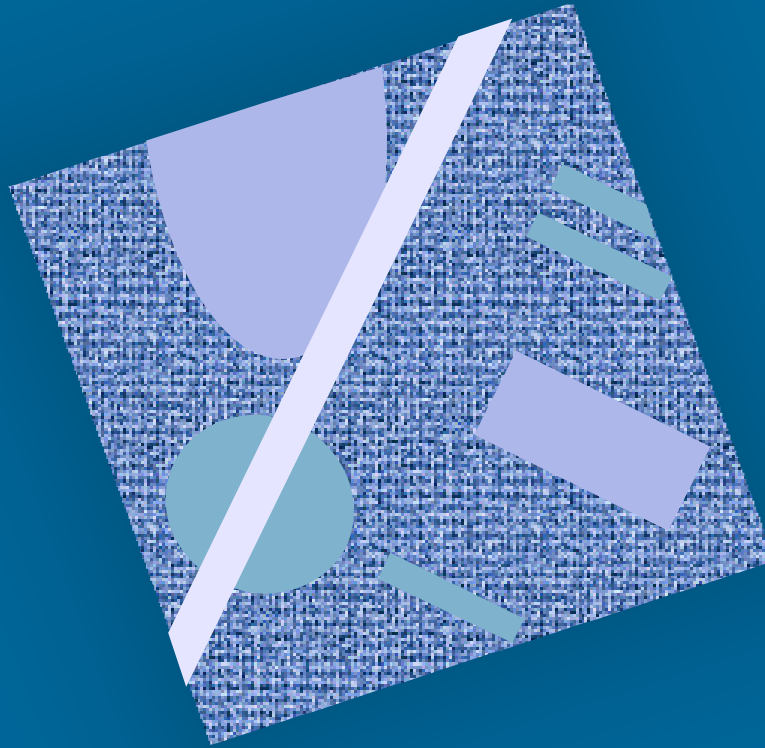


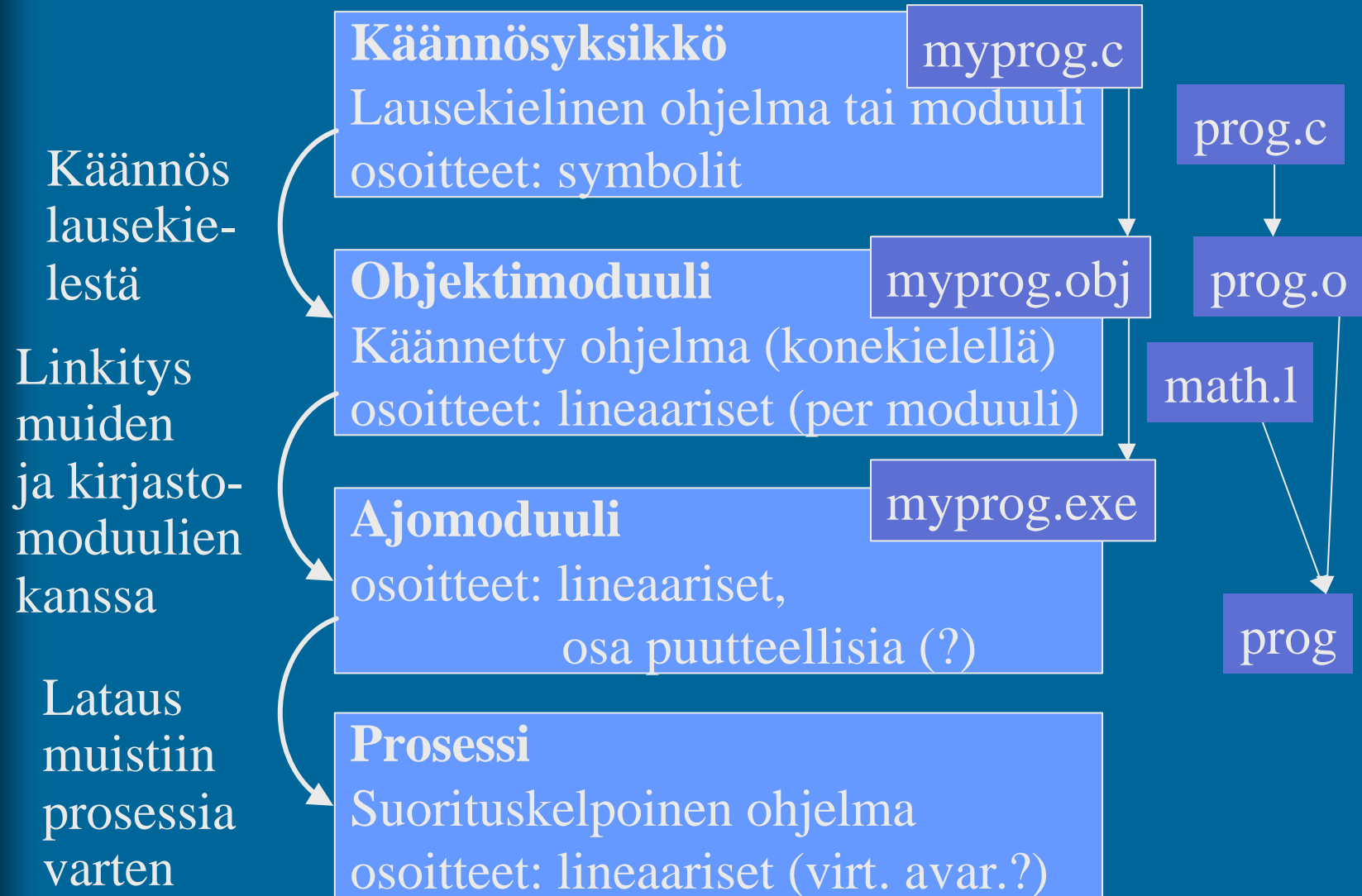
# Jakso 10

## Ohjelman suoritus järjestelmässä



Käännös  
Linkitys  
Dynaaminen linkitys  
Lataus

# Lausekielestä suoritukseen

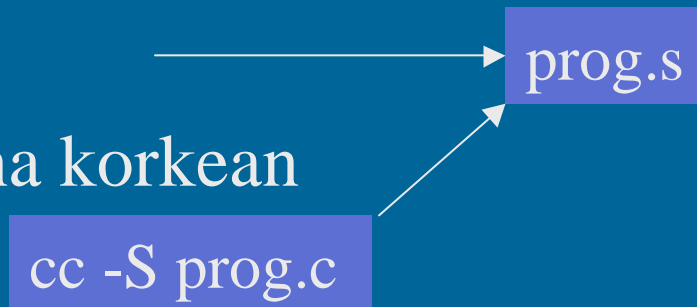


# Käännösyksikkö <sup>(4)</sup>

- Jollain ohjelmointikielillä kuvattu eheä kokonaisuus, joka halutaan aina kääntää yhdessä
  - kaikki yhteen liittyvät aliohjelmat
  - olioperustainen luokka
- Liian suuri kokonaisuus?
  - turhaa aikaa kääntämiseen joka muutoksen jälkeen
- Liian pieni kokonaisuus?
  - turhaa aikaa murehtia ja toteuttaa liitoksia muiden moduulien kanssa
- Käännösyksikön ohjelmointikieli ei ole tärkeä
  - niiden sitominen yhteen tapahtuu objektimoduulien tasolla

# Assembler-kielinen käännösyksikkö

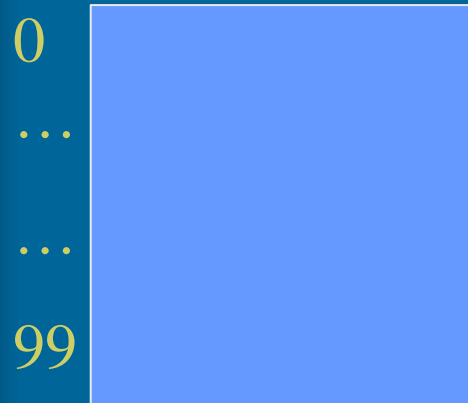
- Käännösyksikkö voi olla myös suoraan k.o. koneen symbolisella konekielellä kirjoitettu
  - suoraan käsin
  - kääntäjän generoimana korkean tason kielestä
- Käännöksen tekee assembler-kääntäjä tavallisen kääntäjän asemesta
  - yleensä osa tavallista kääntäjää



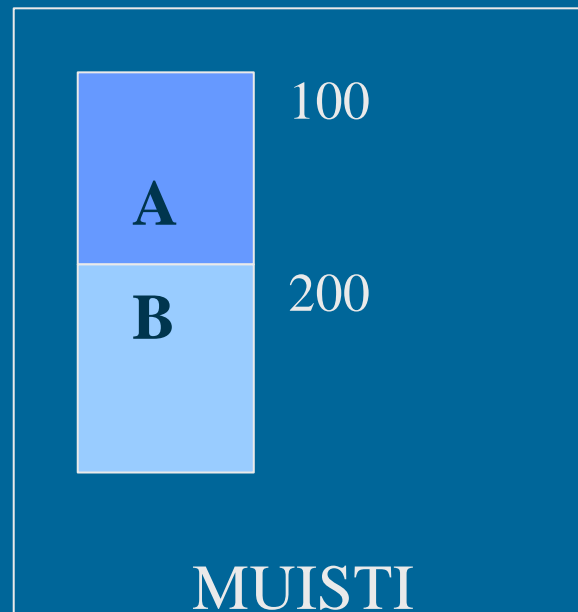
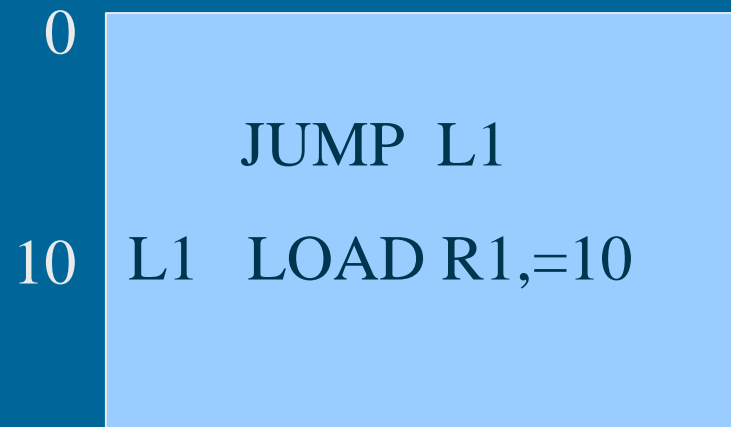
# Objektimoduuli (8)

- Konekielinen koodi
  - moduulin sisäiset viitteet paikallaan (linearisessa muistiavaruudessa)
  - moduulin ulkopuoliset viitteet merkitty
- Linkitystä varten tiedot
  - tiedot niiden osoitteiden sijainnista, jotka täytyy päivittää, jos moduulin sijainti muistissa vaihtuu (suorat muistiosoitteet, joihin ei käytetä virtuaalimuistin osoitteenmuunnosta)
  - tiedot viittauksista moduulin ulkopuolelle **IMPORT**
  - tiedot kohdista joista tähän moduuliin saa viitata ulkopuolelta **EXPORT**
  - symbolitaulu

## Moduuli A



## Moduuli B



# Symbolitaulu

- Kääntäjä generoi
- Ylläpidetään linkityksen aikana
- Joskus ylläpidetään myös latauksen jälkeen virheilmoitusten tekemistä varten
  - ohjelmien kehitysympäristöt ylläpitävät symbolitaulua koko ajan
- Jätetään pois valmiista ohjelmasta
  - vie turhaa tilaa

# Lähdekielinen ohjelma (5)

- Pascal lauseke:  $N := I+J;$
- C lauseke:  $N = I+J$
- Java lauseke:  $N = I+J;$

TTK-91 symbolinen  
konekieli:

```
I          DC    3
J          DC    4
N          DC    0
FORMULA:  LOAD  R1, I
          ADD   R1, J
          STORE R1, N
```

Pentium II, Motorola 680x0 ja  
SPARC symbolinen konekieli:

ks. Fig. 7-2 [Tane99]



# (Assembler) kääntäjän ohjauskäskyt <sup>(4)</sup>

- Eivät varsinaista koodia
- Ohjaavat käännoästä

TTK-91:      DC  
                 DS  
                 EQU

Pentium II:

ks. Fig. 7-3 [Tane99]

# Makrot <sup>(6)</sup>

- Helpottavat ohjelmointia
- Usein toistuville koodisarjoille annetaan nimi  $\Rightarrow$  makro
- Makroilla voi olla parametreja
- Esimerkkejä
  - swap ks. Fig. 7-4 ja 7-6 [Tane99]
  - aliohjelmien prologi ja epilogi
  - itse tehdyt, kääntäjän käyttämät
- Makrot käsitellään ennen kääntämistä
- Erot aliohjelmiin ks. Fig. 7-5 [Tane99]

# Literaalit <sup>(5)</sup>

- Vakioita
- Niin suuria, että eivät mahdu konekäskyn vakio-osaan

ttk-91: käskyn vakiot 2-tavuisia,  
arvoalue: -128...127

- Halutaan pitää datan joukossa eikä käskyjen yhteyteen talletettuna

```
Pi      DC      3.14159265 ; (!!??)
One     DC      1
OneMeg  DC     1024576
```

- Niitä ei saisi muuttaa

```
LOAD R1, One
ADD  R1,=1
STORE R1, One ; ask for trouble
```

# Literaalit

- Korkean tason kielissä kaikki isot vakiot aina literaaleja
  - kääntäjän pitäisi estää literaalien muuttamisen
    - FortranX: `5 = 6;` ???????
  - literaalia ei saisi välittää viiteparametrina
    - aliohjelma voisi muuttaa sen arvoa?
- Myöskään joissakin assemblerkielissä literaaleja ei tarvitse erikseen esitellä
  - helpommin luettavaa koodia Load R14, =F'234567'
  - vakion 234567 tilanvaraus automaattisesti

# Assembler käänнос (10)

- 1. vaihe:
  - laske käskyjen tilanvaraukset
    - ttk-91 helppoa, koska kaikki käskyt 4 tavua!
  - generoi symbolitaulu ks. Kuva 6.2 [Häkk98]
    - arvot, arvon vaatima tavumäärä
    - uudelleensijoitustiedot (omana tauluna?)
  - generoi tai käytä muita tauluja
    - literaalitaulu (tilanvaraus lopuksi)
    - kääntäjän ohjauskäskytaulu
    - operaatiokooditaulu

# Assembler käänнос (7)

- 2. vaihe

- generoi lopullinen objektimoduuli

ks. Kuva 6.3 [Häkk98]

ks. Fig. 7-16 [Tane99]

- tulosta symbolinen assembler -listaus
- generoi taulut linkitystä varten
  - osana objektimoduulia
- anna virheilmoitukset

- 3. vaihe

- koodin optimointi

# TTK-91 Assembler käännös

```
s    DC    0
i    DC    1
0: Taas  LOAD  R1, i
1:      MUL   R1, R1
2:      ADD   R1, s
3:      STORE R1, s
4:      LOAD  R1, i
5:      ADD   R1, =1
6:      STORE R1, i
7:      COMP  R1, =21
8:      JLES  Taas
9:      SVC   SP, =HALT
```

tunnetaan

Taas = 0

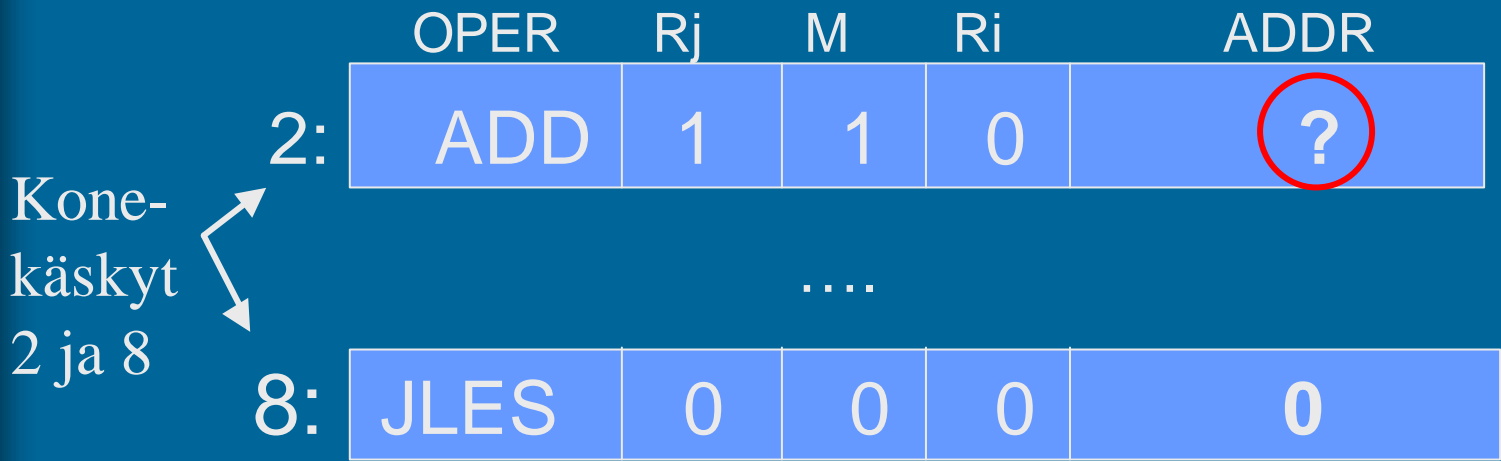
i = ?

s = ?

# Symbolitaulu 1. vaiheen aikana

ks. kalvo 15

Symboli	tyyppi	arvo	uud. sij.tietoa
s	data	?	2, 3
i	data	?	0, 4, 6
Taas	viite	0	8
HALT	vakio	11	





## Koodi & data 1. vaiheen jälkeen

```
s    DC    0
i    DC    1
```

```
0: Taas  LOAD   R1, i
1:      MUL    R1, R1
2:      ADD    R1, s
3:      STORE  R1, s
4:      LOAD   R1, i
5:      ADD    R1, =1
6:      STORE  R1, i
7:      COMP   R1, =21
8:      JLES   Taas
9:      SVC    SP, =HALT
10:     0      ; siis s = 10
11:     1      ;    i = 11
```

Kaikilla  
symboleilla  
tunnettu arvo

# Symbolitaulu 1. vaiheen jälkeen

ks. kalvo 17

Symboli	tyyppi	arvo	uud. sij.tietoa
s	data	10	2, 3
i	data	11	0, 4, 6
Taas	viite	0	8
HALT	vakio	11	

	OPER	Rj	M	Ri	ADDR
2:	ADD	1	1	0	10

....

8:	JLES	0	0	0	0
----	------	---	---	---	---

# TTK-91 objektimoduuli

Moduulin otsake

EXPORT-hakemisto

IMPORT-hakemisto

Uudelleensijoitushakemisto

Koodi ja alustettu data

Moduulin loppuke

( Kuva 6.3  
[Häkk98] )

# TTK-91 objektimoduuli

- Moduulin otsakeosa
  - moduulin nimi
  - linkittäjän tarvitsemia tietoja
    - objektimoduulin osien pituudet
    - käännös päivämäärä
    - kääntäjän nimi ja versio
    - ensimmäisen suoritettavan käskyn osoite
      - ellei aina 0

# TTK-91 objektimoduuli

- EXPORT-hakemisto
  - tunnukset, joihin voidaan viitata muista moduuleista
    - rutiinit, aliohjelmat
    - yhteiskäyttöinen data
  - tunnuksen osoite (= symbolin arvo)
  - mahdollinen käyttöoikeus
    - R/W/E/RW

# TTK-91 objektimoduuli

- IMPORT-hakemisto
  - muissa moduuleissa määritellyt tunnukset
    - tunnus
    - niiden käskyjen osoitteet, jossa tunnus esiintyy
- Koodi ja alustettu data
  - alustamattomille muuttujille ei tarvitse varata tilaa
    - otettava huomioon data-alueen koossa

# TTK-91 objektimoduuli

- Uudelleensijoitushakemisto
  - niiden käskyjen osoitteet, joiden osoiteosaa on korjattava, kun siirrytään moduulien yhteiseen osoiteavaruuteen
    - suoraviivainen lisäys ei toimi, sillä käskyn osoiteosa voi olla vakio, jota ei saa muuttaa
    - erikseen paikalliseen dataan viittaavat ja hyppykäskyt, sillä linkittäessä yhdistetään erikseen data- ja koodialueet

# Korkean tason kielen käännös (7)

- Enemmän vaihteita

- Syntaktisten alkioiden etsintä

- Syntaksipuun generointi ja jäsenitys

- Lauseiden tunnistaminen syntaksipuun avulla

- Välikielen (välikoodin) generointi (ei aina)

Välikieliesitys ja symbolitaulut

- Koodin generointi

(back end)

Lisää  
tietoa?



Kääntäjien  
ja ohj. kielten  
kurssit

ks. syntaksipuun,  
jäsenityspuun  
esimerkit



# Linkitys

- Uudelleensijoitusongelma (relocation problem)
  - jokaisen objektimoduulin osoitteet alkavat 0:sta
  - tulosmoduulissa kaikki yhdessä lineaarisessa osoiteavaruudessa
  - useimpien moduulien kaikkia osoitteita täytyy muuttaa
    - käskyjen osoitteet
    - datan osoitteet

# Linkitys esimerkki

- Neljä moduulia: A, B, C ja D ks. Fig. 7-14 [Tane99]
- Laske joka moduulille uudelleen-  
sijoitusvakio (moduulin alkuosoite) (relocation  
constant)
- Lisää k.o. vakio kunkin moduulin sisäisiin viitteisiin
- Etsi kaikki moduulien väliset viitteet, ks. Fig. 7-15 (a) [Tane99]  
ja aseta kyseisten viitteiden osoitteet oikein ks. Fig. 7-15 (b) [Tane99]

# Muuttujan X viittausten päivitys <sup>(3)</sup>

- Miten löytää kaikki kohdat, jossa muuttujaan X viitataan?
- Vastaus 1: iso taulukko, jossa kaikki kohdat listattu
- Vastaus 2: Muuttujan X viittaukset on kaikki linkitetty keskenään linkitetyksi listaksi objektimoduulissa
  - vain alkuosoite taulukossa

# Muuttujan X viitaukset linkitettynä listana

symbolitaulu, moduuli ABC

Symb	sij	viittaus
X	700	23

lähdekoodi

```
23:   Load  R1, X
...
34:   Store  R3, X(R1)
...
555:  Add    R4, X

700:  DC    0 ; X
```

objektimoduuli

```
23:   Load 1 0 34
...
34:   Store 3 1 555
...
555:  Add  4 0 -1
```



# Staattinen linkitys <sup>(5)</sup>

- Tavallinen (staattinen) linkitys vaatii, että kaikki ohjelmakoodissa viitatus moduulit ja kirjastorutiinit on linkitetty ennen suoritusta
- Ajomoduulista tulee hyvin iso
  - mukana myös paljon moduuleja, joihin ei yhdellä suorituskerralla tule lainkaan viittauksia
    - kääntäjässä koodin optimointikoodi, vaikka optimointia ei suoriteta
    - pelissä tasojen 8-22 moduulit, kun aloittelija ei pääse tasoa 3 ylemmäksi vielä kuukausiin

# Dynaaminen linkitys (4)

- Jätetään linkityksessä kutsukohdat muihin moduuleihin auki
- Pienempi ajomoduli, mutta hitaampi suorittaa
- Viittaus ”ratkaisemattomaan” moduuliin ratkotaan suoritusaikana
- Suoritus keskeytyy ja puuttuva moduuli linkitetään paikalleen (kaikki viittaukset siihen korjataan kuntoon)

# Windows DLL

- DLL - Dynamically Linked Library

- koodia, dataa,  
molempia

.dll	yleinen tapaus
.drv	driver
.fon	font

- Säästää tilaa myös yhteiskäytön vuoksi

ks. Fig. 7.19 [Tane99]

- Helpompi korjata virheitä

- ei tarvita uutta käännöstä!
- riittää kun DLL vaihdetaan uuteen

- Kootaan kuten tavallinen objektimoduuli

- erikoislipuke merkitsee sen DLL:ksi (huomioidaan linkityksen yhteydessä)

# Windows DLL:n linkityksen kaksi tapaa <sup>(3)</sup>

- Epäsuora dynaaminen linkitys (implicit linking)
  - kaikki viitatus moduulit ladataan (lataus aloitetaan) virtuaalimuistiin ja niihin viitataan staattisesti linkitetyn pienemmän liitospalikan (import library) avulla
- Suora dynaaminen linkitys (explicit linking)
  - koodiin generoidaan suoraan viitepaikalle käskyt, joiden avulla linkitys tapahtuu tarvittaessa
  - DLL ladataan vain jos siihen tulee viittaus
- DLL suoritetaan osana kutsuvaa prosessia käyttäen sen omaa aktivointitietuepinoa



# Nimien sidonta <sup>(3)</sup>

- Milloin symbolin L suoritusaikainen muistiosoite sidotaan (lasketaan valmiiksi)? (binding time)
  - ohjelman kirjoitusaikana?
  - käännösaikana?
  - linkityksessä?
  - latauksessa?
  - kantarekisterin asetuksen aikana?
  - osoitteen sisältämän konekäskyn suoritusaikana?
- Jos käskyä siirretään sitomisen jälkeen, mennään metsään ...
- Sijainnista riippumattomassa (position independent) koodissa kaikki viittaukset ovat joko absoluuttisia, suhteessa PC:hen tai pinossa

virtuaaliosoite

≠

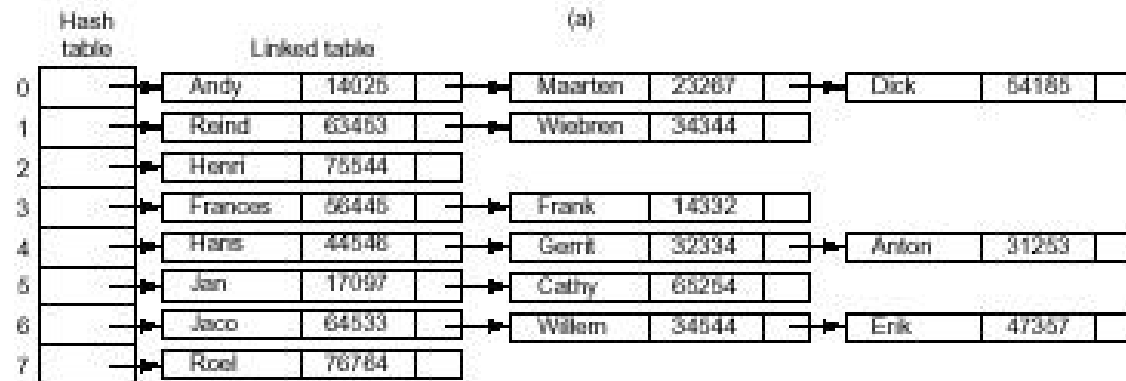
# Lataus <sup>(4)</sup>

- Ajomoduulista luodaan suorituskelpoinen prosessi (rakennetaan PCB ja sen viitteet kuntoon)
- Prosessin koodialueet (tai ainakin sen pääohjelma) ja tarvittava ladataan muistiin, prosessi siirretään R-to-R jonoon
- Sitten kun prosessi saa suoritusvuoron suorittimella, MMU ladataan PCB:n avulla tämän prosessin tiedoilla
  - virtuaalimuistia käytettäessä nimien sidonta tehdään viime hetkellä (konekäskyn suoritusaikana) MMU:n (ja TLB:n) avulla

# -- Luennon 10 loppu --

Andy	14025	0
Anton	31253	4
Cathy	65254	5
Dick	54185	0
Erik	47357	6
Frances	56445	3
Frank	14332	3
Gerrit	32334	4
Hans	44545	4
Henri	75544	2
Jan	17097	5
Jaco	64533	6
Maarten	23287	0
Reind	63453	1
Roel	76784	7
Willem	34544	6
Wiebren	34344	1

(a)



(b)

**Figure 7-12.** Hash coding. (a) Symbols, values, and the hash codes derived from the symbols. (b) Eight-entry hash table with linked lists of symbols and values.

[Tane99]