# Filsystemet i Linux

# Filsystemet “fra innsiden”

- Tidligere sett *brukerkommandoer* for håndtering av filer – filsystemet i Linux sett “fra utsiden”
- Skal nå se på filsystemet “fra innsiden”:
  - Hvordan er filsystemet i Linux bygget opp?
  - Hvordan gjøres *egentlig* lagring av filer?
  - Hvorledes kan vi selv kontrollere filsystemet?

# Lærestoff *

- Oppbyggingen av filsystemet
- Lagring og gjenfinning av filer
- Lagring av metadata for filer: *inoder*
- Toppnivåkatalogene i Linux
- Partisjonering, montering og demontering **
- Styring av lagringskvoter på disk **
- Verktøy for administrasjon og overvåkning **

*: Det meste av lærestoffet finnes i kapittel 10 i læreboken

**: Avsnittene 10.4 og 10.6 i læreboken, foreleses ikke

# Filbegrepet i Linux

- En fil er enten:
  - en kilde hvor data kan leses fra, eller
  - et medium som data kan skrives til
- Betyr at “alt er filer”:
  - Data- og programfiler på disk
  - Alle device (tastur, skjerm, mus, printer, porter..) representeres med spesialfiler
  - Programmer i Linux  kan behandle lesing fra og utskrift til “hva som helst” på en enhetlig måte

# Linux filtyper

1. Regulære filer
2. Kataloger
3. Spesialfiler
   - Block device file – Blokkbasert lagringsenhet, f.eks. disk
   - Character device file – Enhet som leser/skriver ett tegn om gangen, keyboard, skjerm, modem
   - Named pipe file – Buffer for data-utveksling mellom prosesser
   - Symbolic link file – Lenke (soft) til en annen fil på systemet
   - Socket file – For kommunikasjon over nettverk

# Se filtype: `ls -l *`

```
janh@ask:~$ ls -l
total 44
-rw-r--r-- 1 janh janh 39076 Sep 12 11:46 cc.jpg
drwx------ 2 janh janh  4096 Sep 12 12:14 smuss
```

| | |
|---|---|
| - | Regular file |
| d | Directory |
| p | Named pipe (se `man mkfifo` ) |
| b | Block device |
| c | Character device |
| l | Link |
| s | Socket (se f.eks. i `/run` ) |

* Eller `ls -F`

# Datalagring og hardware

- Data som *CPU* bruker ligger i ikke-permanent RAM
- Dataene i *filer* lagres i sekundærminnet, som består av permanente media:
  - Lagrer store datamengder til lav kostnad
  - Ikke direkte tilgjengelig for CPU
  - Aksesseres av CPU gjennom egne I/O-kanaler
  - Data overføres oftest til/fra permanente media i større, sammenhengende *blokker* for å øke hastigheten

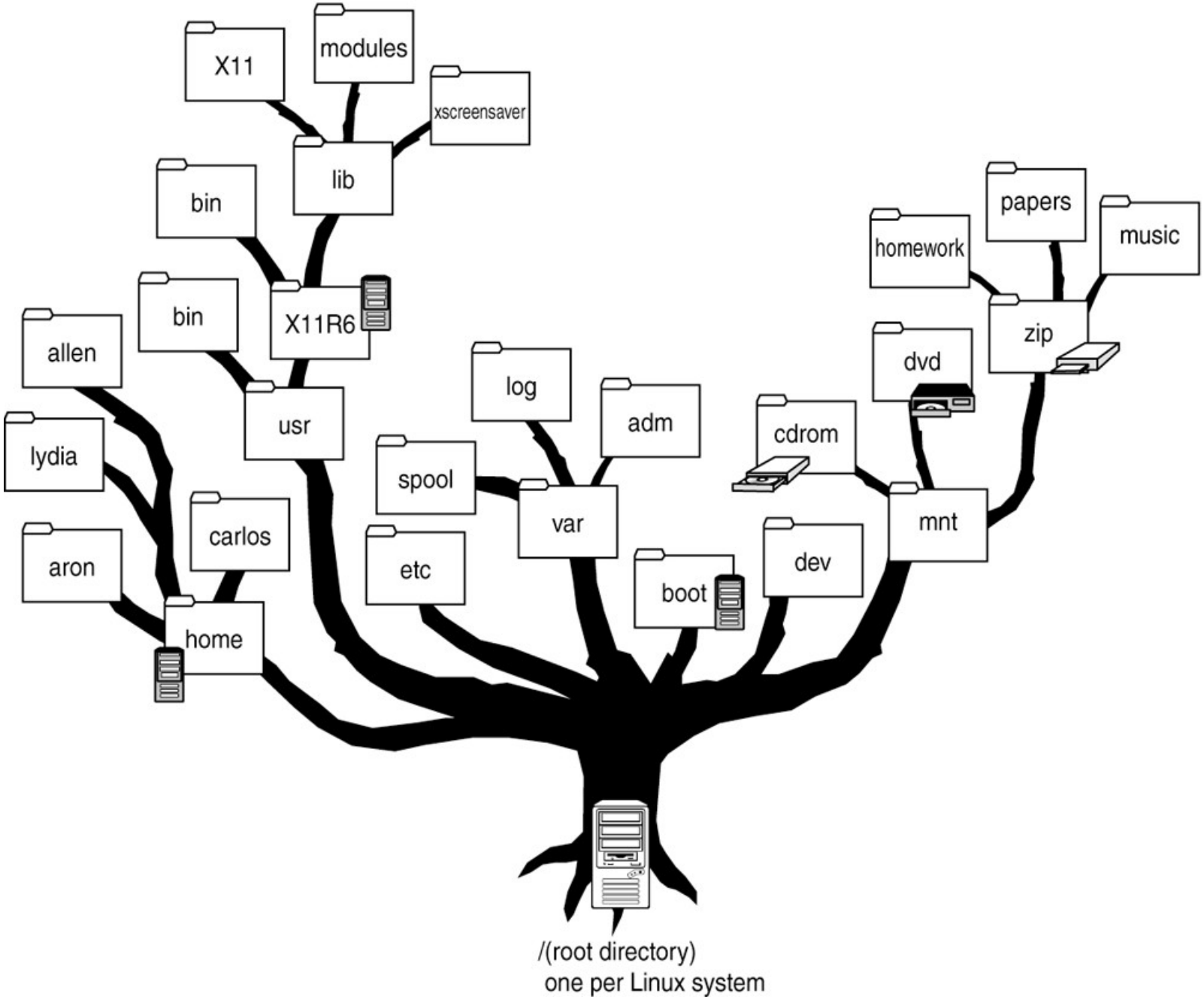
# Vanlig brukte lagringsenheter*

- Permanente media, sortert etter hastighet:
  - RAM-disker (primærminne brukes som “disk”)
  - Solid-state disker (flash-RAM / “minnepinner”)
  - Magnetiske disker
  - Optiske disker (CD, DVD, Blu-ray)
  - Tapestasjoner

*: Se avsnitt 3.7 i læreboken for mer om hardware for datalagring

# Lagringsenheter (devices) i Linux

- Hele det store katalogtreet i et Linux-system kan ligge spredt utover *flere* fysiske lagringsenheter
- Maskinens interne disk er alltid en av disse fysiske lagringsenhetene
- Andre lagringsenheter kan *monteres* inn i katalogtreet på ulike steder (mount points), med egne *lokale* filsystemer
- Lesing og skriving til devices foregår gjennom spesialfiler (drivere) som ligger i katalogen `/dev`


X11
modules
xscreensaver
lib
bin
bin
X11R6
allen
lydia
aron
usr
carlos
home
log
adm
spool
var
etc
boot
dev
cdrom
dvd
homework
papers
music
zip
mnt
/(root directory)
one per Linux system

# Lagringsenheter og blokker

- Fysiske lagringsenheter deles opp i mindre, sammenhengende områder, for raskere I/O
- Disse enhetene kalles en *(disk) blokk*
- Vanlig brukte blokkstørrelser: 1, 4 og 8 KB
- Se blokkstørrelsen for et device i Linux:

```
blockdev --getbsz device-file
```

# Blokker og filer i Linux

- En blokk er den *minste* lagringsplassen som kan settes av i Linux til å lagre en ikke-tom fil
- Når en ikke-tom fil opprettes:
  - Det settes av plass til *minst én blokk* med data, *selv om* antall bytes på filen er mindre enn blokkstørrelse
  - I tillegg lagres filens *metadata* internt i filsystemet
- Linux allokerer automatisk nye blokker til å lagre data i etterhvert som en fil vokser i størrelse

# Kommandoer for å se filstørrelser

- `ls -l`

  L*engden* på fil, *antall bytes* fra start på filen frem til EOF-merket

- `ls -s`

  Antall kilobytes *satt av* til fil

- `ls -s --block-size=4K`

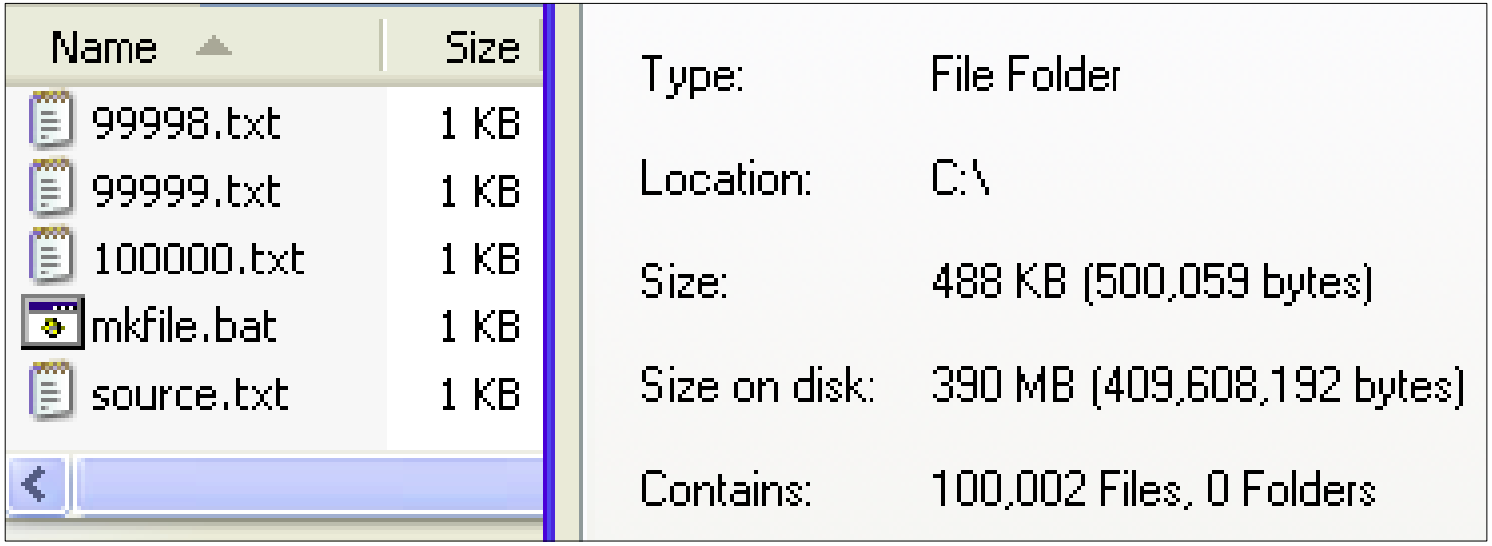
  Antall 4K diskblokker satt av til fil

- `du` *`[opsjoner]`*

  Disk usage: Oppsummering av plassforbruk for filer og kataloger – mange opsjoner og virkemåter

# “Slack space“: Sløsing med diskplass

- Slack space for en fil:
  - Ekstra lagringsplass som brukes fordi filen ikke fyller opp diskblokken(e) som er satt av
  - Gjennomsnittlig slack space for en fil er halvparten av filsystemets blokkstørrelse
- Store diskblokker:
  - Mye slack space, spesielt hvis mange små filer
- Små diskblokker:
  - Mindre slack space, men mye *overhead* i filsystemet hvis det er mange store filer




# Slack space eksempel (Wikipedia):

- 4 KByte = 4096 bytes blokkstørrelse
- 100 000 filer, hver fil er på bare 5 bytes
- 500 000 bytes med faktiske data
- Krever over 409 milloner(!) bytes med diskplass

# Fragmentering av filer

- Disken deles opp i blokker på f.eks. 4 KB
- Filsystemet i OS holder rede på hvilke blokker som er ledige og hvilke som er i bruk
- Fragmentering:
  - Når en fil blir større enn de ledige sammenhengende områdene på disk, må den lagres i områder som er *spredt* rundt på disken – filen blir *fragmentert*

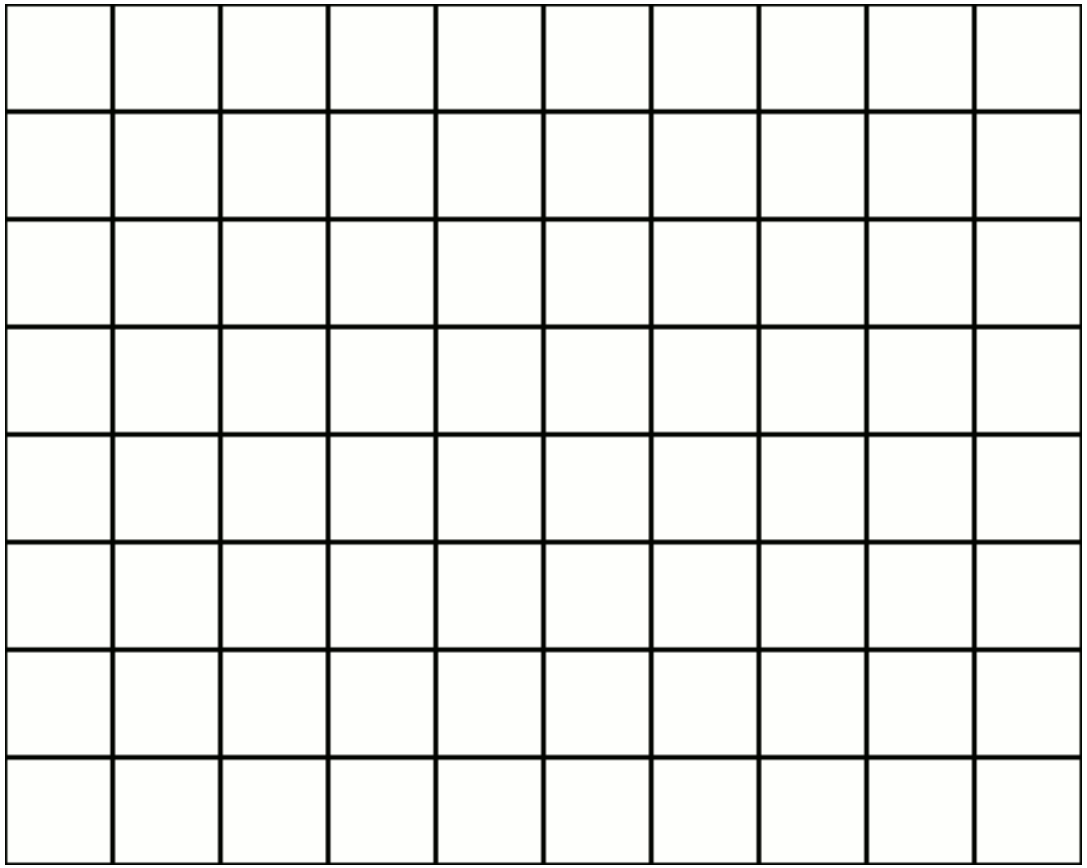
# Fragmentering og diskhastighet

- For hver fil må OS holde rede på:
  - *Hvor* på disken de ulike blokkene som tilsammen utgjør en fil ligger lagret
  - I hvilken *rekkefølge* disse delene av filen skal settes sammen
- Etter lengre tids bruk kan en disk bli *sterkt* fragmentert:
  - Store filer består av svært mange småbiter som ligger tilfeldig spredt på disken
  - Lesing/skriving av sterkt fragmenterte filer er lite effektivt, spesielt for tradisjonelle magnetiske disker

# Håndtering av fragmentering

- Prøv å *unngå* fragmentering:
  - Legg filer som leses mye, men ikke oppdateres ofte, på en egen disk, og f.eks. loggfiler/cache etc. på en annen
  - Prøv å sette av “nok plass” ved opprettelse av filer som “skjøtes på” ofte.
  - Bruk større diskblokker
- Defragmentering:
  - Gå gjennom disken med jevne mellomrom for å gjøre fragmenterte filer sammenhengende igjen
  - Ressurs- og tidkrevende, kan bare gjøres ved lav belastning på systemet

# Visualisering av fragmentering og defragmentering (Wikipedia)

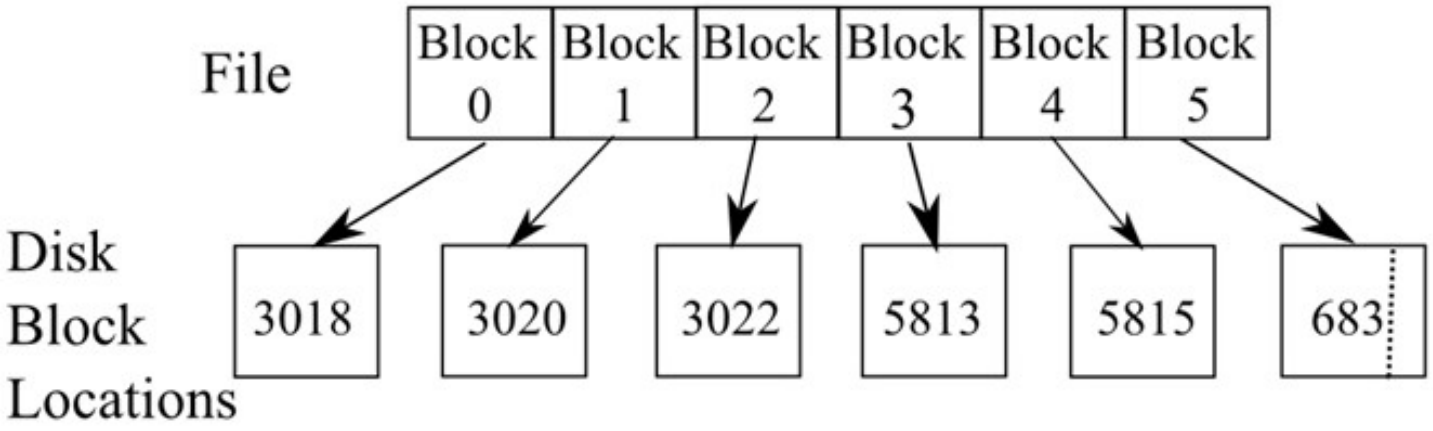


# Lagring og gjenfinning av diskfiler

- Diskfiler stykkes opp i blokker
- Blokkene ligger ikke sammenhengende, men er spredt på ulike steder (diskadresser)
- Filsystemet/OS må ha innebygget en mekanisme for å kunne *lagre* og *finne igjen* en bestemt blokk i en fil
- Det finnes flere ulike måter å gjøre dette på, som varierer mellom forskjellige OS

# Indeksering av filblokkene

- Alle OS bruker en eller annen variant av indeksering:
  - Filen representeres med en “file pointer”, som peker til et sted der informasjon om filen er lagret
  - Den fysiske diskadressen til hver blokk som utgjør filen lagres i en array (eller annen egnet datastruktur) som kan *indekseres* fra starten til slutten av filen.



# File Allocation Table – FAT

- Filsystem brukt i bl.a. eldre Microsoft-OS * :
  - Hver fil har en peker til første blokk
  - Hver blokk på disken lagrer en peker til neste blokk
- Problem:
  - Hvis vi f.eks. skal lese blokk nr. 100 på en fil, må vi gjøre 99 diskaksesser
  - Løses ved at “neste-pekeren” for alle diskblokkene lagres i RAM i en tabell som OS selv vedlikeholder: *File Allocation Table* (FAT)

*: Dagens Windows bruker NTFS der blokk-info. lagres i B-trær

# FAT: Eksempel

File Allocation Table (portion)

| Block | 150 | 151 | 152 | 153 | 154 | 155 | 156 |
|---|---|---|---|---|---|---|---|
| Next location | 381 | 153 | Bad | 156 | 155 | 732 | EOF |

- En fil som starter i blokk 151 slutter i blokk 156
- Blokk 152 på disken er markert som “bad”/uleselig
- Blokk 154 etterfølges av 155 som etterfølges av 732

- For at dette skal virke, må OS i tillegg bl.a. ha en tabell som lagrer fysiske diskadresser for hver blokk

# Filsystemet i Linux

- Linux har støtte for ulike filsystemer, også FAT
- Vanligst: The Extended File System – ext :
  - Opprinnelig fra MINIX (“leke-Unix” for PC, ca 1990)
  - ext er inspirert det gamle Unix File System (UFS)
  - Stor fordel med ext: Journaling
  - ext er i dag betegnelse på en *familie* av filsystemer
  - Mest brukt i Linux er ext 3
- Sentralt element i Linux-filsystemer: inode :
  - En inode lagrer informasjonen om en enkelt fil

# inoder *

- En *inode* i Linux er datastrukturen som lagrer alle *metadataene*, unntatt filnavnet, om *en* fil
- Alle inodene ligger i en tabell med (oftest) *fast lengde*
  - Typisk én inode per 2-8 KB med diskplass
  - Hvis inodetabellen er full er filsystemet fullt
  - Når en fil fjernes, gjenbrukes inoden av neste nye fil
- inode*nummeret* til en fil er indeksen i inodetabellen

* *inode* er antagelig en forkortelse for “index node”

# Kataloger, filnavn og inoder

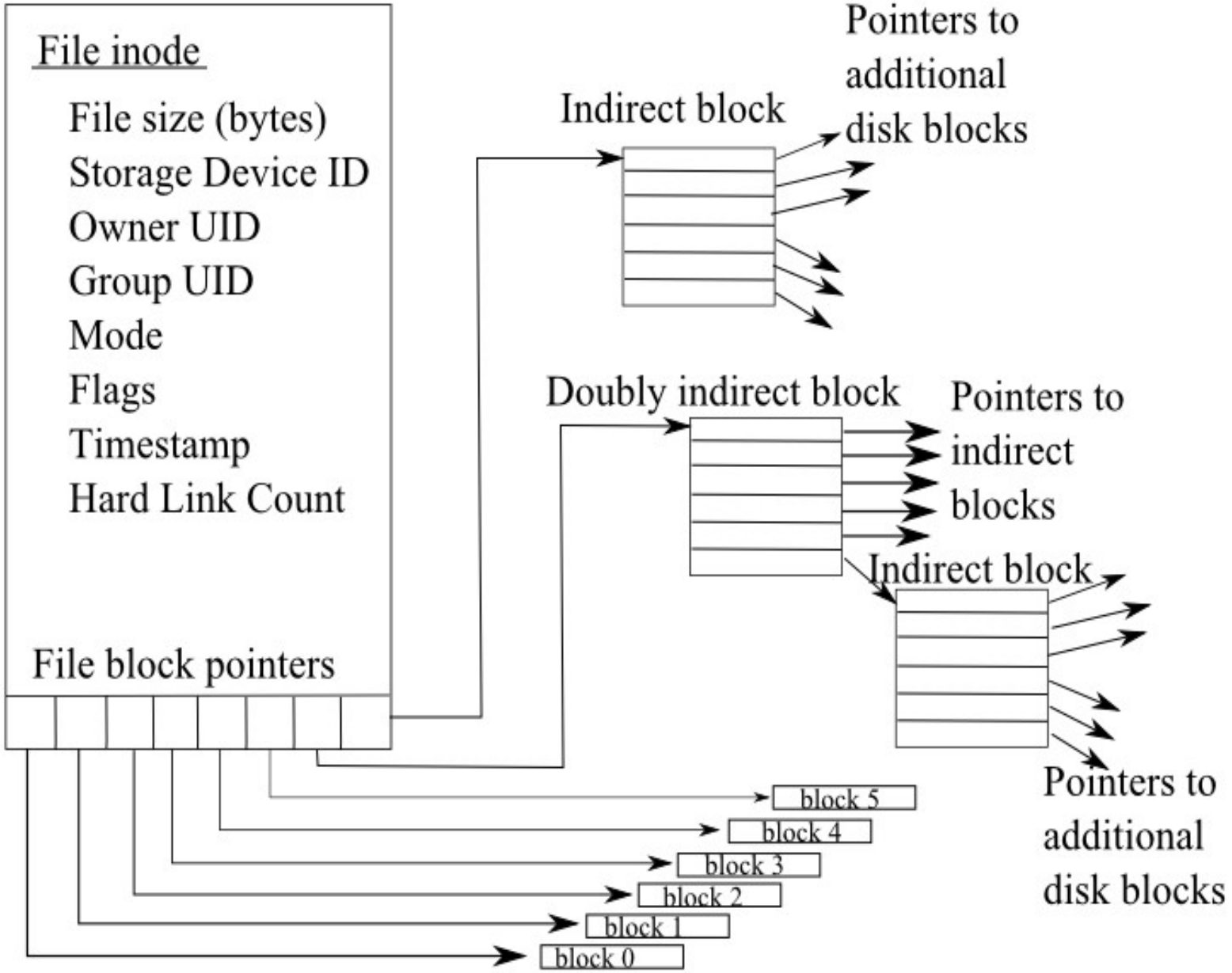
- En katalogfil (directory file) i Linux inneholder bare en liste med *inodenummer* og *filnavn* for hver fil (inkludert katalogfiler) i katalogen
- Linux-kommandoer som håndterer filer:
  - Bruker filnavnet som er angitt til å hente ut inodenummeret fra katalogfilen
  - Henter deretter nødvendig informasjon om filen fra inodetabellen

# Innholdet i en inode, for regulære filer

- Filstørrelse i antall bytes
- Device som filen ligger på
- Eier av filen og filens brukergruppetilhørighet
- File mode / tilgangsinformasjon
- Tidspunkter for siste endring av filen og inoden
- Tidspunkt for siste gang filen ble brukt.
- Antallet link-filer som peker til denne inoden
- Adresser til blokkene der filen fysisk ligger lagret

# Inoder og lagring av diskblokker

- En inode inneholder et lite antall pekere (typisk 12) *direkte* til de første diskblokkene på filen
- Inoden har i tillegg *indirekte* pekere:
  - En peker til en *indirekte blokk* der det er lagret flere direkte pekere til diskblokker
  - Det kan også brukes en peker til en *dobbelt indirekte blokk* (med pekere til indirekte blokker) og en pekere til en *trippel indirekte blokk* (med pekere til dobbelt indirekte blokker)


File inode
File size (bytes)
Storage Device ID
Owner UID
Group UID
Mode
Flags
Timestamp
Hard Link Count
File block pointers
Indirect block
Pointers to additional disk blocks
Doubly indirect block
Pointers to indirect blocks
Indirect block
Pointers to additional disk blocks
block 5
block 4
block 3
block 2
block 1
block 0

# Inoder og diskblokker: Eksempel

- Anta en inode inneholder:
  - 12 direkte pekere
  - 1 indirekte peker
  - 1 dobbelt indirekte peker
  - 1 trippel indirekte peker
- Anta at filsystemet har blokker på 8 KB, og at hver peker er på 8 byte:
  - En blokk kan da lagre 1024 pekere

(eksempel fortsetter) →

# Inoder og diskblokker: Eksempel (forts.)

- Blokk 0 – 11 på filen nås gjennom de 12 direkte pekerne
- Den indirekte pekeren peker til en blokk med med 1024 pekere til de neste blokkene på filen (blokk 12 – 1035)
- Den dobbelt indirekte pekeren peker til 1024 indirekte blokker, hver av disse kan igjen lagre 1024 direkte pekere til diskblokker (1024 * 1024 = 1 MB med direkte pekere)
- Den trippel indirekte pekeren peker til 1024 dobbelt indirekte blokker, totalt en 1 GB med direkte pekere
- Siden hver blokk er på 8KB, blir maksimal størrelse for en fil over 8 TB (!)

# Linux-kommandoer for inoder

- Brukes ikke ofte:
  - OS håndterer inoder og diskblokker “bak kulissene”
  - Sjelden at sys.adm. eller brukere trenger inode-tilgang
- Linux kommandoer for å se inode-informasjon:

| | |
|---|---|
| `ls -i` | Lister inode-nummer for filene |
| `df –i` | `df` viser statusinfo. for filsystem, opsjonen `-i` viser info. om inode-tabell |
| `stat` | Viser status for filer og/eller filsystem |

# `stat` – display file or file system status

```
stat [OPTION]... FILE...
```

- Opsjoner:
  - `–f` Vis status for filsystem i stedet for filer
  - `-L` Følg symbolske lenker
  - `-c "string"`
    Formatér utskriften fra `stat` iht. formateringskodene angitt i *`string`*

# Noen formateringskoder til `stat` for statusinformasjon om filer

| | |
|---|---|
| `%n` | Filnavn |
| `%s` | Størrelse i bytes |
| `%b` | Størrelse i antall blokker |
| `%u` `%U` | UID/brukernavn til eier |
| `%g` `%G` | GID/gruppenavn |
| `%h` | Antall harde lenker til filen |
| `%i` | Inode-nummer |
| `%x` | Tidspunkt for siste aksess |
| `%y` | Tidspunkt for siste endring |

# Et shellprogram som bruker `stat`

- Vil finne filene med minste og største inode-nummer i en mengde med filer
- Input til programmet er filnavnene
- Bruker en `for`-løkke til å gå gjennom alle filene
- Bruker `stat -c "%i%"` for å finne inodenr.
- Kode: `inode_minmax`