Pipes

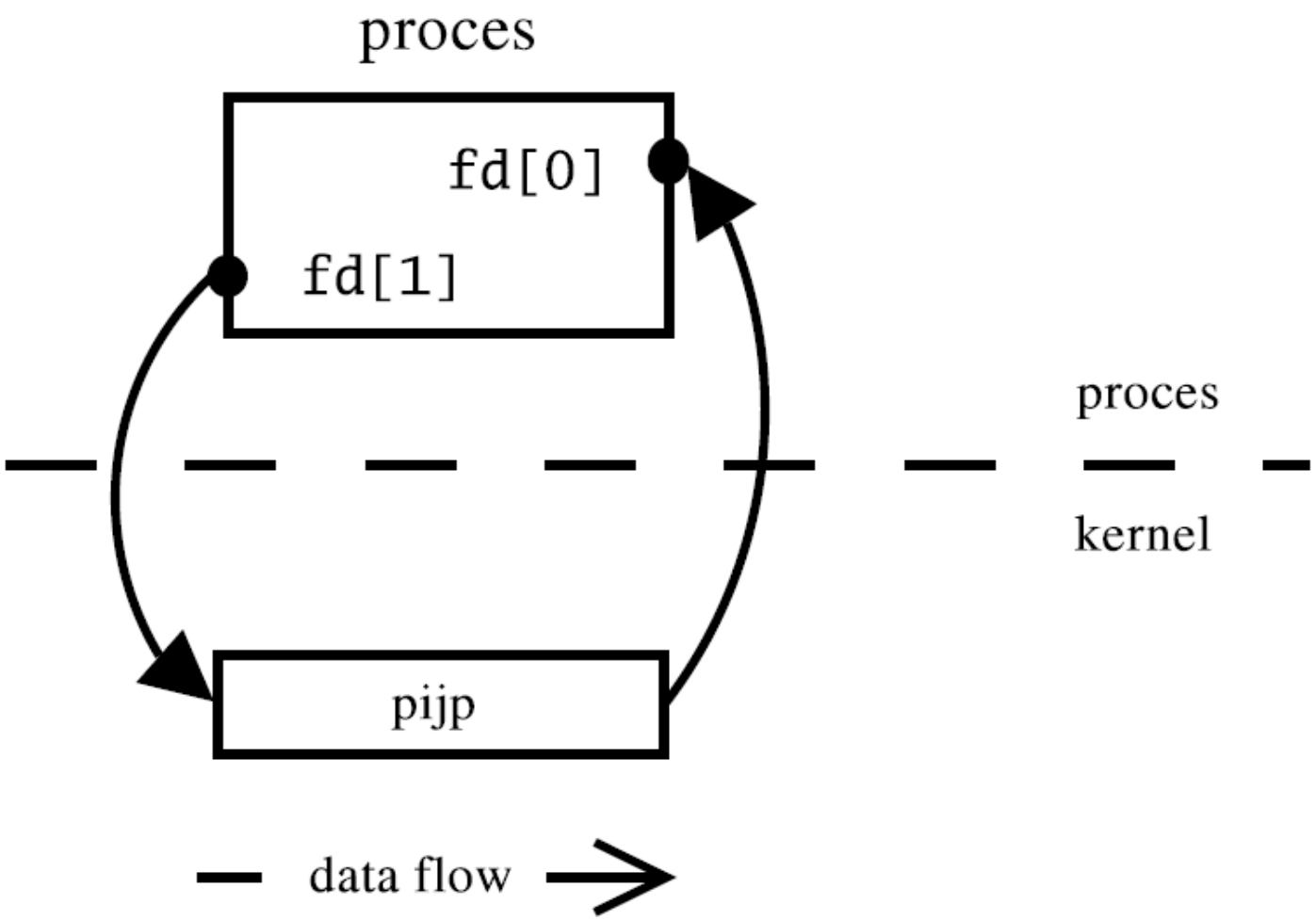
Veerle Ongenae

Pipes

- Basisconcepten
- Pipes in C
- Voorbeeldprogramma
- Pipes zijn half duplex

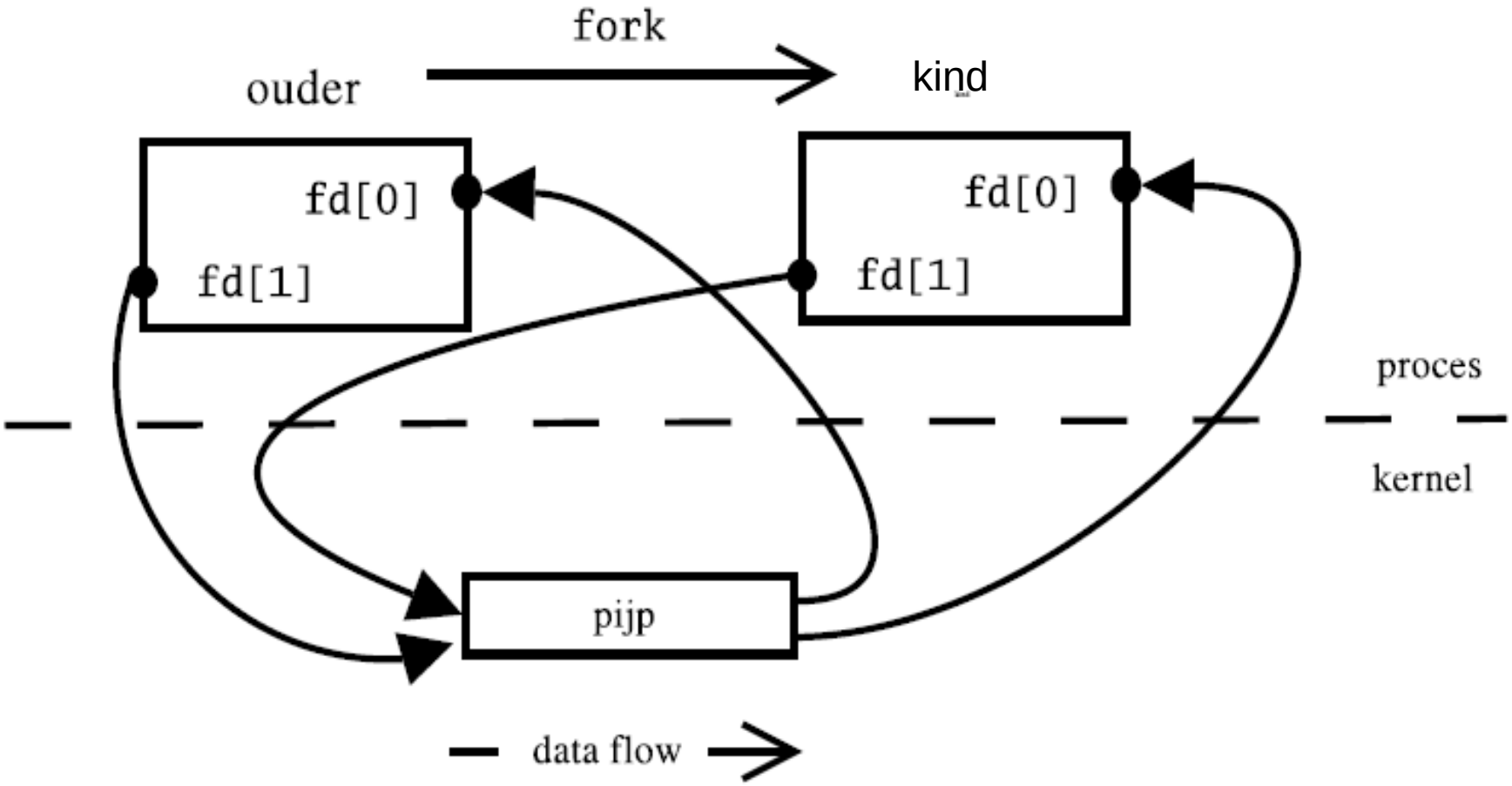
Basisconcepten

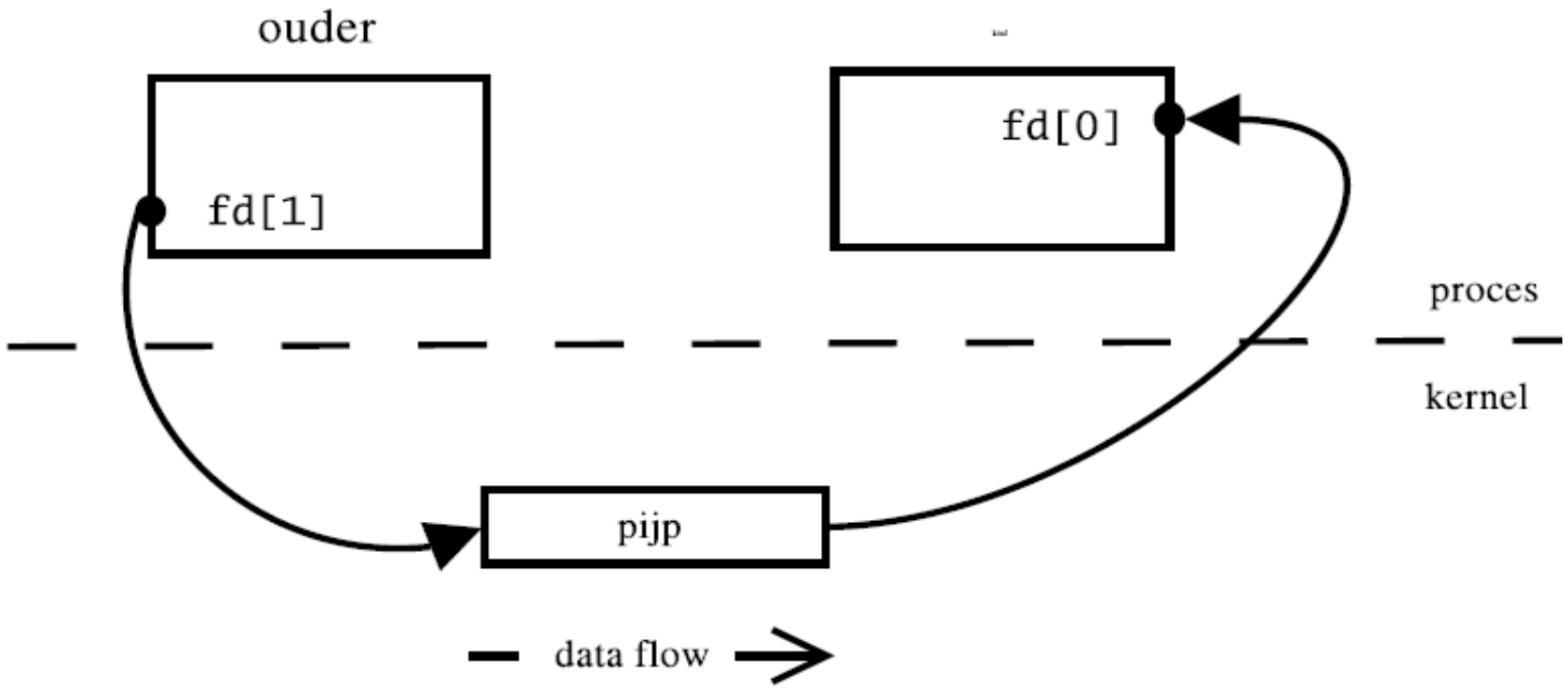
- standaard uitvoer van ene proces wordt standaard invoer van ander
- ls | sort | lp
- de pijp wordt gemaakt door de shell
- Twee descriptors
  - Ingang pijp (input, write), fd[1]
  - Uitgang pijp (output, read), fd[0]


proces
fd[0]
fd[1]
proces
kernel
pijp
data flow

Pipe met kindproces

- Proces met kindproces (via fork())
  - Beide descriptoren werden gekopieerd naar kindproces
  - Communicatie via pipe tussen ouder en kind
  - Richting bepalen
    - De juiste descriptoren afsluiten


fork
ouder
kind
fd[0]
fd[1]
fd[0]
fd[1]
proces
kernel
pijp
data flow


ouder
fd[1]
fd[0]
proces
kernel
pijp
data flow

Pipes

- Basisconcepten
- Pipes in C
- Voorbeeldprogramma
- Pipes zijn half duplex

Pipes in C

int **pipe**(int fd[2]);

return 0 bij succes, -1 bij error (zie errno)

- fd: descriptoren van de pipe
- fd[0]: lezen
- fd[1]: schrijven

Voorbeeld

- kind_schrijft_ouder.cpp
- Kindproces stuurt bericht naar ouderproces via pipe
- Opmerking
  - Read blokkeert proces totdat minstens één byte gelezen kan worden

Read en write

#include <unistd.h>

int **read** (int fd, char *buffer, int aant);
int **write**(int fd, char *buffer, int aant);

return: aantal bytes bij succes, -1 bij error (zie errno)

Read en write

- fd: descriptor
- buffer:
  - buffer om bytes in te lezen (read)
  - te schrijven bericht (write)
- aant
  - maximaal aantal te lezen bytes (read)
  - maximaal aantal te schrijven bytes (write)

Voorbeeld 2

- som_2dim_tabel.cpp
- Tweedimensionale tabel
  - Elk kindproces berekent een rij
- Kindproces schrijft resultaat naar pipe
- Ouderproces lees resultaten van pipe

Pipes zijn half duplex

- Half duplex
  - Éénrichtingscommunicatie
- full duplex pipes maak je door twee half duplex pipes te maken
- eerst pipe en dan pas fork
- werkt enkel bij processen die een gemeenschappelijke voorouder hebben!