Velkommen til INF5110 - Kompilatorteknikk

- Kursansvarlige:
  - Arne Maus [arnem@ifi.uio.no]
  - Birger Møller-Pedersen [birger@ifi.uio.no]
  - Sven-Jørgen Karlsen ('gruppelærer') [svenjok@student.matnat.uio.no]
  - Stein Krogdahl (støtteperson)
- Kursområdet: www.ifi.uio.no/inf5110 → vår 2006
  - Plan over forelesningene, pensum etc. etter hvert som det blir klart
  - Diverse beskjeder
- Vår forhold til kompilatorteknikk etc.:
  - AM har undervist INF1000, så ham kjenner dere!
  - SK og BMP laget en Simula-kompilator sammen, ca 1980 (på NR)
  - SK har undervist dette kurset en del ganger
  - Alle har arbeidet mye med programmeringsspråk (spesielt OO-språk) og implementasjon av forskjellige språkmekanismer

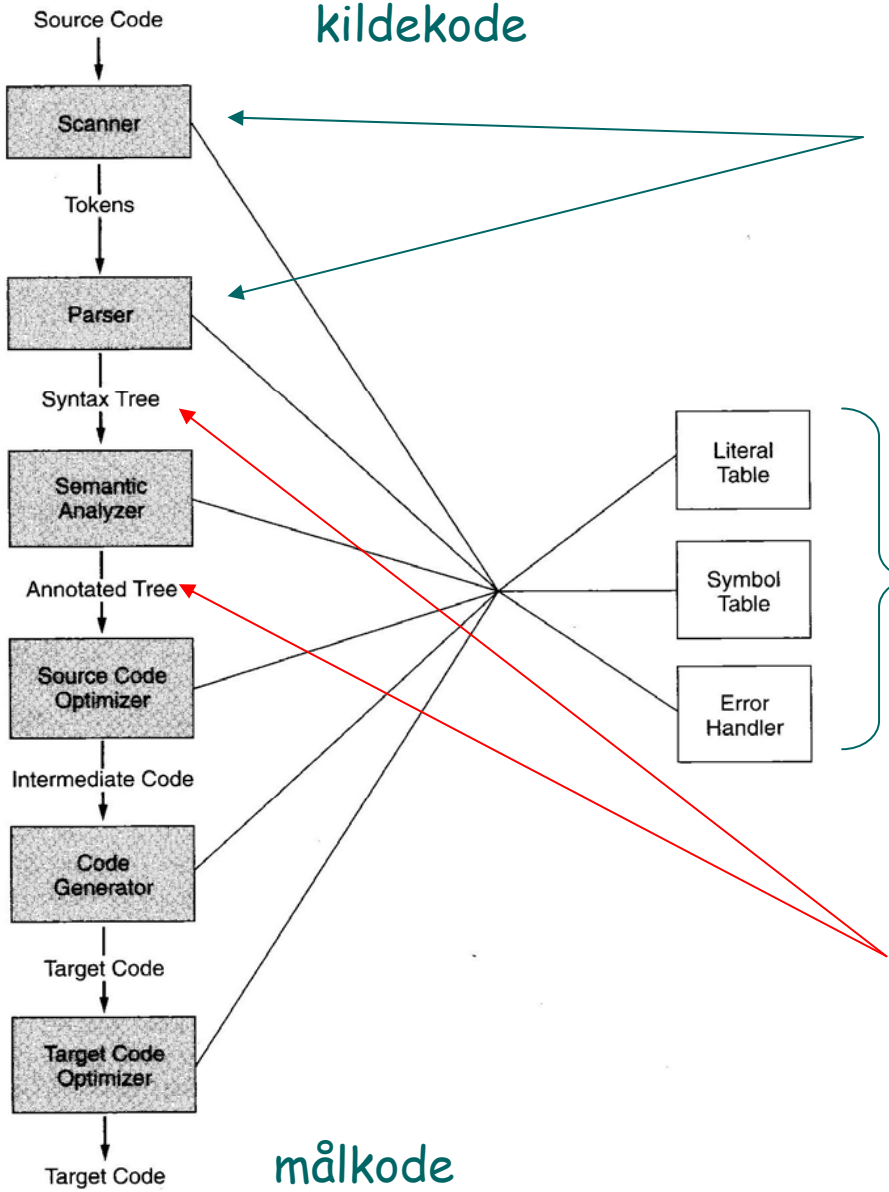
Vårens opplegg

- 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
  - Men vi starter ut med fire timer pr. uke
- Forelesning og oppgaveløsning i passelig blanding
  - Skal grovt sett følge opplegget fra 2005, men antakeligvis noen justeringer
- Oppgaver etc. på veien:
  - Minst en obligatorisk oppgave (uten karakter)
- Eksamen: Har pleid å være muntlig, men skriftlig hvis mange
- Foiler: Blir kopiert opp på papir
  - Deles ut på forelesningen, og kan fåes på senere forelesninger;
  - Legges ut på nett
- Hva har dere av kurs?
  - INF 2100 ? (Skriving av en enkel kompilator)
  - INF 3/4110 (IN 211) ? (Programmeringsspråk)

Dagens tekst

- Kapittel 1: En oversikt over
  - Hvordan en kompilator typisk er delt opp
  - Hvilke teknikker og lagringsformer som typisk brukes i de forskjellige deler
  - Hva er de typiske omgivelser til en kompilator
  - Litt om notasjon for ”bootstrapping” etc.

Bokens oversikt over en typisk kompilator



kildekode

"faser" under kompilering

Sentrale tabeller og mekanismer

Mellomformer av programmet

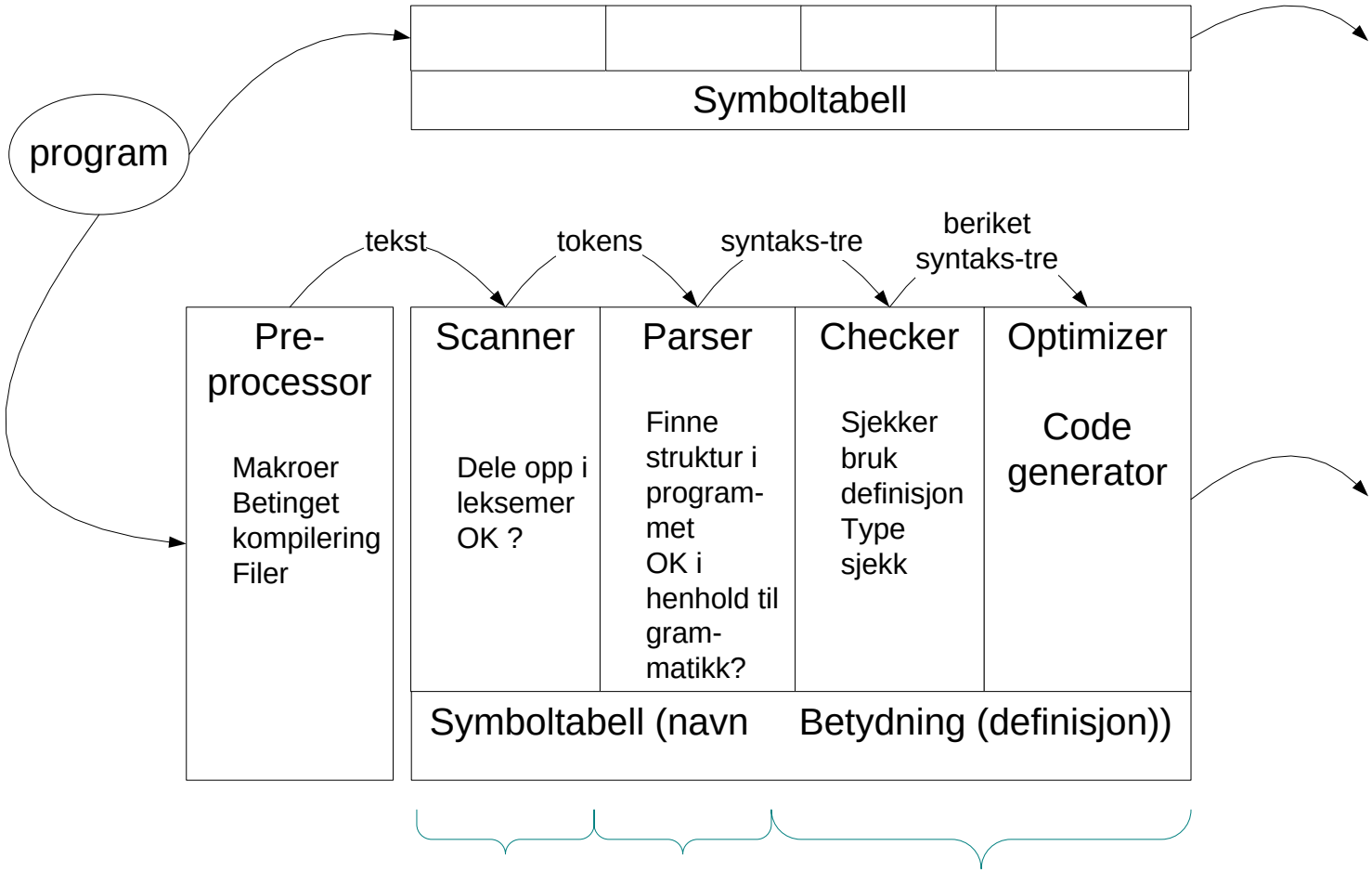
målkode

Fase:
Logisk del av kompilator

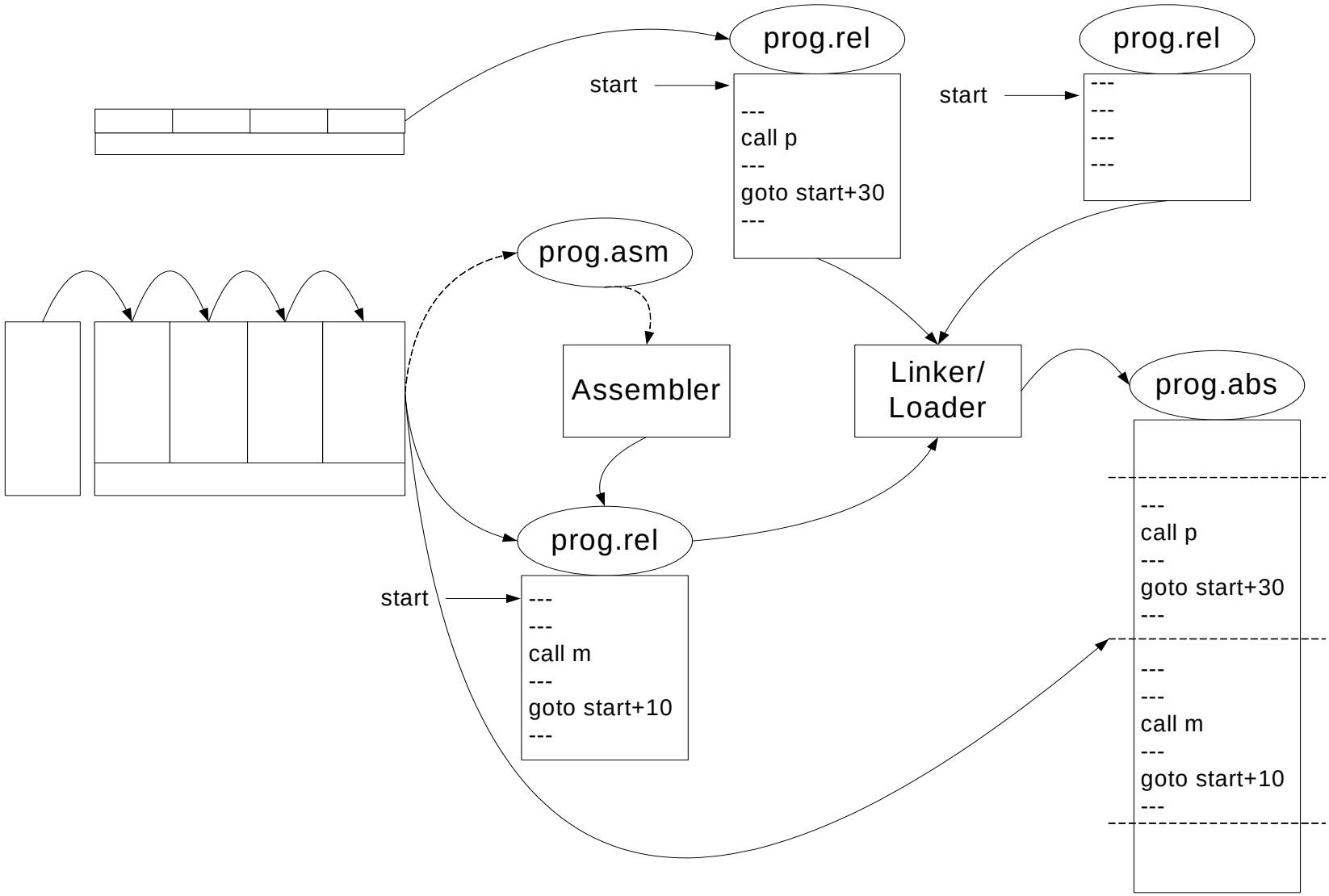
Gjennomløp ("Pass"):
Gjennomgang av teksten/treet

INF 5110 - 2006

Anatomien til en kompilator - I



Anatomien til en kompilator - II



Forbehandling

- Er enten pre-prosessor eller er bygget inn i kompilator
- Hent 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, 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

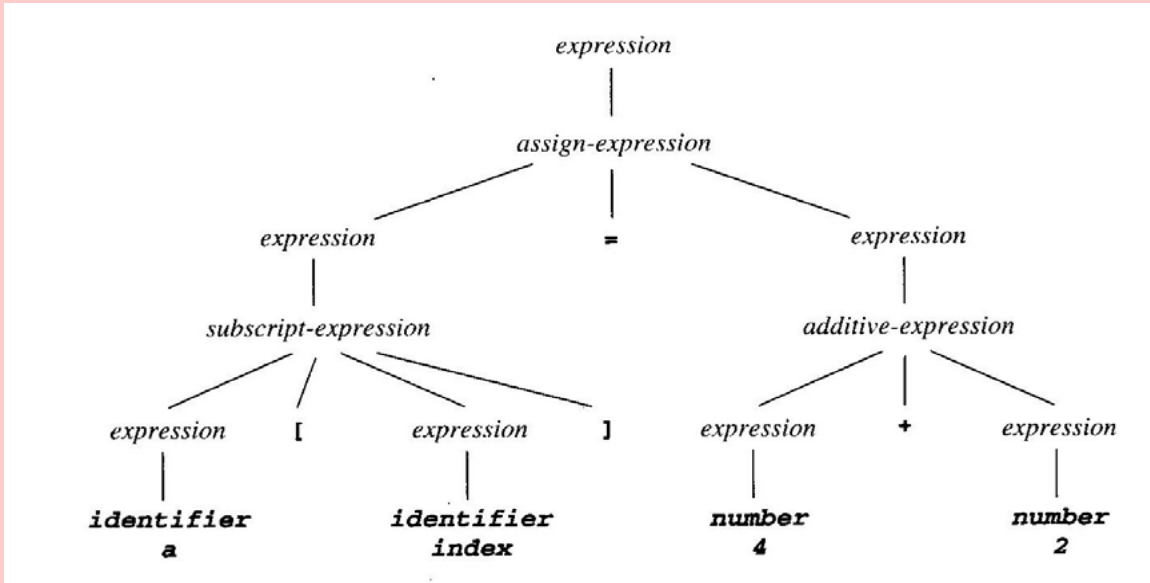
0	
1	
2	a
	. .
21	index
22	

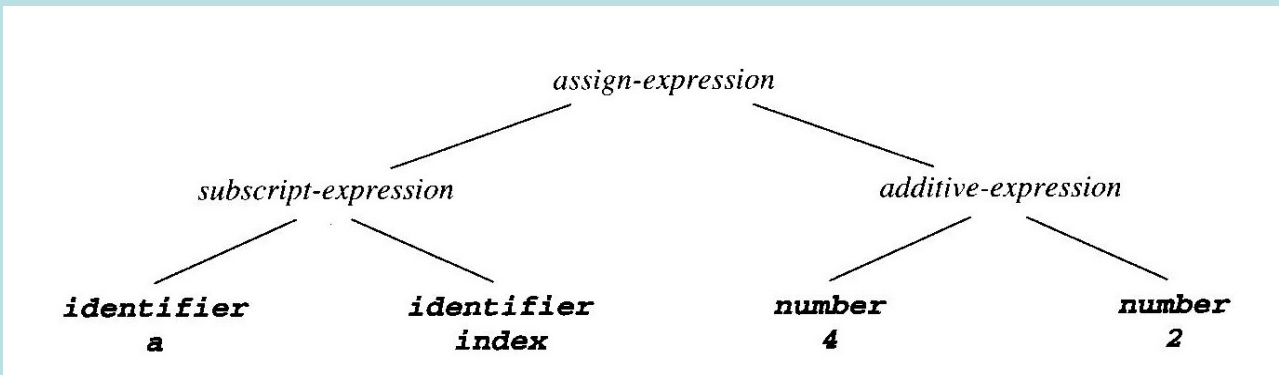
Tilsvarende for tekstkonstanter

Parser

parserings-tre
(syntaks-tre)

resultat av parsering





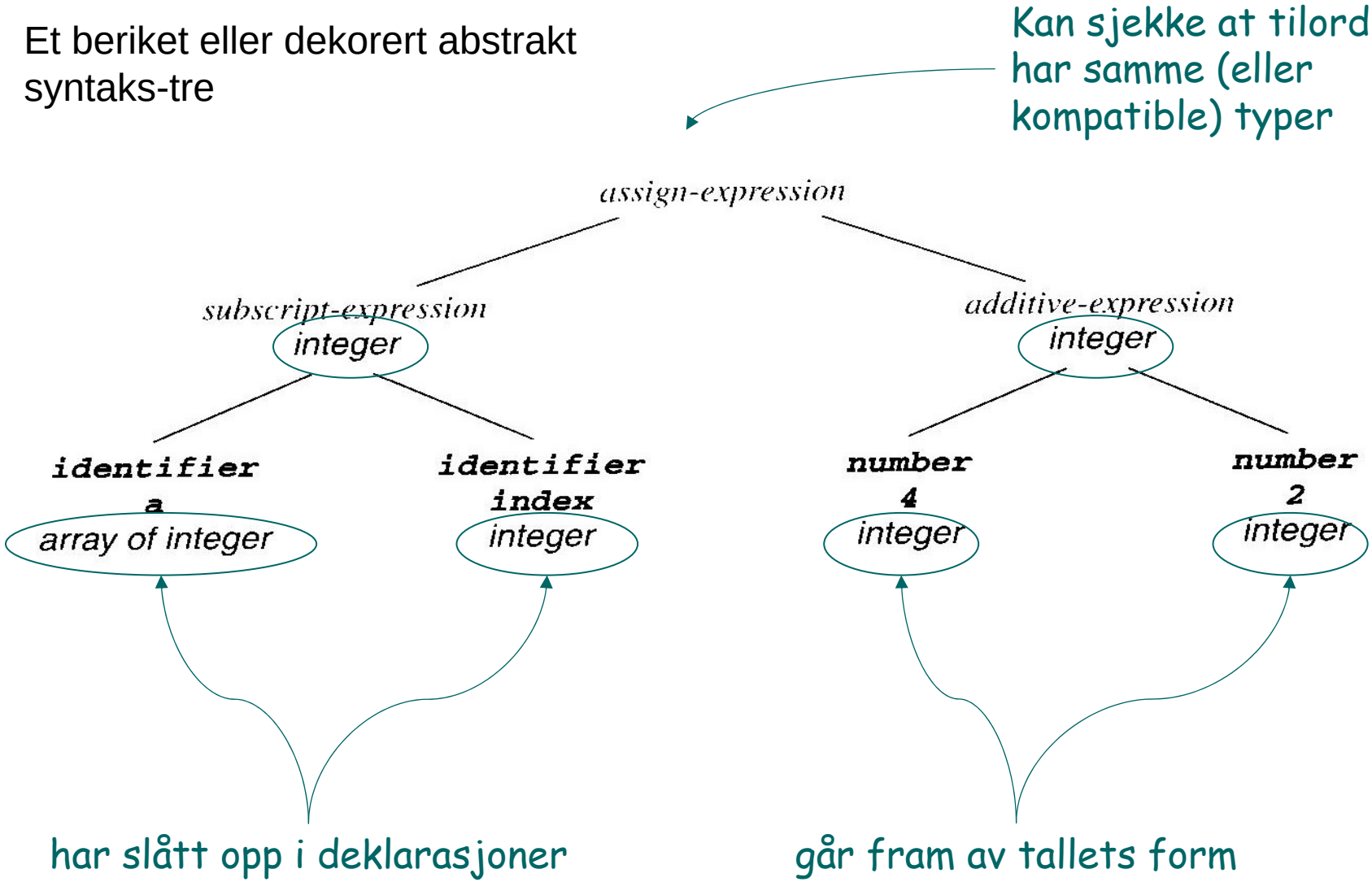
abstrakt
syntaks-tre

syntaktisk
sukker
fjernet

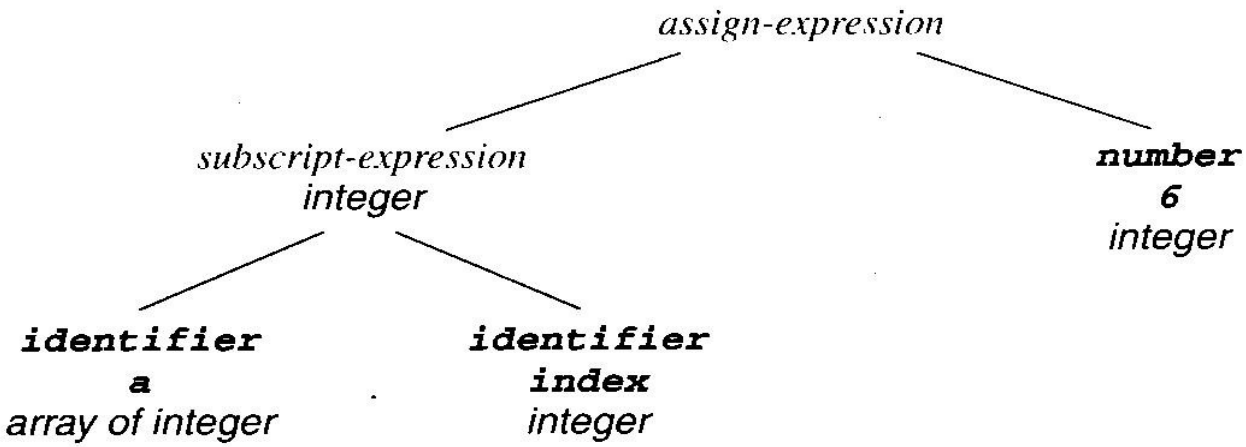
INF 5110 - 2006

Resultat av semantisk analyse

- Et beriket eller dekorert abstrakt syntaks-tre



Optimalisering på kildekode nivå



Tre-adresse-kode

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

opprindelig

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

ett steg optimalisering

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

nok et steg

Kodegenerering

Vanskelig å automatisere (basert på formell beskrivelse va 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

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, også ved gjentakelse
  - 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 ”interpret(ator?)” (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).

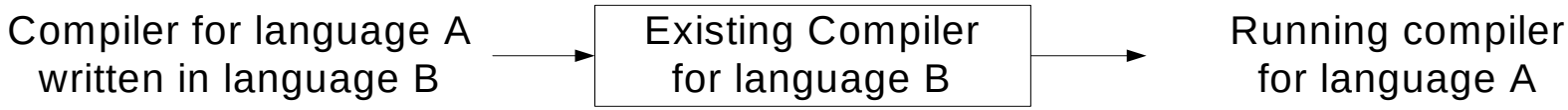
Diverse begreper og problemstillinger

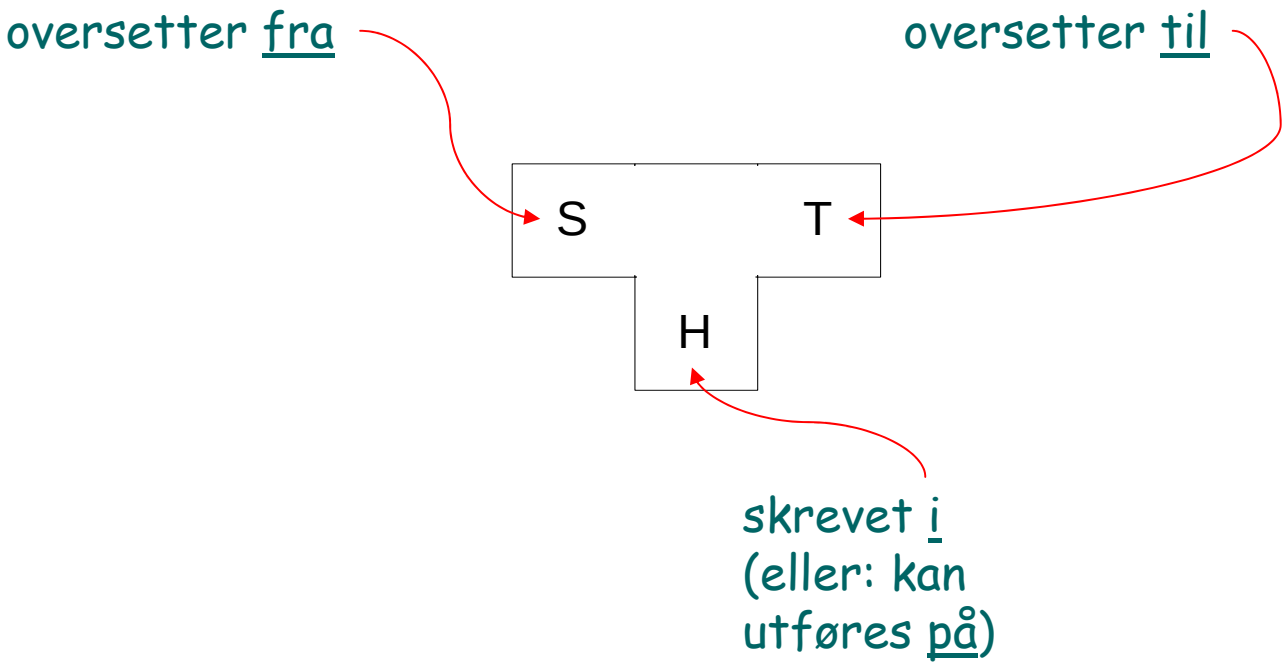
- ”Front-end” og ”Back-end”: Tilsvarer oftest ”analyse” og ”syntese”
- Hvordan behandles separatkompilering av programbiter?
- Hvordan behandler kompilatoren feil i programmet?
- Hvordan er dataene administrert under utførelsen?
  - Statisk, stakk, heap
- Språk som kan oversettes i ett gjennomløp
  - 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”
- Belastningsprofiler: Hvor mye tid er brukt i hvilke deler under kjøring?
- Versjons-håndterere etc.

Nyere ting innen kompilatorer etc.

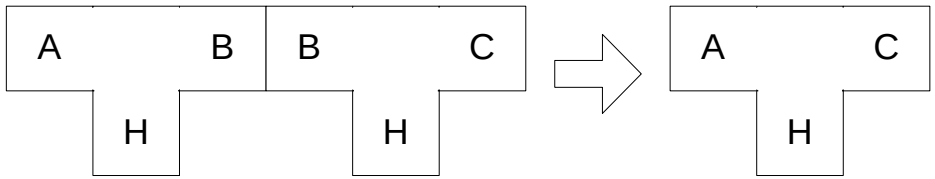
- Lager (spesielt intern-lager) er blitt billig, og dermed stort
  - Man kan ha hele programmer inne i maskinen under kompilering
  - 200 byte pr. linje gir 50 000 linjer på 10 Mbyte
- Skjedd mye om å utnytte flere prosessorer til store beregninger
  - Parallelliserende kompilatorer, som kan gies hint om hva som er lurt
- 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.
  - Denne interpreteres direkte av en JVM eller JIT-kompileres
  - Programdeler kan hentes under utførelse, og kobles inn i programmet
  - Kan også lett hente slike programbiter over nettet
- Prosessor-utviklingen
- Input kan være figurer, skjemaer etc. (f.eks. UML)
  - Metamodeller contra grammatikker

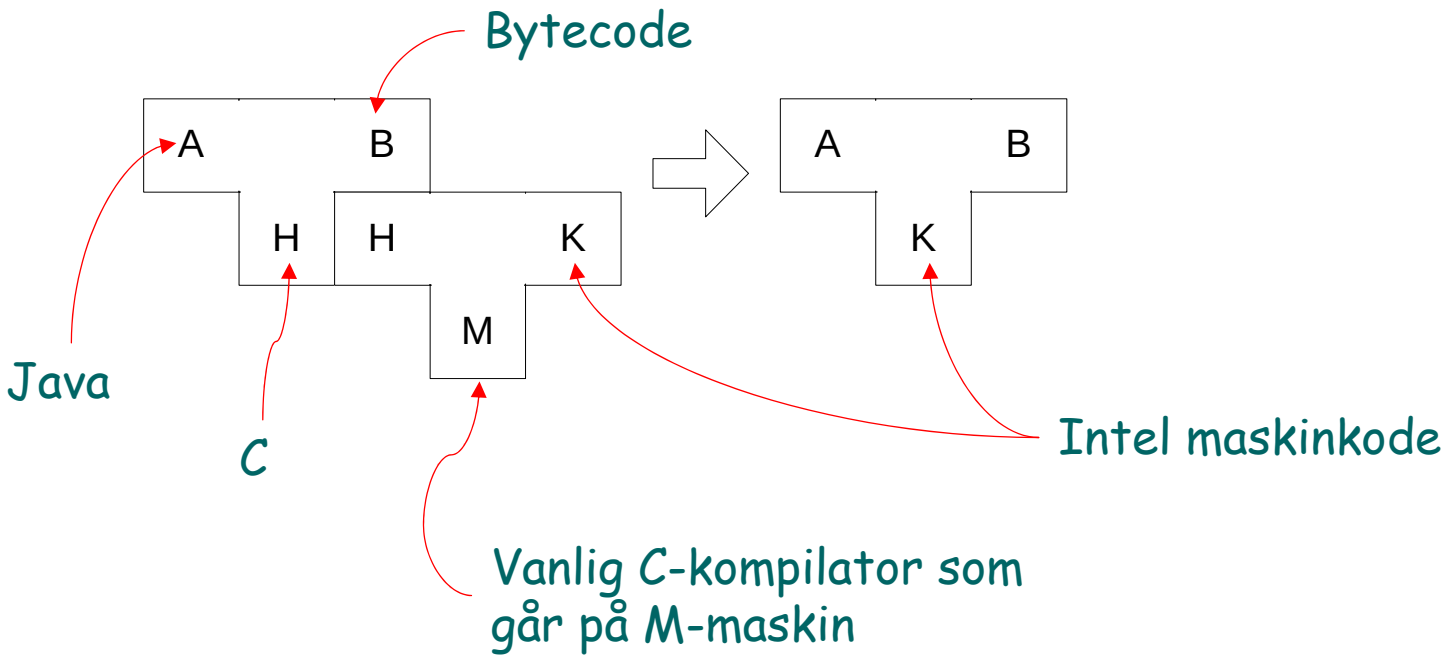
Bootstrapping and porting





To sammensetningsoperasjoner



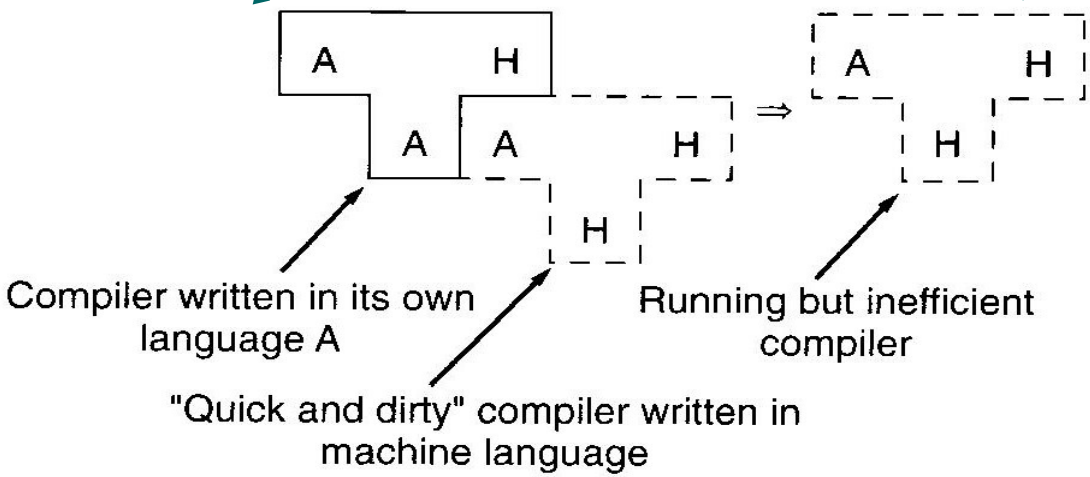


Bootstrapping: Step 1

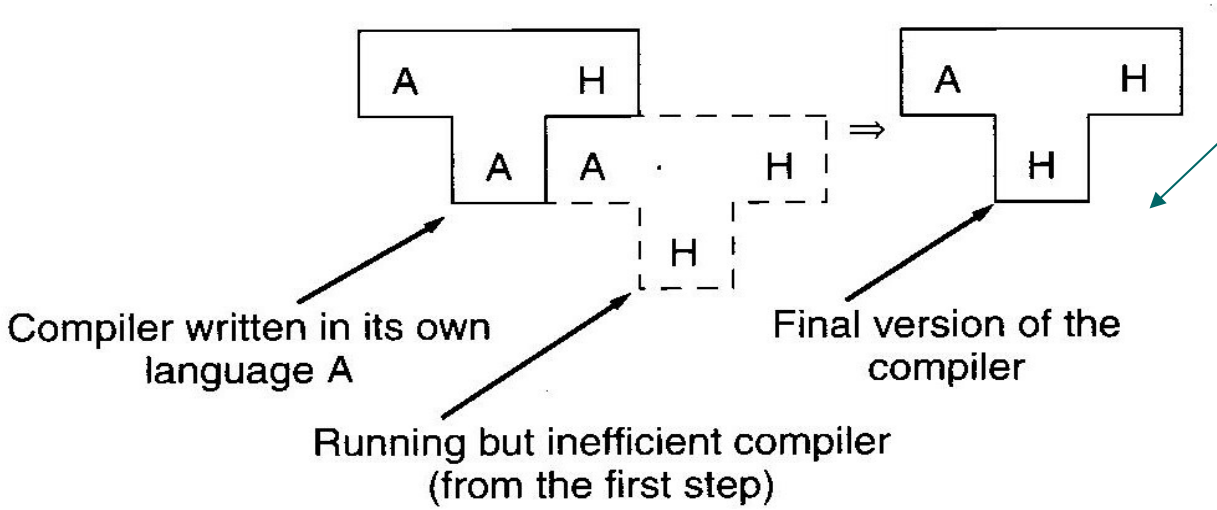
Skrevet i en begrenset del av A

Lager god H-kode

- men sakte



Bootstrapping: Step 2



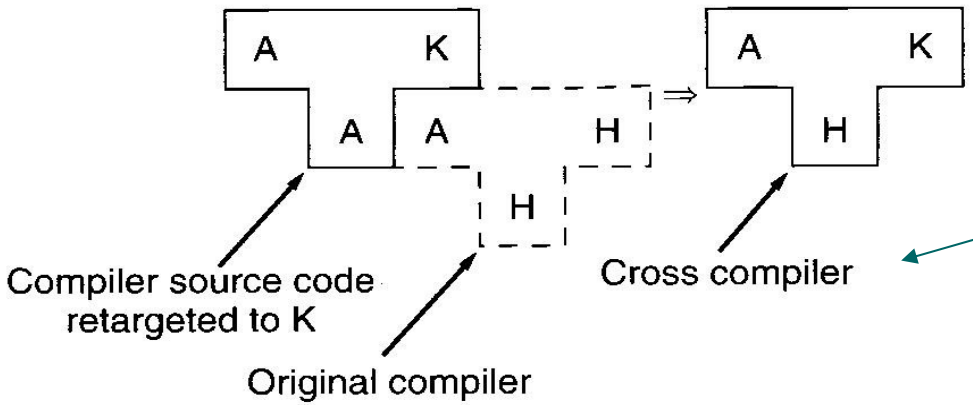
Lager god H-kode

- og fort

Krysskompilering

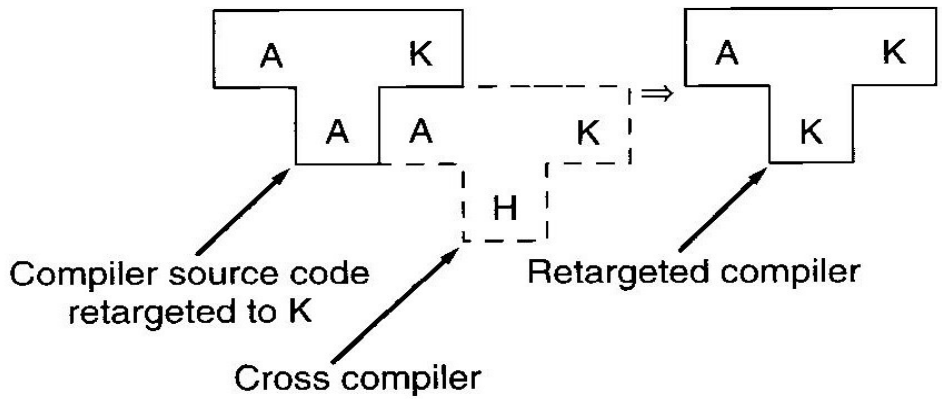
- Har: A kompilator som oversetter til H-maskinkode
- Ønsker: A-kompilator som oversetter til K-maskin kode

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 - 2006