

Processus et Redirections

Présentation: Stéphane Lavirotte

Auteurs: ... et al*

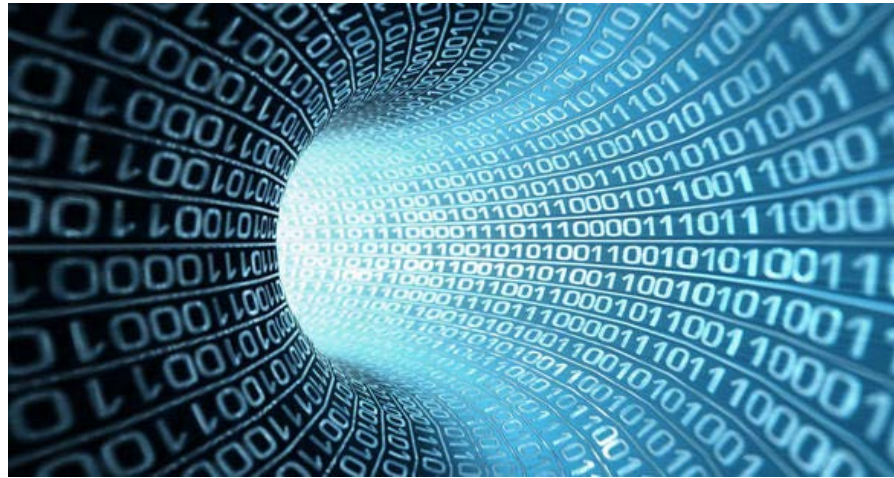
(*) Cours réalisé grâce aux documents de :

Stéphane Lavirotte, Jean-Paul Rigault

Mail: Stephane.Lavirotte@unice.fr

Web: <http://stephane.lavirotte.com/>

Université de Nice - Sophia Antipolis



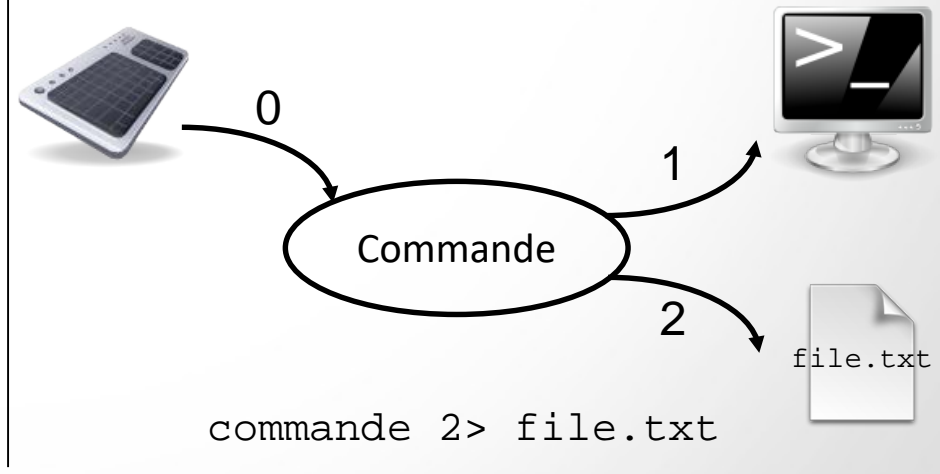
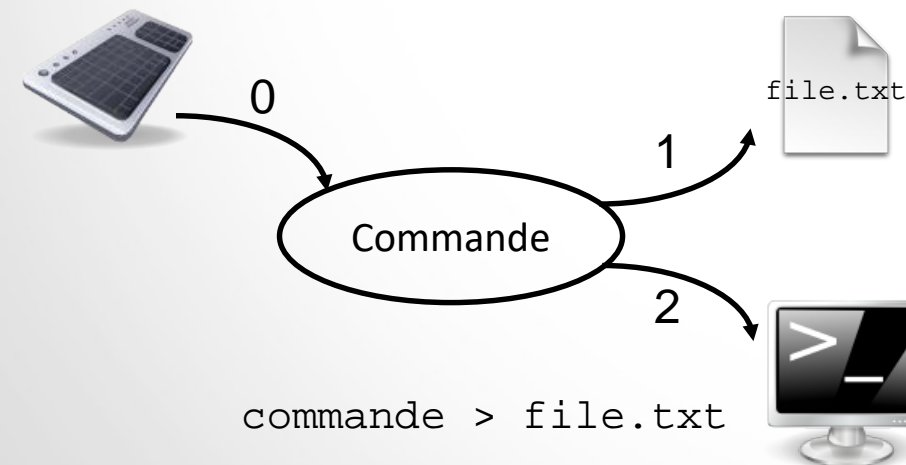
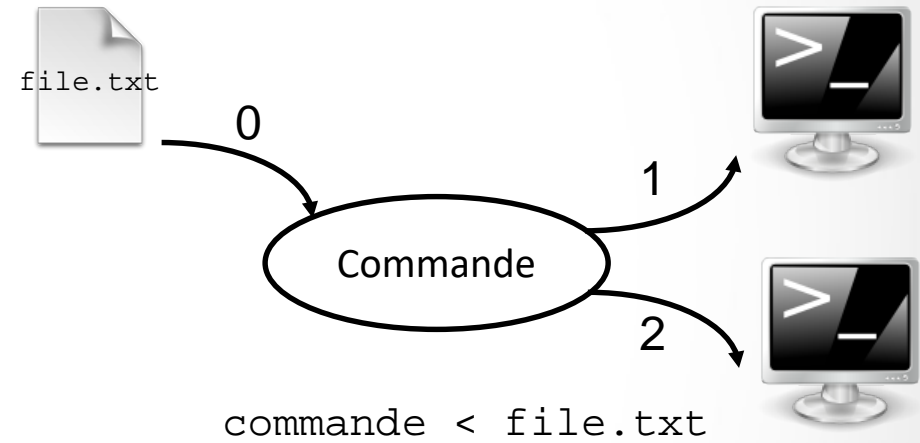
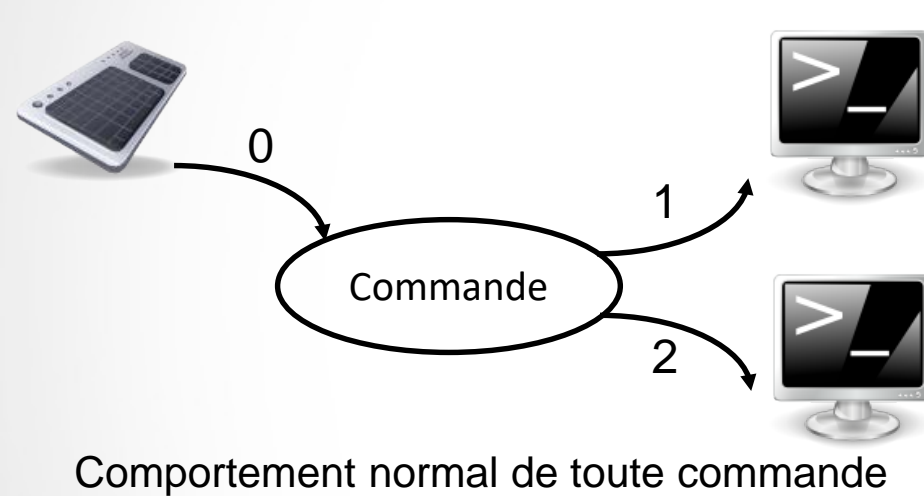
Redirection des Flux d'Entrée-Sorties

Quelques Rappels

Les Redirections en Shell

- ✓ Chaque programme a des canaux de communication par défaut:
 - 0: Entrée standard (par défaut clavier)
 - 1: Sortie standard (par défaut le terminal, donc écran)
 - 2: Sortie standard d'erreur (par défaut le terminal, donc écran)
- ✓ Il est possible de rediriger ces canaux de communication
 - Vers ou bien à partir d'un fichier (<, >, >>)
 - Vers un autre canal du même programme (>&)
 - Vers le canal d'un autre programme (|)
- ✓ Si on ne spécifie pas de numéro avec la redirection > ou >> c'est le canal 1 qui est celui par défaut

Les Redirections en Images Vers des Fichiers



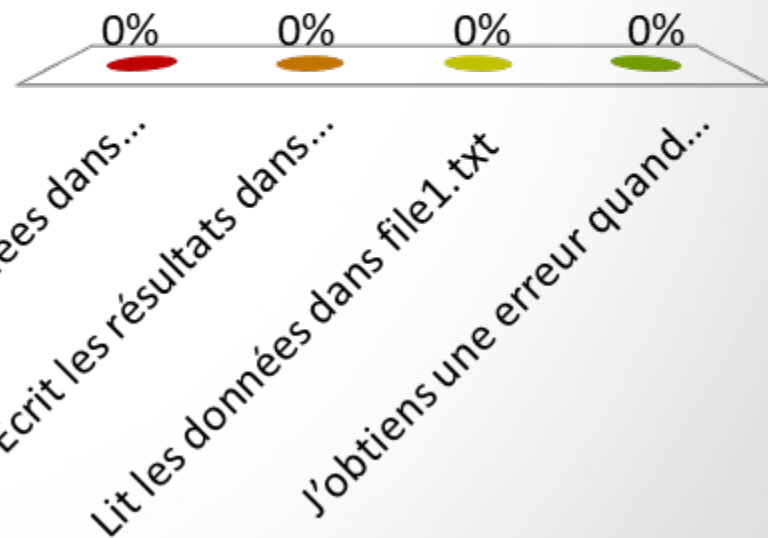
Quel est le résultat de la commande suivante ?

- Soit l'exécution suivante :

```
file1.txt < mon_script.sh > file2.txt
```

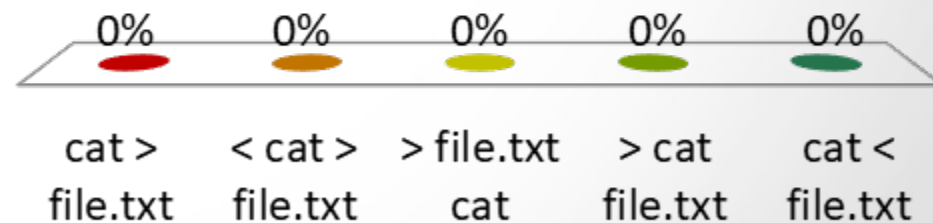
Que fait ce script avec ces deux redirections ?

1. **Lit les données dans file1.txt et écrit les résultats dans file2.txt**
2. **Ecrit les résultats dans file2.txt**
3. **Lit les données dans file1.txt**
4. **J'obtiens une erreur quand j'exécute cette commande**

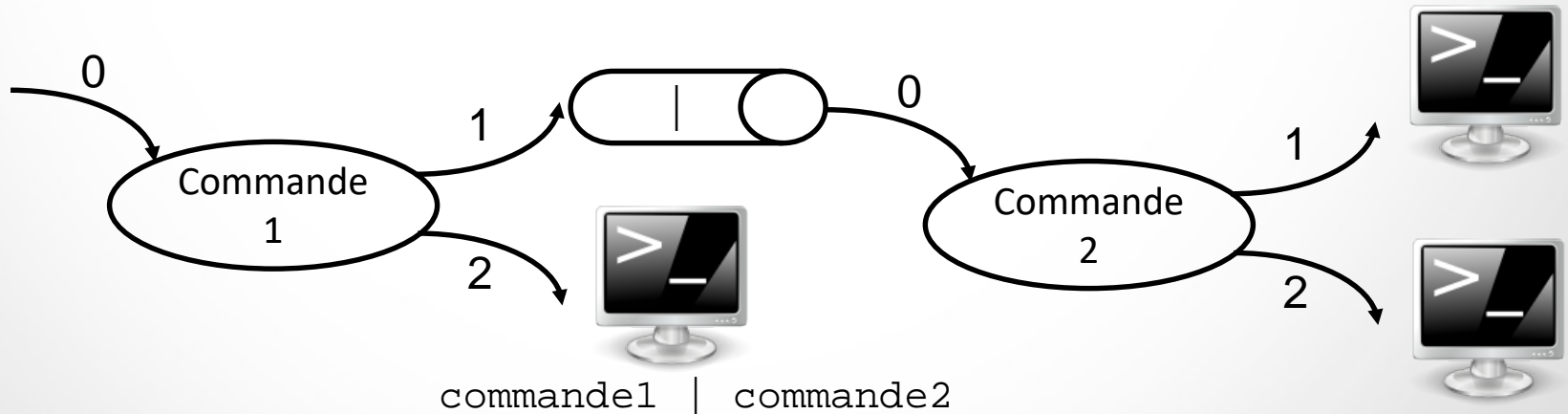
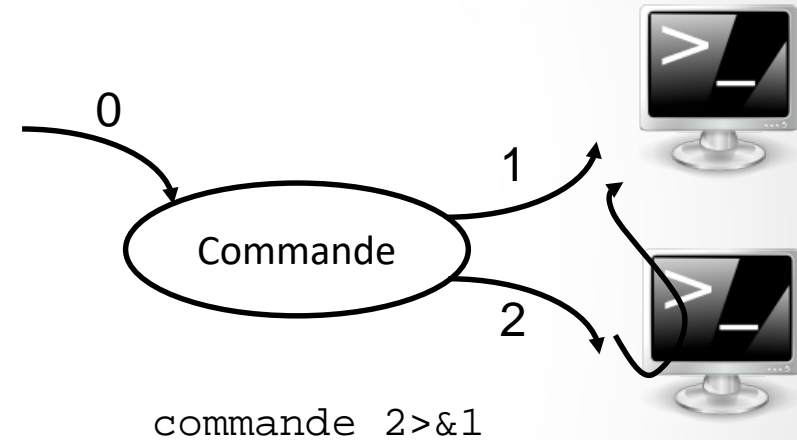
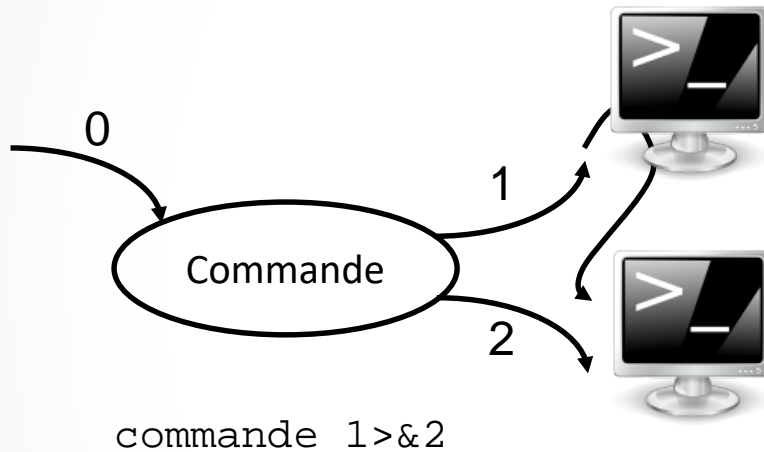


Quelle commande permet de remplir le contenu d'un fichier à partir du clavier ?

1. `cat > file.txt`
2. `< cat > file.txt`
3. `> file.txt cat`
4. `> cat file.txt`
5. `cat < file.txt`



Les Redirections en Images Vers des Canaux



Que fait la redirection suivante ?

- Soit le script `mon_script.sh`:

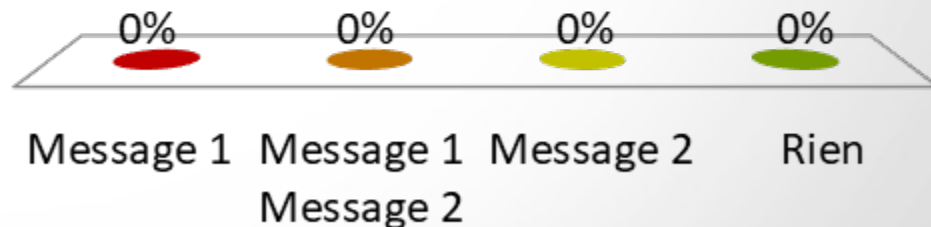
```
#!/bin/bash  
echo -n "Message 1 "  
echo "Message 2" >&2
```

et l'exécution suivante:

```
mon_script.sh > file.txt
```

Que contient le fichier `file.txt` ?

1. Message 1
2. Message 1 Message 2
3. Message 2
4. Rien



Que fait la redirection suivante ?

- Soit le script `mon_script.sh`:

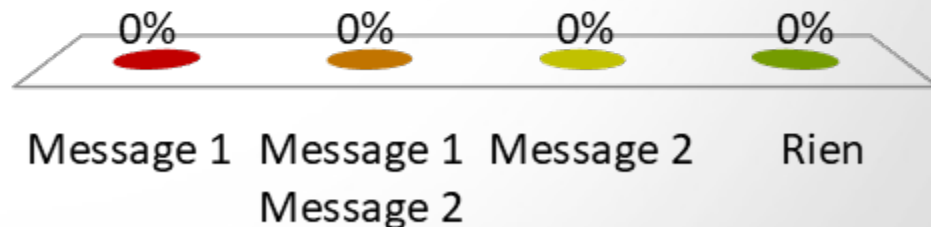
```
#!/bin/bash  
echo -n "Message 1 "  
echo "Message 2" >&2
```

et l'exécution suivante:

```
mon_script.sh 2>&1 > file.txt
```

Que contient le fichier `file.txt` ?

1. Message 1
2. Message 1 Message 2
3. Message 2
4. Rien



Que fait l'enchaînement de commandes suivantes ?

► Soit le fichier `/etc/passwd` suivant:

```
user2:x:11:21:::/home/user2:/bin/bash
```

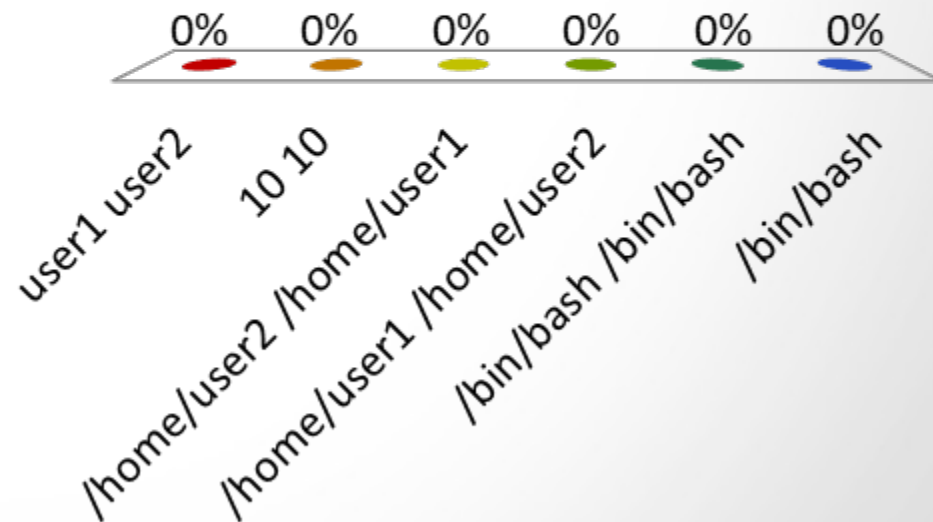
```
user1:x:10:10:::/home/user1:/bin/bash
```

et l'exécution de commandes suivantes:

```
cat /etc/passwd | cut -d ":" -f 6 | sort |
uniq
```

Qu'est ce qui sera affiché ?

1. **user1 user2**
2. **10 10**
3. **/home/user2 /home/user1**
4. **/home/user1 /home/user2**
5. **/bin/bash /bin/bash**
6. **/bin/bash**



Des suites de commandes pour « tout faire »!

- ✓ **Des traitements complexes peuvent être réalisés grâce à la redirections de flux de données**
 - **Compter le nombre de fichiers (ou dossiers) à partir d'un dossier**
 - `ls -alR /etc | grep '^-' | wc -l`
 - **Créer la liste des utilisateurs (ou des groupes) de votre machine**
 - `cat /etc/passwd | cut -d ":" -f 1 | sort`
 - **Connaître le nombre de comptes qui ont bien un mot de passe**
 - `sudo cat /etc/shadow | cut -d ":" -f 2 | grep -v
"^[^*!]+$" | wc -l`
 - **Compter le nombre de mots dans un document**
 - `cat untecte.txt | tr '[:punct:]' '\n' | tr -d '[\t]' |
grep -v "^$" | sort | uniq -c`

Résultats

- ✓ **Si vous avez**
 - 5 bonnes réponses: Vous savez tout sur les redirections. Bravo!
 - 4 bonnes réponses: Pas mal, vous avez des connaissances solides
 - ≤ 3 bonnes réponses: Il y a des choses à revoir

- ✓ **Pour améliorer vos connaissances et votre pratique sur les redirections:**
 - <http://stephane.lavirotte.com/teach/cours/envinfo1/08-09 Redirections.pdf>

- ✓ **Ce sont des connaissances qui doivent être maîtrisées avant d'étudier comment programmer les redirections**



Processus et Redirections

Mise en œuvre sous Posix C
(une descente sous la surface des choses)

Contrôle des Opérations sur Fichier

- ✓ **Permet de réaliser des opérations sur les descripteurs de fichiers**

```
#include <unistd.h>
#include <fcntl>
```

```
int fcntl(int fd, int cmd, ...);
```

- ✓ **Le nombre de paramètres dépend de `cmd`**
- ✓ **Commandes (`cmd`)**
 - `F_DUPFD` **duplique le descripteur (voir diapo suivante)**
 - `F_GETFL` **consulte indicateurs de `open()`**
 - `F_GETLK` **consulte état verrouillage**
 - `F_SETFL` **modifie indicateurs de `open()`**
 - `F_SETLK` **ou `F_SETLWK` modifie état verrouillage**
 - **etc.**

Duplication de descripteurs

1/2

✓ Duplication de descripteur

```
int newfd = fcntl(fd, F_DUPFD, minfd);
```

- **fd et newfd sont synonymes ; ils désignent le même fichier, avec le même pointeur d'E/S**

✓ Fonctions spéciales de duplication

```
int newfd = dup(fd);
```

- **équivalent à** `newfd = fcntl(fd, F_DUPFD, 0);`

```
int newfd = dup2(fd, desiredfd);
```

- **équivalent à:**

```
close(desiredfd);
```

```
newfd = fcntl(fd, F_DUPFD, desiredfd);
```

Duplication de descripteurs

2/2

- ✓ **Exemple de duplication de descripteur :**
 - **Redirections du Shell**

```
$ ls > foo  
int fd = open("foo", O_WRONLY | O_CREAT | O_TRUNC,  
              S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH) ;  
dup2(fd, 1) ;  
close(fd) ;
```


Communication de Données entre Processus: Tube – pipe() 1/3

```
#include <unistd.h>
```

```
int pipe(int fds[2]);
```



- ✓ **Flot non structuré de caractères**
- ✓ **Gestion FIFO (First In, First Out)**
- ✓ **Synchronisation producteur/consommateur**
 - `write()` peut être bloquant (`read()` aussi, bien sûr !)

Communication de Données entre Processus: Tube – pipe() 2/3

- ✓ Plusieurs processus peuvent se partager les deux extrémités du pipe
 - Atomicité des lectures et écritures de moins de PIPE_BUF caractères
 - Aucune structure des E/S n'est conservée dans le pipe

- ✓ Conditions aux limites
 - Si aucun processus n'a le pipe ouvert en écriture, une tentative de lecture reçoit une fin de fichier (EOF)
 - Si aucun processus n'a le pipe ouvert en lecture, un processus tentant d'écrire recevra le signal SIGPIPE (et, par défaut, se terminera)

Communication de Données entre Processus: Tube – pipe() 3/3

```

#define MAXL 100
char Msg[MAXL];

...
int fds[2];
pipe(fds);
if (fork()) {
    close(fds[0]);
    write(fds[1], "Salut !", sizeof("Salut !"));
    ...
} else {
    close(fds[1]);
    read(fds[0], Msg, MAXL);
    printf("%s\n", Msg);
    ...
}
  
```

Communication de données entre processus

Pipe nommé (fichier FIFO)

✓ Inconvénient des pipes

- Le pipe doit être créé par un ancêtre commun aux processus communicants
 - Les descripteurs correspondants sont hérités lors des `fork()`

✓ Pipe nommé (fichier FIFO)

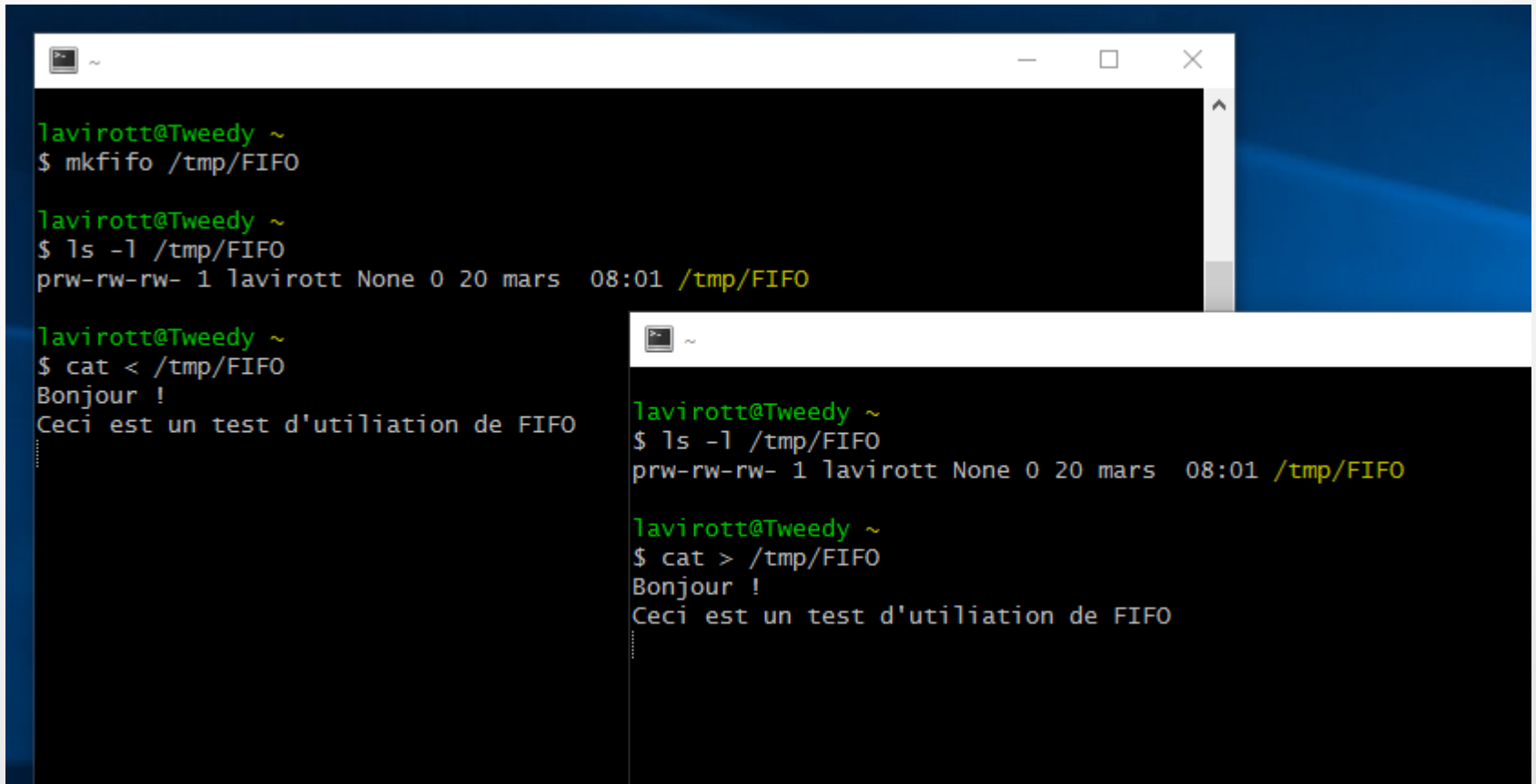
- Structure et synchronisation semblable au pipe
- Désigné par un nom de fichier

```
#include <sys/types.h>
```

```
#include <sys/stat.h>
```

```
int mkfifo(const char *path, mode_t mode);
```

Exemple de Communication de données entre Processus – Pipe Nommé –fichier FIFO)



```

lavirott@Tweedy ~
$ mkfifo /tmp/FIFO

lavirott@Tweedy ~
$ ls -l /tmp/FIFO
prw-rw-rw- 1 lavirott None 0 20 mars 08:01 /tmp/FIFO

lavirott@Tweedy ~
$ cat < /tmp/FIFO
Bonjour !
Ceci est un test d'utilisation de FIFO
.....

lavirott@Tweedy ~
$ ls -l /tmp/FIFO
prw-rw-rw- 1 lavirott None 0 20 mars 08:01 /tmp/FIFO

lavirott@Tweedy ~
$ cat > /tmp/FIFO
Bonjour !
Ceci est un test d'utilisation de FIFO
.....

```

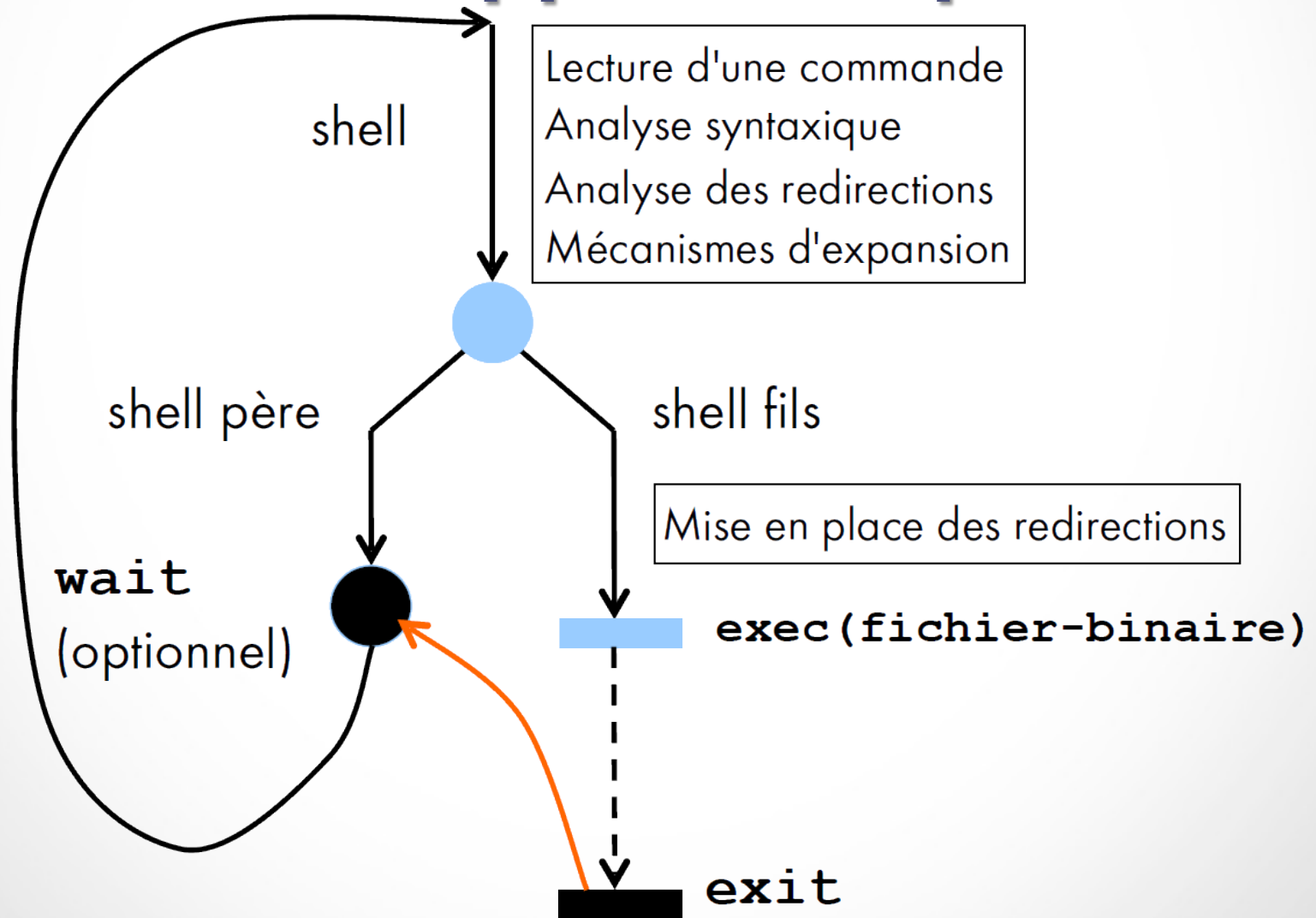


Illustrations de différentes commandes

Avec fork, exec et redirections

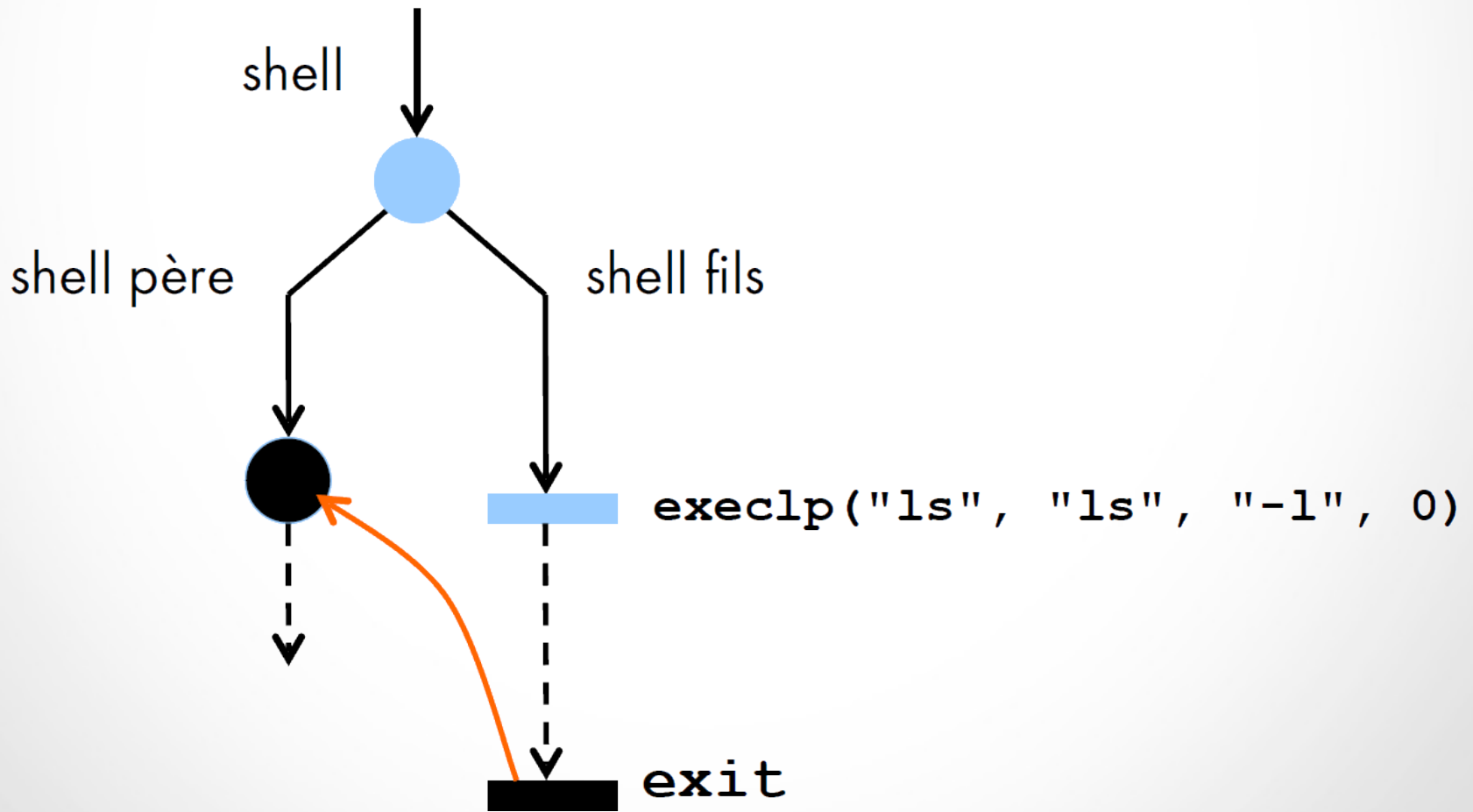
Exécution du Shell

Rappel Principe Général



Exécution du Shell Commande Simple

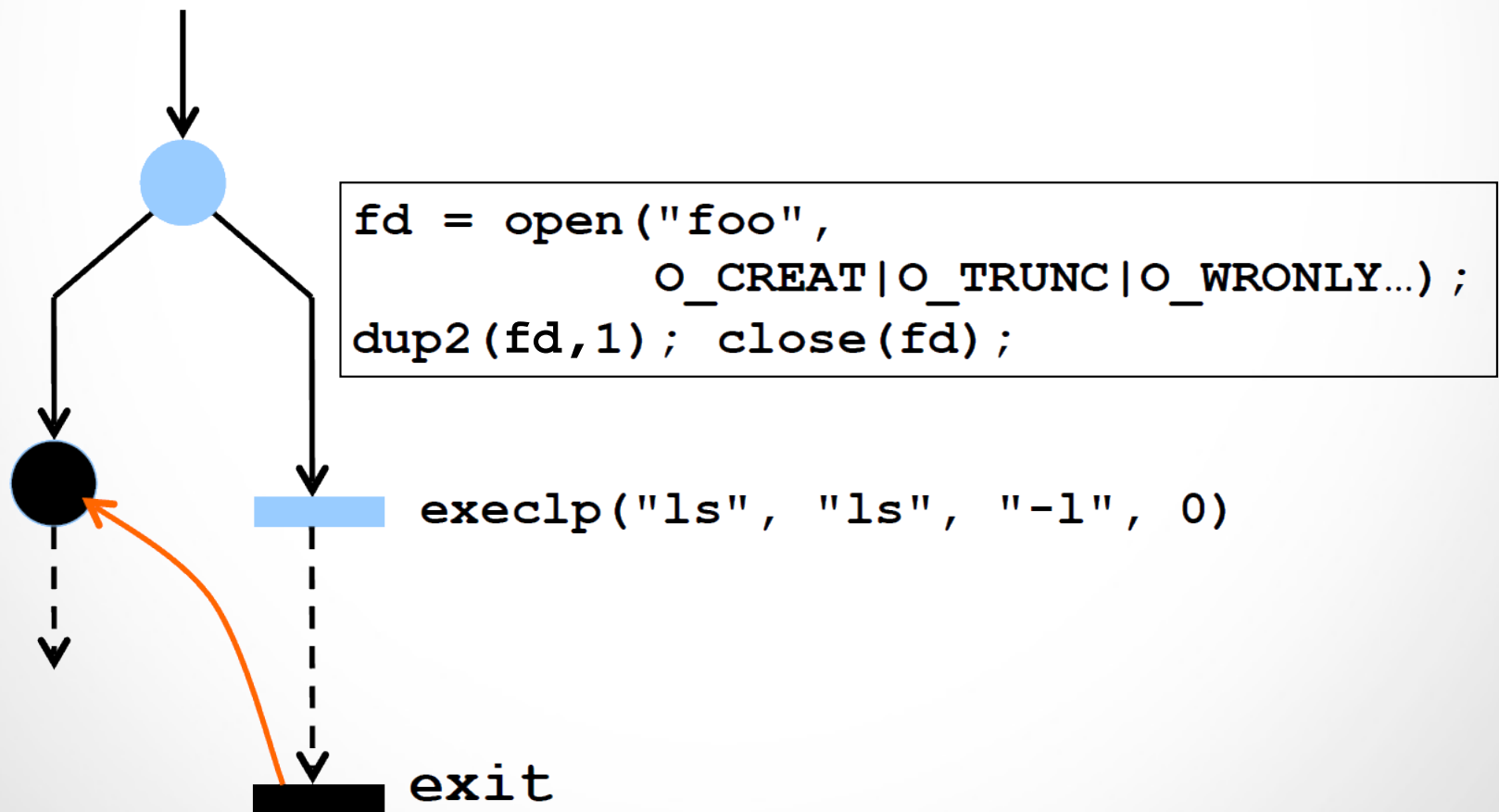
% *ls -l*



Exécution du Shell

Commande avec Redirection

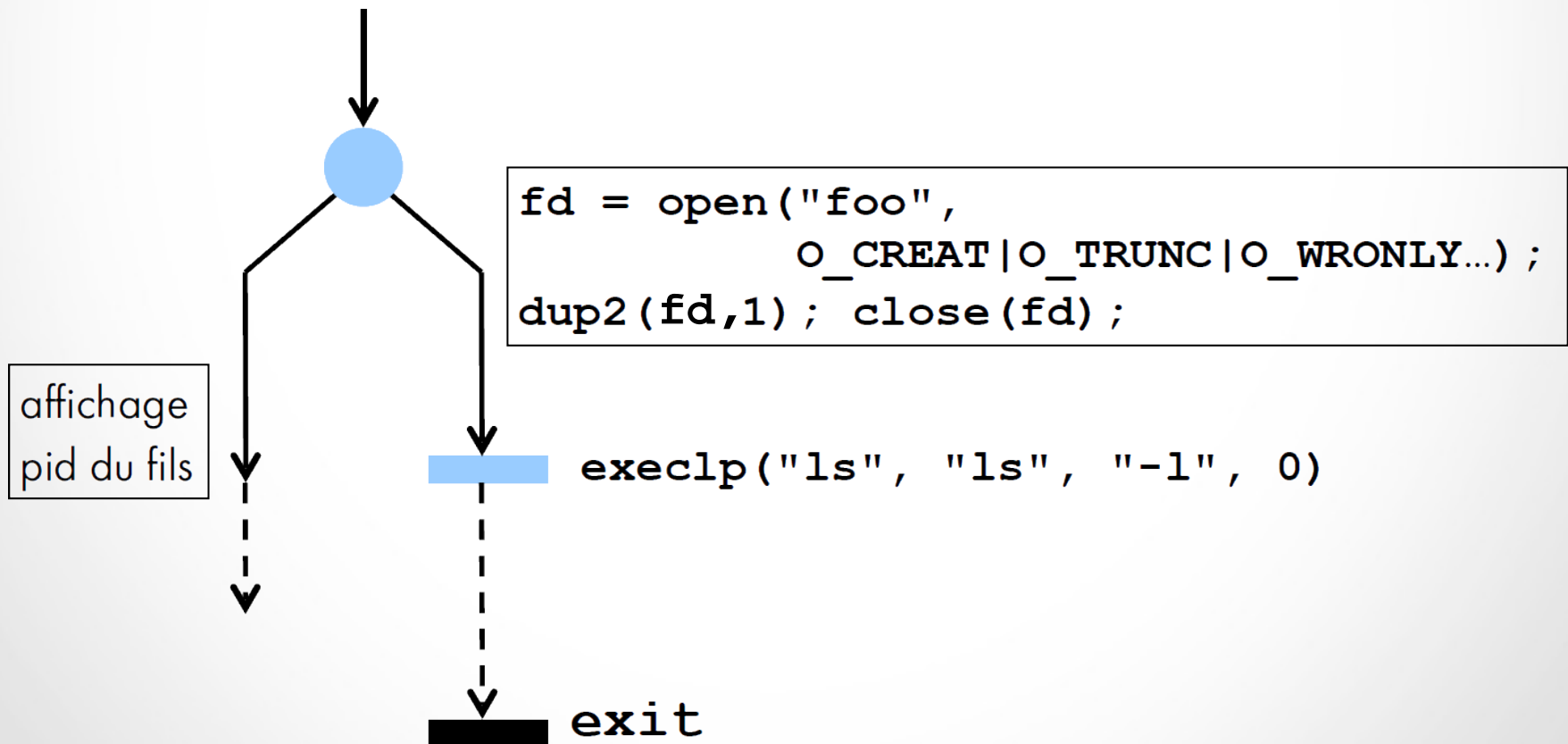
% *ls -l > foo*



Exécution du Shell

Commande en Arrière plan

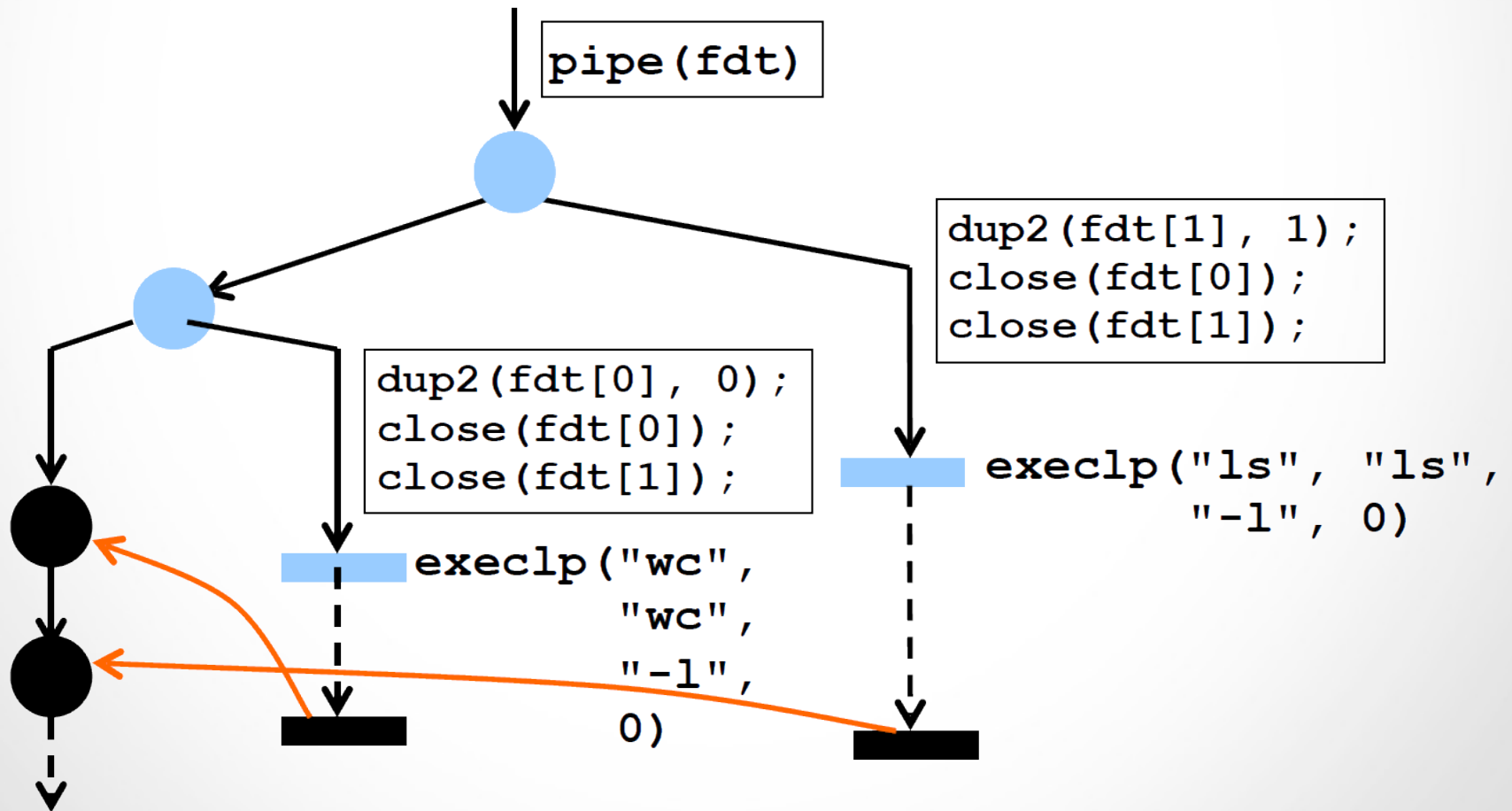
% *ls -l > foo &*



Exécution du Shell

Commande avec Pipe

% *ls -l | wc -l &*

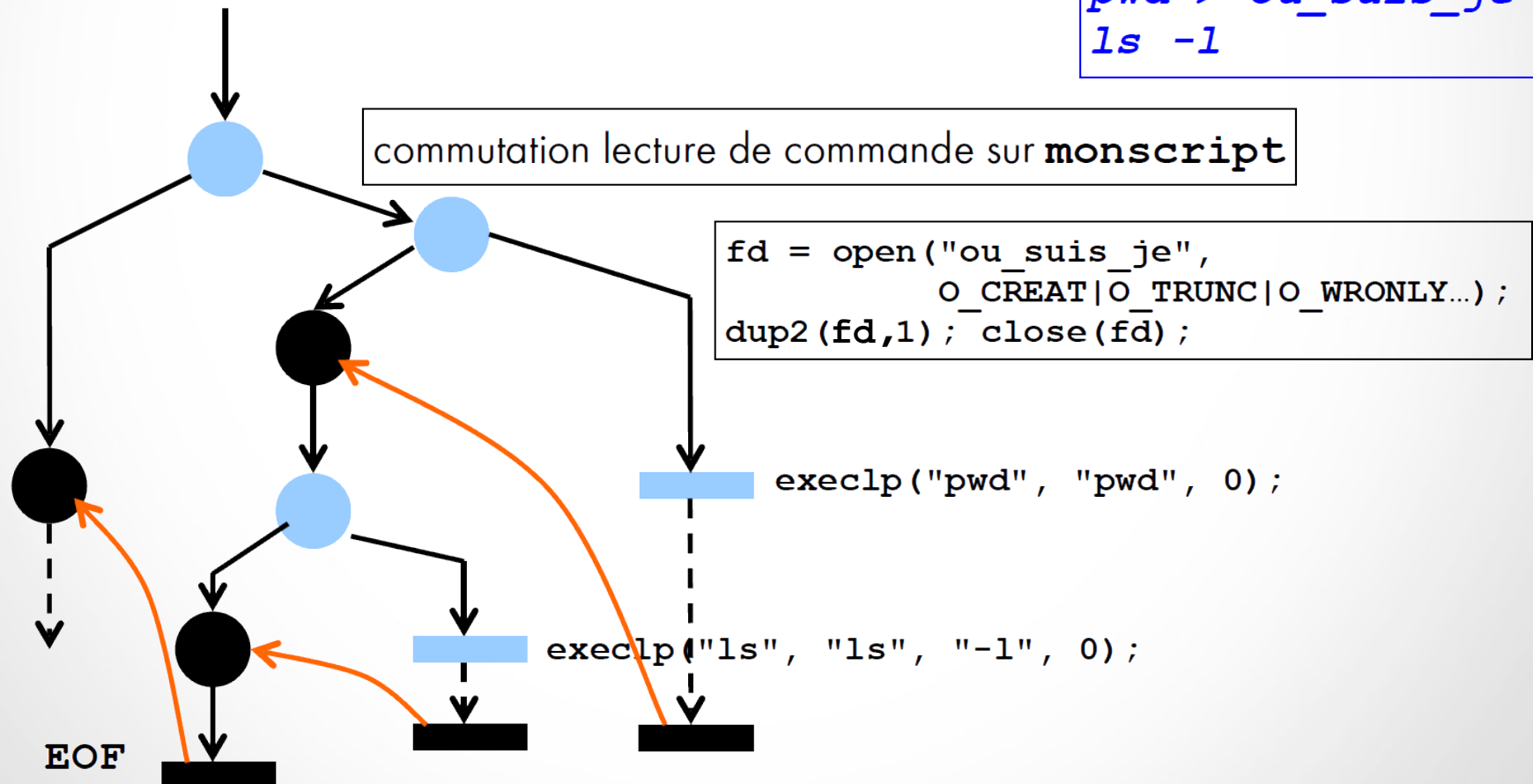


Exécution du Shell Script (fichier de commandes)

% *monscript*

monscript

```
pwd > ou_suis_je
ls -l
```



Exécution du Shell

Commande interne (builtin)

% *monscript*

monscript

```
cd foo
ls -l
```

commutation lecture de commande sur **monscript**

Exécution directe
`chdir("foo");`

`execvp("ls", "ls", "-l", 0);`

EOF