Runtimesystemer   Kap 7 - I

- Generell lagerorganisering (7.1)
- Språk som bare trenger statisk allokering (7.2)
- Språk som trenger stakk-orientert allokering (7.3)
- Språk som trenger mer generell allokering (7.4)
- Parameteroverføring (7.5)

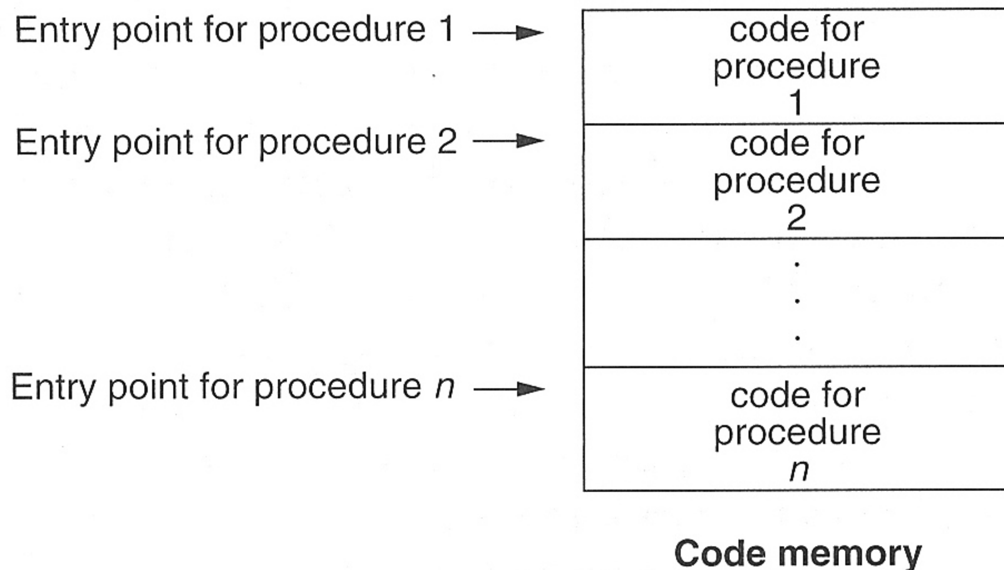
Avhenger av begrepene i språk

INF 5110 - 2014

Den oversatte programkoden

- kan nesten altid betraktes som statisk allokert
- skal hverken flyttes eller forandres under utførelse

- Kompilatoren kjenner alle adresser til kodebiter

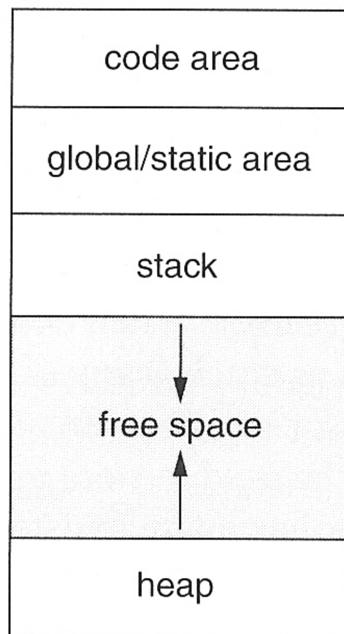


Men husk

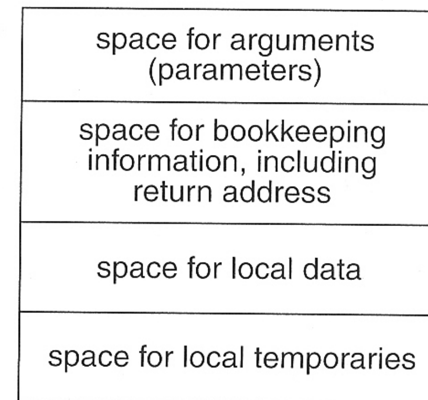
Koden blir ofte produsert som relokerbare kode, som får sin endelige plassering av linker/ loader

Lagerorganisering

- Typisk organisering under utførelse dersom et programmeringsspråk har alle slags data (statisk, stakk, dynamisk)



- Typisk organisering av data for et prosedyrekall (aktiveringsblokk)



Det er gjerne ut fra plasseringen her man karakteriserer språk til være
- statisk organisert
- stakk-organisert
- heap/dynamisk organisert

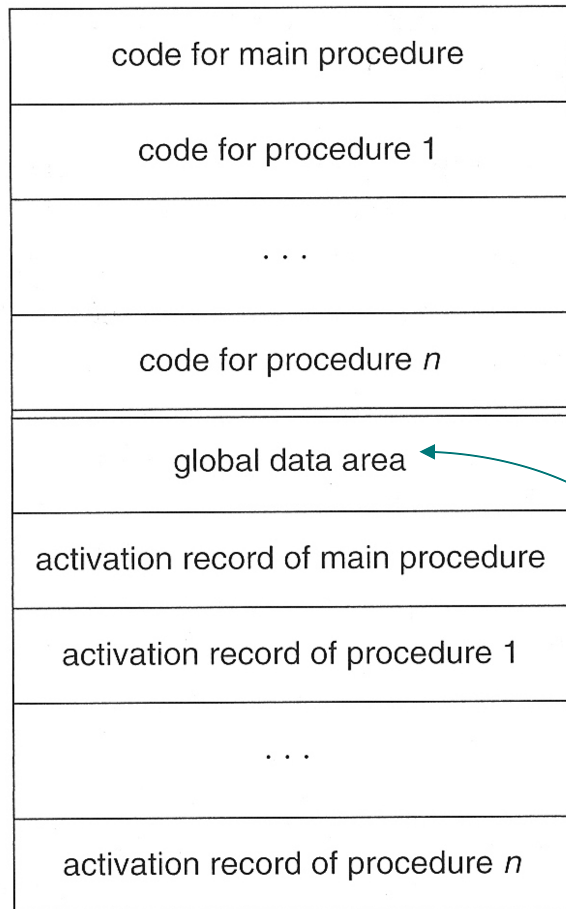
INF 5110 - 2014

7.2

STATISK ORGANISERING

INF 5110 - 2014

Full statisk organisering (eks. Fortran)



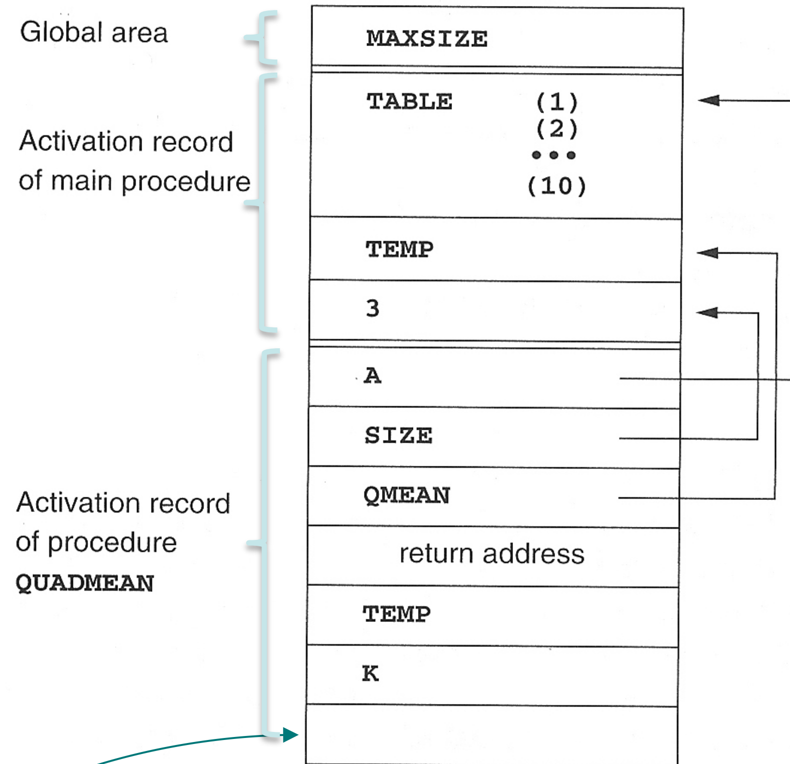
- Kompilatoen kan beregne hvor alt ligger
  - Utførbar kode
  - Variable
  - Alle slags hjelpedata

bl.a. alle slags større konstanter i programmet

INF 5110 - 2014

Et eksempel i Fortran

```
      PROGRAM TEST
      COMMON MAXSIZE
      INTEGER MAXSIZE
      REAL TABLE(10),TEMP
      MAXSIZE = 10
      READ *, TABLE(1),TABLE(2),TABLE(3)
      CALL QUADMEAN(TABLE,3,TEMP)
      PRINT *, TEMP
      END

      SUBROUTINE QUADMEAN(A,SIZE,QMEAN)
      COMMON MAXSIZE
      INTEGER MAXSIZE,SIZE
      REAL A(SIZE),QMEAN, TEMP
      INTEGER K
      TEMP = 0.0
      IF ((SIZE.GT.MAXSIZE).OR.(SIZE.LT.1)) GOTO 99
      DO 10 K = 1,SIZE
         TEMP = TEMP + A(K)*A(K)
10    CONTINUE
99    QMEAN = SQRT(TEMP/SIZE)
      RETURN
      END
```



Plass til mellomresultater o.l. Kompilatoren kan beregne hvor mye som trengs

I Fortran overføres parametere som pekere til de aktuelle verdier/variable

INF 5110 - 2014

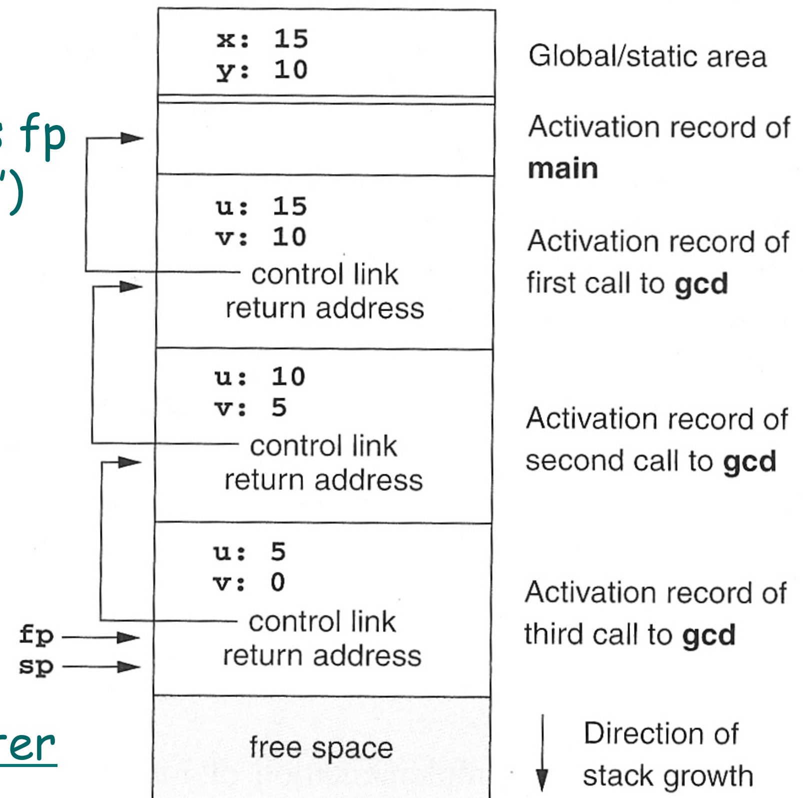
7.2

STAKK-ORGANISERING

Et eksempel i C

```c
#include <stdio.h>

int x,y;

int gcd( int u, int v)
{ if (v == 0) return u;
  else return gcd(v,u % v);
}

main()
{ scanf("%d%d",&x,&y);
  printf("%d\n",gcd(x,y));
  return 0;
}
```

- Aktiverings-blokkene organiseres som en stakk.



return address
program-adressen man er kalt fra

control link
angir kallerens fp ('dynamisk link')

fp - frame pointer
peker på fast sted i den aktuelle aktiveringsblokkken

sp -stack pointer
angir grensen mellom brukt og ledig lagerareal

Variabel-aksess

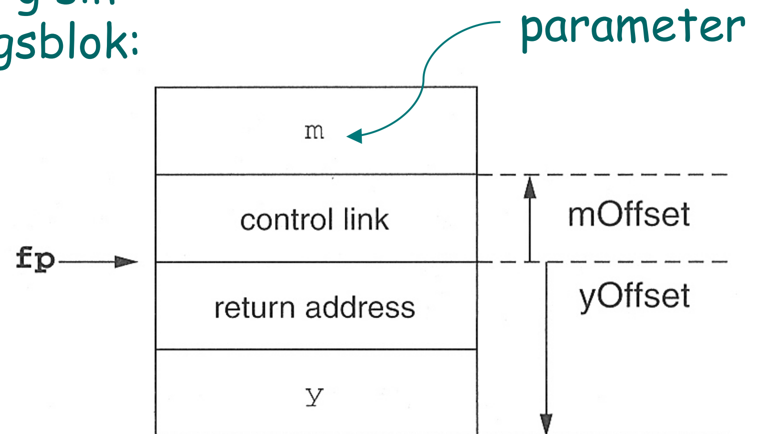
```
int x = 2;

void g(int); /* prototype */

void f(int n)
{ static int x = 1;
  g(n);
  x--;
}
```

```
void g(int m)
{ int y = m-1;
  if (y > 0)
  { f(y);
    x--;
    g(y);
  }
}
```

```
main()
{ g(x);
  return 0;
}
```

Blir én global variabel bare synlig fra f

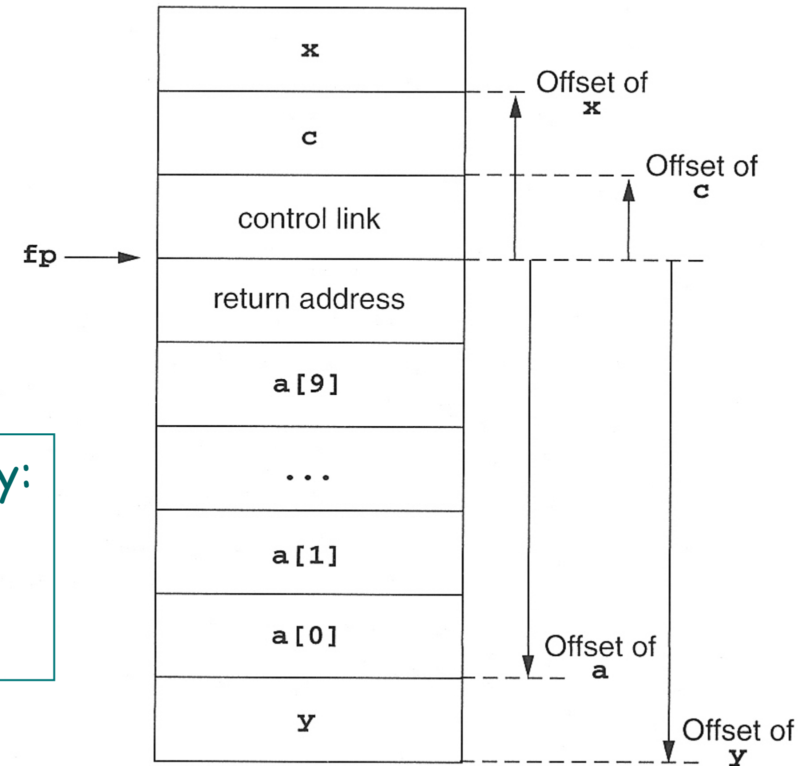
Layout av g sin aktiveringsblok:



Arrayer av kjent (statisk) lengde

```
void f(int x, char c)
{ int a[10];
  double y;
  ...
}
```

Layout av aktiveringsblokk:



Relativ-adresser

Name	Offset
x	+5
c	+4
a	−24
y	−32

Aksess av c og y:

c: 4(fp)
y: -32(fP)

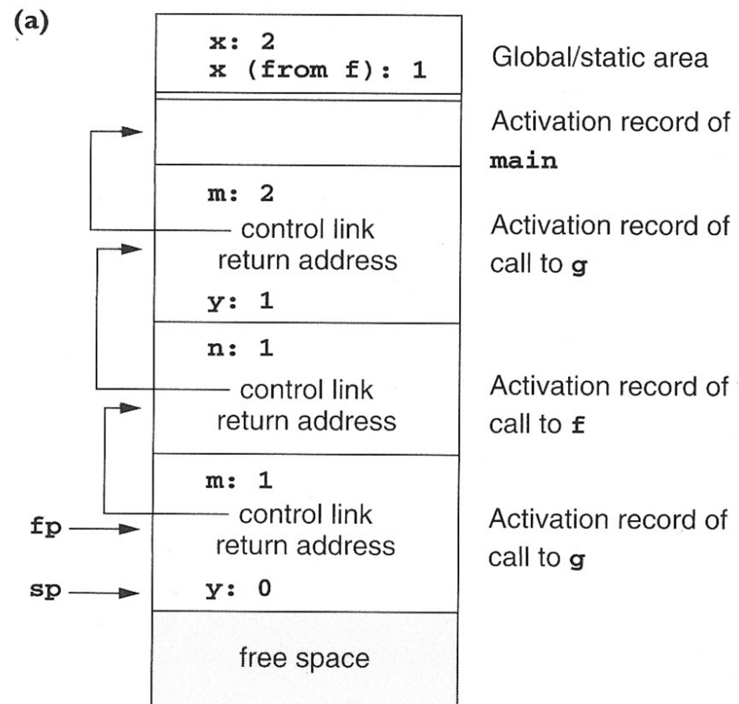
A[i] beregnes som adressen

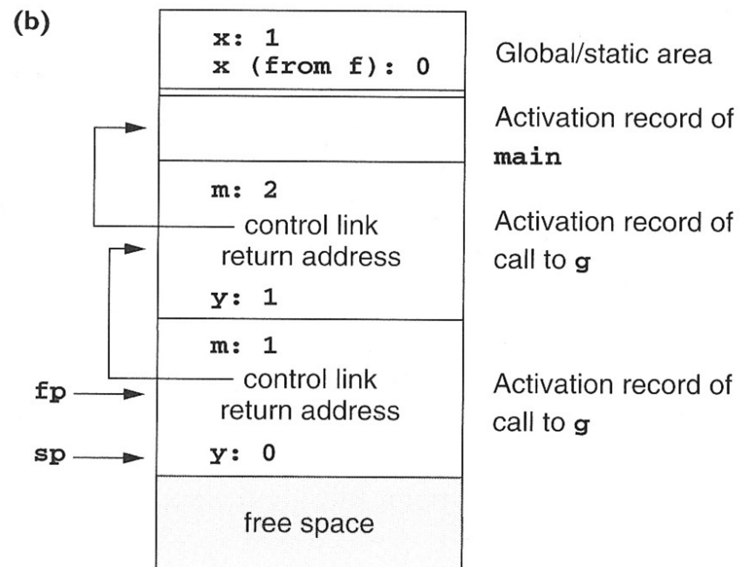
(-24 + 2*i)(fp)

kan ofte gjøres i én instruksjon

INF 5110 - 2014

```
int x = 2;

void g(int); /* prototype */

void f(int n)
{ static int x = 1;
  g(n);
  x--;
}

void g(int m)
{ int y = m-1;
  if (y > 0)
  { f(y);
    x--;
    g(y);
  }
}


main()
{ g(x);
  return 0;
}
```





Hvordan utføre et kall

- Ved prosedyrekall (entry)

1. Compute the arguments and store them in their correct positions in the new activation record of the procedure (pushing them in order onto the runtime stack will achieve this).
2. Store (push) the fp as the control link in the new activation record.
3. Change the fp so that it points to the beginning of the new activation record (if there is an sp, copying the sp into the fp at this point will achieve this).
4. Store the return address in the new activation record (if necessary).
5. Perform a jump to the code of the procedure to be called.

cl:= fp;
fp:= sp;
set return addr

allocate locals
by changing sp

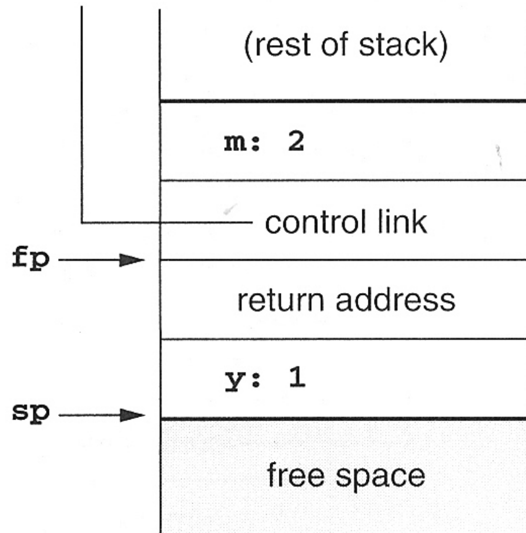
- Ved prosedyre-exit

1. Copy the fp to the sp.
2. Load the control link into the fp.
3. Perform a jump to the return address.
4. Change the sp to pop the arguments.

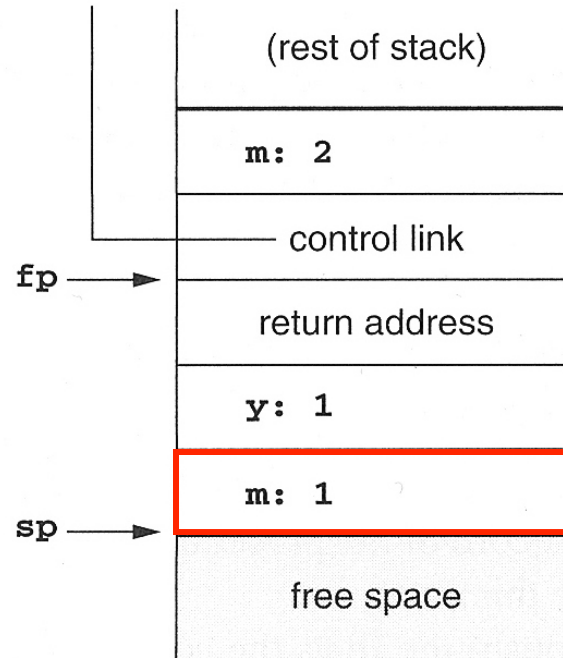
sp:= fp;
fp:= cl;

deallocate by
changing sp

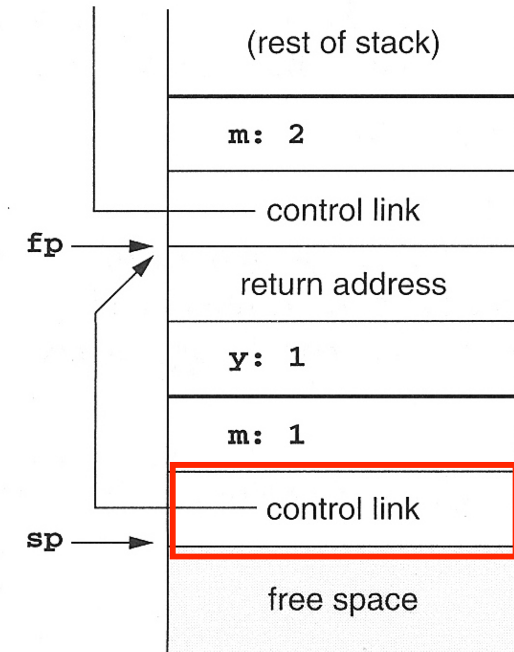
Gjennomføring av et kall - I



før kall på g



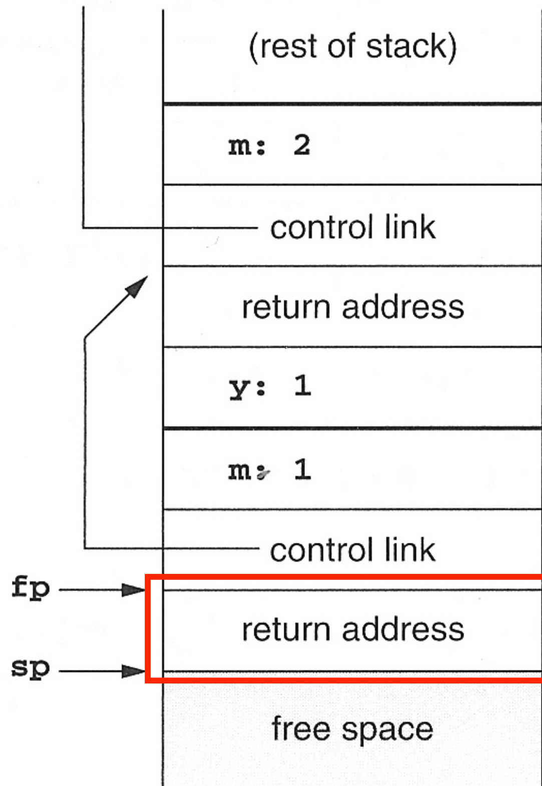
push parameter



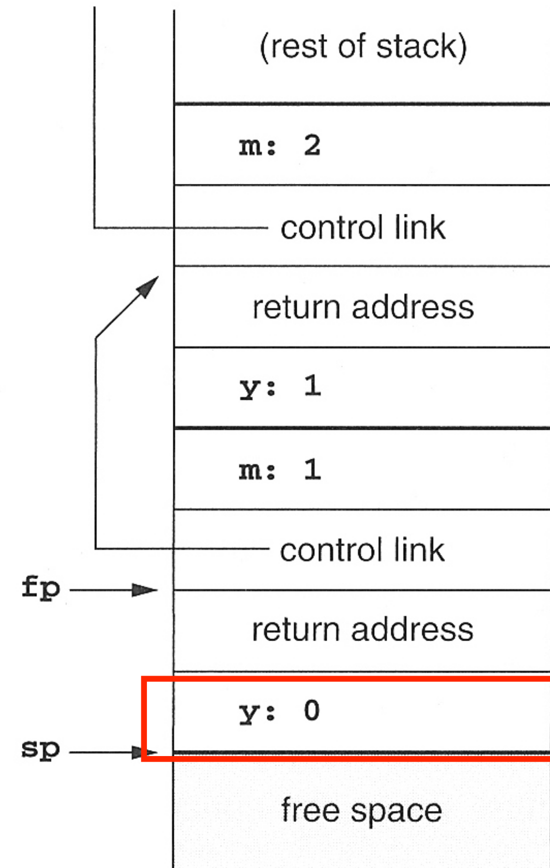
push fp

INF 5110 - 2014

Gjennomføring av et kall – II



1. fp = sp
2. Push returadressse



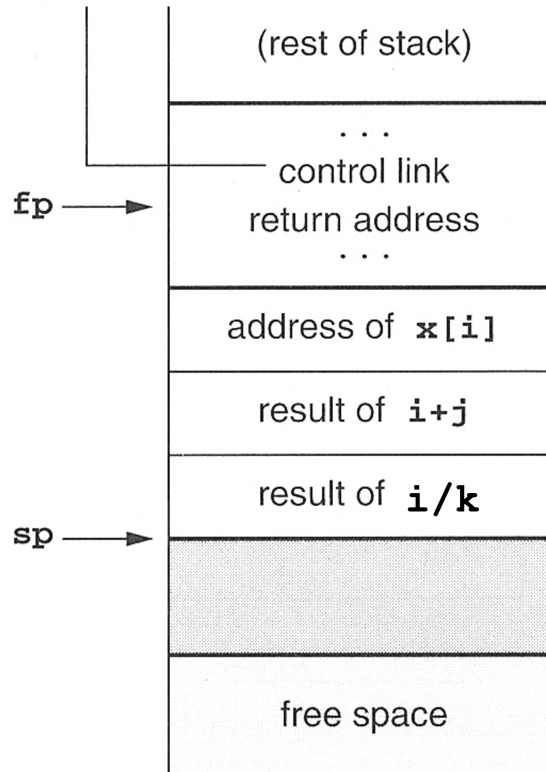
alloker lokal var (y)

Behandling av mellomresultater

```
x[i] = (i + j)*(i/k + f(j))
```

adresse verdi verdi

Antar strikt beregning fra venstre mot høyre. Kallet f(j) kan forandre verdier.



Activation record of procedure containing the expression

Stack of temporaries

New activation record of call to f (about to be created)

Trenger ikke sette av fast maksimal plass til slike mellomresultater for hele blokkens levetid. I modsetning til hva man naturlig gjør i Fortran.

Data av variabel lengde

- 'variabel' betyr at data ikke har samme størrelse ved hvert kall

```
type Int_Vector is
      array(INTEGER range <>) of INTEGER;

procedure Sum (low,high: INTEGER;
                    A: Int_Vector) return INTEGER
is
   i: integer
begin
  ...
end Sum;
```

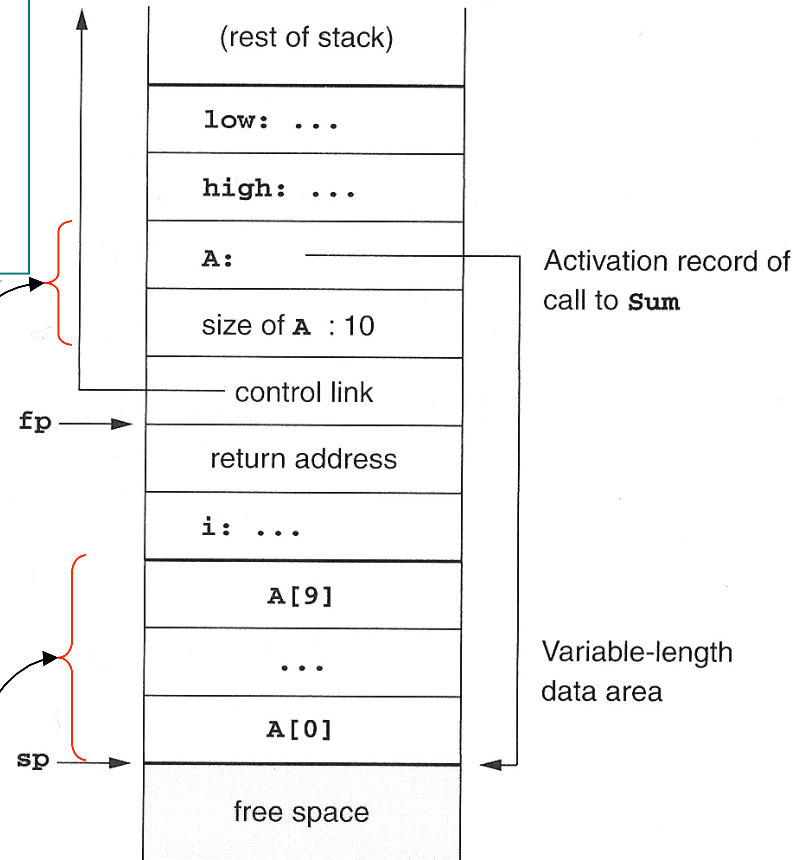
antar at A overføres ved full kopiering

A[i] beregnes som

@6(fp) + 2*i

Fast lengde

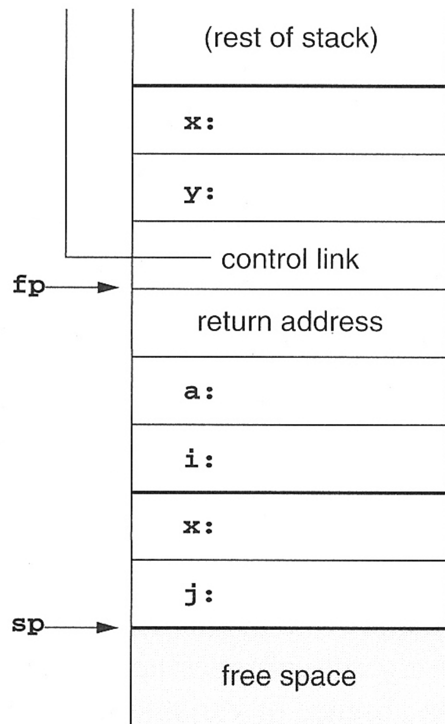
I Java legges disse på heapen



INF 5110 - 2014

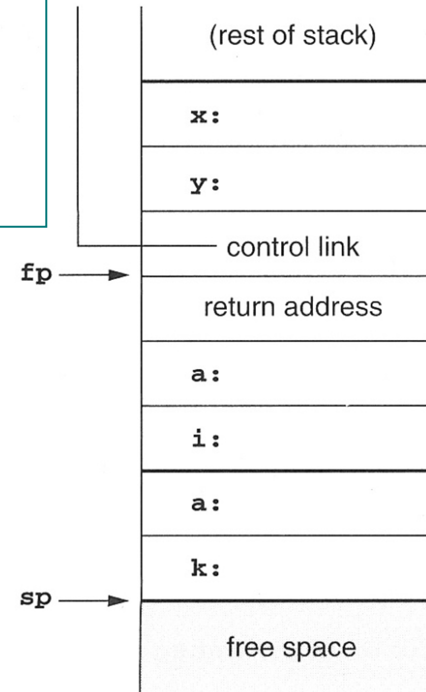
Mulig plass-allokering ved 'indre blokker'

```
void p( int x, double y)
{ char a;
  int i;
  ...
A:{ double x;
    int j;
    ...
  }
  ...
B:{ char * a;
    int k;
    ...
  }
  ...
}
```



Activation record of call to P

Allocated area for block A



Activation record of call to P

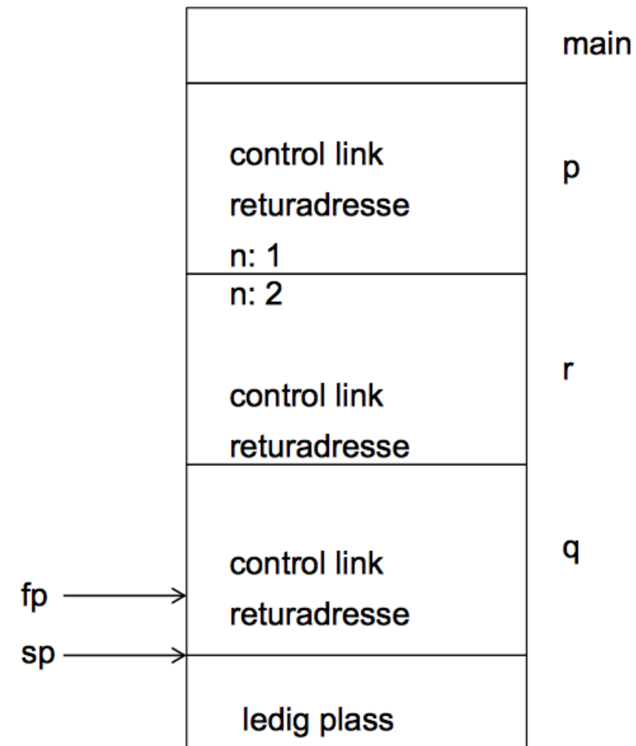
Allocated area for block B

Prosedyrer inne i prosedyrer

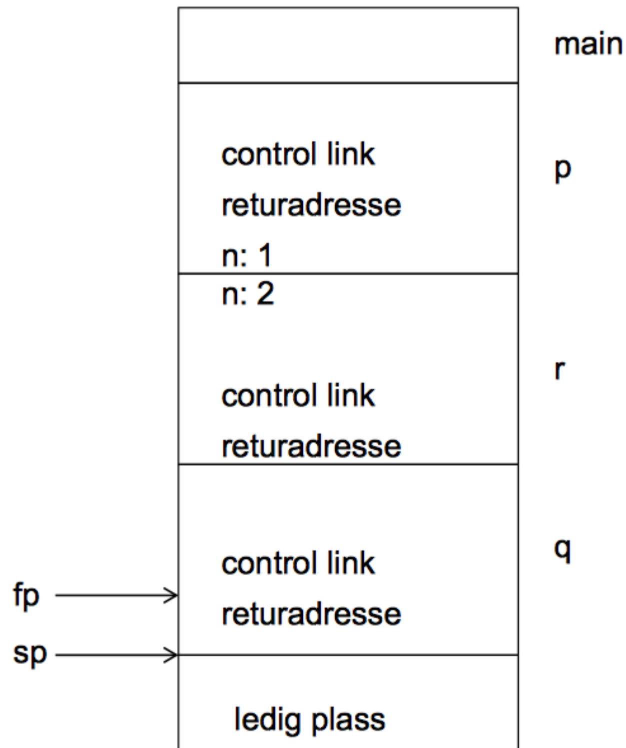
```
program nonLocalRef;

procedure p;
var n: integer;

      procedure q;
      begin
        (* a reference to n is now
          non-local non-global *)
      end; (* q *)


      procedure r(n: integer);
      begin
        q;
      end; (* r *)

begin (* p *)
  n := 1;
  r(2);
end; (* p *)

begin (* main *)
  p;
end.
```

Hvordan kan vi aksessere ’n’ i ’p’ under utførelse av q ?

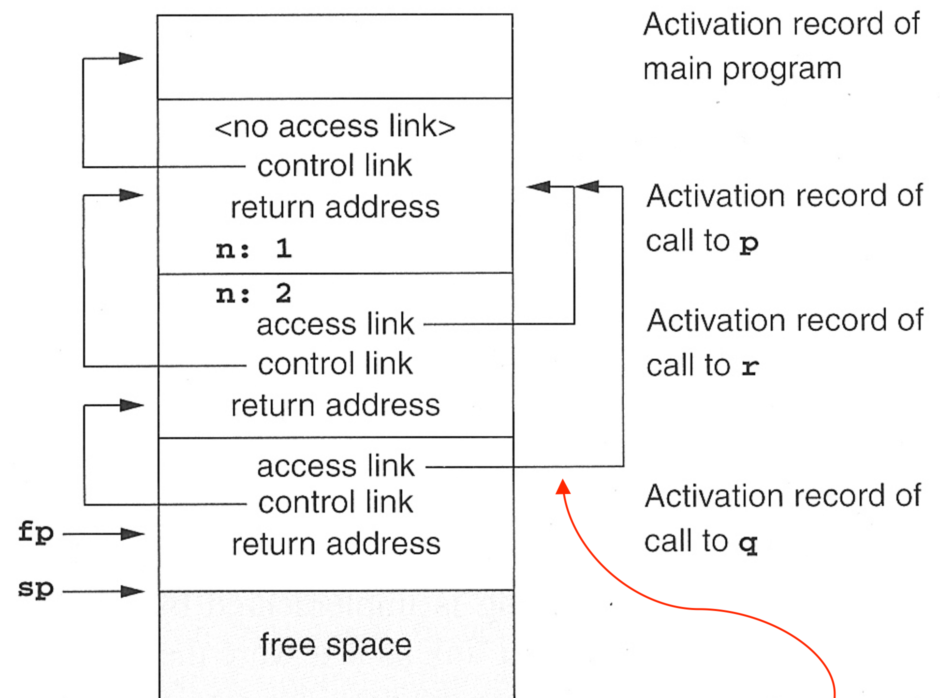
Et første forsøk:



Det første forsøk



Vi trenger noe ekstra
(<u>aksess-link/statisk link</u>)

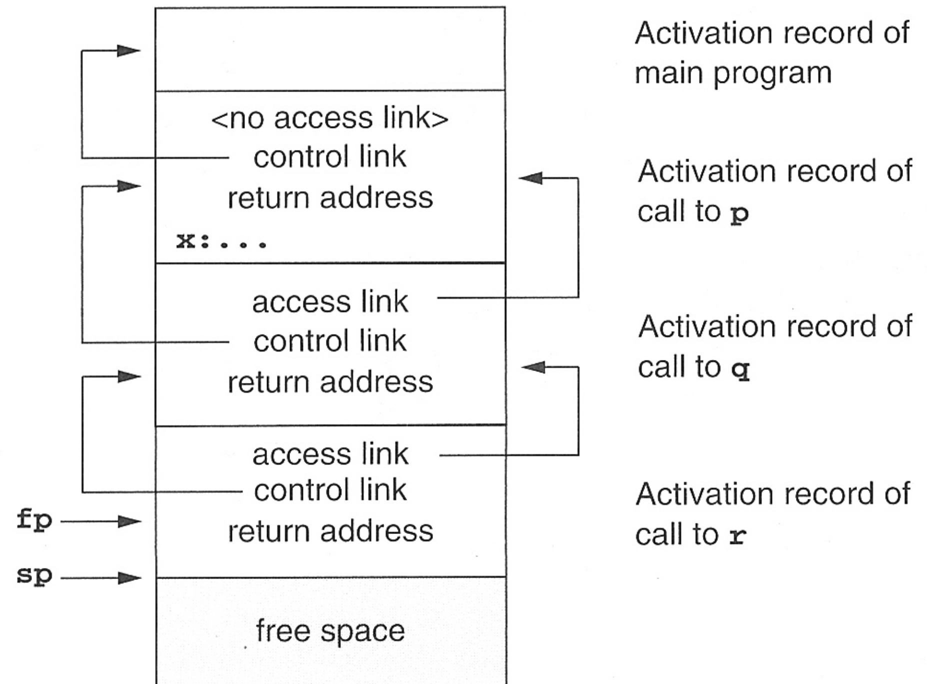


Går altid til aktuell utgave av tekstlig omgivelse

Eksempel med flere nivåer

```
program chain;

procedure p;
var x: integer;

   procedure q;
      procedure r;
      begin
         x := 2;
         ...
         if ... then p;
      end; (* r *)
   begin
     r;
   end; (* q *)

begin
  q;
end; (* p *)

begin (* main *)
  p;
end.
```

Program-blokkene får da et blokk-nivå



display

0	
1	
2	
	.

fp.al.al.x

diff i blokknivå

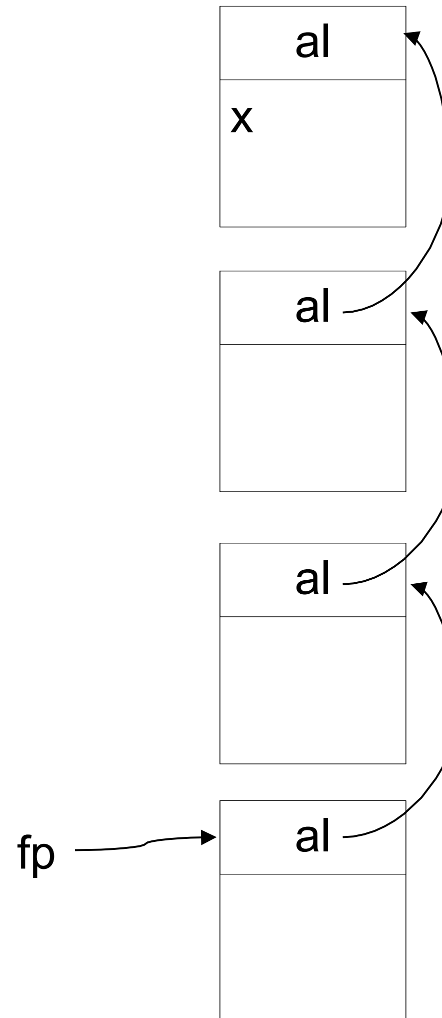
Implementasjon av fp.al.al.al. ... al.x

Antar at fp ligger fast i et register

4(fp) -> reg
4(reg) -> reg
...
4(reg) -> reg

diff i blokknivå

X kan nå aksesseres som 6(reg)



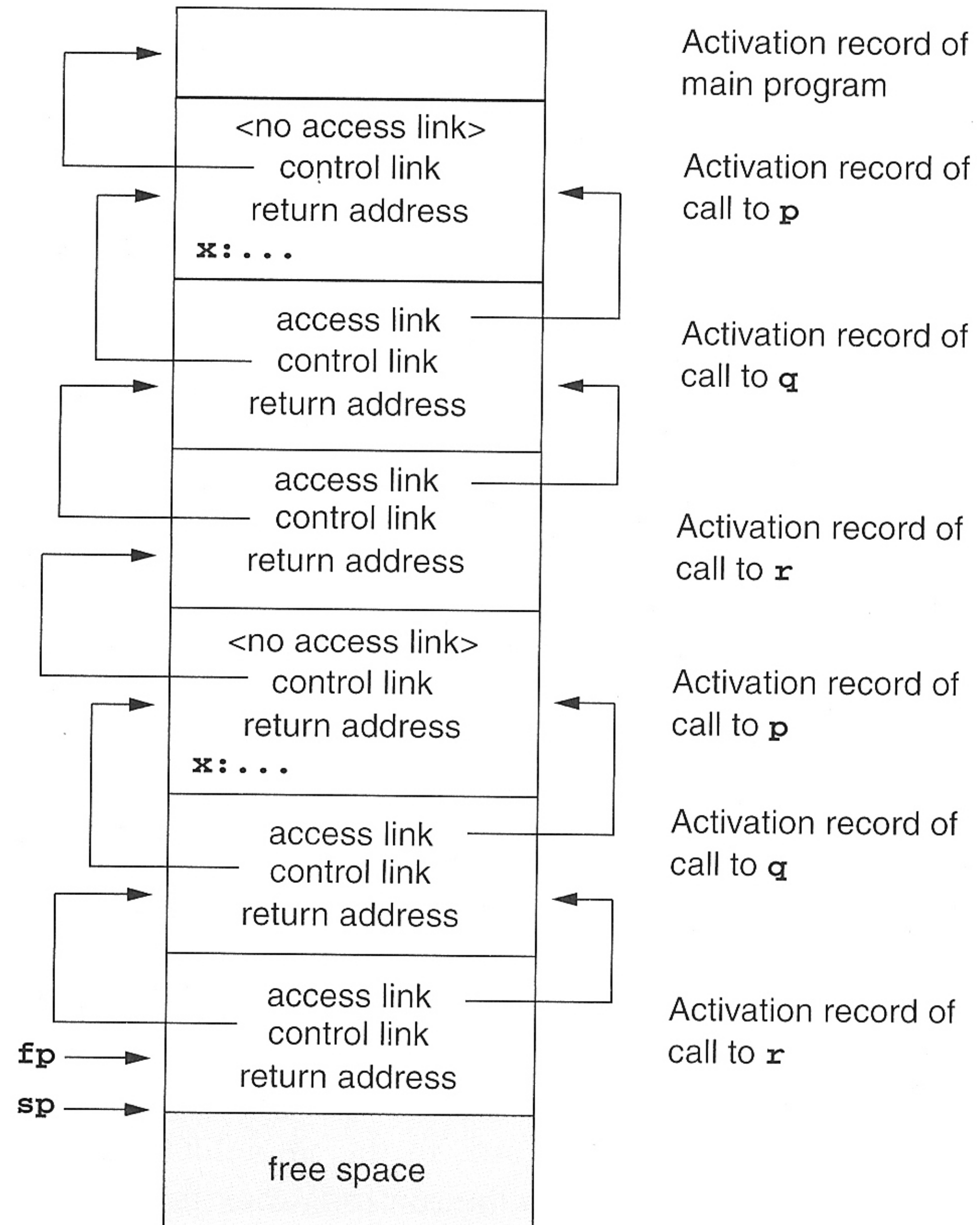
Ofte ikke så mange blokknivåer

Videre utførelse

Hvordan skaffe acces-link ved kall? Kalleren vet hvor den er, og utfører

ny aksess-link = fp.al.al....

(så mange som nivåforskjellen er)



Hva om vi skal ha 'access-link'?

- Ved prosedyrekall (entry)

  1. Compute the arguments and store them in their correct positions in the new activation record of the procedure (pushing them in order onto the runtime stack will achieve this).
  2. Store (push) the fp as the control link in the new activation record.
  3. Change the fp so that it points to the beginning of the new activation record (if there is an sp, copying the sp into the fp at this point will achieve this).
  4. Store the return address in the new activation record (if necessary).
  5. Perform a jump to the code of the procedure to be called.

1. Beregn ny aksess-link som ny-al = fp.al.al ... (tilsvarende diff. i blokknivå mellom den kalte og kalleren – er 0 om den kalte er lokal i kalleren)
2. Push ny-al på stakken

- Ved prosedyre-exit

  1. Copy the fp to the sp.
  2. Load the control link into the fp.
  3. Perform a jump to the return address.
  4. Change the sp to pop the arguments.

+ aksess-link

INF 5110 - 2014