This work is licensed under a Creative Commons license

Attribution-NonCommercial-NoDerivatives 4.0 International
(CC BY-NC-ND 4.0)

You are free to:

Share copy and redistribute the material in any medium or format.

Under the following terms:

Attribution You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.

NonCommercial You may not use the material for commercial purposes.

NoDerivatives If you remix, transform, or build upon the material, you may not distribute the modified material.

Persistenza

Dispositivi a blocchi e File System

Giovanni Lagorio

giovanni.lagorio@unige.it
https://csec.it/people/giovanni_lagorio
Twitter & GitHub: zxgio

DIBRIS - Dipartimento di Informatica, Bioingegneria, Robotica e Ingegneria dei Sistemi
University of Genova, Italy

www.zenhack.it

Outline

Supporti di memoria

Chiamiamo comunemente "dischi" i dispositivi a blocchi di memoria secondaria

- dischi magnetici (hard/floppy disk di vario tipo)
  - La formattazione *a basso livello* prepara questa struttura logica sui dischi magnetici (ma, da decenni, sono venduti "pre-formattati")
- dischi ottici (DVD, Bluray, . . .)
- "pennette" USB
- SSD: Solid-state Storage Device/Drive/Disk/. . .
- . . .

dal punto di vista logico/astratto, sono tutti una sequenza di settori

- tipicamente da 512 byte l'uno

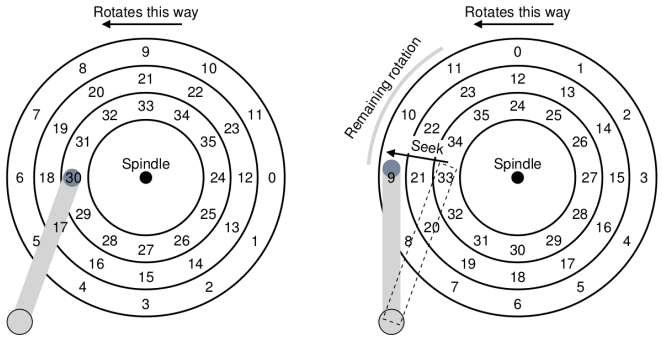
Dischi

Una volta erano *dischi*, cioè oggetti piatti e di forma circolare

`https://commons.wikimedia.org/wiki/File:HardDiskAnatomy.jpg`

Accedere a dati “vicini” costava meno che accedere a dati “lontani”

- ritardo rotazionale; tempo per spostare la testina da una traccia all’altra; . . .



`http://pages.cs.wisc.edu/~remzi/OSTEP/file-disks.pdf`

Interfaccia

- Un moderno SSD non è rotondo, non ruota, etc
- Alcune convenzioni, la terminologia e l'interfaccia sono sostanzialmente rimaste
  - un s.o. potrebbe usare un SSD, che esporta un'interfaccia "da HD", senza saperlo (la cosa non è ottimale, ovviamente)
- L'interfaccia semplificata che considereremo è: un disco ha n settori/blocchi, che possiamo indirizzare con "indirizzi" da 0 a $(n-1)$
  - notare che la "granularità" di accesso è il settore: 512 byte

Per maggiori dettagli fate riferimento al libro:
`http://pages.cs.wisc.edu/~remzi/OSTEP/file-disks.pdf`

Buffer cache

- L'I/O su disco è ordini di grandezza più costoso dell'accesso in RAM
- I sistemi Unix-like utilizzano una (unified) buffer cache
  - Una volta:

    buffer-cache per i blocchi disco acceduti tramite `read`/`write`

    page-cache per i file `mmap`pati in memoria; che, a sua volta, si appoggiava sulla buffer-cache
- Su Linux, per forzare la scrittura dei dati/metadati in cache: `sync(2)`

La presenza di memorie cache è la ragione per cui è importante "espellere" (=smontare) i drive prima di scollegarli!

Partizioni

Per varie ragioni (e.s. installare sistemi operativi diversi) si può voler partizionare un disco

- ogni partizione si può usare come un dispositivo a blocchi a sé stante
  - due standard usualmente utilizzati

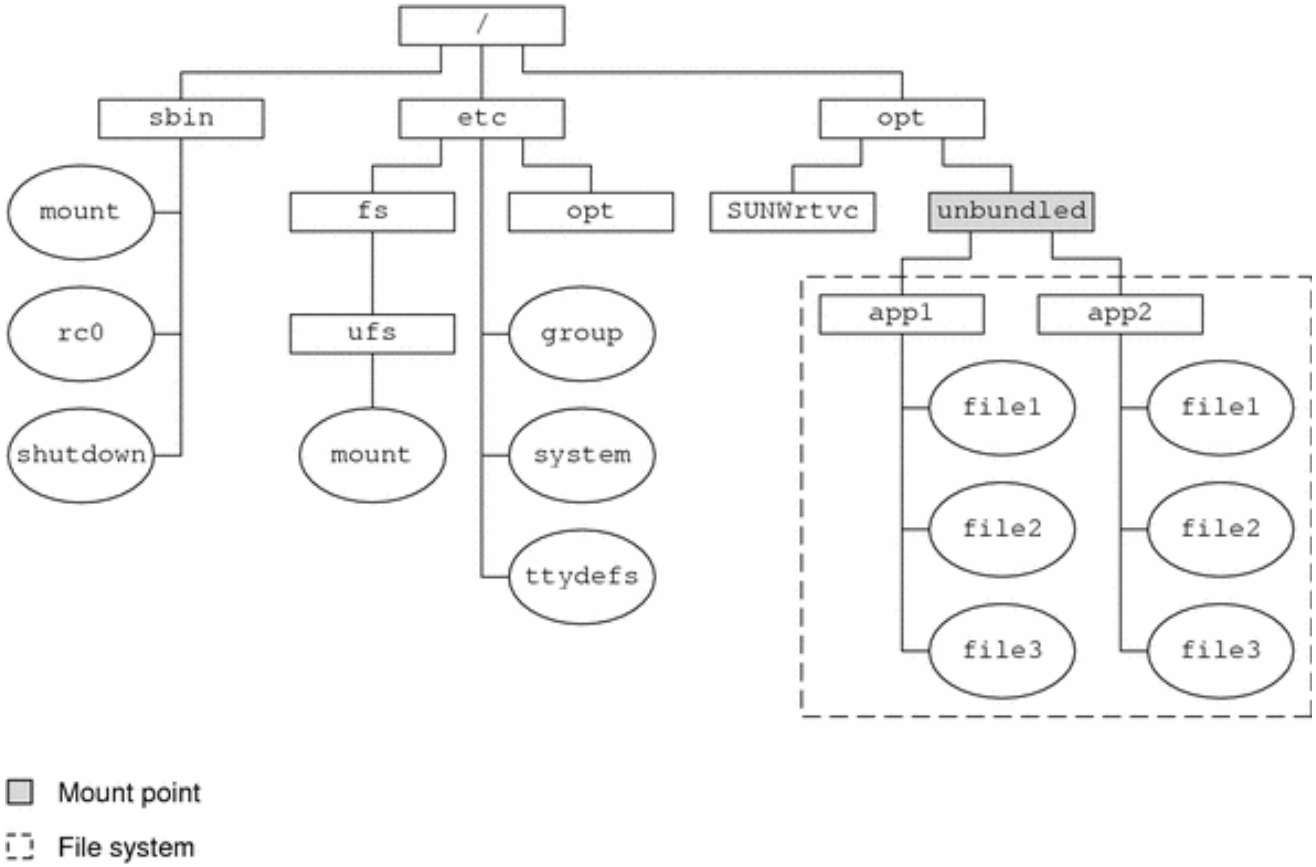
    Master Boot Record (MBR) boot sector + informazioni su partizioni
    - 4 partizioni "primarie" + altre estese

    GUID Partition Table (GPT) partizioni identificate da un UUID (Universally Unique Identifier)
- nei sistemi Unix-like file speciali a caratteri o blocchi corrispondono a dispositivi di I/O, identificati da un major e un minor number
  - per esempio, in Linux `/dev/sda` corrisponde a un intero disco, mentre `/dev/sda1`, `/dev/sda2` e così via le singole partizioni
    - ma anche `/dev/disk/by-partuuid`, etc
  - possono essere creati con `mknod`, da root
  - per essere utilizzati, vanno montati, tramite `mount`

    (per essere montati vanno formattati prima, ne parliamo fra poco)

Mounting/unmounting

In un sistema Unix c'è un solo file system; `mount` "aggancia" l'albero di file e directory di un dispositivo all'albero globale, su un mount-point



Gestione file speciali

- Con l'aumentare dei dispositivi supportati dal kernel e l'arrivo di dispositivi *hot-plug* il sistema mostrava i suoi limiti...
  - *migliaia* di file sotto `/dev`
  - cosa succede se un disco viene (fisicamente) spostato?
    - per esempio, in `/etc/fstab`?
  - come gestire pendrive USB e altri dispositivi hot-plug?
- Le partizioni si possono identificare in modo più stabile tramite UUID; si vedano `blkid(1)` e `fstab(5)`
- Oggi il kernel solleva *eventi* quando vengono (s)collegati dispositivi
- Processi utente, es. `udev(7)`, possono monitorare questi eventi
  - creare/rimuovere file speciali sotto `/dev`, che oggi è un `tmpfs`
  - (s)caricare moduli kernel
  - notificare il file-manager
  - "qualcuno" (udiskd, Nautilus, Thunar, ...) può montare automaticamente il dispositivo appena collegato, tipicamente sotto `/media/`*`username`*`/`*`label`*

File e directory

Naturalmente, gli utenti di un s.o. usano le astrazioni di

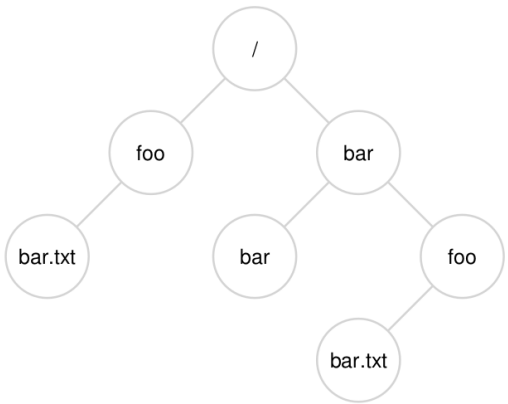
file una sequenza di byte

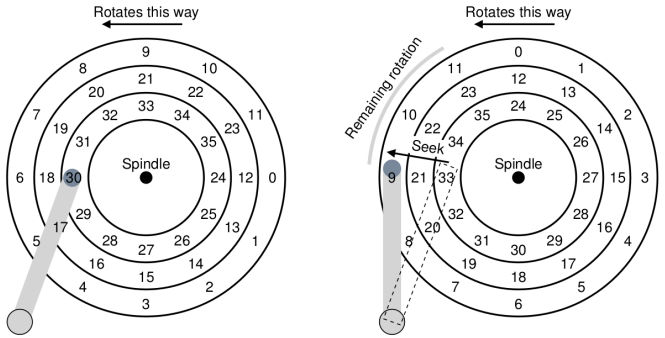
directory contenitori logici di file e directory (ricorsivamente)

a cui sono associati dei nomi; es. `pippo.txt`

Ripasso (e parte che avevamo saltato precedentemente):
`http://pages.cs.wisc.edu/~remzi/OSTEP/file-intro.pdf`

Problema: come implementare le astrazioni di file/directory su un dispositivo a blocchi?





Implementazione

Un file-system è una struttura dati, che risiede su un dispositivo a blocchi

- gestisce dati e metadati
- formattare, rispetto a un certo formato, significa preparare questa struttura dati sul disco
  - Vari formati (FAT, FAT32, NTFS, ext2/3/4, ...); si vedano `mkfs.*`
  - Noi consideriamo una versione semplificata di FS "alla Unix", che il libro chiama *vsfs (Very Simple File-System)*

Non dimentichiamoci che sono necessarie anche altre strutture dati, per gestire l'accesso ai file da parte dei processi utente. Per esempio, quando un processo usa `open`, fra le altre cose il kernel deve:

1. recuperare l'*inode* (struttura su disco) corrispondente al percorso specificato come stringa
2. allocare una struttura che corrisponde al file aperto, che "punta" a (1)
3. allocare un FD nel PCB del processo, che "punta" a (2)

Relazione fra file descriptor e file aperti: POSIX

Ricordatevi (slide vista varie lezioni fa):

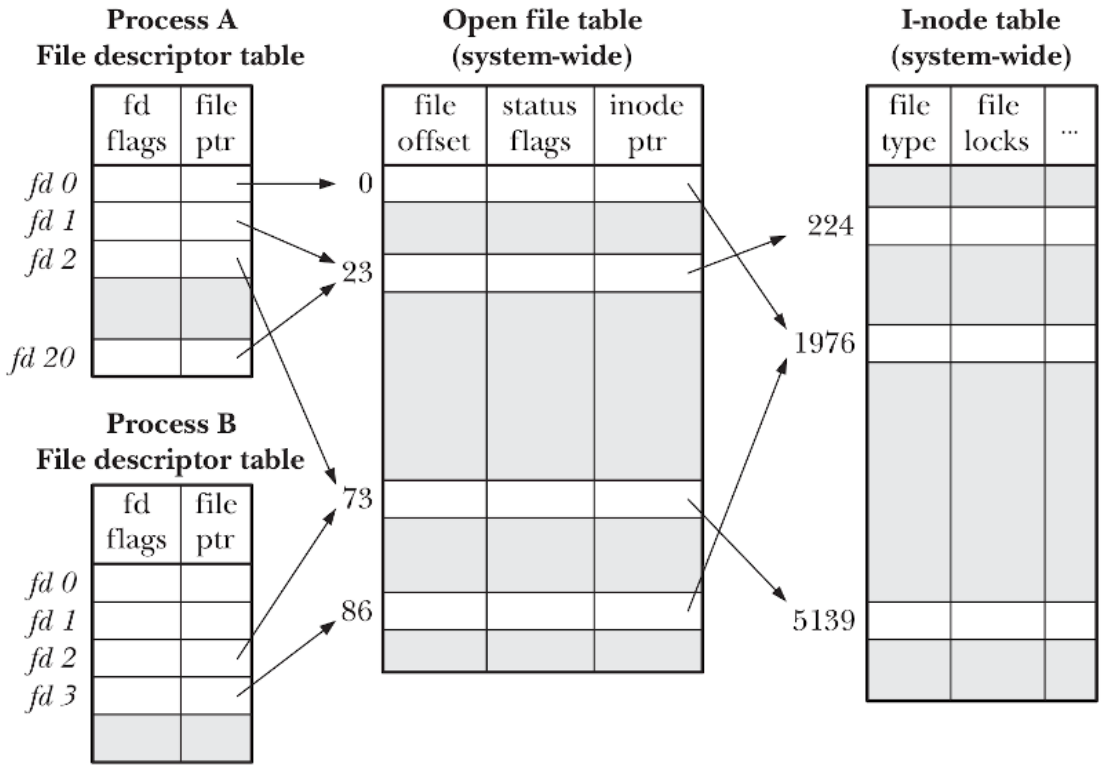


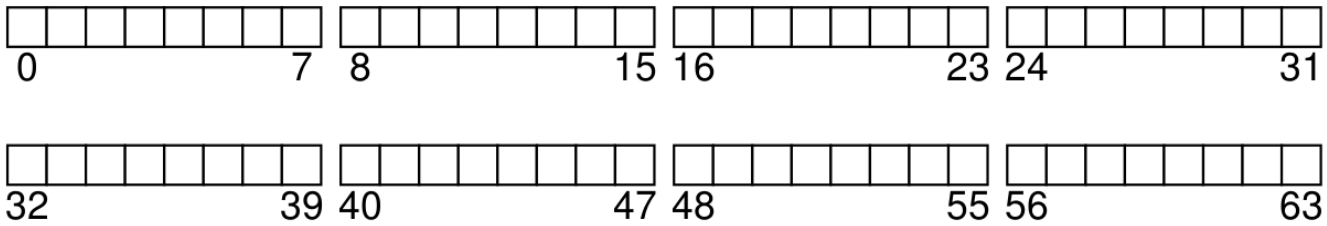
Figure 5-2: Relationship between file descriptors, open file descriptions, and i-nodes

Presa dal libro [Ker10]

Organizzazione

Continuando l'esempio del libro, assumiamo di avere un disco (ridicolo, ma per semplicità. . .), o partizione, di 64 blocchi da 4k

- spesso i settori da 512 byte si raggruppano il blocchi logici, i cluster

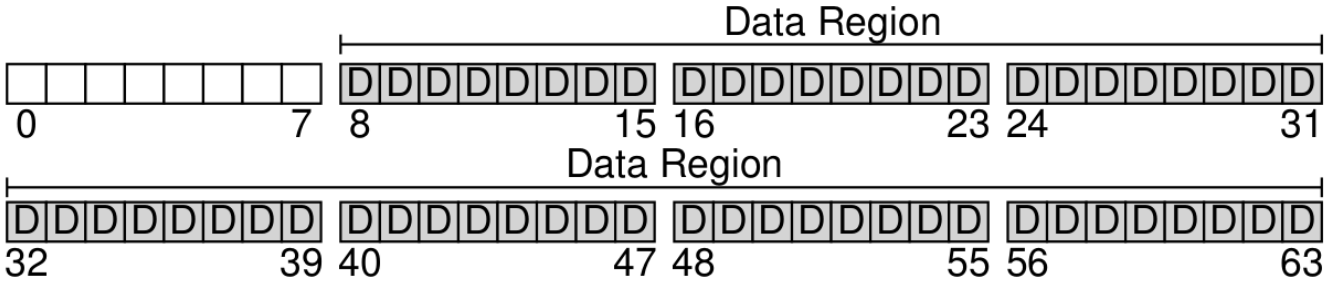


Questa parte di slide è basata su:

`http://pages.cs.wisc.edu/~remzi/OSTEP/file-implementation.pdf`

Dati e metadati (1/2)

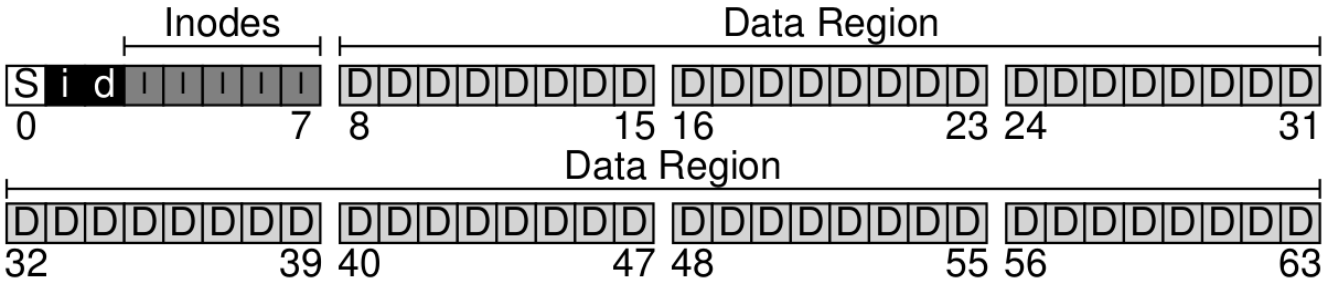
Non possiamo usare *tutto* il disco per i dati...



dobbiamo memorizzare i metadati sui file; in particolare...

Dati e metadati (2/2)

- per ogni file un inode
- gli inode sono contenuti nella tabella degli inode
  - quindi, un numero max di file
- per ogni inode bisogna sapere se usato o meno: inode bitmap
- analogamente, per ogni blocco dati: data bitmap
  - perché delle bitmap? Non possiamo scrivercelo dentro la struttura stessa e/o usare *free-list*?
- infine, un superblocco per identificare il tipo di file-system e le sue caratteristiche (numero di inode, numero di blocchi dati, etc)

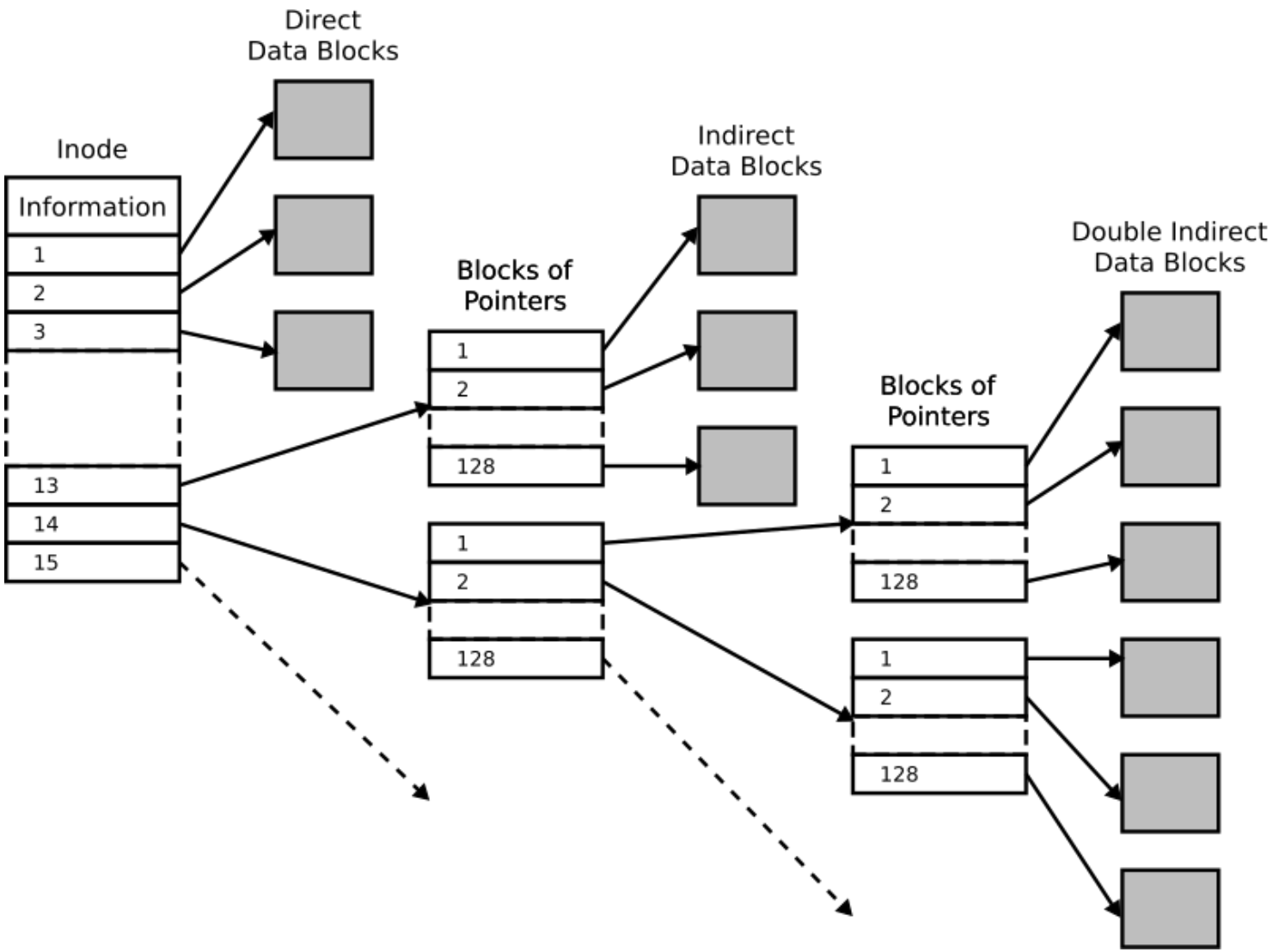


inode

Ogni inode contiene i metadati relativi a un file (ma NON il nome):

- tipo di file, che può essere:
  - regular file
  - directory
  - symbolic link
  - FIFO
  - socket
  - character device
  - block device
- UID/GID di proprietario e gruppo
- dimensione in byte
- maschera dei bit relativi ai permessi
- date di creazione/modifica/...
- numero di (hard) link — ne parliamo fra poco
- "puntatori" ai blocchi dati
- ...

Puntatori ai blocchi dati



https://commons.wikimedia.org/wiki/File:Ext2-inode.svg

Directory, link e cancellazione

- Le directory sono *file*, che contengono associazioni: nome→inode-#
  - in particolare, contengono `.` e `..`
  - queste associazioni sono hard link
- Tramite il comando `ln(1)` potete creare sia
  - (hard) link, cioè *aggiungere un nome* a un file esistente
  - link simbolici/symbolic link — questi sono *file*, il cui contenuto corrisponde a un percorso, *non necessariamente esistente*

  le syscall sono due: `link(2)` e `symlink(2)`
- È possibile creare hard-link solo all'interno dello stesso FS, perché?
- `rm(1)` si appoggia a `unlink(2)`
  - non è detto che `rm` elimini il file, dipende dal numero di hard-link
    - anche se il numero è 0, finché ci sono FD aperti il file corrispondente viene "tenuto in vita"
    - "trucco" spesso usato per i file temporanei

Permessi

Per le directory

- r posso leggere? Ovvero, listare i file contenuti
- w posso modificarne il contenuto?
  Creare/cancellare/rinominare/... nomi contenuti
- x posso "entrare"/accedere?

Quindi, per creare/eliminare nomi da una directory d, dovete avere il permesso di scrittura su d

- es: potete cancellare file read-only di root se la directory è vostra

Risoluzione dei percorsi — `path_resolution(7)`

1. Nel PCB ci sono inode di root r e directory corrente c
   - il percorso inizia con "/"? $d = r$
   - altrimenti, $d = c$
2. per ogni componente (separata da /) non-finale, diciamo n
   i. si hanno i permessi di ricerca in d? (no $\rightarrow$ `EACCESS`)
   ii. si cerca n in d
      - non si trova $\rightarrow$ `ENOENT`
      - altrimenti si recupera inode corrispondente i
   iii. i è una directory? Se sì, si riparte da lì: $d = i$
   iv. i è un link-simbolico? Si risolve a partire da d
      - Il risultato non è una directory? $\rightarrow$ `ENOTDIR`
      - Se lo è, d', si continua da lì: $d = d'$ — un contatore evita loop infiniti
   v. `ENOTDIR`
3. per la componente finale non si pretende che sia una directory
4. `.` e `..` hanno significato speciale
   - Anche se il sottostante FS non li memorizza esplicitamente
   - Ma non si può salire sopra la radice: `/..`$\equiv$`/`

Consistenza del file-system

- Se ci sono varie parti della struttura dati da aggiornare (e.g. blocco dati, bitmap e puntatori ai blocchi), cosa succede se manca la corrente a metà?
  - c'è un ordine migliore di altri?

Fsck

Per controllare l'integrità di un file system (smontato) `fsck(1)`:

- controlla che il superblocco sia "ragionevole"
- consistenza bitmap blocchi liberi e puntatori ai file
  - più inode puntano a uno stesso blocco?
    - fix: si può duplicare il blocco, dando a ognuno la sua copia
  - un blocco puntato risulta libero? fix
  - un blocco non puntato risulta usato? fix
- stato degli inode (e.g. tipo valido)
- link count
  - recupero file senza nome
- puntatori fuori dal range dei blocchi
- directory
  - ognuna ha il suo `.` e `..`?
  - qualcuna è collegata più di una volta nell'albero?
- ...

Journaling

- abbastanza ovviamente, il controllo di integrità è molto lento
  - sempre di più al crescere della dimensione dischi
  - ottimizzazione: un FS smontato "in modo pulito" non viene controllato a ogni mount
    - come si realizza?
- approccio moderno: Journaling (AKA write-ahead logging)
  - rendere atomiche le (sequenze di) operazioni
    - idea: prima di fare le modifiche, vengono segnate in un log; se c'è crash vengono riapplicate al mount
  - Per approfondire potete vedere:
    `http://pages.cs.wisc.edu/~remzi/OSTEP/file-journaling.pdf`
    e capitoli seguenti

Bibliografia

[Ker10] Michael Kerrisk.
The Linux programming interface: a Linux and UNIX system programming handbook.
No Starch Press, 2010.