INSTITUT Mines-Télécom

TELECOM SudParis

Les tubes

CSC 3102

Introduction aux systèmes d'exploitation

Gaël Thomas

Présentation du cours

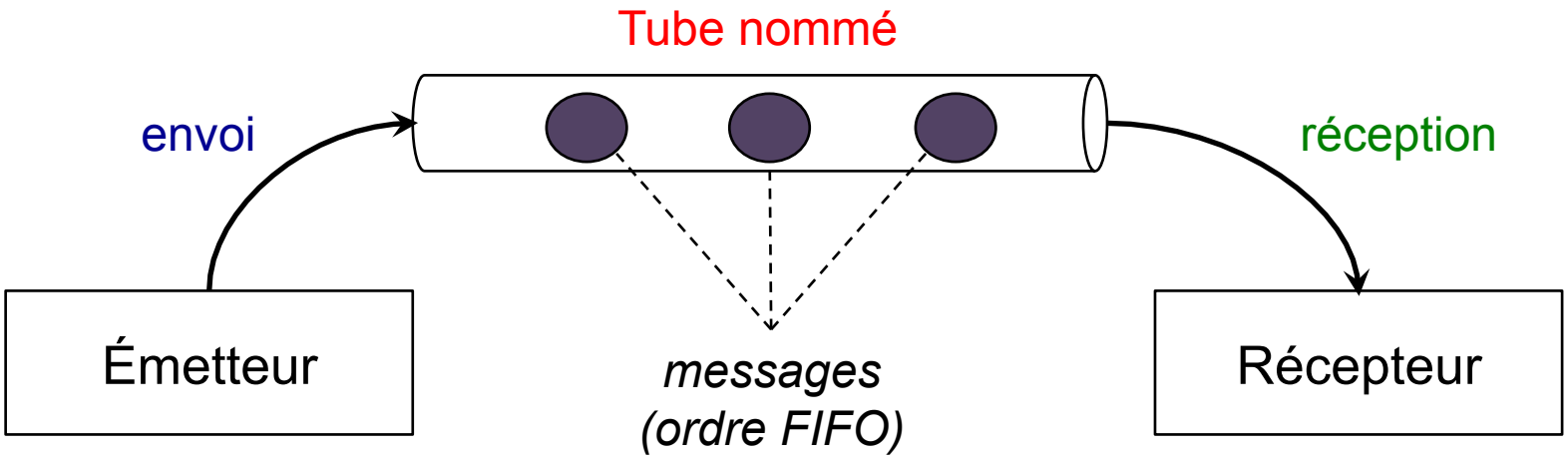
- Contexte : comprendre comment les processus interagissent
- Objectifs :
  - Savoir utiliser les tubes
- Notions clés :
  - Les tubes (`mkfifo`, redirections avancées, `|`)

Les tubes

- Tube = mécanisme de communication par envoi de messages
  - Via un canal de communication
  - Message = données quelconques
  - Pas de perte de message,
  - Réception dans l'ordre d'émission
    - (Réception dite FIFO pour First In First Out

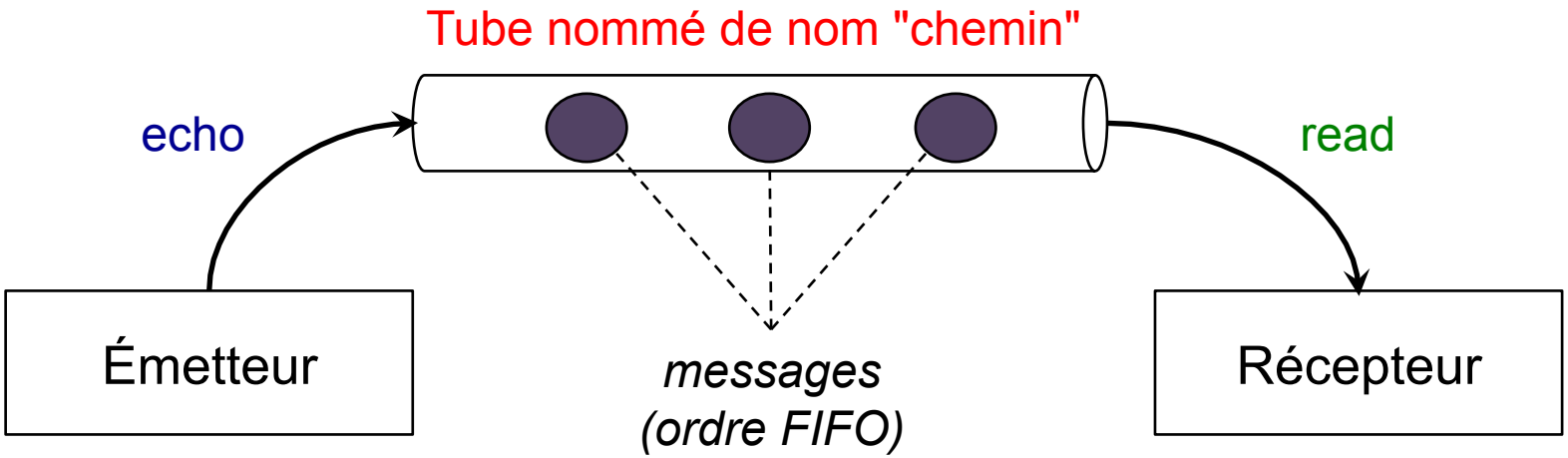
- Utilisé pour l'échange de messages complexes entre processus
  - Base de données + serveur Web
  - Processus d'affichage de notifications pour les autres processus
  - De façon générale, pour mettre en place une architecture de type client/serveur

Tubes nommés



- Principe :
  - Un tube est un fichier spécial dans le système de fichiers
  - L'émetteur écrit dans le tube
  - Le récepteur lit dans le tube

Principe de mise en œuvre



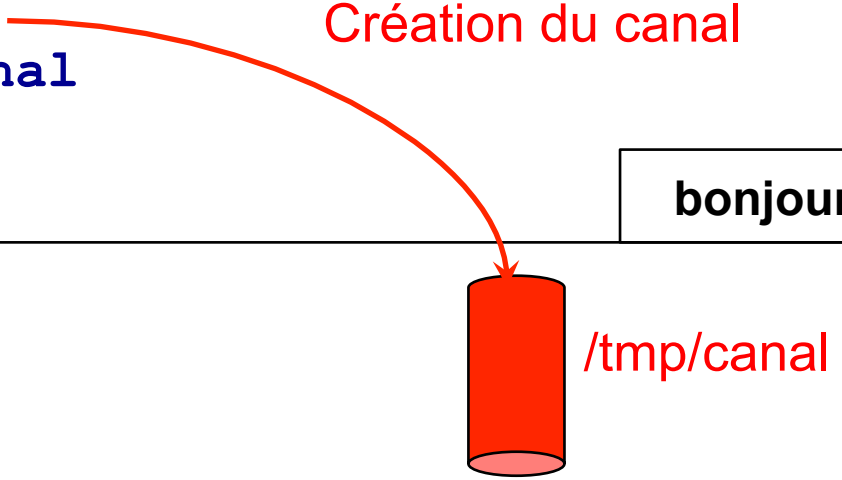
- Principe de mise en œuvre :
  - `mkfifo chemin` : crée un tube nommé (visible avec `ls`)
  - `echo message > chemin` : écrit dans le tube nommé
  - `read message < chemin` : lit à partir du tube nommé
    (bloque si pas encore de message dans le tube)
  - `rm chemin` : détruit le tube nommé

Principe de mise en œuvre

```
#!/bin/bash

mkfifo /tmp/canal
echo "Bonjour" >/tmp/canal
read line </tmp/canal
echo "Reçoit $line"
```

bonjour.sh




```
#!/bin/bash

read line </tmp/canal
echo "Reçoit $line"
echo "Au revoir" >/tmp/canal
```

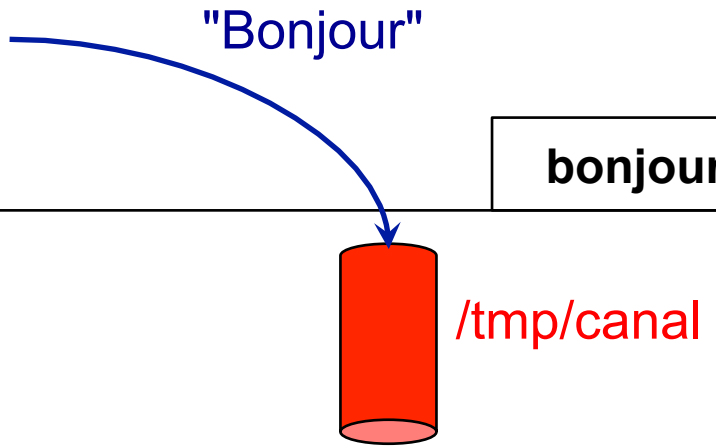
au-revoir.sh

Principe de mise en œuvre

```
#!/bin/bash

mkfifo /tmp/canal
echo "Bonjour" >/tmp/canal
read line </tmp/canal
echo "Reçoit $line"
```

bonjour.sh




```
#!/bin/bash

read line </tmp/canal
echo "Reçoit $line"
echo "Au revoir" >/tmp/canal
```

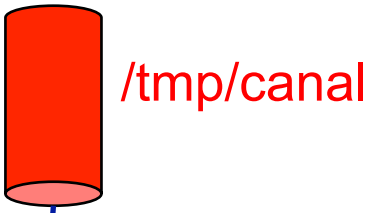
au-revoir.sh

Principe de mise en œuvre

```
#!/bin/bash

mkfifo /tmp/canal
echo "Bonjour" >/tmp/canal
read line </tmp/canal
echo "Reçoit $line"
```

bonjour.sh




```
#!/bin/bash
                          "Bonjour"
read line </tmp/canal
echo "Reçoit $line"
echo "Au revoir" >/tmp/canal
```

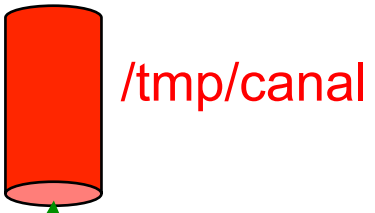
au-revoir.sh

Principe de mise en œuvre

```
#!/bin/bash

mkfifo /tmp/canal
echo "Bonjour" >/tmp/canal
read line </tmp/canal
echo "Reçoit $line"
```

bonjour.sh




```
#!/bin/bash

read line </tmp/canal
echo "Reçoit $line"
echo "Au revoir" >/tmp/canal
```

"Au revoir"

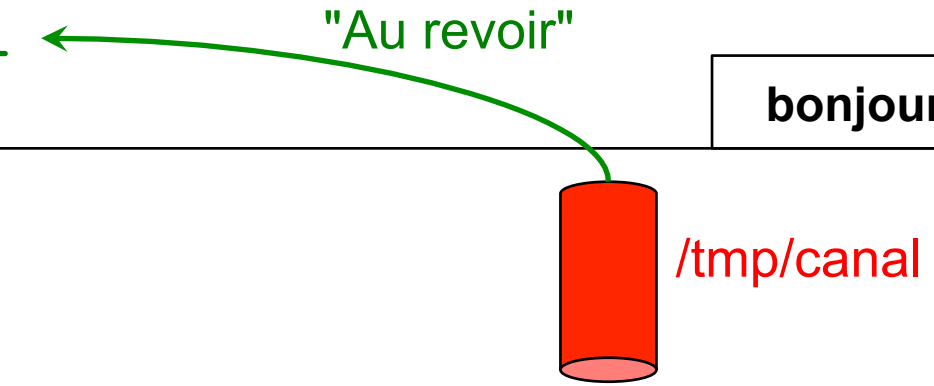
au-revoir.sh

Principe de mise en œuvre

```
#!/bin/bash

mkfifo /tmp/canal
echo "Bonjour" >/tmp/canal
read line </tmp/canal
echo "Reçoit $line"
```

bonjour.sh

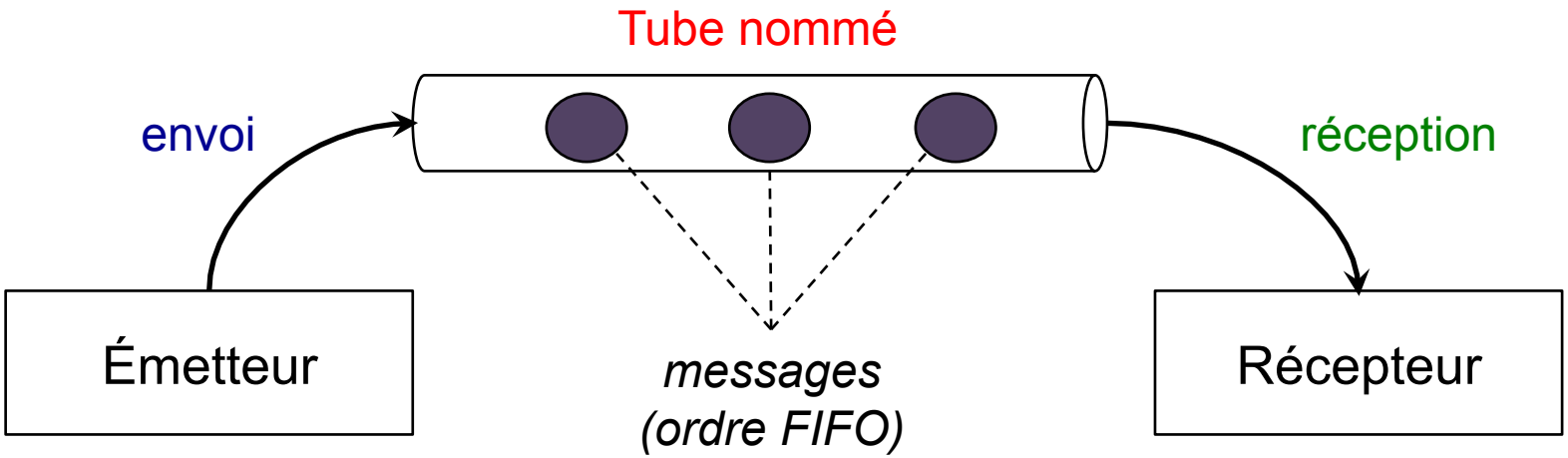



```
#!/bin/bash

read line </tmp/canal
echo "Reçoit $line"
echo "Au revoir" >/tmp/canal
```

au-revoir.sh

La vie est malheureusement complexe !



- Les contraintes
  - Le récepteur bloque en attendant un message
  - Ouverture bloque si interlocuteur n'existe pas

    `read line < f` bloque en attendant `echo msg > f` et vice-versa
  - Erreur si émission alors qu'il n'existe plus de récepteur

    (voir l'exemple donné dans les prochains transparents)

La vie est malheureusement complexe !

- On fait souvent des envois/réceptions dans des boucles

```
#!/bin/bash
mkfifo tube
while true; do
  echo "yes" >tube
done
```

emetteur_tube.sh

```
#!/bin/bash

while true; do
  read line <tube
done
```

recepteur_tube.sh

Démarre `emetteur_tube.sh` puis `recepteur_tube.sh`

- La plupart des tours de boucle s'exécutent sans problème
  - L'émetteur émet "yes"
  - Le récepteur reçoit le "yes"
- Mais parfois...

La vie est malheureusement complexe !

- On fait souvent des envois/réceptions dans des boucles

```
#!/bin/bash
mkfifo tube
while true; do
  echo "yes" >tube
done
```

emetteur_tube.sh

```
#!/bin/bash

while true; do
  read line <tube
done
```

recepteur_tube.sh

```
1. Ouvre tube
```

La vie est malheureusement complexe !

- On fait souvent des envois/réceptions dans des boucles

```
#!/bin/bash
mkfifo tube
while true; do
  echo "yes" >tube
done
```

emetteur_tube.sh

```
#!/bin/bash

while true; do
  read line <tube
done
```

recepteur_tube.sh

```
1. Ouvre tube
                                        2. Ouvre tube
```

La vie est malheureusement complexe !

- On fait souvent des envois/réceptions dans des boucles

```
#!/bin/bash
mkfifo tube
while true; do
  echo "yes" >tube
done
```

emetteur_tube.sh

```
#!/bin/bash

while true; do
  read line <tube
done
```

recepteur_tube.sh

```
1. Ouvre tube
                                        2. Ouvre tube
3. Écrit "yes"
```

La vie est malheureusement complexe !

- On fait souvent des envois/réceptions dans des boucles

```
#!/bin/bash
mkfifo tube
while true; do
  echo "yes" >tube
done
```

emetteur_tube.sh

```
#!/bin/bash

while true; do
  read line <tube
done
```

recepteur_tube.sh

```
1. Ouvre tube
                                        2. Ouvre tube
3. Écrit "yes"
4. Ferme tube
```

La vie est malheureusement complexe !

- On fait souvent des envois/réceptions dans des boucles

```
#!/bin/bash
mkfifo tube
while true; do
  echo "yes" >tube
done
```

emetteur_tube.sh

```
#!/bin/bash

while true; do
  read line <tube
done
```

recepteur_tube.sh

```
1. Ouvre tube
                                        2. Ouvre tube
3. Écrit "yes"
4. Ferme tube
5. Ré-ouvre tube(ok car ∃ récepteur)
```

La vie est malheureusement complexe !

- On fait souvent des envois/réceptions dans des boucles

```
#!/bin/bash
mkfifo tube
while true; do
  echo "yes" >tube
done
```

emetteur_tube.sh

```
#!/bin/bash

while true; do
  read line <tube
done
```

recepteur_tube.sh

```
1. Ouvre tube
                                        2. Ouvre tube

3. Écrit "yes"
4. Ferme tube
5. Ré-ouvre tube(ok car ∃ récepteur)
                                        6. Lit yes
```

La vie est malheureusement complexe !

- On fait souvent des envois/réceptions dans des boucles

```
#!/bin/bash
mkfifo tube
while true; do
  echo "yes" >tube
done
```

emetteur_tube.sh

```
#!/bin/bash

while true; do
  read line <tube
done
```

recepteur_tube.sh

```
1. Ouvre tube
                                        2. Ouvre tube

3. Écrit "yes"
4. Ferme tube
5. Ré-ouvre tube(ok car ∃ récepteur)
                                            6. Lit yes
                                        7. Ferme tube
```

La vie est malheureusement complexe !

- On fait souvent des envois/réceptions dans des boucles

```
#!/bin/bash
mkfifo tube
while true; do
  echo "yes" >tube
done
```

emetteur_tube.sh

```
#!/bin/bash

while true; do
  read line <tube
done
```

recepteur_tube.sh

```
1. Ouvre tube
                                        2. Ouvre tube
3. Écrit "yes"
4. Ferme tube
5. Ré-ouvre tube(ok car ∃ récepteur)
                                            6. Lit yes
                                        7. Ferme tube
8. Écriture ⇒ plantage (silencieux) : pas de récepteur
```

Tube et redirection avancée

- Pour éviter ces fermetures intempestives de tubes

On préconise dans ce cours de toujours ouvrir un tube avec une redirection avancée en lecture/écriture

  - Si le récepteur ferme le tube, l'émetteur agissant comme récepteur, plus de plantage

  - Effet connexe : ni l'émetteur ni le récepteur ne bloquent pendant l'ouverture s'il n'existe pas encore d'interlocuteur

Tube et redirection avancée

- **Toujours ouvrir un tube avec une redirection avancée en lecture/écriture**

```
#!/bin/bash
mkfifo tube
exec 3<>tube
while true; do
  echo "yes" >&3
done
```

emetteur_tube_exec.sh

```
#!/bin/bash

exec 3<>tube
while true; do
  read line <&3
done
```

recepteur_tube_exec.sh

(redirection avancée pas nécessaire chez le récepteur dans ce cas, mais bonne habitude à prendre car souvent, un récepteur est aussi un émetteur)

Retour sur les tubes anonymes

- Tube anonyme : comme tube nommé, mais sans nom

- Le « `|` » entre deux processus shell crée un tube anonyme

```
cmd_gauche | cmd_droite
```

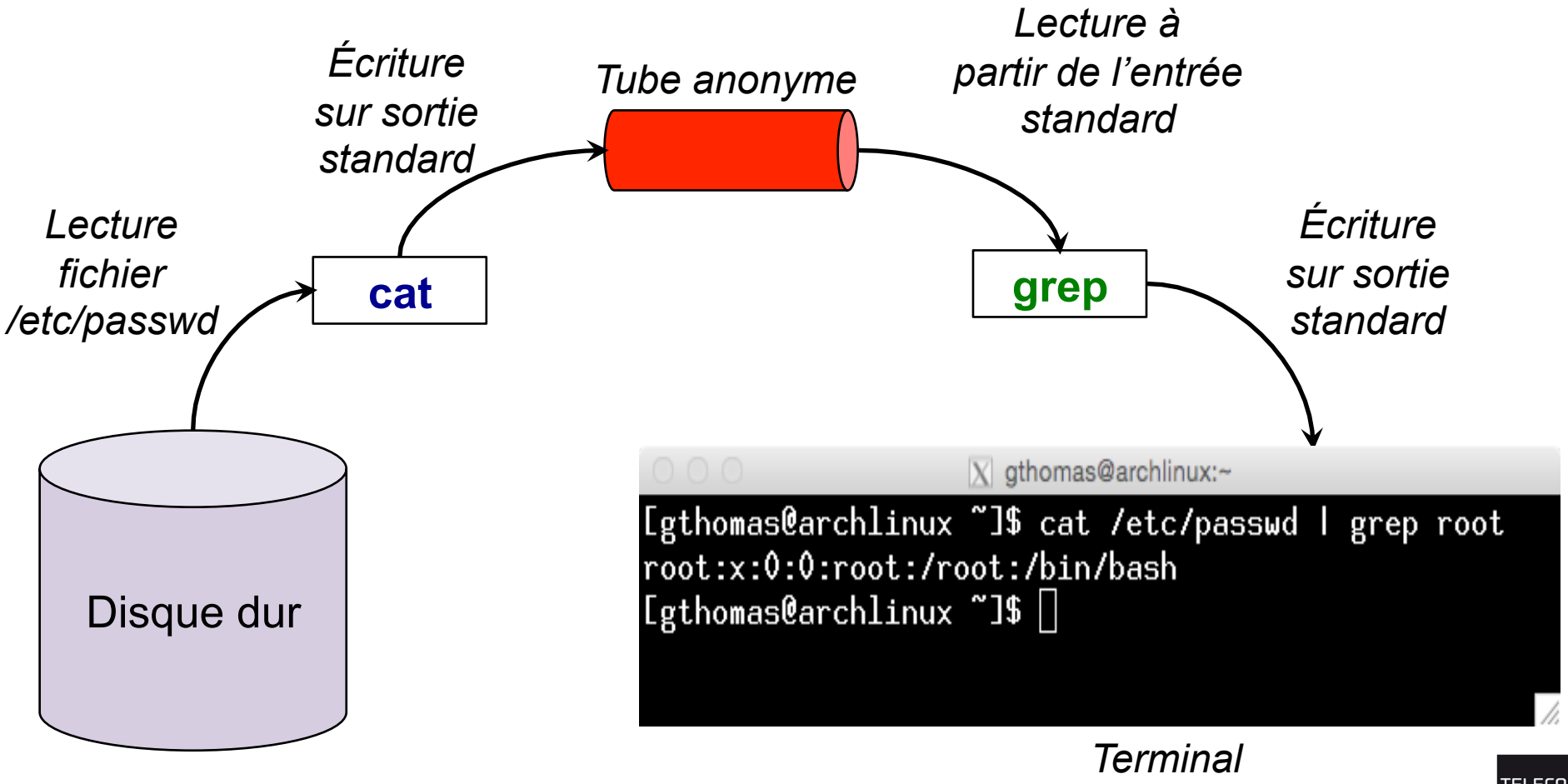
  - Sortie standard de `cmd_gauche` connectée au tube
  - Entrée standard de `cmd_droite` connectée au tube

- A haut niveau, un peu comme si on exécutait

```
mkfifo tube-avec-nom
cmd_gauche > tube-avec-nom &
cmd_droite < tube-avec-nom
```

Retour sur les tubes anonymes

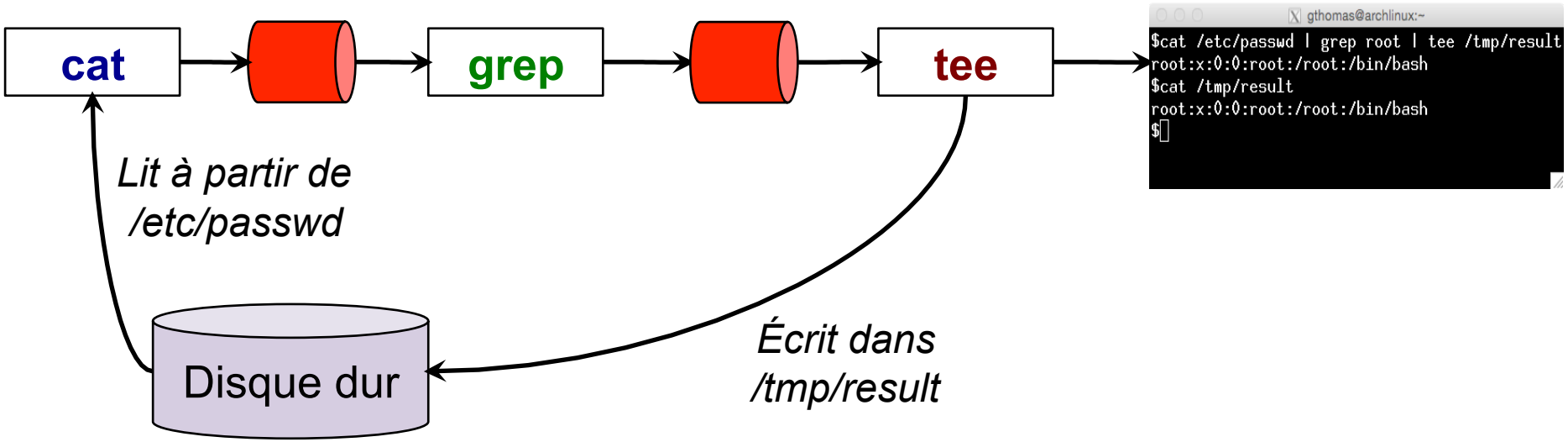
```
cat /etc/passwd | grep root
```



Commande utile : `tee`

- Écrit les données lues à partir de l'entrée standard
  - Sur la sortie standard
  - Et dans un fichier passé en argument

- Exemple : **`cat`** `/etc/passwd | ` **`grep`** `root | ` **`tee /tmp/result`**



Notions clés

- Tube nommé
  - Fichier spécial dans le système de fichier
  - Envoi de messages de taille quelconque, ordre de réception = ordre d'émission, pas de perte de message
  - `mkfifo nom-tube` : crée un tube nommé mon-tube
  - Toujours utiliser des redirections avancées

- Tubes anonyme (|)
  - Comme un tube nommé, mais sans nom dans le système de fichier

À vous de jouer !