

Käyttöjärjestelmät II

Levy I/O Linux ja W2000 levy I/O

Ch 11.5-11 [Stal 05]

Ch 20.8 [DDC 04]

Mitä KJ-I:ssä / KJ-II:ssa?

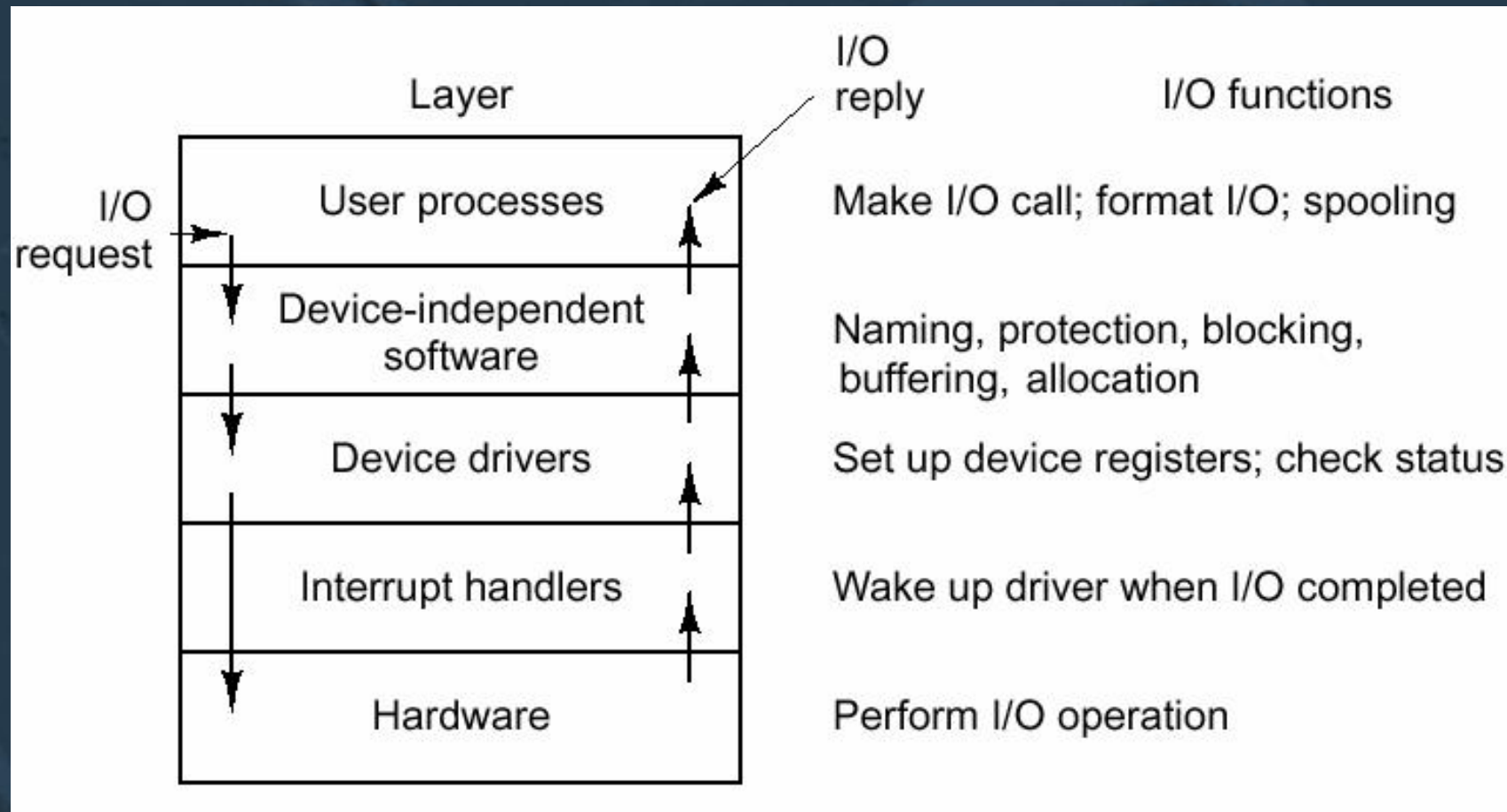
KJ-I

- n I/O-laitteista
- n I/O:n organisointi
- n KJ:n suunnittelusta
- n Puskurointi

Seuraavaksi KJ-II:ssa, tämä luento

- n Levypyöntöjen vuorottaminen
- n RAID
- n Lohkopuskurit (disk cache)
- n Esimerkit: Linux, W2K

Siirränän hierarkia



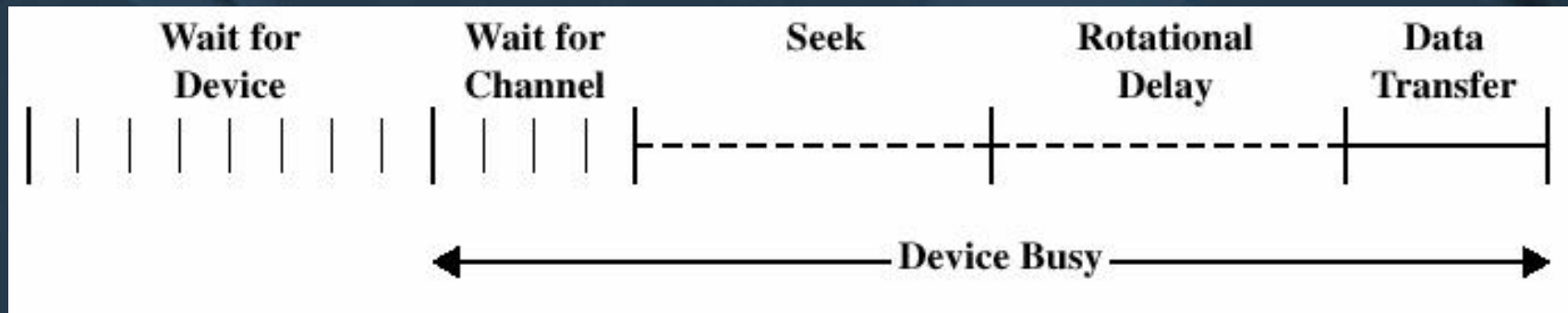
(Fig 5-16 [Tane01])

Käyttöjärjestelmät II

Levytyöntöjen järjestely

Ch 11.5 [Stal 05]

Levyhaku



(Fig 11.6 [Stal05])

- n **Laitteen vapautumisen odotus**
 - u ohjain käsittelee yhden pyynnön kerrallaan
- n **Siirtokanavan odotus**
 - u jos useita levyjä samassa väylässä
- n **Hakuvarren siirto-aika** (seek time)
 - u hakuvarsi oikealle uralle
- n **Pyörähdysviive** (rotational delay/latency)
 - u odota, että oikea sektori pyörähtää kohdalle
- n **Siirtoaika** (transfer time)
 - u yhden lohkon kirjoittamiseen/lukemiseen kuluva aika

saantiaika
(access time)

Algoritmeja

n Hakuvarren siirtoaika pisin

- u kannattaa minimoida siirrot

n Random? FIFO? PRI? LIFO?

- u huonoja, eivät huomioi hakuvarren nykyistä positiota

n Ota huomioon hakuvarren sijainti

- u SSTF
- u SCAN
- u C-SCAN
- u N-step-SCAN ja FSCAN

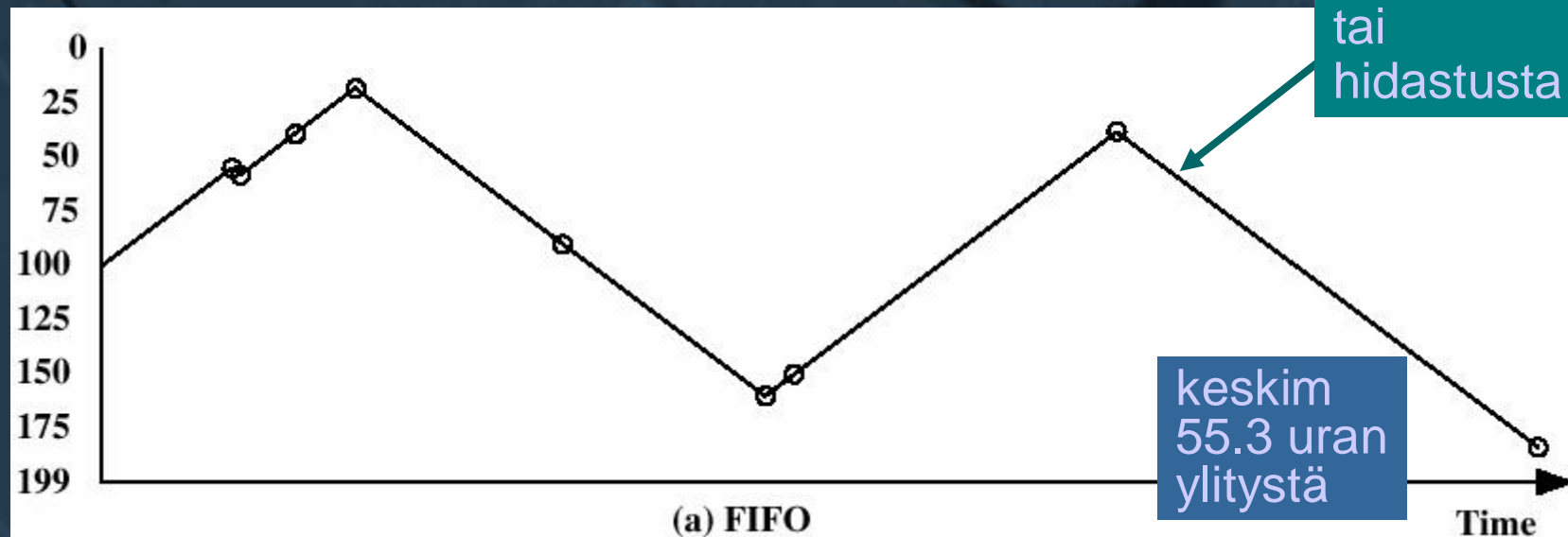
n Esimerkeissä: 200 uraa, hakuvarsi uralla 100,

hae urilta 55, 58, 39, 18, 90, 160, 150, 38, 184

Nyt 100; hae 55, 58, 39, 18, 90, 160, 150, 38, 184

First-in-first-out, FIFO

- n Käsittele saapumisjärjestyksessä
- n Tasapuolinen kaikille prosesseille
- n Prosessin levypyynnöt usein toistensa lähialueilta
 - u FIFO ei ihan yhtä huono kuin Random!
- n Ei hyvä, koska ei ota huomioon levyn tilaa



Nyt 100; hae 55, 58, 39, 18, 90, 160, 150, 38, 184

(Fig 11.7 [Stal 05])

Prioriteetti, PRI

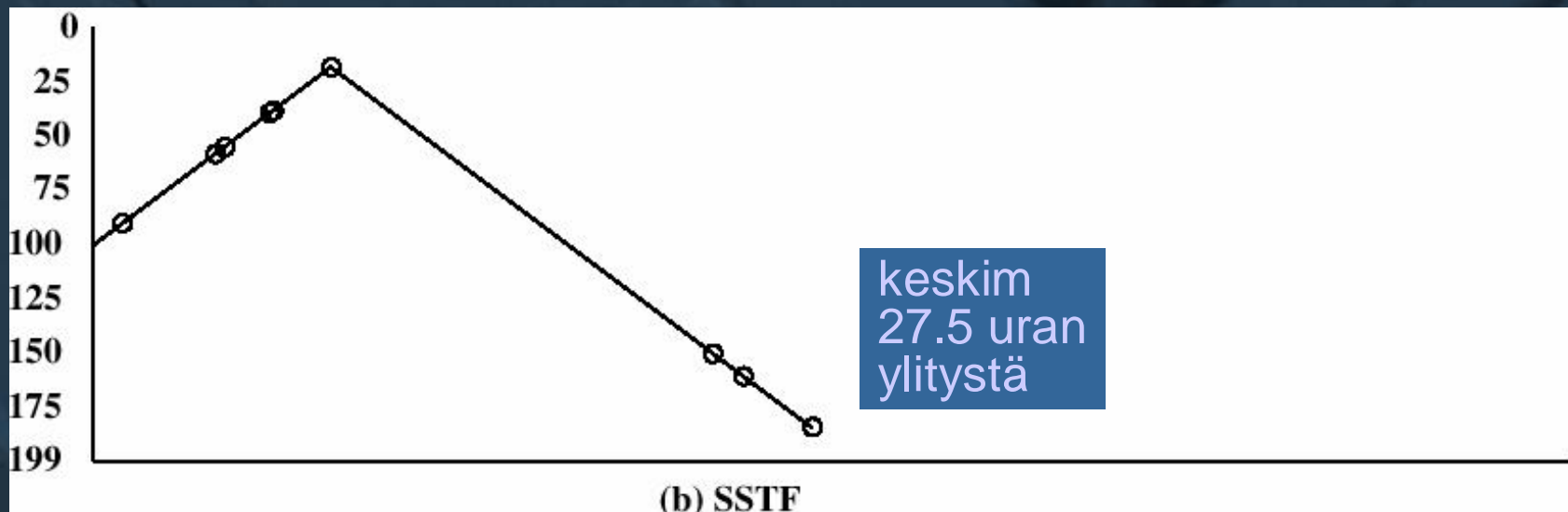
- n Käytä prosessin suoritin prioriteettia myös levyprioriteettina
- n Prosessin prioriteetti määräytyy muiden tekijöiden (suoritin!) kuin levyhakujen perusteella
- n Lyhyillä erätöillä usein suuri (suoritin) prioriteetti
 - u minimoi läpimenoaikaa
- n Interaktiivisilla hyvä (suoritin) prioriteetti
 - u minimoi vastausaikaa
- n Ei hyvä, koska ei ota huomioon levyn tilaa

LIFO, Last-in-first-out

- n **Palvele viimeksi tullut pyyntö ennen muita**
 - u anna laite aina aktiivisimmalle prosessille
 - F idea: vähentää hakuvarren siirtotarvetta, erityisesti peräkkäistiedostoja käsiteltäessä (paikallisuusilmiö)
- n **Nälkiintymisvaara**
 - u jos paljon I/O-sidonnaisia prosesseja, vanha pyyntö voi jäädä jalkoihin
 - F *non-blocking write* – aina läpi
 - F *blocking read* – voi odottaa kauan
- n **Ei hyvä, koska ei ota huomioon levyn tilaa**

SSTF, Shortest Service Time First

- n Palvele pyyntö, jossa selvittää lyhyimmällä hakuvarren siirrolla
- n Esiintyy myös nimellä Shortest Seek First (SSF)
- n Vanhat voi jäädä jalkoihin, kun paljon uusia töitä
 - u I/O sid. työ (korkea CPU prioriteetti)?
 - F lue uralta 25 → käytä CPU:ta → lue uralta 25 → ...

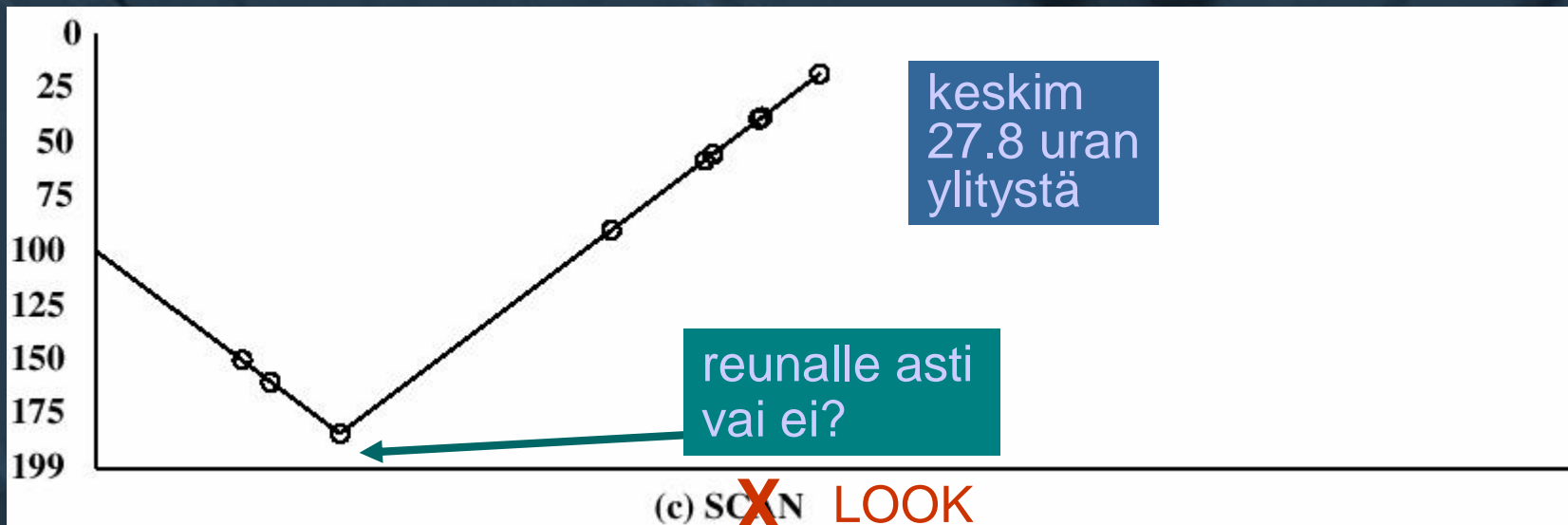


Nyt 100; hae 55, 58, 39, 18, 90, 160, 150, 38, 184

(Fig 11.7 [Stal 05])

SCAN

- n Siirrä hakuvartta samaan suuntaan kunnes reuna vastaan
- n Vaihda sen jälkeen hakuvarren suunta, ja palvele matkan varrelle osuvat pyynnöt
- n Suosii **keskiurille** osuvia pyyntöjä ja uusia töitä
- n Myös nimellä **elevator** (hissi)
- n **LOOK**-versio: käänny takaisin heti, jos edessä ei työtä
 - u ei mennä reunalle asti

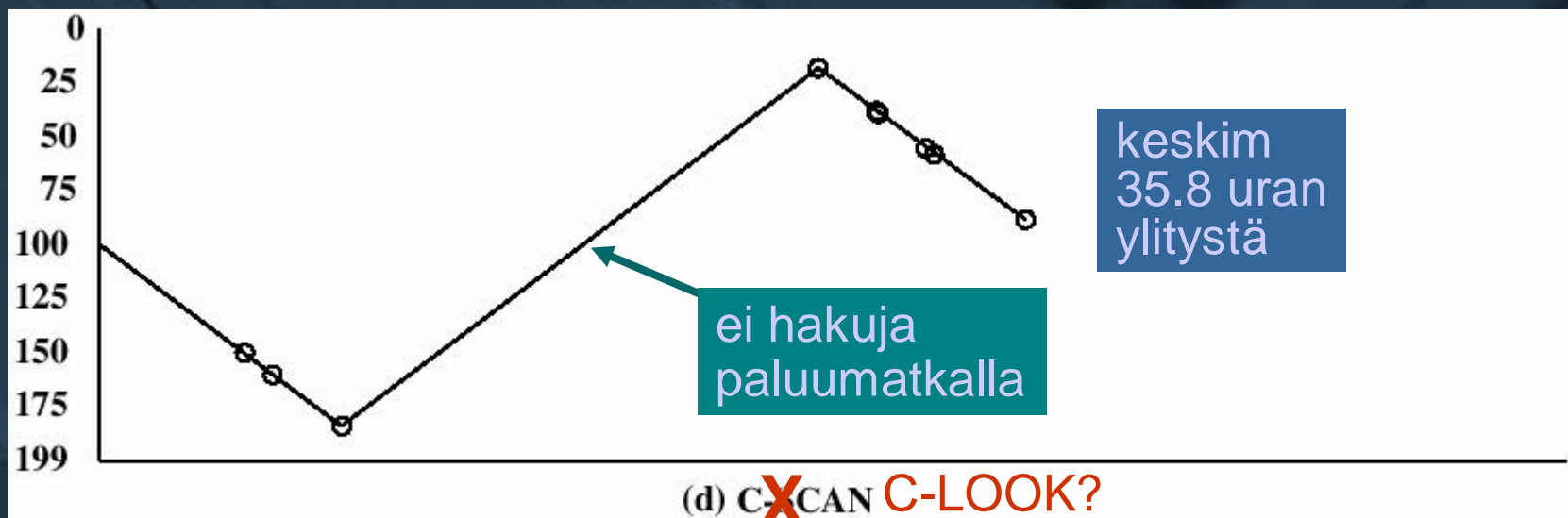


Nyt 100; hae 55, 58, 39, 18, 90, 160, 150, 38, 184

(Fig 11.7 [Stal 05])

Circular-Scan, C-SCAN

- n Siirrä hakuvartta käsittelyn aikana aina samaan suuntaan
- n Kun ko. suunnassa ei ole enää palveltavia, palauta hakuvartsi takaisin ja aloita uudelleen
- n Hukka-aika varren palauttamiseen?
- n Kaikilla urilla sama palvelu
- n C-LOOK versio: ei mennä reunalle asti, jos ei tarvitse



Nyt 100; hae 55, 58, 39, 18, 90, 160, 150, 38, 184

(Fig 11.7 [Stal 05])

FSCAN

- n **Ongelma: paljon uusia töitä samalle uralle**
 - u vanhat työt odottavat ja odottavat...

- n **Ratkaisu: Kaksi jonoa (S ja Q, eli Service ja Queue)**
 - u palvele S-jonoa (SCAN-algoritmillä)
 - F näiden palvelun aikana kaikki uudet työt jäävät jonoon
 - u kerää Q-jonoon sillä aikaa tulevat pyynnöt
 - u kerralla palvellaan kaikki edellisen S-jonon käsittelyn aikana tulleet työt

- n **Vasteajan varianssi pienempi kuin SCAN'illa**

N-step-SCAN

- n **Ongelma: FSCAN:issa paljon uusia töitä samalle uralle**
 - u vanhat työt odottavat ja odottavat...
 - u S-jono hyvin pitkä, jolloin sen alussa olevat työt voivat joutua odottamaan paljon, kunnes oma vuoro tulee
- n **Ratkaisu: kaksi jonoa (S ja Q, Service ja Queue)**
 - u palvele S-jonoa (SCAN algoritmilla)
 - F näiden palvelun aikana kaikki uudet työt jäävät jonoon
 - u kerää Q-jonon loppuun sillä aikaa tulevat pyynnöt
 - u ota uuteen S-jonoon **n ensimmäistä jonottajaa** Q-jonosta
 - F tai kaikki, jos vähemmän kuin n
 - u kerralla palvelee korkeintaan n työtä
 - F $n = 1 \rightarrow$ FIFO
 - F $n = \infty \rightarrow$ SCAN
 - u miksi nimi **N-step**-SCAN?

Algoritmien vertailu

Tbl. 11.2 [Stal 05]

n Hyvyyden mitta?

u läpikäytyjen urien lkm?

n Parempi hyvyyden mitta

u hakuvarren siirtoaika yhteensä?

F ei ole lineaarinen ylitettävien urien suhteen

u kokonaisaika? pyörähdysviive? vasteaika?

n Yhteenveto

Seek Time	80GB	400GB
Average (1/3)	12	4.2
Track-to-track	2.5	2.0
Full stroke	23	21 ms

Tbl. 11.3 [Stal 05]

Pyörähdysviive mukaan - realismia

(ks. [DDC 01])

n SLTF – Shortest Latency Time First

- u minimoi pyörähdysviive
- u oma jono joka sektorille (uralle), ”sector queueing”

n SPTF – Shortest Positioning Time First

- u minimoi hakuarren siirtoaika + pyörähdysviive
- u etusija keskiurille, sisä- ja ulkoreunoja sorsitaan

n SATF - Shortest Access Time First

- u minimoi hakuarren siirtoaika + pyörähdysviive + tiedonsiirtoaika (access time = tiedon sijainnin etsintä + siirtoaika)
- u etusija keskiurille, sisä- ja ulkoreunoja sorsitaan
- u etusija pienillä töillä

Rot. delay	80GB	400GB
Rot. speed	4800	7200 rpm
Aver. rot. delay	6.25	4.2 ms

Loogiset sektorit

- n **Peräkkäiset sektorinumerot eivät välttämättä tarkoita, että sektorit olisivat fyysisesti peräkkäin**
 - u ylimääräiset sektorit virheenkorjausta ja –havaitsemista varten
 - u viallinen sektori 123? Otetaan tilalle varasektori 9876, jonka looginen osoite on silti 123
 - u eivät näy ulospäin lainkaan, levyohjaimen sisäistä tietoa
 - F laiteajuri ei voi ottaa huomioon!
- n **Laiteajurin (kernelin) tekemä optimointi voi perustua väärään tietoon**
 - u perustuu oletukseen, että looginen osoite olisi sama kuin fyysinen levyosoite
- n **Joissakin levyissä voidaan kysellä todellista sektorien sijaintia**

Käyttöjärjestelmät II

RAID

Redundant Array of Independent Disks

Ch 11.6 [Stal 05]

RAID tavoitteet

n RAID (Redundant Array of Inexpensive/Independent Disks)

- u aluksi: ison levyn sijasta monta pienempää edullista levyä
- u nyt: monta kallista erillistä levyä tuo luotettavuutta

n Samanaikaisuus/rinnakkaisuus

- u tiedostot jaettu useammalle levyille
- u lohkon tavut jaettu useammalle levyille
- u hakuvarsien siirrot voivat tapahtua yhtäaikaan

n Vikasietoisuus

- u vaikka levy hajoaa, sillä ollut tieto voidaan toivottavasti luoda uudelleen muiden levyjen perusteella
 - F pariteetti, Mirror, Hamming, XOR ECC
- u ei RAID 0:ssa (onko siis edes RAID?)
 - F “luku nolla tarkoittaa ei mitään”
 - F RAID 0 antaa suorituskykyä, ei redundanssia

RAID tavoitteet

- n **Koko levyjoukko näkyy KJ:lle yhtenä isona levynä**
- n **Tavallisen levyohjaimen sijasta levyä ohjaa 'älykkäämpi' RAID-ohjain**
 - u ei tarvita muutoksia KJ:ään
 - u huolehtii pariteetista
 - u jos levy hajoaa, korjaa 'lennosta' bittijonoa, käyttäjä ei huomaa
 - F levy irti, uusi tilalle, eheyttä se
- n **Myös ohjelmallisia toteutuksia (SW RAID)**
 - u yksi vikaantuva komponentti (RAID-ohjain) vähemmän
 - u RAID-ohjain KJ:n ajurina
 - u redundanssisuuslaskenta CPU:lla erillissuorittimen (levyohjain) asemesta
 - u sama funktionaalisuus kuin HW RAID-ohjaimella

RAID 0 (data hajautettu usealle levyille)

- n **Monta levyä, fiksusti organisoituna**
 - u usean levyn optimointiratkaisu
- n **Ei redundanssia**
- n **Data hajautettu usealle levyille**
- n **Etu: nopeus, samanaikaiset haut eri levyiltä**
- n **Stripe raita**
 - u looginen levy jaettu saman kokoisiin **strip**-lohkoihin (levylohkon monikerta)
 - u peräkkäiset strip-lohkot eri levyille
 - F yhden tiedoston peräkkäiset sripit eri levyillä
 - u samassa kohtaa eri levyillä olevat strip-lohkot muodostavat stripe raidan
 - u tasapainottaa levykuormaa automaattisesti

Fig 11.10 [Stal 01]

Fig 11.8a [Stal 05]

RAID 1 (mirror)

Fig 11.8b [Stal 05]

- n **Levyt tuplattu redundanssin vuoksi**
 - u mirror-levy
 - u alkuaan tavalliset levylohkot, ei stripe'ja
- n **Stripe-lohkot suorituskyvyn lisäämiseksi**
 - u oikeastaan "RAID 0-1" tällöin (kun stripe-lohkot mukana)
- n **Suorituskykyetu**
 - u luku kummalta tahansa levytä (levyjoukolta, stripe'lta)
 - F kummalla lyhyempi saantiaika?
 - u kirjoitus molempiin (samanaikaisesti)
 - u samanaikaisuutta eri levyviitteissä
- n **Levy rikkoutuu?**
 - u vaihda levy, päivitä tiedot "mirror"-levyiltä
- n **Tilakustannus?**
 - u 50% levytilasta (iso!)
- n **Usein käytössä, normaali optio levyohjaimessa ("mirror disk")**

RAID 2 (Hamming)

Fig 11.8c [Stal 05]

- n **Kaikki levyt aina käytössä, strip-lohkon koko 1 bitti**
- n **Hamming koodi**
 - u 4 data bittiä -> 3 pariteettibittiä, 4 levyä datalle ja 3 pariteeteille
- n **Ei suorituskykyetua – päinvastoin?**
 - u luku aina kaikilta levyiltä yhtä aikaa, ja sitten tarkistus
 - u kirjoitus kaikille levyille
 - u ei samanaikaisuutta eri levyviitteissä
- n **Redundanssi**
 - u korjaa lennossa 1 bitin virheet, havaitsee 2 bitin virheet
- n **Levy rikkoontuu?**
 - u vaihda levy, laske bitit Hammingin avulla, alusta levy
- n **Tilakustannus?**
 - u Hamming-koodin mukaan
 - u 7 levyä: 3 pariteettilevyä eli 42%
 - u 15 levyä: 4 pariteettilevyä eli 27%
 - u **liikaa.**

RAID 3 (pariteettibitti)

Fig 11.8d [Stal 05]

- n **Pelkkä pariteettibitti, strip-lohkon koko 1 bitti**
- n **Suorituskykyetu**
 - u luku ja kirjoitus aina kaikista levyistä (mutta rinnakkain)
 - u jonotusaika ei vähene, mutta siirtonopeus kasvaa
 - u ei samanaikaisuutta eri levyviitteissä
- n **Redundanssi**
 - u havaitsee 1 bitin virheet
 - u korjaa lennossa
- n **Levy rikkoontuu?**
 - u vaihda levy, laske bitit pariteetin avulla, alusta levy
- n **Tilakustannus?**
 - u aina 1 levy
- n **Ei paljon käytössä, koska huono suorituskyky**

RAID 4 (pariteettilohko)

Fig 11.8e [Stal 05]

- n **Pariteettilohko, strip-lohkon koko iso**
- n **Suorituskykyetu**
 - u luku rinnakkain eri levyistä, levyt toimivat itsenäisesti
 - u kirjoituksessa yhteen lohkon muut saman stripe'n lohkot pitää ensin lukea pariteettilohkonlaskemista varten
 - F tai sitten lue ensin vanha lohko ja laske siitä
 - u kirjoituksessa pitää **aina kirjoittaa myös pariteettilohko**
 - u jonotusaika vähenee, siirtonopeus kasvaa
 - u samanaikaisuutta eri levyviitteissä
- n **Redundanssi, jos levy rikkoontuu**
 - u korjaa lennossa 1 bitin virheet
 - u vaihda levy, laske lohkot pariteetin avulla, alusta levy
- n **Tilakustannus?**
 - u aina 1 levy
- n **Ei paljon käytössä, koska pariteettilevy on pullonkaula**

RAID 5 (hajautettu pariteettilohko)

Fig 11.8f [Stal 05]

- n Kuten RAID 4, mutta pariteettilohko vuorotellen eri levyille
 - u kirjoittamisen yhteydessä ei enää pariteettilevyn pullonkaulaa
- n Yleisesti käytössä oleva

RAID 6 ja muut

n Ei niin standardoitu kuin RAID 0, ... RAID 5

n RAID 6 (2 hajautettua pariteettiblokkia)

- u kuten RAID 5, mutta tarkistuslohkot laskettu kahdella eri menetelmällä

Fig 11.6g [Stal 05]

- u redundanssi

 - F korjaa lennossa 2 bitin virheet

- u 1 tai 2 levyä rikkoontuu?

 - F vaihda levyt, laske lohkot eri menetelmillä, alusta levyt

- u tilakustannus?

 - F aina kaksi tarkistuslohkolevyä

Tbl 11.4 [Stal 05]

n On muitakin...

- u RAID 0+1, RAID 0+3, RAID 0+5, RAID 1+5, RAID 10, RAID 50, RAID 51, RAID 53, ...

- u RAID 7, Storage Computer'in patentti

 - F nopeampi kuin RAID 3 tai RAID 5, luotettava, kallis

 - F yksi pariteettilevy, levyohjaimessa suuri cache

RAID ja TKTL (K2005)

- n 2 kpl 7 (+ 1 hot spare) levyn RAID 5 serveriä, á 2 TB
 - u /group, /fs (mm. verkkopalvelua) RAID kust. 14% (1/7)
- n 8 kpl 5 levyn (á 146 GB) RAID 5 servereitä, á 550 GB
 - u /fs-0, /fs-1, /fs-2, /fs-3 (tavalliset tiedostopalvelimet)
 - u **backup** varmuuskopiota varten RAID kust: 20% (1/5)
 - F kopioi /fs-i tänne ja stream'aa 2 nauharobotille
 - F SW RAID 5 Linuxin ytimessä, koska nauharobotit eivät sopineet yhteen RAID-ohjaimen kanssa
 - u **db.cs.helsinki.fi** (tietokantapalvelin)
 - u **winserver.cs.helsinki.fi** (Windows 2003 Terminal Server)
 - u **courier.cs.helsinki.fi** (postipalvelin)
- n 2 krt 2 levyn RAID-1 nimipalvelin (Hydra) RAID kust: 50% (1/2)
reund. kust 50%
- n 4 levyn RAID 5 serveri, á 0.9 TB (Bioinformatiikka)
- n 4 levyn RAID 5 serveri, á 0.45 TB (CoSCO) RAID kust: 25% (1/4)
- n 2 kpl 5 levyn RAID 5 serveriä, á 0.6 TB (Vera, Chuck)

Käyttöjärjestelmät II

Lohkopuskurit

Ch 11.7 [Stal 05]

Lohkopuskurit, levypuskurit

- n Buffer cache, block cache, disk cache
- n KJ:n data-alueella oleva puskurin muistiinluettuja levylohkoja varten
 - u jos viitattu lohko muistissa, ei levynoutoa
 - u jokainen I/O-pyyntö ei aiheuta levyliikennettä
 - F useimmat eivät
- n **Tasaa erot kerralla käsiteltävän yksikön koossa**
 - u ohjelma lukee/kirjoittaa tavuja
 - u levyohjain lukee/kirjoittaa lohkoja

Ranskan kielen sana "*cache*" tarkoittaa piilottamista

Lohkopuskurit

n Paikallisuusperiaate pätee myös levyä käytettäessä

- u tiedostoa käydään läpi yleensä peräkkäisjärjestyksessä
- u seuraava viite todennäköisesti samaan lohkoon

n Ennaltanouto

- u kun tiedosto avataan, hae ens. lohko heti lohkopuskuriin
- u seuraavan nouto, heti kun edellistä käsitellään
- u (usean) seuraavan nouto samalla kertaa

n Viivästetty kirjoitus

- u talleta ensin levypuskuriin
- u kirjoita vasta täysi puskuri levyille ...
- u ... tai kirjoita muuttuneet esim. 30 sek välein levyille

vrt. virt.muistin
cleanup

Lohkopuskurin poistoalgoritmi

- n **Tilaa varattu rajallisesti**
 - u UNIX: esim 100 ... 1000 puskuria
 - u Linux: koko vapaana oleva muisti (usein 50% muistista)
- n **Kun ei enää tilaa uusille lohkoille, joku lohko poistettava puskurista**
- n **Samat ongelmat kaikessa puskuroinnissa**
 - u TLB: mikä alkio korvataan?
 - u välimuisti: mikä muistilohko korvataan?
 - u levypuskuri: mikä levylohko korvataan?
 - u virtuaalimuisti: mikä sivutila / segmentti korvataan?
- n **Jos poistettava lohko on muuttunut, se täytyy kirjoittaa takaisin levyille**

LRU: Least Recently Used

- n **Poista lohko, johon viittaamisesta kulunut kauimmin aikaa**
- n **Loogisesti ajatellen:**
 - u levyvälimuisti on jono lohkoja
 - u viimeksi viitattu lohko jonon viimeisenä
 - F päivitys joka viittauskerralla?
 - u poista jonon ensimmäinen lohko
- n **Toteutus: jonossa osoittimia lohkoihin**
 - u lohkoja ei tarvitse järjestellä
 - u tietyn lohkon etsinnän tehokkuus?
- n **Poistettava voi silti olla tarpeellinen - sitä käyttävä prosessi sattui olemaan Blocked-tilassa**
 - u seuraus: pitää lukea levytä pian uudelleen

LFU: Least Frequently Used

- n **Poista lohko, johon vähiten viittauksia**
- n **Tarvitaan lohko kohtainen viitelaskuri**
 - u kasvata aina, kun lohkoon viitataan
 - u laskurin nollaus aika-ajoin
- n **Lohko, jonka viitelaskuri on suuri, saattaa silti olla tarpeeton**
 - u prosessin vaihe, jossa lohkoa tarvittiin, on jo mennyt

Most Recently Used – MRU FIFO

n Yrittää huomioida molemmat edelliset ideat (LRU + LFU)

n **Most Recently Used – MRU FIFO**

Fig 11.9 (a) [Stal 05]

- u jos uusi viite äskettäin viitattuun lohkoon (jonon loppuosassa)
 - F lohko jonon perään
 - F älä kasvata viitelaskuria
 - F useat peräkkäiset viitteet kasvattavat laskuria vain kerran
- u jos viite kauan sitten viitattuun lohkoon (jonon alkuosassa)
 - F lohko jonon perään
 - F kasvata viitelaskuria
- u poista loppuosasta (old section) lohko, jonka viitelaskuri pienin
 - F jos useilla sama arvo, poista takimmaisempi (LRU)

Most Recently Used – MRU Three Sections

n Most Recently Used – MRU FIFO

Fig 11.9 (a) [Stal 05]

- u ajatus hyvä 10, mutta vain vähän parannusta
- u kun lohkoon viitataan se säilyy alkuosassa
- u kun viittaukset loppuvat, lohko putoaa loppuosaan
- u viimeksi loppuosaan joutuneella lohkolla pieni viitelaskurin arvo
 - F vaikka käytetty viimeksi, perällä olevista joutuu helpoimmin poiston kohteeksi, ellei siihen viitata pian uudestaan

n Most Recently Used – MRU Three Sections

Fig 11.9 (b) [Stal 05]

- u parannus: jaa jono kolmeen osaan
 - F poistot aina viimeisestä osasta
 - F etuosasta pudonneelle jää aikaa vanheta
- u tulos: parempi algoritmi kuin LRU tai LFU

Käyttöjärjestelmät II

Linux siirräntä (kernel 2.6)

Deitel Ch 20.8 [DDC 04]
(verkossa aikataulusivulla)

Linux Device Drivers

n **Loadable kernel modules**

- u more than 50% kernel space?

n **Devices in device special files in /dev**

- u major id number (= device type)

- F determines driver

- F device drivers in /proc/devices

- u minor id number (device)

- F separates individual devices in same class

- u read/write/seek/ioctl to file invokes driver

- u device operations look like file ops

n **Hot swappable devices (“plug-and-play” in Linux-land)**

- u on enumerable bus positions (e.g., USB)

- u poll each position every now and then, find new device, identify it, and load kernel module for it

Linux Disk Scheduling

- n Many algorithms provided
- n Default algorithm: elevator variation
 - u sort requests by track
 - u try to merge with existing requests
 - u LSTF – Least-Seek-Time-First

Linux Elevator Starvation Avoidance

- n **Problem:** no guarantee of fast service, request can starve
 - u busy writer can block lazy reader
- n **Solution #1:** deadline scheduler **deadline vuorotus**
 - u each request has deadline (*read* 500 ms, *write* 5 s)
 - u deadlines **expire**, expired requests will be serviced first
 - u *read/write* FIFO queues to find expired requests quickly
 - u group expired *reads* (and *writes*) together to minimize seeks
- n **Solution #2:** anticipatory scheduler **ennakoiva vuorotus**
 - u synchronous (successive) *reads* happen usually once per timeslice
 - u after each *read* wait for 6 ms (aver seek latency) for another *read* to arrive
 - F if it arrives, service it first and avoid one seek
 - u advantageous only if new reads more than 50% of the time
 - F collect history data to decide if this method is used
 - u performance gain 5x-100x

Linux I/O Interrupts

- n **Each device has registered interrupt handler**
 - u interrupt handlers do not have context
 - u interrupt handlers are not tasks (processes)
 - F they can not be suspended or preempted
 - F they can not cause exceptions
 - u want to minimize time in interrupt handler
- n **top half** yläpuolisko
 - u the real interrupt handler, fast, not a task, no context
 - u schedules bottom half
- n **bottom half** alapuolisko
 - u **software interrupt handler**, has context (e.g., device driver)
 - F **softirqs** are suitable for SMP's, many concurrently
 - F **tasklets** are suitable for mutex situations, one at a time
 - u scheduled immediately after top half with high priority
 - F if too many, all done one a time with low priority

Linux Page Cache (for block devices)

- n Same cache for VM pages and for memory mapped files
- n If data is not in cache, place a request to a device request list
 - u kernel sorts (may sort) requests by sector
 - u kernel can optimize list for the device before submitting (part of) it
 - u **bio structure** (Block I/O) maps memory to each request
- n Kernel calls device driver with request list
 - u device completes all requests in list
 - u data transfer via kernel **buffer cache**
- n HW RAID devices are given requests directly
- n SW RAID implemented in kernel (included in std kernel)
- n Linux Direct I/O does not use Page Cache (disk buffer)
 - u direct copying from device to user space
 - F no need to copy through kernel buffer cache
 - u driver still suspends while waiting

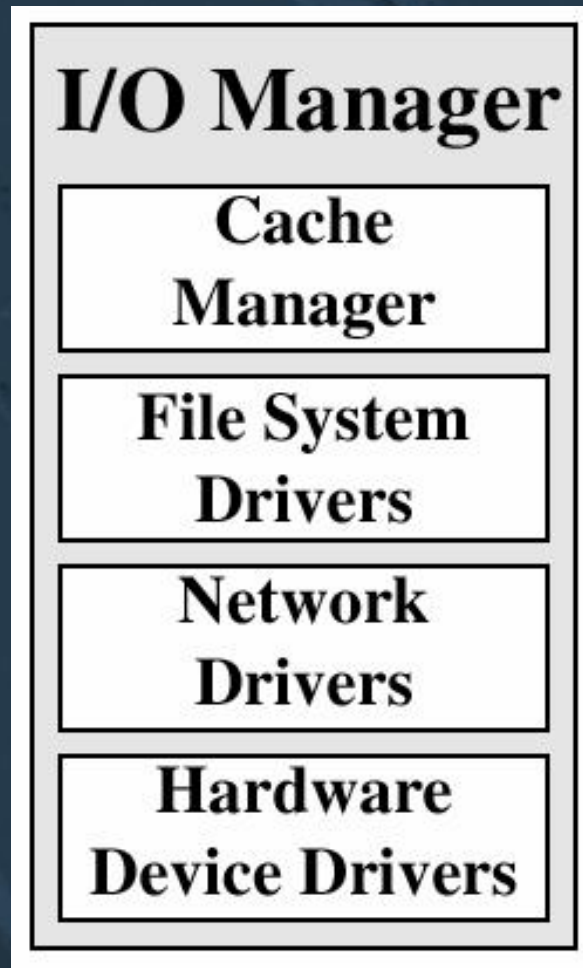
Operating Systems II

Windows 2000 I/O

Ch 11.10 [Stall 05]

Ch 11.6 [Tane 01]

W2K I/O-manager



(Fig 11.15 [Stal05])

- n **Device independent API for all devices**
 - u many different API's for all kinds of devices (device types)
- n **Dynamically loadable**
- n **Cache**
 - u common for all file systems and networks
 - u size varies dynamically
 - u *lazy write* and *commit*
- n **Device drivers**
 - u access device registers via generic HAL interface
 - F DLL for each platform

W2K Device Drivers

n Windows Driver Model

- u plug-and-play
- u re-entrant code, SMP supported
- u for each device, **device object** created in directory \??
- u W2000 and W98 support

n New device?

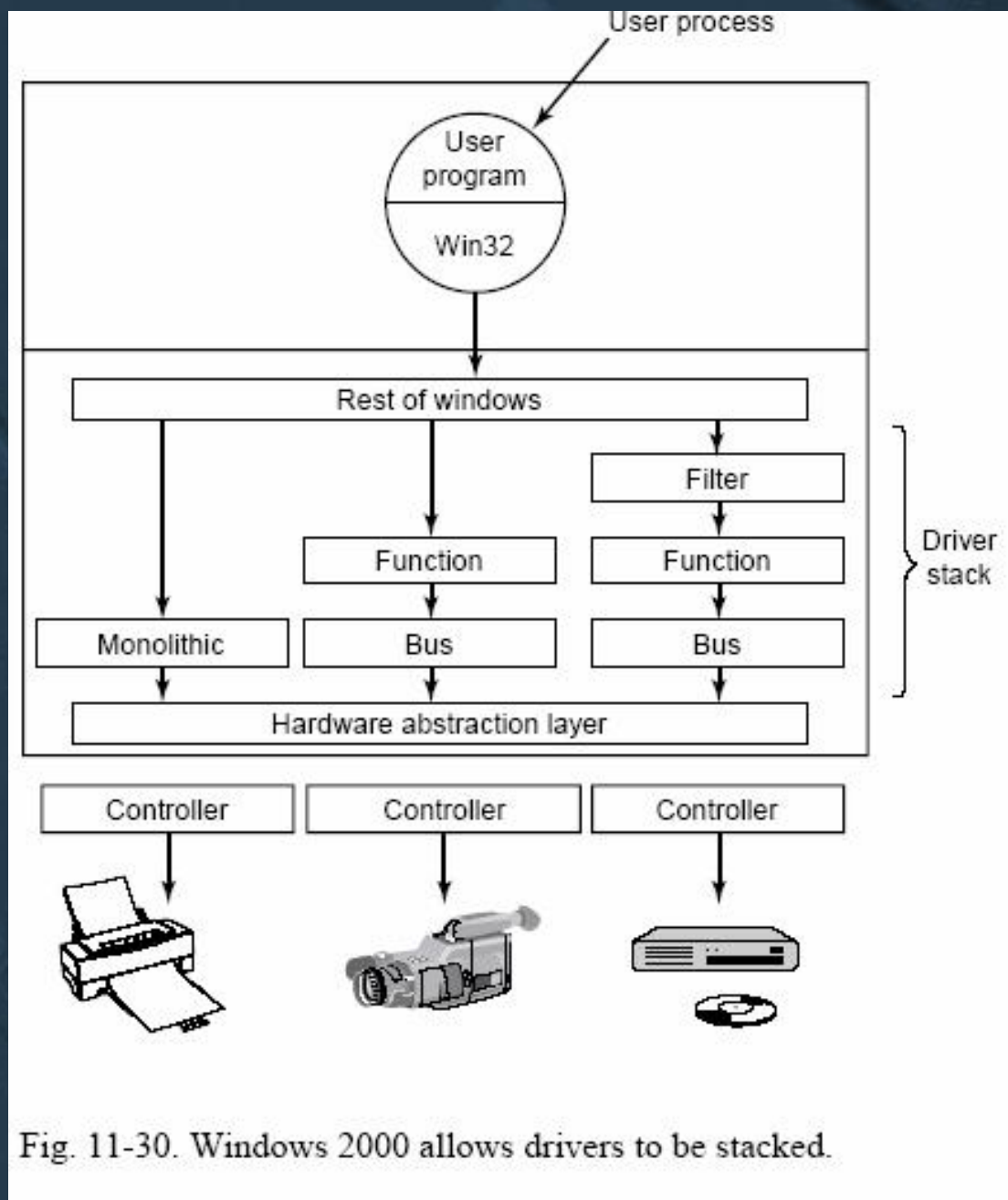
- u plug-and-play manager queries it for manufacturer and model
- u if recognized, load driver to memory from disk
- u if not recognized, ask for CD (or floppy), and then load it

n IRP (I/O Request Packet) for all I/O requests

n Drivers may be stacked

Fig 11-30 [Tane01]

n Filter driver can do transformations for other drivers



([Tane01])

Fig. 11-30. Windows 2000 allows drivers to be stacked.

W2K I/O

- n **File system**
 - u technically just a device with its own device driver
- n **SW RAID1**
 - u disk mirroring
 - F shared device controller
 - u disk duplexing
 - F dedicated device controllers
- n **SW RAID 5**
- n **Synchronous I/O**
 - u wait blocked until I/O completed
- n **Asynchronous I/O**
 - u send request and continue
 - u later on, check that I/O completed and possibly wait
 - u what can I do while waiting for I/O?

W2K Asynchronous I/O

- n **Send request and proceed**
- n **Later on, check that I/O completed and possibly wait**
 - u signal via device kernel object
 - F just one per device, so can wait for just one I/O request
 - u signal via newly created event kernel object
 - F any combination possible, because of many events
 - u signal via thread APC queue (Asynchronous Procedure Call)
 - F result of I/O-op to APC queue
 - F APC executed later on (and signals thread requesting I/O?)
 - u signal via specific I/O completion ports
 - F fast
 - F ready pool of ports

W2K I/O Interrupts

(W XP, Ch 21.5 [DDC04])

- n **Fast response time by splitting interrupt handling to two parts**
 - n **Time critical in hardware interrupt service procedure**
 - u (same or lower level) interrupts disabled
 - u no context
 - u acknowledge interrupt, save interrupt state
 - u invoke DPC or APC by triggering lowest level interrupts for them
 - n **Rest in software interrupts DPC or APC**
 - u DPC (Deferred Procedure Call)
 - u APC (Asynchronous Procedure Call)
- (explained in next slides)

Compare to Linux Top Half and Bottom Half!

W2K DPC

(W XP, Ch 21.5 [DDC04])

- n **DPC (Deferred Procedure Call)**
- n **Software interrupts**
- n **No own context, use interrupted thread context**
 - u for cases where context is not important
- n **Must not block in DPC**
- n **DPC queue at each processor**
- n **Most of interrupt processing here**

W2K APC

(W XP, Ch 21.5 [DDC04])

- n **APC (Asynchronous Procedure Call)**
- n **Belongs to some thread, have context**
 - u invoke a procedure to be executed by someone else!
 - u give thread something to do when he wakes up next time
 - u usually I/O interrupt handler's not-so-time-critical part
- n **Special APC's have priority in execution order**
 - u before normal APC's
- n **Kernel mode APC (joka säikeellä jono)**
 - u software interrupts, generated by kernel mode components
 - u executed only when owning thread is scheduled
 - F owning thread wakes needs to be in ***alertable wait state***
 - u all APC's done before thread can continue
- n **User mode APC (joka säikeellä jono)**
 - u threads can select when to execute APC's (if ever)
 - u If thread never enters ***alertable wait state***, then APC is never executed

W2K Cache Manager

(Ch 11.9 [Tane01])

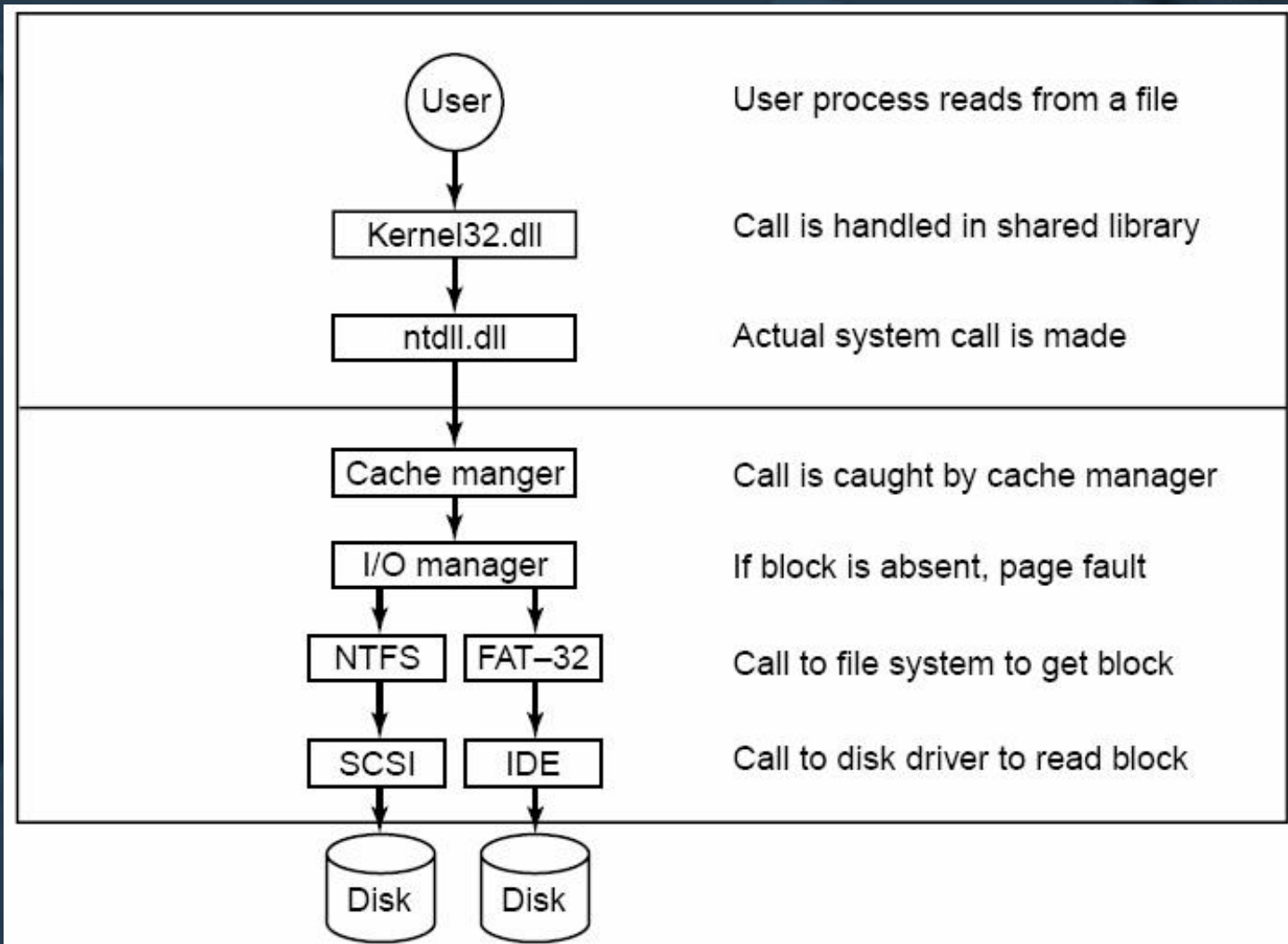
(Tiedostovälimuisti)

n One cache shared for all file systems

- u NTFS, FAT-32, FAT-16, CD-ROM, ...
- u sits on top of the file systems
- u based on logical files (file, offset)
 - F not on physical files (partition, block)

n All files are mapped to memory (in kernel address space)

- u access through virtual memory manager Fig 11-45 [Tane01]
 - F read op: copy from kernel address space to user addr sp.
- u access to disk buffer looks just like any memory access
- u page fault to cache manager handled just like any other page fault
 - F cache manager does not know about it
 - F cache manager does not even see physical memory
 - physical memory handled by VM in 256 KB chunks



User process reads from a file

Call is handled in shared library

Actual system call is made

Call is caught by cache manager

If block is absent, page fault

Call to file system to get block

Call to disk driver to read block

[Tane01] Fig. 11-45. The path through the cache to the hardware.

Kertauskysymyksiä

- n Siirännän hierarkia
- n Mitkä tekijät vaikuttavat levyhaun kestoon?
- n Miten FSCAN poikkeaa SCAN-algoritmista?
- n Miksi FSCAN?
- n RAID-levyjen perusideat
- n Miksi tarvitaan levylohkojen puskurointia?