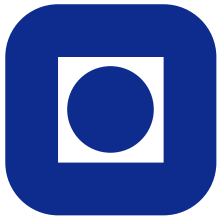


NTNU
Norwegian University of
Science and Technology

# Linux prosesser

Donn Morrison

Department of Computer Science
With slides contributed by Geir Maribu

# Outline

Review

Linux prosesser

- Prosess-tabellen
- Prosessens tilstander
- Nye prosesser
- Eksempel C kode

# Om Linux

- Først utviklet av hvem? Nå?
- Lisens GPL? Hva er det? Hvorfor er det viktig?
- Kjernen til Linux er av hvilken type?
- Hva gjøre `ls -l /etc/*`? Hva med `rm -rf /?tc`?

# Outline

Review

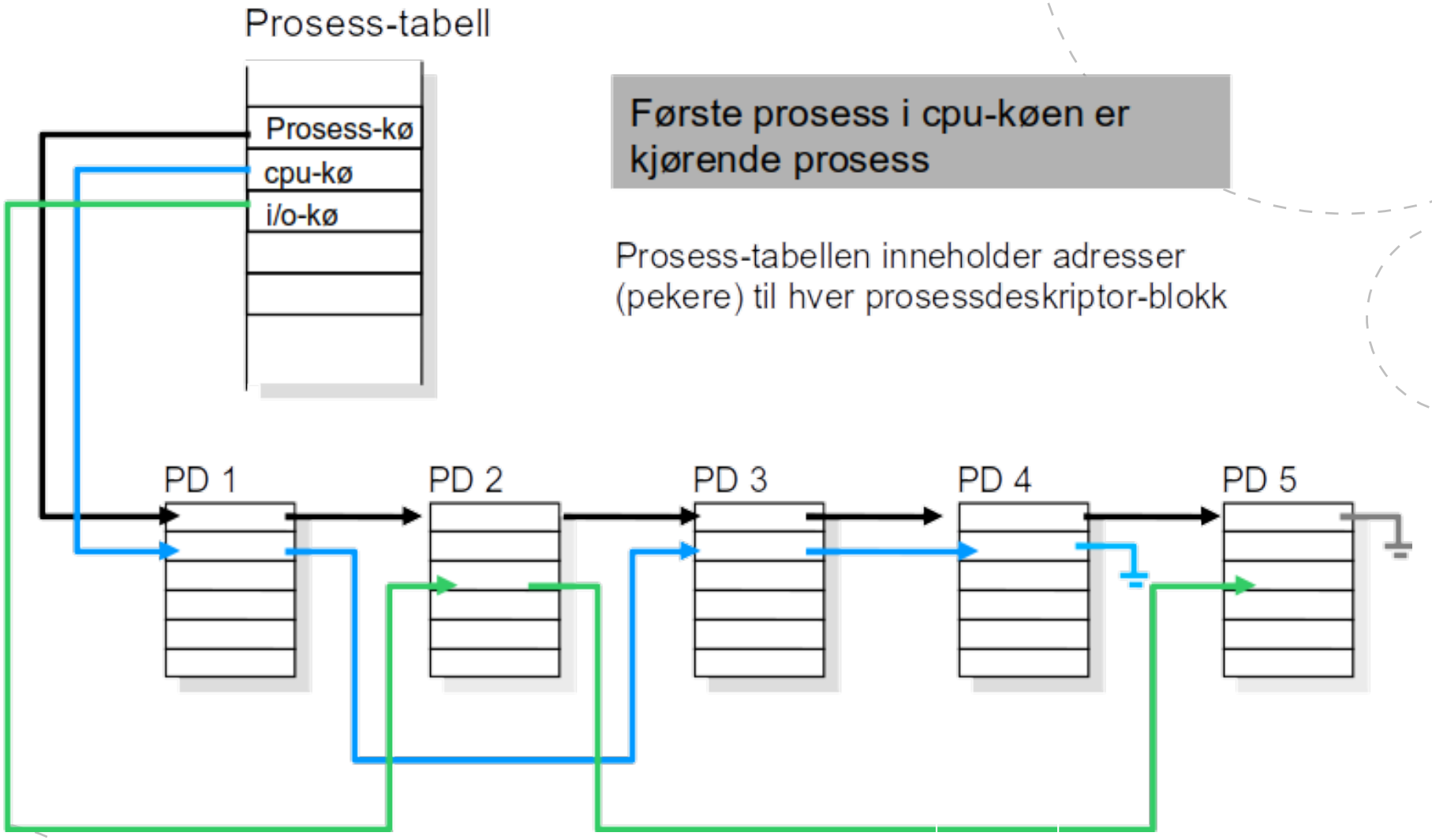
Linux prosesser

- Prosess-tabellen
- Prosessens tilstander
- Nye prosesser
- Eksempel C kode

# Prosess-tabellen

- Hver prosess har en PID
- På Linux er PIDer lagret i en prosess-tabell
- Prosess-tabell er felles for hele Linux-systemet
- Bestemt antall plasser
  - `cat /proc/sys/kernel/pid_max`
  - 32768 (32-bit)
  - 4194303 (64-bit)

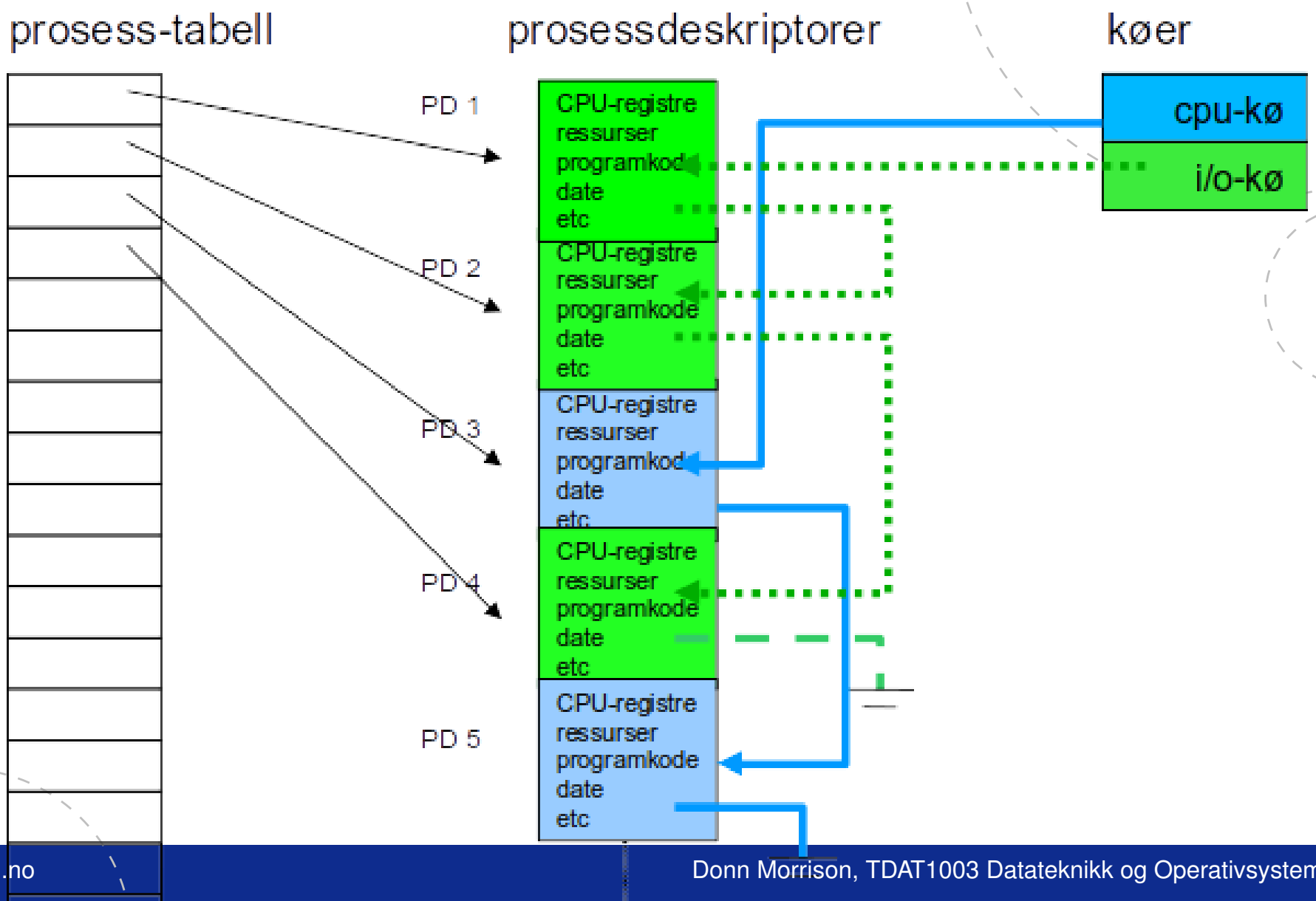
# Prosess-tabellen



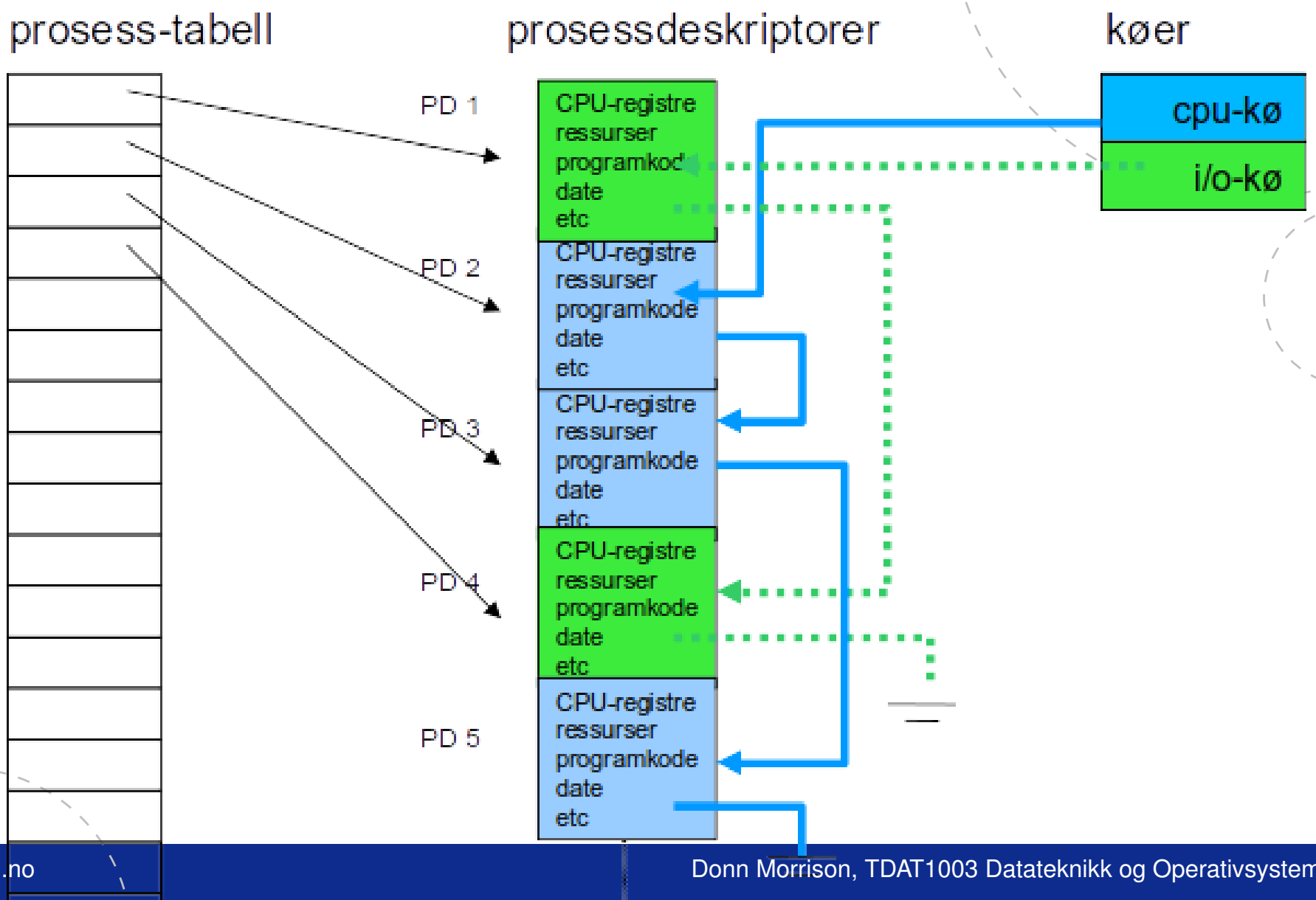

Første prosess i cpu-køen er kjørende prosess

Prosess-tabellen inneholder adresser (pekere) til hver prosessdeskriptor-blokk

# Prosesskøen



# Prosesskøen



# Prosesskøen

— Først i CPU-kø betyr?

# Prosesskøen

— Først i CPU-kø betyr?
  - Kjørende prosess

— Prosess i I/U kø betyr?

# Prosesskøen

- Først i CPU-kø betyr?
  - Kjørende prosess
- Prosess i I/U kø betyr?
  - Blokkert prosess, venter på I/U

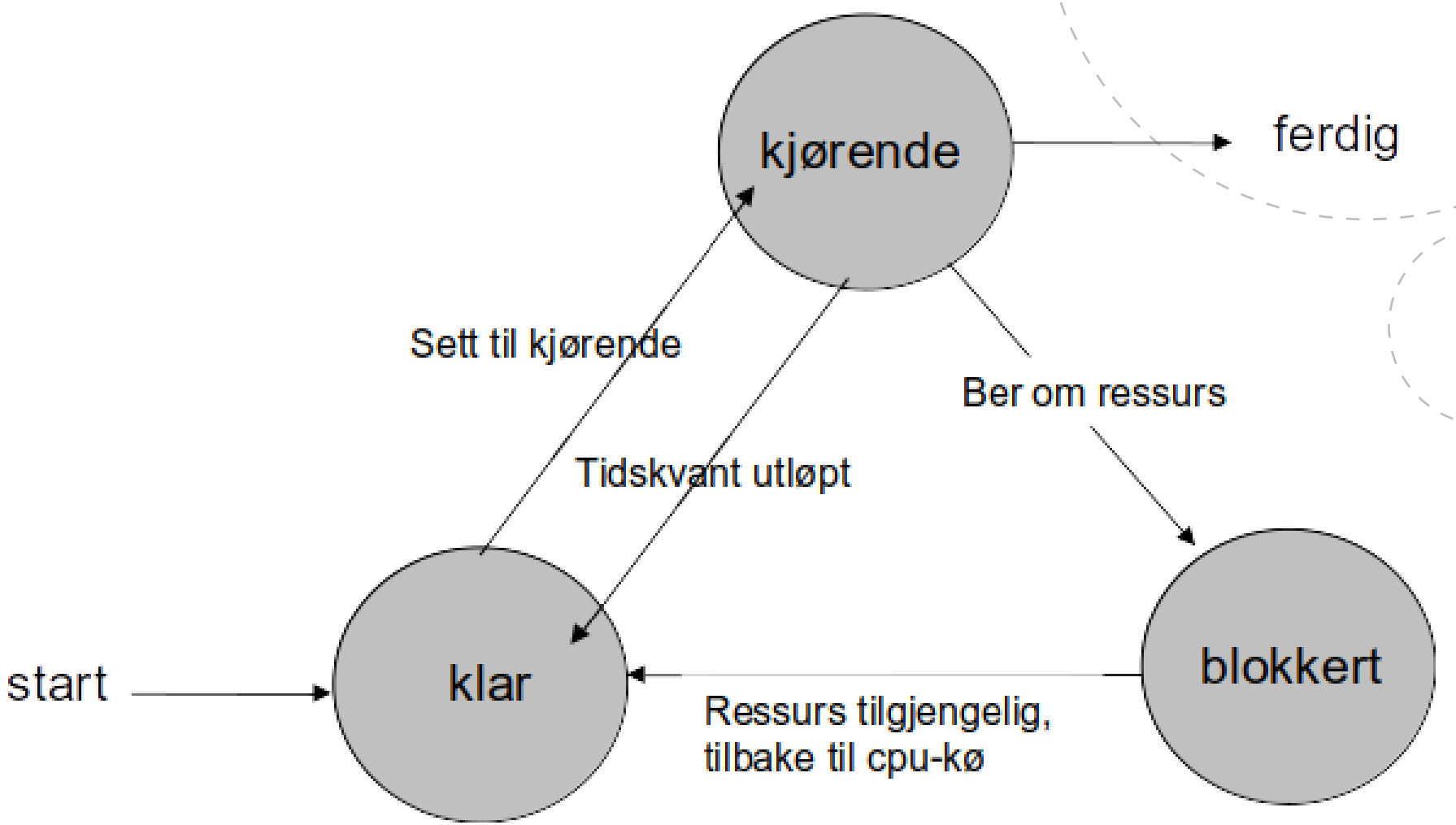
# Prosess-tabellen

— Når endrer vi prosess-tabellen?

# Prosess-tabellen

— Når endrer vi prosess-tabellen?
  - Når en prosess starter
  - Når en prosess slutter
  - Når tilstanden til en prosess endres

# Prosessens tilstander



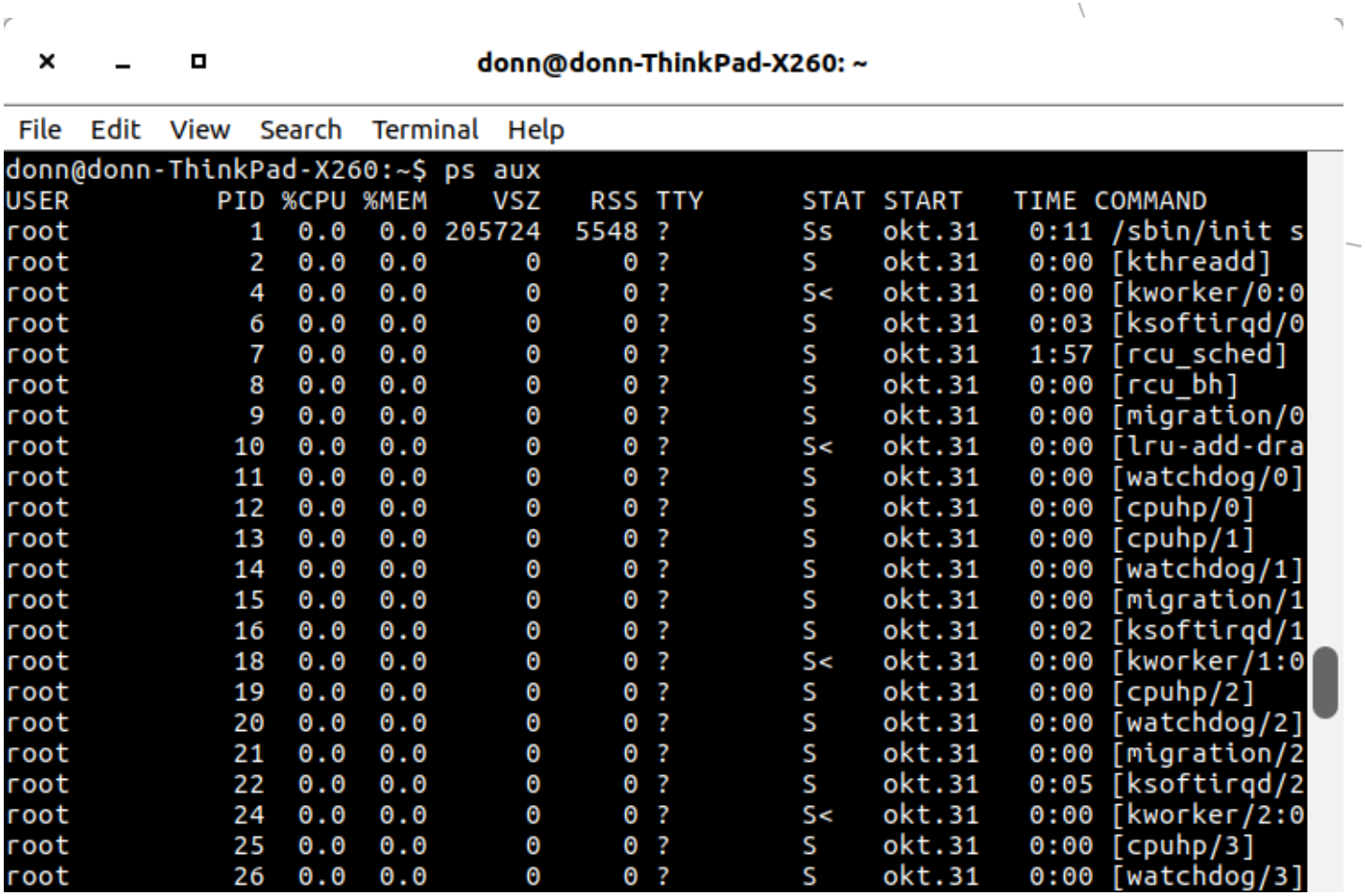
# Prosessens tilstander

— Hvordan ser vi tilstander til prosesser?

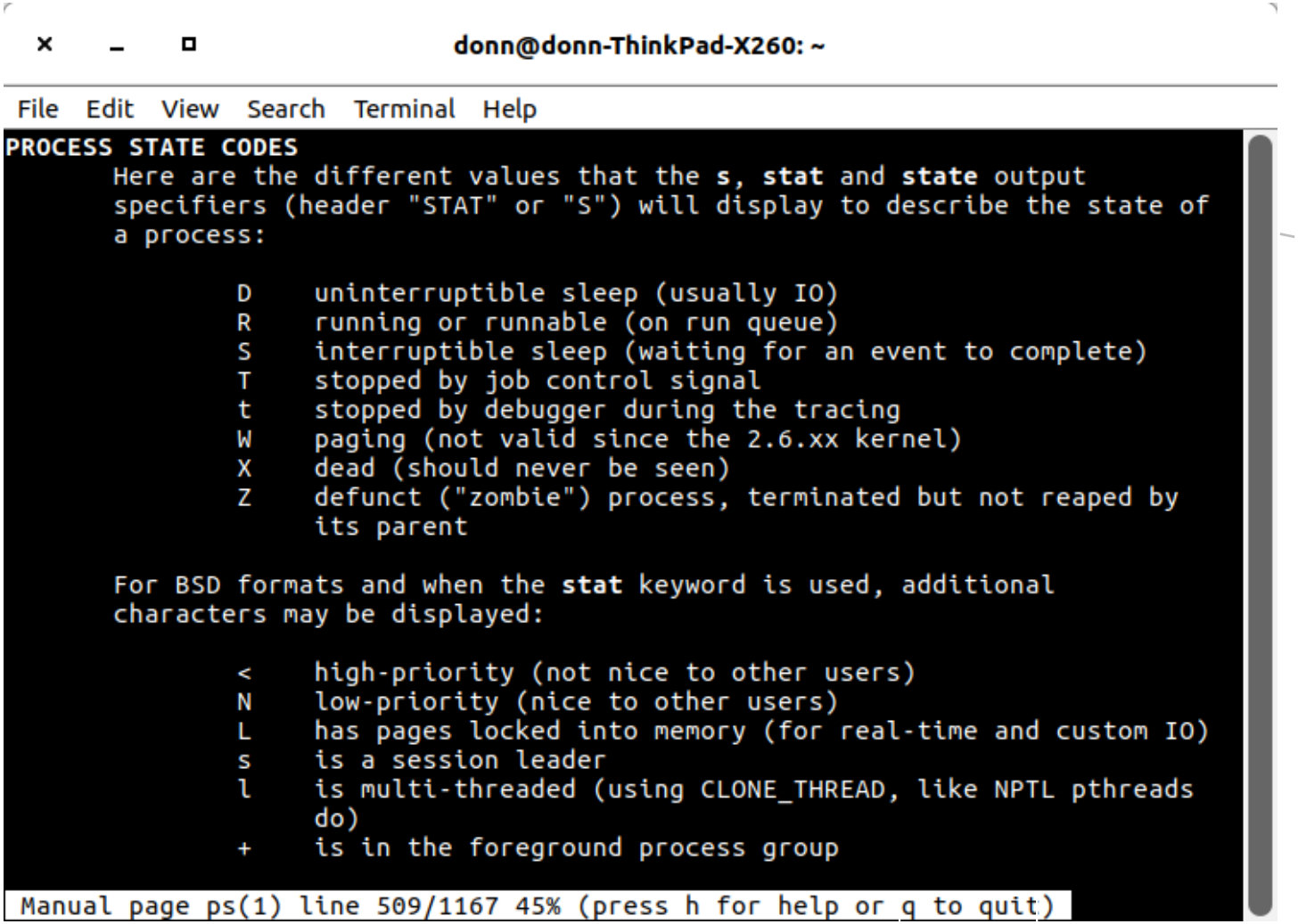
# Prosessens tilstander

— Hvordan ser vi tilstander til prosesser?
  - ps-kommandoen (ps = process status)
  - Viser informasjon fra prosess-tabellen og prosess deskriptorene

# ps-kommandoen



# ps-kommandoen



# Opprettelse av nye prosesser

- Når vi kjører ps-kommandoen, starter vi en ny prosess
- `/bin/ps`
- Hvordan?

# Opprettelse av nye prosesser

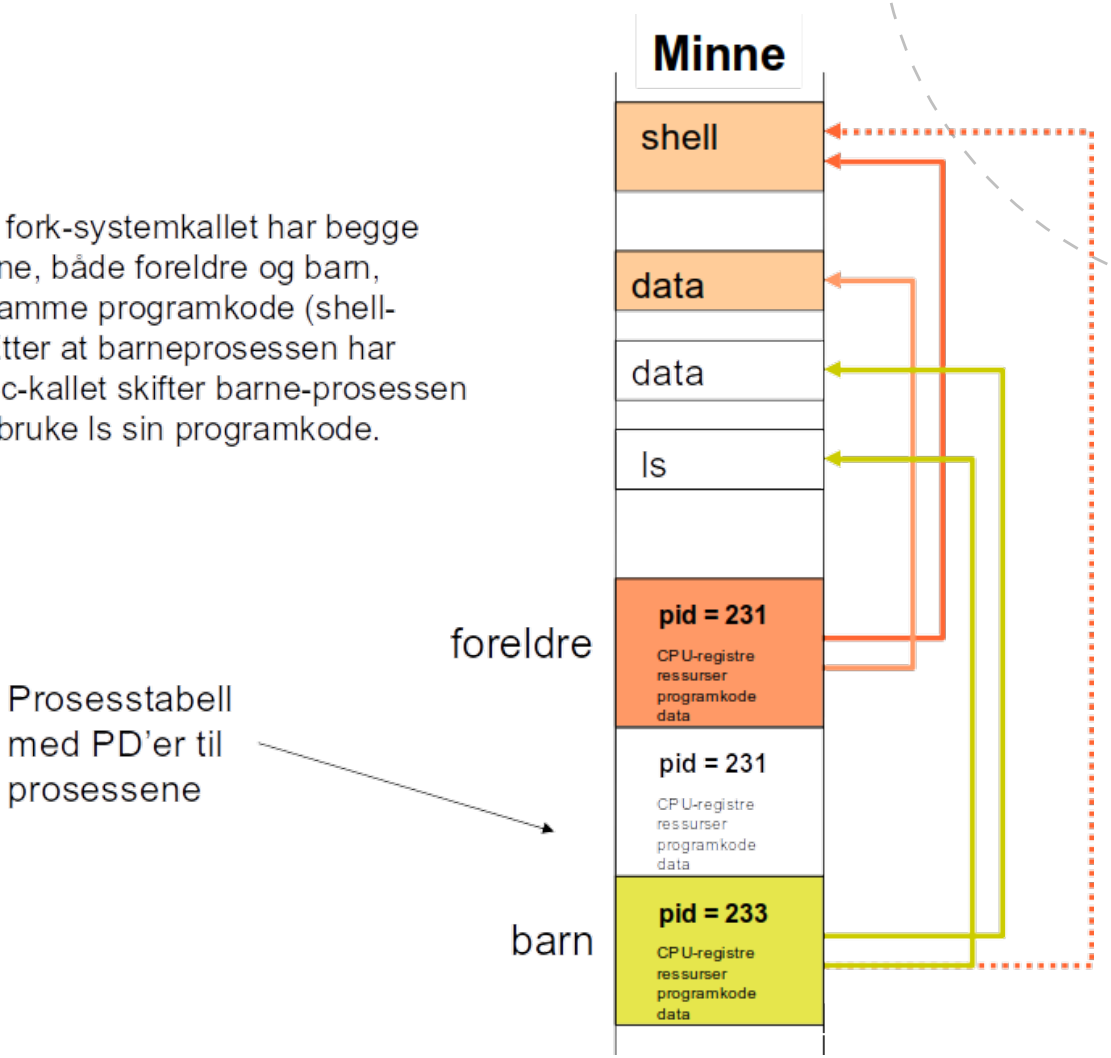
- Når vi kjører ps-kommandoen, starter vi en ny prosess
- `/bin/ps`
- Hvordan?
  - Skall-programmet bruker systemkaller
  - Disse er **fork** og **exec**
  - Skall-programmet blir foreldre-prosess, og ps-kommandoen blir barne-prosess

# Fork/exec

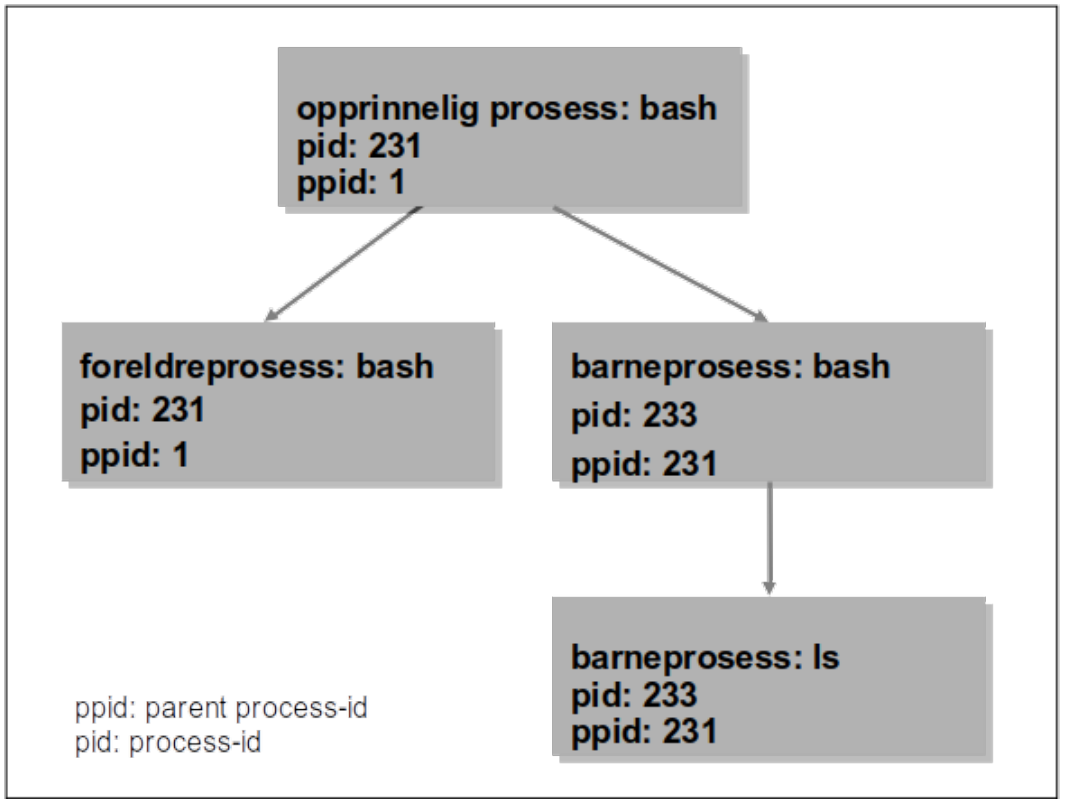
1. Lages det en ny prosessdeskriptor til den nye barneprosessen
2. Kopieres innholdet fra foreldreprosessen sin PD til barneprosessen sin PD
3. Barneprosessen får kopi av programkode, data og stakk til sine egne områder i minnet
    - Barneprosessen får adgang til alle filene som foreldre har adgang til

# Fork

Rett etter fork-systemkallet har begge prosessene, både foreldre og barn, akkurat samme programkode (shell-koden). Etter at barneprosessen har utført exec-kallet skifter barne-prosessen over til å bruke ls sin programkode.



# Fork/exec (ls)


opprinnelig prosess: bash
pid: 231
ppid: 1
foreldreprosess: bash
pid: 231
ppid: 1
barneprosess: bash
pid: 233
ppid: 231
barneprosess: ls
pid: 233
ppid: 231
ppid: parent process-id
pid: process-id
før fork og exec
fork
etter fork
exec
etter exec

# Fork/exec

- I Linux lages en prosess med systemkallet fork().
  - Et systemkall er et funksjonskall til en tjeneste i kjernen av operativsystemet
- Hver prosess har en foreldre
- Det er foreldre-prosessen som utfører fork-kallet
- Barne-prosessen arver alt fra foreldre-prosessen
- Barne-prosessen er helt identisk med foreldre-prosessen bortsett fra returverdien fra fork()
  - Foreldren får returverdi lik barnets PID
  - Barnet får 0 i returverdi
- Prosesser har altså foreldre og barn
  - Vi kan snakke om hele prosesstrær
  - ps axf -f
  - pstree

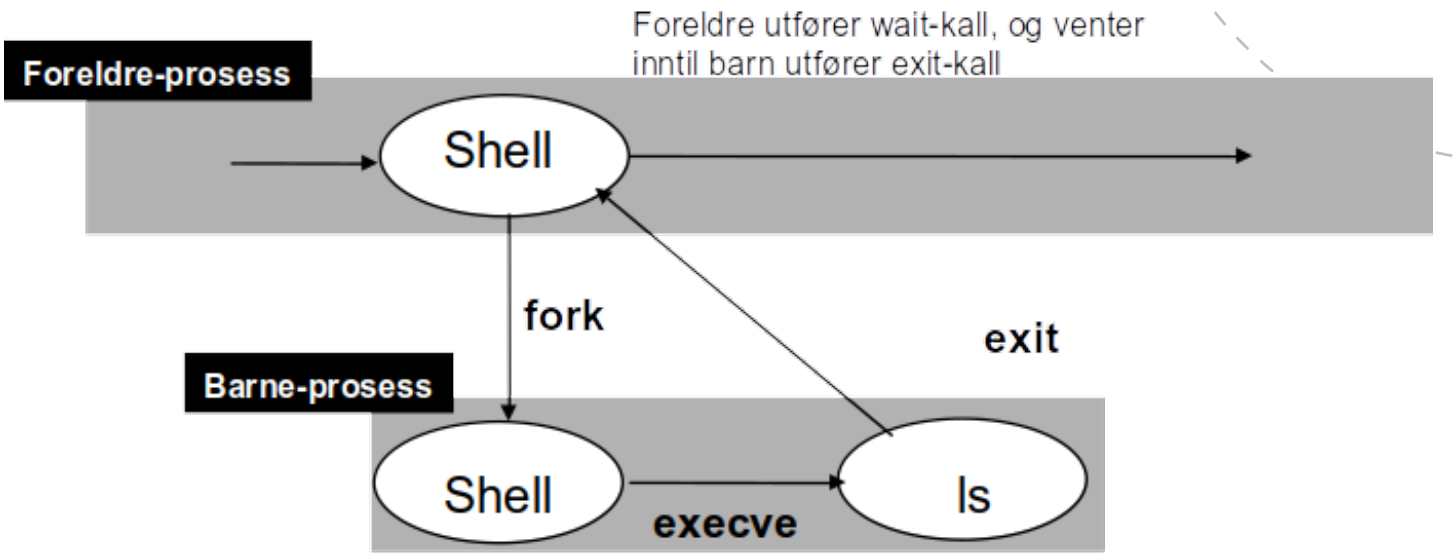
# Fork/exec (ls)

- fork() - oppretter en ny prosess
- execve() - Kallende prosess bytter ut sin egen programkode med en annen
- wait(), waitpid() - foreldreprosess får status om sitt eget barn etter at det har avsluttet, og rydder opp i prosesstabellen etter barnet (hindrer dermed at det dannes en zombie)
- exit() - prosess som utfører exit() avslutter

# Fork i C

```c
// forktest.c
// Compile: gcc forktest.c -o forktest
int main()
{
    int i=0,n;
    n=fork();
    if (n == 0)
        i=i+1;
    else
        i=i-1;
    printf("pid %d has %d\n",n,i);
}
```

# Fork/exec (ls)



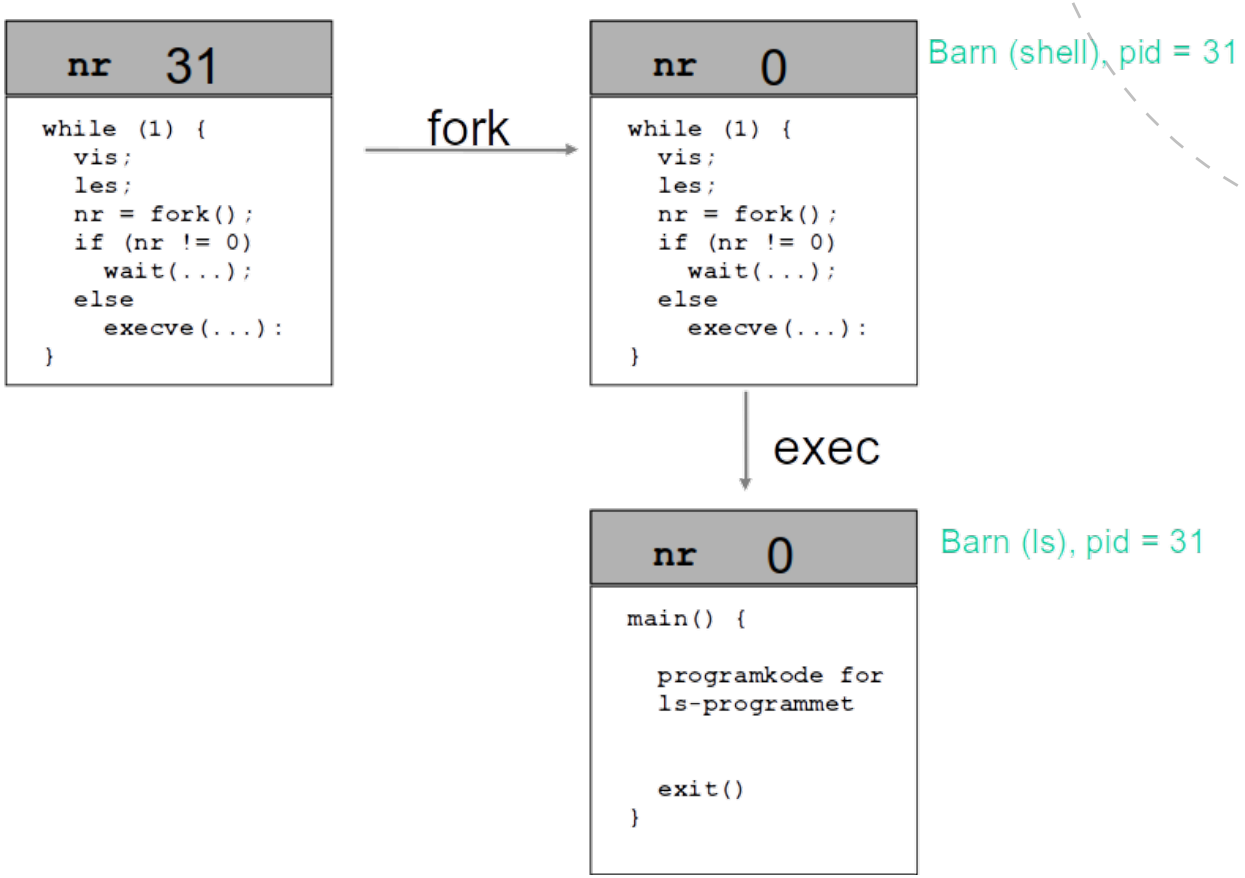

— Zombie prosesser?

`http://yarchive.net/comp/zombie_process.html`

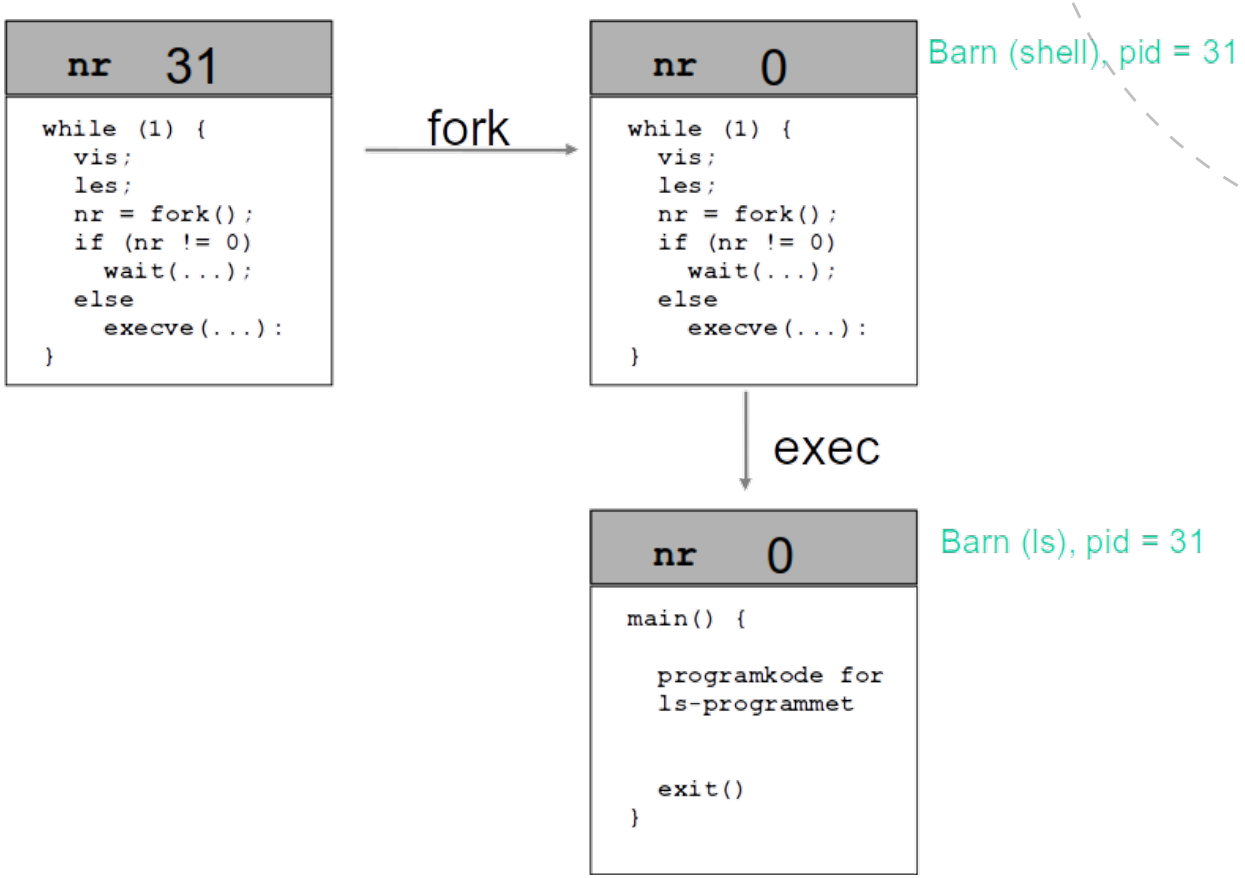
# Enkelt skall-programm

```
int main()
{
    int nr;
    while (TRUE)
    {
        vis_prompt();
        les_kommando(kmd, opsjoner);
        sjekk_kommando(kmd, opsjoner);
        if (kmd == OK)
        {
            nr = fork();
            if (nr != 0)
                wait(&status);              /* foreldre */
            else
                execve(kmd, opsjoner, 0);   /* barn */
        }
    }
}
```

# Fork/exec (ls)


nr 31
while (1) {
vis;
les;
nr = fork();
if (nr != 0)
wait(...);
else
execve(...);
}
fork
nr 0
while (1) {
vis;
les;
nr = fork();
if (nr != 0)
wait(...);
else
execve(...);
}
Barn (shell), pid = 31
exec
nr 0
main() {
programkode for
ls-programmet
exit()
}
Barn (ls), pid = 31

# Fork/exec (ls)


nr 31
while (1) {
vis;
les;
nr = fork();
if (nr != 0)
wait(...);
else
execve(...):
}
fork
nr 0
while (1) {
vis;
les;
nr = fork();
if (nr != 0)
wait(...);
else
execve(...):
}
Barn (shell), pid = 31
exec
nr 0
main() {
programkode for
ls-programmet
exit()
}
Barn (ls), pid = 31

# Ex 1

**Eksempel 1:**

```
printf("hei\n");
fork();
printf("paa deg\n");
```

**Utskrift fra programmet?**

a)
```
hei
```
b)
```
hei
paa deg
```
c)
```
hei
paa deg
paa deg
```
d)
```
hei
paa deg
hei
paa deg
```

# Ex 2

**Eksempel 2:**

```
printf("hei\n");
fork();
printf("paa deg\n");
fork();
printf("gamle venn\n");
```

**Hva blir utskrift fra programmet:**

A)
```
hei
```

B)
```
hei
på deg
gamle venn
```

C)
```
hei
på deg
gamle venn
gamle venn
```

D)
```
hei
på deg
på deg
gamle venn
gamle venn
gamle venn
gamle venn
```

# Ex 2

**Eksempel 2:**

```
printf("hei\n");
fork();
printf("paa deg\n");
fork();
printf("gamle venn\n");
```

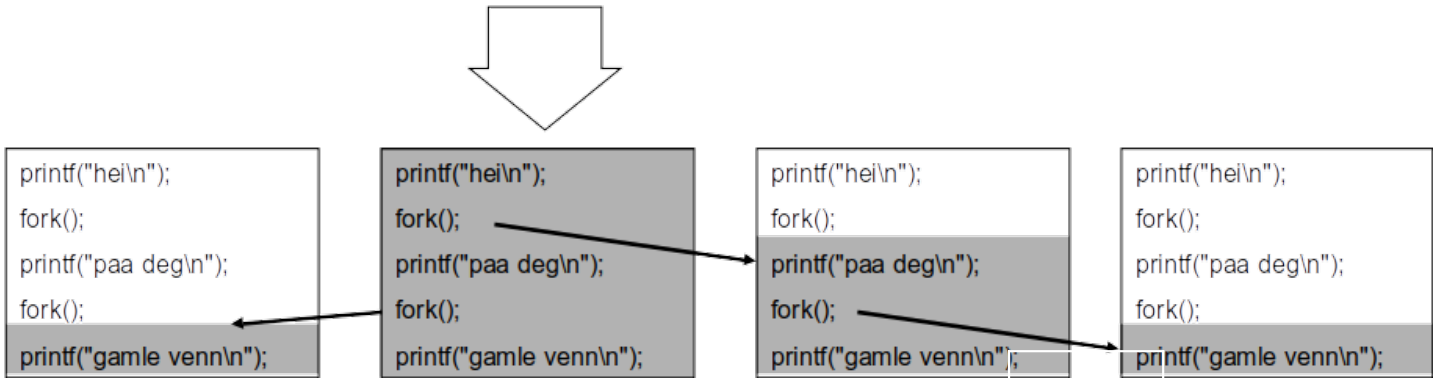
**Hva blir utskrift fra programmet:**

A)
```
hei
```

B)
```
hei
på deg
gamle venn
```

C)
```
hei
på deg
gamle venn
gamle venn
```

D)
```
hei
på deg
på deg
gamle venn
gamle venn
gamle venn
gamle venn
```



# Ex 3

**Eksempel 3:**

```
printf("hei\n");
execl("/bin/ls", "ls", "-al", (char *)0 );
printf("studenter\n");
```

**Hva blir utskrift fra programmet:**

A) hei

B) hei studenter

C) hei studenter + utskrift fra ls -al

D) hei + utskrift fra ls -al

# Ex 4

Gitt følgende program som er lagret på filen **a.out** i stående mappe:

```
#include "stdio.h"
#include "unistd.h"

main()
{
   printf("hei på deg ");
   sleep(2);
   execl("./a.out", "a.out", (char *) 0);
   printf("gamle ørn ");
   exit(0);
}
```

Hva skrives ut?

A) Ingenting
B) hei på deg
C) hei på deg hei på deg hei på deg ..... Etc
D) hei på deg gamle ørn