

Kompilator teknik

Görel Hedin
Datavetenskap
Lunds Tekniska Högskola

Temaföreläsning, Datorer i system, 2014



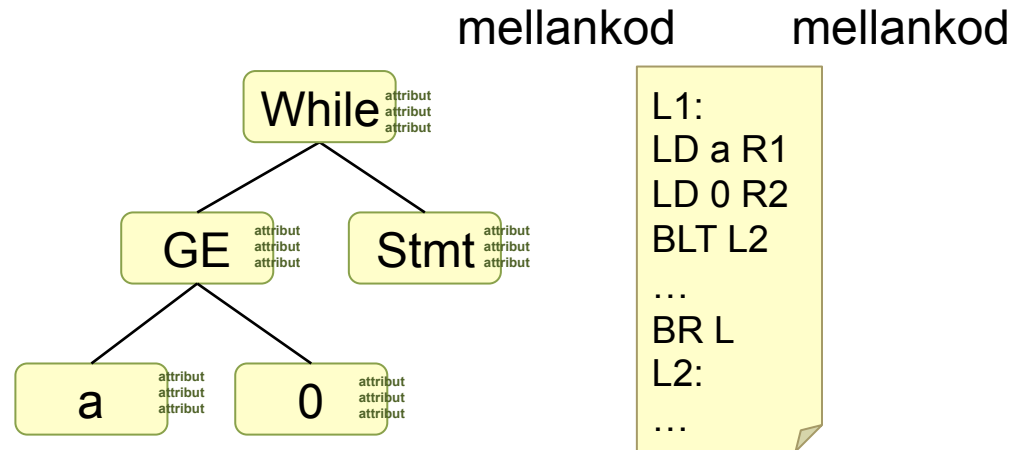
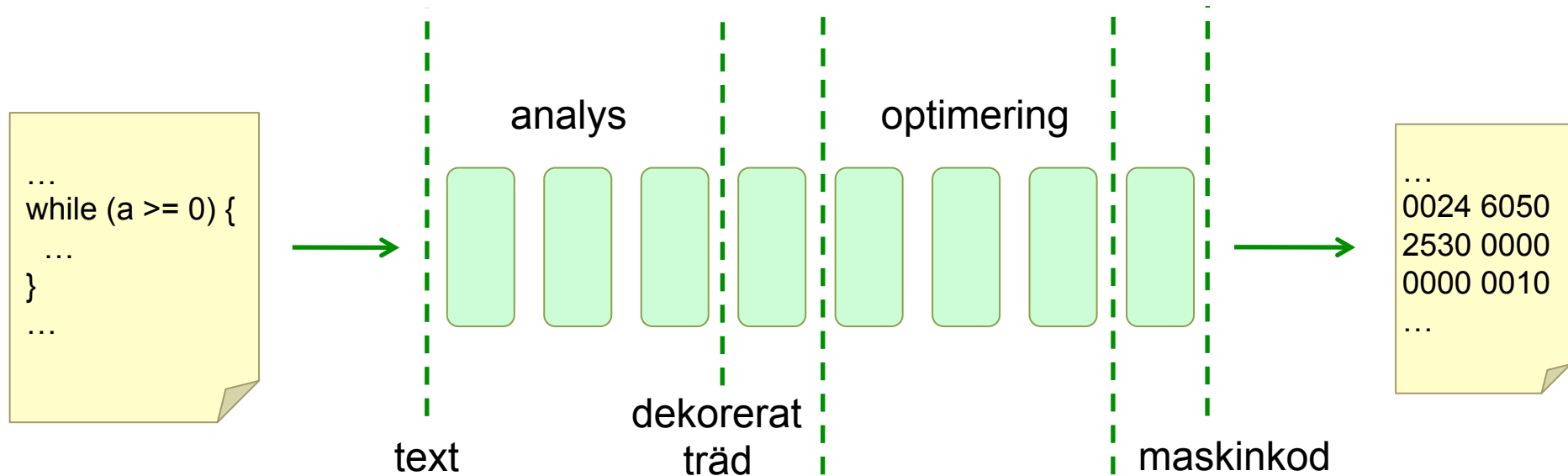
En typisk kompilator



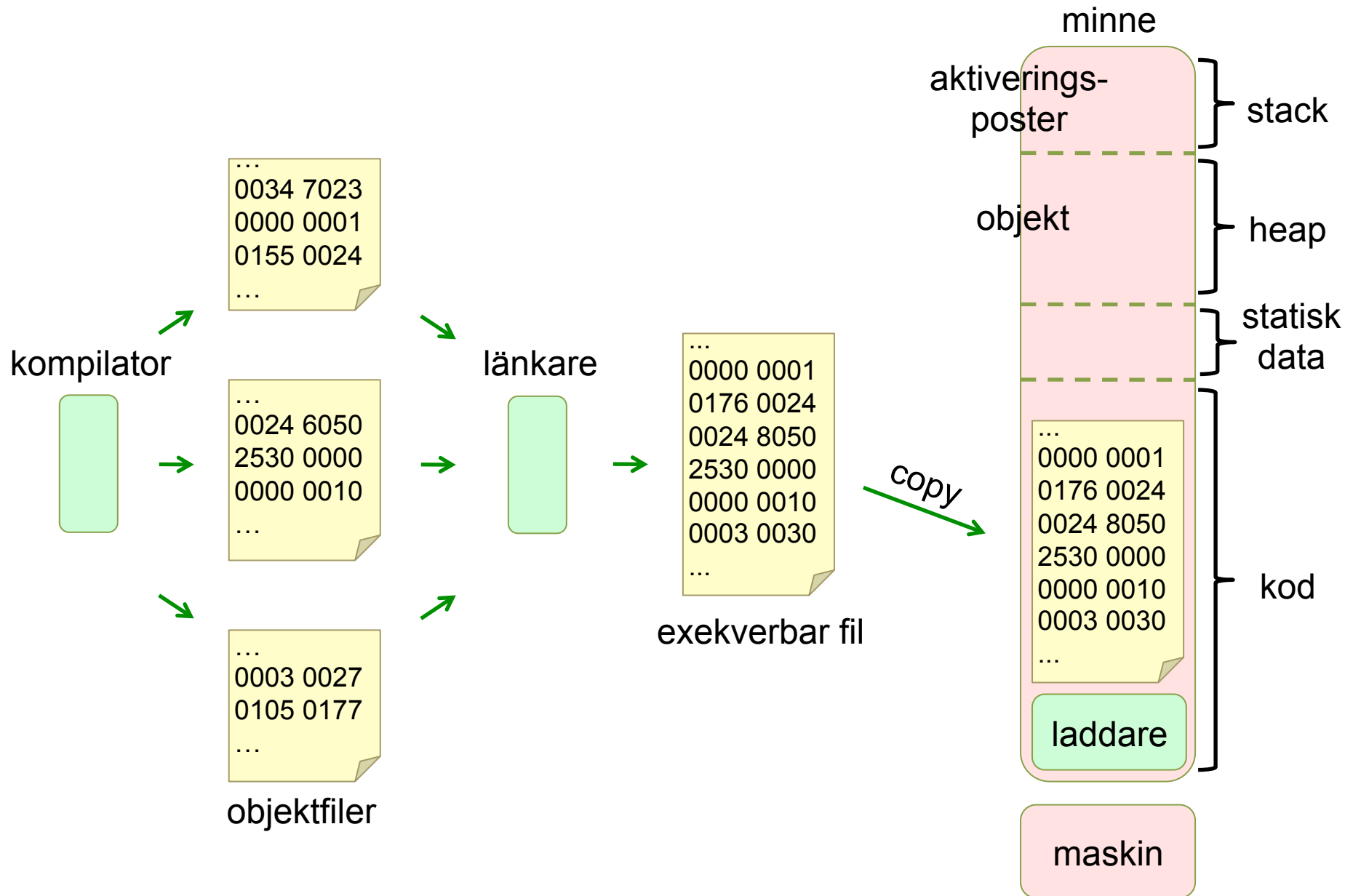
Men tekniken är generell:



Typiska steg i en kompilator

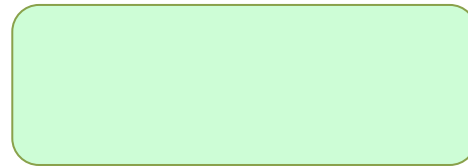


Länkning, laddning, exekvering



Varför lära sig kompilator teknik?

data på format X



data på format Y



- 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.

1967

Simula 67

```
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;
```

- 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

The screenshot displays the Smalltalk environment with several overlapping windows:

- System Browser:** A hierarchical tree view on the left showing categories like 'Collections-Sequence', 'Graphics-Primitives', and 'Graphics-Display'. It includes sub-panels for 'instance' and 'class'.
- Code Editor (Top Left):** Contains the definition for the `collect:` message:


```
collect: aBlock
  "Evaluate aBlock with each of my elements as the argument. Collect
  resulting values into a collection that is like me. Answer with
  collection. Override superclass in order to use add:, not at:put:."

  | newCollection |
  newCollection ← self species new.
  self do: [:each | newCollection add: (aBlock value: each)].
  tnewCollection
```
- User Interrupt (Middle Left):** Shows a stack of messages:


```
Paragraph>>characterBlockAtPoint:
Paragraph>>mouseSelect:to:
CodeController(ParagraphEditor)>>processRedButton
CodeController(ParagraphEditor)>>processMouseButtons
CodeController(ParagraphEditor)>>controlActivity
CodeController(Controller)>>controlLoop
```
- controlActivity (Bottom Left):** Shows a snippet of code:


```
self scrollBarContainsCursor
  ifTrue:
    [self scroll]
  ifFalse:
    [self processKeybo
    self processMouseE
```
- File List (Bottom Center):** A list of files:


```
File List
[ ]<Robson>SF>*
[File] <Robson>SF>ScreenForm.st
[File] <Robson>SF>ScreenForm.text
[File] <Robson>SF>ScreenFormChanges.st
[File] <Robson>SF>WordGraphics.form
```
- Code Editor (Bottom Center):** Shows a snippet of code:

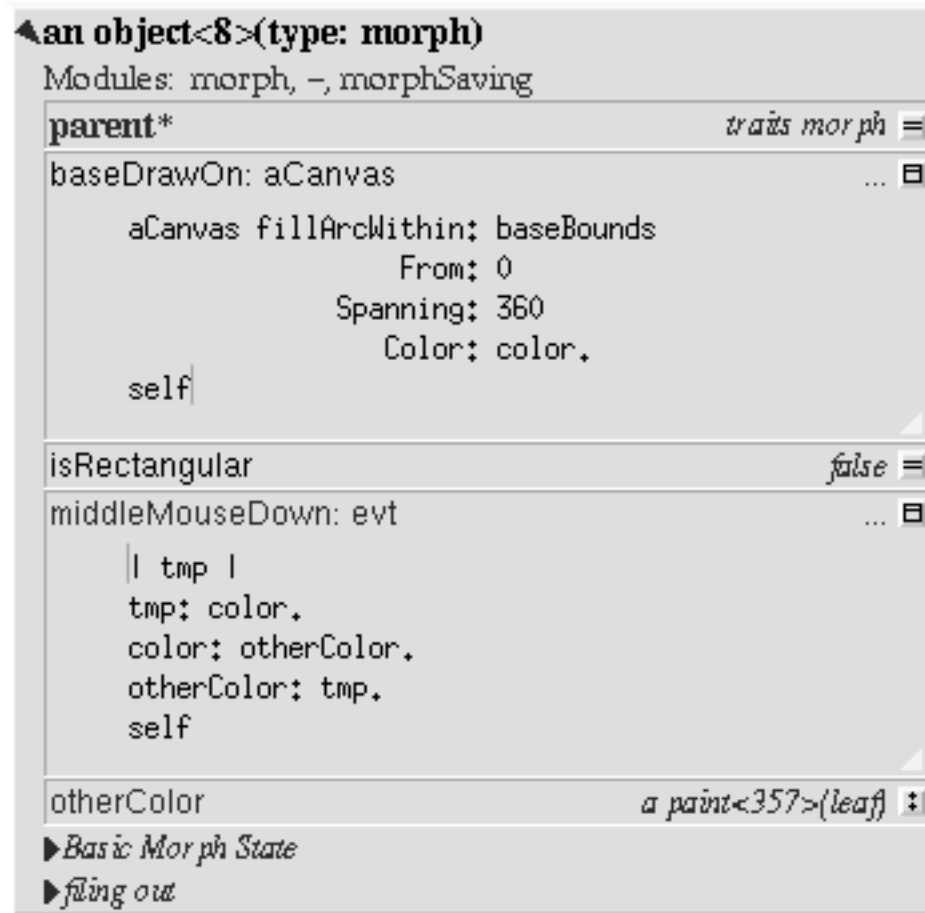

```
Rectangle fromUser origin

ScreenForm setFullPageWidth.
ScreenForm
  printRectangle:
    (30@5 extent: 674@790)
  onFileName: 'ExampleScreen.press'
```
- Fig. 1 (Right):** A technical drawing of a bolt and nut, labeled 'Fig. 1'.
- System Browser (Top Right):** Includes a clock icon, a calendar icon, and a 'Form Editor' window.

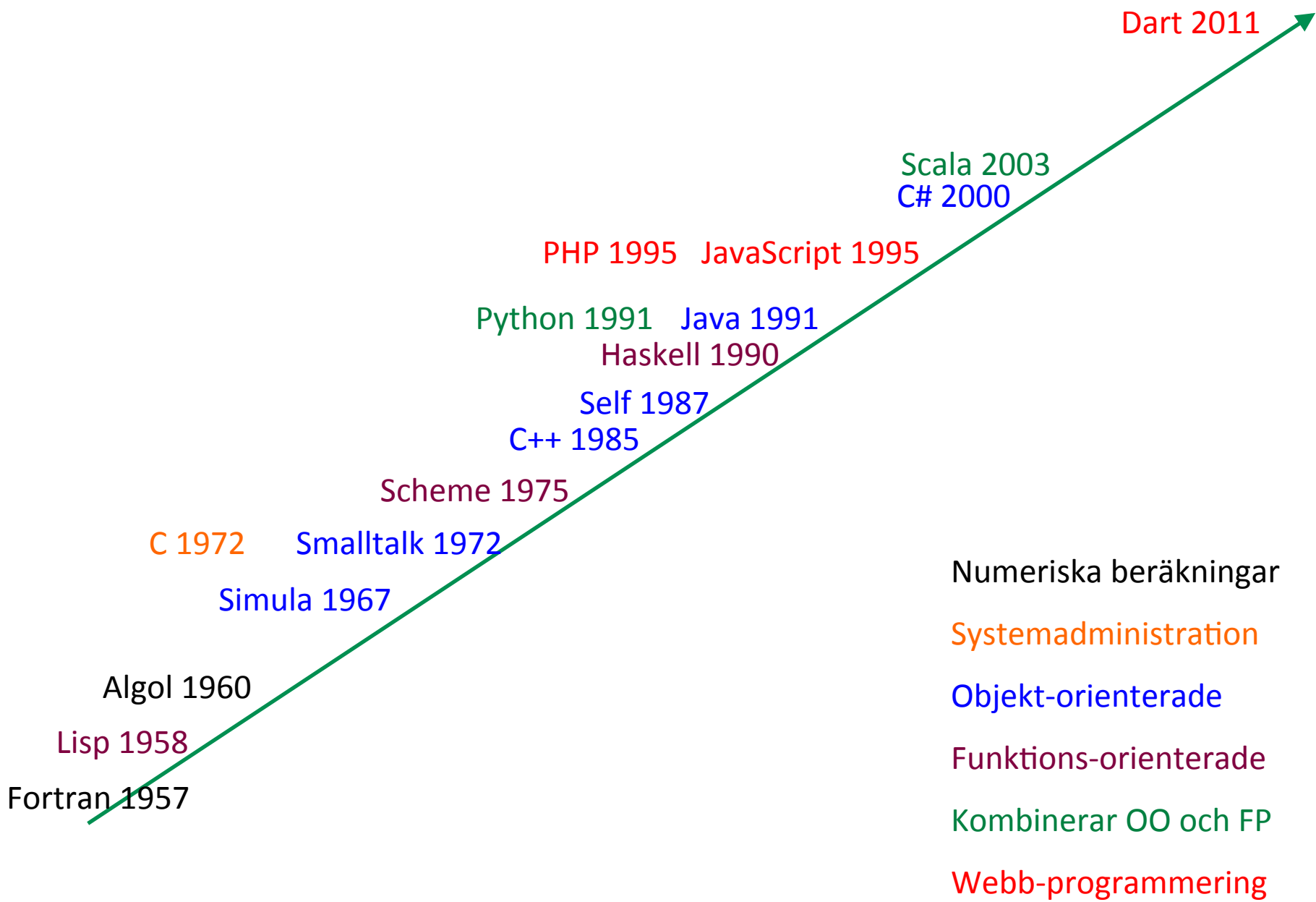
- Smalltalk, av Alan Kay m. fl.
- Influenser från Lisp och Simula
- *"Everything is an object"*
- Dynamiskt typat
- Reflection
- Bytecode
- JIT: Just-in-time compilation (kompilera under runtime).

1987

Self



- David Ungar m. fl.
- Influenser från Smalltalk
- Prototyper i stället för klasser: kлона 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.



Numeriska beräkningar

Systemadministration

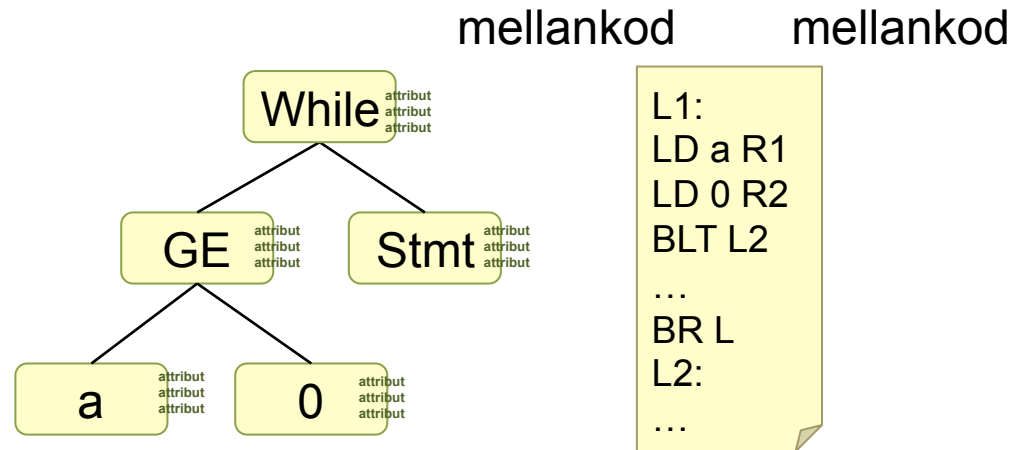
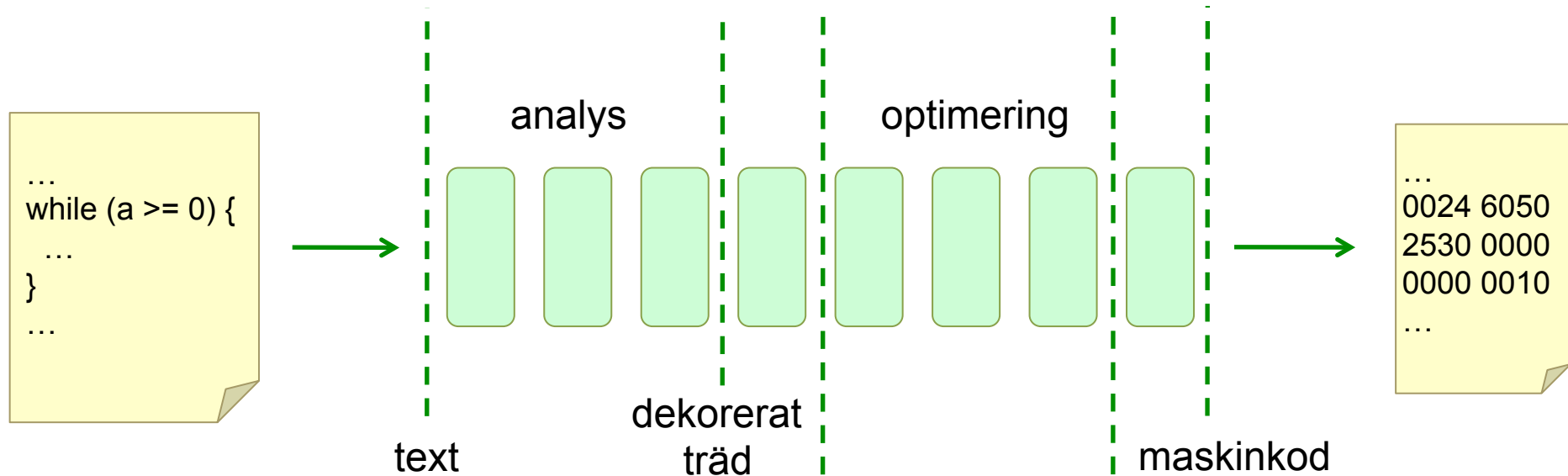
Objekt-orienterade

Funktions-orienterade

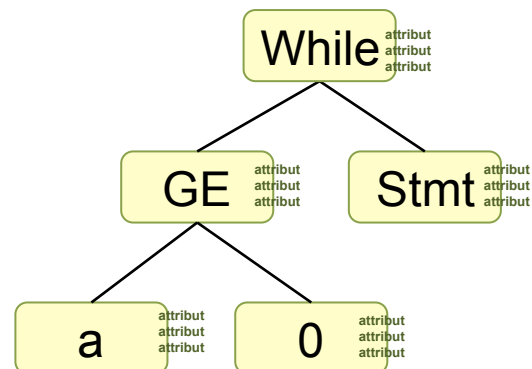
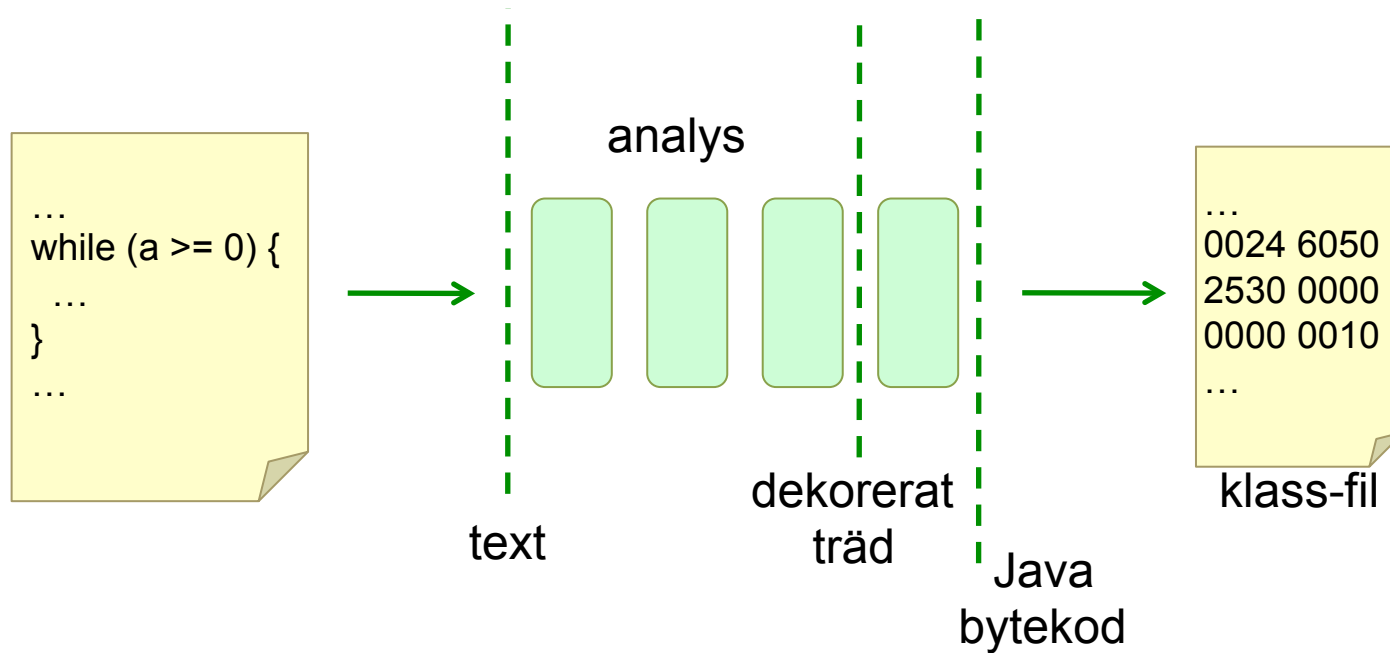
Kombinerar OO och FP

Webb-programmering

Typiska steg i en kompilator

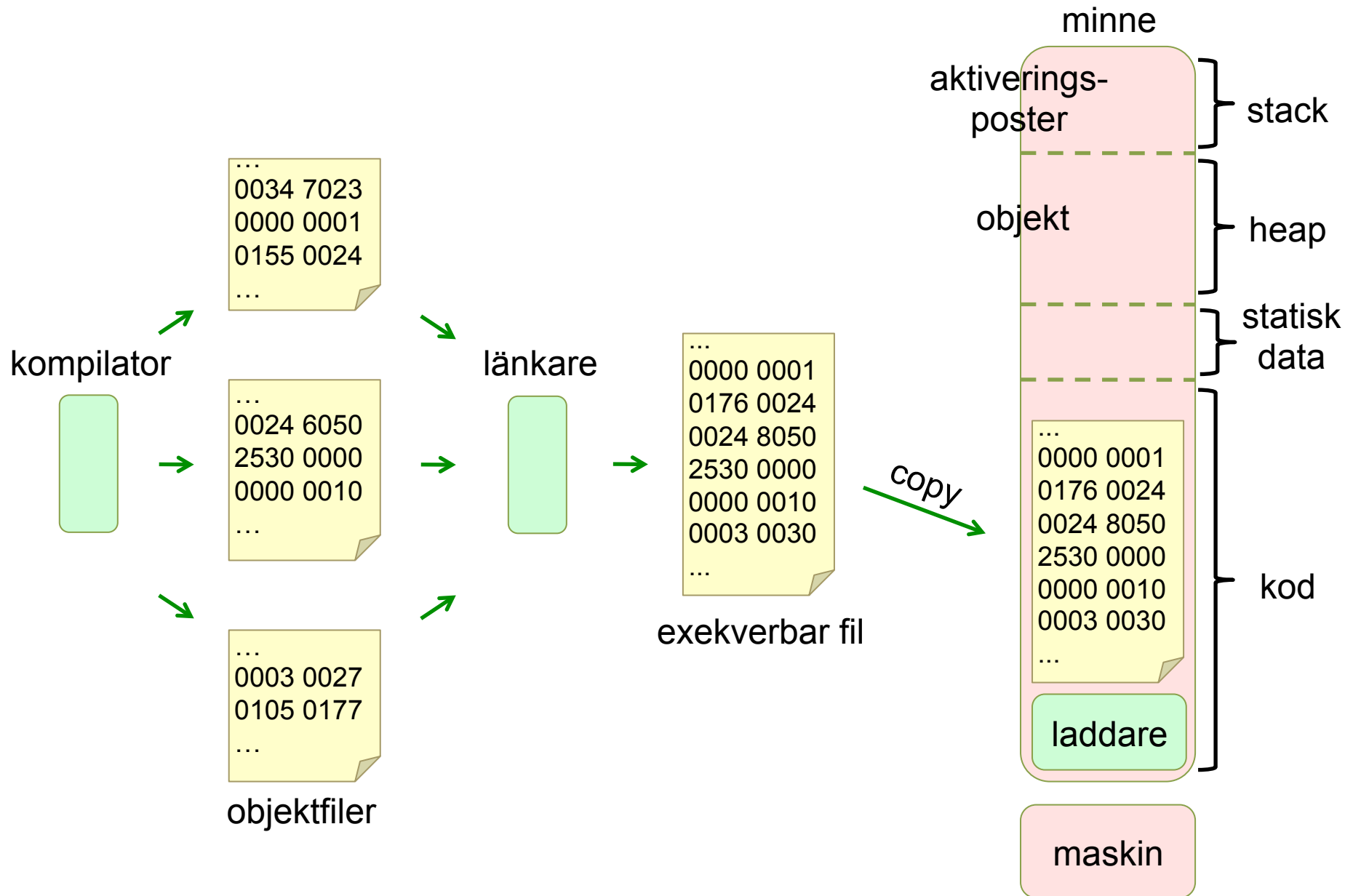


Java-kompilatorn

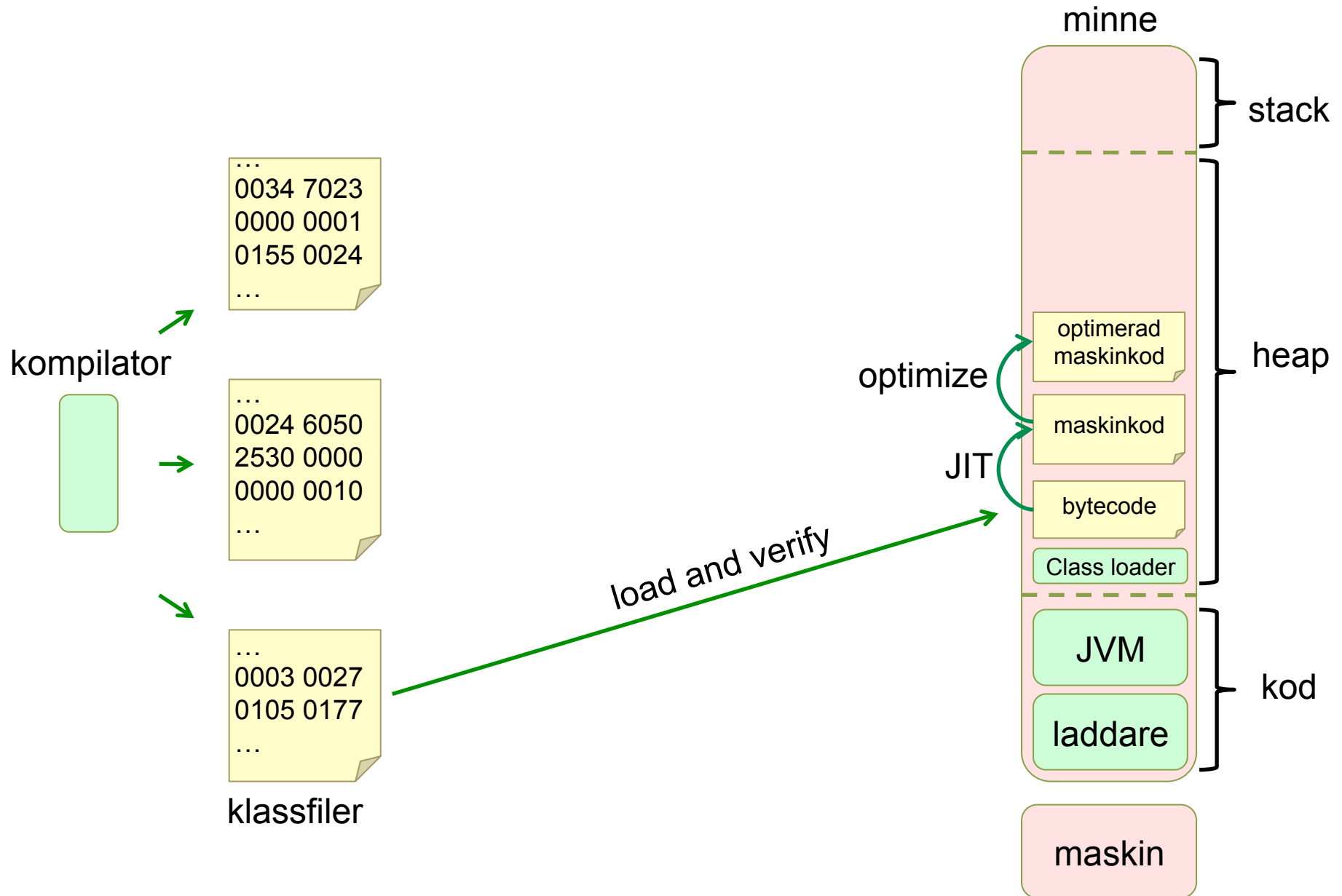


- Klassfiler: binärkod för en *virtuell maskin*
- Mellankodsnivå, ej optimerade

Länkning, laddning, exekvering



Java-exekvering



Implementation av programspråk

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

Bootstrapping!



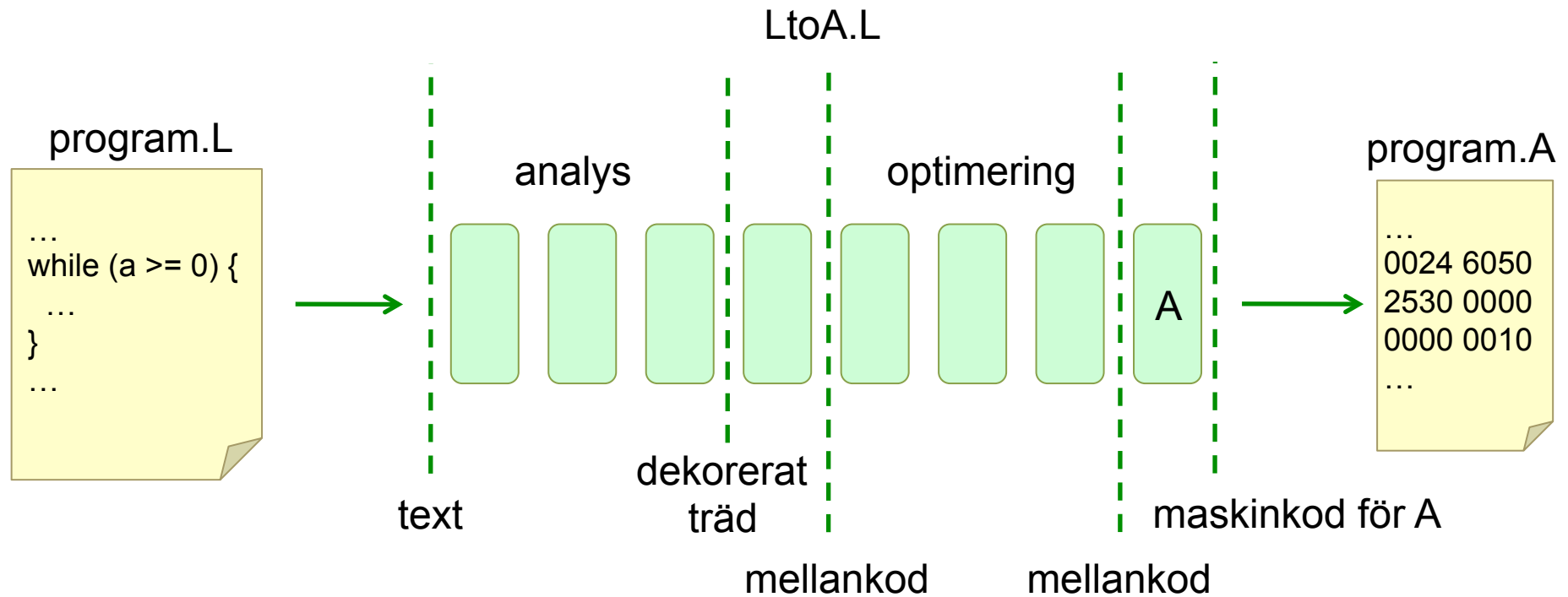
Bootstrapping

Språkversion	Implementeras i
C ₁	Assembler
C ₂	C ₁
C ₃	C ₂
...	

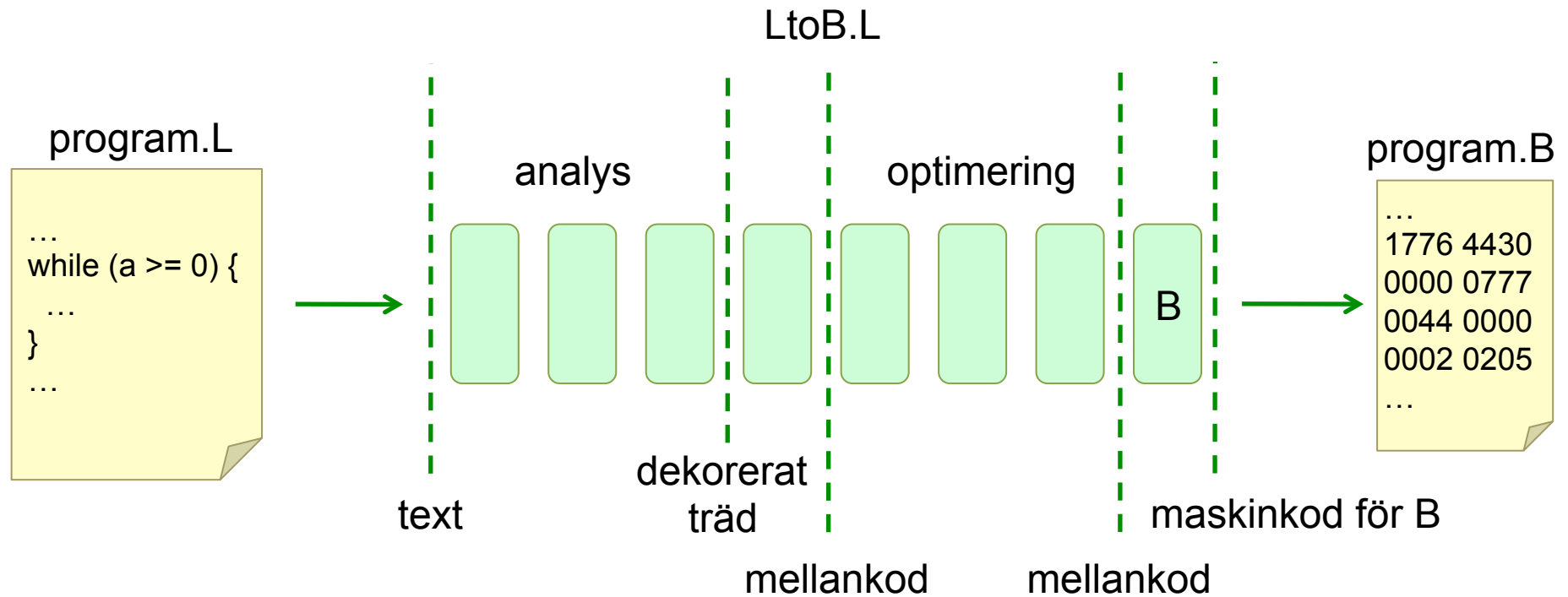
Språkversion	Implementeras i
Java ₁	C
Java ₂	Java ₁
Java ₃	Java ₂
...	

Språkversion	Implementeras i
Pizza	Java
Scala ₁	Pizza
Scala ₂	Scala ₁
...	

Kompilera till ny plattform



Kompilerera till ny plattform



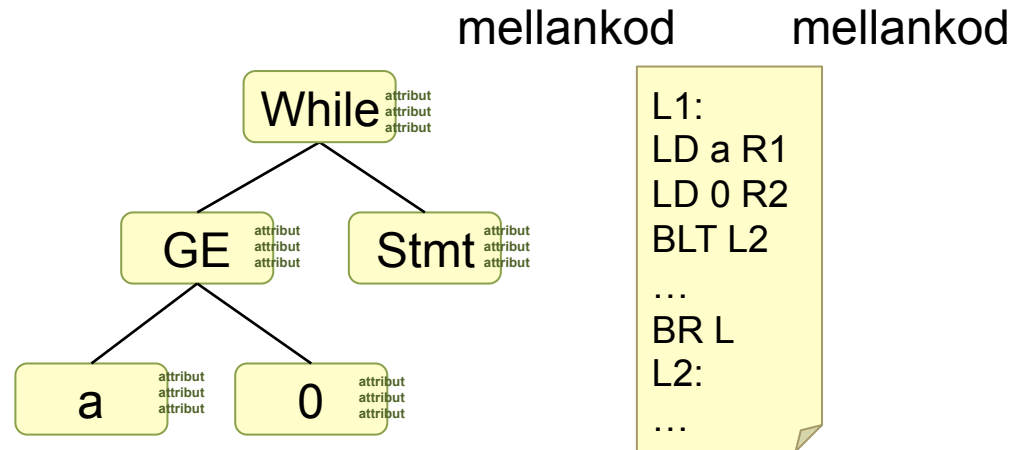
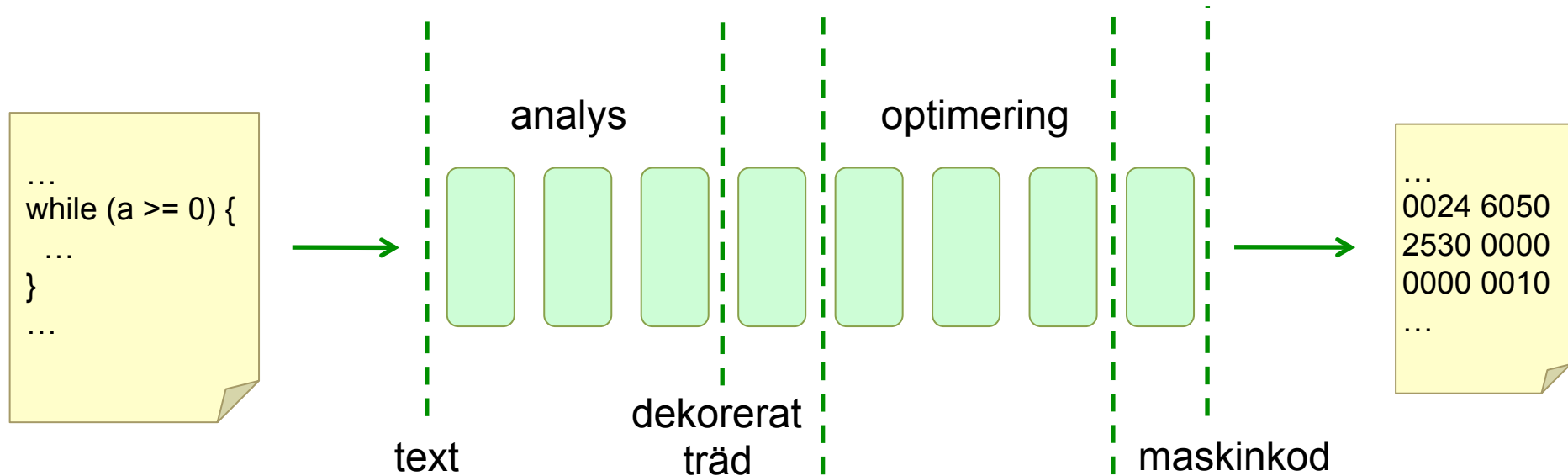
Endast sista steget (backend) behöver bytas

Korskompilering till ny plattform

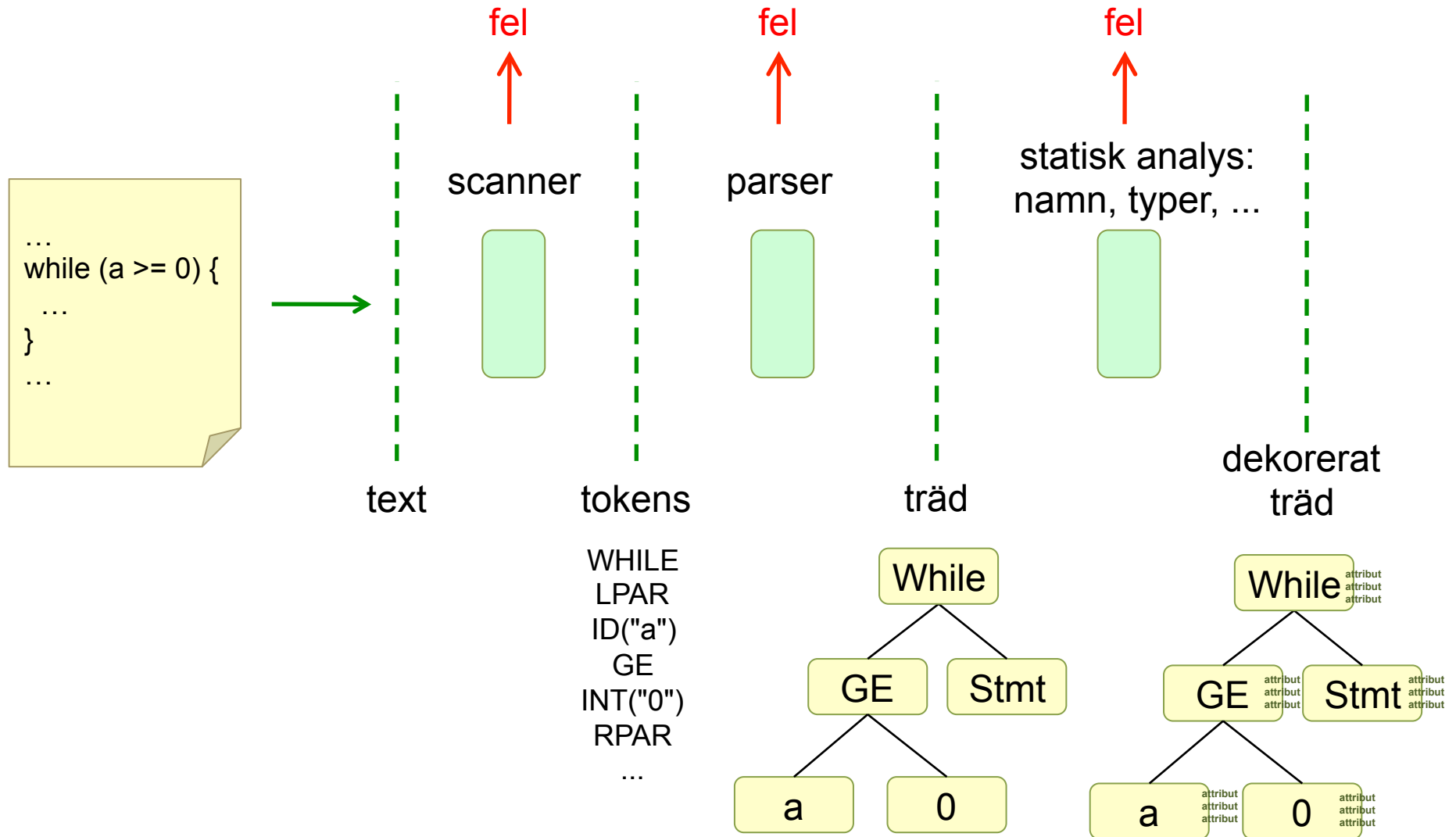
- Vi har LtoA.L
 - Kompilerar språket L till maskinen A
 - Implementerad i språket L självt.
- Vi har LtoA.A (skapad med bootstrapping)
- Vi har LtoB.L (implementerat ny backend för maskin B)
- Hur kan vi skapa LtoB.B (kompilerar till B-kod, på maskin B)?

- 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.

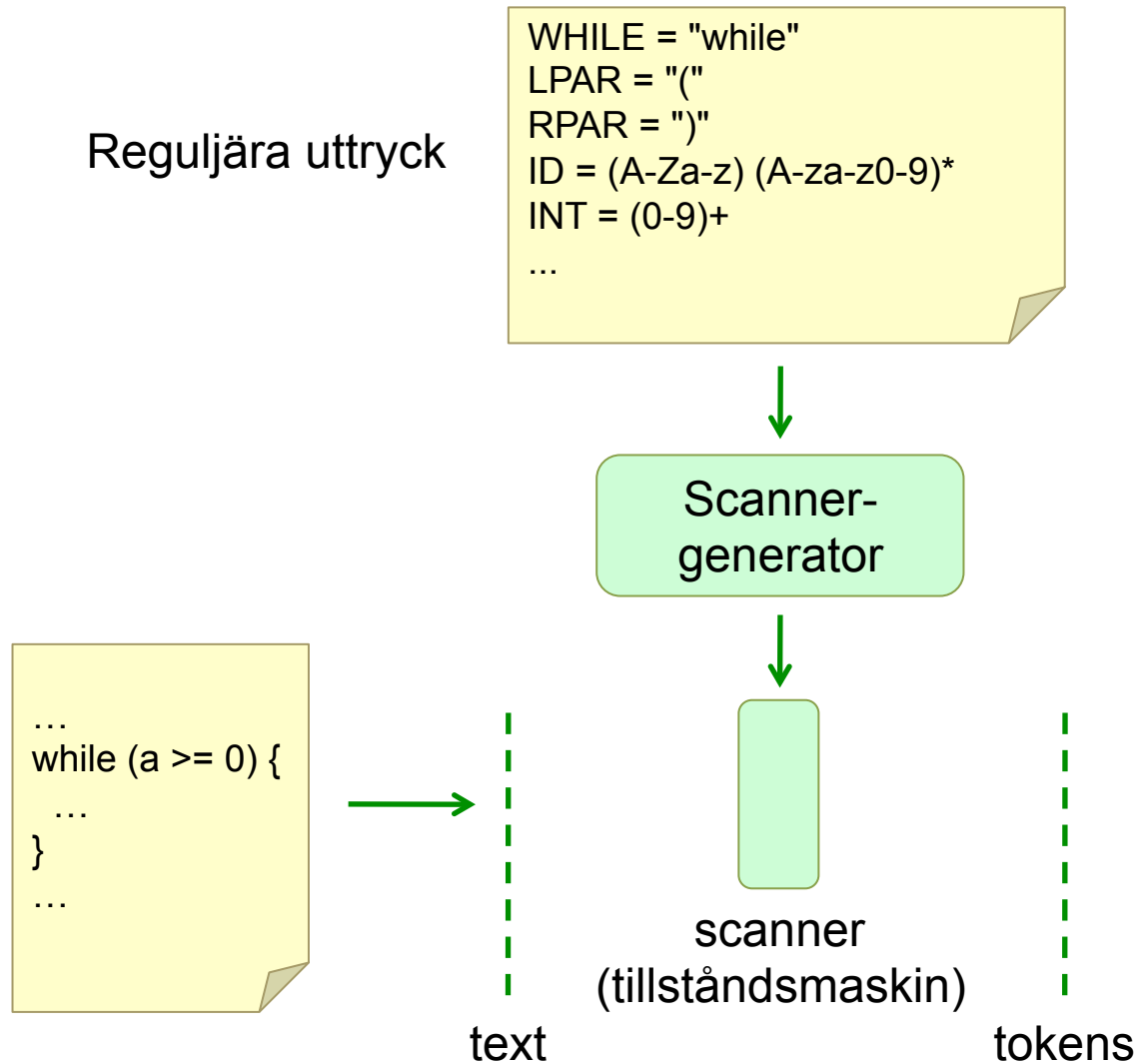
Mer detaljer i kompilatorn



Analysdelen



Generering av Scanner



Generering av parser

Kontextfri grammatik

```
Stmt ::= While | If | ...  
While ::= WHILE LPAR Exp RPAR Stmt  
Exp ::= Exp BinOp Exp | ID | INT | ...  
BinOp ::= LT | LE | GT | GE | ...  
...
```

Parser-
generator

"compiler compiler"

```
...  
while (a >= 0) {  
  ...  
}  
...
```



text



scanner

tokens



parser
(tillståndsmaskin
med stack)

träd

Generering av statisk analysator

Attributgrammatik:
Trädnoderna förses med
attribut som definieras av
ekvationer.

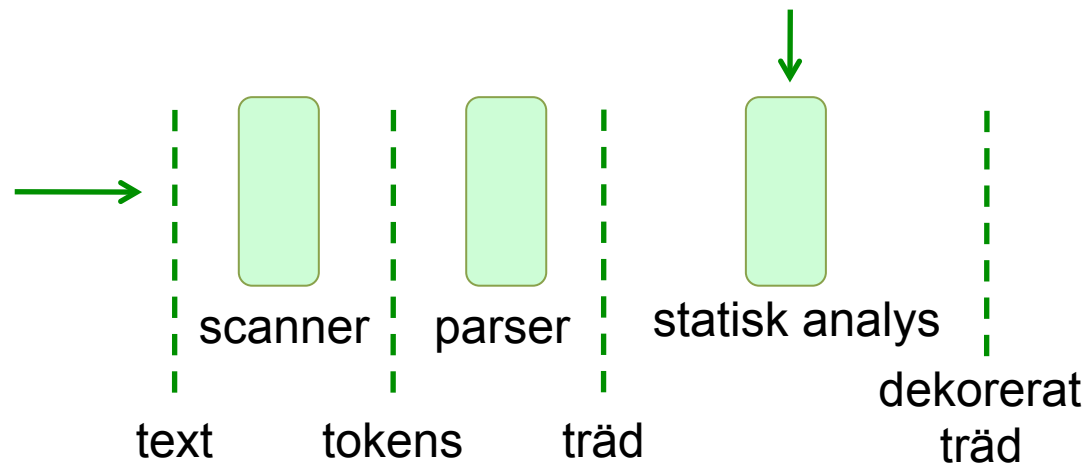
```
syn Decl Use.decl = lookup(ID);  
inh Decl Use.lookup(String s);  
eq Block.body.lookup(String s) = decls.localLookup(s);  
syn Type Expr.type;  
eq Use.type = decl.type;  
...
```

Attributevaluator-
generator



Forskningsprojekt vid
Datavetenskap, LTH

```
...  
while (a >= 0) {  
...  
}  
...
```



jastadd

Forskningsprojekt om utvidgningsbara kompilatorer



java 1.4



java 5



java 7



java 8



Refaktorisering



IDE-stöd



Modelica



Visuella språk



Orienteringsstöd

Orienteringsscenario

1 Editorn orienterar utvecklaren till deklARATIONEN.

2 Utvecklaren ber om dess deklARATION.

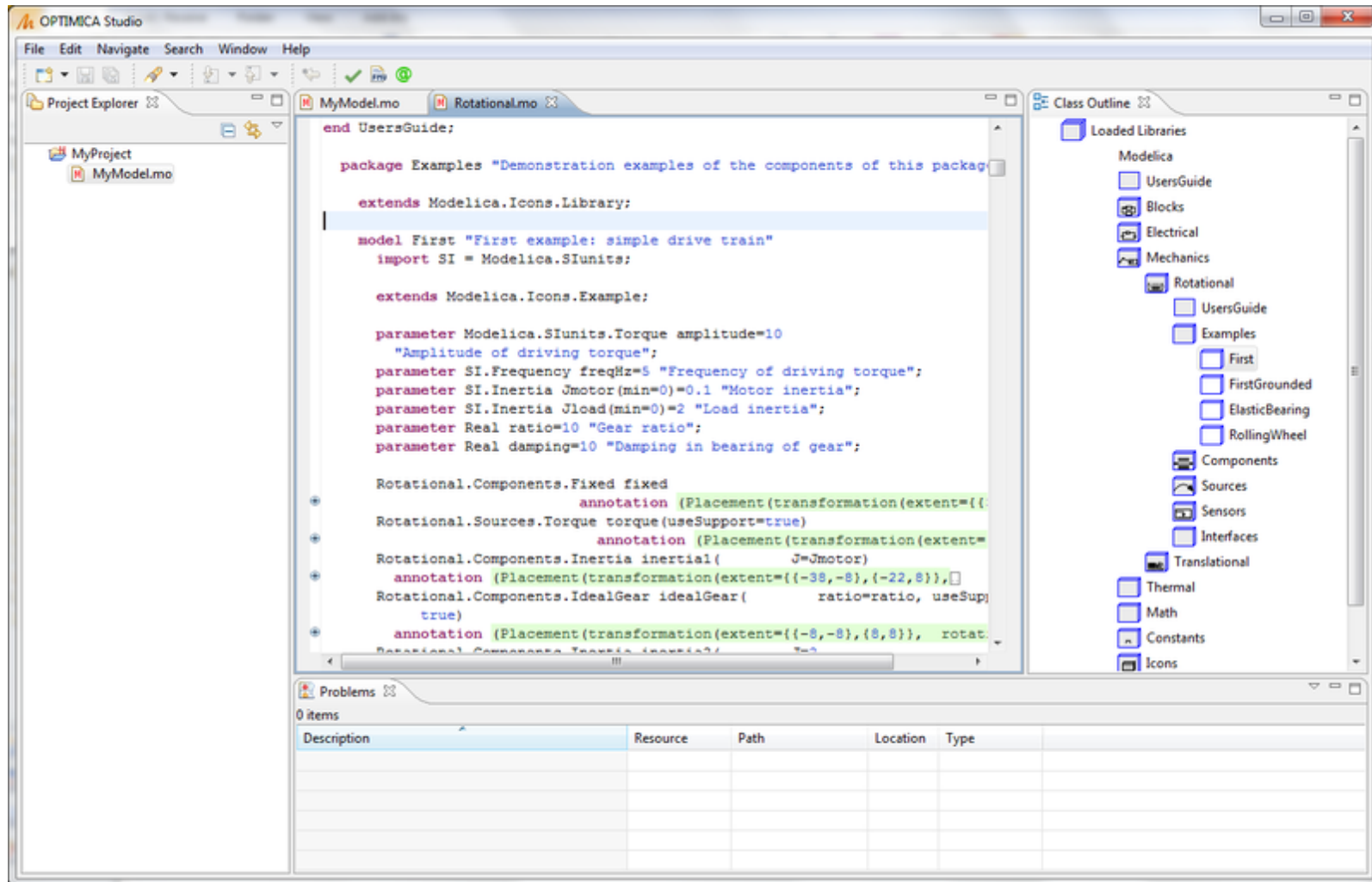
3 Utvecklaren väljer ett namn.

4 Utvecklaren ber om alla användningar.

5 Editorn visar en lista med alla användningar.

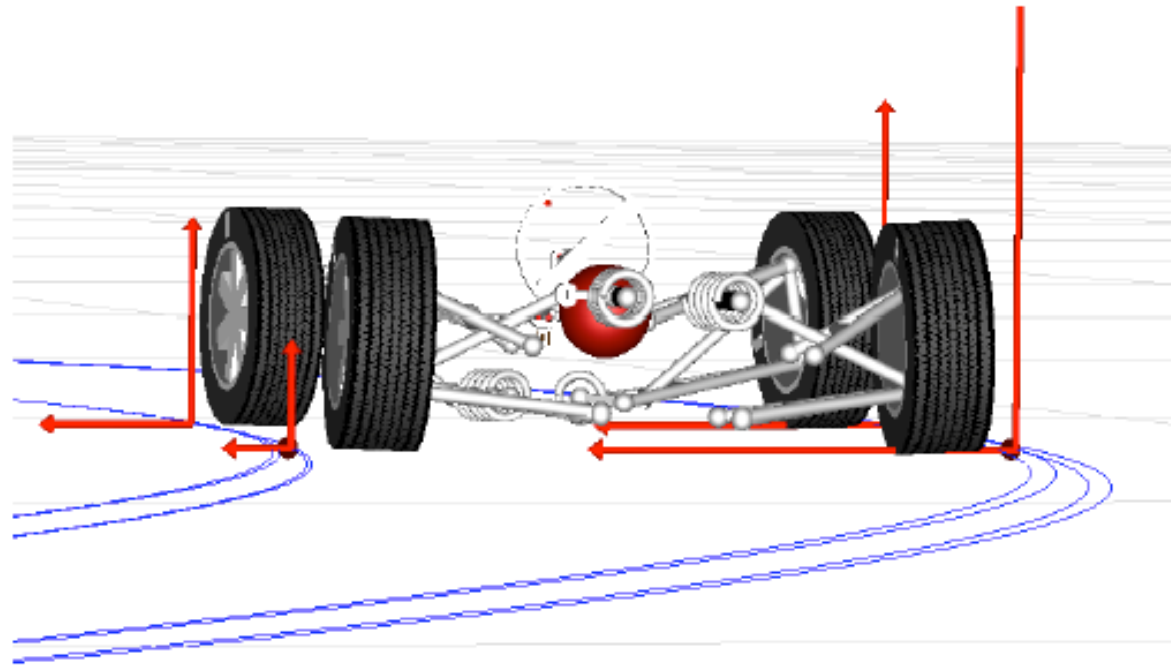
The screenshot shows an IDE with several windows. The main window displays code for 'TypeAnalysis.jrag'. A project explorer on the right shows a tree structure with 'NameResolution.jrag' expanded. Five numbered callouts (1-5) are overlaid on the image, connected by arrows to specific elements in the code and the project explorer. Callout 1 points to a 'decl' entry in the project explorer. Callout 2 points to a 'syn lazy decl AccessDecl()' line in the code. Callout 3 points to a 'lookup(name)' function call in the code. Callout 4 points to a 'lookup(name)' function call in the code. Callout 5 points to a 'superClass' entry in the project explorer.

JModelica.org



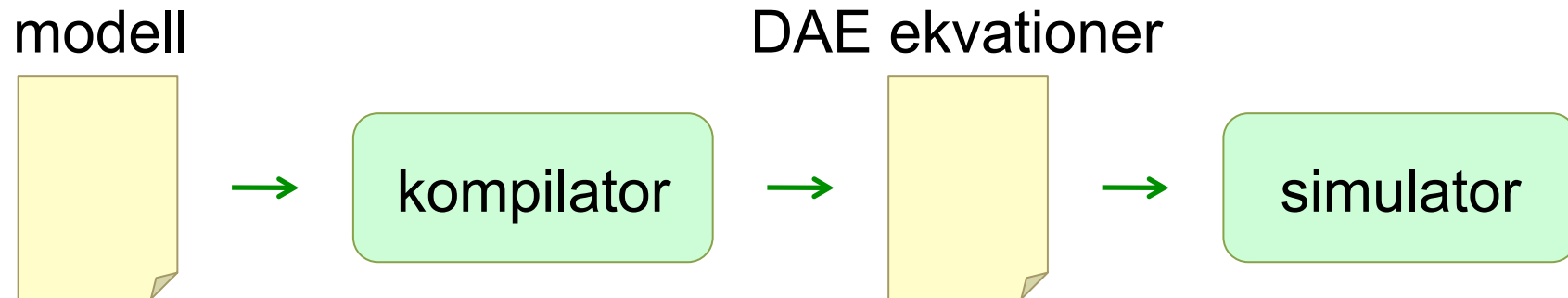
Kompilator och IDE utvecklade med hjälp av JastAdd





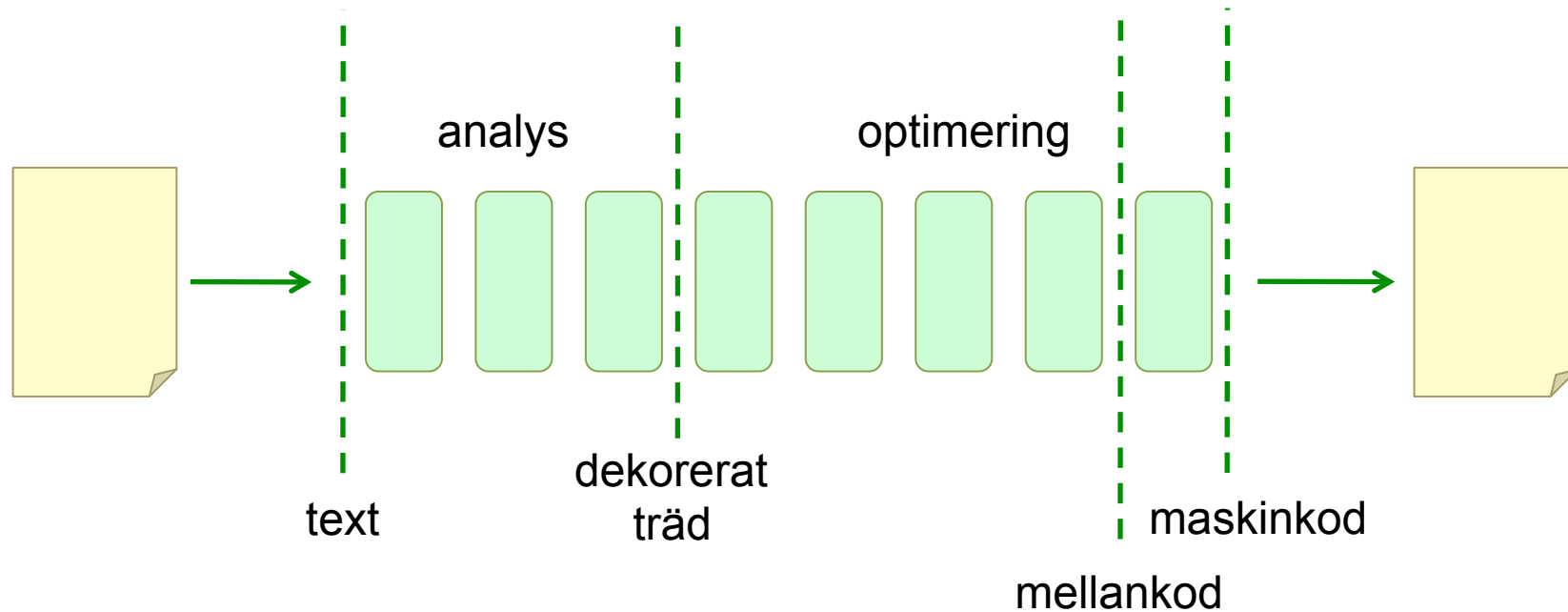
Exempel på modellering och optimering av fysikaliskt system med hjälp av JModelica.

JModelica-kompilatorn



Kompilorteknik

generell teknik för många tillämpningar



Fortsättningskurser

EDAN65, Kompilorteknik

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.