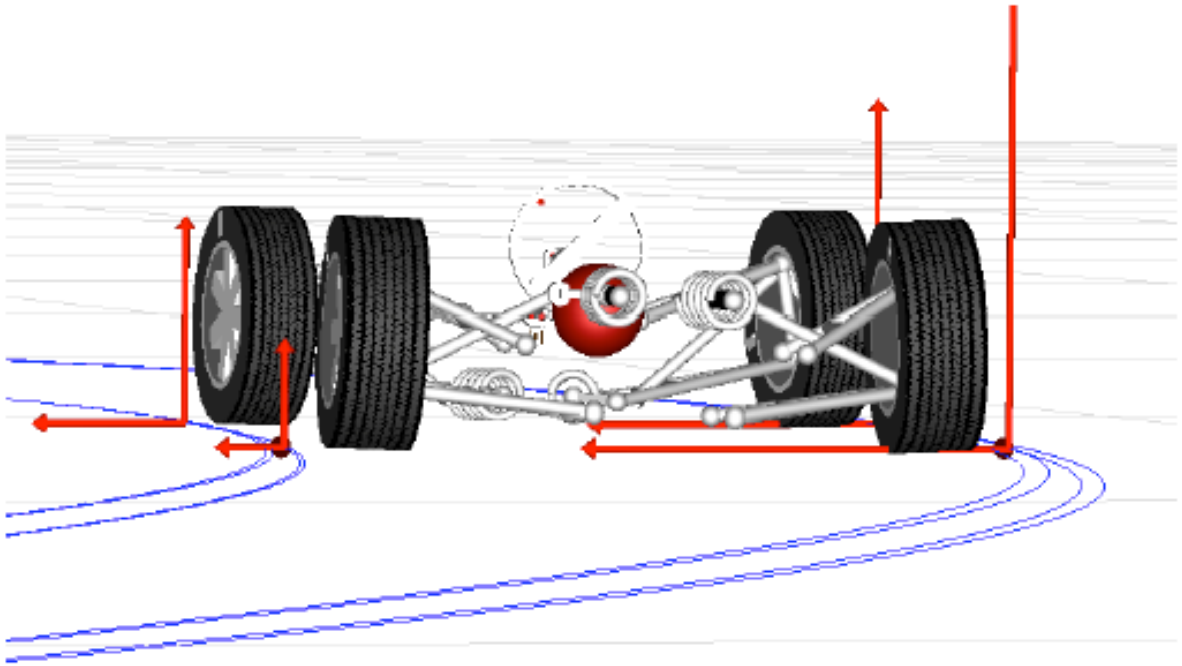
Orienteringsstöd



JModelica.org

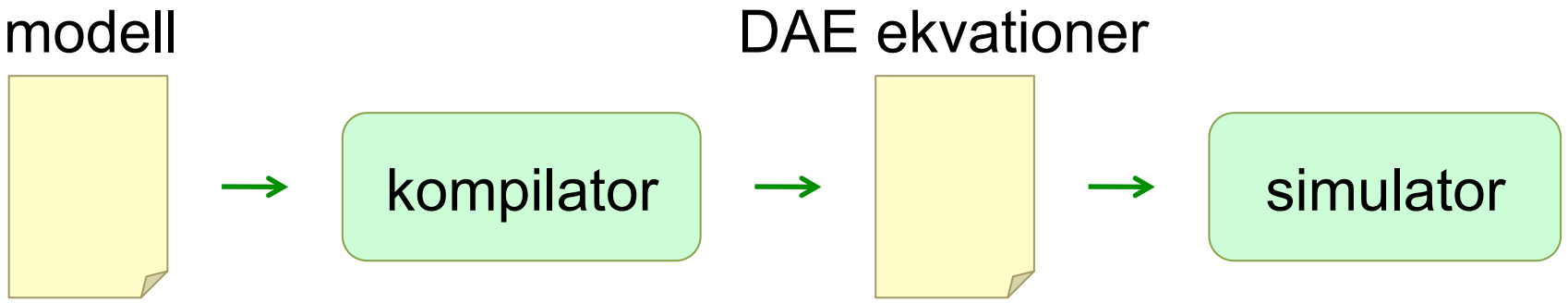


Kompilator och IDE utvecklad med hjälp av JastAdd



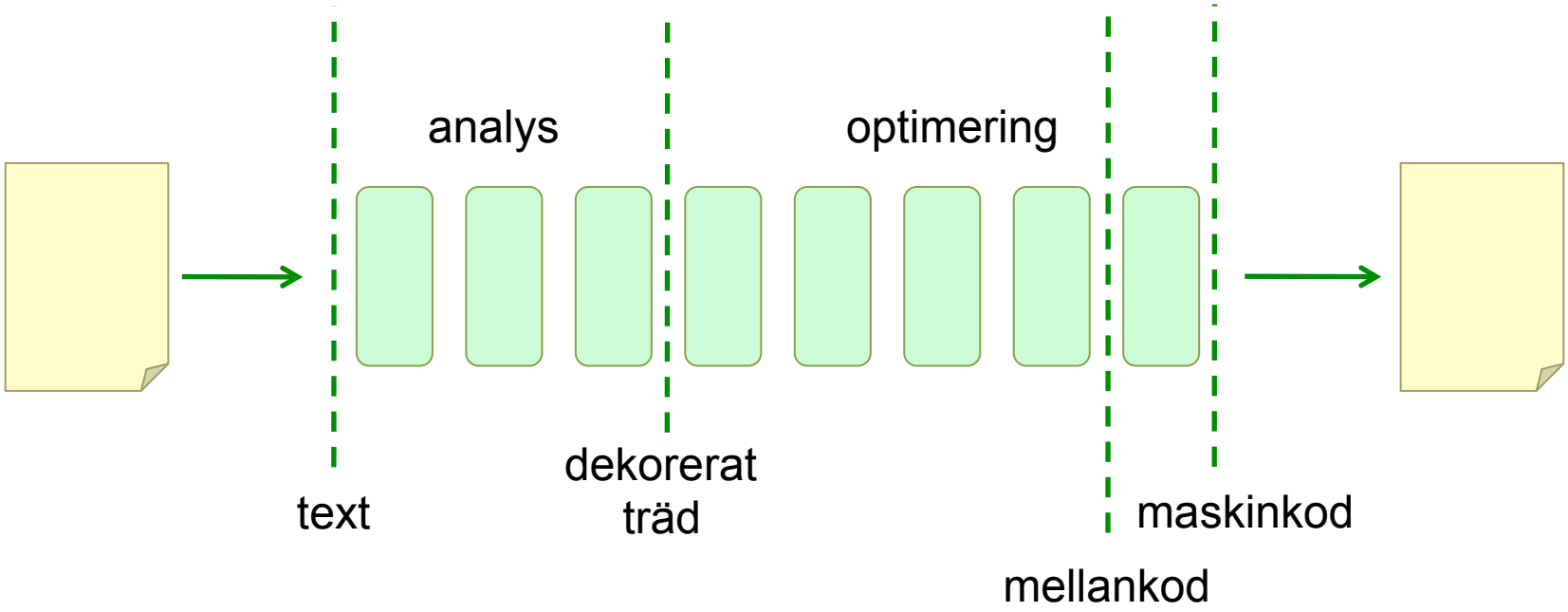
Exempel på modellering och optimering av fysikaliskt system med hjälp av JModelica.

JModelica-kompilatorn



Kompilatorteknik

generell teknik för många tillämpningar



Fortsättningskurser

EDAN65, Kompilatorteknik
Scanning, Parsing, Analys, Kodgenerering.
Implementera liten kompilator.

EDA230, Optimerande kompilatorer
Förstå möjligheter och begränsningar hos moderna optimerande kompilatorer.

EDAN40, Funktionsprogrammering
EDAN01, Constraintprogrammering

EDAN20, Språkteknologi
Om analys av naturligt språk

EDAN70, Projekt i datavetenskap
Forskningsnära projekt i t.ex. kompilatorer.