F5: Högnivåprogrammering

- **Parameteröverföring**
- **Koppling mellan låg- och högnivåprogrammering**
  - **Lokala variabler**
  - **Heapen**
  - **Datatyper**

Mittuniversitetet
MID SWEDEN UNIVERSITY

Subrutin, parameteröverföring: 1(3)

- **Via register genom värde**
  - **Skicka data via ett register**
  - **Resultat = KALLE(Indata)**

```
...
MOVE.W Indata,D0
BSR KALLE
MOVE.W D0,Resultat
...
```

D0 innehåller värdet av funktionens inparameter

- **Via register genom referens**
  - **Skicka referens till data via ett register**
  - **Resultat = KALLE(&Indata)**

```
...
LEA Indata,A0
BSR KALLE
MOVE.W D0,Resultat
...
```

A0 innehåller pekaren till funktionens inparameter

Subrutin, parameteröverföring: 2(3)

- **Via minnesposition och genom värde**
  - **Skicka data via en global minnesposition (global variabel)**
  - **Transfer = Indata; KALLE(); Resultat = Transfer;**

```
...
MOVE.W    Indata,Transfer
BSR       KALLE
MOVE.W    Transfer,Resultat
...
```

Minnesposition Transfer tilldelas värdet av funktionens inparameter.

- **Via minnesposition och genom referens**
  - **Skicka referens till data via en global minnesposition (global variabel)**
  - **Transfer = &Indata; KALLE(); Resultat = Transfer;**

```
...
MOVE.L    #Indata,Transfer
BSR       KALLE
MOVE.W    Transfer,Resultat
...
```

Minnesposition Transfer tilldelas en referens till funktionens inparameter

Subrutin, parameteröverföring, 3(3)

- **Via stacken**
  - **Skicka data via stacken**

```
...
MOVE.W Indata,-(A7)
BSR KALLE
MOVE.W (A7)+,Resultat
...
```

```
KALLE:  MOVE.W (4,A7),Indata
        ...
        MOVE.W Resultat,(4,A7)
        RTS
```

- **Sammanfattning**
  - **Via register**
    - + Snabb
    - - Klarar bara ett fåtal parametrar
  - **Via minnesposition**
    - + Relativ snabb
    - - Inte så flexibelt, ostrukturerat
  - **Via stacken**
    - + Mycket flexibel
    - - Långsam

Att tänka på när ni använder subrutiner

- **För att underlätta återanvändning av kod bör:**
  - **Alla register ska vara opåverkade efter exekveringen av subrutinen**
    - **Utom register som används för parameter överföring från subrutin till huvudprogram.**
  - **Spara registren ni vill ska vara opåverkade efter subrutinen på stacken**
- **Välj den parameteröverföring som passar den aktuella applikationen**
  - **Register**
  - **Minnesposition**
  - **Via stacken**

Assembler vs. Högnivåspråk

- **När ska man använda assembler?**
  - **Optimering**
    - **Programmering av processorer med speciella arkitekturer**
      - **T ex DSP-processorer**
    - **Programmering av objekt, där det är hårda krav på minnesandvändning och/eller exekveringstid.**
  - **Behov av assemblerinstruktioner**
    - **Utveckling av realtids/operativsystemkärnor**
      - **Behöver använda stacken och interrupthantering**
    - **Utveckling av kompilatorer**
      - **Instruktioner för att implementera högnivåkod**

Högnivåspråk parameteröverföring i C

Parameteröverföringen i en C-funktion

- Beroende av kompilator
  - På stacken
    - Värde
    - Referens (dvs. pekare)
  - Via register (oftast returvärden)
- Globala variabler kan också användas
  - Användning bestäms av programmeraren

Högnivåspråk, C

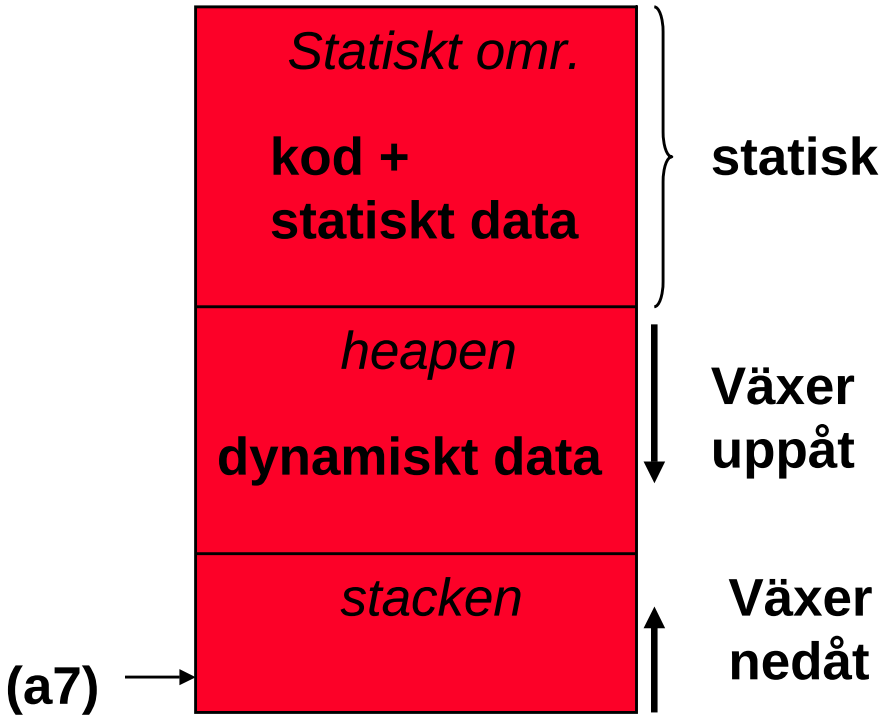
- C kod är (ska) vara oberoende av implementationen
  - Kompileras till assemblerkod
  - Assemblerkoden översätts till maskinkod
- Koden är oberoende av vart i minnet den exekveras
- Stöder rekursion

```
void baklanges(char *enstring) {  // skriver ut en textsträng baklänges
   char c;
   c = *enstring;
   if( c != null)
   {
      baklanges(enstring+1);  //anropa baklanges igen med nästa tecken
      typeToScreen(c) ;       //skriv tecknet till skärmen
   }
}
```

Om c lagras i en fast minnesposition kommer data att ändras för varje ny exekvering i den rekursive kedjan.

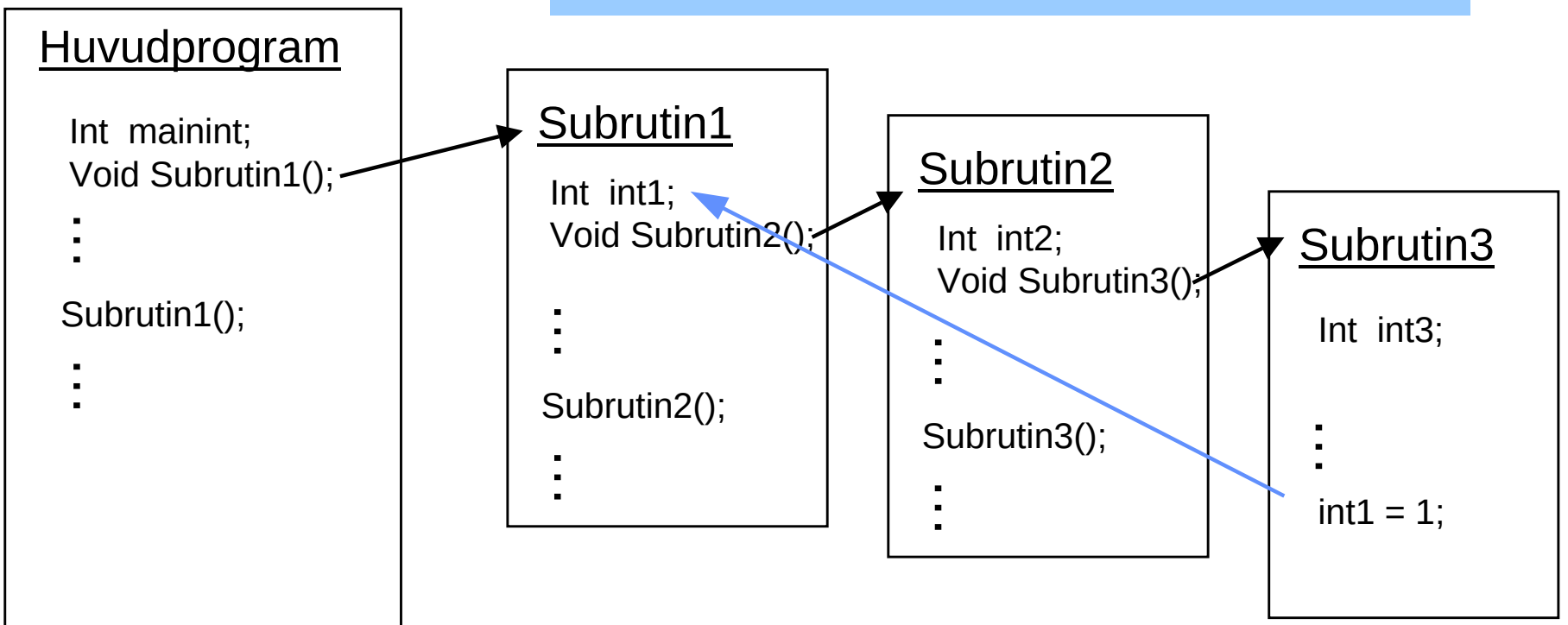
C funktioner

- **C funktioner**
  - **Koden lagras i någonstans i minnet (statiskt)**
  - **Lokala variabler lagras på stacken (dynamiskt)**
    - **Möjliggör rekursion**



Lokala variabler i högnivåspråk

Subrutin1 är deklarerad i huvudprogrammet
Subrutin2 är deklarerad i Subrutin1 osv



Hur kan som i ex. Pascal, en subrutin få tillgång till en variabel som deklarerats flera nivåer tidigare i kedjan av funktionsanrop?

Allokera plats för lokala variabler på stacken

- **För att reservera plats på stacken för lokala variabler används instruktionsparet:**
  - **LINK An, offset**
    - Skapar en länkad lista med instanser av funktionsanrop på stacken
      - **Spar An på stacken**
      - **An tilldelas värdet av stackpekaren**
      - **Stegar stackpekare framåt med ett antal steg som bestäms av offset**
  - **UNLK An**
    - Tar bort en instans av en funktion
- **Adressregistret A6 används ofta av kompilatorer för att hanterad lokala variabler på stacken.**

Allokera plats för lokala variabler

LINK A6, #-8

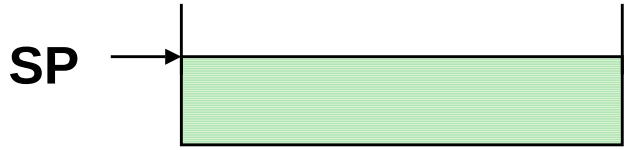
.

.

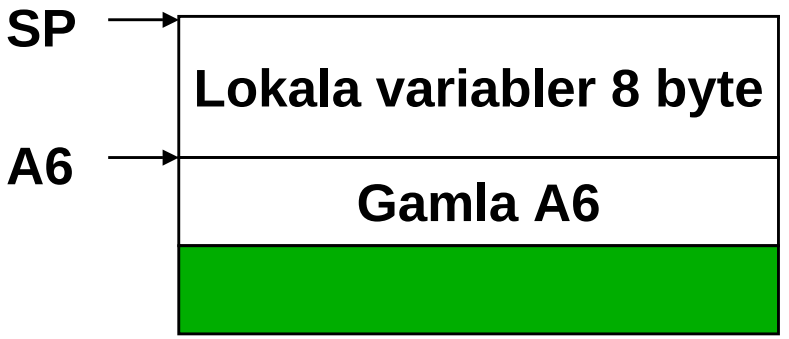
.

UNLK A6

SP-4 -> SP
A6 -> (SP)
SP -> A6
SP-8 - SP



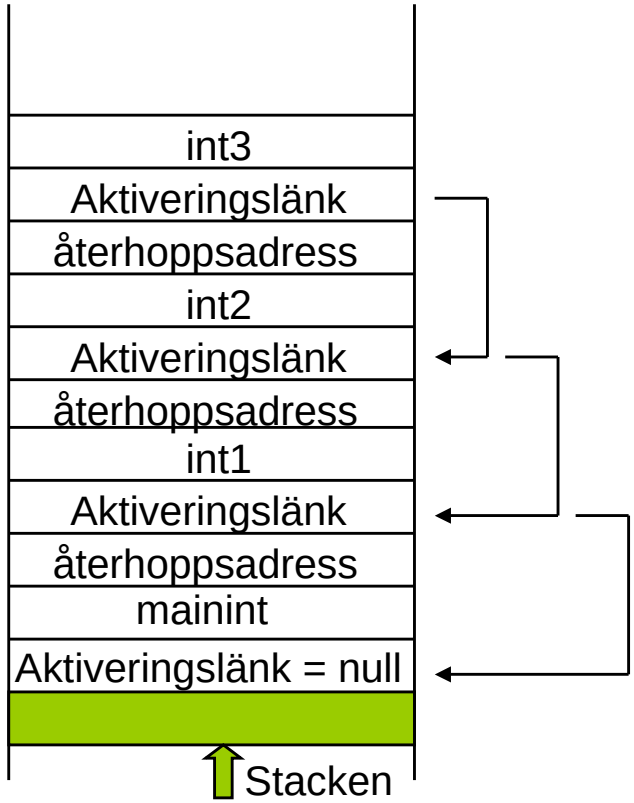
Stacken före LINK



Stacken efter LINK

Mittuniversitetet
MID SWEDEN UNIVERSITY

Länkad lista av aktiveringsposter



Konstruera med aktiveringsposter 1(3)

Uppg) Implementera i assembler parameteröverföringen samt reservera utrymme för lokala variabler motsvarande följande C-funktion:

```
void prototype( longint param) {
    longint a=0;
    char c = 'k';

}
```

Inget returvärde

Initiering

Utför även initieringen av de lokala variablerna. Du får själv välja lagringsordning för de lokala variablerna. Parametern skall överföras via stacken genom värde.
Visa även genom anrop till subrutinen hur parametern tilldelas ett värde före anrop samt åtkomst av parametern i subrutinen.

Konstruera med aktiveringsposter 2(3)

Sekvens i anropande subrutin

```
MOVE.L   #$11111111,-(SP)          *Parameter = 11111111 HEX till stacken
BSR      prototype                 *Call prototype(11111111 Hex)
ADDQ.L   #4,SP                     *Ta bort parameter från stacken
STOP     #0                        *Stanna exekvering
```

Reservera utrymme för närmast högre heltal bytes i de fall det aktuella lagringsbehovet är udda antal bytes. Dvs 5 blir 6 i detta fall. (Adressen till 16 bits eller 32 bits ord måste nämligen alltid vara jämn.)

Subrutin

```
prototype:

  LINK   A6,#-6              *Skapa utrymme på stacken för lokala variabler
  MOVE.B #107,(-1,A6)        *Initiera lokal variabel b = 'k'
  CLR.L  (-6,A6)             *Initiera lokal variabel a = 0

*       Här skrivs nu implementationen av subrutinen
*       Åtkomst av inparametern sker genom effektiva adressen (14,SP)
*       Till ex. MOVE.L (14,SP),D0 kopierar inparametern till D0

  UNLK   A6                  *Släpp allokerat utrymme på stacken
  RTS
```

Konstruera med aktiveringsposter 3(3)

Subrutin

```
prototype:
LINK    A6,#-6
MOVE.B  #107,(-1,A6)  * b
CLR.L   (-6,A6)       * a

UNLK    A6
RTS
```

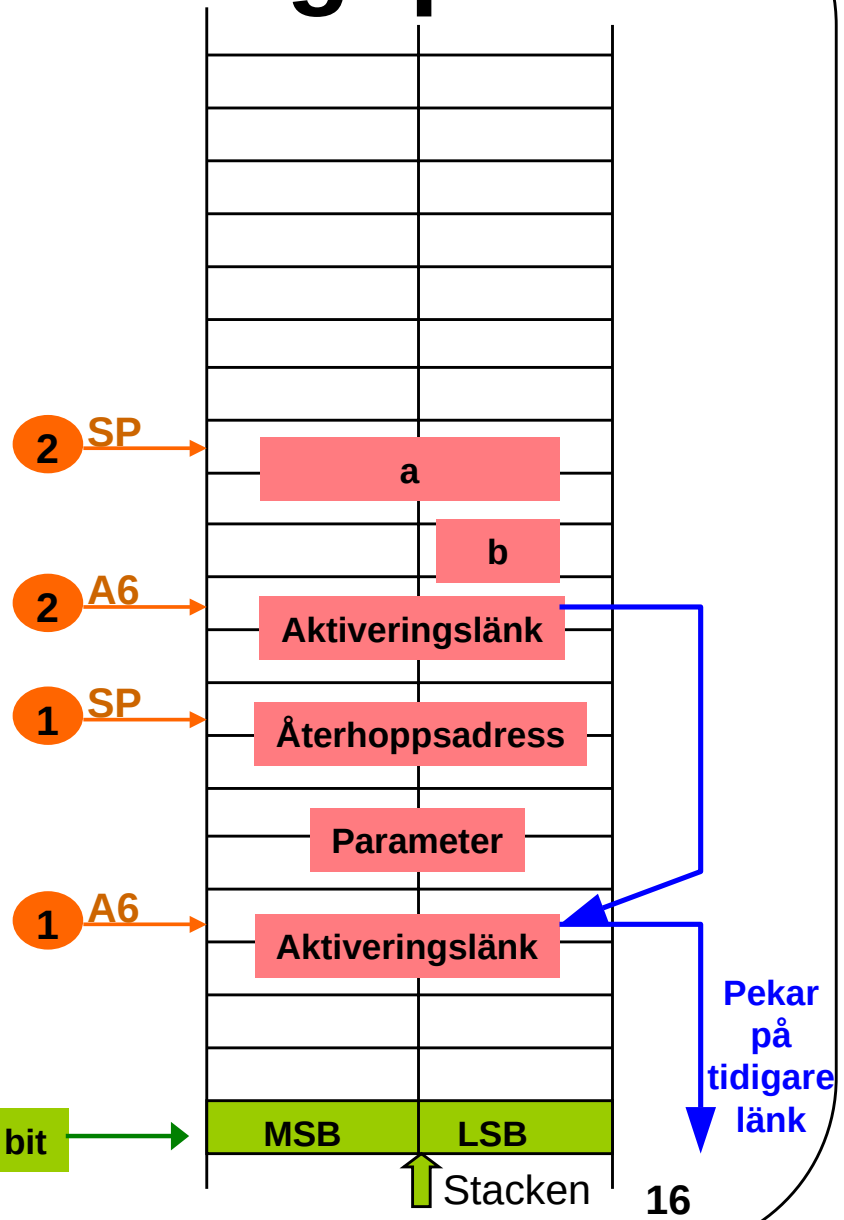
1 → LINK A6,#-6

2 → MOVE.B #107,(-1,A6) * b

SP-4 -> SP
A6 -> (SP)
SP -> A6
SP-6 - SP

1 → UNLK A6

A6 -> SP
(SP) -> A6
SP+4 -> SP

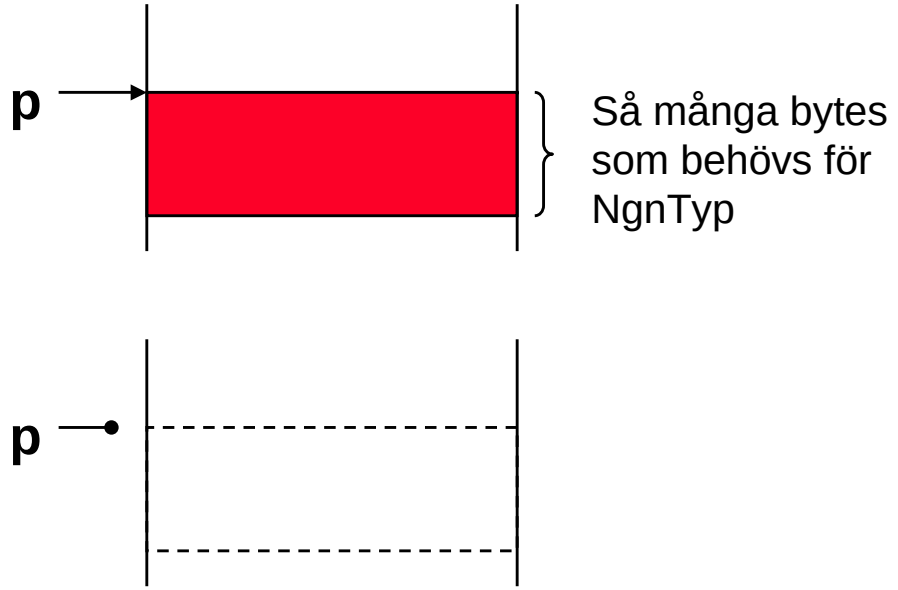


Observera att varje rad är 16 bit

Heapen

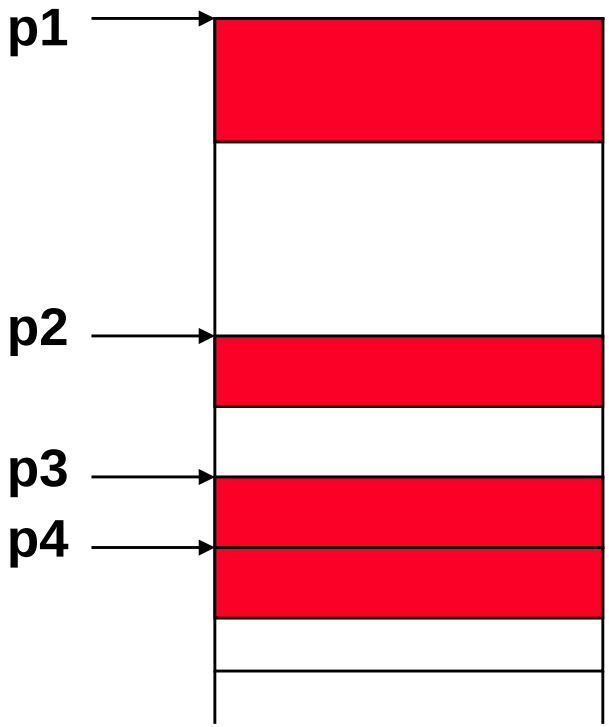
- **Betyder hög**
- **Heapen växer mot höga adresser**

```
NgnTyp *p;
...
 p = new NgnTyp;
...
delete p;
```



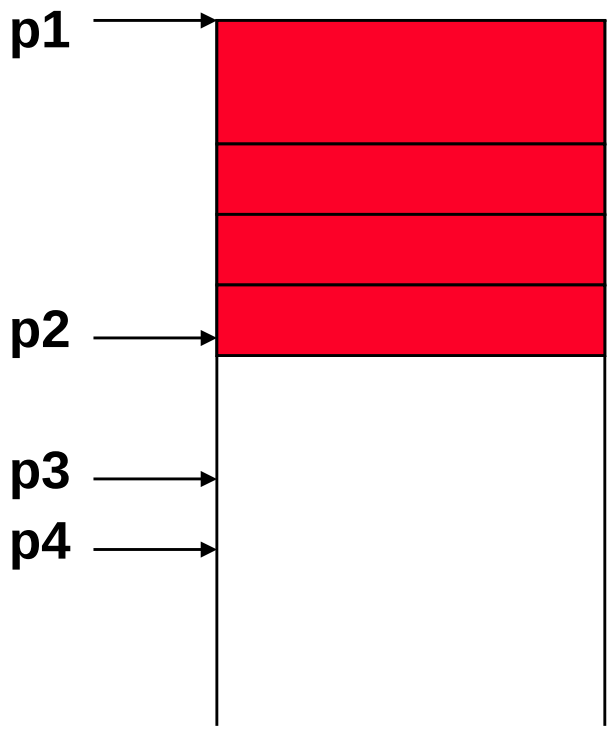
Fragmentering av heapen

- **När block frigörs kommer luckor att uppstå i heapen**
  - **Heapen blir fragmenterad**



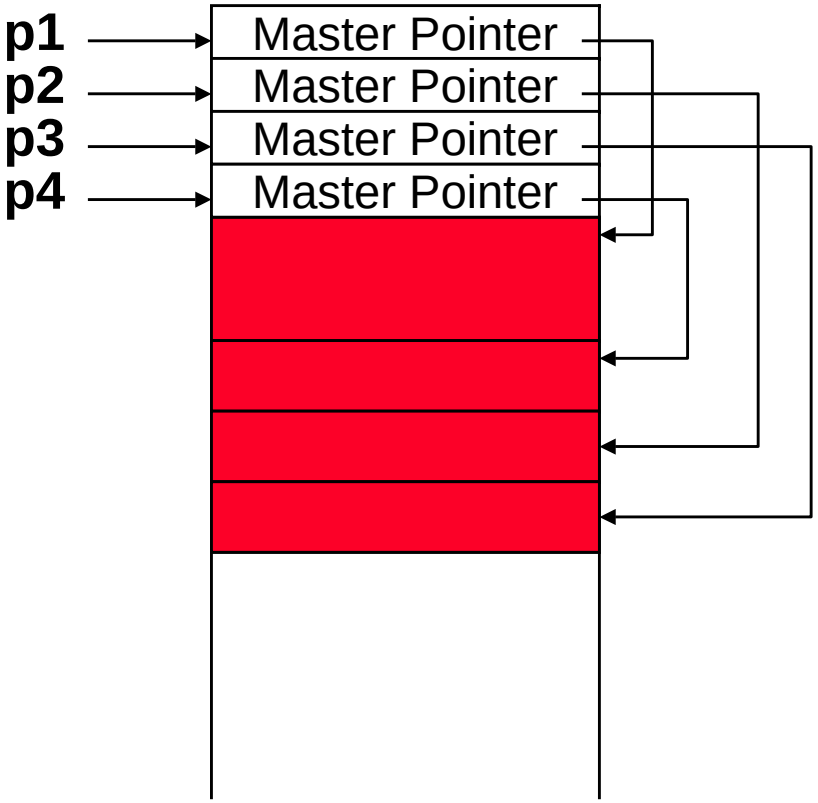
Defragmentering av heapen 1(3)

- **Genom att sortera kommer systemet utnyttja minnet bättre**
  - **Problem: Pekarna kommer att peka på fel position i minnet**

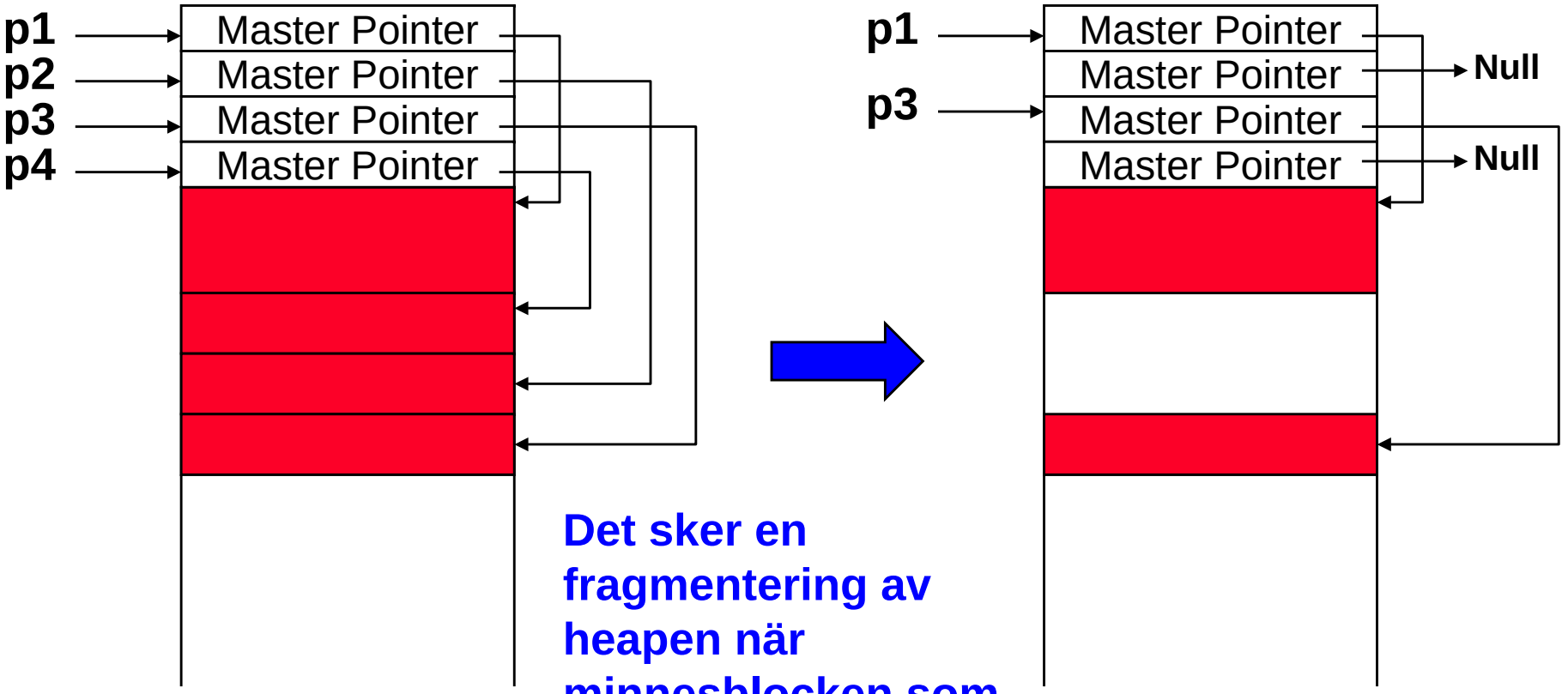


Defragmentering av heapen 2(3)

- **För att kunna flytta block i minnet, används masterpekare som referens till blocket**
  - **Pekaren som användaren använder(handtag), är en pekare till pekare (MP)**

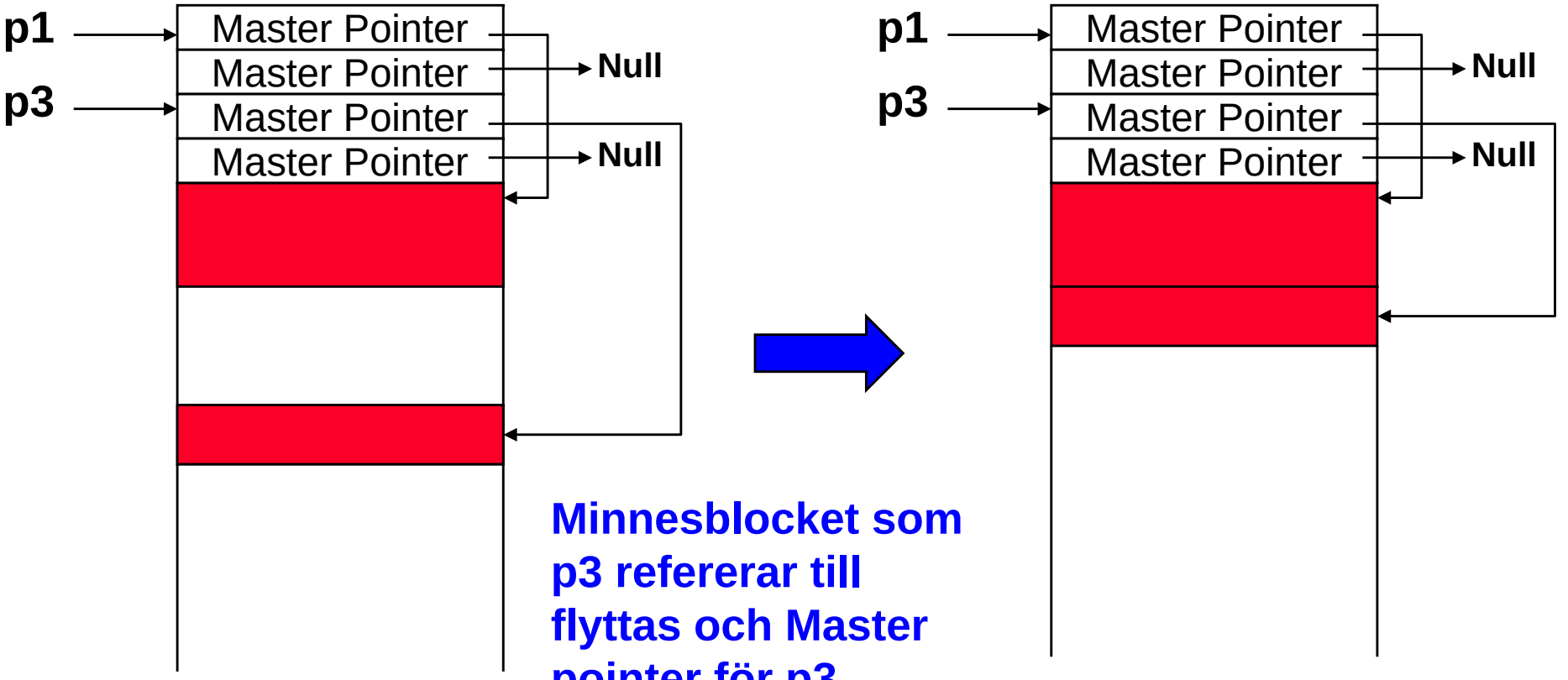


Ett exempel 1(3)



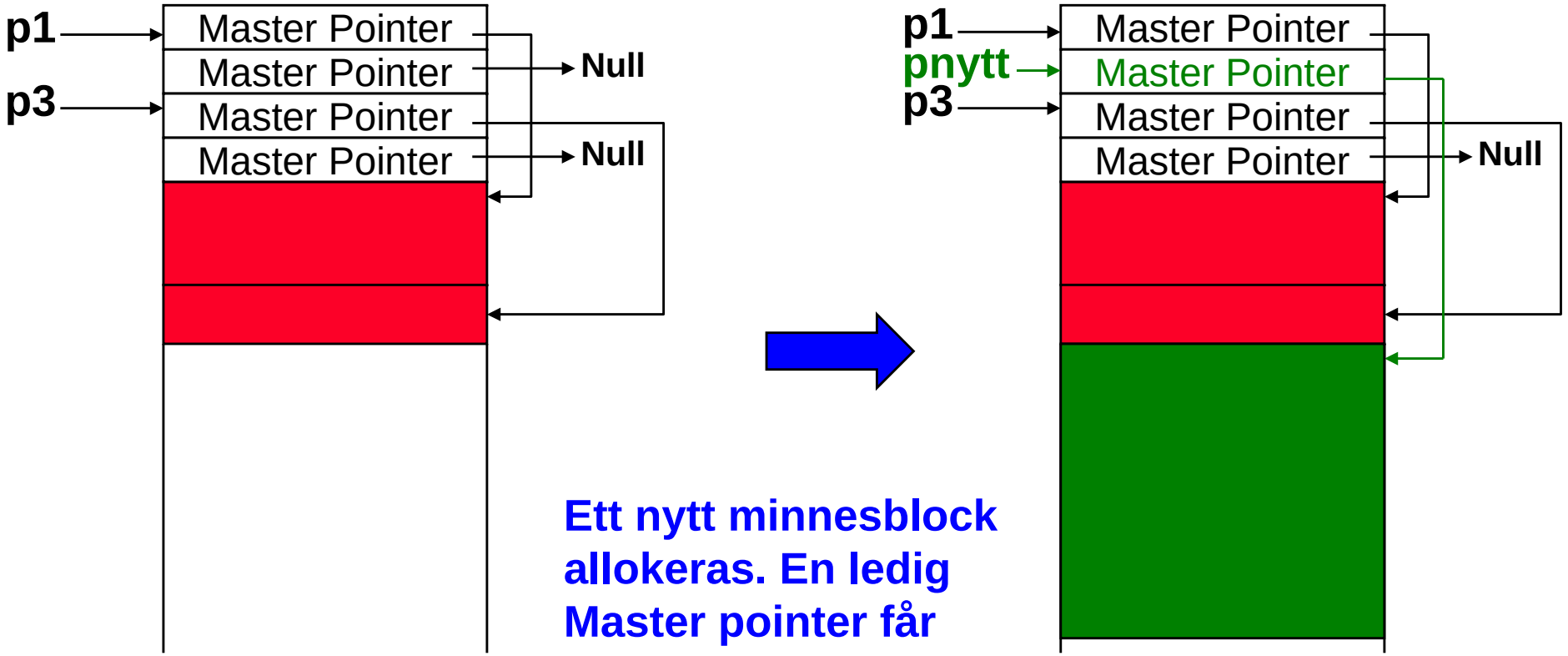
Det sker en fragmentering av heapen när minnesblocken som handtagen p2 och p4 refererar till tas bort.

Ett exempel 2(3)



Minnesblocket som p3 refererar till flyttas och Master pointer för p3 uppdateras.

Ett exempel 3(3)



Ett nytt minnesblock allokeras. En ledig Master pointer får utgöra ett nytt handtag pnytt.

Högnivå datatyper, C

- **Float (IEEE 754)**
  - **32 bits**
    - $-3.4 \times 10^{38} - 3.4 \times 10^{38}$
      - **s eeee eeee mmmmmmmmmmmmmmmmmmmmmmm**
      - **s = tecken (1)**
      - **e = exponent (2-komp) (8)**
      - **m = mantissa (23)**
- **double**
  - **64 bits**
    - $-1.8 \times 10^{309} - 1.8 \times 10^{309}$
      - **s eeeeeeeeee mmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmm**
      - **s = tecken (1)**
      - **e = exponent (2-komp) (11)**
      - **m = mantissa (52)**

Högnivå datatyper, C

- Char (8bit)
  - Signed (-128 till 127)
  - Unsigned (0 till 255)
- int (16 eller 32 bit)
  - short int (16 bit)
    - signed (-32768 till 32767)
    - unsigned (0 till 65535)
  - long int (32 bit)
    - signed (-2147483648 till 2147483647)