Velkommen til INF5110 –Kompilatorteknikk

15. januar 2013

- Kursansvarlige:
  - Stein Krogdahl [steink@ifi.uio.no]
  - Ragnhild Kobro Runde [ragnhilk@ifi.uio.no]
  - Henning Berg (oblig-ansvarlig) [hennb@ifi.uio.no]

- Kursområdet: www.uio.no/studier/emner/matnat/ifi/INF5110/v12
  - Plan over forelesningene, pensum etc. etter hvert som det blir klart
  - Diverse beskjeder

Vårens opplegg 2013

- 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
  - Vil grovt sett følge opplegget fra 2012
- 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
  - Blir til sammen en kompilator for et enkelt språk som vi definerer
  - Normalt at 2 og 2 arbeider sammen og det er tillatt å jobbe alene.
  - Kan søke om å få arbeide 3 sammen
- Eksamen:
  - 5. 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-kompilatorer!

Men ...

- Mange applikasjoner vil inneholde håndtering av input som kan sees som enkle former for programmeringsspråk, og kan greiest behandles som det.

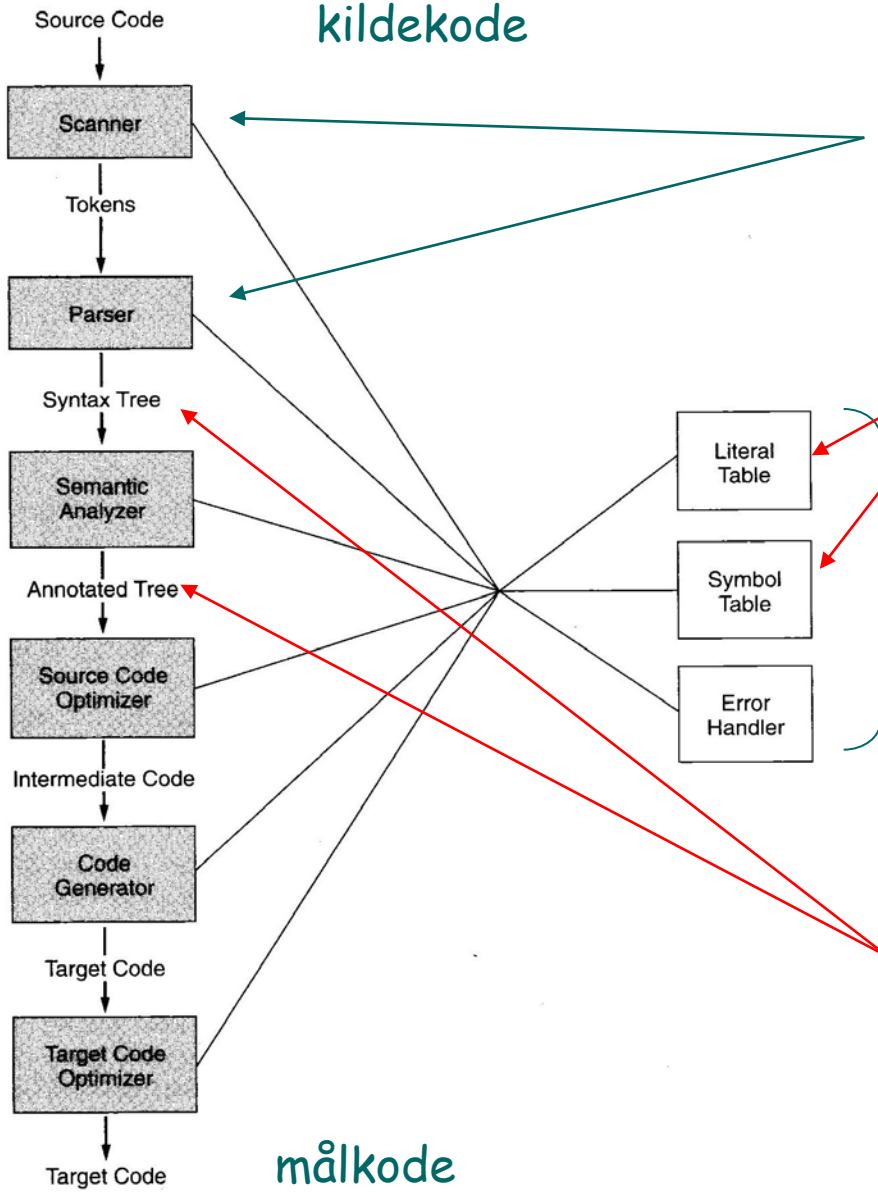
- En del kommer til å lage eller modifisere kodegeneratorer av forskjellige typer, f.eks. enkel produksjon av kode basert på modeller (f.eks. UML)

- Man kan ønske å eksperimentere med nye språkbegreper. Noen vi har sett på her på huset er:
  - Aspekt-orientering: Håndtere visse aspekter av et program ved spesiell aspekt-kode (krever aspekt-kompilator)
  - Traits: Samlinger av metoder som klasser kan ta inn, uavhengig av klassehierarkier.
  - Package Templates: Generelle samlinger av klasser som kan hentes inn i programmer, og samtidig tilpasses lokale behov.
  - Domenespesifikke språk: språk som er rettet mot helt spesielle anvendelser

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.

Bokens oversikt over en typisk kompilator



kildekode

"faser" under kompilering

Er nå ofte en del av treet

Sentrale tabeller og mekanismer

Mellomformer av programmet

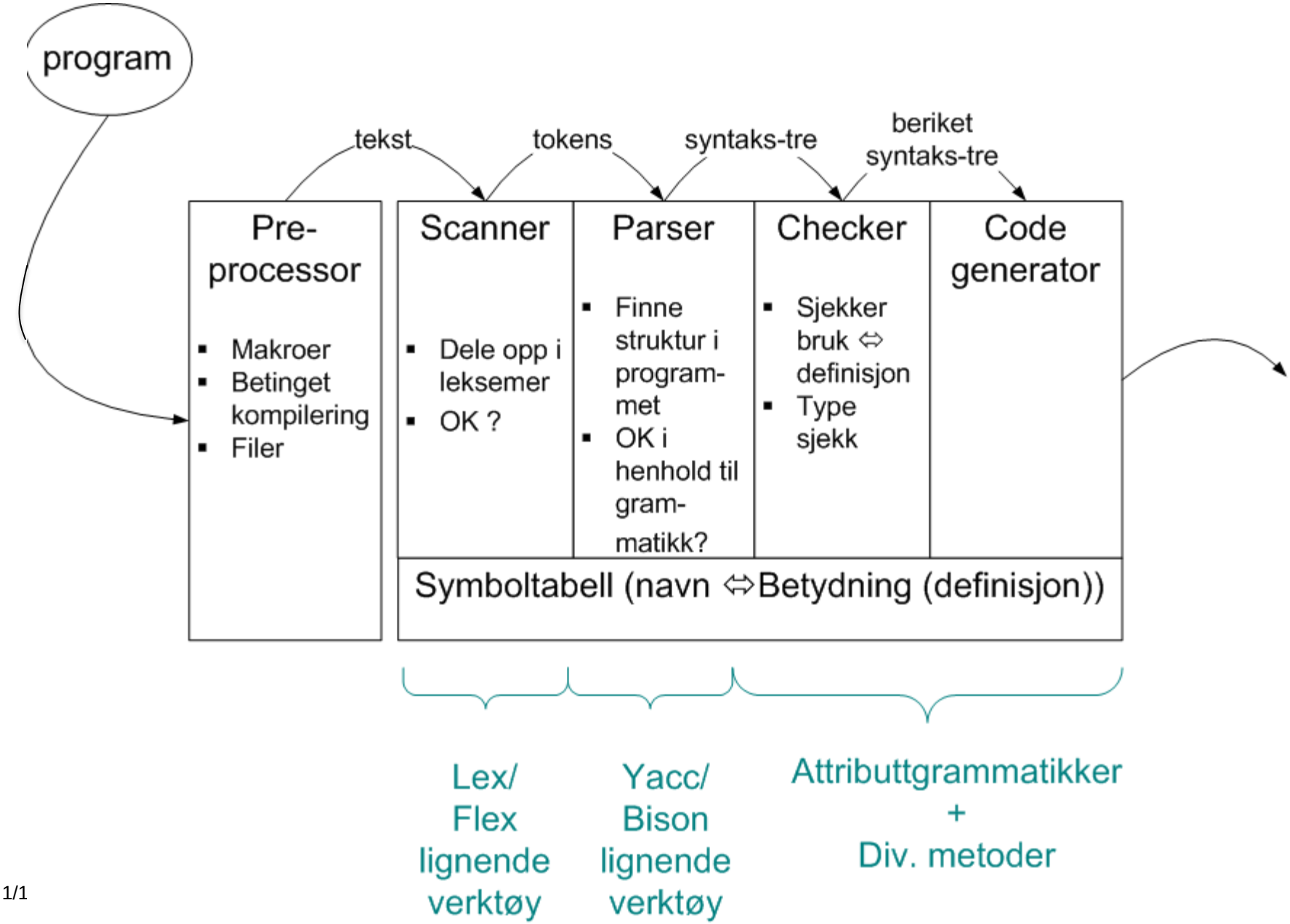
målkode

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 - 2013

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	

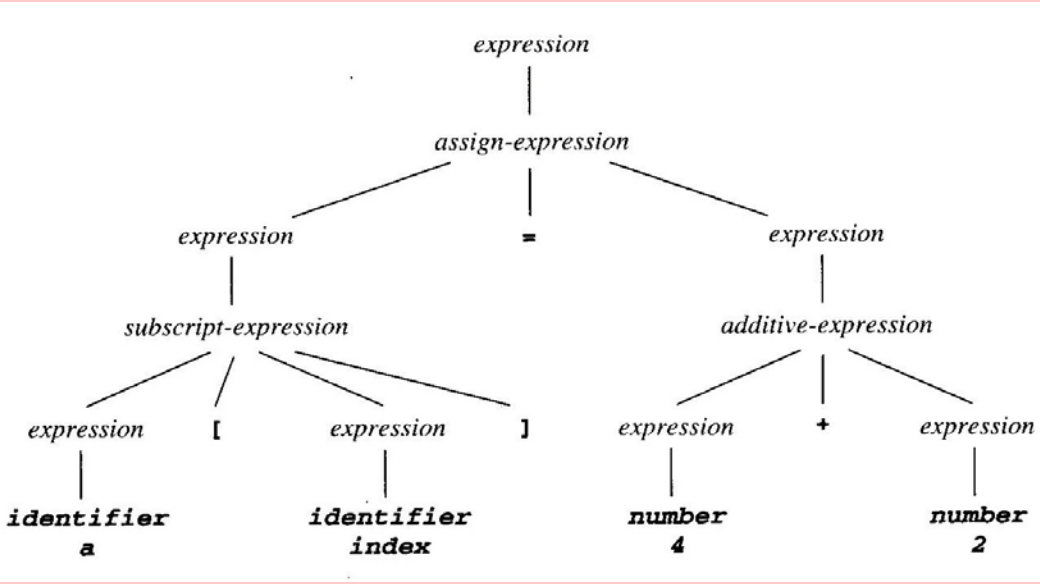
Tilsvarende for tekstkonstanter

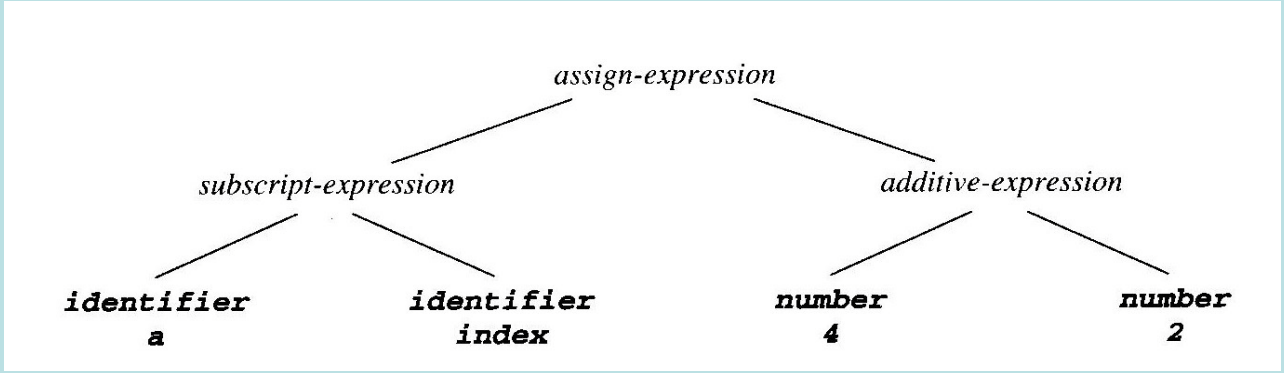
Parser

parserings-tre
(syntaks-tre)

resultat av parsering

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





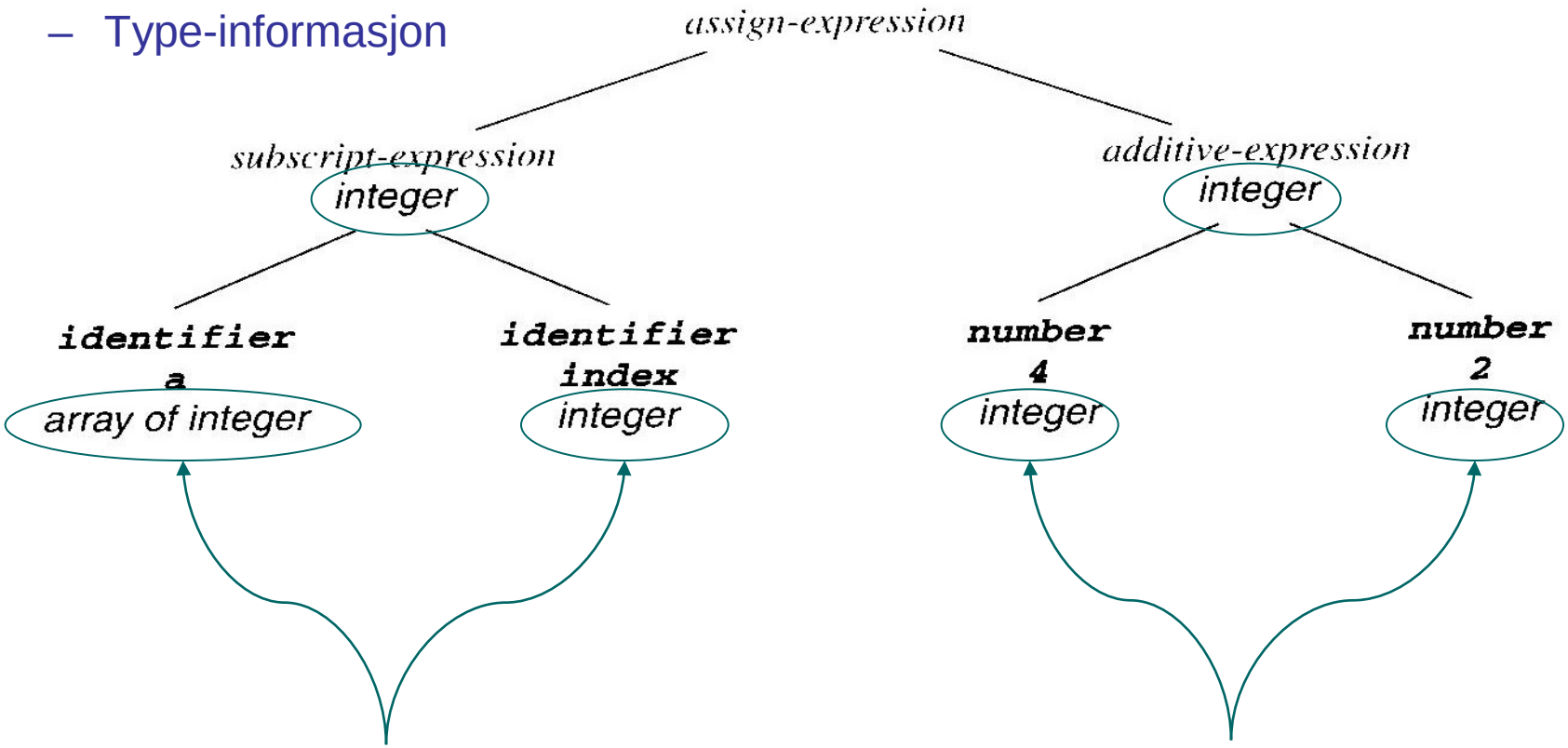
abstrakt
syntaks-tre

«syntaktisk
sukker»
er fjernet

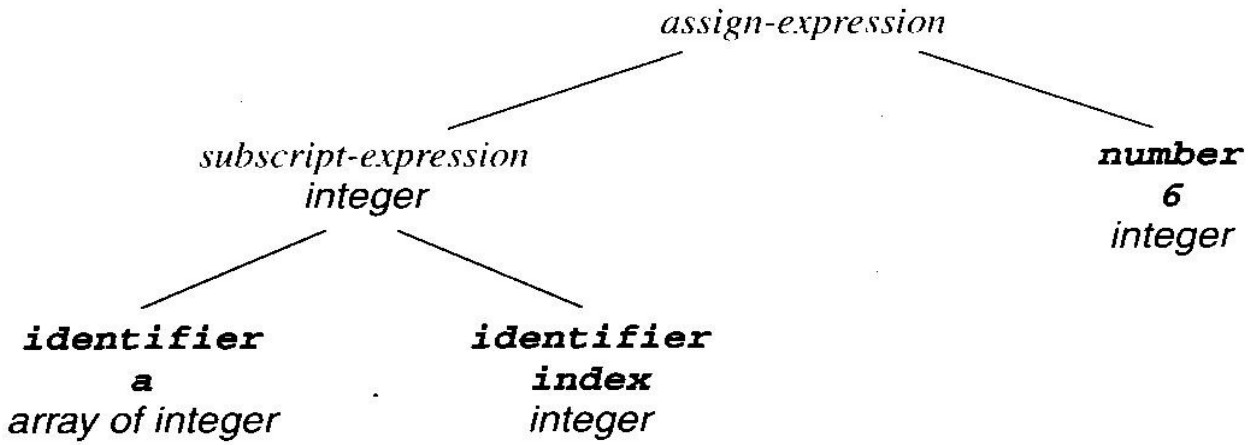
INF 5110 - 2013

Resultat av semantisk analyse

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



Optimalisering på kildekode nivå (ikke markert på figuren)




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

opprinnelig

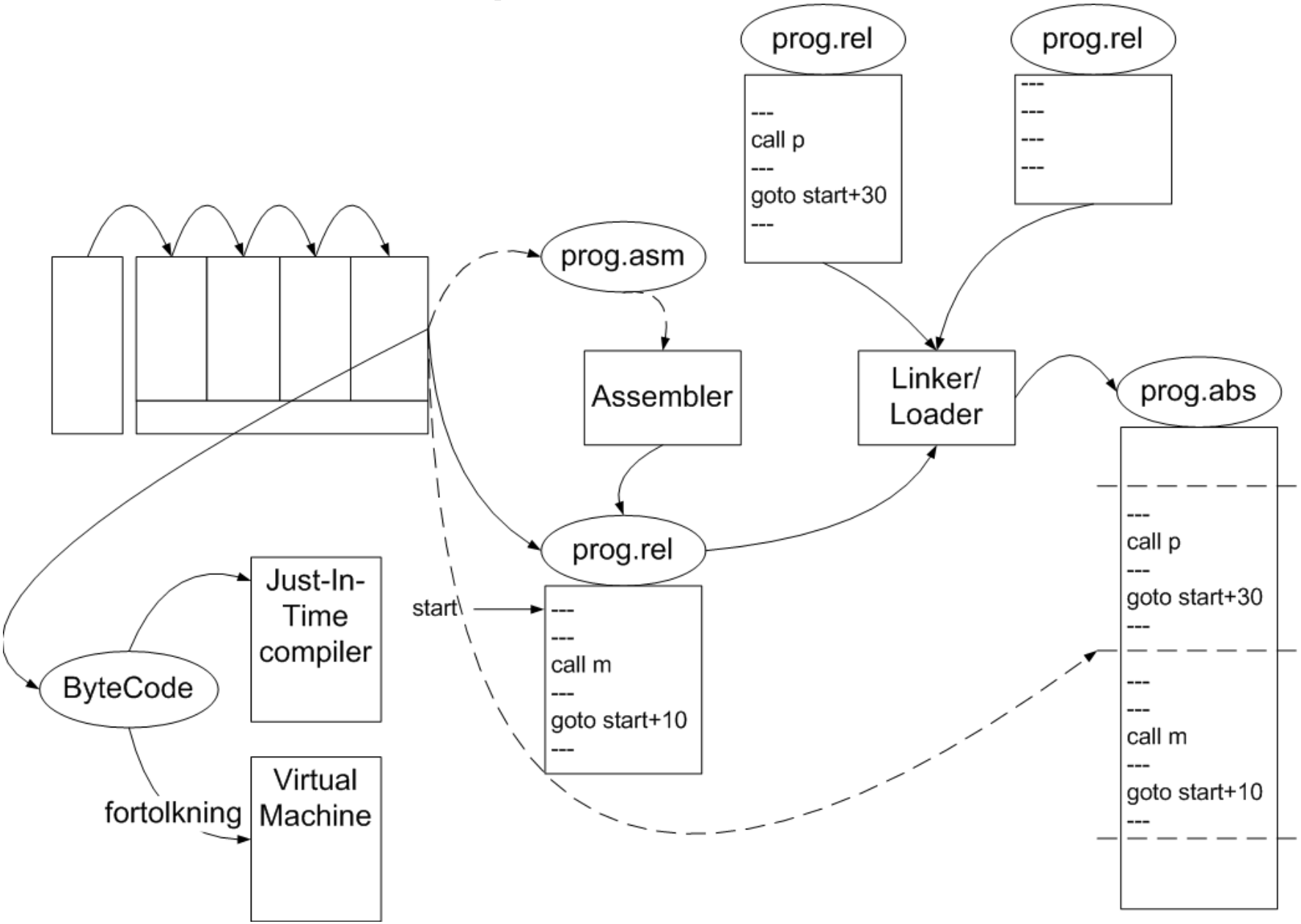
```
t = 6
a[index] = t
```

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

INF 5110 - 2013

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

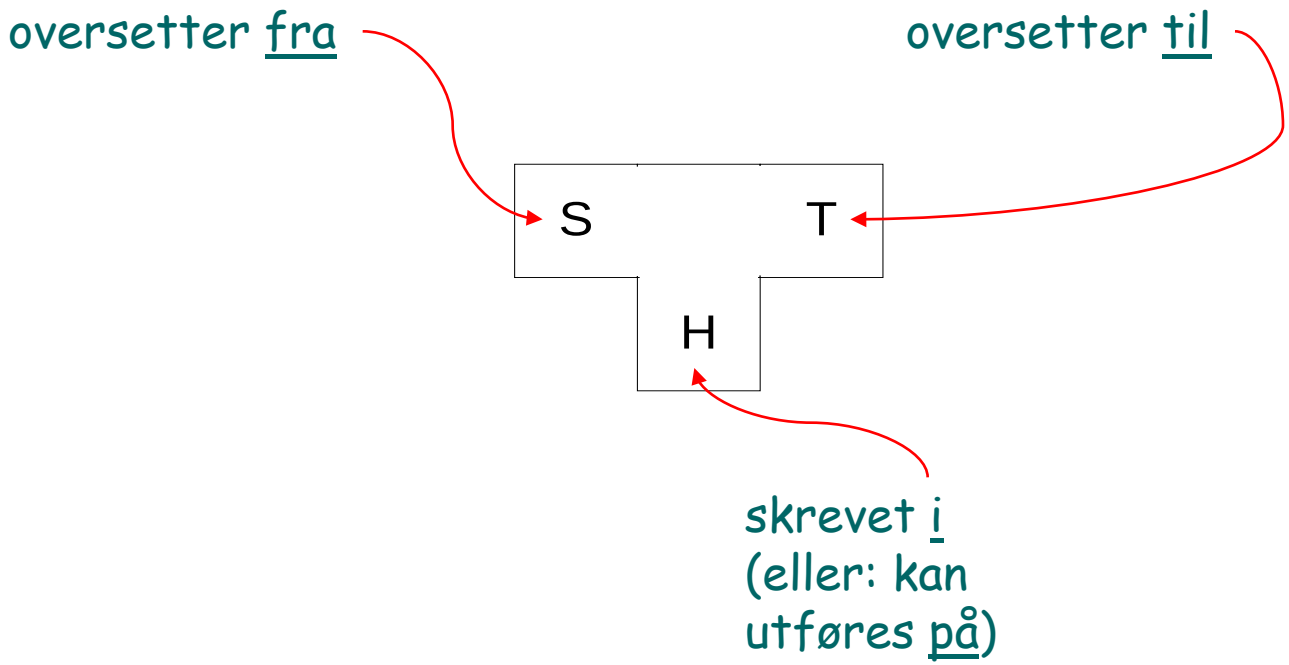
- Full interpretering
  - Utføres direkte fra programteksten/syntakstre
  - Brukes mest for kommandospråk til operativsystem etc.
  - Utførelse typisk 10 – 100 ganger saktere enn ved full oversettelse

- Oversettelse til mellomkode som interpreteres (eller kan oversettes til maskin-kode)
  - Brukes for Java (class-filer), og mye for Smalltalk
  - Mellomkoden er valgt slik at den er grei å utføre (byte-kode for Java)
  - Kan utføres av en enkel interpreter (Java: Java Virtual Machine)
  - Interpretering går typisk 3 - 30 ganger så sent som oversatt kode.
  - Dog: I de fleste moderne Java-systemer oversettes byte-koden til maskinkode (av JVM-en) umiddelbart før den utføres (JIT, Just-In-Time kompilering).
  - Også diverse mellomvarianter brukes.

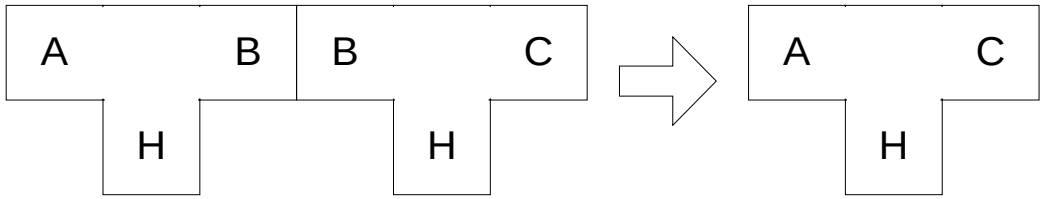
Senere utvikling innen kompilatorer etc.

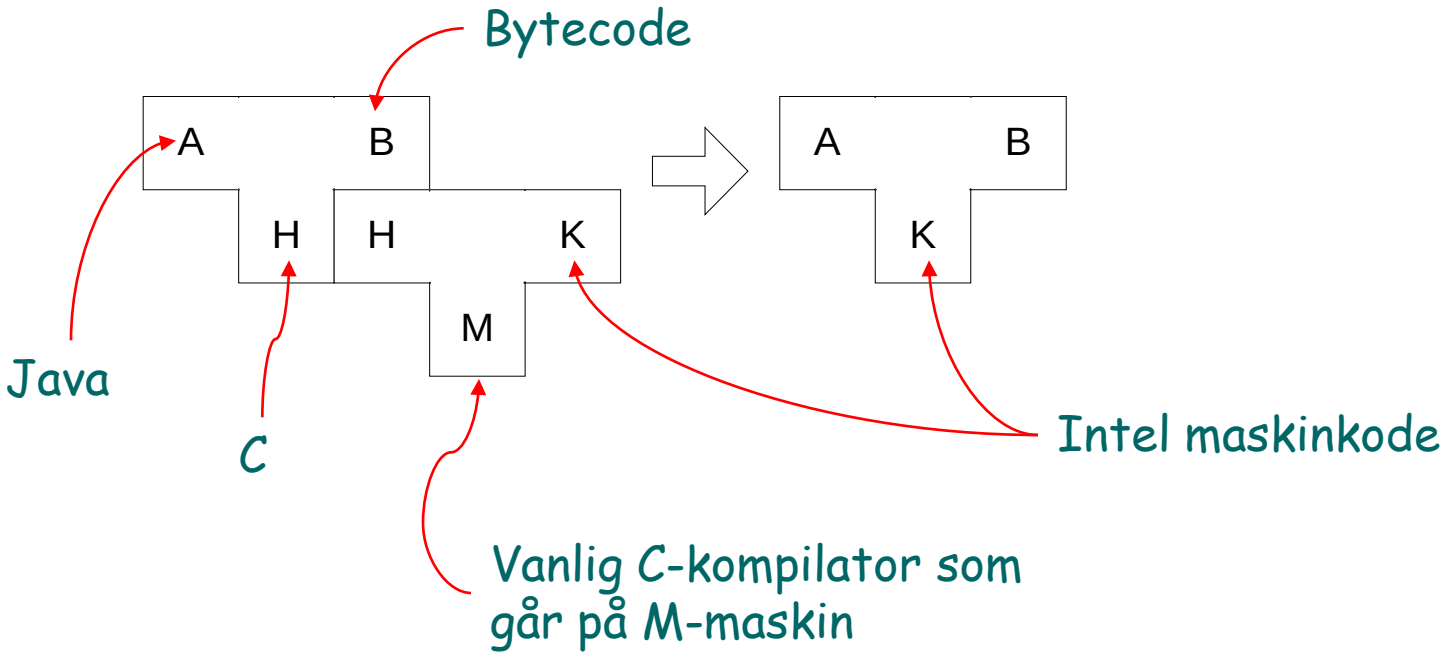
- Lager (spesielt intern-lager) er blitt billig, og dermed stort
  - Man kan ha hele programmer inne i maskinen under kompilering
  - 200 byte pr. linje gir 50 000 linjer på 10 Mbyte
  - Lagrene ble store nok til dette rundt 1990
- Objektorienterte språk er blitt meget populære
  - Spesielle teknikker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

”Bootstrapping” og ”porting” – En beskrivelses-måte



To sammensetningsoperasjoner





Bootstrapping:

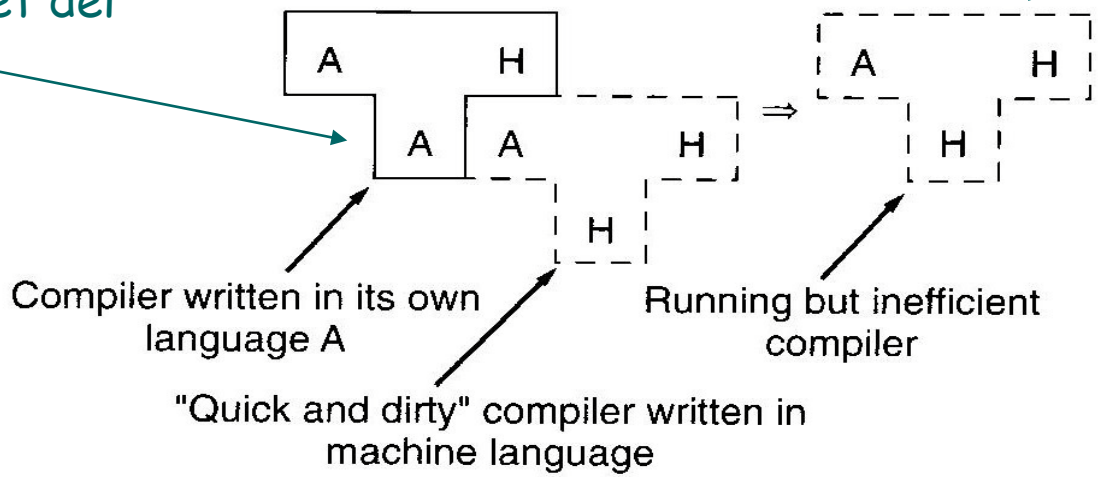
Lage en kompilator som er skrevet i eget språk A, går fort, og lager god kode

Steg 1:

Skrevet i en begrenset del av A

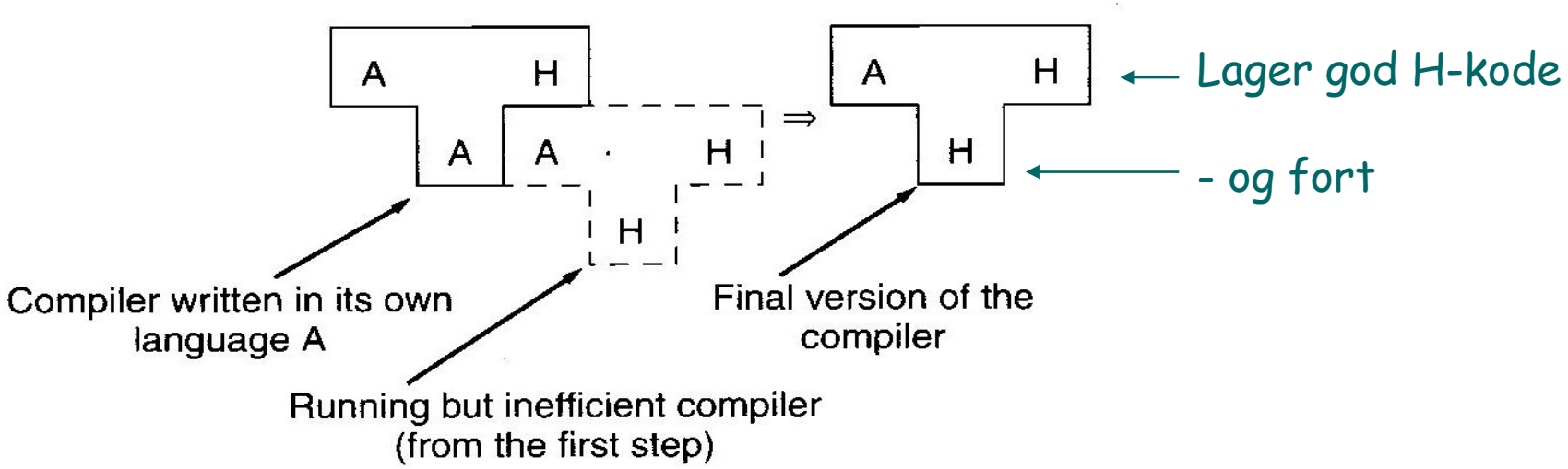
Lager god H-kode

- men sakte



Bootstrapping:

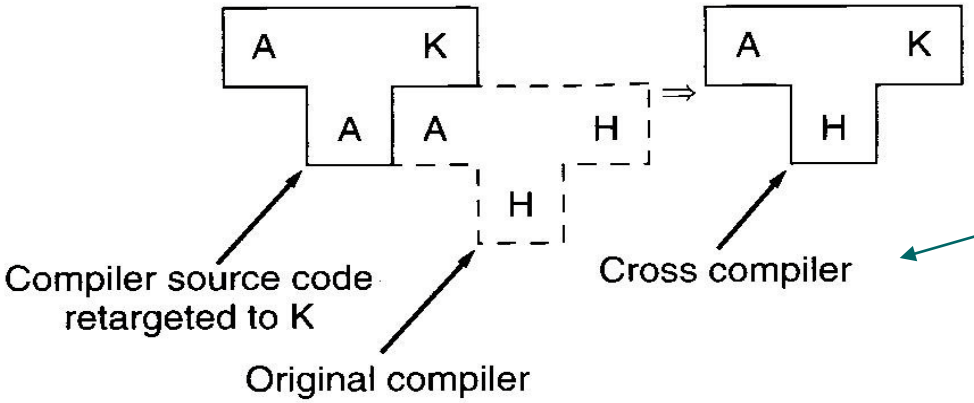
Steg 2



Krysskompilering

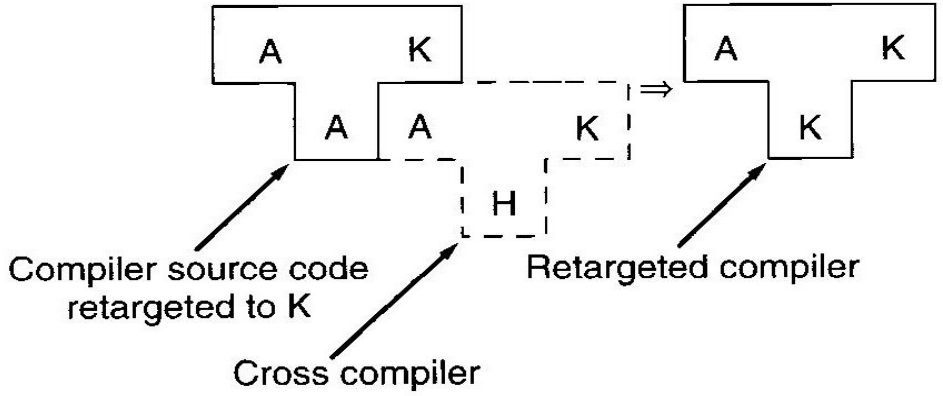
- Har A kompilator som oversetter til H-maskinkode
- Ønsker: A-kompilator som går på K-maskin, som oversetter til K-kode

Steg 1: Skriv om kompilatoren 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 - 2013