Luentokerta 5

Ohjelman toteutus järjestelmässä (prosessi, käyttöjärjestelmä) Järjestelmän ulkoinen muisti

Prosessi, sen esitysmuoto

Käyttöjärjestelmä

KJ-prosessit

Muistihierarkia

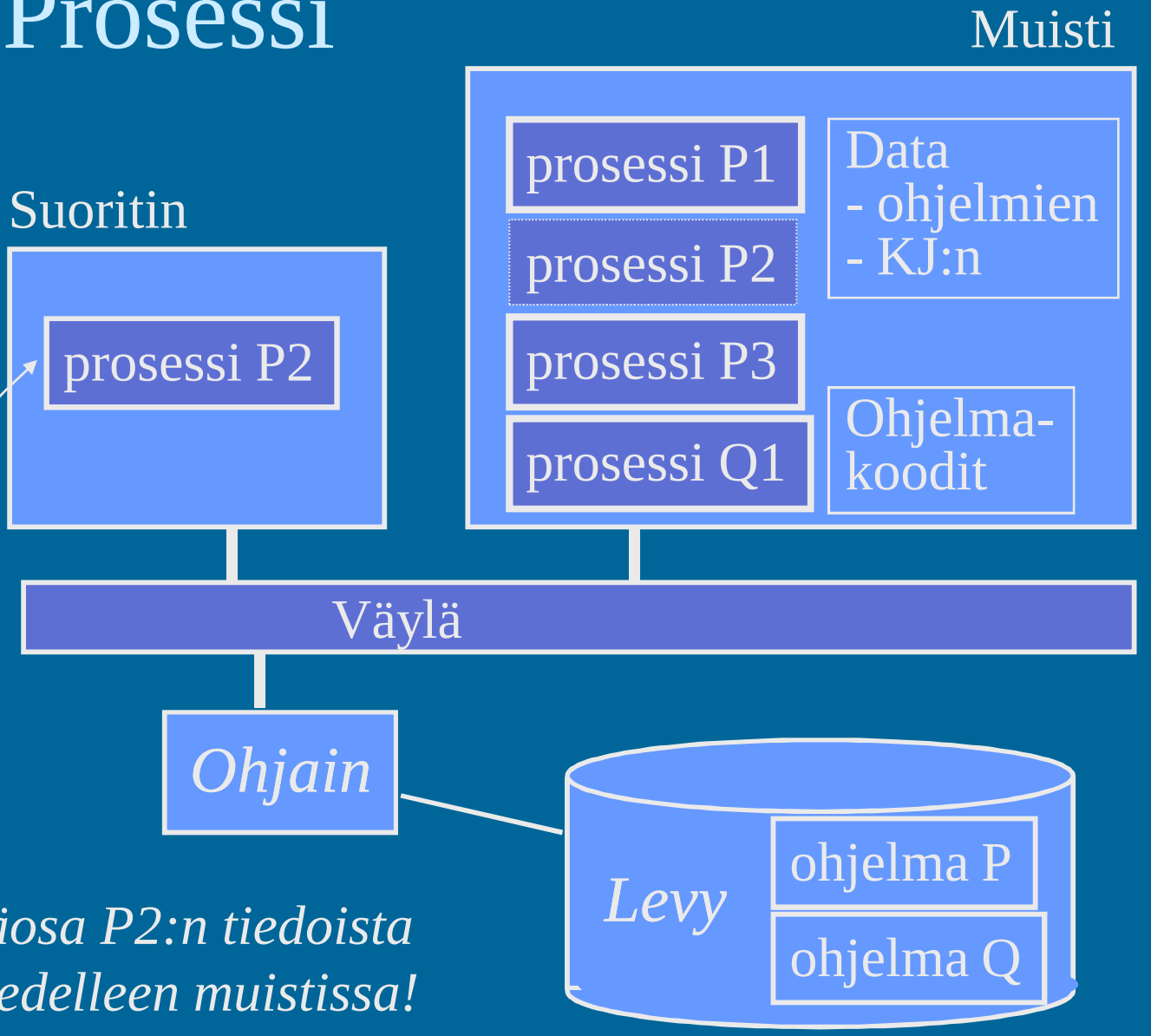
Kiintolevyt

I/O:n toteutus

Prosessi = ohjelma järjestelmässä

- Järjestelmässä voi olla ”samalla kertaa” **monta prosessia** joko samasta tai eri ohjelmasta
  - käyttäjän (ihmisen) näkökulma ja aikaskaala (1 min, 1 sek?)
- Suorittimella **suorituksessa on vain yksi** prosessi kerrallaan
  - laitteiston näkökulma ja aikaskaala (1 ms, 1 µs, 1 ns?)
- **Muut prosessit ovat odottamassa** jotakin
  - suoritinta? I/O:ta? viestiä toiselta prosessilta?
  - vapaata muistitilaa?

Prosessi

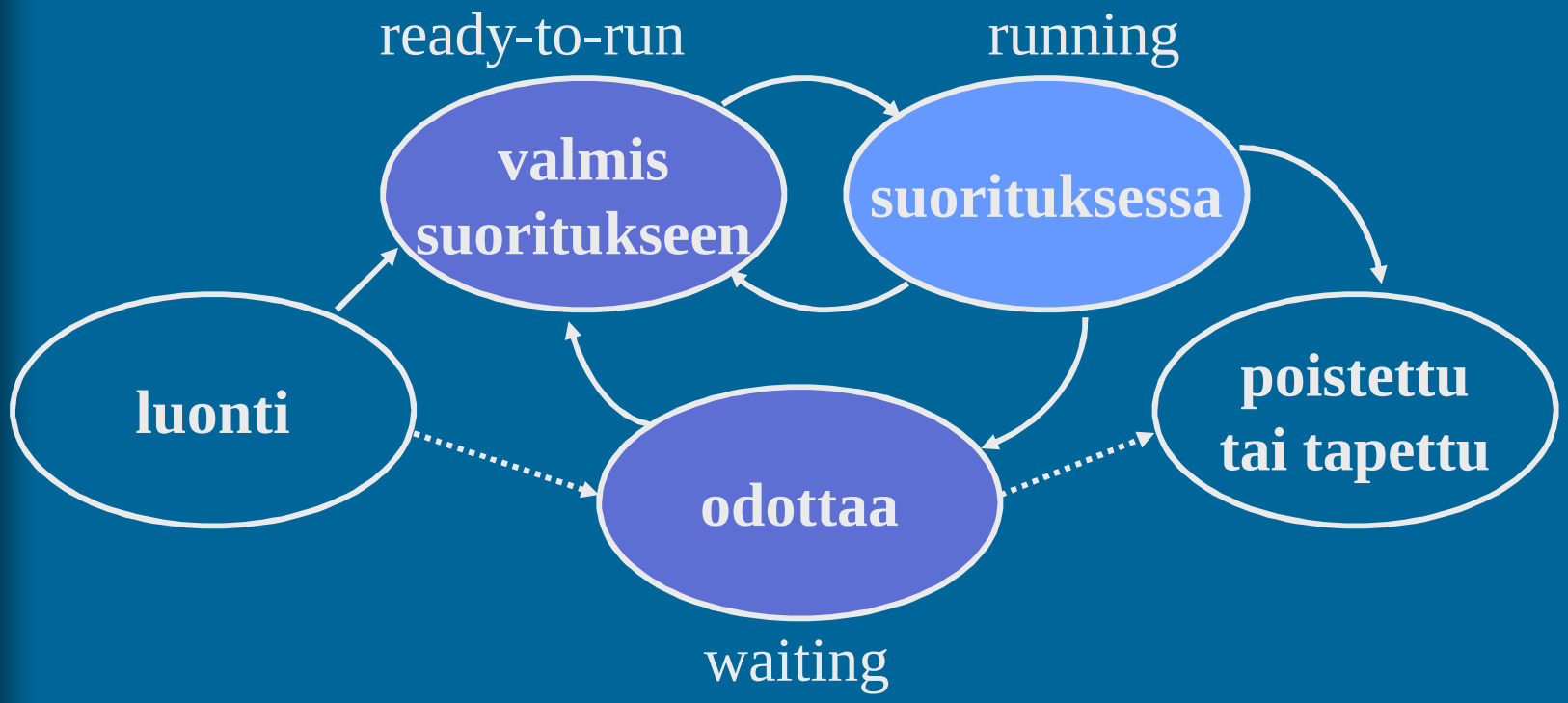


Prosessin vaihto

- Suorittimella suoritusvuorossa olevan prosessin vaihtaminen
- Tapahtuu aika usein
  - keskimäärin noin 2000-3000 konekäskyn välein?
  - esim. 50-500 kertaa sekunnissa?
- Iso operaatio - paljon kopiointia
  - montako konekäskyä tähän kuluu?

50-500?
0?

Prosessin elinkaari



- Prosessin 5 suoritustilaa
  - Milloin tilanvaihto tapahtuu?
  - Mitä tilanvaihdossa tapahtuu?

Mitä suoritin tietää suorituksessa olevasta prosessista?

Prosessin kuvaaja

- PCB - Prosessin kuvaaja eli kontrollilohko (Process Control Block, myös process table entry)
  - isohko tietue, joka sisältää kaiken yhdestä prosessista
    - muistialueet, tiedostot, tiedostojen käsittelykohdat
    - odottavilla prosesseilla myös: suorittimen tila (laiterekisterit, MMU:n rekisterit, kontrollirekisterit)
  - joka prosessista oma kuvaaja
  - luodaan prosessin luonnin yhteydessä ja tuhotaan prosessin päättyessä
  - käyttöjärjestelmärutiinit manipuloivat PCB:tä

Prosessin kuvaajan sisältö

- Prosessin tunniste 14023
- Prioriteetti suorittimen vuoronantoa varten 143
- Prosessin tila ja/tai odottamisen syy R-to-R
- Suoritinympäristö talletettuna odottamisen aikana
  - rekisterit, PC, SP, FP, tilarekisterit
- Seuraavaksi suoritettavan käskyn osoite Main {}
- Poikkeuskäsittelijöiden osoitteet (ellei oletusarv.)
- Aikaviipale
- Käytössä olevat muistialueet, aukiolevat tiedostot
- KJ:n hallintotietoa (kokonaisaika, etc etc)

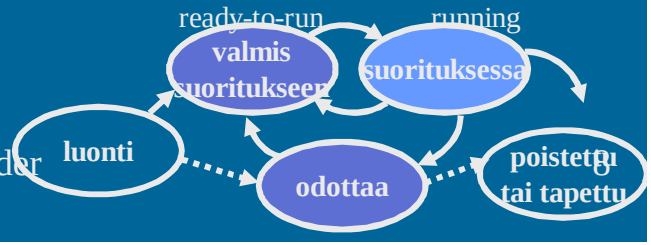
Prosessin tilanvaihdon toteutus

- Prosessin tilanvaihto tapahtuu siirtämällä prosessi (sen PCB) jonosta toiseen
  - ready-to-run jono (tai jonot)
  - running jono
    - ei oikeastaan ole olemassa
  - waiting jono
    - joka tyypille oma jononsa
    - esim: laitteen Disk1 I/O:n valmistumista odottavat
    - esim: näppäimistön painallusta odottavat
    - esim: kellolaitekeskeytystä odottavat
    - esim: prosessilta 1345 signaalia odottavat

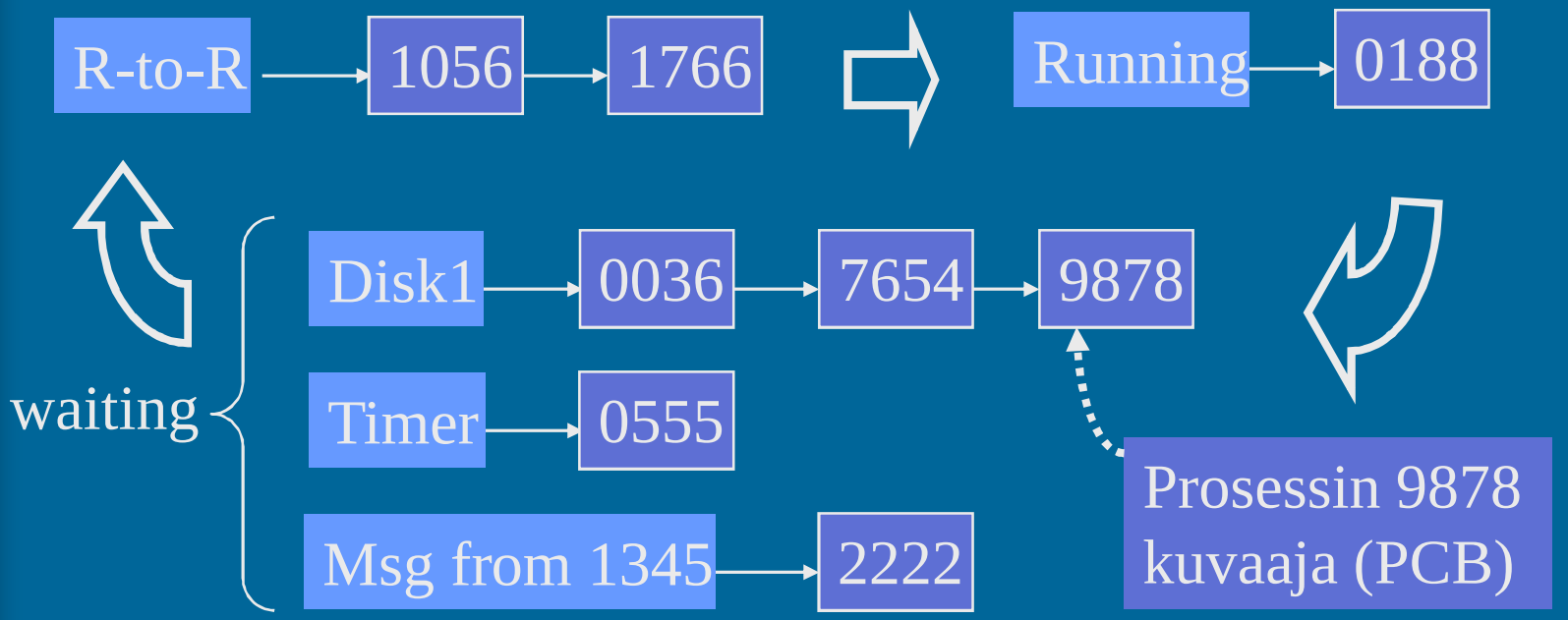
odottaa suoritinta

suorituksessa

odottaa jotakin



Prosessit jonoissa



Vuoronanto:
valitse seuraava prosessi Ready-to-Run -jonosta ja
siirrä se suoritukseen CPU:lle
(kopioi tämän prosessin **suorittimen tila** suorittimelle)

Prosessin vaihto

- Vaihdon tekee KJ rutiini sillä hetkellä suorittavan prosessin ympäristössä
- Talleta vanhan prosessin suoritinympäristö (eli suorittimen tila) suorittimelta omalle talletusalueelle muistiin
  - Rekisterit, myös PC (tästä jatketaan joskus …)
  - Ei talteen, jos vanha prosessi tapetaan
- Kopioi uuden prosessin suoritinympäristö omalta talletusalueeltaan suorittimelle
  - lataa kaikki suorittimen rekisterit (myös PC!)
- Uuden prosessin suoritus jatkuu täsmälleen siitä mihin viime kerralla jäätiin
  - sama konekäsky, käytännössä sama suoritusympäristö
  - usein keskellä prosessin vaihtoa suorittavaa KJ rutiinia

Prosessin prioriteetti

- Prosessin tärkeysjärjestys suorittimella
  - esim. pieni numero ⇒ iso (parempi) prioriteetti (tai toisinpäin)
- Joka prioriteetti(luokalle) oma R-to-R jononsa
  - KJ prosesseilla parempi prioriteetti kuin käyttäjätason prosesseilla
  - tosiaikasovelluksen prosesseilla parempi prioriteetti kuin KJ prosesseilla
    - muistakaa antaa KJ:lle aikaa aina joskus .... !
- Prioriteetti voi vaihdella prosessin elinaikana
  - paljon suoritinaikaa ⇒ huonompi prioriteetti
  - kauan R-to-R jonossa ⇒ parempi prioriteetti
    - prosessi siirretään korkeamman prioriteetin R-to-R jonoon

Porinaa prosesseista

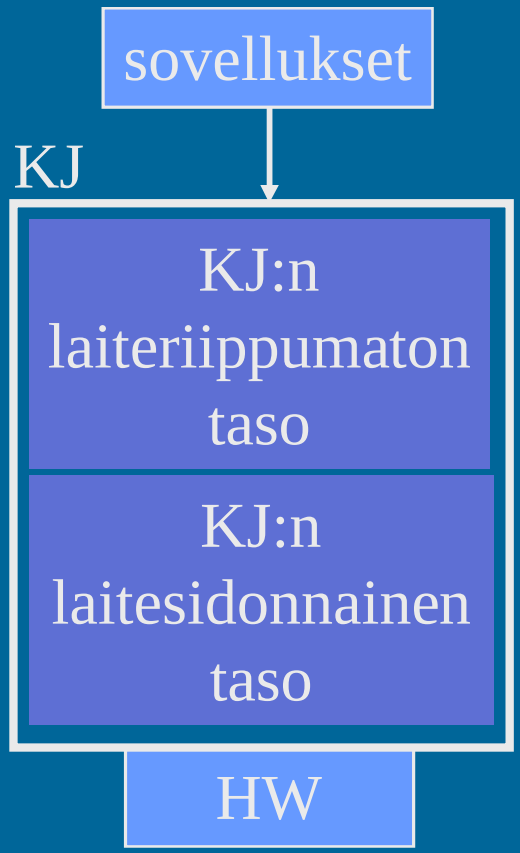
- Pikaäänestys: Kuinka usein prosessia vaihdetaan?
- Kauanko prosessin vaihto kestää?

- Jos levytapahtumaa oli odottamassa 3 prosessia, miten tiedetään mikä niistä pitää herättää?

- Mitä hyötyä saavutetaan, kun joka prioriteetti-luokalle on oma Ready-to-Run (R-toR) -jono sen sijaan, että olisi vain yksi jono?

Käyttöjärjestelmä

- Laiteriippumaton (HW-riippumaton) rajapinta laitteistoon
  - Helpottaa käyttöä
  - Jakaa palvelua kaikille reilusti
  - Resurssien hallinta ja valvonta
  - Sovellukset on helpompi toteuttaa ja siirtää muualta



Resurssien hallinnan tavoitteita

- Kaikista resursseista maksimihyöty
  - Miten optimoida resurssien käyttö?
  - Kuka osti liikaa levyjä?
- Joustava resurssien yhteiskäyttö
  - Lue tiedosto levyltä vai verkkopalvelimelta?
- Tiukka tietosuoja
  - Kuka muu kuin Pekka tai Maija luki tietojani?
  - Estä luvaton käyttö

Resurssien hallinta ja valvonta

- Suorittimen vuorottaminen
  - jaa suoritinaikaa reilusti kaikille
  - kukaan ei odota suoritinta ikuisesti
  - kriittiset prosessit saavat ajoissa suoritinaikaa
- Tiedostojen (koodi, data) tehokas käyttö
  - laitteesta ja sijainnista riippumaton käyttö
  - helppo yhteiskäyttö ja samalla tietojen suojaus
- Tietoliikenneverkkojen käyttö
  - laiteriippumaton käyttö
  - helppo yhteiskäyttö ja samalla tietojen suojaus
- Hallintokirjanpito

KJ eheyden turvaajana

- Varauduttu kaikkiin mahdollisiin virheisiin
- Sovellusohjelmat eivät voi häiritä KJ:tä tai muita prosesseja
  - tahallaan (esim. tietokonevirukset)
  - vahingossa (yleisin tapaus)
- Järjestelmä ei lukkiudu tai ”kaadu”
  - KJ:n omat tietorakenteet ovat aina eheitä
  - sovellusohjelmat eivät voi koskea KJ:n tietorakenteisiin
  - sovellusohjelmat aina lopulta antavat vuoron KJ:lle

Käyttöjärjestelmän rakenne

- Prosessien hallinta
  - prosessien luonti, tuhoaminen
  - prosessien välinen viestintä (IPC, Inter-Process Comm)
  - kenelle suoritinaikaa ja milloin?
- Muistin hallinta
  - miten keskusmuistia varataan eri prosessien käyttöön?
  - kunkin prosessin muistitilan hallinta
  - yhteiskäyttö ja tiedon suojaus
- Tiedostojen ja laitteiden hallinta
  - miten tiedostoja voidaan lukea/kirjoittaa?
  - yhteiskäyttö ja tiedon suojaus
- Verkon hallinta
  - miten kommunikoida muiden koneiden kanssa?

Käyttöjärjestelmän toteutus

- Joukko prosesseja ja/tai aliohjelmia
  - prosessit elävät omaa elämäänsä (root'ina?)
    - swapper (Unix) - muistinhallintaprosessi
    - init prosessi (Unix) - kaikkien käyttäjätason prosessien ”äiti”
    - laiteajurit
  - aliohjelmat suoritetaan sen hetkisen prosessin ympäristössä (etuoikeutetussa tilassa?)
    - keskeytyskäsittelijät
  - saavat kontrollin aina tarvittaessa
    - aliohjelmakutsut, SVC, viestit
    - ajastimet ja muut keskeytykset

KJ palvelun kontrollin palautus

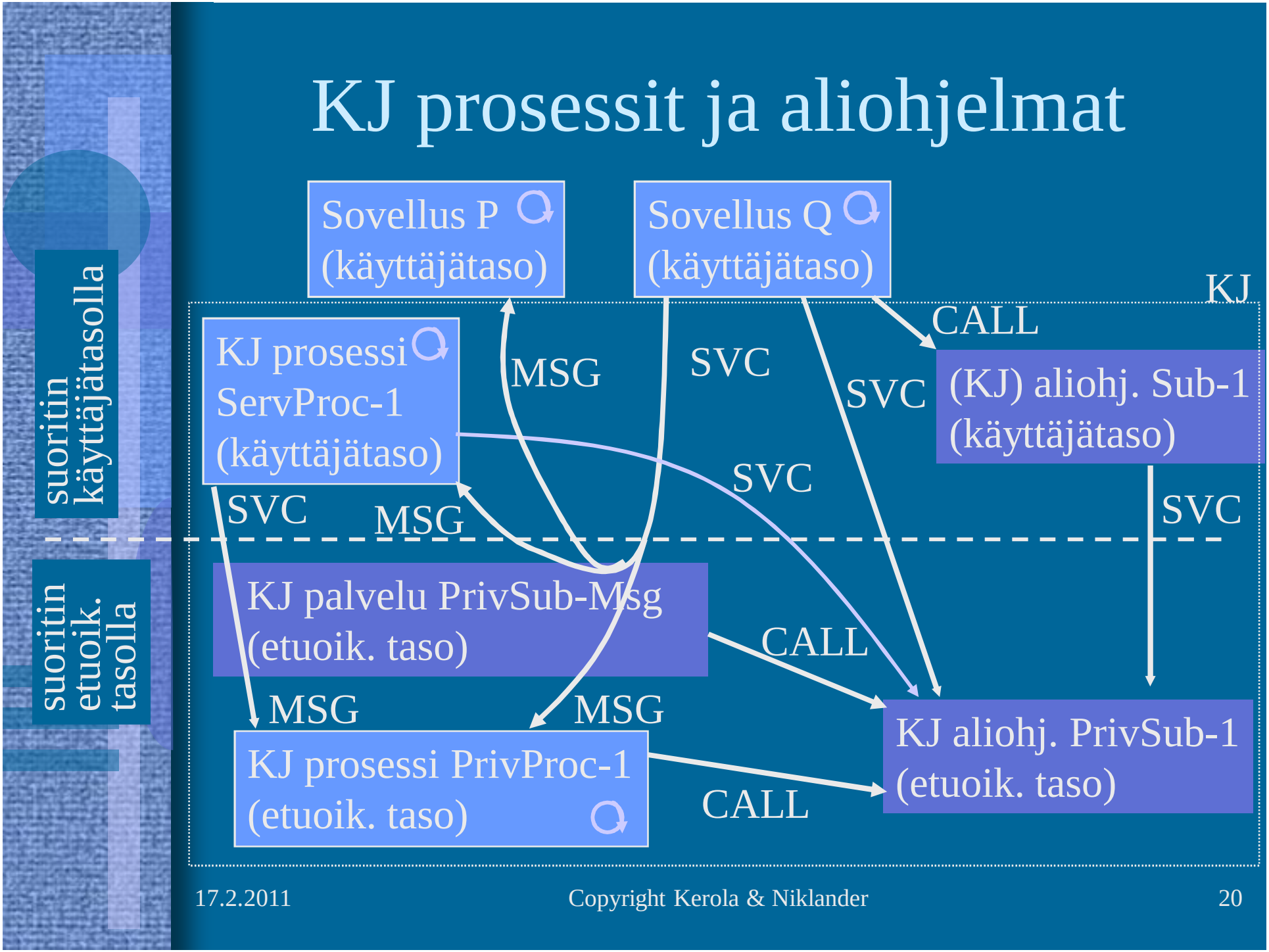
Explisiittinen KJ-palvelun kutsu = Palvelupyyntö

- Aliohjelmakutsut
  - CALL → RETURN
- SVC
  - SVC → IRET
- Viestit
  - viesti → vastausviesti
    (lähettäjä odottaa vastausta RECEIVE:ssä)

Implisiittinen KJ-palvelun kutsu

- Ajastimet ja muut keskeytykset
  - keskeytys → IRET

KJ prosessit ja aliohjelmat

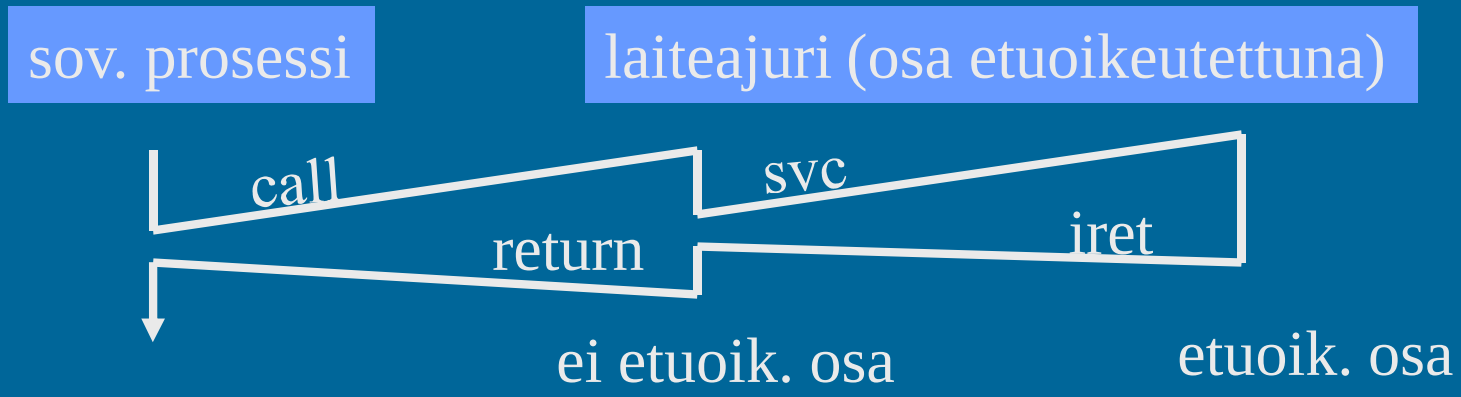


KJ esimerkki: laiteajuri

- Aliohjelmana (eli proseduurina)
  - laiteajuri suoritetaan KJ-rutiinina tavallisen SVC-kutsun kautta käyttäjätilassa
    - vain yksi kutsu kerrallaan suorituksessa? miksi? miten voidaan valvoa?

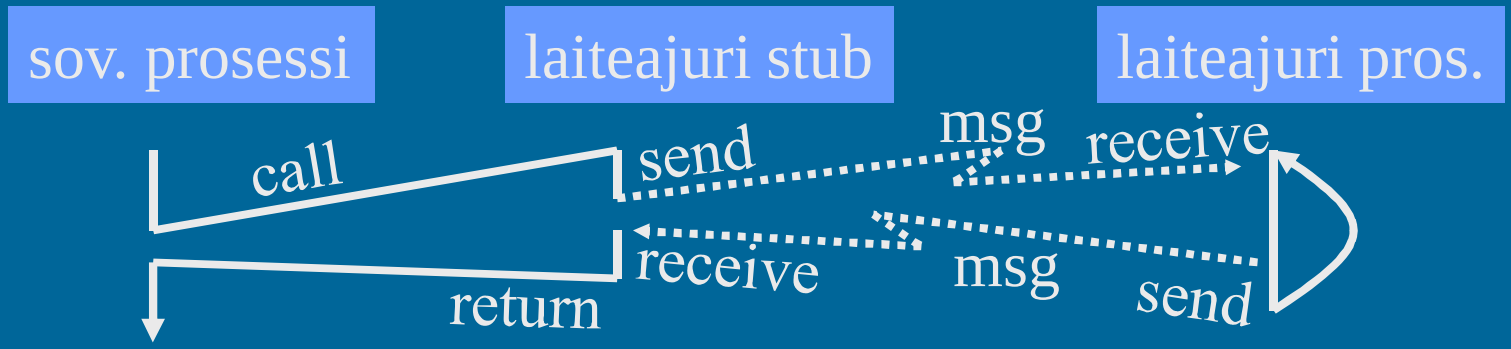
KJ esimerkki: laiteajuri (jatk.)

- Aliohjelmana (eli proseduurina)
  - laiteajuri suoritetaan KJ-rutiinina tavallisen aliohjelmakutsun ja/tai SVC-kutsun kautta
    - osa tai kaikki koodista voi olla etuoikeutettua
    - vain yksi kutsu kerrallaan suorituksessa? miksi? miten voidaan valvoa?



KJ esimerkki: laiteajuri

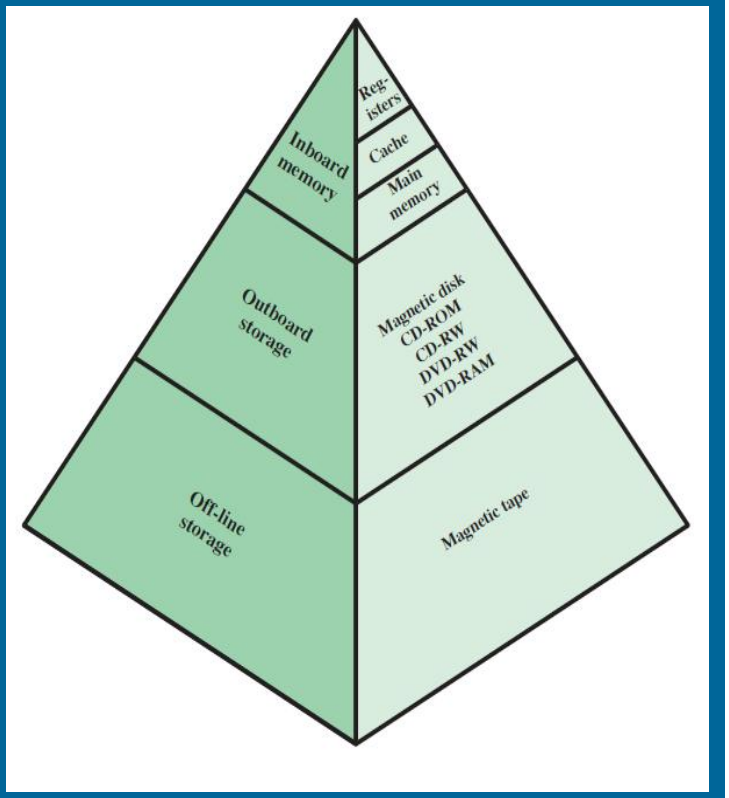
- Prosessina
  - proseduurina kutsuttu laiteajurin tynkä (stub) lähettää I/O-pyynnön viestinä laiteajuriprosessille ja odottaa vastausta
    - tynkä voi olla käyttäjätilainen
    - ajuriprosessi voi olla (joskus) etuoikeutettu
    - vaatii prosessien välistä viestintää



Muistihierarkia

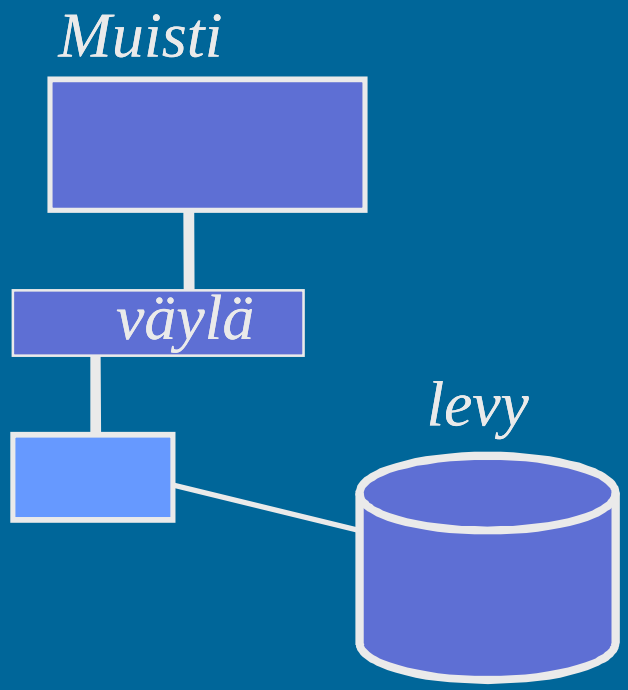
- Ulkoinen muisti on halvempaa toteuttaa per tavu
- Ulkoinen muisti on paljon hitaampaa kuin sisäinen muisti
- Aika/tila optimointi
  - suuret tietomäärät täytyy (kannattaa) kustannussyistä pitää ulkoisessa muistissa
  - pienet tietomäärät täytyy (kannattaa) tehokkuussyistä pitää sisäisessä muistissa
- Kaiken viitatun tiedot tulee suoritusaikana olla sisäisessä muistissa!

Fig 4.1 [Stal10]



Virtuaalimuisti

- Osa muistihierarkiaa
- Vastaus ongelmaan
  - miten tehdä suoritusaikaisesta muistista ”yhtä suuri” kuin levymuisti ja ”yhtä nopea” kuin keskusmuisti?
- Kaksitasoinen:
  - keskusmuistissa kulloinkin käytössä olevat alueet
  - levyllä kaikki tiedot
  - kopiointi tarvittaessa



Virtuaalimuistin toteutus

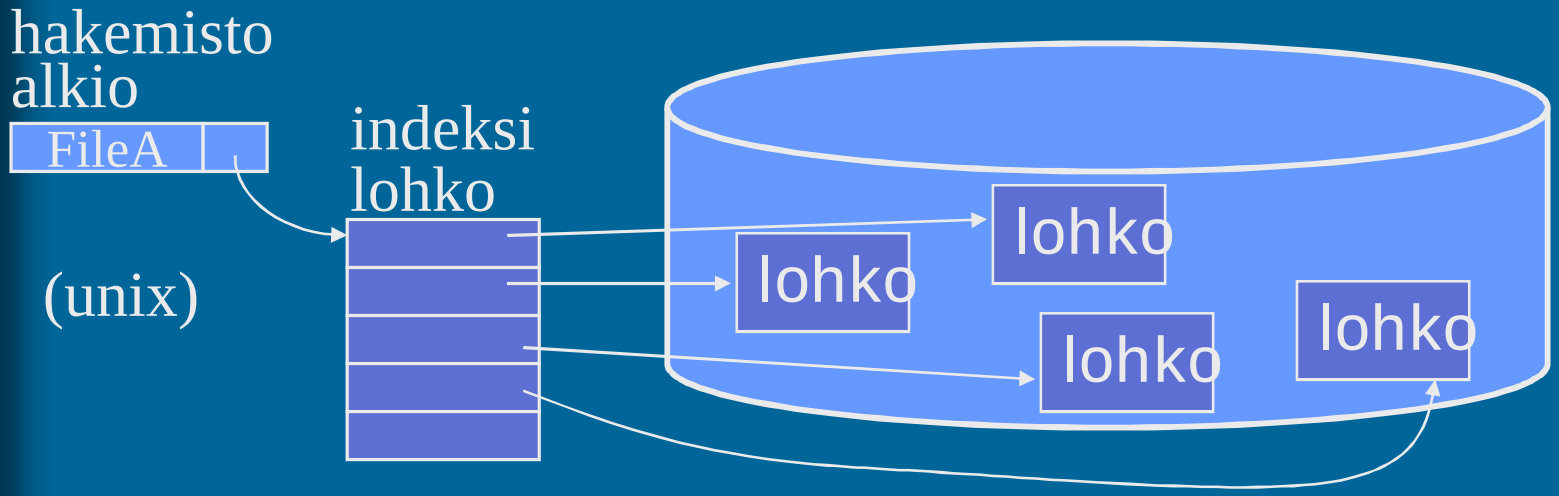
- Toteutustavat
  - kanta- ja rajarekisterit
  - sivutus
  - (segmentointi ja sivuttava segmentointi)
- Pääosa toteutuksesta ohjelmistotasolla
- Laitteistotuki
  - MMU – muistinhallintayksikkö
  - nopeuttaa viitatun muistipaikan todellisen osoitteen laskentaa
    - osoitetta ei tarvitse laskea usealla konekäskyllä, kun MMU tekee sen laitteistotasolla
  - rakenne ja toiminta vaihtelee virtuaalimuistin toteutustavan mukaan

Tiedostojärjestelmä

- KJ:n osa, hallitsee kaikkia tiedostoja
- Valvoo oikeuksia tiedostoa avattaessa
- Muuntaa tiedostonimet fyysisiksi osoitteiksi
- Ylläpitää taulukoita, joista näkee mitä kohtaa mistäkin tiedostosta kukin prosessi on käsittelemässä
- Tiedostojärjestelmä lukee ja kirjoittaa tiedostoja suurina kerralla käsiteltävinä lohkoina (0.5-8 KB?)
  - käyttäjätason prosessit käsittelevät tiedostoja tavuittain eikä niiden tarvitse tietää tiedoston todellista fyysistä rakennetta (KJ:n laiteajuri huolehtii siitä)

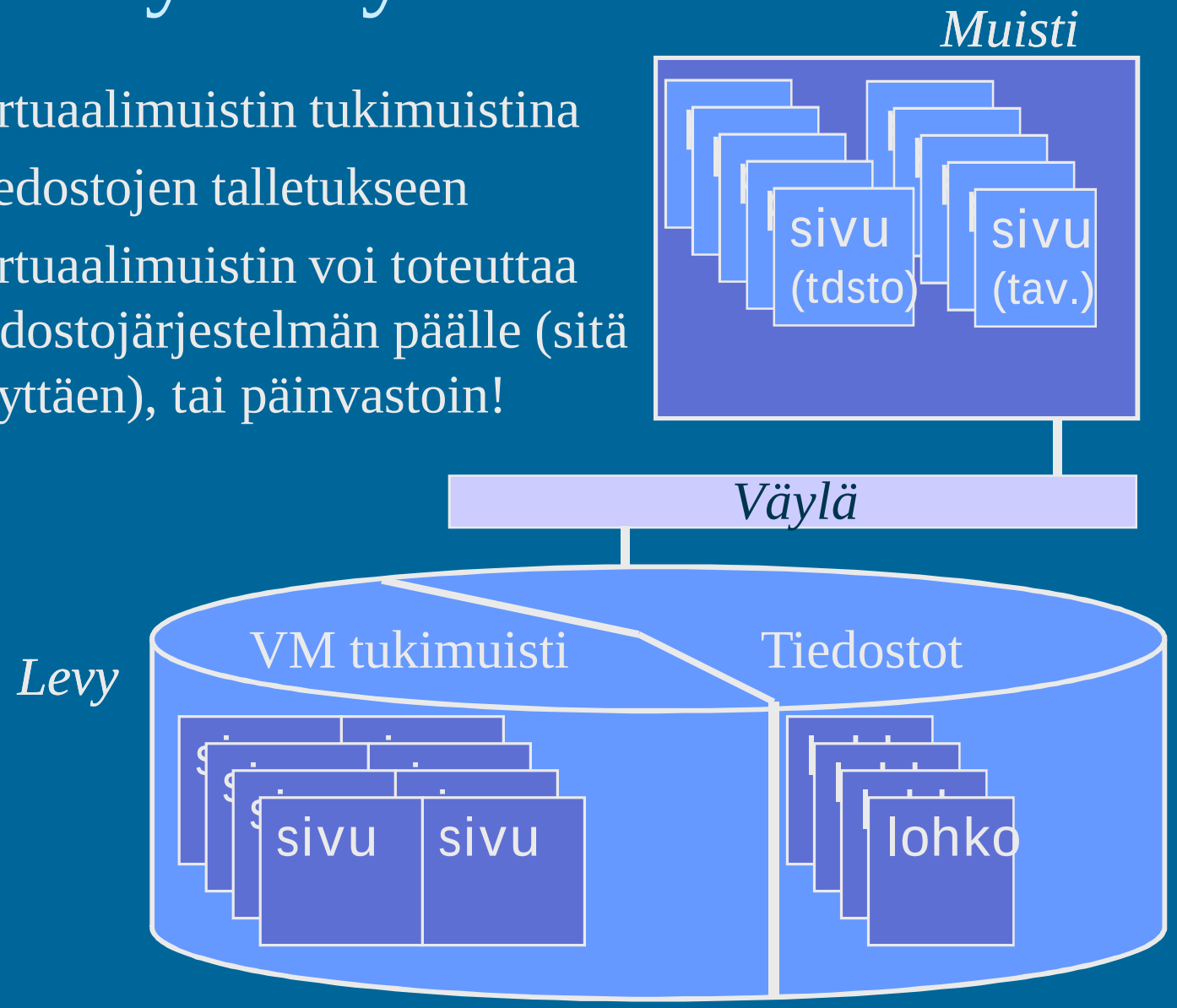
Tiedoston talletus levylle

- Tiedosto koostuu useista lohkoista
  - lohko per sektori (lohko per usea sektori?)
- Levyn hakemistossa on tieto kunkin tiedoston käyttämistä lohkoista
  - luetaan lohkot annetussa järjestyksessä



Levyn käyttö

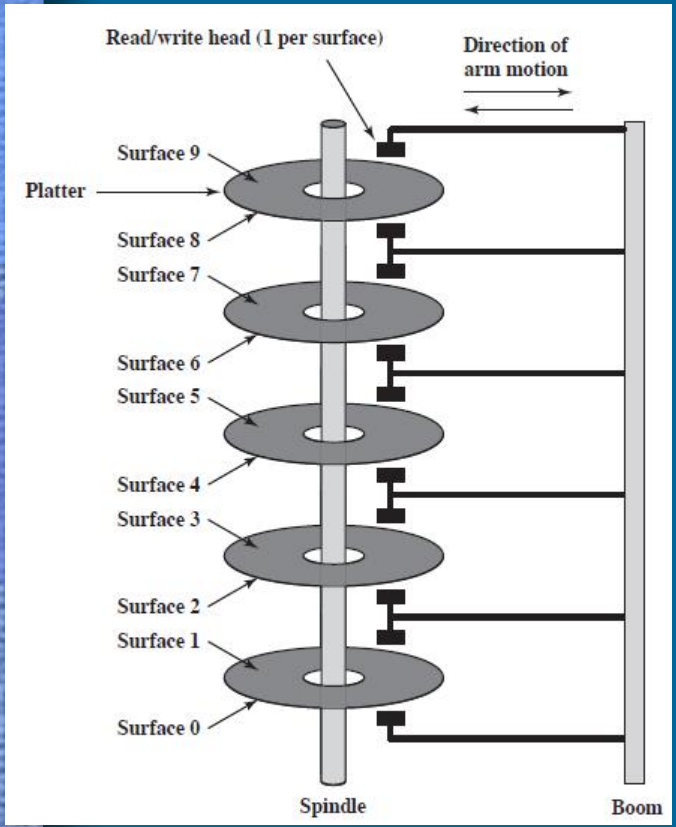
- Virtuaalimuistin tukimuistina
- Tiedostojen talletukseen
- Virtuaalimuistin voi toteuttaa tiedostojärjestelmän päälle (sitä käyttäen), tai päinvastoin!

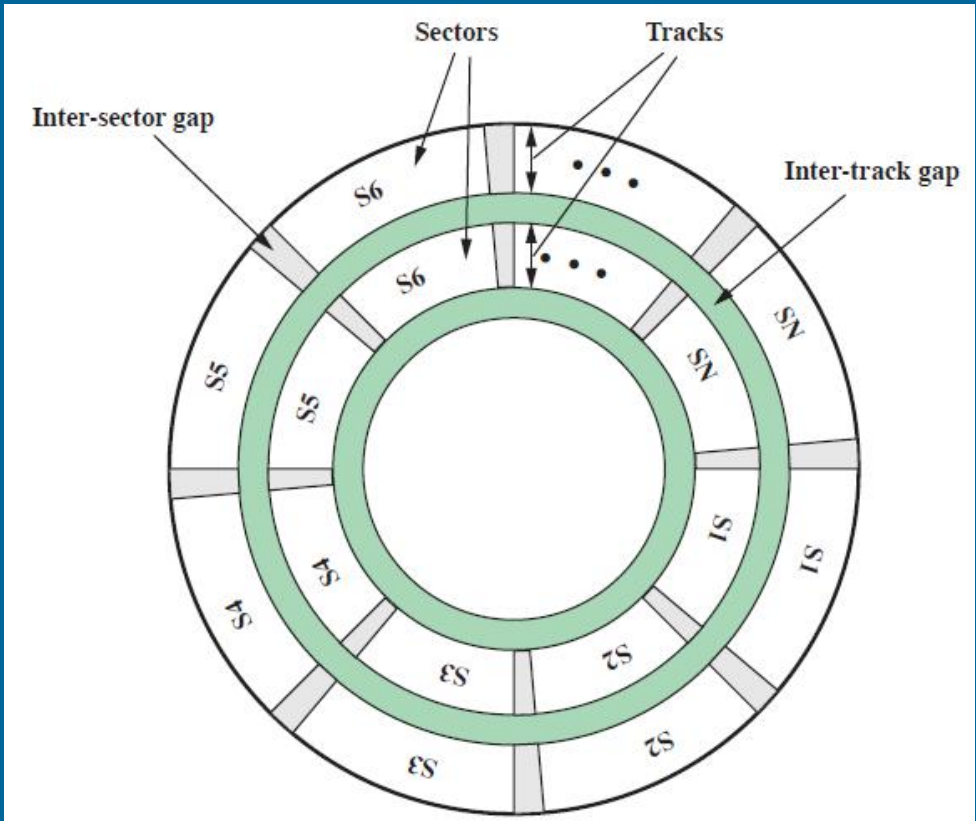


Levymuistin saantiaika

- Tiedon (lohkon) osoite: levypinta + ura + sektori
  - laiteajuri etsii KJ-taulukoista loogisen osoitteen perusteella

Figs 6.5&6.2 [Stal10]





Levymuistin saantiaika

- Saantiaika: (seek time)
  - hakuvarren siirtoaika
    Esim: keskiarvo 6.3 ms, min-max 2-15 ms?
  - odota kunnes sektori kohdalla
    Esim: pyörähdysviive kun 3600 rpm: 8.33 ms
    (keskim. puolen kierroksen aika) (rotational delay)
  - siirrä sektorin verran tietoa
    Esim: pyör.aika / sekt. lkm = 0.42 ms
    (data transfer time)

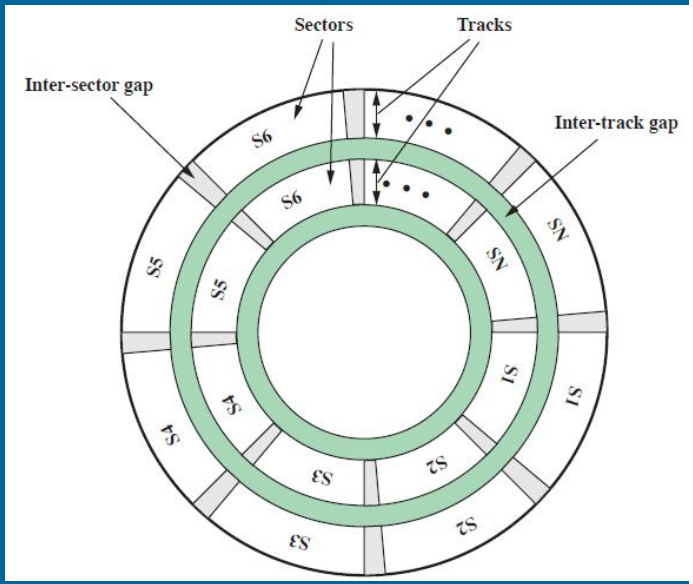
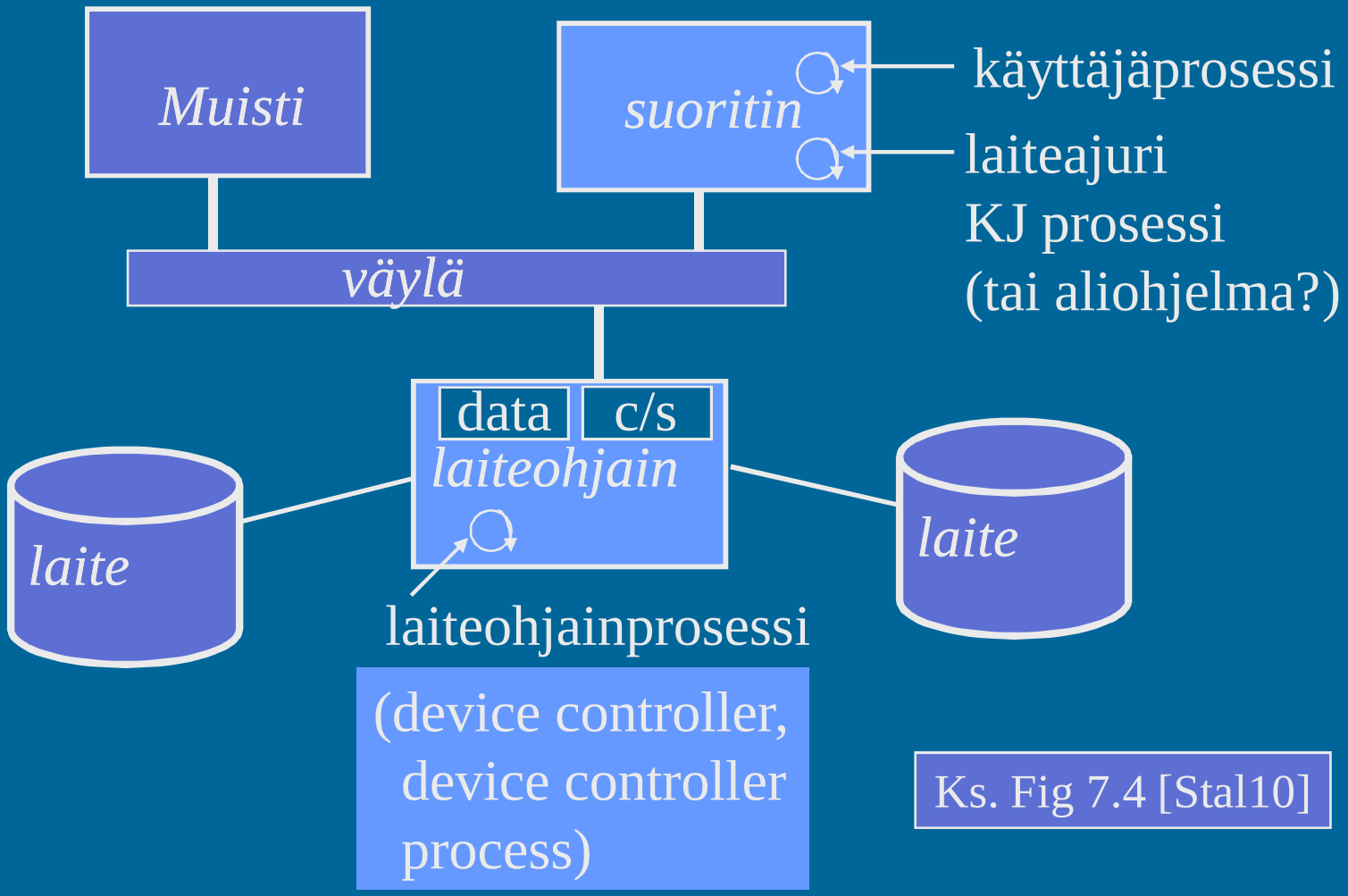


Fig 6.2 [Stal10]

Laiteohjain (I/O Moduuli)



Ks. Fig 7.4 [Stal10]

Laitteiden käytön toteutus

ks. laiteohjainkuva (ed. kalvo)

- Käyttäjäohjelma kutsuu käyttöjärjestelmän laiteajuria tekemään I/O:n. Laiteajuri suoritetaan samalla suorittimella kuin käyttäjäohjelmakin.
- Laiteajuri ohjaa laitteen toimintaa laitteen laiteohjaimella olevien kontrollirekisterien (muistialue ”c”) avulla
- Laiteajuri voi lukea laitteen tilatietoa laiteohjaimella olevien statusrekisterien (muistialue ”s”) avulla
- Laiteajuri voi lukea (kirjoittaa) laitteen lukemaa (laitteelle kirjoitettavaa) tietoa laiteohjaimella olevien datarekistereiden (muistialue ”data”) avulla
- Kontrolli-, status- ja datarekisteri kolmikko muodostaa ”I/O portin” suorittimen näkökulmasta

Laiteohjaimen rekistereihin viittaaminen

- Ongelma: miten suorittimella suorittavan laiteajuri viittaa eri kortilla oleviin ”rekistereihin” (muistiin)?

ks. laiteohjainkuva

- Ratkaisu 1: omat I/O-konekäskyt tätä tarkoitusta varten
  - käskyssä annetaan laiteohjaimen identifikaatio ja laiterekisterin nro (oma I/O osoiteavaruus)
  - vaikea laajentaa käyttöä uusiin laitteisiin, joilla ”laiterekisterit” voivat olla hyvinkin erilaisia
  - suorittimen konekäskyjä ei voi muuttaa

x86: IN, OUT
INS, OUTS

KOKSI:
IN, OUT

Ratkaisu 2: muistiinkuvattu I/O

ks. laiteohjainkuva

- Laiteajuri lukee/kirjoittaa laiteohjaimella olevia rekistereitä (data, status/kontrolli) tavallisilla muistin luku/kirjoitus käskyillä
  - ei tarvita erillisiä I/O-konekäskyjä!
  - laiteohjaimella olevat ”laiterekisterit” ovat samanlaista viitattavaa muistia kuin ”normaali muisti”
  - muistisoitteen ensimmäiset bitit (ei siis käskykoodi) valitsevat, mille laitteelle (vai tavallisen muistiin) viittaus kohdistuu
  - voidaan käyttää rinnan I/O käskyjen kanssa (laiterekistereihin voi siis viitata sekä I/O-käskyillä että muistiinkuvatun I/O:n avulla)

```
load  R1,=DiskRd
store R2, DiskCtr
```

```
DiskCtr EQU  0x80000001
```

esim. Intelin arkkitehtuurit

Voisiko tiedostoja lukea muistiosoitteiden kautta?

I/O tyypit

- **Suora I/O**: laiteajuri odottaa tiukassa silmukassa, kunnes laiteohjaimen statusrekisteri ilmoittaa I/O-pyynnön valmistuneen
  - laiteajuri siirtää tietoa muistin ja datarekisterin välillä

(direct I/O, programmed I/O)

- **Epäsuora I/O**: I/O:n odotusaikana suorittimella suoritetaan jotain muuta ohjelmaa

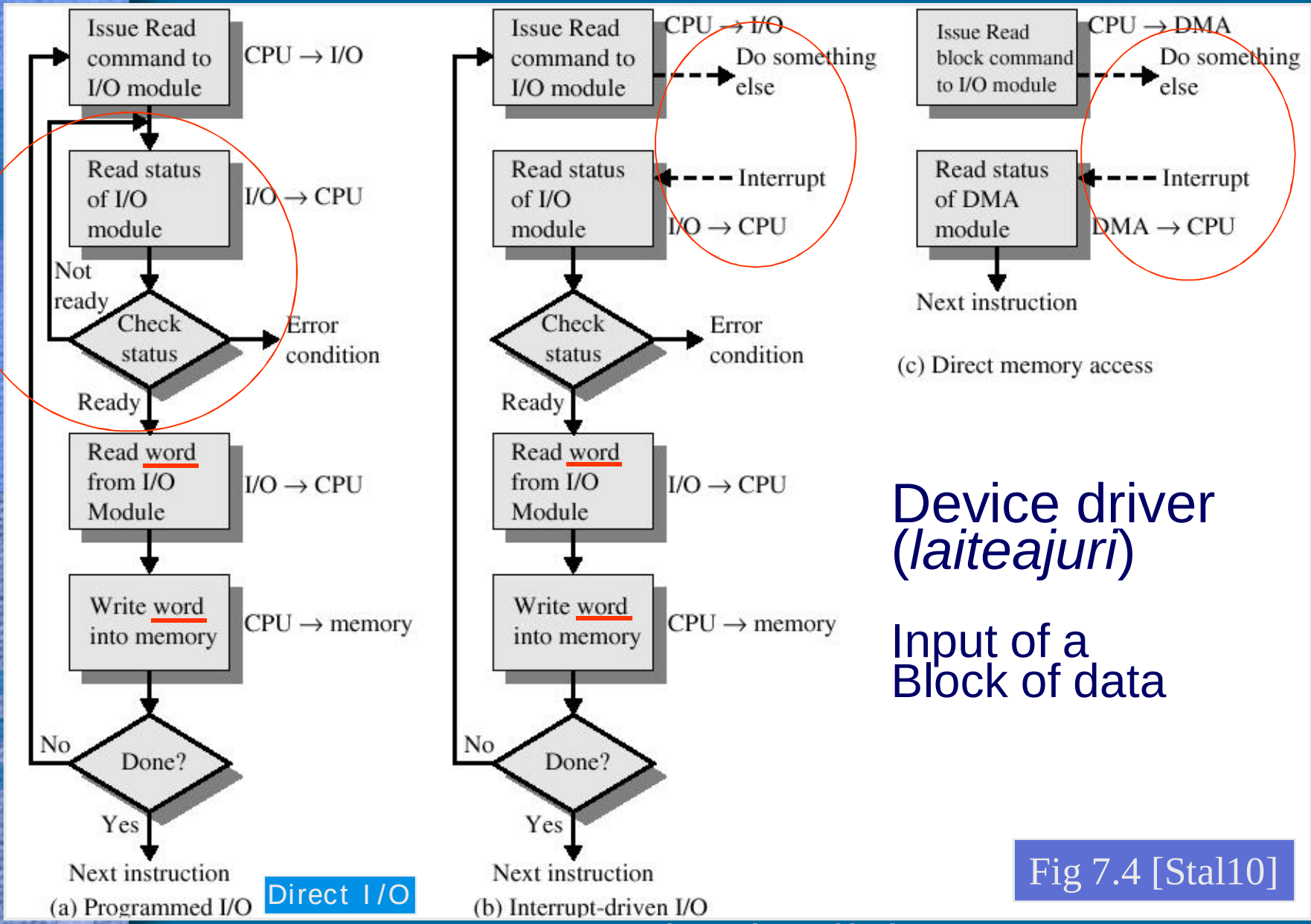
(indirect I/O interrupt driven I/O)

  - Kun I/O-pyyntö valmistuu, laiteohjain antaa keskeytyksen (laitekeskeytys, I/O interrupt) suorittimelle, joka (jonkin ajan kuluttua) jatkaa kesken jäänyttä I/O-pyynnön esittänyttä ohjelmaa.
  - laiteajuri siirtää tietoa muistin ja datarekisterin välillä

Suora I/O

Epäsuora I/O

DMA I/O



Direct I/O

Device driver (*laiteajuri*)

Input of a Block of data

Fig 7.4 [Stal10]

I/O tyypit (jatkoa)

- **DMA -** Direct Memory Access
  - Älykkäämpi laiteohjain
  - Laiteohjain voi suoraan kopioida tiedot keskusmuistiin
    - laiteajurin ei tarvitse laiterekistereitä käyttäen siirtää tietoa muistin ja datarekisterin välillä
    - Tieto kulkee väylän kautta vain yhden kerran!
  - Laiteohjain tekee paljon suuremman määrän työtä itsenäisesti (kuin epäsuorassa I/O:ssa) ennen suorittimelle annettavaa laitekeskeytystä

Mitä haittaa DMA-laitteesta voi olla suorittimella suorittavalle prosessille?

Esimerkki: kirjoittimen laiteajuri ttk-91 koneelle

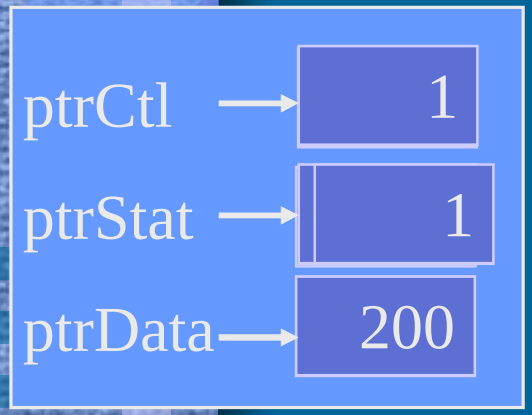
- Laitteella voi tulostaa kokonaislukuja yksi kerrallaan
- Muistiinkuvattu I/O, suora I/O
- Laiteportti
  - kontrollirekisteri muistipaikka 1048576 = 0x80000
  - tilarekisteri muistipaikka 1048577 = 0x80001
  - datarekisteri muistipaikka 1048578 = 0x80002
- Laiteajuri Print toimii etuoikeutetussa tilassa
- Kutsu:

```
PUSH  SP, =0       ; space for return value
PUSH  SP, X        ; parameter to print
SVC   SP, =Print   ; returns Success/Failure
POP   SP, R1
JNZER R1, TakeCareOfTrouble
```

Esim: laiteajurin toteutus

```
ptrCtr    DC  1048576 ; control register address
ptrStat   DC  1048577 ; status register address
ptrData   DC  1048578 ;
retVal    EQU  -3
parData   EQU  -2
```

Oleta: SVC:n ja IRET:n toteutus samalla tavalla kuin CALL ja EXIT



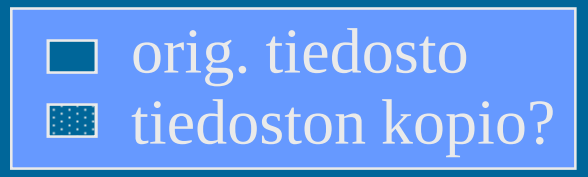
See: driver.k91

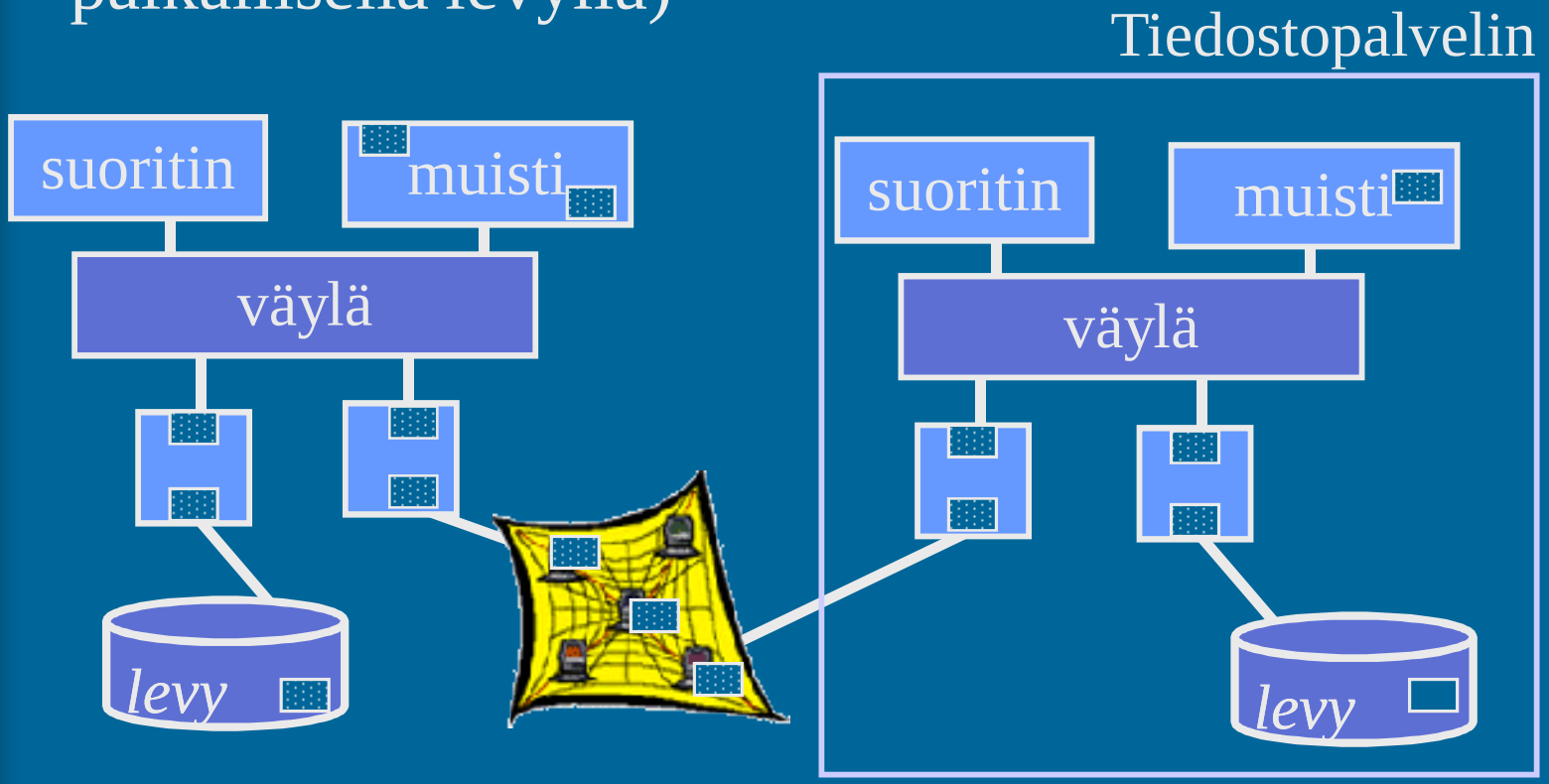
Solution with no timeout

```
Print   PUSHR  SP              ;save regs
        LOAD   R1, parData(FP)
        STORE  R1, @ptrData ; data to print
        LOAD   R1, =0
        STORE  R1, @ptrStat  ; init (clear) state register
        LOAD   R1, =1
        STORE  R1, @ptrCtr  ; give command to print

Wait    LOAD   R1, @PtrStat ; check state register
        JNZER  R1, Done
        JUMP   Wait             ; wait until I/O done

Done    LOAD   R1, =0           ; return "Success"
        STORE  R1, retVal(FP)
        POPR   SP               ; recover regs
        IRET   SP, =1
```

Tiedostopalvelin

- (Lähi)verkossa oleva palvelin
- Käytettäessä tiedoston (osien) kopio on muistissa (ja ehkä myös paikallisella levyllä)





-- Luennon 5 loppu --

[Tane99]

```
// Open files for input and output.
inhandle = CreateFile("data", GENERIC_READ, 0, NULL, OPEN_EXISTING, 0, NULL);
outhandle = CreateFile("newf", GENERIC_WRITE, 0, NULL, CREATE_ALWAYS,
        FILE_ATTRIBUTE_NORMAL, NULL);

// Copy the file.
do {
        s = ReadFile(inhandle, buffer, BUF_SIZE, &count, NULL);
        if (s > 0 && count > 0) WriteFile(outhandle, buffer, count, &ocnt, NULL);
while (s > 0 && count > 0);

// Close the files.
CloseHandle(inhandle);
CloseHandle(outhandle);
```

Figure 6-40. A program fragment for copying a file using the Windows NT API functions. This fragment is in C because Java hides the low-level system calls and we are trying to expose them.

Lisää tietoa?  KJ kurssi