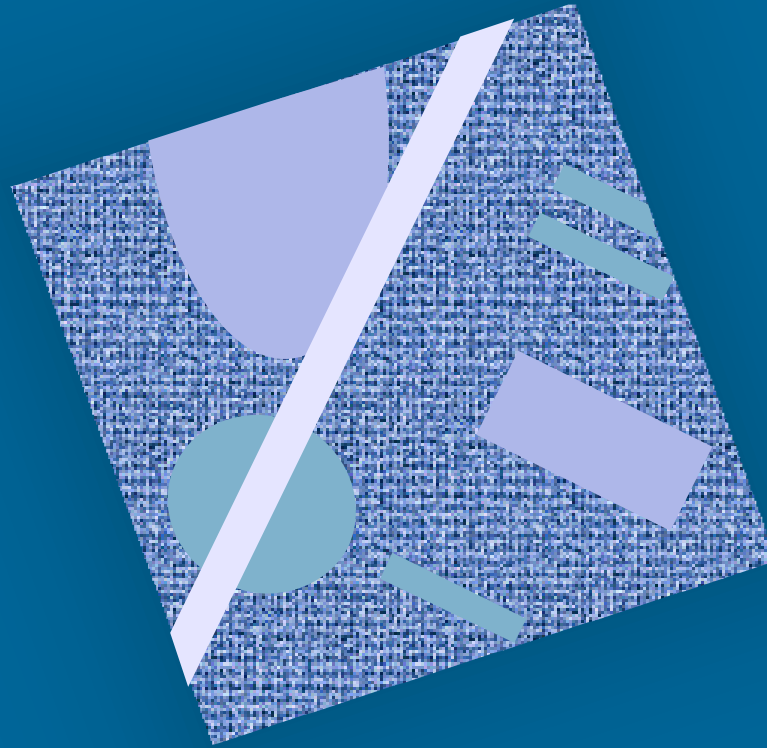


# Jakso 4

## Aliohjelmien toteutus



Tyypit  
Parametrit  
Aktivoititietue (AT)  
AT-pino  
Rekursio

# Aliohjelmatyypit <sup>(2)</sup>

- Korkean tason ohjelmointikielen käsitteet:
  - aliohjelma, proseduuri
    - parametrit
  - funktio
    - parametrit, paluuarvo
  - metodi
    - parametrit, ehkä paluuarvo
- Konekielen tason vastaavat käsitteet:
  - aliohjelma
    - parametrit ja paluuarvo(t)

# Parametrit ja paluuarvo <sup>(2)</sup>

- Muodolliset parametrit

- määritelty aliohjelmassa ohjelmointihetkellä
- tietty järjestys ja tyyppi
- paluuarvot

```
Tulosta (int x, y)  
Laske(int x): int
```

- käsittely hyvin samalla tavalla kuin parametreillekin

- Todelliset parametrit ja paluuarvo

- tod. parametrit sijoitetaan muodollisten parametrien paikalle kutsuhetkellä suoritusaikana
- paluuarvo saadaan paluuhetkellä ja sitä käytetään kuten mitä tahansa arvoa

```
Tulosta (5, apu);  
x = Laske( y+234);
```

# Parametryypit

- **Arvoparametri**
  - välitetään parametrin arvo kutsuhetkellä
  - arvoa ei voi muuttaa
- **Viiteparametri**
  - välitetään parametrin osoite
  - arvo voidaan lukea, arvo voidaan muuttaa
- **Nimiparametri**
  - välitetään parametrin nimi
  - nimi (merkkijono) kuvataan arvoksi kutsuhetkellä
  - semantiikka määräytyy vasta kutsuhetkellä

# Arvoparametri (10)

Tulosta (A+3, B)

- Välitetään todellisen parametrin arvo
  - muuttuja, vakio, lauseke, pointeri, olioviite
- Aliohjelma ei voi muuttaa mitenkään todellisena parametrina käytettyä muuttujaa
  - muuttujan X tai B arvo
  - olioviitteen arvo
  - lausekkeen arvo
  - muuta muodollisen parametrin arvoa aliohjelmassa  
⇒ muutetaan todellisen parametrin arvon kopiota!
  - Todellisen parametrin ptrX arvoa ei voi muuttaa
  - osoitinmuuttujan osoittamaa arvoa voidaan muuttaa
- Javassa ja C:ssä vain arvoparametreja

arvon  
kopio

```
Tulosta (int y, *ptrX);  
{  
    ...  
    y = 5;  
    *ptrX = 10  
}
```

# Viiteparametri (4)

Summaa (54, Sum)

- Välitetään todellisen parametrin osoite  
– muuttujan osoite
- Aliohjelma voi muuttaa parametrina annettua muuttujan arvoa
- Pascalin *var* parametri

pointteri

Vrt. C:ssä arvoparametrina välitetyn osoitinmuuttujan osoittaman arvon (PrtX, ed. kalvo) muuttaminen

```
Summaa (x: int; var cum_sum: int)
{
    ...
    cum_sum = cum_sum + x;
    ...
}
```

Summaa(6, Kok\_lkm)

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

- Välitetään todellisen parametrin nimi
  - merkkijono!
  - Algol 60
  - yleensä makrot
  - sivuvaikutuksia
  - nimiparametri korvataan todellisella parametrilla joka viittauskohdassa

```
void swap (name int x, y)
{
  int t;
  t := x; x := y; y := t;
}
```

Ei käsitellä  
enää jatkossa



```
swap (n, A[n]) % n ↔ A[n]
```

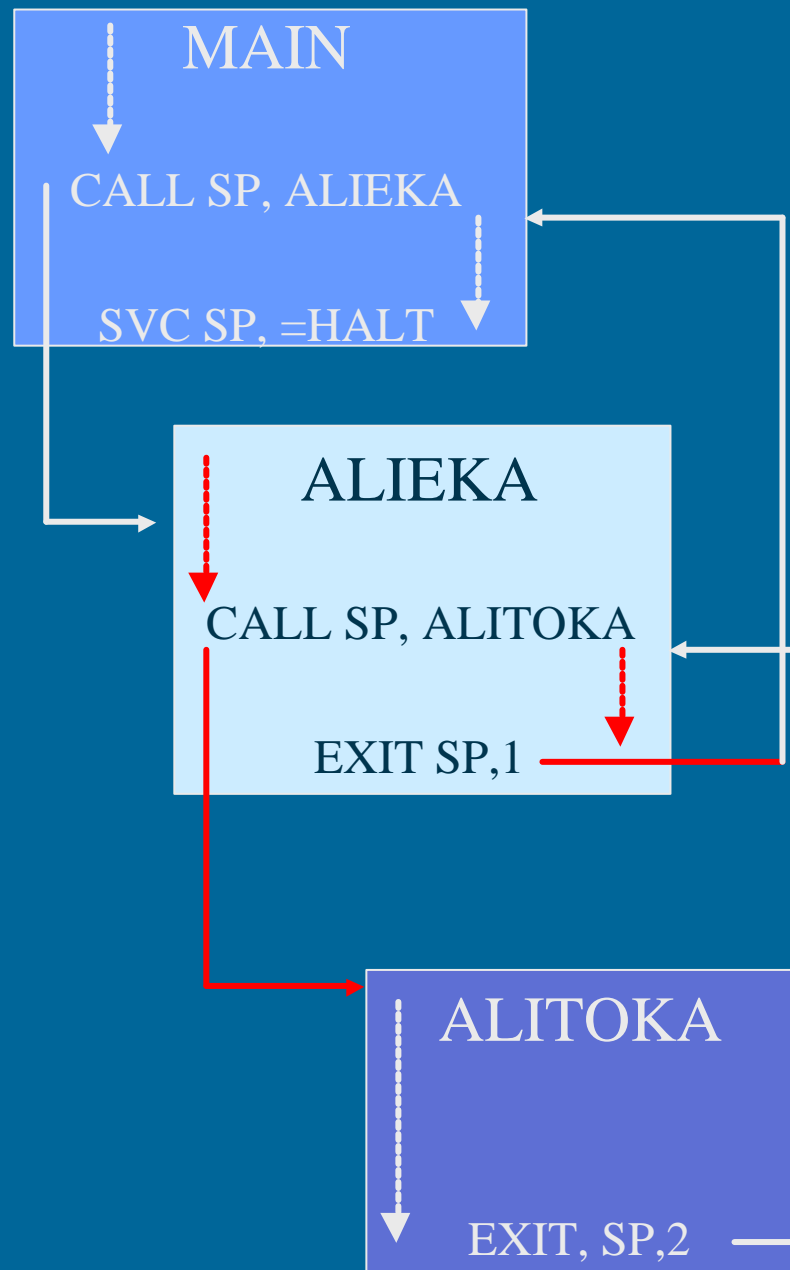
```
t := n; n := A[n]; A[n] := t;
```

väärä n

# Aliohjelmien toteutuksen osat (5)

- Paluuosoite
  - kutsukohtaa seuraava käskyn osoite
- Parametrien välitys
- Paluuarvon välitys
- Paikalliset muuttujat
- Rekistereiden allokointi (varaus)
  - kutsuvalla ohjelman osalla voi olla käytössä rekistereitä, joiden arvon halutaan säilyä!
    - pääohjelma, toinen aliohjelma, sama aliohjelma, metodi, ...
  - käytettyjen rekistereiden arvot pitää aluksi tallettaa muistiin ja lopuksi palauttaa ennalleen





Aliohjelmat voivat olla sisäkkäisiä

- aliohjelman siirtyminen
- aliohjelmasta paluu
- aliohjelman tarvitsemat tiedot



Aliohjelman ympäristö eli **aktivointitietue**

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

(activation record,  
activation frame)

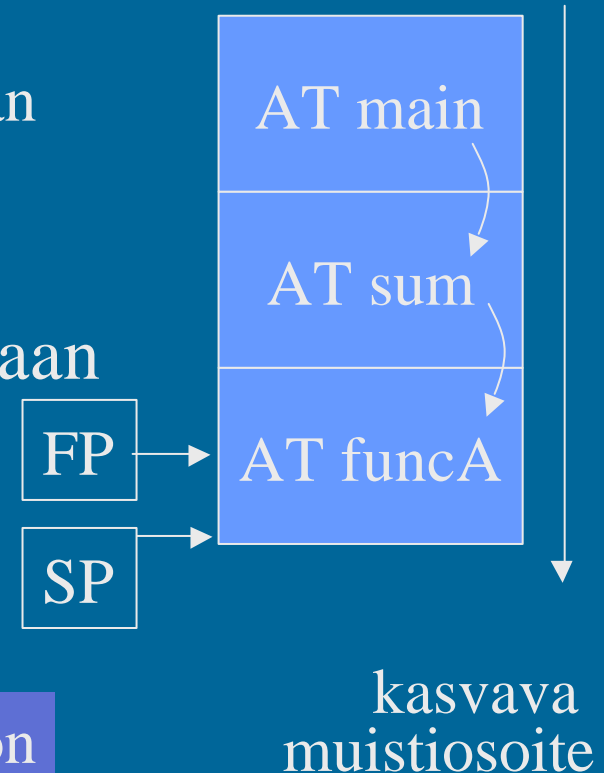
```
int funcA (int x,y);
```

- Aliohjelman toteutusmuoto (ttk-91)
  - kaikkien ulostuloparametrien arvot
  - kaikkien (sisäänmeno) parametrien arvot
  - paluuosoite
  - kutsukohdan aktivointitietue
  - kaikki paikalliset muuttujat ja tietorakenteet
  - aliohjelman ajaksi talletettujen rekistereiden arvot

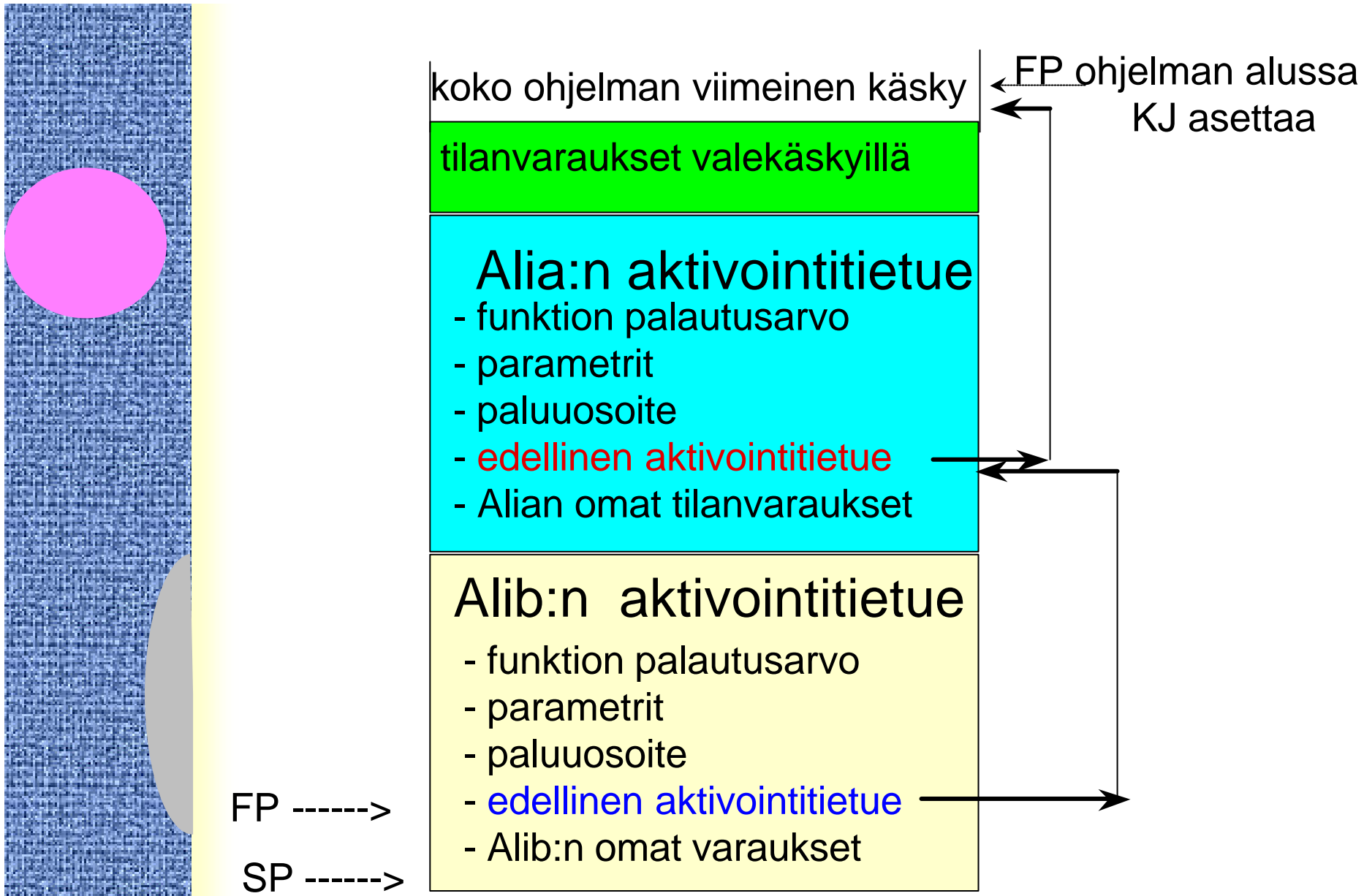


# Aktivointitietueiden hallinta (4)

- Aktivointitietueet (AT) varataan ja vapautetaan dynaamisesti (suoritusaikana) pinosta
  - **SP** (=R6) osoittaa pinon pinnalle
- Aktivointitietuepino
  - **FP** (R7) osoittaa voimassa olevan AT:n sovittuun kohtaan (ttk-91: vanhan FP:n osoite)
- Pinossa olevaa AT:tä rakennetaan ja puretaan käskyillä:
  - PUSH, POP, PUSHR, POPR
  - CALL, EXIT



Talleta R0-R5 pinoon



**Pinon tila ennen Alib:sta poistumista**

# Aliohjelman käytön toteutus (12)

- Toteutus jaettu eri yksiköille

Kutsuva  
rutiini

- varaa tilaa paluuarvolle pinosta
- laita parametrit (arvot tai osoitteet) pinoon

CALL  
käsky

- talleta PC ja FP

Kutsuttu  
rutiini

- talleta käytettyjen rekistereiden arvot pinoon
- varaa tilaa paikallisille muuttujille
- (itse aliohjelman toteutus)

prolog

EXIT  
käsky

- vapauta paikallisten muuttujien tila
- palauta rekistereiden arvot

epilog

Kutsuva  
rutiini

- palauta PC ja FP
- vapauta parametrien tila
- ota paluuarvo pinosta

# Aliohjelmaesimerkki (13)

**käyttö:**

```
int fA (int x, y)
{
    int z = 5;

    z = x * z + y;
    return (z);
}
```

...

```
T = fA (200, R);
```

```
R      DC 24
...
PUSH SP,=0 ; tilaa paluuarvolle
PUSH SP, =200
PUSH SP, R
CALL SP, fA
POP SP, R1
STORE R1, T
...
```

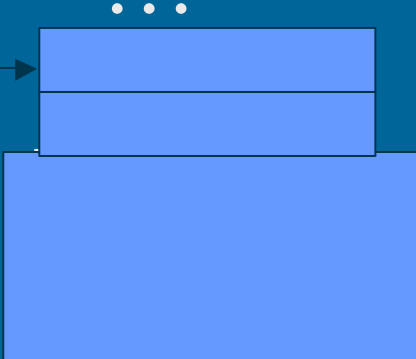
muistista  
muistiin!!

talleta PC, FP  
asetta PC,  
kutsu & paluu  
palauta FP, PC

2. operandi  
aina rekisteri

tämän-  
hetkinen,  
nykyinen  
FP

FP



# Aliohjelmaesimerkki (ei animm)

**käyttö:**

```
int fA (int x, y)
{
    int z = 5;

    z = x * z + y;
    return (z);
}
```

...

T = fA (200, R);

R DC 24

...

PUSH SP,=0 ; tila paluuarvolle

PUSH SP, =200

PUSH SP, R

muistista  
muistiin!!

CALL SP, fA

talleta PC, FP  
asetta PC,  
kutsu & paluu  
palauta FP, PC

POP SP, R1

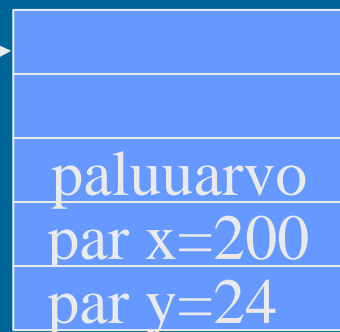
STORE R1, T

2. operandi  
aina rekisteri

...

tämän-  
hetkinen,  
nykyinen  
FP

FP



# Aliohjelma- esimerkki <sup>(11)</sup>

```
int fA (int x, y)
{
    int z = 5;

    z = x * z + y;
    return (z);
}
...
T = fA (200, R);
```

Kaikki viitteet  
näihin tehdään  
suhteessa FP:hen

paluuarvo

aliohjelman toteutus:

```
retfA EQU -4 ; paluuarvo
parX EQU -3 ; 1. param. = x
parY EQU -2 ; 2. param. = y
locZ EQU 2 ; paikall. muutt
```

```
fA PUSH SP, =0 ; alloc Z
   PUSH SP, R1 ; save R1
```

```
LOAD R1, =5; init Z
STORE R1, locZ (FP)
```

```
LOAD R1, parX (FP)
MUL R1, locZ (FP)
ADD R1, parY (FP)
```

```
STORE R1, retfA (FP)
SUB SP, =1 ; free Z
```

```
POP SP, R1; recover R1
```

```
EXIT SP, =2 ; 2 param.
```

prolog

epilog



# Aliohjelma esimerkki

aliohjelman toteutus:

```
int fA (int x, y)
{
    int z = 5;

    z = x * z + y;
    return (z);
}
...
T = fA (200, R);
```

```
retfA EQU -4
parX EQU -3
parY EQU -2
locZ EQU 2
```

ks. fA.k91

```
fA PUSH SP, =0 ; alloc Z
   PUSH SP, R1 ; save R1
```

```
LOAD R1, =5; init Z
STORE R1, locZ (FP)
```

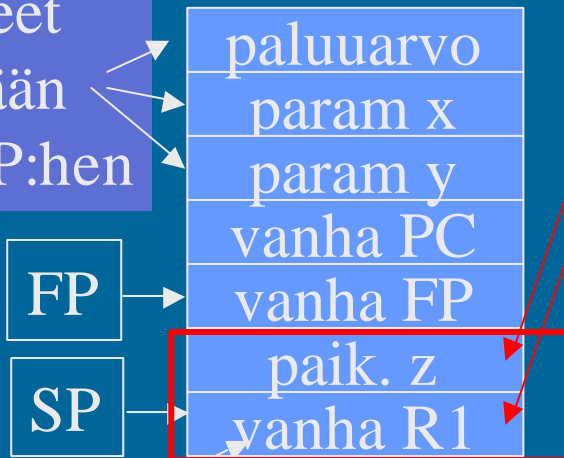
```
LOAD R1, parX (FP)
MUL R1, locZ (FP)
ADD R1, parY (FP)
```

```
STORE R1, retfA (FP)
SUB SP, =1 ; free Z
POP SP, R1; recover R1
EXIT SP, =2 ; 2 param.
```

prolog

epilog

Kaikki viitteet  
näihin tehdään  
suhteessa FP:hen



# Viiteparametri esimerkki (2)

(Pascal)

```
procB (x, y: int, var pZ:int)
{
    pZ = x * 5 + y;
    return;
}
```

...

```
procB (200, R, T);
```

käyttö:

```
...
PUSH SP, =200
PUSH SP, R
PUSH SP, =T ; T's address!
```

```
CALL SP, procB
```

```
; T has new value
```

...

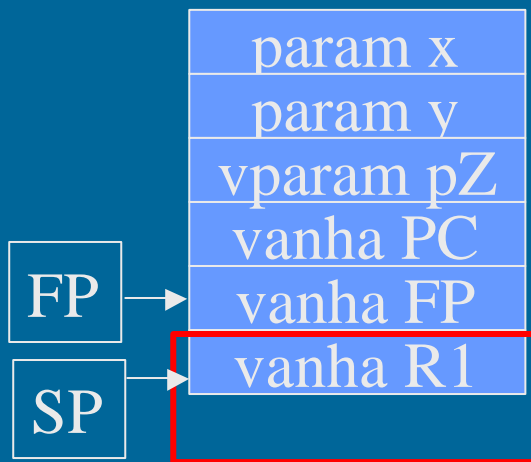
Ei välitetä taulukkoa T (ja sen kaikkia alkioita),  
vaan ainoastaan T:n osoite (yksi arvo)

# Viiteparam. (jatk) <sup>(1)</sup>

```
procB (x, y: int, var pZ:int)
{
    pZ = x * 5 + y;
    return;
}
```

...

```
procB (200, R, T);
```



aliohjelman toteutus:

```
parX EQU -4 ; relative to FP
parY EQU -3
parpZ EQU -2
```

```
procB PUSH SP, R1 ; save R1
```

```
LOAD R1, parX (FP)
```

```
MUL R1, =5
```

```
ADD R1, parY (FP)
```

```
STORE R1, @parpZ (FP)
```

```
POP SP, R1; restore R1
```

```
EXIT SP, =3 ; 3 param.
```

prolog

epilog

ks. procB.k91

# Aliohjelma kutsuu funktiota (1)

itse aliohjelman  
käyttö kuten ennen:

```
procC (x, y: int, var pZ:int)
{
  pZ = fA(x,y);
  return;
}
```

```
...
procC (200, R, T);
```

```
...
PUSH SP, =200
PUSH SP, R
PUSH SP, =T ; T's address
```

```
CALL SP, procC
```

```
; T has new value
```

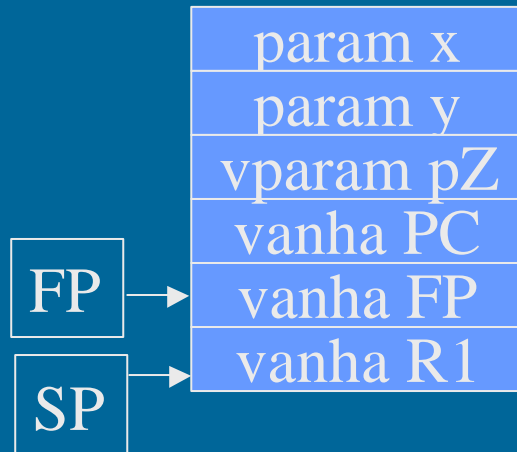
```
...
```

# Aliohjelma kutsuu funktiota (2)

aliohjelman toteutus:

```
procC (x, y: int, var pZ:int)
{
  pZ = fA(x,y);
  return;
}
...
procC (200, R, T);
```

AT kuten ennen:



```
parXc EQU -4 ; relative to FP
parYc EQU -3
parpZ EQU -2
```

ks. procC.k91

```
procC PUSH SP, R1 ; save R1
      ; call fA(parXc, parYc)
```

```
PUSH SP,=0 ; ret. value
PUSH SP, parXc(FP)
PUSH SP, parYc(FP)
```

```
CALL SP, fA
```

```
POP SP, R1
```

```
STORE R1, @parpZ (FP)
```

```
POP SP, R1; restore R1
EXIT SP, =3 ; 3 param.
```

# Rekursiivinen aliohjelma (4)

- Aliohjelma, joka kutsuu itseään
- Ei mitään erikoista muuten
- Aktivointitietue hoitaa tilanvarauksen automaattisesti paikallisille muuttujille joka kutsukerralla
  - Rekursio ei onnistu, jos paikallisten muuttujien tilanvaraus on aliohjelman ohjelmakoodin yhteydessä
    - jotkut Fortran-versiot
- Joka kutsukerralla suoritetaan sama koodi-alue (aliohjelman koodi), mutta dataa varten on käytössä oma aktivointitietue

# Rekursio esimerkki (1)

```
fPow (n: int)
{
  if (n=1)
    return (1);
  else
    return (n * fPow (n-1));
}
...
k = fPow (4);
```

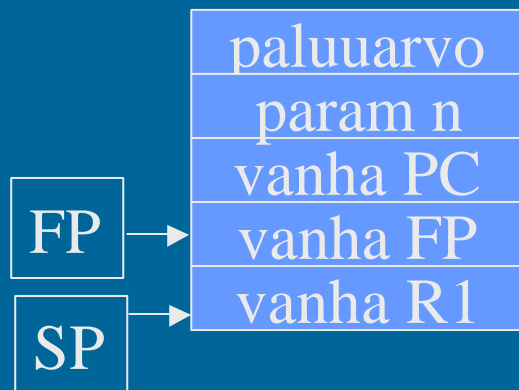
kutsu:

```
K      DC  0
; k = fPow (4)
PUSH SP, =0
PUSH SP, =4
CALL SP, fPow
POP  SP, R1
STORE R1, K
```

# Rekursio

## toteutus (2)

```
fPow (n: int)
{
  if (n=1)
    return (1);
  else
    return (n * fPow (n-1));
}
...
k = fPow (4);
```



```
parRet EQU -3
parN EQU -2
```

```
fPow PUSH SP, R1 ; save R1
```

```
LOAD R1, parN(FP)
COMP R1,=1
JEQU One ; return 1 ?
```

```
; return fPow(N-1) * N
SUB R1, =1 ; R1 = N-1
PUSH SP, =0 ; ret. value space
PUSH SP, R1
CALL SP, fPow
POP SP, R1 ; R1 = fPow(N-1)
```

```
One MUL R1, parN(FP)
STORE R1, parRet(FP)
```

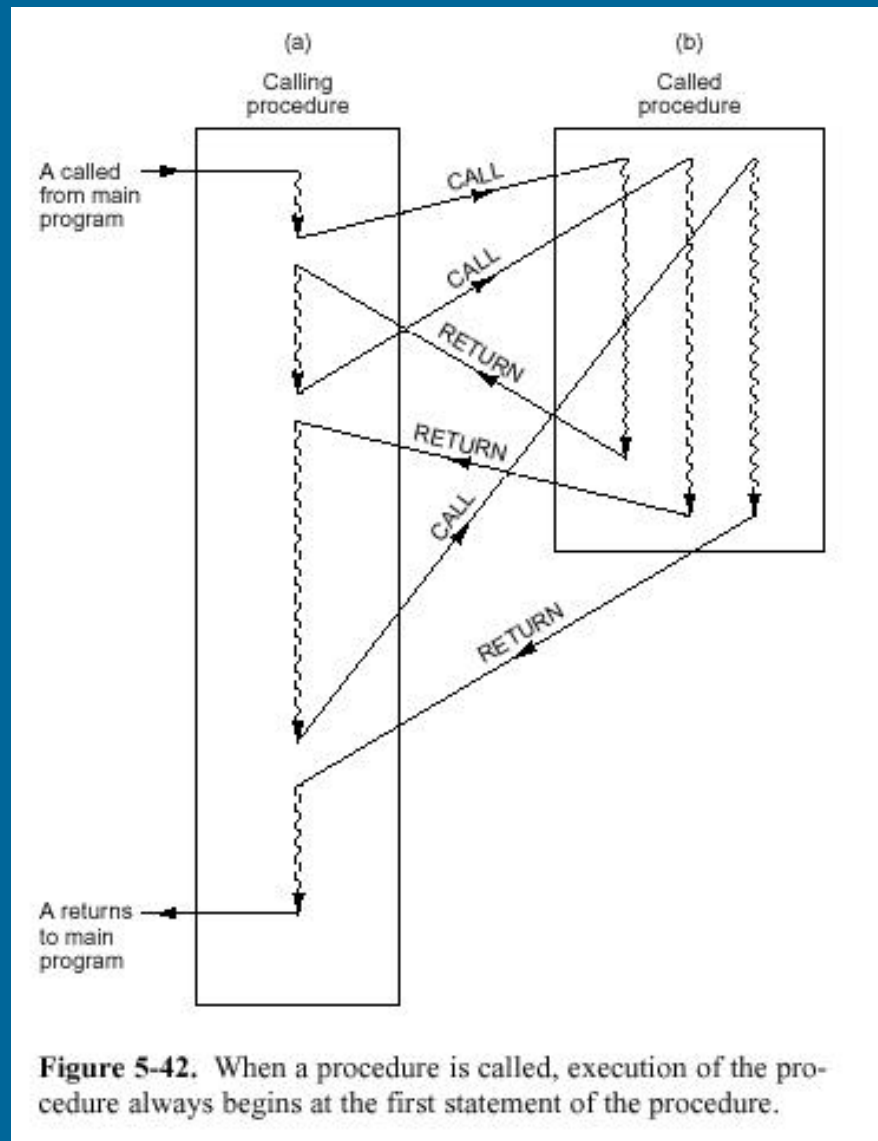
```
POP SP, R1; restore R1
EXIT SP, =1 ; 1 param.
```



# KJ-palvelun kutsu

- Samalla tavalla kuin aliohjelman kutsu
  - CALL-käskyn asemasta SVC
- Tilaa paluuarvolle?
- Parametrit pinoon
- SVC-kutsu
- IRET-paluu
- Paluuarvo (OK, virhe) pois pinosta tarkistusta varten

# -- Jakson 4 loppu --



[Tane99]