Systèmes d'exploitation

Chapitre 5 : Systèmes de fichiers

Chargé de cours :
Emery Kouassi Assogba
Tél : 95 22 20 73
Emery.assogba@uac.bj/
emery.assogba@ptgfengineering.com

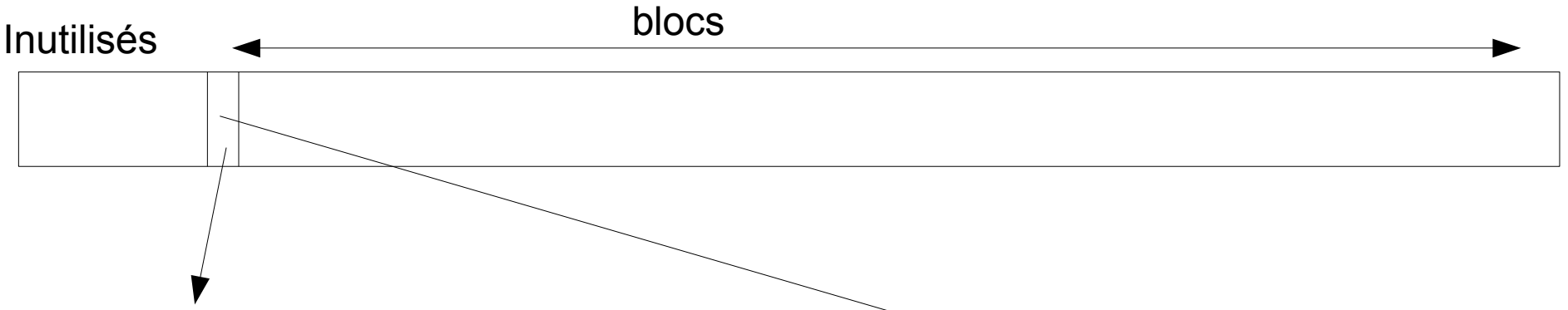
Implémentation des systèmes de fichiers

- Problèmes à résoudre
  - Comment allouer et répertorier les secteurs disponibles sur le disque ?
  - Comment représenter un fichier sur disque
    - un fichier sera une suite de secteurs, pas nécessairement contigus
    - comment savoir quels secteurs représentent un fichier et dans quel ordre ?
  - Comment représenter un répertoire sur disque
    - un répertoire sera un suite de secteurs, pas nécessairement contigus
    - comment savoir quels secteurs représentent un répertoire et dans quel ordre ?

Types de fichiers

- Fichier texte
  - Données en ASCII
- Fichier exécutable
  - Via un interpréteur
    - Fichier débute par magic number #!path_interpréteur
    - Exemple : #!/bin/sh
  - Directement
    - Fichier débute par magic number indiquant le format du fichier ELF, A.OUT, ...
    - Structure définie (code, text, ...)
- Archive
- Fichiers spéciaux (/dev, /proc, ...)

ISO9660 level1 - CDROM

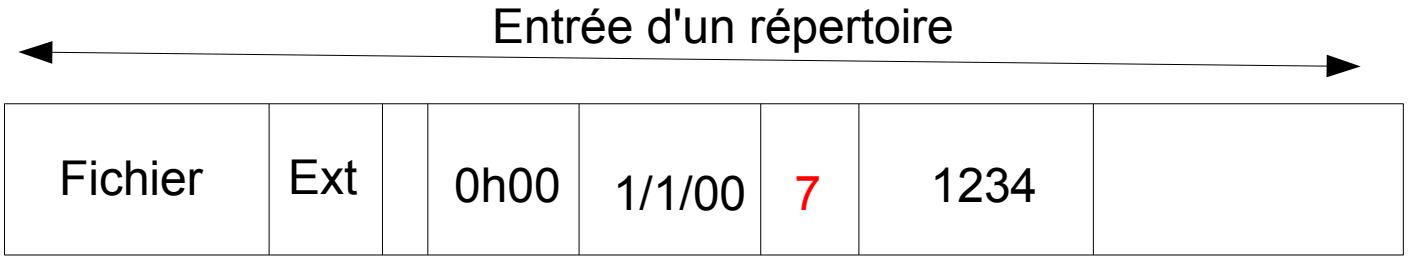


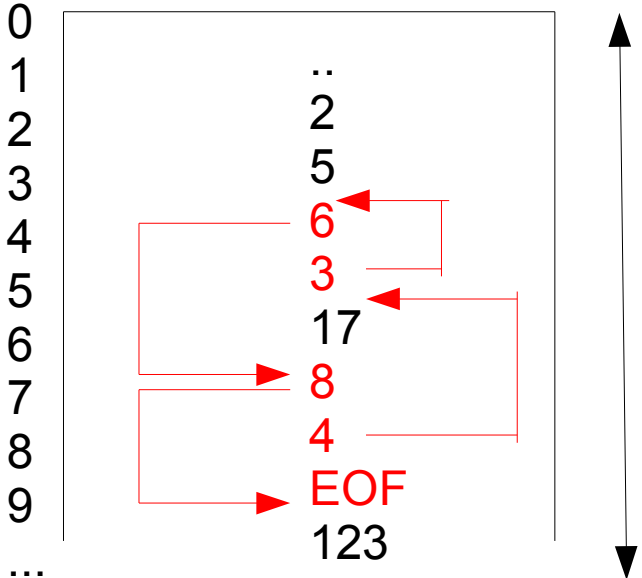
- Primary vol. desc.
- System id
- Volume id
- Publisher id
- Fichier abstract
- copyright copyright
- info bibliographique
- taille blocs
- # blocks
- premier bloc du répertoire /

- répertoire /
- N blocs consécutifs
- Entrée d'un répertoire
- (ordre alphabétique, sauf ./..)
- longueur de l'entrée
- localisation (bloc début et CD#)
- taille
- date et heure
- attributs
- caché, fichier/directory, ...
- Nom du fichier
- Marqueur de fin du répertoire

FAT

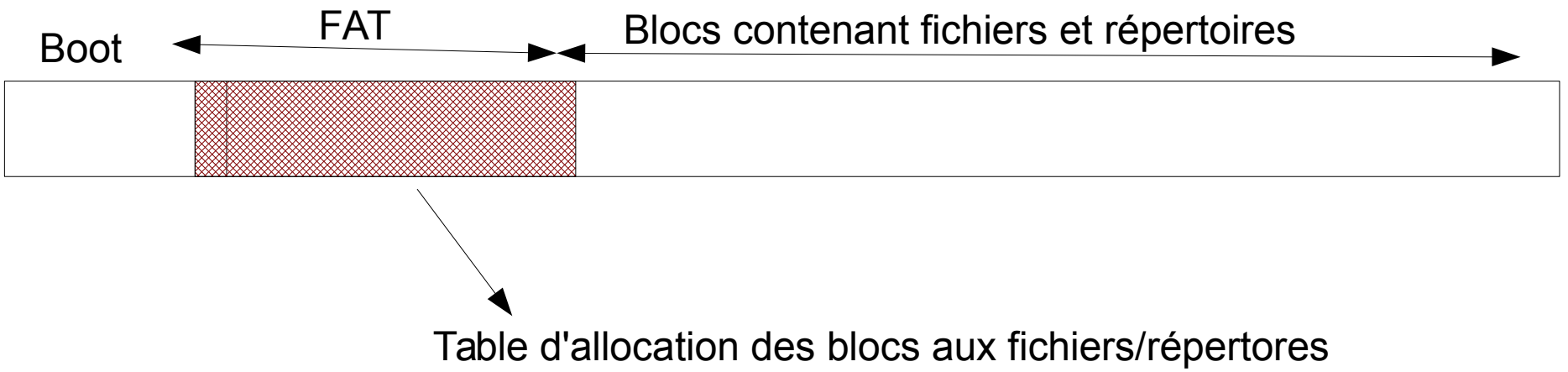
- Comment localiser les blocs d'un fichier ?
  - Diviser le disque en blocs (ex: 4 Kbytes)
  - Utiliser une table d'allocation des fichiers
    - stockée au début du disque et cachée en RAM





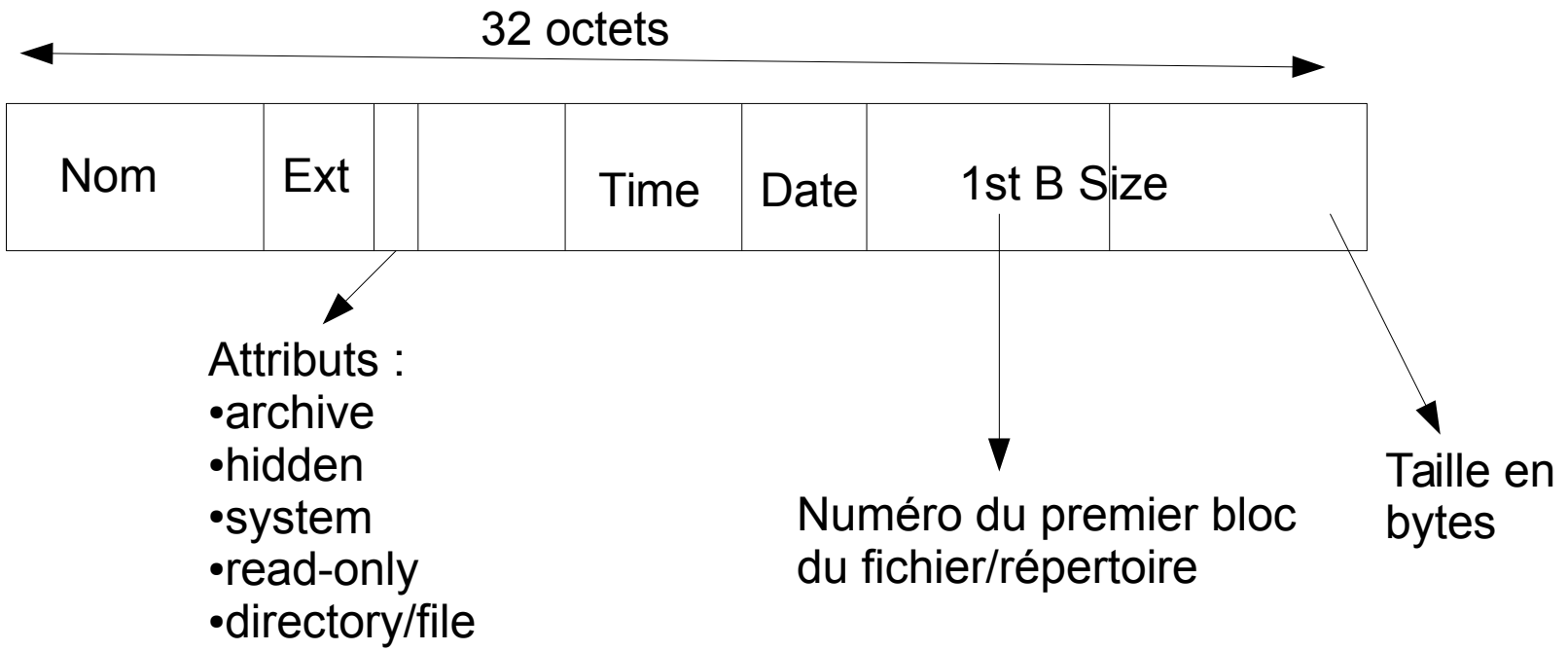
File Allocation Table
Stockée sur disque à endroit bien connu

Système de fichiers MSDOS



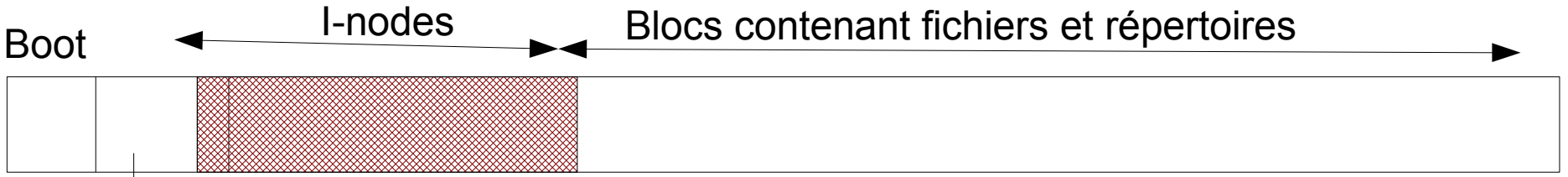
MSDOS

- Répertoire
  - Nombre variable d'entrées de 32 octets



- Différentes variantes utilisées par Microsoft
  - extension pour longs noms dans Windows 95
  - extension pour grands disques

Système de fichiers Unix

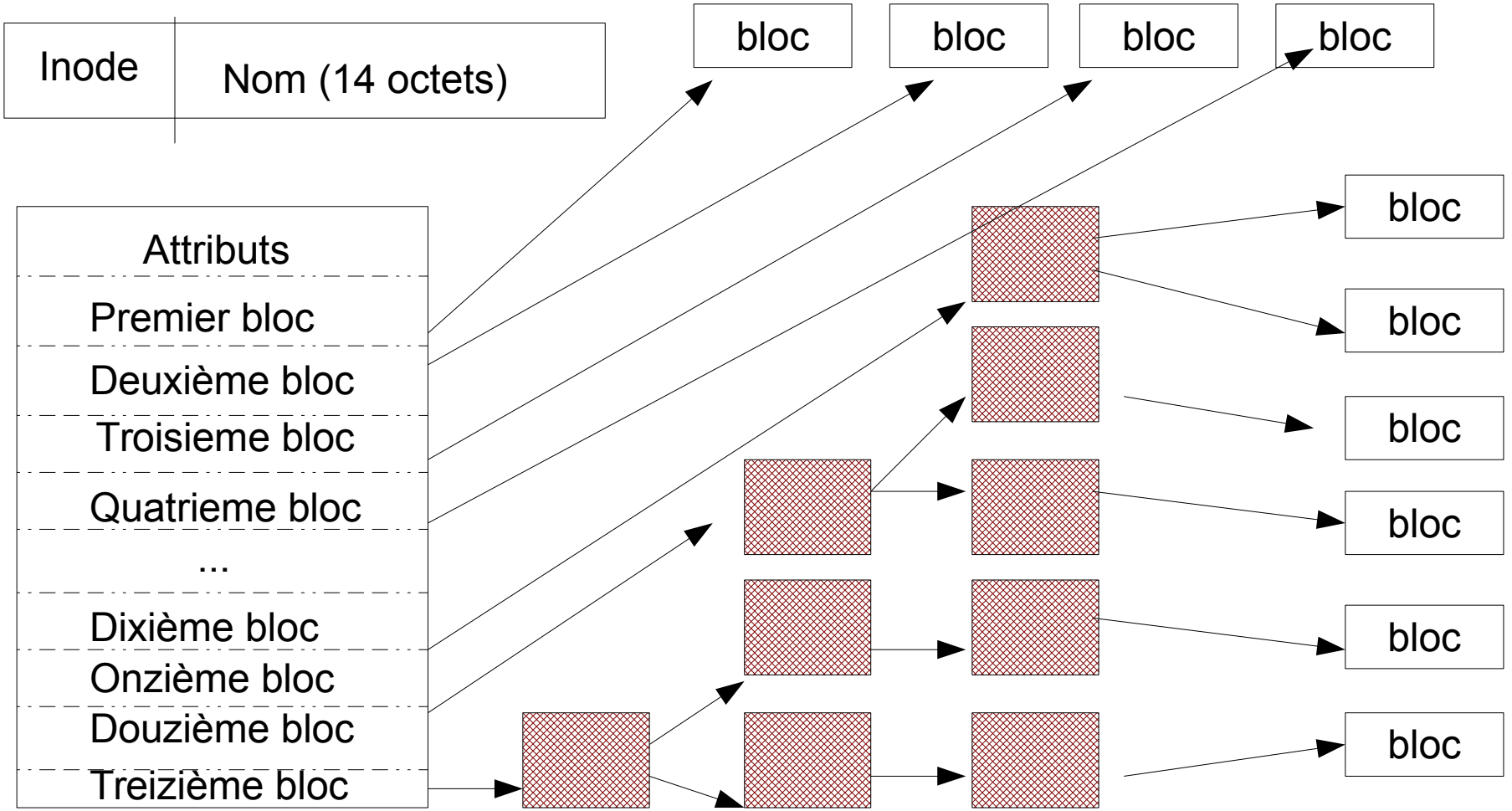


Un I-node connu pointe vers répertoire /

Superblock
- taille du filesystem
- taille des blocs
- nombre inodes
- position liste blocs libres

Systèmes de fichiers Unix v7

- Représentation d'un répertoire
  - Nombre variable d'entrées de 16 octets

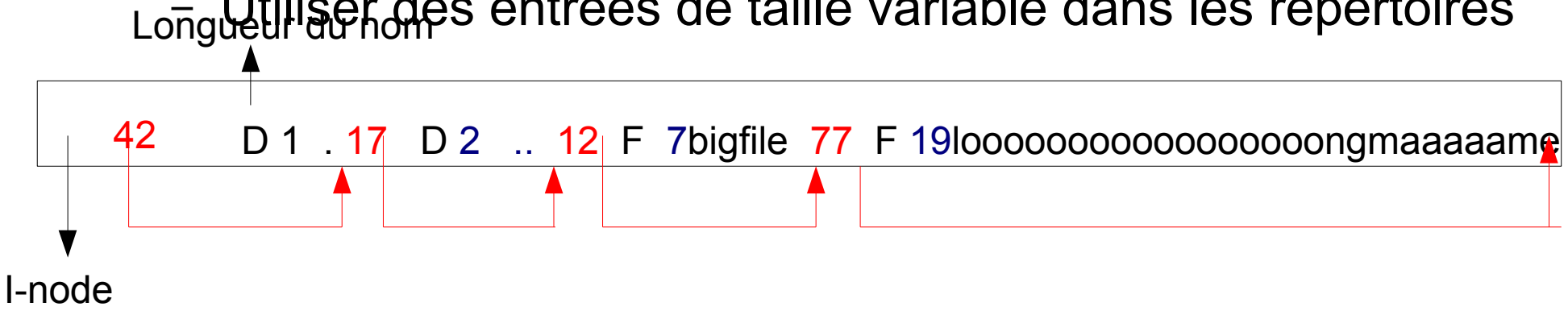


Les I-nodes d'Unix v7

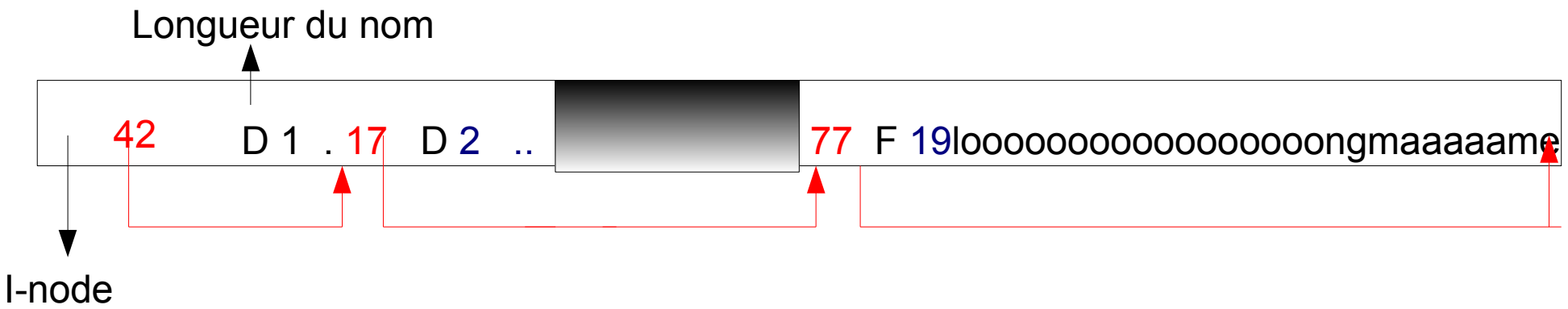
- Contenu de chaque inode
  - Mode : drapeau sur 16 bits
    - type : normal, répertoire, char/block device, FIFO pipe
    - permissions : rwx rwx rwx
  - Link count
    - nombre de liens vers cet inode
  - Propriétaire
    - userid, groupid
  - Taille en octets
  - Génération
    - compteur incrémenté à chaque réutilisation de l'inode
  - Blocs sur disque (directs, indirects)
  - Date de dernier accès au fichier (atime)
  - Date de dernière modification du fichier (mtime)
  - Date de dernière modification de l'inode (ctime)

Système de fichiers Unix récents

- Comment supporter longs noms de fichiers ?
  - Utiliser des entrées de taille variable dans les répertoires

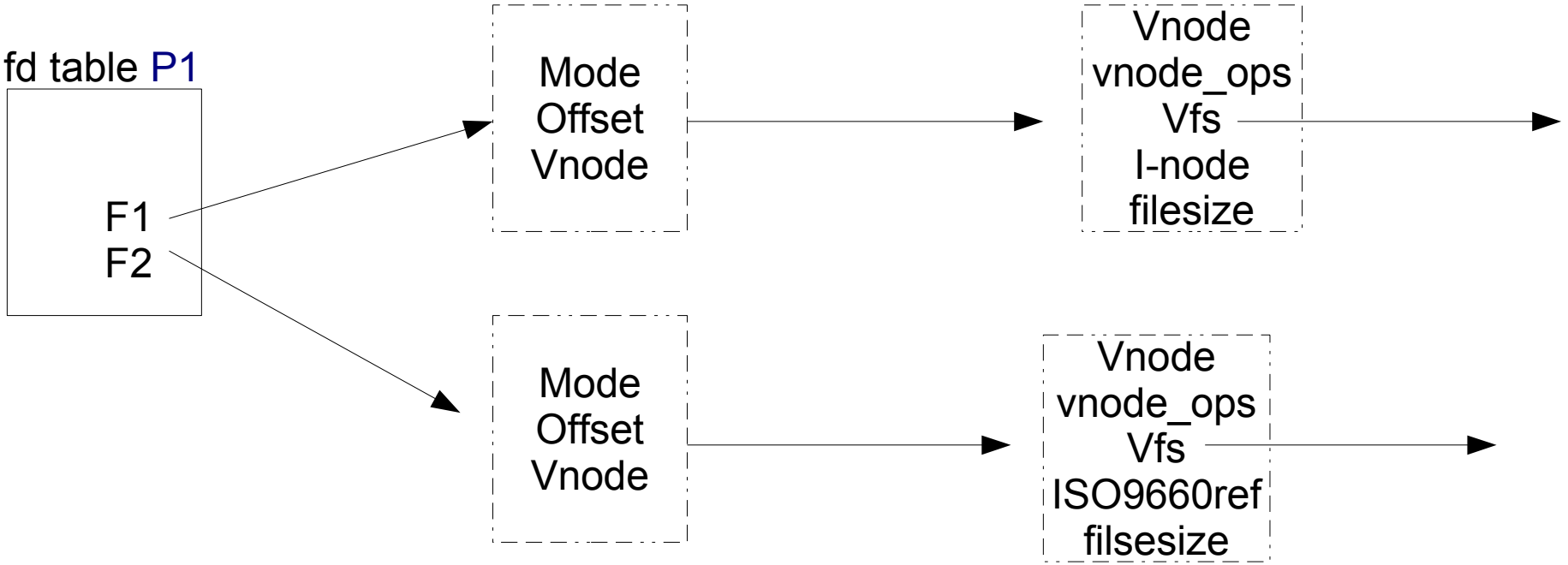


  - accès à un fichier dans un gros répertoire
  - après rm bigfile



Virtual FileSystem

- Comment utiliser plusieurs filesystems sur une même machine ?
  - Solution
    - définir un filesystem virtuel qui englobera tous les
    - filesystems présents sur la machine
    - tous les accès se feront via ce FS virtuel



Gestion des blocs libres

- Comment l'OS peut-il connaître quels sont les blocs libres sur le disque ?
  - Parcourir tous les inodes pour voir quels blocs sont référencés
    - Inefficace mais utile en cas de problème sur le disque
  - Maintenir sur disque et/ou en mémoire une liste liée contenant l'ensemble des blocs libres
    - Mise à jour de la liste à chaque ajout/suppression de bloc
- Maintenir sur disque et/ou en mémoire un bitmap indiquant pour l'ensemble des blocs les libres
    - Occupe moins d'espace disque qu'une liste liée

Inconsistence du système de fichiers

- Problème
  - De nombreuses opérations nécessitent plusieurs accès disque pour être correctement réalisées
    - Suppression d'un fichier
      - Retirer l'inode du répertoire
      - Retirer les blocs de la liste/table des blocs libres
      - Mettre à jour l'inode
    - Ajout d'un lien
      - Mise à jour du répertoire
      - Mise à jour du link count dans l'inode
      - ...
  - Que se passe-t-il si l'ordinateur s'arrête au milieu de ces opérations ?

Inconsistence du système de fichiers (2)

- Vérification de la consistence et si nécessaire correction du système de fichiers
- Principe de fsck
  - Parcourir l'ensemble des répertoires pour vérifier quels inodes sont alloués
    - Détection des fichiers/répertoires « perdus »
  - Parcourir l'ensemble des inodes pour déterminer la liste des blocs alloués
    - Comparer la liste ainsi calculée avec la table/liste des blocs libres et corriger en cas d'inconsistence
      - Bloc associé à aucun inode
      - Bloc associé à un inode
      - Bloc associé à plusieurs fichiers
  - Mise à jour du système de fichiers, éventuellement avec l'aide de l'administrateur

Virtual fileSystem (2)

- Contenu d'un vnode
  - vfs
    - pointeur vers descripteur du filesystem auquel le vnode appartient
  - vnode_ops
    - pointeur vers la liste de fonctions implémentant les accès classiques à ce fichier sur le filesystem
      - open
      - close
      - read
      - write
      - link
      - mkdir
      - rmdir
      - ...
  - référence vers la structure décrivant le fichier dans le filesystem réel

Locks

- Comment permettre à plusieurs processus de manipuler le même fichier (base de données) ?
  - Processus coopératifs
    - utiliser des sémaphores pour manipuler l'accès à la base de données, e.g. readers-writers
  - Processus non-coopératifs
    - Comment éviter que deux processus distincts ne modifient en même temps un fichier ?
      - exemple : sendmail et elm/pine
    - Solution
      - permettre à un processus de demander au kernel de bloquer les accès en lecture à une zone particulière d'un fichier
      - advisory locking
        - les processus doivent vérifier les locks eux-mêmes
      - mandatory locking
        - le kernel vérifie les locks à chaque accès

Lockf

- Extrait de la page de manuel

  #include <unistd.h>

  int lockf(int fd, int cmd, off_t len);

  - DESCRIPTION

    Apply, test or remove a lock on a section of an open file. ... the section consists of byte positions pos..pos+len-1...

    Valid operations are :

    F_LOCK

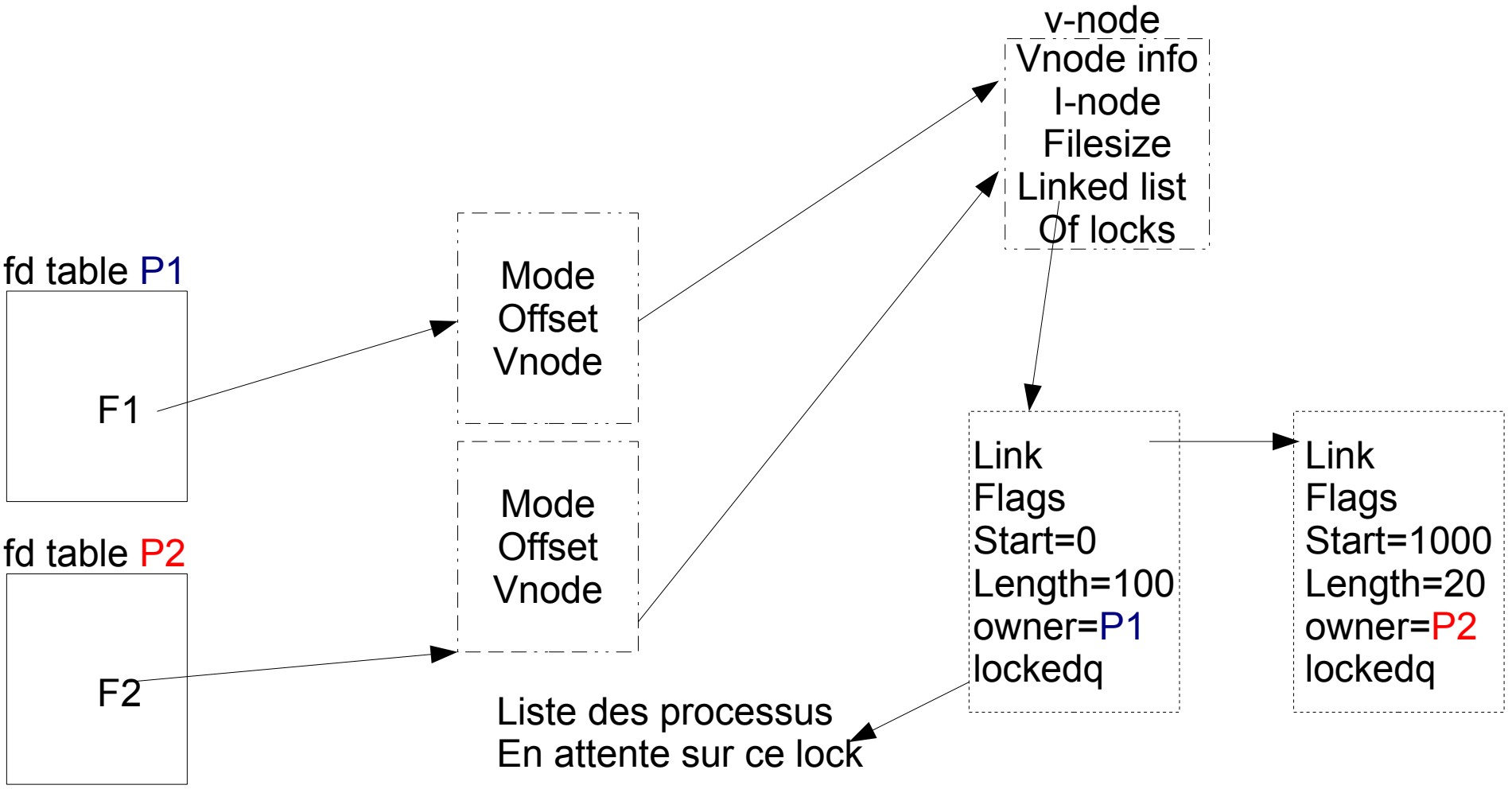
    Set an exclusive lock on the specified section of the file. If (part of) this section is already locked, the call blocks until the previous lock is released. ... File locks are released as soon as the process holding the locks closes ... the file. A child process does not inherit these locks.

    F_ULOCK

    Unlock the indicated section of the file.

Implémentation des Locks

- P1: f1=open(“F1”); lock(fd,F_LOCK,100);
- P2: f2=open(“F1”); lseek(fd,SEEK_SET,1000);
- lock(fd,F_LOCK,20)



Utilisateurs et groupes

- Sous Unix, chaque utilisateur dispose d'un
  - UserId (représenté par un entier)
  - root (userid=0) est l'administrateur système
  - mots de passe stockés dans /etc/passwd

    ...

    root:12345abc:0:0:root:/root:/bin/bash

    ...
- Chaque utilisateur fait partie d'un ou de plusieurs groupes
  - groupe principal dans /etc/passwd
  - groupes secondaires dans /etc/groups
    - ...
    - tape:x:26:knoppix,backup
    - users:x:100:obo,suh,bqu

Permissions du système de fichiers

- Identification du propriétaire (userid/groupid) de chaque fichier/répertoire
  - seul le propriétaire et root peuvent modifier les permissions
- Permissions de base
  - -rw-rw-r-- 1 obo users … /tmp/t.ps
- Permissions applicables au propriétaire
  - r : lecture
  - w : écriture
  - x : exécution
- Permissions applicables à tout membre du même groupe que le groupe du fichier
- Permissions applicables à n'importe qui
- Le kernel vérifie les permissions à l'ouverture du fichier uniquement, pas pour chaque accès

Permissions du système de fichiers (2)

- Permissions avancées sur les fichiers
  - suid bit
    - durant l'exécution de ce fichier, l'utilisateur courant deviendra le propriétaire du fichier
  - Exemple d'utilisation
    - -rwsr-xr-x 1 root root ... /usr/bin/passwd
- sgid
  - durant l'exécution de ce fichier, le groupe courant deviendra celui du propriétaire du fichier
  - Exemple d'utilisation
    - -rwxr-sr-x 1 root tty ... /usr/bin/write
    - -rwxr-sr-x 1 root tty ... /usr/bin/wall
    - crw-rw-rw- 1 root tty 5, 0 ... /dev/tty

Permissions du système de fichiers (3)

- Permissions sur les répertoires
  - r : lecture du contenu du répertoire avec opendir et readdir (ex: ls)
  - w : modification du contenu du répertoire
  - x : possibilité d'accéder aux inodes des fichiers contenus dans le répertoire ou possibilité d'utiliser ce répertoire comme répertoire courant
  - sticky bit
  - les fichiers de ce répertoire ne peuvent être renommés/supprimés que par leur propriétaire, root ou le propriétaire du répertoire
    - ls -l /

      ...

      drwxrwxrwt 42 root root ... tmp

Informations maintenies par le kernel

- Informations maintenues par le kernel pour chaque processus courant
  - uid
    - (real) userid de l'utilisateur qui a lancé le processus
  - gid
    - (real) groupid dans lequel l'utilisateur qui a lancé le processus se trouve
  - euid
    - (effective) userid de l'utilisateur avec les permissions duquel le processus s'exécute
  - egid
    - (effective) groupid du groupe avec les permissions duquel le processus s'exécute
  - groups
    - identification des autres groups auquel l'utilisateur qui a lancé le processus appartient

Appels systèmes

- Manipulation des userId et groupId
  - getuid()
    - userid de l'utilisateur qui a lancé le processus courant
  - geteuid()
    - userid effectif sous lequel le processus courant s'exécute
  - getgid()
    - groupid de l'utilisateur qui a lancé le processus courant
  - getegid()
    - groupid effectif sous lequel le processus courant s'exécute
  - setreuid(real_uid,effective_uid)
    - modifie les userid réel et effectif du processus courant
  - setregid(real_gid, effective_gid)
    - modifie les groupid réel et effectif du processus courant

Appels systèmes (2)

- manipulation des permissions
  - s=chmod(path,mode)
    - modification des permissions du fichier/répertoire path
  - s=access(path,mode)
    - vérification que le fichier/répertoire path est accessible par l'utilisateur réel en mode R_OK, W_OK, X_OK
    - access checks whether the process would be allowed to read, write or test for existence of the file... The check is done with the process's real uid and gid, rather than with the effective ids as is done when actually attempting an operation.
  - s=chown(path,owner,group)
  - s=fchown(fd,owner,group)
    - modifie le propriétaire d'un fichier/répertoire dont le chemin ou le file descriptor est connu

Procédure de login sous Unix

- -rwsr-xr-x 1 root root ... /bin/login
  - authentification sur base du username et mot de passe
    - vérification du mot de passe sur base de l'information stockée dans /etc/passwd
    - nombre maximum d'essais avant d'arrêter le login
  - si authentification réussie
    - userid devient userid authentifié
    - groupid devient groupid par défaut de cet utilisateur
    - association du clavier à stdin
    - association de l'écran à stdout et stderr
    - exécution du shell spécifié dans /etc/passwd
      - tous les processus créés par le shell hériteront du userid du shell