Velkommen til INF5110 - Kompilatorteknikk

- Kursansvarlige:
  - Stein Krogdahl [steink@ifi.uio.no]
  - Birger Møller-Pedersen [birger@ifi.uio.no]
  - Henning Berg (oblig-ansvarlig) [hennb@ifi.uio.no]
- Kursområdet: www.uio.no/studier/emner/matnat/ifi/INF5110/v14
  - Plan over forelesningene, pensum etc. etter hvert som det blir klart
  - Diverse beskjeder

Vårens opplegg 2014

- Lærebok etc:
  - Kenneth C. Louden: ”Compiler Construction, Principles and Practice”
  - Antakeligvis også noe stoff fra andre kilder
- Skal i gjennomsnitt være tre timer undervisning pr. uke
- Forelesning og oppgaveløsning i passelig blanding
  - Skal grovt sett følge opplegget fra 2013
- Obligatoriske oppgaver:
  - Oblig 1: Legges ut ca midten av februar, frist midten av mars
  - Oblig 2: Legges ut ca starten av april, frist starten av mai
  - Gruppearbeid: 2 og 2 sammen og det er tillatt å jobbe alene
  - Kan søke om å få arbeide 3 sammen
- Eksamen: 3. juni kl. 14:30 (4 timer). Har pleid å være skriftlig
- Foiler
  - Legges ut på nett

Hvorfor bør man vite noe om kompilatorer?

- De færreste av dere kommer til å lage f.eks. en Java kompilator!!
- Men …

- Mange applikasjoner vil inneholde håndtering av input som kan sees som enkle former for programmer i et programmeringsspråk, og kan greiest behandles som det
- En del kommer til å lage eller modifisere kodegeneratorer av forskjellige typer, f.eks. enkel produksjon av kode basert på modeller (f.eks. UML)
- Eksperimentere med nye språkbegreper, som f.eks.
  - **Aspekt-orientering**: håndtere visse aspekter av et program ved spesiell aspekt-kode (krever aspekt-kompilator)
  - **Traits:** (samlinger av metoder som klasser kan få uavhengig av klassehierarki), generiske traits
  - **Package Templates**: Generelle samlinger av klasser som kan hentes inn i programmer, og samtidig tilpasses lokale behov.
  - **Domene-spesifikke språk**: Språk som er rette mot spesielle anvendelser

Domene-spesifikke språk

- MoSiS: EU prosjekt (IKT-Norge, SINTEF, ...)
  - TCL: domene-språk for fremtidens togkontrollsystemer
  - CVL: standardisert språk for å beskrive variasjoner mellom produkter i en produktfamilie
  - Kopling av disse for å beskrive variasjoner (og likheter) mellom stasjoner

- De skal defineres ordentlig (INF 3110 - Programmeringsspråk)
- Og implementeres (INF 5110 - Kompilatorteknikk)

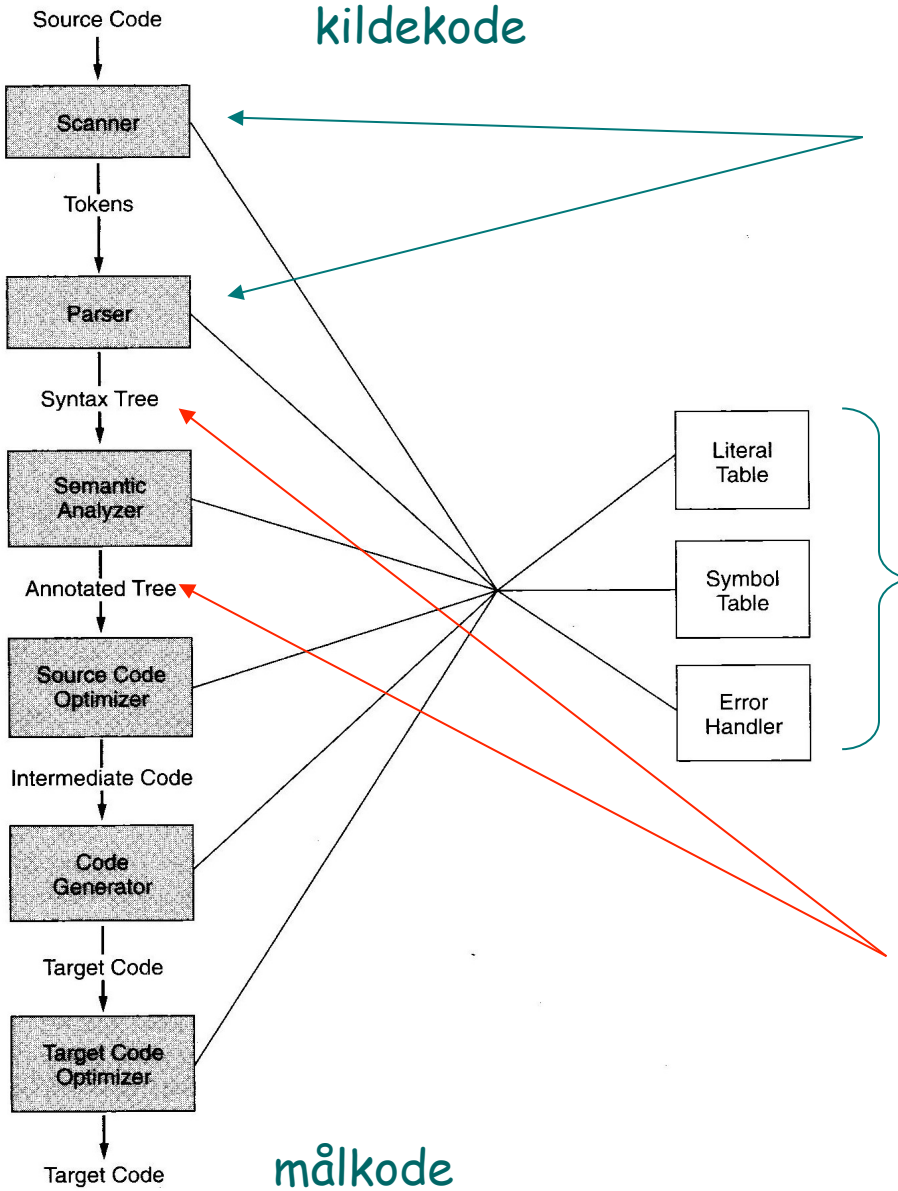
- Metamodeller i stedet for/ i tillegg til grammatikker

Dagens tekst

- Kapittel 1: En oversikt over
  - Hvordan en kompilator typisk er bygget opp
  - Hvilke teknikker og lagringsformer som typisk brukes i de forskjellige deler
  - Litt om notasjon for kompilering, bootstrapping, etc.

  - Litt om metamodeller, men det kommer mer i en senere forelesning

Bokens oversikt over en typisk kompilator



kildekode

"faser" under kompilering

Sentrale tabeller og mekanismer

Mellomformer av programmet

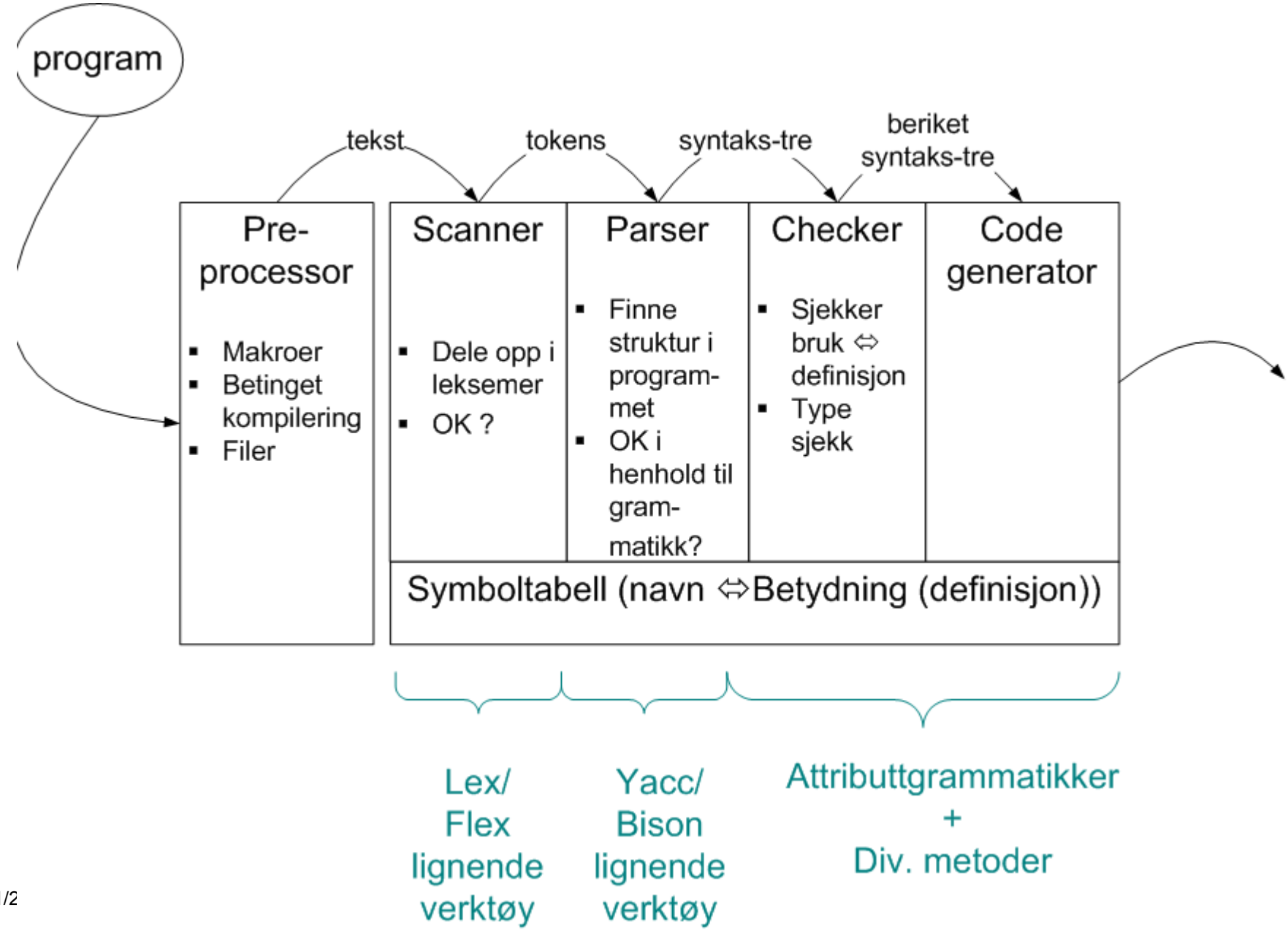
målkode

Fase:
Logisk del av kompilator

Gjennomløp ("Pass"):
Gjennomgang av teksten/treet.
Men: Er mindre viktig enn før siden treet kan holds i interlageret.

INF 5110 - 2014

Anatomien til en kompilator - I



Pre-prosessor

- Enten eget program eller bygget inn i kompilator
- Henter f.eks. inn filer

```
#include  <filnavn>
```

- Betinget kompilering

```
#vardef  #a = 5; #c = #a + 1
    - - -
 #if (#a < #b )
    - - -
 #else
    - - -
 #endif
```

- ”Makroer”, definisjon

```
#makrodef hentdata (#1, #2)
    - - - - #1 - - - - - - -
    - - #2 - - - #1 - - -
#enddef
```

- Bruk av makroer (”ekspansjon”)

```
#hentdata(kari, per)  --->   - - - - kari - - - - - - -
                             - - per - - - kari - - -
```

- Passer f.eks. til å utvide språket med nye konstruksjoner
- PROBLEM: Ofte tull med linjenummer og med syntaktiske konstruksjoner, bygges derfor helst inn

Scanner

- Deler opp programmet i tokens
- Fjerner kommentarer, blanke, linjeskift ()
- Teori: Tilstandsmaskiner, automater, regulære språk, m.m.

```
a[index] = 4 + 2
```

Leksem	Token	
a	identifier	2
[	left bracket	
index	identifier	21
]	right bracket	
=	assignment	
4	number	4
+	plus sign	
2	number	2

0	
1	
2	a
	.
	.
21	index
22	

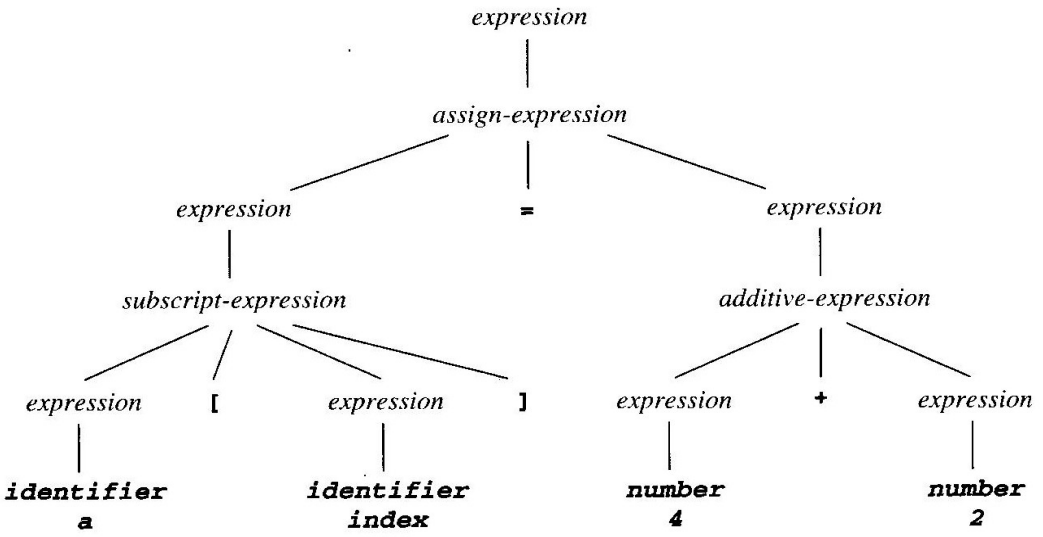
Tilsvarende for tekstkonstanter

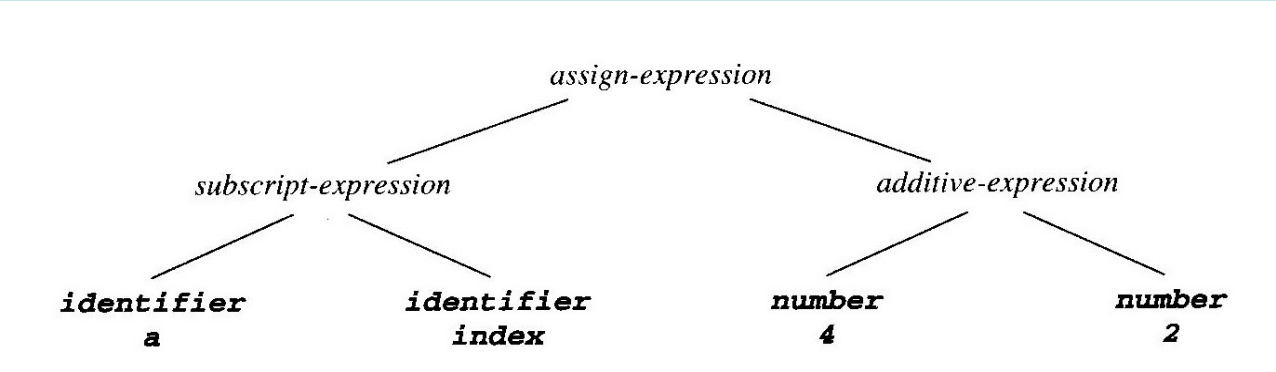
Parser

parserings-tre
(syntaks-tre)

resultat av parsering

```
a[index] = 4 + 2
```





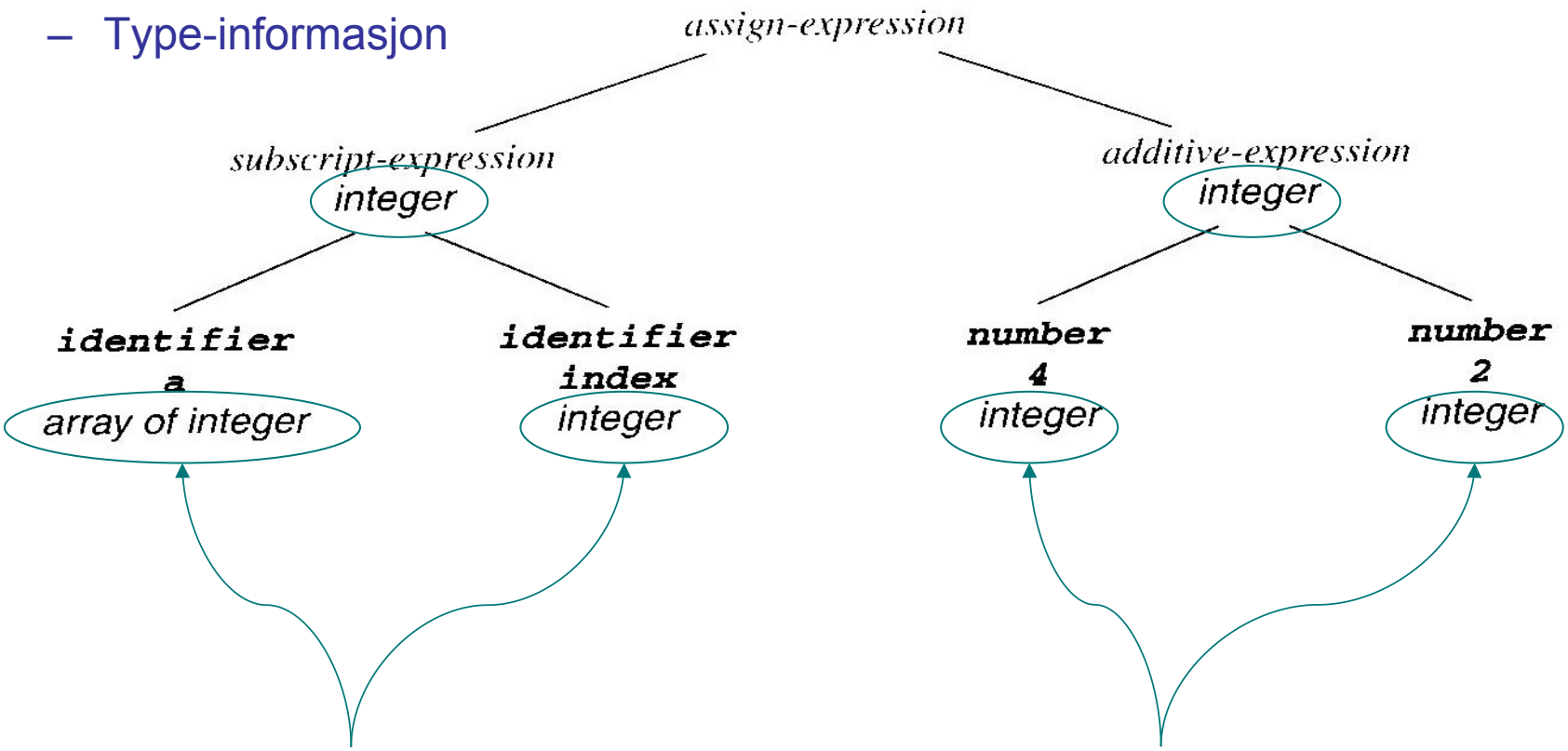
abstrakt
syntaks-tre

”syntaktisk
sukker”
fjernet

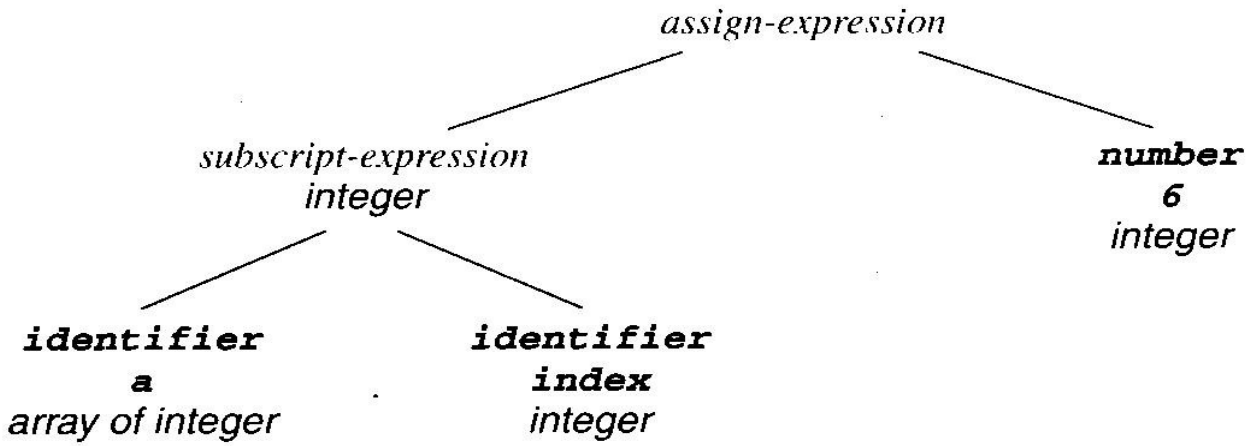
INF 5110 - 2014

Resultat av semantisk analyse

- Et beriket eller dekorert abstrakt syntaks-tre. Har satt inn:
  - Bindinger til deklarasjoner
  - Type-informasjon



Optimalisering på kildekode nivå




```
t = 4 + 2
a[index] = t
```

opprindelig

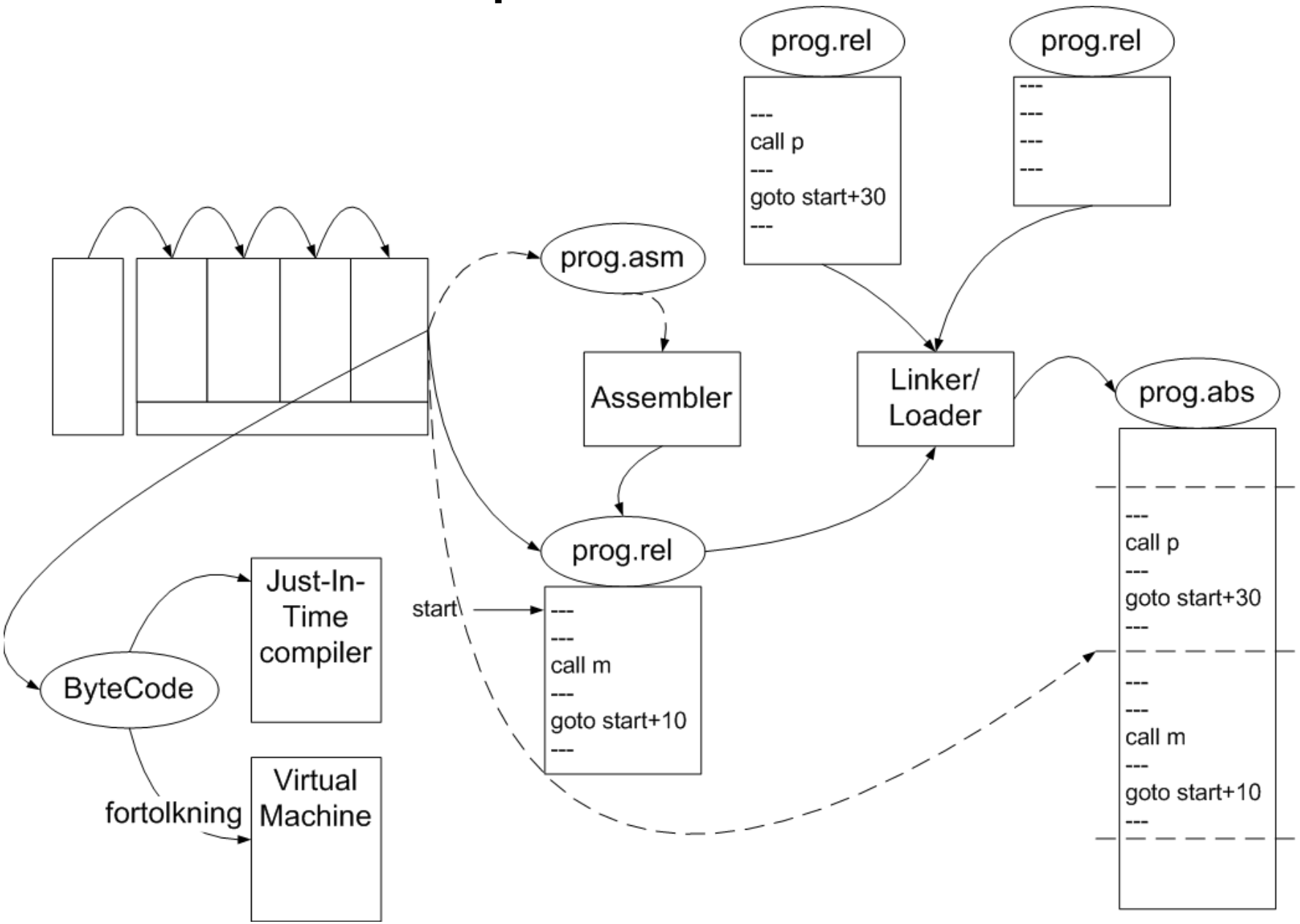
```
t = 6
a[index] = t
```

ett steg optimalisering

```
a[index] = 6
```

nok et steg

Anatomien til en kompilator - II



Kodegenerering og optimalisering

Vanskelig å automatisere (basert på formell beskrivelse av språk og maskin)

- Resultat av rett fram kodegenerering

```
MOV   R0, index   ;; value of index -> R0
MUL   R0, 2       ;; double value in R0
MOV   R1, &a      ;; address of a -> R1
ADD   R1, R0      ;; add R0 to R1
MOV   *R1, 6      ;; constant 6 -> address in R1
```

Beregn adressen til a[index]

- Etter optimalisering på mål-kode nivå

```
MOV  R0, index    ;; value of index -> R0
SHL  R0           ;; double value in R0
MOV  &a[R0], 6    ;; constant 6 -> address a + R0
```

Shifter istedet for å doble

Bruker maskinens adresseringsmekanismer fullt ut

Diverse begreper og problemstillinger

- ”Front-end” og ”Back-end”: Tilsvarer oftest ”analyse” og ”syntese”
- Hvordan behandles separatkompilering av programbiter?
- Hvordan behandler kompilatoren feil i programmet?
- Hvordan er data administrert under utførelsen?
  - Statisk, stakk, heap
- Språk som kan oversettes i ett gjennomløp
  - F.eks. C og Pascal: Deklarasjoner kan ikke brukes før de er nevnt i prog.teksten
  - Er ikke så viktig lenger, pga. mye intern lagerplass
- Feilfinnings-hjelpemidler (”debuggers”)
  - Kan arbeide interaktivt med en programutførelse vha. variablenavn etc.
  - Kan legge inn ”breakpoints”

Oversettelse og interpretering

- Oversettelse
  - Man oversetter til maskinkoden for en gitt maskin
  - Maskinkoden leveres i forkjellige former fra kompilatoren:
    - Ferdig utførbar binær kode (må alltid til denne formen før utførelse)
    - Relokerbar kode, kan settes sammen med andre relokerbare biter
    - Tekstlig assembler-kode, må prosesseres av assembler

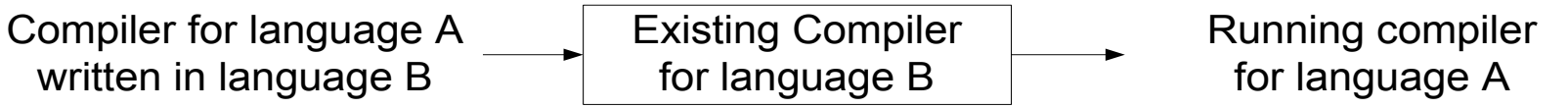
- Full interpretering
  - Utføres direkte fra programteksten/syntakstre
  - Brukes mest for kommandospråk til operativsystem etc.
  - Utførelse typisk 10 – 100 ganger saktere enn ved full oversettelse

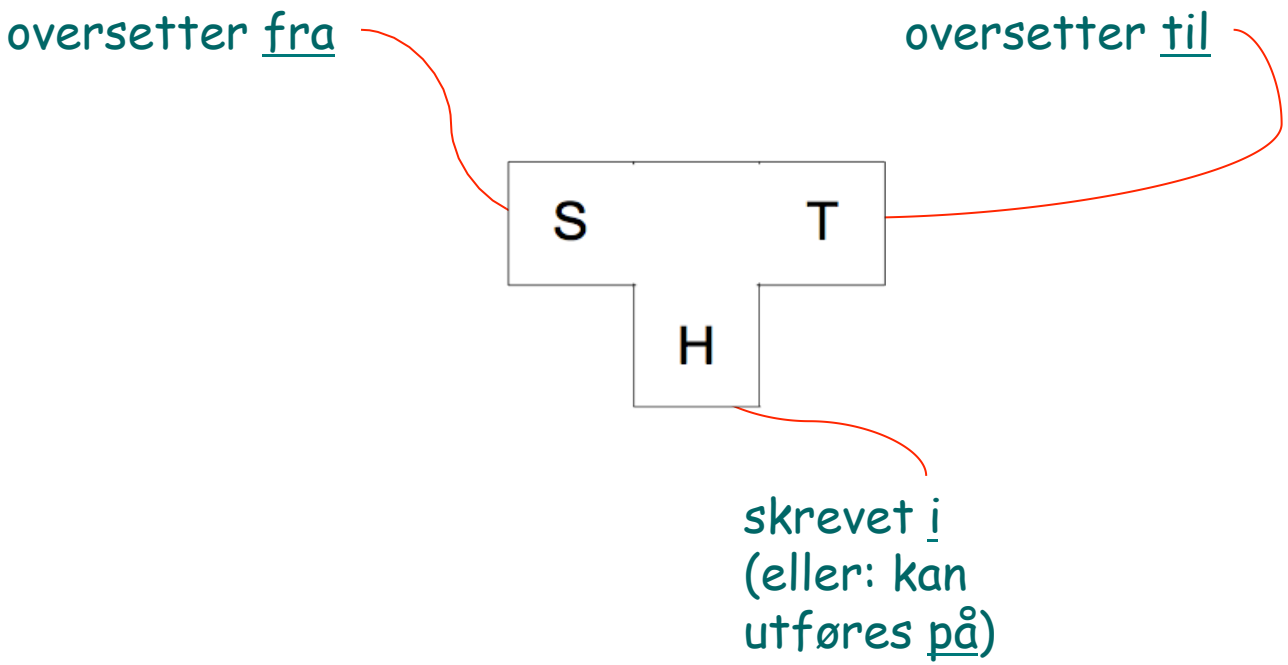
- Oversettelse til mellomkode som interpreteres
  - Brukes for Java (class-filer), og mye for Smalltalk
  - Mellomkoden er valgt slik at den er grei å utføre (byte-kode for Java)
  - Utføres av en enkel interpreter (Java: Java Virtual Machine)
  - Går typisk 3 - 30 ganger så sent som direkte utførelse
  - Dog: I de fleste moderne Java-systemer oversettes byte-koden til maskinkode umiddelbart før den utføres (JIT, Just-In-Time kompilering).

Senere utviklinginnen kompilatorer etc.

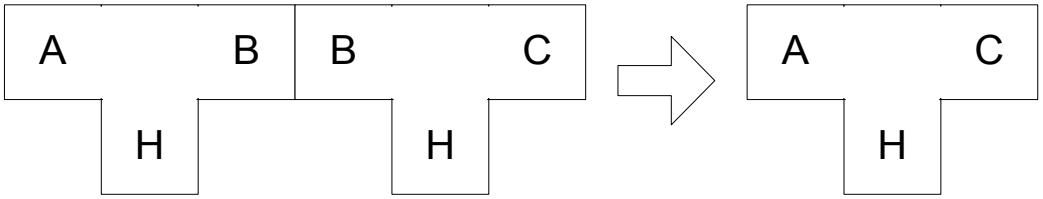
- Lager (spesielt intern-lager) er blitt billig, og dermed stort
  - Man kan ha hele programmer inne i maskinen under kompilering
  - 200 byte pr. linje gir 50 000 linjer på 10 Mbyte
  - Lagrene ble store nok til dette rundt 1990
- Objektorienterte språk er blitt meget populære
  - Spesielle teknikker for optimalisering mm. blir da viktige
- Java tilbyr spesiell form for utførelse:
  - Kompilator lager ”byte-kode”, som også har alle programmets navn etc.
  - Programdeler kan hentes under utførelse, og kobles inn i programmet
- Input kan være figurer, skjemaer etc. (f.eks. UML)
  - Metamodeller i tillegg til grammatikker

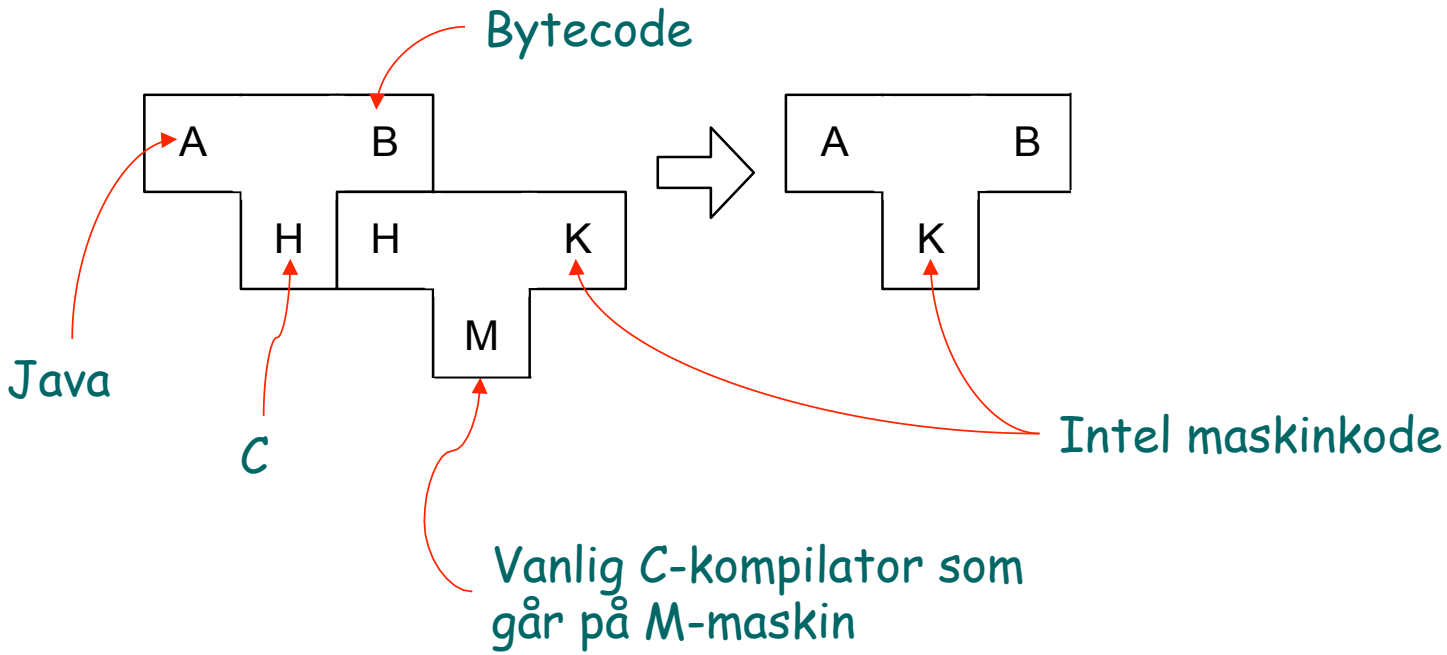
Bootstrapping and porting



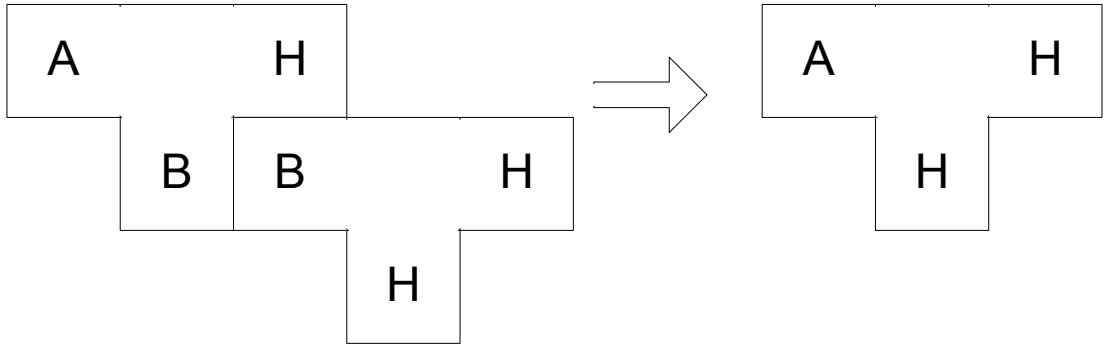


To sammensetningsoperasjoner






Compiler for language A
written in language B
Existing Compiler
for language B
Running compiler
for language A


A
H
B
B
H
H
A
H
H

Bootstrapping

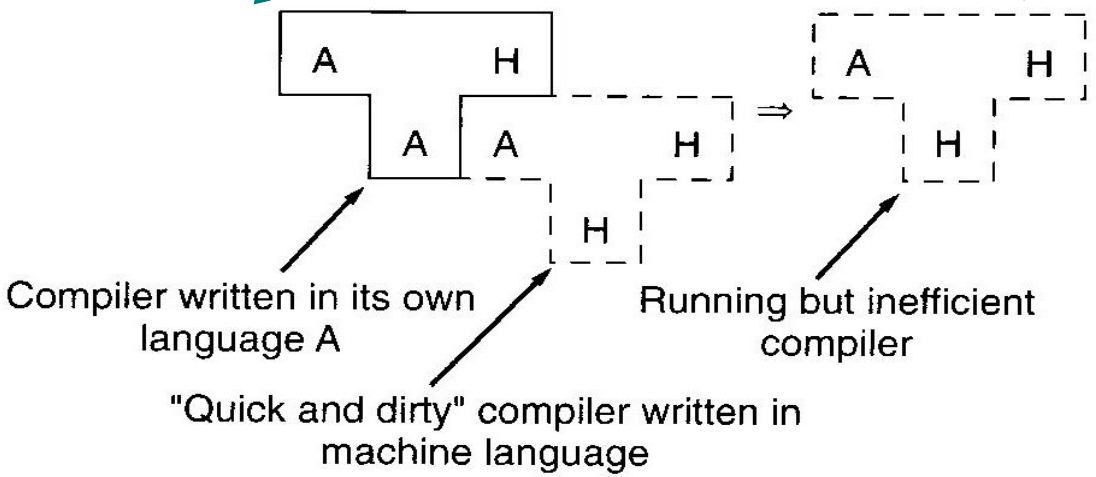
Lage en kompilator som er skrevet I eget språk, går fort og lager god kode

Steg 1

Skrevet i en begrenset del av A

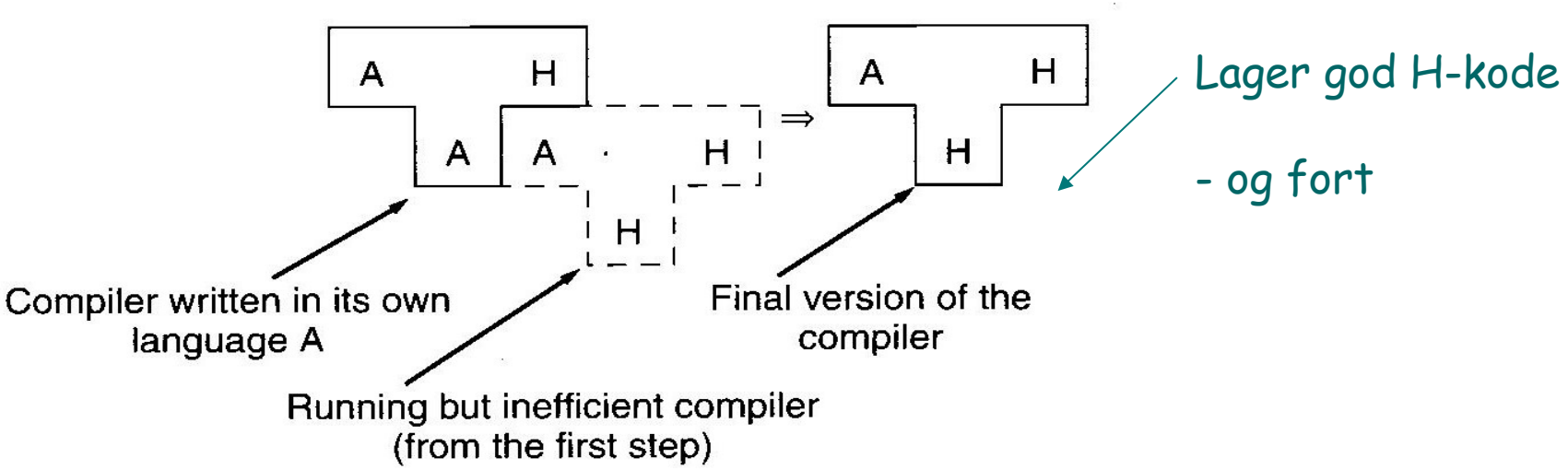
Lager god H-kode

- men sakte



Bootstrapping

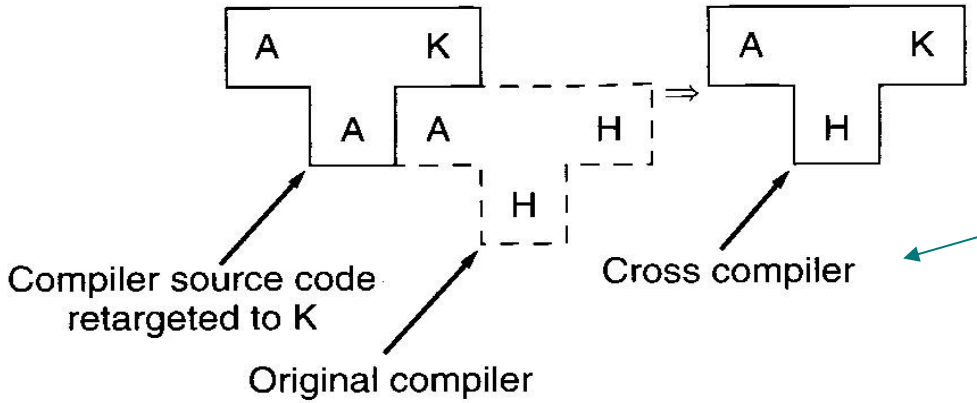
Steg 2



Krysskompilering

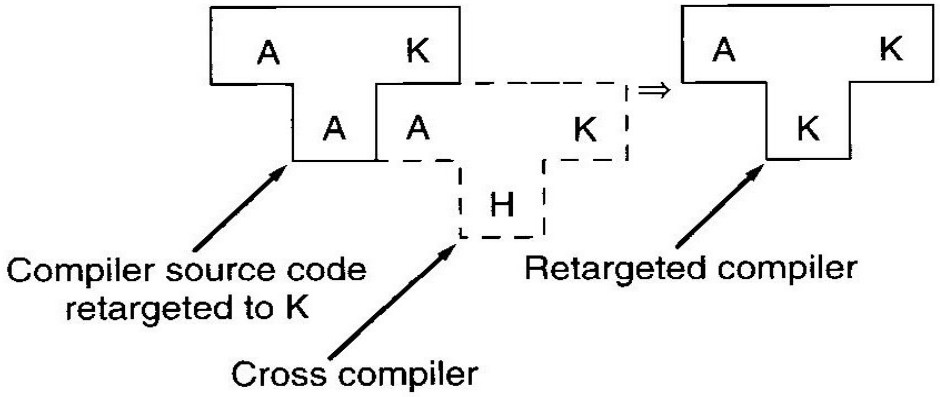
- Har: A kompilator som oversetter til H-maskinkode
- Ønsker: A-kompilator som oversetter til K-maskin kode

Steg 1: Skriv kompilator slik at den produserer K-kode (f.eks. vha ny back-end)



går på H-maskin, produserer K-kode (kryss-kompilator)

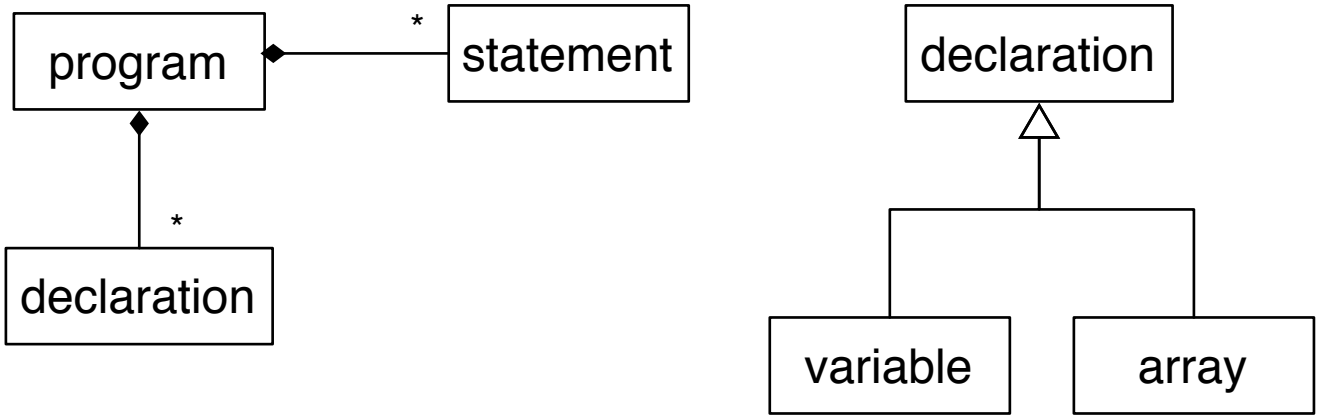
Steg 2: Oversetter den nye kompilatoren til K-kode. Gjøres på en H-maskin vha krysskompilatoren



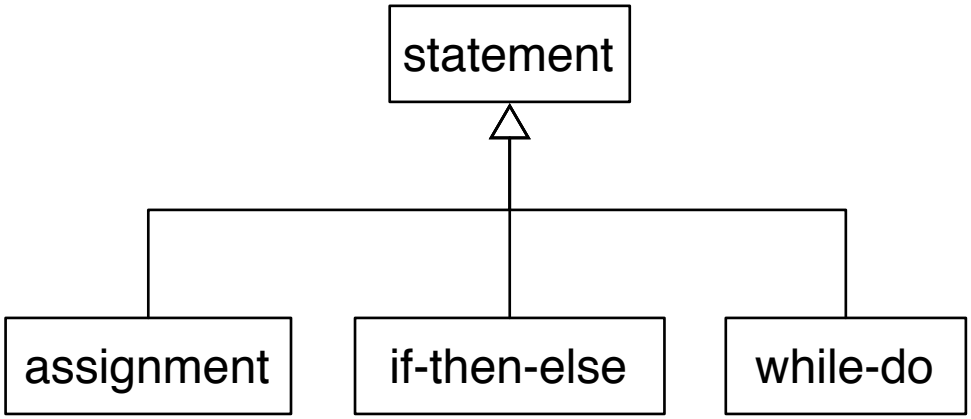
INF 5110 - 2014

Metamodels

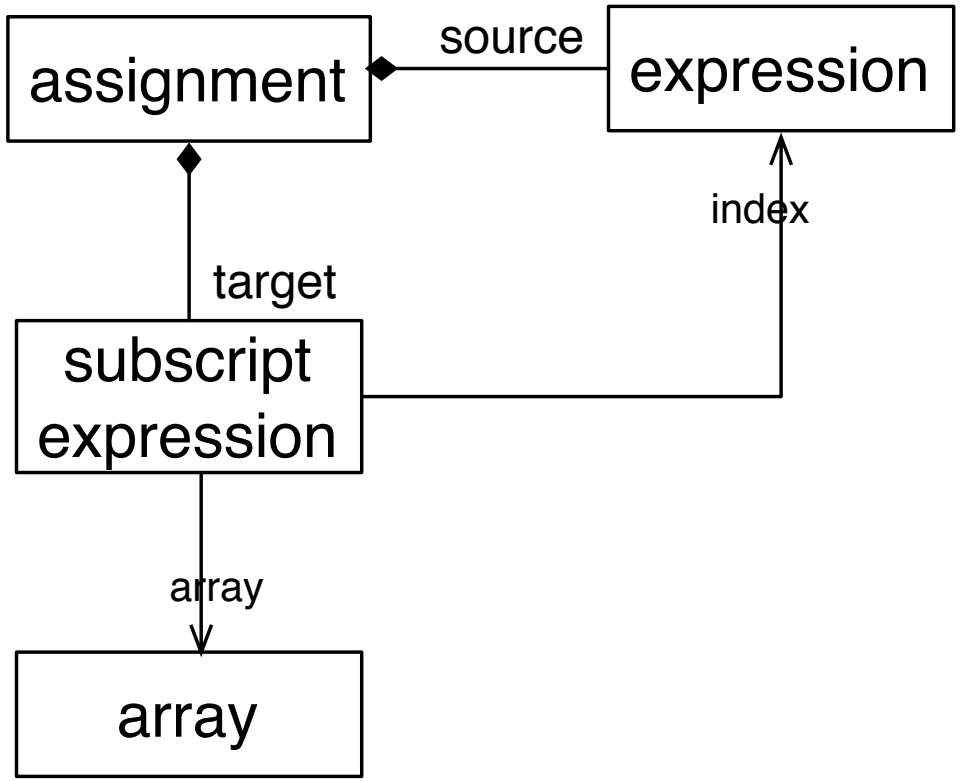
- Alternative to grammars and syntax trees
- Object model representing the program (*not* the execution)



<statement> → <assignment> | <if-then-else> | <while-do>



```
a[index] = 4 + 2
```


assignment
source
expression
index
target
subscript
expression
array
array